CS 4440 A

Midterm Review

Lecture 9

02/05/25

Midterm Logistics

- Midterm will be held Monday Feb 10th from 3:30pm - 4:45pm (during class time).
- Please arrive early - the exam is going to start at 3:30pm.
- Open notes, no electronic devices. Can bring calculator.
- Contents covered: Lec 2 (Relational Algebra) – Lec 7 (B+ Tree)

Relational Algebra

The Relational Model: Data

Student

sid	name	gpa
001	Bob	3.2
002	Joe	2.8
003	Mary	3.8
004	Alice	3.5

An **attribute** (or **column**) is a typed data entry present in each tuple in the relation

The number of tuples is the **cardinality** of the relation

A **tuple** or **row** (or *record)* is a single entry in the table having the attributes specified by the schema

A **relational instance** is a ***set*** of tuples all conforming to the same *schema*

The number of attributes is the **arity** of the relation

Relational Algebra (RA)

- Five **basic** operators:
  1. Selection: σ
  2. Projection: Π
  3. Cartesian Product: $\times$
  4. Union: $\cup$
  5. Difference: -
- Derived or auxiliary operators:
  - Intersection, complement
  - Joins (natural,equi-join, theta join, semi-join)
  - Renaming: ρ
  - Grouping: γ

RDBMSs use *multisets*, however in relational algebra formalism we will consider **sets!**

1. Selection (σ)

- Returns all tuples which satisfy a condition
- Notation: $\sigma_c(R)$
- The condition c can be =, <, >, <>

Students(sid,sname,gpa)

SQL:

```
SELECT *
FROM Students
WHERE gpa > 3.5;
```

RA:

$$\sigma_{gpa > 3.5}(Students)$$

2. Projection (Π)

- Eliminates columns, then removes duplicates
- Notation: $\Pi_{A1,\ldots,An}(R)$

Students(sid,sname,gpa)

SQL:

```
SELECT DISTINCT
  sname,
  gpa
FROM Students;
```

RA:

$$\Pi_{sname,gpa}(Students)$$

3. Cross-Product (×)

- Each tuple in R1 with each tuple in R2
- Notation: R1 × R2
- Rare in practice; mainly used to express joins

Students(sid,sname,gpa)
People(ssn,pname,address)

SQL:

```
SELECT *
FROM Students, People;
```

RA:

$$Students \times People$$

Renaming (ρ)

- Changes the schema, not the instance
- A ‘special’ operator- neither basic nor derived
- Notation: $\rho_{B1,\ldots,Bn}(R)$

- **Note: this is shorthand for the proper form (since names, not order matters!):**
  - $\rho_{A1 \rightarrow B1,\ldots,An \rightarrow Bn}(R)$

Students(sid,sname,gpa)

SQL:

```
SELECT
  sid AS studId,
  sname AS name,
  gpa AS gradePtAvg
FROM Students;
```

RA:

$$\rho_{studId,name,gradePtAvg}(Students)$$

Natural Join (⋈)

- Notation: $R_1 \bowtie R_2$
- Joins R_1 and R_2 on *equality of all shared attributes*
  - If R_1 has attribute set A, and R_2 has attribute set B, and they share attributes A⋂B = C, can also be written: $R_1 \bowtie_{C} R_2$
- Our first example of a *derived* RA operator:
  - Meaning: $R_1 \bowtie R_2 = \Pi_{A \cup B}(\sigma_{C=D}(\rho_{C \to D}(R_1) \times R_2))$
  - Where:
    - The rename $\rho_{C \to D}$ renames the shared attributes in one of the relations
    - The selection $\sigma_{C=D}$ checks equality of the shared attributes
    - The projection $\Pi_{A \cup B}$ eliminates the duplicate common attributes

Students(sid,name,gpa)
People(ssn,name,address)

SQL:

```
SELECT DISTINCT
  ssid, S.name, gpa,
  ssn, address
FROM
  Students S,
  People P
WHERE S.name = P.name;
```

RA:

$$Students \bowtie People$$

Converting SFW Query -> RA

You should also be able to convert RA -> SQL query

```
SELECT DISTINCT A1,…,An
FROM            R1,…,Rm
WHERE           c1 AND … AND ck;
```

$$\Rightarrow \quad \Pi_{A_1,\ldots,An}(\sigma_{c_1} \ldots \sigma_{c_k}(R_1 \bowtie \cdots \bowtie R_m))$$

Why must the selections “happen before” the projections?

Design Theory

Data Anomalies

Student	Course	Room
Mary	CS145	B01
Joe	CS145	B01
Sam	CS145	B01
..	..	..

Student	Course
Mary	CS145
Joe	CS145
Sam	CS145
..	..

Course	Room
CS145	B01
CS229	C12

Eliminate anomalies by **decomposing relations.**

- Redundancy
- Update anomaly
- Delete anomaly
- Insert anomaly

FDs for Relational Schema Design

High-level idea: **why do we care about FDs?**

1. Start with some relational *schema*
2. Find out its *functional dependencies (FDs)*
3. Use these to *design a better schema*
    1. One which minimizes possibility of anomalies

Finding Functional Dependencies

Equivalent to asking: Given a set of FDs, $F = \{f_1, \ldots f_n\}$, does an FD g hold?

Inference problem: How do we decide?

Three simple rules called **Armstrong's Rules.**

1. **Reflexivity,**
2. **Augmentation, and**
3. **Transitivity…**

Armstrong's axioms

- Does $AB \rightarrow D$ follow from the FDs below?

$AB \rightarrow C$

$BC \rightarrow AD$

$D \rightarrow E$

$CF \rightarrow B$

1. $AB \rightarrow C$ (given)
2. $BC \rightarrow AD$ (given)
3. $AB \rightarrow BC$ (Augmentation on 1)
4. $AB \rightarrow AD$ (Transitivity on 2,3)
5. $AD \rightarrow D$ (Reflexivity)
6. $AB \rightarrow D$ (Transitivity on 4,5)

Closure of a set of Attributes

Given a set of attributes $A_1, \ldots, A_n$ and a set of FDs **F:**
Then the <u>**closure**</u>, $\{A_1, \ldots, A_n\}^+$ is the set of attributes **B** s.t. $\{A_1, \ldots, A_n\} \rightarrow B$

<u>Example:</u> F =

{name} → {color}
{category} → {department}
{color, category} → {price}

Example Closures:

$\{name\}^+$ = {name, color}
$\{name, category\}^+$ =
{name, category, color, dept, price}
$\{color\}^+$ = {color}

Closure algorithm

Start with $X = \{A_1, \ldots, A_n\}$ and set of FDs F.

