ROYAL INSTITUTE OF TECHNOLOGY

ID2210 Project 2013

Dr. Jim Dowling
School of Information and Communication Technology
jdowling@kth.se

Project Outline

- Implement a decentralized search protocol in Kompics.
- Analyze the results.

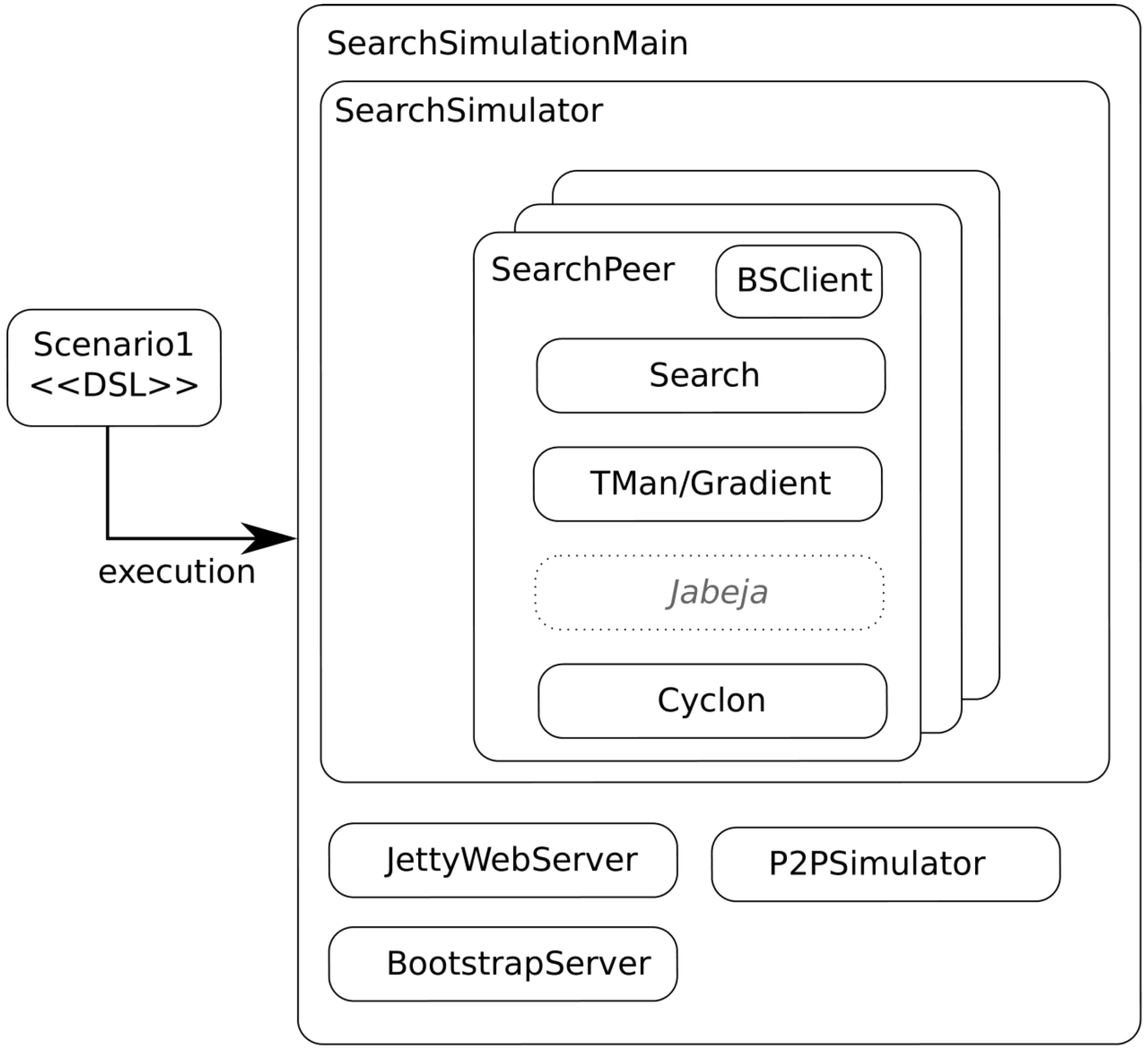
- Kompics is a Java actor/component framework that contains a P2P simulator
- Available at http://kompics.sics.se

Things to Deliver

- Your source code.
- A report in pdf format, in which you should present your measurements and discuss the results.

- Deadline: 24th May 2013
  - You will receive a grade on the scale F, FX, E, D, C, B, A.
  - If you receive an F for the project, you must repeat the project.

Scaffolding Code



https://github.com/jimdowling/id2210-vt13

Random Networks and Gossiping

- Efficient and robust information propagation
  - Low diameter networks, redundant paths.

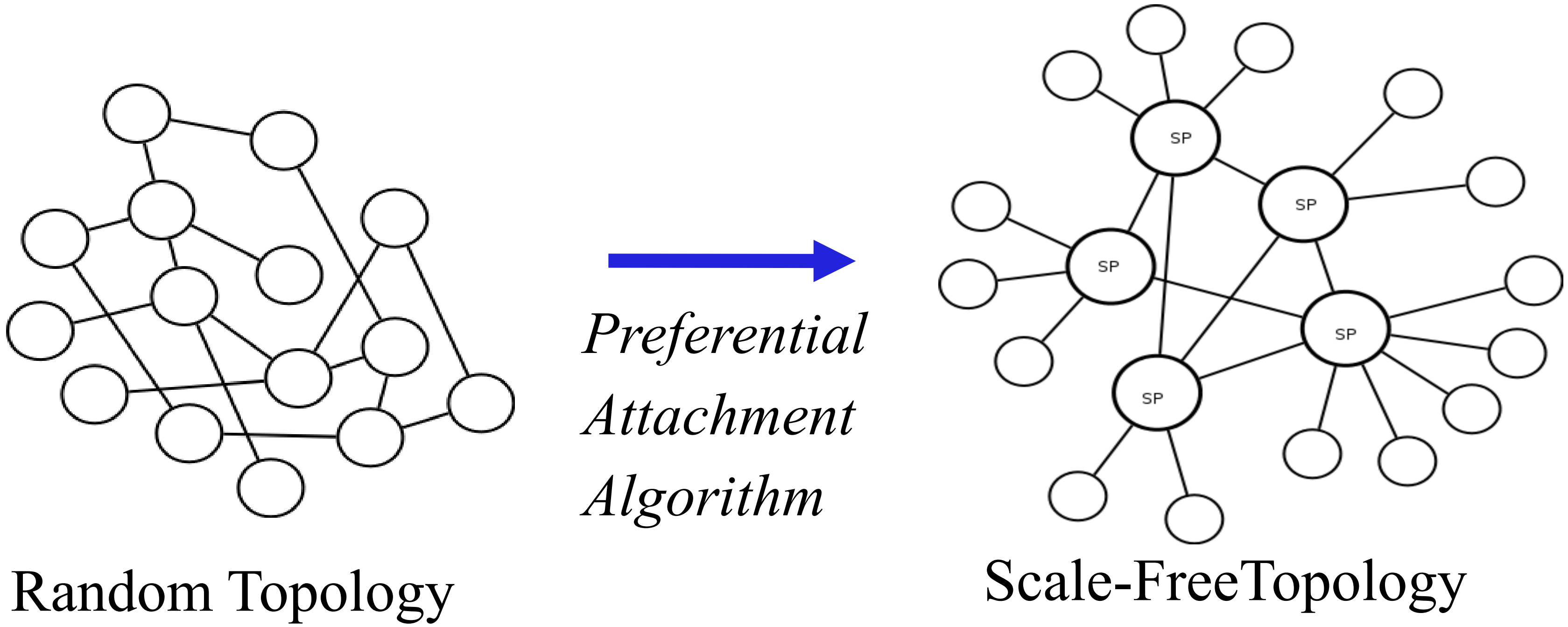
- Symmetry of random networks means we can only use message-passing to share information between nodes.

- Leader-selection requires breaking the symmetry in random networks.

