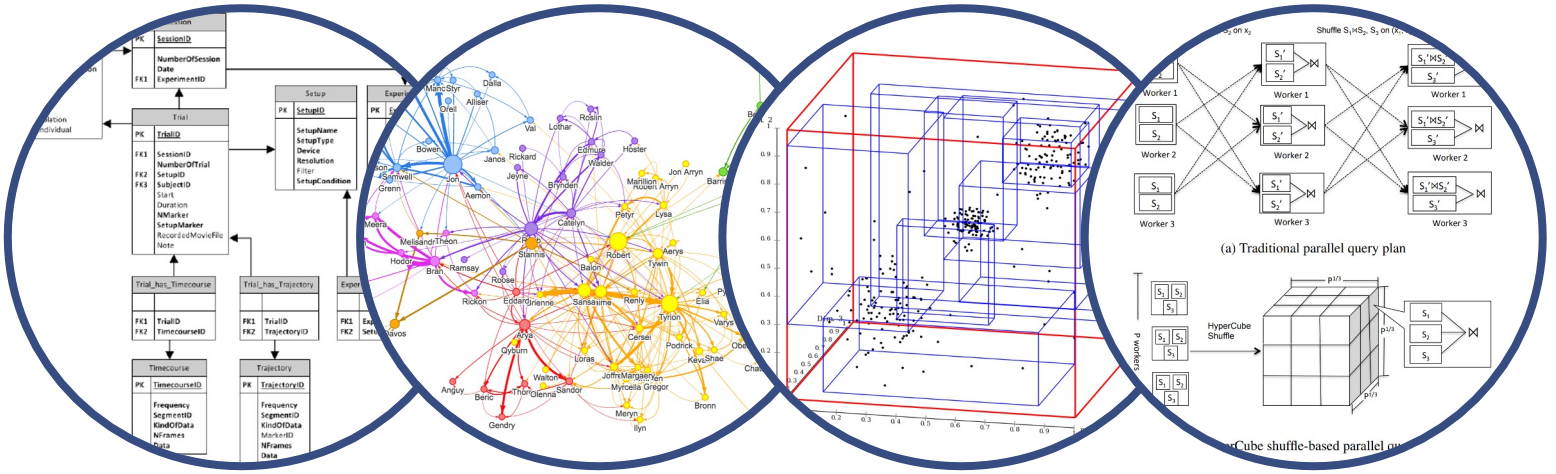


Introduction to Data Management

Query Optimization

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

Outline

- **Recap**

- Indexes
  - Scans
  - Selections
  - Joins

- Pipelined Execution

- Full Query Plan Costing

Recap: Optimization Pipeline

- Optimization:
  - find the equivalent RA Plan that **minimizes operator cardinality**
  - find the physical operator algorithms that **minimizes IO cost**

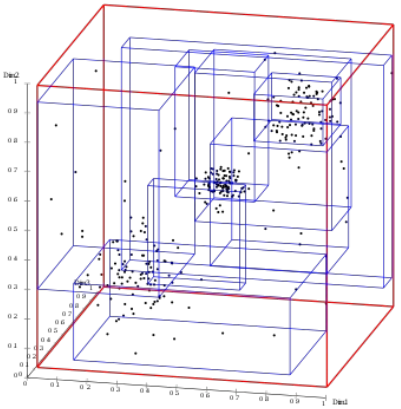
- Real RDBMS uses sophisticated cost models
  - I/O estimate in reads/writes
  - Compute estimate in FLOPS
  - Memory estimate in bytes

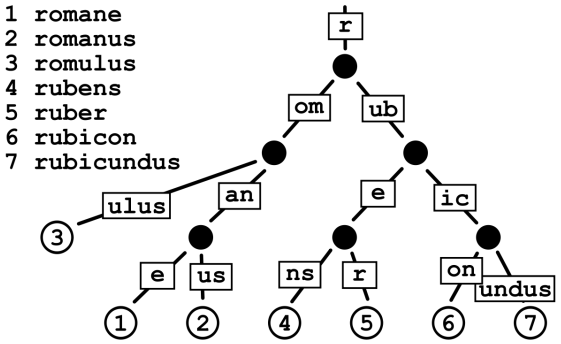
Recap: Indexes

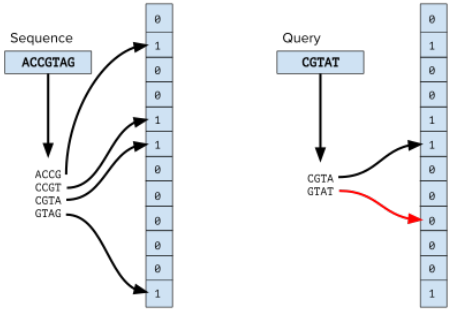
- An index is a **separate file that allows more direct access to a row** based on attribute(s)
  - Does not change the underlying relation's representation!

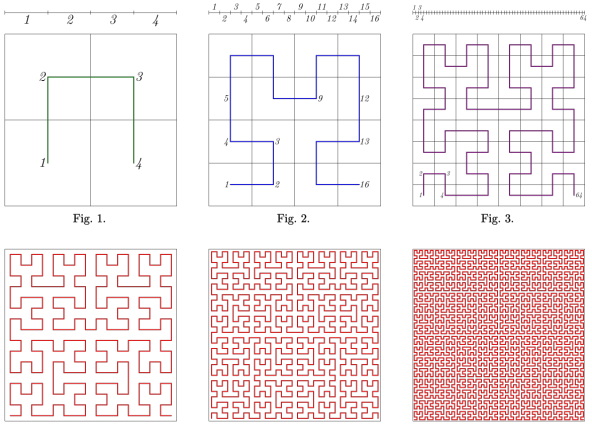
Recap: Index Structures

- **B+ Tree Index**
  - Clustered
  - Unclustered
- Hash Index
- R Tree
- Radix Tree
- Bloom Filter
- Hilbert Curves
- Learned Index
- LSMT
- …









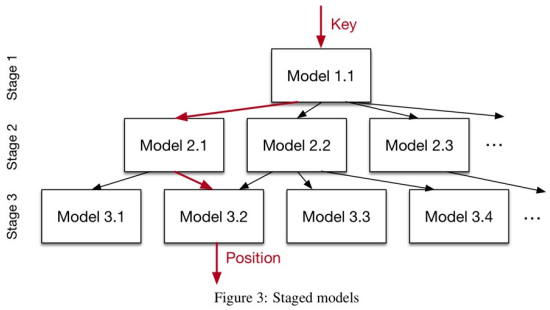
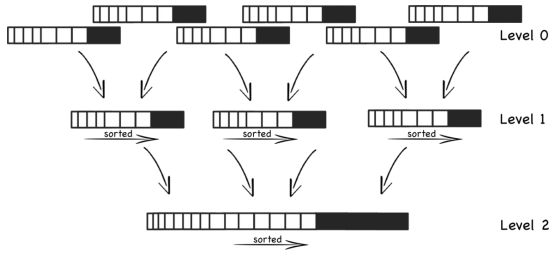


