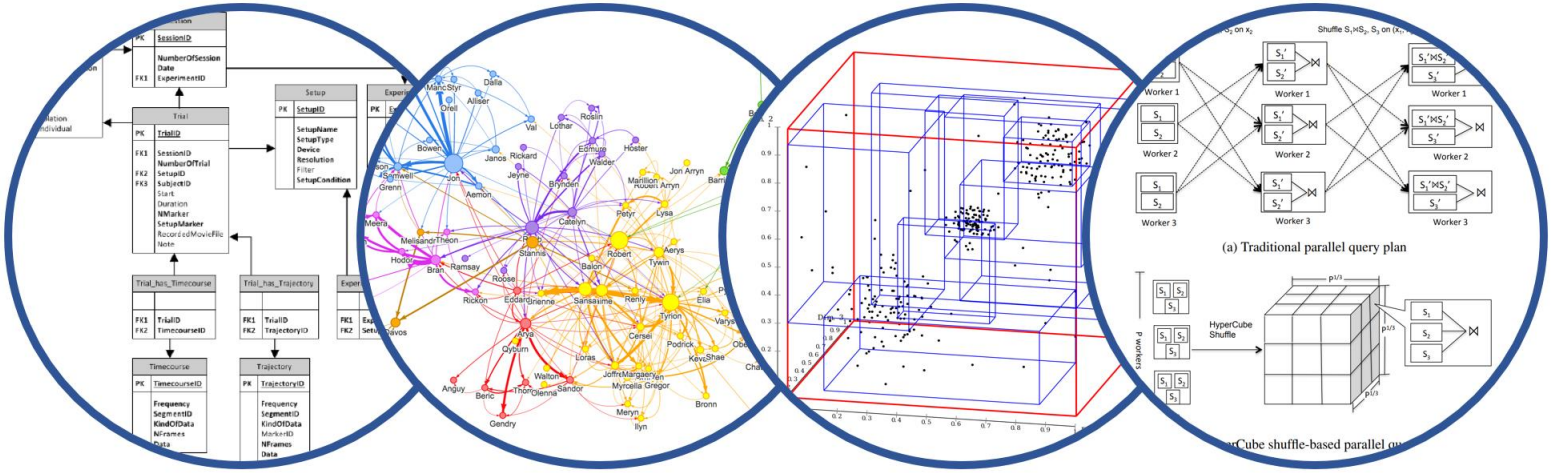


Introduction to Data Management

Disk Storage and Indexes

Paul G. Allen School of Computer Science and Engineering
University of Washington, Seattle

Announcements

HW6:

- Part 1 due today No late days (for quick feedback)
- Part 2 due 11/27. Much more work than part 1

Announcements

- Lecture on Wednesday, Nov. 27:
  - No in-person lecture
  - Recording only
- Monday, Dec. 9, Final Review session
  - CSE2 G10
  - 10:30 – 12:20 (we may stop earlier)

Review

Three join algorithms:

- Nested loop: always $O(n^2)$
- Hash join: usually $O(n)$, but can be $O(n^2)$
- Merge join: $O(n \log n)$

Review: Merge Join

$$\text{Payroll} \bowtie_{\text{UserID=UserID}} \text{Regist}$$

```
Payroll(UserID,Name,Job,Salary)
Regist(UserID,Car)
```

If |Payroll| = |Regist| = n
then runtime = O(n log n)

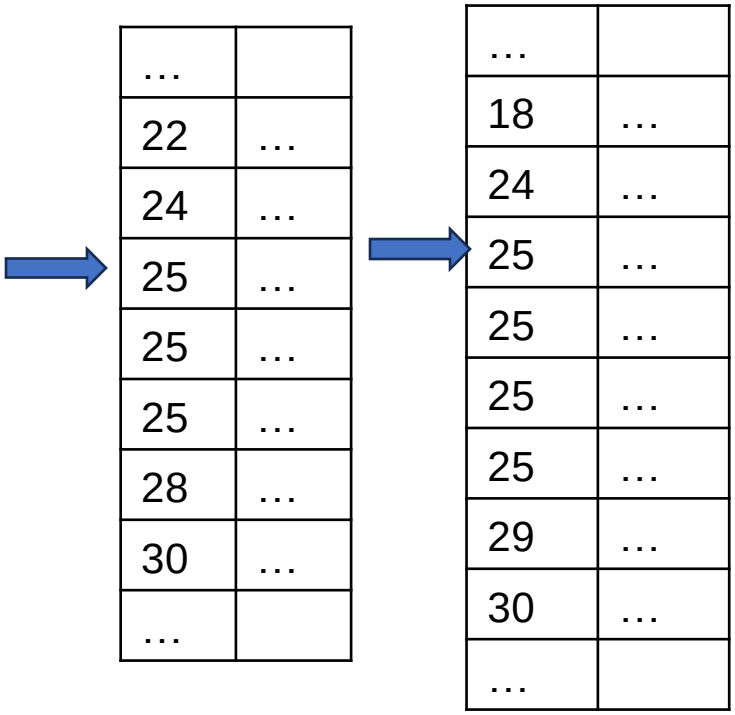
```
sort(Payroll); sort(Regist);
x = Payroll.first()
y = Regist.first()
while y!=NULL do:
  case:
    x.UserID < y.UserID: x.next();
    x.UserID = y.UserID: output(x,y); y.next();
    x.UserID > y.UserID: y.next();
```

FK-FK Merge Join

$A \bowtie_{\text{UserID}=\text{UserID}} B$

A(<u>UserID</u>,…)
B(UserID,…)

Neither is a key: called an FK-FK join.

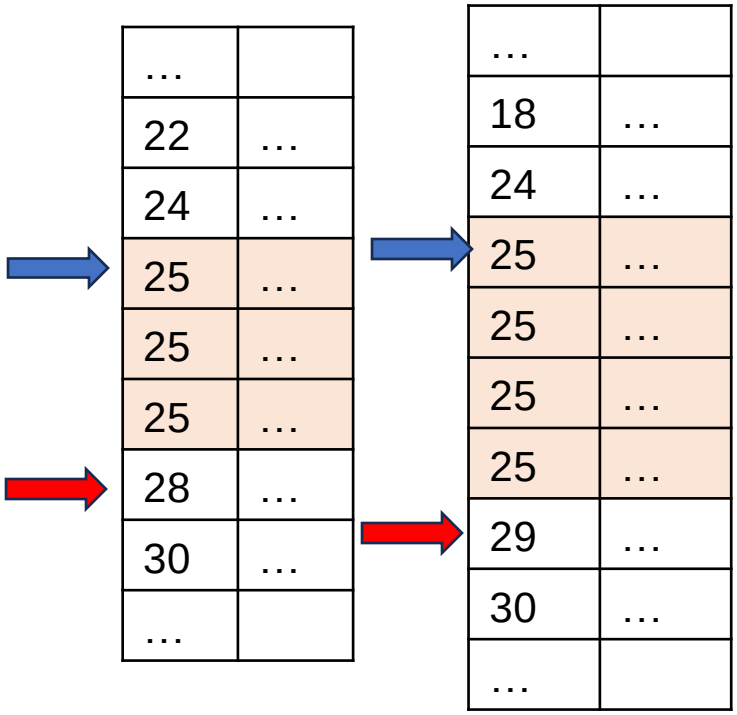


FK-FK Merge Join

$A \bowtie_{\text{UserID}=\text{UserID}} B$

```
A(UserID,…)
B(UserID,…)
```

Neither is a key: called an FK-FK join.



FK-FK Merge Join

$A \bowtie_{UserID=UserID} B$

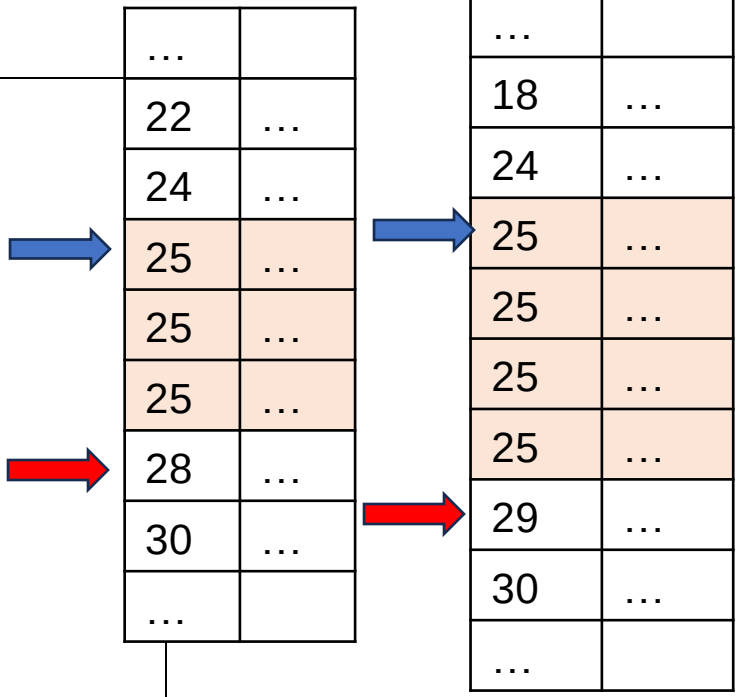
```
A(UserID,…)
B(UserID,…)
```

Neither is a key: called an FK-FK join.

```
sort… i=0; j=0;
while … do:
 case:…
  A[i]= B[j]:




  A[i]<A[j]: …
  A[i]>A[j]:…
```



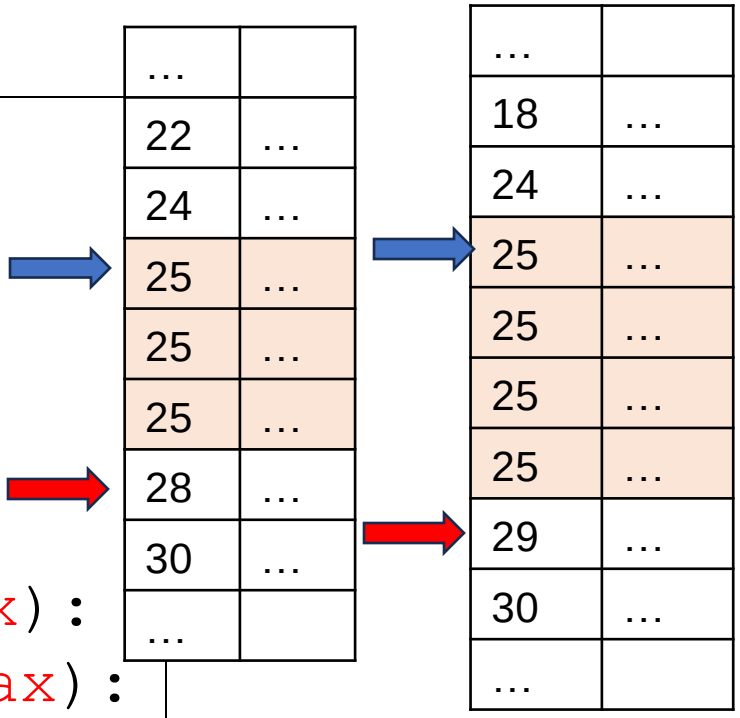
FK-FK Merge Join

$A \bowtie_{\text{UserID=UserID}} B$

A(UserID,…)
B(UserID,…)

Neither is a key: called an FK-FK join.

```
sort… i=0; j=0;
while … do:
 case:…
  A[i]= B[j]:
   i_min = i; i_max = …
   j_min = j; j_max = …
   for i in range(i_min, i_max):
    for j in range(j_min, j_max):
       output(A[i],B[j])
  A[i]<A[j]: …
  A[i]>A[j]:…
```



Other Physical Operators

Physical Operators

- Selection $\sigma_{pred}(R)$: iterate over R, return matches
- Projection $\Pi_{attrs}(R)$: iterate over R, return $attrs$
- Join $R \bowtie S$: we saw this
- Duplicate elimination δ or group by $\gamma_{attrs,agg}$
  - Nested loop, or Hash-based, or Sort-based
- Union $R \cup S$:
  - Bag semantics v.s. Set semantics (discuss)
- Difference: like join, but more complicated

