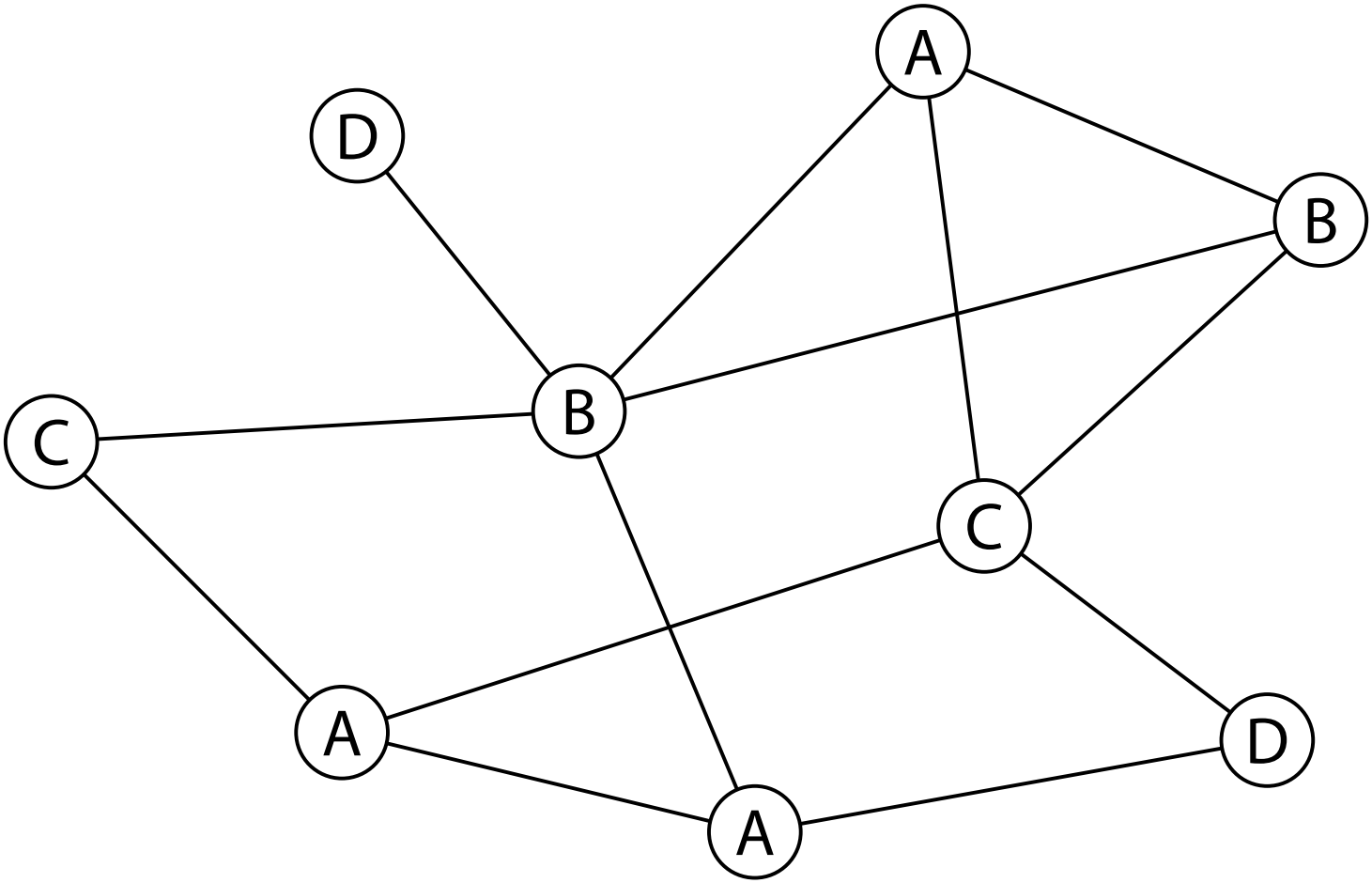
Mediated Population Protocols: Leader Election and Applications

TAMC 2017

Shantanu Das, Giuseppe Antonio Di Luna, Paola Flocchini, Nicola Santoro, Giovanni Viglietta

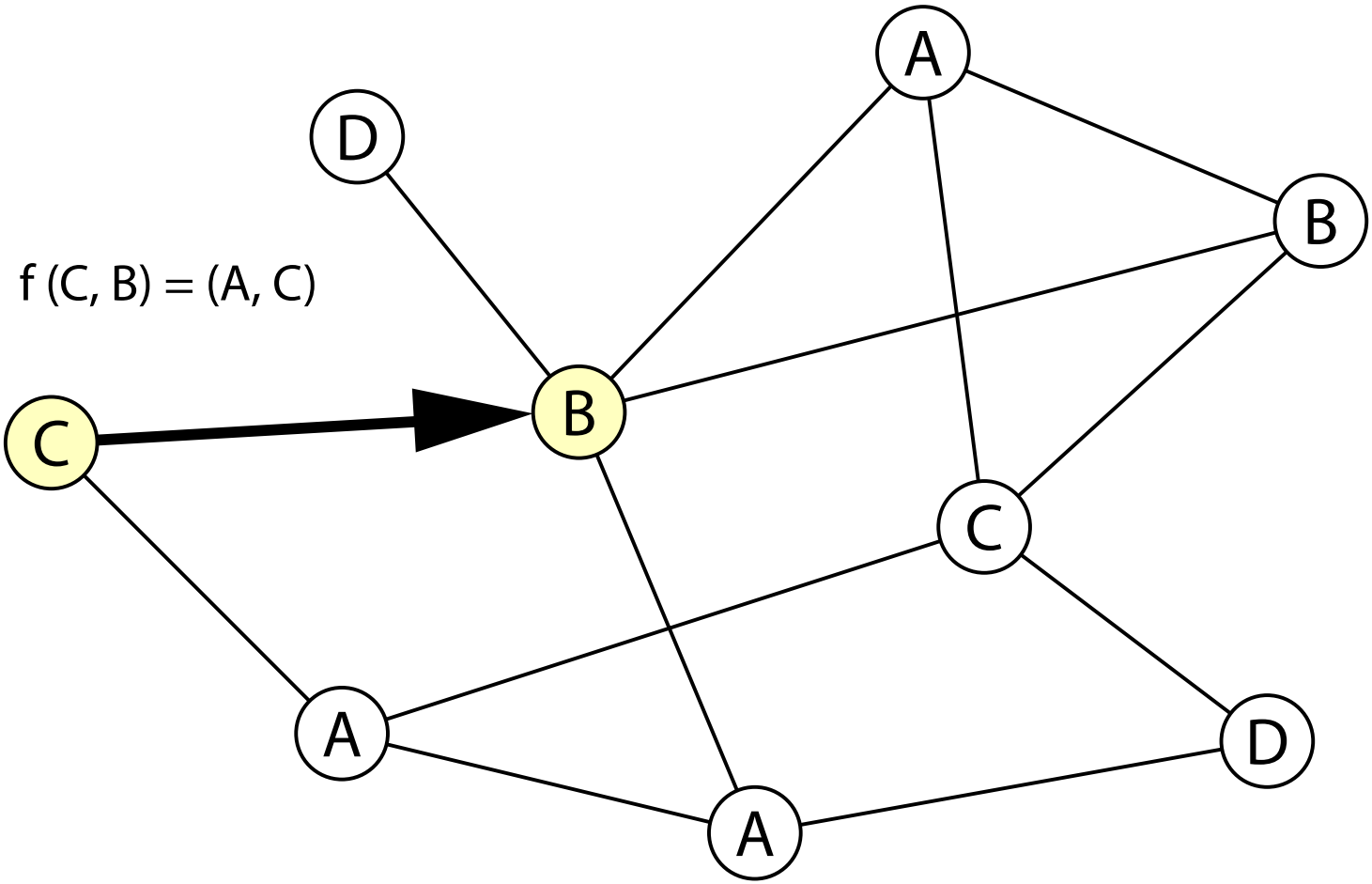
Bern – April 20, 2017

Population Protocol



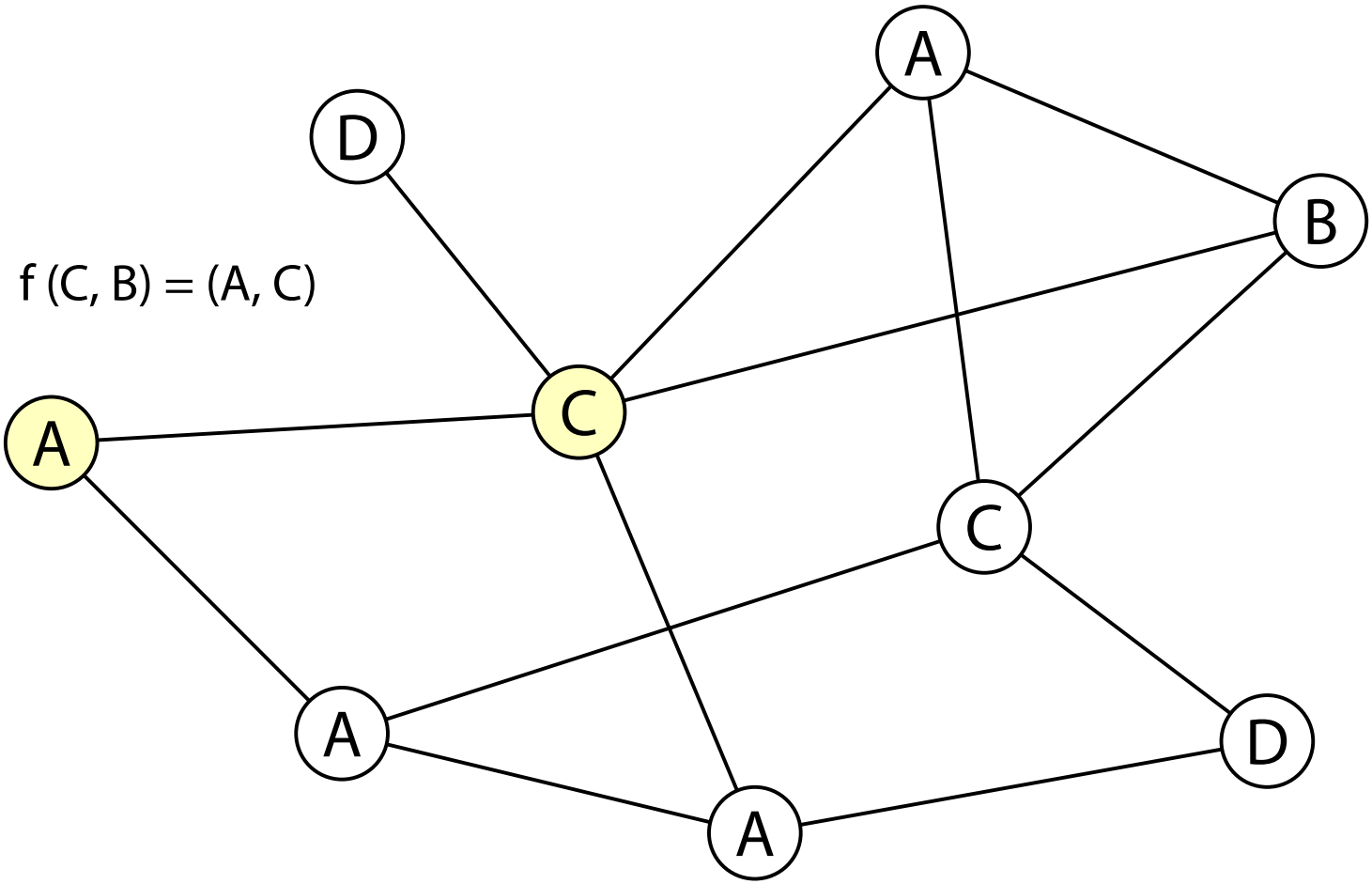
Setting: a network of finite-state agents.

Population Protocol



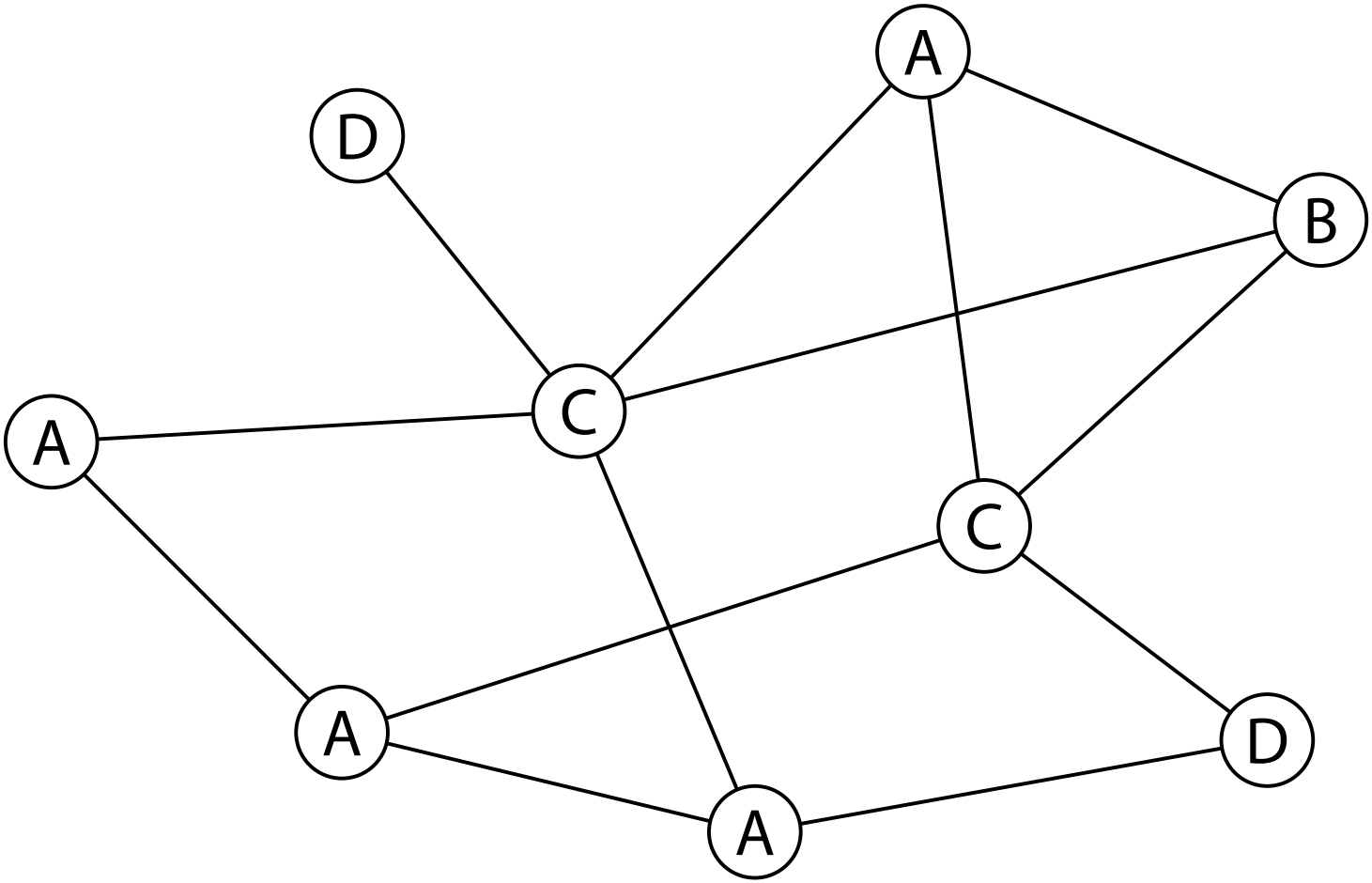
Pairs of adjacent agents interact in a non-deterministic order...

Population Protocol



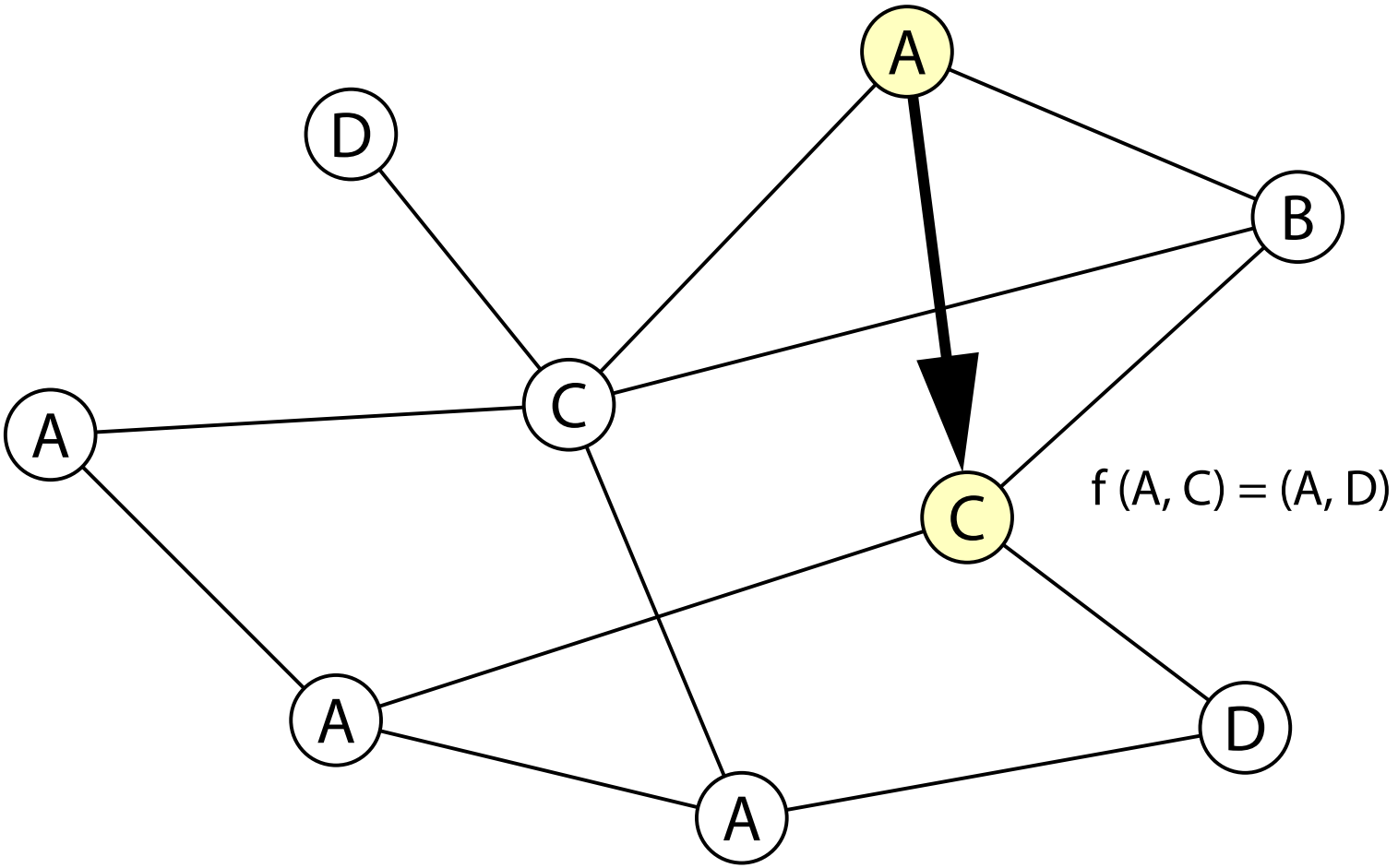
...and change states according to a transition function.

Population Protocol



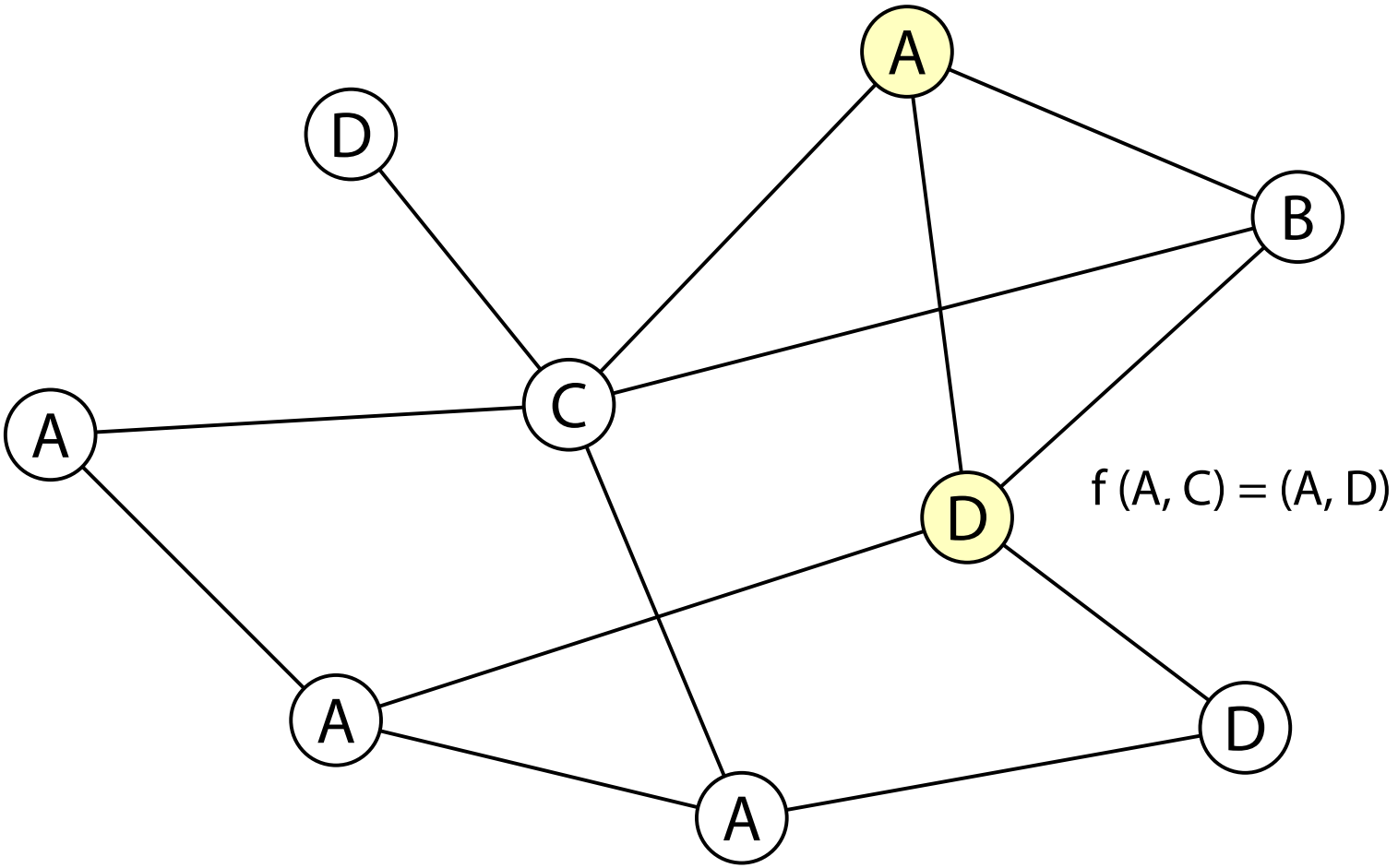
...and change states according to a transition function.

Population Protocol



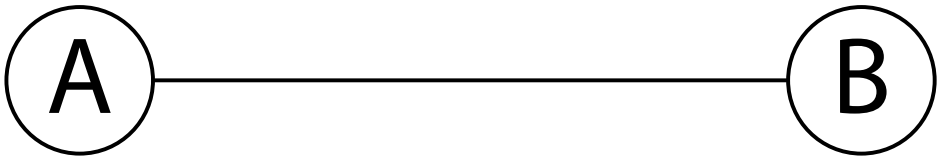
...and change states according to a transition function.

Population Protocol



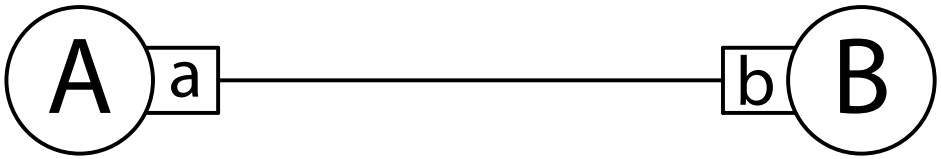
...and change states according to a transition function.

Mediated Population Protocol



Mediated agents: we add ports with (finite) states.

Mediated Population Protocol



Mediated agents: we add ports with (finite) states.

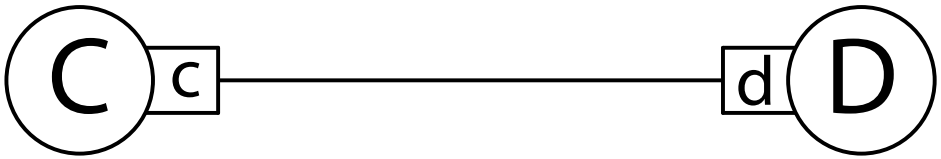
Mediated Population Protocol

$$f(A, B, a, b) = (C, D, c, d)$$

The transition function affects both agent and port states.

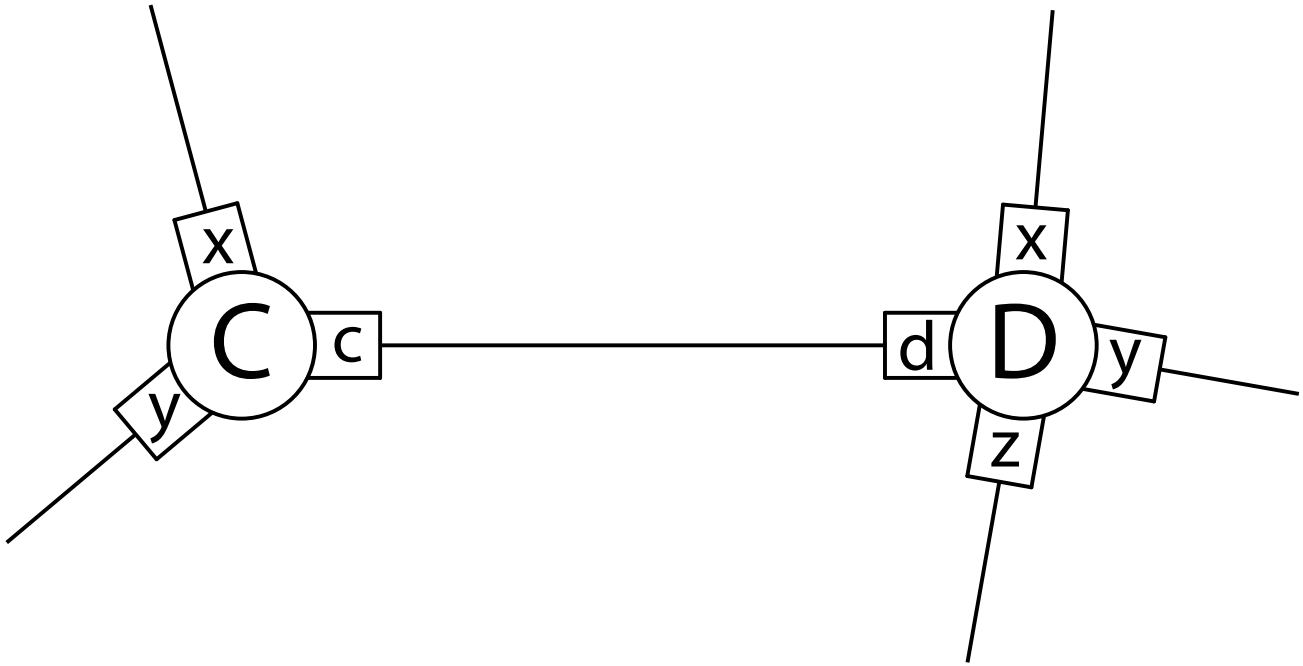
Mediated Population Protocol

$$f(A, B, a, b) = (C, D, c, d)$$



The transition function affects both agent and port states.

Mediated Population Protocol



Each agent has a port for each neighbor.

Schedulers

We distinguish two types of scheduler:

- **Recurrent:** each pair of neighboring agents interacts infinitely often (in both directions)
- k-**Bounded:** it is recurrent and, between two consecutive interactions of the same pair of agents, no other pair interacts more than k times

Note: under a 1-bounded scheduler, the sequence of interactions is periodic

Stability and Termination

A protocol can be:

- **Stable:** eventually, no agent changes state
- **Terminating:** eventually, all agents are in a *terminal state* (i.e., "explicit stability")

Usually, with the recurrent scheduler our protocols are stable; with the k-bounded schedulers, they are terminating

Paper Overview

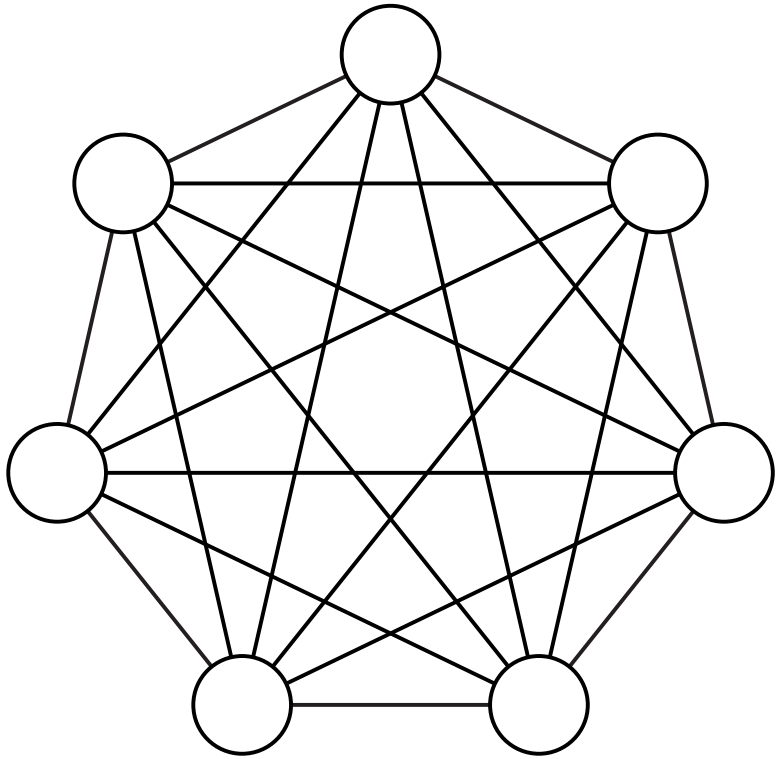
Leader Election Problem: all agents start in the same state, and eventually there is a unique agent in a *leader state*

- Complete graphs
- Complete bipartite graphs
- Trees

Applications of a Unique Leader:

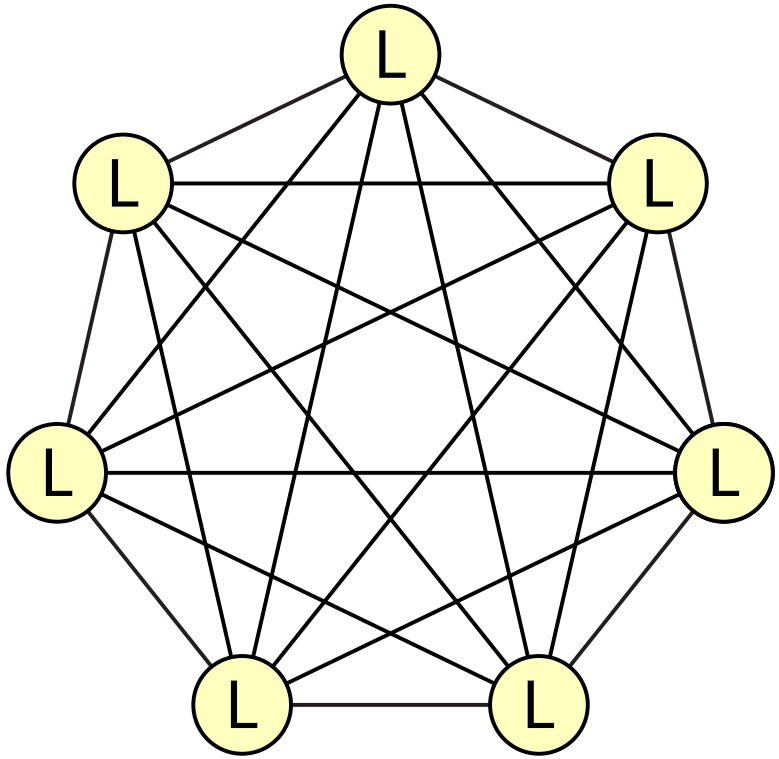
- Token circulation
- Construction of a shortest-path spanning tree
- Stability detection (turning stable protocols into terminating ones)
- Equivalence of k-bounded schedulers for all $k > 1$

Leader Election in a Complete Graph



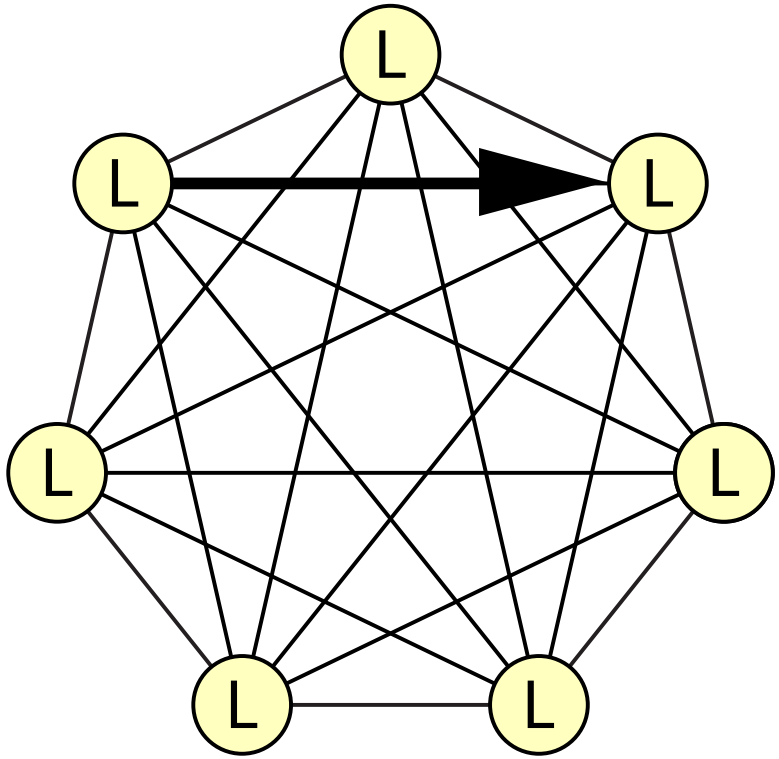
Theorem: in the complete graph K_n, it is possible to elect a leader under the recurrent scheduler.

Leader Election in a Complete Graph



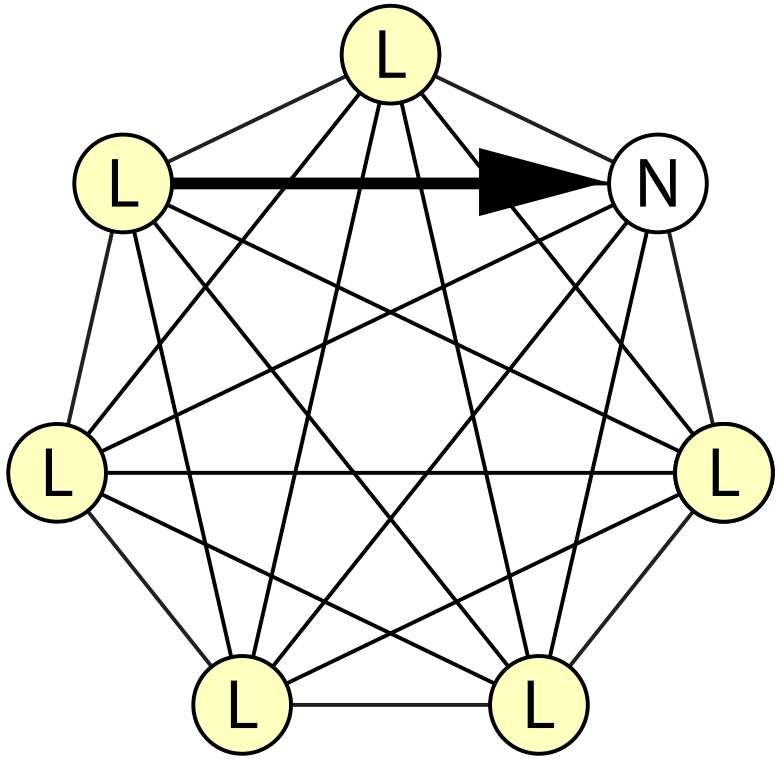
Initially, all agents have the leader state.

Leader Election in a Complete Graph



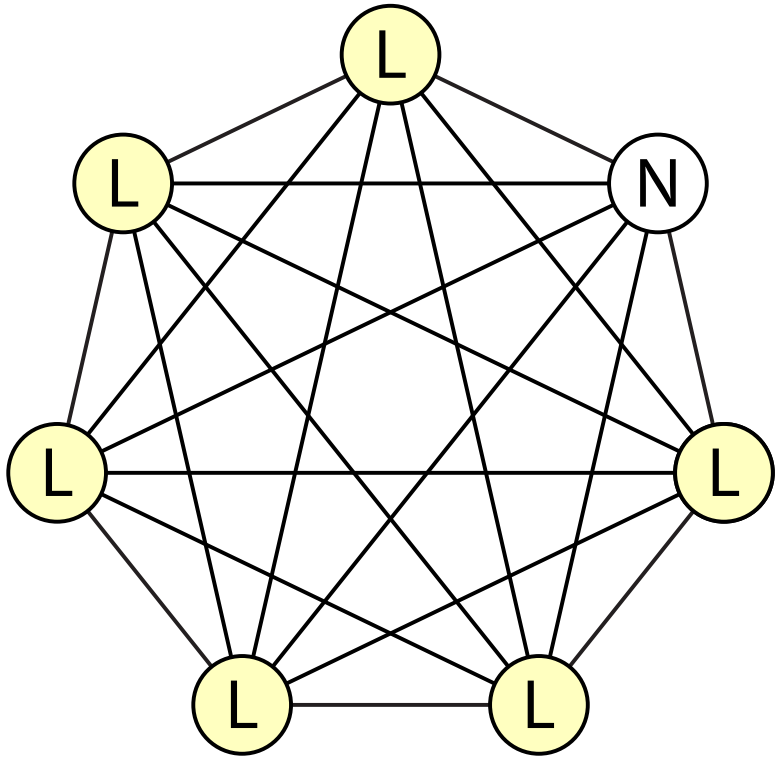
When two leaders interact, one is “eliminated”.

Leader Election in a Complete Graph



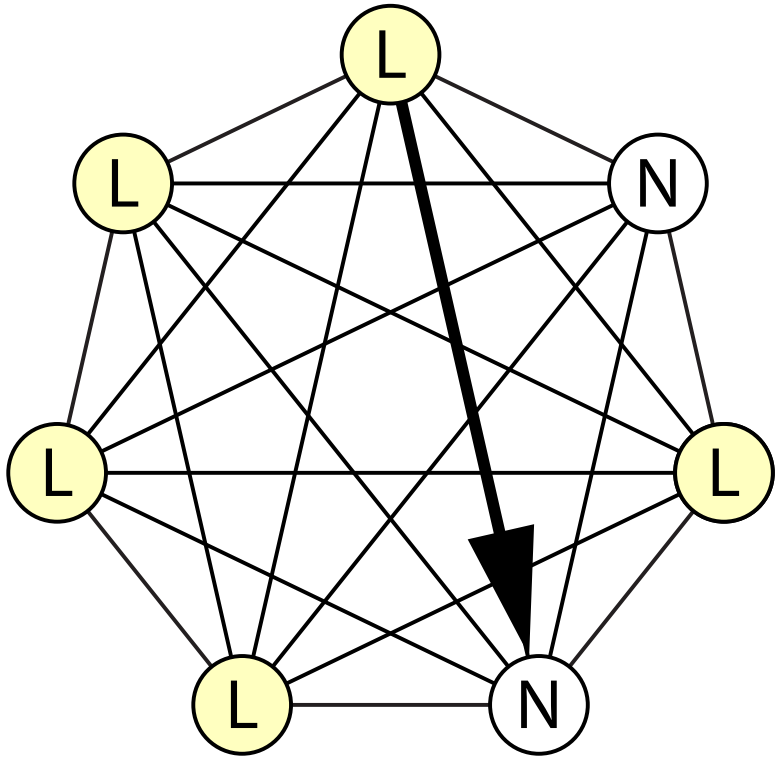
When two leaders interact, one is “eliminated”.

Leader Election in a Complete Graph



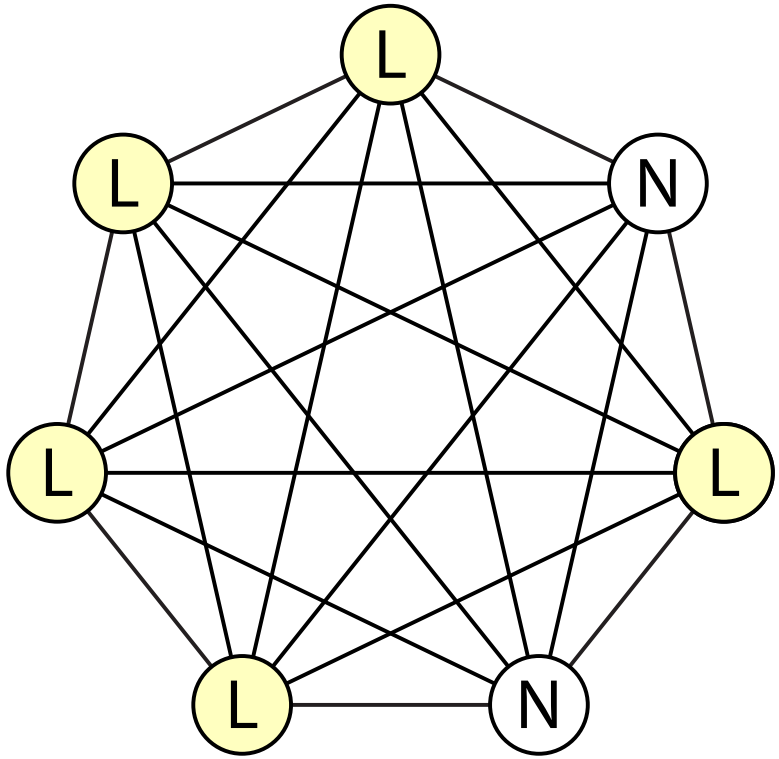
When two leaders interact, one is “eliminated”.

Leader Election in a Complete Graph



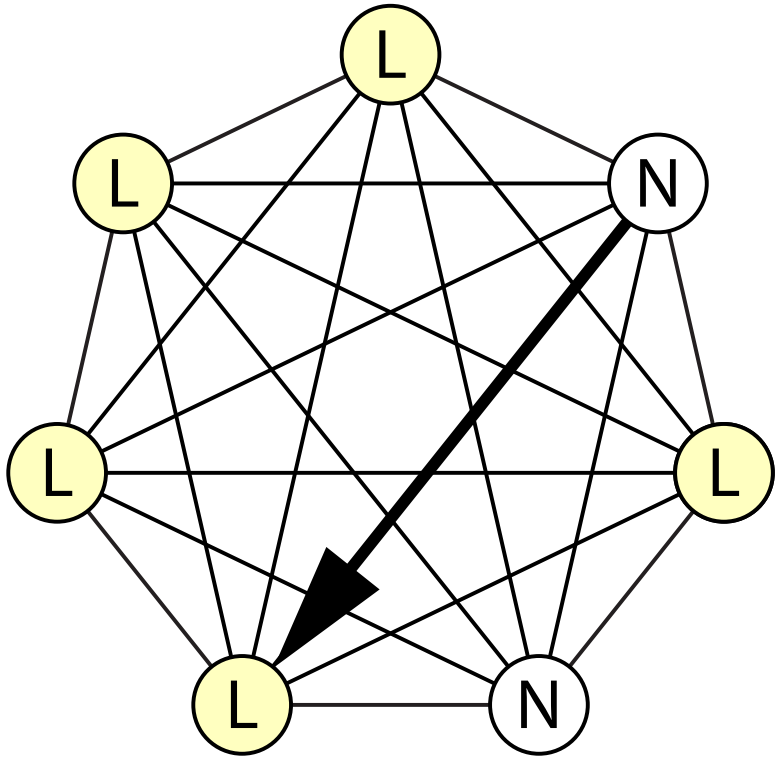
When two leaders interact, one is “eliminated”.

Leader Election in a Complete Graph



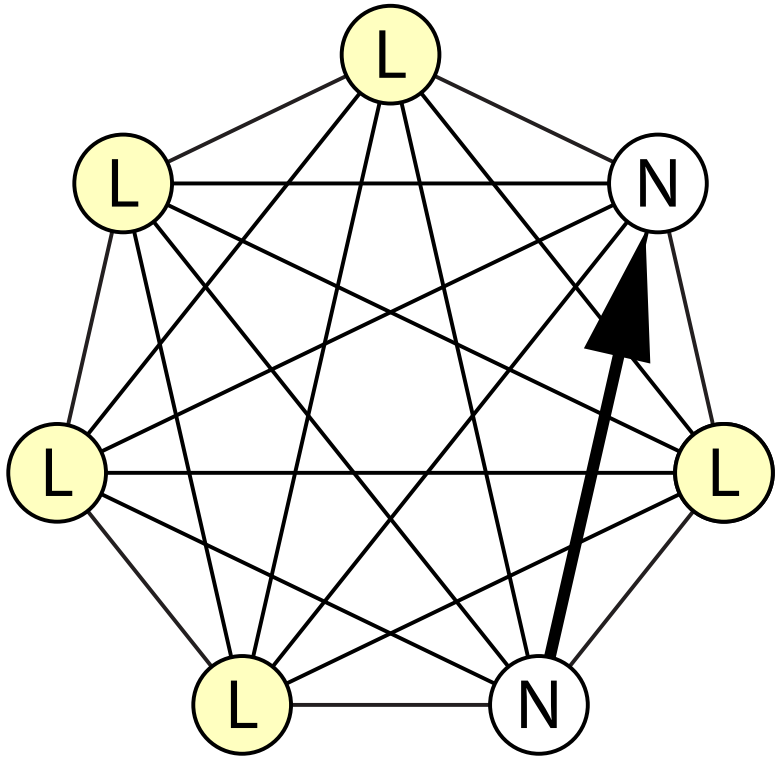
When two leaders interact, one is “eliminated”.

Leader Election in a Complete Graph



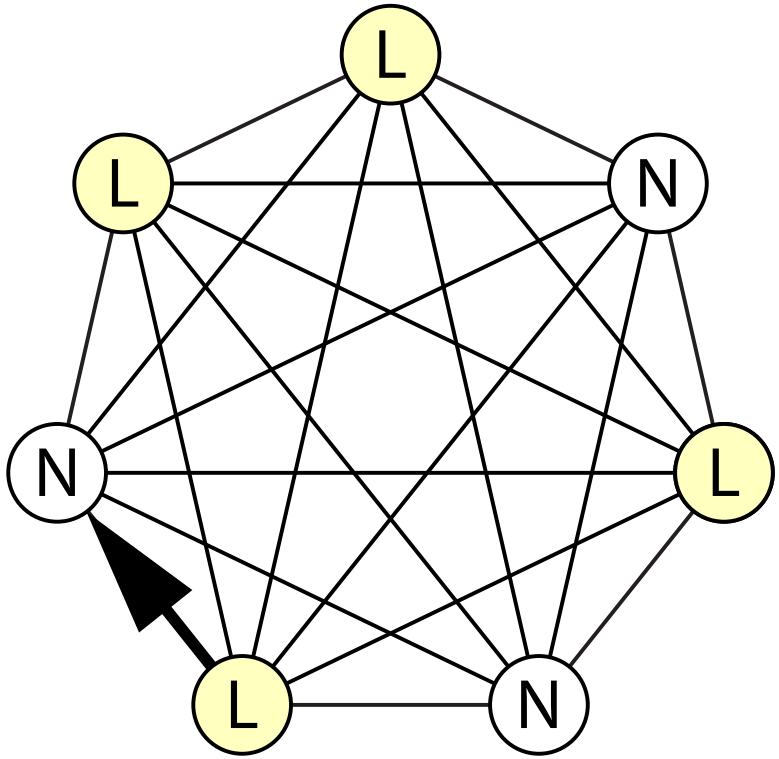
Otherwise, nothing happens.

Leader Election in a Complete Graph



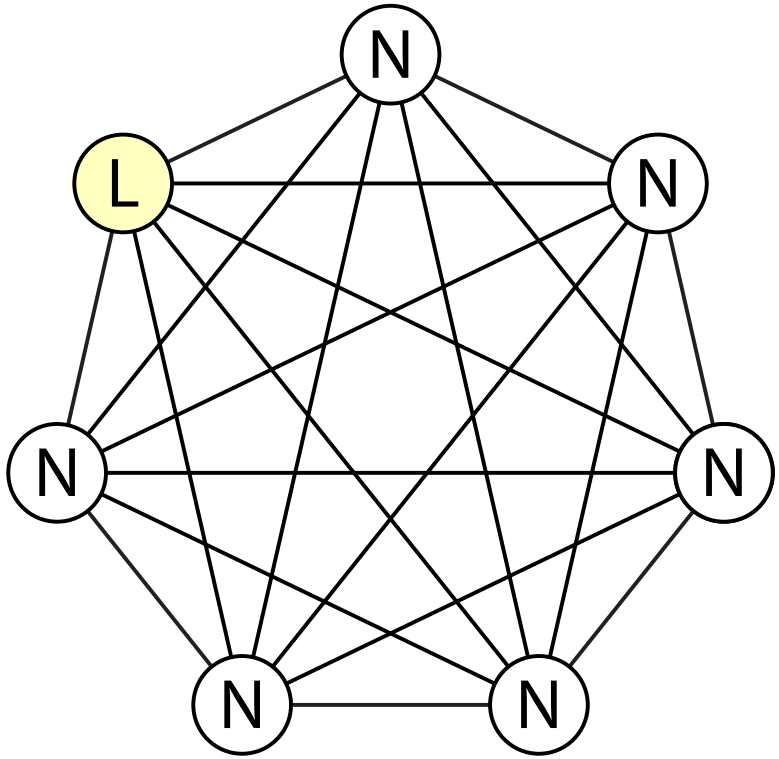
Otherwise, nothing happens.

Leader Election in a Complete Graph



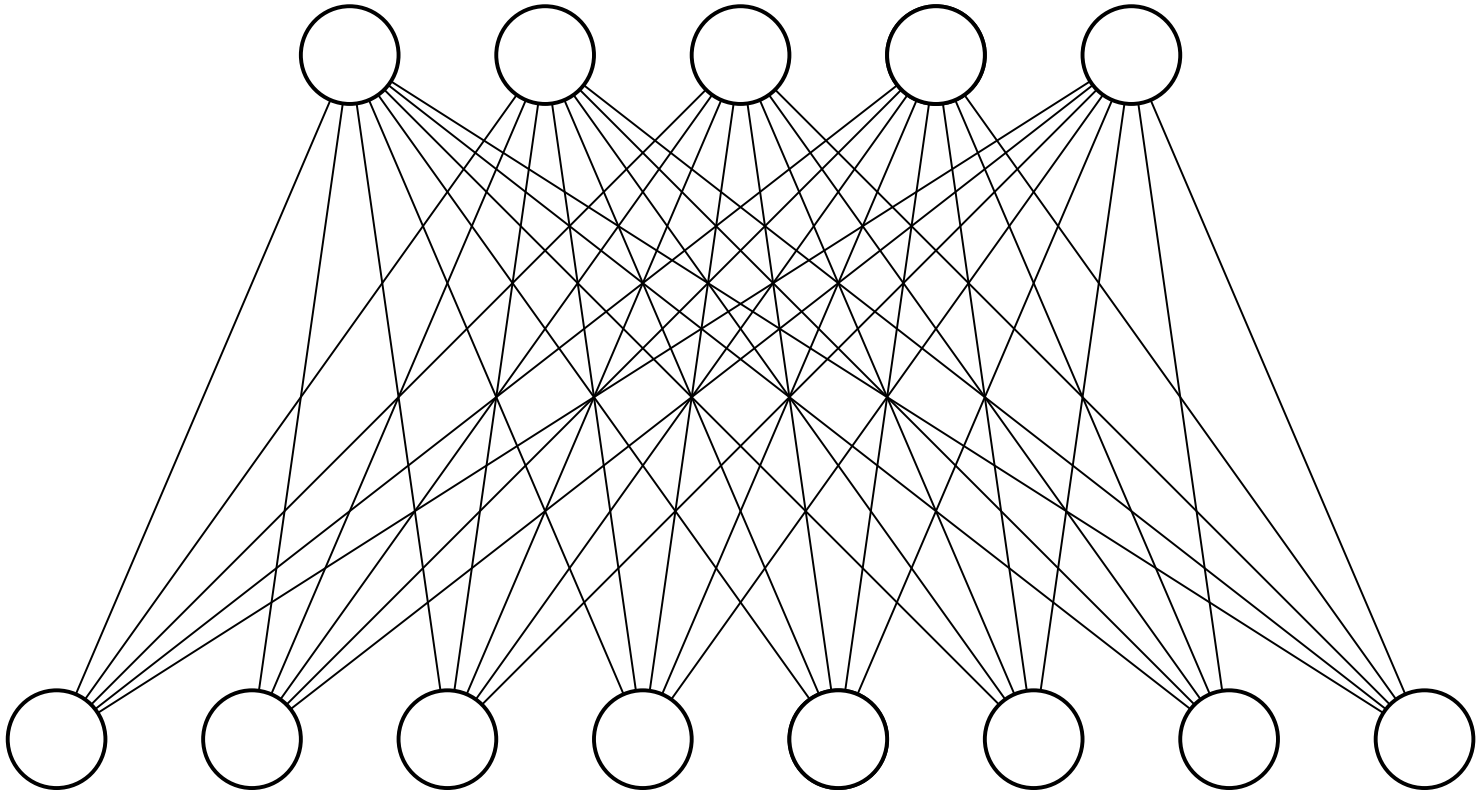
Otherwise, nothing happens.

Leader Election in a Complete Graph



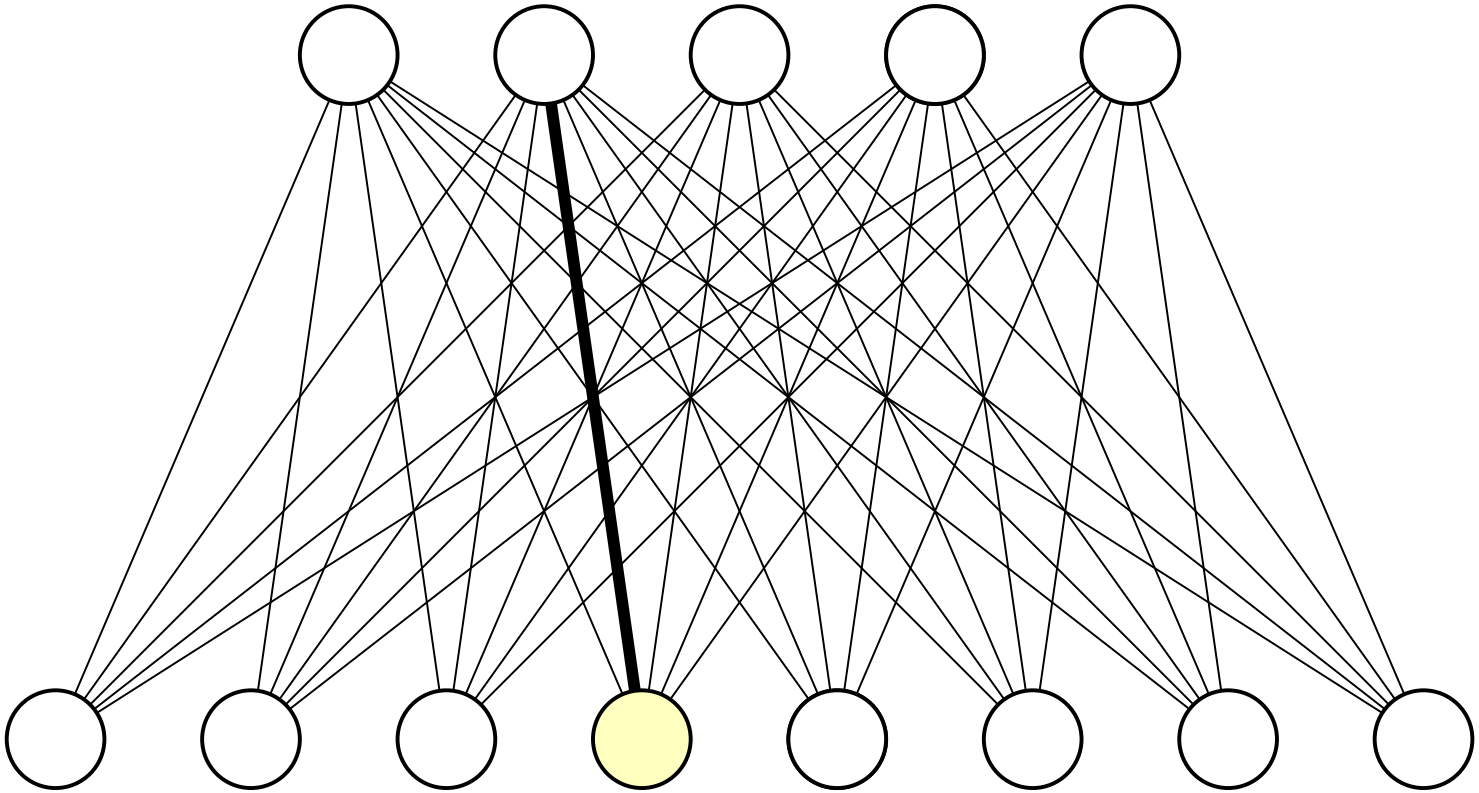
Eventually, only one leader remains.

Leader Election in a Complete Bipartite Graph



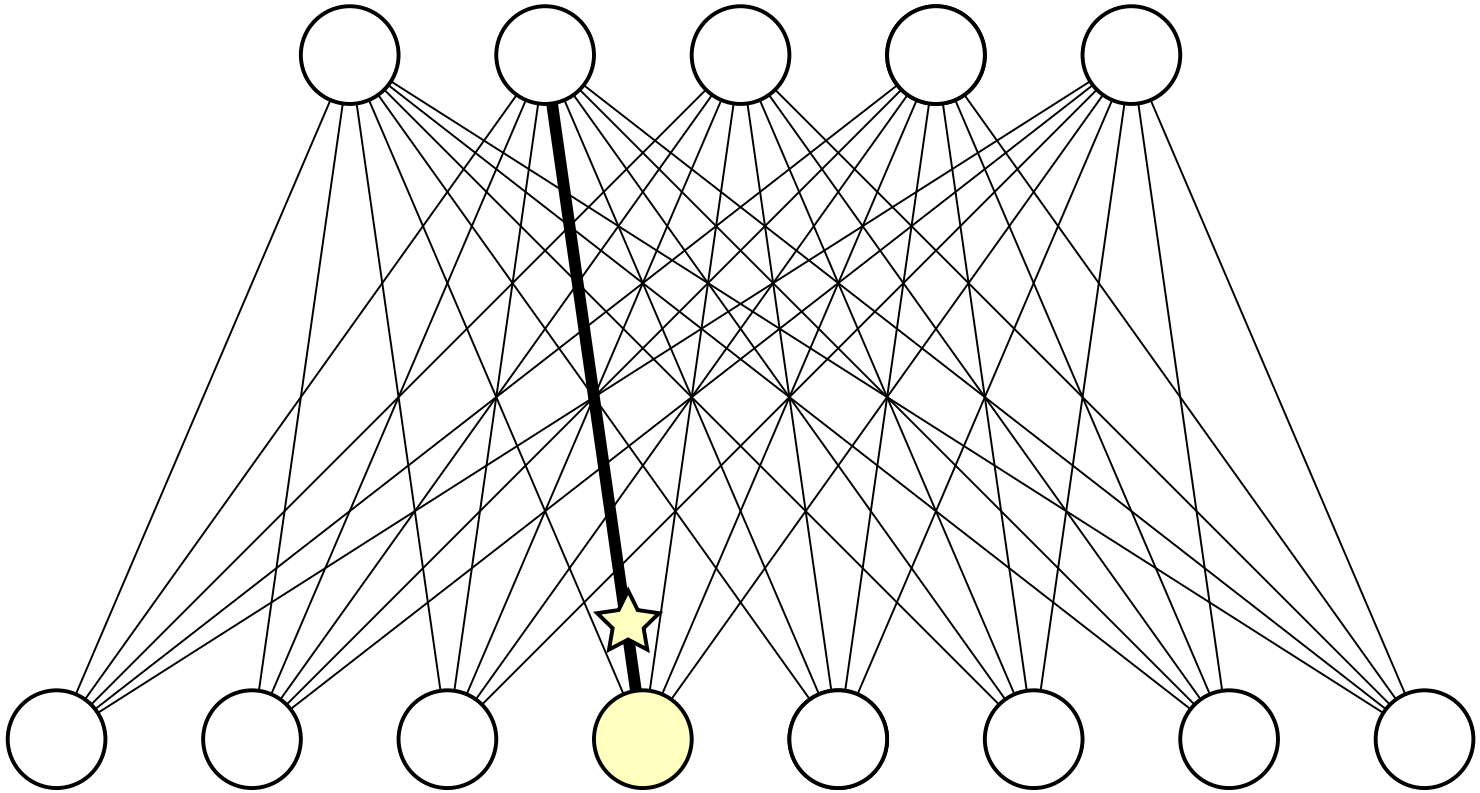
Theorem: in $K_{m,n}$, it is possible to elect a leader under the 1-bounded scheduler if and only if m and n are coprime.

Leader Election in a Complete Bipartite Graph



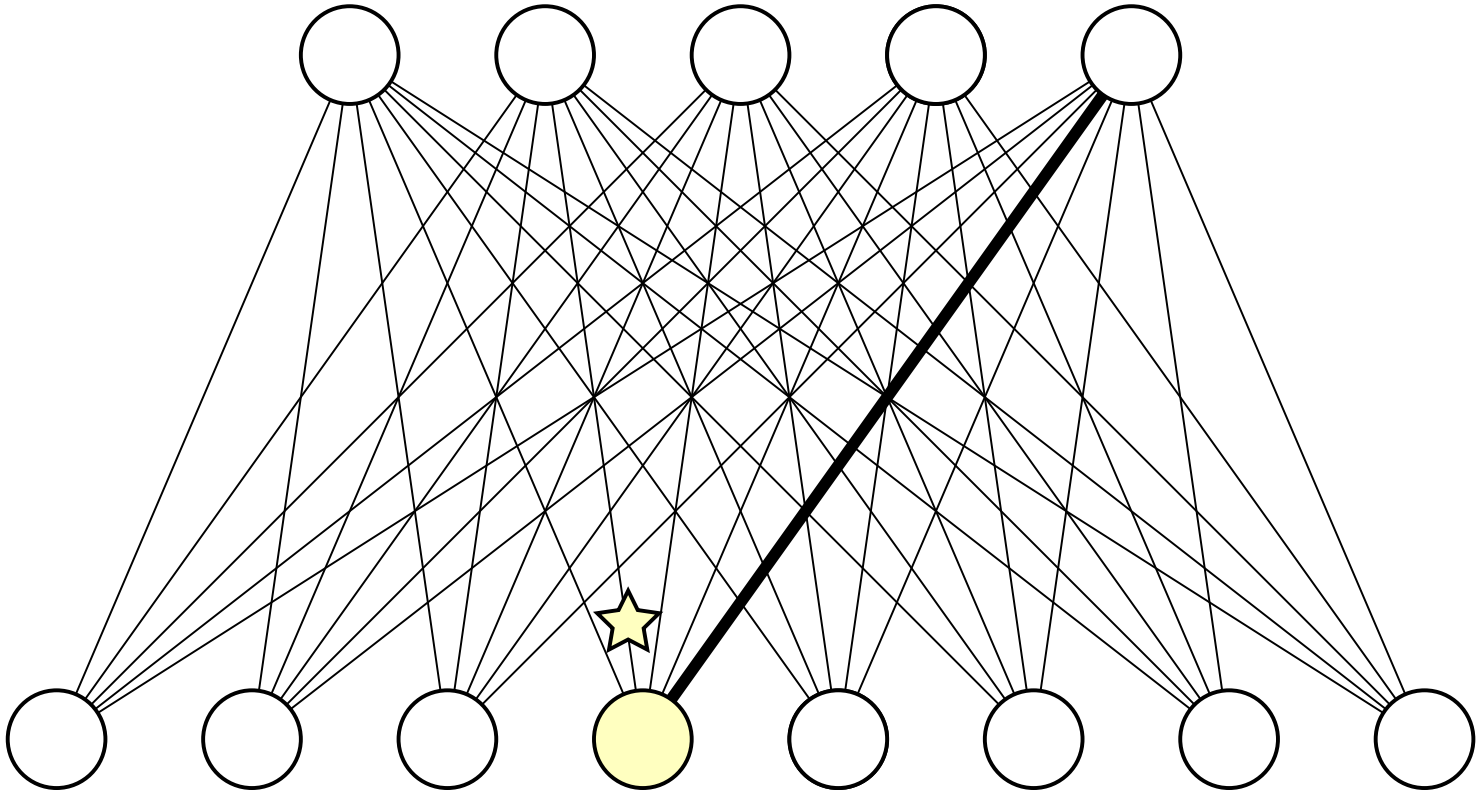
Suppose that m and n are coprime.

