Two applications of maximum capacitated matchings in random bipartite graphs

Mathieu Leconte (Technicolor - INRIA)

Marc Lelarge (ENS - INRIA)

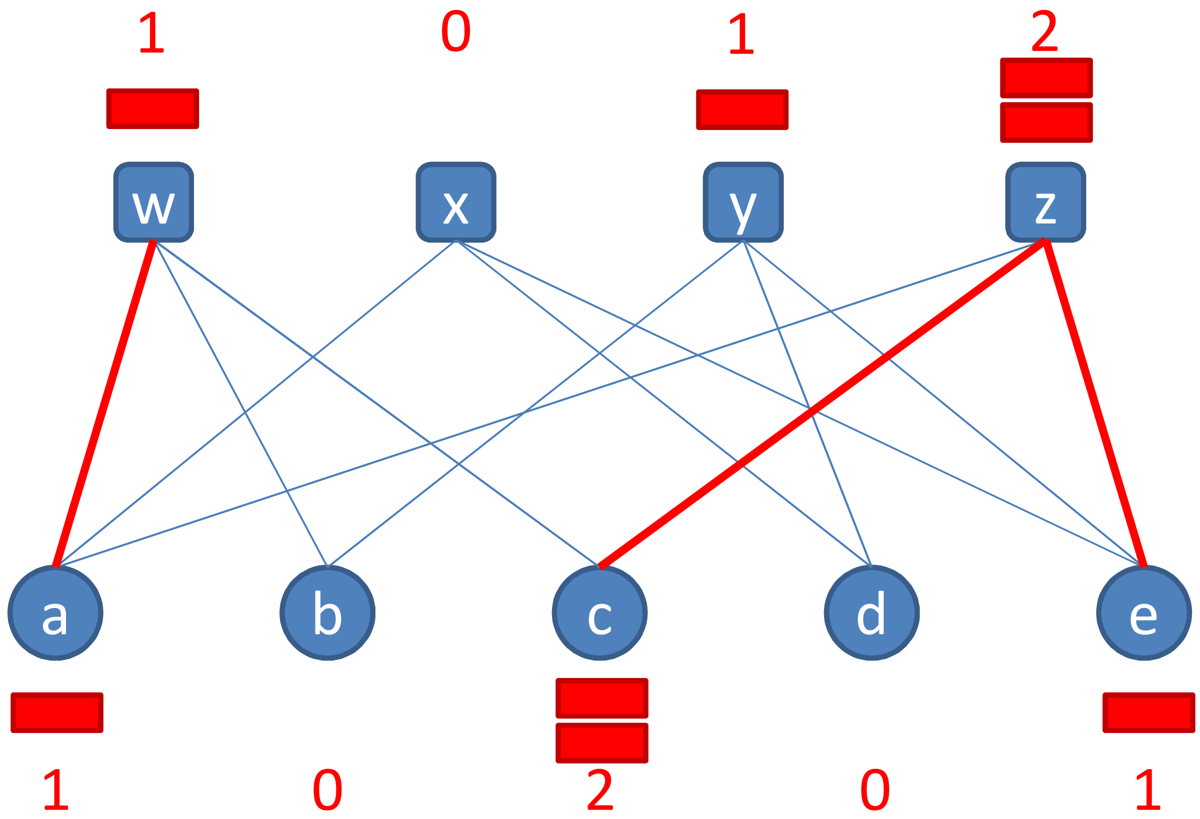
Laurent Massoulié (Microsoft – INRIA)

What is a capacitated matching?

- Bipartite graph with vertex- and edge-capacities
- Capacitated matching = subset of the edges that does not violate any capacity constraint

for simplicity, edge-capacity =1

ex: put weight 1 on the red edges



Outline

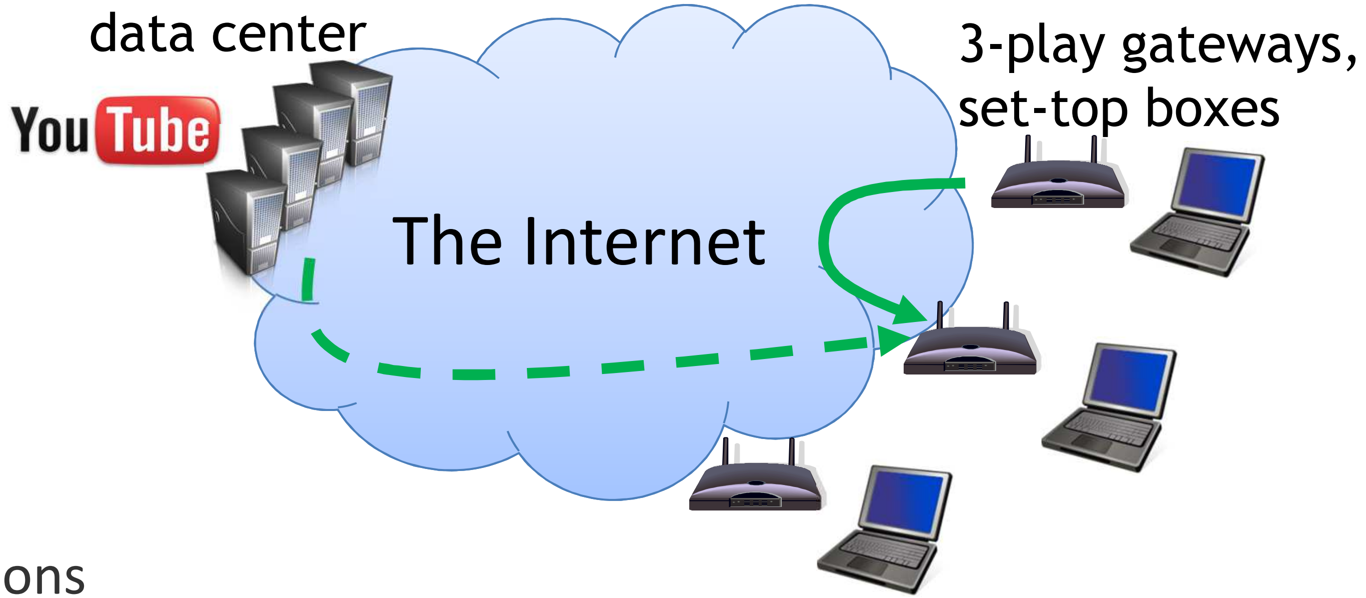
- Two motivating applications
  - Distributed CDN (content delivery networks)
  - Cuckoo hashing

- A brief overview of the techniques used
  - Message passing algorithms

DISTRIBUTED CDN

Current architecture: Internet content delivered from "cloud"

