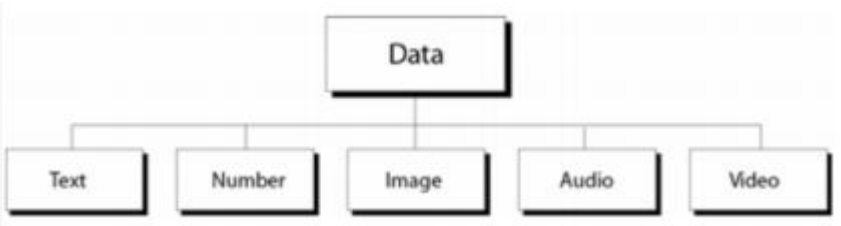
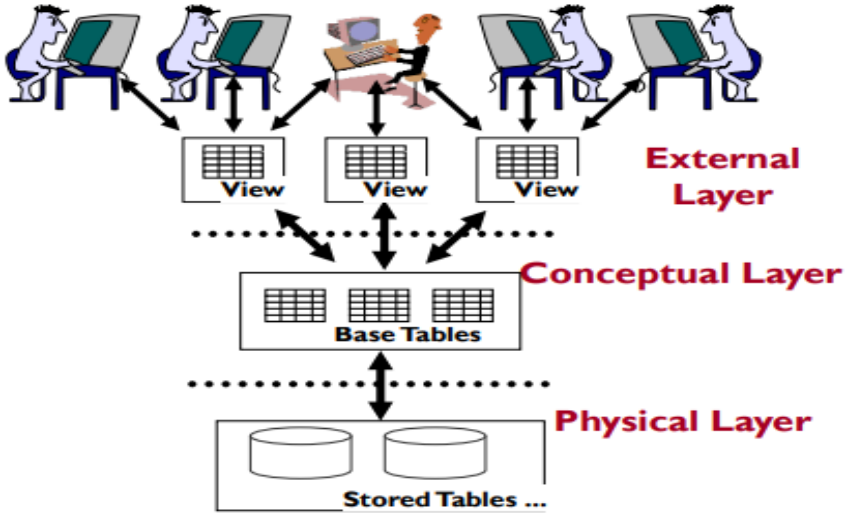
1) *Ders01 - Introduction:*



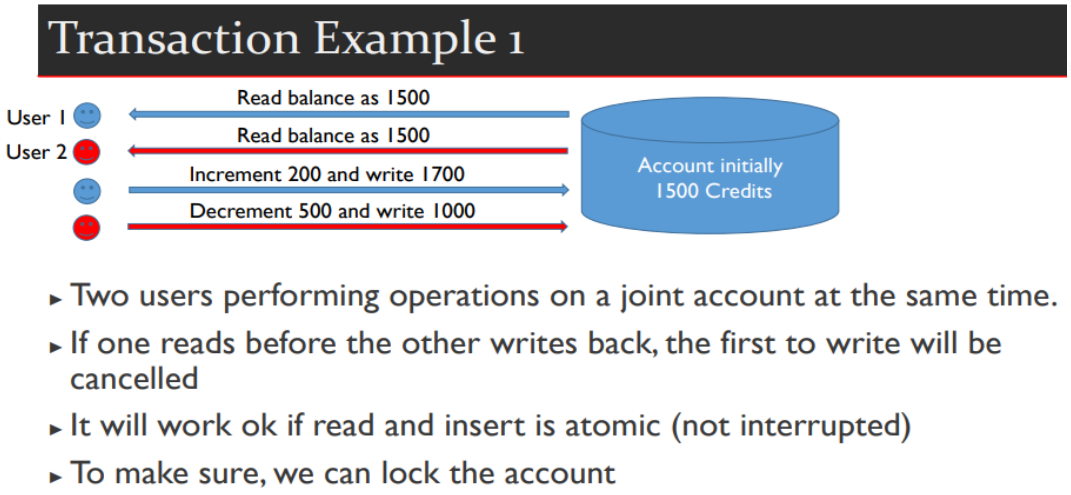
- **Data,** hemen hemen her türlü örgütlenmemiş gerçek(ler). Ör: Bilgisayar Dosyası.
- **Signal,** iletim için gerekli olan verilerin kodlanmasıdır.
- Veri işlendiğinde ve organize edildiğinde bilgi olur ve böylece yararlı olur.
- Komplex Verileri; ***relational (ilişkisel), graph (grafik)*** veya ***Structured (Yapısal)*** olarak ifade edilebilir.
- **VTYS,** bir yazılım paketidir, veritabanlarını depolamak ve yönetmek için tasarlanmış. Çok büyük, entegre bir veri koleksiyonu.
- **Neden VTYS kullanırız?**
  - ✓ Veri bağımsızlığı ve verimli erişim,
  - ✓ Veri bütünlüğü ve güvenliği,
  - ✓ Eşzamanlı erişim,
  - ✓ Azaltılmış uygulama ve geliştirme süresi,
  - ✓ Tekdüzen veri yönetimi,
  - ✓ Çökmelerden kurtarma.
- **Veri Modelleri,** verileri açıklamak için oluşturulan bir kavramlar topluluğudur.

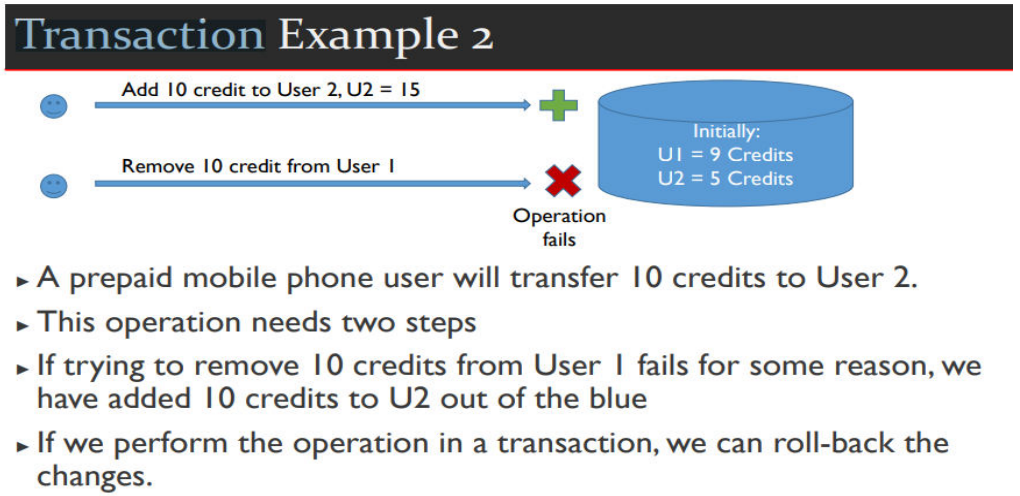


  - ✓ Bir şema belirli bir veri koleksiyonunun, verilen veri modelini kullanarak tanımlanmasıdır.
  - ✓ İlişkisel (relational) veri modeli bugün en çok kullanılan modeldir.
  - ✓ Ana konsept ilişki, temelde satır ve sütun içeren bir tablo.
  - ✓ Her ilişkide sütunları veya alanları tanımlayan bir şema vardır.
  - ✓ Şema şu şekilde tanımlanır: şema adı, her alanın adı (veya niteliği veya sütunu) ve her alanın türü. Ör: Öğrenci(numara: integer, isim: string)
- **Veri Bağımsızlığı,** verilerin nasıl yapılandırıldığından ve depolandığından yalıtılmış uygulamalar;

✓ ***Mantıksal Veri Bağımsızlığı:*** Verilerin mantıksal yapısındaki değişikliklerden korunma. (Güzel Açıklama :başparmak: :D)

✓ ***Fiziksel Veri Bağımsızlığı:*** Verilerin fiziksel yapısındaki değişikliklerden korunma.





2) Ders02 - Storage Devices:

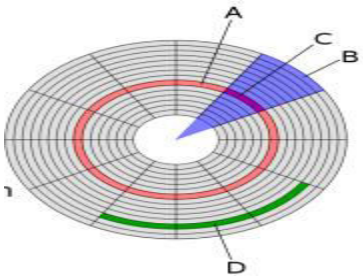
❖ **Disk Organizasyonu;**

✓ ***Platters:*** Verileri depolamak için her iki tarafı kullanabilir.

✓ ***Track:*** Verilerin manyetik olarak saklandığı dairesel bir yoldur (A şeklinde gösterilir).

✓ ***Sector:*** Parçanın bir alt bölümü. (C)Normal olarak sektör, iki yarıçap ve yay ile çevrelenmiş disk kısmı anlamına gelir.(B)

✓ ***Cluster:*** Aktarılan veri birimi, çoklu sektörler (örn. 3 sektör) cinsinden tanımlanır (D olarak gösterilir). Bu genellikle bir dosya sistemi ile ilgili ayardır.



✓ ***Sector:*** Veri depolama için minimum parça bölümü.

✓ ***Cluster:*** Sektörler grubu

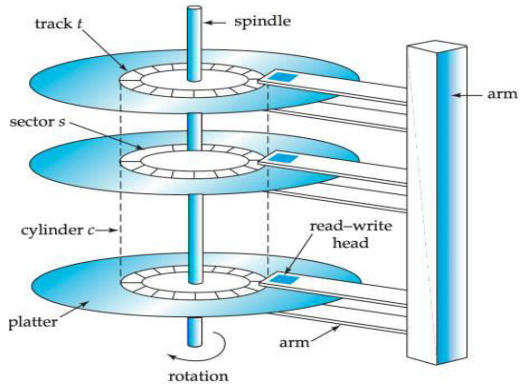
✓ ***Track:*** Platter'da bir döngü.

