Humboldt University

Computer Science Department
Systems Architecture Group
http://sar.informatik.hu-berlin.de

Peer-to-Peer Systems

SoSe 2011

Introduction and Overview

What is P2P?

pastry can jxta fiorana

napster united devices freenet

open cola

aim ocean store

?

netmeeting

gnutella farsite

icq ebay

morpheus

limewire seti@home

uddi bearshare

grove

jabber popular power

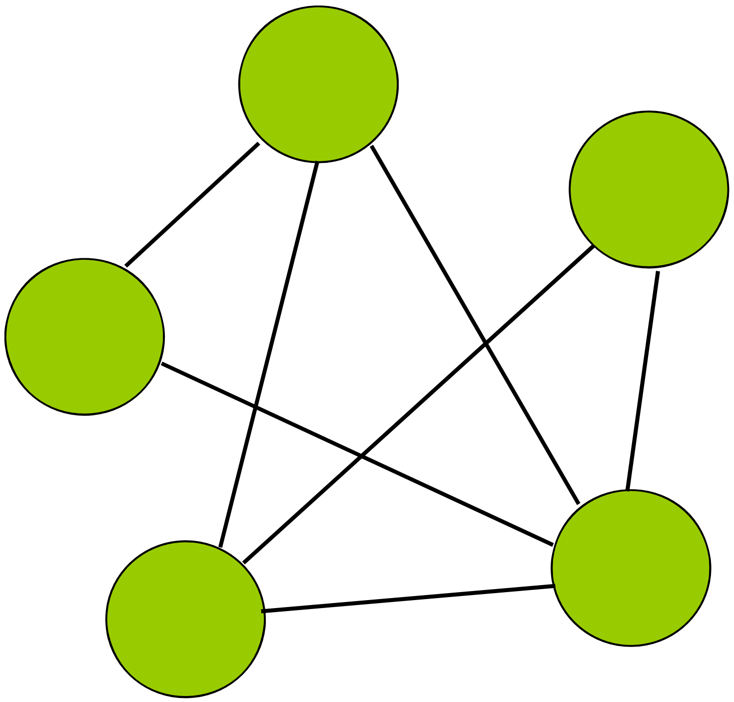
kazaa folding@home

tapestry

process tree chord mojo nation

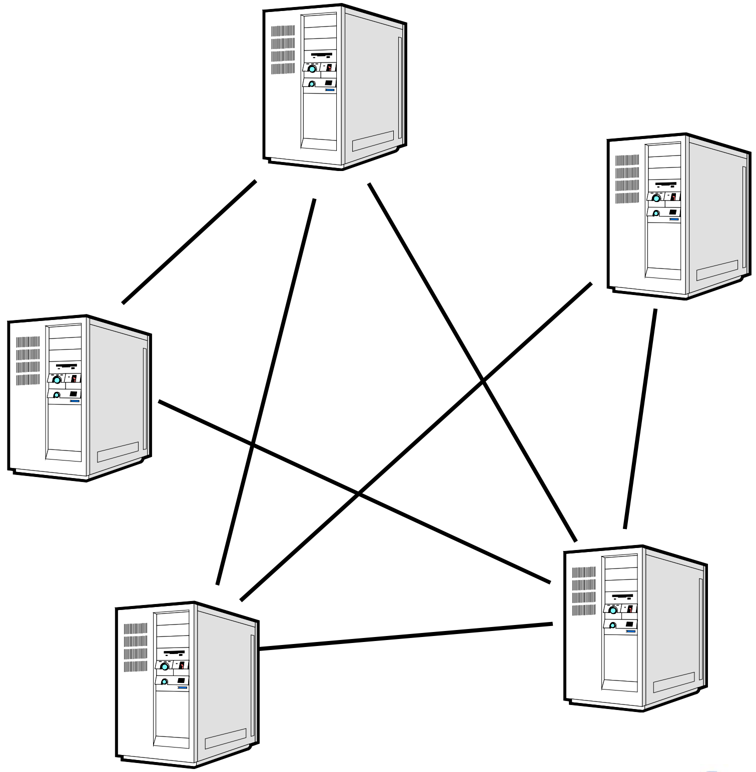
Peer-to-Peer Systems

- **Distributed application where nodes are:**
  - Autonomous
  - Very loosely coupled
  - Equal in role or functionality
  - Share and exchange resources with each other



Peer-to-Peer Systems

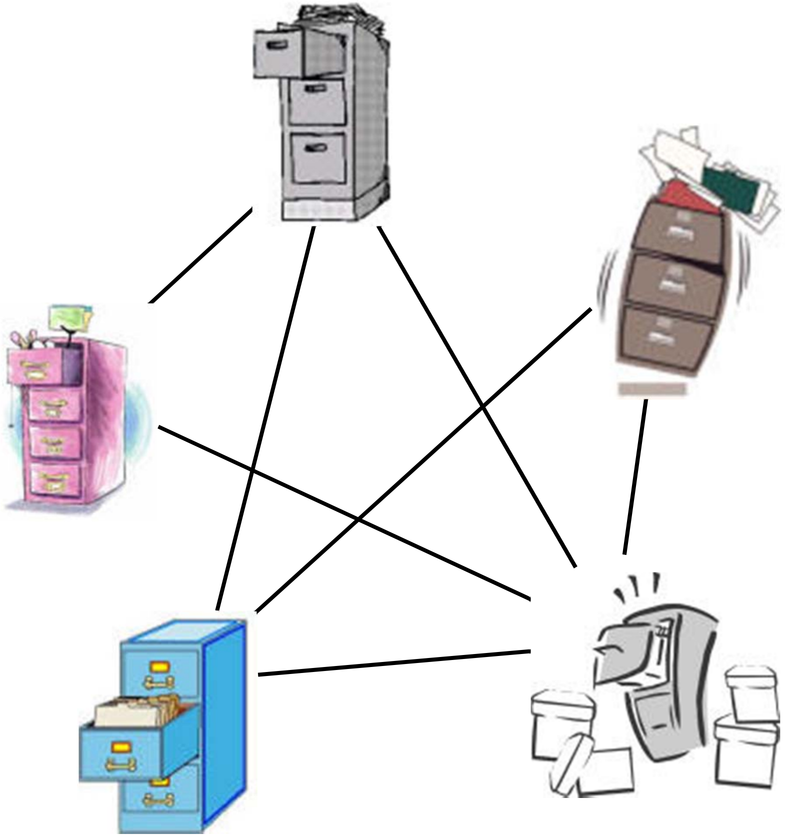
- **Distributed application where nodes are:**
  - Autonomous
  - Very loosely coupled
  - Equal in role or functionality
  - Share and exchange resources
- **Grid Computing**



Peer-to-Peer Systems

- **Distributed application where nodes are:**
  - Autonomous
  - Very loosely coupled
  - Equal in role or functionality
  - Share and exchange resources

- **Grid Computing**
- **File-sharing**

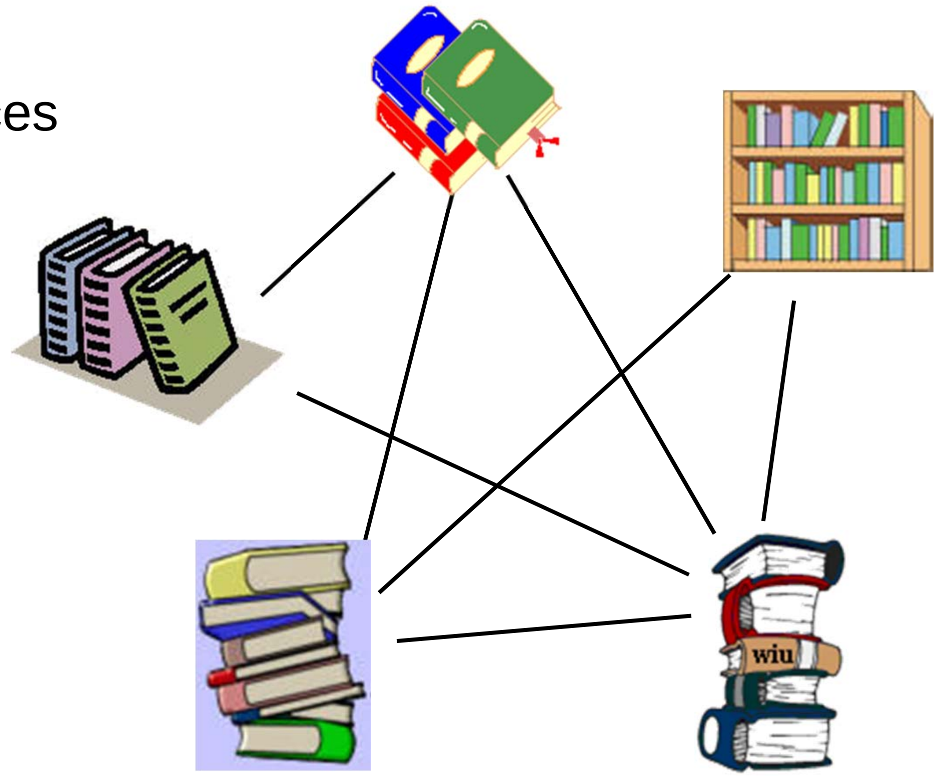


Humboldt Universität Informatik
Systems Architecture Group
http://sar.informatik.hu-berlin.de

