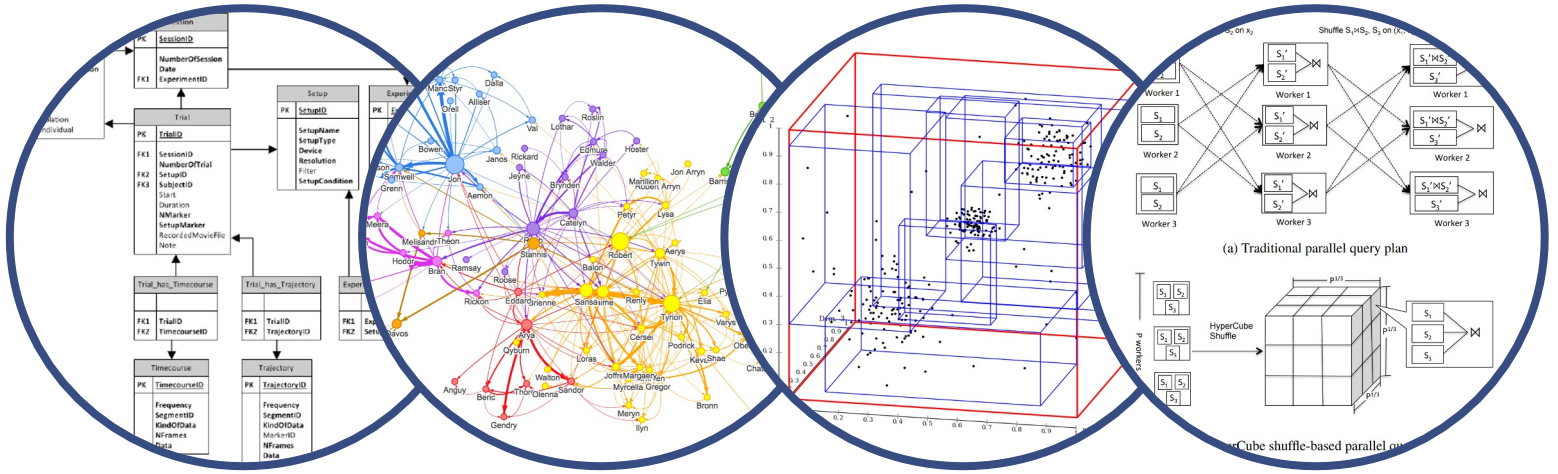


Introduction to Data Management

Parallel Processing

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

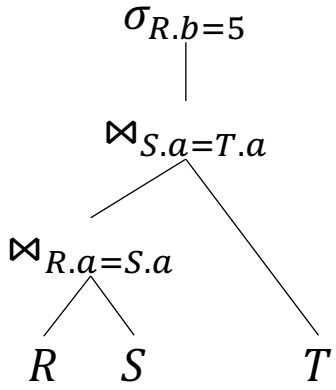
Outline

- **Multiple Operation Execution**
- Full Query Cost Estimation
- Parallelizing Data Management
  - Problems
  - Concepts
    - Performance Expectations
    - Parallelism Flavors
- Parallelism Examples

Executing Multiple Operations

- **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!

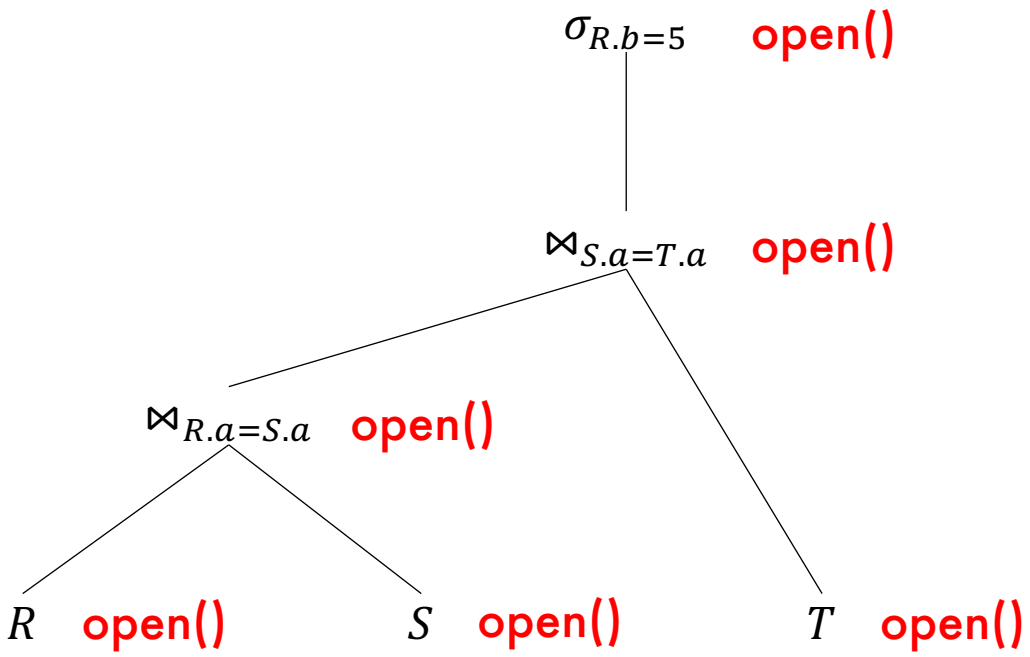


Pipelined Execution

- 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

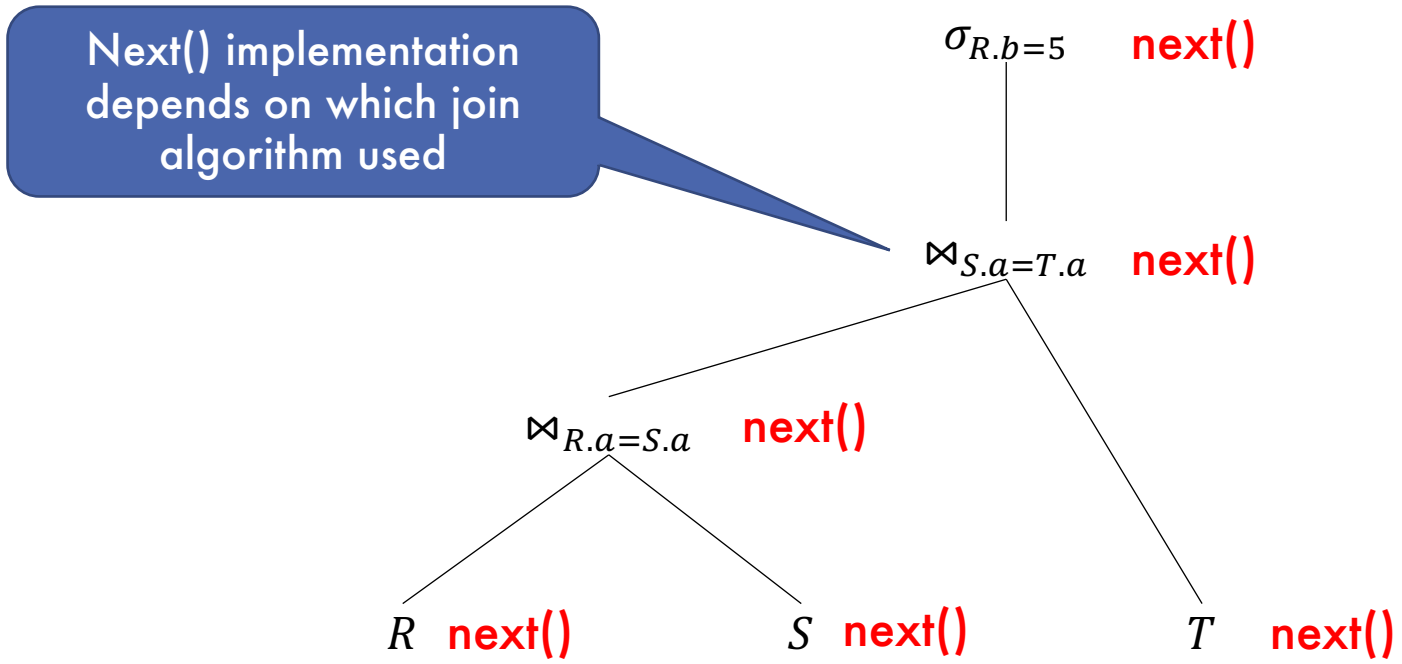
Pipelined Execution Example

```
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
```



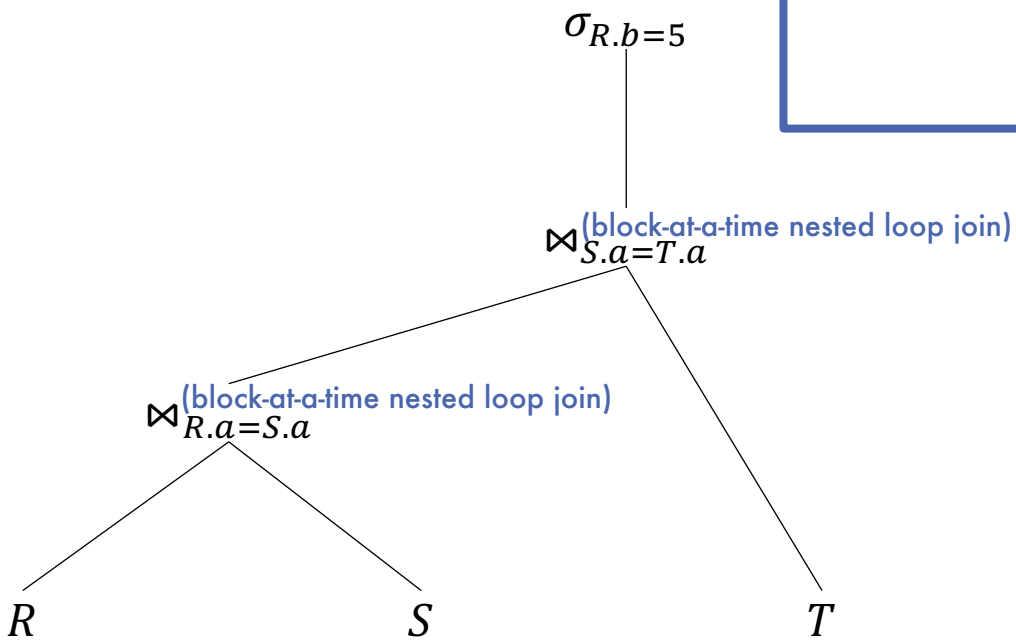
Pipelined Execution Example

```
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!

