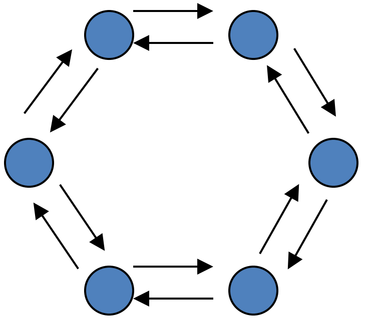
6.852: Distributed Algorithms
Fall, 2015

Lecture 2

Today's plan

- Leader election in a synchronous ring:
  - Lower bound for comparison-based algorithms.
- Basic computation in general synchronous networks:
  - Leader election
  - Breadth-first search
  - Broadcast and convergecast
  - Shortest paths (Bellman-Ford)
- Reading: Sections 3.6, 4.1-4-3
- Next time:
  - Shortest paths, continued
  - Minimum Spanning Tree
  - Maximal Independent Set
  - Reading: Sections 4.3-4.5, related papers (see last slide)

Leader Election in a Synchronous Ring



Last time

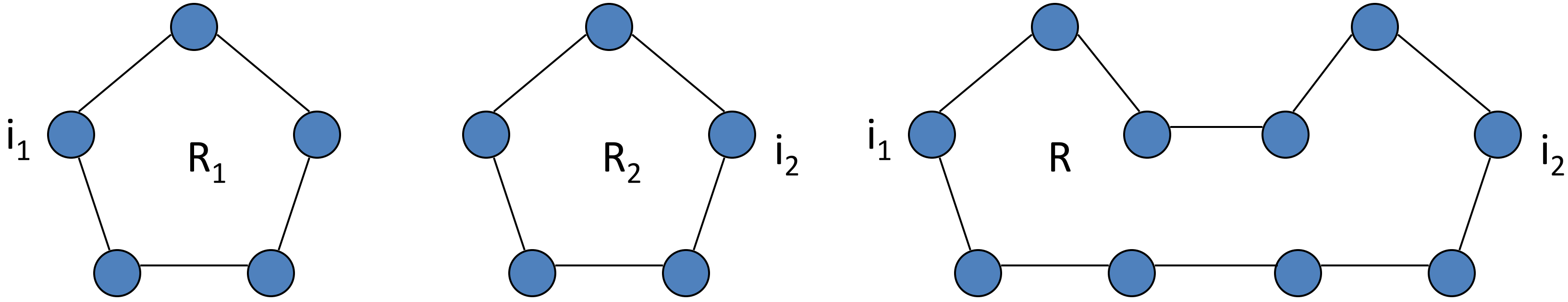
- Model for synchronous networks
- Leader election problem, in simple ring networks
- Algorithms:
  - [LeLann], [Chang, Roberts]
    - Pass UID tokens one way, elect max
    - Proofs, using invariants
    - Time complexity: n for a ring of size n
    - Communication (message) complexity: $O(n^2)$
  - [Hirshberg, Sinclair]
    - Send UIDs to successively-doubled distances, in both directions.
    - Message complexity: $O(n \log n)$
    - Time complexity: $O(n)$ (dominated by the final phase)

Last time

- Q: Can the message complexity be lowered still more?
- Non-comparison-based algorithms
  - Wait quietly until it's your "turn", determined by UID.
  - Message complexity: $O(n)$
  - Time complexity: $O(u_{min}\, n)$ if n is known, $O(n\, 2^{u_{min}})$ if n is unknown

Lower bounds for leader election

- Q: Can we get time complexity less than n?
- Easy $n/2$ lower bound (n unknown)
  - Suppose an algorithm always elects a leader in time $< n/2$.
  - Consider two separate rings of size n (n odd), R_1 and R_2.
  - Algorithm elects processes i_1 and i_2 respectively, each in time $< n/2$.



  - Now cut R_1 and R_2 at points furthest from the leaders, paste them together to form a new ring R of size $2n$.
  - Then in R, both i_1 and i_2 get elected, because the time until they get elected is less than the time needed for information about the pasting to propagate from the pasting points to i_1 or i_2.

Lower bounds for leader election

- Q: Can we get message complexity less than $O(n \log n)$, for comparison-based algorithms?
- We can prove an $\Omega(n \log n)$ lower bound.
- Assumptions:
  - Comparison-based algorithm.
  - Bidirectional ring.
  - Known n.
  - Deterministic.

Comparison-based algorithms

- All decisions are determined only by comparisons of UIDs:
  - All processes are identical, except that they have different UIDs in their start states.
  - Manipulate UIDs only by copying, sending, receiving, and comparing them $(<, =, >)$.
  - Use results of comparisons to decide what to do:
    - State transitions,
    - What (if anything) to send to your neighbors,
    - Whether to elect yourself leader.

Lower bound theorem

- Theorem 1: Let A be a comparison-based algorithm that elects a leader in rings of size n. Then A has an execution in which $\Omega(n \log n)$ messages are sent by the time the leader is elected.
- This holds for any n.

- Proof overview:
  - For any n, define a ring R_n of size n in which any leader election algorithm has:
    - $\Omega(n)$ "active" rounds (in which messages are sent).
    - $\Omega(n \,/\, i)$ messages sent in the i^{th} active round.
    - So, $\Omega(n \log n)$ total messages.
  - The key is to choose ring R_n with a lot of symmetry in the ordering pattern of UIDs.

Proof overview, cont'd

- Choose ring R_n with a lot of symmetry in the ordering pattern of UIDs.
- Informal lemma statements:
- Lemma 2: Processes whose neighborhoods have the same ordering pattern act the same, until information from outside their neighborhoods reaches them.
- Lemma 3: If two processes have large order-equivalent neighborhoods, then many active rounds are needed to break symmetry between them.
- Lemma 4: If the ring has enough processes with large-enough order-equivalent neighborhoods, then during each active round many processes send messages.

- Now, the details…

Definitions

- A round is active if some process sends a (non-null) message in that round.
- k-neighborhood of a process: The $2k + 1$ processes within distance k on both sides.

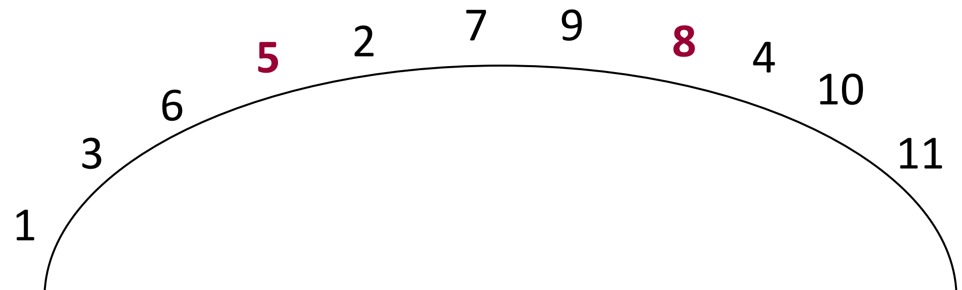
- Tuples $(u_1, u_2, \ldots, u_k)$ and $(v_1, v_2, \ldots, v_k)$ are order-equivalent provided that $u_i \leq u_j$ iff $v_i \leq v_j$ for all pairs i, j.
  - Implies the same $(<, =, >)$ relationships for all corresponding pairs.
  - Example: $(1\ 3\ 6\ 5\ 2\ 7\ 9)$ vs. $(2\ 7\ 9\ 8\ 4\ 10\ 11)$

- Two process states s and t correspond with respect to $(u_1, u_2, \ldots, u_k)$ and $(v_1, v_2, \ldots, v_k)$ if they are identical except that occurrences of u_i in s are replaced by v_i in t, for every i.
- Analogous definition for corresponding messages.

Key Lemma: Lemma 2

- Lemma 2: Suppose A is a comparison-based algorithm on a synchronous ring network. Suppose i and j are processes whose sequences of UIDs in their k-neighborhoods are order-equivalent.

  Then at any point after at most k active rounds, the states of i and j correspond wrt their k-neighborhoods' UID sequences.
- That is, processes with order-equivalent k-neighborhoods are indistinguishable until after "enough" active rounds.
- Enough: Information has had a chance to reach the processes from outside the k-neighborhoods.
- Example: 5 and 8 have order-equivalent 3-neighborhoods, so must remain in corresponding states through 3 active rounds.



Proof of Lemma 2

- Lemma 2: Suppose i and j are processes whose sequences of UIDs in their k-neighborhoods are order-equivalent. Then at any point after $\leq k$ active rounds, the states of i and j correspond wrt their k-neighborhoods' UID sequences.

- Proof:
  - Induction on $r\ = $ number of completed rounds (for each r, consider every i, j, and every $k\ \geq\ 0$).
  - Base: $r\ =\ 0$
    - Start states of i and j are identical except for UIDs.
    - Correspond with respect to their k-neighborhoods, for every k.
  - Inductive step: Assume for $r - 1$, show for r.

Proof of Lemma 2

- Lemma 2: Suppose i and j have order-equivalent k-neighborhoods. Then at any point after $\leq k$ active rounds, i and j are in corresponding states wrt their k-neighborhoods.
- Inductive step:
  - Assume this is true after round $r-1$, for all i, j, k.
  - Prove it's true after round r, for all i, j, k.
  - Fix i, j, k, where i and j have order-equivalent k-neighborhoods.
  - Assume $i \neq j$ (trivial otherwise).
  - Assume at most k of the first r rounds are active.
  - We must show that, after r rounds, i and j are in corresponding states wrt their k-neighborhoods.
  - By inductive hypothesis, after $r-1$ rounds, i and j are in corresponding states wrt their k-neighborhoods.
  - If neither i nor j receives a non-null message at round r, they make corresponding transitions, to corresponding states (wrt their k-neighborhoods), so the conclusion is true.
  - So suppose that at least one of i, j receives a message at round r.

Proof of Lemma 2

- Lemma 2: Suppose i and j have order-equivalent k-neighborhoods. Then at any point after $\leq k$ active rounds, i and j are in corresponding states wrt their k-neighborhoods.
- Inductive step, cont'd:
  - So assume at least one of i, j receives a message at round r.
  - Then round r is active, and the first $r - 1$ rounds include at most $k - 1$ active rounds.
  - The $(k-1)$ −neighborhoods of $i - 1$ and $j - 1$ are order-equivalent, since they are included within the k-neighborhoods of i and j.
  - By inductive hypothesis, after $r - 1$ rounds, $i - 1$ and $j - 1$ are in corresponding states wrt their $(k-1)$ −neighborhoods, and thus wrt the k-neighborhoods of i and j.
  - Thus, messages from $i - 1$ to i and from $j - 1$ to j at round r correspond.
  - Similarly for messages from $i + 1$ to i and from $j + 1$ to j.
  - So, i and j start round r in corresponding states and receive corresponding messages at round r, so they make corresponding transitions and end up in corresponding states at the end of round r.
  - As needed.

Proof of Theorem 1, cont'd

- We have shown:
- Lemma 2: Suppose i and j have order-equivalent k-neighborhoods. Then at any point after $\leq k$ active rounds, i and j are in corresponding states wrt their k-neighborhoods.

- Lemma 2 implies that many active rounds are needed to break symmetry, if there are large order-equivalent neighborhoods.
- It remains to show:
  - There are rings with many, and large, order-equivalent neighborhoods.
  - Having all these order-equivalent neighborhoods implies high communication complexity.
- First, let's see how order-equivalent neighborhoods yield high communication complexity…

Lemma 3

- Lemma 3: Suppose A is a comparison-based leader-election algorithm on a synchronous ring network, and k is an integer.
  Suppose that, for every process i, there is another process j such that i and j have order-equivalent k-neighborhoods.
  Then A has more than k active rounds.

- Proof: By contradiction.
  - Suppose A elects i in at most k active rounds.
  - By assumption, there is a distinct process j with an order-equivalent k-neighborhood.
  - By Lemma 2, i and j are in corresponding states, so j is also elected—a contradiction.

Lemma 4

- Lemma 4: Suppose A is a comparison-based algorithm on a synchronous ring network, and k, m are integers.

  Suppose that the $(k-1)$-neighborhood of every process is order-equivalent to that of at least $m-1$ other processes.

  Then at least m messages are sent in A's k^{th} active round.

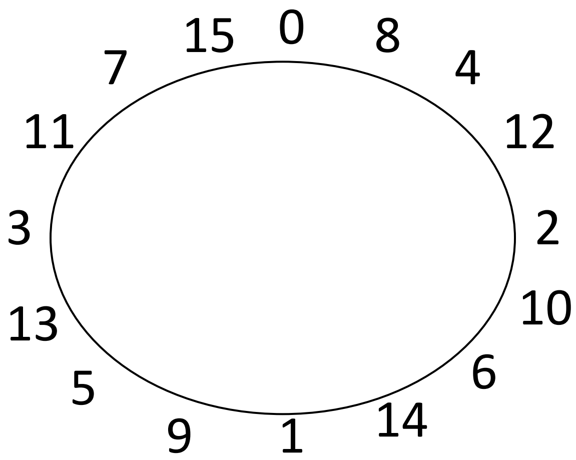
- Proof:
  - By definition, some process sends a message in the k^{th} active round.
  - By assumption, at least $m-1$ other processes have order-equivalent $(k-1)$-neighborhoods.
  - By Lemma 2, immediately before this round, all these processes are in corresponding states wrt their $(k-1)$ −neighborhoods. Thus, they all send messages in this round, so at least m messages are sent.

Proof of Theorem 1, cont'd

- We have shown:
- Lemma 3: Suppose that, for every process i, there is another process j such that i and j have order-equivalent k-neighborhoods. Then A has more than k active rounds.
- Lemma 4: Suppose the $(k-1)$-neighborhood of any process is order-equivalent to that of at least $m-1$ other processes. Then at least m messages are sent in A's k^{th} active round.
- Lemmas 3 and 4 together imply that order-equivalent neighborhoods yield high communication complexity:
  - Lemma 3 says there are many active rounds.
  - Lemma 4 says that each active round has many messages.
- To finish the proof of Theorem 1, it is enough to show the existence of rings with many, large order-equivalent neighborhoods.
- Example special case: n a power of 2.

n a power of 2

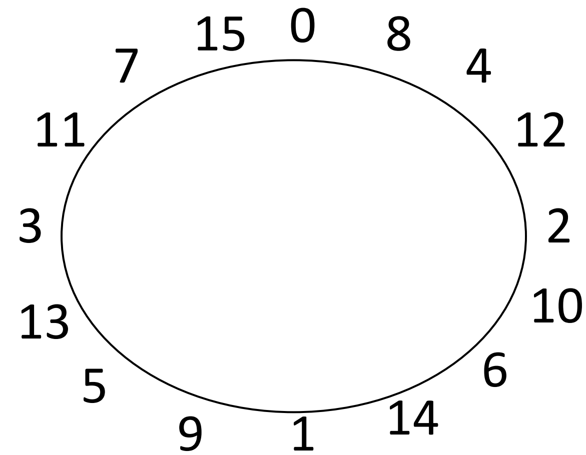
- Bit-reversal ring
  - UID is bit-reversed process number.
- Example:



– For every segment of length $n/2^b$, there are (at least) 2^b order-equivalent segments (including the given segment).

n a power of 2

- Bit-reversal ring.
- For every segment of length $n/2^b$, there are (at least) 2^b order-equivalent segments (including the given segment).
- Implies that every process i has at least $n/(4k)$ processes (including i) with order-equivalent k-neighborhoods, for $k \leq n/4$.

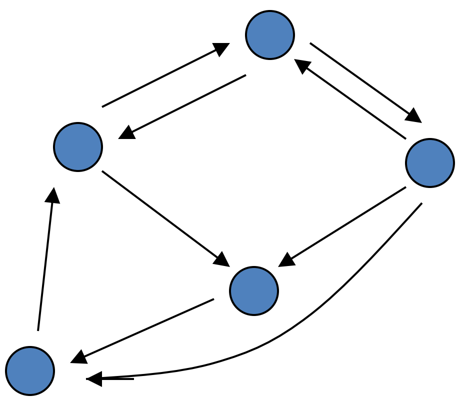


- More than $n/8$ active rounds, by Lemma 3.
- Number of messages $\geq\ n/4\ +\ n/8\ +\ n/12\ +\ n/16\ +$ ... $+\ 2$, by Lemma 4, which is $\Omega(n \log n)$.
- Calculations LTTR.

Proof idea for arbitrary n

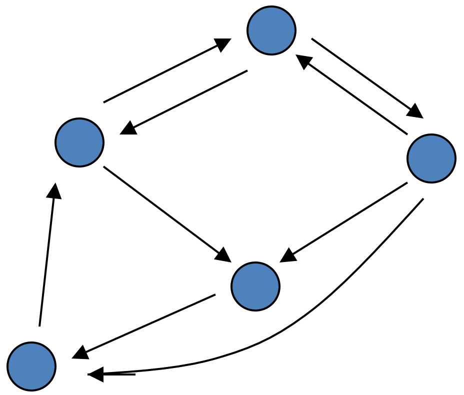
- c-symmetric ring: For every l such that $\sqrt{n} < l < n$, and every sequence of length l in the ring, there are at least $\lfloor cn/l \rfloor$ order-equivalent occurrences.
- [Frederickson-Lynch] There exists c such that for every positive integer n, there is a c-symmetric ring of size n.
- Given c-symmetric ring, argue similarly to before.

Basic Computation in General Synchronous Networks (not just rings)



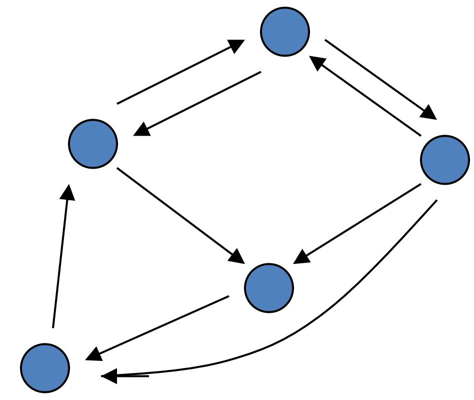
General synchronous networks

- Not just rings, but arbitrary digraphs.



- Today: Consider simple algorithms, for basic tasks like broadcasting messages, collecting responses, setting up communication structures.
- These algorithms are simplified versions of algorithms that work in asynchronous networks. We will revisit them in a few weeks.
- Soon: Maximal Independent Set, coloring.

Assumptions



- Digraph $G = (V, E)$:
  - V = set of processes
  - E = set of communication channels
  - $distance(i, j)$ = shortest distance from i to j
  - $diam$ = max $distance(i, j)$ for all i, j
  - Assume: Strongly connected ($diam$ is finite), UIDs
- Set M of messages
- Each process has $states$, $start$, $msgs$, $trans$.
- Processes communicate only over digraph edges.
- Generally don't know the entire network, just local neighborhood.
- Local names for neighbors.
  - No particular order for neighbors, in general.
  - But (technicality) if incoming and outgoing edges connect to same neighbor, the names are the same (so the node "knows" this).

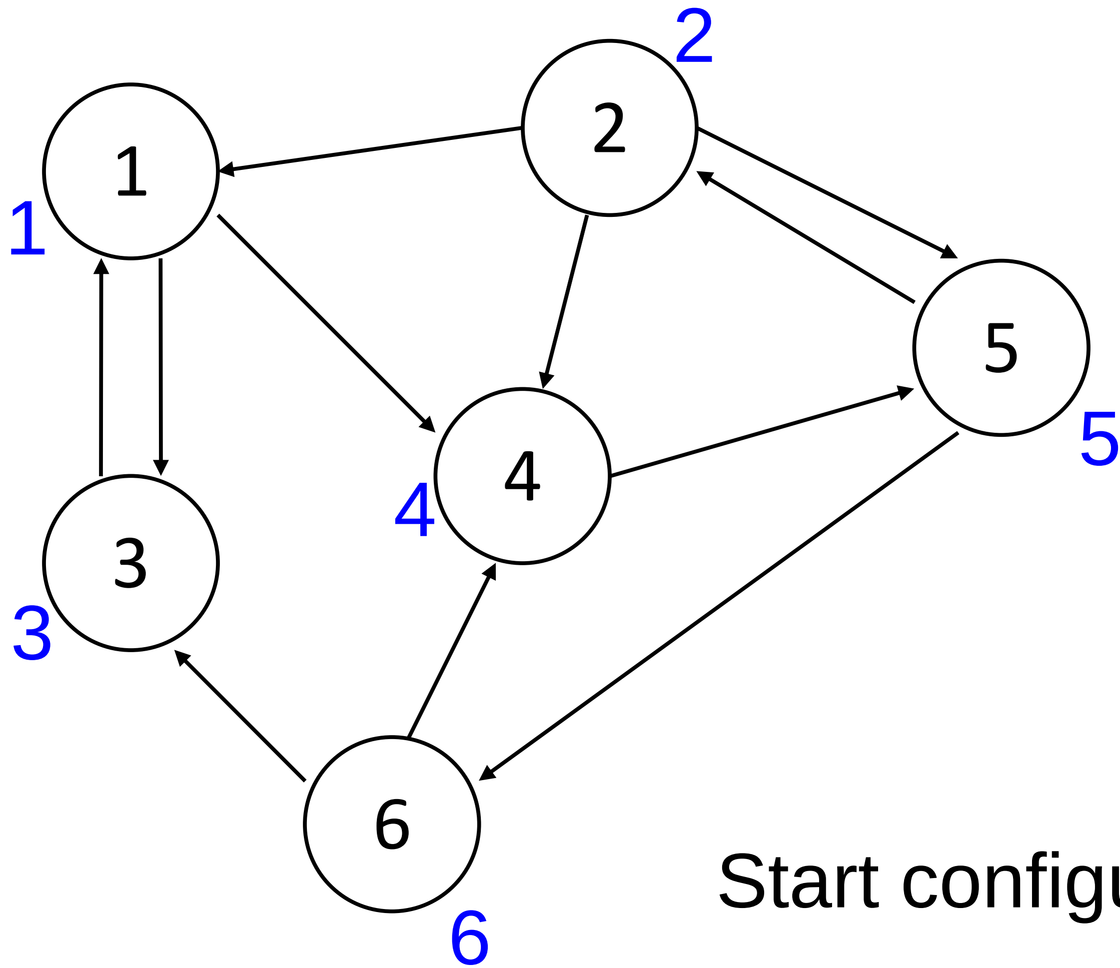
Leader election in general synchronous networks

- Assume:
  - UIDs with comparisons only.
  - No constraints on which UIDs appear, or where they are in the graph.
  - Processes know the graph diameter (or a good upper bound).
- Required: Everyone should eventually set $status \in$ {leader, non-leader}, exactly one leader.
- We will:
  - Show a basic flooding algorithm, sketch a proof using invariants.
  - Show an optimized version, sketch a proof that relates it formally to the basic algorithm (new idea: simulation relations).
- Basic flooding algorithm, any process:
  - Every round: Send max UID you have seen so far to all your neighbors.
  - Stop after $diam$ rounds.
  - Elect yourself iff your own UID is the max you have seen.

