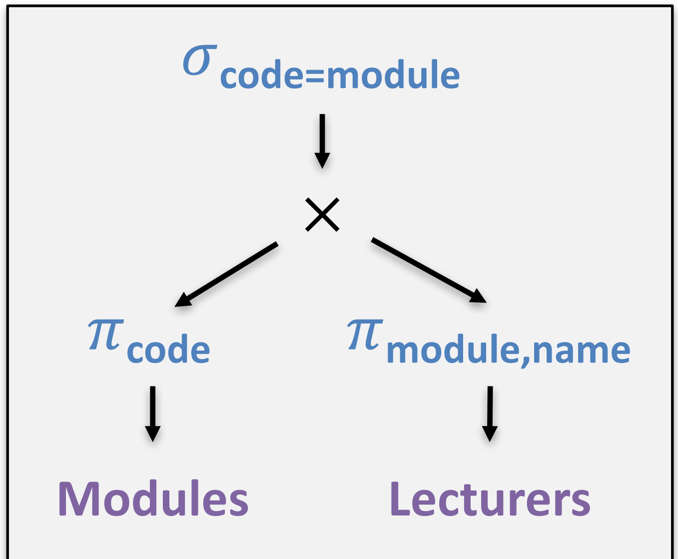
COMP207
Database Development

Lecture 14

Query Processing:
Faster With Indexes

Reminder: Query Plans

- "Recipe" for computing the result to a query



- Executed bottom-up
- Typically several algorithms for computing operators

Reminder: Computing $\sigma_{condition}(R)$

- Basic procedure:

R:

tuple 1
tuple 2
tuple 3
tuple 4
…

for each tuple t in R:
    if t satisfies condition:
        output t

Is there a faster way?

- Needs to read the entire relation
- Yes, sometimes!

Example

$\sigma_{programme='G401'}(Students)$

Students

id	name	programme
…	…	….
1234	Anna	G401
2345	Ben	G701
3456	Chloe	G401
4567	Dave	G401
…	…	…

- Selection can be performed faster if we know **where to find the rows for 'G401'**
- Two solutions: **sorting** & **index**

External sorting
(not in exam)

Merge Sort

- (Internal memory) merge sort:

Divide input in two parts P_1, P_2

Sort P_1 & P_2 **recursively**

While P_1 or P_2 is not empty:

  Add smallest remaining element from P_1 or P_2 to output

Merge

External Merge Sort

- External merge sort:

Number of disk blocks that fit in RAM

Divide input in **M** parts $P_1, P_2, \ldots, P_M$

Sort $P_1, P_2, \ldots, P_M$ **recursively**

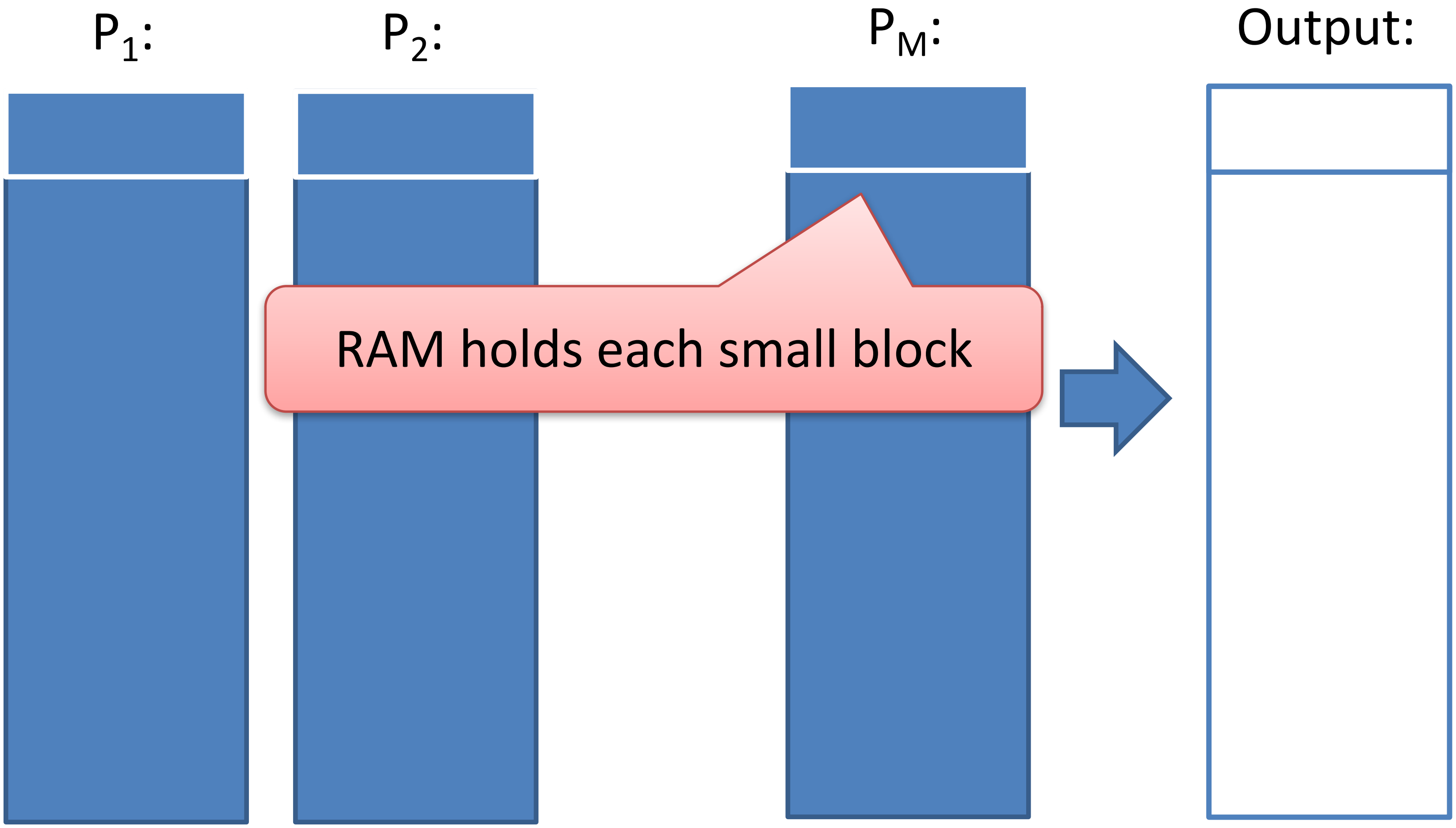
While not all P_i are empty:

  Add smallest remaining element from any part P_i to output

Merge

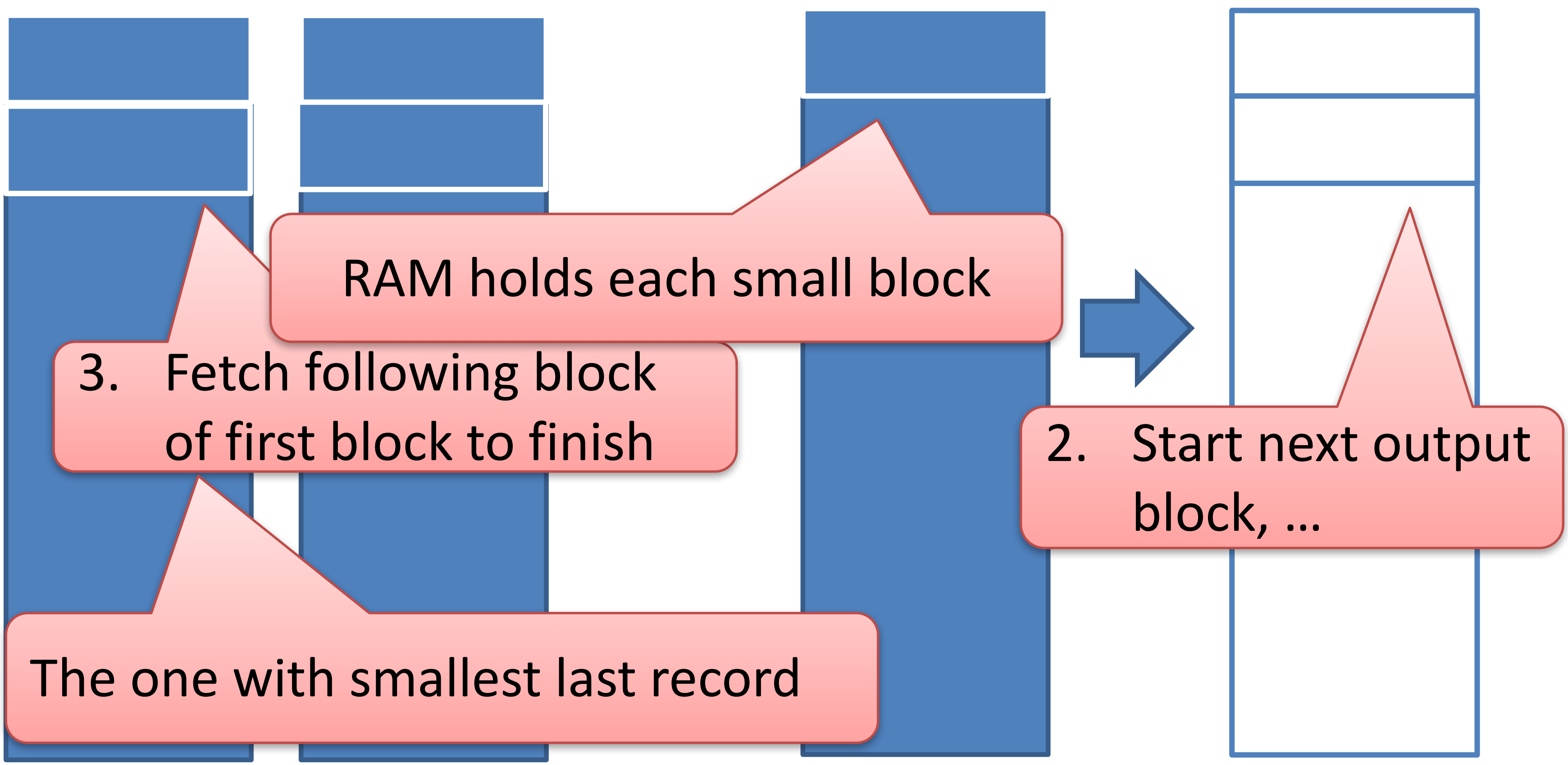
Merge in more details

- Goal: Must merge M (perhaps large) parts into one



Merge in more details

- Goal: Must merge M (perhaps larg



ge Sort

- E

We split in M buckets in each level of recursion until reminder is below MB (where it fits in RAM and can be sorted directly): $\log_M(N/(MB))$ levels

Divide inp... in **M** parts $P_1, P_2, \ldots, P_M$
Sort $P_1, P_2, \ldots, P_M$ **recursively**
... not all P_i are empty:
Add smallest remaining element from any part P_i to output

Total disk operations: $O(N/B \log_M(N/(MB)))$

On each level of recursion we scan through all blocks once = $O(N/B)$ disk operations, where B is block size

Comparison

- In practice, since harddisks are slow:
  - Running time is basically time spend on disk operations
  - Thus, time is $O(N/B \log_M(N/(MB)) \times D)$
  - D is time for a disk operation
- Quicksort (or other internal memory sorting algorithms) uses $O(N/B \log_2(N) \times D)$ time!

Numeric example

- Want to sort N=4 TB in 4 MB of RAM with B=4096

External Merge Sort:

  - $M = 4\ \text{MB}/4096 = 1024$
  - $\log_M(N/(M\,B))$ is then $\approx \log_{1024}(1024^2) = 2$

- We are thus using 2 recursive calls, for 3 scans in total

(Internal) Quick Sort:

  - $\log_2(N)=42$

Assuming we are lucky about pivot elements

- We are thus using 42 recursive calls, for 43 scans in total

Example

$\sigma_{programme='G401'}(\text{Students})$

Students

id	name	programme
…	…	….
1234	Anna	G401
2345	Ben	G701
3456	Chloe	G401
4567	Dave	G401
…	…	…

- Selection can be performed faster if we know **where to find the rows for ‘G401’**
- Two solutions: **sorting** & **index**

Indexes

(Basics: COMP102/CSE103)

Index

- Takes the values for one or more attributes of a relation **R**, provides quick access to the tuples with these values