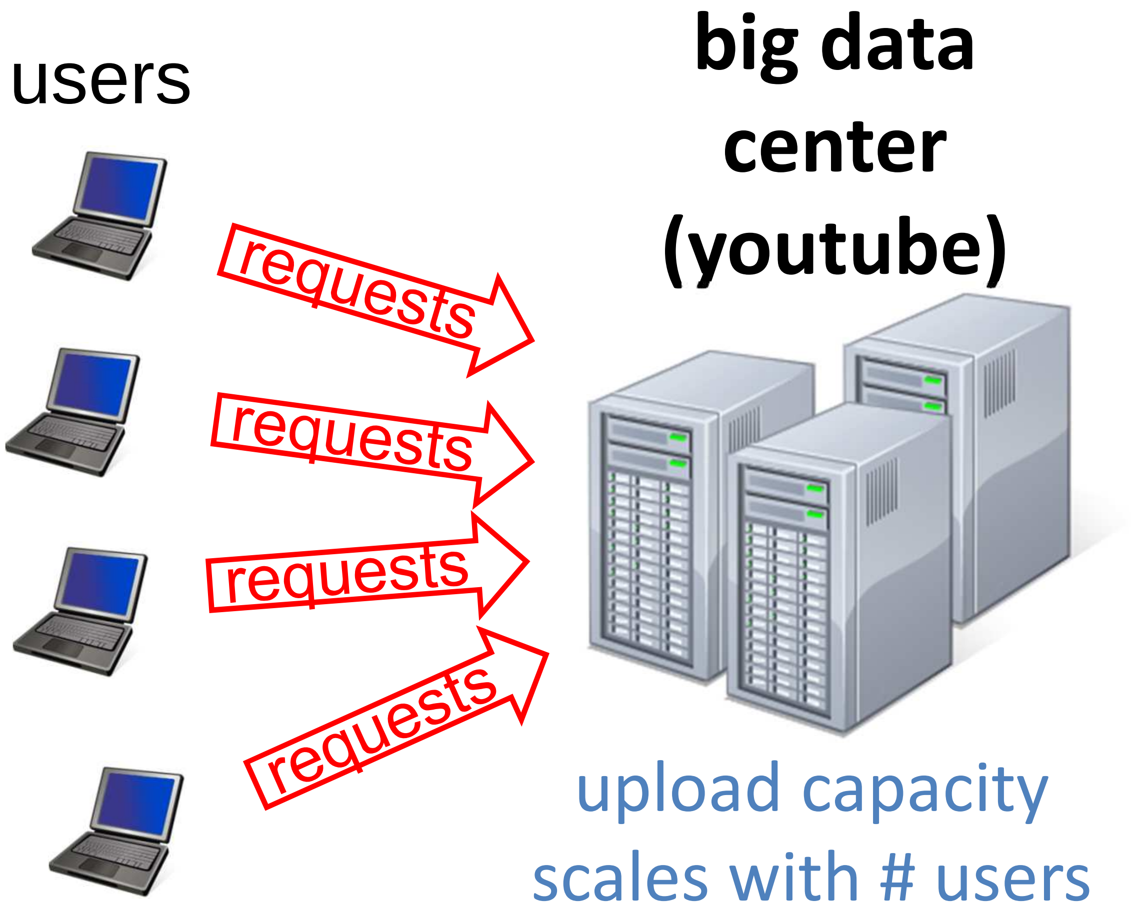
- ➢ Why not leverage bandwidth & memory resources at the network's edge?



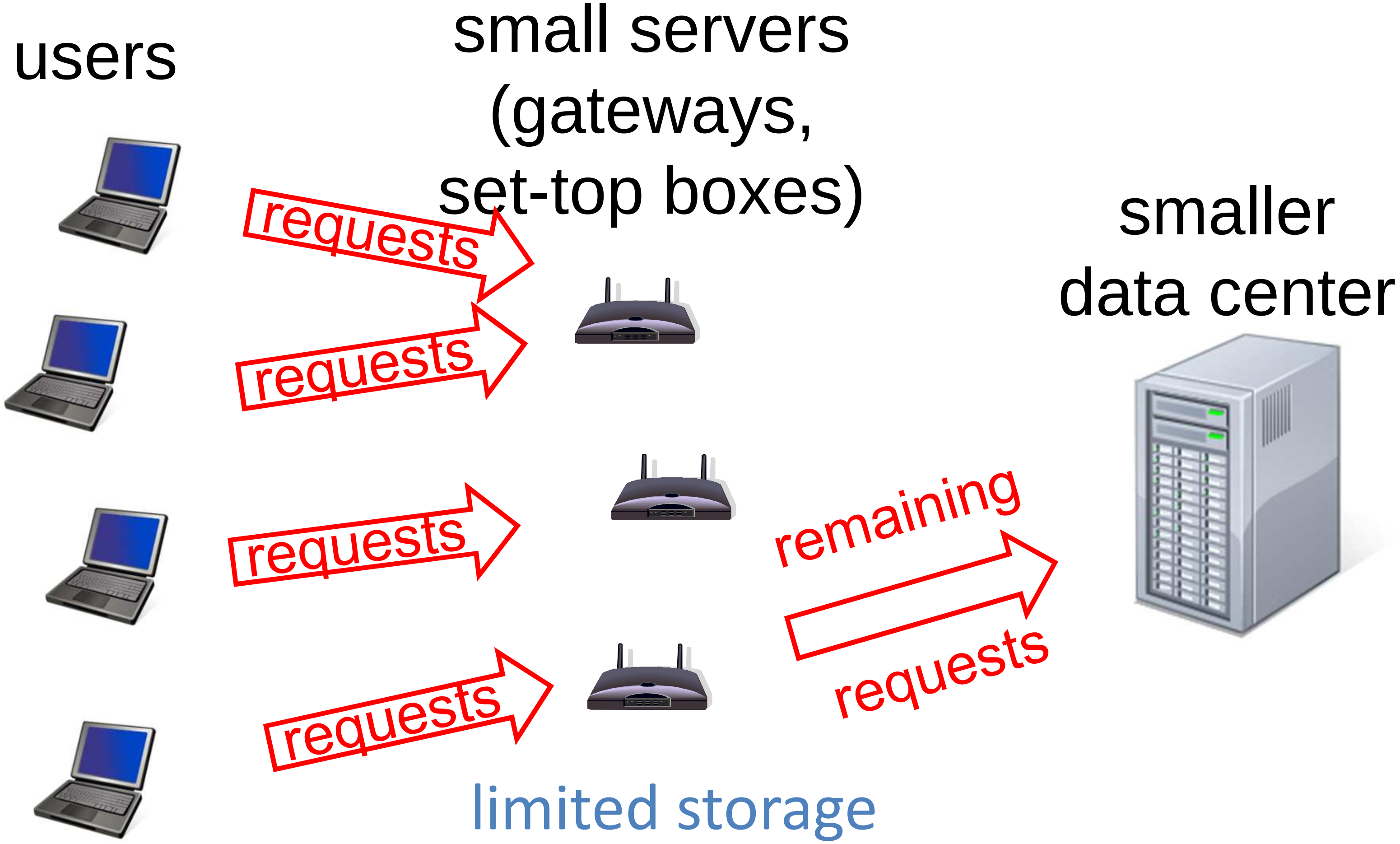
Questions

- What to store where, given heterogeneous content popularity
- how to match requests to servers
- resulting load reduction on data center

