

# DATA516/CSED516

## Scalable Data Systems and Algorithms

### Lecture 5

Parallel Query Execution (cont.) +  
Graphs

# Administrivia

- AWS Charges Due to Databricks
  - NAT Gateway charges (not in Free Trial)
    - Contact AWS Customer Service (UW student, Free Trial)
- Assignment 2 Postponed
  - TA's looking into work around

# Announcements

- Project proposals were due on Friday
  - If you planned to use Databricks, you can optionally use MS Cloud Credits
- HW3 (Snowflake) will be released today

# Outline

- Distributed Joins

- Skew

- Parallel Query Processing Wrap-up

- Graph

# Parallel/Distributed Join

Three “algorithms”:

- Hash-partitioned
- Broadcast
- Combined: “skew-join” or other names

# Distributed Hash-Join

# Hash Join: $R \bowtie_{A=B} S$

Data:  $R(A, C), S(B, D)$

Query:  $R \bowtie_{A=B} S$



Initially, R and S are block partitioned.

Notice: they may be stored in DFS (recall MapReduce)

Some servers hold R-chunks, some hold S-chunks, some hold both

# Hash Join: $R \bowtie_{A=B} S$

Data:  $R(A, C), S(B, D)$

Query:  $R \bowtie_{A=B} S$

Reshuffle R on R.A  
and S on S.B



Initially, R and S are block partitioned.

Notice: they may be stored in DFS (recall MapReduce)

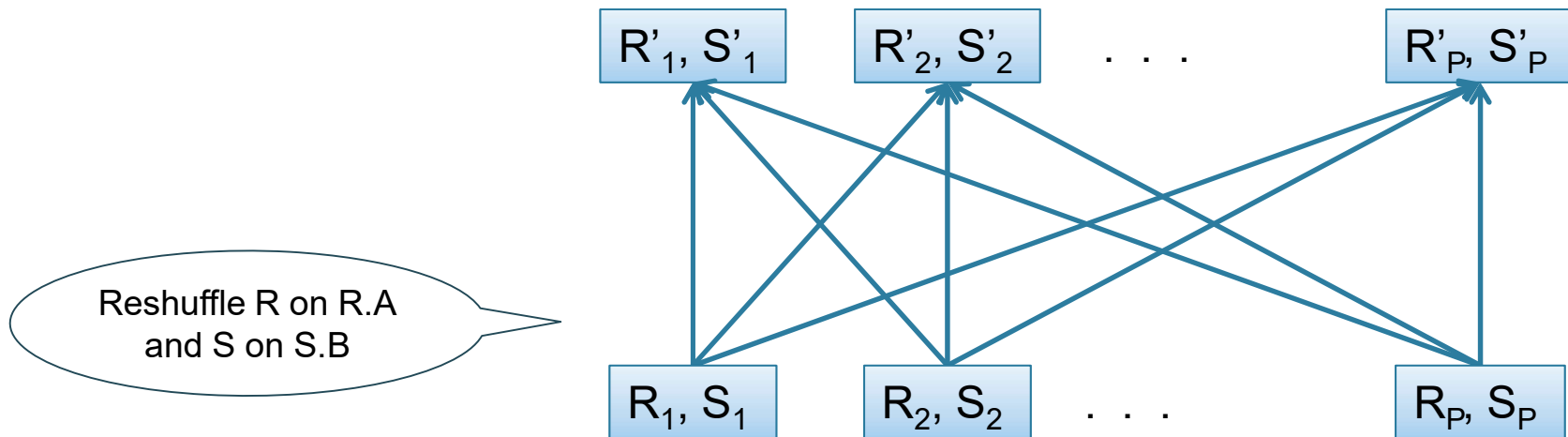
Some servers hold R-chunks, some hold S-chunks, some hold both



# Hash Join: $R \bowtie_{A=B} S$

Data:  $R(A, C), S(B, D)$

Query:  $R \bowtie_{A=B} S$



Initially, R and S are block partitioned.

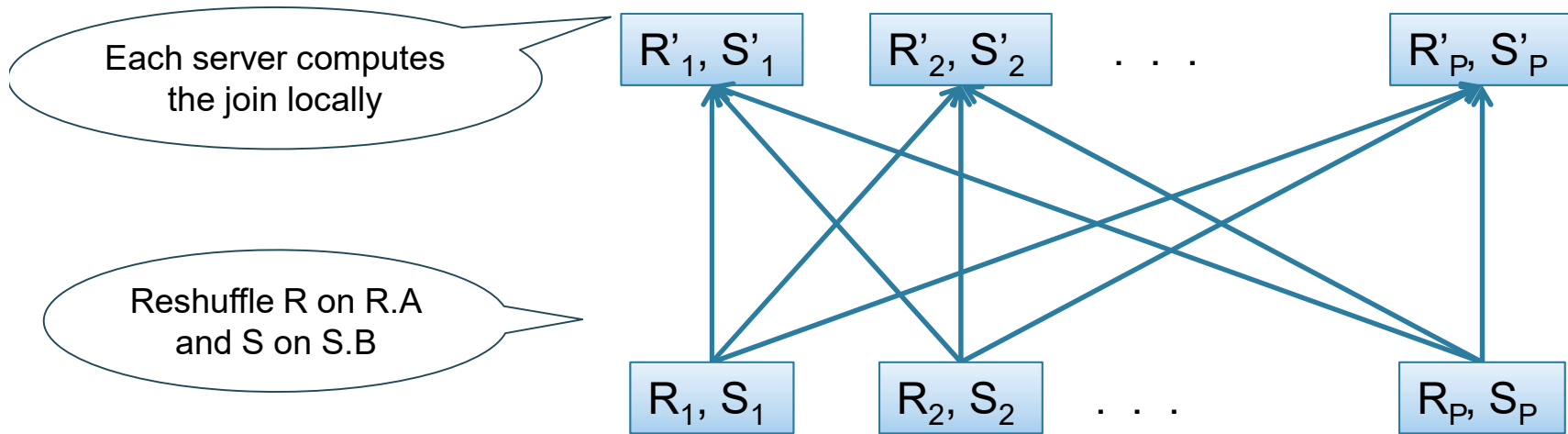
Notice: they may be stored in DFS (recall MapReduce)

Some servers hold R-chunks, some hold S-chunks, some hold both

# Hash Join: $R \bowtie_{A=B} S$

Data:  $R(A, C), S(B, D)$

Query:  $R \bowtie_{A=B} S$



Initially, R and S are block partitioned.

Notice: they may be stored in DFS (recall MapReduce)

Some servers hold R-chunks, some hold S-chunks, some hold both

# Hash Join: $R \bowtie_{A=B} S$

- Step 1
  - Every server holding any chunk of R partitions its chunk using a hash function  $h(t.A)$
  - Every server holding any chunk of S partitions its chunk using a hash function  $h(t.B)$
- Step 2:
  - Each server computes the join of its local fragment of R with its local fragment of S

# Broadcast Join A.k.a. “Small Join”

# Broadcast Join

- When joining R and S
- If  $|R| \gg |S|$ 
  - Leave R where it is
  - Replicate entire S relation across R-nodes
- Called a **small join** or a **broadcast join**

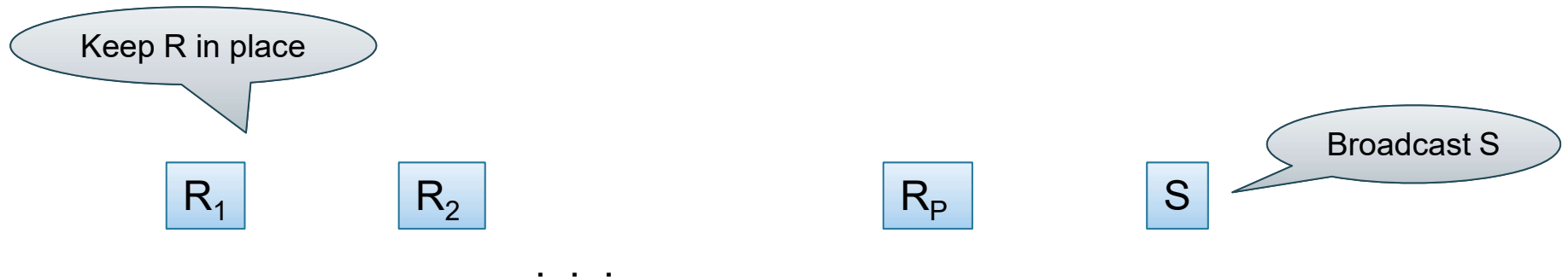
Query:  $R \bowtie S$

# Broadcast Join



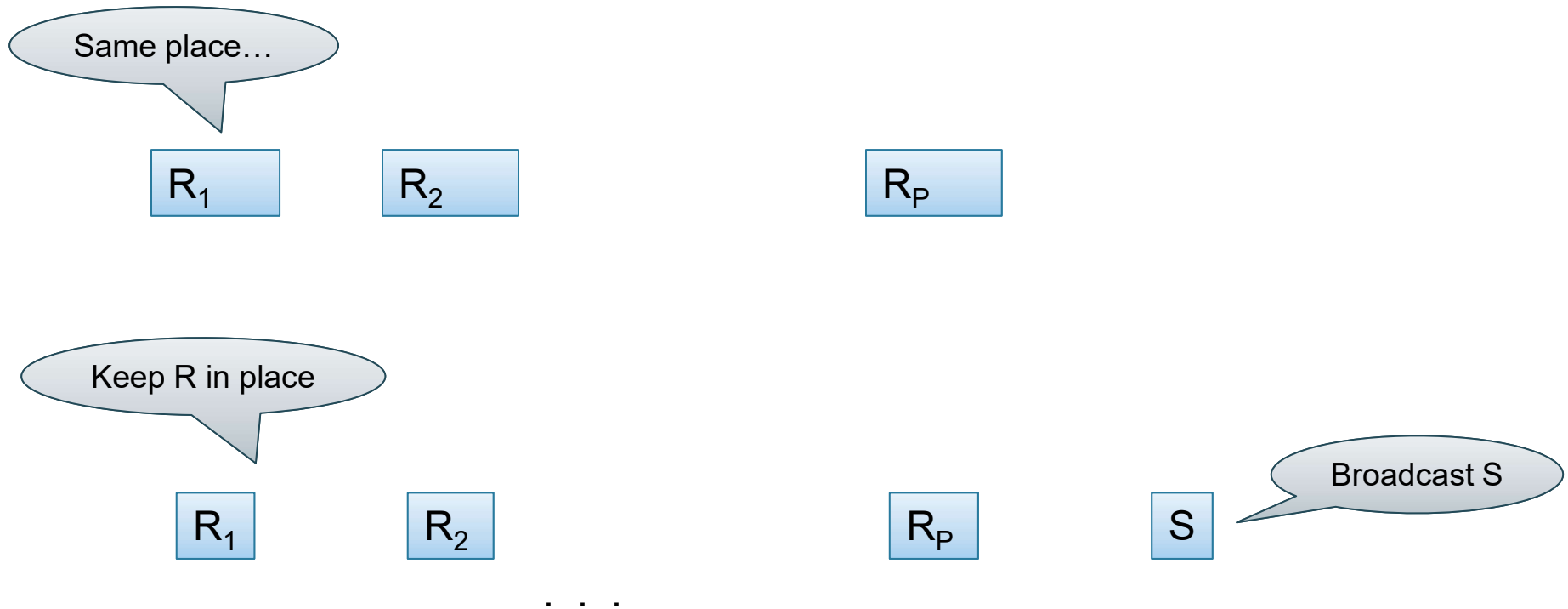
Query:  $R \bowtie S$

# Broadcast Join



Query:  $R \bowtie S$

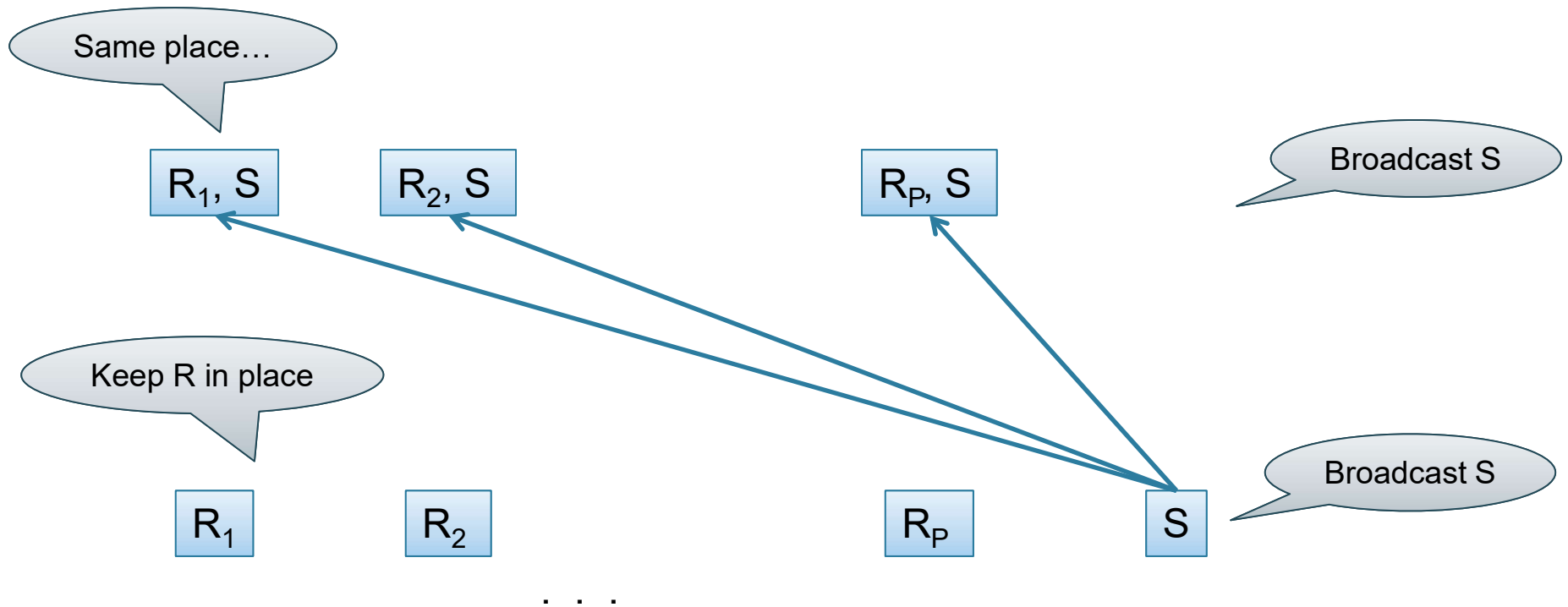
# Broadcast Join





Query:  $R \bowtie S$

# Broadcast Join

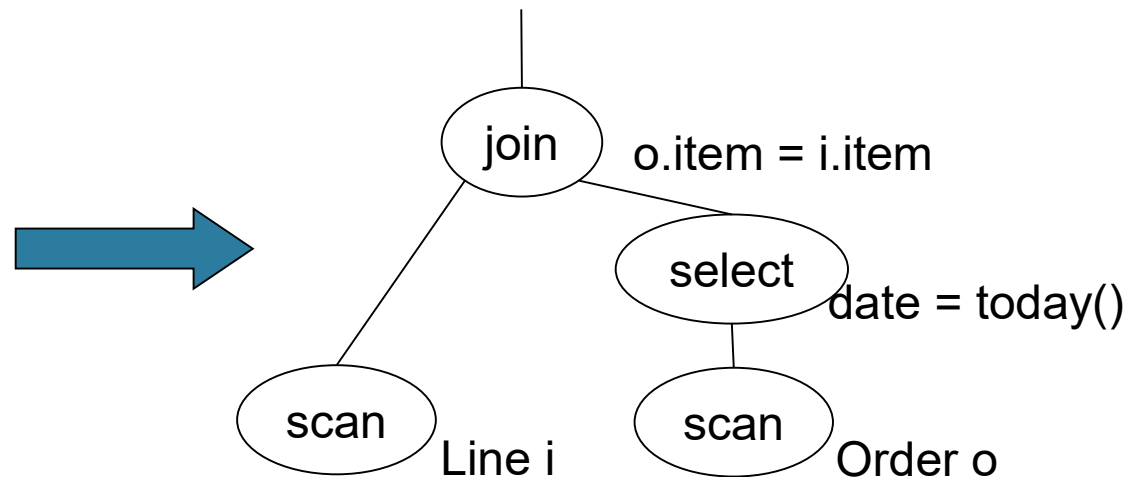


Order(oid, item, date), Line(item, ...)

# Example Query Execution

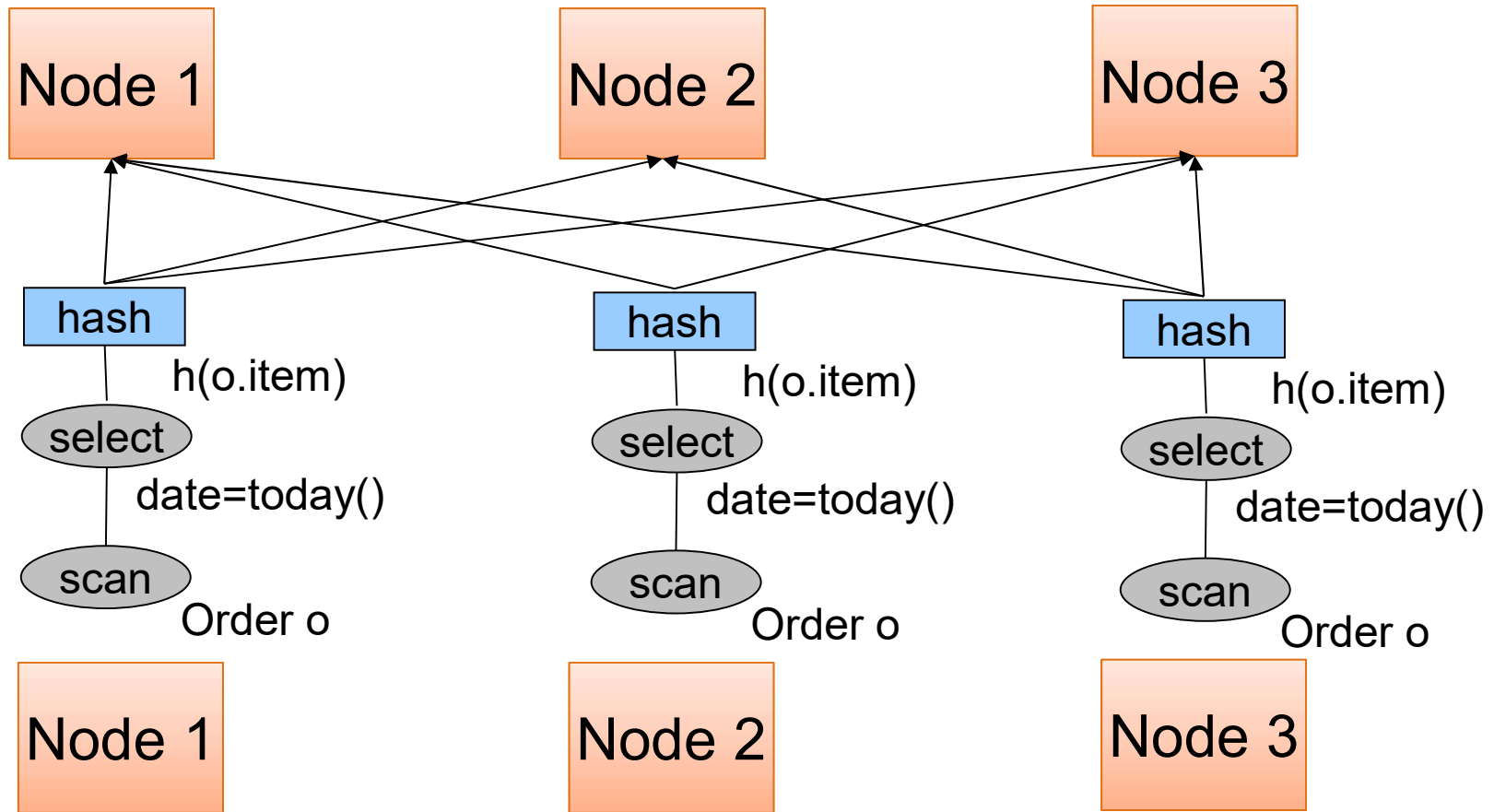
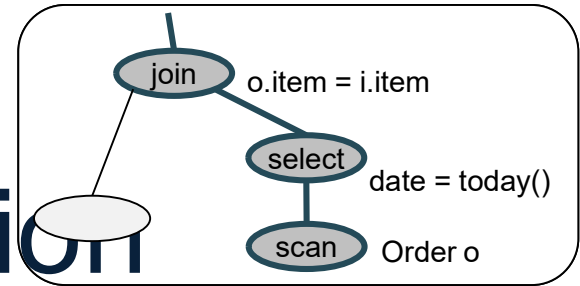
*Find all orders from today, along with the items ordered*

```
SELECT *  
FROM Order o, Line i  
WHERE o.item = i.item  
      AND o.date = today()
```



