

CmpE 275

Leader elections in an asynchronous
overlay network (part 1), **revision 2.2**

Creating agreement – what are the possibilities?

On track

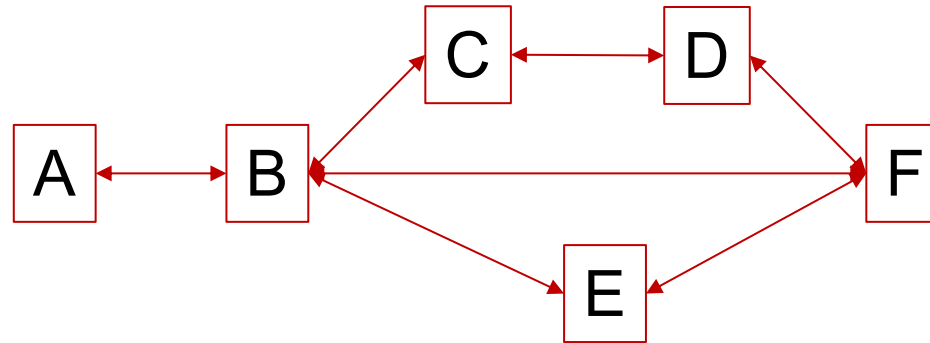
- Agenda (Leader elections, continued in voting)
- Overlay network lessons learned
- Lab 3 Cooperation and Delegation
 - ◆ Building from lab 2, extend to support additional processes to share work and control
- Introduction to Elections
- Key points
 - ◆ What is an overlay network
 - ◆ Resolving SPOF – Dynamic assignments (e.g., a leader)

Lab 3: Cooperation and Delegation

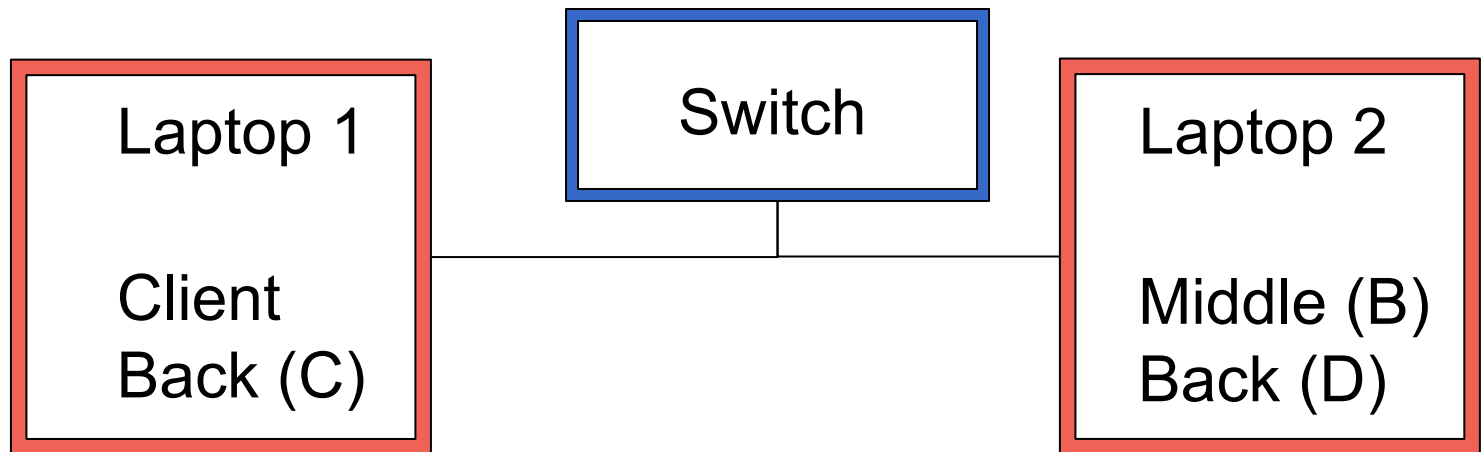
- Construct a basic network – **Client-Middle-Backs (4)**
 - ♦ Create an algorithm to distribute messages to multiple processes (Middle: B, Backs: C, D, E, F)
 - ♦ Design for flexible (not hard code) which process receives the message (endpoint)
 - How is the leader (middle, B) a SPOF?
 - How can Backs provide feedback to the Middle (B)?
 - ♦ Failure detection
 - Modify the relationships between B,C,D,E,F to function as peers. Identify where potential failures can occur?
 - What are your recovery options?
- Advanced
 - ♦ Allow the three processes to determine who are the Middle and Backs
 - ♦ Dynamically add more Backs (G, H, ...)

Lab 3 Cooperation's Logical and Physical Layouts

Overlay



Physical



Delegation design

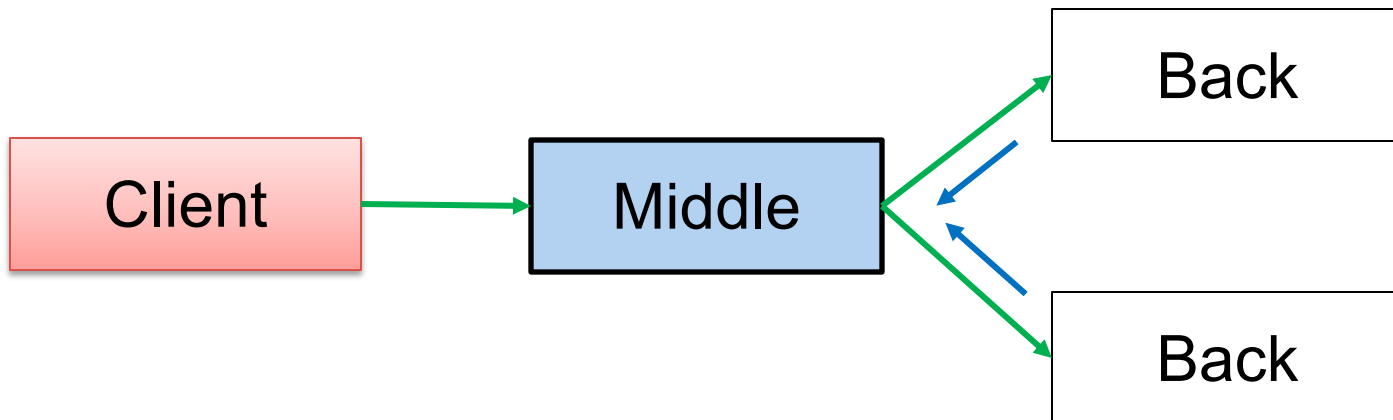
- Develop four or more processes
 - ♦ 1 – Client
 - ♦ 3 – Servers (Middle, Back x 4)

Design

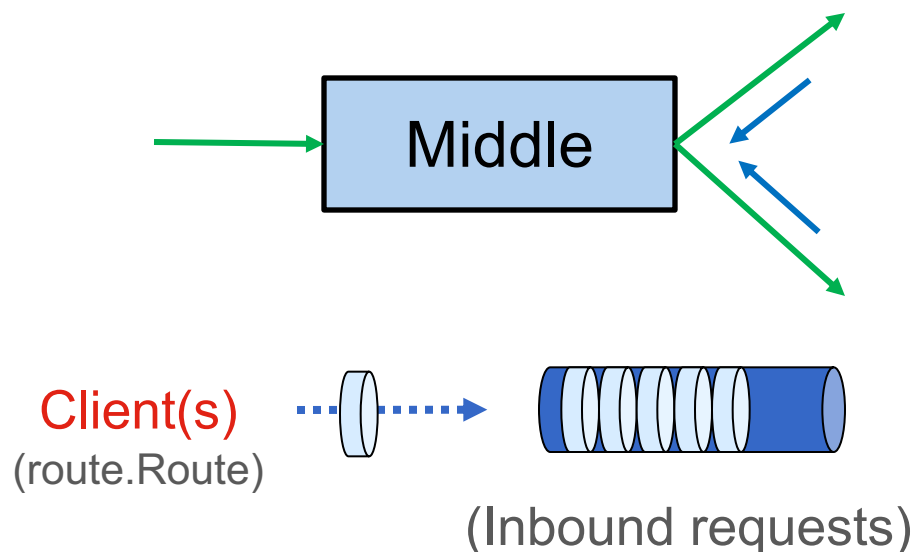
- **gRPC** to allow code to manage resources (memory, threads) and processing (QoS)
- Introduce **queues** in Middle to support inbound and outbound processing
 - ♦ Decouples (temporal) requests – Allow us to manage resources
 - ♦ Consider a queue to support responses from Back(s)

Determining which are clients and servers

- Roles
 - ♦ Client – sends messages
 - ♦ Middle – receives client messages, forwards to Back(s), and receives results from Back(s)
 - ♦ Back(s) – receives messages and replies
- Middle and Backs are both clients and servers
 - ♦ TODO We could allow the Middle to reply to the client



Requests are decoupled from the response

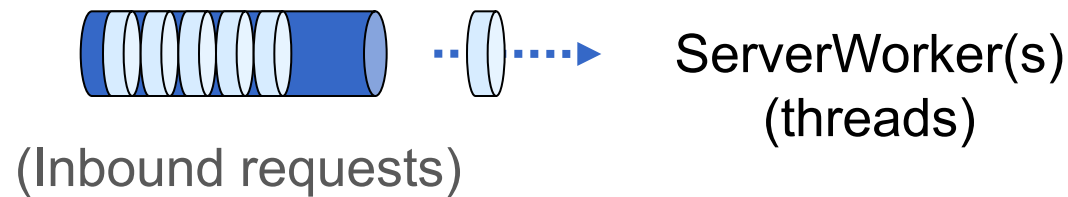


```
public void request(route.Route request, StreamObserver<route.Route>
responseObserver) {
    queue.enqueue(new ServerTask(request, responseObserver));
}
```

Decouples M-to-B request() -

- Manage resources (memory, cycles)
- Reduce latency → increase loads, >> requests (client-to-middle only, overall dt increases)

Works are blocking until a request is enqueued



```
public void run() {  
    while (forever) {  
        try {  
            var task = queue.take();  
            process(task);  
        } catch (Exception e) {}  
    }  
}
```

Apply QoS, and resource
(Back) management

- Circuit-Breaker
- Apply fairness algo.
- Re-route destination
(spatial decoupling)

Besides our decoupled, in-memory design, what other options exist?

1. Conserve memory, use a database in place of a queue
 - ◆ Increases cycles and delay (Transactions)
 - ◆ Introduces another component (DBMS)
 - CONs: Maintenance cost increases, licensing fees, and more compute resources
 - PROs: Survive server crashes, QoS queries can use SQL
2. Remove indirection (Middle)
 - ◆ Client discovers and communicates directly to Backs
 - ◆ What are the drawbacks of this approach?