Repeat until X doesn't change; **do**:

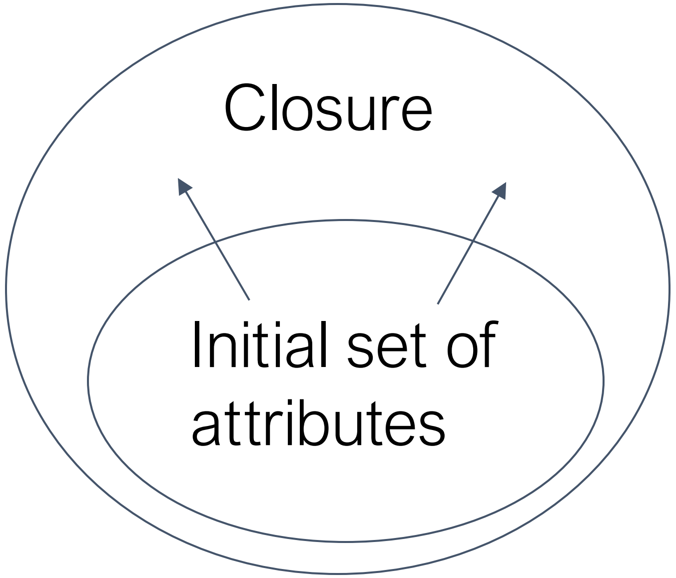
if $\{B_1, \ldots, B_n\} \rightarrow C$ is entailed by F

and $\{B_1, \ldots, B_n\} \subseteq X$

then add C to X.

Return X as X^+

Helps to split the FD's of F so each FD has a single attribute on the right



Keys and Superkeys

A **superkey** is a set of attributes $A_1, \ldots, A_n$ s.t. for *any other* attribute B in R,
we have $\{A_1, \ldots, A_n\} \rightarrow B$

I.e. all attributes are *functionally determined* by a superkey

A **key** is a *minimal* superkey

Meaning that no subset of a key is also a superkey

Computing Keys and Superkeys

- <u>Superkey?</u>
  - Compute the closure of A
  - See if it = the full set of attributes
- <u>Key?</u>
  - Confirm that A is superkey
  - Make sure that no subset of A is a superkey
    - *Only need to check one 'level' down!*

Let A be a set of attributes, R set of all attributes, F set of FDs:

```
IsSuperkey(A, R, F):
A+ = ComputeClosure(A, F)
Return (A+==R)?
```

```
IsKey(A, R, F):
If not IsSuperkey(A, R, F):
        return False
For B in SubsetsOf(A, size=len(A)-1):
        if IsSuperkey(B, R, F):
                return False
return True
```

Boyce-Codd Normal Form

BCNF is a simple condition for removing anomalies from relations:

A relation R is <u>in BCNF</u> if:

if $\{A_1, ..., A_n\} \rightarrow B$ is a *non-trivial* FD in R

then $\{A_1, ..., A_n\}$ **is a superkey** for R

Equivalently: ∀ sets of attributes X, either ($X^+ = X$) or (X^+ = all attributes)

Example

BCNFDecomp(R):

- Find an FD $X \rightarrow Y$ that violates BCNF
  (X and Y are sets of attributes)
- Compute the closure X+
- <u>let</u> $Y = X^+ - X$, $Z = (X^+)^C$
  decompose R into $R_1(X \cup Y)$ and $R_2(X \cup Z)$
- Recursively decompose R_1 and R_2

R(A,B,C,D,E)

$\{A\} \rightarrow \{B,C\}$

$\{C\} \rightarrow \{D\}$

Lossy vs. Lossless

Name	Price	Category
Gizmo	19.99	Gadget
OneClick	24.99	Camera
Gizmo	19.99	Camera

Name	Category
Gizmo	Gadget
OneClick	Camera
Gizmo	Camera

Price	Category
19.99	Gadget
24.99	Camera
19.99	Camera

Name	Price	Category
Gizmo	19.99	Gadget
OneClick	19.99	Camera
OneClick	24.99	Camera
Gizmo	19.99	Camera
Gizmo	24.99	Camera

Lossy vs. Lossless

Name	Price	Category
Gizmo	19.99	Gadget
OneClick	24.99	Camera
Gizmo	19.99	Recorder

{Category} → {Name}

Name	Category
Gizmo	Gadget
OneClick	Camera
Gizmo	Recorder

Price	Category
19.99	Gadget
24.99	Camera
19.99	Recorder

Name	Price	Category
Gizmo	19.99	Gadget
OneClick	24.99	Camera
Gizmo	19.99	Recorder

A Problem with BCNF

Unit	Company	Product
…	…	…

$\{Unit\} \rightarrow \{Company\}$
$\{Company, Product\} \rightarrow \{Unit\}$

<u>Unit</u>	Company
…	…

$\{Unit\} \rightarrow \{Company\}$

Unit	Product
…	…

We do a BCNF decomposition on a "bad" FD:
$\{Unit\}^+ = \{Unit, Company\}$

We lose the FD $\{Company, Product\} \rightarrow \{Unit\}$!!

The problem with BCNF

- We started with a table R and FDs F
- We decomposed R into BCNF tables R_1, R_2, …
with their own FDs F_1, F_2, …
- We insert some tuples into each of the relations—which satisfy their local FDs but when reconstruct it violates some FD **across** tables!

<u>Practical Problem</u>: To enforce FD, must reconstruct R—on each insert!

Third normal form (3NF)

A relation R is <u>in 3NF</u> if:

For every non-trivial FD $A_1, \ldots, A_n \rightarrow B$, either

- $\{A_1, \ldots, A_n\}$ is a superkey for R
- B is a prime attribute (i.e., B is part of some candidate key of R)

Example:

- The keys are AB and AC
- $B \rightarrow C$ is a BCNF violation, but not a 3NF violation because C is prime (part of the key AC)

R(A,B,C)

$AC \rightarrow B$
$B \rightarrow C$

Minimal basis generation

Given a set of FD's F, any set of FD's equivalent to F is a **basis** for F

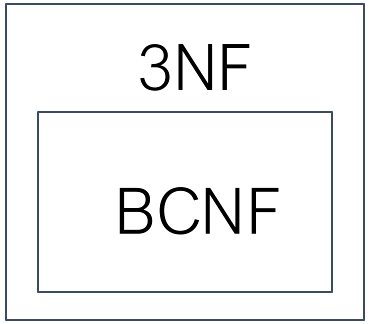
Input: $F = \{A \rightarrow AB, AB \rightarrow C\}$

1. Split FD's so that they have singleton right sides
   $G = \{A \rightarrow B, A \rightarrow A, AB \rightarrow C\}$
2. Remove trivial FDs
   $G = \{A \rightarrow B, AB \rightarrow C\}$
3. Minimize the left sides of each FD
   $G = \{A \rightarrow B, A \rightarrow C\}$
4. Remove redundant FDs
   $G = \{A \rightarrow B, A \rightarrow C\}$

Step 3:

For each FD $X \rightarrow A$ in F:
For each attribute B in X:
If $(X - \{B\})+$ contains A,
remove B from X.

