Self-Management of Large Scale Distributed Systems

Boriss Mejías

Université catholique de Louvain,
Louvain-la-Neuve, Belgium
boriss.mejias@uclouvain.be

23^{th} Jan, 2009

Self-Management of Large Scale Distributed Systems

Boriss Mejías

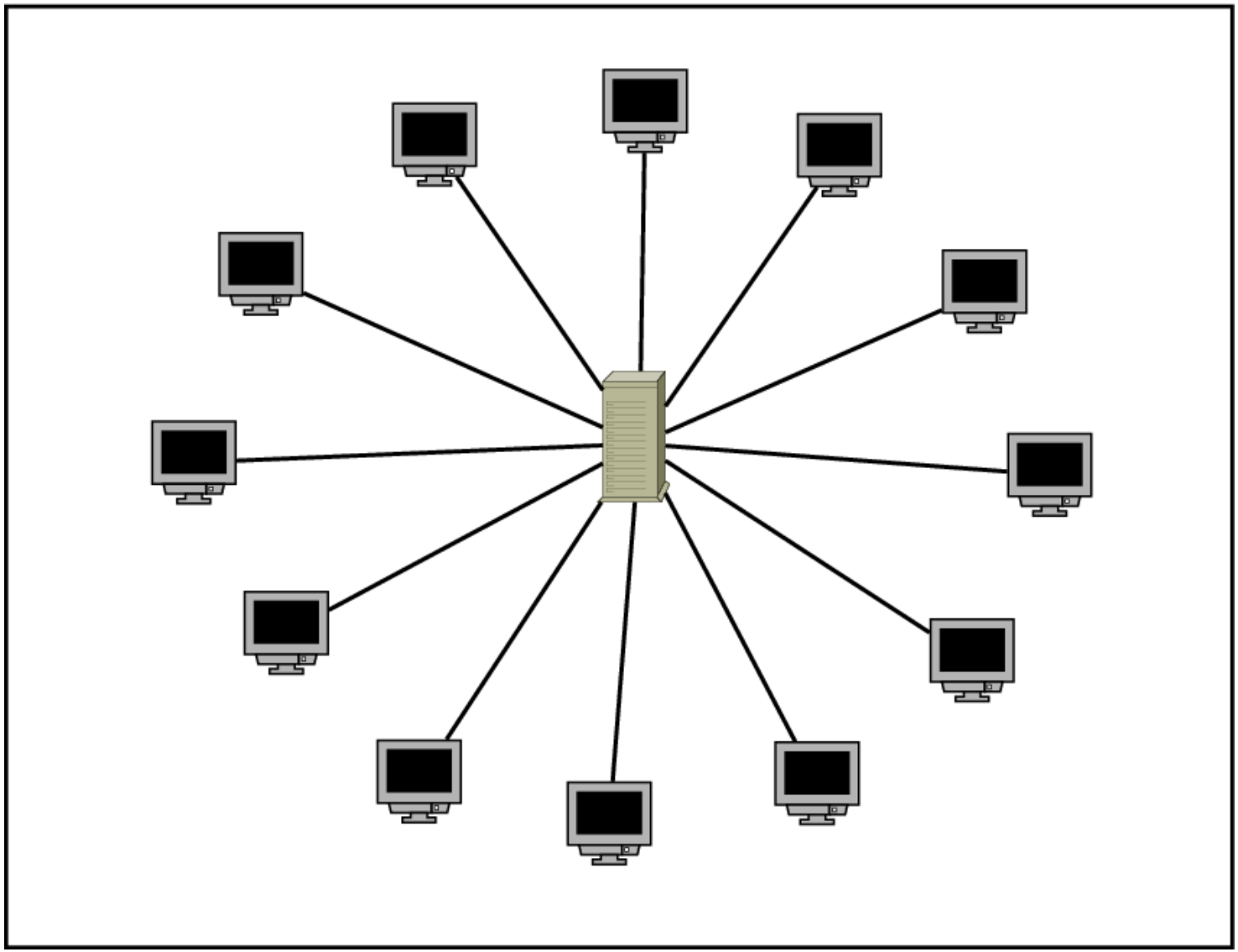
WARNING!

No jokes nor hot chicks on this presentation

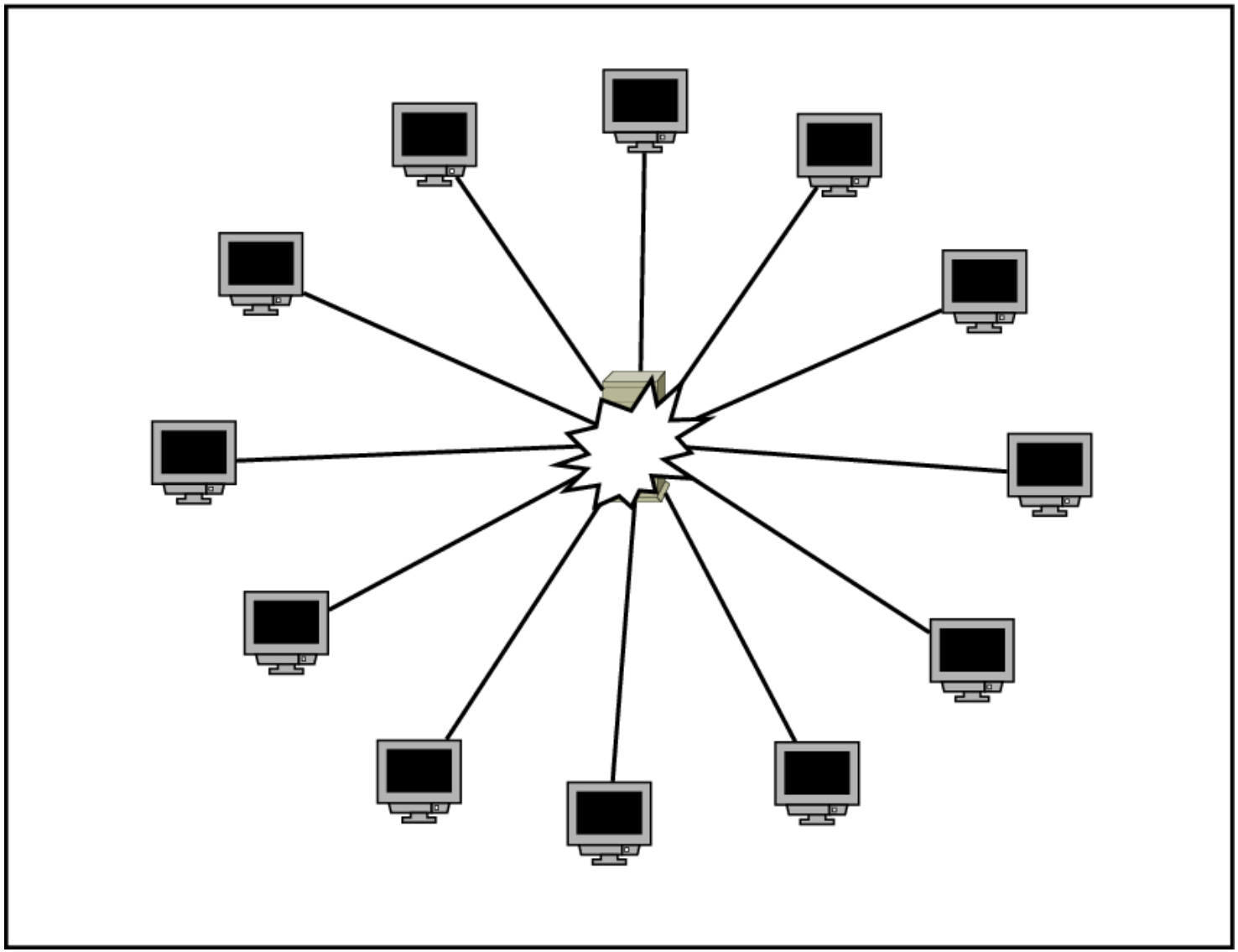
Université catholique de Louvain,
Louvain-la-Neuve, Belgium
boriss.mejias@uclouvain.be