Leader Election in a Complete Bipartite Graph



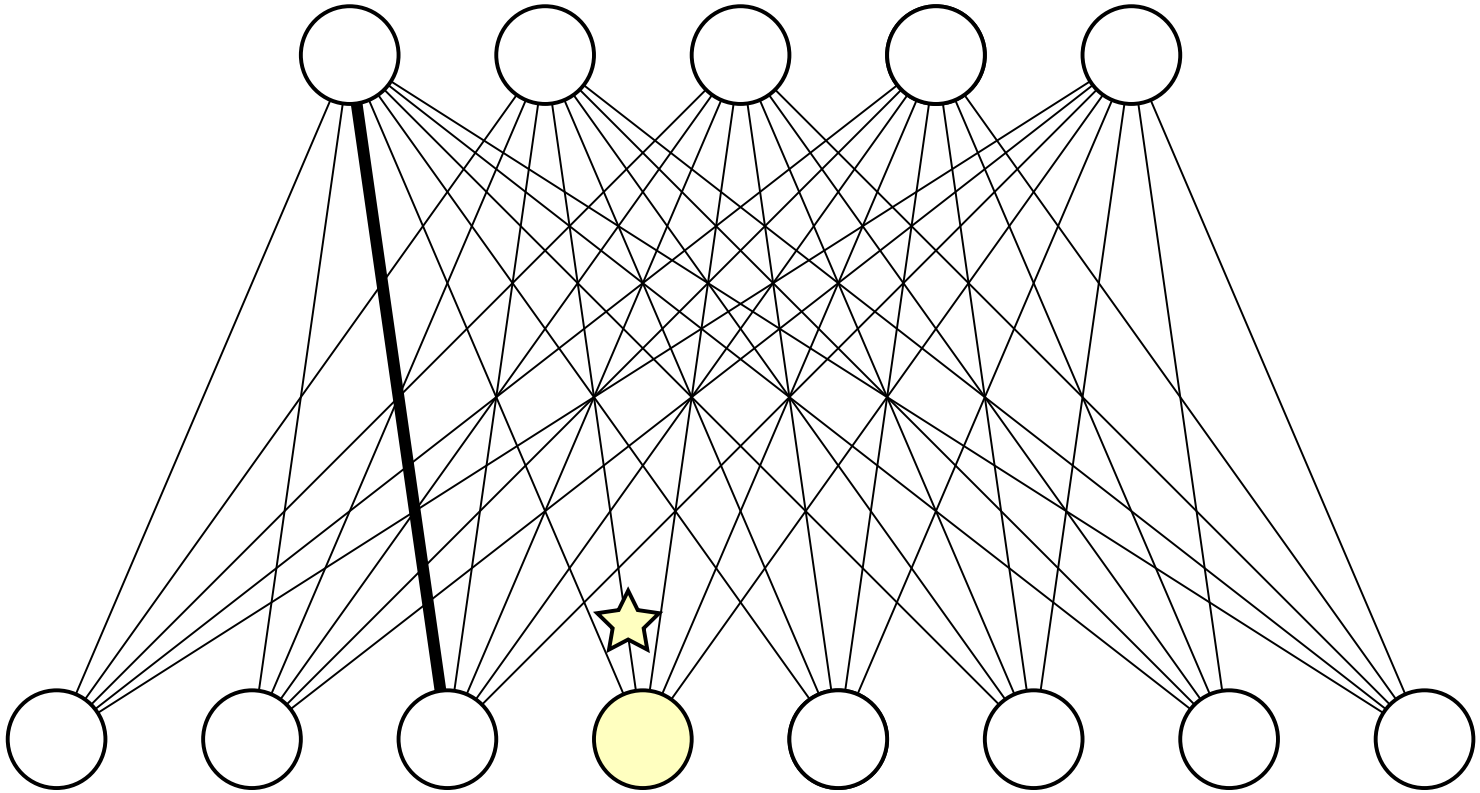
Since the 1-bounded scheduler is periodic, an agent can tell when a new period starts by marking the first edge that it "sees".

Leader Election in a Complete Bipartite Graph



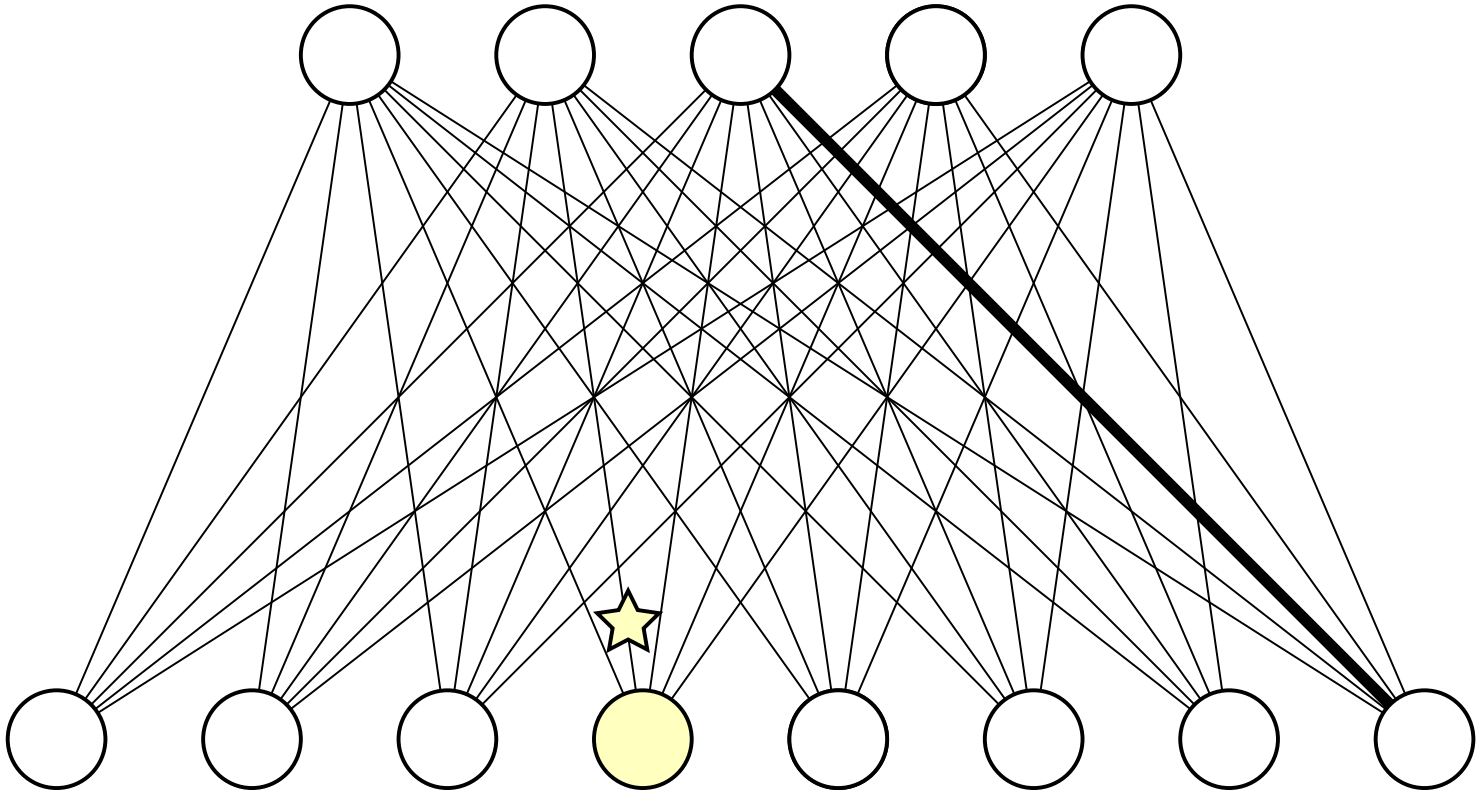
Since the 1-bounded scheduler is periodic, an agent can tell when a new period starts by marking the first edge that it "sees".

Leader Election in a Complete Bipartite Graph



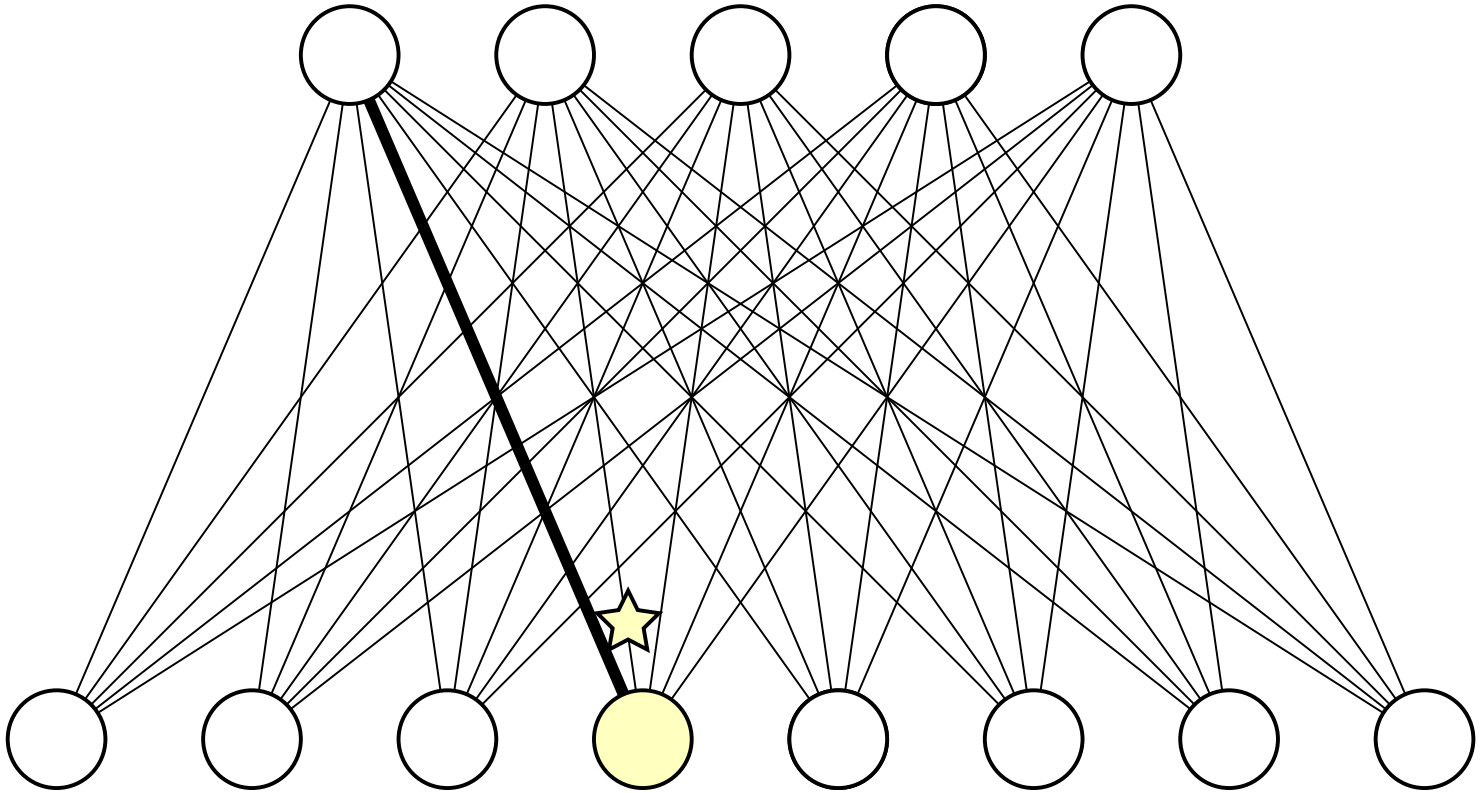
Since the 1-bounded scheduler is periodic, an agent can tell when a new period starts by marking the first edge that it "sees".

Leader Election in a Complete Bipartite Graph



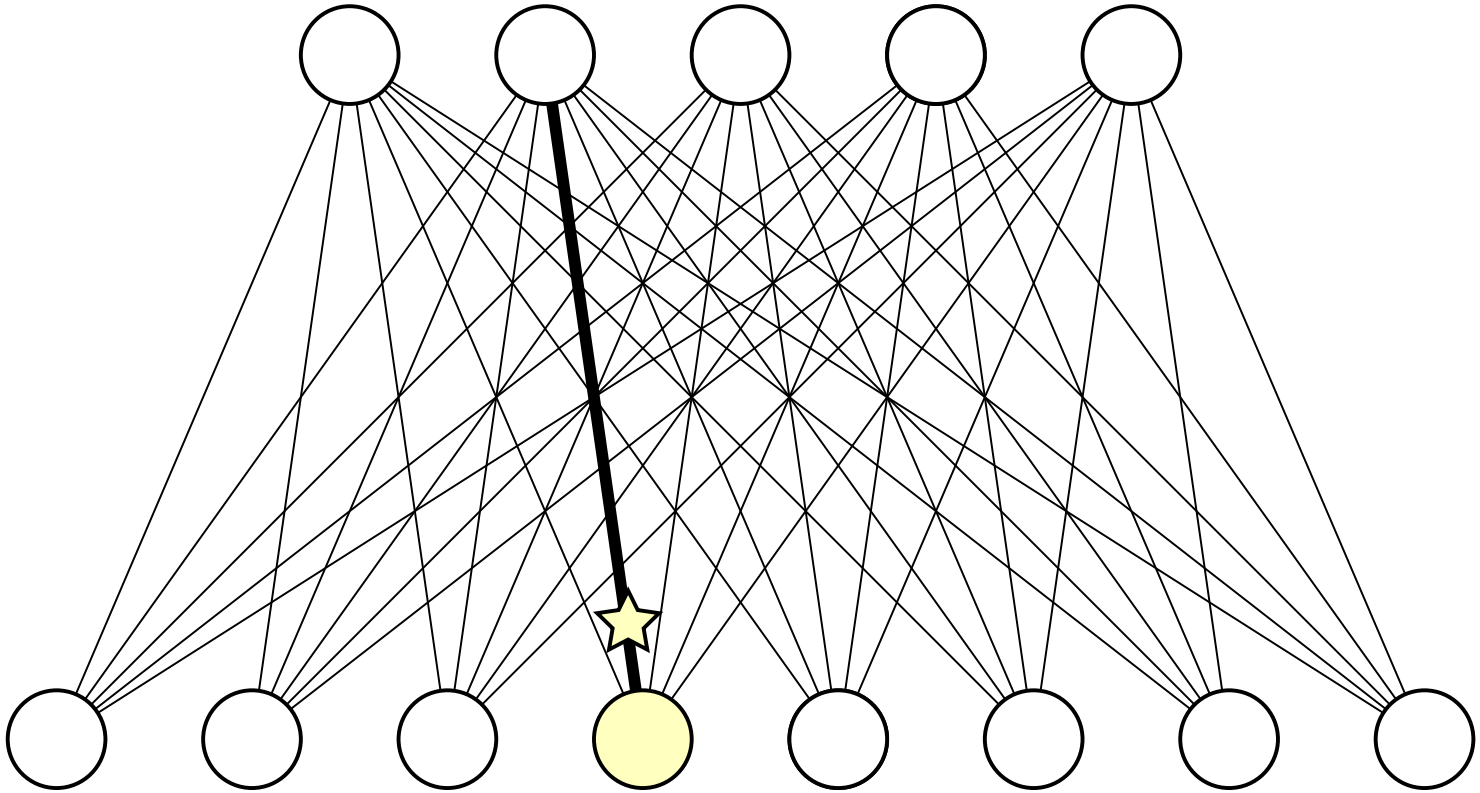
Since the 1-bounded scheduler is periodic, an agent can tell when a new period starts by marking the first edge that it "sees".

Leader Election in a Complete Bipartite Graph



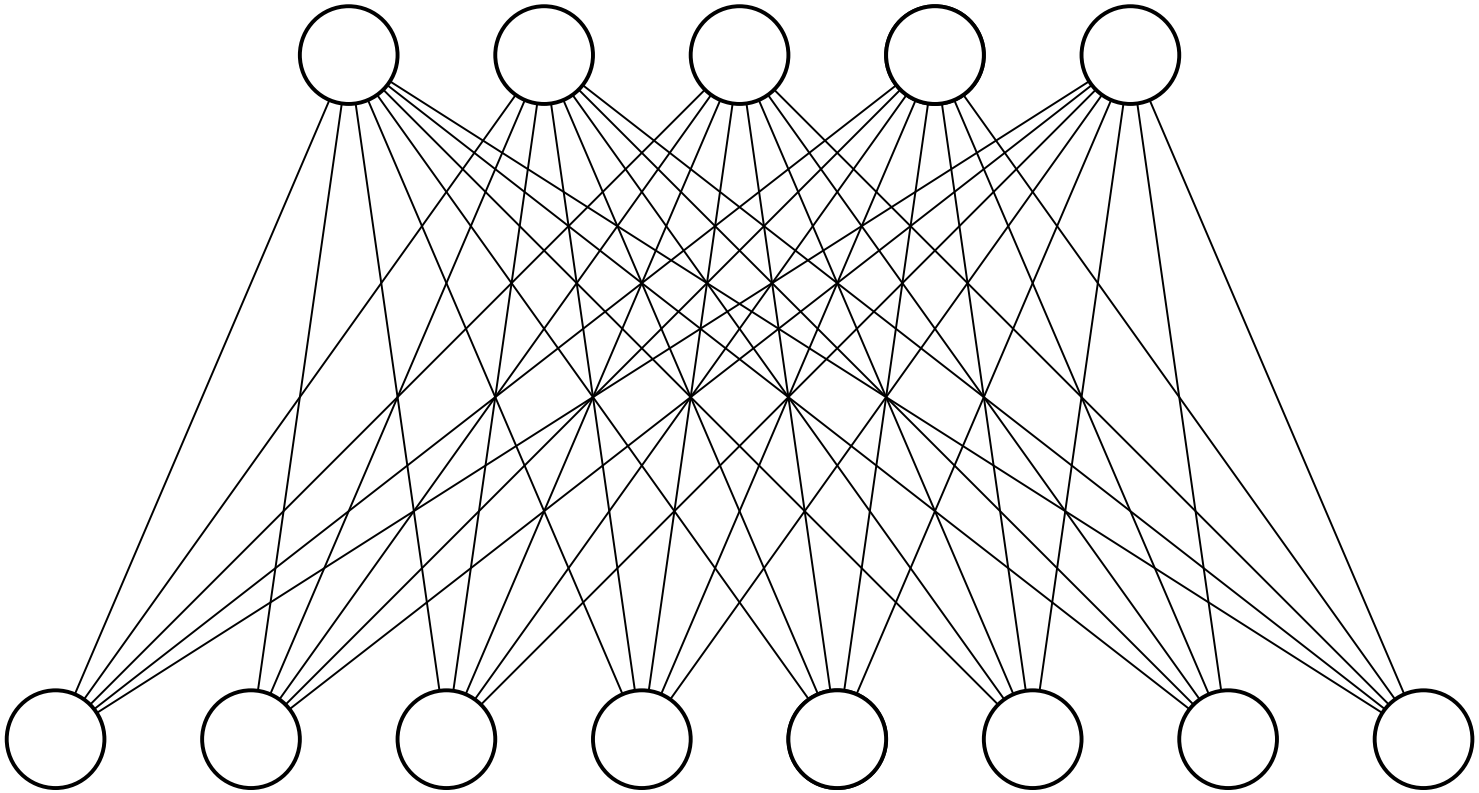
Since the 1-bounded scheduler is periodic, an agent can tell when a new period starts by marking the first edge that it "sees".

Leader Election in a Complete Bipartite Graph



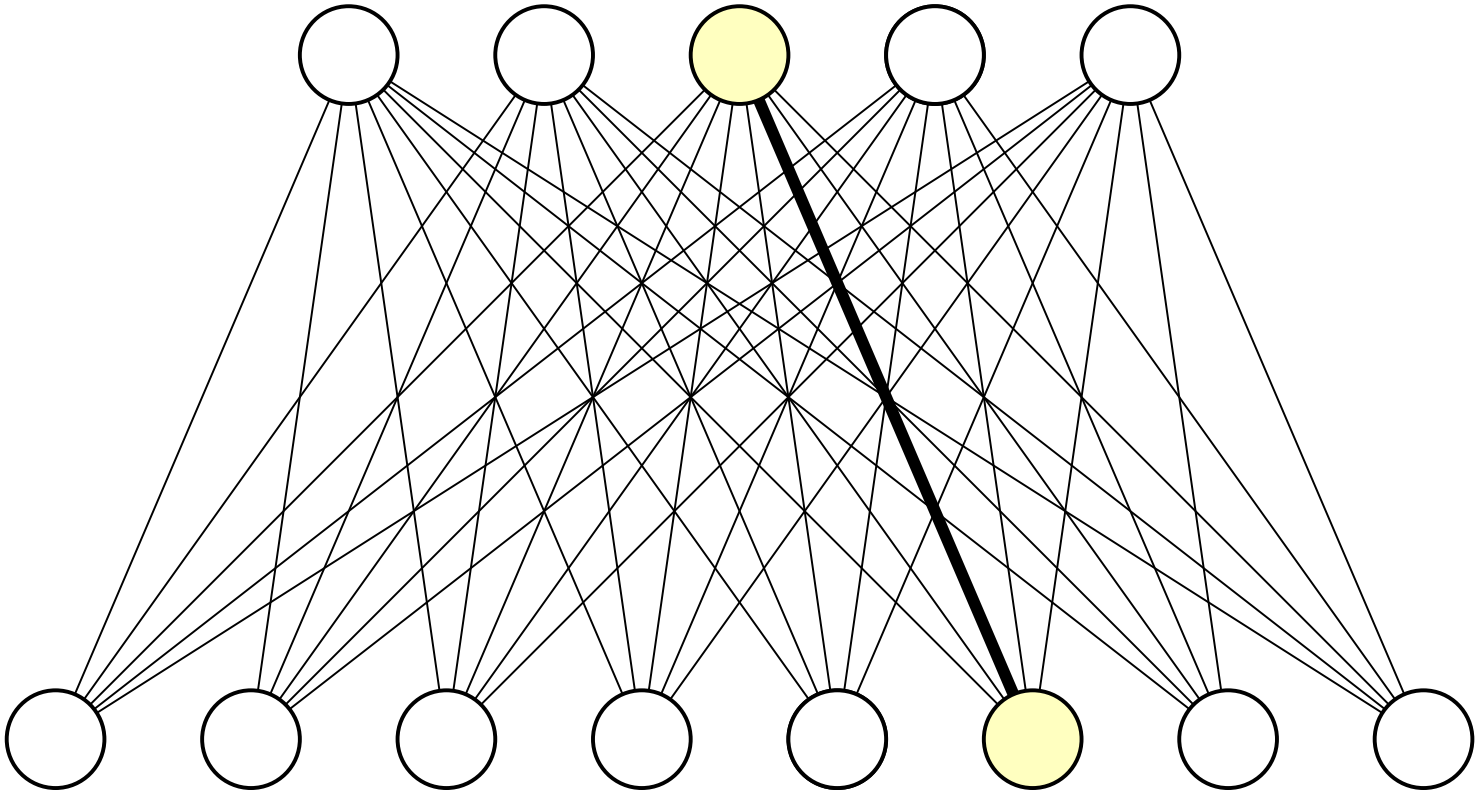
The next time it encounters the marked edge, it knows that a new period has started.

Leader Election in a Complete Bipartite Graph



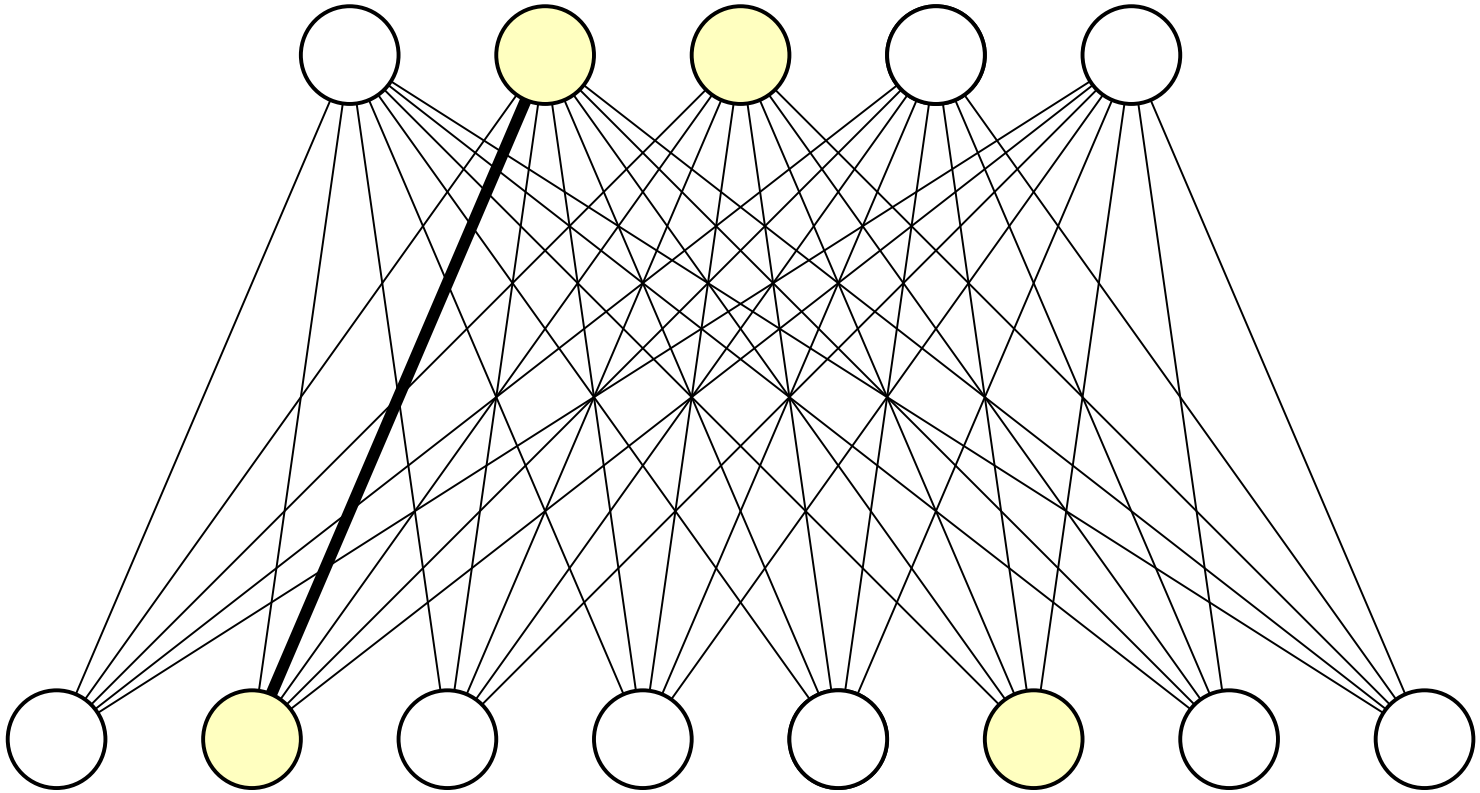
In the first phase, we construct a maximal matching.

Leader Election in a Complete Bipartite Graph



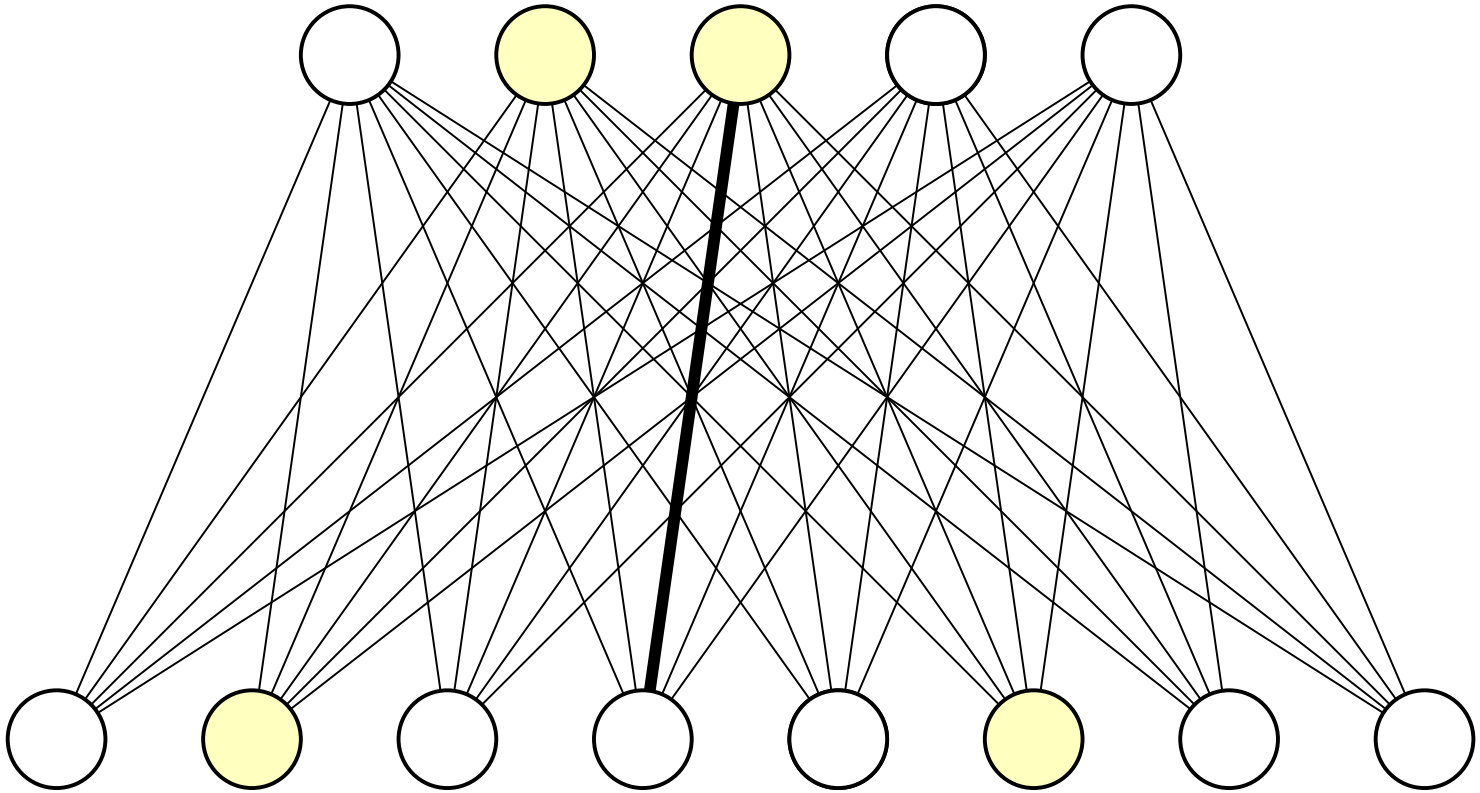
When two unmatched agents meet, they become matched.

Leader Election in a Complete Bipartite Graph



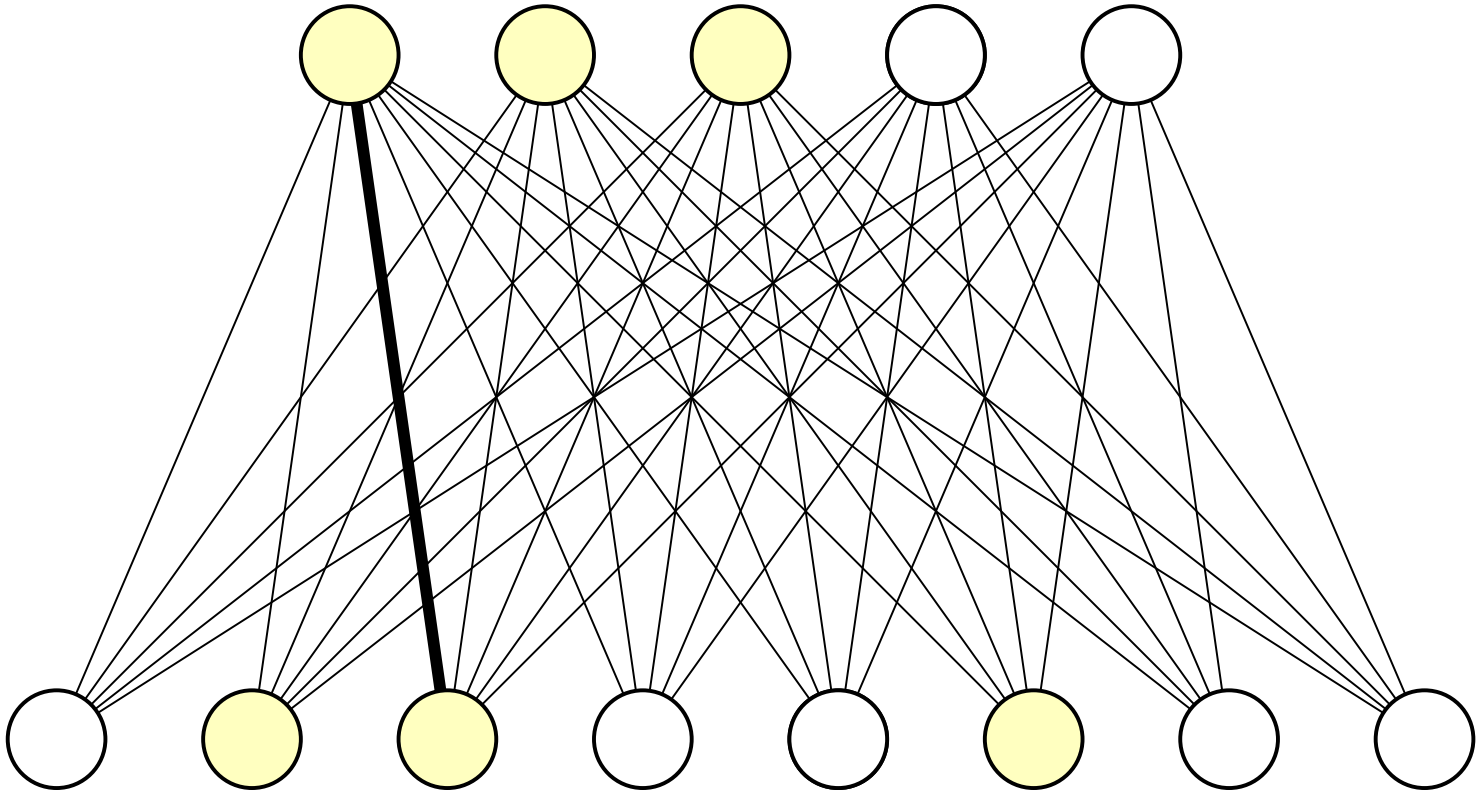
When two unmatched agents meet, they become matched.

Leader Election in a Complete Bipartite Graph



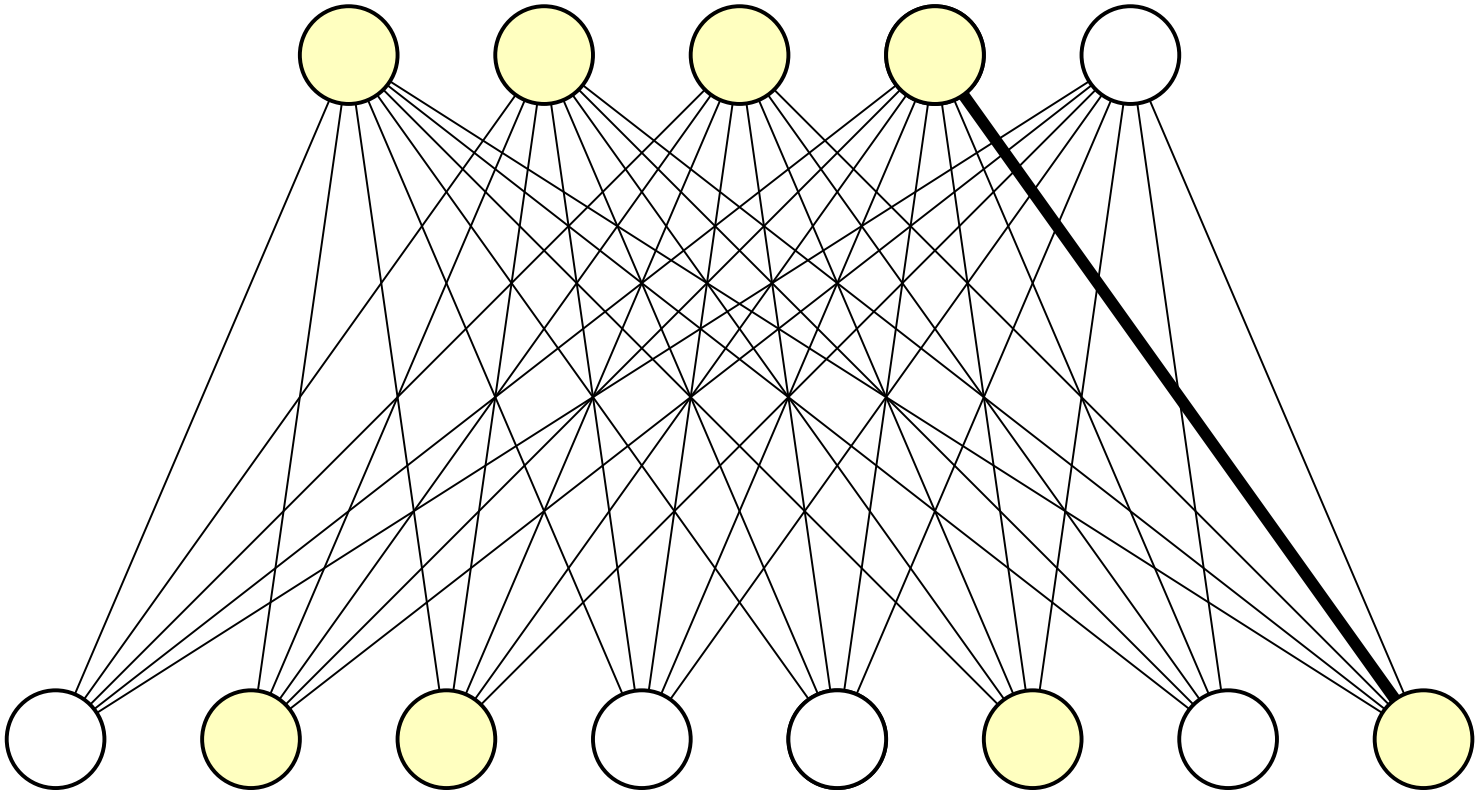
When two unmatched agents meet, they become matched.

Leader Election in a Complete Bipartite Graph



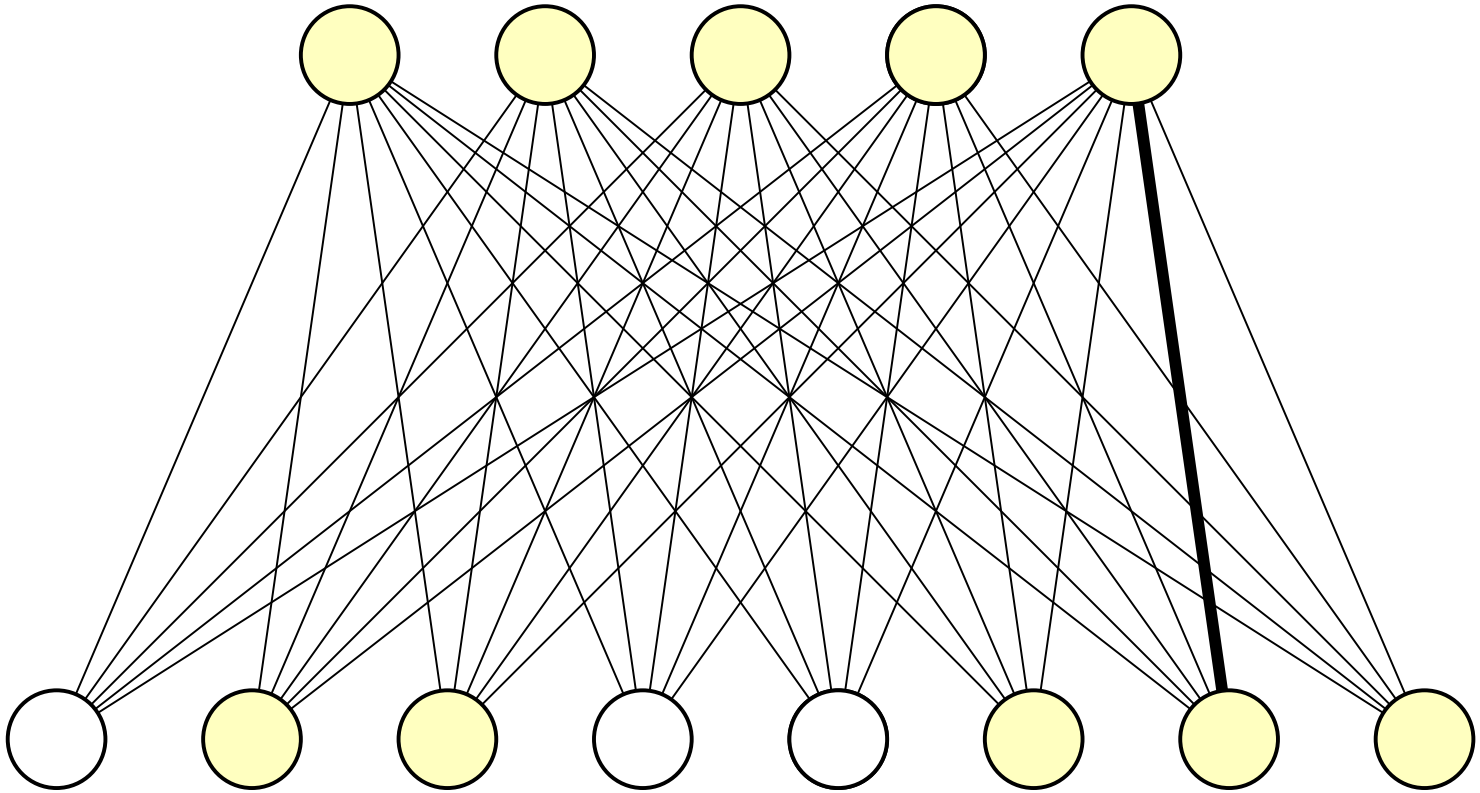
When two unmatched agents meet, they become matched.

Leader Election in a Complete Bipartite Graph



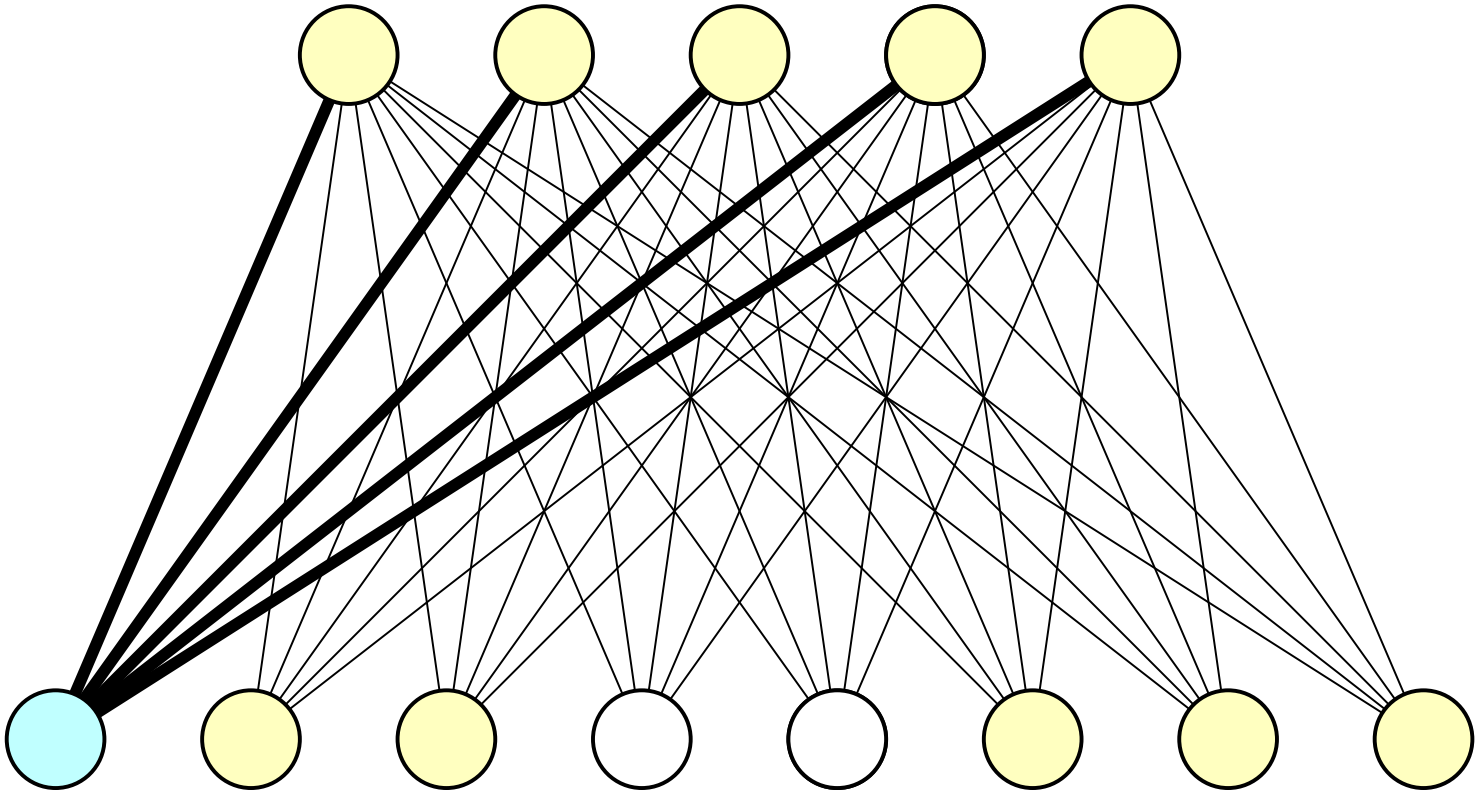
When two unmatched agents meet, they become matched.

Leader Election in a Complete Bipartite Graph



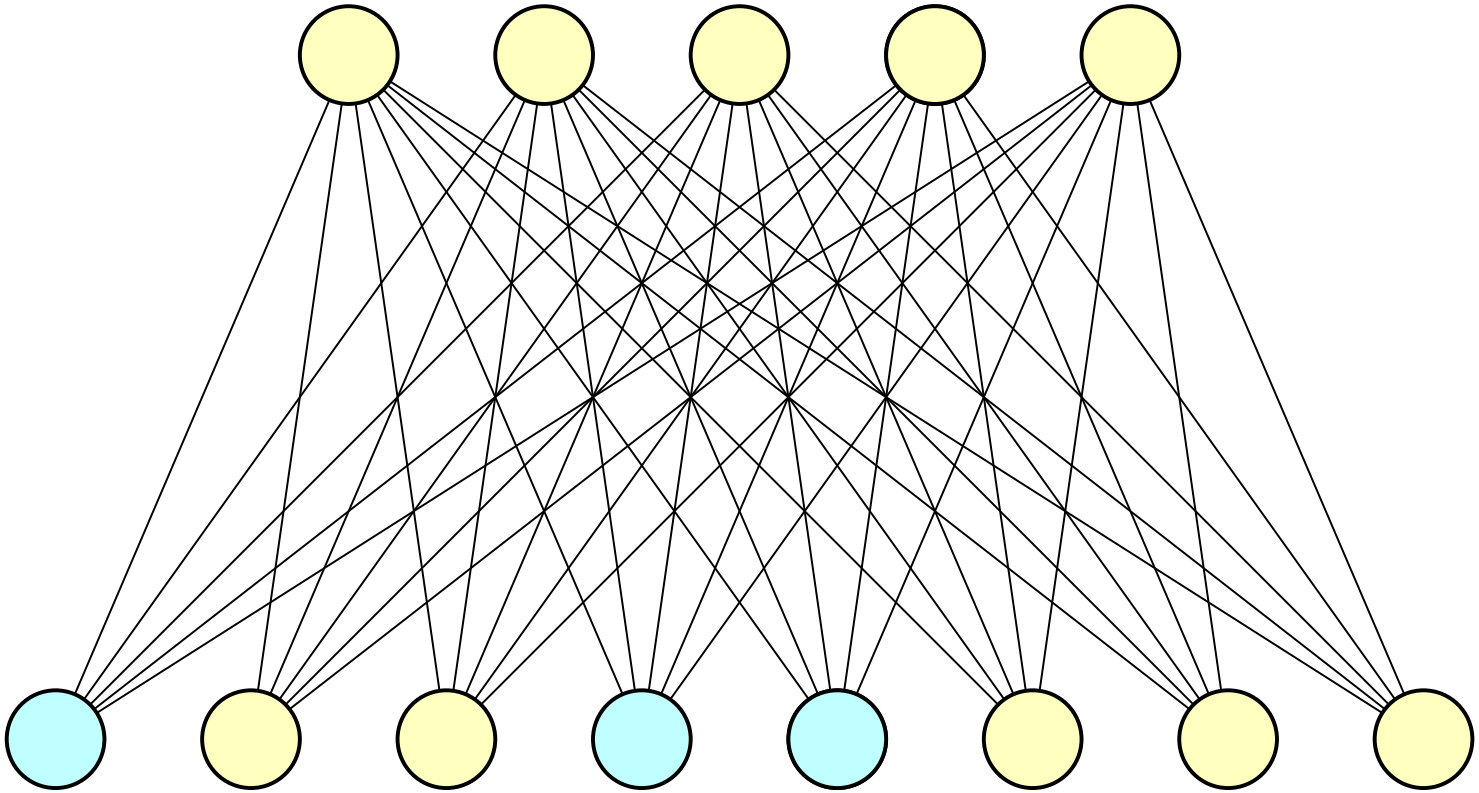
When two unmatched agents meet, they become matched.

Leader Election in a Complete Bipartite Graph



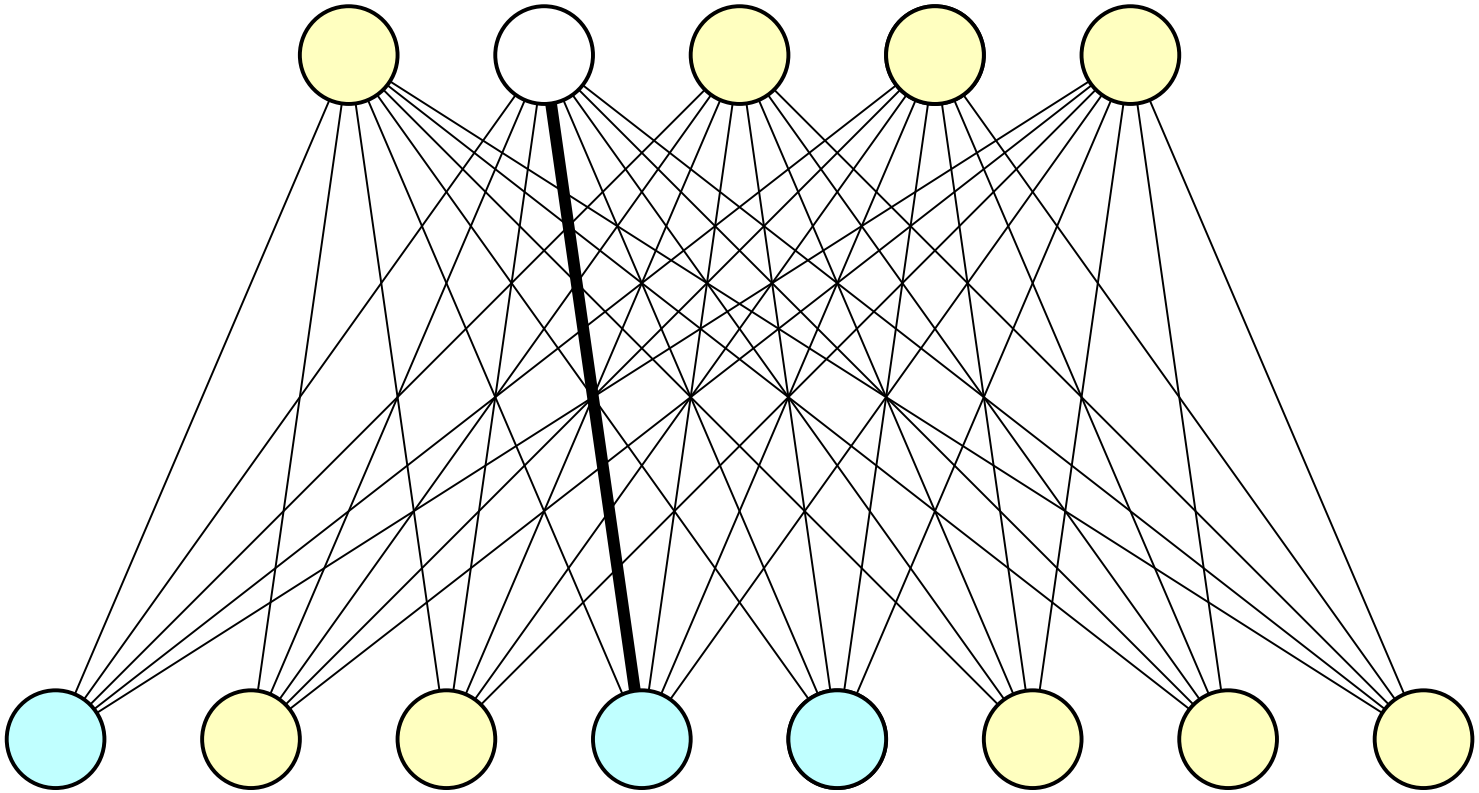
When an unmatched agents sees only matched agents for an entire period, it knows that the matching is maximal.

Leader Election in a Complete Bipartite Graph



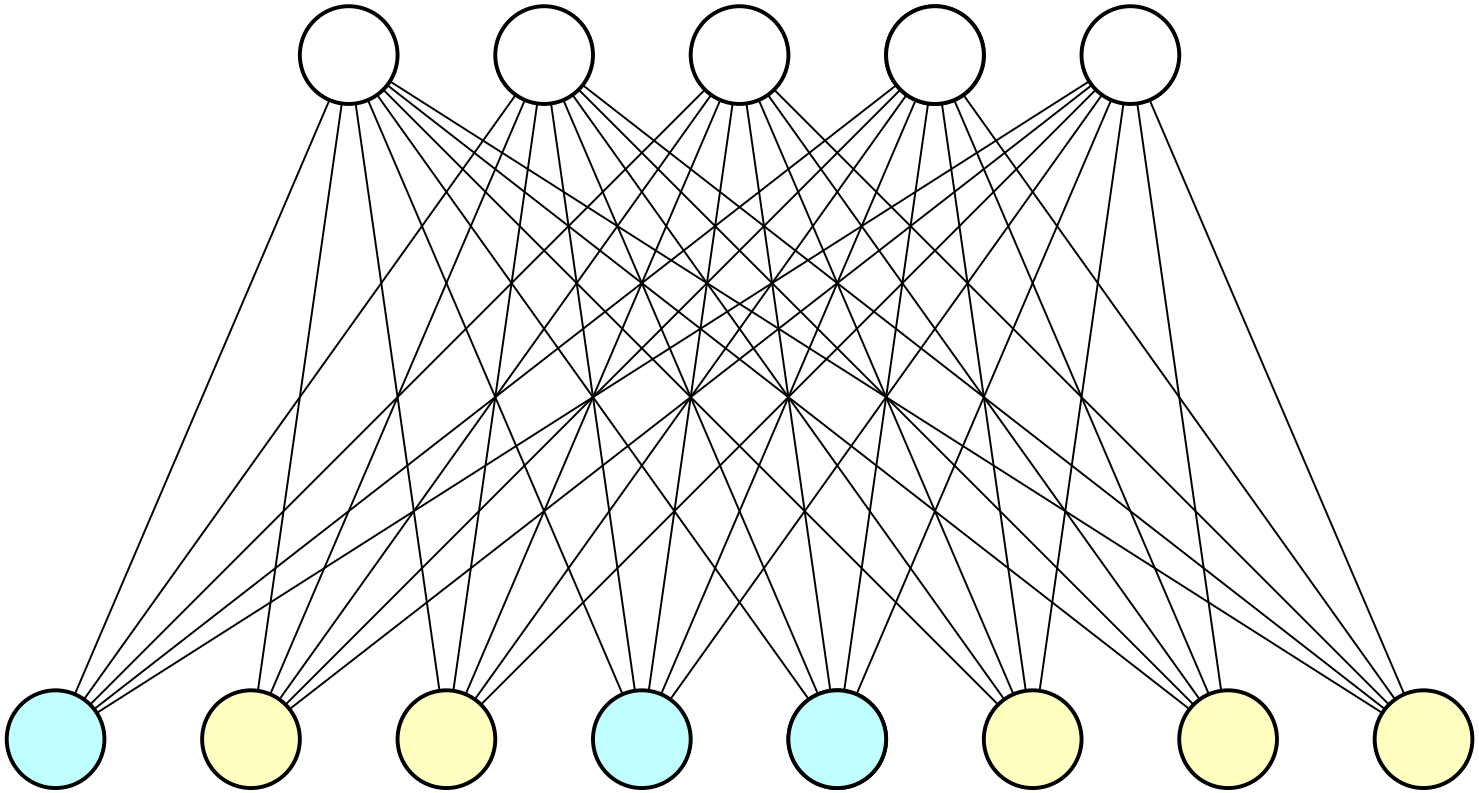
These unmatched agents assume a “reset” state.

Leader Election in a Complete Bipartite Graph



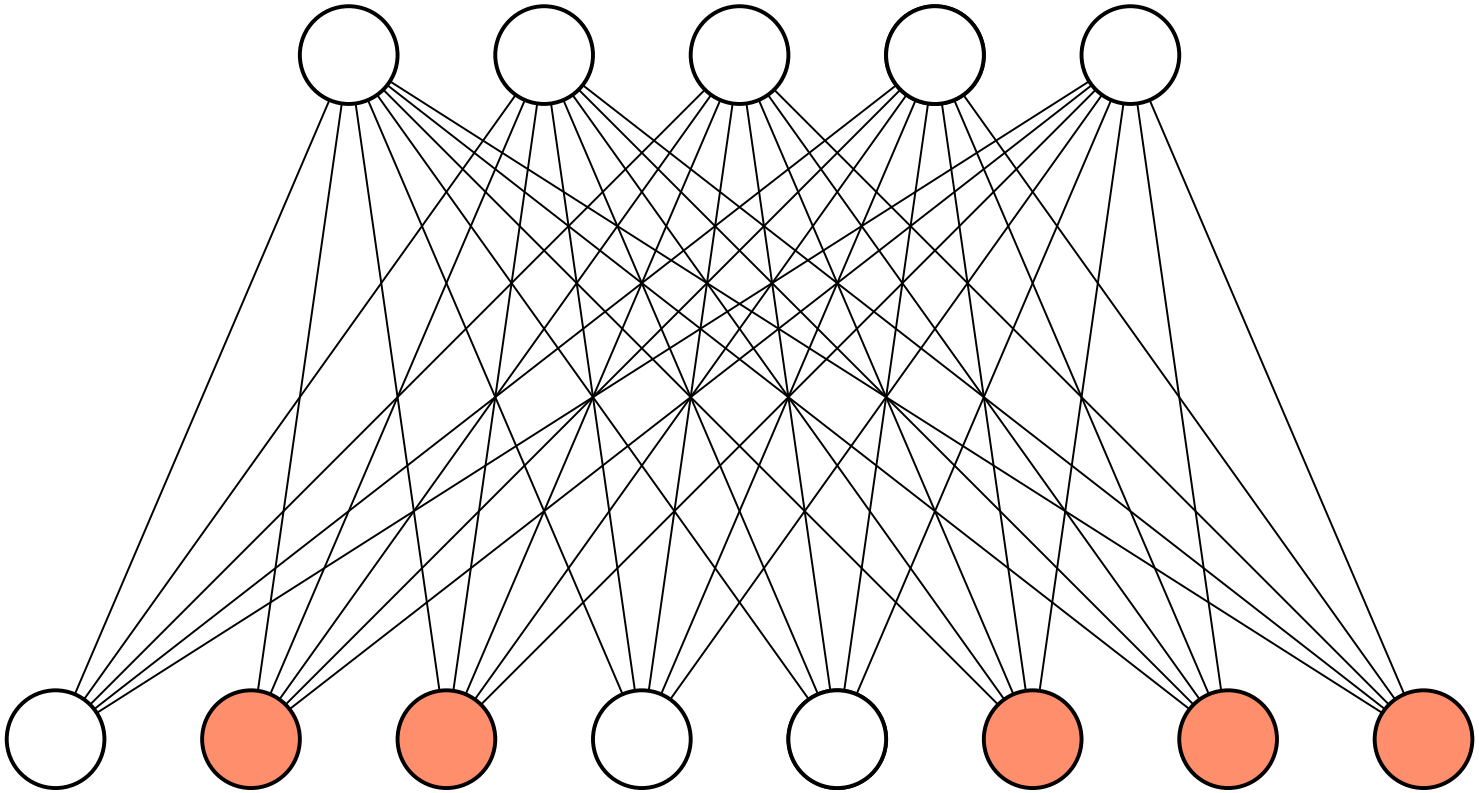
After another period, whoever sees a “reset” agent becomes unmatched again.