BCNF vs 3NF



- Given a non-trivial FD X → B (X is a set of attributes)
  - BCNF: X must be a superkey
  - 3NF: X must be a superkey or B is prime

- Both BCNF and 3NF give lossless joins
- 'Use 3NF over BCNF if you need dependency preservation
- However, 3NF may not remove all redundancies and anomalies

MVD Example

Movie_ Star (A)	Address (B)	Movie (C)
Leonardo DiCaprio	Los Angeles	Titanic
Leonardo DiCaprio	Los Angeles	Inception
Leonardo DiCaprio	New York	Titanic
Leonardo DiCaprio	New York	Inception
Scarlett Johansson	Los Angeles	Black Widow
Scarlett Johansson	Los Angeles	Her
Scarlett Johansson	Paris	Black Widow
Scarlett Johansson	Paris	Her

- **Independence**: The set of addresses is independent of the set of movies for a given movie star.
- **Redundancy**: Notice how each movie is repeated for every address that the star lives in, and vice versa.

MVD Example

	Movie_ Star (A)	Address (B)	Movie (C)
t_1	Leonardo DiCaprio	Los Angeles	Titanic
t_3	Leonardo DiCaprio	Los Angeles	Inception
	Leonardo DiCaprio	New York	Titanic
t_2	Leonardo DiCaprio	New York	Inception
	Scarlett Johansson	Los Angeles	Black Widow
	Scarlett Johansson	Los Angeles	Her
	Scarlett Johansson	Paris	Black Widow
	Scarlett Johansson	Paris	Her

We write $\mathbf{A} \twoheadrightarrow \mathbf{B}$ if for any tuples t_1, t_2 s.t. $t_1[A] = t_2[A]$ there is a tuple t_3 s.t.

- $t_3[A] = t_1[A]$
- $t_3[B] = t_1[B]$
- and $t_3[R\backslash B] = t_2[R\backslash B]$

Where R\B is "R minus B" i.e. the attributes of R not in B

Multi-Value Dependencies (MVDs)

One less formal, literal way to phrase the definition of an MVD:

The MVD $\boldsymbol{X} \twoheadrightarrow \boldsymbol{Y}$ holds on R if for any pair of tuples with the same X values, the tuples with the same X values, but the other permutations of Y and A\Y values, is also in R

Ex: X = {x}, Y = {y}:

x	y	z
1	0	1
1	1	0

For $\boldsymbol{X} \twoheadrightarrow \boldsymbol{Y}$ to hold must have…

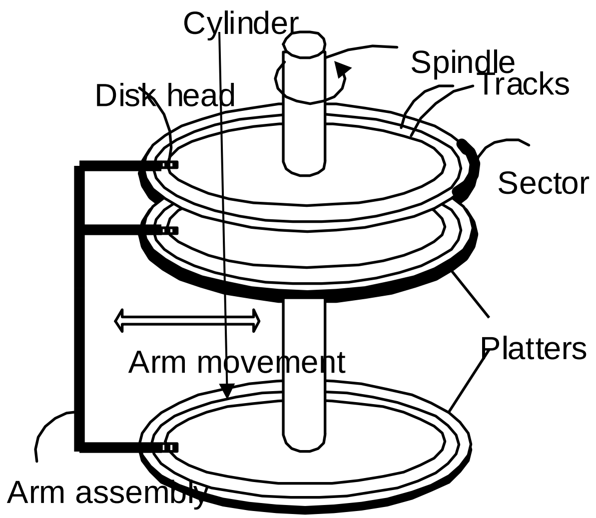
x	y	z
1	0	1
1	1	0
1	0	0
1	1	1

Practice Question

- Data Model and Design Theory (15 points)

Storage

High-level: Disk vs. Main Memory



Disk:

- ***Fast:*** sequential block access
  - Read a blocks (not byte) at a time, so sequential access is cheaper than random
  - Disk read / writes are expensive
- ***Durable:*** We will assume that once on disk, data is safe!
- ***Cheap***

Random Access Memory (RAM) or Main Memory:

- ***Fast:*** Random access, byte addressable
  - ~10x faster for sequential access
  - ~100,000x faster for random access!
- ***Volatile:*** Data can be lost if e.g. crash occurs, power goes out, etc!
- ***Expensive:*** For $100, get 16GB of RAM vs. 2TB of disk!