23^{th} Jan, 2009

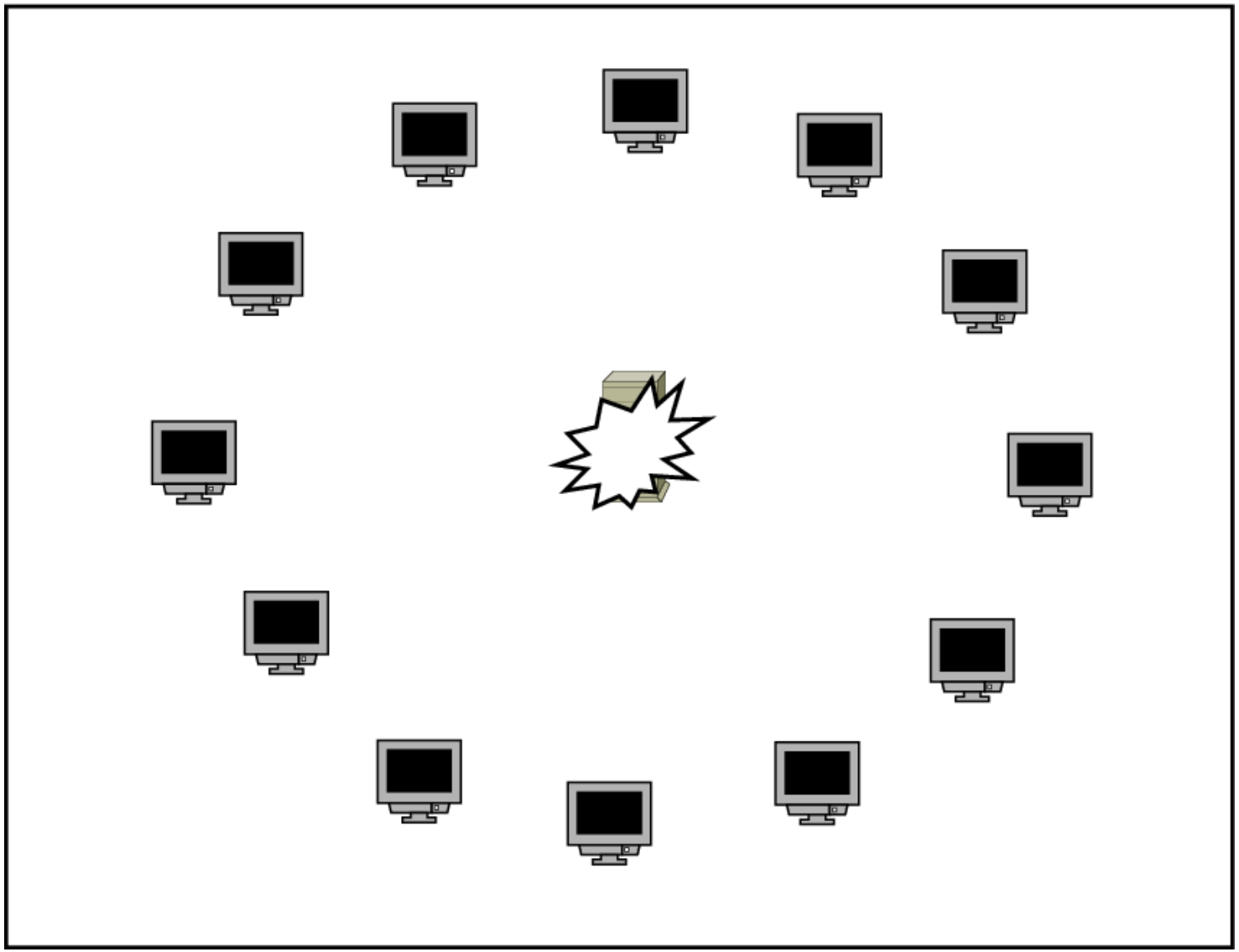
Motivation



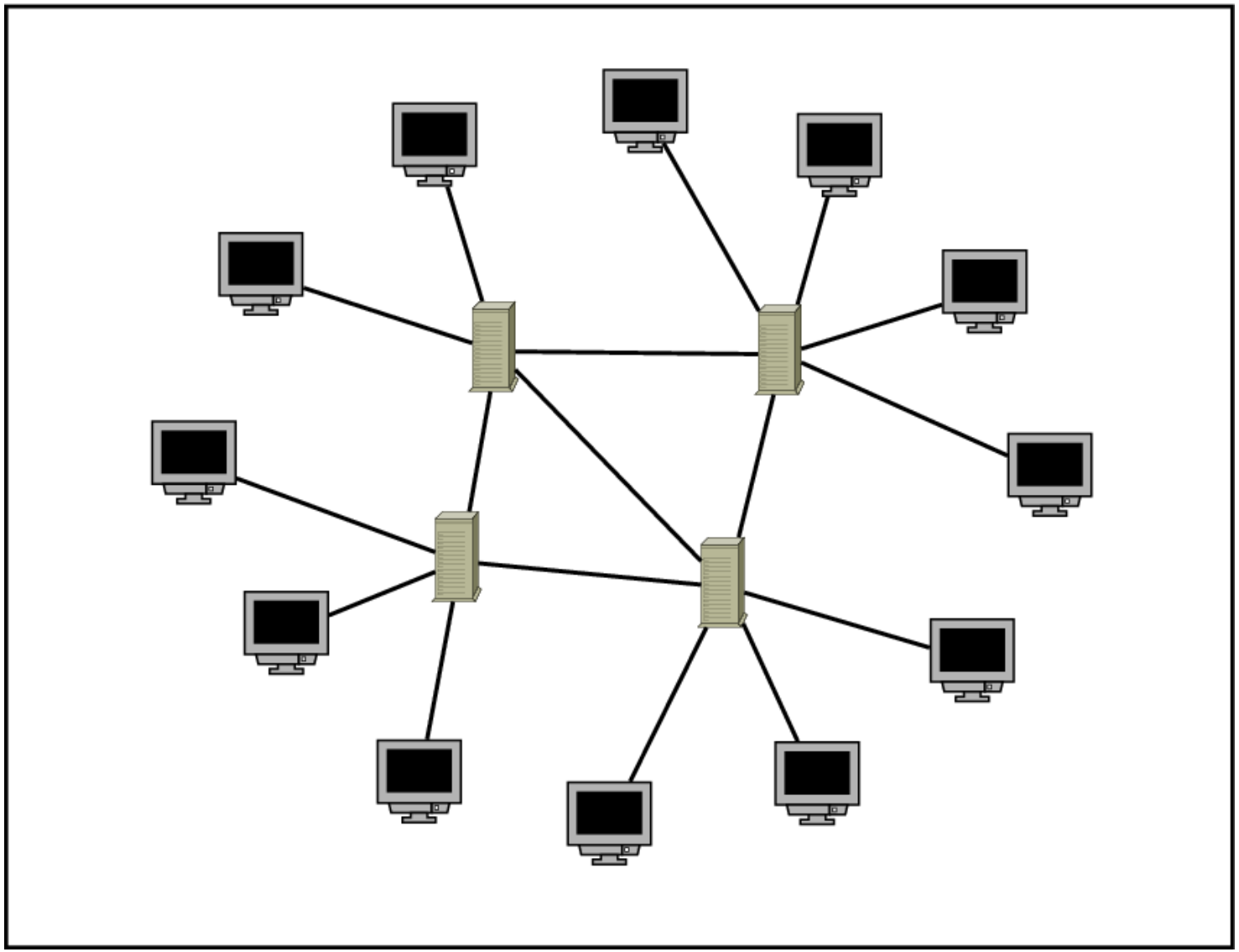
Motivation



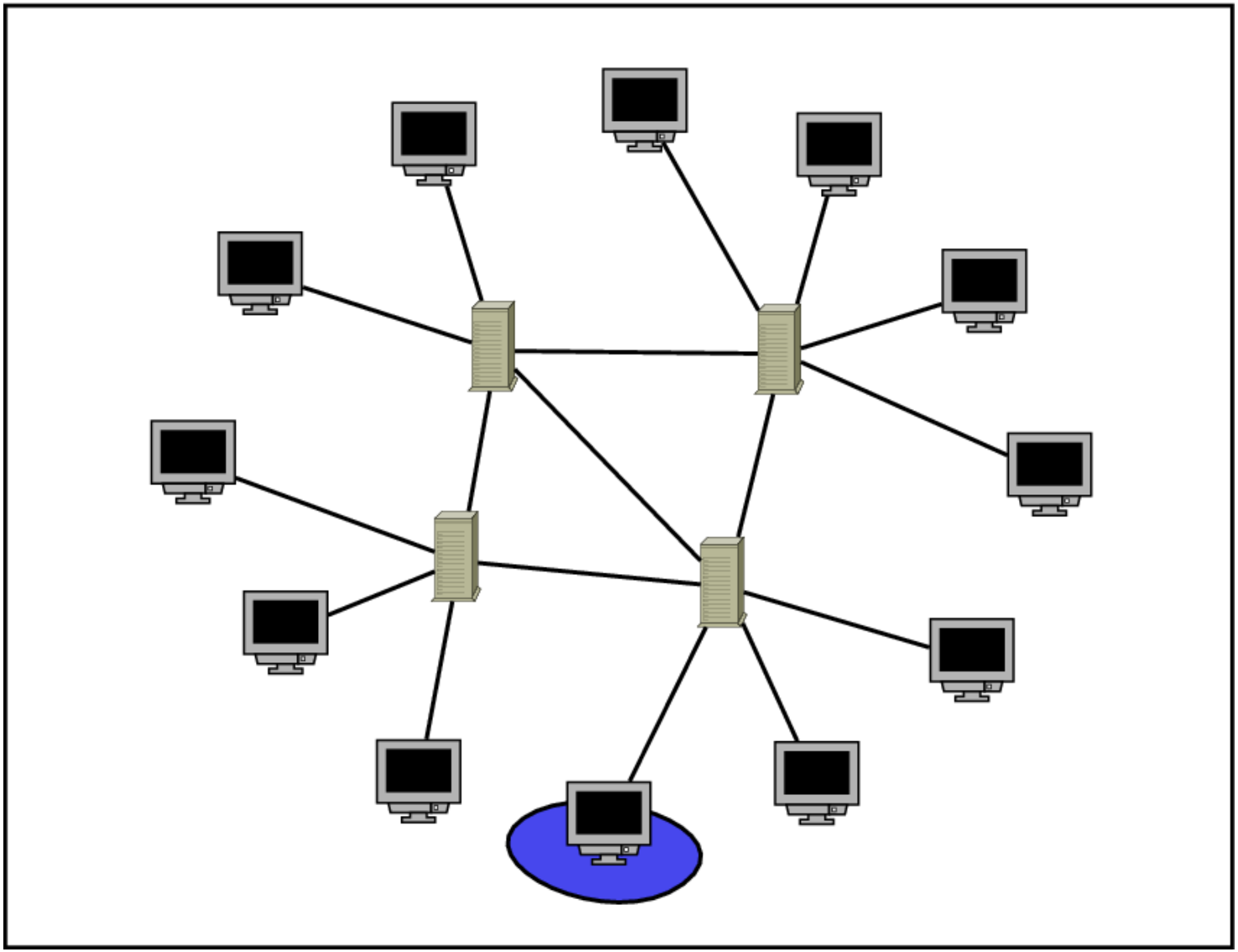
Motivation



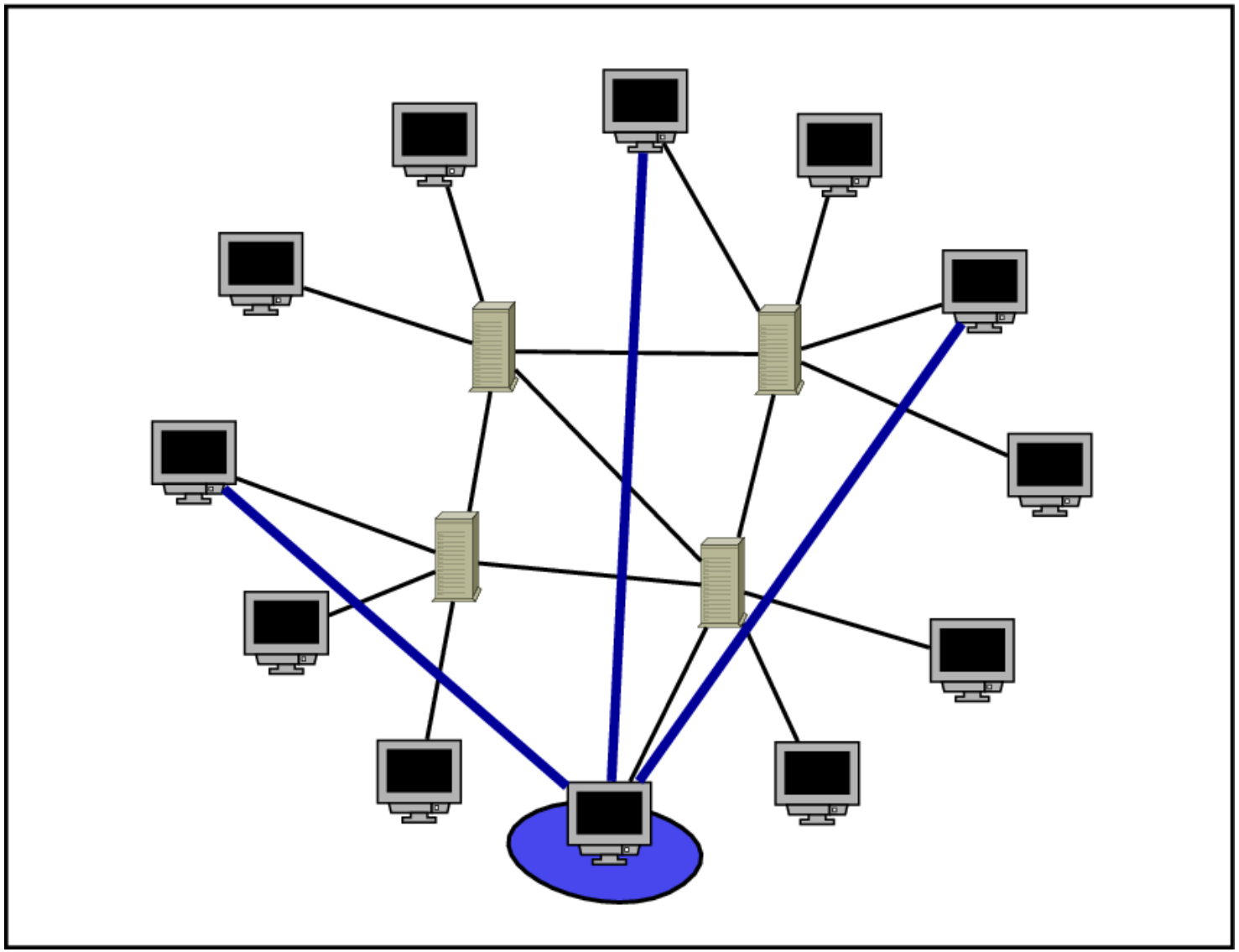
Motivation



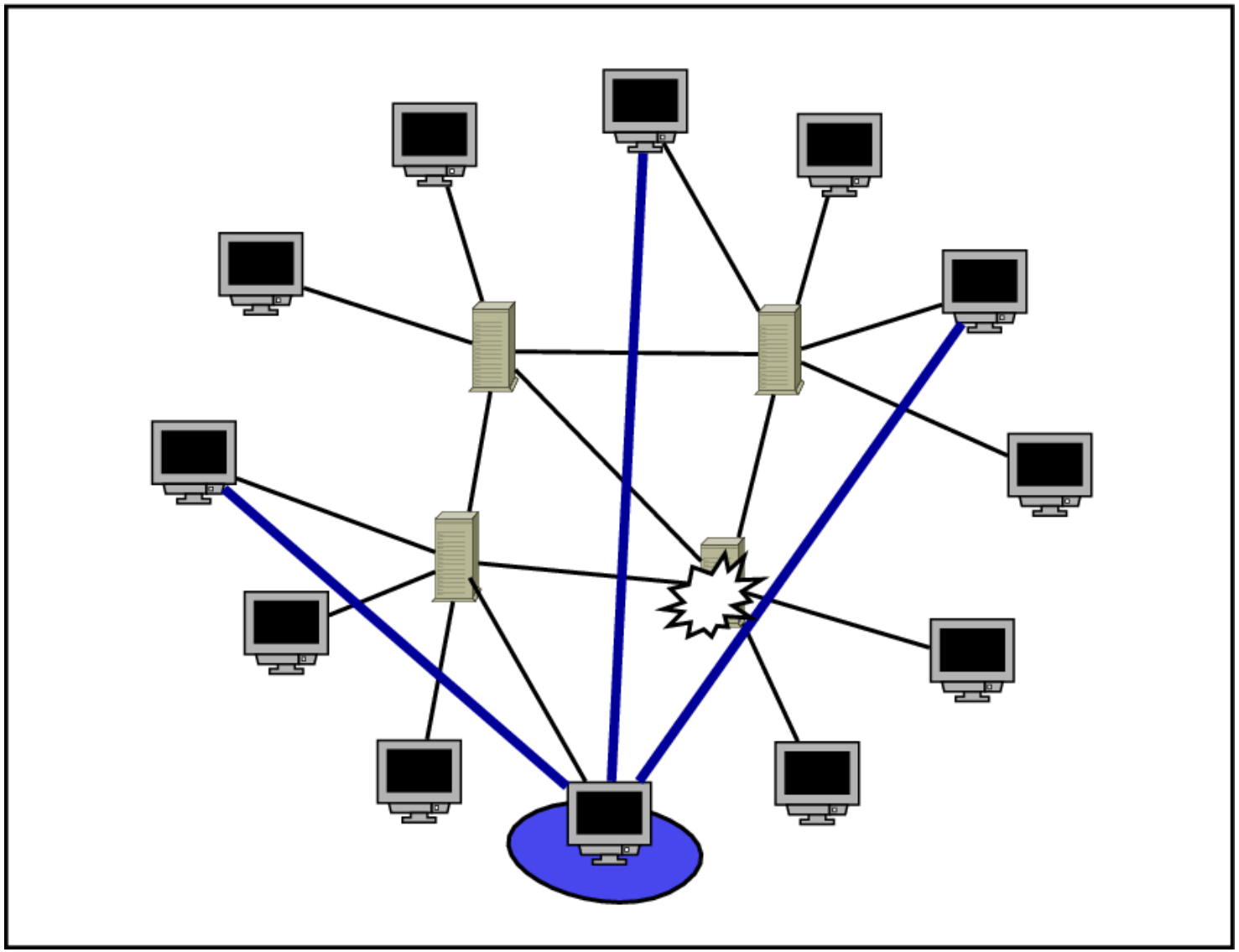
Motivation



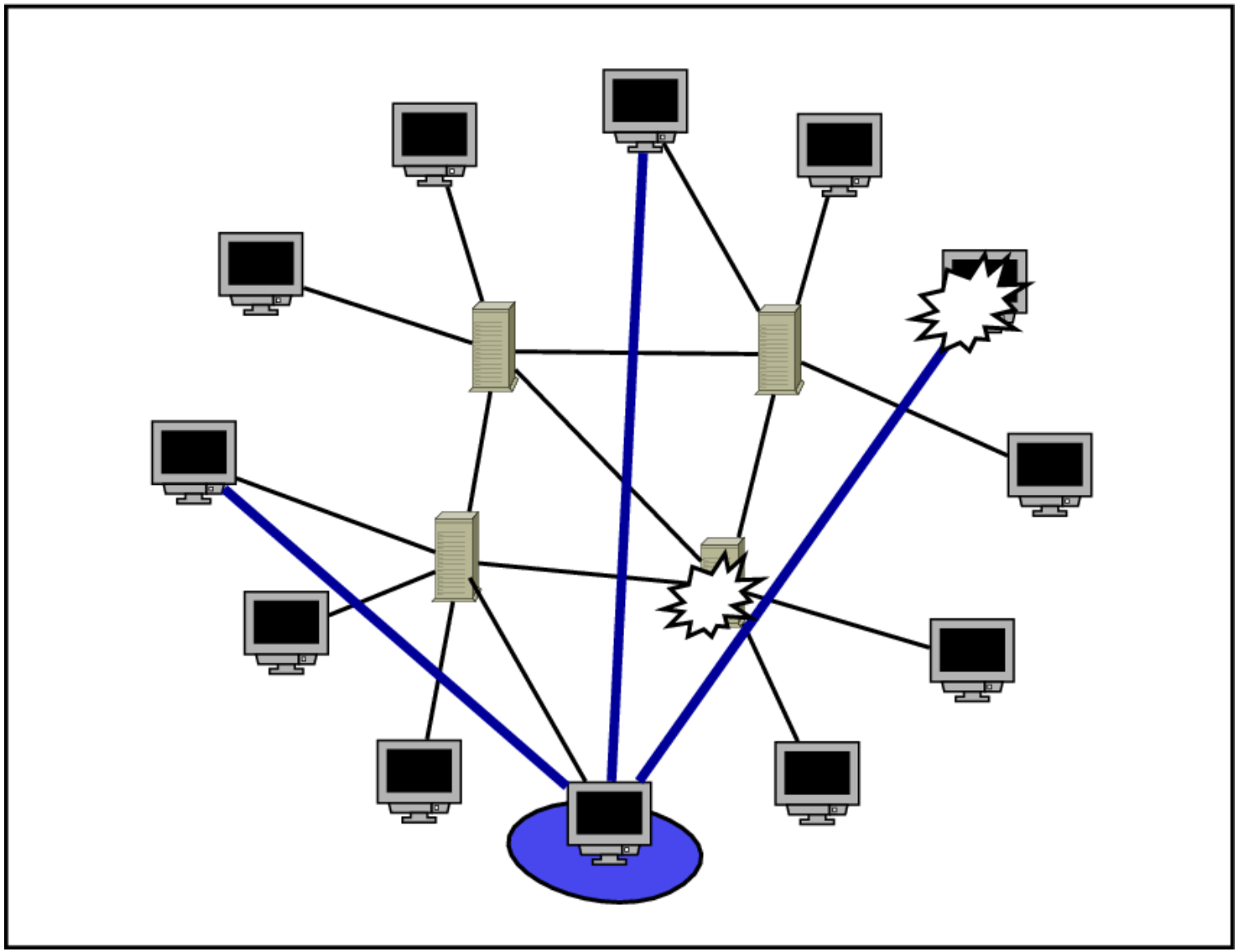
Motivation



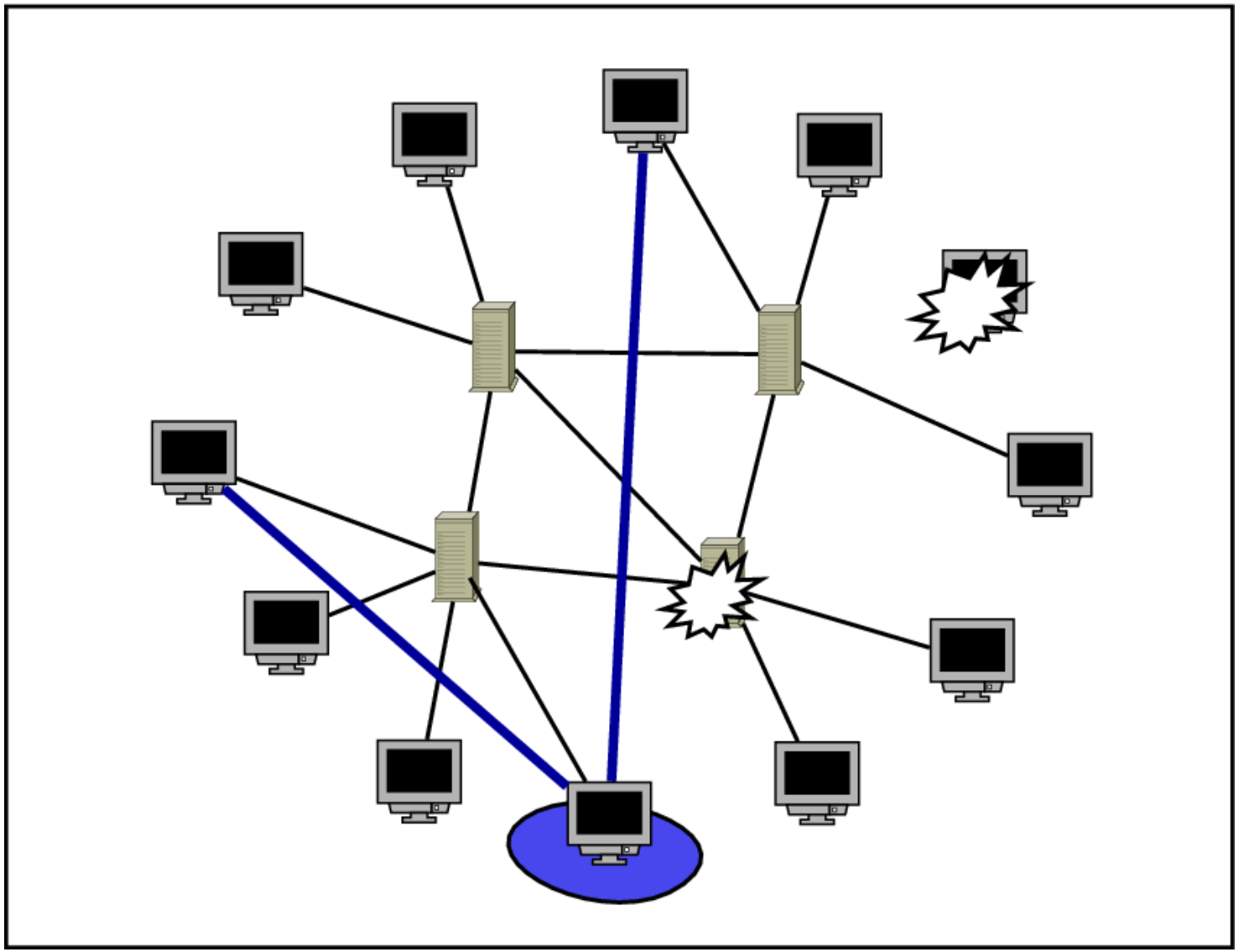
Motivation



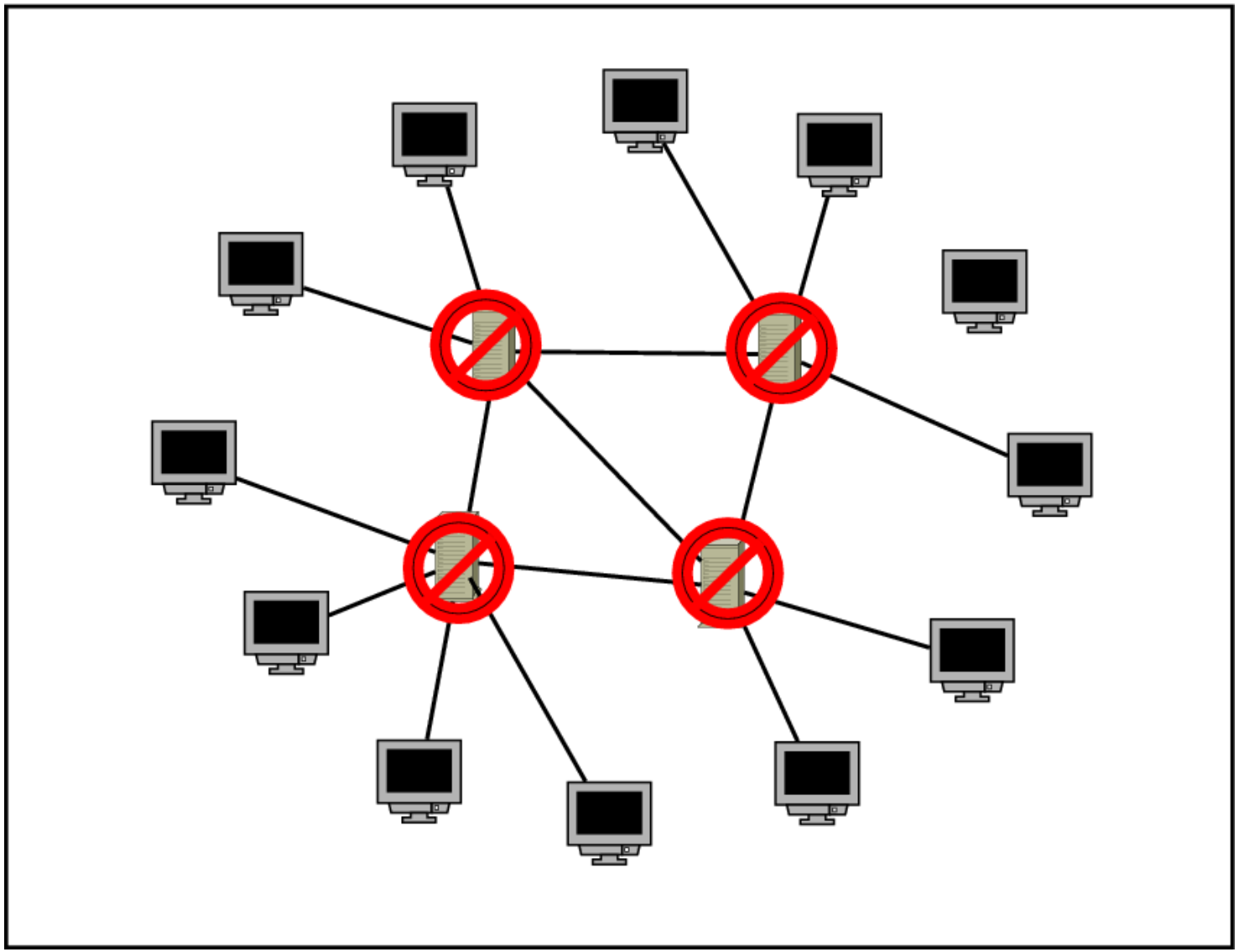
Motivation



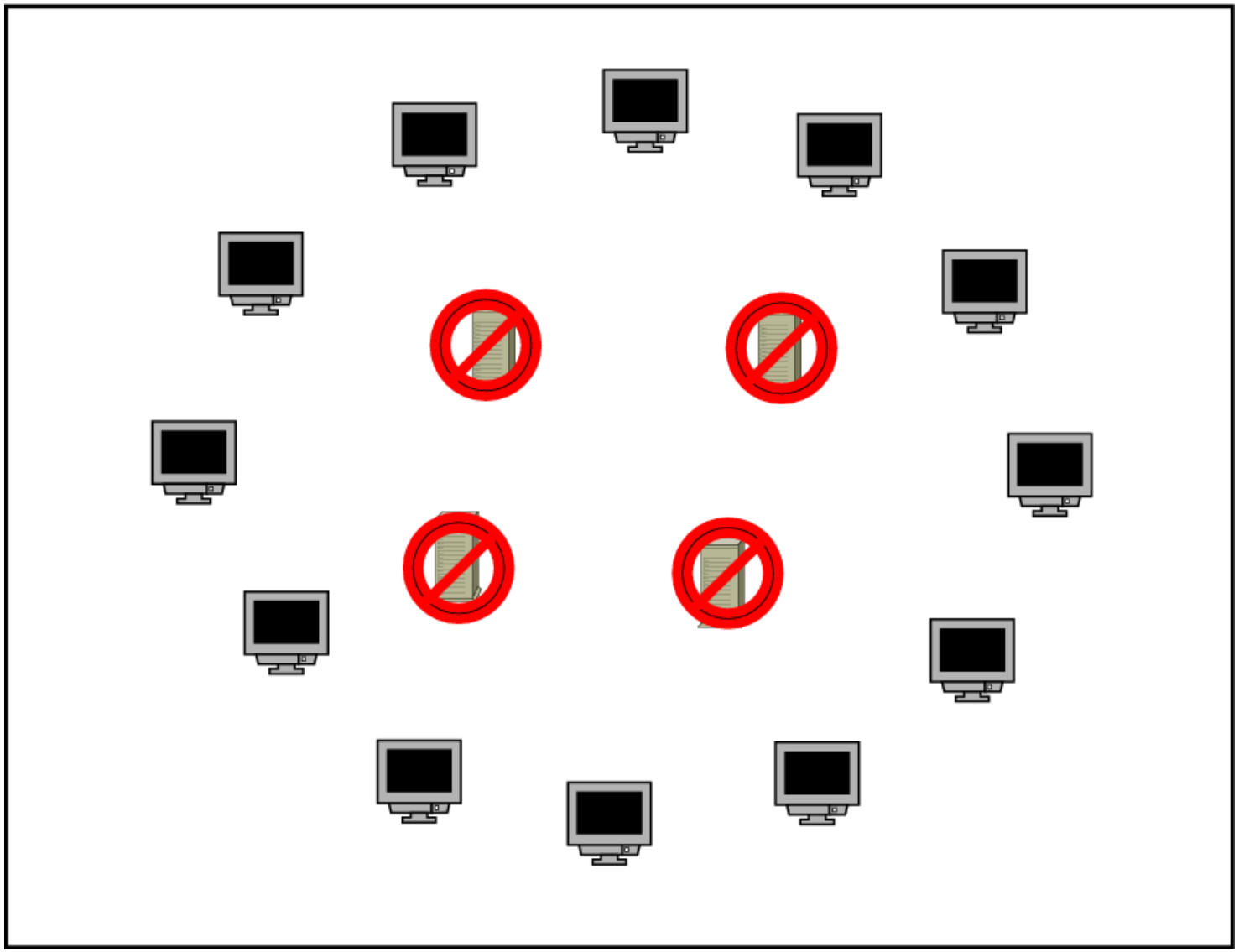
Motivation



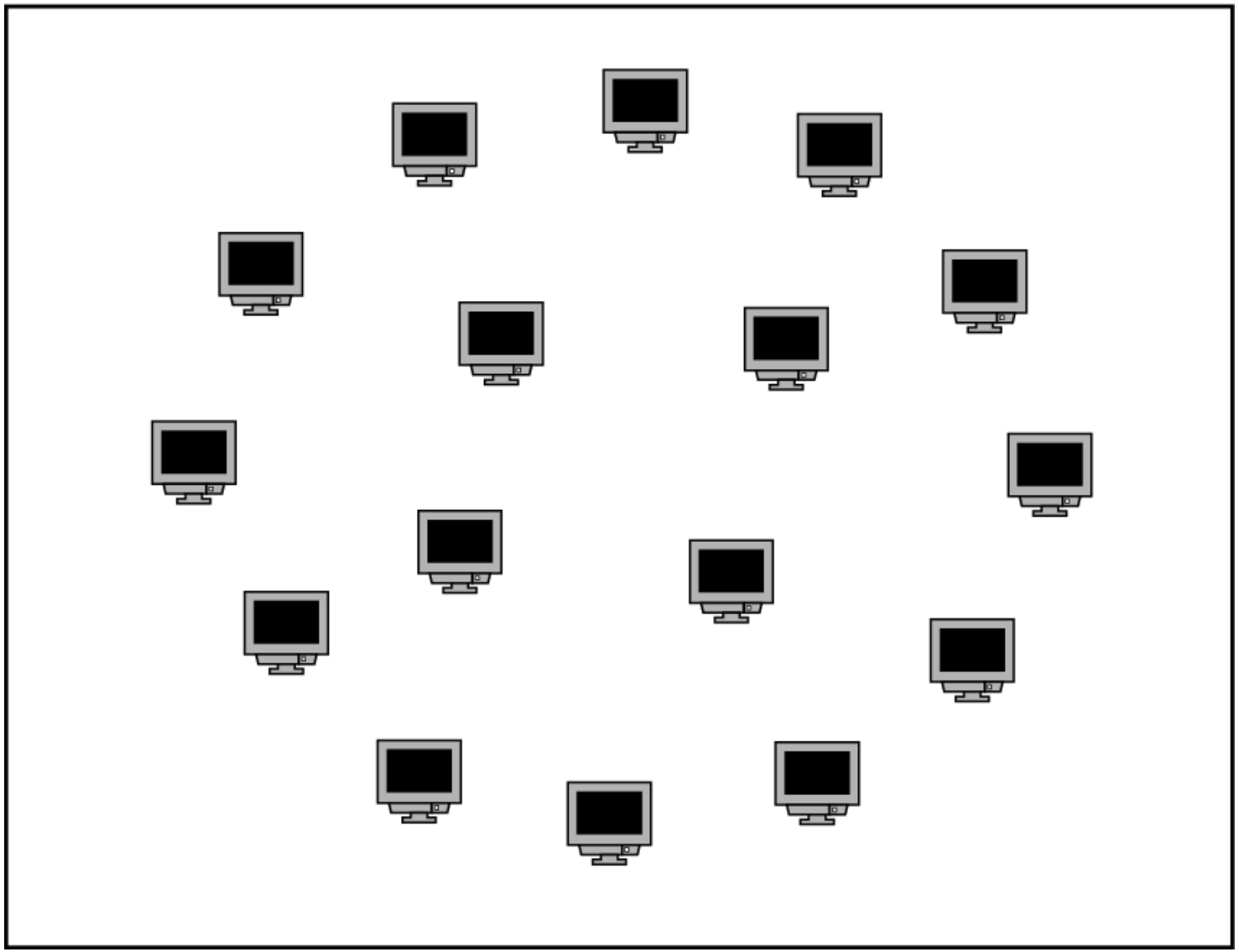
Motivation



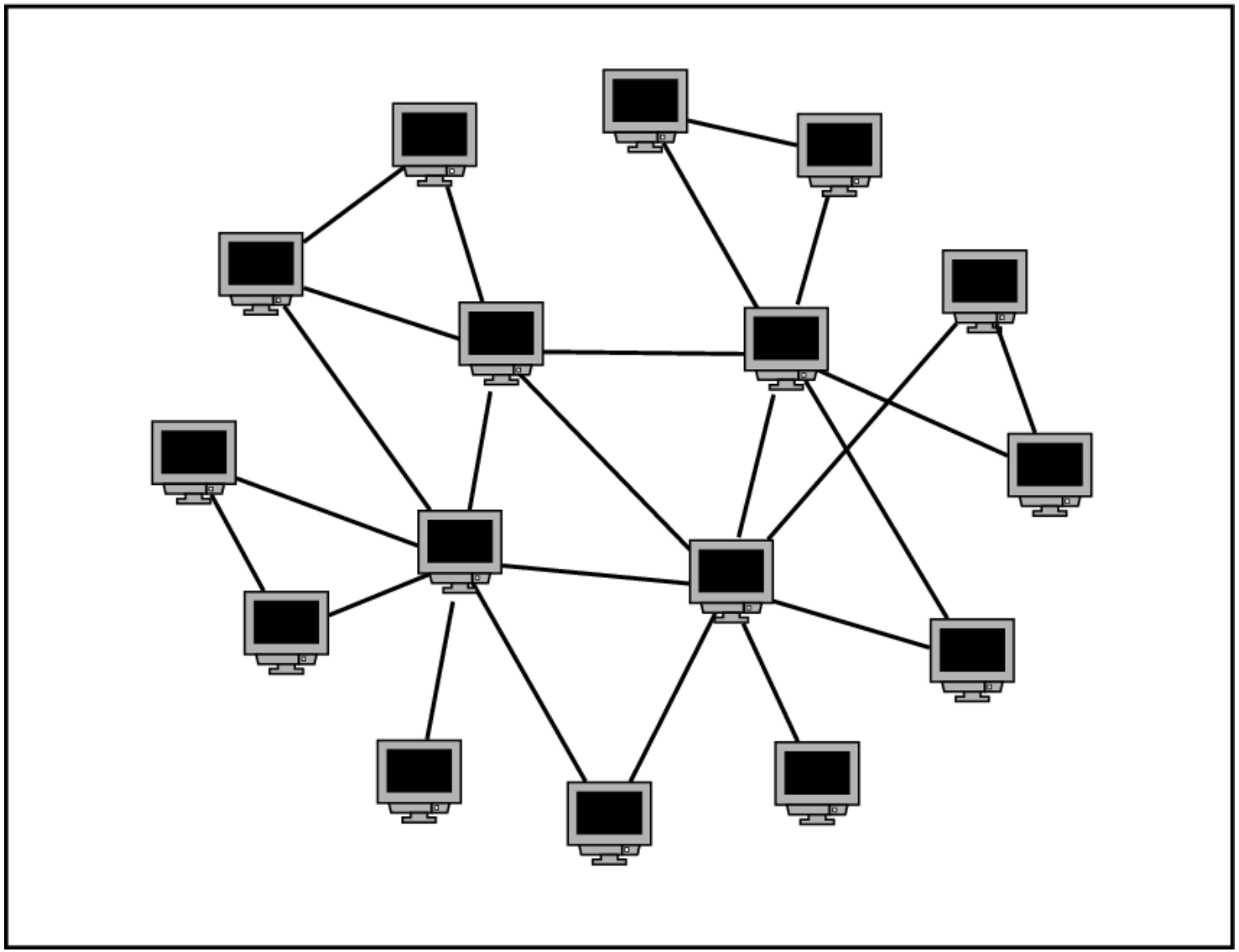
Motivation



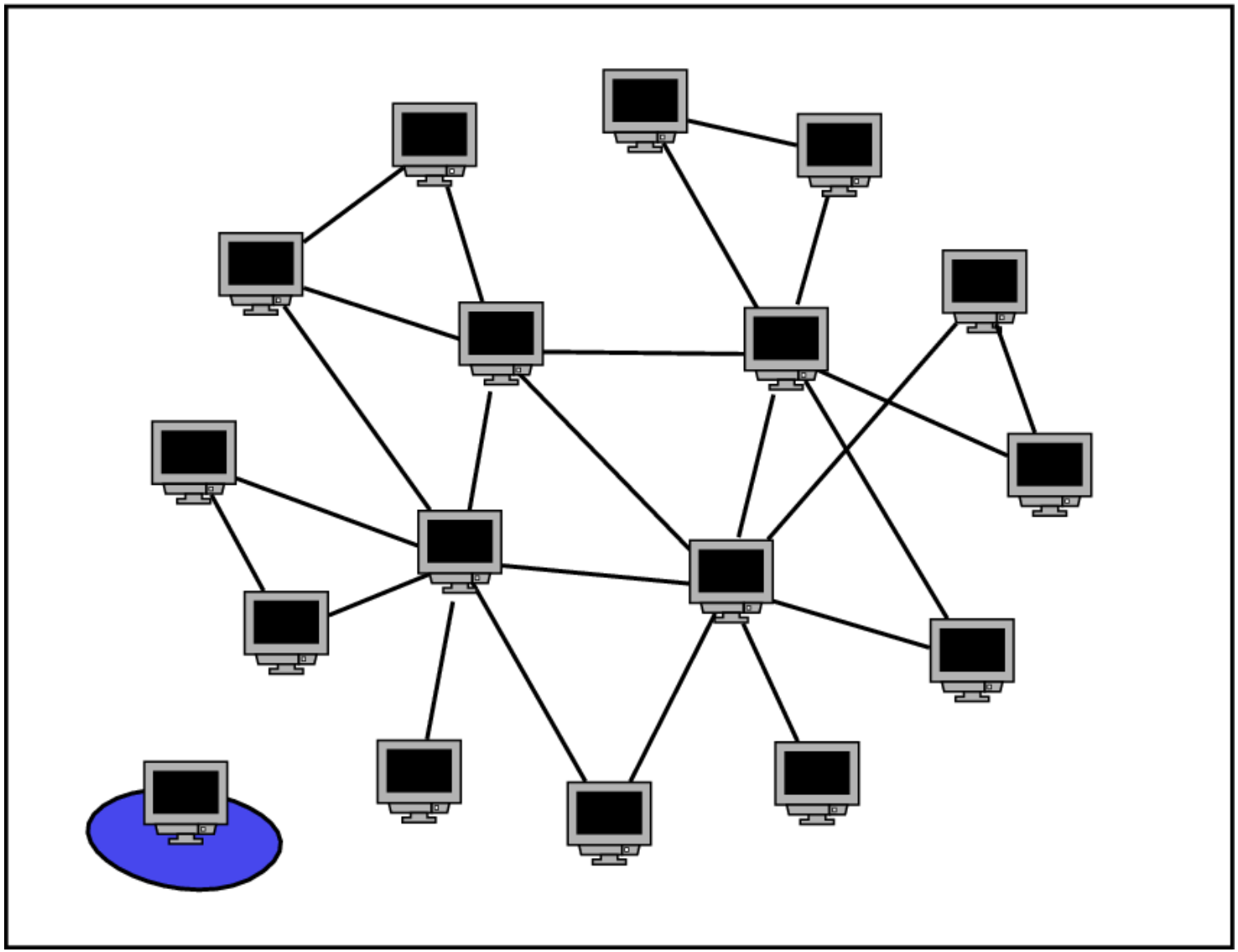
Motivation



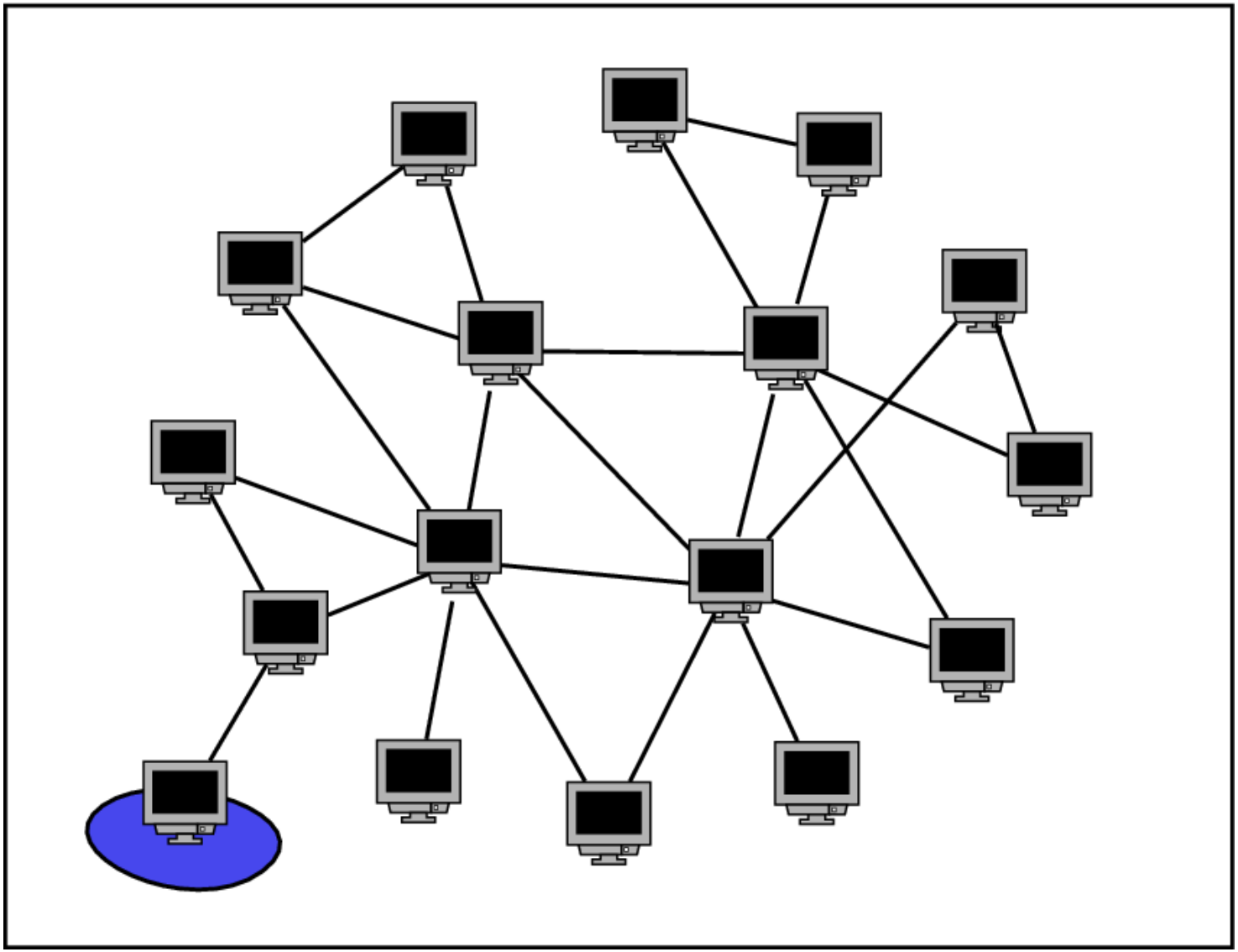
Motivation



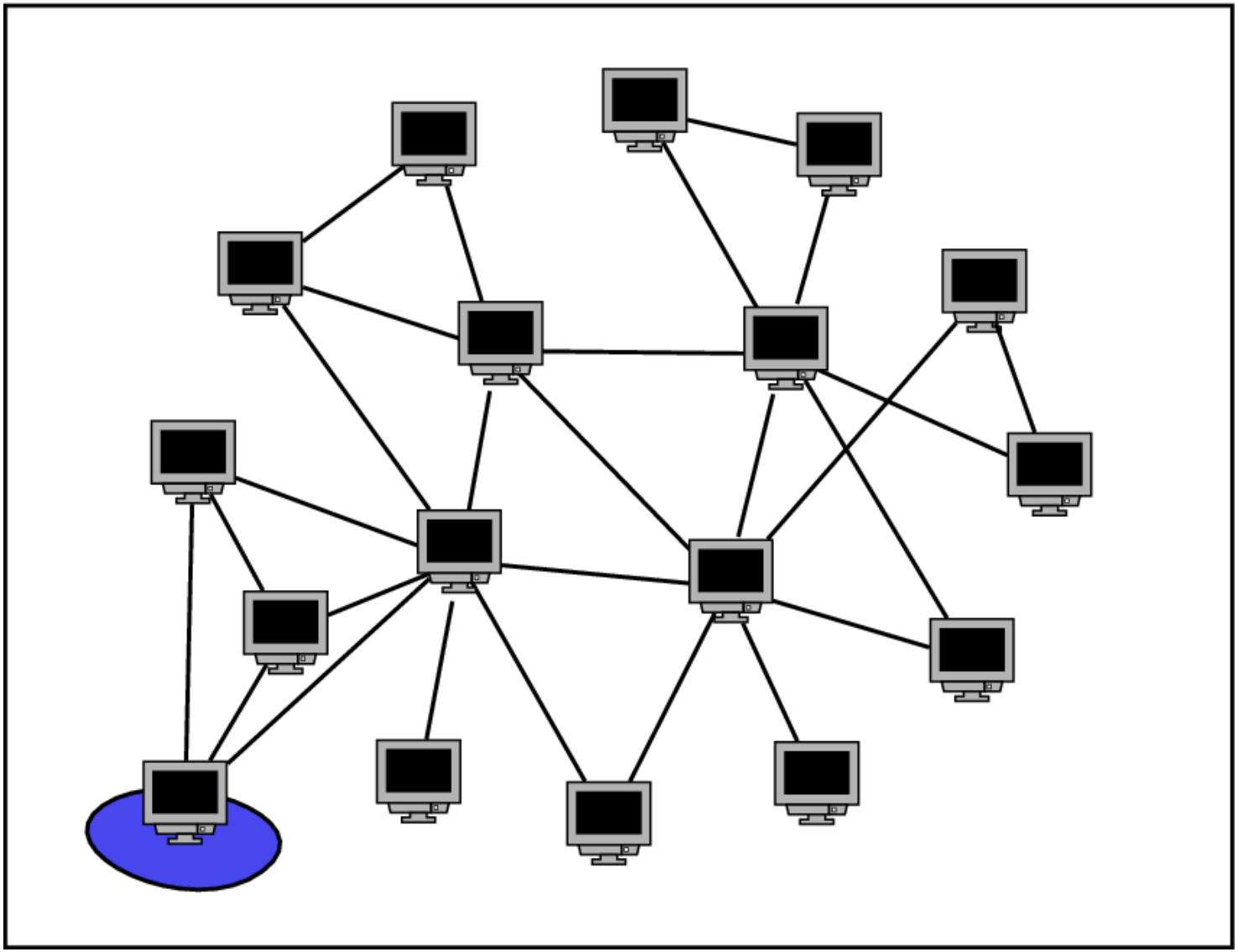
Motivation



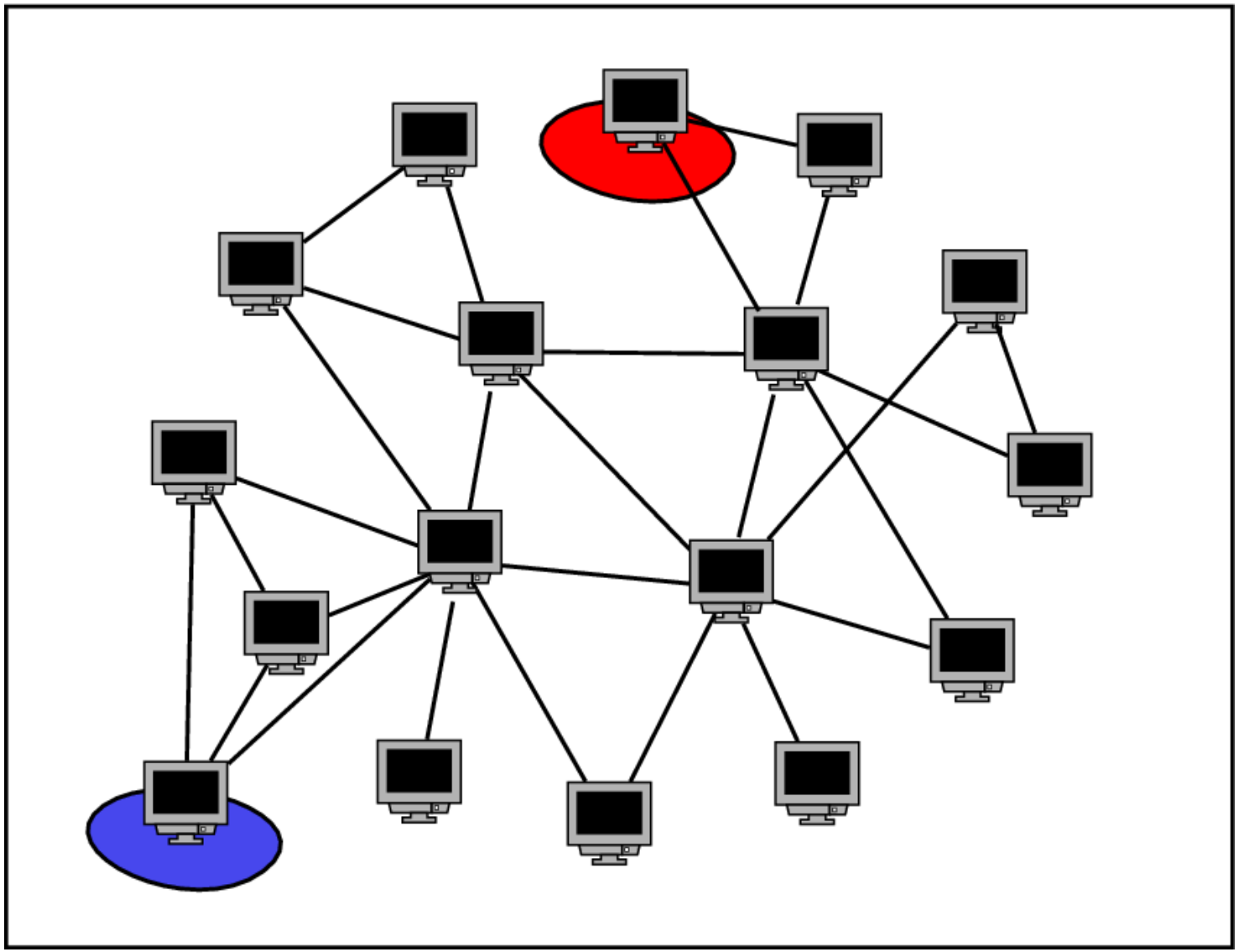
Motivation



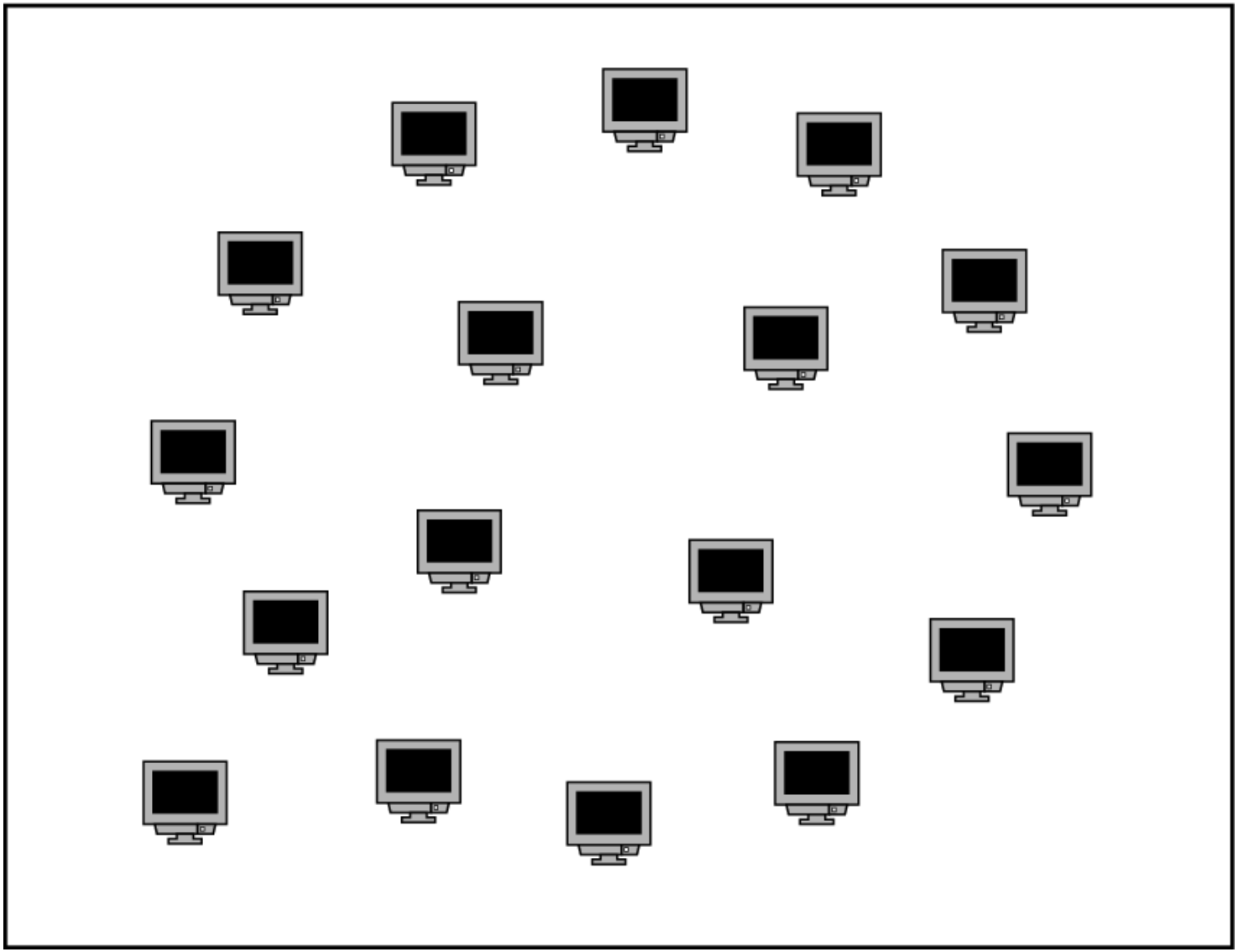
Motivation



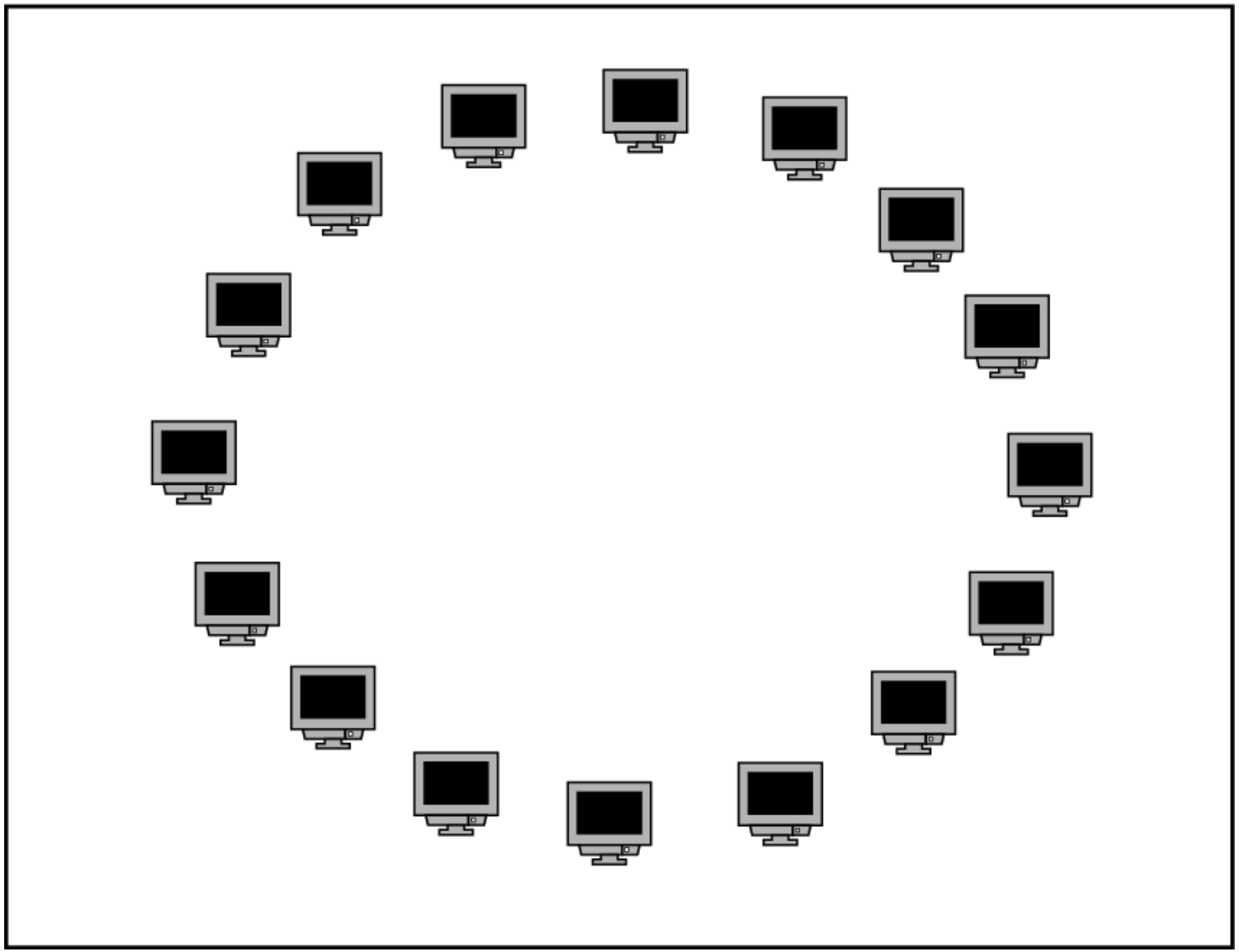
