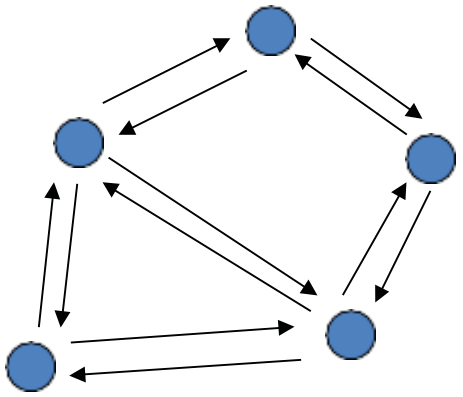
Asynchronous Distributed Algorithms

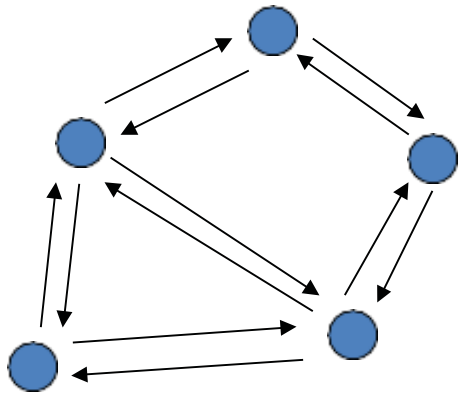


Asynchronous Network Model

- Complications so far:
  - Processes act concurrently
  - A little nondeterminism.
- Now things get much worse:
  - No rounds---process steps and message deliveries happen at arbitrary times, in arbitrary orders.
  - Processes get out of synch.
  - Much more nondeterminism.

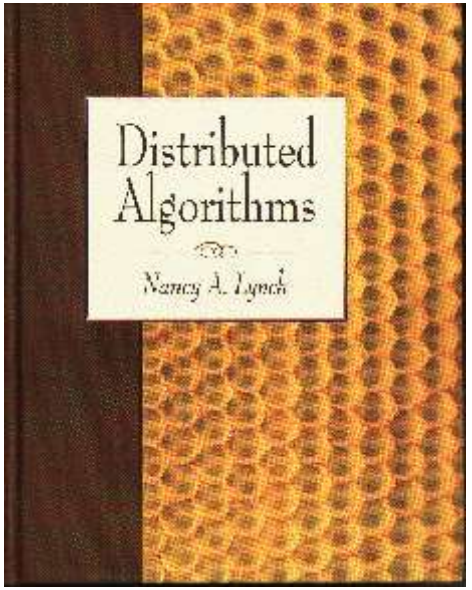
synchronous nw model

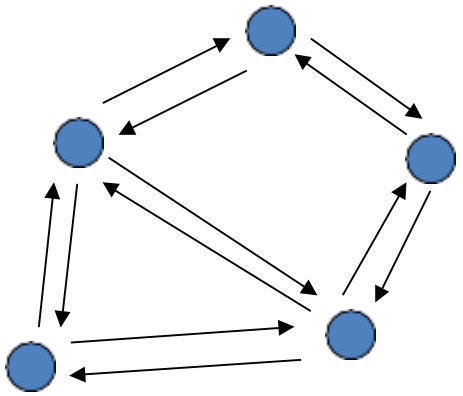
- Understanding asynchronous distributed algorithms is hard because we can't understand exactly how they execute.
- Instead, we must understand abstract properties of executions.



Aynchronous Network Model

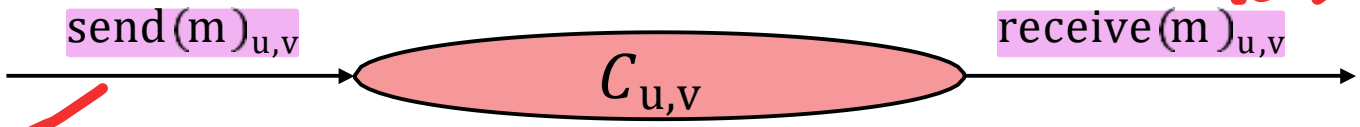
- Lynch, Distributed Algorithms, Chapter 8.
- Processes at nodes of an undirected graph $G = (V, E)$, communicate using messages.
- Communication channels associated with edges (one in each direction on each edge).
  - $C_{u,v}$, channel from vertex n to vertex v.
- Each process has output ports and input ports that connect it to its communication channels.
- Processes need not be distinguishable.





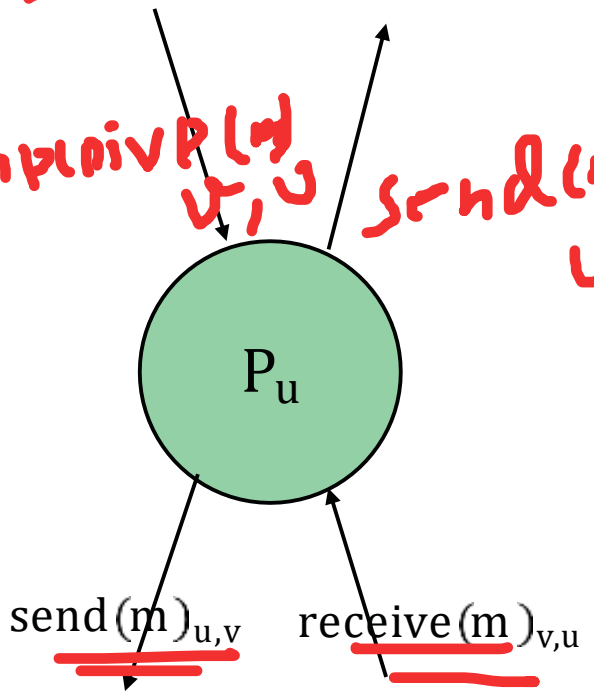
Channel Automaton $C_{u,v}$

- Formally, an input/output automaton.
- Input actions: $\text{send}(m)_{u,v}$
- Output actions: $\text{receive}(m)_{u,v}$
- State variable:
  - $mqnene$, a FIFO queue, initially empty.
- Transitions:
  - $\text{send}(m)_{u,v}$
    - Effect: add m to $mqnene$.
  - $\text{receive}(m)_{u,v}$
    - Precondition: m = head($mqnene$)
    - Effect: remove head of $mqnene$



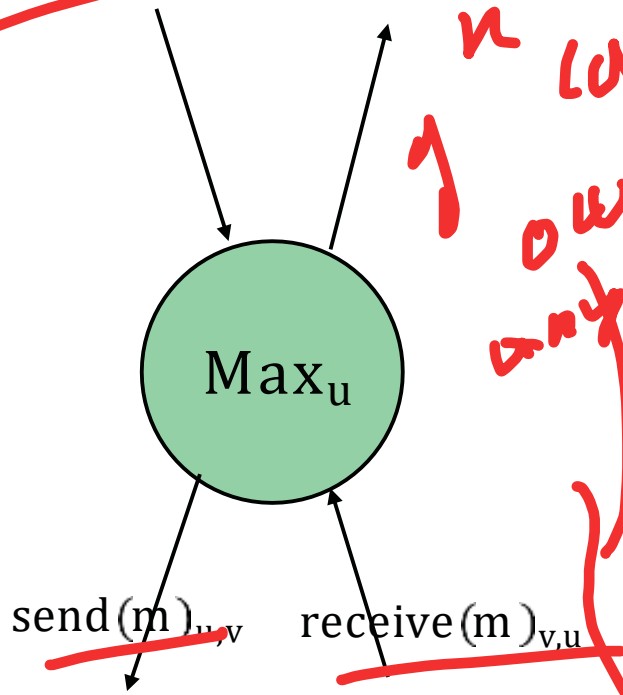
Process Automaton P_u

- Associate a process automaton with each vertex of G.
- To simplify notation, let P_u denote the process automaton at vertex n.
  - But the process does not "know" n.
- P_u has $send(m)_{u,v}$ outputs and $receive(m)_{v,u}$ inputs.
- May also have external inputs and outputs.
- Has state variables.
- Keeps taking steps (eventually).



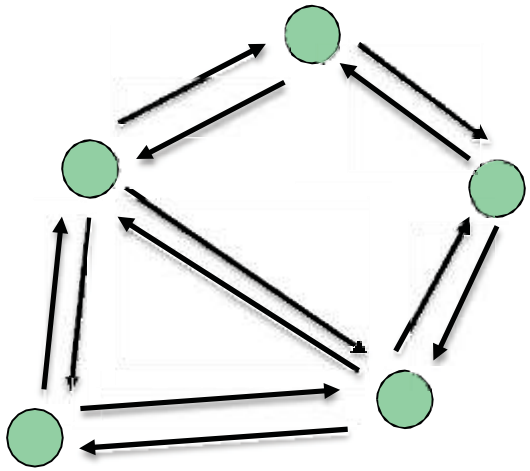
Example: Max_u Process Automaton

- Input actions: $\mathrm{receive}(m)_{v,u}$
- Output actions: $\mathrm{send}(m)_{u,v}$
- State variables:
  - max, a natural number, initially x_u
  - For each neighbor v:
    - $send(v)$, a Boolean, initially $true$
- Transitions:
  - $\mathrm{receive}(m)_{v,u}$
    - Effect: if $m > max$ then
      - $max := m$
      - for every w, $send(w) := true$
  - $\mathrm{send}(m)_{u,v}$
    - Precondition: $send(v) = true$ and $m = max$
    - Effect: $send(v) := false$



Combining Processes and Channels

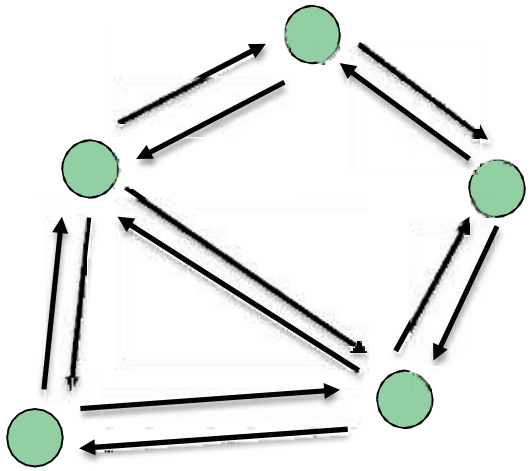
- Undirected graph $G = (V, E)$.
- Process P_u at each vertex n.
- Channels $C_{u,v}$ and $C_{v,u}$, associated with each edge $\{n, v\}$.
- $send(m)_{u,v}$ output of process P_u gets identified with $send(m)_{u,v}$ input of channel $C_{u,v}$.
- $receive(m)_{v,u}$ output of channel $C_{v,u}$ gets identified with $receive(m)_{v,u}$ input of process P_u.
- Steps involving such a shared action involve simultaneous state transitions for a process and a channel.



Execution

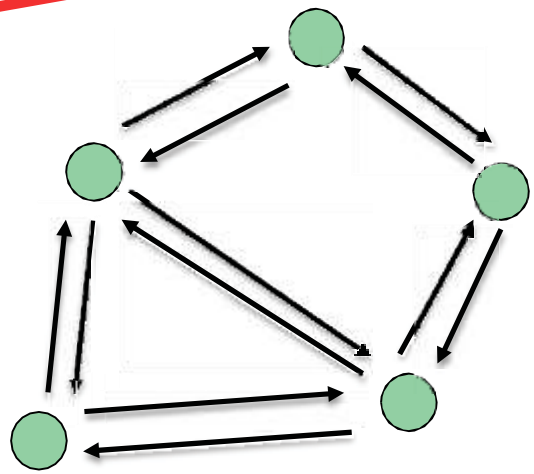
- No synchronous rounds anymore.
- The system executes by performing enabled steps, one at a time, in any order.
- Formally, an execution is modeled as a sequence of individual steps.
- Different from the synchronous model, in which all processes take steps concurrently at each round.

- Assume enabled steps eventually occur:
  - Each channel always eventually delivers the first message in its queue.
  - Each process always eventually performs some enabled step.