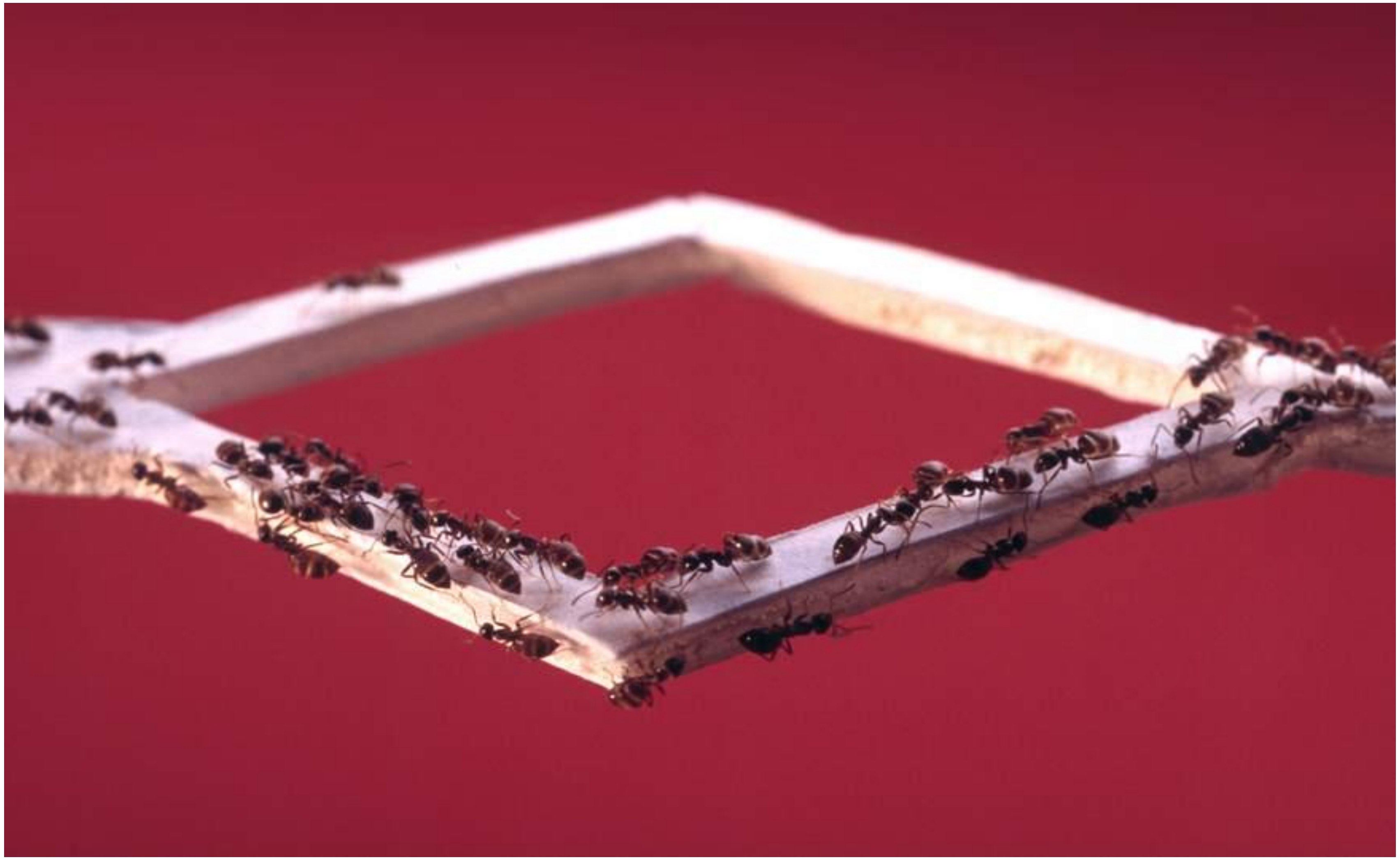
ROYAL INSTITUTE OF TECHNOLOGY

Scale-Free Networks [Barabasi]

- New nodes preferentially create links to those nodes with a higher number of links (positve feedback).
- *Symmetry breaking* from a random network.
  - Nodes now can use information encoded in the topology to send search requests to hubs.



Ant Foraging – from Random to Ordered*



*Foraging patterns break both spatial and temporal symmetry

Symmetry Breaking

- Symmetry breaking is about going from a more disordered state to a more ordered state.

- In overlay networks terminology, this means going from a random overlay network to a network with some structure or order.

Leader Election

Leader Election in Distributed Systems

- Leader election concerns the designation of a single process as the coordinator of a task/service distributed among several processs.
- Why have a leader?
  - A centralized coordinator simplifies the synchronization of processs.
- Main problem to solve
  - If the central coordinator fails, the service provided by the coordinator and processs may fail.
  - **The processes need to re-elect a new leader, if the current leader fails.**

Leader Election Algorithm Properties

1. The algorithm is decentralized among a set of processes. Each node executes the same local algorithm.
2. The election algorithm must somehow break symmetry among processes in the system.
3. The algorithm reaches a terminal configuration in each computation, where there is exactly one process in the state *leader*.
4. After the election algorithm has terminated, all participants must know who the leader is.

Breaking node symmetry (ranking processes)

- Breaking symmetry requires globally ranking all processes
  - E.g. rank ordering $p_1>p_2>p_3>p_4$

- Ranking is typically based on one of either:

1. Static global knowledge - leader election is based on global priority (e.g., process-id).
2. Preference-based – processes in the group vote for the leader based on personal preferences (reliability, capacity, locality).

Leader Election Saftey and Liveness

- A good leader election algorithm should have the following properties:

- **Safety**: A participating process `P(i)` has a variable `elected(i) = ⊥` or `elected(i) = P`, where `P` is the non-crashed process at the end of the run with the highest identifier.
- **Liveness**: All processes `P(i)` participate and eventually `elected(i) != ⊥` or `crash`.

Leader Election Performance

- Leader election performance is measured using the number of messages necessary to complete the algorithm
  - #msgs is proportional to the total network bandwidth used.

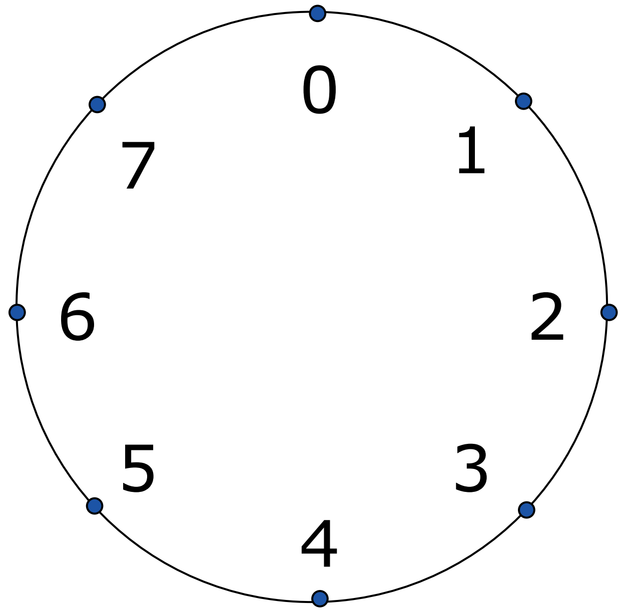
Example: Ring based leader election

- There is no deterministic algorithm for leader election in an anonymous ring.
- The initial state of all processes is completely **symmetric**.
- All processes take the same action.
- The new state is also symmetric.
- This can be repeated forever.

Break Symmetry by Introducing an ID into the ring

Chang-Robert Leader Election Algorithm

- Every node sends an *election* message with its id to the node on its left, if it has not seen a message from a higher node.
- Forward any message with an id greater than its own id to its left.
- If a process receives its own *election* message, it is the leader.
- It then declares itself to be the leader by sending a *leader* message to all processes.



Aside: Distributed System Models

- In a *synchronous system model*, timeouts are enough to detect leader failure.

- In an *asynchronous system model*, unreliable failure detectors are typically used to identify leader failure.
  - Failure detectors to encapsulate timing assumptions
  - An unreliable failure detector gives suspicions regarding node failures

Bully Algorithm (Garcia-Molina)

- Assumptions
  - Nodes are numbered (static global knowledge).
  - Synchronous network model.
    - There are no network communication errors.
    - The network communication subsystem does not fail
- Property
  - The highest-numbered node (bully) becomes the leader.

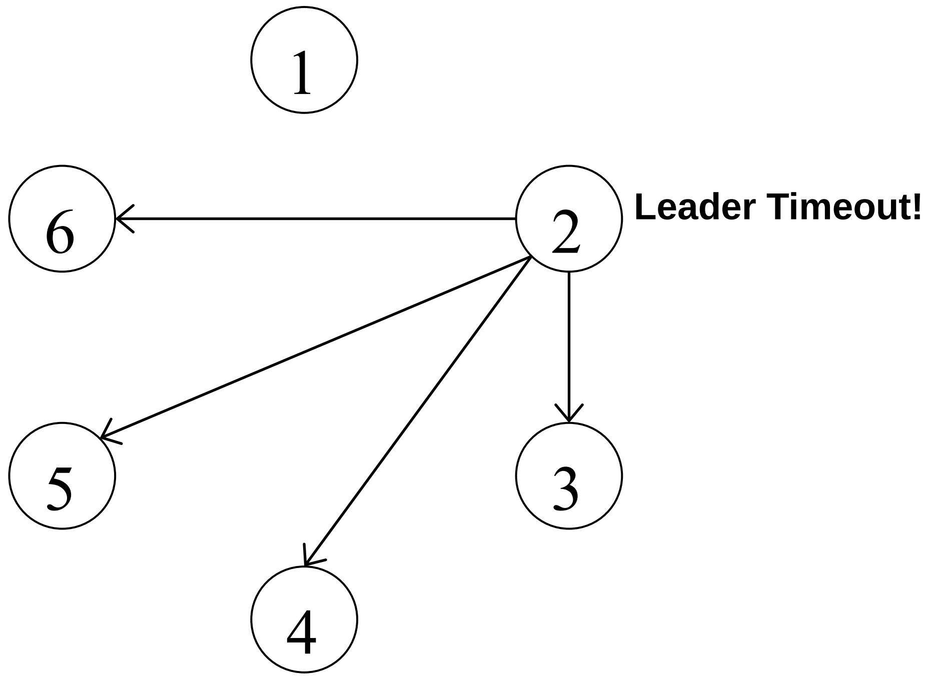
Bully Algorithm

- Three types of messages:
  - **election messages** used to announce an election;
  - **ok messages** sent in reply to election messages;
  - **coordinator messages** sent when the election is complete to announce the election result—the new coordinator.
- An election is started by any process that detects (by timeouts) that the current leader has failed.
- If the process that identifies the leader failure has the highest remaining ID, it becomes the leader by sending a coordinator message to all nodes.
  - No election is required.
- Otherwise, if not the highest ID, an election is called.