Index on attribute programme

sorted

Value	Pointers to rows
…	…
G401	1234, 3456, 4567, …
G701	2345, …
…	…

Students

id	name	programme
…	…	….
1234	Anna	G401
2345	Ben	G701
3456	Chloe	G401
4567	Dave	G401
…	…	…

- Types:
  - Secondary: merely points to location of records on disk
  - Primary: in addition, defines how data is sorted on disk

Always dense

Good when attributes involve primary key

Example Revisited

$$\sigma_{\text{programme='G401'}}(\textbf{Students})$$

Index on attribute **programme**

Value	Pointers to rows
…	…
G401	1234, 3456, 4567, …
G701	2345
…	…

Students

id	name	programme
…	…	….
1234	Anna	G401
2345	Ben	G701
3456	Chloe	G401
4567	Dave	G401
…	…	…

- Selection with index:
  - Find entry for '**G401**' in index
  - Visit all rows in **Students** whose ids occur in the index entry for '**G401**'

Running time?

Example 2

$$\sigma_{\text{programme='G401' AND year=2}}(\text{Students})$$

Index on **programme**

Value	Pointers to rows
…	…
G401	1234, 3456, 4567, …
G701	2345
…	…

Students

id	name	programme	year
…	…	…	…
1234	Anna	G401	2
2345	Ben	G701	2
3456	Chloe	G401	3
4567	Dave	G401	1
…	…	…	

Index on **year**

Value	Pointers
…	…
1	4567, …
2	1234, 2345, …
3	3456, …
…	…

- Selection with two indexes:
  - Find entries for '**G401**' & **2** in indexes for **programme** & **year**
  - Visit all rows in **Students** whose ids occur in both index entries

Exercise

Students(id, name, programme, year)

Modules(module_code, module_title, programme, year)

Which indexes could be useful to answer these queries:

- $\pi_{\text{id,name}}(\sigma_{\text{year=1}}(\text{Students}))$
- Students ⋈ Modules

Forms of Indexes

- **B+ Trees**
  - Good if selection condition specifies a range
  - E.g., $\sigma_{\text{programme='G401' AND year > 1}}$
  - Most widely used

default

```
CREATE INDEX
  ON Students USING btree
  (programme, year);
```

- **Hash tables**
  - Good if selection involves equality only
  - E.g., $\sigma_{\text{programme='G401'}}$