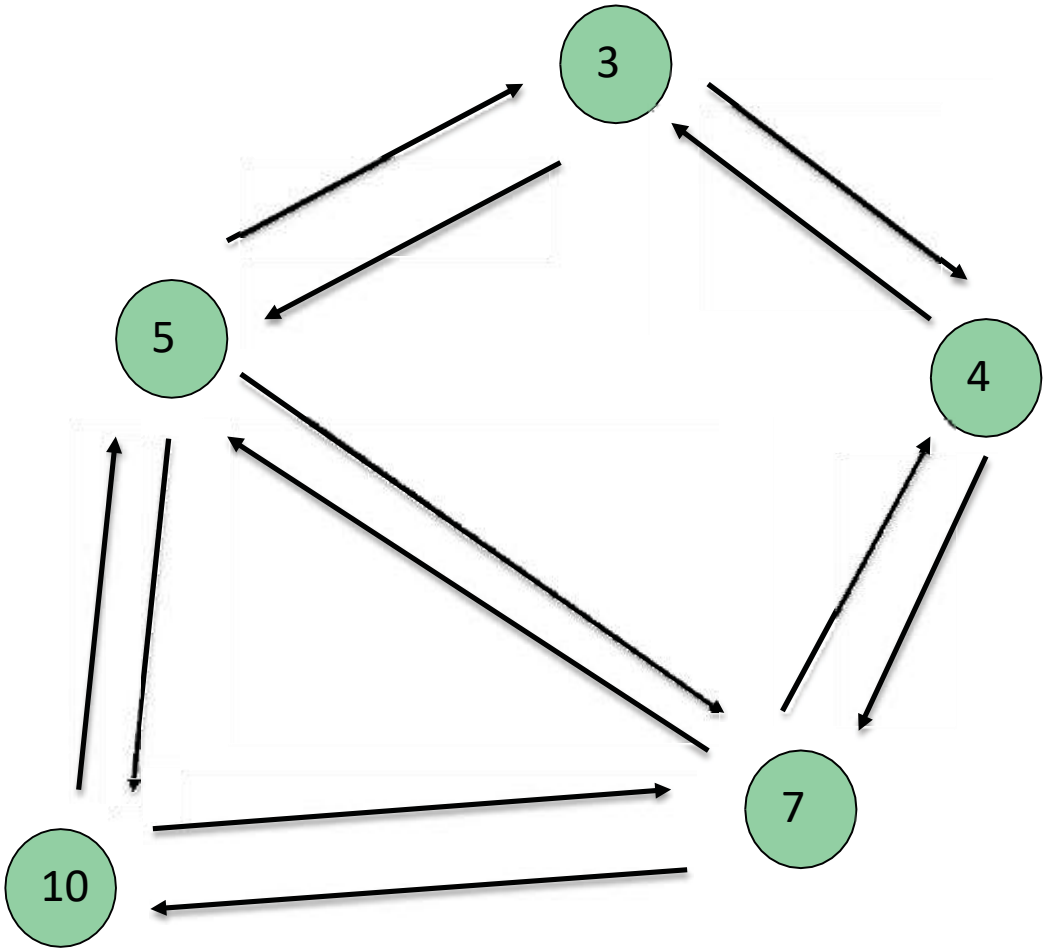


Combining Max Processes and Channels

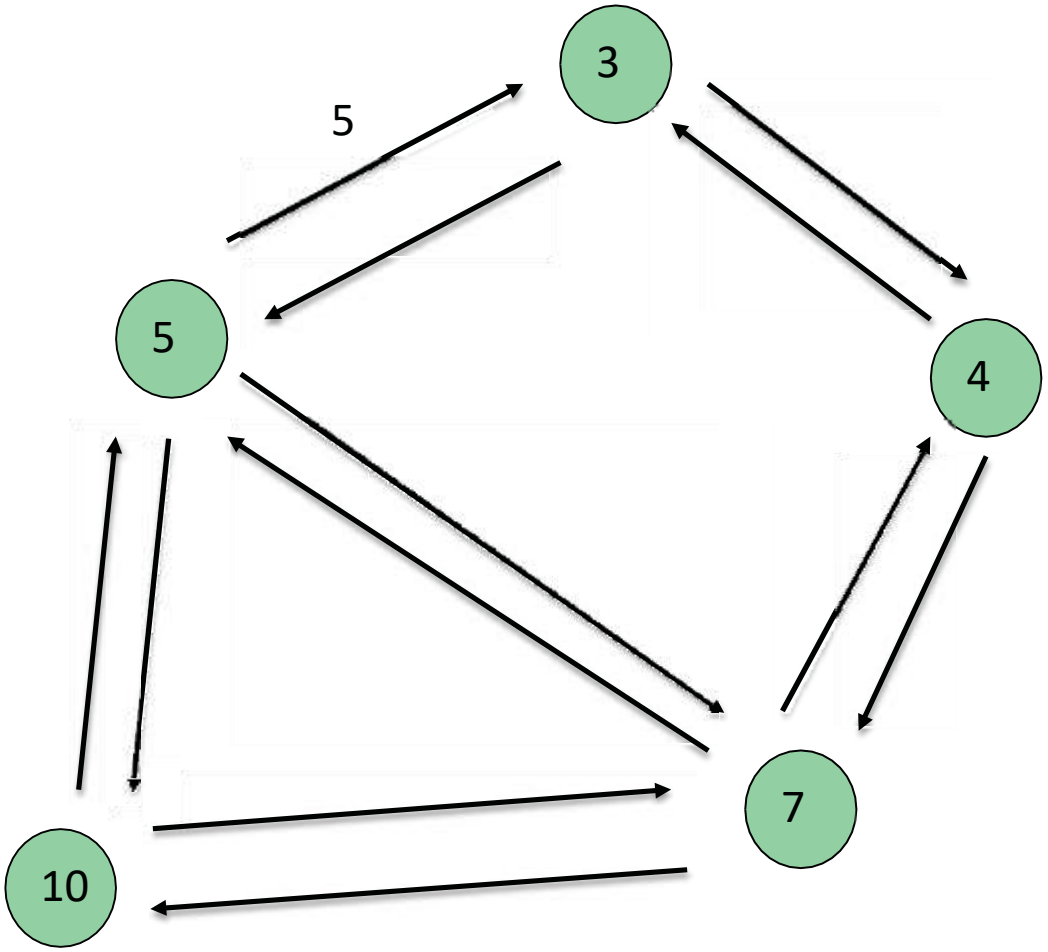
- Each process Max_u starts with an initial value x_u.
- They all send out their initial values, and propagate their $\max$ values, until everyone has the globally-maximum value.
- Sending and receiving steps can happen in many different orders, but in all cases the global max will eventually arrive everywhere.



