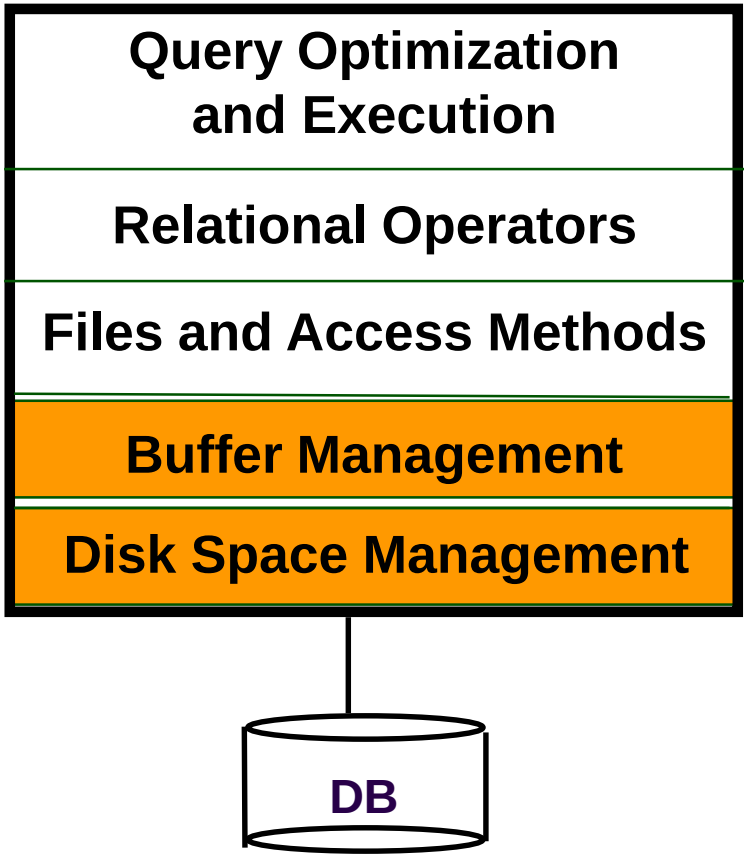
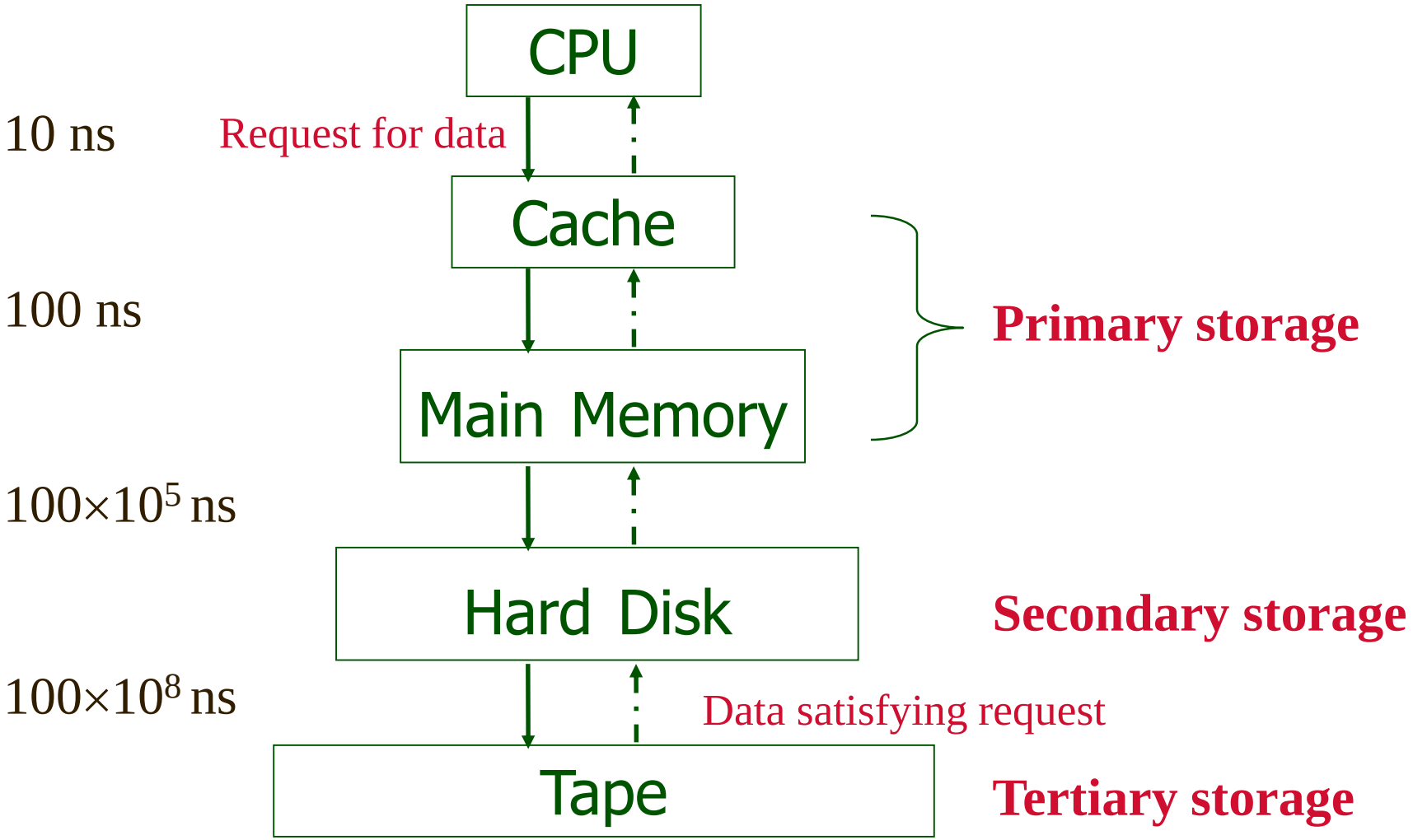
COMP7640

Data Storage and Access Methods

DBMS Architecture



Data Storage



Store Everything in Main Memory?

- Amount of data is very large
- Main memory is expensive
  - You can use roughly HK$500 to buy either 16GB of RAM or 2000GB (2TB) of hard disk today (http://www.price.com.hk/)
- Main memory is volatile
  - We want data to be saved between runs

Typical Storage Hierarchy

- Main memory (random access memory, RAM) for currently used data
- Disk (secondary storage) for the main database
- Tape (tertiary storage) for archiving older versions or infrequently accessed data

RAM

Disk

Tape

Disks

- Preferred secondary storage device
- Data is stored and retrieved in units called *disk blocks* or *pages*

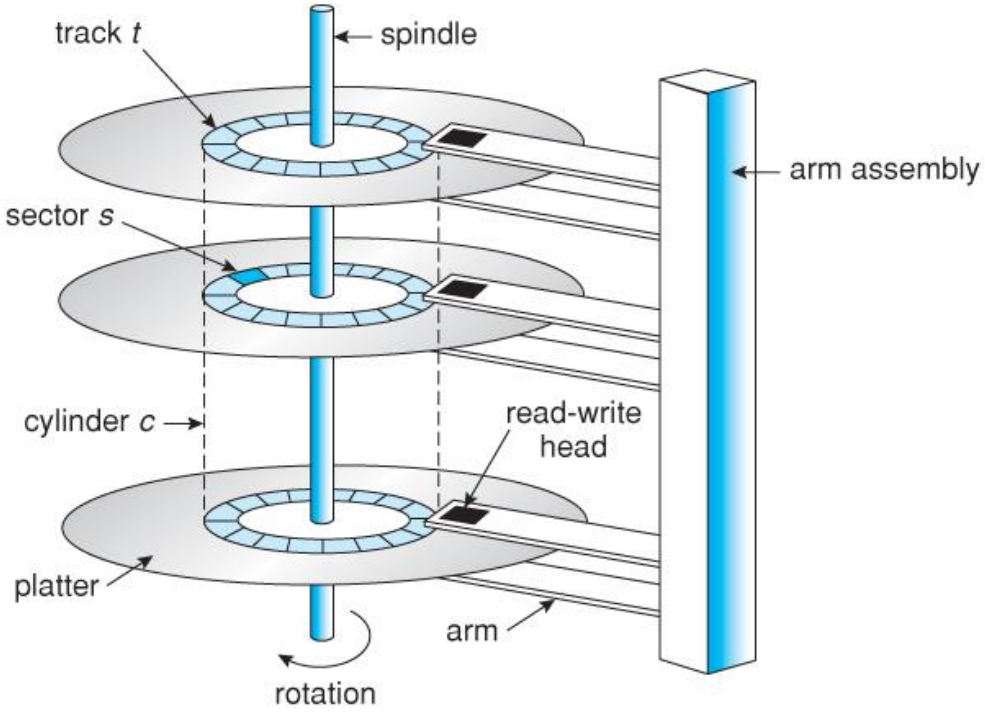
Disk Operations

- Read: transfer data from disk to RAM
- Write: transfer data from RAM to disk

- Both are high-cost operations, relative to in-memory operations, so must be planned carefully!

Components of a Disk

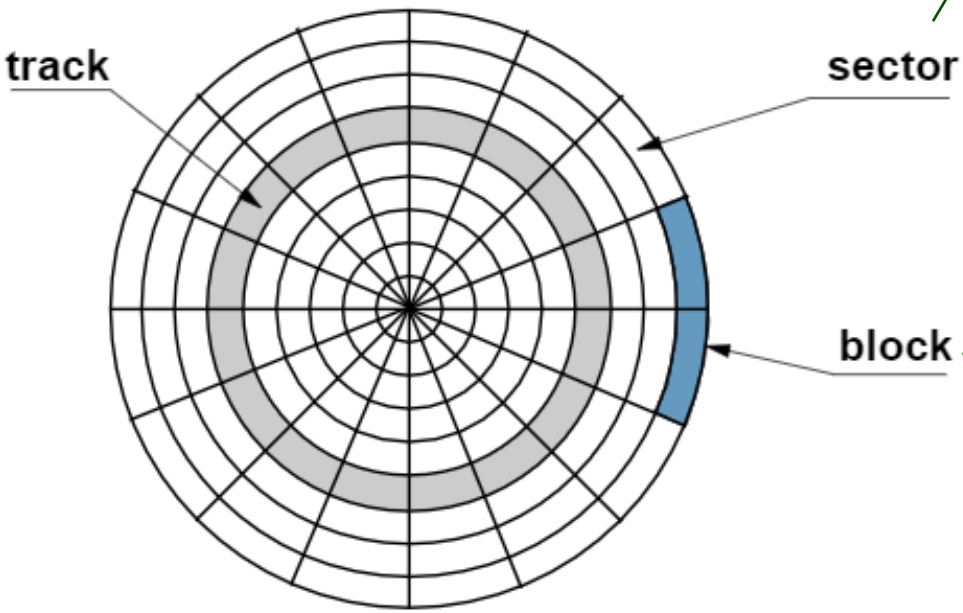
- The *spindle* makes platters spin.
- Each *platter* contains multiple *tracks*
- The *arm assembly* is moved in or out to position a head on a desired track.
- A *cylinder* is the set of all tracks that all the heads are currently located at. (all the heads can read/write)



Top View

Each track is divided into multiple sectors.
Each sector is an actual physical area
the smallest logical unit of the disk

sector size is fixed
(e.g., 512 bytes)



A unit of space allocation in a filesystem. It contains multiple sectors and its size varies in different OS (typically 4KB).