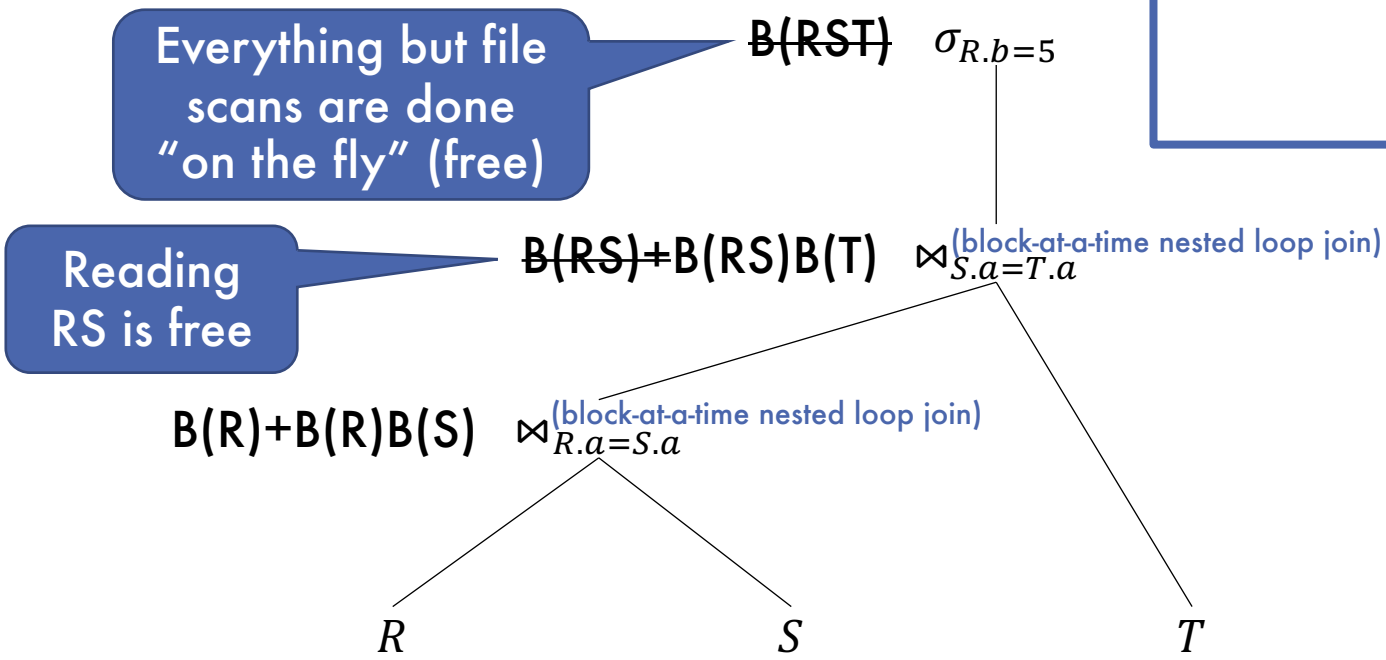
Est. Cost =



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

- Multiple Operation Execution
- **Full Query Cost Estimation**
- Parallelizing Data Management
  - Problems
  - Concepts
    - Performance Expectations
    - Parallelism Flavors
- Parallelism Examples

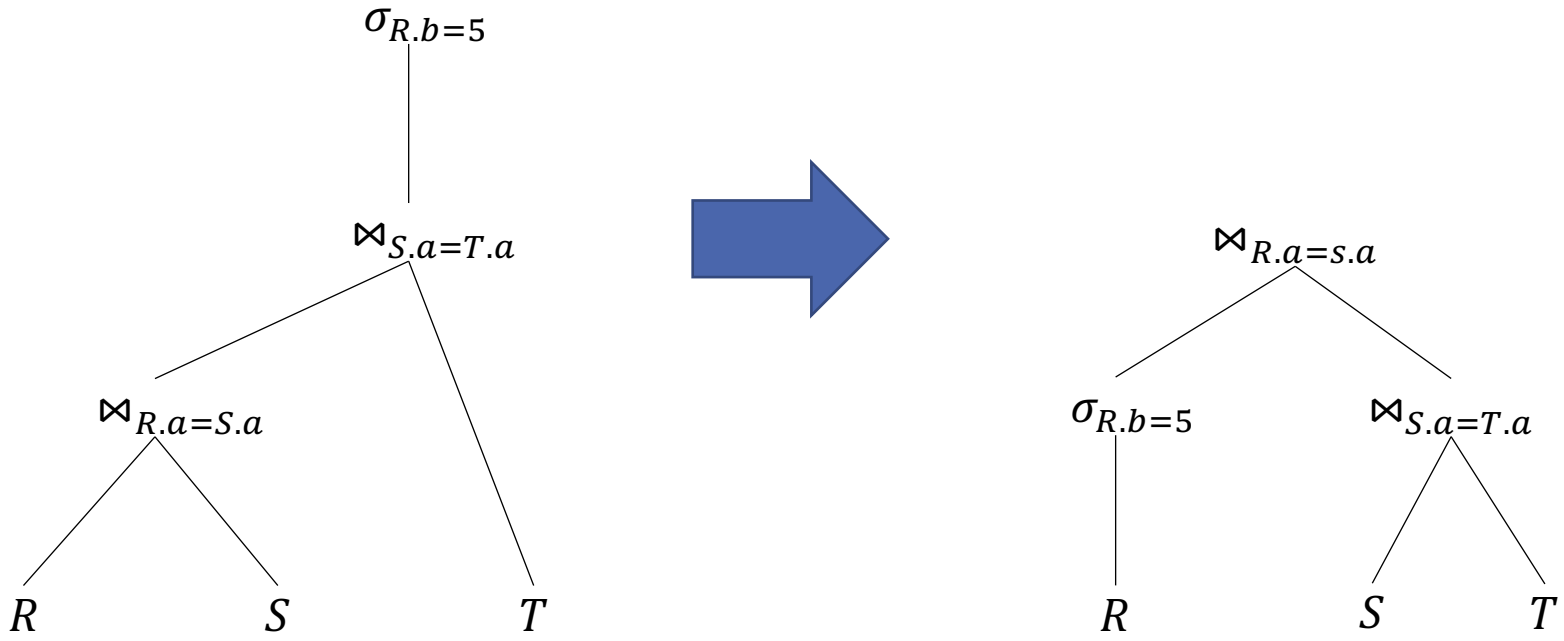
Cost Reduction Strategies

- 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*

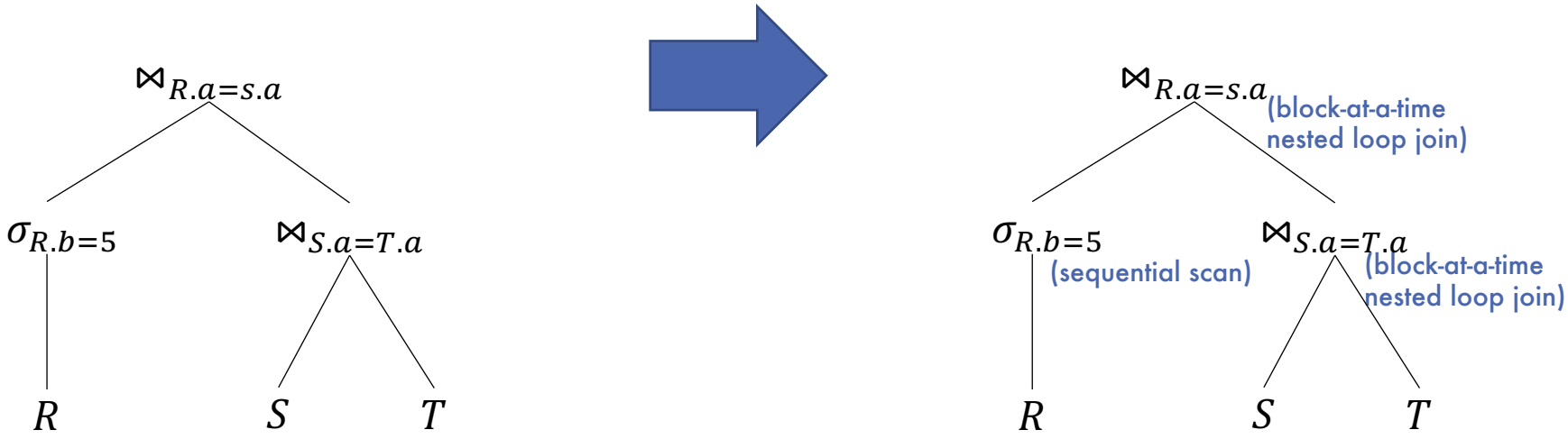
```
SELECT *
FROM   R, S, T
WHERE  R.a = S.a
AND    S.a = T.a
AND    R.b = 5
```



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

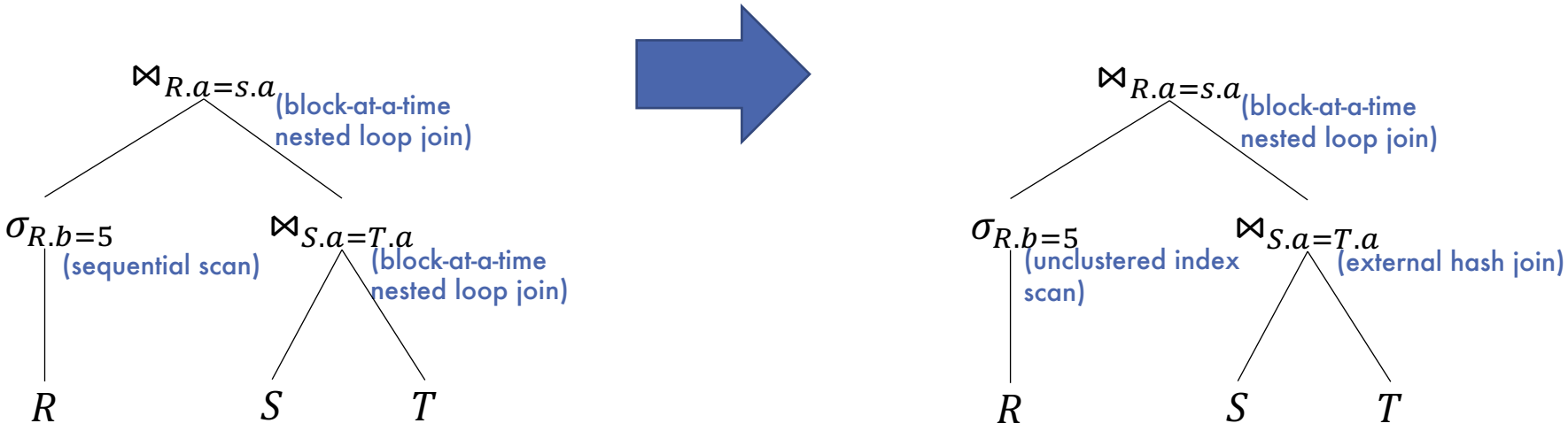


Full Plan Cost Estimation

- *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*

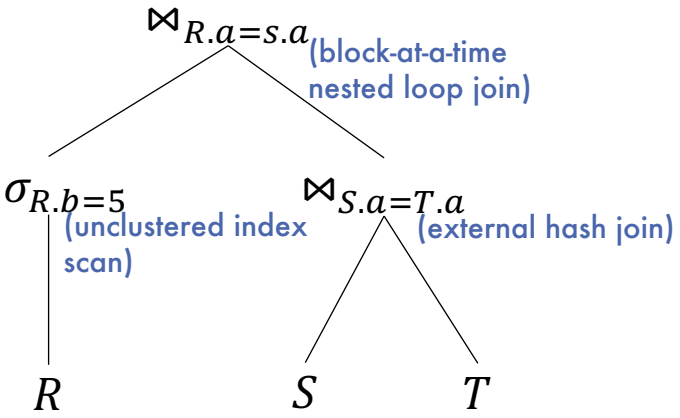


Full Plan Cost Estimation

- *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 =

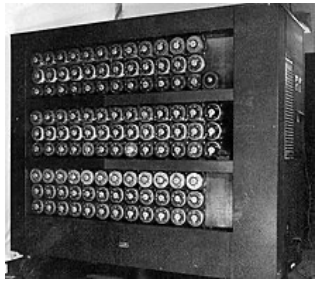


Outline

- Multiple Operation Execution
- Full Query Cost Estimation
- Parallelizing Data Management
  - **Problems**
  - Concepts
    - Performance Expectations
    - Parallelism Flavors
- Parallelism Examples

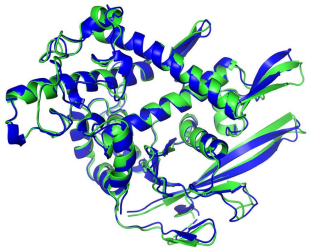
You Must Construct Additional Pylons

- Humans tend to tackle problems that are too big to compute:
  - Breaking the WWII Enigma code
    - Using automation (the bombe)



  - Computing rocket trajectories (Space Race)
    - Using programming languages (FORTRAN)

  - Now: Data driven applications
    - Protein folding
    - Internet of things
    - Financial forecasting
    - Weather prediction
    - Social media platforms
    - …



More Data, More Problems

- The rates at which we generate and use information have **outpaced the capabilities of a single computer**

