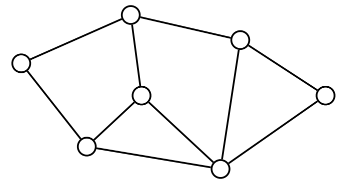
Distributed Algorithms

Congest Model

Congest Model

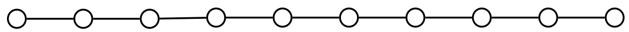
- Network with n computers (nodes) connected via communication channels (edges).



- Identifiers. Nodes has a unique identifier id: $V \to \{1,2,\ldots,n^c\}$ for some constant c.
- Messages. Nodes can exchange messages with neighbors.
- Communication rounds. All nodes perform the same algorithm synchronously in parallel:
  - Receive messages
  - Process
  - Send
- Message size. In each round over each edge send message of size O(logn) bits.

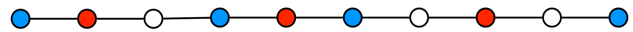
Path colouring

- Path coloring. No neighbouring nodes have the same color.

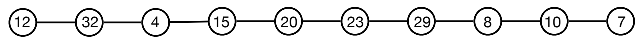


Path colouring

- Path coloring. No neighbouring nodes have the same color.



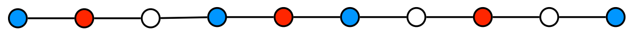
- 3-coloring. Color path with 3 colors $\{1,2,3\}$.
- Impossible without identifiers.



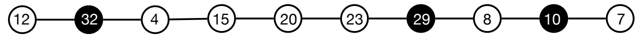
- P3C algorithm.
  - c = id.
  - Repeat forever:
    - Send message c to all neighbors.
    - Receive messages M from neighbors.
    - If $c \neq \{1,2,3\}$ and $c >$ all messages received in this round:
      - $c \leftarrow \min(\{1,2,3\} \setminus M\})$

Path colouring

- Path coloring. No neighbouring nodes have the same color.



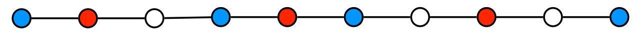
- 3-coloring. Color path with 3 colors $\{1,2,3\}$.
- Impossible without identifiers.



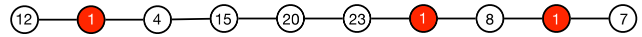
- P3C algorithm.
  - c = id.
  - Repeat forever:
    - Send message c to all neighbors.
    - Receive messages M from neighbors.
    - If $c \neq \{1,2,3\}$ and $c >$ all messages received in this round:
      - $c \leftarrow \min(\{1,2,3\} \setminus M)$

Path colouring

- Path coloring. No neighbouring nodes have the same color.



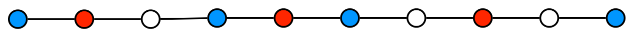
- 3-coloring. Color path with 3 colors $\{1,2,3\}$.
- Impossible without identifiers.



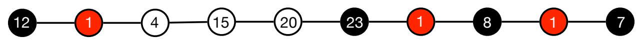
- P3C algorithm.
  - c = id.
  - Repeat forever:
    - Send message c to all neighbors.
    - Receive messages M from neighbors.
    - If $c \neq \{1,2,3\}$ and $c >$ all messages received in this round:
      - $c \leftarrow \min(\{1,2,3\} \setminus M)$

Path colouring

- Path coloring. No neighbouring nodes have the same color.



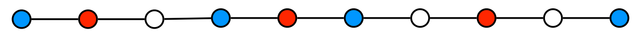
- 3-coloring. Color path with 3 colors $\{1,2,3\}$.
- Impossible without identifiers.



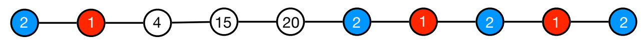
- P3C algorithm.
  - c = id.
  - Repeat forever:
    - Send message c to all neighbors.
    - Receive messages M from neighbors.
    - If $c \neq \{1,2,3\}$ and $c >$ all messages received in this round:
      - $c \leftarrow \min(\{1,2,3\} \setminus M)$

Path colouring

- Path coloring. No neighbouring nodes have the same color.



