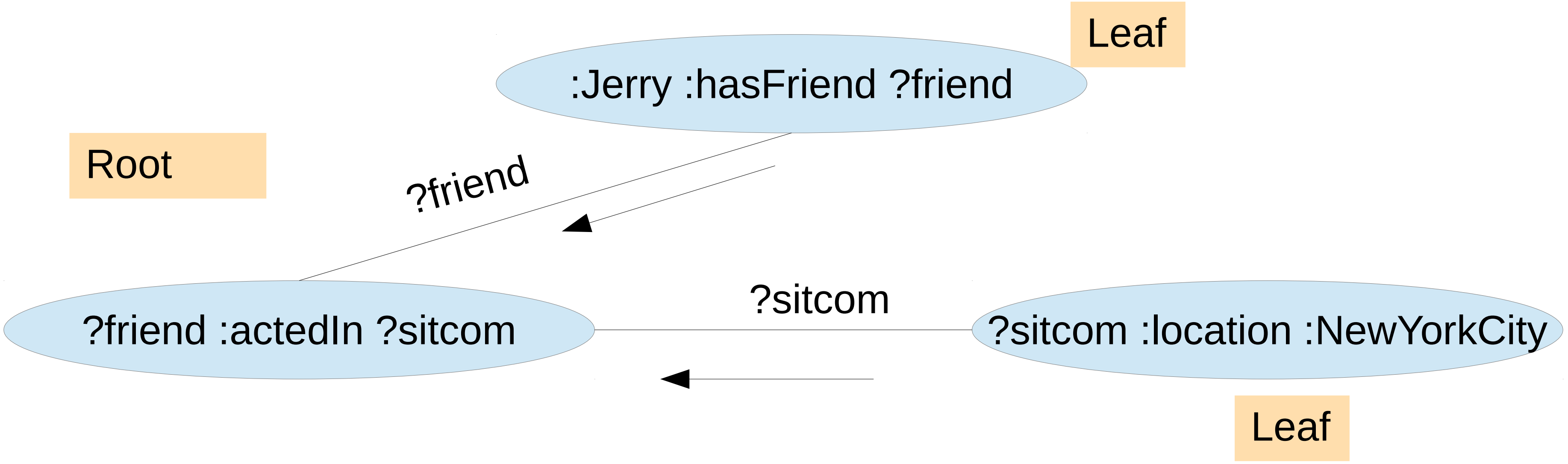
CS698F Advanced Data Management

Instructor: Medha Atre

Reminder

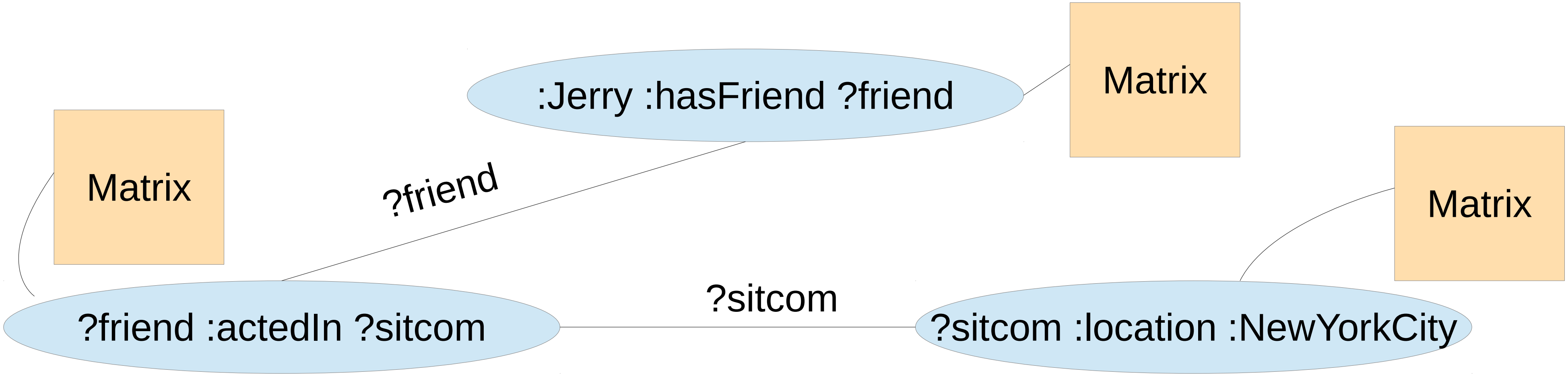
First assignment presentations
On
Sept 6 and 8