- Problems:
  - Need more speed
  - Need more scale

Two Types of Workload

OLTP (Online Transaction Processing)	OLAP (Online Analytical Processing)*
No guarantees on technical or domain-specific knowledge	Data analyst or data scientist; few users, usually "insiders"
Queries typically read/update single records	Queries typically touch large portions of the data
e.g., read bank account balance, then UPDATE it	e.g., multiple JOINs followed by a GROUP-BY
Many INSERT/UPDATE/DELETE	Little to no INSERT/UPDATE/DELETE
Challenges: Correctness of updates (transactions!), performance over large userbase	Challenges: Query optimization (joins/aggregates), performance over large datasets

* Also called Decision Support workload

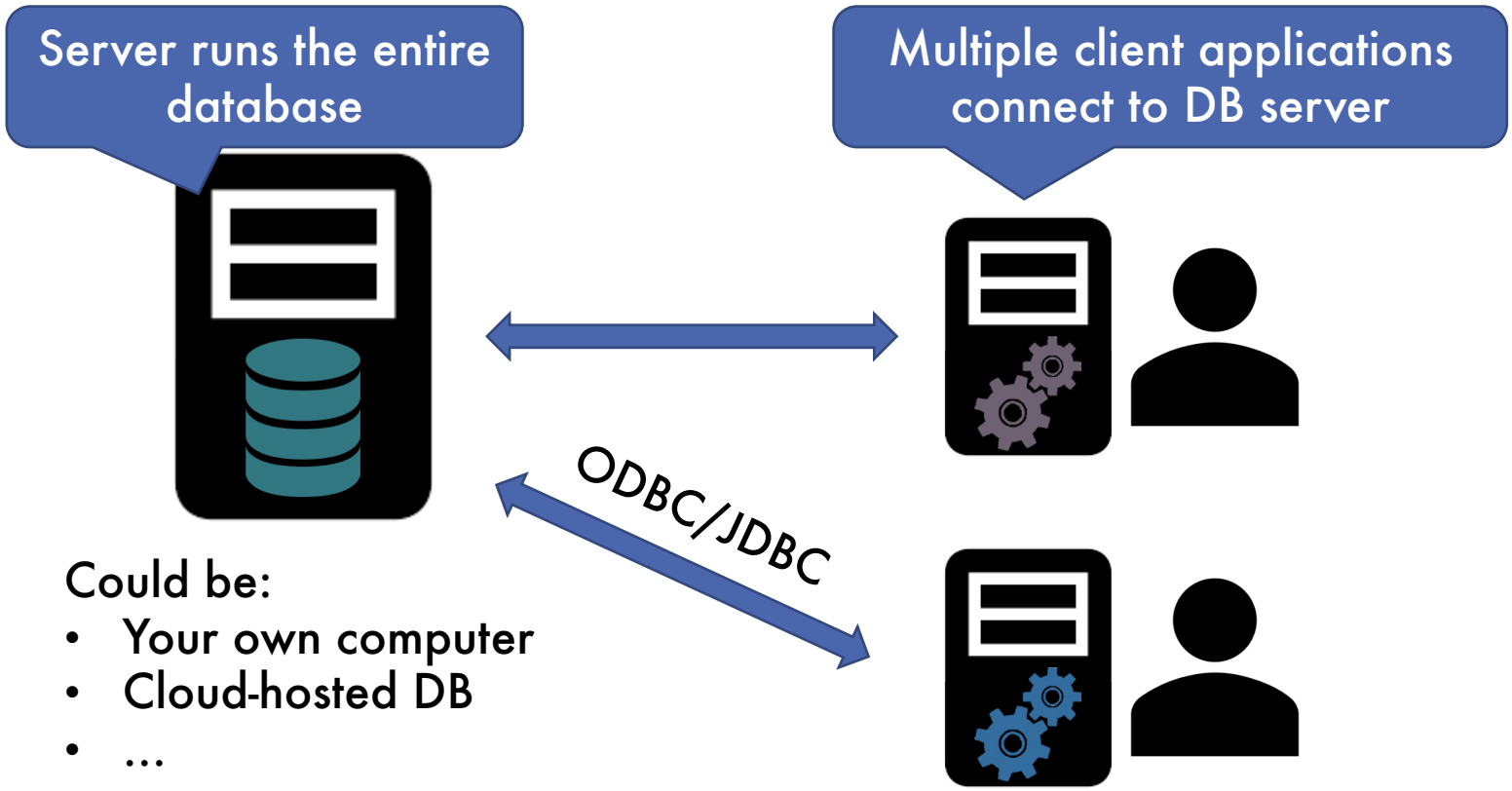
Single Client

- **Single client** means that the application and the data are on the same machine
  - Eg, SQLite, PostgreSQL
  - Eg, HW1, HW2

- Applications:
  - Mostly OLAP applications (small-scale data science)
  - OLTP for local programs

Client-Server

- A **two-tier architecture** has a server running a resource (eg, a DBMS) that is shared amongst multiple (sometimes concurrent) users



3-Tiered Architecture

- As we get even more users, how do we architect an OLTP solution?

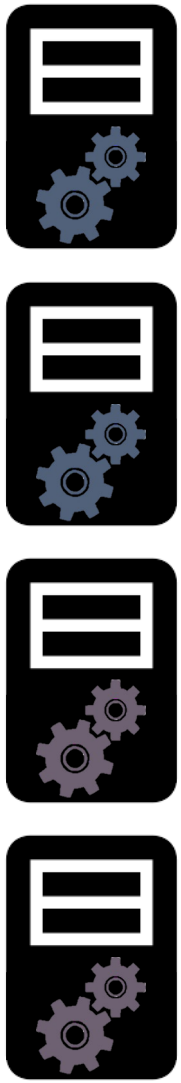
3-Tiered Architecture

Web/App servers (easily replicated for more users)

eg, Java, Python, Ruby-on-Rails

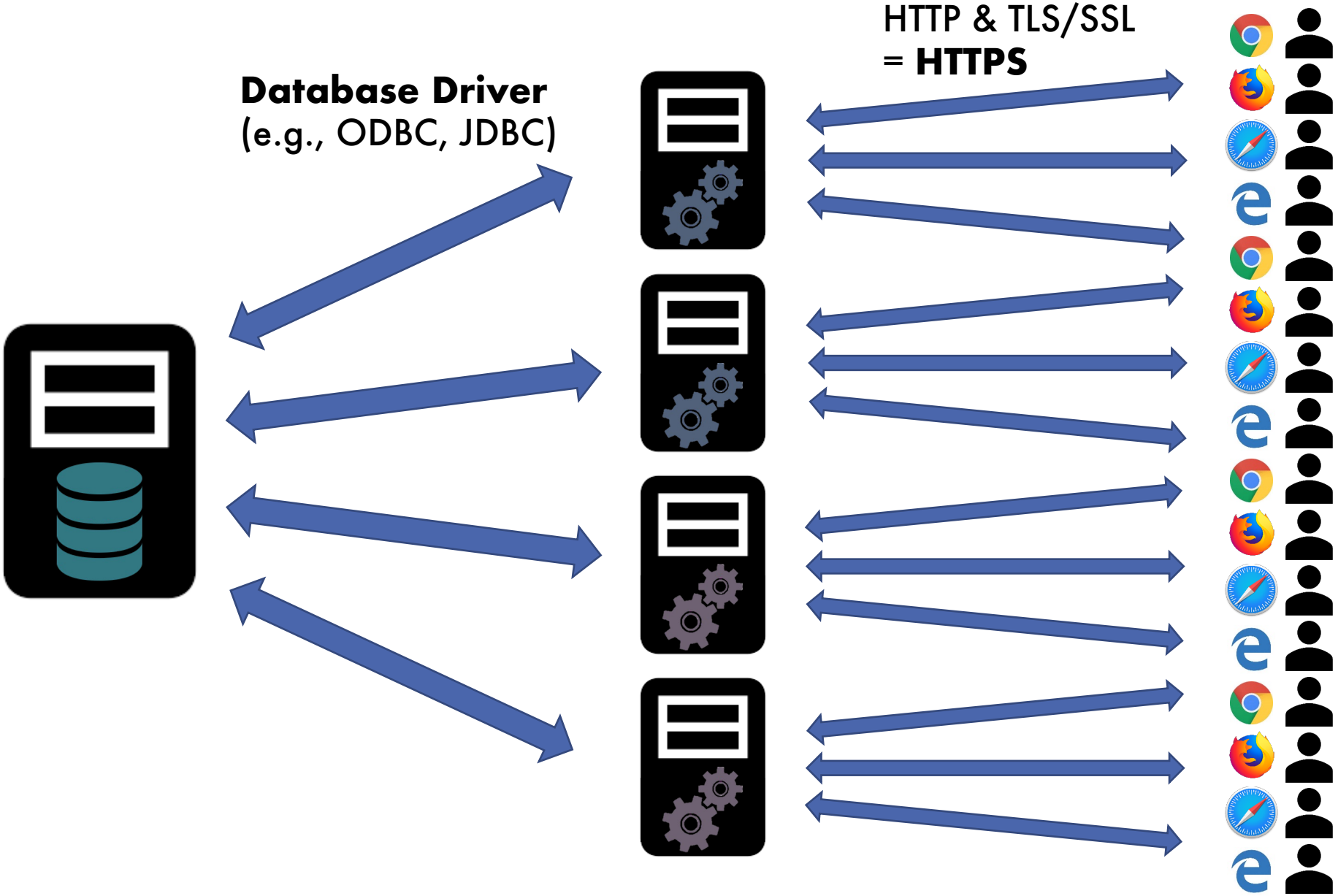
Database server

eg, SQLServer, Oracle, DB2, MySQL, PostgreSQL

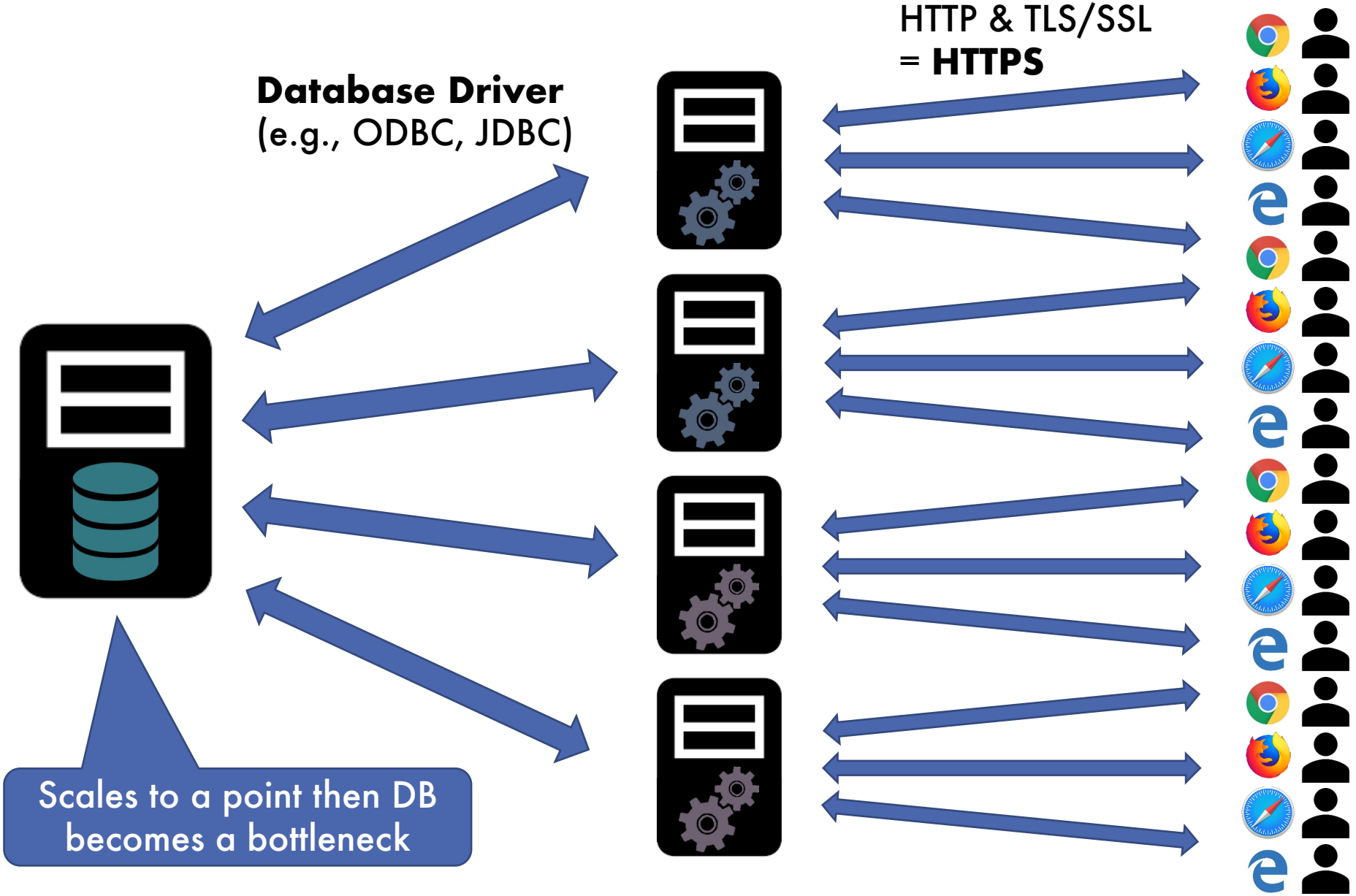


Browsers allow communication to servers

3-Tiered Architecture



3-Tiered Architecture



Big Data

- Data volume is not an issue!