Leader Election in a Complete Bipartite Graph



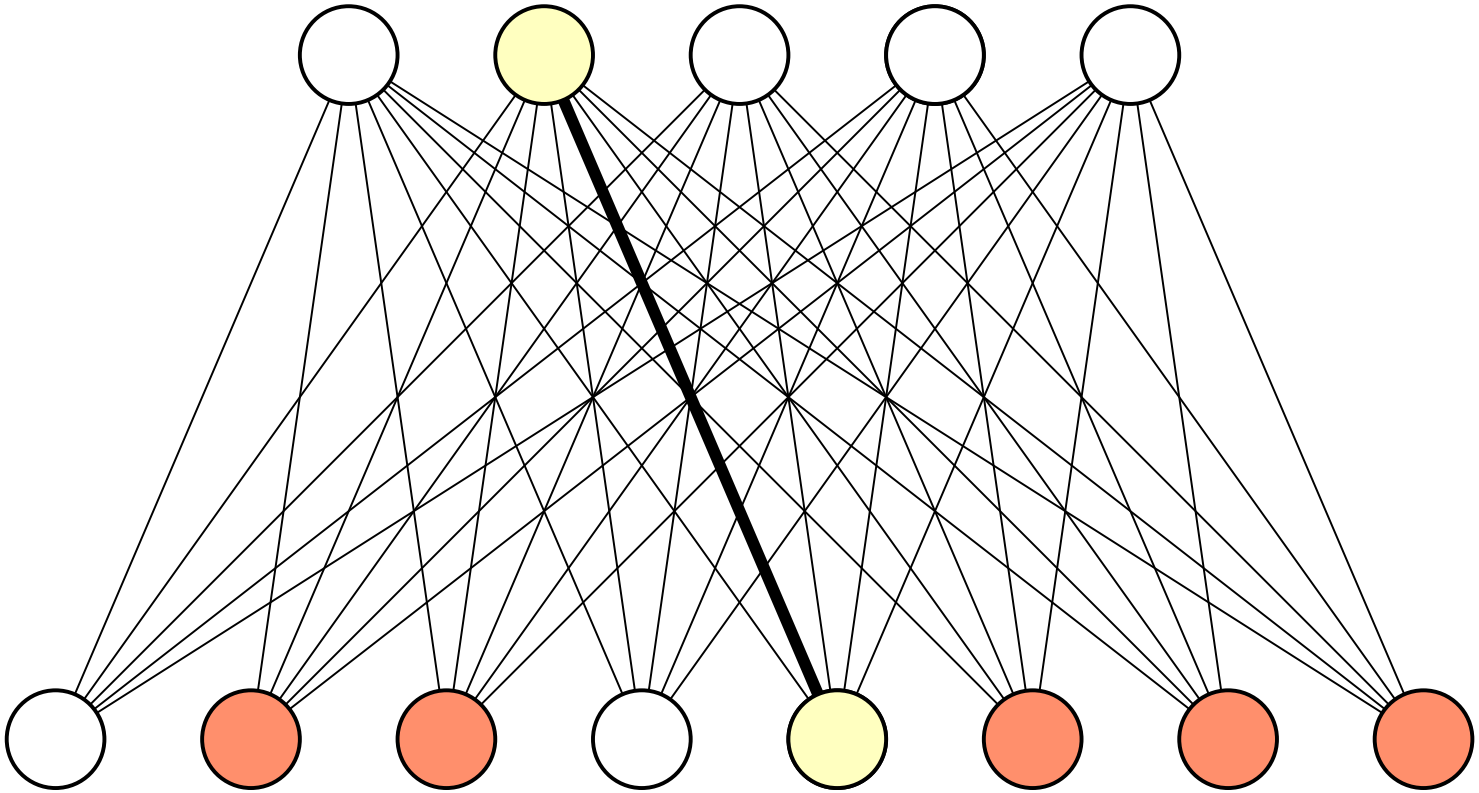
After another period, whoever sees a “reset” agent becomes unmatched again.

Leader Election in a Complete Bipartite Graph



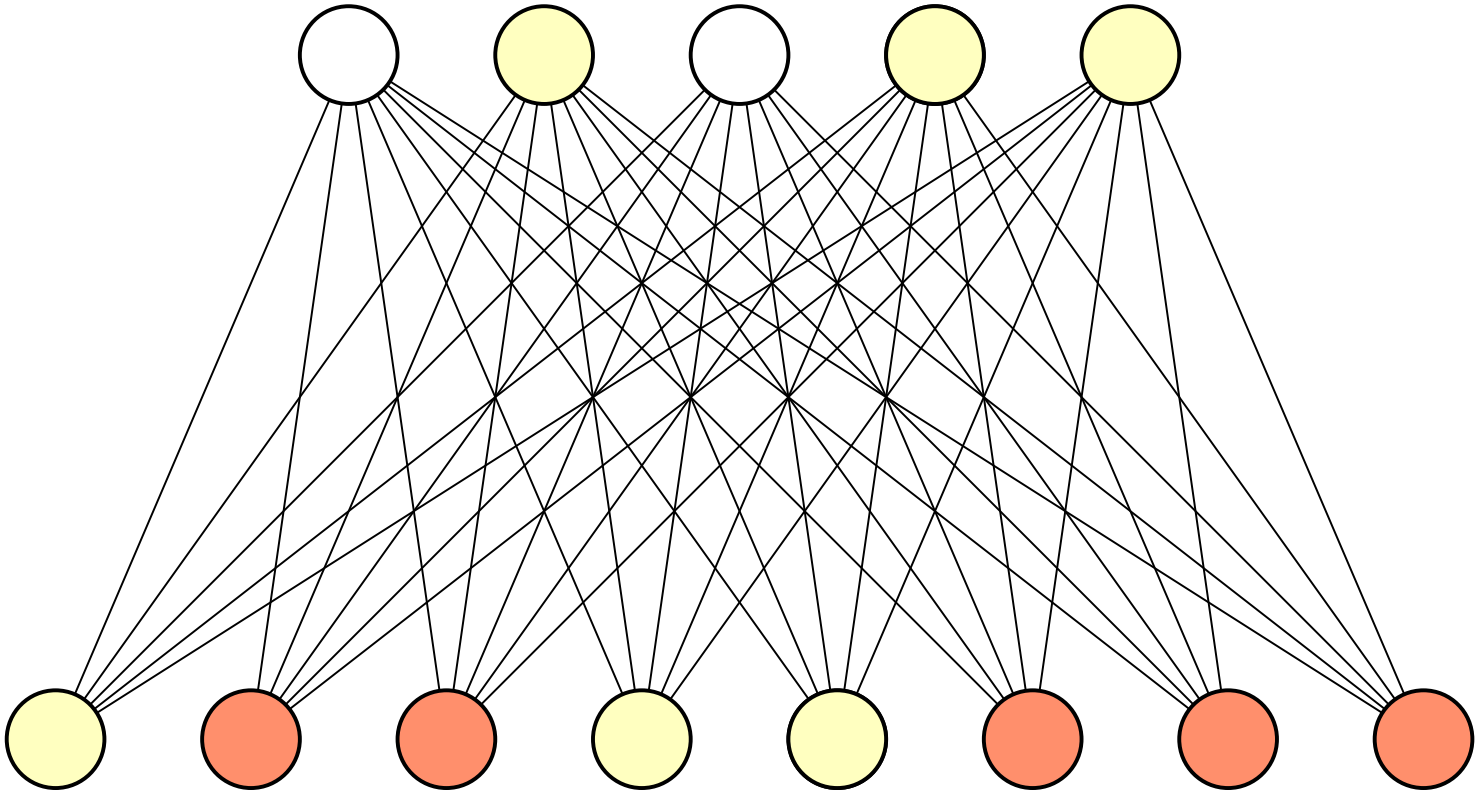
In the next period, the agents that are still matched become "eliminated".

Leader Election in a Complete Bipartite Graph



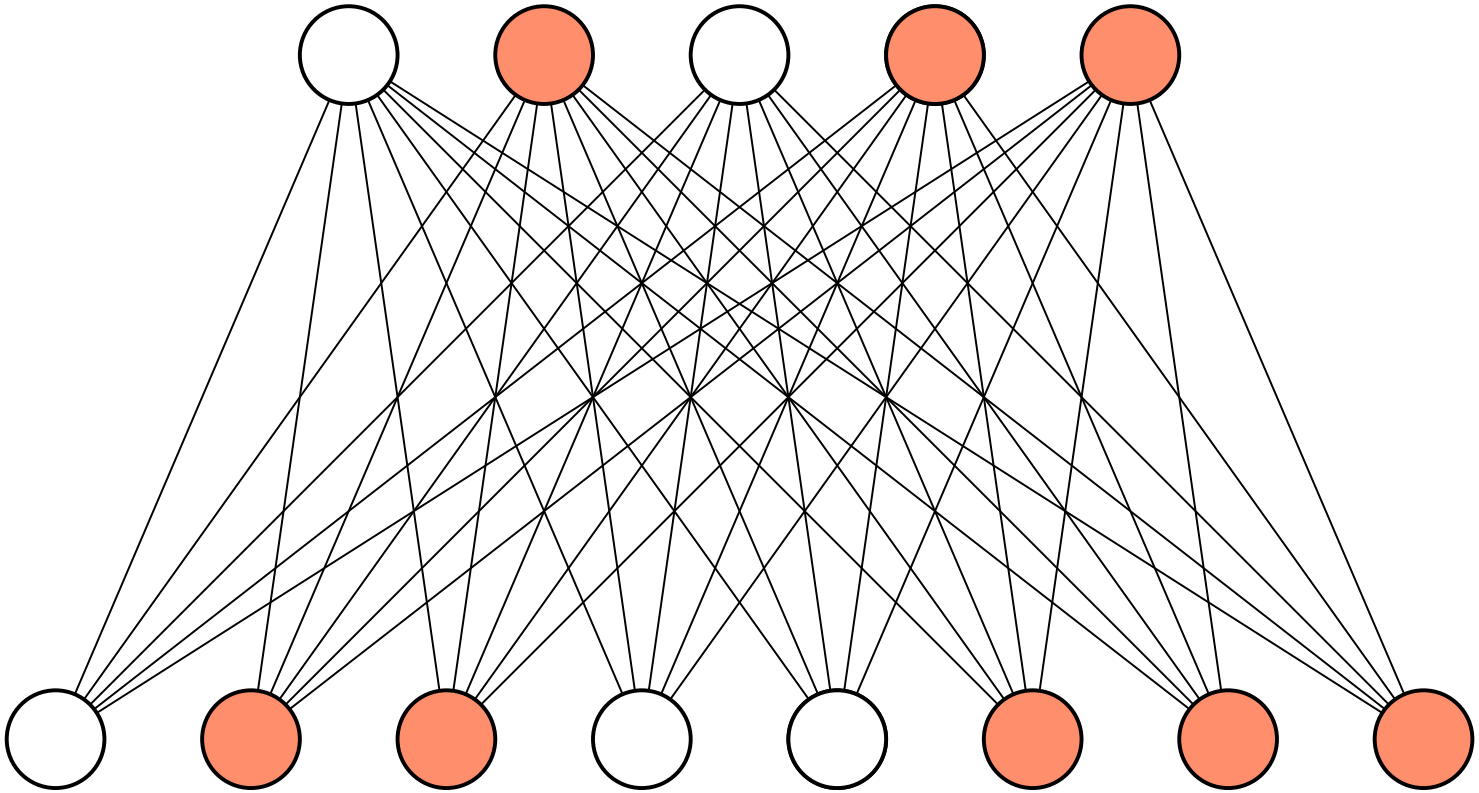
Then another matching phase starts, but the "eliminated" agents are ignored. The same protocol is repeated.

Leader Election in a Complete Bipartite Graph



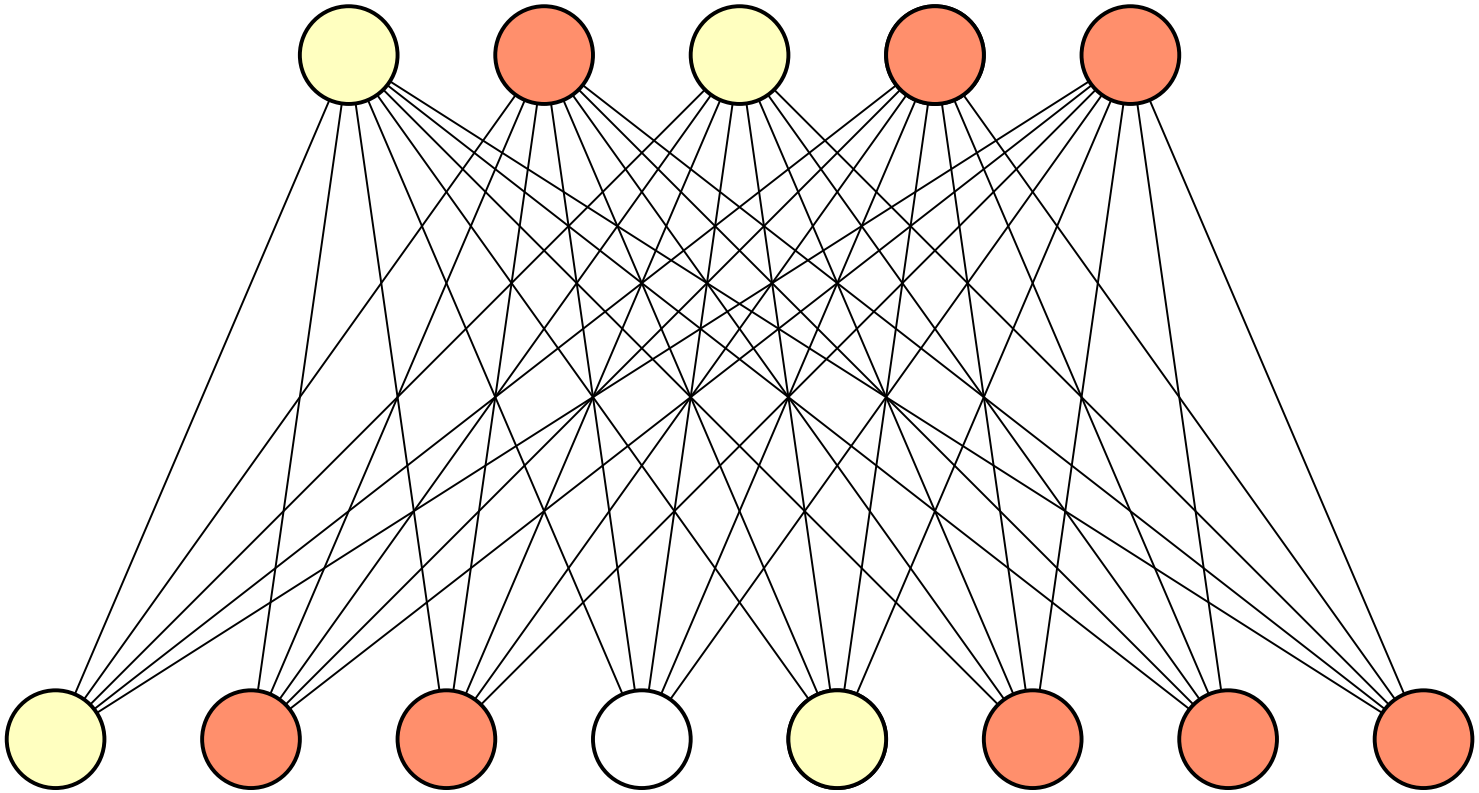
Then another matching phase starts, but the "eliminated" agents are ignored. The same protocol is repeated.

Leader Election in a Complete Bipartite Graph



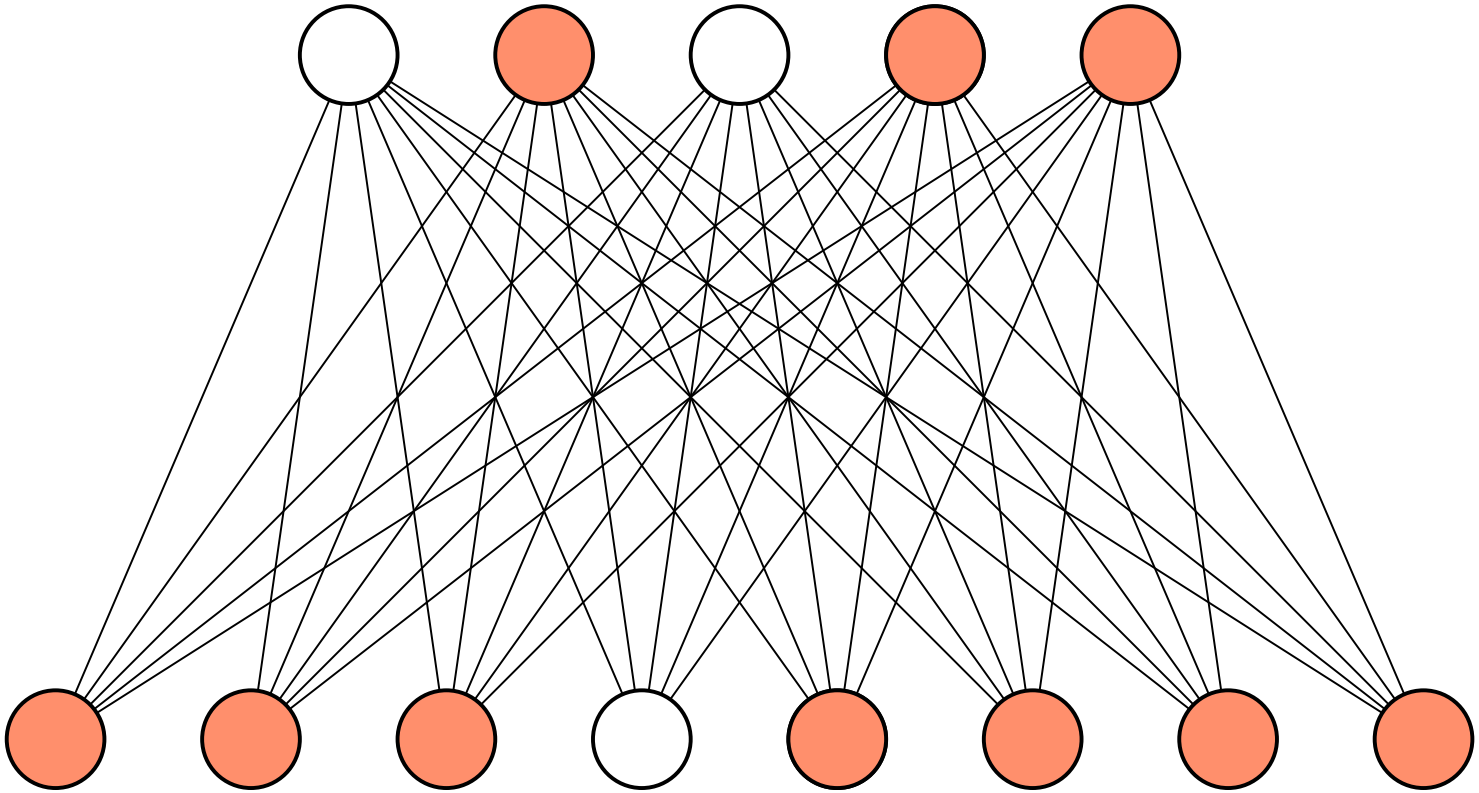
Then another matching phase starts, but the "eliminated" agents are ignored. The same protocol is repeated.

Leader Election in a Complete Bipartite Graph



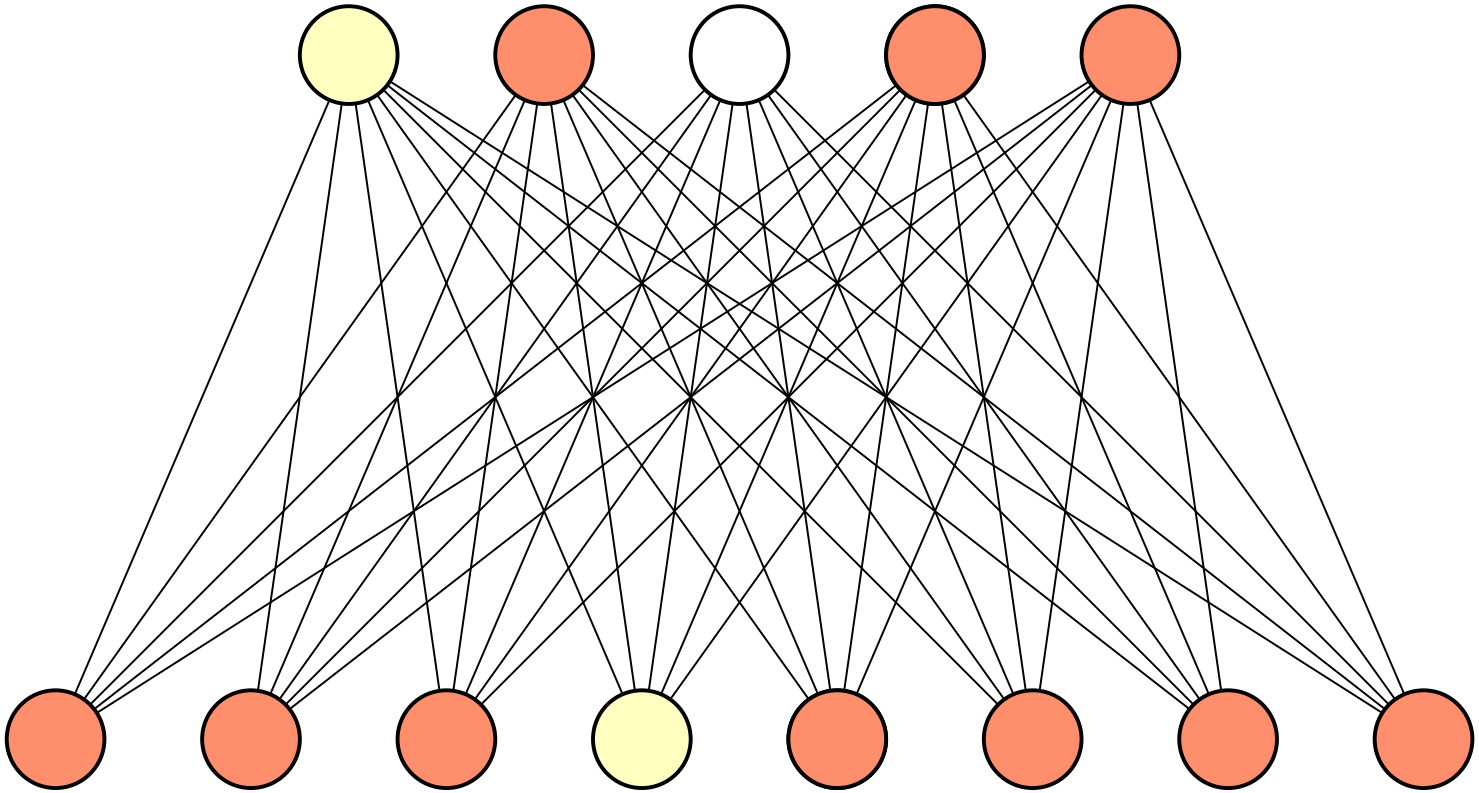
Then another matching phase starts, but the "eliminated" agents are ignored. The same protocol is repeated.

Leader Election in a Complete Bipartite Graph



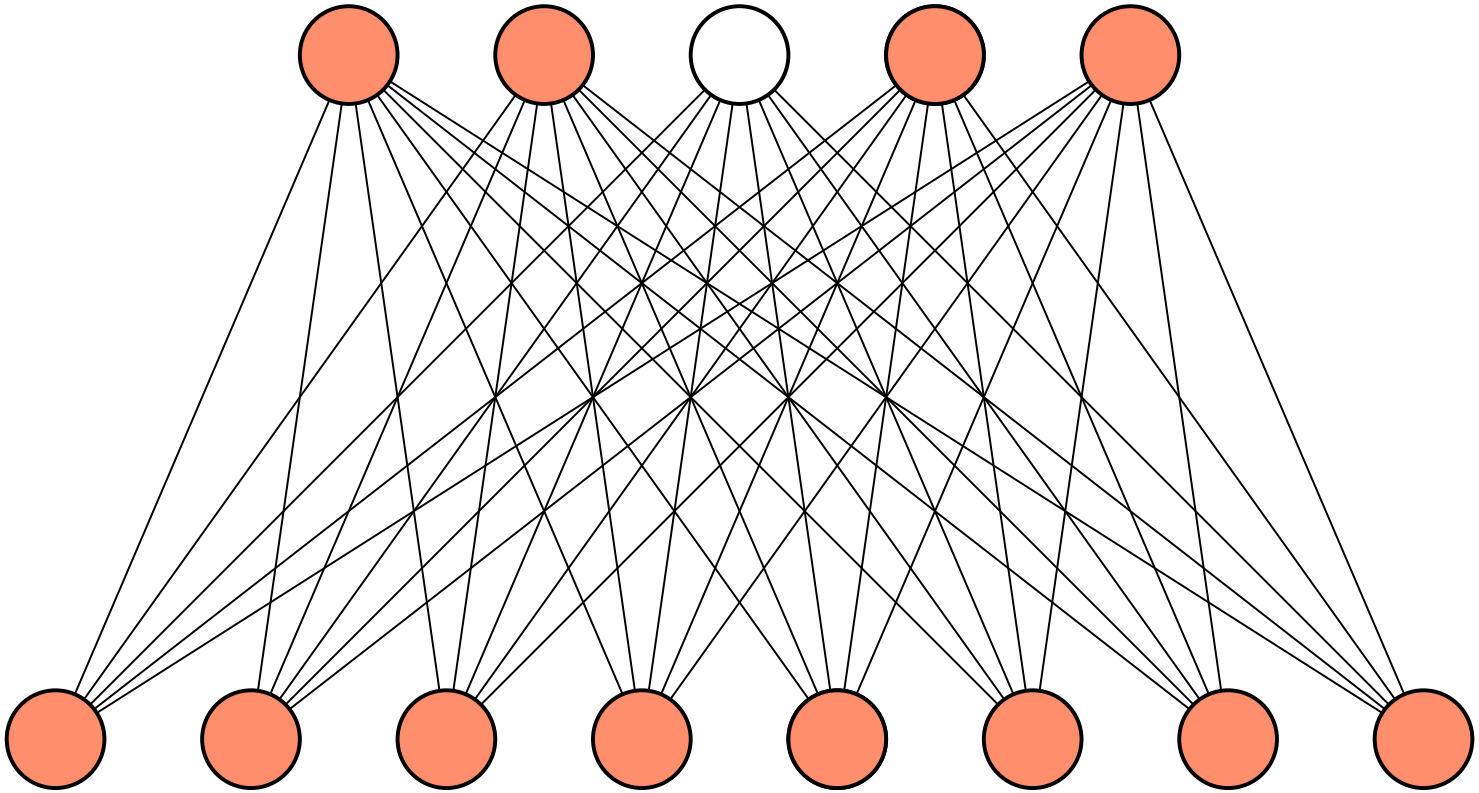
Then another matching phase starts, but the "eliminated" agents are ignored. The same protocol is repeated.

Leader Election in a Complete Bipartite Graph



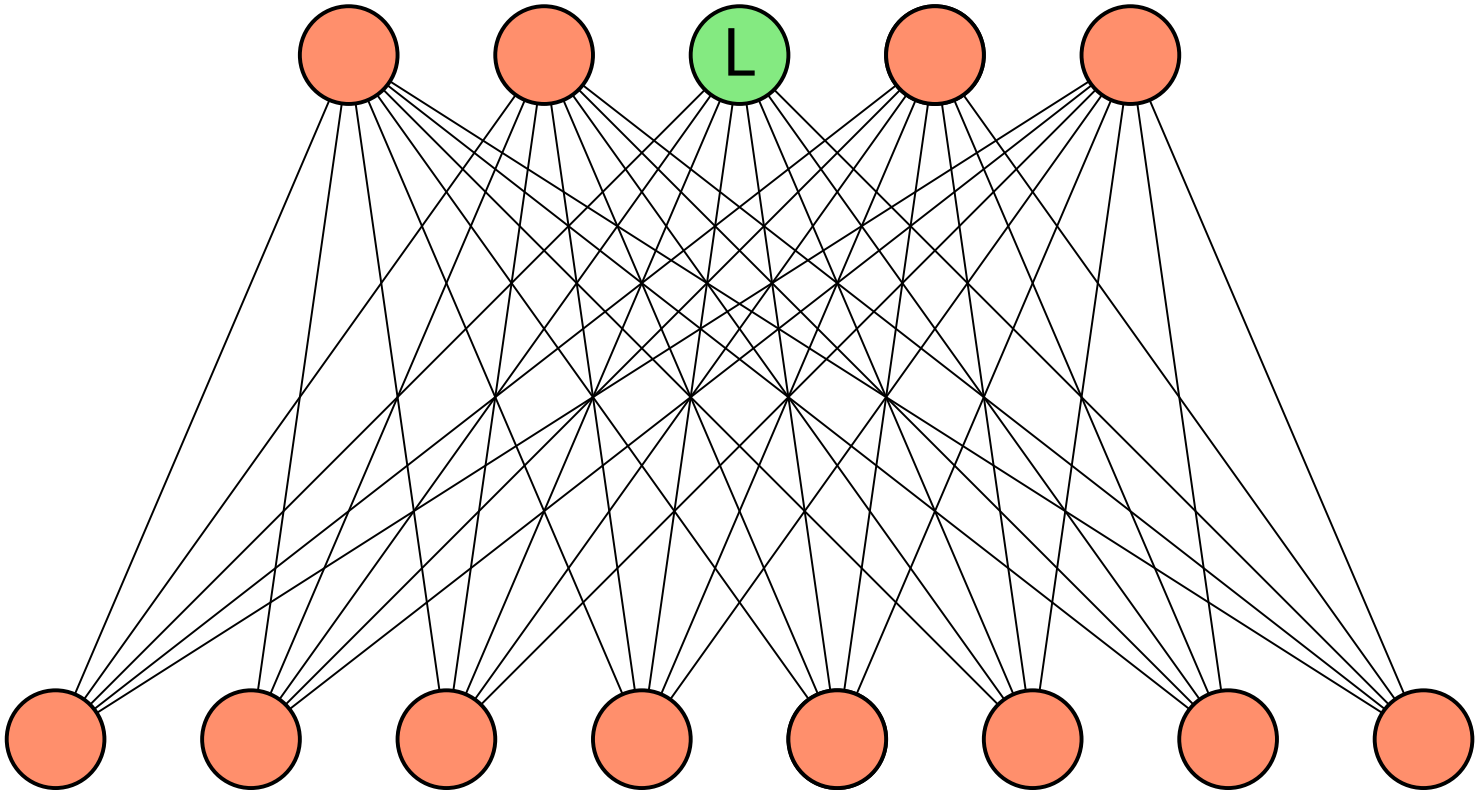
Then another matching phase starts, but the "eliminated" agents are ignored. The same protocol is repeated.

Leader Election in a Complete Bipartite Graph



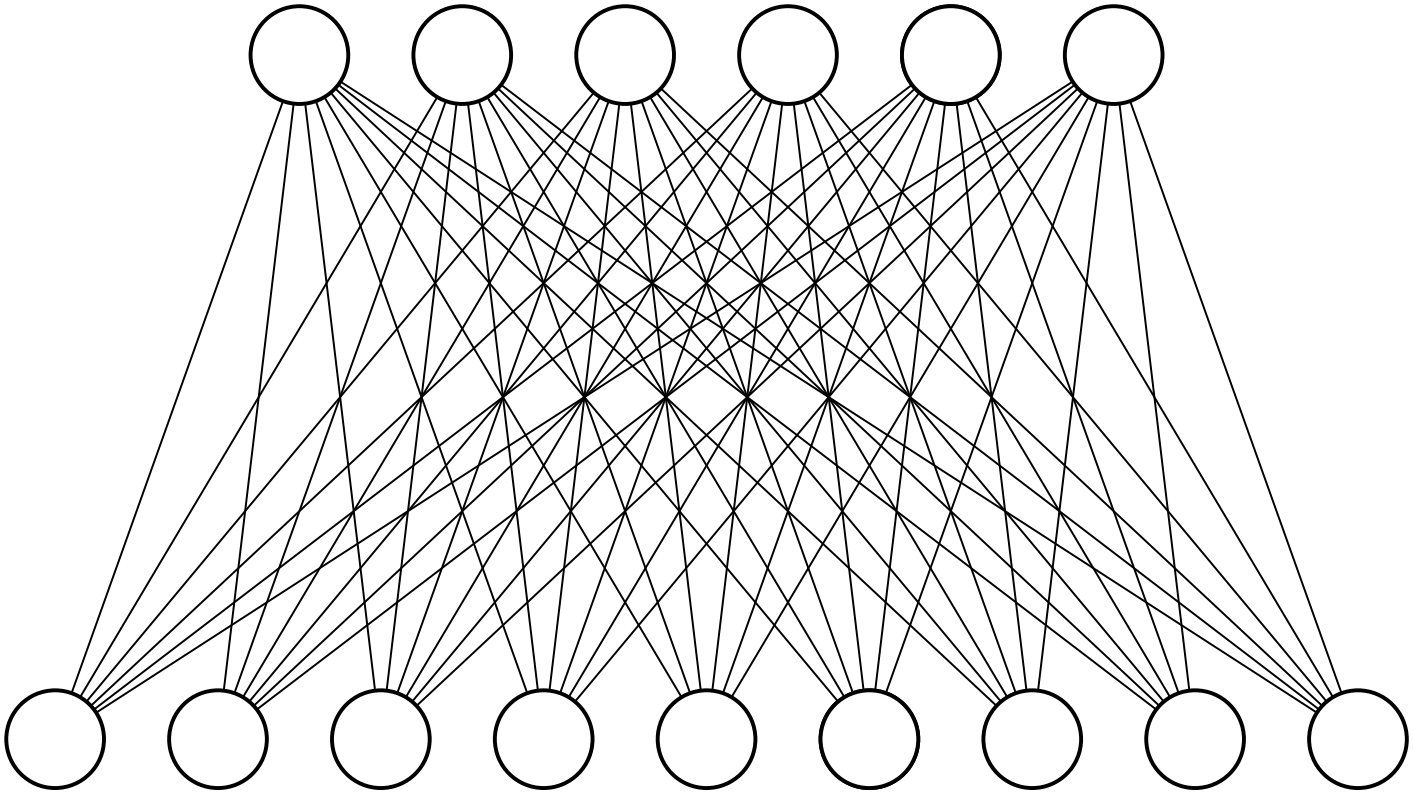
Since m and n are coprime, eventually only one agent will remain available.

Leader Election in a Complete Bipartite Graph



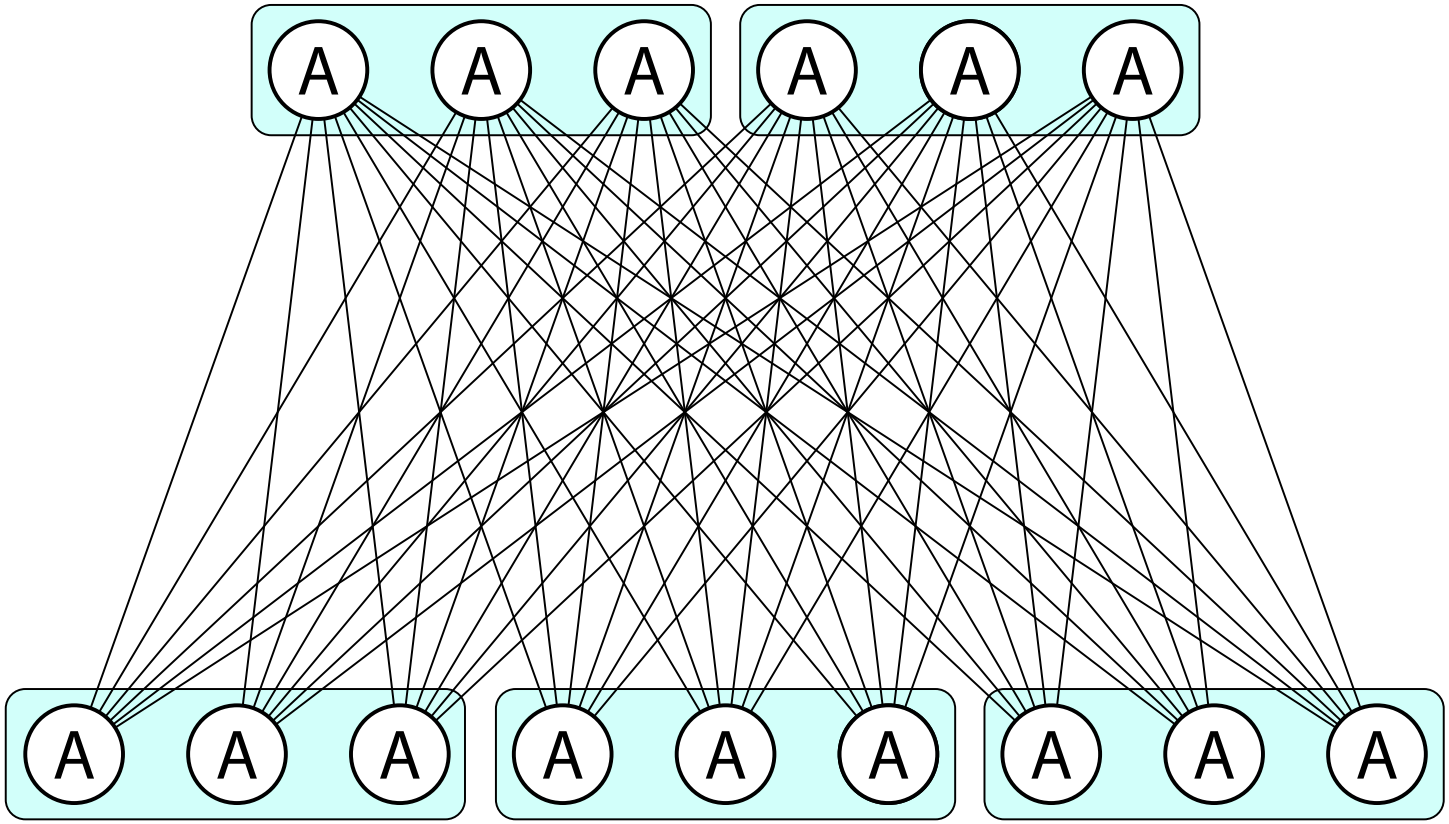
When this agent sees that all its neighbors are "eliminated", it becomes the leader. This protocol is terminating.

Leader Election in a Complete Bipartite Graph



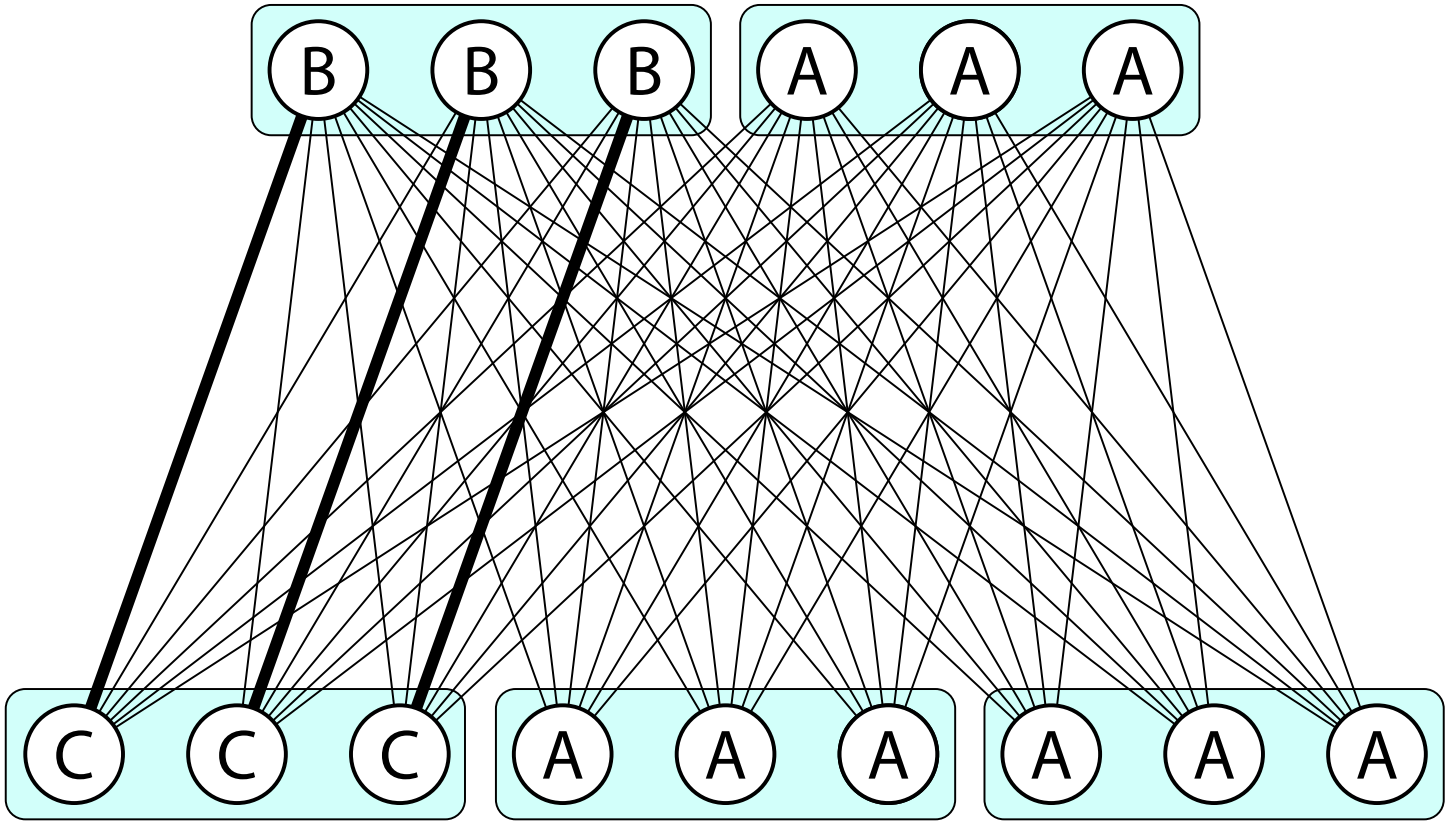
Suppose that m and n have a common divisor $d > 1$.

Leader Election in a Complete Bipartite Graph



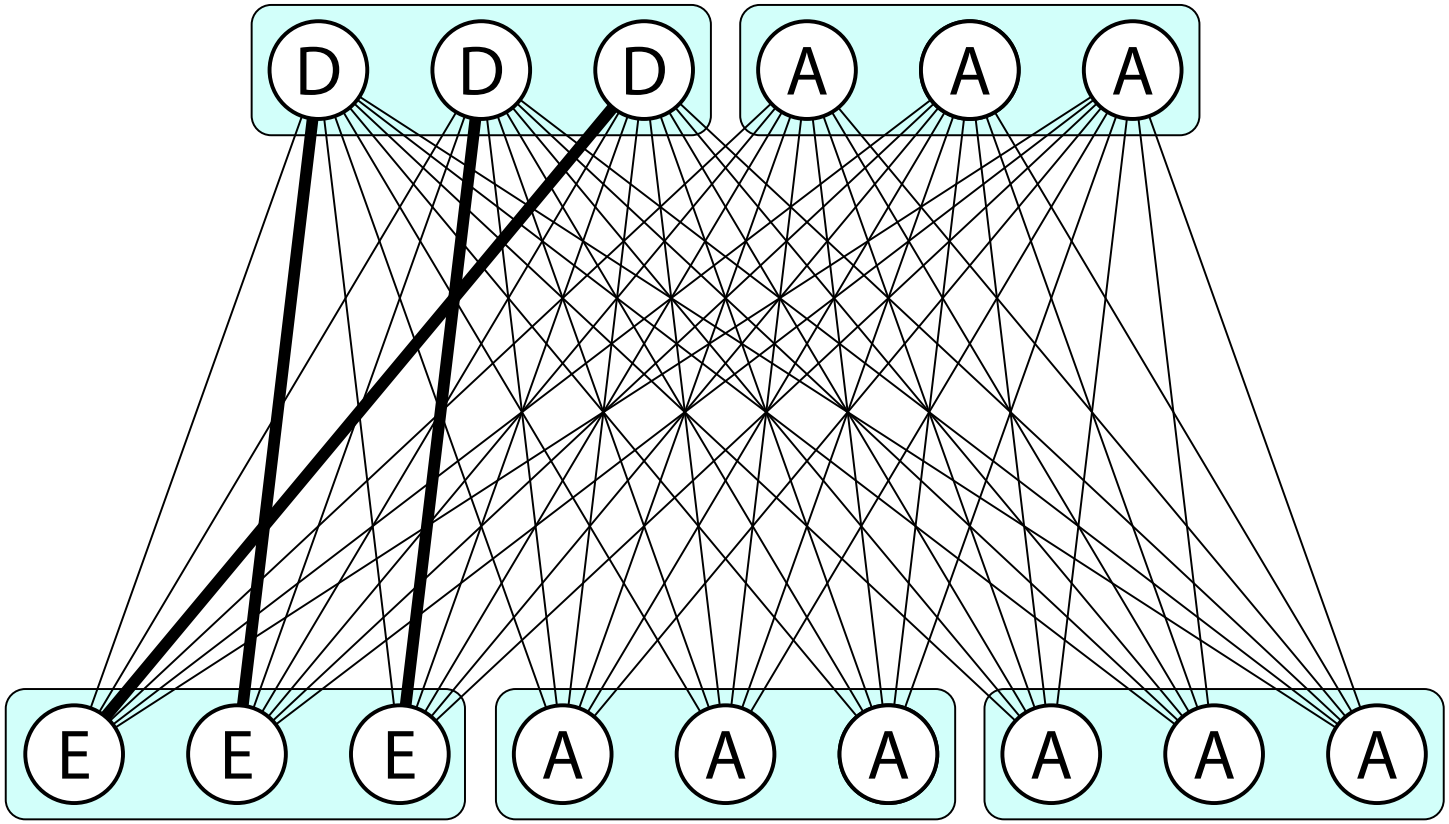
We partition each side of the network into groups of d agents, and we assign all agents the same initial state.

Leader Election in a Complete Bipartite Graph



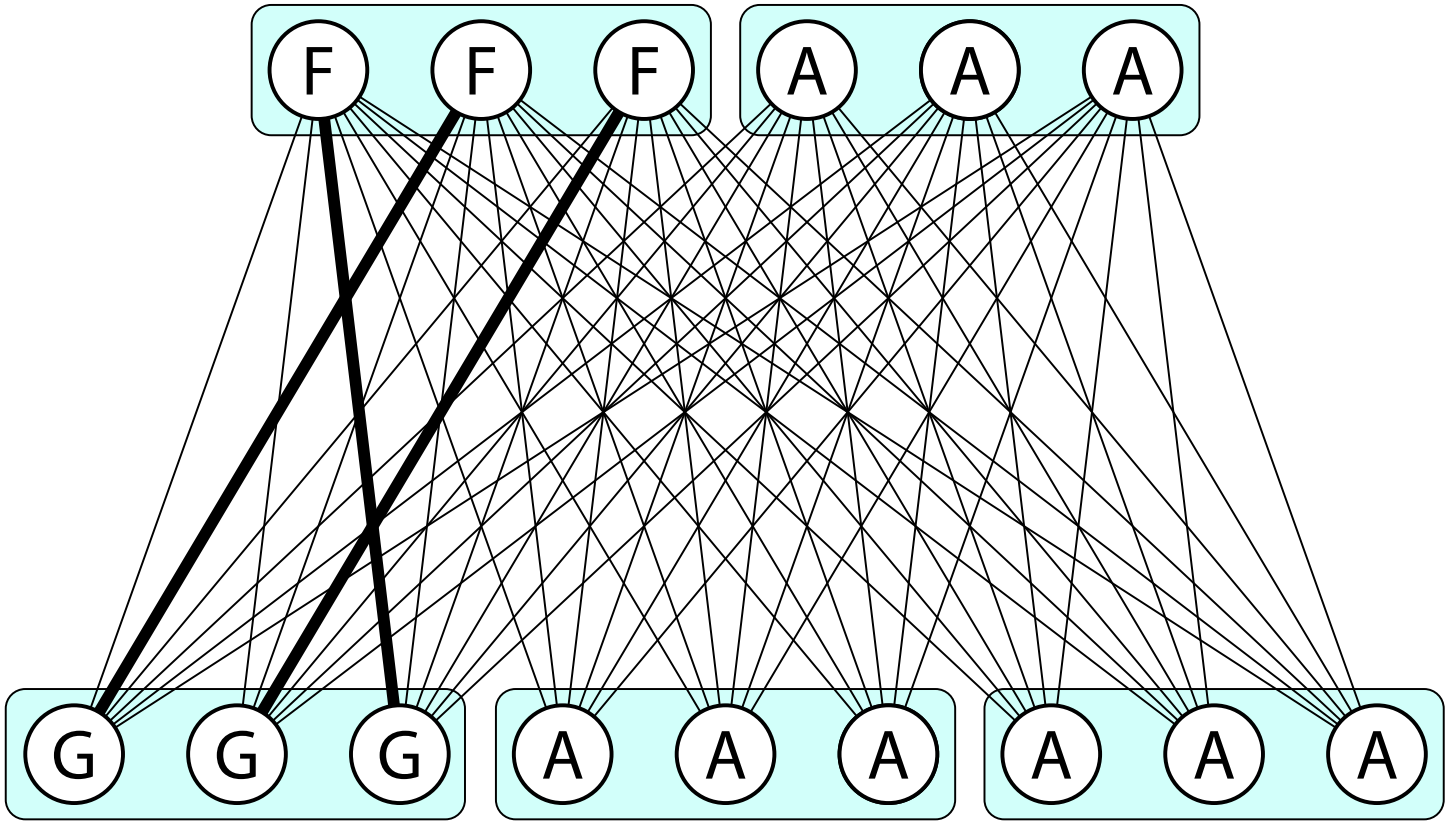
The scheduler chooses two groups on opposite sides, and activates the agents according to a perfect matching.

Leader Election in a Complete Bipartite Graph



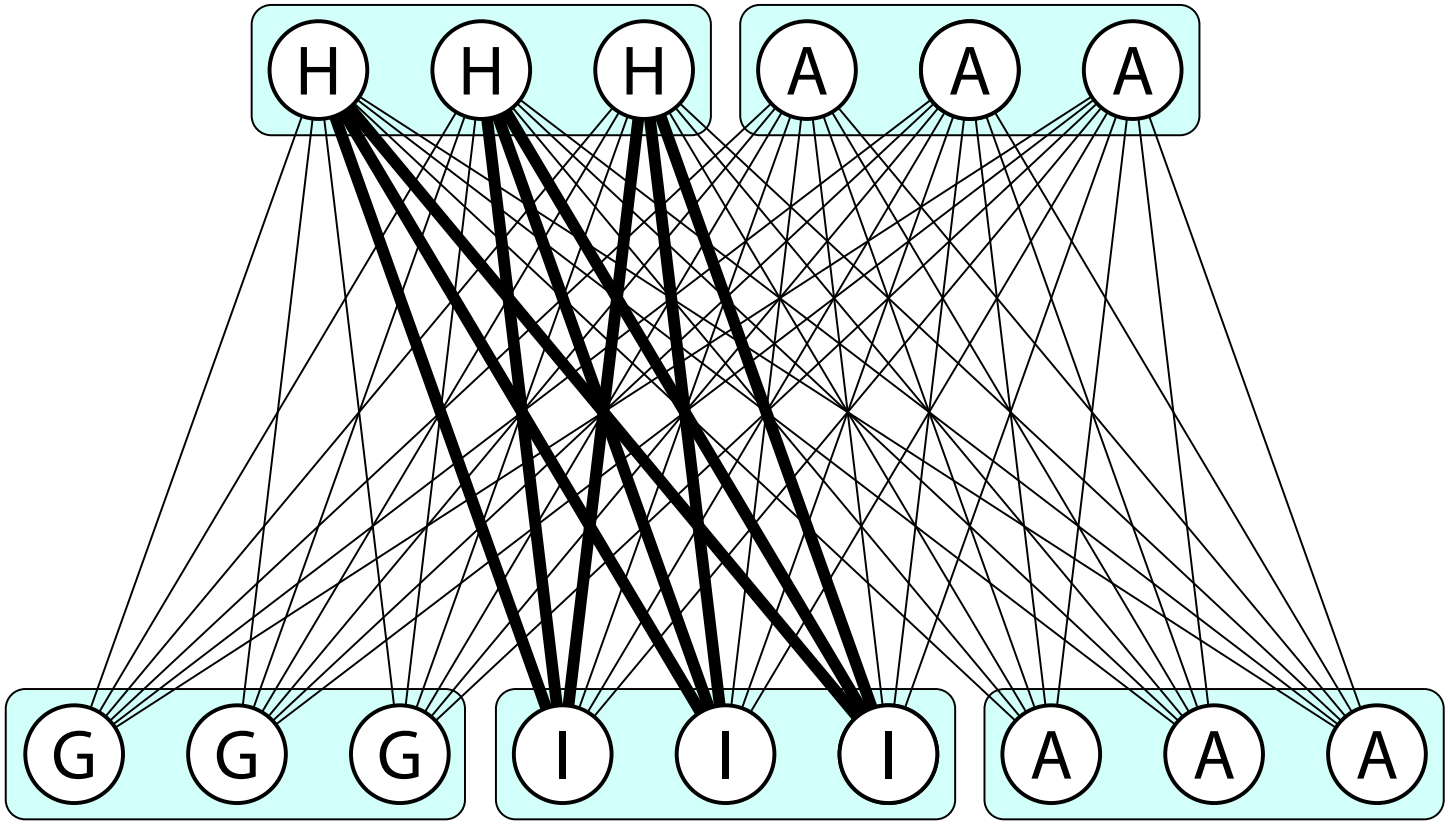
Then it chooses another perfect matching, and so on, until all pairs of neighbors have been activated.

Leader Election in a Complete Bipartite Graph



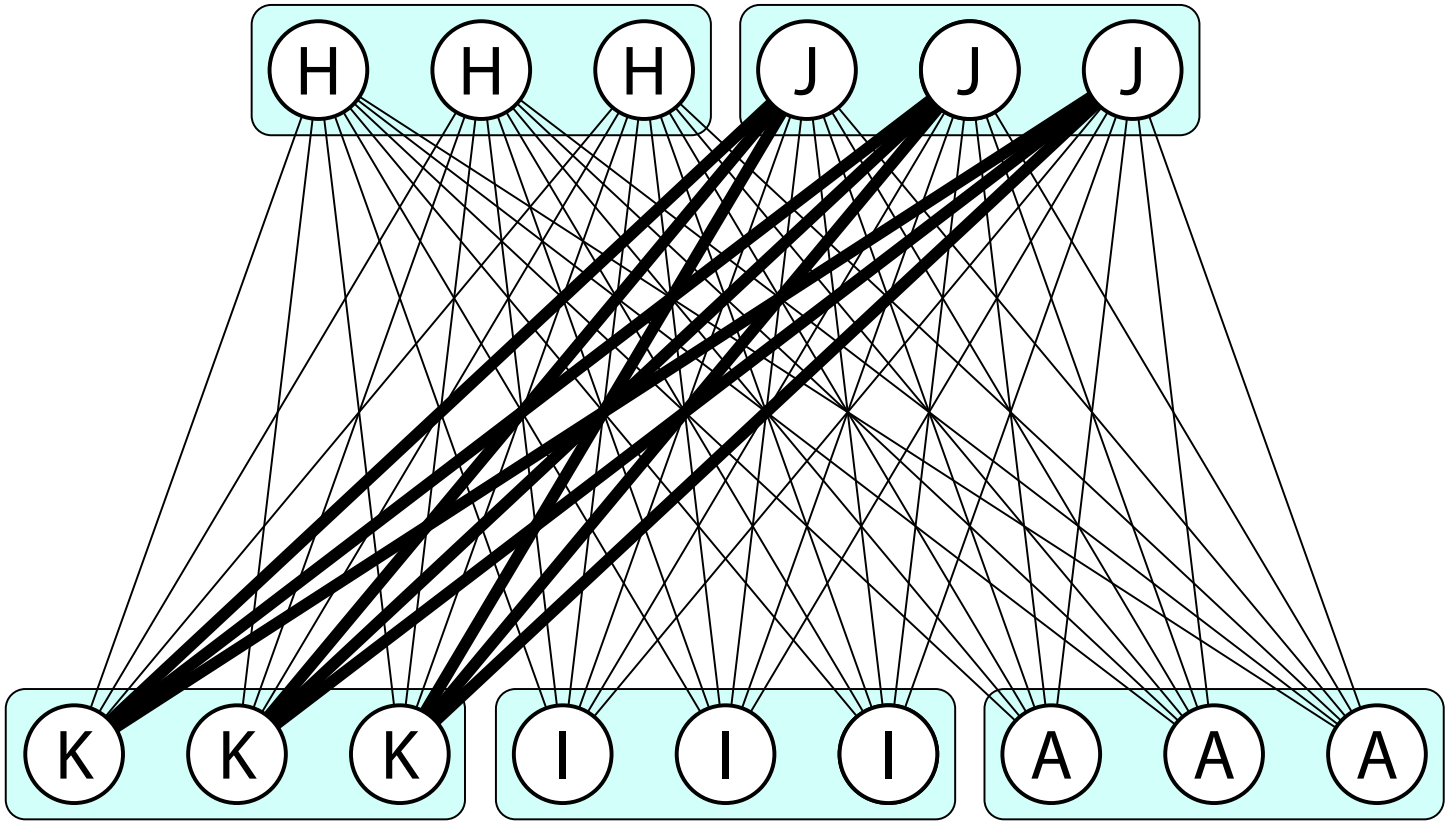
Then it chooses another perfect matching, and so on, until all pairs of neighbors have been activated.

Leader Election in a Complete Bipartite Graph



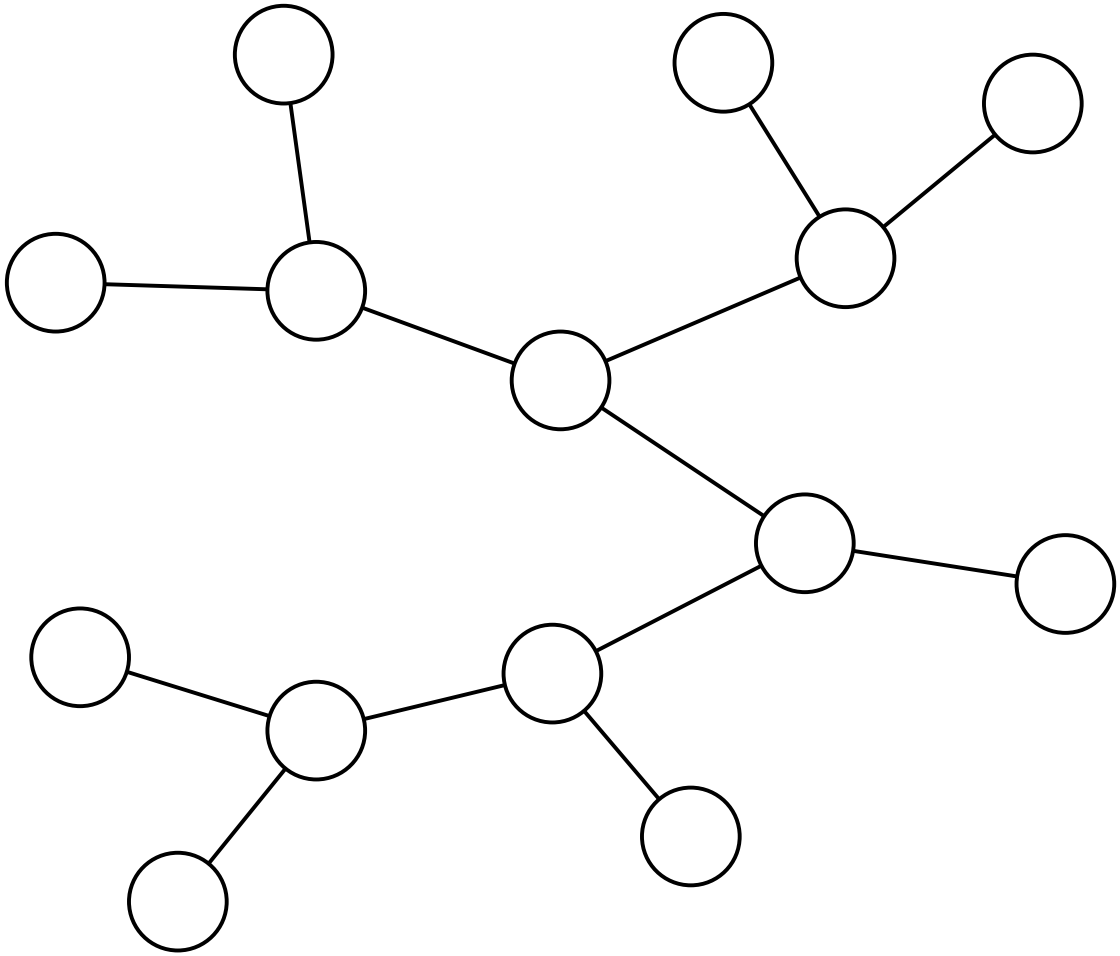
Then it does the same with two other groups, and so on, until all pairs of neighbors have been activated.

Leader Election in a Complete Bipartite Graph



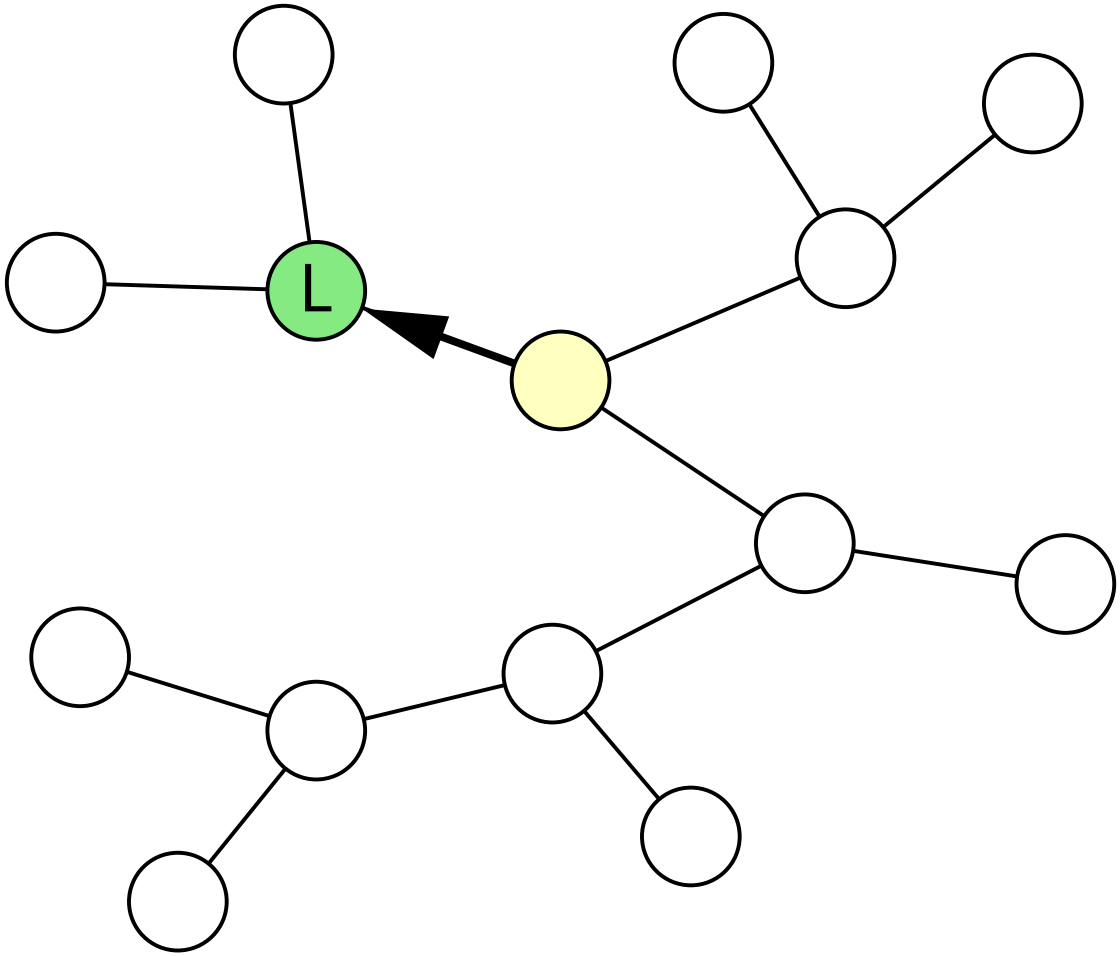
Every d interactions, all agents in the same group have the same state. Hence a leader cannot be elected.