Accessing a Disk Page

- Time to access (read/write) a disk block:
  - Seek time (for locating the track)
    - moving arms to position disk head on track
    - varies from about 1 to 20ms
  - Rotational delay (for locating the block)
    - waiting for block to rotate under head
    - varies from 0 to 10ms
  - Transfer time (for moving the data)
    - actually moving data to/from the disk
    - about 1ms per 4KB page

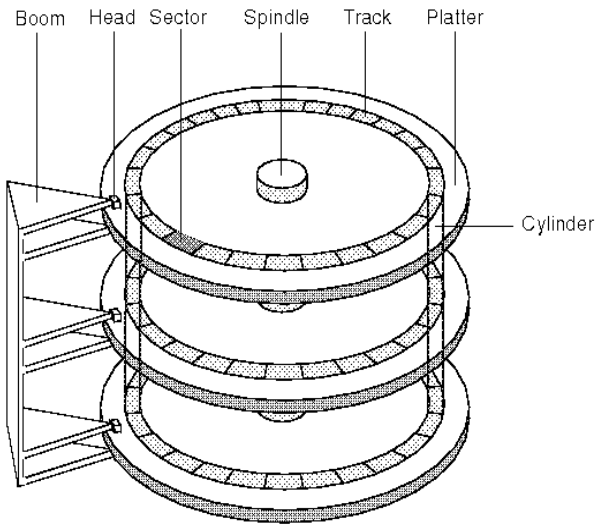
Arranging Pages on Disk

- *Seek time* and *rotational delay* performance dominate!
- Relative placement of pages on disk has major impact on DBMS performance
- Key to lower I/O cost:
  - Reduce seek/rotation delays!
  - Place the pages, that are frequently accessed together, close to each other on disk

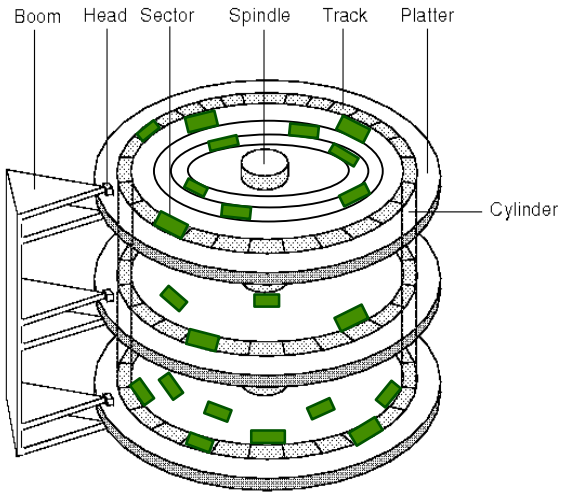
Example

- Compute the time taken to read a 2,048,000-byte file that is divided into 8,000 256-byte records assuming the following disk characteristics:

average seek time	18ms
track-to-track seek time	**5ms (for adjacent tracks)**
rotational delay	8.3ms
max transfer rate	16 ms/track
sector size	512 bytes
sectors/track	40
tracks/cylinder	10
tracks/surface	1,331



Store records randomly

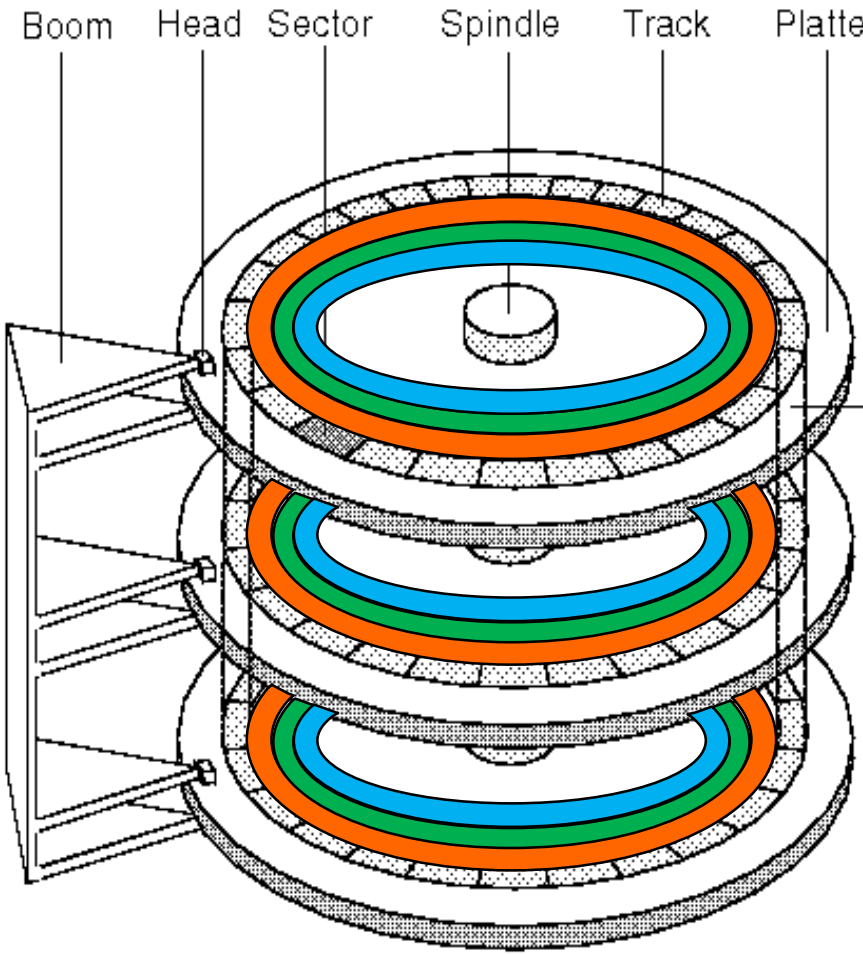


Records are on the sectors on *different tracks of different platters*

Any two records can be on two different sectors

- Reading the file requires 8,000 random accesses
- Each access time
  18 (avg. seek) + 8.3 (rotational delay) + 0.4 (transfer one sector 16ms/40) = 26.7 ms
- Total access time
  = 8,000*26.7 = 213,600 ms = 213.6 s

Store records by "next"



Records are on the sectors on *3 adjacent cylinders*
(3 adjacent tracks on each platter)

After reading/writing one cylinder (vertically adj tracks), move the heads to its adjacent cylinder (vertically adj tracks).

seek time = track-to-track seek time

1st cylinder, 2nd cylinder, 3rd cylinder

Store records by "next"

- 1 track contains 40 * 512 = 20, 480 bytes
- File needs 100 tracks = 10 cylinders
- Read first cylinder
  = 18 + 8.3 + 10*16 = 186.3 ms
- Read next 9 cylinders
  = 9 * (5 + 8.3 + 10*16) = 1,559.7 ms
- Total access time
  = 186.3 + 1,559.7 = 1,746 ms = 1.746 s

- *Blocks in a file should be arranged sequentially on disk to minimize seek and rotational delay.*

Summary

- Disks provide cheap, non-volatile storage
  - Random access
  - Cost depends on location of page on disk
  - Important to arrange data sequentially to minimize *seek* and *rotation* delays

Question 1

- Assume the disk has the following characteristics.

average seek time	20 ms
track-to-track seek time	4 ms
rotational delay	13.6 ms
max transfer rate	27.2 ms/track
bytes/sector	1024
sectors/track	50
tracks/cylinder	5
tracks/surface	1,000

Suppose that the file contains 1000 256-byte records. Calculate the time incurred by transferring all these records from the disk to the main memory if:
a) the "random" strategy is adopted.
b) the "next" strategy is adopted.

Question 2

- Answer Question 1 if there are 100,000 256-byte records and the tracks/cylinder is increased to 10.

Record and File Organization

Data Items

Records

Pages

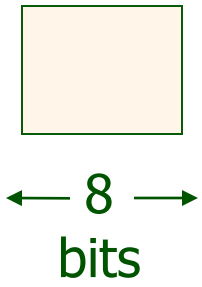
Database Files

Data Item

What are the data items we want to store?

- A name
- An ID
- A date
- A time

⇨ What we have available: bytes

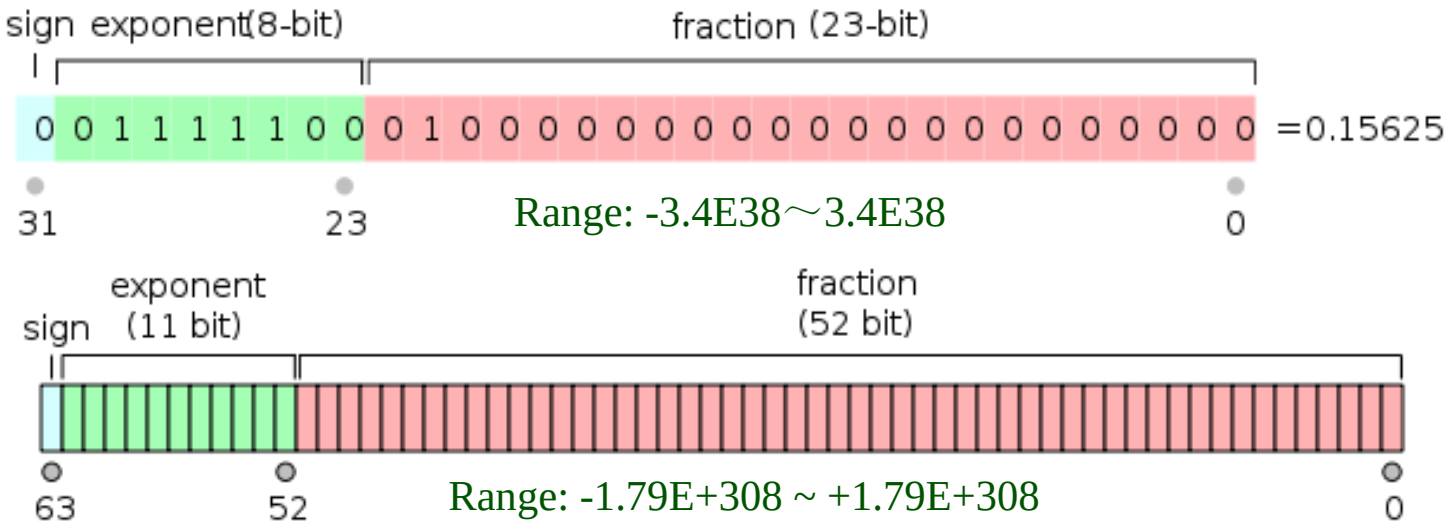


To Represent:

