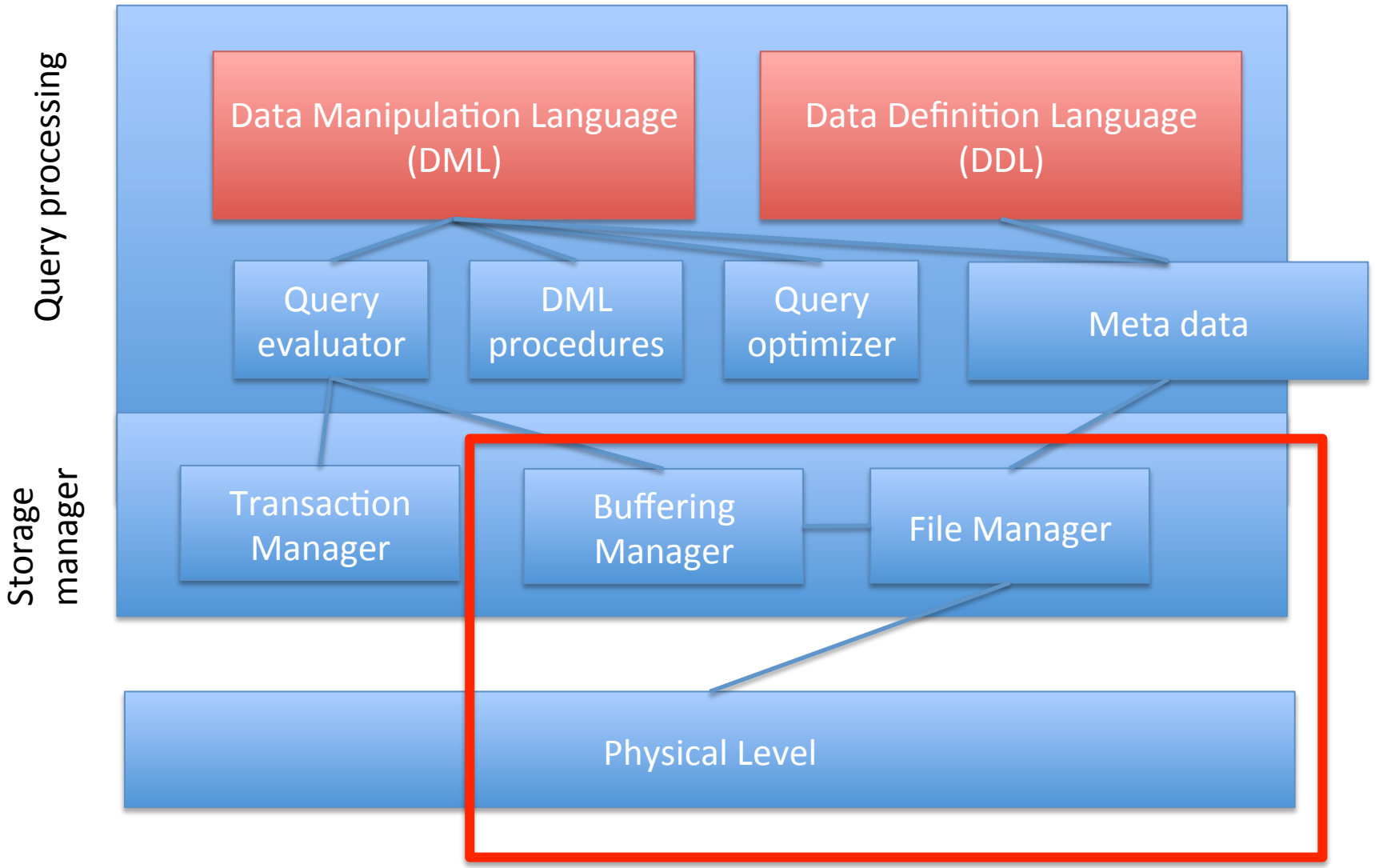
Tree Shaped Indexes

Carsten Schürmann

Database System Architecture



Physical Level

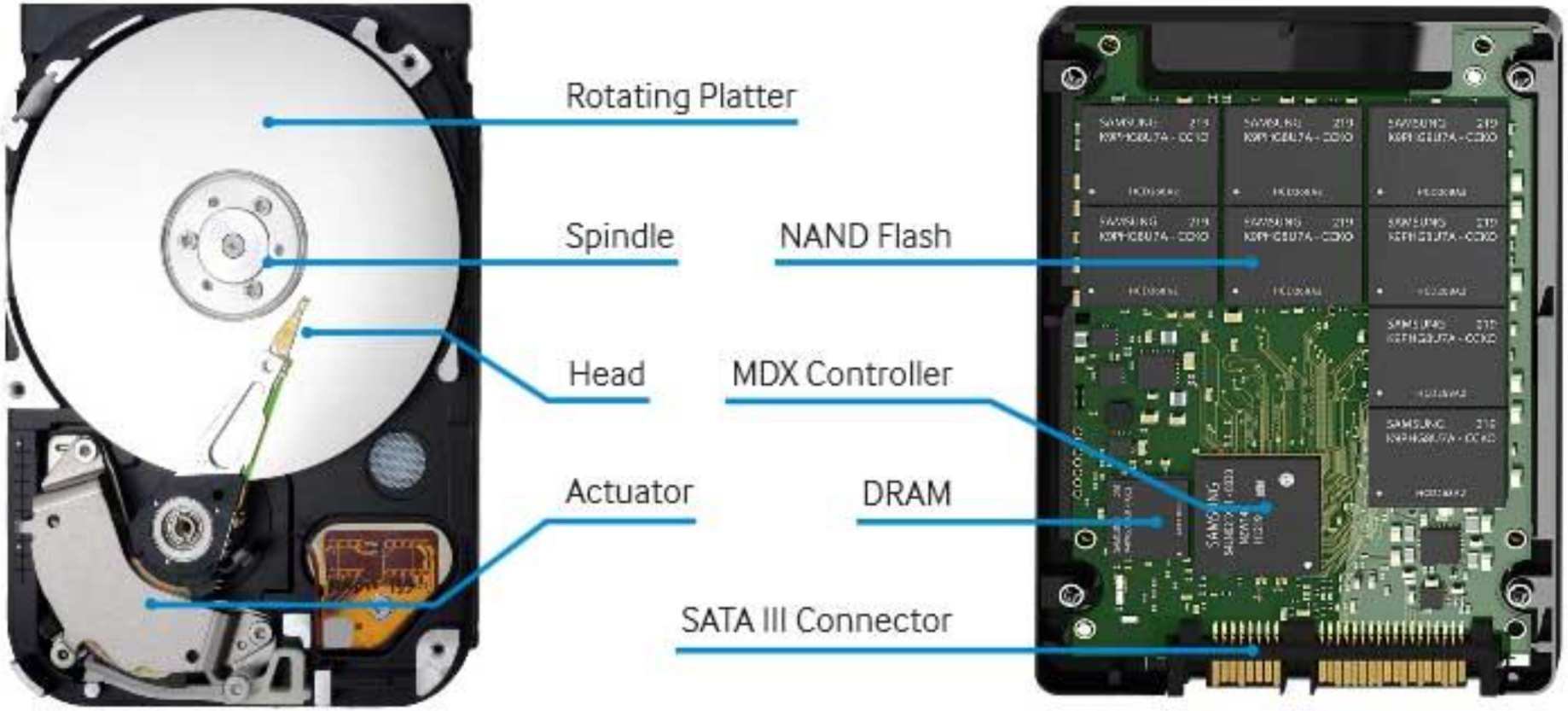
Ephemeral Storage

- RAM
- Registers on a CPU

Persistent Storage

- Magnetism
  - Disk drives
  - Hard drives
- Trapped electrons
  - SSD

HardDrive vs. SSD



Disk and Files

DBMS store information on disks!

- Disks are relatively slow
- READ: Disk -> main memory
- WRITE: Main memory -> disk

READ/WRITE operation are expensive!

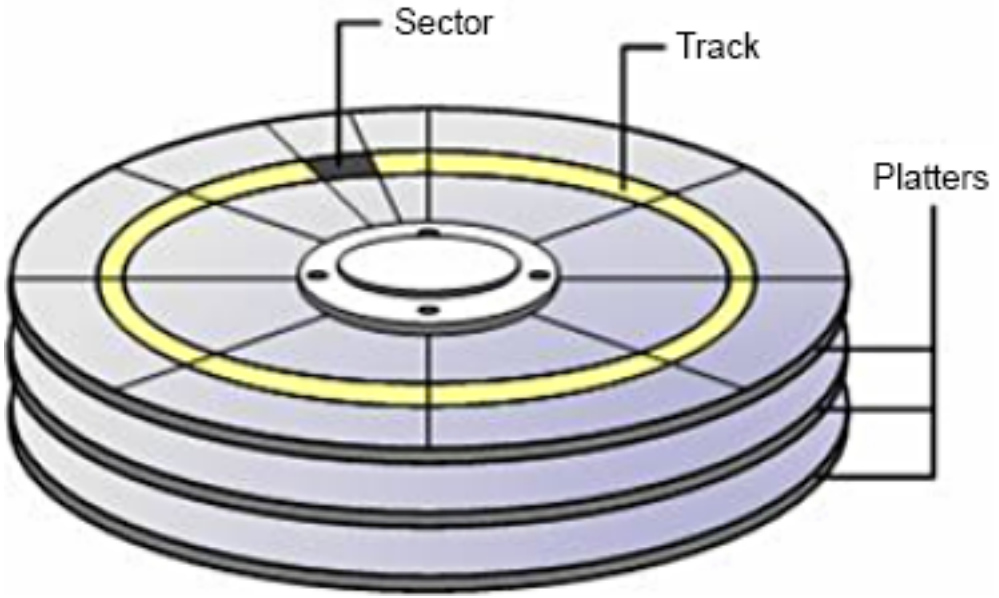
Hard Disks

Storage device of choice.

- *random access* vs. *sequential* access
- Data is stored and retrieved in units called *disk blocks* or *pages*.
- Time to retrieve a disk page depends upon location on disk.
- Relative placement of pages on disk is affects READ/WRITE speed

Anatomy of a Hard Disk

- Sector
- Track
- Cylinder
- Platter

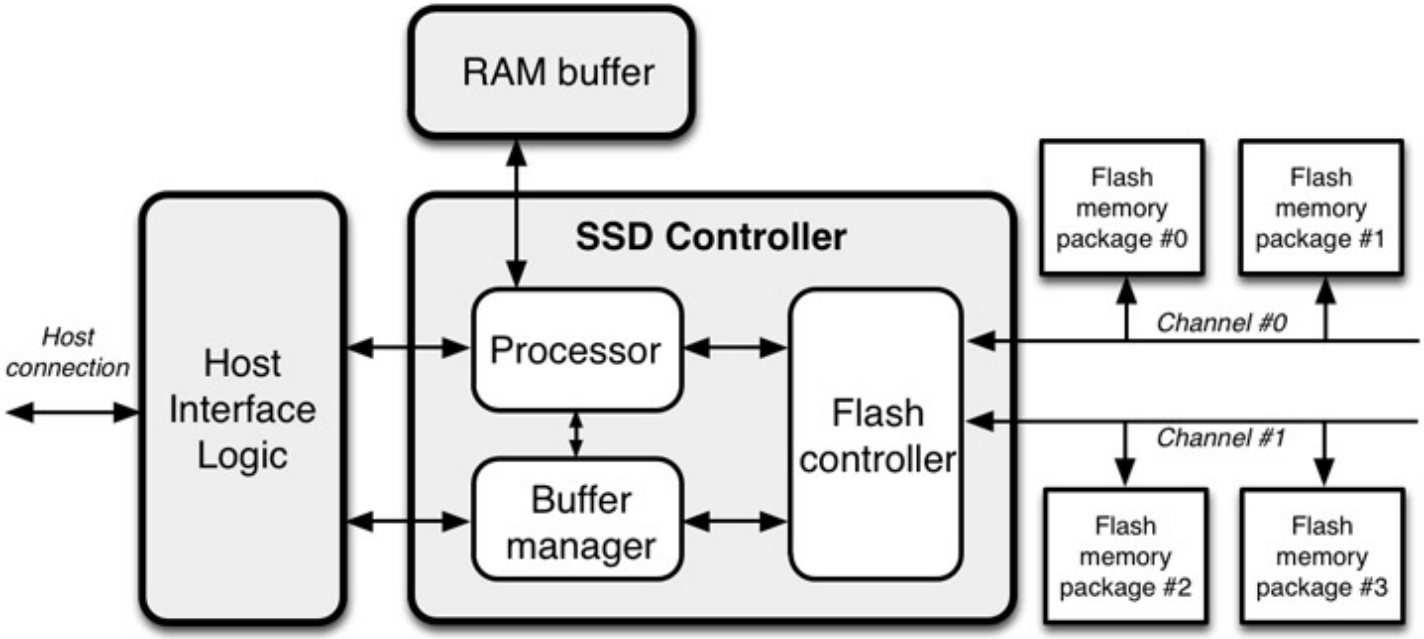


Sector size: Fixed

Block size: Multiple of sector size

Anatomy of a SSD Drive

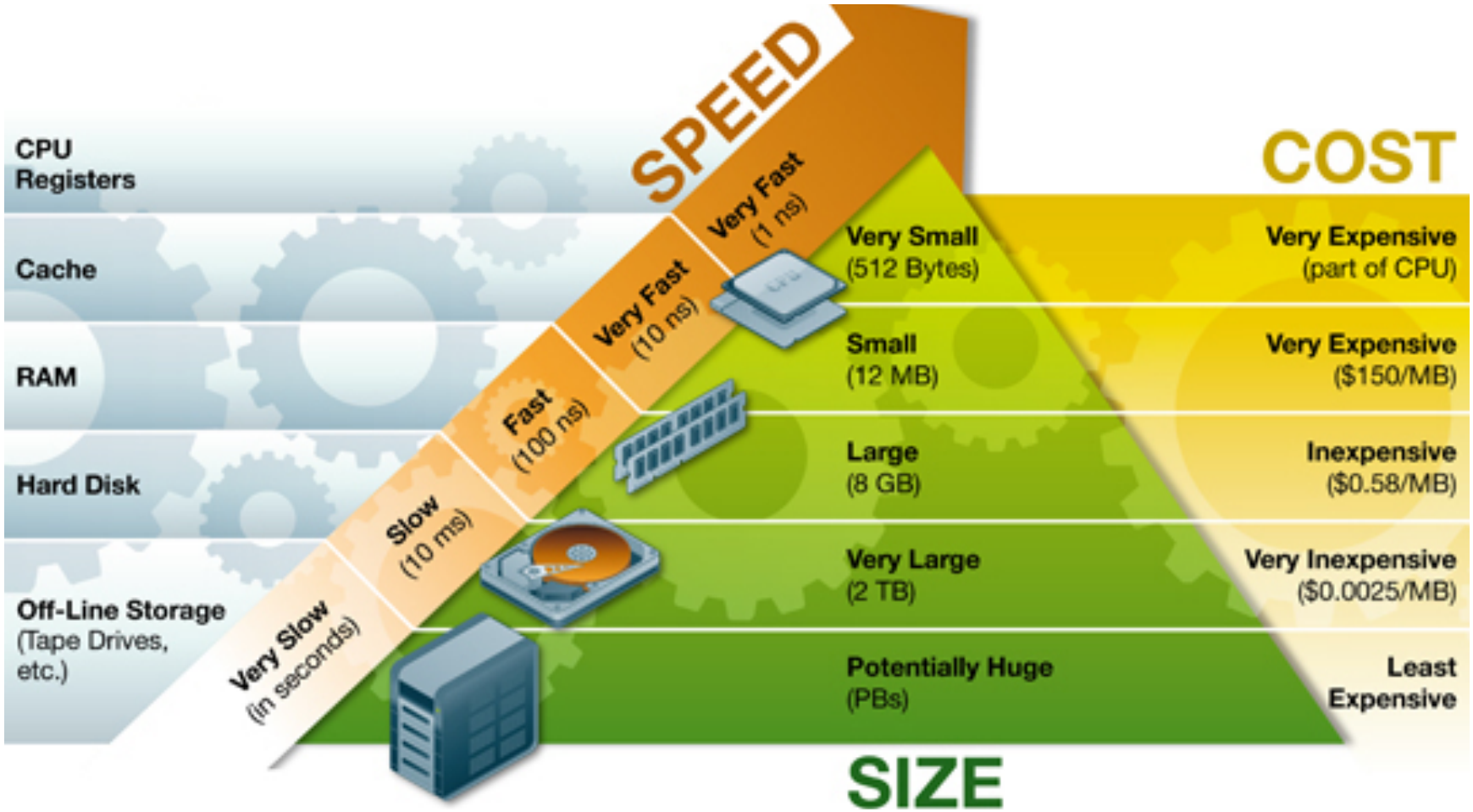
- Sector
- Track
- Cylinder
- Platter



Sector size: Fixed

Block size: Multiple of sector size

Storage Hierarchy



Disk Page = Disk Block

Time to access (read/write) a disk block:

