Introduction to Data Structure (Data Management) Lecture 15 – Parallel Databases (Ch. 20.1)

Felipe P. Vista IV

Reminder

- Everybody, make sure that your name in ZOOM is in the following format:
  - University ID Num Name (no “( )“)
  - Ex: 202054321 Juan Dela Cruz
  -
  - Not changing your name to this format
    - you might be marked Absent

    * → absent?

- Why Parallel Processing

- Architectures

- Distributed Query Processing

INTRO TO DATA STRUCTURE

Why Compute in Parallel?

Why Compute in Parallel?

- Multi-cores:
  - Most processors have multiple cores
  - This trend will increase in the future

- Big data: too large to fit in main memory
  - Distributed query processing on 100-1000 servers
  - Widely available now using cloud services

Big Data

- Companies, org's, scientists have data that is **too big**
  - sometimes too complex,
  - to be managed without changing tools and processes

- **Complex** data processing:
  - Decision support queries (SQL w/ aggregates)
  - Machine learning (adds linear algebra and iteration)

Two Kinds of Parallel Data Processing

- Parallel databases, developed starting with the 80s
  - OLTP (Online Transaction Processing)
  - OLAP (Online Analytic Processing, or Decision Support)

- General purpose distributed processing:
  MapReduce, Spark
  - Mostly for Decision Support Queries

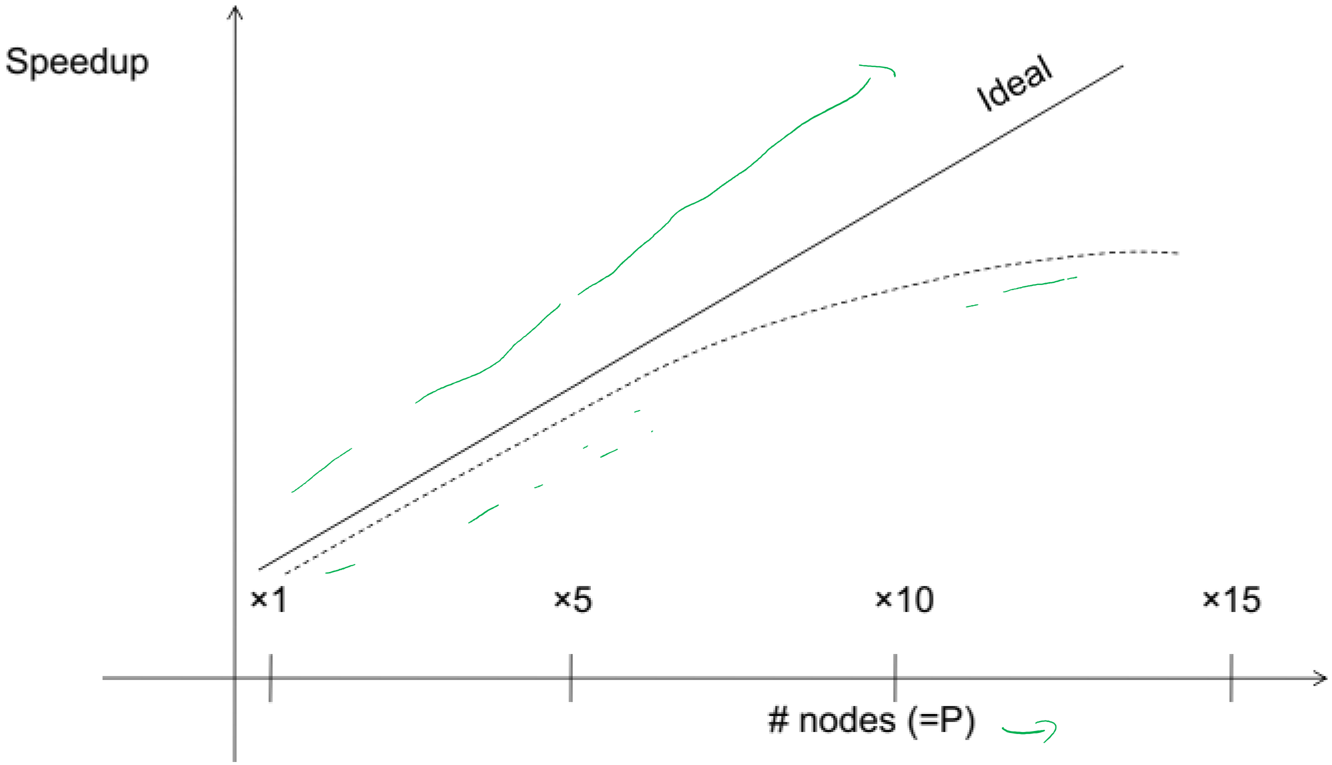
Performance Metrics for Parallel DBMSs

P = the number of nodes (processors, computers)

- Speedup:
  - More nodes, same data ➔ higher speed
- Scaleup:
  - More nodes, more data ➔ same speed

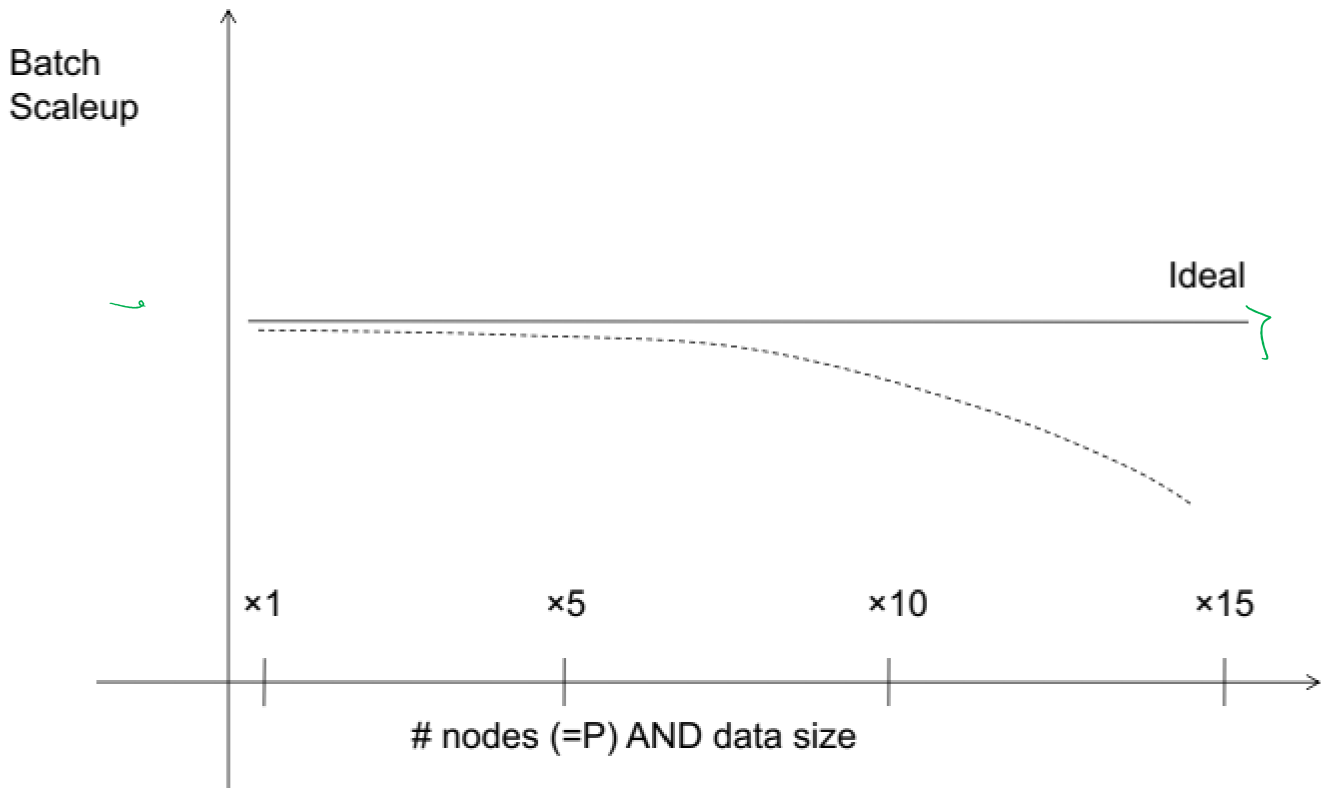
- OLTP: “Speed” = transactions per second (TPS)
- Decision Support: “Speed” = query time

Linear vs. Non-linear Speedup



* Speedup - More nodes, same data → higher speed
* Scaleup - More nodes, more data → same speed

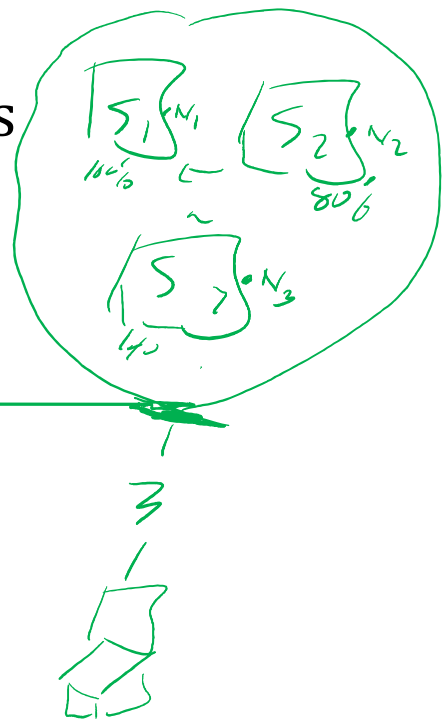
Linear vs. Non-linear Scaleup



* Speedup - More nodes, same data → higher speed
* Scaleup - More nodes, more data → same speed