Figure 3: Staged models

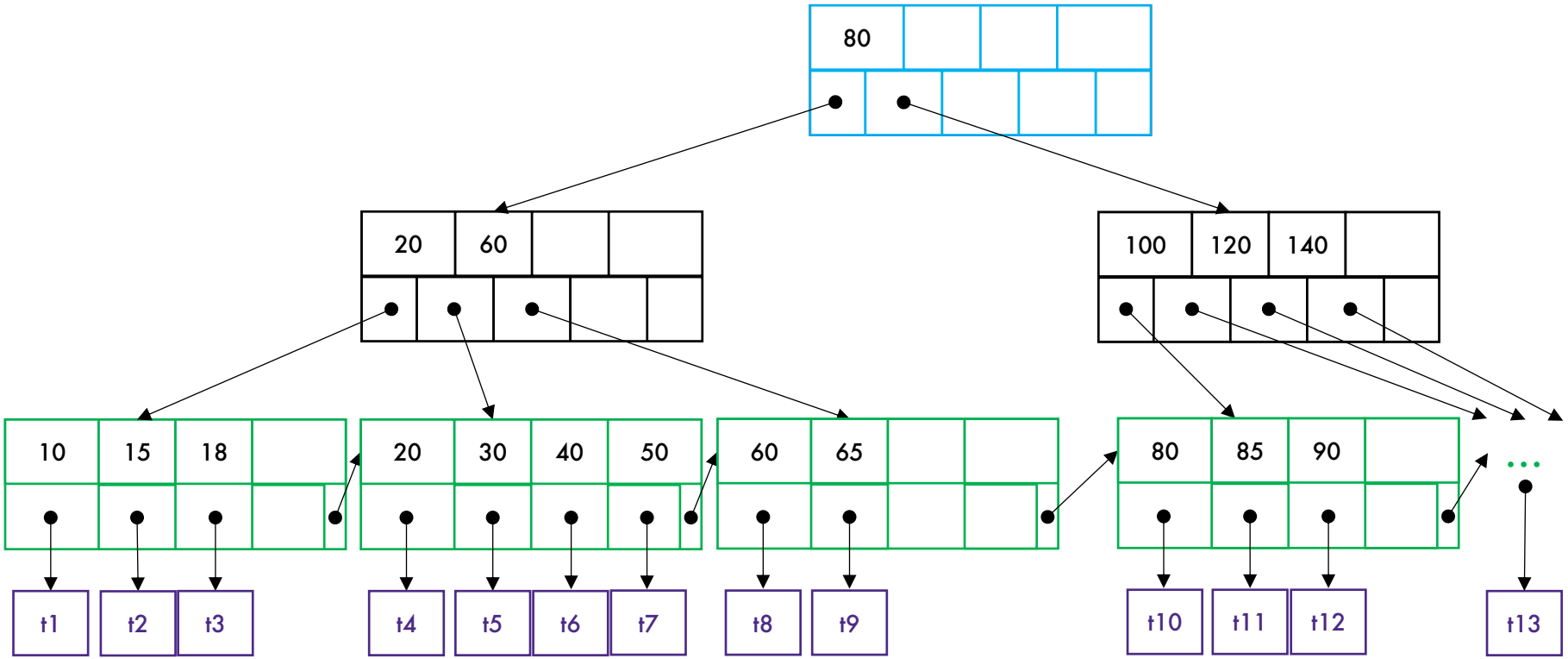


Compaction continues creating fewer, larger and larger files

Outline

- Recap
- **Indexes**
  - Scans
  - Selections
  - Joins
- Pipelined Execution
- Full Query Plan Costing

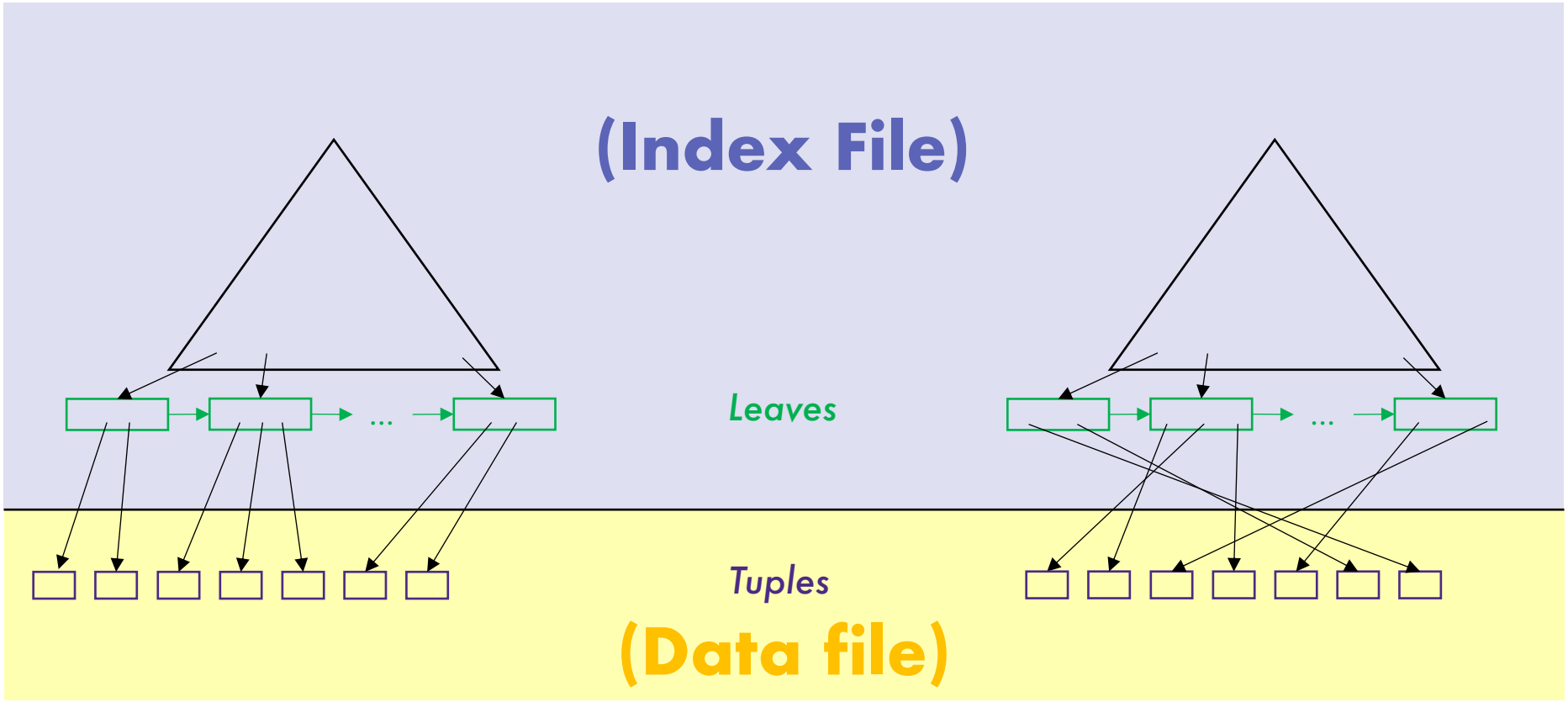
What is a B+ Tree?