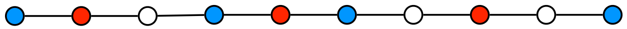
- 3-coloring. Color path with 3 colors $\{1,2,3\}$.
- Impossible without identifiers.



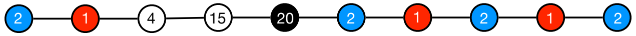
- P3C algorithm.
  - c = id.
  - Repeat forever:
    - Send message c to all neighbors.
    - Receive messages M from neighbors.
    - If $c \neq \{1,2,3\}$ and $c >$ all messages received in this round:
      - $c \leftarrow \min(\{1,2,3\} \setminus M)$

Path colouring

- Path coloring. No neighbouring nodes have the same color.



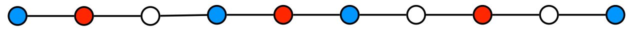
- 3-coloring. Color path with 3 colors $\{1,2,3\}$.
- Impossible without identifiers.



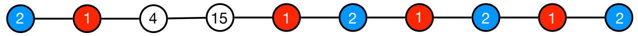
- P3C algorithm.
  - c = id.
  - Repeat forever:
    - Send message c to all neighbors.
    - Receive messages M from neighbors.
    - If $c \neq \{1,2,3\}$ and $c >$ all messages received in this round:
      - $c \leftarrow \min(\{1,2,3\} \backslash M\})$

Path colouring

- Path coloring. No neighbouring nodes have the same color.



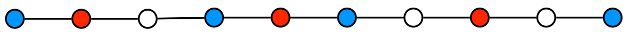
- 3-coloring. Color path with 3 colors $\{1,2,3\}$.
- Impossible without identifiers.



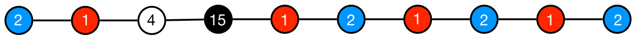
- P3C algorithm.
  - c = id.
  - Repeat forever:
    - Send message c to all neighbors.
    - Receive messages M from neighbors.
    - If $c \neq \{1,2,3\}$ and $c >$ all messages received in this round:
      - $c \leftarrow \min(\{1,2,3\} \backslash M\})$

Path colouring

- Path coloring. No neighbouring nodes have the same color.



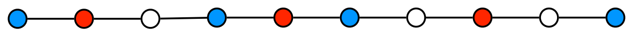
- 3-coloring. Color path with 3 colors $\{1,2,3\}$.
- Impossible without identifiers.



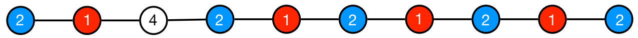
- P3C algorithm.
  - c = id.
  - Repeat forever:
    - Send message c to all neighbors.
    - Receive messages M from neighbors.
    - If $c \neq \{1,2,3\}$ and $c >$ all messages received in this round:
      - $c \leftarrow \min(\{1,2,3\} \backslash M\})$

Path colouring

- Path coloring. No neighbouring nodes have the same color.



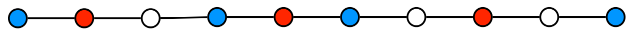
- 3-coloring. Color path with 3 colors $\{1,2,3\}$.
- Impossible without identifiers.



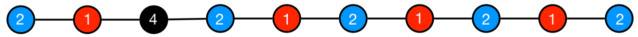
- P3C algorithm.
  - c = id.
  - Repeat forever:
    - Send message c to all neighbors.
    - Receive messages M from neighbors.
    - If $c \neq \{1,2,3\}$ and $c >$ all messages received in this round:
      - $c \leftarrow \min(\{1,2,3\} \backslash M\})$

Path colouring

- Path coloring. No neighbouring nodes have the same color.



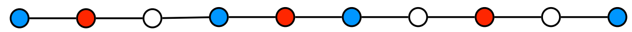
- 3-coloring. Color path with 3 colors $\{1,2,3\}$.
- Impossible without identifiers.



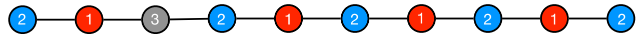
- P3C algorithm.
  - c = id.
  - Repeat forever:
    - Send message c to all neighbors.
    - Receive messages M from neighbors.
    - If $c \neq \{1,2,3\}$ and $c >$ all messages received in this round:
      - $c \leftarrow \min(\{1,2,3\} \backslash M)$

Path colouring

- Path coloring. No neighbouring nodes have the same color.