Seek time:

Moving arms to position disk head on track

Rotational delay:

Waiting for block to rotate under head

Transfer time:

Actually moving data to/from disk surface

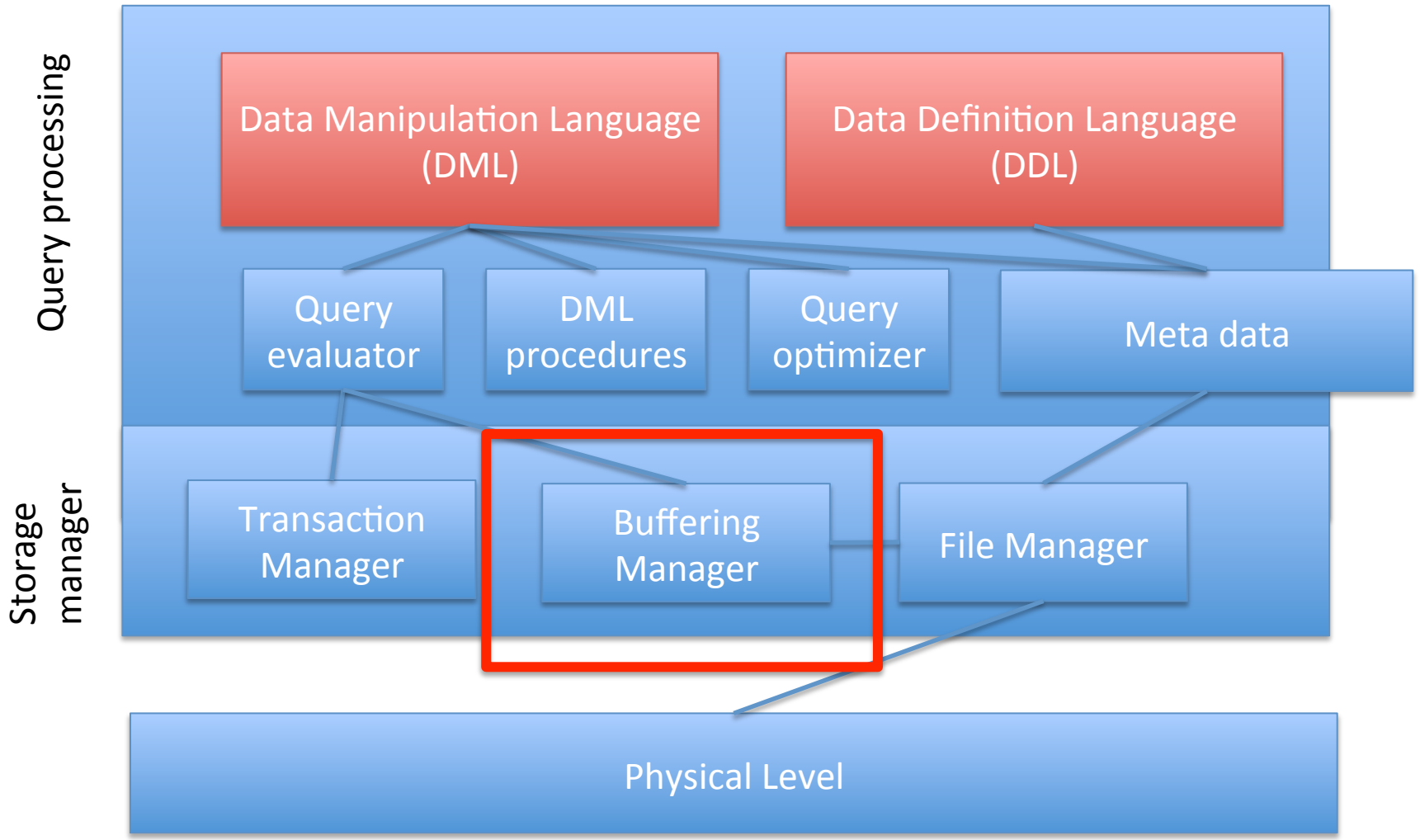
Speed Comparison

	Hard Drive	SSD
Seek time	about 1 to 20ms	0.05-0.5 ms
Rotational delay	0 to 10ms	N/A
Transfer time	< 1msec per 4KB page	200MB/s

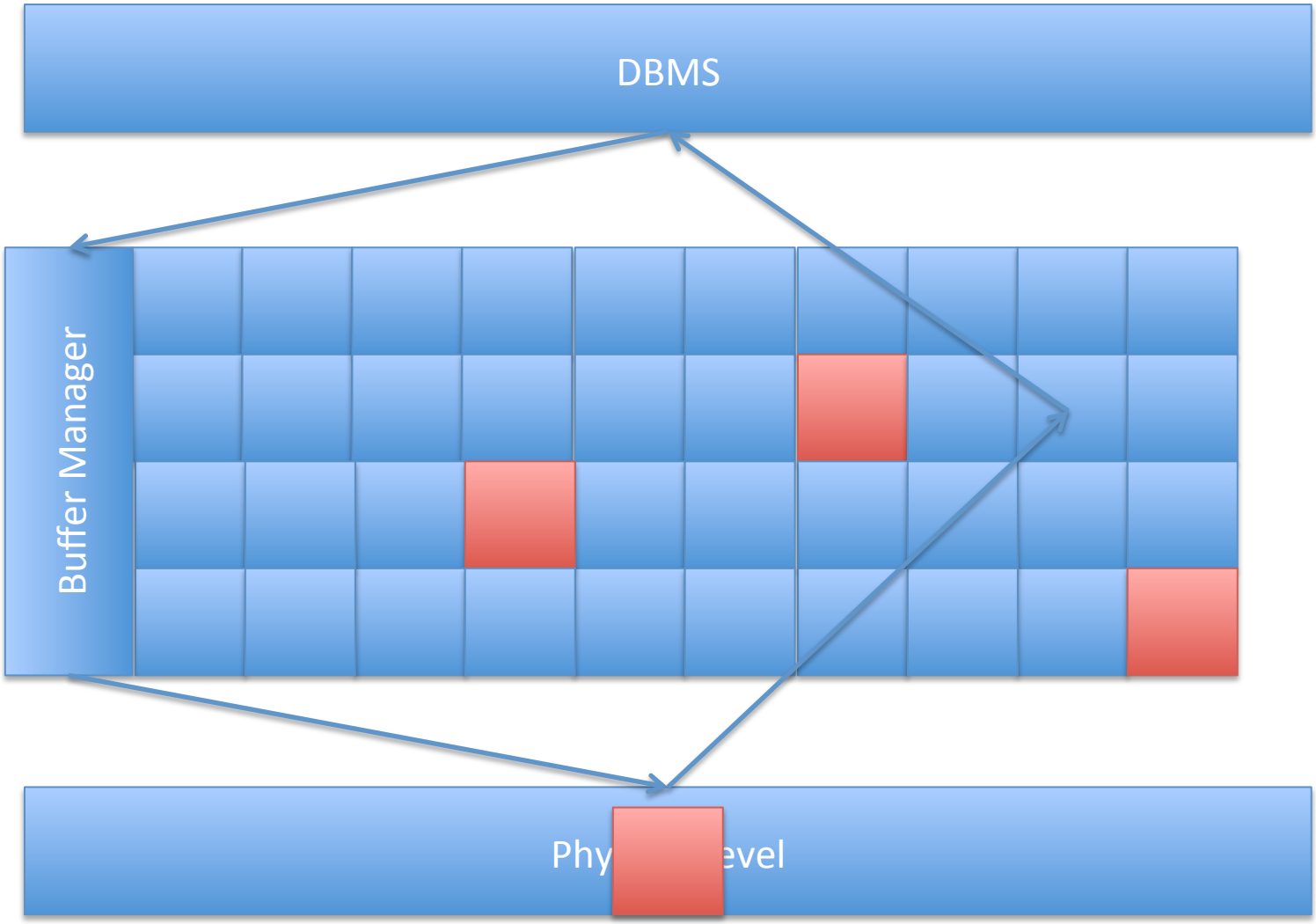
Arranging Pages on a Disk

- *Next* block concept:
  - blocks on same track, followed by
  - blocks on same cylinder, followed by
  - blocks on adjacent cylinder
- Accessing 'next' block is cheap
- An important optimization: pre-fetching
- More information (See R&G page 323)