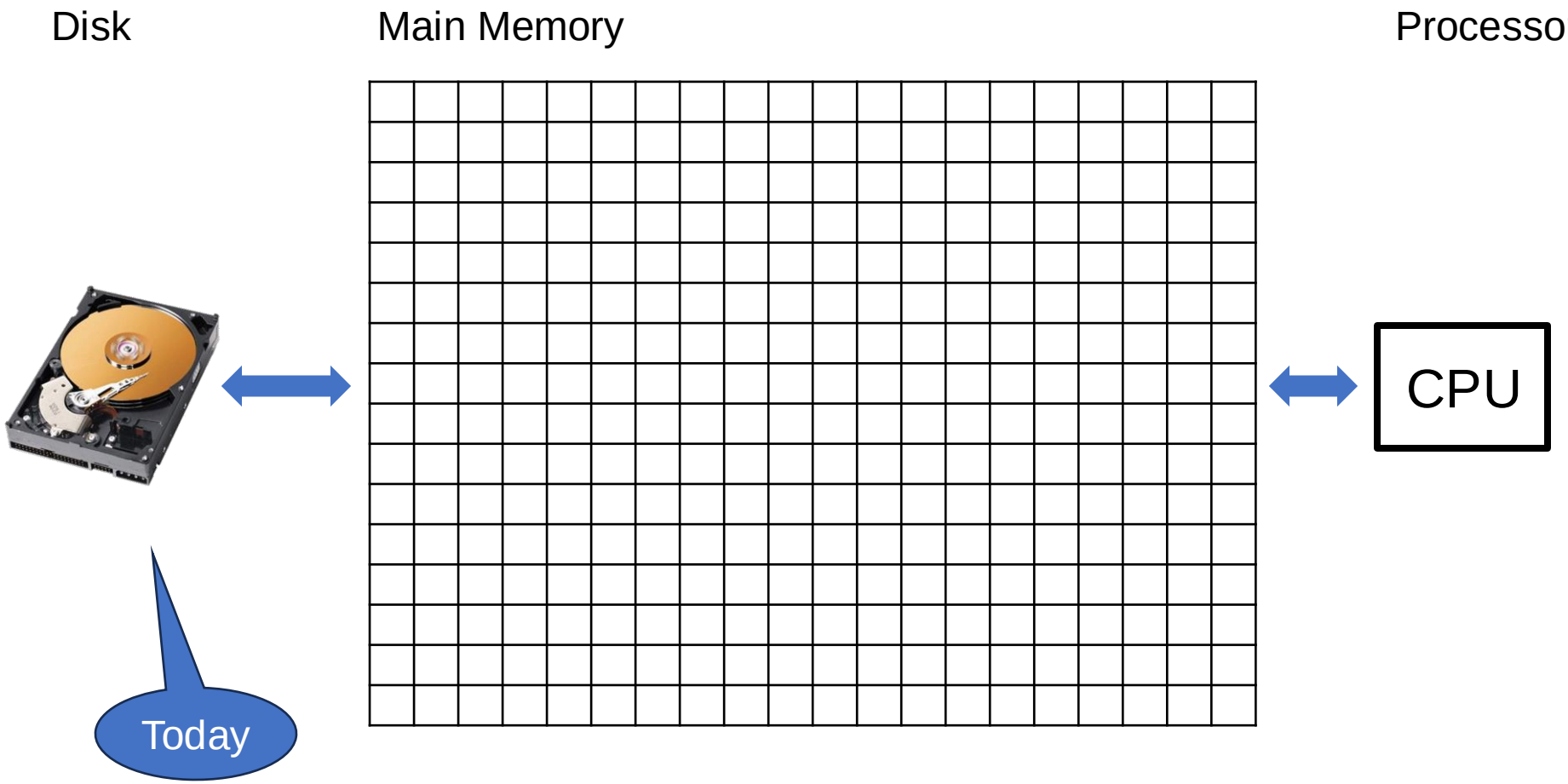
Final Comments

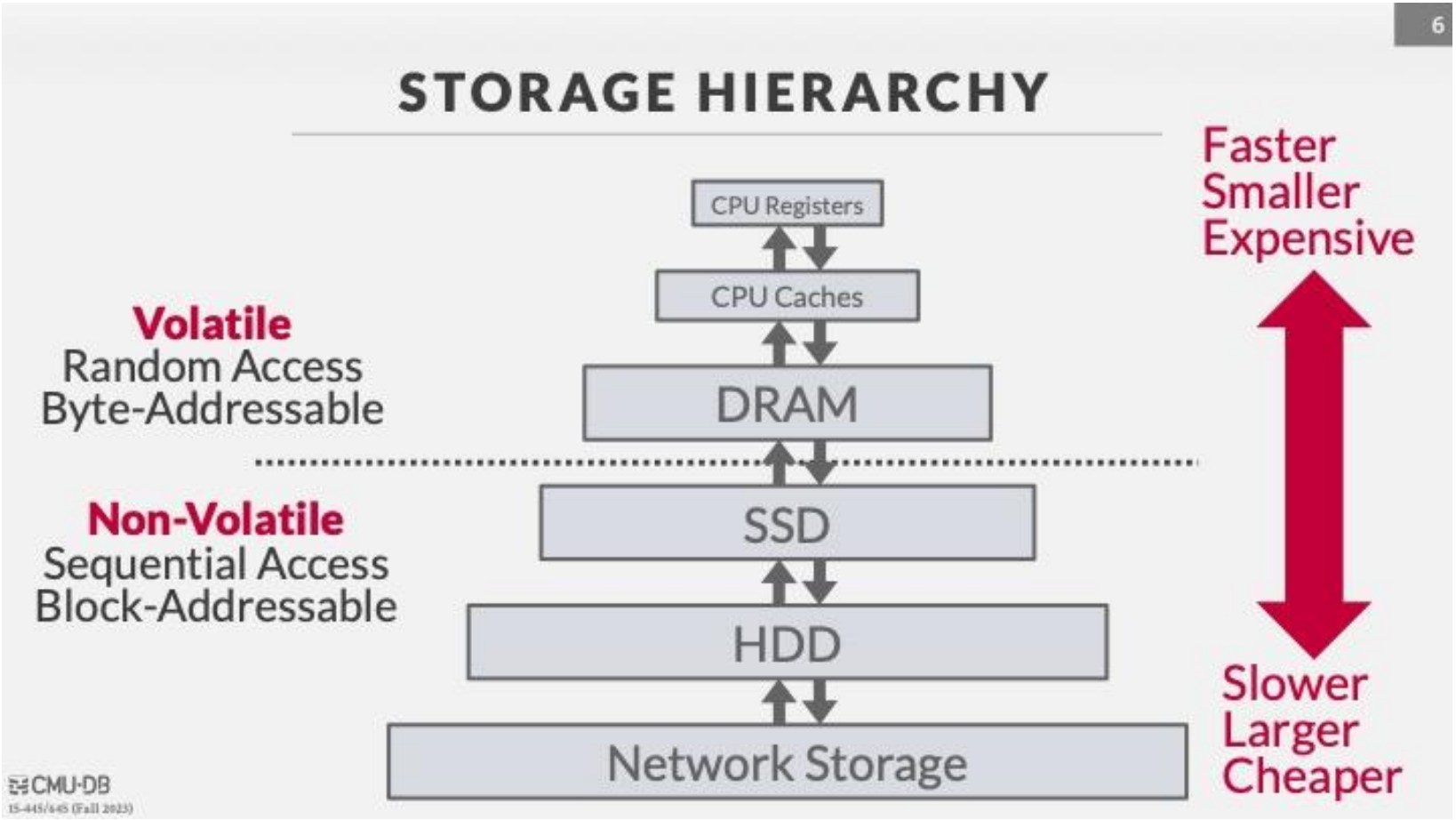
Each operator has several implementations

- Join = most important and complex
- When data is on disk: specialized algorithms
- The physical operator: chosen by optimizer

