DATA516/CSED516 Scalable Data Systems and Algorithms

Lecture 6

Parallel Execution Wrapup
Graph Processing

Announcements

- Feedback for project proposals
- Redshift + Snowflake review was due
- AWS Academy class
- HW3 due on Monday

Today

- Skew
- Parallel Query Processing Wrap-up
- Graphs, datalog

Skew

Skew

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

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 $v_1, \ldots, v_N$

- 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 $v_1, \ldots, v_N$
  s.t. all are mapped to the same value:
  $h(v_1)=h(v_2)= \ldots h(v_N)$

Discussion

There is no deterministic "good" hash function:

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

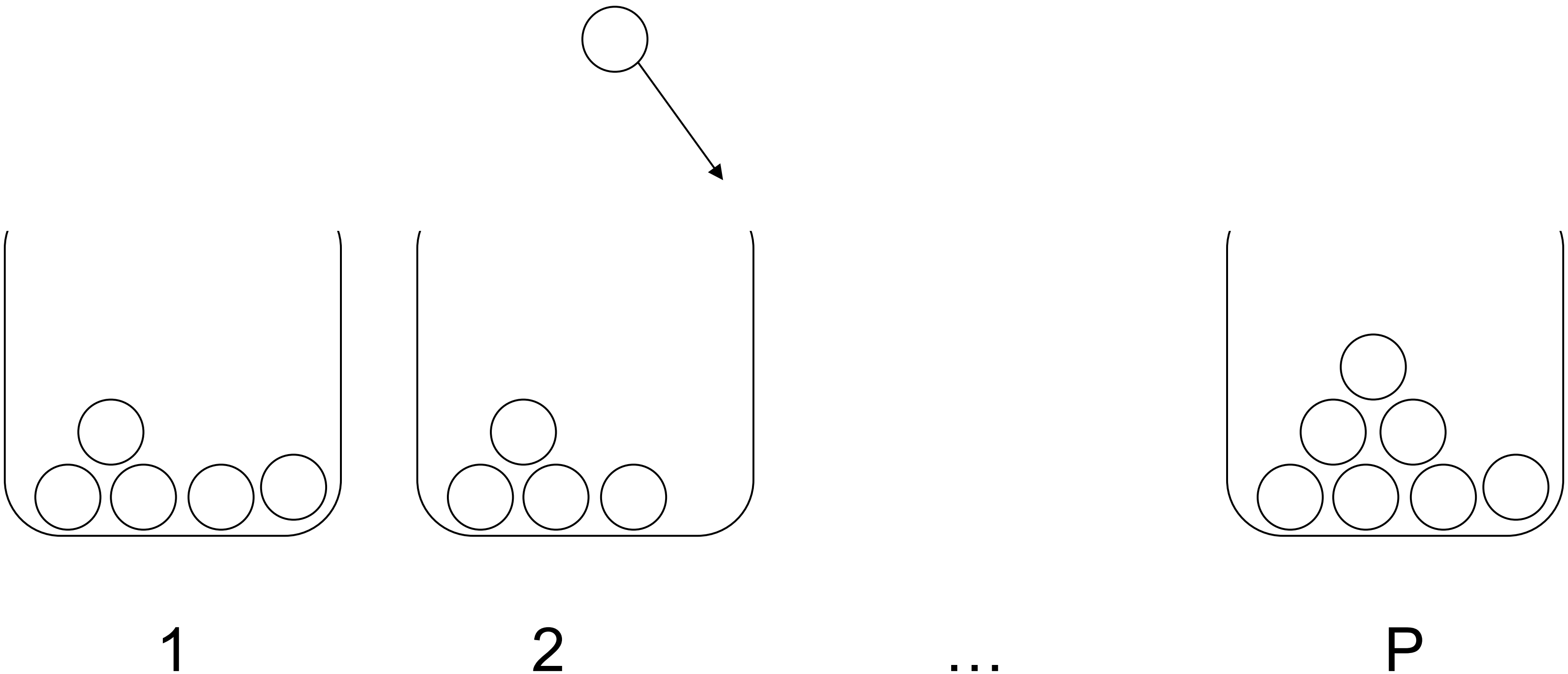
A "good" hash function is random function:

- 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:

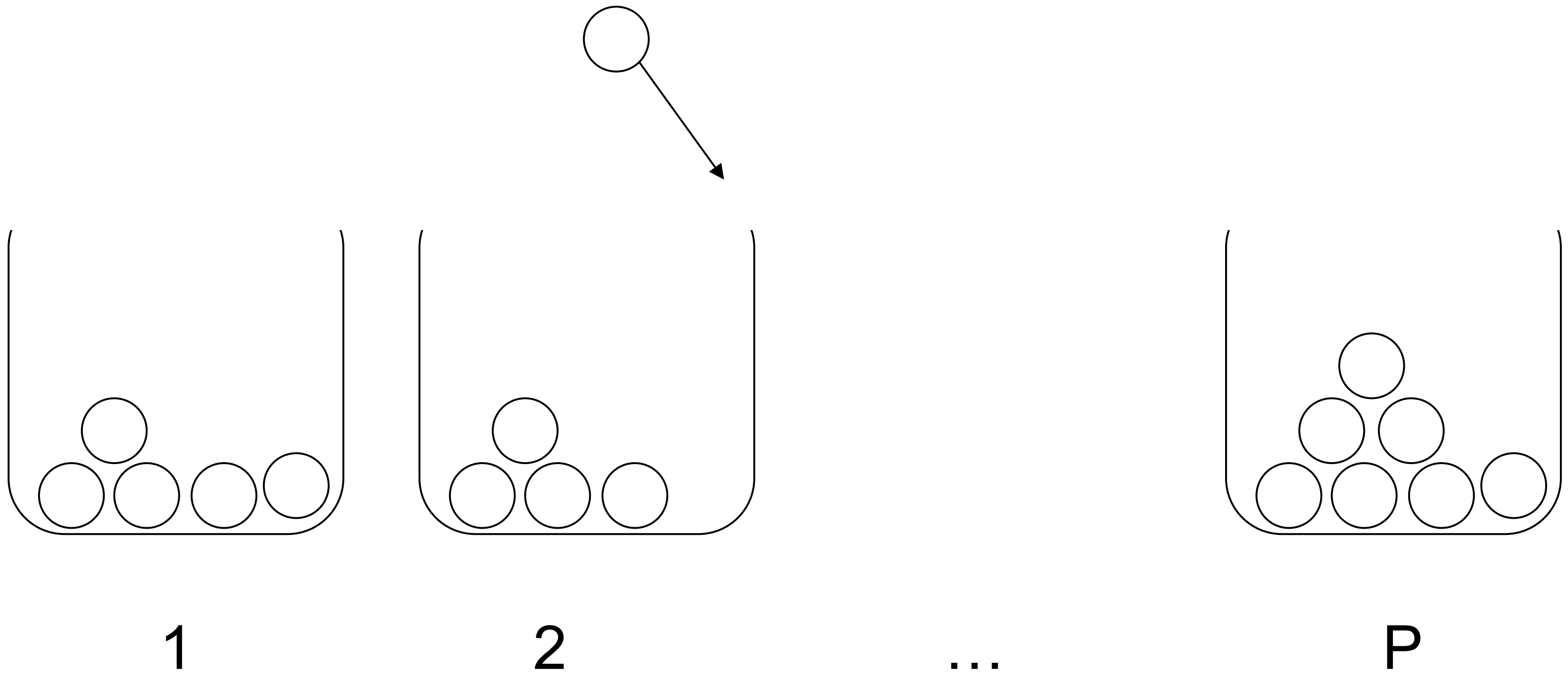
We throw N balls randomly into P bins:



Balls into Bins

Probability of a ball getting into bin #5: $\frac{1}{P}$

We throw N balls randomly into P bins:

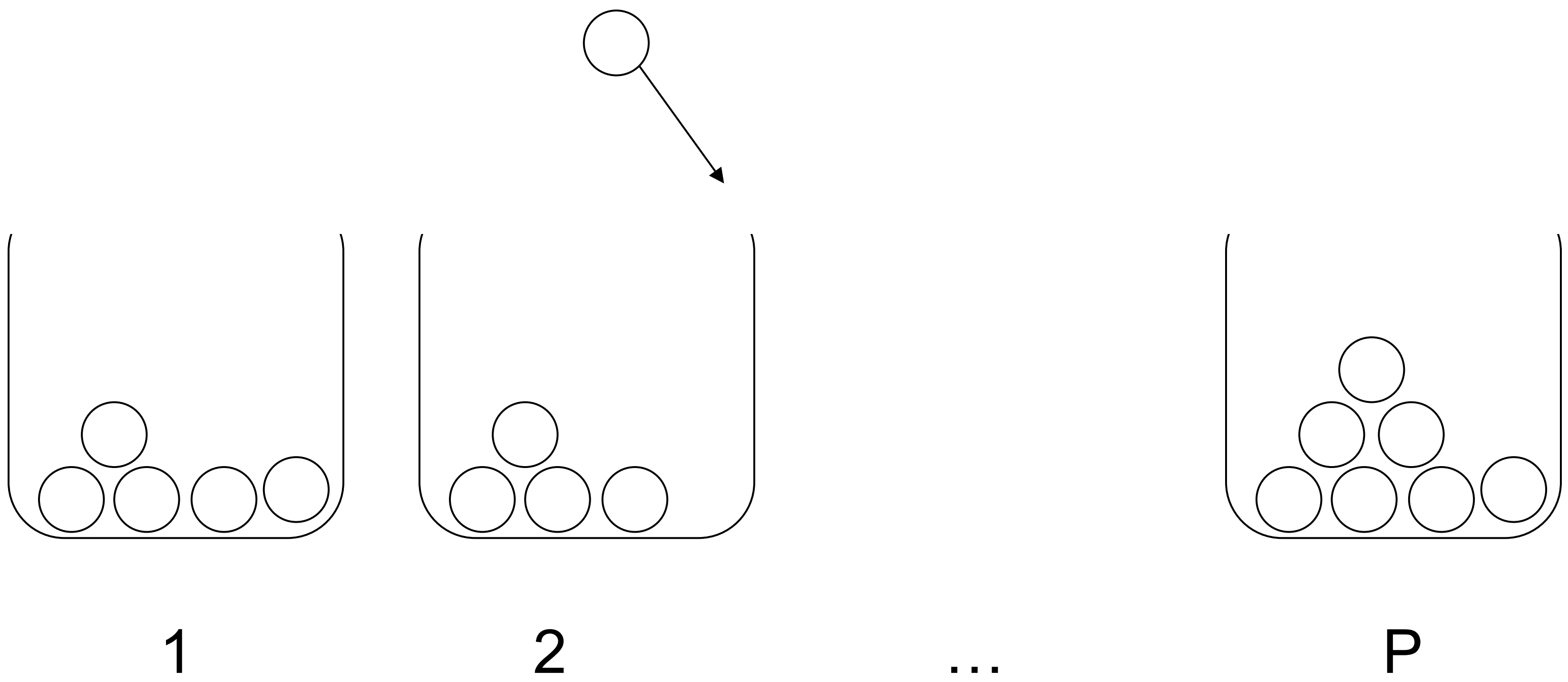


Balls into Bins

Probability of a ball getting into bin #5: $\frac{1}{P}$

We throw N balls randomly into P bins:

Expected size of bin #5:

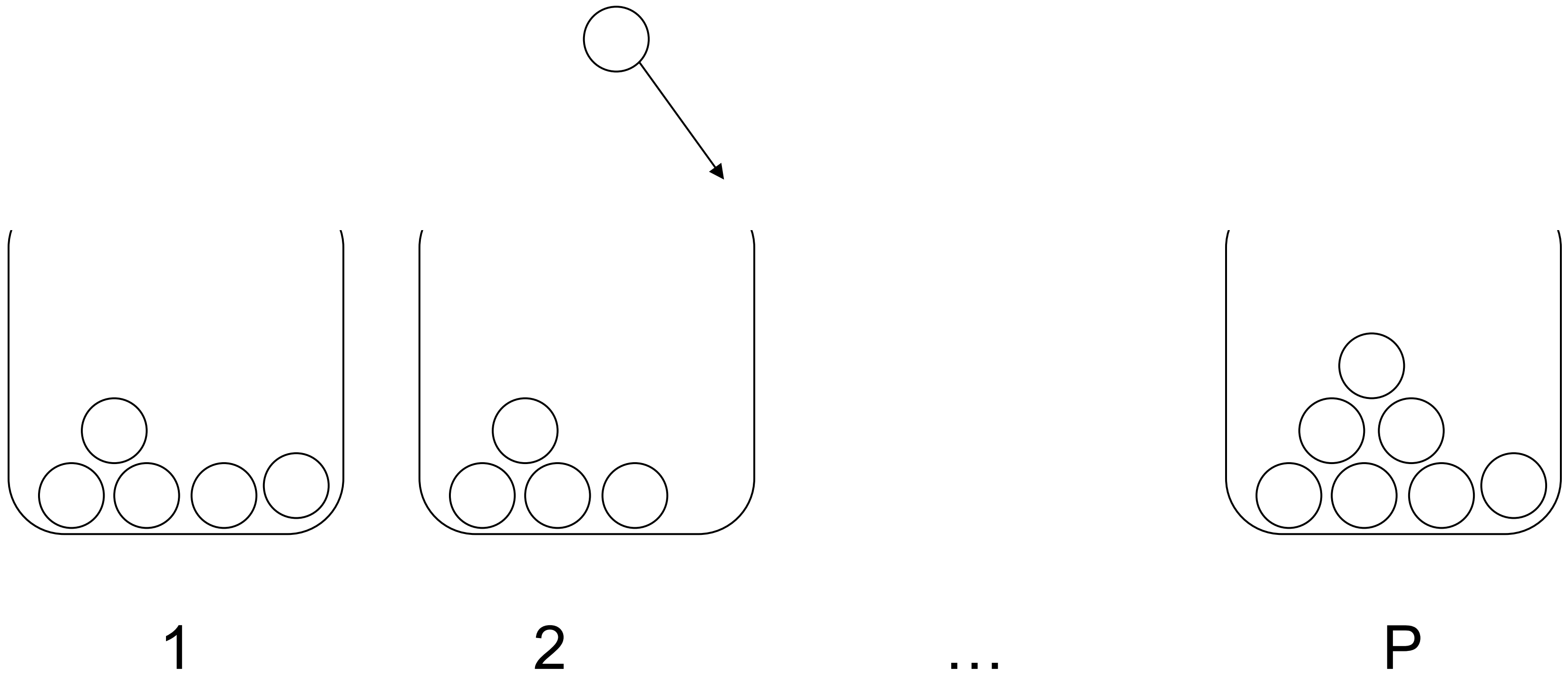


Balls into Bins

Probability of a ball getting into bin #5: $\frac{1}{P}$

Expected size of bin #5: $\frac{N}{P}$

We throw N balls randomly into P bins:



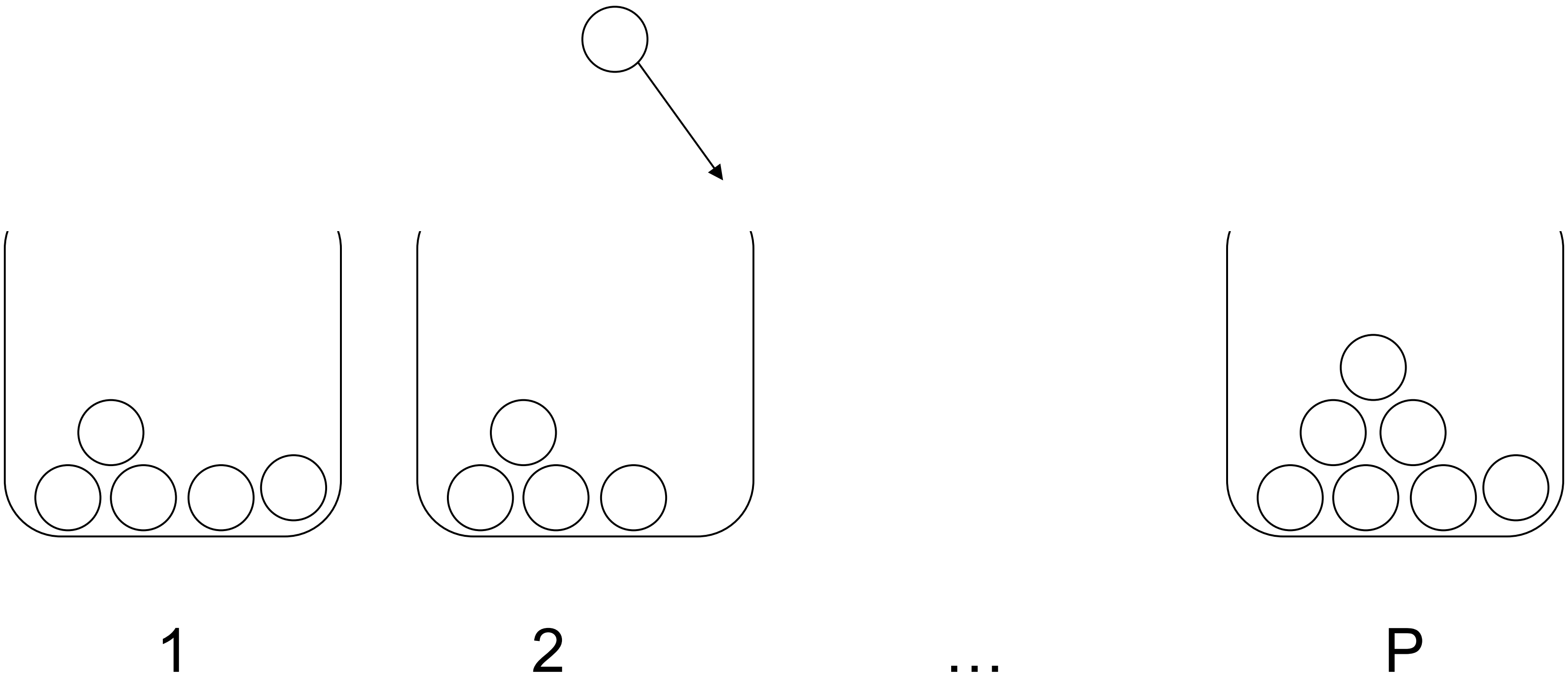
Balls into Bins

We throw N balls randomly into P bins:

Probability of a ball getting into bin #5: $\frac{1}{P}$

Expected size of bin #5: $\frac{N}{P}$

Expected size of the max bin:



Balls into Bins

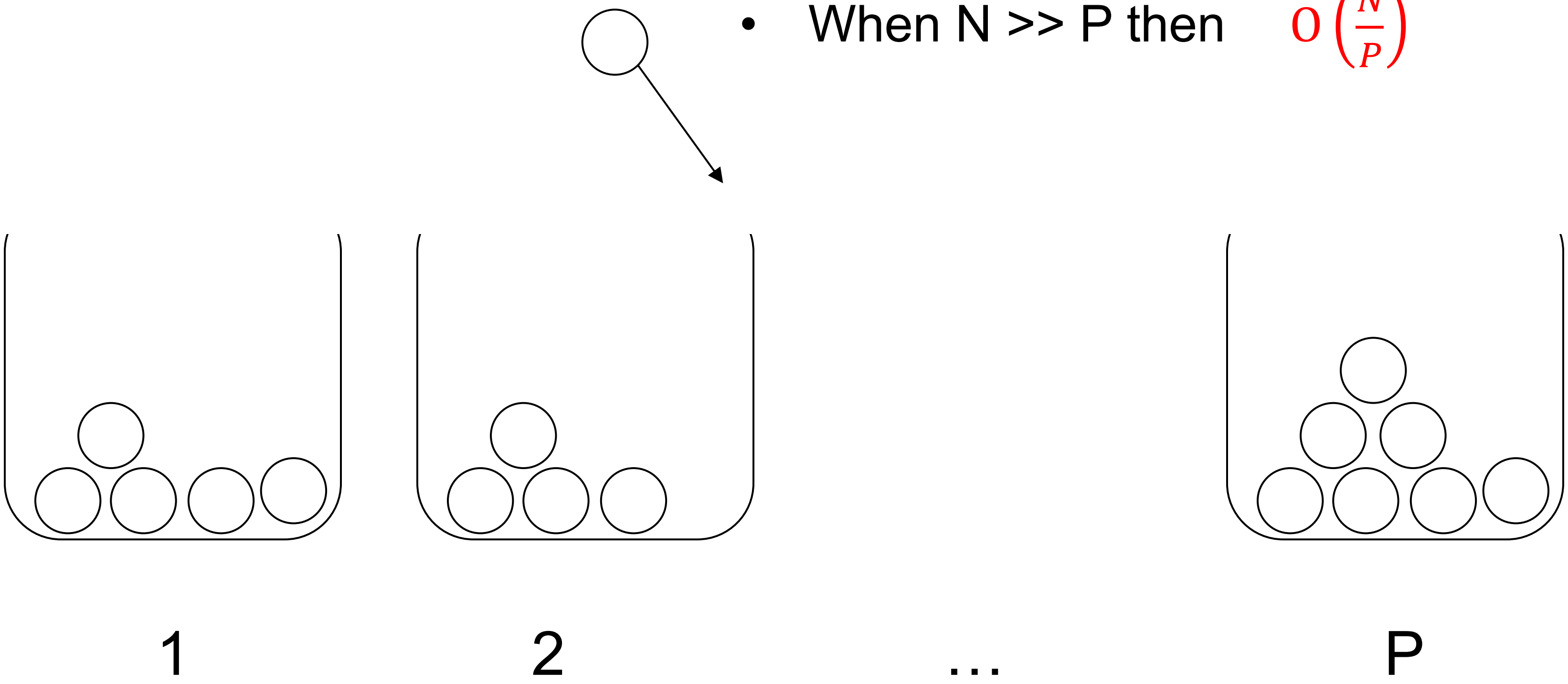
We throw N balls randomly into P bins:

Probability of a ball getting into bin #5: $\frac{1}{P}$

Expected size of bin #5: $\frac{N}{P}$

Expected size of the max bin:

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



Balls into Bins

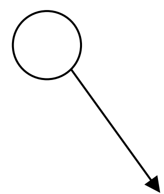
We throw N balls randomly into P bins:

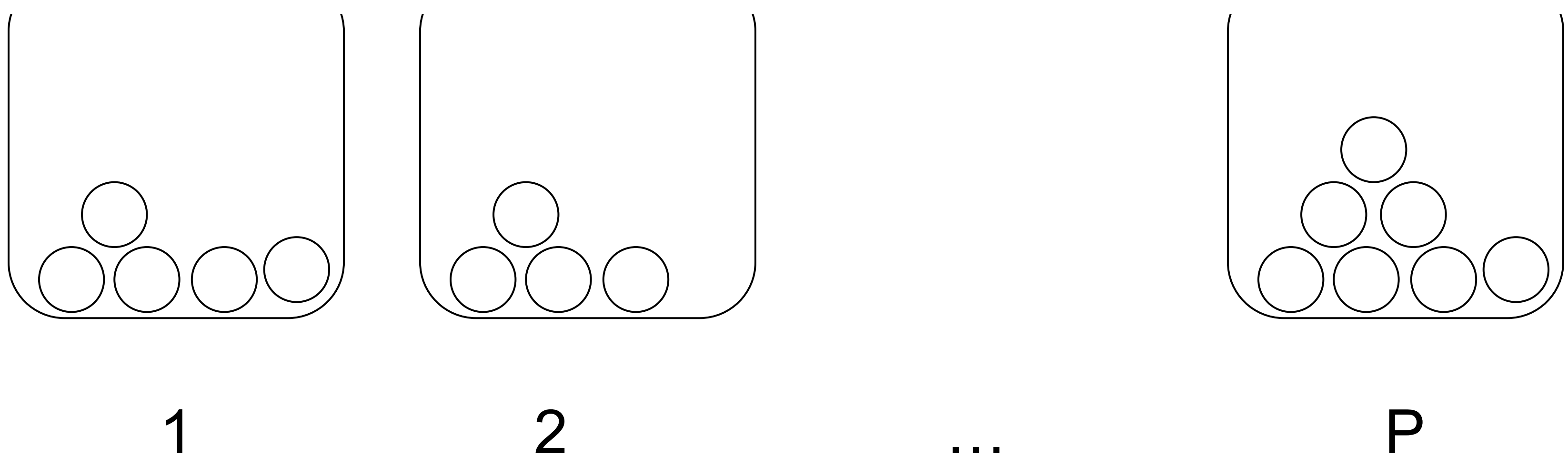
Probability of a ball getting into bin #5: $\frac{1}{P}$