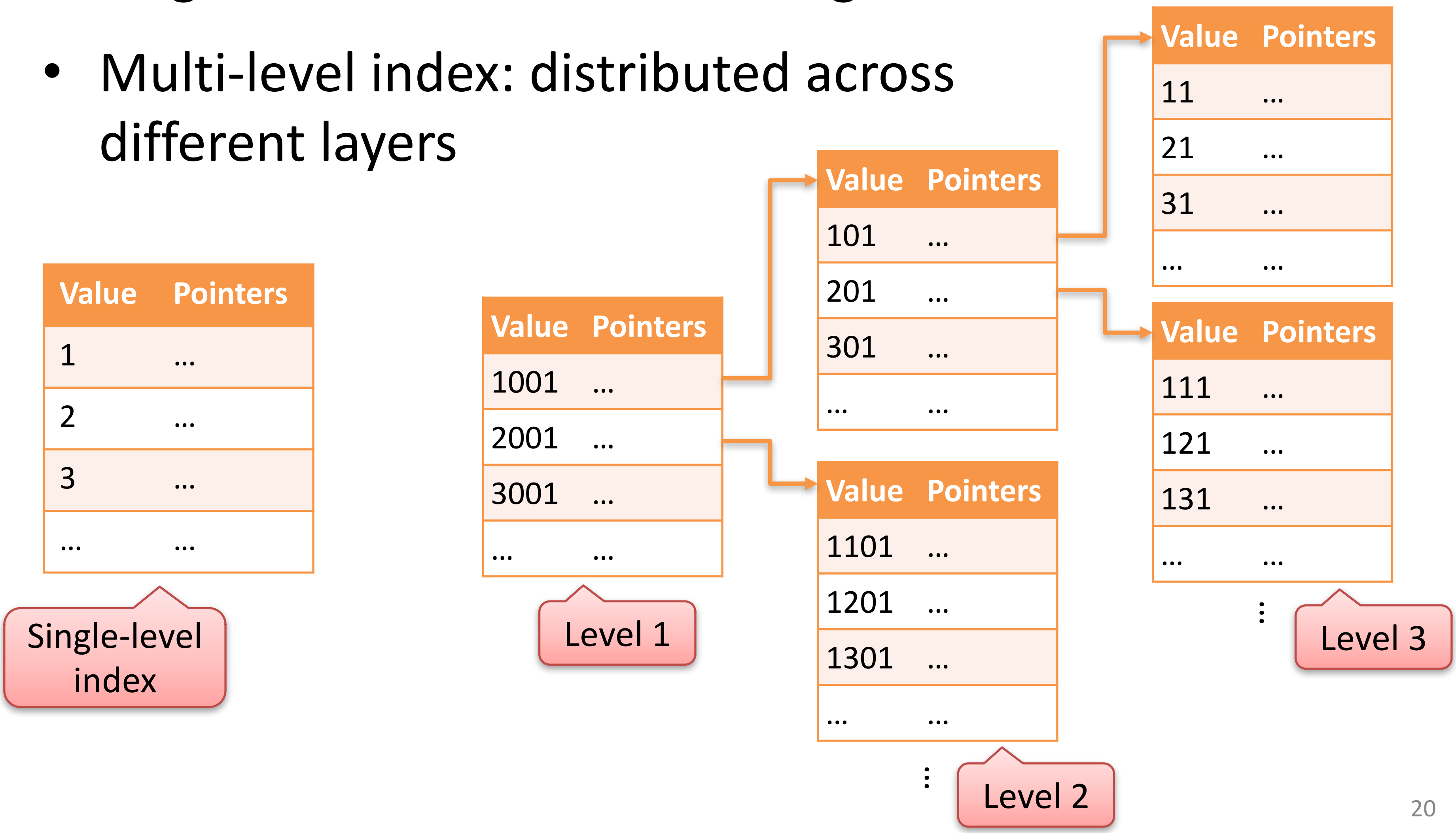
```
CREATE INDEX
  ON Students USING hash
  (programme);
```

```
CREATE INDEX
  ON Students USING hash
  (lower(name));
```

- Many more...