Moving from centralized CDNs…

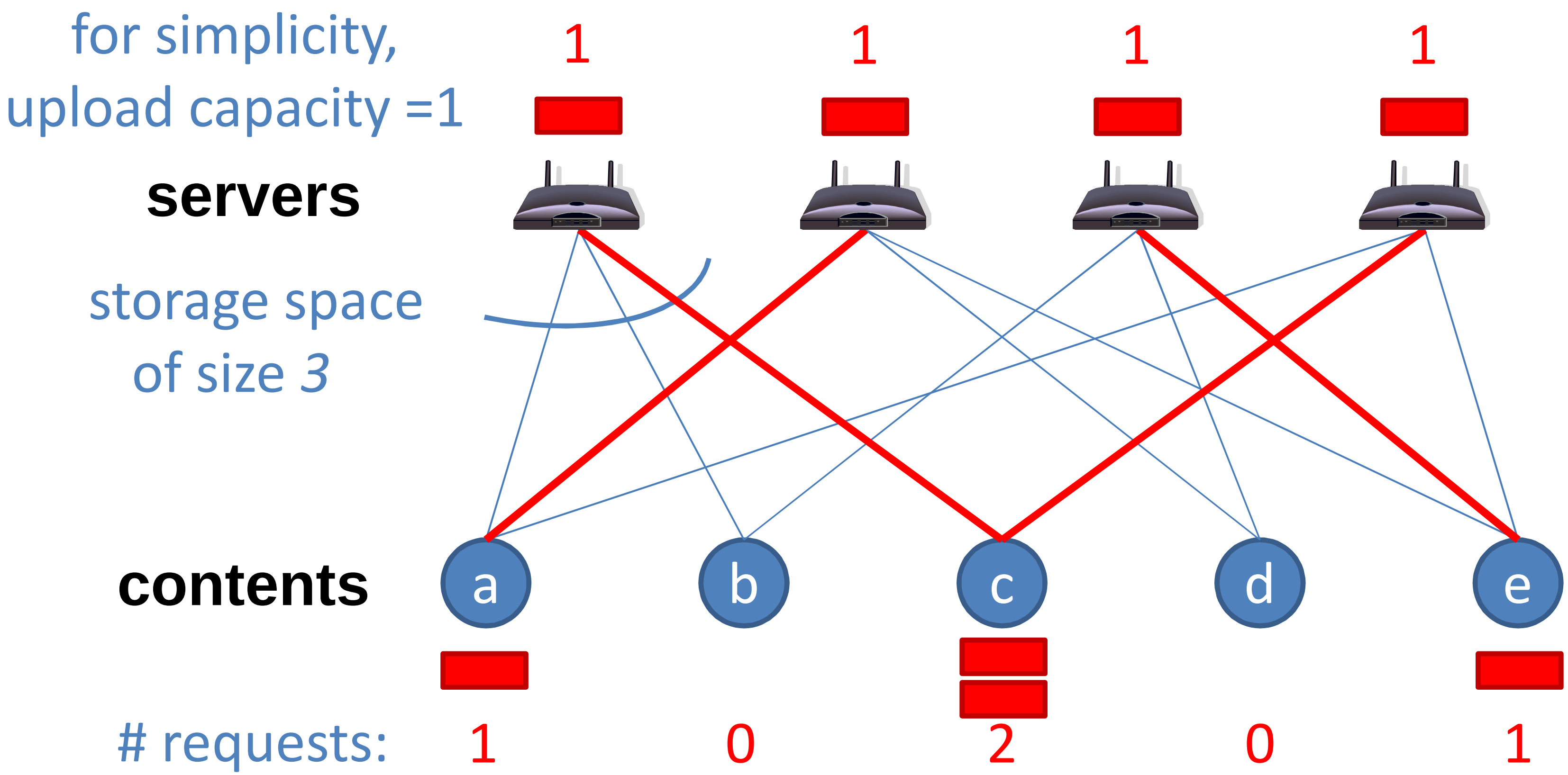


…to distributed CDNs



limited storage
& upload capacity,
# servers scales with # users (P2P-like)

Bipartite graph representation



- Edges between contents and servers storing them
- Allocation of requests to servers = capacitated matching
- Load reduction on data center = size of matching used

Many sources of randomness

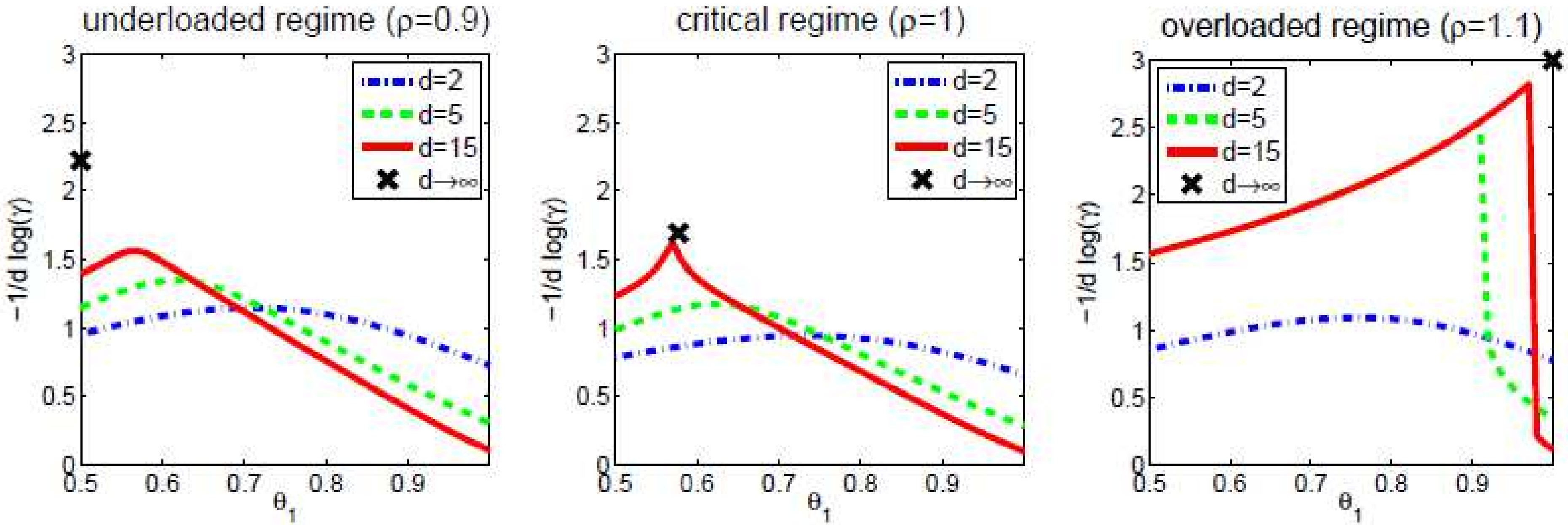
- # requests for each content is random
  - Only *a priori* popularity is known (ex: past measurements yield only estimates of # requests)
  - Capacity constraints are random
- Server caches constituted at random
  - We can only specify the total # replicas for each content; which server gets which replica is random
  - Edges of the graph are random
- Allocation policy may be random
  - Matching used may be random
  - Evacuate this: assume optimal matching at any time

What happens for large networks and large storage?

- In practice, very large networks (lots of contents/users/servers)
  - Our results: for given replication policy, we compute the size of maximum capacitated matching = load reduction on the data center
  - Allows to compare different replication policies

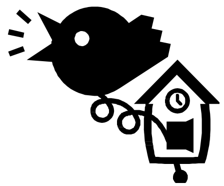
- Storage is cheap $\Rightarrow$ large storage asymptotic
  - Explicit expression for the optimal replication policy