Leader Election in a Tree



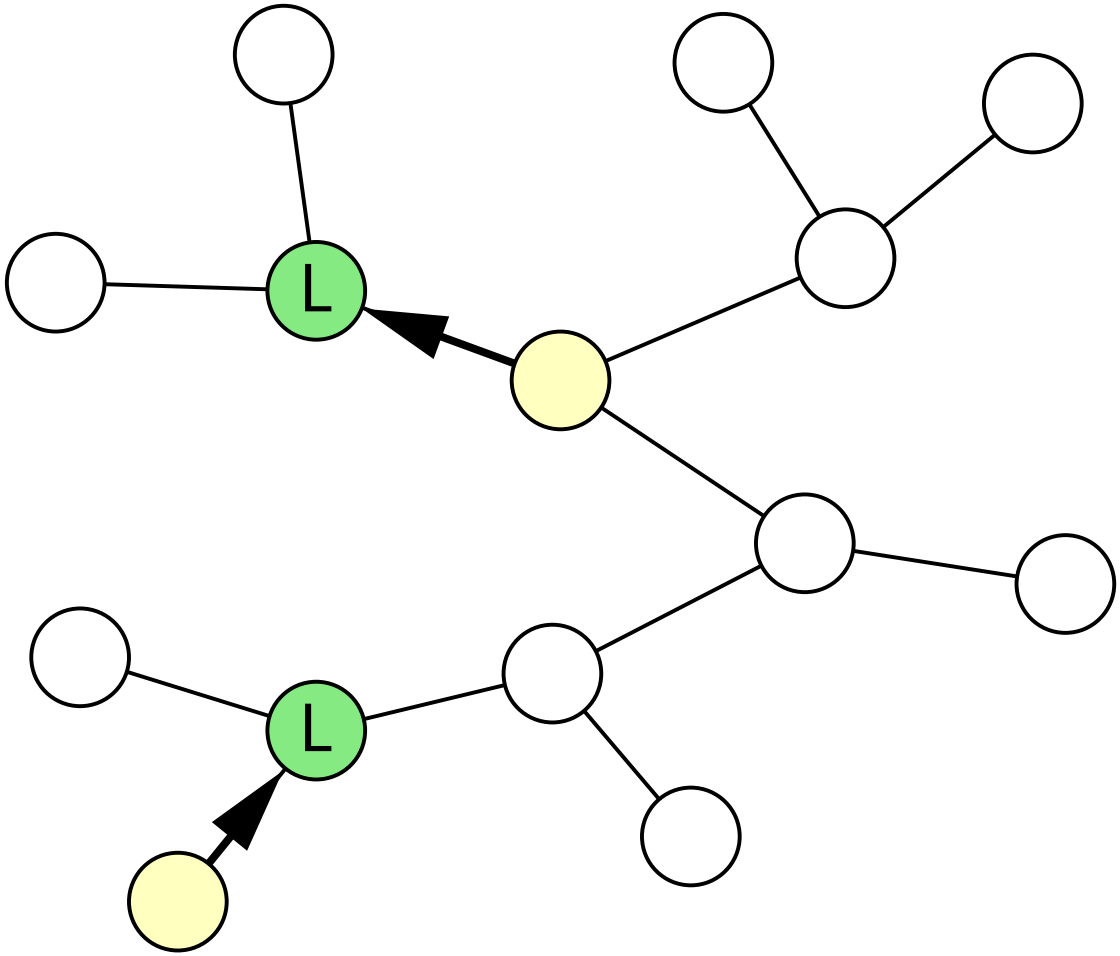
Theorem: in a tree, it is possible to elect a leader under the recurrent scheduler.

Leader Election in a Tree



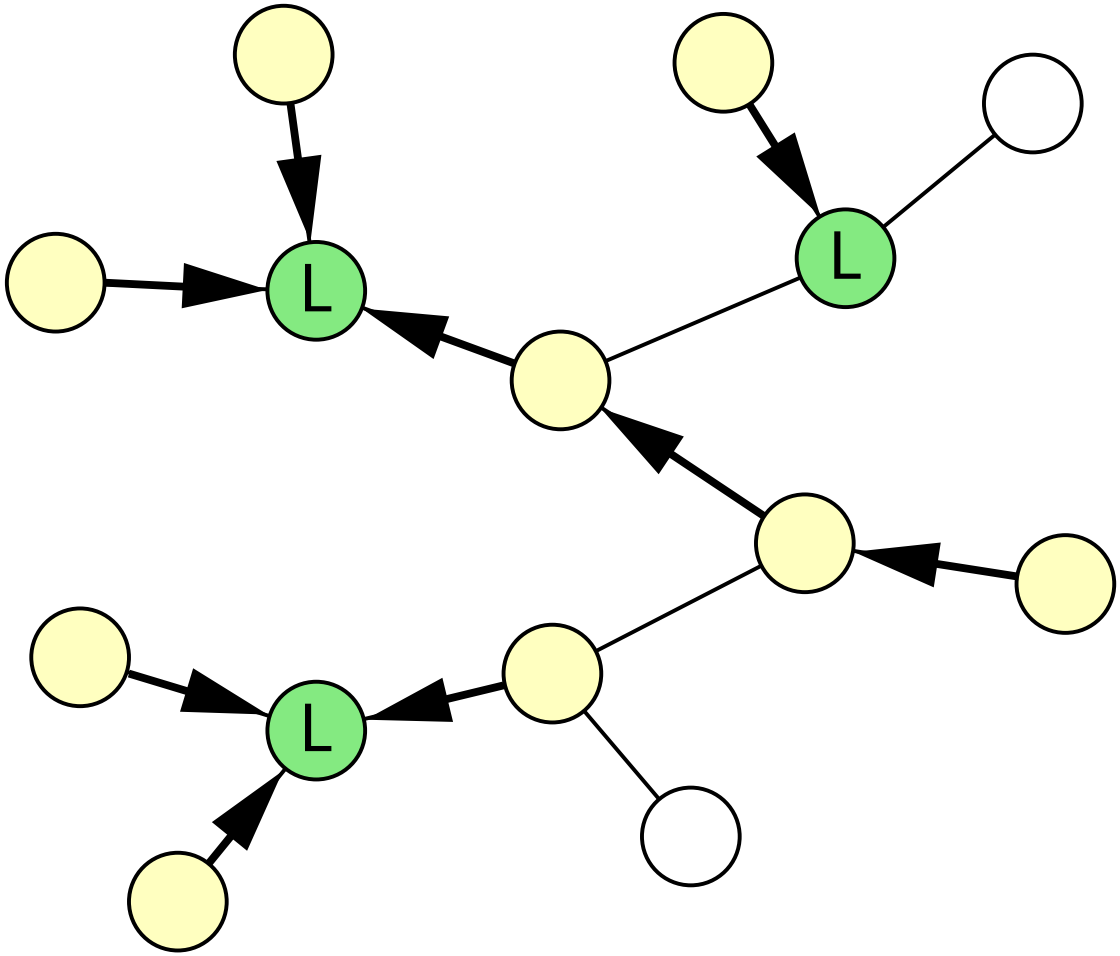
When two agents meet, they form a small rooted tree, where the root is a leader.

Leader Election in a Tree



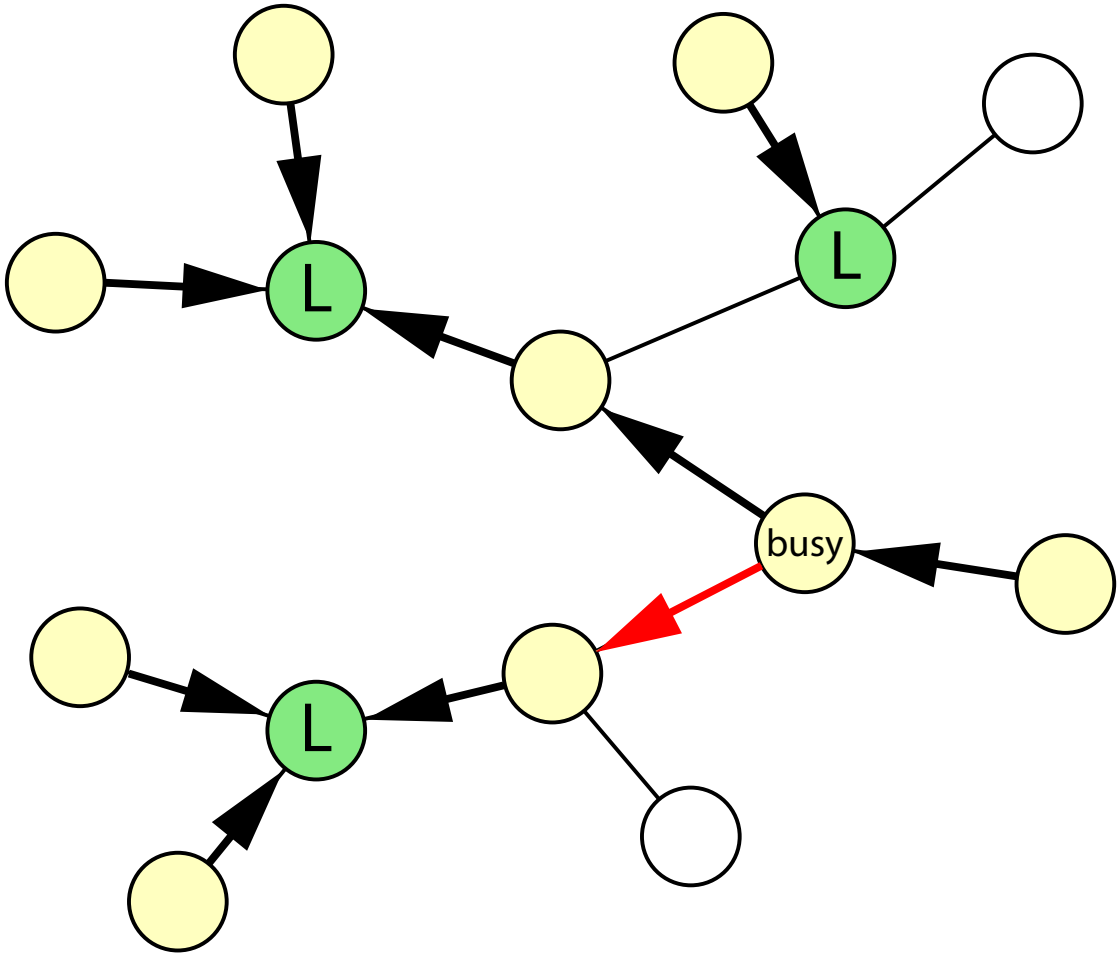
Arrows are encoded as port states.

Leader Election in a Tree



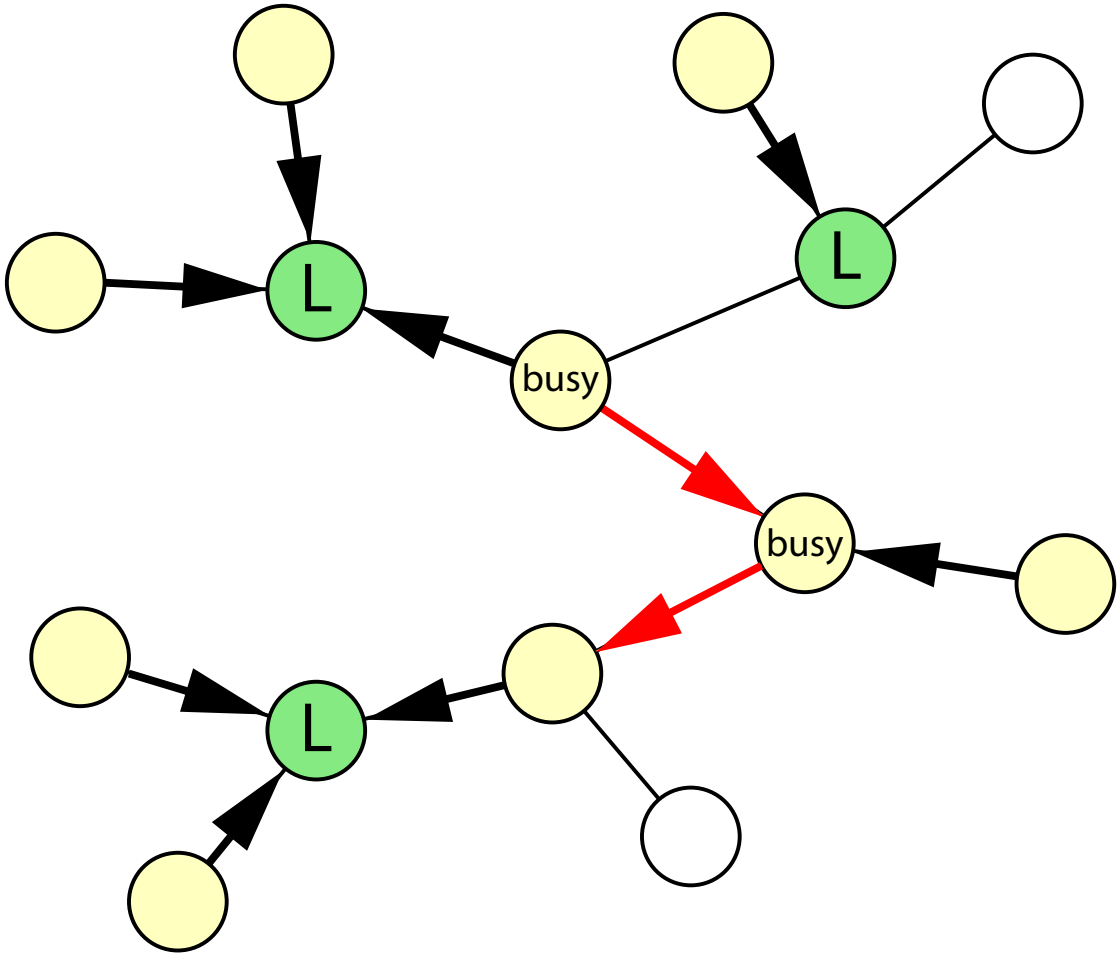
New agents may join existing trees, and a forest is formed.

Leader Election in a Tree



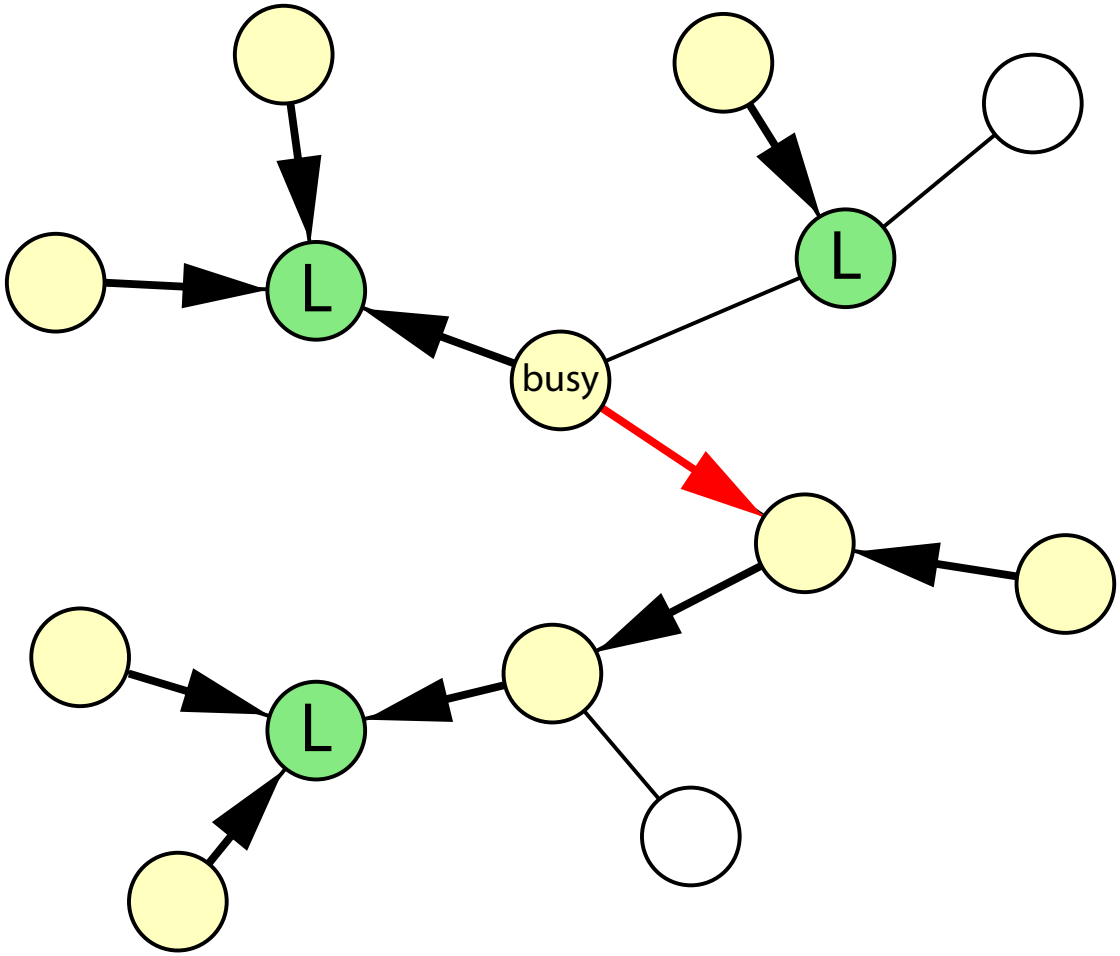
When two trees merge, one agent becomes “busy”. Its task is to tell its leader that it is no longer a leader.

Leader Election in a Tree



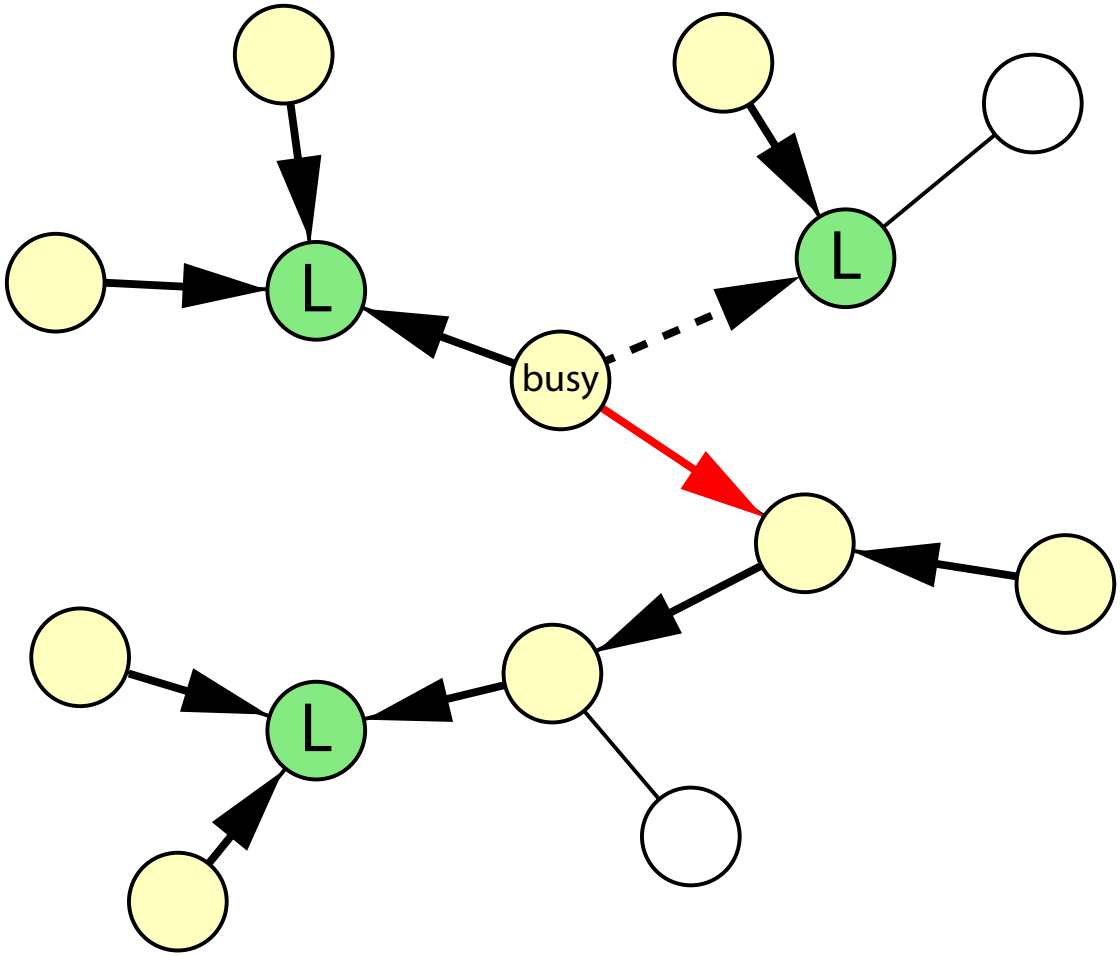
The parent of a busy agent becomes busy too, and reverses the corresponding arrow.

Leader Election in a Tree



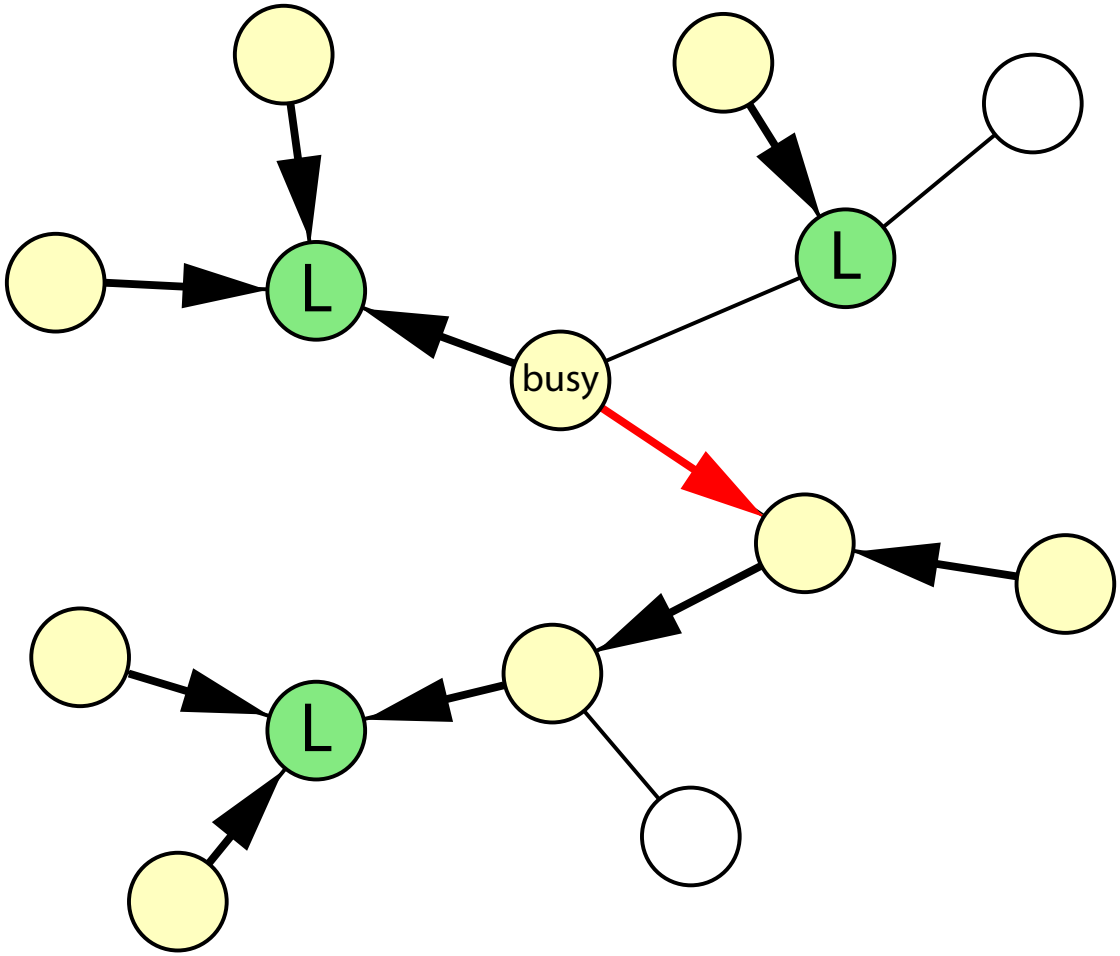
The child then ceases to be busy.

Leader Election in a Tree



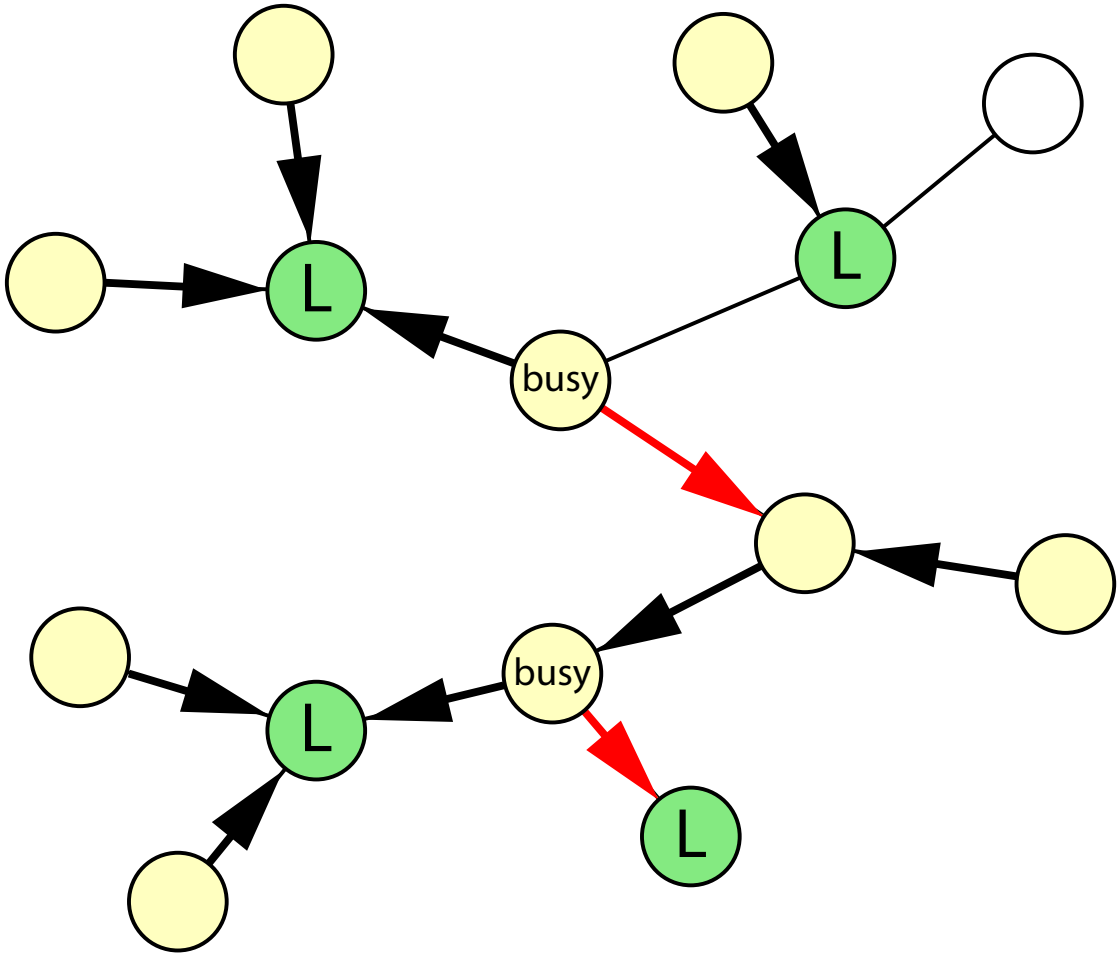
A busy agent rejects all requests to merge.

Leader Election in a Tree



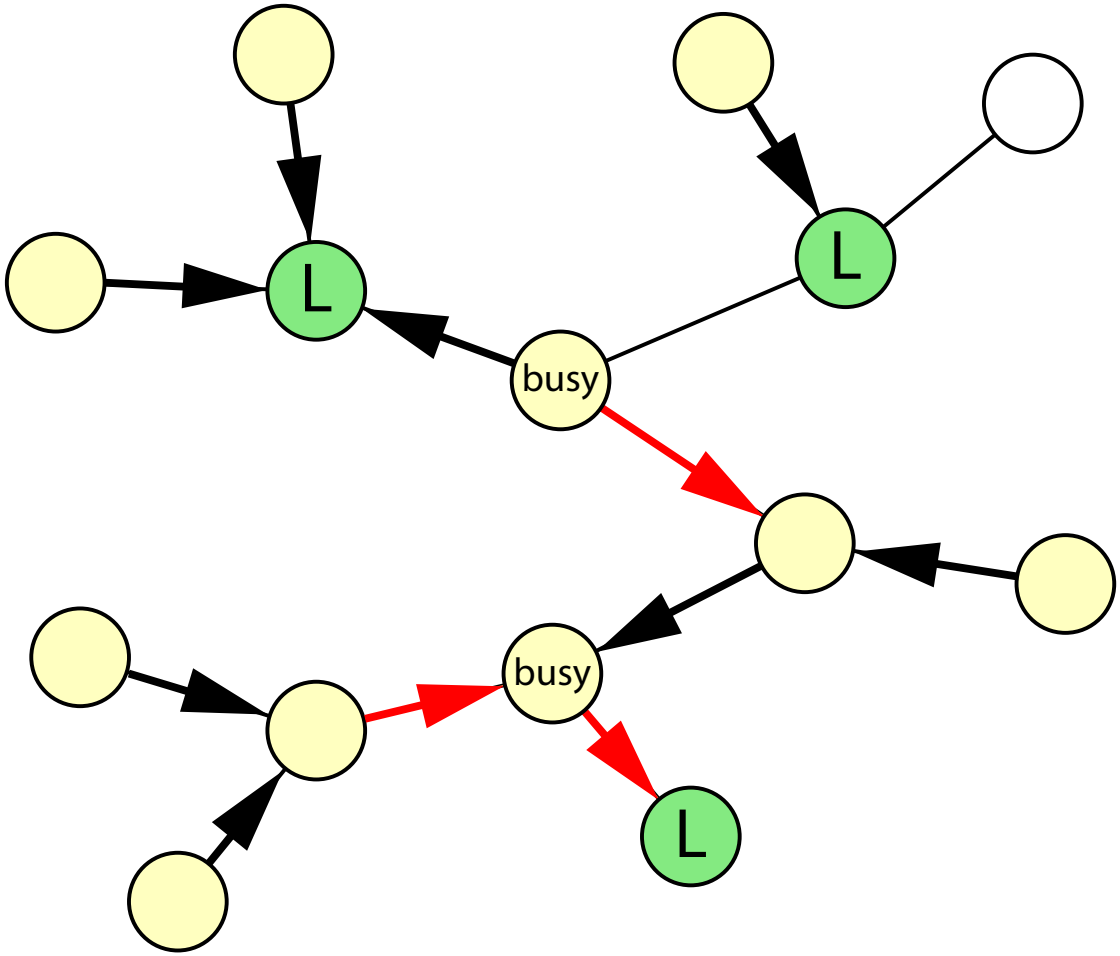
A busy agent rejects all requests to merge.

Leader Election in a Tree



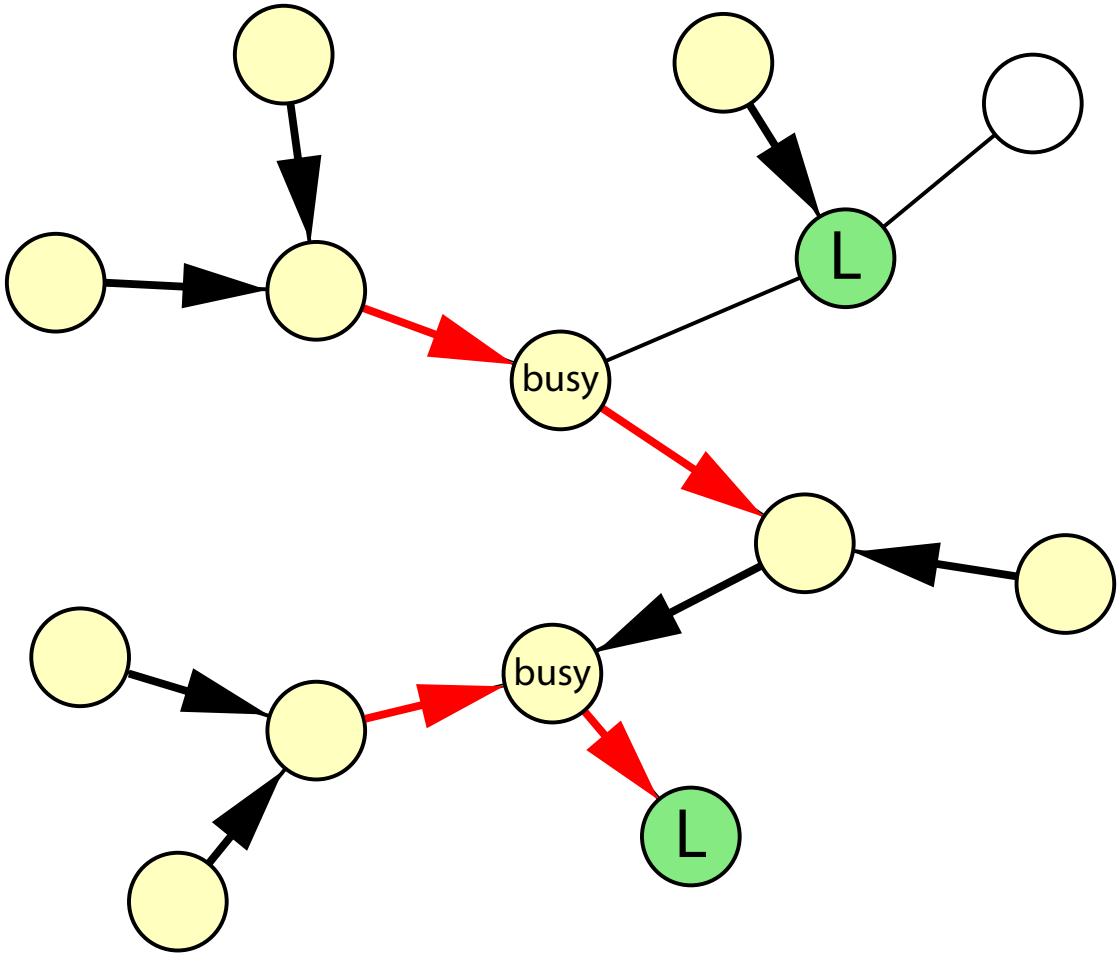
When a leader notices that one of its children is busy, it stops being a leader.

Leader Election in a Tree



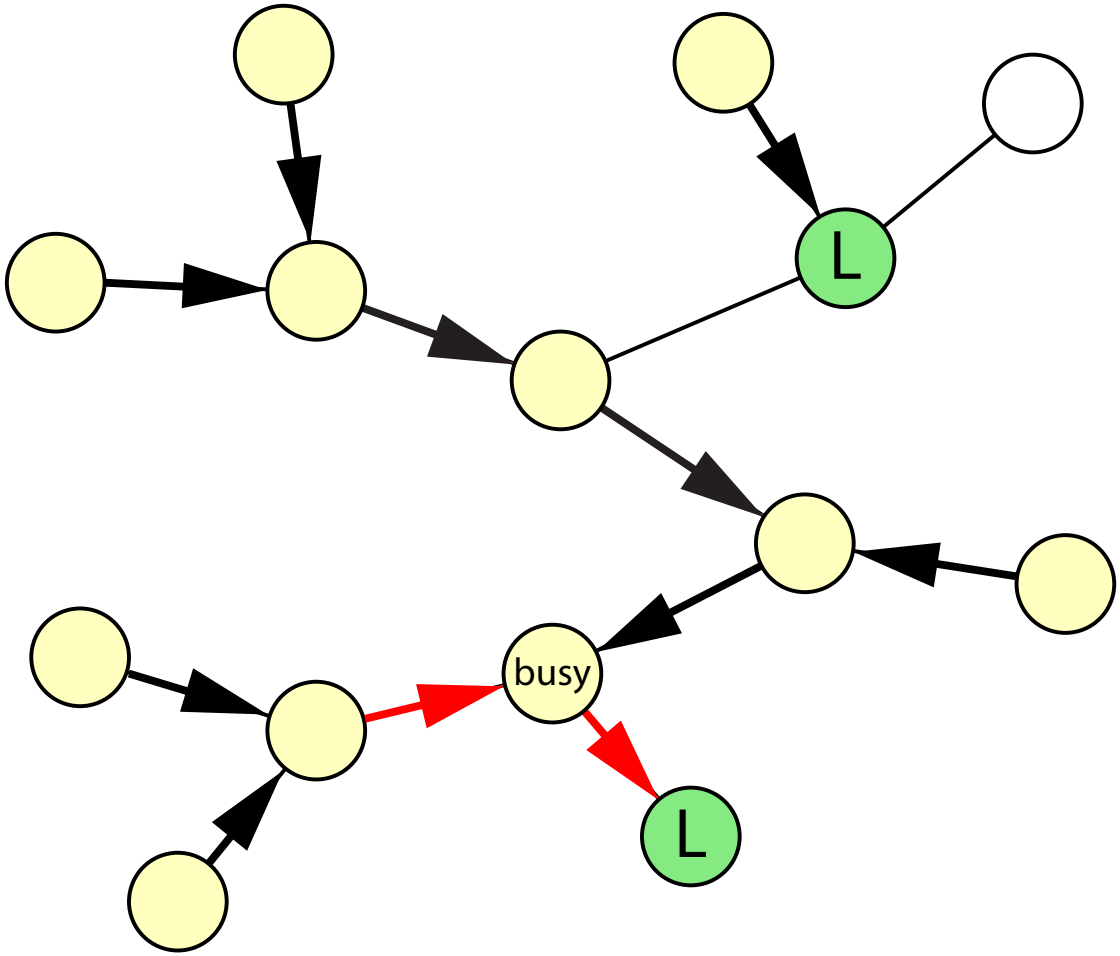
When a leader notices that one of its children is busy, it stops being a leader.

Leader Election in a Tree



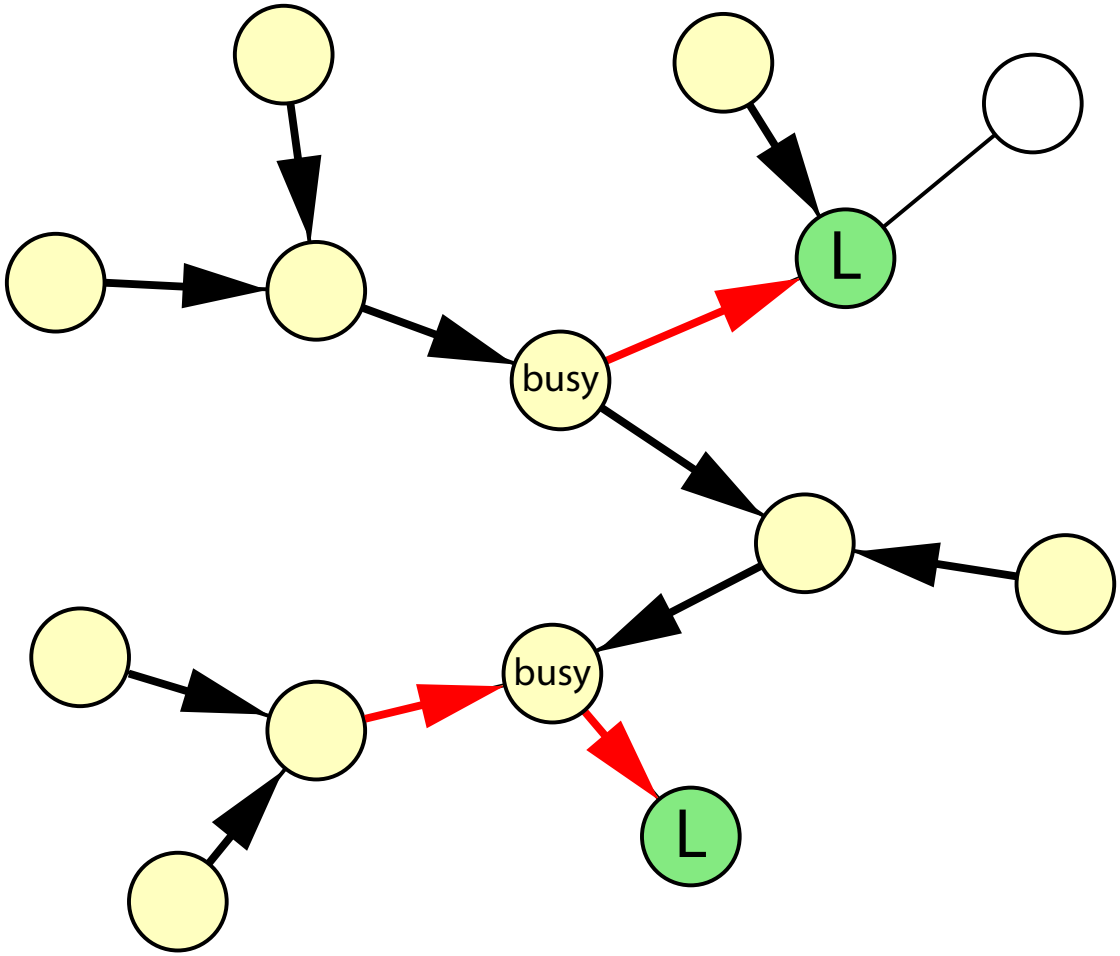
When a leader notices that one of its children is busy, it stops being a leader.

Leader Election in a Tree



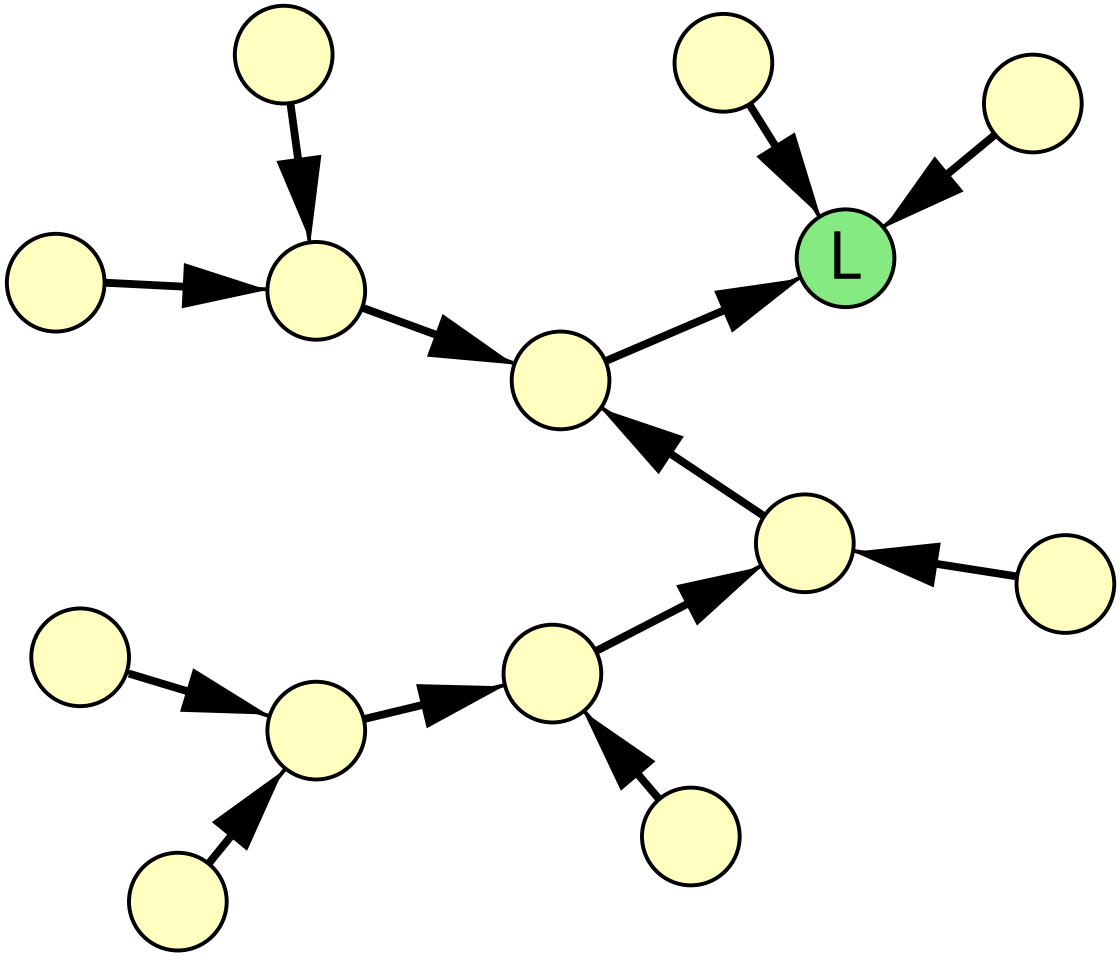
When a leader notices that one of its children is busy, it stops being a leader.

Leader Election in a Tree



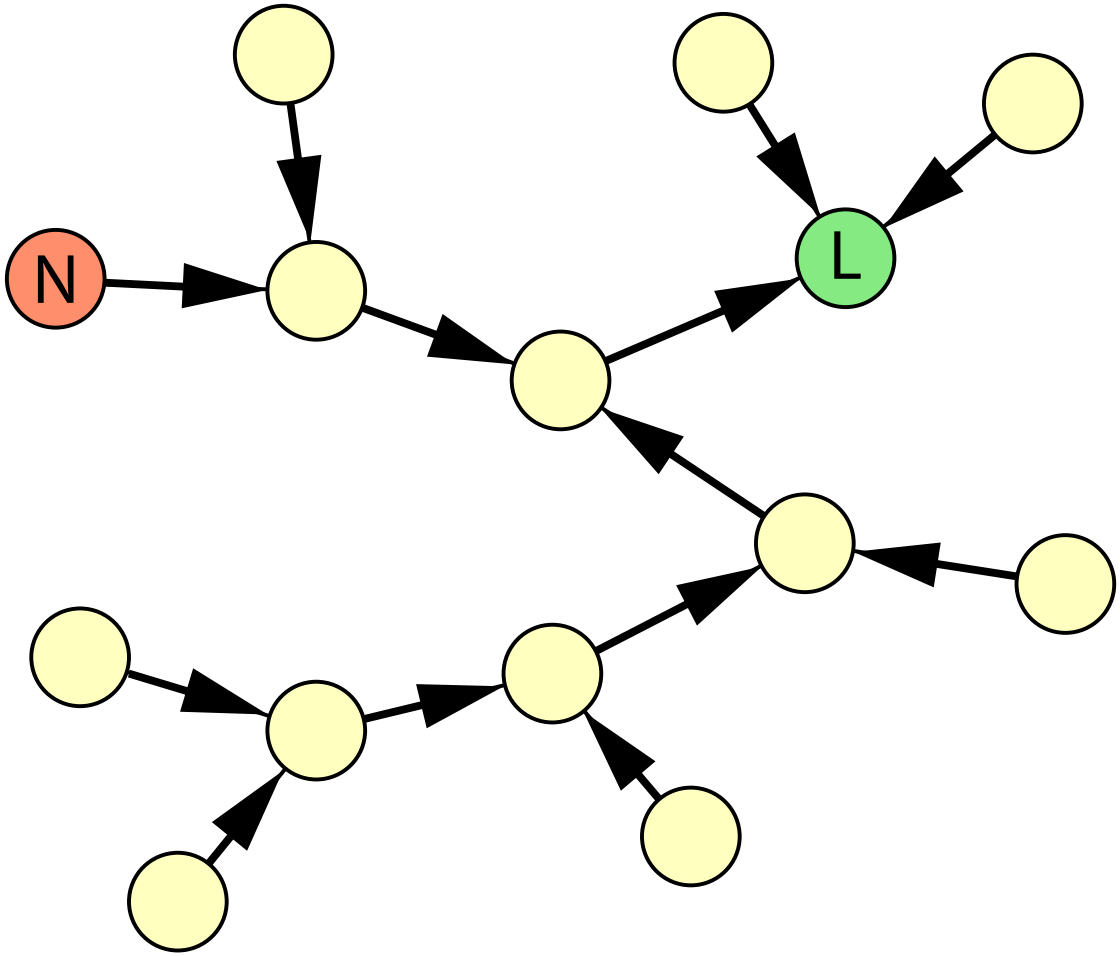
Agents that are no longer busy accept new merge requests.

Leader Election in a Tree



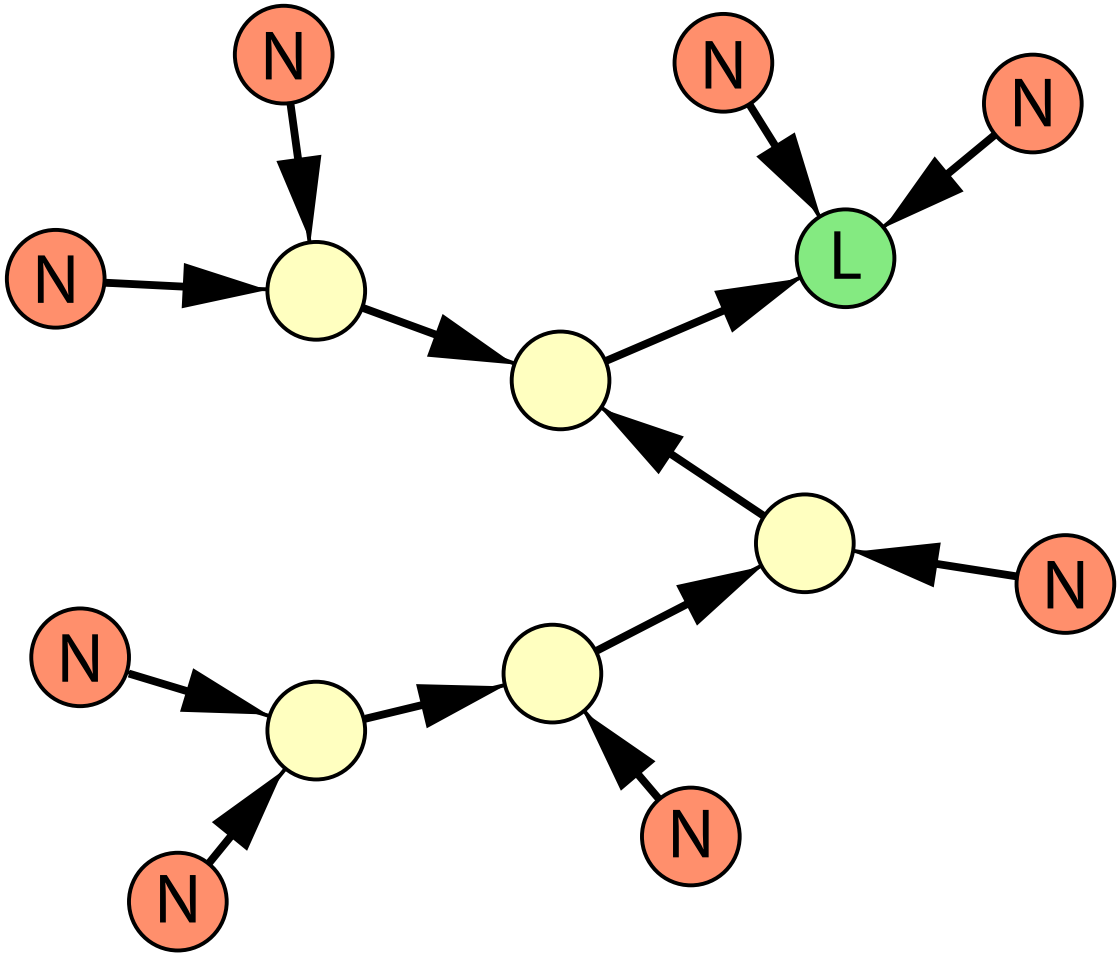
Eventually, only one leader is left, and the whole tree is oriented toward it.

Leader Election in a Tree



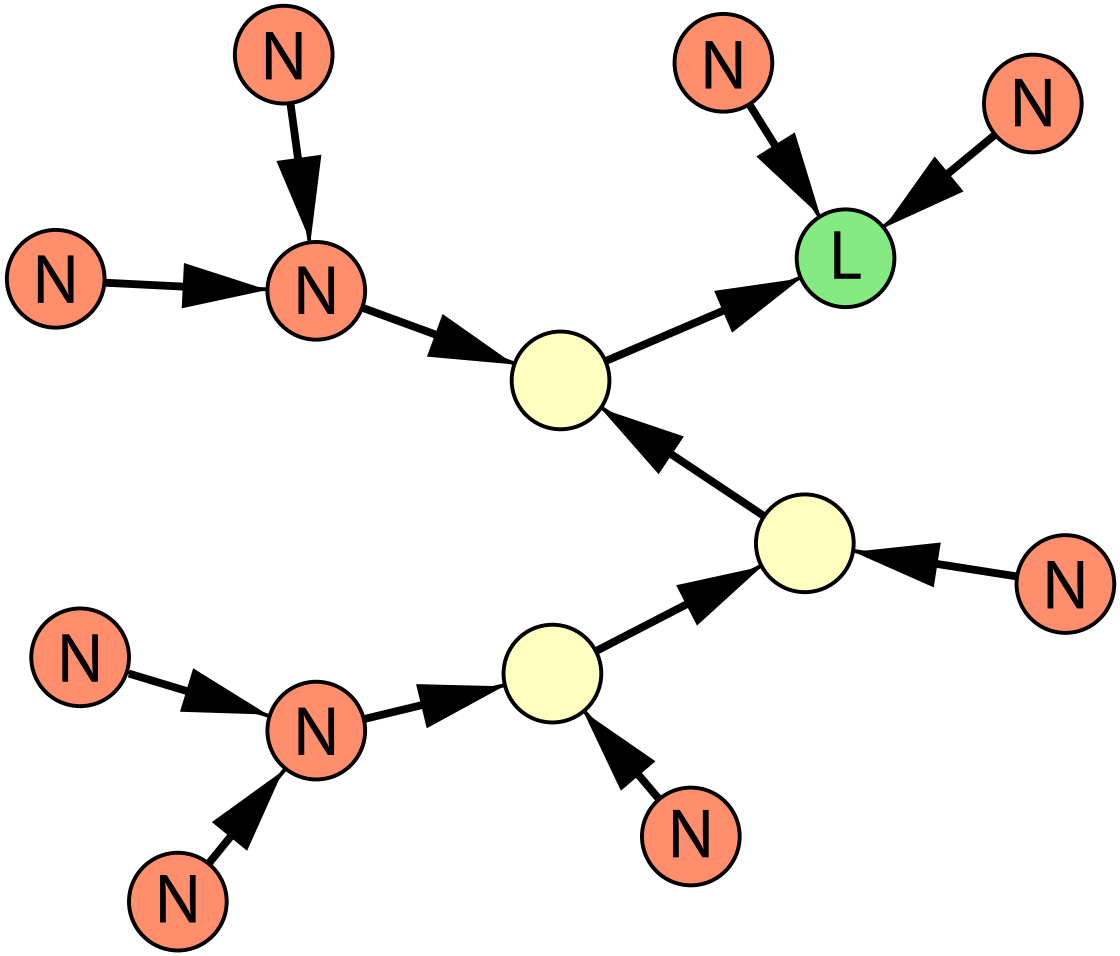
If the scheduler is k-bounded, a leaf eventually knows that it is a leaf. A non-leader leaf can safely terminate.

Leader Election in a Tree



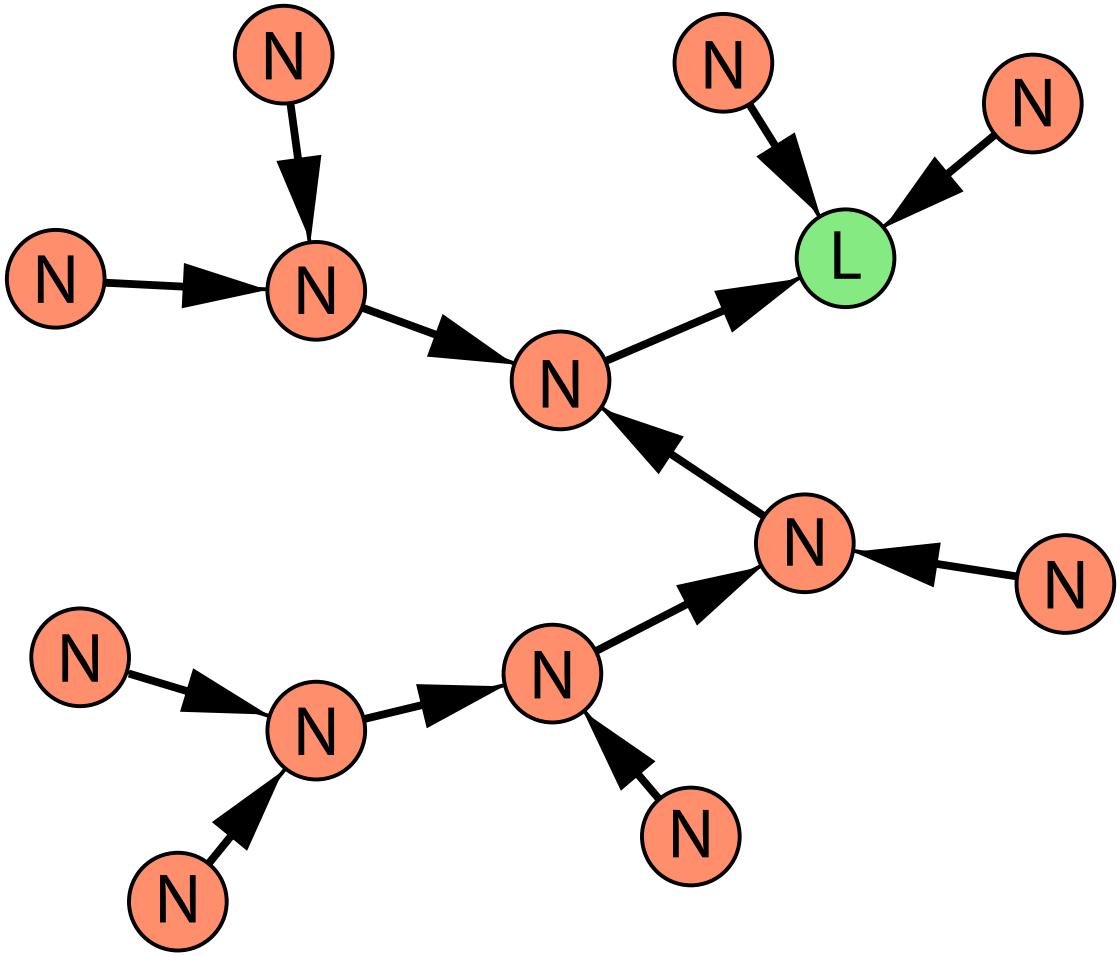
If the scheduler is k-bounded, a leaf eventually knows that it is a leaf. A non-leader leaf can safely terminate.

Leader Election in a Tree



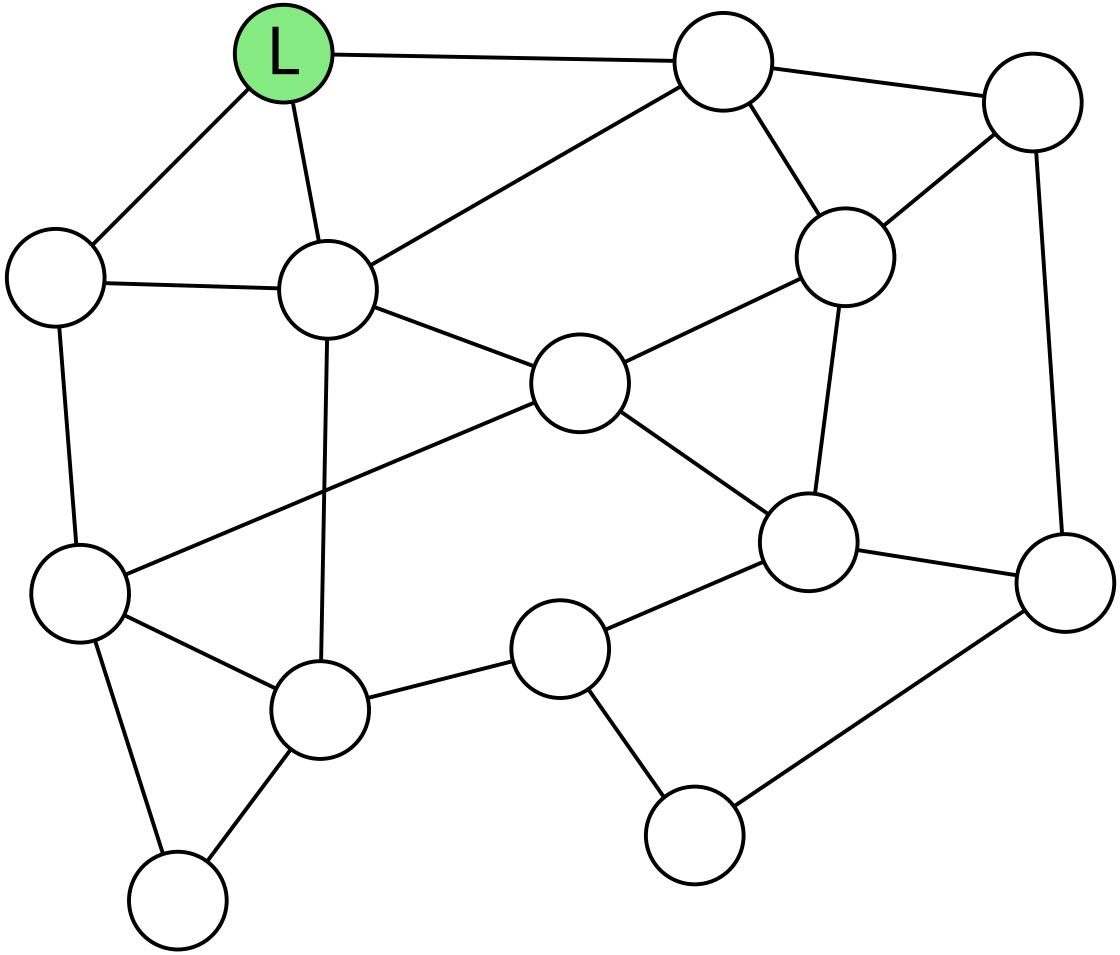
If an agent's children all have terminated, the agent eventually realizes and terminates.

Leader Election in a Tree



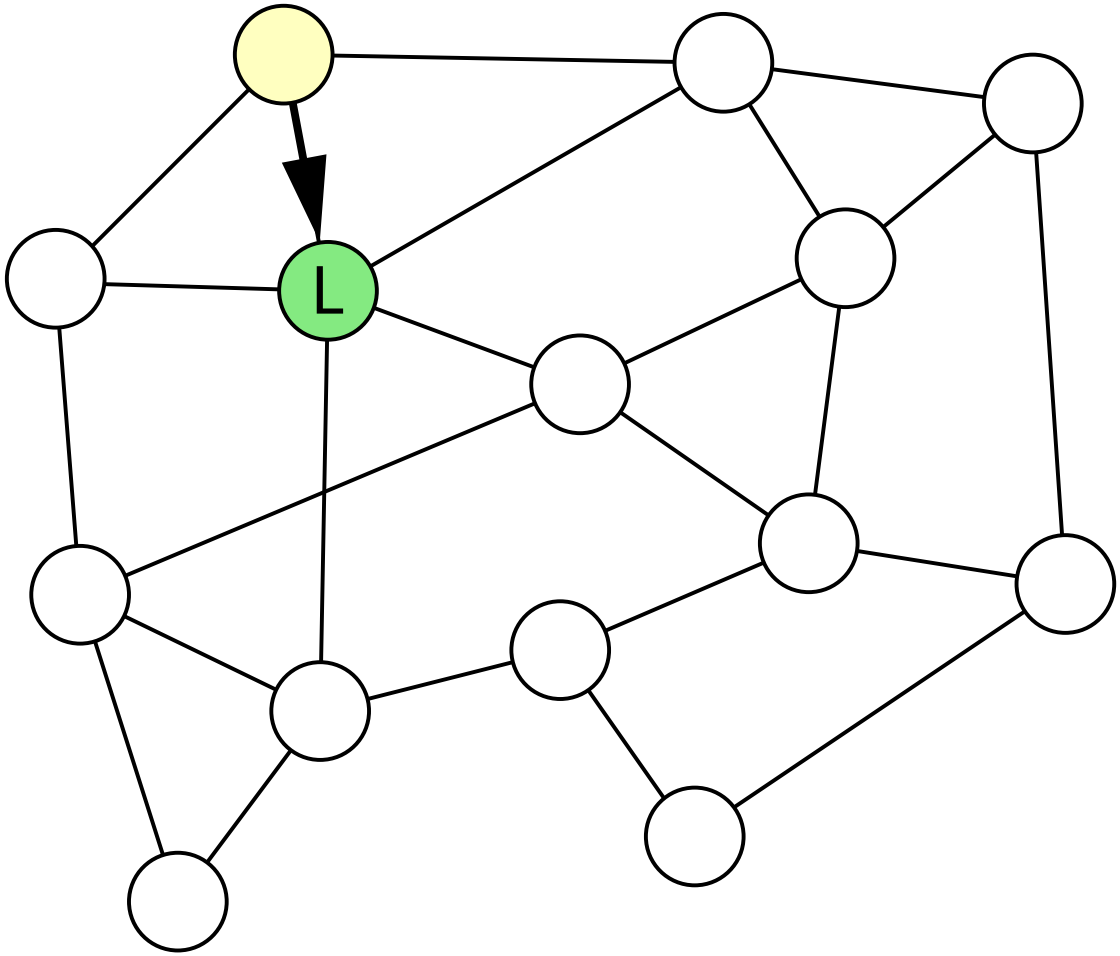
Eventually, all non-leader agents terminate, and hence the protocol is terminating.