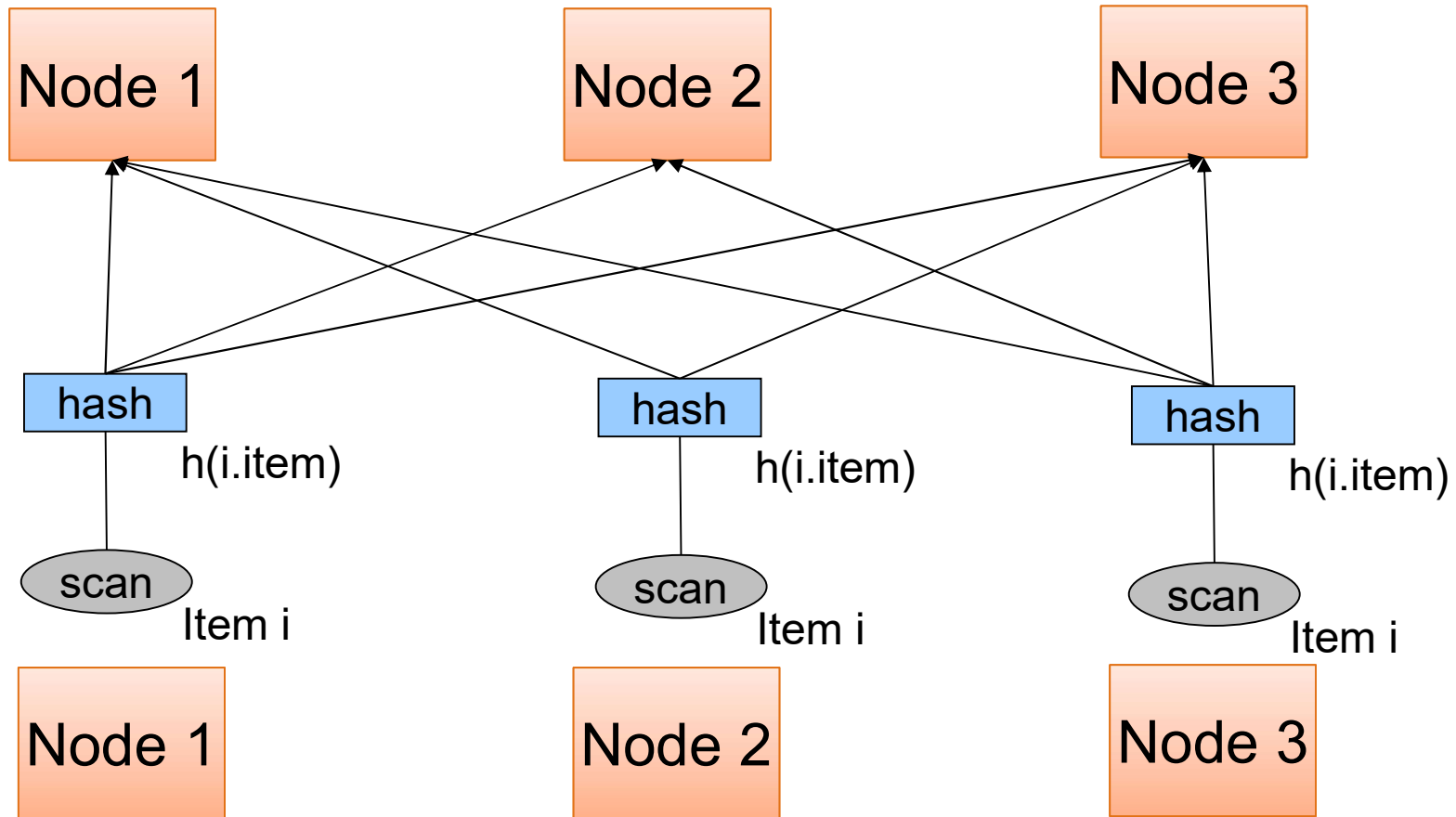
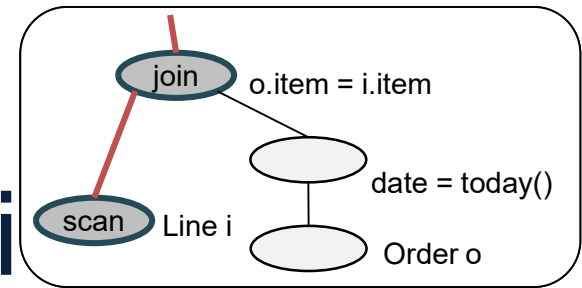
Order(oid, item, date), Line(item, ...)

# Query Execution



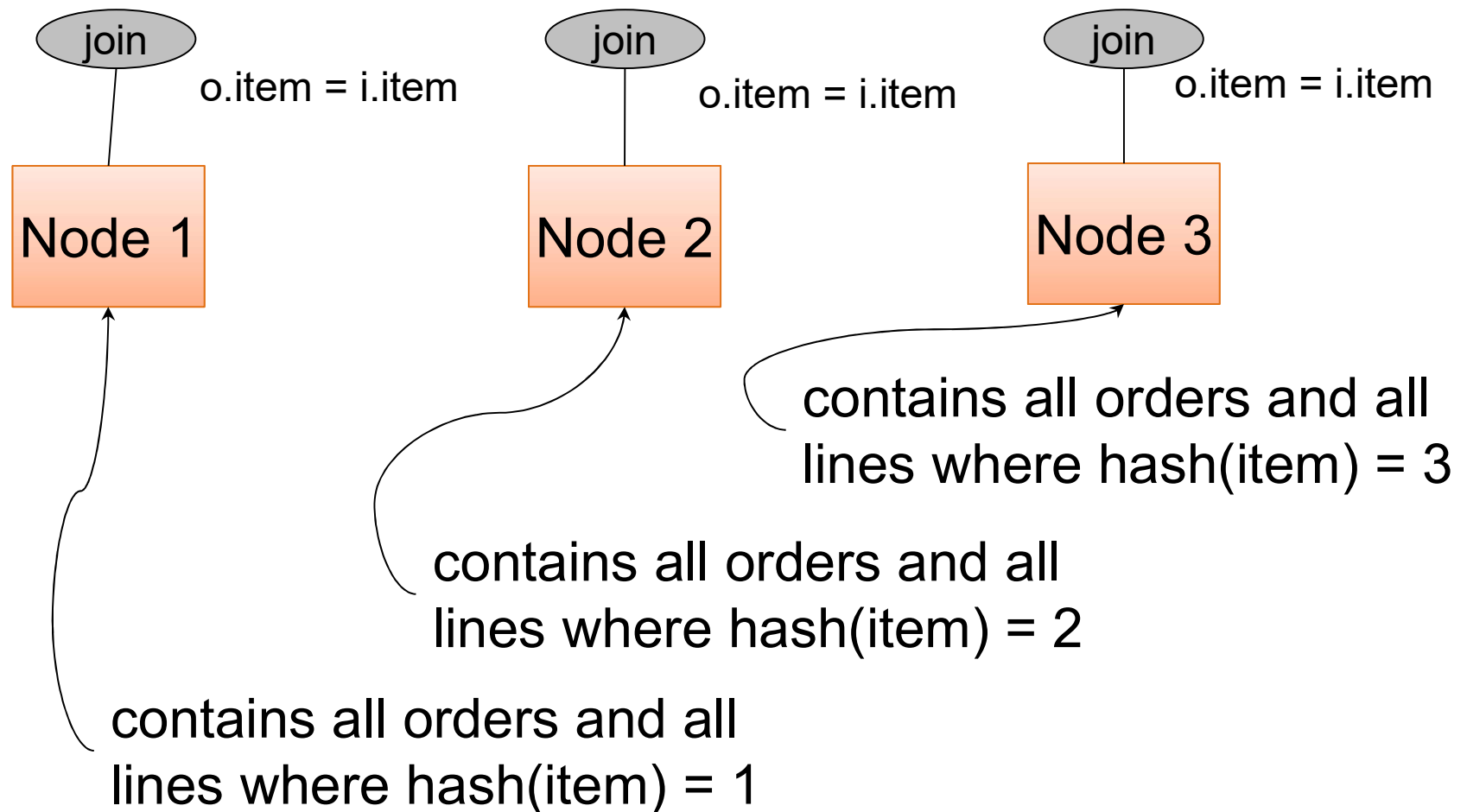
Order(oid, item, date), Line(item, ...)

# Query Executi



Order(oid, item, date), Line(item, ...)