Application to two classes of contents

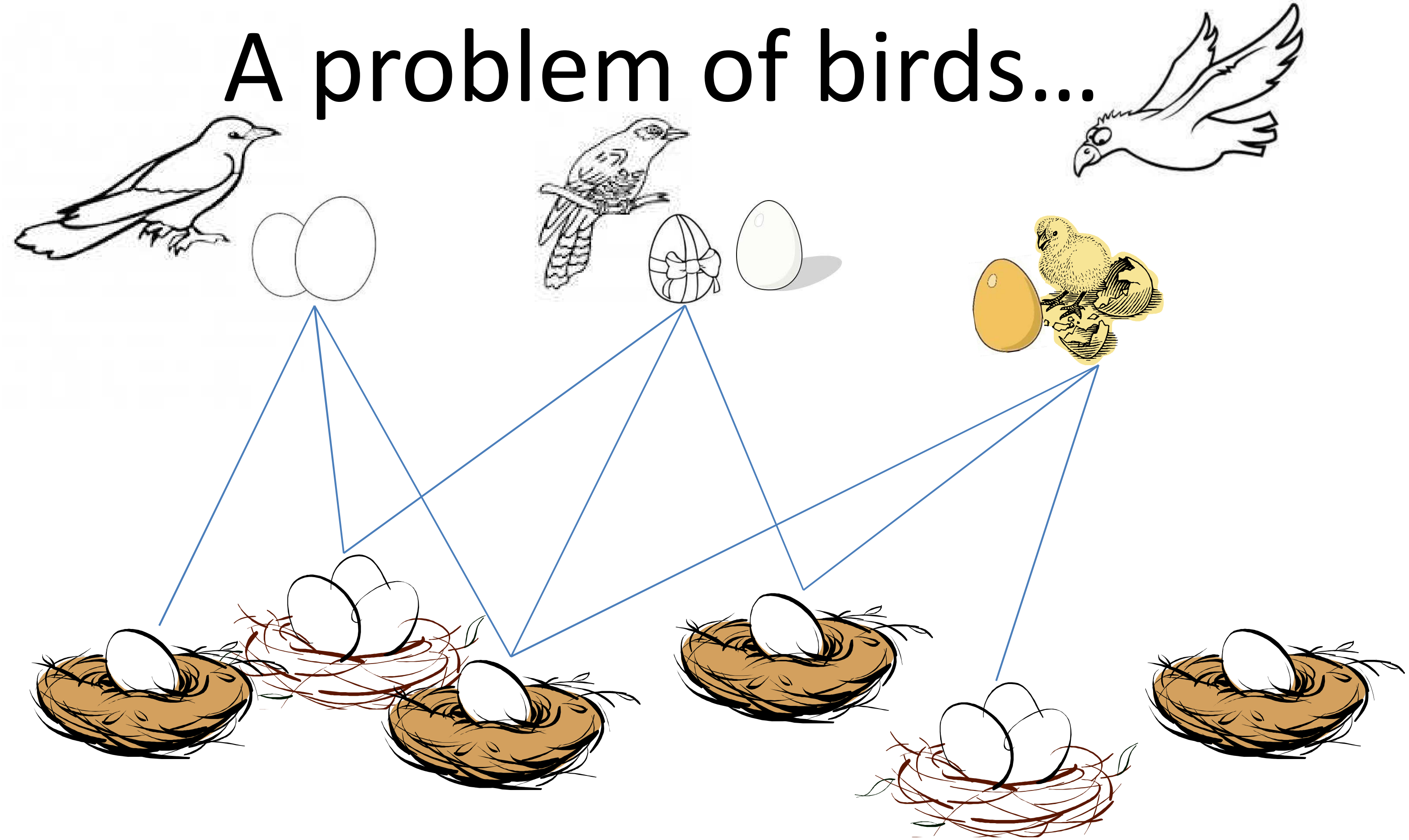


- Equal-sized classes; 1st class more popular than 2nd one
- Vertical axis = some measure of efficiency
  - indicates how fast inefficiency drops as storage capacity grows
- Horizontal axis spans replication policies
  - θ_1 = fraction of storage used for class 1

CUCKOO HASHING

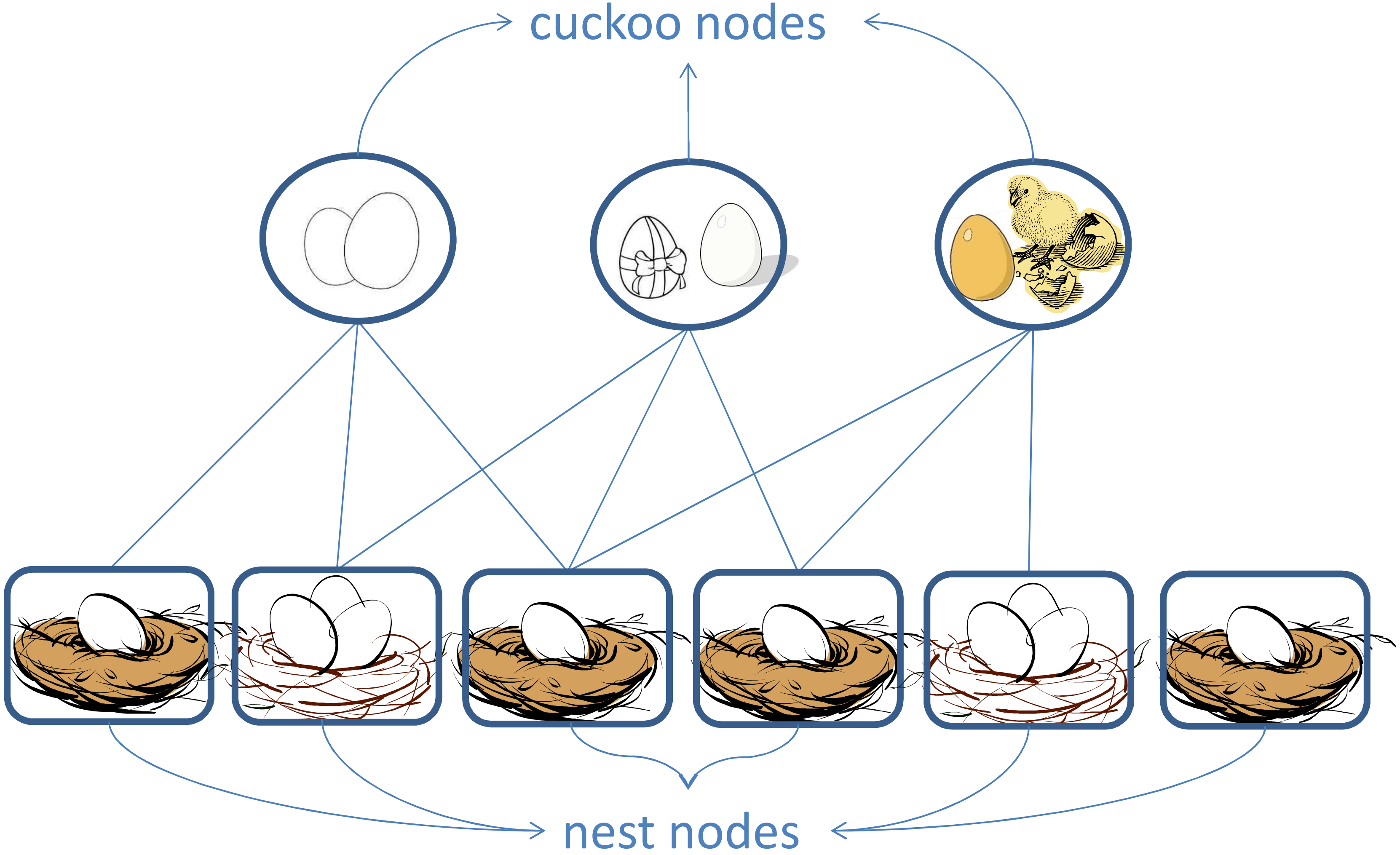


A problem of birds…



- The cuckoos want to lay their eggs in some other birds' nests
- However, birds can count: the number of eggs in each nest must remain constant (but kicking out non-cuckoo eggs is okay)
- Each cuckoo must replace eggs in *different* nests, else it will show
- They are lazy birds and only try ***3*** nests at random before giving up

Link with capacitated matchings



- Capacity of cuckoo nodes = # eggs of the cuckoo = 2, here
- Capacity of nest nodes = # eggs in the nest = 1 or 3, here

…and a problem of hash tables

- items = cuckoo eggs & keys = nests
  - Want to assign a key to each item, so as to be able to retrieve the items efficiently
- Multiple-choice hashing
  - Each item is given the choice between k random keys
- Cuckoo hashing
  - When the k keys are already assigned, re-assign one to new item and kick out old one, like a cuckoo would do!