Peer-to-Peer Systems

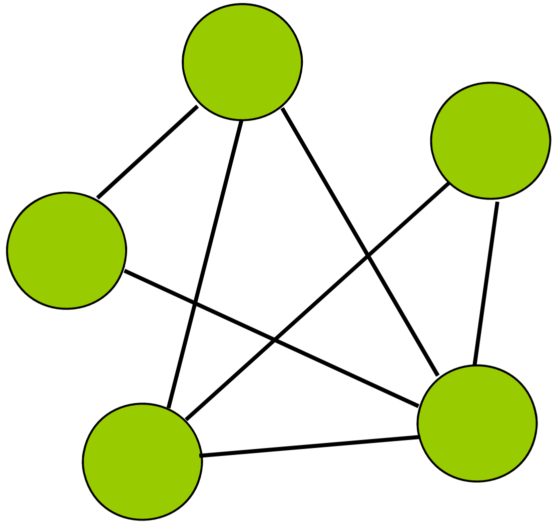
- **Distributed application where nodes are:**
  - Autonomous
  - Very loosely coupled
  - Equal in role or functionality
  - Share and exchange resources

- **Grid Computing**
- **File-sharing**
- **Digital Libraries/ Archive**



Is this new?

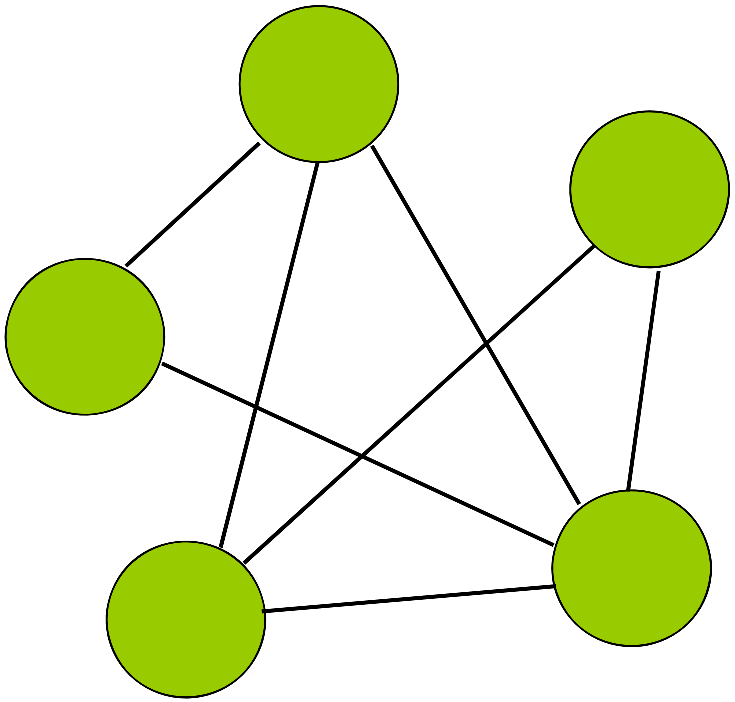
- Past Instances:
  - IP routing (1970's)
  - Distributed Databases!



- **Implicit Assumptions**
  - Scale: millions (billions?) of peers
  - Nature of peers: Weak (PCs, sensors, PDAs)
  - Application: lightweight semantics (e.g., file-sharing)

Benefits

- Pool together and harness (latent) resources at large scale
  - Petabytes of storage
  - > 72 TeraFLOPs (Seti@home)
- Consolidating resources across autonomous nodes
- Robust, self-organizing, self-healing



Humboldt Universität Informatik
Systems Architecture Group
http://sar.informatik.hu-berlin.de

P2P key challenges

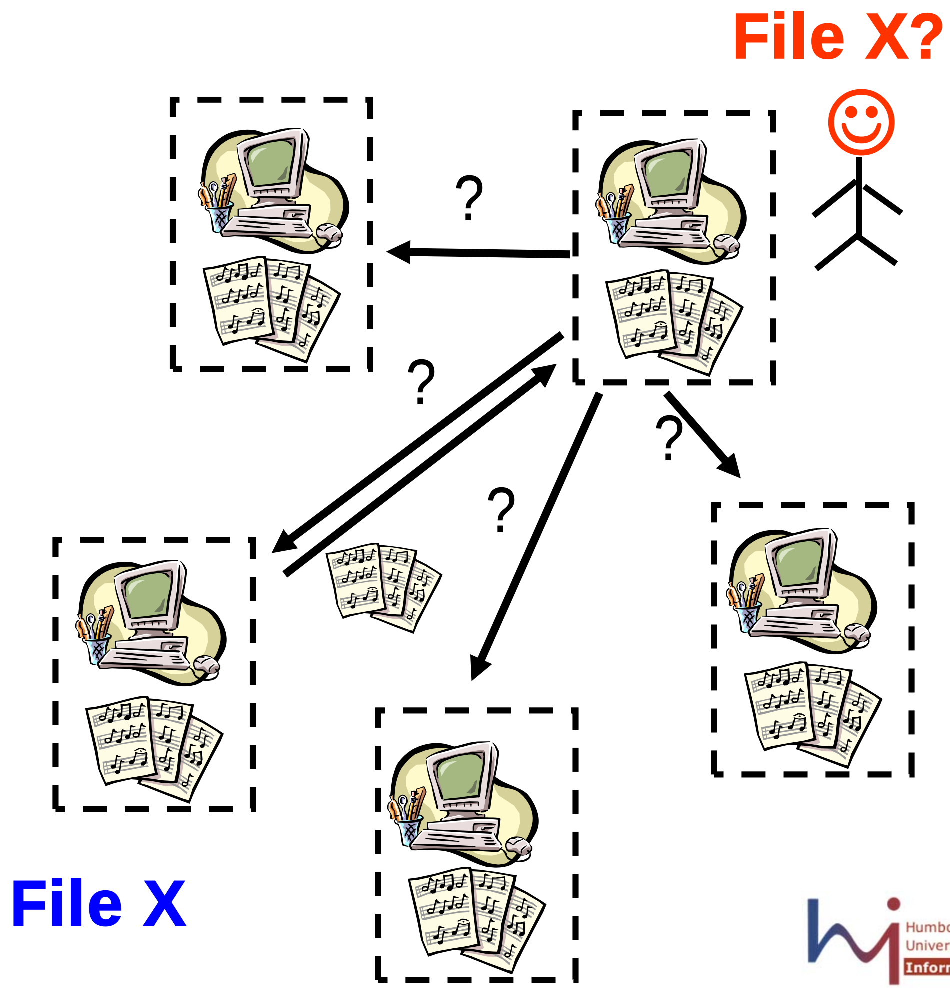
- What are they?