Challenges to Linear Speedup and Scaleup

- Startup cost
  - Cost of starting an operation on many nodes
- Interference
  - Contention for resources between nodes
- Stragglers
  - Slowest node becomes the bottleneck



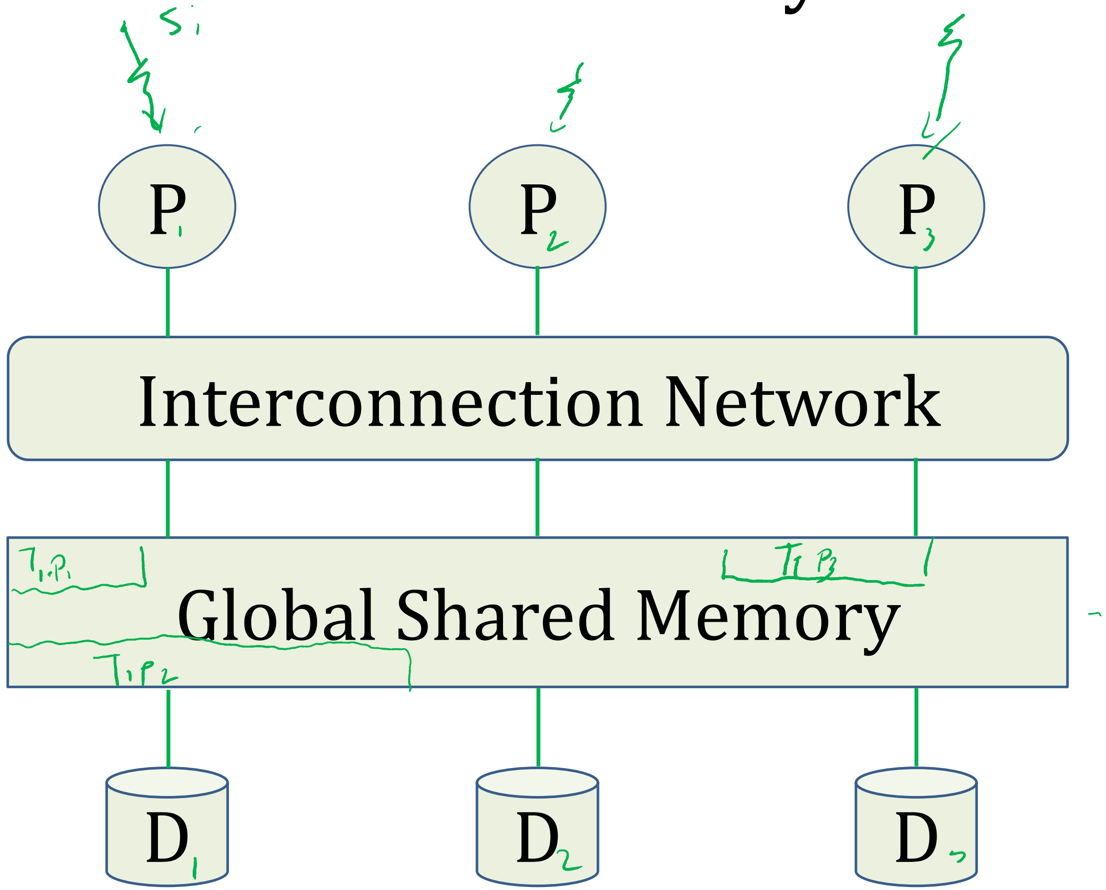
INTRO TO DATA STRUCTURE

Parallel DB Architectures

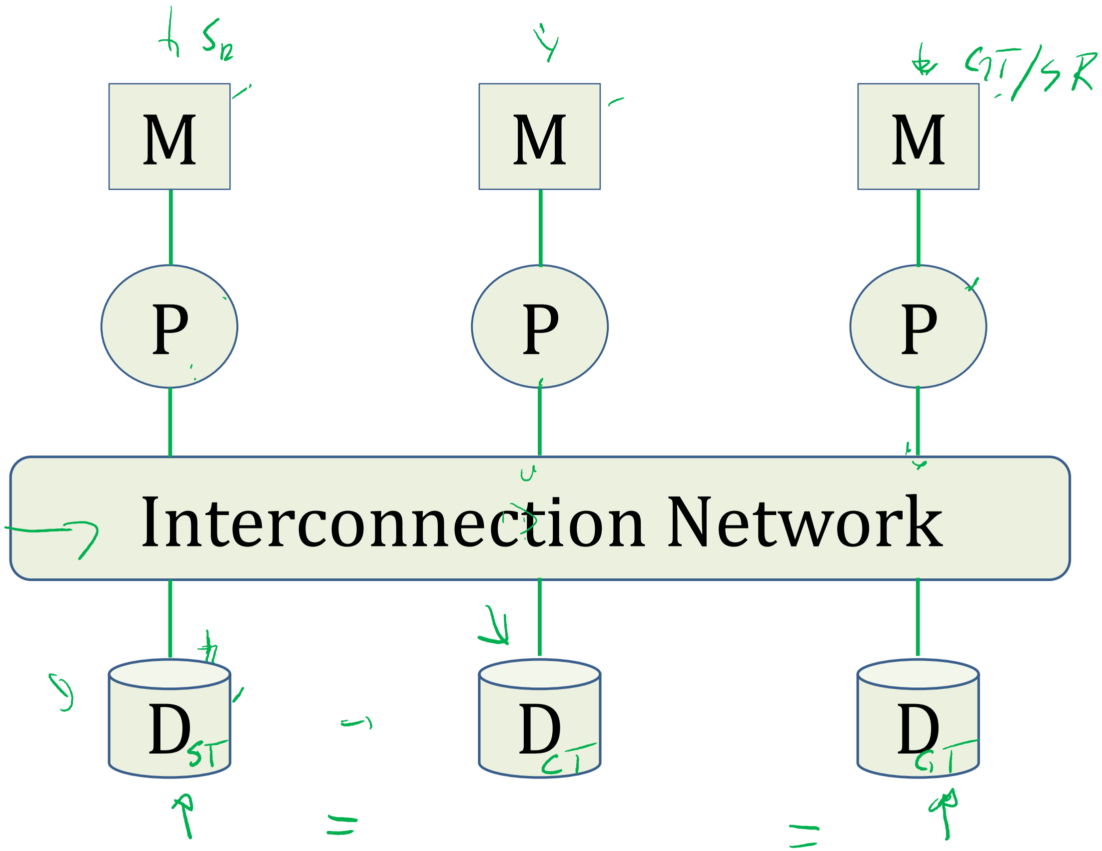
Architectures for Parallel DBs

- Shared memory
- Shared disk
- Shared nothing

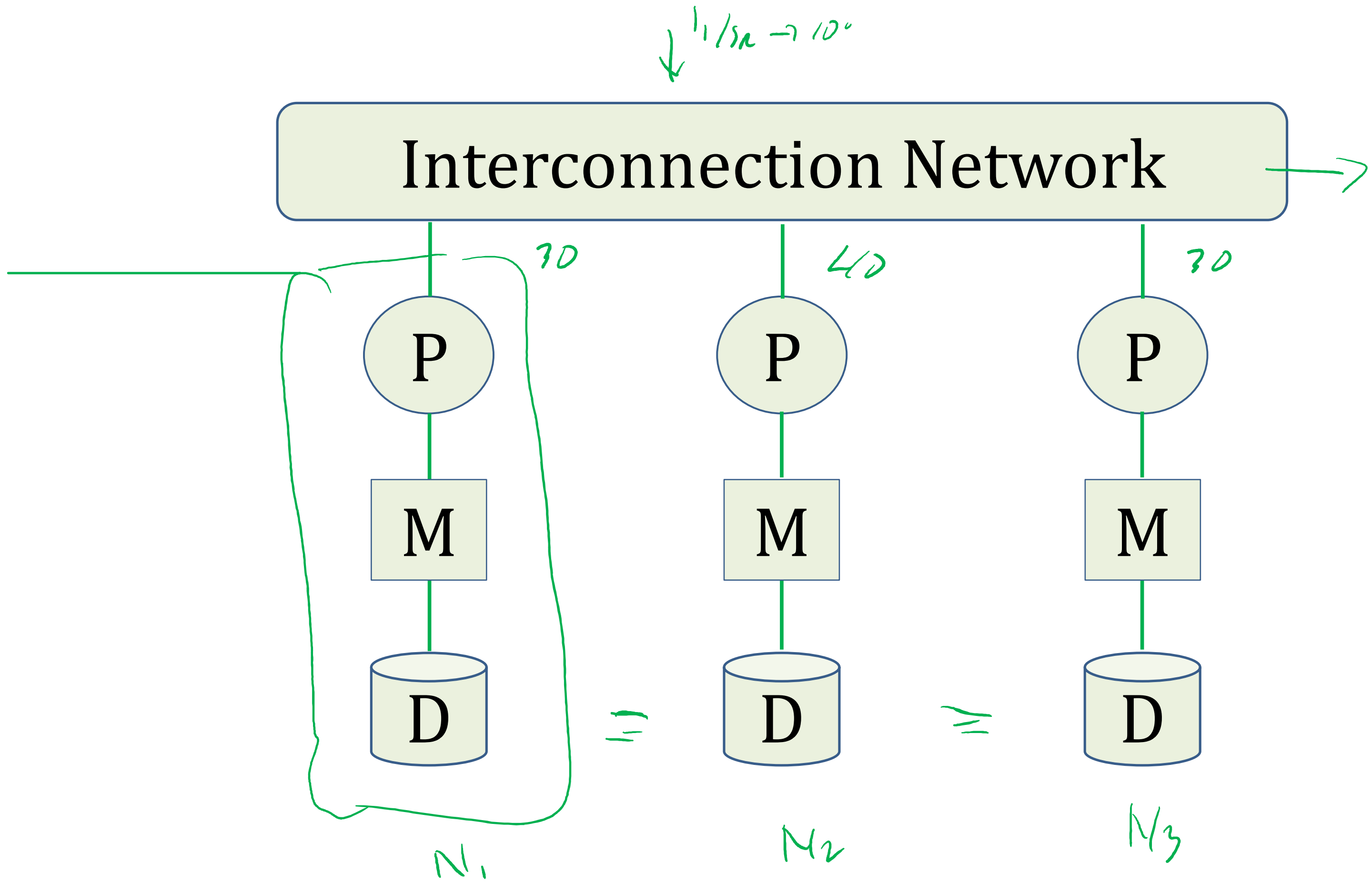
Shared Memory



Shared Disk



Shared Nothing



A Professional View

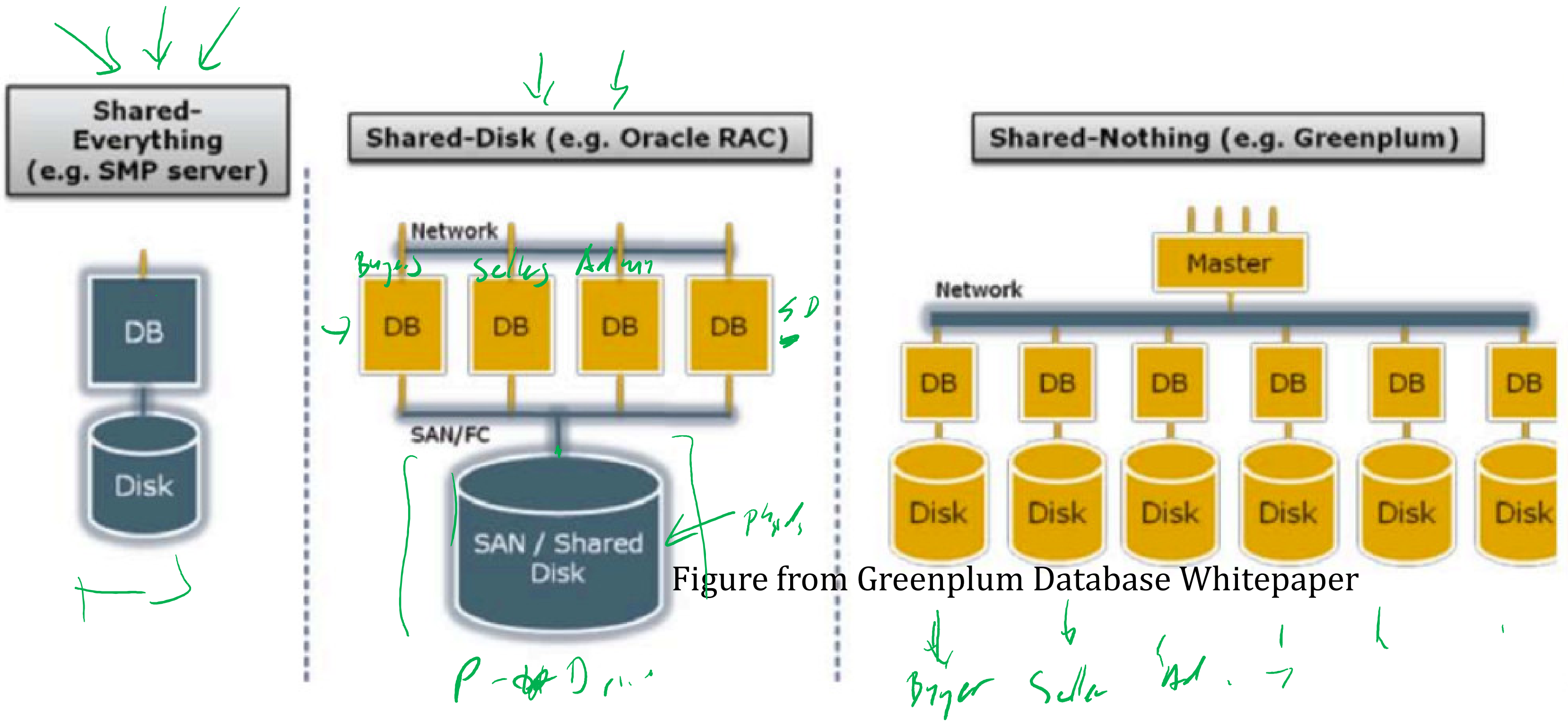


Figure from Greenplum Database Whitepaper

SAN = "Storage Area Network"

Shared Memory

Random Access Memory

- Nodes share both RAM and disk
- Dozens to hundreds of processors

Licensed   dB Server

Example: ***SQL Server*** runs on a single machine and can leverage many threads to get a query to run faster (see query plans)

- Easier to program and easy to use
- But very expensive to scale: last remaining cash cows in the hardware industry

