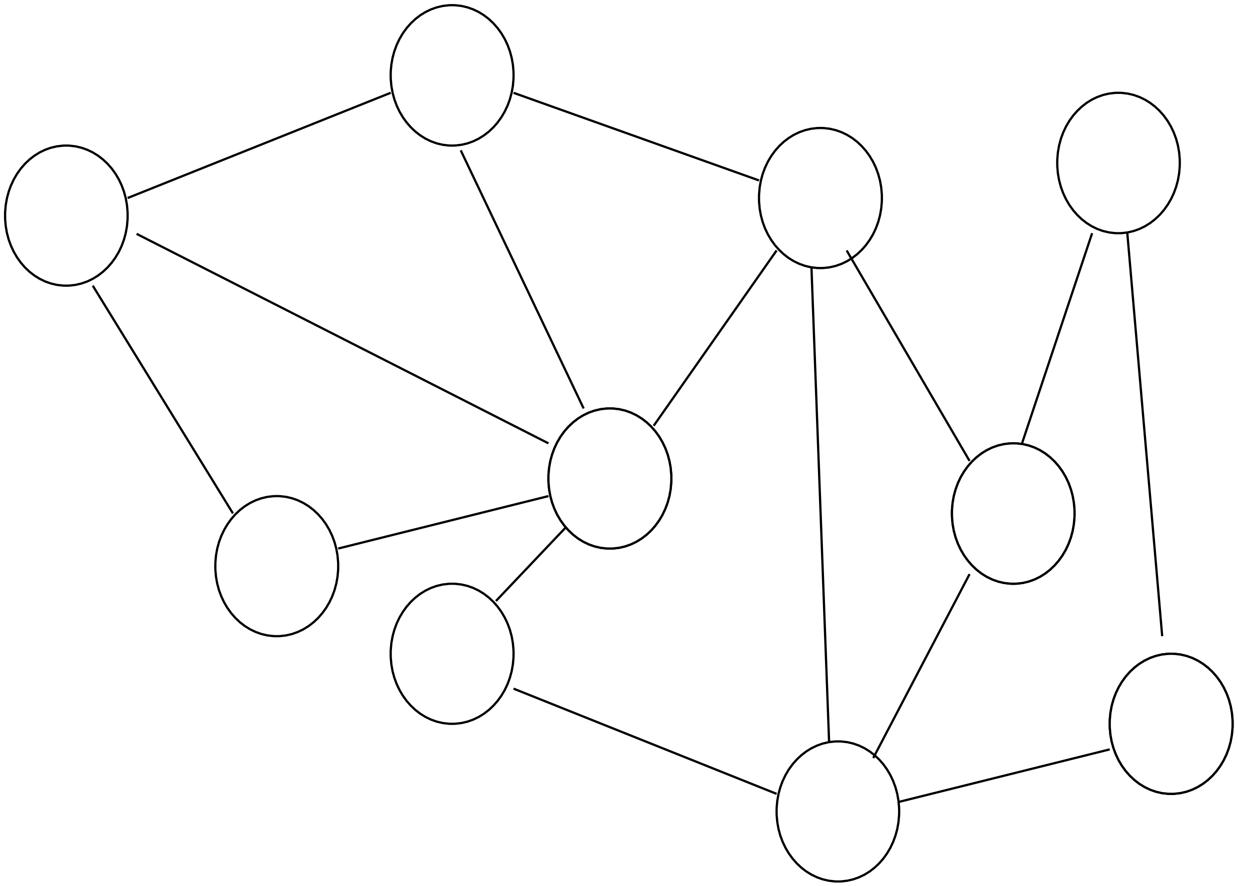
GIAN Course on Distributed Network Algorithms