- Integer (short): 2 bytes, e.g., 35 is

00000000 00100011 $= 2^5 + 2^1 + 2^0 = 35$

- Real, floating number: *n* bits for exponent, *m* bits for fraction



To Represent:

- Characters (1byte)

  → Various coding schemes available,
  most popular is ASCII (http://www.asciitable.com/)

  Example:

  A: 01000001 (65)
  a: 01100001 (97)
  5: 00110101 (53)

- Boolean (1byte)

  e.g., TRUE 1111 1111
  FALSE 0000 0000

To Represent:

- Date

  e.g.: - Integer, # days since Jan 1, 1900
  - 8 characters, YYYYMMDD
  - 7 characters, YYYYDDD

- Time

  e.g.: - Integer, seconds since midnight
  - characters, HHMMSS

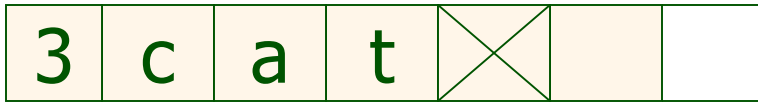
To Represent:

- String of characters
  - Null terminated

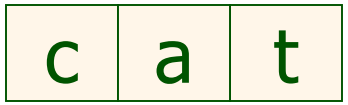
    e.g.,

  - Length given

    e.g.,



  - Fixed length



Key points for data item

- Fixed length items
- Variable length items
  - usually length given at beginning
- Type of an item
  - tells us how to interpret an item

Records

- Record: collection of related data items (called ***fields*** or ***attributes***)

  E.g.: MovieStar record:

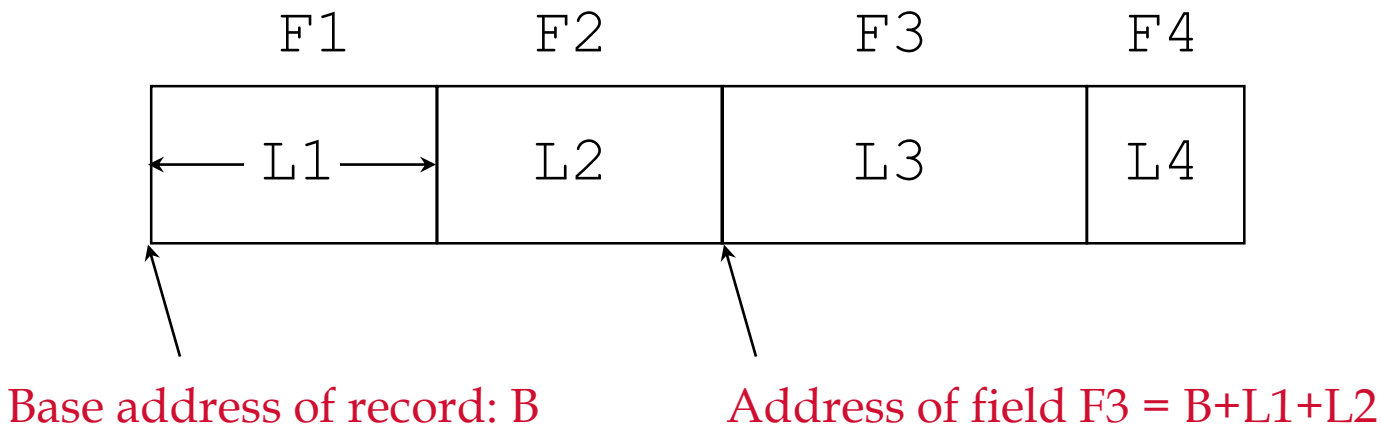
  name field,

  gender field,

  birthday field, ...

  - (Tom Hanks, Male, 1956, …)
- How to organize field within a record?
  - To efficiently retrieve or modify fields in a record

Fixed Length Records



- Each field has fixed length
  - Information about field types and lengths is stored in system catalogs
- Number of fields is fixed
- Store fields consecutively

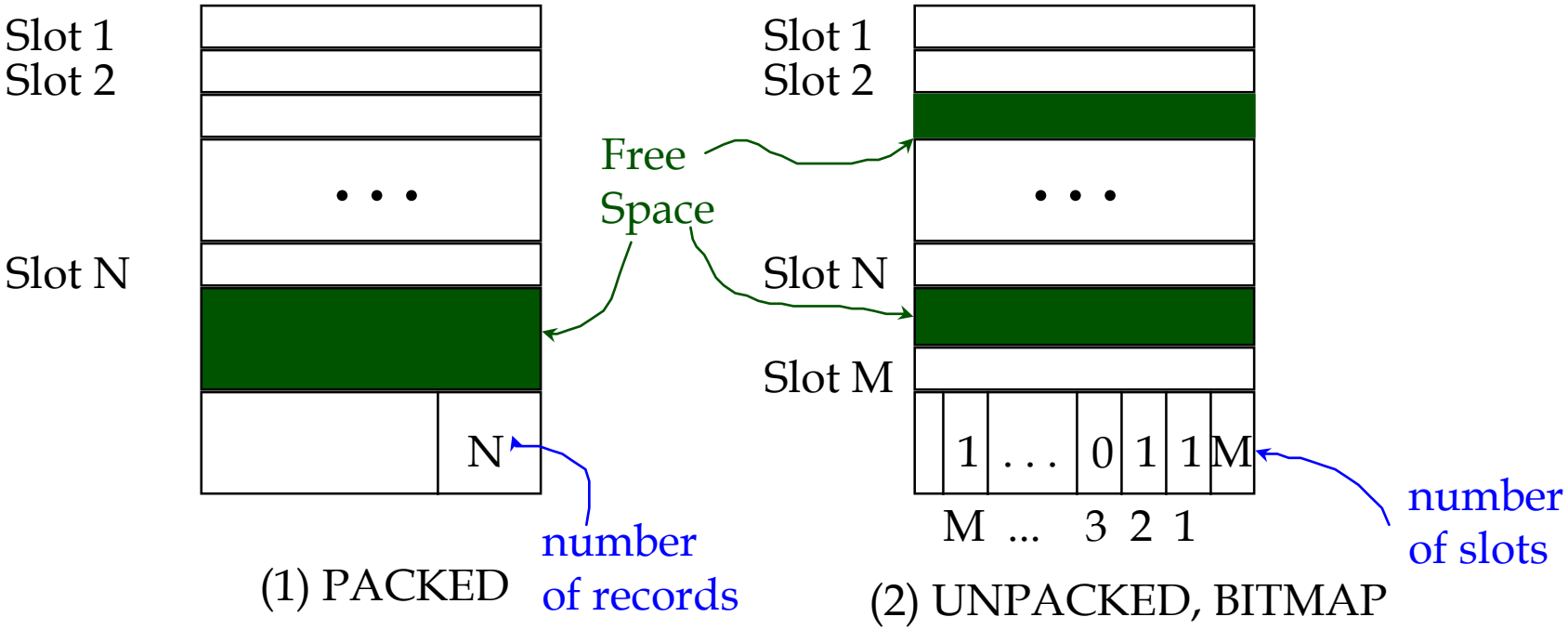
Fixed Length Records (Example)

- Example MovieStar relation
  - name: char(30), 30 bytes
  - address: char(255)
  - gender: 1 byte
  - birthday: 10 bytes
- Record of type MovieStar will take 30 + 255 + 1 + 10 = 296 bytes

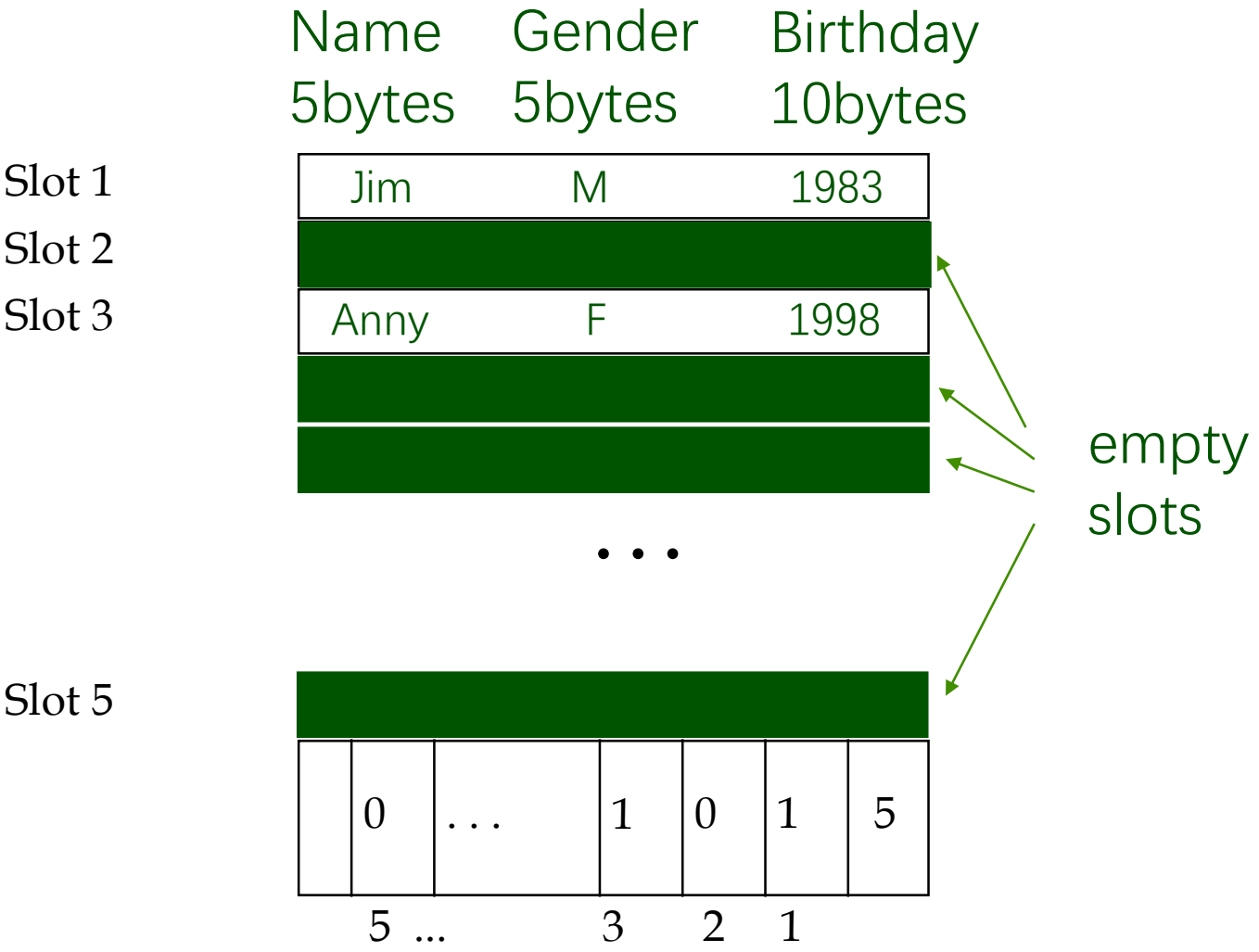
Page Formats

- Page
  - Collection of slots each containing a record
- Record id
  - Logical id *<pageID, slot#>*
  - Physical id *<deviceID, cylinder#, track#, block#, offset in block>*
- How to manage slots on a page?
  - Search, insert, delete records on a page