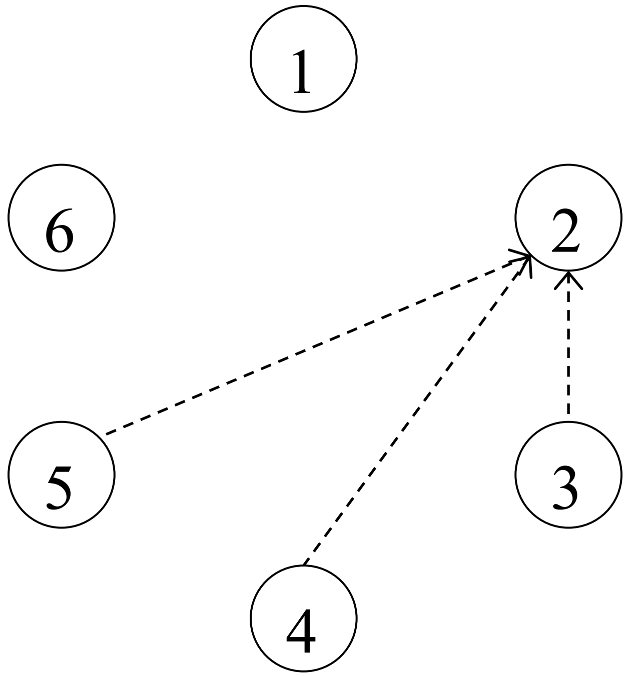
Bully Algorithm

1. P sends an `election` msg to all processes with higher IDs than itself. P waits T seconds before becoming leader.
2. If P does not receive an `OK` msg from a node with a higher ID than itself, it wins the election and sends a `coordinator` msg to all other processes.
3. If P receives an `OK` msg from a node with a higher ID, P waits ~(T*2) seconds for that node to send a `coordinator` msg. If it does not receive this msg in time, it re-broadcasts the `election` msg.
4. If P receives an `election` msg from another process with a lower ID it sends an `OK` msg back and starts a new election.

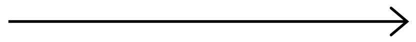
Bully Algorithm Operation (1/3)



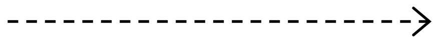
Step 1: Node 6 (the leader) is detected as failed by node 2. Node 2 starts an election.



Step 2: Nodes 3-5 respond with an OK msg. Node 2 now stops and waits for others to act.



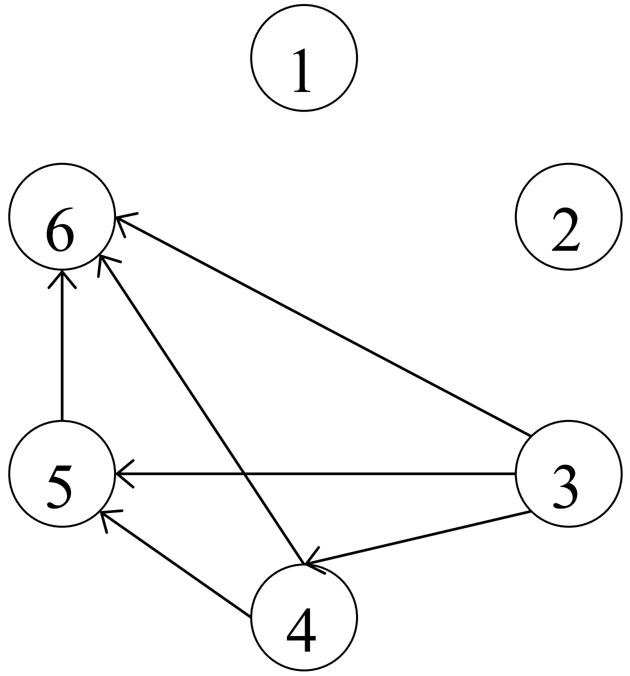
ELECTION msg



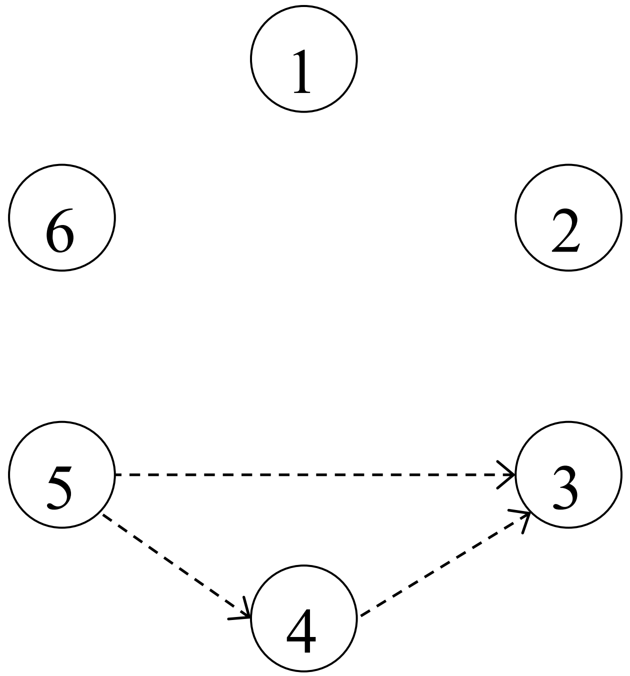
OK msg

ROYAL INSTITUTE OF TECHNOLOGY

Bully Algorithm Operation (2/3)



Step 3: Nodes 3-5 all received election messages from 2, which has a lower ID, so they each independently start election algorithm.



Step 4: Nodes 5 and 4 send OK msgs to processes 3 and 4. Only node 5 doesn't receive an OK msg.

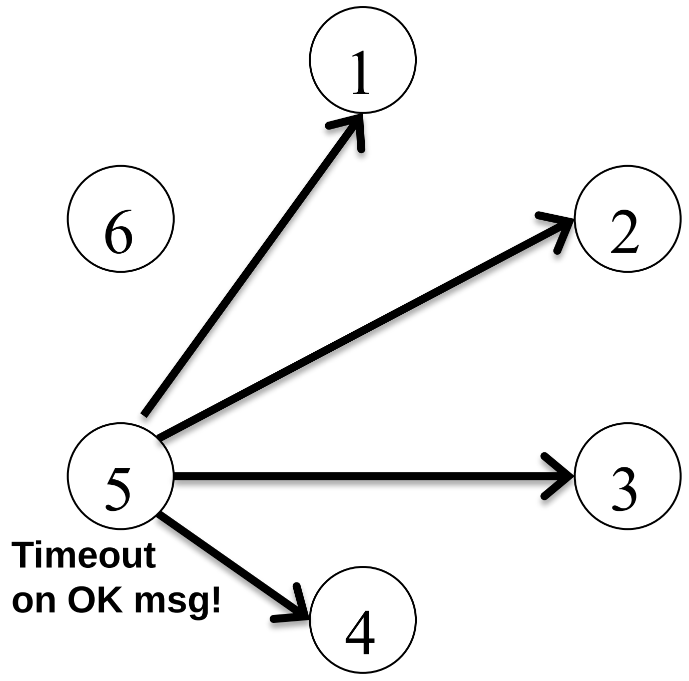
$\longrightarrow$

ELECTION msg

$\dashrightarrow$

OK msg

Bully Algorithm Operation (3/3)



Step 5: Node 5 times out waiting for an OK response.
It now becomes the new leader and broadcasts to surviving processes that it is the new leader.

COORDINATOR msg

Bully Algorithm Performance

- N processes

- Worst Case:
  - Smallest node initiates election
  - Requires O(n*n) messages

- Best Case:
  - Eventual leader initiates election
  - Requires (n-1) messages

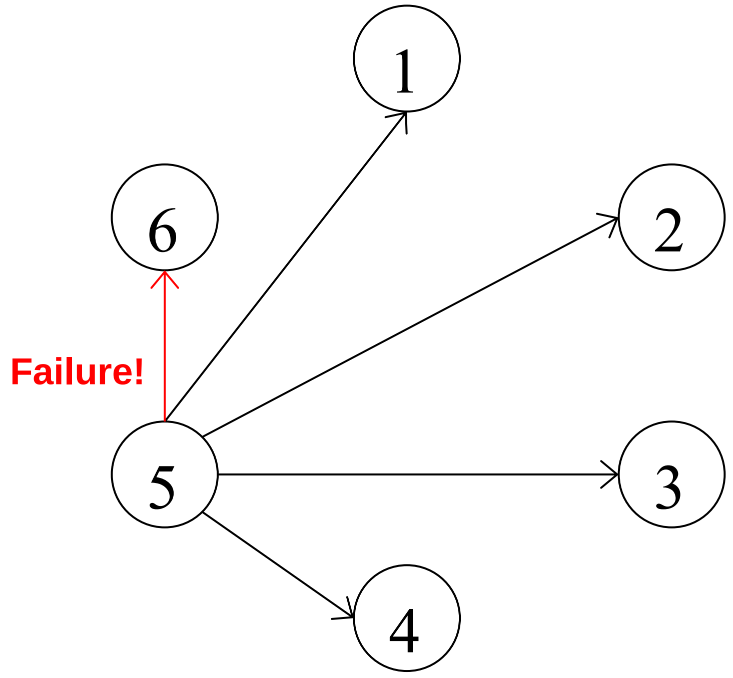
Question

- If we run the bully algorithm between processes on the Internet, can there ever be a point where the highest number process is not the leader?