Spanning Tree Constructions

Spanning Trees

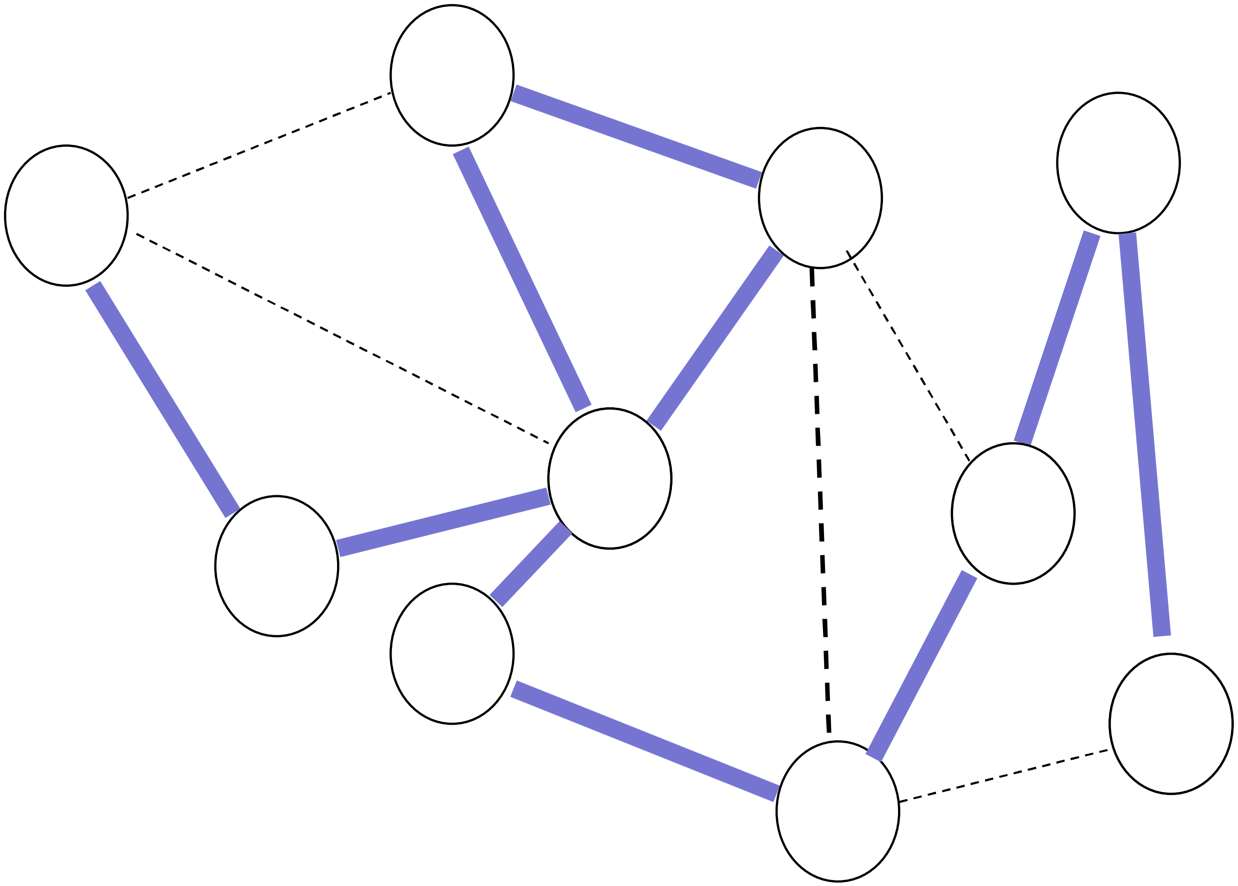
Attactive „infrastructure“: sparse subgraph („loop-free backbone“) connecting all nodes. E.g., cheap flooding.