Application: Token Circulation



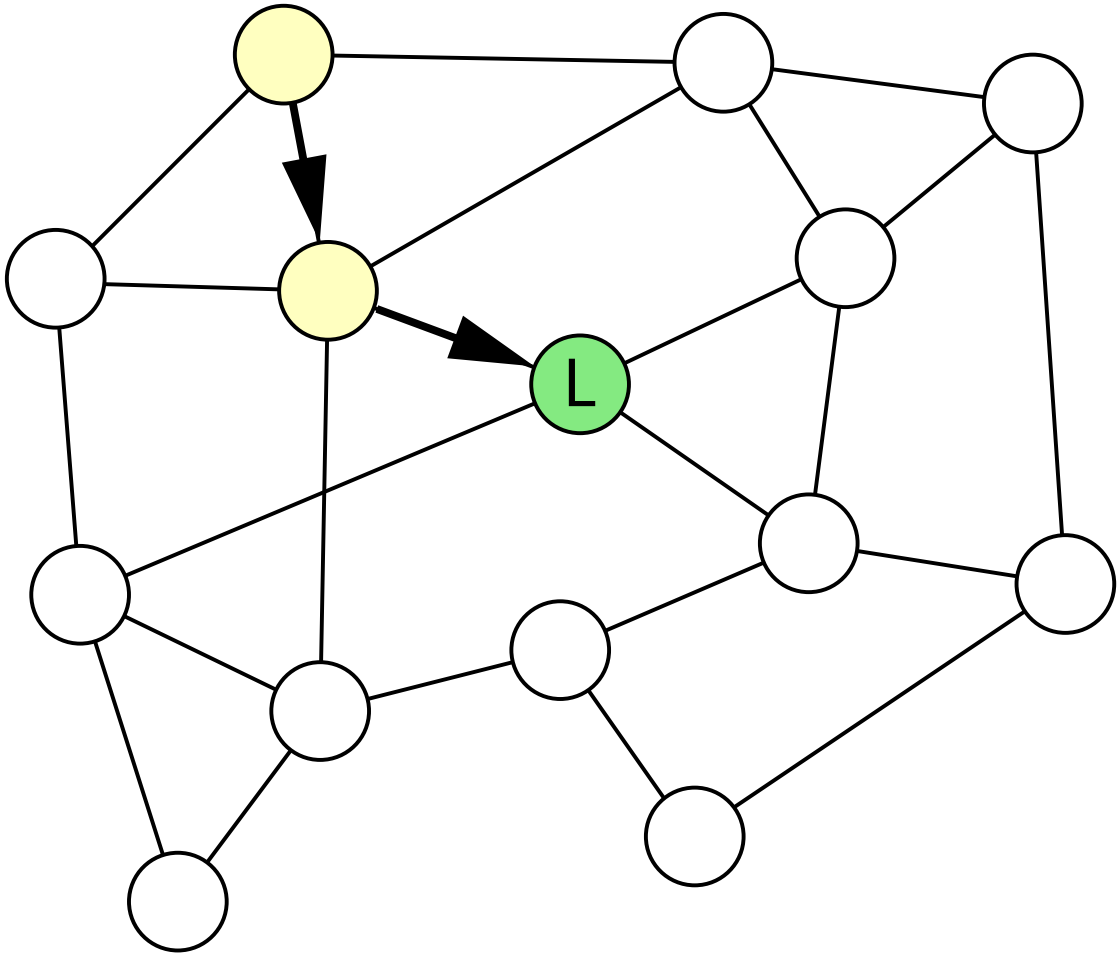
Suppose there is a unique leader and we want to make it "visit" the entire network.

Application: Token Circulation



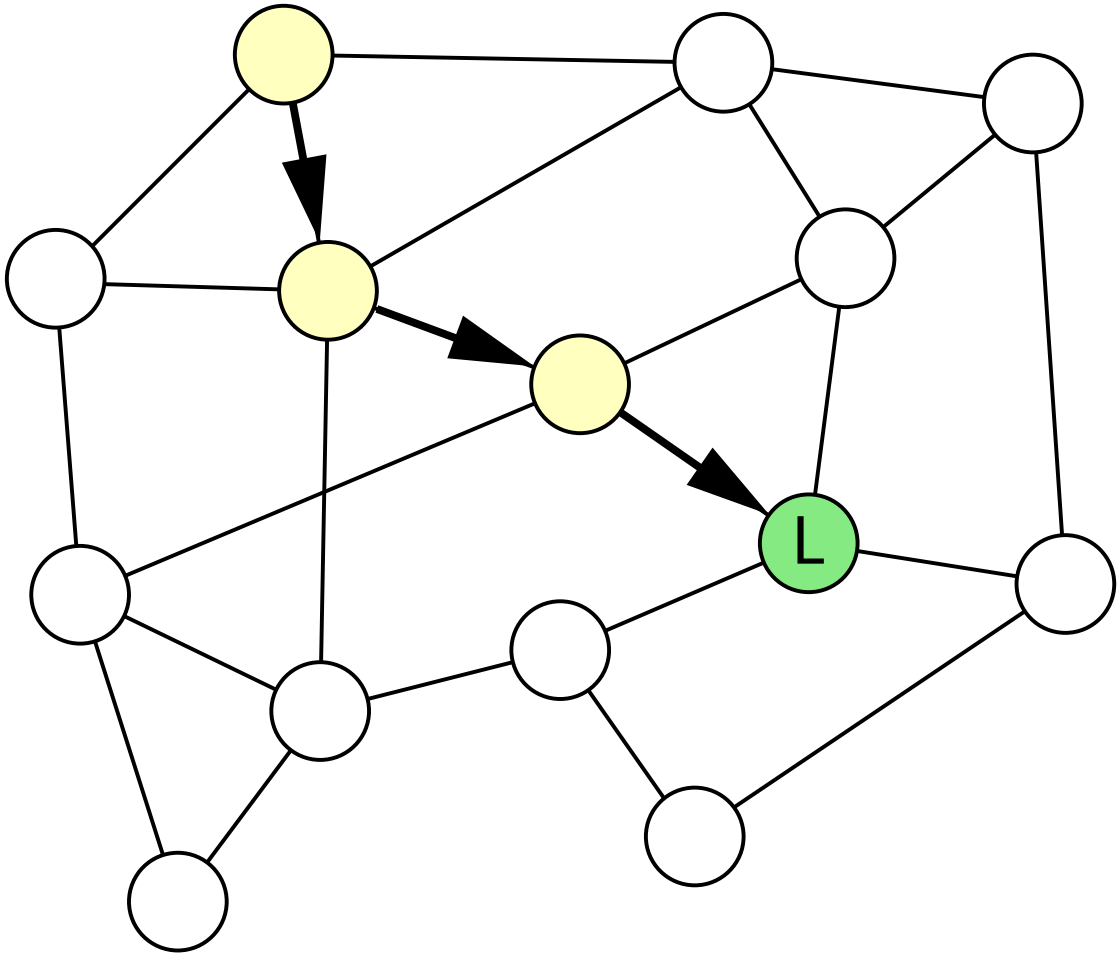
By that we mean that the leadership is “transferred” to a different agent during an interaction.

Application: Token Circulation



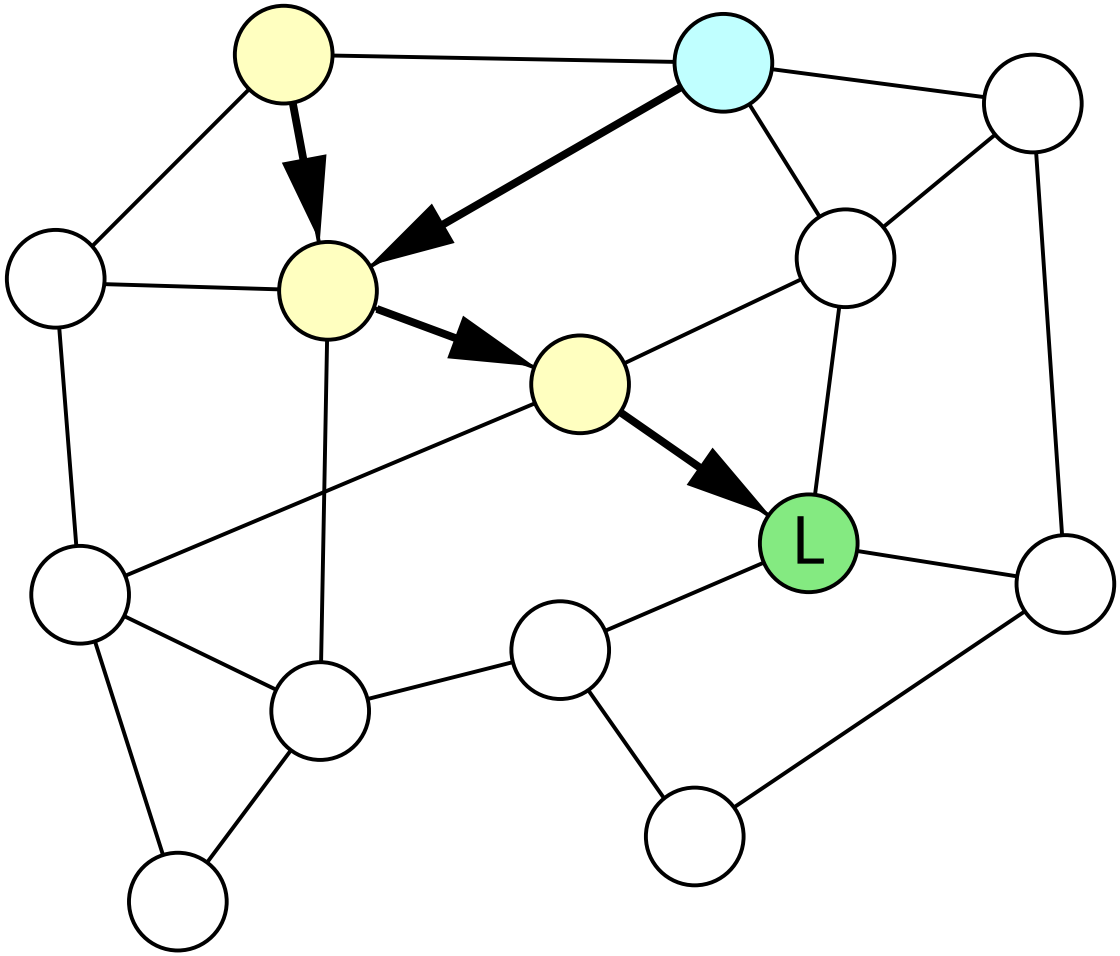
When an agent that has never been leader meets the leader, it takes the leadership.

Application: Token Circulation



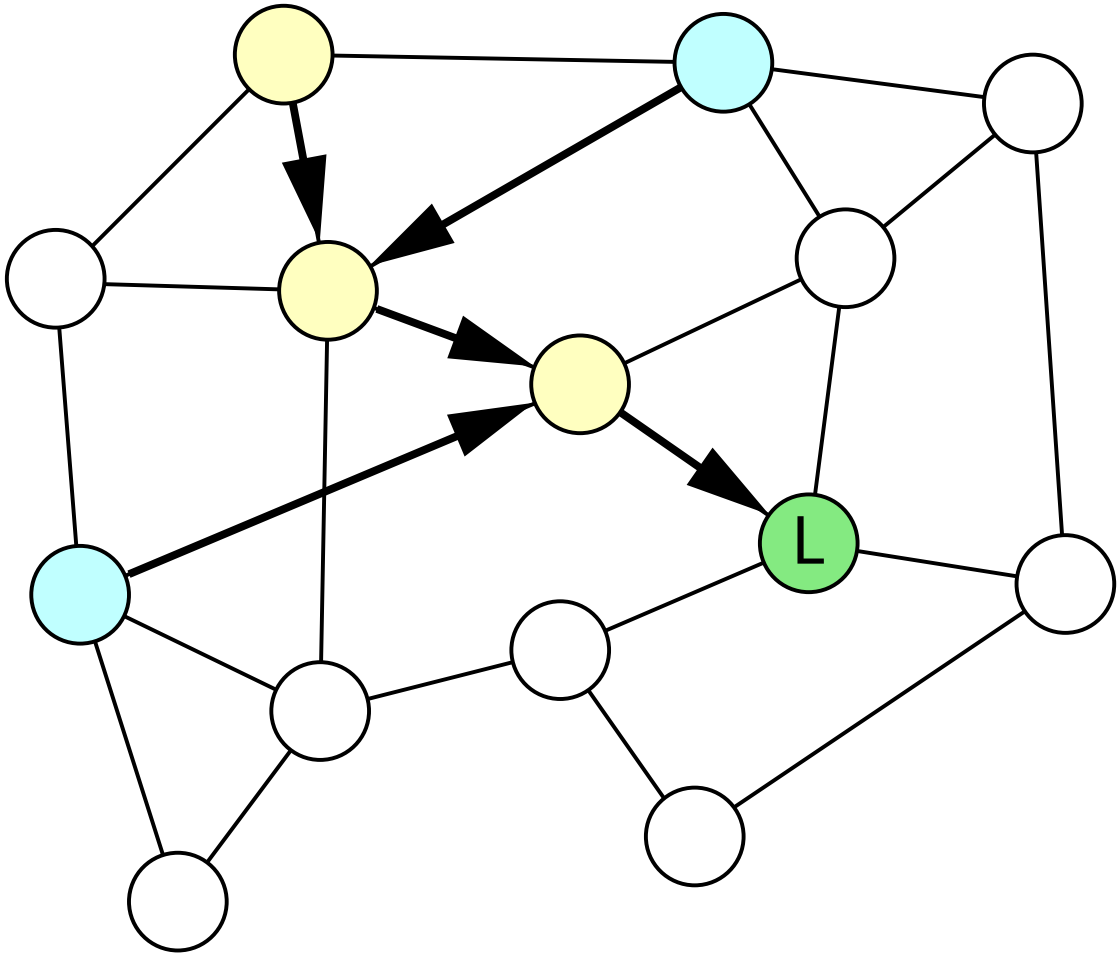
As the leader goes, it leaves a "trail" of arrows.

Application: Token Circulation



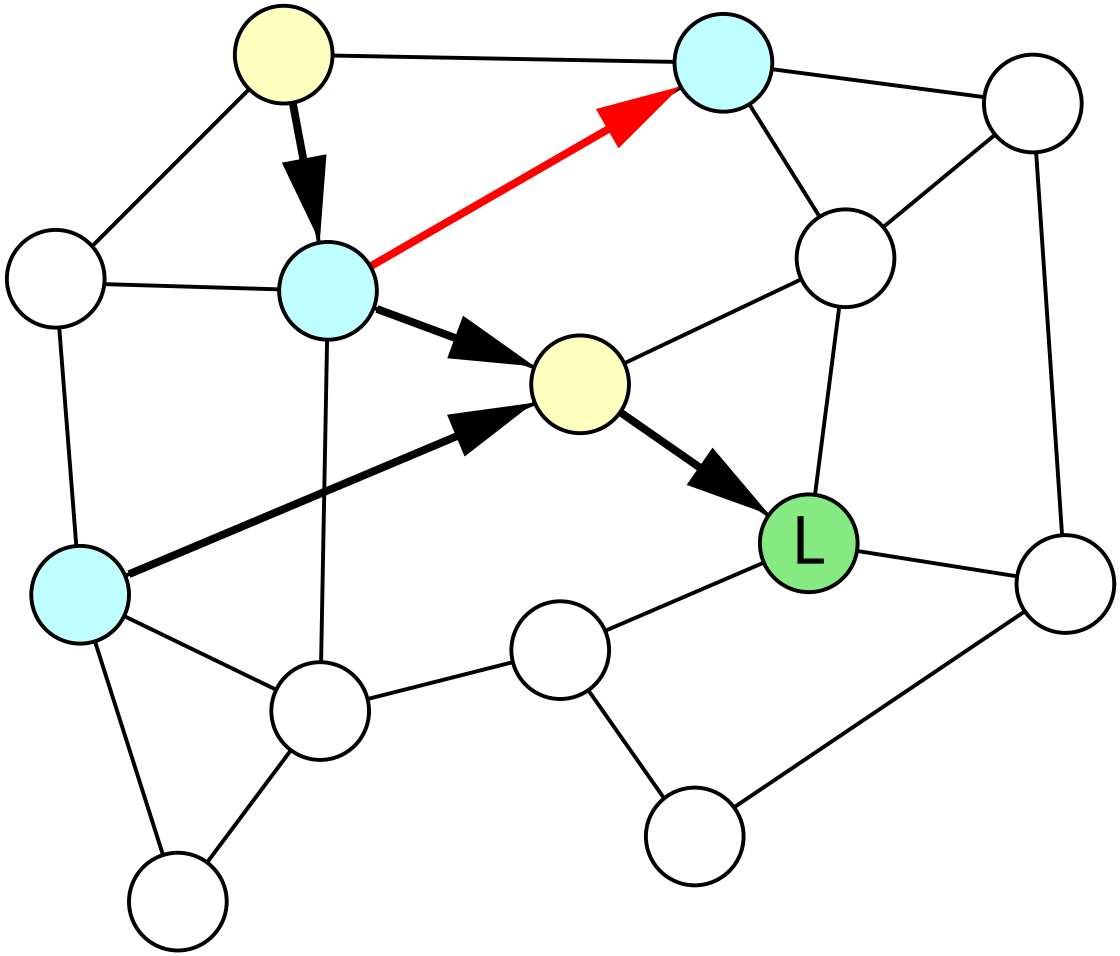
When a new agent meets an agent that has already been leader, it becomes a “summoner”.

Application: Token Circulation



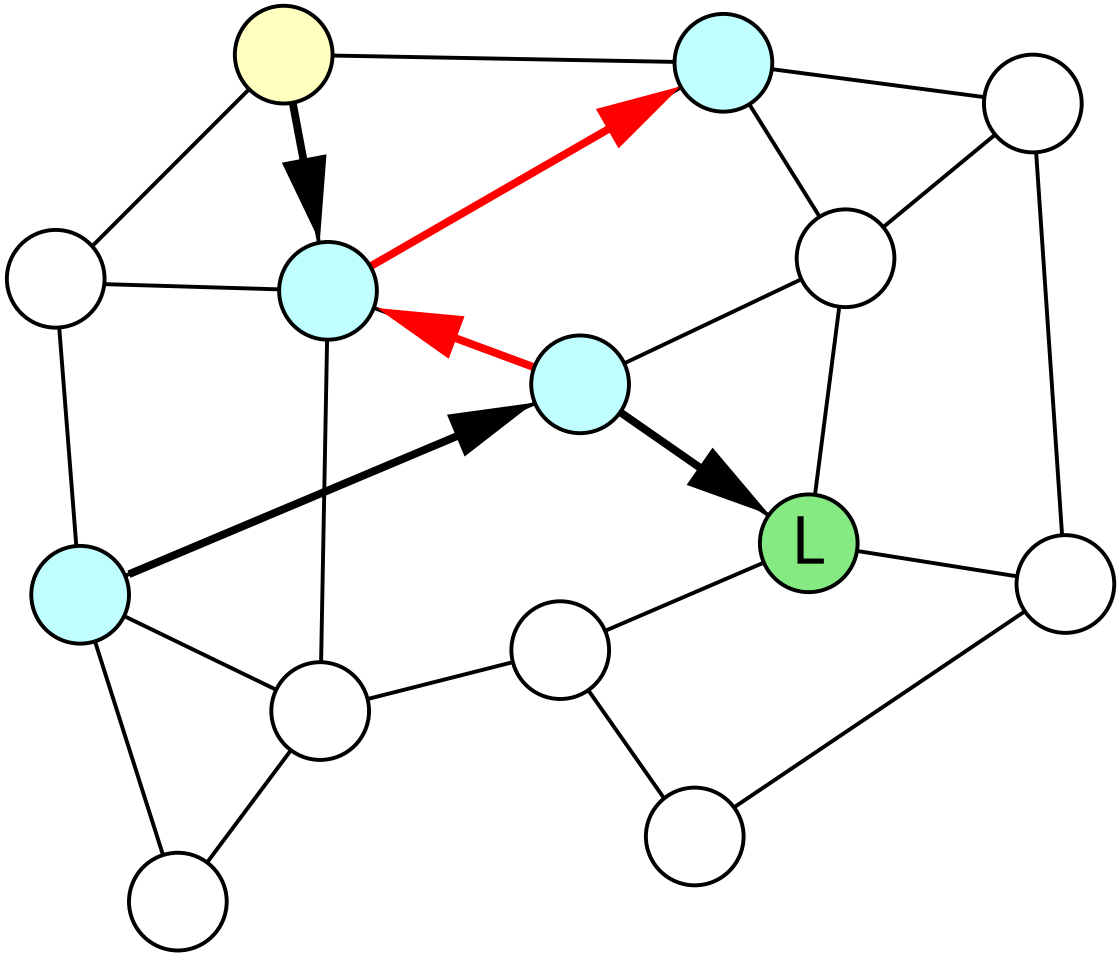
When a new agent meets an agent that has already been leader, it becomes a “summoner”.

Application: Token Circulation



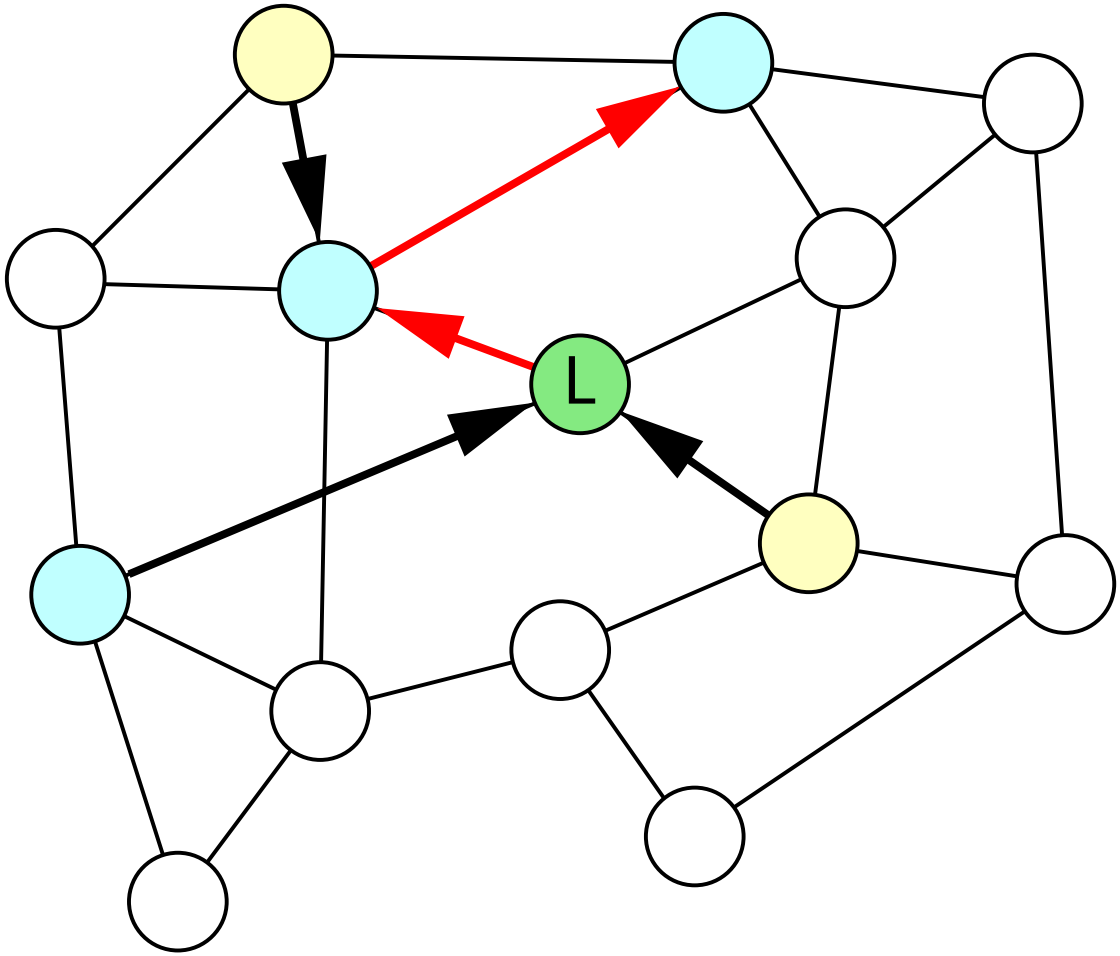
The parent of a summoner becomes a summoner as well, and reverses the corresponding arrow.

Application: Token Circulation



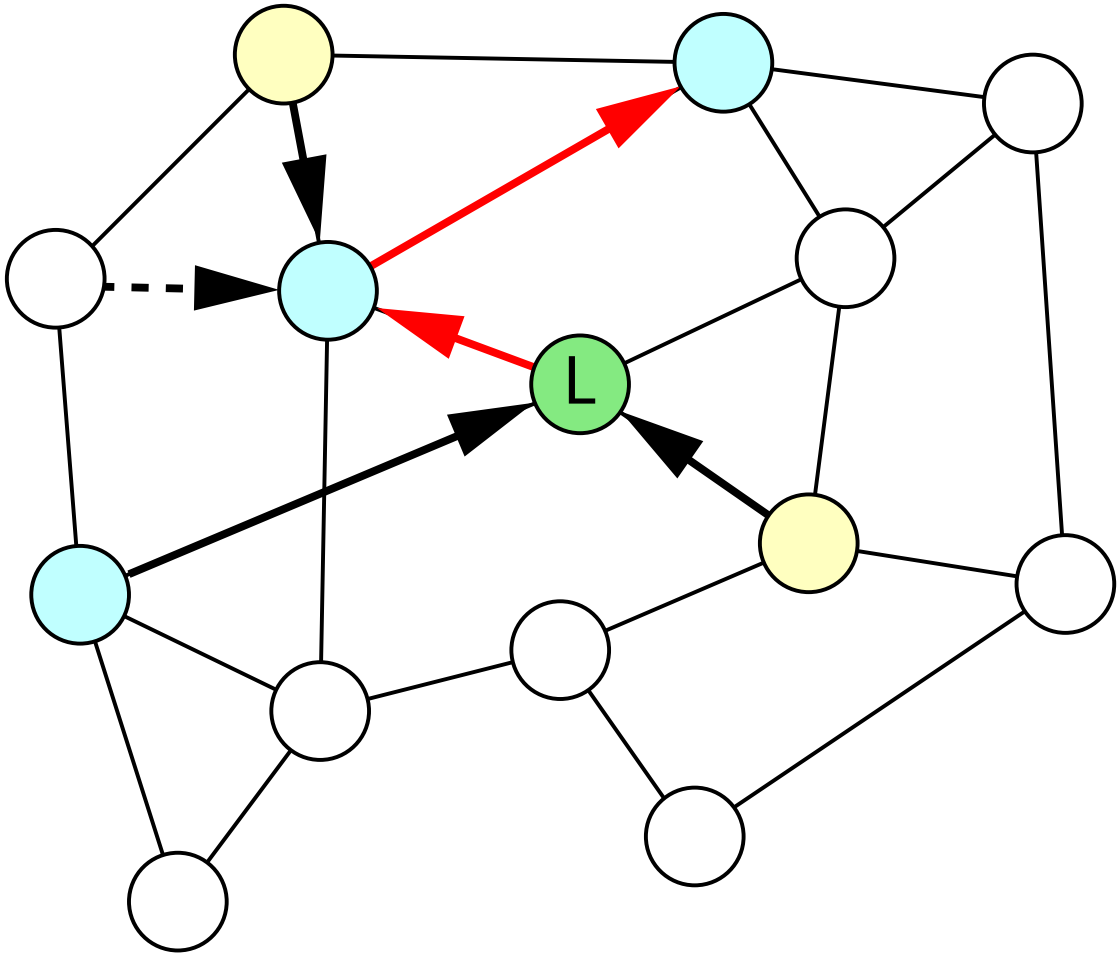
The parent of a summoner becomes a summoner as well, and reverses the corresponding arrow.

Application: Token Circulation



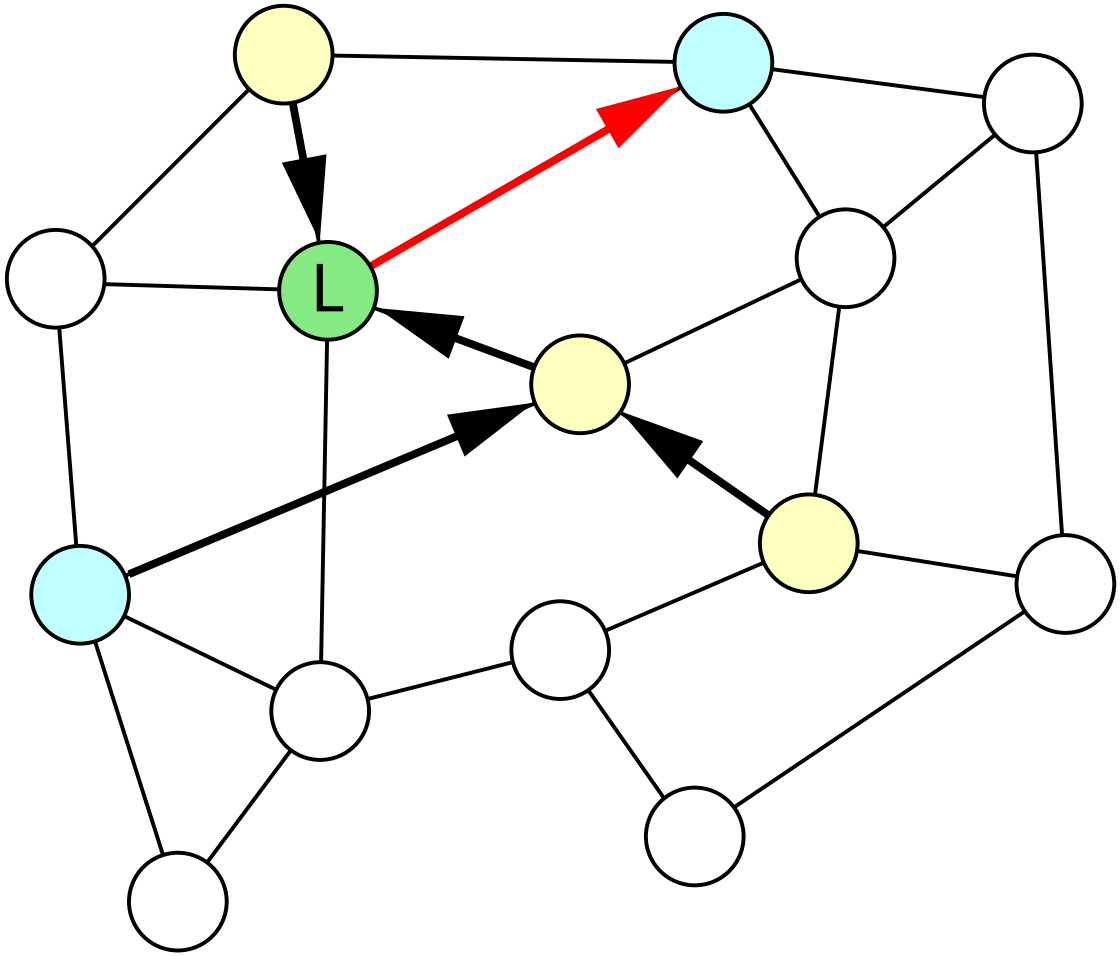
When the leader meets a summoner, it gives it the leadership.

Application: Token Circulation



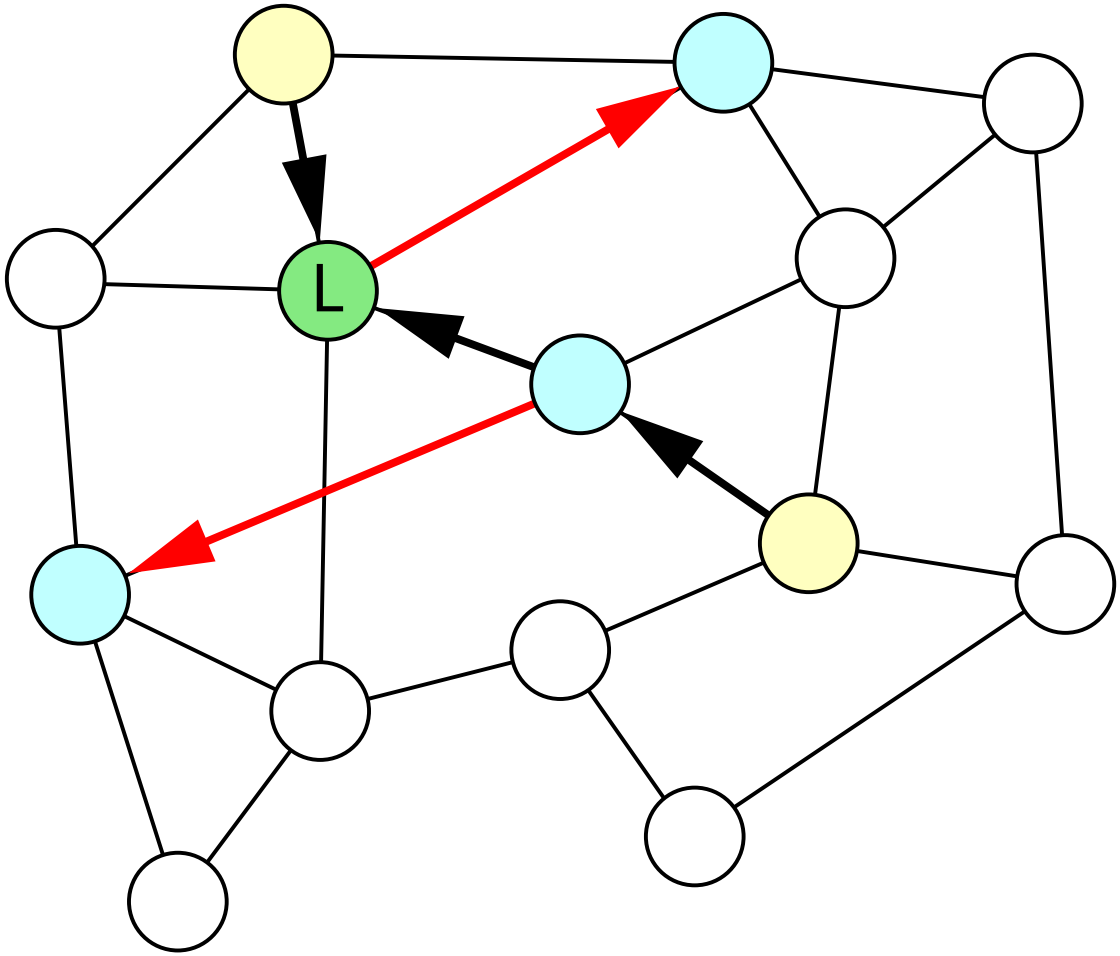
A summoner ignores all requests from new agents.

Application: Token Circulation



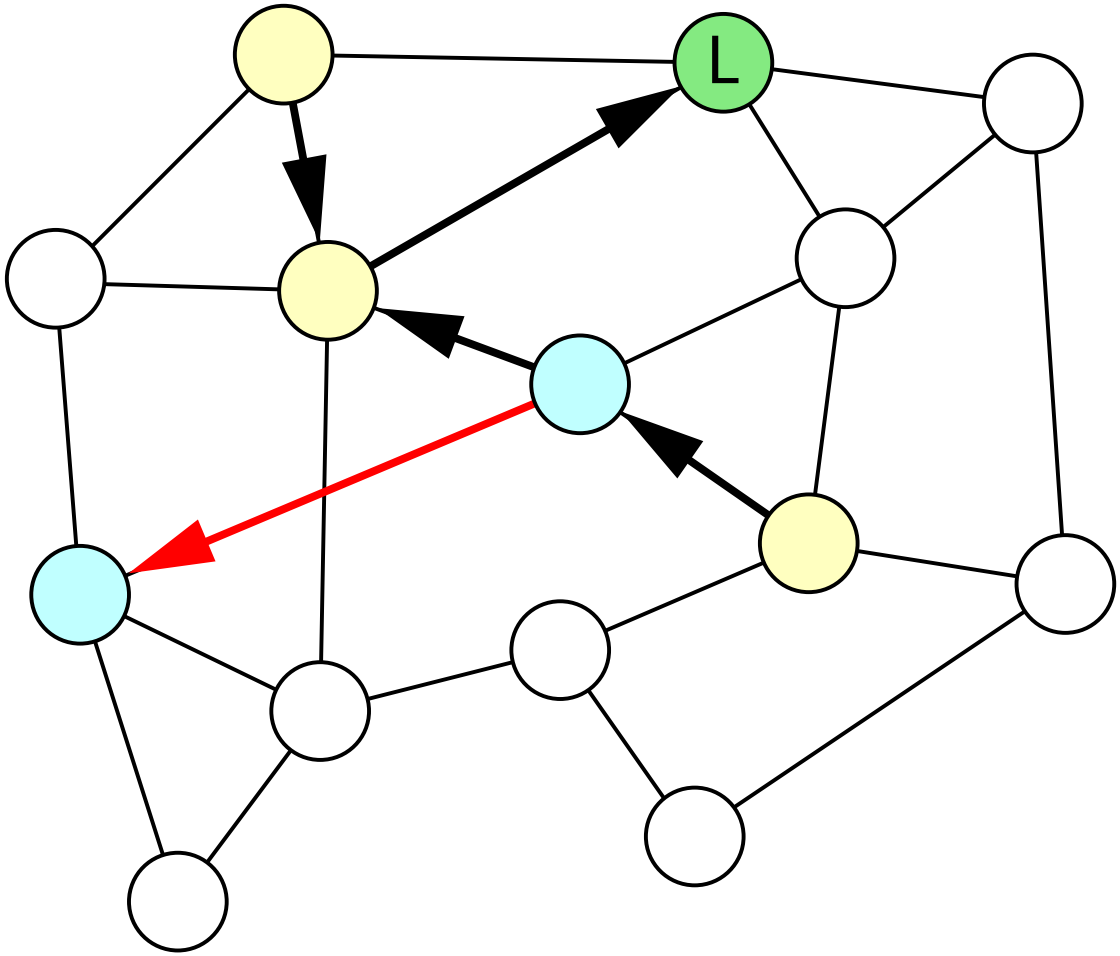
The leader keeps following the arrows through summoners, reversing them as it goes.

Application: Token Circulation



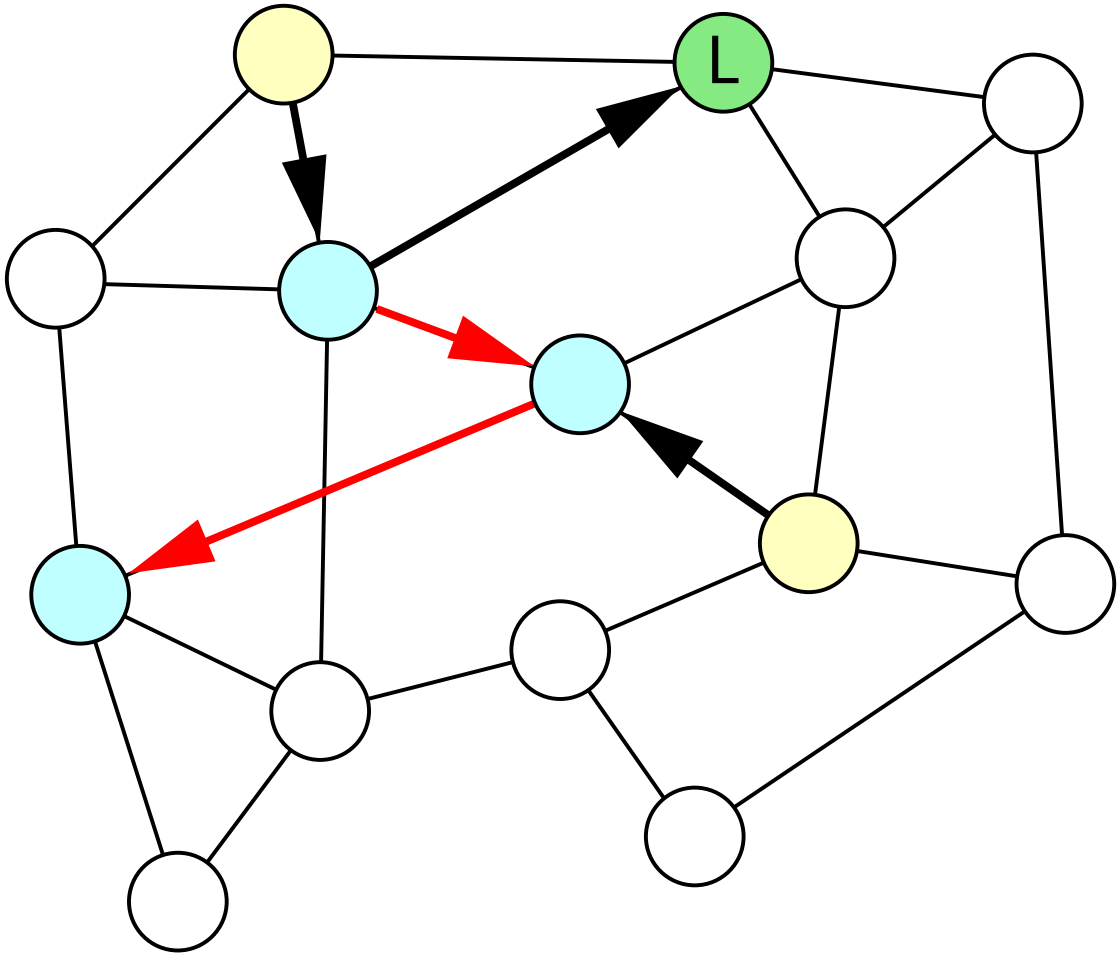
Different agents may summon the leader in parallel, but they never interfere with each other.

Application: Token Circulation



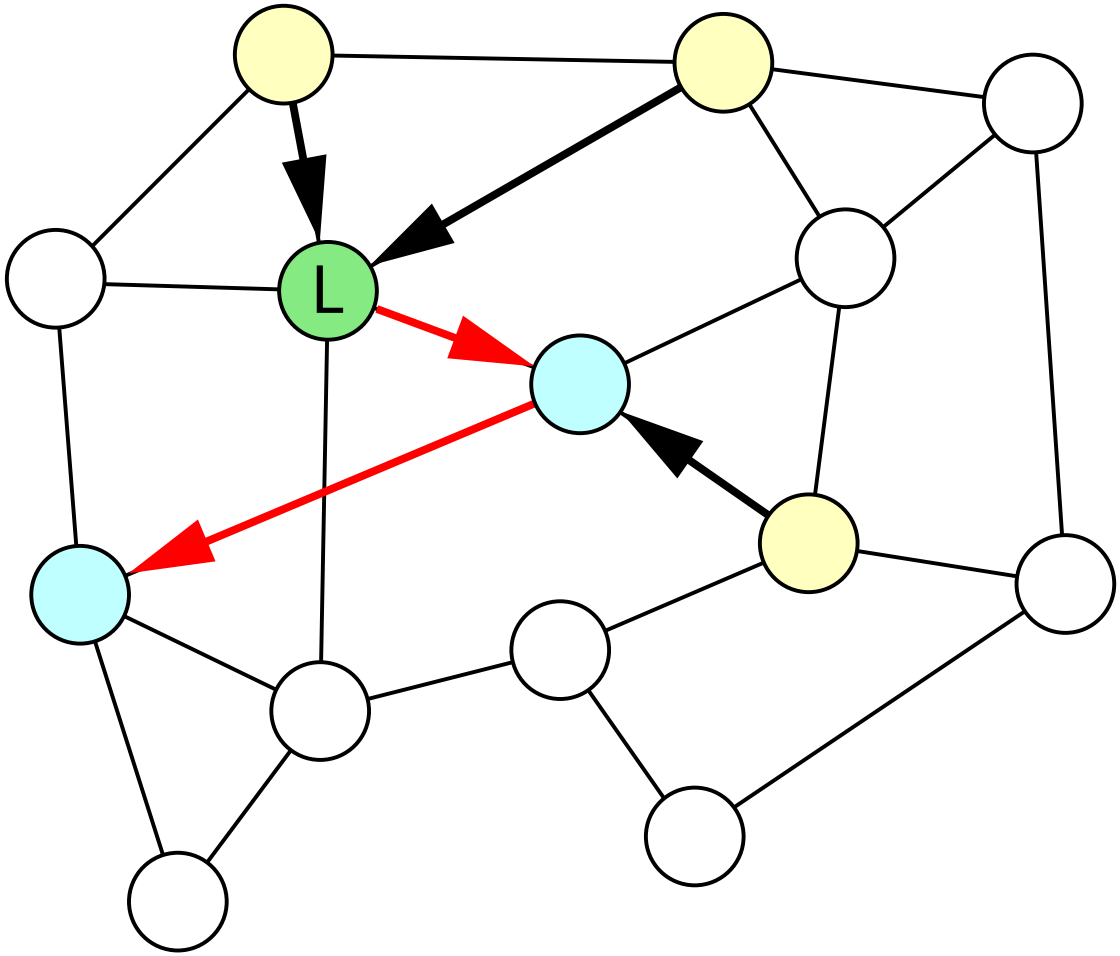
Different agents may summon the leader in parallel, but they never interfere with each other.

Application: Token Circulation



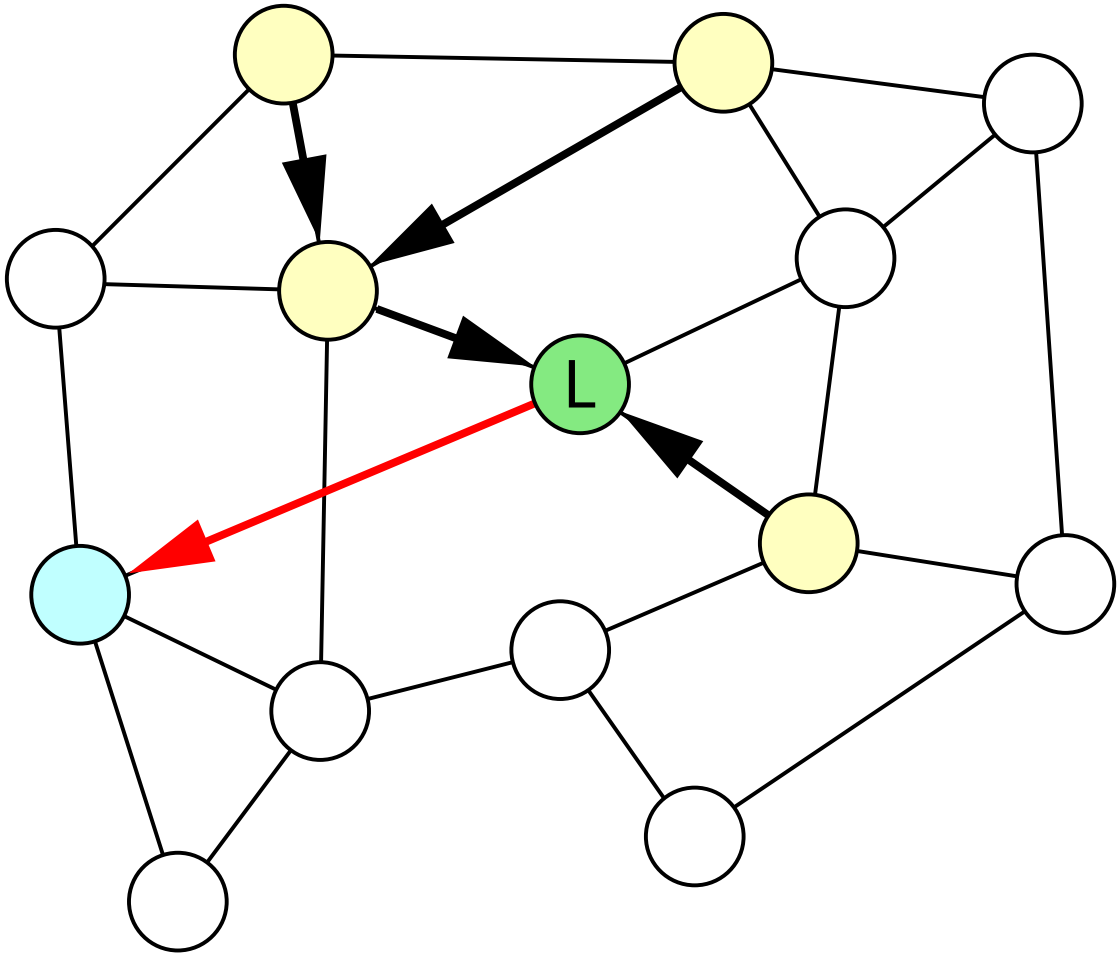
Different agents may summon the leader in parallel, but they never interfere with each other.

Application: Token Circulation



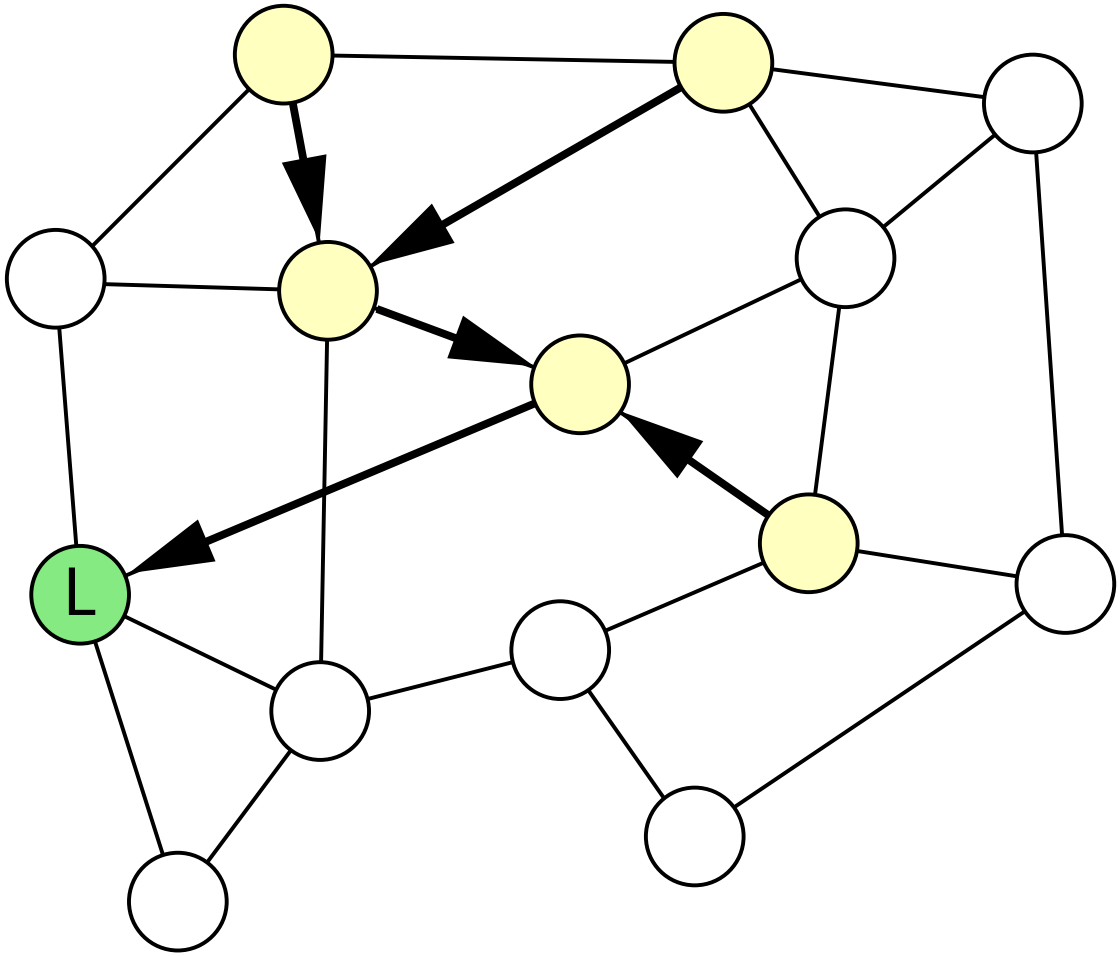
This is because all operations are performed on a subtree of the network.

Application: Token Circulation



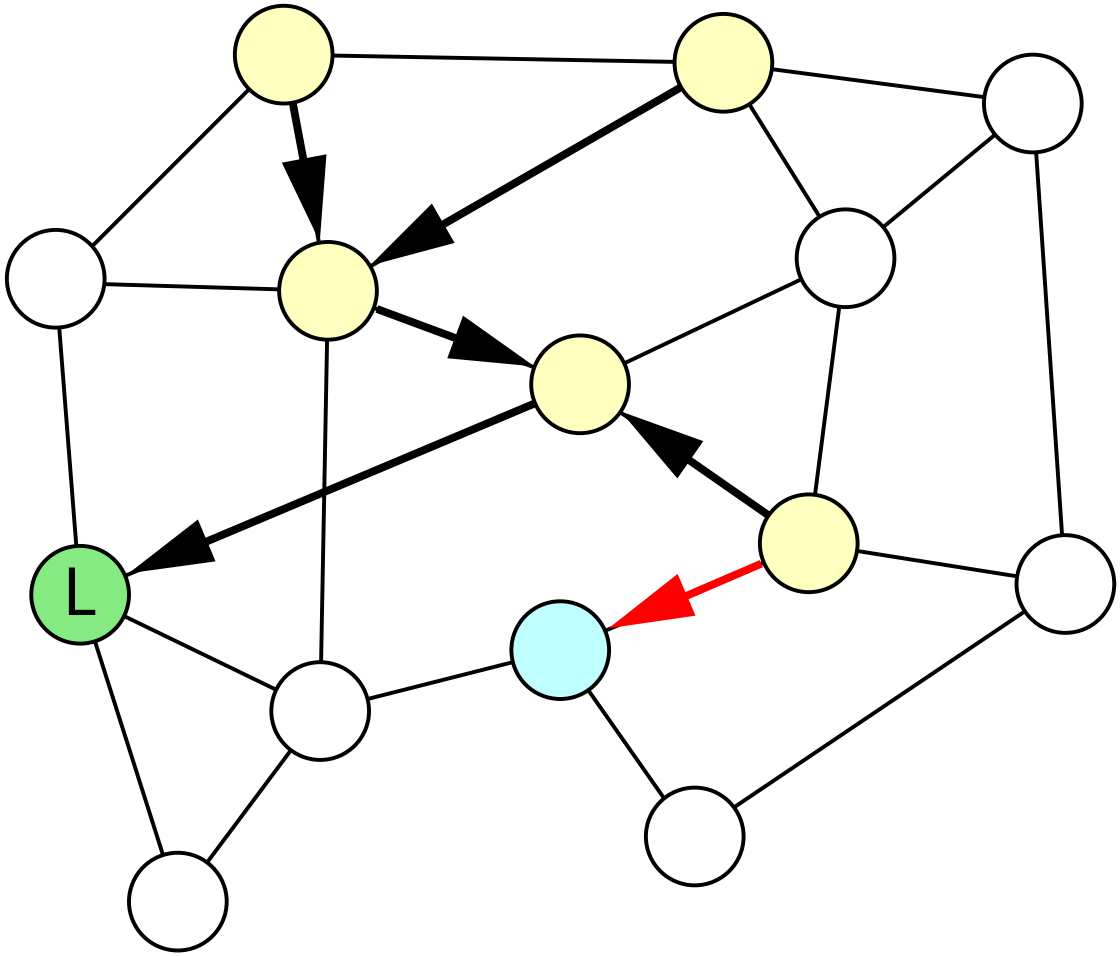
This is because all operations are performed on a subtree of the network.

Application: Token Circulation



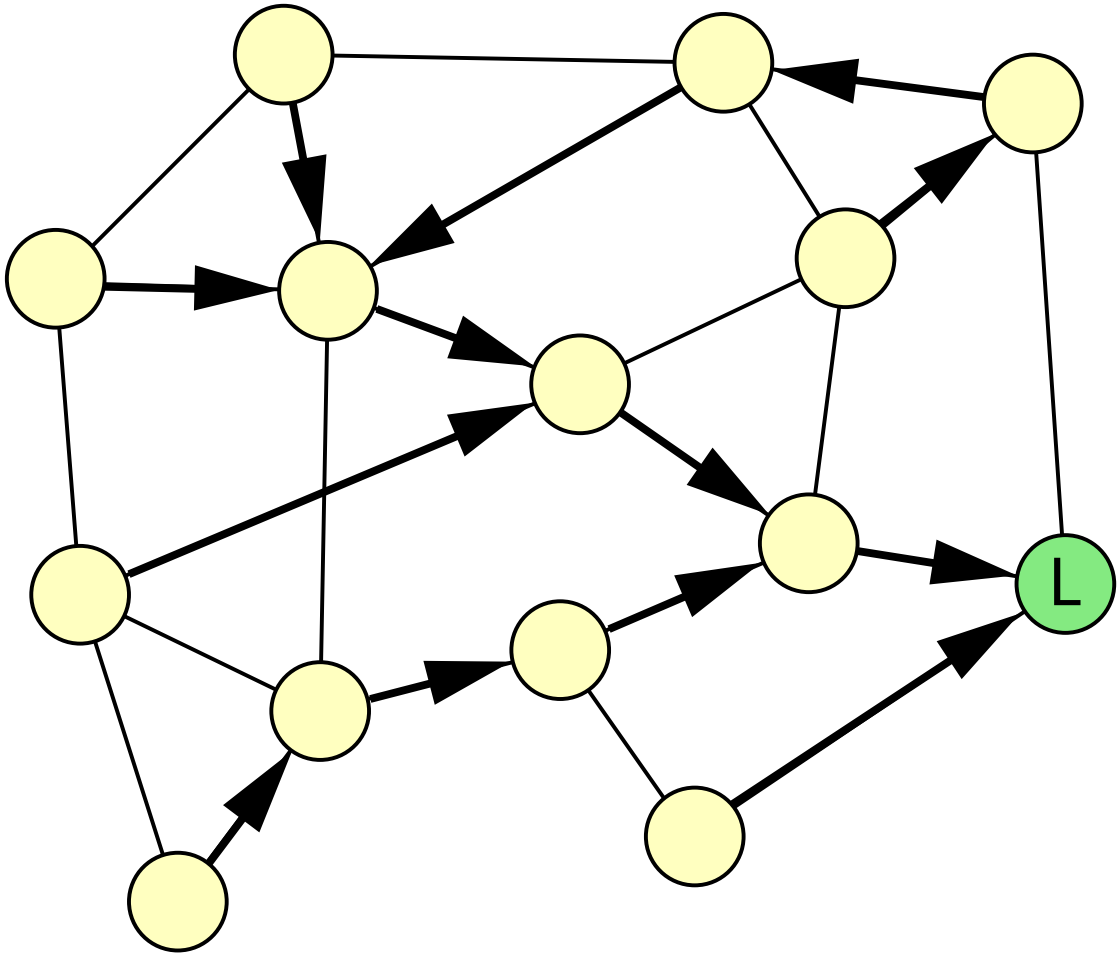
This is because all operations are performed on a subtree of the network.

Application: Token Circulation



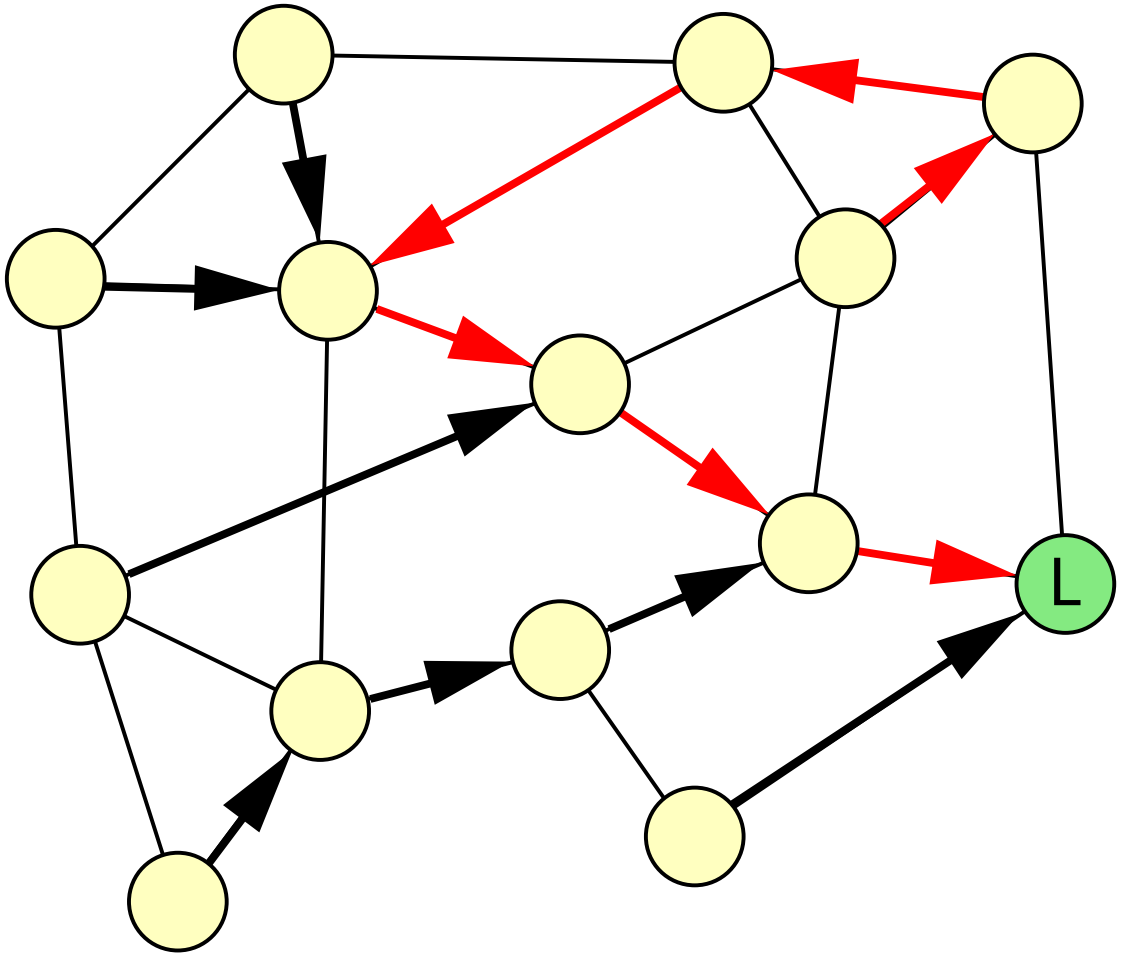
This is because all operations are performed on a subtree of the network.