Illustrate with an example…

Humboldt Universität Informatik
Systems Architecture Group
http://sar.informatik.hu-berlin.de

Example: file sharing

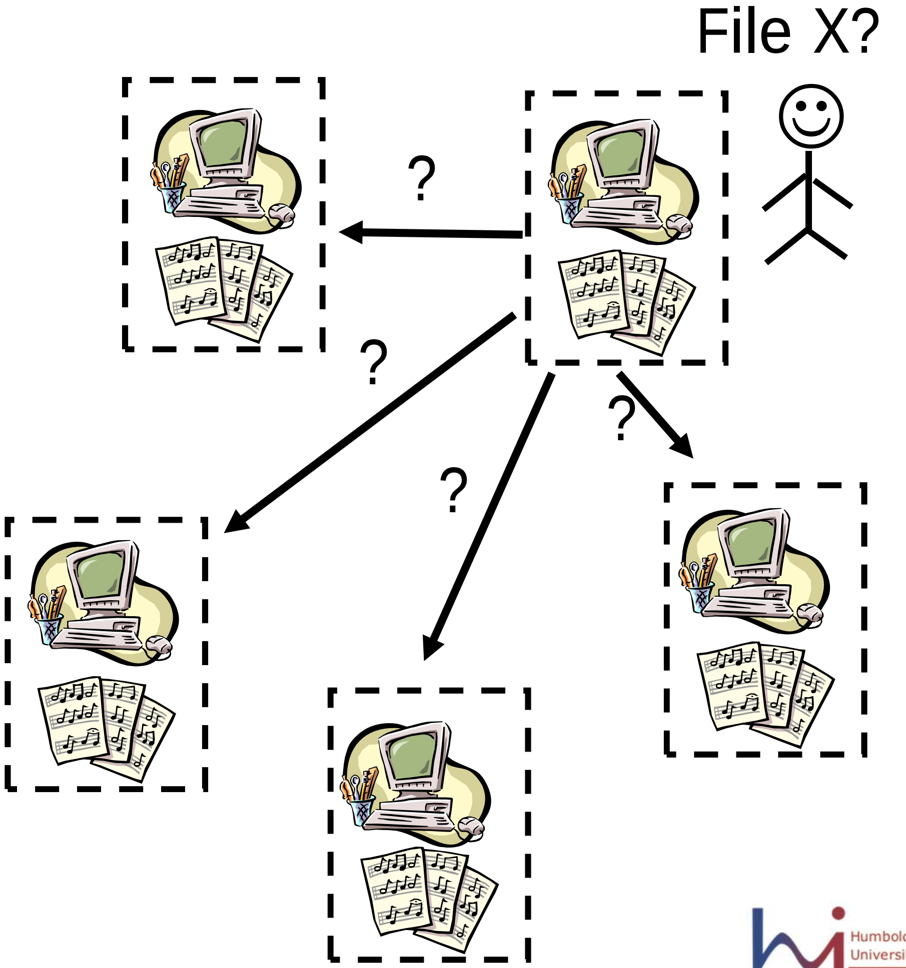
- Every peer stores and shares files
- How do I find **File X** ?



Humboldt Universität Informatik
Systems Architecture Group
http://sar.informatik.hu-berlin.de

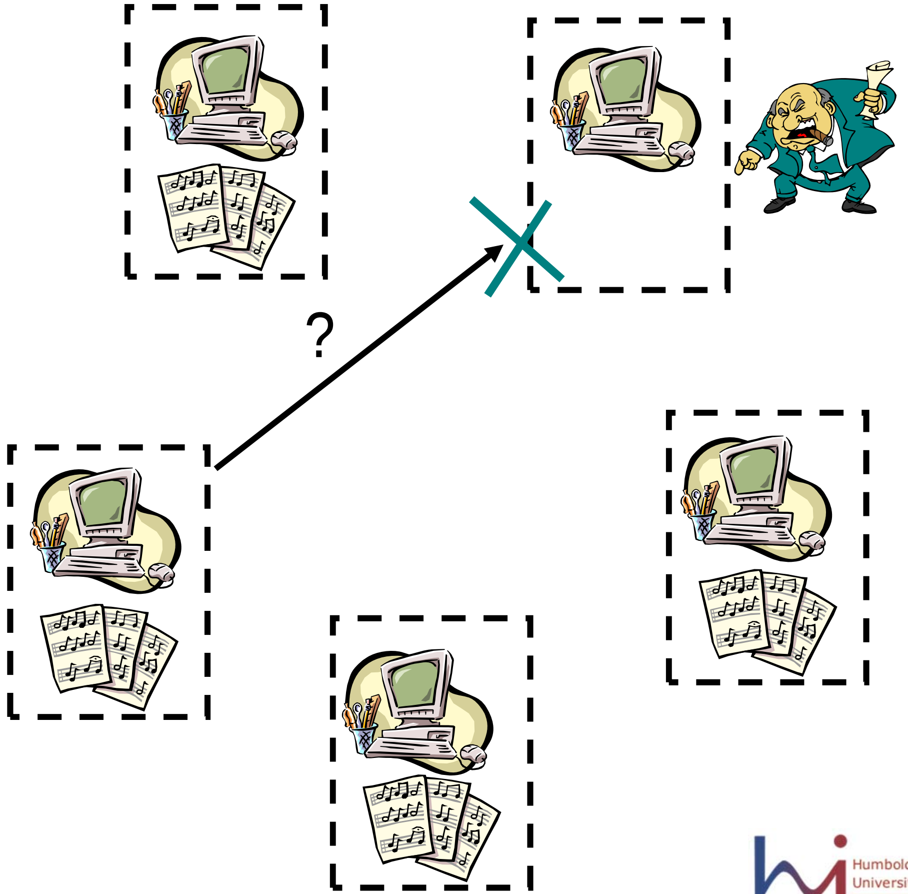
Example: file sharing

- Challenge #1: **Performance**
  - Asking everyone is *expensive!*
  - If I am smart,
    I only need to ask one peer



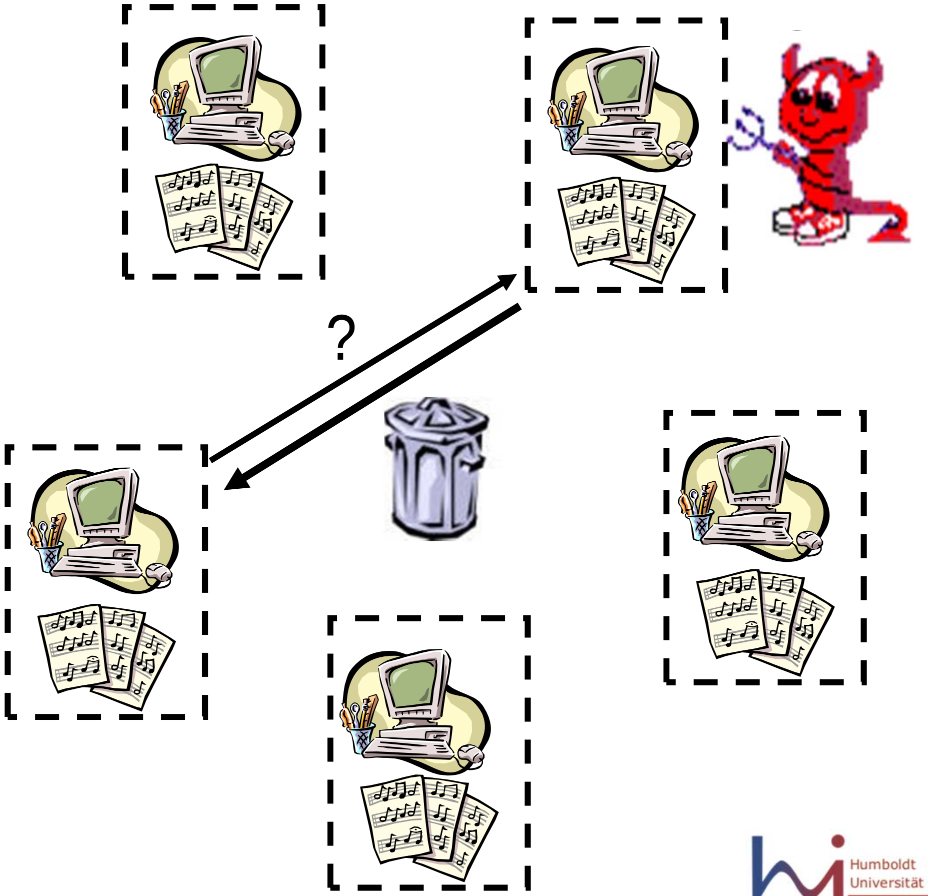
Example: file sharing

- Challenge #2: **Participation**
  - What if I do not want to store my share of the files?
  - *"Free-riding"* problem
  - How do we prevent **selfish** people from cheating?