Expected size of bin #5: $\frac{N}{P}$

Expected size of the max bin:

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



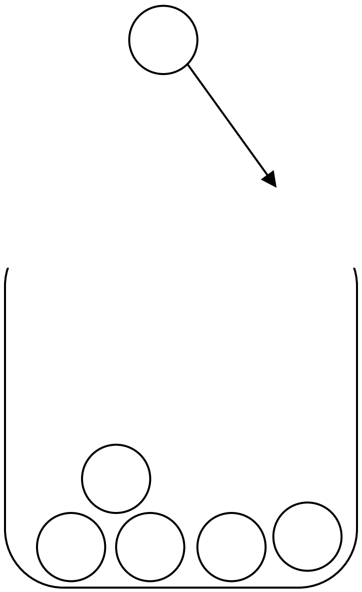


Discussion

- 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 $v_1, \ldots, v_N$



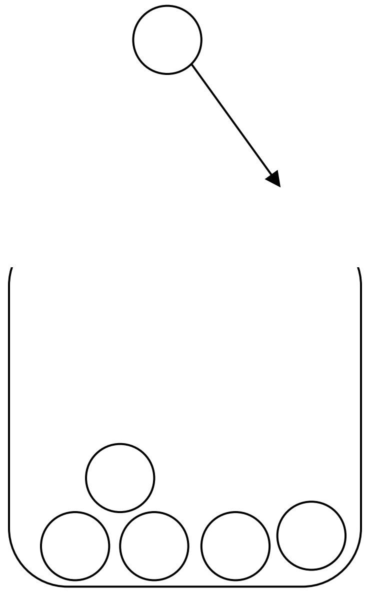
One fixed bin

One Bin

N data items $v_1, \ldots, v_N$

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

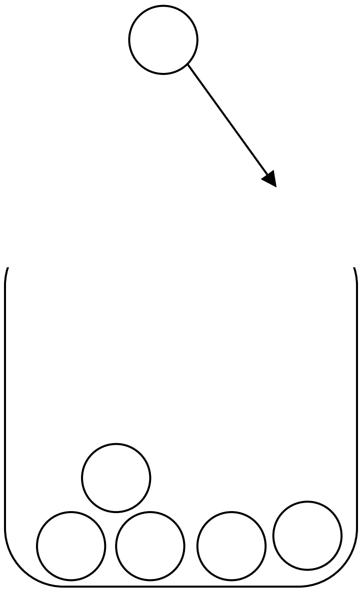
$X_i=1$ if item v_i is sent to our bin
$X_i=0$ if item v_i is sent to a different bin



One fixed bin

One Bin

N data items $v_1, \ldots, v_N$



One fixed bin

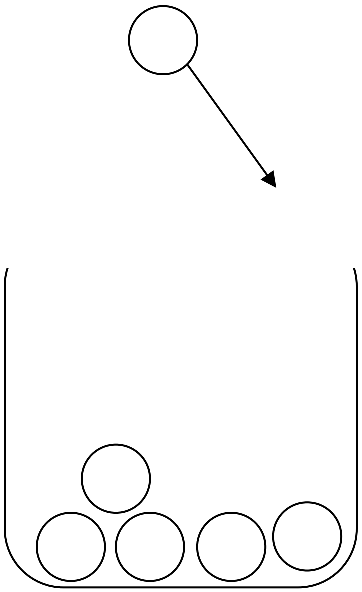
For each data item v_i, let X_i be a r.v. s.t.:

$X_i=1$ if item v_i is sent to our bin
$X_i=0$ if item v_i is sent to a different bin

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

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

One Bin



One fixed bin

N data items $v_1, \ldots, v_N$

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

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

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

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

The Cernoff Bound

Note:
very many
variants

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

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

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

The Cernoff Bound

Note:
very many
variants

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

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

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

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

The Cernoff Bound

Note: very many variants

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

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

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

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

Theorem (Cernoff 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 $v_1, \ldots, v_N$

Distribute on P servers

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

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

Application to Hash Partition

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

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

Skew at j

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

Application to Hash Partition

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

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

Skew at j

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

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

Skew at 1 or at 2 … or at P

Discussion

- We have not computed the expected value of the maximum load
  - I don't even know how to do that
- Instead, we have computed the probability that we exceed the expected load by more than delta

Example 1

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

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

We expect: L = 20,000 items/server

Example 1

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

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

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

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

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

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

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

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

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

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

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

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

Example 2

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

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

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

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

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

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

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

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

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

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%?

$$Pr(\text{bad}) \le 10000 \times \exp\left(-\frac{0.1^2}{3} \times 200\right) = 10000 \times \exp\left(-\frac{2}{3}\right) = 5134$$

>>1

Skew almost certain

Main Take-away

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

To avoid skew, we need N >> P

Application to Heavy Hitters

N data items $v_1, \ldots, v_N$ some are repeated

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

Application to Heavy Hitters

- Assume worst 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 worst 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:

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

Application to Heavy Hitters

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

$$\underbrace{v_1, v_1, \ldots, v_1}_{t}, \underbrace{v_2, v_2, \ldots, v_2}_{t}, \ldots, \ldots, \underbrace{v_{N/t}, v_{N/t}, \ldots, v_{N/t}}_{t}$$

- N/t iid Bernoulli variables:

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

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

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

Application to Heavy Hitters

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

Application to Heavy Hitters

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

We apply Cernoff:

- Skew at one, fixed server:

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

Application to Heavy Hitters

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

We apply Cernoff:

- Skew at one, fixed server:

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

- Skew at any server

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

Main Take-away

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

To avoid skew, we need t << 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

Today

- Skew
- Parallel Query Processing Wrap-up
- Graphs, datalog

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
- Scaleup v.s. Scaleout

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

Case study: Snowflake

Snowflake

- It is an SaaS – what is this? Give other examples of types of cloud services…

Snowflake

- It is an SaaS – what is this? Give other examples of types of cloud services…
- SaaS = software as a service
- Other examples:
  - Platform as a service (PaaS): e.g. Amazon's EC
  - Infrastructure as a service (virtual machines)
  - Function as a Service: Amazon's Lambda

Snowflake

- Describe Snowflake’s Data Storage

Snowflake

- Describe Snowflake's Data Storage

In class:

- S3:PUT/GET/DELETE
- Table → horizontal partition in *files*
- Blobs+PAX
- Temp storage→S3

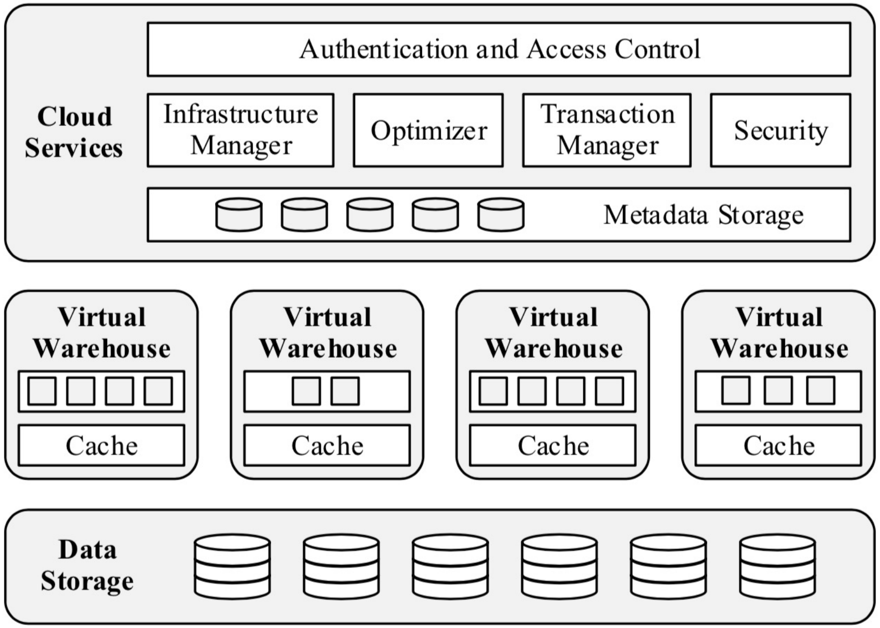


Figure 1: Multi-Cluster, Shared Data Architecture

Snowflake

- Describe Elasticity in Snowflake

- Describe failure handling in Snowflake

Snowflake

- Describe Elasticity in Snowflake
  - Virtual Warehouse (VW) serves one user
  - T-Shirt sizes: X-Small … XX-Large
  - Small query may run on subset of VW
- Describe failure handling in Snowflake

Snowflake

- Describe Elasticity in Snowflake
  - Virtual Warehouse (VW) serves one user
  - T-Shirt sizes: X-Small … XX-Large
  - Small query may run on subset of VW
- Describe failure handling in Snowflake
  - Restart the query
  - No partial retries (like MapReduce or Spark)

Snowflake

- Describe its execution engine

Snowflake

- Describe its execution engine
- Column-oriented (in class)
- Vectorized (“tuple batches” – in class)
- Push-based (in class)

Snowflake

- What does Snowflake use instead of indexes?

Snowflake

- What does Snowflake use instead of indexes?

- “Pruning”: for each file (recall: this is a horizontal partition of a table) and each attribute, it stores the min/max values in that column in that file; may skip files when not needed.