Page Formats: Fixed Length Records



Example


Name
5bytes
Gender
5bytes
Birthday
10bytes
Slot 1
Slot 2
Slot 3
Slot 5
Jim
M
1983
Anny
F
1998
empty
slots
0
. . .
1
0
1
5
5 ...
3
2
1

File Structure

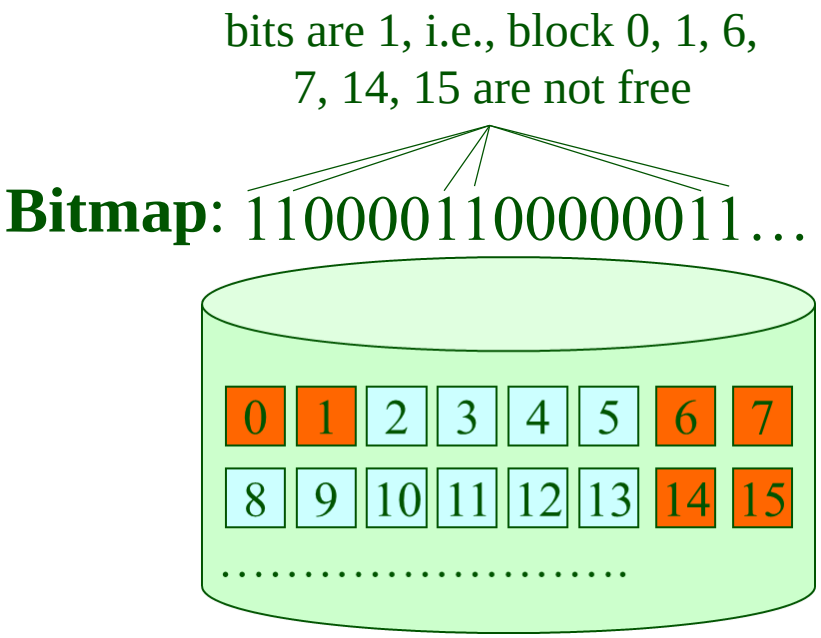
- File
  - Collection of pages, each containing a collection of records
    - Every record has a unique ***record id (rid)***
    - Every page in a file is of the same size
- File structure must support:
  - insert/delete/modify record
  - read a particular record (specified using *rid*)
  - scan all records (possibly with some conditions on the records to be retrieved)

Disk Space Management

- Disk space manager manages space on disk
  - A page is a concept unit of data
  - Allocate/de-allocate a page
  - Read/write a page
- Many files will be stored on a single disk
- Allocate a *sequence* of pages for a file as a contiguous sequence of blocks on disk
  - Disk space is effectively utilized
  - Files can be quickly accessed

Disk Space Management

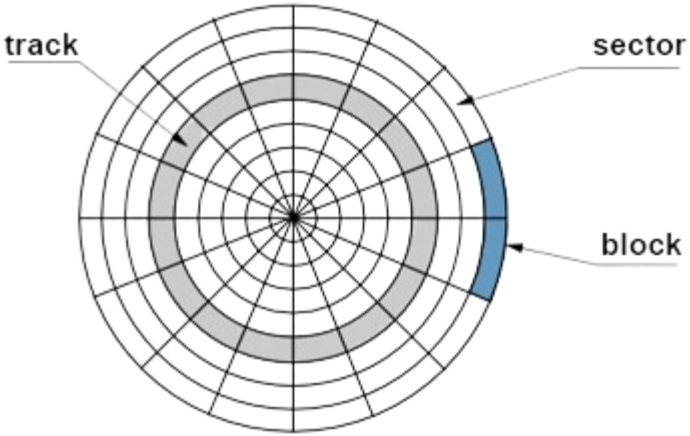
- Management of free space in a disk
  - System maintains a ***list of free blocks***
  - Implemented as ***bitmaps or linked lists***



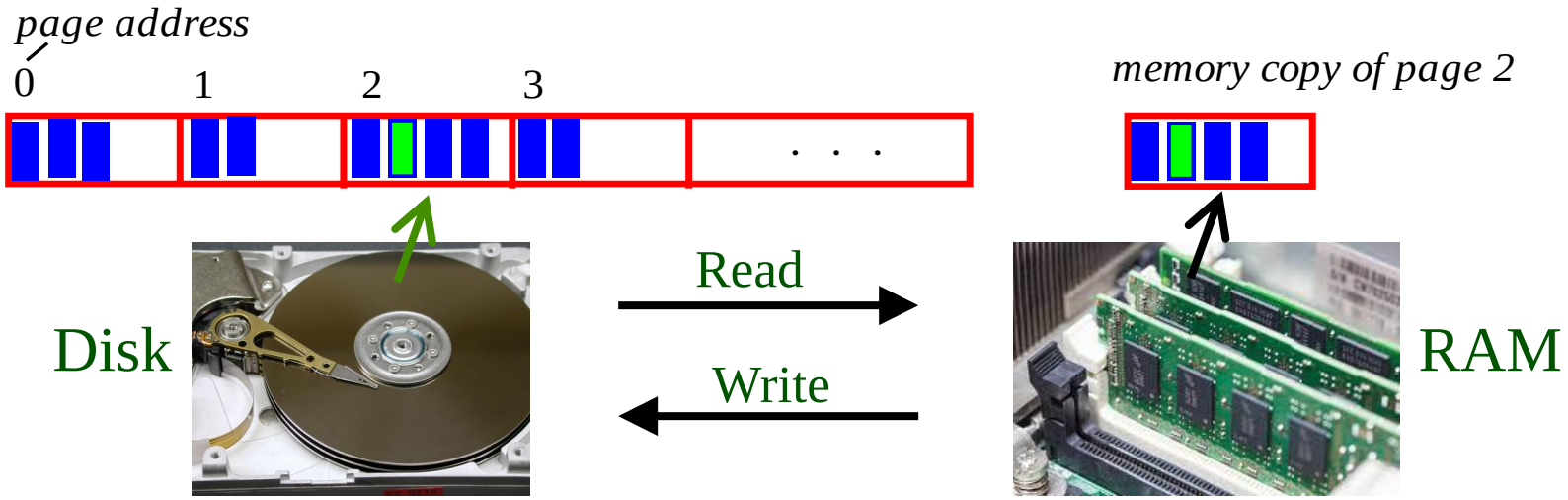
To allocate space, scan the bitmap for 0s

Disk Access

- The database reads from (writes to) the disk always in pages (*also known as blocks*).
  - A page is the smallest unit of information transferred between memory and disk.
- Each page (block) occupies the same number of bytes.
  - Common page sizes: 4k bytes, 8k bytes.



Reading/Writing a Record



- DB needs to change the information of the green record.
  - Read the page 2 into the memory;
  - Update the green record in the memory;
  - Write the whole memory page back to page 2 on the disk.

Alternative File Organizations

- File organization: Method of arranging a file of records on external storage (i.e., disk).
- Many alternatives exist, *each ideal for some situations, and not so good in others:*
  - Heap (random order) files: Suitable when typical access is a file scan retrieving all records.
  - Sorted Files: Best if records must be retrieved in some order, or only a `range' of records is needed.

Example Relation Instance

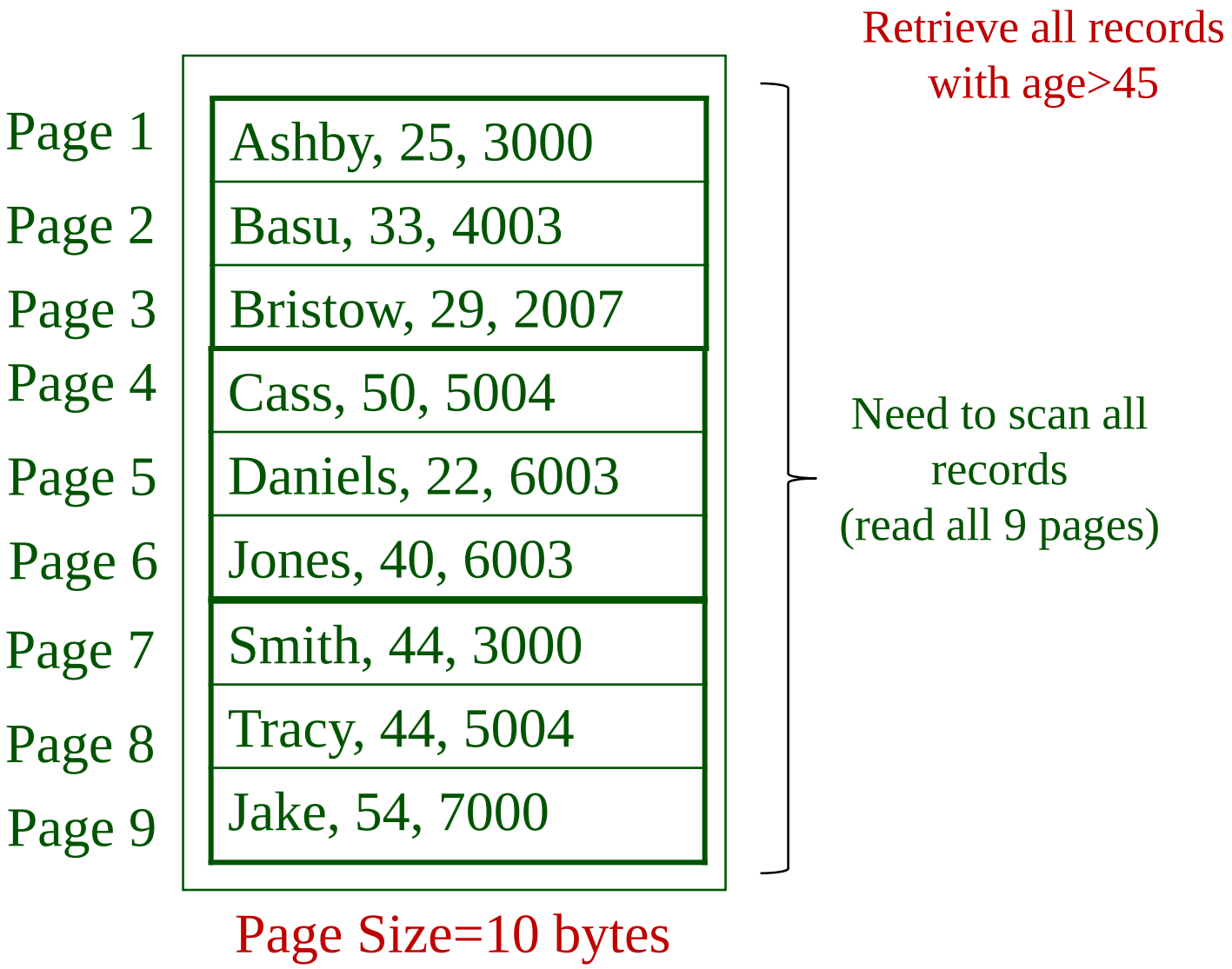
Employee (90 bytes in total)