Basic flooding algorithm

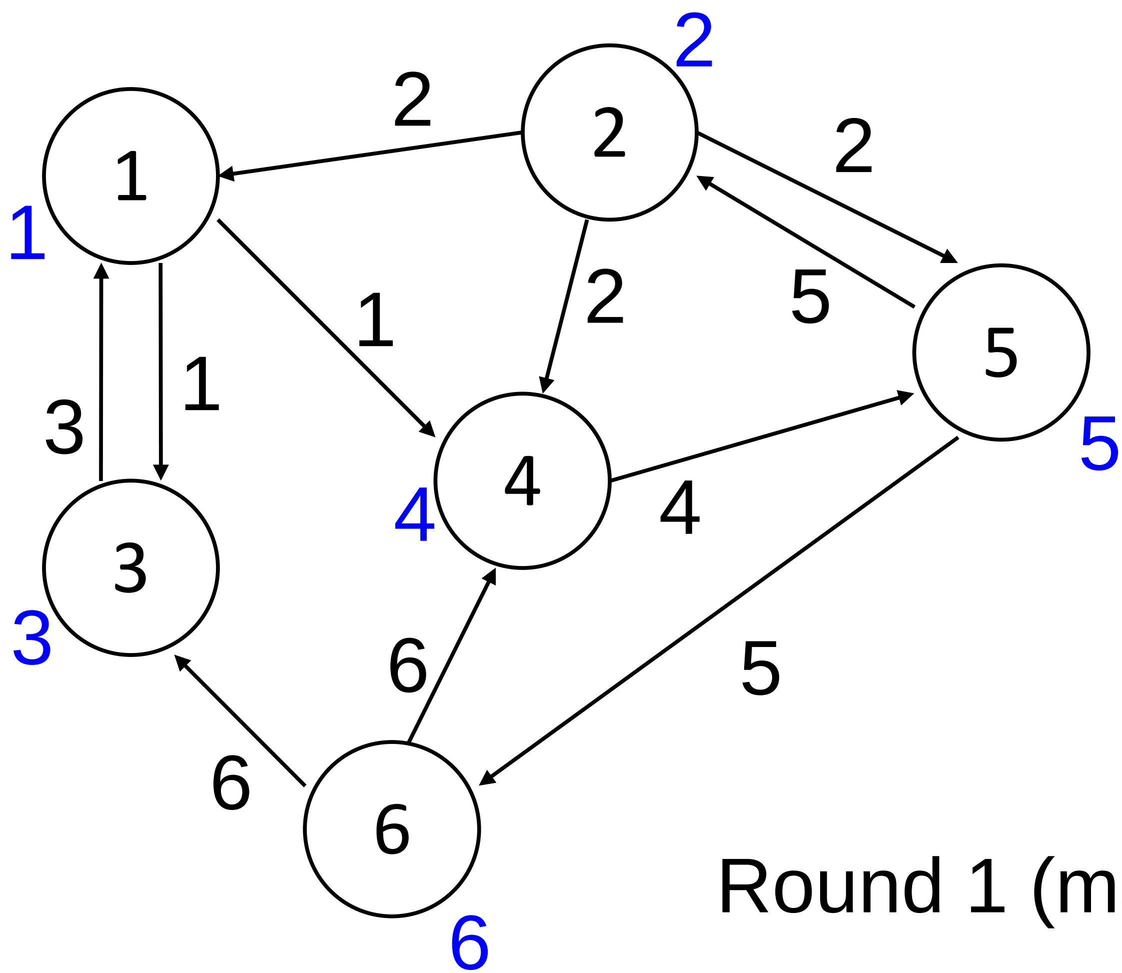
- *states*
  - u, initially UID
  - $maxuid$, initially UID
  - $status \in$ {?, leader, not-leader}, initially ?
  - $round$, initially 0
- *msgs*
  - if $round < diam$ then send $maxuid$ to all $outnbrs$
- *trans*
  - increment $round$
  - $maxuid$:= max ($maxuid$, UIDs received)
  - if $round = diam$ then
    - $status$:= leader if $maxuid = u$, not-leader otherwise

Basic flooding algorithm



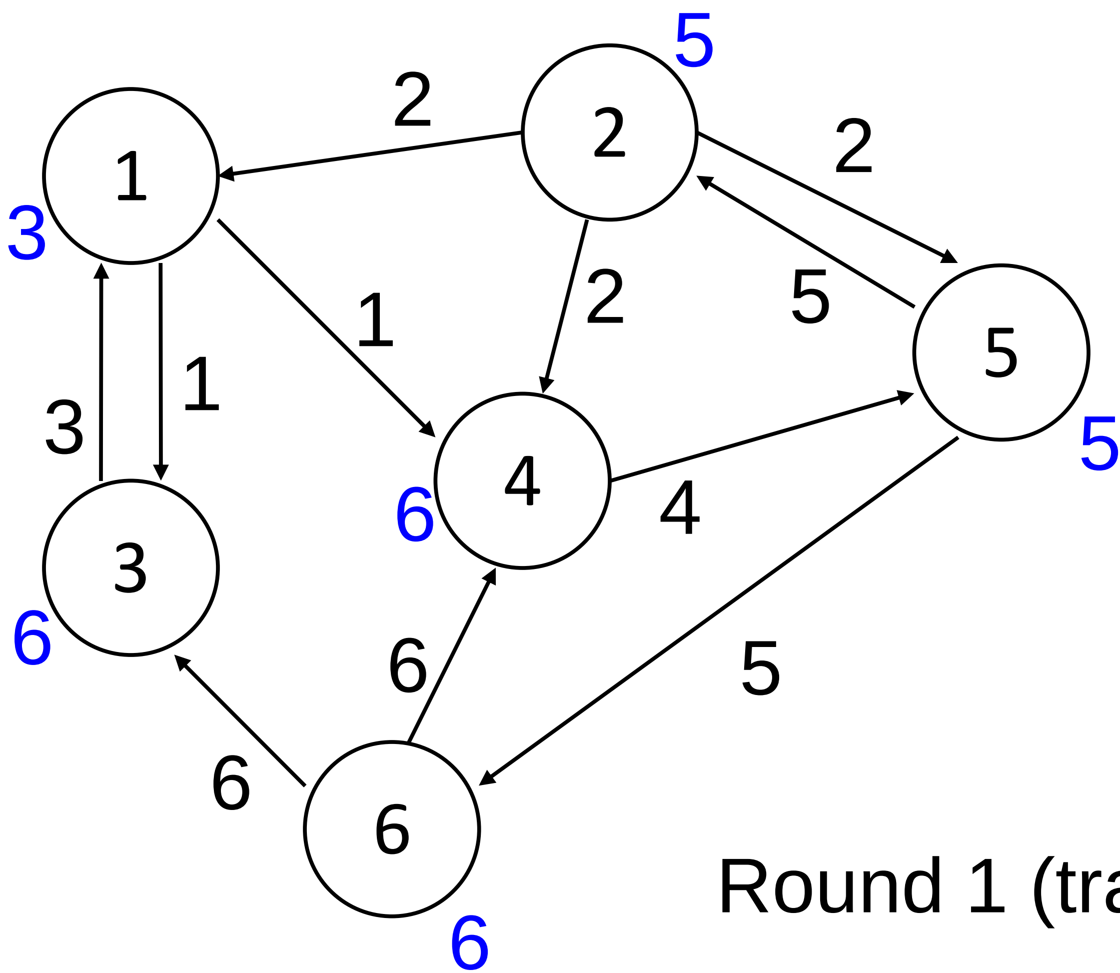
Start configuration

Basic flooding algorithm



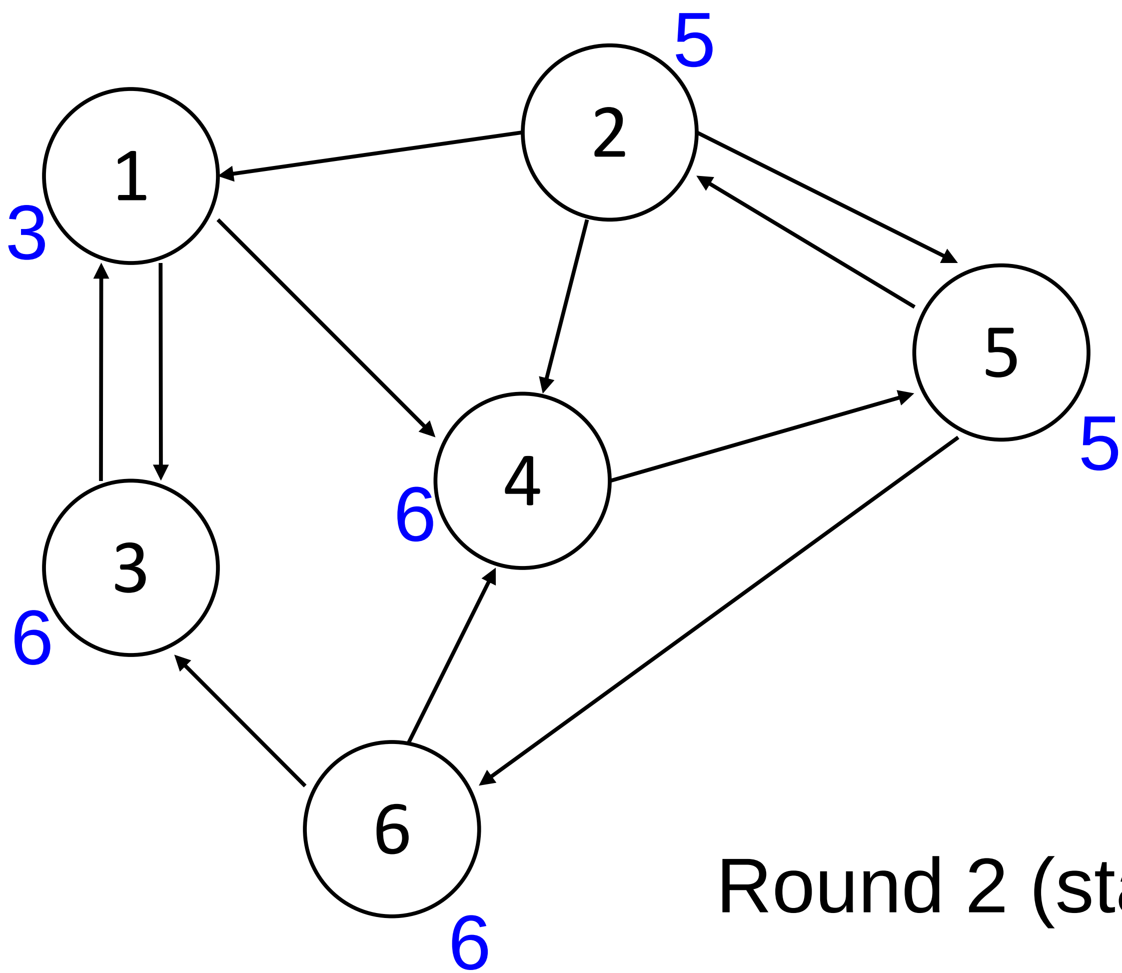
Round 1 (msgs)

Basic flooding algorithm



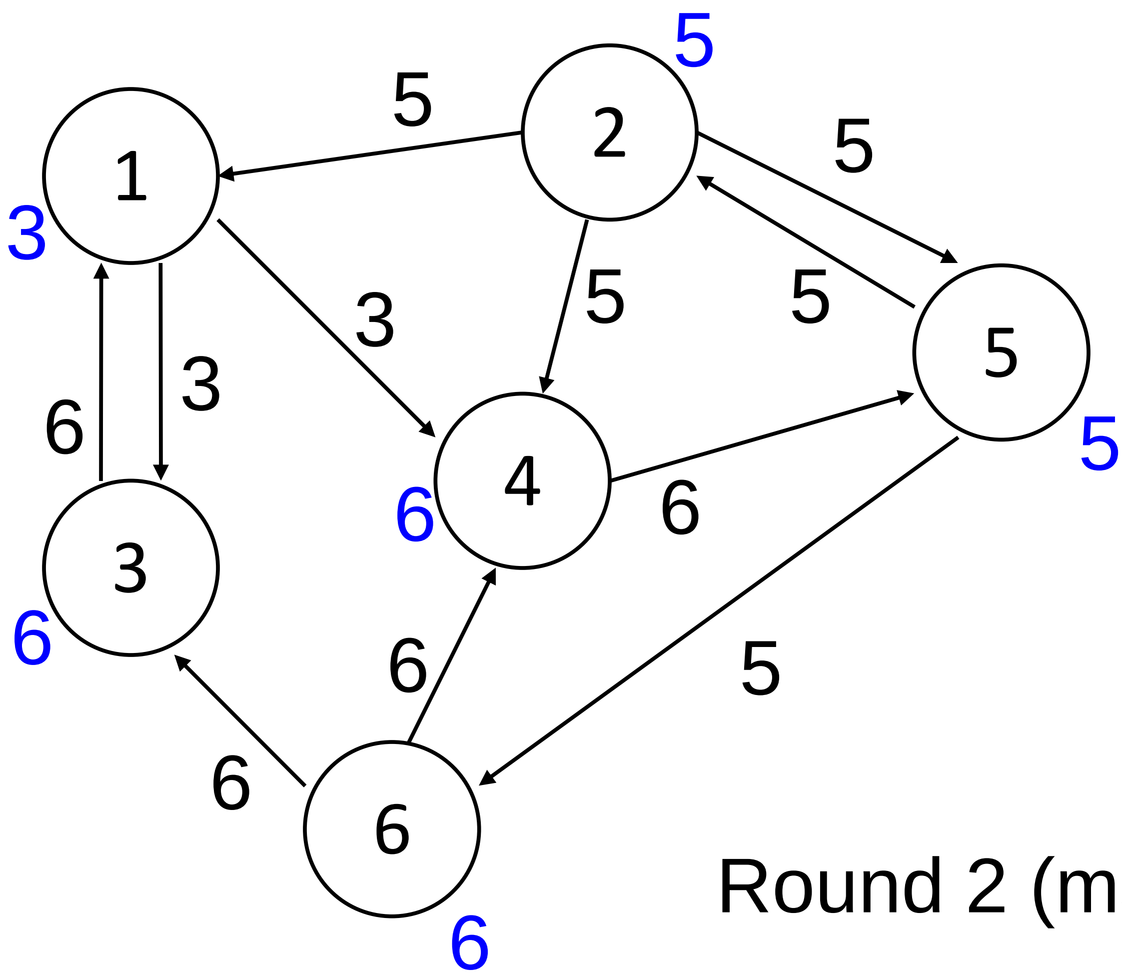
Round 1 (trans)

Basic flooding algorithm



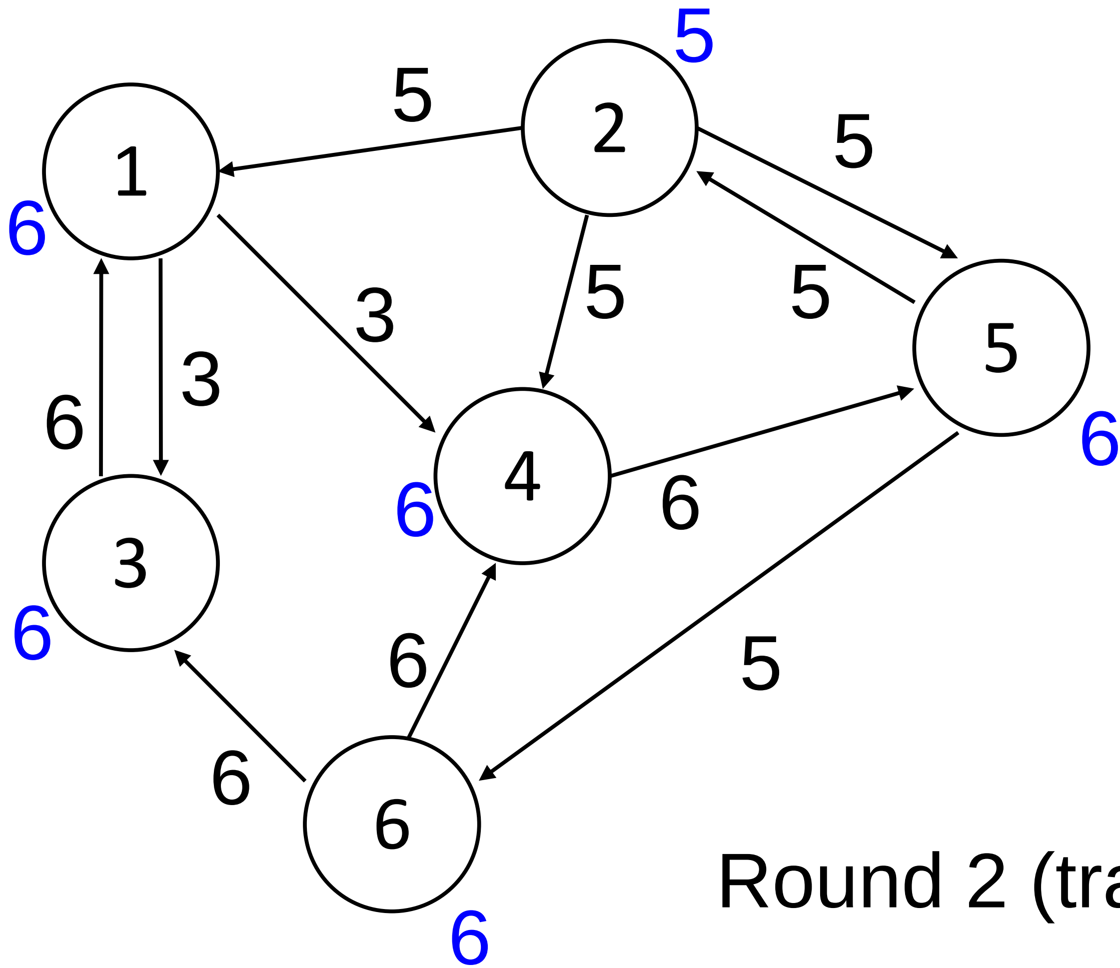
Round 2 (start)

Basic flooding algorithm



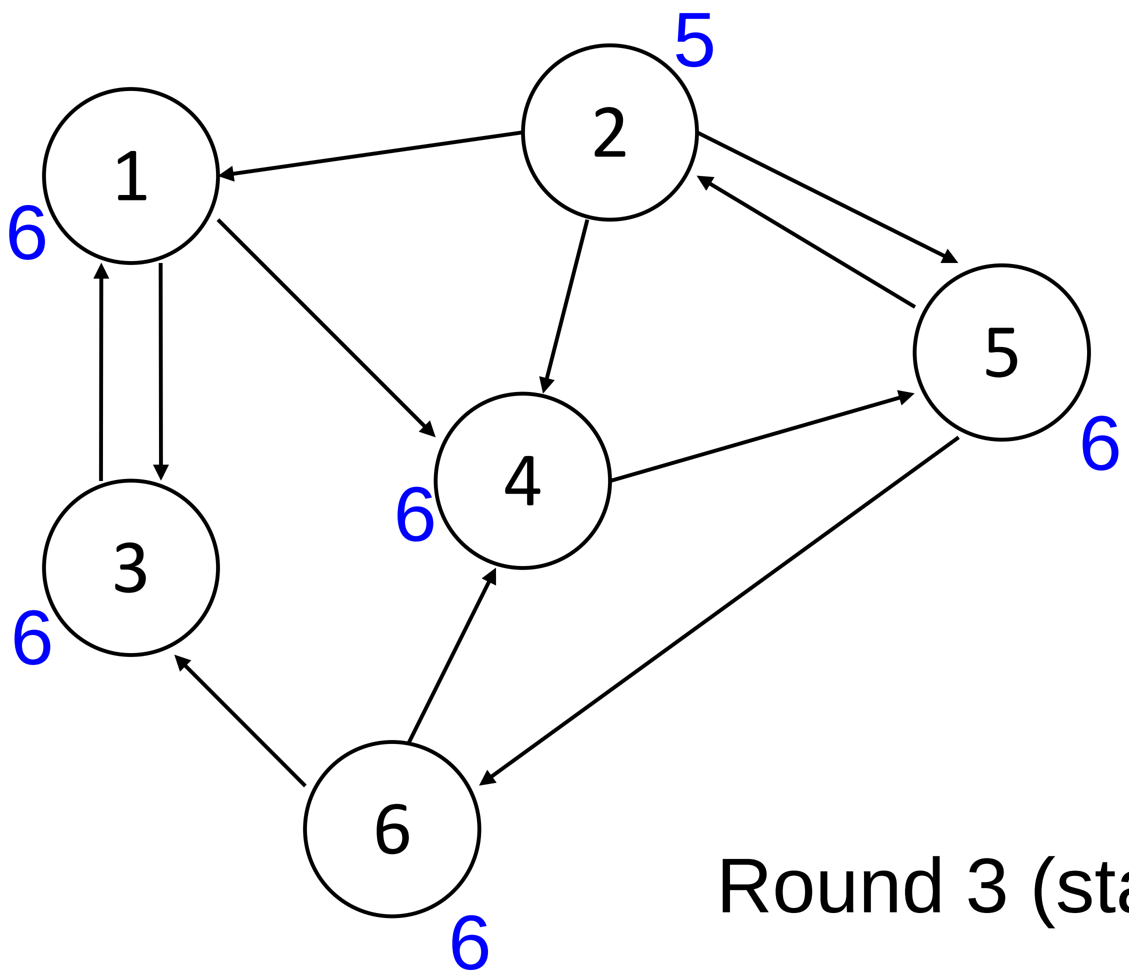
Round 2 (msgs)