✓ ***Cylinder:*** Plakalardaki aynı seviye izlerden oluşan semantik bir şekil.

✓ ***Platters,*** istenen sektör için pozisyona dönüyor.

✓ ***Kol tertibatı,*** kafayı istenilen bir parça üzerine konumlandırmak için içeri veya dışarı doğru hareket ettirilir.

- ✓ Sadece bir kafa herhangi bir zamanda okur / yazar.
- ✓ ***Blok boyutu,*** sektör büyüklüğünün (sabit olan) bir katıdır.



❖ **Bloklama (Blocking);**

- ✓ ***Fiziksel Blok,*** Birincil ve ikincil hafıza arasında aktarılan minimum veri
- ✓ ***Mantıksal Blok,*** Birincil ve ikincil hafıza arasında transfer edilen minimum kayıt seti
- ✓ ***Engelleme Faktörü,*** Mantıksal blok başına mantıksal kayıt sayısı
- ✓ Varsayalım, bir küme = bir blok.

❖ **Disk Kapasitesinin Tahmini;**

- ✓ ***Track Capacity = Number Of Sectors Per Track * Bytes Per Sector***
- ✓ ***Cylinder Capacity = Number Of Tracks Per Cylinder * Track Capacity***
- ✓ ***Driver Capacity = Number Of Cylinder * Cylinder Capacity***
- ✓ ***Access Time = Seek Time + Rotational Delay***
- ✓ ***Time To Read = Seek Time + Rotational Delay + Transfer Time***
- ✓ ***Transfer Time = (Number Of Sectors Transferred / Number Of Sectors On a Track) * Rotation Time***

▸ 20 plate, 800 track/plate, 25 sector/track, 512 byte/sector, 3600 rpm
▸ 7 ms from track to track, 28 ms avg. seek time, 50 msec max seek time

Avg. Rotational time	=1/2 * (60000/3600) = 16.7/2 = 8.3msec
Disk capacity	= 25 * 512 * 800 * 20 = 204.8 MB
Time to read track	= 1 rotation = 16.7 msec
Time to read cylinder	= 20 * 16.7 = 334msec
Time to read whole disc	= 800 * time to read cylinder + 799* time to pass from one cylinder to another =800*334 + 799*7msec =267 sec + 5.59 sec = 272.59 sec

▸ Disc specs: avg. seek time 8 msec, 10.000 rpm, 170 sector/track, 512 byte/sector

Time to read one sector = avg. seek time + rotational delay + time to transfer one sector
= 8 + (0.5 * 60.000 / 10.000) + (6* 1/170) = 8 + 3+ 0.035 = 11.035 msec

Time to read 10 sequential blocks = avg. seek time + rotational delay + 10 * time to transfer one sector = 8 + 3 + 10 * 0.035 = 11.35 msec

Time to read random 10 block= 10 * (avg. seek time + rotational delay + time to transfer one sector)= 10 * (8 + 3 + 0.035) = 113.5 msec

You have total 200.000 records stored in heap file and length of each record is 250 byte. Here are the specifications of disk used for storage: 512 byte/sector, 20 sector/track, 5 sector/cluster, 3000 track/platter, one sided total 10 plate, rotational time 7200 rpm, seek time 8 msec. Records do not span between sectors

- ▸ Blocking factor of heap file: 10
- ▸ Number of blocks to store whole file: 20.000 blocks
- ▸ Number of blocks to estimate disk capacity: 120.000 blocks
- ▸ Cylinder number to store the data with cylinder based approach : 500
- ▸ Time to read one block : 14,225 ms
- ▸ Time to read file from beginning to end (in worst case): 284.500sec, 4741,67 min, 79,02 hour

❖ **RAID;**

- ✓ ***RAID Seviye 1:*** Her diski, bir veri diski, diğeri de yedek disk olarak yansıtırız.
- ✓ ***RAID Seviye 2:*** Parity bloklarını depolayan sadece bir yedek disk kullanın.
- ✓ Aynı anda iki veya daha fazla disk çökmesi varsa ne olur?
- ✓ ***RAID Seviye 5:*** Her disk, bazı bloklar için yedek bir disktir.
- ✓ Örnek: 4 disk varsa, disk 0, 4, 8, 12, vb. Bloklar için yedeklidir; disk 1; 1, 5, 9 vb. için gereksizdir.
- ✓ ***RAID Seviye 6:*** Birden çok disk çökmesi için tasarlanmıştır. Hamming kodunu kullanır.

3) Ders03 - File Concepts:

❖ **Pin_count,** sayfayı kullanan işlemlerin sayısını takip etmek için kullanılır. "0" kimse kullanmadığı anlamına gelir.

❖ **Dirty,** diskten okunduğundan beri bir sayfanın değiştirildiğini belirtmek için bayrak (kirli bit) olarak kullanılır. Sayfa havuzdan çıkarılacaksa, diski yıkamanız gerekir.

Sample Buffer Pool

Page_no = 1 Pin_count = 3 Dirty = 1 Last Used: 12:34:05	Page_no = 2 Pin_count = 0 Dirty = 1 Last Used: 12:35:05	Page_no = 3 Pin_count = 1 Dirty = 0 Last Used: 12:36:05	Page_no = 4 Pin_count = 2 Dirty = 0 Last Used: 12:37:05	Page_no = 5 Pin_count = 0 Dirty = 0 Last Used: 12:38:05
Page_no = 6 Pin_count = 0 Dirty = 0 Last Used: 12:29:05	Page_no = 7 Pin_count = 1 Dirty = 1 Last Used: 12:20:05	Page_no = 8 Pin_count = 0 Dirty = 1 Last Used: 12:40:05	Page_no = 9 Pin_count = 2 Dirty = 0 Last Used: 12:27:05	Page_no = 10 Pin_count = 0 Dirty = 1 Last Used: 12:39:05

Which page should be removed if LRU is used as the policy:..... 6

Which page should be removed if MRU is used as the policy :..... 8

Which pages do not need to be written to disc, if it is removed:..... 5, 6

Which pages could not be removed in this situation:..... P.c != 0

❖ Genellikle, (bf is blocking factor. N is the size of the file in terms of the number of records) :

❖ ***At least 1 block is accessed (I/O cost : 1),*** ***At most N/bf blocks are accessed.*** *On average N/2bf.*

❖ ***Time To Fetch One Record = (N/2bf) * Time To Read One Block***

❖ ***Time To Read One Block = Seek Time + Rotational Delay + Block Transfer Time***

❖ ***Time To Read All Records = N/bf * Time To Read Per Block***

❖ ***Time To Add New Record = Time To Read One Block (For Last Block) + Time To Write One Block (For Last Block)***

❖ ***If the last block is full; Time To Add New Record= Time To Read One Block (For Last Block) + Time To Write New One Block (For New Last Block)***

❖ ***Time To Update One Fixed Length Record = Time To Fetch One Record + Time To Write One Block***

❖ ***Time To Update One Variable Length Record = Time To Delete One Record + Time To Add New Record***

- FileA: 10000 records , BF = 100, 4 extents
- File B: 5000 records, BF = 150, 3 extents

- Time to find the number of common records of FileA and B

  Time to read FileA= 4 * (seek time + rotational delay) + (10000/100) * block transfer time

  Time to read FileB = 3 * (seek time + rotational delay) + (5000/150) * block transfer time

  = Time to read FileA + 100 * Time to read FileB

  (imagine you've got only two frames in the buffer pool.)
- Read FileA and compare each record of FileA with whole records in FileB

- We can do binary search (assuming fixed-length records) in the sorted part.

- Worst case to fetch a record :

$$T_F = \log_2 (m/bf) * \text{time to read per block.}$$

- If the record is not found, search the overflow area too. Thus total time is:

$$T_F = \log_2 (m/bf) * \text{time to read per block} + k/bf * \text{time to read per block}$$

4) *Ders04 - Index Files:*

❖ **Birkaç Terim;**

✓ Bir ***veri dosyası*** tüm kayıtları saklar.

✓ Veri dosyası bir veya daha fazla ***dizin dosyasına*** sahip olabilir.

✓ Dizin dosyaları, ***arama anahtarları*** ve ***işaretçileri*** veri dosyası kayıtlarına kaydeder.

✓ Dizinler ***birincil*** veya ***ikincil*** olabilir.

✓ ***Birincil Dizin:*** Veri dosyası, arama anahtarına göre yapılandırılmıştır. Örneğin. Masanın birincil anahtarı.

✓ ***İkincil İndeks:*** Tablonun diğer bazı özelliklerine dayanarak, veri dosyasındaki kayıtların sırası arama anahtarından bağımsızdır.

✓ ***Sıralı dosya:*** Kayıtlar birincil anahtara göre sıralanır.

❖ **Dense Index:** [https://www.youtube.com/watch?v=lg8S2s_yTh4]

✓ Bir dosyadır.

✓ Ilk satırı işaret eder.