Motivation



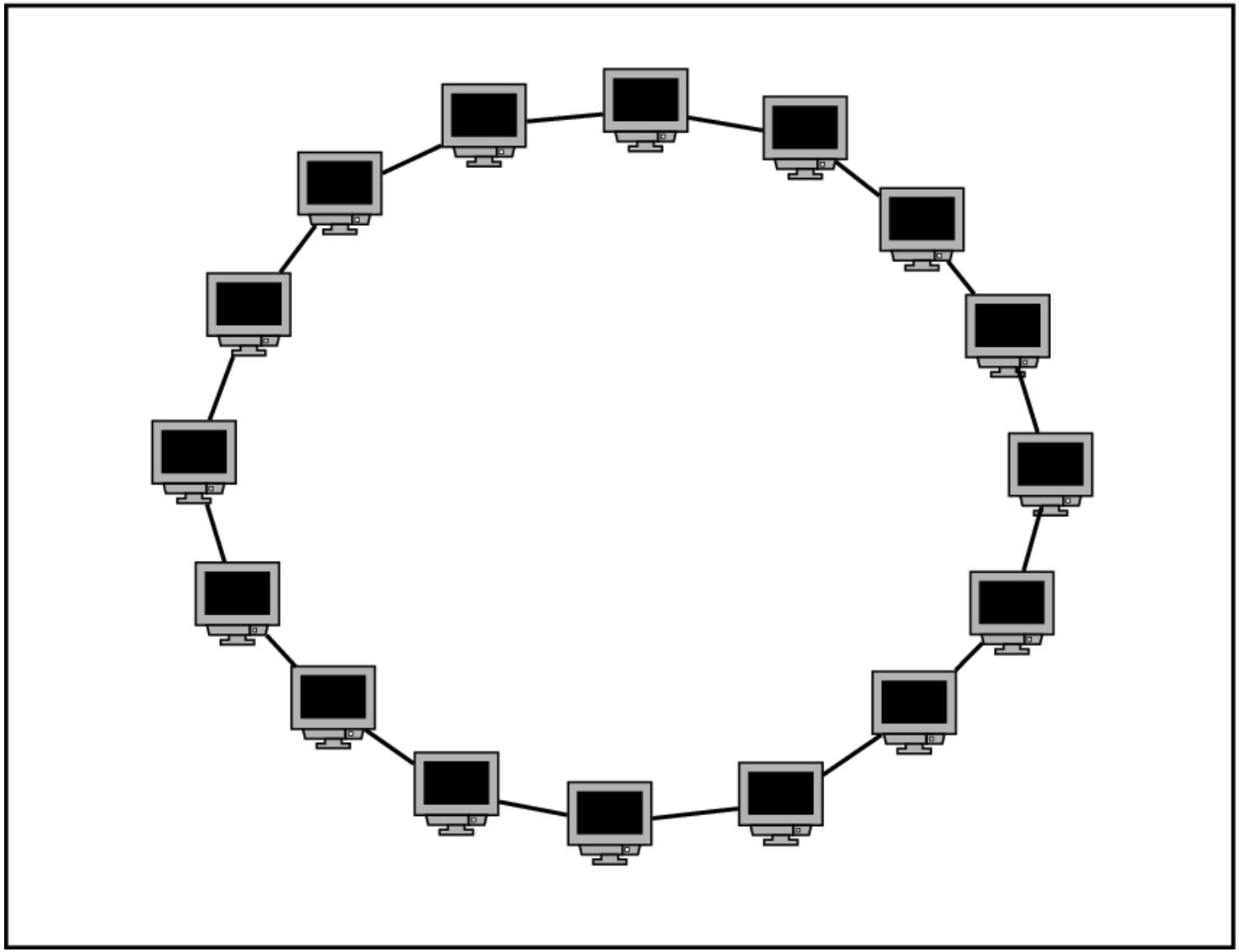
Motivation



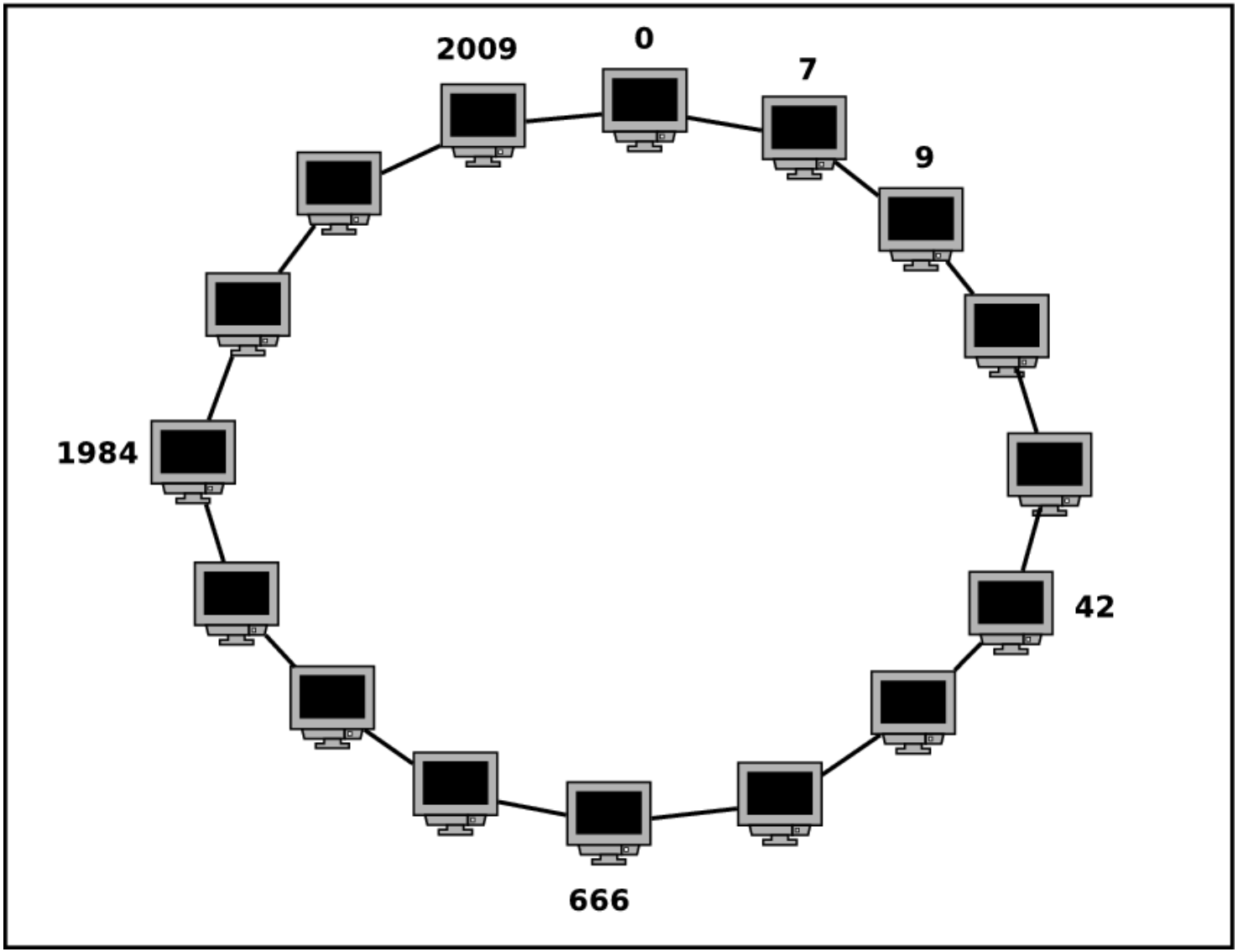
Motivation



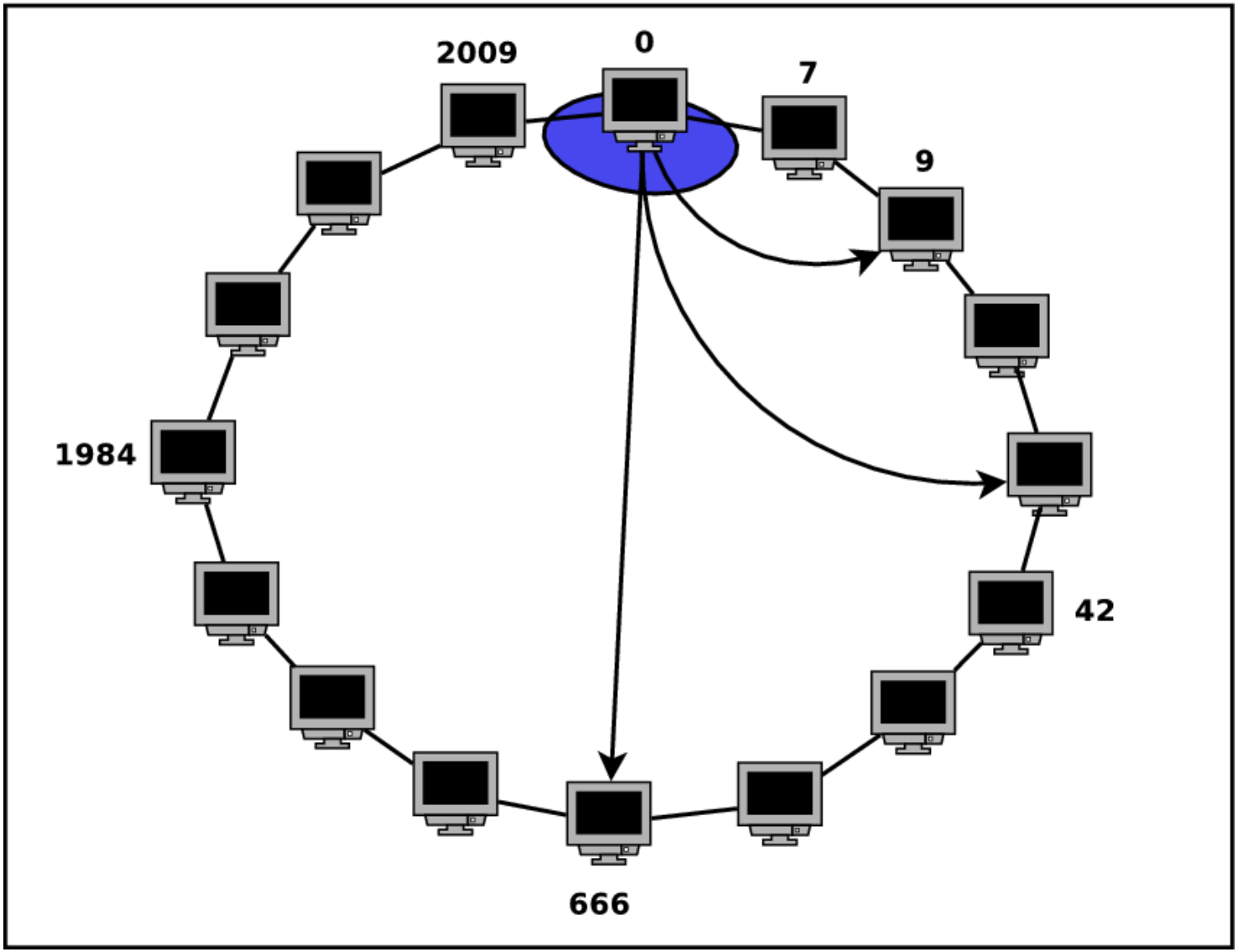
Motivation



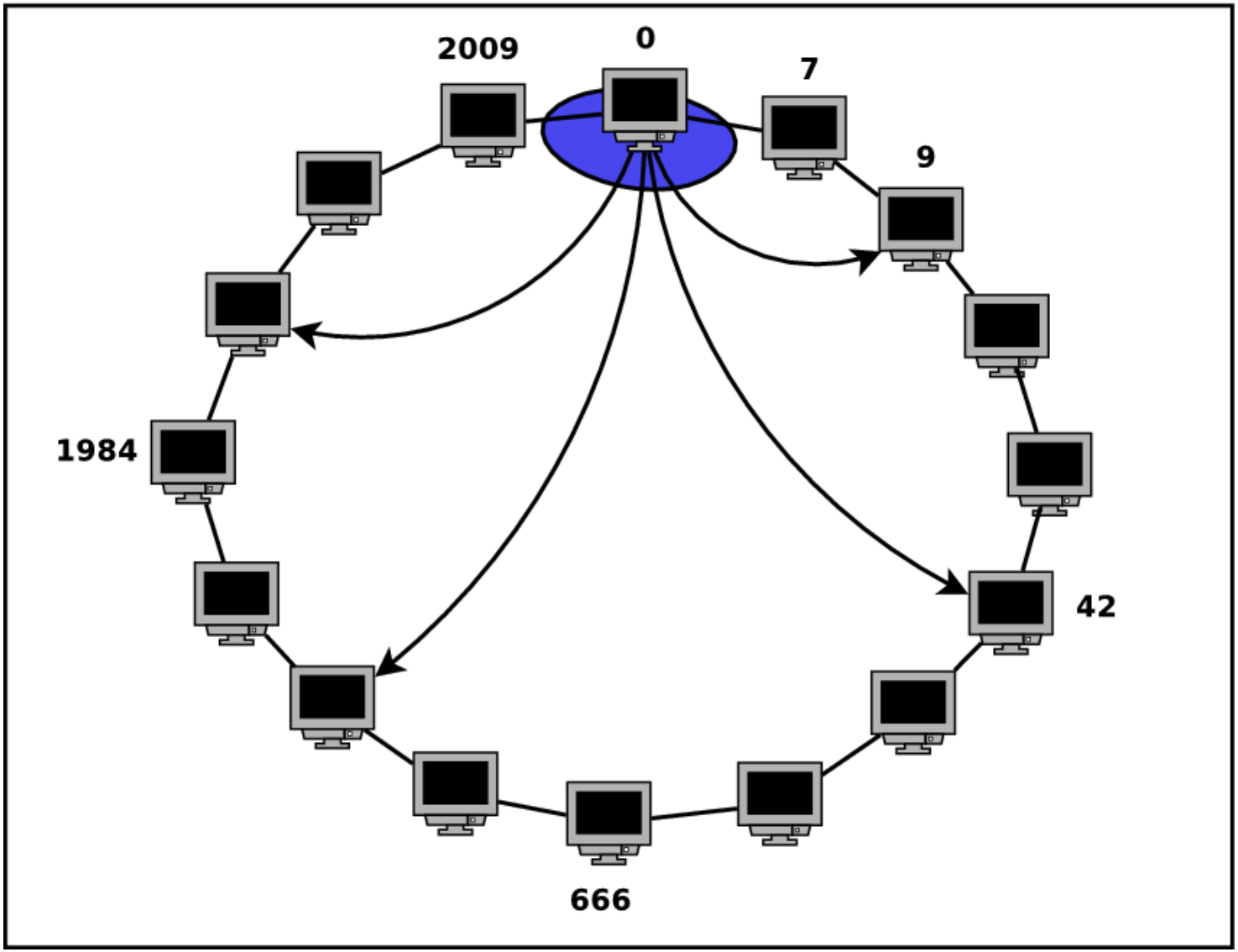
One ring to rule them all



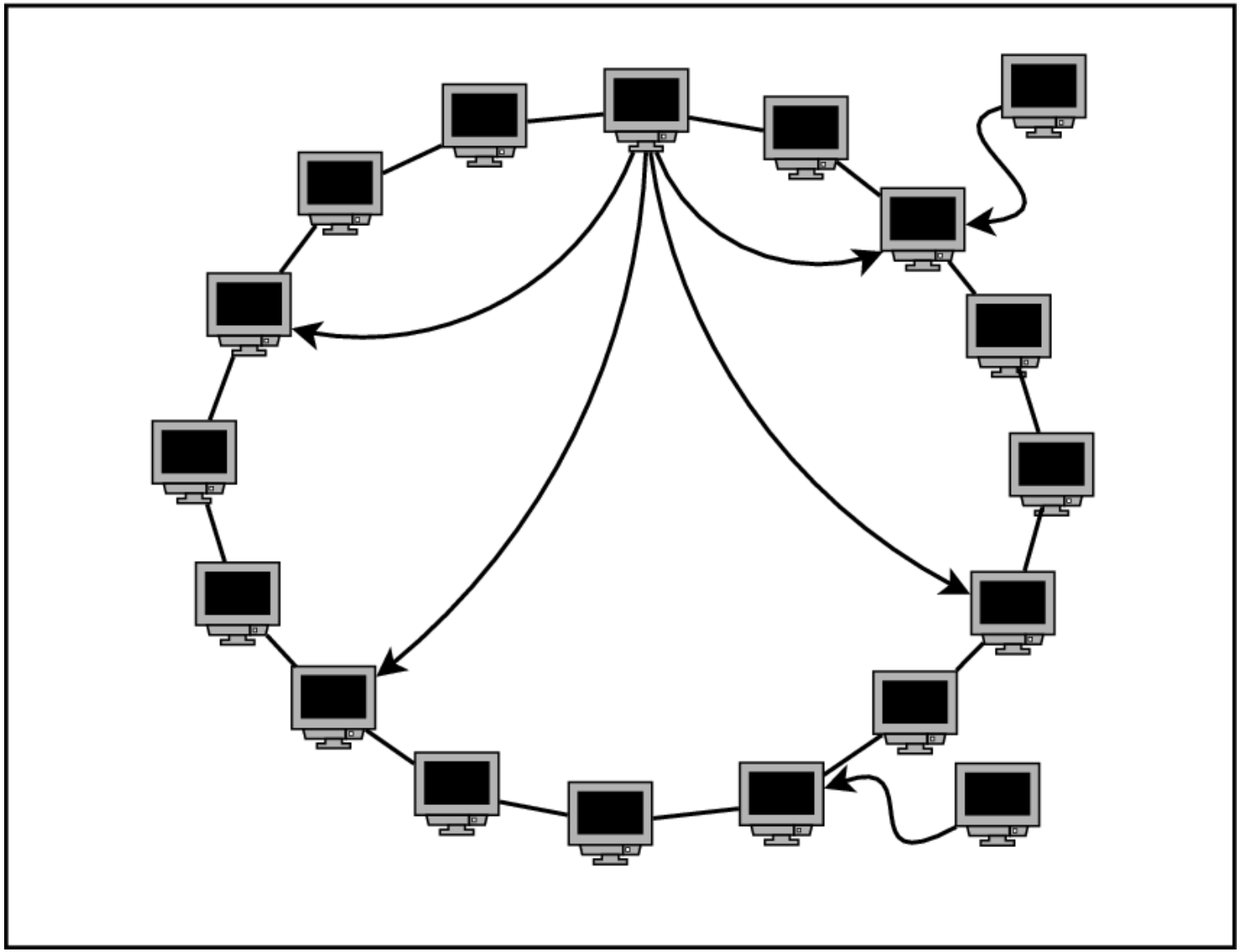
One ring to find them



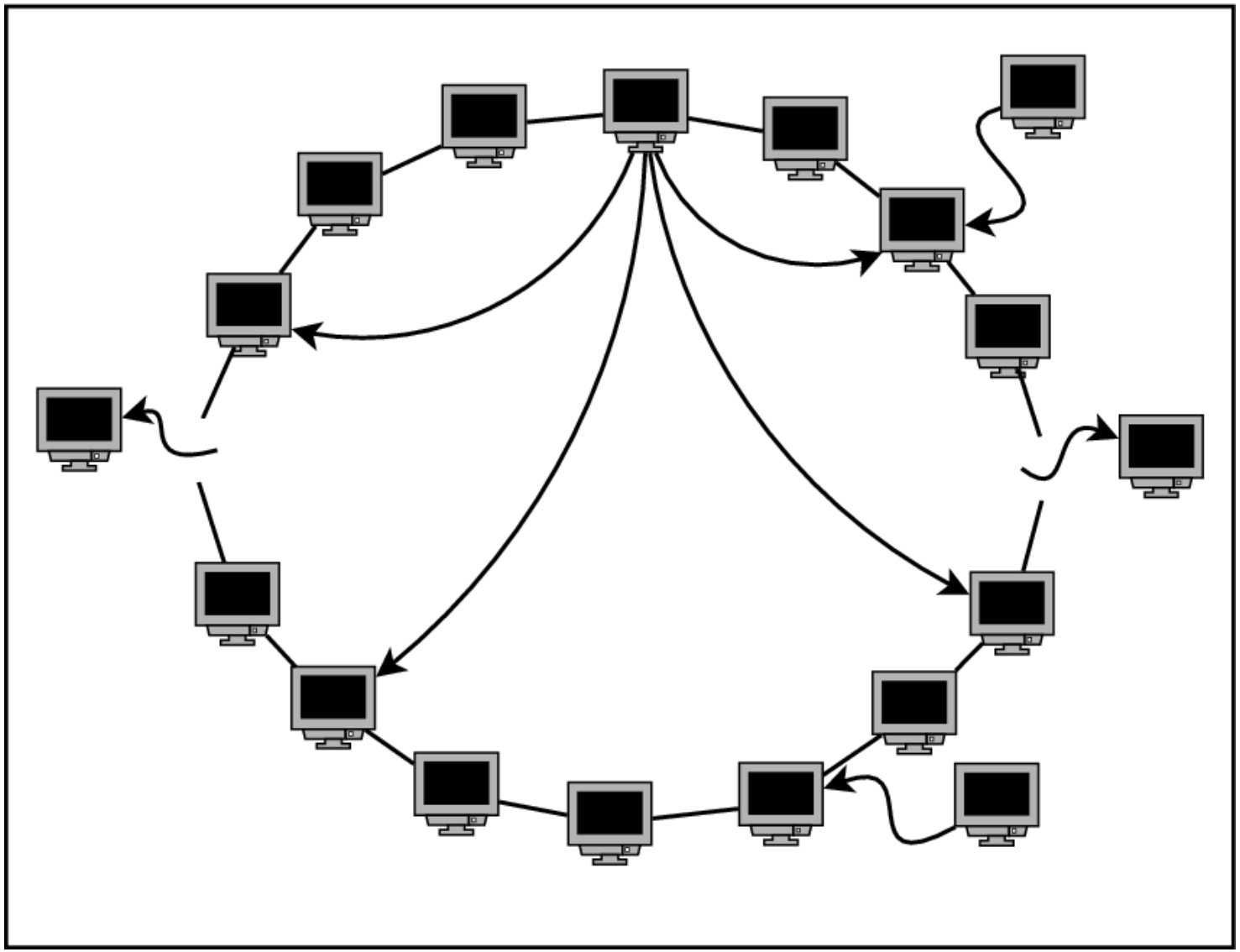
One ring to find them



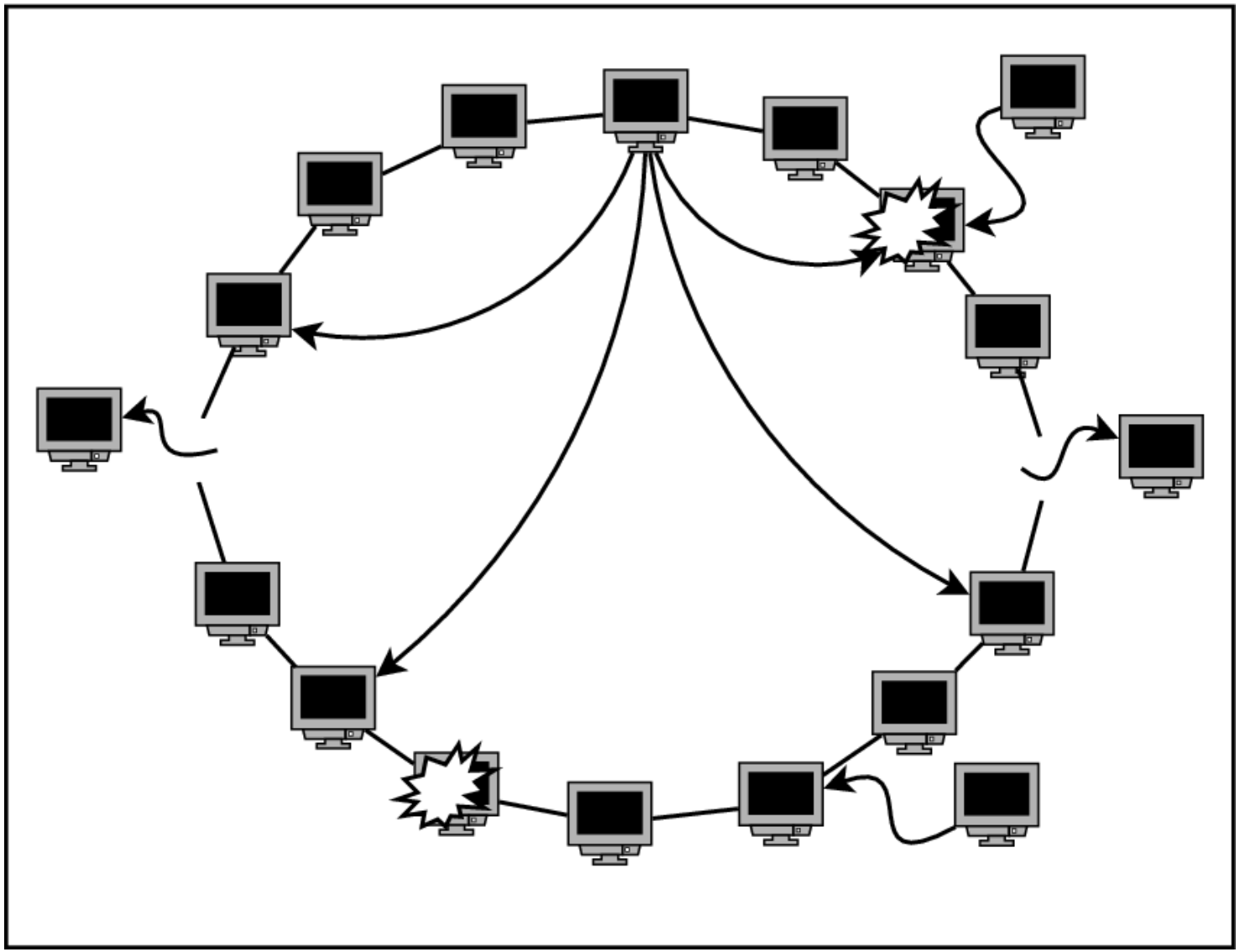
Motivation



Motivation



Motivation



Self-Management

- The system should be able to reconfigure itself to handle changes in its environment or its requirements without human intervention but according to high-level management policies
- Human intervention is lifted to the level of the policies

Self-Management

- The system should be able to reconfigure itself to handle changes in its environment or its requirements without human intervention but according to high-level management policies