* cash cow – provide steady income or profit that is far bigger than cot to acquire, start or upgrade (scale)

Shared Disk

- All nodes access the same disks
- Found in largest "single-box"(noncluster) multiprocessors

Oracle dominates this class of systems.

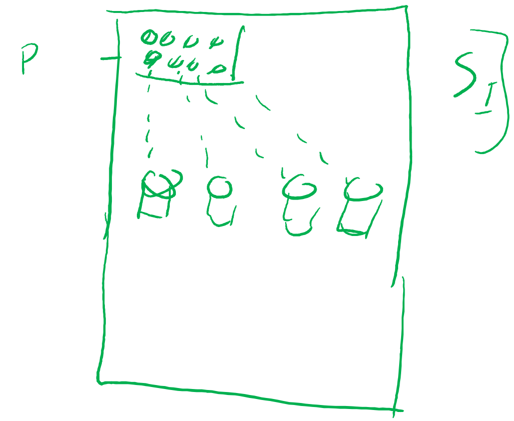
Characteristics:

- Also hard to scale past a certain point: existing deployments typically have fewer than 10 machines

Shared Nothing

- Cluster of machines on high-speed network
- Each machine has its own memory and disk: lowest contention

NOTE: Because all machines today have many cores and many disks, then shared-nothing systems typically run many "nodes" on a single physical machine.



Characteristics:

- Today, this is the most scalable architecture.
- Most difficult to administer and tune

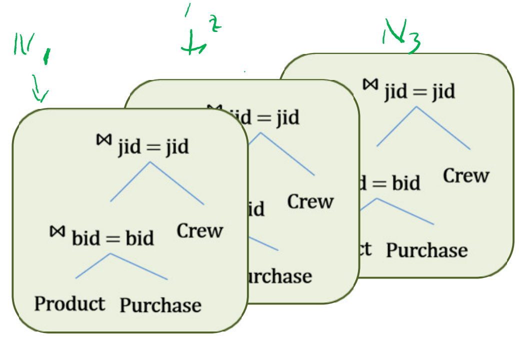
* Contention – competition for resources

INTRO TO DATA STRUCTURE

Distributed Query Processing

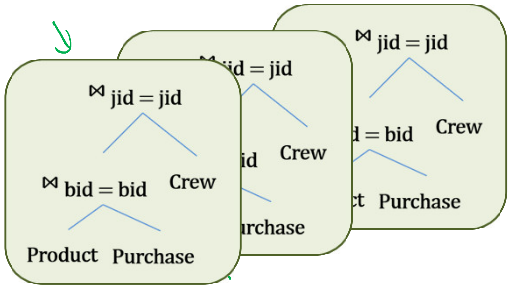
Approaches to Parallel Query Evaluation

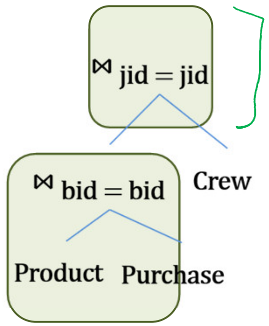
- Inter-query parallelism
  - Transaction per node
  - OLTP