Disk access time

Latency = seek time + rotational delay + transfer time + other

- Transfer time: time to read/write data in sectors

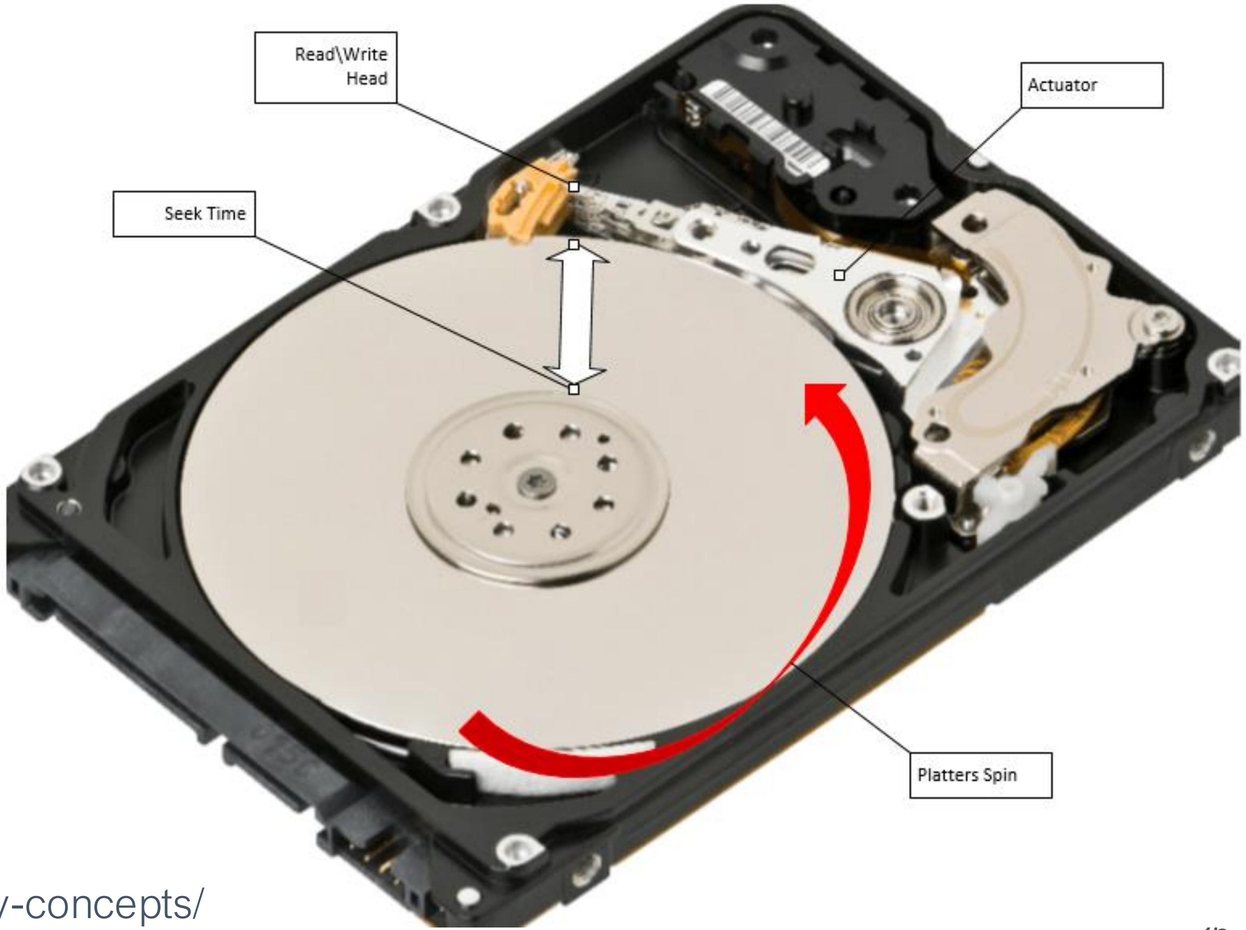
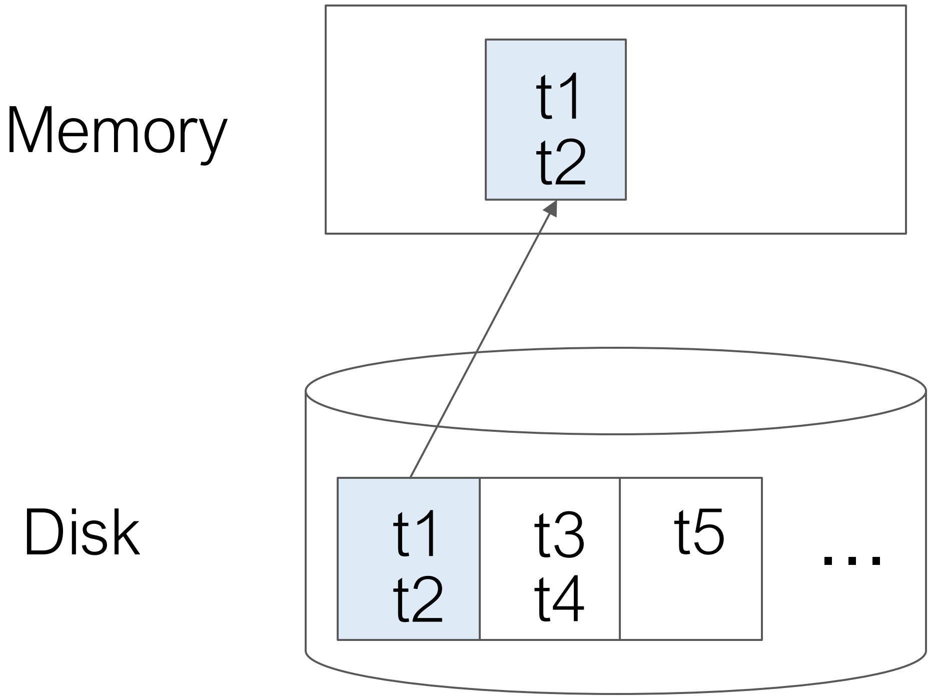


Image source: https://theithollow.com/2013/11/18/disk-latency-concepts/

I/O model of computation

- Time to read a block from disk >> time to search a record within that block
- Algorithm time ≈ Number of disk I/Os



Storing Records

Record (tuple): consecutive bytes in disk blocks

- e.g. employee record:
  - name field
  - salary field
  - date-of-hire field

Design choices:

- Fixed vs variable **length**
- Fixed vs variable **format**

Data items
↓
Records
Blocks
Files

Adapted from Stanford CS245 from Matei Zaharia

Place Data for Efficient Access

Locality: which items are accessed together

- When you read one field of a record, you're likely to read other fields of the same record
- When you read one field of record 1, you're likely to read the same field of record 2

Searchability: quickly find relevant records

- E.g. sorting the file lets you do binary search

Adapted from Stanford CS245 from Matei Zaharia

Locality Example: Row Stores vs Column Stores

Row Store

name	age	state
Alex	20	CA
Bob	30	CA
Carol	42	NY
David	21	MA
Eve	26	CA
Frances	56	NY
Gia	19	MA
Harold	28	AK
Ivan	41	CA

