CS213
Principles of Database Systems(H)

Chapter 13

Shiqi YU 于仕琪

yusq@sustech.edu.cn

13.1 Query Optimizer

Shiqi Yu 于仕琪

yusq@sustech.edu.cn

Most contents are from Stéphane Faroult and Yuxuan Ma's slides

1978 Version 1

1985 Version 5

1988 Version 6

1987 Version 1

1989

1988 Prototype

1994 First release

ALL major DBMS products were designed in the 1980s (MySQL is more recent but was based on SQL Server). They CANNOT change their architecture because it would be, business-wise, suicidal for them to require from customers dumping and reloading today's massive databases for a migration. Only incremental improvements are possible.

This is what a $M 1 machine was

Mid 1980s looking like in the 1980s.

VAX 8600 (high-end)

32-bit architecture

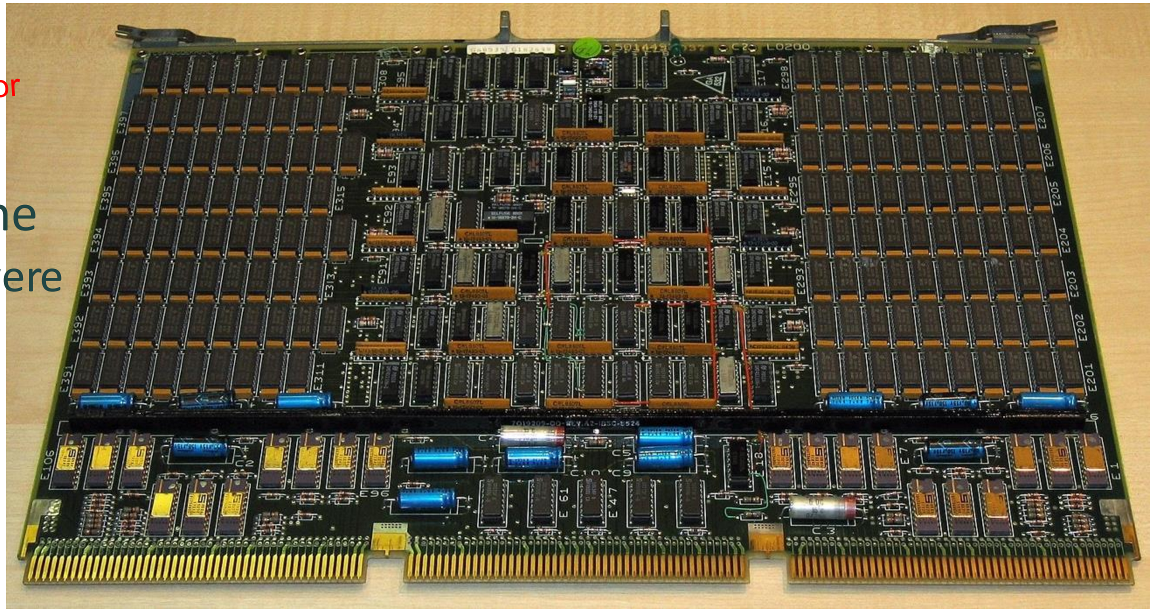
Processor, ~10 to 20MHz clock

Beginning of multi-processor computers

4 to 256 Mb of memory

Keep in mind that the big DBMS products were designed for this.

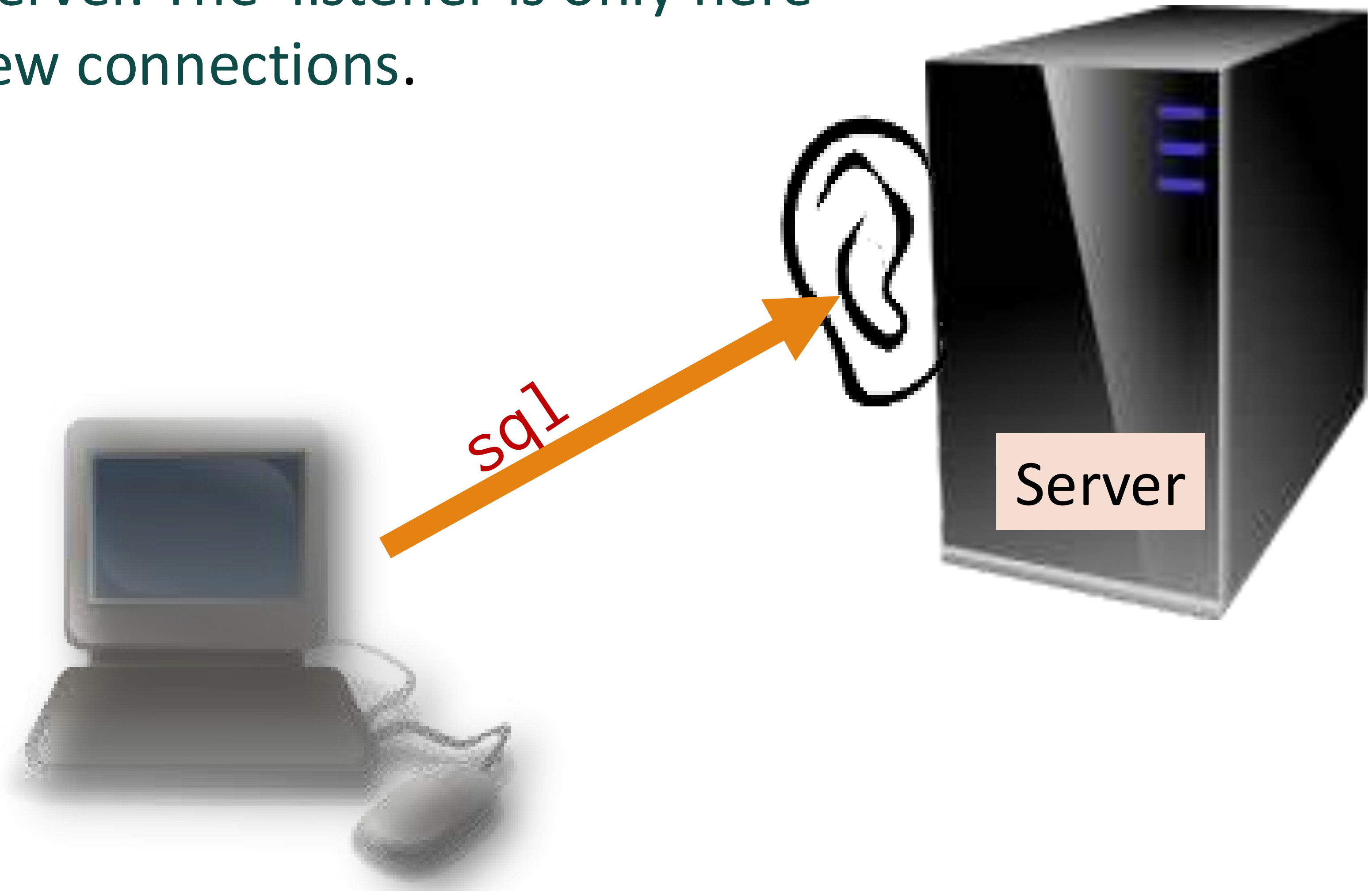
I/Os ~10 to 30 Mb/s



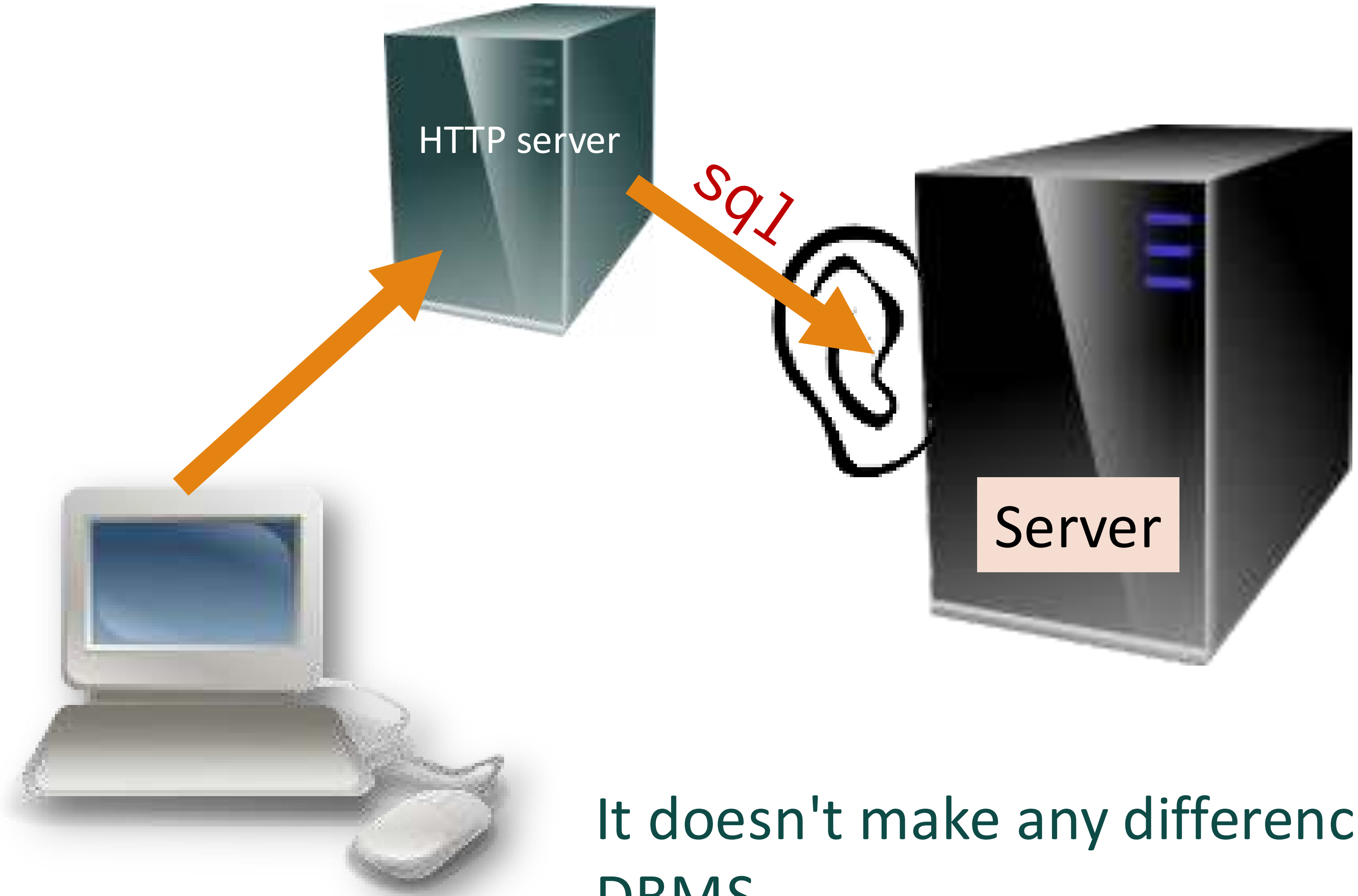
A 4 MB memory board for the VAX 8600

https://en.wikipedia.org/wiki/File:4mbramvax.jpg

SQL queries will be directly sent to this server. The listener is only here for new connections.