- Yes! The Internet is an asynchronous system.
  - We could re-write the Bully algorithm for asynchronous systems.
  - But failure-detectors on the Internet are unreliable.

ROYAL INSTITUTE OF TECHNOLOGY

Bully Algorithm for asynchronous systems



Node 5's failure detector incorrectly believes node 6 has failed, and starts an election. Nodes 1..4 also have failure detectors to node 6. What do processes 1..4 answer? What happens if only some of the processes have detected that node 6 has failed?

$\xrightarrow{\hspace{4cm}}$

Election msg

Invitation Algorithm

- Works for asynchronous systems
  - Assumes that delay can be arbitrary and that there is no global coordinator

- The invitation algorithm classifies processes into groups and elects a coordinator for every group.

- The invitation algorithm does not make assumptions about bounded response time and can work correctly in the presence of timing failures.

Partitioning and the Need for Multiple Groups

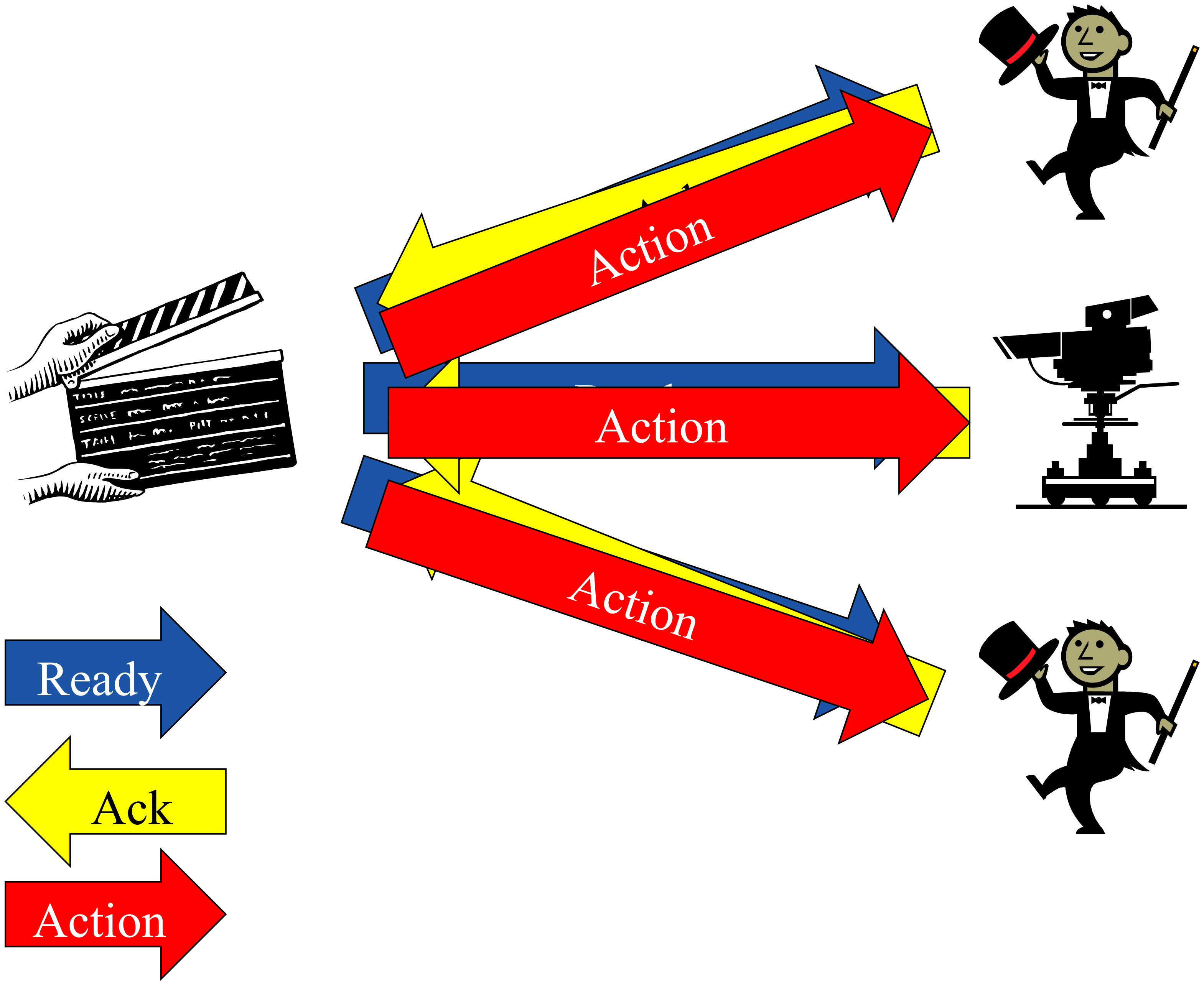
- If a network partitions, making communication impossible between two subsets of the processes within a group, it no longer makes sense to think of a single global coordinator.
- Instead it is more appropriate to think in terms of a coordinator for each subgroup.
- However, if our system is very large, we can assume the probability of it partitioning is very low
- It is reasonable to assume that updates in the smaller partition will be lost on a merge.

Agreement Protocols for Leader Selection

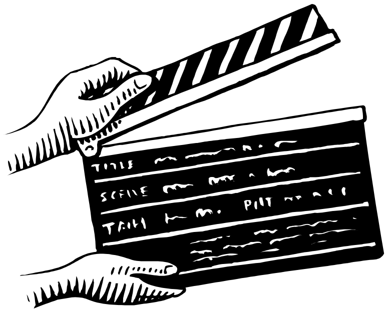
Agreement Protocols are Everywhere

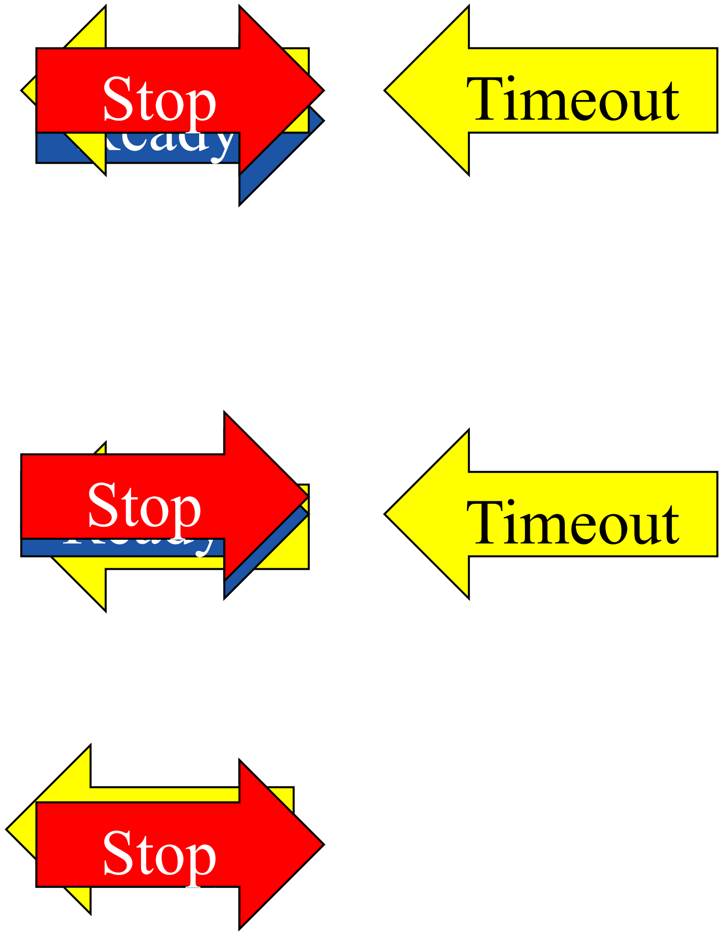
• Marriage ceremony	• Do you ...? I do. I now pronounce you...
• Athletics	• On your marks... Get set... Go!
• Aircraft take-off	• Ready to take off X? Check. Clear to take off! Check.

Agreement protocol for movies



If a majority say no the deal is off.