  - Databases *do* parallelize easily; techniques available from the 80's
    - Data partitioning
    - Parallel query processing
  - **SQL/RA is "embarrassingly parallel"**

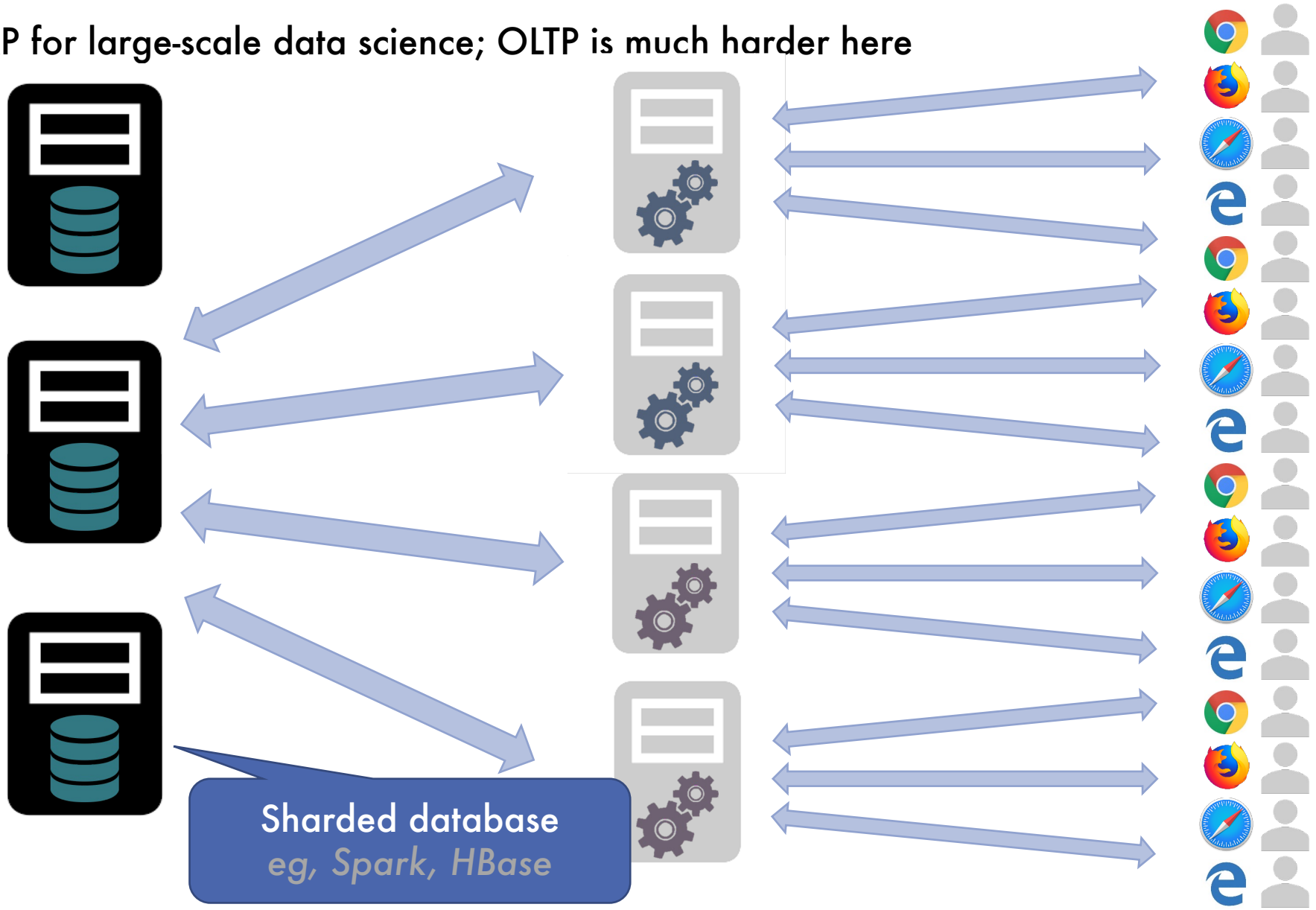
- New workloads are an issue
  - Large-scale machine learning
  - Globally consistent data views
  - …
  - (Requires innovation; active research area)

What Comes Next?

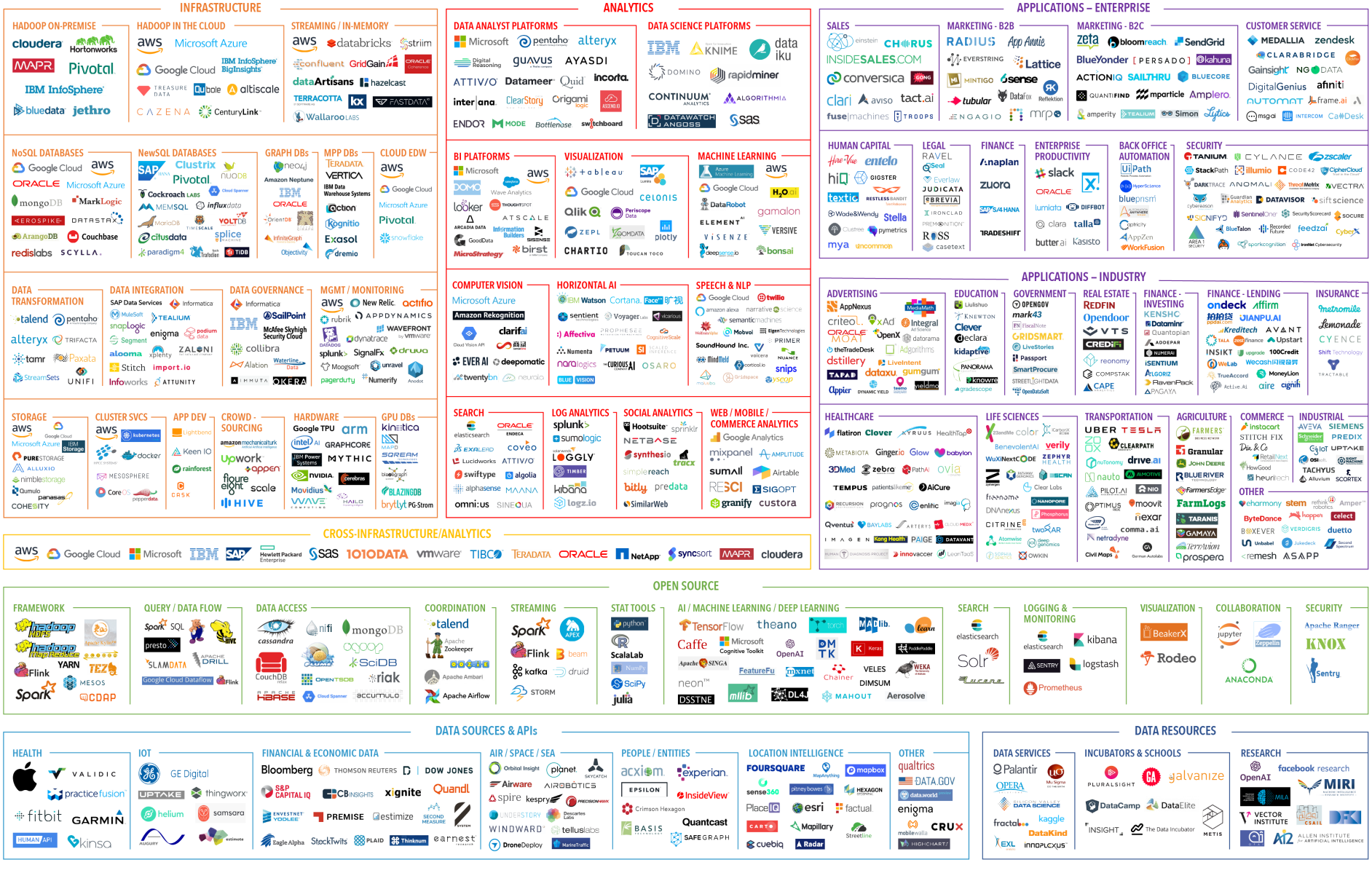
- As we get even more complex, how do we continue growing?
  - Can easily replicate read-only assets like web and application servers ("endpoints")
  - Cannot simply replicate database servers
- How to scale up the DBMS?
- How to maintain ACID principles?