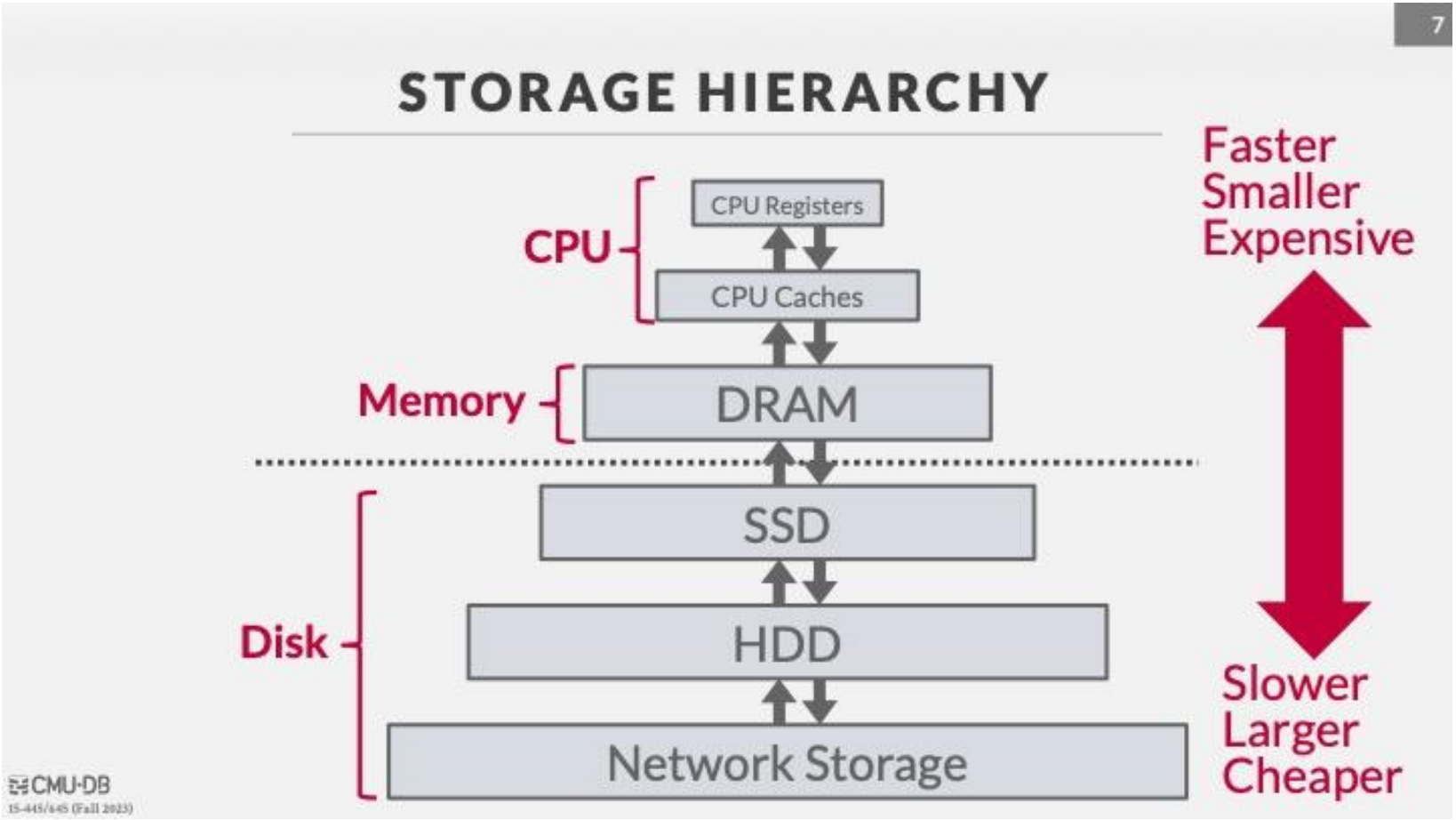
Disk Storage

How the Main Memory Works

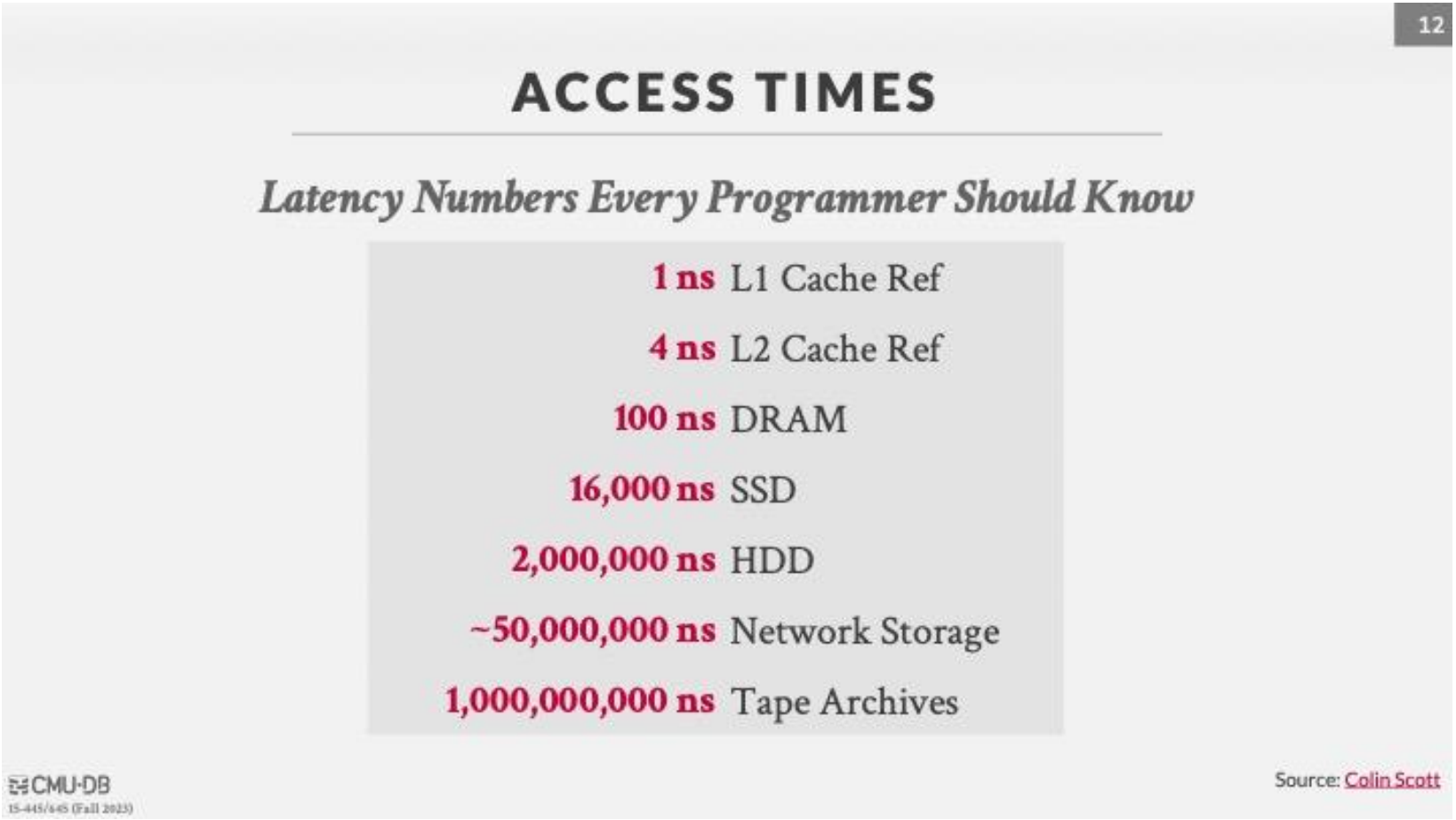




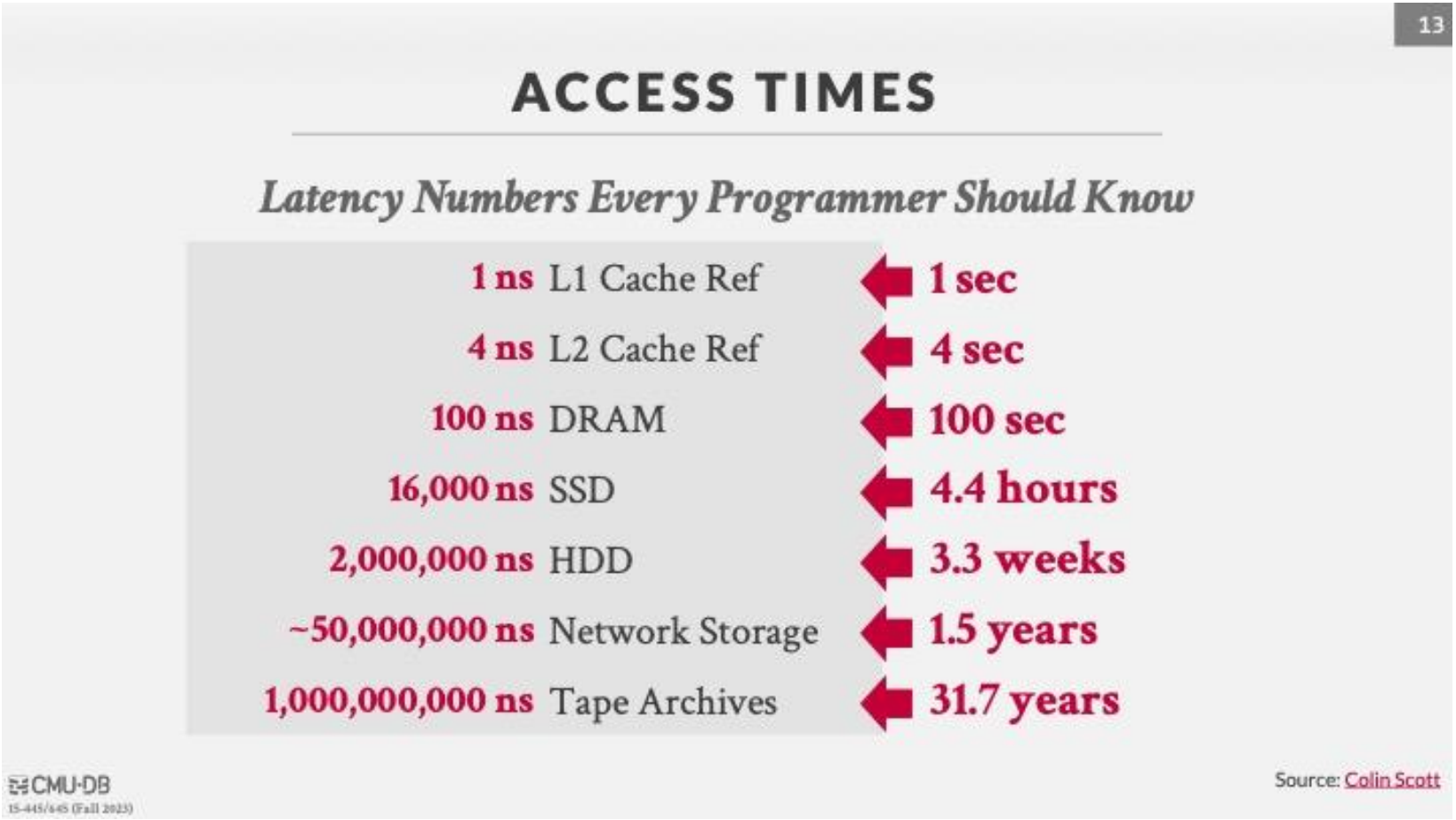
Credit: https://15445.courses.cs.cmu.edu/fall2023/



Credit: https://15445.courses.cs.cmu.edu/fall2023/



Credit: https://15445.courses.cs.cmu.edu/fall2023/



Credit: https://15445.courses.cs.cmu.edu/fall2023/

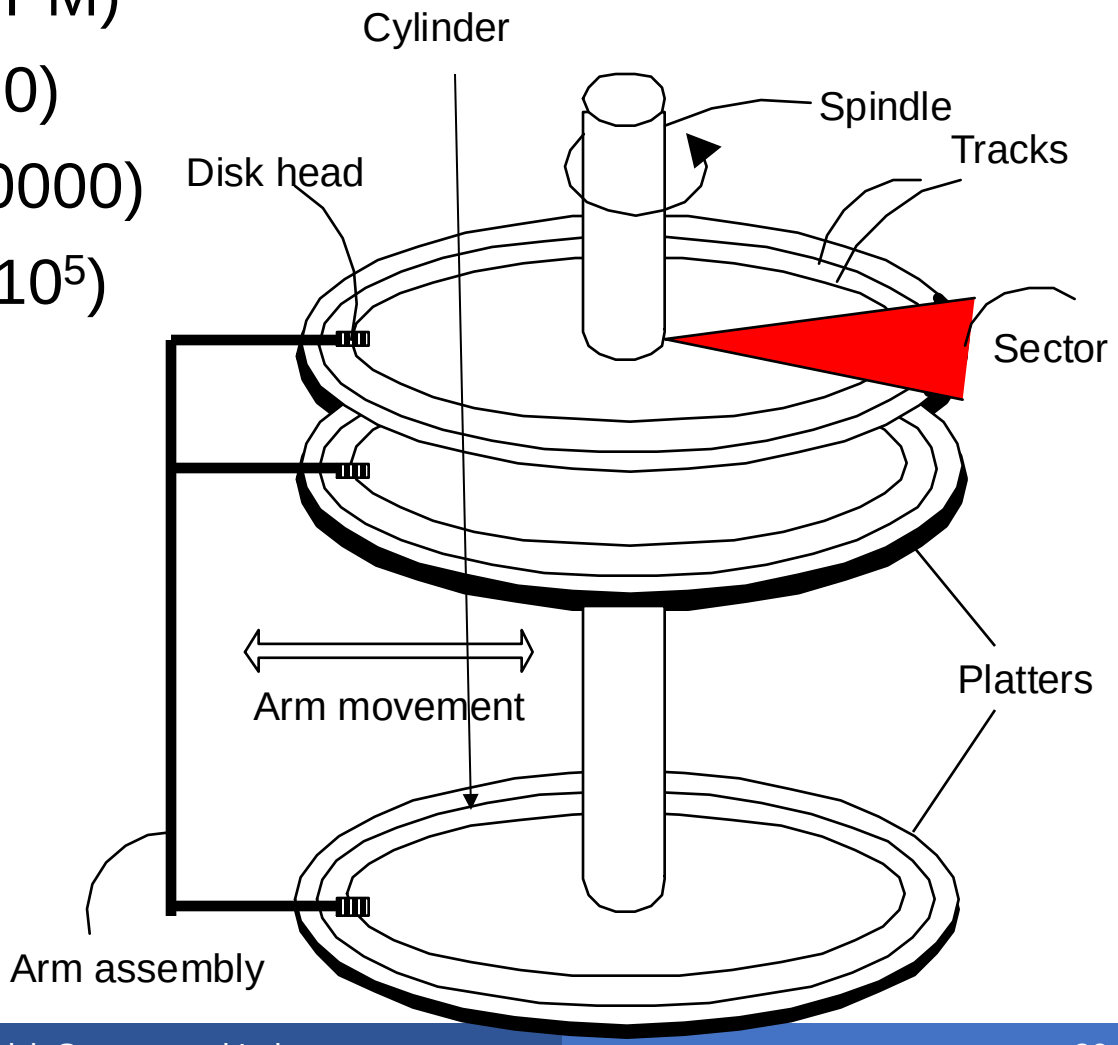
Disk

- Stores data persistenly
- Different technologies:
  - Tapes: long-term archives
  - HDD: most today's data is stored here
  - SDD: your laptop, or cache for HDD
- Unit of Read/Write operation:
  1 block = 0.5k .. 32k

Hard Drive Disk (HDD)

