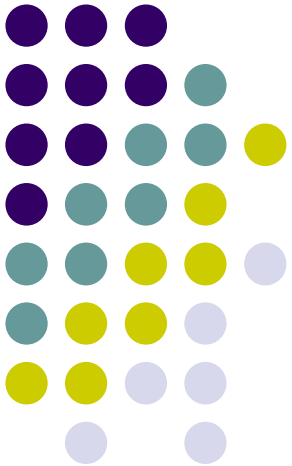
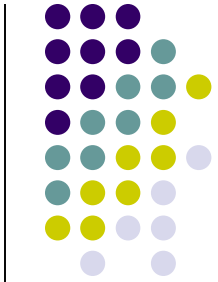
به نام خدا

Lecture 1

آشنايي با محتواي
درس ذخيره وبازيابي اطلاعات
(File Management)



آشنایی با محتوای
درس ذخیره وبازیابی اطلاعات
(File Management)



مهمترین مراجع درس به ترتیب اولویت:

- ***File Structures: An Object-Oriented Approach With C++***

  Authors:

  Folk, Michael J.

  Zoellick, Bill

  Riccardi, Greg

- *سیستم و ساختار فایلها (مهندسی فایلها) ، روحانی رنکوهی*

آشنایی با محتوای
درس ذخیره وبازیابی اطلاعات
(File Management)

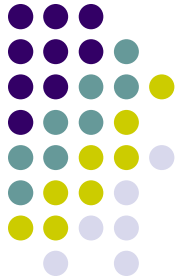
ادامه...

- ***File Organization and Processing***

  Author:

  Tharp, Alan L.

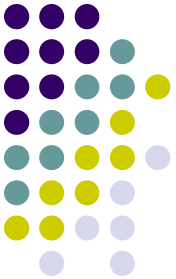
آشنایی با محتوای درس ذخیره وبازیابی اطلاعات (File Management)



موارد مربوط به امتحان:

	تاریخ	بارم از 20 نمره
میان ترم	نیمه اول اردیبهشت	8
پروژه	-	2
پایان ترم	تاریخ اعلام شده	10

آشنایی با محتوای درس ذخیره وبازیابی اطلاعات (File Management)



در این درس چه موضوعاتي مورد نظر ما میباشند؟

انواع عملیات روي داده ها از دیدگاه کامپیوتري کدامند؟

- ✓ ذخیره سازي داده ها (Storage)
- ✓ سازماندهي داده ها (Organization)
- ✓ دسترسي به داده ها (Access)
- ✓ انجام عملیات روي داده ها (Data Processing)

آشنايي با محتواي
درس ذخيره وبازيابي اطلاعات
(File Management)

اين درس با درس ساختمان داده ها چه تفاوتها يا تشابه هايي دارد؟

تشابه:

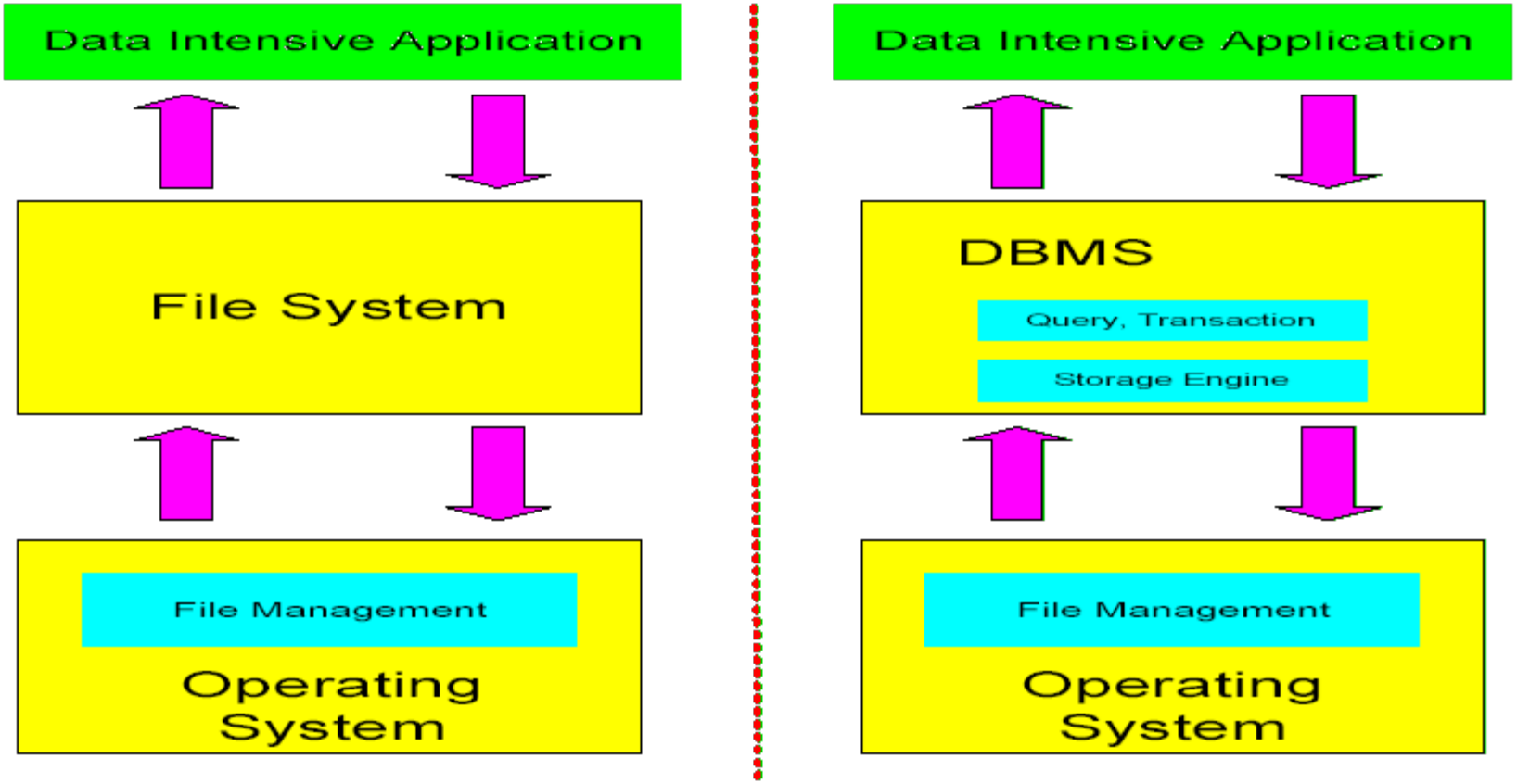
- ✓ هر دو درس در مورد ساختار داده ها و عمليات بر روي آنها بحث مي نمايند.

تفاوت:

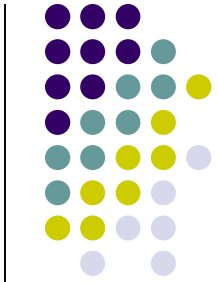
- ✓ در اين درس تاكيد بر عمليات بر روي فايلهاي داده و
- ✓ مسائل مرتبط با انواع حافظه هاي ثانويه (Secondary Storage) ميباشد

آشنایی با محتوای
درس ذخیره وبازیابی اطلاعات
(File Management)

مقایسه با درس پایگاه داده ها:



آشنایی با محتوای درس ذخیره وبازیابی اطلاعات (File Management)

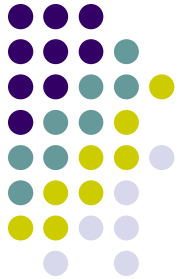


حافظه هاي ثانوي با حافظه اصلي سيستم چه تفاوتها يي دارند؟

حافظه اصلي سيستم (Main Memory) چه خواصي دارد؟

- ✓ سريع (fast) چون الكترونيكي ميباشد
- ✓ كوچک (small) چون قيمت آن بالا مي باشد
- ✓ فرار (Volatile) در صورت قطع برق پاک ميشود
- ✓ سرعت دسترسي به داده : حدود **12 نانو ثانيه**

آشنايي با محتواي
درس ذخيره وبازيابي اطلاعات
(File Management)



حافظه ثانوي با حافظه اصلي سيستم چه تفاوتها يي دارد؟

حافظه ثانوي (Secondary Storage) چه خواصي دارد؟

- ✓ كند (Slow) چون اجزاي مكانيكي دارد
- ✓ بزرگ (Large) چون قيمت آن ارزان است
- ✓ ثابت و پايدار (Stable & Persistent) در صورت قطع برق پاک نميشود
- ✓ سرعت دسترسي به داده : حدود **30 ميلي ثانيه**

آشنايي با محتواي
درس ذخيره وبازيابی اطلاعات
(File Management)

حافظه ثانوي با حافظه اصلي سيستم چه تفاوتها يي دارد؟

اختلاف زمان دسترسي به اين دو حافظه چقدر ميباشد؟

مثال:

- ✓ زمان جستجوي داده در ايندكس يک کتاب (20 sec)
- ✓ زمان جستجوي همان داده بدون ايندكس در يک کتابخانه بزرگ (58 days)

بنابراين دو هدف اصلي اين درس چه خواهد بود؟

1) پايين آوردن زمان دسترسي به داده در حافظه ثانوي
2) پايين آوردن فاصله ميان داده هاي مرتبط با يکديگر

آشنايي با محتواي
درس ذخيره وبازيابي اطلاعات
(File Management)

تاريخچه حافظه هاي ثانوي چگونه بوده است؟

1) در آغاز از باندهاي مغناطيسي (Magnetic Tapes) نه تنها براي نگهداري داده ها بلكه براي انجام عمليات بر آنها نيز استفاده مي شد.

2) تنها امكان دسترسي به داده ها ، دسترسي سري (Sequential Access) بود و زمان انجام عمليات نسبت مستقيم با اندازه فايل داشت.

3) با ورود ديسكهاي مغناطيسي تحولات عظيمي بوجود آمد:

- ✓ دسترسي مستقيم (Direct Access) به داده امكان پذير شد.
- ✓ با اختراع ايندكس ها امكان قرار دادن داده هاي كليدي در فايلهاي كوچكتر و
- ✓ استفاده بهينه از حافظه RAM براي انجام عمليات روي اين فايلها مهيا گشت و
- ✓ سرعت يافتن اطلاعات در فايلهاي داده را بالا برد.

آشنايي با محتواي
درس ذخيره وبازيابي اطلاعات
(File Management)

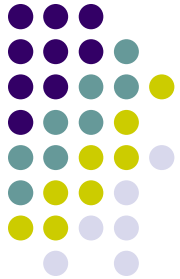
تاريخچه حافظه هاي ثانوي چگونه بوده است؟

4) در سالهاي 1960 ساختارهاي درختواره (Tree Structure) براي بهينه سازي عمليات روي ايندكس ها مطرح شدند.

5) در سال 1979 ساختار B-Tree و سپس B+Tree براي نگهداري فايلهاي داده اختراع شد كه امكان دسترسي به داده را در ميان ميليونها ركورد با 3 يا 4دسترسي به ديسك (I/O)امكانپذير نمود.

6) ساختار Hashing وارد عرصه عمل شد و آرزوي ديرينه دسترسي به هر داده فقط با يک I/O را ميسر ساخت.

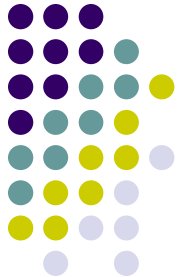
آشنايي با محتواي
درس ذخيره وبازيابي اطلاعات
(File Management)



سر فصلهاي اين درس كدامند؟

- ✓ اطلاعات بر روي ديسكها، نوارها و CD به چه صورتي ذخيره ميشود؟
- ✓ اطلاعات چگونه از روي ديسک خوانده مي شود؟
- ✓ ركوردهاي اطلاعاتي را چگونه مي توان ايجاد و مديريت كرد؟
- ✓ ايندكسها چه قابليتهايي به ما مي دهند؟
- ✓ انواع ايندكس ها كدامند؟
  - ايندكس ساده چيست؟
  - ايندكس دودويي ساده ويا صفحه بندي شده چيست؟
  - B-Tree، B+Tree، B*Tree چيست؟
  - Hash، Linear Hash، Extendible Hash چيست؟

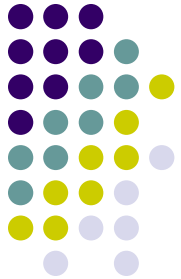
آشنایی با محتوای
درس ذخیره وبازیابی اطلاعات
(File Management)



اشیا در C++ :

- صفات و خصوصیات
- متدها
  - سازنده
  - مخرب
  - دادن بار اضافی به عملگرها
  - سایر

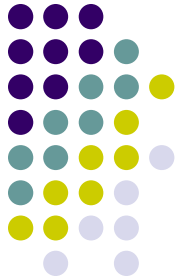
آشنایی با محتوای
درس ذخیره وبازیابی اطلاعات
(File Management)



- مثال:

```
class Person{
   public:
   char name[20],family[20];

   Person();
};
```

آشنایی با محتوای
درس ذخیره وبازیابی اطلاعات
(File Management)



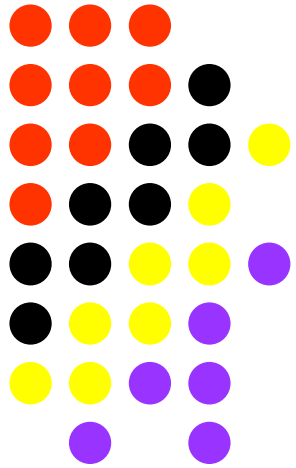
- **یک برنامه نمونه:**

```
main(){
    Person p;
    cout<< “enter name:  ”;
    cin>> p.name;
    cout<< “enter family:  ”;
    cin>> p.family;

    // working with the class
}
```

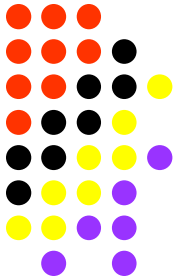
به نام خدا

Lecture 2

عملیات مهم پردازش فایل

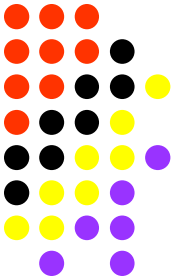


عملیات مهم پردازش فایل



فایل (File) چِیست ؟

- مجموعه اي از داده ها (Data) ميباشد ،
- که بطور واحد بوسيله سيستم عامل (Operating System) قابل شناسايي و مديريت است.
- هر فايل يک واحد مستقل و پايدار (Persistent) از داده ها ميباشد.



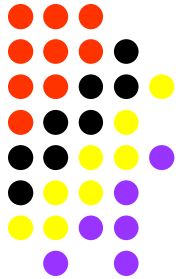
فایل فیزیکی و فایل منطقی

- **فایل فیزیکی**: مجموعه ای از بایتها است که روی دیسک یا نوار ذخیره شده اند. یک دیسک ممکن است حاوی صدها و حتی هزاران فایل فیزیکی باشد.

- **فایل منطقی**: از دید برنامه کاربردی فایل مانند یک کانال دارای نام است که بایتها در آن ریخته شده یا از آن خوانده میشوند.

- برای استفاده از یک فایل فیزیکی باید یک کانال منطقی از برنامه کاربردی به آن فایل فیزیکی برقرار شود،

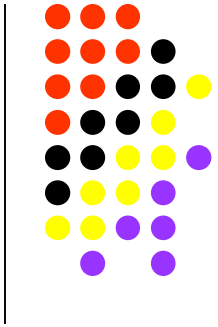
- این کانال منطقی هنگام باز کردن فایل با دستور **open** ایجاد میشود و پس از خاتمه کار با دستور **close** از بین میرود



یک برنامه نمونه

میخواهیم برنامه ای بنویسیم که محتویات یک فایل را روی صفحه نمایش نشان دهد:

- در ابتدا فایل را برای خواندن باز میکنیم
- تا زمانیکه کاراکترهایی برای خواندن وجود داشته باشد:
  - یک کاراکتر از فایل میخوانیم
  - آن کاراکتر را روی صفحه نمایش چاپ میکنیم
- فایل را میبندیم

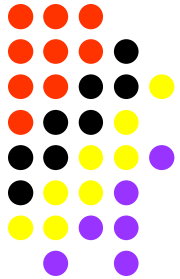


برنامه نمونه

در زبان C:

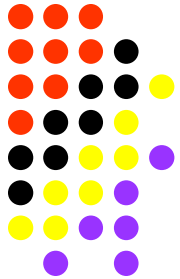
```
#include<stdio.h>
void main(){

        char ch;
        FILE * infile;

        infile = fopen("A.txt", "r");

        while (fread(&ch , 1, 1, infile )!= 0 )
              fwrite(&ch , 1, 1,stdout );

        fclose(infile );
}
```

برنامه نمونه



در زبان C++:

```
#include<fstream.h>
int  main(){
      char ch;
      fstream infile;
      infile.open("A.txt ", ios::in);
      infile.unsetf(ios::skipws);
      while (1) {
              infile>>ch;
              if(infile.fail()) break;
              cout << ch;
      }
      infile.close ();
      return 0;
}
```



باز کردن فایلها در C

- با استفاده از دستور **fopen** انجام میشود:

```
infile = fopen("A.txt", "r");
```

پارامتر اول نام فیزیکی فایل و
پارامتر دوم مد باز کردن فایل میباشدکه:

- r برای خواندن
- w برای نوشتن (در صورت موجود نبودن فایل آنرا ایجاد میکند)
- a برای افزودن به انتهای فایل (در صورت موجود نبودن فایل آنرا ایجاد میکند)
- r+ برای خواندن و نوشتن یک فایل موجود
- w+ برای ایجاد و خواندن و نوشتن (اگر فایل از قبل داده داشته باشد آن داده ها از بین خواهد رفت)

باز کردن فایلها در C++

- با استفاده از متد open از کلاس fstream انجام میشود:

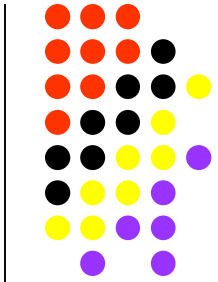
```
fstream infile;
infile.open("A.txt ", ios::in);
```

پارامتر اول نام فیزیکی فایل و
پارامتر دوم مد باز کردن فایل میباشد

باز کردن فایلها در C++

مد باز کردن فایل در C++:

- ios::in برای خواندن
- ios::out برای نوشتن
- ios::app برای افزودن به انتهای فایل
- ios::trunc برای ایجاد یک فایل جدید
- ios::nocreate اگر فایل از قبل وجود نداشته باشد پیام خطا ایجاد میکند
- ios:noreplace یک فال جدید ایجاد میکند اما اگر فایل از قبل وجود داشته باشد پیام خطا ایجاد میکند



بستن فایل

- در زبان C:

```
fclose(infile);
```

- در C++ :

```
Infile.close();
```

خواندن از فایل در زبان C

- با دستور fread انجام میشود:

```
fread(ptr, size, n, file);
```

با اجرای دستور بالا n ایتم size بایتی از file خوانده شده و در ptr کپی میشود.

مثال:

```
fread(&ch, 1, 1, infile);
```

خواندن از فایل در زبان C++

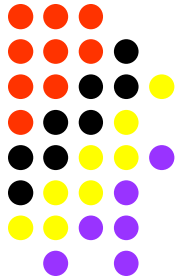
- **مثال:**

```
infile>>ch;
```

استفاده از read:

```
infile.read(&ch,1);
```

نوشتن در فایل در زبان C



- با دستور fwrite انجام میشود:

```
fwrite(ptr, size, n, file);
```

با اجرای دستور بالا n ایتم size بایتی از ابتدای ptr در file نوشته میشود.

مثال:

```
fwrite(&ch, 1, 1, outfile);
```

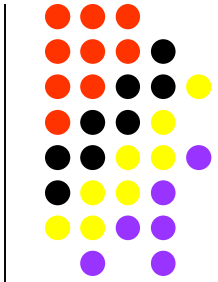
نوشتن در فایل در زبان C++

- **مثال:**

```
outfile>>ch;
```

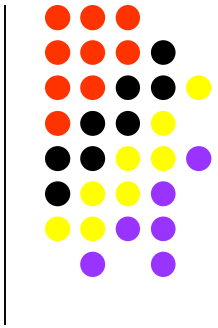
استفاده از write:

```
outfile.write(&ch,1);
```



تشخیص انتهای فایل

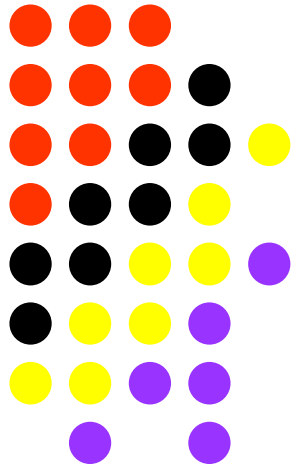
- در C تابع fread در صورت رسیدن به انتهای فایل مقدار 0 برمیگرداند
- در C++ همانطور که در مثال دیدیم از fstream::fail() استفاده میشود. این تابع در صورت رسیدن به انتهای فایل true بر میگرداند.



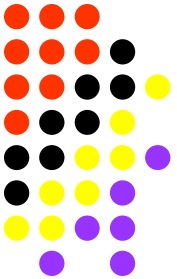
تمرین

- برنامه ای بنویسید که محتویات یک فایل را در فایل جدیدی کپی کند

Lecture 3
A Secondary Storage Device: Magnetic Disk (section 3.1)

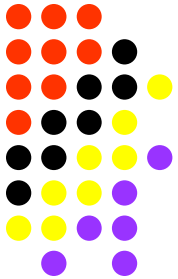


حافظه هاي ثانوي
Secondary Storage Devices



- انواع مختلف حافظه هاي ثانوي كدامند؟
- مقايسه انواع حافظه ها از نظر سرعت و هزينه چگونه ميباشد؟
- چه نوع حافظه براى چه حجم از داده ها مناسب ميباشد؟
- ساختار ديسكهاي سخت چگونه ميباشد؟
- اطلاعات سربار (Non Data Overhead) چيست؟
- زمان دسترسي به ديسكها باعث چه مشكلاتي ميشود؟

حافظه هاي ثانوي
Secondary Storage Devices



انواع مختلف حافظه هاي ثانوي كدامند؟

- **حافظه هاي با دسترسي مستقيم (Direct Access Devices)**
  - ✓ **ديسكهاي مغناطيسي (Magnetic Disks)**
    - ديسكهاي سخت (Hard Disks) : ظرفيت بالا
    - ديسكت ها (Floppy Disks) : ظرفيت پايين و سرعت كم
  - ✓ **ديسكهاي نوري CD-ROM :** ظرفيت بالا
  - ✓ **ديسكهاي نوري DVD :** ظرفيت خيلي بالا

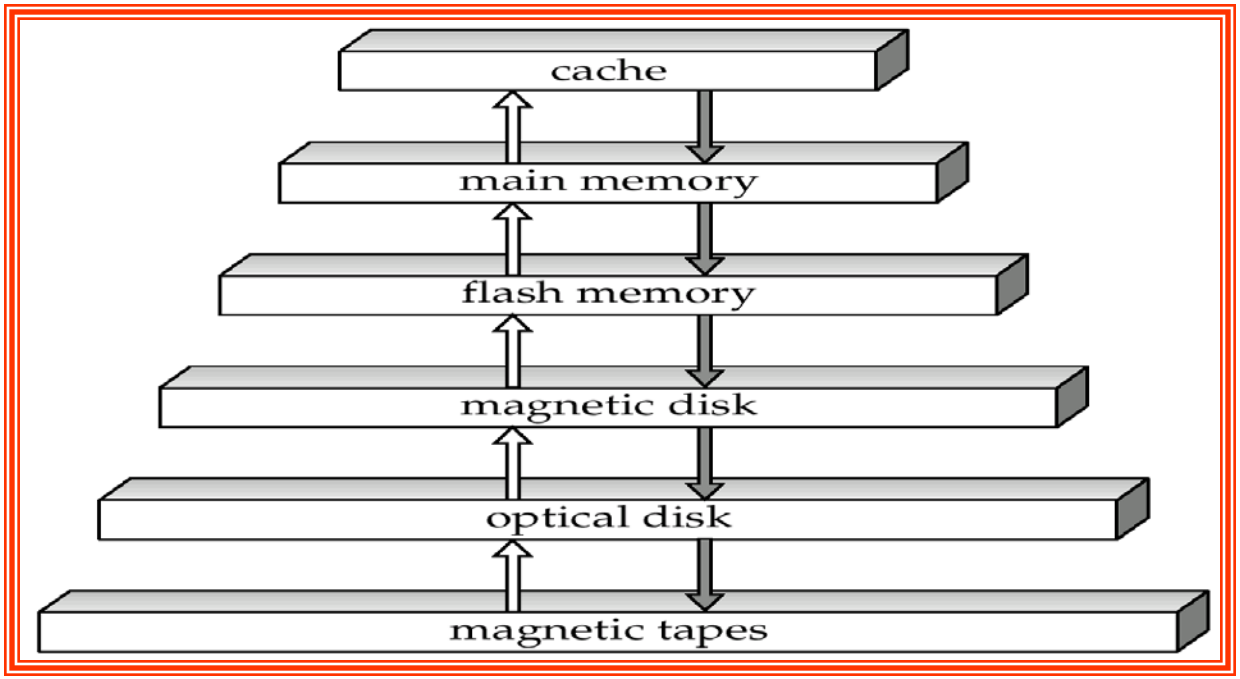
- **حافظه هاي با دسترسي سريال** (Sequential Access Devices)
  - ✓ **نوارهاي مغناطيسي (Magnetic Tapes) :** دسترسي Sequential سريع

مقایسه انواع حافظه ها
Comparing Storage Devices

مقایسه انواع حافظه ها از نظر سرعت و هزینه چگونه میباشد؟ (هزینه؟)

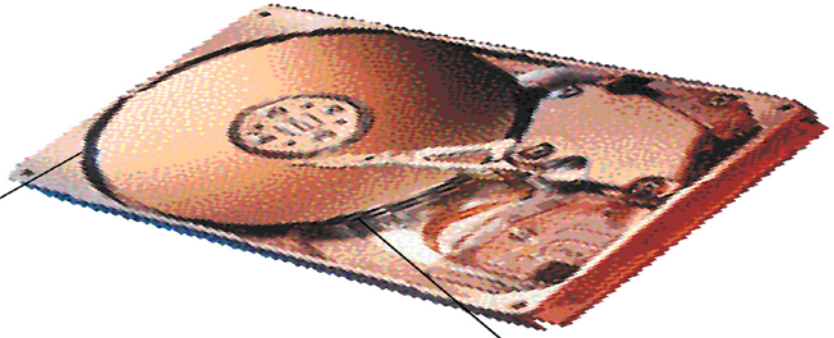
چه نوع حافظه برای چه حجم از داده ها مناسب میباشد؟ (حجم؟)

سرعت بالا - هزینه زیاد - احجام کم داده

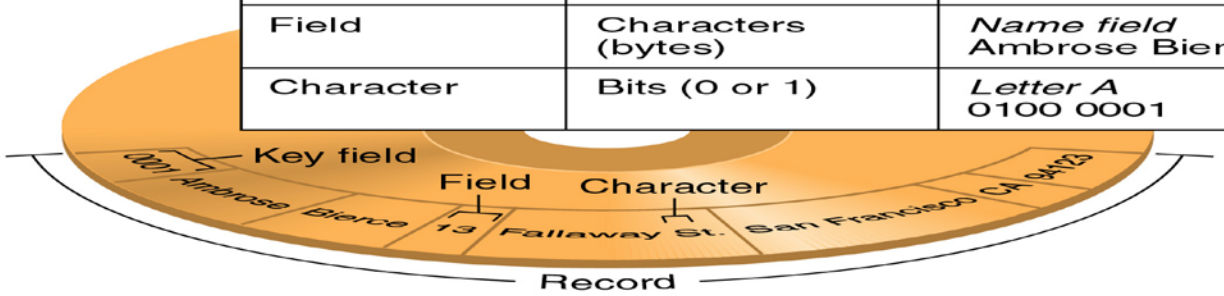


سرعت کم – هزینه پایین - احجام بالاي داده

یک حافظه ثانوی: دیسک مغناطیسی
(Magnetic Disk)



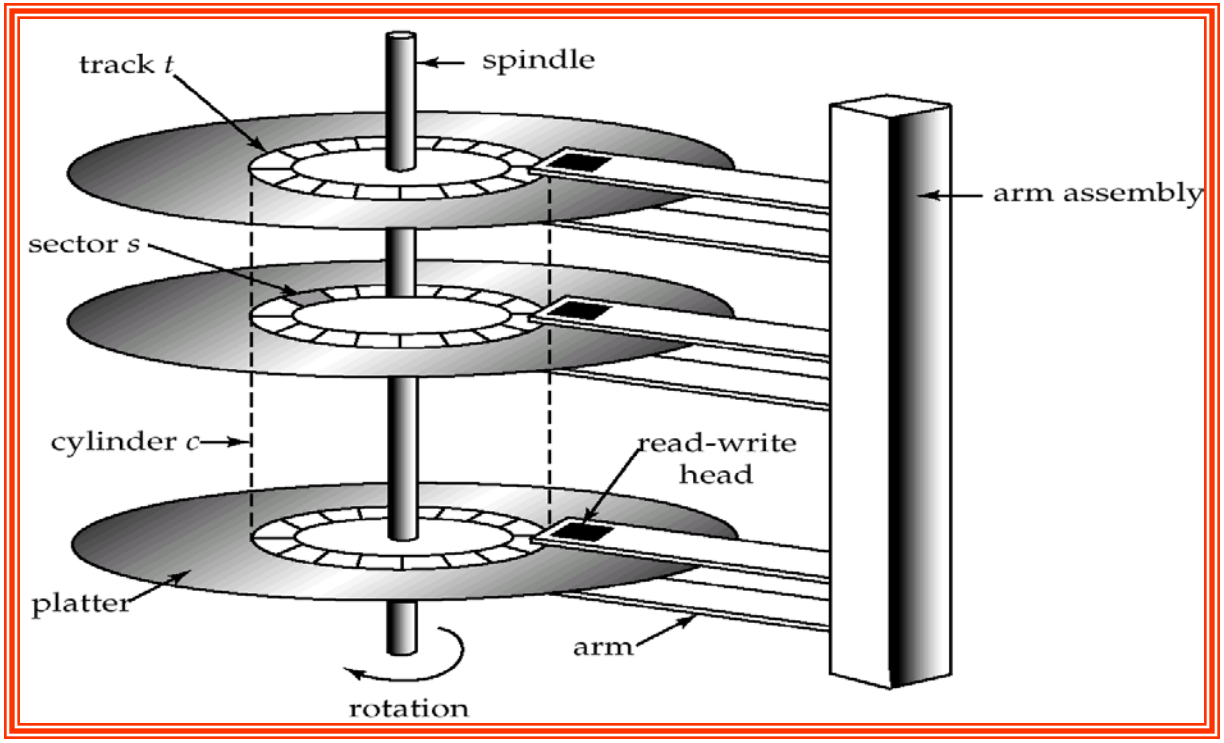
Type of data	Contains	Example
Database	Several files	*Your personal database* Friends' addresses file CD titles file Term papers file etc.
File	Several records	*Friends' addresses file* Bierce, Ambrose 0001 London, Jack 0234 Stevenson, Robert L. 0081 etc.
Record	Several fields	*Ambrose Bierce's address* File no. 0001 13 Fallaway St. San Francisco, CA 94123
Field	Characters (bytes)	*Name field* Ambrose Bierce
Character	Bits (0 or 1)	*Letter A* 0100 0001



ساختار ديسكهاي سخت
(Hard Disks)

ساختار ديسكهاي سخت چگونه ميباشد؟

- مجموعه اي از صفحات مغناطيسي سوار شده روي يک محور که به وسيله تعدادي هد(Head) به طور همزمان خوانده يا نوشته مي شوند.



ساختار ديسكهاي سخت
(Hard Disks)

ساختار ديسكهاي سخت چگونه ميباشد؟

- مجموعه اي از **صفحات مغناطيسي** سوار شده روي يک محور که به وسيله تعدادي **هد(Head)** به طور همزمان خوانده يا نوشته می شوند.

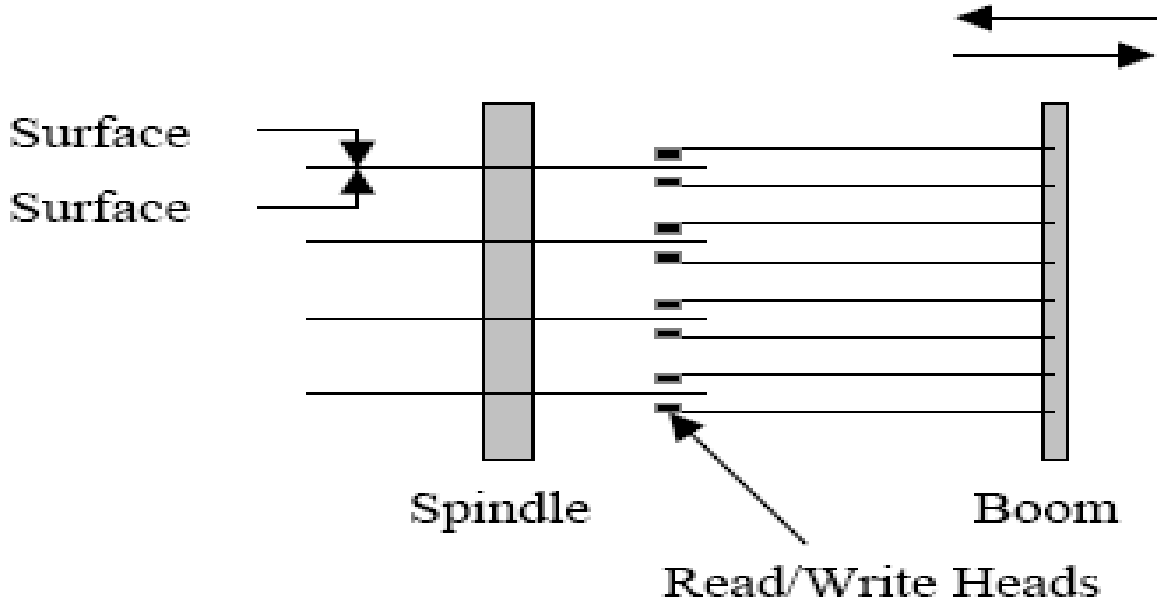


Figure 1: Disk drive with 4 platters and 8 surfaces

ساختار ديسكهاي سخت
(Hard Disks)

شيار (Track) چيست؟

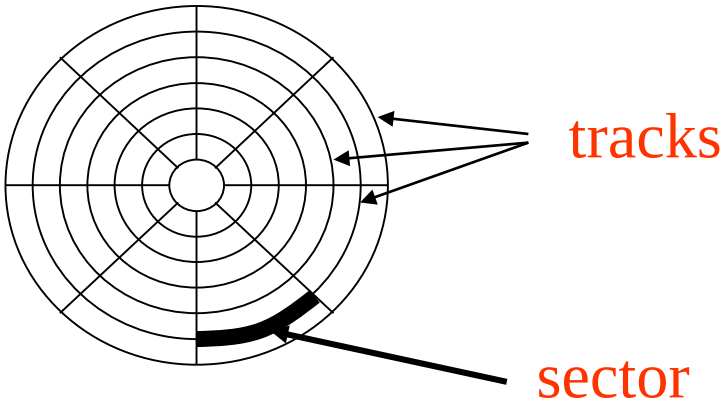
- ✓ هر صفحه به چندين شيار بصورت **دايره هاي متحد المركز** تقسيم ميشوند.

بخش (Sector) چيست؟

- ✓ هر شيار به تعدادي بخش كه **كوچكترين واحد آدرس دهي** (addressable units) ميباشند تقسيم ميشود.

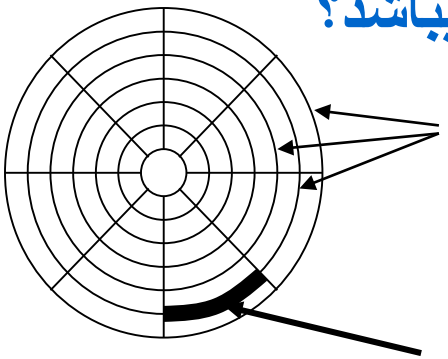
سيلندر (Cylinder) چيست؟

- ✓ **شيارهاي صفحات مجاور** تشكيل يک سيلندرمجازي مي دهند كه بطورهمزمان بوسيله مجموعه هد ها قابل خواندن يا نوشتن ميباشند.



ساختار دیسکهای سخت
(Hard Disks)

ساختار دیسکهاي سخت (Hard Disks) چگونه میباشد؟



- ✓ تعداد سیلندرها ؟

= تعداد شیارها در یک صفحه

- ✓ ظرفیت هر شیار؟

= تعداد سکتور در شیار * تعداد بایت در سکتور

- ✓ ظرفیت هر سیلندر؟

= تعداد سطوح مغناطیسي * ظرفیت هر شیار

- ✓ ظرفیت دیسک؟

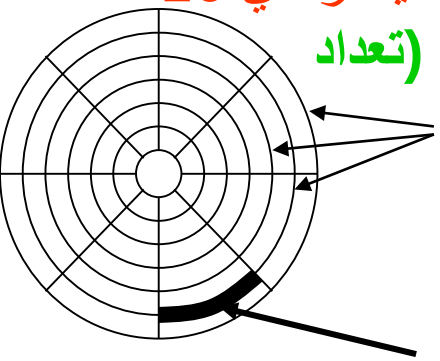
= تعداد سیلندرها * ظرفیت هر سیلندر

يک حافظه ثانوي: ديسک مغناطيسي
(Magnetic Disk)

مثال:

1) فايلي با تعداد 50,000 رکورد 256 بايتي در نظر ميگيريم.
2) ديسکي با سکتورهاي 512 بايتي ، شيارهاي 63 سکتوري و سيلندرهاي 16 شياري به تعداد 4092 در نظر ميگيريم.

(تعداد صفحات؟)



سوال:

چند سيلندر براي نگاهداري اين فايل لازم ميباشد؟

- ✓ تعداد **رکورد** در هر **شيار**= 63 * 2 = 126
- ✓ تعداد **رکورد** در هر **سيلندر**= 126 * 16 = 2016
- ✓ **تعداد سيلندر** لازم = 50000 / 2016 =24.8

(اگر ديسک فضاي آزاد با اين تعداد سيلندر بطور متوالي نداشته باشد ؟)

ساختار ديسكهاي سخت
(Hard Disks)

انواع سازماندهي شيارها روي ديسكهاي سخت چگونه ميباشد؟

1) سازماندهي شيارها بر حسب سكتور

2) سازماندهي شيارها بر حسب بلوك

(ديسكهاي ...)

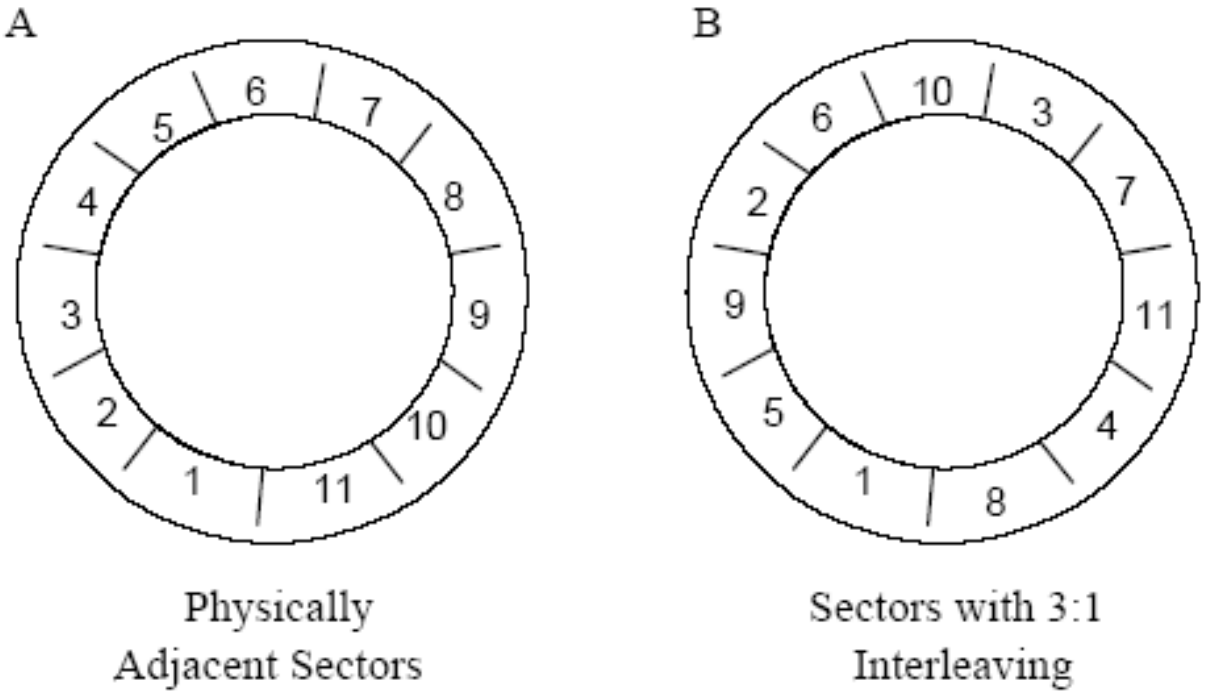
نوع اول: سازماندهي شيارها بر حسب سكتور:

- ✓ هر شيار به چند بخش مساوي به نام سكتور تقسيم ميشود.
- ✓ سكتورها كوچكترين واحد قابل آدرس دهي روي شيار ميباشند .
- ✓ شماره گذاري سكتورها ممكن است بطور متناوب باشد! (چرا؟)

نوع اول: سازماندهی شیارها بر حسب سکتور

شماره گذاري سکتورها چگونه است؟

چرا شماره گذاري سکتورها ممکن است بطور متناوب باشد؟



نوع اول: سازماندهی شیارها بر حسب سکتور

کلاستر(Cluster) چیست ؟

- ✓ تعدادي مشخص و ثابت از **سکتورهاي** متوالي ميباشد. (منطقا متوالي؟)
- ✓ که بوسيله File Manager خوانده ، نوشته ، رزرو يا حذف مي شود.

(چه تعداد؟)

(کجا تعيين ميشود؟)

قسمت (Extent) چیست؟

- ✓ تعدادي کلاستر متوالي (منطقا؟) که **بطور يکجا** براي يک فايل **رزرو** شده باشند.
- ✓ يک extent ميتواند شامل چند **شيار** يا حتي چند **سيلندر** متوالي نيز باشد.

(چه تعداد؟)

(کجا تعيين ميشود؟)

نوع اول: سازماندهي شيارها بر حسب سكتور

ناپيوستگي (Fragmentation) چيست؟

- ✓ تقسيم فضاي ديسک به اجزاء غير قابل استفاده ...

چگونه ناپيوستگي ايجاد ميشود؟

1) **ناهمخواني طول رکوردهاي** يک فايل با **طول سکتورهاي** ديسک ...

- ✓ در صورتي که نخواهيم که يک رکورد روي دو سکتور تقسيم شده باشد !
- ✓ **مثال:**
  - اگر طول رکورد 300 و
  - طول سکتور 512 باشد
  - براي هر رکورد 212 بايت بي استفاده خواهد ماند.

نوع اول: سازماندهي شيارها بر حسب سكتور

چگونه ناپيوستگي ايجاد ميشود؟ (ادامه...)

2) ناهمخواني طول فايل با طول كلاسترها ...

- ✓ با فرض اينكه هر extent برابر با يک كلاستر باشد
- ✓ ممكن است آخرين كلاستر فايل فضاي خالي داشته باشد.
- ✓ **مثال:**
  - اگرفايلي به طول يک بايت و
  - هر كلاستر برابر با سه سكتور512 بايتي باشد،
  - در اين صورت 1535 بايت از فضاي رزرو شده بي استفاده خواهد ماند.

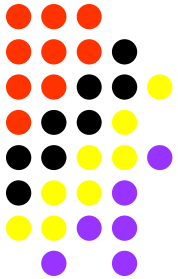
ساختار ديسكهاي سخت (Hard Disks)

نوع دوم: سازماندهی شیارها برحسب بلوک:

- ✓ هر شيار به چند بخش به نام بلوک تقسيم ميشود.
- ✓ بلوکها هيچ ربطي با سکتورها ندارند! (چرا؟)
- ✓ تعداد رکوردها در هر بلوک را فاکتور بلوک (Blocking Factor) ميناميم.
- ✓ هر بلوک شامل چند قسمت (subblock) ميباشد:

1) Count Sub Block : حاوي طول بلوک بر حسب بايت.
2) Key subblock :حاوي کليد دسترسي (Hard) به بلوک.
3) Data subblock :حاوي داده هاي بلوک.

ساختار ديسكهاي سخت
(Hard Disks)



اطلاعات سربار (Non Data Overhead) چيست؟

- ✓ انواع داده هاي مخصوص سيستم مديريت ديسک ...
- ✓ كه ربطي به داده هاي فايلها ندارند.

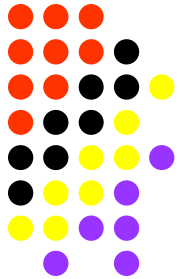
اطلاعات سربار در ديسكهاي سكتوري كدامند؟

- ✓ آدرس سكتور ، آدرس شيار ، شرط صحت سكتور (Condition)
- ✓ و نيز فضاي خالي (Gap) بين دو سكتور.

اطلاعات سربار در ديسكهاي بلوكي كدامند؟

- ✓ زير بلوكهاي غيرداده اي (Count و KEY)
- ✓ و نيز فضاي خالي (Gap) بين بلوكها.

اطلاعات سربار
(Non Data Overhead)



مثال:

1) یک دیسک بلوکي با شیارهاي 20,000 بایتي و با 300 بایت اطلاعات سربار بر هربلوک در نظر میگیریم،
2) تعداد رکوردهاي 100 بایتي در هر شیار را براي دو حالت مختلف حساب میکنیم:

حالت اول: اگر هر بلوک حاوي 10 رکورد باشد:

- ✓ فاکتور بلوک = 10
- ✓ داده هاي هر بلوک = 1000 = (10*100)
- ✓ اطلاعات سر بار= 300
- ✓ تعداد بلوک در هر شیار= 20000 / 1300 = 15.38 = **15**
- ✓ تعداد رکورد در هر شیار = (15*10) = **150**

(**Fragmentation rate?**)

اطلاعات سربار
(Non Data Overhead)

مثال (ادامه...):

1) یک دیسک بلوکي با شیارهاي 20,000 بایتي و با 300 بایت اطلاعات سربار بر هربلوک در نظر میگیریم،
2) تعداد رکوردهاي 100 بایتي در هر شیار را براي دو حالت مختلف حساب میکنیم:

حالت دوم: اگرهر بلوک حاوي 60 رکورد باشد:

- ✓ فاکتور بلوک = 60
- ✓ داده هاي هر بلوک = 6000
- ✓ اطلاعات سر بار= 300
- ✓ تعداد بلوک در هر شیار = 20,000 / 6300 = 3.17 = **3**
- ✓ تعداد رکورد در هر شیار= 60 * 3 = **180**

(**Fragmentation rate?**)

زمان دسترسی به دیسکهای سخت
(Hard Disks Access Time)

زمان دسترسي به داده هاي ديسكهاي سخت چگونه ميباشد؟

زمان دسترسي به داده هاي يک سكتور چگونه ميباشد؟

1) **زمان جستجو (Seek Time) :**

- ✓ براي قرار گرفتن هد روي سيلندر مورد نظر

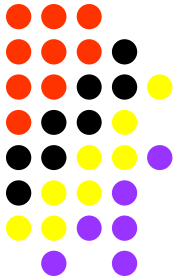
2) **تاخير چرخشي (Rotational Delay) :**

- ✓ براي قرار گرفتن هد روي سكتور مورد نظر

3) **زمان انتقال داده (Transfer Time) :**

- ✓ براي خواندن يا نوشتن داده هاي سكتور

زمان دسترسی به دیسکهای سخت
(Hard Disks Access Time)



زمان دسترسي به داده هاي ديسكهاي سخت چگونه ميباشد؟

مثال: ديسكي با مشخصات زير در نظر ميگيريم:

- ✓ زمان متوسط جستجو = **8** ميلي ثانيه (Average Seek Time)
- ✓ تاخير متوسط چرخشي = **3** ميلي ثانيه (Average Rotation Time)
- ✓ تاخير چرخشي ماكزيمم = **6** ميلي ثانيه (Maximum Rotation Time)
- ✓ سرعت چرخش = **10000** دور در دقيقه (RPM) **(رابطه با قبلي؟)**
- ✓ تعداد سكتورها در هر شيار = **170**
- ✓ اندازه هر سكتور = **512** بايت

زمان دسترسي به ديسكهاي سخت
(Hard Disks Access Time)

مثال (ادامه...)

سوال (1) : زمان متوسط براي خواندن يک سکتور؟

- ✓ زمان انتقال يک سکتور = حاصل تقسيم (زمان چرخش) بر (تعداد سکتور در شيار)
- ✓ زمان انتقال يک سکتور= (60sec/10,000)/170 = 0.035 ميلي ثانيه
- ✓ زمان متوسط خواندن يک سکتور = حاصل جمع:
  - ❑ زمان متوسط جستجو ،
  - ❑ تاخير متوسط چرخشي و
  - ❑ زمان انتقال يک سکتور
- ✓ زمان متوسط خواندن يک سکتور= 0.035+3+8 = **11.035** ميلي ثانيه

زمان دسترسي به ديسكهاي سخت
(Hard Disks Access Time)

مثال (ادامه...)

حال فايلي با مشخصات زير را در نظر ميگيريم:

- ✓ تعداد ركوردها = **34,000**
- ✓ اندازه يک ركورد = **256** بايت
- ✓ تعداد شيارها **(غير متوالي؟)** = **100** **(چرا غير متوالي؟)**

سوال (2) : زمان خواندن فايل با دسترسي Sequential ؟

- ✓ زمان متوسط جستجو = **8** ميلي ثانيه
- ✓ تاخير چرخشي متوسط = **3** ميلي ثانيه
- ✓ زمان انتقال متوسط براي يک شيار= **60/10,000=6** ميلي ثانيه
- ✓ زمان كل براي خواندن شيار = **8+3+6=17** ميلي ثانيه
- ✓ زمان كل براي خواندن فايل = **17*100=1.7** ثانيه

زمان دسترسی به دیسکهای سخت
(Hard Disks Access Time)

مثال (ادامه...)

- همان مشخصات قبل را در نظر میگیریم:

✓ تعداد رکوردها = **34,000**

✓ اندازه یک رکورد = **256** بایت

✓ تعداد شیارها (غیر متوالي؟) = **100** **(چرا غیر متوالي؟)**

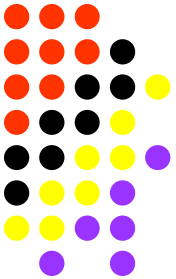
سوال (3) : زمان خواندن فایل با دسترسي مستقیم (Random) ؟

✓ زمان متوسط خواندن یک رکورد = زمان متوسط خواندن یک سکتور **(چرا؟)**

✓ زمان متوسط خواندن یک رکورد = **11.035** میلي ثانیه

✓ زمان کل براي خواندن فایل= **34,000 *11.035** = **371.1** ثانیه

زمان دسترسی به دیسکهای سخت
(Hard Disks Access Time)



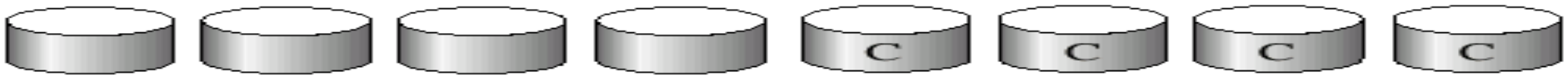
زمان دسترسي به ديسكها باعث چه مشكلاتي ميشود؟

- همواره **CPU** و شبكه (**Network**) منتظر ديسكها ميباشند!