# Query Execution



## Example 2

```
SELECT *  
FROM R, S, T  
WHERE R.b = S.c AND S.d = T.e AND (R.a - T.f) > 100
```

Machine 1

1/3 of R, S, T

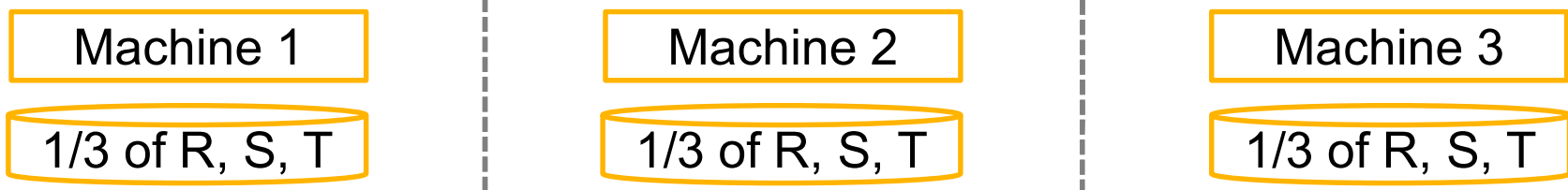
Machine 2

1/3 of R, S, T

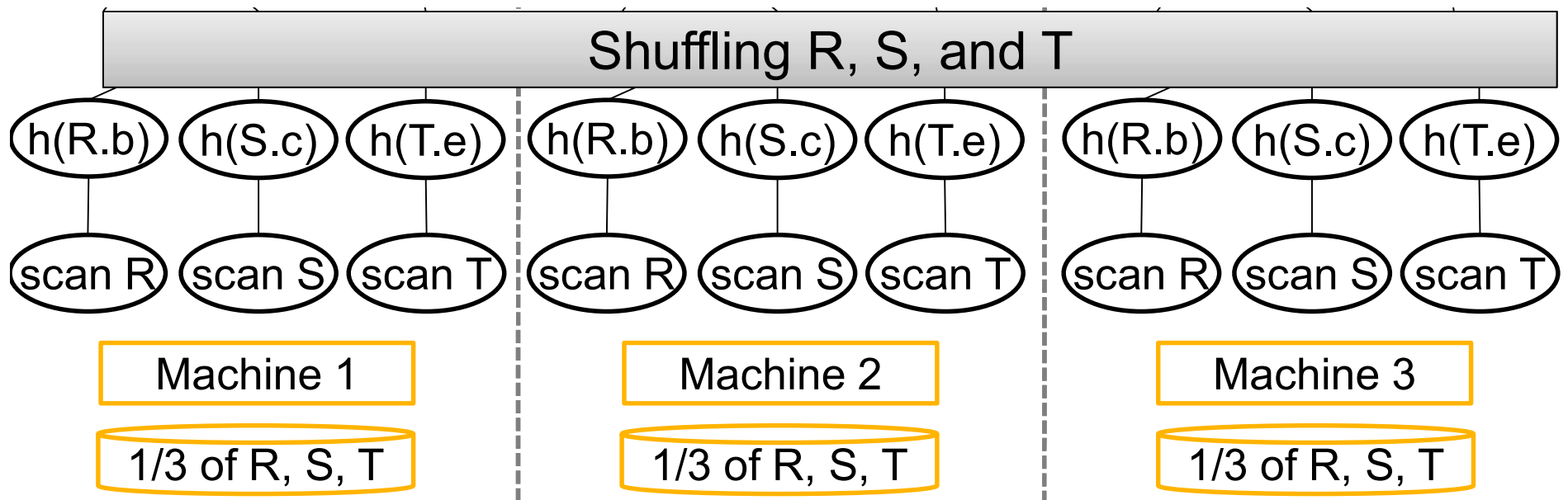
Machine 3

1/3 of R, <sup>22</sup>S, T

... WHERE R.b = S.c AND S.d = T.e AND (R.a - T.f) > 100

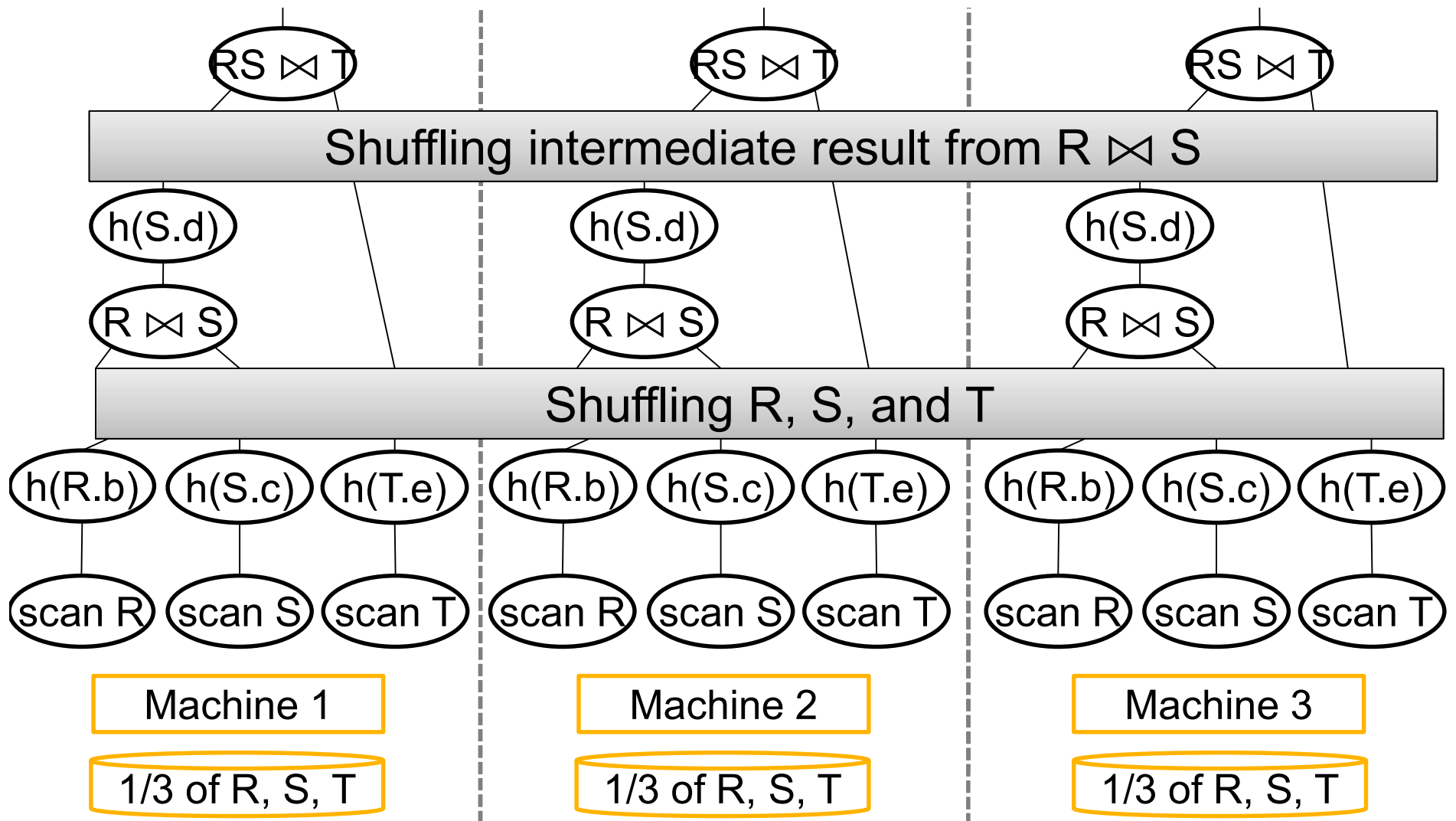


... WHERE R.b = S.c AND S.d = T.e AND (R.a - T.f) > 100

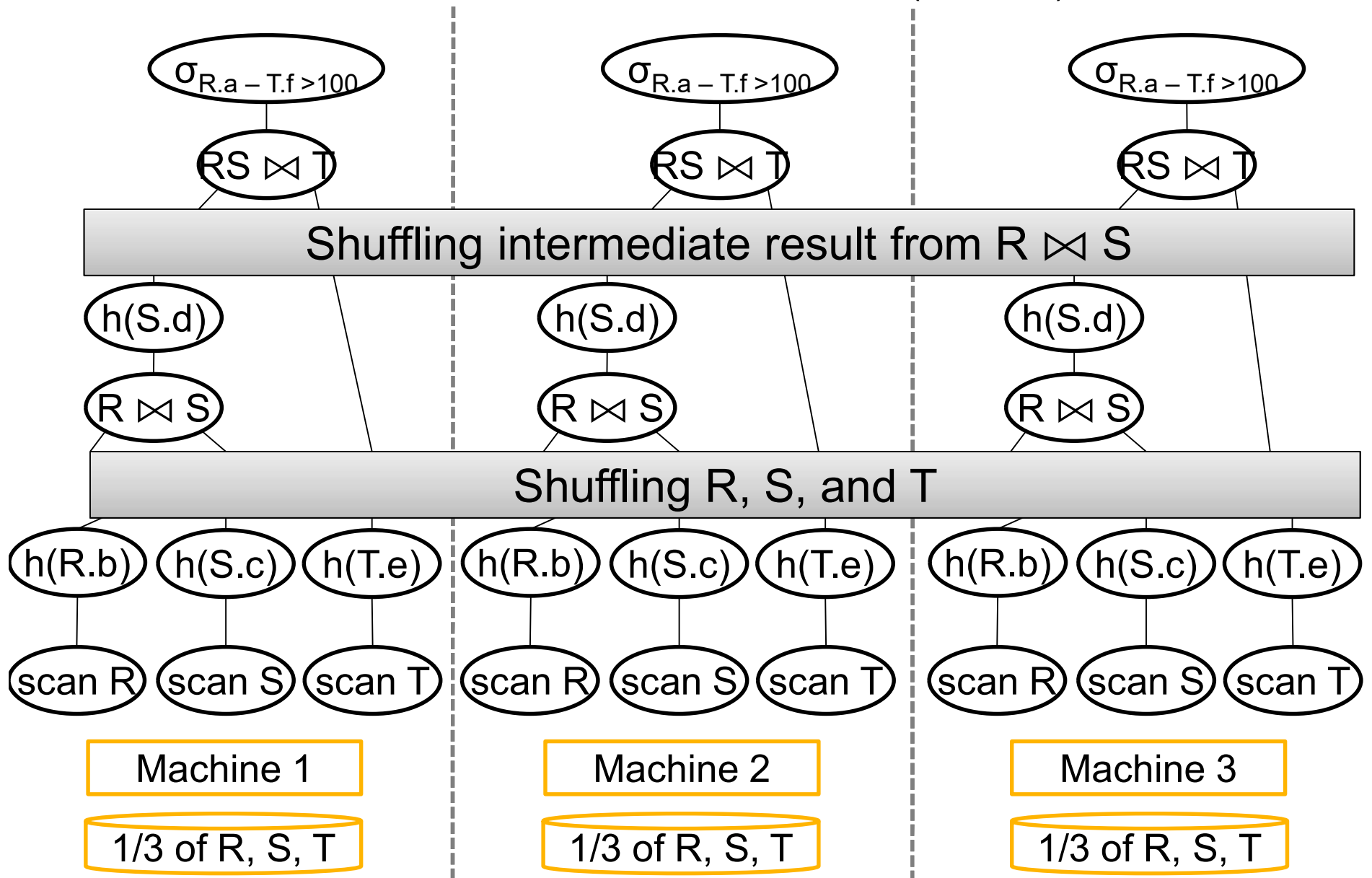




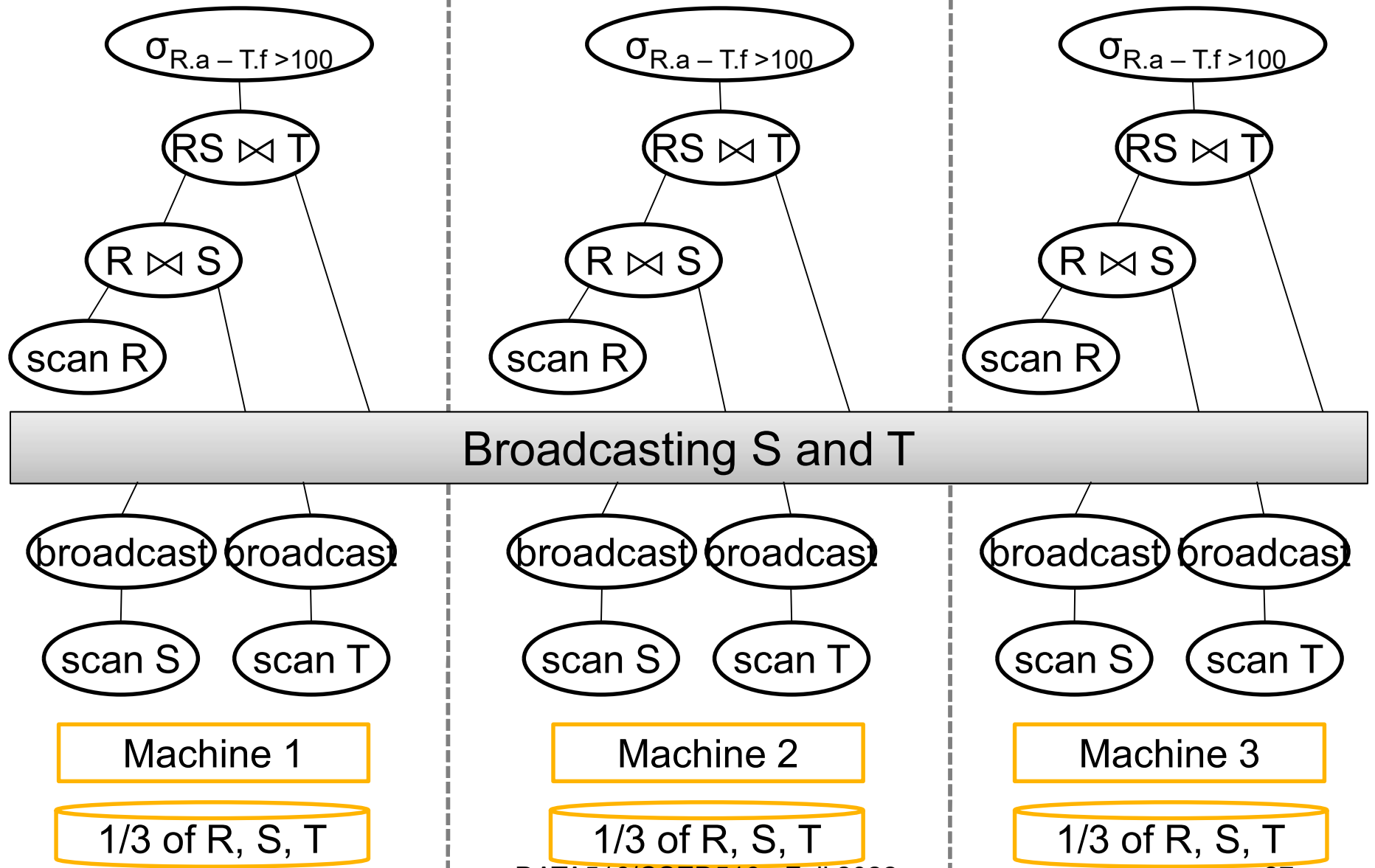
... WHERE  $R.b = S.c$  AND  $S.d = T.e$  AND  $(R.a - T.f) > 100$



... WHERE  $R.b = S.c$  AND  $S.d = T.e$  AND  $(R.a - T.f) > 100$



... WHERE  $R.b = S.c$  AND  $S.d = T.e$  AND  $(R.a - T.f) > 100$



# Discussion

- Hash-join:
  - Both relations are partitioned (**good**)
  - May have skew (**bad**)

# Discussion

- Hash-join:
  - Both relations are partitioned (**good**)
  - May have skew (**bad**)
- Broadcast join
  - One relation must be broadcast (**bad**)
  - No worry about skew (**good**)

# Discussion

- Hash-join:
  - Both relations are partitioned (**good**)
  - May have skew (**bad**)
- Broadcast join
  - One relation must be broadcast (**bad**)
  - No worry about skew (**good**)
- Skew join (has other names):
  - Combine both: in class

# Outline

- Distributed Joins

- Skew

- Parallel Query Processing Wrap-up

- Graph

# Skew



# Skew

- Skew means that one server runs much longer than the other servers
- Reasons:
  - Computation skew
  - Data skew

# Computation Skew

- All workers receive the same amount of input data, but some need to run much longer than others
- E.g. perform some image processing whose runtimes depends on the image
- Solution: use virtual servers

# Virtual Servers

Main idea:

- If we send the data uniformly to the  $P$  servers, and one of them is stuck with the complicated image, then we have skew
- Solution: Pretend we have many “virtual” servers

# Virtual Servers

Large number  $P_v$  of “virtual servers”

- Design algorithm for  $P_v$  virtual servers
- Scale down to  $P \ll P_v$  physical servers, by simulating them round-robin

E.g. MapReduce:  $P$ =workers,  $P_v$ =map tasks

# Data Skew

- We fail to distribute the data uniformly to the servers
- Question: why can this happen?

# Data Skew

- We fail to distribute the data uniformly to the servers
- Question: why can this happen?
- Answer:
  - Range partition may have many more tuples in one bucket than another
  - Hash partition may suffer from heavy hitters

# Data Skew

Assume we hash-partition data items

- All records with same partition key are sent to the same server
- Heavy hitter

# Analyzing Heavy Hitters

- How many times can an item occur before it is called a “heavy hitter”?
- The answer requires a deep analysis of what a good hash function can do.



# Problem Statement

Given: **N** distinct data items  $v_1, \dots, v_N$

- We hash-partition them to **P** nodes
- How uniform is the partition?

# Discussion

There is no **deterministic** “good” hash function:

- For any hash function  $h$ ,  
there exists distinct data items  $v_1, \dots, v_N$   
s.t. all are mapped to the same value:  
 $h(v_1)=h(v_2)= \dots h(v_N)$

# Discussion

There is no **deterministic** “good” hash function:

- For any hash function  $h$ ,  
there exists distinct data items  $v_1, \dots, v_N$   
s.t. all are mapped to the same value:  
 $h(v_1)=h(v_2)= \dots h(v_N)$

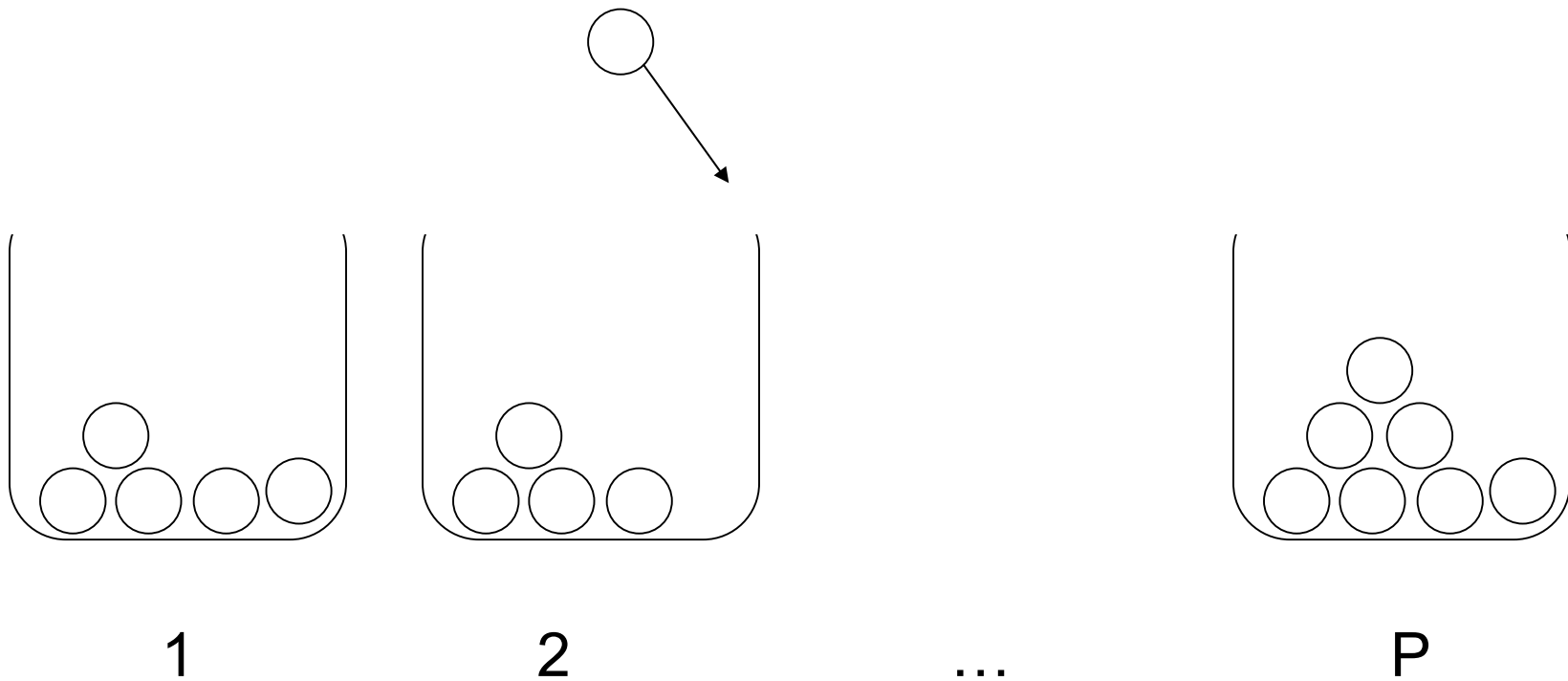
A “good” hash function is **random** function:

- Intuition: every day we choose another  $h$ , we  
want the partition to be uniform in *expectation*

# Balls into Bins

Probability of a ball getting into bin #5:

We throw  $N$  balls randomly into  $P$  bins:

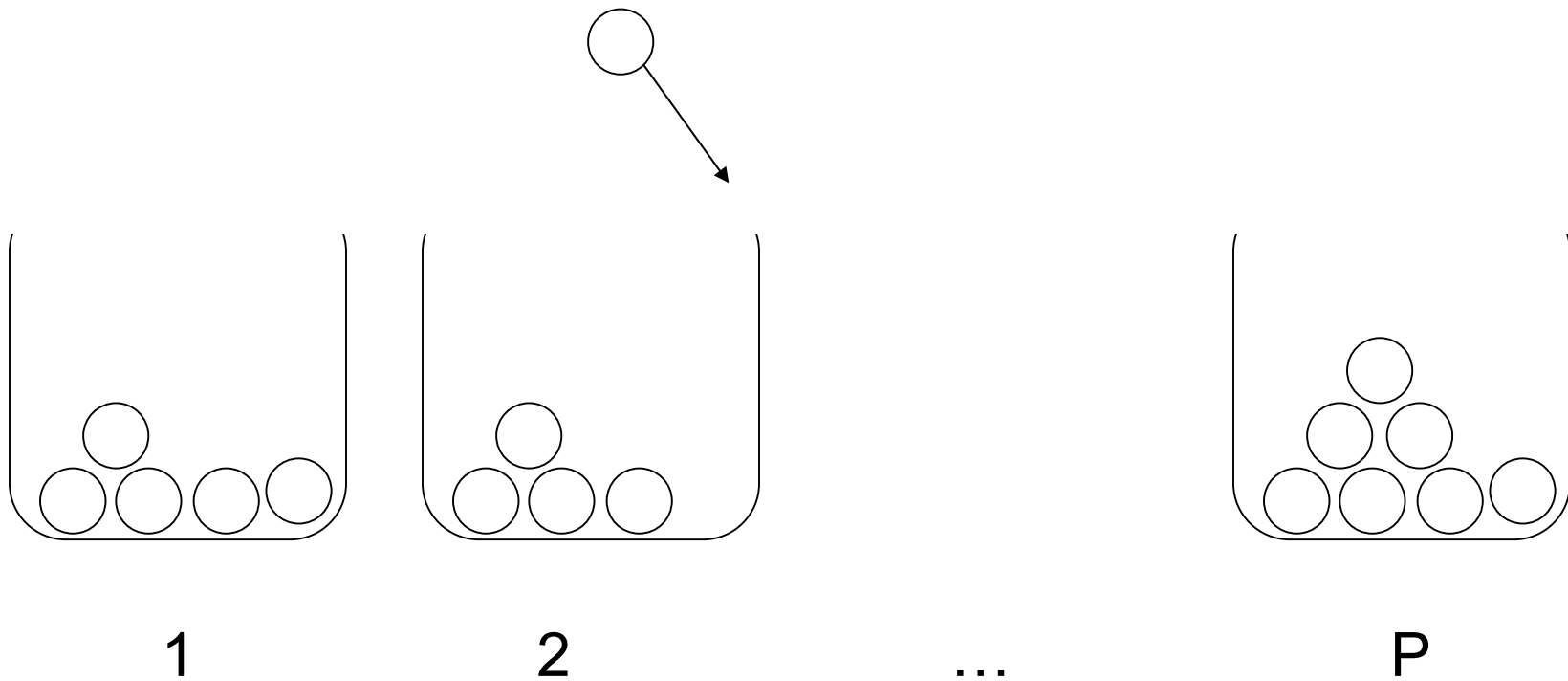


# Balls into Bins

Probability of a ball getting into bin #5:

$$\frac{1}{P}$$

We throw  $N$  balls randomly into  $P$  bins:



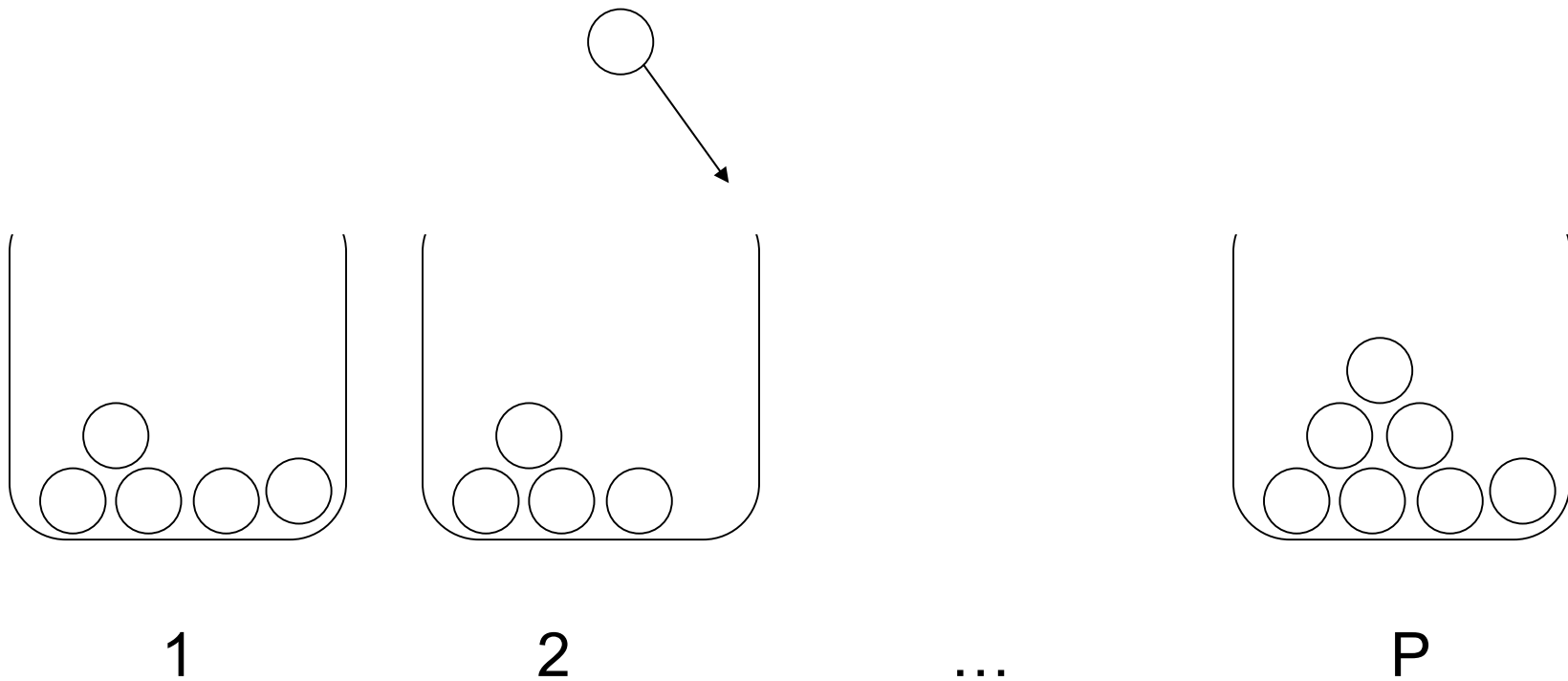
# Balls into Bins

Probability of a ball getting into bin #5:

$$\frac{1}{P}$$

We throw  $N$  balls randomly into  $P$  bins:

Expected size of bin #5:



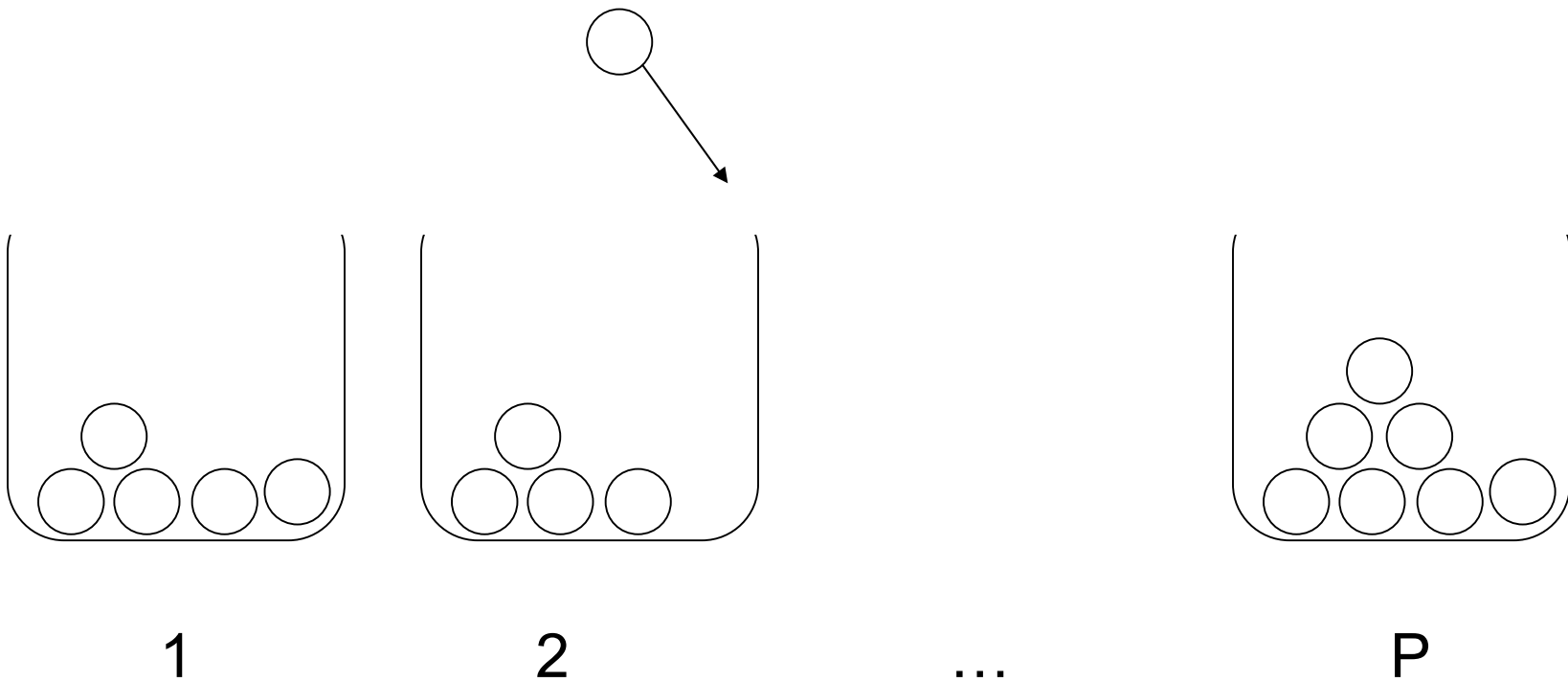
# Balls into Bins

We throw  $N$  balls randomly into  $P$  bins:

Probability of a ball getting into bin #5:

Expected size of bin #5:

$$\frac{1}{P}$$
$$\frac{N}{P}$$



# Balls into Bins

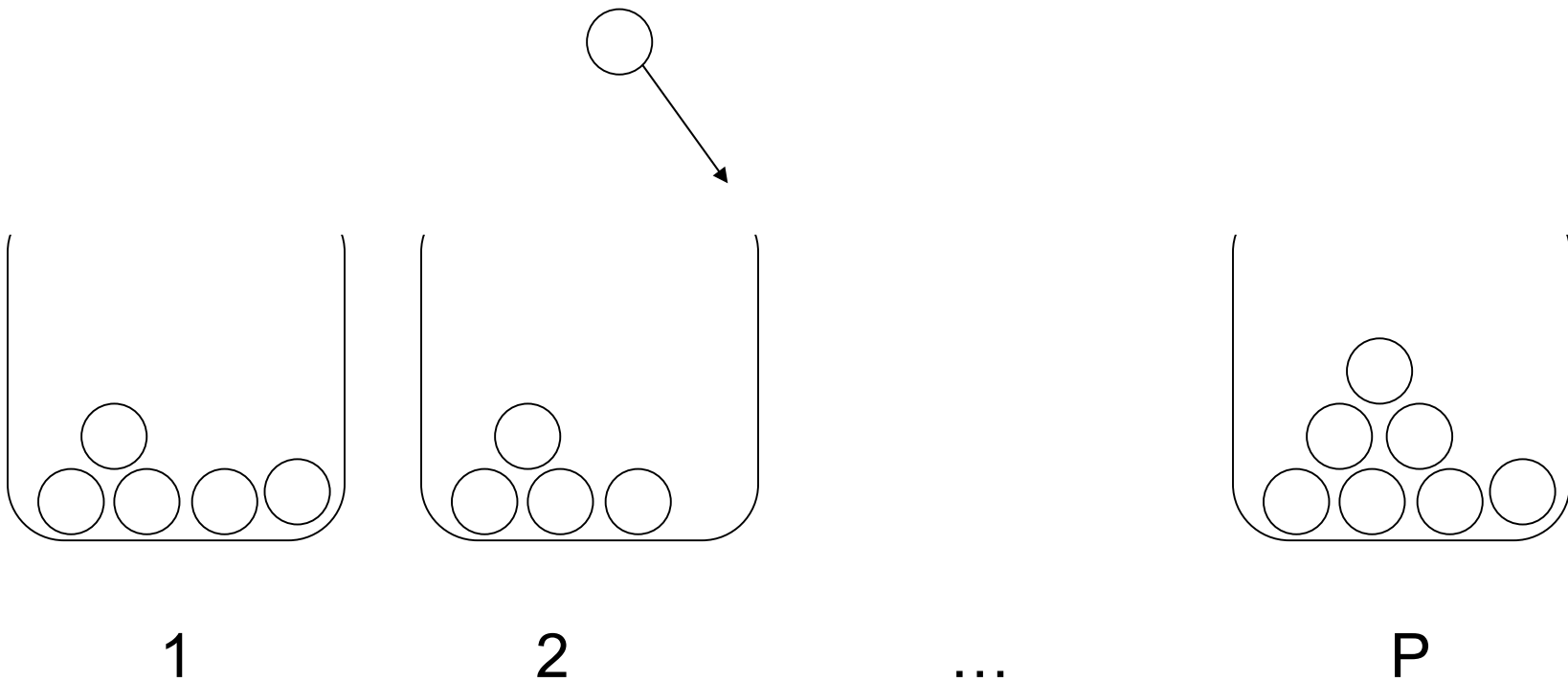
We throw  $N$  balls randomly into  $P$  bins:

Probability of a ball getting into bin #5:

Expected size of bin #5:

Expected size of the max bin:

$$\frac{1}{P}$$
$$\frac{N}{P}$$





# Balls into Bins

We throw  $N$  balls randomly into  $P$  bins:

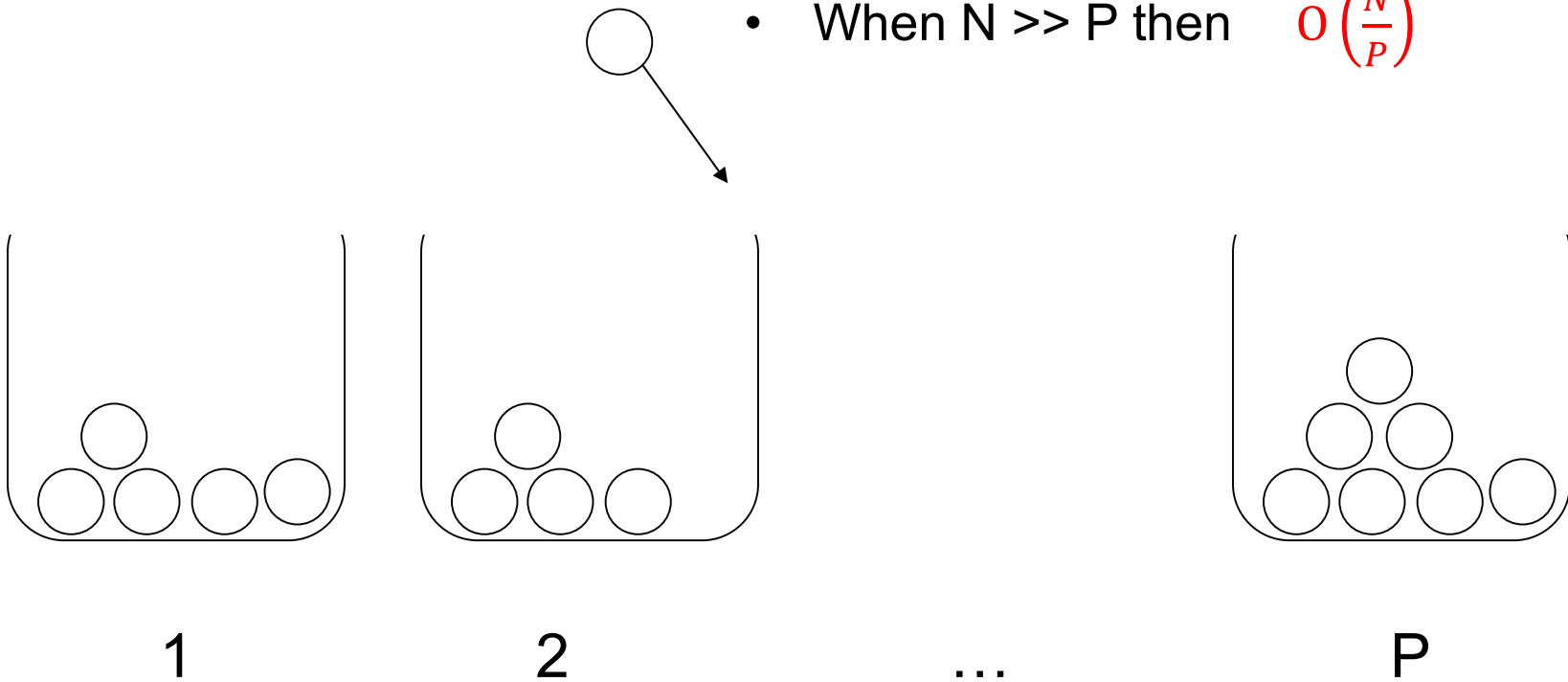
Probability of a ball getting into bin #5:

Expected size of bin #5:

Expected size of the max bin:

- When  $N \gg P$  then  $O\left(\frac{N}{P}\right)$

$$\frac{1}{P}$$
$$\frac{N}{P}$$



# Balls into Bins

We throw  $N$  balls randomly into  $P$  bins:

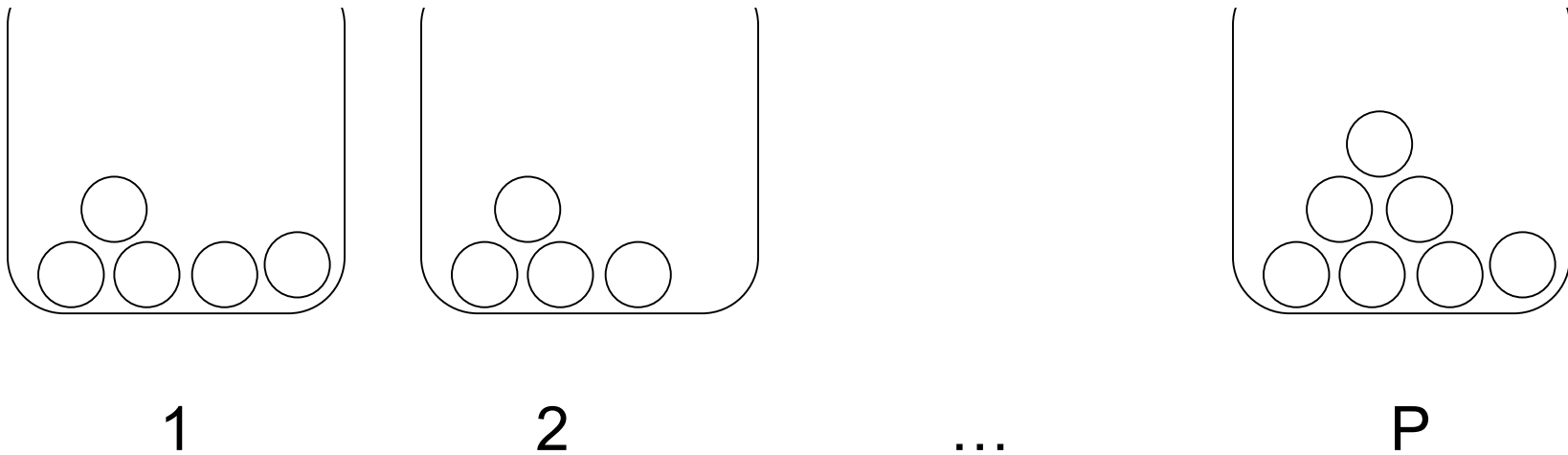
Probability of a ball getting into bin #5:

Expected size of bin #5:

Expected size of the max bin:

- When  $N \gg P$  then  $O\left(\frac{N}{P}\right)$
- When  $N=P$  then  $\log N$

$$\frac{1}{P}$$
$$\frac{N}{P}$$

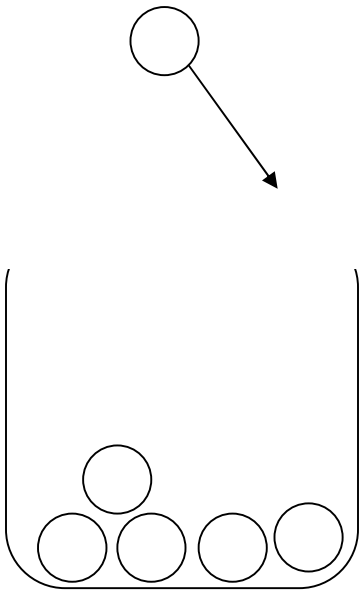


# Discussion

- Hash-partition is like throwing  $N$  balls into  $P$  bins
- Partition is uniform when max load is approx  $N/P$
- To analyze the max load, lets analyze the load of one fixed bin

# One Bin

**N** data items  $v_1, \dots, v_N$



One fixed bin

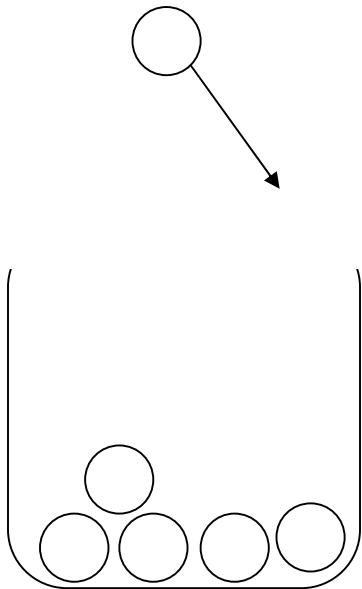
# One Bin

**N** data items  $v_1, \dots, v_N$

For each data item  $v_i$ , let  $X_i$  be a r.v. s.t.:

$X_i=1$  if item  $v_i$  is sent to our bin

$X_i=0$  if item  $v_i$  is sent to a different bin



One fixed bin

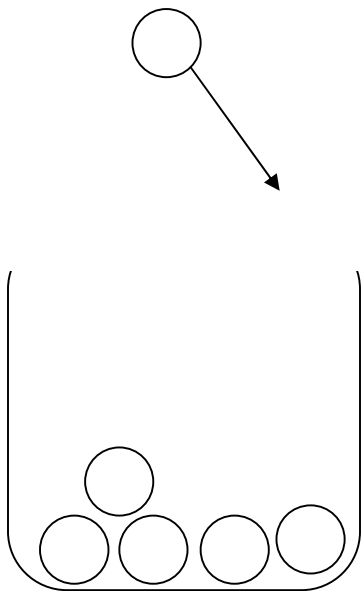
# One Bin

**N** data items  $v_1, \dots, v_N$

For each data item  $v_i$ , let  $X_i$  be a r.v. s.t.:

$X_i=1$  if item  $v_i$  is sent to our bin

$X_i=0$  if item  $v_i$  is sent to a different bin



One fixed bin

$X_i \in \{0,1\}$  is a Bernoulli random variable

$$\Pr(X_i = 1) = 1/P$$

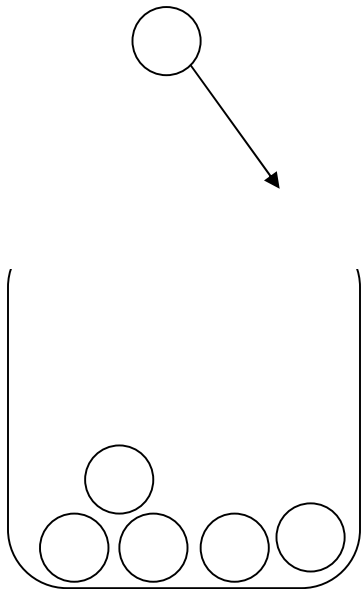
# One Bin

**N** data items  $v_1, \dots, v_N$

For each data item  $v_i$ , let  $X_i$  be a r.v. s.t.:

$X_i=1$  if item  $v_i$  is sent to our bin

$X_i=0$  if item  $v_i$  is sent to a different bin



One fixed bin

$X_i \in \{0,1\}$  is a Bernoulli random variable

$$\Pr(X_i = 1) = 1/P$$

Load of the bin is:  $Y = X_1 + X_2 + \dots + X_N$

# The Chernoff Bound

Note:  
very many  
variants

Bernoulli r.v.:  $X_1, \dots, X_N \in \{0,1\}$

For all  $i$ ,  $\Pr(X_i = 1) = \mu \in (0,1)$

We are interested in  $Y = X_1 + X_2 + \dots + X_N$



# The Chernoff Bound

Note:  
very many  
variants

Bernoulli r.v.:  $X_1, \dots, X_N \in \{0,1\}$

For all  $i$ ,  $\Pr(X_i = 1) = \mu \in (0,1)$

We are interested in  $Y = X_1 + X_2 + \dots + X_N$

**Fact:**  $E[Y] = N\mu$

# The Chernoff Bound

Note:  
very many  
variants

Bernoulli r.v.:  $X_1, \dots, X_N \in \{0,1\}$

For all  $i$ ,  $\Pr(X_i = 1) = \mu \in (0,1)$

We are interested in  $Y = X_1 + X_2 + \dots + X_N$

**Fact:**  $E[Y] = N\mu$

**Theorem** (Chernoff bound). If they are iid then:

$$\Pr(Y > (1 + \delta)E[Y]) \leq \exp\left(-\frac{\delta^2}{3}E[Y]\right)$$

# Application to Hash Partition

**N** data items  $v_1, \dots, v_N$

Distribute on **P** servers

1

2

P

Fix one server **j**

Indicator variables:

$$X_i = [h(v_i) = j]$$

$$\Pr(X_i = 1) = 1/P$$

# Application to Hash Partition

**Load of server  $j$ :**  $\text{Load}(j) = X_1 + X_2 + \cdots + X_N$

**Expected load:**  $E[\text{Load}(j)] = \frac{N}{P}$

# Application to Hash Partition

**Load of server  $j$ :**  $\text{Load}(j) = X_1 + X_2 + \cdots + X_N$

**Expected load:**  $E[\text{Load}(j)] = \frac{N}{P}$

Skew at  $j$

**Cernoff:**  $\Pr\left(\text{Load}(j) > (1 + \delta) \frac{N}{P}\right) \leq \exp\left(-\frac{\delta^2 N}{3P}\right)$

# Application to Hash Partition

**Load of server  $j$ :**  $\text{Load}(j) = X_1 + X_2 + \dots + X_N$

**Expected load:**  $E[\text{Load}(j)] = \frac{N}{P}$

Skew at  $j$

**Cernoff:**  $\Pr\left(\text{Load}(j) > (1 + \delta) \frac{N}{P}\right) \leq \exp\left(-\frac{\delta^2 N}{3P}\right)$

**Union bound:**  $\Pr(\text{Skew}) \leq P \cdot \exp\left(-\frac{\delta^2 N}{3P}\right)$

Skew at 1 or at 2 ... or at  $P$

# Discussion

- We have not computed the expected value of the maximum load
- Instead, we have computed the probability that we exceed the expected load by more than  $\delta$

# Example 1

$$\Pr \left( \text{for some } j: \text{Load}(j) > (1 + \delta) \frac{N}{P} \right) \leq P \cdot \exp \left( -\frac{\delta^2 N}{3 P} \right)$$

N=2,000,000 items, P=100 servers:

We expect: L = 20,000 items/server



# Example 1

$$\Pr \left( \text{for some } j: \text{Load}(j) > (1 + \delta) \frac{N}{P} \right) \leq P \cdot \exp \left( -\frac{\delta^2 N}{3 P} \right)$$

N=2,000,000 items, P=100 servers:

We expect: L = 20,000 items/server

What is the prob that some server exceeds L by > 10%?

# Example 1

$$\Pr \left( \text{for some } j: \text{Load}(j) > (1 + \delta) \frac{N}{P} \right) \leq P \cdot \exp \left( -\frac{\delta^2 N}{3 P} \right)$$

N=2,000,000 items, P=100 servers:

We expect: L = 20,000 items/server

What is the prob that some server exceeds L by > 10%?

$$\Pr(\text{bad}) \leq$$

# Example 1

$$\Pr \left( \text{for some } j: \text{Load}(j) > (1 + \delta) \frac{N}{P} \right) \leq P \cdot \exp \left( -\frac{\delta^2 N}{3 P} \right)$$

N=2,000,000 items, P=100 servers:

We expect: L = 20,000 items/server

What is the prob that some server exceeds L by > 10%?

$$\Pr(\text{bad}) \leq 100 \times \exp \left( -\frac{0.1^2}{3} \times 20000 \right)$$

# Example 1

$$\Pr \left( \text{for some } j: \text{Load}(j) > (1 + \delta) \frac{N}{P} \right) \leq P \cdot \exp \left( -\frac{\delta^2 N}{3 P} \right)$$

N=2,000,000 items, P=100 servers:

We expect: L = 20,000 items/server

What is the prob that some server exceeds L by > 10%?

$$\Pr(\text{bad}) \leq 100 \times \exp \left( -\frac{0.1^2}{3} \times 20000 \right) = 100 \times \exp(-66)$$

# Example 1

$$\Pr \left( \text{for some } j: \text{Load}(j) > (1 + \delta) \frac{N}{P} \right) \leq P \cdot \exp \left( -\frac{\delta^2 N}{3 P} \right)$$

N=2,000,000 items, P=100 servers:

We expect: L = 20,000 items/server

What is the prob that some server exceeds L by > 10%?

$$\Pr(\text{bad}) \leq 100 \times \exp \left( -\frac{0.1^2}{3} \times 20000 \right) = 100 \times \exp(-66)$$

Virtually zero!