Let's focus on the relationship between the Middle and Backs

- Discovery
 - ◆ Fixed vs Dynamic
 - ◆ Failure and recovery
 - ◆ Capacity escalation
- Sharing work
 - ◆ Overloading vs Starvation
 - ◆ Coordination and orchestration

Overlay network

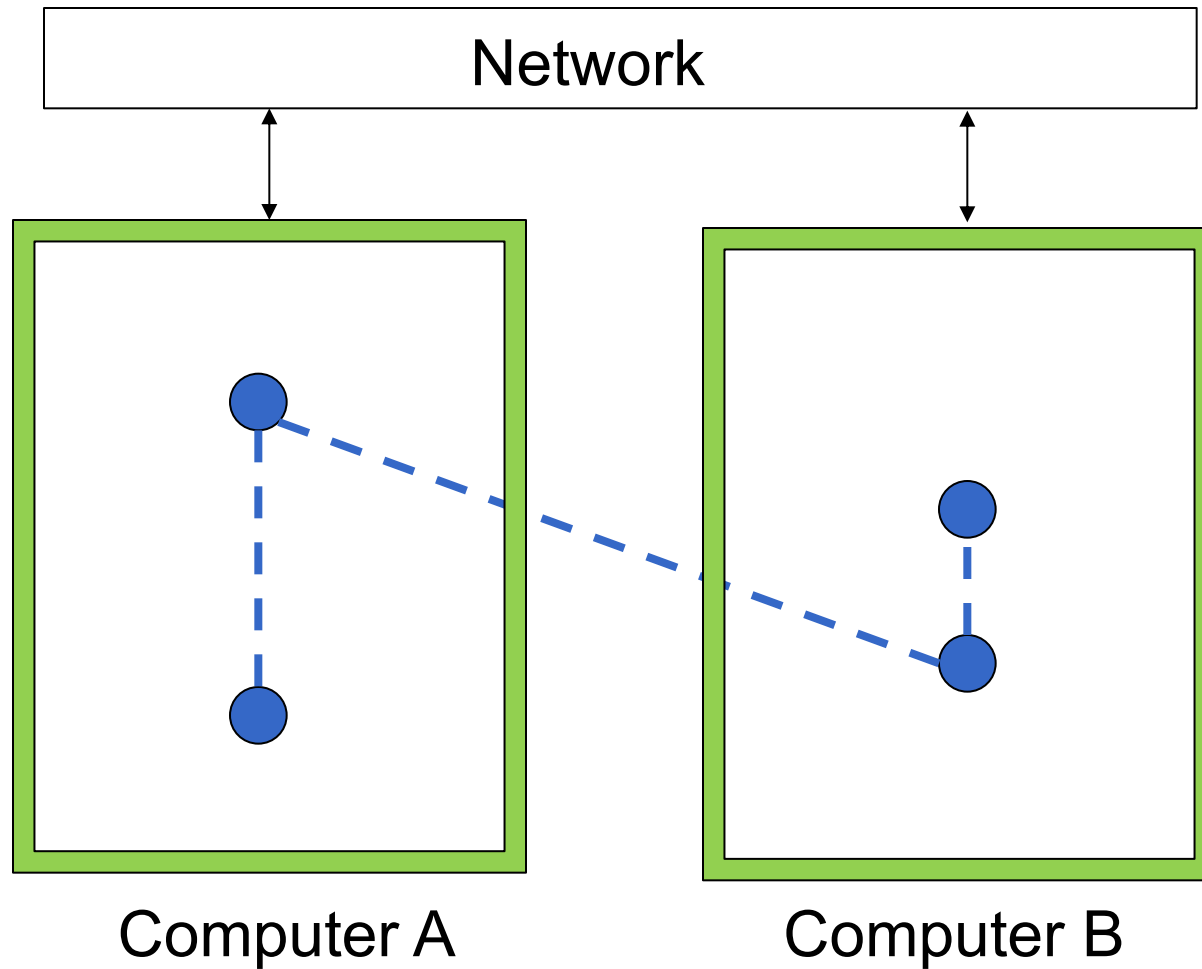
- **Physical network** – how the network (nodes and connections) exist in the real world
- **Overlay network** – a logical (programmatic) representation of the network
- Examples
 - ◆ Hash
 - E.g., ordered by name – alphabetically (or by IP, key)
 - ◆ Divided into domains/groups (Finance, Web, Development)
 - ◆ Ring, star, gridded networks

More on this later

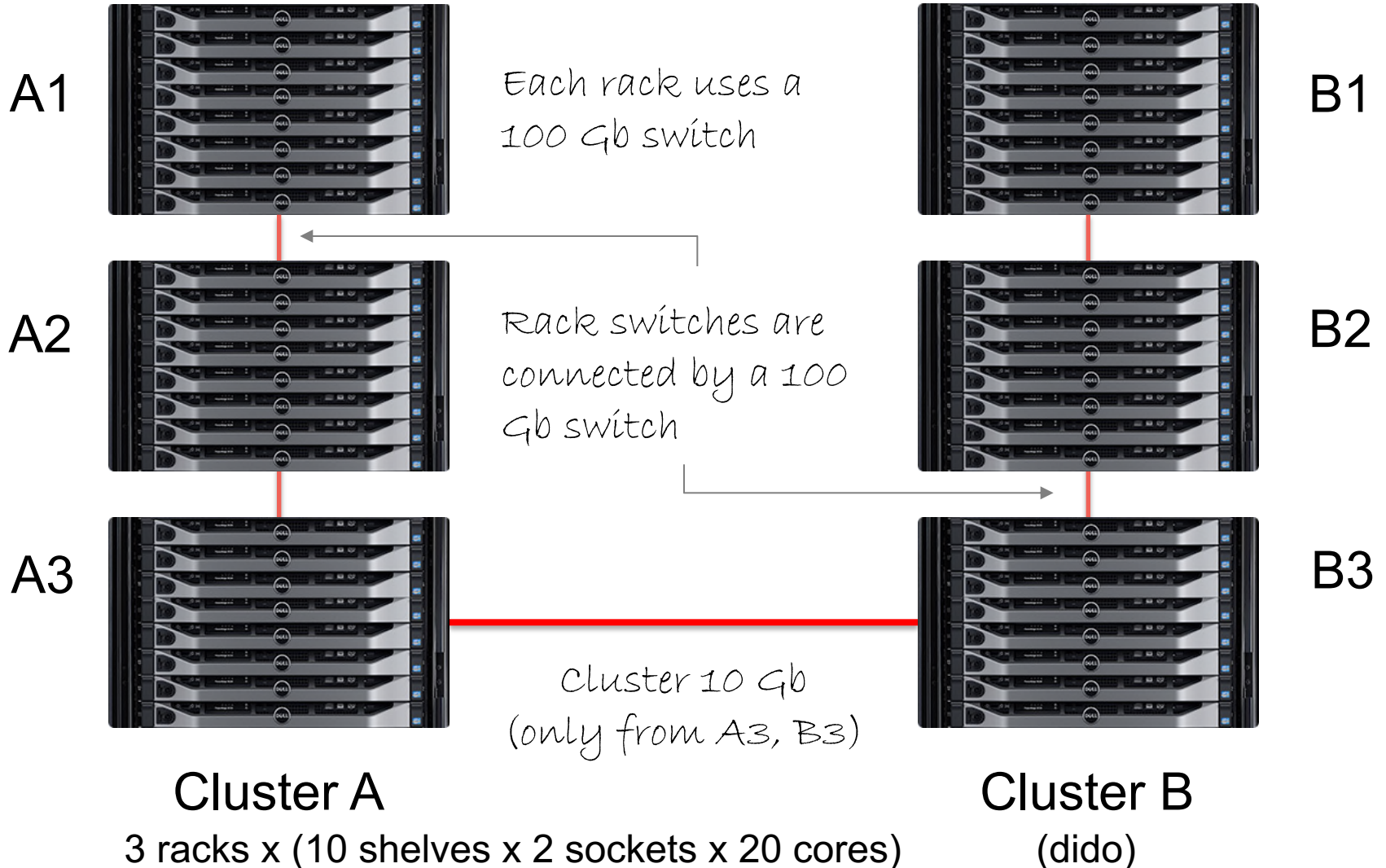
Connecting processes is an overlay network

- A collection of processes connected (socket/port) to form client–server relationships (nodes and edges).
- Organization of these processes into logical layout is creating an overlay network (topology)
- Uses
 - ◆ File sharing (parallel downloading)
 - ◆ Routing (adaptive, latency detection, failure, and discovery)
 - ◆ Decision making (leaders)
- Example overlay networks
 - ◆ Ring
 - ◆ Well formed/connected (clique)
 - ◆ Spanning Tree
 - ◆ Distributed Hash Tables (DHT)

Physical vs Overlay



Example: Applying overlay strategies to a physical network



Example: Placement of our processes is key. Why?



Cluster A

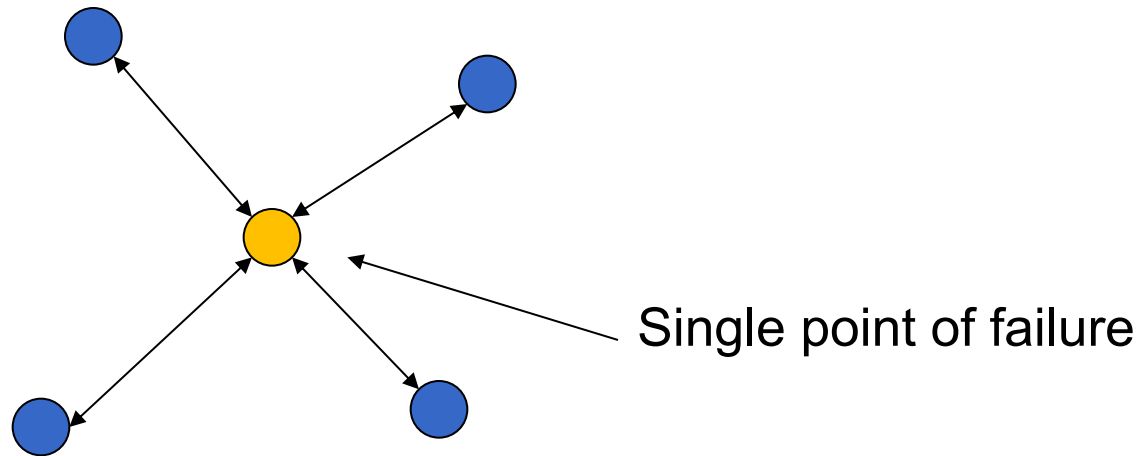
- Locality – communication (connected in series)
 - ◆ Same Shelf
 - Utilize shared memory, no external switching
 - ◆ Within a rack (100g)
 - ◆ Adjacent Racks (100g)
 - e.g., A1 to A2
 - ◆ Between Clusters (10g)

Common overlay designs

- Recap: An overlay network is any organization that is used by a distributed system. This can include:
 - ◆ Functional decomposition
 - ◆ Role-based (public, private)
 - ◆ Subscription or need based
- Examine several common organizations (hub, tree, ring, graph)