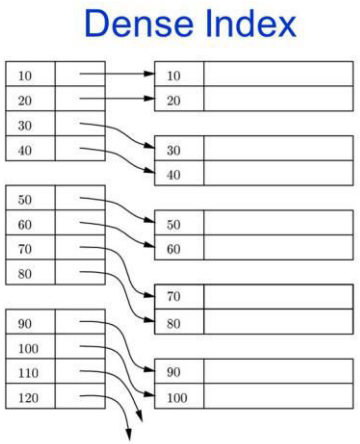
✓ Key - Pointer (Her kayıt için)

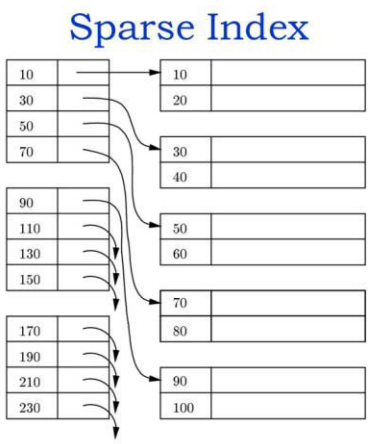
❖ **Sparse Index:**

✓ Bir dosyadır.

✓ En düşük arama anahtarını işaret eder.

✓ Key - Pointer (Her blok için)





- Suppose there is a data file of 4 GB (2^{32}) in a system with blocks of 1KB and fixed length records of 256Bytes.
- The records are stored in sorted order with respect to the key Student ID.
- Index stores a search key of 4 Bytes and a 4 Bytes of pointer. So, an index entry is 8 bytes.
- How many disk accesses do we need to find a record with a given Student ID:
  - Using sorted data file
  - Using dense index
  - Using sparse index

- $2^{32} / 2^{10} = 2^{22}$ blocks are in the data file.
- Blocking Factor of data file = $2^{10}/2^{8} = 2^{2}$
- 2^{22} x BF of data file = 2^{24} records in the file
- With binary search we can have 22 disk accesses to find the record we are searching for in the worst case.
- If an index entry is 8 bytes we can fit into a block $2^{10}/2^{3}=2^{7}$ entries
  - Dense Index should be: $2^{24}/ 2^{7}=2^{17}$ blocks = 2^{17} x 2^{10} = 2^{27} = 128MB index file. So, a binary search is 17 disk accesses.
  - Sparse Index should be: $2^{22}/2^{7}=2^{15}$ blocks = 2^{15} x 2^{10} = 2^{25} = 32MB index file. So, a search is 15 disk accesses.
- What would happen if we had a two-level sparse index?

5) Ders05 - Cost:

Cost Model for Our Analysis

We ignore CPU costs, for simplicity:

- B: The number of data pages
- R: Number of records per page
- D: (Average) time to read or write disk page
- Measuring number of page I/O's ignores gains of pre-fetching a sequence of pages; thus, even I/O cost is only approximated.
- Average-case analysis; based on several simplistic assumptions.

Operations to Compare

- Scan: Fetch all records from disk
- Equality search
- Range selection
- Insert a record
- Delete a record

Cost of Operations

	Heap File	Sorted File	Hashed File
Scan all recs	BD	BD	1.25 BD
Equality Search	0.5 BD	D $\log_2$B	D
Range Search	BD	D ($\log_2$B + # of pages with matches)	1.25 BD
Insert	2D	Search + BD	2D
Delete	Search + D	Search + BD	2D

6) *Ders06 & Ders07 - Hast Tables:*

Load Factor

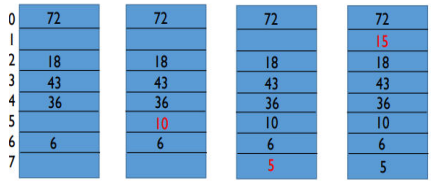
- Loading factor (LF), α = n / m
  n: number of keys
  m: number of slots
- If uniform distribution (1/m) to get mapped to a slot, a slot will have an expectation of α elements.
- If m increases
  - Collision decreases
  - LF decreases
  - 0.5 > LF > 0.8 is unacceptable
  - Storage requirements increases.
- Reduce collisions while keeping storage requirements low.

Linear Probing

$$h(k,i) = (h'(k) + i) \bmod m$$

- Always check the next index
- Increments index linearly with respect to i.
- Clustering problem

hash(10) = 2
hash(5) = 5
hash(15) = 7

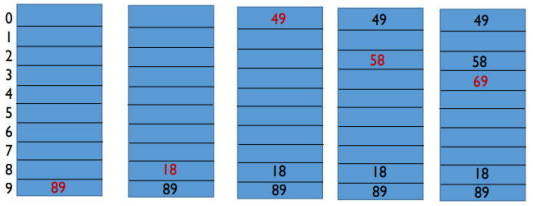


Open Addressing – Quadratic Probing

$$h(k,i) = (h'(k) + c_1 i + c_2 i^2) \bmod m$$

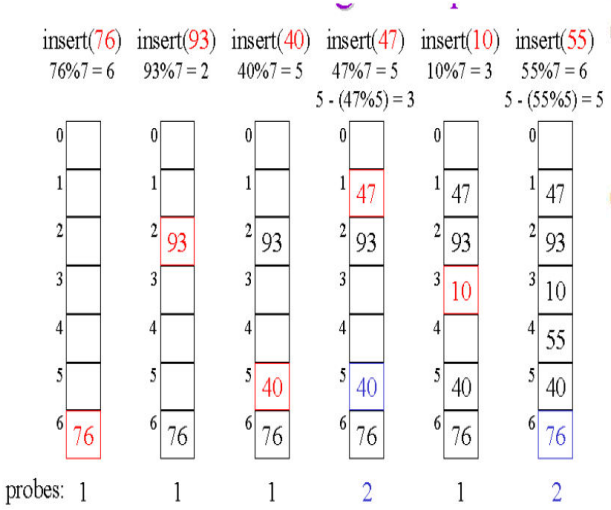
- Instead of moving by one, move i^2

c_1=0, c_2=1
hash(89)=9
hash(18)=8
hash(49)=9
hash(49, 1) = 0
hash(58) = 8
hash(58, 1) = 9
hash(58,2) = 2
hash(69) = 9
hash(69,1) = 0
hash(69,2) = 3



Closed Hashing >>> https://www.cs.usfca.edu/~galles/visualization/ClosedHash.html **Open Hashing >>>** https://www.cs.usfca.edu/~galles/visualization/OpenHash.html

Example 2 – Double Hashing

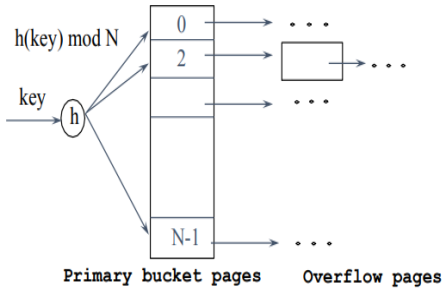


Dynamic Hashing Methods

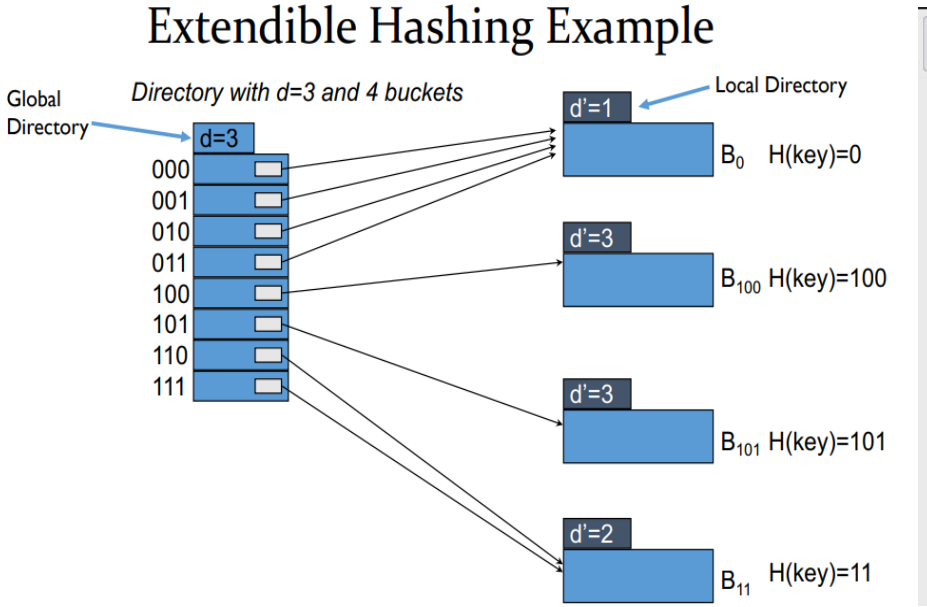
- As for any index, 2 alternatives for data entries **k***:
  - <**k**, rid of data record with search key value **k**>
  - <**k**, list of rids of data records with search key **k**>
  - Choice orthogonal to the *indexing technique*
- *Hash-based* indexes are best for *equality selections*. ***Cannot*** support range searches.