In many cases (HTTP server, application server) the end user isn't directly talking to the DBMS server.



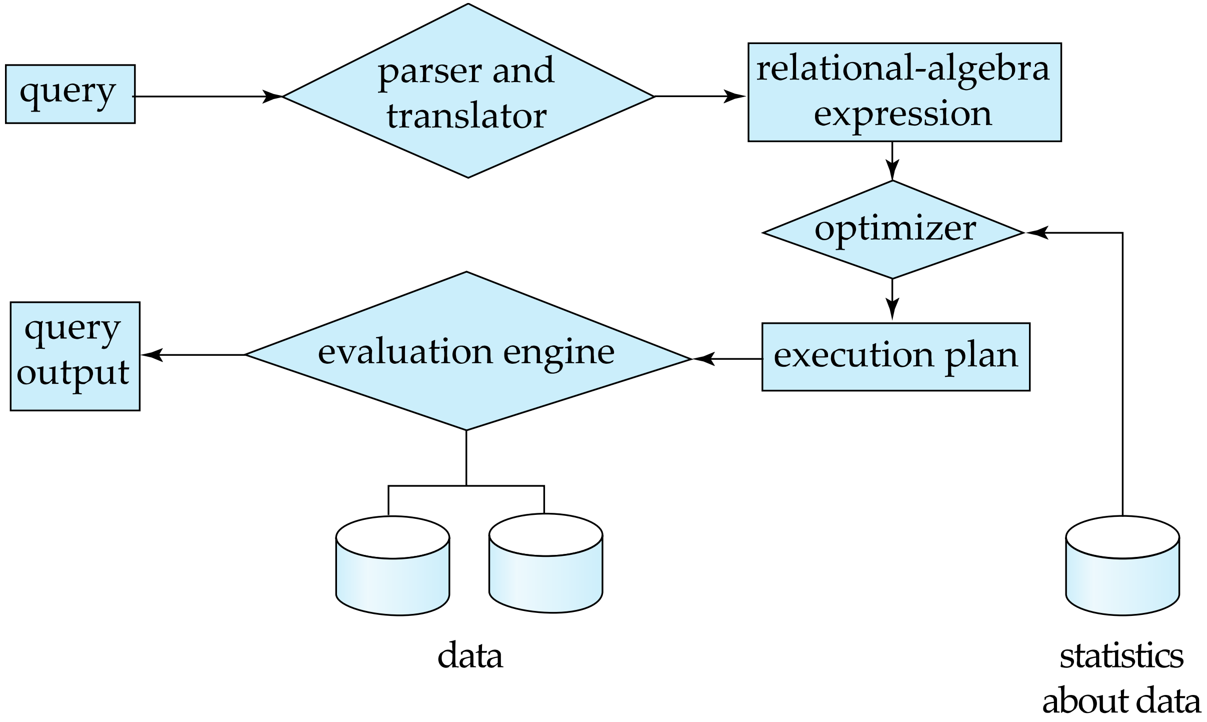
It doesn't make any difference for the DBMS.

Basic Steps in Query Processing

Parsing and Translation

Optimization

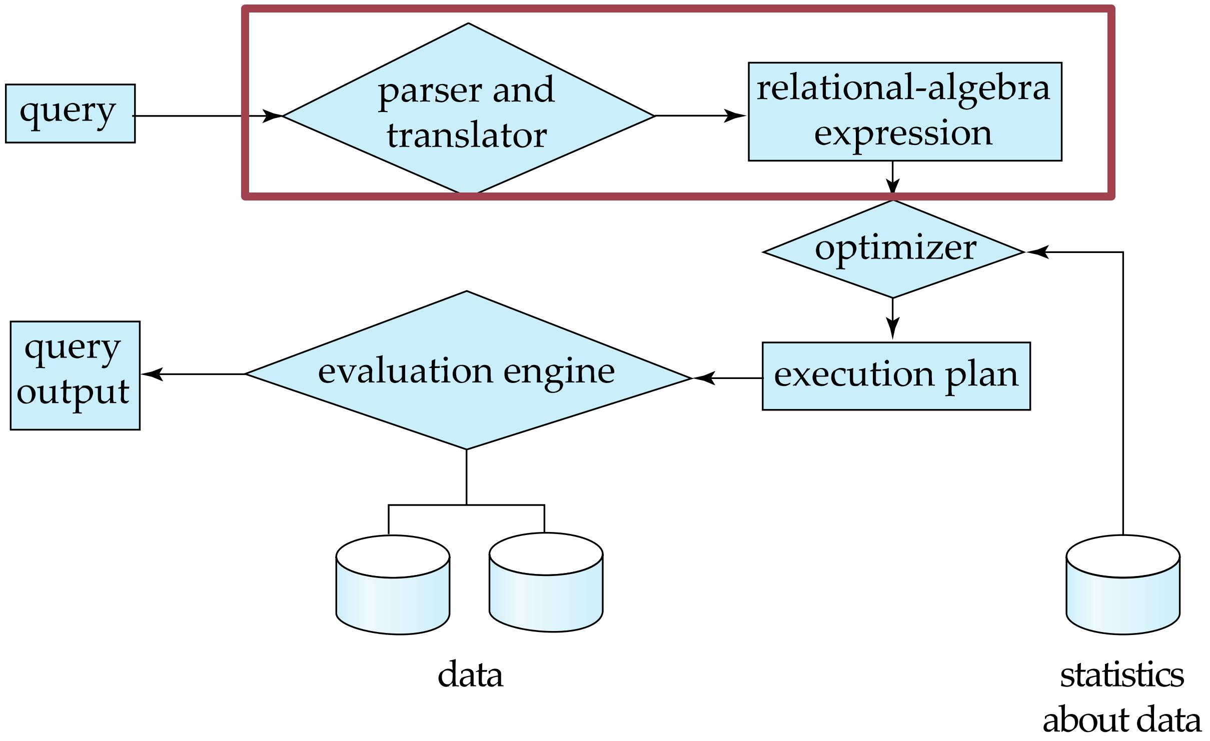
Evaluation



Basic Steps in Query Processing

Parsing and Translation

- Translate the query into its internal form
  - The internal form is then translated into relational algebra
- Parser checks syntax and verifies relations



Basic Steps in Query Processing

Optimization

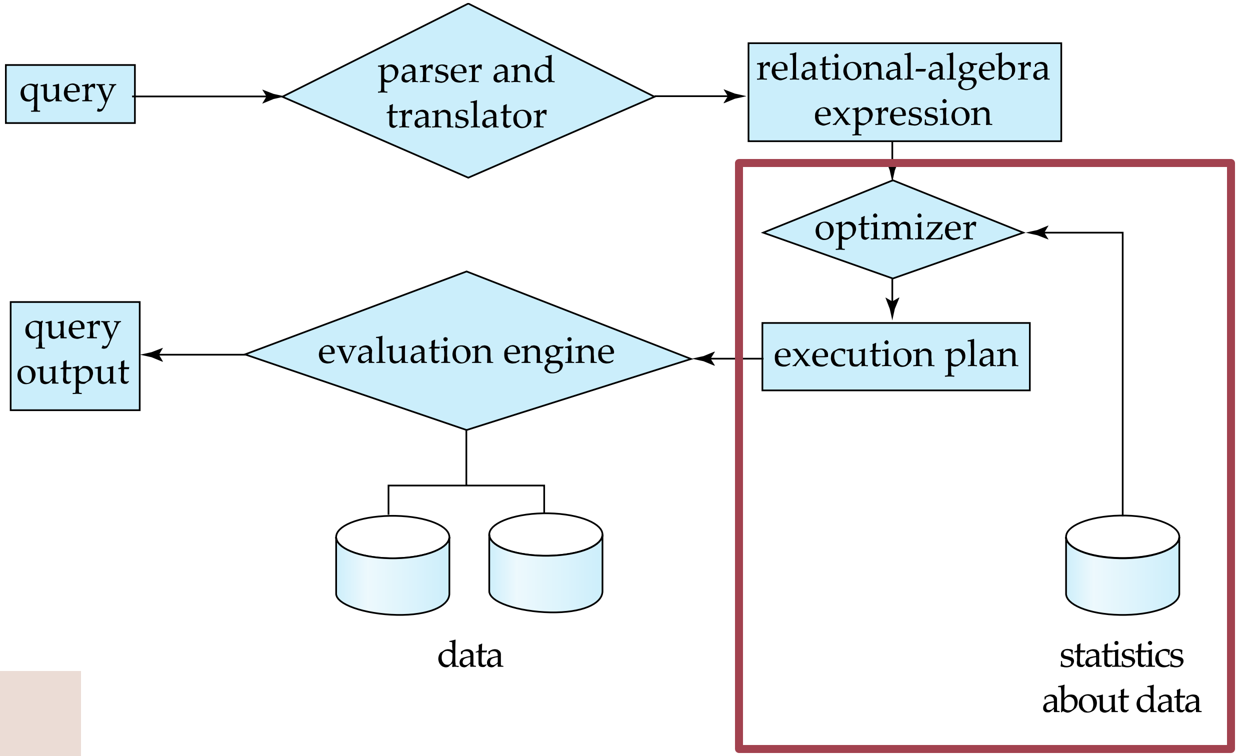
- A relational algebra expression may have many equivalent expressions
- E.g.,

$$\sigma_{salary<75000}(\Pi_{salary}(\text{instructor}))$$

is equivalent to

$$\Pi_{\text{salary}}(\sigma_{\text{salary}<75000}(\text{instructor}))$$

But the number of rows involved in the projection operation may be (significantly) smaller in the second expression