Fields stored contiguously in one file

Column Store

name
Alex
Bob
Carol
David
Eve
Frances
Gia
Harold
Ivan

age
20
30
42
21
26
56
19
28
41

state
CA
CA
NY
MA
CA
NY
MA
AK
CA

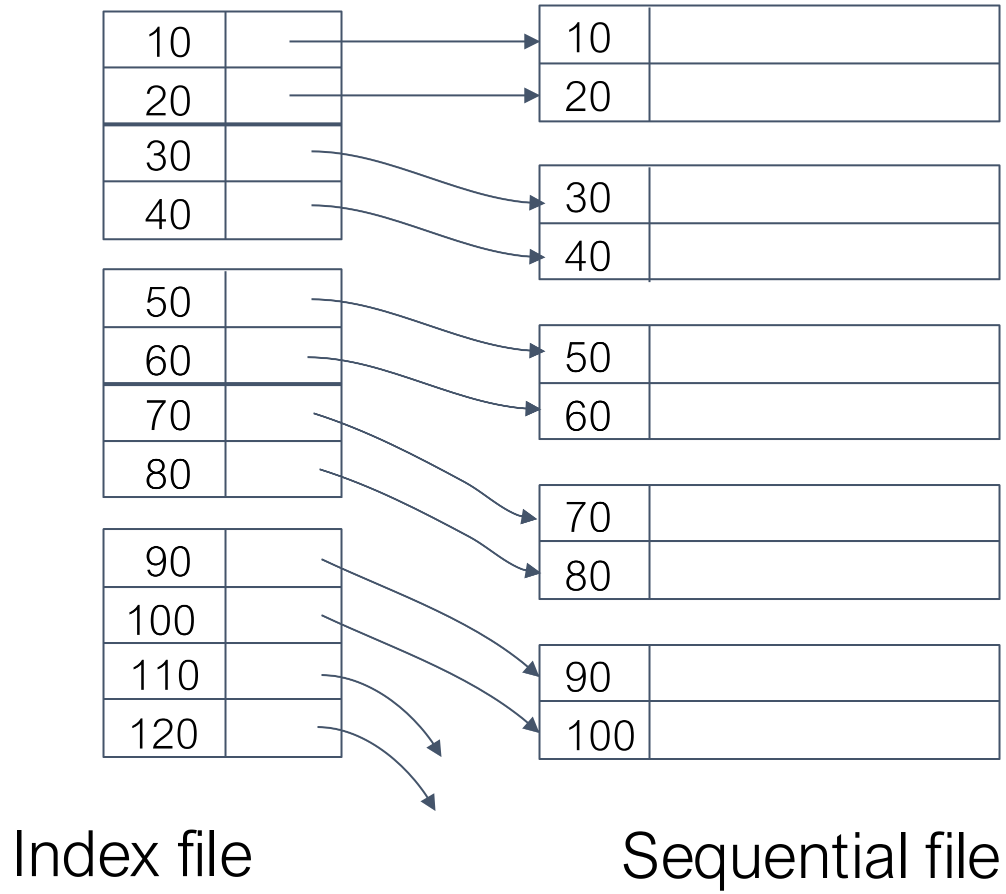
Each column in a different file

Adapted from Stanford CS245 from Matei Zaharia

Index Basics

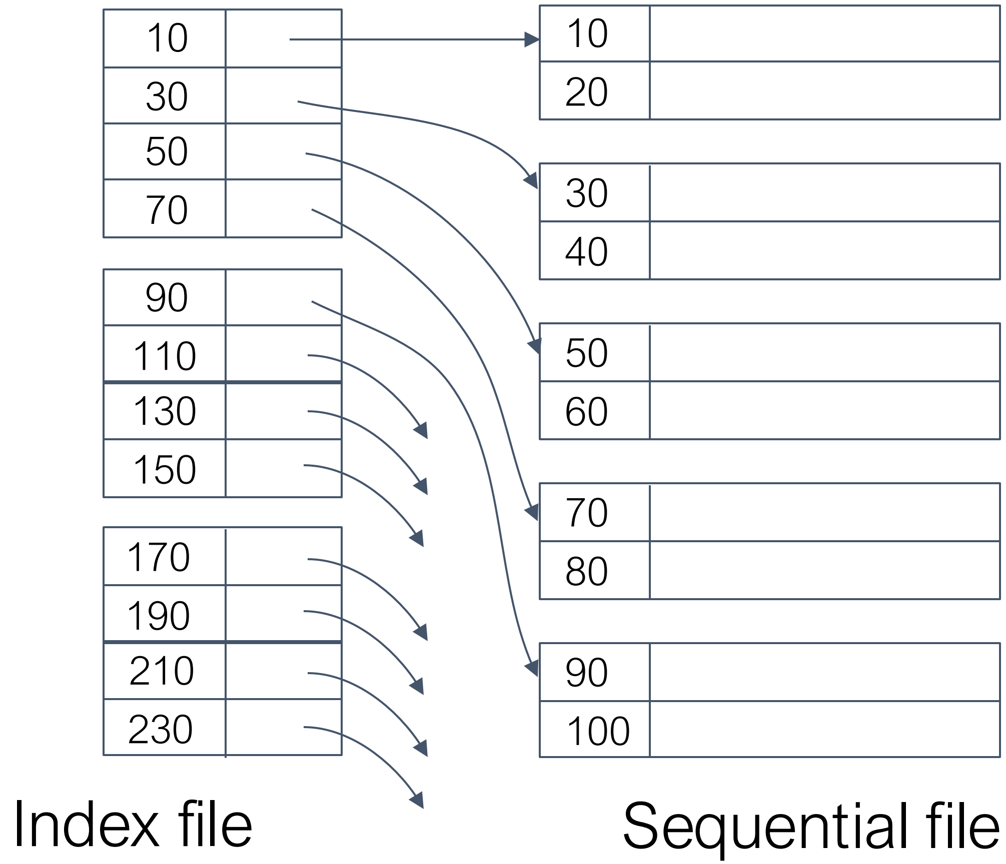
Dense index

- A sequence of blocks holding keys of records and pointers to the records



Sparse index

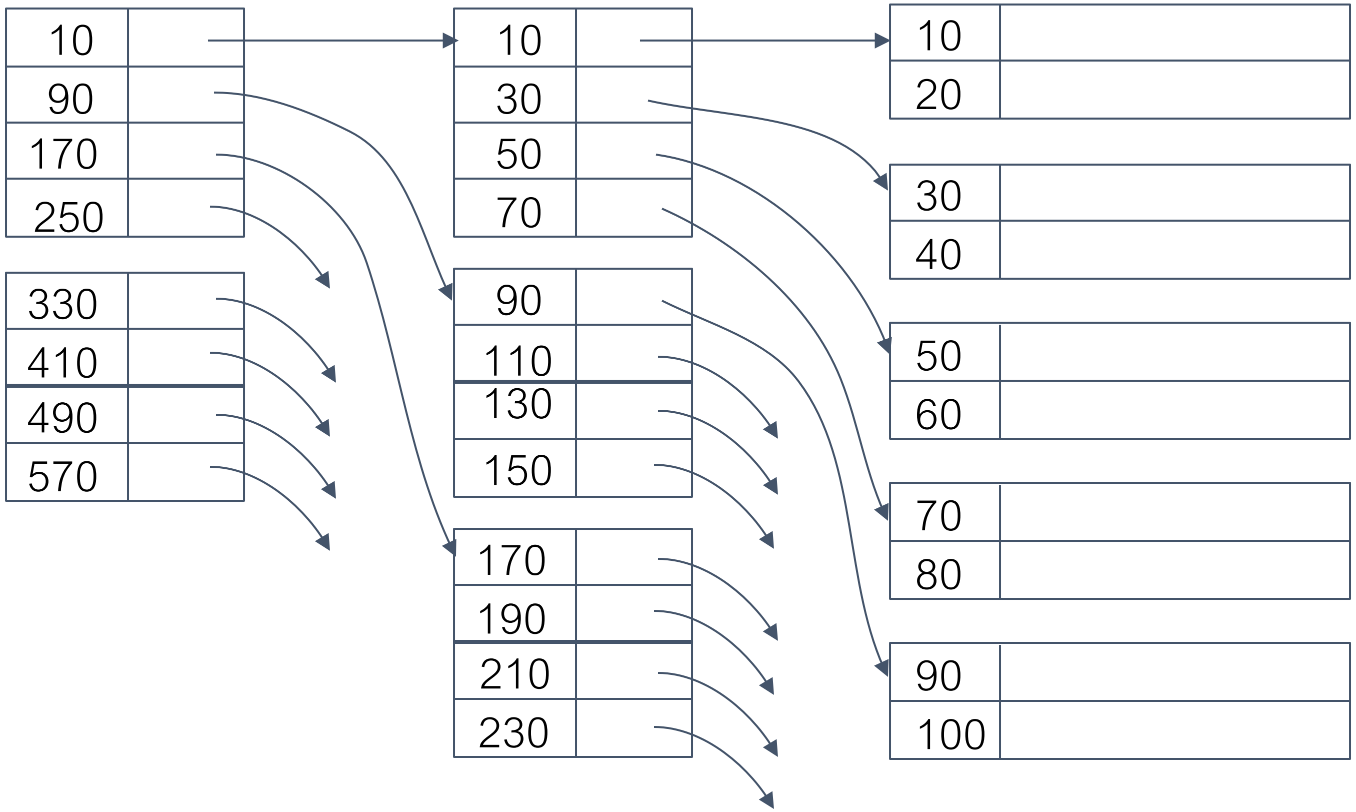
- Has one key-pointer pair per block of the data file