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

Today

- Skew
- Parallel Query Processing Wrap-up
- Graphs, datalog

Graphs

Graph Processing Motivation

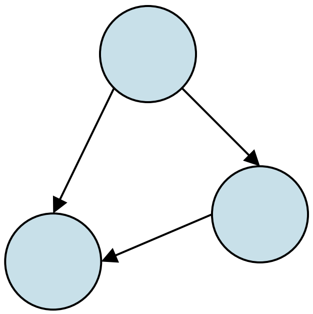
- Many apps need to do analytics on graphs
  - Web graph
  - Social networks
  - Transportation routes
  - Citation graphs
  - Disease propagation graphs
  - …
- A graph: G(V,E)
  - V: Vertices in the graph
  - E: Edges between the vertices
  - Large graph means many edges, not many gigabytes

Graph Analysis

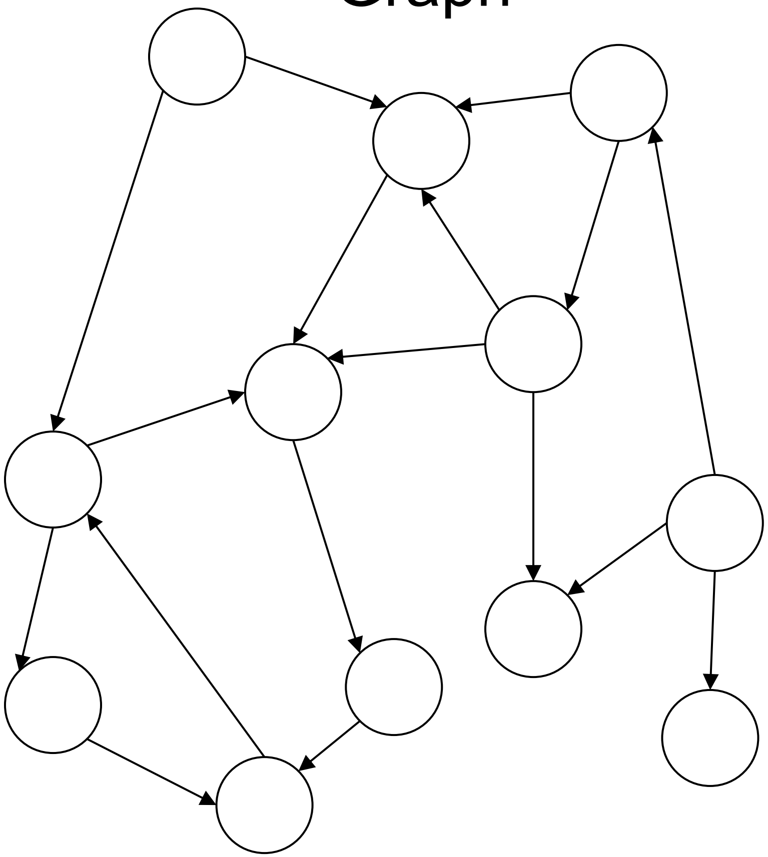
- Graph analytics has several unique properties
  - One large object: the graph
  - Difficult to partition and process in parallel
  - Iterative processing
    - Little work per vertex at each iteration
    - Many iterations & significant amount of communication
- Example applications
  - Shortest path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