- Question: how many items can we handle?
  - Threshold τ such that if # items < τ # keys, there exists an assignment with probability tending to 1 as size of system grows

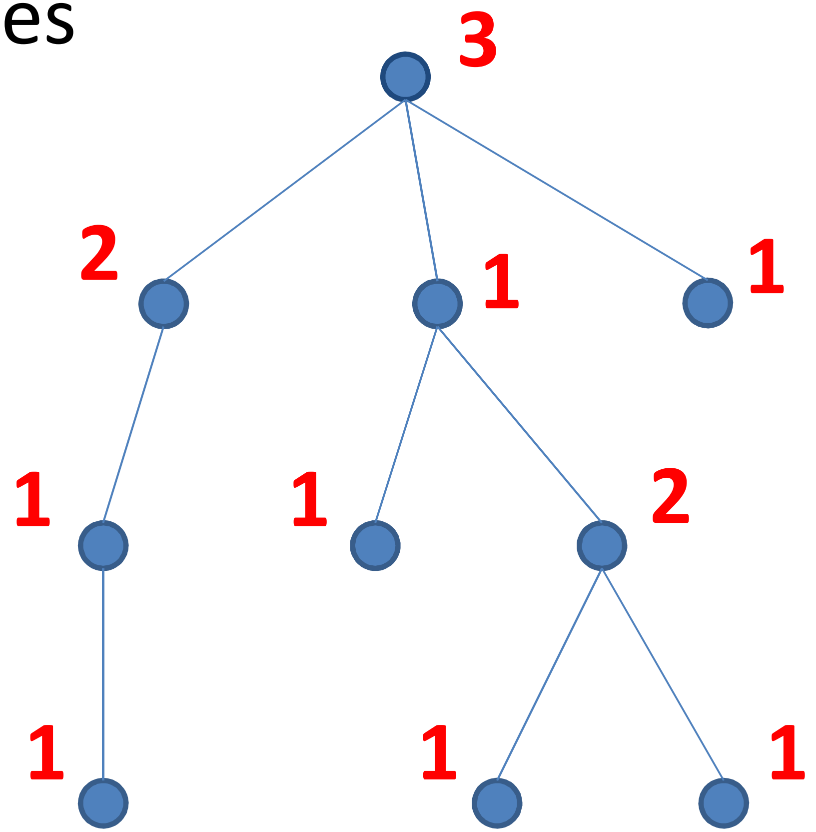
What happens for large bird populations / large hash tables?

- Equivalent to ask how many cuckoos can there be so that no cuckoo egg is lost
  - Size of maximum capacitated matching
    = # cuckoo eggs with new home
    = # items with a key successfully assigned
  - Our results: we compute the threshold τ under which no cuckoo egg lost & valid hash table
  - Allows dimensioning of hash tables
    & performance evaluation of cuckoo hashing

MESSAGE PASSING ALGORITHMS

How to compute the maximum weight of a capacitated matching in a tree?

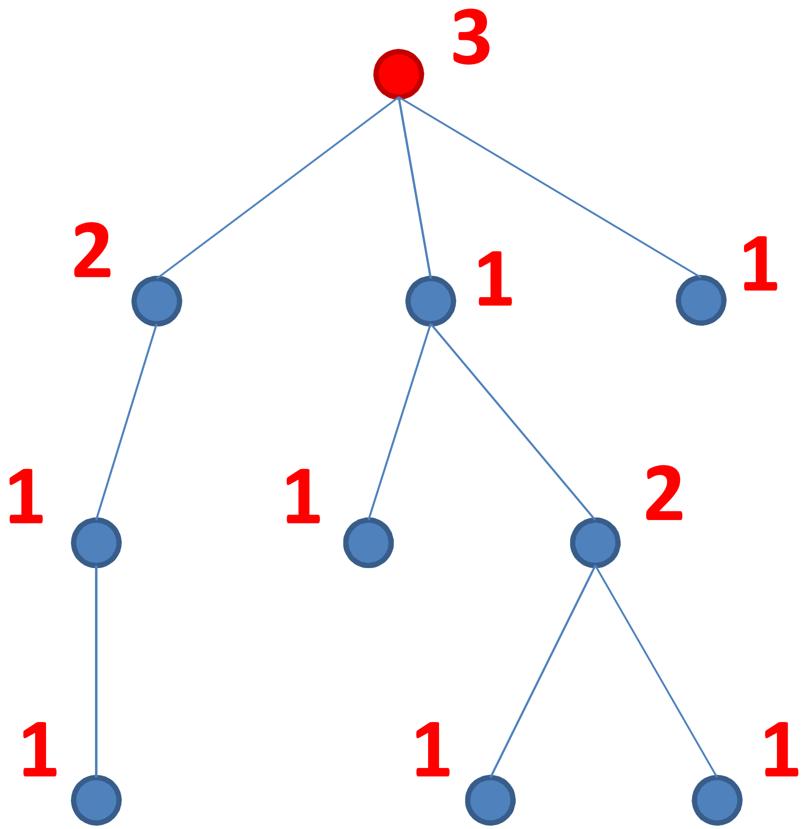
- The random graph we used asymptotically look like trees