Find the tuples associated whose keys are in the range [40, 75)

Same search process, then follow linked list!

Clustered vs Unclustered B+-Trees



CLUSTERED

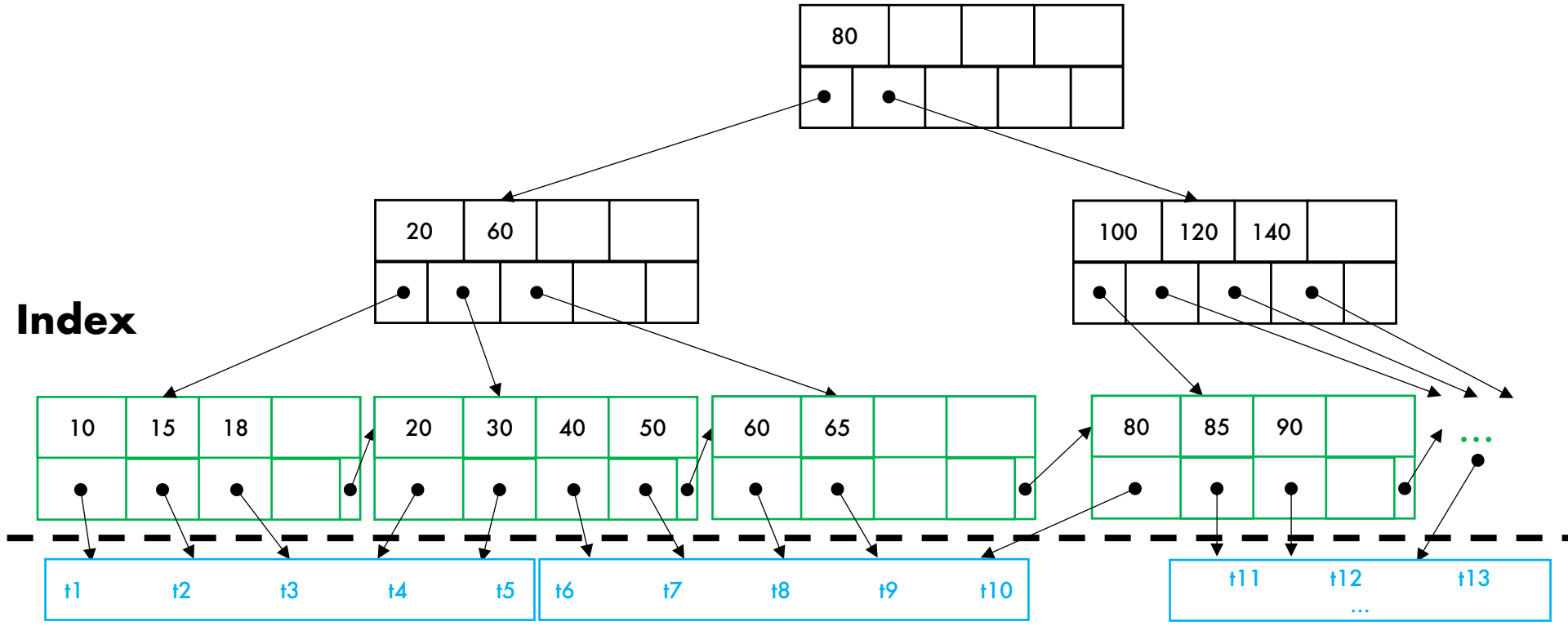
UNCLUSTERED

Note: can also store tuples directly in leaves

Outline

- Recap
- Indexes
  - **Scans**
  - Selections
  - Joins
- Pipelined Execution
- Full Query Plan Costing

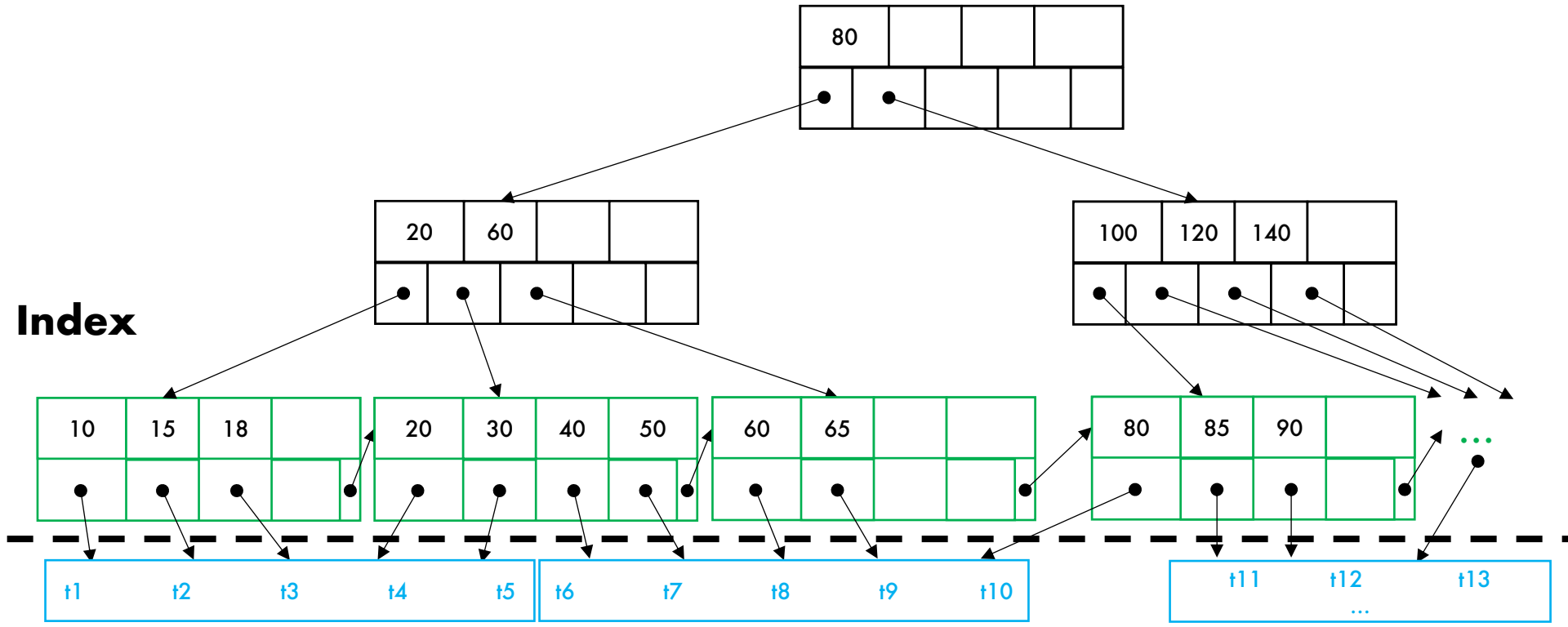
Clustered Index



A **clustered index** is one whose tuples are ordered on disk in the same as **the index's key order**