KTH VETENSKAP OCH KONST
ROYAL INSTITUTE OF TECHNOLOGY

Consensus Algorithms

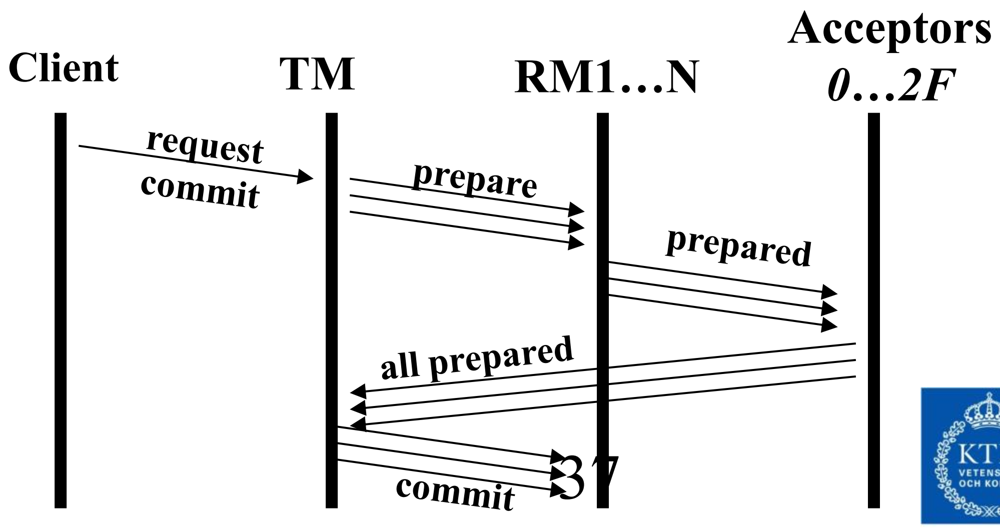
- “Reaching Agreement in the Presence of Faults”
  - Shostak, Pease, & Lamport, JACM, 1980

- *N* nodes try to *agree* on a *value*, even if *F* of the nodes have failed.
  - When *F=0* (no fault-tolerance) == 2PC

- Properties of consensus algorithms
  - Any value decided is a value proposed
  - No two correct nodes decide differently
  - Every correct node eventually decides
  - A node decides at most once

Paxos Commit

- *N* resource managers RMs (participants)
- *2F+1* acceptors (~*2F+1* TMs)
- If *F+1* acceptors see all RMs prepared, then the transaction is committed.
- Protocol Cost: *2F(N+1)* + *3N* + *1* messages
  5 message delays
  2 stable write delays.

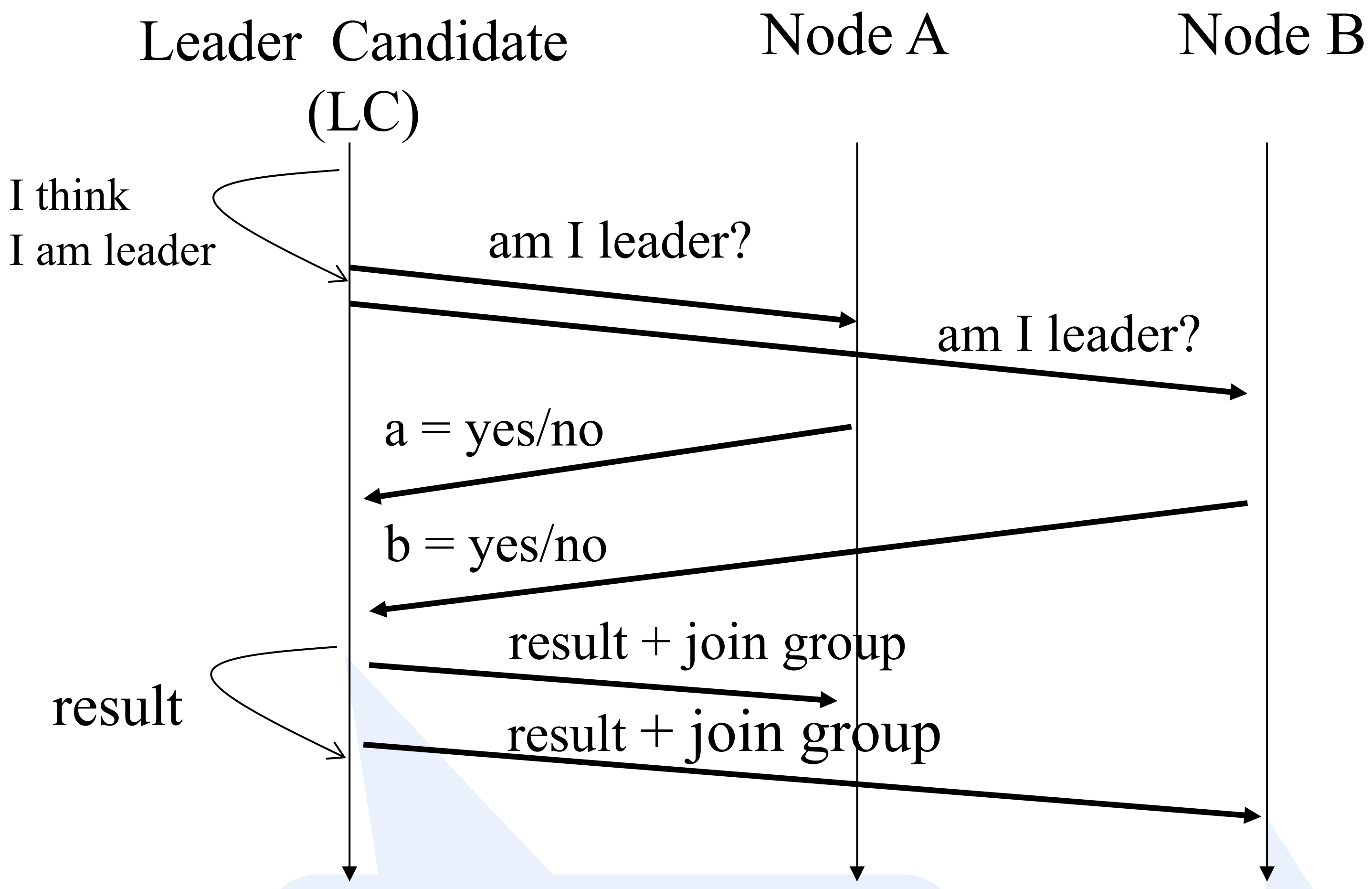


Paxos Commit

- $3N+ 2F(N+1) +1$ messages
- $N+2F+1$ stable writes
- 5 message delays
- 2 stable-write delays

Quorum voting for leader election

Leader Election as Quorum Decision



```
If A is yes && B is yes
    result = "new leader"
else
    result = "abort"
```

B updates leader upon receiving "new leader"

Practical Issues to consider

- Which node initiates the LC protocol?
  - Under what conditions does a node start the LC protocol?
- What about timeouts?
  - LC times out waiting for A (or B)’s “yes/no” response
  - A times out waiting for LC’s result message
- How do we decide on the election group members?
  - What happens if a node dies – will the election group members be changed?
  - How do we reach agreement on the election group members?

ROYAL INSTITUTE OF TECHNOLOGY

Handling Leader Failure

- Heartbeats should be used by nodes to identify if the leader fails.
- Heartbeats should be used by the leader to identify if members of the election group fail.
- Nodes in the election group should reach agreement that the leader has failed.
  - A majority will suffice – but they have to inform all members of the election group.

ROYAL INSTITUTE OF TECHNOLOGY

Main challenges

- What happens if a new node with an even higher utility than the leader joins the system?

- Should it run the leader election algorithm immediately when it joins the system or wait until it has converged in the Gradient?

Failure Detectors

- Simple failure detector for leader failure
  - Periodically sent a heartbeat message to the leader.
  - Start a timeout based on the worst case msg round trip.
  - If the timeout expires, then suspect the leader has failed.
    - Inform all nodes about the suspected leader failure
  - If the heartbeat response is recvd from a suspected leader, remove the suspicion and increase time-out.

Leader Selection for Large-Scale Dist. Sys.

Leader Selection

- In a large-scale distributed system, it is too costly to run election algorithms over all processes.
  - Takes too long to elect a new leader.
  - Generates too much network traffic.

- We want to weaken the properties of the leader election algorithm to enable it to scale to large-scale distributed systems.

Leader Selection Properties

- When a leader election algorithm is executed, all nodes in the system reach agreement on a single leader node. All nodes know who the new leader is.
- When a leader selection algorithm is executed, all nodes in the system reach agreement that a single node has become the leader.
- Only a (small) subset of nodes may actually know who the leader is at any given time instant.
- Any node in the system can discover who the leader is in reasonable time.

ROYAL INSTITUTE OF TECHNOLOGY

Leader Selection Properties

- **Safety**: Only max 1 node at a time is a leader, with high probability.
- **Liveness**: Eventually, there will be a leader in the system.
- **Discovery**: Nodes can discover who the leader is in a short bounded period of time without generating an excessive number of messages.

Distributed Ranking with Overlay Networks

Distributed Ranking using Overlay Networks

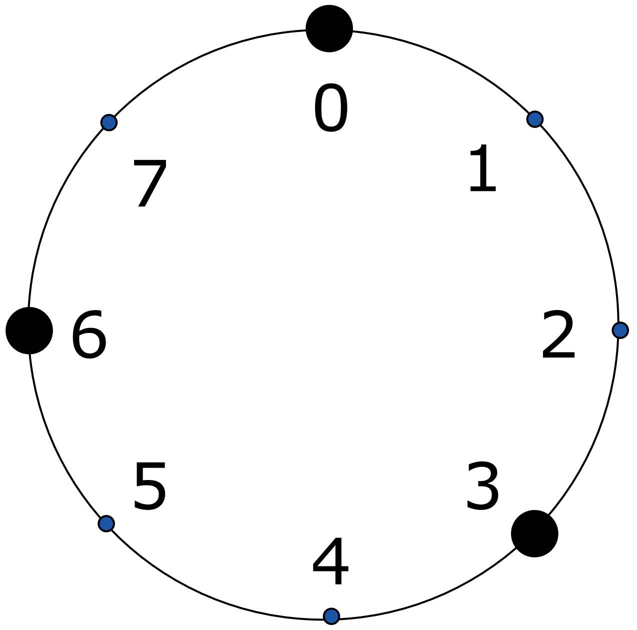
- We would like a distributed algorithm for constructing an overlay network that orders nodes by some ranking function.