Mechanical characteristics:

- Rotation speed (5400RPM)
- Number of platters (1-30)
- Number of tracks (<=10000)
- Number of bytes/track(10^5)



Disk Access Characteristics

Disk latency = seek time + rotational latency

- Seek time = time for the head to reach cylinder
  - 10ms – 40ms
- Rotational latency = time for sector to rotate
  - Rotation time = 10ms

Throughput = typically 40-80MB/s

Blocks or Pages

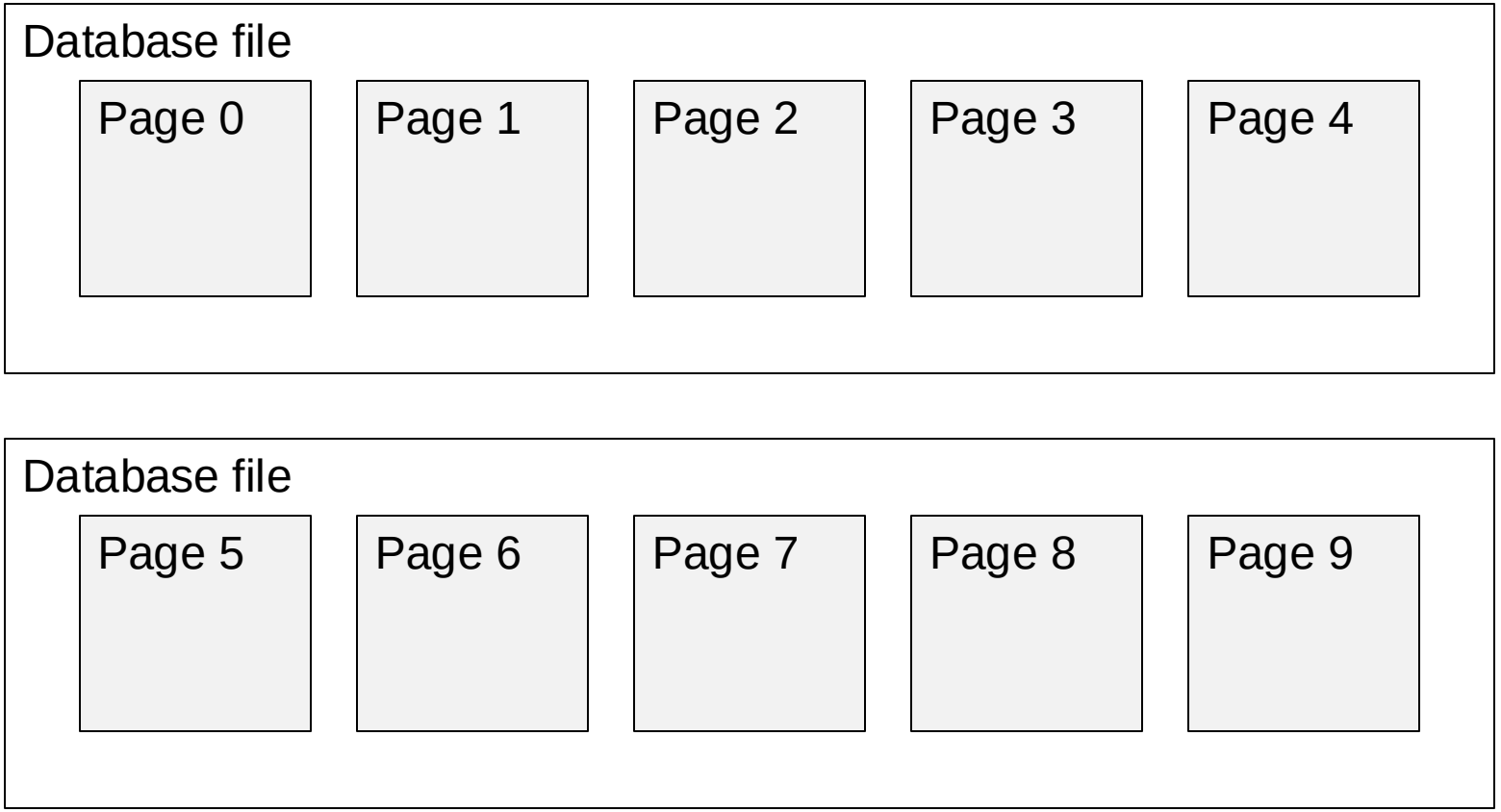
- The unit of disk read or write is a block
- Once in main memory, we call it a page
- Block size is fixed. Typically, 4k or 8k or 16k

Storage Manager

DBMS creates one (or more?) large files

- 1 file = multiple (continuous?) blocks
- 1 block = multiple records, free space

Storing Pages on Disk

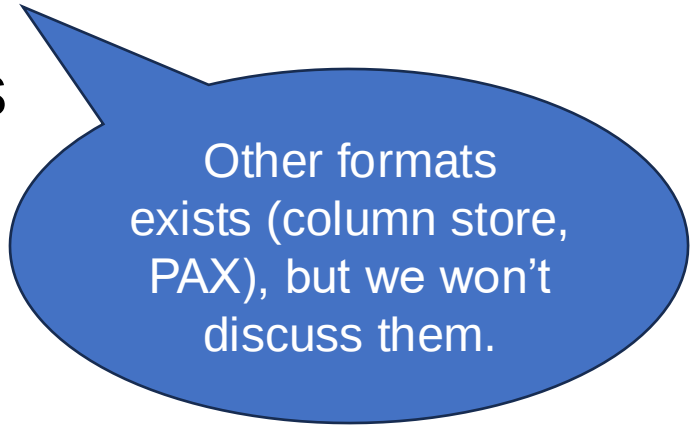


Page Format

Row-oriented Format

A.k.a. N-ary storage model NSM

- Each page stores several records
- Each records stores its attributes



Row-Oriented Storage

Payroll

UserID	Name	Job
123	Jack	TA
345	Allison	TA
567	Magda	Prof
789	Dan	Prof

Row-Oriented Storage

Page 0

123	Jack	TA
345	Allison	TA
567	Magda	Prof
…		

Page 1

789	Dan	Prof
…		
…		

…

Payroll

UserID	Name	Job
123	Jack	TA
345	Allison	TA
567	Magda	Prof
789	Dan	Prof

Row-Oriented Storage

Physical layout

Page 0

123	Jack	TA
345	Allison	TA
567	Magda	Prof
…		

Page 1

789	Dan	Prof
…		
…		

…

Logical schema

Payroll

UserID	Name	Job
123	Jack	TA
345	Allison	TA
567	Magda	Prof
789	Dan	Prof

Row-Oriented Storage

Physical layout

Logical schema

Page 0

123	Jack	TA
345	Allison	TA
567	Magda	Prof
…		

Page 1

789	Dan	Prof
…		
…		

…

Payroll

UserID	Name	Job
123	Jack	TA
345	Allison	TA
567	Magda	Prof
789	Dan	Prof

The schema stored separately, in the *database catalog*

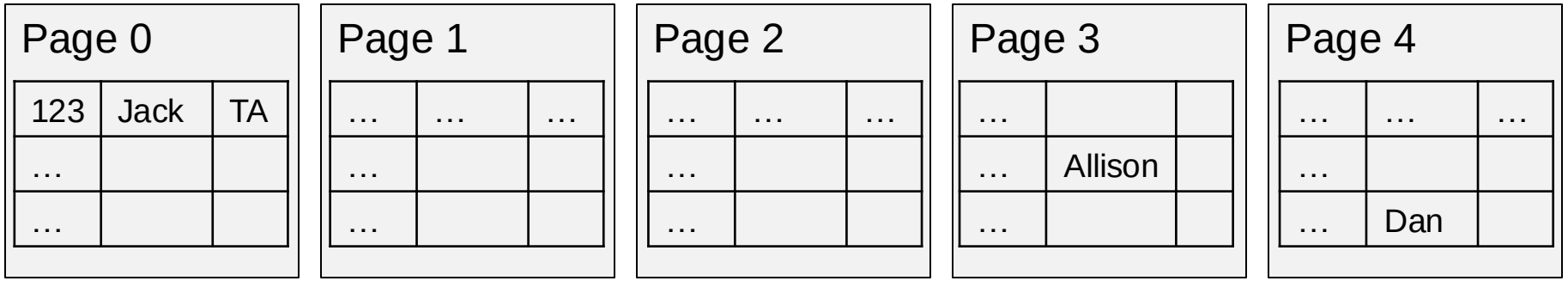
Indexes

Indexes

- The relation is stored in a file (= set of blocks)
- An index is an auxiliary file that facilitates faster access to the data