- Human intervention is lifted to the level of the policies
- Typical self-management operations include:
  - add/remove nodes
  - tune performance
  - auto-configure
  - replicate data
  - failure detection and recovery
  - intrusion detection and recovery

Chord peer-to-peer network

- Scalable
- Provides a Distributed Hash Table (DHT)
- Fully decentralized
- Self-organized
- Efficient routing $O(\log(N))$

Chord peer-to-peer network

- Scalable
- Provides a Distributed Hash Table (DHT)
- Fully decentralized
- Self-organized
- Efficient routing $O(\log(N))$
- Lookup inconsistencies (due to churn)
- Expensive maintenance (periodic stabilization)

Chord peer-to-peer network

- Scalable
- Provides a Distributed Hash Table (DHT)
- Fully decentralized
- Self-organized
- Efficient routing $O(\log(N))$
- Lookup inconsistencies (due to churn)
- Expensive maintenance (periodic stabilization)

One solution: atomic join/leave

- Locks on successor, predecessor and the node

Chord peer-to-peer network

- Scalable
- Provides a Distributed Hash Table (DHT)
- Fully decentralized
- Self-organized
- Efficient routing $O(\log(N))$
- Lookup inconsistencies (due to churn)
- Expensive maintenance (periodic stabilization)

One solution: atomic join/leave

- Locks on successor, predecessor and the node
- Only two locks: the node and its successor (A. Ghodsi)

Chord peer-to-peer network

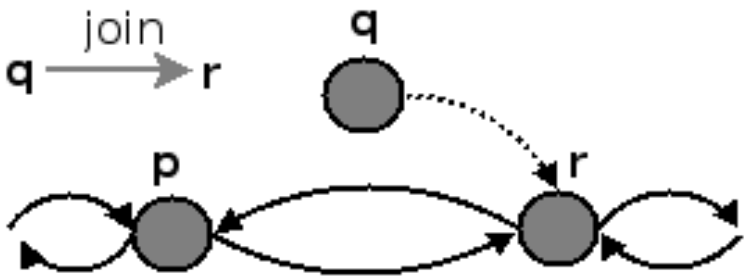
- Scalable
- Provides a Distributed Hash Table (DHT)
- Fully decentralized
- Self-organized
- Efficient routing $O(\log(N))$
- Lookup inconsistencies (due to churn)
- Expensive maintenance (periodic stabilization)

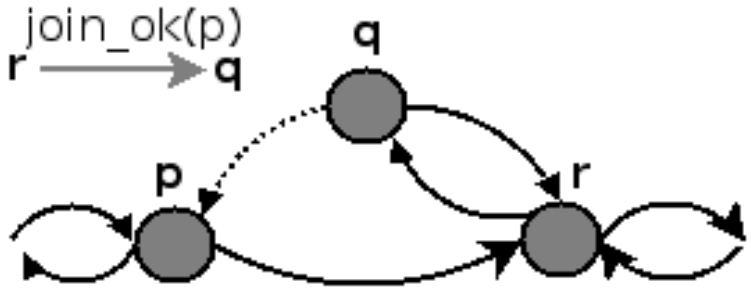
One solution: atomic join/leave

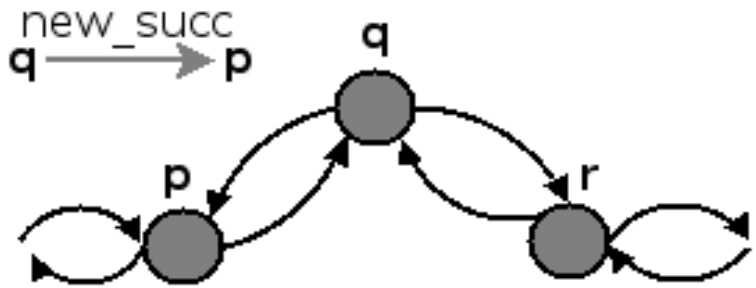
- Locks on successor, predecessor and the node
- Only two locks: the node and its successor (A. Ghodsi)
- and the failures?