Basic flooding algorithm



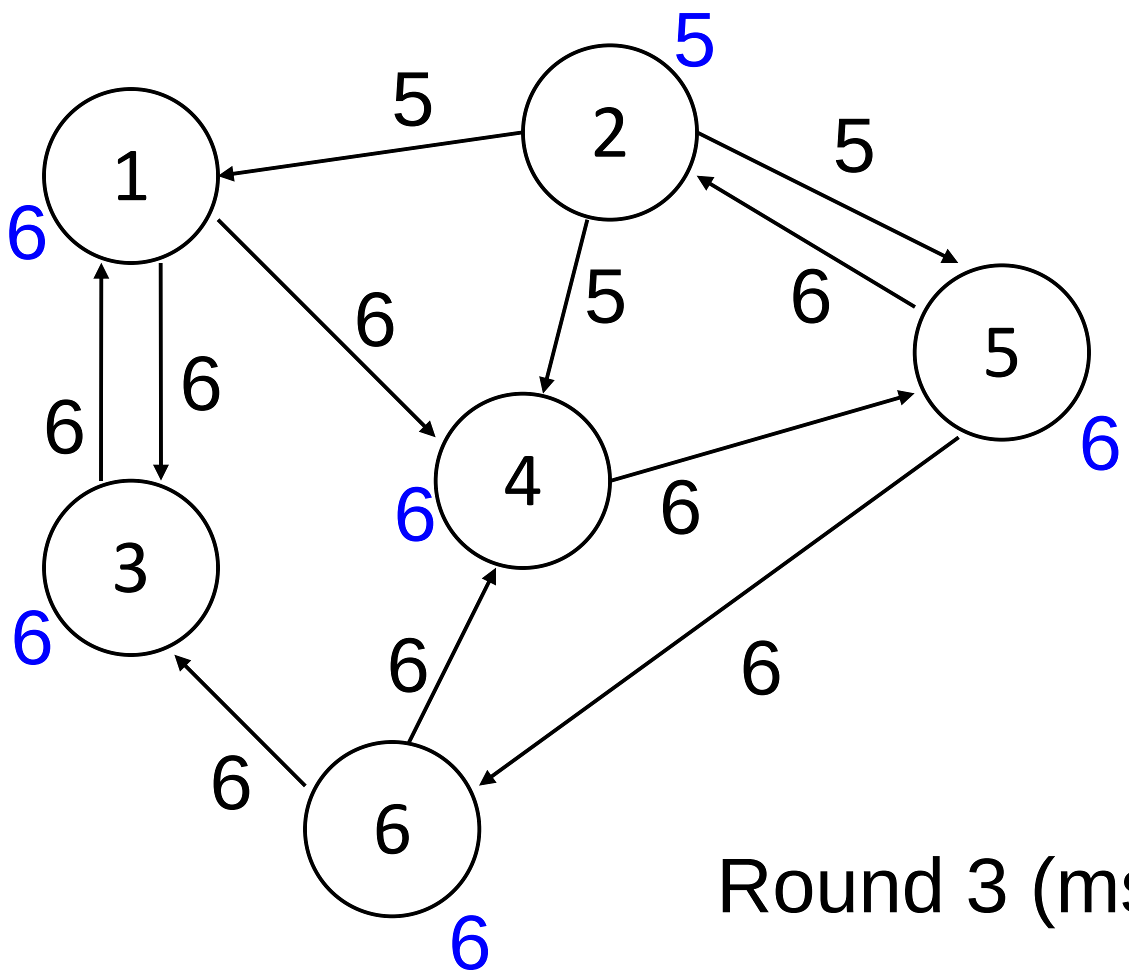
Round 2 (trans)

Basic flooding algorithm

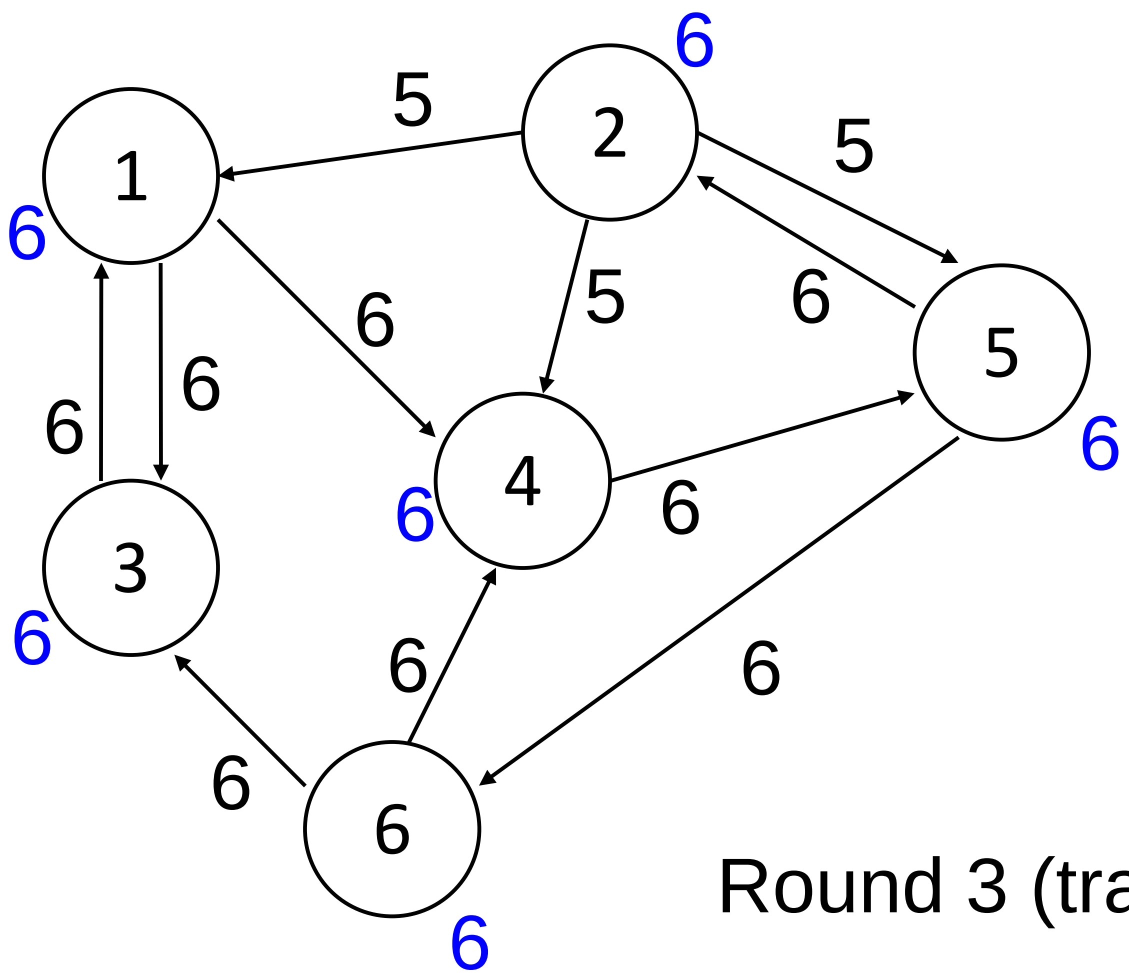


Round 3 (start)

Basic flooding algorithm

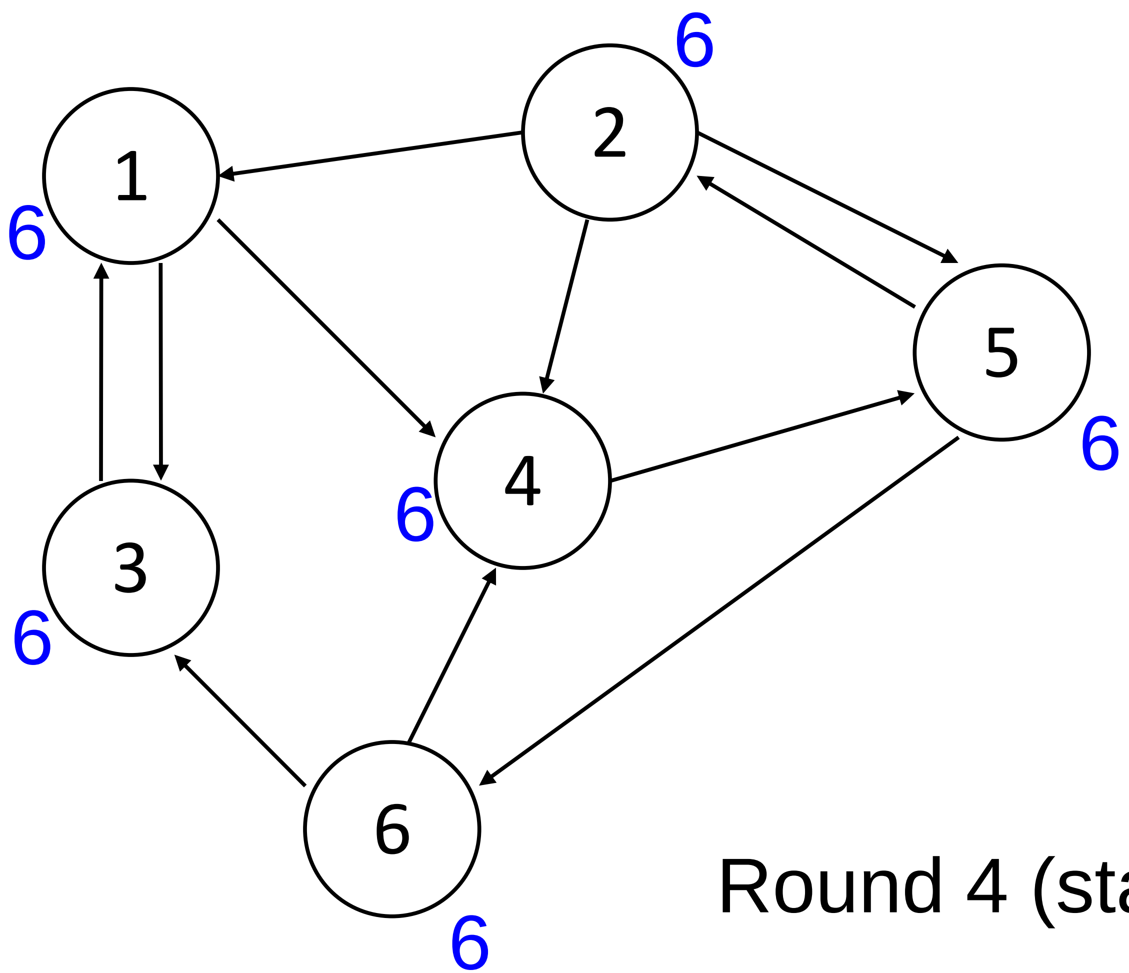


Basic flooding algorithm



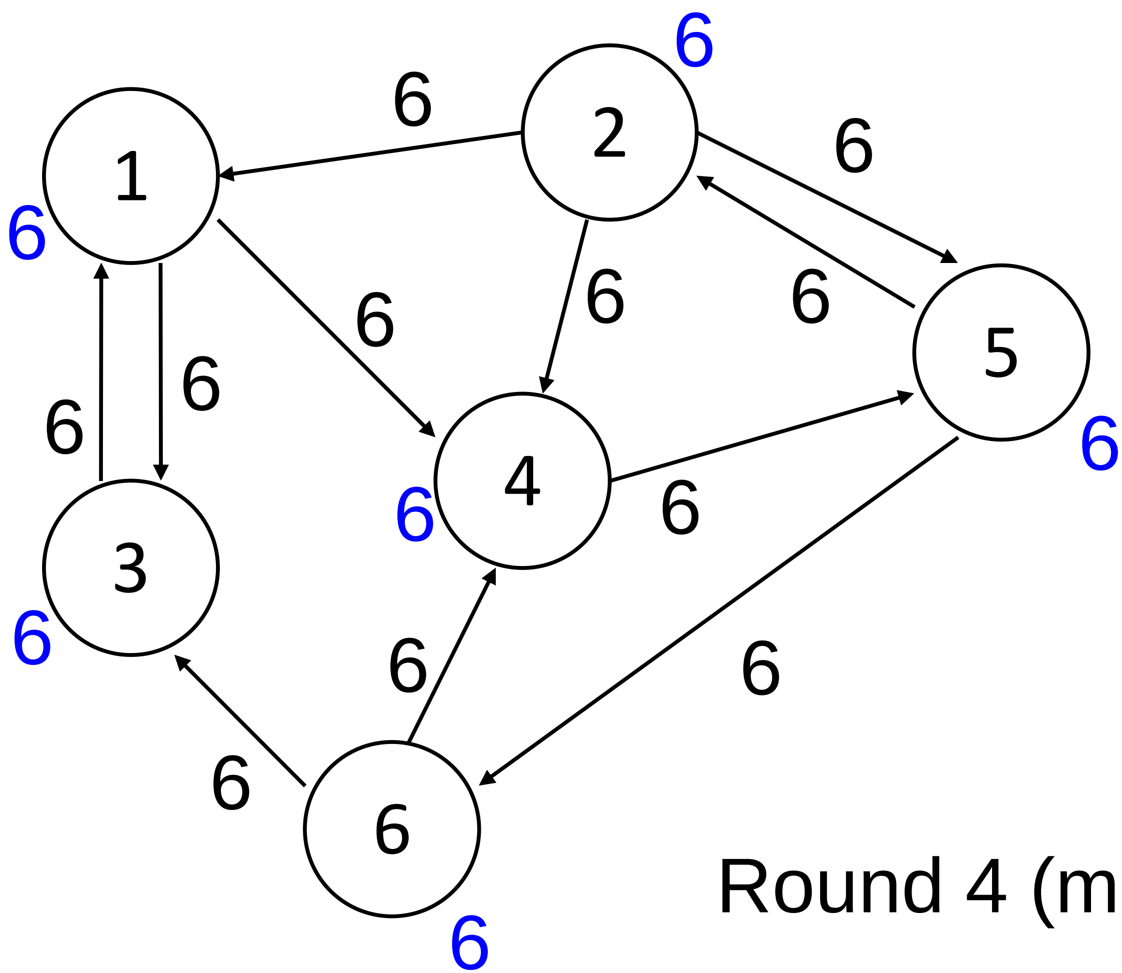
Round 3 (trans)

Basic flooding algorithm



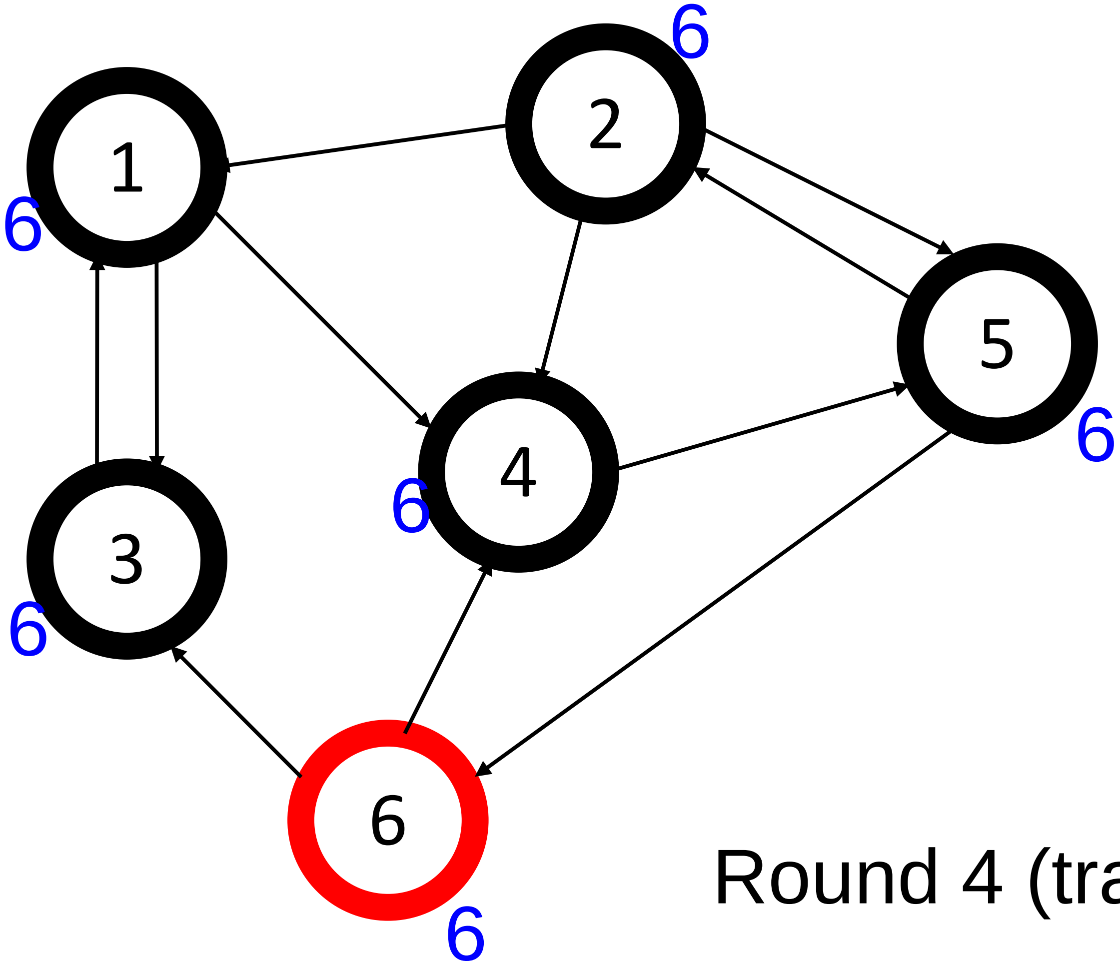
Round 4 (start)

Basic flooding algorithm



Round 4 (msgs)

Basic flooding algorithm



Round 4 (trans)

Basic flooding algorithm

- Algorithm:
  - Assume diameter is known ($diam$).
  - Every round: Send the max UID you have seen to all neighbors.
  - Stop after $diam$ rounds.
  - Elect self iff your own UID is the max you have seen.

- Complexity:
  - Time complexity (rounds): $diam$
  - Message complexity: $diam\ |E|$
- Correctness proof?

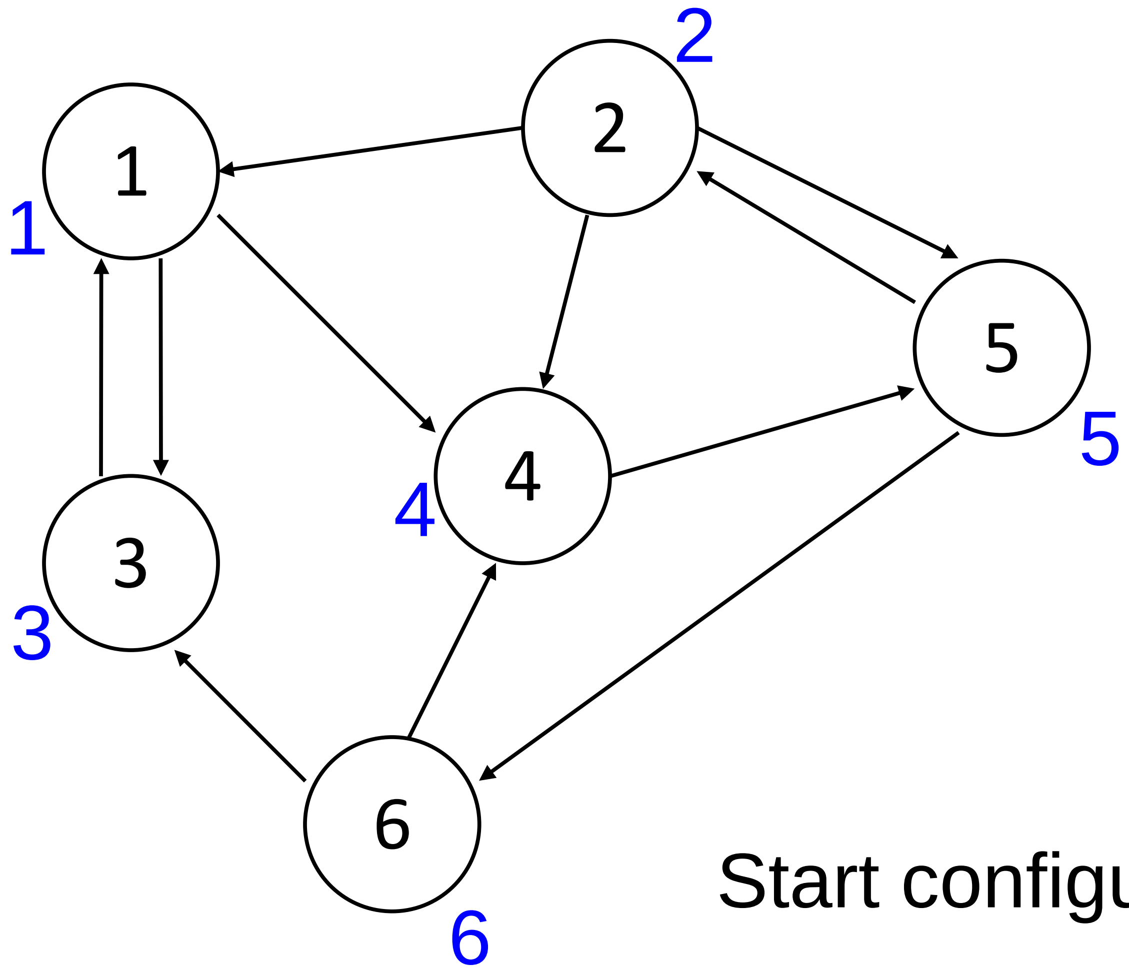
Key invariant

- Invariant: Just after round r, if $distance(i,j) \leq r$ then $maxuid_j \geq UID_i$.
- Proof:
  - Induction on r.
  - Base: $r = 0$
    - $distance(i,j) = 0$ implies $i = j$, and $maxuid_i = UID_i$.
  - Inductive step: Assume for $r - 1$, prove for r.
    - Assume $distance(i,j) \leq r$.
    - Then there is a node $k \in innbrs_j$ with $distance(i,k) \leq r - 1$.
    - By inductive hypotheses, after round $r - 1$, $maxuid_k \geq UID_i$.
    - Since k sends its $maxuid$ to j at round r, $maxuid_j \geq UID_i$ after round r.