- Typically only have one clustered index per table

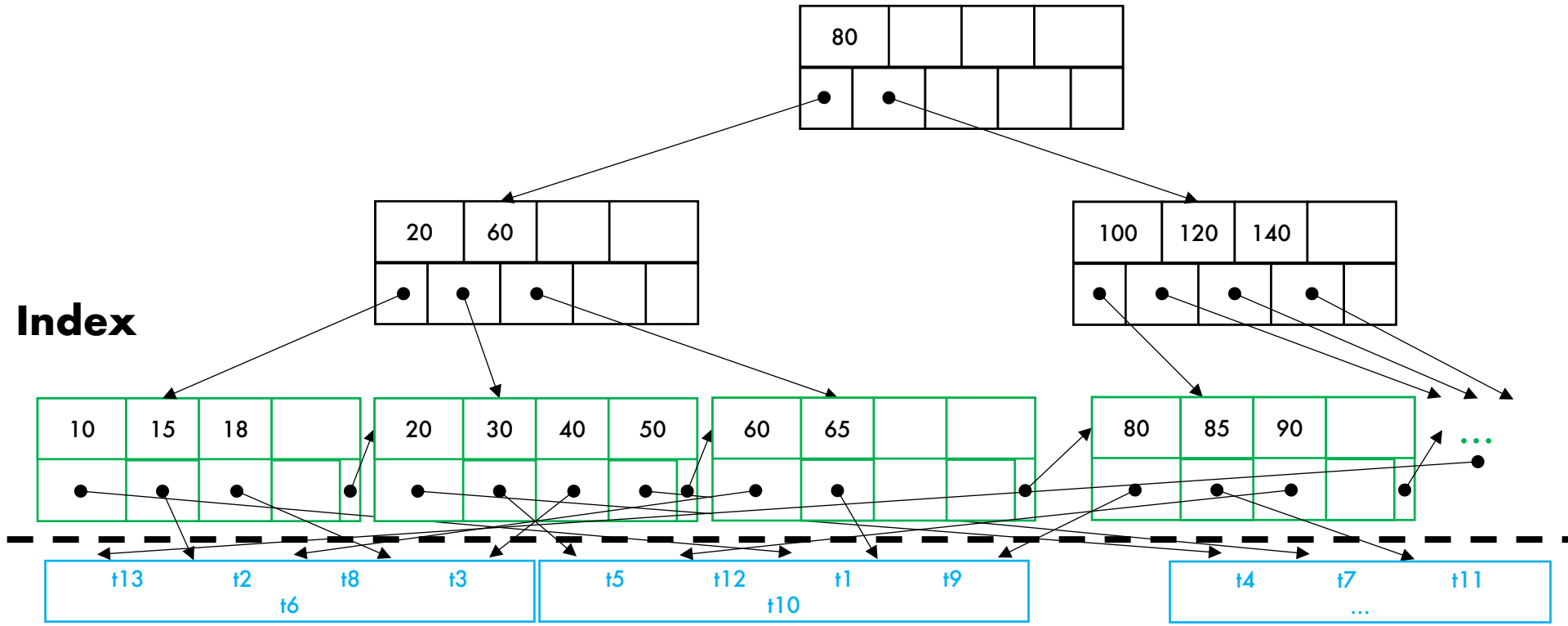
Clustered Index Scan



Find the first tuple on disk, then scan forward using our "linked list" pointers

- eg, Find tuples associated with keys (40-85]

Unclustered Index

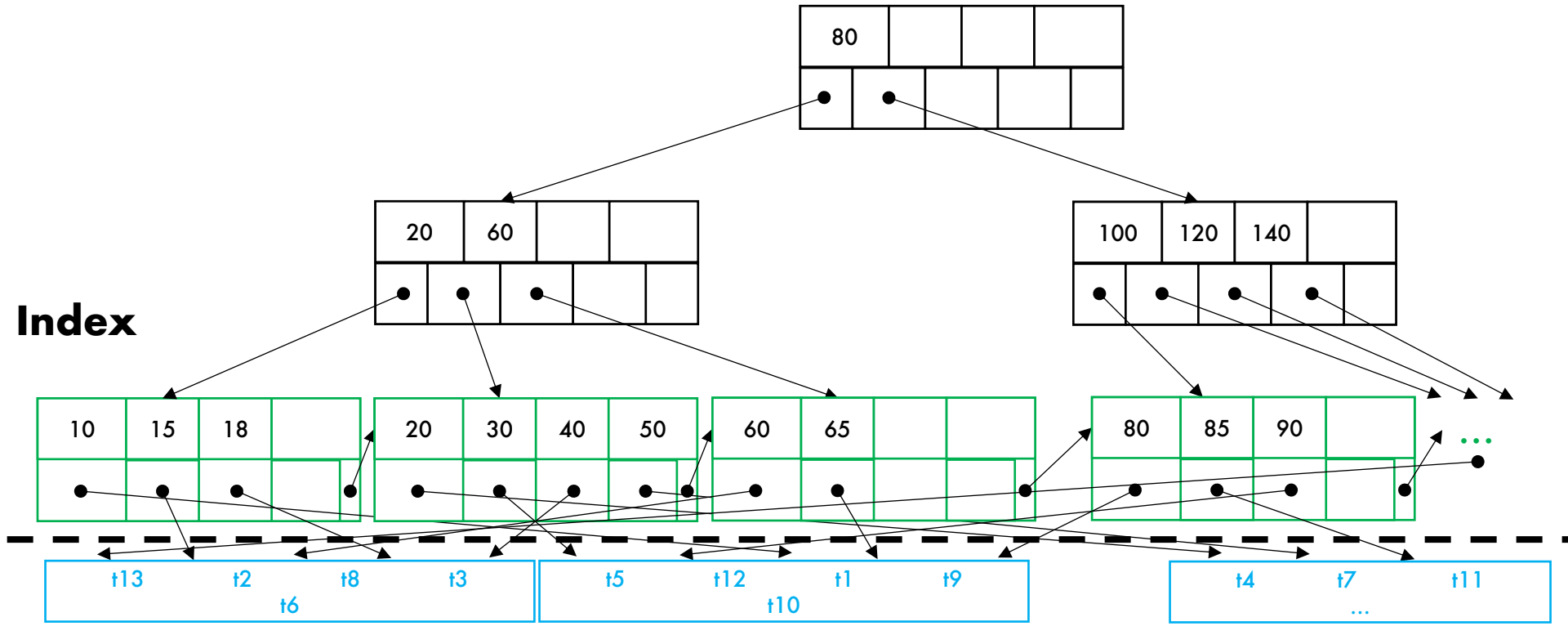


Sequential File with a different key, or **Heap File**

An **unclustered index**'s ordering differs from the on-disk ordering

- A table can have any number of unclustered indexes per table

Unclustered Index Scan



Sequential File with a different key or **Heap File**

Find the first tuple in the in-memory "leaf", then load tuples into memory

- eg, Find tuples associated with keys (40-85]

Sequential Scans

- When is a sequential scan better than an unclustered index scan?
  - To find the tuples associated with keys (40-85] it may be faster to do a single scan of the entire table instead of loading and unloading the same data file block repeatedly

- When might a sequential scan be better than a <u>clustered</u> index scan?