The Relaxed-Ring

- joining peer *q* requests a lookup for its key
- *q* sends the *join* message to its successor candidate *r*
- *r* accepts new pred and sends reference *p* to *q*
- *q* contacts *p* to inform that it is its new succ

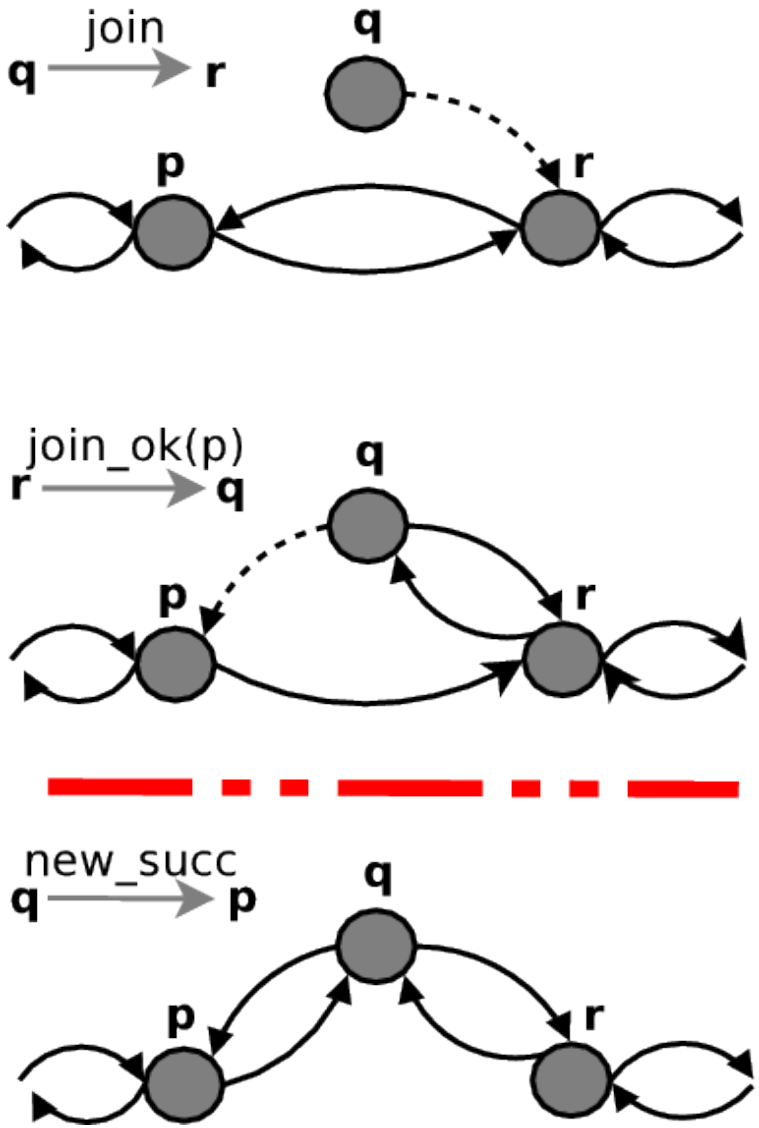






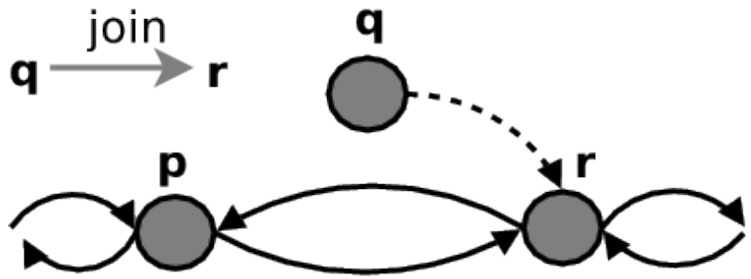
The Relaxed-Ring

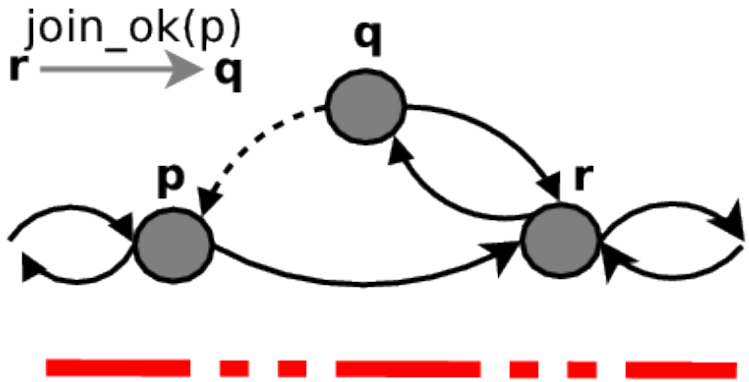
- joining peer *q* requests a lookup for its key
- *q* sends the *join* message to its successor candidate *r*
- *r* accepts new pred and sends reference *p* to *q*
- *q* contacts *p* to inform that it is its new succ

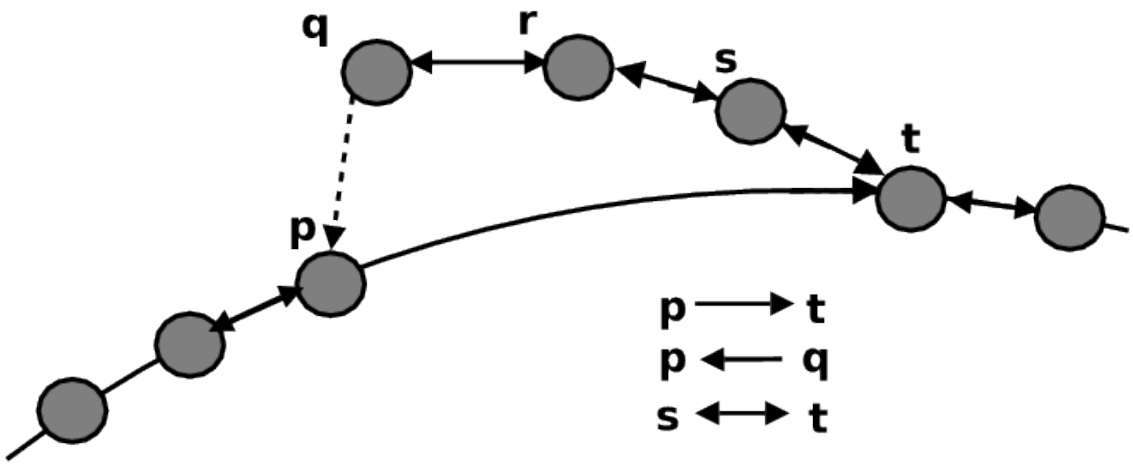


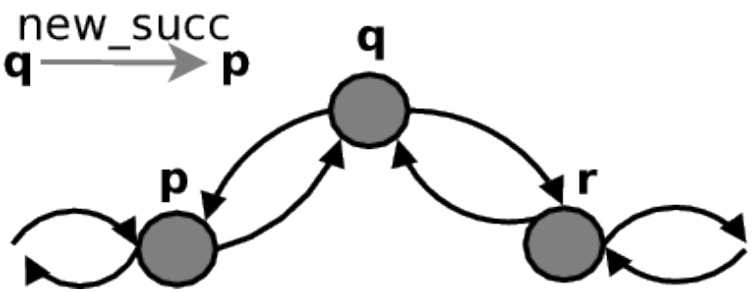
The Relaxed-Ring

- joining peer *q* requests a lookup for its key
- *q* sends the *join* message to its successor candidate *r*
- *r* accepts new pred and sends reference *p* to *q*
- *q* contacts *p* to inform that it is its new succ

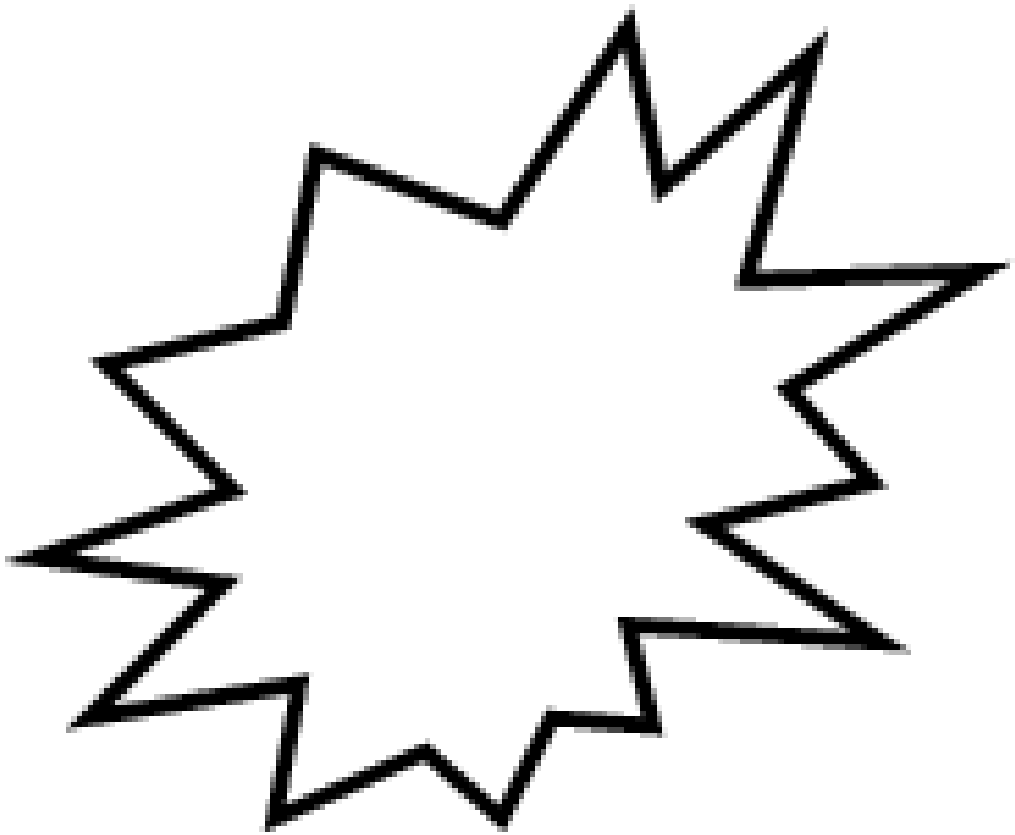






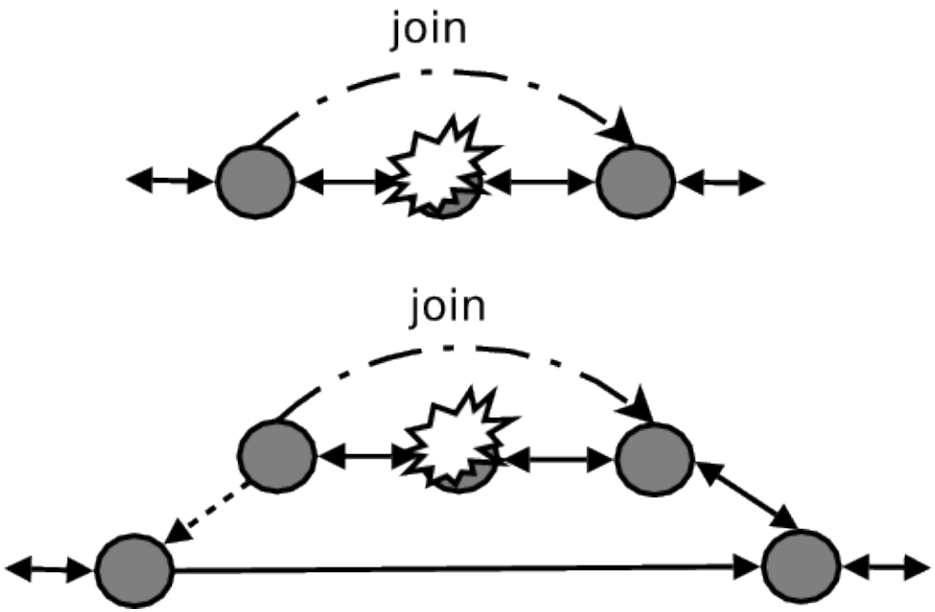


Failures



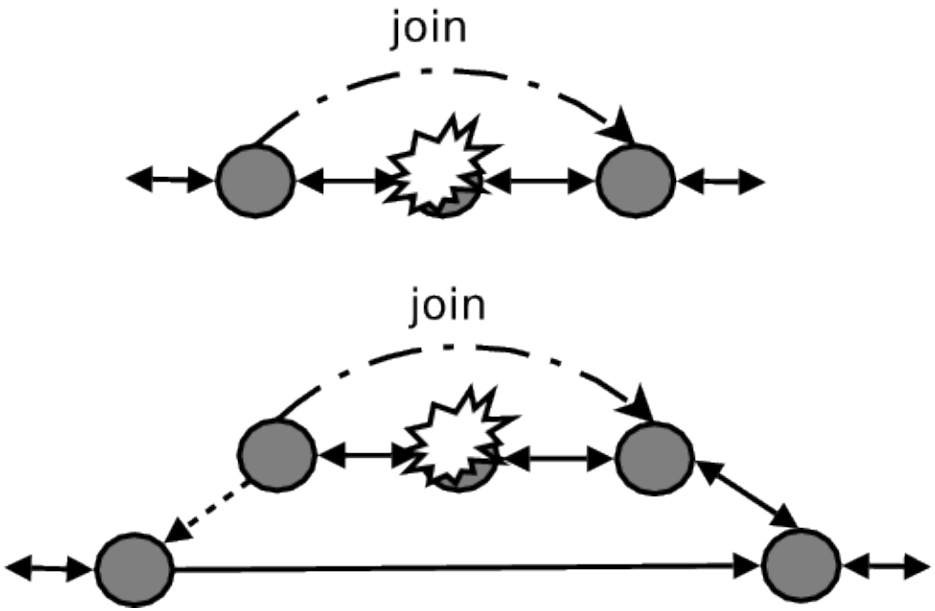
Failure Recovery

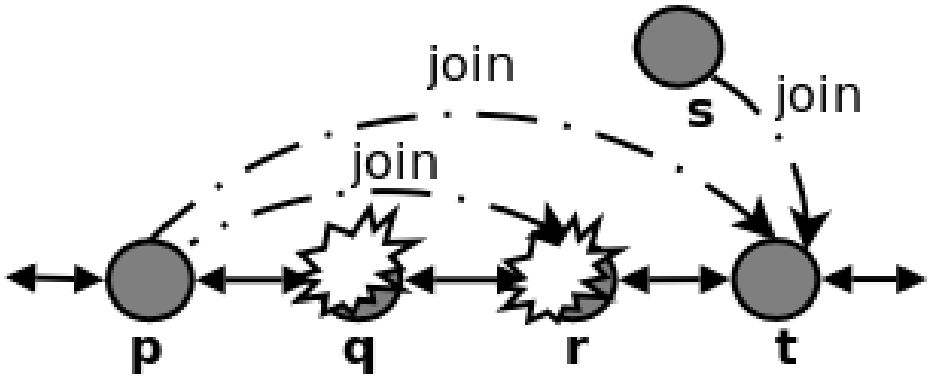
- When a peer detects that its successor has crashed, it contacts the first peer in its successor list for recovery



Failure Recovery

- When a peer detects that its successor has crashed, it contacts the first peer in its successor list for recovery





Relaxed-Ring

- Scalable
- Provides a Distributed Hash Table (DHT)
- Fully decentralized
- Self-organized
- Fault tolerant (Self-healing)

Relaxed-Ring

- Scalable
- Provides a Distributed Hash Table (DHT)
- Fully decentralized
- Self-organized
- Fault tolerant (Self-healing)
- Relies only on point-to-point link (no transitivity)
- join/fail algorithms requires the agreement of only two nodes (2 steps with 2 nodes, instead of 1 step with three)

Relaxed-Ring

- Scalable
- Provides a Distributed Hash Table (DHT)
- Fully decentralized
- Self-organized
- Fault tolerant (Self-healing)
- Relies only on point-to-point link (no transitivity)
- join/fail algorithms requires the agreement of only two nodes (2 steps with 2 nodes, instead of 1 step with three)
- Almost no lookup-inconsistency
- Cost-efficient ring maintenance (no periodic stabilization)

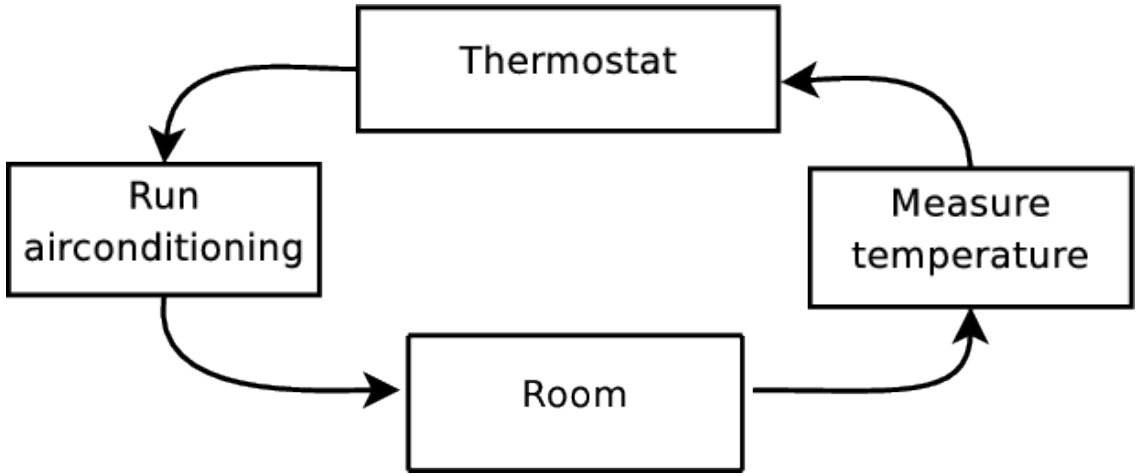
Relaxed-Ring

- Scalable
- Provides a Distributed Hash Table (DHT)
- Fully decentralized
- Self-organized
- Fault tolerant (Self-healing)
- Relies only on point-to-point link (no transitivity)
- join/fail algorithms requires the agreement of only two nodes (2 steps with 2 nodes, instead of 1 step with three)
- Almost no lookup-inconsistency
- Cost-efficient ring maintenance (no periodic stabilization)
- Efficient routing $O(\log(N) + b)$