Basic Steps in Query Processing

Optimization

- A relational algebra expression may have many equivalent expressions
  - … and each relational algebra operation can be evaluated using one of several different algorithms

- *Correspondingly, a relational-algebra expression can be evaluated in many ways*

Basic Steps in Query Processing

Optimization

- **Evaluation Plan**: Annotated expression specifying detailed evaluation strategy
- E.g.,:
  - Use an index on `salary` to find instructors with `salary<75000`
  - Or perform complete relation scan and discard instructors with `salary<75000`

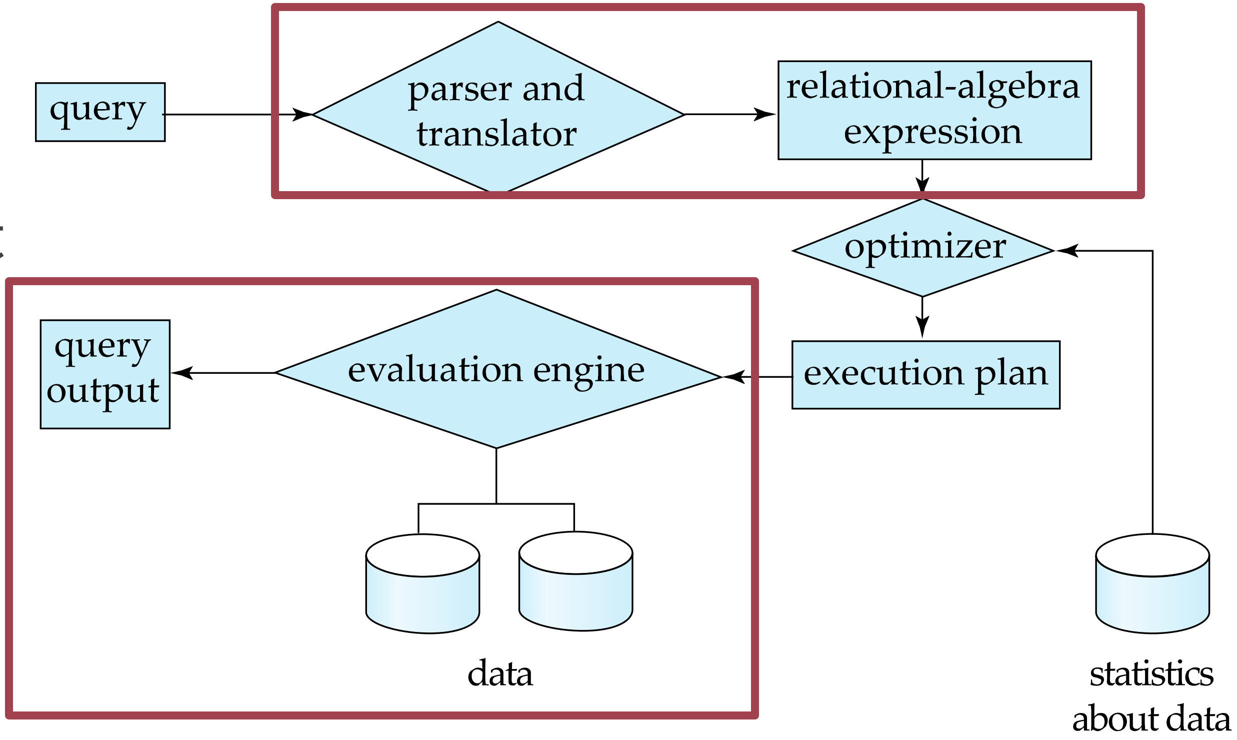
Query Optimization: Choose the one with the lowest cost among all equivalent evaluation plans

- Cost can be estimated using statistical information from the database catalog
  - E.g., Number of tuples in each relation, size of tuples, etc.

Basic Steps in Query Processing

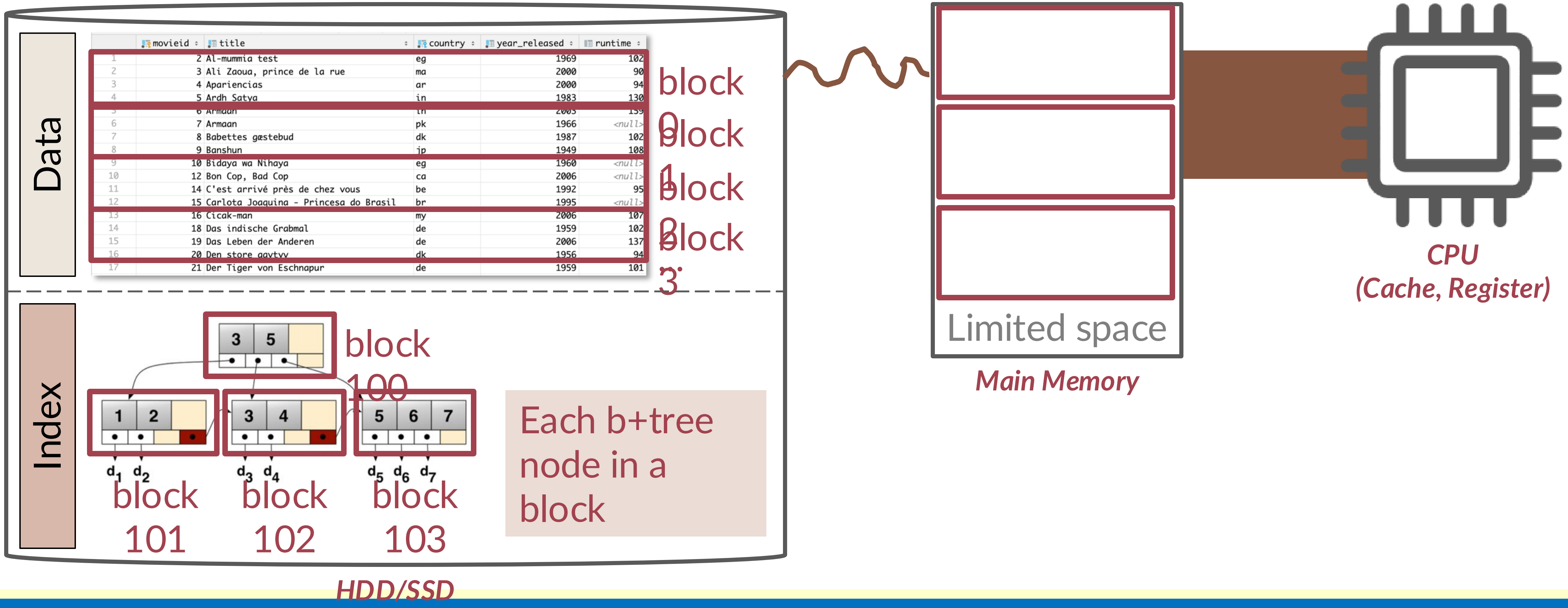
Evaluation

- The query-execution engine takes a query-evaluation plan, executes that plan, and returns the answers to the query



Prerequisite

Storage model



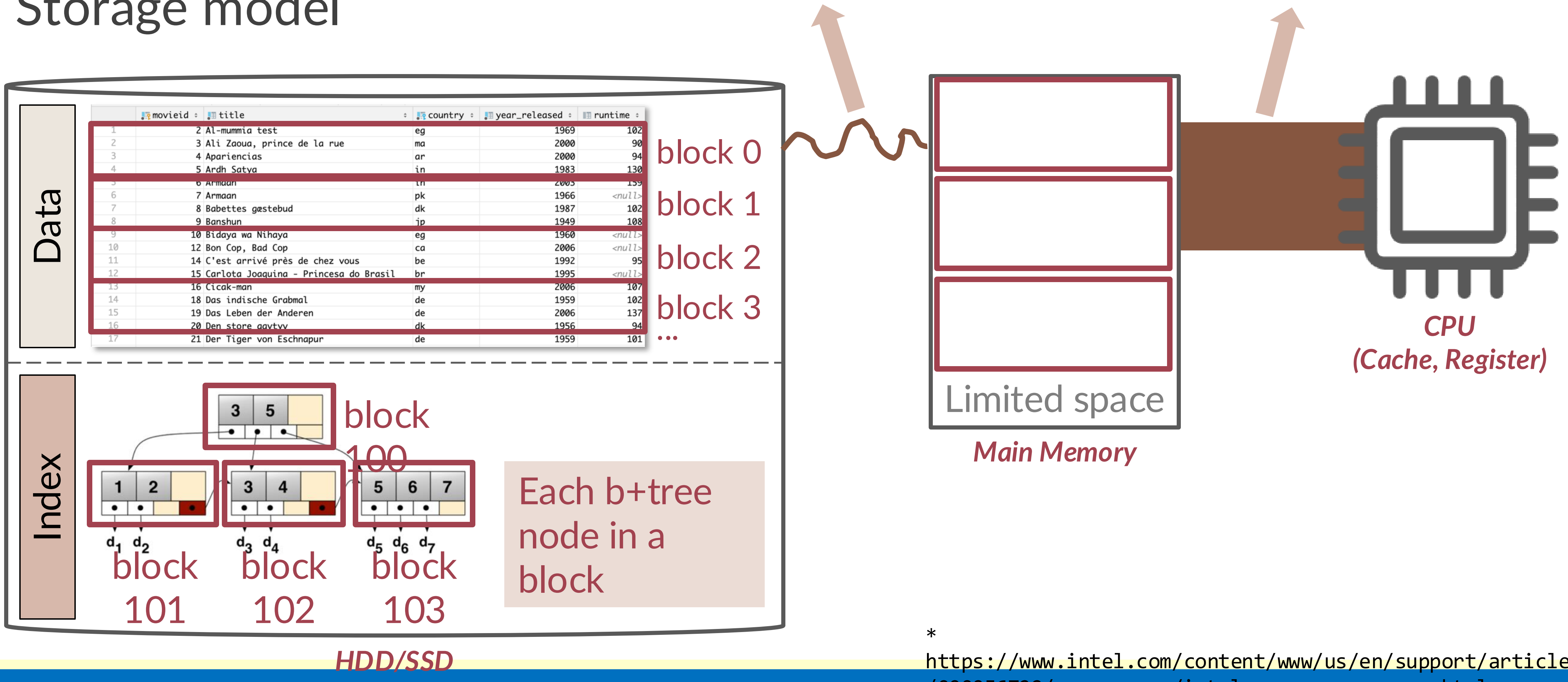
	movieid	title	country	year_released	runtime
1	2	Al-mummia test	eg	1969	102
2	3	Ali Zaoua, prince de la rue	ma	2000	90
3	4	Apariencias	ar	2000	94
4	5	Ardh Satya	in	1983	130
5	6	Armaan	in	2003	159
6	7	Armaan	pk	1966	<null>
7	8	Babettes gæstebud	dk	1987	102
8	9	Banshun	jp	1949	108
9	10	Bidaya wa Nihaya	eg	1960	<null>
10	12	Bon Cop, Bad Cop	ca	2006	<null>
11	14	C'est arrivé près de chez vous	be	1992	95
12	15	Carlota Joaquina - Princesa do Brasil	br	1995	<null>
13	16	Cicak-man	my	2006	107
14	18	Das indische Grabmal	de	1959	102
15	19	Das Leben der Anderen	de	2006	137
16	20	Den store gavtyv	dk	1956	94
17	21	Der Tiger von Eschnapur	de	1959	101