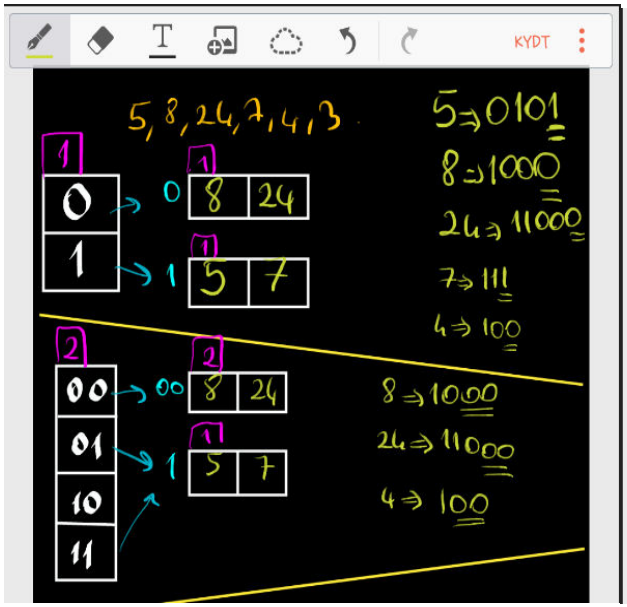
Static Hashing

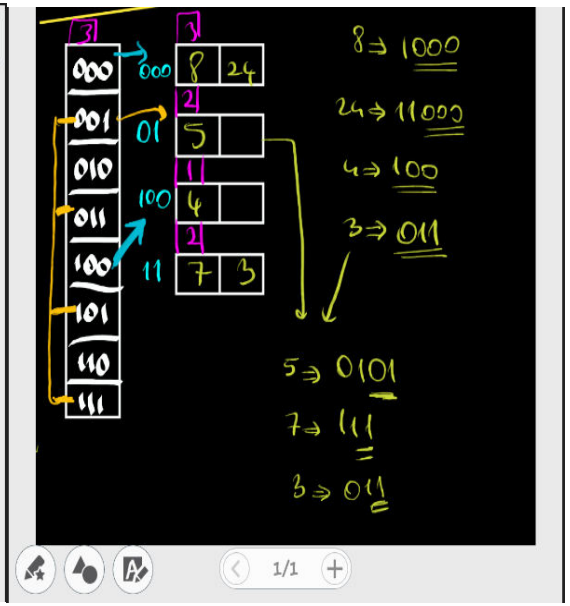
- # primary pages fixed, allocated sequentially, never de-allocated; overflow pages if needed.
- **h**(*k*) mod M = bucket to which data entry with key *k* belongs. (M = # of buckets)



Extendible Hashing >>> https://www.youtube.com/watch?v=TtkN2xRAgv4&t=519s







Linear Hashing With Next Pointer >>> https://www.youtube.com/watch?v=h37Jhr21ByQ

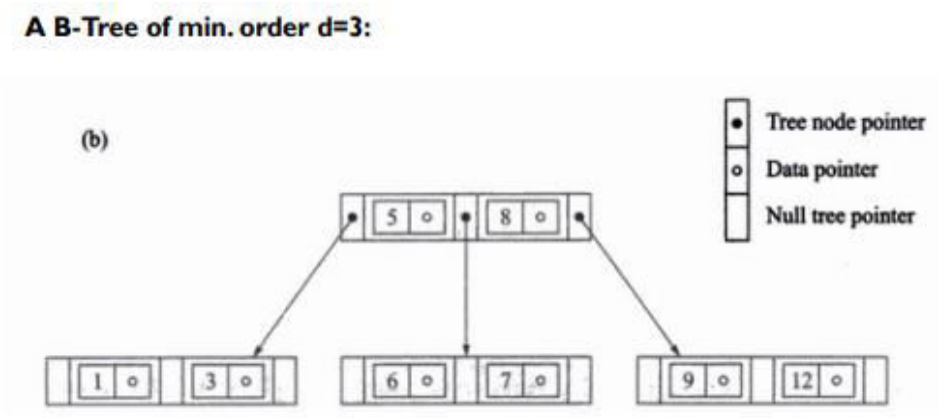
7) Ders08 - Tree Based Indexing:

❖ **Hash temelli endekslemenin** eşitlik araması (linear search) için kullanıldığını ve aralık araması (range search) için çalışmadığını unutmayın.

❖ Aralık tabanlı arama (range search) için **ağaç tabanlı indeksleme (tree based indexing)** (ve eşitlik araması için de) kullanılır.

❖ **B TREE;**

- ✓ Bir B-ağacında, her bir düğüm, çok sayıda anahtar içerebilir.
- ✓ B-ağacı, çok sayıda yöne dallanmak ve her düğümde çok sayıda anahtar bulundurmak için tasarlanmıştır, böylece ağacın yüksekliği nispeten küçüktür.
- ✓ Ağaç her zaman dengelidir.



❖ **B Ağacı Unsurları;**

- ✓ Minimum "d+1" order'a sahip B Ağacı'nın özellikleri:

1. İç düğümler (kök hariç) en az d, en fazla 2d anahtarlarına sahiptir.
2. Kök (bir yaprak değilse) en az 2 çocuğa sahiptir.
3. Tüm yaprak düğümleri aynı seviyededir (dengeli ağaç).
4. Seviye artışı ve azalışlar yukarı doğru (aşağı değil) yapılır.
5. İç düğümde en az d ve d + 1 çocuk var.

❖ **B TREE Visualization >>> https://www.cs.usfca.edu/~galles/visualization/BTree.html**

❖ **B+TREE;**

- ✓ B + ağacı her zaman dengeli.
- ✓ Kök düğüm hariç her düğüm için en az yüzde 50'lik bir kullanım garantisi verilir.
- ✓ Bir kaydın aranması, kökten uygun kanala geçiş yapılmasını gerektirir.
- ✓ Sadece yaprak düğümleri veri girişleri içerir.
- ✓ İç düğümler dizin içerir.
- ✓ Her veri girişi bir yaprak sayfasında görünmelidir.
- ✓ Yaprak düğümleri iki kez birbirine bağlanır. (Doubly Linked List)

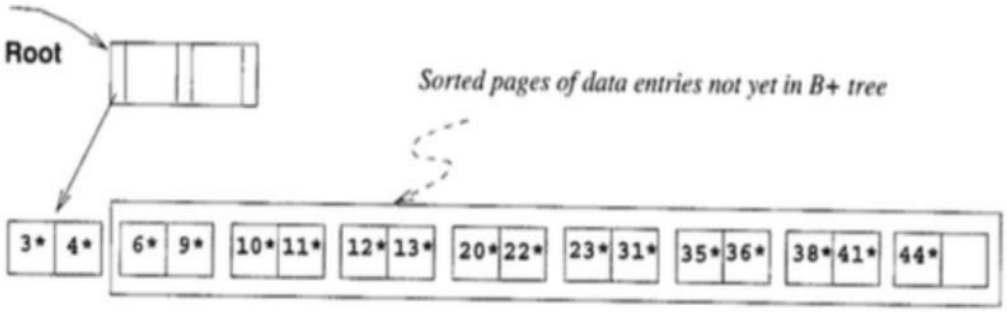
The insert algorithm for B+ Tree

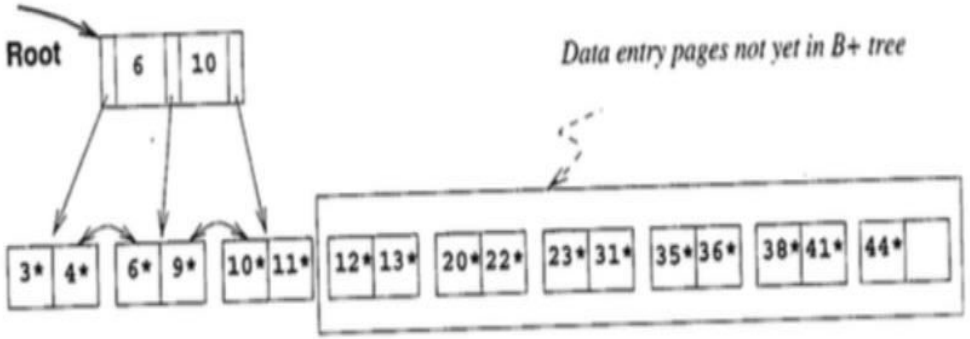
Data Page Full	Index Page Full	Action
NO	NO	Place the record in sorted position in the appropriate leaf page
YES	NO	1. Split the leaf page 2. Place Middle Key in the index page in sorted order. 3. Left leaf page contains records with keys below the middle key. 4. Right leaf page contains records with keys equal to or greater than the middle key.
YES	YES	1. Split the leaf page. 2. Records with keys < middle key go to the left leaf page. 3. Records with keys >= middle key go to the right leaf page. Split the index page. 4. Keys < middle key go to the left index page. 5. Keys > middle key go to the right index page. 6. The middle key goes to the next (higher level) index. IF the next level index page is full, continue splitting the index pages.