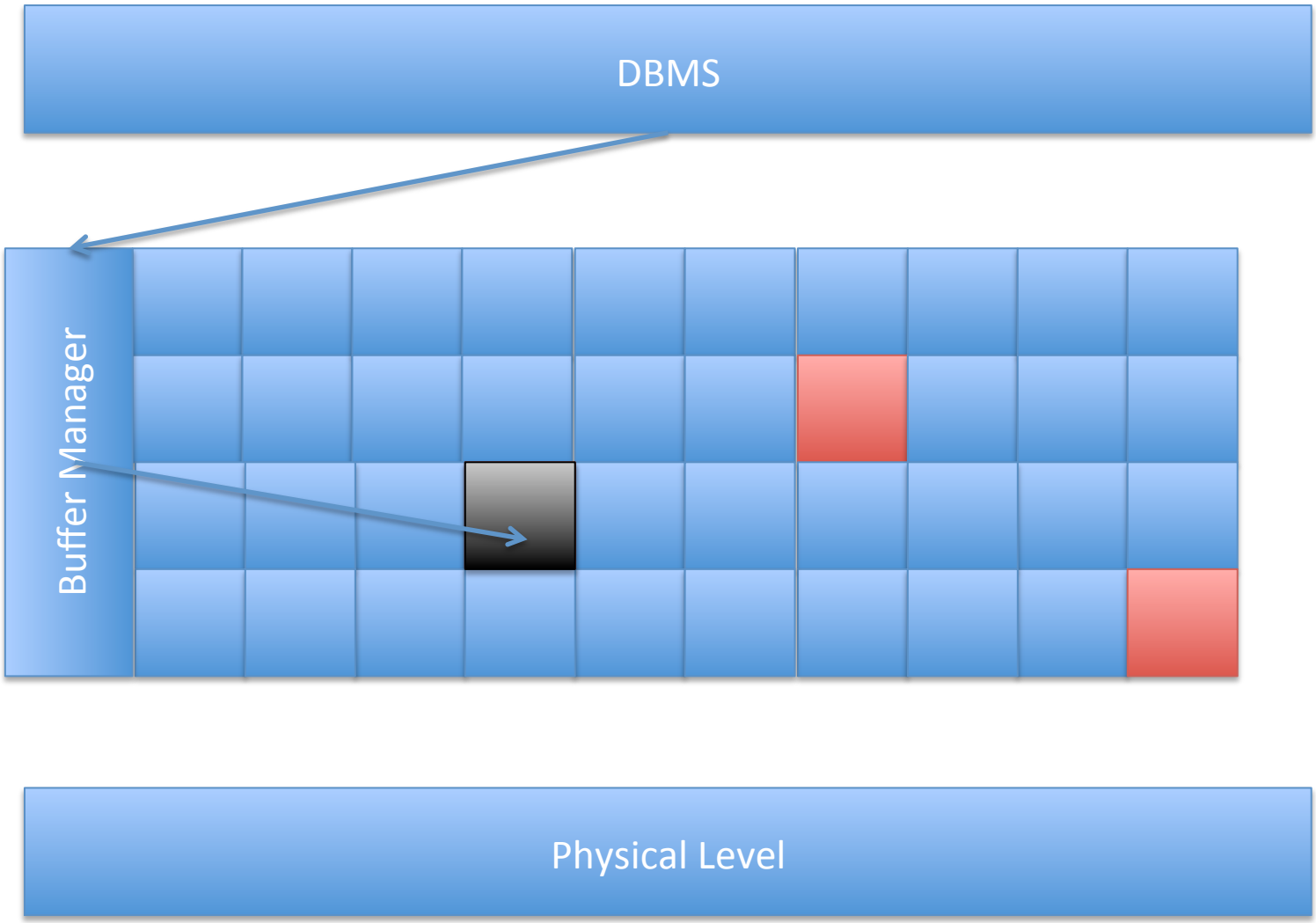
Database System Architecture



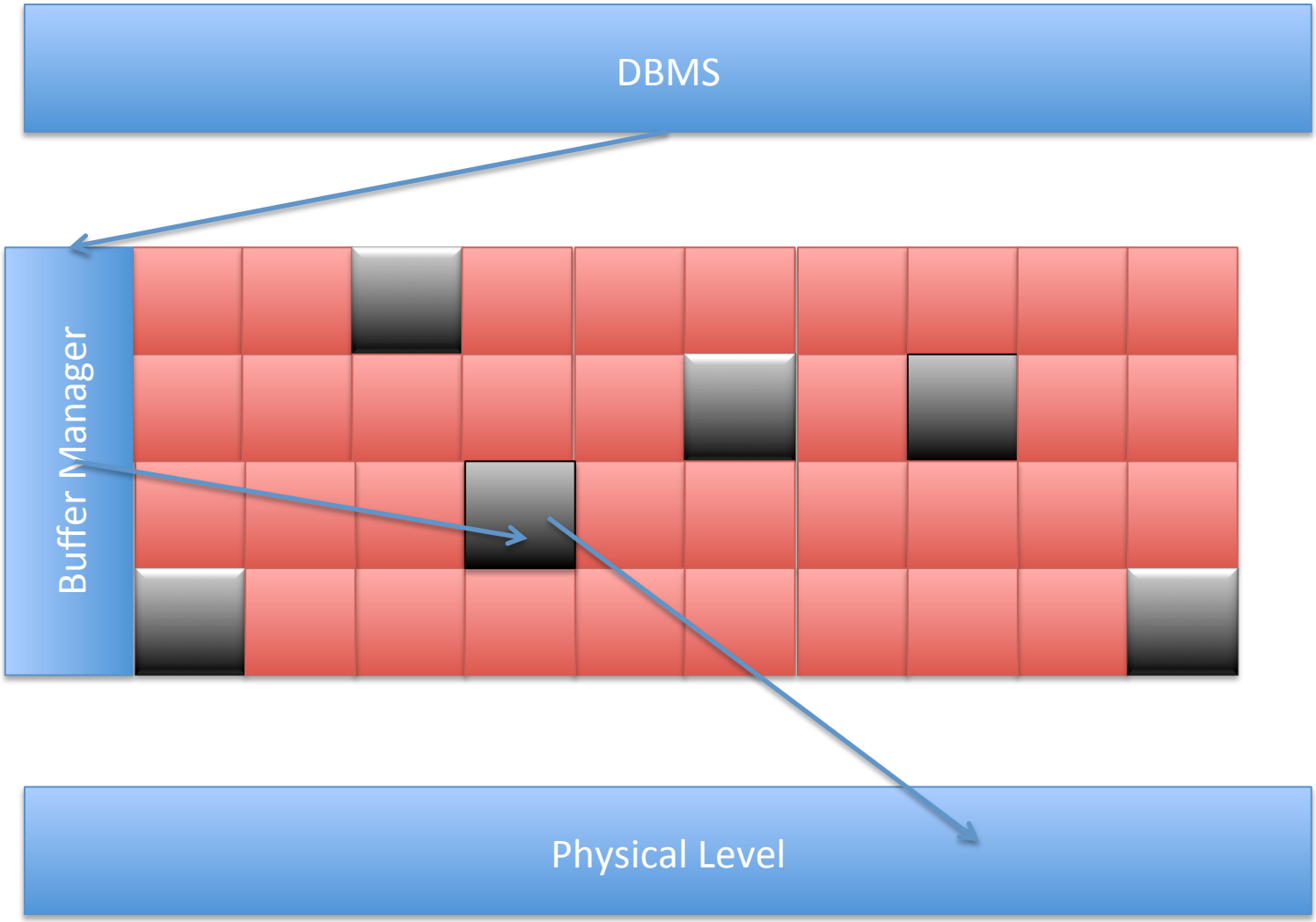
Buffer



Storing Data/Dirty Page



Page Fault



Buffer manager

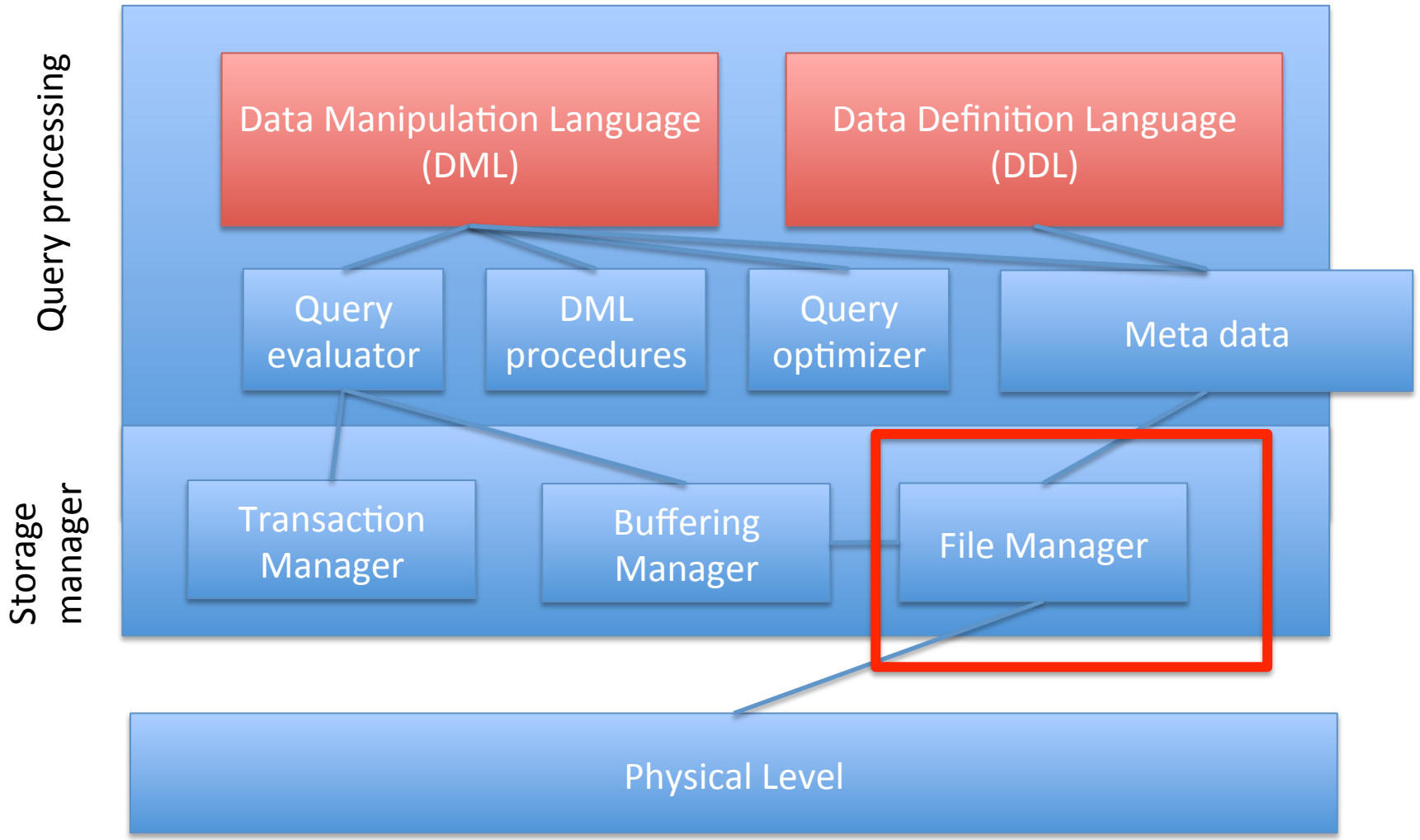
- DBMS require all pages to be in RAM
- Buffer manager encapsulate access disk
- Implements a strategy to handle page faults

Strategy

- Keep page counter
  - Counter = 0 and page not dirty: flag as replaceable
  - Counter = 0 and page dirty: write to disk
- Which page to pick?
  - Least Recently Used (LRU) page
  - But there are others

[See R&G]

Database System Architecture



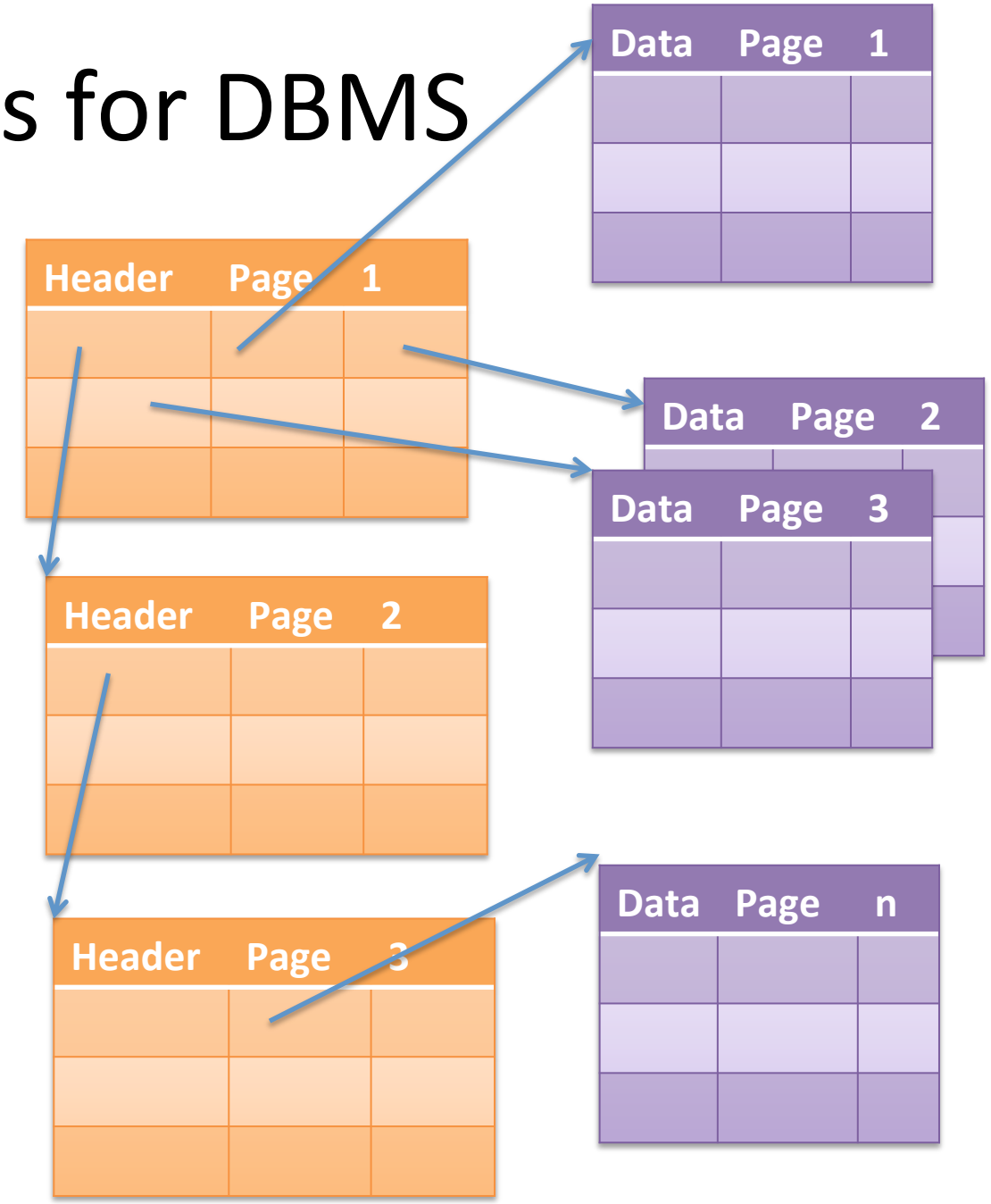
Files for DBMS

Pages

- Data pages
- Header pages

Information

- Empty space
- Alternative Header page structures

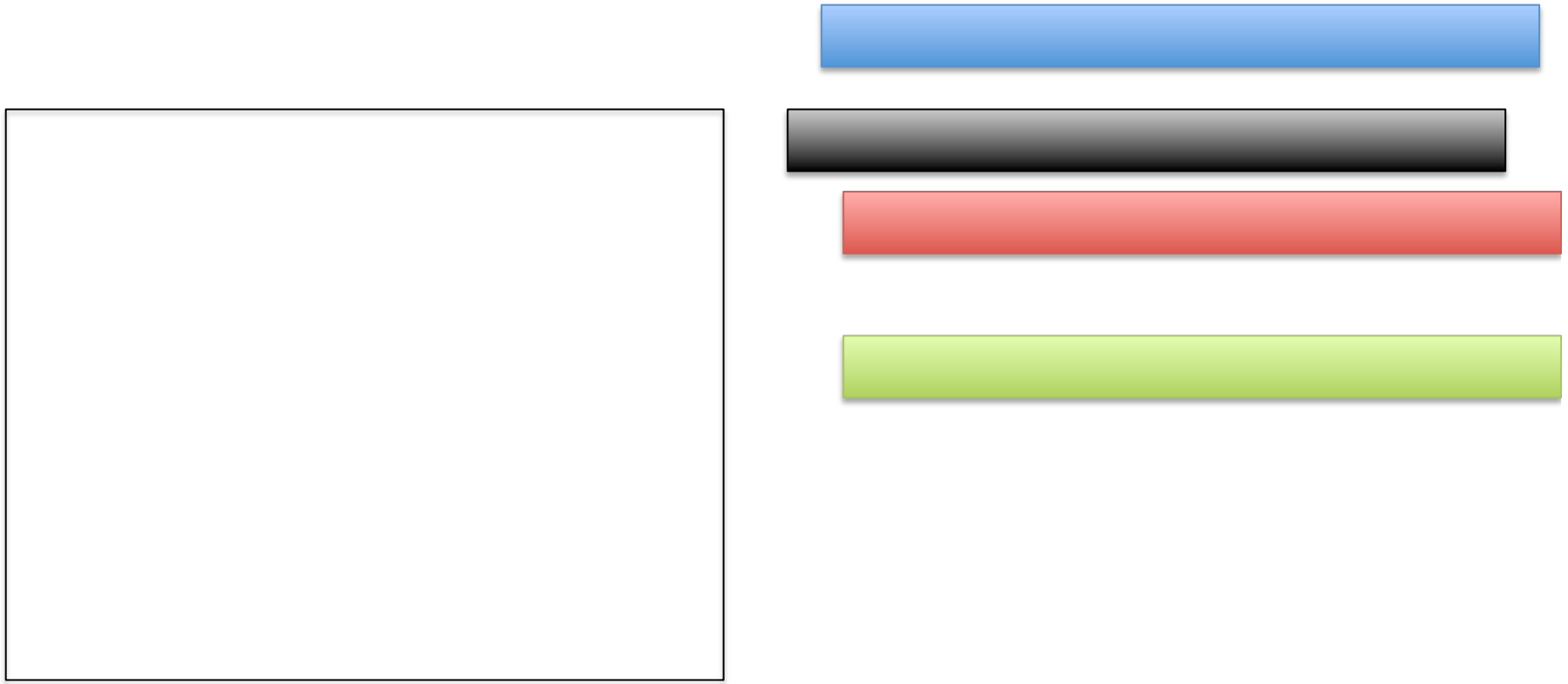


Page Formats

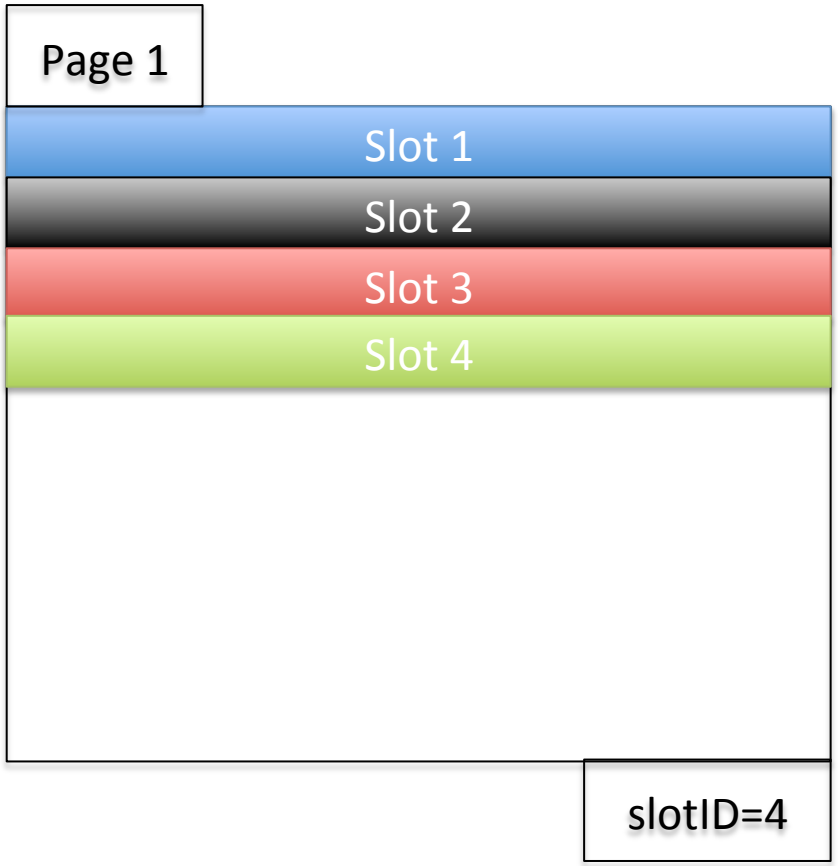
How to fit records into a (4kB page!)?
Fixed Length Records and Variable Length Records

- How to retrieve a record?
- How to insert a record?
- How to delete a record?