- Uses less space than dense index, but needs more time to find a record



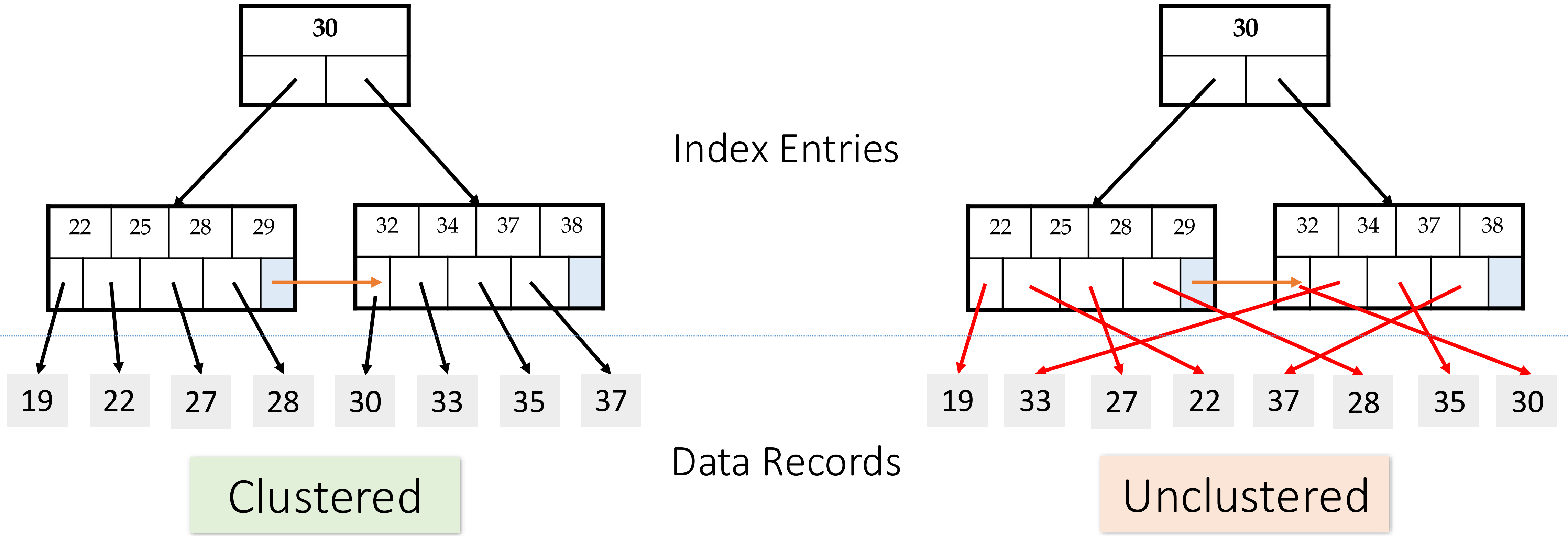
Multiple levels of index

If the index file is still large, add another level of indexing

- Basic idea of the B+-tree index (next lecture)



Clustered vs. Unclustered Index



1 Random Access IO + Sequential IO
(# of pages of answers)

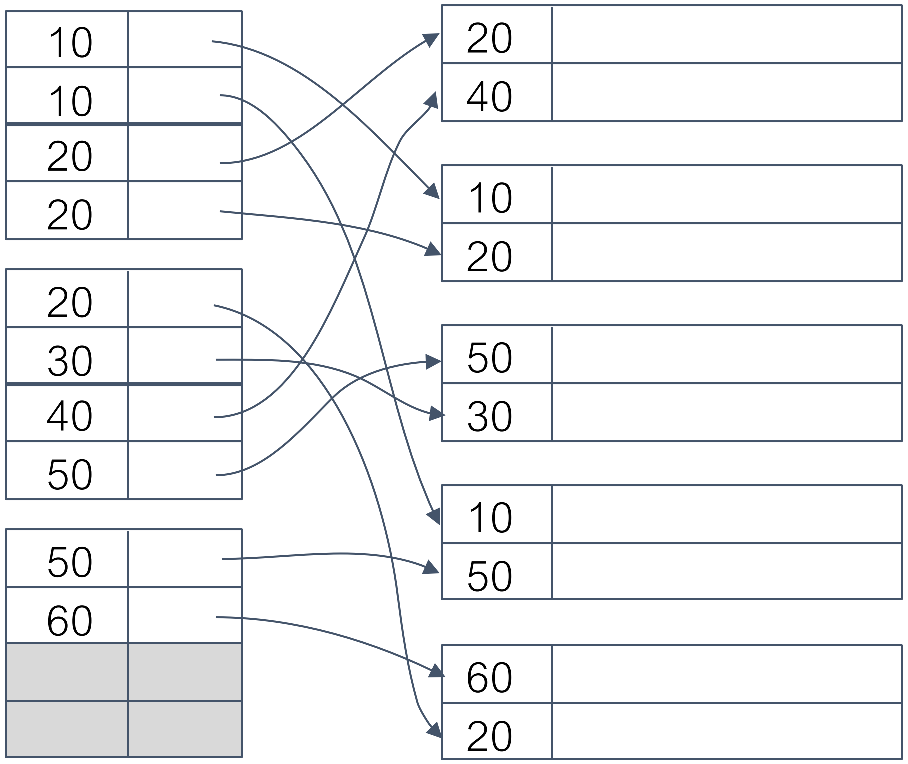
Random Access IO for each **value**
(i.e. # of tuples in answer)

Clustered can make a ***huge*** difference for range queries!