for simplicity, edge-capacity =1

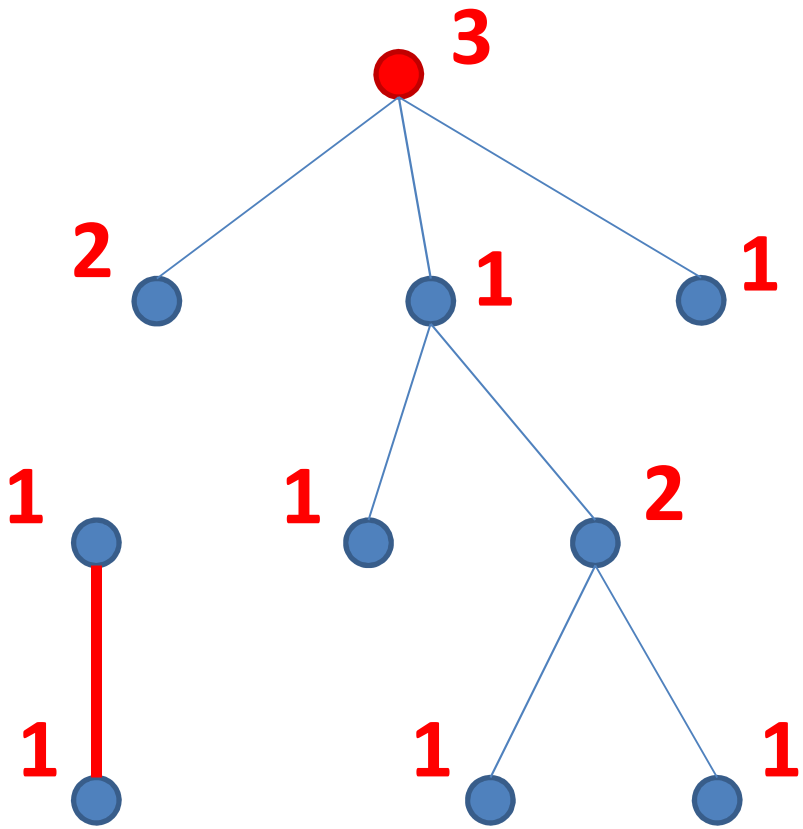
Greedy algorithm in finite rooted trees

- Choose a root



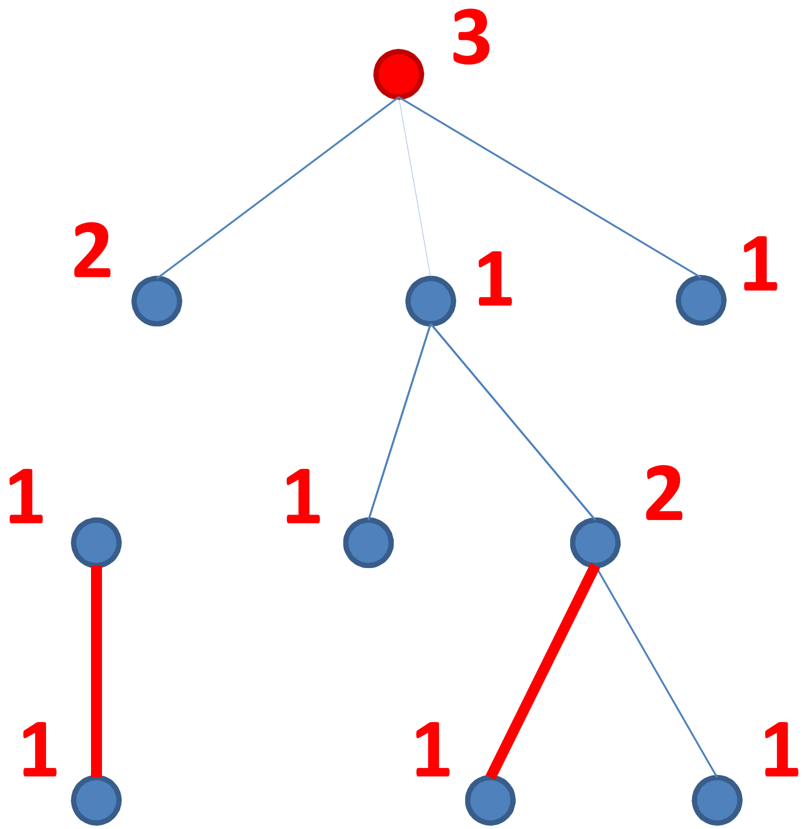
Greedy algorithm in finite rooted trees

- Iterately select dangling edges = leaf-removal



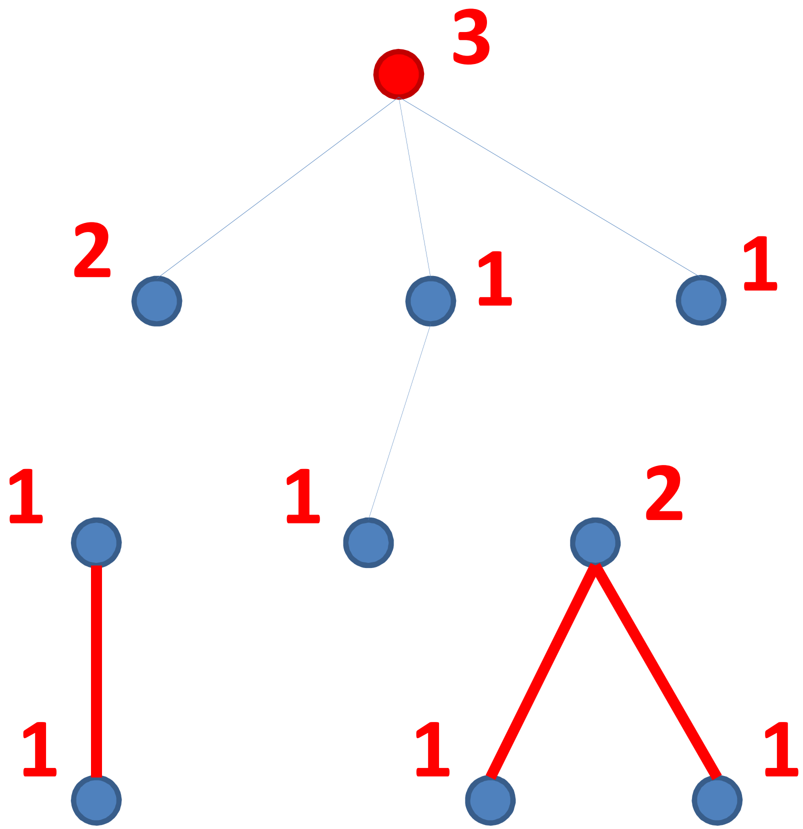
Greedy algorithm in finite rooted trees

- Leaf-removal



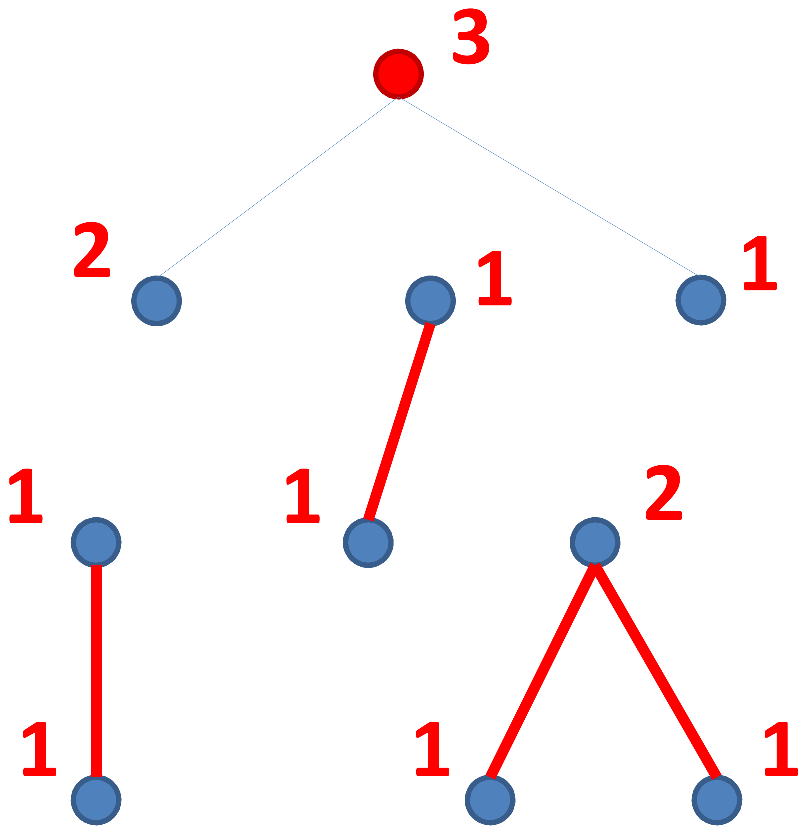
Greedy algorithm in finite rooted trees

- Leaf-removal



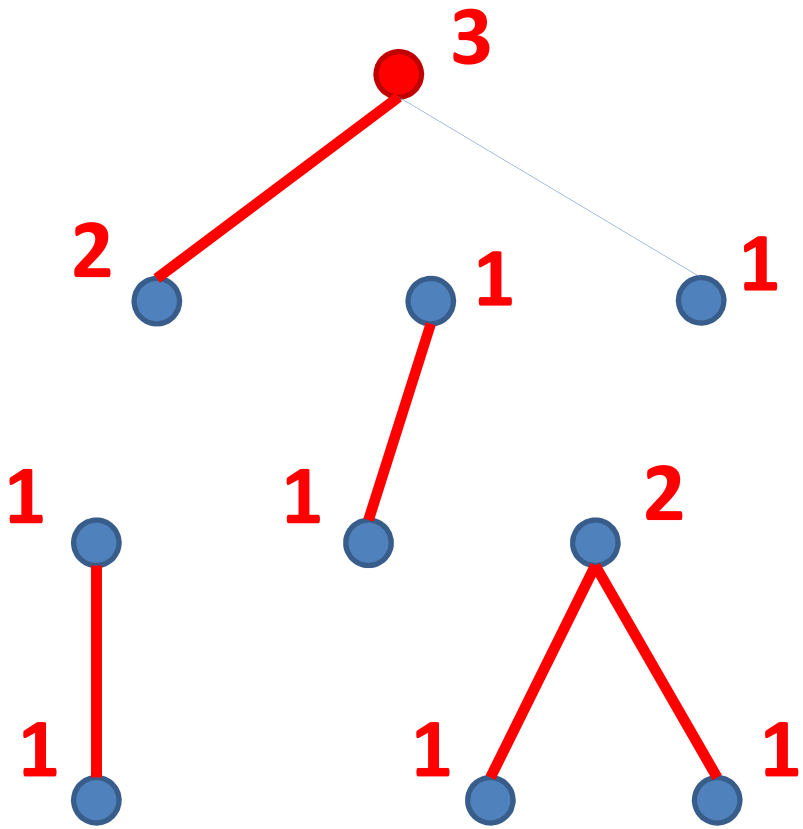
Greedy algorithm in finite rooted trees