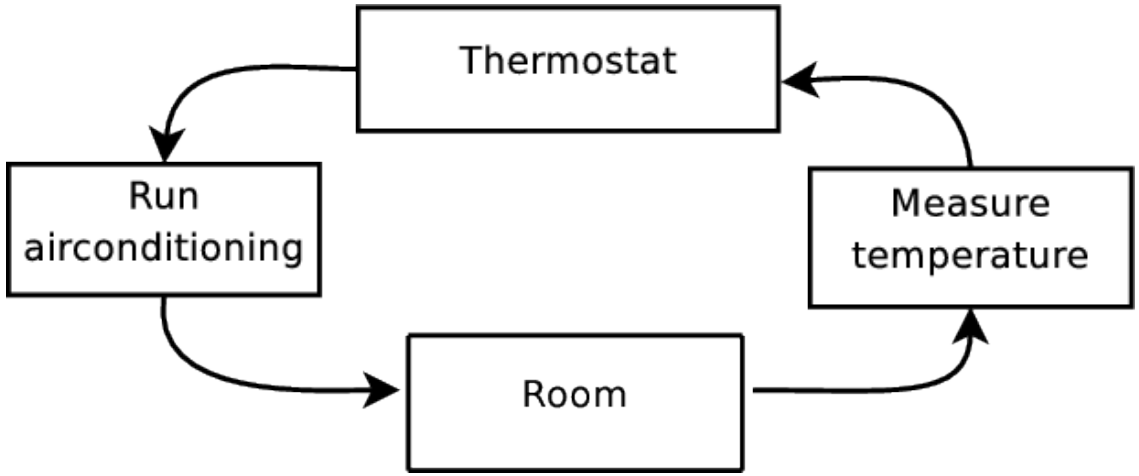
Relaxed-Ring

- Scalable
- Provides a Distributed Hash Table (DHT)
- Fully decentralized
- Self-organized
- Fault tolerant (Self-healing)
- Relies only on point-to-point link (no transitivity)
- join/fail algorithms requires the agreement of only two nodes (2 steps with 2 nodes, instead of 1 step with three)
- Almost no lookup-inconsistency
- Cost-efficient ring maintenance (no periodic stabilization)
- Efficient routing $O(\log(N) + b)$
- PALTA self-adapting from fully connected to relaxed-ring
- It handles peers with limited connectivity

Feedback Loop

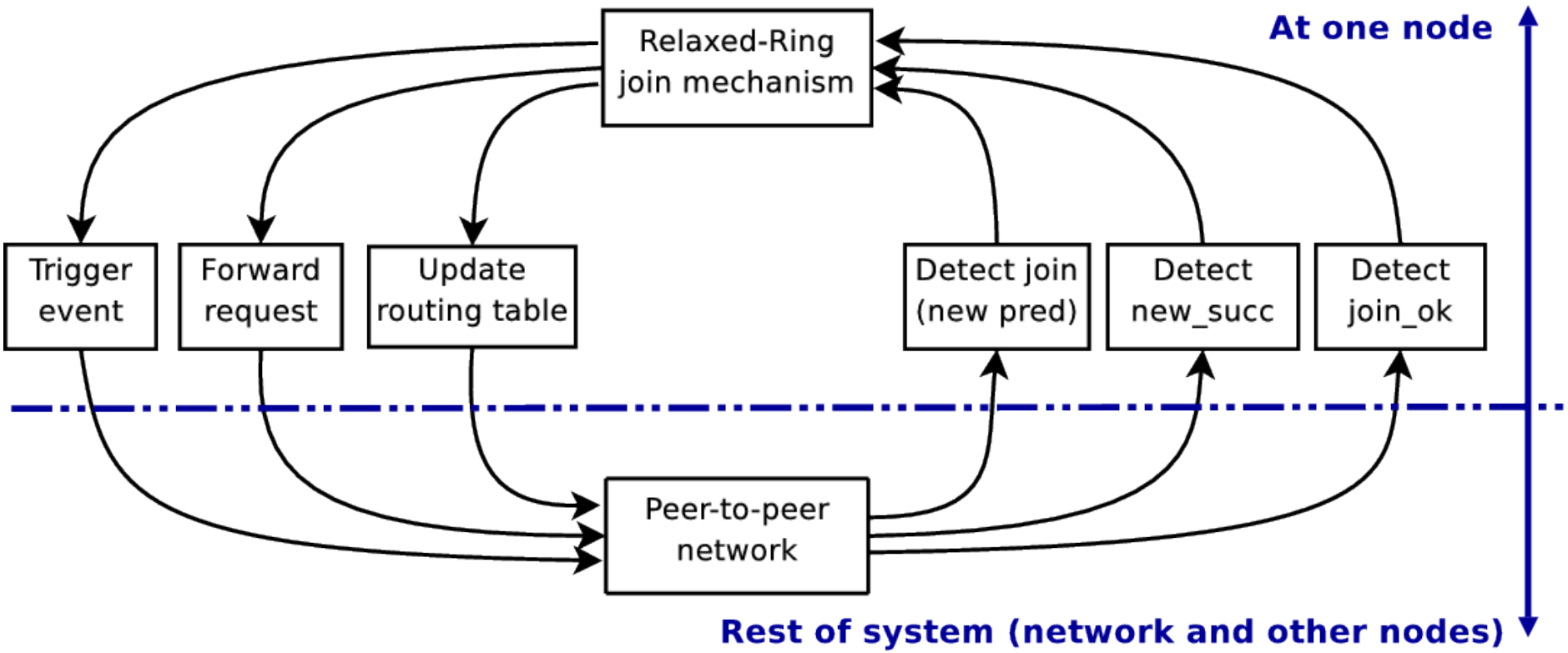


Feedback Loop



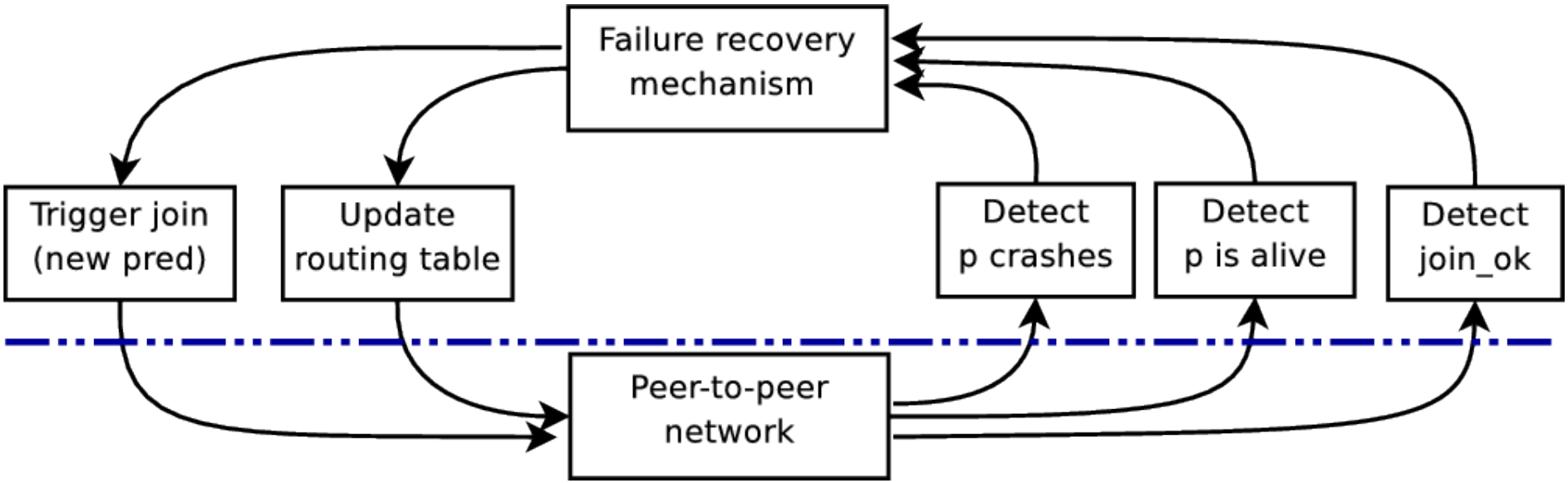
- Sub-system is constantly monitored
- One or more components in charge of calculating correction
- Actuators perform corrective actions
- Convergence to a stable global state

Join Algorithm as a Feedback Loop



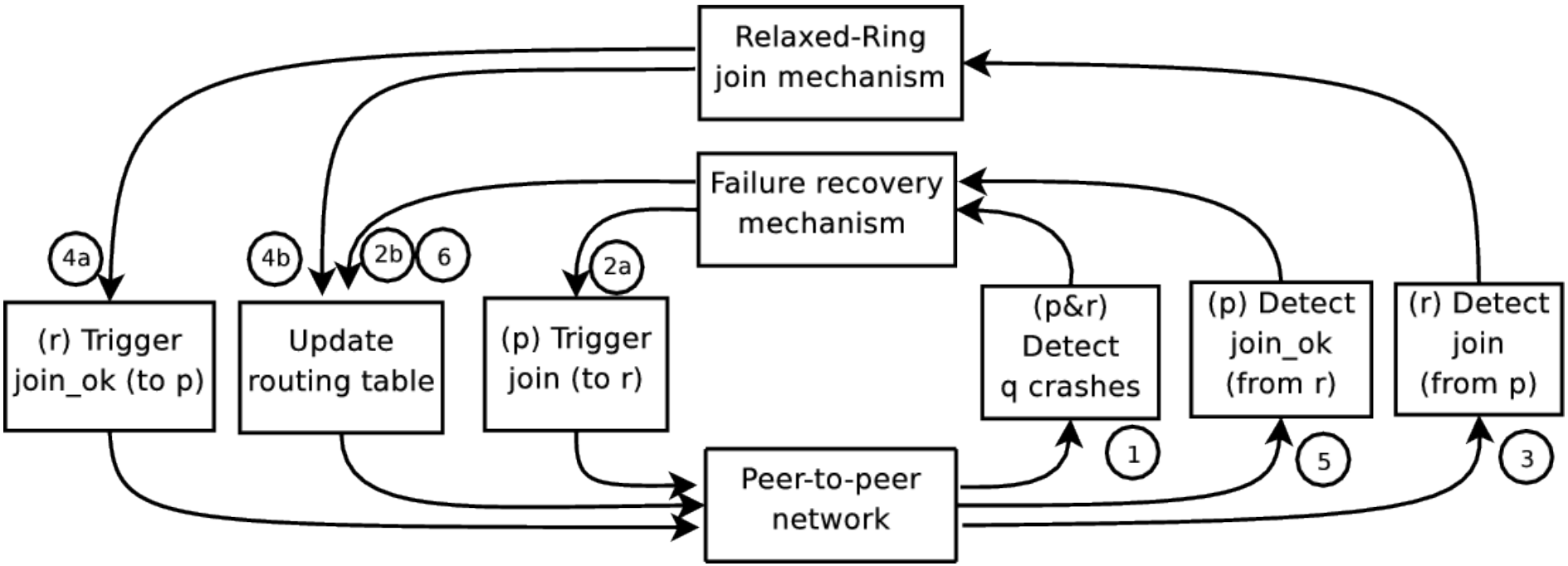
- Events perturbing the stability of the ring are constantly monitored
- Corresponding corrective actions are triggered
- Resilient information (successor list) is always updated

Failure Recovery as a Feedback Loop



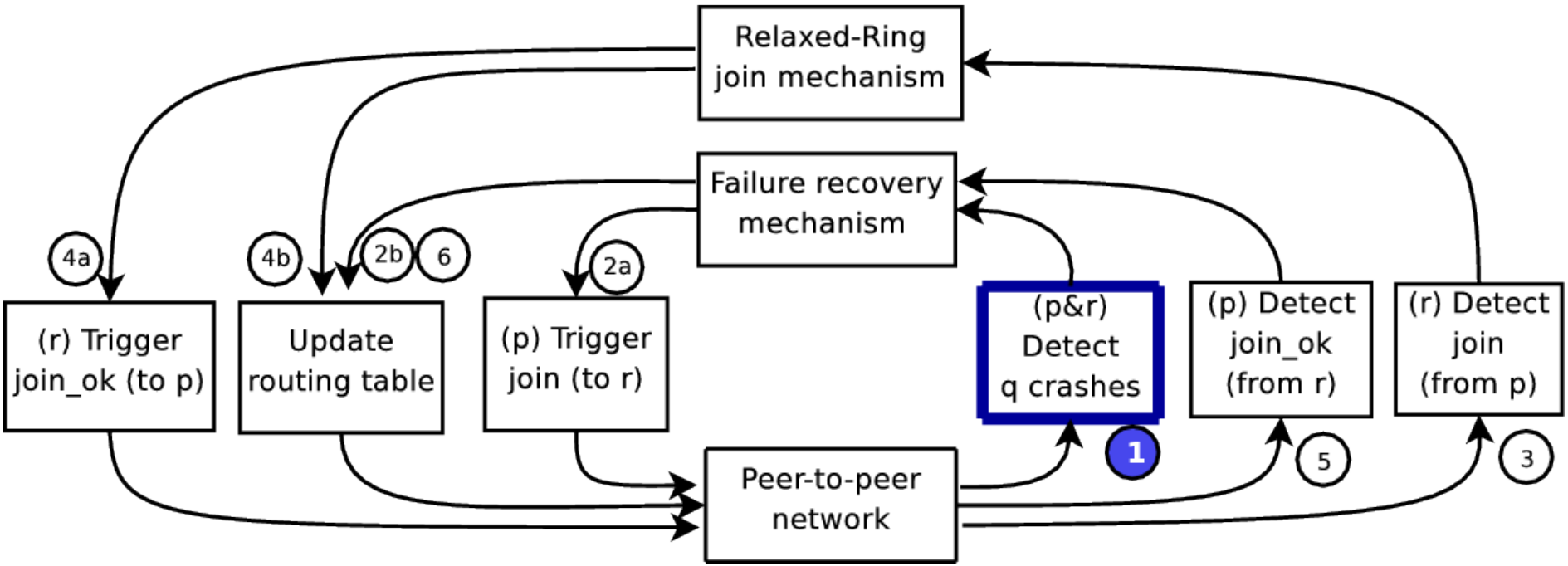
- Failures are perturbations on the relaxed-ring topology
- Join mechanism is triggering for fixing

Interaction of Loops in the Recovery Mechanism



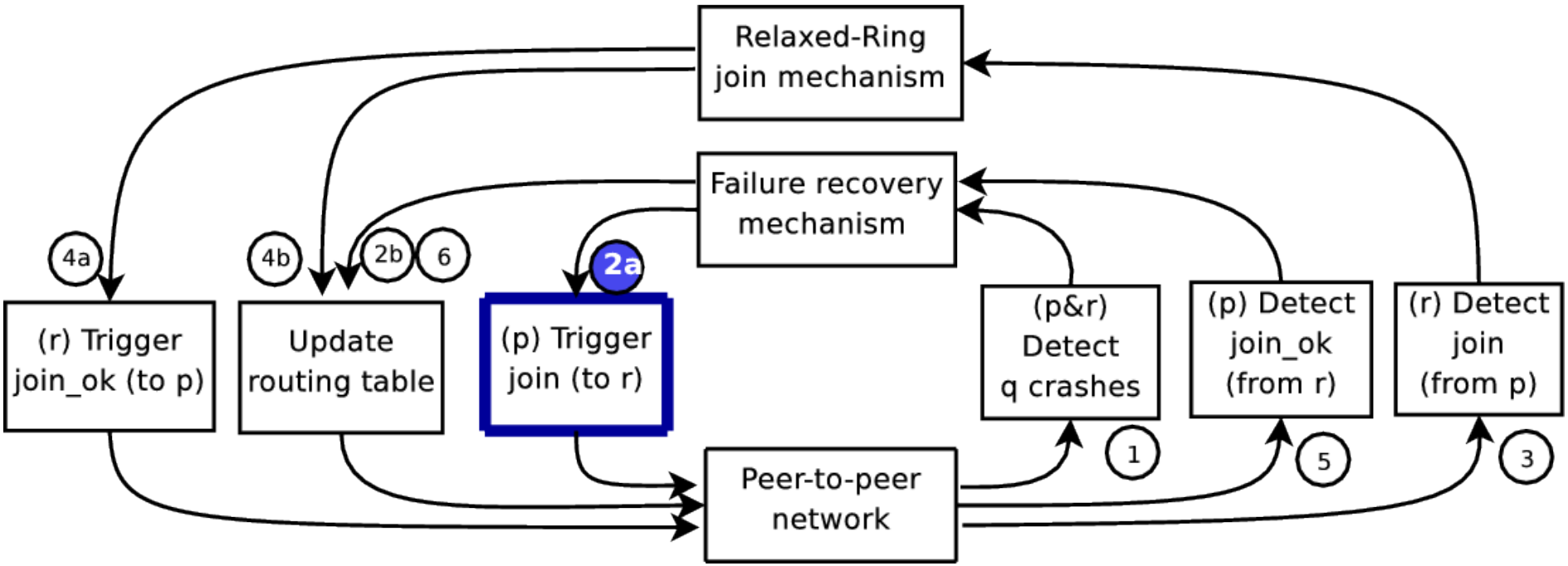
- Peers *p* and *r* detect failure of *q*, fixing the ring with an interaction of feedback loops

Interaction of Loops in the Recovery Mechanism



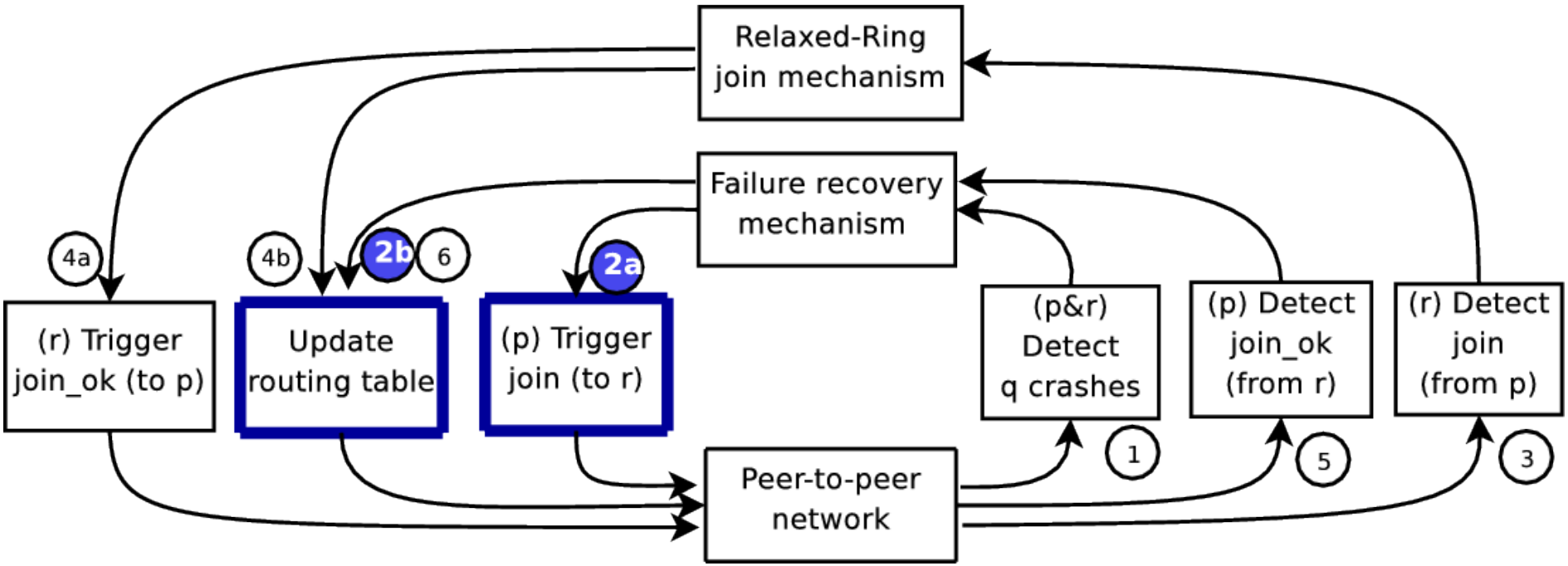
- Peers *p* and *r* detect failure of *q*, fixing the ring with an interaction of feedback loops