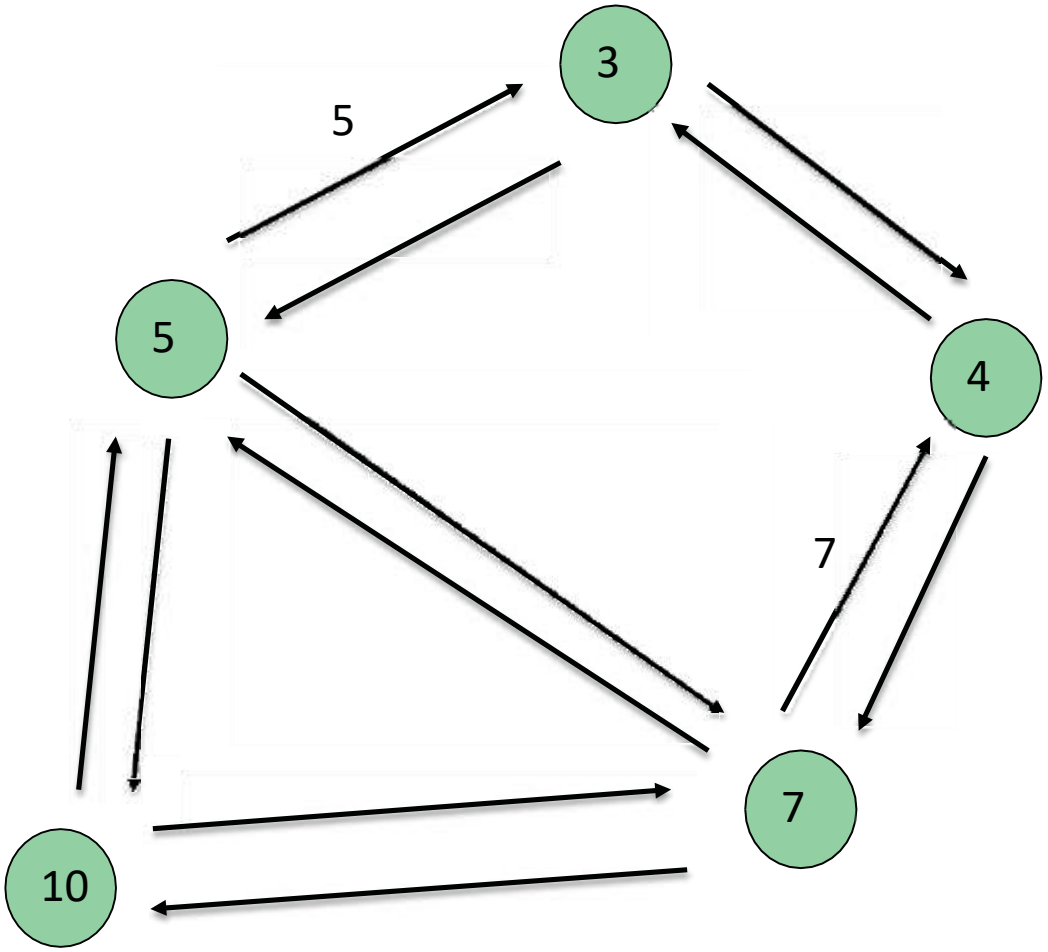
Max System



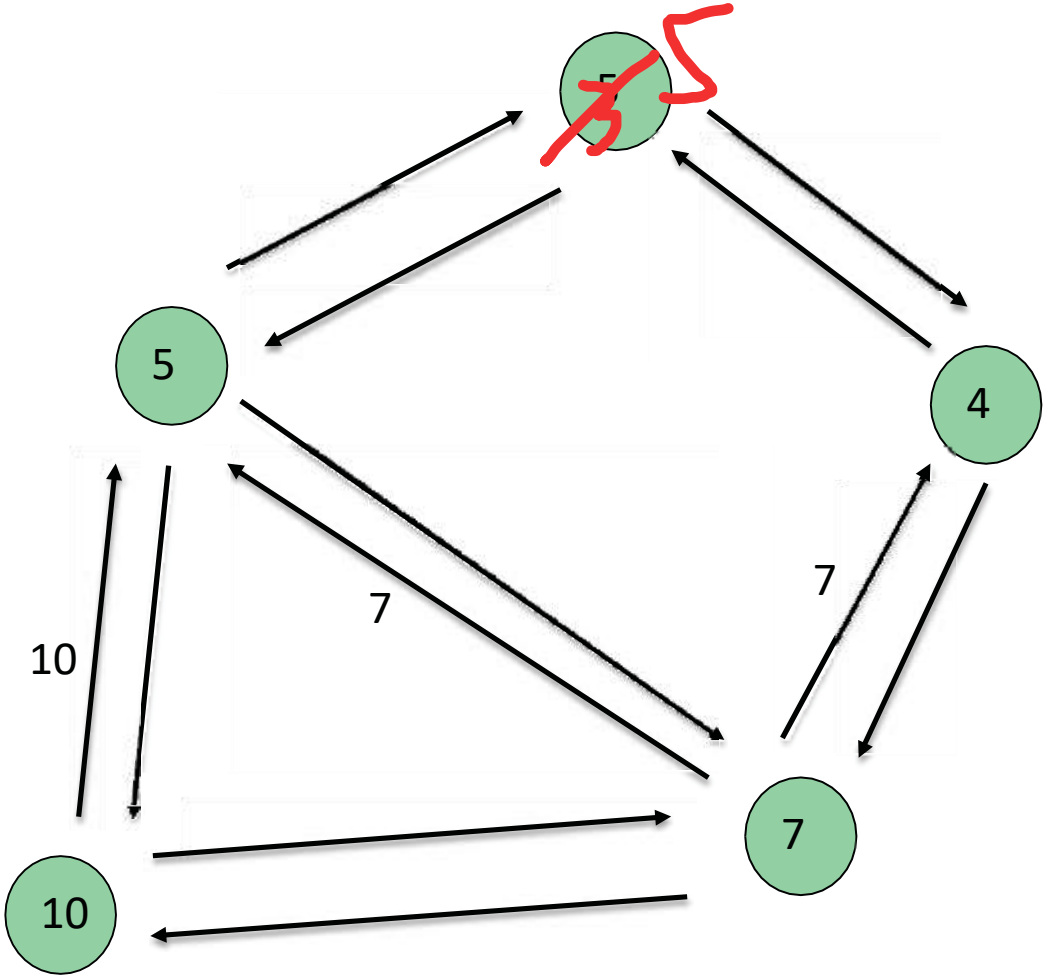
Max System



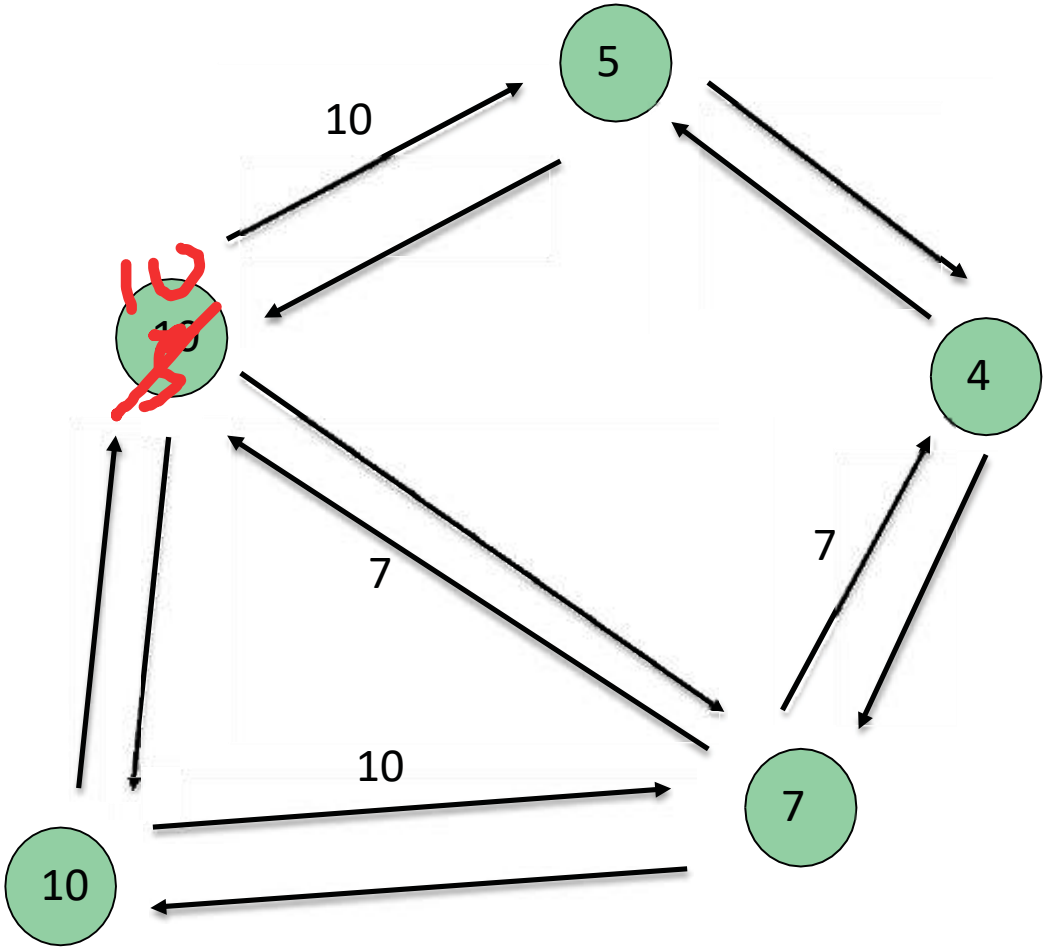
Max System



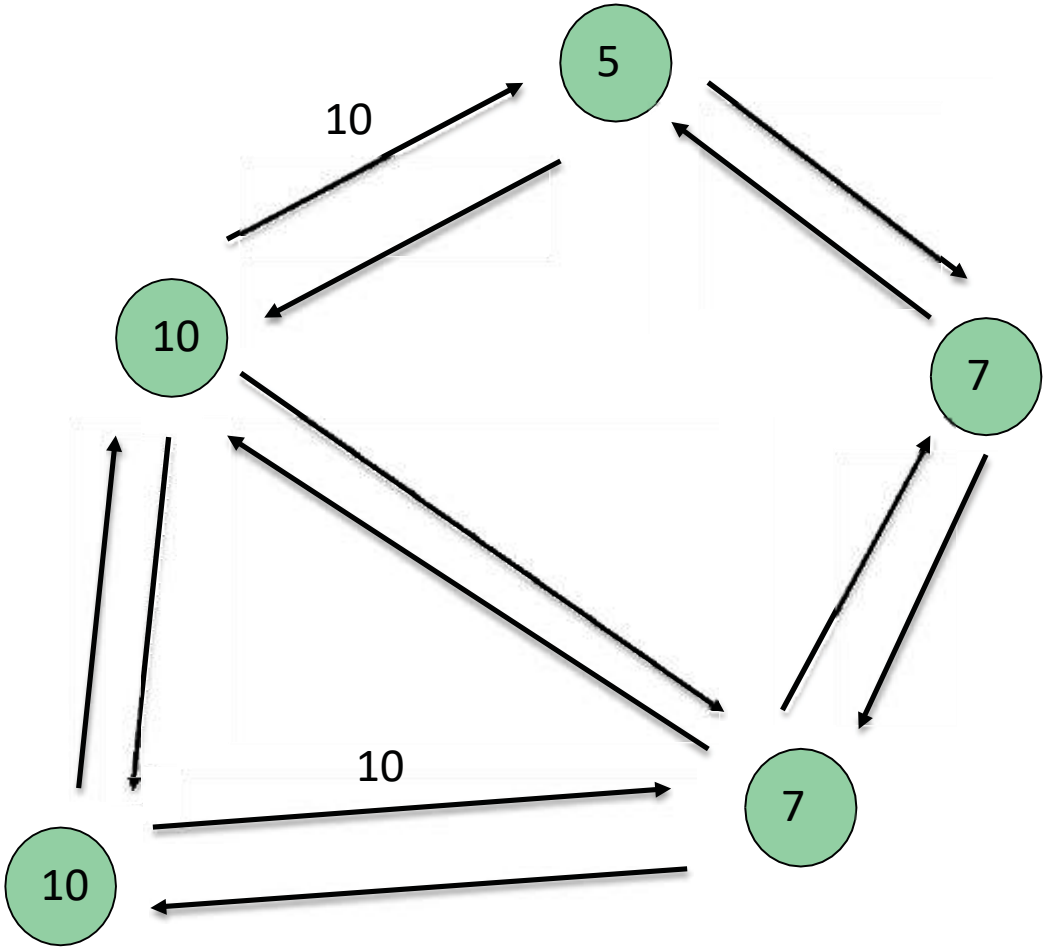
Max System



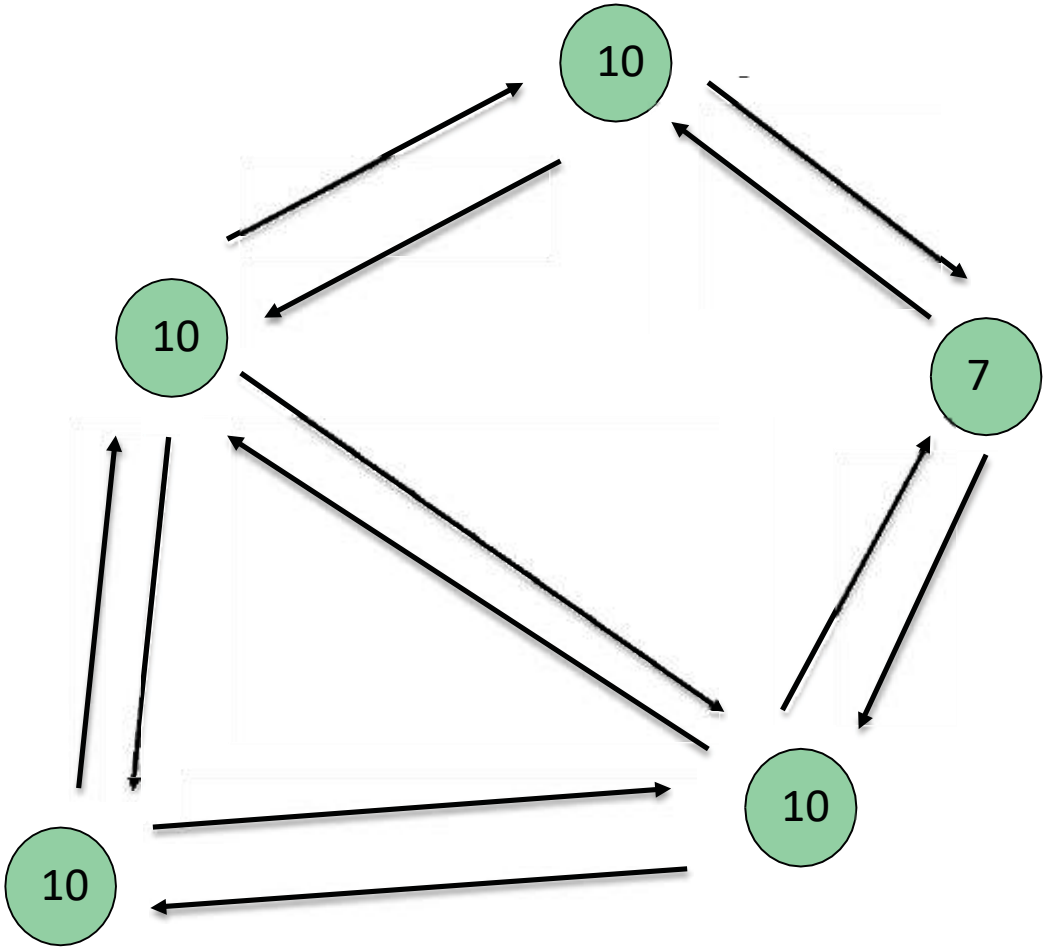
Max System



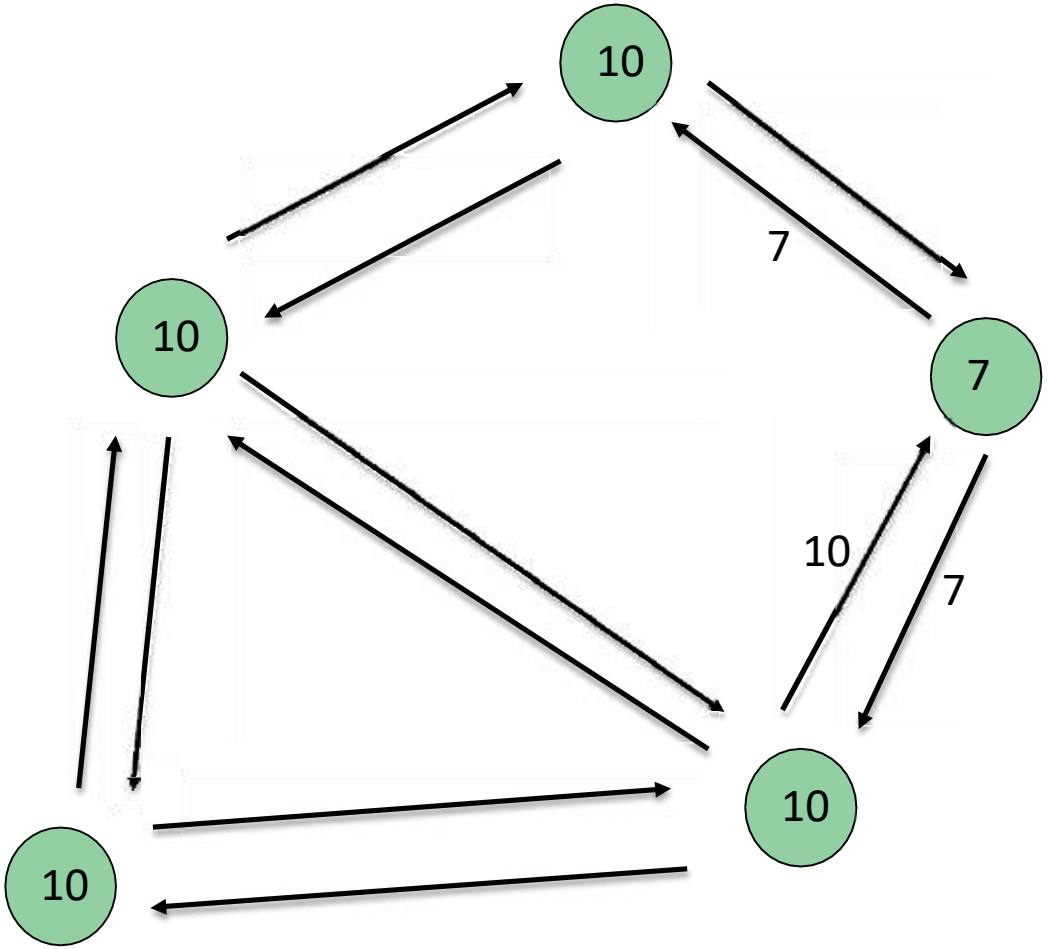
Max System



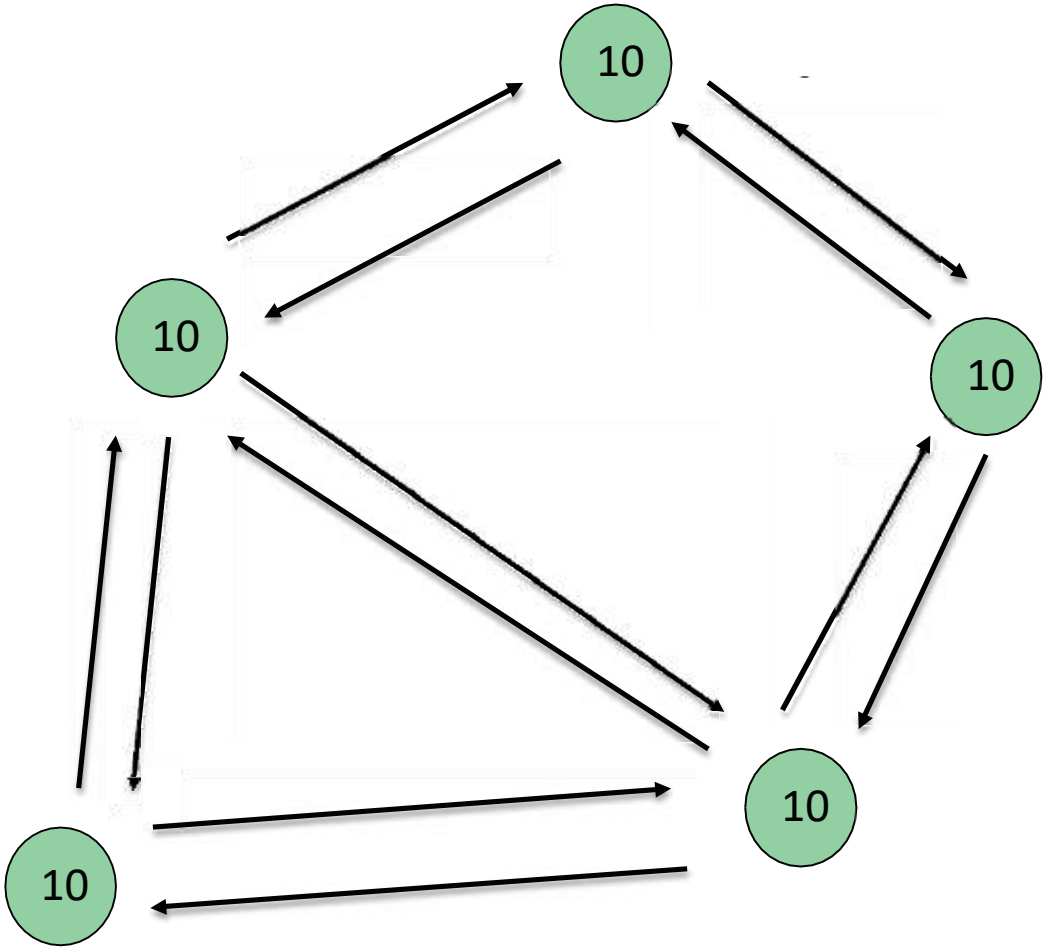
Max System



Max System



Max System



Complexity

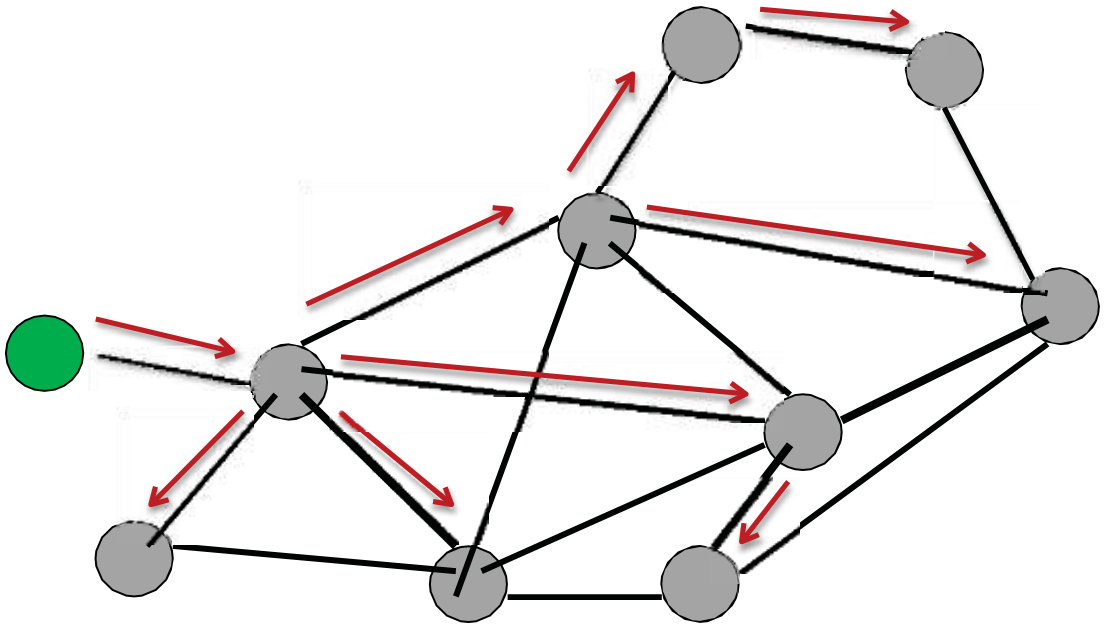
- Message complexity:
  - Number of messages sent by all processes during the entire execution.
  - $O(n \cdot |E|)$
- Time complexity:
  - Q: What should we measure?
  - Not obvious, because the various components are taking steps in arbitrary orders---no "rounds".
  - A common approach:
    - Assume real-time upper bounds on the time to perform basic steps:
      - d for a channel to deliver the next message, and
      - l for a process to perform its next step.
    - Infer a real-time upper bound for solving the overall problem.

Complexity

- Time complexity:
  - Assume real-time upper bounds on the time to perform basic steps:
    - d for a channel to deliver the next message, and
    - l for a process to perform its next step.
  - Infer a real-time upper bound for solving the problem.

- For the Max system:
  - Ignore local processing time ($l = 0$), consider only channel sending time.
  - Straightforward upper bound: $O(diam \cdot n \cdot d)$
    - Consider the time for the max to reach any particular vertex n, along a shortest path in the graph.
    - At worst, it waits in each channel on the path for every other value, which is at most time $n \cdot d$ for that channel.

Breadth-First Spanning Trees



Breadth-First Spanning Trees

- Problem: Compute a Breadth-First Spanning Tree in an asynchronous network.
- Connected graph $G = (V, E)$.
- Distinguished root vertex v_a.
- Processes have no knowledge about the graph.
- Processes have UIDs
  - i_a is the UID of the root v_a.
  - Processes know UIDs of their neighbors, and know which ports are connected to each neighbor.
- Processes must produce a BFS tree rooted at v_a.
- Each process $i = i_a$ should output $\mathrm{parent}(j)$, meaning that j's vertex is the parent of i's vertex in the BFS tree.

First Attempt

- Just run the simple synchronous BFS algorithm asynchronously.
- Process i_a sends search messages, which everyone propagates the first time they receive it.
- Everyone picks the first node from which it receives a search message as its parent.

- Nondeterminism:
  - No longer any nondeterminism in process decisions.
  - But plenty of new nondeterminism: orders of message deliveries and process steps.