- 3-coloring. Color path with 3 colors $\{1,2,3\}$.
- Impossible without identifiers.



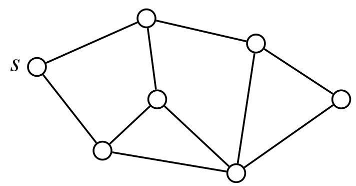
- P3C algorithm.
  - c = id.
  - Repeat forever:
    - Send message c to all neighbors.
    - Receive messages M from neighbors.
    - If $c \neq \{1,2,3\}$ and $c >$ all messages received in this round:
      - $c \leftarrow \min(\{1,2,3\} \backslash M)$

All-Pairs Shortest Paths

- All-Pairs Shortest Paths. The local output of a node is the identities of all other nodes and the distance to them.
- Algorithm.
  - BFS tree from a specific node (leader)
  - Use BFS tree without a leader
  - Pipeline BFS computations.

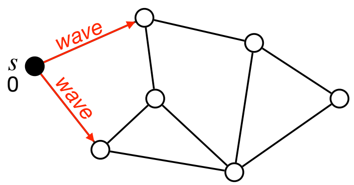
BFS

- BFS. Local output from each node is the distance to the leader s.
- Algorithm.
  - Round 0: leader sends "*wave*" to all neighbors, switch to state 0 and stops.
  - Round i: Each node that is not stopped
    - if it receives "*wave*" from some port(s)
      - switch to state i.
      - send message "*wave*" to all neighbors and stop.



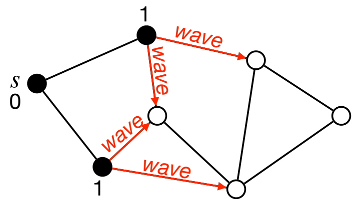
BFS

- BFS. Local output from each node is the distance to the leader s.
- Algorithm.
  - Round 0: leader sends "*wave*" to all neighbors, switch to state 0 and stops.
  - Round i: Each node that is not stopped
    - if it receives *"wave"* from some port(s)
      - switch to state i.
      - send message *"wave"* to all neighbors and stop.



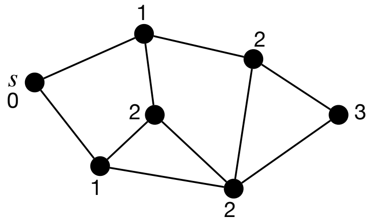
BFS

- BFS. Local output from each node is the distance to the leader s.
- Algorithm.
  - Round 0: leader sends "*wave*" to all neighbors, switch to state 0 and stops.
  - Round i: Each node that is not stopped
    - if it receives *"wave"* from some port(s)
      - switch to state i.
      - send message *"wave"* to all neighbors and stop.



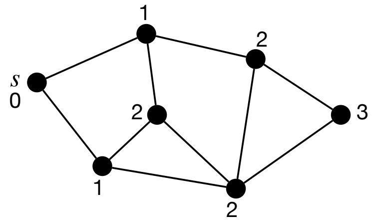
BFS

- BFS. Local output from each node is the distance to the leader s.
- Algorithm.
  - Round 0: leader sends "*wave*" to all neighbors, switch to state 0 and stops.
  - Round i: Each node that is not stopped
    - if it receives *"wave"* from some port(s)
      - switch to state i.
      - send message *"wave"* to all neighbors and stop.



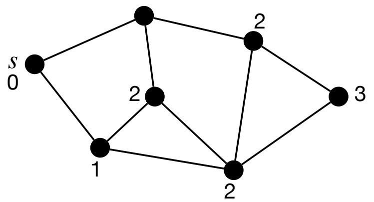
BFS

- BFS. Local output from each node is the distance to the leader s.
- Algorithm.
  - Round 0: leader sends "*wave*" to all neighbors, switch to state 0 and stops.
  - Round i: Each node that is not stopped
    - if it receives *"wave"* from some port(s)
      - switch to state i.
      - send message *"wave"* to all neighbors and stop.
  - Additional information: parent and children in BFS tree?