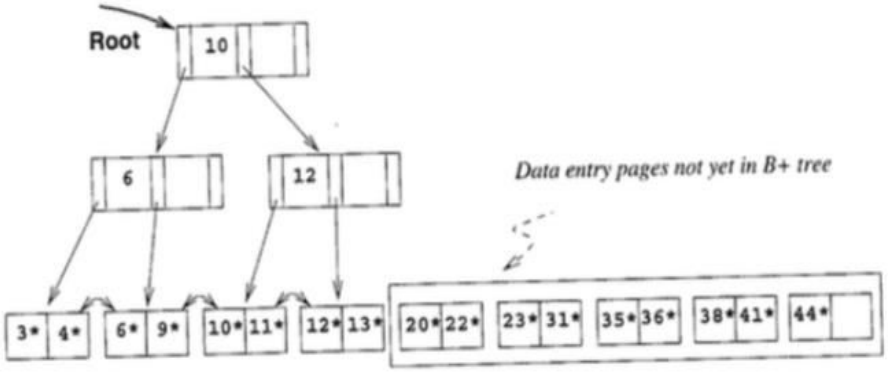
▸ Delete algorithm for B+ trees

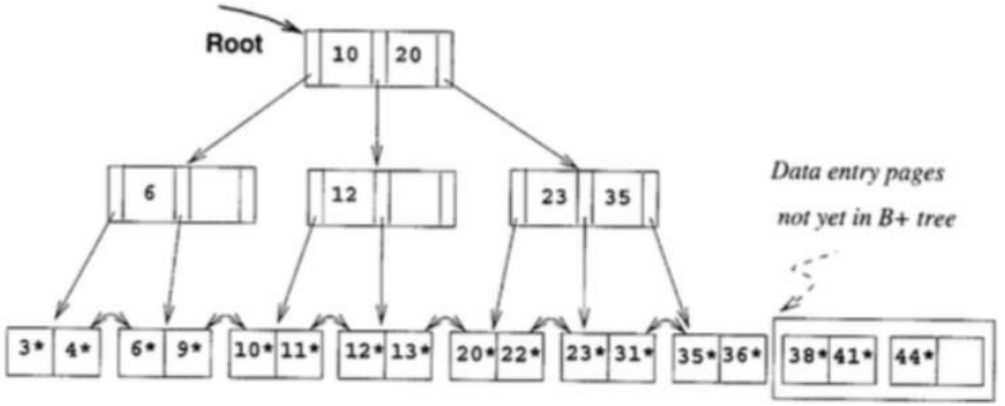
Data Page Below Fill Factor	Index Page Below Fill Factor	Action
NO	NO	Delete the record from the leaf page. Arrange keys in ascending order to fill void. If the key of the deleted record appears in the index page, use the next key to replace it.
YES	NO	Combine the leaf page and its sibling. Change the index page to reflect the change.
YES	YES	1. Combine the leaf page and its sibling. 2. Adjust the index page to reflect the change. 3. Combine the index page with its sibling. Continue combining index pages until you reach a page with the correct fill factor or you reach the root page.

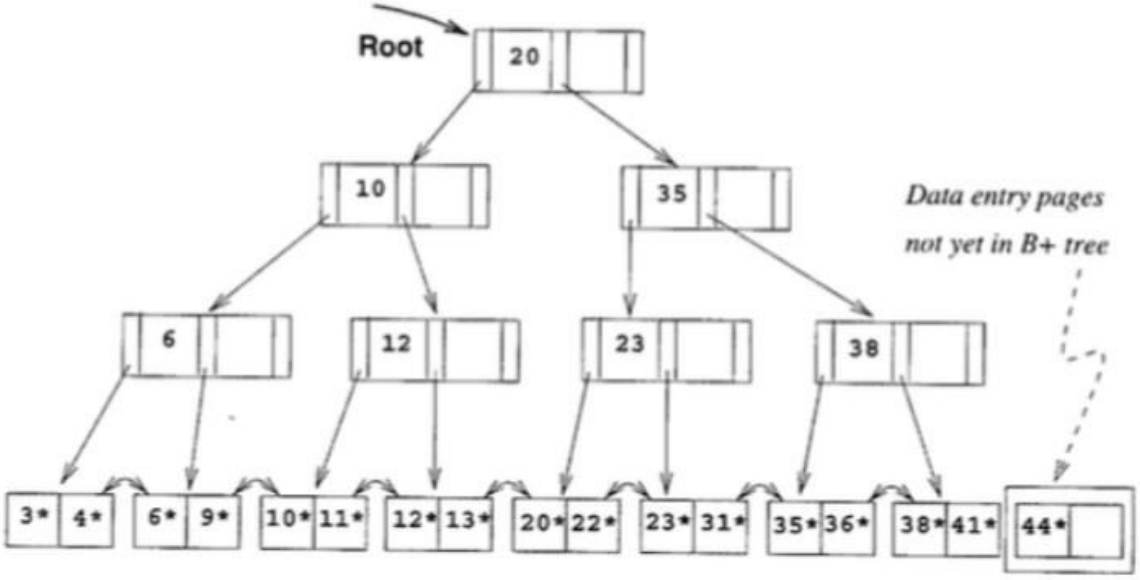
- **B+TREE Visualization >>> https://www.cs.usfca.edu/~galles/visualization/BPlusTree.html**
- Bulk Loading in B+TREE;
  - ✓ Girişlerin birer birer eklenmesi pahalıdır, çünkü her bir eklemede yaprak düğümüne kadar kökten arama yapılmasını gerektirir.
  - ✓ Bunun yerine, eklenecek tüm veriler bir veri girişleri koleksiyonu olarak sıralanır ve eklenir.





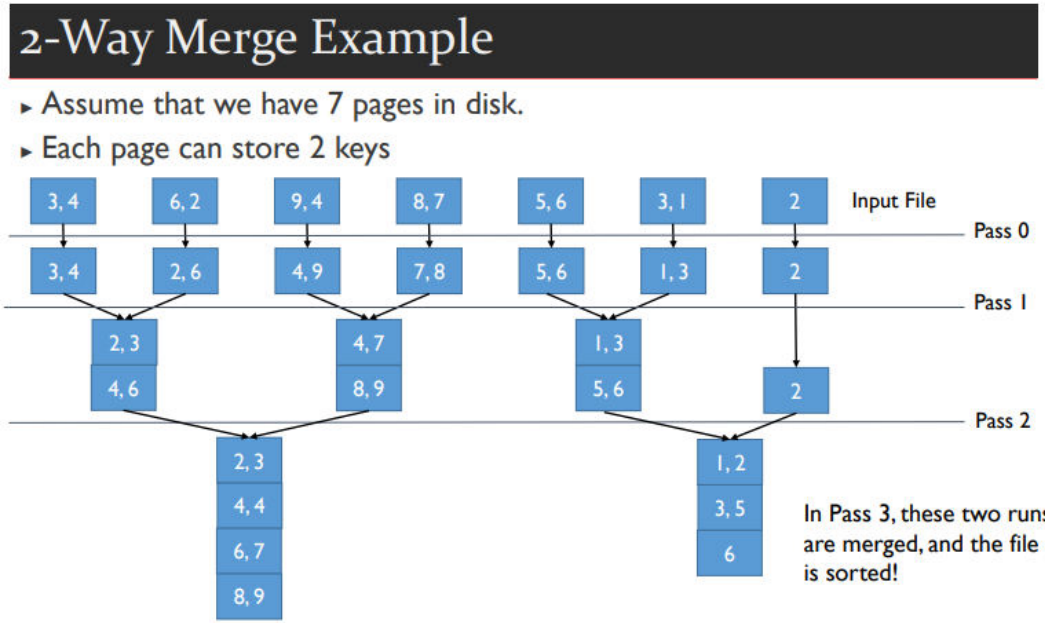

Root
10
6
12
Data entry pages not yet in B+ tree
3* 4*
6* 9*
10* 11*
12* 13*
20* 22*
23* 31*
35* 36*
38* 41*
44*

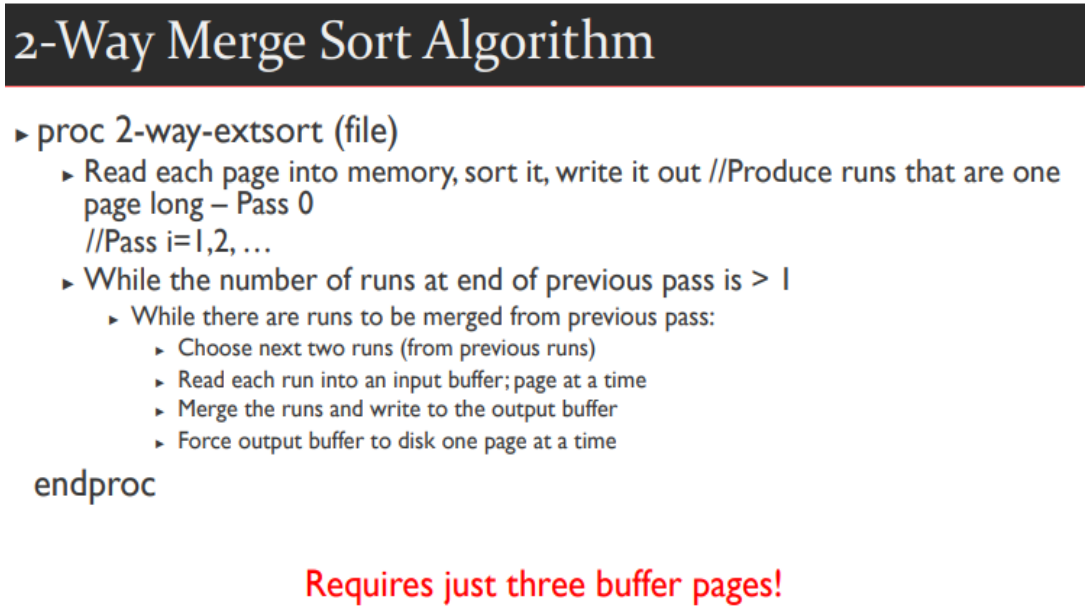

Root
10 20
6
12
23 35
Data entry pages
not yet in B+ tree
3* 4*
6* 9*
10* 11*
12* 13*
20* 22*
23* 31*
35* 36*
38* 41*
44*