Approaches to Parallel Query Evaluation

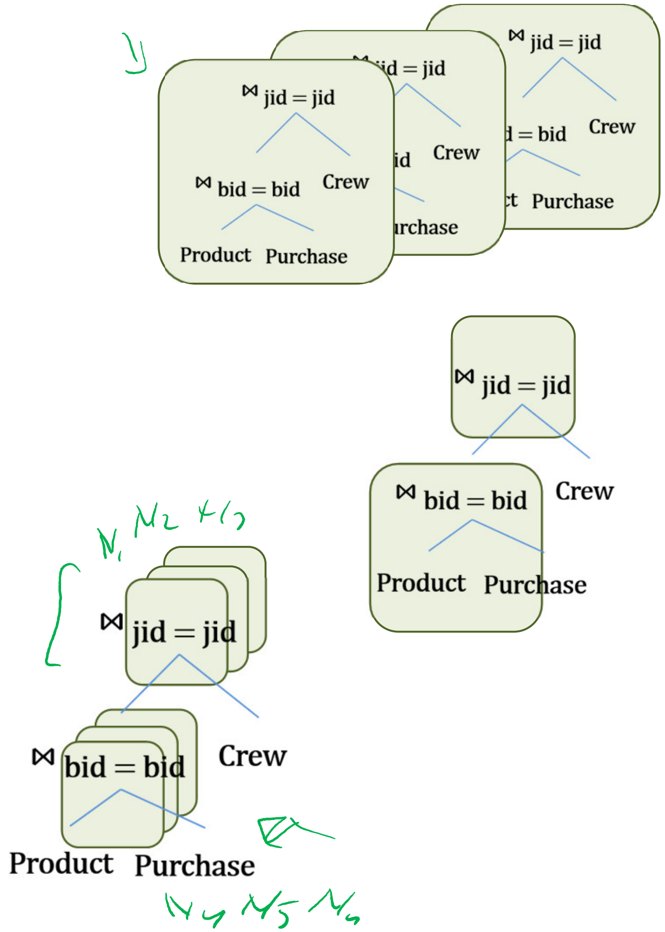
- Inter-query parallelism
  - Transaction per node
  - OLTP
- Inter-operator parallelism
  - Operator per node
  - Both OLTP and Decision Support





Approaches to Parallel Query Evaluation

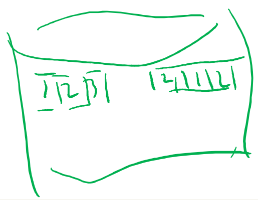
- Inter-query parallelism
  - Transaction per node
  - OLTP
- Inter-operator parallelism
  - Operator per node
  - Both OLTP and Decision Support
- Intra-operator parallelism
  - Operator on multiple node
  - Decision Support
  - Most scalable



Review on Single Node Query Processing

Given relations R(A, B) and S(B, C), no indexes:

- **Selection**: $\sigma_{A=123}(R)$
  - Scan file R, select records with A=123
- **Group-by**: $\gamma_{A,\ sum(B)}(R)$
  - Scan file R, insert into a hash table using attribute A as key
  - When a new key is equal to an existing one, add B to value
- **Join**: $R \bowtie S$
  - Scan file S, insert into a hash table using attribute B as key
  - Scan file R, probe the hash table using attribute B

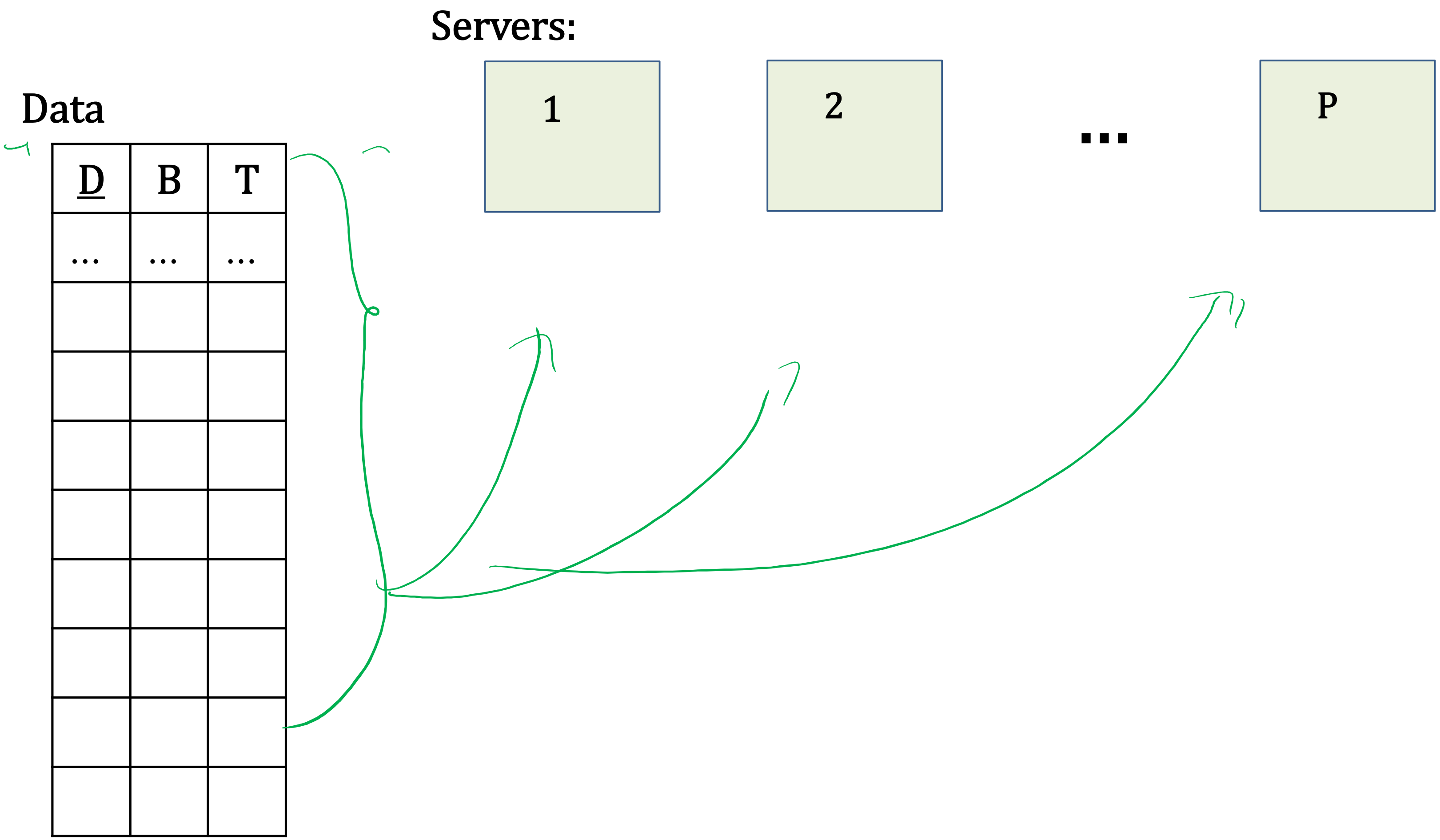


Distributed Query Processing

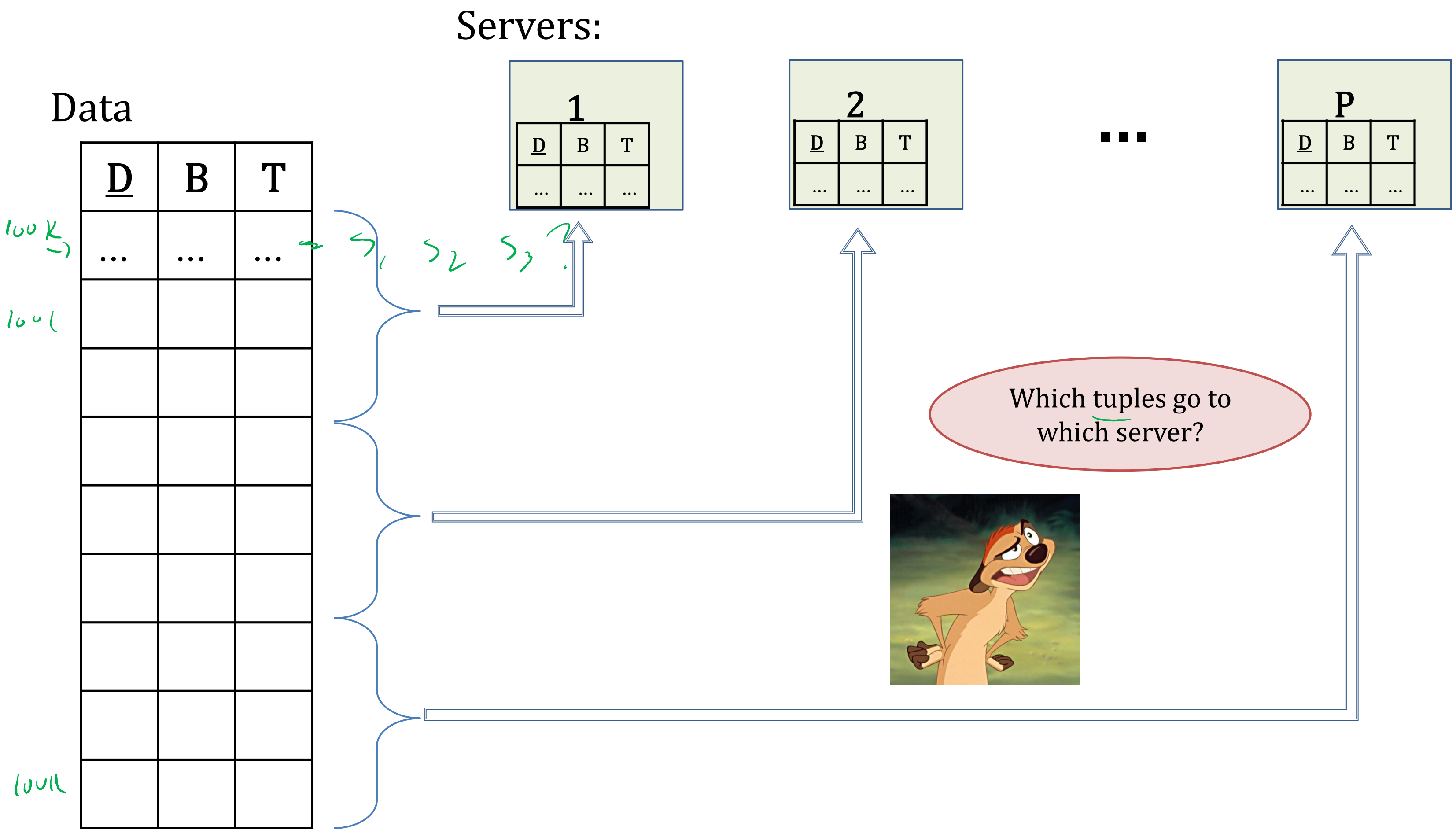
- Data is **horizontally partitioned** across many servers