Graph of Tables



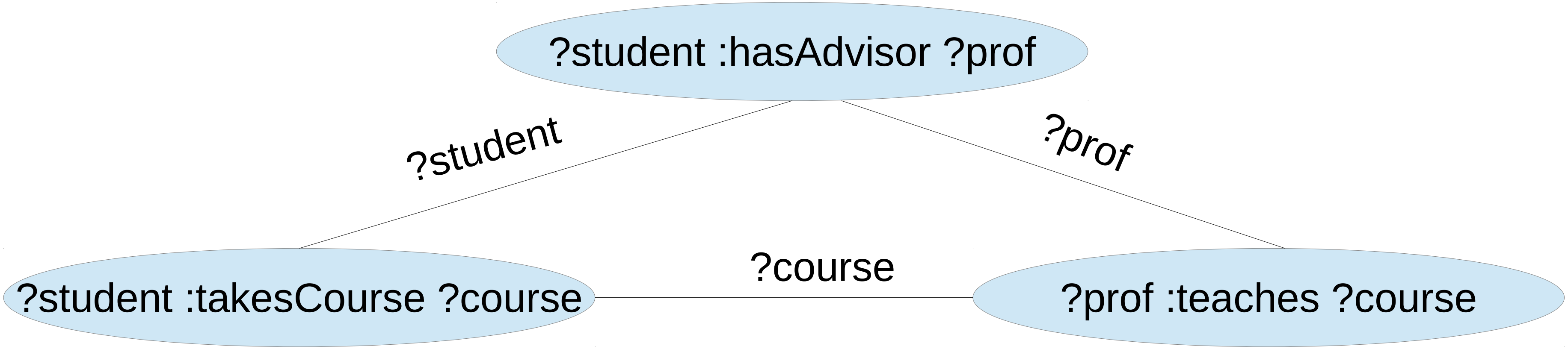
We stopped here in our query, hence *:hasFriend* matrix has redundent tuple.

Graph of Tables



Once done with semi-joins, perform multi-way-pipelined join. Starting from any table/matrix, continue recursively matching the cells from its neighbors, output one result when done matching across all matrices.
When matched **all** the cells in **all** the matrices → you have generated all the results