- Leaf-removal



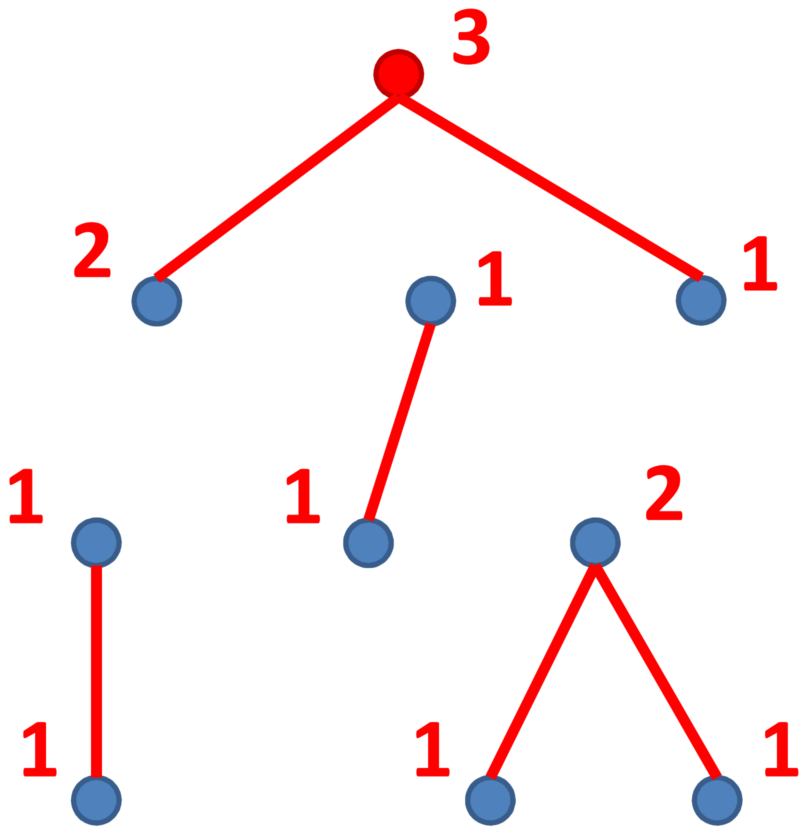
Greedy algorithm in finite rooted trees

- Leaf-removal



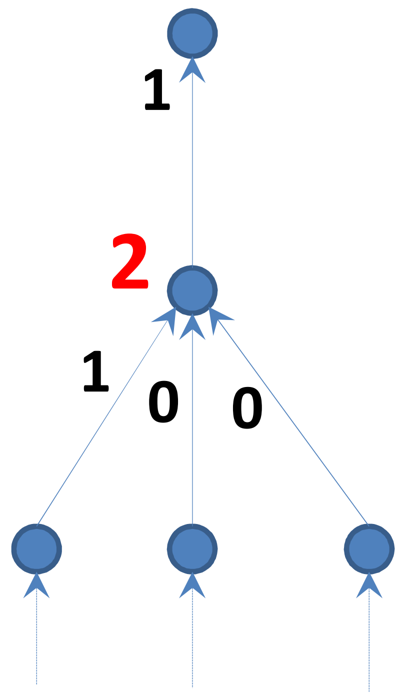
Greedy algorithm in finite rooted trees

- Maximum size = 6

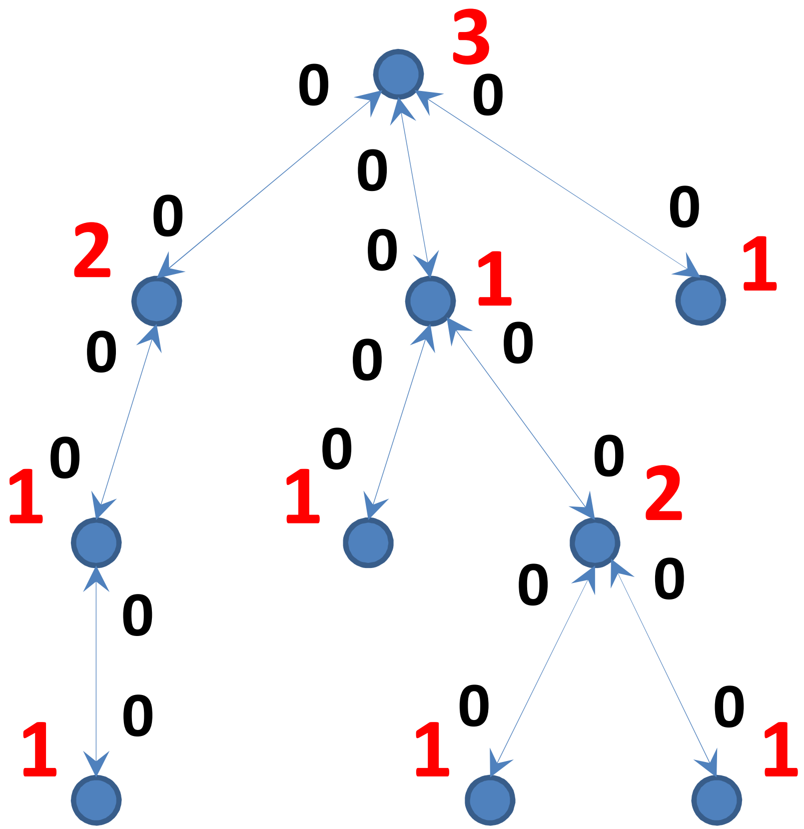


In the infinite limiting tree?

- Similar method, implemented through iterating local rules
  - Message passing over the directed edges of the graph
  - Message at iteration t indicates whether edge is required for maximum matching in the subtree below cut at depth t