Reducing the message complexity

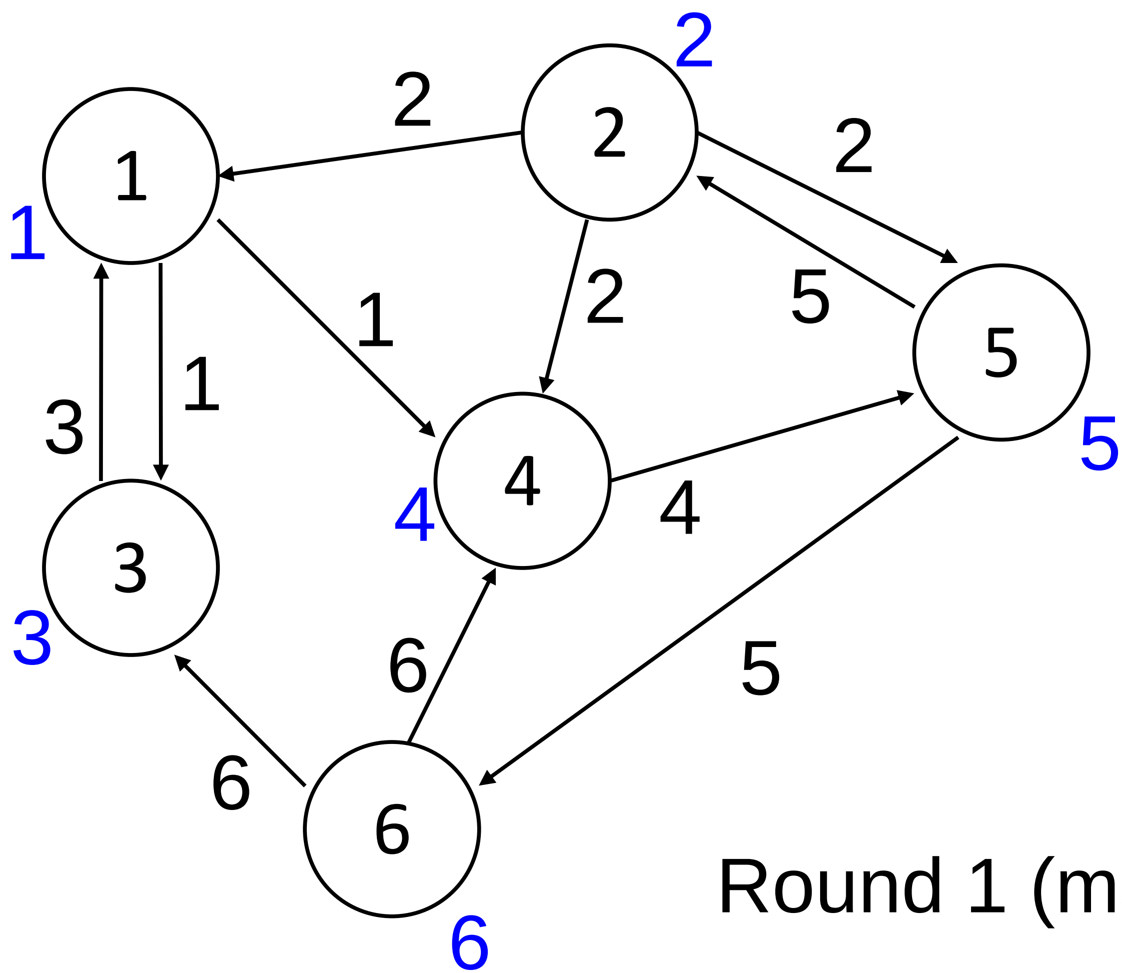
- Slightly improved algorithm:
  - Don't send same UID twice.
  - Additional state variable: $newinfo$, a Boolean, initially true
  - Send $maxuid$ only if $newinfo = $ true
  - Set $newinfo :=$ true iff the max UID received at this round > $maxuid.$

Improved algorithm



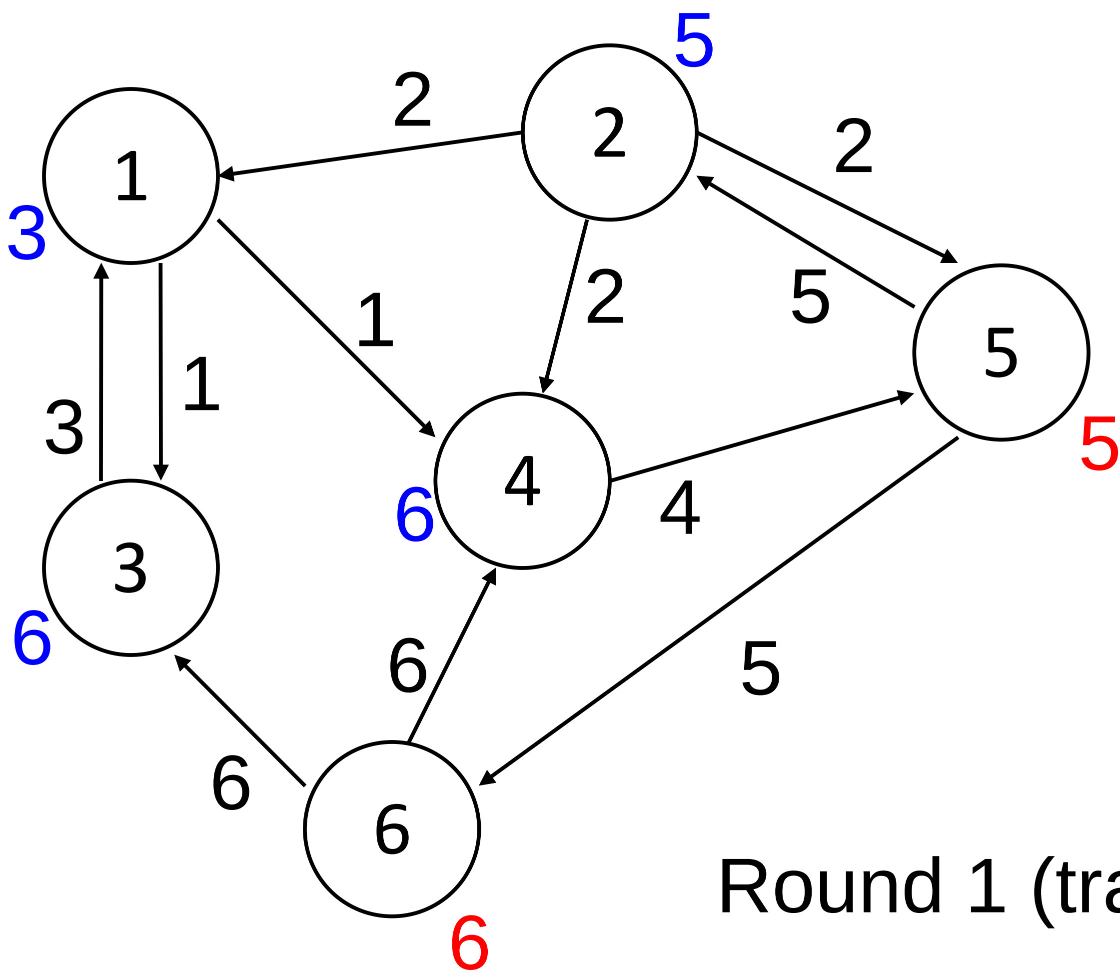
Start configuration

Improved algorithm



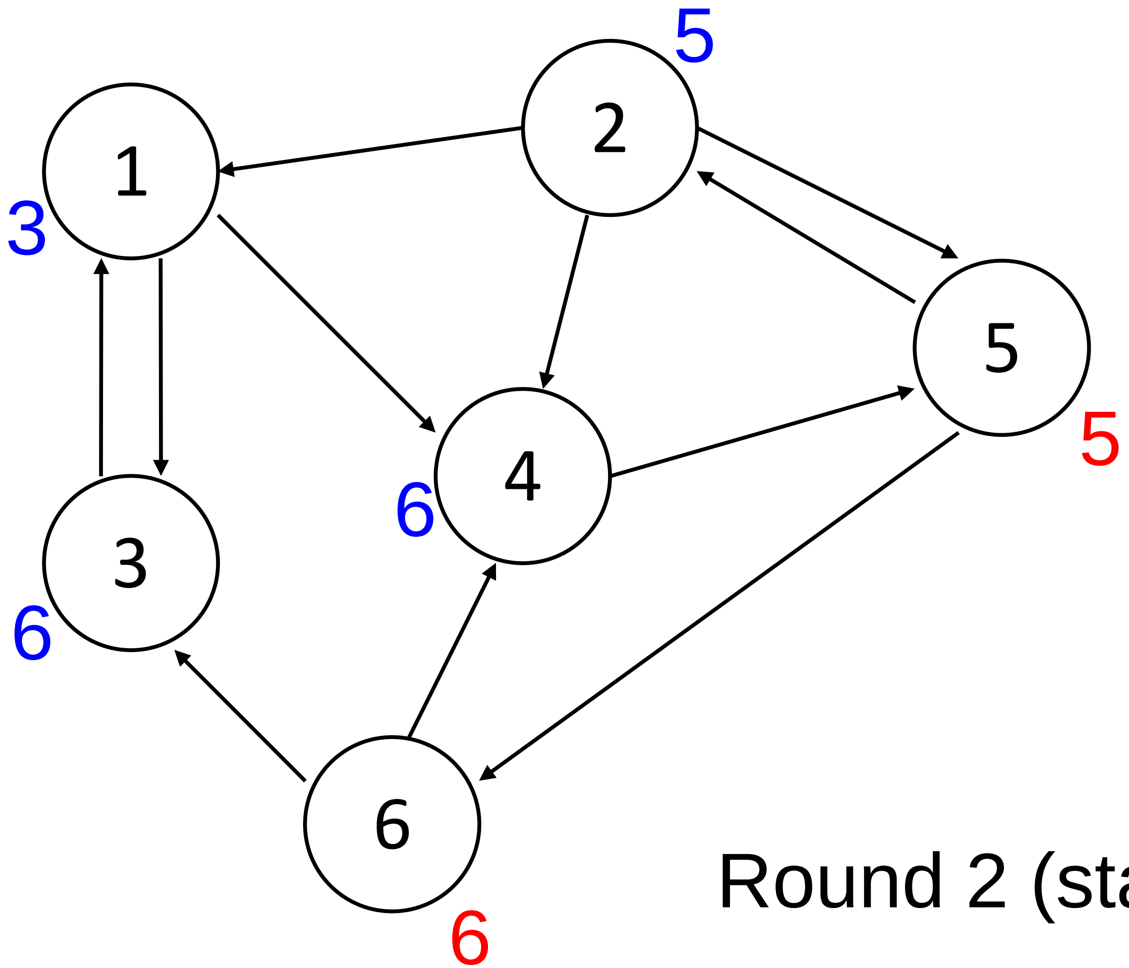
Round 1 (msgs)

Improved algorithm



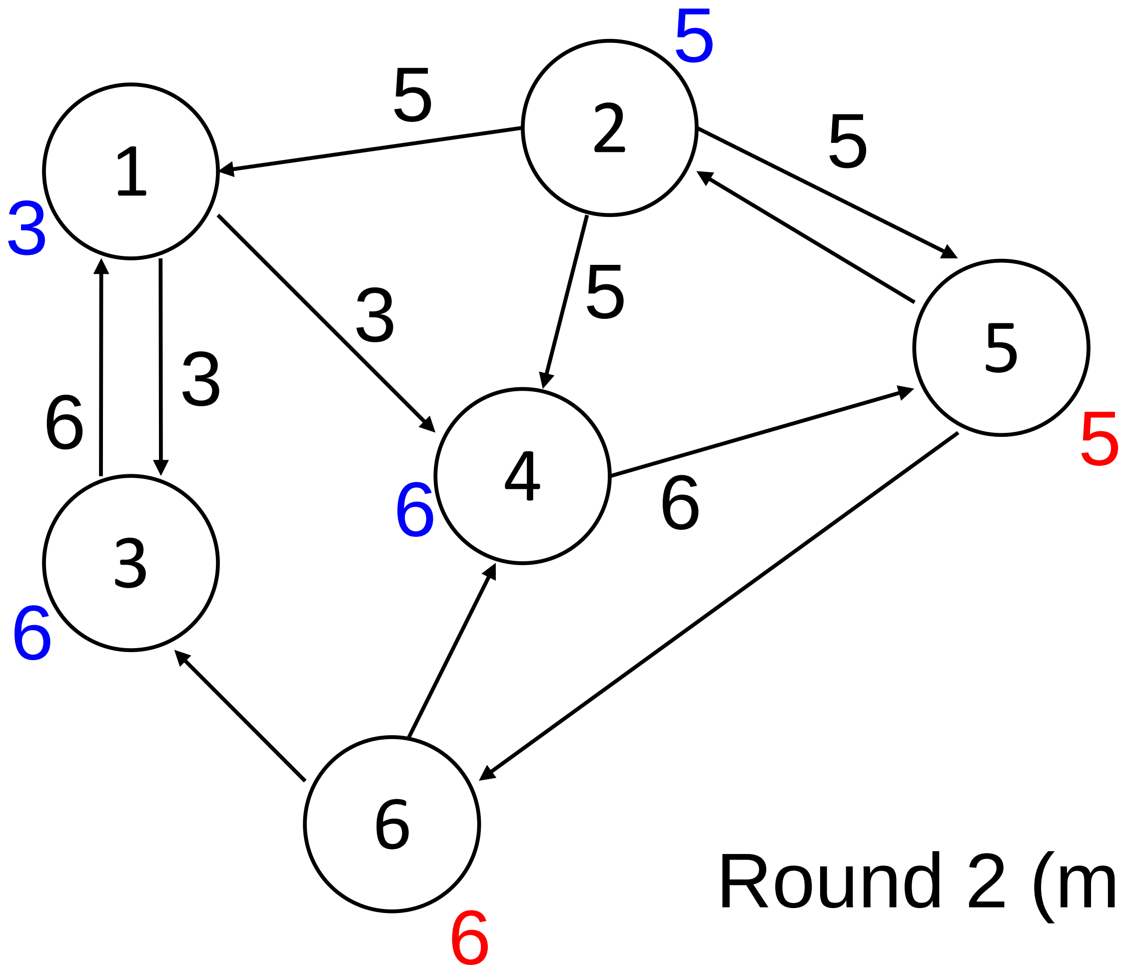
Round 1 (trans)

Improved algorithm



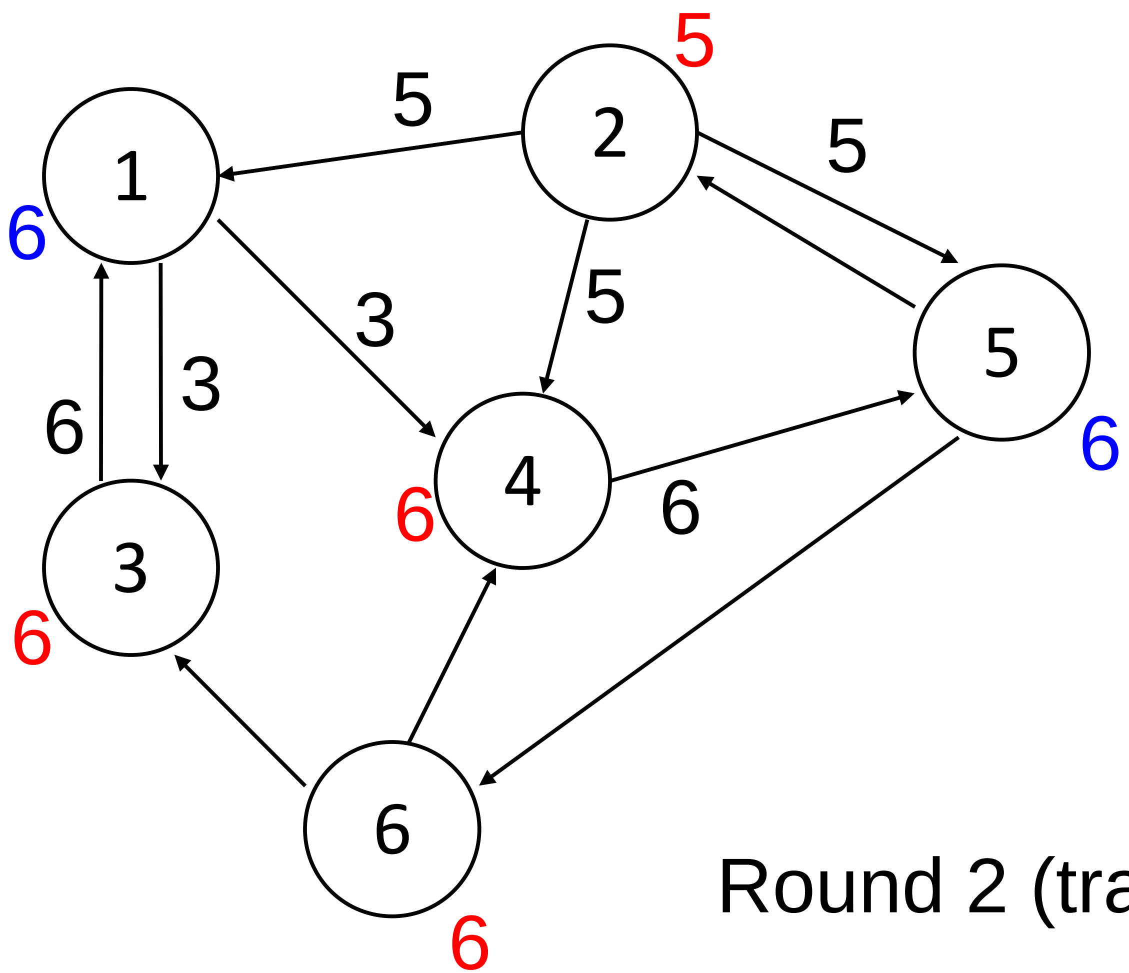
Round 2 (start)

Improved algorithm



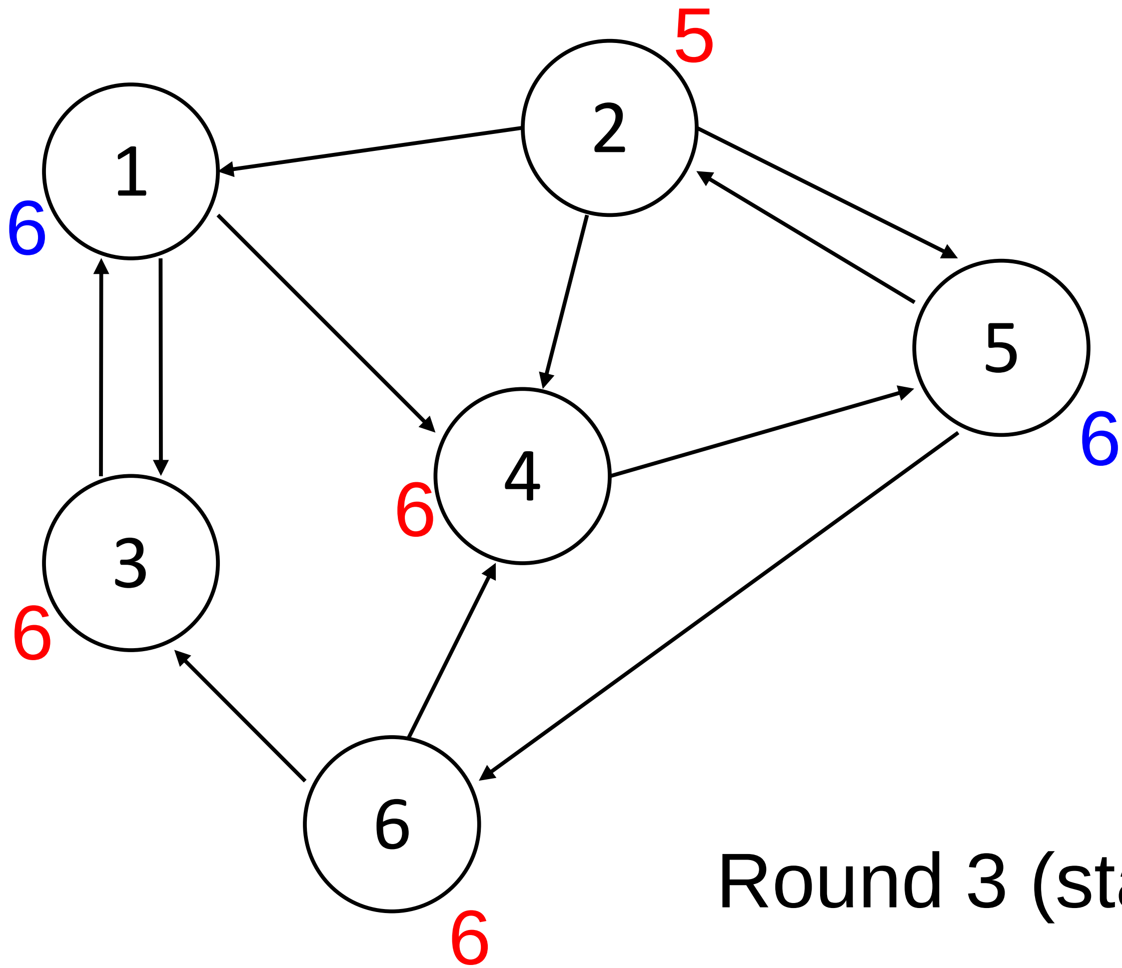
Round 2 (msgs)