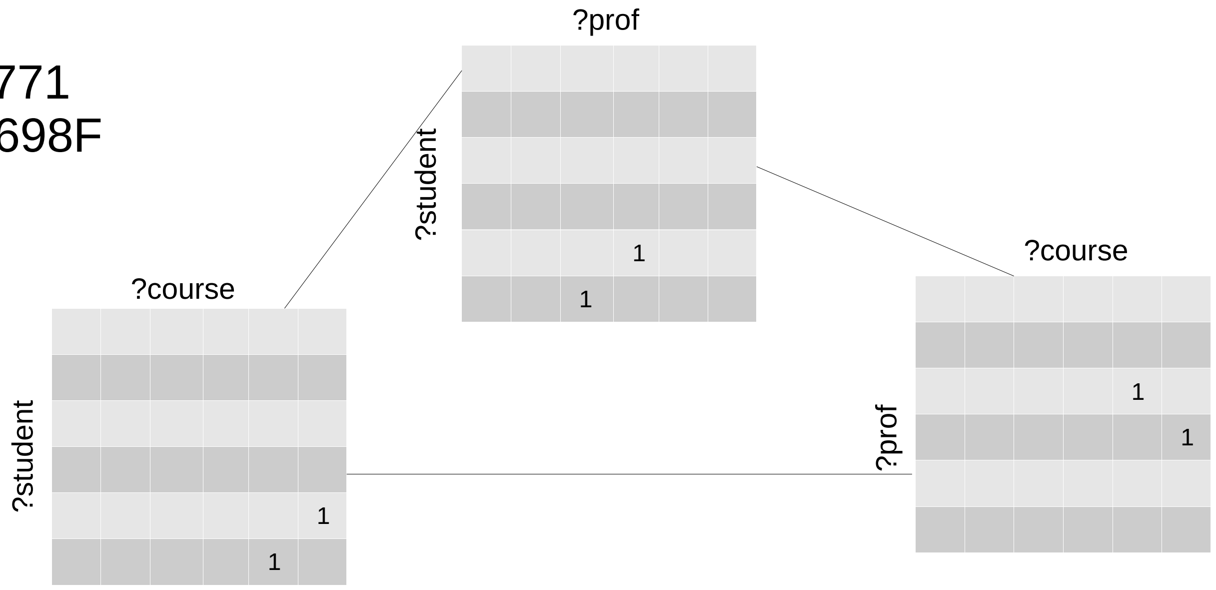
Cyclic graph of tables



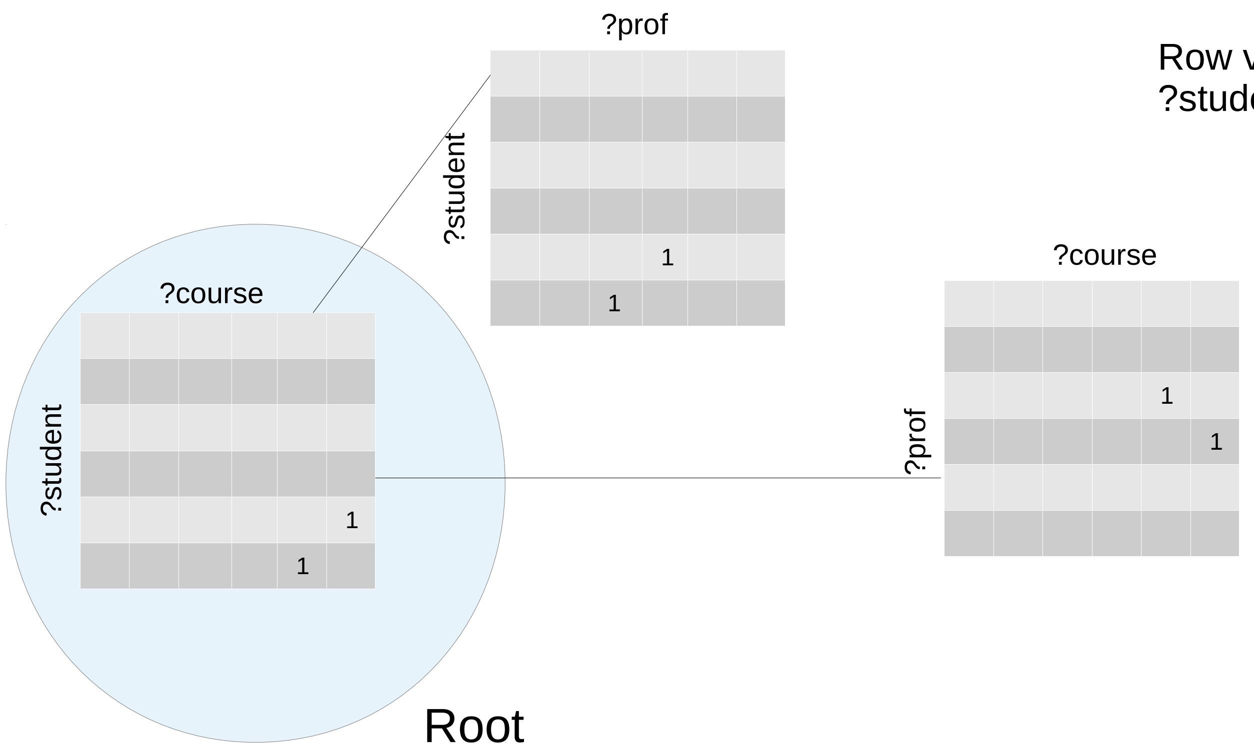
Cyclic graph of tables

:Rajesh :hasAdvisor :Atre
:Suresh :hasAdvisor :Ganguly
:Atre :teaches :CS698F
:Ganguly :teaches :CS771
:Rajesh :takesCourse :CS771
:Suresh :takesCourse :CS698F

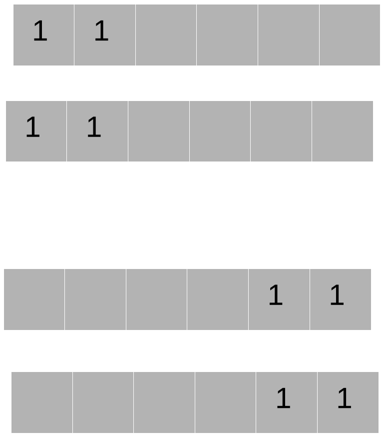
:Rajesh → 1, :Suresh → 2, :Atre → 3,
:Ganguly → 4, :CS771 → 5, :CS698F → 6