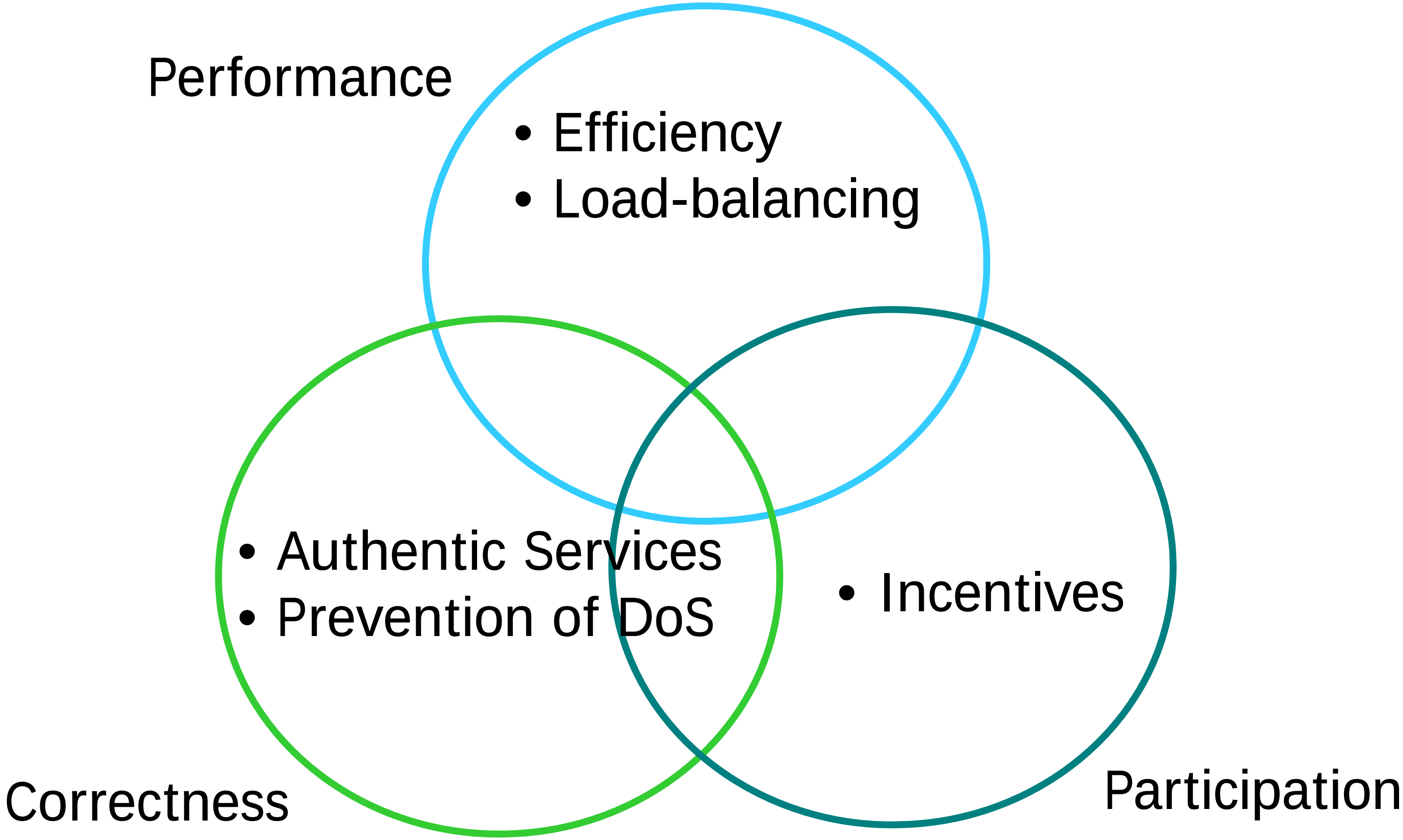
Example: file sharing

- Challenge #3: **Correctness**
  - What if I share a corrupted file?
  - How do we prevent **malicious** people from hurting others?



Humboldt Universität Informatik
Systems Architecture Group
http://sar.informatik.hu-berlin.de

Challenges



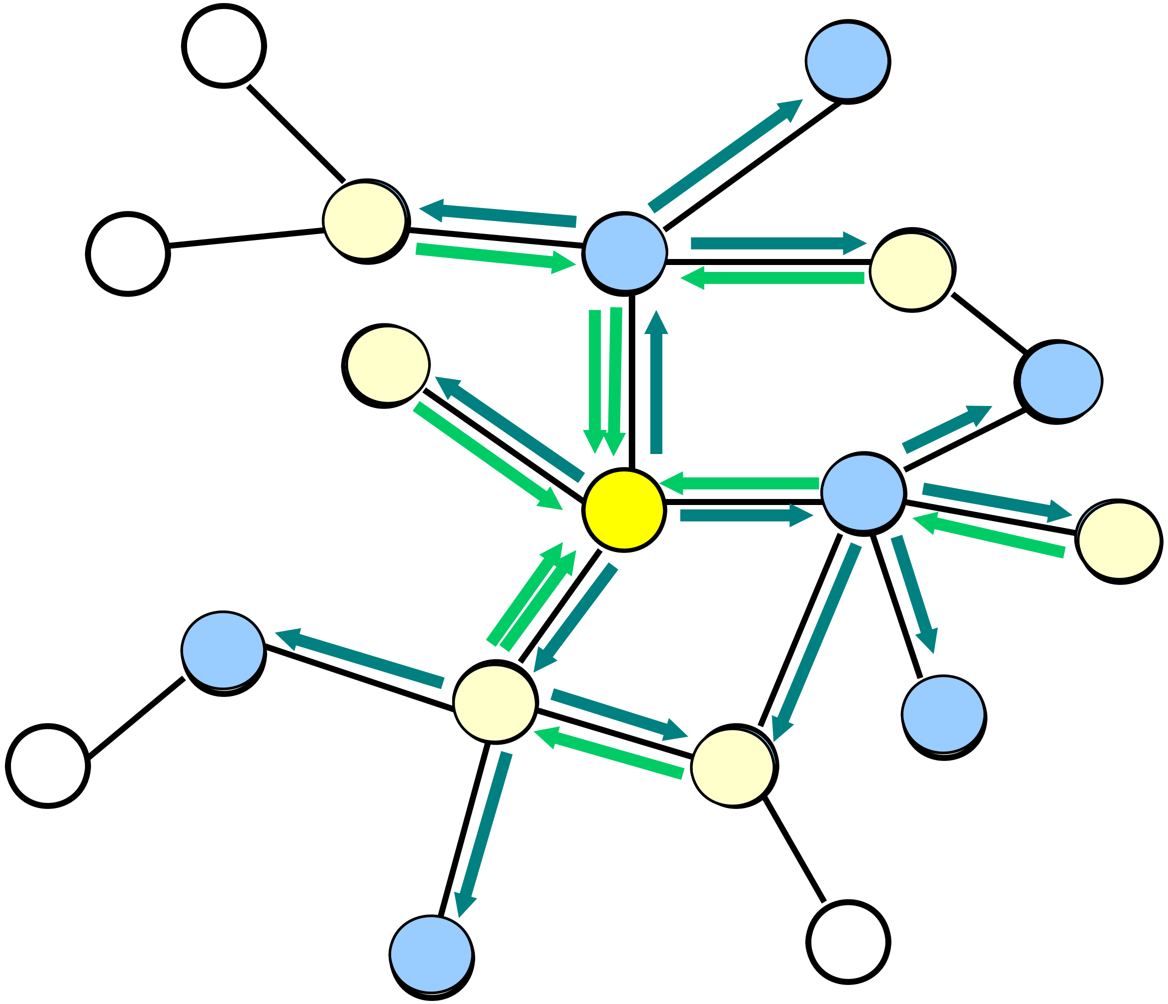
Performance
• Efficiency
• Load-balancing
• Authentic Services
• Prevention of DoS
• Incentives
Correctness
Participation

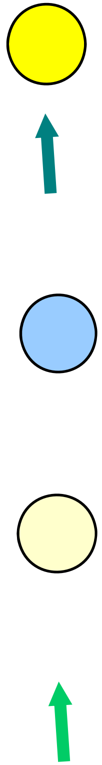
Search in P2P

- **Overlay Network** controls:
  - Connections made by users (topology)
  - Data placement
- Tight control: “Structured”
  - Efficient, comprehensive
- Loose control: “Unstructured”
  - Inefficient, not comprehensive, simple, expressive
  - **Used in *real life***

Humboldt Universität Informatik
Systems Architecture Group
http://sar.informatik.hu-berlin.de

Unstructured – Query Flooding





Problems with unstructured

- Inefficient
  - Query messages are flooded
  - Even if routing is intelligent, *worst case* load is still O(n), where *n* is # nodes in system
- Not comprehensive
  - If I do not get a result for my query, is it because none exists?
- (Of course, many optimizations are possible…)