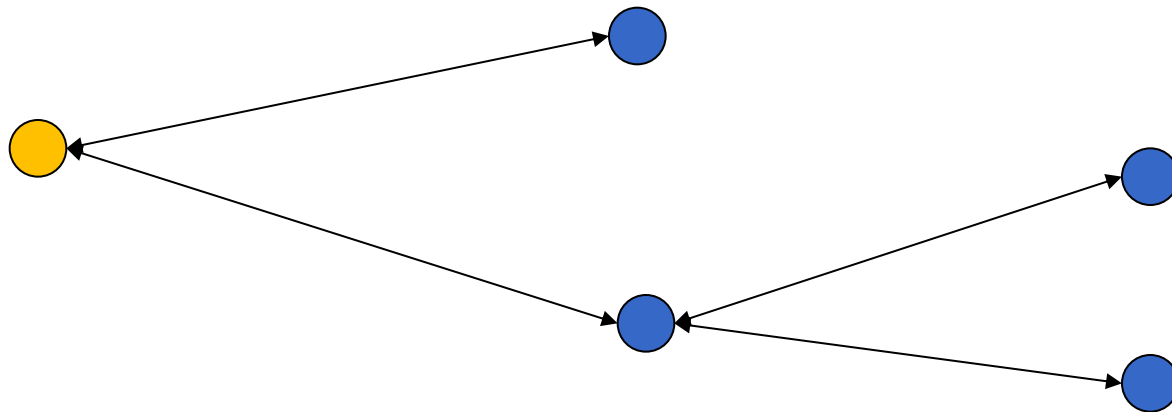
Overlay models: Spoke–Hub

- Each node (spoke) directly connect to the leader (hub)
- Least recoverable from a failure (hub)



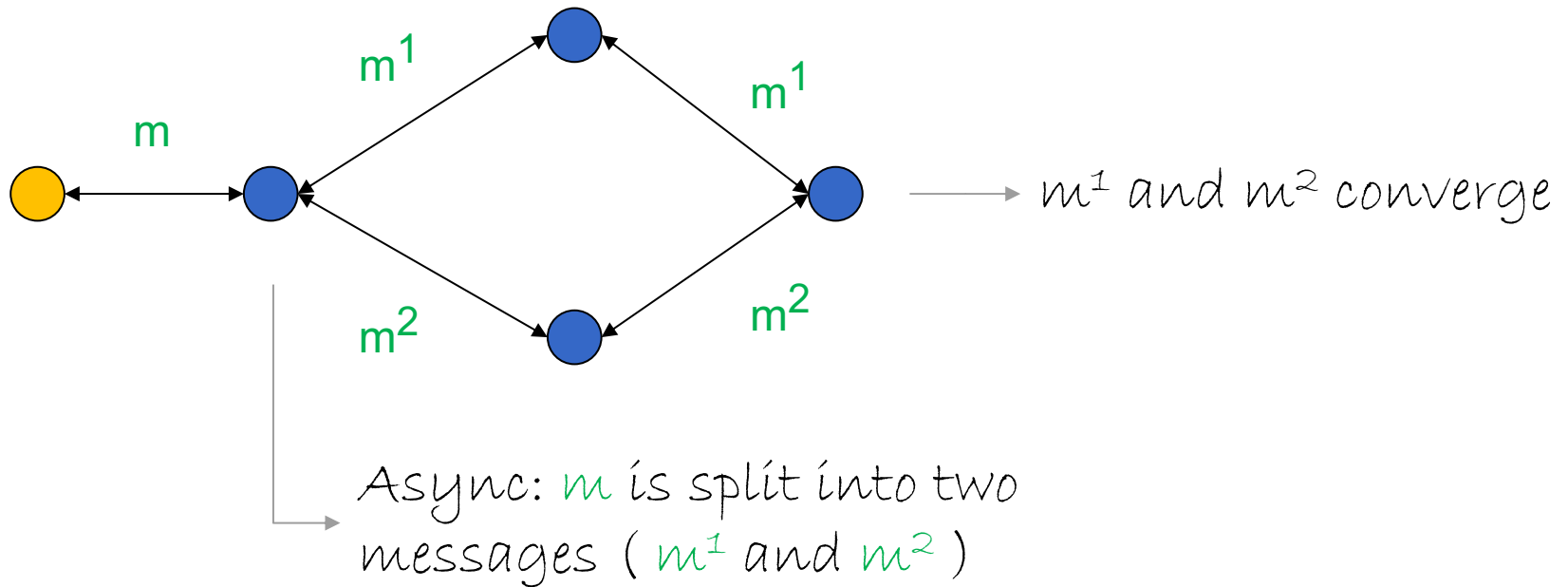
Overlay models: Tree

- Nested traversal
- Can provide search, gather-scatter patterns
- Sub-setting (grouping) of nodes



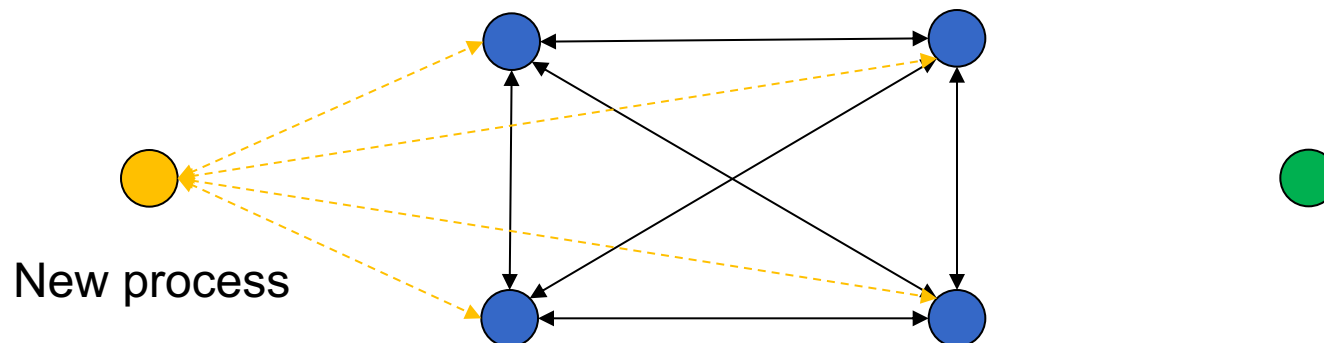
Overlay models: Cycles

- Cycles – more than one path to an endpoint
 - ♦ E.g., a diamond



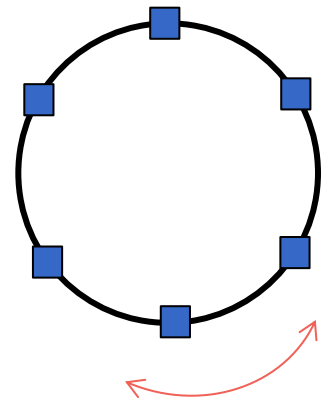
Overlay models: Well-Connected (Clique)

- Fastest traversal (direct connection, $N-1$)
- Scaling issues as $N \gg$
 - ♦ Memory for tracking table grows linearly as $N \gg$
- Network updates are not isolated
 - ♦ Sensitivity to joins/departures
 - ♦ This can be a big issue with highly volatile memberships (thundering herd problem)



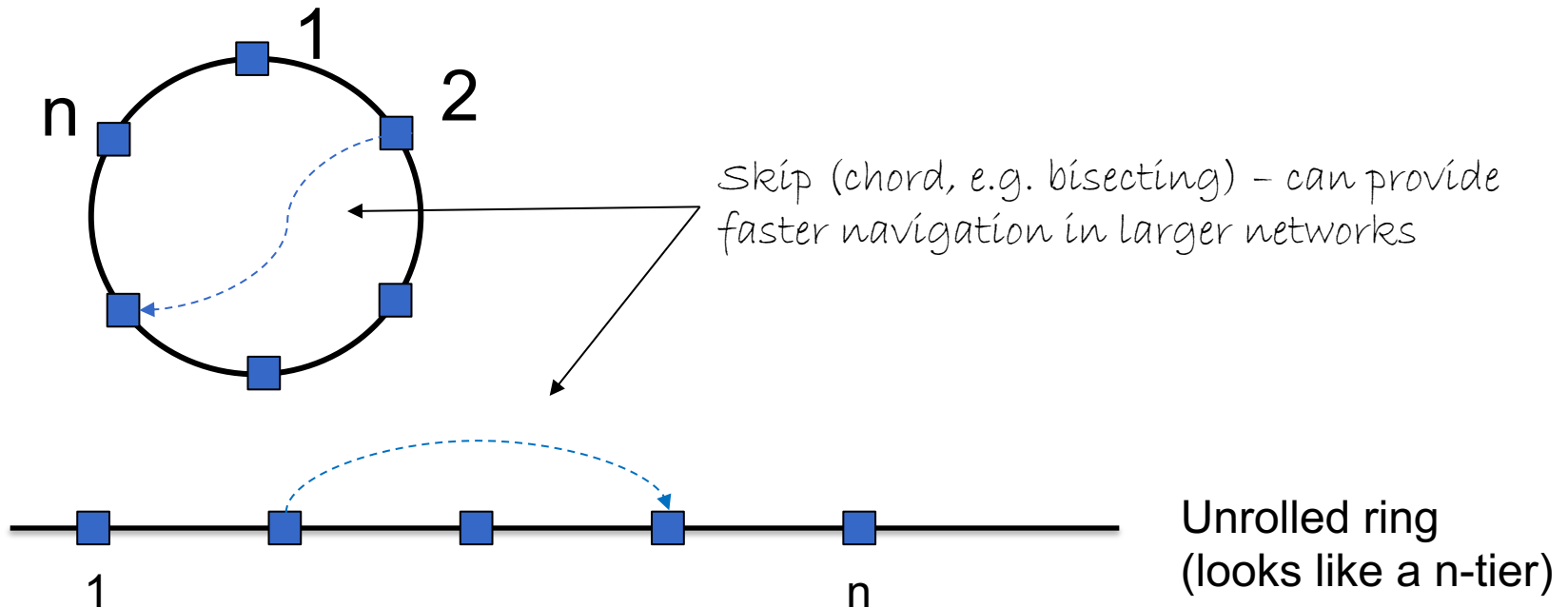
Overlay: Ring Networks

- An overlay network where nodes are arranged in (bi) **unidirectional linked chain (ring)**
 - ♦ Requires ordered organization, usually monotonically increasing (n + 1) number/ID
 - ♦ Or IDs are generated with a hash function, $F(\text{node})$ where the ring represents the range of 0 to 2^{255}
- Messages are passed from node to node, returning to the originator
- Latency increases as $N \gg$
 - ♦ Chord or other hop algo. Can reduce latency