Non-clustered/Secondary index

Unlike a clustered index, does not determine the placement of records

As a result, secondary indexes are always dense



Practice Question

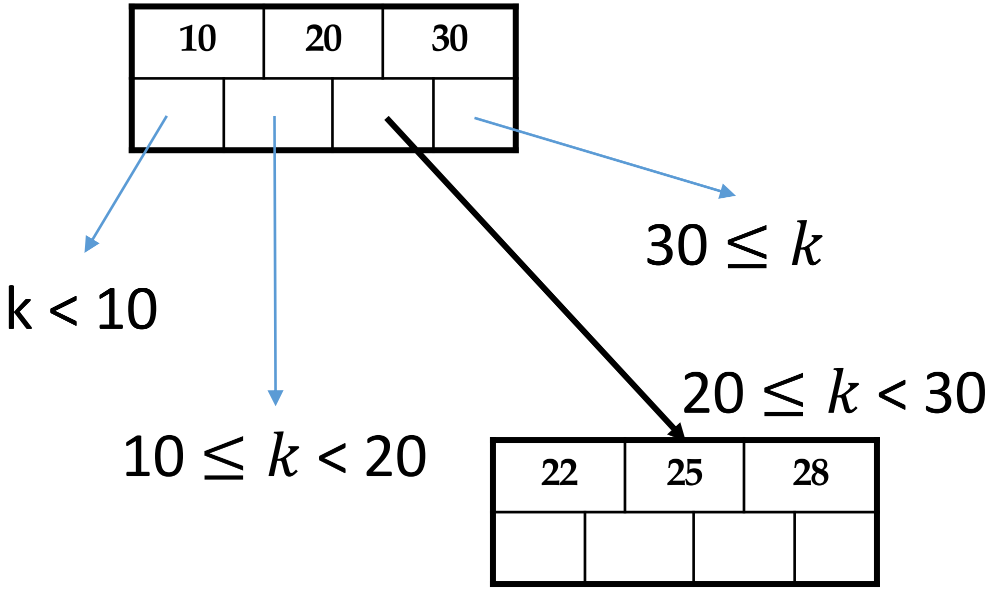
- Storage and Indexing (10 points)

B+ Tree

B+ Tree Basics

Parameter $\boldsymbol{n}$ = the degree

Non-leaf or *internal* node



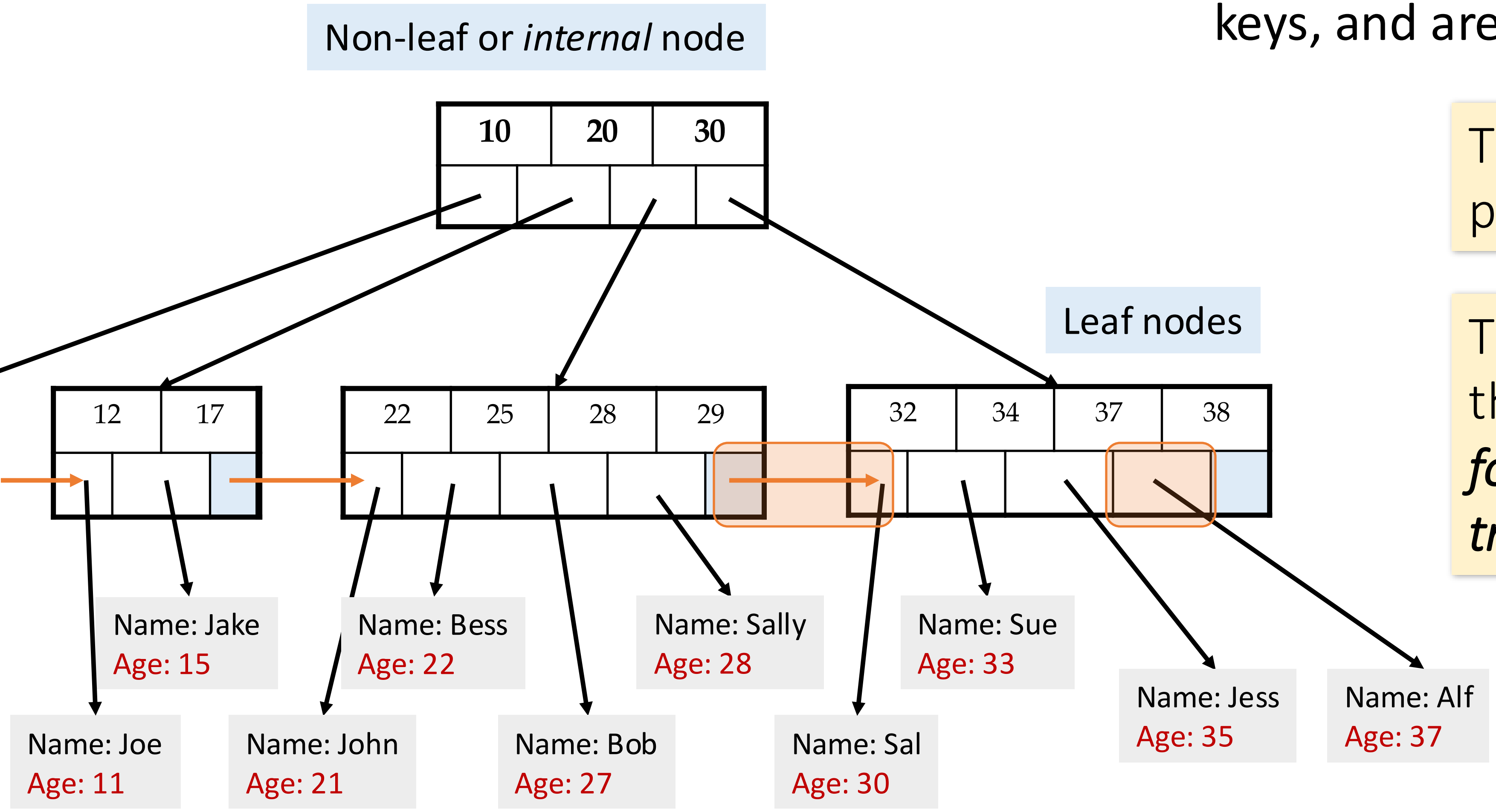
Each *non-leaf ("interior")* ***node*** has $\geq \frac{n}{2}$ and $\leq$ n ***keys****