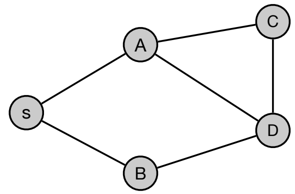
BFS

- BFS. Local output from each node is the distance to the leader s.
- Algorithm.
  - Round 0: leader sends *"wave"* to all neighbors, switch to state 0 and stops.
  - Round i: Each node that is not stopped
    - if it receives *"wave"* from some port(s)
      - switch to state i.
      - send message *"wave"* to all neighbors and stop.
  - Additional information: parent and children in BFS tree.
    - When receiving *"wave"* request, choose one to accept and send accept back.



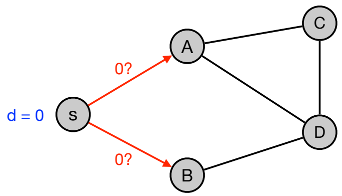
Wave

	Computation	Send
Round 1		s: 0? -> A, B



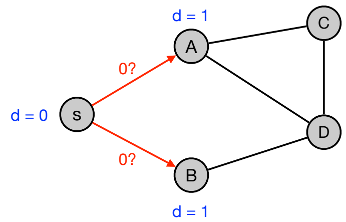
Wave

	Computation	Send
Round 1		s: 0? -> A, B

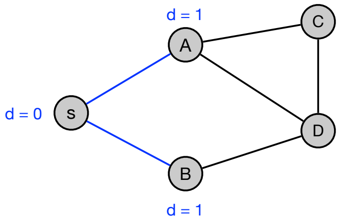


Wave

	Computation	Send
Round 1		s: 0? -> A, B
Round 2	d(A) = 1, p(A) = s	A: accept -> s, A: 1? -> C, A: 1? -> D
	d(B) = 1, p(B) = s	B: accept -> s B: 1? -> D

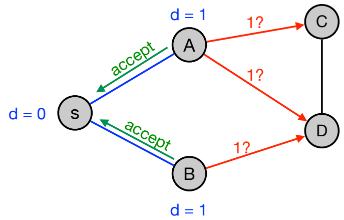


Wave



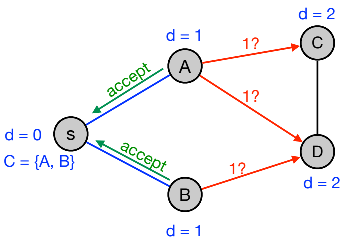
	Computation	Send
Round 1		s: 0? -> A, B
Round 2	d(A) = 1, p(A) = s	A: accept -> s, A: 1? -> C, A: 1? -> D
	d(B) = 1, p(B) = s	B: accept -> s B: 1? -> D

Wave



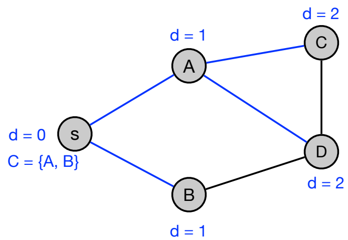
	Computation	Send
Round 1		s: 0? -> A, B
Round 2	d(A) = 1, p(A) = s	A: accept -> s, A: 1? -> C, A: 1? -> D
	d(B) = 1, p(B) = s	B: accept -> s B: 1? -> D

Wave



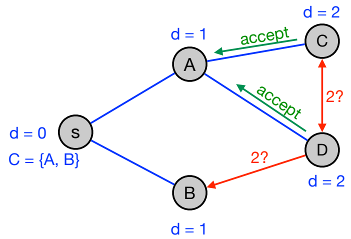
	Computation	Send
Round 1		s: 0? -> A, B
Round 2	d(A) = 1, p(A) = s	A: accept -> s, A: 1? -> C, A: 1? -> D
	d(B) = 1, p(B) = s	B: accept -> s B: 1? -> D
Round 3	C(s) = {A, B}	
	d(C) = 2, p(C) = A	C: accept -> A C: 2? -> D
	d(D) = 2, p(D) = A	D: accept -> A D: 2? -> C D: 2? -> B