Overlay models: ring

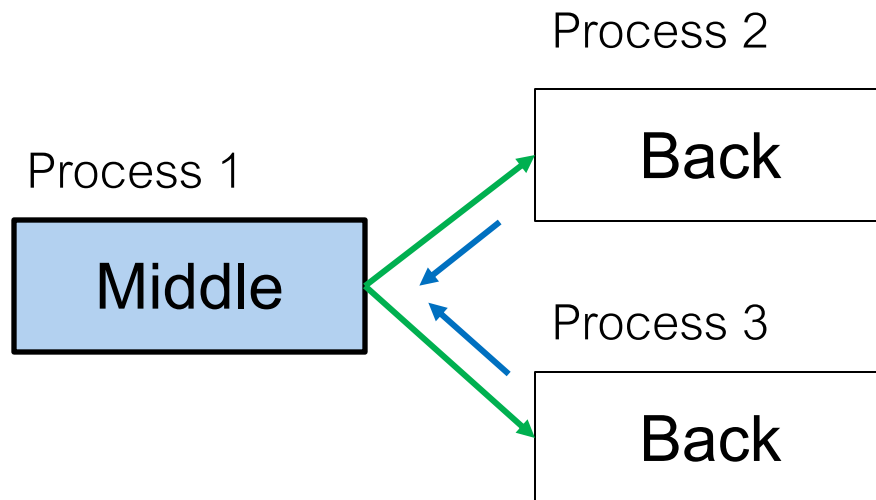
- A linked list (unidirectional, bidirectional, skipping (chords))
- Traversal to another node is not centralized (spoke) → More hops to reach nodes



Elections: Creating leaders

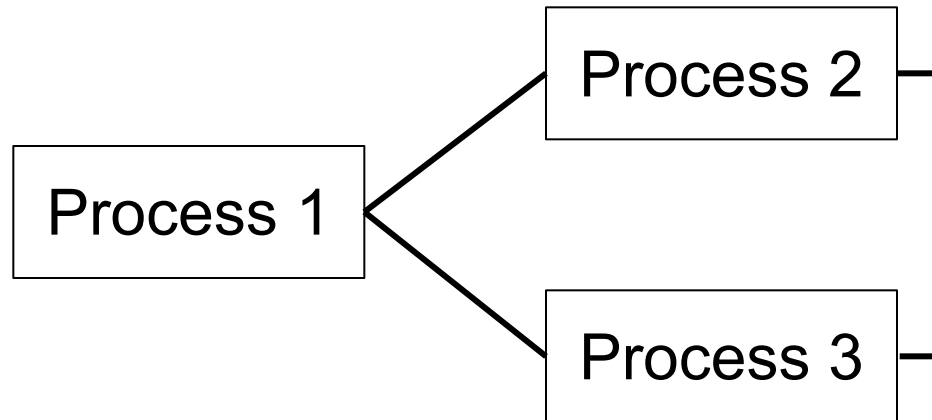
- Reduce Single Point of Failures (SPOFs)
- Decision making
- Data sources and sinks
 - ♦ Examples include dedicated compute resources, data/information, communication, I/O devices, a queuing system
- Leaderless coordination: Peers and Baton passing algorithms

How did process 1 determine it is the leader?



When starting our example processes (1, 2, and 3)

Question: Who is the Middle?



- Fixed Roles
 1. Assign roles at startup (args), configuration file
 2. Unique code bases
- Dynamic Roles
 - ♦ First one wins (Middle)
 - ♦ Negotiate ← Elections

Election algorithms philosophy on success (failure)

- Deterministic
 - ♦ Consistent results, can be calculated
 - ♦ Examples: LCR, HS, Bully, FloodMax
- Non-deterministic
 - ♦ Outcome not the same each time (element of randomness)
 - ♦ Examples: Raft Algo, FloodMax (w/ timeout)

Leader election using a ring network closely aligns with fixed approaches

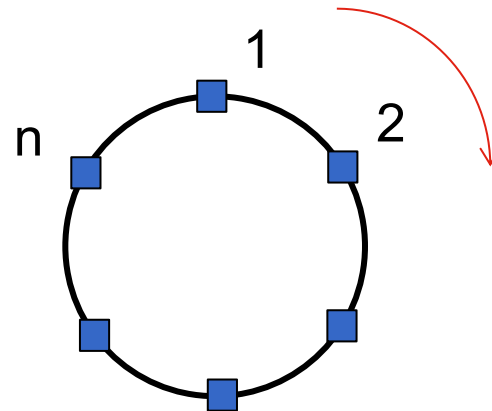
- Origins
 - ◆ Token ring networks where a message travels the network and ends with the sender
- Ring network (overlay)
 - ◆ Nodes arranged with monotonically increasing IDs ($n+1$)
 - ◆ ID as a generated hash, $F(\text{node})$
 - ◆ Optionally
 - The ring can be unidirectional or bidirectional

Leader election (cont)

- Basic steps

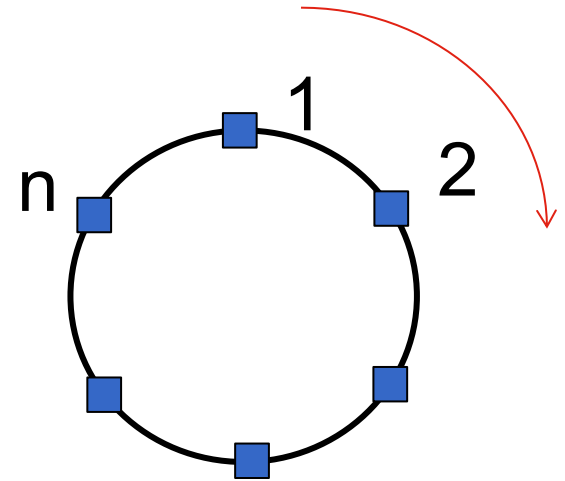
- ♦ One process will output (declare) they are the leader (for instance node 1)
- ♦ Declaration is then forwarded to the next node
- ♦ If the next node does not agree, the message is not forwarded
- ♦ If Node 1 receives it's own request, it was successful in declaring itself the leader

$$F(n_{i+1}) > F(n_i)$$



Le Lann, Chang and Roberts (LCR)

- Setup
 - ♦ (Bi) Unidirectional
 - ♦ Unknown number of nodes
 - ♦ All nodes use the same algorithm
- Complexity
 - ♦ Time: $O(n)$
 - ♦ Communication: $O(n^2)$
 - assuming a request and reply travels all nodes
- Variations
 - ♦ Hirschberg–Sinclair (HS), $O(n \log n)$
 - Bidirectional
 - Phases: node i sends message $2 \cdot p$ hops out
 - Same comparison as LCR



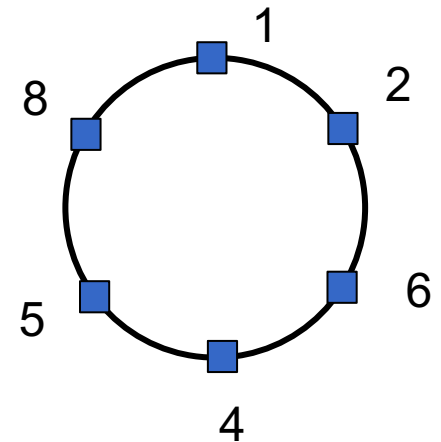
Le Lang Chang Roberts (LCR)

- Cycles are not supported
- Works for both synchronous and asynchronous communication
- Each node as a unique ID (UID) used for comparisons
- Ordered network (e.g., ring) of increasing or decreasing by UIDs
- Unidirectional though bidirectional helps in detection of a partitioned network and reduces complexity (see HS)
- Algorithm (**for increasing UIDs**)
 - ♦ Each node sends its UID into the network

```
if  $UID_{rec} > UID_{self}$  then forward  $UID_{rec}$   
else if  $UID_{rec} == UID_{self}$  then declare self as the leader  
else discard message
```

LCR decision code

- Algorithm (comparison-based, max ID wins)
 - ♦ Each process sends their ID around the ring
 - ♦ On receive(msg)
 - if $\text{this.ID} < \text{msg.ID}$ then send msg to the next node
 - if $\text{this.ID} > \text{msg.ID}$ then discard message
 - if $\text{this.ID} == \text{msg.ID}$ then declare self as the leader

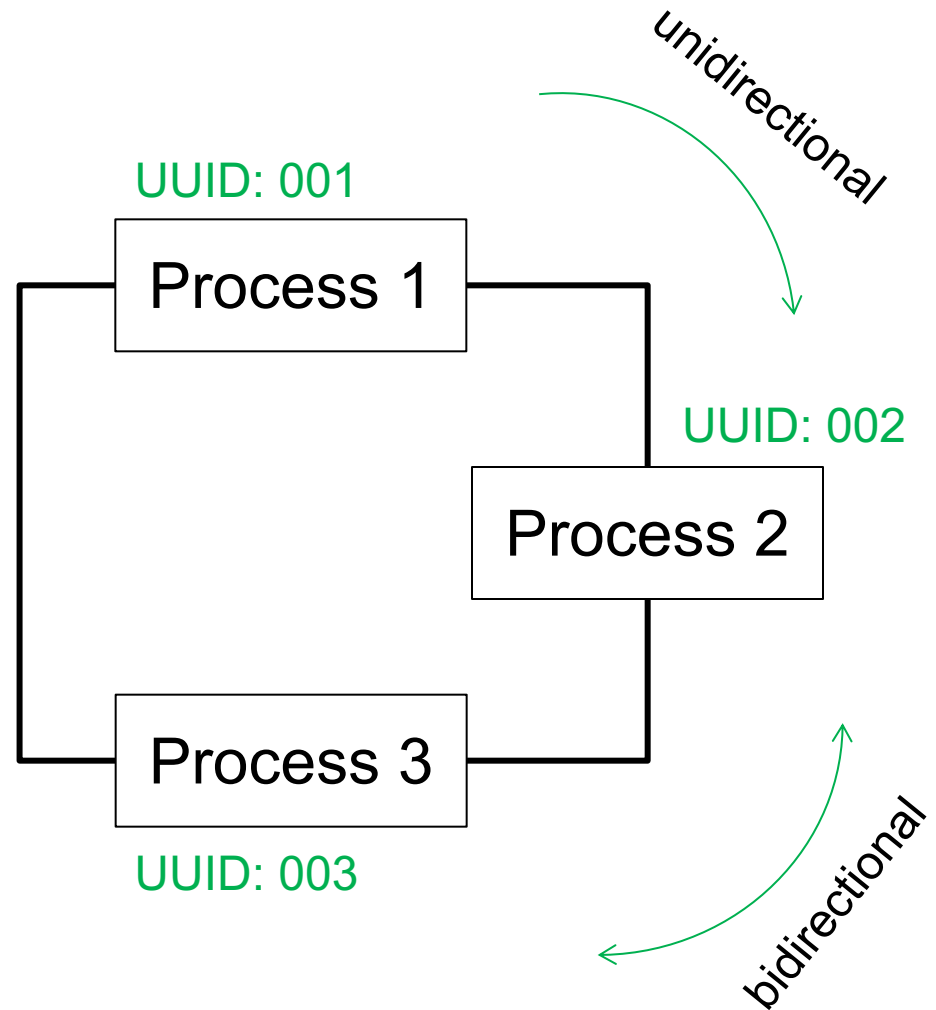


LCR

- Number of messages is $O(n^2)$ – somewhat high
 - ♦ Every node sends to every node
- Options to reduce communication:
 - ♦ Halting – the algorithm terminates with a message from the leader declaring the election is over
 - ♦ Breaking symmetry – The algorithm is generally deterministic and repeatable