Root
20
10
35
Data entry pages
not yet in B+ tree
6
12
23
38
3* 4*
6* 9*
10* 11*
12* 13*
20* 22*
23* 31*
35* 36*
38* 41*
44*

1) *Ders10 - External Sorting (Harici Sıralama)* [https://visualgo.net/bn/sorting]:

- Bir sıralama algoritmasının tamamının bilgisayarın hafızasına (Memory, RAM) yüklü olmaması durumudur. Yani klasik olarak bir dizi (array) veya bağlı liste (linked list) üzerinde yapılan sıralamaları dahili sıralama (internal sort) olarak isimlendirmek mümkündür.
- Harici sıralama, klasik sıralamalardan farklı olarak, verinin ancak bir kısmının RAM'de durması durumunda devreye girer. Örneğin hafızamızın 100MB alan ile sınırlı olduğunu ve 100GB veriyi sırlamamız gerektiğini düşünelim. Bu durumda verinin hafızaya sığması mümkün olmayacak ve verinin harici bir alanda (örneğin disk veya ağ üzerindeki bir kaynakta) durması gerekecek.
- Harici sıralama algoritmaları verinin bir kısmını sıralayıp sonra hafızadaki verinin yerini değiştirip yeni veriyi sıralamak ve en nihayetinde tüm veriyi doğru sıraya sokmak gibi bir yol izlerler.
- Örneğin en çok kullanılan harici sırlama algoritmalarından, ***harici birleştirme sıralaması (external merge sort)*** aynen yukarıda anlatıldığı gibi önce verileri parçalara böler, sonra her parçayı kendi içerisinde sıralar ve en sonunda da verileri birleştirir.
- Elbette verilerin birleşmesi sırasında, verinin tamamının hafızaya sığmaması söz konusudur. Bu durumda verinin parça parça hafızaya yüklenmesi ve sıralanması gerekir.
- **2-Way Merge Sort;**





2-Way Merge Analysis

- The number of passes needed for sorting a file of 2^k is k
- At each step we are merging two runs. So ceil($\log_2$ N), where N is the number of pages in the file. If we add the initial Pass, ceil($\log_2$N) + 1
- In each pass we have to read and write each page. So, the cost for a pass is 2N
- The total cost of this procedure is 2N* ceil($\log_2$N + 1)
- So for our example, with 7 pages. We have 4 passes. At each pass we have 2*7 disk access. The total cost is 56 disk accesses.

Cost of External Merge Sort

- Number of passes:
- Cost = 2N * (# of passes) $\quad 1+\lceil \log_{B-1}\lceil N/B \rceil \rceil$
- E.g., with 5 buffer pages, to sort 108 page file:
  - Pass 0: $\lceil 108/5 \rceil$ = 22 sorted runs of 5 pages each (last run is only 3 pages)
  - Pass 1: $\lceil 22/4 \rceil$ = 6 sorted runs of 20 pages each (last run is only 8 pages)
  - Pass 2: 2 sorted runs, 80 pages and 28 pages
  - Pass 3: Sorted file of 108 pages

A note on complexity

- The asymptotic complexity of both algorithms is O(N log N)
- The base of logarithm can be changed by the rule

$\log_a b = \log_c b / \log_c a$

So a different base changes only the constant of the cost!

- However, when the base is (B-1) the number of passes will drastically decrease

Example

- Assuming that our most general external sorting algorithm is used. For a file with 2,000,000 pages and 17 available buffer pages, answer the following

1. How many runs will you produce in the first pass?
2. How many passes will it take to sort the file completely?
3. What is the total I/O cost of sorting the file?
4. How many buffer pages do you need to sort the file completely in just two passes?

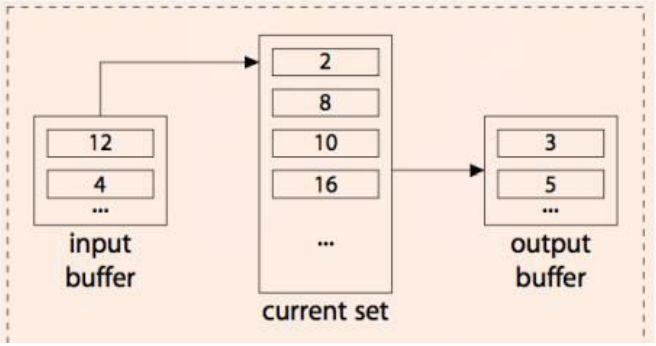
Answer

- How many runs are produced in Pass 0
  - Ceil(2000000/17) = 117648 sorted runs.
- Number of passes required
  - *Ceil($\log_{16}$*117648) + 1 = 6 passes.
- Total number of disk accesses
  - 2*2000000*6 = 24000000.
- How many Buffer pages do we need, to complete sort in two passes
  - We have to produce less than equal to B-1 runs after first pass. So ceil(N/B) must be less than equal to B-1. For $2*10^6$ pages, if $B=10^3$ we have 2000 runs, B=1415 produces 1414 runs which can be merged in single pass.

❖ **Replacement Sort;**

Replacement Sort

- **Replacement sort** can help to further cut down the number of initial runs $[N/B]$: try to **produce initial runs with more than** B **pages**.
- Assume a buffer of B pages. Two pages are dedicated **input** and **output buffers**. The remaining $B - 2$ pages are called the **current set**:



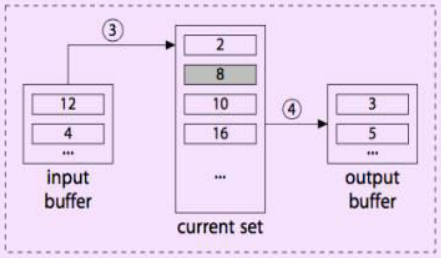
Replacement Sort Algorithm

Replacement sort

1. Open an empty run file for writing.
2. Load next page of file to be sorted into input buffer. If input file is exhausted, go to 4.
3. While there is space in the current set, move a record from input buffer to current set (if the input buffer is empty, reload it at 2).
4. In current set, pick record r with smallest key value k such that $k \geqslant k_{out}$ where k_{out} is the maximum key value in output buffer.[1] Move r to output buffer. If output buffer is full, append it to current run.
5. If all k in current set are $< k_{out}$, append output buffer to current run, close current run. Open new empty run file for writing.
6. If input file is exhausted, stop. Otherwise go to 3.

Replacement Sort Example

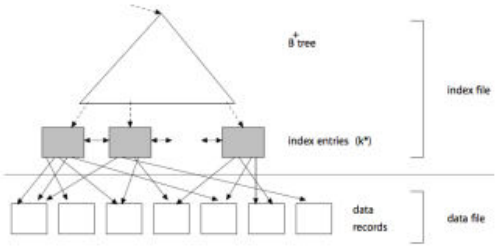
Example (Record with key $k = 8$ will be the next to be moved into the output buffer; current $k_{out} = 5$)



- The record with key $k = 2$ remains in the current set and will be written to the subsequent run.

B+ Trees for Sorting

- If a B+-tree matches a sorting task (*i.e.*, B+-tree organized over key k with ordering θ), we *may* be better off to **access the index and abandon external sorting**.
- 1) If the B+-tree is **clustered**, then
  - the data file itself is already θ-sorted,
  - simply sequentially read the sequence set (or the pages of the data file).
- 2) If the B+-tree is **unclustered**, then
  - in the worst case, we have to initiate one I/O operation per record (not per page)! ⇒ do not consider the index.



2) Ders11 - External Sorting (multi-disk, multi-core):

❖ **Memory Hierarchy;**

- ✓ ***Kayıtlar (Registers),*** CPU hızında çalışır, birkaç yüz Byte.
- ✓ CPU hızının 2 ila 50 katında çalışan ve boyutu birkaç Megabayt kadar değişen farklı ***önbellek (cache memory)*** seviyeleri.
- ✓ ***Ana bellek (Main memory),*** birkaç CPU hızında birkaç kez çalışıyor ve Gigabaytları içeriyor.
- ✓ Erişim süresi milyonlarca devir ve büyüklüğün birkaç Terabayt olduğu ***sabit disk veya katı hal diski (hard disk | solid state disk).*** Diskleri kullanmanın ekonomik yolu, verileri büyük boyutlarda taşımaktır.
- ✓ Benzer bir ifade, hiyerarşinin bitişik iki seviyesi için geçerlidir. Öbek boyutu, bir öbek aktarma süresi ile bir öbek erişim süresine yaklaşık olarak eşit olacak şekilde seçilmelidir.

❖ **Aggarwal / Vitter'in Paralel Disk Modeli;**

- ✓ Genellikle diskler cinsinden ifade edilir, ancak hafıza sıradüzeninin iki bitişik seviyesine uygulanır.
- ✓ Makine bir CPU'ya ve "M" büyüklüğünde bir ana belleğe sahiptir.
- ✓ Ana bellek ile diskler arasındaki veriler "B" büyüklüğünde bloklarla aktarılır.
- ✓ Makine paralel olarak kullanılabilen "D" disklere sahiptir.
- ✓ Bir G/Ç işleminde, ana bellek ve her disk arasında bir B boyutunda blok aktarılabilir.
- ✓ Algoritmalar G/Ç işlemlerinin sayısı açısından analiz edilir.