Wave



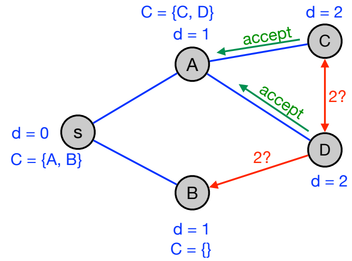
	Computation	Send
Round 1		s: 0? -> A, B
Round 2	d(A) = 1, p(A) = s	A: accept -> s, A: 1? -> C, A: 1? -> D
	d(B) = 1, p(B) = s	B: accept -> s B: 1? -> D
Round 3	C(s) = {A, B}	
	d(C) = 2, p(C) = A	C: accept -> A C: 2? -> D
	d(D) = 2, p(D) = A	D: accept -> A D: 2? -> C D: 2? -> B

Wave



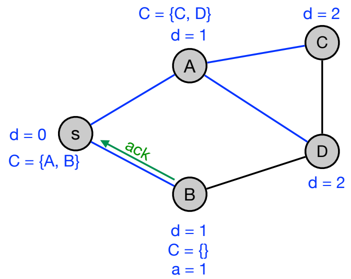
	Computation	Send
Round 1		s: 0? -> A, B
Round 2	d(A) = 1, p(A) = s	A: accept -> s, A: 1? -> C, A: 1? -> D
	d(B) = 1, p(B) = s	B: accept -> s B: 1? -> D
Round 3	C(s) = {A, B}	
	d(C) = 2, p(C) = A	C: accept -> A C: 2? -> D
	d(D) = 2, p(D) = A	D: accept -> A D: 2? -> C D: 2? -> B

Wave



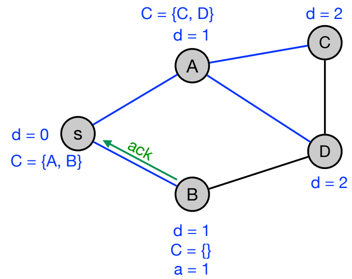
	Computation	Send
Round 1		s: 0? -> A, B
Round 2	d(A) = 1, p(A) = s	A: accept -> s, A: 1? -> C, A: 1? -> D
	d(B) = 1, p(B) = s	B: accept -> s B: 1? -> D
Round 3	C(s) = {A, B}	
	d(C) = 2, p(C) = A	C: accept -> A C: 2? -> D
	d(D) = 2, p(D) = A	D: accept -> A D: 2? -> C D: 2? -> B
Round 4	C(A) = {C,D}	
	C(B) = {}, a(B) = 1	B: ack -> s

Wave



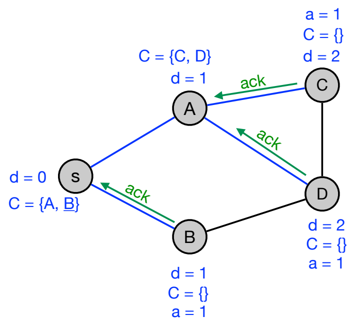
	Computation	Send
Round 1		s: 0? -> A, B
Round 2	d(A) = 1, p(A) = s	A: accept -> s, A: 1? -> C, A: 1? -> D
	d(B) = 1, p(B) = s	B: accept -> s B: 1? -> D
Round 3	C(s) = {A, B}	
	d(C) = 2, p(C) = A	C: accept -> A C: 2? -> D
	d(D) = 2, p(D) = A	D: accept -> A D: 2? -> C D: 2? -> B
Round 4	C(A) = {C,D}	
	C(B) = {}, a(B) = 1	B: ack -> s

Wave



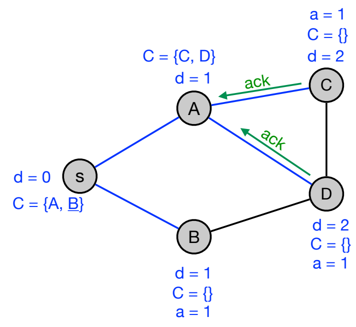
	Computation	Send
Round 1		s: 0? -> A, B
Round 2	d(A) = 1, p(A) = s	A: accept -> s, A: 1? -> C, A: 1? -> D
	d(B) = 1, p(B) = s	B: accept -> s B: 1? -> D
Round 3	C(s) = {A, B}	
	d(C) = 2, p(C) = A	C: accept -> A C: 2? -> D
	d(D) = 2, p(D) = A	D: accept -> A D: 2? -> C D: 2? -> B
Round 4	C(A) = {C,D}	
	C(B) = {}, a(B) = 1	B: ack -> s
Round 5	C(C) = {}, a(C) = 1	C: ack -> A
	C(D) = {}, a(D) = 1	D: ack -> A