Cyclic graph of tables



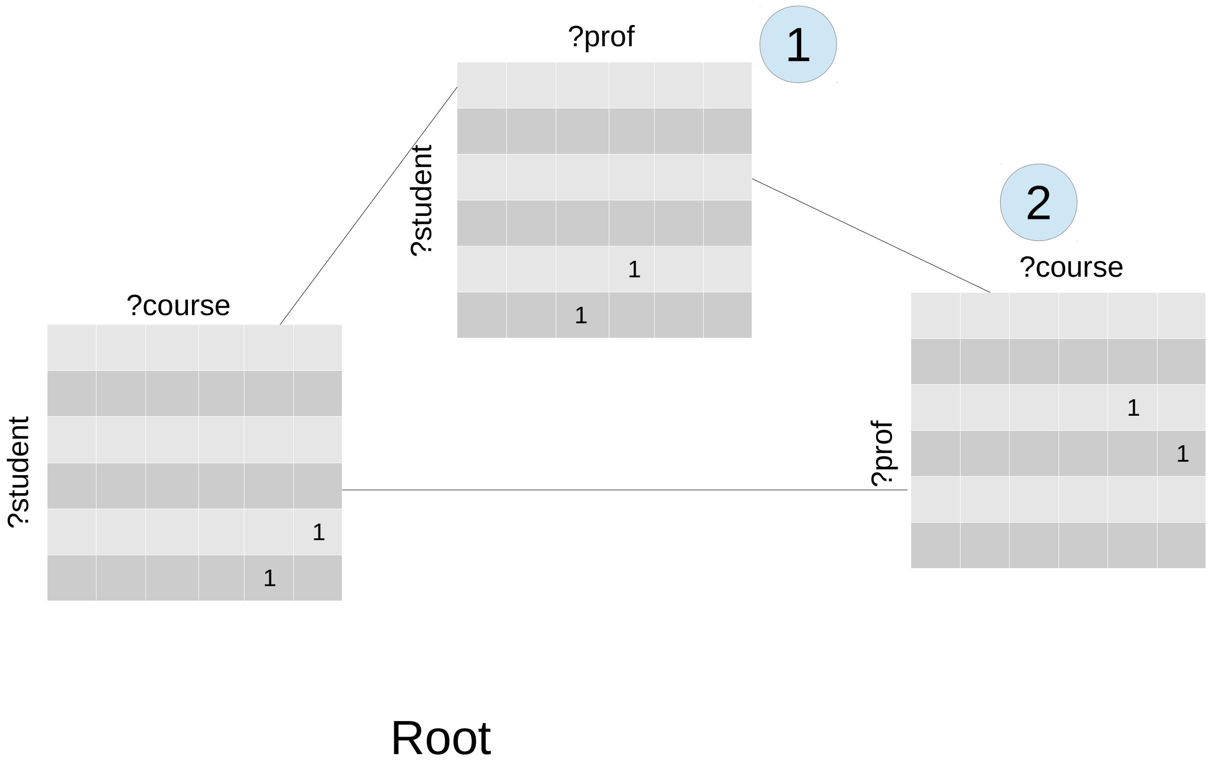
Row vecs for ?student



Col vecs for ?course

Nothing changes!