چه راه حلهايي وجود دارد؟

1) پردازنده (CPU) به **چند كاربر** سرويس دهد. (**Multiprocessing**) **(چرا؟)**
2) فايلهاي خيلي بزرگ روي **چند ديسک** تقسيم شوند. (**Disk Striping**) **(چرا؟)**
3) استفاده از ديسكهاي **RAID** جهت **تقسيم** هر **بلوک داده** روي ديسكهاي مختلف.
4) استفاده از ديسكهاي **RAM** كه رفتار يک ديسک (يا ديسكت) را سيموله مي كنند.
5) استفاده از **Disk Caching** براي جواب دادن **سريع** به درخواستهاي I/O. **(چگونه؟)**

(a) RAID 0: non-redundant striping



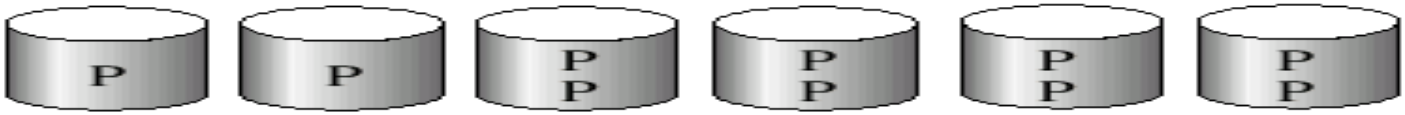
(b) RAID 1: mirrored disks

(c) RAID 2: memory-style error-correcting codes

(d) RAID 3: bit-interleaved Parity

(e) RAID 4: block-interleaved parity

(f) RAID 5: block-Interleaved distributed parity

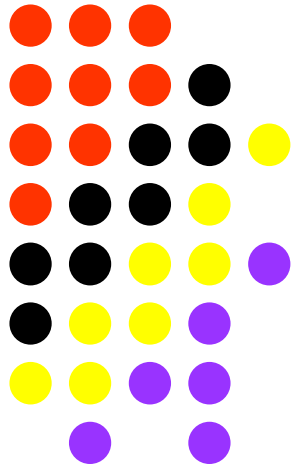


(g) RAID 6: P + Q redundancy

Lecture 4

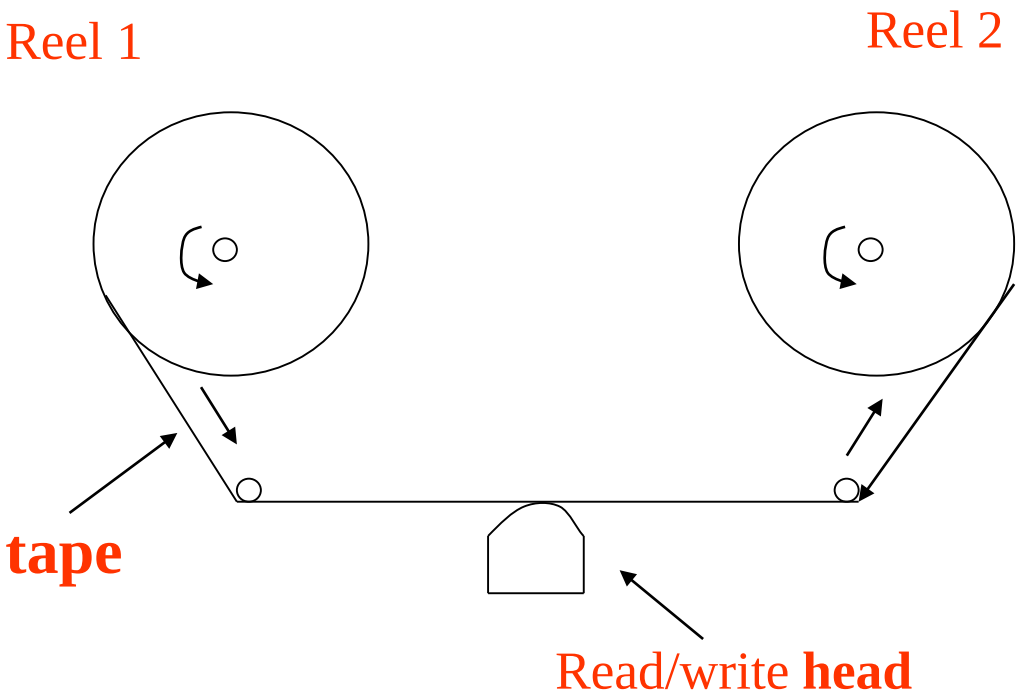
A Secondary Storage Device: Magnetic Tape

(sections 3.2 – 3.3)

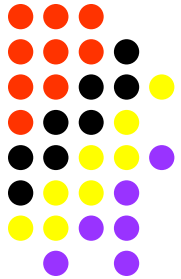


یک حافظه ثانوی: باند مغناطیسی (Magnetic Tape)

باند مغناطیسی (Magnetic Tape) چیست؟



خواص باند مغناطیسی



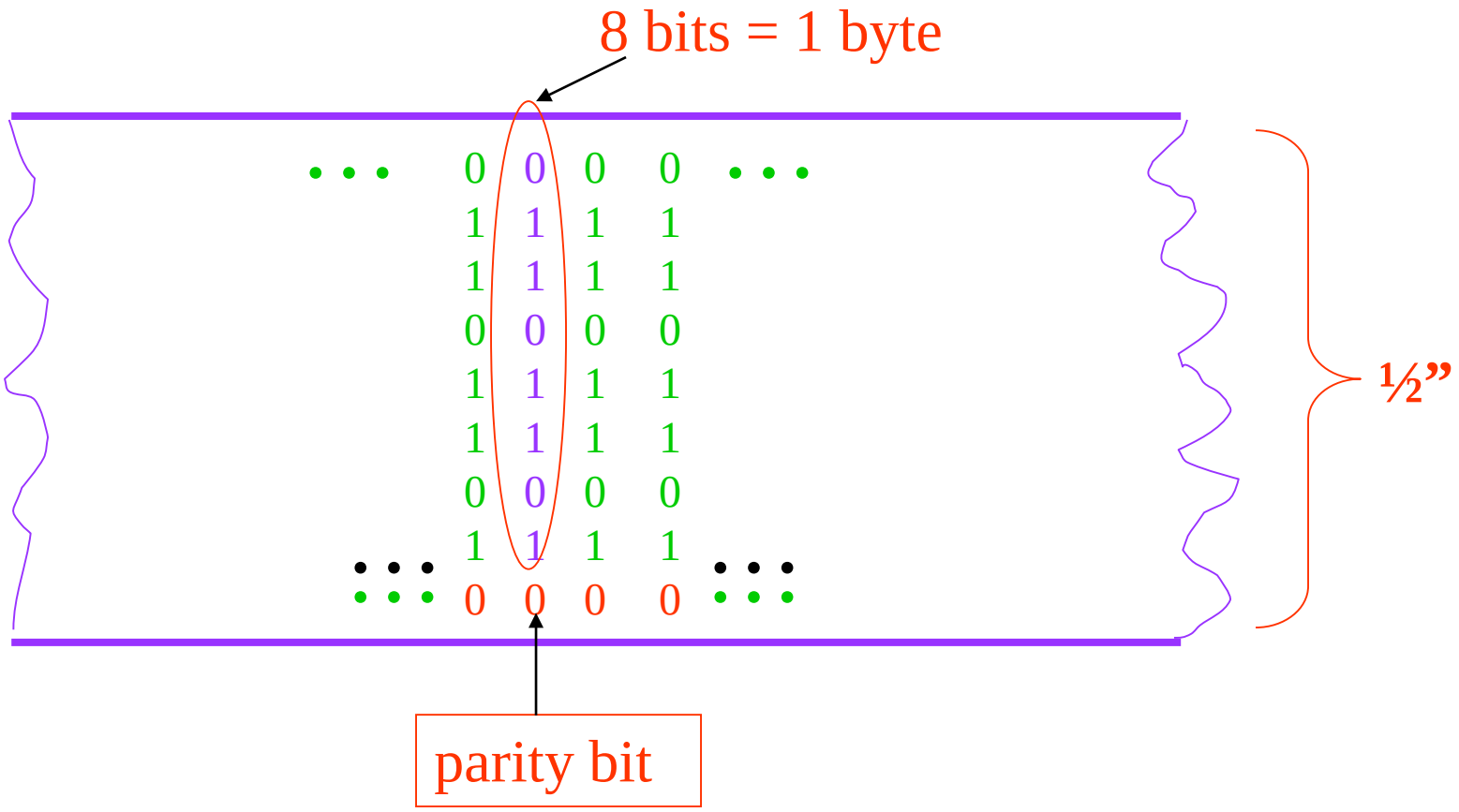
خواص باند مغناطيسي چيست؟

- ✓ امكان دسترسي مستقيم (direct access) به ركوردها را **نمي دهد!** (چرا؟)
- ✓ ولي امكان دسترسي سري (sequential access) را با سرعت بالا دارد.
- ✓ در مقابل شرايط مختلف محيطي (environment) **پايداري** خوبي دارد.
- ✓ براحتي حمل و **نگهداري** مي شود .
- ✓ از ديسكهای سخت **ارزانتر** است.
- ✓ در گذشته براي نگهداري **فايلهای بزرگ** (بجاي ديسكهای سخت) استفاده مي شد. (چرا؟)
- ✓ ولي اكنون **فقط** براي آرشيو داده ها (backup) استفاده ميشود. (چرا؟)

ساختار باند مغناطیسی

ساختار یک باند مغناطیسي چگونه است؟

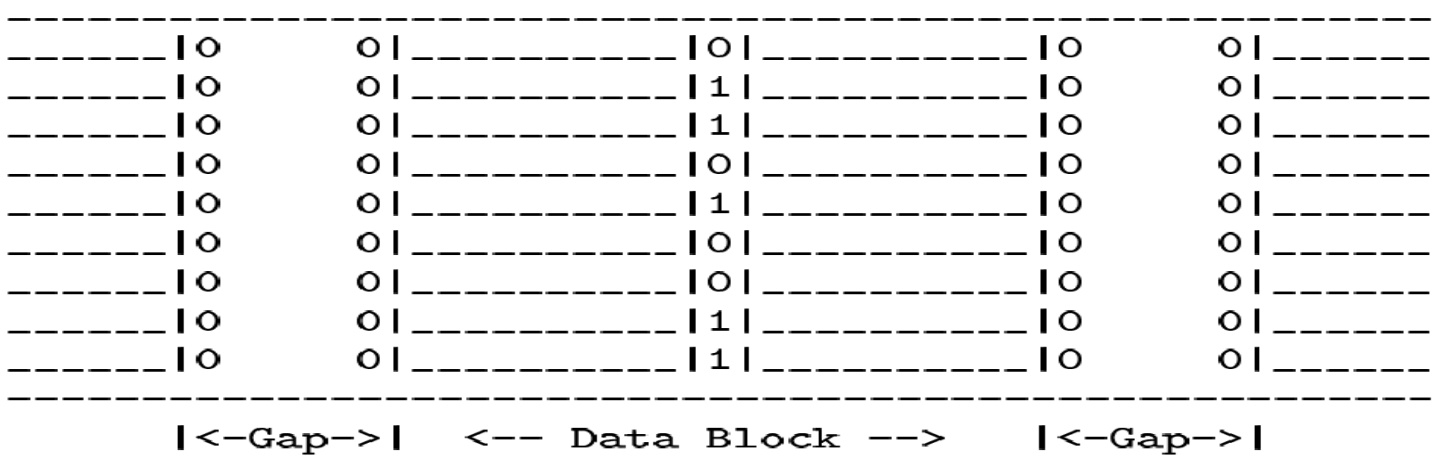
باند مغناطیسي 9 شیاری (Nine-Track Tapes) چیست؟



ساختار باند مغناطیسی

باند مغناطیسي 9 شیاری (Nine-Track Tapes) چیست؟

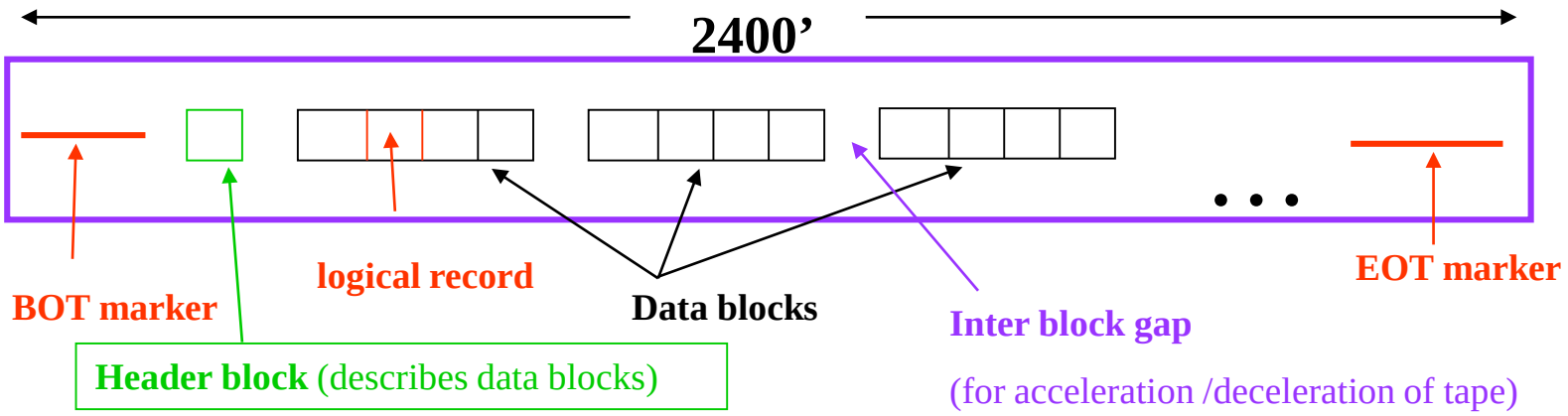
- ✓ داده ها روی **9 شیار موازي** به صورت **دنباله اي از بیت ها** (bits) ثبت میشوند.
- ✓ بنابراین **هر مقطع** از باند (Frame) شامل **9 بیت** و معادل **یک بایت داده** میباشد.
- ✓ در هر **9** بیت ، **هشت بیت** براي داده ها و **یک بیت** براي کنترل صحت (Parity) داده ها وجود دارد. (چگونه؟)



ساختار باند مغناطیسی

ساختار یک باند مغناطیسي چگونه است؟

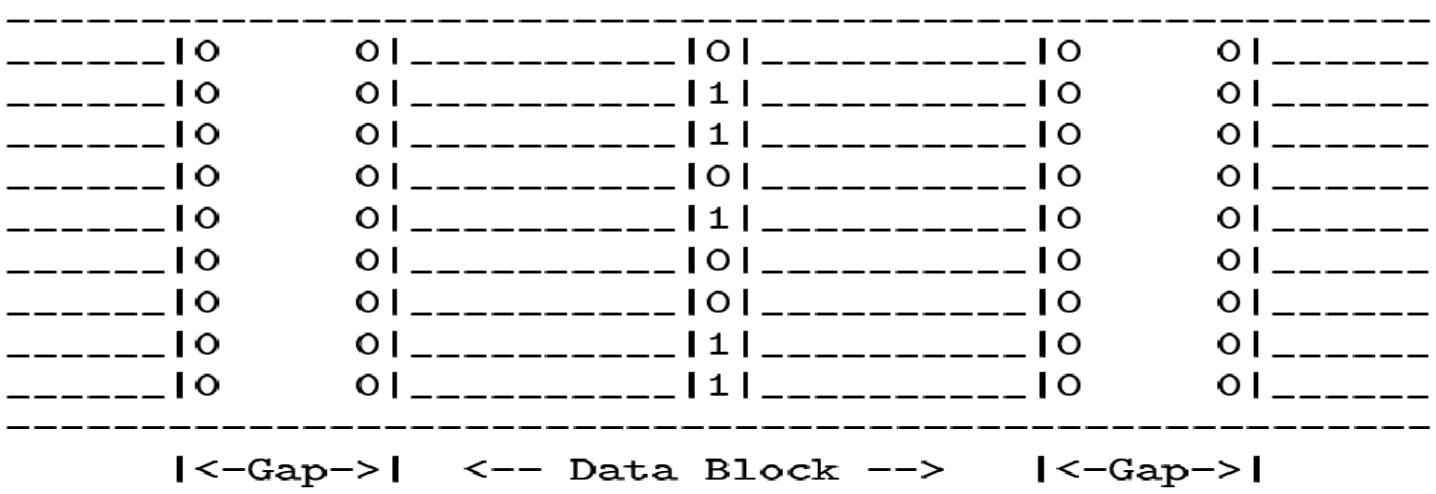
- ✓ **ترتيب منطقي** (Logical Position) هربايت همان **ترتيب فيزيكي** آن ميباشد.
- ✓ يعني **ركوردها** به طور مرتب **پشت سرهم** قرار دارند. (Sequential Access)



ساختار باند مغناطیسی

ساختار یک باند مغناطیسي چگونه است؟

- ✓ در **فاصله بین بلوک داده ها** فضاي استفاده نشده اي به نام **Gap** وجود دارد که در آن تمام بیتها **صفر** مي باشند
- ✓ در گذشته نوع رایج باندها حاوي **6250** بایت دراینچ (**bpi**) بوده است.
- ✓ انواع **جدیدتر** باندها **30,000 bpi** یا **بیشتر** میباشند



ظرفیت باند مغناطیسی

تخمین طول باند مورد نیاز چگونه است؟

راندمان استفاده از باند به چه عواملي بستگي دارد؟

1) **فشردگي** داده هاي باند (Tape **density**)
   - ✓ مثال: **6250** bpi
2) **سرعت** حرکت باند (Tape **Speed**)
   - ✓ مثال: **200** ips
3) **فاصله** بین بلوکهاي داده (inter block **gap**)
   - ✓ مثال: **0.3** inch

ظرفیت باند مغناطیسی

تخمین طول باند مورد نیاز چگونه است؟

مثال:

- ✓ فايلي با مشخصات 1,000,000 رکورد 100 بايتي را در نظر ميگيريم.
- ✓ طول باند مورد نياز را با فشردگی **bpi 6250** براي دو حالت مختلف حساب ميکنيم.

حالت اول: فاکتور بلوک (blocking factor) برابر با یک:

- ✓ طول مورد نياز= **تعداد** بلوک ها * (**طول** هر بلوک + **فاصله** بين بلوکها)
- ✓ طول مورد نياز=(100/6250+0.3)*1,000,000=316,000 اينچ
- ✓ يعني **بيش از 10 کارتريج 2400** فوتي (Cartridge)
- ✓ درگذشته رايجترين طول کارتريجها **2400** فوت بوده است.
- ✓ اکنون کارتريجهاي معمول با طول **3600 فوت** يا **بيشتر**ميباشند.

(ظرفیت؟)

ظرفیت باند مغناطیسی

تخمین طول باند مورد نیاز چگونه است؟

مثال (ادامه...):

- ✓ همان مشخصات **1,000,000** رکورد **100** بایتي را در نظر میگیریم.
- ✓ طول باند مورد نیاز را با فشردگی **6250 bpi** براي حالت دوم حساب میکنیم.

حالت دوم : فاکتور بلوک (blocking factor) برابر با 50 :

- ✓ طول هر بلوک = (100/6250)*50 = **0.8** اینچ
- ✓ تعداد بلوکها = 1,000,000/50 = **20,000**
- ✓ طول باند مورد نیاز = (0.8+0.3)*20,000 = **22,000**اینچ = **1833** فوت
- ✓ فضاي به هدر رفته بین بلوکها در حالت اول **300,000** اینچ بود،
- ✓ ولي در حالت دوم فقط **6,000** اینچ میباشد! **(چرا؟)**

ظرفیت باند مغناطیسی

فشردگي حقيقي يا موثر (Effective Recording Density) چيست؟
و چگونه محاسبه ميشود؟

E.R.D = طول مورد نیاز هر بلوک / تعداد بایت در بلوک

- ✓ **در حالت اول:**

E.D.R = 100 / 0.316 = **316.4** bpi

- ✓ **در حالت دوم:**

E.D.R = 5,000 / 1.1 = **4545** bpi

- ✓ این اعداد را با **فشردگي نامي باند** (Nominal Density) که برابر **6250** bpi میباشد مقایسه کنید!
  (توضیح؟)

سرعت انتقال داده ها
در باند مغناطیسی

سرعت انتقال داده ها در باند مغناطیسی چگونه است؟

سرعت انتقال نامي (Nominal Transmission Rate) چیست؟

سرعت انتقال حقيقي يا موثر (Effective Transmission Rate) چیست؟

- ✓ سرعت انتقال نامي = فشردگي نامي باند * سرعت باند

سرعت انتقال نامي = 6250 * 200 = 1250 (kB/sec)

- ✓ سرعت انتقال حقيقي = فشردگي حقيقي باند * سرعت باند

سرعت انتقال داده ها
در باند مغناطیسی

سرعت انتقال حقیقي يا موثر (Effective Transmission Rate) چیست؟

- ✓ **سرعت انتقال حقیقي = فشردگي حقیقي باند * سرعت باند**

- ✓ **در حالت اول:**

E.T.R = 316.4 * 200 = 63,280 byte/sec =**63.3** kB/sec

- ✓ **در حالت دوم:**

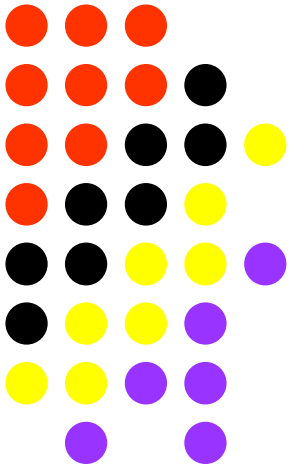
E.T.R = 4545 * 200 = 909,080 byte/sec = **909** kB/sec

- ✓ این اعداد را با **سرعت انتقال نامي** باند (**1250**) مقایسه کنید! **(توضیح؟)**

Lecture 5

A Secondary Storage Device: CD-ROM

(sections 3.4 – 3.6)

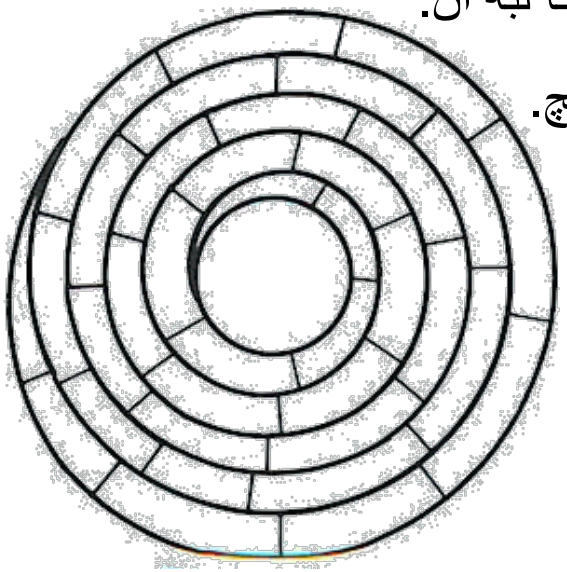


یک حافظه ثانوي: ديسک نوري (CD-ROM)

ديسک نوري (Compact Disk – read only! memory) چيست؟

يک صفحه دايره شکل و منعکس کننده نور(ليزري)، حاوي:

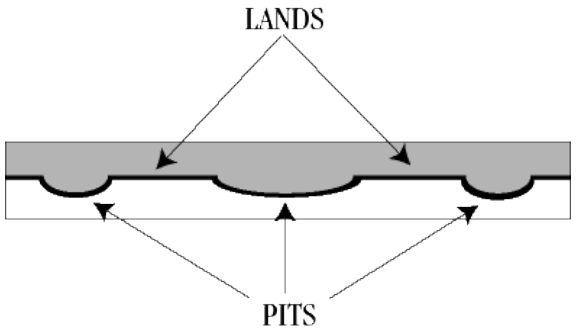
- ✓ **يک پيست مارپيچ** (spiral) از مرکز صفحه تا لبه آن.
- ✓ بعلاوه **تعدادي حفره** (Pits) روي پِيست مارپيچ.



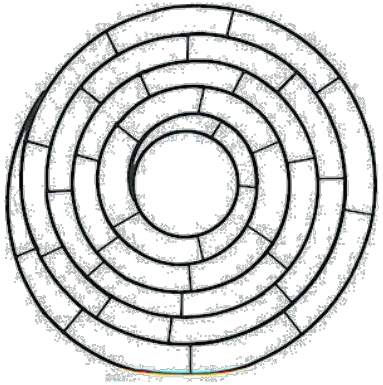
(640-700 MB per platter)

(چرا ROM ؟)

(چرا نور ليزري؟)

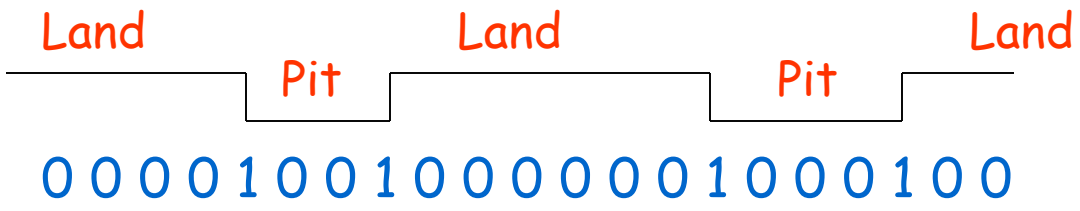


خواص دیسک نوری

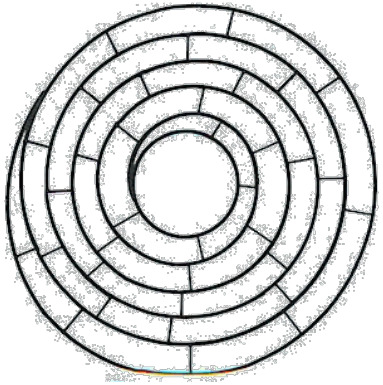


خواص دیسک نوري (یا لیزري) چیست؟

- ✓ داده ها به کمک **تشعشع ليزري** **نوشته** یا **خوانده** میشوند.
- ✓ ظرفیت آن حدود **600** تا **700** **مگا بایت** داده میباشد.
- ✓ تنها یک **شیار مارپیچ** طولاني شامل تعداد زیادي **سکتور** دارد.
- ✓ داده هاي **ديجيتالي** بصورت یک سري **حفره** روي این شیار ثبت میشوند.
- ✓ به سطح بالایي شیار **Land** و به حفره ایجاد شده روي شیار **Pit** گفته میشود.



خواندن ديسک نوري

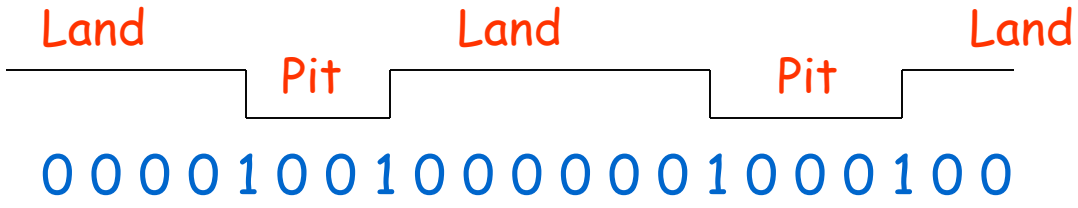


عمل خواندن ديسک نوري چگونه است؟

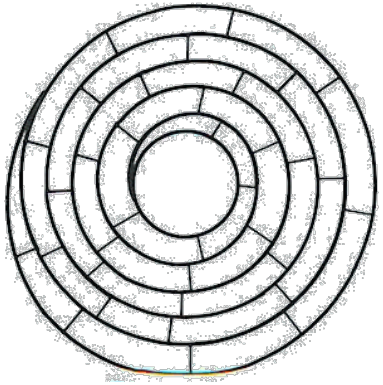
- ✓ به وسيله تابش نور ليزري روي شيار.
- ✓ و تشخيص تغييرات درشدت انعکاس نور (intensity).

تشخيص صفر و يک چگونه است؟

- ✓ عدد يک = تغيير ارتفاع (از Land به Pit يا بر عکس).
- ✓ عدد صفر = تعداد فواصل زماني معين بين دو عدد يک. (فواصل زماني؟)



خواندن ديسک نوري



تشخيص صفر و يک چگونه است؟

- ✓ **مابين دو عدد يک**، بايستي **لااقل دو عدد صفر** وجود داشته باشد! **(چرا؟)**
- ✓ براي **کد گذاري** **256** **حروف** جدول ASCII احتياج به **14** **بيت** خواهد بود! **(چرا؟)**
- ✓ **تبديل کد** گذاري حروف از **8** بيت به **14** بيت بکمک يک جدول (**E**ight to **F**ourteen **M**odulation) انجام ميشود.

مثال:

- ✓ نمونه اي از جدول E.F.M. :

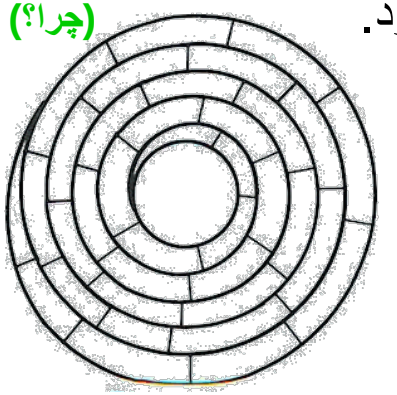
0 → 0000 0000 → 0100 1000 100000

1 → 0000 0001 → 1000 0100 000000

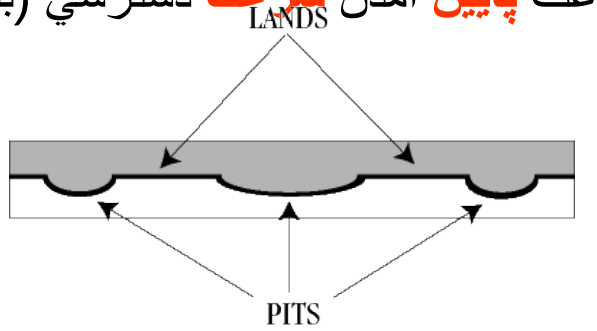
2 → 0000 0010 → 1001 0000 100000

سرعت وظرفیت دیسک نوری

روش سرعت خطي ثابت (Constant Linear Velocity) چیست؟



- ✓ حرکت نور ليزري روي شيار با **سرعت خطي ثابت** انجام ميشود. (چرا؟)
- ✓ **طول شيار** مار پيچ (Spiral track) تقريبا **سه مايل** ميباشد.
- ✓ **طول سکتورها** از مرکز تا لبه ديسک همواره **ثابت** است.
- ✓ اين تکنولوژي از **ديسک هاي صوتي** به **ارث** گرفته شده،
- ✓ و باعث بالا بردن **ظرفيت** ديسک (**تا دو برابر**) ميشود. (چرا؟)
- ✓ ولي باعث **پايين** آمدن **سرعت** دسترسي (بين نيم تا يک ثانيه) نيز ميگردد. (چرا؟)



آدرس دهي ديسک نوري

روش آدرس دهي (Addressing) چگونه است؟

- روش سيلندر:شيار:سکتور نمي تواند جواب دهد! (چرا؟)
- ولي فاصله زماني يک سکتور نسبت به مبداء شيار (Root) قابل اندازه گيري ميباشد.
- **روش آدرس دهي زماني چگونه است؟**
  - ✓ هر ثانيه چرخش به 75 سکتور تقسيم ميشود.
  - ✓ اندازه هر سکتورمعادل 2 KB داده ميباشد.
  - ✓ طول شيار هر ديسک معادل لااقل 60 دقيقه پيمايش ظرفيت دارد.
  - ✓ ظرفيت ديسک = 60*60*75= 270000 سکتور= 540000 کيلو بايت ميگردد.
- هر سکتور بکمک شاخص "minute:second:sector" آدرس دهي ميشود.

مثال:

- ✓ شاخص 16:22:34 آدرس 34مين سکتور در دقيقه 16 و ثانيه 22 ميباشد.

مقايسه ديسک نوري با ديسک مغناطيسي

مزايا و معايب ديسک نوري درمقايسه با ديسک مغناطيسي چيست؟

Magnetic Disk	CD-ROM
✓ روش سرعت زاويه اي ثابت (CAV) ✓ سازمان دهي سکتورها روي چندين شيار متحد المرکز ✓ طول زاويه اي سکتورها ثابت ✓ عدم استفاده از ظرفيت فضاي سکتور هاي نزديک به لبه خارجي ✓ **حسن:** استفاده از سرعت چرخشي ثابت ✓ **عيب:** عدم استفاده از ظرفيت ماکزيمم	✓ روش سرعت خطي ثابت (CLV) ✓ سازمان دهي سکتور ها روي يک شيار مارپيچ ✓ طول خطي سکتور ها ثابت ✓ امکان استفاده از بالاترين ظرفيت فضاي سکتورها ✓ **حسن:** بالا بودن ظرفيت ✓ **عيب:** لزوم سرعت چرخشي متغير (آهسته تر به طرف لبه خارجي)

ساختار یک سکتور

ساختار یک سکتور چگونه است؟

- ✓ هر سکتور حاوي داده هاي گوناگوني ميباشد:

12 bytes Synch.	4 bytes Sector ID	2048 **bytes** User data	4 bytes error detection	8 bytes null	276 bytes Error correction

امکان خطا چگونه است؟

- ✓ امکان خطا: **یک** بایت در **2** دیسک.
  (چرا؟)
- ✓ امکان خطاي **غیر قابل تصحیح**: **یک** بایت در **20000** دیسک.
  (چرا؟)

دسترسي به يک سکتور

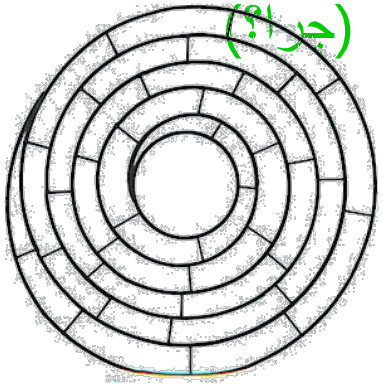
دسترسي به يک سکتور (seek) چگونه انجام ميشود؟

- ✓ آدرس يک سکتور بستگي به سرعت صحيح چرخش دارد! (چرا؟)
- ✓ ولي خود سرعت متغير است! (چرا؟)
- ✓ پس بايستي اطلاعات مربوط به آدرس نيزحتما خوانده شوند!
- ✓ و احتياج به روش سعي و خطا (trial and error) ميباشد، (چرا؟)
- ✓ که خود راندمان کار را پايين مي آورد.

12 bytes Synch.	4 bytes Sector ID	2048 **bytes** User data	4 bytes error detection	8 bytes null	276 bytes Error correction

نقاط ضعف یا قوت دیسک نوری

نقاط ضعف یا قوت دیسک نوري چیست؟



- زمان جستجو (seek) بسیار طولاني میباشد:
  - ✓ نیم ثانیه (یا بیشتر)

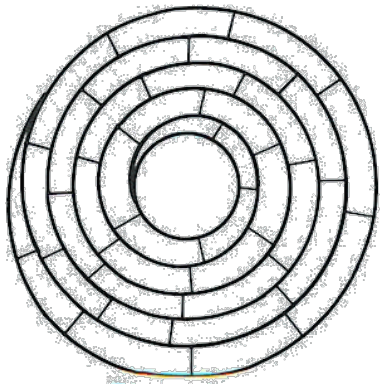
- **مثال:**
  - ✓ اگر زمان دسترسي به حافظه RAM **20 ثانیه** باشد،
  - ✓ زمان دسترسي به دیسک مغناطیسي برابربا **58 روز**،
  - ✓ و زمان دسترسي به CD-ROM برابربا **دو سال و نیم** خواهد بود.

نقاط ضعف یا قوت دیسک نوری

نقاط ضعف یا قوت دیسک نوري چیست؟

- **سرعت انتقال** داده (Data transmission rate) **پایین** است:
  - ✓ دیسک نوري: **150** KB/Sec
  - ✓ دیسک مغناطیسي: **3000** KB/Sec

(چرا؟)



- **ظرفیت** ذخیره سازي داده **بالا** میباشد:
  - ✓ حدود **600** MB – **700** MB

- خاصیت **Read Only** بودن آن **میتوانست** مفید باشد:
  - ✓ یک دستگاه مناسب بعنوان Publishing Medium

(چرا؟)

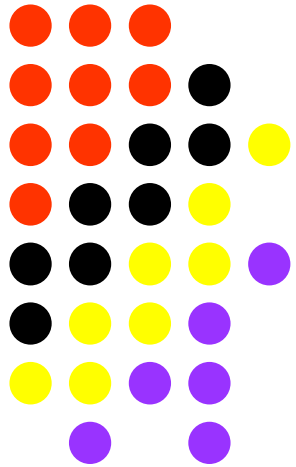
In the Name of God

Lecture 6

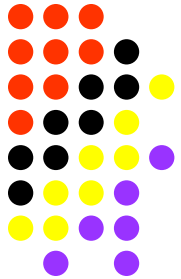
سیستم مدیریت I/O

I/O Management System

(Sections 3.8, 3.9, 3.10)



سیستم مدیریت I/O
I/O Management System



سیستم مدیریت I/O چیست ؟

مسیرI/O براي نوشتن داده روي ديسک چگونه است؟

چه استراتژي هايي براي مديريت بافرهاي I/O وجود دارد؟

ساختار سیستم مدیریت I/O در Unix چگونه است؟

انواع سیستمهاي I/O در Unix کدامند؟

سیستم مدیریت I/O

مسیر I/O براي نوشتن داده روي ديسک چگونه است؟

مثال:

- ✓ **یک برنامه C در نظر میگیریم که درخواست نوشتن یک بایت داده را در یک فایل مینماید.**
- ✓ **براي انجام اين درخواست چه مراحلي در سيستم طي ميشود؟**

مرحله (1): برنامه C:

- ✓ درخواست I/O : **write (textfile, ch, 1)**

مرحله (2): سیستم مدیریت فایلها (File Manager):

- ✓ به جدول **O**pened **F**ile **T**able مراجعه میکند.
- ✓ بافر I/O مربوط به سکتور مورد نظر را **آماده** میکند. (Load)
- ✓ بایت را در محل مناسب در بافر مینویسد.
- ✓ سپس I/0 Processor را صدا (invoke) میکند.

سیستم مدیریت I/O

مسیرI/O براي نوشتن داده روي ديسک چگونه است؟

مثال (ادامه...):

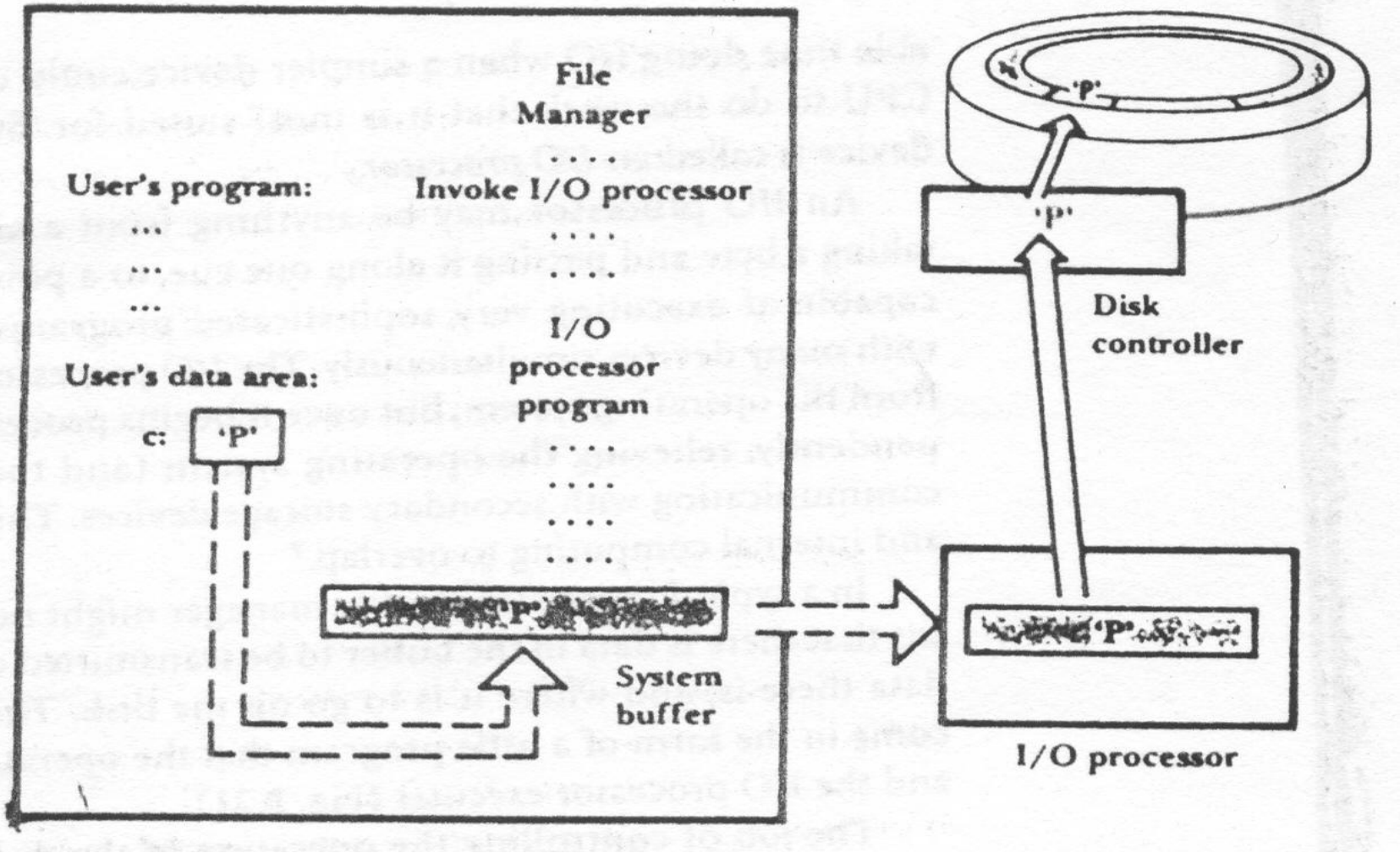
مرحله (3): پردازنده I/O (I/O Processor):

- ✓ پردازنده I/O بطورمستقل از پردازنده اصلي (CPU) عمل ميبكند. (چرا؟)
- ✓ بافر I/O را به **فرمت** مناسب ديسک تبديل و آماده تحويل ميكند.
- ✓ منتظر آمادگي **كنترلر** ديسک (Disk Controller) براي دريافت ميشود.
- ✓ سپس محتواي بافر را براي كنترلر ديسک **ارسال** ميكند.

مرحله (4): سيستم كنترل ديسک (Disk Controller):

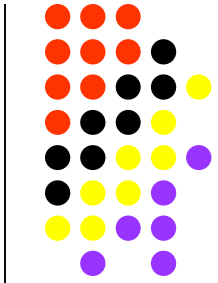
- ✓ دستور قرار گرفتن **هد** Read/Write روي **شيار** مربوطه را ميدهد.
- ✓ روی شيار مربوطه، در انتظار رسيدن هد به سكتور مورد نظر ميماند.
- ✓ سپس محتواي بافر را براي ديسک **ارسال** ميكند.

مسیرI/O براي یک بایت



(شکل 3.21 صفحه 90)

مديريت بافرهاي I/O



چه استراتژي هايي براي مديريت بافرهاي I/O وجود دارد؟

انواع بافرهاي I/O كدامند و مديريت آنها با كيست؟

1) **بافرهاي I/O سيستم (System I/O Buffer)**
   - ✓ مسوولیت مدیریت آن با سیستم است.

2) **بافرهاي I/O برنامه (Program I/O Buffer)**
   - ✓ مسوولیت مدیریت آنها با خود برنامه است.

مديريت بافرهاي I/O چه اهميتي دارد؟

- ✓ مديريت بافرهاي I/O در كارآيي (Performance) سيستم و برنامه ها نقش بسيارموثري دارد.
- ✓ براي پايين آوردن تعداد مراجعات به ديسک ها بايستي سيستم تعدادي بافر I/O رزرو نمايد (Multiple Buffering).

مديريت بافرهاي I/O

چه استراتژي هايي براي مديريت بافرهاي I/O وجود دارد؟

روشهاي Multiple Buffering كدامند؟

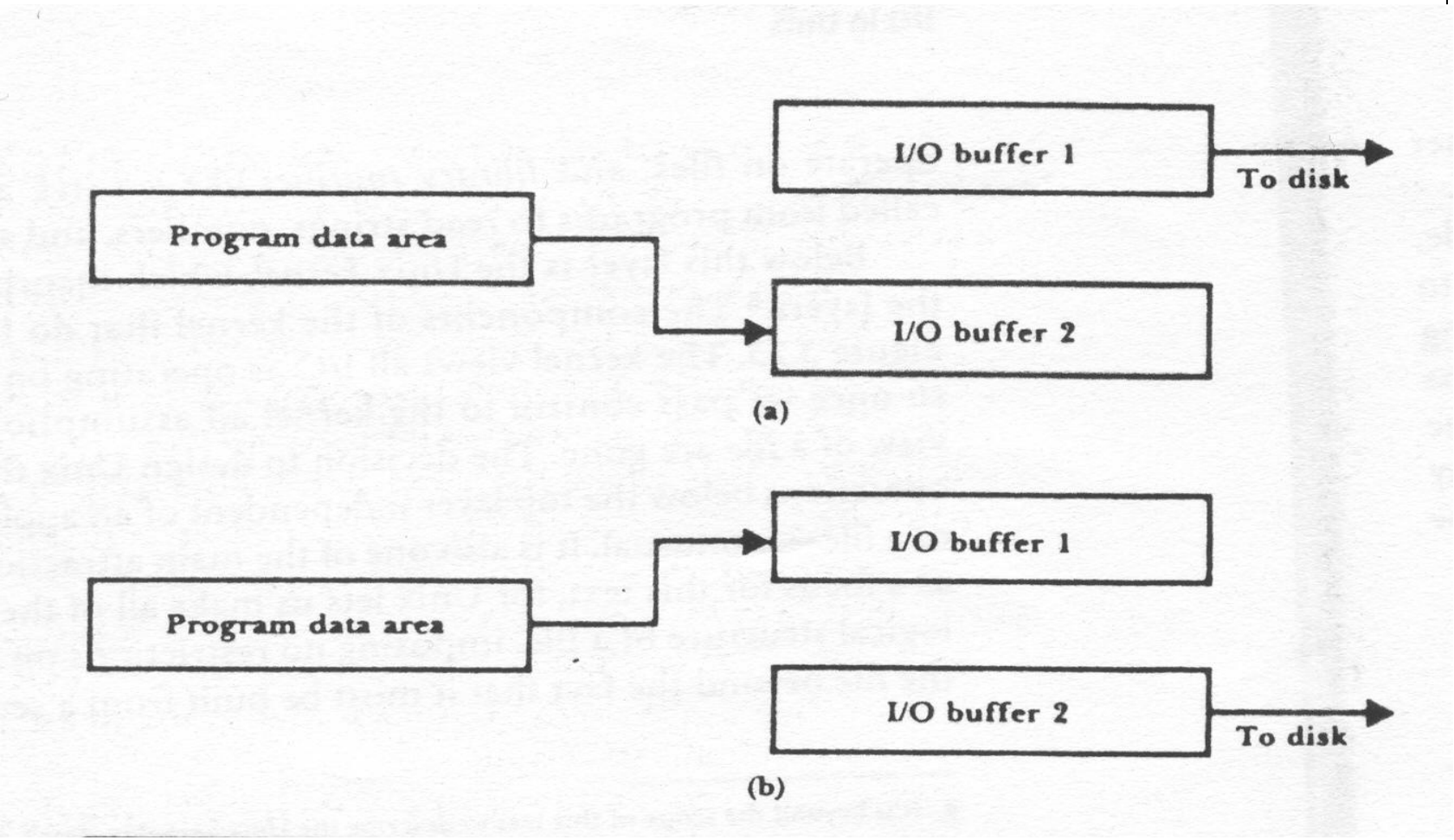
1) **روش Double Buffering:**

- ✓ حالتي است كه سيستم دو بافر I/O به يک برنامه (Process يا Job) اختصاص ميدهد.
- ✓ اين تعداد ميتواند بيشتر نيز تعيين شود. **(كجا؟)**

2) **روش Buffer Pooling:**

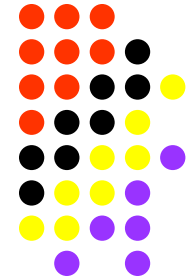
- ✓ سيستم **تعداد زيادي بافرI/O** رزرو مي كند و سكتور هاي مورد استفاده برنامه ها را حتي الامكان در RAM حفظ مي كند.
- ✓ در موقع احتياج به **آزاد كردن يكي از بافرها** از روش Least Recently Used استفاده مي شود.

Double Buffering روش



(شکل 3.22 صفحه 92)

روش Double Buffering



مثال:

- ✓ برنامه زیر را در نظر بگیرید.
- ✓ تعداد I/O را قبل و بعداز استفاده از روش Double Buffering محاسبه کنید.

```
While (1){
   infile>>ch;
   if ( file.fail() ) Break;
   outfile<<ch;
}
```

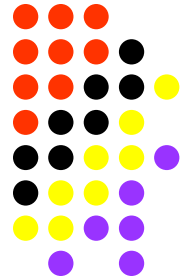
حالت اول: بدون روش Double Buffering:

براي كپي كردن فقط يک سكتور 512 بايتي بايستي:

- ✓ فايل ورودي (infile) تعداد 512 بار خوانده شود
- ✓ فايل خروجي (outfile) 512 بار خوانده و 512 بار نوشته شود.

(چرا؟)

روش Double Buffering



مثال:

- ✓ برنامه زیر را در نظر بگیرید.
- ✓ تعداد I/O را قبل و بعداز استفاده از روش Double Buffering محاسبه کنید.

```
While (1){
   infile>>ch;
   if ( file.fail() ) Break;
   outfile<<ch;
}
```

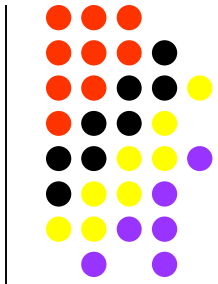
حالت دوم: با روش Double Buffering :

براي كپي كردن فقط يک سكتور 512 بايتي بايستي:

- ✓ فايل ورودي (infile) فقط يک بار خوانده شود
- ✓ فايل خروجي (outfile) فقط يک بار خوانده و فقط يک بار نوشته شود.