Fixed Length Records



Retrieval



- Retrieval

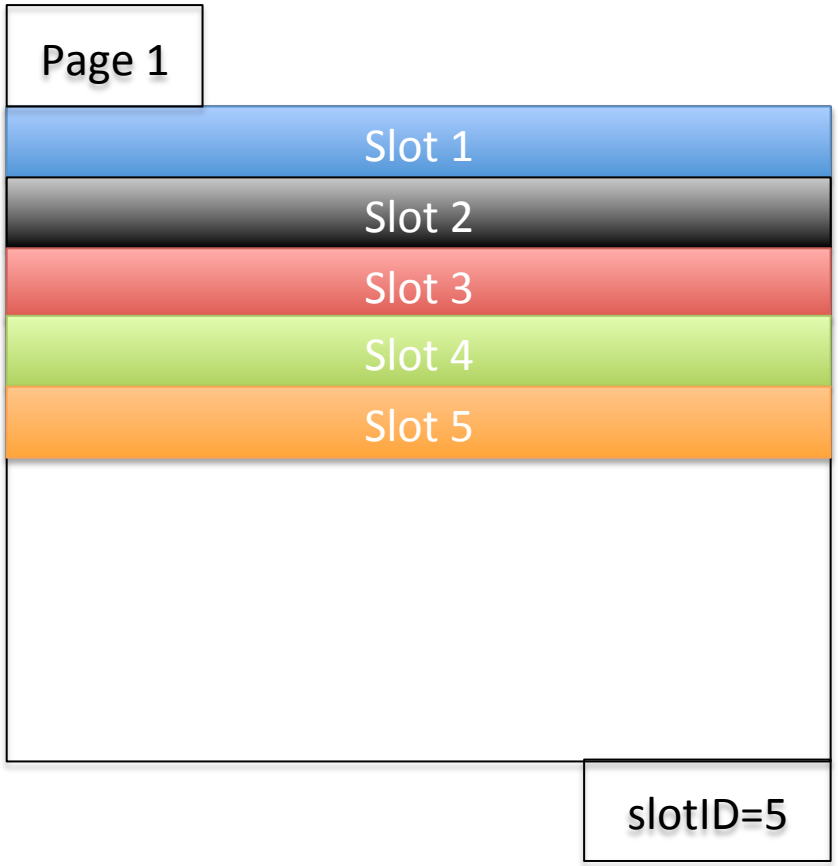
  Define RecordID

  rid = pid:slotID

- Insertion

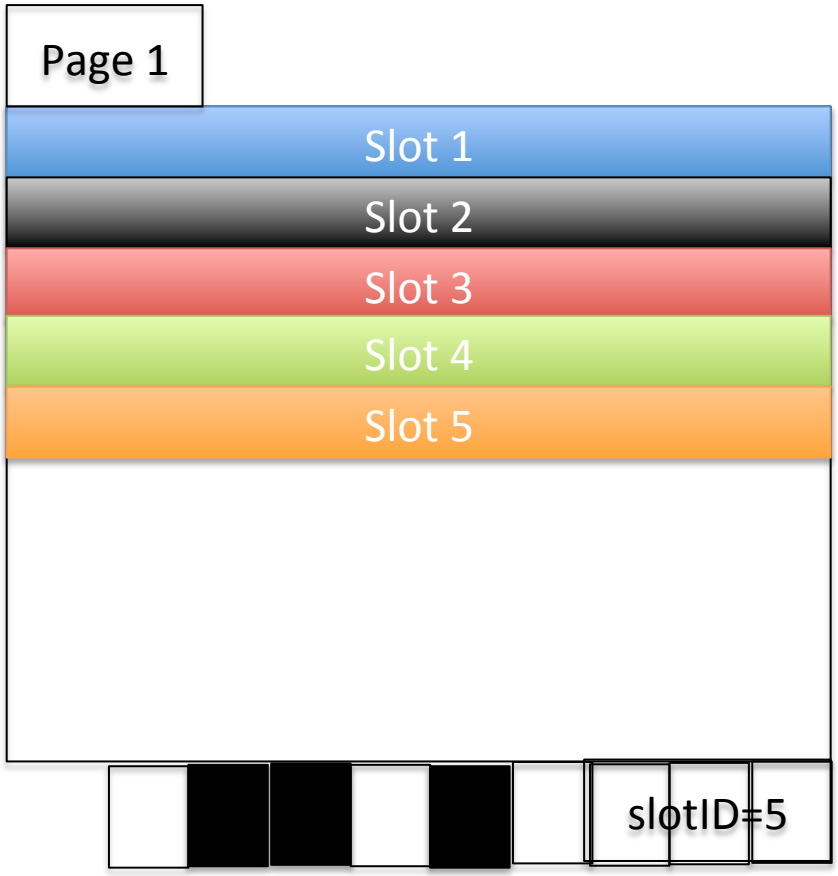
- Deletion

Insertion



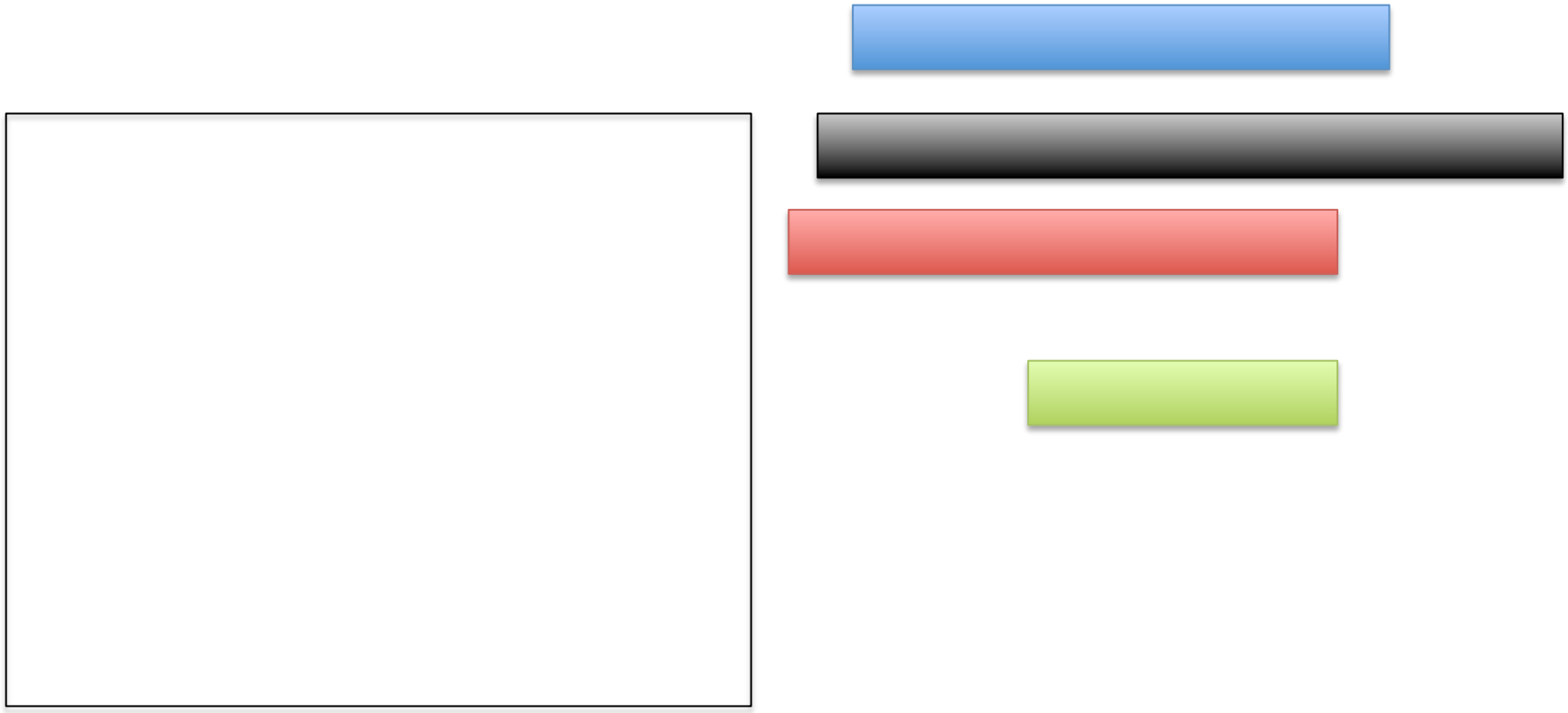
- Retrieval
  - Define RecordID
    - rid = pid:slotID
- Insertion
  - Insert and increase
- Deletion

Deletion

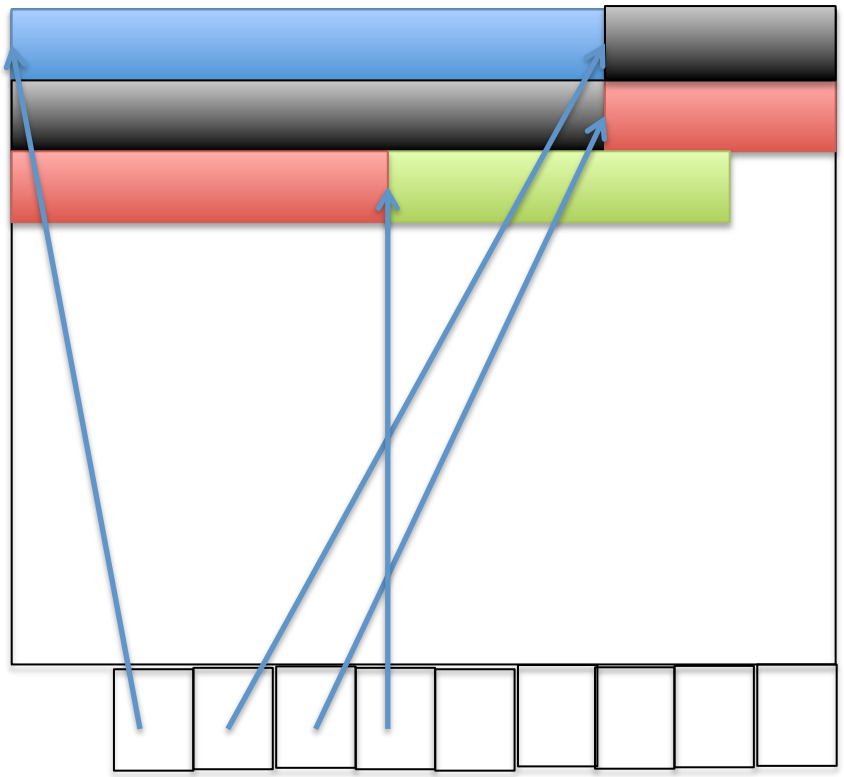


- Retrieval
  - Define RecordID
    - rid = pid:slotID
- Insertion
  - Insert and increase
- Deletion
  - Bit Indexes

Variable Length Records



Retrieval



- Retrieval

  Slot Directory

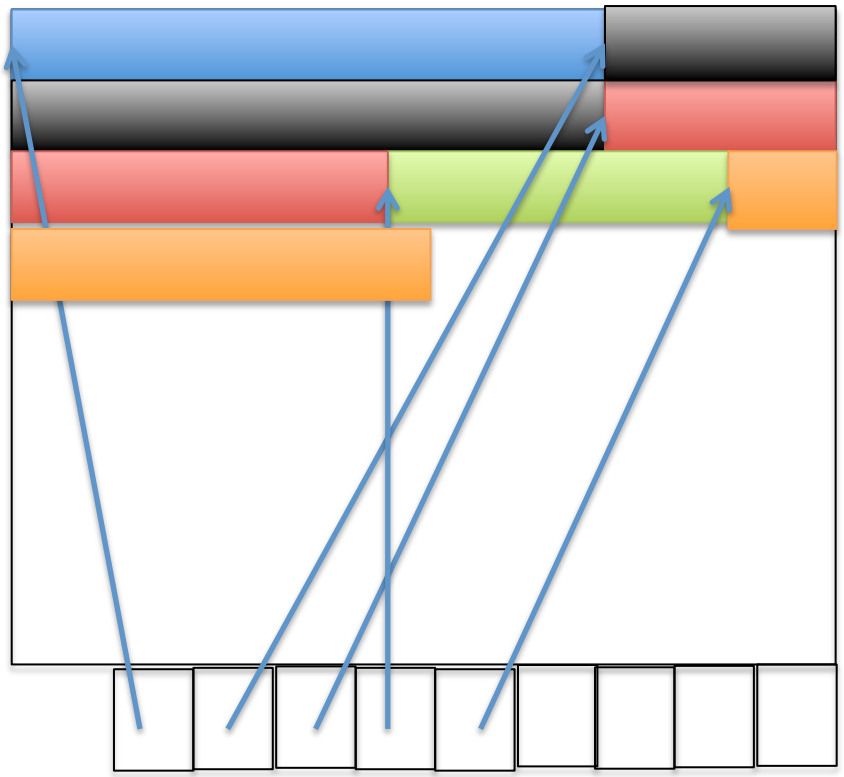
  Define RecordID

  rid = pid:slotID

- Insertion

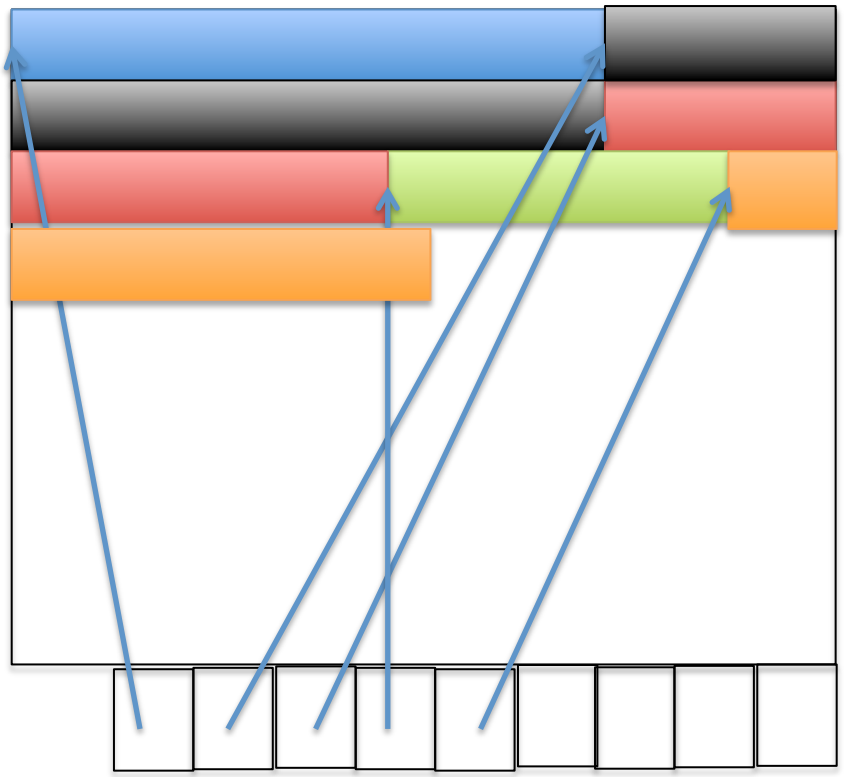
- Deletion

Insertion



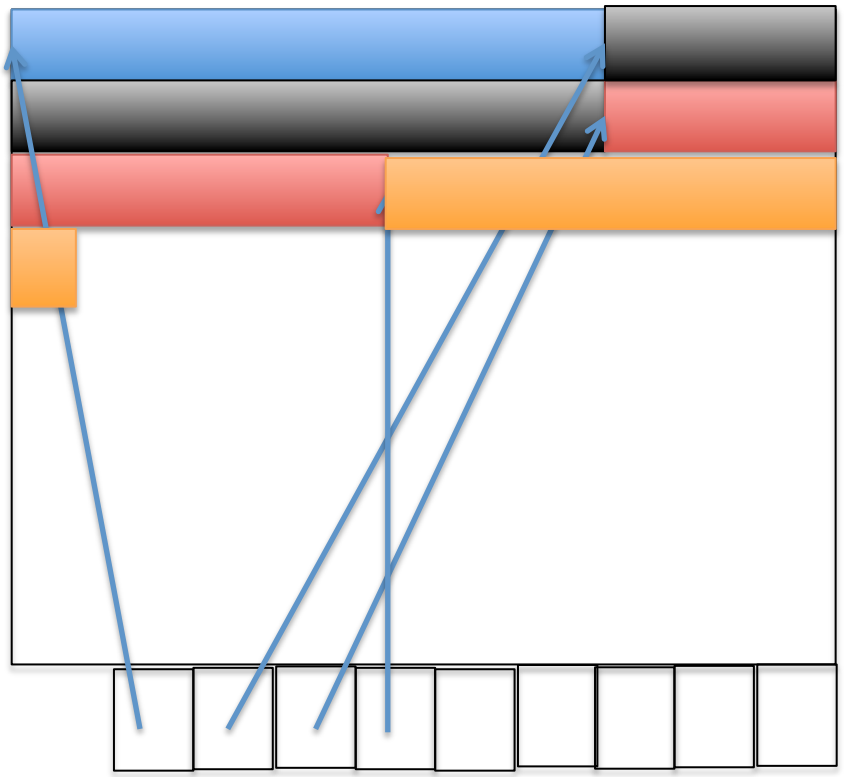
- Retrieval
  - Slot Directory
  - Define RecordID
    - rid = pid:slotID
- Insertion
  - Add new record
- Deletion

Deletion



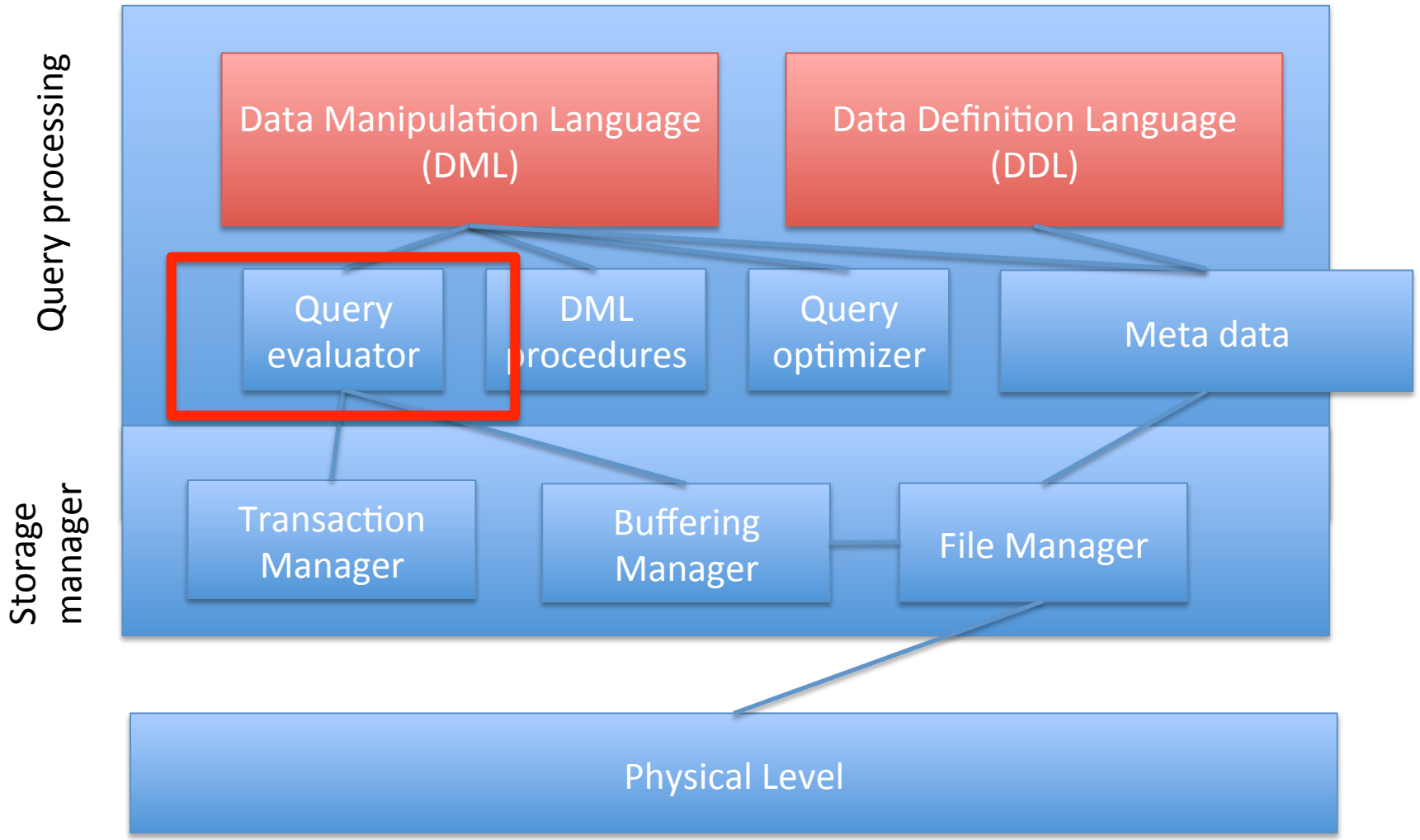
- Retrieval
  - Slot Directory
  - Define RecordID
    - rid = pid:slotID
- Insertion
  - Add new record
- Deletion