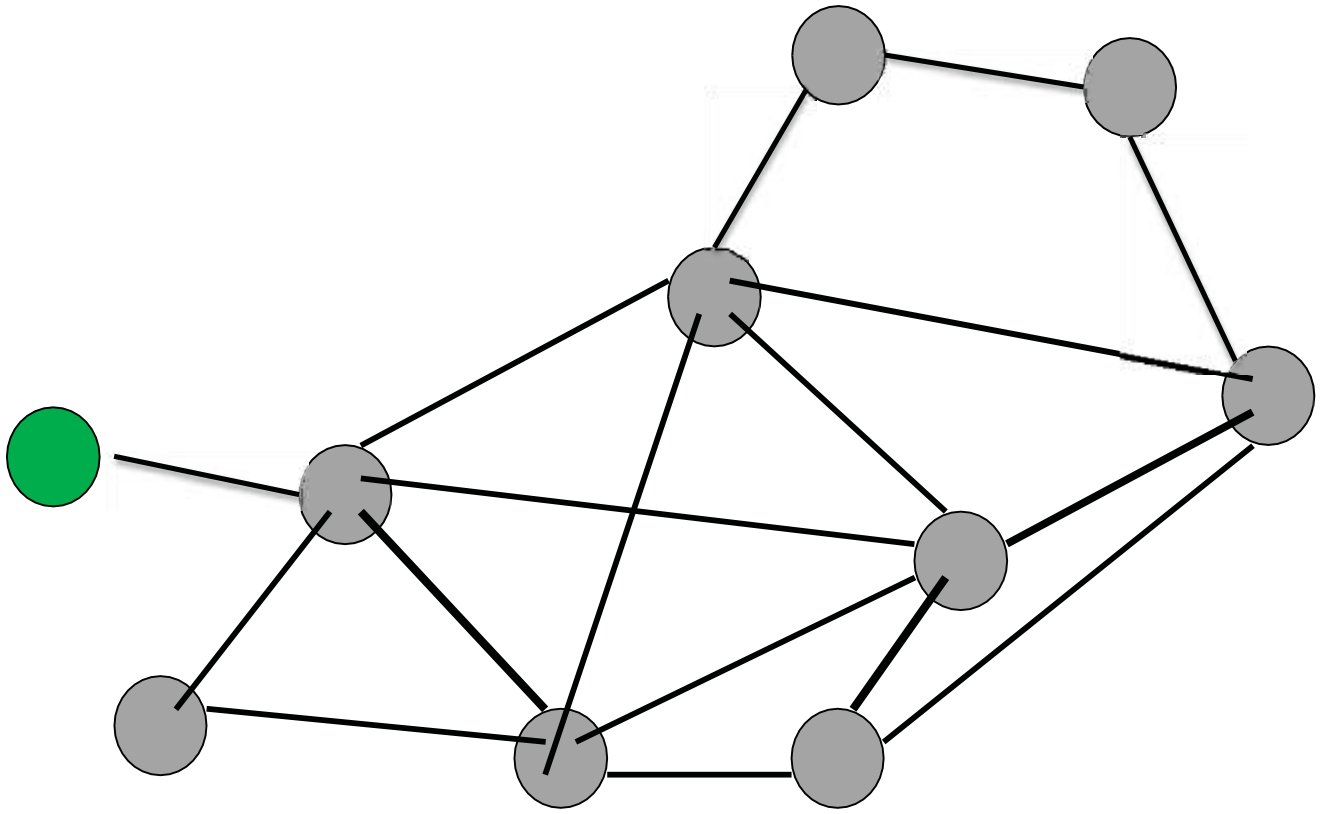
Process Automaton P_u

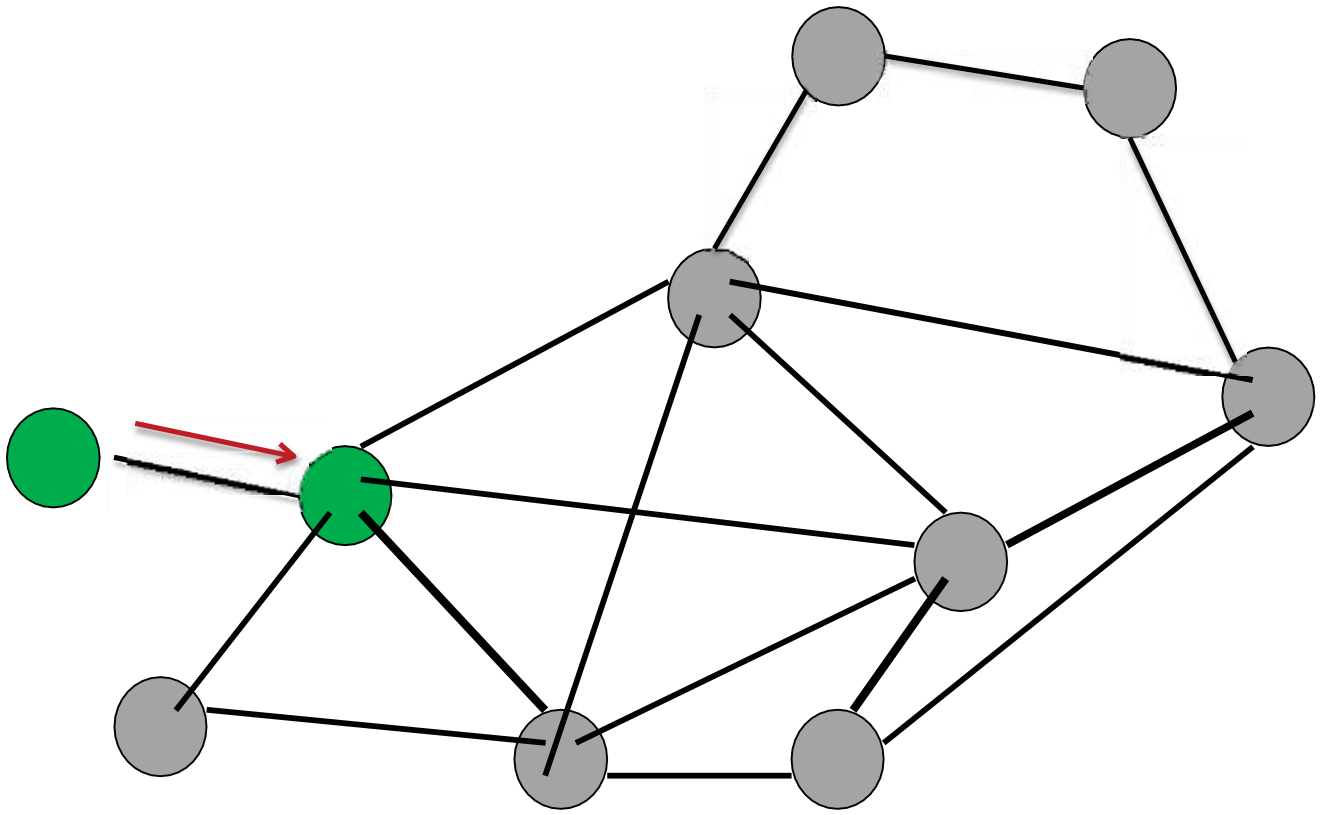
- Input actions: $receive(search)_{v,u}$
- Output actions: $send(search)_{u,v}$; $parent(v)_u$
- State variables:
  - parent: $T(n) \cup \{1\}$, initially 1
  - reported: Boolean, initially false
  - For every $v \in T(n)$:
    - $send(v) \in \{search, 1\}$, initially search if $n = v_a$, else 1
- Transitions:
  - $receive(search)_{v,u}$
    - Effect: if $n = v_a$ and $parent = 1$ then
      - parent := v
      - for every w, $send(w)$:= search

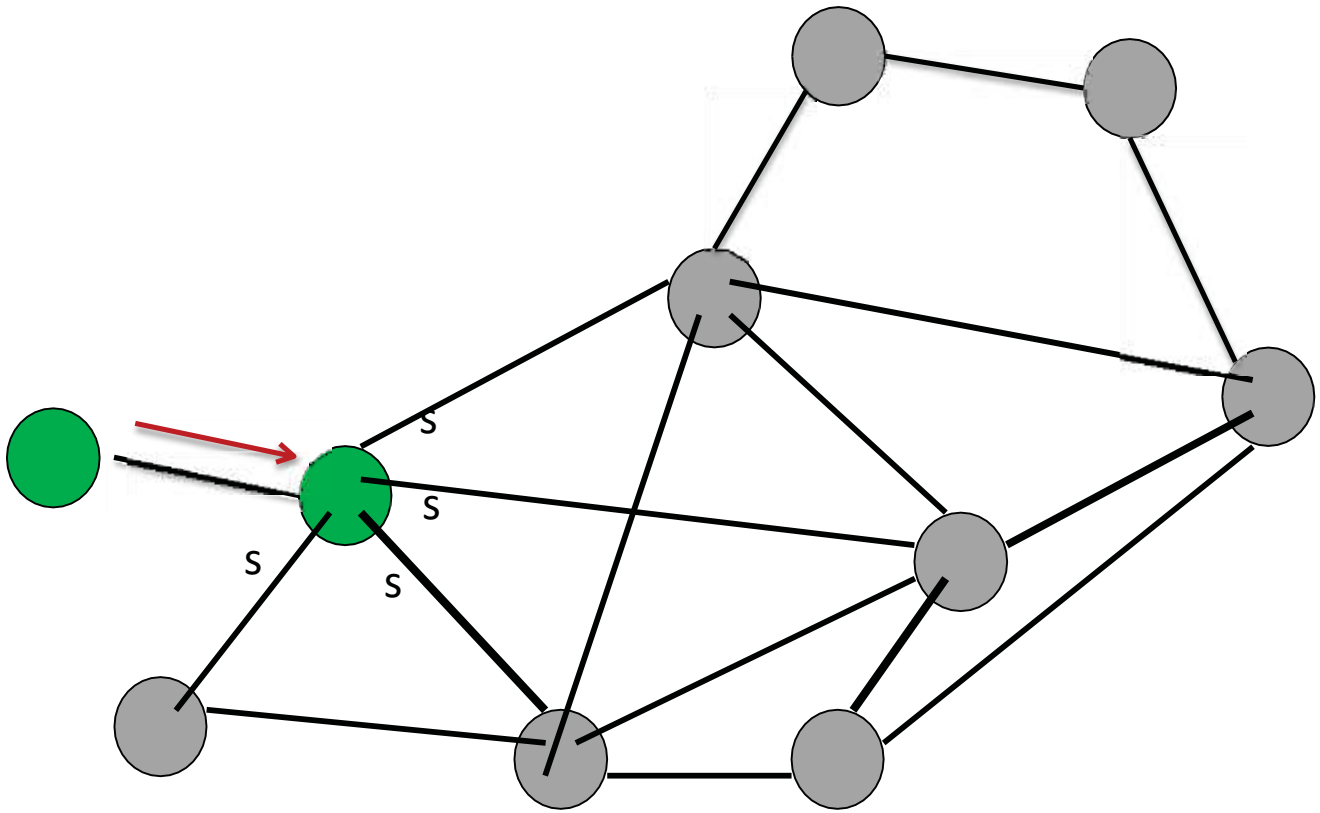
Process Automaton P_u

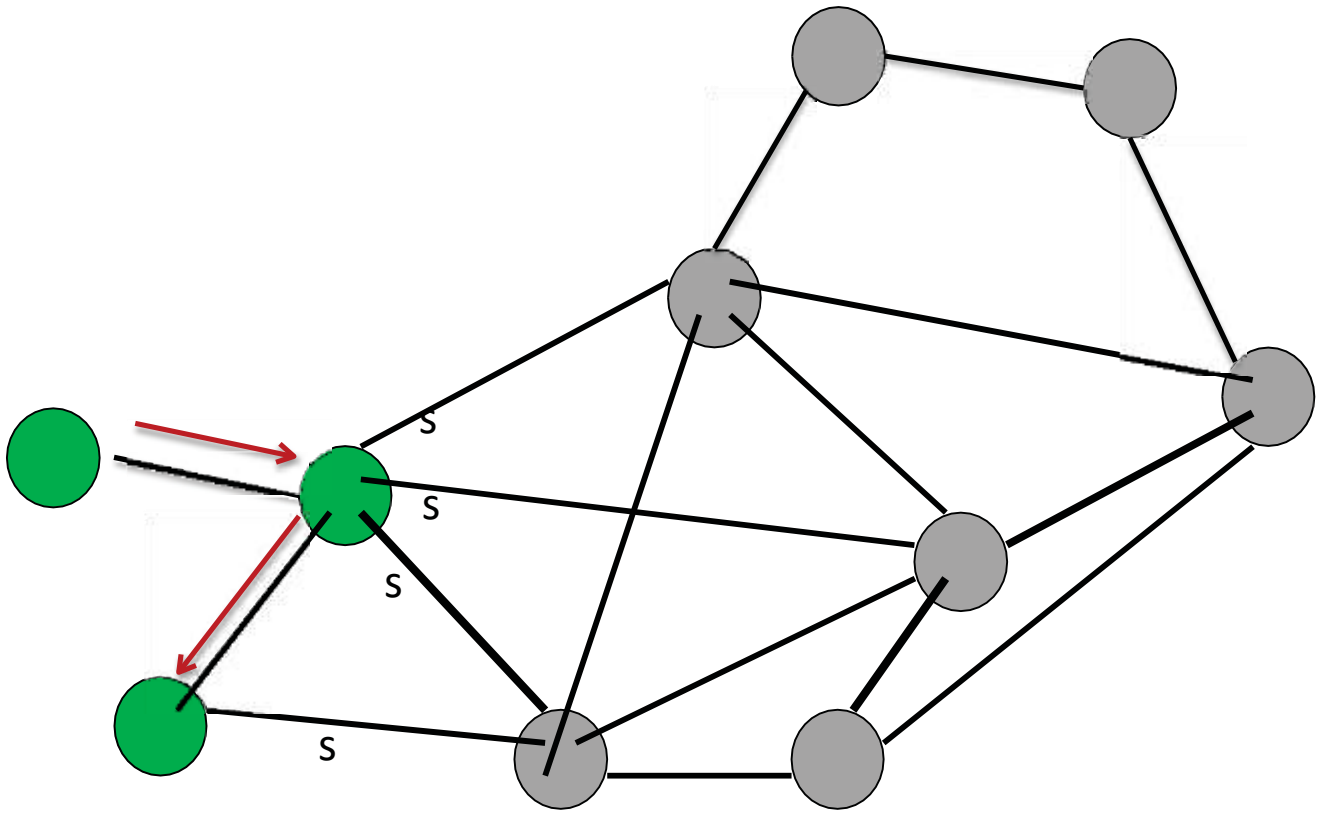
- Transitions:
  - $\text{receive}(\text{search})_{v,u}$
    - Effect: if $n = v_a$ and $\text{parent} = 1$ then
      - $\text{parent} := v$
      - for every w, $\text{send}(w) := \text{search}$
  - $\text{send}(\text{search})_{u,v}$
    - Precondition: $\text{send}(v) = \text{search}$
    - Effect: $\text{send}(v) : 1$
  - $\text{parent}(v)_u$
    - Precondition: $\text{parent} = v$ and $\text{reported} = \text{false}$
    - Effect: $\text{reported} : \text{tr}n\text{e}$