Spanning Tree

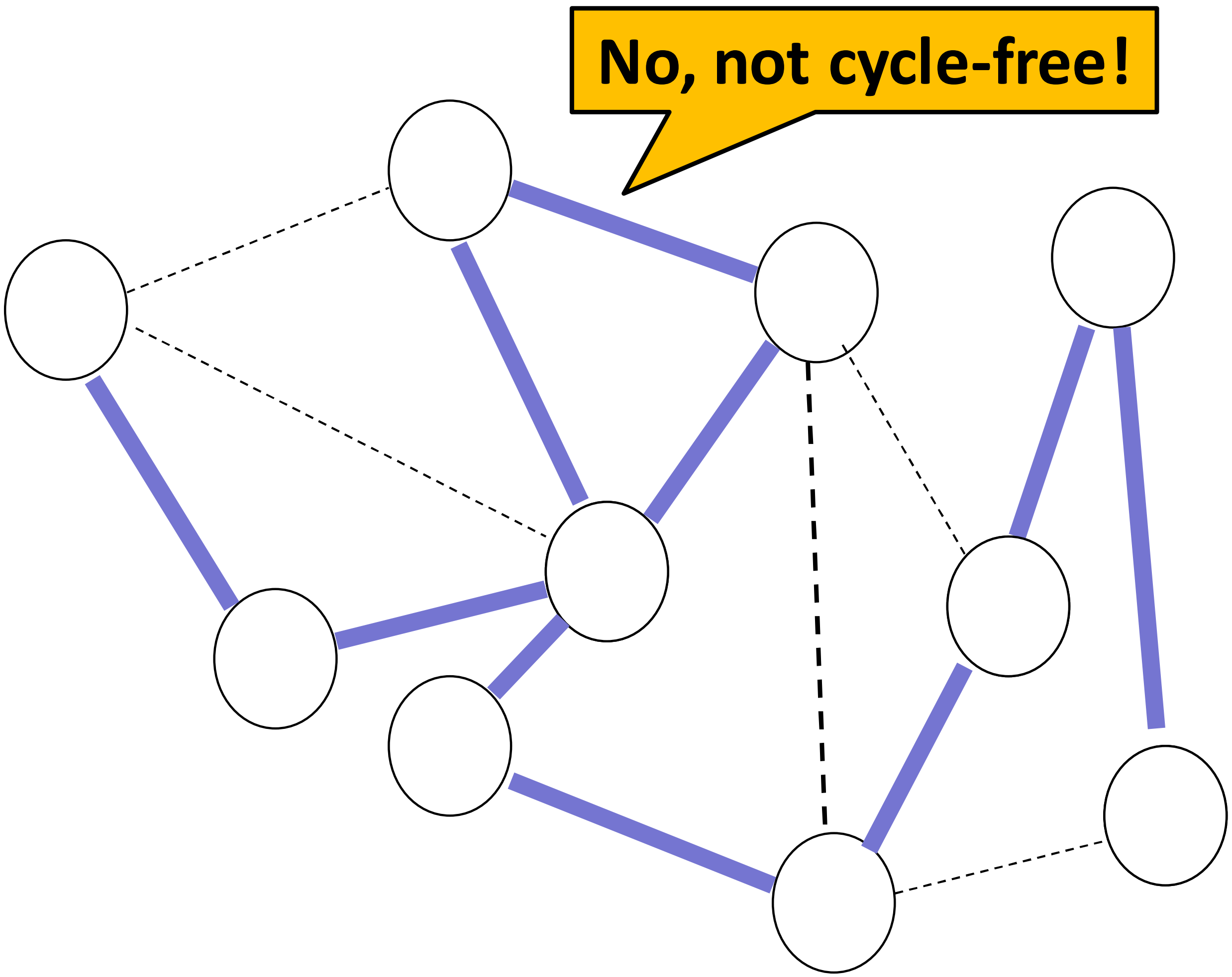
Cycle-free subgraph spanning all nodes.