⇨ Structured systems address these problems

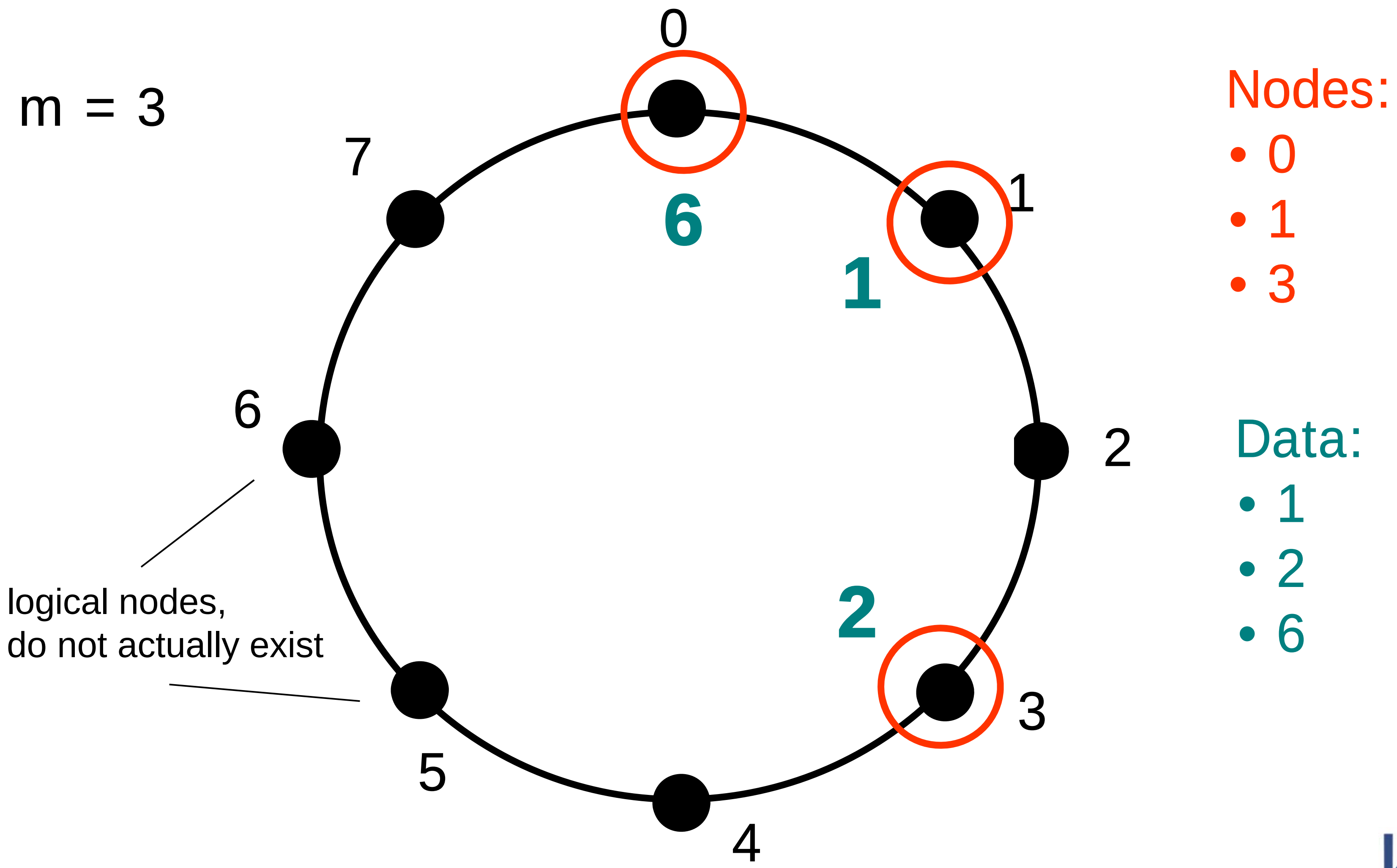
Distributed Hash Table (DHTs)

- Hash Table
  - Key/Object pair
  - Key is *hashed* to get an ID
  - Operation: lookup(ID) → object(s) with corresponding ID

  Ex. Object → file; Key → file name; ID → hash of file name

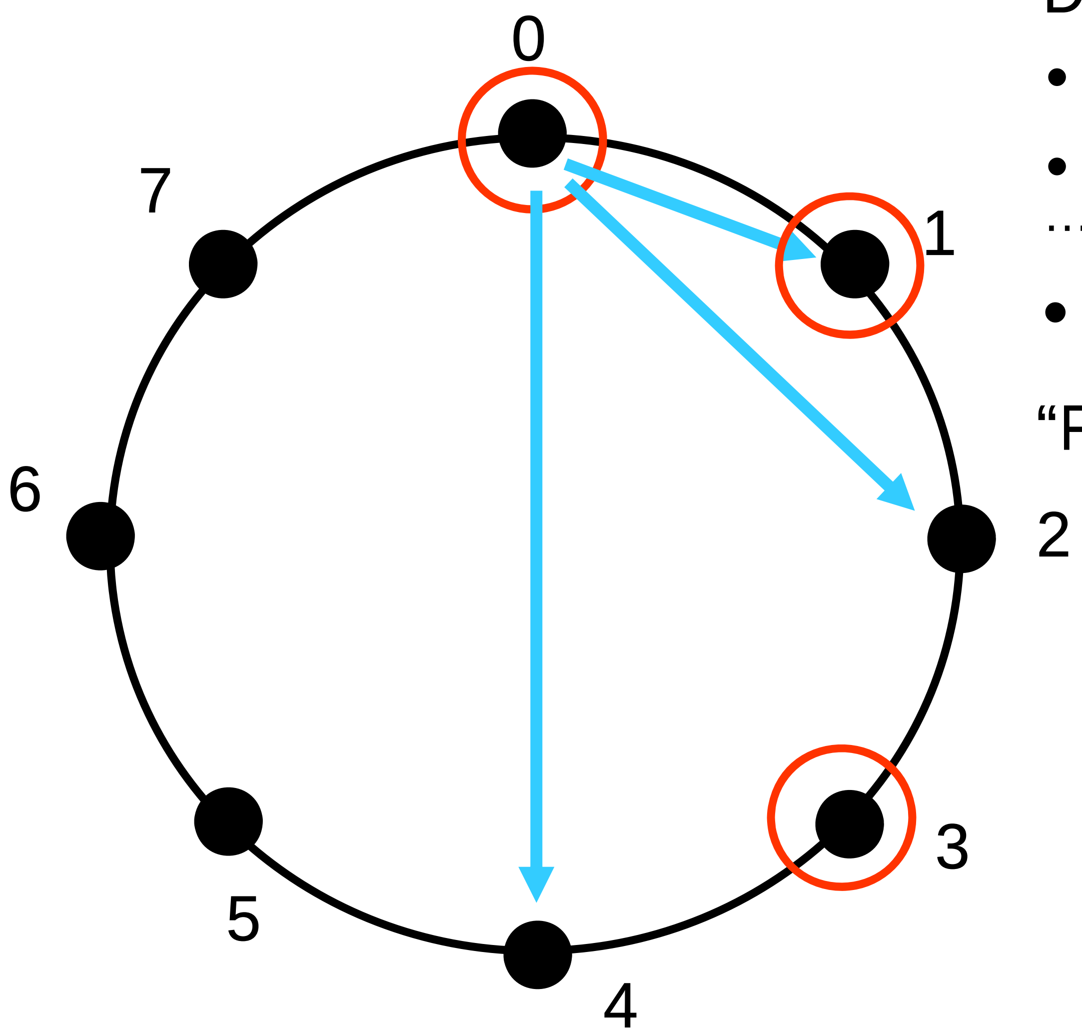
- Nodes are assigned IDs
  - An object is stored on the node following the node with the largest ID smaller than the object ID
- Problem. Find node that stores object(s) for a given ID