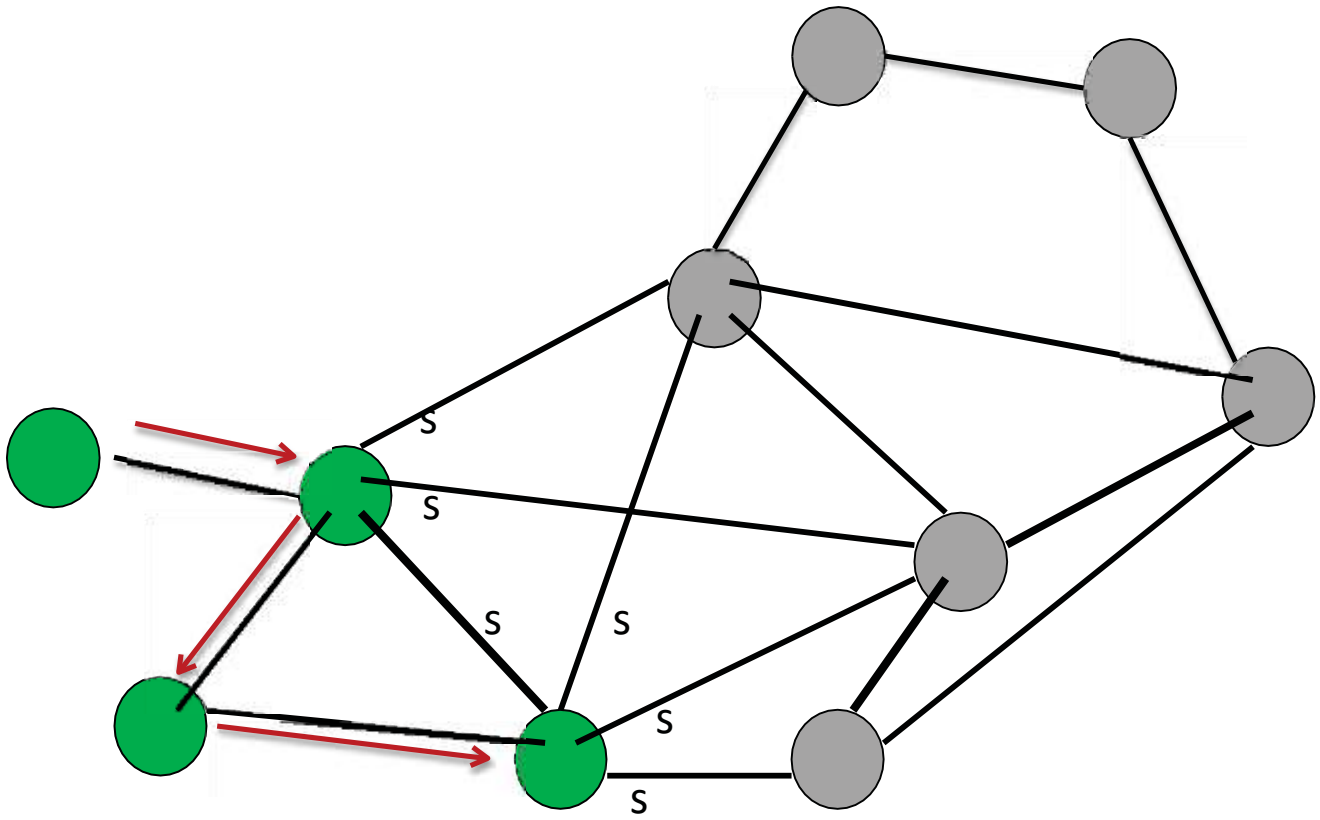
Running Simple BFS Asynchronously

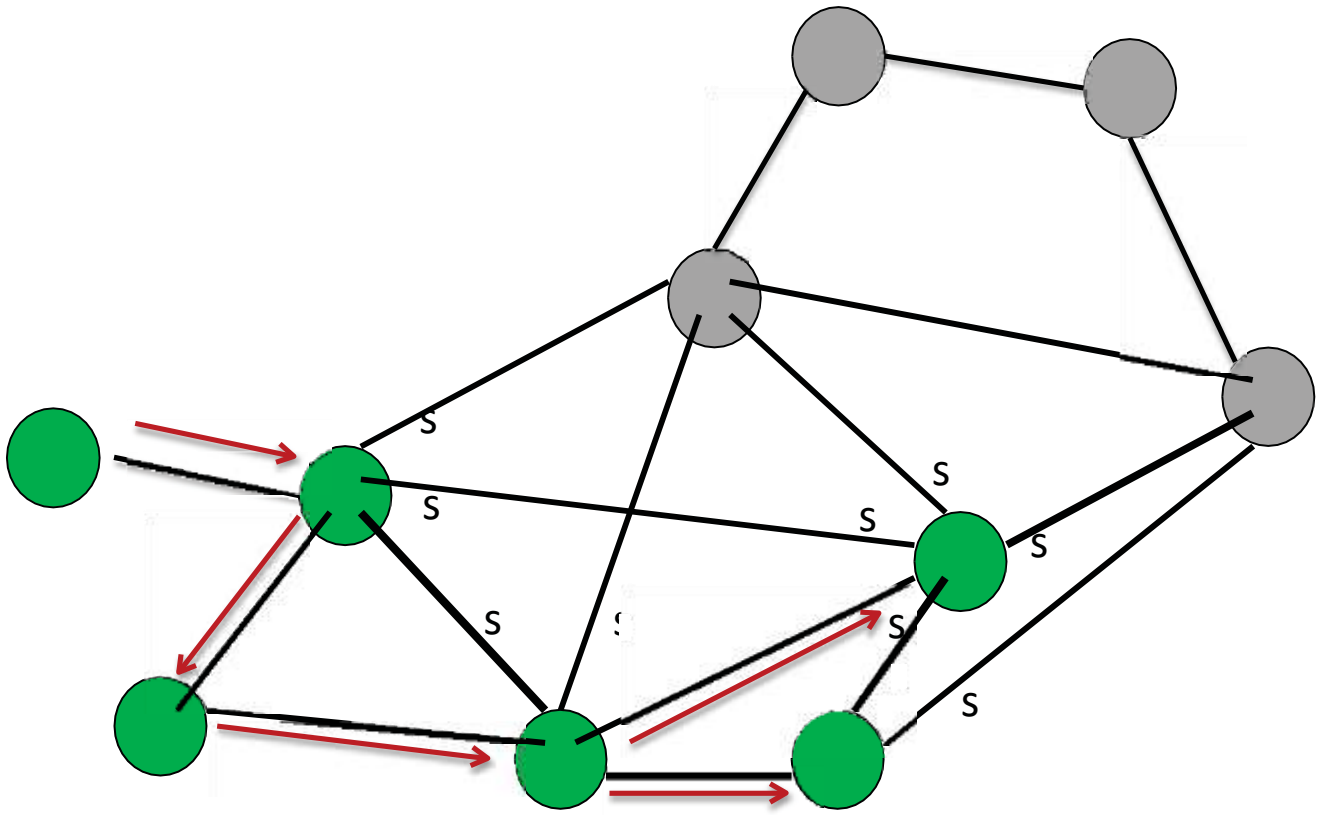


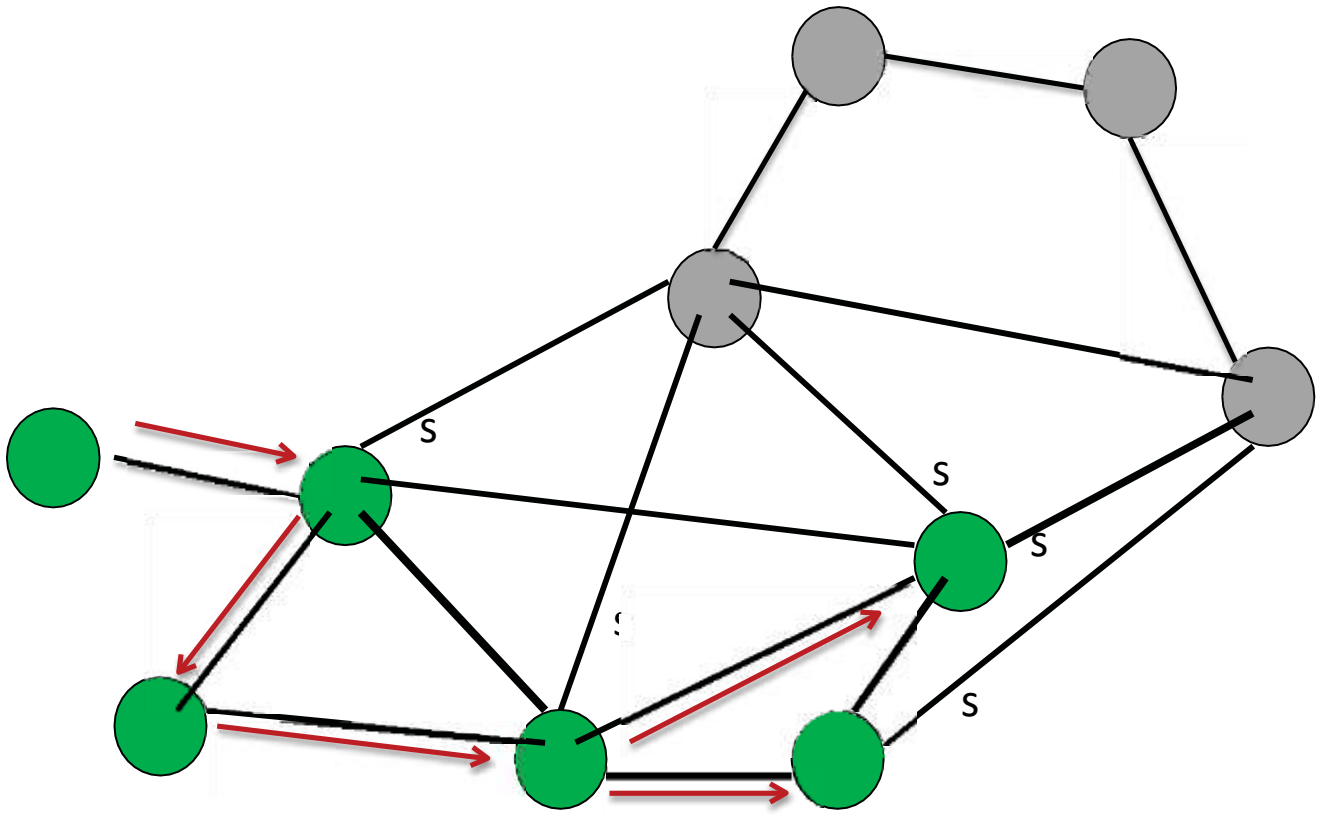


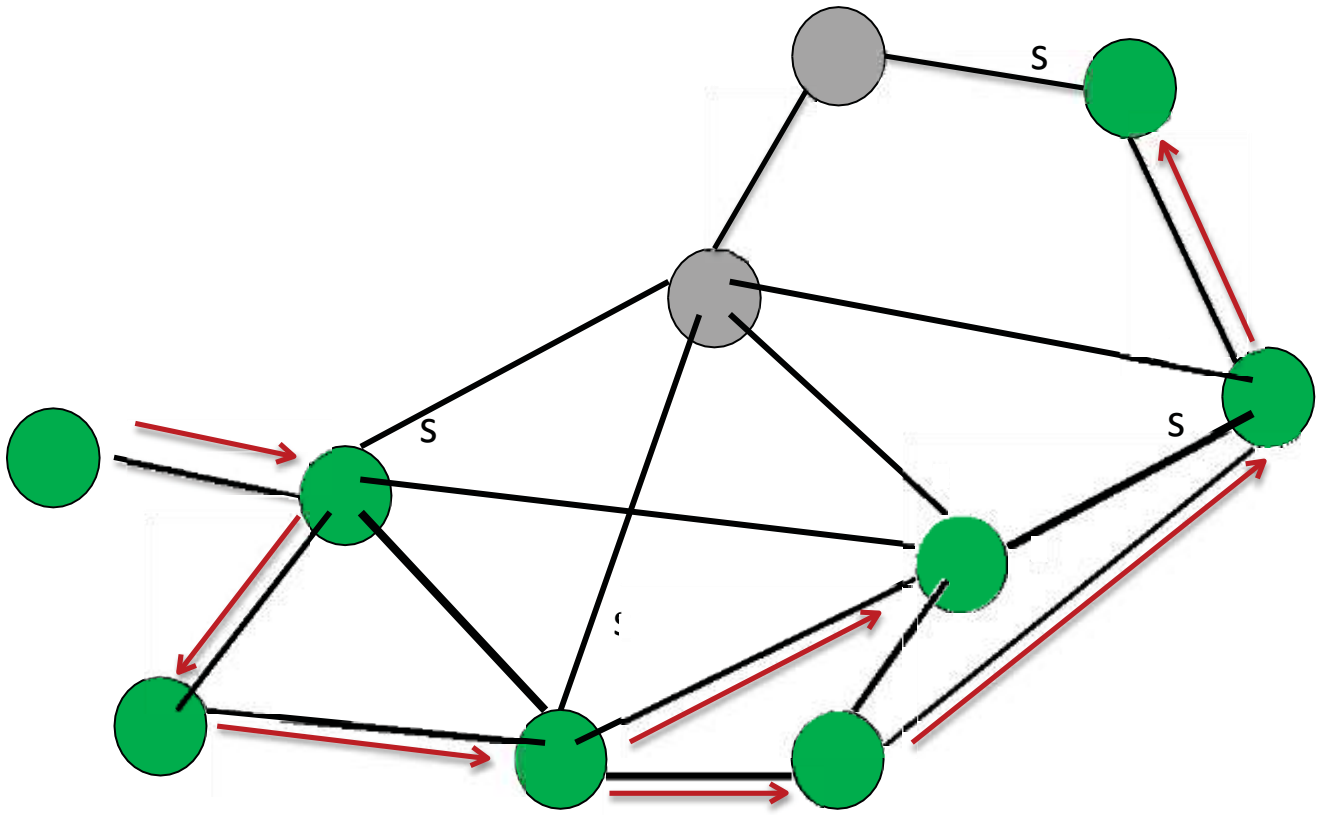

S
S
S
S


S
S
S
S

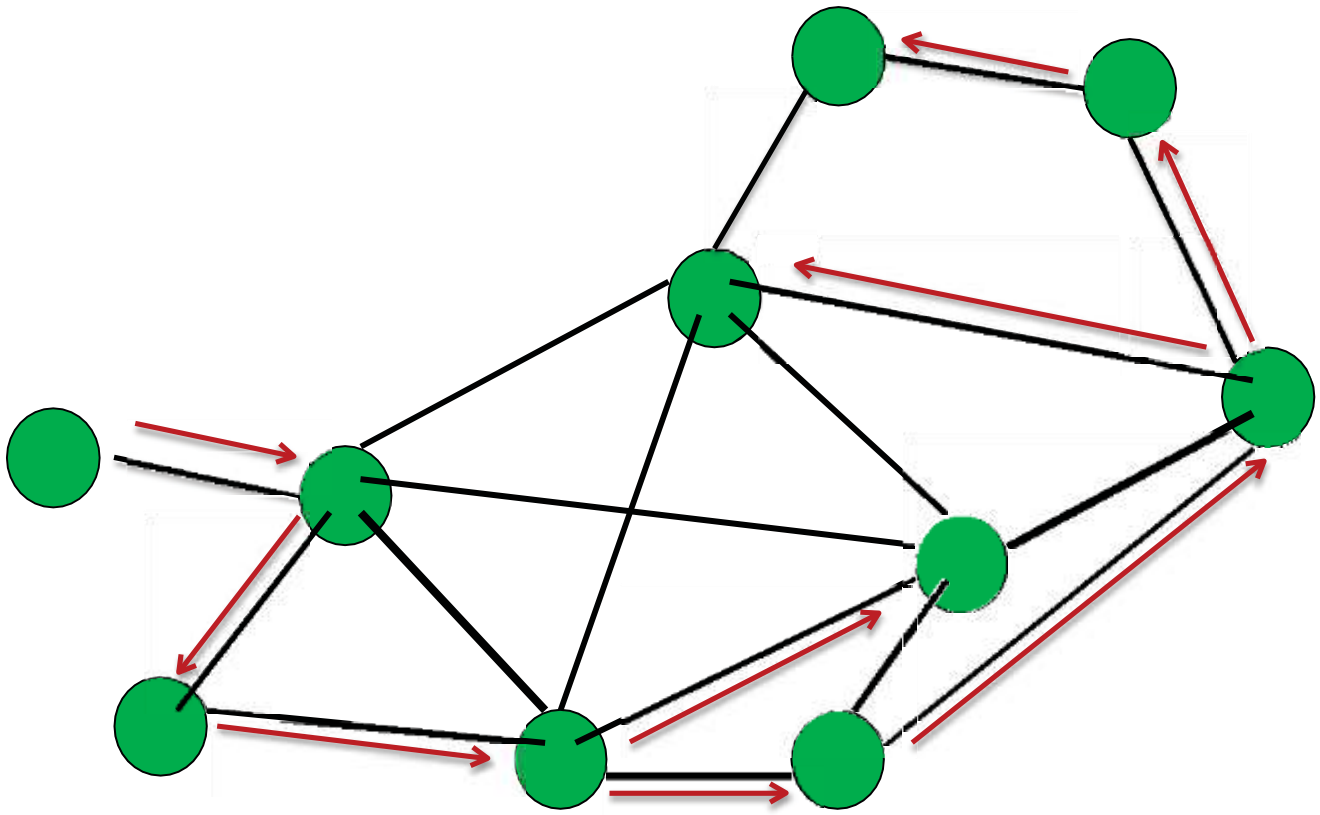

S
S
S
S
S
S


S
S
S
S
S
S
S
S
S

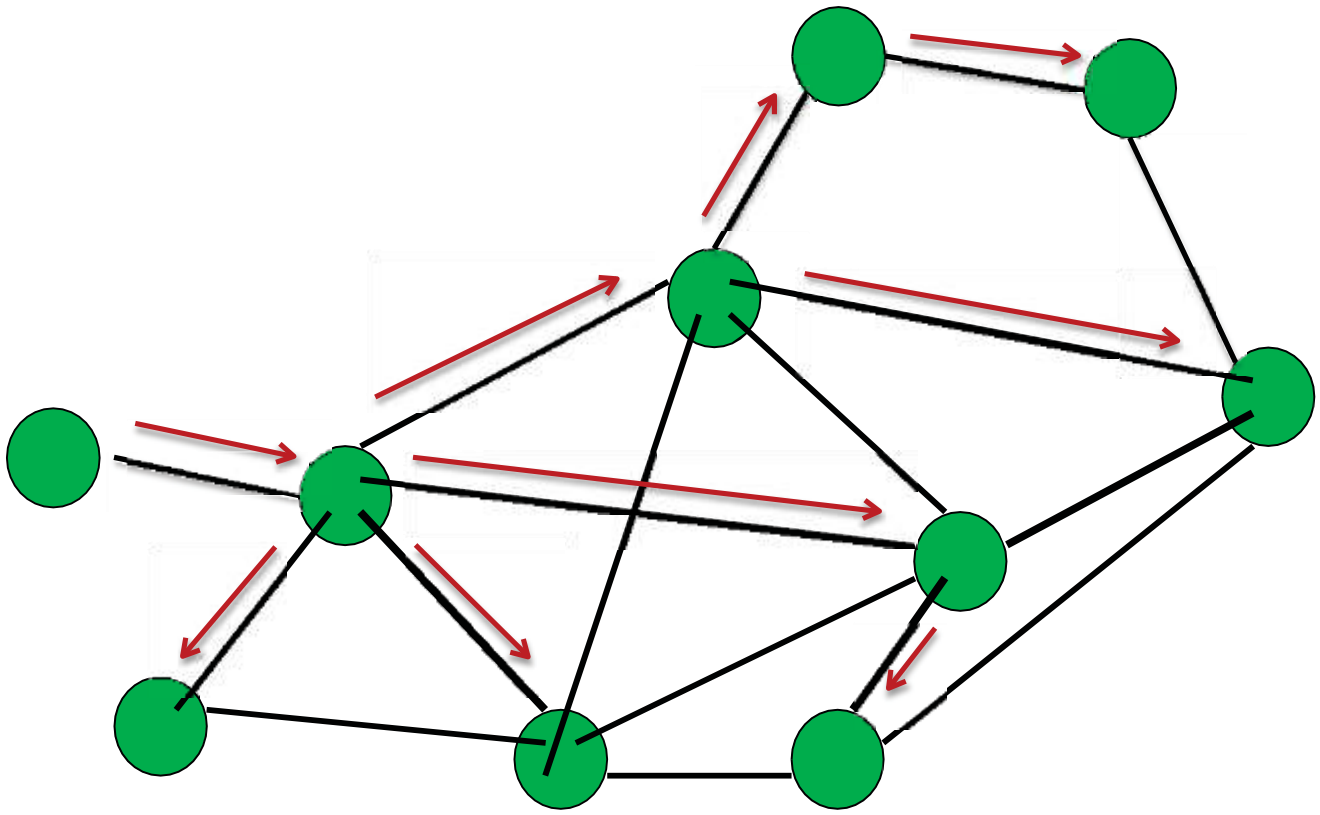

S
S
S
S


S
S
S

Final Spanning Tree

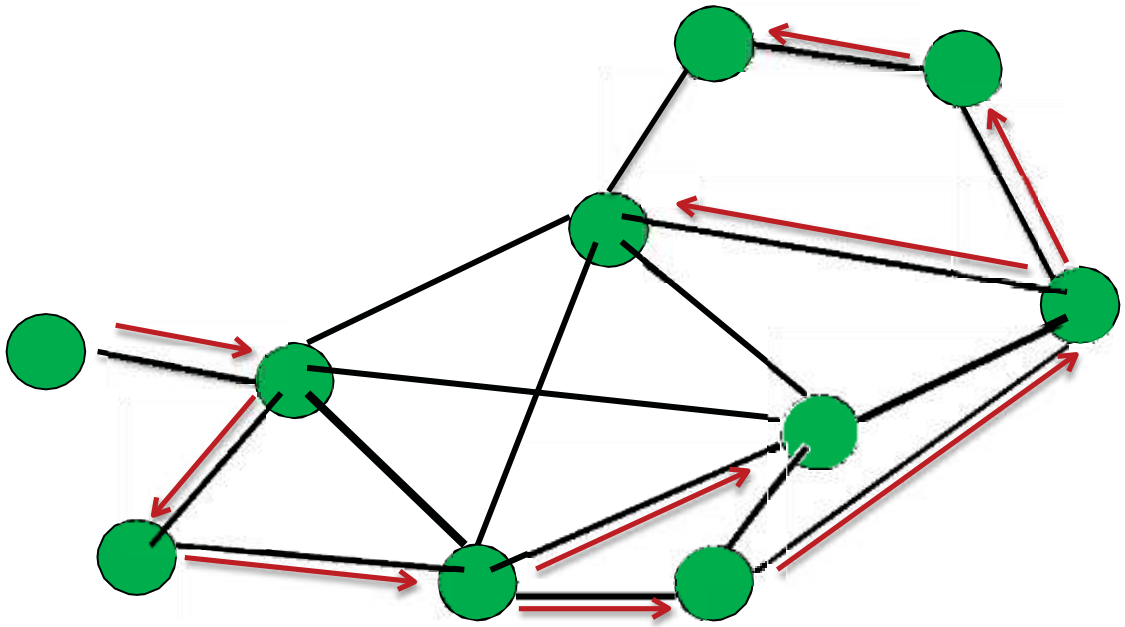


Actual BFS



Anomaly

- Paths produced by the algorithm may be longer than the shortest paths.
- Because in asynchronous networks, messages may propagate faster along longer paths.



Complexity

- Message complexity:
  - Number of messages sent by all processes during the entire execution.
  - $O(|E|)$

- Time complexity:
  - Time until all processes have chosen their parents.
  - Neglect local processing time.
  - $O(\text{diam} \cdot d)$ — all nodes reached.
  - Q: Why diam, when some of the paths are longer?
  - The time until a node receives a search message is at most the time it would take on a shortest path.

Extensions

- Child pointers:
  - As for synchronous BFS.
  - Everyone who receives a `search` message sends back a `parent` or `nonparent` response.
- Termination:
  - After a node has received responses to all its `search` its messages, it knows who its children are, and knows they are marked.
  - The leaves of the tree learn who they are.
  - Use a convergecast strategy, as before.
  - Time complexity: After the tree is done, it takes time $O(n \cdot d)$ for the done information to reach i_a.
  - Message complexity: $O(n)$

Applications

- Message broadcast:
  - Process i_a can use the tree (with child pointers) to broadcast a message.
  - Takes $O(n \cdot d)$ time and n messages.
- Global computation:
  - Suppose every process starts with some initial value, and process i_a should determine the value of some function of the set of all processes' values.
  - Use convergecast on the tree.