## Example 2

$$\Pr \left( \text{for some } j: \text{Load}(j) > (1 + \delta) \frac{N}{P} \right) \leq P \cdot \exp \left( -\frac{\delta^2 N}{3 P} \right)$$

N=2,000,000 items, P=10,000 servers:

## Example 2

$$\Pr \left( \text{for some } j: \text{Load}(j) > (1 + \delta) \frac{N}{P} \right) \leq P \cdot \exp \left( -\frac{\delta^2 N}{3 P} \right)$$

N=2,000,000 items, P=10,000 servers:

We expect: L = 200 items/server

What is the prob that some server exceeds L by > 10%?

## Example 2

$$\Pr \left( \text{for some } j: \text{Load}(j) > (1 + \delta) \frac{N}{P} \right) \leq P \cdot \exp \left( -\frac{\delta^2 N}{3 P} \right)$$

N=2,000,000 items, P=10,000 servers:

We expect: L = 200 items/server

What is the prob that some server exceeds L by > 10%?

$Pr(\text{bad}) \leq$



## Example 2

$$\Pr \left( \text{for some } j: \text{Load}(j) > (1 + \delta) \frac{N}{P} \right) \leq P \cdot \exp \left( -\frac{\delta^2 N}{3 P} \right)$$

N=2,000,000 items, P=10,000 servers:

We expect: L = 200 items/server

What is the prob that some server exceeds L by > 10%?

$$\Pr(\text{bad}) \leq 10000 \times \exp \left( -\frac{0.1^2}{3} \times 200 \right)$$

## Example 2

$$\Pr \left( \text{for some } j: \text{Load}(j) > (1 + \delta) \frac{N}{P} \right) \leq P \cdot \exp \left( -\frac{\delta^2 N}{3 P} \right)$$

N=2,000,000 items, P=10,000 servers:

We expect: L = 200 items/server

What is the prob that some server exceeds L by > 10%?

$$\Pr(\text{bad}) \leq 10000 \times \exp \left( -\frac{0.1^2}{3} \times 200 \right) = 10000 \times \exp \left( -\frac{2}{3} \right) =$$

5134

## Example 2

$$\Pr \left( \text{for some } j: \text{Load}(j) > (1 + \delta) \frac{N}{P} \right) \leq P \cdot \exp \left( -\frac{\delta^2 N}{3 P} \right)$$

N=2,000,000 items, P=10,000 servers:

We expect: L = 200 items/server

What is the prob that some server exceeds L by > 10%?

$$\Pr(\text{bad}) \leq 10000 \times \exp \left( -\frac{0.1^2}{3} \times 200 \right) = 10000 \times \exp \left( -\frac{2}{3} \right) =$$

5134

>>1

Skew almost certain

# Main Take-away

$$\Pr(\text{Skew}) \leq P \cdot \exp\left(-\frac{\delta^2 N}{3P}\right)$$

To avoid skew, we need  $N \gg P$

# Application to Heavy Hitters

**N** data items  $v_1, \dots, v_N$  some are **repeated**

- We hash-partition them to **P** nodes
- How uniform is the partition?

# Application to Heavy Hitters

- Assume case: each item  $t$  times
- $N/t$  distinct items

$$\underbrace{v_1, v_1, \dots, v_1}_t, \underbrace{v_2, v_2, \dots, v_2}_t, \dots, \dots, \underbrace{v_{N/t}, v_{N/t}, \dots, v_{N/t}}_t$$

# Application to Heavy Hitters

- Assume case: each item  $t$  times
- $N/t$  distinct items

$$\underbrace{v_1, v_1, \dots, v_1}_t, \underbrace{v_2, v_2, \dots, v_2}_t, \dots, \dots, \underbrace{v_{N/t}, v_{N/t}, \dots, v_{N/t}}_t$$

- $N/t$  iid Bernoulli variables:

$$X_1, X_2, \dots, X_{N/t}, \quad \Pr(X_i = 1) = \frac{1}{P}$$

# Application to Heavy Hitters

- Assume case: each item  $t$  times
- $N/t$  distinct items

$$\underbrace{v_1, v_1, \dots, v_1}_t, \underbrace{v_2, v_2, \dots, v_2}_t, \dots, \dots, \underbrace{v_{N/t}, v_{N/t}, \dots, v_{N/t}}_t$$

- $N/t$  iid Bernoulli variables:

$$X_1, X_2, \dots, X_{N/t}, \quad \Pr(X_i = 1) = \frac{1}{P}$$

- Load at a fixed server is  $t \times Y$  where:

$$Y = (X_1 + X_2 + \dots + X_{N/t}), \quad E[Y] = \frac{N}{t} \frac{1}{P} = \frac{N}{tP}$$



# Application to Heavy Hitters

$$Y = (X_1 + X_2 + \cdots + X_{N/t}), \quad E[Y] = \frac{N}{t} \frac{1}{P} = \frac{N}{tP}$$

# Application to Heavy Hitters

$$Y = (X_1 + X_2 + \cdots + X_{N/t}), \quad E[Y] = \frac{N}{t} \frac{1}{P} = \frac{N}{tP}$$

We apply Chernoff:

- Skew at one, fixed server:

$$\Pr(Y > (1 + \delta)E[Y]) \leq \exp\left(-\frac{\delta^2}{3}E[Y]\right) = \exp\left(-\frac{\delta^2}{3}\frac{N}{tP}\right)$$

# Application to Heavy Hitters

$$Y = (X_1 + X_2 + \cdots + X_{N/t}), \quad E[Y] = \frac{N}{t} \frac{1}{P} = \frac{N}{tP}$$

We apply Chernoff:

- Skew at one, fixed server:

$$\Pr(Y > (1 + \delta)E[Y]) \leq \exp\left(-\frac{\delta^2}{3}E[Y]\right) = \exp\left(-\frac{\delta^2}{3}\frac{N}{tP}\right)$$

- Skew at any server

$$\Pr(\text{skew}) \leq P \exp\left(-\frac{\delta^2}{3}\frac{N}{tP}\right)$$

# Main Take-away

$$\Pr(\text{skew}) \leq P \exp\left(-\frac{\delta^2 N}{3 tP}\right)$$

To avoid skew, we need  $t \ll N/P$

# Discussion

- Many distributed query processors do not handle data skew well
- (Project idea: how does your favorite engine handle skewed data?)
- In practice, you may need to partition skewed data manually

# Outline

- Distributed Joins
- Skew
- Parallel Query Processing Wrap-up
- Graph

# Parallel Query Processing Wrap-up

# Recap: Parallel Architectures:

1.

2.

3.



# Recap: Parallel Architectures:

1. Shared Memory
2. Shared Disk
3. Shared Nothing – aka distributed

# Recap: Motivation

- Discuss when to use distributed data processing v.s. single server
- [in class]

# Recap: Explain these terms

- Speedup v.s. Scaleup

# Recap:

## Horizontal Data Partitioning

Describe three strategies:

1.

2.

3.

# Recap:

## Horizontal Data Partitioning

Describe three strategies:

1. Block partition
2. Hash partition
3. Range partition

# Recap: Distributed Join

Describe/discuss these algorithms:

1. Parallel Hash Join
2. Broadcast join, a.k.a. small join

# Conclusion

- Distributed data processing:
  - Spread the data to fit in main memory
  - Take advantage of parallelism
- “SQL is embarrassingly parallel”
  - Relational algebra: easy to parallelize
  - Hash-based algorithm suffer from skew

# Outline

- Distributed Joins
- Skew
- Parallel Query Processing Wrap-up
- Graph



# Graphs

# Graph Processing Motivation

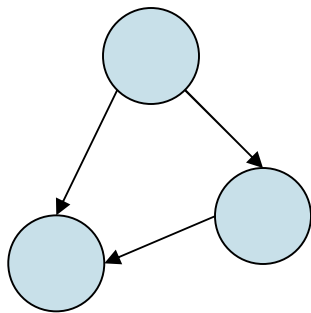
- Many apps need to do analytics on graphs
  - Web graph
  - Social networks
  - Transportation routes
  - Citation graphs
  - Disease propagation graphs
  - ...
- A graph:  $G(V,E)$ 
  - V: Vertices in the graph
  - E: Edges between the vertices
  - Large graph means many edges, not many gigabytes

# Graph Analysis

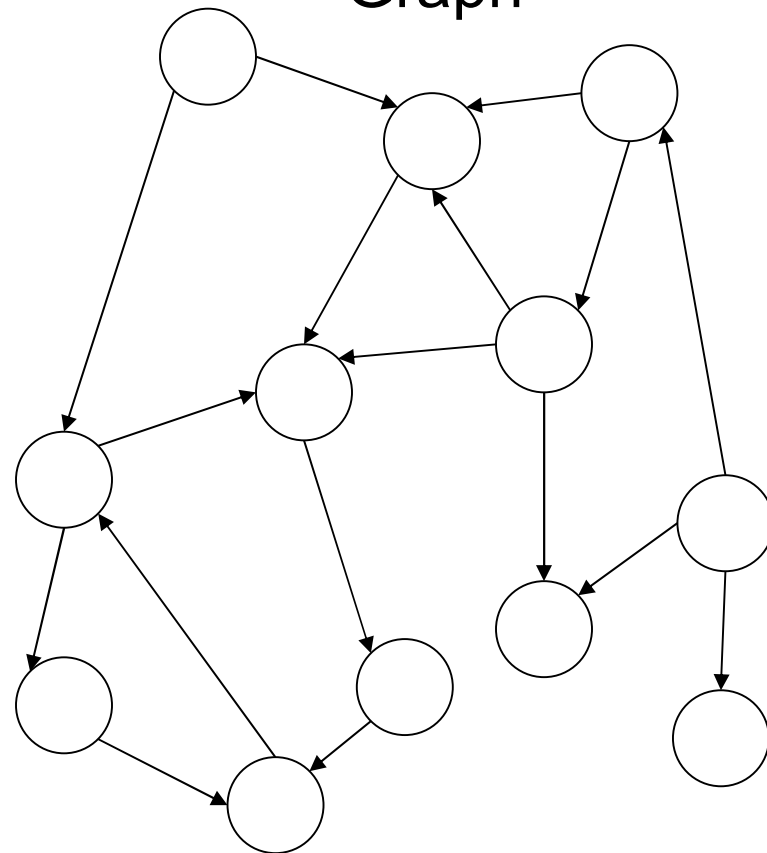
- Graph analytics has several unique properties
  - One large object: the graph
  - Difficult to partition and process in parallel
  - Iterative processing
    - Little work per vertex at each iteration
    - Many iterations & significant amount of communication
- Example applications
  - Shortest path
  - Clustering
  - Page rank and variants
  - Triangles and other structure
  - ...

# Example 1: Pattern Matching

Pattern

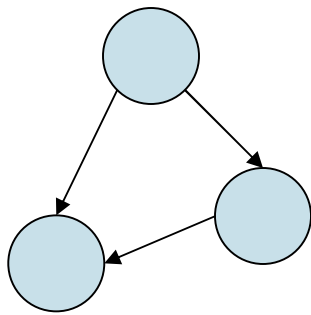


Graph

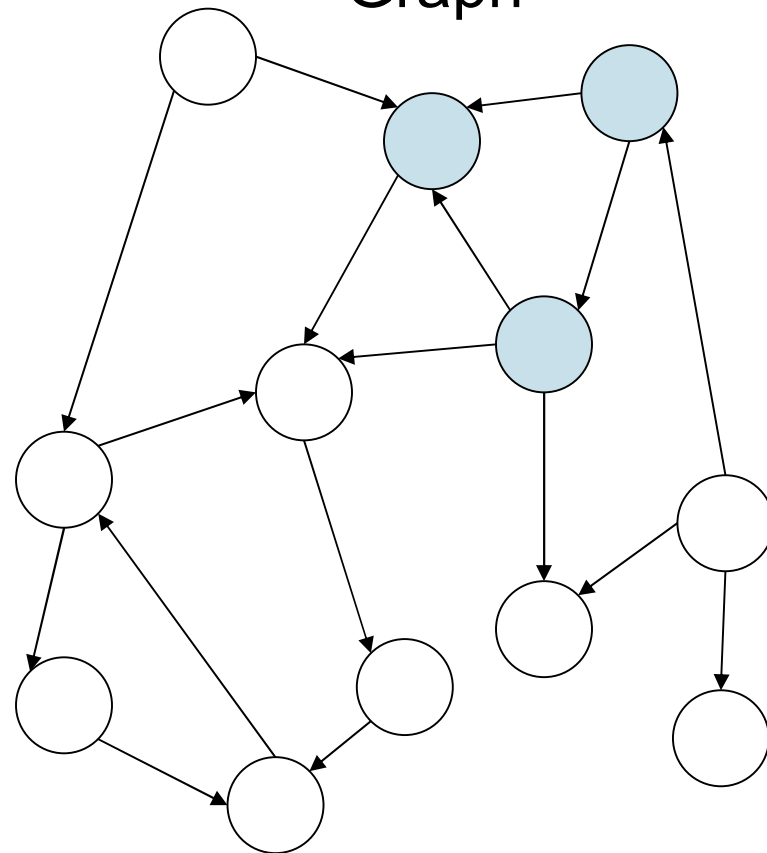


# Example 1: Pattern Matching

Pattern



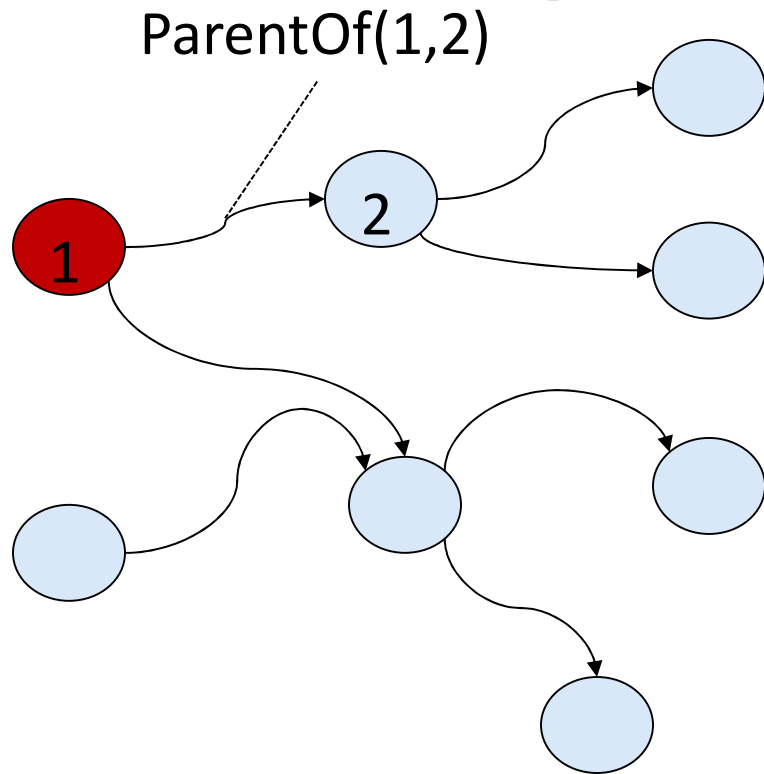
Graph







# Example 2: Descendants

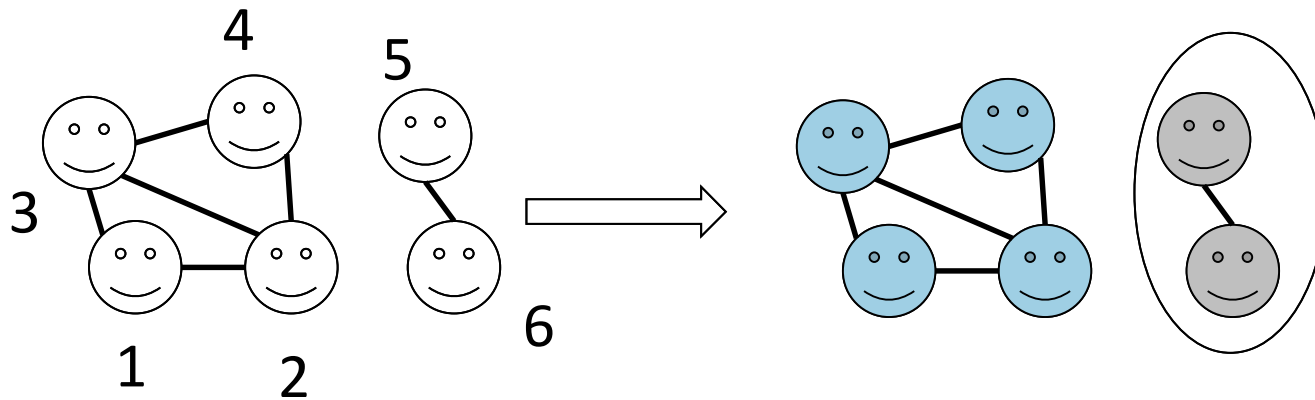


Find all descendants  
of the red node

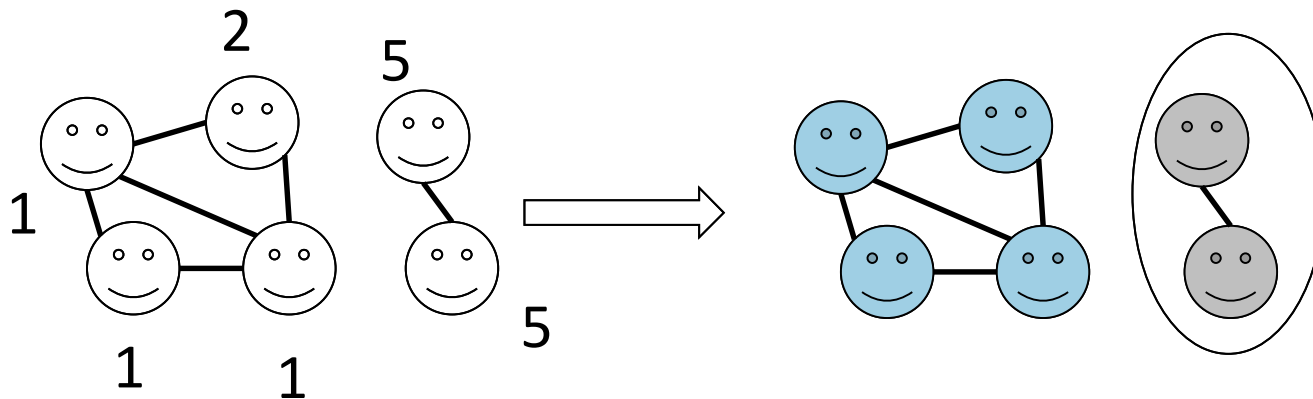
Recursively follow the ParentOf links  
until no new descendants are found



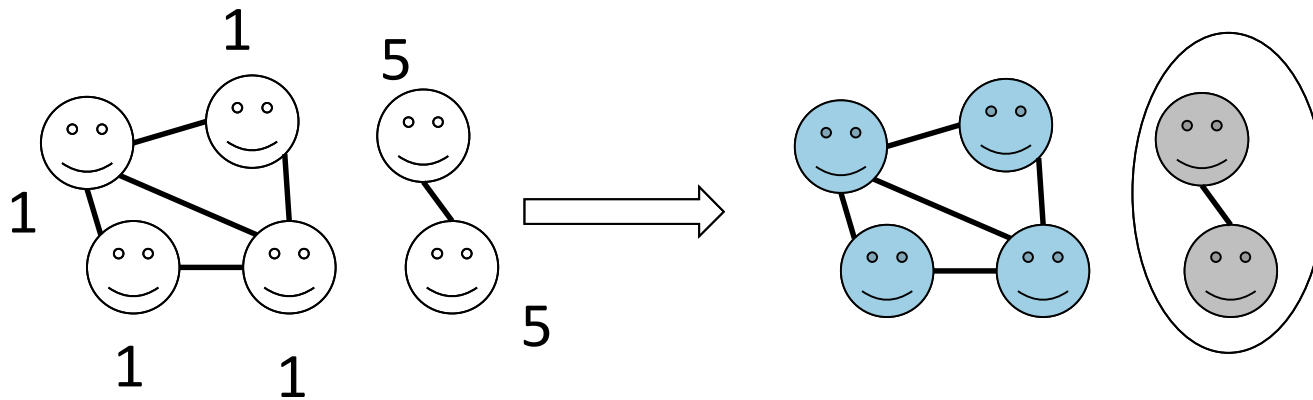
# Example 3: Connected Components



# Example 3: Connected Components

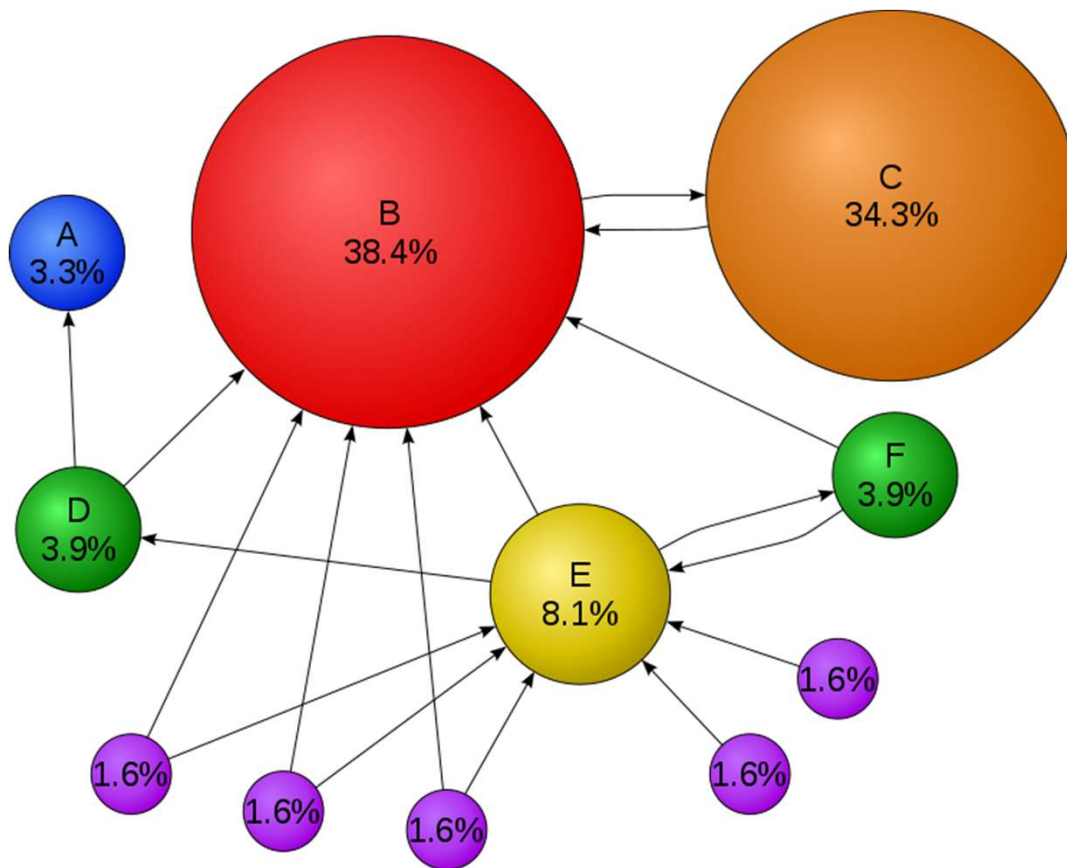


# Example 3: Connected Components



# Example 4: PageRank

The PageRank algorithm outputs a probability distribution used to represent the likelihood that a person randomly clicking on links will arrive at any particular page.