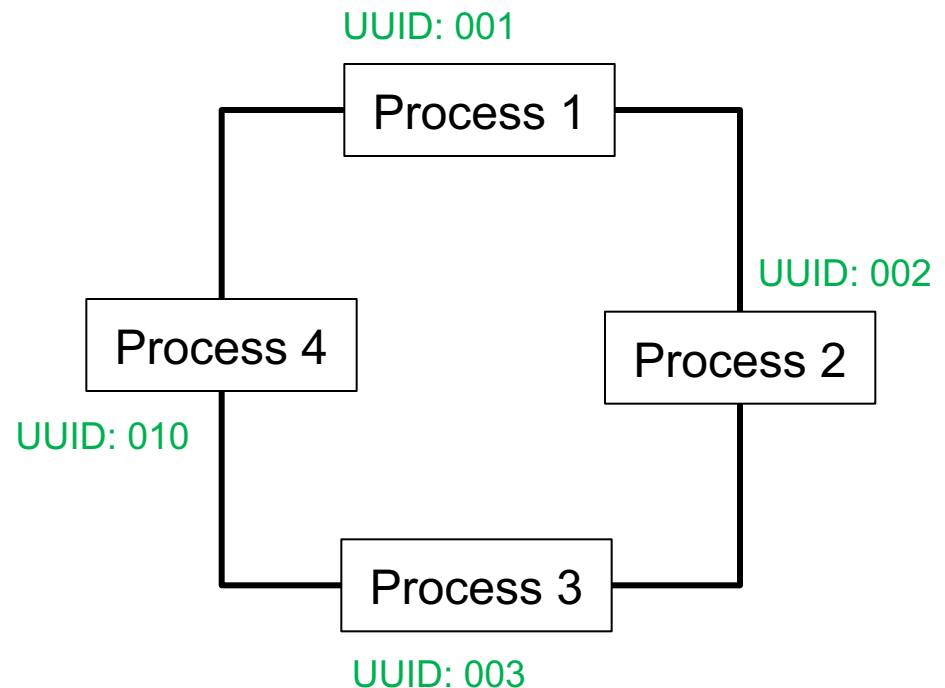
LCR for our Middle and Backs

- Each process has a unique ID (UUID)
 - ♦ Assigned
- Discovery:
 - ♦ Connection list
 - ♦ Dynamic
 - ♦ Combination
- Uni or bi-directional



Thinking more about our Middle and Backs

- Can we use PIDs as UUIDs?
- Are processes 1, 2, and 3 an overlay network?
- If we introduce a fourth process (4), how does this affect an LCR election?
- How are roles assigned?
- What can this fail?
- Mitigation of failure?

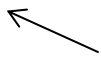


Hirschberg and Sinclair (HS)

- Complexity reduced to $O(n \log n)$ compared to LCR
- Bidirectional communication
- Works for both synchronous and asynchronous communication
- Algorithm
 - ♦ Node 'i' sends (outbound) a message in phases (p) to its adjacent nodes w/ max hops 'h', where h is 2^p
 - ♦ Node 'j' receives message and discards the message if $UID_j < UID_i$ else forwards the message up to a distance of h
 - ♦ Once max hops is reached, return (inbound) the message. Inbound messages are not blocked/discarded
 - ♦ If a node receives its UID in an outbound message, it declares itself as the leader

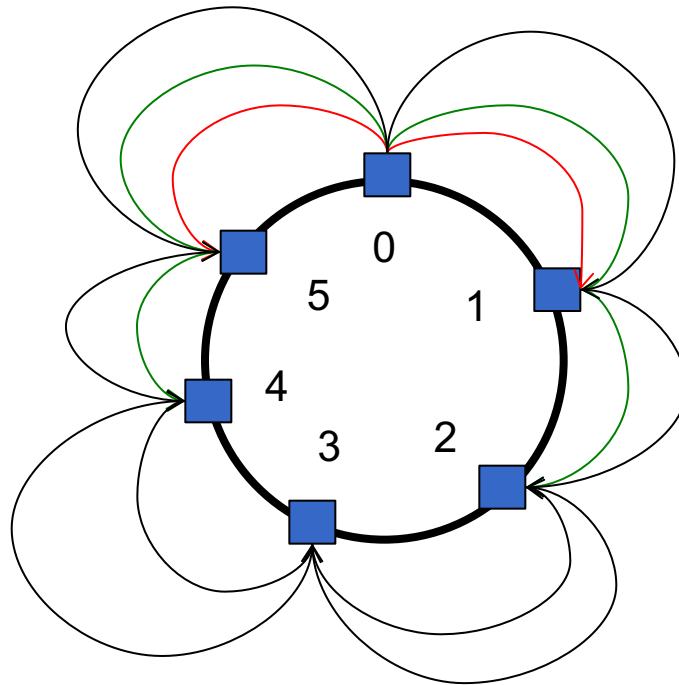
HS message travels 2^p hops

- The HS requires the message to carry additional routing metadata to track
 - ♦ Max hops (distance from sender)
 - ♦ Outbound or inbound passing

```
Message {  
    int maxHops;  
    int hopsRemaining;  
    boolean outbound;  
    byte[] payload;  UUID of sender  
}
```

HS traveling illustration

(3 phases to cover a 6 node ring)



Node 0's messages

$p=0 \{0,1,5\}$

$p=1 \{0,1,4,5\}$

$p=2 \{0,1,2,3,4,5\}$

Peterson Leader Algorithm

- Unidirectional communication in a ring network (asynch and synch)
- communication complexity of $O(n \log n)$. $2n((\log n)+1)$ messages sent which is better than HS which has a $8n(\log n)+1$ → PL a constant of 2 vs. HS constant of 8
- A node is either 'active' or 'relay'. All nodes start as 'active'
- Algorithm
 - ♦ Phases (at most $\log n$ phases)
 - ♦ Node sends max UID two active node hops (itself initially) clockwise*. Note relay nodes are not considered a hop.
 - ♦ Each node compares its UID with the two UIDs it receives. It stores the max UID for subsequent phases. If $UID_{self} < UID_{received}$, the node declares itself as a 'relay' node
 - ♦ When a node receives its own UID, it declares itself as the leader

**clockwise ('left') is a reference when depicting the network as a ring (similar to slide 24)*

FloodMax

- Arbitrary graph (of a known width)
- Nodes have UUIDs (like we saw with the LCR and HS)
- Synchronous communication. Additional work is needed to track rounds in an asynchronous system
- Algorithm
 - ♦ Each node knows the number of nodes and the diameter (max distance a node in the graph must travel, **longest path**, to reach another node == max hops of a graph). Max distance can be a calculated upper bound if the actual is not known.
 - ♦ Each node keeps the known largest UUID it has received
 - ♦ Each round it sends this value (UUID). A round is effectively (n-1 messages received)
 - ♦ After diameter rounds, the node which receives its own UUID is the leader

Floodmax in an asynchronous system

- In a synchronous mode, the nodes know to wait for n nodes before broadcasting the next round of messages.
- Asynchronous – each node must wait until all messages are received before processing messages

General network solution using Flood Max

- Setup
 - ♦ Network graph (all nodes are reachable and connected)
 - ♦ Diameter of graph known (max number of vertices/edges between two nodes)
 - ♦ All nodes will identify themselves
- Algorithm (max ID wins)
 - ♦ Each node tracks the max ID it has seen (at t_0 , its own ID)
 - ♦ A round: each node sends the max ID to every outgoing neighbor (node)
 - ♦ At 'graph diameter' rounds, the node with max ID equal to its own ID, declares itself as the leader
- Complexity
 - ♦ Time: $O(\text{diameter})$
 - ♦ Communication: $O(\text{diameter} * \text{messages-per-round})$

Decentralized (or peer) design

- Consider a design where the network can function without a leader (or a strictly enforced leader)
 - ♦ What advantages would this provide?
 - ♦ If a coordinator is needed within a process, what design would you use?

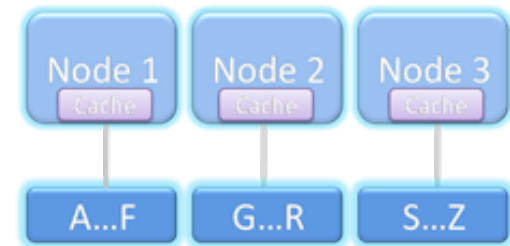
Next week: PAXOS and RAFT

- Both PAXOS and RAFT are approaches for coordinated consensus amongst multiple processes
 - ♦ RAFT includes an election capability
- However, elections (leader assignment) can be used for operations other than replication (i.e. Raft)

Application of overlay networks in NoSQL (Replication)

Separating data into independent data presents the potential for huge gains in storage capacity

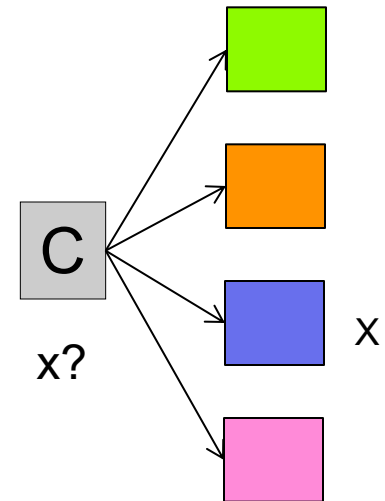
- **Shared-nothing (SN)** persistence
 - ◆ Early papers (1986, Stonebraker) presented the idea of SN though it did not gain traction within the industry until the popularity of the web created a need for very large data repositories
 - ◆ Data distributed across multiple servers (nodes) do not depend on other nodes for data
 - Partitioning allows processors/nodes to service subsets of data thus, performance and scaling is achieved by adding servers/nodes
 - Popular with data warehousing, web sites (most well known: google)
 - Provides near-infinite scalability (of course nodes must be able to find participating nodes)



<http://www.benstopford.com/2009/11/24/understanding-the-shared-nothing-architecture/>

Using a consistent hash for distributed storage lookup

- Consider four servers storing data...
- How do you determine which server has the data?
 - ♦ **Gossip** – ask all four?
 - Expensive operation and scaling is $O(n)$
 - ♦ **Table** – lookup where the data is stored?
 - Adds an extra operation (TX), a SPOF, and does not scale (limited by table)
 - ♦ **Natural organization?**
 - Doesn't ensure balance – could create hot spots
 - ♦ **Algorithmically?**
 - Calculated \rightarrow fast and consistent, no SPOF
 - All code must share the same hash algorithm
 - This is consistent hashing



Consistent hashing using virtual ranges to create consistent bucket sizes (physical nodes << virtual/vnode)