Outline

- Recap
- Indexes
  - Scans
  - **Selections**
  - Joins
- Pipelined Execution
- Full Query Plan Costing

Index-Based Selection

$$\sigma_{Name='Alice'}(R)$$

B(R) = # blocks
T(R) = # tuples
V(R, attr) = # distinct vals

- Physical Plan 1:
  - Sequential scan (full table scan on R)
  - Selection "on-the-fly"
  - Cost = B(R)

- Physical Plan 2:
  - Index lookup records with Name='Alice'
    - Random access
  - Unclustered index cost: T(R)/V(R,Name)
  - Clustered index cost: B(R)/V(R,Name)

Why the difference?

Index-Based Selection

$$\sigma_{\text{Name}='\text{Alice}'\wedge\text{City}='\text{Seattle}'\wedge\text{Job}='\text{TA}'}(\text{R})$$

- Physical Plan 1: sequential scan
- Physical Plan 2: index lookup on Name='Alice'
- Physical Plan 3: index lookup on City='Seattle'
- Physical Plan 4: index lookup on Job='TA'

Hard choice for the optimizer!

Outline

- Recap
- Indexes
  - Scans
  - Selections
  - **Joins**
- Pipelined Execution
- Full Query Plan Costing

Index Join

$$R \bowtie_{R.A=S.B} S$$

- Conceptually like a hash join

- For each record x in R:
  - Use index S.B to find x
  - Follow pointers to read all records where S.B=x
  - Unclustered index cost: T(R) * T(S)/V(S,B)
  - Clustered index cost: T(R) * B(S)/V(S,B)

- Makes sense when T(R) is small, for example if R is the result of some previous selection

Outline

- Recap
- Indexes
  - Scans
  - Selections
  - Joins
- **Pipelined Execution**
- Full Query Plan Costing

Multiple Pass or Single Pass?

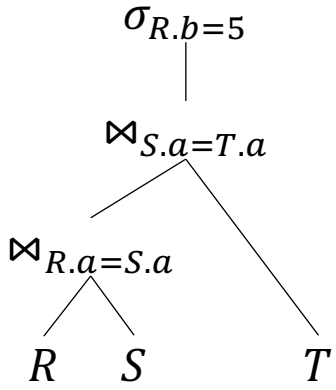
- Ideally, database only reads data from disk once per table
  - **Single-pass**

- Due to memory constraints, often have to throw out data already in memory to hold the next batch
  - Forces database to read same data from disk multiple times: **Multi-pass**

Executing Multiple Joins

- **Blocking Execution**
  - Subplans are completed and results stored before parent operation can start
  - Simple to implement!

- **Pipelined Execution**
  - Tuples are processed through the entire query plan as they become "ready" for the next operation
  - Low-latency results!

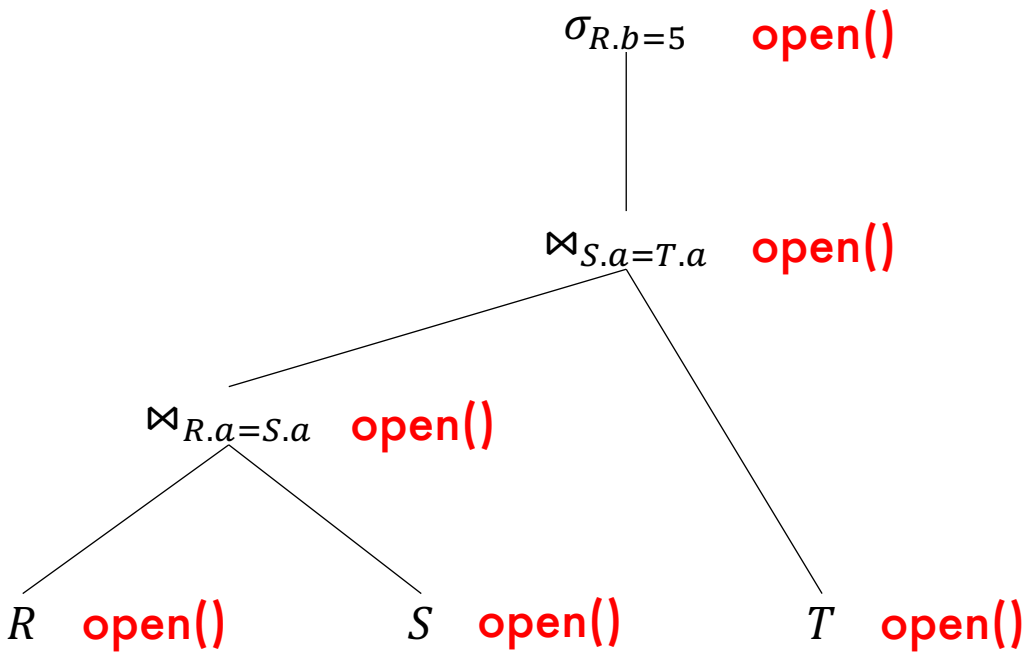


Pipelined Execution

- Iterator interface of RA operators
  - `open()` on every operator at start
  - `close()` every operator at end
  - `next()` to get the next tuple from a child operator or input table
- A.K.A. Volcano Iterator Model