(چرا؟)

مديريت بافرهاي I/O



چه استراتژي هايي براي مديريت بافرهاي I/O وجود دارد؟

روشهاي Move Mode و Locate Mode كدامند؟

- ✓ **Move Mode** حالتي است كه در آن بافر I/O سيستم از بافر I/O برنامه **مجزا** ميباشد.
- ✓ **Locate Mode** حالتي است كه سيستم مستقيما از بافر I/O برنامه استفاده ميكند يا اينكه برنامه مستقيما به بافر I/O سيستم دسترسي دارد.

روشهاي Scatter I/O و Gather I/O كدامند؟

- ✓ **Scatter I/O** حالتي است كه سيستم بطور يكجا چندين بافر I/O را ا ز روي ديسک ميخواند.
- ✓ **Gather I/O** حالتي است كه سيستم چندين بافر I/O را يكجا روي ديسک مينويسد.

مدیریت I/O در Unix

ساختار سیستم مدیریت I/O در Unix چگونه است؟

جداولي كه براي مديريت فايل ها استفاده ميشوند كدامند؟

(a) *descriptor* table

File descriptor	File table entry
0 (keyboard)	
1 (screen)	
2 (error)	
3 (normal file)	
4 (normal file)	
5 (normal file)	

to *open file* table

(b) *open file* table

R/W mode	Number of processes using it	Offset of next access	ptr to write routine	...	*inode* table entry
write	1	100		...	

to *inode* table

write() routine for this type of file

1) جدول File Descriptor Table:
   - ✓ متعلق به هر برنامه (Process).
   - ✓ جدول فايلهاي يک برنامه.

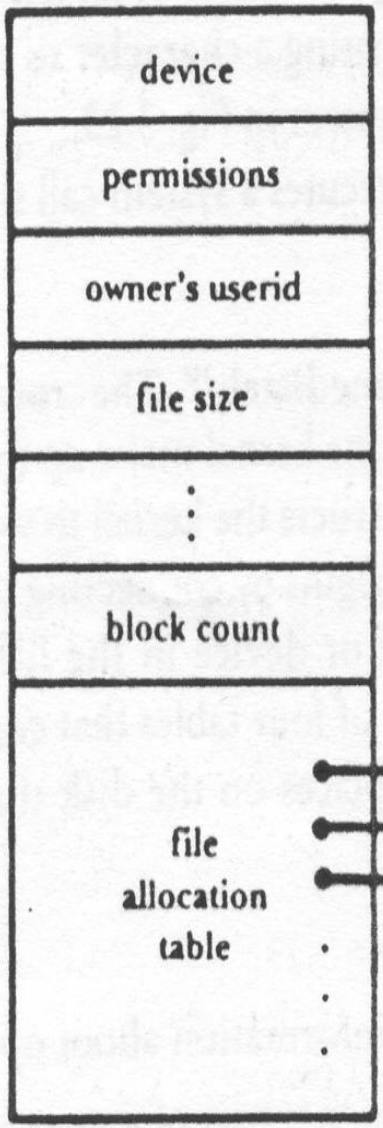
2) جدول Open File Table:
   - ✓ متعلق به kernel.
   - ✓ جدول فايلهاي باز شده در سطح سيستم.

(شکل 3.24 صفحه 90)

مدیریت I/O در Unix

ساختار سیستم مدیریت I/O در Unix چگونه است؟

جداولي که براي مدیریت فایل ها استفاده میشوند کدامند؟



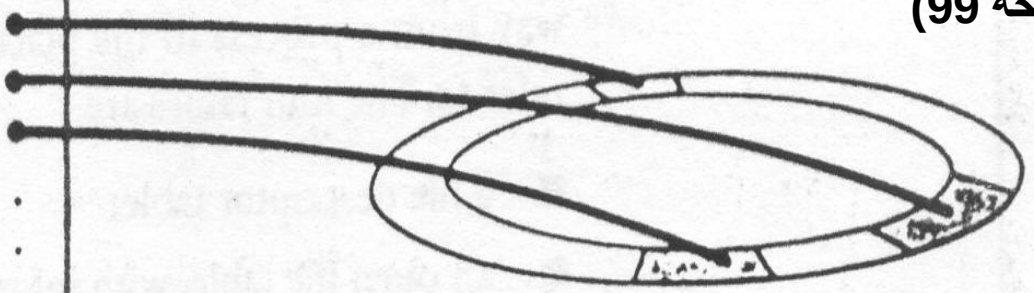
(3 جدول Tables of Index nodes:

- ✓ **متعلق به File System میباشد.**
- ✓ جدول **کل فایلها در یک File System.**
- ✓ (براي هر فایل یک i-node وجود دارد.)

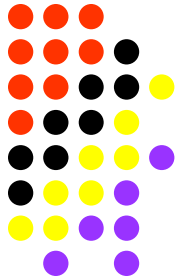
(4 جدول File Allocation Table:

- ✓ **متعلق به Kernel** و بخشي از Index Node میباشد.
- ✓ شامل آدرس extentهاي مربوط به هر فایل روي دیسک.

(شکل 3.25 صفحه 99)



مديريت I/O در Unix

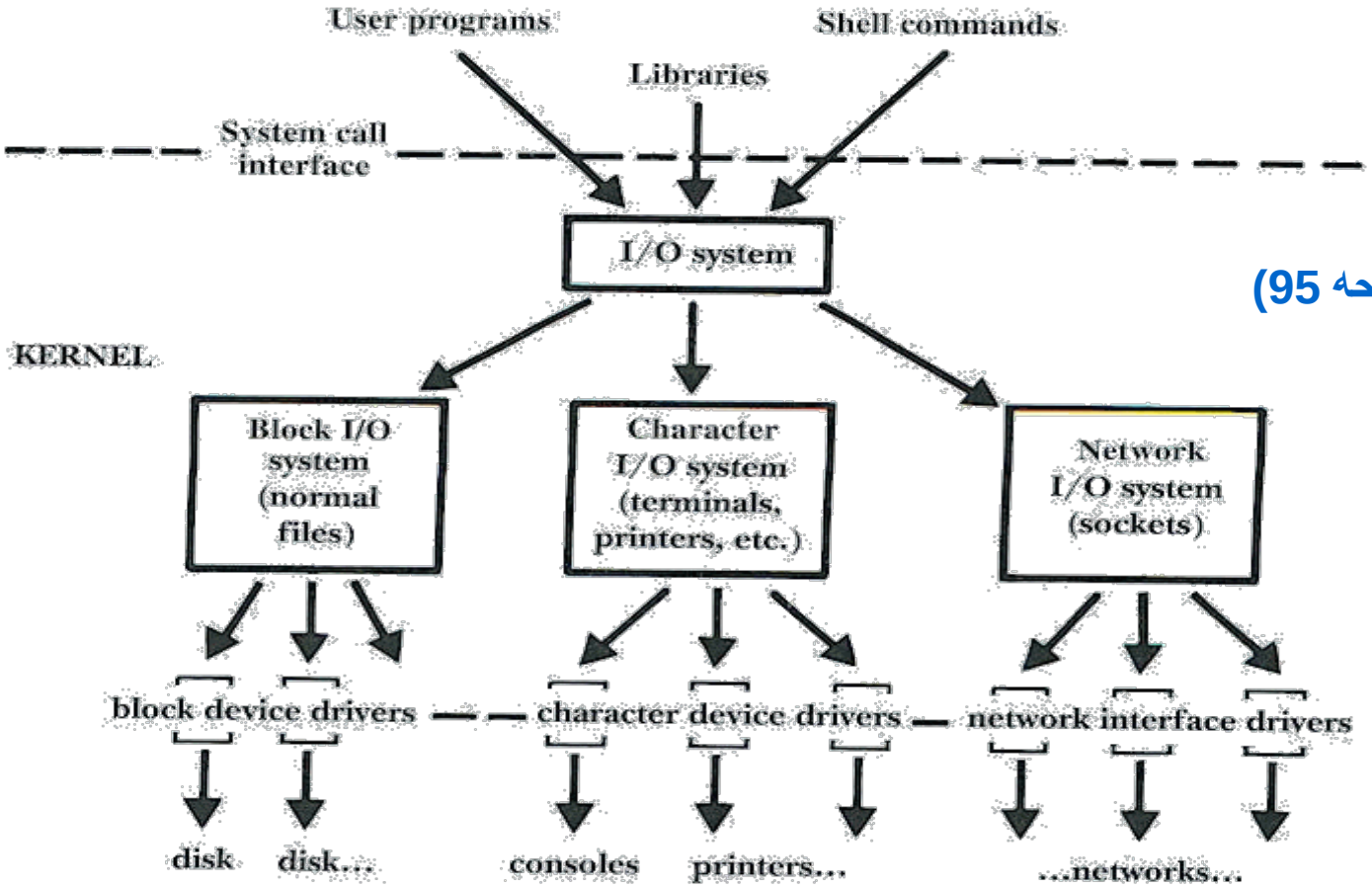


انواع پروتوكل هاي I/O در Unix كدامند؟

1) پروتوكل Character I/O براي:
   - ✓ ترمينال ها
   - ✓ پرينترها
   - ✓ يا Tapeها
2) پروتوكل Block I/O براي:
   - ✓ فايل هاي روي ديسک
   - ✓ يا Tapeها
3) پروتوكل Network I/O براي:
   - ✓ شبكه (تبادل Sockets)

- ❖ Tapeها با دو پروتوكل اول كار ميكنند! ولي با بعضي محدوديت هاي خاص خود.

مدیریت I/O در Unix



(شکل 3.23 صفحه 95)

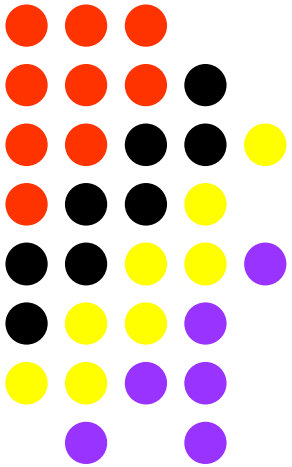
Device Driver چیست؟

✓ ارتباط با هر **device** فقط از طریق برنامه خاصي به نام **driver** مربوط به آن **device** انجام شود.

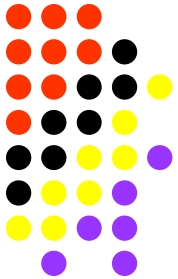
Lecture 7

Fundamental concepts to managing files of records

(sections 4.1, 5.1, 5.6)



مفاهیم اولیه در ساختار یک فایل



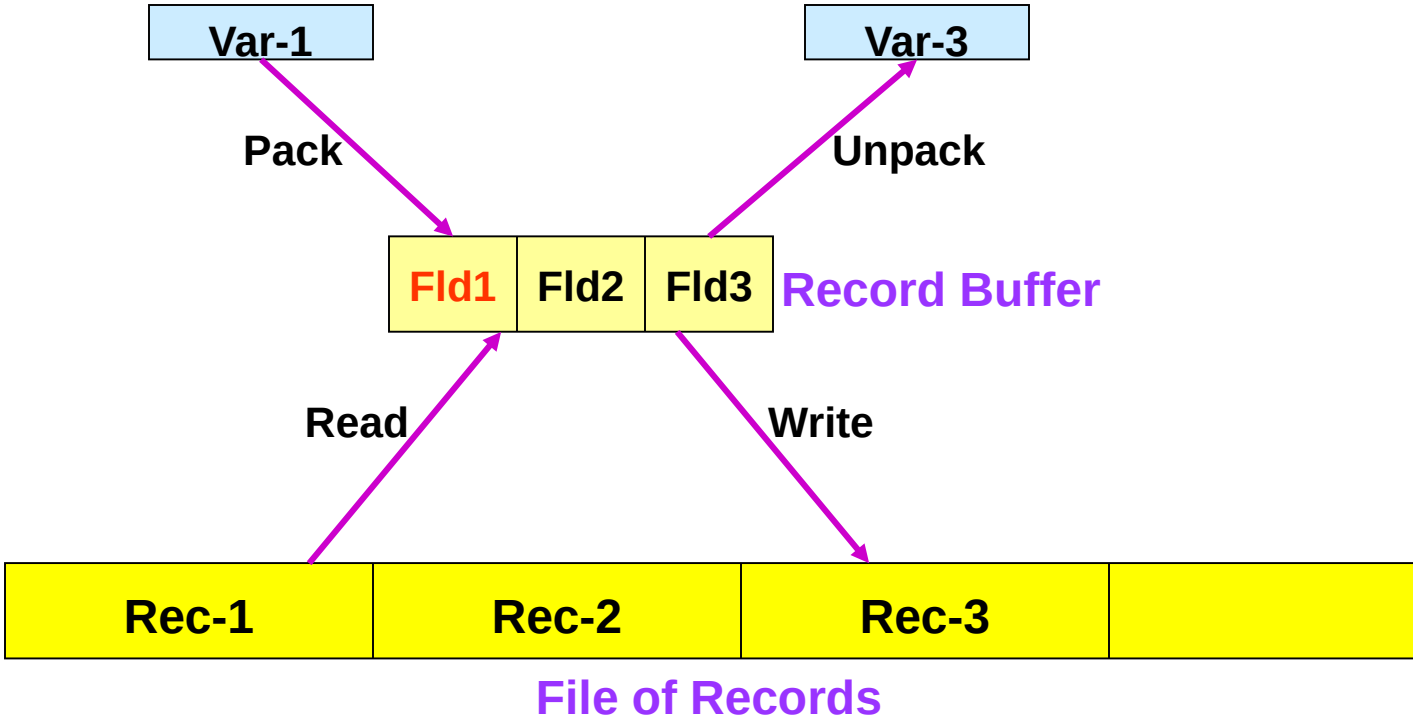
- ساختار یک فایل چگونه میباشد؟
- سازماندهی داده های یک فایل به چه صورت است؟
- تعاریف رکورد و فیلد چیست؟
- منظور از کلید اصلی یا ثانوی چیست؟
- انواع ساختار رکورد و فیلد چگونه است؟
- روشهای دسترسی به رکوردها کدامند؟
- همخواني بین سیستم ها به چه معنی میباشد؟

مفاهیم اولیه در ساختار یک فایل

ساختار یک فایل چگونه میباشد؟

سازماندهی داده های یک فایل به چه صورت است؟

تعاریف رکورد و فیلد چیست؟



مفاهیم اولیه در ساختار یک فایل

ساختار یک فایل چگونه میباشد؟

سازماندهی داده های یک فایل به چه صورت است؟

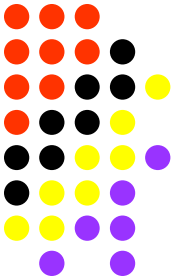
تعاریف رکورد و فیلد چیست؟

از دیدگاه کاربران:

- ✓ هر فایل مجموعه ای از بخشهای منطقی بنام **رکورد** (Record) میباشد. (منطقی؟)
- ✓ هر رکورد مجموعه ای از واحدهای مفهومی بنام **فیلد** (Field) میباشد. (مفهومی؟)
- ✓ فیلد هایی که براي متمایز نمودن یک رکورد مفید باشند **کلید** (Key) خوانده میشوند.
- ✓ کلیدي که یک رکورد را بطور یکتا متمایز مینماید، **کلید اصلي** (Primary) مینامیم.
- ✓ کلیدهایي که برای مواردي از جستجو مفید هستند، **کلید ثانوي** (Secondary) مینامیم.

Book nb	Author	Book Title	Comments

مفاهیم اولیه در ساختار یک فایل



سازماندهی داده های یک فایل به چه صورت است؟

انواع ساختار رکورد و فیلد چگونه است؟

انواع ساختار فیلد:

1) فیلدهاي با طول مشخص
2) فیلدهاي محتوي شاخص طولي
3) فیلدهاي محتوي كاراكتر پاياني (يا جدا كننده)
4) فیلدهاي مشخص شده بوسيله كلمه كليدي

انواع ساختار فيلد - مزايا و معايب

1) **فيلدهاي با طول مشخص (Fixed Length):**

- ✓ دسترسي به هر فيلد راحت و سريع مي باشد . (چرا؟)
- ✓ ولي مقداري از فضاي رزرو شده ممكن است بيهوده مصرف شود.
- ✓ فضاي اضافی با كاراكتر Space (يا صفر) پرميگردد.

Ames	Mary	911 High St.	Austin	TX78701
KammermeisteAlan		102 Main St.	Buda	TX78642

انواع ساختار فيلد - مزايا و معايب

2) **فيلدهاي محتوي شاخص طولي (Length indicator):**

- ✓ فضاي اضافي رزرو نشده و بيهوده به هدر نميرود.
- ✓ امكان پرش به فيلدهاي بعدي براحتي ميسر است.
- ✓ ولي لااقل يک بايت براي هر فيلد اضافه ميشود (با محدوديت طول فيلد: 255) (چرا؟)

```
04Ames04Mary12911 High St.06Austin02TX0578701
13Kammermeister04Alan12102 Main St.04Buda02TX0578642
```

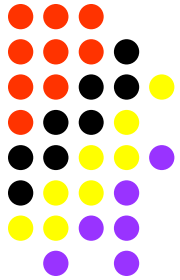
انواع ساختار فیلد - مزایا و معایب

3) **فیلدهاي محتوي کاراکتر پاياني (يا جدا کننده) (Separator) :**

- ✓ فضاي اضافی رزرو نشده و بيهوده به هدر نميرود.
- ✓ ولي يک بايت براي هر فيلد اضافه ميشود.
- ✓ **محدوديت** روي **طول فيلد** وجود **ندارد**.
- ✓ ولي کاراکتر جداکننده **نبايستي** در خود فيلد **استفاده** شود.
- ✓ در ضمن امکان **پرش سريع** به فيلدهاي بعدي **وجود ندارد**.
- ✓ **کاراکترهاي** هر فيلد بايستي **يک به يک** چک شوند.

```
Ames|Mary|911 High St.|Austin|TX|78701
Kammermeister|Alan|102 Main St.|Buda|TX|78642
```

انواع ساختار فيلد - مزايا و معايب

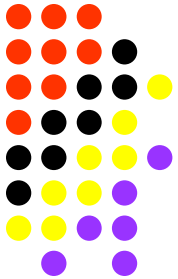


4) **فيلدهاي مشخص شده بوسيله کلمه کليدي:**

- ✓ هر فيلد بصورت Keyword=value مشخص ميشود. **(HTML style)**
- ✓ مانند حالت سوم نياز به **کاراکتر پاياني** (يا جدا کننده) ميباشد.
- ✓ **ترتيب** فيلد ها **اهميتی ندارد**.
- ✓ در صورت **عدم احتياج** به يک فيلد، مستقيما **فيلد بعدي** در رکورد ثبت ميگردد.
- ✓ چند کاراکتر اضافي (براي Keyword) به طول هرفيلد اضافه ميشود.

```
Last=Ames|First=Mary|Address=911 High St.|City=Austin|State=TX|
Zip=78701
```

مفاهیم اولیه در ساختار یک فایل



سازماندهی داده های یک فایل به چه صورت است؟

انواع ساختار رکورد و فیلد چگونه است؟

انواع ساختار رکورد:

1) رکوردهاي با **طول** مشخص.
2) رکوردهاي تعريف شده بر حسب **تعداد فيلد**.
3) رکوردهاي محتوي **شاخص طولي**.
4) رکوردهاي محتوي **کاراکتر پاياني** (يا جدا کننده).
5) رکوردهاي مشخص شده با کمک **ايندکس**

انواع ساختار رکورد

1) **رکوردهاي با طول مشخص:**

- ✓ معمولا با فيلدهاي **بطول** مشخص هستند.
- ✓ ولي **انواع** ديگر فيلدها نيز ميتوانند در چنين رکوردي تعريف شوند.
- ✓ به هر حال **فضاي** باقيمانده با کاراکتري مثل Space پر خواهد شد. (Padding)

```
Ames          Mary          911 High St.      Austin        TX78701
KammermeisteAlan            102 Main St.      Buda          TX78642
```

انواع ساختار رکورد

2) رکوردهاي تعريف شده بر حسب تعداد فيلد:

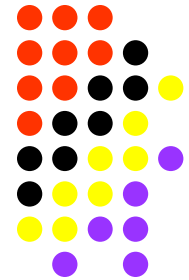
- ✓ اين رکوردها با طول متغيرهستند.
- ✓ معمولا با فيلدهاي بطول متغير تعريف ميشوند.

```
04Ames04Mary12911 High St.06Austin02TX0578701
13Kammermeister04Alan12102 Main St.04Buda02TX0578642
```

```
Ames|Mary|911 High St.|Austin|TX|78701
Kammermeister|Alan|102 Main St.|Buda|TX|78642
```

```
Last=Ames|First=Mary|Address=911 High St.|City=Austin|State=TX|
Zip=78701
```

انواع ساختار رکورد



3) رکوردهاي همراه با شاخص طول:

- ✓ در آغاز رکورد حداقل یک یا دو کاراکتربراي شاخص طول لازم است.
- ✓ معمولا با فیلدهاي بطول متغیر تعریف میشوند.

```
4504Ames04Mary12911 High St.06Austin02TX0578701
5213Kammermeister04Alan12102 Main St.04Buda02TX0578642
```

```
38Ames|Mary|911 High St.|Austin|TX|78701
45Kammermeister|Alan|102 Main St.|Buda|TX|78642
```

```
72Last=Ames|First=Mary|Address=911 High St.|City=Austin|State=TX|Zip=78701
```

انواع ساختار رکورد

4) رکوردهاي محتوي کاراکتر پاياني:

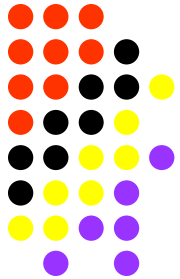
- ✓ کد Ascii پایان رکورد (End-of-Record)
- ✓ یا یک کاراکتر دیگر مثل '/' یا '\n' یا '\LF'
- ✓ روش بسیار متداول برای فایلهای متني ساده (Text files).

```
04Ames04Mary12911 High St.06Austin02TX0578701\n
13Kammermeister04Alan12102 Main St.04Buda02TX0578642\n
```

```
Ames|Mary|911 High St.|Austin|TX|78701\n
Kammermeister|Alan|102 Main St.|Buda|TX|78642\n
```

```
Last=Ames|First=Mary|Address=911 High St.|City=Austin|State=TX|
Zip=78701\n
```

انواع ساختار رکورد



5) **رکوردهاي تعريف شده به کمک ايندکس:**

- ✓ يک فايل جداگانه بنام **Index file** لازم ميباشد.
- ✓ که حاوي آدرس اولين بايت هر رکورد ميباشد.
- ✓ معمولا با **فيلدهاي بطول متغير** تعريف ميشوند.

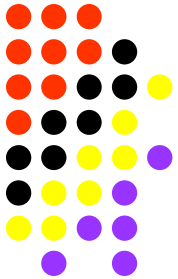
Index file:

```
00 39 85 ...
```

Data file:

```
Ames|Mary|911 High St.|Austin|TX|78701#Kammermeister|Alan|102 Main
St.|Buda|TX|78642#...
```

انواع ساختار رکورد - مزایا و معایب



مزایا و معایب انواع ساختار رکورد چگونه است؟

رکوردهاي با طول فیکس:

- ✓ از نظر دسترسي سریع به هر رکورد بهتر هستند. (چرا؟)
- ✓ ولي مقدار فضاي رزرو شده ممکن است بیهوده بماند.

رکوردهاي با طول متغیر:

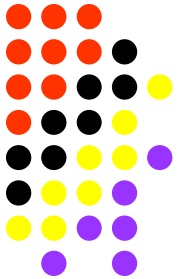
- ✓ از به هدر رفتن فضای اضافی پیشگیري میکنند.
- ✓ ولي دسترسي سریع به هر رکورد مشکل خواهد داشت . (چرا؟)
- ✓ استفاده از ایندکس امکان دسترسي سریع را میسازد.

انواع ساختار رکورد و فیلد

همخوانی بین انواع ساختار رکورد و فیلد چگونه است؟

	Fixed-Length Fields	Specified-Length Fields	Delimited Fields
Fixed-Length Records	**Best**	Not good	Not good
Specified-Length Records	Not good	**Best**	**OK**
Delimited Records	Not good	**OK**	**Best**

روشهای دسترسی به رکوردها
(Record Access)



انواع روشهای دسترسی به رکوردها کدامند؟

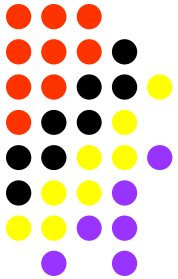
❖ **دسترسي مستقيم (Direct Access) :**

- ✓ فقط در مورد رکوردهاي **بطول فیکس** امکان پذیر میباشد.
- ✓ با استفاده از Relative Record Number انجام میشود.
- ✓ زمان دسترسی بستگي به تعداد **رکوردها** در فایل ندارد. (تابع O(1)). (چرا؟)

مثال:

- ✓ اگرطول رکورد **101** بایت باشد.
- ✓ براي دسترسي به رکورد سي ام (**RRN =30**)
- ✓ آدرس بایت رکورد **3030** میباشد. (چرا؟)

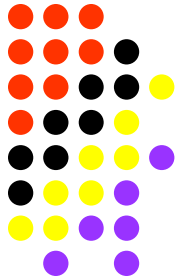
روشهای دسترسی به رکوردها
(Record Access)



انواع روشهای دسترسی به رکوردها کدامند؟

- **دسترسي سری (Sequential Access):**
  - ✓ رکوردها يكي بعد از ديگري خوانده ميشوند.
  - ✓ تنها روش امكان پذير در مورد ركوردهاي با طول متغير ميباشد. (ایندکس؟)
  - ✓ زمان دسترسی بستگي به تعداد رکوردها در فایل **دارد**. (تابع O(n)). (چرا؟)

روشهای دسترسی به رکوردها
(Record Access)

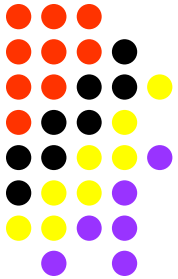


موارد استفاده روش دسترسی سری به رکوردها کدامند؟

- ✓ جستجو در فايلهاي متني (Text files)
- ✓ جستجو در فايلهاي كوچک (چرا؟)
- ✓ جستجو در فايلهاي روي باند مغناطيسی (*) (چرا؟)
- ✓ دسترسی به کليه رکوردهای يک فايل (*) (چرا؟)

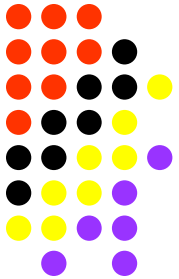
(*) با استفاده از بلوکهاي بزرگ I/O برای بهبود زمان دسترسی.

مفاهیم پیشرفته در ساختار یک فایل



- فراتر از ساختار رکوردها و فیلدها چیست؟
- سازماندهی داده های یک فایل چند رسانه ای چگونه است؟
- فایلهاي self-describing کدامند؟
- فایلهاي حاوي Object ها، تصاویر، صدا و غیره چگونه میباشند؟

مفاهیم پیشرفته در ساختار یک فایل



- همخواني بين سيستم ها به چه معنی میباشد؟
- اختلاف بين OS ها چگونه است؟
- اختلاف بين زبانهای برنامه نويسی کدامند؟
- اختلاف بين Processor ها چيست؟
- اختلاف در کد گذاري حروف و اعداد چگونه است؟

Real-World File Structures

by

Tom Davis
Asst. Professor, Computer Science

St. Edward's University
3001 South Congress Avenue
Austin, Texas 78704

http://www.stedwards.edu/

Metadata

- **Data *About* Data**
  - Usually in the **form of** a **file header**
  - **Example** in text
    - ✓ **Astronomy image** storage format
    - ✓ **HTML format** (name = value)
    - ✓ But look on page 177: coding style makes a BIG difference
  - **Parsing** this kind of **data**
    - ✓ **Read** field **name**; read field **value**
    - ✓ **Convert ASCII** value to **type required** for storage & use
    - ✓ Store converted value into right variable
  - **Why** use **this type** of header?

More Metadata

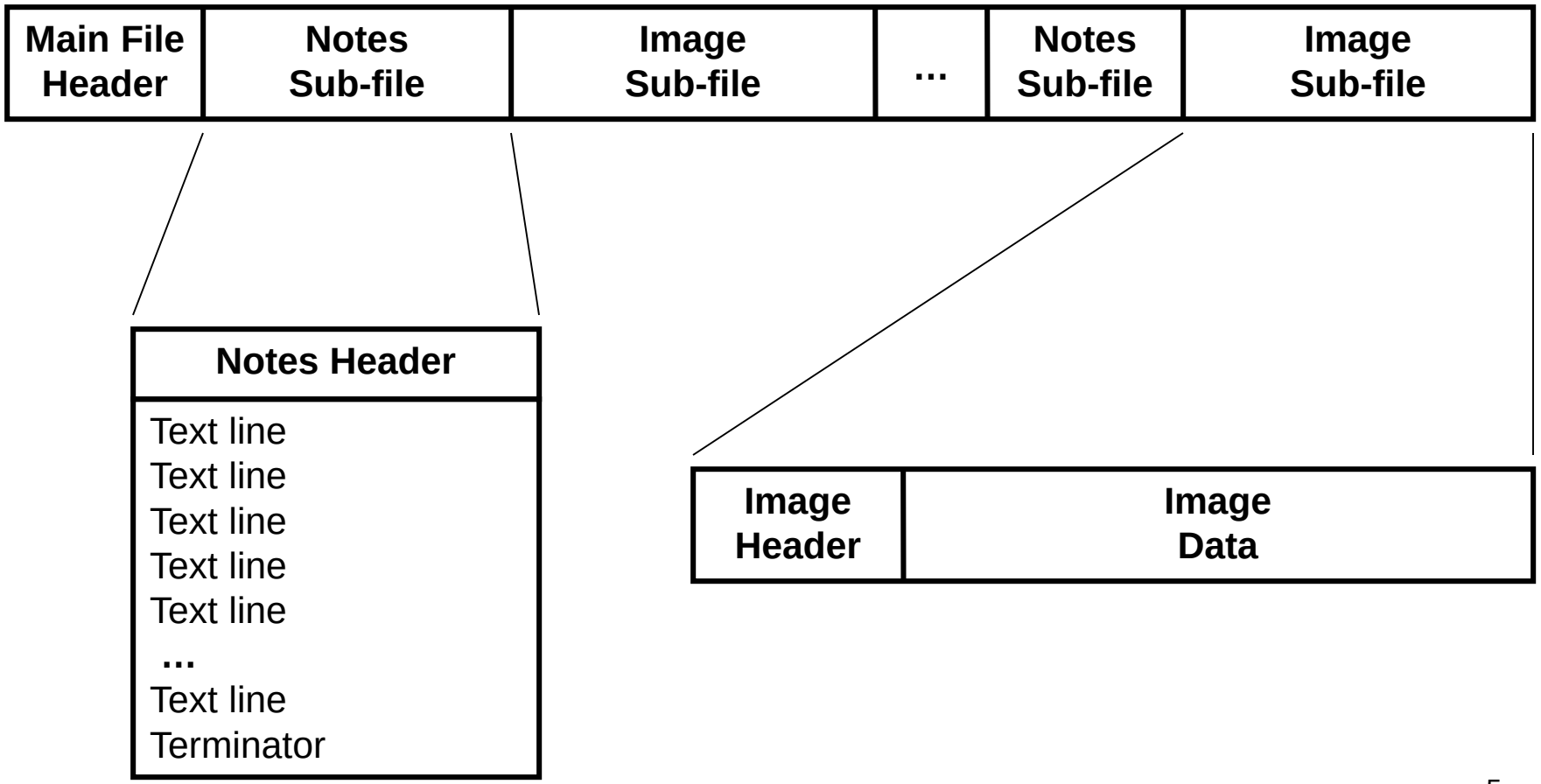
- **Graphics Storage Formats**
  - **Data**
    - ✓ **Color** values for **each pixel** in image
    - ✓ **Data compression** often used (GIF, JPG)
    - ✓ Different color "depth" possibilities
  - **Metadata**
    - ✓ **Height** & **width** of image
    - ✓ **Number** of **bits** per pixel (color depth)
    - ✓ If not true color (24 bits / pixel)
      - – **Color** look-up **table**
        - » Normally **256** entries
        - » Indexed by values stored for **each pixel** (normally 1 byte)
        - » Contains **R/G/B values** for color combination
      - – Often formatted to be **loaded** directly into **graphics RAM**

Mixing Kinds of Data in a File

- **Objective**
  - Store **different types** of data in the **same file**
  - Textbook example – **mix** of **astronomy data**
    - ✓ **Main file header** (HTML-style)
    - ✓ **Sub-files** of **notes** – lines of ASCII text
    - ✓ **Sub-files** of **image** data – in whatever format is needed
  - So our main file becomes **a file of sub-files**
    - ✓ **Each sub-file** (header, notes, or image) is really **a** "**record**" in the main file
    - ✓ These "**records**" are of varying **length** & **format**
    - ✓ How do we store the *actual* records in the sub-file "records"?
      - – Could use another level of specified-length record software
      - – Better – do what makes sense in the situation(s)

Our Main File

- **Organization**



More on Our Mixed-Data File

- **Access**
  - **Can** we just **read** it **sequentially**?
    - ✓ **Why** or why not?
    - ✓ What if we wanted to **skip** a notes sub-file?
    - ✓ What if some image didn't even have a notes sub-file?
  - **Can** we **access** it **directly**?
    - ✓ **What** would the **header** have **to include** to allow that?
      - – **An index** of the "records" in the file
      - – We call the entries in that index "**tags**"
    - ✓ Each tag in the tag list has:
      - – Type of sub-file referred to
        - » Special-case type: end of file
      - – RBA of sub-file in main file
      - – Length of sub-file (not necessary, but helpful)
      - – Key information, if any, for the sub-file

Even More on Our Mixed-Data File

- **Access, continued**
  - So **how** can we **access** the **mega-file** now?
    - ✓ **Read** and process the **header**
      - **Get information** about the whole main file
      - **Build** in-memory **table of tags** (keys + locations) for sub-files
    - ✓ **Sequential** access
      - Same as before
      - May be able to program in some speed-ups from tag table
    - ✓ **Direct** access
      - Locate sub-file in tag table
      - Go right to it

Extensibility

- **Look at Our Main File Format Again**
  - **Main header** tells us **things about** the **sub-files**:
    - ✓ What **kinds** of files they are
    - ✓ **Where** to find them
  - Sub-files themselves
    - ✓ To the main-file processor, they are just random bytes
    - ✓ To each sub-file processor, they are **meaningful information**
- **What If We Need a New Type of Sub-File?**
  - **Define** a **new** type of main **header entry**
  - **Extend** main header **processor** to understand that entry
  - Write (or borrow or buy) **code** to **handle** new sub-file
- **Cardinal Rule:**
  - Everything changes – file types, data types, …

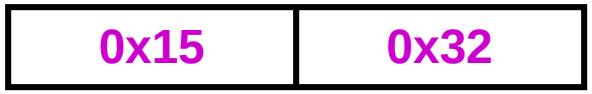
File Portability

Factors Affecting Portability - 1

- **Operating System Differences**
  - Example – text lines
    - ✓ End with **line-feed** character
    - ✓ End with **carriage-return** and line-feed
    - ✓ Prefixed by a **count** of **characters** in the line
- **Natural Language Differences**
  - Example – **character coding**
    - ✓ Single-byte coding – **ASCII**, **EBCDIC**
    - ✓ Double-byte coding – **Unicode**
- **Programming Language Differences**
  - **Pascal** can't directly process varying-length records
  - **Different C++ compilers** use different byte lengths for the standard data types

Factors Affecting Portability - 2

- **Computer Architecture Differences**
  - Byte order in 16-bit and 32-bit integer values
    - ✓ "Big-endian" – leftmost byte is most significant
    - ✓ "Little-endian" – rightmost byte is most significant



Big-endian
interpretation:
0x1532

Little-endian
interpretation:
0x3215

Don't ask.

  - Storage of data in memory
    - ✓ Most architectures require values that are **N** bytes long to start at a byte whose address is divisible by **N**

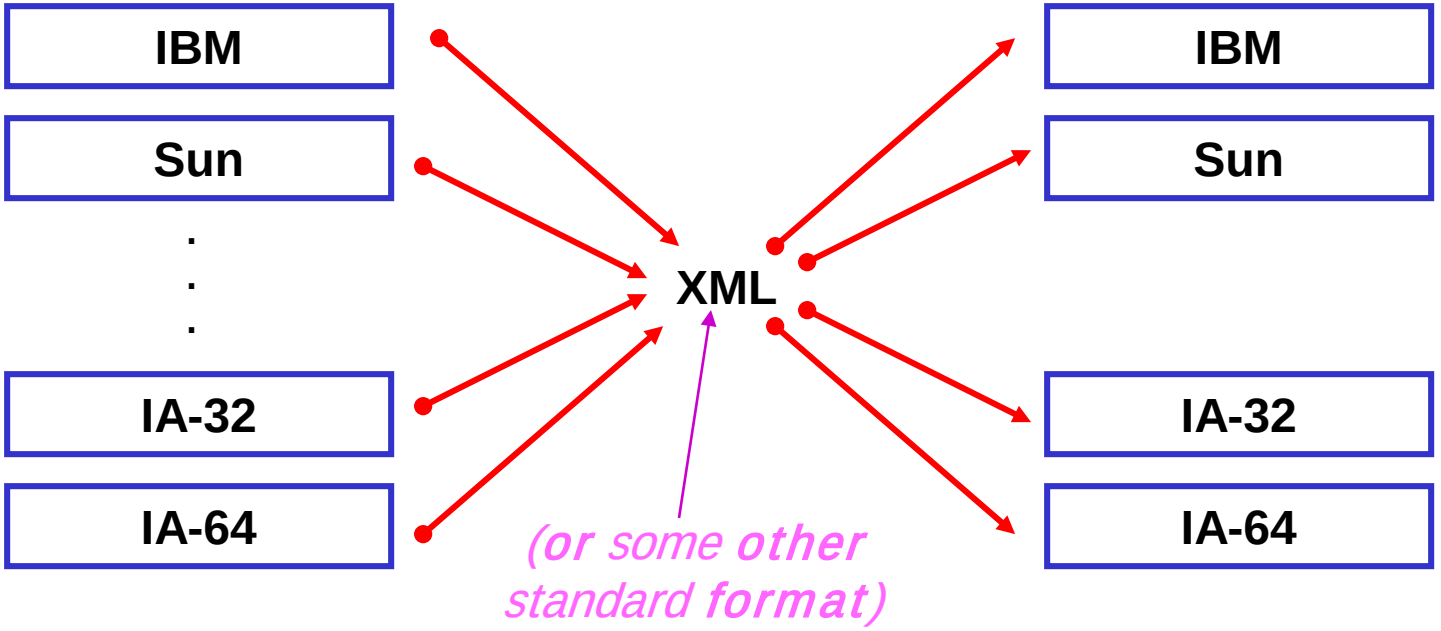
How to Port Files

- **Define Your Format C*A*R*E*F*U*L*L*Y**
  - Once a file **format** is **defined**, **never change** it
    - ✓ If you need a **new** file **format**, add it so as **not** to **invalidate** the **existing formats**
    - ✓ If you **need to change** a format, **add a new one** instead, and let programs that need the new version use it
  - **Decide** on a **standard format** for data elements
    - ✓ Text lines
      - – **ASCII** , **EBCDIC**, or **Unicode**?
      - – Which character(s) to end lines?
    - ✓ Binary
      - – **Tightly packed** or **multiple-of-N addressing**?
      - – Which "**endian**"?
  - You can always write code to **convert** to & from the **standard format** on a **new language**, **computer**, etc.

The Conversion Problem

- **Only a Few Environments – do it directly:**

- **Many Env'ts. – need an intermediate form:**



Beyond Record Structures

Dr. Robert J. Hammell
Assistant Professor

Towson University
Computer and Information Sciences Department
8000 York Road - Suite 406
Towson, MD 21252

http://triton.towson.edu/~rhammell/

Objectives

- Examine file structures in terms of
  - Abstract data models
  - Metadata
  - Object-oriented file access
  - Extensibility
- Examine portability and standardization

Beyond Record Structures

- Abstract Data Models for file access
  - Computers can process sound, images, documents
    - "Information" not data stored as fields & records
    - Envision the data as objects
      - Sound objects; image objects; document objects
  - Abstract data model
    - Application-oriented view of data
    - Not a medium-oriented view
    - Describe organization & access from the application's point of view

Headers & self-describing fields

- Want to keep user from having to know about objects
  - One way is to put information in the file
  - Allows file-access software to understand objects
- Put more information in the header
  - Makes the file self-describing
  - Information such as:
    - Name for each field
    - Width of each field
    - Number of fields per record
  - Can write program to read and print
    - Regardless of number of fields per record
    - With any combination of fixed-length fields

- **Trade-off**
  - Programs must be more sophisticated
  - Need flexibility to interpret the self-descriptions
- **Example:**
  - Class `FixedFieldBuffer`
  - Extend header to include more information
    - Requires a variable-sized header
  - Objects can be initialized from the header

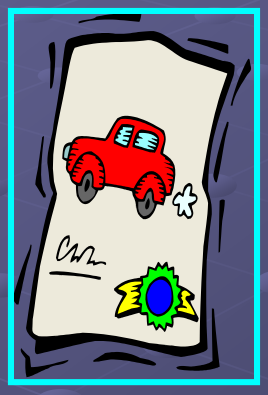
- **Metadata**
  - Data that describes data
  - Can be stored in the header
  - A standard format may be defined
    - If the usage of a type of data is common
    - Example:
      - Digital representation of pictures by astronomers
      - Use FITS (Flexible Image Transport System)
      - FITS header is collection of 2800-byte blocks
        - Made up of 80-byte ASCII records
        - Each record contains one piece of metadata
      - Metadata is ASCII; “real” data is binary
    - Good example of abstract data model
      - Data meaningless without information in header

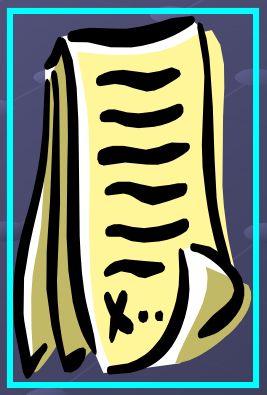
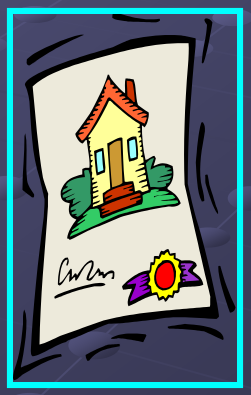
- **Color raster images**
  - Rectangular array of colored dots (pixels)
  - Lots of types of metadata
    - Dimensions
    - Number of bits per pixel
      - 1-bit: two colors; 2-bits: four colors; 8-bits: 256 colors
    - Color lookup table
      - To assign color to each pixel value
  - Methods for image ADT
    - Display an image in a window
    - Associate an image with a color lookup table
    - Overlay images to produce a composite image
    - Display images in succession (animation)

- **Mixing object types in one file**
  - **Keywords**
    - Keyword = value format
    - Is a small percentage in an image file
  - **Tags**
    - May want a couple of images plus a document
    - Also include the usual metadata
    - Now have mixture of very different objects
    - **Use keyword idea to solve, but:**
      - Let each record be big enough to hold the entire object
      - Put the keywords in an index table (offset & length)
    - **Tag** used to describe this type of file structure

Header	Notes	Image	Header	Notes	Image





- **Tag structures are common**
  - **TIFF:** Tagged Image File Format
  - **HDF:** Hierarchical Data format
    - Lots of different types of scientific data
  - **SGML:** Standard General Markup Language
    - Language for describing document structure
    - Defines tags used to make up the structure

- **Accessing files with mixture of data objects**
  - **Mix of types frees us of all records being the same**
  - **There is a price**
    - **How to search for a particular type object**
    - **Where exactly to store an object and put its tag**
    - **What is correct method for storing/retrieving an object**
  - **First two questions**
    - **Deal with accessing table of tags and pointers**
    - **Deal with in Chapter 6**
  - **Last question**
    - **Talk about briefly now**

- **Representation-independent file access**
  - **Abstract data model view**
    - Application-oriented view of an object
    - Ignores the physical file storage format
  - **Provides software two things:**
    - Lets application modules do the main processing job
      - Require separate modules to do translation to and from the physical format
    - At some level, different objects have same abstract data model
      - In-memory representations same
      - File formats may be different

Extensibility

- An advantage of tags:
  - Do not have to know ahead of time what all the objects will look like
    - Translation routines choose correct access methods
    - Easy to extend allowable types in the future
      - Just build new translations as get new objects
    - Application program stays the same

Portability and Standardization

- Want to be able to share files
  - Must be accessible on different computers
  - Must be compatible with different programs that will access them
  - Several factors affect portability
    - Operating systems
    - Languages
    - Machine architectures

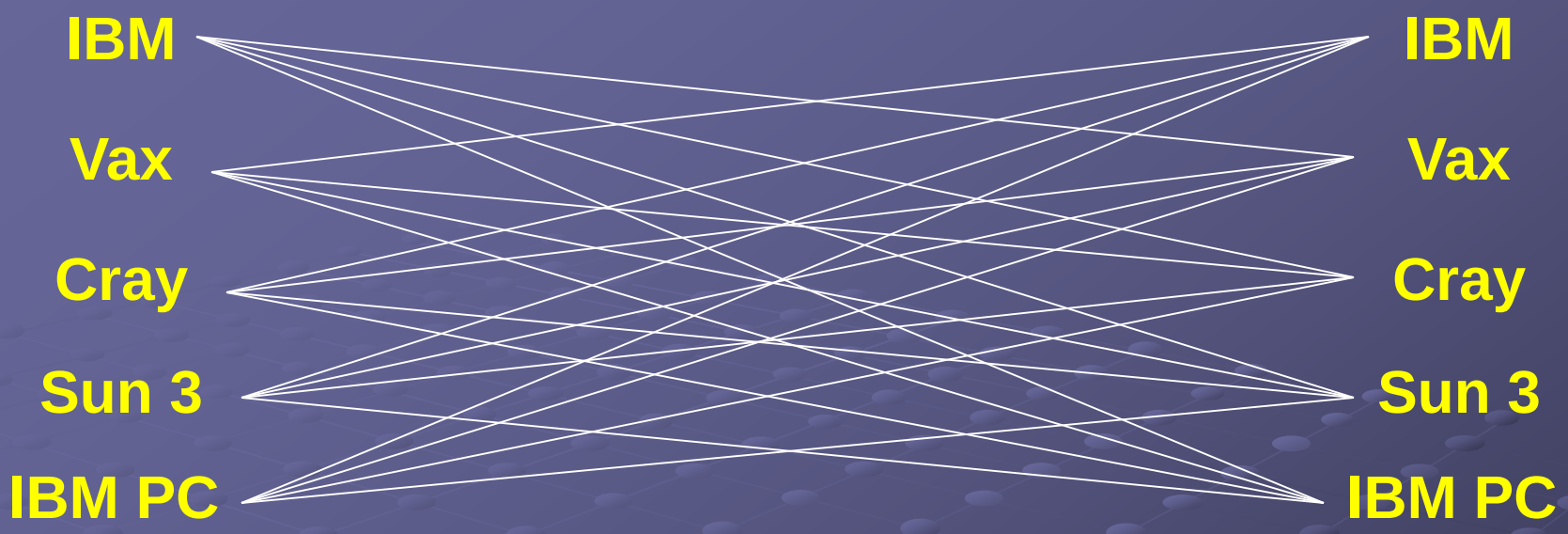
- **Differences among operating systems**
  - **In Chapter 2:**
    - **Saw DOS adds extra line-feed character when it sees CR**
    - **Not the case on most other file systems**
  - **Ultimate physical format of the same logical file can vary depending on the OS**
- **Differences among languages**
  - **Talked about C++ versus Pascal**
    - **C++ can have header and data records of different sizes**
    - **Pascal cannot**
  - **Physical layout of files may be constrained by the way languages allow file structure definitions**

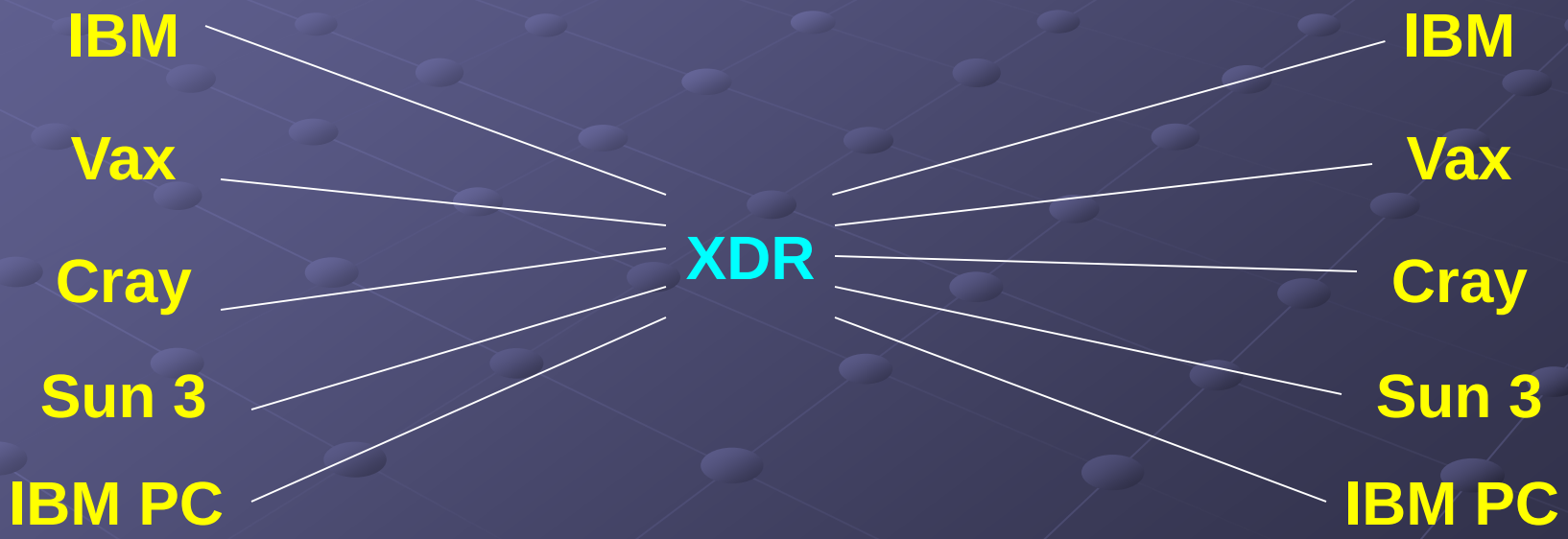
- **Differences in machine architectures**
  - **Saw problem of "Endean-ness"**
    - **Multi-byte integers:**
      - **Store high-order byte first or low-order byte first?**
  - **Word size may affect file layout**
    - **For a `struct` item, may allocate:**
      - **8-bytes (64-bit word)**
      - **4-bytes (32-bit word)**
      - **3-bytes (24-bit word)**
  - **Different encodings for text**
    - **ASCII**
    - **EBCDIC**
    - **Maybe other problems with international languages**

- **Achieving portability**
  - **Must determine how to deal with differences among languages, OSs, and hardware**
    - **It is not a trivial matter**
    - **Text offers some guidelines**
  - **Agree on standard physical record format**
    - **FITS is a good example**
      - **Specifies physical format, keywords, order of keywords, bit pattern for binary numbers**
    - **Once get standard, stay with it**
      - **Make the standard extensible**
      - **Make it simple enough for wide range of machines, languages, and OSs**