Spanning Trees

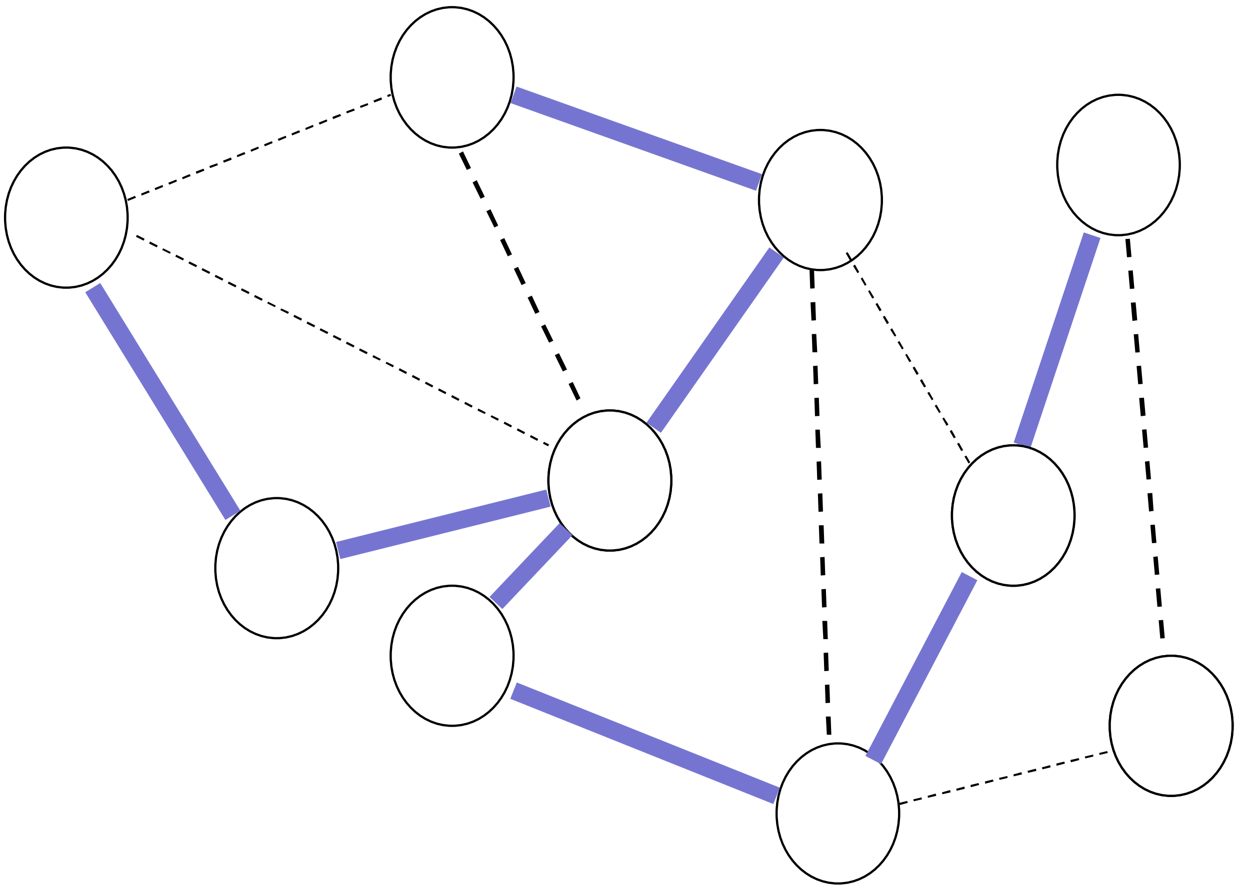


Is this a spanning tree?