Wave



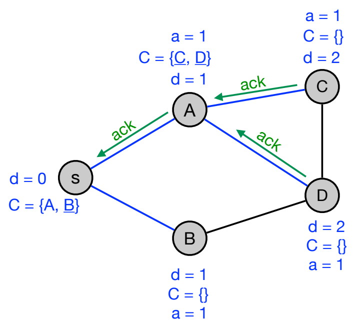
	Computation	Send
Round 1		s: 0? -> A, B
Round 2	d(A) = 1, p(A) = s	A: accept -> s, A: 1? -> C, A: 1? -> D
	d(B) = 1, p(B) = s	B: accept -> s B: 1? -> D
Round 3	C(s) = {A, B}	
	d(C) = 2, p(C) = A	C: accept -> A C: 2? -> D
	d(D) = 2, p(D) = A	D: accept -> A D: 2? -> C D: 2? -> B
Round 4	C(A) = {C,D}	
	C(B) = {}, a(B) = 1	B: ack -> s
Round 5	C(C) = {}, a(C) = 1	C: ack -> A
	C(D) = {}, a(D) = 1	D: ack -> A

Wave



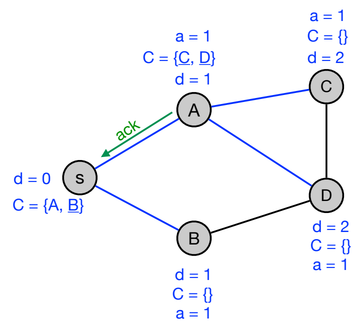
	Computation	Send
Round 1		s: 0? -> A, B
Round 2	d(A) = 1, p(A) = s	A: accept -> s, A: 1? -> C, A: 1? -> D
	d(B) = 1, p(B) = s	B: accept -> s B: 1? -> D
Round 3	C(s) = {A, B}	
	d(C) = 2, p(C) = A	C: accept -> A C: 2? -> D
	d(D) = 2, p(D) = A	D: accept -> A D: 2? -> C D: 2? -> B
Round 4	C(A) = {C,D}	
	C(B) = {}, a(B) = 1	B: ack -> s
Round 5	C(C) = {}, a(C) = 1	C: ack -> A
	C(D) = {}, a(D) = 1	D: ack -> A
Round 7	a(A) = 1	A: ack -> s

Wave



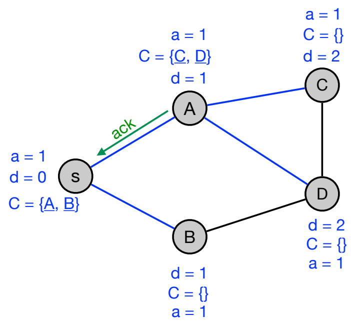
	Computation	Send
Round 1		s: 0? -> A, B
Round 2	d(A) = 1, p(A) = s	A: accept -> s, A: 1? -> C, A: 1? -> D
	d(B) = 1, p(B) = s	B: accept -> s B: 1? -> D
Round 3	C(s) = {A, B}	
	d(C) = 2, p(C) = A	C: accept -> A C: 2? -> D
	d(D) = 2, p(D) = A	D: accept -> A D: 2? -> C D: 2? -> B
Round 4	C(A) = {C,D}	
	C(B) = {}, a(B) = 1	B: ack -> s
Round 5	C(C) = {}, a(C) = 1	C: ack -> A
	C(D) = {}, a(D) = 1	D: ack -> A
Round 7	a(A) = 1	A: ack -> s

Wave



	Computation	Send
Round 1		s: 0? -> A, B
Round 2	d(A) = 1, p(A) = s	A: accept -> s, A: 1? -> C, A: 1? -> D
	d(B) = 1, p(B) = s	B: accept -> s B: 1? -> D
Round 3	C(s) = {A, B}	
	d(C) = 2, p(C) = A	C: accept -> A C: 2? -> D
	d(D) = 2, p(D) = A	D: accept -> A D: 2? -> C D: 2? -> B
Round 4	C(A) = {C,D}	
	C(B) = {}, a(B) = 1	B: ack -> s
Round 5	C(C) = {}, a(C) = 1	C: ack -> A
	C(D) = {}, a(D) = 1	D: ack -> A
Round 7	a(A) = 1	A: ack -> s
Round 8	a(s) = 1	