Application: Token Circulation



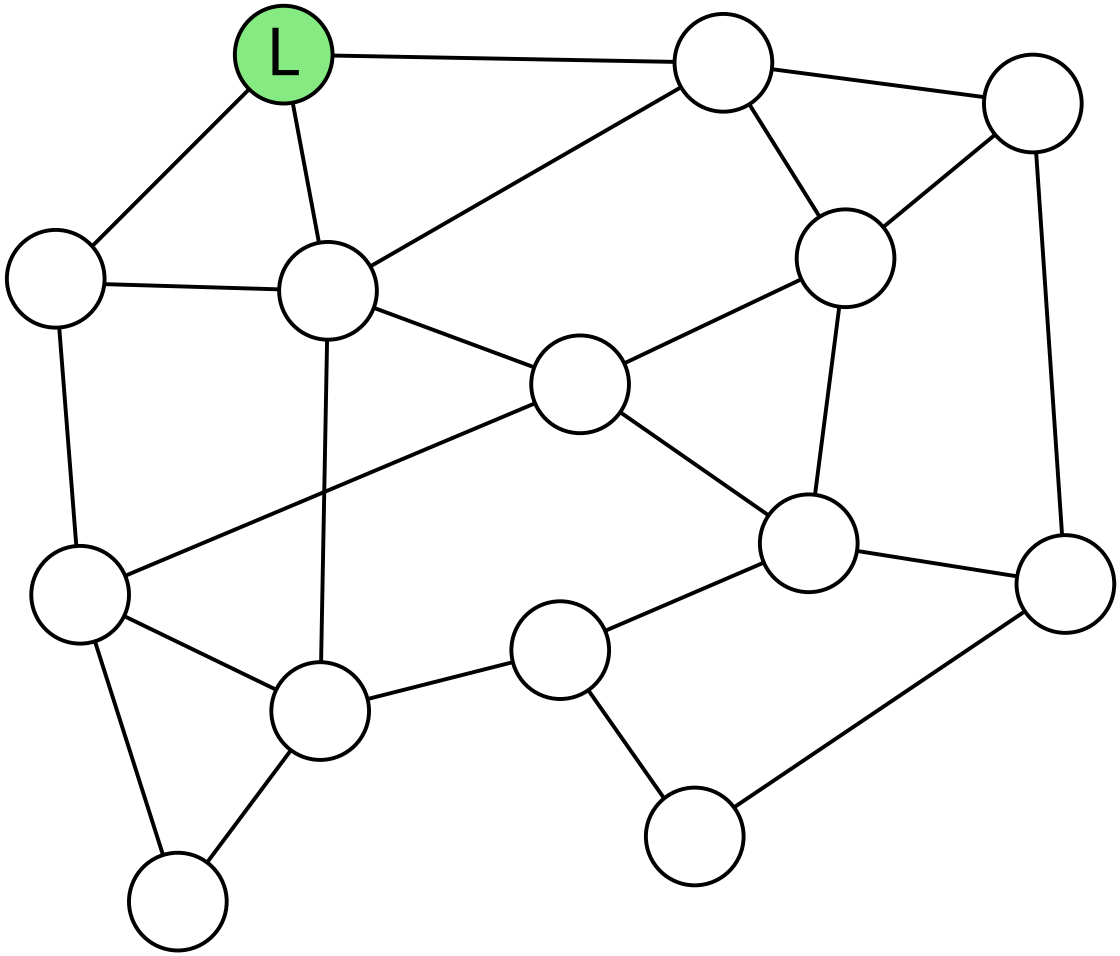
Eventually, the leader visits the entire network. As a byproduct, a rooted spanning tree has been constructed.

Application: Token Circulation



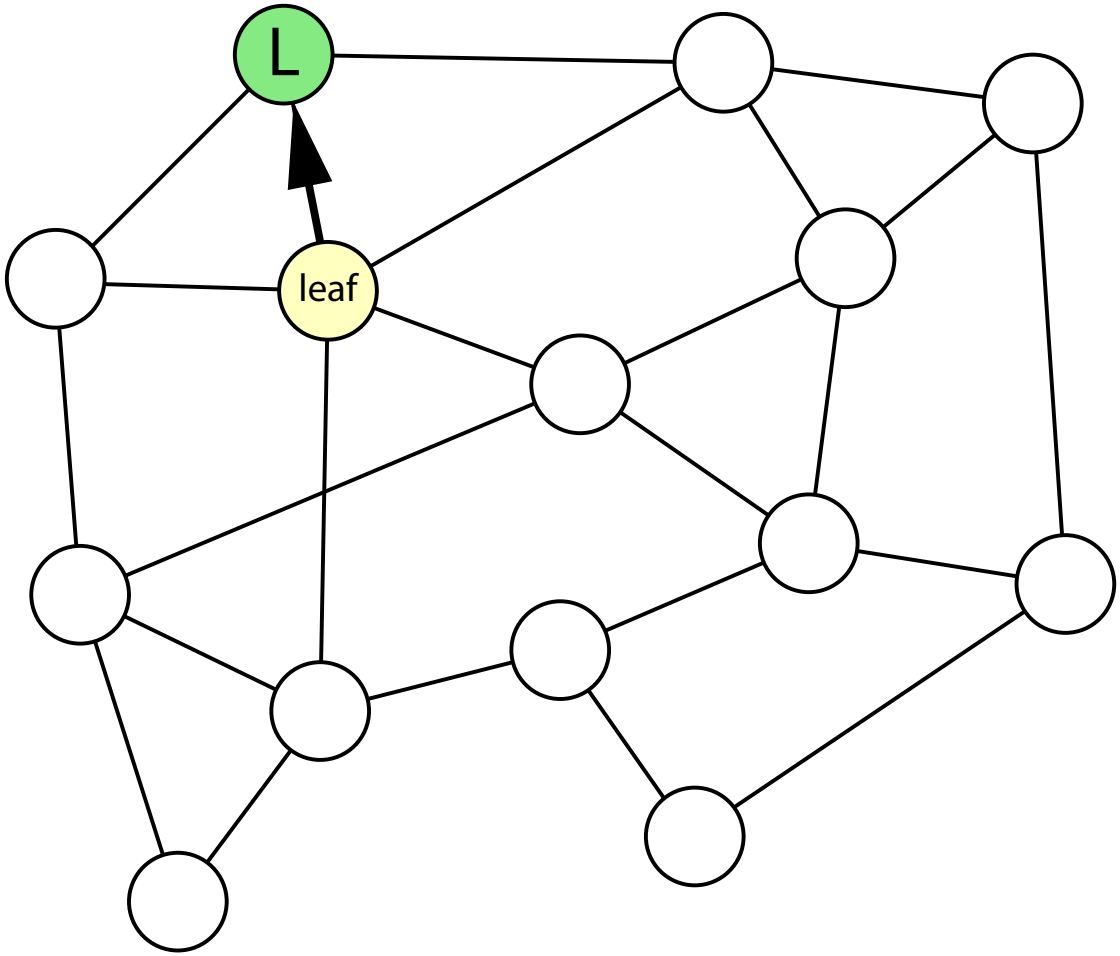
Note that this spanning tree may not be balanced.

Application: Shortest-Path Spanning Tree Construction



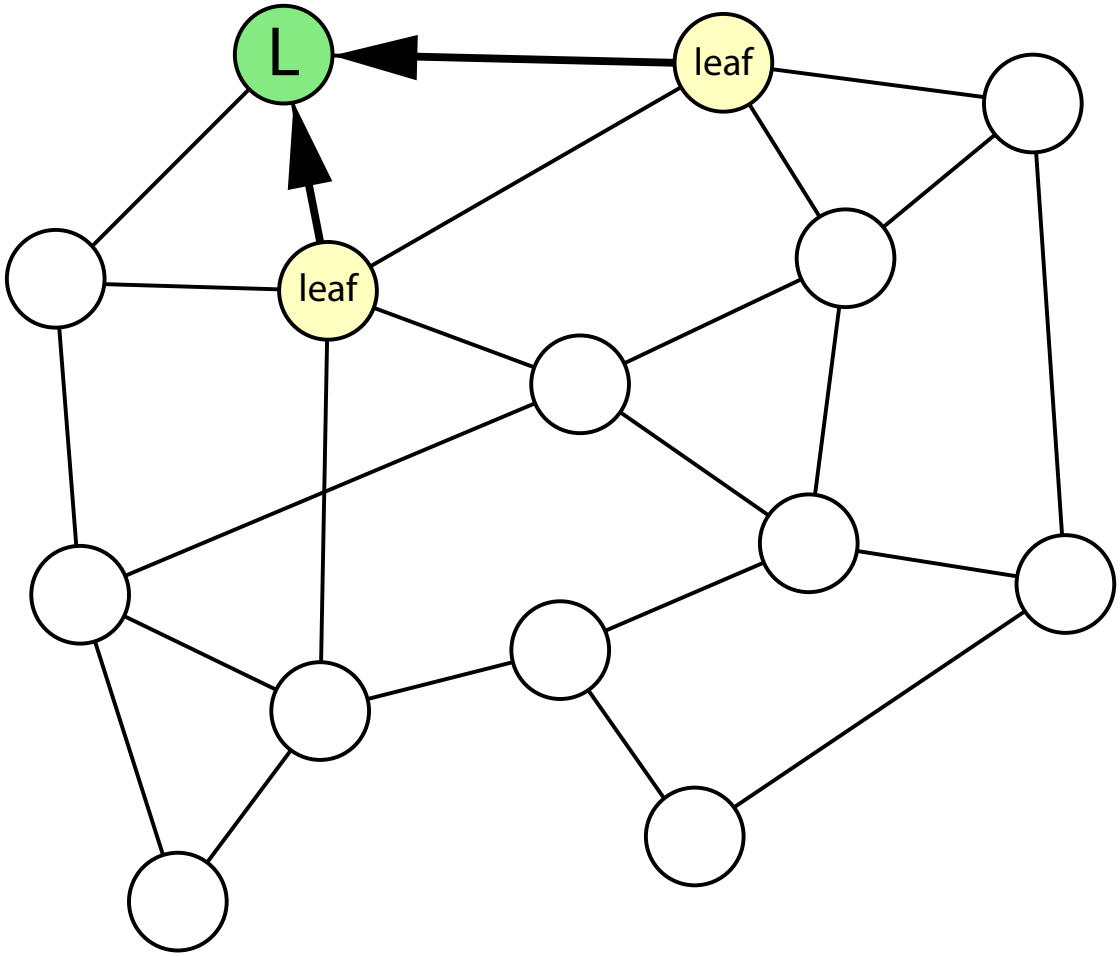
Say we want to construct a better spanning tree rooted at the leader, under the k-bounded scheduler.

Application: Shortest-Path Spanning Tree Construction



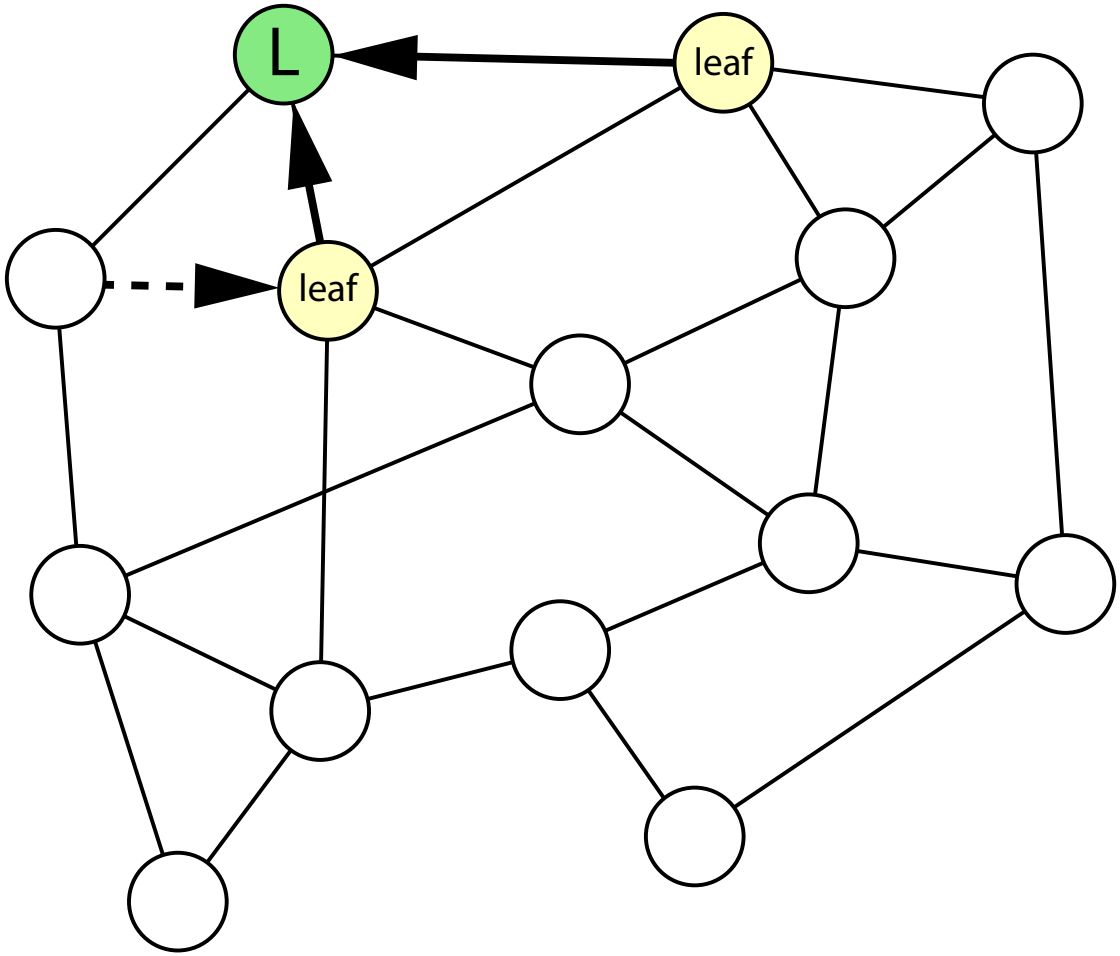
When a new agent interacts with the leader, it becomes a “leaf”.

Application: Shortest-Path Spanning Tree Construction



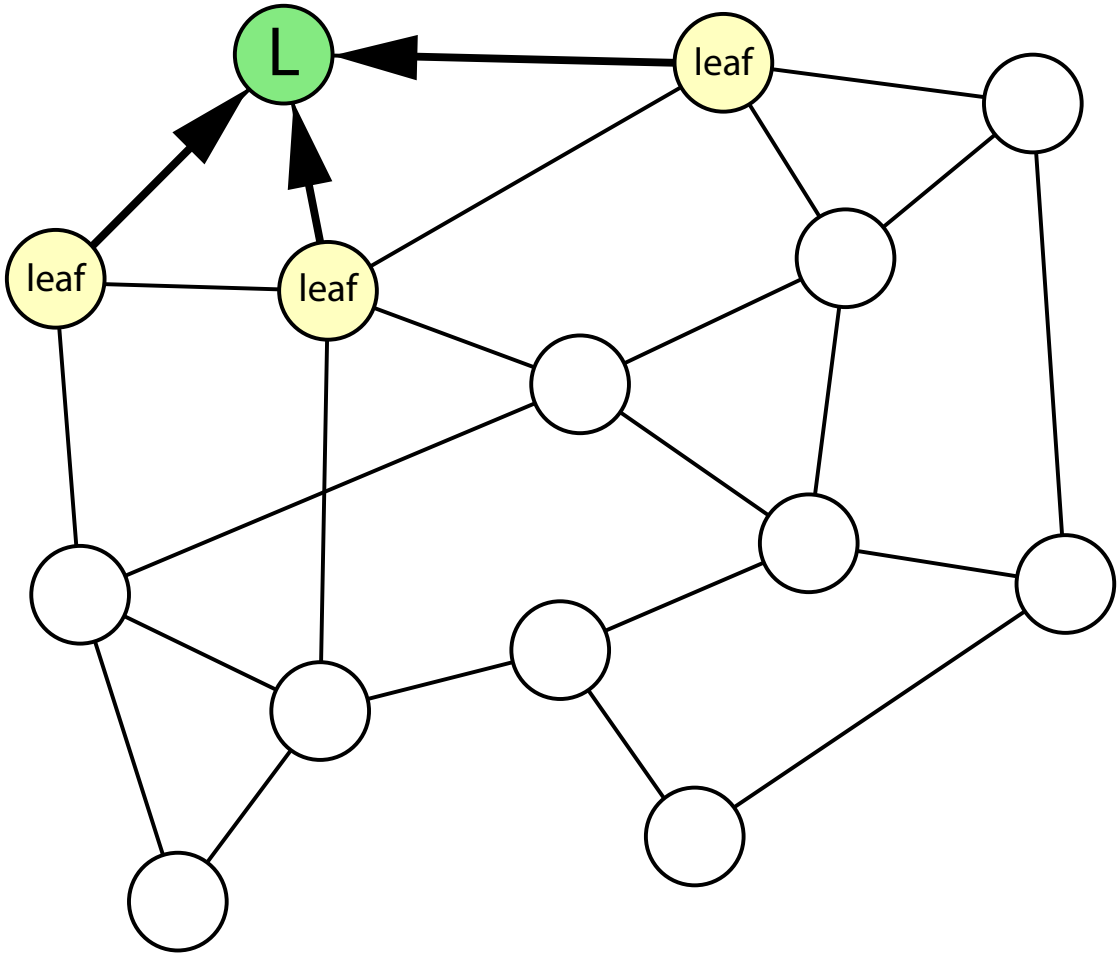
When a new agent interacts with the leader, it becomes a “leaf”.

Application: Shortest-Path Spanning Tree Construction



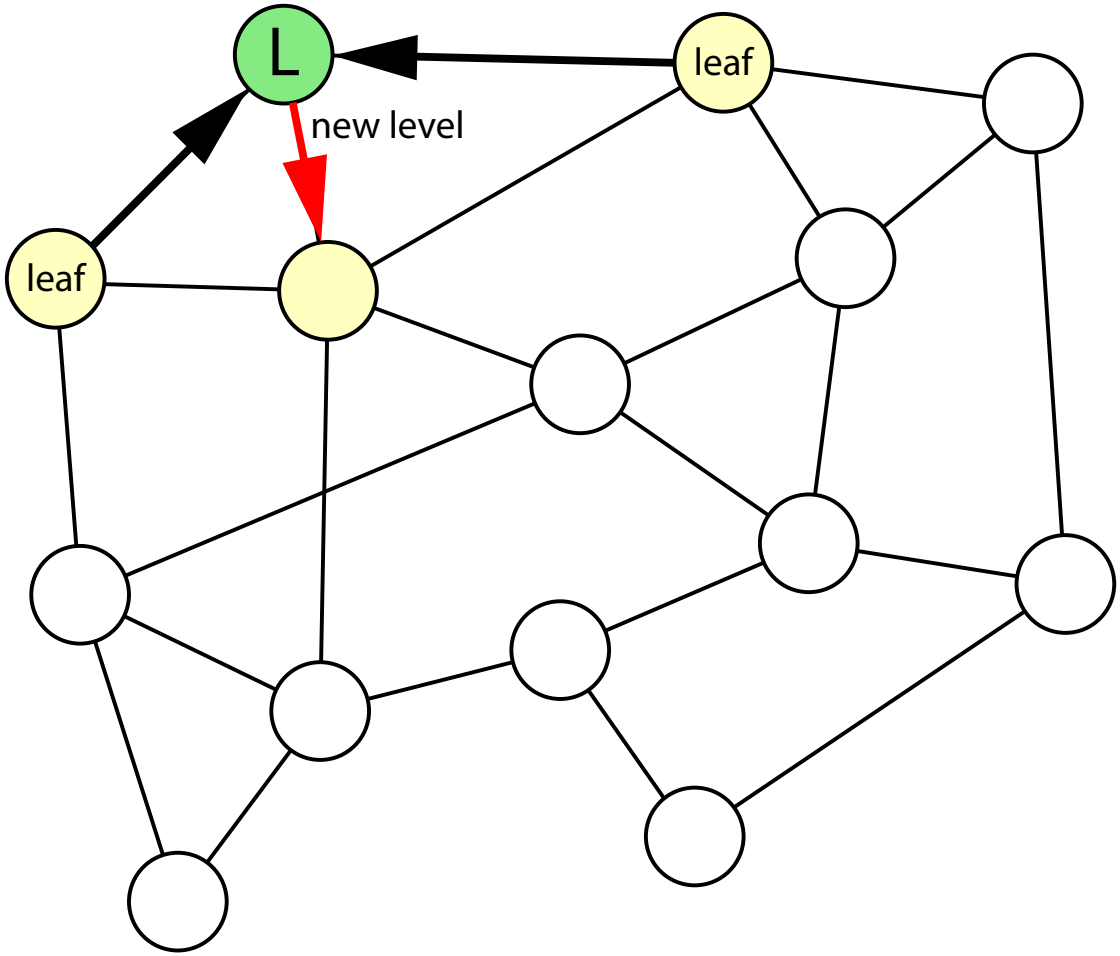
When a new agent interacts with the leader, it becomes a “leaf”.

Application: Shortest-Path Spanning Tree Construction



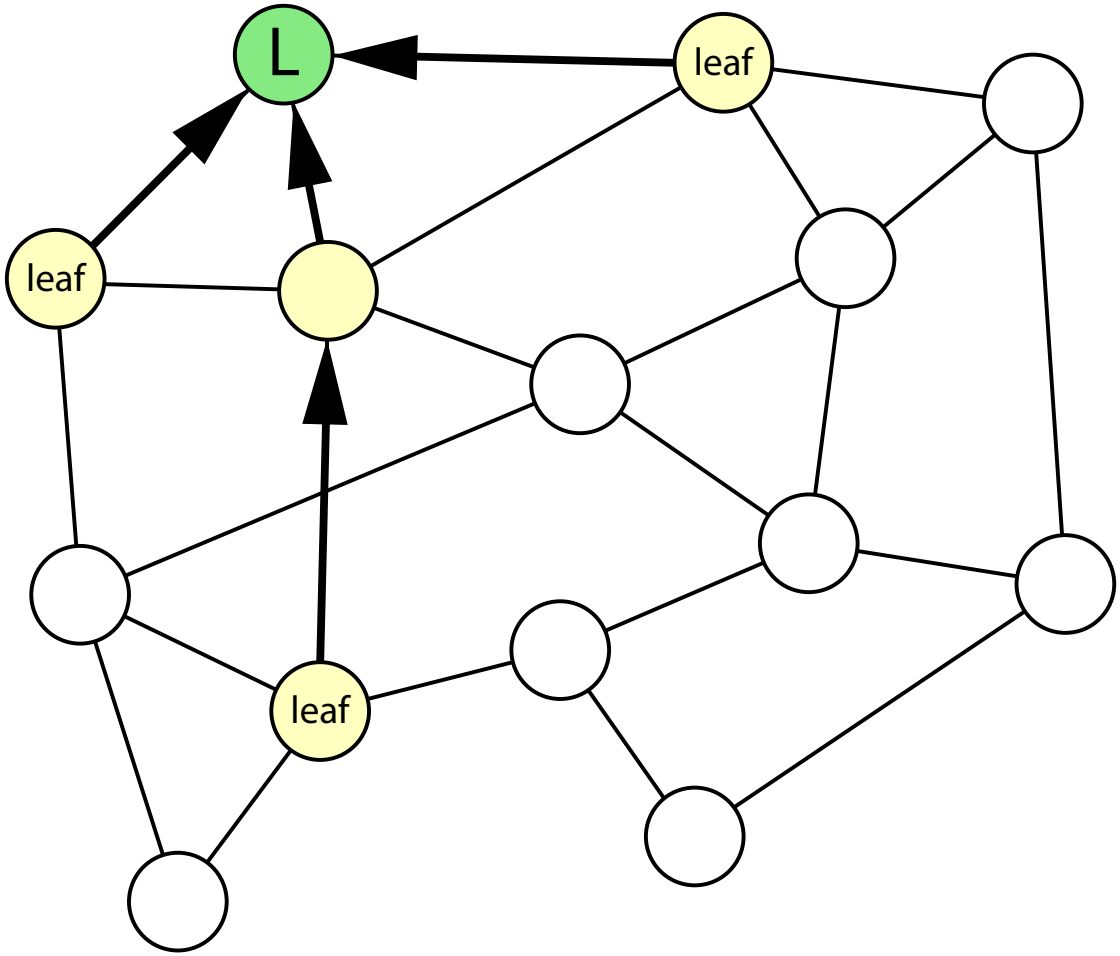
Since the scheduler is k-bounded, the leader knows when all its neighbors are leaves.

Application: Shortest-Path Spanning Tree Construction



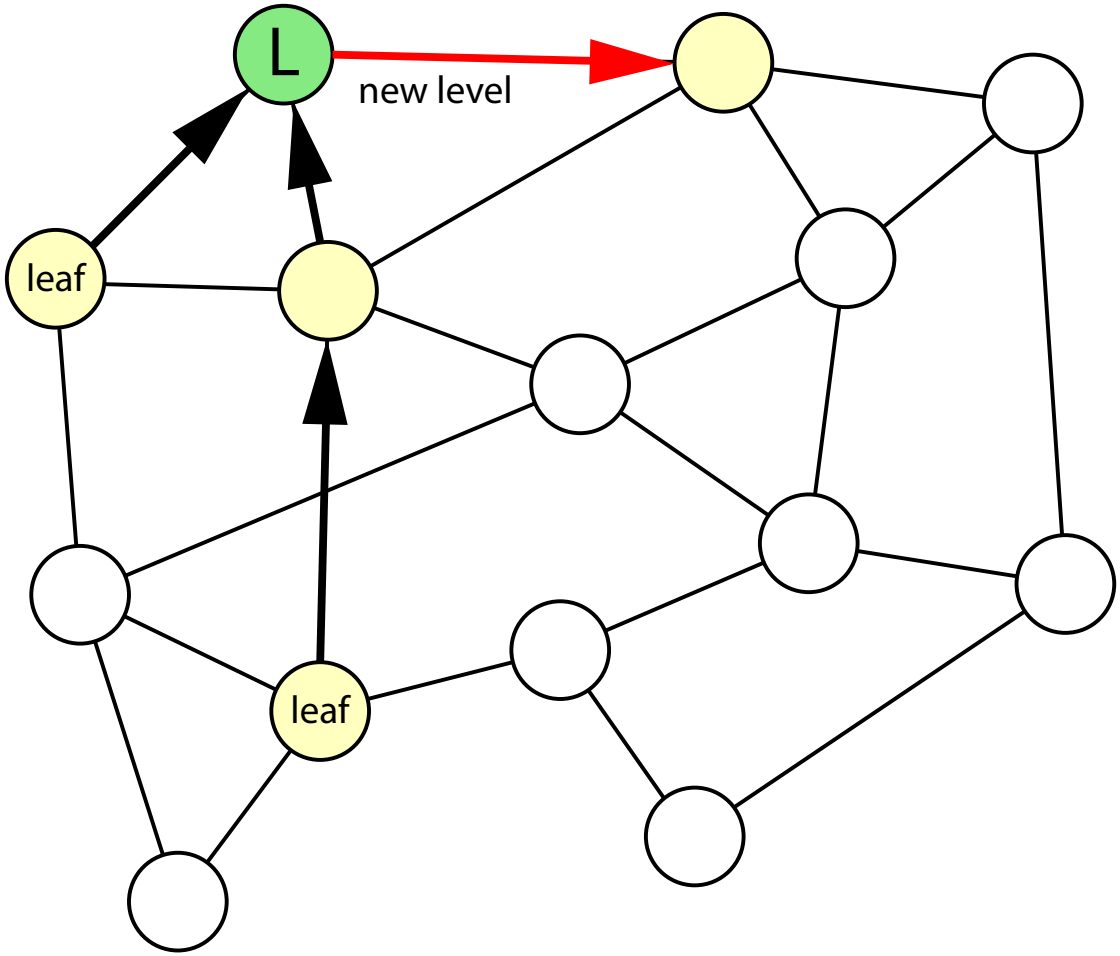
The leader issues a "new level" command along the tree.

Application: Shortest-Path Spanning Tree Construction



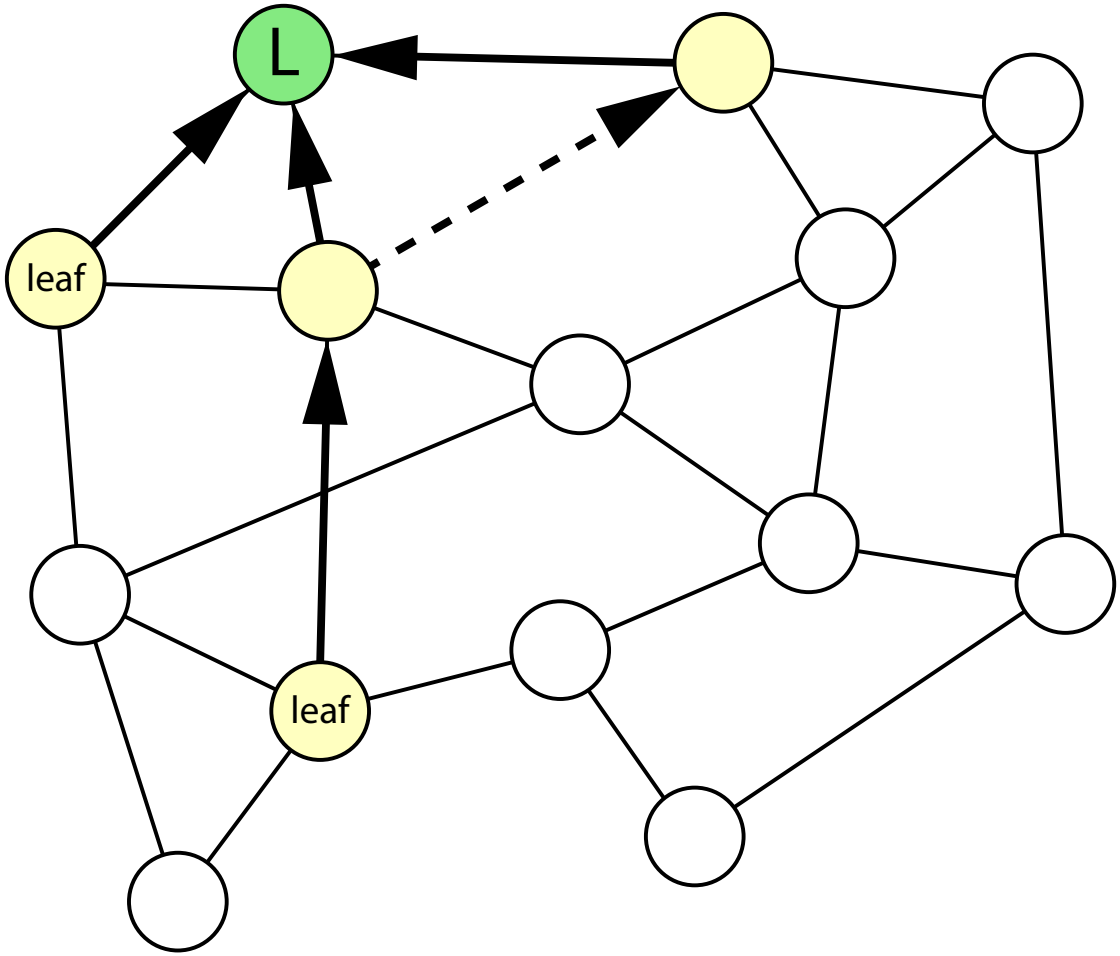
The leaves that receive a "new level" message start a new level of the spanning tree.

Application: Shortest-Path Spanning Tree Construction



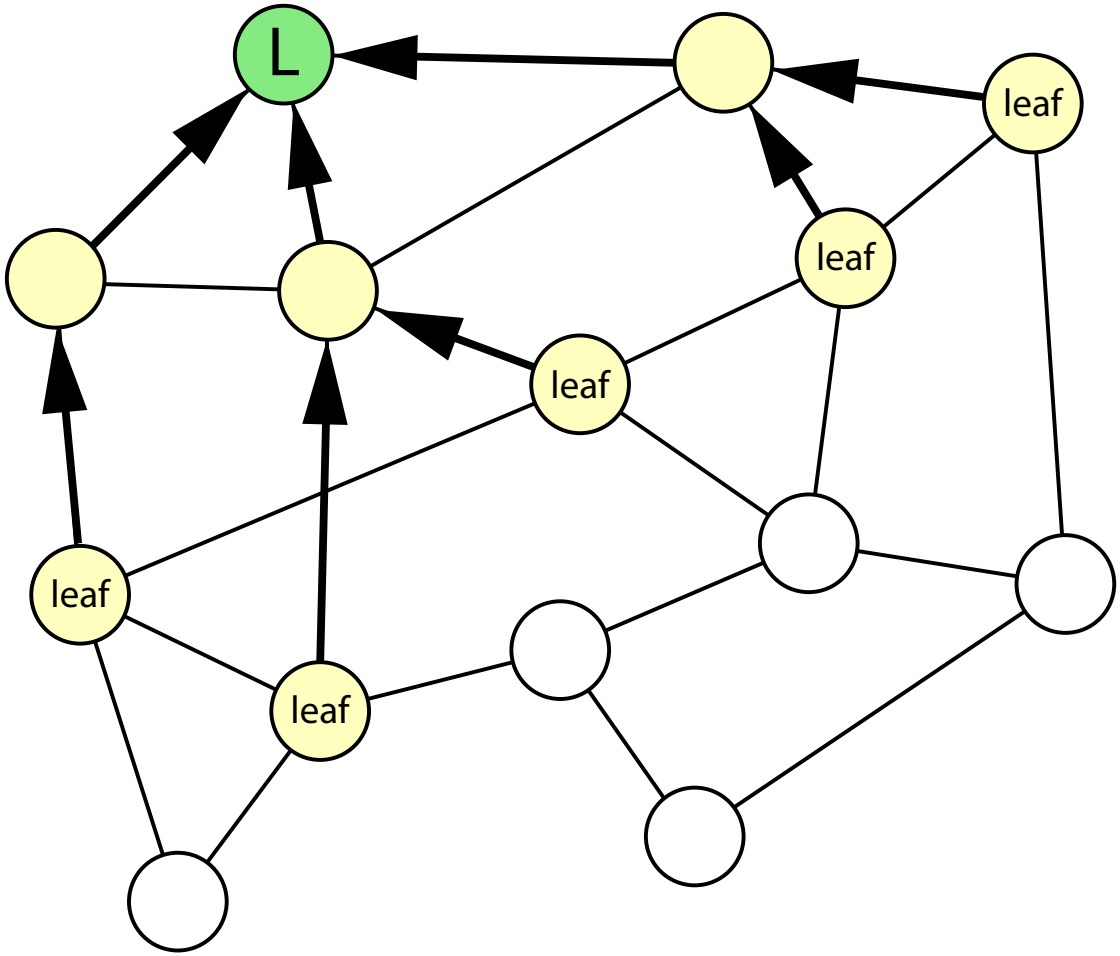
The leaves that receive a "new level" message start a new level of the spanning tree.

Application: Shortest-Path Spanning Tree Construction



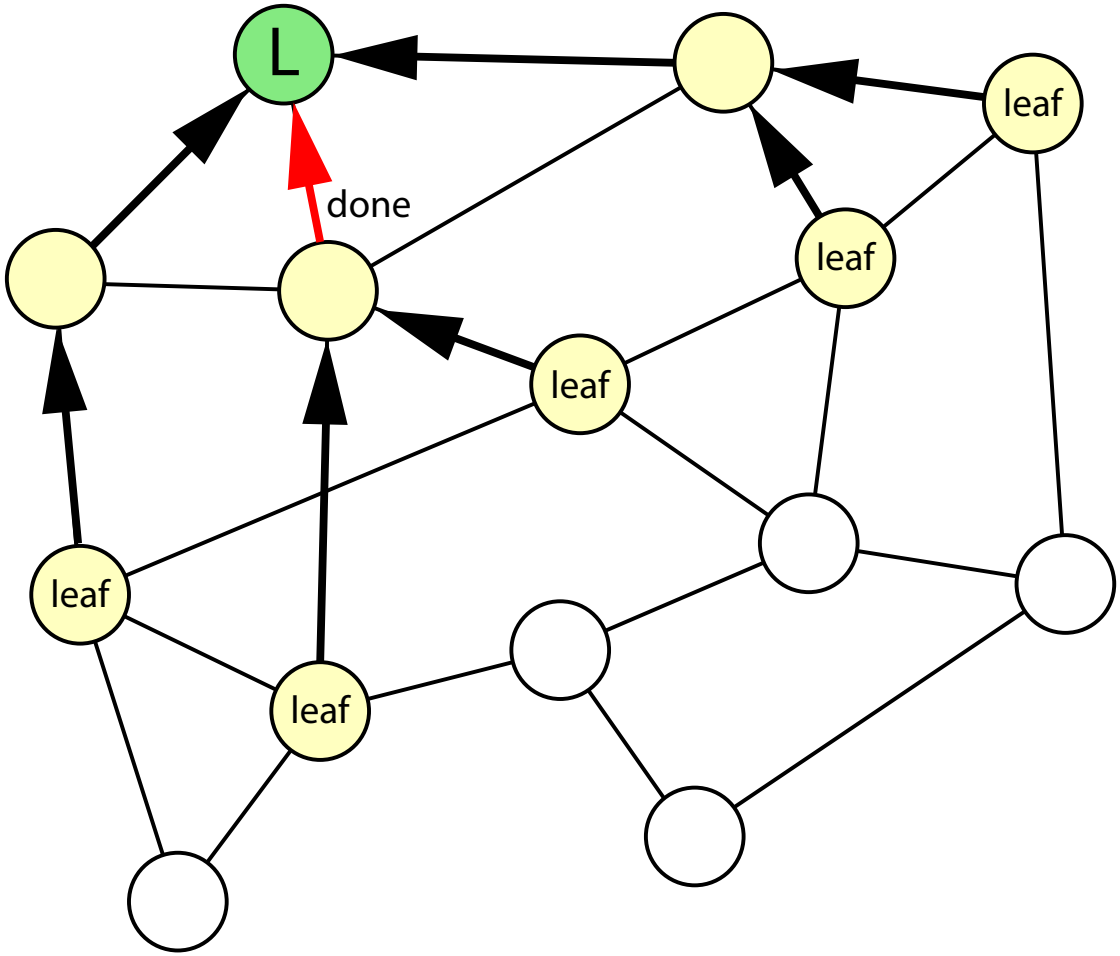
The leaves that receive a "new level" message start a new level of the spanning tree.

Application: Shortest-Path Spanning Tree Construction



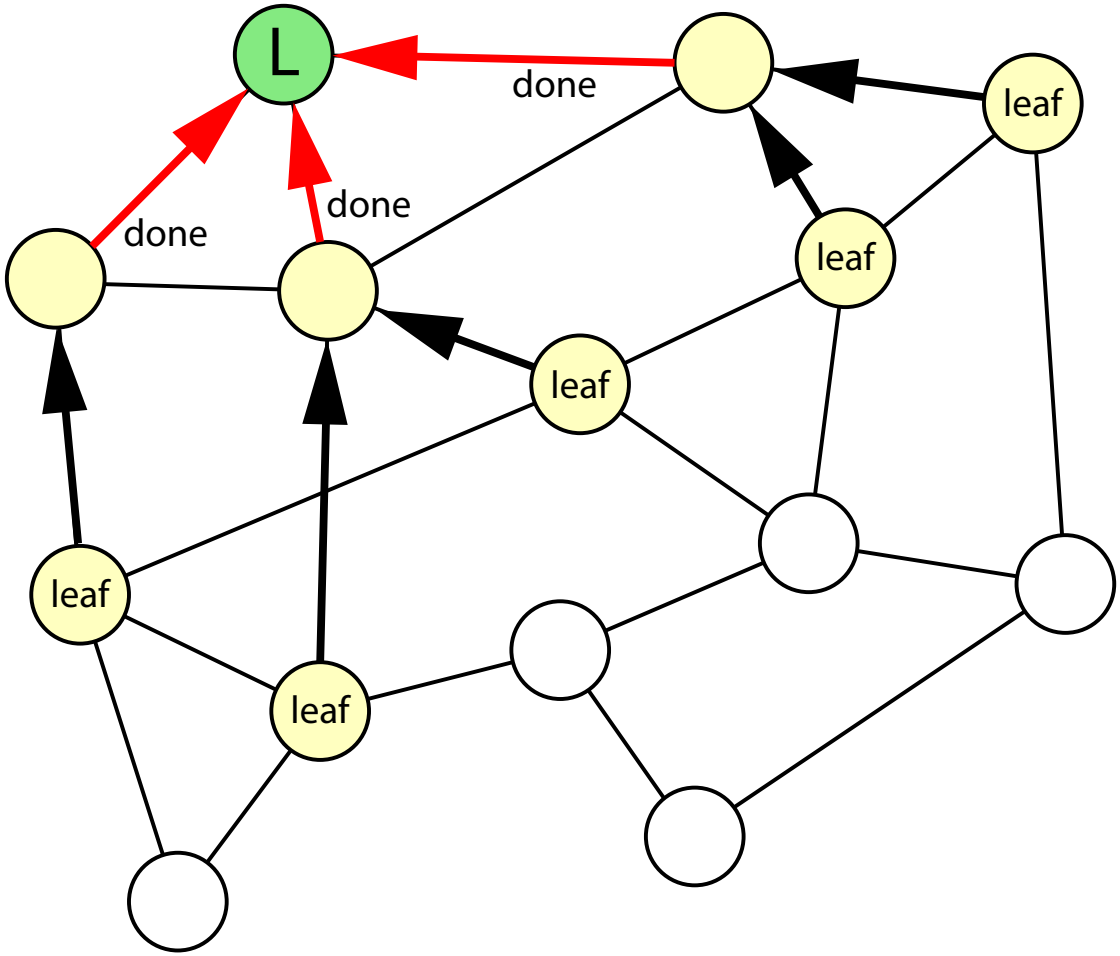
The leaves that receive a "new level" message start a new level of the spanning tree.

Application: Shortest-Path Spanning Tree Construction



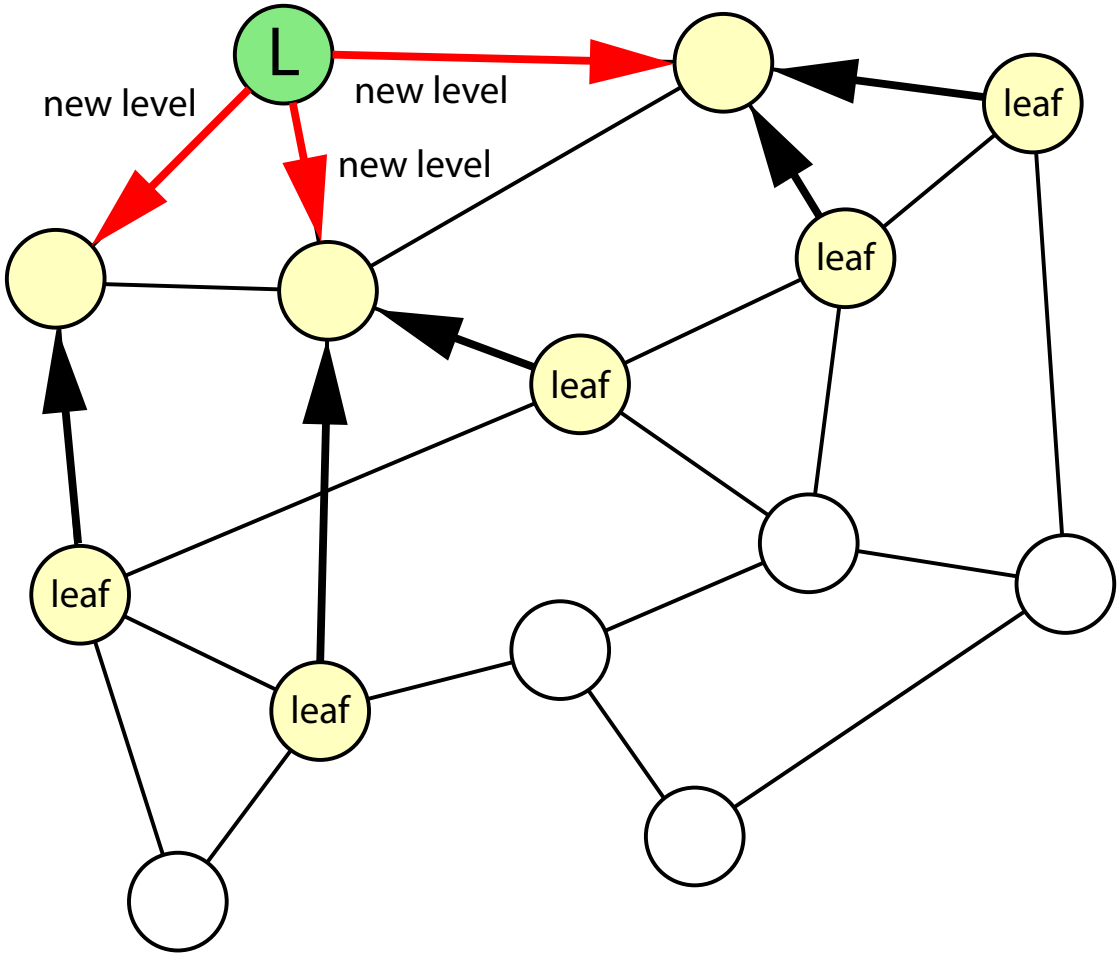
When they realize that all their neighbors have been included in the spanning tree, they send a "done" message to the leader.

Application: Shortest-Path Spanning Tree Construction



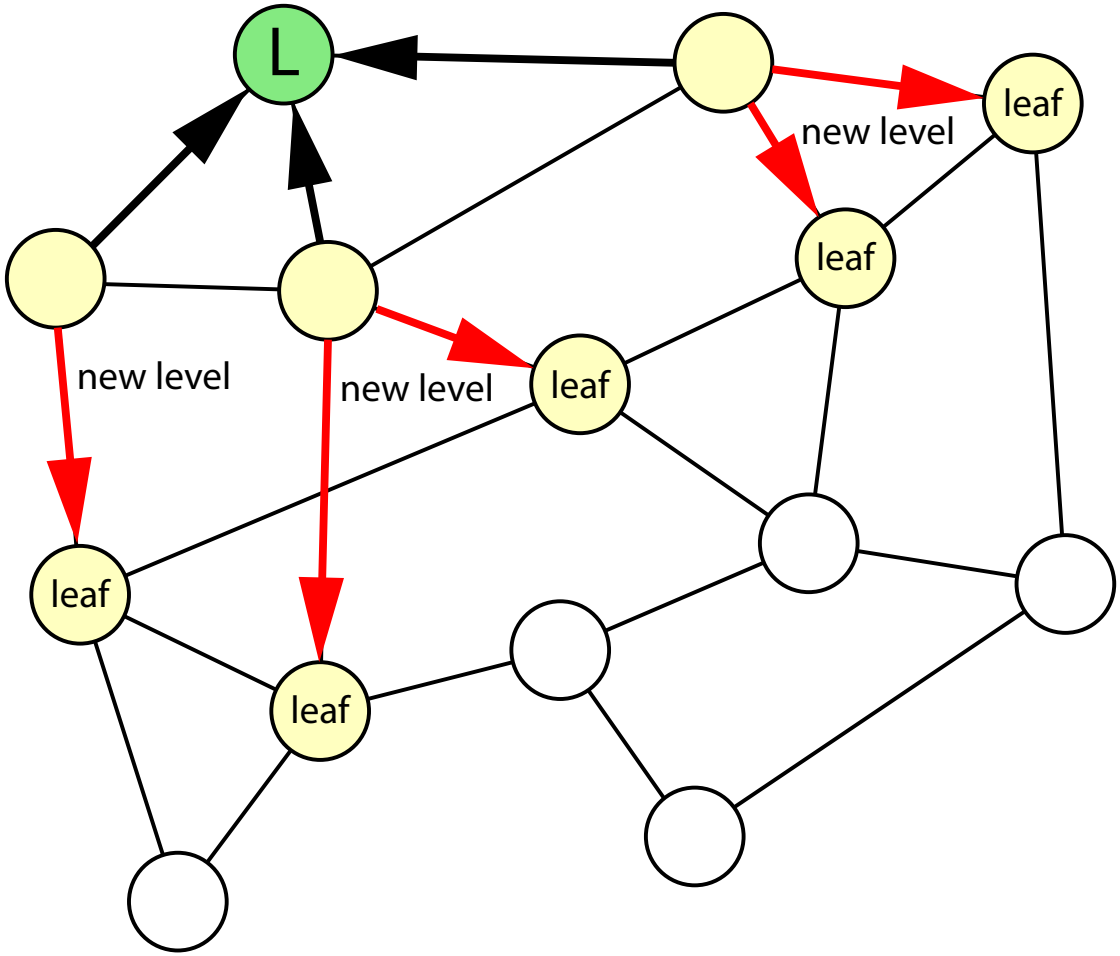
When they realize that all their neighbors have been included in the spanning tree, they send a "done" message to the leader.

Application: Shortest-Path Spanning Tree Construction



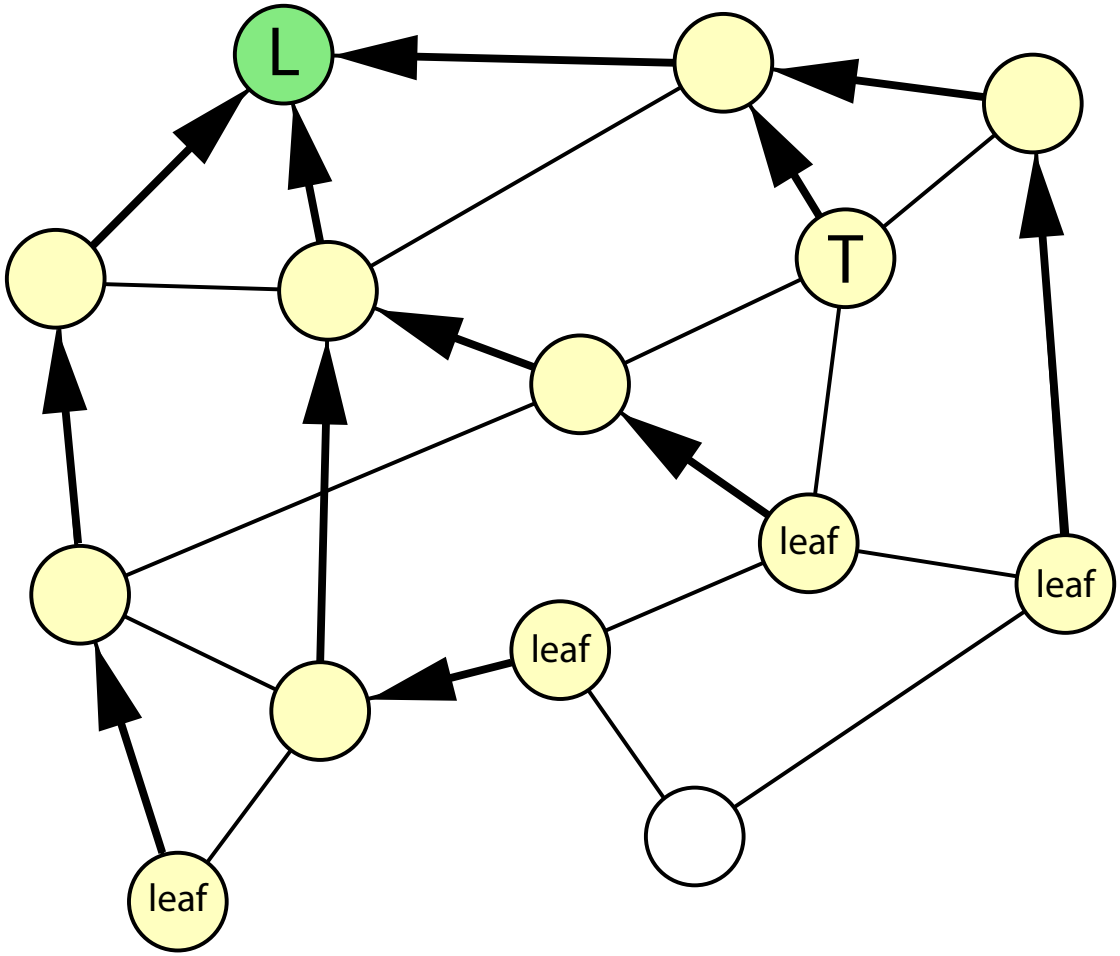
The leader issues another "new level" command, which is forwarded along the spanning tree.

Application: Shortest-Path Spanning Tree Construction



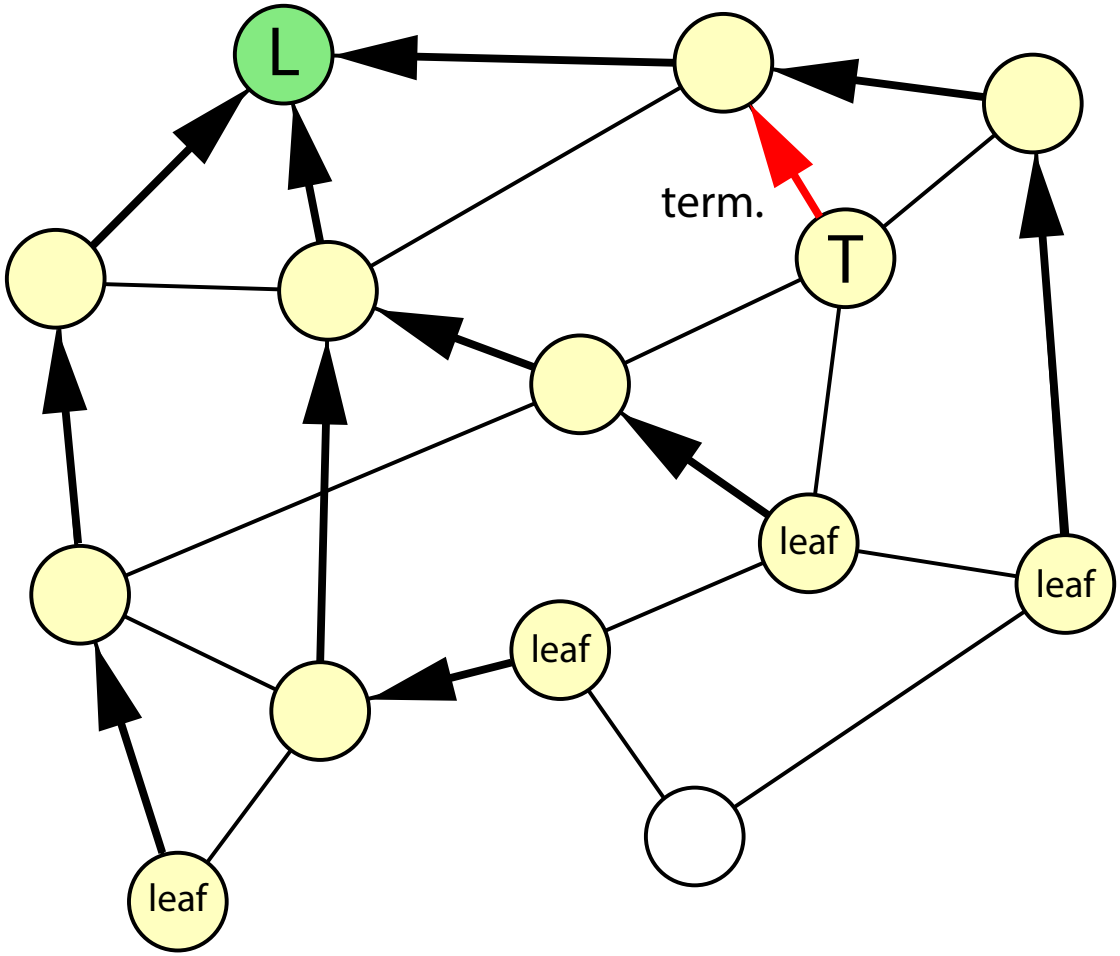
The leader issues another "new level" command, which is forwarded along the spanning tree.

Application: Shortest-Path Spanning Tree Construction



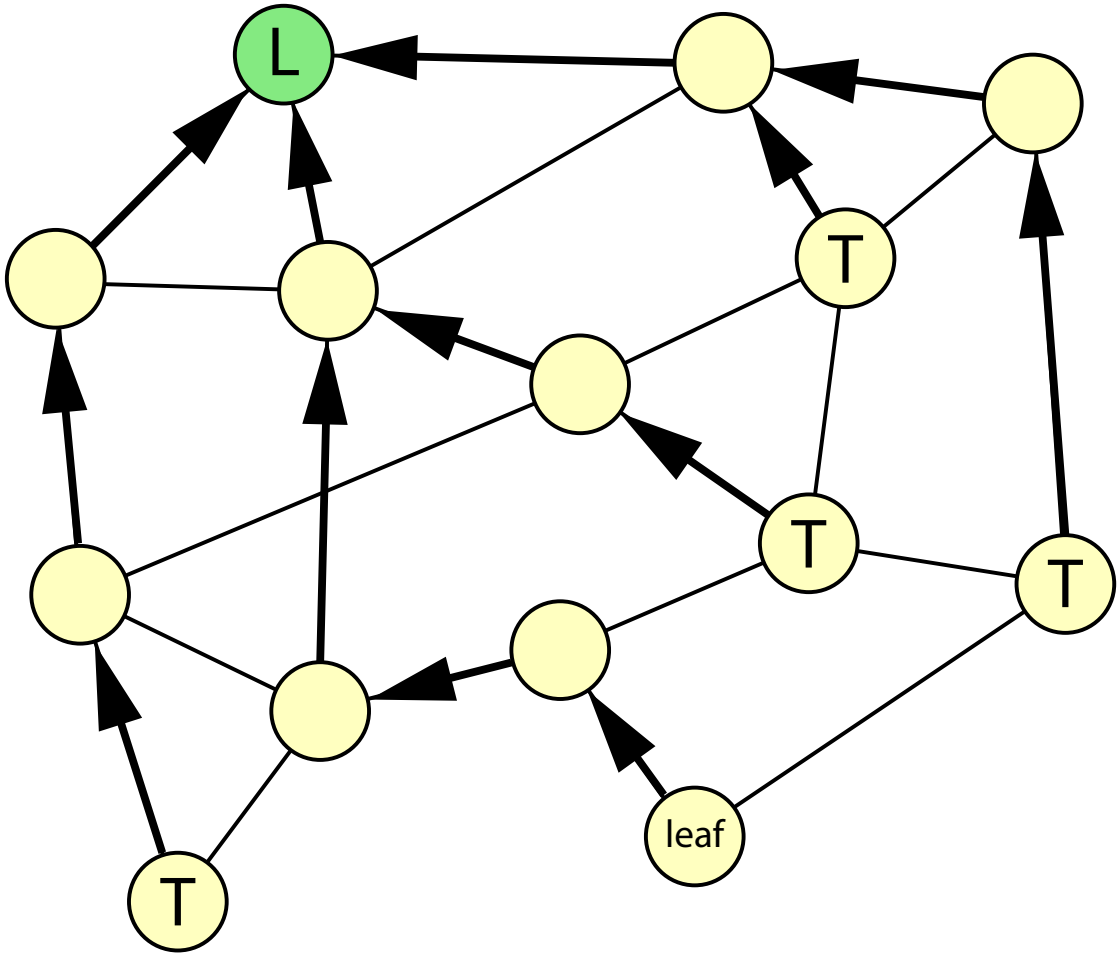
A new level of the spanning tree is constructed.

Application: Shortest-Path Spanning Tree Construction



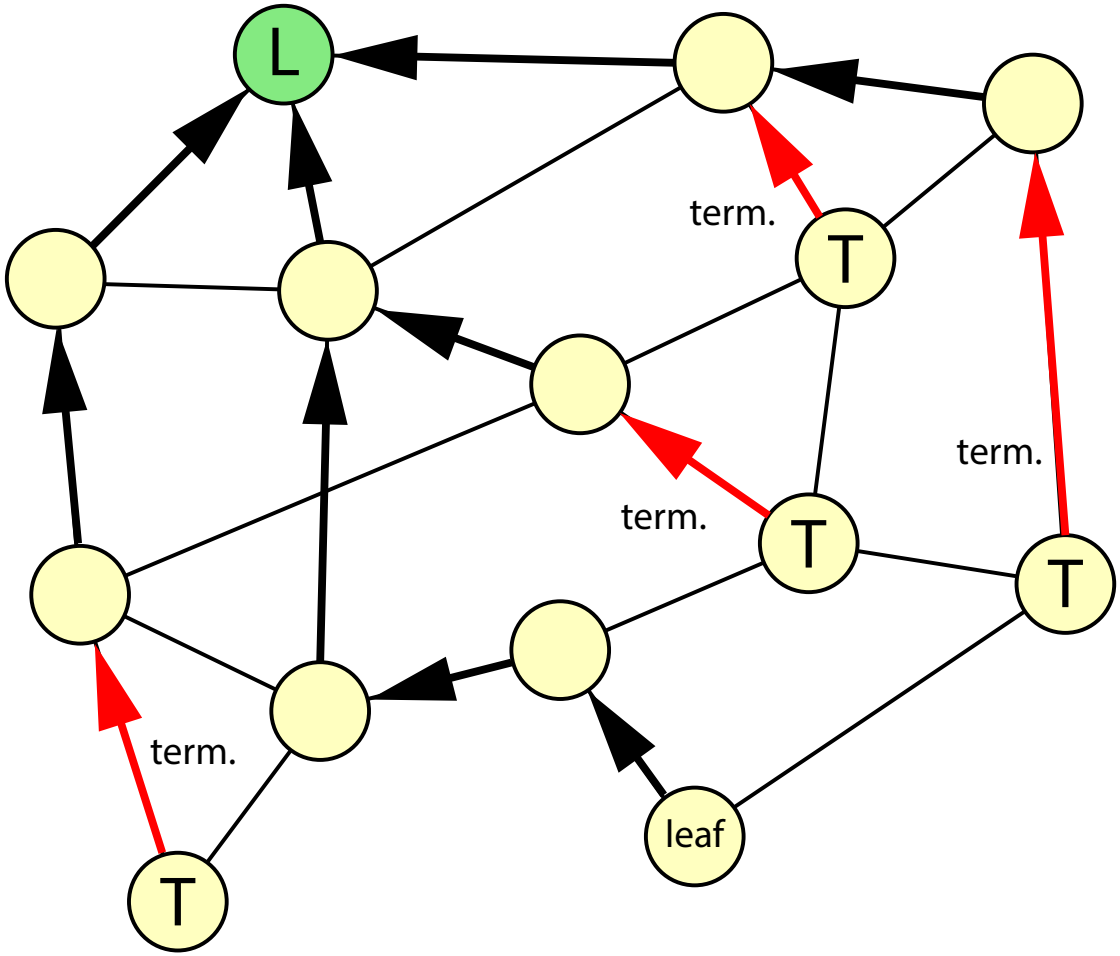
The leaves that are unable to expand assume a terminal state and send a "terminated" message toward the leader.