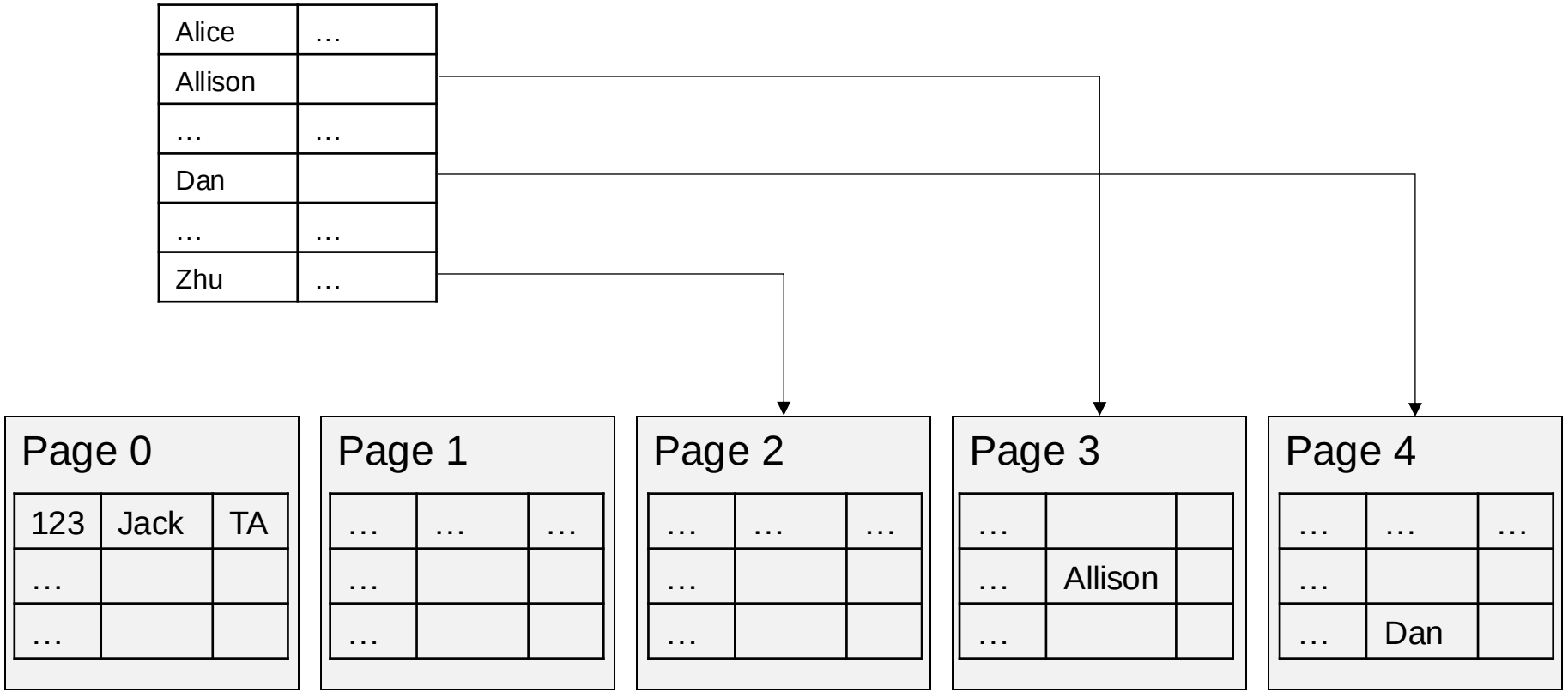
Indexes

An index is an auxiliary file that facilitates faster access to the data



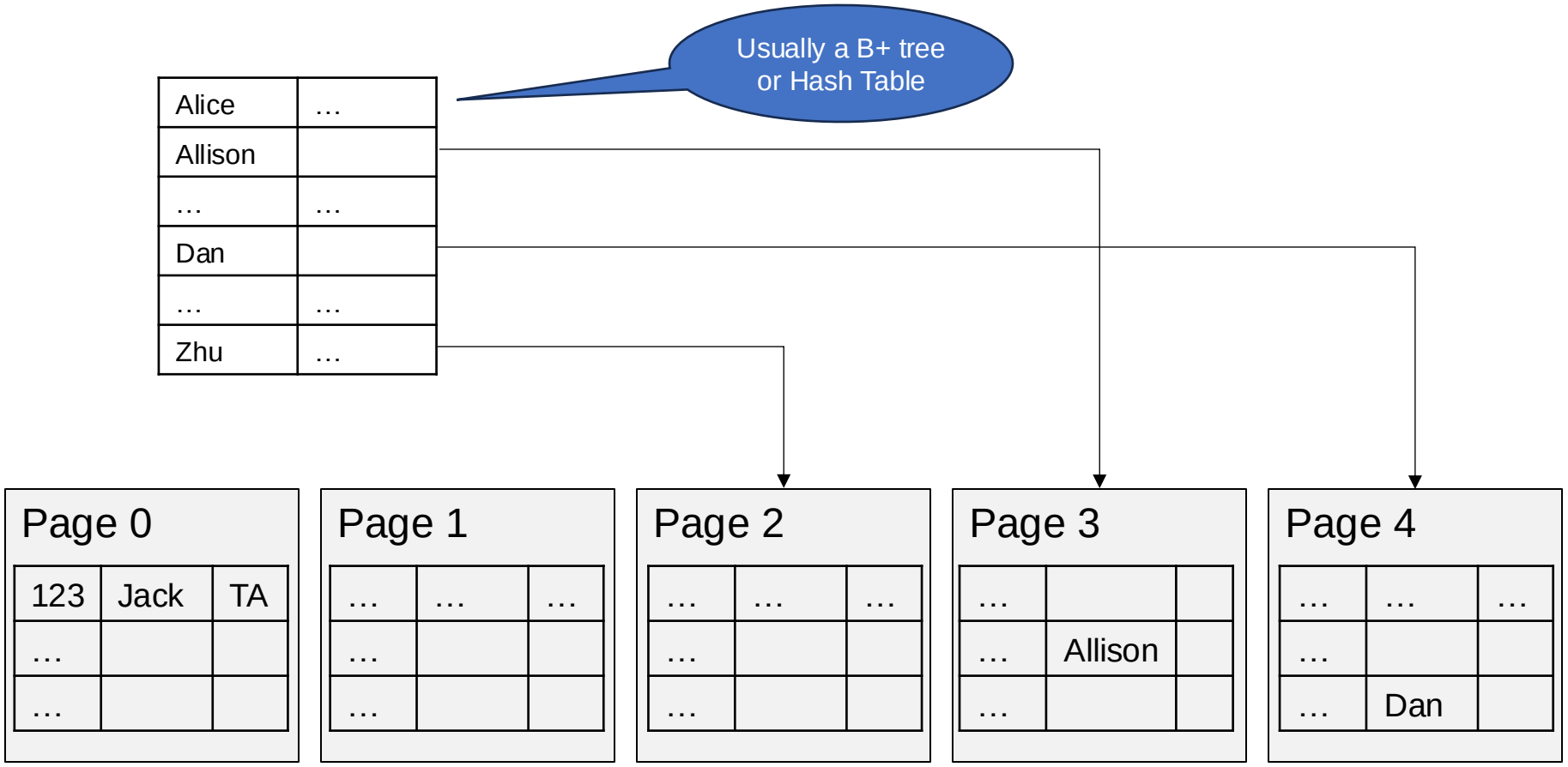
Indexes

An index is an auxiliary file that facilitates faster access to the data



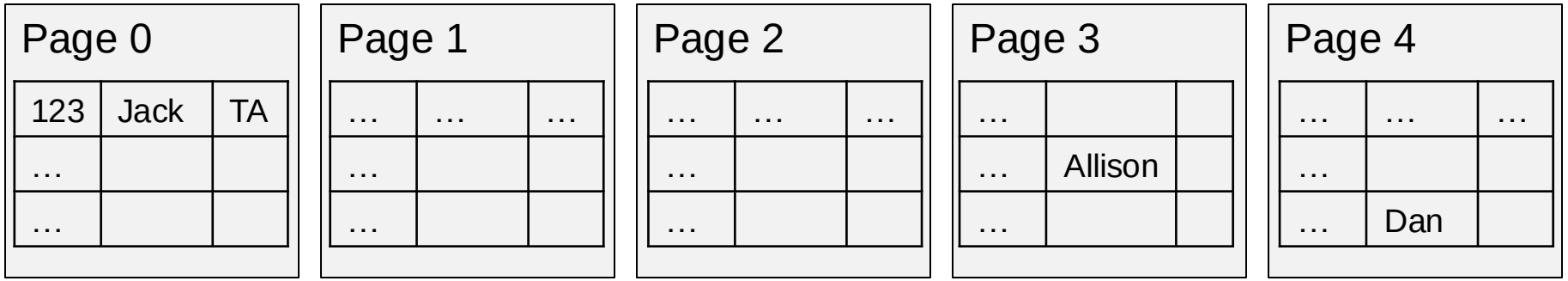
Indexes

An index is an auxiliary file that facilitates faster access to the data



Indexes

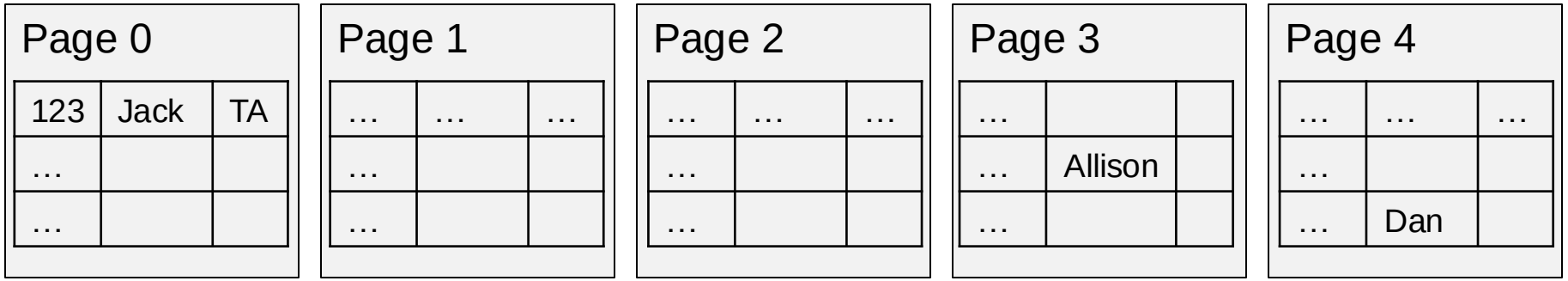
An index is an auxiliary file that facilitates faster access to the data



Indexes

An index is an auxiliary file that facilitates faster access to the data

```
SELECT *
FROM Payroll
WHERE Name = 'Allison';
```

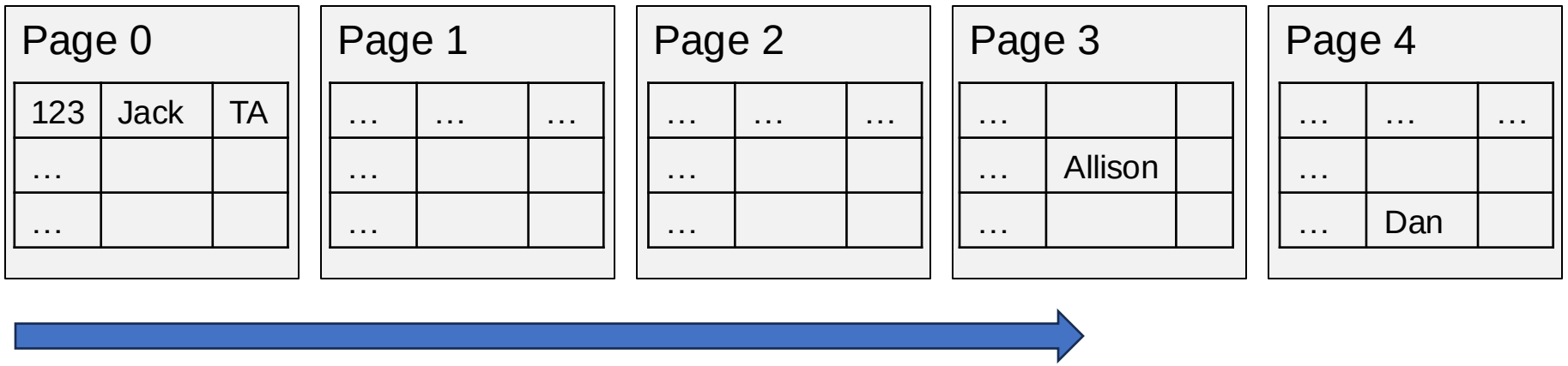


Indexes

An index is an auxiliary file that facilitates faster access to the data

```
SELECT *
FROM Payroll
WHERE Name = 'Allison';
```

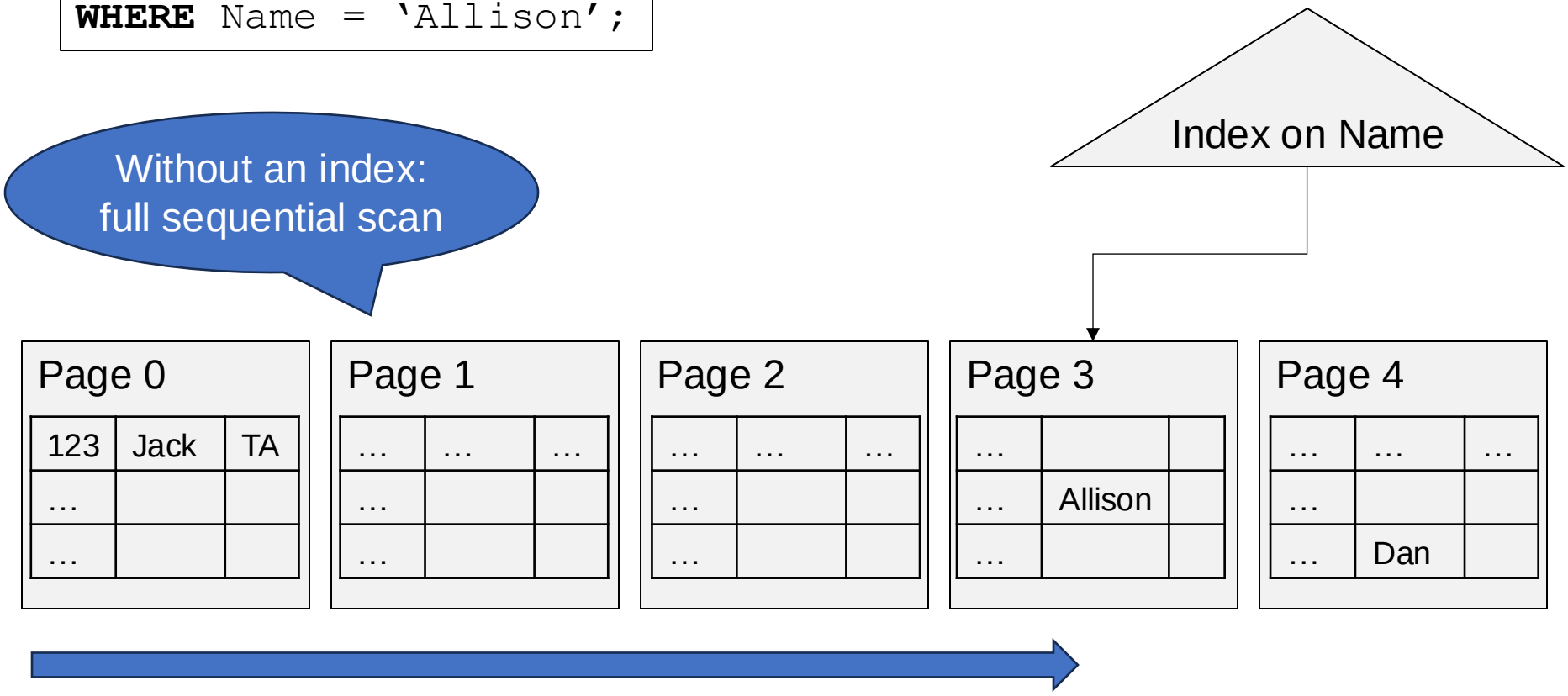
Without an index:
full sequential scan



Indexes

An index is an auxiliary file that facilitates faster access to the data

```
SELECT *
FROM Payroll
WHERE Name = 'Allison';
```