- Agree on a standard binary encoding
  - ASCII vs EBCDIC for text
  - Binary numbers have more options
    - IEEE standard
      - Specifies format for 32, 64, & 128-bit floating point
      - Specifies format for 8, 16, &32-bit integers
      - Most computers follow
    - XDR
      - External Data Representation
      - Specifies IEEE formats
      - Also provides routines to convert to/from XDR format and host machine format

- **Number and text conversion**
  - **May not want conversions all the time**
    - **Waste time on every read/write**
    - **May lose some accuracy**
  - **But may need conversion for different platforms**
    - **Can write routines to convert among all encodings**
      - ***n* encodings requires *n*(*n*-1) translators!**
    - **Better to use a standard intermediate format**
      - **Such as XDR**
      - **Less translators, but 2 translations between each platform**


IBM
Vax
Cray
Sun 3
IBM PC
IBM
Vax
Cray
Sun 3
IBM PC


IBM
Vax
Cray
Sun 3
IBM PC
XDR
IBM
Vax
Cray
Sun 3
IBM PC

- **File structure conversion**
  - **Suppose have X-ray images; want to:**
    - Look at and zoom in and out
    - Animate the images to see changes
    - Annotate images and store in an archive
  - **Complex objects & representations usually tied to specific applications**
    - May require 3 different formats
  - **Different solutions to the problem**
    - Require user to supply compatible format
      - User must convert
    - Process images of only a certain standard format
      - FITS approach
    - Include translate routines for several formats
      - Burden placed on software developer

- **File system differences**
  - **Are differences in physical format among file systems**
  - **Example:**
    - **Unix systems write tape files in 512-byte blocks**
    - **Non-Unix systems use different block sizes**
  - **This problem may need to be solved when transferring files between systems**

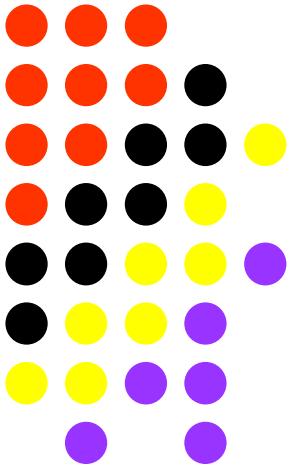
- **Unix and portability**
  - **Unix provides a utility called dd**
    - Intended for copying tape data
    - Can be used for converting data from any physical source
  - **Options include:**
    - Convert from one block size to another
    - Convert fixed-length records to var-length, and vice-versa
    - Convert ASCII to EBCDIC and vice versa
    - Convert all characters to uppercase (or lowercase)
    - Swap every pair of bytes
  - **Unix alone goes a long way toward file transfer**
    - Same OS, file system, device view, file org on any HW
    - Many platforms have a version of Unix
    - Files not perfectly portable, but Unix availability helps

Lecture 8

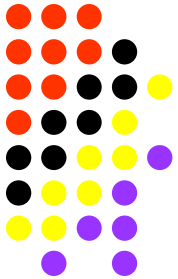
بازيابي فضاي ركوردها در يک فايل

(Reclaiming Record space in files)

(Section 6.2)

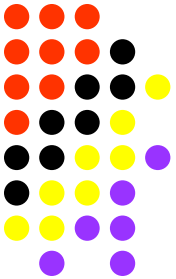


بازيابي فضاي ركوردها در يک فايل



- ايجاد يک ركورد در فايل چگونه انجام ميشود؟
- چگونه يک ركورد از فايل حذف ميگردد؟
- فضاي ركورد حذف شده چگونه بازيابي ميشود؟
- چه استراتژيهايي براي بازيابي فضاي فايل وجود دارد؟
- انواع ناپيوستگي (fragmentation) در داخل يک فايل كدامند؟

بازیابی فضای رکوردها در یک فایل
(Reclaiming Record space in a file)

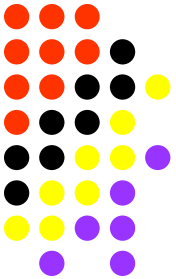


ایجاد یک رکورد در فایل چگونه انجام میشود؟

چگونه یک رکورد از فایل حذف میگردد؟

- ✓ **توابع** اوّلیه فایل سیستم (open, write, read و seek) به ما اجازه **ایجاد فایل**، **ایجاد رکورد** یا **تغییر محتواي** آن را مي دهند. (فیزیکي؟)
- ✓ **ولي براي حذف رکوردها** (delete) **در یک فایل تابعي نداریم!** (فیزیکي؟)
- ✓ **نمیتوانیم** قسمتي از **فضاي** رزرو شده یک فایل را **به سیستم برگردانیم!**

بازيابی فضای رکوردها در يک فايل



چگونه يک رکورد از فايل حذف ميگردد؟

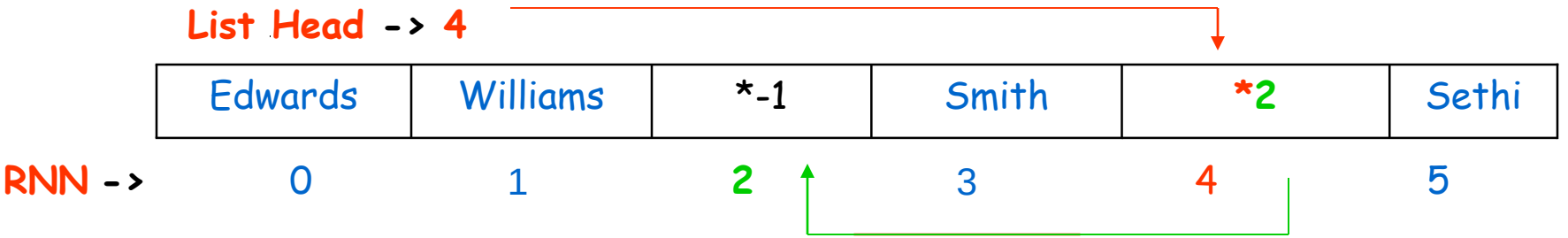
فضاي رکورد حذف شده چگونه بازيابي ميشود؟

- ✓ مسؤليت **حذف رکورد** در فايل و **استفاده مجدد** از فضاي خالي شده **برعهده کاربر** ميباشد. (**user program**)
- ✓ براي **حذف** رکورد بطور **منطقي** (**Logical**) ميتوان از روش علامت گذاري (**Marking**) استفاده نمود. مثلا در کاراکتر اول رکورد **علامت** "*****" قرار داد.
- ✓ **استفاده مجدد** از فضاي رکورد هاي علامت گذاري شده **برعهده خود کاربر** خواهد بود.
- ✓ اگر تعداد رکوردهاي **حذف** شده **زياد** باشد بايستي برنامه مخصوص ديگري نيز عمل **بازسازي** فضاي فايل را برعهده بگيرد. (**Storage Compaction**)

بازیابی فضای رکوردها در یک فایل

یک روش بازیابي رکوردهاي با طول ثابت چیست؟

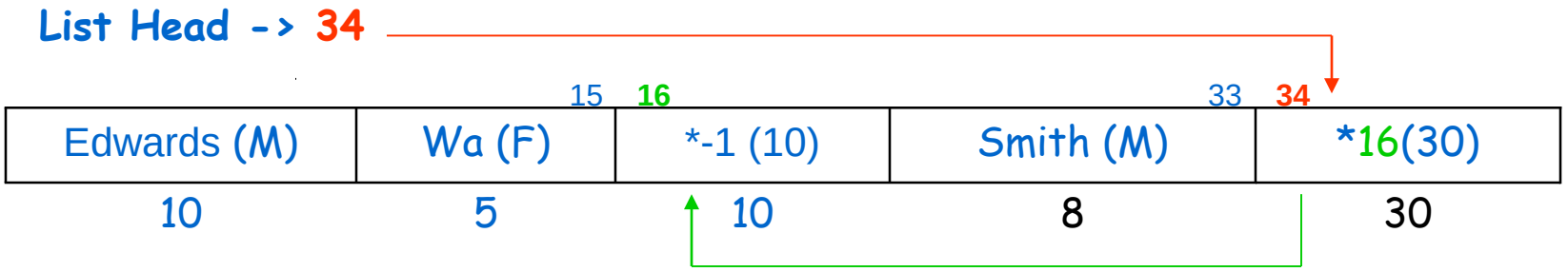
- ✓ روش تشکیل یک لیست از رکورد هاي حذف شده (Avail list).
- ✓ فضاهاي آزاد شده با یک Linked list به یکدیگر مرتبط میگردد.
- ✓ در آغاز فایل یک رکورد به نام Header Record لازم میباشد.
- ✓ از شماره RRN رکوردها استفاده میشود.
- ✓ انتهاي لیست با شماره '-1' مشخص میگردد.



بازیابی فضای رکوردها در یک فایل

- ✓ یک روش بازیابي رکوردهاي با طول متغیر چیست؟

- ✓ روش تشکیل یک لیست از رکورد هاي حذف شده (Avail list).
- ✓ ولي با در نظر گرفتن طول متغیرفضاهاي آزاد شده.
- ✓ از شماره RRN رکوردها نمیتوان استفاده نمود.
- ✓ بایستي از آدرس بایتي رکوردها (Byte offset) استفاده کرد.
- ✓ در ضمن در هر رکورد آزاد شده بایستي طول آن به بعنوان یک فیلد حفظ شود.



بازيابی فضای رکوردها در يک فايل

چه استراتژيهايي براي بازيابي فضاهاي آزاد (Avail list) وجود دارد؟
(Placement Strategies)

1) روش First-fit:

- ✓ هنگام ثبت يک رکورد جديد، اوّلين فضايي که طول آن کافي باشد انتخاب مي شود.
- ✓ در اينصورت، نيازي به مرتب سازي Avail list نميباشد. (چرا؟)

2) روش Best-fit:

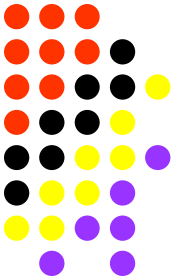
- ✓ هنگام ثبت يک رکورد جديد، کوچکترين فضايي که طول آن کافي باشد انتخاب مي شود.
- ✓ در اينصورت بايستي Avail list به طور صعودي مرتب شده باشد. (چرا؟)

3) روش Worst-fit:

- ✓ هنگام ثبت يک رکورد جديد، بزرگترين فضاي آزاد موجود انتخاب مي شود.
- ✓ در اينصورت بايستي Avail list به طور نزولي مرتب شده باشد. (چرا؟)
- ✓ فضاي باقيمانده احتمالي نيز مجددا به Avail list اضافه مي شود. (چرا؟)

(مزايا و معايب؟)

بازیابی فضای رکوردها در یک فایل



انواع ناپیوستگی (fragmentation) در داخل یک فایل کدامند؟

- ✓ فضاهای کوچک موجود در **Avail list** که قابل استفاده مجدد **نمیباشد**. (External)
- ✓ فضاهای به هدر رفته در داخل خود رکوردها. (Internal)

چه روشهایی برای کم کردن ناپیوستگیهای External وجود دارد؟

- ✓ دو فضای آزاد شده **مجاور**هم را میتوان به هم **پیوند** زد. (Coalescing the holes)
- ✓ استفاده از روشهای **Placement** متناسب با شرایط هر فایل.

در چه شرایطی روش worst-fit می تواند بهتراز Best-fit باشد؟

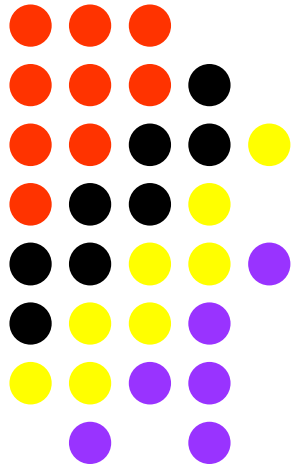
In the Name of God

Lecture 9

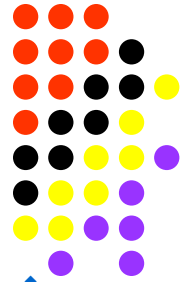
بازيابي سريع داده ها – مرتب سازي

Finding data quickly - Sorting

(Sections 6.3, 6.4 , 7.1, 7.2)



بازیابی سریع داده ها – مرتب سازی
(Finding data quickly – Sorting)



- روشهاي بازيابي سريع داده ها چگونه ميباشند؟
- يادآوري جستجوي دودويي (Binary Searching)؟
- مقايسه با جست وجوي سري(sequential)؟
- محدوديت ها يا معايب جست و جوي دودويي كدامند؟
- مرتب سازي كليدها (key sorting) چگونه است؟
- روش Indexing چيست؟
- مزاياي Indexing كدامند؟

بازیابی سریع داده ها

روشهاي بازيابي سريع داده ها چگونه ميباشند؟

يادآوري جستجوي دودويي (Binary Searching)؟

مثال:

- ✓ يک فايل با رکورد هاي به طول ثابت را در نظر ميگيريم.
- ✓ فرض کنيم که در جست و جوي رکوردي با مقدار کليدي مشخصي ميباشيم.

حالت اول: اگر فايل مرتب نشده باشد:

- ✓ بايستي رکورد هاي آنرا **يک به يک خوانده** و کليد آنها را با مقدار مورد نظر **مقايسه** کنيم.
- ✓ اين کار ممکن است به **خواندن کليه رکورد ها** منتهي شود. **(چرا؟)**

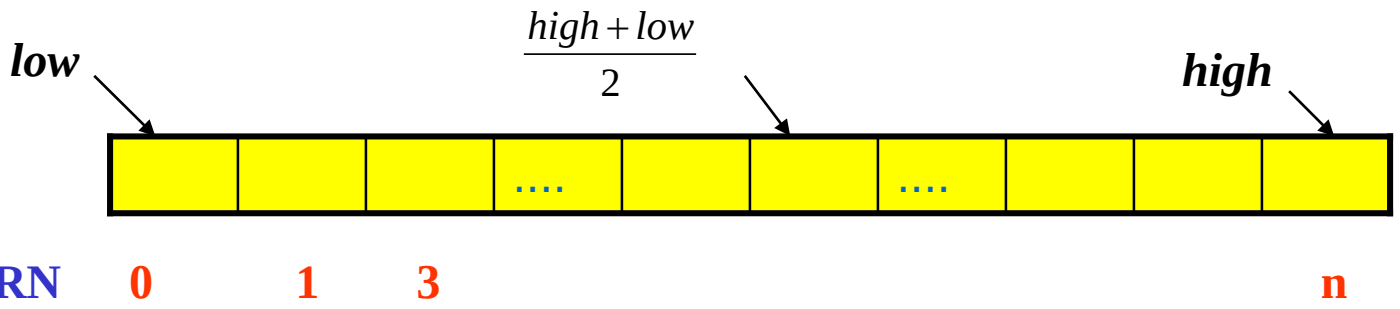
حالت دوم: اگر فايل بر حسب کليد مورد نظر مرتب شده باشد:

- ✓ روش بهينه همان **جست و جوي دودويي** ميباشد. **(چرا؟)**
- ✓ الگوريتم آن در شکل 13-6 کتاب موجود است. **(با اشتباه چاپي!)**

بازیابی سریع داده ها

یادآوري الگوریتم جستجوي دودویي :

```
int BinarySearch
(FixedRecordFile & File, RecType & obj, KeyType & key)
{
     int low = 0; int high = file.NumRecs()-1;
     While (low <= high)
     {
            int guess = (high + low) / 2;
            file.ReadByRRN (obj, guess);
            if (obj.Key() == key) return 1;
            if (obj.Key() < key ) low = guess +1;
                        else high = guess - 1;
     }
     return 0;
}
```



بازیابی سریع داده ها

مقایسه با جست وجوي سري(sequential)؟

مثال:

- ✓ جستجوي كليد در يک فايل با تعداد 2000= n ركورد.

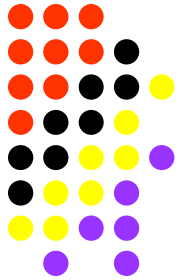
حالت اول: جست و جوي سري:

- ✓ تعداد ماكزيمم ركورد هاي خوانده شده برابر با تعداد كل ركورد ها خواهد بود.
- ✓ ممكن است تا 2000 ركورد خوانده شود.
- ✓ اگر تعداد **ركورد ها** دوبل شود، **تعداد خواندن** ركورد نيز دوبل خواهد شد. **(چرا؟)**

حالت دوم: جست و جوي دودويي:

- ✓ تعداد ماكزيمم ركورد هاي خونده شده برابر با 1+log(n) خواهد بود.
- ✓ ممكن است تا 1+log(2000) يعني 11 ركورد خوانده شود.
- ✓ اگر تعداد **ركورد ها** دوبل شود، فقط **يک خواندن** ركورد **اضافه** مي گردد.
- ✓ براي جست و جوي دودويي بايستي طول ركورد **ها ثابت** باشد. **(چرا؟)**

بازیابی سریع داده ها



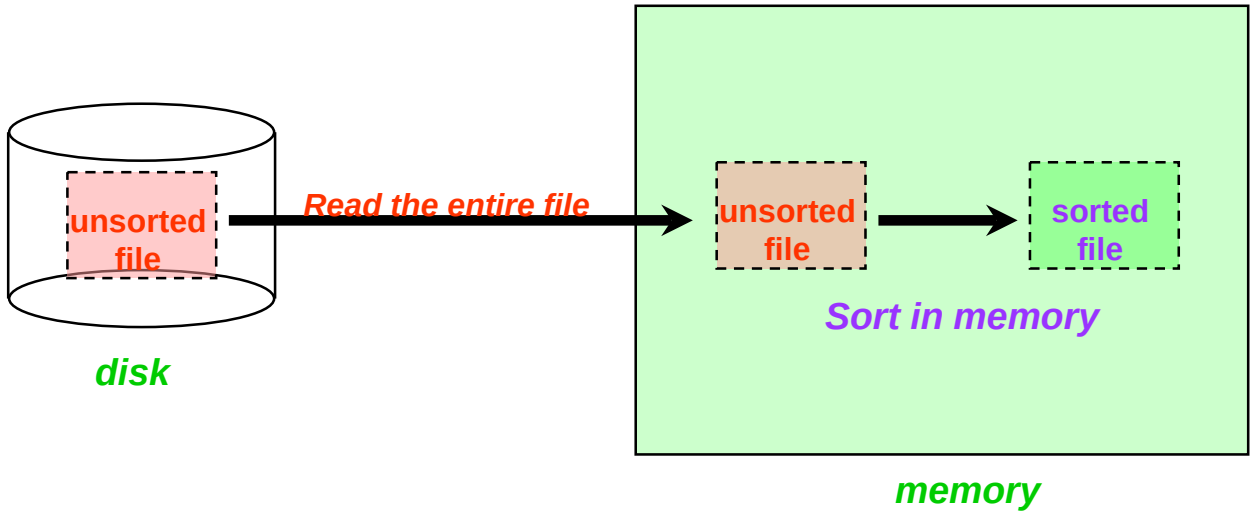
محدوديت ها يا معايب جست و جوي دودويي كدامند؟

- ✓ **جست و جوي يک کليد** مشخص معمولا **بيش از يک يا دو دسترسي** به ديسک نياز دارد. (چرا؟)
- ✓ مثلا در يک فايل با **10000**رکورد، **16** يا **17 دسترسي به ديسک** لازم خواهد بود.
- ✓ **نگهداري** يک **فايل** بطور **مرتب شده هزينه بالايي** خواهد داشت. (کدام؟)
- ✓ **هزينه ها؟ (CPU ، I/O ، متد برنامه نويسي، ...)**
- ✓ انجام **مرتب سازي** فايل **در حافظه** اصلي (RAM) فقط در مورد **فايل هاي کوچک عملي** ميباشد.
- ✓ در مورد **فايل هاي بزرگتر** بايستي **تعداد زيادي دسترسي به ديسک** پيش بيني شود. (چرا؟)
- ✓ استفاده از **RRN** براي فايل هاي حاوي **رکورد متغير** عملي **نخواهد بود**.

بازیابی سریع داده ها

روش مرتب سازي کليدها (key sorting) چگونه است؟

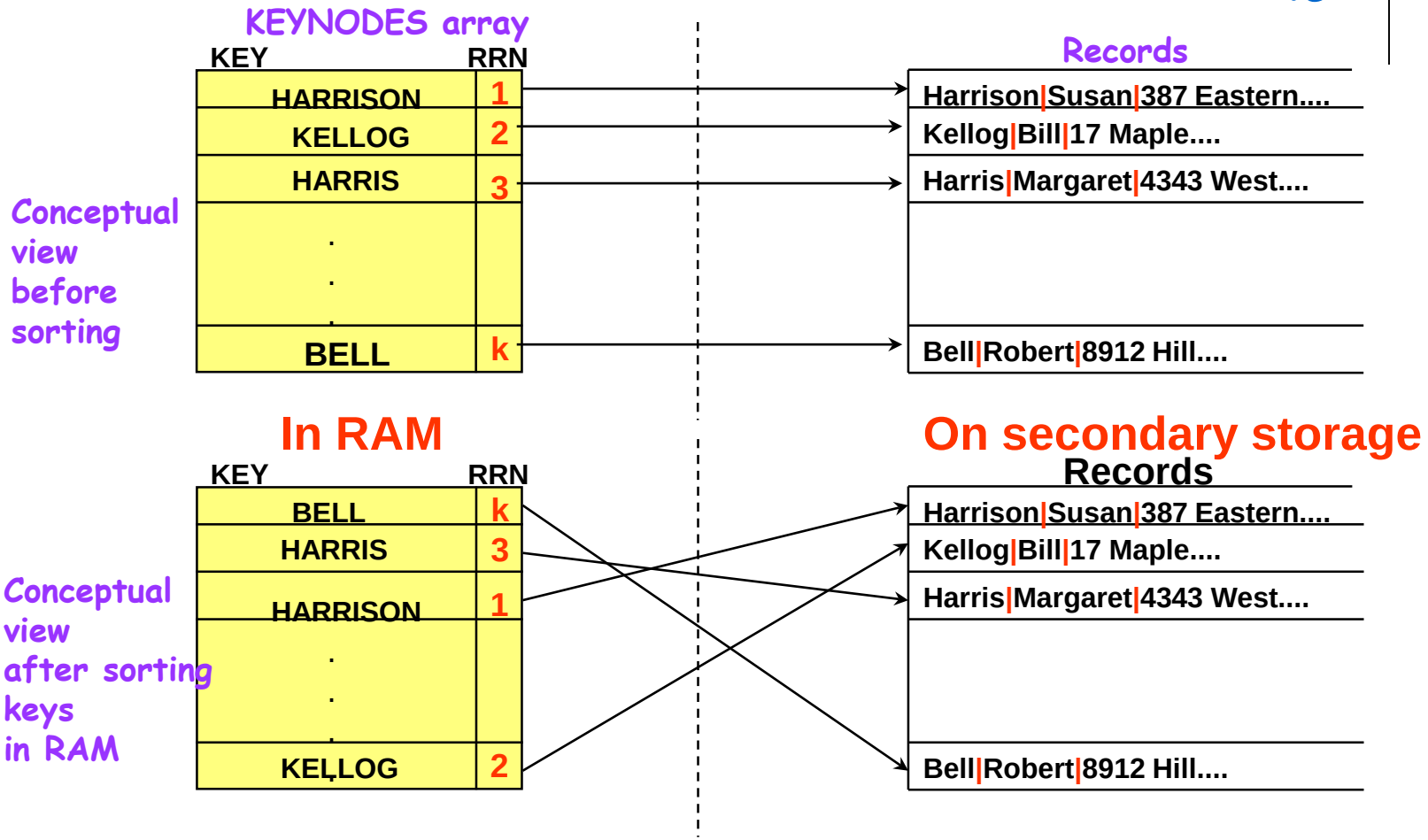
- ✓ روشي براي **مرتب سازي فايل هاي بزرگ** که در حافظه RAM جا نميگيرند.
- ✓ هنگام مرتب سازي، از **آوردن کل رکورد ها** به حافظه **خودداري** ميگردد.
- ✓ برای مرتب سازی کافیست **فقط مقادیر کلید** رکوردها در حافظه موجود باشد.
- ✓ **همراه با** RRN رکوردها! **(چرا؟)**
- ✓ دراينصورت **مرتب سازي کل کليد ها در حافظه** انجام ميشود. (Internal Sort)
- ✓ سپس **بترتيب کليدها**، **رکوردها را خوانده** و در **فايل جديدي** مينويسيم.



مرتب سازي كليدها (key sorting)

مرتب سازي كليدها (key sorting) چگونه است؟

مثال:



مرتب سازي كليدها (key sorting)

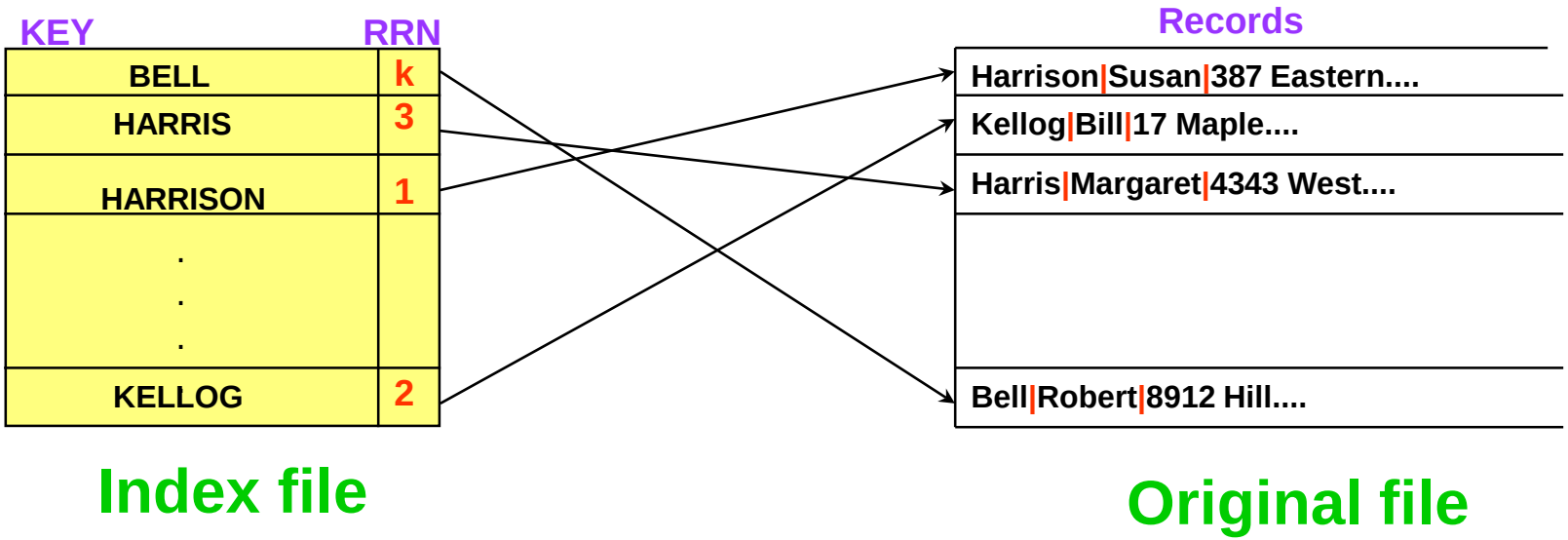
چه تعداد دسترسي به ديسک نياز خواهد بود؟

در مرحله اول: کل رکوردهاي فايل بايستي **بطور سري** (**sequential**) خوانده شوند.

در مرحله دوم: تک تک **رکوردها** بطور **Random** با استفاده از **RRN** خوانده شده و در فايل جديد **نوشته** خواهند شد. (نوشتن بطور سري ؟)

چه احتياجي به دوباره نويسي فايل وجود دارد؟

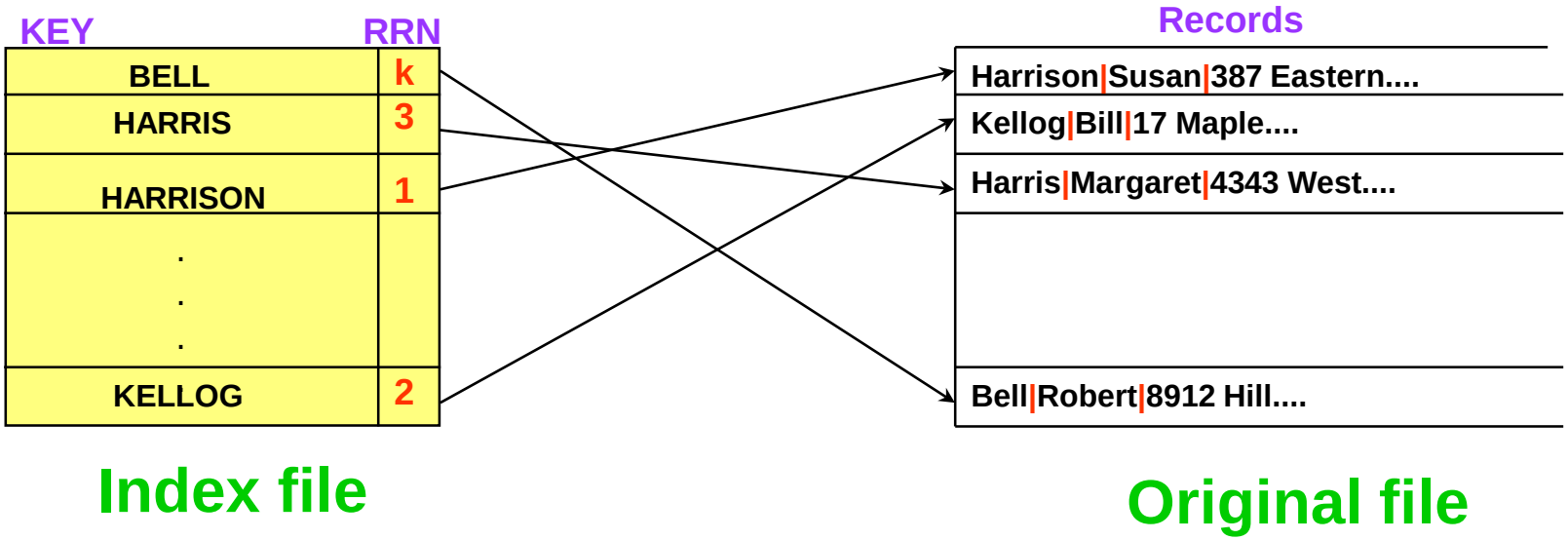
آيا كافي نيست كه ليست مرتب شده كليد ها را حفظ كنيم؟



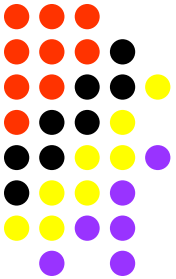
بازیابی سریع داده ها - Indexing

روش Indexing چیست؟

- ✓ **کلیدهاي مرتب شده** یک فایل را در جایي مثلا **یک فایل دیگر** حفظ میکنیم.
- ✓ **این فایل را index مینامیم.**
- ✓ براي **دسترسي سریع** به یک رکورد با کلید مشخص، **از آن استفاده** میکنیم. (چگونه؟)



بازیابی سریع داده ها - Indexing



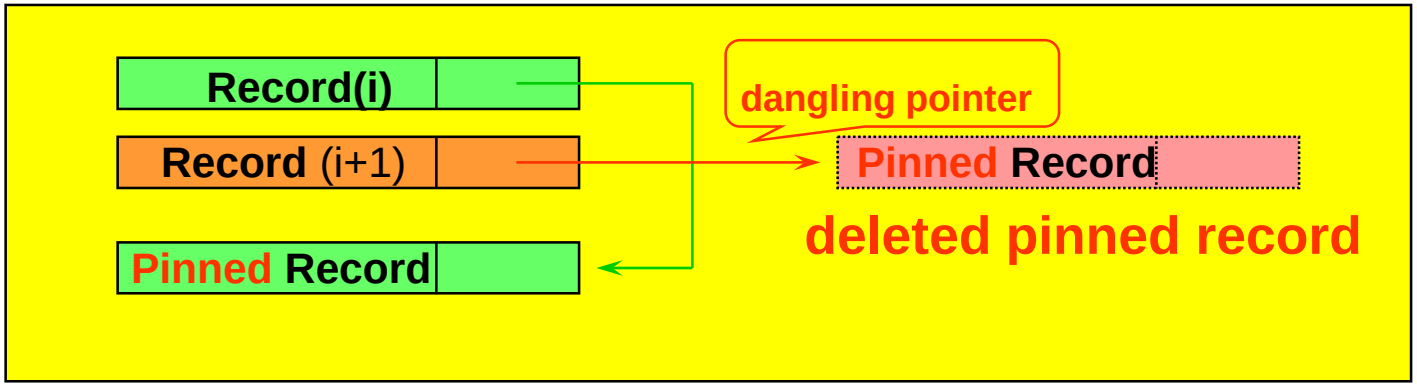
مزاياي Indexing کدامند؟

- ✓ امکان **مرتب سازي** داده ها **بدون** نیاز به **جابجایی رکوردها** در فایل. (چرا؟)
- ✓ امکان **تعریف مسیرهای مختلف** براي **بازیابي** سریع داده ها. (چگونه؟)
- ✓ امکان **دسترسي سریع** به فایل هاي با **رکورد متغیر** بر حسب کلید.
- ✓ امکان **استفاده بهینه** از حافظه **RAM** براي جست و جوی کلید ها. (چرا؟)
- ✓ امکان انجام عمل **جست و جوي دودويي در حافظه** RAM.
- ✓ جلوگیری از ایجاد **اشاره گرهای سرگردان** (**dangling pointers**) در داخل فایل. (چگونه؟)

Indexing - بازیابی سریع داده ها

اشاره گرهای سرگردان (dangling pointers) چیست؟

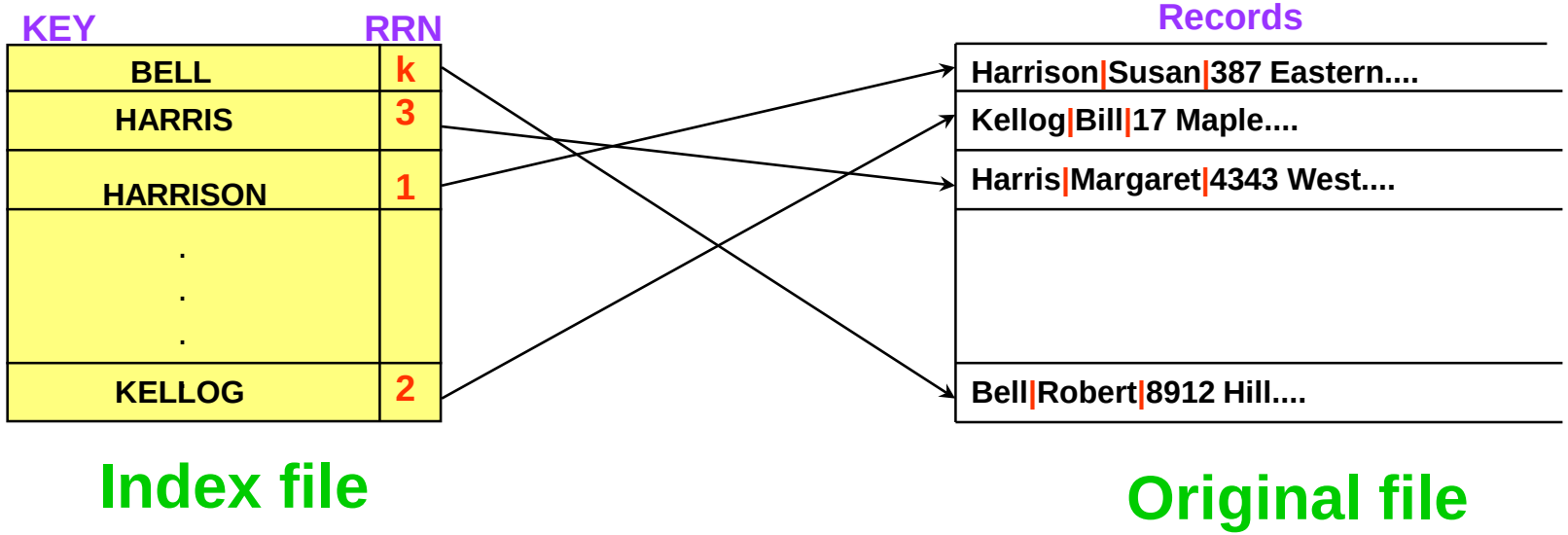
- ✓ در روشهاي بازيابي فضاي فايل ها و استفاده از Avail List ديديم كه ركوردها بوسيله نوعي اشاره گر (مثل RRN يا Byte Offset) به يكديگر مرتبط ميباشند.
- ✓ این **رکوردها** را Pinned Record میخوانیم
- ✓ تغییر محل فیزیکی آنها باعث **ایجاداشاره گرهای سرگردان** (dangling pointers) میشود.
- ✓ استفاده از indexing **مانع ایجاد این مشکل** خواهد شد. (چرا؟)



File with pinned records

Indexing - بازیابی سریع داده ها

- نگهداري index ها در خارج از حافظه RAM چگونه خواهد بود؟
- حفظ صحت اطلاعات در index ها چگونه خواهد بود؟

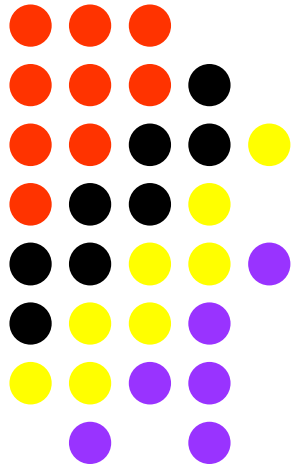


Lecture 10

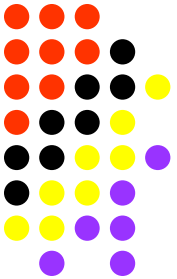
نگهداري فايلهاي ايندكس دار

Maintenance of Indexed files

(Sections 7.5-7.6)



نگهداري فايلهاي ايندكس دار
Maintenance of Indexed files



نگهداري فايلهاي ايندكس دار چه مسائلي را بهمراه دارد؟

چه عملياتي روي فايل يا ايندكس آن بايستي درنظرگرفت؟

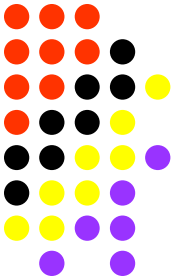
مشكلات ايندكس هاي بزرگتر از فضاي حافظه چيست؟

موارد استفاده ايندكس هاي متعدد چيست؟

ساختار ايندكس هاي ثانوي چگونه است؟

چه عملياتي روي ايندكس ثانوي بايستي درنظرگرفت؟

نگهداري فايلهاي ايندكس دار
Maintenance of Indexed files

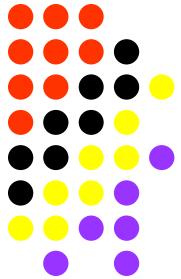


نگهداري فايلهاي ايندكس دارچه مسائلي را بهمراه دارد؟

چه عملياتي روي فايل يا ايندكس آن بايستي درنظرگرفت؟

1) ايجاد اوّليه ايندكس به همراه خود فايل
2) آوردن ايندكس در حافظه RAM قبل از استفاده از فايل
3) بازنويسي ايندكس روي ديسک بعد از استفاده از فايل
4) ايجاد ركوردها
5) حذف ركوردها
6) به روز آوردن ركورد ها (Update)

نگهداري فايلهاي ايندكس دار



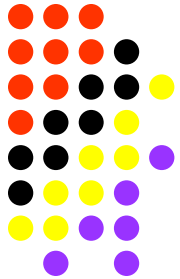
1) **ايجاد اوّليه ايندكس به همراه خود فايل:**

- ✓ هنگام **ايجاد فايل** (**Create File**).
- ✓ بايستي **ايجاد ايندكس** مربوطه را نيز پيش بيني نمود.
- ✓ اگر چه در آغاز هر دو **تهي** از داده ها (**Data**) مي باشند.

2) **آوردن ايندكس در حافظه RAM قبل از استفاده از فايل:**

- ✓ هنگام شروع استفاده از فايل (**Open File**).
- ✓ بايستي ايندكس نيز **باز** شده.
- ✓ داده هاي ايندكس به حافظه RAM آورده شوند (**Load**).

نگهداري فايلهاي ايندكس دار



3) بازنويسي ايندكس روي ديسک بعد از استفاده از فايل:

- ✓ در پايان استفاده از فايل (**Close File**)،
- ✓ بايستي داده هاي ايندكس نيز در فايل ايندكس نوشته شده (**Rewrite**)،
- ✓ و سپس هر دو فايل **بسته** شوند.

مديريت صحت (به روز بودن) فايل ايندكس:

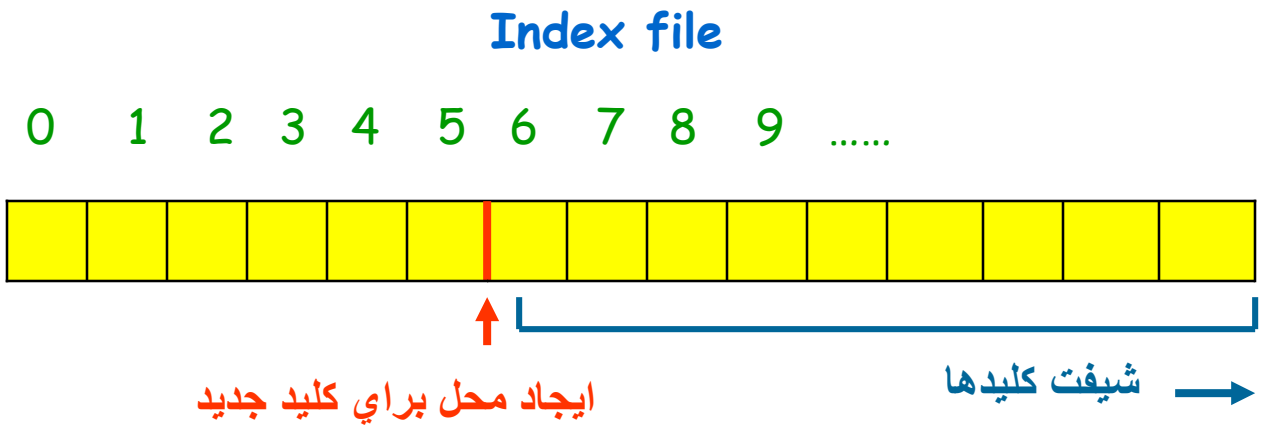
- ✓ براي جلوگيري از حوادث **پيش بيني نشده** (مثل **Power Failure**).
- ✓ بايستي **يک** علامت (**Flag**) در آغاز فايل ايندكس پيش بيني شود.
- ✓ تا وقتيکه ايندكس به روز نشده است اين **Flag** در حالت "**ON**" قرار داشته باشد. (کي؟)
- ✓ هنگام استفاده مجدد اگر **Flag=ON** باشد بايستي **ايندكس** بازسازي شود. (چرا؟)

نگهداري فايلهاي ايندكس دار

(4

ايجاد ركوردها:

- ✓ هنگام ايجاد يک **رکورد جديد** در فايل اين رکورد **در آخر فايل** اضافه مي شود
- ✓ ولي **کليد مربوط** به اين رکورد بايستي **در محل مناسب** خود در ايندکس اضافه (**Insert**) شود
- ✓ بطوريکه همواره کليدها در ايندکس مرتب شده باشند (**Sorted**)

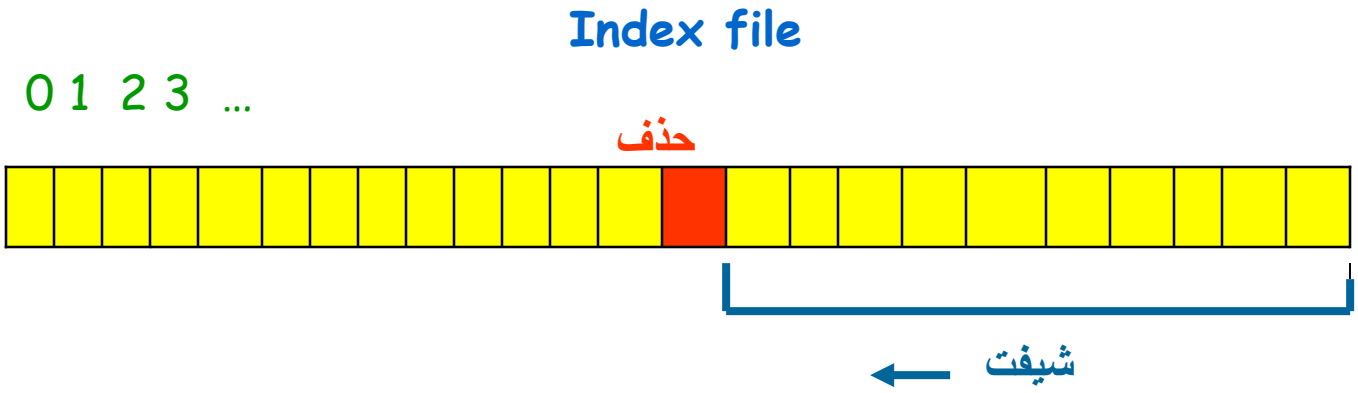


نگهداري فايلهاي ايندكس دار

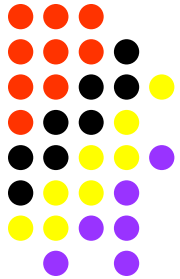
(5 حذف ركوردها:

- ✓ هنگام حذف يک ركورد در فايل،
- ✓ فضاي ايجاد شده در فايل به **Avail list** اضافه ميشود.
- ✓ ولي **در مورد ايندكس اينطور نيست**،
- ✓ دو راه حل وجود دارد:

  - o **يا** كليد مربوطه از ليست كليدها حذف مي شود و كليدهاي بعد از آن **يک مرحله** شيفت داده ميشوند
  - o **يا** فقط در محل كليد مربوطه علامت گذاري مي شود (**delete flag**) .



نگهداري فايلهاي ايندكس دار



6) **به روز آوردن رکوردها (Update):**

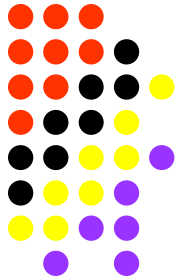
هنگام به روز آوردن يک رکورد داده:

- ✓ در صورت **تغيير طول** رکورد بايستي:
  - o اوّل مانند حالت حذف،
  - o و سپس مانند حالت اضافه نمودن رکورد عمل کرد.
- ✓ در غير اينصورت **در همان مکان قبلي رکورد** به روز مي شود.

اما در هر حال در مورد ايندکس مربوطه:

- ✓ اگر **مقدارکليد تغيير** کرده باشد بايستي:
  - o اوّل مانند حالت حذف،
  - o و سپس مانند حالت اضافه نمودن رکورد عمل کرد.
- ✓ در غير اينصورت **هيچ عملي** لازم نمي باشد.

ايندكس هاي بزرگتر از فضاي حافظه



اگرايندكس بزرگتر از فضاي حافظه (RAM) باشد چه اشكالاتي دارد؟

درصورتيكه امكان آوردن كل ايندكس به حافظه RAM نباشد:

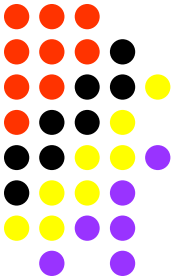
- ✓ براي جستجوي دودئي هركليدها چندين دسترسي به ديسک خواهيم داشت!
- ✓ هنگام ايجاد يا حذف هركليد، **عمل شيفت باعث چندين دسترسي** به ديسک خواهد شد!

در اين موارد چه راه حل هايي وجود دارد؟

كداميک از مزاياي يک ايندكس ساده حتّي روي ديسک نيز به قوت خود باقيست؟

- ✓ به هرحال امكان جستجوي دوديي را براي فايلي با ركورد متغير فراهم ميكند.
- ✓ حتّي روي ديسک هم، مرتّب سازي آن كم هزينه تراز فايل اصلي ميباشد. (چرا؟)
- ✓ به هرصورت از ايجاد نشانگرهاي سرگردان جلوگيري ميكند. (چرا؟)

استفاده از ایندکس های متعدد



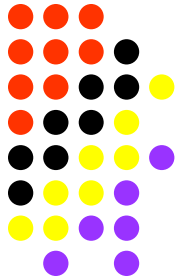
موارد استفاده ی ایندکس های متعدد چیست؟

- ✓ ایجاد چند مسیر با کلیدهاي مختلف براي دسترسي به داده هاي یک فایل.
- ✓ براي هر فایل مي توانیم یک ایندکس اصلي (**Primary**) و چند ایندکس ثانوي (**Secondary**) تعریف کنیم.

مثال:

- ✓ یک فایل شامل اطلاعات مربوط به آهنگ ها در نظر میگیریم.
- ✓ **ایندکس اصلي** به کمک **دو فیلد** (**Label + ID no.**) تعریف شده است.
- ✓ میخواهیم امکان دسترسي از طریق نام سازنده (**Composer**) را نیز بدهیم.
- ✓ بایستي یک ایندکس ثانوي روي این فیلد تعریف کنیم.

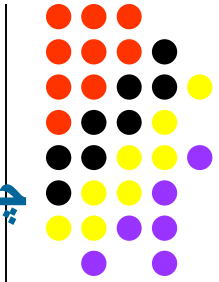
استفاده از ایندکس های متعدد



ایندکس اصلي به کمک دو فیلد (Label+ID no.) تعریف شده است.

Record Address	Label	ID No.	Title	Composer(s)	Artist(s)
17	LON	2312	Romeo and Juliet	Prokofiev	Maazel
62	RCA	2626	Quartet in C Sharp Minor	Beethoven	Julliard
117	WAR	23699	Touchstone	Corea	Corta
152	ANG	3795	Symphony No.9	Beethoven	Giulini
196	COL	38358	Nebraska	Springsteen	Springsteen
241	DG	18807	Symphony No.9	Beethoven	Karajan
285	MER	75016	Coq d'Or Suite	Rimsky-Korsakov	Ltinsdorf
338	COL	31809	Symphony No.9	Dvorak	Bernstein
382	DG	1E+05	Violin Concerto	Beethoven	Ferras
427	FF	245	Good News	Sweet Honey in the Rock	Sweet Honey in the Rock

استفاده از ايندكس هاي متعدد



چگونه امكان دسترسي از طريق نام سازنده (Composer) را بدهيم؟

ساختار متداول يک ايندكس ثانوي چگونه است؟

Secondary key	Primary key
Beethoven	ANG 3795
Beethoven	DG 139201
Beethoven	DG 18807
Beethoven	RC A2626
Corea	WAR 23699
Dvorak	COL 318091
Prokofiev	LON 2312

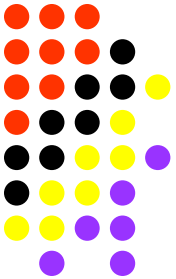
ساختار متداول:

- ✓ استفاده از كليد اصلي بجاي Byte Offset ركوردها.
- ✓ اين روش را "Postponing the binding" يا به تاخير انداختن اتصال مي گويند.

مزيت اين ساختار چيست؟

- ✓ اعمال مربوط به **ايجاد، حذف يا به روز نمودن** ركوردها را **ساده تر**، **سريعتر** و **مطمئن تر** مي نمايد. (چرا؟)

عمليات روي ايندكس ثانوي



چه عملياتي روي ايندكس ثانوي بايستي درنظرگرفت؟

1) **ايجاد** ركورد:

- ✓ هنگام ايجاد ركورد در فايل و ايندكس اصلي،
- ✓ بايستي كليد ثانوي نيز در ايندكس ثانوي ايجاد شود.
- ✓ بايستي محتواي كليد به فرم كانونيک (**Canonical Form**) ثبت شوند. (چرا؟)
- ✓ (مثلا با حروف كاپيتال و با طول مشخص)
- ✓ در صورت وجود **مقادير تكراري براي كليد ثانوي** بهتر است آنها **به ترتيب كليد اصلي** مرتّب (**Sort**) شوند.