Example 1: Pattern Matching

Pattern

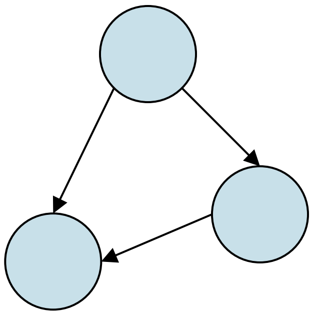


Graph

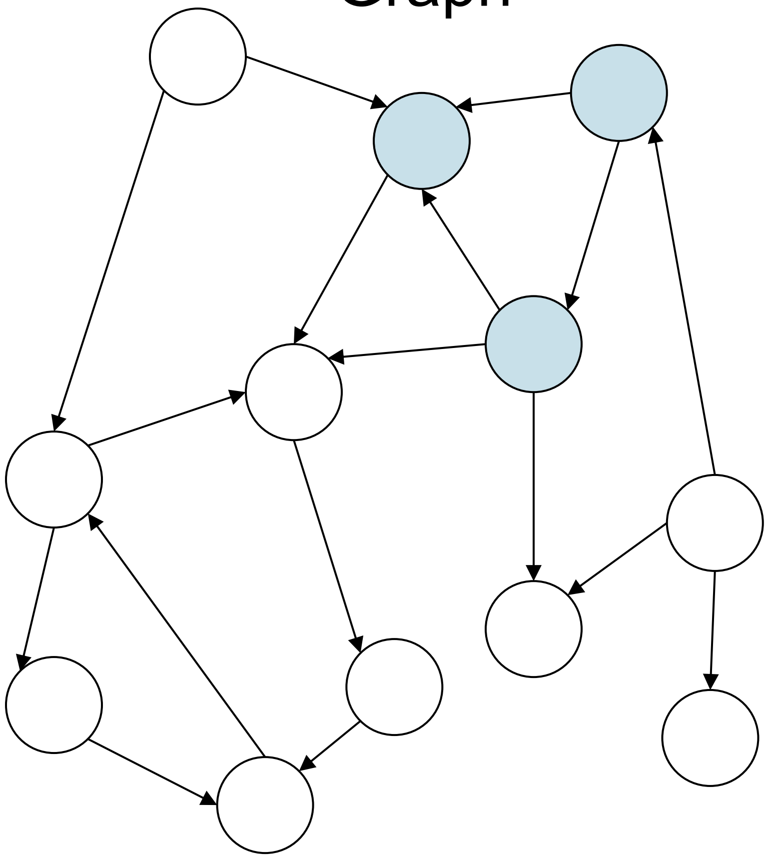


Example 1: Pattern Matching

Pattern

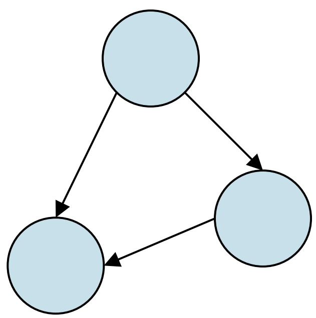


Graph

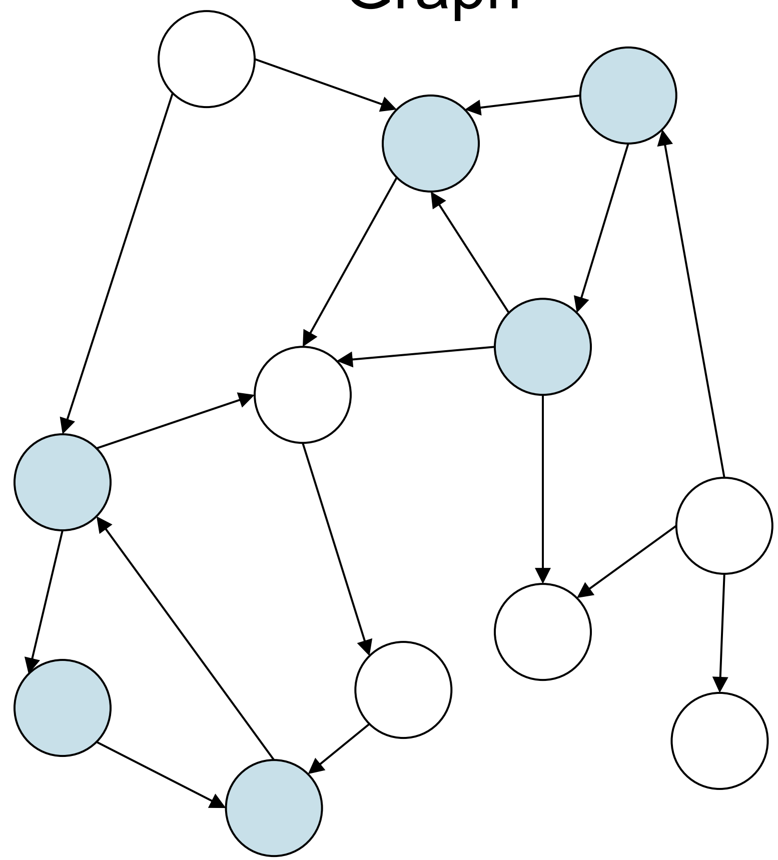


Example 1: Pattern Matching

Pattern

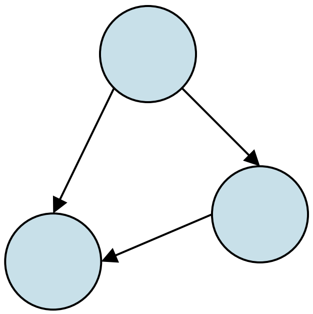


Graph

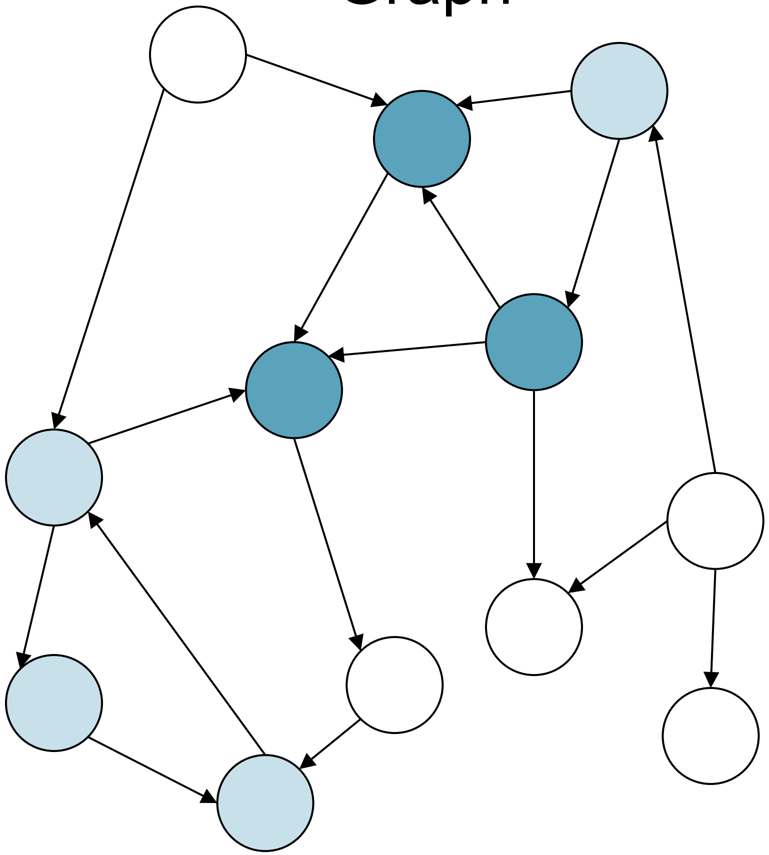


Example 1: Pattern Matching

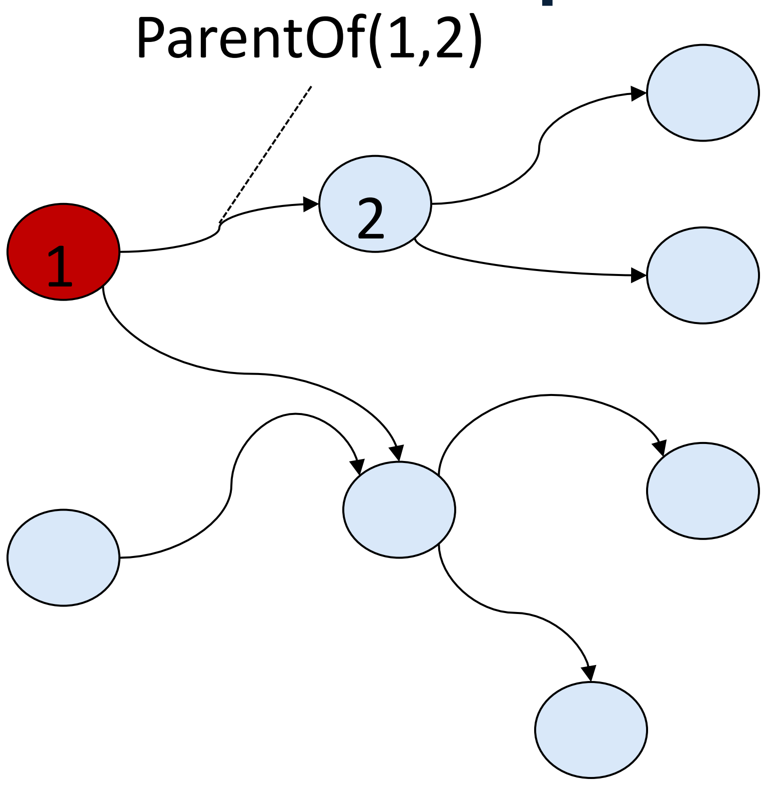
Pattern



Graph



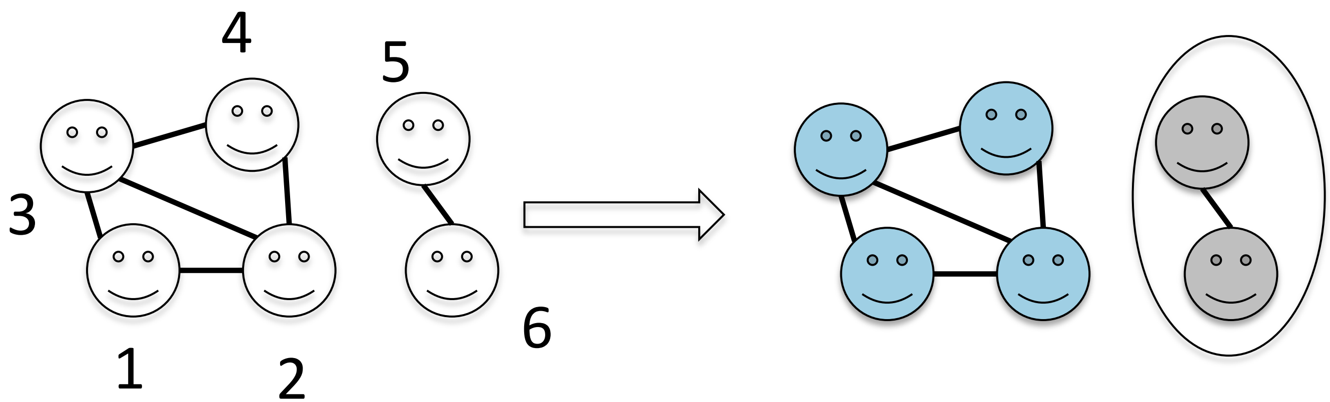
Example 2: Descendants



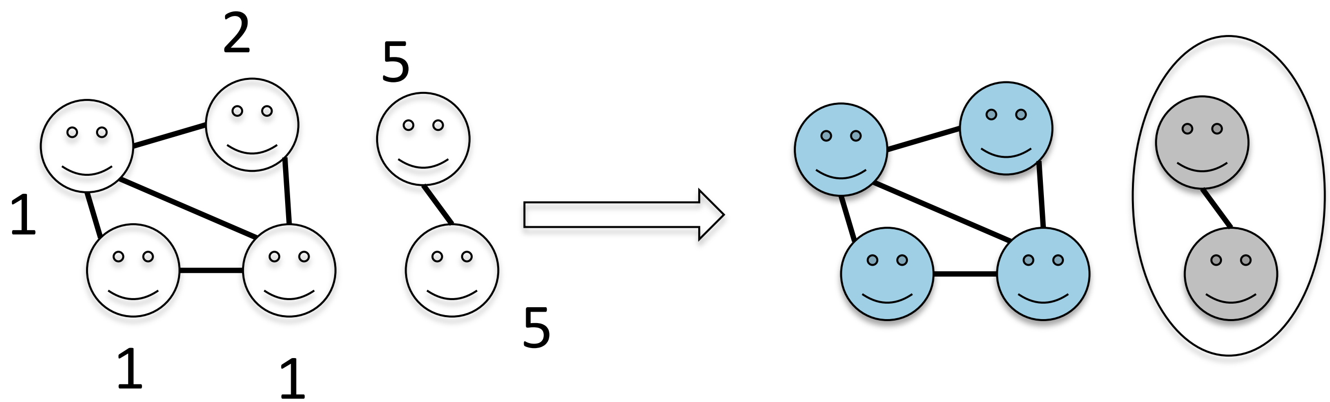
Find all descendants of the red node

Recursively follow the ParentOf links until no new descendants are found

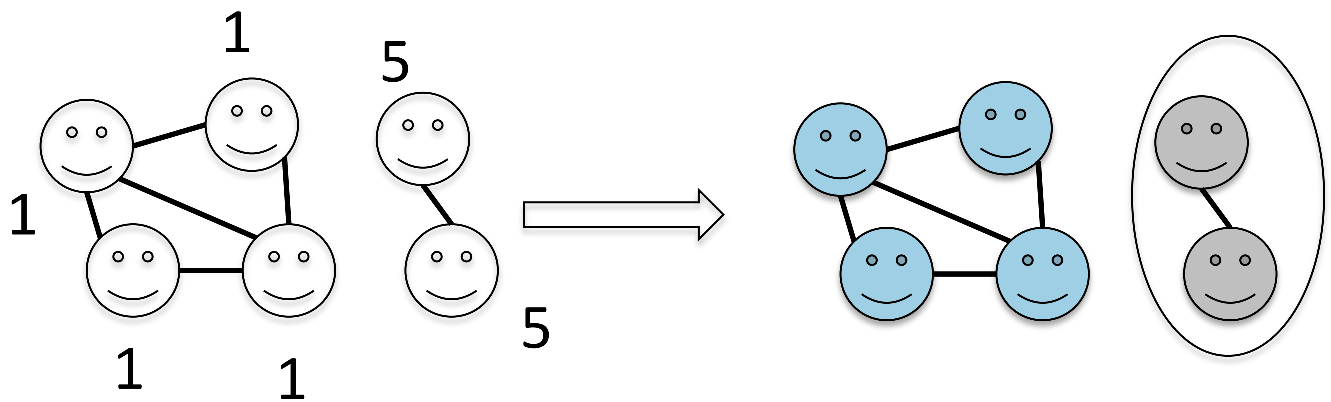
Example 3: Connected Components



Example 3: Connected Components

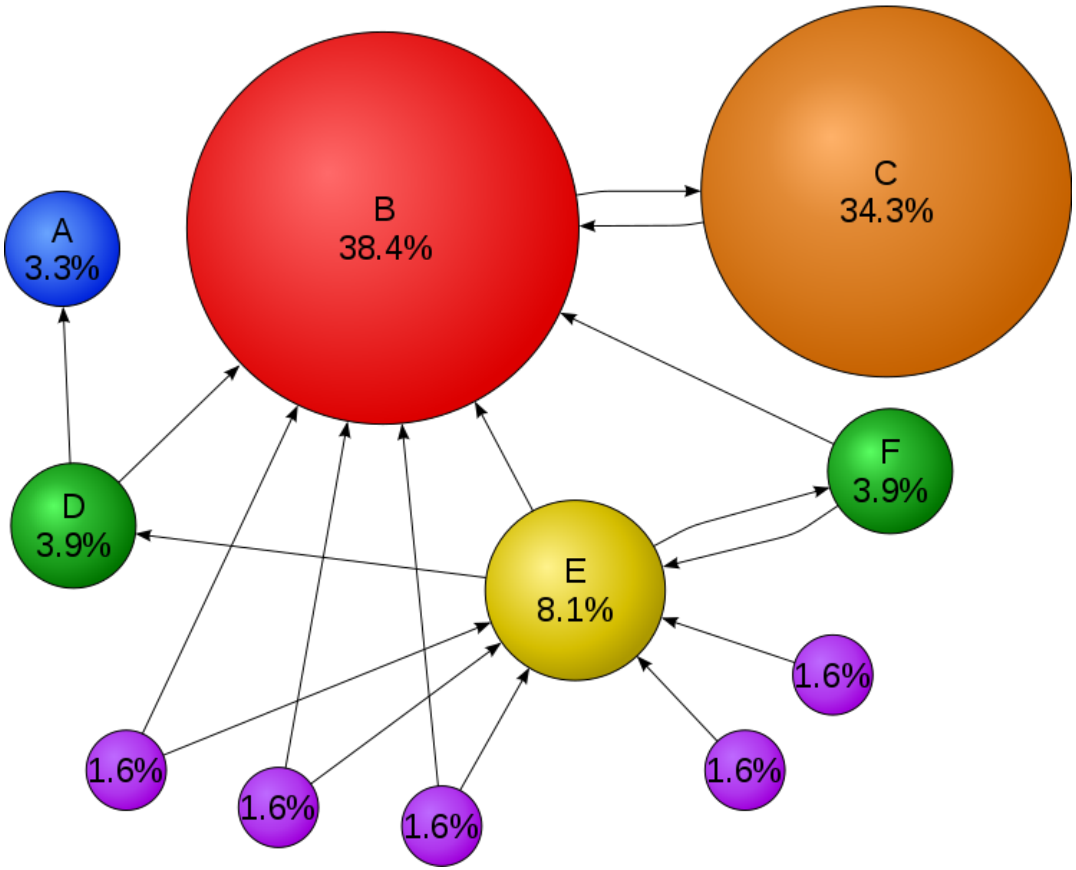


Example 3: Connected Components



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

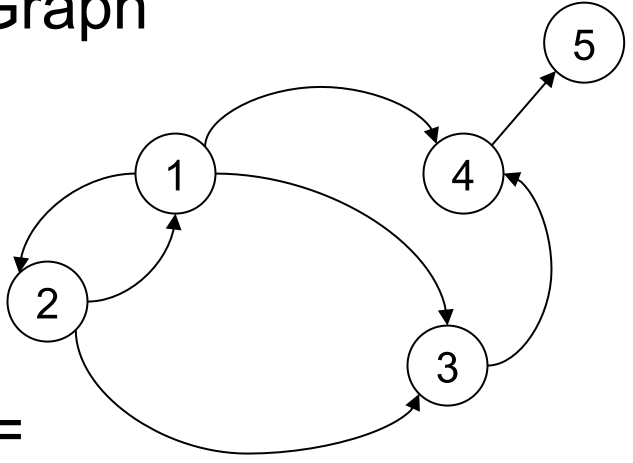
From Wikipedia: https://en.wikipedia.org/wiki/PageRank

How to Model Graph Analytics

- Option 1: Relational Model
  - Relation Edges(v1,v2)
  - 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

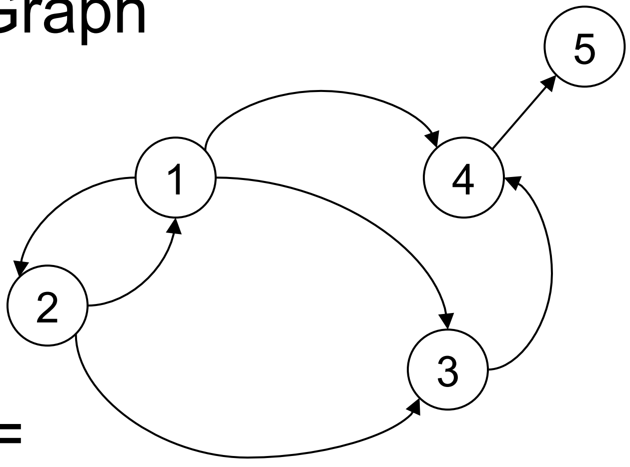


R=

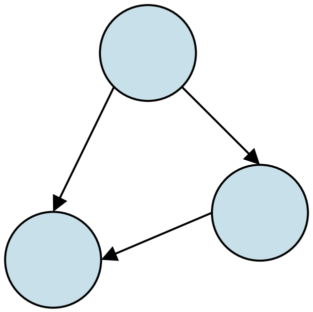
src	dst
1	2
2	1
2	3
1	4
3	4
4	5
1	3

Processing Graphs in SQL

Graph



Pattern Matching

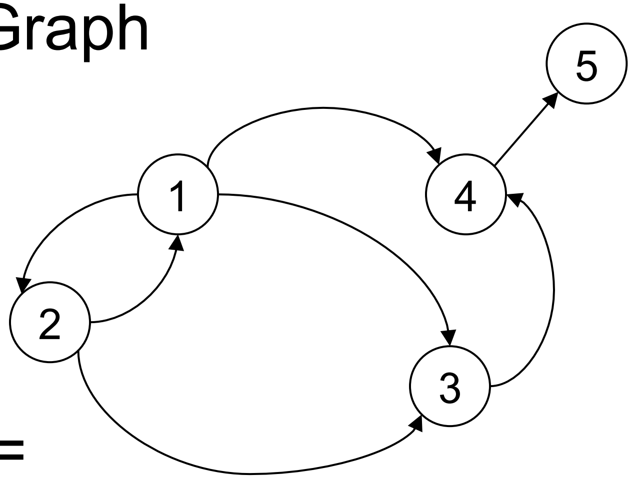


R=

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

Processing Graphs in SQL

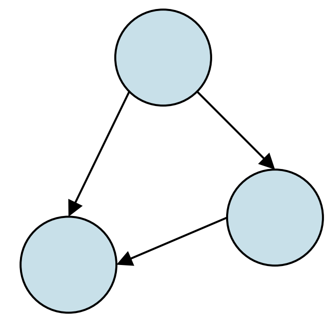
Graph



R=

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

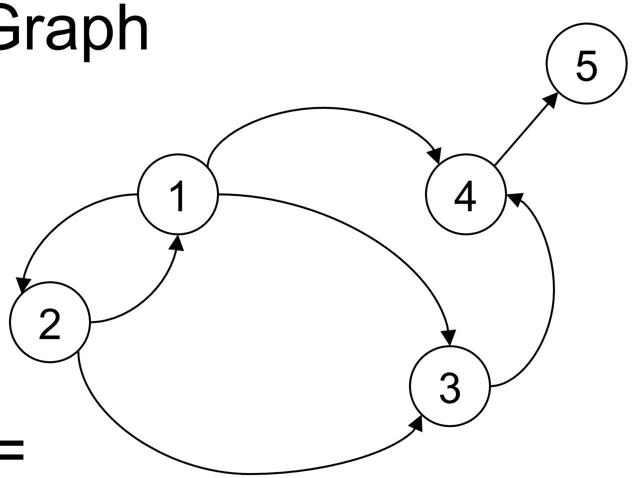
Pattern Matching



```
SELECT …
FROM …
WHERE …
```