Indexes

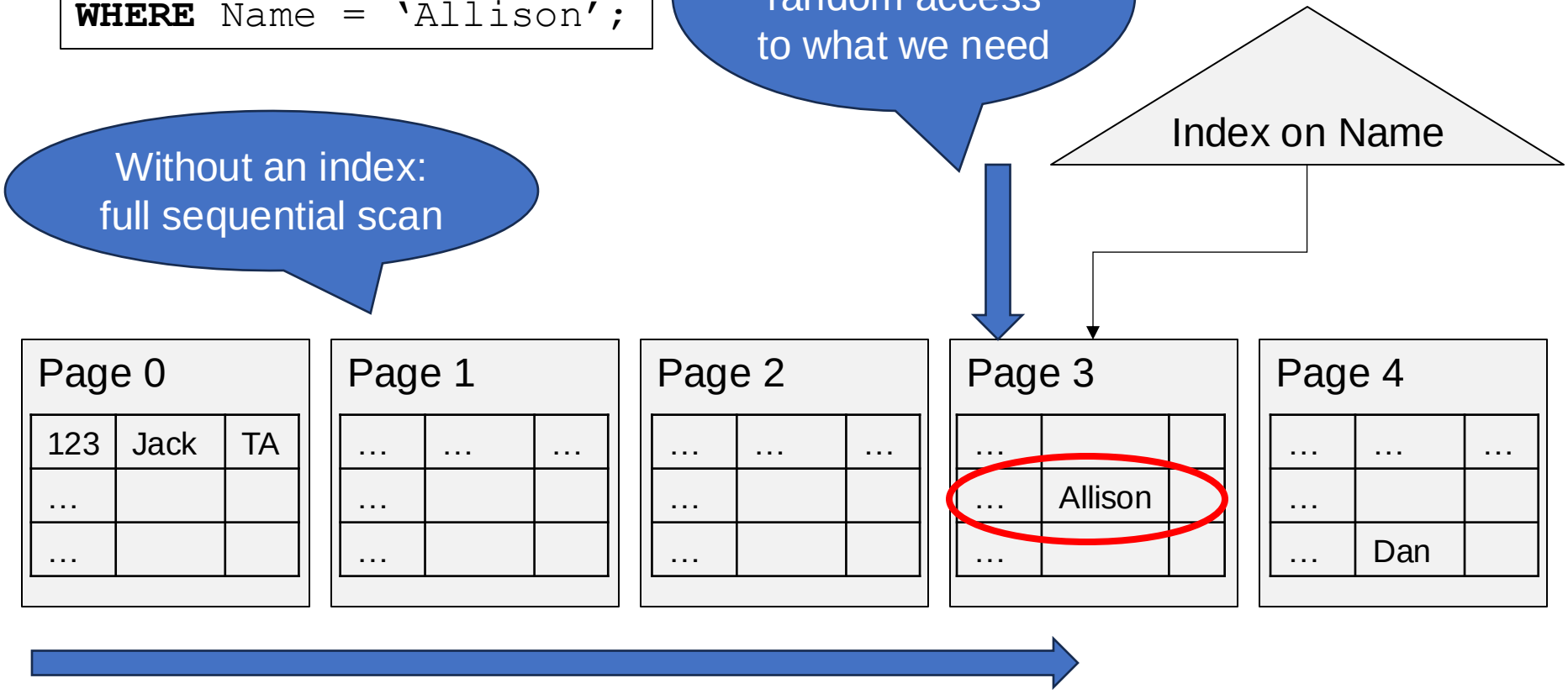
An index is an auxiliary file that facilitates faster access to the data

```
SELECT *
FROM Payroll
WHERE Name = 'Allison';
```

With an index:
random access
to what we need

Without an index:
full sequential scan

Index on Name

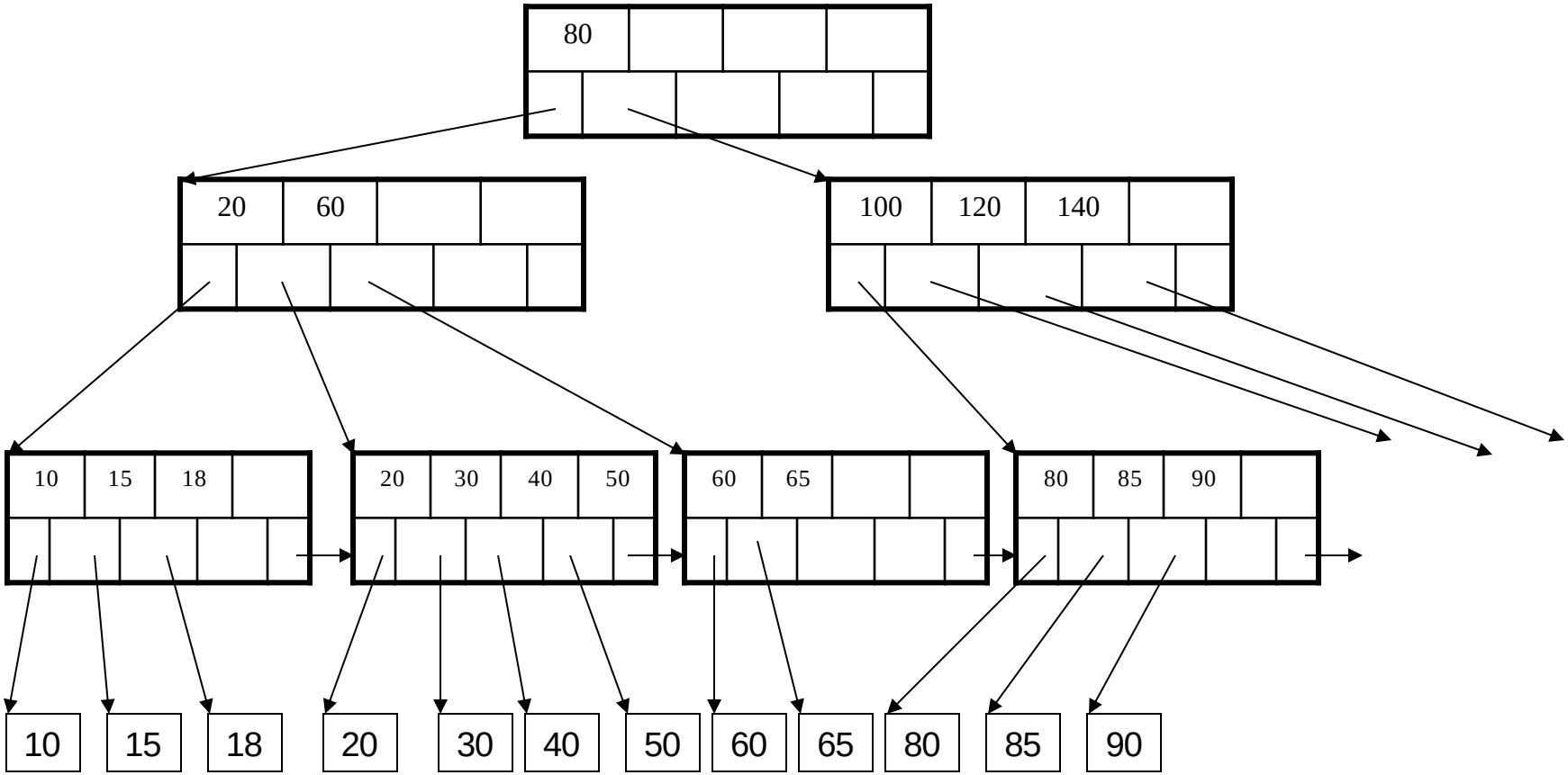


B+ Trees Basics

(only some basics: more on Monday)