3-Tiered Web Architecture

OLAP for large-scale data science; OLTP is much harder here



BIG DATA & AI LANDSCAPE 2018



Outline

- Multiple Operation Execution
- Full Query Cost Estimation
- Parallelizing Data Management
  - Problems
  - Concepts
    - **Performance Expectations**
    - Parallelism Flavors
- Parallelism Examples

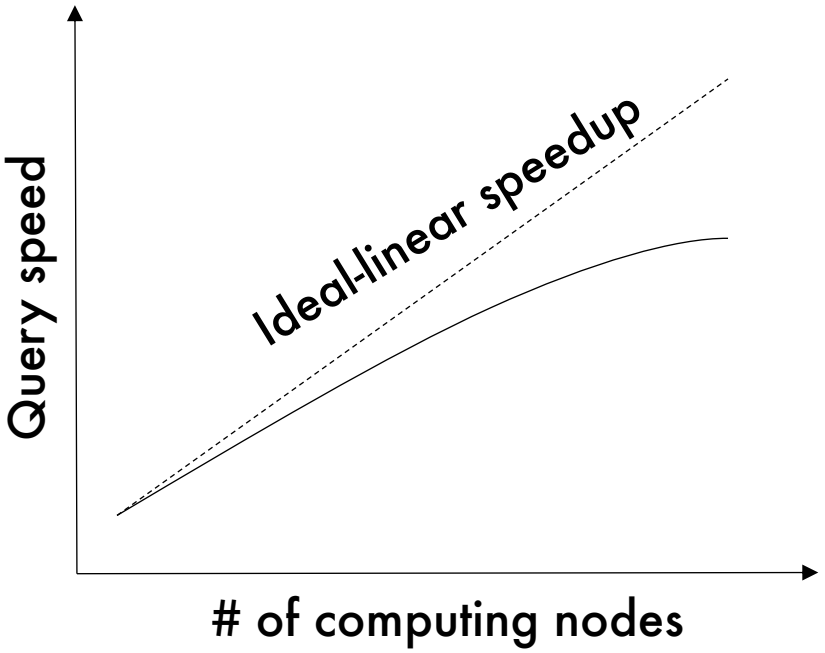
Speed Up and Scale Up

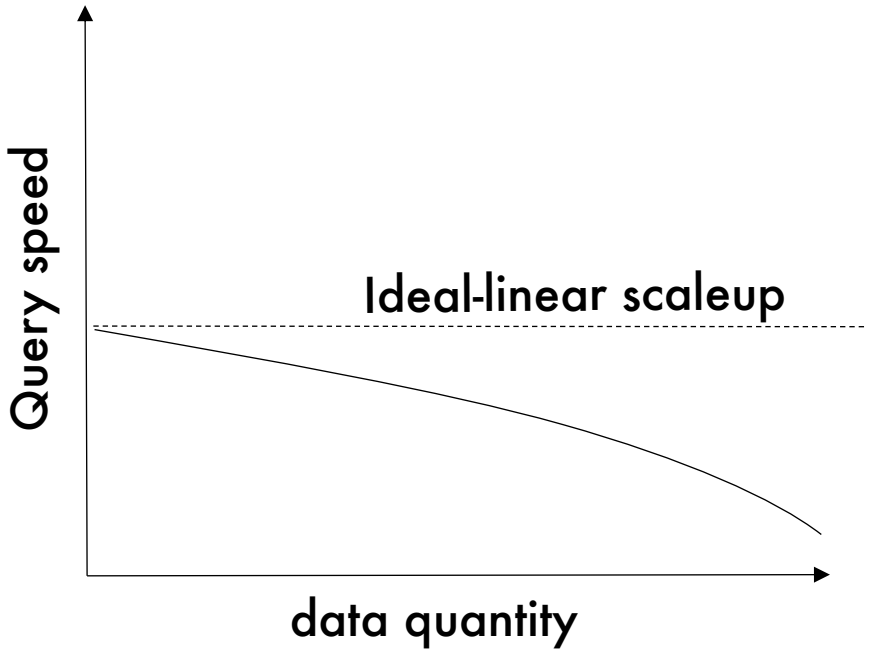
Speed up:

- Speedup = #Workers/Runtime
- *"How much faster can we get if we add more nodes?"*

Scale up:

- Scaleup = #Tuples/Runtime
- *"How much slower do we become with more data?"*





Sublinear Expected Performance

- Parallel computing is not a magic bullet

- Common reasons for sublinear performance:
  - **Overhead cost**
    - Starting and coordinating operations on many nodes
  - **Interference/Contention**
    - Shared resources are not perfectly split
  - **Skew**
    - Process is only as fast as the slowest node

Outline

- Multiple Operation Execution
- Full Query Cost Estimation
- Parallelizing Data Management
  - Problems
  - Concepts
    - Performance Expectations
    - **Parallelism Flavors**
- Parallelism Examples

Implementations for Database Parallelism