Processing Graphs in SQL

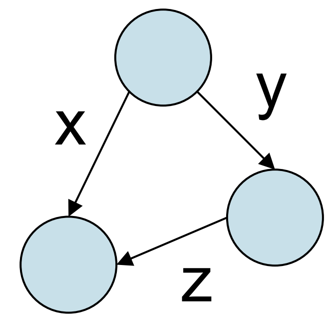
Graph



R=

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

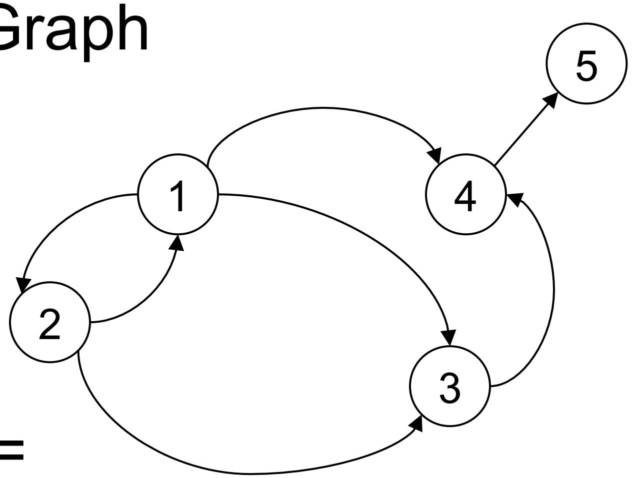
Pattern Matching



SELECT …
FROM …
WHERE …

Processing Graphs in SQL

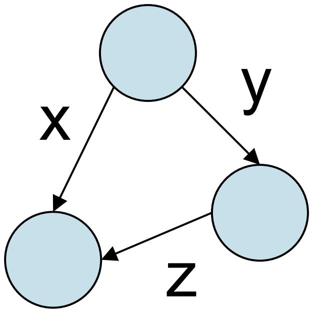
Graph



R=

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

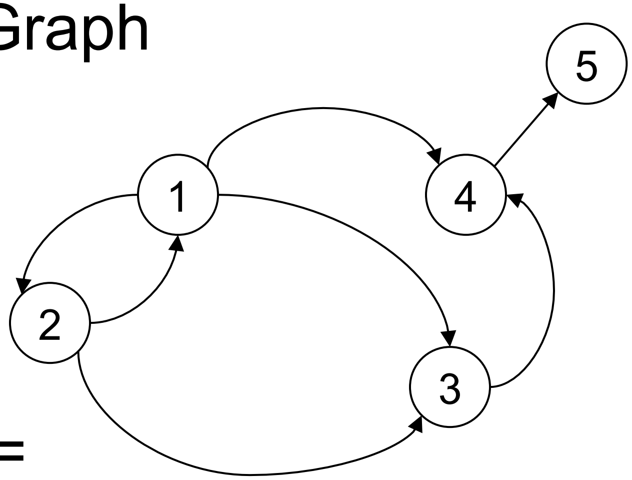
Pattern Matching




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

Processing Graphs in SQL

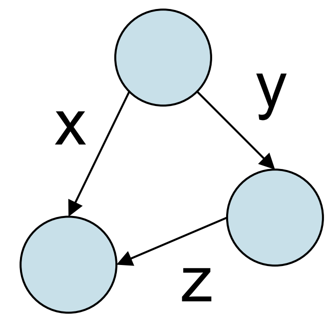
Graph



R=

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

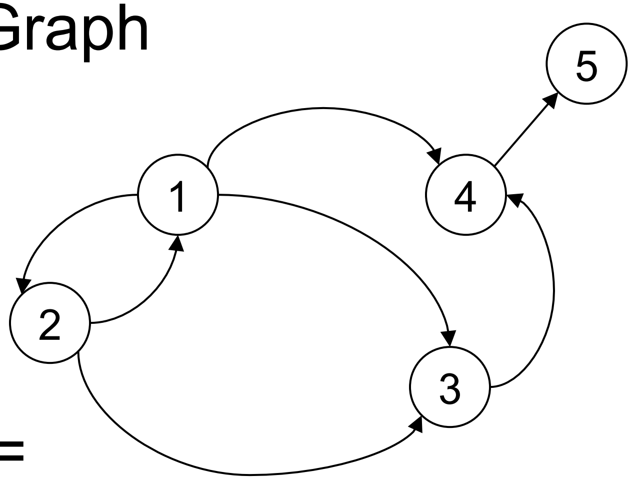
Pattern Matching




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

Processing Graphs in SQL

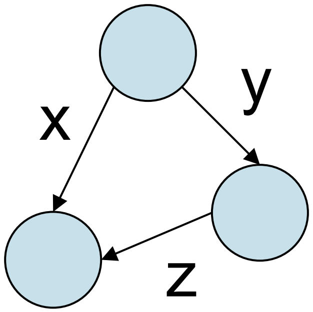
Graph



R=

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

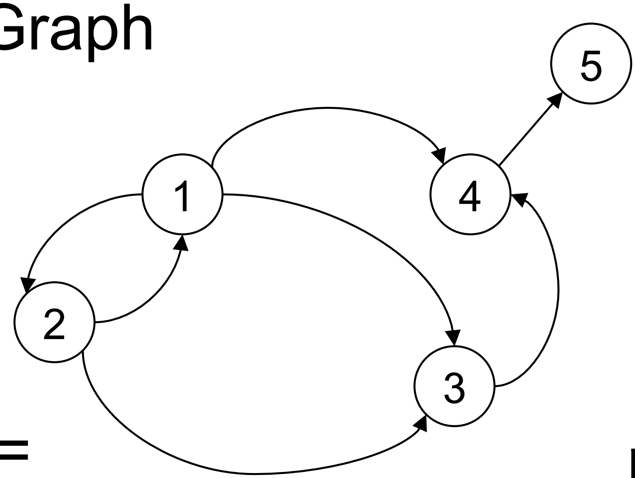
Pattern Matching



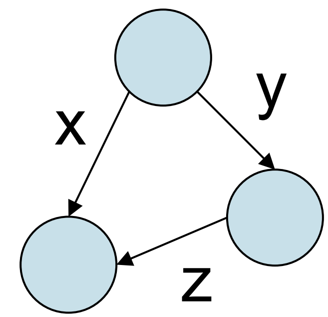

```
SELECT x.src, y.dst, z.dst
FROM R x, R y, R z
WHERE x.src = y.src
      and x.dst = z.dst
      and y.dst = z.src
```

Processing Graphs in SQL

Graph



Pattern Matching



R=

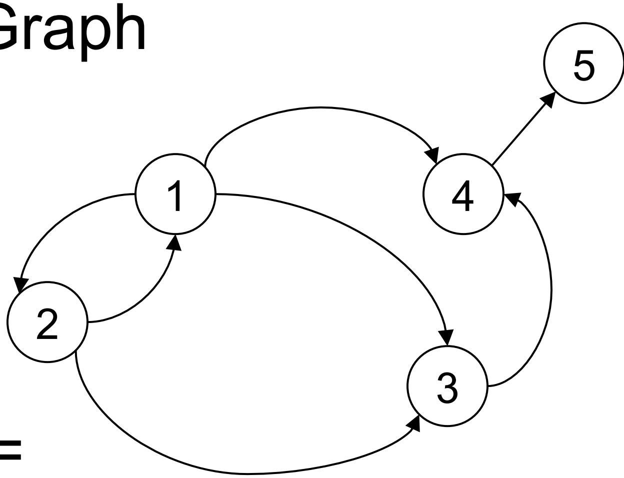
src	dst
1	2
2	1
2	3
1	4
3	4
4	5
1	3

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
```

Processing Graphs in SQL

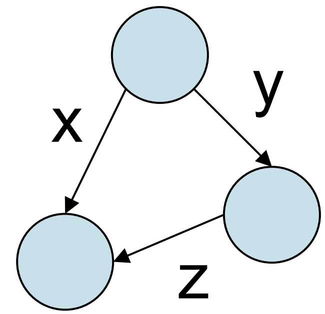
Graph



R=

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

Pattern Matching



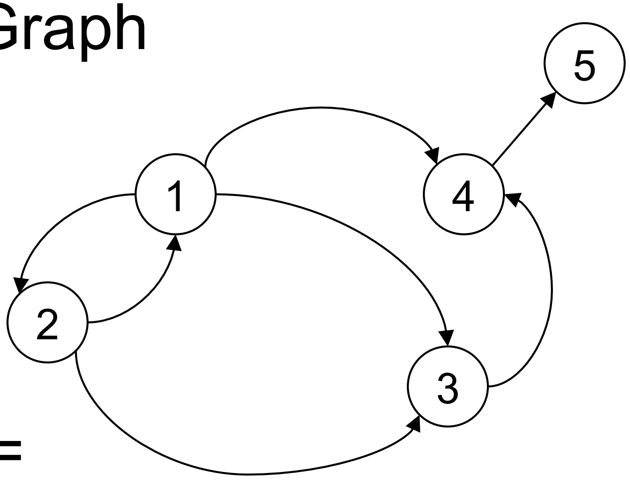
x.src	y.dst	z.dst
1	2	3
1	3	4
2	1	3

```
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



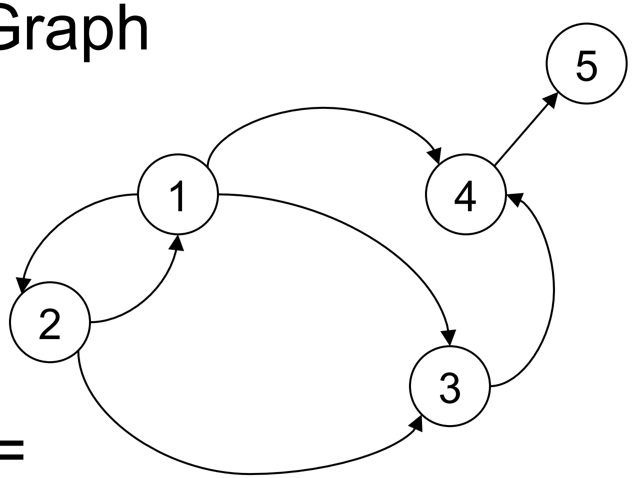
Find Descendants of node 2

R=

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

Processing Graphs in SQL

Graph



R=

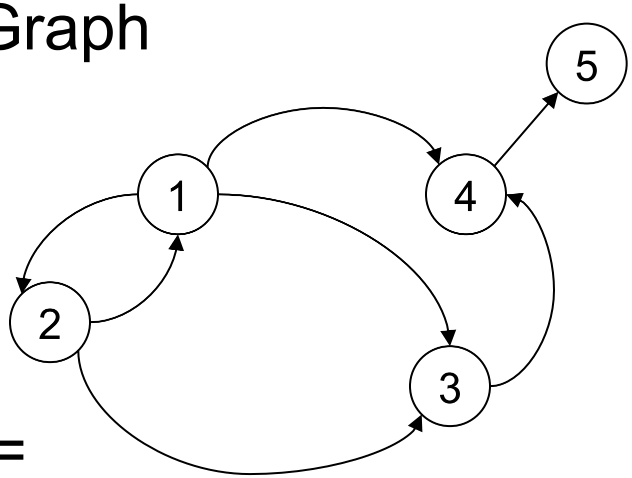
src	dst
1	2
2	1
2	3
1	4
3	4
4	5
1	3

Find Descendants of node 2

Find children:

Processing Graphs in SQL

Graph



R=

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

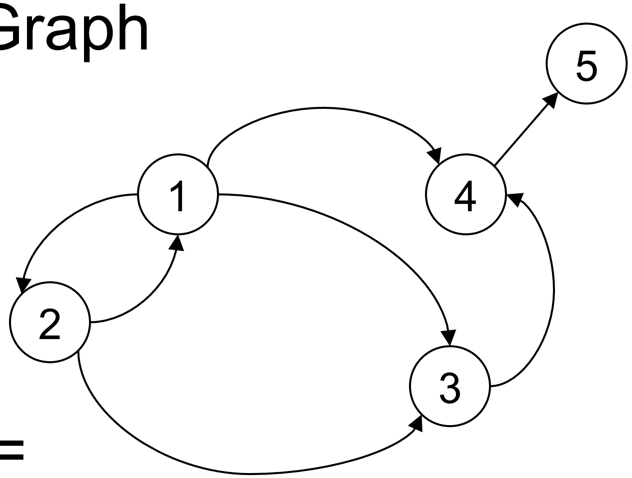
Find Descendants of node 2

Find children:

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

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

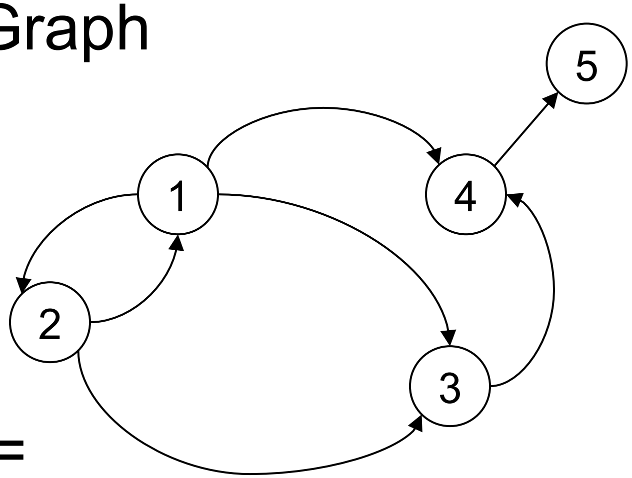
Find Descendants of node 2

Find children:

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

Processing Graphs in SQL

Graph



R=

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

Find Descendants of node 2

Find children:

d
1
3

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

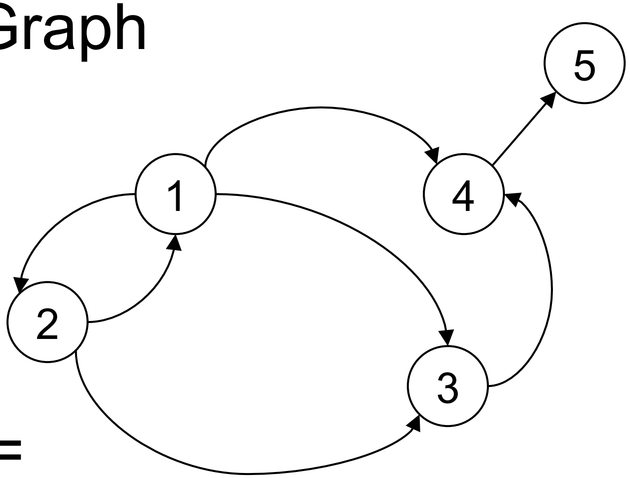
…and their children

2
4

```
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

Find Descendants of node 2

Find children:

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

d
1
3

…and their children

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

2
4

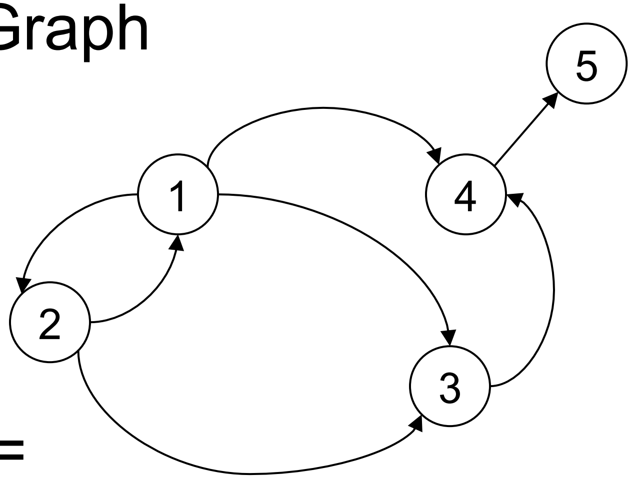
…and their children…

5

…

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

Find Descendants of node 2

Find children:

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

d
1
3

…and their children

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

2
4

…and their children…

5

…

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

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

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

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).
```

```
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

head body

atom atom atom (aka predicate)

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

Anatomy of a Rule

head body

atom atom atom (aka predicate)

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

f, l = head variables

x,y,z = existential variables

Outline

- Datalog rules
- Recursion
- Semantics

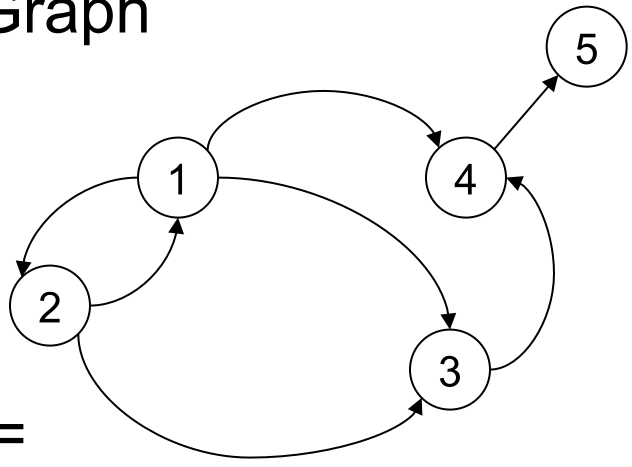
Next time: extensions, semi-naïve algo.

Datalog program

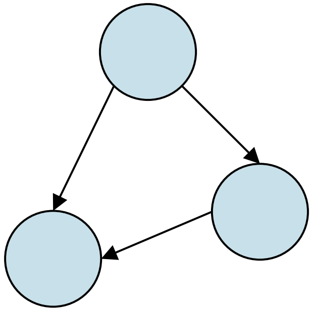
- A datalog program = several rules
- Rules may be recursive
- Set semantics only

Processing Graphs in Datalog

Graph



Pattern Matching

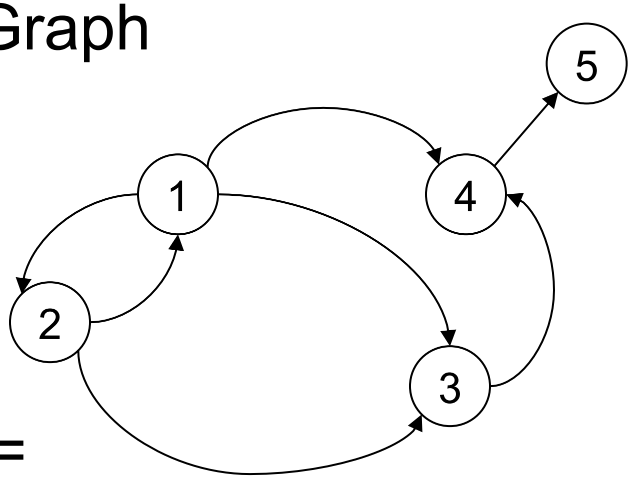


R=

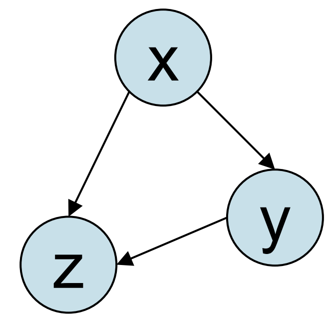
src	dst
1	2
2	1
2	3
1	4
3	4
4	5
1	3

Processing Graphs in Datalog

Graph



Pattern Matching



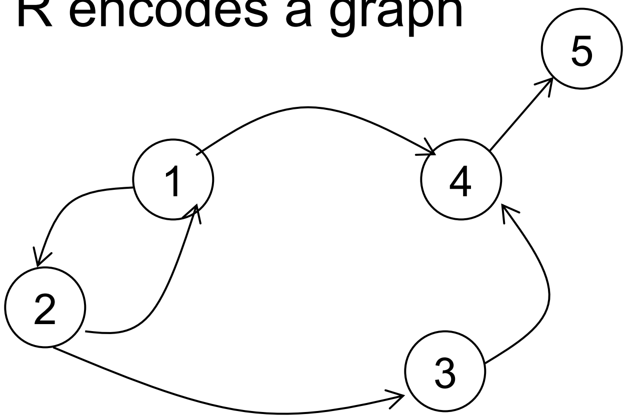
R=

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

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

Example

R encodes a graph



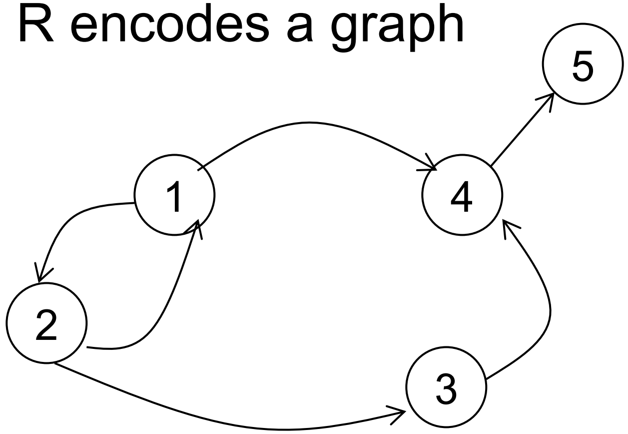
Descendants of node 2

R=

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

Example

R encodes a graph



R=

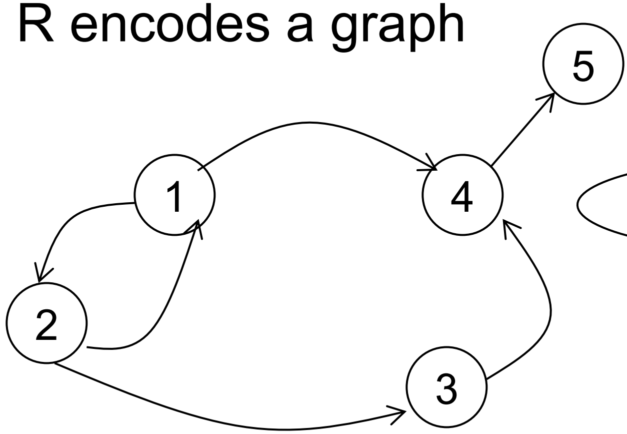
1	2
2	1
2	3
1	4
3	4
4	5

Descendants of node 2

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

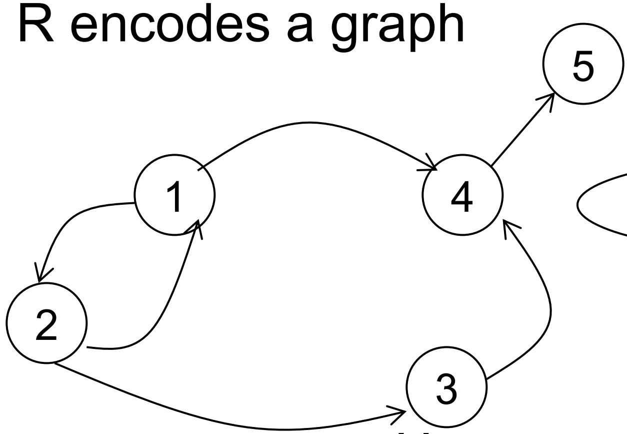
Recursive rule

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

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

Example

R encodes a graph



Descendants of node 2

Recursive rule

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

How recursion works in datalog:

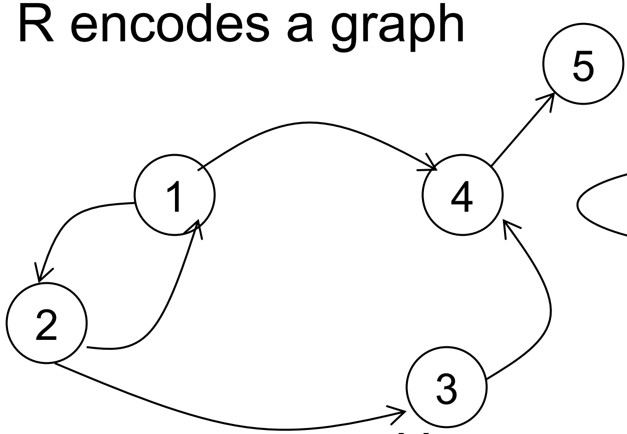
Initially D = empty

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) :- 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:

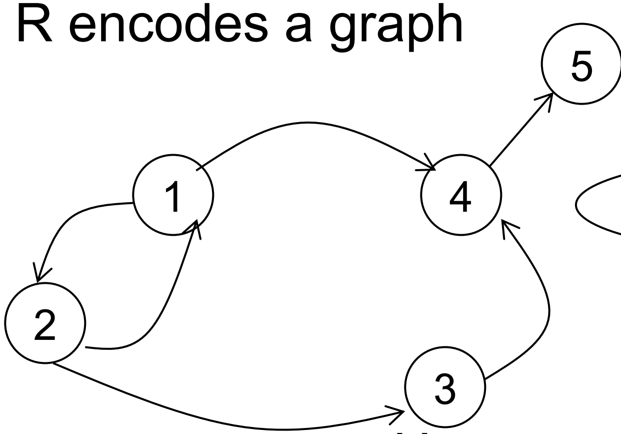
Initially D = empty

- Compute both rules:

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

How recursion works in datalog:

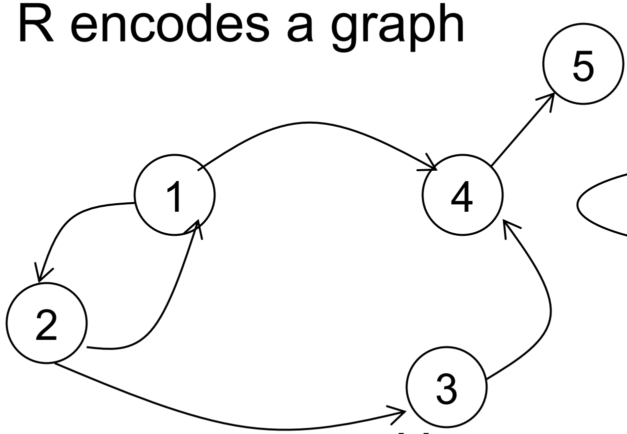
Initially D = empty

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

{1,3}
D(x) :- R(2, x)
{ }
D(y) :- D(x), R(x,y)

Example

R encodes a graph



Descendants of node 2

Recursive rule

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:

{1,3}

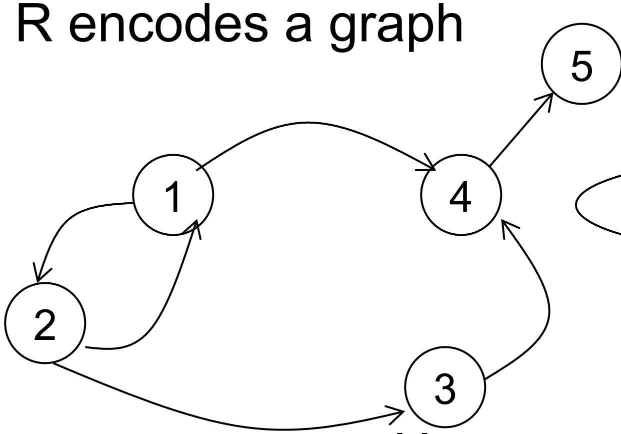
{ }

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

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

Example

R encodes a graph



Descendants of node 2

Recursive rule

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

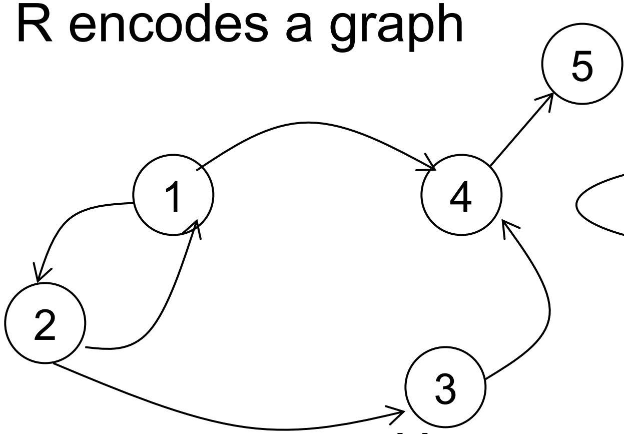
- Compute both rules:
  …now D = {1,3}
- Compute both rules:
  …now D = {1,3,2,4}

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

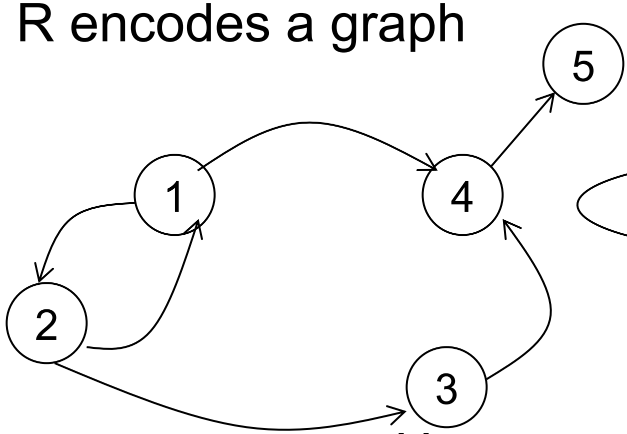
{1,3}
{2,4 }

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

D(x) :- R(2, x)
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) :- R(2, x)
D(y) :- 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: …now D = {1,3,2,4,5}

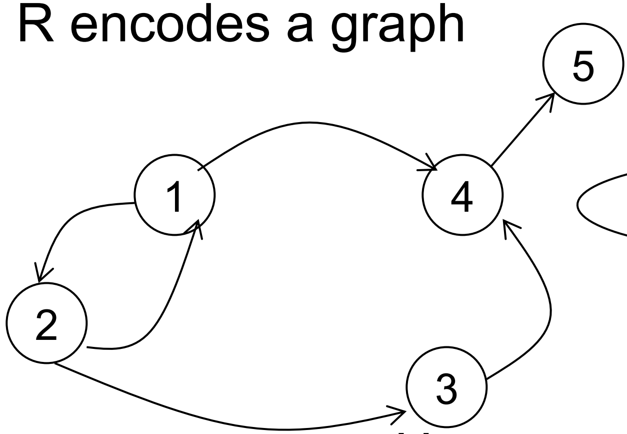
{1,3}
D(x) :- R(2, x)
{ }
D(y) :- D(x), R(x,y)

{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



Descendants of node 2

Recursive rule

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}
- Compute both rules:
  …nothing new. STOP

{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)

Outline

- Datalog rules
- Recursion
- Semantics

Next time: extensions, semi-naïve algo.

Naïve Evaluation Algorithm

- Every rule → SPJ[*] query

[*]SPJ = select-project-join
[+]USPJ = union-select-project-join

Naïve Evaluation Algorithm

- Every rule → SPJ* query

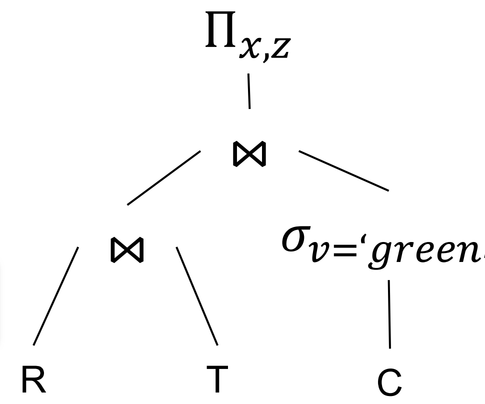
```
T(x,z) :- R(x,y), T(y,z), C(y,'green')
```

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

Naïve Evaluation Algorithm

- Every rule → SPJ* query

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

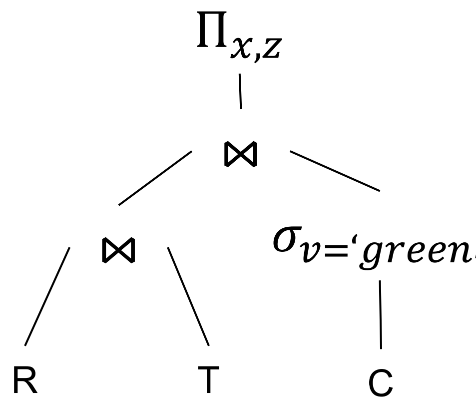


*SPJ = select-project-join
+USPJ = union-select-project-join

Naïve Evaluation Algorithm

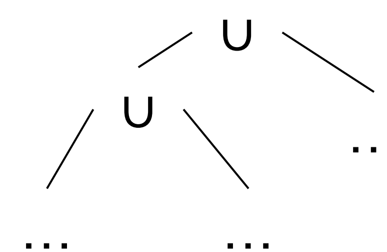
- Every rule → SPJ* query

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



- Multiple rules same head → USPJ+

```
T(x,y) :-  …
T(x,y) :-  …
      …
```

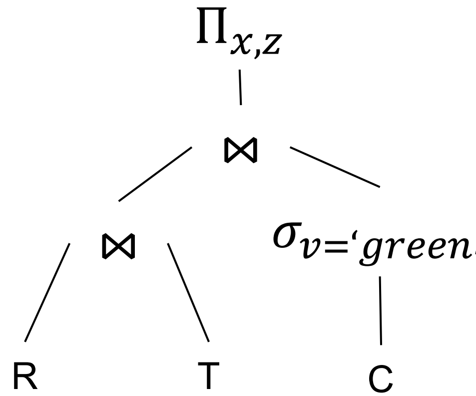


*SPJ = select-project-join
+USPJ = union-select-project-join

Naïve Evaluation Algorithm

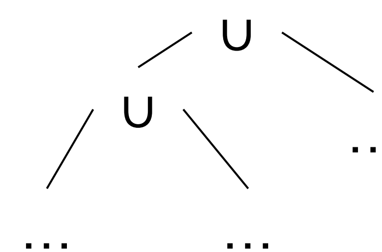
- Every rule → SPJ* query

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



- Multiple rules same head → USPJ+

T(x,y) :- …
T(x,y) :- …
…



- 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