Name (4 bytes)	Age (2 bytes)	Salary (4 bytes)
Ashby	25	3000
Basu	33	4003
Bristow	29	2007
Cass	50	5004
Daniels	22	6003
Jones	40	6003
Smith	44	3000
Tracy	44	5004
Jake	54	7000

9 Records

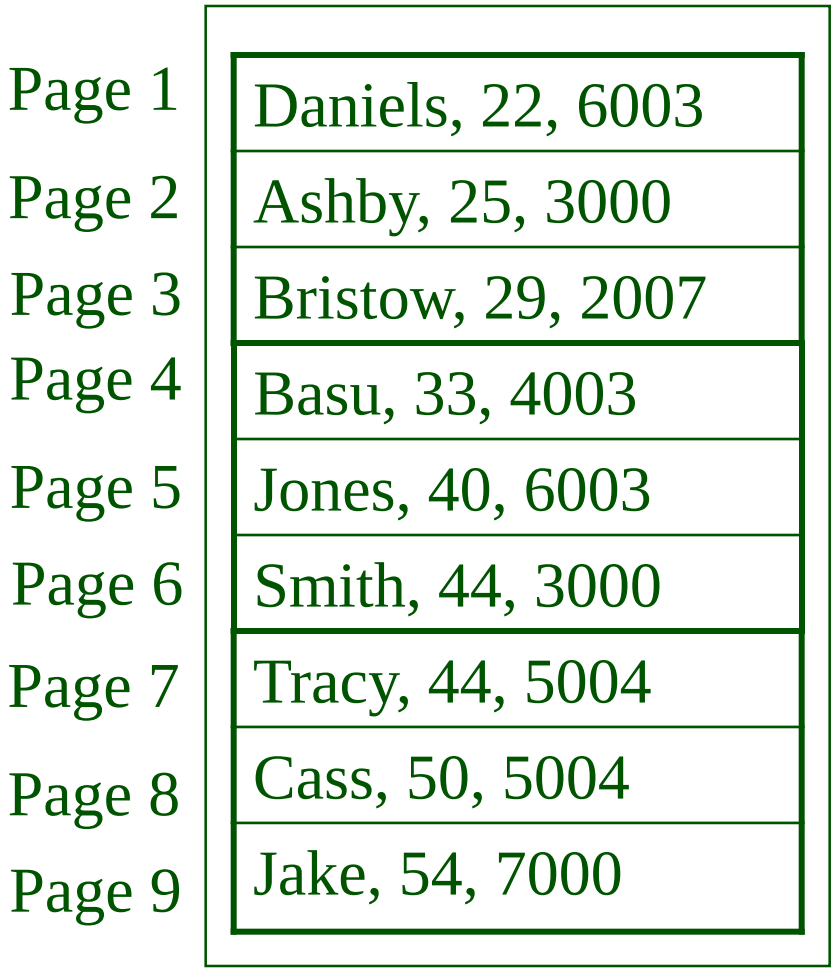
10 bytes

Heap File (Random Order)



Sorted File (by Age)

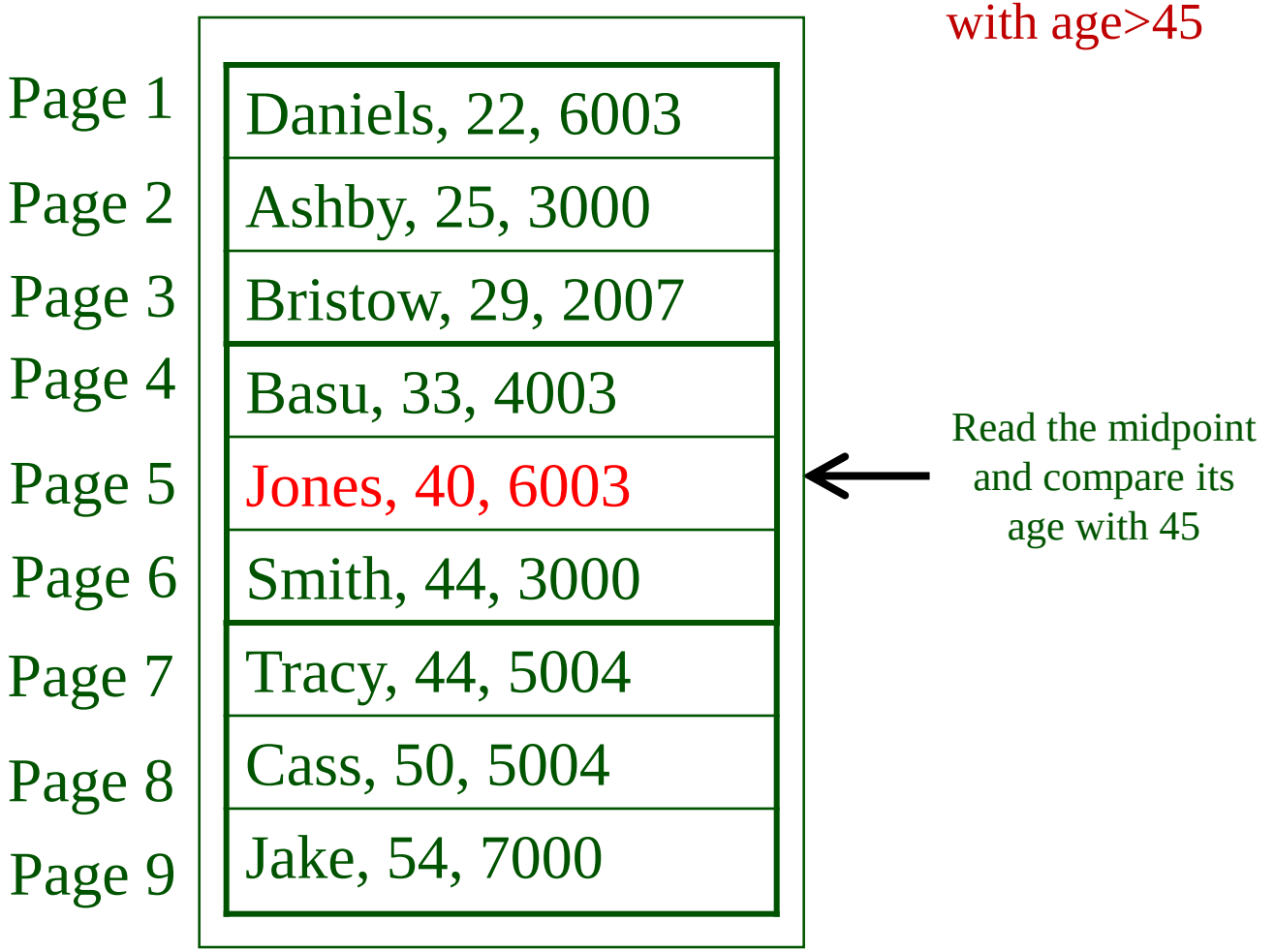
Retrieve all records with age>45



Page Size=10 bytes

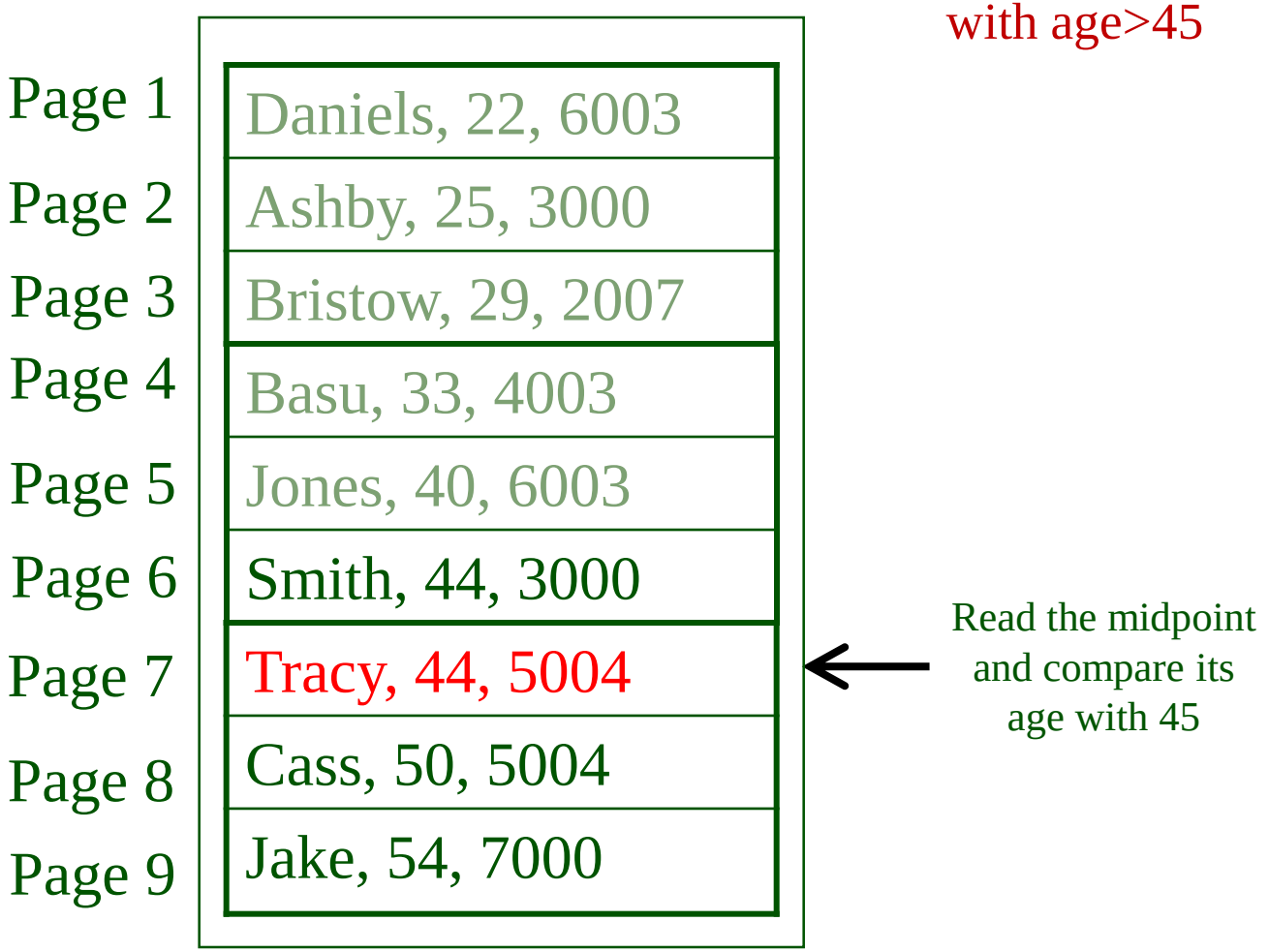
Binary Search on Sorted File (by Age)