- Operators may require data reshuffling
  - Not all the needed data is in one place

Horizontal Data Partitioning



Horizontal Data Partitioning

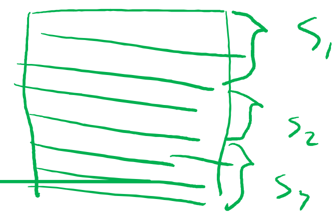


Horizontal Data Partitioning

- Block Partition:
  - Partition tuples arbitrarily s.t. $size(R_1) \approx \ldots \approx size(R_P)$

Ps mod 5

- Hash partitioned on attribute A:
  - Tuple t goes to chunk i, where $i = h(t.A) \bmod P + 1$



- Range partitioned on attribute A:
  - Partition the range of A into $-\infty = v_0 < v_1 < \ldots < v_P = \infty$
  - Tuple t goes to chunk i, if $v_{i-1} < t.A < v_i$

Parallel GroupBy

Data: R(<u>K</u>, A, B, C)

Query: $\gamma_{A,\ sum(C)}(R)$

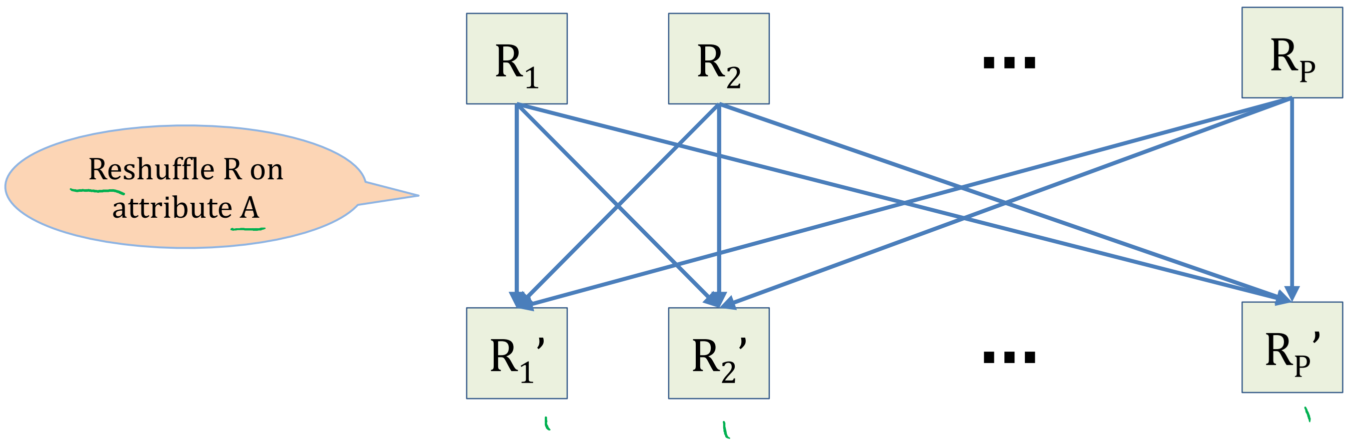
- How can we compute in each case
  - R is hash-partitioned on A
  - R is block-partitioned
  - R is hash-partitioned on K

Parallel GroupBy

Data: R(<u>K</u>, A, B, C)

Query: $\gamma_{A,\, sum(C)}(R)$

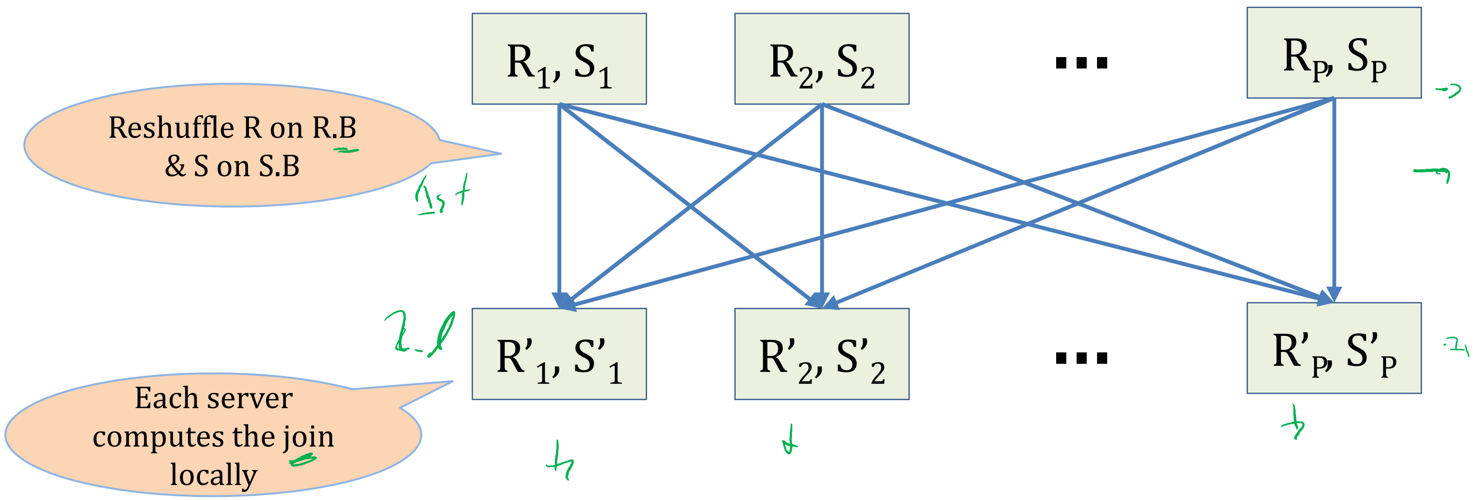
- R is block-partitioned or hash-partitioned on K



Parallel Join

- **Data:** R(<u>K1</u>, A, B), S(<u>K2</u>, B, C)
- **Query:** R(<u>K1</u>, A, B) ⋈ S(<u>K2</u>, B, C)

Initially, both R and S are horizontally partitioned on K1 and K2



```
Data: R(K1, A, B), S(K2, B, C)
Query: R(K1, A, B) ⋈ S(K2, B, C)
```

Parallel Join

```
Data: R(K1, A, B), S(K2, B, C)
Query: R(K1, A, B) ⋈ S(K2, B, C)
```

Parallel Join

Range

Partition

M1

R1

K1	B
1	20
2	50

S1

K2	B
101	50
102	50

M2

R2

K1	B
3	20
4	20

S2

K2	B
201	20
202	50

```
Data: R(K1, A, B), S(K2, B, C)
Query: R(K1, A, B) ⋈ S(K2, B, C)
```

Parallel Join

Partition

M1

R1

K1	B
1	20
2	50

S1

K2	B
101	50
102	50

M2

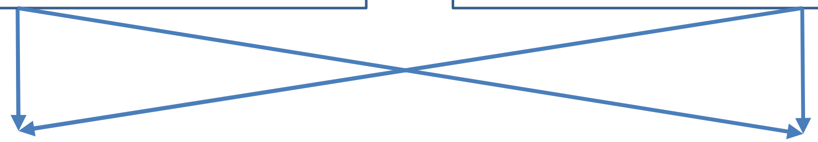
R2

K1	B
3	20
4	20

S2

K2	B
201	20
202	50

Shuffle



Parallel Join

```
Data: R(K1, A, B), S(K2, B, C)
Query: R(K1, A, B) ⋈ S(K2, B, C)
```

Partition

M1

R1

K1	B
1	20
2	50

S1

K2	B
101	50
102	50

M2

R2

K1	B
3	20
4	20

S2

K2	B
201	20
202	50

Shuffle on R.B, S.B.