Multi-way-join cyclic queries



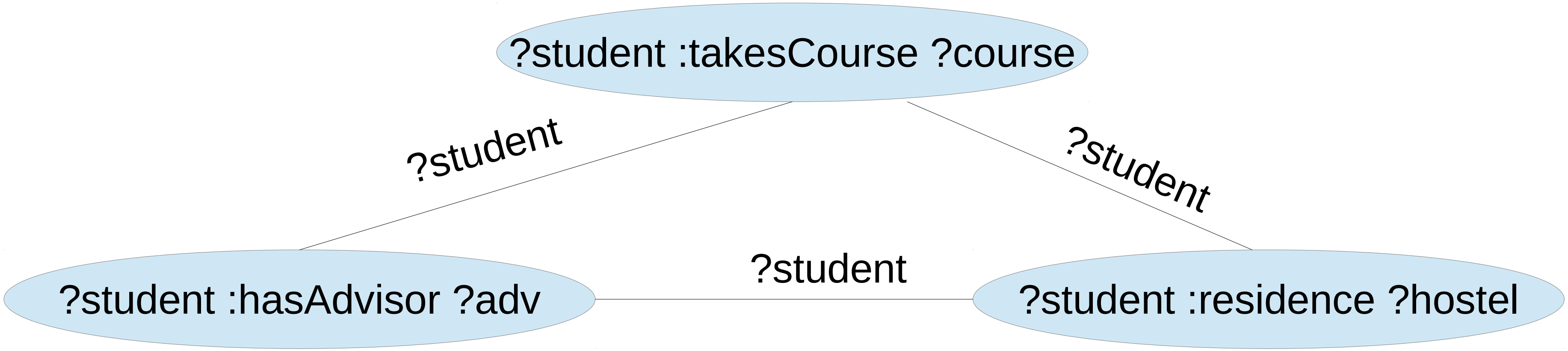
(1, 3) match (3,…) → (1, 3), (3, 6)
Match (3, 6) to (…., 6)
(1, 3), (3, 6), (2, 6)

WAIT!
Mismatch in (**1**, 3) (**2**, 6)
Discard the match, and backtrack.

3rd row in mat-2 has only 1 bit, so again backtrack.

(2, 4) match (4,…) → (2, 4), (4, 5)
Match (4, 5) to (…, 5)
(**2**, 4) (4, 5) (**1**, 5) mismatch!

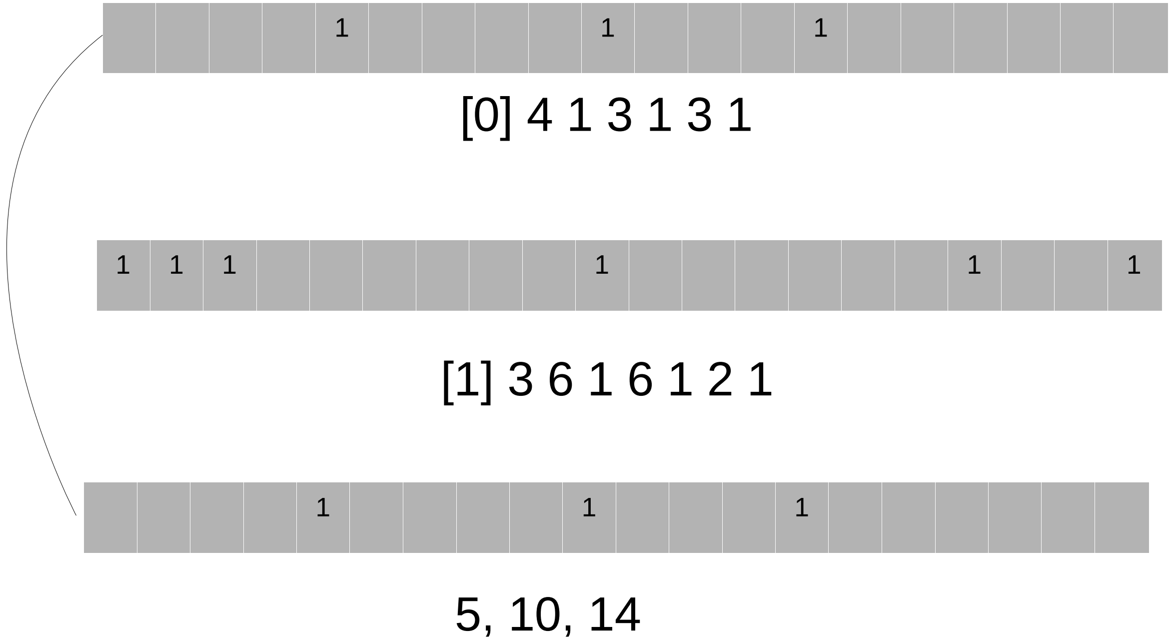
Redundant cycles



Data compression

- Adjacency matrices are very sparse.
- Few 1 bits and lot of 0 bits.
- Compression techniques
  - Run-length-encoding
  - Byte-aligned Bitmap Code (BBC)
  - Word Aligned Hybrid (WAH)
  - Patitioned Word-Aligned Hybrid (PWAH)
  - Others