Retrieve all records with age>45



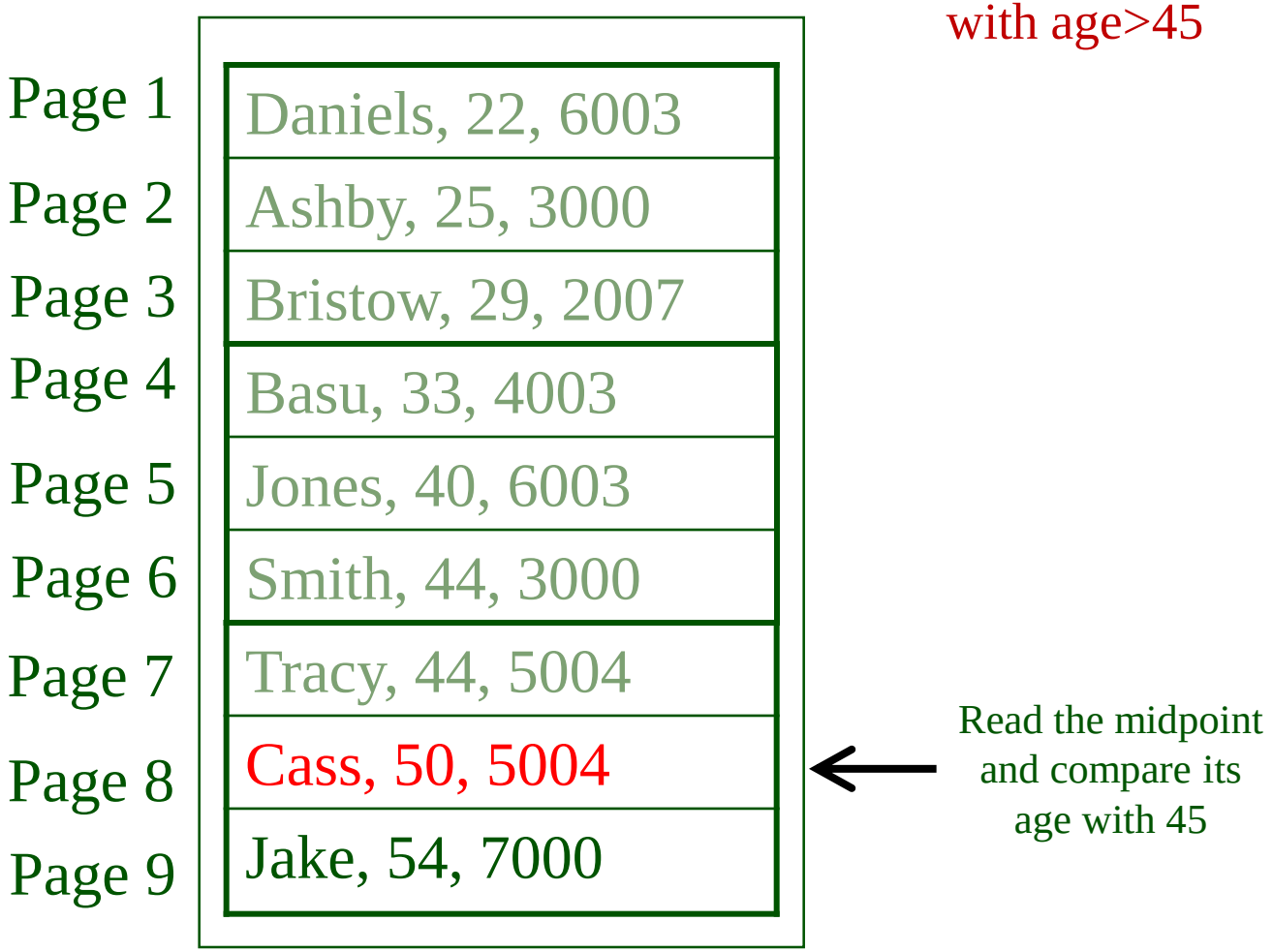
Binary Search on Sorted File (by Age)

Retrieve all records with age>45

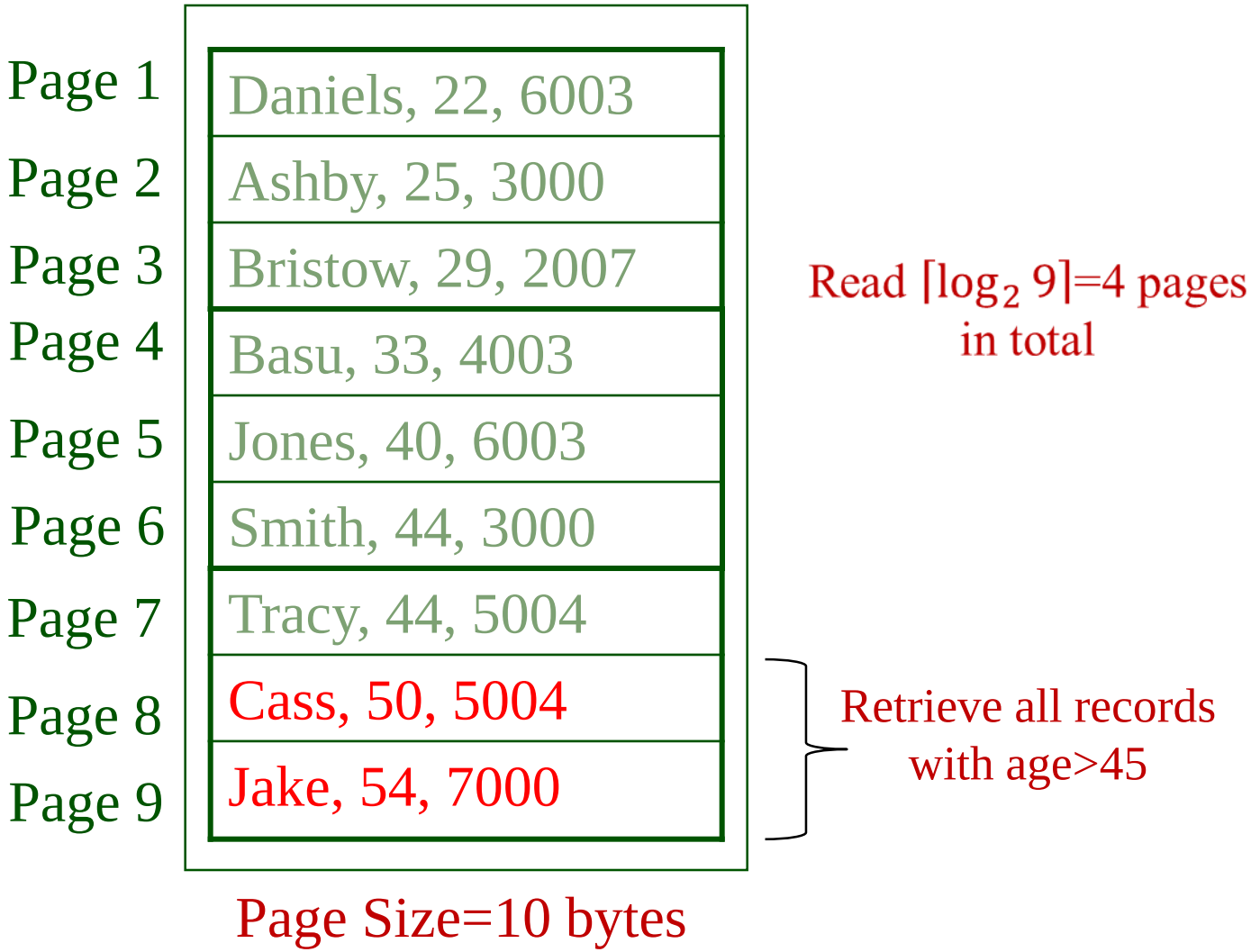


Binary Search on Sorted File (by Age)

Retrieve all records with age>45



Binary Search on Sorted File (by Age)

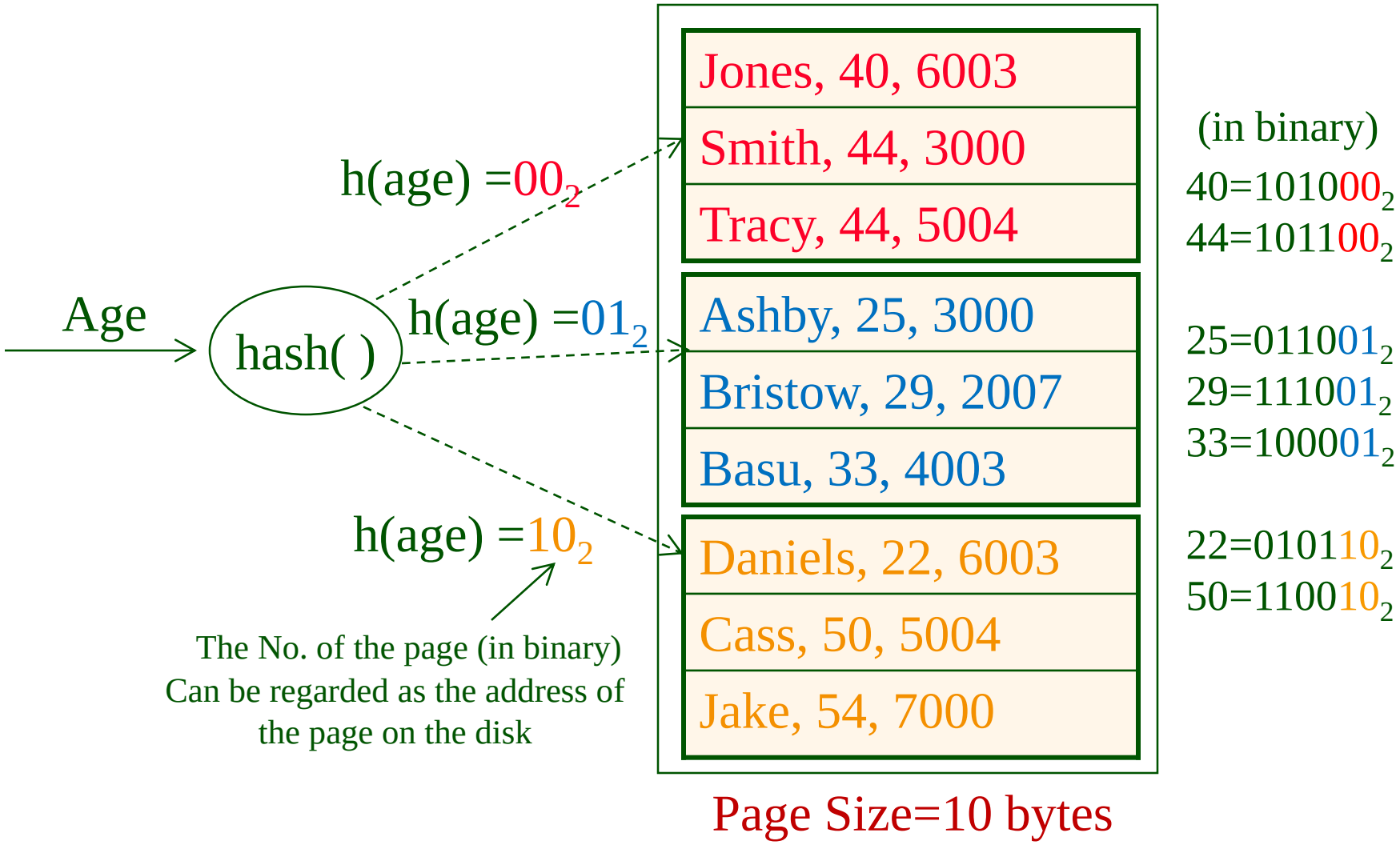


Alternative File Organizations (cont'd)