Prerequisite

Storage model

- Relatively small bandwidth
  - 100MB/s ~ <10GB/s
- High latency
  - Millisecond-level

- Very large bandwidth
  - 94GB/s (for DDR4 2933*)
- Very low latency
  - Nanosecond-level



	movieid	title	country	year_released	runtime
1	2	Al-mummia test	eg	1969	102
2	3	Ali Zaoua, prince de la rue	ma	2000	90
3	4	Apariencias	ar	2000	94
4	5	Ardh Satya	in	1983	130
5	6	Armaan	in	2003	159
6	7	Armaan	pk	1966	<null>
7	8	Babettes gæstebud	dk	1987	102
8	9	Banshun	jp	1949	108
9	10	Bidaya wa Nihaya	eg	1960	<null>
10	12	Bon Cop, Bad Cop	ca	2006	<null>
11	14	C'est arrivé près de chez vous	be	1992	95
12	15	Carlota Joaquina - Princesa do Brasil	br	1995	<null>
13	16	Cicak-man	my	2006	107
14	18	Das indische Grabmal	de	1959	102
15	19	Das Leben der Anderen	de	2006	137
16	20	Den store gavtyv	dk	1956	94
17	21	Der Tiger von Eschnapur	de	1959	101

* https://www.intel.com/content/www/us/en/support/articles/000056722/processors/intel-core-processors.html

Prerequisite

Measuring query cost

- Disk cost can be estimated as:
  - Number of seeks * average-seek-cost
  - Number of blocks read* average-block-read-cost
  - Number of blocks written * average-block-write-cost
- For simplicity, we just use the **number of block transfers** *from disk and the* **number of seeks** as the cost measures
  - t_T – time to transfer one block
    - Assuming for simplicity that write cost is same as read cost
  - t_S – time for one seek
- E.g., cost for *b* block transfers plus *S* seeks

  $b * t_T + S * t_S$

Prerequisite

Measuring query cost

- t_S and t_T depend on where data is stored. With 4 KB blocks:
  - High end magnetic disk: t_S = 4 msec and t_T =0.1 msec
  - SSD: t_S = 20-90 microsec and t_T = 2-10 microsec for 4KB
- Required data may be buffer resident already, avoiding disk I/O
  - But hard to take into account for cost estimation
- Worst case estimates assume that no data is initially in buffer and only the minimum amount of memory needed for the operation is available
  - But more optimistic estimates are used in practice
- We ignore CPU costs for simplicity
  - Real systems do take CPU cost into account
  - Network costs must be considered for parallel systems

Selection Operation

Let's start from this simple query:

```
select * from movies where [CONDITION];
```

- If you are the designer of the database engine, what do you think is the best way to fulfill this requirement?
- Two factors to consider:
  - What comparison is it in the CONDITION (equality / comparison)?
  - Does the column involved in the CONDITION have an index?

Basic Linear Scan

Linear Search (displayed as `Seq Scan` in PostgreSQL)

- Scan each file block and test all records to see whether they satisfy the selection condition
- Cost estimate = b_r block transfers + 1 seek
  - b_r denotes number of blocks containing records from relation r

Linear search can be applied regardless of

- Selection condition
- Ordering of records in the file
- Availability of indices

Basic Linear Scan

However, a full-table linear scan on extremely-large tables can be a disaster

- E.g., billions of records in database

- That's why we need other optimized ways

Index Scan

Index scan – Search algorithms that use an index

- Selection condition must be on search-key of index

```
select * from movies where movieid = 125;
```

We have a B+ tree index on `movieid`

- Plan: `Index Scan`

```
select * from movies where runtime = 100;
```

We don't have any index on `runtime`

- Plan: `Seq Scan`

Index Scan

Index scan – Search algorithms that use an index

- Selection condition must be on search-key of index

Unlike linear scan, we need to talk about different types of indexes and CONDITIONs

- Clustered / Non-clustered index (Primary / Secondary index)
- Equality / Comparison test

Index Scan

h_i: height of the B+-tree

Clustered index, equality on key

Retrieve a single record that satisfies the corresponding equality condition

- *key => no duplicated values*
- *Cost* = $(h_i + 1) * (t_T + t_S)$

Clustered index, equality on non-key

Retrieve multiple records

- *non key attributes => possible to have duplicated values*
- Records will be on consecutive blocks
  - Let b = number of blocks containing matching records
- *Cost* = $h_i * (t_T + t_S) + t_S + t_T * b$