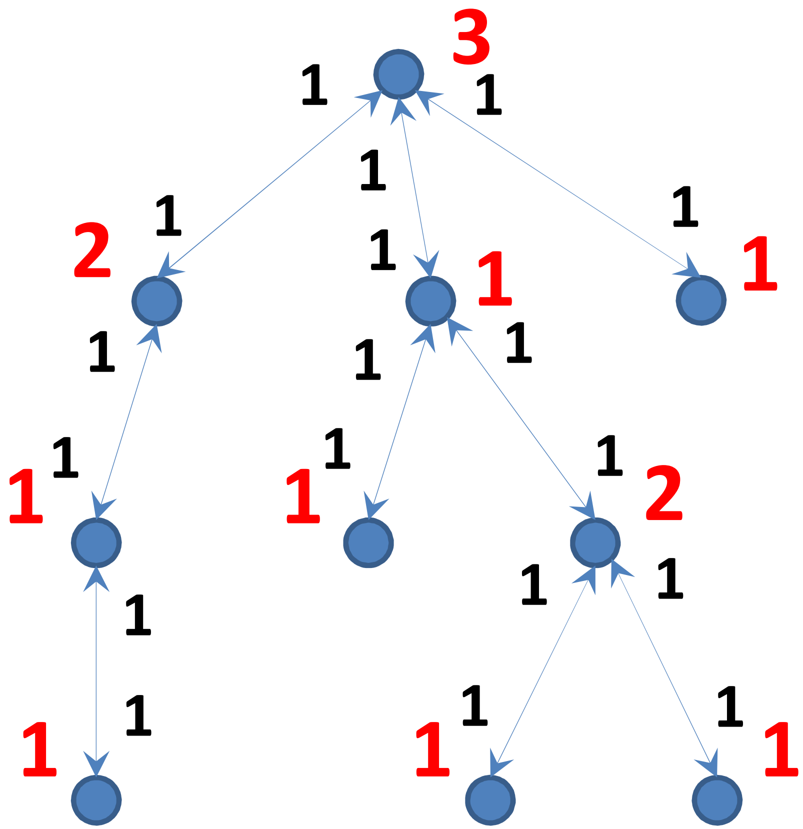


Message-passing in infinite trees



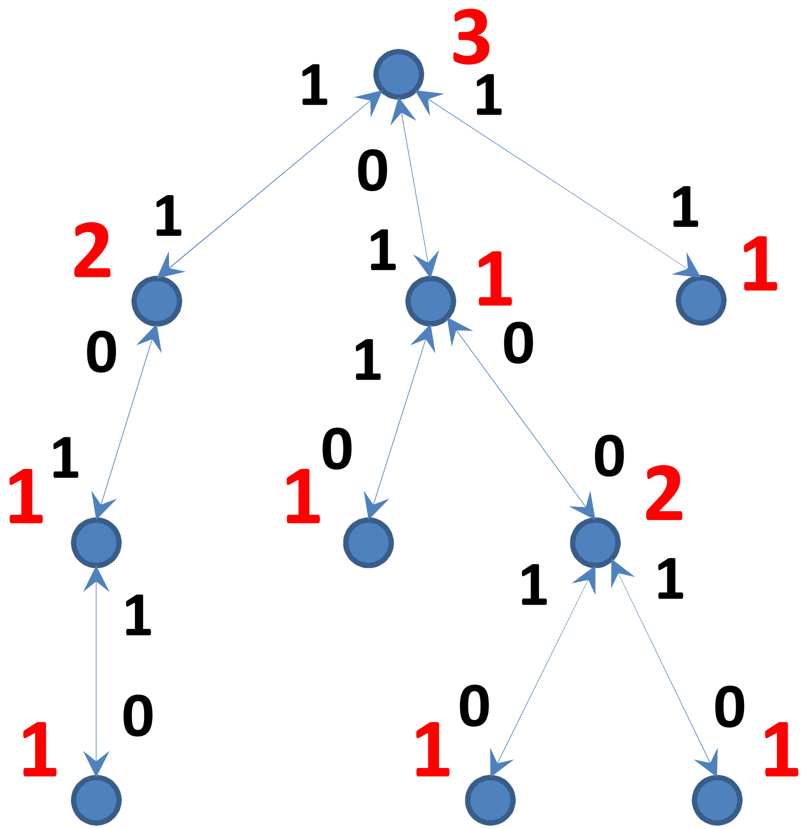
iteration 0

Message-passing in infinite trees



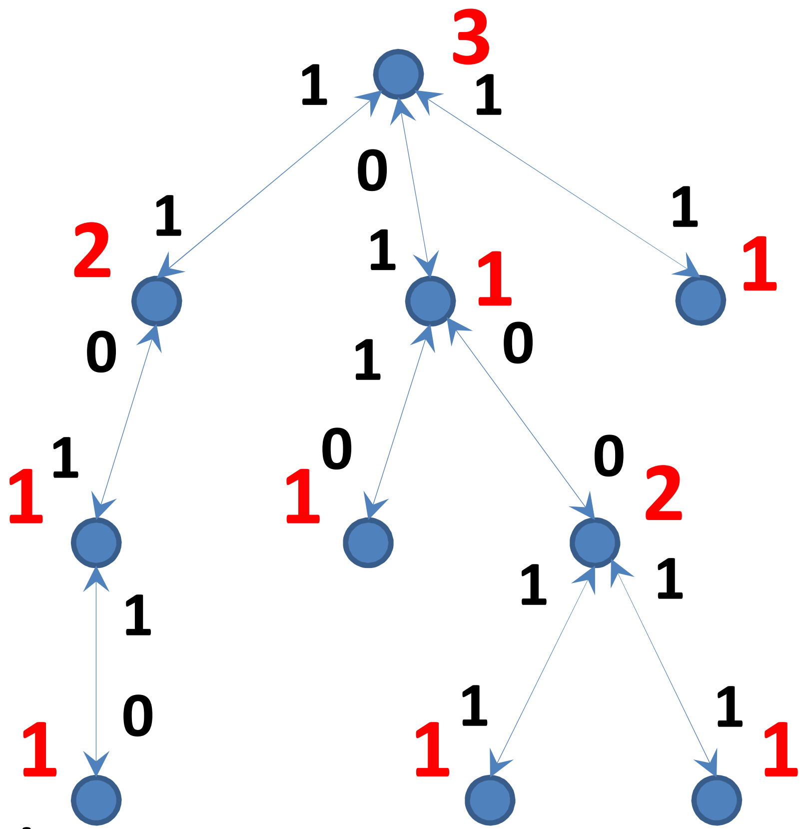
iteration 1

Message-passing in infinite trees



iteration 2

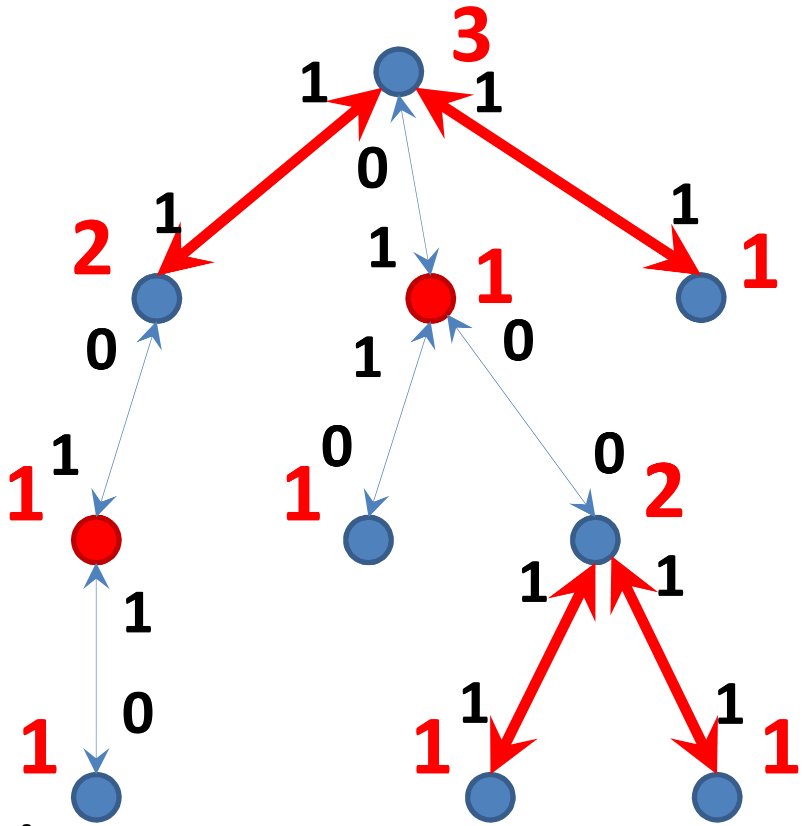
Message-passing in infinite trees



iteration 3 : **fixed-point**

Message-passing in infinite trees

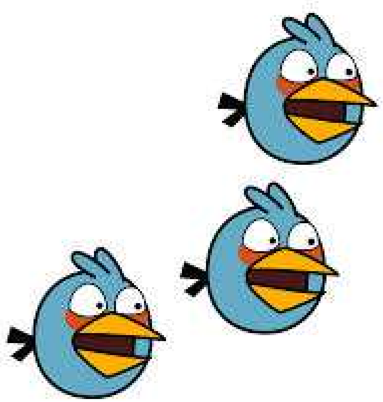
- From fixed-point, maximum size = 6



iteration 3 : **fixed-point**

Conclusion

- Compute the size of maximum capacitated matchings in random graphs
- Yields performance evaluation of large distributed CDNs
  - ➢ Optimization of their organization and dimensioning of residual data center
- Compute cuckoo hashing thresholds
  - ➢ Dimensioning of hash tables
  - ➢ Understand more of the life of cuckoos
- Message passing techniques (borrowed from statistical physics)



Thank you!!!