The *k* keys in a node define *k+1* ranges

except for root node, which can have between **1 and n keys*

For each range, in a *non-leaf* node, there is a **pointer** to another node with keys in that range

B+ Tree Basics

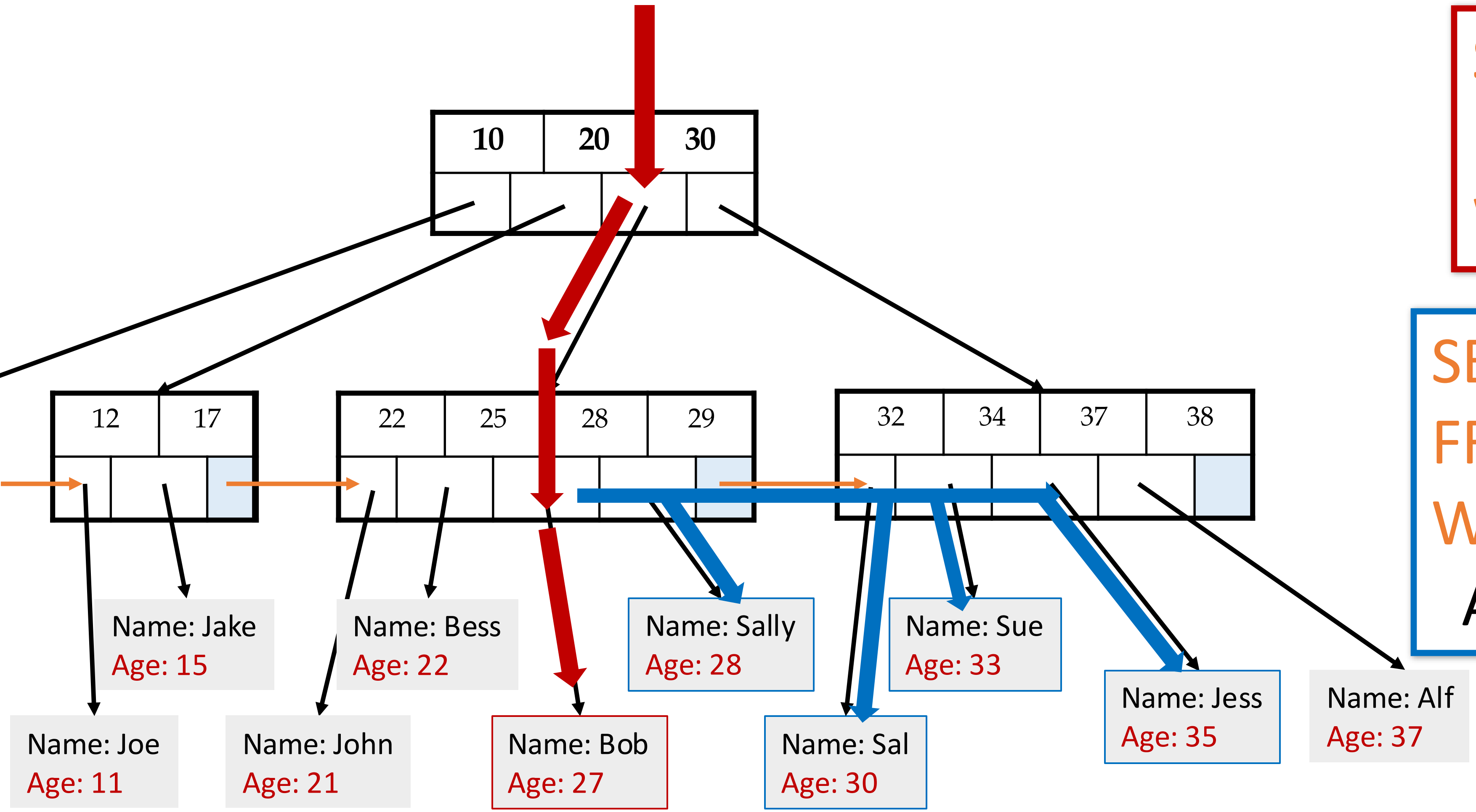


Leaf nodes also have between $\frac{n}{2}$ and n keys, and are different in that:

Their key slots contain pointers to data records

They contain a pointer to the next leaf node as well, ***for faster sequential traversal***

Searching a B+ Tree




```
SELECT name
FROM   people
WHERE  age = 27
```

```
SELECT name
FROM   people
WHERE  27 <= age
  AND  age <= 35
```

B+ Tree Cost Model

Note that exact search is just a special case of range search (R = 1)

Goal: Get the results set of a range (or exact) query with minimal IO

Key idea:

- A B+ Tree has high ***fanout (d ~= 10^2-10^3)***, which means it is very shallow → we can get to the right root node within a few steps!
- Then just traverse the leaf nodes using the horizontal pointers

Details:

- One node per page (thus page size determines *d*)
- Fill only some of each node's slots (the ***fill-factor***) to leave room for insertions
- We can keep some levels of the B+ Tree in memory!

The fanout f is the number of pointers coming out of a node. Thus:

$$d + 1 \leq f \leq 2d + 1$$

Note that we will often approximate f as constant across nodes!

We define the ***height*** of the tree as counting the root node. Thus, *given constant fanout f*, a tree of height *h* can index f^h pages and has f^{h-1} leaf nodes

B+ Tree Cost Model

Given:	• Fill-factor ***F*** • ***B*** available pages in buffer • A B+ Tree over ***N*** pages • **f** is the average fanout	
Input:	A a range query.	
Output:	The ***R*** values that match	
IO COST:	$\left\lceil \log_f \frac{N}{F} \right\rceil - L_B + \mathbf{Cost}(Out)$ *where* $B \geq \sum_{l=0}^{L_B - 1} f^l$	***Depth of the B+ Tree:*** For each level of the B+ Tree we read in one node = one page ***# of levels we can fit in memory:*** These don't cost any IO! ***This equation*** is just saying that the sum of all the nodes for L_B levels must fit in buffer