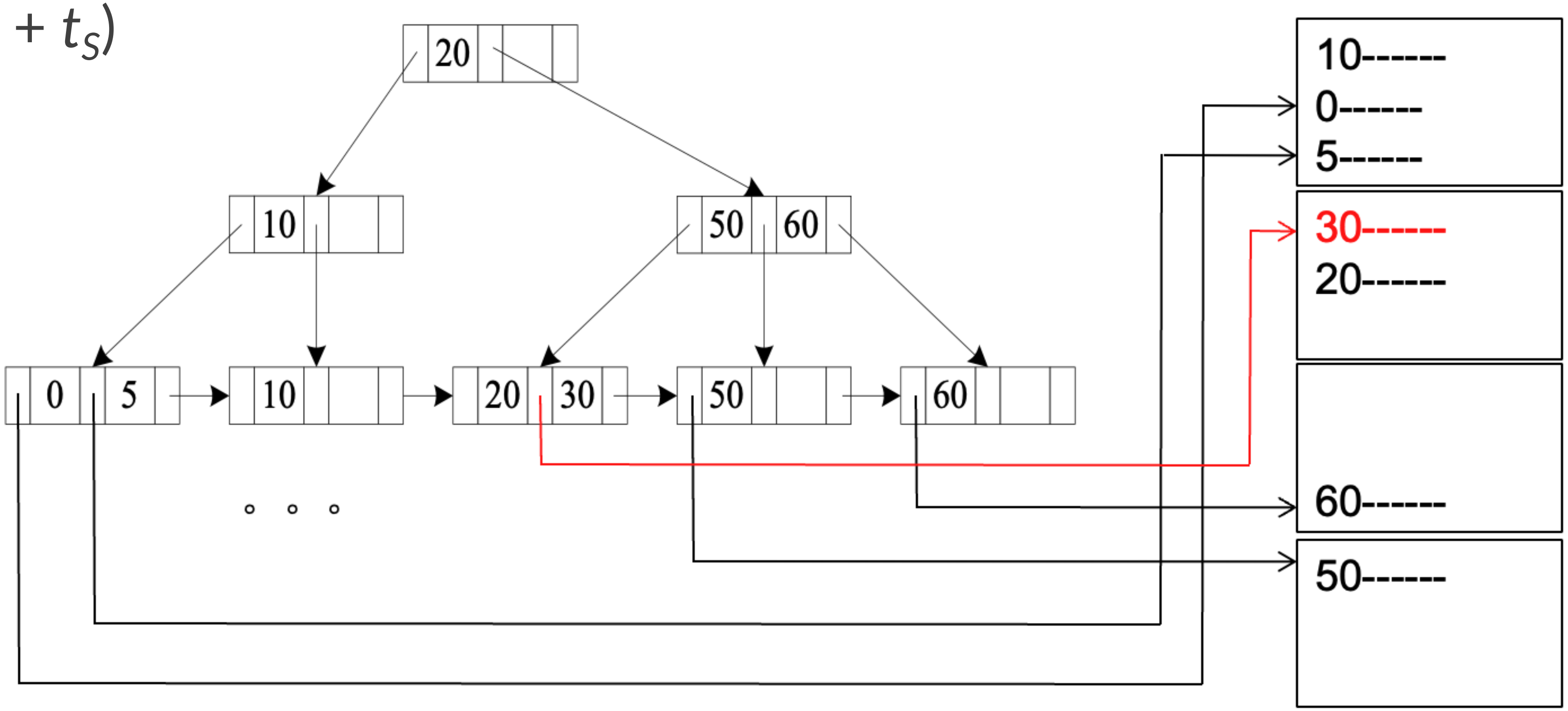
Index Scan

h_i: height of the B+-tree

Secondary index, equality on key/non-key

Retrieve a single record if the search-key is a candidate key

- $Cost = (h_i + 1) * (t_T + t_S)$

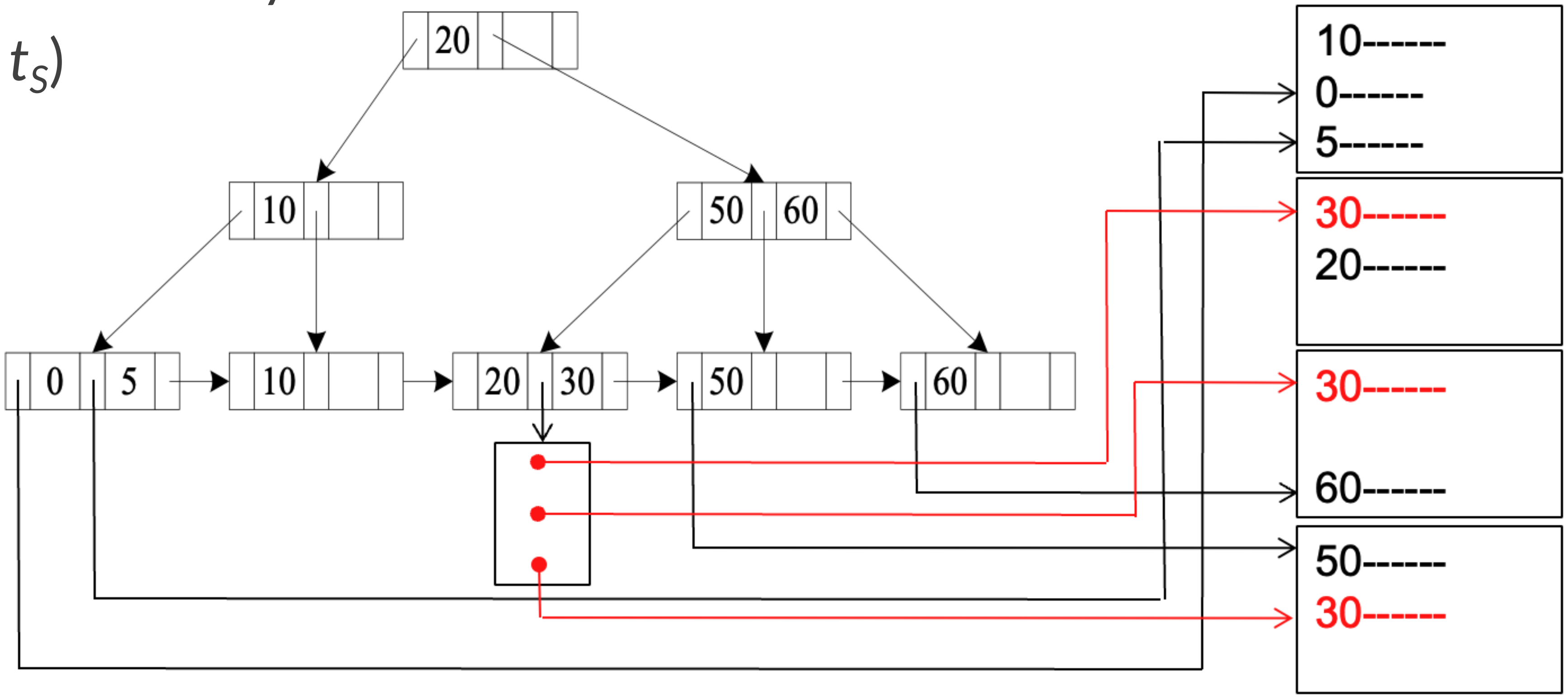


Index Scan

Secondary index, equality on key/non-key

Retrieve multiple records if search-key is not a candidate key

- Each of *n* matching records may be on a different block
- Cost = $(h_i + n) * (t_T + t_S)$
  - Can be very expensive!

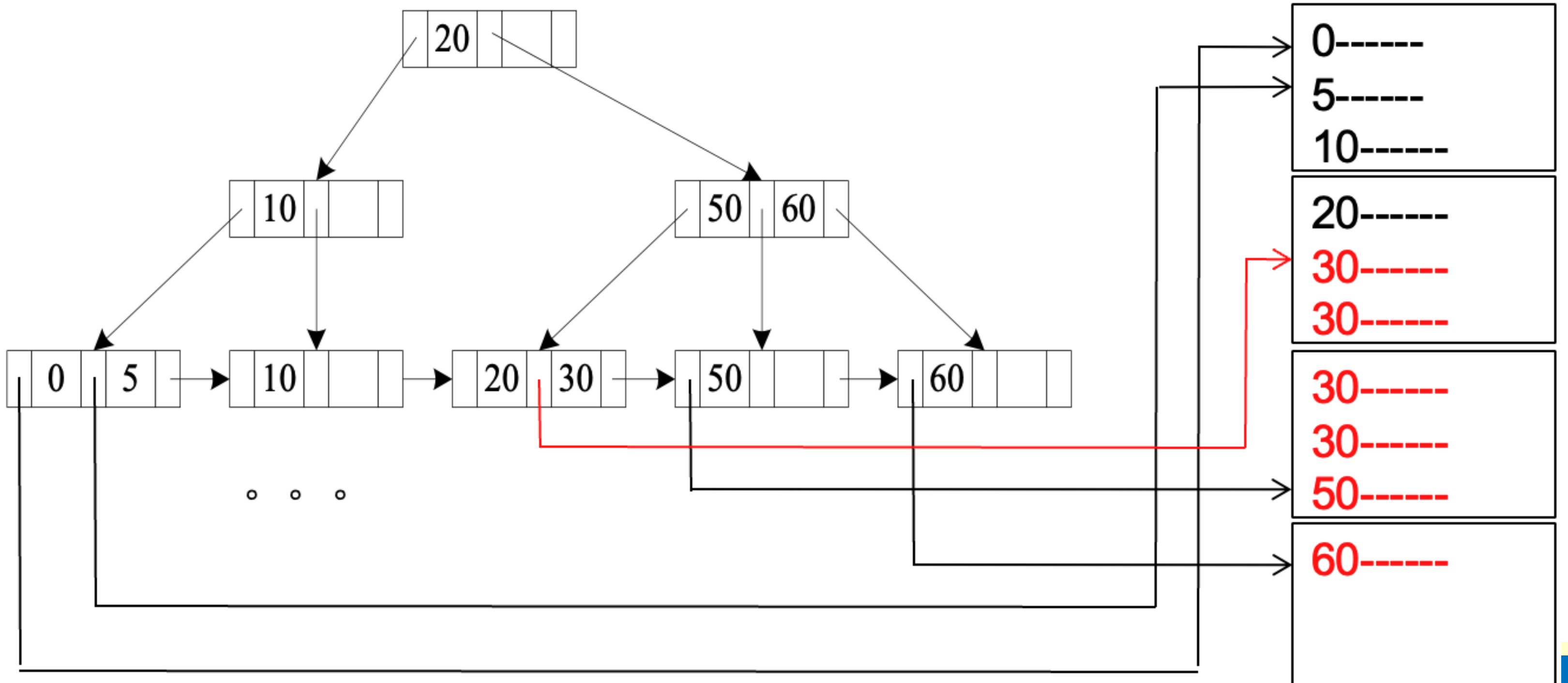


Index Scan

> *Tip:* Comparison tests can always be fulfilled with *linear scans*, which is the fallback solution

Clustered index, comparison (i.e., Relation is sorted on *A*)

- For $\sigma_{A \geq V}(r)$, use index to find first tuple $\geq v$ and scan relation sequentially from there
- For $\sigma_{A \leq V}(r)$, just scan relation sequentially till first tuple > *v*; do not use index

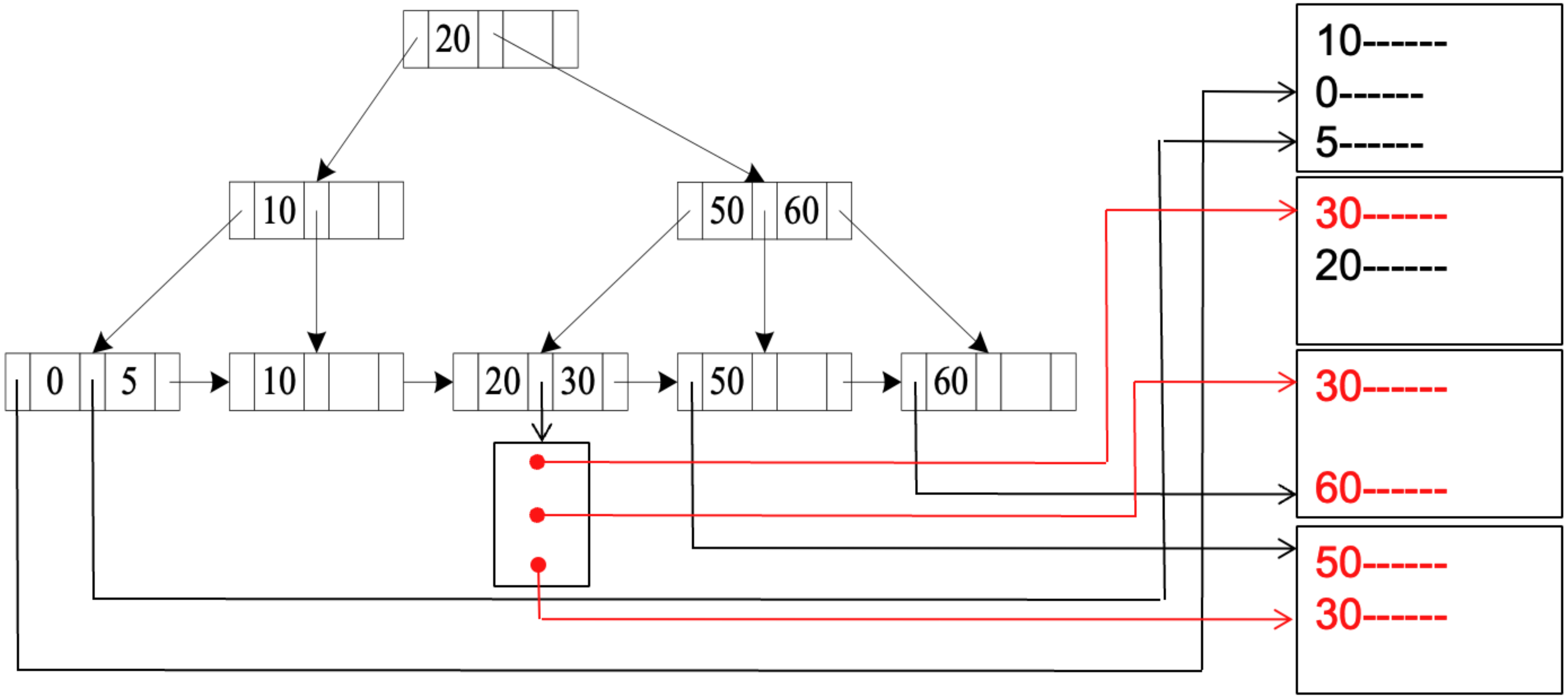


Index Scan

Non-clustered index, comparison

- For $\sigma_{A \geq V}(r)$, use index to find first index entry $\geq v$ and scan index sequentially from there, to find pointers to records.
- For $\sigma_{A \leq V}(r)$, just scan leaf pages of index finding pointers to records, till first

- In either case, retrieving records that are pointed to requires an I/O per record
- Linear scan may be cheaper!



```
select m.title, m.year_released
from movies m
 inner join credits c
 on c.movieid = m.movieid
 inner join people p
 on p.peopleid = c.peopleid
where p.first_name = 'Tim'
 and p.surname = 'Burton'
 and c.credited_as = 'D'
```

- Syntax ✓
- Do tables exist? ✓
- Right to access? ✓
- Do columns exist? ✓
- Indexes we can use? Best way to access data? ✓

One way to improve efficiency is to keep data dictionary information (meta-data) in a shared cache to avoid additional queries.