  - Takes $O(n \cdot d)$ time and n messages.

Second Attempt

- A relaxation algorithm, like synchronous Bellman-Ford.
- Before, we corrected for paths with many hops but low weights.
- Now, instead, correct for errors caused by asynchrony.
- Strategy:
  - Each process keeps track of the hop distance, changes its parent when it learns of a shorter path, and propagates the improved distances.
  - Eventually stabilizes to a breadth-first spanning tree.

Process Automaton P_u

- Input actions: $receive(m)_{v,u}$, m a nonnegative integer
- Output actions: $send(m)_{u,v}$, m a nonnegative integer

- State variables:
  - parent: $T(n) \cup \{1\}$, initially 1
  - dist $\in N \cup \{0\}$, initially 0 if $n = v_a$, 0 otherwise
  - For every $v \in T(n)$
    - $send(v)$, a FIFO queue of N, initially (0) if $n = v_a$, else empty

- Transitions:
  - $receive(m)_{v,u}$
    - Effect: if $m + 1 < dist$ then
      - $dist := m + 1$
      - $parent := v$
      - for every w, add $dist$ to $send(w)$

m = no. of hops

Process Automaton P_u

- Transitions:
  - $\text{receive}(m)_{v,u}$
    - Effect: if $m + 1 < \text{dist}$ then
      - $\text{dist} := m + 1$
      - $\text{parent} := v$
      - for every w, add $m + 1$ to $\text{send}(w)$
  - $\text{send}(m)_{u,v}$
    - Precondition: $m = \text{head}(\text{send}(v))$
    - Effect: remove head of $\text{send}(v)$

- No terminating actions…

Correctness

- For synchronous BFS, we characterized precisely the situation after r rounds.
- We can't do that now.
- Instead, state abstract properties, e.g., invariants and timing properties, e.g.:
- Invariant: At any point, for any node $n = v_a$, if its $\mathrm{dist} = 0$, then it is the actual distance on some path from v_a to n, and its parent is n's predecessor on such a path.
- Timing property: For any node n, and any r, $0 < r < \mathrm{diam}$, if there is an at-most-r-hop path from v_a to n, then by time $r \cdot n \cdot d$, node n's dist is $< r$.