B+ Tree Example

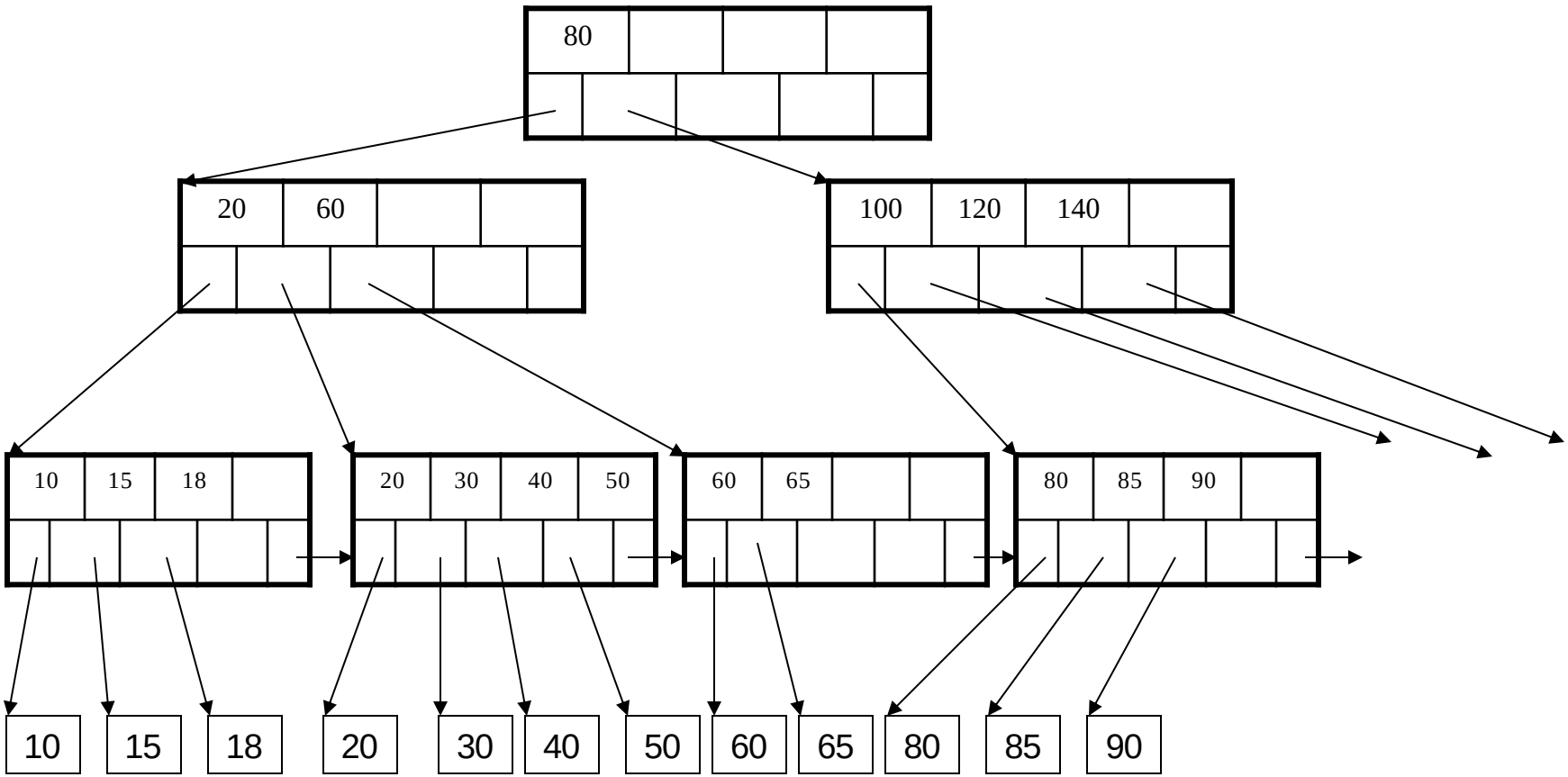
d = 2



B+ Tree Example

d = 2

Find the key 40

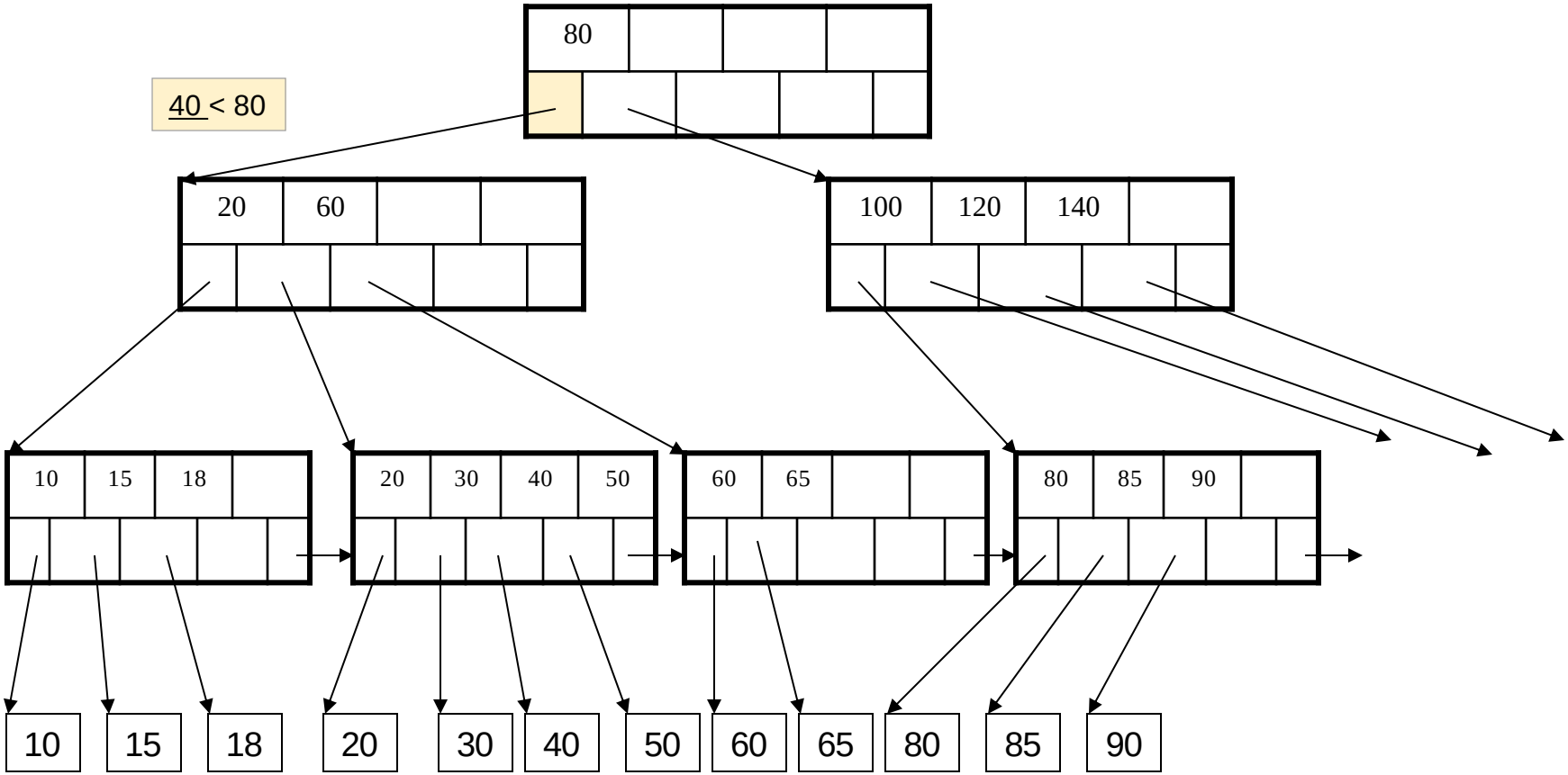


B+ Tree Example

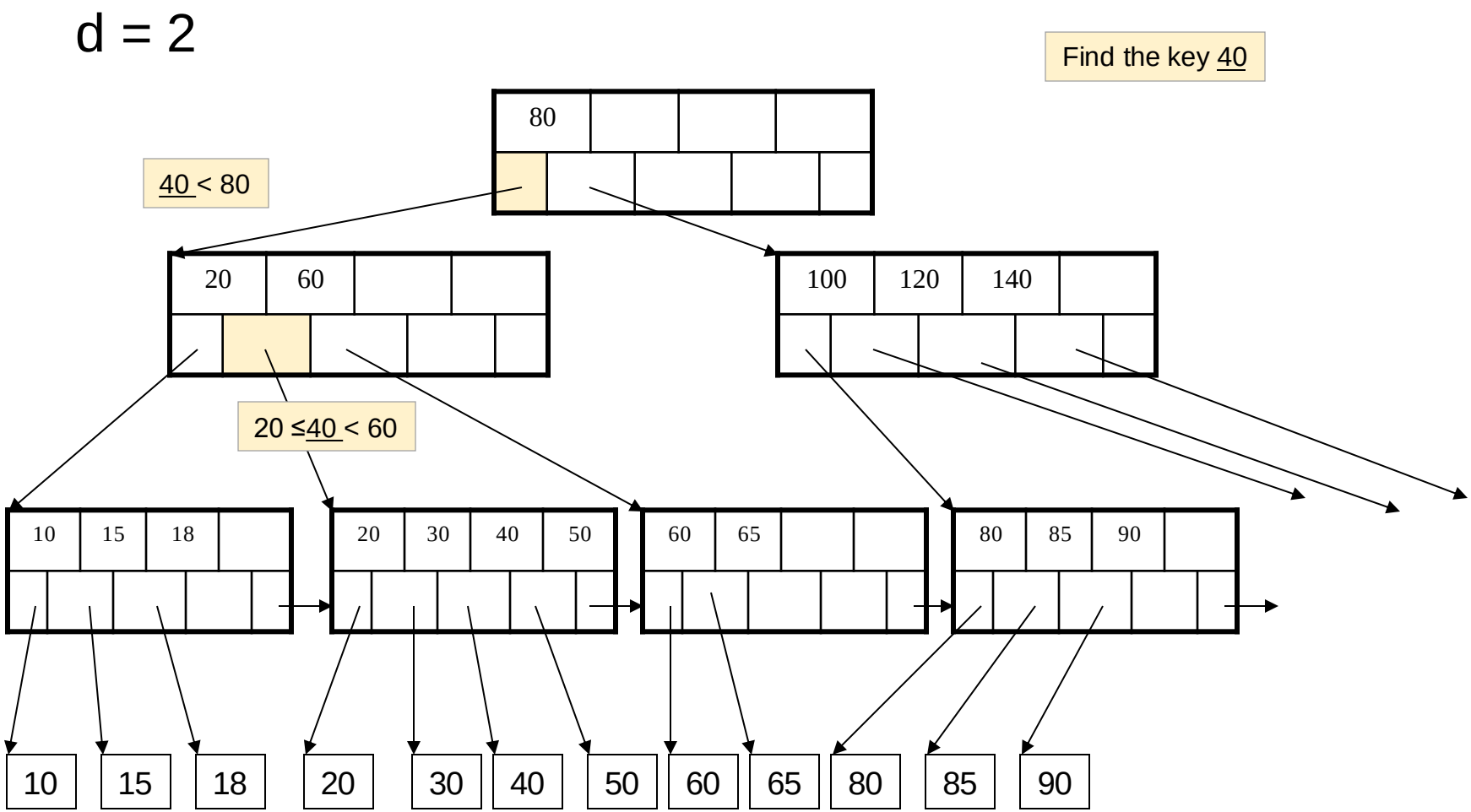
d = 2

Find the key 40

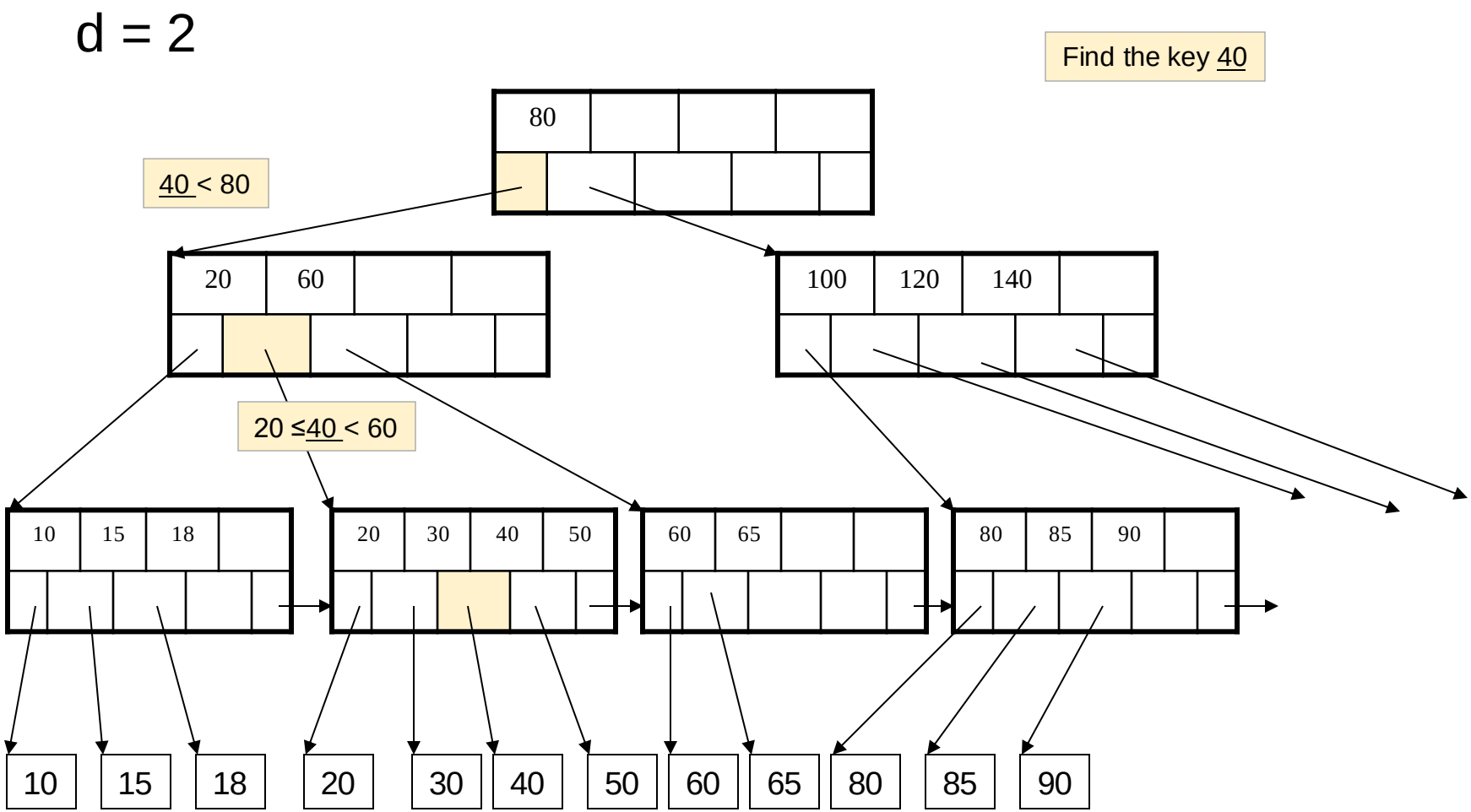
40 < 80