Kept in memory

shared

meta-data

```sql
select m.title, m.year_released
from movies m
 inner join credits c
 on c.movieid = m.movieid
 inner join people p
 on p.peopleid = c.peopleid
where p.first_name = 'Tim'
 and p.surname = 'Burton'
 and c.credited_as = 'D'
```

Syntax ✓

Do tables exist? ✓

Right to access? ✓

Do columns exist? ✓

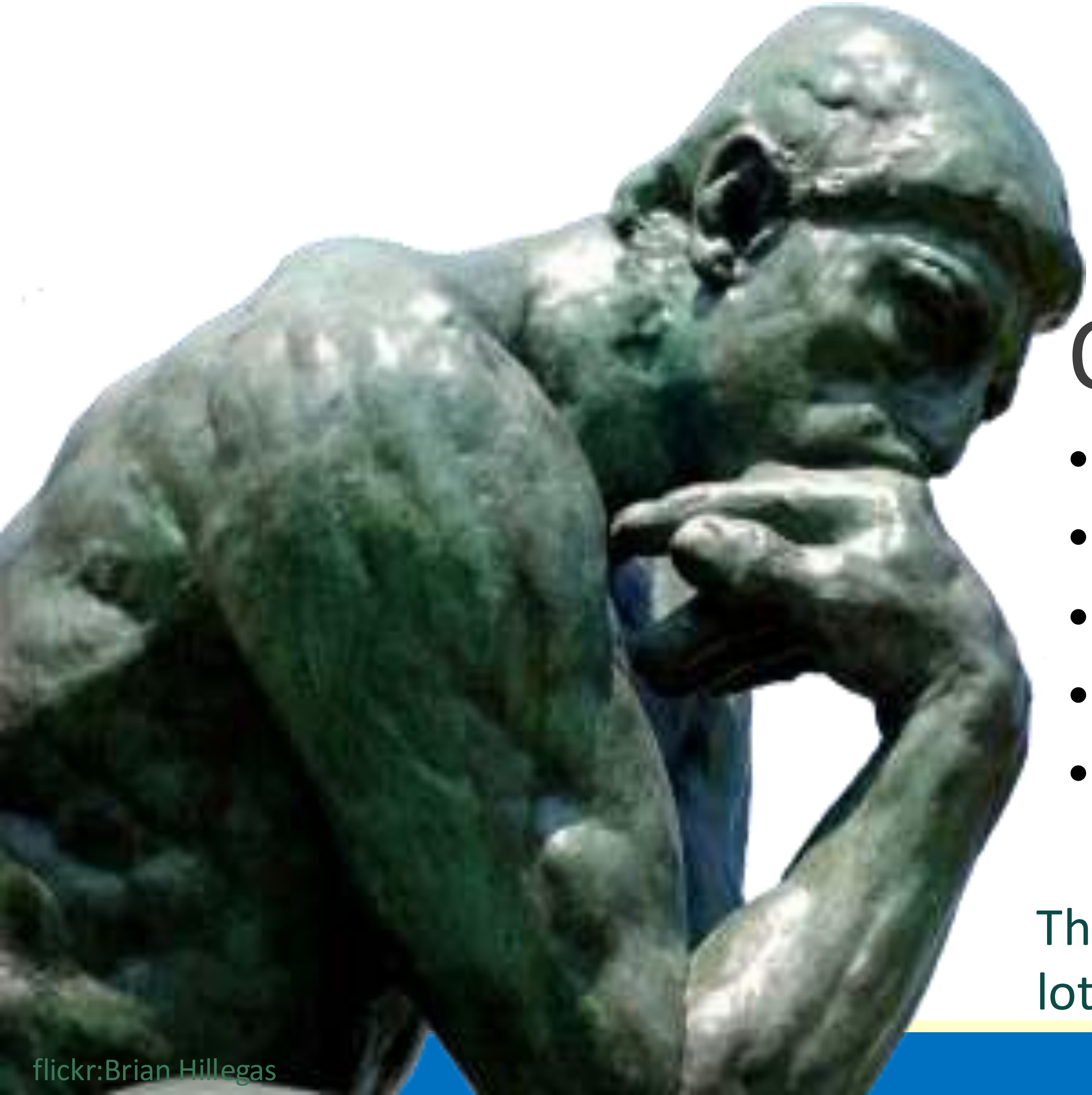
Indexes we can use? Best way to access data? ✓

Another crucial phase is the one when the optimizer tries to determine the most efficient way to access data.

Kept in memory

shared

meta-data



flickr:Brian Hillegas

QUERY OPTIMIZER

- Logical transforms
- Indexes Volumes Storage
- Hardware performance
- System load
- Settings

The optimizer has to (or can) take into account a lot of factors.

As a result

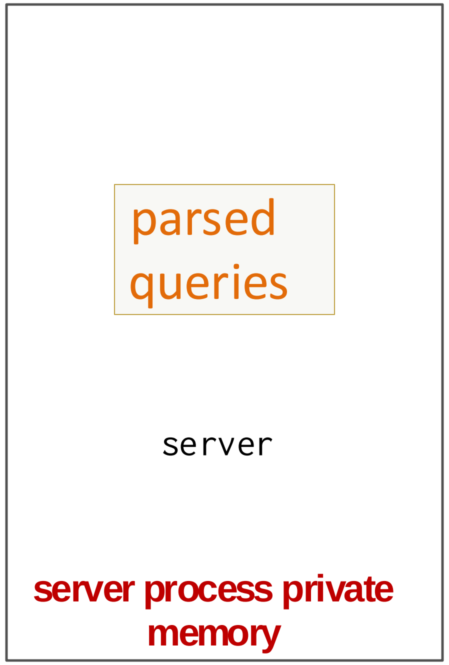
PARSING
takes time

Let's put it another way: we'd rather not parse exactly the same query many times.

How about

Using an ML/AI algorithm to predict the cost?

Keep parsed queries in memory



As most applications run exactly the same SQL statements again and again, a DBMS will cache a parsed query for reuse. For MySQL, it will be cached for a session.

Query cache management
LRU

Least Recently Used

Of course we cannot hold in cache zillions of parsed queries. We need to manage the cache, and replace queries that haven't been executed in a while with new ones.

```sql
select m.title, m.year_released
from movies m
  inner join credits c
  on c.movieid = m.movieid
  inner join people p
  on p.peopleid = c.peopleid
where p.first_name = 'Tim'
 and p.surname = 'Burton'
 and c.credited_as = 'D'
```

Checksum

We primarily recognize identical queries by computing a text checksum.