- **Architecture Parallelism**
  - Shared Memory
  - Shared Disk
  - Shared Nothing

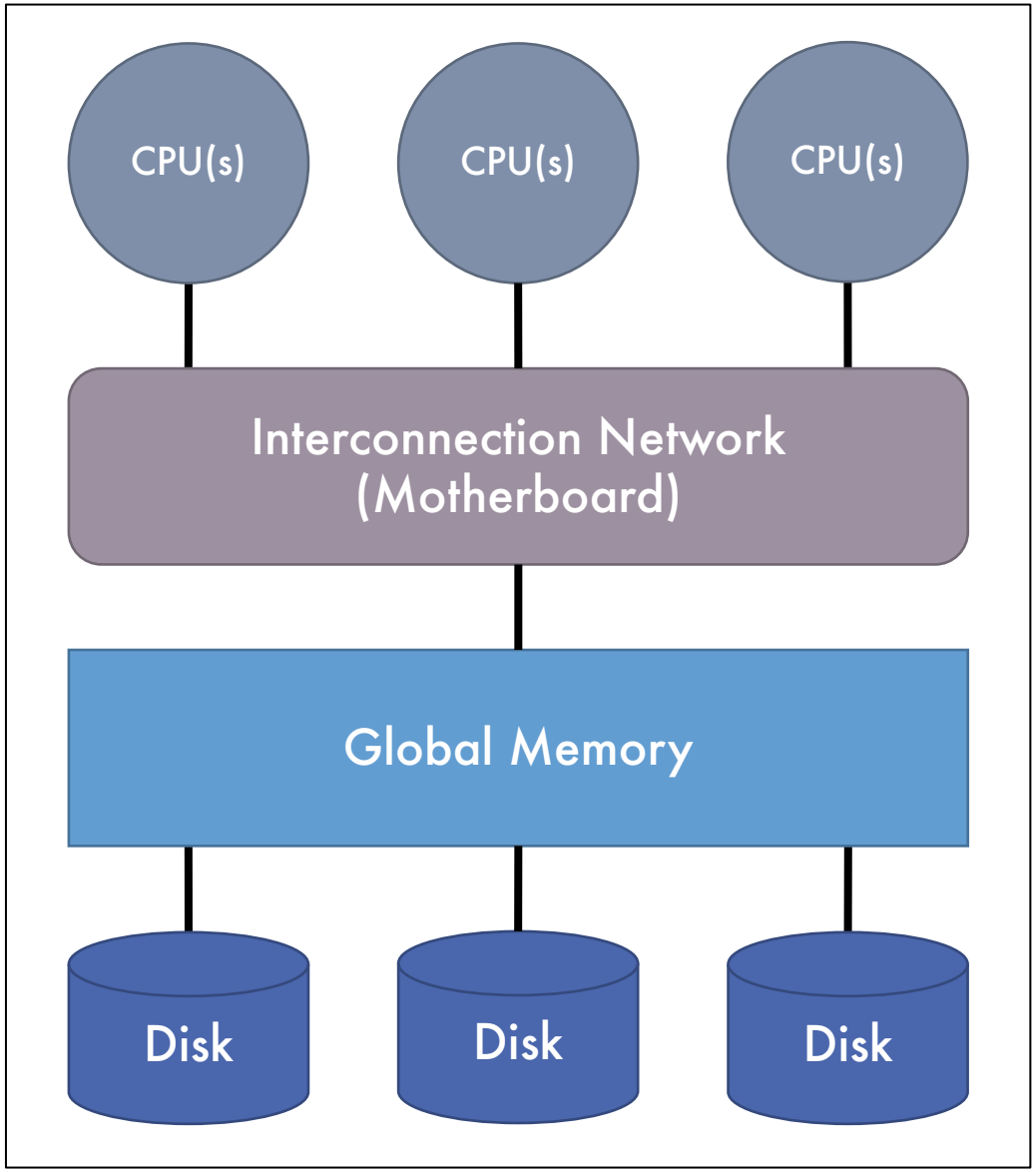
Hardware considerations

- **Query Parallelism**
  - Inter-Query Parallelism
    "parallelism between queries"
  - Intra-Query Parallelism
    "parallelism within queries"

Software considerations

- **Data partitioning**

Shared-Memory Architecture



- Shared main memory and disks
- Your laptop or desktop uses this architecture
- Expensive to scale
- Easiest to implement on

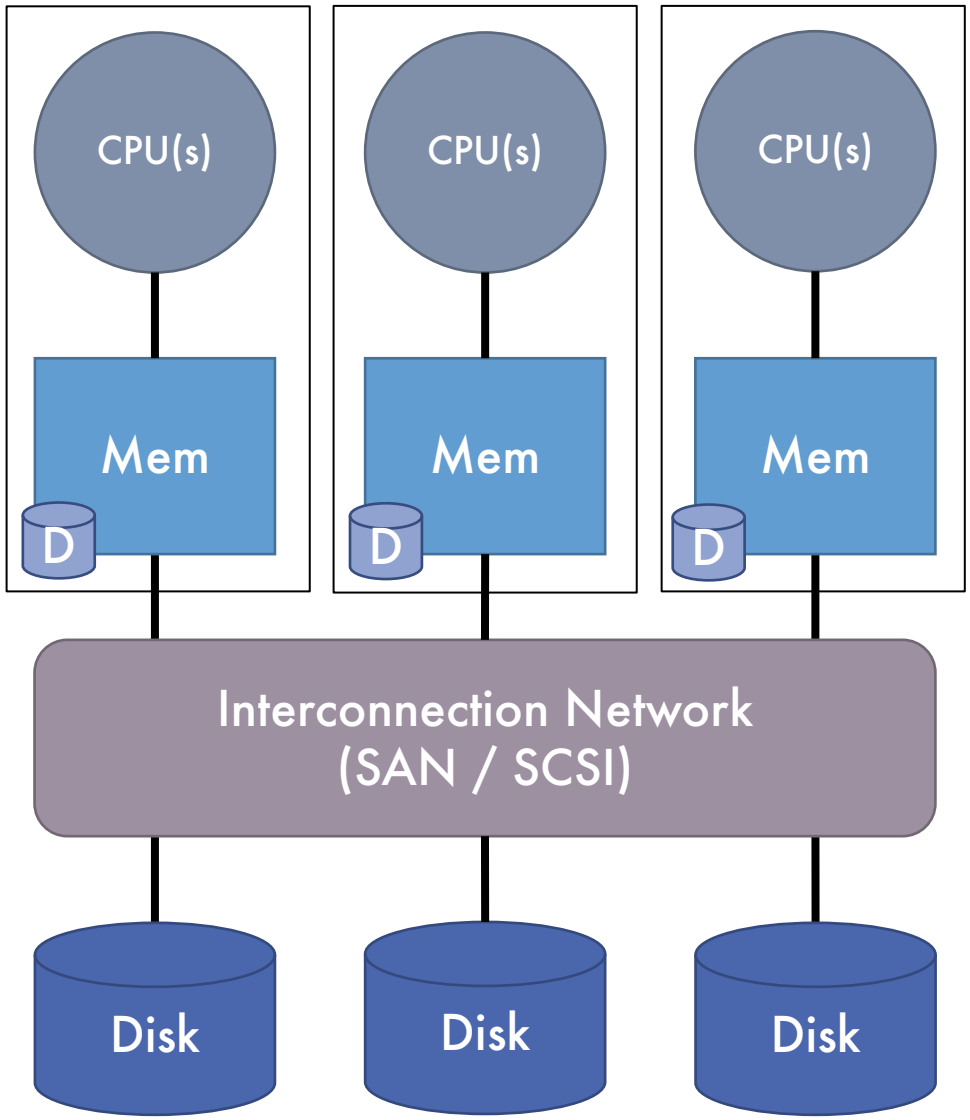
Microsoft SQL Server

PostgreSQL

SQLite

MySQL

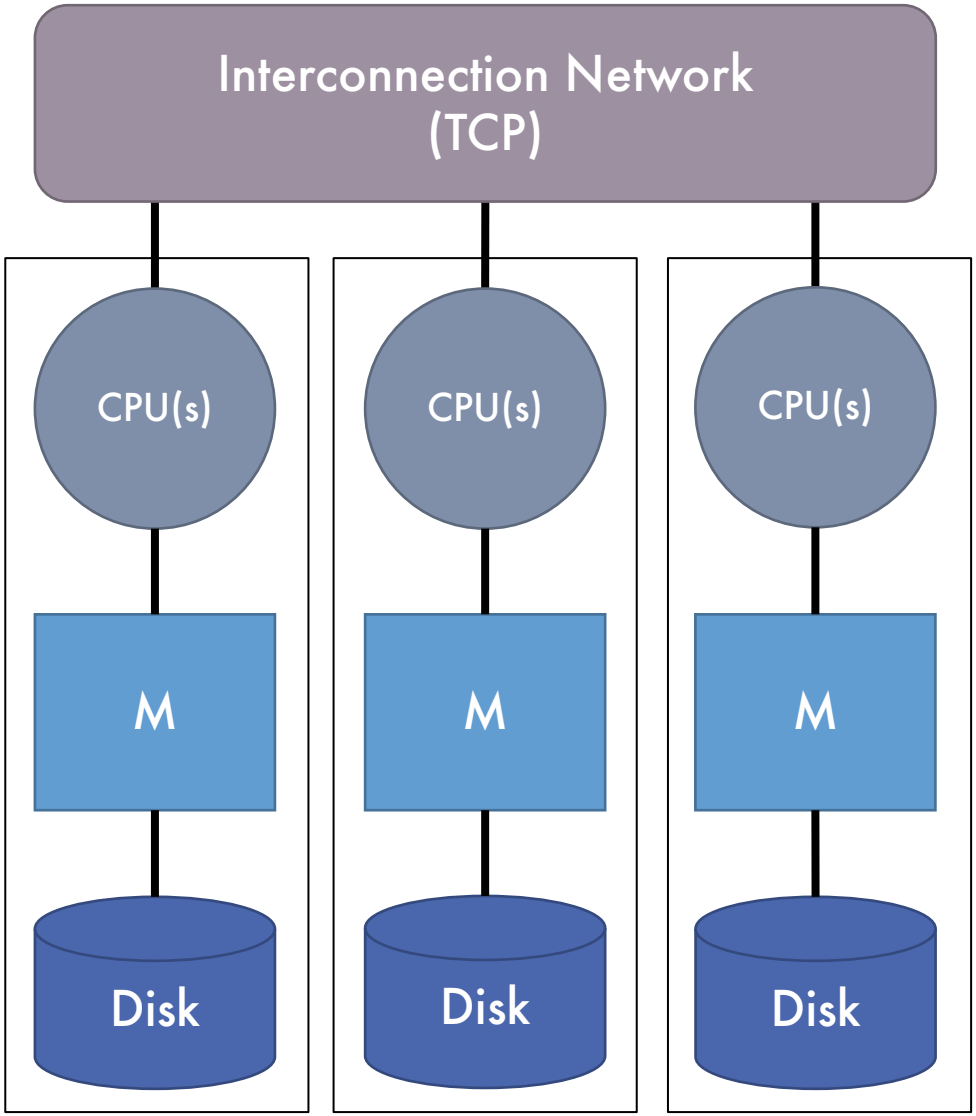
Shared-Disk Architecture



- Storage-dedicated network
- No contention for memory and high availability
- Typically 1-10 machines

ORACLE®
DATABASE

Shared-Nothing Architecture



- Uses (relatively) cheap, commodity hardware
- No contention for memory and high availability
- Theoretically can **scale infinitely**
- Hardest to implement on

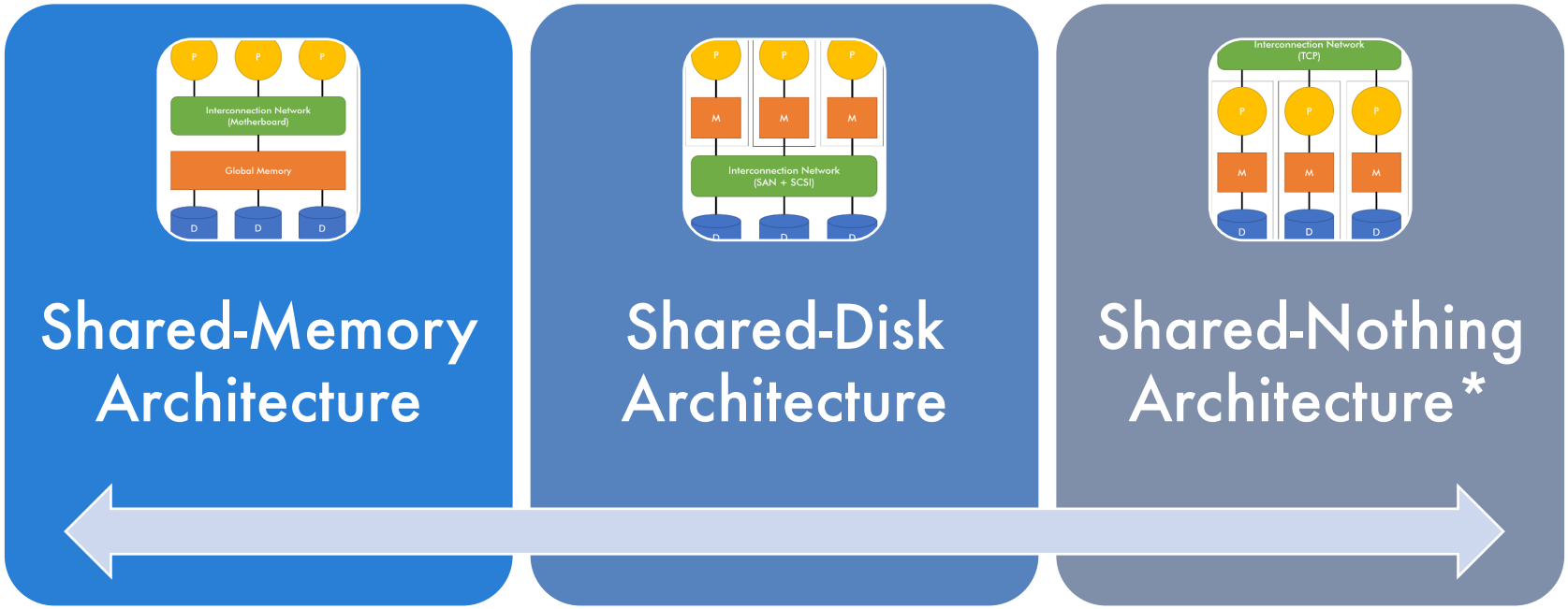
teradata.

APACHE Spark™

MySQL™ Cluster

Architecture Tradeoffs

- Traditional tradeoff was administration difficulty vs ability to scale
  - This is becoming less applicable