Merge Sort

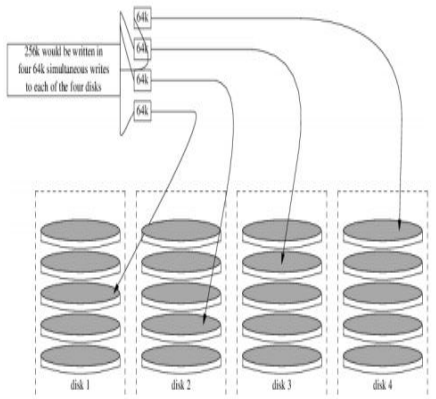
- For N items to be sorted on a buffer of size B each:
  - Merge Sort on a single disk: $2\frac{N}{B}\left(1+\left\lceil log_{M/B}\frac{N}{M}\right\rceil\right)=2\frac{N}{B}\left\lceil log_{M/B}\frac{N}{B}\right\rceil$ disk accesses
  - Merge Sort on D parallel disks: $2\frac{N}{DB}\left\lceil log_{M/(DB)}\frac{N}{DB}\right\rceil$ disk accesses
  - This looks like the internal sorting.
  - Number of blocks is n=N/B. Instead of binary log, we have the logarithm to the memory size measured in number of blocks.
  - How good is this bound?
    - Merge Sort is optimal for one disk, but suboptimal for many disks.

Disk Striping

- We treat the *D* disks as a single disk with block size *DB*. A super-block of size *DB* consists of *D* blocks of size *B*.
- When a super-block is to be transferred, we transfer one standard block to each disk.
- We can generalize all single-disk results to *D* disks.
  - There might be more effective ways of using the *D* disks and that main memory can only hold *M/(DB)* super-blocks.

Disk Striping

- Treat D disks as a single disk with block size DB.
- A super-block of size DB consists of D blocks of size B.



3) *Ders12 - Spatial:*

❖ Mekansal ve Coğrafi Veritabanları:

- ✓ Mekansal veritabanları, mekansal konumlarla ilgili bilgileri depolar ve mekansal verilerin verimli bir şekilde depolanmasını, endekslenmesini ve sorgulanmasını destekler.
- ✓ Özel amaçlı dizin yapıları, mekansal verilere erişmek ve mekansal sorguları işlemek için önemlidir.
- ✓ ***Bilgisayar Destekli Tasarım (CAD)*** veritabanları, nesnelerin nasıl yapıldığı hakkında tasarım bilgilerini depolar. Örneğin; binaların tasarımları, uçaklar, entegre devrelerin düzenleri.
- ✓ Coğrafi veritabanları, coğrafi bilgileri (örneğin haritalar) depolar. Genellikle coğrafi bilgi sistemleri veya GIS olarak adlandırılır.

❖ Mekansal Veri Türleri (Types of Spatial Data):

- ✓ ***Tarama verileri (Raster data),*** iki veya daha fazla boyutta, bit haritalarından veya piksel haritalarından oluşur.
  - ➢ Örnek 2-B raster görüntü: her pikselin belirli bir alanda bulut görünürlüğünü sakladığı bulut kapağının uydu görüntüsü.
  - ➢ Ek boyutlar, farklı bölgelerdeki farklı irtifalardaki sıcaklığı veya zaman içindeki farklı noktalardan alınan ölçümleri içerebilir.
- ✓ Tasarım veritabanları genellikle raster verilerini depolamaz.
- ✓ ***Vektör verileri (Vector data),*** temel geometrik nesnelerden oluşturulmuştur: iki boyutlu noktalar, çizgi parçaları, üçgenler, diğer çokgenler ve üç boyutlu silindirler, küreler, küpler ve diğer polihedronlar.
- ✓ Harita formatını göstermek için sıklıkla kullanılan vektör formatı.
  - ➢ Yollar iki boyutlu olarak kabul edilebilir ve çizgiler ve eğriler ile temsil edilebilir.
  - ➢ Nehirler gibi bazı özellikler, genişliklerinin uygun olup olmadığına bağlı olarak karmaşık eğriler veya karmaşık çokgenler olarak gösterilebilir.
  - ➢ Bölgeler ve göller gibi özellikler çokgenler olarak gösterilebilir.

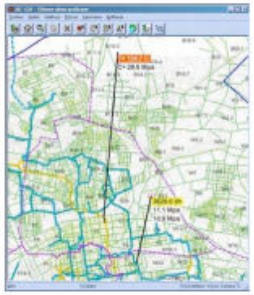
❖ Mekansal Sorgu Çeşitleri (Types of Spatial Queries):

Types of Spatial Queries

- Spatial Range Queries
  - *"Find all cities within 50 miles of Madison"*
  - Query has associated region (location, boundary)
  - Answer includes ovelapping or contained data regions
- Nearest-Neighbour Queries
  - *"Find the 10 cities nearest to Madison"*
  - Results must be ordered by proximity
- Spatial Join Queries
  - *"Find all cities near a lake"*
  - Expensive, join condition involves regions and proximity
- Region queries deal with spatial regions. e.g., ask for objects that lie partially or fully inside a specified region.

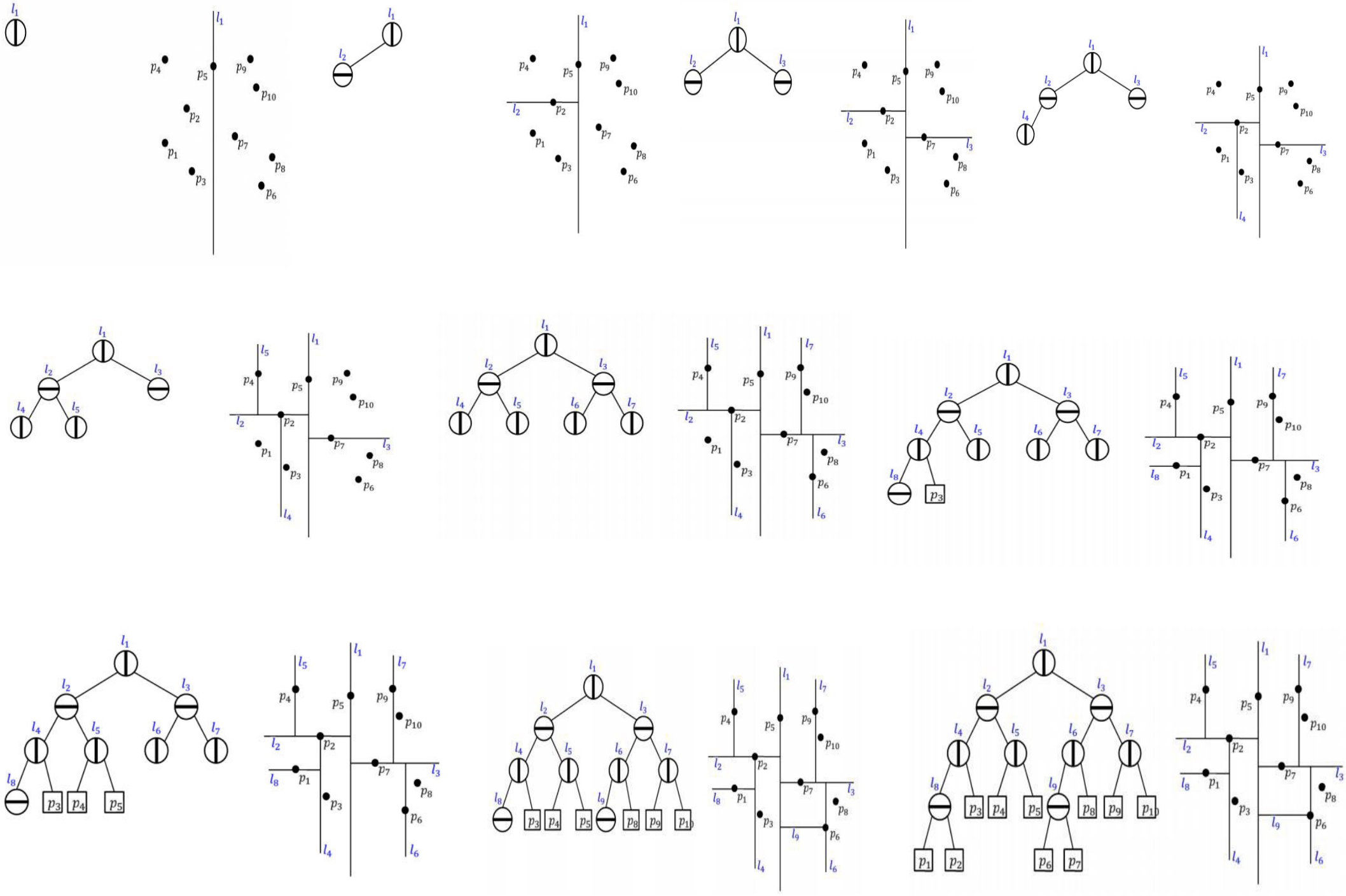
Applications of Spatial Data

- Geographic Information Systems (GIS)
  - E.g., ESRI's ArcInfo; OpenGIS Consortium
  - Geospatial information
  - All classes of spatial queries and data are common
- Computer-Aided Design/Manufacturing
  - Store spatial objects such as surface of airplane fuselage
  - Range queries and spatial join queries are common
- Multimedia Databases
  - Images, video, text, etc. stored and retrieved by content
  - First converted to *feature vector* form; high dimensionality
  - Nearest-neighbor queries are the most common



❖ **Mekansal Verilerin Endekslenmesi (Indexing of Spatial Data):**

- ✓ ***k-d Ağacı;*** çoklu boyutta indeksleme için kullanılan erken yapı.
- ✓ Bir "k-d ağacının" her seviyesi alanı ikiye böler.
  - ➢ Ağacın kök düzeyinde bölümleme için bir boyut seçin.
  - ➢ Bir sonraki seviyedeki düğümlerde bölümleme için başka bir boyut seçin ve böylece, boyutlar arasında geçiş yapın.
- ✓ Her bir düğümde, alt ağaçta depolanan noktaların yaklaşık yarısı bir tarafa, diğer yarısı da diğer tarafa ayrılır.
- ✓ Bir düğüm belirli bir maksimum nokta sayısından daha az noktaya sahip olduğunda, bölümleme durur.
- ✓ ***k-d-B Ağacı,*** her iç düğüm için birden fazla alt düğüme izin vermek için k-d ağacını uzatır; ikincil depolama için çok uygundur.