Local Join

M1

R1'

K1	B
1	20
2	~~50~~ 201
4	20

⋈

S1'

K2	B
201	20

M2

R2'

K1	B
2	50

S2'

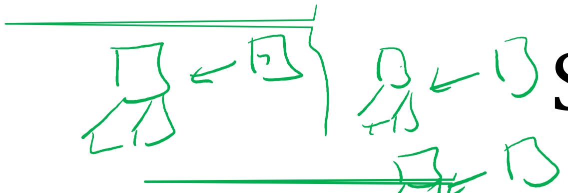
K2	B
101	50
102	50
202	50

Speedup and Scaleup

- Consider:
  - Query: $\gamma_{A,\ sum}(C)(R)$ → $\gamma_{A,\ sum(C)}(R)$
  - Runtime: dominated by reading chunks from disk

Speedup and Scaleup

- Consider:
  - Query: $\gamma_{A,\ sum}(C)(R)$
  - Runtime: dominated by reading chunks from disk
- If we double the number of nodes P, what is the new running time?
  - Half (each server holds ½ as many chunks)



Speedup and Scaleup

$t_2 = \frac{t_1}{2}$

- Consider:
  - Query: $\gamma_{A,\, sum}(C)(R)$
  - Runtime: dominated by reading chunks from disk
- If we double the number of nodes P, what is the new running time? s.t #1: #P => (#P) x 2 ; $R_1 = R_2$; $T_2 = \frac{T_1}{2}$
  - Half (each server holds ½ as many chunks)
- If we double both P and the size of R, what is the new running time? #P => (#P) x 2 ; $R_2 = R_1 \times 2$; T_2
  - Same (each server holds the same # of chunks)

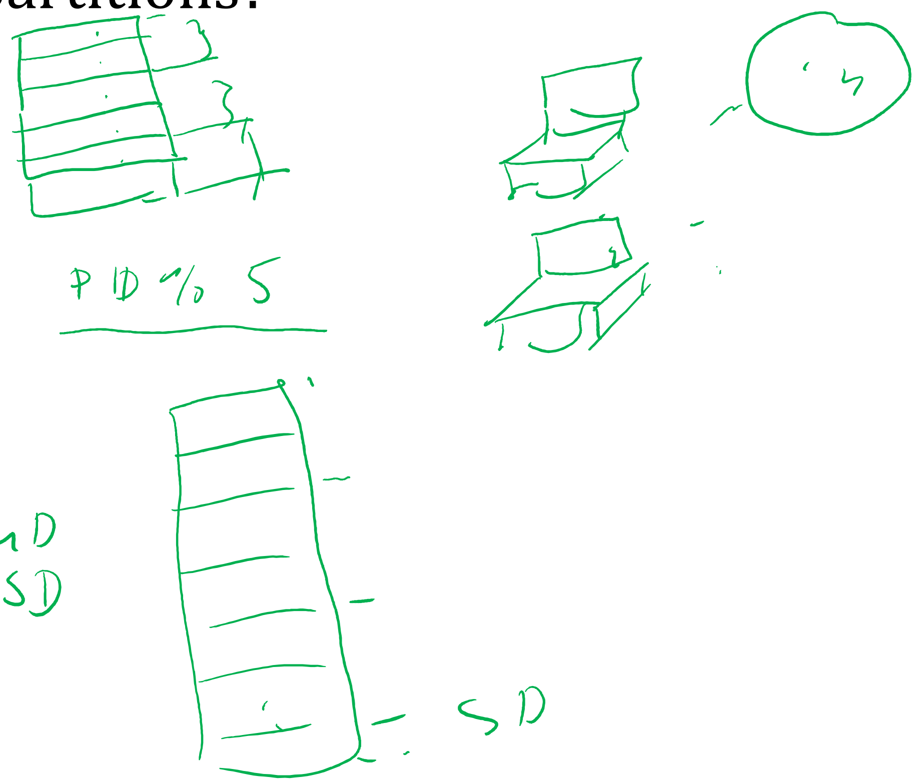
B: same running time (t_1)
C: half running time (t_1)

Uniform Data vs. Skewed Data

- Let R(K, A, B, C); which of the following partition methods may result in skewed partitions?

UD

- Block partition
- Hash-partition UD PID % 5
  - On the key K
  - On the attribute A
- Range-partition UD SD
  - On the key K
  - On the attribute A



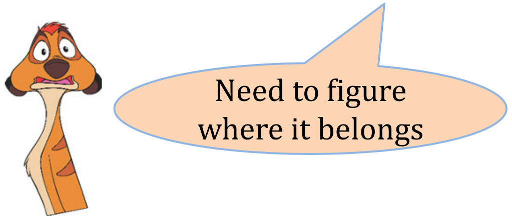
Uniform Data vs. Skewed Data

- Let R(K, A, B, C); which of the following partition methods may result in skewed partitions?

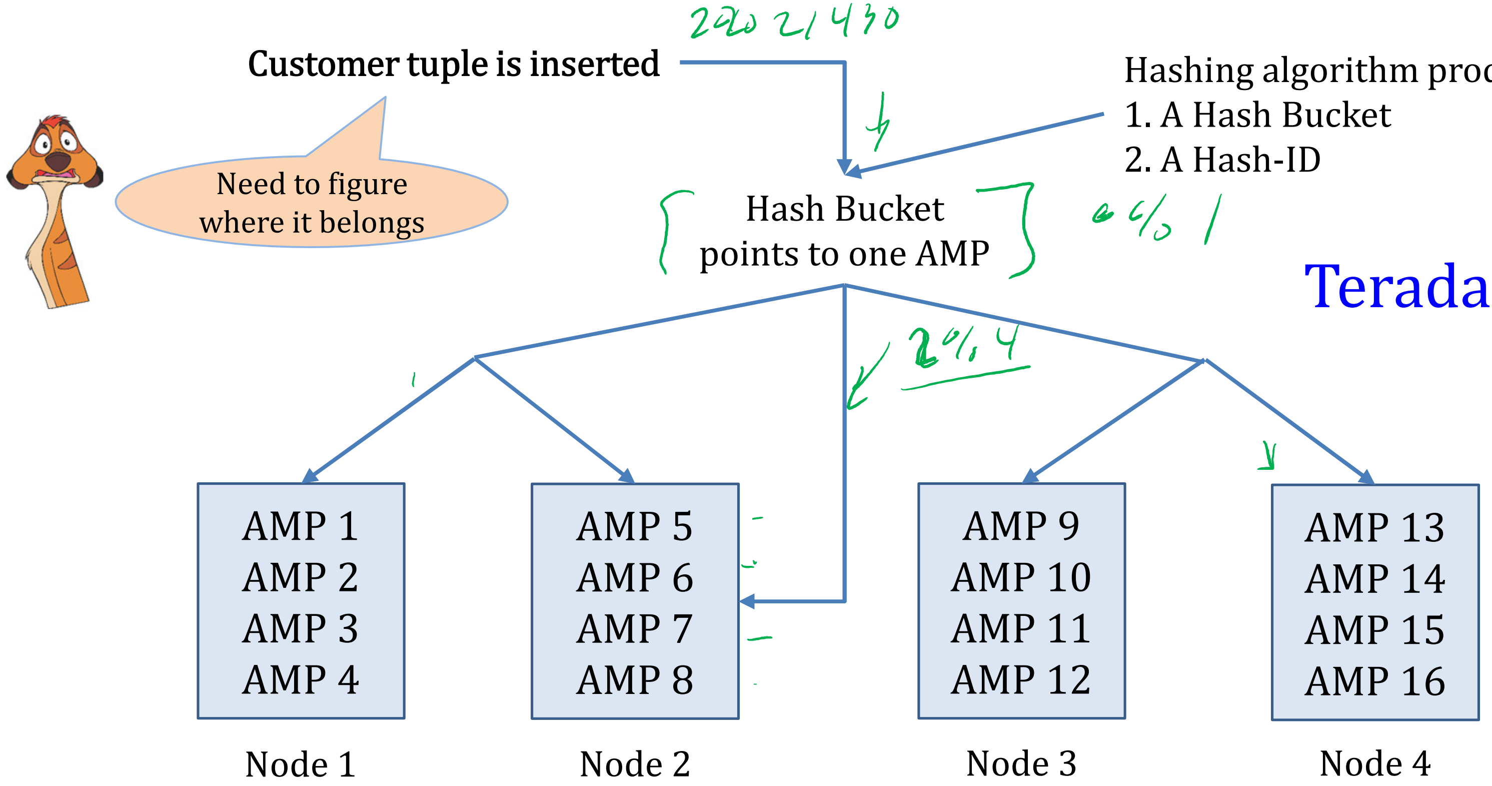
- Block partition — Uniform
- Hash-partition
  - On the key K — Uniform — Assuming good hash function
  - On the attribute A — May be skewed — Ex.: When all tuples have the same value of the attribute A, then all records end up in the same partition
- Range-partition
  - On the key K
  - On the attribute A — May be skewed — Difficult to partition the range of A uniformly