Deletion



- Retrieval
  - Slot Directory
  - Define RecordID
    - rid = pid:slotID
- Insertion
  - Add new record
- Deletion
  - Update entire page

Database System Architecture



The Need For Speed

- Full table scans

```
SELECT *
FROM R
WHERE <condition>
```

- Range queries

```
SELECT *
FROM Person
WHERE year(birthdate)
BETWEEN 1975 and 1994
```

- Point queries

```
SELECT *
FROM Person
WHERE birthyear=1975
```

Indexes

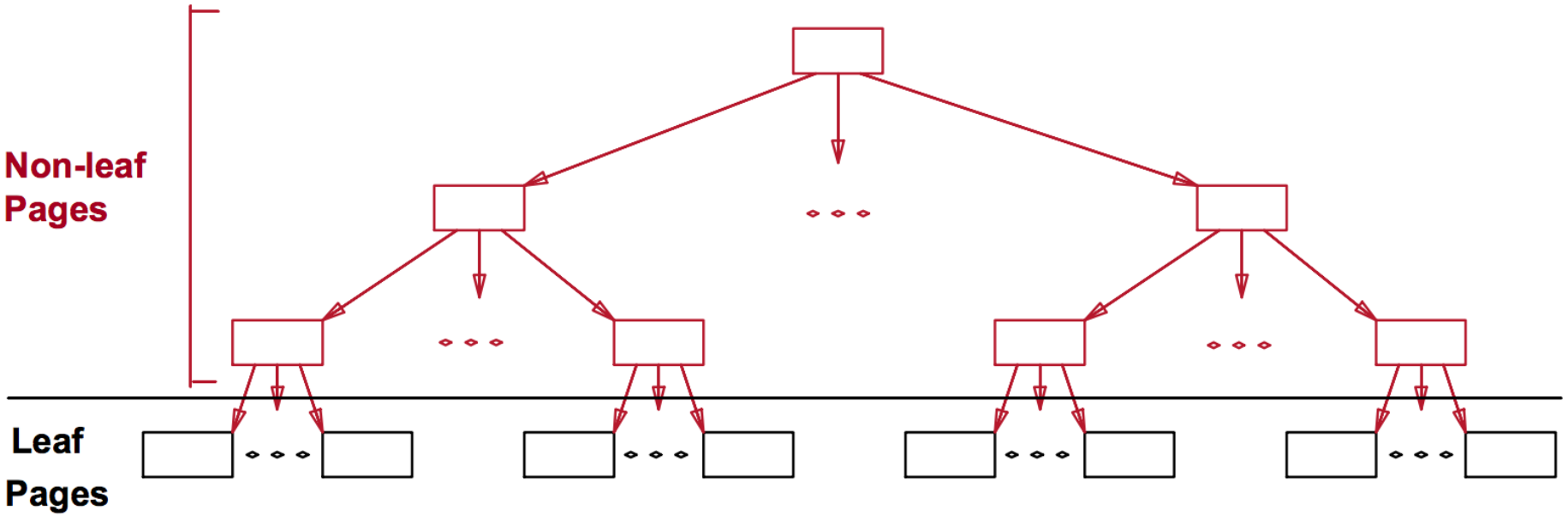
```
SELECT *
FROM Person
WHERE year(birthdate)
BETWEEN 1975 and 1994
```

Contents

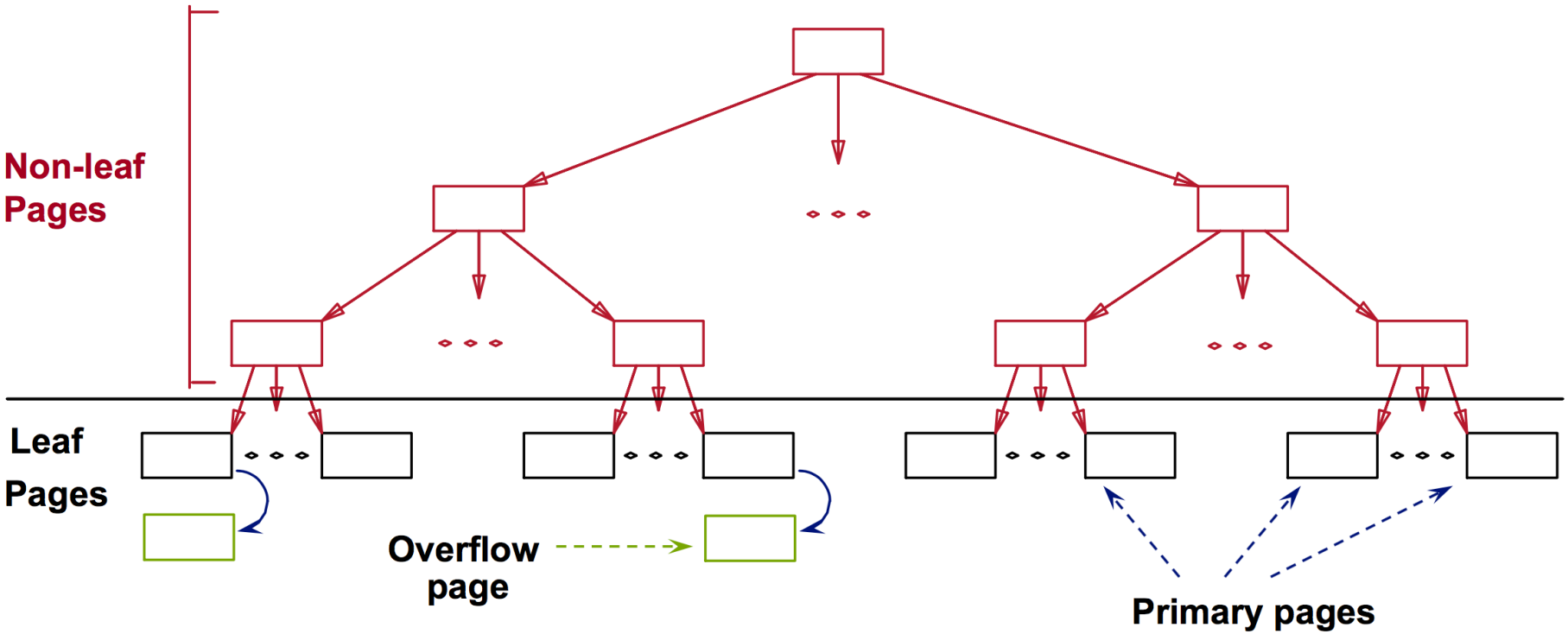
- Indexed Sequential Access Method (ISAM)
- B – trees
- B+ – trees
- MySQL index

ISAM

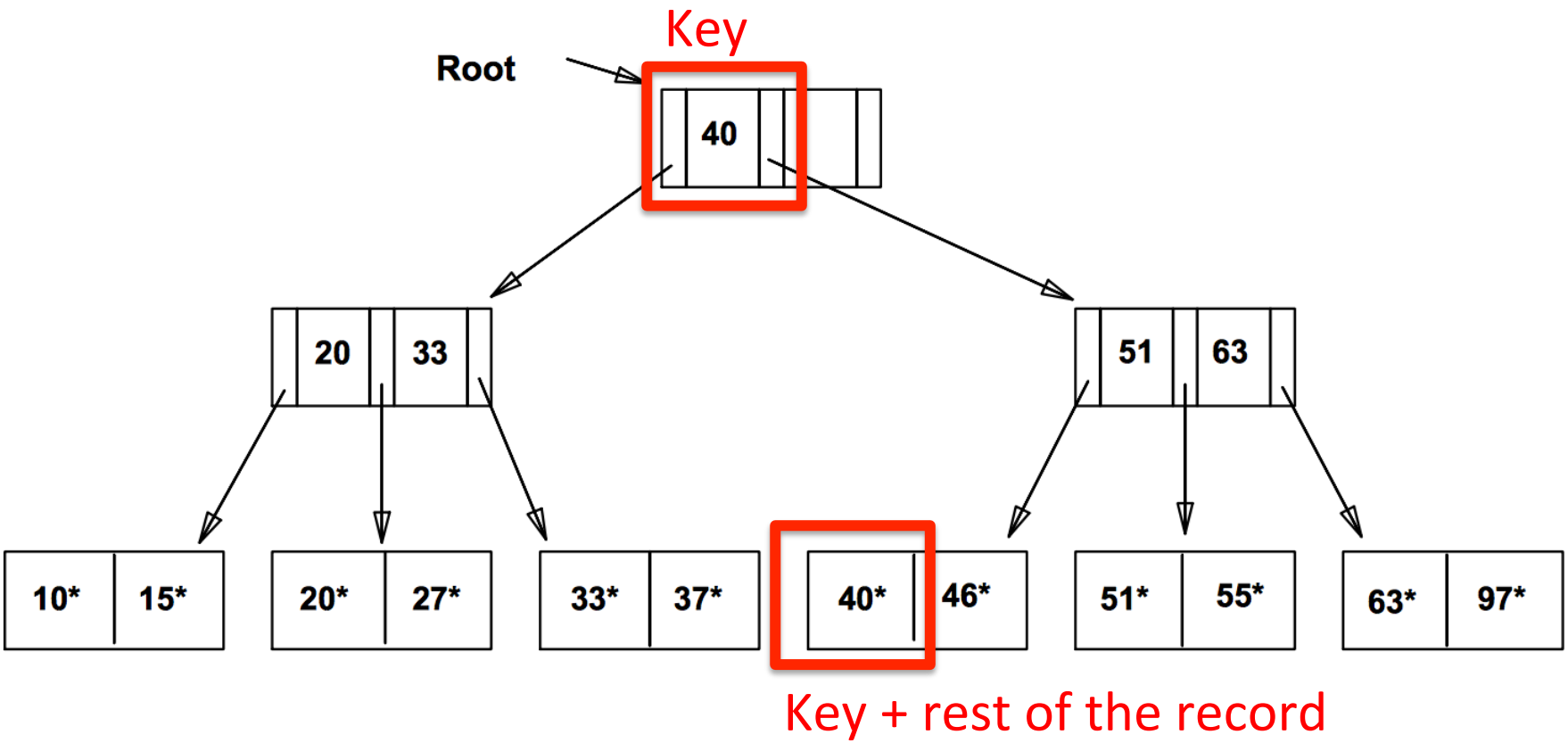
- Originally developed by IBM
- Fixed Length Records
- Sequential data + Index datastructure