Improved algorithm



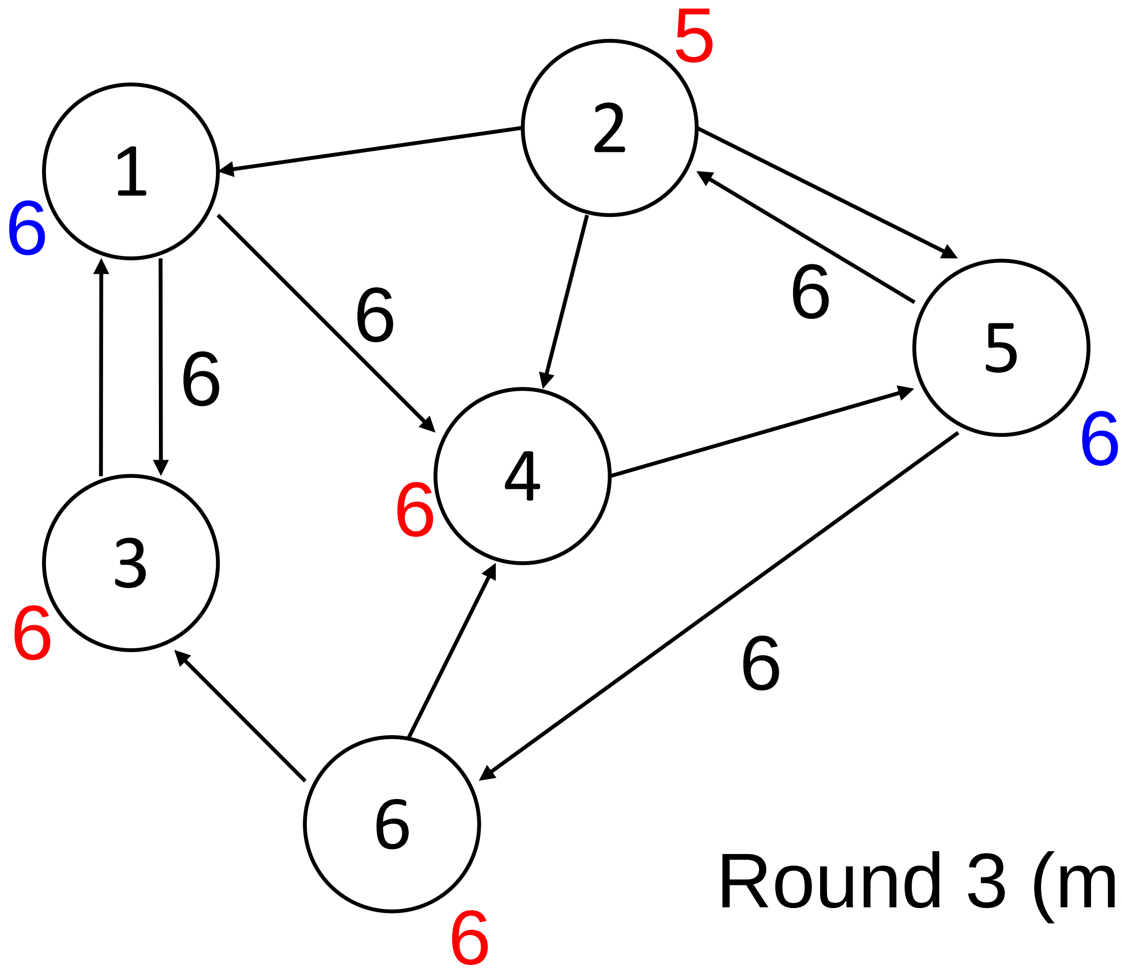
Round 2 (trans)

Improved algorithm



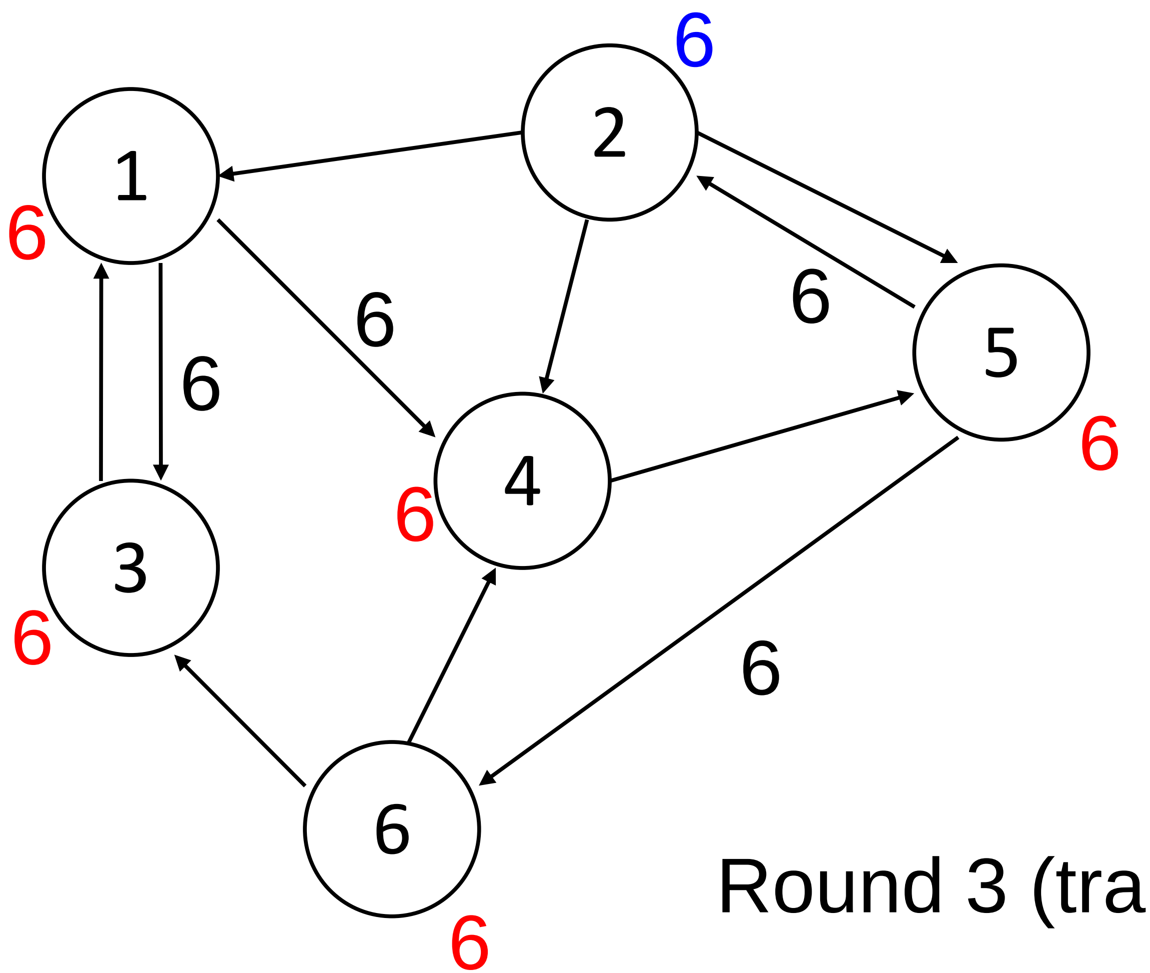
Round 3 (start)

Improved algorithm



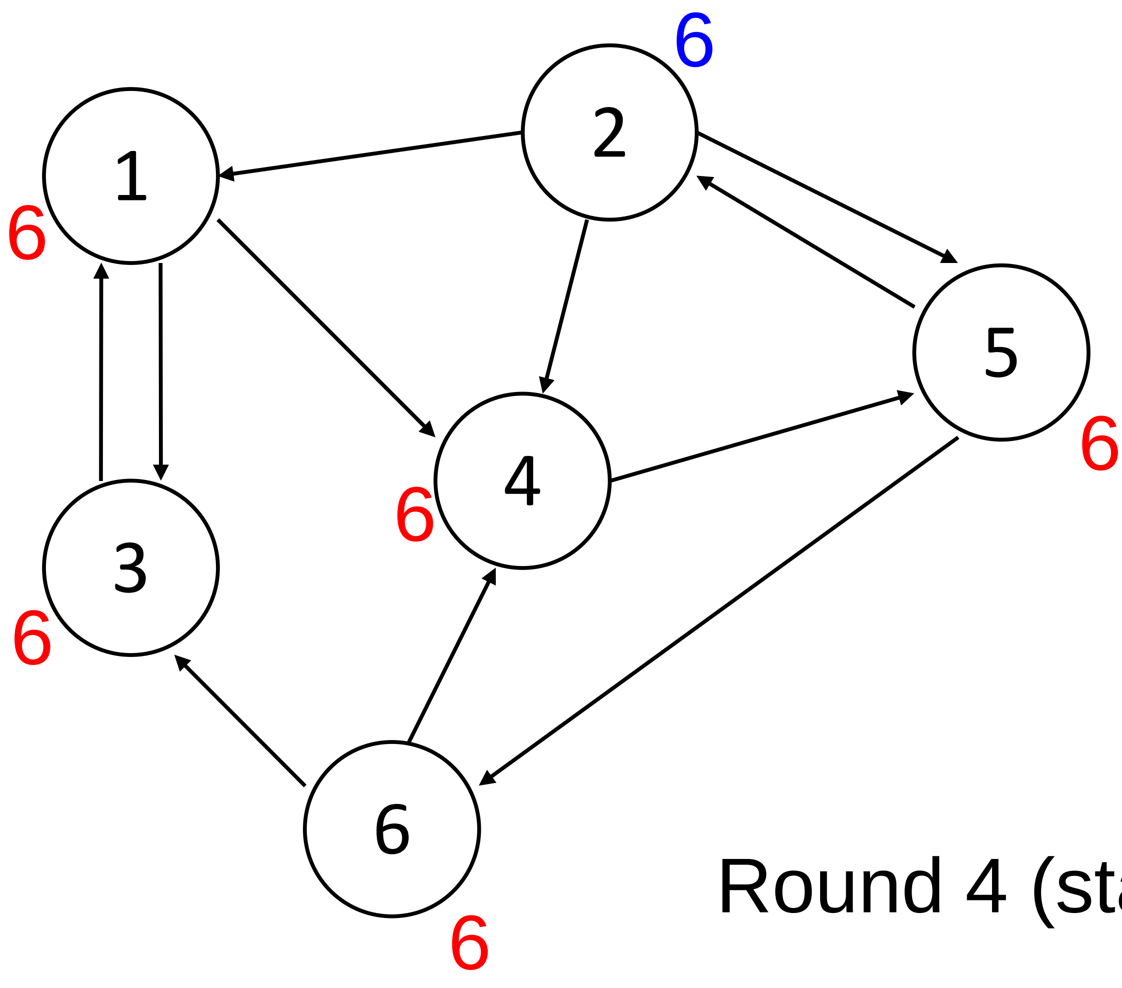
Round 3 (msgs)

Improved algorithm



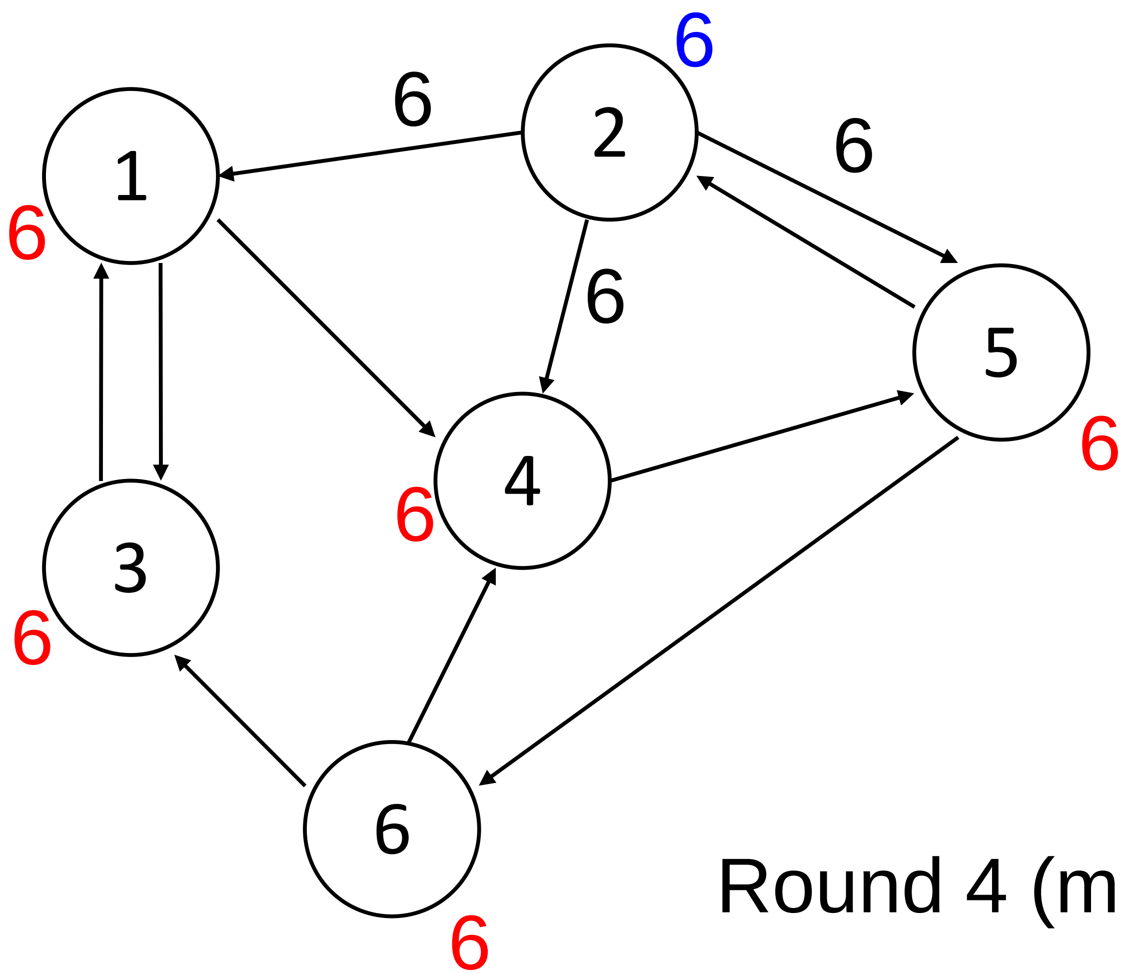
Round 3 (trans)

Improved algorithm



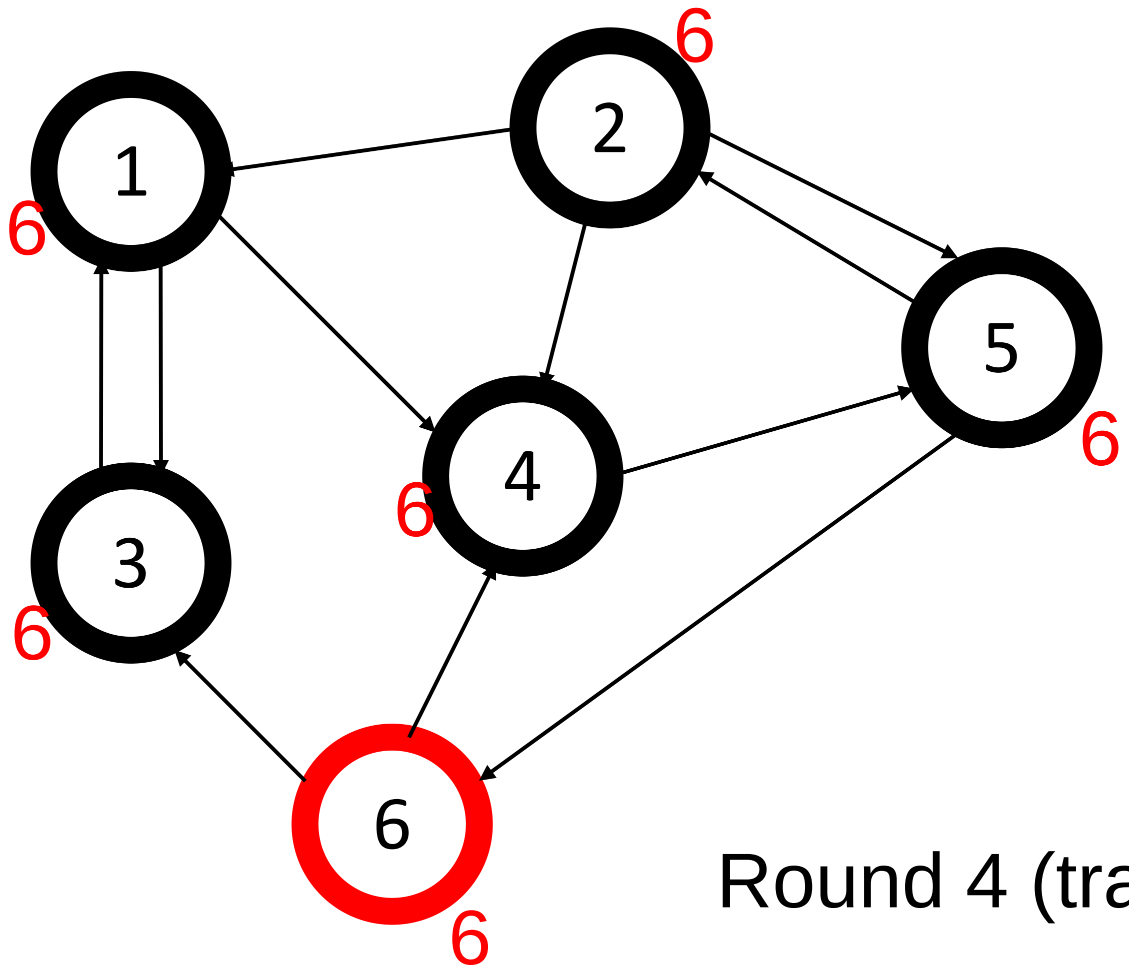
Round 4 (start)

Improved algorithm



Round 4 (msgs)

Improved algorithm



Round 4 (trans)

Improved algorithm

- Improved algorithm:
  - Don't send same UID twice.
  - New state variable: $newinfo$, a Boolean, initially true
  - Send $maxuid$ only if $newinfo = \text{true}$
  - $newinfo :=$ true iff the max UID received at this round is strictly greater than $maxuid$
- Algorithm sometimes improves communication cost significantly, but the worst-case bound is the same, $diam\ |E|$.
- Correctness Proof:
  - Can prove this similarly to before.
  - Or, we can use another important method for proving correctness of distributed algorithms: Simulation Relations.

Simulation relation

- Relates a new algorithm formally to an original one that has already been proved correct.
- Correctness then carries over from the old algorithm to the new algorithm.
- Often used to show correctness of optimized algorithms.
- Can repeat this in several stages, adding more optimizations.

- "Run the two algorithms side by side and relate them."
- Define a simulation relation between states of the two algorithms:
  - Satisfied by start states.
  - Preserved by every transition.
  - Outputs should be the same from related states.

Simulation relation between the improved and basic algorithms

- Key invariant of the improved algorithm:
  - If $i \in innbrs_j$ and $maxuid_i > maxuid_j$ then $newinfo_i$ = true.
  - That is, if i has better information than j, then i is planning to send it to j on the next round.
  - Can prove this by induction on the number of rounds.
- Simulation relation: All state variables of the basic algorithm (all but $newinfo$) have the same values in both algorithms.
- Start condition: By definition.
- Preserved by every transition:
  - Key property: $maxuid$s are always the same in the two algorithms.
  - Consider $i \in innbrs_j$.
  - If $newinfo_i$ = true before the step, then the two algorithms behave the same with respect to (i, j).
  - Otherwise, only the basic algorithm sends a message. However, by the key invariant, this means that $maxuid_i \leq maxuid_j$ before the step, and so the message has no effect in the basic algorithm anyway.

Why all these proofs?

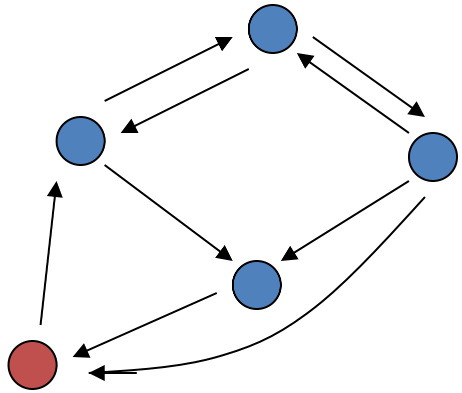
- Distributed algorithms can be very subtle and complicated.
- Easy to make mistakes.
- Careful reasoning about algorithm steps is generally needed.
- It's more necessary here than for sequential algorithms.

- Moreover, we prefer proofs that are systematic, like invariant and simulation relation proofs.
- Structure makes it easier to design (and read) new proofs.
- Makes it possible to keep track of numerous details.
- Proofs lend themselves to machine assistance, using theorem-provers, model-checkers, etc.