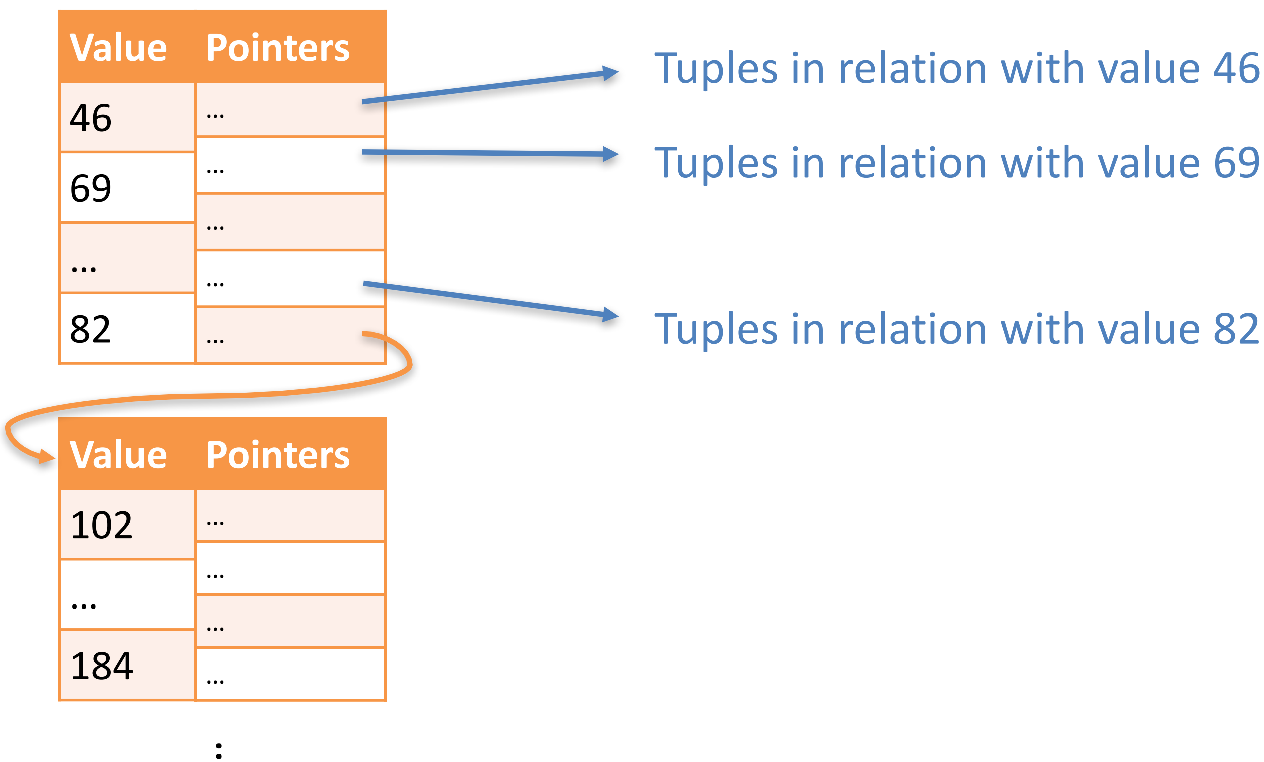
Single Level vs Multi-Level Indexes

- Single level index: stored in single list
- Multi-level index: distributed across different layers

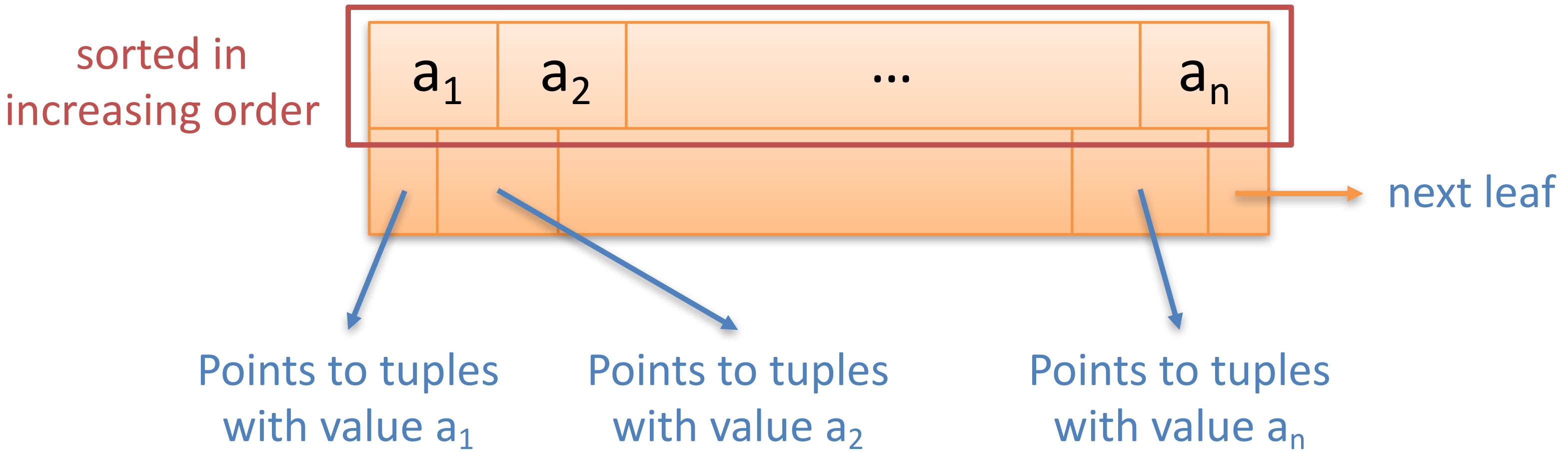


B+ Trees

- A multi-level index, but different shape than the one shown on the previous slide
- E.g., shape of a leaf in a B+ tree:



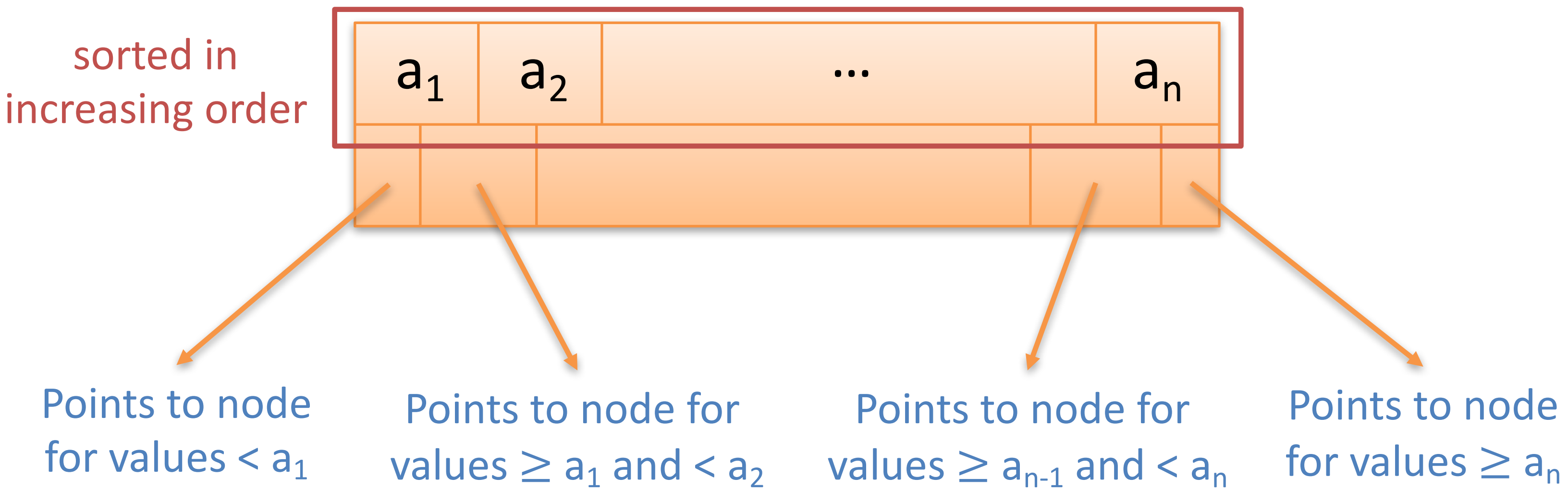
B+ Tree: Leaves (Idea)



- **n** = chosen such that node fits into a single disk block
- Example:
  - Disk block size = 512 byte
  - Values: 4 byte integers
  - Pointers: 8 bytes

  $\left.\right\}$ **n** = 42

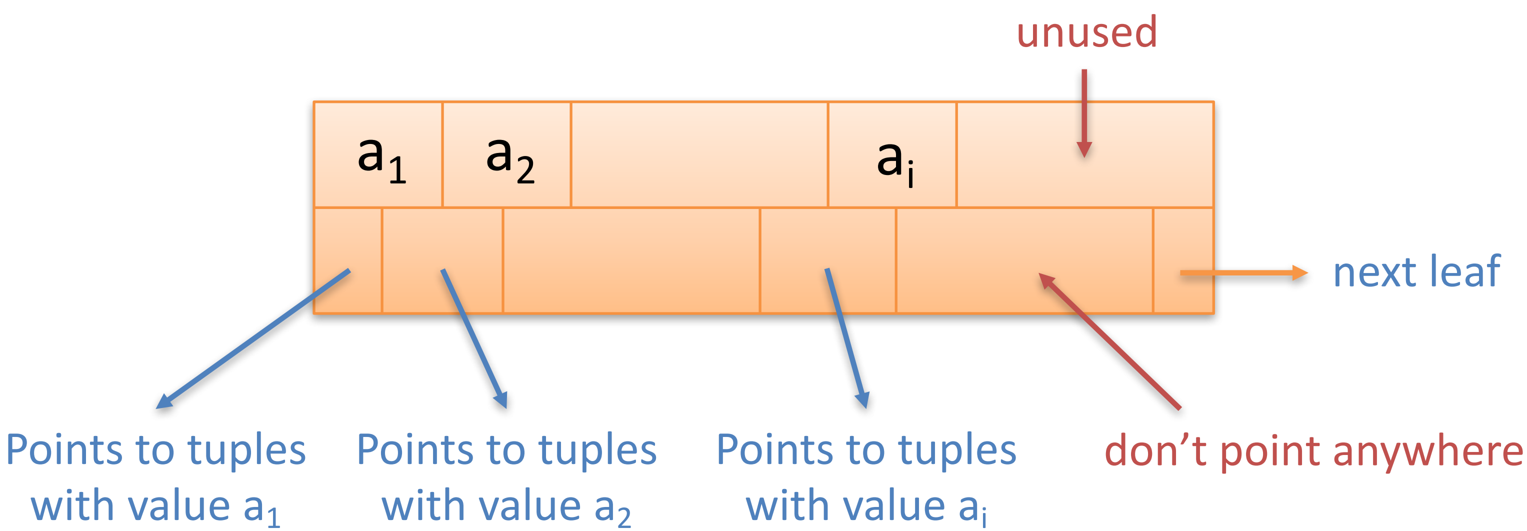
B+ Tree: Inner Nodes (Idea)