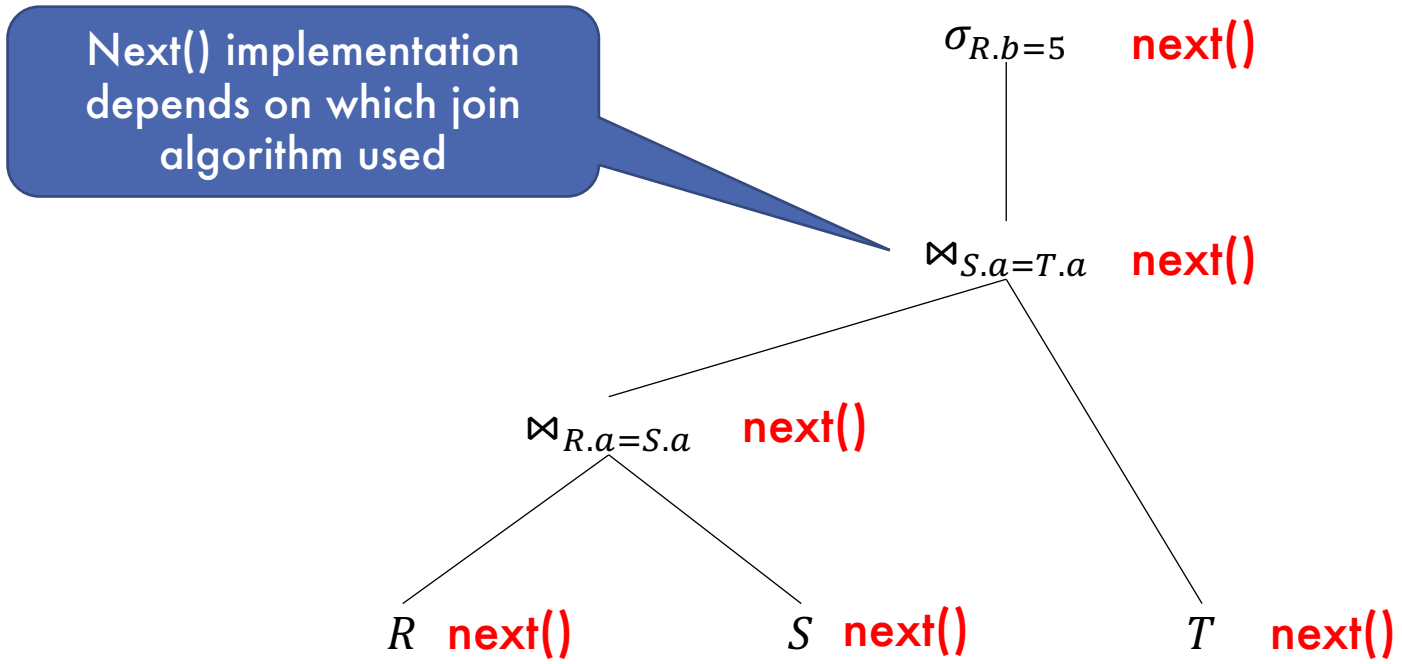
Pipelined Execution Example

```
SELECT *
FROM   R, S, T
WHERE  R.a = S.a
AND    S.a = T.a
AND    R.b = 5
```



Pipelined Execution Example

```
SELECT  *
FROM    R, S, T
WHERE   R.a = S.a
AND     S.a = T.a
AND     R.b = 5
```



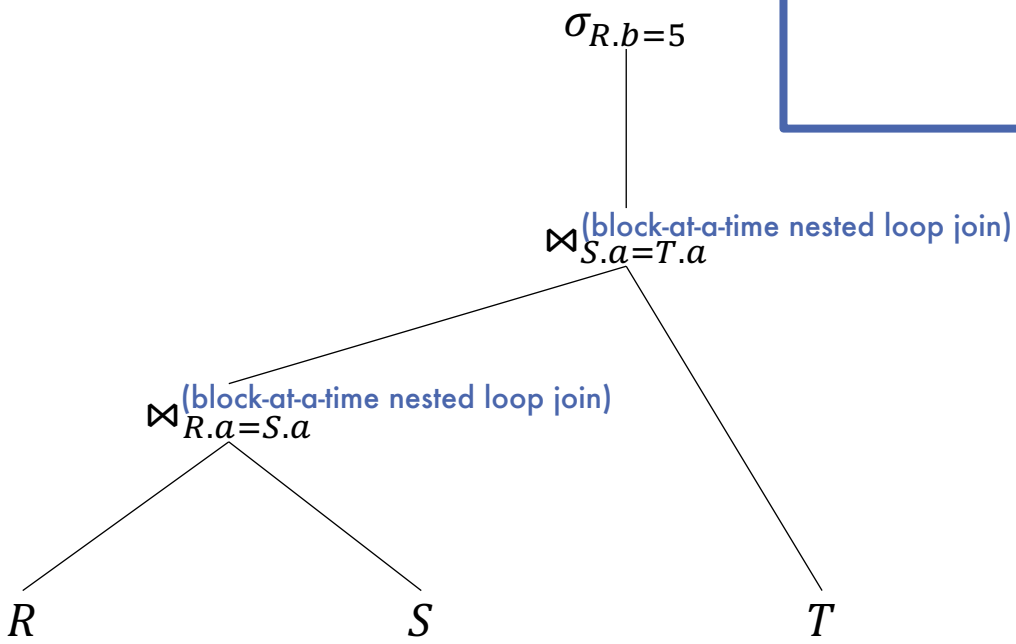
Pipelined Execution Example

```
SELECT  *
FROM    R, S, T
WHERE   R.a = S.a
AND     S.a = T.a
AND     R.b = 5
```

IO Cost for Blocking Execution

- Rare to be able to keep results in memory
  - Usually need to read entirety of previous stage off disk!

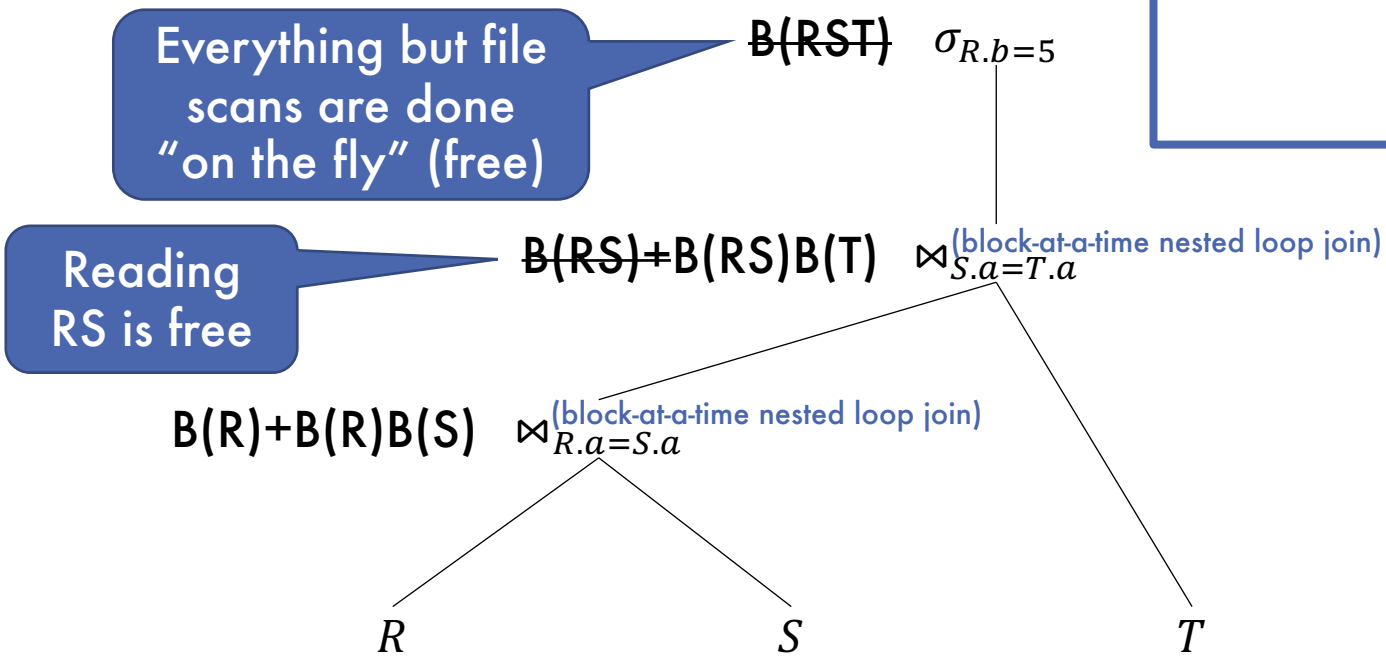
Est. Cost =



IO Cost for Pipelined Execution

- IO cost may be lower with pipelined execution
  - No need to read entirety of previous stage off disk
  - Generated tuples are already in main memory, so future access is "free"