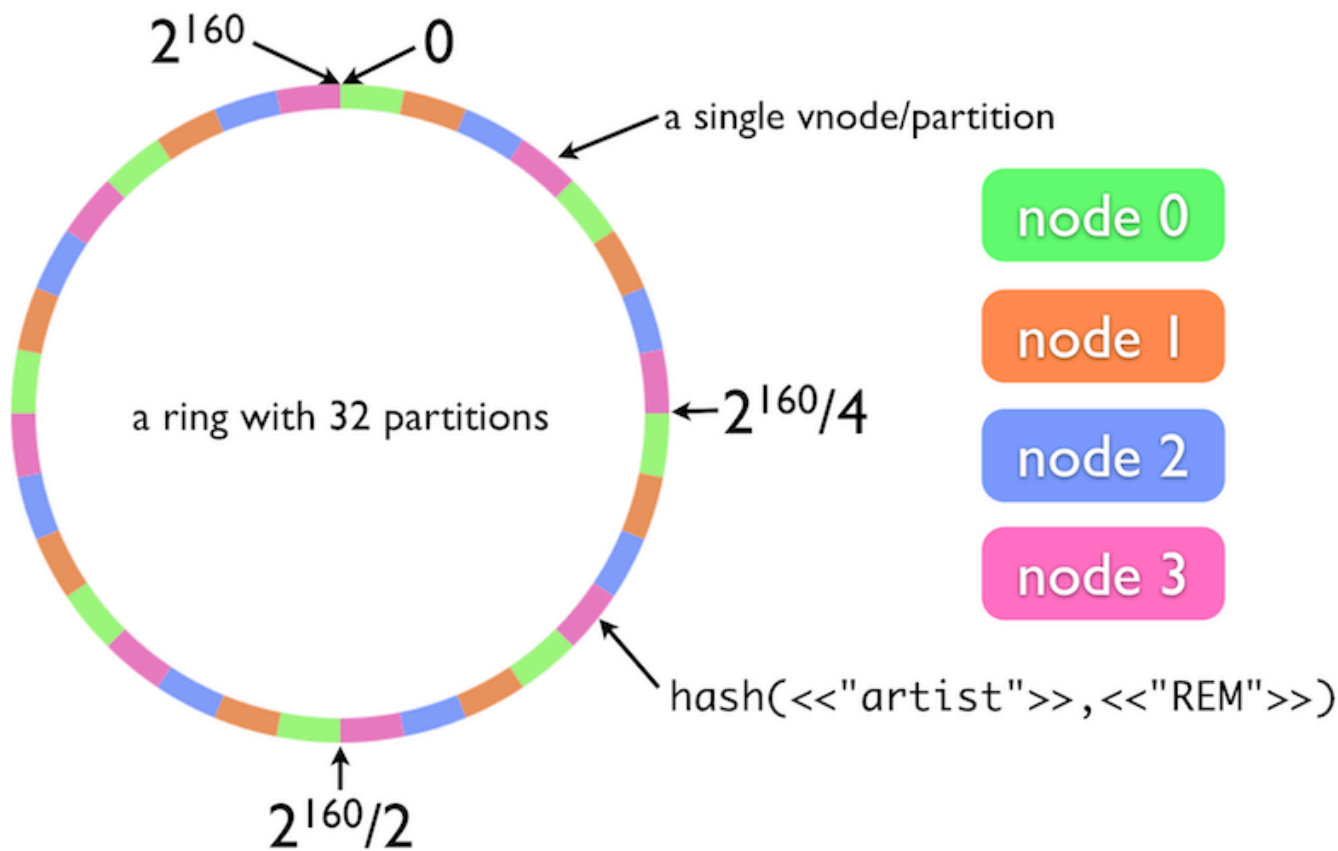


Image from basho.com

Replication with quorums

- N,W,R
 - ♦ Quorum approach to consistency
 - ♦ Defines Replication, failover, and agreement
 - ♦ Performance good
 - ♦ Storage reduced by a factor of N
- Example (N=3)
 - ♦ $N < W + R$ ($3 < 4$)
 - ♦ Writes: $W = 2$
 - Three writes must succeed (Primary + 2 replicas)
 - ♦ Reads: $R = 2$
 - Two reads provide verification

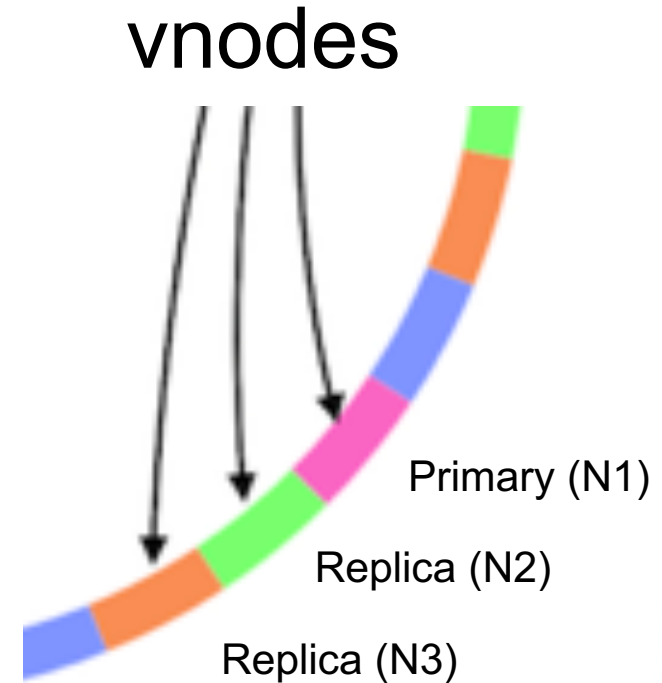


Image from basho.com

Hinted Handoff (Anti-entropy): Data replication

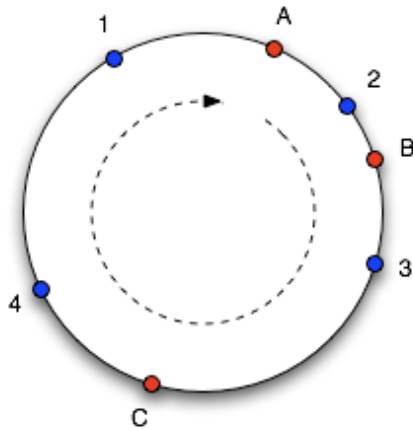
- Data replication to adjacent nodes ensure recovery and availability of the data
 - ◆ Default = 3 (at configuration of cluster). This assumes a 3-node cluster
 - ◆ Replication should be less than the number of physical nodes (no value is obtained from replicating data on the same node: vnode mapping)
 - ◆ Like Master-Slave, a higher replication reduces performance



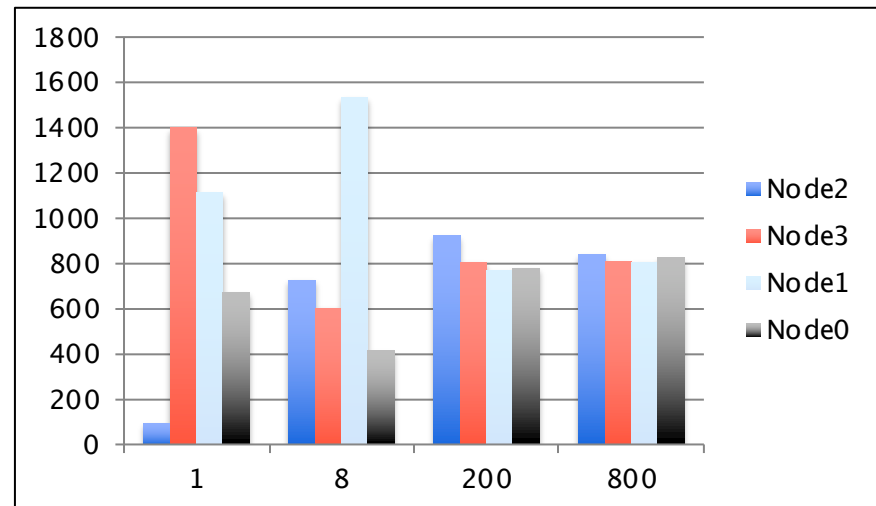
Replication (n_val) < physical nodes

Why vnodes are important?

- Map data to servers based on where the data's $\text{hash}(\text{key})$ falls relative to a server's $\text{hash}(\text{identity})$
 - ♦ Uniformly distributed data relies upon the density of servers (virtual servers are used to increase the distribution)



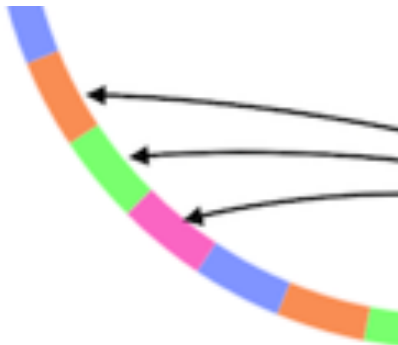
nodes A, B, C do not divide up the hash space evenly



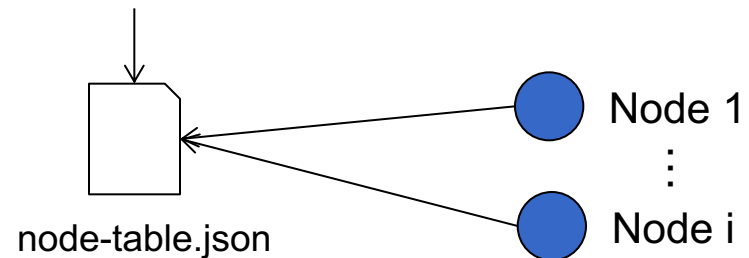
4 nodes with virtual nodes (1, 8, 200, 800)

Hinted handoff calculated (algo.) vs. using a lookup table

- Virtual bucket ranges (e.g., Dynamo)
 - ♦ Distribution of buckets (range) more evenly distributed across nodes
 - ♦ Fixed buckets increase the likelihood of equal distribution
 - Table lookup required to find a node
 - Distribution does not guarantee load or storage distribution



Node n = findNodeForKey(String key);



Re-distribution of vnode assignments

- Rebalancing involves moving (re-assigning) buckets to nodes
 - ♦ Randomization (mixing) reduces probability of hot spots
 - Randomization factors: true random selection offset by factors such as rack distribution, replication spread, hot spots, storage capacity, and server capacity
 - ♦ Management algorithms can re-assign buckets to balance activity
 - ♦ CON: randomization/balancing requires a lookup table → distribution mgmt complexity
- Example implementations: Riak, MongoDB

Replication with quorums

- N,W,R
 - ♦ Defines behavior for replication, failover, and performance
- Examples
 - ♦ Writes: $W = 3$
 - Three writes must succeed (Primary + replica x 2)
 - ♦ Reads: $R = 2$
 - Two reads are used for consistency