Loading Data into a Parallel DBMS

Customer tuple is inserted



Loading Data into a Parallel DBMS



Teradata Systems

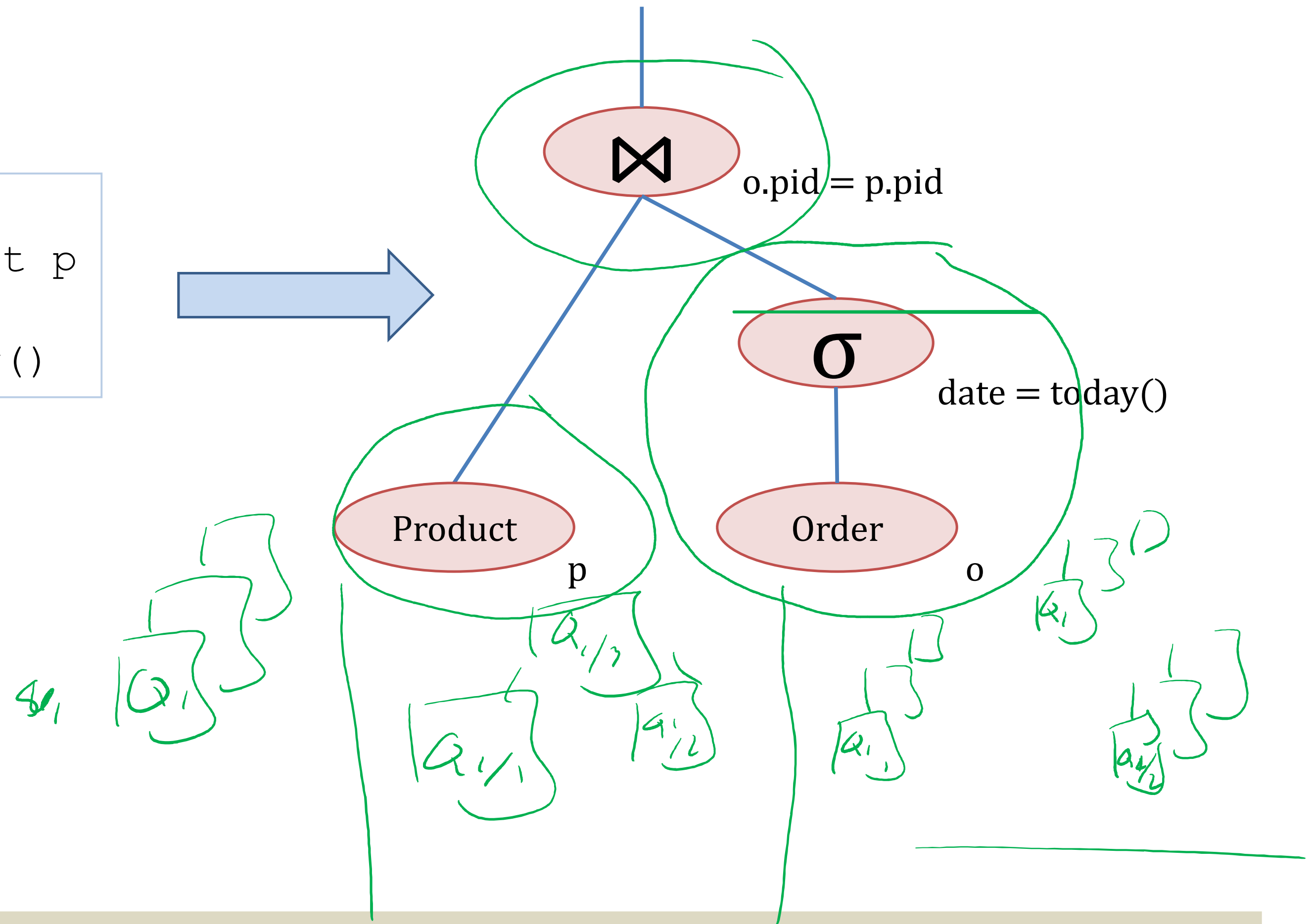
1%4 = 1
2%4 = 2
3%4 = 3
4%4 = 0

AMP = "Access Module Processor" = unit of parallelism

Example Parallel Query Execution

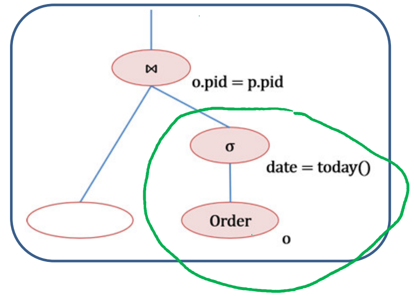
```
Order(oid, pid, date)
Product(pid, …)
```

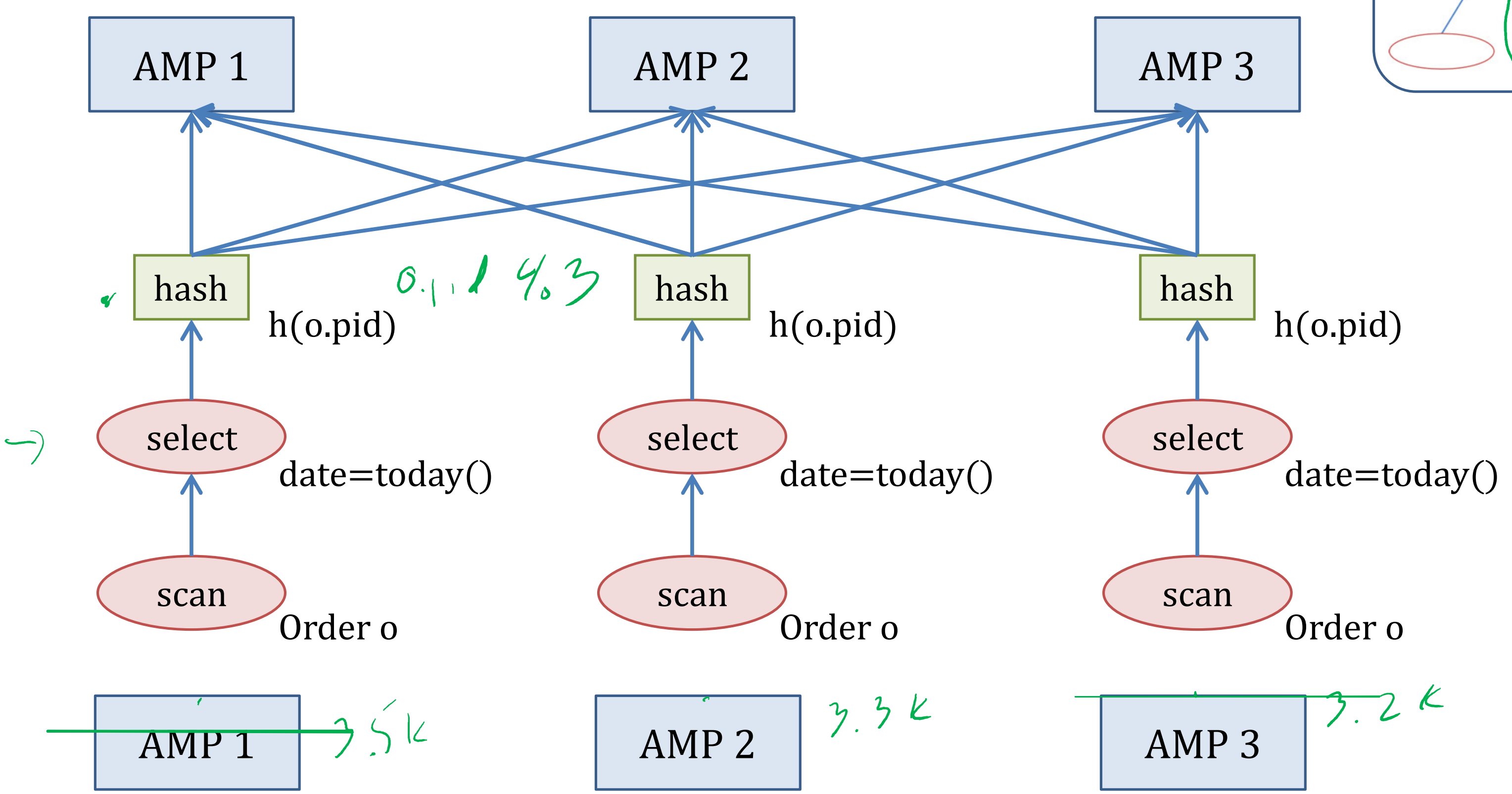
```
SELECT *
FROM Order o, Product p
WHERE o.pid = p.pid
  AND o.date = today()
```