Insertion and Overflow

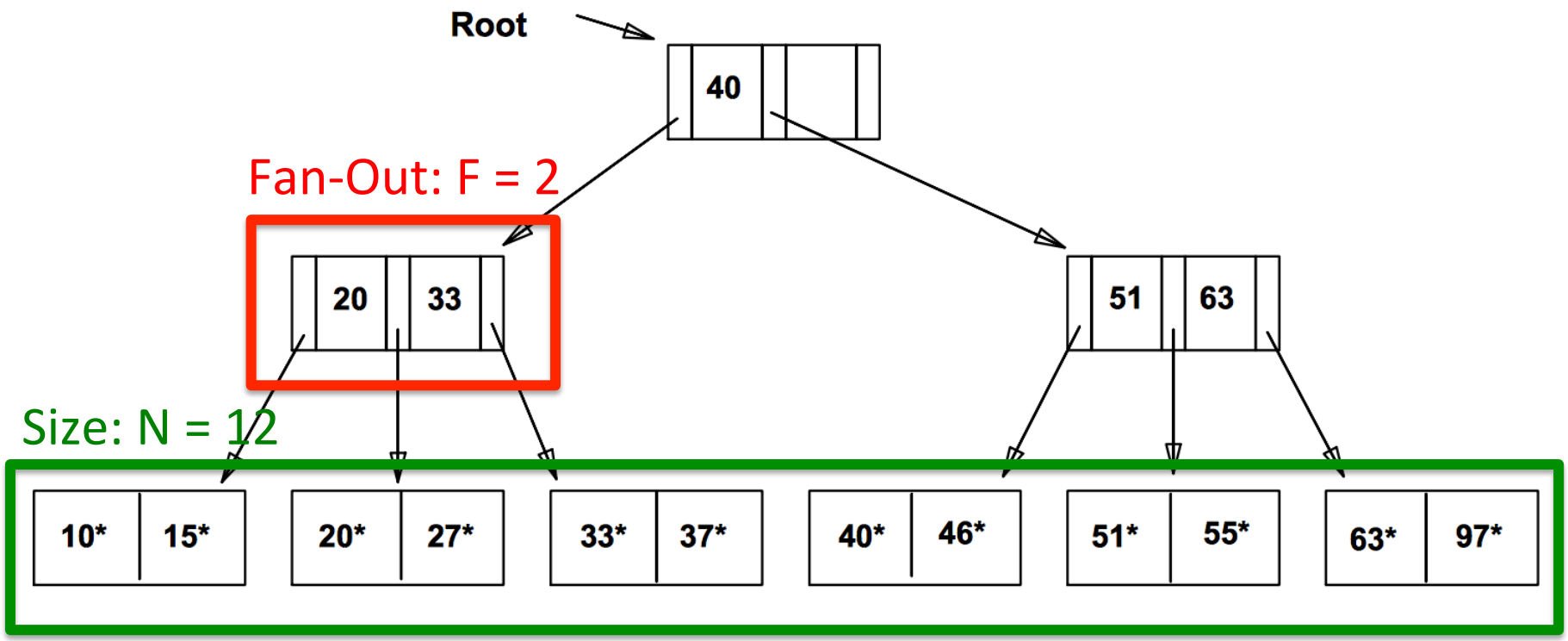


ISAM in Action

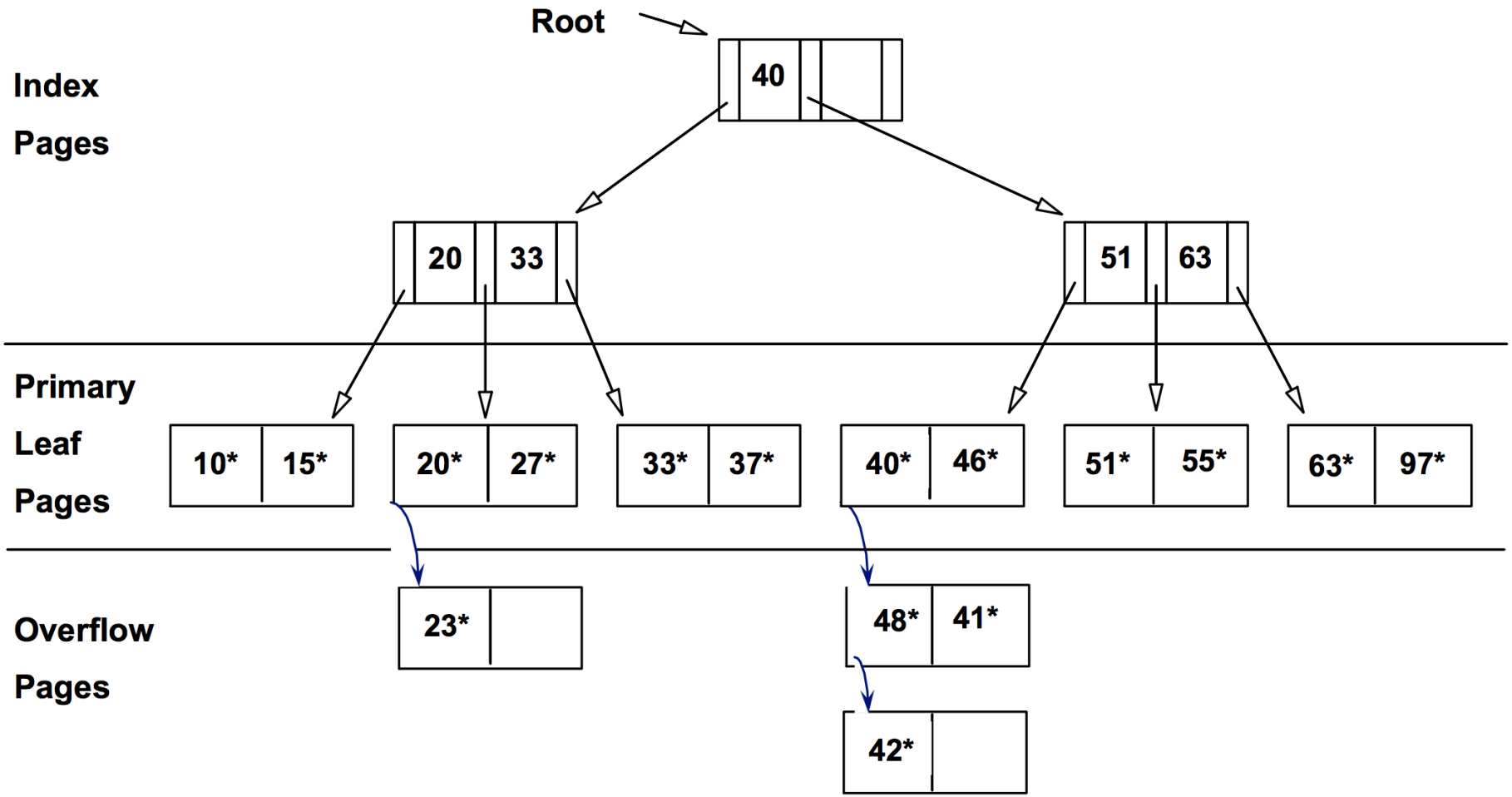


Properties

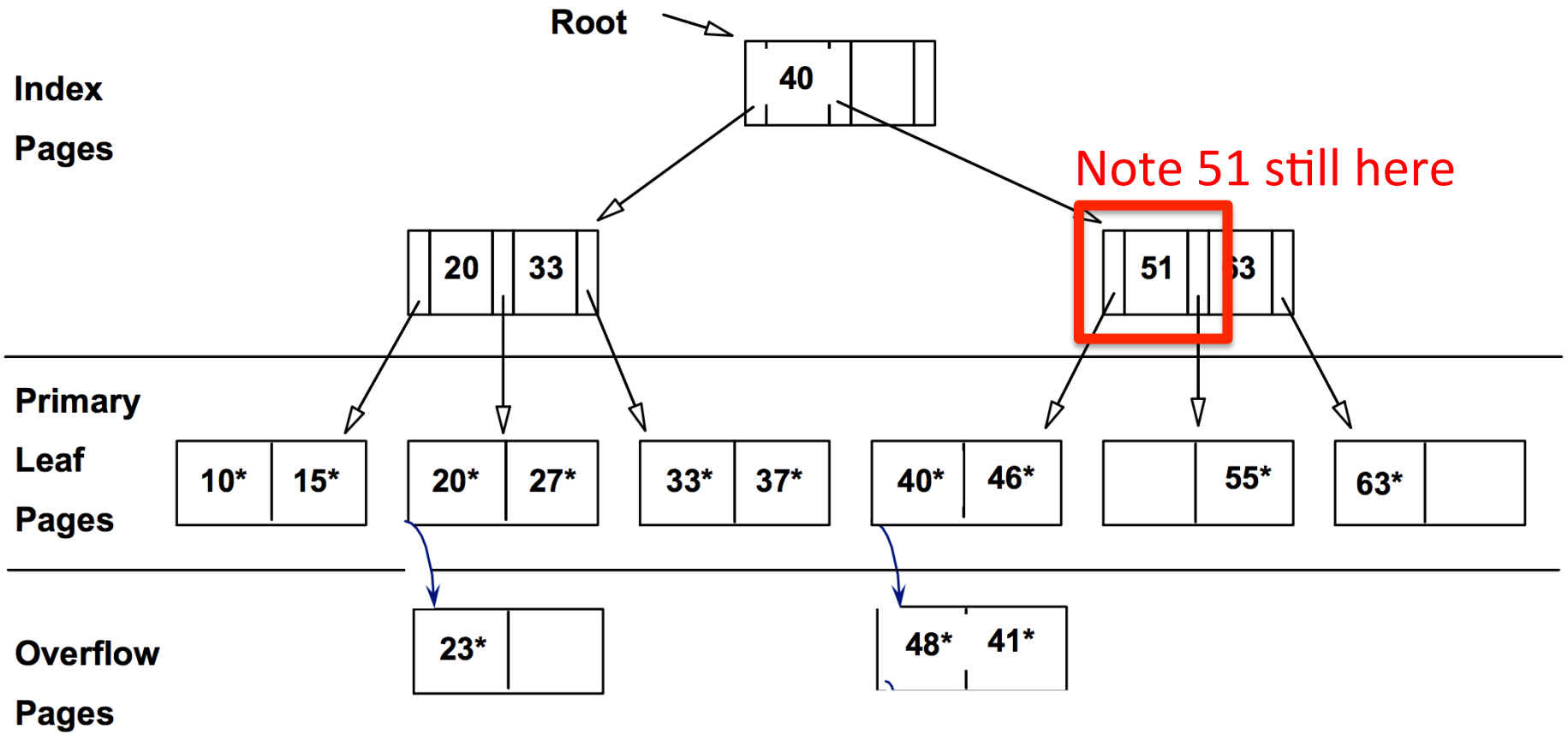
- ISAM is a static structure
- Retrieval: Cost: O ($\log_F$ N)



Insertion 23*, 48*, 41*, 42*



Deletion 42*, 51*, 97*



Contents

- Indexed Sequential Access Method (ISAM)
- B – trees
- B+ – trees
- MySQL index

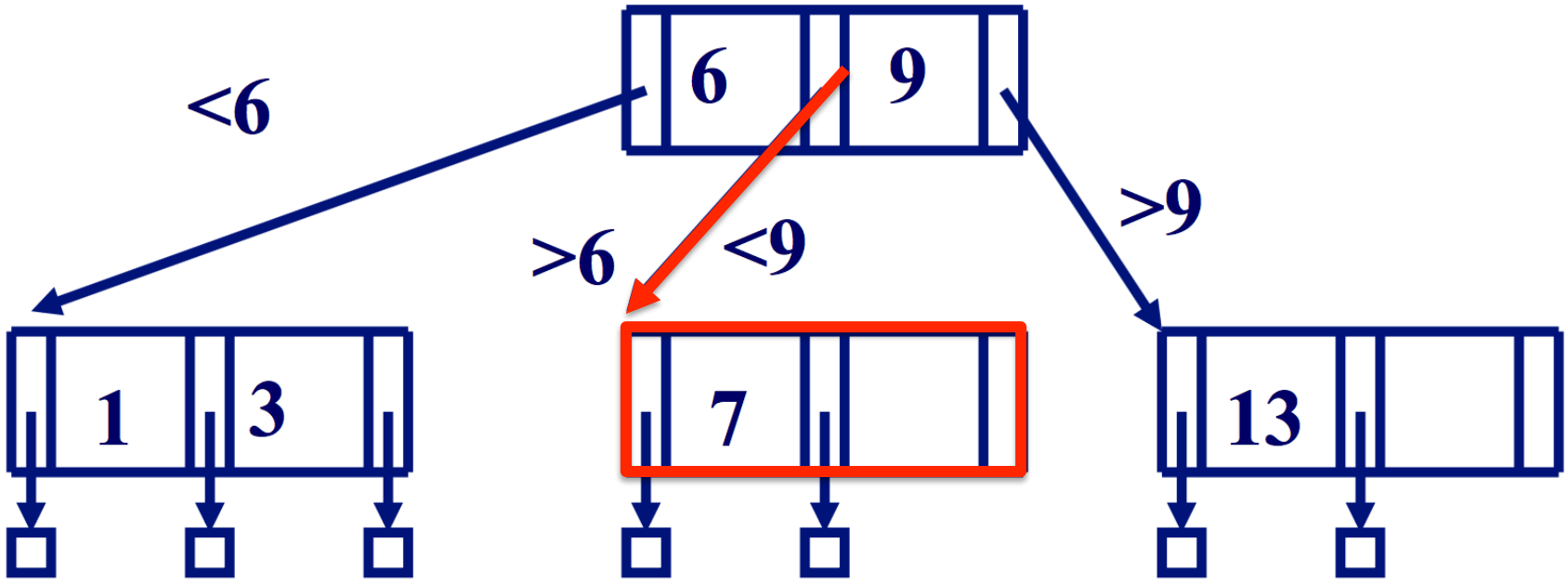
B-Trees

[Rudolf Bayer and McCreight, E. M. Organization and Maintenance of Large Ordered Indexes. Acta Informatica 1, 173-189, 1972.]

http://slady.net/java/bt/

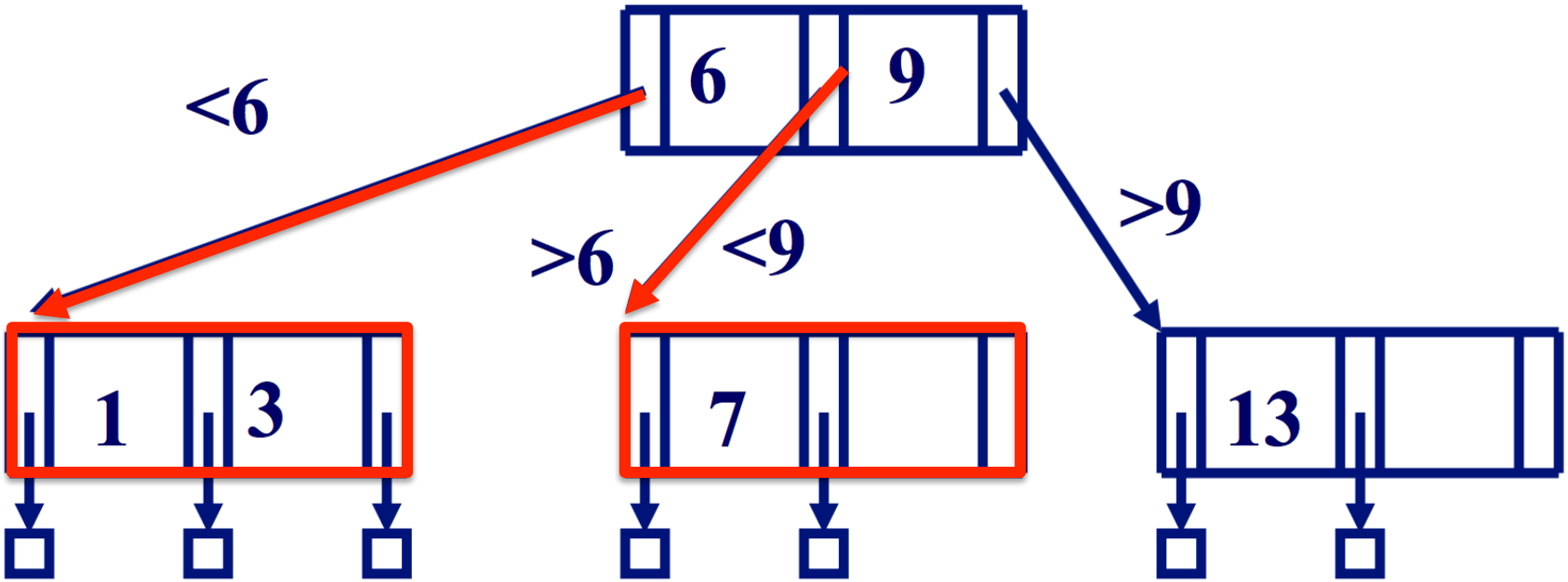
Point Queries

```
SELECT *
FROM Person
WHERE birthyear=8
```



Range Queries

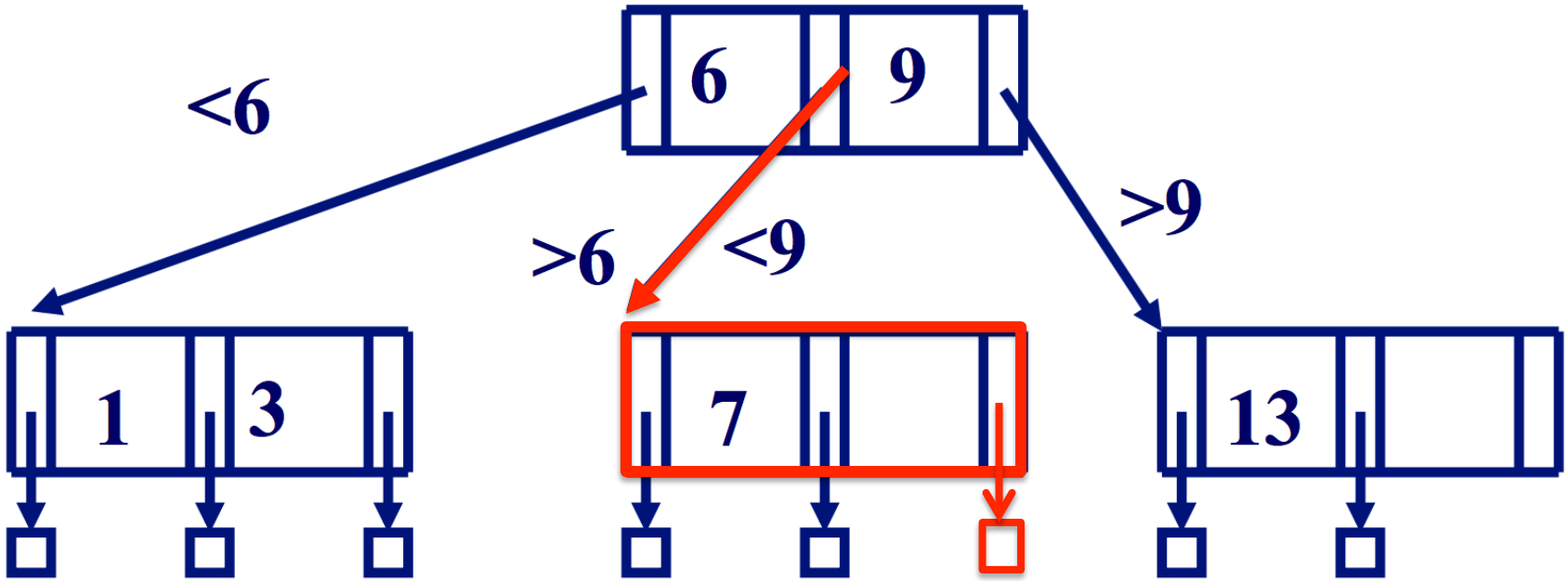
```
SELECT *
FROM Person
WHERE 5 < age < 8
```



Insertion

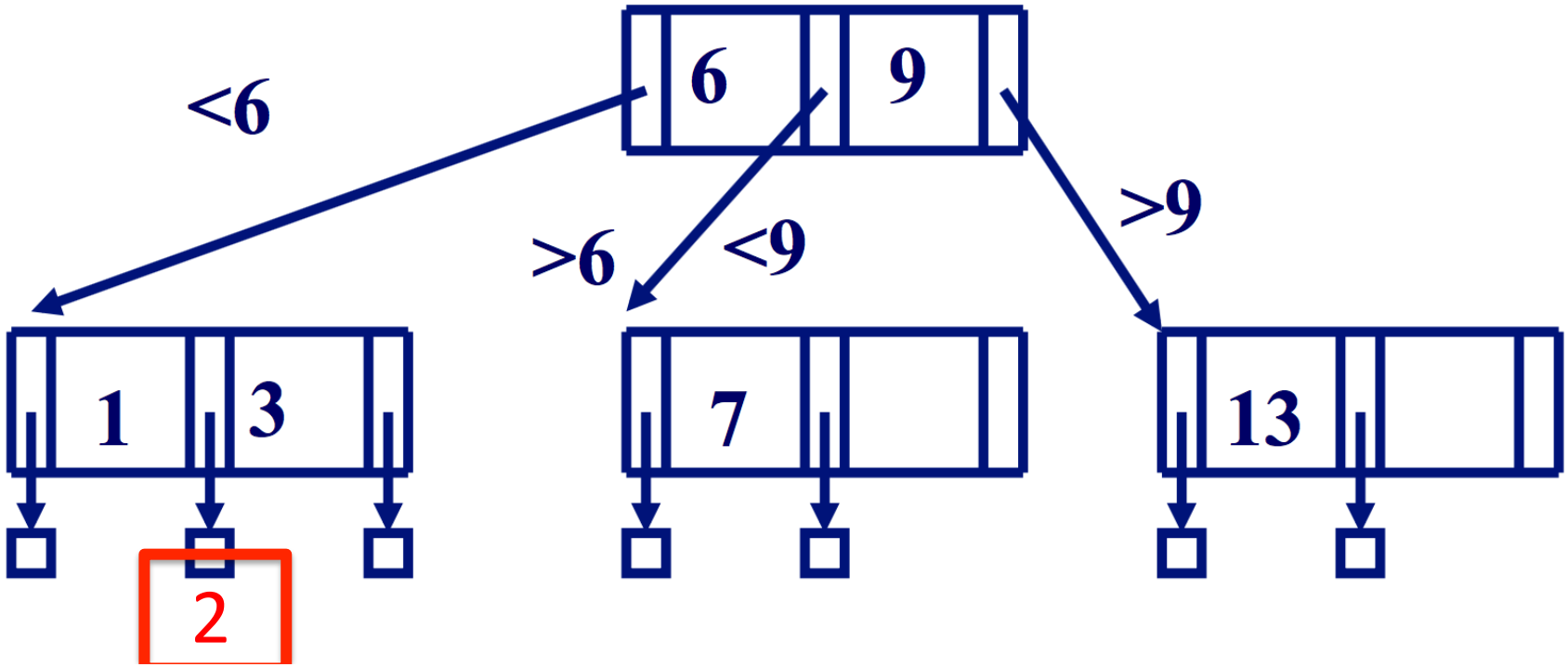
Insert 8 into the tree below!