Application: Shortest-Path Spanning Tree Construction



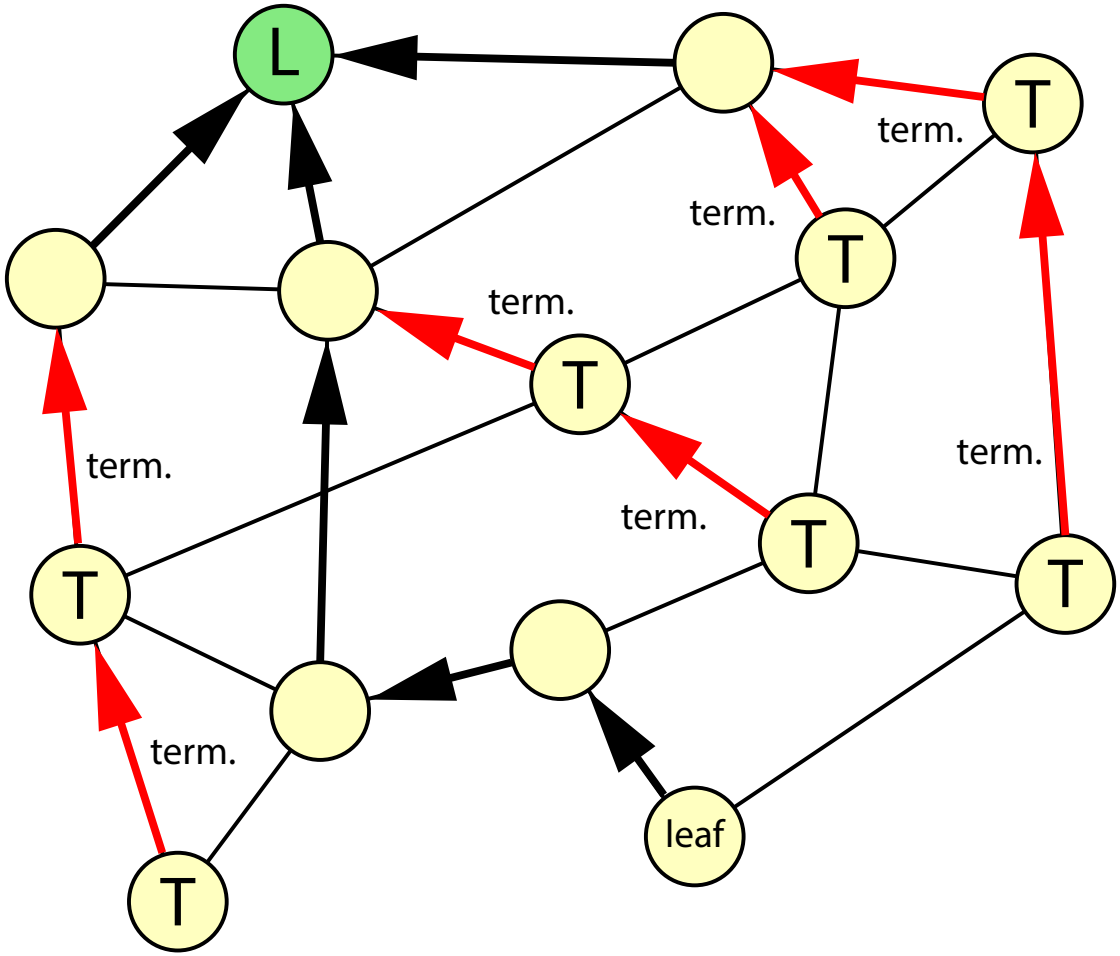
The leaves that are unable to expand assume a terminal state and send a "terminated" message toward the leader.

Application: Shortest-Path Spanning Tree Construction



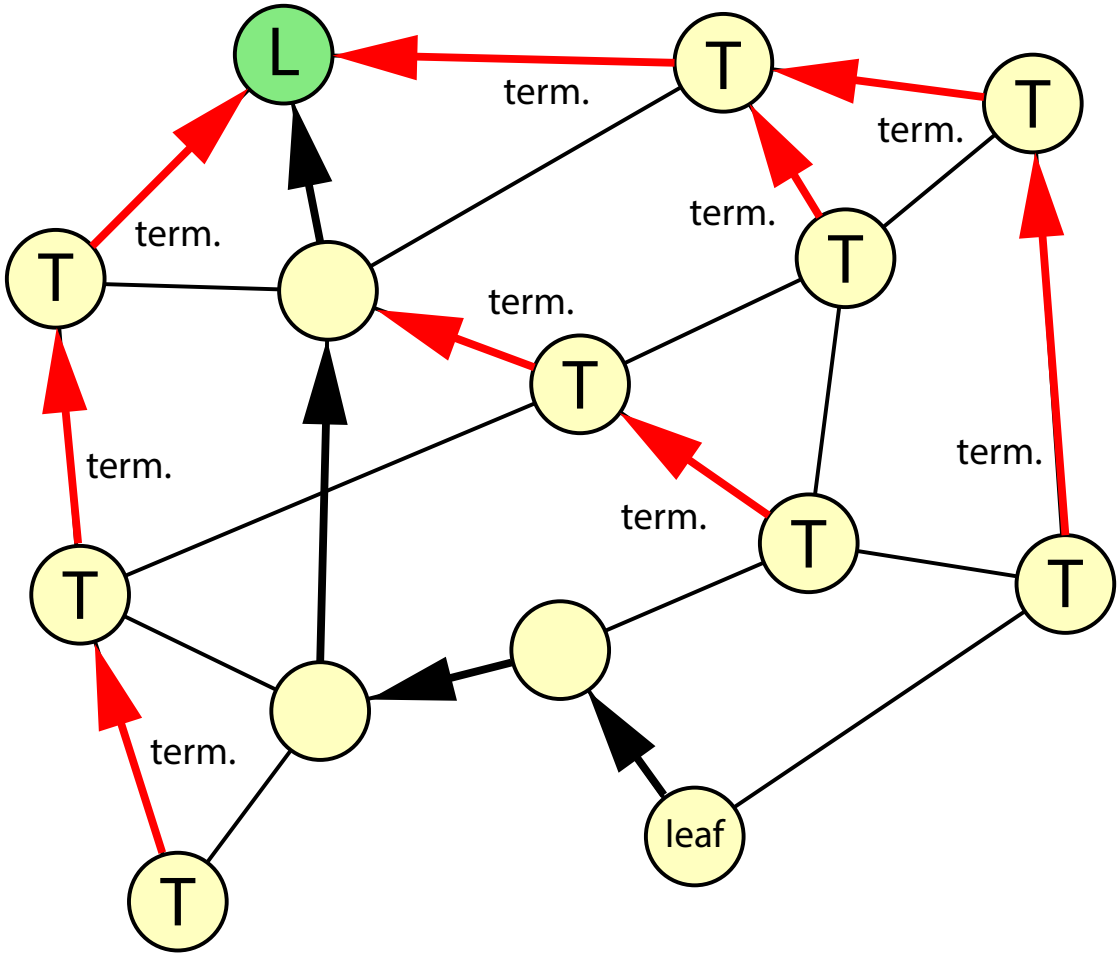
The leaves that are unable to expand assume a terminal state and send a "terminated" message toward the leader.

Application: Shortest-Path Spanning Tree Construction



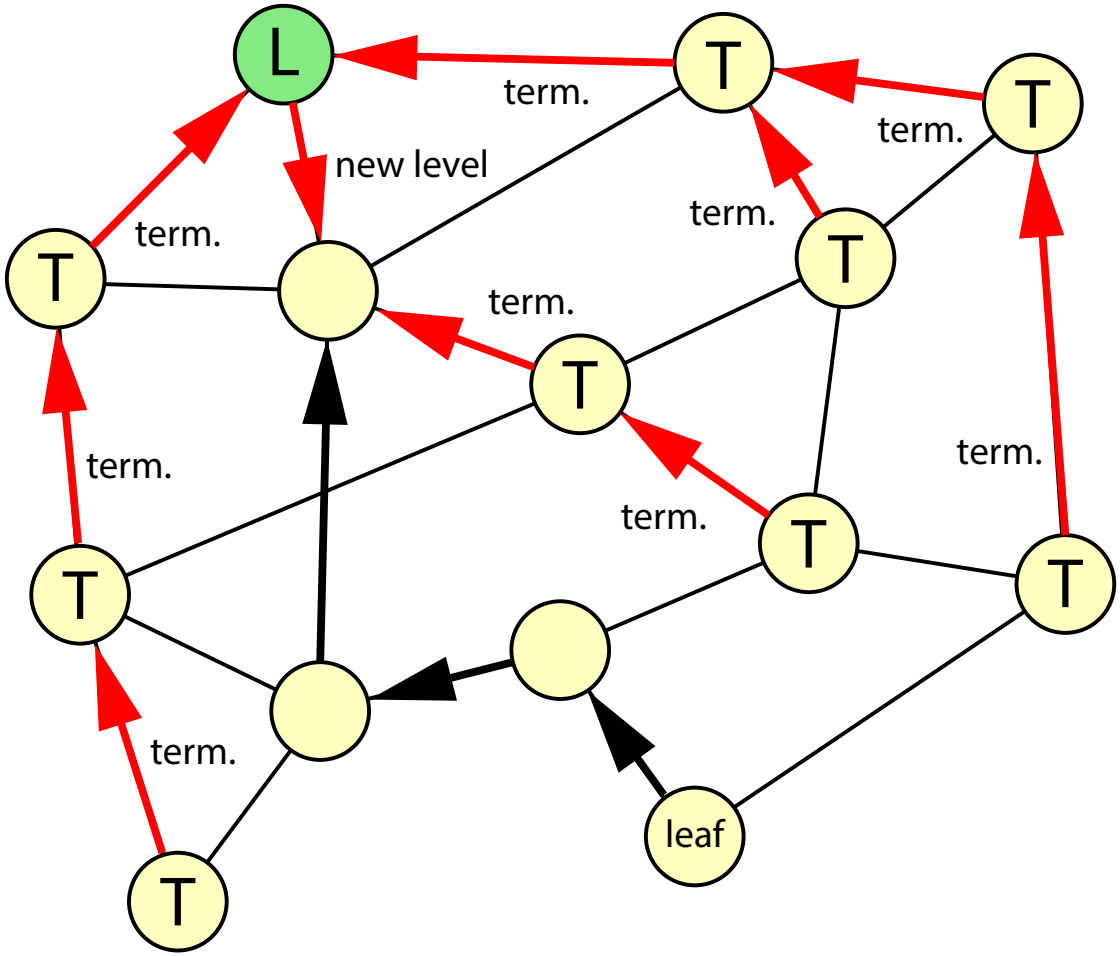
When an agent's children are all sending a "terminated" message, the agent forwards it and terminates as well.

Application: Shortest-Path Spanning Tree Construction



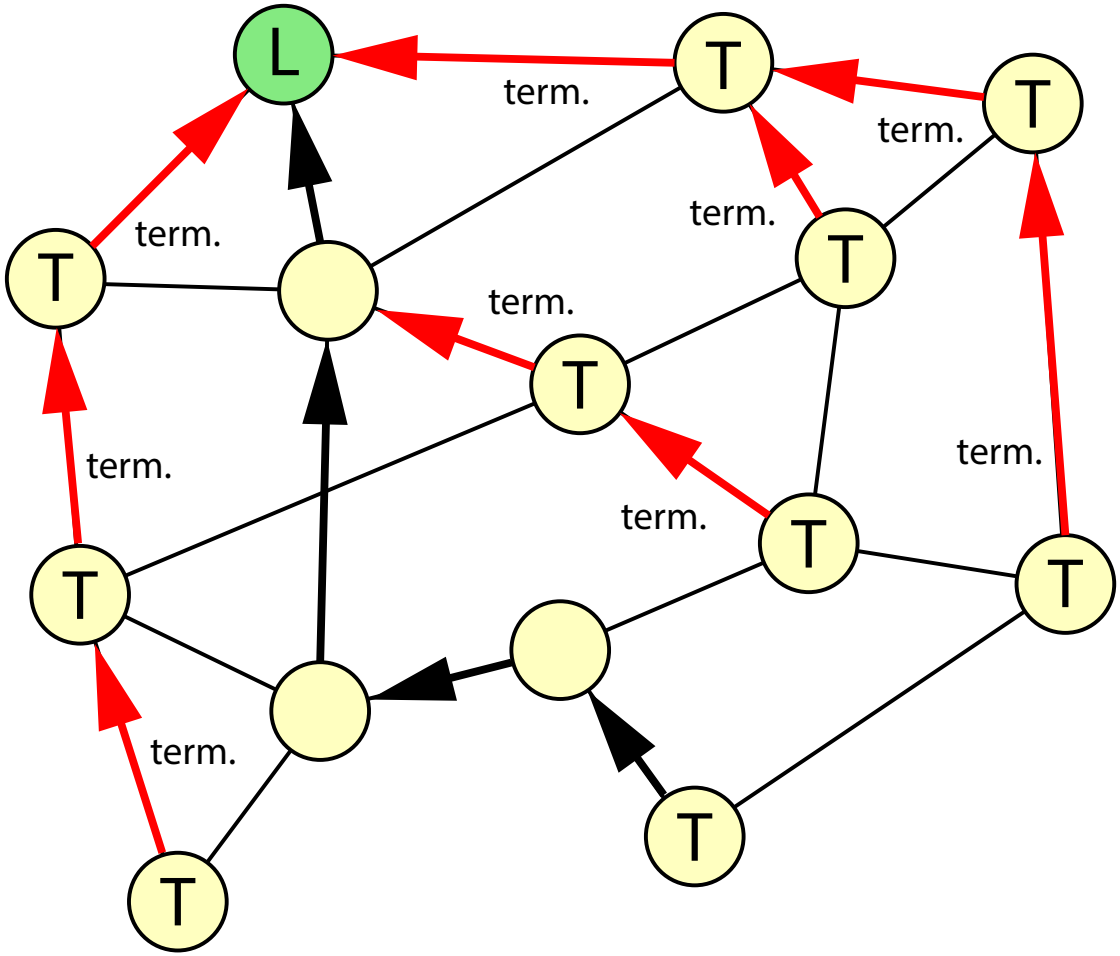
When an agent's children are all sending a "terminated" message, the agent forwards it and terminates as well.

Application: Shortest-Path Spanning Tree Construction



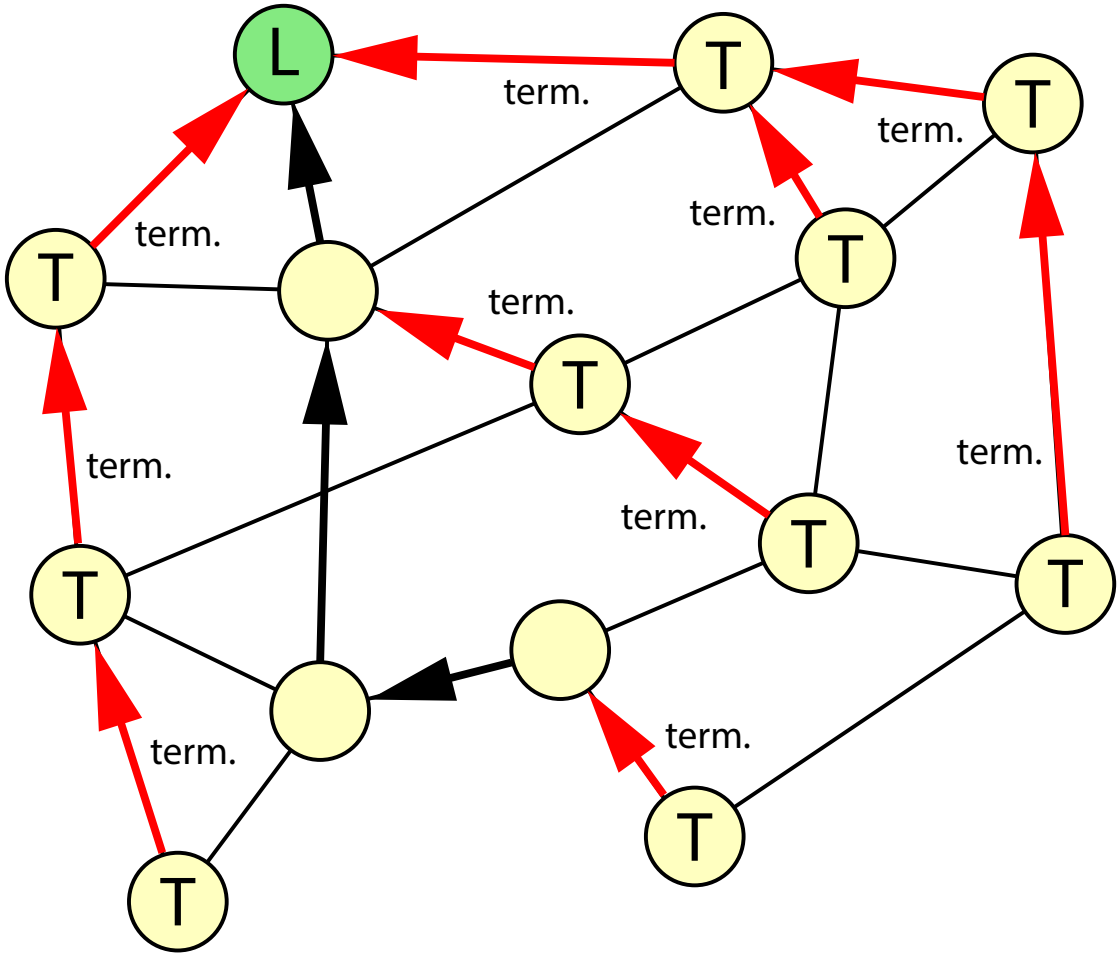
When an agent's children are all sending a "terminated" message, the agent forwards it and terminates as well.

Application: Shortest-Path Spanning Tree Construction



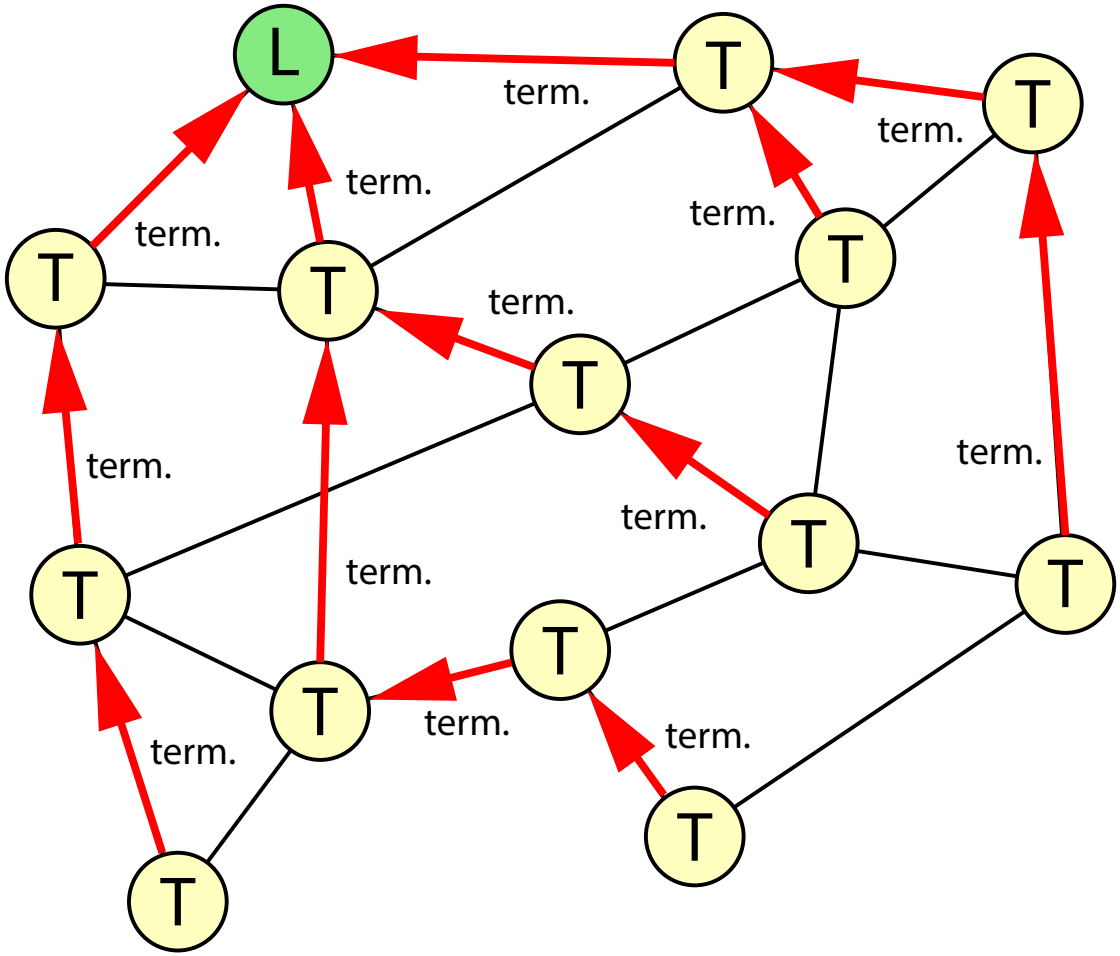
When an agent's children are all sending a "terminated" message, the agent forwards it and terminates as well.

Application: Shortest-Path Spanning Tree Construction



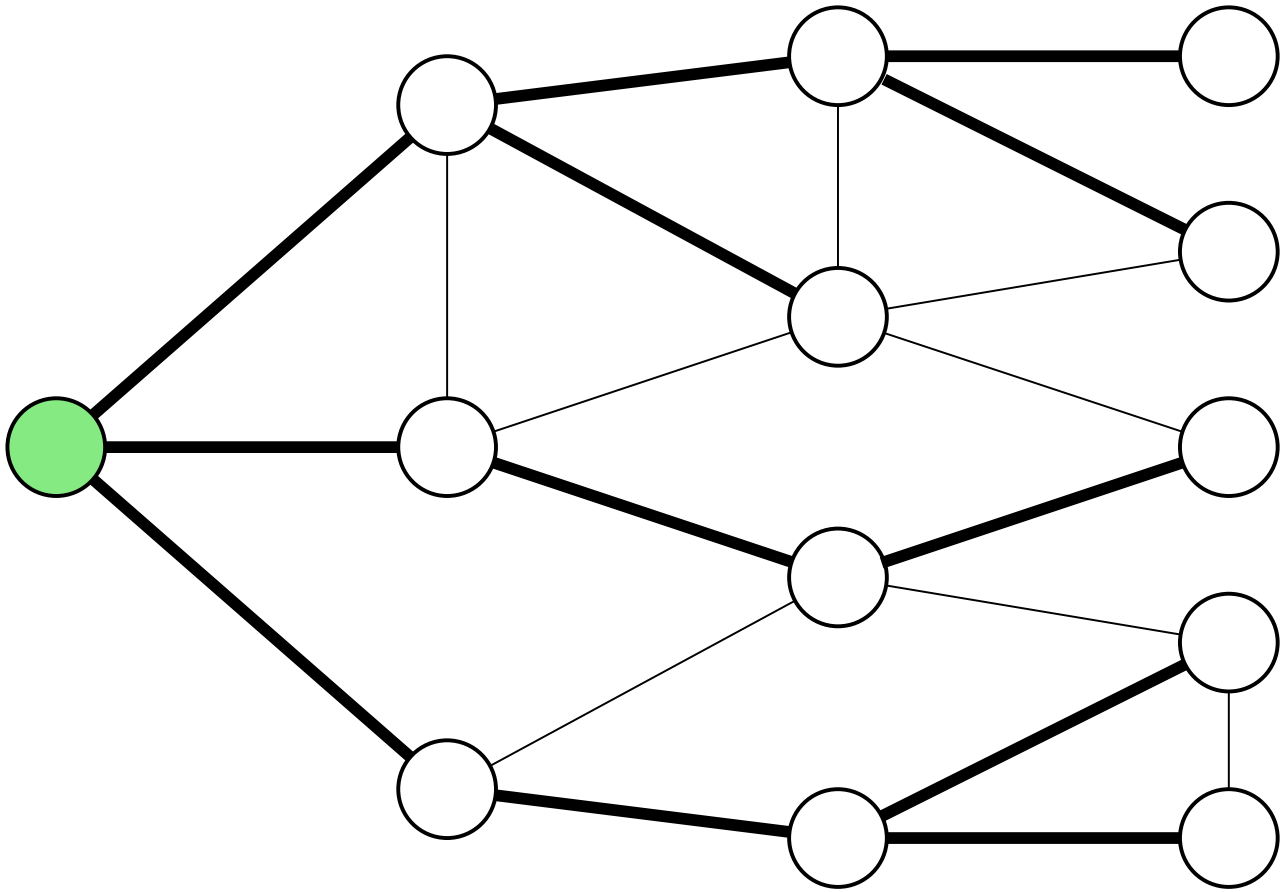
When an agent's children are all sending a "terminated" message, the agent forwards it and terminates as well.

Application: Shortest-Path Spanning Tree Construction



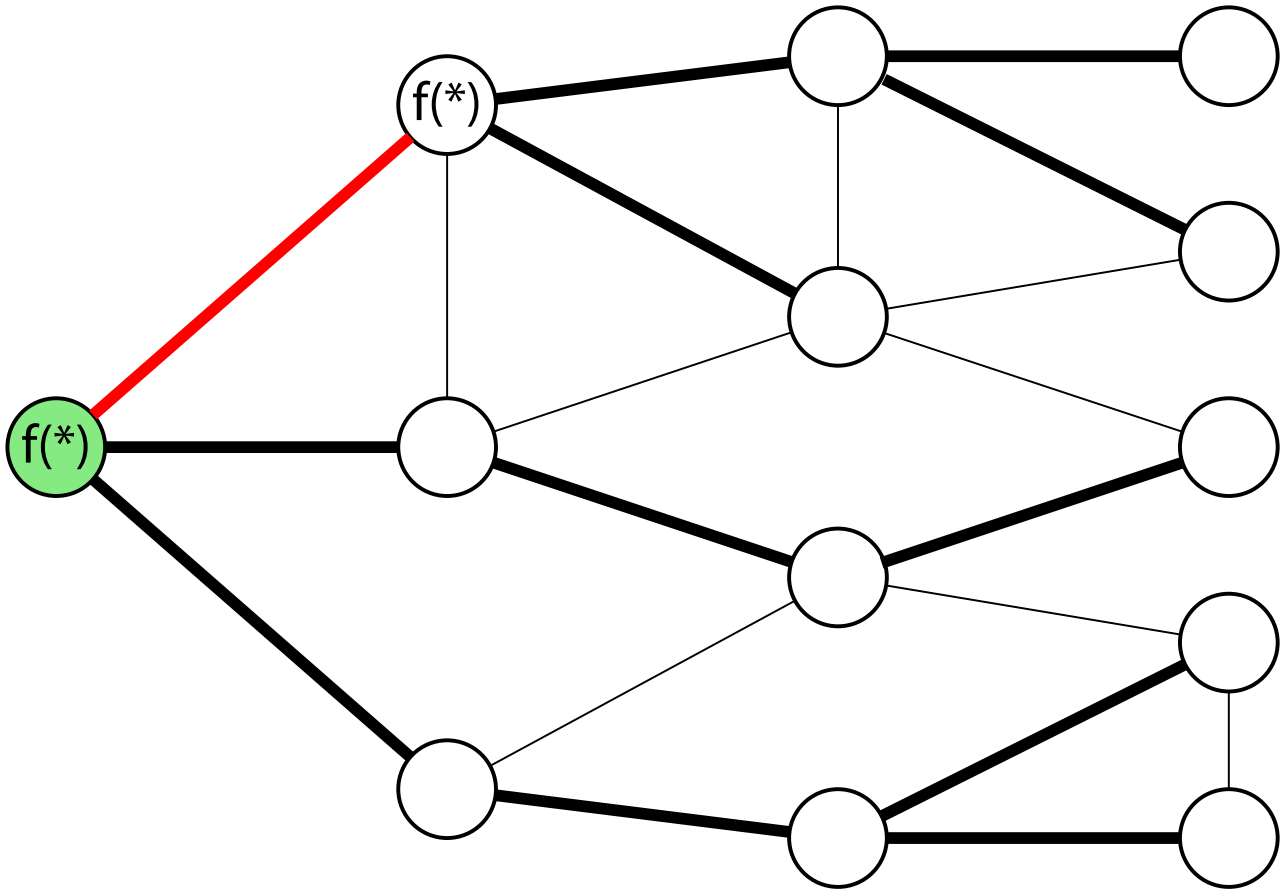
When the leader receives a "terminated" message from all its children, it terminates.

Application: Stability Detection



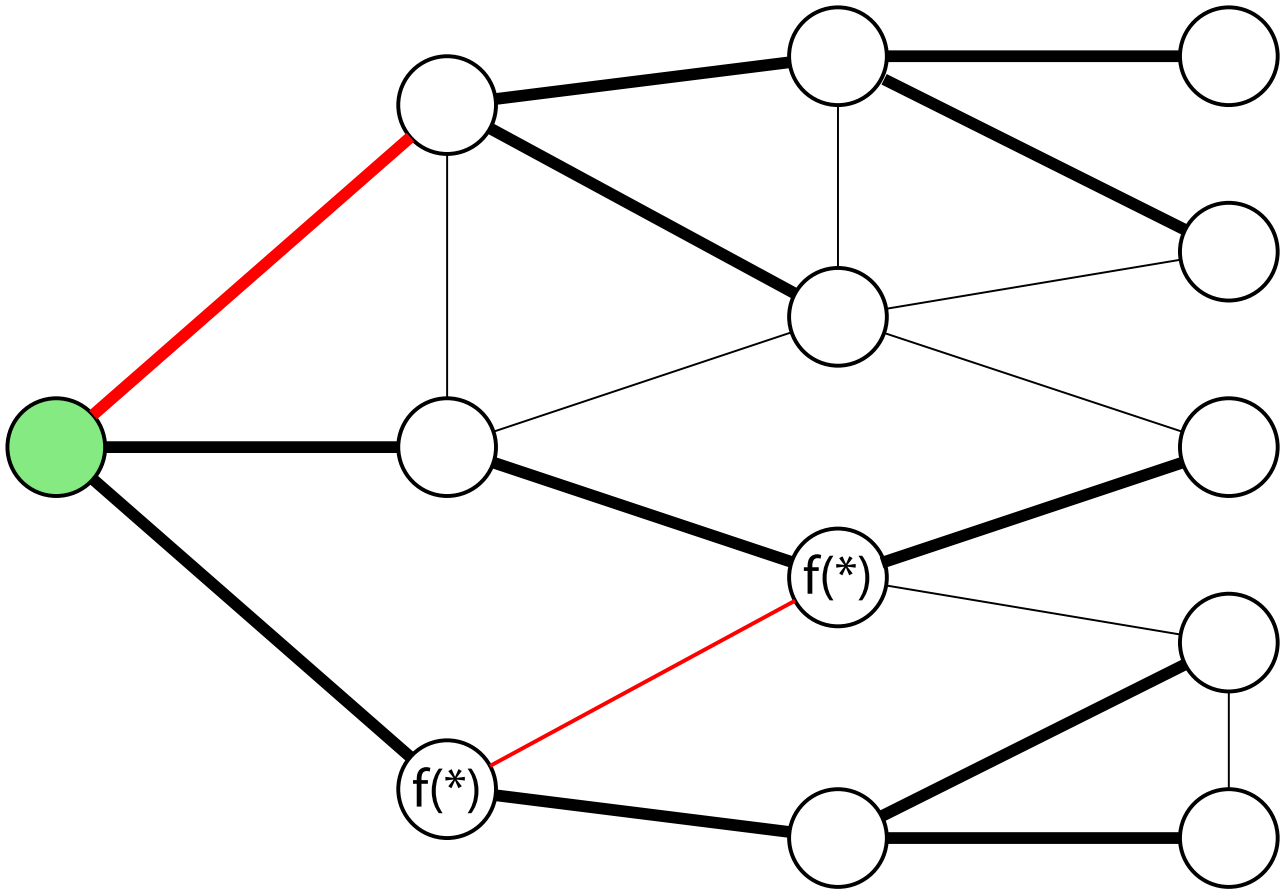
Say we have a leader and a spanning tree, and we want to detect (under the k-bounded scheduler) when a protocol P stabilizes.

Application: Stability Detection



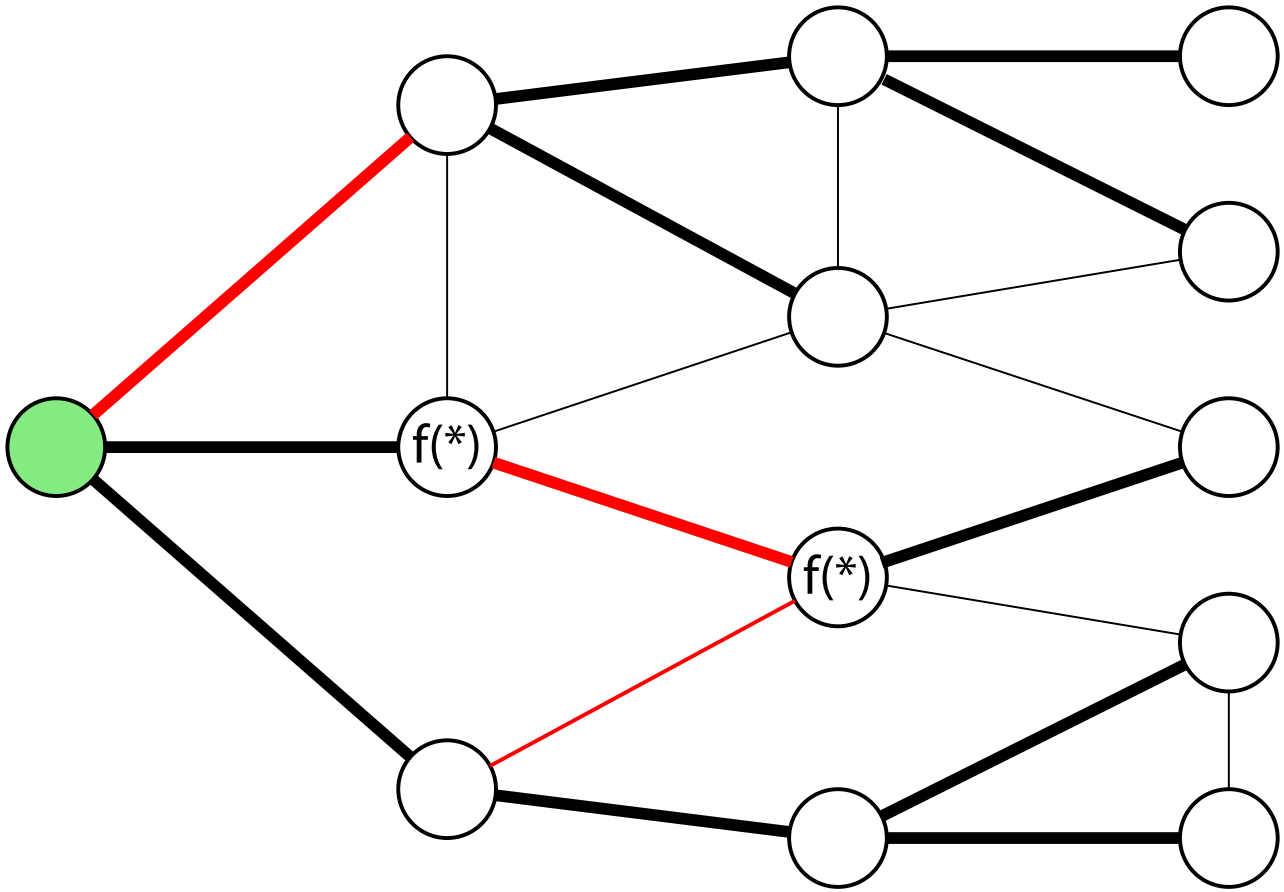
Whenever a new edge is activated, its endpoints "simulate" a transition according to P.

Application: Stability Detection



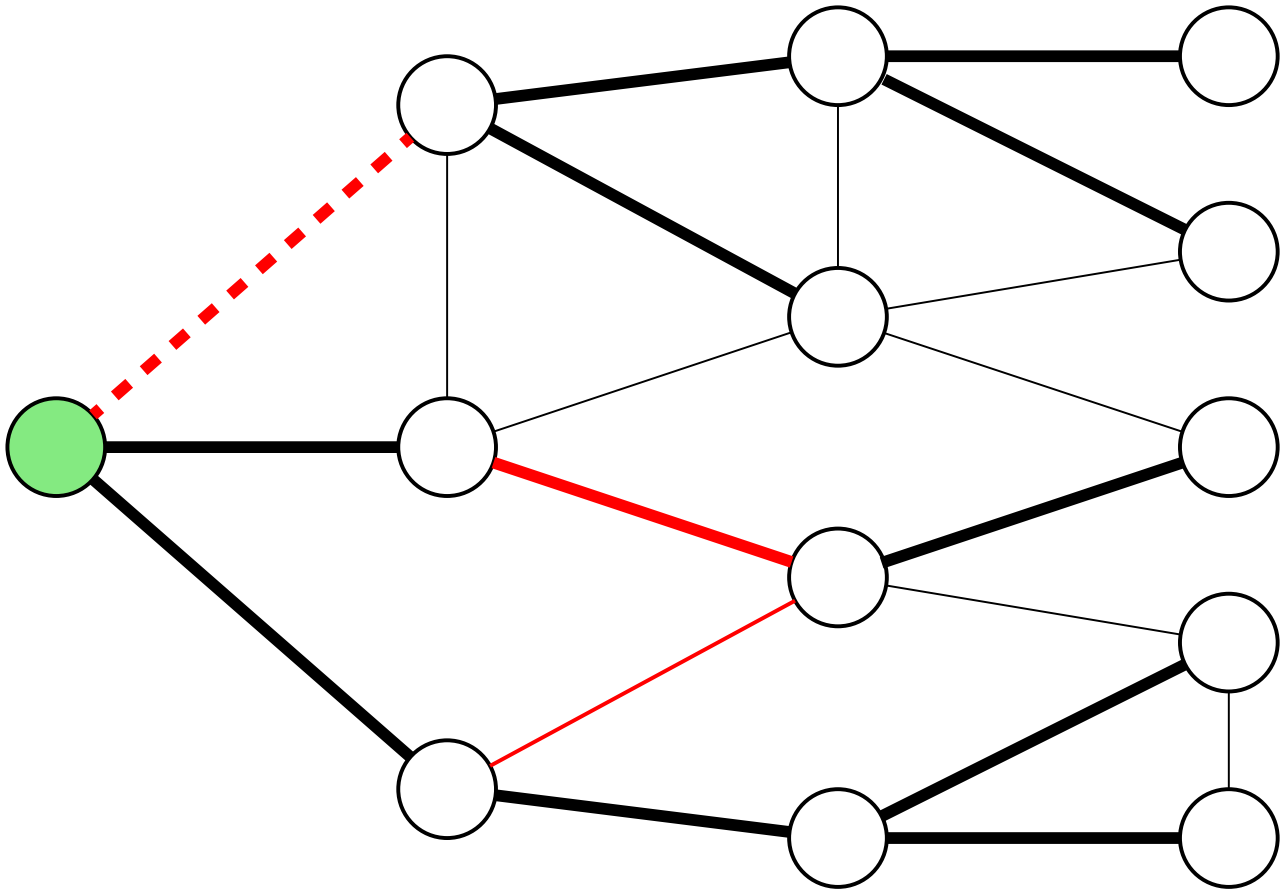
Whenever a new edge is activated, its endpoints "simulate" a transition according to P.

Application: Stability Detection



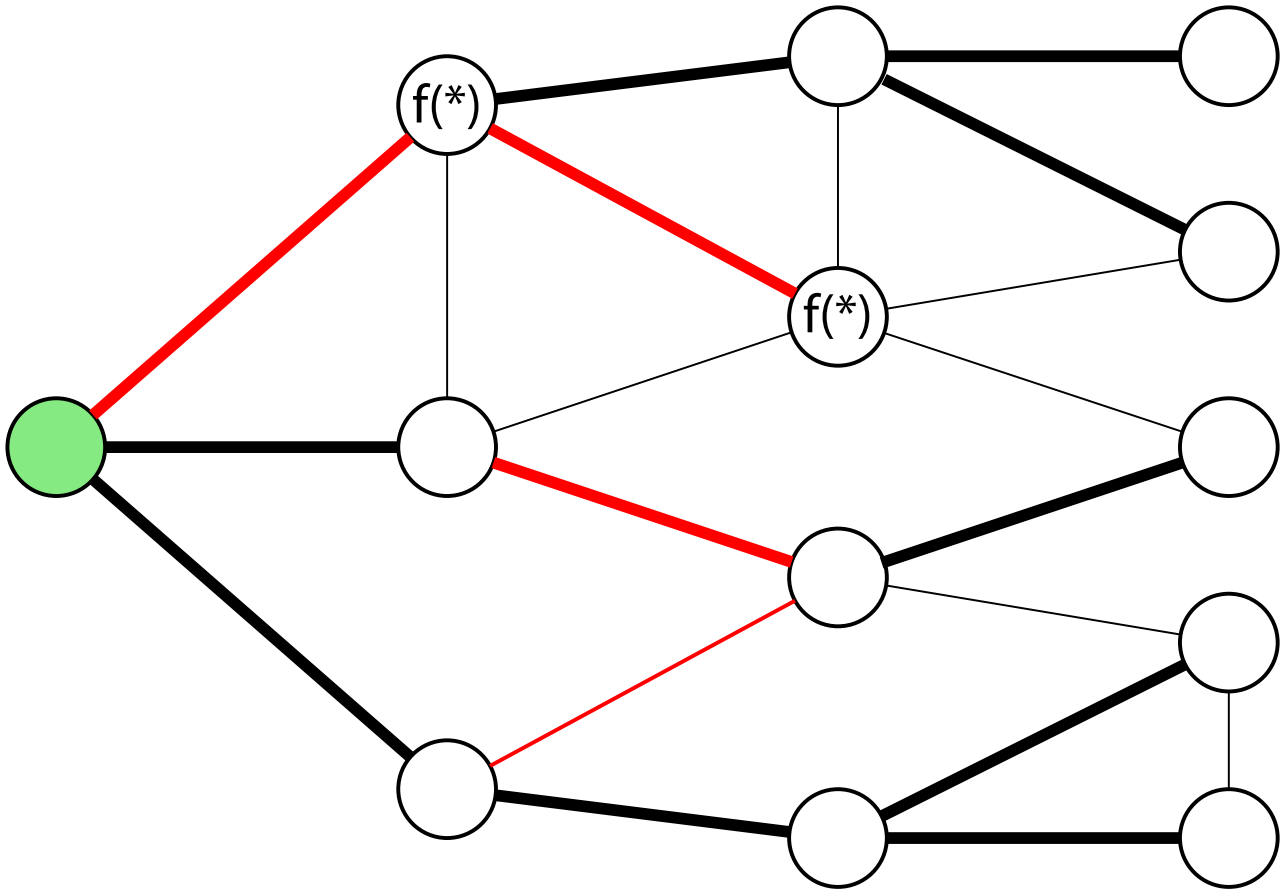
Whenever a new edge is activated, its endpoints "simulate" a transition according to P.

Application: Stability Detection



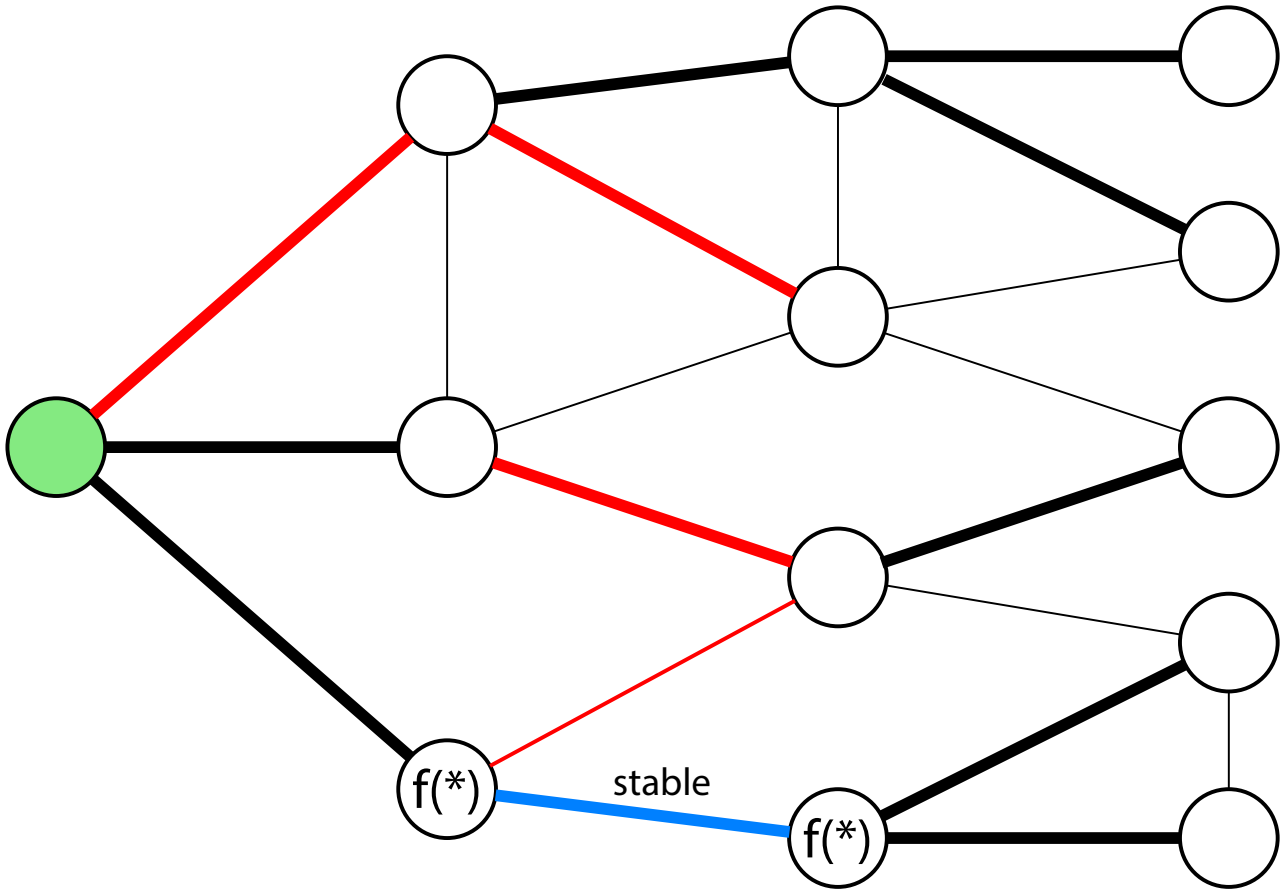
These edges are marked, so the corresponding simulated interaction does not occur twice.

Application: Stability Detection



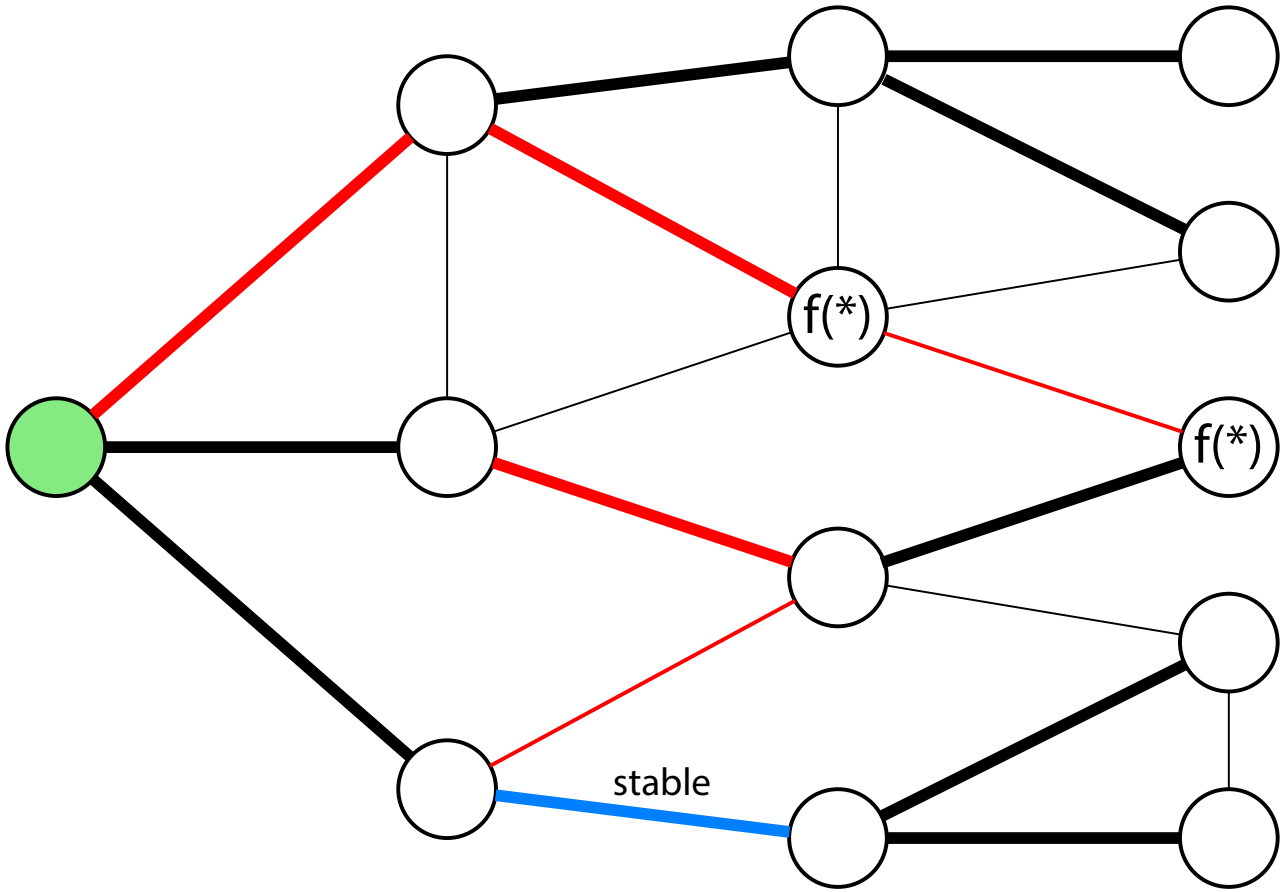
These edges are marked, so the corresponding simulated interaction does not occur twice.

Application: Stability Detection



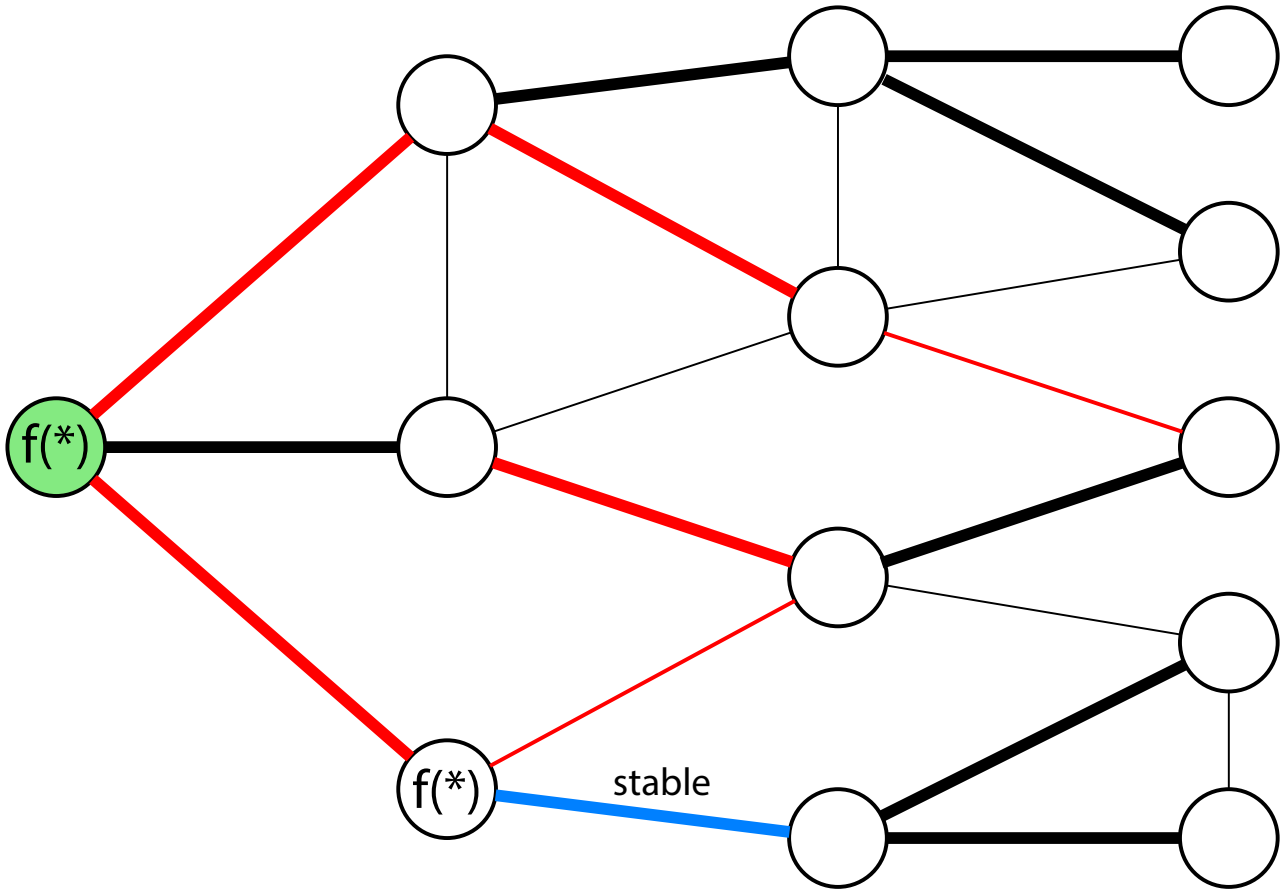
If a simulated interaction over an edge leaves the simulated states unchanged, the edge is marked as “stable”.

Application: Stability Detection



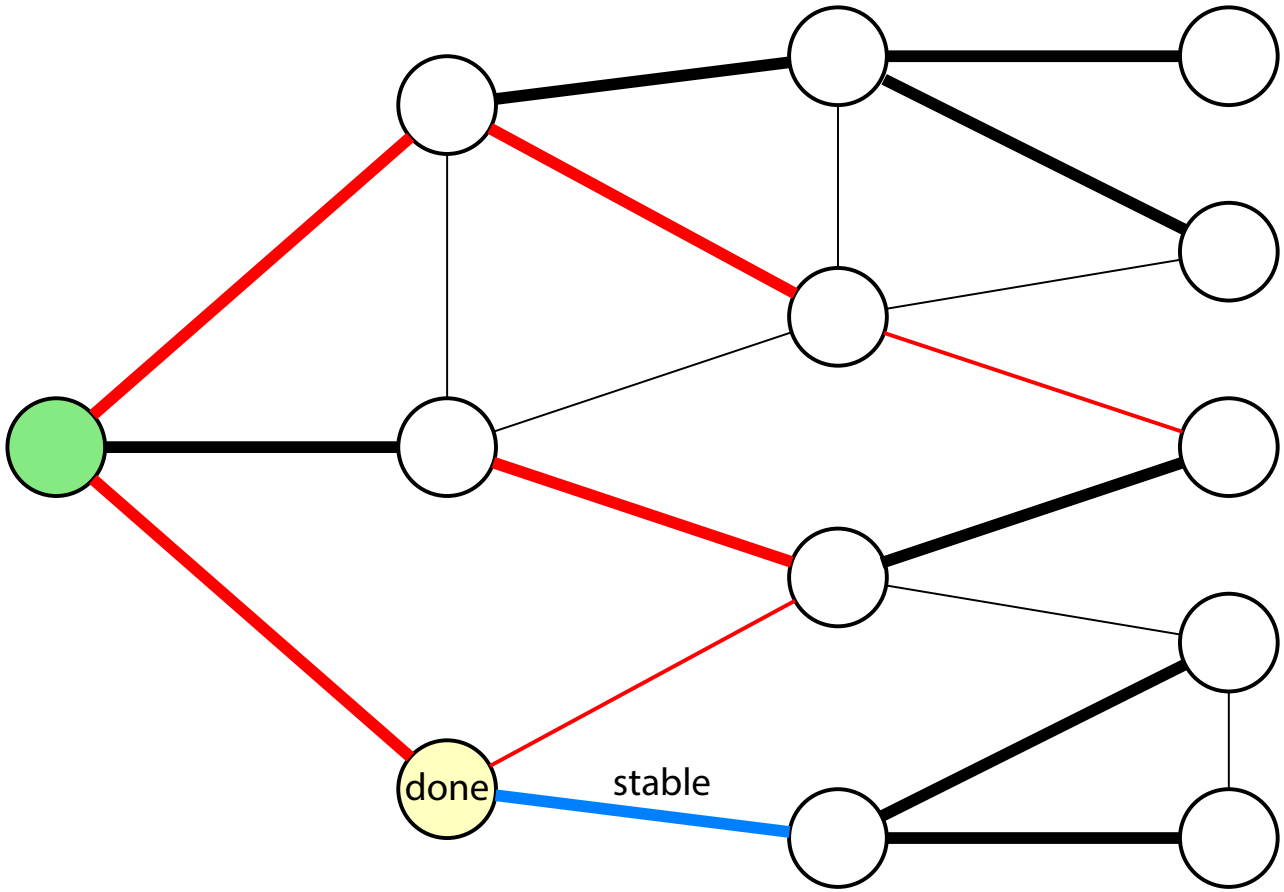
If a simulated interaction over an edge leaves the simulated states unchanged, the edge is marked as “stable”.

Application: Stability Detection



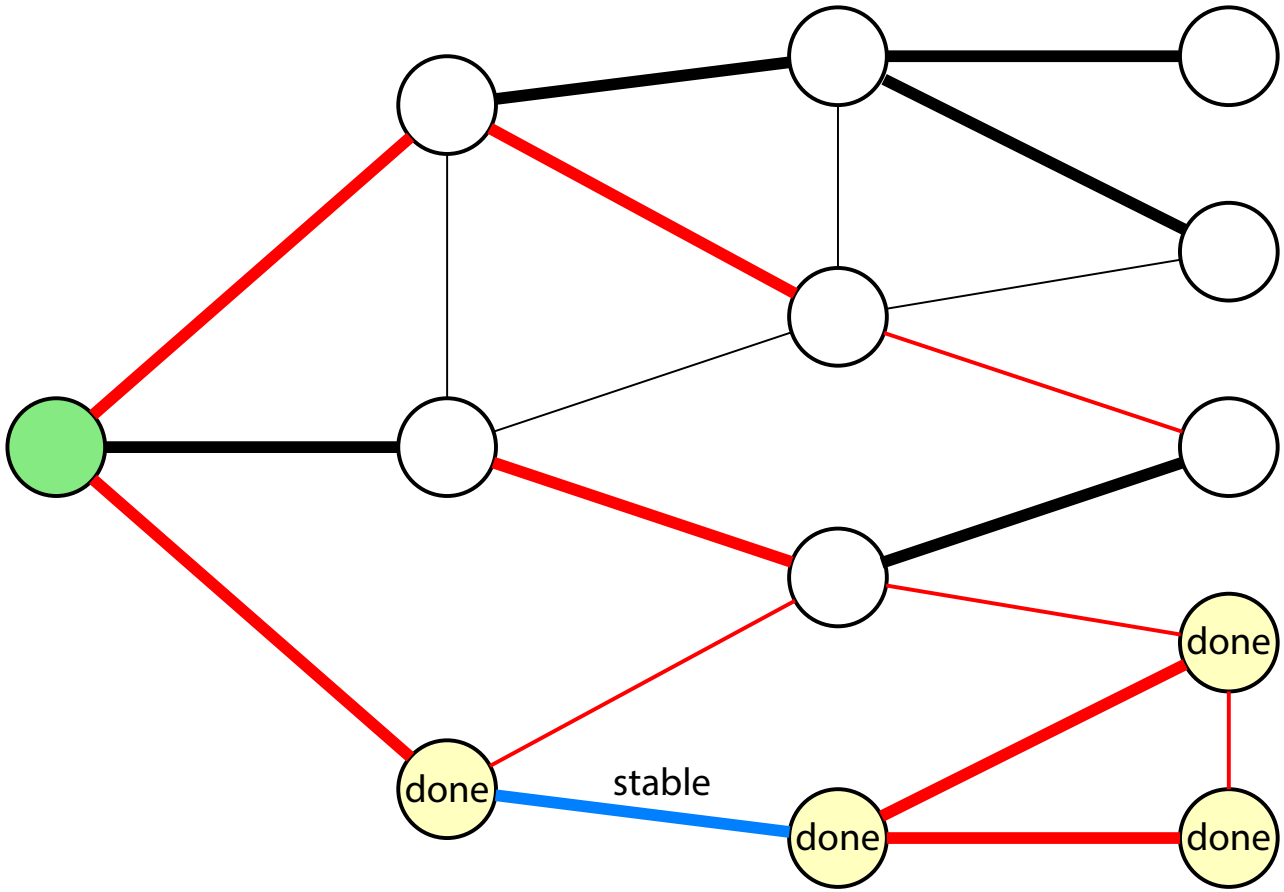
Since the scheduler is k-bounded, an agent eventually realizes that it has interacted with all its neighbors.

Application: Stability Detection



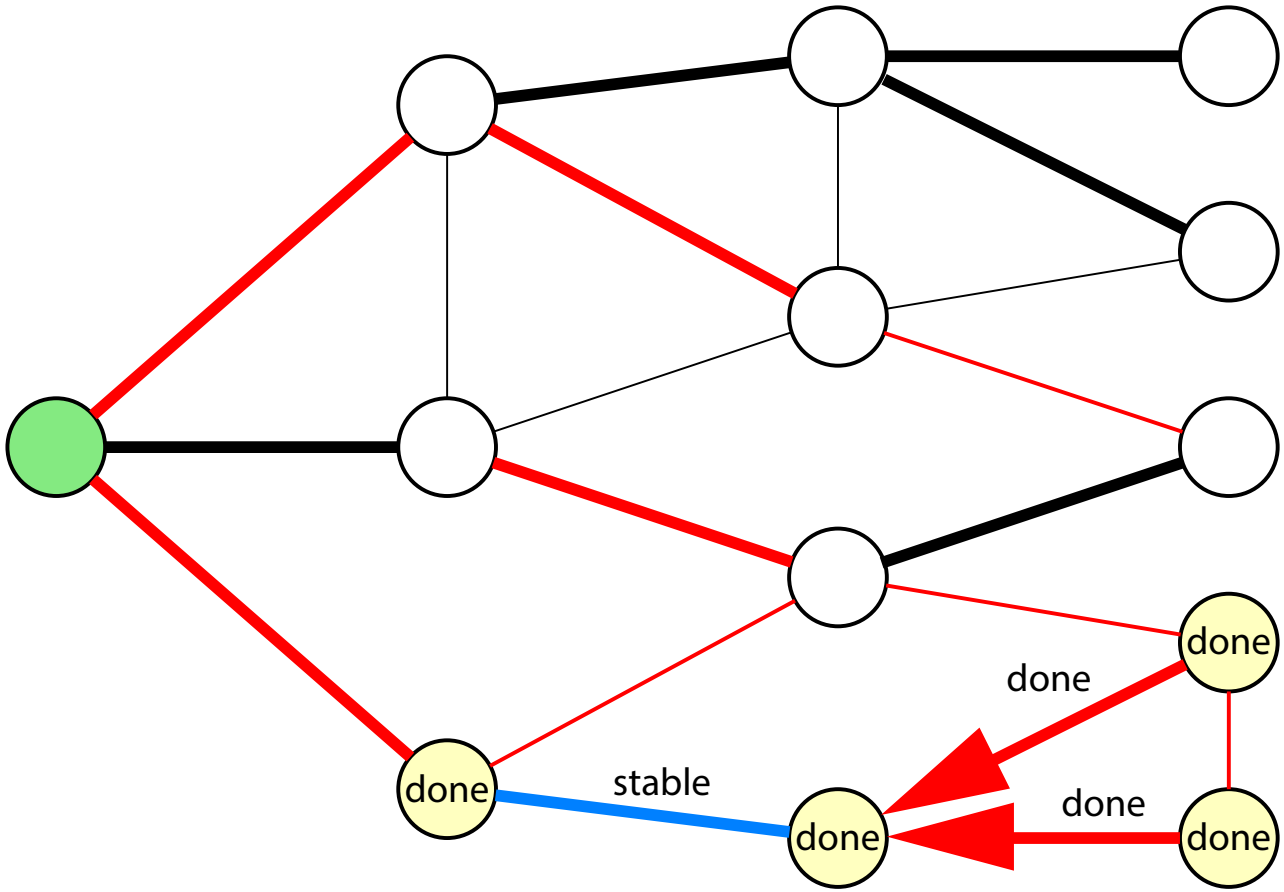
When this happens, the agent becomes “done”.