- Pointers point to B+ tree nodes at level below
- **n** = chosen as before (≥ 1), so there are ≥ 2 pointers

B+ Tree: Leaves (Actually)

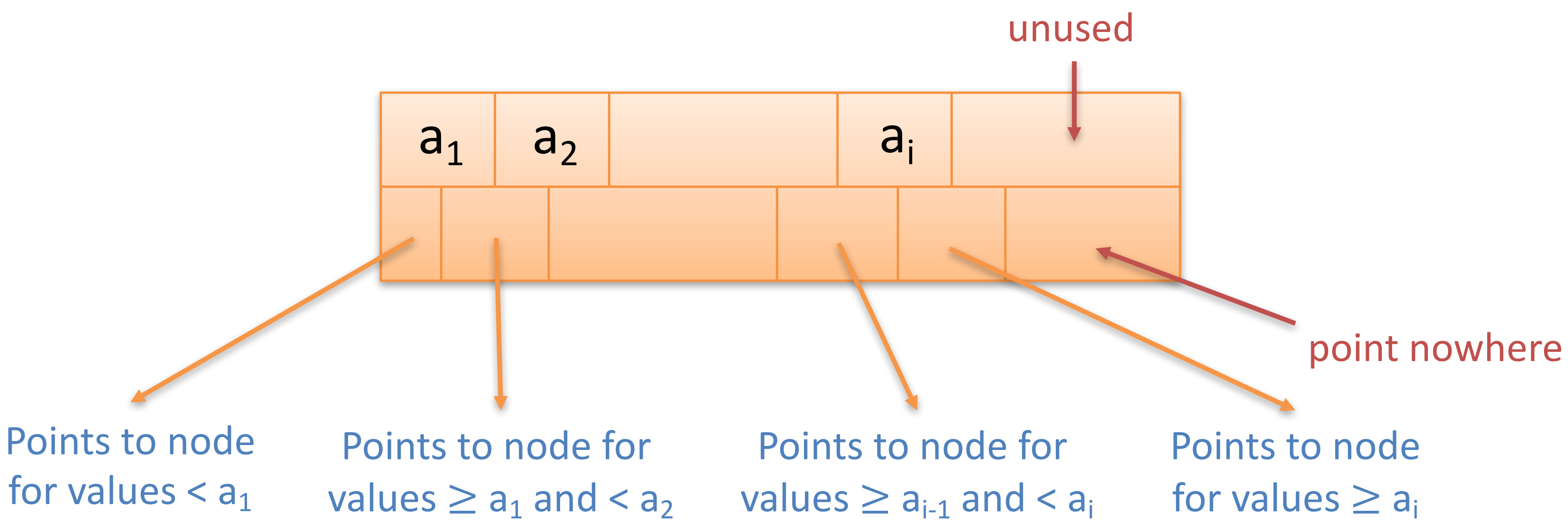
- Not all of the fields have to be used
- Fields are filled from left to right



- Ensure that at least $\left\lfloor \frac{n+1}{2} \right\rfloor$ pointers are used

B+ Tree: Inner Nodes (Actually)

- Similar as for leaves:



- Ensure that at least $\left\lceil \frac{n+1}{2} \right\rceil$ pointers are used
- Exception: root must use ≥ 2 pointers

To be continued…