8



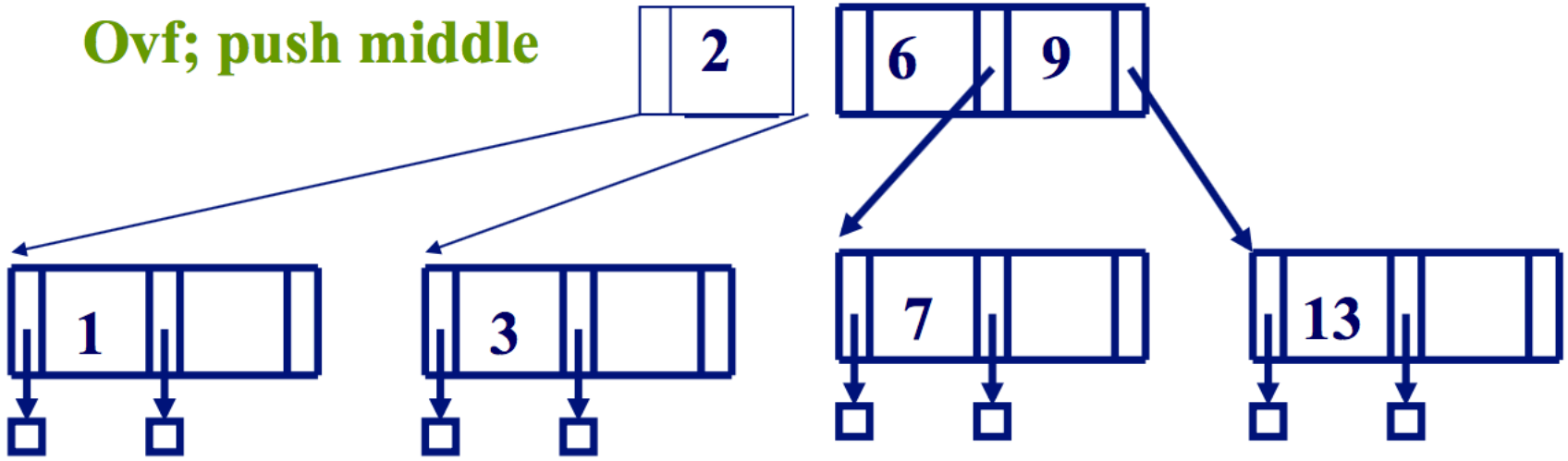
Insertion

Push the 2 up into the tree

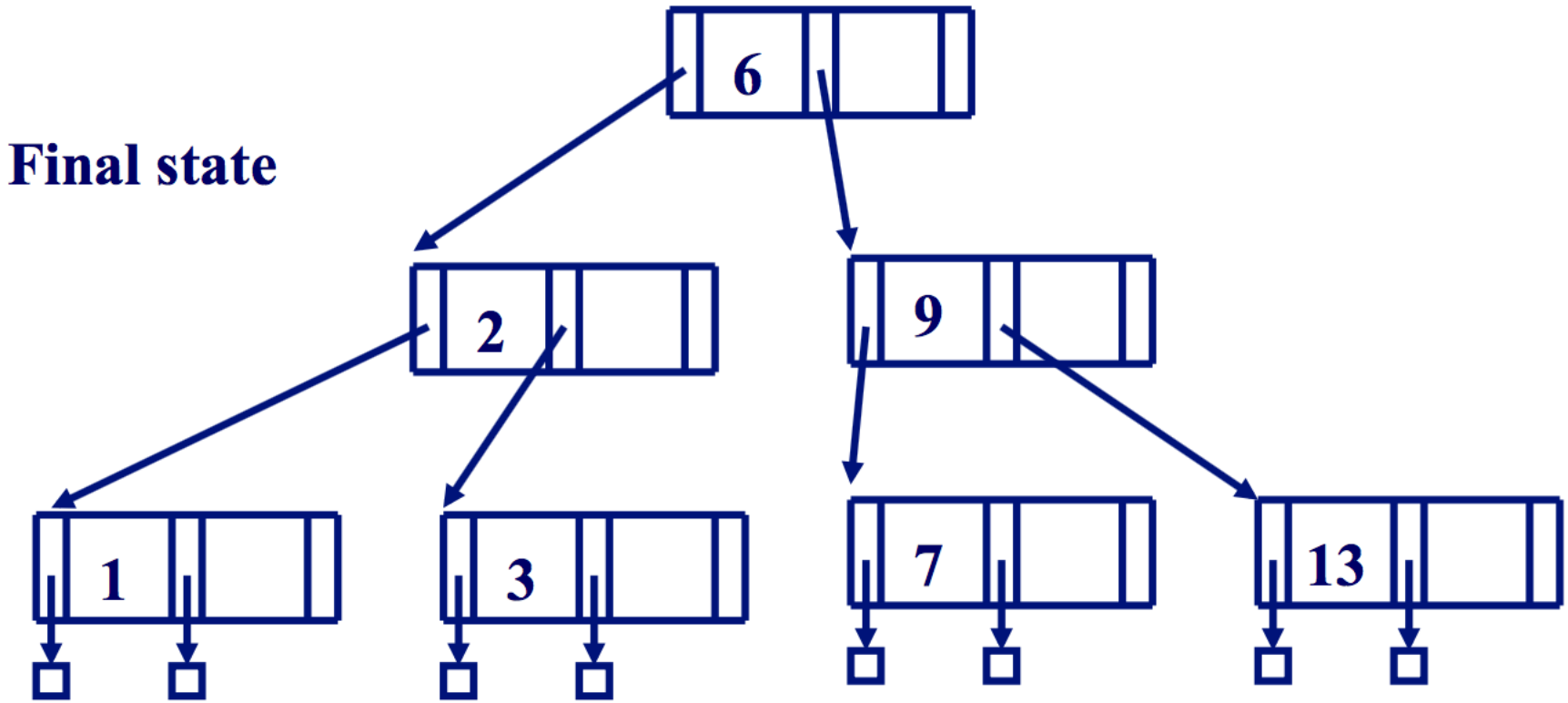


Insertion

Push the 2 up into the tree



Insertion



Insertion Algorithm B – Tree

```
fun insert(k:key) =
  (find the correct leaf node l;
   if (l overflows) then
     (split l and push middle to
         parent node p;
      if p overflows then
          push recursively)
   else insert k into l)
```

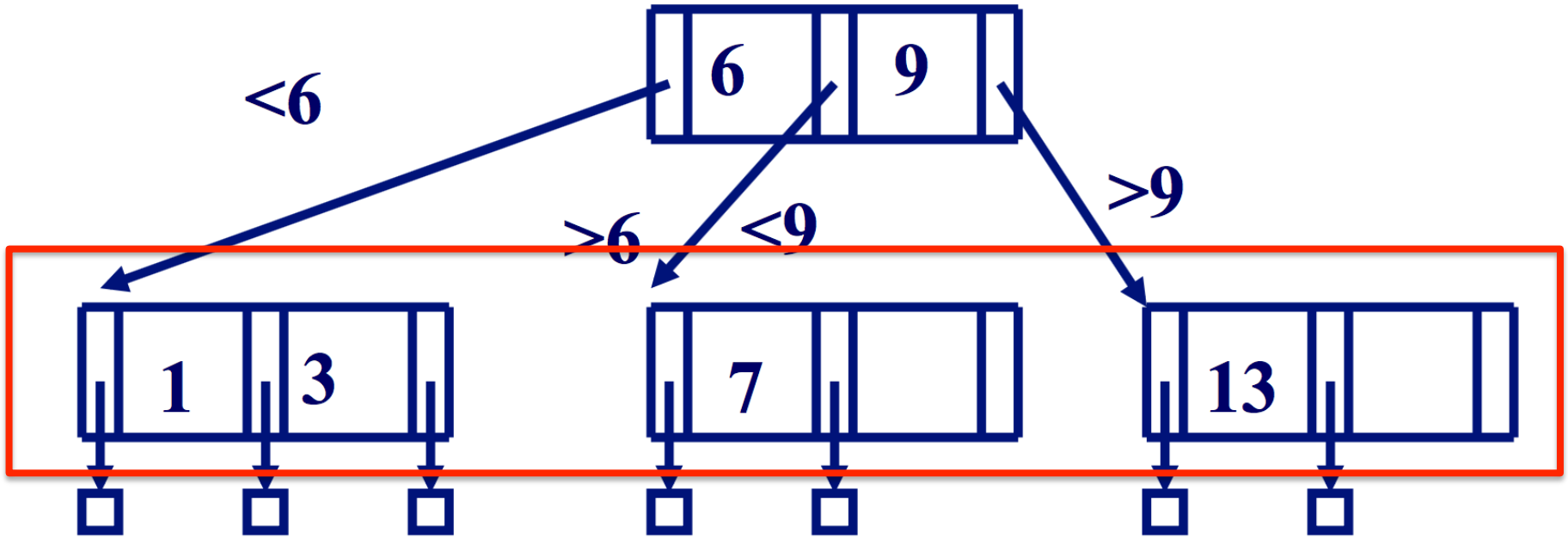
B-Trees

- Always balanced
- The nodes are always at least 50% full
- Height increases in case of overflow
- Deletion inverse of insertion but much much more complicated. See [R&G]

Contents

- Indexed Sequential Access Method (ISAM)
- B – trees
- B+ – trees
- MySQL index

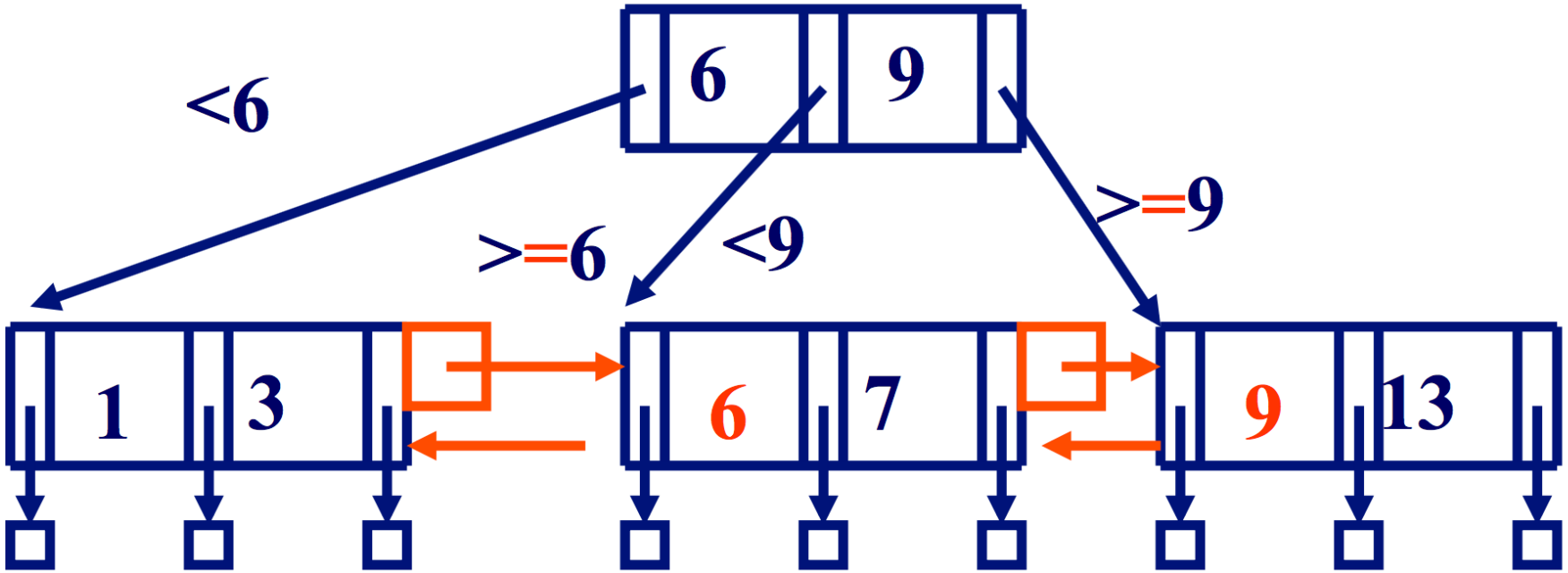
Clustered and Non-Clustered Index



- Clustered index: Leaf nodes are in order
- Non-clustered index: Leaf nodes are not in order