Interaction of Loops in the Recovery Mechanism



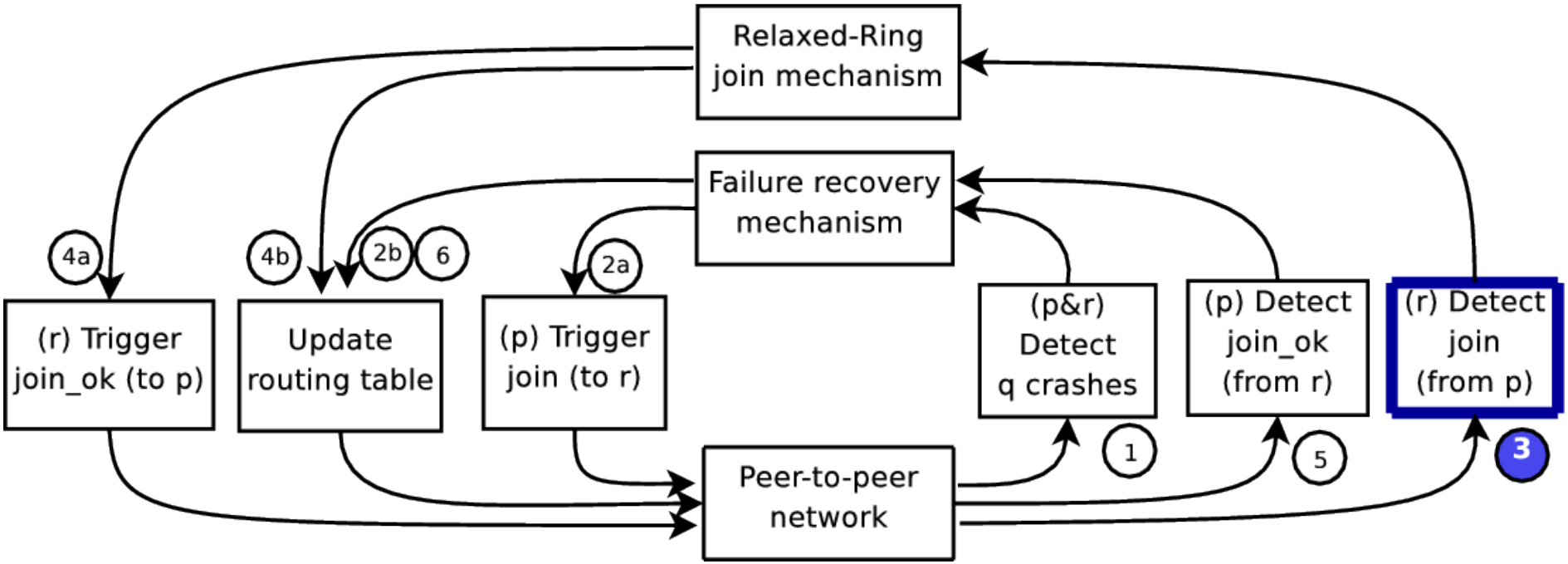
- Peers *p* and *r* detect failure of *q*, fixing the ring with an interaction of feedback loops

Interaction of Loops in the Recovery Mechanism



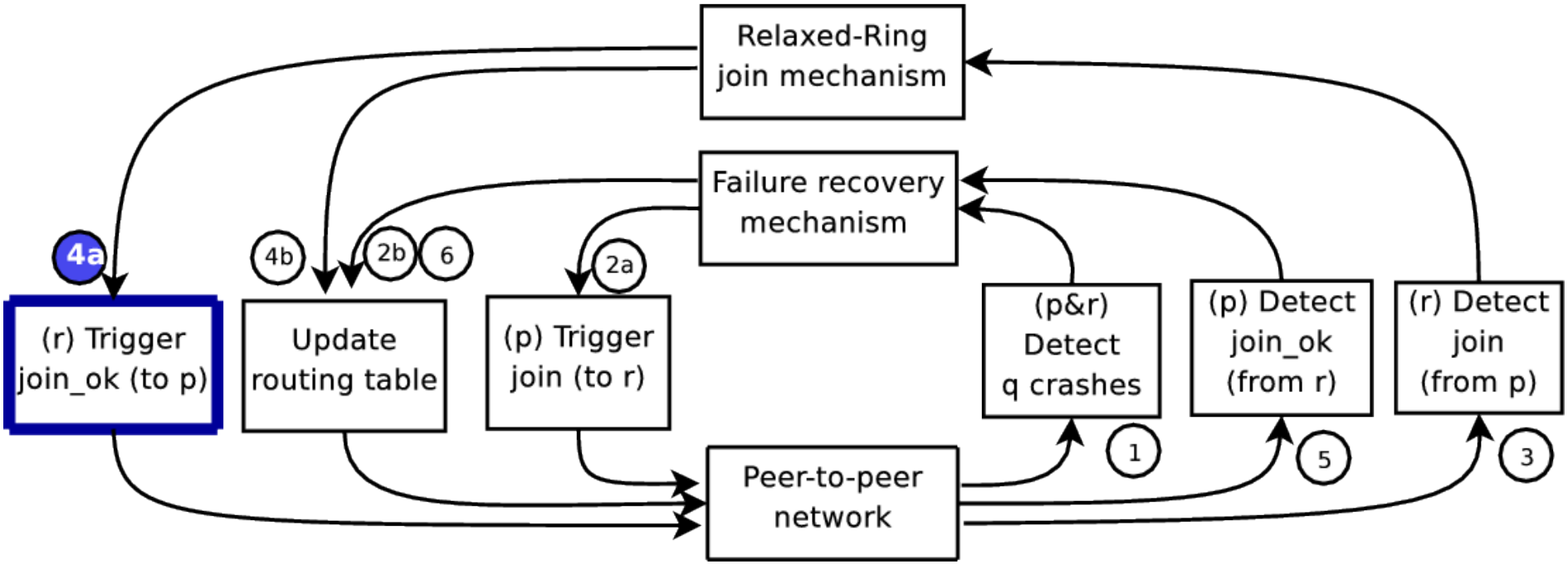
- Peers *p* and *r* detect failure of *q*, fixing the ring with an interaction of feedback loops

Interaction of Loops in the Recovery Mechanism



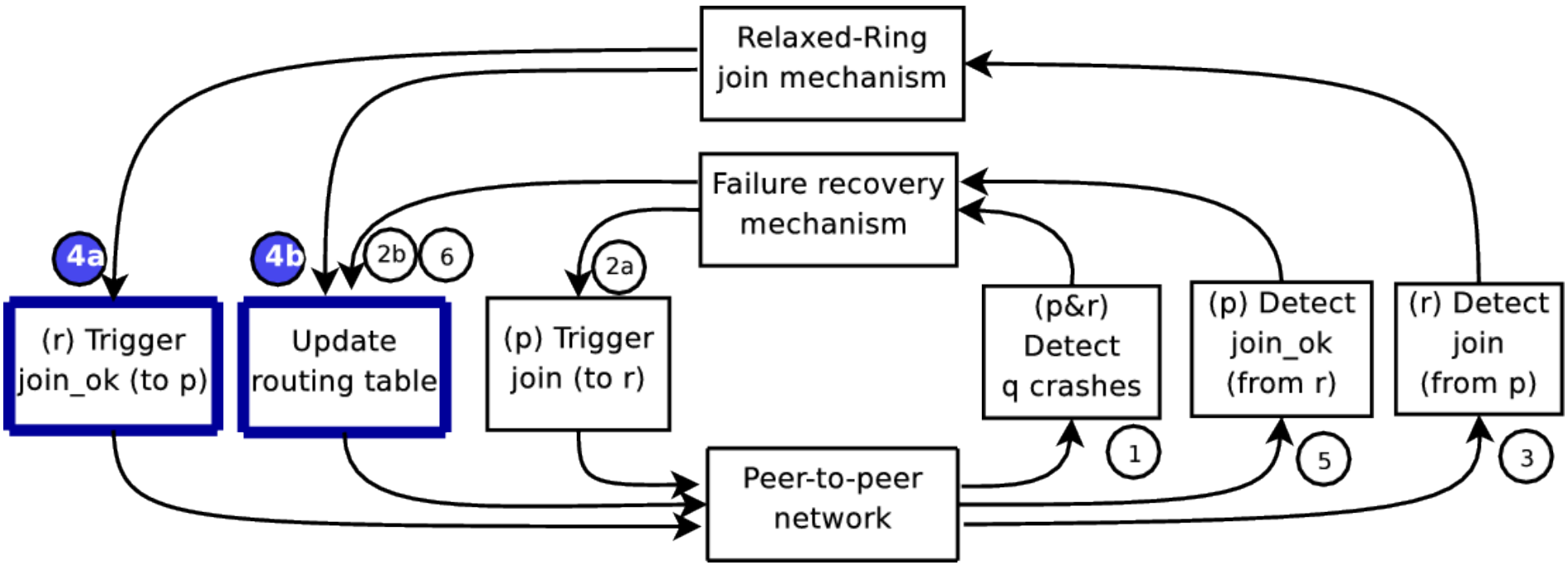
- Peers *p* and *r* detect failure of *q*, fixing the ring with an interaction of feedback loops

Interaction of Loops in the Recovery Mechanism



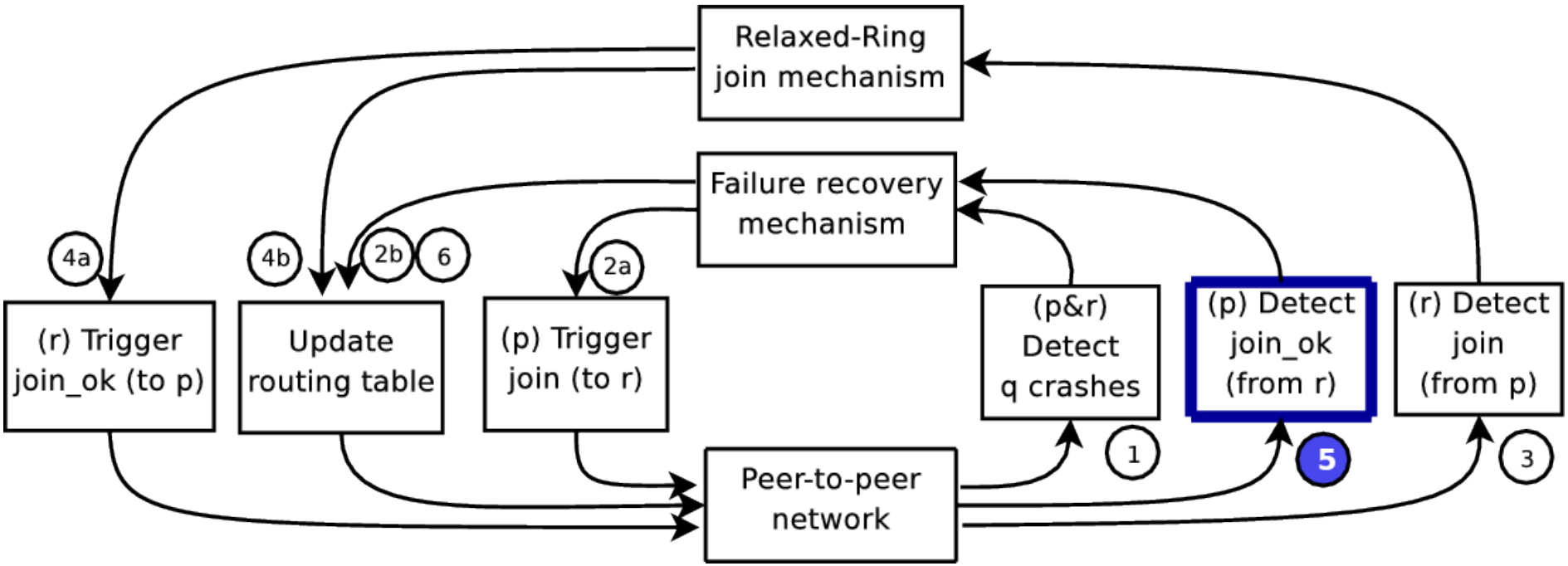
- Peers *p* and *r* detect failure of *q*, fixing the ring with an interaction of feedback loops

Interaction of Loops in the Recovery Mechanism



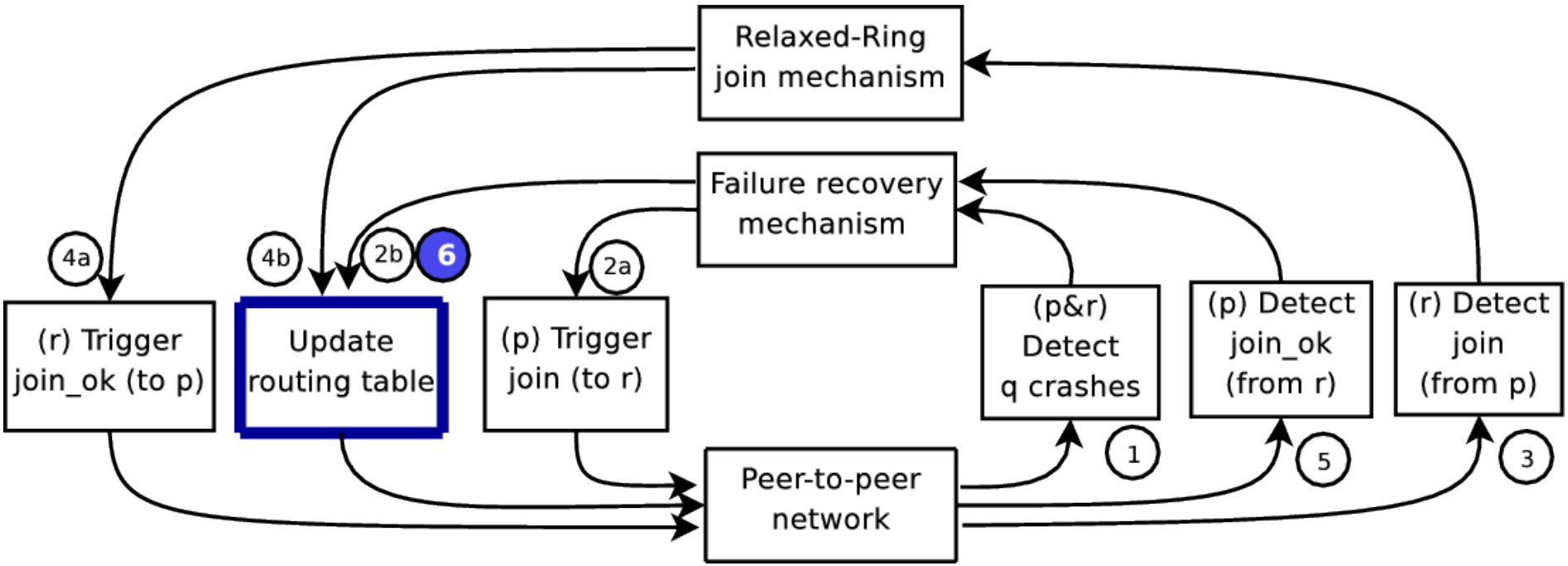
- Peers *p* and *r* detect failure of *q*, fixing the ring with an interaction of feedback loops

Interaction of Loops in the Recovery Mechanism



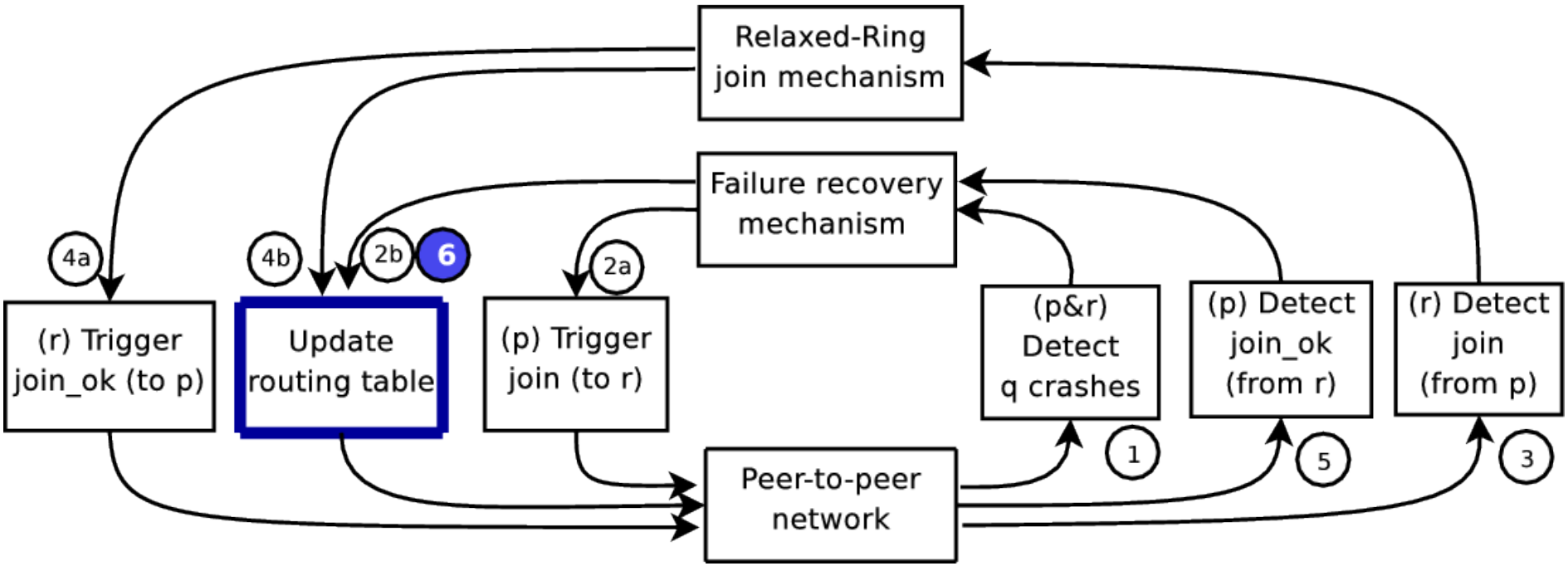
- Peers *p* and *r* detect failure of *q*, fixing the ring with an interaction of feedback loops

Interaction of Loops in the Recovery Mechanism



- Peers *p* and *r* detect failure of *q*, fixing the ring with an interaction of feedback loops

Interaction of Loops in the Recovery Mechanism



- Peers *p* and *r* detect failure of *q*, fixing the ring with an interaction of feedback loops
- Solving failure handling there is no need for handling *leaves*. They are just *explicit* failures

Beernet

- Mozart-Oz implementation of the relaxed-ring (P2PS's successor)
- Component-based architecture
- Event-driven algorithms with asynchronous message passing and no shared state concurrency
- DHT + Symmetric replication
- Transaction layer with Paxos consensus algorithm (M. Moser)
- Survives network partition
- Ring merge based on gossip algorithm (T. Mahmood)

Conclusion

- The Relaxed-Ring
  - Consistent lookup
  - Realistic failure detection
  - Self-organisation: It handles joins/leaves of peers re-organising the network in a autonomous fashion.
  - Self-healing: It recovers from failures and survives network partitioning.
  - Algorithms designed using feedback-loops
  - Beernet implementation with transactional layer

Conclusion and Future Work

- The Relaxed-Ring
  - Consistent lookup
  - Realistic failure detection
  - Self-organisation: It handles joins/leaves of peers re-organising the network in a autonomous fashion.
  - Self-healing: It recovers from failures and survives network partitioning.
  - Algorithms designed using feedback-loops
  - Beernet implementation with transactional layer
- TODO
  - Implement an application with Beernet to conquer the world
  - Work on reversible phase transitions

Last slide where I should ask for *questions*
or simply put a big *question mark*