عمليات روي ايندكس ثانوي

2) **حذف** رکورد:

- ✓ هنگام حذف رکورد قاعدتا بايستي **کليه** ايندکس ها به روز شوند.
- ✓ اما اگر فايل **ايندکس هاي متعدد** داشته باشد، اين کار **پرهزينه** خواهد بود. (چرا؟)

روش ديگر:

- ✓ **فقط فايل اصلي و ايندکس اصلي به روز مي شوند** و کاري با ايندکس هاي ثانوي نداريم.
- ✓ هنگام استفاده از ايندکس ثانوي چک مي کنيم که رکورد مربوطه از ايندکس اصلي حذف نشده باشد.
- ✓ اگر تعداد رکوردهاي حذف شده زياد باشد، بايستي ايندکس ثانوي **باز سازي** شود. (چرا؟)

عمليات روي ايندكس ثانوي

3) **به روز آوردن رکوردها:**

حالت اول: مقدار کليد ثانوي تغيير کرده:

- ✓ بايستي ايندکس ثانوي به روزآورده و دوباره **مرتّب** شود.

حالت دوم: مقدار کليد اصلي تغيير کرده:

- ✓ بعد از به روزآوردن ايندکس اصلي و مرتّب نمودن آن،
- ✓ بايستي **ايندکس ثانوي نيز به کليد اصلي جديد اشاره کند** و دوباره مرتّب شود.

حالت سوم: فقط فيلدهاي غير کليدي تغيير کرده اند:

- ✓ معمولا عملي روي ايندکس ها لازم نيست. (چرا؟)
- ✓ مگر اينکه طول رکورد تغيير کرده باشد و محل آن در فايل جابجا شود. (چرا؟)
- ✓ در نتيجه **Offset** در **ايندکس اصلي** تغيير خواهد کرد.

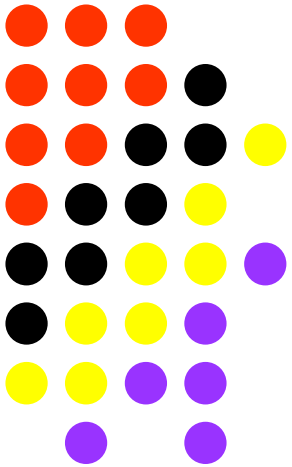
In the Name of God

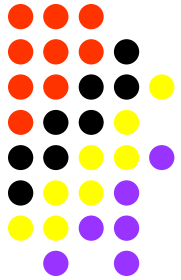
Lecture 11

ساختارهاي ايندكس ثانوي،
پردازش همزمان داده ها

Secondary Index structures,
Co-sequential processing

(Sections 7.7-7.9, 8.1-8.2)





ساختارهاي ايندكس ثانوي،
پردازش همزمان داده ها

- چگونه ايندكس هاي ثانوي جهت ايجاد مسيري تركيبي استفاده ميگردند؟
- تركيب چند ايندكس ثانوي چگونه انجام ميشود؟
- روشهاي بهينه سازي ساختار ايندكس ثانوي كدامند؟
- چگونه از ليست هاي معكوس در ساختار ايندكس استفاده ميگردد؟
- چگونه ميتوان از ايندكس ها جهت دسته بندي اطلاعات استفاده نمود؟
- انواع روشهاي اتصال ايندكس ها به داده ها كدامند؟
- منظوراز پردازش همزمان داده ها چيست؟
- الگوريتم مقايسه يا ادغام داده ها چگونه است؟

ساختارهاي ايندكس ثانوي
Secondary Index structures

چگونه ايندكس هاي ثانوي جهت ايجاد مسيري تركيبي استفاده ميگردند؟

تركيب چند ايندكس ثانوي چگونه انجام ميشود؟ (combination)

مثال :

- ✓ فايل اطلاعات مربوط به **آهنگ ها** در نظر ميگيريم.
- ✓ مي خواهيم **تمام آهنگ هاي BEETHOVEN با تيتر symphony No. 9** را پيدا كنيم.
- ✓ جدول زير با **تركيب دو ايندكس composer** و **title** اين نتيجه را به ما خواهد داد.
- ✓ با استفاده از ليست نهايي (**mached list**) و با كمك **ايندكس اصلي** ركوردها را ميخوانيم.

Matches from composer index	Matches from title index	Matched list (A & B)
ANG3795 →	ANG3795 →	ANG3795
DG139201	COL31809	DG18807
DG18807 →	DG18807	
RCA2626		

ساختارهاي ايندكس ثانوي

چه اشكالاتي در ساختار اوليه ايندكس ثانوي وجود دارد؟

- ✓ براي هر كليد جديد (حتي با مقدار تكراري) بايستي ايندكس **دوباره** مرتب شود.
- ✓ مقادير تكراري كليد ثانوي فضايي را اشغال مي كنند كه مي توانستيم صرفه جويي نماييم.

مثال:

Secondary key	Primary key
Beethoven	ANG 3795
Beethoven	DG 139201
Beethoven	DG 18807
Beethoven	RC A2626
Corea	WAR 23699
Dvorak	COL 318091
Prokofiev	LON 2312

ساختارهاي ايندكس ثانوي

چه اشكالاتي در ساختار اوليه ايندكس ثانوي وجود دارد؟

روشهاي بهينه سازي ساختار ايندكس ثانوي كدامند؟

راه حل اول: **استفاده از يک ماتريس** كه براي آن چند ستون پيش بيني شده باشد.

مثال:

Beethoven	ANG3795	DG139201	DG18807	RCA2626
COREA	VAR23699			
DVORAC	COL31809			
……	…….			

معايب اين راه حل كدامند؟

- ✓ تعداد ستون ها ممكن است كافي نباشد.
- ✓ فضاي اضافي رزرو شده به هدر ميرود.

ساختارهاي ايندكس ثانوي

روشهاي بهينه سازي ساختار ايندكس ثانوي كدامند؟

راه حل دوم : استفاده از **ليست هاي معكوس** (**inverted** lists):

- ✓ در ايندكس ثانوي فقط **يک مکان** براي **هرمقدار کليد** رزرو مي شود.
- ✓ از آنجا بکمک **يک اشاره گر** به ليست جداگانه اي از کليدهاي اصلي اشاره مي شود.

مثال:

	Key Value	ptr
0	BETTHOVEN	3
1	COREH	2
2	DVORAK	5
3	……	7

	Label ID	ptr
0	LON2312	-1
1	RCA2626	-1
2	WAR23699	-1
3	ANG3795	6
4	DG18807	1
5	COL31809	-1
6	DG139201	4
7	……	

ساختارهاي ايندكس ثانوي

روشهاي بهينه سازي ساختار ايندكس ثانوي كدامند؟

مزايا و معايب راه حل استفاده از ليست هاي معكوس كدامند؟

مزايا:

- ✓ هنگام ايجاد **كليد تكراري** عمل **مرتب سازي** ايندكس **لازم نمي باشد**. (چرا؟)
- ✓ هنگام **حذف ركوردها** كافيست از **يک علامت** مانند " **-1**" در محل اشاره گر استفاده شود.
- ✓ **مرتب سازي** ايندكس **سريعتر** مي باشد چون اندازه آن كوچكتر است. (چرا؟)
- ✓ **فضاي كمتري** براي **مرتب سازي** (حتي روي ديسک) لازم مي شود.
- ✓ **ليست معكوس** نيازي به **مرتب سازي ندارد** و فضاي آن **براحتی قابل بازيابی** مي باشد. (چرا؟)

معايب:

- ✓ **پراكندگی كليدها** در **ليست معكوس**. (منظور؟)
  - (راه حل : استفاده از مكانيسم **paging**)

ساختارهاي ايندكس ثانوي

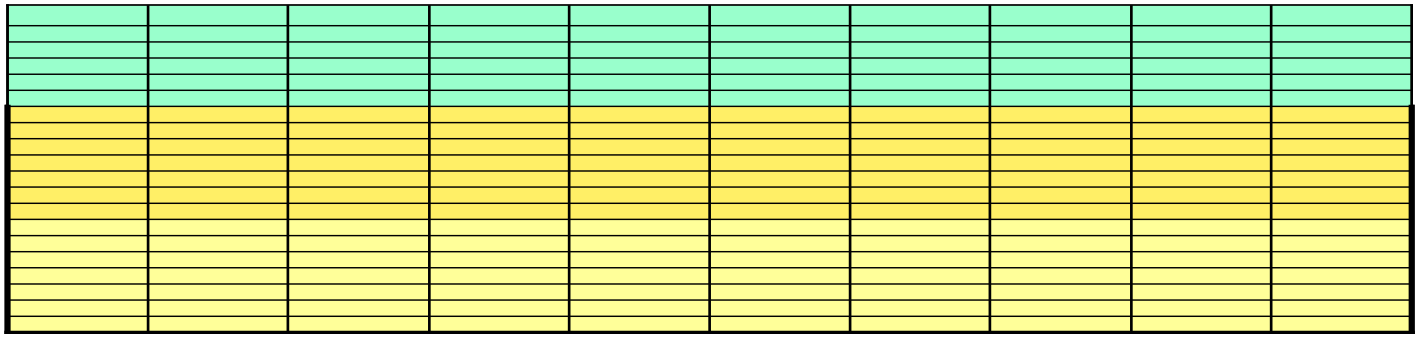
چگونه ميتوان از ايندكس ها جهت دسته بندي اطلاعات استفاده نمود؟

- يكي ديگر از **موارد استفاده** ايندكس ها **دسته بندي افقي** اطلاعات در **فايل هاي بزرگ** ميباشد.

(Selective indexes)

مثال:

- فايل اطلاعات مربوط به **آهنگ ها** در نظر ميگيريم.
- يک ايندكس مي تواند فقط شامل اطلاعات مربوط به قبل از سال 1970 باشد
- ايندكسي ديگر نيز شامل اطلاعات بعد از اين تاريخ باشد.



ساختارهاي ايندكس ثانوي

انواع روشهاي اتصال ايندكس ها به داده ها كدامند؟

- ✓ **اتصال** ايندكس با محل فيزيكي ركورد (**Byte Offset**) را **binding** مي گويند.
- ✓ در مورد **ايندكس اصلي** عمل اتصال **هنگام ايجاد كليد** در ايندكس انجام مي شود.

(Tight Binding)

- ✓ در مورد **ايندكس ثانوي** عمل اتصال **هنگام استفاده از كليد** ايندكس انجام مي شود.

(Postponing Binding)

Secondary key	Primary key
Beethoven	ANG 3795
Beethoven	DG 139201
Beethoven	DG 18807
Beethoven	RC A2626
Corea	WAR 23699
Dvorak	COL 318091
Prokofiev	LON 2312

ساختارهاي ايندكس ثانوي

مزايا و معايب روشهاي اتصال ايندكس ها به داده ها كدامند؟

مزاياي postponing binding

- ✓ عمليات لازم هنگام **ايجاد يا حذف** ركورد ها **ساده تر** و **سريعتر** انجام مي شوند. (چرا؟)
- ✓ اين **روش مطمئن تر** است زيرا **تغييرات** مهم **فقط در يک محل** اعمال مي شوند. (كدام؟)

معايب postponing binding

- ✓ **دسترسي** به فايل از طريق كليد ثانوي **كندتر** مي شود. (چرا؟)

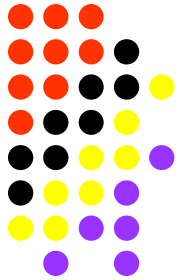
موارد استفاده postponing binding

- ✓ فايل هايي كه در آن ها اعمال **ايجاد حذف يا به روز** كردن **دائما انجام مي شود**. (چرا؟)

موارد استفاده tight binding

- ✓ فايل هايي كه **داده هاي** آنها **ثابت** هستند يا زياد تغيير نمي كنند. (چرا؟)
- ✓ فايل هايي كه **سرعت خواندن** آنها **مهم** است (فايل هاي روي CD-ROM). (چرا؟)

پردازش همزمان داده ها
Co-sequential Processing



منظوراز پردازش همزمان داده ها چیست؟

- ✓ اجراي **عمليات همزمان** (مثلا خواندن) **بطور سري** روي دو ليست (يا **فايل**) مرتب شده.

(Co-sequential processing)

موارد استفاده پردازش همزمان داده ها کدامند؟

- ✓ **مقايسه** اعضاي دو ليست (يا فايل) **(Matching)**
- ✓ **ادغام** اعضاي دو ليست (يا فايل) **(Merging)**

مثال 1:

- ✓ **مقايسه حسابهاي** دو فايل **accounts** و **transaction** در يک سيستم بانکي

Accounts (account number, person name, account balance)

Transactions (account number, credit debit info)

پردازش همزمان داده ها

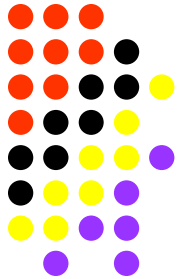
موارد استفاده پردازش همزمان داده ها کدامند؟

مثال 2:

- ✓ **ادغام (Merging) لیست اسامی** دانشجویان در دو کلاس:

List1	List2	Matched list	Merged list
Adams	Adams	Adams	Adams
Carter	Bech	Carter	Bech
Chin	Burns	Davis	Burns
Davis	Carter		Carter
Miller	Davis		Chin
Reston	Peters		Davis
	Rosewald		Miller
	Schmit		Peters
	Willis		Reston
			Rosewald
			Schmit
			Willis

پردازش همزمان داده ها



الگوریتم مقایسه یا ادغام داده ها چگونه است؟

```
item(1) = current item from list 1
item(2) = current item from list 2
if( item(1) < item(2) )
    [output item(1) to output list]
    Get next item from list(1)
if( item(1) > item(2) )
    [output item(2) to output list]
    Get next item from list(2)
if( item(1) = item(2) )
    output the item to output list
    Get next item from list(2) and list(1)
```

(توضیح؟) ←

(توضیح؟) ←

(صفحه 298 کتاب شکل 8-5)

پردازش همزمان داده ها

موارد استفاده پردازش همزمان داده ها کدامند؟

مثال کاربردي:

- ✓ در يک سيستم حسابداري بانکي دو فايل زير را در نظر مي گيريم:

1) Master File : شامل موجودي ماهانه حسابها.

2) Transaction File : شامل عمليات انجام شده در يک ماه.

- ✓ بايستي برنامه اي بنويسيم که:

1) **عمليات انجام شده روي هر حساب** را در Master File **منعکس** نمايد.

2) **گزارشي از عمليات هر حساب** را نيز ارائه دهد. (Report)

پردازش همزمان داده ها

1) Master File : شامل موجودي ماهانه حسابها.

Acct. No.	Account title	Jan	Feb	Mar
101	Checking account # I	1032.57	2114.56	5219.23
102	Checking account #2	543.78	3094.17	1321.2
505	Advertising expense	25	25	25
510	Auto expenses	195.4	307.92	501.12
515	Bank charges	0.00	0.00	0.00
520	Books and publications	27.95	27.95	87.4
525	Interest expense	103.SO	255.2	380.27
535	Miscellaneous expense	12.45	17.87	23.87
540	Office expense	57.SO	105.25	138.37
585	Postage and shipping	21	27.63	57.45
550	Rent	500	1000	1500
555	Supplies	112	167.5	2441.8

(صفحه 301 – شکل 8.6)

پردازش همزمان داده ها

(2 **Transaction** File : شامل **عملیات** انجام شده **در یک ماه**.

Acct.No	Check No.	Date	Description	Debit/Credit
101	1271	04/02/1997	**Auto expense**	**-78.7**
510	1271	04/02/1997	Tune-up and minor repair	78.7
101	1272	04/02/1997	**Rent**	**-500**
550	1272	04/02/1997	Rent for April	500
101	1273	04/04/1997	**Advertising**	**-87.5**
505	1273	04/04/1997	Newspaper ad re: new product	87.5
102	670	04/02/1997	**Office expense**	**-32.78**
540	670	04/02/1997	Printer cartridge	32.78
101	1274	04/02/1997	**Auto expense**	**-31.83**
510	1174	04/O9/1997	Oil change	31.83

(صفحه 302 – شکل 8.7)

پردازش همزمان داده ها

گزارشي از عمليات هر حساب. (Report)

101 Checking account #1

1271	04/02/97	Auto expense	-78.70	
1272	04/02/97	Rent	-500.00	
1273	04/04/97	Advertising	-87.50	
1274	04/02/97	Auto expense	-31.83	
		Prev. bal: 5219.23	New bal :	4521.20

102 Checking account #2

670	04/02/97	Office expense	-32.78	
		Prev. bal: 1321.20	New bal :	1288.42

505 Advertising expense

1273	04/04/97	Newspaper ad re: new product	87.50	
		Prev. bal: 25.00	New bal:	112.50

510 Auto expenses

1271	04/02/97	Tune-up and minor repair	78.70	
1274	04/09/97	Oil change	31.83	
		Prev. bal: 501.12	New bal :	611.65

(صفحه 302 – شکل 8.8)

پردازش همزمان داده ها

مثال کاربردي (ادامه...):

بايستي برنامه اي بنويسيم كه:

1) عمليات انجام شده روي هر حساب را در **Master File** منعكس نمايد.

روش كارچگونه است؟

- ✓ **شماره حساب** به عنوان **كليد مشترک** بين دو فايل انتخاب مي شود.
- ✓ **Transaction File** بايستي **مرتب** شود (بر حسب **شماره حساب** و سپس **تاريخ**) (چرا؟)
- ✓ الگوريتم **پردازش همزمان** انجام مي شود.

پردازش همزمان داده ها

الگوریتم پردازش همزمان چگونه است؟

```
Item(1): always stores the current master record
Item(2): always stores the current transactions record
- Read first master record
- Print title line for first account
- Read first transactions record
While (there are more masters or there are more transactions) {
    if item(1) < item(2) then {
            Finish this master record:
            - Print account balances, update master record
            - Read next master record
            - If read successful, then print title line for new account
    }
```

(صفحه 307 – شکل 8.13)

پردازش همزمان داده ها

الگوریتم پردازش همزمان چگونه است؟

```
if item(1) = item(2) {
        Transaction matches master:
        - Add transaction amount to the account balance for
                new month
        - Print description of transaction
        - Read next transaction record
    }
    if item(1) > item(2) {
        Transaction with no master:
        - Print error message
        - Read next transaction record
    }
}
```

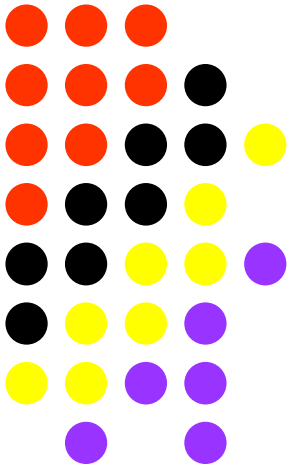
(صفحه 307 – شکل 8.13)

Lecture 12

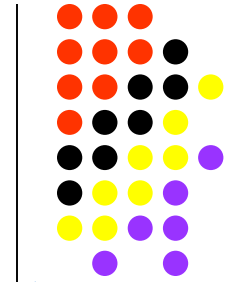
مرتب سازي و ادغام فايلها

(Sorting and Merging files)

(Sections 8.3 - 8.5)



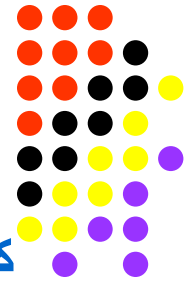
مرتب سازي و ادغام فايلها
(Sort and Merge of files)



- کاربرد هاي ديگر پردازش همزمان (Co-sequential processing) کدامند؟
- الگوريتم ادغام چندتايي (K-way Merge) چگونه است؟
- روش مرتب سازي Selection Tree چيست؟
- روش مرتب سازي Heap Sort چگونه است؟
- روش Overlapping در Heap Sort چگونه است؟
- مرتب سازي فايلهاي بزرگ چه مشکلاتي دارد؟
- روش مرتب سازي Merge-Sort چيست؟

طراحی وبسایت – برنامه نویسی – پروژه پایگاه داده – SQL Server
مهندسی نرم افزار UML و SSADM
۰۹۱۳۱۲۵۳۶۲۰ www.a00b.comآ صفر صفر بی دات کام

مرتب سازي و ادغام فايلها
(Sort and Merge of files)



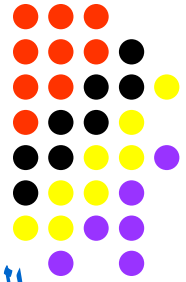
کاربرد هاي ديگر پردازش همزمان کدامند؟ (Co-sequential processing)

- ✓ الگوريتم **ادغام چندتايي** (K-way Merge)
- ✓ روش **مرتب سازي** **Selection Tree**
- ✓ روش **مرتب سازي** **Merge-Sort**

الگوريتم ادغام چندتايي (K-way) چگونه است؟

- ✓ ادغام تعداد **K ليست مرتب شده** و توليد **يک ليست واحد** (مرتب شده).
- ✓ **تعميم** الگوريتم قبلي (2-way).
- ✓ تعريف برداري با تعداد K ليست به نام : **List [1]...List [K]**
- ✓ تعريف برداري با تعداد K آيتم به نام: **item[1]...item [K]**

مرتب سازي و ادغام فايلها



الگوريتم ادغام چندتايي (K-way) چگونه است؟

الگوريتم ادغام (با حذف آيتم هاي تكراري):

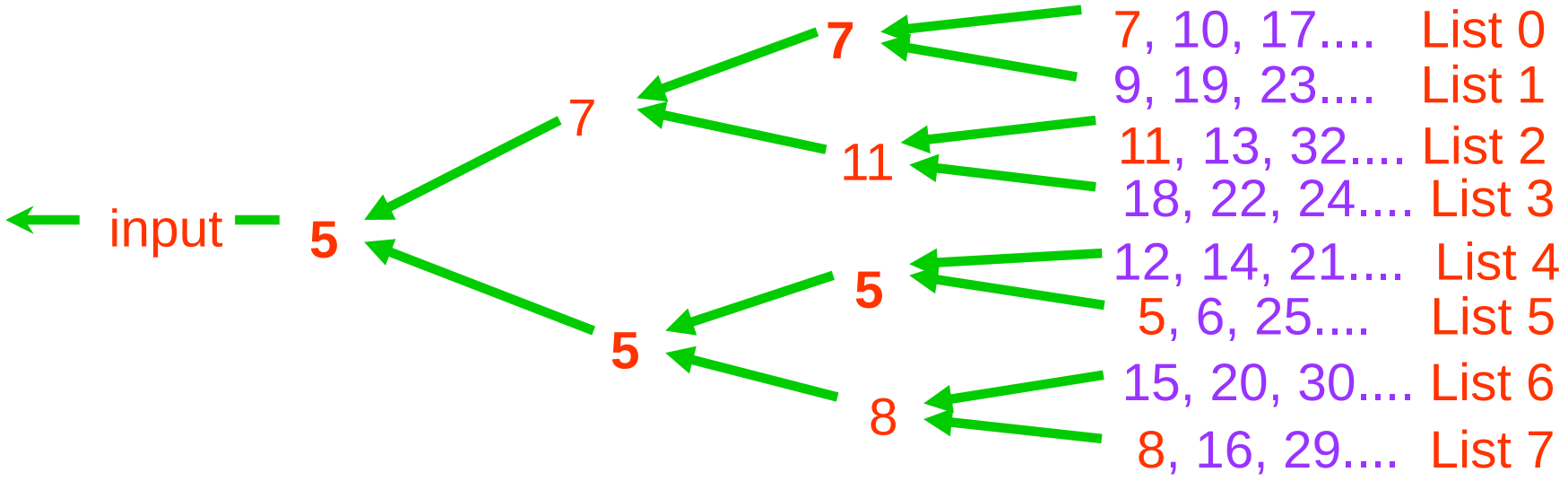
1) **MinItem** = set to min of item[1]…item[K]
2) Output **MinItem** to output list **(ص 309 كتاب)**
3) For i = 1 to K do:
   If item[i]= **MinItem** then
   Get next item[i] from List[i]

مرتب سازي و ادغام فايلها

روش مرتب سازي Selection Tree چيست؟

- ✓ يک الگوريتم ديگر براي ادغام ميبا شد که **مشابه** **مسابقات دوره اي** (مثل فوتبال) عمل ميکند.
- ✓ وقتي **تعداد** **ليست ها** **بيش از 8** باشد اين الگوريتم ارجحيت دارد. **(چرا؟)**
- ✓ مانند يک **Binary Tree** با عمق $Log_2(K)$ عمل مي کند.

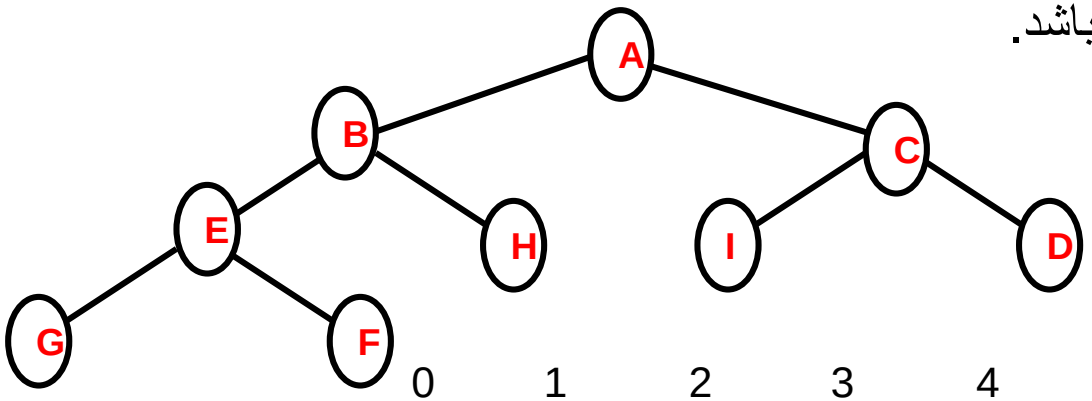
مثال: ص 311 کتاب شکل 8.15

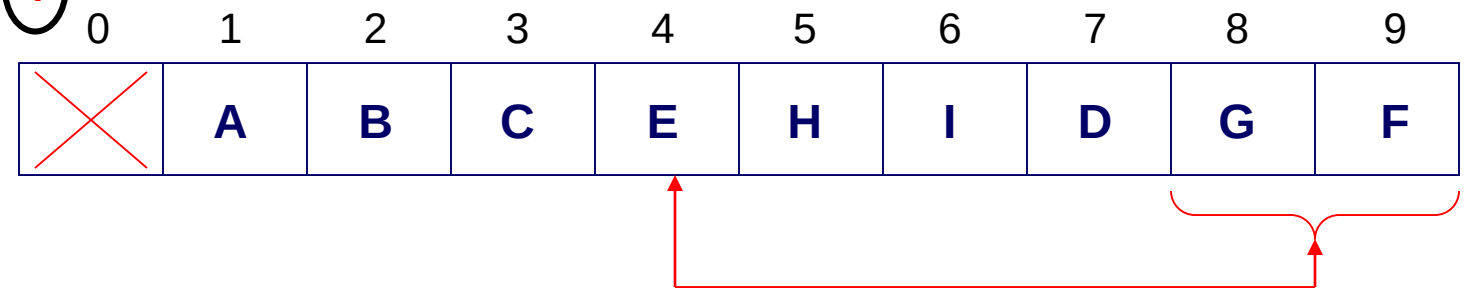


مرتب سازي و ادغام فايلها

روش مرتب سازي Heap Sort چگونه است؟

- ✓ يک الگوريتم مرتب سازي در حافظه (RAM) ميباشد.
- ✓ Heap **يک درخت دودويي کامل** است با ارتفاع $\lfloor \log n \rfloor$ = Height
- ✓ **هر گره** (node) **يک کليد** بيشتر ندارد که بزرگتر يا برابر کليد گره پدر (parent) ميباشد.
- ✓ بصورت **يک آرايه** (Array) ذخيره ميشود.
- ✓ براي هر گره (i) **فرزندان آن** در گره هاي (2i) و (2i+1) ذخيره شده اند.
- ✓ **پدر** هر گره (j) در گره (j/2) ميباشد.





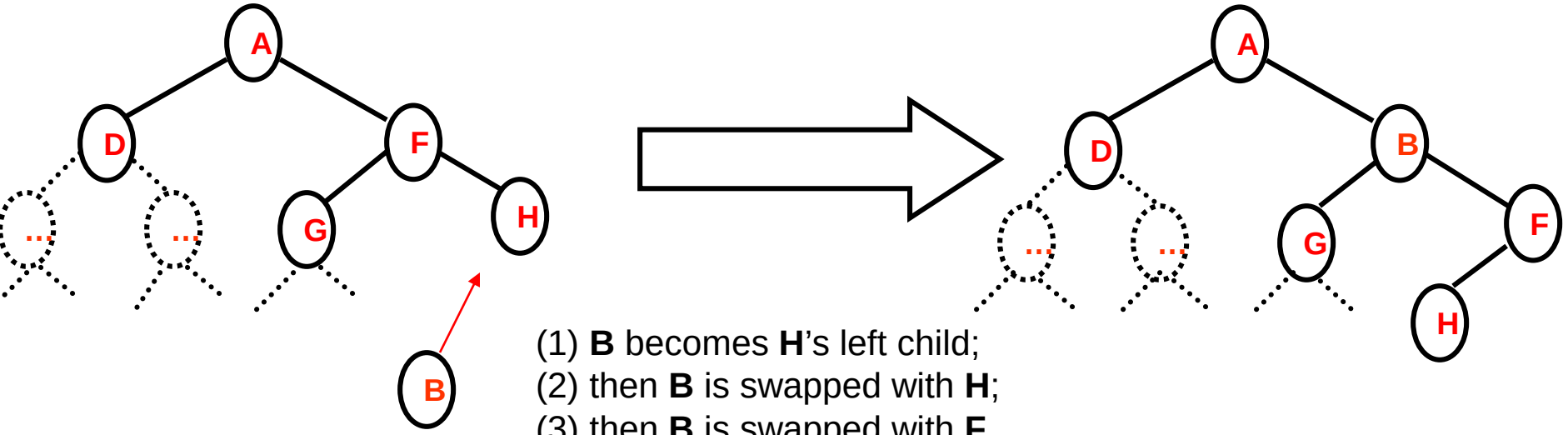
مرتب سازي و ادغام فايلها

الگوريتم Insert در Heap Sort چگونه است؟

1) ركورد جديد در آخر Heap اضافه ميشود.

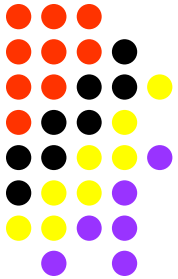
2) كليد آن با كليد گره پدر مقايسه مي شود و اگر مقدار آن كوچكتر بود محل آن با محل گره پدر تعويض ميشود.

3) در صورت لزوم عمل (2) تا ريشه درخت (Root) ادامه مييابد.



(1) **B** becomes **H**'s left child;
(2) then **B** is swapped with **H**;
(3) then **B** is swapped with **F**.

الگوریتم Insert در Heap Sort



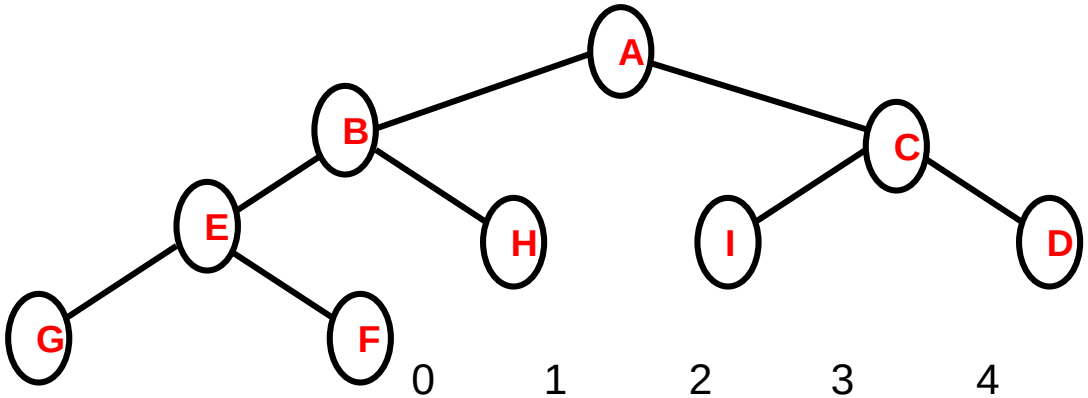
```
int Heap::Insert(char * newKey)
{
    if (NumElements == MaxElements) return FALSE;
    NumElements++; // add the new key at the last
    position
    HeapArtay[NumElementsJ = newKey;
    // re-order the heap
    int k = NumElements; int parent;
    while (k > 1) //I k has a parent
    {
        parent = k / 2;
        if IComparelk. parent) >= 0) break;
                // HeapArray[k] is in the ri9ht place
        // else exchange k and parent
        Exchange(k , parent);
        k = parent;
    }
    return TRUE;
}
```

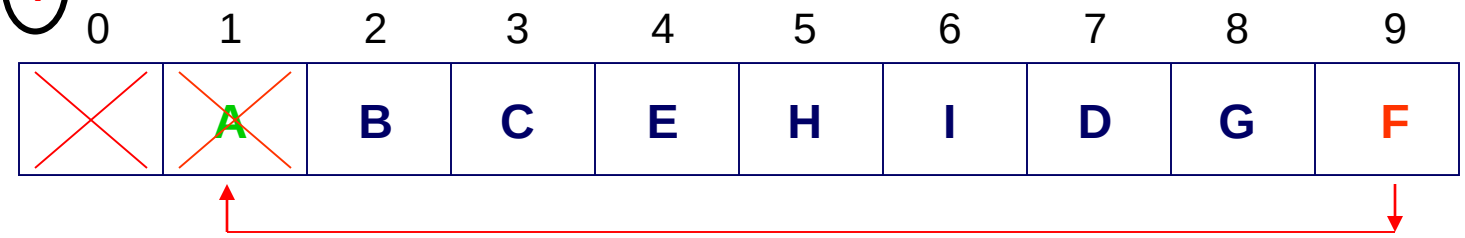
(ص 314، شکل 8.17)

مرتب سازي و ادغام فايلها

الگوريتم Remove در Heap Sort چگونه است؟

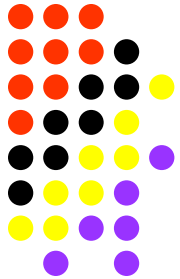
1) **کوچکترین کلید** که در گره **Root** ميباشد **خارج** ميشود.
2) بزرگترين كليد (**آخرين** گره) به گره **Root** **منتقل** ميگردد.
3) كليد آن با **کوچکترین کلید فرزند** **مقايسه** مي شود و اگر بيشتر بود جاي آن دو **تعويض** ميشود.
4) در صورت لزوم عمل (3) تا **آخر** Heap تكرار ميگردد.





طراحی وبسایت – برنامه نویسی – پروژه پایگاه داده – SQL Server
مهندسی نرم افزار UML و SSADM
۰۹۱۳۱۲۵۳۶۲۰ www.a00b.com آ صفر صفر بی دات کام

الگوریتم Remove در Heap Sort



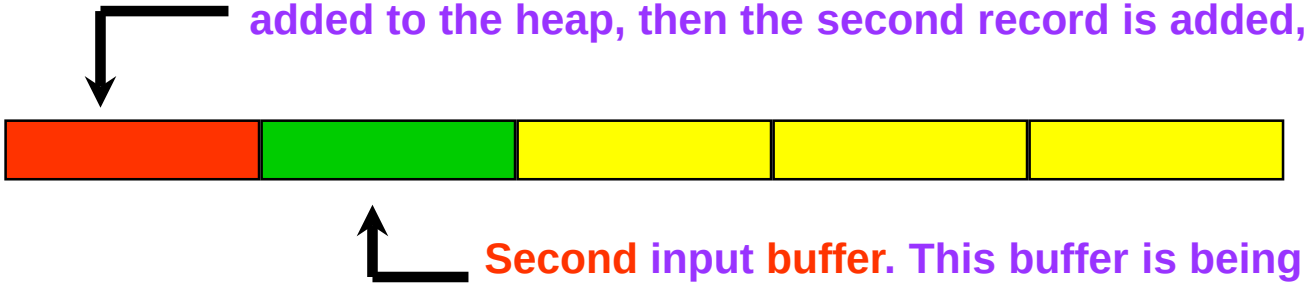
```
char * Heap::Remove()
{ //remove the smallest element, reorder the heap, and return the smallest
    element. put the smallest value into 'val' for use in return
   char * val = HeapArray[1];
   HeapArray[1] = HeapArray[NumElements]; //put largest value into root
   NumElements--; // decrease the number of elements
   // reorder the heap by exchanging and moving down
   int k = 1; // node of heap that contains the largest value
   int newK;  // node to exchange with largest value
   while (2*k <= NumElements) // k has at least one child
   {  // set newK to the index of smallest child of k
      if (Compare(2*k, 2*k+1)<0) newK = 2*k;
      else newK = 2*k+1;
      if (Compare(k, newK) < 0) break; // done if k and newK are in order
      Exchange(k, newK); // k and newK out of order
      k = newK; // continue down the tree
   }
   return val;
}
```

(ص 317 ، شکل 8.20)

مرتب سازي و ادغام فايلها

روش Overlapping در Heap Sort چگونه است؟

- ✓ اجازه مي دهد اعمال I/O و **پردازش** را **بموازات يكديگر** انجام دهيم.
- ✓ براي **شروع پردازش**، نياز نيست كه تمام داده ها در حافظه Load شده باشند (input)
- ✓ براي **شروع output** نيازي نيست كه پردازش داده ها كاملا تمام شده باشد.

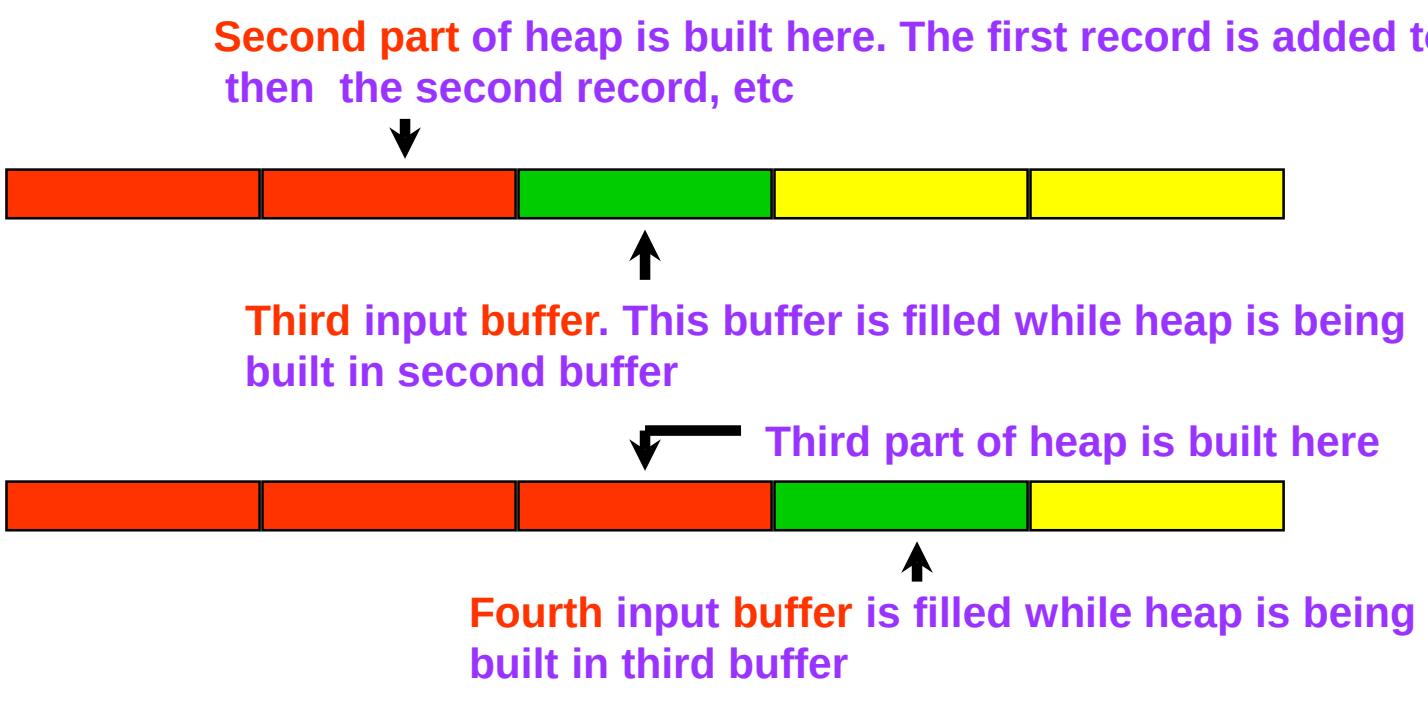


(ص 316 ، شكل 8.19)

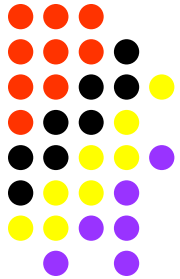
مرتب سازي و ادغام فايلها

روش Overlapping در Heap Sort چگونه است؟

- ✓ اجازه مي دهد اعمال I/O و پردازش را **بموازات** يكديگر انجام دهيم.
- ✓ براي شروع **پردازش**، نياز نيست كه تمام داده ها در حافظه **Load** شده باشند (input)
- ✓ براي شروع **output** نيازي نيست كه پردازش داده ها كاملا **تمام شده باشد**.



مرتب سازي و ادغام فايلها



روش مرتب سازي Merge-Sort چيست؟

چرا از Merge براي مرتب سازي فايلهاي بزرگ استفاده ميشود؟

مرتب سازي فايلهاي بزرگ چه مشكلاتي دارد؟

مثال:

- فايلي با مشخصات زير در نظر مي گيريم:
  - ✓ **تعداد** ركوردها: **8000000**
  - ✓ **طول** هر ركورد: **100** بايت
  - ✓ طول **كليد**: **10** بايت

- فرض كنيم كه حافظه قابل استفاده براي sort محدود به **10** مگا بايت باشد

مرتب سازي فايلهاي بزرگ

مرتب سازي فايلهاي بزرگ چه مشكلي دارد؟

مثال (ادامه...):

- **چه مشكلي براى مرتب سازى وجود دارد؟**
  - ✓ حافظه لازم براي كل فايل **800** مگابايت ميباشد
  - ✓ حافظه لازم **فقط براي كليد** هاي فايل **80** مگا بايت ميباشد
  - ✓ **مرتب سازي** روي **ديسک** بسيار **طولاني** خواهد بود.
  - ✓ **زمان لازم براي فقط يكبار خواندن** بطور Random برابر با:
    - 8000000 * 11 msec يا **88000** ثانيه
    - يا **بيش از 24** ساعت (**24:26:40**) خواهد بود

Seagate cheetah9 disk drive (seek + Rotational delay)

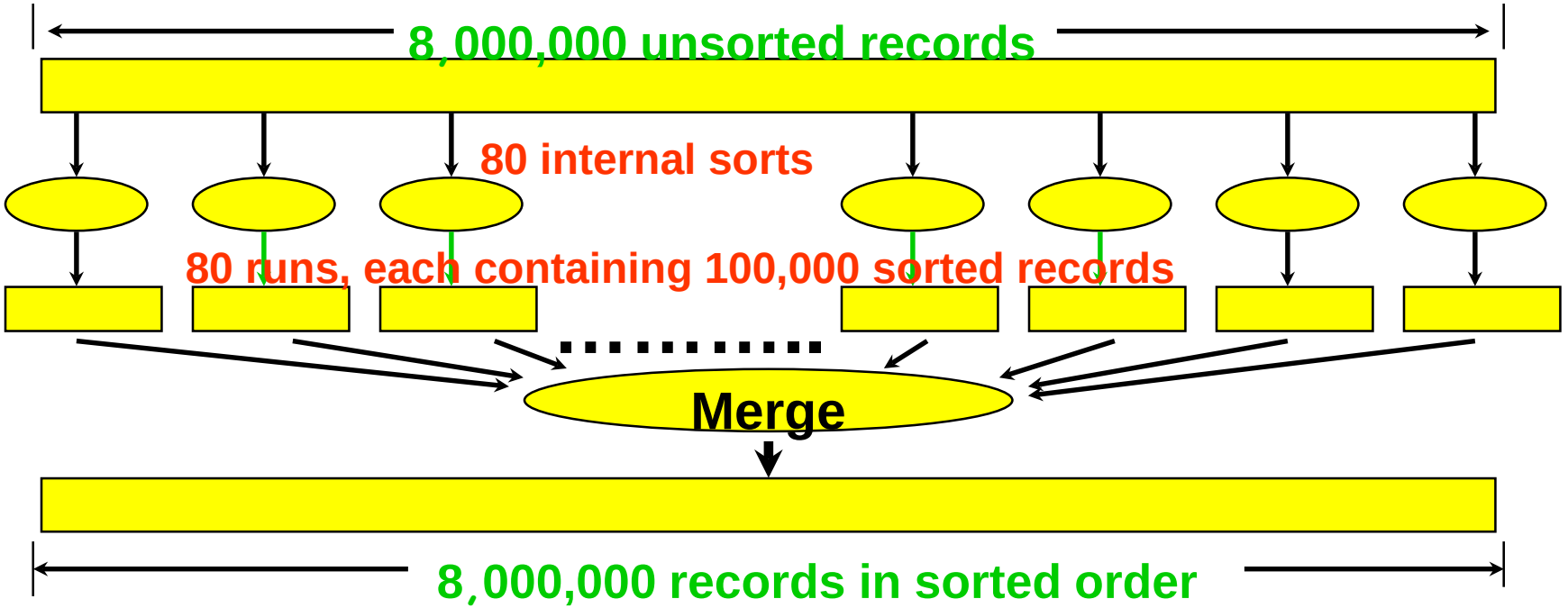
مرتب سازي فايلهاي بزرگ

مثال (ادامه...):

چه راه حلي براي اين مشكلات وجود دارد؟

1) هر تعداد ركورد كه حافظه اجازه دهد به حافظه آورده ، مرتب نموده (internal) و سپس در يک فايل كوچكتر بنويسيم.
2) عمل (1) را تا آخر فايل ادامه دهيم.
3) فايل هاي بدست آمده را با هم ادغام كنيم

(ص 320، شكل 8.21)

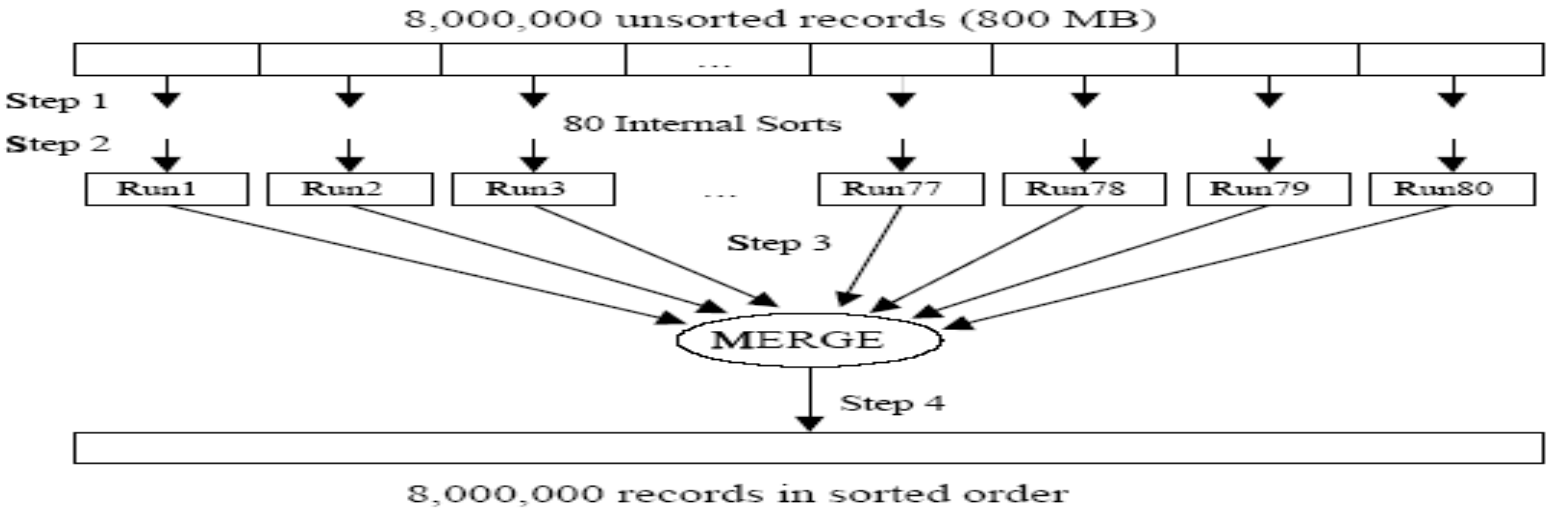


مرتب سازي فايلهاي بزرگ

مثال (ادامه...):

تعداد دفعات مرتب سازي (RUN) ؟

- ✓ ظرفيت حافظه: 10000000 بايت
- ✓ اندازه هر ركورد: 100 بايت
- ✓ تعداد ركوردها در هر دفعه (RUN): 100/10000000=100000
- ✓ تعداد كل دفعات (RUN): 100000/8000000=80



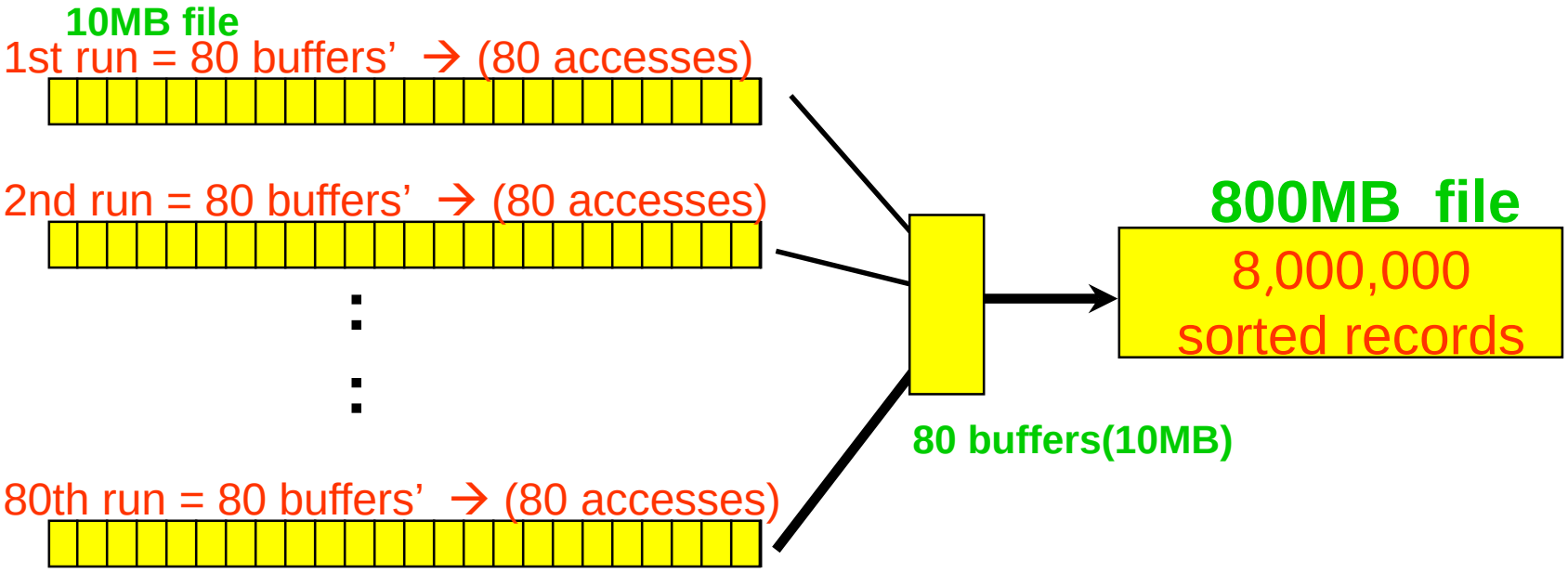
مرتب سازي فايلهاي بزرگ

مثال (ادامه...):

اندازه قطعه اي كه از هر فايل ميتوان هنگام Merge به حافظه آورد چقدر است؟

10000000/80 = **125000** بايت = **1250** ركورد

(چرا؟)



(ص 322، شكل 8.22)

مرتب سازي فايلهاي بزرگ

مثال (ادامه...):

محاسبه زمانها چگونه است؟

1) **زمان خواندن رکوردها** (براي **تشکيل فايلهاي کوچکتر**) چقدر است؟

- ✓ زمان **دسترسي** کل: 80 seeks * 11 msec = 1 sec (چرا؟)
- ✓ زمان **انتقال** کل: 800 MB @ 14500 B/msec =60 sec
- ✓ **زمان** کل **خواندن** رکوردها: 61 sec

2) **زمان نوشتن** رکوردها (در **فايلهاي کوچک**) چقدر است؟

- ✓ برابر است با **همان زمان** خواندن رکوردها: 61 sec (چرا؟)

مرتب سازي فايلهاي بزرگ

مثال (ادامه...):

محاسبه زمانها چگونه است؟

3) **زمان خواندن قطعات** فايل هاي كوچک (براي Merge) چقدر است؟

- ✓ هر فايل كوچک به **80** قطعه تقسيم مي شود (يعني **80** جستجو يا seek)
- ✓ تعداد كل جستجو (seek): **80*80=6400**
- ✓ زمان كل جستجو: **6400 *11 msec = 70 sec**
- ✓ زمان كل انتقال همان **60** ثانيه (مانند قبل)
  (چرا؟)
- ✓ زمان كل خواندن قطعات به ثانيه: **60 + 70 = 130**

مرتب سازي فايلهاي بزرگ

مثال (ادامه...):

محاسبه زمانها چگونه است؟

4) **زمان نوشتن نتايج Merge** روي ديسک چقدر است؟

- ✓ فرض کنيم که اندازه بافر I/O براي write برابر با **200000** بايت باشد
- ✓ **تعداد** کل **دفعات** جستجو : **800000000 / 200000 = 4000 seeks**
- ✓ **زمان کل جستجو: 4000 * 11= 44 sec**
- ✓ زمان کل **انتقال** همان **60** ثانيه (چرا؟)
- ✓ زمان کل **نوشتن** نتايج Merge به ثانيه: **60 + 44 = 104**

5) **زمان کل عمليات** Sort-Merge چقدر است؟

61 + 61 + 130 + 104 = 356 sec

60 + 1 60 + 1 70 + 60 44 + 60

مرتب سازي فايلهاي بزرگ

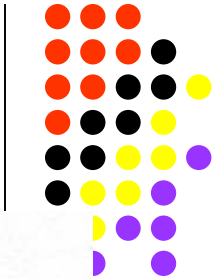
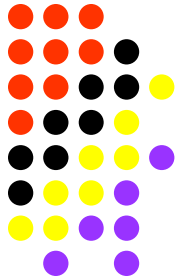


Table 8.1 Time estimates for merge sort of 80-megabyte file, assuming use of the Seagate Cheetah 9 disk drive described in Table 3.1. The total time for the sort phase (steps 1 and 2) is 14 seconds, and the total time for the merge phase is 126 seconds.