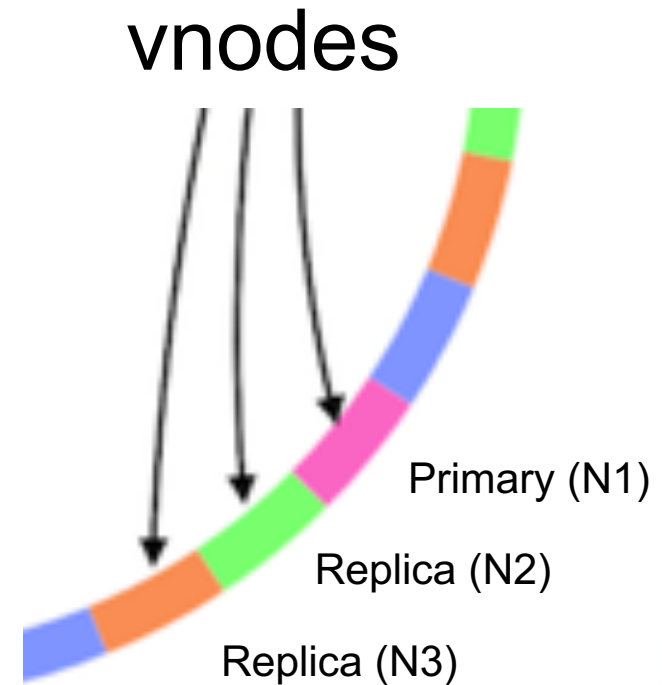


Image from basho.com

N,R,W choices affect behavior of the storage model

- N,R,W affects the availability and durability of the stored data
 - ♦ Implications affect the possibility of multiple values for the same key
 - ♦ Ensure data is always available and updates not lost
- Variations
 - ♦ Consecutive computed virtual nodes
 - ♦ Pick list of ordered virtual nodes
 - ♦ Variability by data (N varies for category of data)
- Configuring behavior
 - ♦ Synchronous (consistent, slower)
 - ♦ Asynchronous (inconsistent, fastest)
- Examples
 - ♦ $N = 3, W = 3, R = 1$ (tuned for reading)
 - ♦ $N = 3, W = 2, R = 2$ (66% quorum, flexible)

Quorums and handoffs

- Trade off by the application (eventually consistent, quorum)
- NRW
 - ♦ on PUT can set the W value!
 - ♦ `put(key, value, w=N)` or `put(key,value)`
 - ♦ What are we giving up for increased replication?

Some questions...

Synchronous communication viewed as a nested function call

Let's setup a overlay network consisting of 4 nodes (A,B,C,D) in a unidirectional organization (A \rightarrow B ..., D \rightarrow A)

How do algorithms like Flood Max and Bully compare in the number of messages sent in a ring vs. a fully connected graph vs. a tree?

Synchronous coordination

- If an asynchronous system enqueues requests and a synchronous system is blocking.
- How does a synchronous system work?
 - ♦ If send and receive are mutually exclusive how do you coordinate communication?

More reading

- Distributed Algorithms, Nancy A. Lynch, 1997
- Apache Zookeeper/Curator use of LCR

Summary

In a peer-inspired system, there may be no coordinator or the role of coordinator. Rather, this role is relative to the request. Implementations exist as multi-phase commits, and consensus approaches use approaches such as PAXOS.

- By not requiring a single node to act as a coordinator, a system can minimize failures of the leader.
- Secondly, if requests tend toward atomic, cooperation (leveling) of leadership responsibilities can reduce coordinator induced delays

Extended Lab (part 2)

(Challenges for you – look for highlighted text)

Designing a system to manage shared resources/information is challenging in a distributed or concurrent system

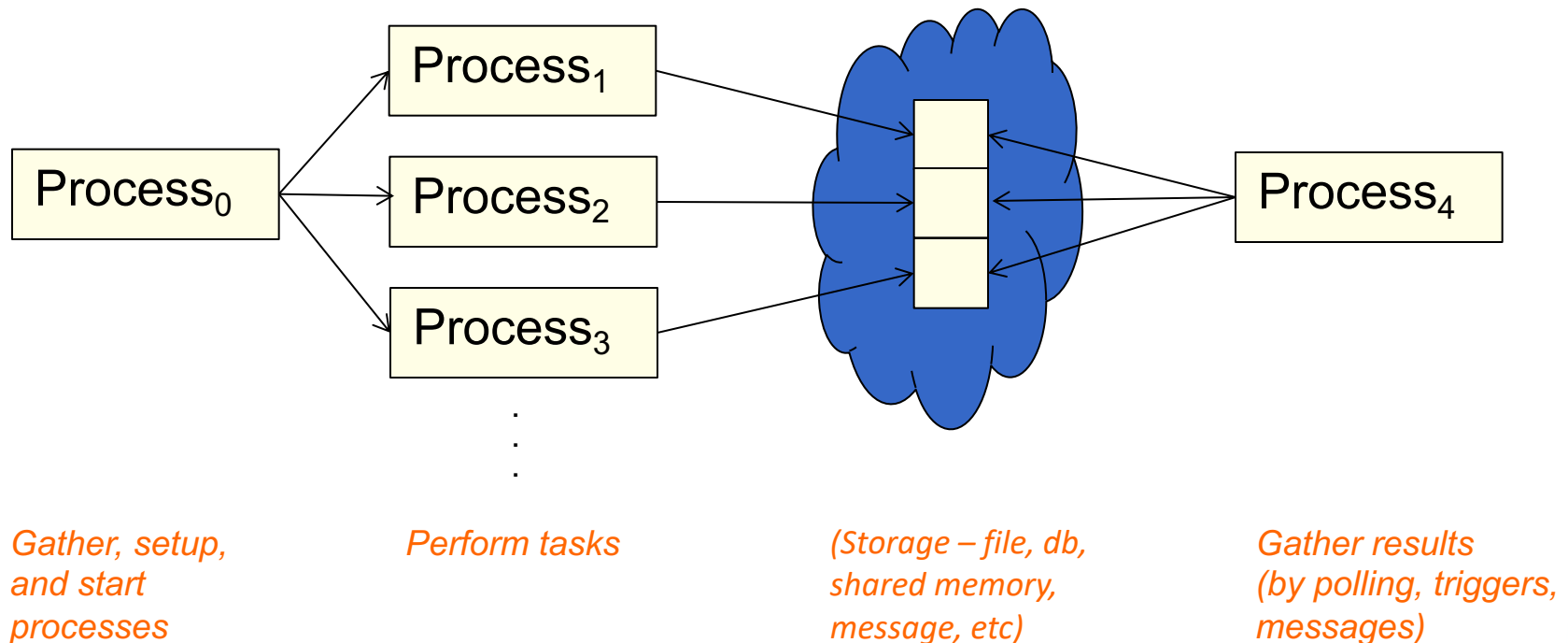
- Design exercises and testing your options
 - ◆ Shared mutable (Read-Write by all)
 - ◆ Isolated mutable (Controlled RW)
 - ◆ Immutable (Read-only)

Shared mutable

- All objects (actors) have access to data and/or state
 - ♦ Changes do not incur bottlenecks due to coordination and synchronization
 - ♦ Very fast
 - ♦ Very error prone if used incorrectly
- Examples
 - ♦ Generative data models
 - ♦ Caches

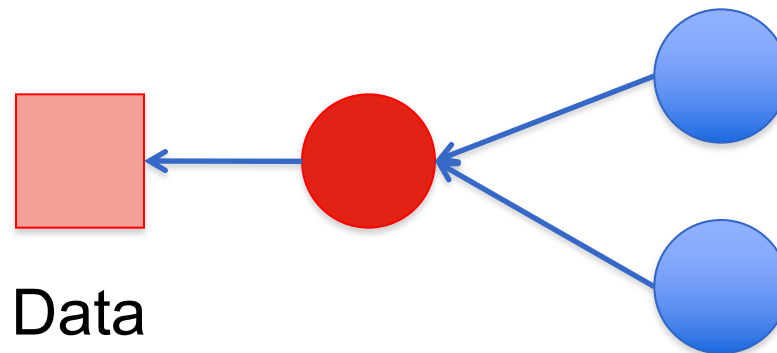
We have seen Generative models (streaming chunks or dividing work)

- **Create:** Multiple processes working in parallel → writers (1–3), and readers (4)
 - ♦ How do you ensure P_4 is ready to read results from 1,2,3?



Isolated mutable resources by limiting interactions using a guardian

- **Construct:** Guardian process to limit access to data/resource
 - ◆ How do you prioritize, ordered, and managed requests?

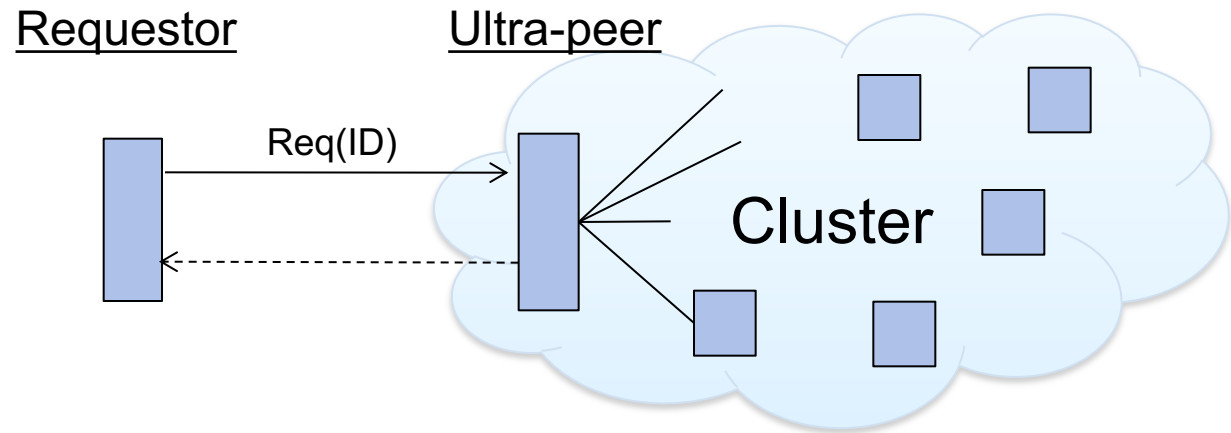


There are two general network (overlay network) designs

- **Structured** overlay networks
 - ◆ DHT, Trees, Rings
 - ◆ Organized and stable (low churn)
 - ◆ Bounded and unbounded networks
- **Unstructured** networks
 - ◆ Dynamic, self forming, adhoc
 - ◆ High churn, typically unbounded
- **Hybrid**
 - ◆ Semi-centralized networks
 - ◆ localized central behavior with a larger, overall, unstructured organization
 - ◆ Examples: super-nodes, lookup nodes, ultra-peers

Discovery/Coordination through super or ultra-nodes using a guardian design

- Within a cluster of nodes, a node can be designated to act as a coordinator or facilitator of connections or messages.