- Static global knowledge
  - Structured overlay networks that have a regular node labeling scheme could be used to select a leader.

- Preference-based
  - T-MAN
  - **Gradient overlay network**

Ring-based Structured Overlay Networks

- We could use a well-known position in the ring to identify the leader
  - O(log N) search-time to find the leader.
  - Liveness property satisfied.
  - **Safety property violated.**
    - Failure detectors in ring-based SONs mean that more than 1 node may be identified as the leader at any point in time.
    - To get SONs to work for leader selection, you have to reach strong agreement on membership, as done in Scatter [Scatter, SOSP 2011].

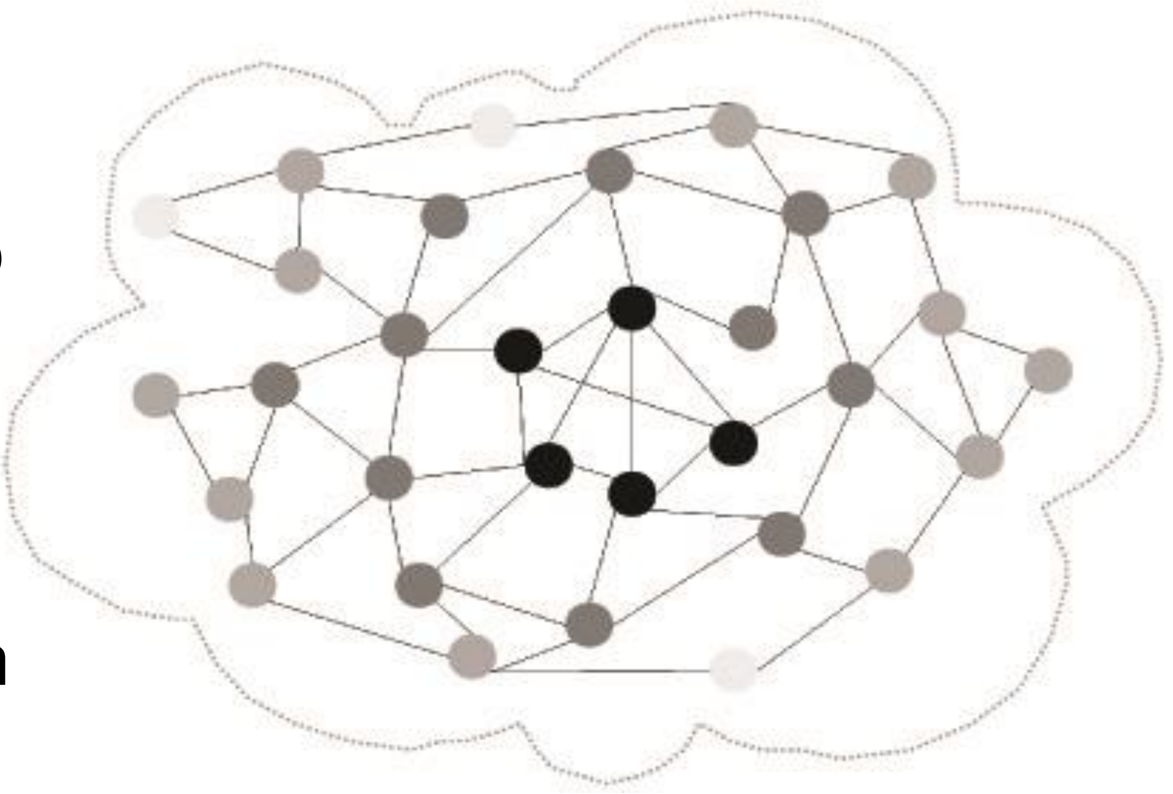


T-MAN

- T-man is a gossip-based protocol framework that can construct an overlay network using a *preference function*
  - The *preference function* orders any set of nodes according to their desirability to be neighbors of a given node
- It’s only a framework
  - You have to decide what the *preference function* is, and then discover what type of topology the preference function constructs, and then analyze its properties, etc.
  - For example, in “Ordered slicing of very large-scale overlay networks”, Jelasity orders nodes into groups, but the search-time to find the leader is O(N).

Gradient Overlay Network

- App-specific *utility function* at every node.
- Nodes gossip to preferentially connect to nodes with higher utility values as close as possible to their own utility value.
- Built by sampling from a peer sampling service
  - E.g., Cyclon
- Eventual convergence proved [Terelius '11]



Preference Function in the Gradient

- Peer p prefers neighbour a over neighbour b if and only if

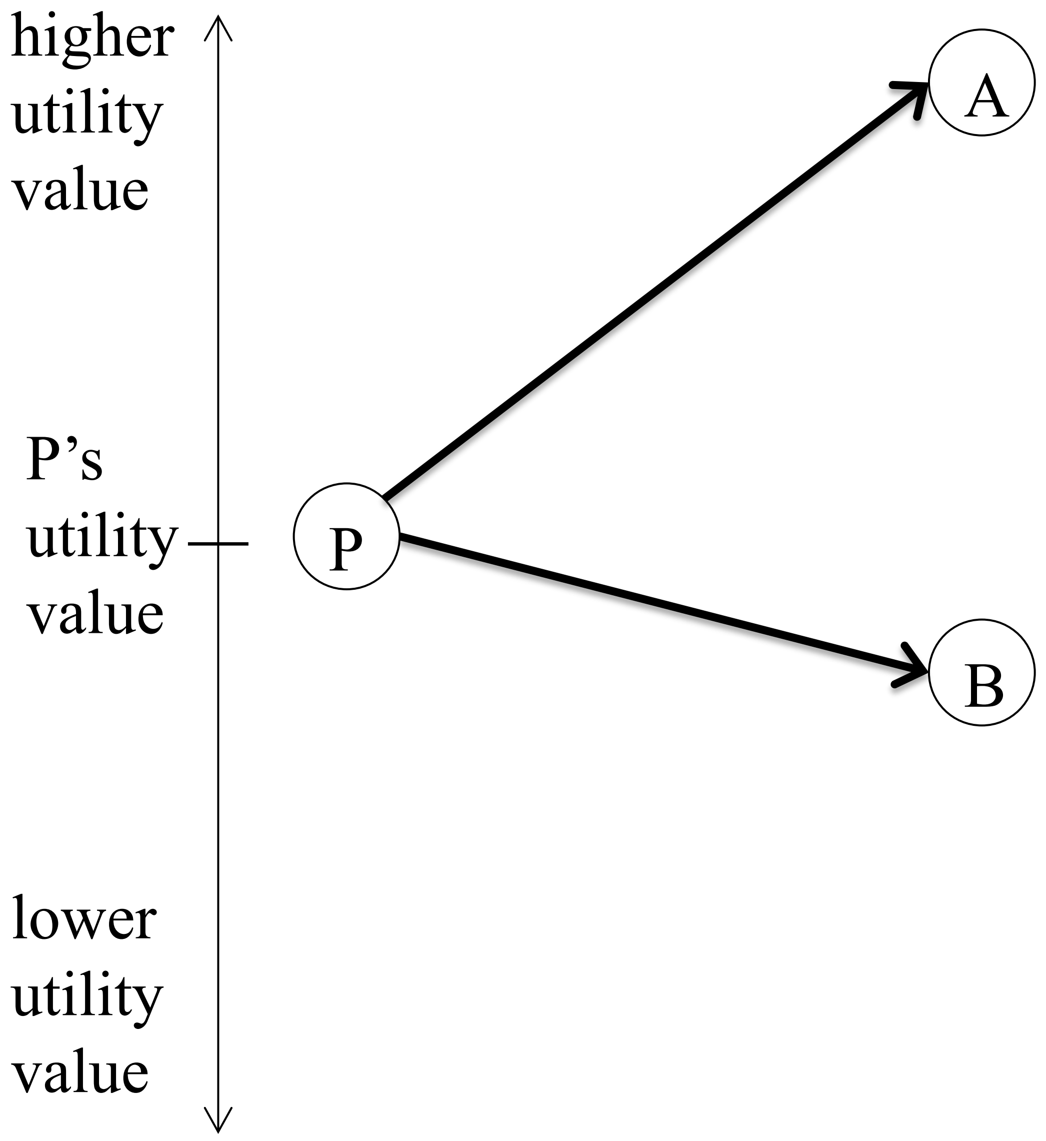
$$\left(U_p(a) > U(p)\right) \wedge \left(U_p(b) < U(p)\right)$$

or

$$\left|U_p(a) - U(p)\right| < \left|U_p(b) - U(p)\right|$$

where U_p(a) and U_p(b) are p's *cached* utility values for neighbours a and b. U(p) is not the cached but the actual utility value of p, as it is local.