- Hashed Files: Good for equality selections.
  - File is a collection of buckets
    - **Bucket** = primary page plus zero or more overflow pages
  - Hashing function **h**:
    - **h(r)** = the No. of the bucket to which record ***r*** belongs. **h(r)** looks at only some of the fields of ***r***, i.e., *search fields*.

the No. of the bucket stands for the address of the primary page on the disk

Hashed File (on Age field)



Cost Model for Analysis

- Measure *the number of pages* that need to be read or written, i.e., **the I/O cost**

- Why not CPU cost?
  - In practice, the disk overhead dominates the total access time
  - On average, a page access takes 10ms, during which today's CPU can do a large amount of work

- Also ignore the effect of caching; thus, even the I/O cost is only approximated

  * *Good enough to show the overall trends!*

Single Record and Range Search

- Single record search
  - "Find employee name whose age= 22"
- Range search
  - "Find all employees with age between 20 and 30"
  - Heap File or Hashed File: sequentially scanning
  - Sorted File:
    - Binary search to find first such employee (*cost of binary search can still be quite high!!!*)
    - Scan to find others

Question 3

CREATE TABLE **Employee**
(Name CHAR(20),
Age INTEGER,
Salary REAL,
Address CHAR(96))

INTEGER=4 bytes REAL=8 bytes
1 byte per character
Page size = 4k bytes = 4096 bytes

Employee (stored as a heap file)

Name	Age	Salary	Address
…	…	…	…
Basu	33	4003	Yau Ma Tei
Bristow	29	2007	Mong Kok
Cass	50	5004	Central
Daniels	22	6003	Tsim Sha Tsui
Jones	40	4003	Tai Wai
…	…	…	…

3200 records

- Given the table Employee created by the above SQL, which has 3,200 employee records
  - What is the I/O cost for reading the records of all employees whose salaries are below $5,000?
  - What is the *average* I/O cost for reading the record of "Bob"? (only one Bob here)

I/O Cost of Operations (An Example)

- The size of the file: 10G bytes
- **B:** The number of data pages =2.5M
- The size of a data page: 4k bytes
- **D:** (Average) time to read or write a disk page =20ms

	Heap File	**Sorted File**	**Hashed File**
Scan all records	BD 14 hours	BD 14 hours	1.25BD 17.5 hours
Equality Search	0.5BD 7 hours	$D\log_2 B$ 0.4 sec	D 20 ms
Range Search	BD 14 hours	$D(\log_2 B$+# of pages with matches) > 0.4 sec	1.25BD 17.5 hours
Single record insert	2D 40 ms	Search + 2 * (0.5*BD) 14 hours	2D 40 ms
Single record delete	Search + D 7 hours	Search + BD 14 hours	2D 40 ms

** Several assumptions underlie these (**rough**) estimates!*

Conclusion

- No perfect file organizations
  - Heap file is bad in the equality search, range search, and record deletion
  - Sorted file is bad in record insertion and deletion
  - Hashed file is bad in scanning and range search
- Searching for the records *directly* on the file is not a good choice!
  - Need auxiliary data structures

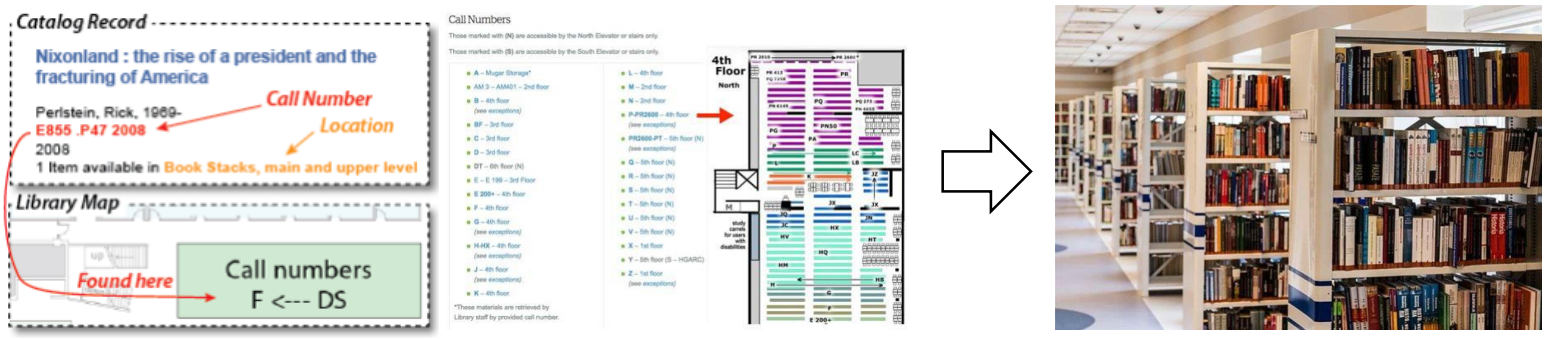
Inspiration: A Real-world Case

- How do you locate the book "Database Management Systems" in the HKBU library?
  - Going through every book on each bookshelf?

HKBU library

Inspiration: A Real-world Case

- How do you locate the book "Database Management Systems" in the HKBU library?
  - Use the **catalog (an index)**
    - Given the book title
    - Give us Which floor? Which bookshelf? Which row?



Catalog **HKBU library**

- Library as the disk, row as the page, book as the record

Indexes

- An **index** is a data structure that allows us to *quickly* find the records with given values in *search key fields*
  - e.g., author catalog in library search
  - Search key fields can be any subset of the fields of the relation (e.g., {*salary*}, {*age*}, or {*salary, age*})

    SELECT * FROM E WHERE E.age < "40"
    SELECT * FROM E WHERE E.salary > "3,000" AND E.age = "44"

  - Reduce I/O cost
    - Index is much smaller than the data

  - *Note: Search key* is **not** the same as *primary key* (which is the minimal set of fields that uniquely identify a record in a relation)

Indexes (cont'd)

- An index is a collection of ***data entries*** that support efficient retrieval of all data entries ***k**** with a given key value ***k***.

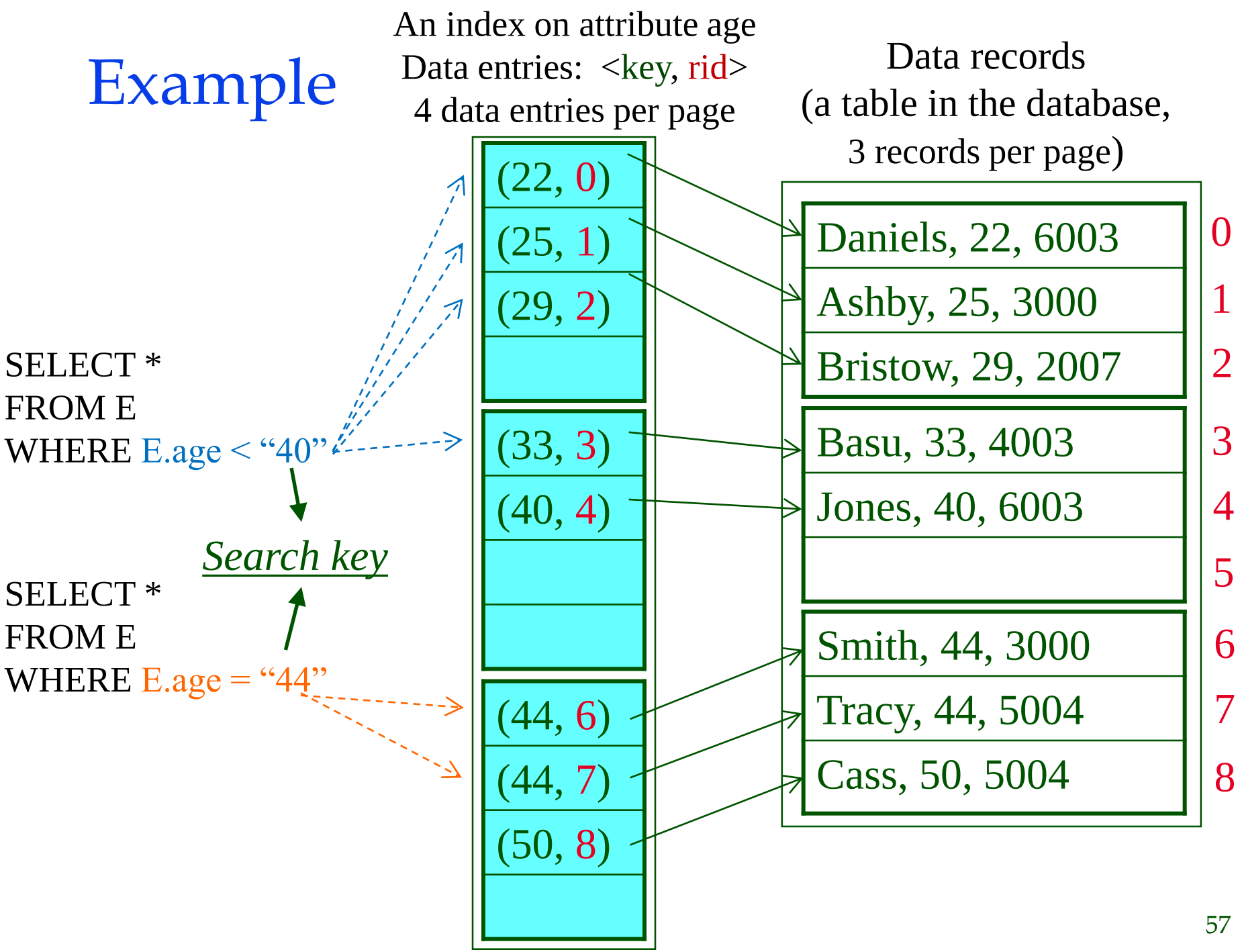
- Two issues:
  - What is stored as a *data entry k** in an index?
  - How are data entries organized?

What is a data entry *k** in Index?