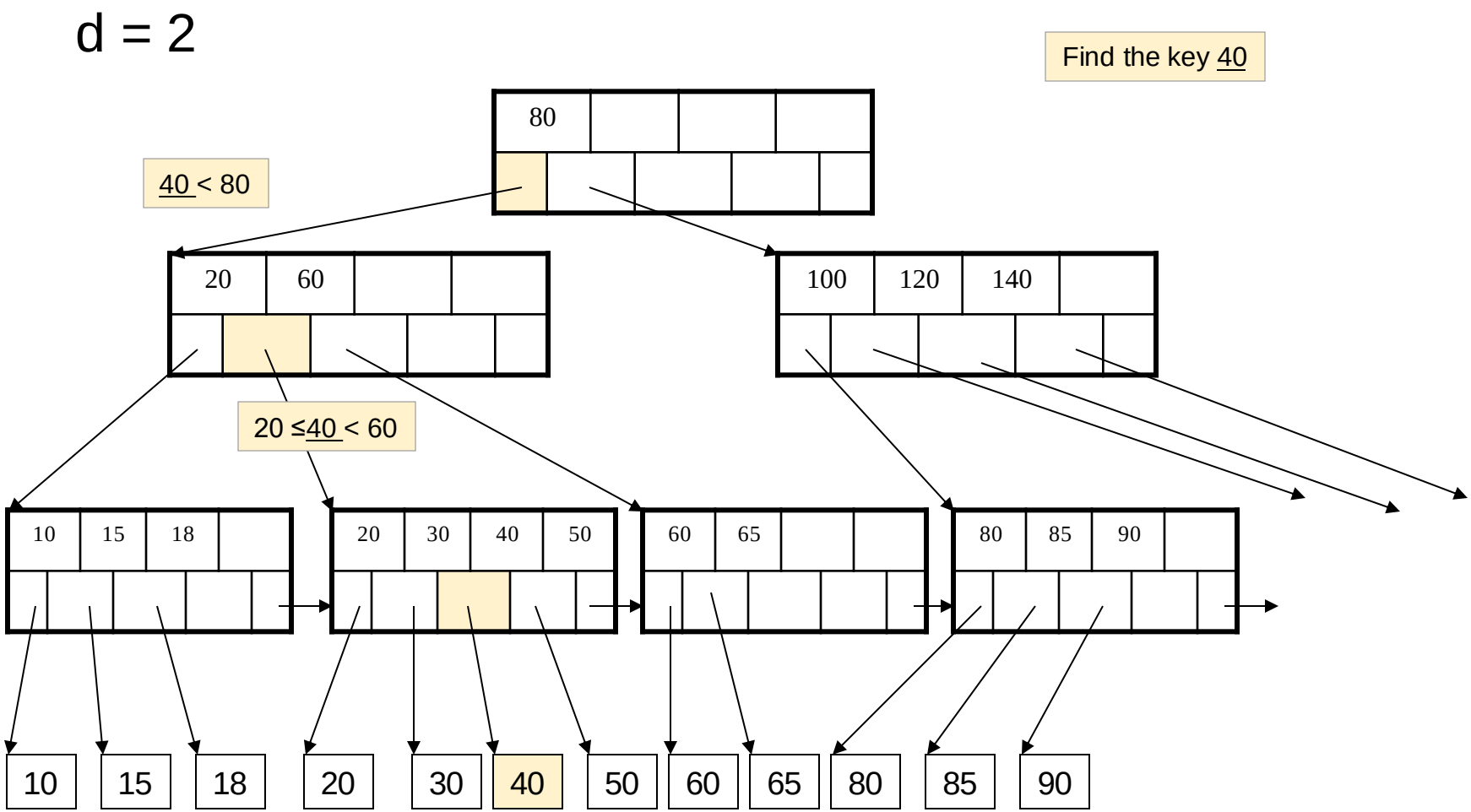
B+ Tree Example



B+ Tree Example

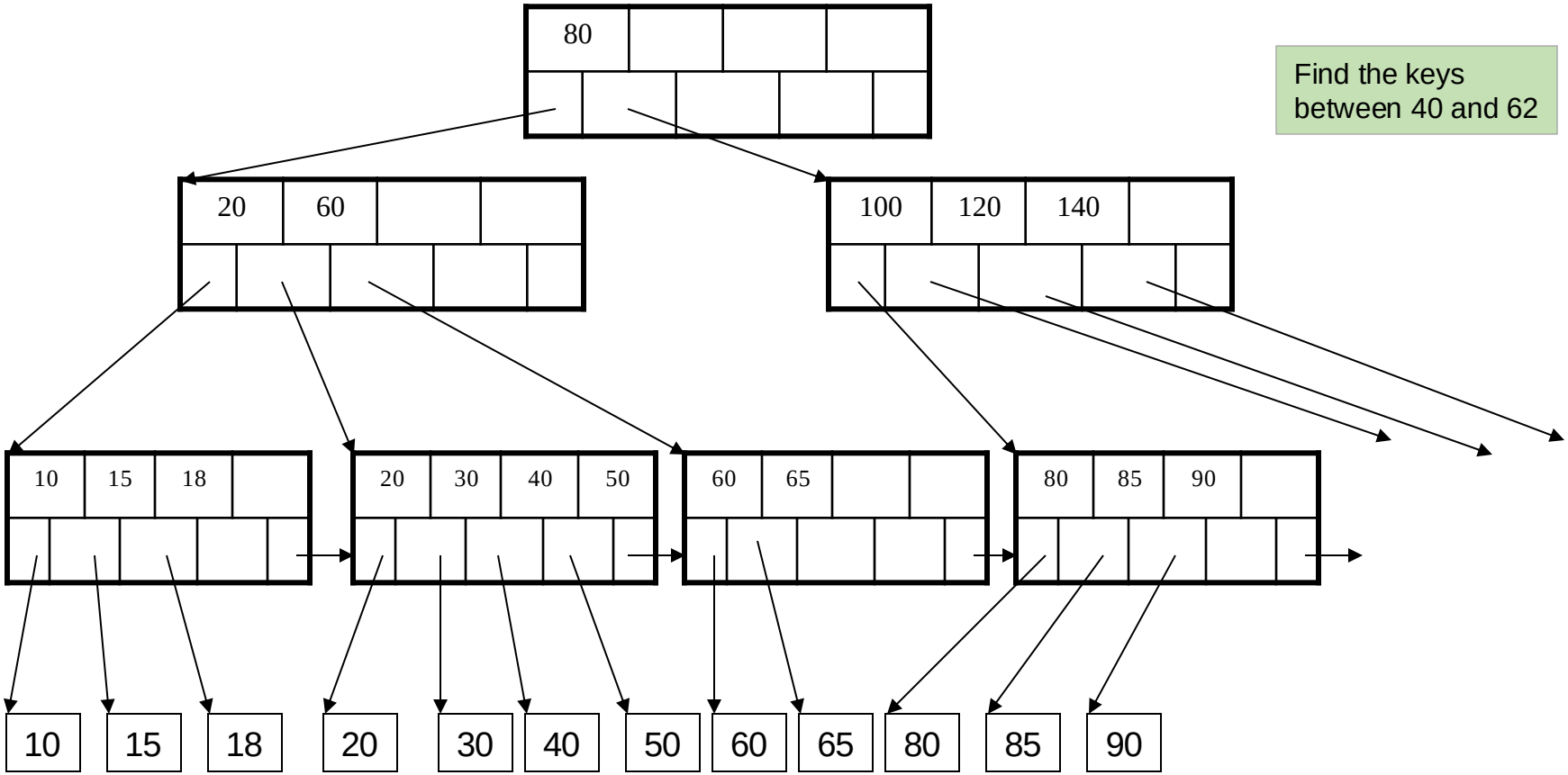


B+ Tree Example



B+ Tree Example

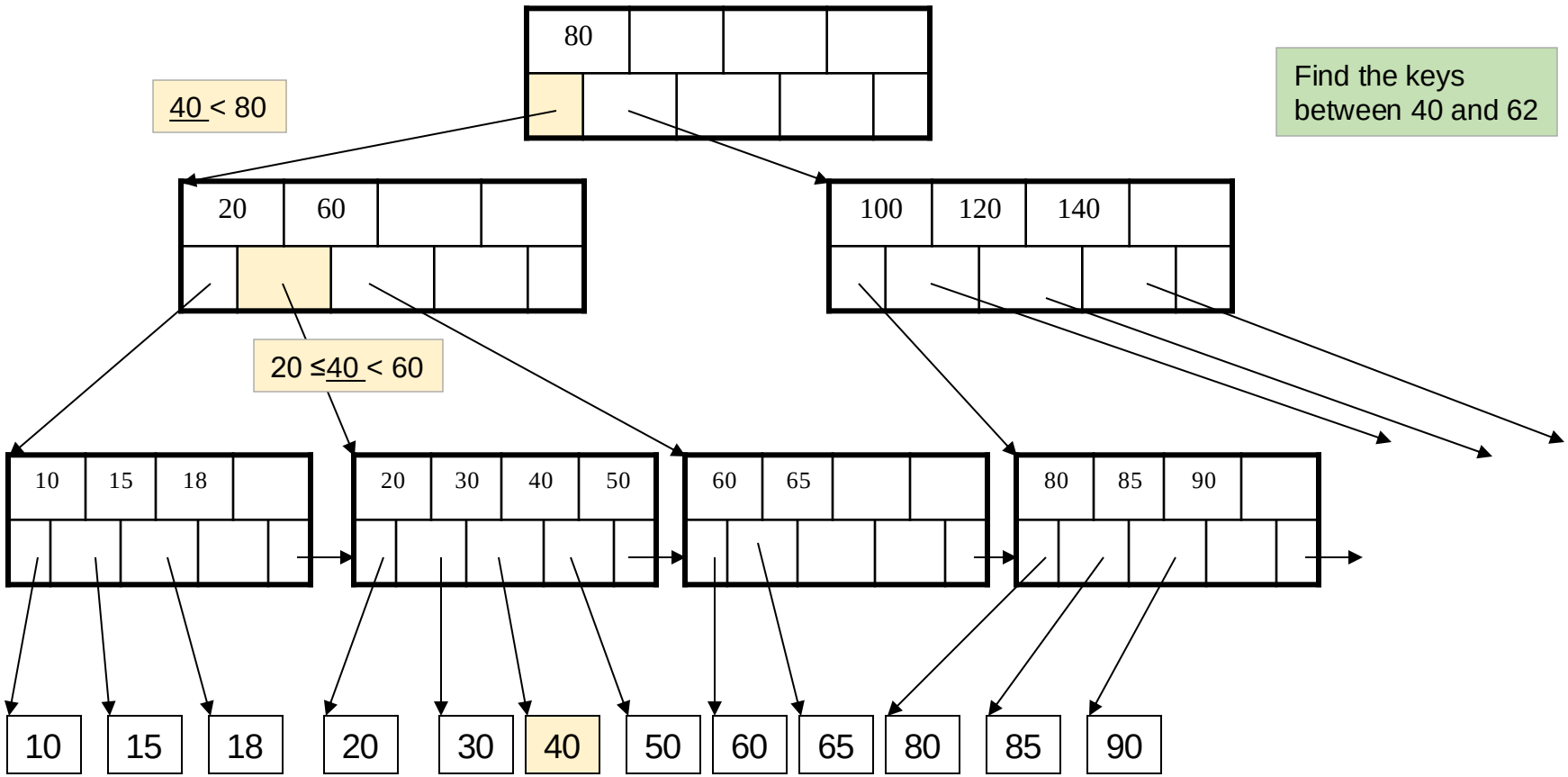
d = 2



Find the keys between 40 and 62

B+ Tree Example

d = 2



B+ Tree Example

d = 2