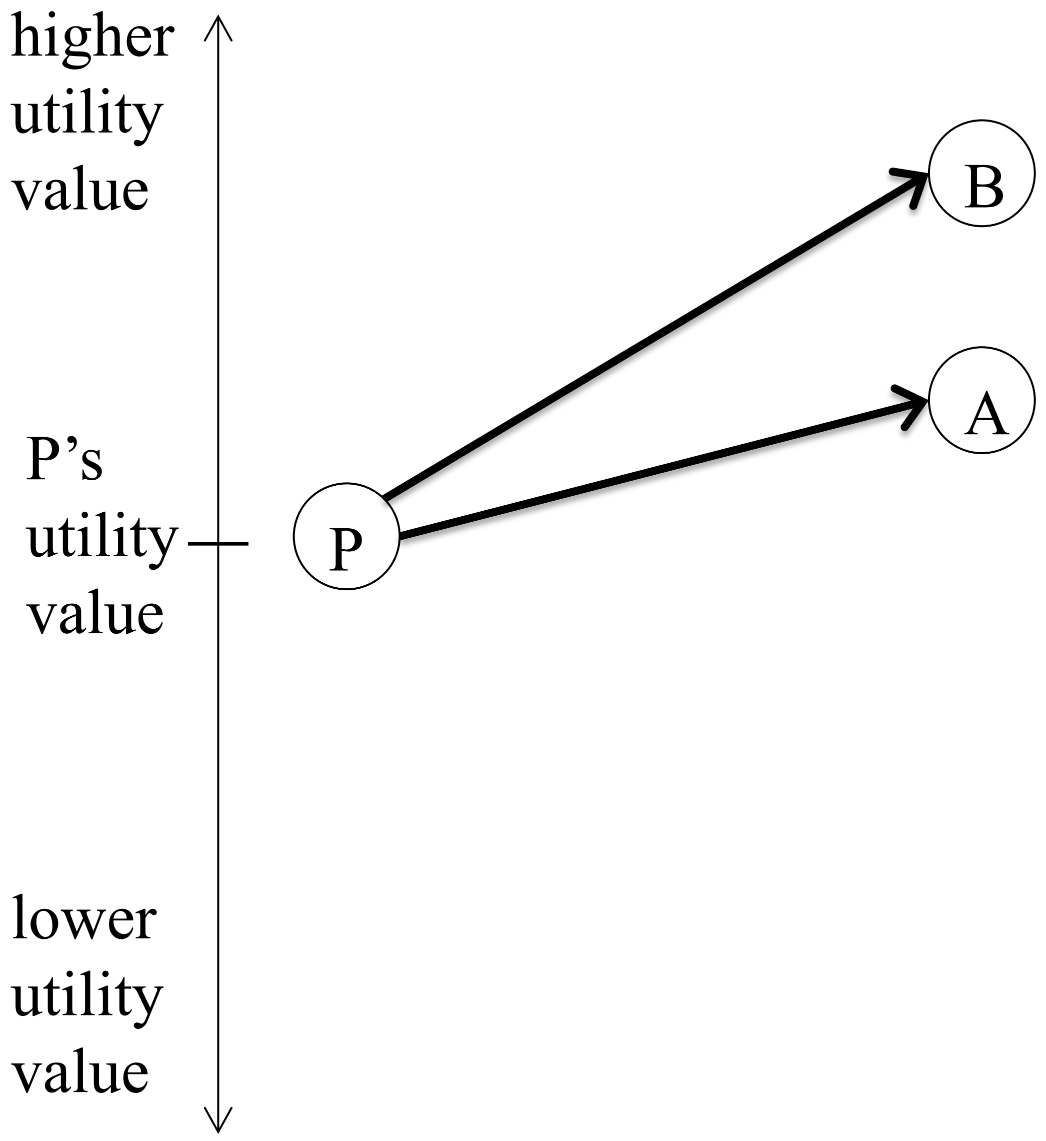
Preference Function in the Gradient (1/3)



Up(B) < U(P) < Up(A)

=> Prefer A

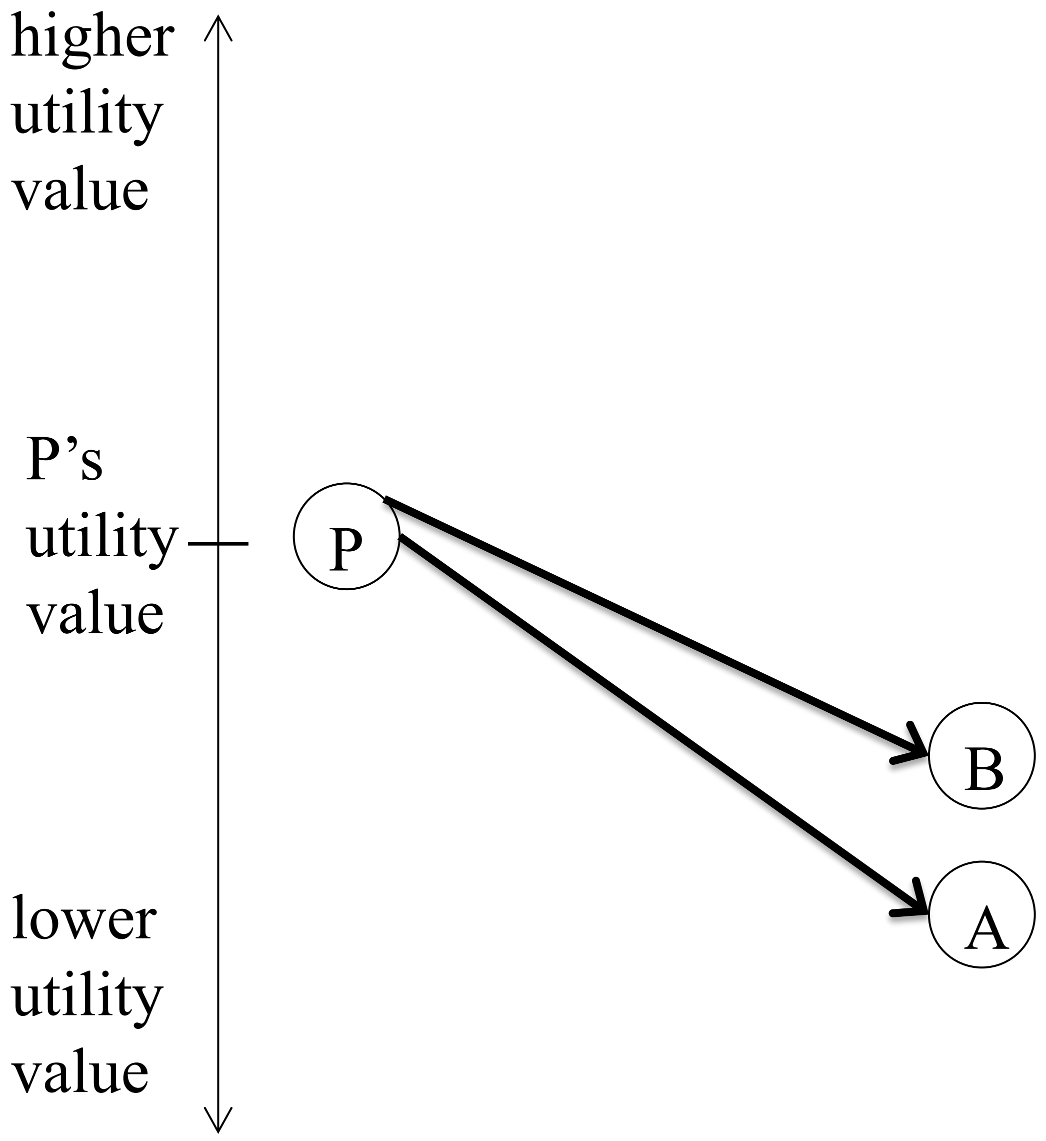
Preference Function in the Gradient (2/3)