+ check tables are same and context identical

Query Compilation

Especially for main-memory database

To avoid the overhead due to interpretation

To compile the query plans into machine code or byte-code

Up to 10x faster

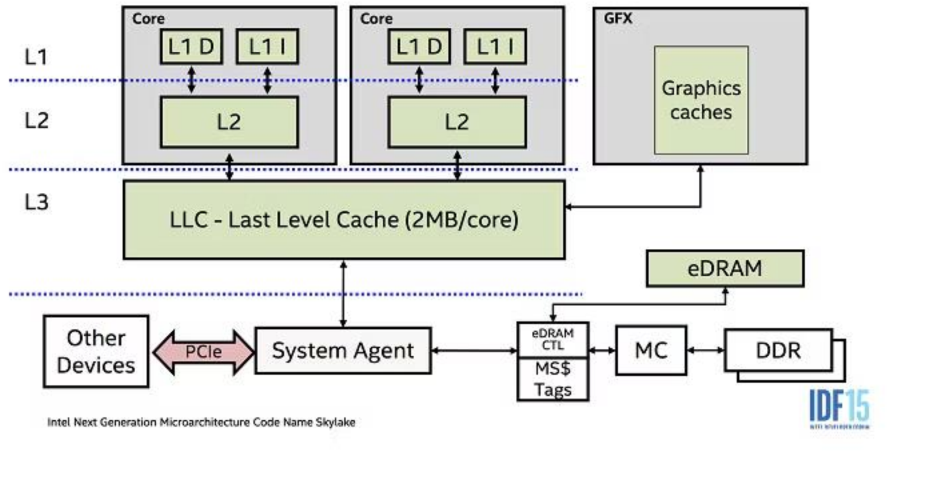
Cache-Conscious Algorithms

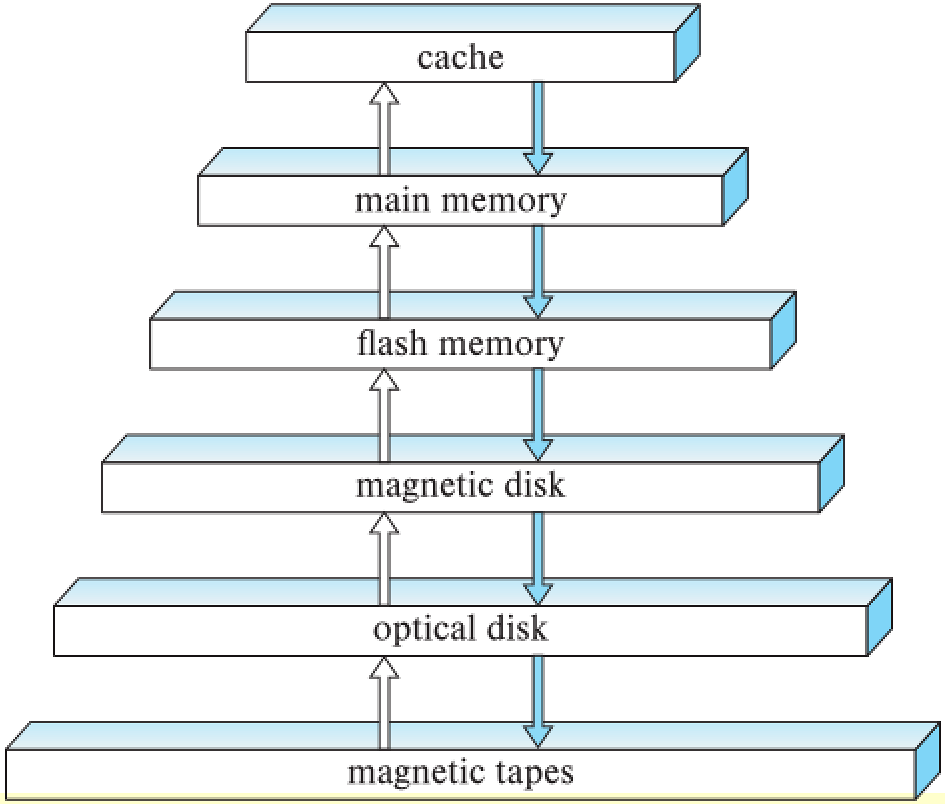
Cache, 100x faster than the main memory

13th Generation Intel® Core™ i9 Processor i9-13900K

- L1: 2.1MB, ~1ns
- L2: 32MB, ~5ns
- L3: 36MB, ~50ns

Main memory: 1-256GB, ~100ns or more





Some Cache-Conscious algorithms

* Put some sorting algorithms in L3

* Hash-join: partitioning the relations into smaller pieces to fit in the cache

* Arranging the attributes in a row consecutively, including some frequently used aggregations

For data larger than the cache, the algorithms should load the data into the cache from memory, and improve the cache hits

There are a lot of things to do for optimization because the resource is limited even we have a powerful server.

13.2 SCALING

Shiqi Yu 于仕琪

yusq@sustech.edu.cn

Most contents are from Stéphane Faroult's slides

SCALING UP

For many years, the answer to a database outgrowing the processing power of its server has been to replace the server by a bigger server.

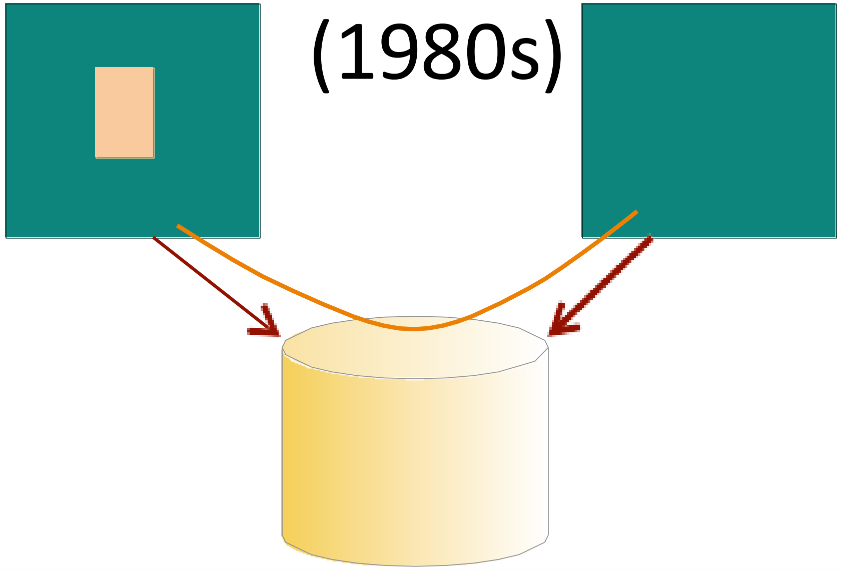
Having a single server is taking a risk if it breaks, and additionally migrating to another server is disruptive.

SCALING OUT

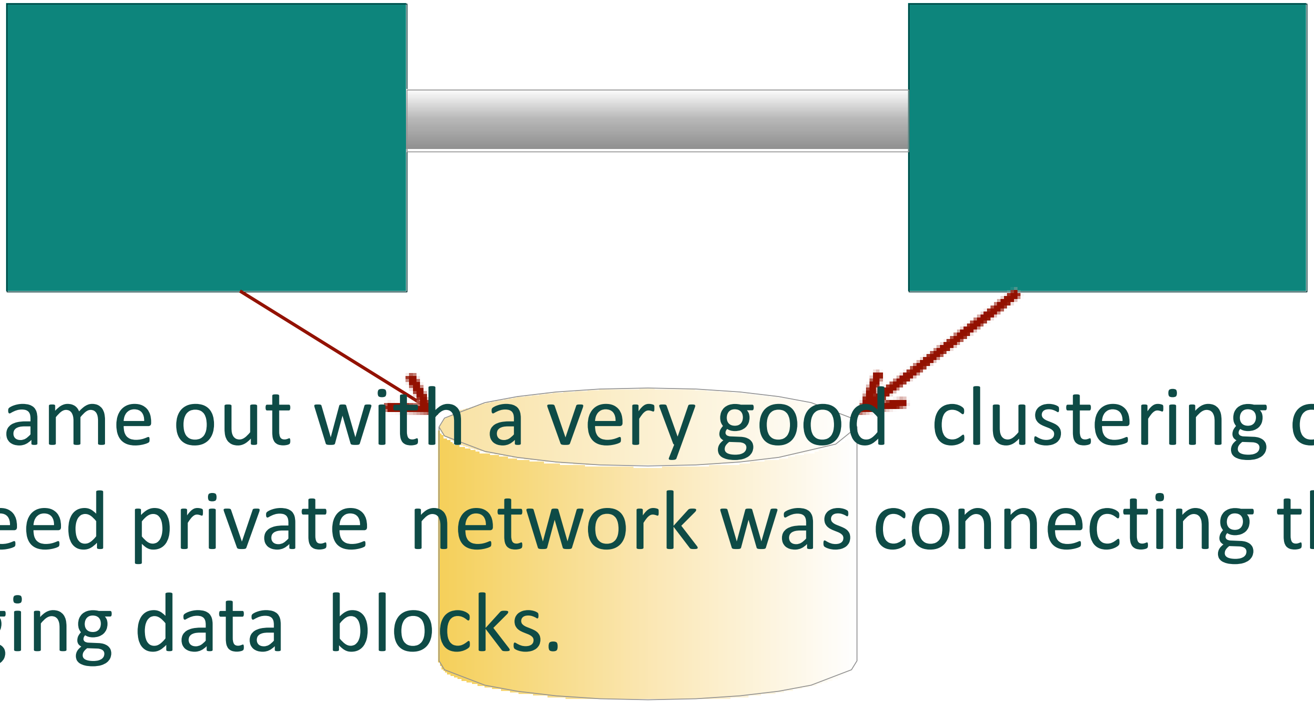
This is why people quickly thought of an alternative, adding more servers and making them share the load.

In the 1980s, Oracle tried connecting multiple servers to a single database. Complete disaster when the two servers wanted to modify the same data (or simply when one wanted to see what had just been modified by the other), data had to transit via the files.

Oracle Parallel Server
(1980s)



Oracle RAC
(2001)
Real Application Clusters



Oracle came out with a very good clustering offer, in which a very-high-speed private network was connecting the servers for exchanging data blocks.

Neil Gunther’s

Universal Law of Computational Scalability

Relative capacity C(*N*) of a computational platform:

www.perfdynamics.com

$$C(N) = \frac{N}{1 + \alpha (N-1) + \beta N (N-1)}$$

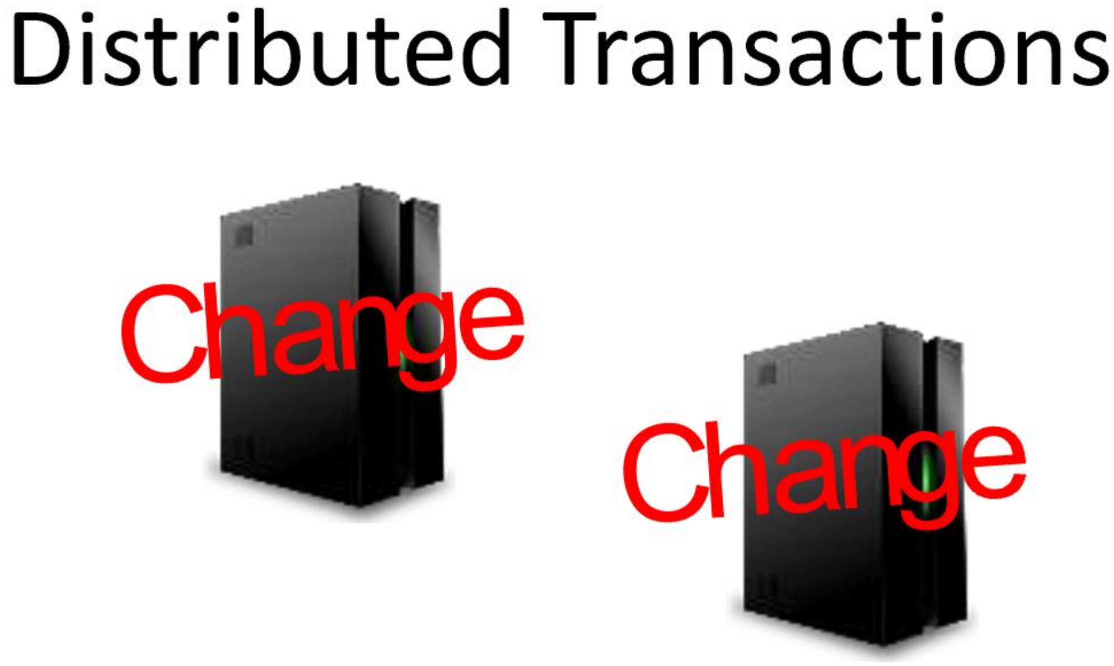
$$0 <= \alpha, \beta < 1$$

α Level of contention

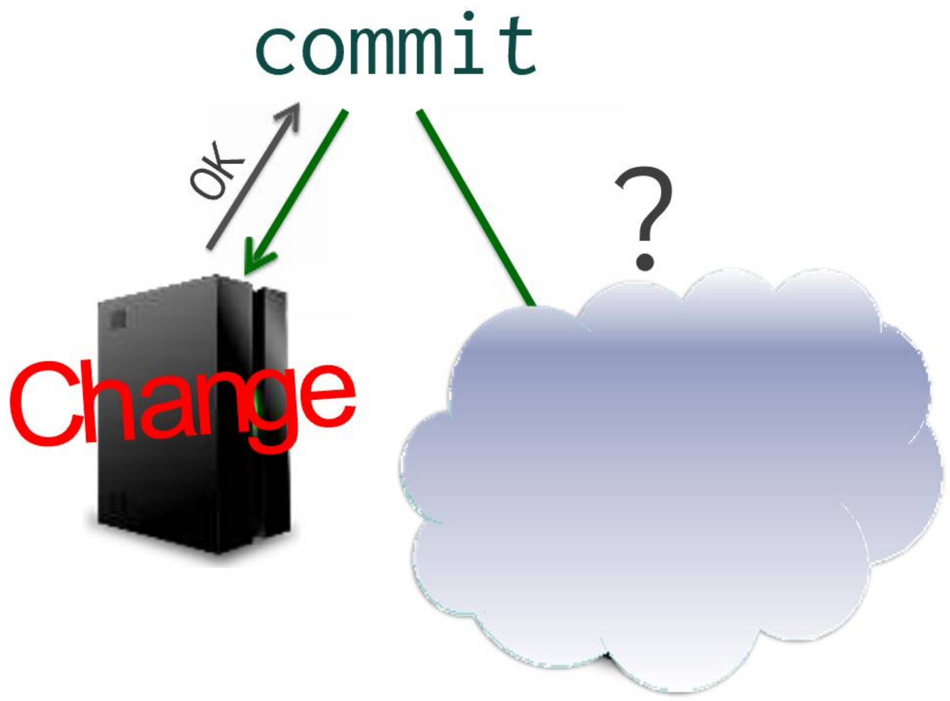
β Coherency delay (latency for data to become consistent)

The problem with clustering is that it follows a law of diminishing returns: adding a second server will less than double your capacity, and in practice people have clusters of 2, 3 or 4 machines at most (Neil Gunther is a famous Australian consultant/academic specializing on performance)

One big problem is with transactions that involve several servers. Remember that transactions are meant to be atomic operations.



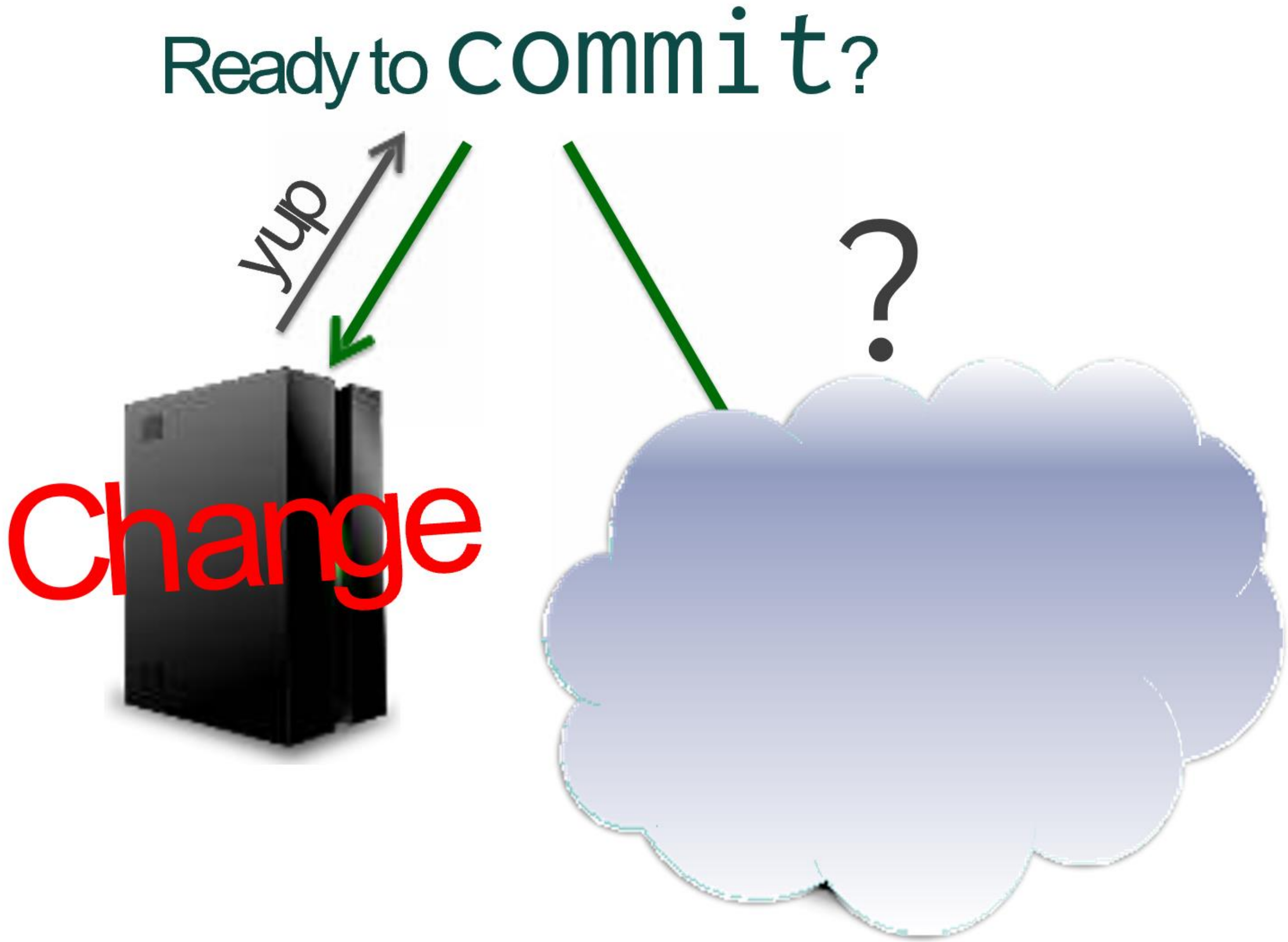
It may happen that when we commit we know for sure it worked on one server, and we get no acknowledgment from the other.



No way to rollback where it's committed, and we don't know if the other node failed before or after having committed the change.

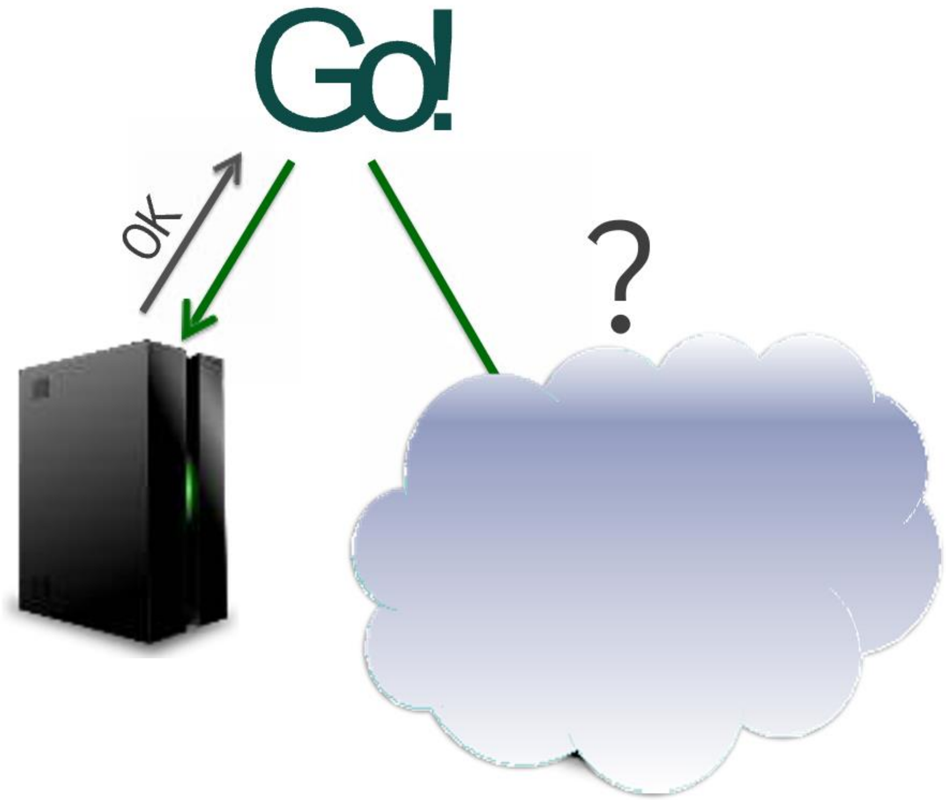
TWO-PHASE COMMIT

One algorithm was devised (a long time ago), called a "two-phase commit".



JUST before committing, you ask to all servers involved whether they are ready to commit. If you don't get an all clear, you can rollback.

Otherwise, you can then (2nd phase) send the official "commit" signal



Odds that something will fail are far lower than with a single-phase commit, but not zero.

Latency

Additionally, you have latency issues. All machines in a cluster may not be sitting next to each other, they may be a few miles apart in different data centers for security reasons (fire, flood ...)

1 KM = 0.0033ms
2000 KM = 6.6ms

Distance from Shenzhen to Beijing

Even if information travels fast, multiplying exchanges (two-phase commit) may become a sensitive issue.