Data Placement



Systems Architecture Group
http://sar.informatik.hu-berlin.de

Connections – “Finger” Tables



Distance

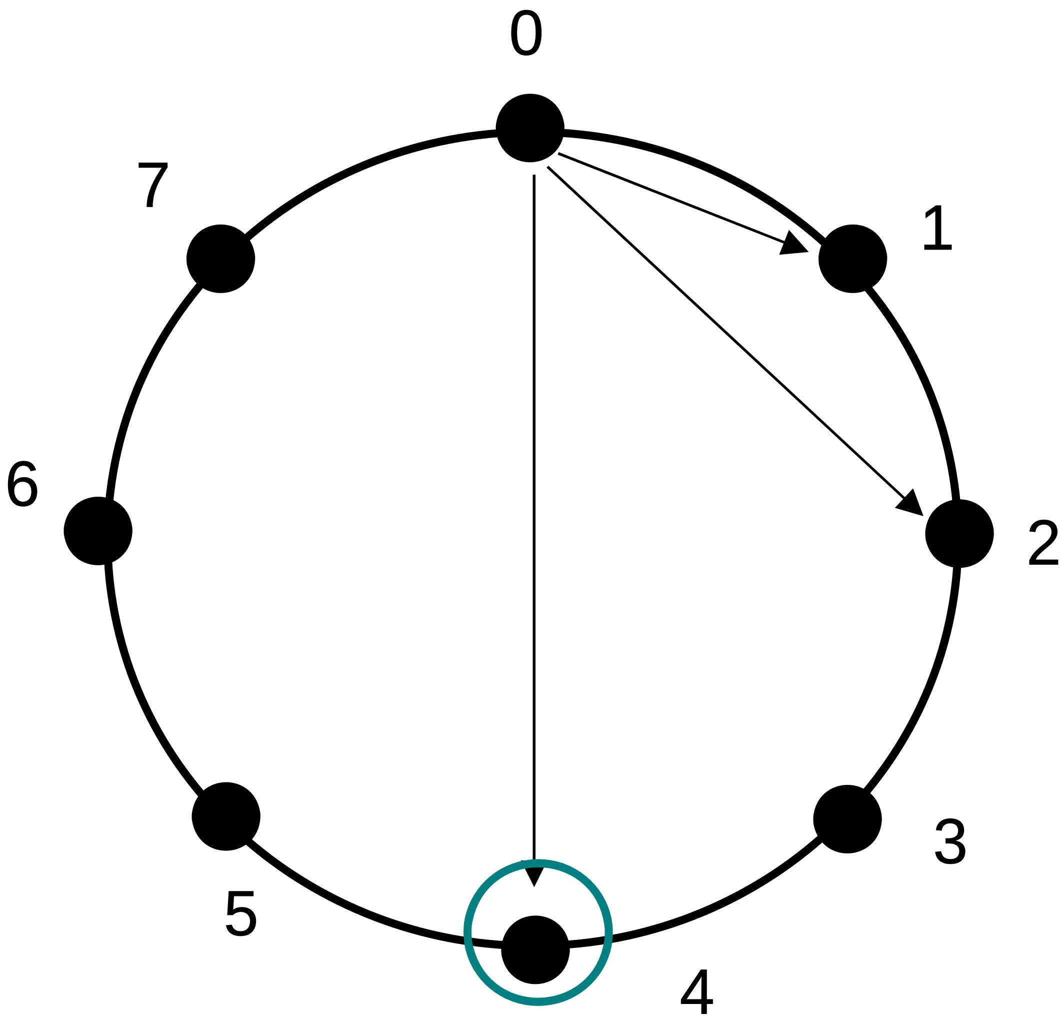
- 2^0
- 2^1

….

- 2^{m-1}

“Finger pointers”

Query Example

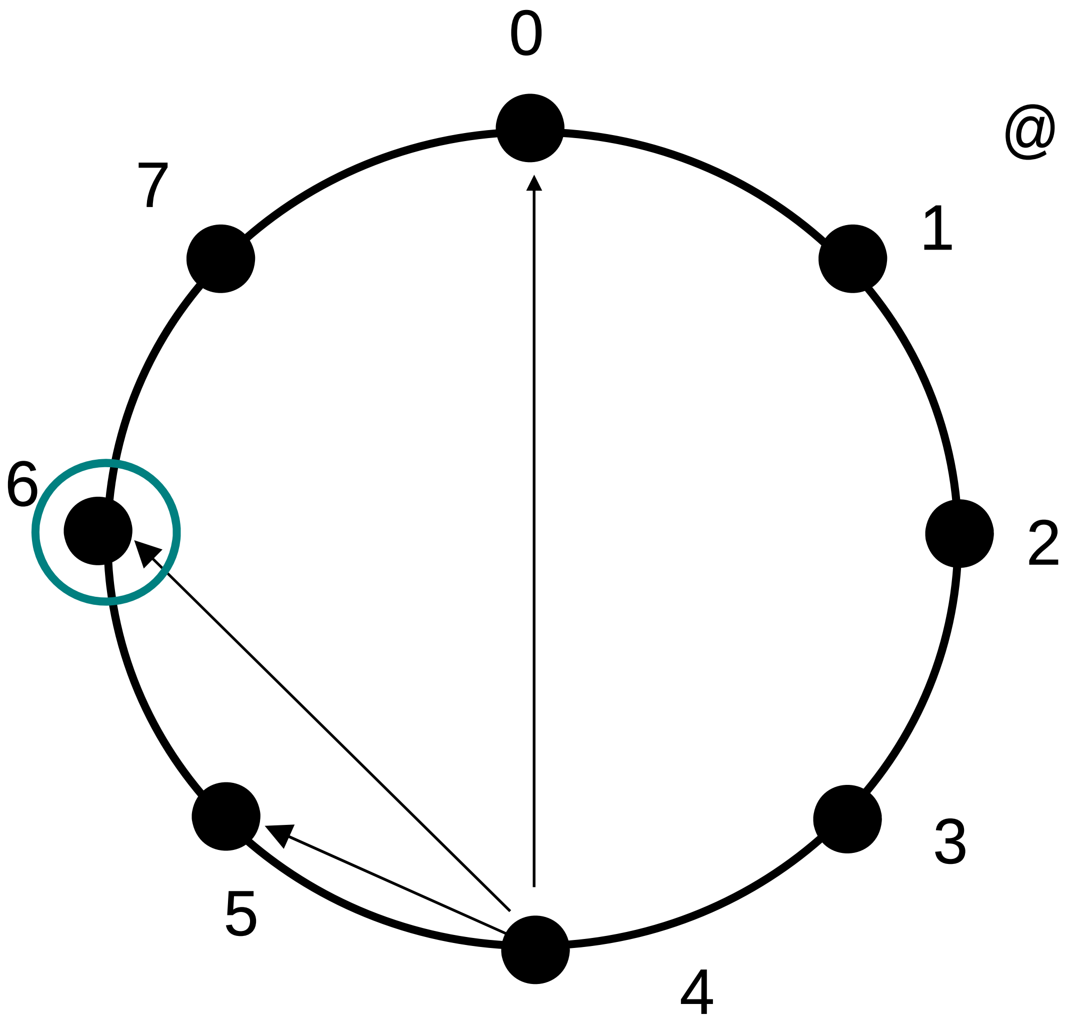


Say node 0 wants to find the object with ID = 7
For simplicity, we will assume a node exists at every ID in the space

Node 0: Lookup(7)

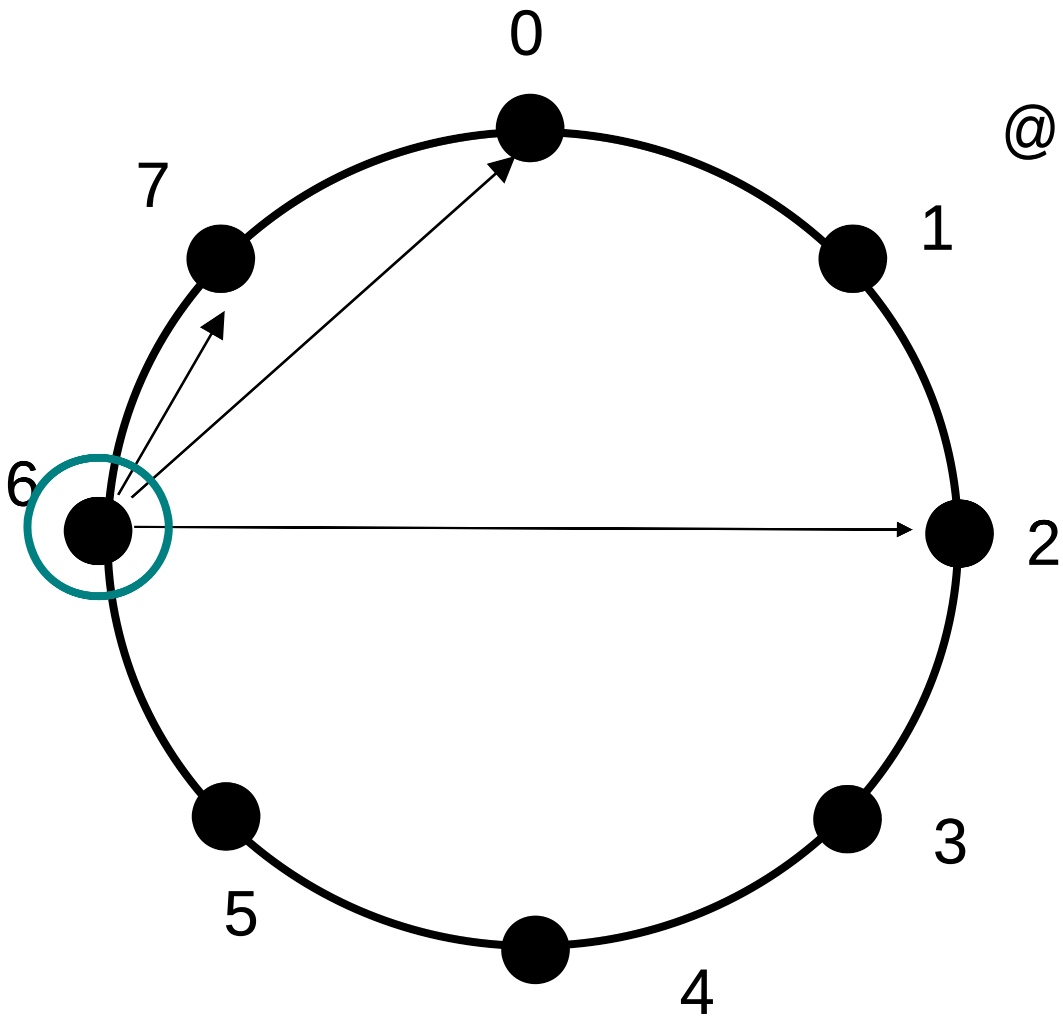
@ Node 0: FindPred (7)