- Two main alternatives:
  - Alternative 1
    - Each data entry stores a <key, rid> pair, indicating it is associated with a data record. (rid=record id)
      - <*k*, rid of the data record with search key value *k*>
    - The data records (a table) are stored separately in the database.
    - Each data entry is *much smaller* than a data record.
  - Alternative 2
    - <*k*, list of rids of data records with search key *k*>

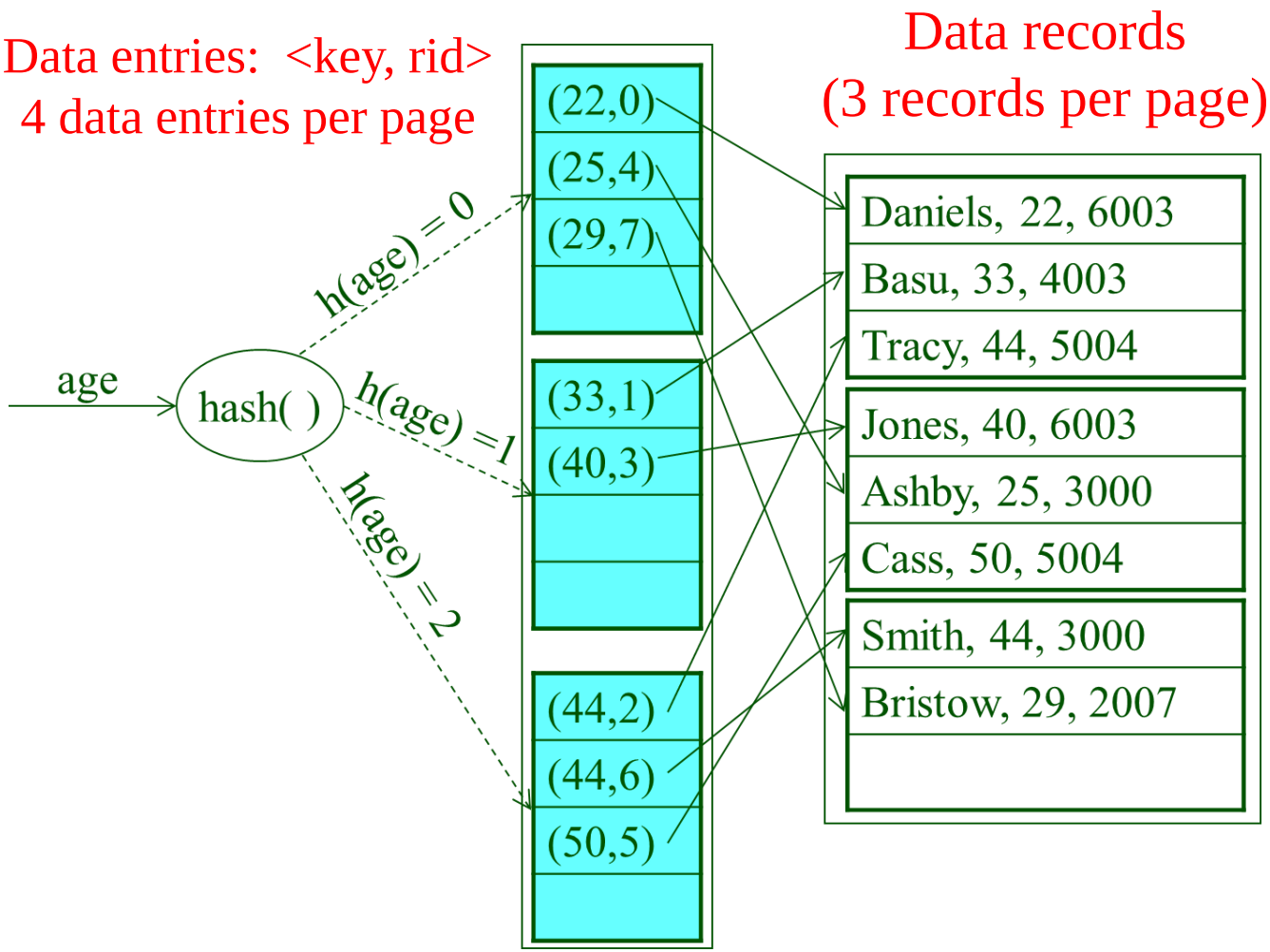
Record id (rid) is sufficient to physically locate records

Example


An index on attribute age
Data entries: <key, rid>
4 data entries per page
Data records
(a table in the database,
3 records per page)
(22, 0)
(25, 1)
(29, 2)
(33, 3)
(40, 4)
(44, 6)
(44, 7)
(50, 8)
Daniels, 22, 6003
Ashby, 25, 3000
Bristow, 29, 2007
Basu, 33, 4003
Jones, 40, 6003
Smith, 44, 3000
Tracy, 44, 5004
Cass, 50, 5004
0
1
2
3
4
5
6
7
8
SELECT *
FROM E
WHERE E.age < "40"
Search key
SELECT *
FROM E
WHERE E.age = "44"

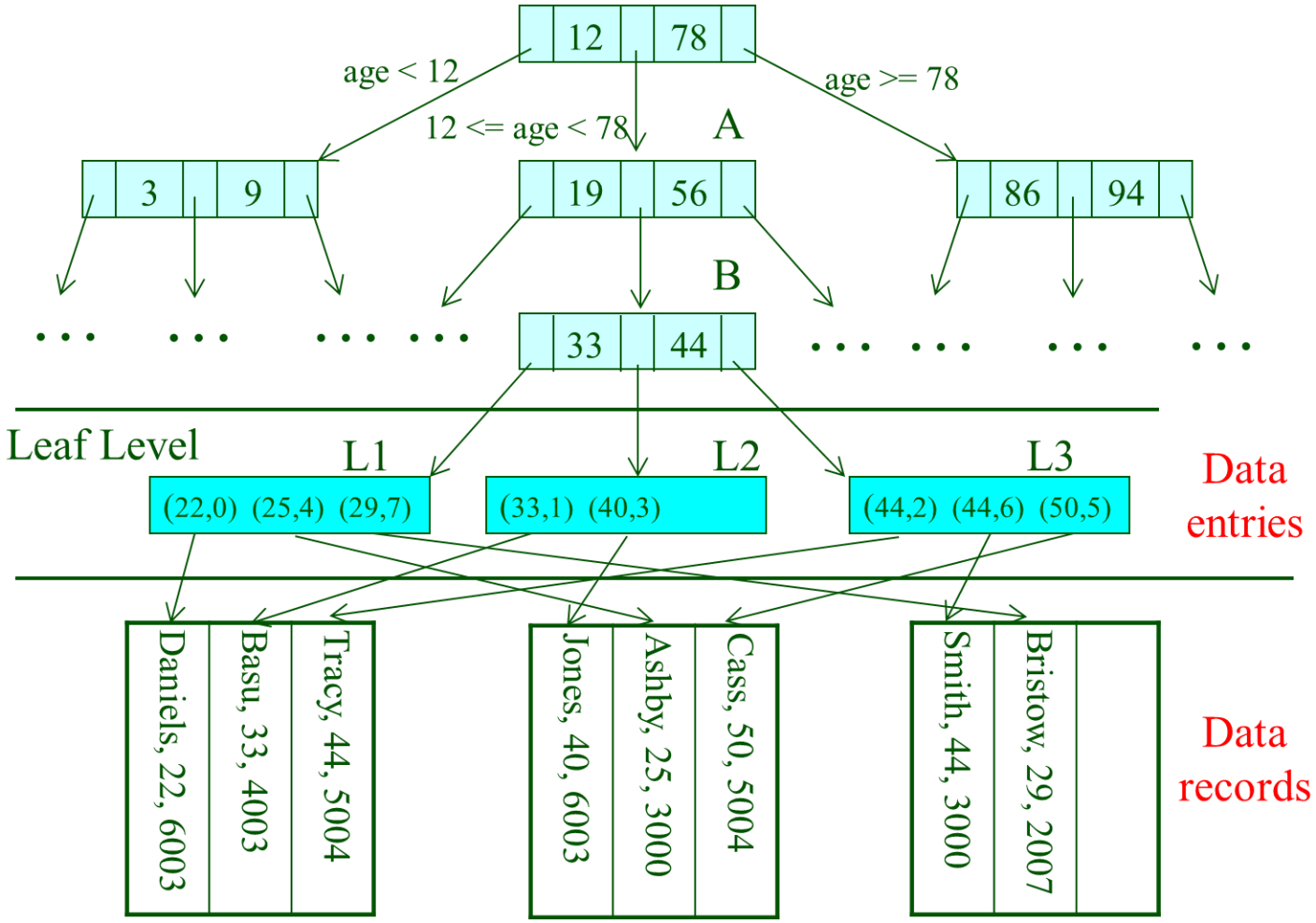
How are data entries organized?

- Hash-based Index



How are data entries organized?

- Tree-structured Index



Solution to Question 1

average seek time	20 ms
track-to-track seek time	4 ms
rotational delay	13.6 ms
max transfer rate	27.2 ms/track
bytes/sector	1024
sectors/track	50
tracks/cylinder	5
tracks/surface	1,000

the file contains 1000 256-byte records.

The access time for each record is

$$20 + 13.6 + \frac{27.2}{50} = 34.144 \text{ ms}$$

For all the 1000 records, we need

$$1000 \times 34.144 = 34144 \text{ ms}$$

Solution to Question 1

average seek time	20 ms
track-to-track seek time	4 ms
rotational delay	13.6 ms
max transfer rate	27.2 ms/track
bytes/sector	1024
sectors/track	50
tracks/cylinder	5
tracks/surface	1,000

the file contains 1000 256-byte records.

Each track contains $1024 \times 50 = 51200$ bytes.

1000 256-byte records require $\frac{256 \times 1000}{51200} = 5$ tracks, i.e., 1 cylinder.

The total time is $20+13.6+5 \times 27.2 = 169.6$ ms

Solution to Question 2

average seek time	20 ms
track-to-track seek time	4 ms
rotational delay	13.6 ms
max transfer rate	27.2 ms/track
bytes/sector	1024
sectors/track	50
tracks/cylinder	10
tracks/surface	1,000

the file contains 100,000 256-byte records.

The access time for each record is

$$20 + 13.6 + \frac{27.2}{50} = 34.144 \text{ ms}$$

For all the 100,000 records, we need

$$100{,}000 \times 34.144 = 3{,}414{,}400 \text{ ms}$$

Solution to Question 2

average seek time	20 ms
track-to-track seek time	4 ms
rotational delay	13.6 ms
max transfer rate	27.2 ms/track
bytes/sector	1024
sectors/track	50
tracks/cylinder	10
tracks/surface	1,000

the file contains 100,000 256-byte records.

Each track contains $1024 \times 50 = 51200$ bytes.
100,000 256-byte records require $\frac{256 \times 100{,}000}{51200} = 500$ tracks, i.e., 50 cylinders.

The time for the 1st cylinder is $20+13.6+27.2 \times 10=305.6$ ms.
The time for the following 49 cylinders is $49 \times (4+13.6+27.2 \times 10)=14{,}190.4$ ms.

Solution to Question 3

- A record takes: 20+4+8+96=128 bytes

  #Pages for Employee: 128×3200/4096=100 pages

  The I/O cost is then 100 I/Os.

- "Bob" may appear at the first record or the last record. On average, the I/O cost is (1+100)/2=50.5 I/Os.