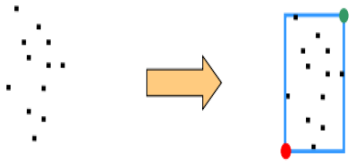


❖ **R-Ağacı;**

- ✓ R ağacı, ekleme ve silme işlemlerinde dengeli kalan, ağaç yapılı bir dizindir.
- ✓ Bir yaprak girişinde depolanan her anahtar sezgisel olarak bir kutu veya bir aralık; boyut başına bir aralıktır.
- ✓ R ağacı, verileri minimum sınırlayıcı kutuyla temsil ederek herhangi bir boyutlu verileri düzenleyebilir.
- ✓ Her düğüm çocuğuna bağlanır. Bir düğümde birçok nesne olabilir.
- ✓ Yapraklar gerçek nesnelere işaret eder (muhtemelen diskte depolanır).
- ✓ Yükseklik her zaman "log n" (yükseklik dengelidir).

Bounding Rectangle

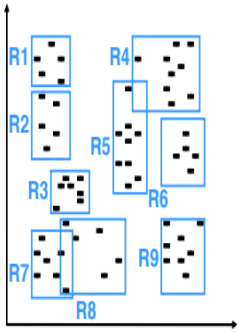
- Suppose we have a cluster of points in 2-D space...
  - We can build a "box" around points. The smallest box (which is axis parallel) that contains all the points is called a Minimum Bounding Rectangle (MBR)
    - also known as minimum bounding box



$$MBR = \{(L.x, L.y)(U.x, U.y)\}$$

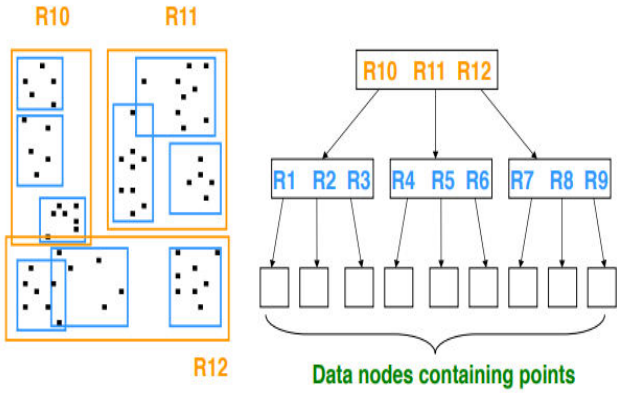
Clustering Points

- We can group clusters of datapoints into MBRs
  - Can also handle line-segments, rectangles, polygons, in addition to points



We can further recursively group MBRs into larger MBRs....

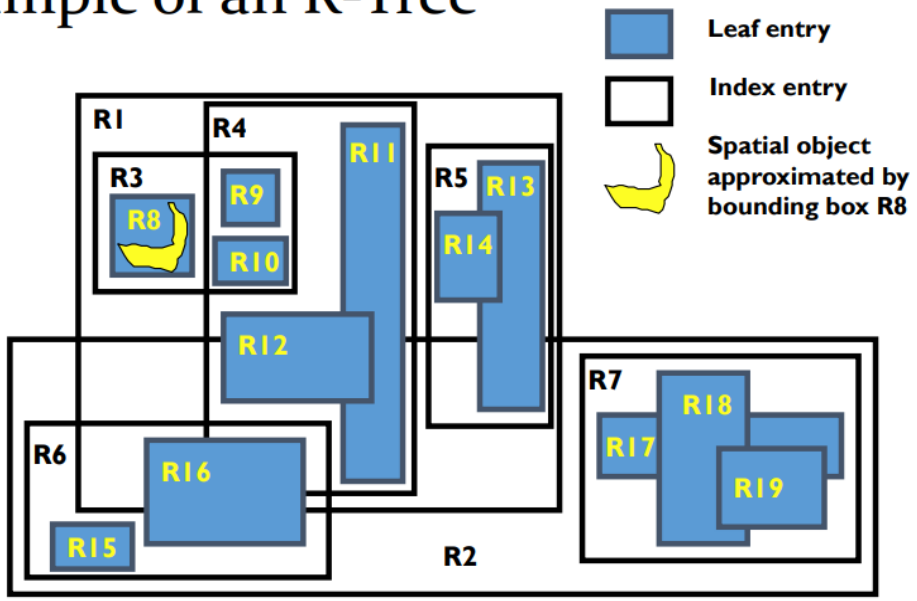
R-Tree Structure



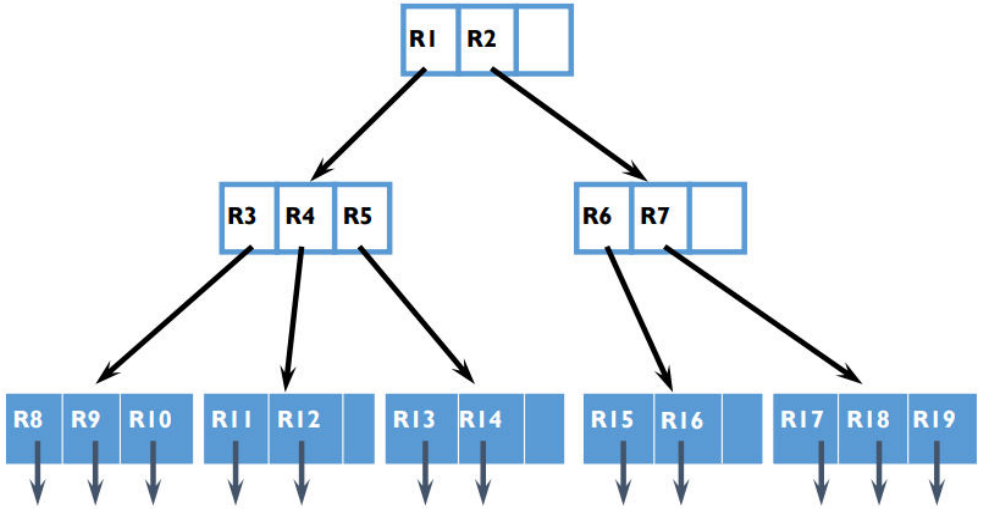
- ✓ Her ağaç düğümü, ***dikdörtgen bir sınırlama kutusu (Bounding box)*** ile (a.k.a. MBR) ilişkilidir.
  - ➢ Bir yaprak düğümünün ***sınırlayıcı kutusu (Bounding box),*** yaprak düğümü ile ilişkili tüm dikdörtgenleri / çokgenleri içeren minimum boyutlu bir dikdörtgendir.
  - ➢ Yapraksız bir düğümle ilişkili ***sınırlayıcı kutu (Bounding box),*** tüm çocukları ile ilişkili sınırlayıcı kutu içerir.
  - ➢ Bir düğümün ***sınırlayıcı kutusu (Bounding box),*** ana düğümünde (varsa) anahtarı olarak işlev görür.
  - ➢ Bir düğümün çocuklarının ***sınırlayıcı kutuları (Bounding box)*** üst üste gelebilir.
- ✓ Bir çokgen yalnızca bir düğümde depolanır ve düğümün ***sınırlayıcı kutusu (Bounding box)*** çokgeni içermelidir.

✓ Depolama verimliliği veya R-ağaçları, bir poligonu yalnızca bir kez depoladığından, “k-d ağaçlarının” veya “dörtlü ağaçlarınkinden (quadtrees)” daha iyidir.

Example of an R-Tree



Example R-Tree (Contd.)



Search for Objects Overlapping

Start at root.

1. If current node is non-leaf, for each entry <E, ptr>, if ***box*** E overlaps Q, search subtree identified by ptr.
2. If current node is leaf, for each entry <E, rid>, if E overlaps Q, rid identifies an object that might overlap Q.

Note: May have to search several subtrees at each node! (In contrast, a B-tree equality search goes to just one leaf.)

Splitting a Node During Insertion

- The entries in node L plus the newly inserted entry must be distributed between L1 and L2.
- Goal is to reduce likelihood of both L1 and L2 being searched on subsequent queries.
- Idea: Redistribute so as to minimize area of L1 plus area of L2.

Exhaustive algorithm is too slow; quadratic and linear heuristics are described in the paper.