Example Parallel Query Execution

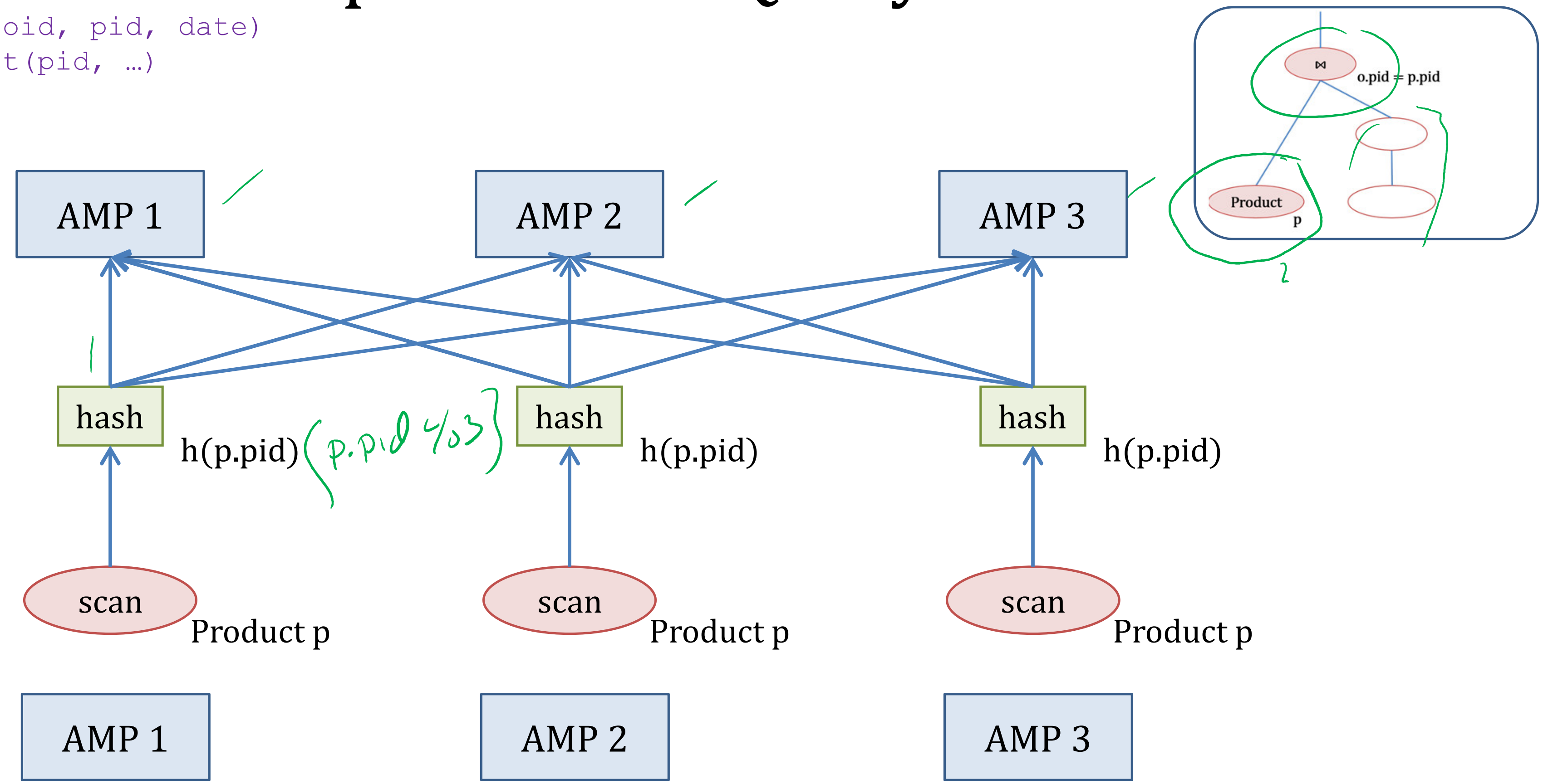
```
Order(oid, pid, date)
Product(pid, …)
```





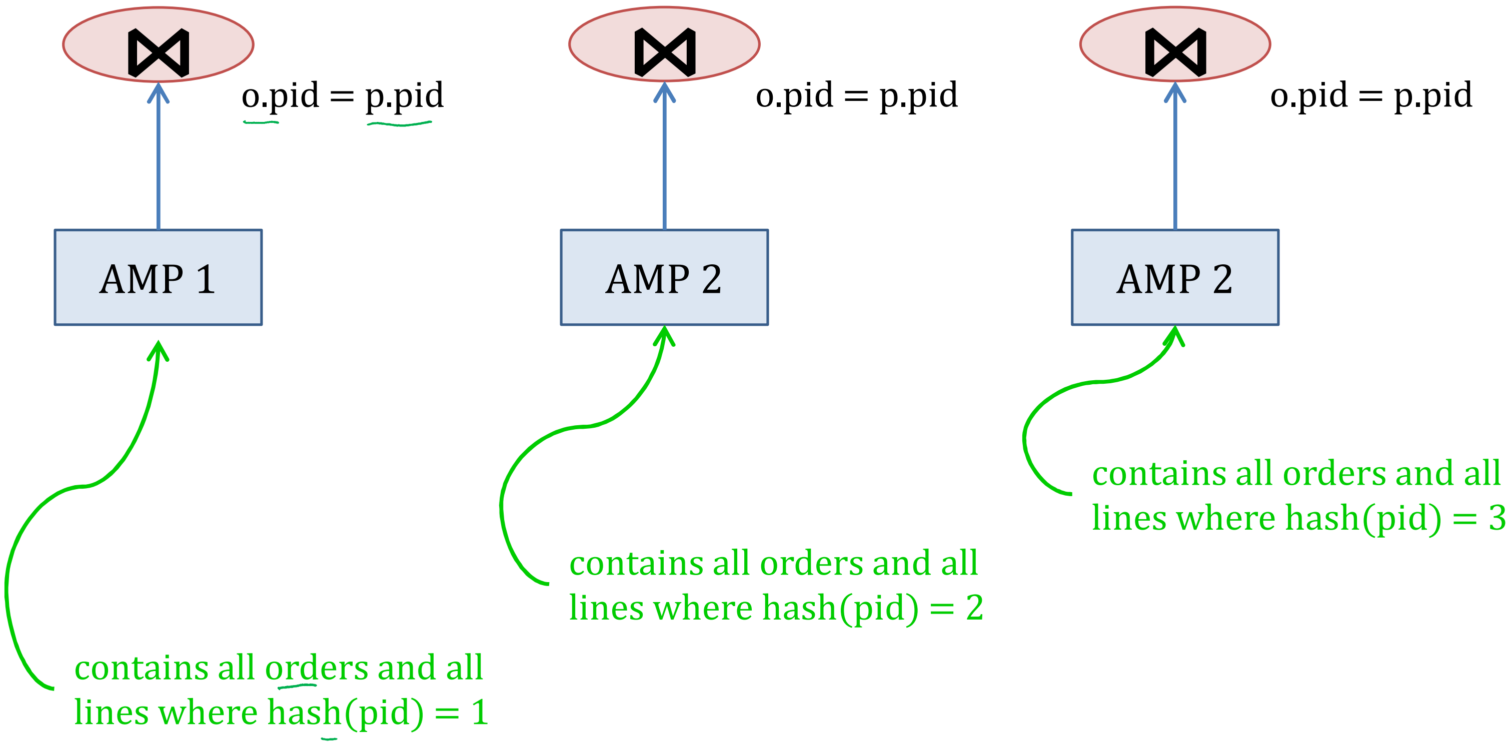
Example Parallel Query Execution

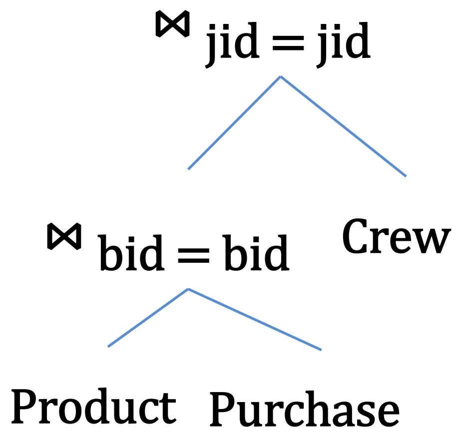
```
Order(oid, pid, date)
Product(pid, …)
```

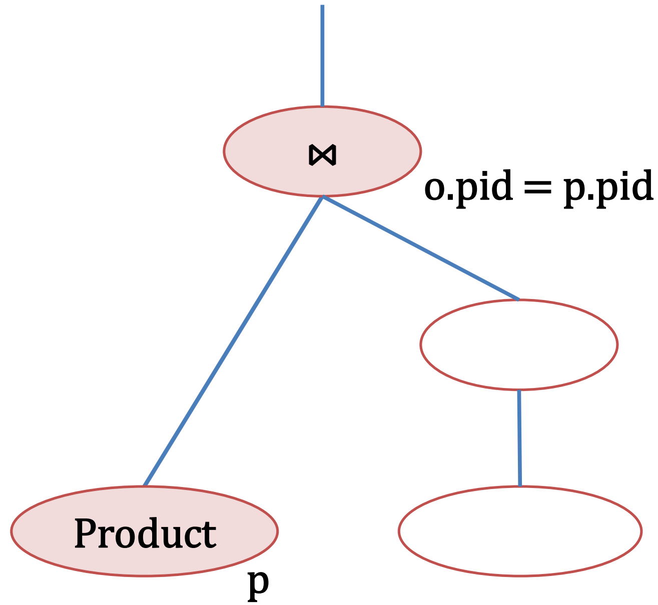


Example Parallel Query Execution

```
Order(oid, pid, date)
Product(pid, …)
```




⋈ jid = jid
⋈ bid = bid
Crew
Product
Purchase


⋈
o.pid = p.pid
Product
p

Thank you.