Application: Stability Detection



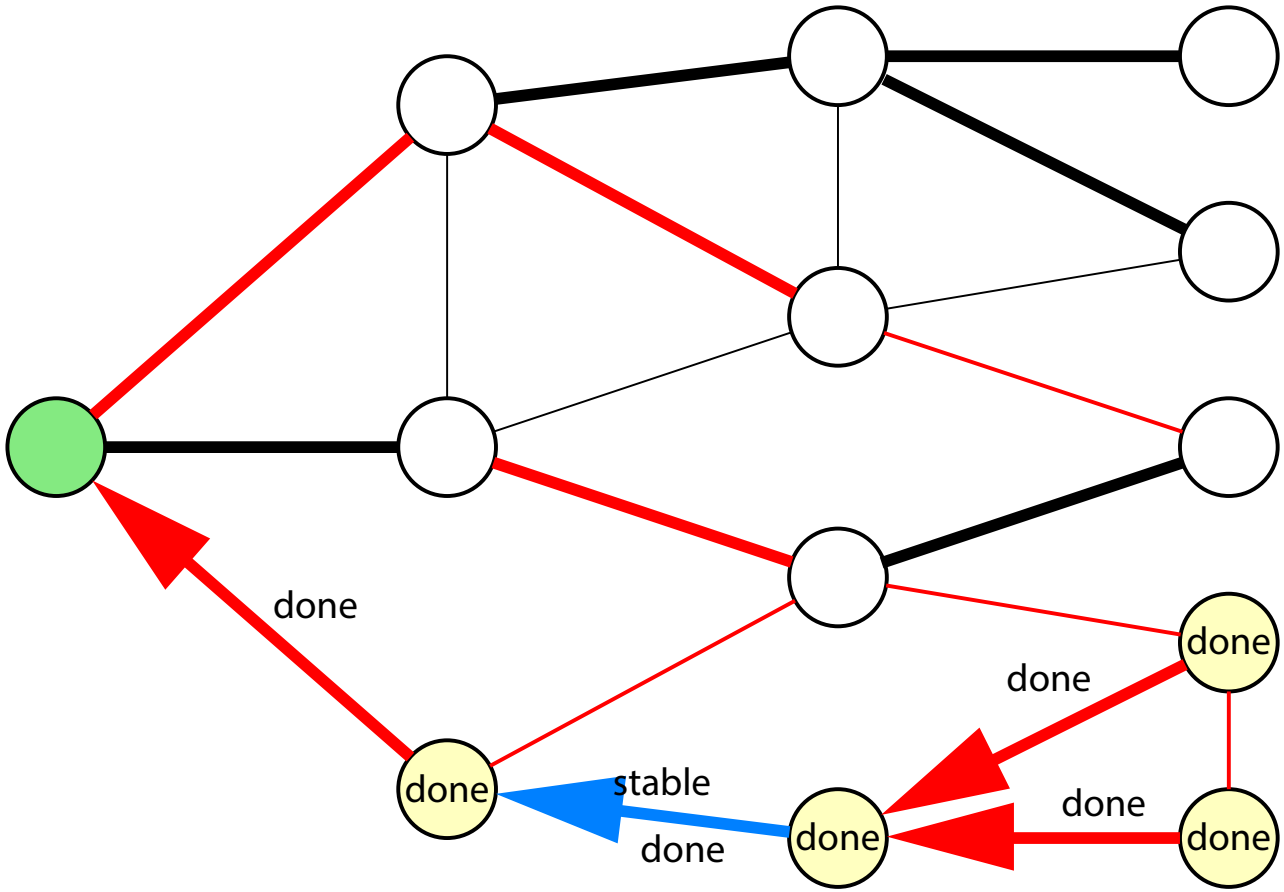
If all the children of a "done" agent (in the spanning tree) are "done", the agent forwards a "done" message to its parent.

Application: Stability Detection



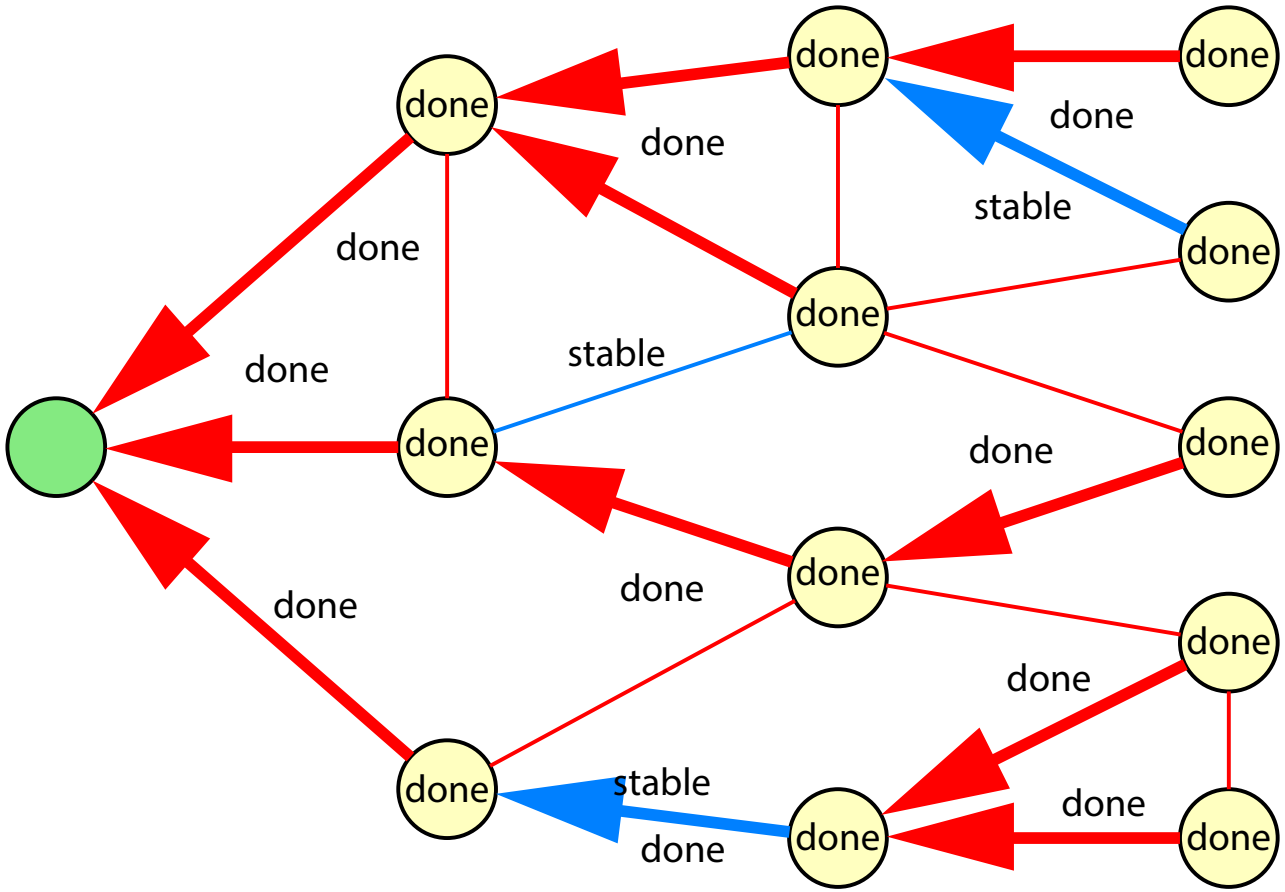
If all the children of a "done" agent (in the spanning tree) are "done", the agent forwards a "done" message to its parent.

Application: Stability Detection



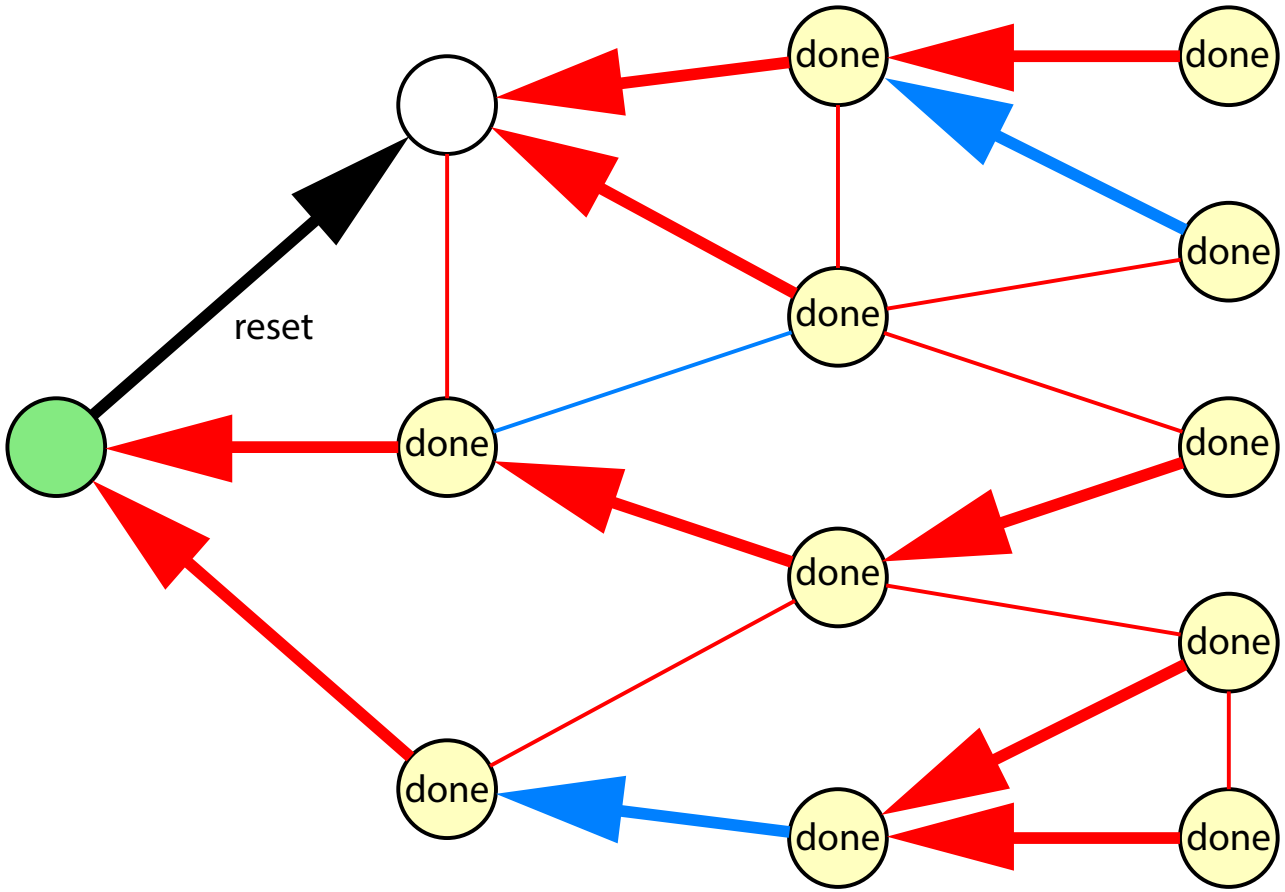
If all the children of a "done" agent (in the spanning tree) are "done", the agent forwards a "done" message to its parent.

Application: Stability Detection



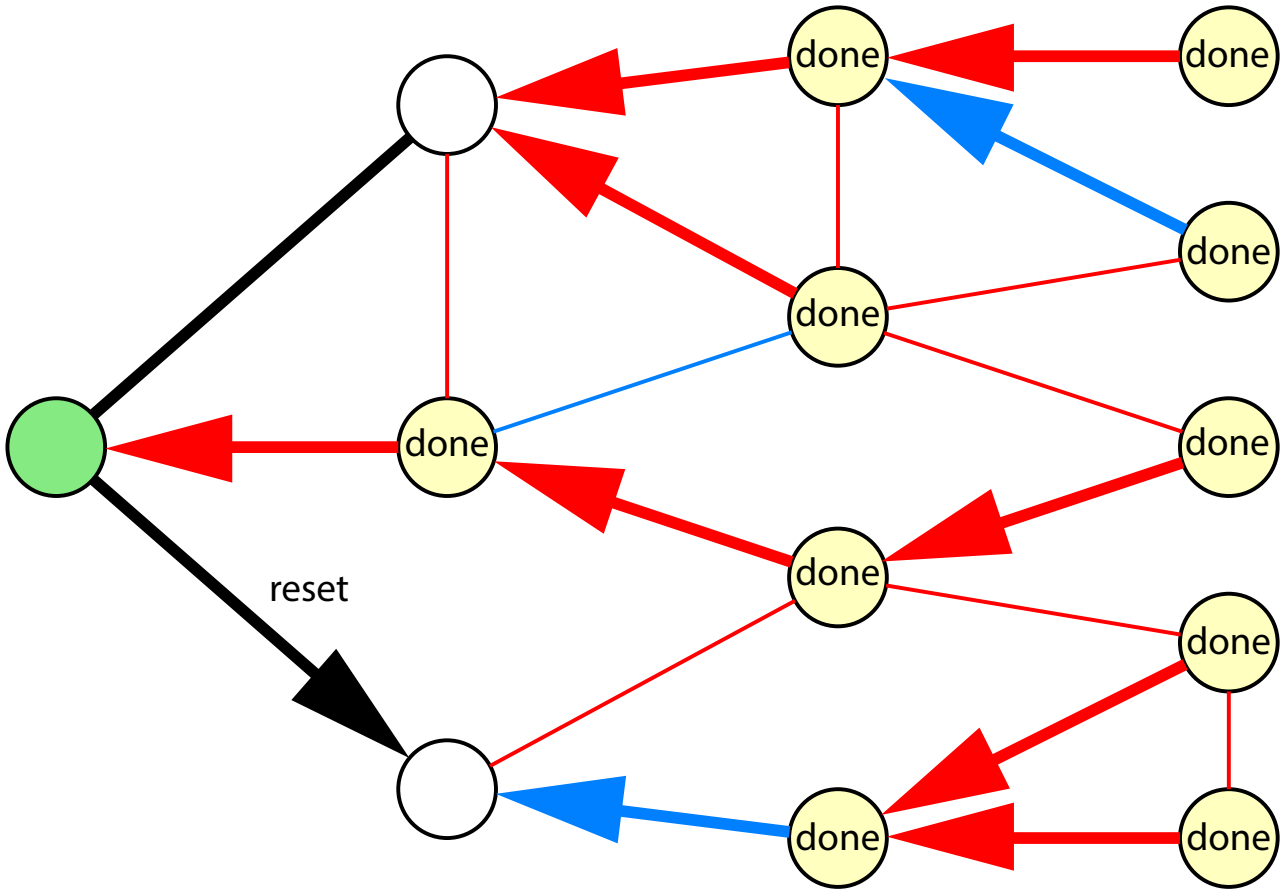
Eventually, the leader receives "done" messages from all its children.

Application: Stability Detection



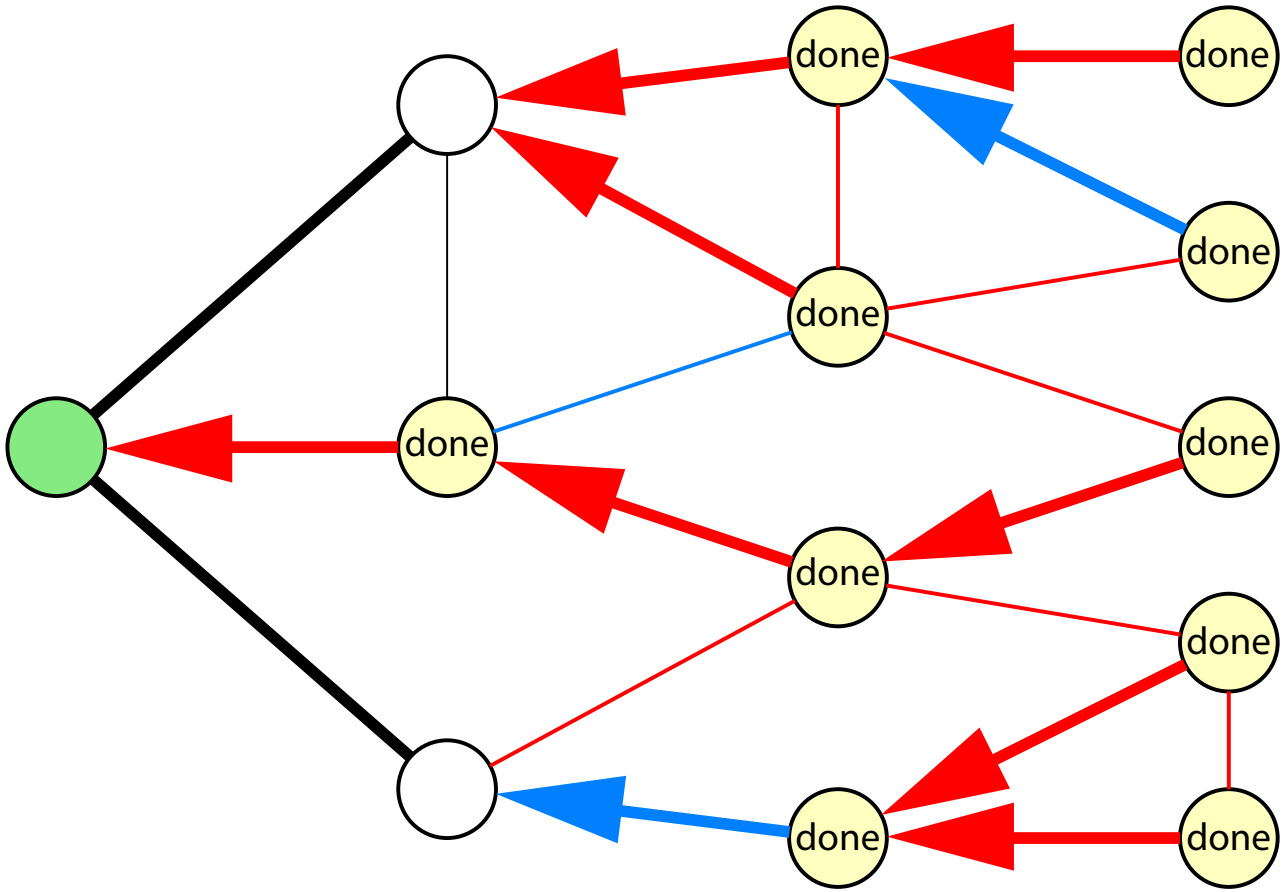
At this point, the leader broadcasts a "reset" message.

Application: Stability Detection



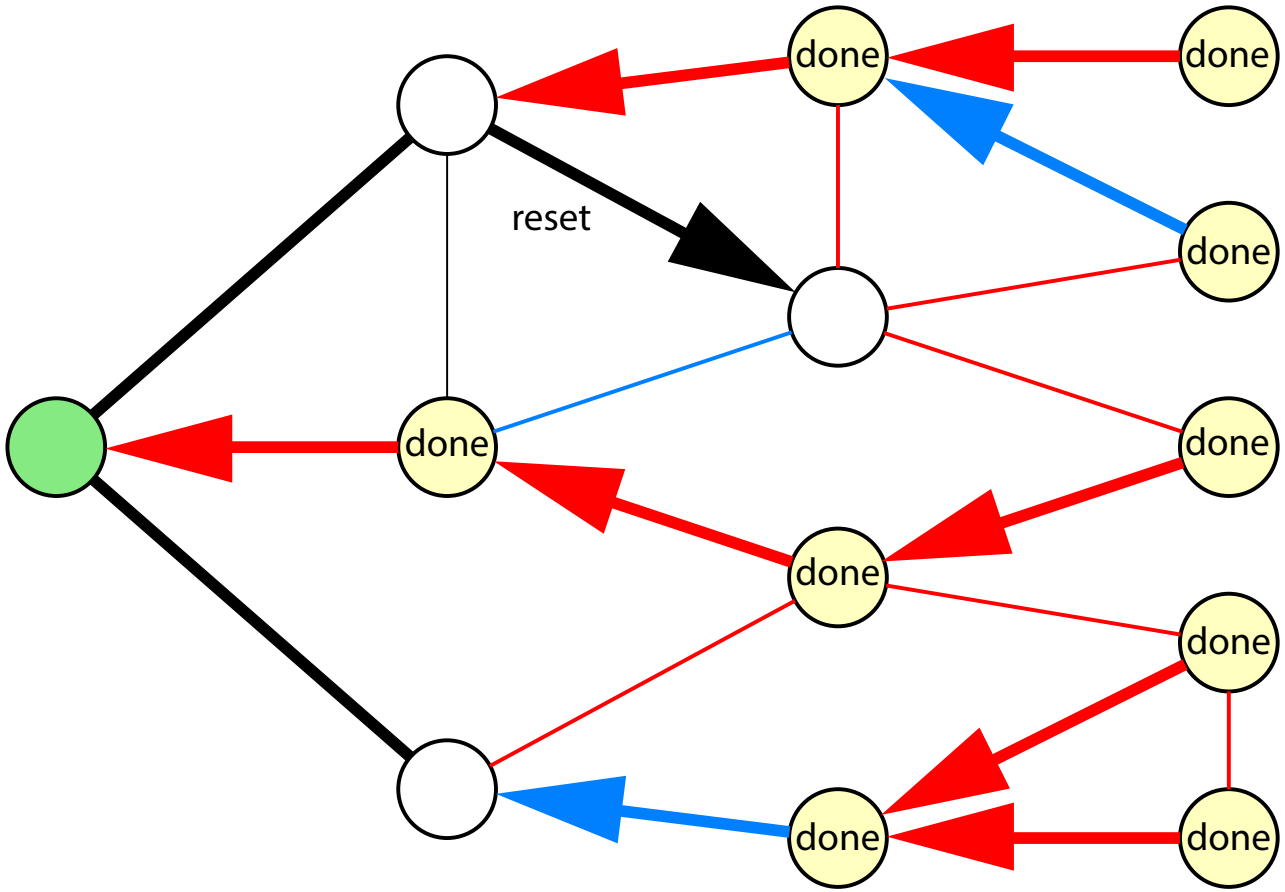
At this point, the leader broadcasts a "reset" message.

Application: Stability Detection



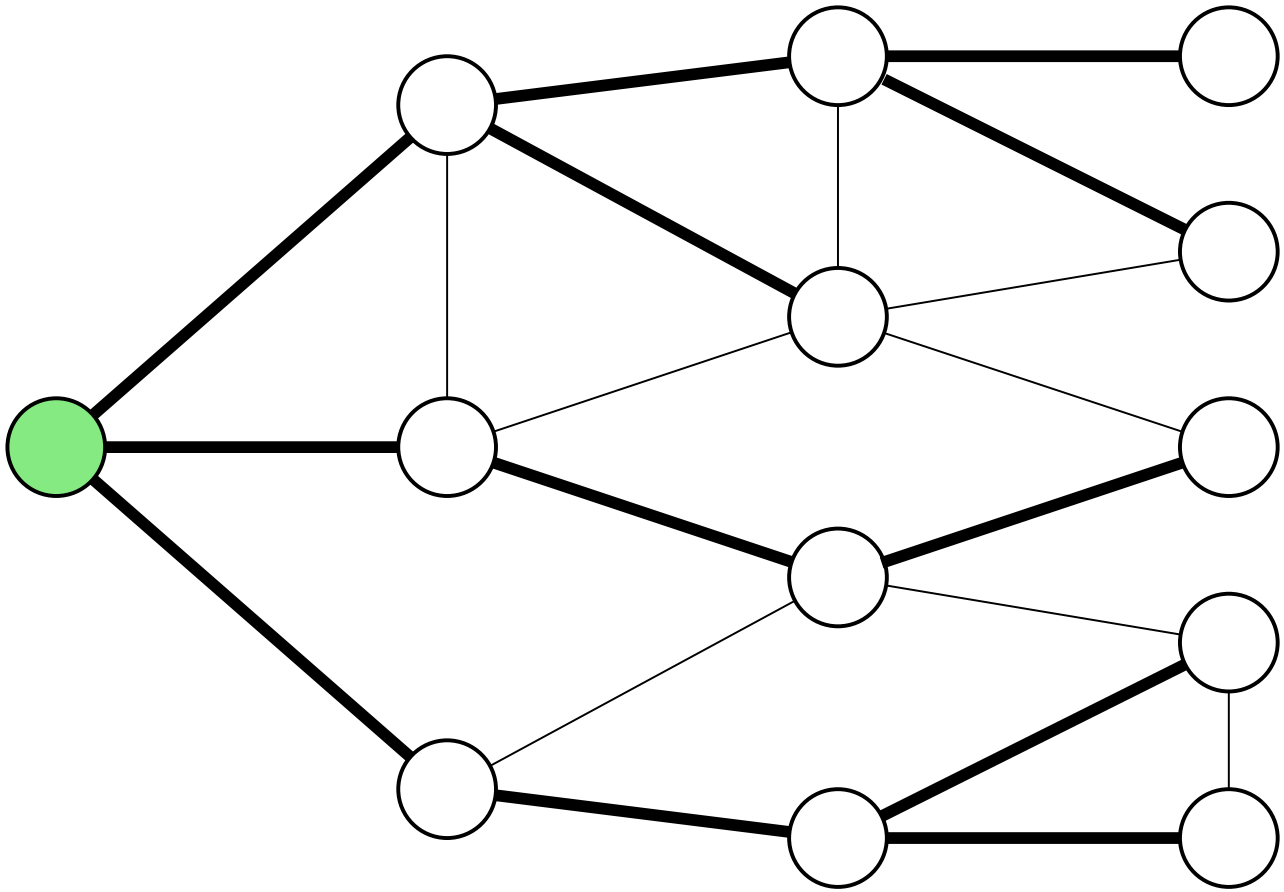
All edges that are incident to a "reset" agent become unmarked.

Application: Stability Detection



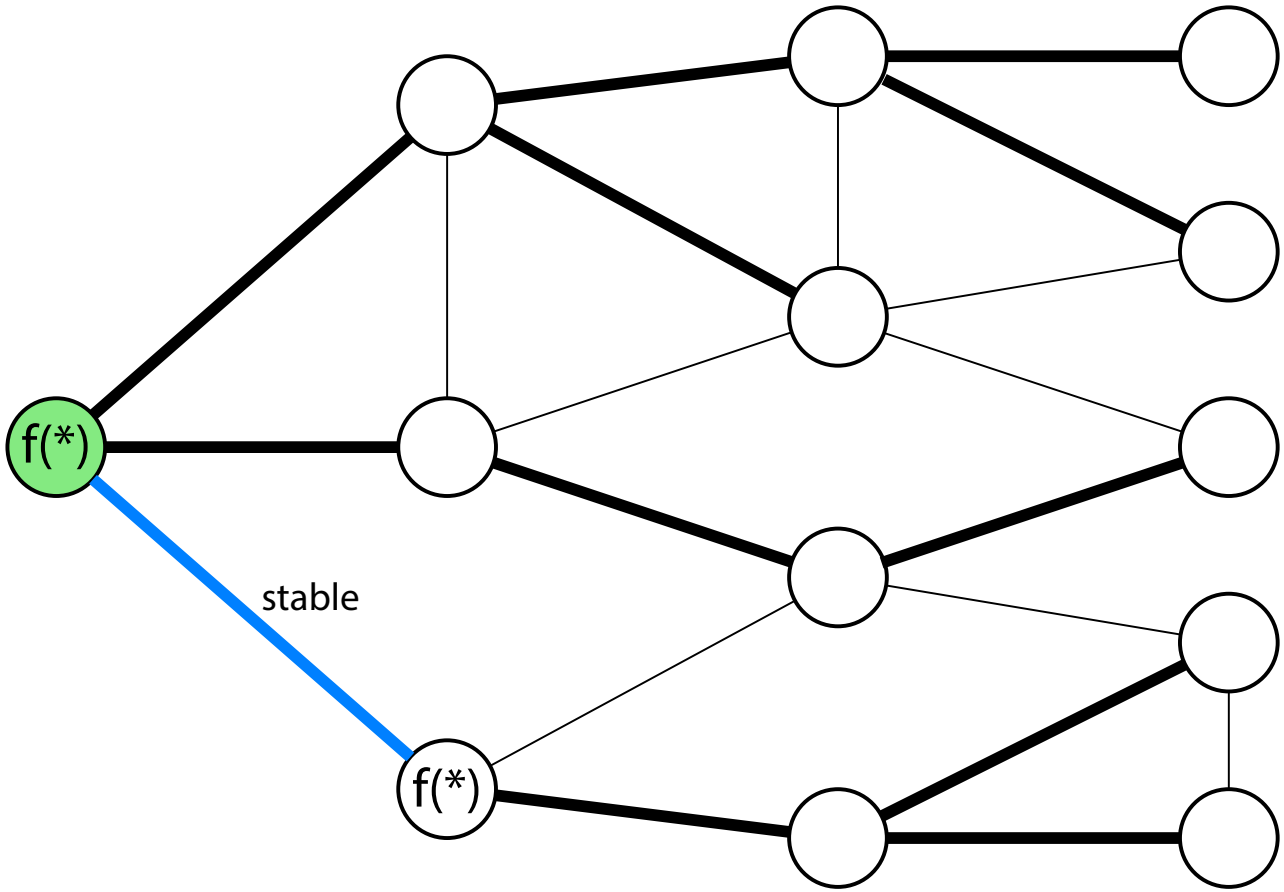
The "reset" message is forwarded along the spanning tree.

Application: Stability Detection



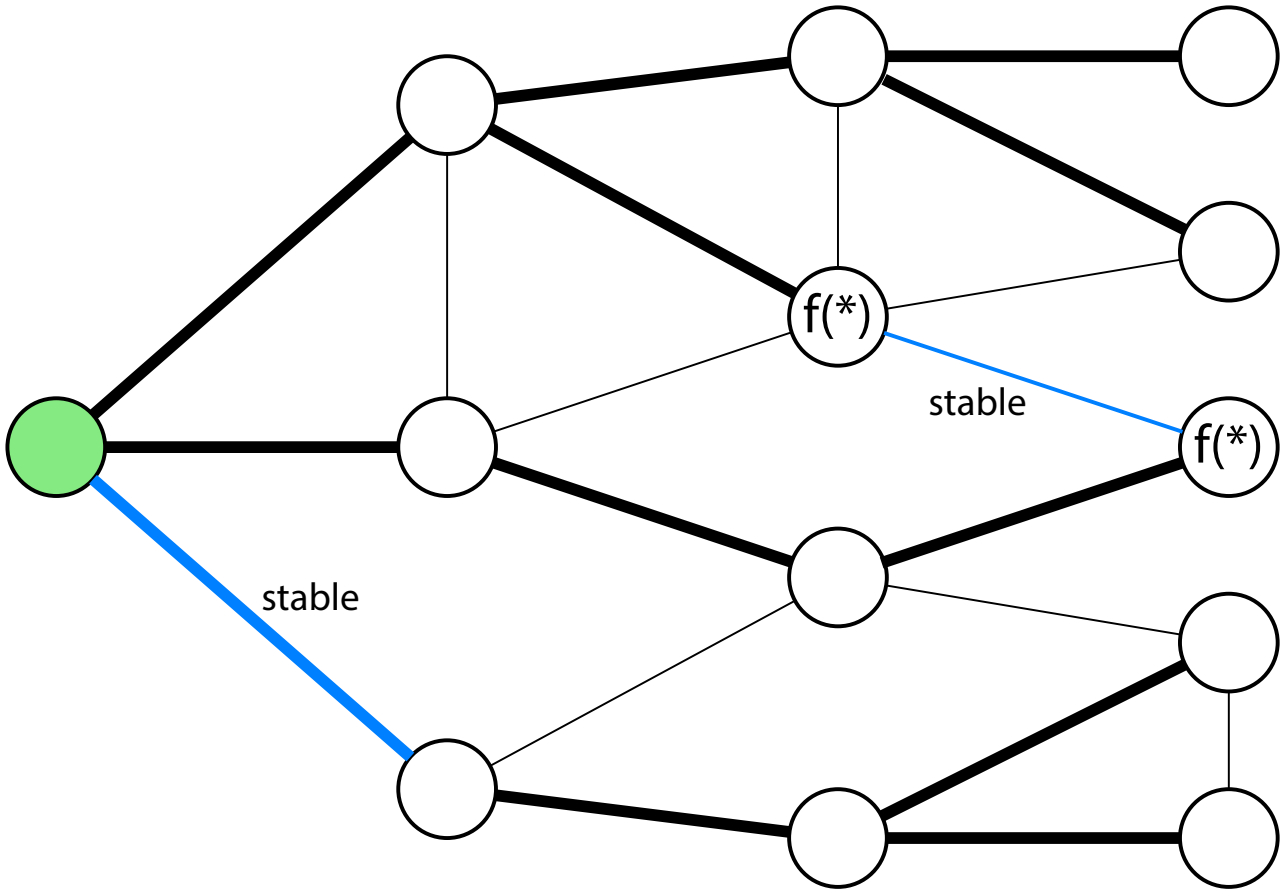
Eventually, the whole network is reset. The leader is notified, and starts a new simulation phase.

Application: Stability Detection



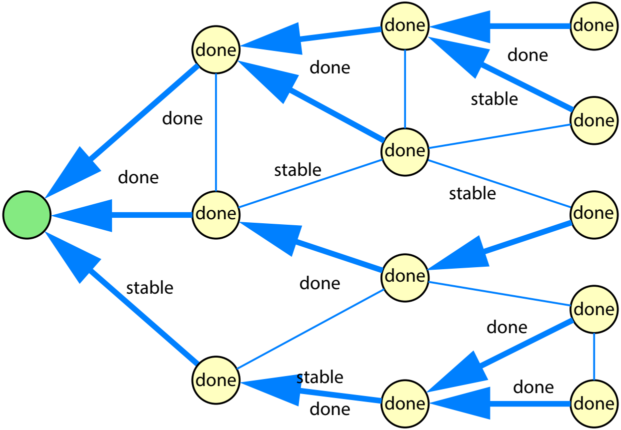
If P is stable, eventually all edges will be marked as "stable".

Application: Stability Detection



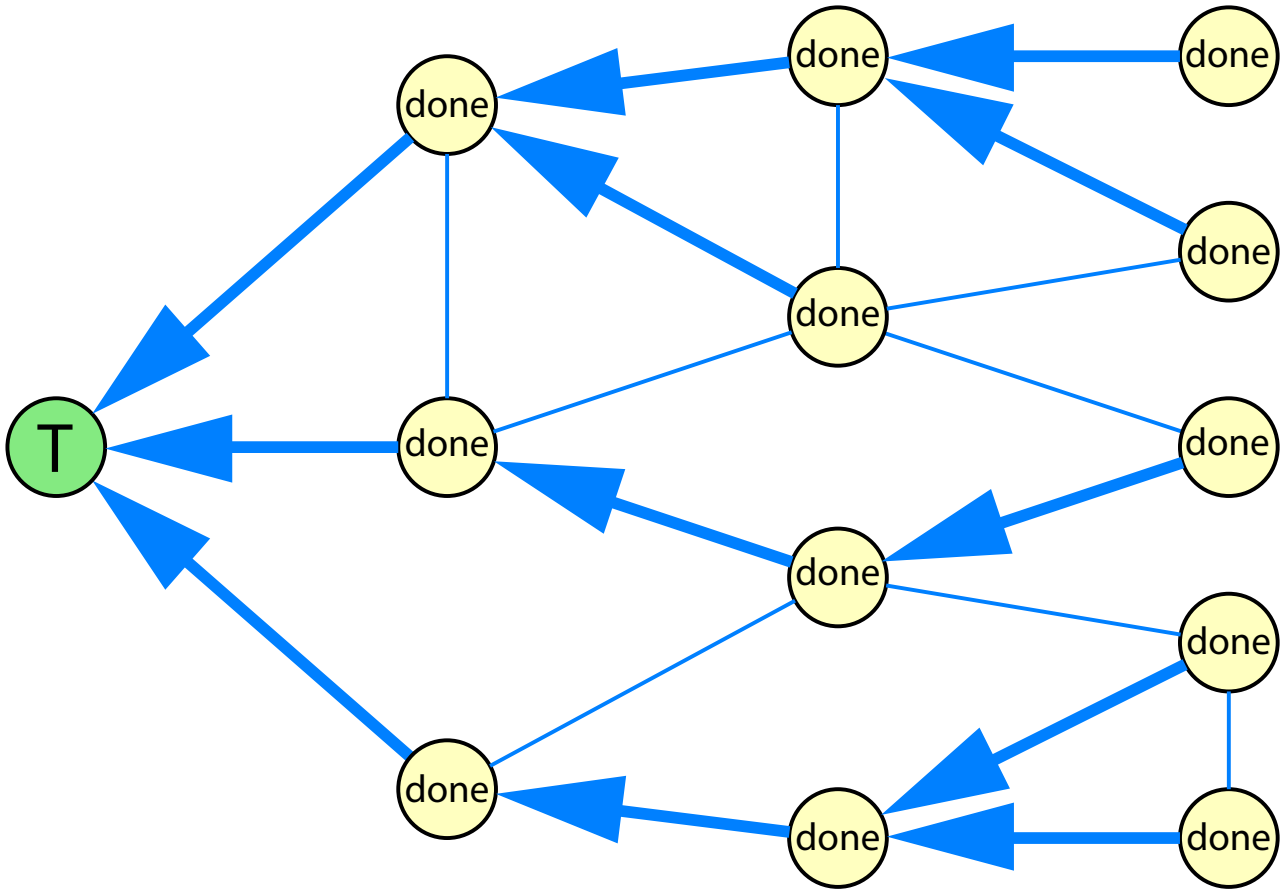
If P is stable, eventually all edges will be marked as "stable".

Application: Stability Detection



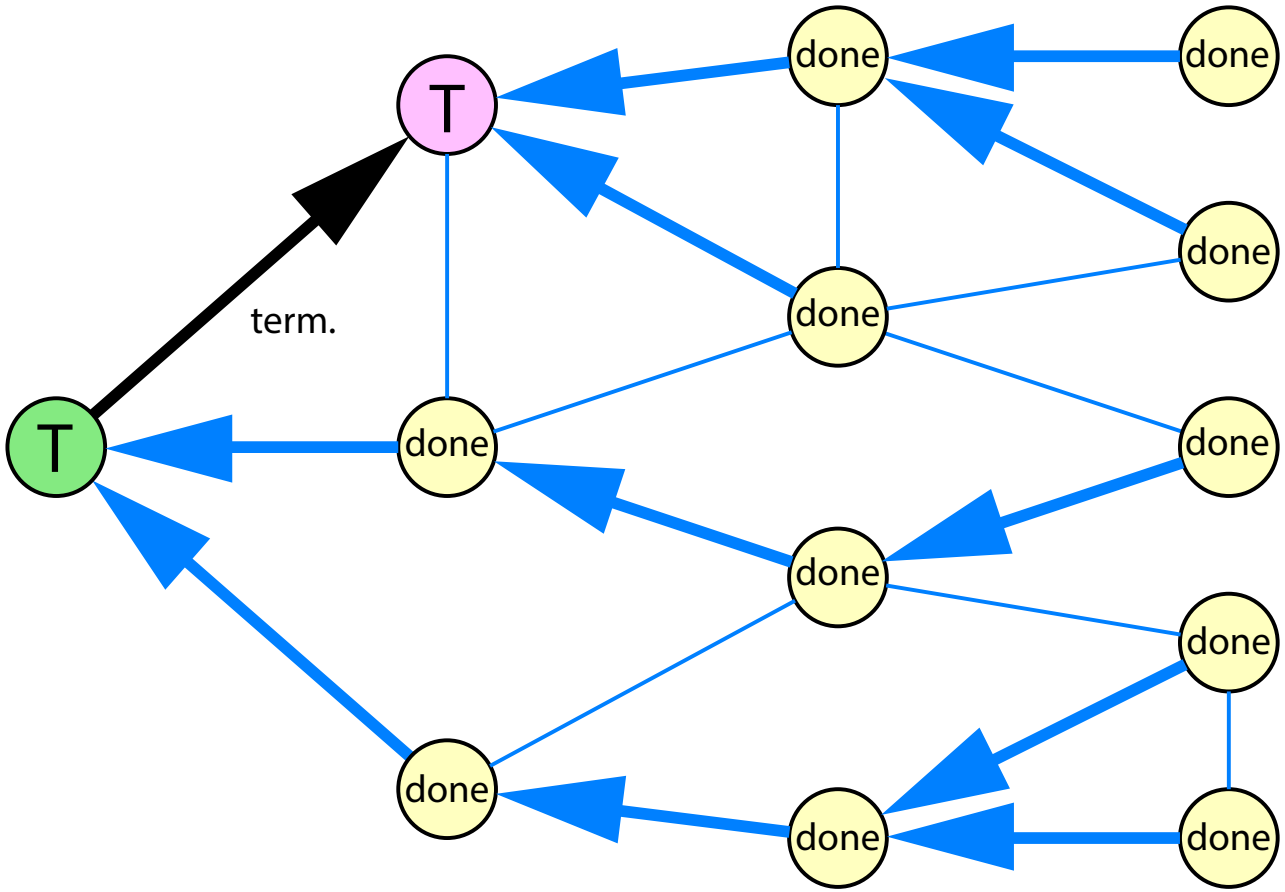
All agents send “done” and “stable” messages to their parents.

Application: Stability Detection



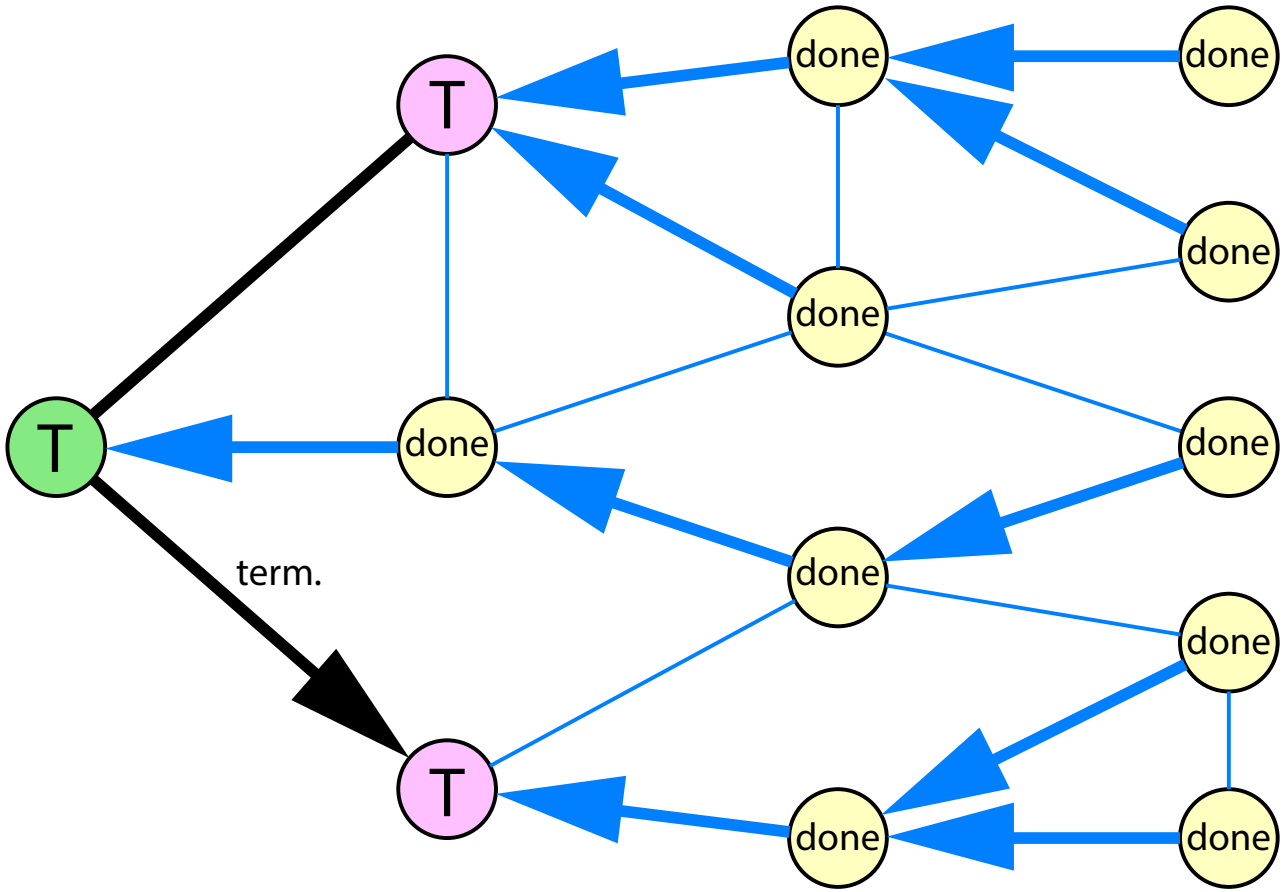
Eventually, the leader receives “done” and “stable” messages from all its children.

Application: Stability Detection



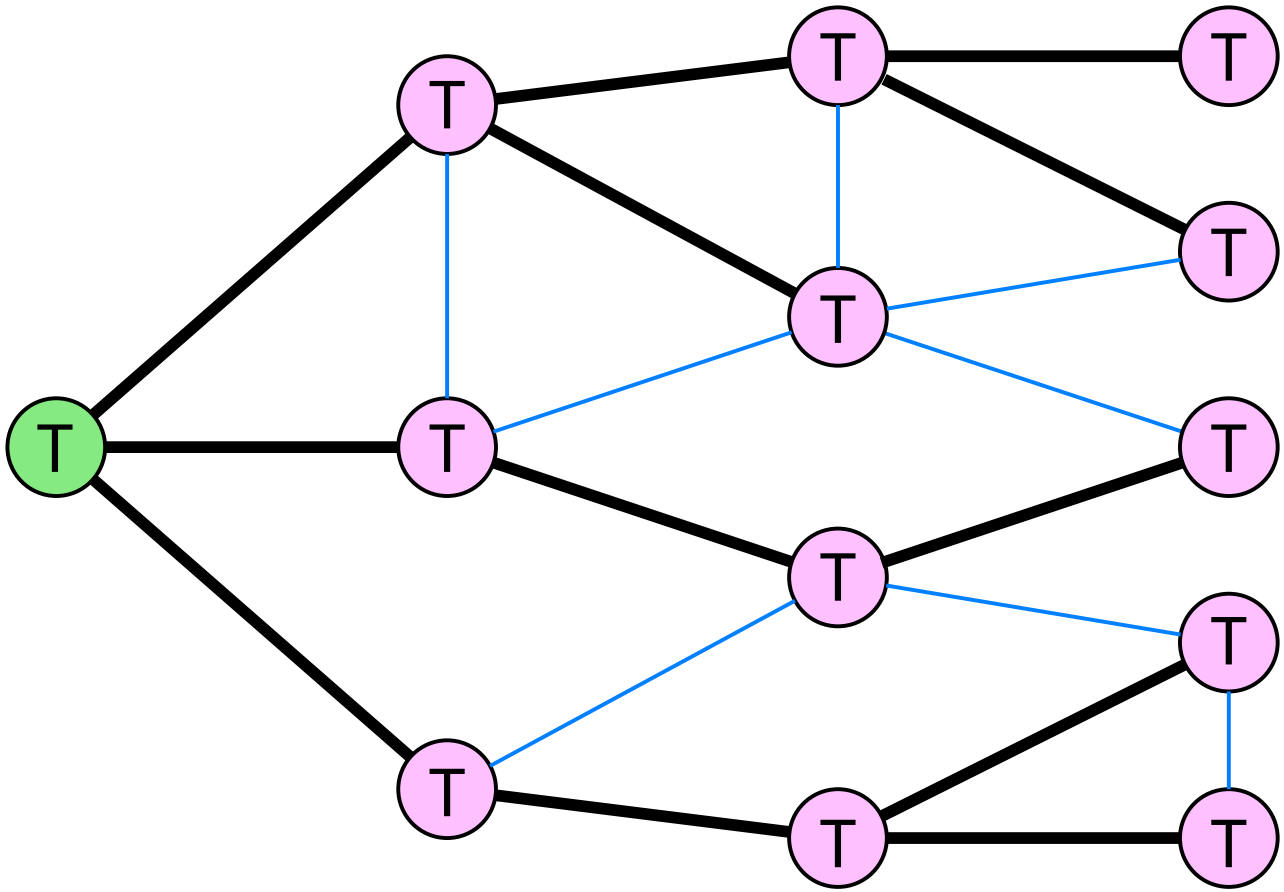
The leader then terminates and broadcasts a "terminate" message, which is forwarded along the spanning tree.

Application: Stability Detection



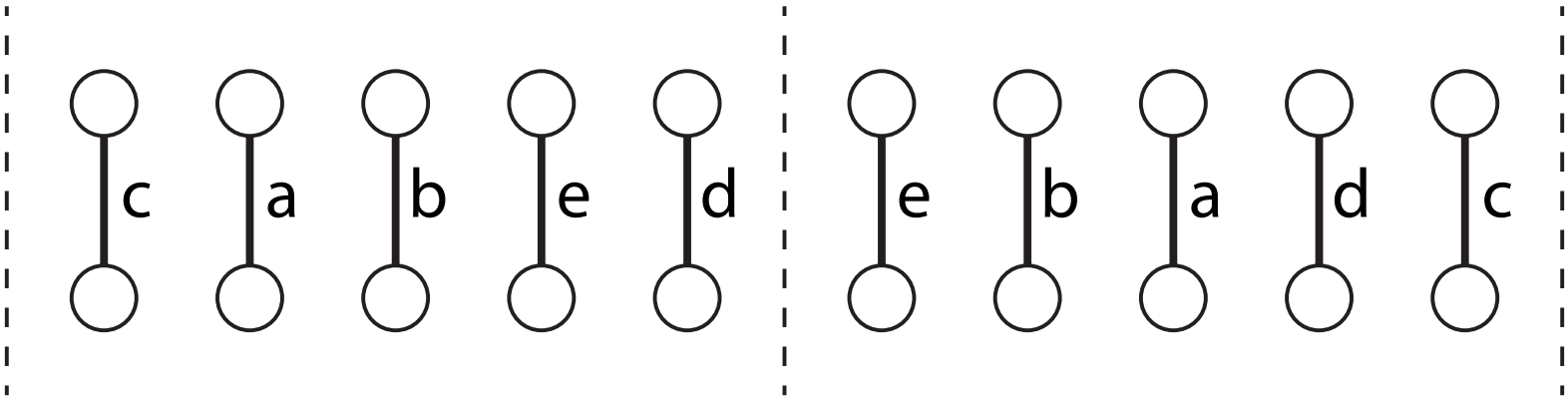
The leader then terminates and broadcasts a "terminate" message, which is forwarded along the spanning tree.

Application: Stability Detection



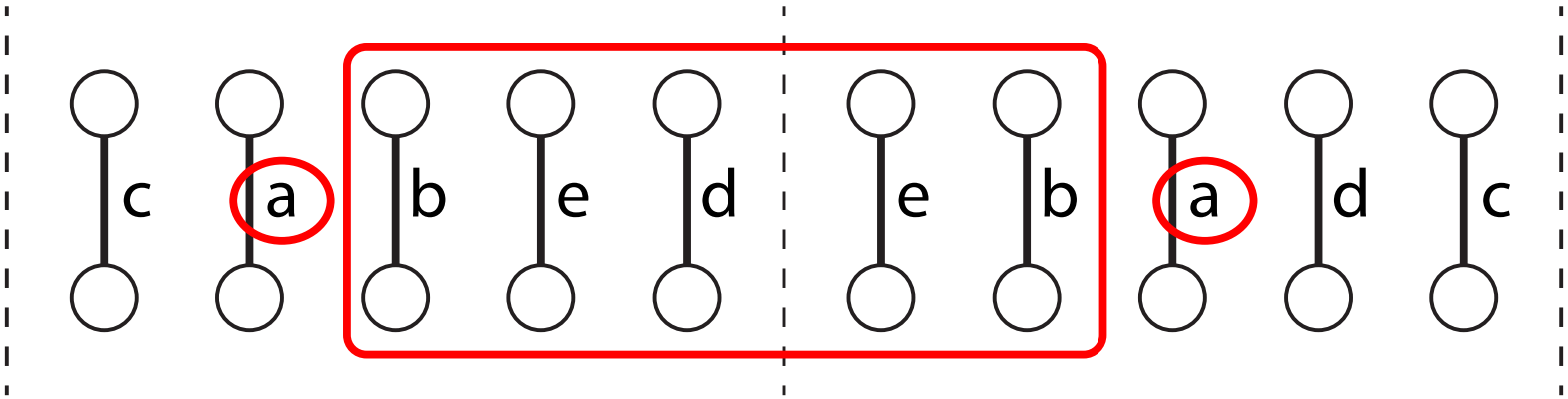
This converts the stable protocol P into a terminating one.

Application: Simulation of 2-Bounded Schedulers



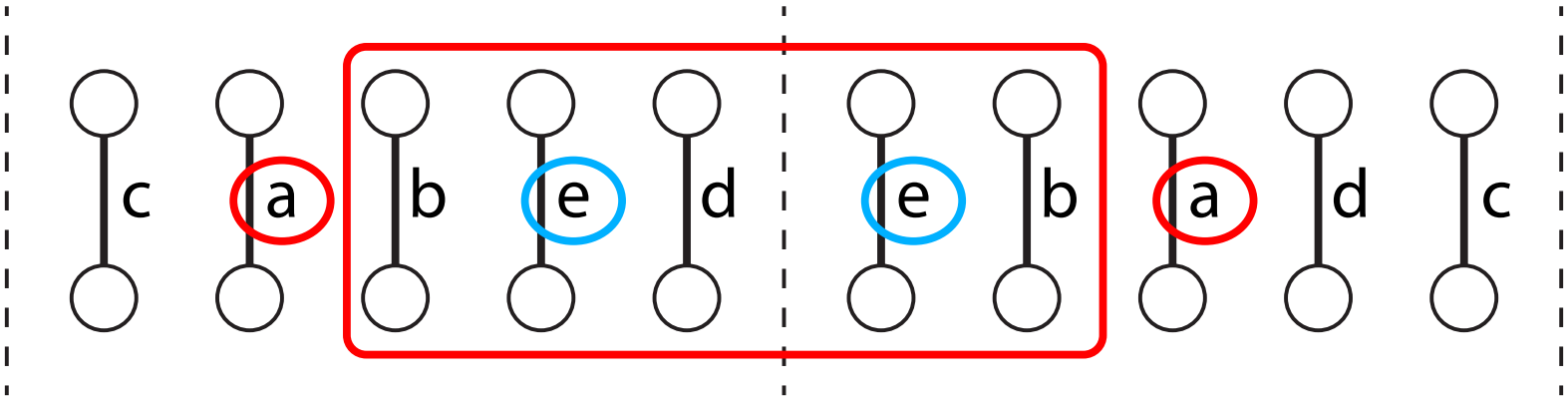
Our simulated schedule activates all edges of the network in some order, then it activates them again in some other order, etc.

Application: Simulation of 2-Bounded Schedulers



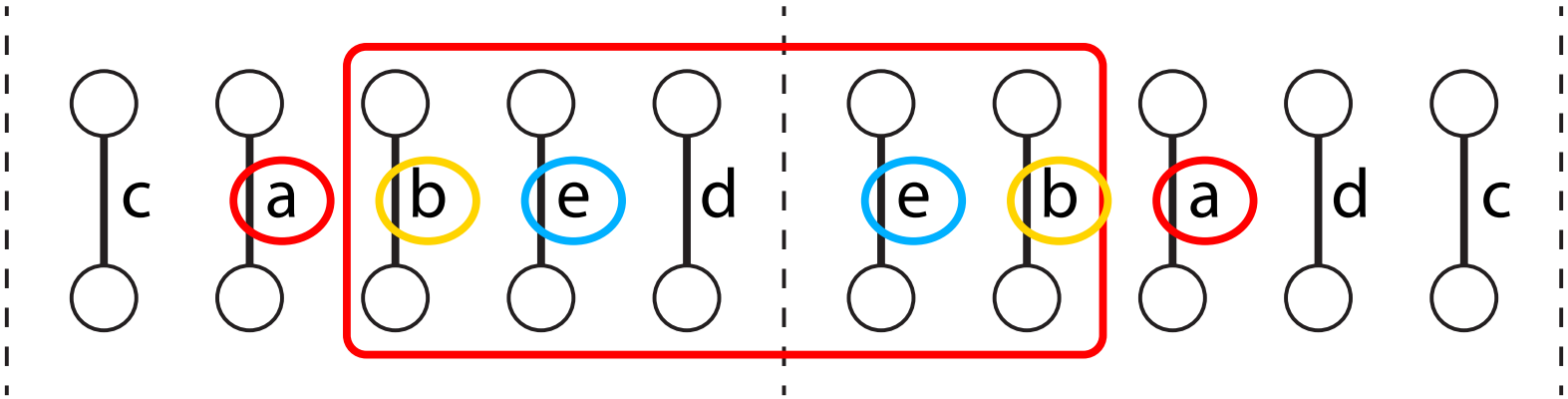
So, between two activations of an edge (say, a), each other edge is activated at most twice.

Application: Simulation of 2-Bounded Schedulers



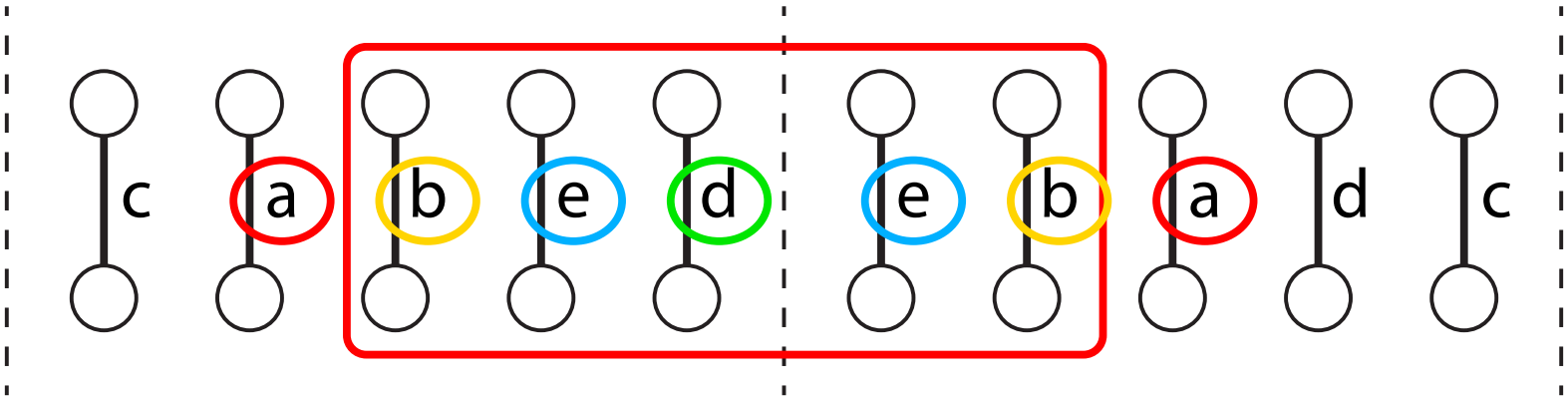
So, between two activations of an edge (say, a), each other edge is activated at most twice.

Application: Simulation of 2-Bounded Schedulers



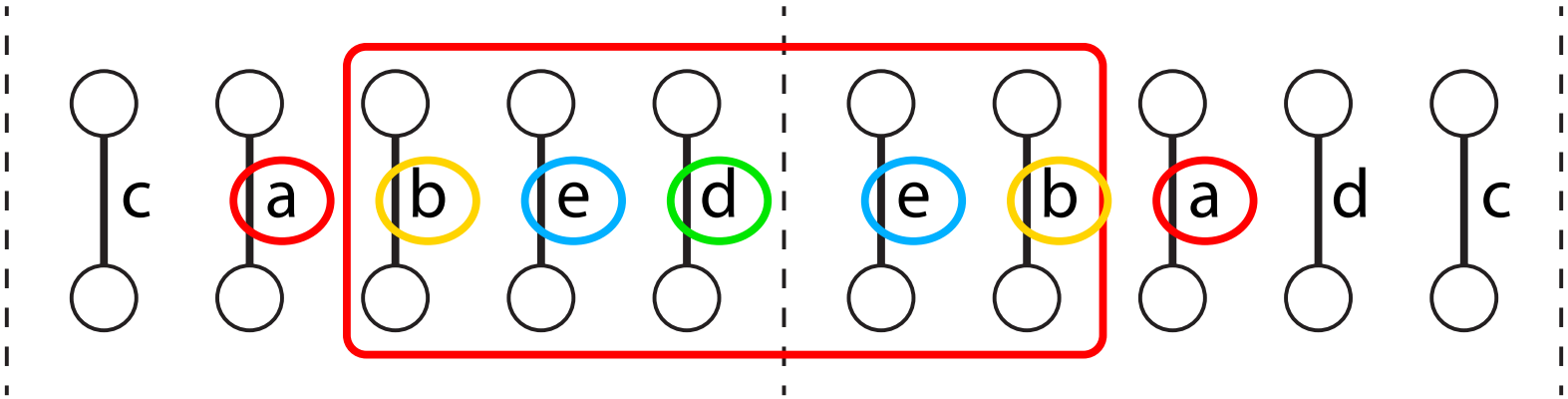
So, between two activations of an edge (say, a), each other edge is activated at most twice.

Application: Simulation of 2-Bounded Schedulers



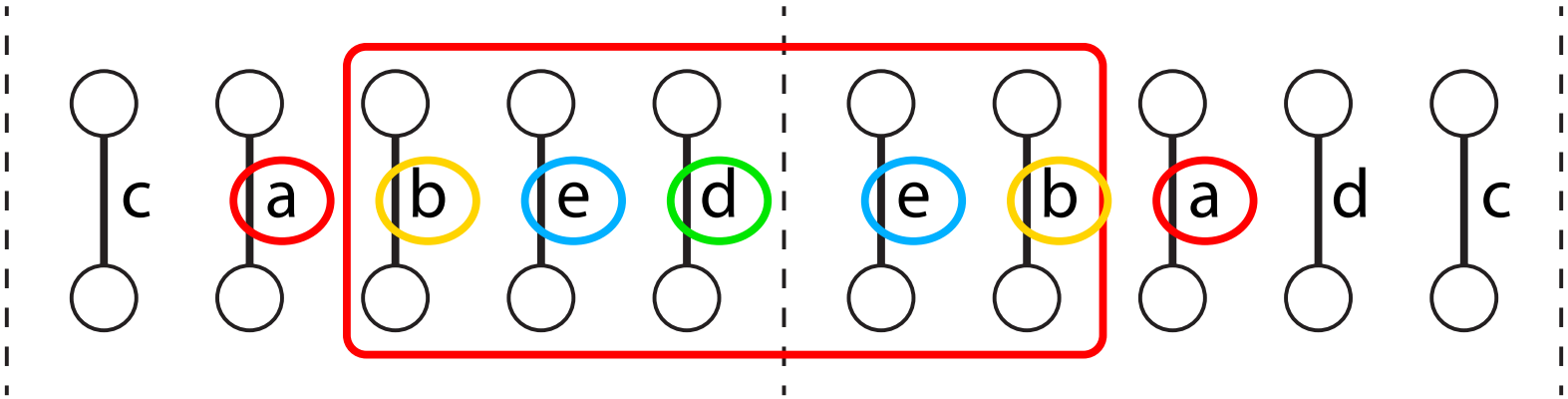
It follows that our simulated schedule is 2-bounded.

Application: Simulation of 2-Bounded Schedulers



So, the protocols that work under the 2-bounded scheduler also work under all k-bounded schedulers, for all $k > 2$.

Application: Simulation of 2-Bounded Schedulers



Theorem: in every network where a leader can be elected, the k-bounded schedulers are all equivalent, for $k > 1$.