Query Example



@ Node 4: FindPred(7)

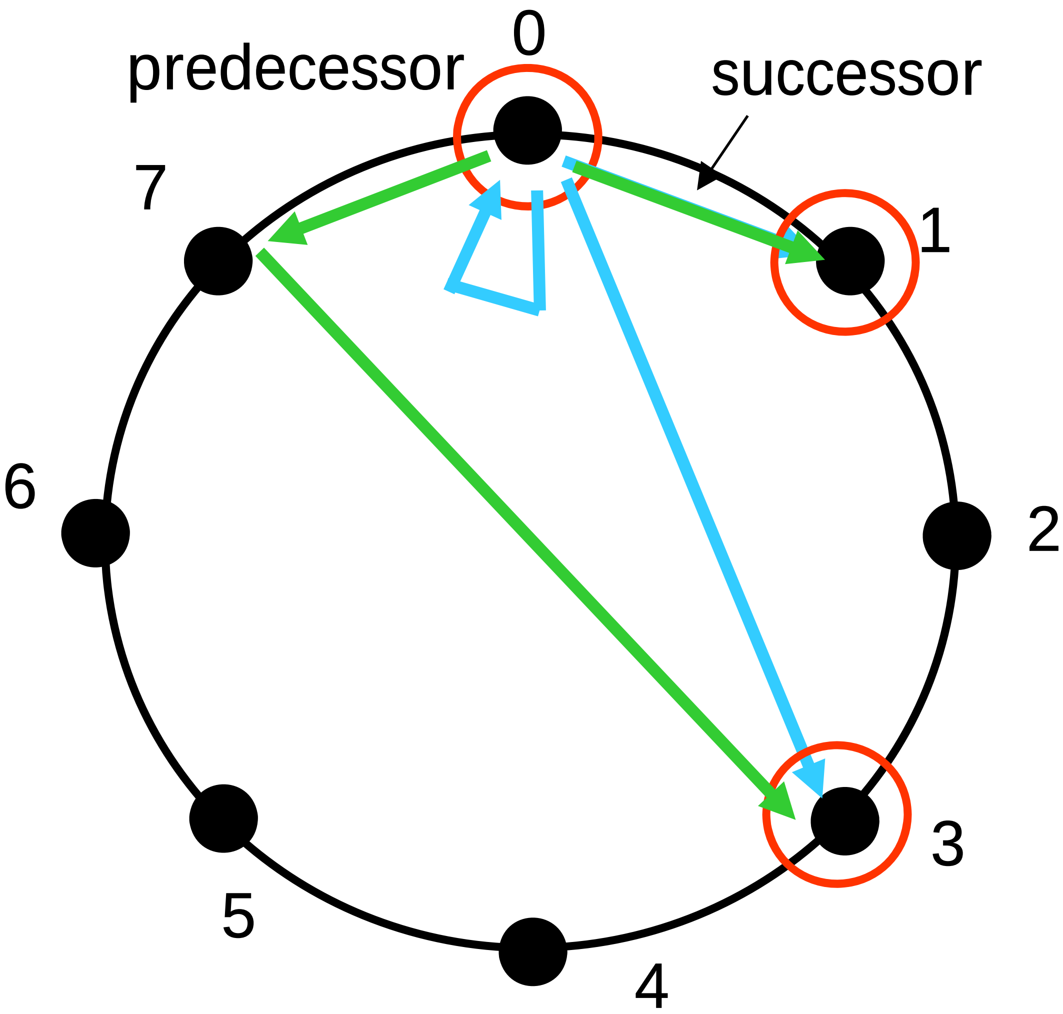
Query Example



@ Node 6: FindPred(7)

Node 6 is predecessor
Return successor node 7

Connections – “Finger” Tables



Query characteristics

- N = total nodes in the network
- With high probability, a query can be answered by contacting O(log N) nodes
  - ➔Efficient!
- If an object with the ID exists in the network, it will be found
  - ➔Comprehensive!
- State is also O(log N) in size

Disadvantages?

- Cost of joining and leaving
  - $O(\log^2 N)$ messages
  - Moving objects (potentially large files!) around
- Instability
  - If one node joins or leaves, no problem
  - If many nodes join and leave at the same time, can the finger pointers really fix themselves?
  - Even if they can, how slow are queries in the meantime?
- Availability of Data
  - If a node dies suddenly, what happens to the data it was storing?
  - MUST replicate data across multiple nodes

Problems?

- What exactly is an ID?
  - IP address? Very easy to spoof
  - If a peer can have many IDs, it would be easy for him to take control of the “secure” score management
  - The “Sybil attack”
    - If IDs are easy to generate, no system is secure
  - How can we make IDs difficult to generate?
    - Centralized authority, crypto puzzles, etc

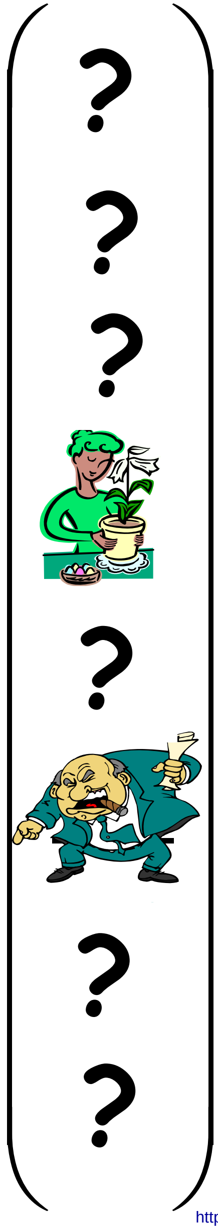
- How to motivate Participation?
- Reliability
- Correctness / Quality of result (Security)
- Scalability

Reputation

- Past History
  - Good past experience with peer → more likely to interact again with that peer
  - Bad past experience with peer → more likely to avoid that peer
- Implementation
  - Each peer ***i*** has a "trust vector" $\mathbf{c}_i$ to determine how likely they are to interact with other peers