Implementations for Database Parallelism

- *Architecture Parallelism*
  - *Shared Memory*
  - *Shared Disk*
  - *Shared Nothing*

Hardware considerations

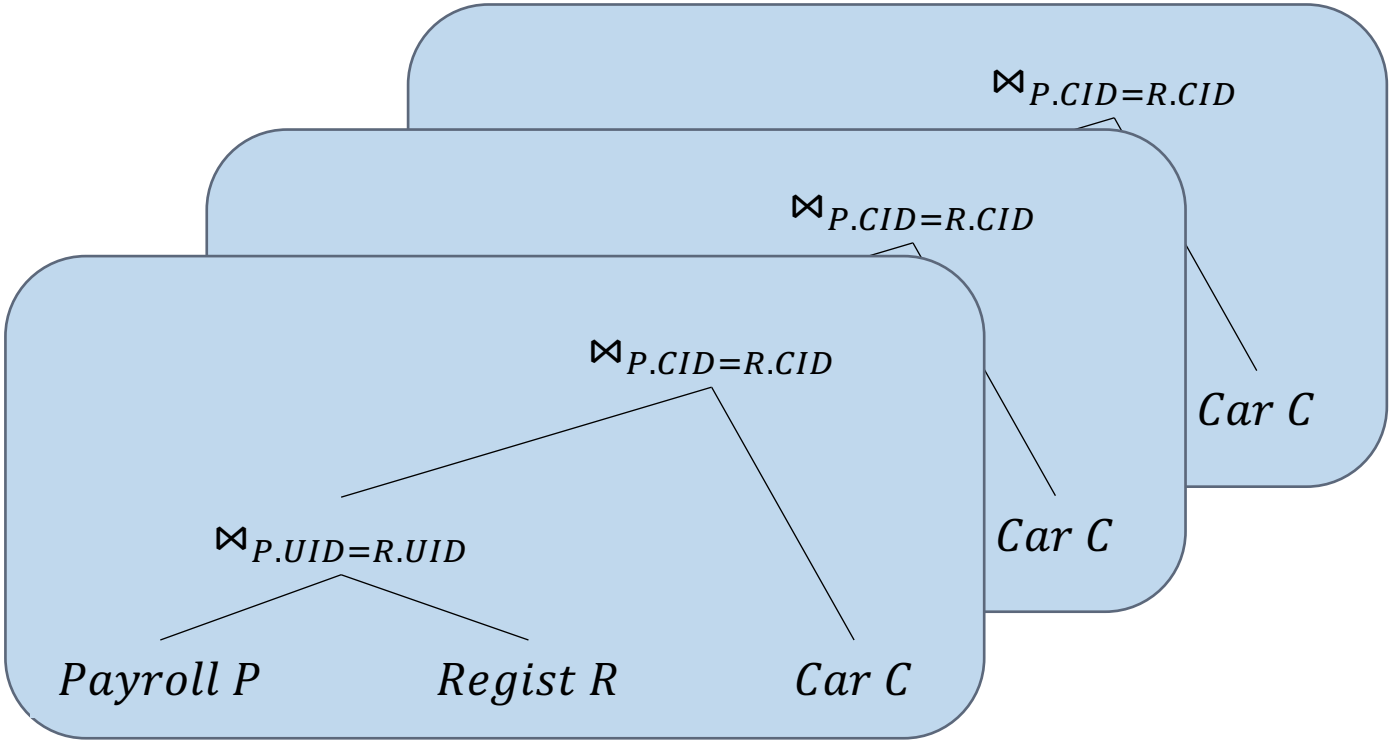
- **Query Parallelism**
  - Inter-Query Parallelism
    "parallelism between queries"
  - Intra-Query Parallelism
    "parallelism within queries"

Software considerations

- **Data partitioning**

Inter-Query Parallelism

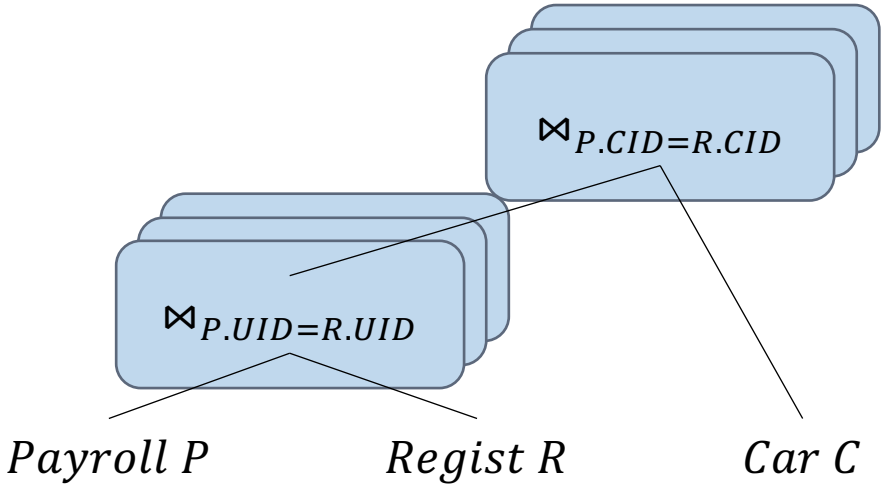
- Each <u>entire query</u> is processed on its own node
- Scales well for **lots of simple transactions** (OLTP)



Intra-Query Parallelism

- Each operator instance is processed by multiple nodes

- Scales well for **complex analytical queries** (OLAP)



Implementations for Database Parallelism

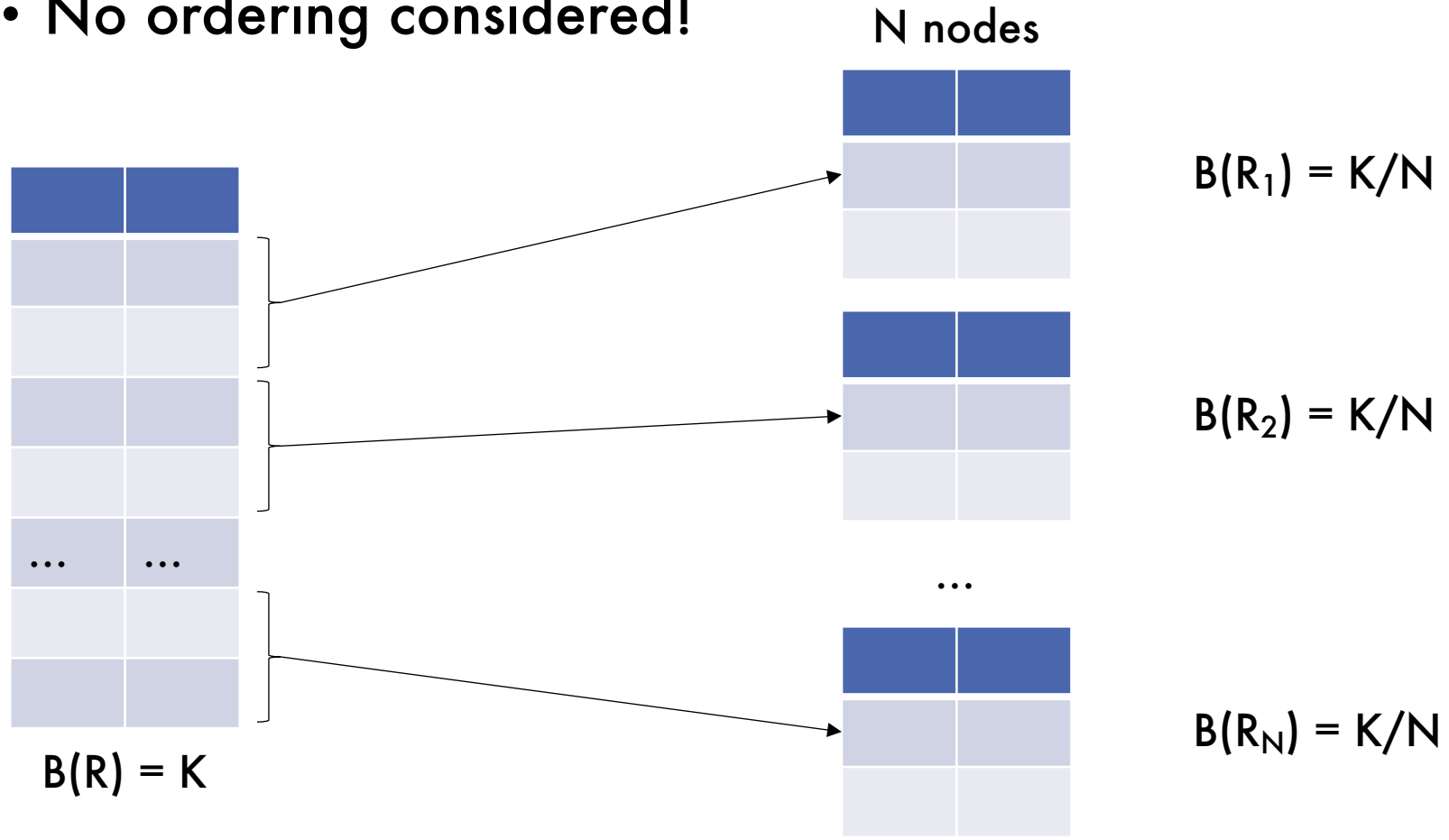
- *Architecture Parallelism*
  - *Shared Memory*
  - *Shared Disk*
  - *Shared Nothing*

- **Query Parallelism**
  - Inter-Query Parallelism
    "parallelism between queries"
  - Intra-Query Parallelism
    "parallelism within queries"

- **Data partitioning**

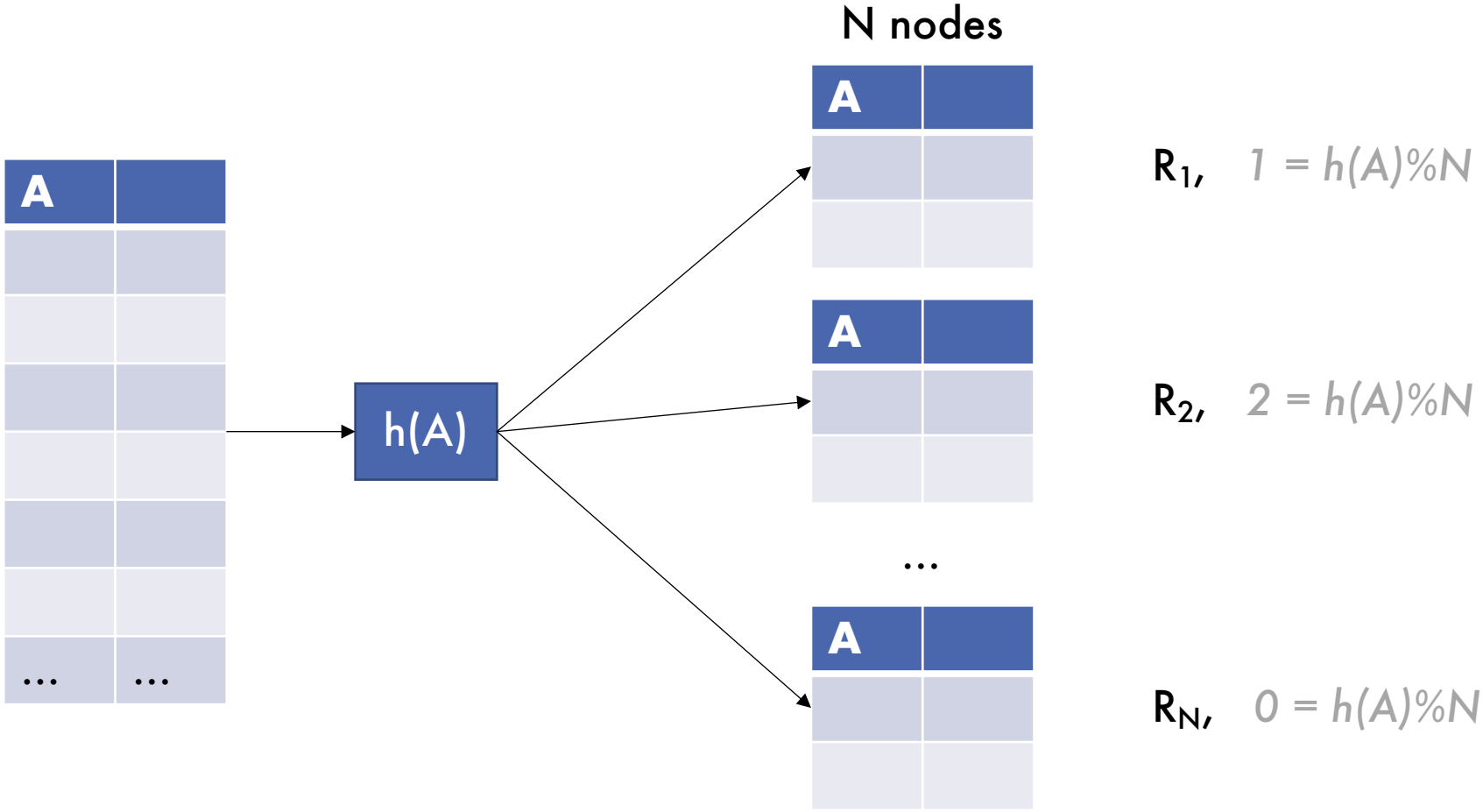
Block Partitioning

- Tuples are horizontally partitioned by raw size
  - No ordering considered!