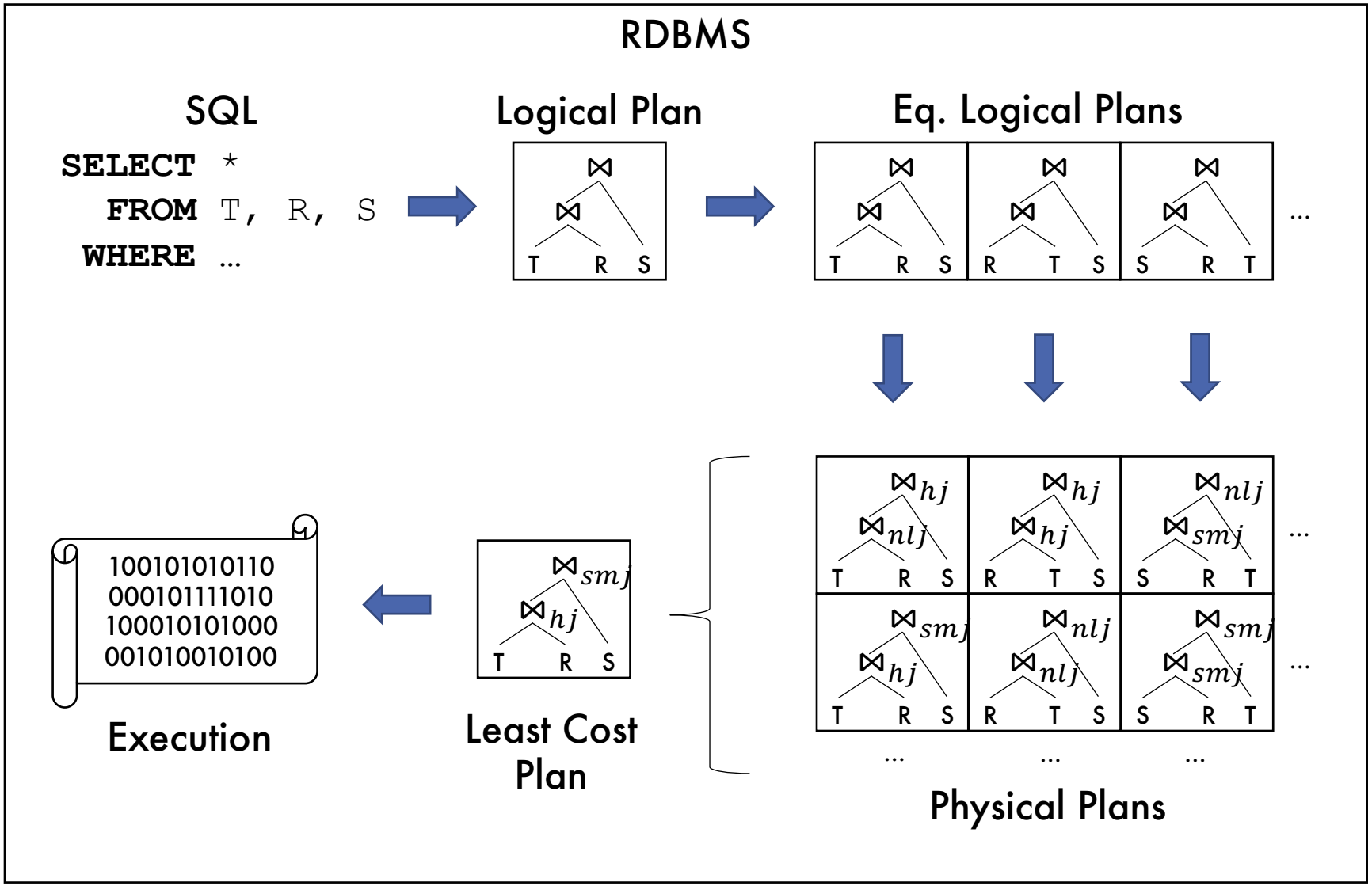
Est. Cost =



Outline

- Recap
- Indexes
  - Scans
  - Selections
  - Joins
- Pipelined Execution
- **Full Query Plan Costing**