Communication flow to delegated requests

- How do you manage your cluster (local network)?
Options:
 - ◆ Direct
 - Messages send direct from the responder to the requestor
 - PRO: Most direct
 - CON: Exposes internal organization.
 - ◆ Along the path traveled
 - Messages returned by reversing the path traveled (unwinding the call)
 - PRO: Hides internal organization – this allows load balancing and fail over options
 - CON: Increases latency and complexity as message incur additional ‘hops’ to reach the requestor

Joining or discovering networks

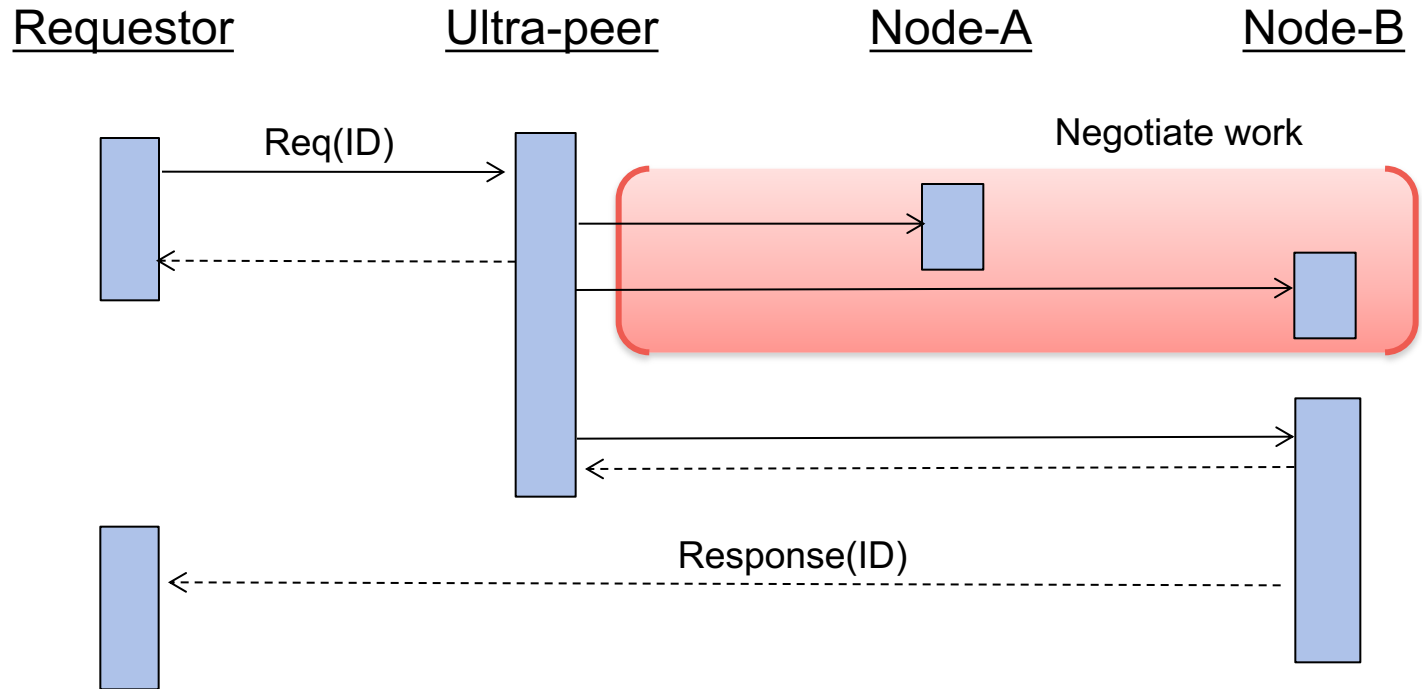
- Centralized indexing to connect nodes
 - ♦ Established communication directly between nodes
 - ♦ Examples: Napster (one of the more infamous)
- Hybrid central lookup
 - ♦ Look for super (ultra) peers
 - ♦ Examples: KaZa
- Friend-to-friend
 - ♦ Look to a list of local (trusted, friends) nodes
 - ♦ Examples: Gnutella, Freenet, OneSwarm

Searching – finding data

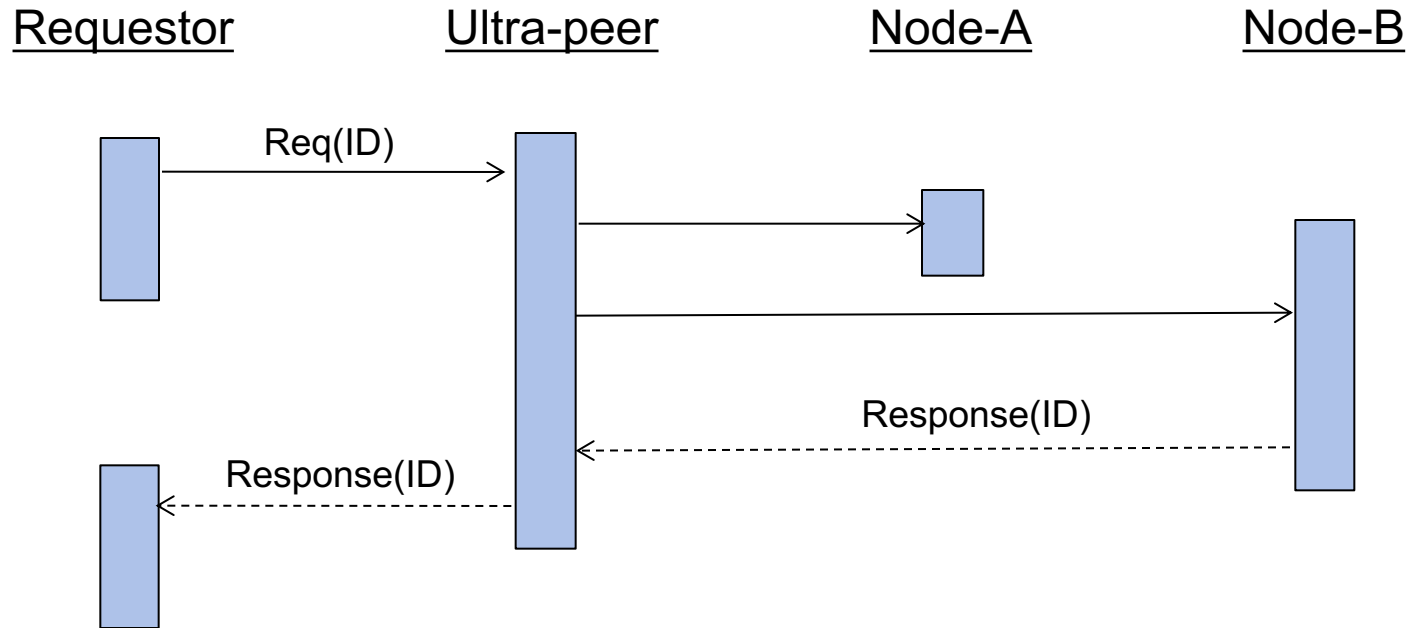
- Basic strategies
 - ♦ Centralized or semi-central indexing
 - ♦ Algorithmic – commuted lookup (DHT)
 - ♦ Flood – variations include broadcast to directed
- Challenges
 - ♦ Consistency (changing graph edges can affect repeatable results)
 - ♦ Scalability (number of of messages sent can impact network bandwidth)
 - ♦ Trust

Using an Ultra-peer: Directly to the requestor

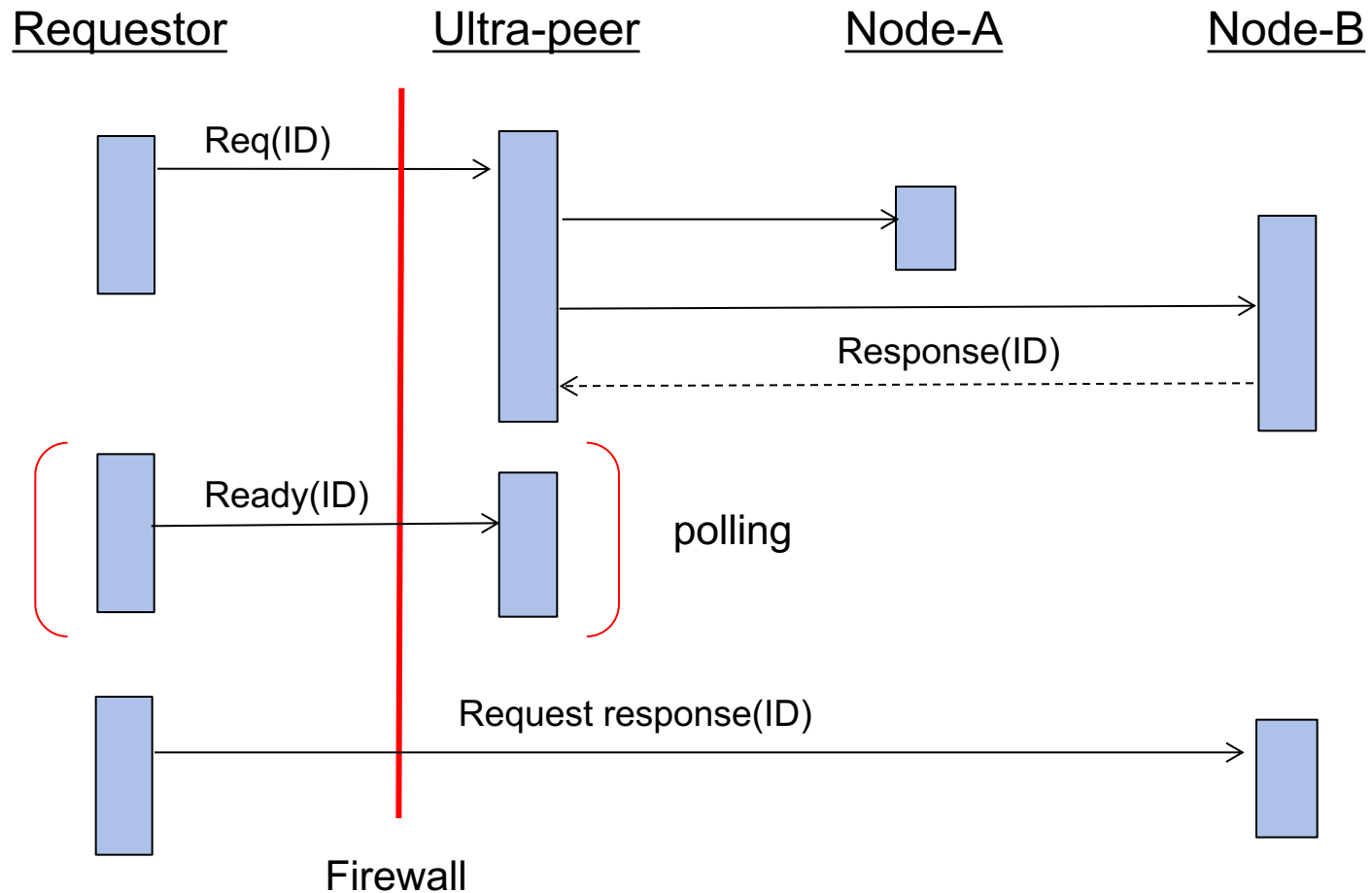
(Ultra-peer used to locate resources)



Ultra-peer: Along the request's path



Ultra-peer: Requestor initiated response where firewalls prevent direct interaction (Node-B → requestor)



Four challenges of an overlay network

- Connecting (joining a network)
 - ♦ Discovery – join/leave networks
- Trust
 - ♦ Identity and trust between requestor and responder
 - ♦ Trust between requestor and path traveled
- Search and returning data
 - ♦ Finding data and services
 - ♦ Delivery of data
- Advertising services/data
 - ♦ How do services reach clients