Now, other problems besides leader election…

- This week:
  – Breadth-First Search (BFS), B-F spanning trees
  – Shortest-paths spanning treed
  – Minimum Spanning Trees (MSTs)
  – Maximal Independent Sets (MISs)
- Next week (Stephan Holzer):
  – MIS, revisited
  – Graph coloring
  – MST, revisited

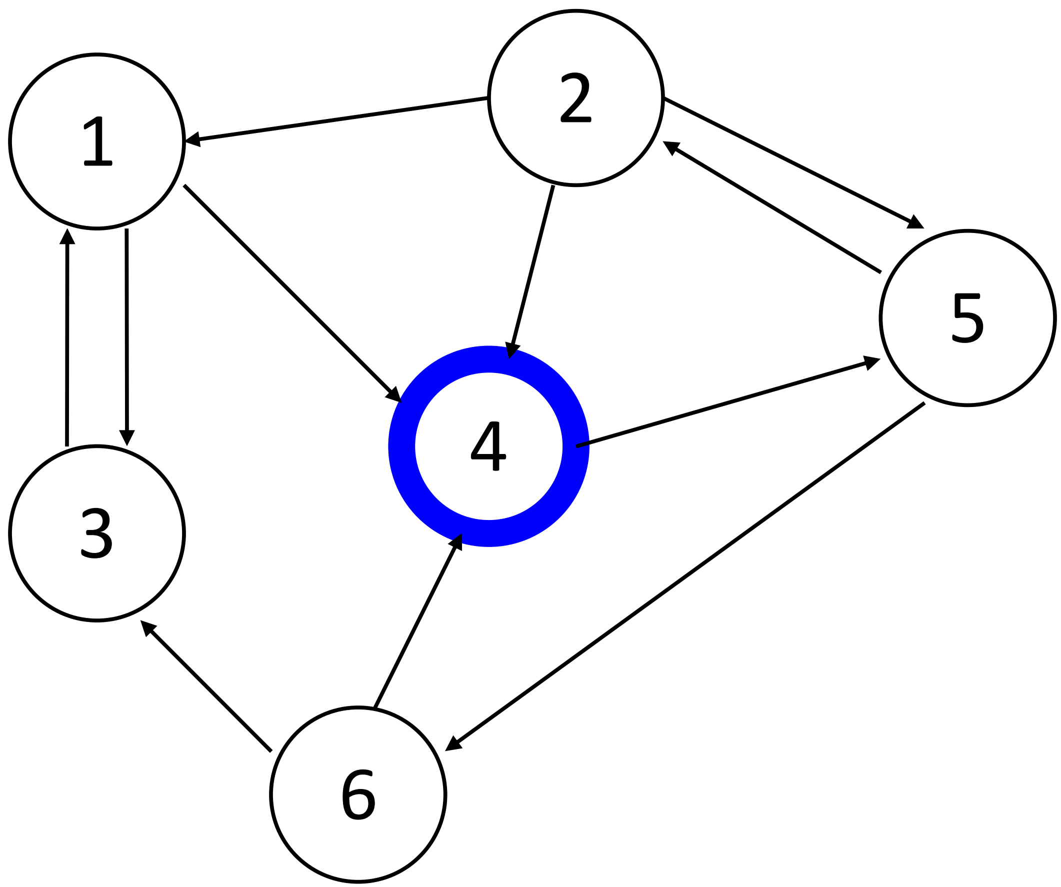
Breadth-First Search



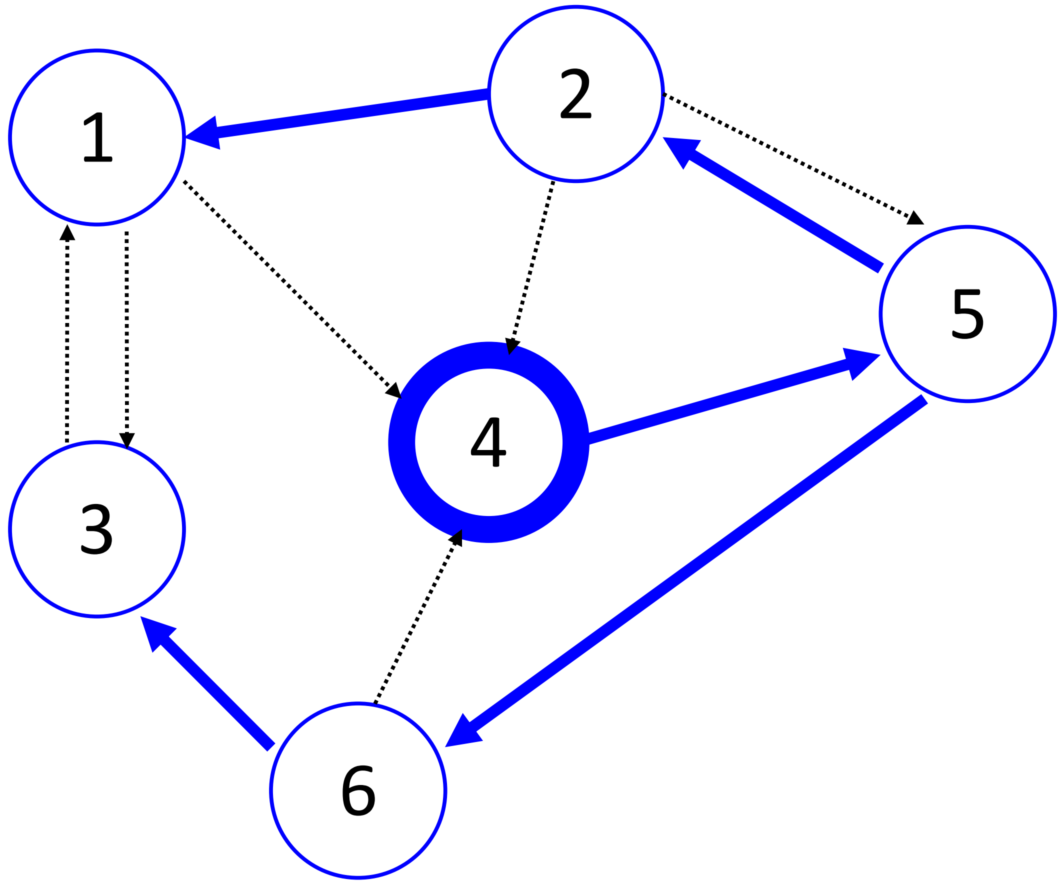
Breadth-first search

- Assume:
  - Strongly connected digraph, UIDs.
  - No knowledge of size or diameter of the network.
  - Distinguished source node (leader) i_0.
- Required: Breadth-first spanning tree, rooted at source node i_0.
  - Branches are directed paths in the given digraph.
  - Spanning: Includes every node.
  - Breadth-first: Node at distance d from i_0 appears at depth d in tree.
  - Output: Each node (except i_0) sets a $parent$ variable to indicate its parent in the tree.

Breadth-first search



Breadth-first search

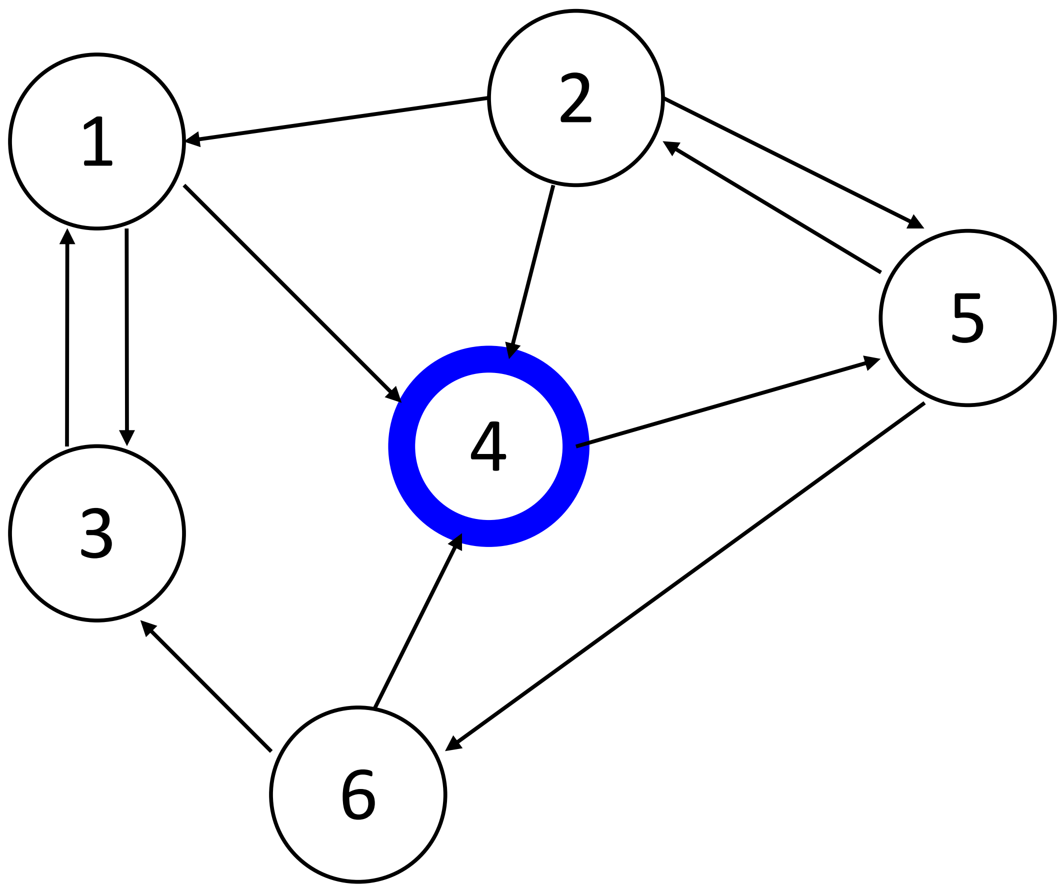


Breadth-first search algorithm

- Mark nodes as they get incorporated into the tree.
- Initially, only i_0 is marked.
- Round 1: i_0 sends $search$ message to out-nbrs.
- At every round: An unmarked node that receives a $search$ message:
    - Marks itself.
    - Designates one process from which it received $search$ as its parent.
    - Sends $search$ to out-nbrs at the next round.

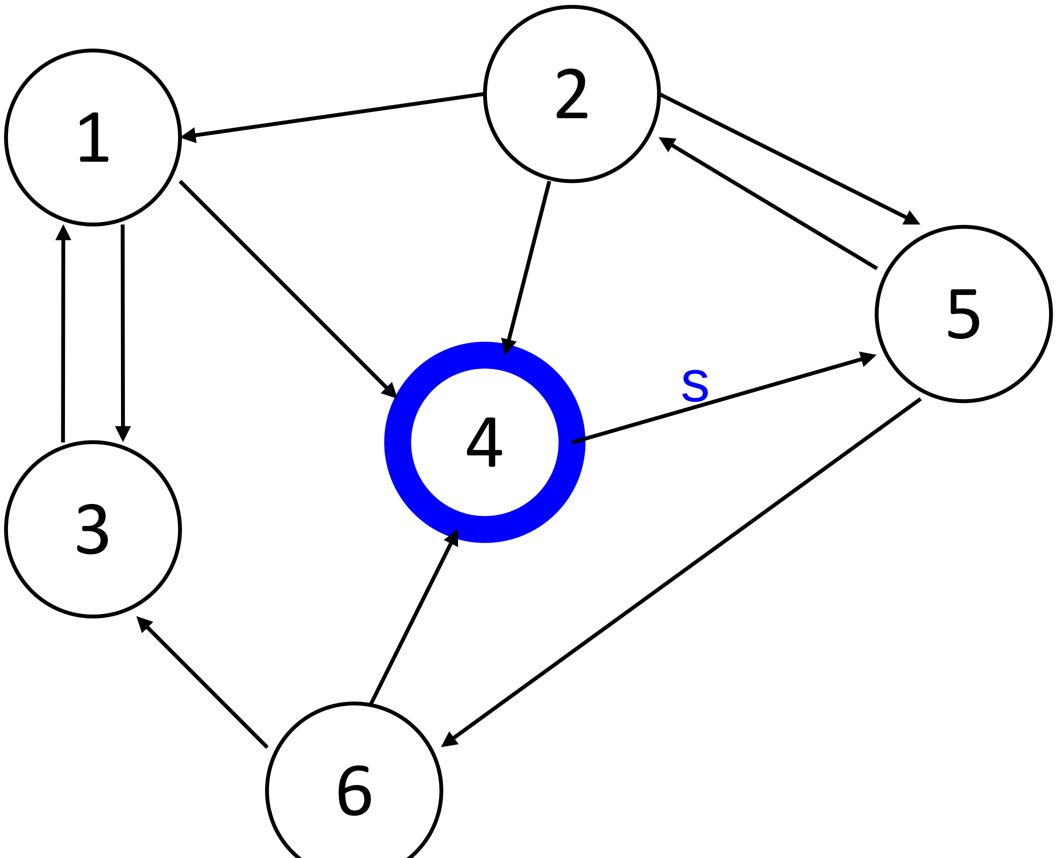
- Q: What state variables do we need?
- Q: Why does this yield a BFS tree?

Breadth-first search



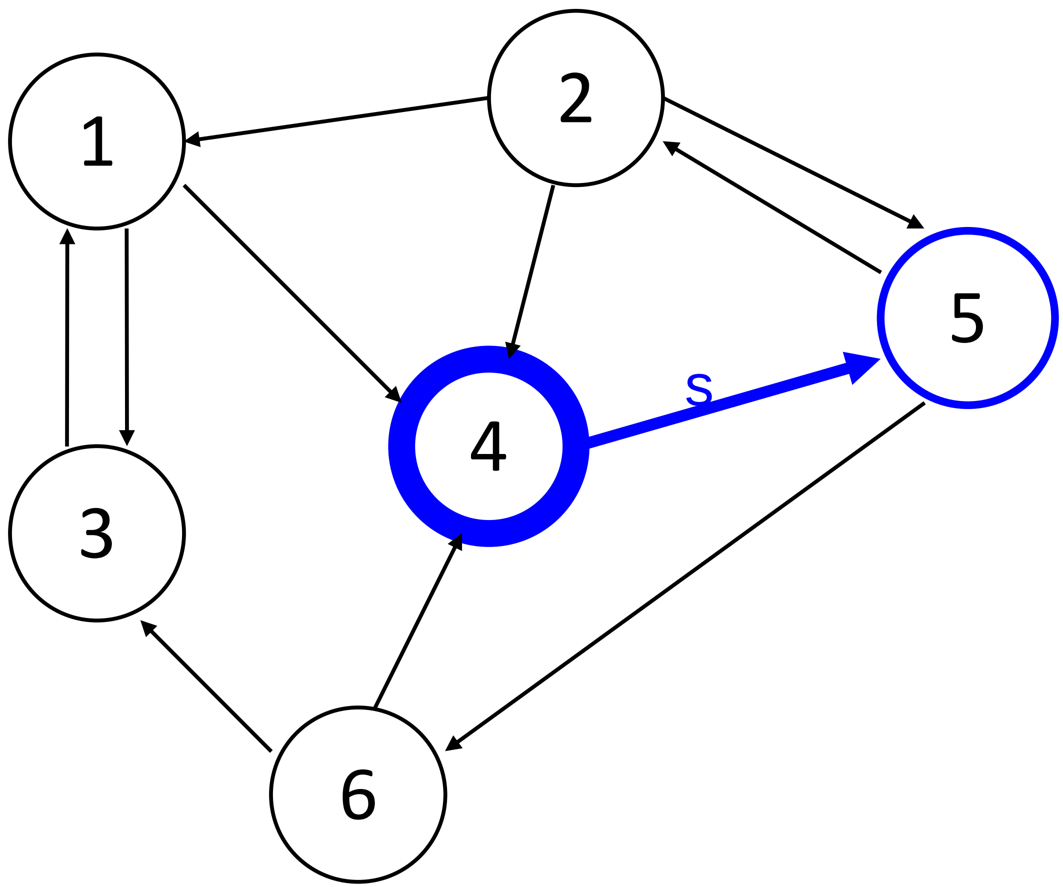
Round 1 (start)

Breadth-first search



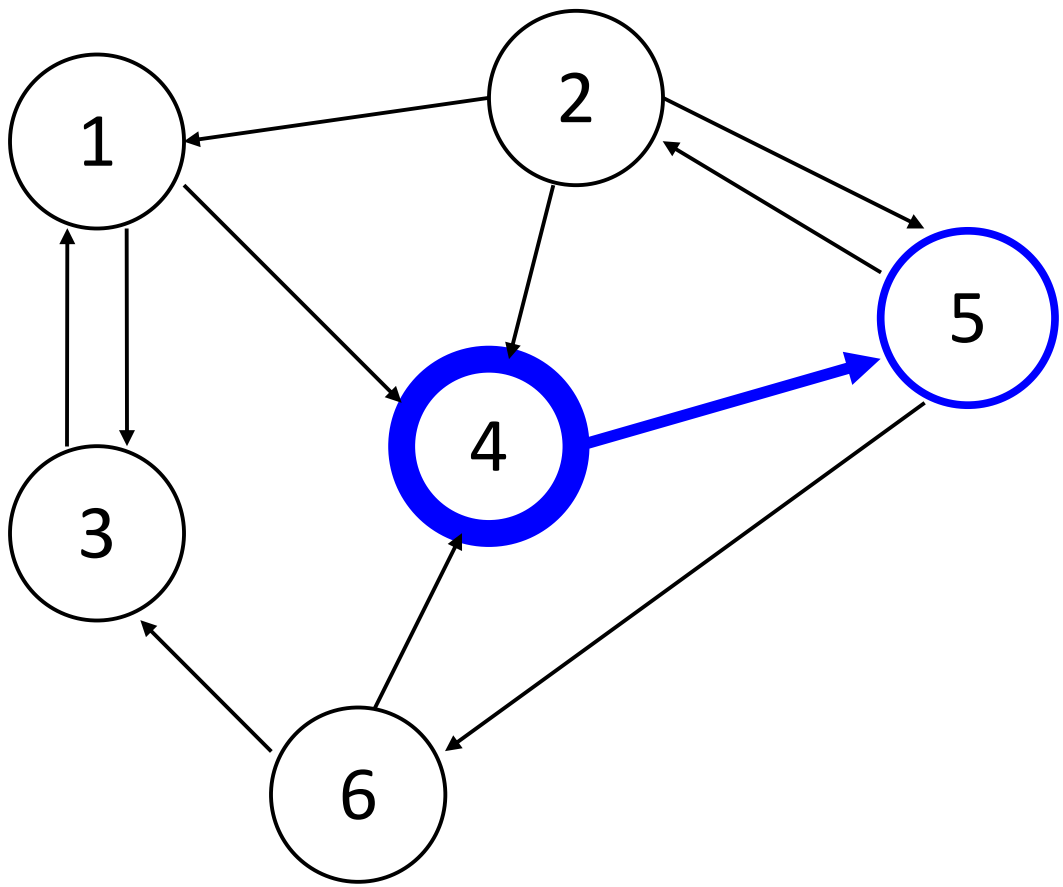
Round 1 (msgs)

Breadth-first search



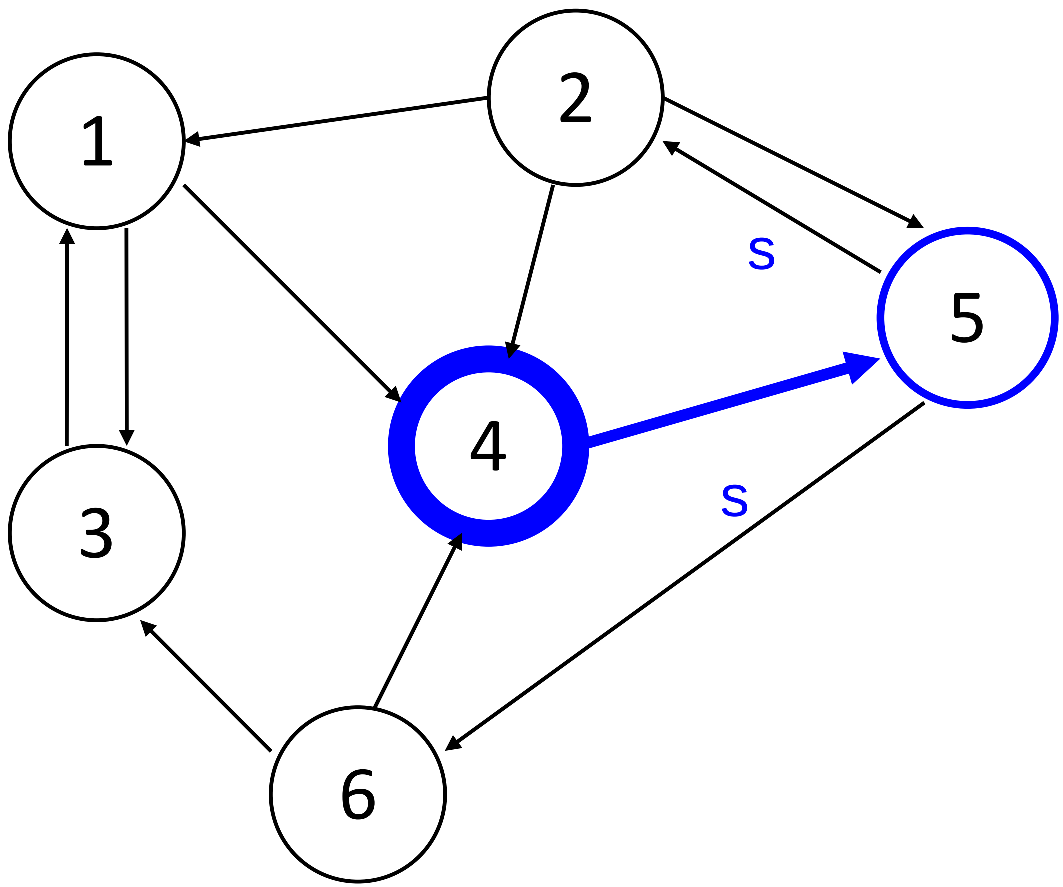
Round 1 (trans)