	Number of seeks	Amount transferred (megabytes)	Seek + rotation time (seconds)	Transfer time (seconds)	Total time (seconds)
Sort: reading	800	800	1	60	61
Sort: writing	800	800	1	60	61
Merge: reading	6400	800	70	60	130
Merge: writing	4000	800	44	60	104
Totals	10 560	3200	116	240	356

(ص 323، جدول 8.1)

مرتب سازي فايلهاي بزرگ



مثال (ادامه...):

اگر فايل ده برابر بزرگتر باشد چطورميشود؟

1) زمان **خواندن** رکوردها براي **Sort** در فايلهاي کوچک چقدر است؟ **610** ثانيه (چرا؟)

2) زمان **نوشتن** رکوردها براي **Sort** در فايلهاي کوچک چقدر است؟ **610** ثانيه (چرا؟)

3) زمان **نوشتن** نتايج **Merge** چقدر است؟ 1040 ثانيه (چرا؟)

4) زمان **خواندن** قطعات فايلهای کوچک براي **Merge** چقدر است؟

مرتب سازي فايلهاي بزرگ

مثال (ادامه...):

اگر فايل ده برابر بزرگتر باشد چطورميشود؟

4) زمان خواندن قطعات فايلهاي كوچک براي **Merge** چقدر است؟

- ✓ **هر فايل** كوچک به **800 قطعه** تقسيم مي شود (يعني **800** جستجو يا seek)
- ✓ **تعداد** كل **جستجو** (seek): **800*800=640000**
- ✓ **زمان** كل **جستجو**: **640000*11msec =7040 sec**
- ✓ **زمان** كل **انتقال**: **600 sec** (چرا؟)
- ✓ زمان كل خواندن قطعات براي Merge : **7640** ثانيه (بيش از دو ساعت)

(مقايسه اين زمان با حالت قبلي؟)

مرتب سازي فايلهاي بزرگ

چه روش هايي براى بهبود زمان مرتب سازي فايلهاي بزرگ وجود دارد؟

1) **چه روش هاي سخت افزاري براى بهبود زمان مرتب سازي وجود دارد؟**

- ✓ بالا بردن ظرفيت حافظه RAM
- ✓ بالا بردن تعداد ديسک ها و تقسيم فايلها روي ديسک هاي مختلف. **(چرا؟)**
  - ❑ (براي پايين آوردن seek time)
- ✓ بالا بردن تعداد كانالهاي (I/O Channels)
  - ❑ مثلا به جاي ديسک هاي Master-Slave همه Master باشند
  - ❑ و همزمان نمودن I/O روي كانالهاي مختلف (I/O overlapping)

مرتب سازي فايلهاي بزرگ

چه روش هايي براى بهبود زمان مرتب سازي فايلهاي بزرگ وجود دارد؟

1) چه روش هاي نرم افزاري بهبود زمان مرتب سازي وجود دارد؟

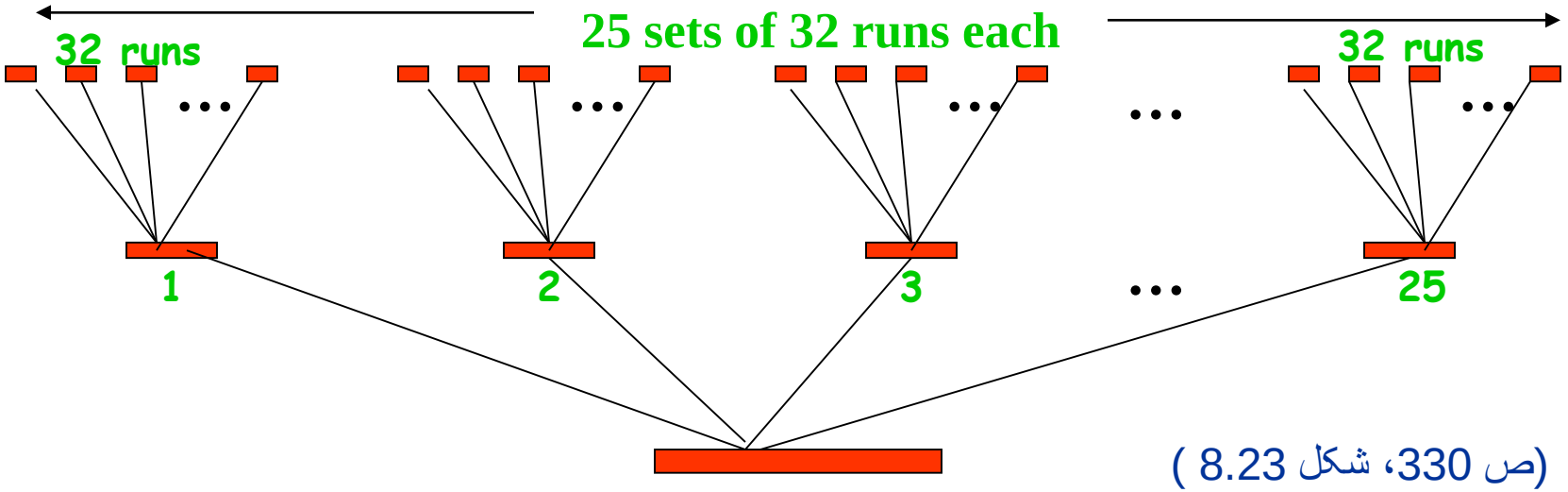
- ✓ استفاده از روش **ادغام چند مرحله اي** (Multiple-step Merge)
- ✓ **مثال:** در فايل قبلي به جاي اينكه تمام **800** فايل كوچک را يكجا با هم ادغام كنيم
  - ❑ مي توانيم آنها را به **25** دسته **32** تايي تقسيم كرده
  - ❑ و در مرحله اوّل براي هر دسته يک ادغام 32-way انجام دهيم
  - ❑ و سپس در مرحله دوم يک ادغام 25-way انجام دهيم.

مرتب سازي فايلهاي بزرگ

استفاده از روش ادغام چند مرحله اي (Multiple-step Merge)

مزايا:

- ✓ تعداد فايلها در هر مرحله كمتر
- ✓ و در نتيجه اندازه قطعات در حافظه بزرگتر مي شود
- ✓ و به تعداد كمتري seek احتياج خواهد بود
- ✓ و در نتيجه زمان لازم براي Merge كمتر خواهد شد.



(ص 330، شكل 8.23)

Two-step merge of 800 runs

مرتب سازي فايلهاي بزرگ

Table 8.3 Time estimates for two-step merge sort of 8000-megabyte file, assuming use of the Seagate Cheetah 9 disk drive described in Table 3.1. The total time is 27 minutes.

	Number of seeks	Amount transferred (megabytes)	Seek + rotation time (seconds)	Transfer time (seconds)	Total time (seconds)
1st Merge: reading	25 600	8000	282	600	882
1st Merge: writing	40 000	8000	440	600	1040
2nd Merge: reading	20 000	8000	220	600	820
2nd Merge: writing	40 000	8000	440	600	1040
Totals	125 600	32 000	1382	2400	3782

(ص 331 ، جدول 8.3)

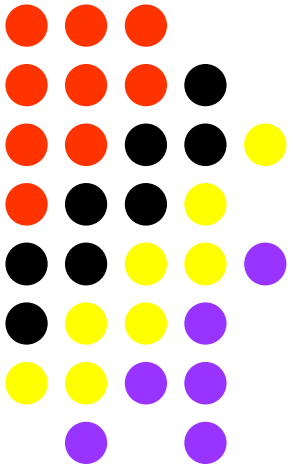
In the Name of God

Lecture 13

آشنايي با ايندكسهاي چند سطحي و درختواره اي

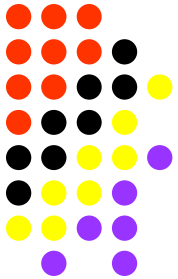
(Multi level indexing & B-Trees)

(Sections 9.1-9.6)



آشنايي با ايندكسهاي چند سطحي و درختواره اي

(Multi level indexing & B-Trees)



- نگاهداري ايندكس هاي ساده روي ديسک چه مشکلاتي بهمراه دارد؟
- انواع درخت هاي دودويي کدامند؟ (Binary Trees)
- ايندکس چند سطحي چگونه است؟ (multi level indexing)
- ايندکس B-Tree چيست؟ (Balanced Trees)

آشنايي با ايندكسهاي چند سطحي و درختواره اي

(Multi level indexing & B-Trees)

نگاهداري ايندكس هاي ساده روي ديسک چه مشكلاتي بهمراه دارد؟

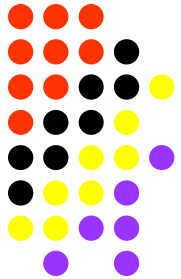
- ✓ عمل **جستجوي دودويي** روي ديسک **تعداد زيادي I/O** احتياج دارد. (چرا؟)

N	Log (N+1)
15	4
1000	~10
100000	~17
1000000	~20

- ✓ **عمليات** مربوط به **ايجاد و حذف كليدها گران** تمام مي شود. (چرا؟)
- ✓ **ايندكس** بايد **دائما بطور مرتب شده** نگهداري شود. (چرا؟)

(راه حل چيست؟)

آشنايي با ايندكسهاي چند سطحي و درختواره اي



انواع درخت هاي دودويي كدامند؟ (Binary Trees)

1) درخت دودويي ساده چيست؟ (Simple Binary Tree)

2) درخت دودويي Adel'son-Vel'skii-Landis چيست؟ (AVL Tree)

3) درخت دودويي صفحه اي چيست؟ (Paged Binary Tree)

آشنايي با ايندكسهاي چند سطحي و درختواره اي

انواع درخت هاي دودويي كدامند؟

1) درخت دودويي ساده چيست؟ (Simple Binary Tree)

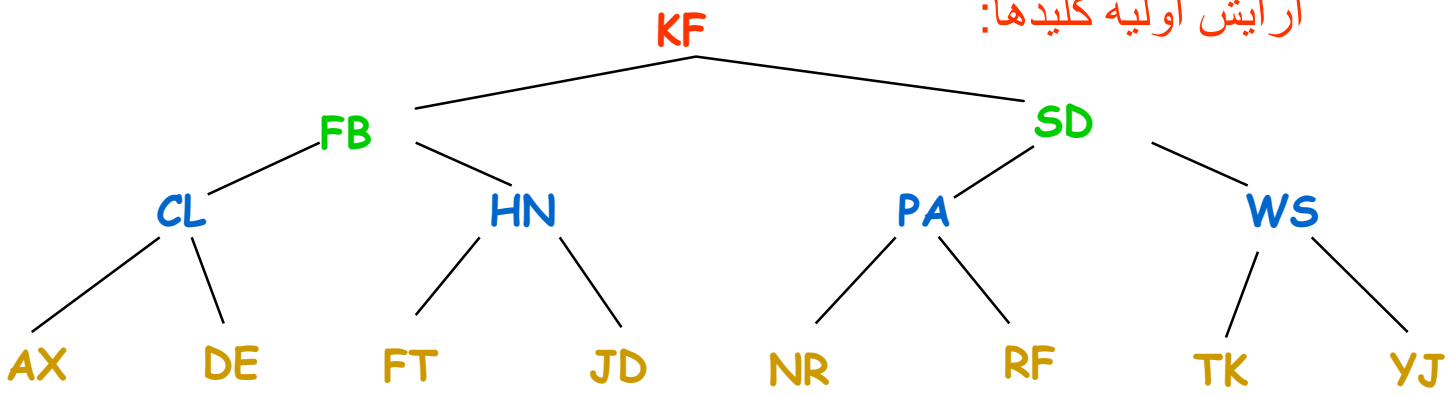
- ✓ **نوعي** نمايش درختواره اي **كليدها** ميباشد.
- ✓ بطوريكه آرايش اوليه كليدها **امكان جستجوي دودوئي** را فراهم ميسازد.
- ✓ ولي هنگام **حذف** يا **ايجاد** كليدهاي جديد، **مرتب سازي** مجدد انجام **نميشود**.
- ✓ در اينصورت **با ايجاد و حذف** كليدهاي بعدي **توازن درخت** ميتواند **بهم بخورد**.
- ✓ در حالت توازن، **هزينه جستجو** مانند **جستجوي دودوئي** ميباشد. (چرا؟)

مثال:

يک **ليست مرتب شده** از کليدها را در نظر ميگيريم:

AX, CL, DE, FB, FT, HN, JD, KF, NR, PA, RF, SD, TK, WS, YJ

آرايش اوليه کليدها:



آشنايي با ايندكسهاي چند سطحي و درختواره اي

انواع درخت هاي دودويي كدامند؟

2) درخت AVL Tree چِيست؟

- ✓ نوعي **درخت دودويي** با **ارتفاع متوازن** (Height Balanced Tree).
- ✓ كه در آن تفاوت بين **كوتاه ترين** شاخه و **بلندترين** شاخه بيش از **يک سطح** نمي باشد.
- ✓ هنگام جستجوي كليد **تعداد I/O** در **بدترين** حالت $1.44 * \log_2(n+2)$ مي باشد.

مثال:

براي جستجوي يک كليد در فايلي با **1000000** ركورد **چند I/O** لازم است؟

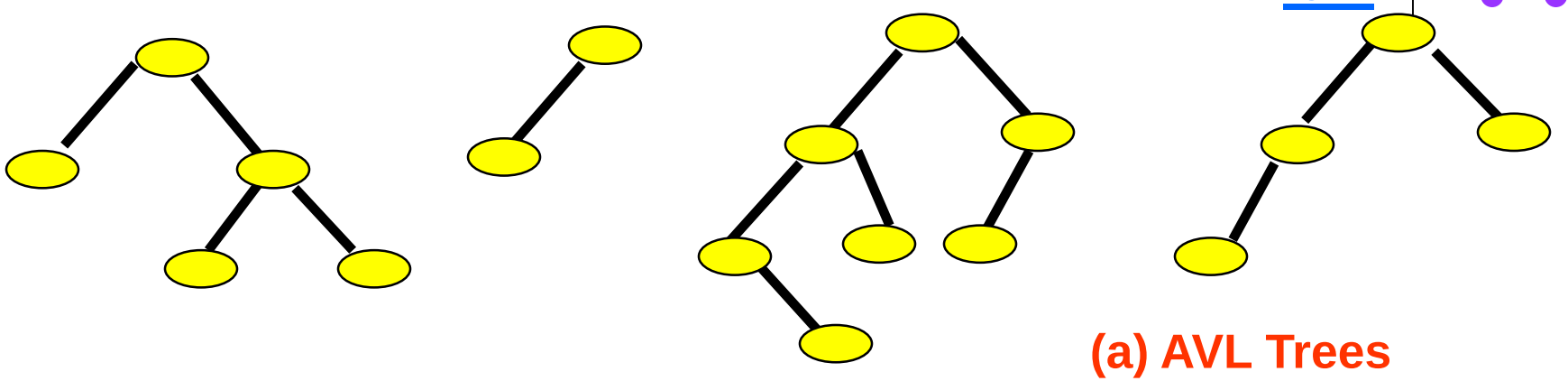
- ✓ در **بدترين** حالت بايد تعداد **29** جستجو (I/O) انجام داد!
- ✓ اين **تعداد I/O** هنوز **زياد** است!

(راه حل چيست؟)

آشنایی با ایندکسهای چند سطحی و درختواره ای

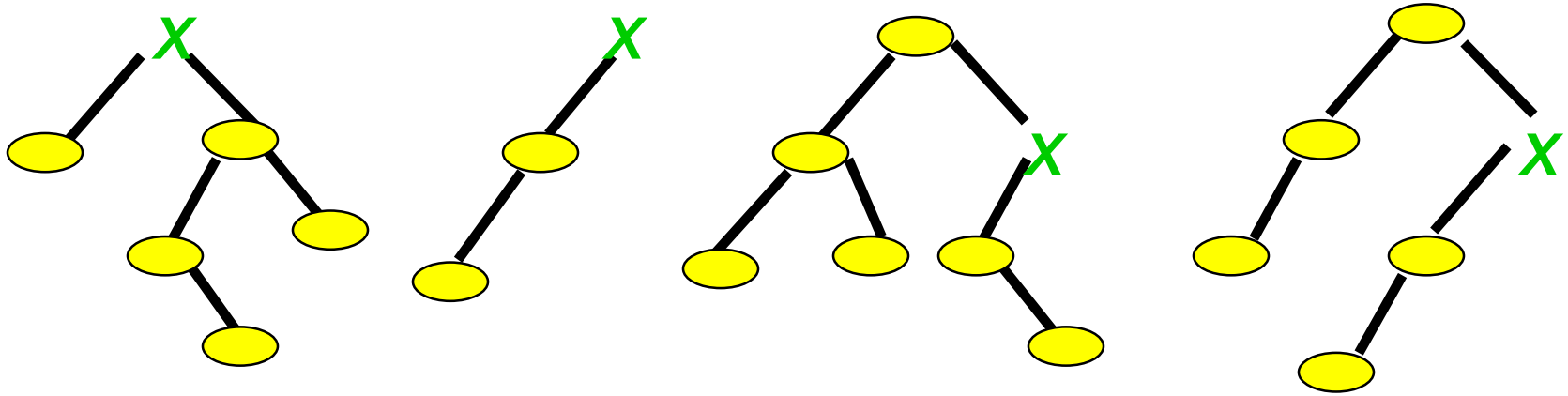
درخت AVL Tree چیست؟

مثال:



(a) AVL Trees

(b) Non - AVL Trees

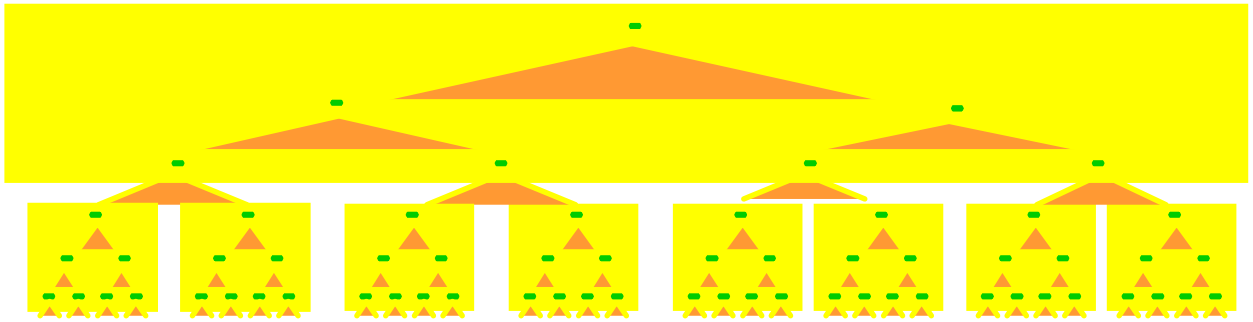


آشنايي با ايندكسهاي چند سطحي و درختواره اي

انواع درخت هاي دودويي كدامند؟

درخت Paged Binary Tree چِيست؟

- ✓ نوعي درخت **دودويي** است.
- ✓ كه **هر گره** (Node) آن شامل **چندين گره درخت دودويي** ساده ميباشد. (چرا؟)
- ✓ درچنين ايندكسي **چندين كليد** در **يک صفحه** (Page) نگهداري ميشوند.
- ✓ در اينصورت هنگام جستجوي كليد **تعداد I/O** به طرز قابل ملاحظه اي **پايين** مي آيد. (چرا؟)
- ✓ اگر **تعداد كليد** در صفحه **k** باشد، **تعداد جستجو** بين **n** **كليد** چقدرخواهد بود؟
  - ❑ در **بدترين** حالت: $\log_{k+1}(n+1)$



آشنايي با ايندكسهاي چند سطحي و درختواره اي

درخت Paged Binary Tree چِيست؟

مثال:

- ✓ يک درخت **دودويي ساده** با تعداد **n=134,217,727** کليد در نظر ميگيريم،
- ✓ **تعداد جستجوي** لازم براي يافتن **يک کليد** چقدر ميشود؟
  - ❑ در **بدترين** حالت: **27**
- ✓ **اگر** اين درخت با **k=511** کليد در **يک گره** باشد،
- ✓ **تعداد جستجوي** لازم براي يافتن **يک کليد** چقدر ميشود؟
  - ❑ در **بدترين** حالت: **3**

- ❖ اين **نتيجه خوبي** ميباشد!
- ❖ ولي حالا **مشکل اصلي**، **نگهداري** يک **paged binary tree** مي باشد.
- ❖ يعني پيدا نمودن **الگوريتم بهينه** جهت **ايجاد وحذف کليدها** با **حفظ توازن** درخت.

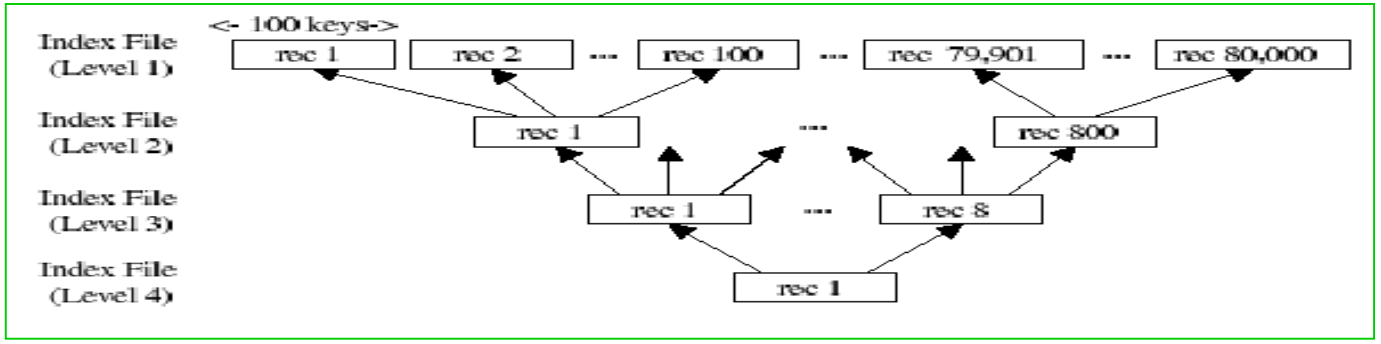
آشنايي با ايندكسهاي چند سطحي و درختواره اي

راه حل ايندكس چند سطحي چگونه است؟ (multi level indexing)

- ✓ فايلي با **8000000** **ركورد** و كليد **10** **بايتي** در نظر ميگيريم.
- ✓ اندازه فايل ايندكس آن **80** مگا بايت ميشود.
- ✓ با قرار دادن **100** **كليد** در **يک صفحه** (page) يا ركورد، تعداد **ركوردها** **80000** ميشود.
- ✓ **جستجوي يک كليد** در اين ايندكس به **16** **دسترسي** به ديسک نياز خواهد داشت. (چرا؟)

ايندكس سطح دوم چيست؟ (Second Level Index)

- ✓ حال ميتوانيم يک **ايندكس سطح دوم** براي **تسهيل دسترسي** به ايندكس اول تعريف كنيم.
- ✓ بطوريكه **هر ركورد** آن **100** **كليد** و **هر كليد** به يكي از ركوردهاي ايندكس اول **اشاره** كند.
- ✓ **تعداد ركوردهاي** اين ايندكس **800** خواهد بود.
- ✓ **جستجوي يک كليد** در ايندكس دوم به **8** **دسترسي** به ديسک نياز خواهد داشت. (چرا؟)



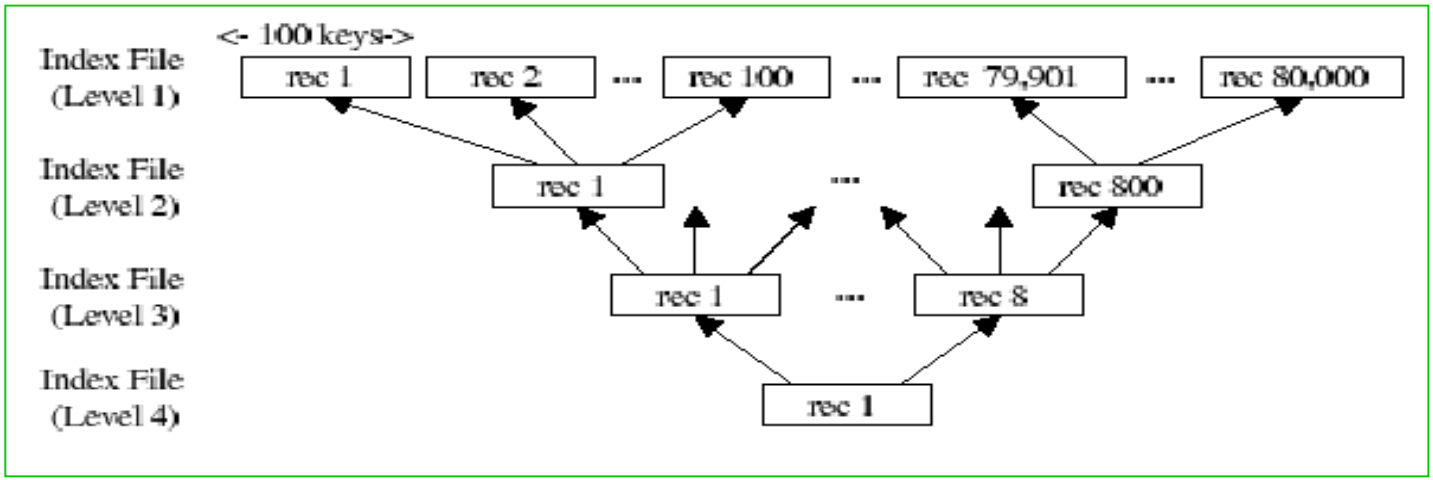
آشنايي با ايندكسهاي چند سطحي و درختواره اي

ايندكس سطح سوم چيست؟ (Third Level Index)

- ✓ حال ميتوانيم يک **ايندكس** سطح سوم براي **تسهيل دسترسي** به ايندكس دوم تعريف كنيم.
- ✓ بطوريكه هر ركورد آن **100 كليد** و **هر كليد** به يكي از ركوردهاي ايندكس دوم **اشاره** كند.
- ✓ **تعداد ركوردهاي** اين ايندكس **8** خواهد بود. (چرا؟)

ايندكس سطح چهارم چيست؟ (Fourth Level Index)

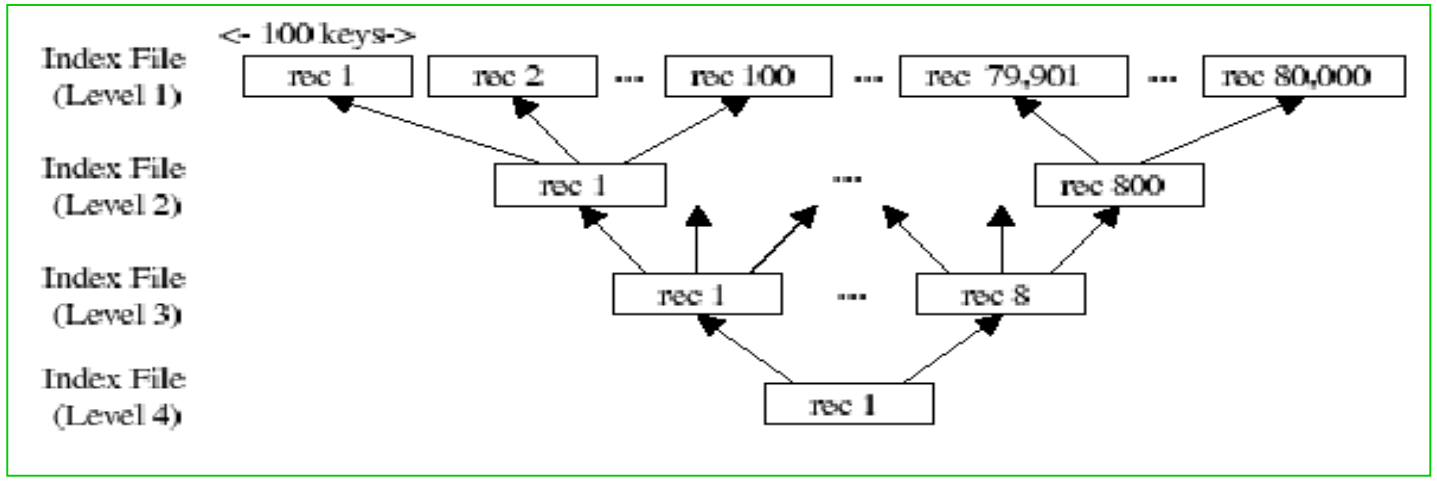
- ✓ در سطح چهارم **فقط يک ركورد** حاوي **8 كليد** خواهيم داشت.



آشنايي با ايندكسهاي چند سطحي و درختواره اي

چند نكته مهم:

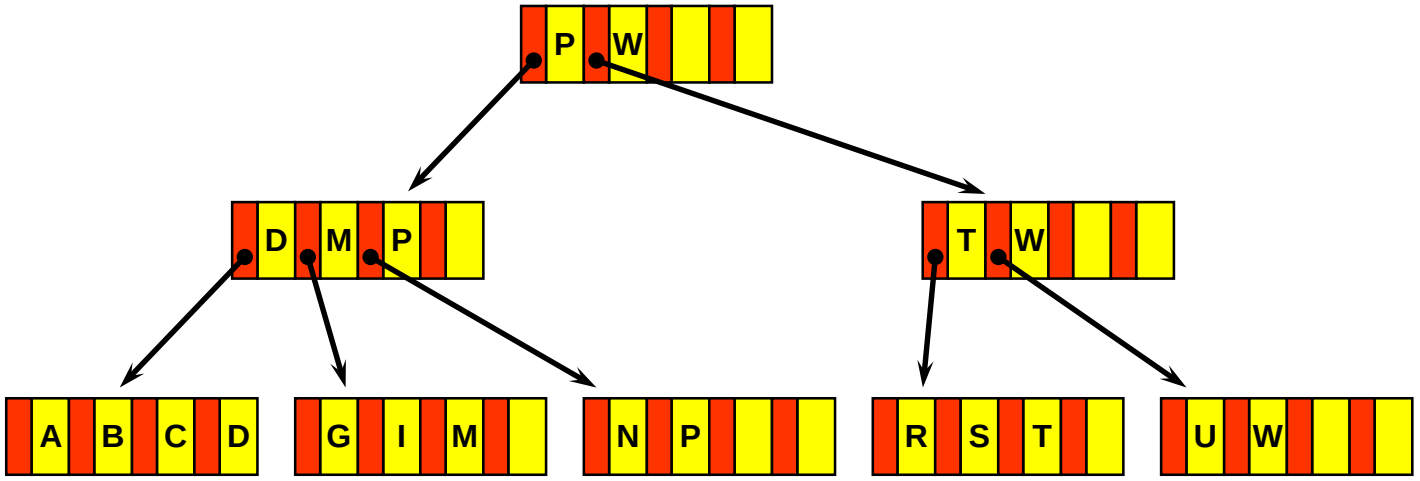
- ✓ با اين ساختار ايندكس **تعداد دسترسي** به ديسک براي يافتن يک كليد **محدود به 4** ميشود.
- ✓ **فضاي اضافي** براي نگهداري ركوردهاي ايندكس **فقط 1%** اندازه ايندكس اوليه ميباشد.
  - (در اين مثال **809 ركورد**)
- ✓ همين ساختار (**تا 4 سطح**) **ظرفيت** نگهداري تا **12 برابراين تعداد ركوردها** را خواهد داشت.
  - (يعني **100 ميليون** ركورد)



آشنايي با ايندكسهاي چند سطحي و درختواره اي

ايندكس B-Tree چيست؟ (Balanced Trees)

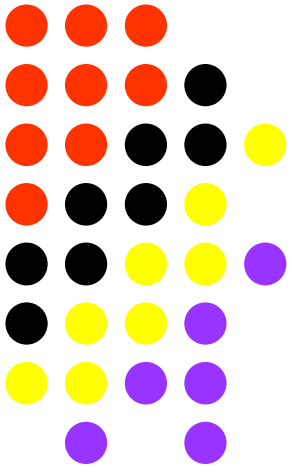
- ✓ يک نوع ايندکس **چند سطحي** (**multi level index**) با ساختاري **شبيه مثال قبل** ميباشد.
- ✓ که با يک **الگوريتم ابداعي bottom up** ساخته ميشود،
- ✓ تا **هزينه ايجاد و حذف** کليدها **ثابت** و **پايين** بماند.
- ✓ در اين روش **هر کليد جديد** در يکي از گره هاي **سطح 1** وارد شده،
- ✓ و سپس در صورت لزوم **سطوح دوم تا ريشه (root) به روز** ميشوند.



Lecture 14

B-trees, B*trees and Virtual B-trees

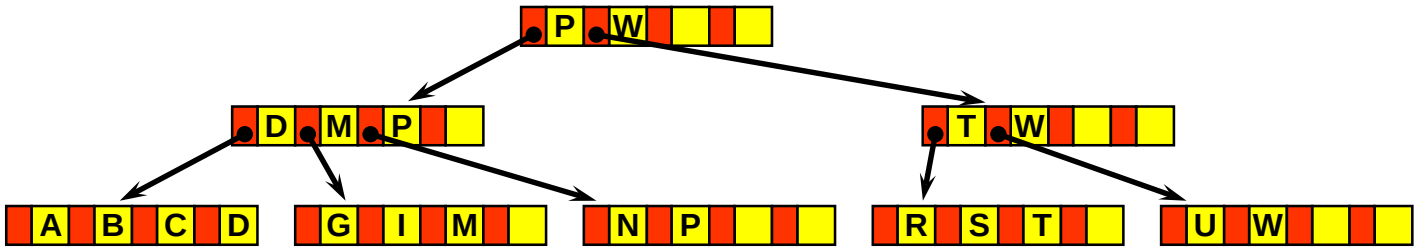
(Sections 9.8-9.15)



آشنایی با ایندکسهای B-Tree

ساختاریک ایندکس B-Tree چگونه است؟

- ✓ **هر نود** میتواند یک رکورد با **تعداد ثابتي کلید** (مثلا **100**) باشد.
- ✓ **تعداد کلید** در **هر گره** **بین نصف تا تمام ظرفیت** آن میباشد.
- ✓ براي **اضافه نمودن** کلید به نودي که **ظرفیت** آن **تکمیل** شده:
  - ❑ آن نود را به **2 نود جدید** تقسیم میکنند،
  - ❑ و **بزرگترین کلید** يکي از **2** نود جدید به **سطح بالاتر** ارتقا پیدا میکند.
- ✓ **حذف نمودن** کلید از نودي که **ظرفیت** آن به **مینیمم** رسیده است:
  - ❑ ممکن است باعث **ادغام نود با نود مجاور** یا **متوازن نمودن** کلیدها بین آنها گردد،
  - ❑ و پس از آن، **نود سطح بالاتر** نیز باید به روز شود.



جستجوی کلید در ایندکس B-Tree

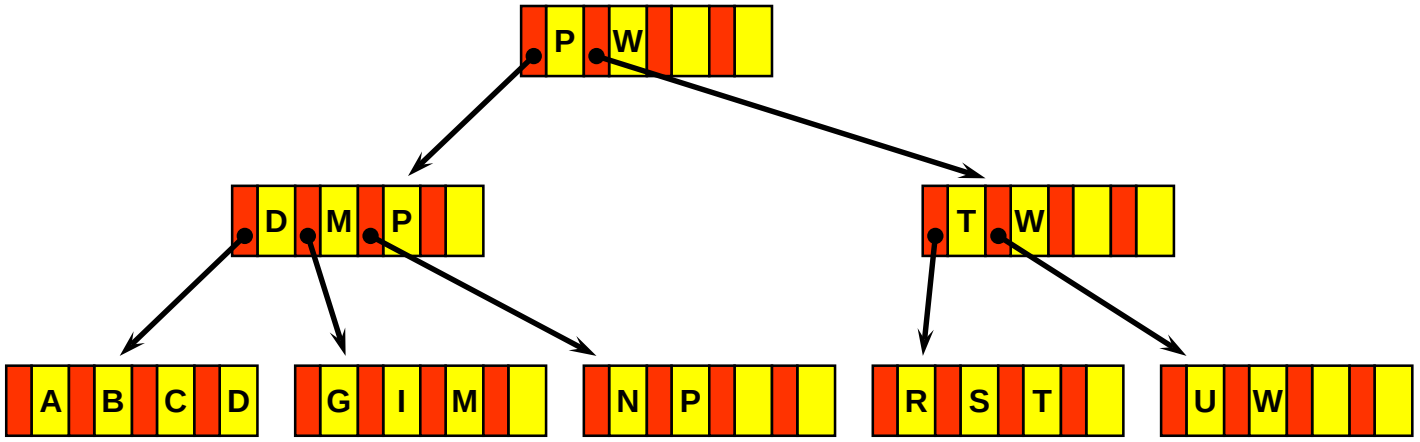
روش جستجوي کليد دريک ايندکس B-Tree چيست؟

1) براي **جستجوي** کليد **k** ، بايستي اوّل نود **ريشه** (**Root**) به حافظه آورده شود.

2) در بين کليدهاي اين نود، کليد K_iجستجو ميشود ، بطوريکه:
   - يا K_iاولين کليد در نود و $k \leq K_i$ باشد
   - يا $K_i - 1 < k \leq K_i$ باشد.

3) در صورت يافتن K_i ، نود مربوطه به حافظه آورده ميشود،

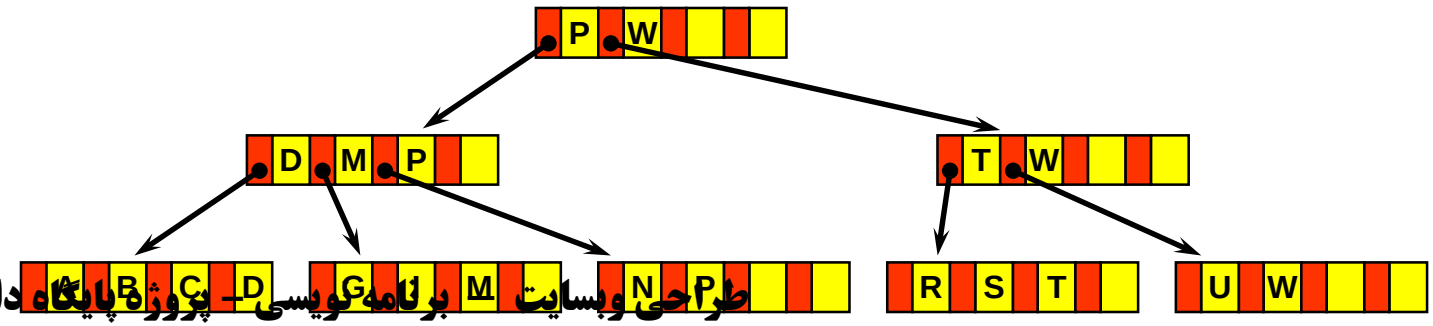
   - و **عمل** **2** **تکرار**مي گردد تا به نود برگ (**Leave**) برسيم و آدرس داده مورد نظر پيدا شود.



ایجاد کلید در ایندکس B-Tree

روش ایجاد کلید (Insert) در B-Tree چگونه است؟

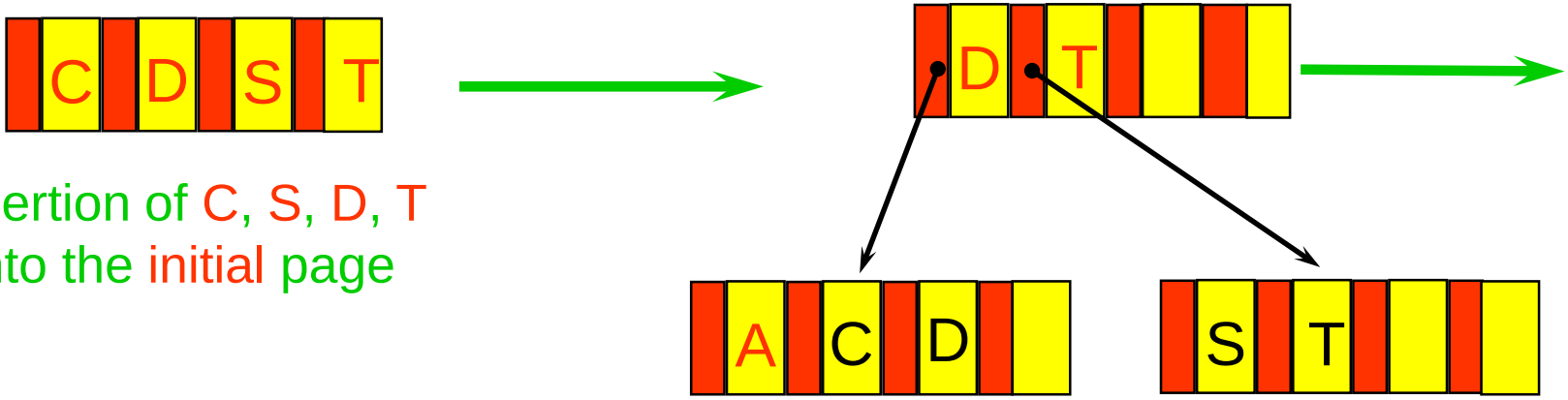
1) با روش قبل **نود برگ** (n) مربوط به کلید **k** جستجو میشود.
2) در صورت **وجود فضاي لازم**:
   - کلید **k** به نود **اضافه** میشود،
   - و اگر **k** از **بزرگترین کلید** موجود در نود **بزرگتر** باشد، **نود سطح بالاتر** نیز بروز میشود.
3) در صورت **پر بودن نود**:
   - بایستي آن را به دو نود (**n**) و (**n+1**) تقسیم نمود،
   - کلید **k** را در **یکي از دو نود** جدید اضافه نمود،
   - و سپس **نود سطح بالاتر** را نیز بروز نمود،
   - که خود ممکن است باعث **تکرار اعمال 2 و 3** تا **ریشه** بشود.