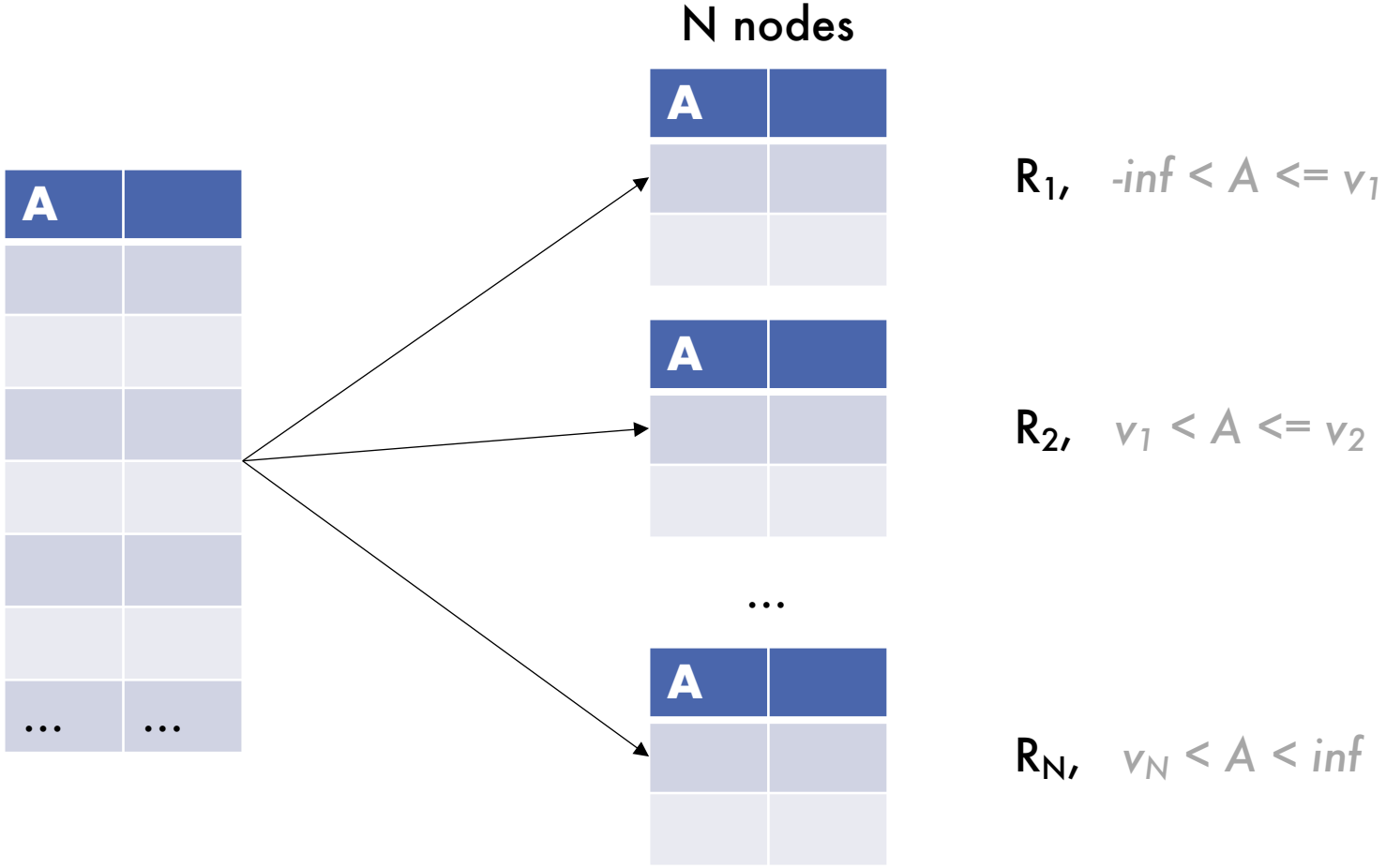
Hash Partitioning

- Node contains tuples with chosen attribute hashes



Range Partitioning

Node contains tuples in chosen attribute ranges



Skew: The Justin Bieber Effect

- Distributing data using some attribute of the data is works when the data follows a uniform distribution

- Certain nodes will become bottlenecks if a **poorly chosen attribute is used**

Outline

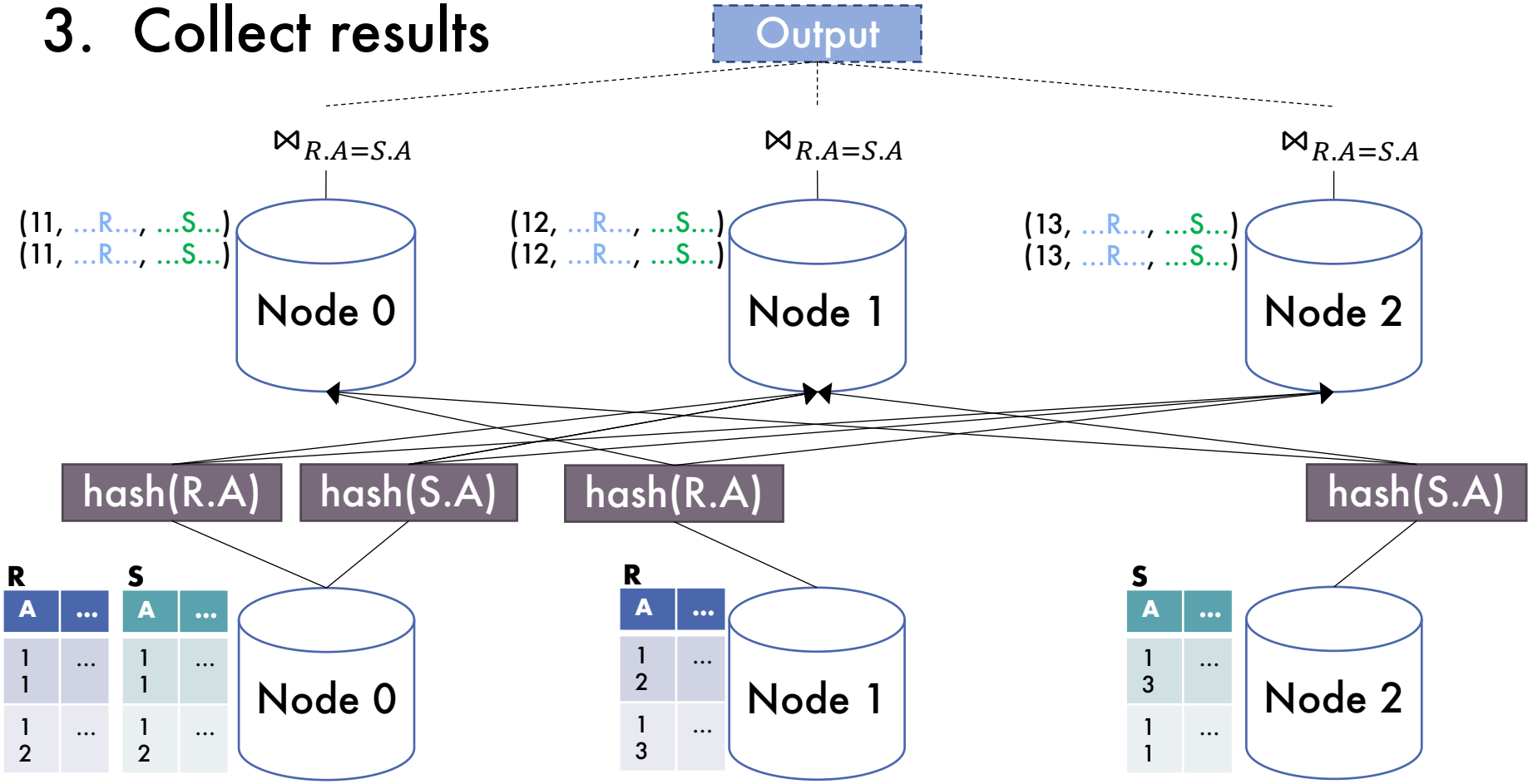
- Multiple Operation Execution
- Full Query Cost Estimation
- Parallelizing Data Management
  - Problems
  - Concepts
    - Performance Expectations
    - Parallelism Flavors
- **Parallelism Examples**

Partitioned Hash Equijoin

1. Hash shuffle tuples *on join attributes*
2. Local join
3. Collect results

R and S are *block* partitioned

```
SELECT *
  FROM R, S
 WHERE R.A = S.A
```

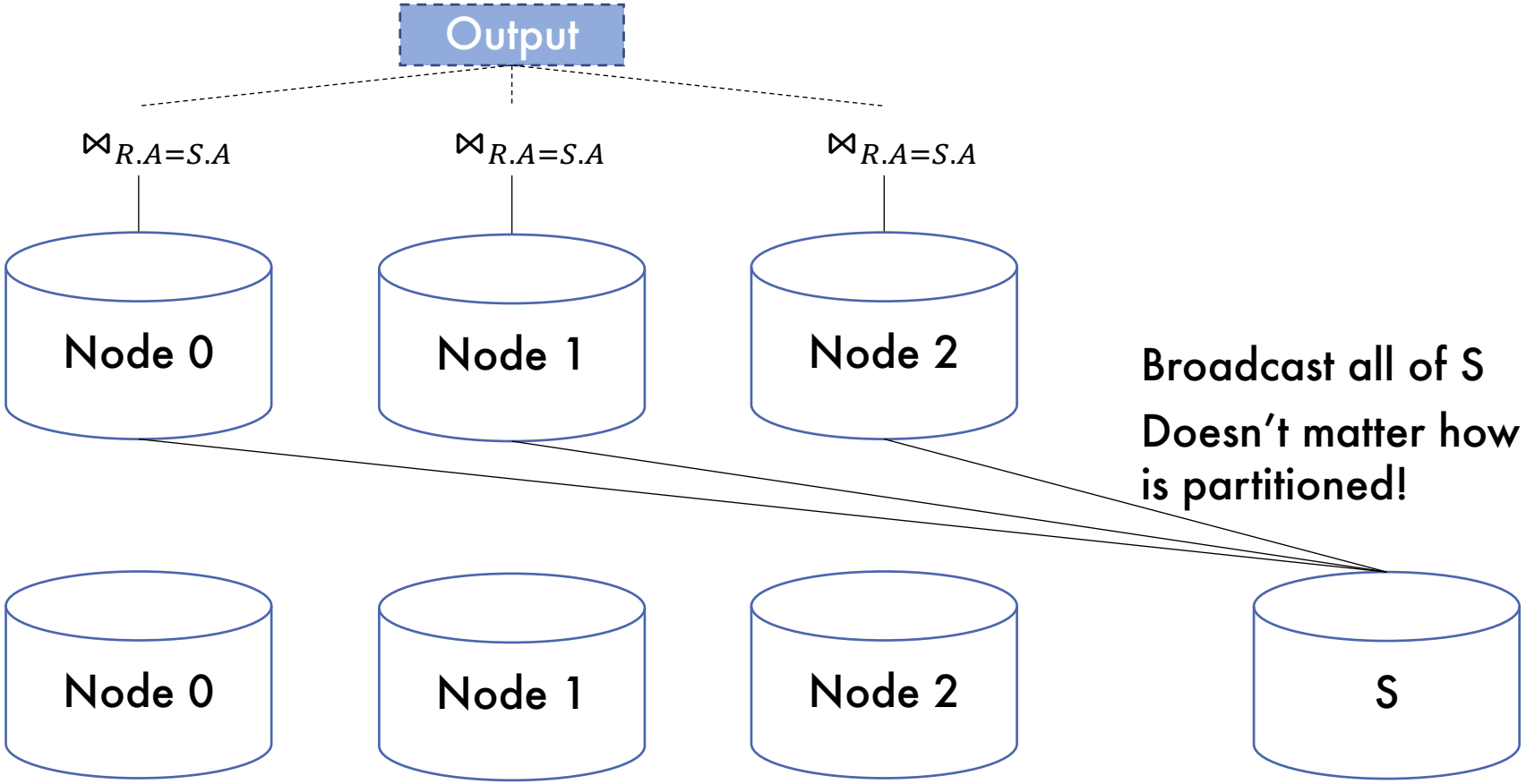


Broadcast Equijoin

R is *block* partitioned;
S is unpartitioned

```
SELECT *
  FROM R, S
 WHERE R.A = S.A
```

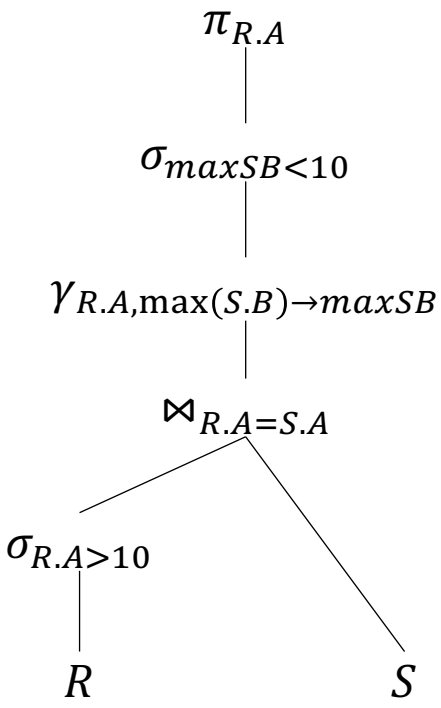
1. Broadcast unpartitioned table
2. Local join
3. Collect results



Broadcast all of S

Doesn't matter how R is partitioned!

Parallel Query Plan Example



R is *block* partitioned and S is *hash* partitioned on A