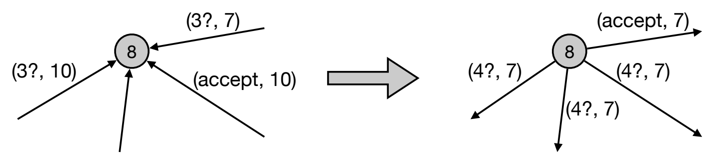
Wave



	Computation	Send
Round 1		s: 0? -> A, B
Round 2	d(A) = 1, p(A) = s	A: accept -> s, A: 1? -> C, A: 1? -> D
	d(B) = 1, p(B) = s	B: accept -> s B: 1? -> D
Round 3	C(s) = {A, B}	
	d(C) = 2, p(C) = A	C: accept -> A C: 2? -> D
	d(D) = 2, p(D) = A	D: accept -> A D: 2? -> C D: 2? -> B
Round 4	C(A) = {C,D}	
	C(B) = {}, a(B) = 1	B: ack -> s
Round 5	C(C) = {}, a(C) = 1	C: ack -> A
	C(D) = {}, a(D) = 1	D: ack -> A
Round 7	a(A) = 1	A: ack -> s
Round 8	a(s) = 1	

Electing a Leader

- Use BFS!?
- Algorithm.
  - Run Wave(v) from every node.
  - Augment messages with identity of root node.
  - A node only sends messages related smallest id seen so far.
  - When a node has received acknowledgment from all its children it sends a message (using the BFS tree) to all other nodes that it is the leader.



Electing a Leader

Electing a Leader

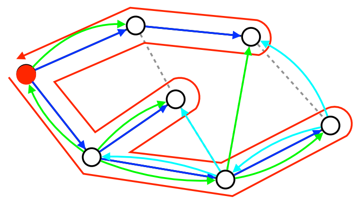
- Correctness.
  - Exactly one node will receive acknowledgment from all its children in its BFS tree (namely s = min V).
- Number of rounds.
  - O(diam(G))
- CONGEST model.
  - Every node sends only messages related to one BFS process in each round.

APSP

- Local output. Every node knows the identity of all other nodes and the distance to them.
- Run Wave(v) from all nodes:
  - In parallel? Messages too large!
  - Sequentially? O(n diam(G)) rounds
- Token Walk.
  - Move a token in the BFS tree T_s of the leader.
  - Spend 2 rounds in each node before continuing.
  - First time we meet a node v in the walk start Wave(v).

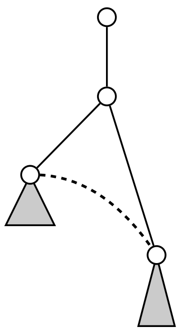
APSP

- Token Walk.
  - Move a token in the BFS tree T_s of the leader.
  - Spend 2 rounds in each node before continuing.
  - First time we meet a node v in the walk start Wave(v).



APSP

- Local output. Every node nodes the identity of all other nodes and the distance to them.
- Token Walk.
  - Move a token in the BFS tree T_s of the leader.
  - Spend 2 rounds in each node before continuing.
  - First time we meet a node v in the walk start Wave(v).
- Claim. Two waves Wave(u) and Wave(v) never collides.
  - Assume Wave(u) starts before Wave(v).
  - $d = d_G(u, v)$
  - T_s is a subgraph of G.
  - It takes at least $2d$ rounds to move the token from u to v.
  - It takes d rounds for Wave(u) to reach v.
  - When Wave(v) is started Wave(u) has already passed.
  - Wave(v) never catches up with Wave(u) (move at same speed).



APSP

- Local output. Every node nodes the identity of all other nodes and the distance to them.
- Token Walk.
  - Move a token in the BFS tree T_s of the leader.
  - Spend 2 rounds in each node before continuing.
  - First time we meet a node v in the walk start Wave(v).
- Rounds.
  - After O(n) rounds all Waves have been started.
  - Number of rounds: O(n + diam(G)) = O(n).