Recap: Plan Enumeration to Execution



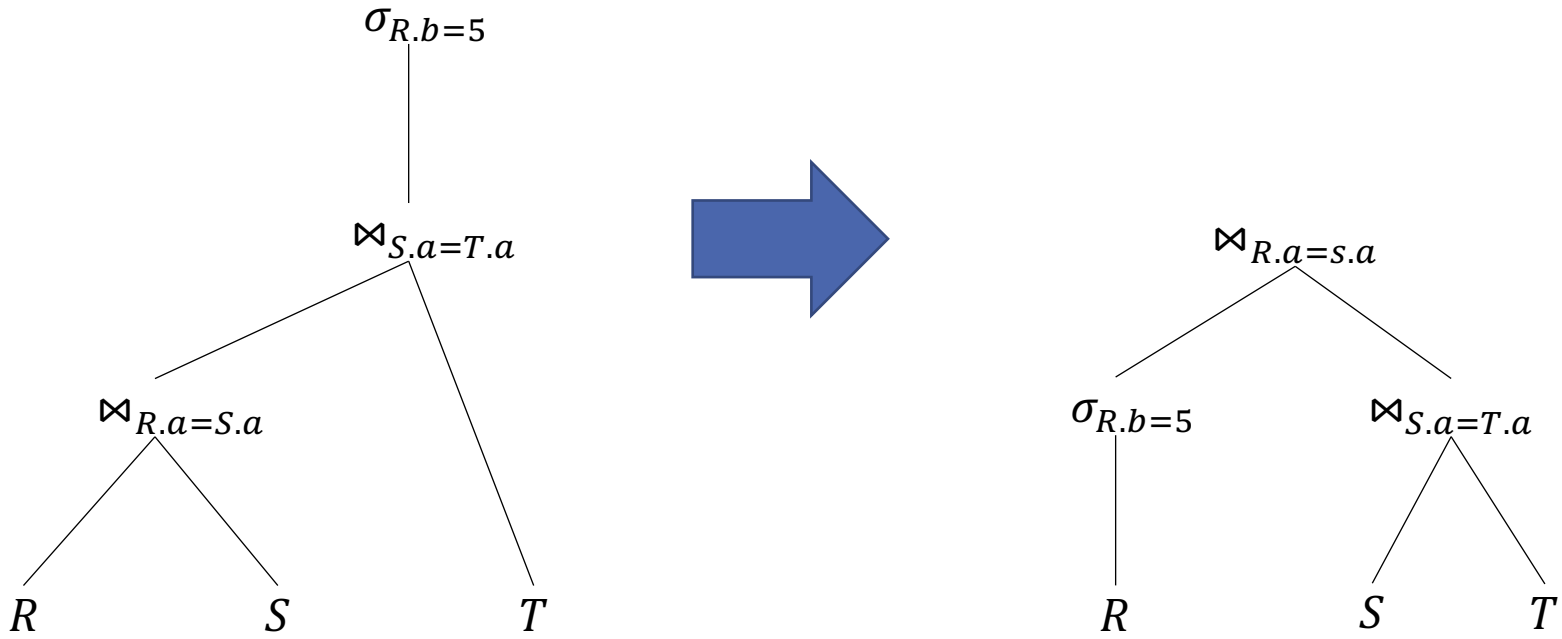
Cost Reduction Strategies

- Change the plan logically
  - Use RA-to-RA identities
- Change the plan physically
  - Use different operator implementations
- Choose an execution model

Full Plan Cost Estimation

- Change the plan logically
  - Use RA-to-RA identities
- *Change the plan physically*
  - *Use different operator implementations*
- *Choose an execution model*

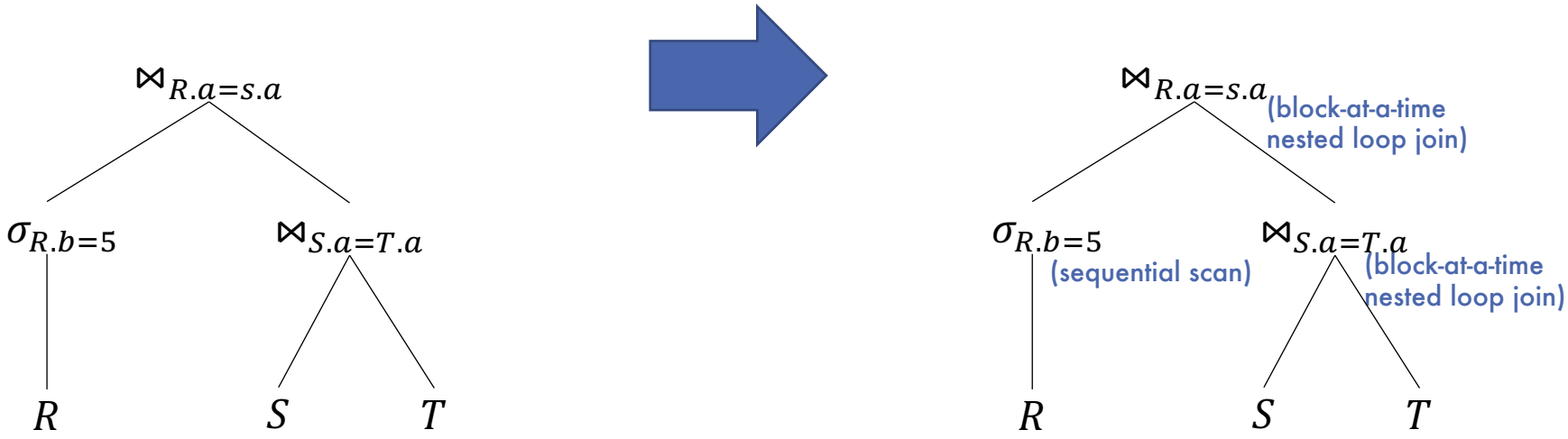
```
SELECT *
FROM   R, S, T
WHERE  R.a = S.a
AND    S.a = T.a
AND    R.b = 5
```



Full Plan Cost Estimation

- *Change the plan logically*
  - *Use RA-to-RA identities*
- **Change the plan physically**
  - **Use different operator implementations**
- *Choose an execution model*

Attempt #1

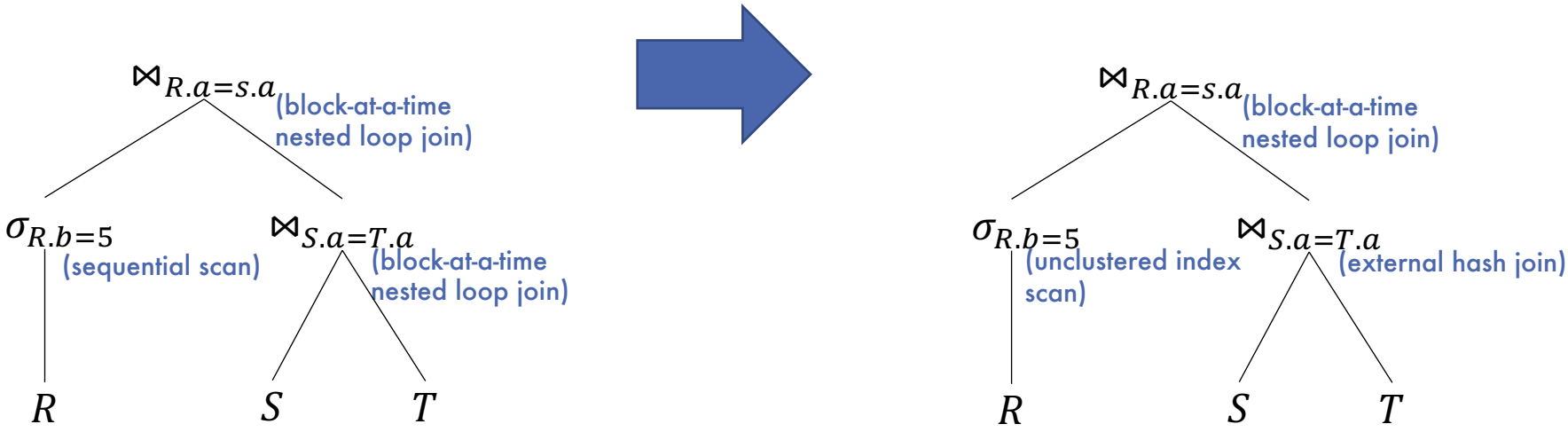


Full Plan Cost Estimation

- *Change the plan logically*
  - *Use RA-to-RA identities*
- **Change the plan physically**
  - **Use different operator implementations**
- *Choose an execution model*

Attempt #2:

- *Use unclustered index on R.b*
- *Change join algorithm*



Full Plan Cost Estimation

- *Change the plan logically*
  - *Use RA-to-RA identities*
- *Change the plan physically*
  - *Use different operator implementations*
- **Choose an execution model**

B(R) = # blocks
T(R) = # tuples
V(attr, R) = # distinct vals

Est. Cost =