Iterate until convergence

$$PR(p_i, t+1) =$$

$$(1-d)/N +$$

$$d \sum_{p_j \in M(p_i)} PR(p_j, t) / L(p_j)$$

Where

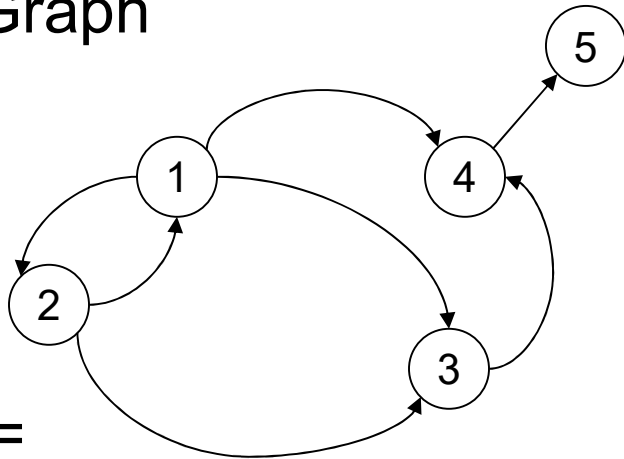
- $PR(x)$ : Page rank of  $x$
- $L(x)$ : # Outgoing links from  $x$
- $M(x)$ : Incoming pages to  $x$

# How to Model Graph Analytics

- Option 1: Relational Model
  - Relation Edges( $v_1, v_2$ )
  - Optionally can also have a relation Vertices( $v$ )
  - Relational queries
- Option 2: Graph Model
  - The graph is a first-class citizen
  - Vertex-based API
  - Pattern-based and/or traversal-based queries
- Option 3: Linear Algebra
  - Graph as a matrix

# Processing Graphs in SQL

Graph

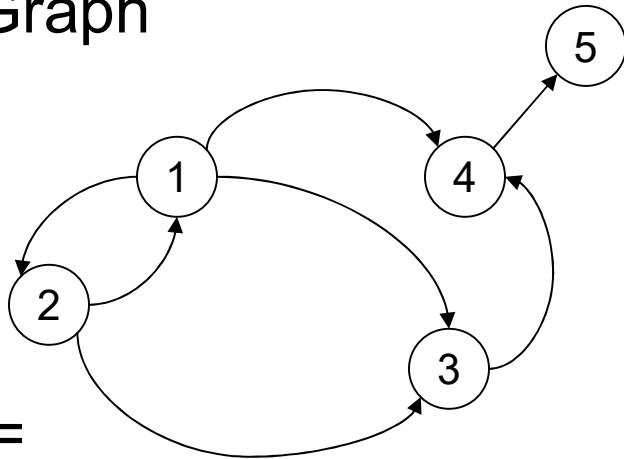


R=

src	dst
1	2
2	1
2	3
1	4
3	4
4	5
1	3

# Processing Graphs in SQL

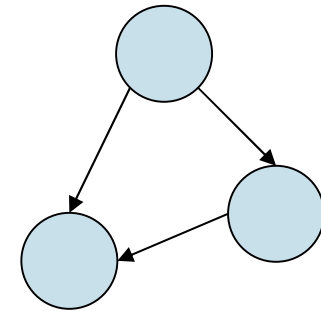
Graph



R=

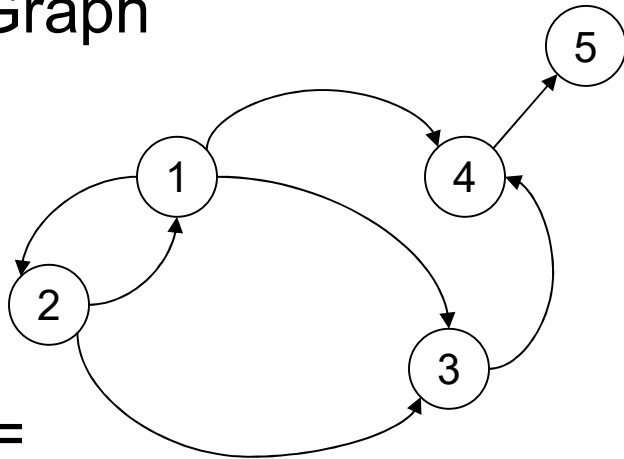
src	dst
1	2
2	1
2	3
1	4
3	4
4	5
1	3

Pattern Matching



# Processing Graphs in SQL

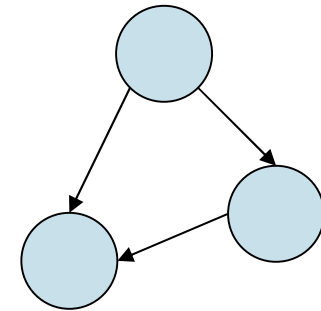
Graph



R=

src	dst
1	2
2	1
2	3
1	4
3	4
4	5
1	3

Pattern Matching

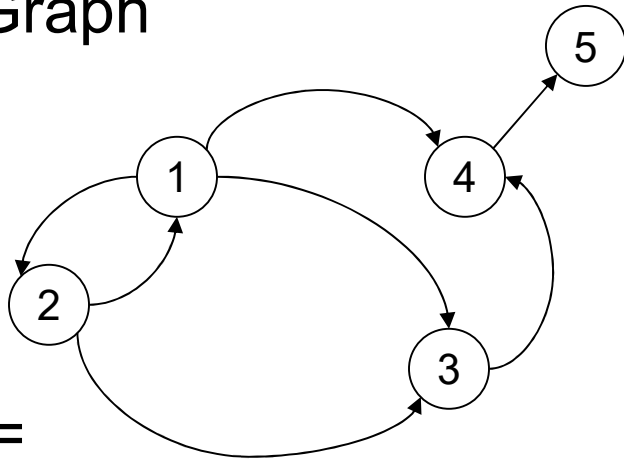


```
SELECT ...  
FROM ...  
WHERE ...
```



# Processing Graphs in SQL

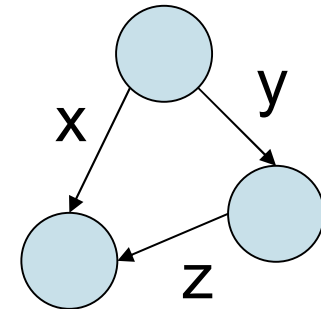
Graph



R=

src	dst
1	2
2	1
2	3
1	4
3	4
4	5
1	3

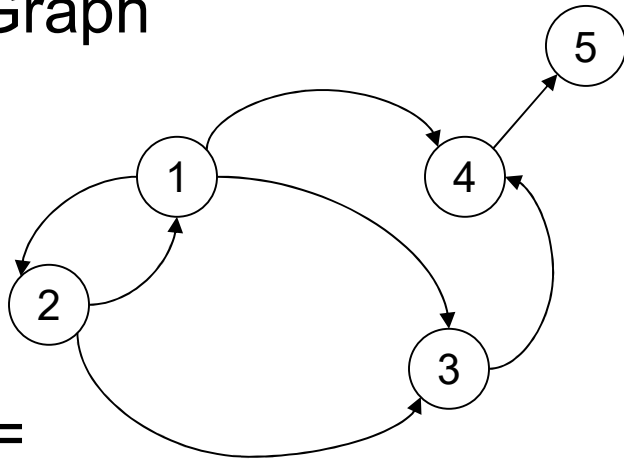
Pattern Matching



```
SELECT ...  
FROM ...  
WHERE ...
```

# Processing Graphs in SQL

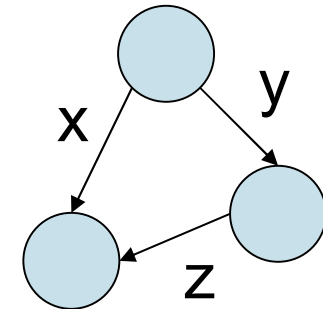
Graph



R=

src	dst
1	2
2	1
2	3
1	4
3	4
4	5
1	3

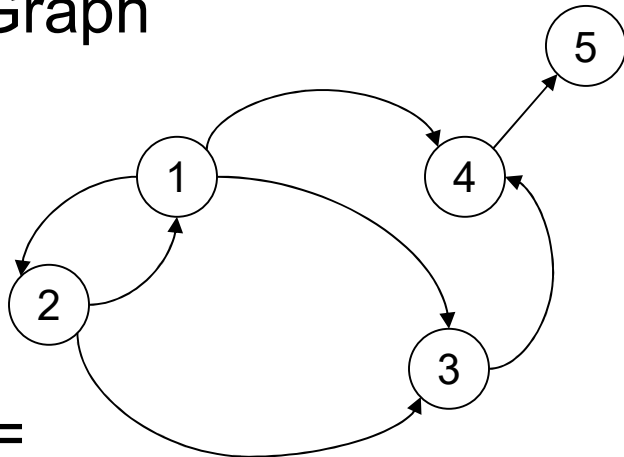
Pattern Matching



```
SELECT
FROM R x, R y, R z
WHERE
```

# Processing Graphs in SQL

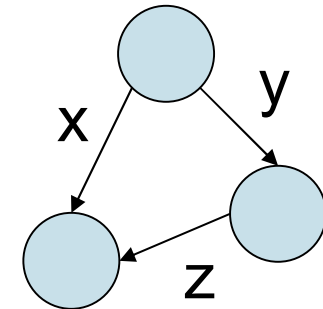
Graph



R=

src	dst
1	2
2	1
2	3
1	4
3	4
4	5
1	3

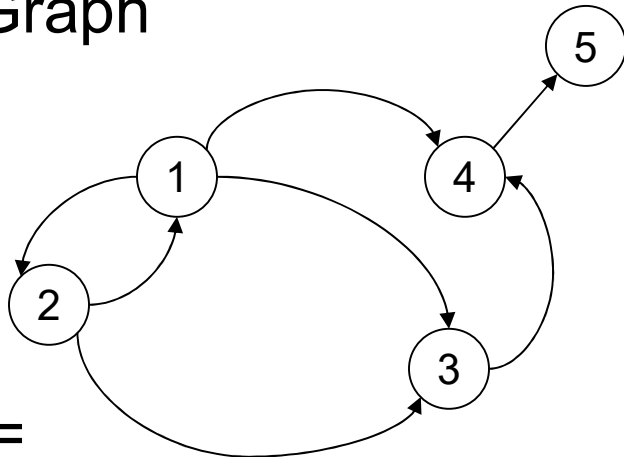
Pattern Matching



```
SELECT x.src, y.dst, z.dst
FROM R x, R y, R z
WHERE
```

# Processing Graphs in SQL

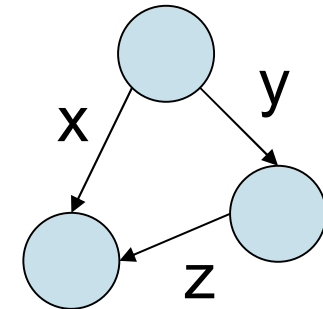
Graph



R=

src	dst
1	2
2	1
2	3
1	4
3	4
4	5
1	3

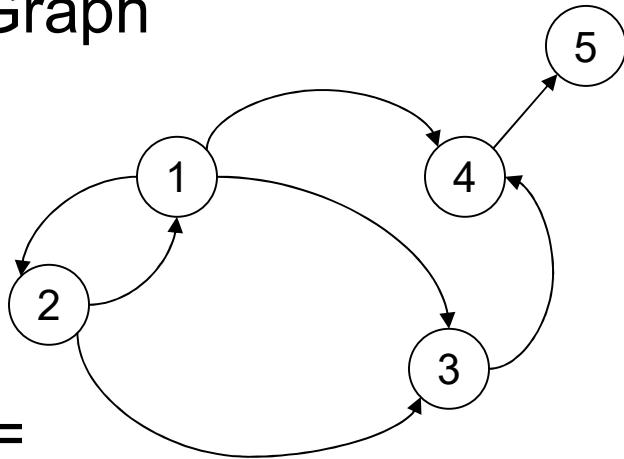
Pattern Matching



```
SELECT x.src, y.dst, z.dst
FROM R x, R y, R z
WHERE x.src = y.src
      and x.dst = z.dst
      and y.dst = z.src
```

# Processing Graphs in SQL

Graph

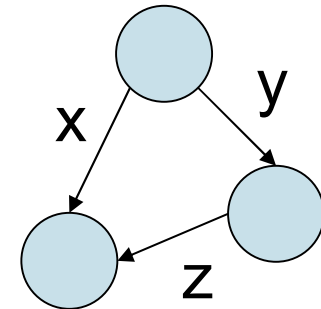


R=

src	dst
1	2
2	1
2	3
1	4
3	4
4	5
1	3

x.src	y.dst	z.dst
1	2	3
1	3	4
2	1	3

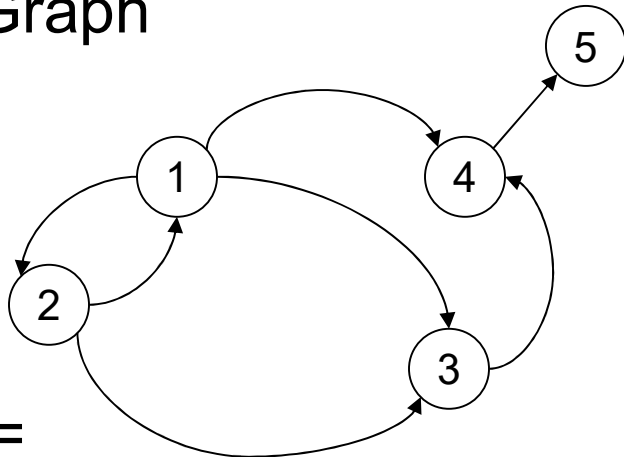
Pattern Matching



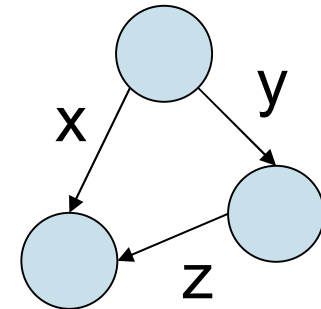
```
SELECT x.src, y.dst, z.dst
FROM R x, R y, R z
WHERE x.src = y.src
      and x.dst = z.dst
      and y.dst = z.src
```

# Processing Graphs in SQL

Graph



Pattern Matching



R=

src	dst
1	2
2	1
2	3
1	4
3	4
4	5
1	3

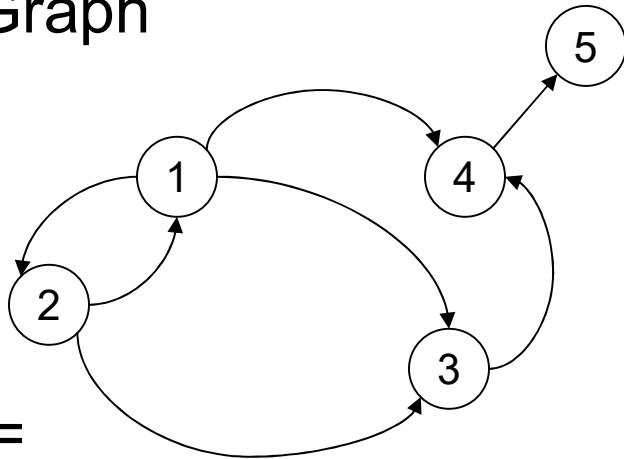
x.src	y.dst	z.dst
1	2	3
1	3	4
2	1	3

```
SELECT x.src, y.dst, z.dst
FROM R x, R y, R z
WHERE x.src = y.src
      and x.dst = z.dst
      and y.dst = z.src
```

A pattern with n edges  
becomes an n-way selfjoin

# Processing Graphs in SQL

Graph



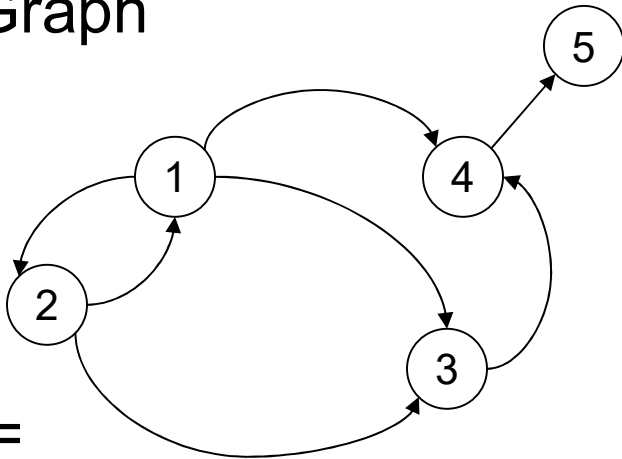
Find Descendants of node 2

R=

src	dst
1	2
2	1
2	3
1	4
3	4
4	5
1	3

# Processing Graphs in SQL

Graph



Find Descendants of node 2

Find children:

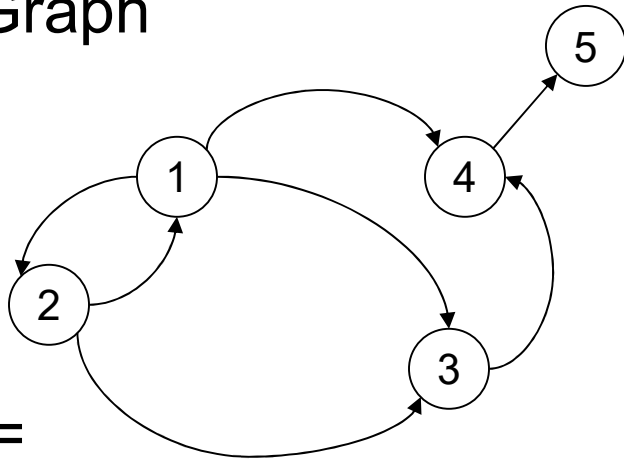
R=

src	dst
1	2
2	1
2	3
1	4
3	4
4	5
1	3



# Processing Graphs in SQL

Graph



Find Descendants of node 2

Find children:

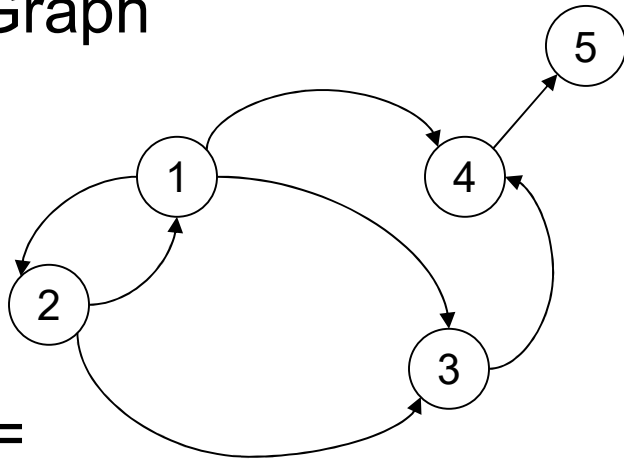
```
SELECT x.dst as d
FROM R x
WHERE x.src = 2
```

R=

src	dst
1	2
2	1
2	3
1	4
3	4
4	5
1	3

# Processing Graphs in SQL

Graph



Find Descendants of node 2

Find children:

```
SELECT x.dst as d
FROM R x
WHERE x.src = 2
```

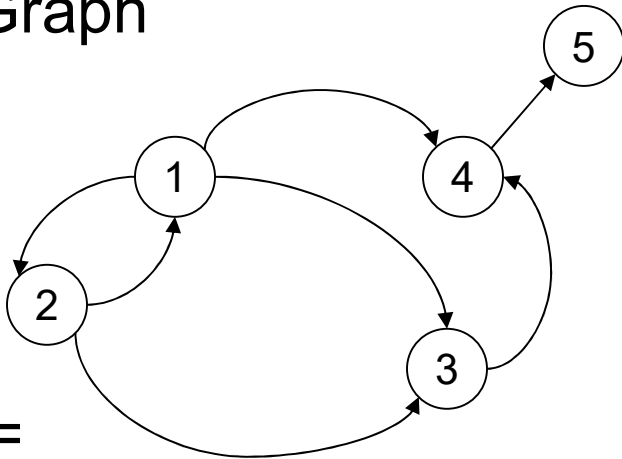
d
1
3

R=

src	dst
1	2
2	1
2	3
1	4
3	4
4	5
1	3

# Processing Graphs in SQL

Graph



R=

src	dst
1	2
2	1
2	3
1	4
3	4
4	5
1	3

d
1
3

2
4

Find Descendants of node 2

Find children:

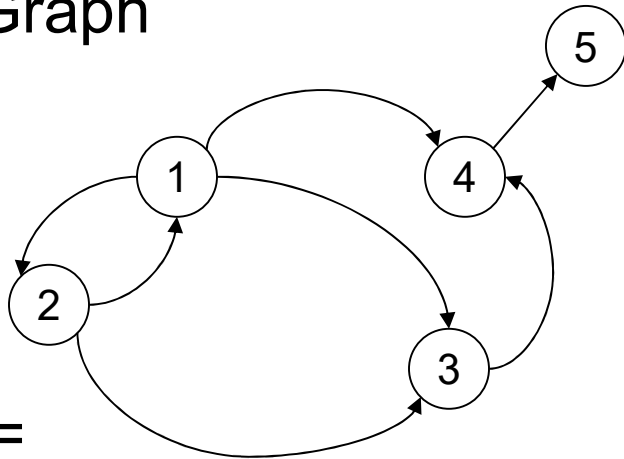
```
SELECT x.dst as d
FROM R x
WHERE x.src = 2
```

...and their children

```
UNION
SELECT DISTINCT y.dst as d
FROM R x, R y
WHERE x.src = 2 and x.dst = y.srt
```

# Processing Graphs in SQL

Graph



R=

src	dst
1	2
2	1
2	3
1	4
3	4
4	5
1	3

d
1
3

2
4

5
---

...

Find Descendants of node 2

Find children:

```
SELECT x.dst as d
FROM R x
WHERE x.src = 2
```

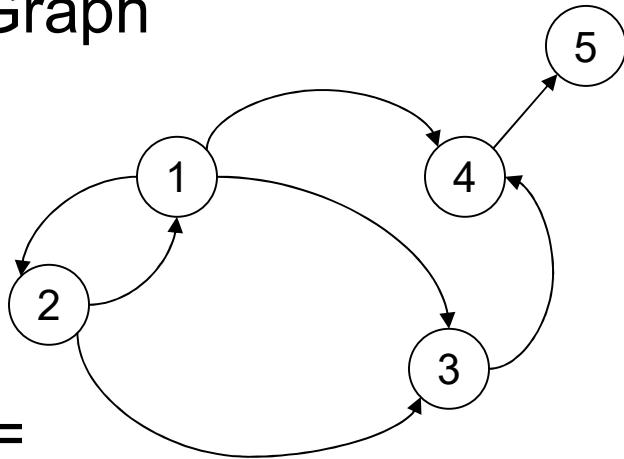
...and their children

```
UNION
SELECT DISTINCT y.dst as d
FROM R x, R y
WHERE x.src = 2 and x.dst = y.srt
```

...and their children...

# Processing Graphs in SQL

Graph



R=

src	dst
1	2
2	1
2	3
1	4
3	4
4	5
1	3

Cannot  
compute  
in SQL

d
1
3

2
4

5
---

...

Find Descendants of node 2

Find children:

```
SELECT x.dst as d
FROM R x
WHERE x.src = 2
```

...and their children

```
UNION
SELECT DISTINCT y.dst as d
FROM R x, R y
WHERE x.src = 2 and x.dst = y.src
```

...and their children...

# Discussion

- Graph processing often requires recursion:
  - Descendants, connected components, etc
- SQL does support recursion using WITH and CTE (Common Table Expression)
  - Lots of restrictions
- Origin of recursion in SQL: datalog

# Datalog

- Designed in the 80's: simple, concise, elegant, very popular in research
- All techniques for recursive relational queries were developed for datalog
- But: no standard, no reference implementation; in HW4 we use Souffle

# Outline

- Datalog rules

- Recursion

- Semantics

Next time: extensions, semi-naïve algo.



Actor(id, fname, lname)  
Casts(pid, mid)  
Movie(id, name, year)

← Schema

# Datalog: Facts and Rules

Actor(id, fname, lname)

Casts(pid, mid)

Movie(id, name, year)

# Datalog: Facts and Rules

**Facts** = tuples in the database

**Rules** = queries

Actor(id, fname, lname)

Casts(pid, mid)

Movie(id, name, year)

# Datalog: Facts and Rules

**Facts** = tuples in the database

**Rules** = queries

Actor(344759, 'Douglas', 'Fowley').

Casts(344759, 29851).

Casts(355713, 29000).

Movie(7909, 'A Night in Armour', 1910).

Movie(29000, 'Arizona', 1940).

Movie(29445, 'Ave Maria', 1940).

Actor(id, fname, lname)

Casts(pid, mid)

Movie(id, name, year)

# Datalog: Facts and Rules

**Facts** = tuples in the database

Actor(344759, 'Douglas', 'Fowley').  
Casts(344759, 29851).  
Casts(355713, 29000).  
Movie(7909, 'A Night in Armour', 1910).  
Movie(29000, 'Arizona', 1940).  
Movie(29445, 'Ave Maria', 1940).

**Rules** = queries

Q1(y) :- Movie(x,y,z), z='1940'.

Actor(id, fname, lname)

Casts(pid, mid)

Movie(id, name, year)

# Datalog: Facts and Rules

**Facts** = tuples in the database

Actor(344759, 'Douglas', 'Fowley').  
Casts(344759, 29851).  
Casts(355713, 29000).  
Movie(7909, 'A Night in Armour', 1910).  
Movie(29000, 'Arizona', 1940).  
Movie(29445, 'Ave Maria', 1940).

**Rules** = queries

Q1(y) :- Movie(x,y,z), z='1940'.

Find Movies made in 1940

Actor(id, fname, lname)

Casts(pid, mid)

Movie(id, name, year)

# Datalog: Facts and Rules

**Facts** = tuples in the database

Actor(344759, 'Douglas', 'Fowley').

Casts(344759, 29851).

Casts(355713, 29000).

Movie(7909, 'A Night in Armour', 1910).

Movie(29000, 'Arizona', 1940).

Movie(29445, 'Ave Maria', 1940).

**Rules** = queries

Q1(y) :- Movie(x,y,z), z='1940'.

Q2(f, l) :- Actor(z,f,l), Casts(z,x),  
Movie(x,y,'1940').

Actor(id, fname, lname)

Casts(pid, mid)

Movie(id, name, year)

# Datalog: Facts and Rules

**Facts** = tuples in the database

Actor(344759, 'Douglas', 'Fowley').  
Casts(344759, 29851).  
Casts(355713, 29000).  
Movie(7909, 'A Night in Armour', 1910).  
Movie(29000, 'Arizona', 1940).  
Movie(29445, 'Ave Maria', 1940).

**Rules** = queries

Q1(y) :- Movie(x,y,z), z='1940'.

Q2(f, l) :- Actor(z,f,l), Casts(z,x),  
Movie(x,y,'1940').

Find Actors who acted in Movies made in 1940

Actor(id, fname, lname)

Casts(pid, mid)

Movie(id, name, year)

# Datalog: Facts and Rules

**Facts** = tuples in the database

Actor(344759, 'Douglas', 'Fowley').

Casts(344759, 29851).

Casts(355713, 29000).

Movie(7909, 'A Night in Armour', 1910).

Movie(29000, 'Arizona', 1940).

Movie(29445, 'Ave Maria', 1940).

**Rules** = queries

Q1(y) :- Movie(x,y,z), z='1940'.

Q2(f, l) :- Actor(z,f,l), Casts(z,x),  
Movie(x,y,'1940').

Q3(f,l) :- Actor(z,f,l), Casts(z,x1), Movie(x1,y1,1910),  
Casts(z,x2), Movie(x2,y2,1940)



Actor(id, fname, lname)

Casts(pid, mid)

Movie(id, name, year)

# Datalog: Facts and Rules

**Facts** = tuples in the database

Actor(344759, 'Douglas', 'Fowley').  
Casts(344759, 29851).  
Casts(355713, 29000).  
Movie(7909, 'A Night in Armour', 1910).  
Movie(29000, 'Arizona', 1940).  
Movie(29445, 'Ave Maria', 1940).

**Rules** = queries

Q1(y) :- Movie(x,y,z), z='1940'.

Q2(f, l) :- Actor(z,f,l), Casts(z,x),  
Movie(x,y,'1940').

Q3(f,l) :- Actor(z,f,l), Casts(z,x1), Movie(x1,y1,1910),  
Casts(z,x2), Movie(x2,y2,1940)

Find Actors who acted in a Movie in 1940 and in one in 1910

Actor(id, fname, lname)

Casts(pid, mid)

Movie(id, name, year)

# Datalog: Facts and Rules

**Facts** = tuples in the database

Actor(344759, 'Douglas', 'Fowley').  
Casts(344759, 29851).  
Casts(355713, 29000).  
Movie(7909, 'A Night in Armour', 1910).  
Movie(29000, 'Arizona', 1940).  
Movie(29445, 'Ave Maria', 1940).

**Rules** = queries

Q1(y) :- Movie(x,y,z), z='1940'.

Q2(f, l) :- Actor(z,f,l), Casts(z,x),  
Movie(x,y,'1940').

Q3(f,l) :- Actor(z,f,l), Casts(z,x1), Movie(x1,y1,1910),  
Casts(z,x2), Movie(x2,y2,1940)

**Extensional Database Predicates = EDB** = Actor, Casts, Movie

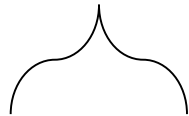
**Intensional Database Predicates = IDB** = Q1, Q2, Q3

# Anatomy of a Rule

Q2(f, l) :- Actor(z,f,l), Casts(z,x), Movie(x,y,'1940').

# Anatomy of a Rule

head

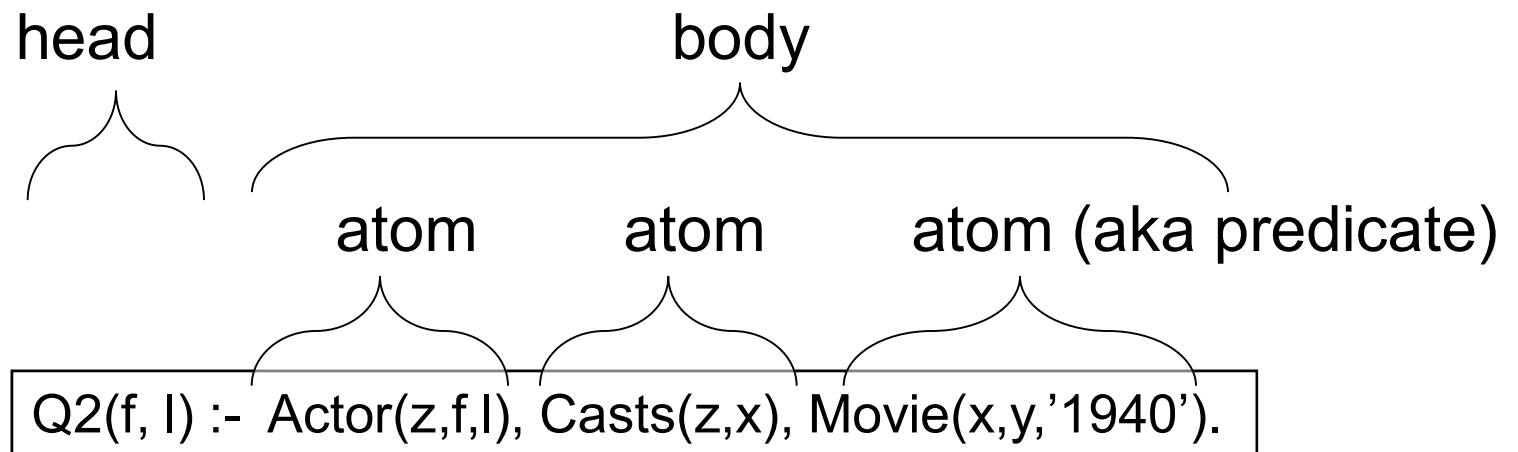


body

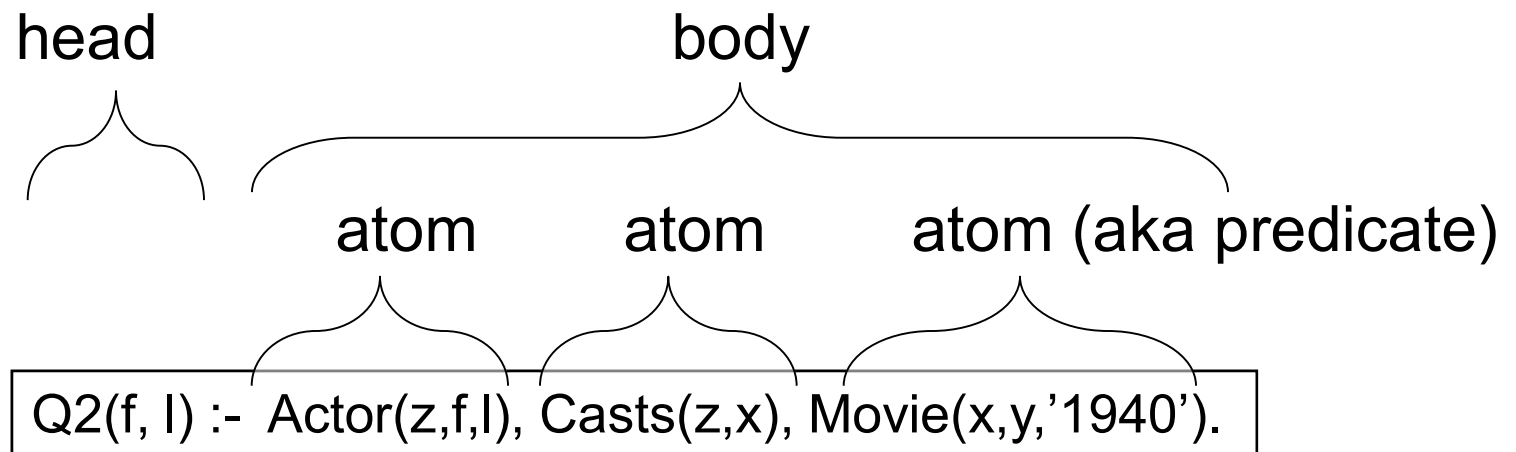


Q2(f, l) :- Actor(z,f,l), Casts(z,x), Movie(x,y,'1940').

# Anatomy of a Rule



# Anatomy of a Rule



f, l = head variables

x, y, z = existential variables

# Outline

- Datalog rules

- Recursion

- Semantics

Next time: extensions, semi-naïve algo.

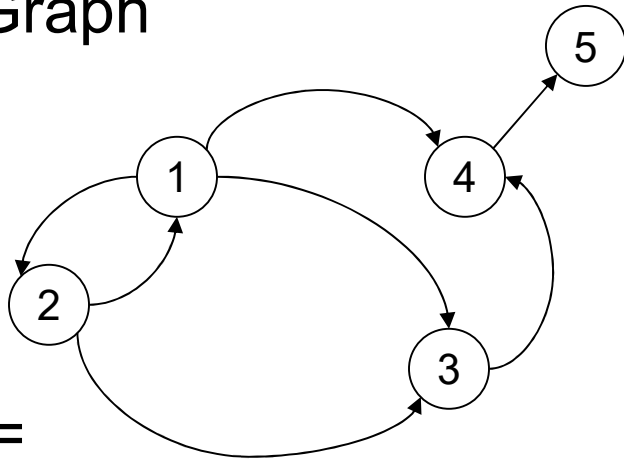
# Datalog program

- A datalog program = several rules
- Rules may be recursive
- Set semantics only



# Processing Graphs in Datalog

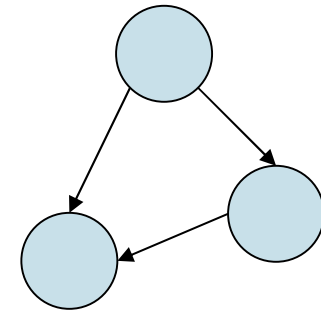
Graph



R=

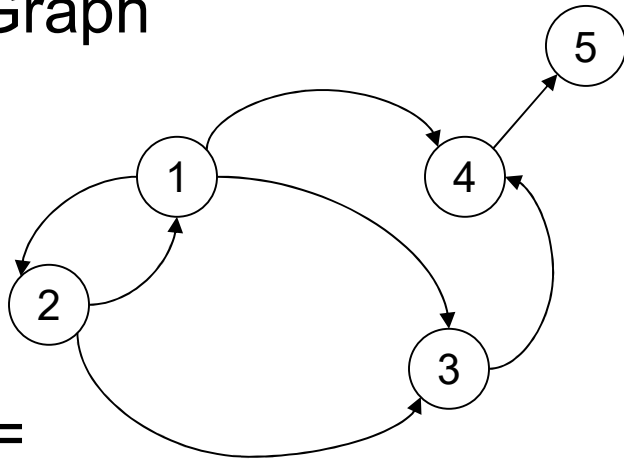
src	dst
1	2
2	1
2	3
1	4
3	4
4	5
1	3

Pattern Matching



# Processing Graphs in Datalog

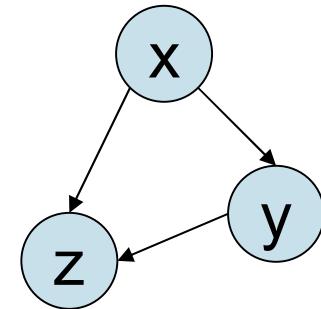
Graph



R=

src	dst
1	2
2	1
2	3
1	4
3	4
4	5
1	3

Pattern Matching

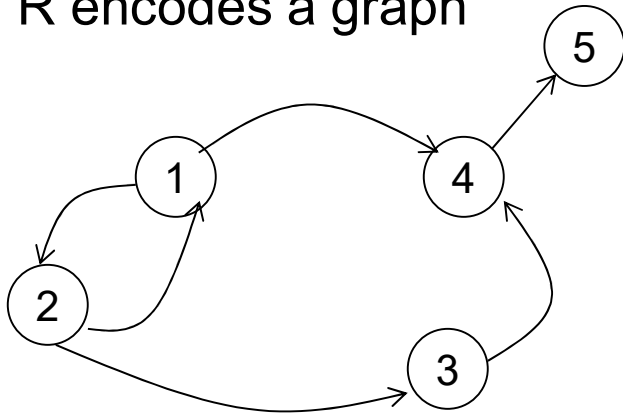


Answer(x,y,z) :- R(x,y), R(x,z), R(y,z)