$U(P) < U_p(A) < U_p(B)$

=> Prefer A

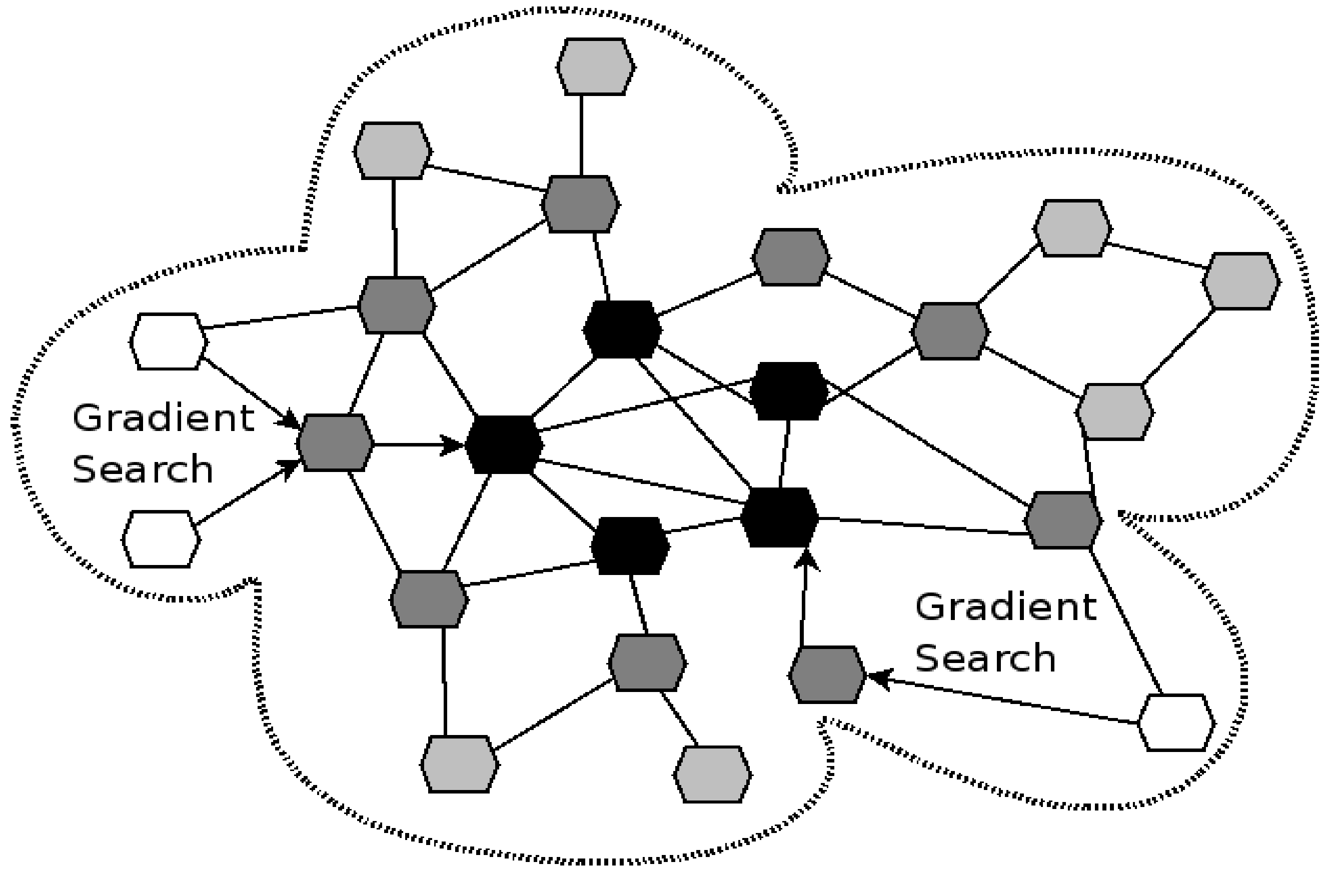
Preference Function in the Gradient (3/3)



Up(A) < Up(B) < U(P)

=> Prefer B

Gradient Overlay Network Example

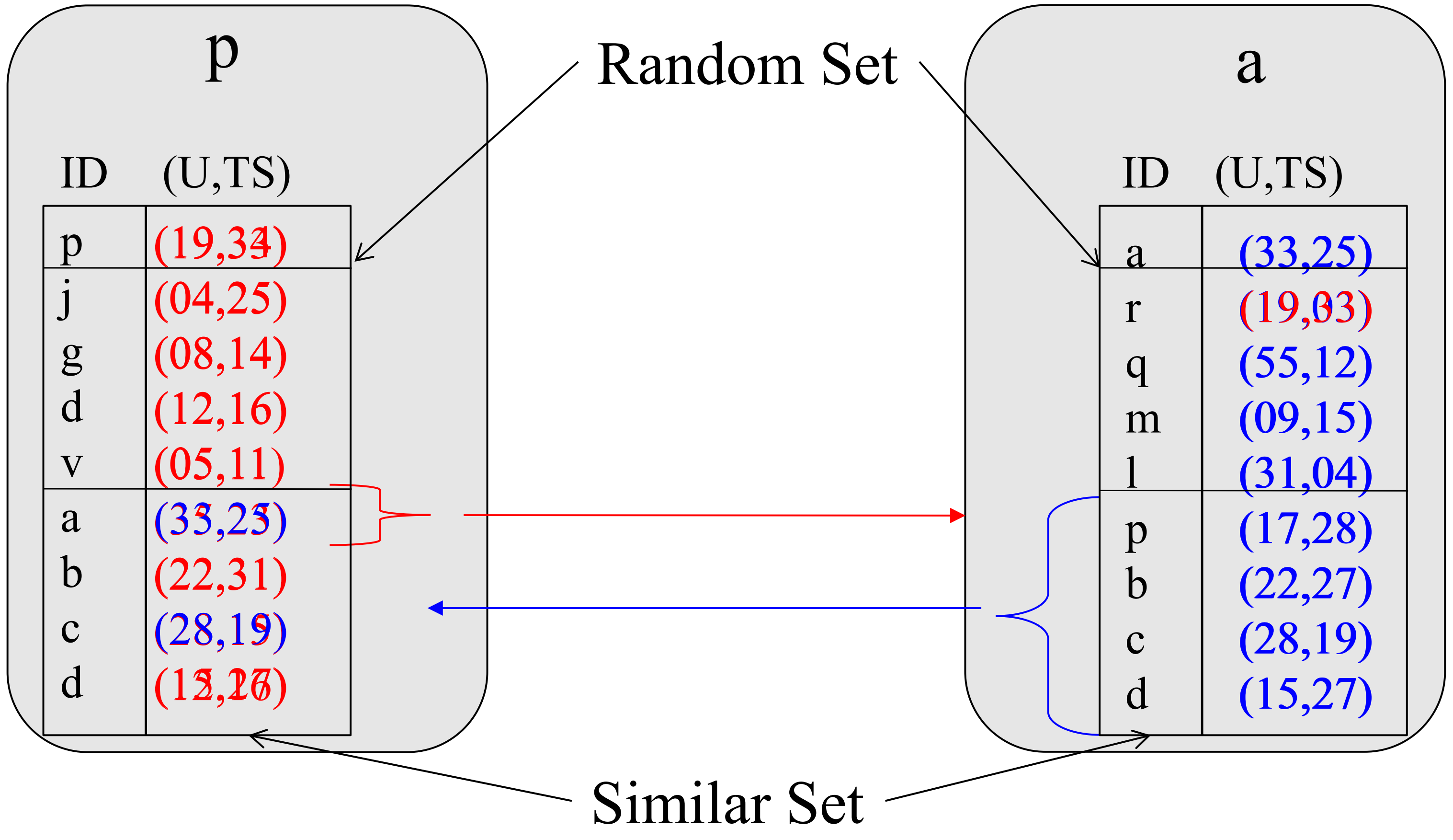


⬢	Highest Utility SPs	⬢	Medium Utility SPs	⬢	Lower Utility SPs
		⬡	Ordinary Peer		

Similar Set: Neighbour Exchange & Merging

1. Peers P selects a similar Neighbour using the Preference Function.

2. Peer P retrieves the similar set from Neighbour a and merges the received set by preserving most recent values in the cache by timestamps.



SelectPeer

- When you select a peer to perform neighbour exchange with, choose better peers with higher probability using the Gradient preference function.
  - This will make the topology converge quickly.
- Selecting *better neighbours* with higher probability can be done using something like the Boltzmann distribution (softmax action selection)
  - A temperature parameter, T, controls the level of *exploration* versus the level of *exploitation.*

Gradient Issues for Leader Selection

- You could use a globally unique identifier as the node’s utility value
  - Nodes would be ordered in the Gradient by their ID

- Almost sure convergence
  - Given a few gossiping cycles, the node with the highest utility value should be at the centre of the Gradient.
  - Assuming no communication failure and no message loss.

- Need to decide on the group of nodes that run the leader election protocol.

Leader Election Group

- Let the node that believes it has the highest utility value start the protocol and define the set of nodes that run the leader election protocol.
- What happens if there is more than one node that believes it has the highest utility value?
- Try to ensure that that the node that believes it has the highest utility value, actually has the highest utility value!
- What happens if the neighbours of the highest utility node in the Gradient change?
  - Do we update the election group?

Broadcast using a Leader

Push-Based Information Dissemination

- One node wants to disseminate some messages to all nodes
- Every node does the following:
  - Buffers every message it receives up to a certain buffer capacity b
  - Forwards the message each time to f neighbours, f called the fanout of the dissemination
  - Neighbours can be selected for forwarding using
    1. Random policy
    2. Gradient descent policy (forward only to neighbours with lower utility values)

Pull-Based Information Dissemination

- One node wants to disseminate to all nodes messages that are ordered sequentially by identifiers
- Every node does the following:
  - keeps the id of the latest message it received.
  - Buffers every message it receives up to a certain buffer capacity b
  - periodically pulls new messages with ids higher than its current id from a neighbour selecting using a policy P
    - E.g., the policy could be a Gradient ascent policy would be to select a neighbour with a higher utility value.
  - replies to any node that requests new messages higher than a given id with those messages in its buffer b with ids higher than the given id

References

- Elections in a Distributed Computing System, H. Garcia-Molina, 1981.
- Distributed Systems , Tenenbaum and van Sten, 2008.
- A Leader Election Algorithm in a Distributed Computing System, Tai Woo Kimo, Eui Hong Kim, Joong Kwon Kim ,Korea Institute of Science and Technology, 1995.
- Jelasity et al. ”T-Man: Gossip-based Overlay Topology Management”, 2007.
- Leader Election in Asynchronous Distributed Systems, Scott D. Stoller, 1999.
- *Exploiting Heterogeneity in Peer-to-Peer Systems using Gradient Topologies, Jan Sacha, 2009*
- Converging an Overlay network to a Gradient Topology, Terelius et al., 2011
- Paxos Commit. Jim Gray. Leslie Lamport, 2004.