Complexity

- Message complexity:
  - Number of messages sent by all processes during the entire execution.
  - $O(n\,|E|)$
- Time complexity:
  - Time until all processes' $dist$ and $parent$ values have stabilized.
  - Neglect local processing time.
  - $O(diam \cdot n \cdot d)$
    - Time until each node receives a message along a shortest path, counting time $O(n \cdot d)$ to traverse each link.

Termination

- Q: How can processes learn when the tree is completed?
- Q: How can a process know when it can output its own dist and parent?
- Knowing a bound on n doesn't help here: can't use it to count rounds.

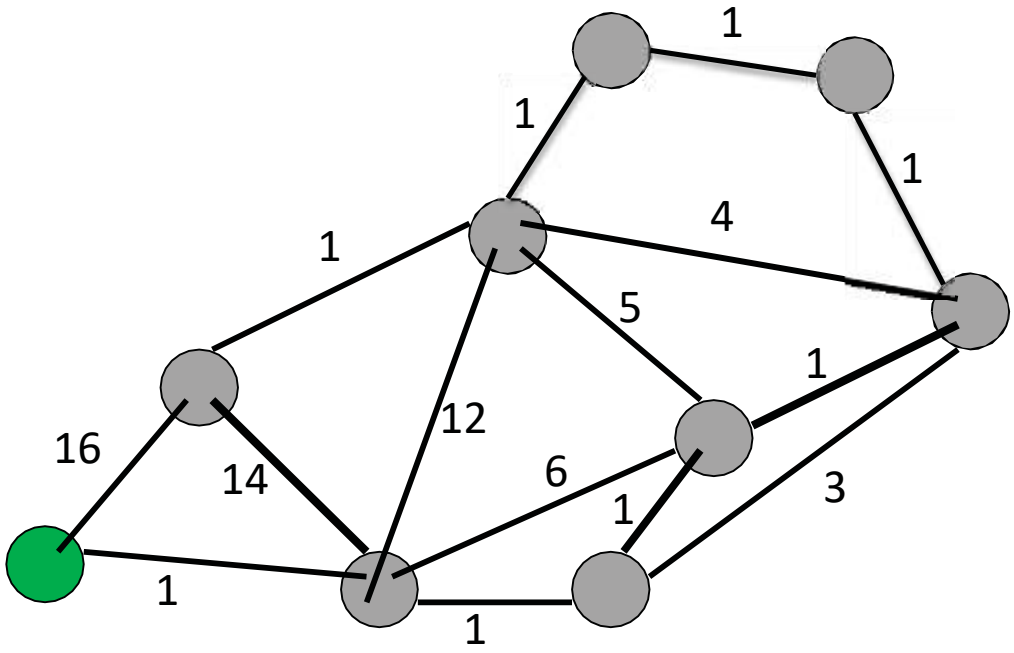
- Can use convergecast, as for synchronous Bellman-Ford:
  - Compute and recompute child pointers.
  - Process $= v_a$ sends done to its current parent after:
    - It has received responses to all its messages, so it believes it knows all its children, and
    - It has received done messages from all of those children.

  The same process may be involved several times, based on improved estimates.

Uses of Breadth-First Spanning Trees

- Same as in synchronous networks, e.g.:
  - Broadcast a sequence of messages
  - Global function computation
- Similar costs, but now count time d instead of one round.

Shortest Paths Trees



Shortest Paths

- Problem: Compute a Shortest Paths Spanning Tree in an asynchronous network.
- Connected weighted graph, root vertex v_a.
- $\text{weight}_{\{u,v\}}$ for edge $\{n, v\}$.
- Processes have no knowledge about the graph, except for weights of incident edges.
- UIDs

- Processes must produce a Shortest Paths spanning tree rooted at v_a.
- Each process $n = v_a$ should output its distance and parent in the tree.

Shortest Paths

- Use a relaxation algorithm, once again.
- Asynchronous Bellman-Ford.

- Now, it handles two kinds of corrections:
  - Because of long, small-weight paths (as in synchronous Bellman-Ford).
  - Because of asynchrony (as in asynchronous Breadth-First search).
- The combination leads to surprisingly high message and time complexity, much worse than either type of correction alone (exponential).

Asynch Bellman-Ford, Process P_u

- Input actions: $receive(m)_{v,u}$, m a nonnegative integer
- Output actions: $send(m)_{u,v}$, m a nonnegative integer

- State variables:
  - parent: $T(n)$ U $\{1\}$, initially 1
  - dist E N U $\{0\}$, initially 0 if $n = v_a$, 0 otherwise
  - For every v E $T(n)$
    - $send(v)$, a FIFO queue of N, initially (0) if $n = v_a$, else empty

- Transitions:
  - $receive(m)_{v,u}$
    - Effect: if $m + weight_{\{v,u\}} < dist$ then
      - $dist := m + weight_{\{v,u\}}$
      - $parent := v$
      - for every w, add dist to $send(w)$

Asynch Bellman-Ford, Process P_u

- Transitions:
  - $\text{receive}(m)_{v,u}$
    - Effect: if $m + \text{weight}_{\{v,u\}} < \text{dist}$ then
      - $\text{dist} := m + \text{weight}_{\{v,u\}}$
      - $\text{parent} := v$
      - for every w, add dist to send(w)
  - $\text{send}(m)_{u,v}$
    - Precondition: $m = \text{head}(\text{send}(v))$
    - Effect: remove head of send(v)

- No terminating actions…

Correctness: Invariants and Timing Properties

- Invariant: At any point, for any node $n = \mathrm{v_a}$, if its $\mathtt{dist} = 0$, then it is the actual distance on some path from $\mathrm{v_a}$ to n, and its $\mathtt{parent}$ is n's predecessor on such a path.

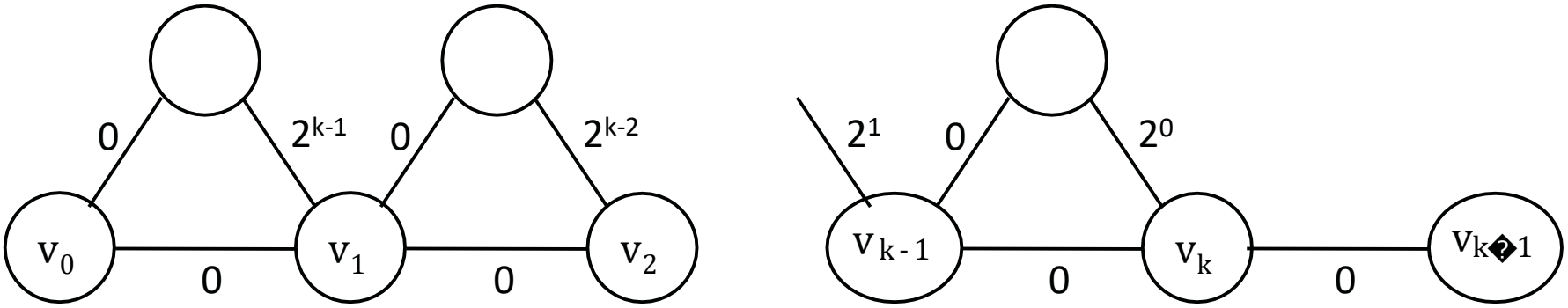
- Timing property: For any node n, and any r, $0 < \mathrm{r} < \mathrm{diam}$, if p is any at-most-r-hop path from $\mathrm{v_a}$ to n, then by time ???, node n's $\mathtt{dist}$ is $<$ total weight of p.

- Q: What is ??? ?
- It depends on how many messages might pile up in a channel.
- This can be a lot!

Complexity

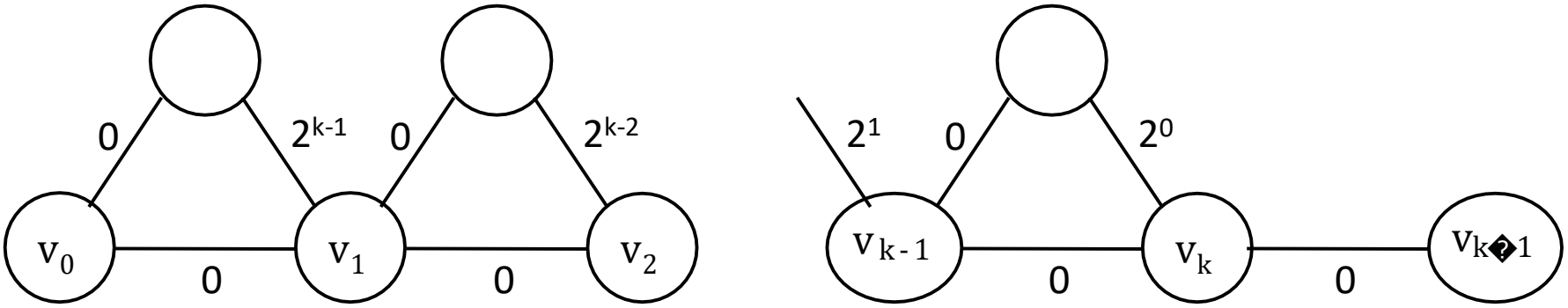
- $O(n!)$ simple paths from v_0 to any other node n, which is $O(n^n)$.
- So the number of messages sent on any channel is $O(n^n)$.
- Message complexity: $O(n^n |E|)$.
- Time complexity: $O(n^n \cdot n \cdot d)$.

- Q: Are such exponential bounds really achievable?



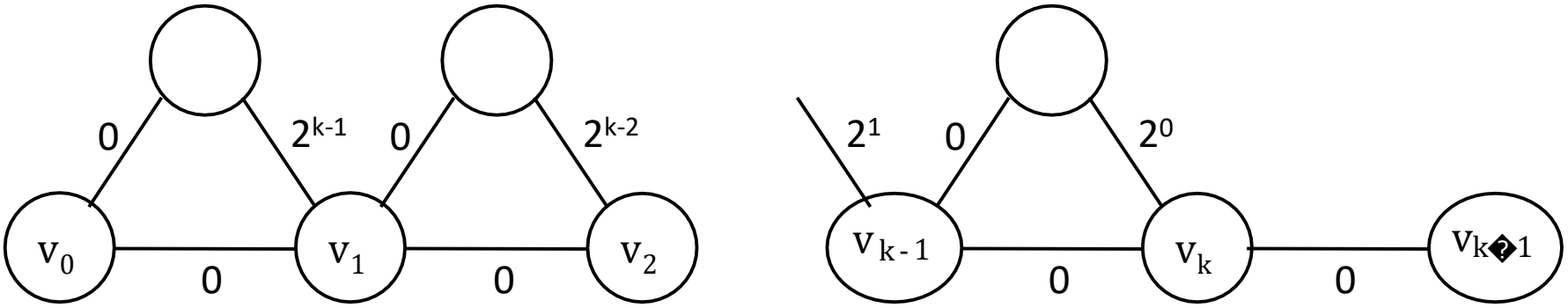
Complexity

- Q: Are such exponential bounds really achievable?
- Example:
  - There is an execution of the network below in which node v_k sends 2^k $\diamond$ $2^{n/2}$ messages to node v_{k+1}.
  - Message complexity is $O(2^{n/2})$.
  - Time complexity is $O(2^{n/2} d)$.



Complexity

- Execution in which node v_k sends 2^k messages to node v_{k+1}.
- Possible distance estimates for v_k are $2^k - 1, 2^k - 2, \ldots, 0$.
- Moreover, v_k can take on all these estimates in sequence:
  - First, messages traverse upper links, $2^k - 1$.
  - Then last lower message arrives at v_k, $2^k - 2$.
  - Then lower message $v_{k-2} \rightarrow v_{k-1}$ arrives, reduces v_{k-1}'s estimate by 2, message $v_{k-1} \rightarrow v_k$ arrives on upper links, $2^k - 3$.
  - Etc. Count down in binary.
  - If this happens quickly, get pileup of 2^k search messages in $C_{k,k-1}$.



Termination

- Q: How can processes learn when the tree is completed?
- Q: How can a process know when it can output its own dist and parent?

- Convergecast, once again
  - Compute and recompute child pointers.
  - Process $= v_a$ sends done to its current parent after:
    - It has received responses to all its messages, so it believes it knows all its children, and
    - It has received done messages from all of those children.
  - The same process may be involved several (many) times, based on improved estimates.