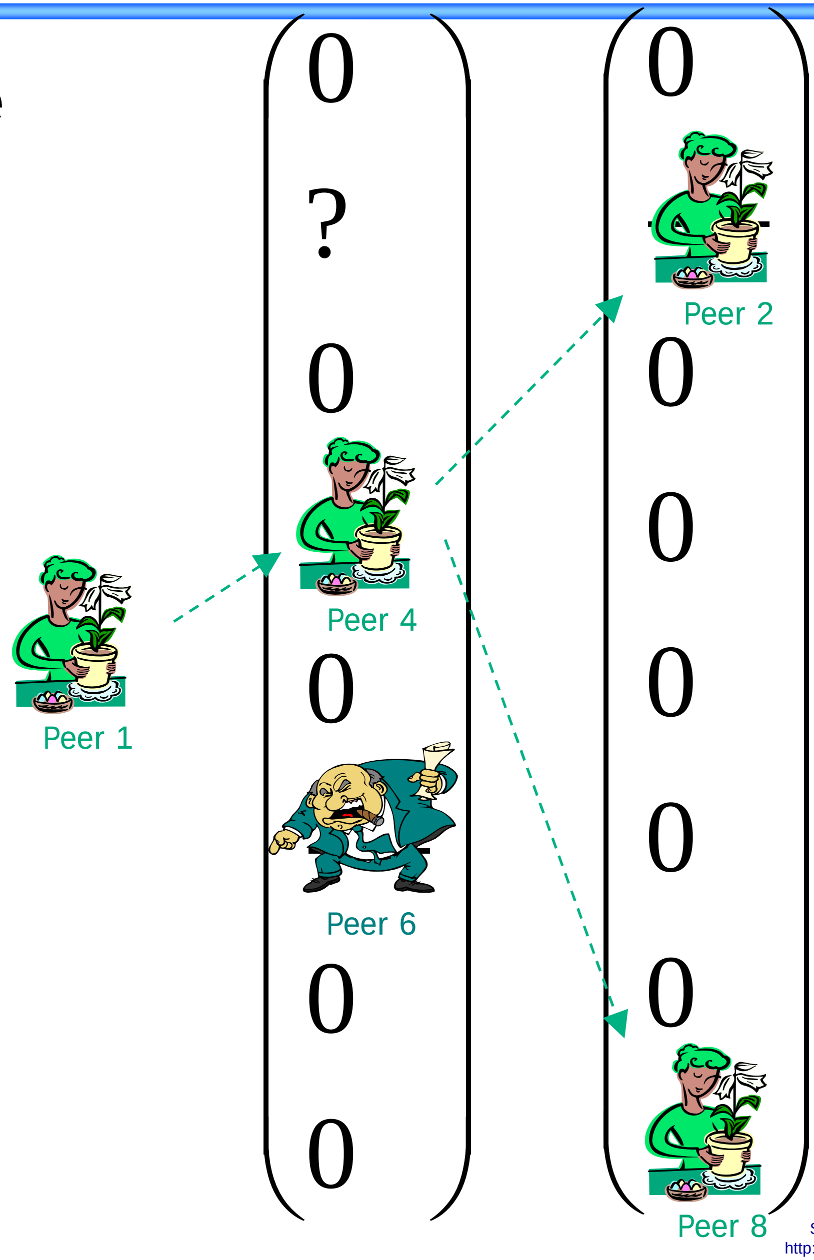
Past History

- Problem?
- Each peer has limited past experience
  - I know few peers out of the entire network
  - Most of the time, I will not have an opinion on a peer
- Solution?



EigenTrust: Friends of Friends

- Ask for the opinions of the people you know
- Weight their opinions by your trust in them



The Math

$$c'_{ik} = \sum_j c_{ij} \cdot c_{jk}$$

You are peer i

Ask your friends j

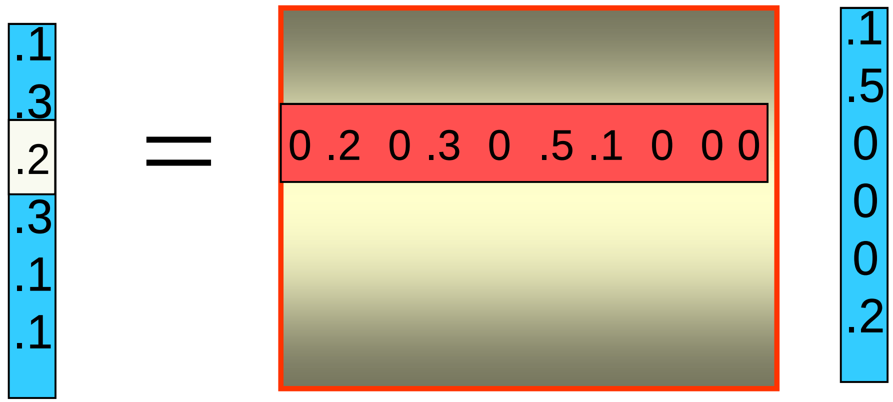
And weight each friend's opinion by how much you trust him.

What they think of peer k.

Humboldt Universität Informatik
Systems Architecture Group
http://sar.informatik.hu-berlin.de

The Math

$$c'_{ik} = \sum_j c_{ij} \cdot c_{jk}$$



$$\mathbf{c}'_{\mathbf{i}} = \mathbf{C}^{\mathbf{T}} \mathbf{c}_{\mathbf{i}}$$

My new trust vector

Is the multiplication of

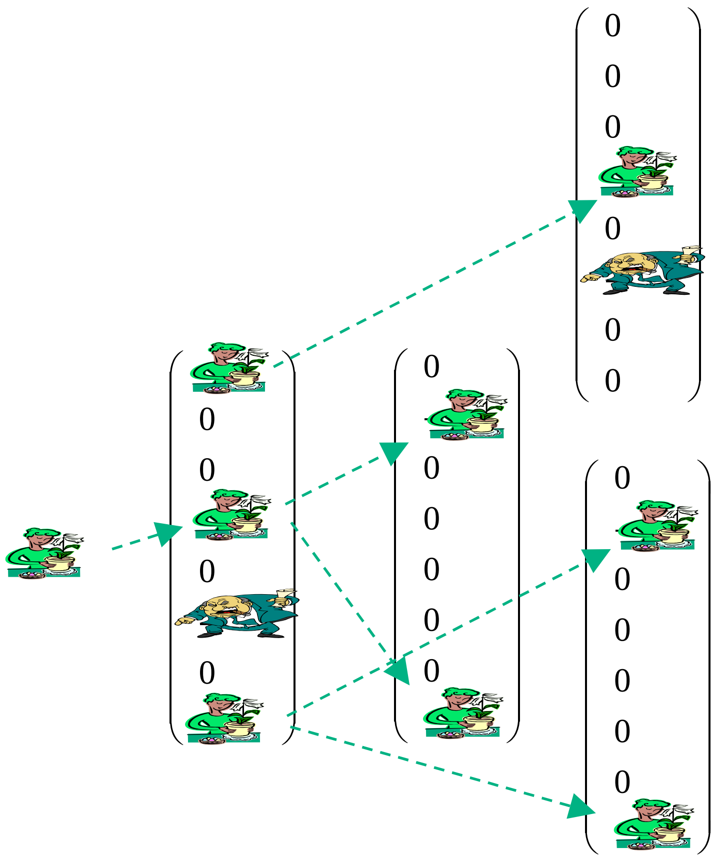
And the transpose of the trust matrix

My old trust vector

Problem with Friends

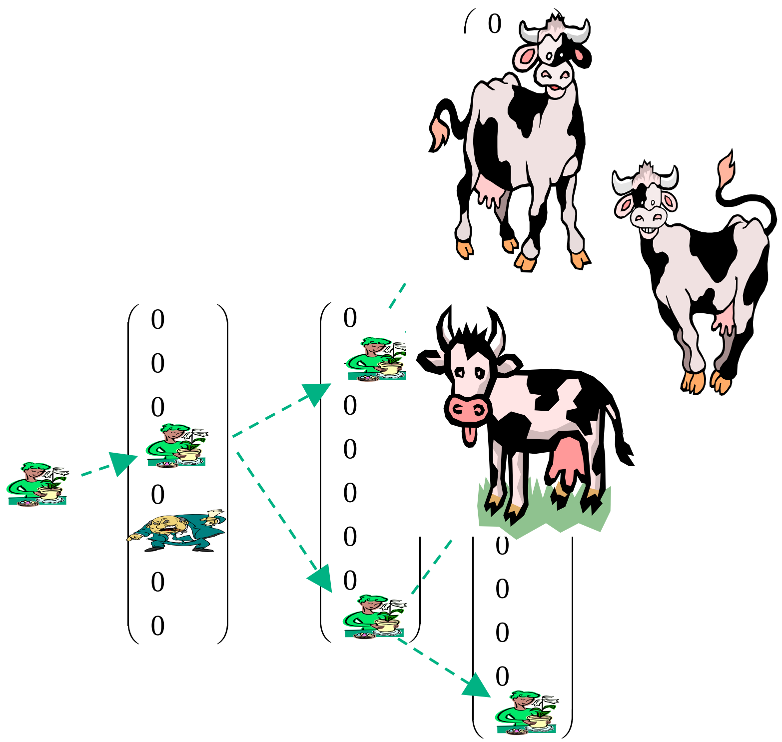
- You know a lot of peers
  - You have to compute and store many values.

- You know few peers
  - You won't know many peers, even after asking your friends.



Knowing All Peers

- Ask your friends: $t = C^T c_i$
- Ask their friends: $t = (C^T)^2 c_i$
- And their friends: $t = (C^T)^3 c_i$
- Keep asking…. …forever?



Minimal Computation

- Luckily, the *trust vector* **t**, if computed in this manner, ***converges*** to the same thing for every peer!
  - I ask my friends…forever…
  - You ask your friends…forever…
  - ➔ After a while, my trust vector stops changing
  - ➔ When my vector stops changing, and your vector stops changing, we end up with the ***same vector***