Run-length-encoding



Handling compressed data

- How to do Boolean AND/OR on compressed bitvector?
  - Without uncompressing, go on reading run-lengths
  - e.g. [0] 3 1 3 AND [1] 1 3 1 => [0] 3… slide the window
  - [1] 1 3... AND [0] 1 1 => [0] 1 add to the prev => [0] 4… so on
  - For very sparse vs dense vector, go over set bits in sparse vector and check respective set bits in dense one (AND)
  - OR on dense vectors expensive

Boolean AND

[0] 3 1 3 Total 7 bits

[1] 1 3 1 Total 5 bits

[0] 3 …. First 3 bits consumed

[1] 1 3

Remainder

[0] 1 1

[0] 0 …. [0] 4 ….

[0] 3

Remainder

[1] 1

[0] 3…..

[0] 7

Boolean OR

[0] 3 1 3 Total 7 bits

[1] 1 3 1 Total 5 bits

[1] 1 …. First 3 bits consumed

[0] 2 1 3

Remainder

[0] 3 1

[0] 2 …. [1] 1 2 ….

[1] 1 3

Remainder

[0] 1 1

[1] 1….. [1] 1 2 1

[0] 3

Remainder

[1] 1

[1] 1….. [1] 1 2 2

Delta-encoding

1234, 1236, 1240, 2000, 2011, 2015…...

1234, 2, 4, 760, 11, 4…...

Only very first integer requires 4 bytes. The following integers can be stored using 2 bytes.

Used in B+ tree clustered indexes

Can you use it in unclustered indexes?

Can you use it in hash-indexes?

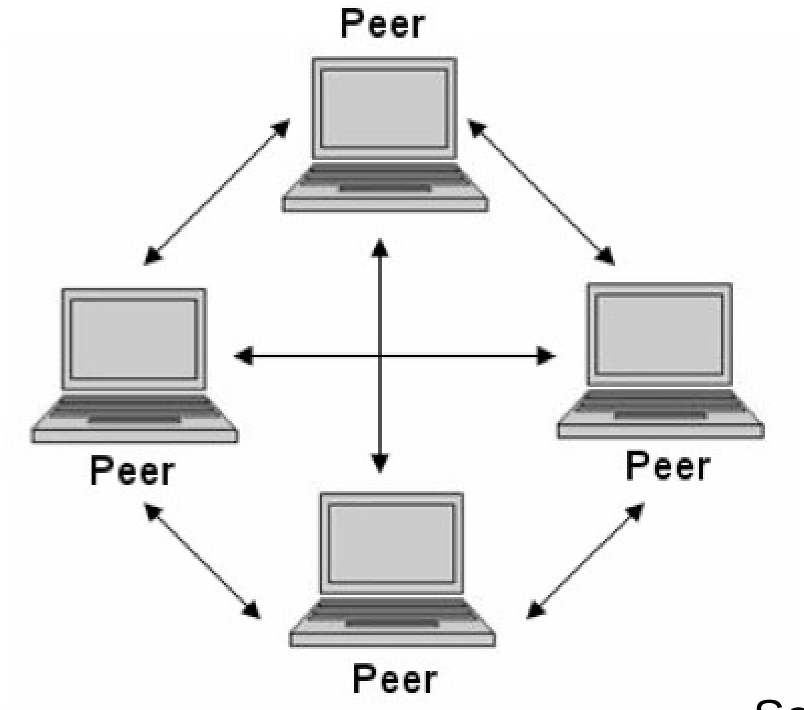
Delta-encoding

- Similar to the RLE encoding, keep a sliding window of 4-byte integer to slide over the sorted list computing the original int value on the fly.
- Gives about 50% storage space saving
  - To perform lesser I/O
  - Save disk space
  - Maintain same indexing performance (or better due to lesser I/O)