مثال ایجاد کلید در ایندکس B-Tree

- Input Sequence:

C S D T A *M P I B W N G U R K E H O L J Y Q Z F X V*



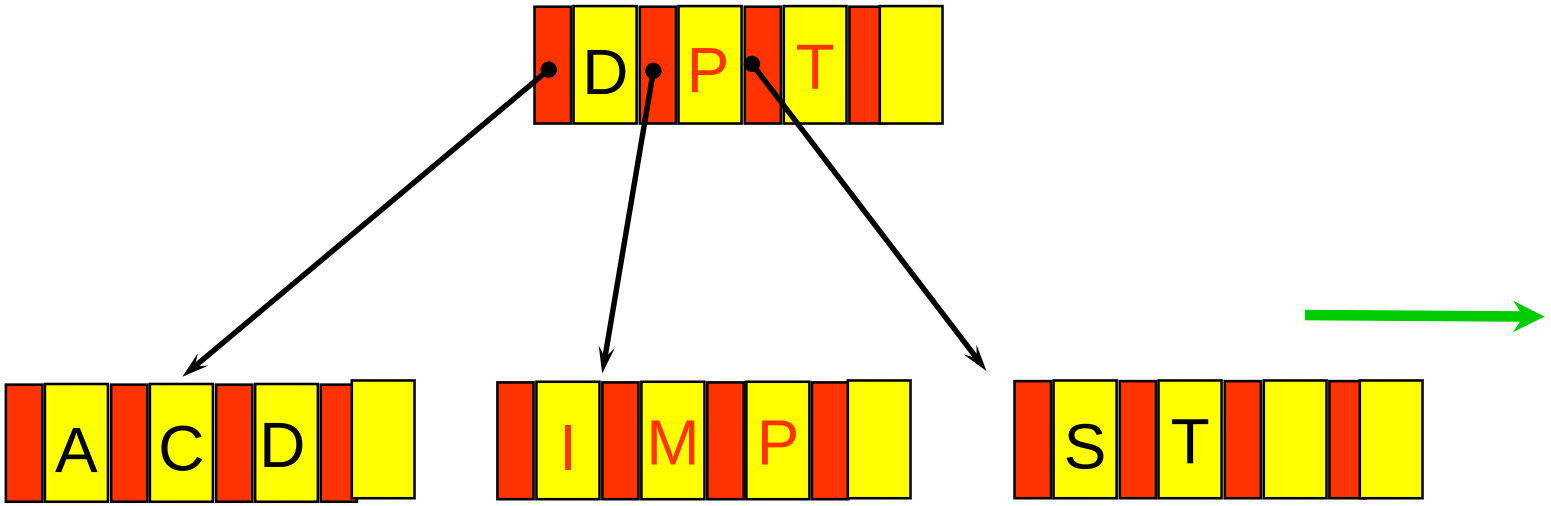
Insertion of C, S, D, T into the initial page

Insertion of A causes node to split and the largest key in each leaf node(D and T)to be placed in the root node

مثال ایجاد کلید در ایندکس B-Tree

- Input Sequence:

*C S D T A **M P I** B W N G U R K E H O L J Y Q Z F X V*

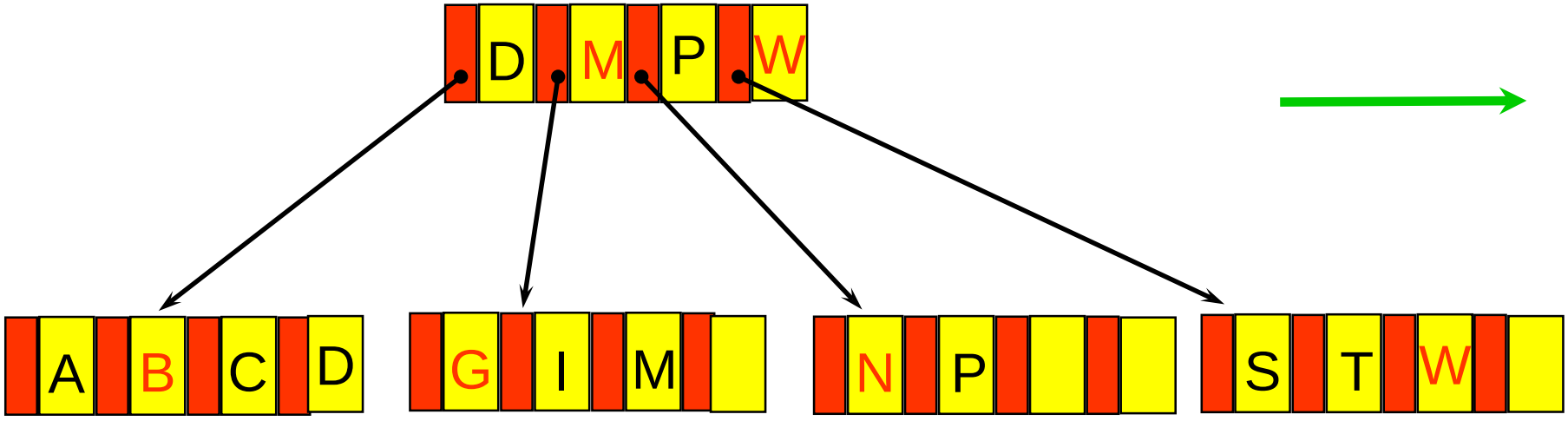


M and **P** are inserted into the rightmost leaf node,
then insertion of **I** causes it to split

مثال ایجاد کلید در ایندکس B-Tree

- Input Sequence:

*C S D T A M P I **B W N G** U R K E H O L J Y Q Z F X V*

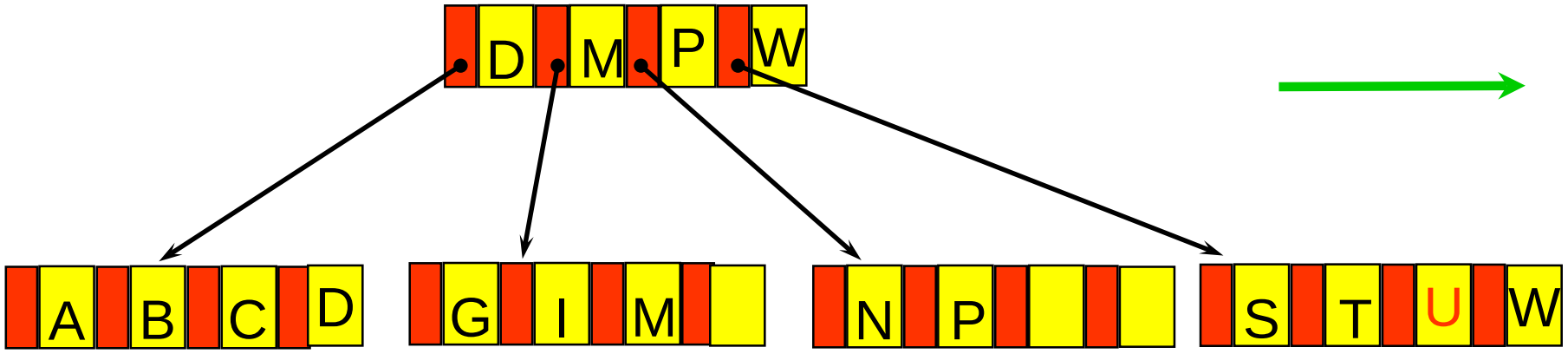


Insertions of B,W,N, and G into leaf nodes causes another split and the root is now full

مثال ایجاد کلید در ایندکس B-Tree

- Input Sequence:

*C S D T A M P I B W N G **U** R K E H O L J Y Q Z F X V*

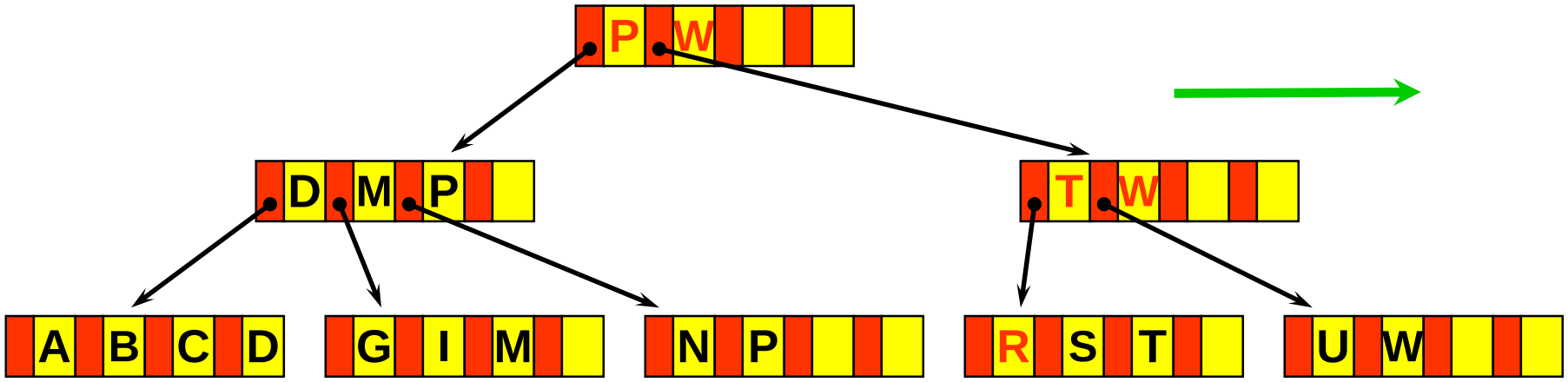


Insertion of U proceeds without incident, but R would have to be inserted into the rightmost leaf, which is full

مثال ایجاد کلید در ایندکس B-Tree

- Input Sequence:

C S D T A M P I B W N G U R K E H O L J Y Q Z F X V

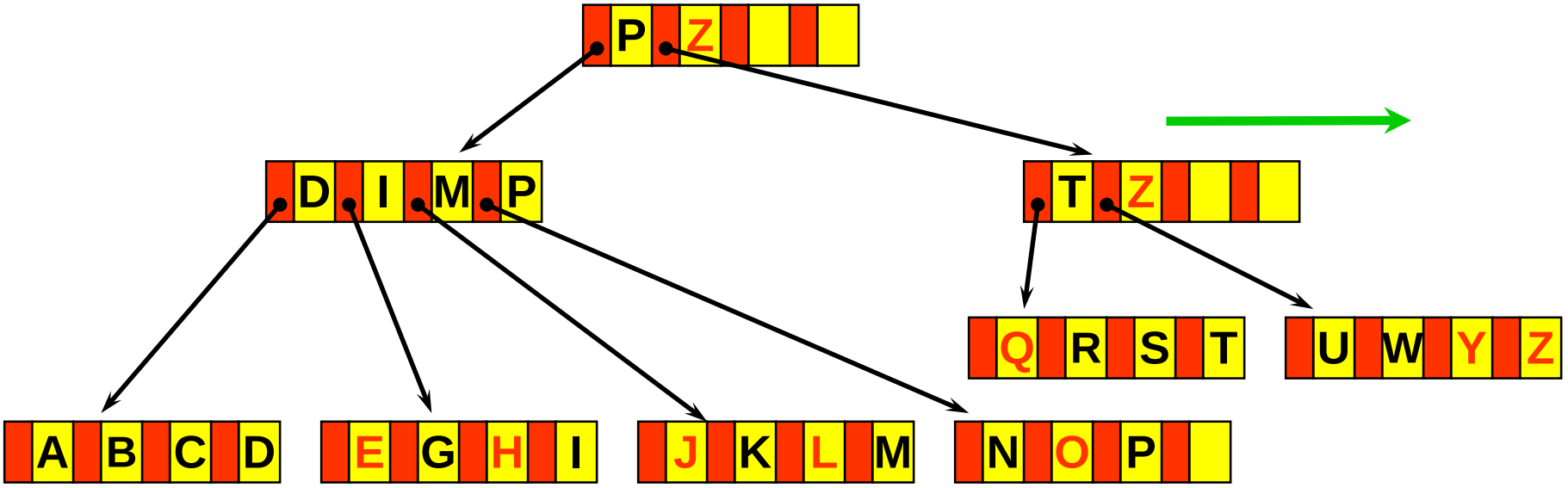


Insertion of R causes the rightmost leaf node to split, insertion into the root causes the **root to split** and the tree grows to level three

مثال ایجاد کلید در ایندکس B-Tree

- Input Sequence:

*C S D T A M P I B W N G U R K **E H O L J Y Q Z** F X V*

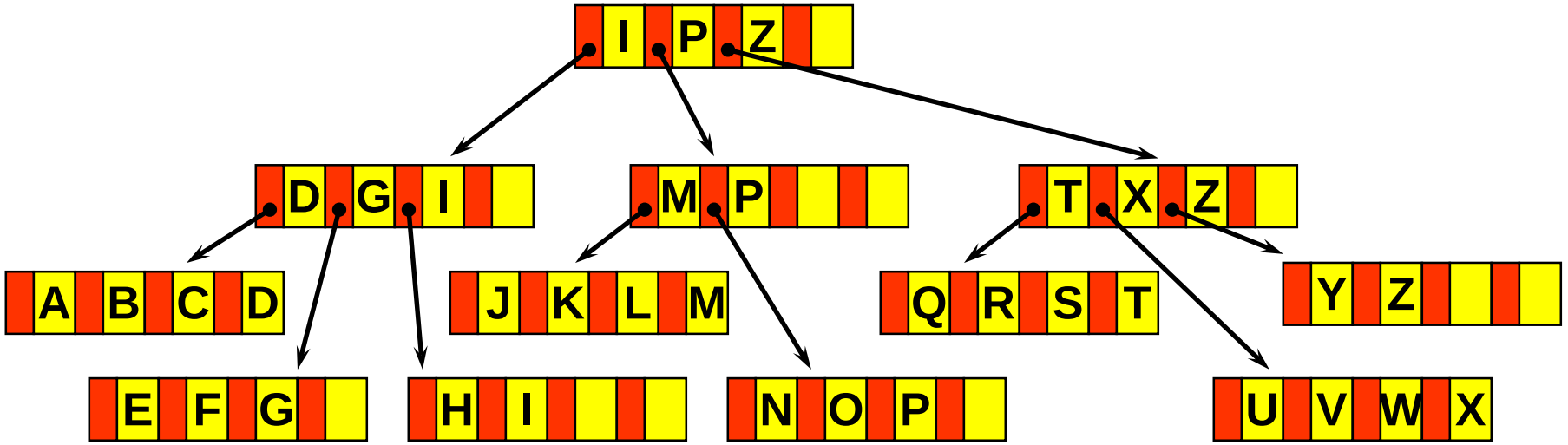


Insertions of K,E,H,O,L,J,Y,Q, and Z, continue with another node split

مثال ایجاد کلید در ایندکس B-Tree

- Input Sequence:

*C S D T A M P I B W N G U R K E H O L J Y Q Z **F X V***



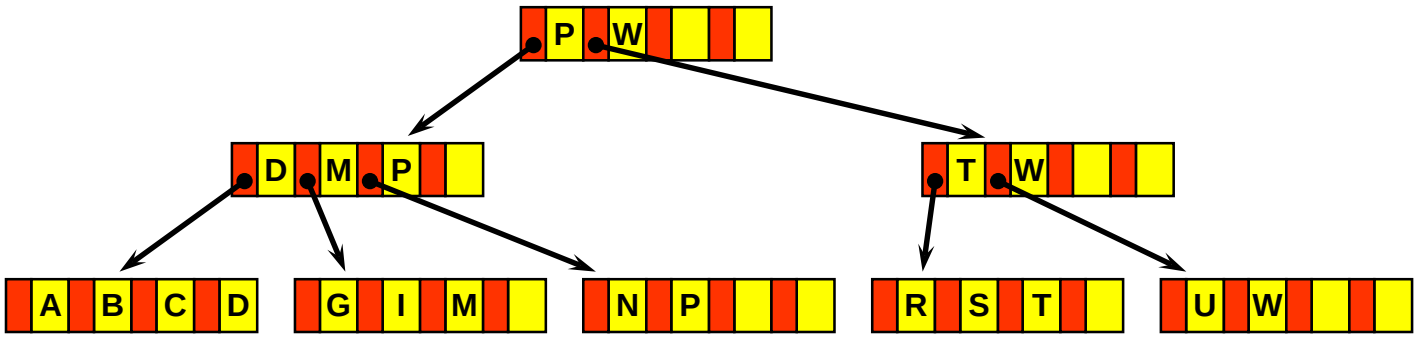
Insertions of F, X, and V finish the insertion of the alphabet

خواص ایندکس B-Tree

ایندکس B-Tree بطور رسمي چه خواصي دارد؟ (Formal definition)

یک ایندکس B-Tree با درجه m (Order) داراي خواص زير مي باشد:

- ✓ هر نود ماکزیمم **m فرزند** دارد.
- ✓ هر نود غیر از ریشه (**Root**) و برگها (**Leaves**) لااقل **m/2 فرزند** دارد.
- ✓ نود **ریشه** لااقل **دو فرزند** دارد مگر هنگامي که ریشه همان برگ باشد.
- ✓ تمام **برگها** (**Leaves**) در **یک سطح** قرار دارند.
- ✓ **مجموعه برگها** یک ایندکس **کامل و مرتب** شده از کلیدها را تشکیل میدهد.



خواص ایندکس B-Tree

تعداد جستجودر B-Tree در بدترین حالت؟ (Worst-Case Search)

- ✓ **تعداد جستجوی** لازم در B-Tree براي يافتن يک کليد بستگي به **تعداد سطوح** دارد.
- ✓ در بدترین حالت **فقط نيمي از ظرفيت** هر نود استفاده شده و **تعداد سطوح ماکزیمم** میباشد.
- ✓ در یک B-Tree با درجه m، رابطه **تعداد کلید N** و **تعداد سطوح d** در بدترین حالت برابر است با:

$$N \geq (2*[m/2]^{d-1}) \rightarrow d \leq (1 + \log_{m/2}(N/2))$$

مثال:

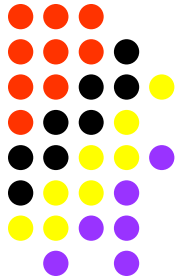
اگر تعداد کلید **N=1000000** و B-Tree از درجه **m=512** باشد:

$$d \leq 1 + \log_{256} 500000 \rightarrow d \leq 3.37$$

بنابر این **تعداد سطوح ماکزیمم 3** میباشد

طراحی وبسایت – برنامه نویسی – پروژه پایگاه داده – SQL Server
مهندسی نرم افزار UML و SSADM
۰۹۱۳۱۲۵۳۶۲۰ www.a00b.com آ صفر صفر بی دات کام

حذف کلید در ایندکس B-Tree



روش حذف کلید (Deletion) در B-Tree چگونه است؟

براي حذف کلید k از نود (n):

1) اگر نود (n) بیش از مینیمم ظرفیت مجاز کلید داشته باشد:
   - کلید k حذف شده،
   - و در صورتیکه این کلید بزرگترین کلید نود (n) باشد نود سطح بالايي نیز بایستي بروز شود.

2) Merge: اگر نود (n) به مینیمم ظرفیت مجاز کلیدها رسیده باشد و فضاي موجود در نود مجاور آن (Sibling) اجازه بدهد:
   - هر دو نود با یکدیگر ادغام شده،
   - کلید k حذف مي شود،
   - و نود سطح بالايي نیز بروز میگردد. (چرا؟)

طراحی وبسایت – برنامه نویسی – پروژه پایگاه داده – SQL Server
مهندسی نرم افزار UML و SSADM
۰۹۱۳۱۲۵۳۶۲۰ www.a00b.comآ صفر صفر بی دات کام

حذف کلید در ایندکس B-Tree

روش حذف کلید (Deletion) در B-Tree چگونه است؟

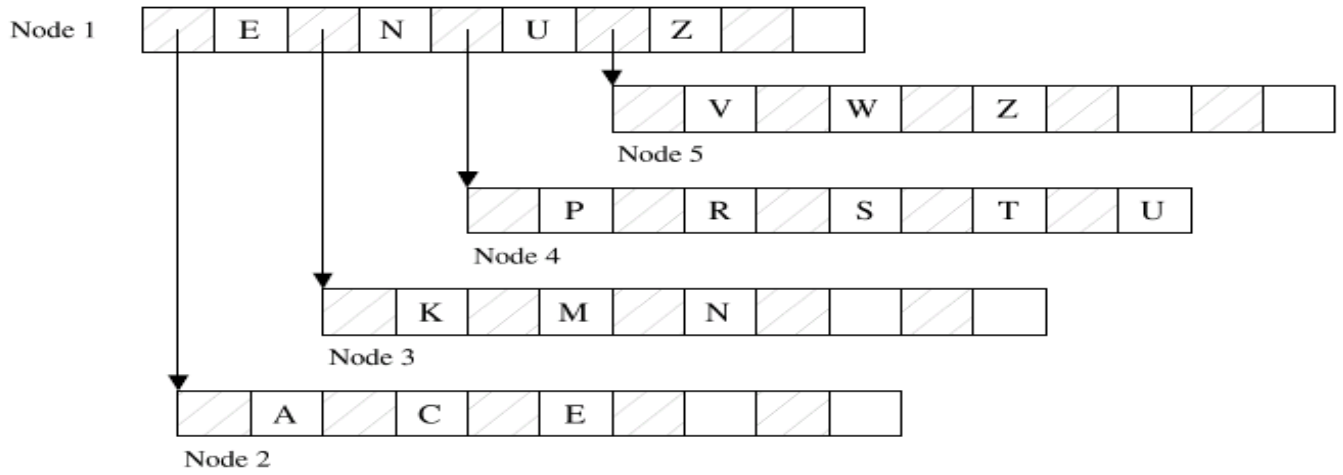
براي حذف کلید k از نود (n) (ادامه...):

3) Redistribute: اگر نود (n) به مینیمم ظرفیت مجاز رسیده باشد و يكي از نودهاي مجاورکه نود پدر آنها يكي باشد (Sibling) بیش از مینیمم مجاز کلید داشته باشد:

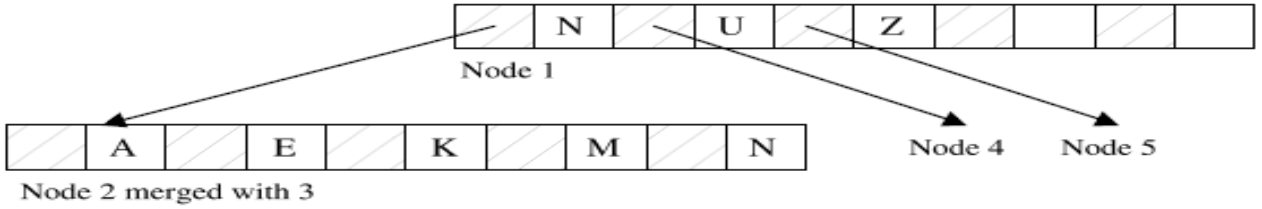
- کلیدها بین دو نود تقسیم مي شوند،
- سپس کلید k حذف شده،
- و پس از آن، نود سطح بالاتر نیز بروز میگردد.

مثال حذف کلید در ایندکس B-Tree

مثال(1):

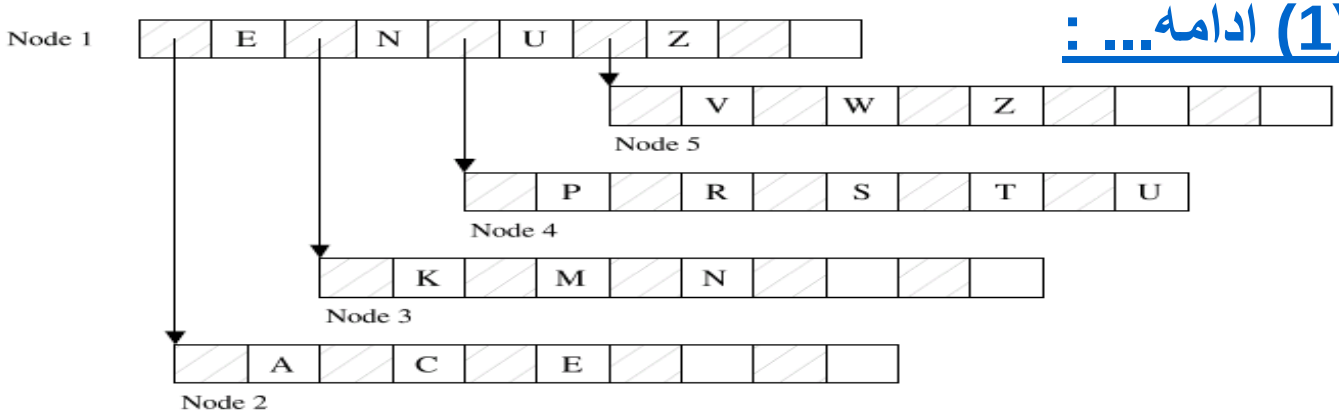


- Deleting "**T**" falls into case 1
- Deleting "**U**" falls into case 1
- Deleting "**C**" falls into case 2: Merge node 2 with node 3

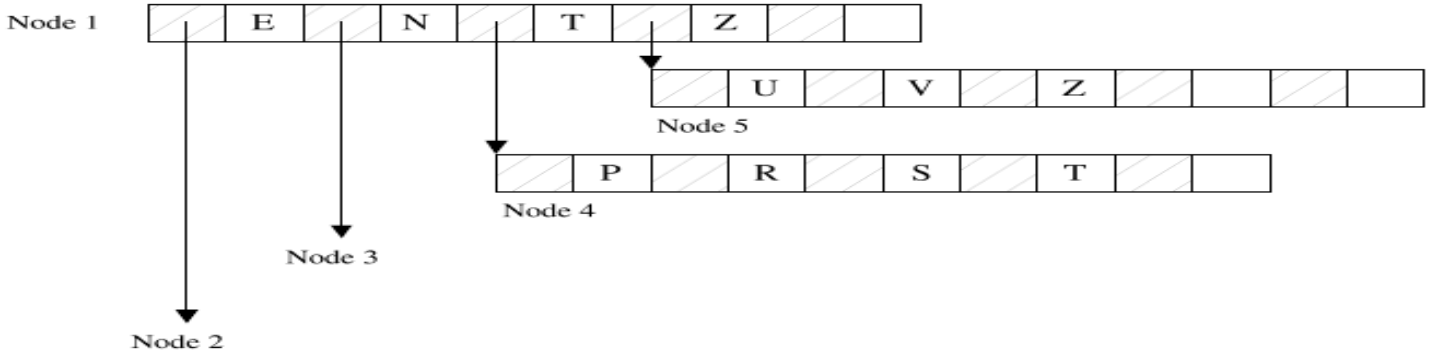


مثال حذف کلید در ایندکس B-Tree

مثال(1) ادامه... :

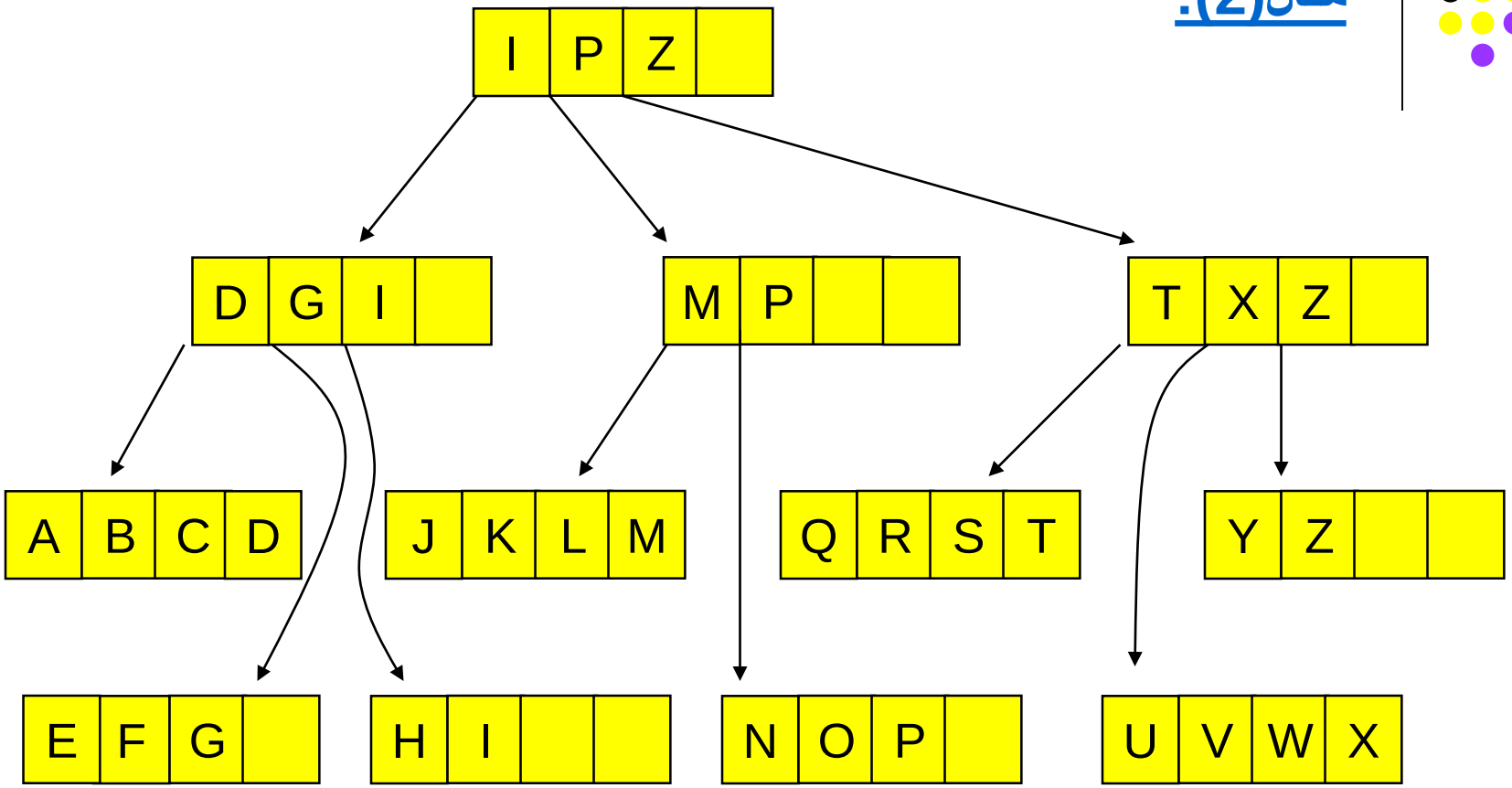


- Deleting "W" falls into case 3:
  - Redistribute keys between node 4 and node 5
- Deleting "M" allows for two possibilities: case 3 or 1
  - Merge Node 3 with Node 2; or
  - Redistribute keys between Node 3 and Node 4



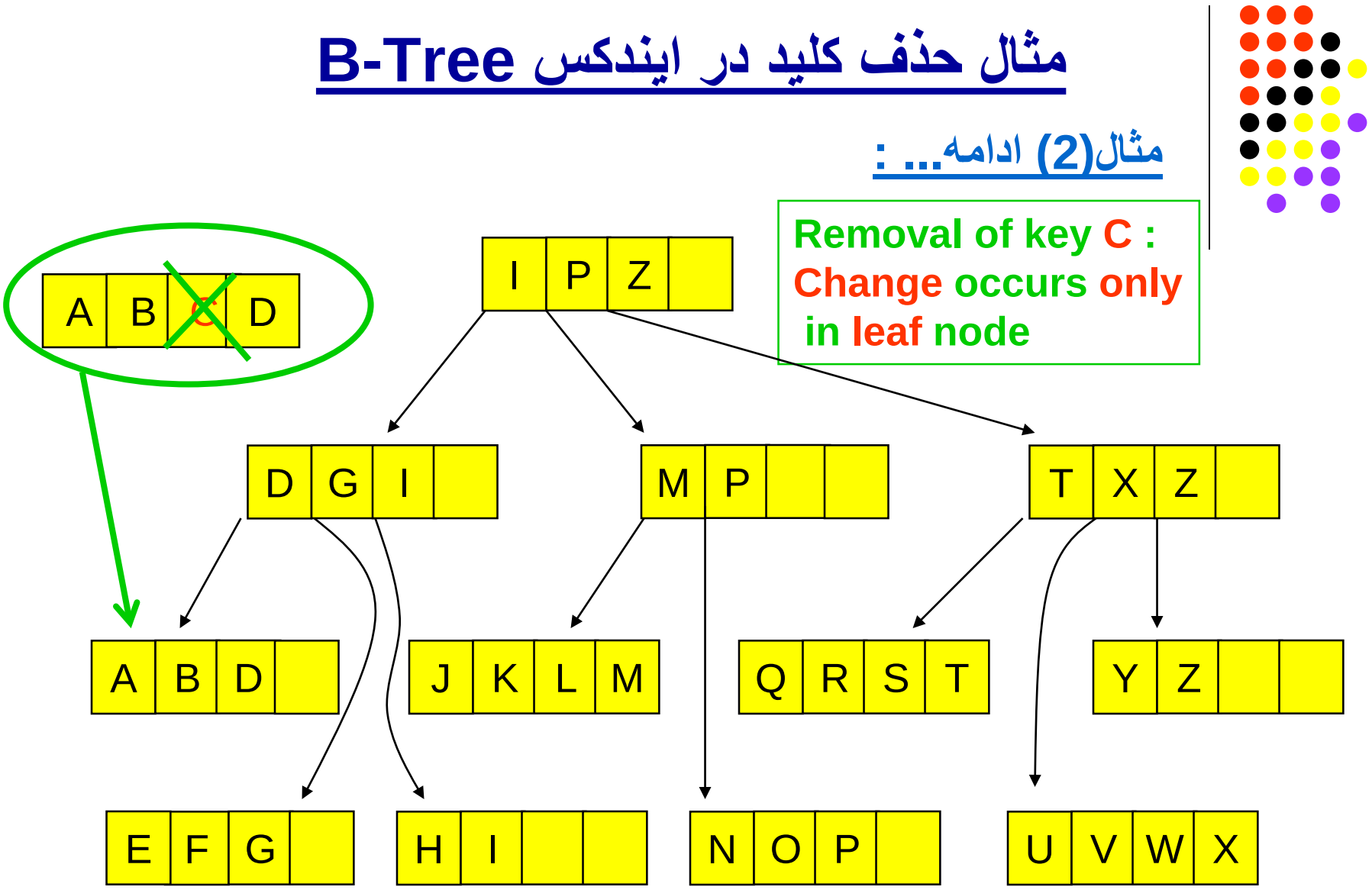
مثال حذف کلید در ایندکس B-Tree

مثال(2):



مثال حذف کلید در ایندکس B-Tree

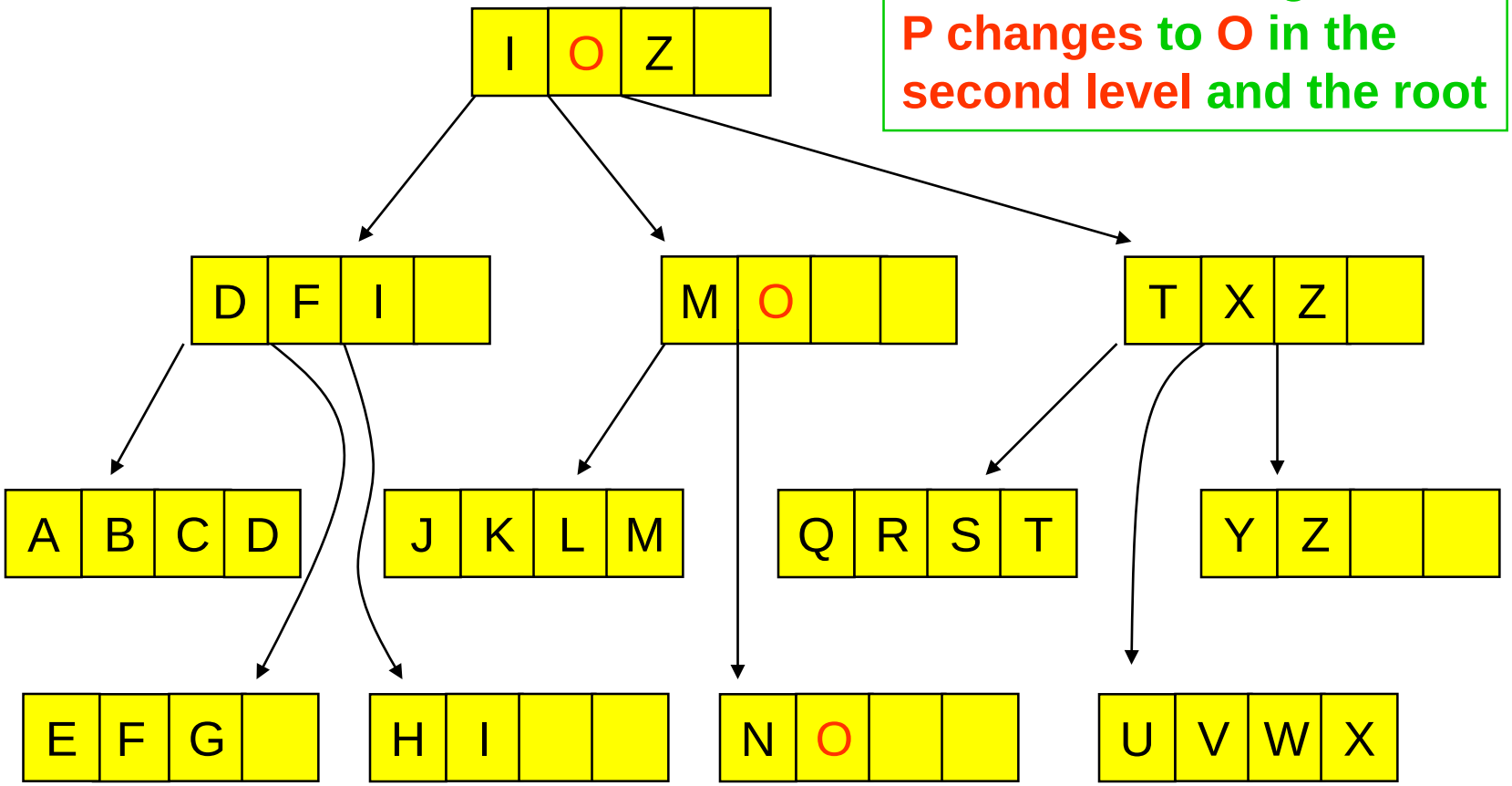
مثال(2) ادامه... :



مثال حذف کلید در ایندکس B-Tree

مثال(2) ادامه... :

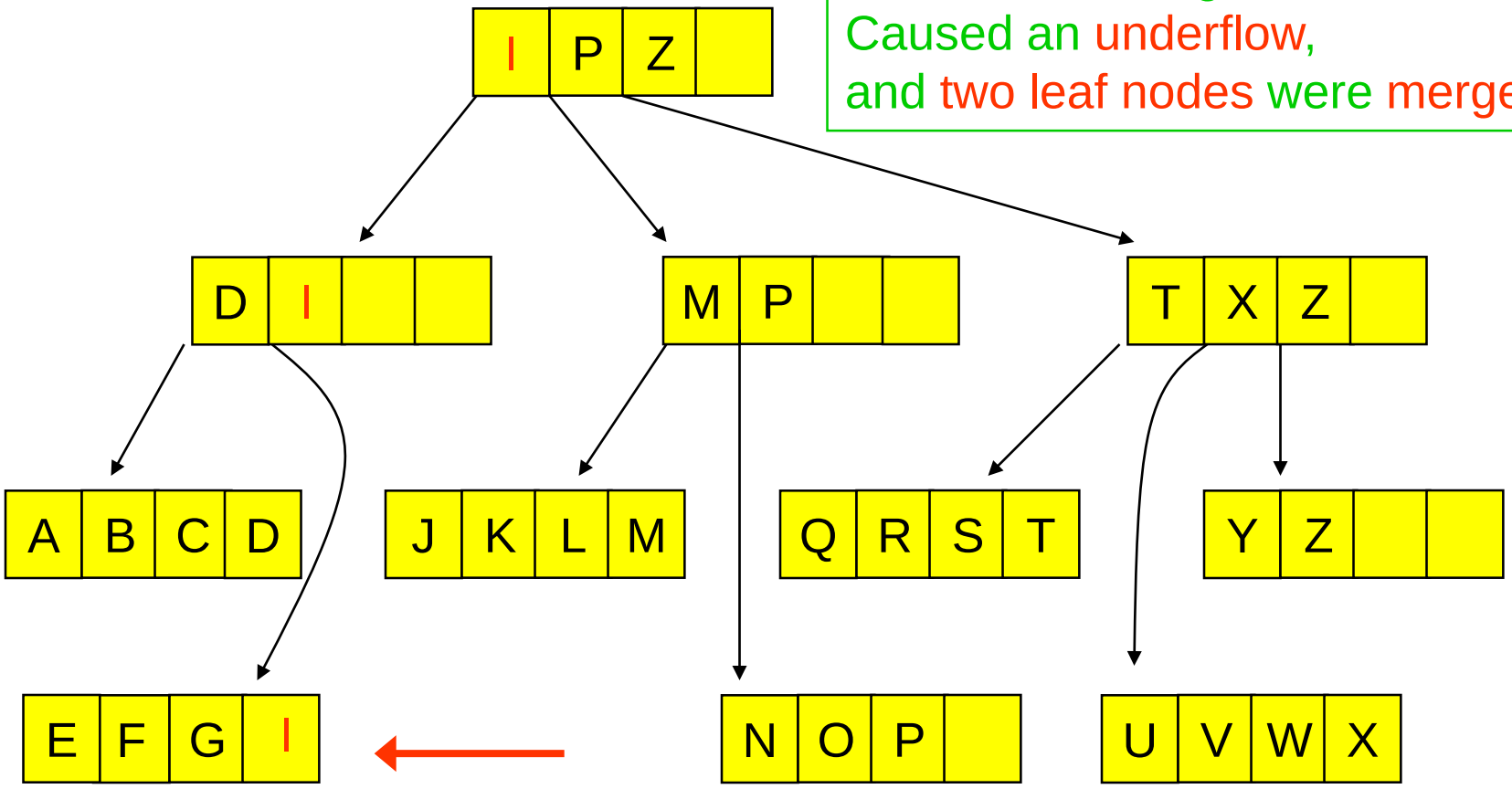
Result of deleting P :
P changes to O in the second level and the root



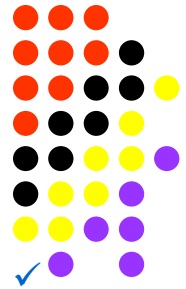
مثال حذف کلید در ایندکس B-Tree

مثال(2) ادامه... :

Result of deleting H:
Caused an underflow,
and two leaf nodes were merged



توزیع مجدد کلیدها در B-Tree



کاربردهای توزیع مجدد کلیدها (Redistribution) در B-Tree کدامند؟

- ✓ توزیع مجدد کلیدها بین دو نود مجاورکه از یک نود پدر باشند (sibling) انجام پذیراست.
- ✓ هنگام حذف یا ایجاد کلید باعث صرفه جویي در I/O یا به تاخیر اندختن آن میشود.
- ✓ هنگام ایجاد کلید، اگر تعداد کلیدهاي نود (n) به ماکزیمم رسیده باشد (Key overflow) :
  - ❑ در صورتي که یکي از نودهاي مجاور فضاي لازم را داشته باشد،
  - ❑ با توزیع مجدد کلیدها بین دو نود از شکسته شدن نود (n) و ایجاد نود جدید جلوگیري میشود.
- ✓ هنگام حذف، اگر تعداد کلیدهاي نود (n) به مینیمم مجاز رسیده باشد (Key underflow) :
  - ❑ در صورتي که یکي از نودهاي مجاور کلید اضافي داشته باشد،
  - ❑ با توزیع مجدد کلیدها بین دو نود از حذف نود (n) جلوگیري میشود.

انواع دیگر B-Tree

ایندکس B*Tree چگونه است؟

نوعي B-Tree مي باشد كه در آن:

1) هر نود با لااقل 2/3 ظرفيت خود كليد دارد.

2) عمل شكسته شدن نودها به كمک توزيع مجدد كليدها حتي الامكان به تاخير انداخته مي شود.

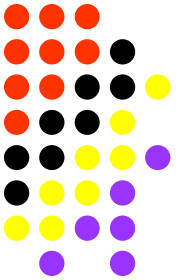
3) ظرفيت نود ريشه بيش از نودهاي ديگر مي باشد تا:

- درصورت Splitting، نودهاي جديد هر كدام 2/3 ظرفيت كليد داشته باشند.

4) هنگام Splitting، هيچگاه يک نود به دو نود جديد تقسيم نمي شود بلكه:

- **دو نود** مجاور با هم **ادغام**،
- و سپس **تبديل به سه نود** مي شوند،
- بطوريكه هر كدام 2/3 ظرفيت كليد داشته باشند.

انواع دیگر B-Tree



ایندکس Virtual B-Tree چگونه است؟

نوعي B-Tree ميباشد كه در آن از روش Buffering of Pages استفاده ميگردد:

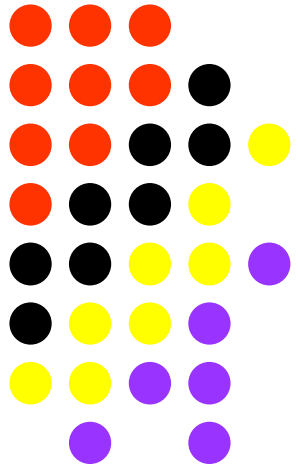
- ✓ نگهداري تعدادي از نودها (Pages) در حافظه RAM باعث صرفه جويي در تعداد دسترسي به ديسک يا I/O ميشود.
- ✓ در اينصورت هنگام لزوم دسترسي به يک نود، اوّل به فضاي رزرو شده و نودهاي موجود در حافظه رجوع مي شود و اگر نود پيدا شد احتياجي به يک I/O جديد نميباشد.
- ✓ درصورت لزوم انجام I/O و آوردن يک نود جديد به حافظه، يکي از صفحات که مدتي استفاده نشده است حذف شده و نود جديد جاي آنرا ميگيرد. (Least Recently Used)
- ✓ روش ديگر بجاي روش LRU، اين مي باشد که حتّي الامکان صفحات مربوط به سطوح بالاتر در حافظه نگاه داشته شده و صفحات مربوط به نودهاي برگ جايگزين شوند.

In the Name of God

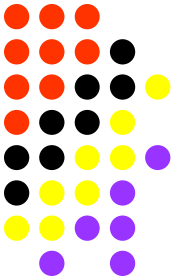
Lecture 15

Indexed Sequential Access, B+trees, Simple prefix B+trees

(Sections 10.1 – 10.5)



Indexed Sequential Access B+trees, Simple prefix B+trees



انواع روش های مورد نیازجهت دسترسي به داده هاي يک فايل کدامند؟

منظوراز روش Indexed Sequential چيست؟

ساختاراينديکس ISAM چگونه بوده است؟

آيا ايندکس B-tree امکان دسترسي سري به رکوردها را بترتيب کليد ميدهد؟

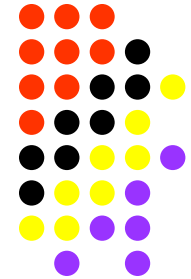
چگونه دسترسي سري به رکوردهاي يک فايل بترتيب کليد ميسر ميشود؟

ساختار يک Sequence Set چگونه است؟

ساختاراينديکس B+tree چگونه است؟

ساختاراينديکس Simple Prefix B+tree چگونه است؟

Indexed Sequential Access



انواع روش های مورد نیازجهت دسترسي به داده هاي يک فايل کدامند؟

1) روش دسترسي بکمک ايندکس (Indexed Access Method)

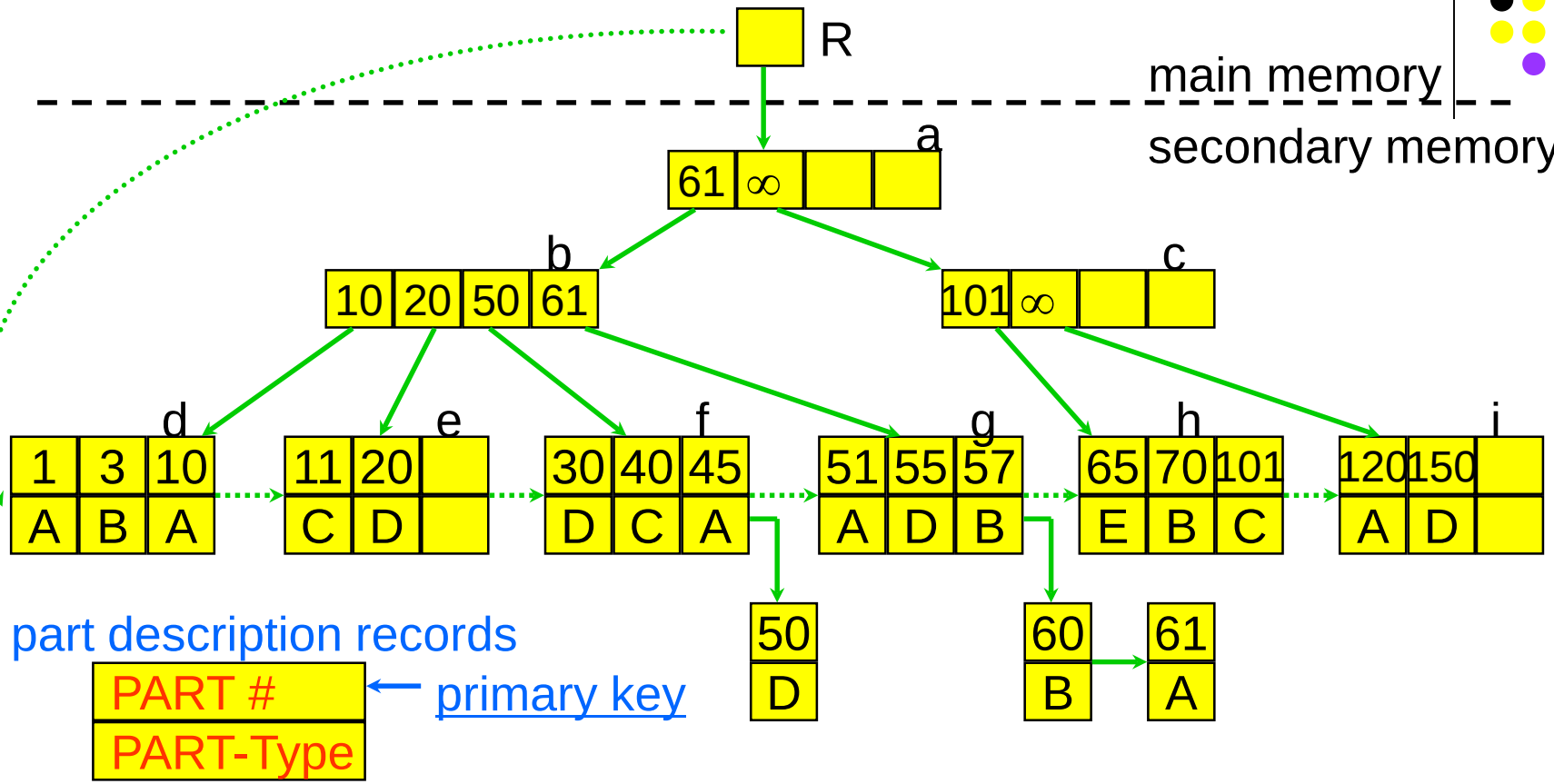
- ✓ دسترسي به بعضي ازرکوردهاي فايل با استفاده از کليد و ايندکس.

2) روش دسترسي سري (Sequential Access Method)

- ✓ دسترسي به کليه رکوردهاي فايل بترتيب کليد اصلي ولی بدون استفاده از ايندکس. (چرا؟)
- ✓ در اينصورت بايد رکوردهاي فايل بطور فيزيکي بر حسب کليد اصلي مرتب شده باشند.
- ✓ کاربرد اين روش در بعضي پردازش ها (Batch Processing) که احتياج به تکرار عمليات روي تمام رکوردهاي فايل دارند ميباشد.
- ✓ مثال : پرداخت حقوق ماهيانه کارمندان يک سازمان.

History : ISAM File

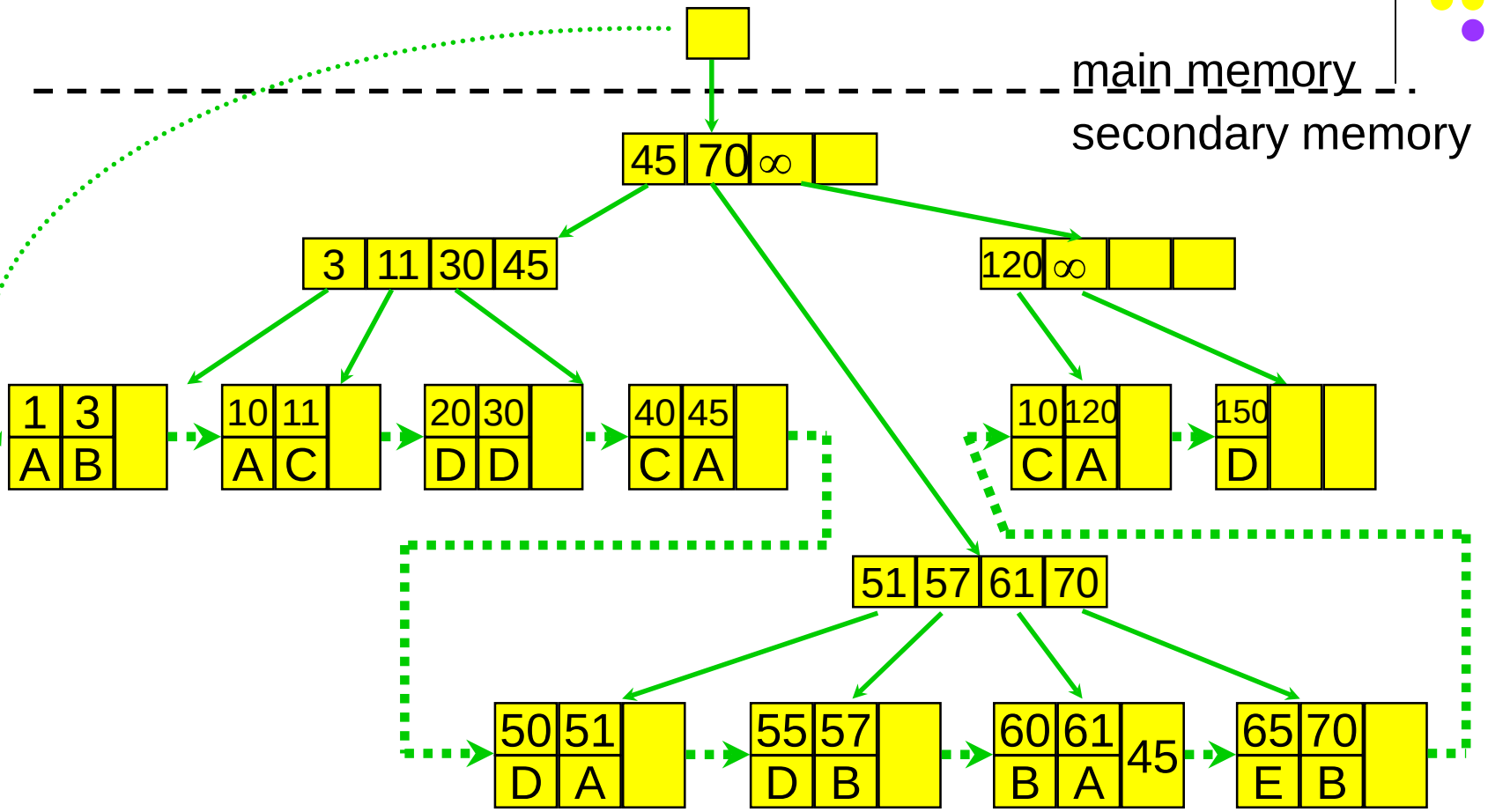
ساختار ایندکس های ISAM چگونه بوده است؟



Ex: **Indexed sequential** structure (when using **overflow chain**)

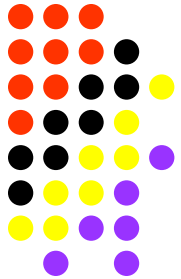
History : ISAM File

ساختار ایندکس های ISAM چگونه بوده است؟



Ex: **Reorganization**

Indexed Sequential Access



آیا ایندکس B-tree امکان دسترسي سري به رکوردها را بترتیب کلید میدهد؟

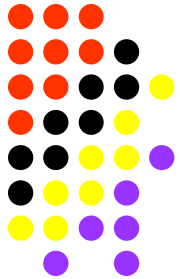
در ایندکس B-tree :

- ✓ نودهای برگی فقط شامل کلیدها واشاره گرهایي به رکوردهای داده میباشند.
- ✓ **هیچگونه ترتیب خاصي** برای رکوردهای داده تعریف نگردیده است.
- ✓ **دسترسي سری** به رکوردهای داده **بترتیب کلید ممکن نمیباشد**. **(چرا؟)**

❖ **چگونه دسترسي سري به رکوردهاي یک فایل بترتیب کلید میسر میشود؟**

- ✓ برای اجتناب از لزوم مرتب سازي (sort) کلیه رکورد هاي یک فایل،
- ✓ میتوان فایل را به صورت بلوکهایي از رکوردهاي مرتب شده نگهداري نمود.
- ✓ این ساختار موسوم به Sequence Set میباشد.