Breadth-first search



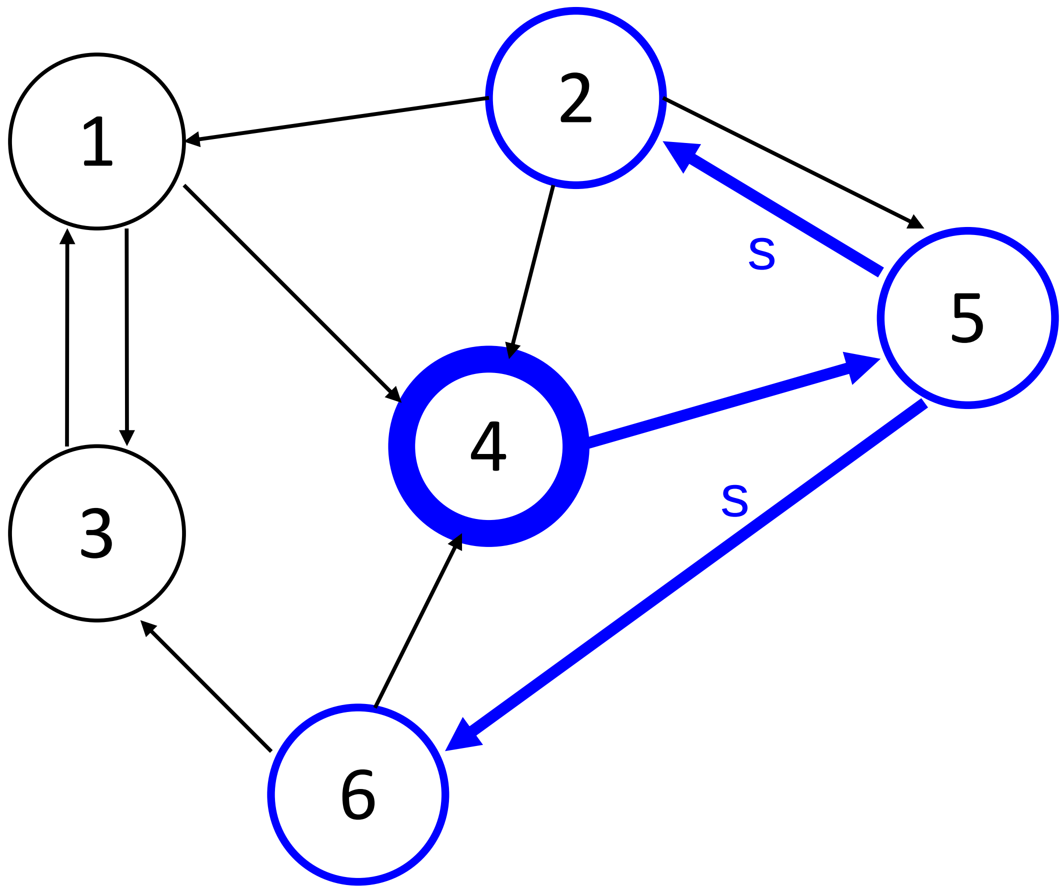
Round 2 (start)

Breadth-first search



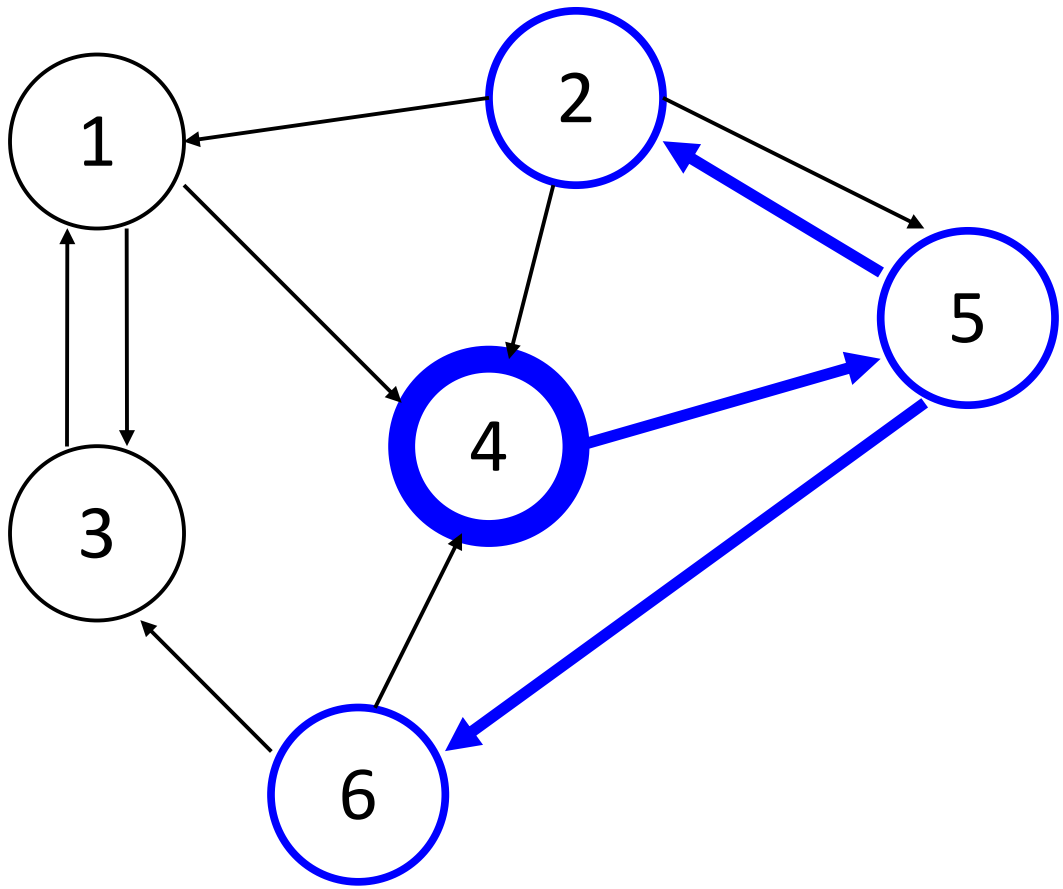
Round 2 (msgs)

Breadth-first search



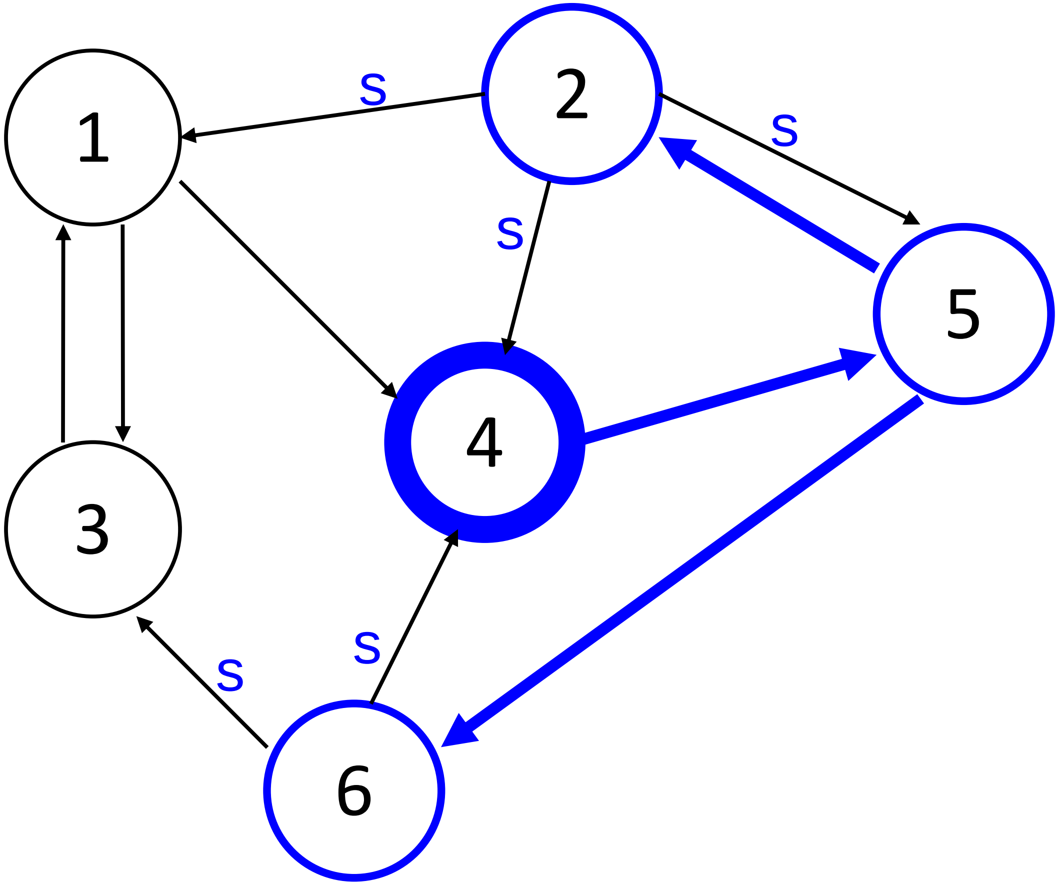
Round 2 (trans)

Breadth-first search



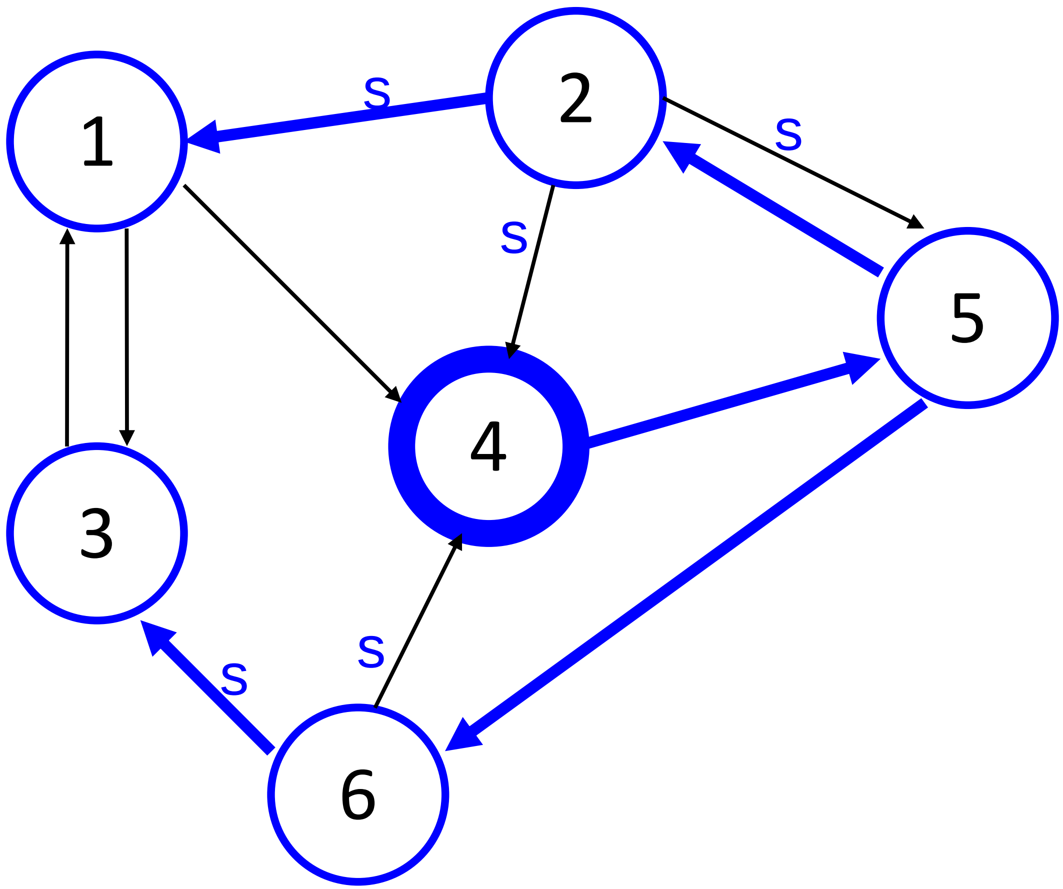
Round 3 (start)

Breadth-first search



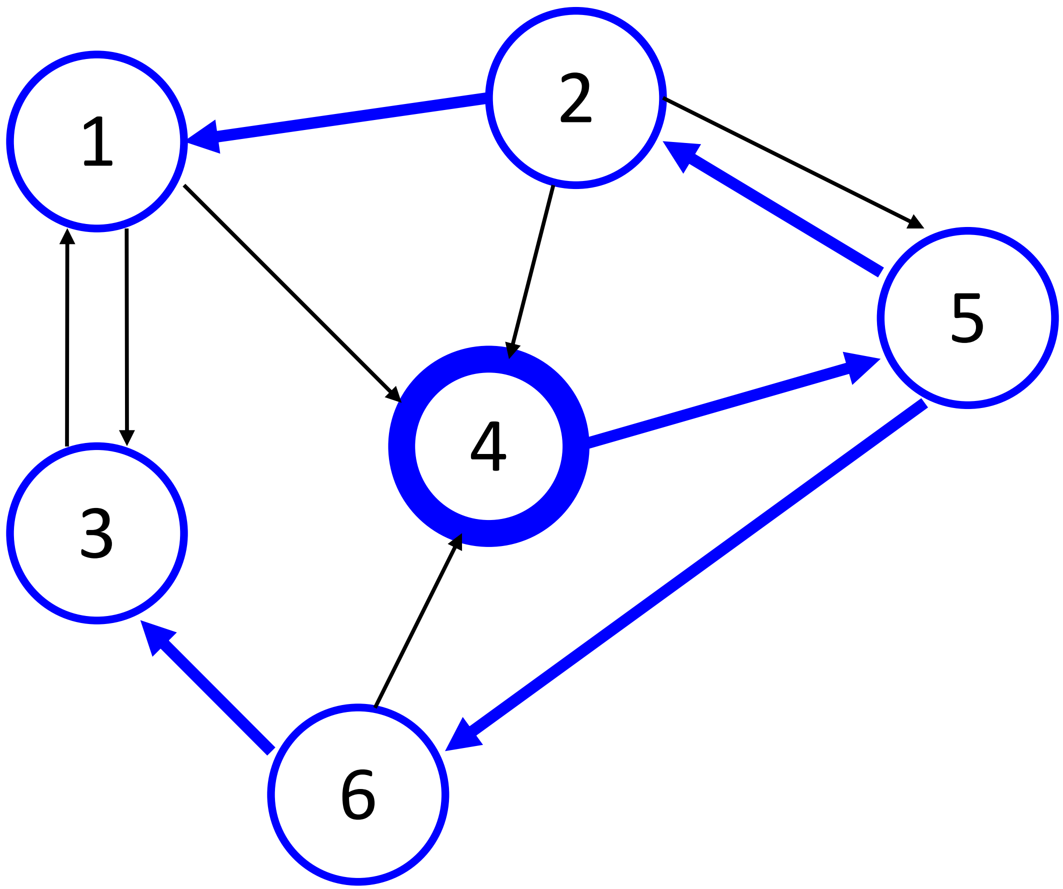
Round 3 (msgs)

Breadth-first search



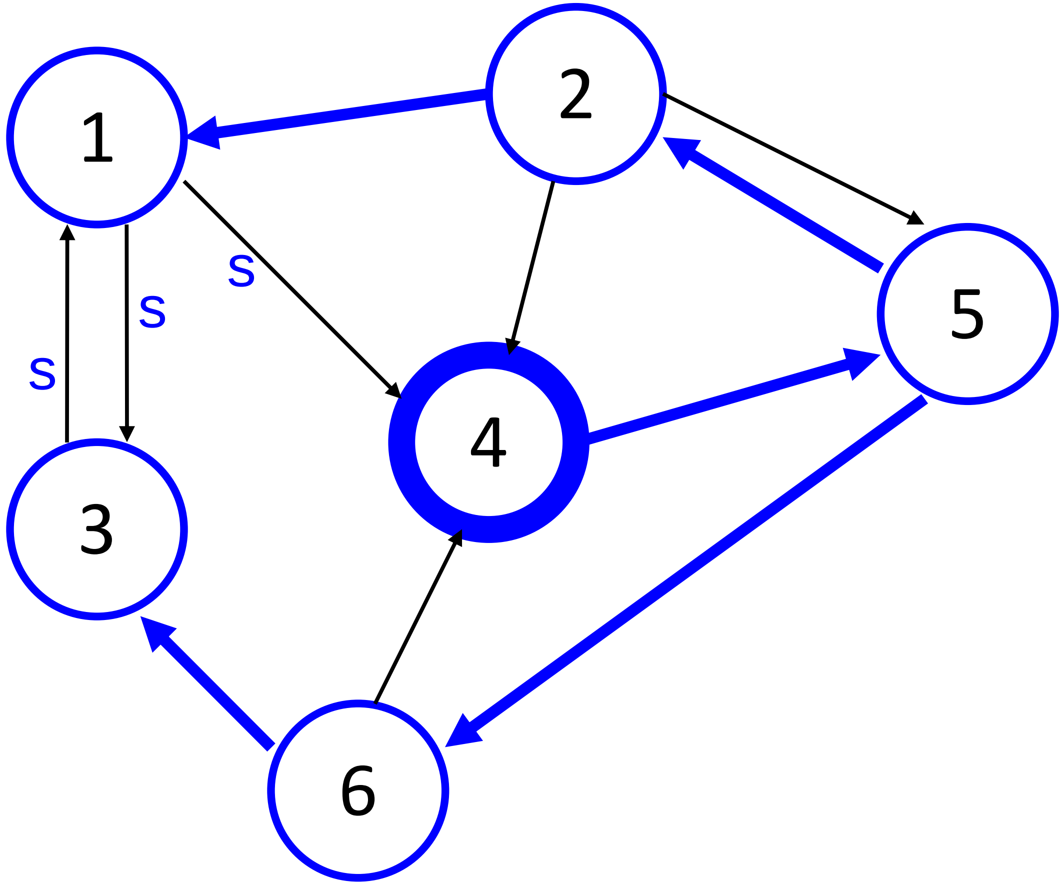
Round 3 (trans)

Breadth-first search



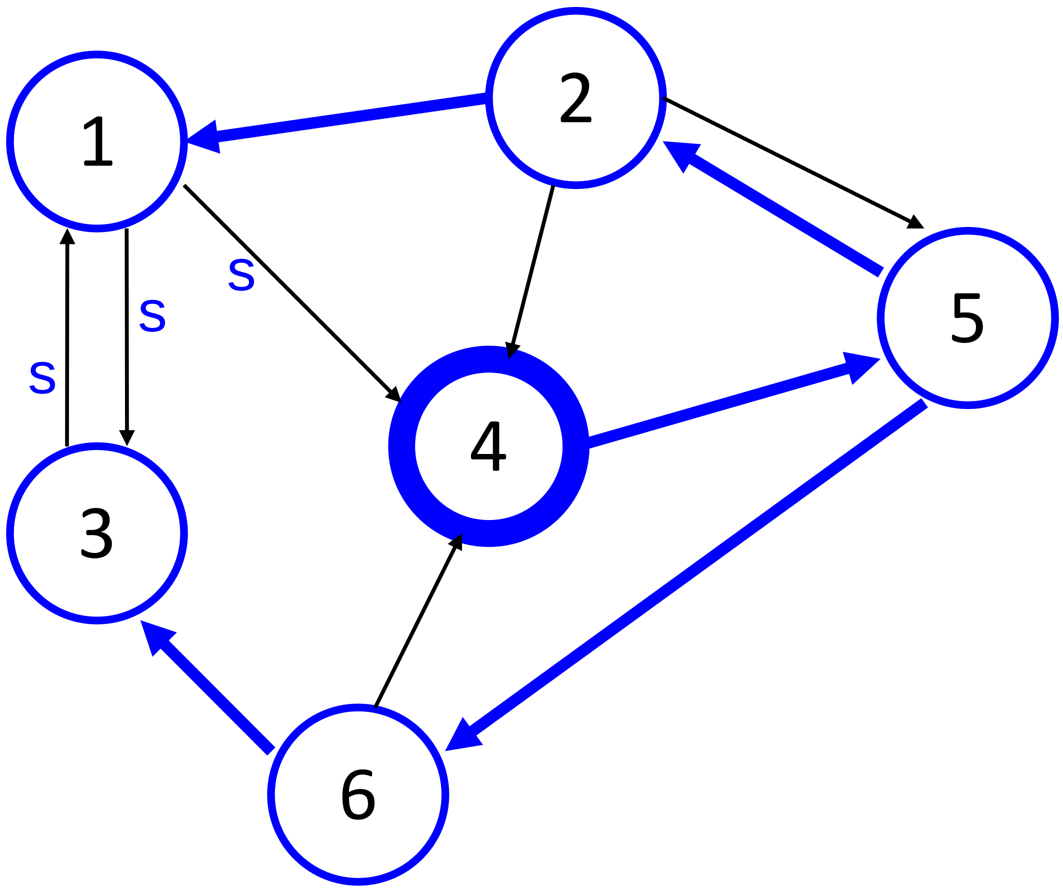
Round 4 (start)