B+ – trees

- Link leaf nodes
- Each key appears in the leaf nodes

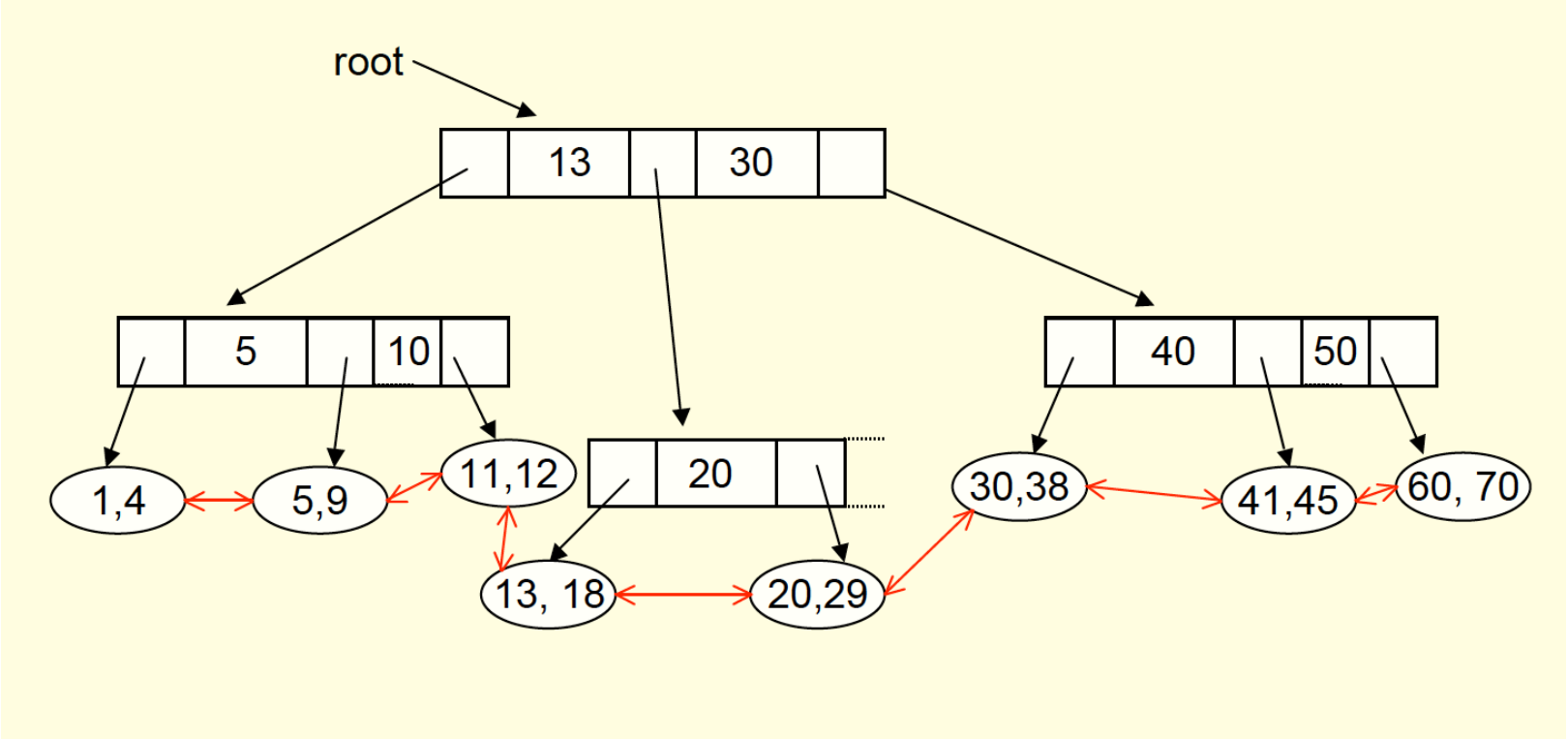


Insertion Algorithm B+ – Tree

```
fun insert(k:key) =
   (find the correct leaf node l;
    if (l overflows) then
       (split l and push middle key
                 to parent node p;
        if p overflows then
                 push recursively)
    else insert k into l)
```

copy

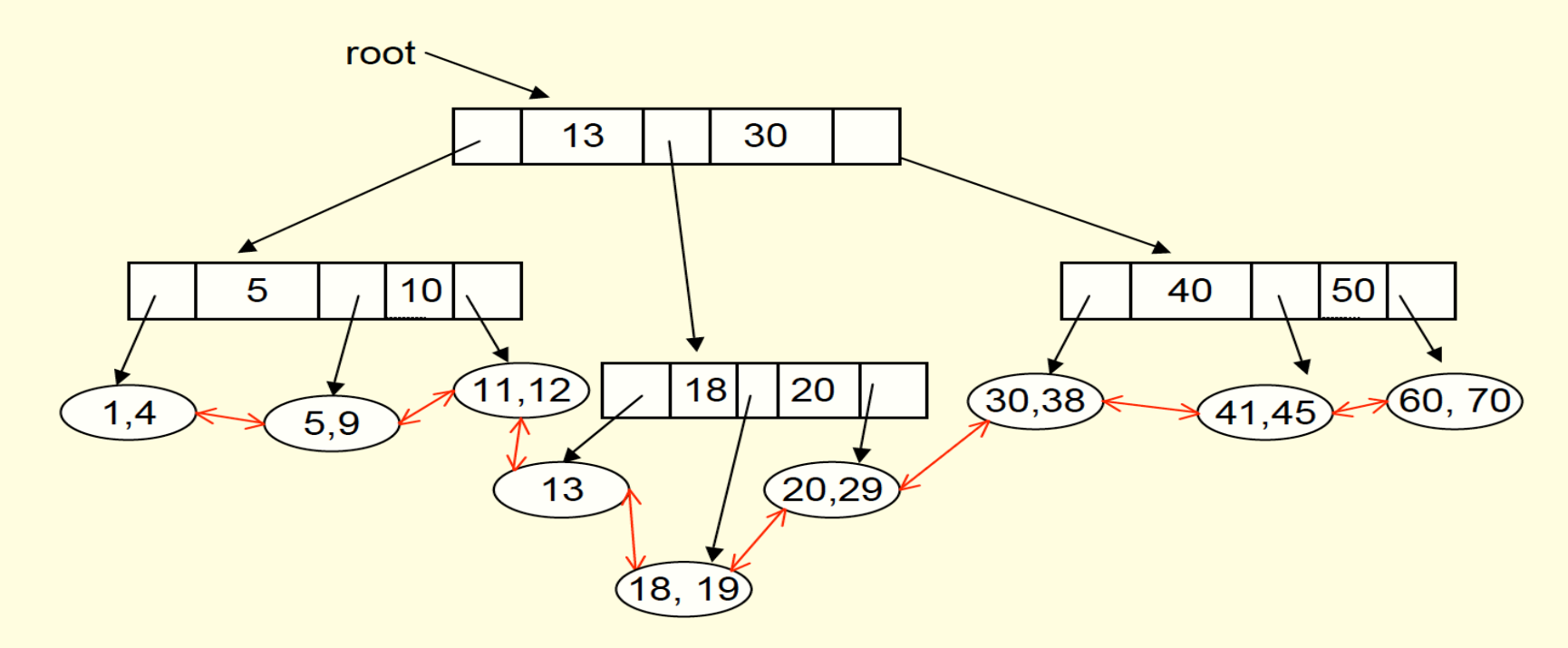
Example B+ - Tree



Insert 19

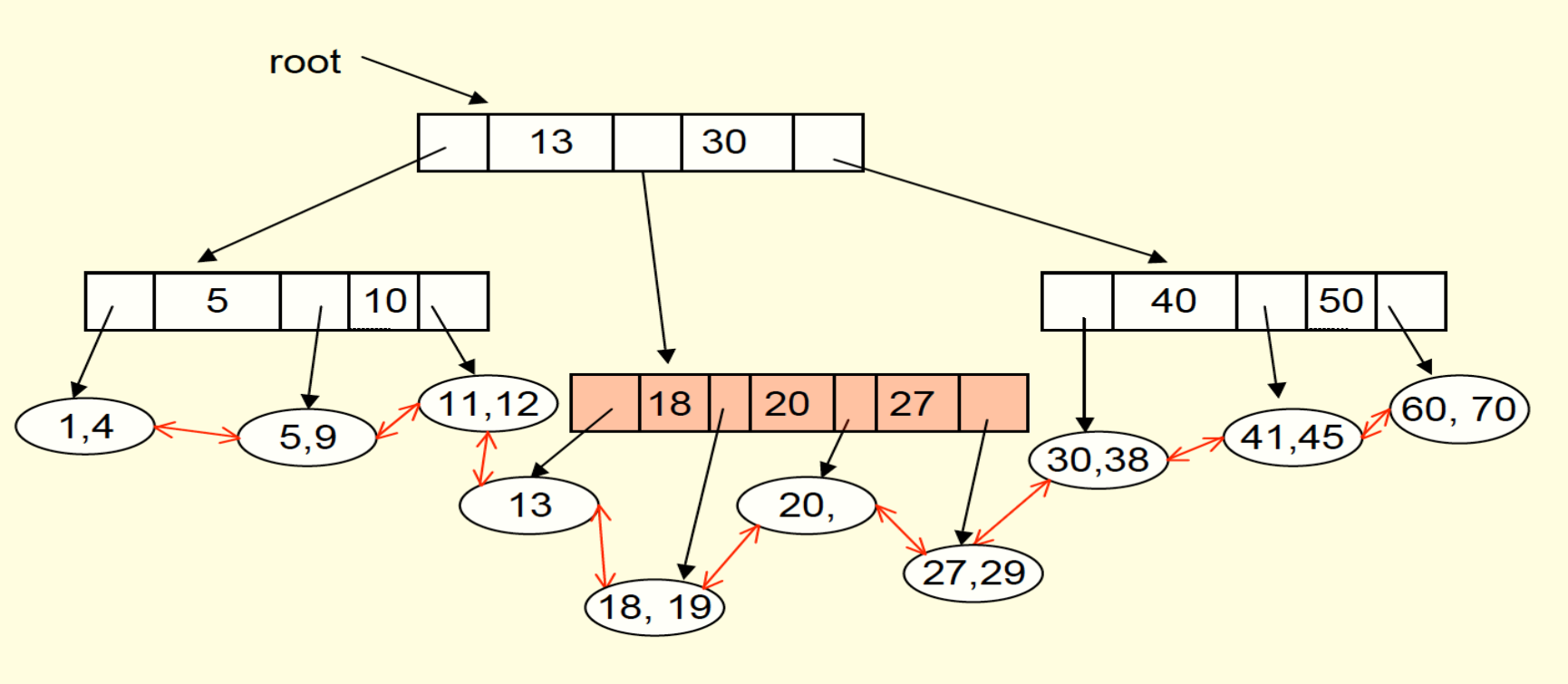
[thanks to Princeton]

Example B+ - Tree

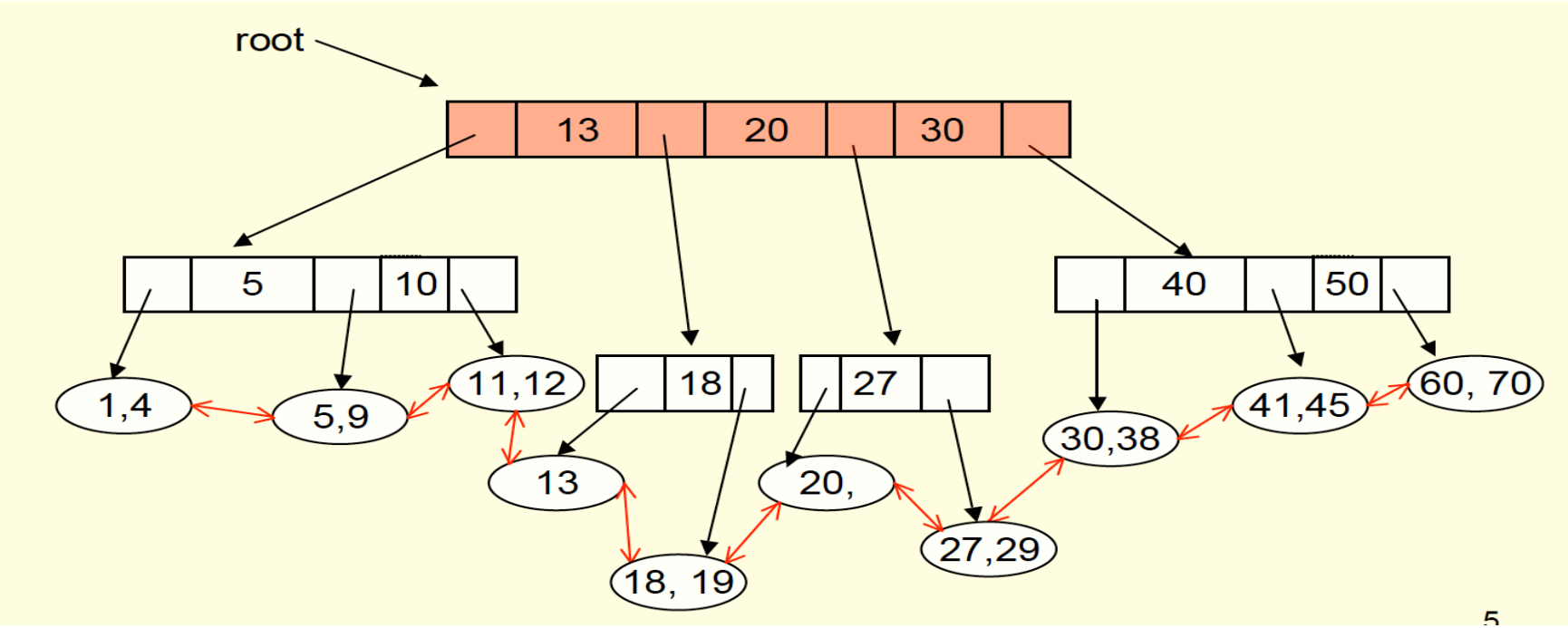


Insert 27

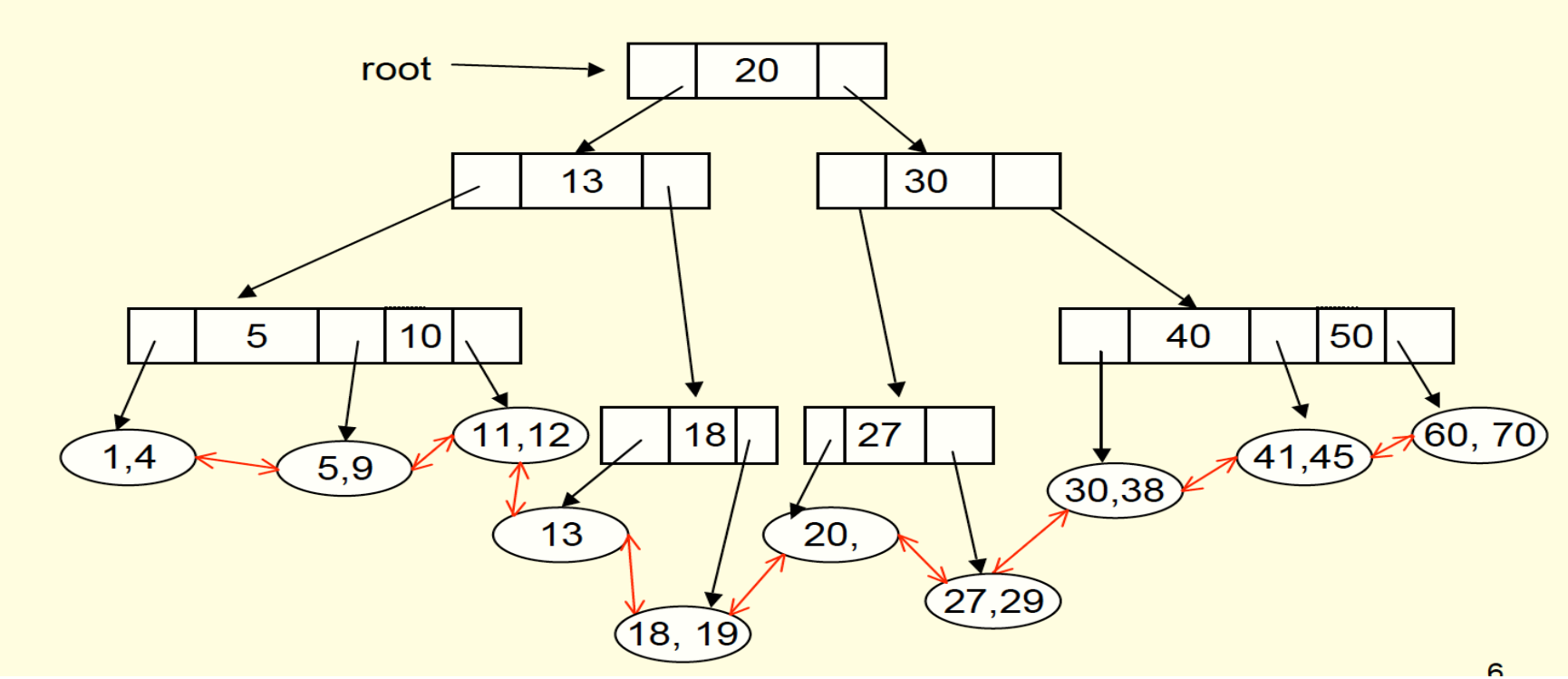
Example B+ - Tree



Example B+ - Tree



Example B+ - Tree



Summary

B+ trees

- Great data structure for indexes
- Order d of B+tree: minimal number of keys in a non-leaf node
- Maximal number of 2d keys in a non-leaf node
- Maximal fan-out of 2d+1 of children nodes
- Supports clustered index, full table scans, range queries, and point queries.

Contents

- Indexed Sequential Access Method (ISAM)
- B – trees
- B+ – trees
- MySQL index

Clustered Index / Primary Index

```
CREATE TABLE Person
(
  pid int NOT NULL,
  lastName varchar(255) NOT NULL,
  firstName varchar(255),
  birthdate DATE,
  city varchar(255),
  PRIMARY KEY (pid)
)
```

Secondary Index

```
CREATE INDEX myIndex1
ON Person(lastname);

CREATE INDEX myIndex2
ON Person(firstname, city);
```

Creates two non-clustered indexes on table `Person`.

Efficiency Gain

Index

```
CREATE INDEX myIndex2
ON Person(firstname, city);
```

Range query

```
SELECT *
   FROM person
   WHERE firstname='John'
   and birthdate < '1950-1-1'
```

DBMS computes best execution plan!

More about this next time!