Indexed Sequential Access



ساختار یک Sequence Set چگونه است؟

1) رکوردهاي فايل به **تعدادي بلوک** گروه بندي ميشوند.

2) هر بلوک حاوي لااقل **نصف ظرفيت** خود از رکوردها ميباشد.

3) **رکوردهاي** داخل هر بلوک **مرتب شده** (sorted) ميباشد.

4) **بلوک ها** نيز در رابطه با يکديگر **مرتب شده** ميباشند.

5) ولي ترتيب آنها بطور **فيزيکي نيست**. بلکه با استفاده ازاشاره گر تامين ميشود.

6) عمليات **حذف** و **اضافه** رکوردها **شبيه** عمليات در گره های **B-Tree** ميباشند.

- ✓ **ايجاد** (insertion) يک رکورد در بلوک مخصوص خود (با توجه به کليد آن) ممکن است باعث شکسته شدن (Block **Splitting**) بشود. (**overflow**)
- ✓ **حذف** (deletion) يک رکورد در يک بلوک ممکن است باعث ادغام دو بلوک Block **Merging** يا Block **Redistribution** بشود. (**underflow**)

Sequence Set

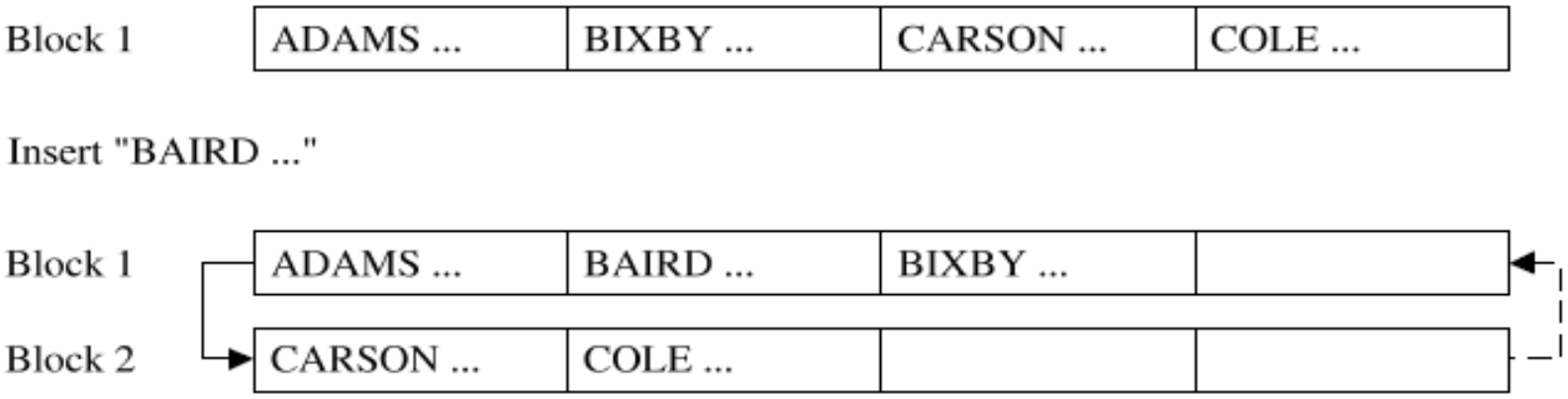
مثال:

Example:
Block size = 4
key : **Last Name**

Forward Pointer

Backward Pointer

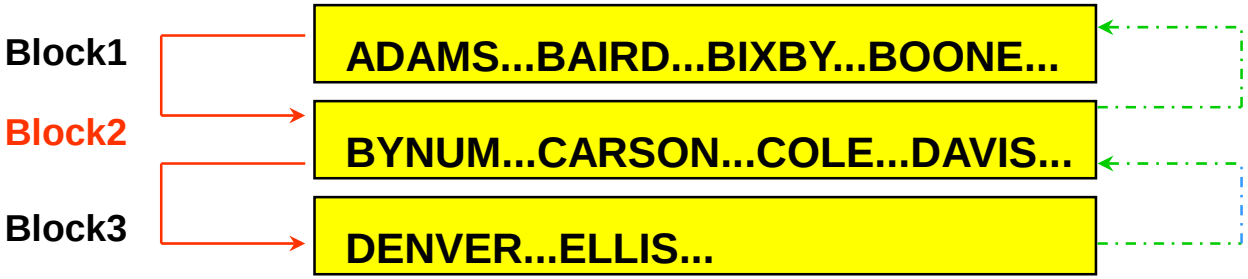
- **Insertion with overflow:**



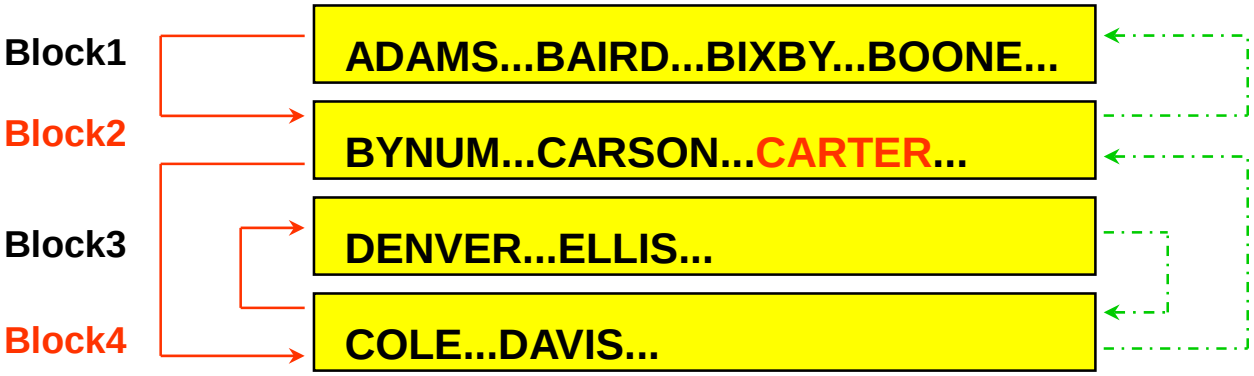
(شکل 10.1 صفحه 427 کتاب)

Sequence Set

مثال (ادامه...):



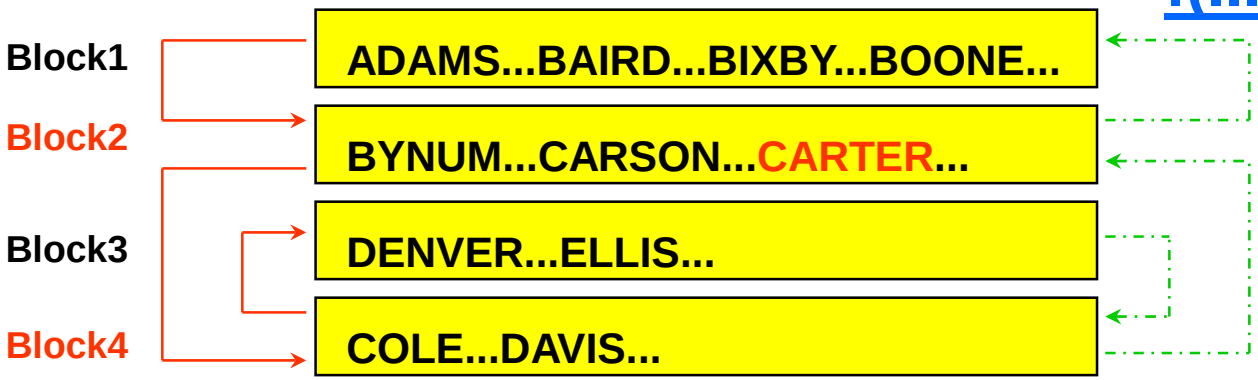
(a) Initial blocked sequence set



(b) Sequence set after insertion of CARTER record
- block 2 splits, and the contents are divided between blocks 2 and 4

Sequence Set

مثال (ادامه...):



(b) Sequence set after insertion of CARTER …

(c) Sequence set after deletion of DAVIS record
- block 4 is less than half full, so it is merged with block3

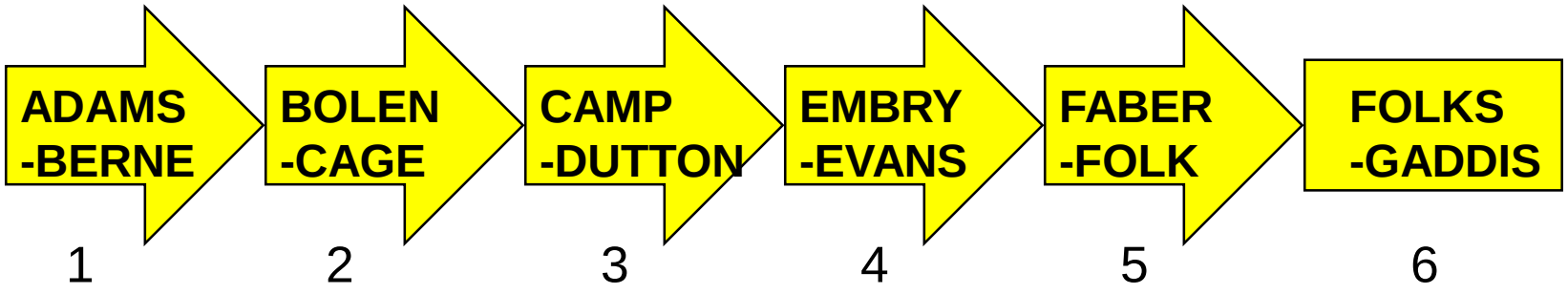
Sequence Set

مزاياي ساختار sequence set چيست؟

- احتياجي به مرتب سازي كليه ركوردهاي فايل بعد ازهرعمل ايجاد يا حذف ركورد نميباشد.

معايب ساختار sequence set چيست؟

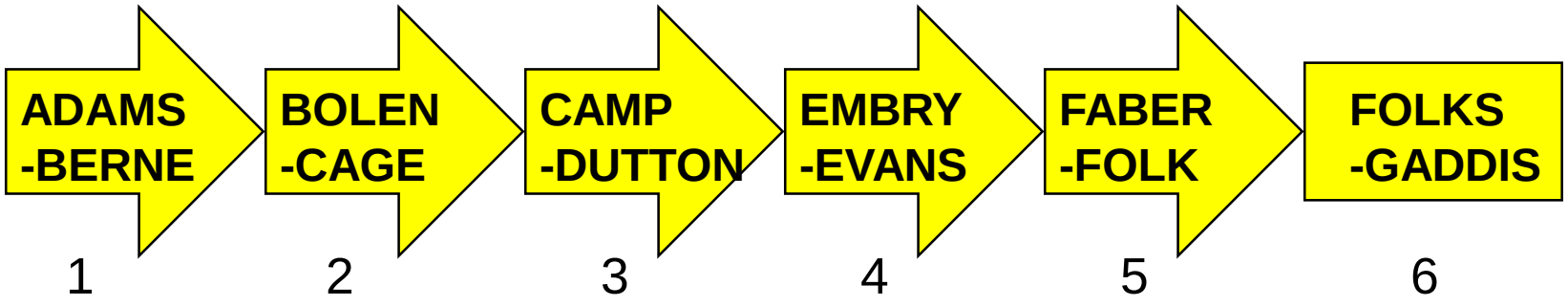
- فضاي ديسک بيشتري براي نگهداري فايل لازم است.
- چون بلوک ها مي توانند 50% ظرفيت خود ركورد داشته باشند.
- ترتيب فيزيكي ركوردها فقط در داخل يک بلوک صادق است (نه در كل فايل).



Sequence Set

شرايط انتخاب اندازه هر بلوک چگونه است؟

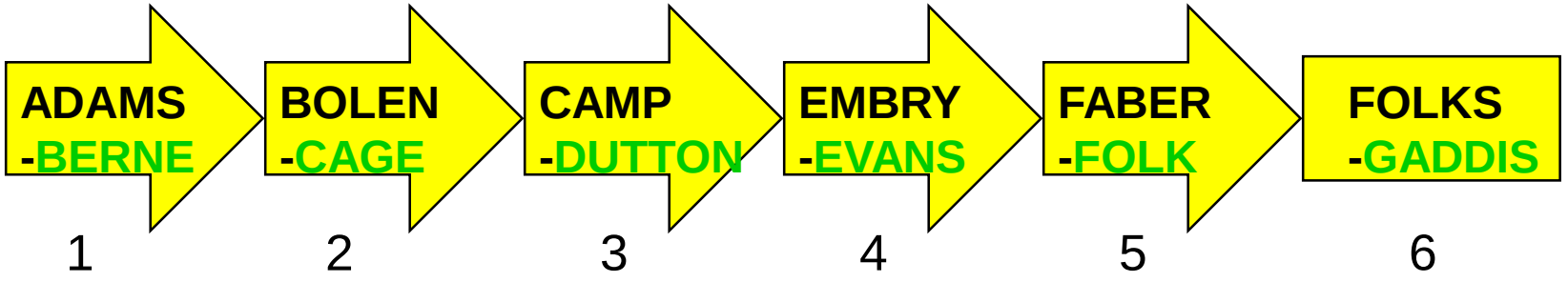
- ✓ بسته به روش Merge / Redistribution موردنظر بايستي حافظه RAM فضاي لازم براي **لااقل 2 يا 3 بلوک** را داشته باشد.
- ✓ بهتر است که براي خواندن هر بلوک **فقط يک دسترسي** به ديسک (seek) احتياج باشد.
- ✓ در ديسک هاي **سکتور بندي** شده اندازه هر بلوک ميتواند معادل **يک cluster** انتخاب شود.
- ✓ در ديسک هاي **بلوک بندي** شده اندازه هر بلوک مي تواند معادل **يک Track** (يا **نصف** آن) انتخاب شود.



Sequence Set

Sequence Set روش ایجاد ایندکس **B-tree** با توجه به ساختار چگونه است؟

Key	Block Number
BERNE	1
CAGE	2
DUTTON	3
EVANS	4
FOLK	5
GADDIS	6



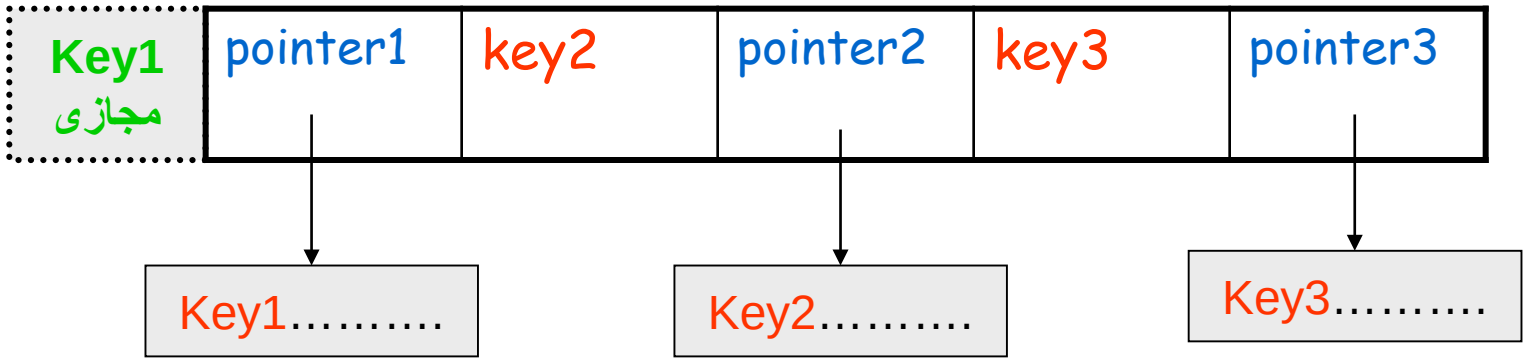
ساختار ایندکس B+Tree

ساختاریک ایندکس B+tree چگونه است؟

- ✓ ساختار ایندکس B+Tree **شبیه** به ساختار ایندکس در **B-Tree** میباشد، ولی **با دو تفاوت**:

1) در B+Tree **کوچکترین کلید** هر نود به عنوان reference در نود parent ظاهر میشود.

2) حضور **اولین کلید** در نود parent به طور **مجازی** میباشد.



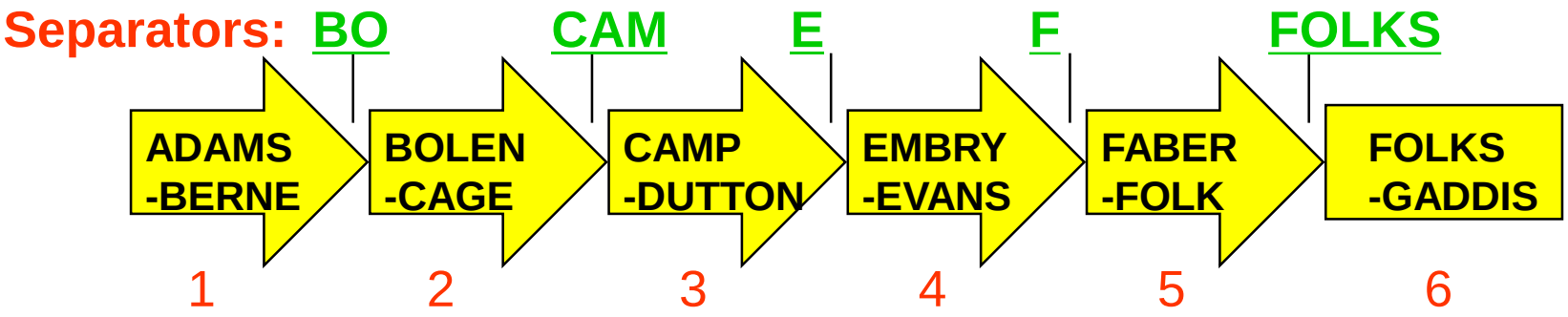
Simple Prefix B+Tree

ساختاریک ایندکس Simple Prefix B+tree چگونه است؟

- ✓ با توجه به ساختار sequence set ميتوان با استفاده از separator هاي كوتاه بين محتواي بلوک ها تميز قائل شد.

مثال:

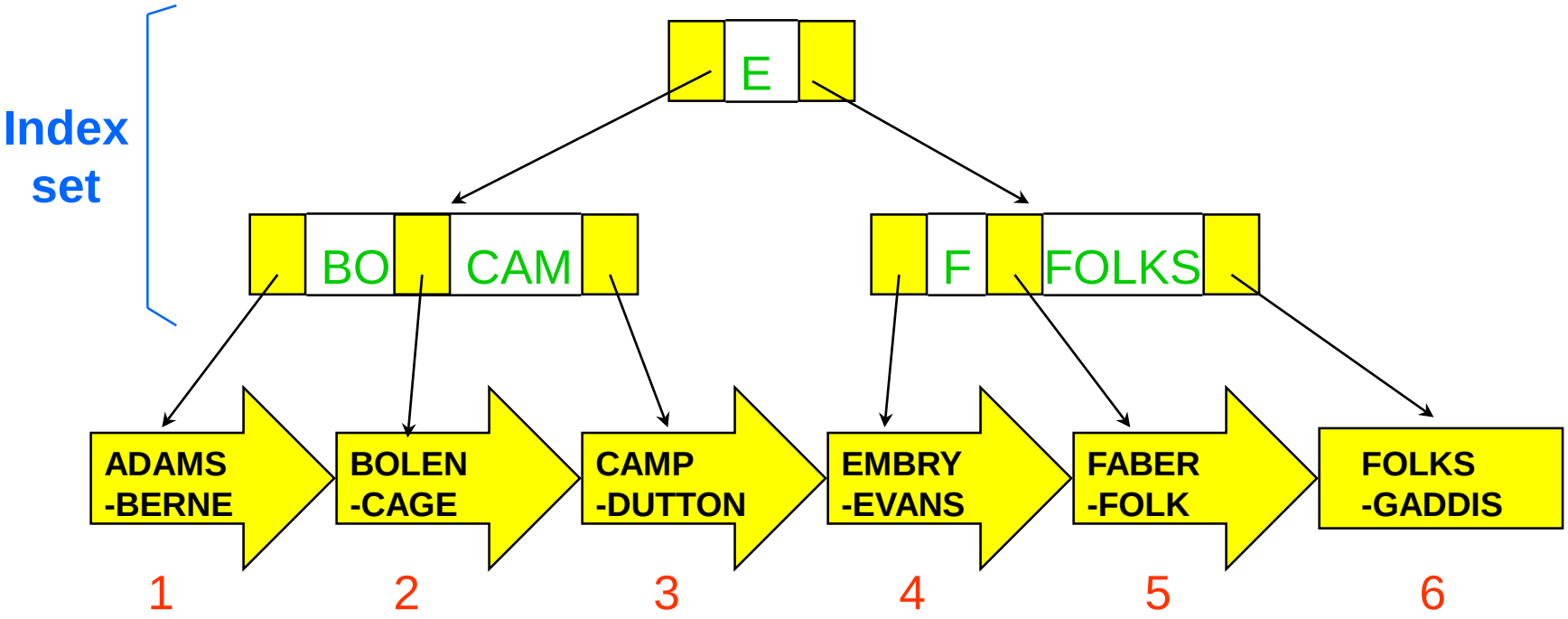
Block No.	Range of Keys	Separator
1	Adams-Berne	BO
2	Bolen-Cage	CAM
3	Camp-Dutton	E
4	Embry-Evans	F
5	Faber-Folk	
6	Folks-Gaddis	FOLKS



Simple Prefix B+Tree

ساختاریک ایندکس Simple Prefix B+tree چگونه است؟

مثال: (شکل 10.7 صفحه 434 کتاب)

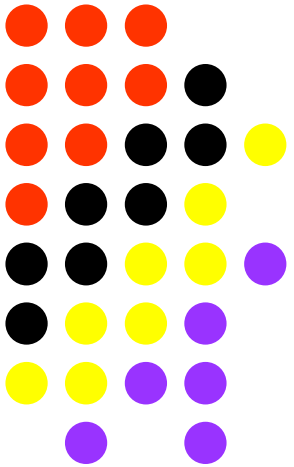


In the Name of God

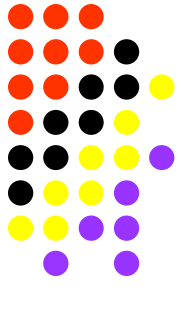
Lecture 16

More on B+Trees: Maintenance, Loading, Perspectives

(Sections 10.6 -10.11)



More on B+Trees



نگاهداري يک ايندکس Simple Prefix B+tree چگونه است؟

شرايط انتخاب اندازه هر بلوک Index Set چگونه است؟

ساختاريک ايندکس Variable-Order B+tree چگونه است؟

مزايا و معايب Variable Order B+Tree کدامند؟

روش بهينه ايجاد (loading) يک B+Tree چگونه است؟

خواص مشترک انواع B-Tree و B+Tree کدامند؟

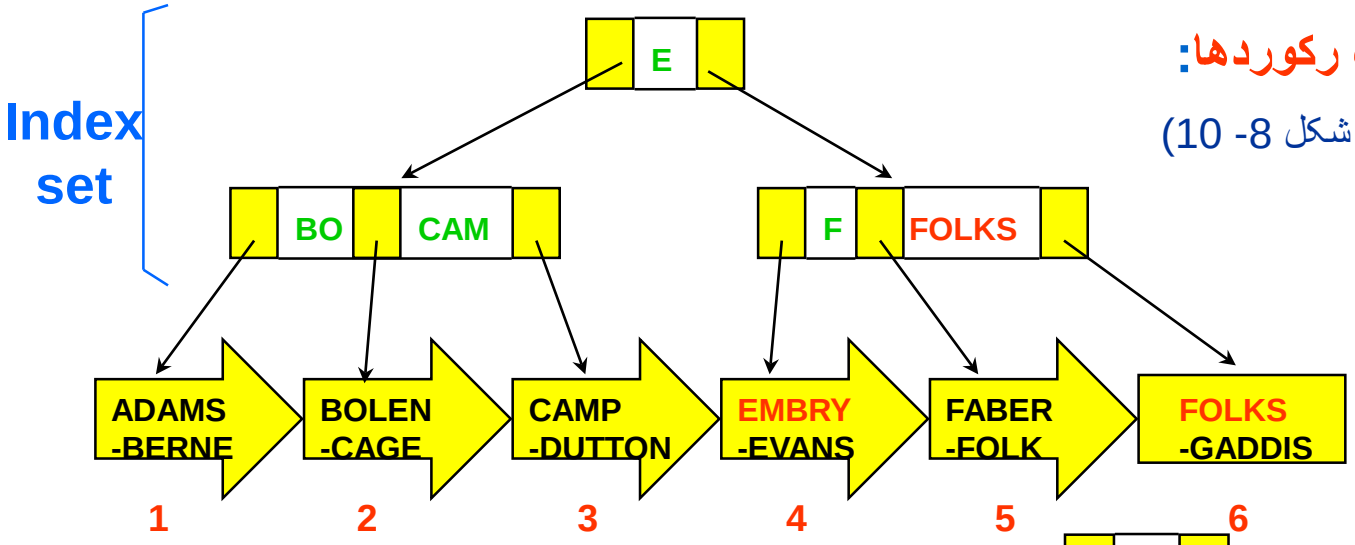
طراحی وبسایت – برنامه نویسی – پروژه پایگاه داده – SQL Server
مهندسی نرم افزار UML و SSADM
۰۹۱۳۱۲۵۳۶۲۰ www.a00b.comآ صفر صفر بی دات کام

Simple Prefix B+Tree

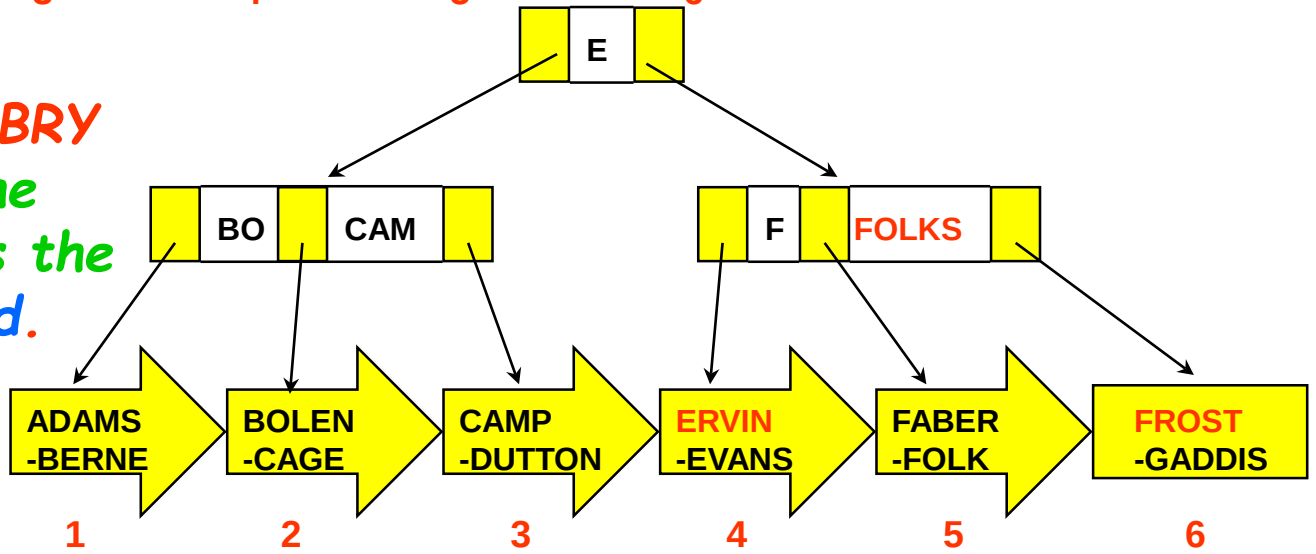
نگاهداري يک ايندکس Simple Prefix B+tree چگونه است؟

مثال (1): حذف رکوردها:

(صفحه 436 کتاب شکل 8- 10)



Deletion of the EMBRY and FOLKS from the sequence set leaves the index set unchanged.



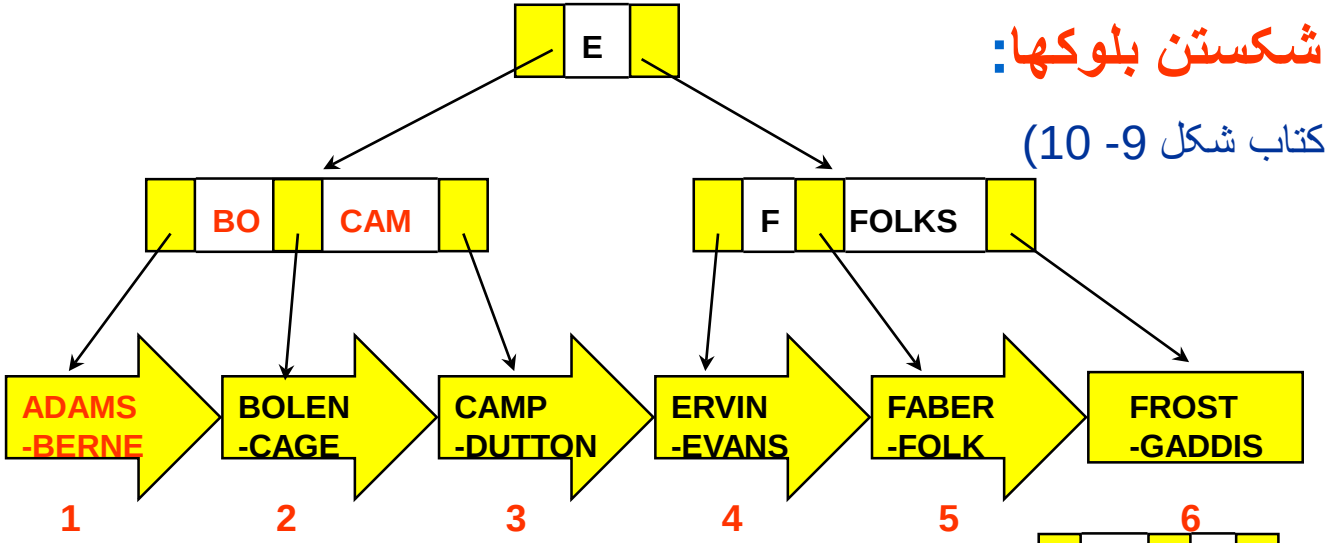
Prof. Hyoung-Joo Kim, Comp Eng, Seoul National Univ

Simple Prefix B+Tree

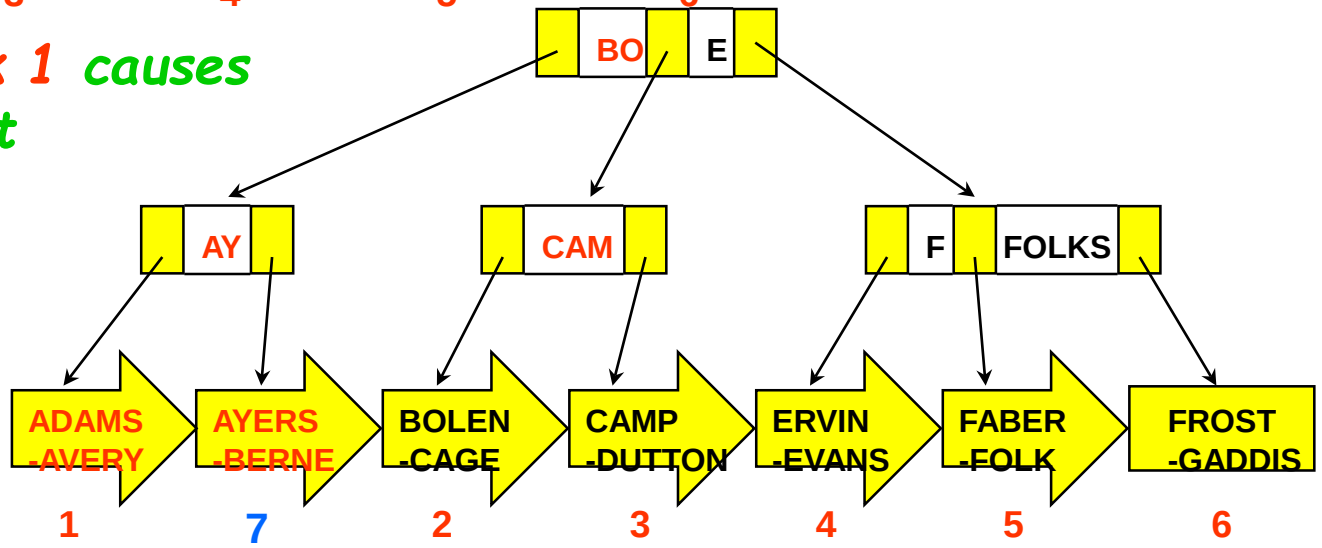
نگاهداري يک ايندکس Simple Prefix B+tree چگونه است؟

مثال (2): شکستن بلوکها:

(صفحه 437 کتاب شکل 9- 10)



An insertion into block 1 causes a split, the consequent addition of block 7 and the index set changes.

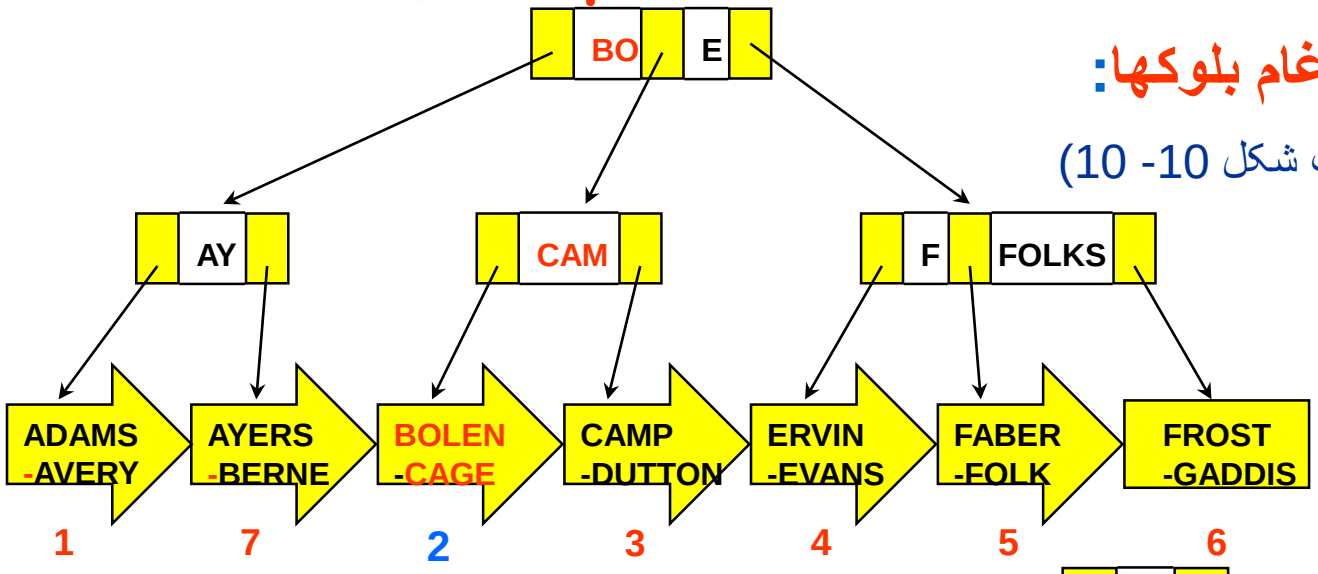


Simple Prefix B+Tree

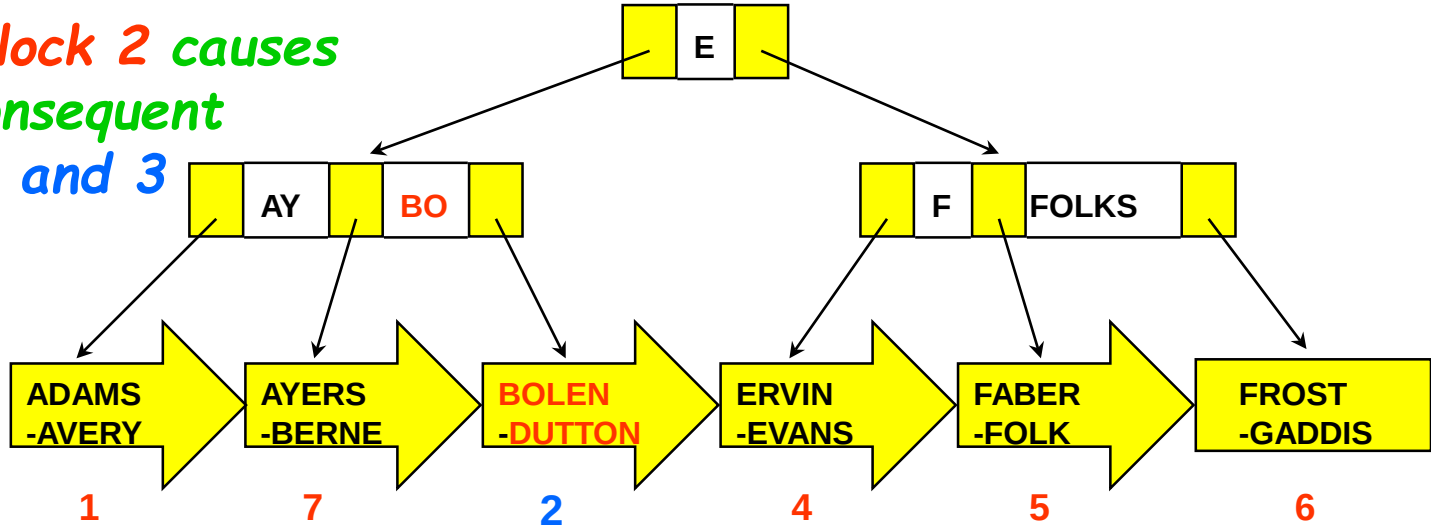
نگاهداري يک ايندکس Simple Prefix B+tree چگونه است؟

مثال (3): ادغام بلوکها:

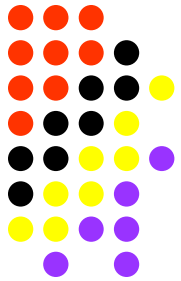
(صفحه 438 کتاب شکل 10- 10)



A deletion from block 2 causes Underflow, the consequent merge of blocks 2 and 3 and the index set changes.



انتخاب اندازه بلوکهاي Index Set



شرايط انتخاب اندازه هر بلوک Index Set چگونه است؟

چرا بهتر است که اندازه بلوکهاي index set برابر با اندازه بلوکهاي sequence set باشد؟

- ✓ انتخاب اندازه بلوکهاي sequence set با در نظر گرفتن عواملي بوده است که در تعيين index set نيز همانقدر اهميت دارند، مثل:
  - ❑ ظرفيت حافظه RAM و
  - ❑ مشخصات مربوط به ديسک ها.
- ✓ استفاده از **بافرهاي مشترک** براي نگهداري بلوکها در حافظه (Caching) ساده تر ميشود. (چرا؟)
- ✓ بلوکهاي **ايندکس و داده** مي توانند **در يک فايل** ذخيره شده و به يکديگر نزديکتر باشند. (چرا؟)

Variable-Order B+Tree

ساختاریک ایندکس Variable-Order B+tree چگونه است؟

نوعي B+Tree که در آن:

1) **ظرفیت** (order) نودهاي ایندکس **متغیر** میباشد و

2) **اطلاعات** موجود در این نودها حتي الامکان **فشرده** شده میباشد.

مثال: (صفحه 441 کتاب شکل11- 10)

separators

As, Ba, Bro, C, Ch, Cra, Dele, Edi, Err, Fa, File

AsBaBroCChCraDeleEdiErrFaFile	00 02 04 07 08 10 13 17 20 23 25

Variable-length separators and corresponding index

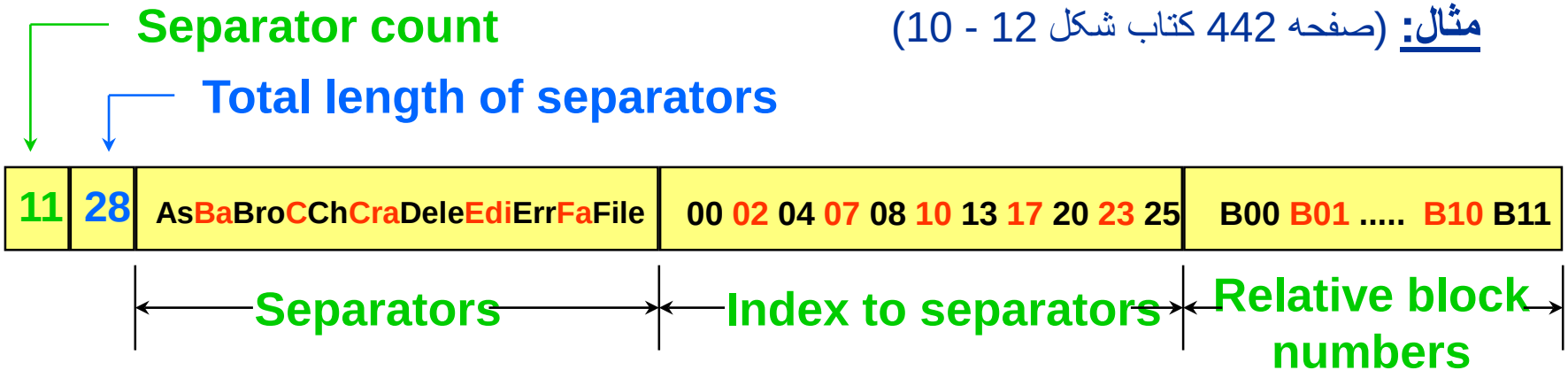
Variable-Order B+Tree

ساختاریک ایندکس Variable-Order B+tree چگونه است؟

در این ساختار:

- ✓ فضاي **موجود** براي نگهداري separator ها **بطور كامل** **استفاده** شده است.
- ✓ **ايندكس** مربوط به separator ها امكان **جستجوي دودويي** را ميدهد.
- ✓ **بلوكها** بوسيله (Relative Block Number) بطور مستقيم قابل **آدرس دهي** هستند.

مثال: (صفحه 442 كتاب شكل 12 - 10)



Structure of an index set block

Variable-Order B+Tree

ساختاریک ایندکس Variable-Order B+tree چگونه است؟

مثال: (صفحه 445 کتاب شکل 10.15)

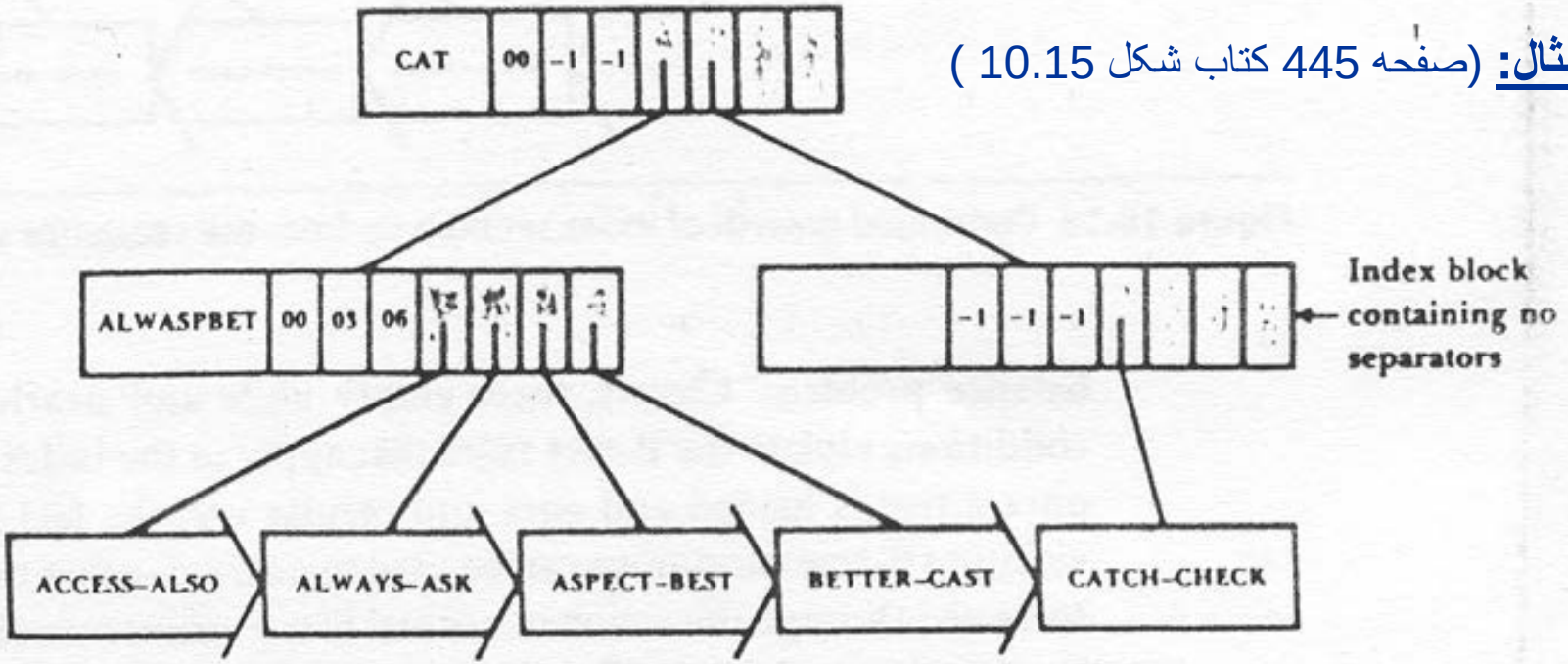
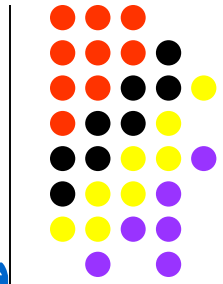


Figure 10.15 Simultaneous building of two index set levels as the sequence set continues to grow.

Variable-Order B+Tree



مزاياي Variable Order B+Tree کدامند؟

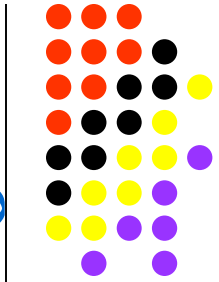
- ✓ **درجه** یا **order** ایندکس به **ماکزیمم** ممکن (با توجه به اندازه بلوک) رسیده و
- ✓ **عمق درخت** (**depth**) به **مینیمم** ممکن خود میرسد و
- ✓ بنابراین در **تعداد I/O صرفه جويي** میشود. (**seek**)

معایب Variable Order B+Tree کدامند؟

- ✓ **تشخیص** اینکه چه زماني یک بلوک به **ظرفیت مینیمم** یا **ماکزیمم** خود رسیده **مشکل** میباشد.
- ✓ اعمال مربوط به **تجزیه**، **ادغام** و **توزیع** مجدد کلیدها در گره هاي مختلف **مشکل تر** خواهندبود.

طراحی وبسایت – برنامه نویسی – پروژه پایگاه داده – SQL Server
مهندسی نرم افزار UML و SSADM
۰۹۱۳۱۲۵۳۶۲۰ www.a00b.comآ صفر صفر بی دات کام

a B+Tree Loading



روش بهينه ايجاد (loading) يک B+Tree چگونه است؟

براي تبديل يک **فايل بزرگ** به B+Tree بهتر است که:

- ✓ از **روش معمولي** ايجاد رکورد ها به طور random **استفاده نشود**،
- ✓ چون عملي بسيار **طولاني** و **سنگين** خواهد بود. **(چرا؟)**

روش بهتر اين خواهد بود که :

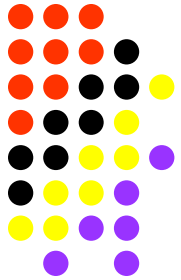
1) ابتدا، رکوردهاي فايل **مرتب** شوند (**sort**)

2) سپس، **رکوردهاي متوالي** که مي توانند در هر بلوک قرار بگيرند **دسته بندي** شده و بطور **يکجا نوشته** شوند. (يعني با **يک I/O** براي **هر بلوک** داده)

3) در ضمن **separator** هاي بلوک هاي متوالي **به مرور جمع آوري** شده و در حافظه نگهداري شوند و هر نود ايندکس پس از تکميل ظرفيت **بطور يکجا نوشته** شود. (يعني با **يک I/O** براي **هر نود** ايندکس)

طراحی وبسایت – برنامه نویسی – پروژه پایگاه داده – SQL Server
مهندسی نرم افزار UML و SSADM
آ صفر صفر بی دات کام www.a00b.com ۰۹۱۳۱۲۵۳۶۲۰

a B+Tree Loading



مزاياي اين روش loading چيست؟

1) **نوشتن بلوک ها** بصورت سري (**sequential**) انجام ميشود.

2) فقط **يک بار** احتياج به **خواندن داده ها** (و فقط داده ها) ميباشد.

3) **احتياجي** به تجزيه، ادغام و توزيع مجدد کليدها در بلوک هاي مختلف **نميباشد**.

4) **درجه** استفاده از ظرفيت بلوک ها (**order**) براحتي **قابل کنترل** است ودر صورت لزوم ميتواند حتي **100** % نيز تعيين شود.

5) **بلوک ها** از نظر فيزيکي نيز **مجاور يکديگر** قرار مي گيرند و

6) زمان **seek** هنگام استفاده مجدد **کوتاه تر** خواهد بود.

خواص انواع B-Tree و B+Tree

خواص مشترک انواع B-Tree و B+Tree کدامند؟

1) همه از روش **paged index** استفاده میکنند، در نتیجه:
   - ✓ با **هر I/O بلوک هاي بزرگي** از مجموعه کلیدها را به **حافظه** مي آورند،
   - ✓ **فرم درختواره** آنها **broad & shallow** يعني وسيع و با **عمق** (**level**) **کم** ميباشد.
2) **عمق** آنها **متوازن** ميباشد. (**Height-Balanced Trees**)
3) به روش **Bottom-Up** و با اعمال تجزیه، ادغام و توزیع مجدد کلیدها **رشد میکنند**.
4) **کارآيي** آنها با روش هاي تجزیه، ادغام و **توزیع مجدد** کلیدها **بین 2 تا 3 نود** بسیار **بهتر** میشود.
5) **کارآيي** آنها با روشهاي **caching** يعني نگهداري تعدادي از بلوک ها در حافظه **بهتر** ميشود.
6) **قابلیت تطبیق** با رکوردهاي **با طول متغیر** را دارند.

طراحی وبسایت – برنامه نویسی – پروژه پایگاه داده – SQL Server
مهندسی نرم افزار UML و SSADM
۰۹۱۳۱۲۵۳۶۲۰ www.a00b.com آ صفر صفر بی دات کام