Breadth-first search



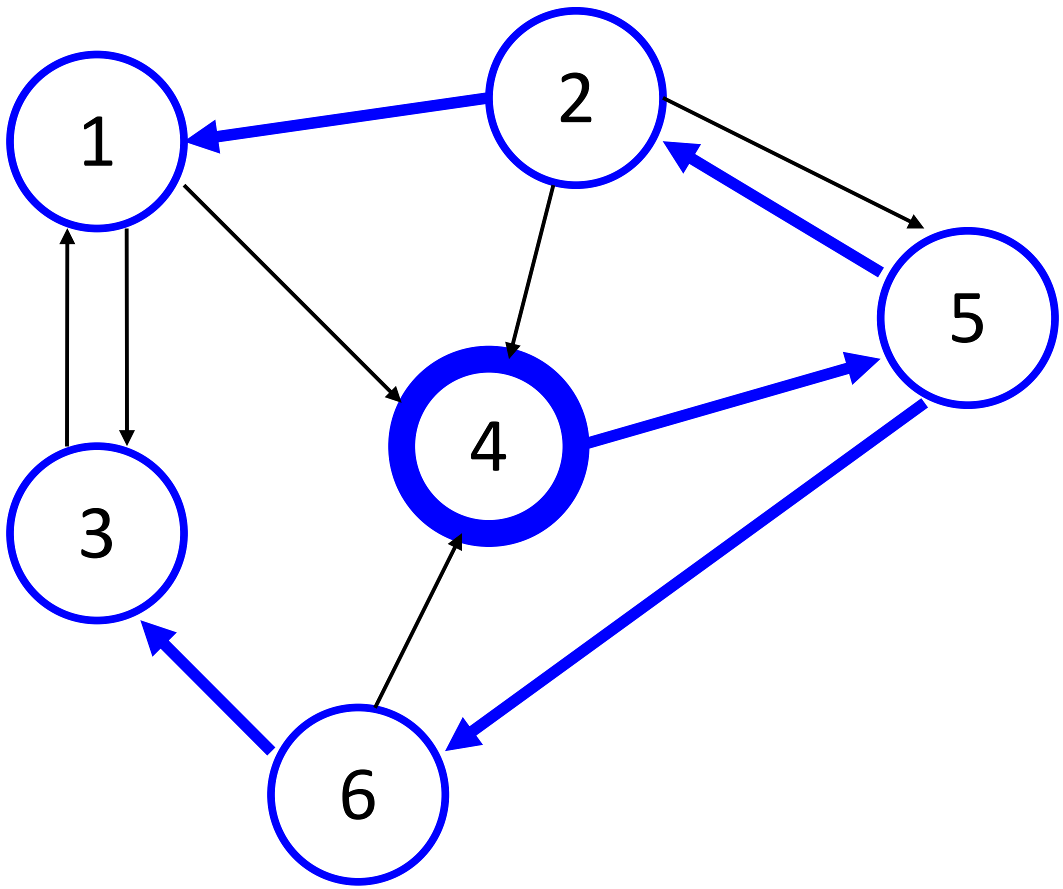
Round 4 (msgs)

Breadth-first search



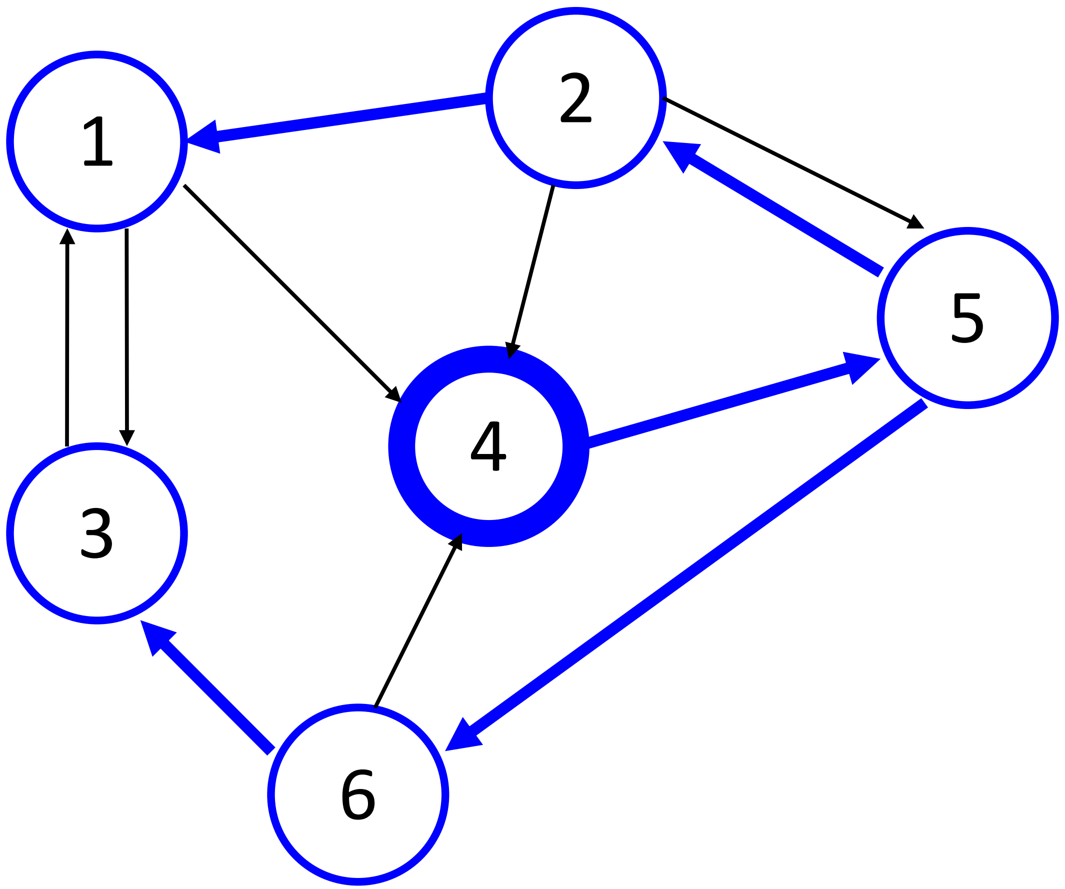
Round 4 (trans)

Breadth-first search



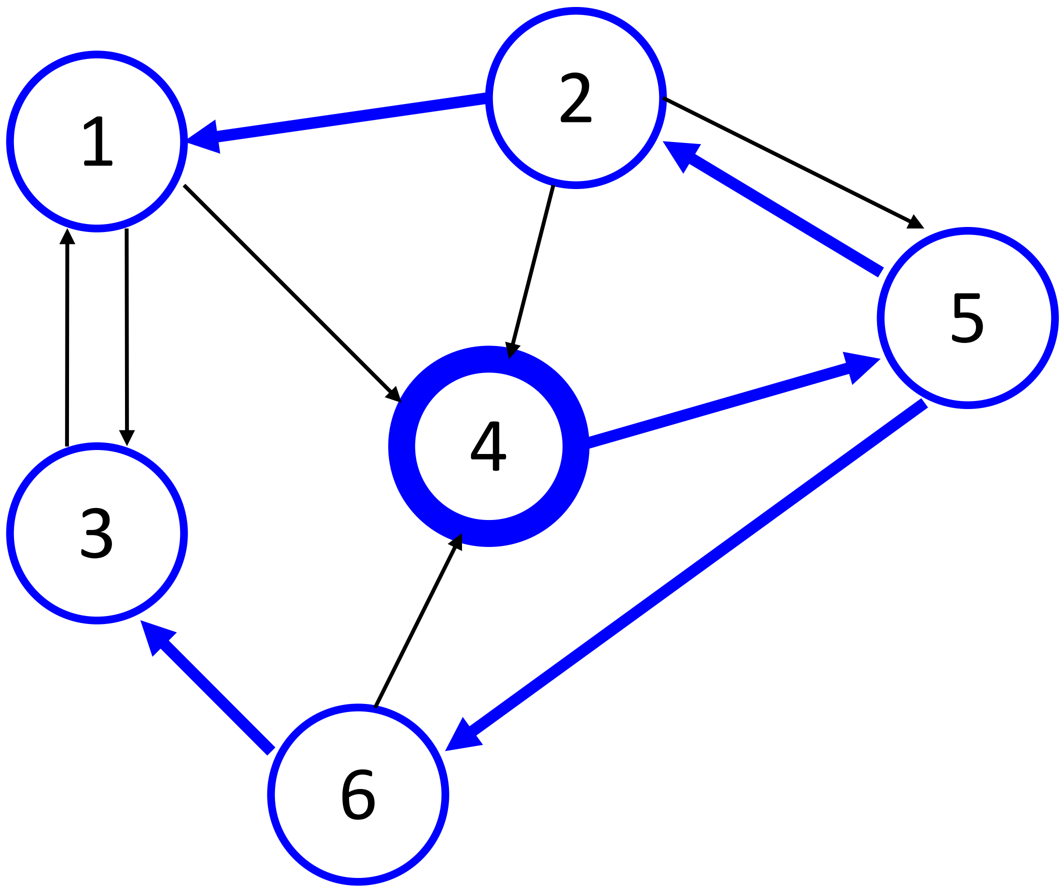
Round 5 (start)

Breadth-first search



Round 5 (msgs)

Breadth-first search



Round 5 (trans)

Breadth-first search algorithm

- Mark nodes as they get incorporated into the tree.
- Initially, only i_0 is marked.
- Round 1: i_0 sends $search$ message to out-nbrs.
- At every round: An unmarked node that receives a $search$ message:
  - Marks itself.
  - Designates one process from which it received $search$ as its parent.
  - Sends $search$ to out-nbrs at the next round.
- Yields a BFS tree because all the branches are created synchronously.
- Time complexity: $diam + 1$
- Message complexity: $|E|$

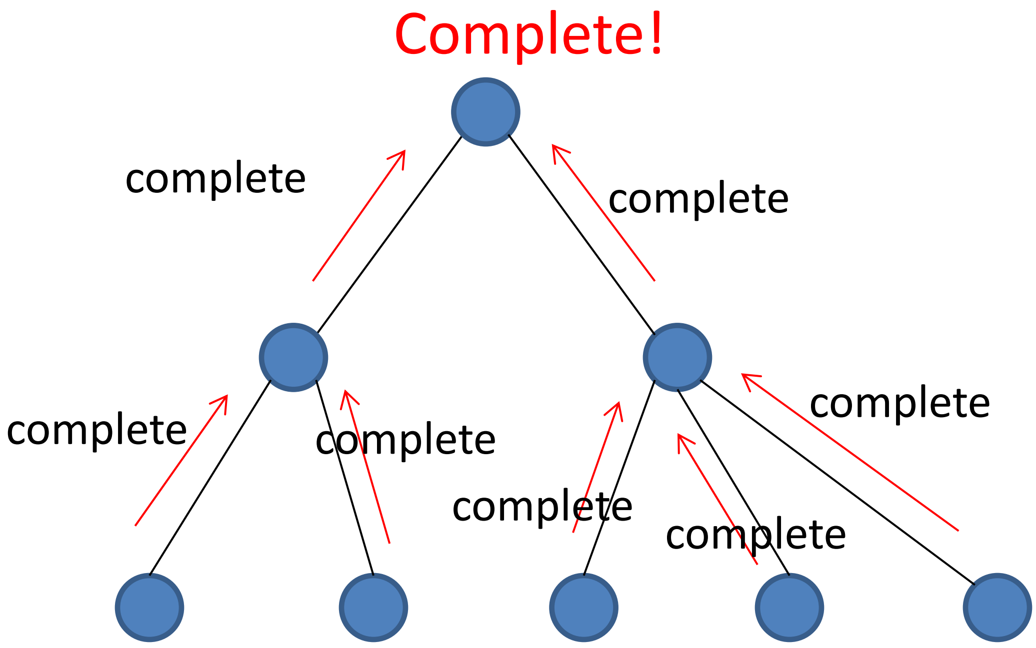
Adding child pointers to BFS

- Each $search$ message receives a response, $parent$ or $not - parent$.
- Easy with bidirectional communication.
- Harder with unidirectional communication:
- E.g. could use BFS again to search for parents.
  - High message bit complexity.

Termination for BFS

- Suppose i_0 wants to know when the BFS tree is completed.
- Assume each $search$ message receives a response, $parent$ or $not - parent$.
- After a node has received responses to all its outgoing $search$ messages, it knows who its children are, and knows they are all marked.
- The leaves of the tree discover who they are (they receive only $not - parent$ responses).
- Convergecast:
  - Starting from the leaves, the nodes fan in $complete$ messages to i_0, along the edges of the BFS tree.
  - A node can send a $complete$ message to its parent after:
    - It has received responses to all its outgoing $search$ messages (so it knows who its children are), and
    - It has received $complete$ messages from all its children.
- When i_0 has received $complete$ messages from all its children, it knows that the BFS tree is completed.

Convergecast



Applications of BFS

- Message broadcast:
  - Can broadcast a message while setting up the BFS tree ("piggyback" the message).
  - Or, first establish a BFS tree, with child pointers, then use it for broadcasting.
    - Can reuse the tree for many broadcasts
    - Each takes time only $O(diameter)$, messages $O(n)$.

- Now assume bidirectional edges (undirected graph).

Applications of BFS

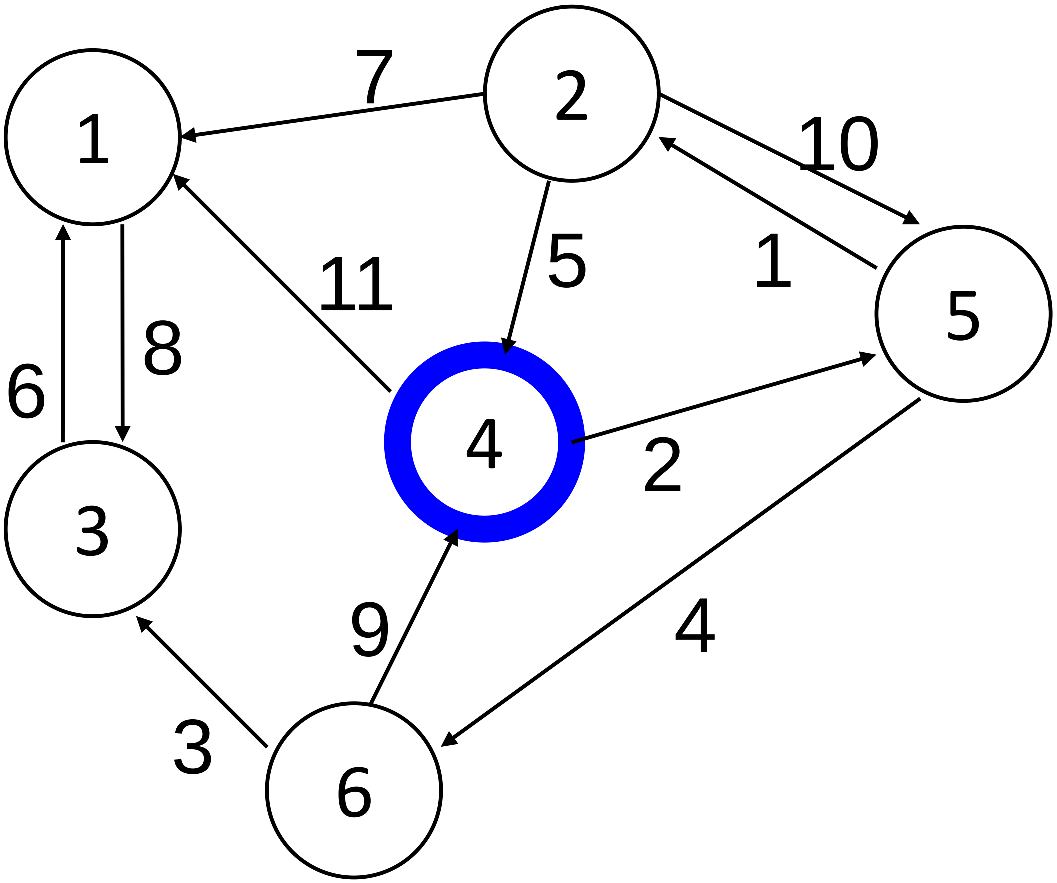
- Global computation:
  - Sum, max, or any kind of data aggregation: Convergecast on BFS tree.
  - Complexity: Time $O(diam)$; Messages $O(n)$
- Leader election (without knowing diameter)
  - Everyone starts BFS, determines max UID.
  - Complexity: Time $O(diam)$; Messages $O(n\,|E|)$ (actually, $O(\,diam\,|E|)$).
- Compute diameter:
  - All do BFS.
  - Convergecast to find height of each BFS tree.
  - Convergecast again to find max of all heights.

Shortest Paths

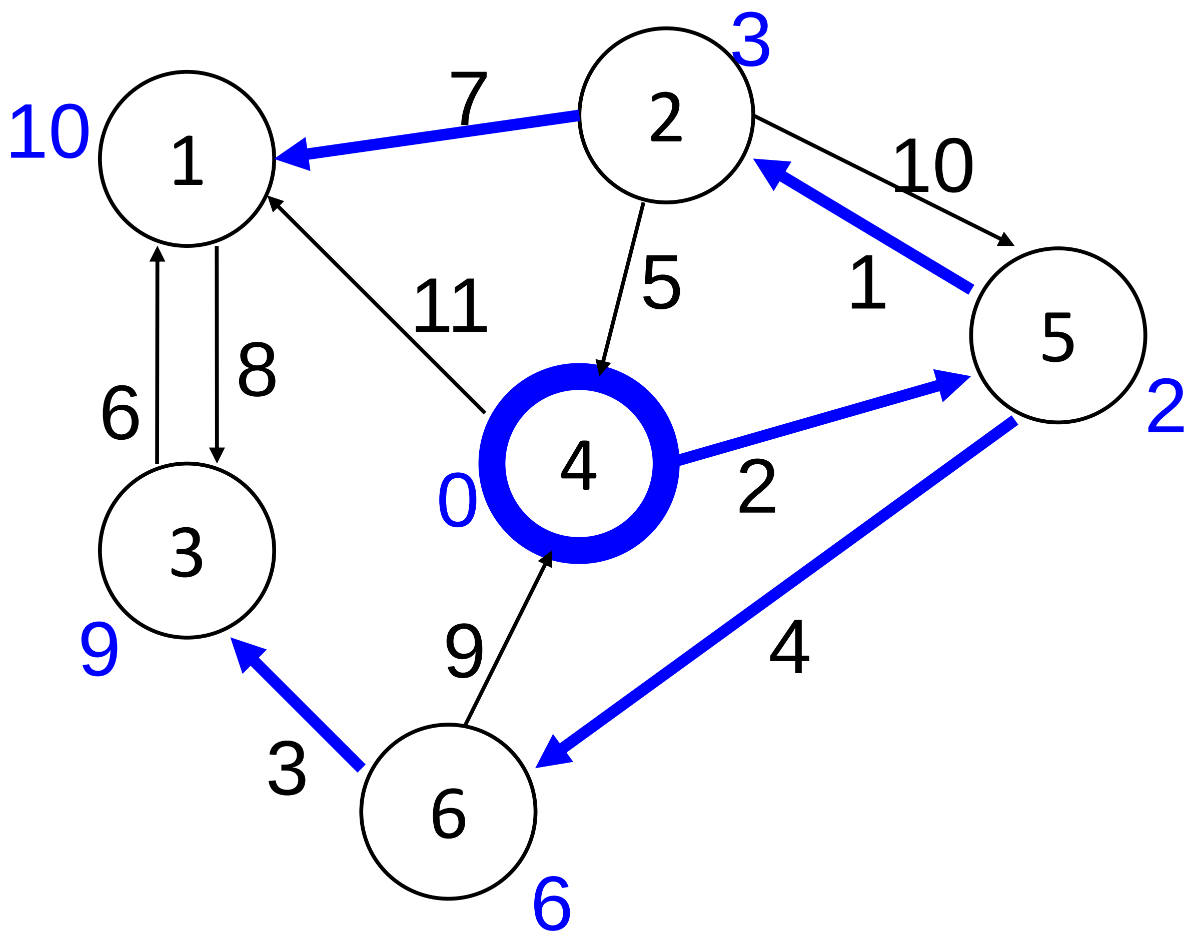
Shortest paths

- Motivation: Establish a structure for efficient communication.
  - Generalizes Breadth-First Search.
  - Now edges have associated costs (weights), w_{ij} for edge (i, j).
- Assume:
  - Strongly connected digraph, root i_0.
  - Weights (nonnegative reals) on edges.
    - Weights represent some type of communication cost, e.g. latency.
  - UIDs.
  - Nodes know weights of incident edges.
  - Nodes know n (use this just for termination).
- Required:
  - Shortest-paths tree, giving shortest path from i_0 to every other node.
  - Shortest path $=$ path with minimum total weight.
  - Each node should output:
    - Its weighted distance from i_0, and
    - Its parent on a shortest path from i_0.

Shortest paths



Shortest paths



Shortest paths algorithm

- Bellman-Ford (adapted from sequential Bellman-Ford algorithm)
- Each process maintains:
  - $dist$, shortest distance it knows about so far, from i_0
  - $parent$, its parent in some path with total weight $= dist$
  - $round$
- Initially:
  - i_0 has $dist = 0$, all others have $dist = \infty$.
  - Everyone's $parent = \perp$.
- At each round, each process:
  - Sends $dist$ to all $outnbrs$
  - Relaxation step:
    - Compute new $dist = \min(dist, min_j(dist_j + w_{ji}))$.
    - If $dist$ decreases then reset $parent$ to the corresponding $innbr$.
- Stop after $n - 1$ rounds.
- Then (claim) each process's $dist$ contains its distance from i_0, $parent$ contains the parent on a shortest path from i_0.

Next time

- More distributed algorithms for general synchronous networks:
  - Shortest paths, Bellman-Ford algorithm, continued
  - Minimum spanning tree, Gallager-Humblet-Spira algorithm
  - Maximal independent set, Luby's algorithm
- Readings:
  - Sections 4.3-4.5.
  - [Gallager, Humblet, Spira] (optional)
  - [Luby] (optional)
  - [Metivier, Robson,…] (optional)