Distributed Storage

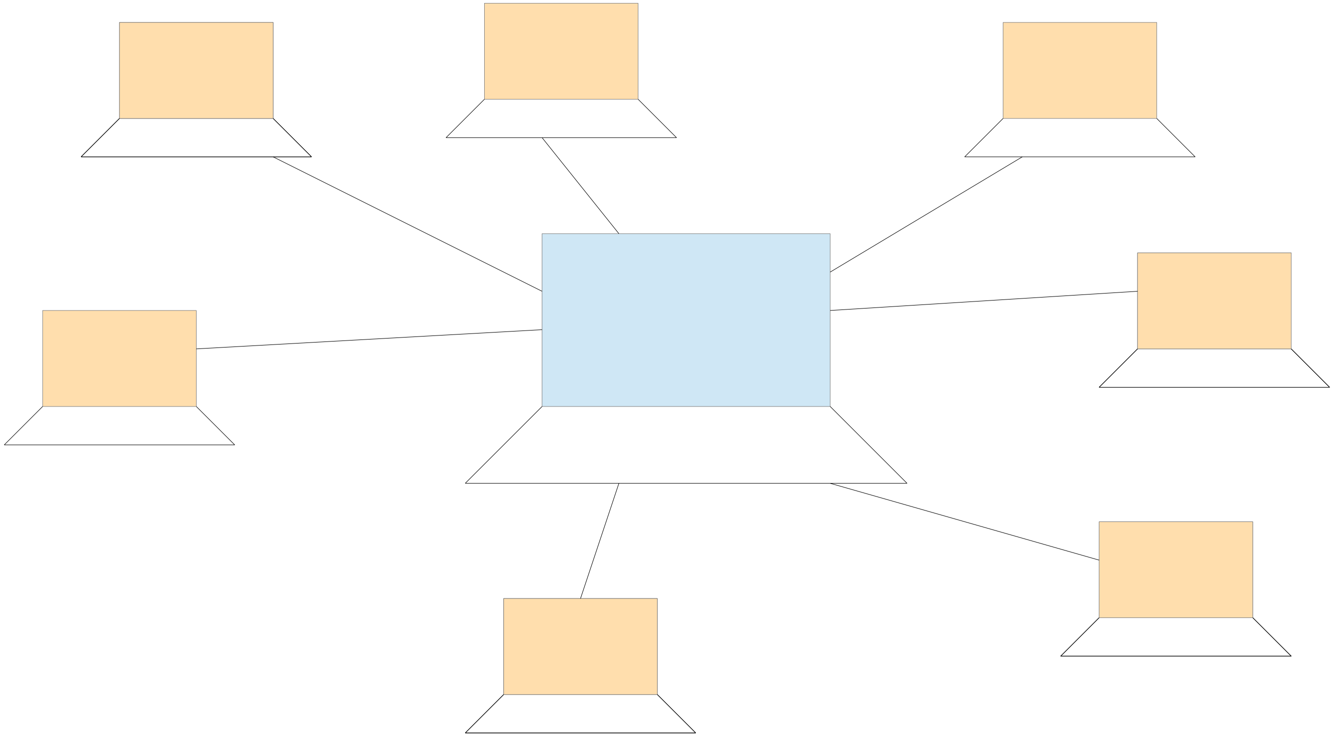
- Topologies (architecture):
  - Peer-to-peer – no hierarchy, shared nothing
    - Pastry, Chord, CAN systems
  - Hierarchical – one or more masters, several slaves
    - MapReduce based frameworks – Hadoop, SPARK, and many others

Peer-to-peer



Source: Google images

Hierarchical



Cloud

Source: Google images

P2P

- Every compute-node knows every other node.
- Every compute-node has a unique ID
- Data gets distributed by some *function* $f(d_i)$ applied on each data item d_i
- The output of the $f(d_i)$ compared with node-id
- Data item sent to node whose ID is closest to $f(d_i)$

P2P

- How to decide the *function*?
  - Simplest is "distribted hash table" (DHT)
  - Uses a *hash* function.
- Compute-node IDs generated using the same function, using IP address or MAC etc as the data to be hashed.
  - It can even be random ID generation and allocation
- Data can be anything, text, graphs, tables.
  - Data item to hash on changes as per data type!

P2P (graphs)

- Decide data-unit to hash
  - A node? <hash(vertex-label), adj vertices> (incoming or outgoing)?
  - An edge? <hash(s, p, o), s,p,o>
- How do you decide data-unit to hash?
  - Depends on the query types.
  - Why?

P2P (graphs)

- What do you join on?
  - Vertices or edges?
- So for distributed data what information do you need in one place?
  - Vertex labels?
  - Edge labels?
  - Why?