# Example

Descendants of node 2

R encodes a graph

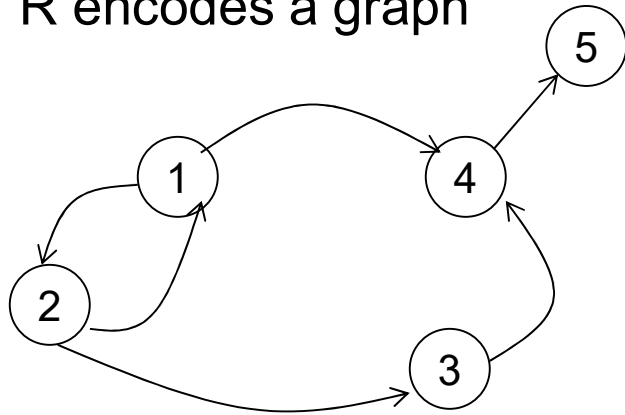


R=

1	2
2	1
2	3
1	4
3	4
4	5

# Example

R encodes a graph



R=

1	2
2	1
2	3
1	4
3	4
4	5

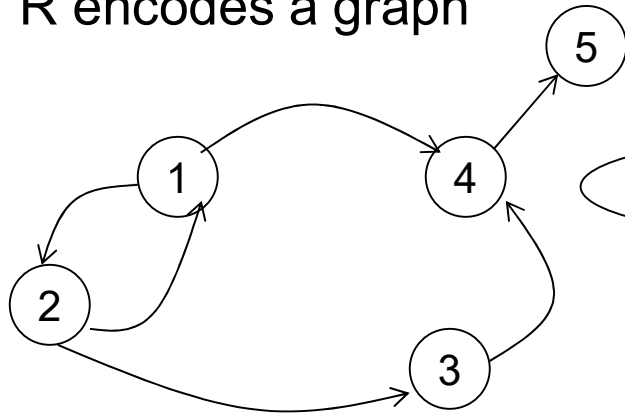
Descendants of node 2

$D(x) \text{ :- } R(2, x)$

$D(y) \text{ :- } D(x), R(x, y)$

# Example

R encodes a graph



Descendants of node 2

Recursive rule

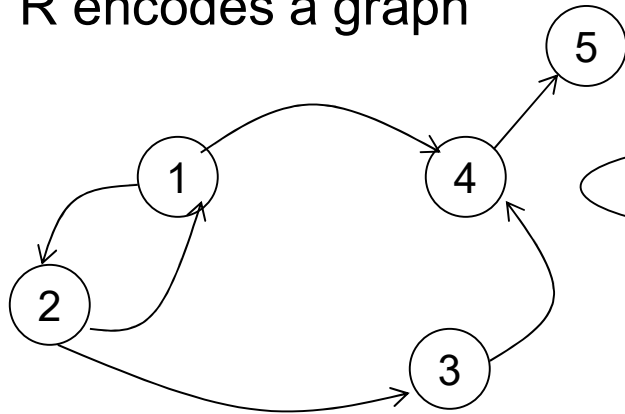
$$D(x) \text{ :- } R(2, x)$$
$$D(y) \text{ :- } D(x), R(x, y)$$

R=

1	2
2	1
2	3
1	4
3	4
4	5

# Example

R encodes a graph



Descendants of node 2

Recursive rule

$$D(x) \text{ :- } R(2, x)$$
$$D(y) \text{ :- } D(x), R(x, y)$$

R=

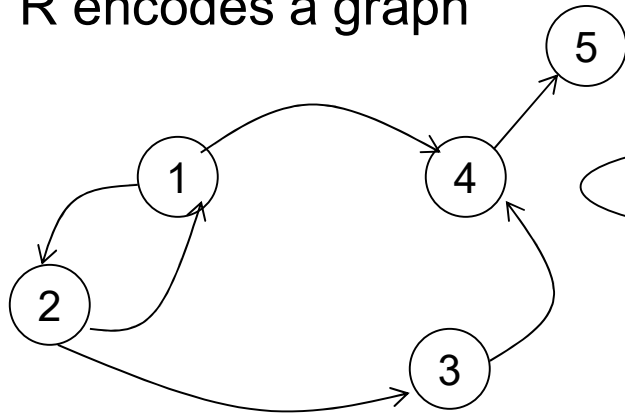
1	2
2	1
2	3
1	4
3	4
4	5

How recursion works in datalog:

Initially D = empty

# Example

R encodes a graph



Descendants of node 2

Recursive rule

$D(x) \text{ :- } R(2, x)$   
 $D(y) \text{ :- } D(x), R(x, y)$

R=

1	2
2	1
2	3
1	4
3	4
4	5

How recursion works in datalog:

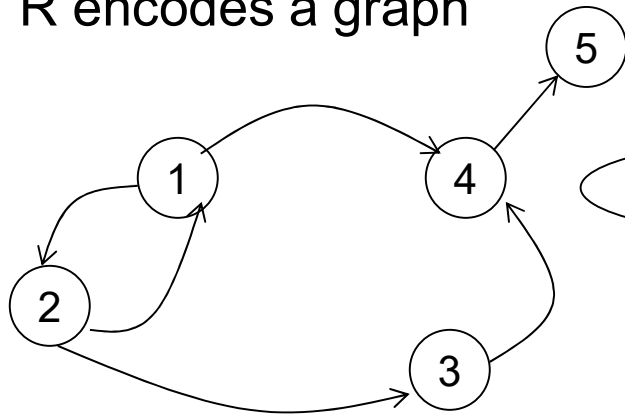
Initially D = empty

- Compute both rules:

$D(x) \text{ :- } R(2, x)$   
 $D(y) \text{ :- } D(x), R(x, y)$

# Example

R encodes a graph



R=

1	2
2	1
2	3
1	4
3	4
4	5

How recursion works in datalog:

Initially D = empty

- Compute both rules:  
...now D = {1,3}

Descendants of node 2

Recursive rule

$D(x) \text{ :- } R(2, x)$

$D(y) \text{ :- } D(x), R(x, y)$

{1,3}

$D(x) \text{ :- } R(2, x)$

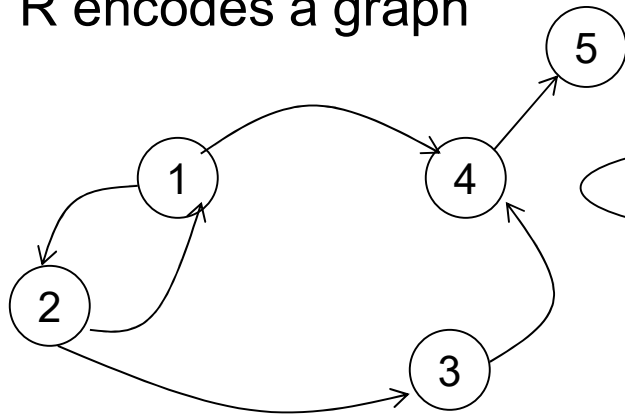
{}

$D(y) \text{ :- } D(x), R(x, y)$



# Example

R encodes a graph



R=

1	2
2	1
2	3
1	4
3	4
4	5

Descendants of node 2

Recursive rule

$$D(x) \text{ :- } R(2, x)$$

$$D(y) \text{ :- } D(x), R(x, y)$$

How recursion works in datalog:

Initially D = empty

- Compute both rules:  
...now D = {1,3}
- Compute both rules:

{1,3}

$$D(x) \text{ :- } R(2, x)$$

{}

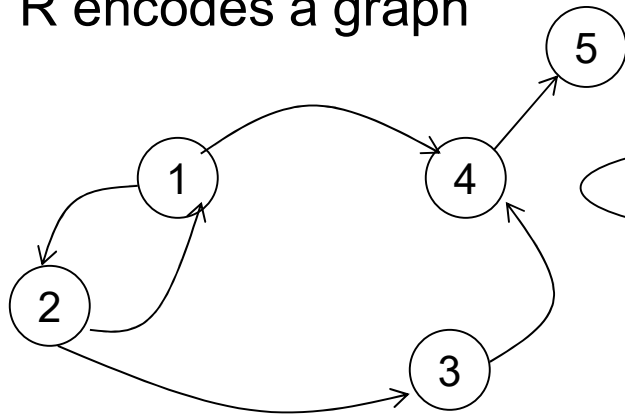
$$D(y) \text{ :- } D(x), R(x, y)$$

$$D(x) \text{ :- } R(2, x)$$

$$D(y) \text{ :- } D(x), R(x, y)$$

# Example

R encodes a graph



R=

1	2
2	1
2	3
1	4
3	4
4	5

How recursion works in datalog:

Initially D = empty

- Compute both rules:  
...now D = {1,3}
- Compute both rules:  
...now D = {1,3,2,4}

Descendants of node 2

Recursive rule

$D(x) :- R(2, x)$   
 $D(y) :- D(x), R(x, y)$

{1,3}

$D(x) :- R(2, x)$

{}

$D(y) :- D(x), R(x, y)$

{1,3}

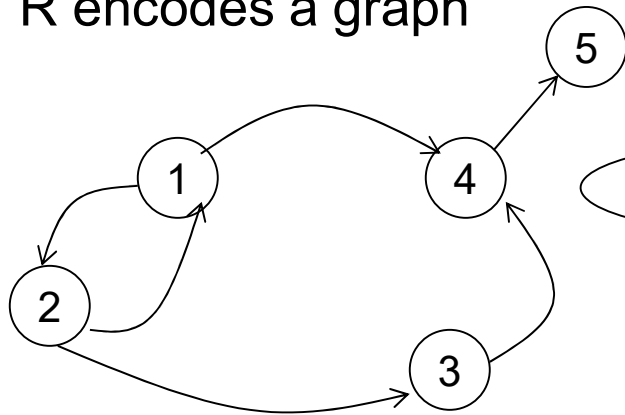
$D(x) :- R(2, x)$

{2,4}

$D(y) :- D(x), R(x, y)$

# Example

R encodes a graph



R=

1	2
2	1
2	3
1	4
3	4
4	5

Descendants of node 2

Recursive rule

$$D(x) \text{ :- } R(2, x)$$

$$D(y) \text{ :- } D(x), R(x, y)$$

How recursion works in datalog:

Initially D = empty

- Compute both rules:  
...now D = {1,3}
- Compute both rules:  
...now D = {1,3,2,4}
- Compute both rules:

{1,3}

$$D(x) \text{ :- } R(2, x)$$

{}

$$D(y) \text{ :- } D(x), R(x, y)$$

{1,3}

$$D(x) \text{ :- } R(2, x)$$

{2,4}

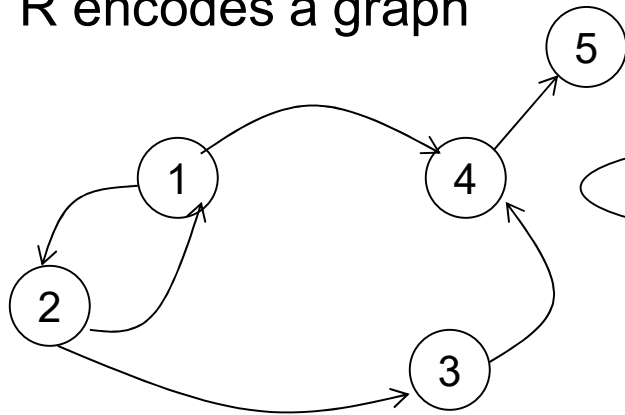
$$D(y) \text{ :- } D(x), R(x, y)$$

$$D(x) \text{ :- } R(2, x)$$

$$D(y) \text{ :- } D(x), R(x, y)$$

# Example

R encodes a graph



Recursive rule

Descendants of node 2

$D(x) :- R(2, x)$   
 $D(y) :- D(x), R(x,y)$

R=

1	2
2	1
2	3
1	4
3	4
4	5

How recursion works in datalog:

Initially D = empty

- Compute both rules:  
...now D = {1,3}
- Compute both rules:  
...now D = {1,3,2,4}
- Compute both rules:  
...now D = {1,3,2,4,5}

{1,3}

$D(x) :- R(2, x)$

{}

$D(y) :- D(x), R(x,y)$

{1,3}

$D(x) :- R(2, x)$

{2,4}

$D(y) :- D(x), R(x,y)$

{1,3}

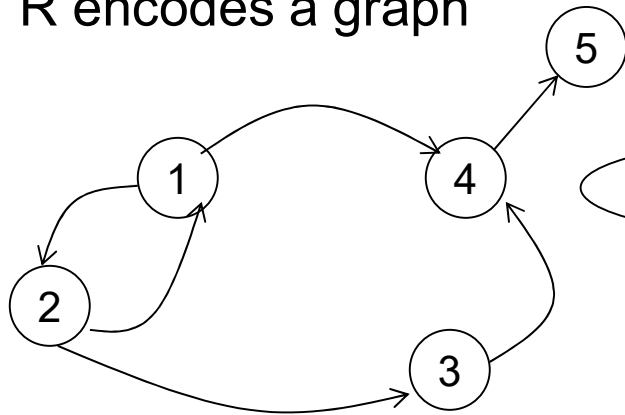
$D(x) :- R(2, x)$

{2,4,1,3,5}

$D(y) :- D(x), R(x,y)$

# Example

R encodes a graph



Recursive rule

Descendants of node 2

$D(x) \text{ :- } R(2, x)$   
 $D(y) \text{ :- } D(x), R(x,y)$

R=

1	2
2	1
2	3
1	4
3	4
4	5

How recursion works in datalog:

Initially D = empty

- Compute both rules:  
...now D = {1,3}
- Compute both rules:  
...now D = {1,3,2,4}
- Compute both rules:  
...now D = {1,3,2,4,5}
- Compute both rules:  
...nothing new. STOP

{1,3}

$D(x) \text{ :- } R(2, x)$

{}

$D(y) \text{ :- } D(x), R(x,y)$

{1,3}

$D(x) \text{ :- } R(2, x)$

{2,4}

$D(y) \text{ :- } D(x), R(x,y)$

{1,3}

$D(x) \text{ :- } R(2, x)$

{2,4,1,3,5}

$D(y) \text{ :- } D(x), R(x,y)$

# Outline

- Datalog rules
- Recursion
- Semantics

Next time: extensions, semi-naïve algo.

# Naïve Evaluation Algorithm

- Every rule  $\rightarrow$  SPJ\* query

\*SPJ = select-project-join

+USPJ = union-select-project-join

# Naïve Evaluation Algorithm

- Every rule  $\rightarrow$  SPJ\* query

$T(x,z) :- R(x,y), T(y,z), C(y,'green')$

\*SPJ = select-project-join

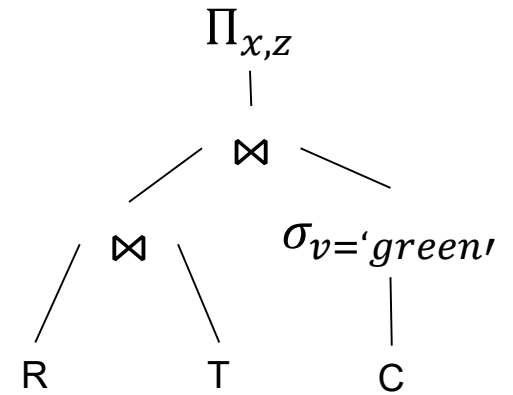
+USPJ = union-select-project-join



# Naïve Evaluation Algorithm

- Every rule  $\rightarrow$  SPJ\* query

$T(x,z) :- R(x,y), T(y,z), C(y,'green')$



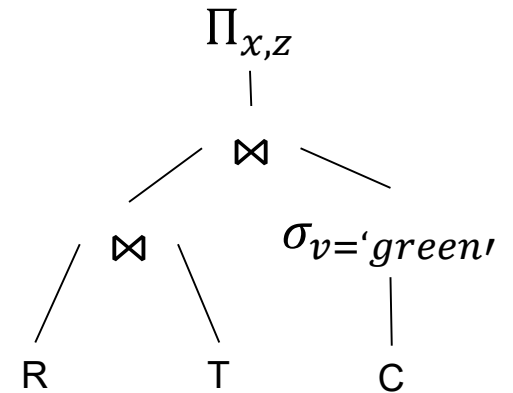
\*SPJ = select-project-join

+USPJ = union-select-project-join

# Naïve Evaluation Algorithm

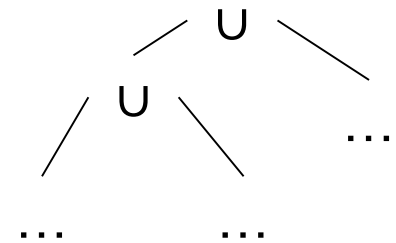
- Every rule  $\rightarrow$  SPJ\* query

$T(x,z) :- R(x,y), T(y,z), C(y,'green')$



- Multiple rules same head  $\rightarrow$  USPJ+

$T(x,y) :- \dots$   
 $T(x,y) :- \dots$   
 $\dots$



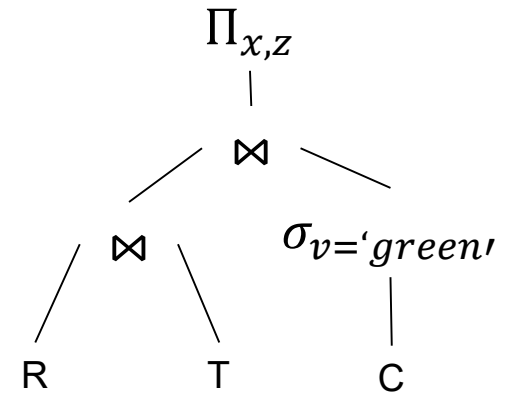
\*SPJ = select-project-join

+USPJ = union-select-project-join

# Naïve Evaluation Algorithm

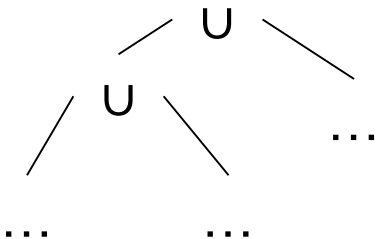
- Every rule  $\rightarrow$  SPJ\* query

$T(x,z) :- R(x,y), T(y,z), C(y,'green')$



- Multiple rules same head  $\rightarrow$  USPJ+

$T(x,y) :- \dots$   
 $T(x,y) :- \dots$   
 $\dots$



- Naïve Algorithm:

$IDBs := \emptyset$   
**repeat**  $IDBs := USPJs$   
**until** no more change

\*SPJ = select-project-join  
 +USPJ = union-select-project-join

# Naïve Evaluation Algorithm

$$D(x) :- R(2,x)$$
$$D(y) :- D(x),R(x,y)$$

# Naïve Evaluation Algorithm

$D(x) :- R(2,x)$

$D(y) :- D(x),R(x,y)$

$\Pi_{R.dst}(\sigma_{R.src=2}(R))$

# Naïve Evaluation Algorithm

$D(x) :- R(2,x)$

$D(y) :- D(x),R(x,y)$

$\Pi_{R.dst}(\sigma_{R.src=2}(R)) \cup \Pi_{R.dst}(D \bowtie_{D.node=R.src} R);$

# Naïve Evaluation Algorithm

$D(x) :- R(2,x)$

$D(y) :- D(x),R(x,y)$

$\Pi_{R.dst}(\sigma_{R.src=2}(R)) \cup \Pi_{R.dst}(D \bowtie_{D.node=R.src} R);$

# Naïve Evaluation Algorithm

$$D(x) :- R(2,x)$$
$$D(y) :- D(x),R(x,y)$$
$$D := \emptyset;$$

**repeat**

$$D := \Pi_{R.dst}(\sigma_{R.src=2}(R)) \cup \Pi_{R.dst}(D \bowtie_{D.node=R.src} R);$$

**until** [no more change]



# Naïve Evaluation Algorithm

The Naïve Evaluation Algorithm:

- Always terminates
- Always terminates in a number of steps that is polynomial in the size of the database