Spanning Trees

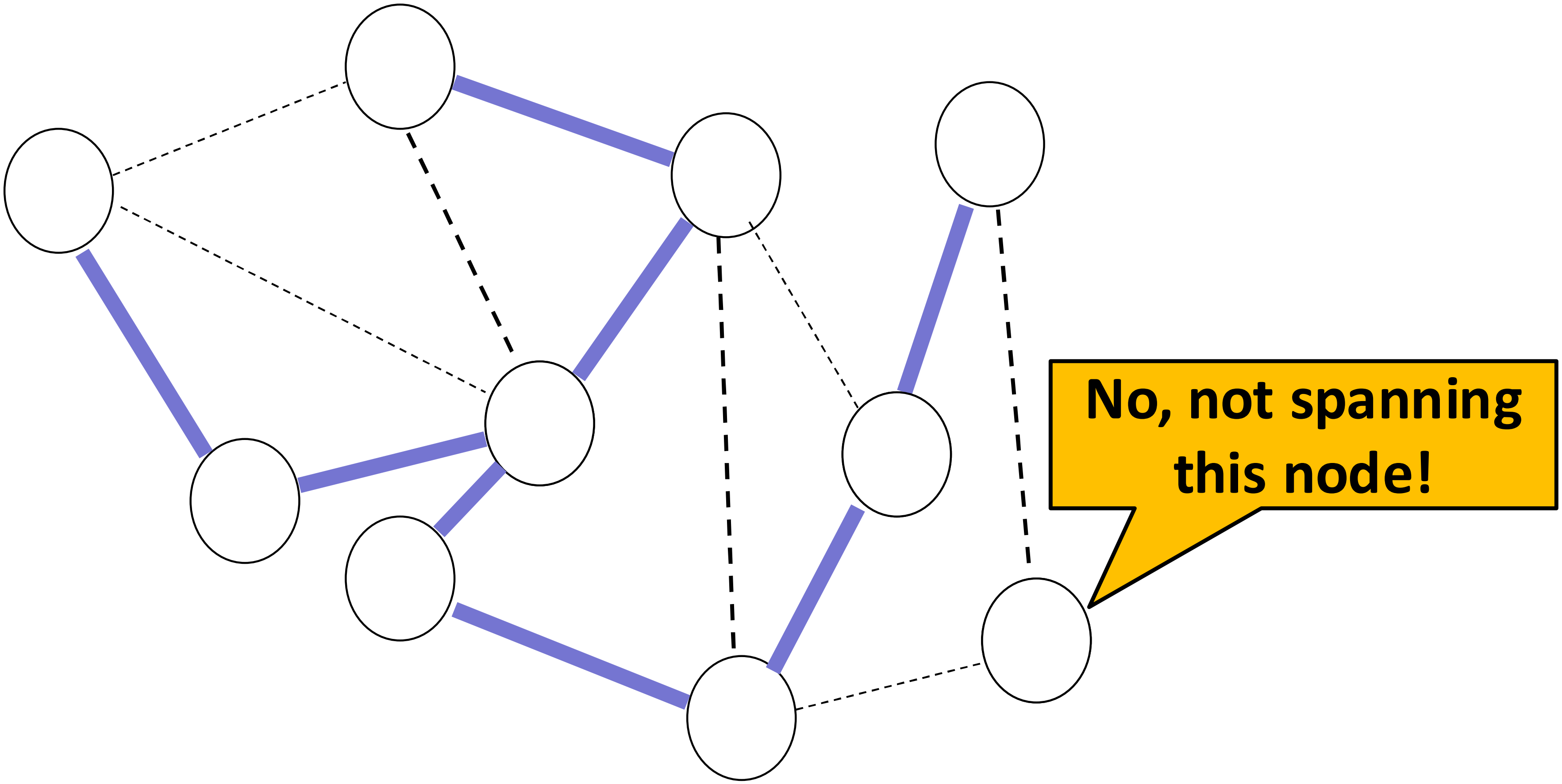


Spanning Trees

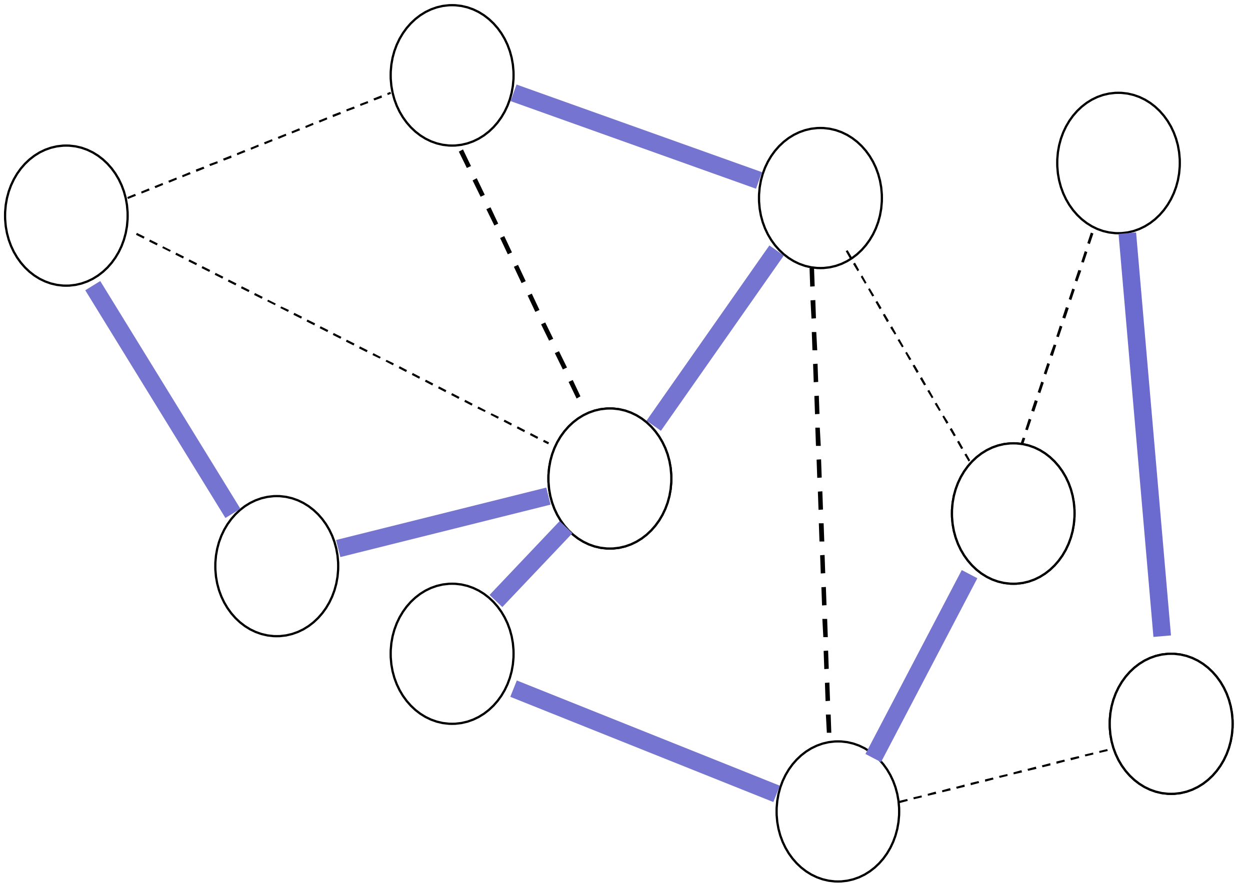


Is this a spanning tree?

Spanning Trees

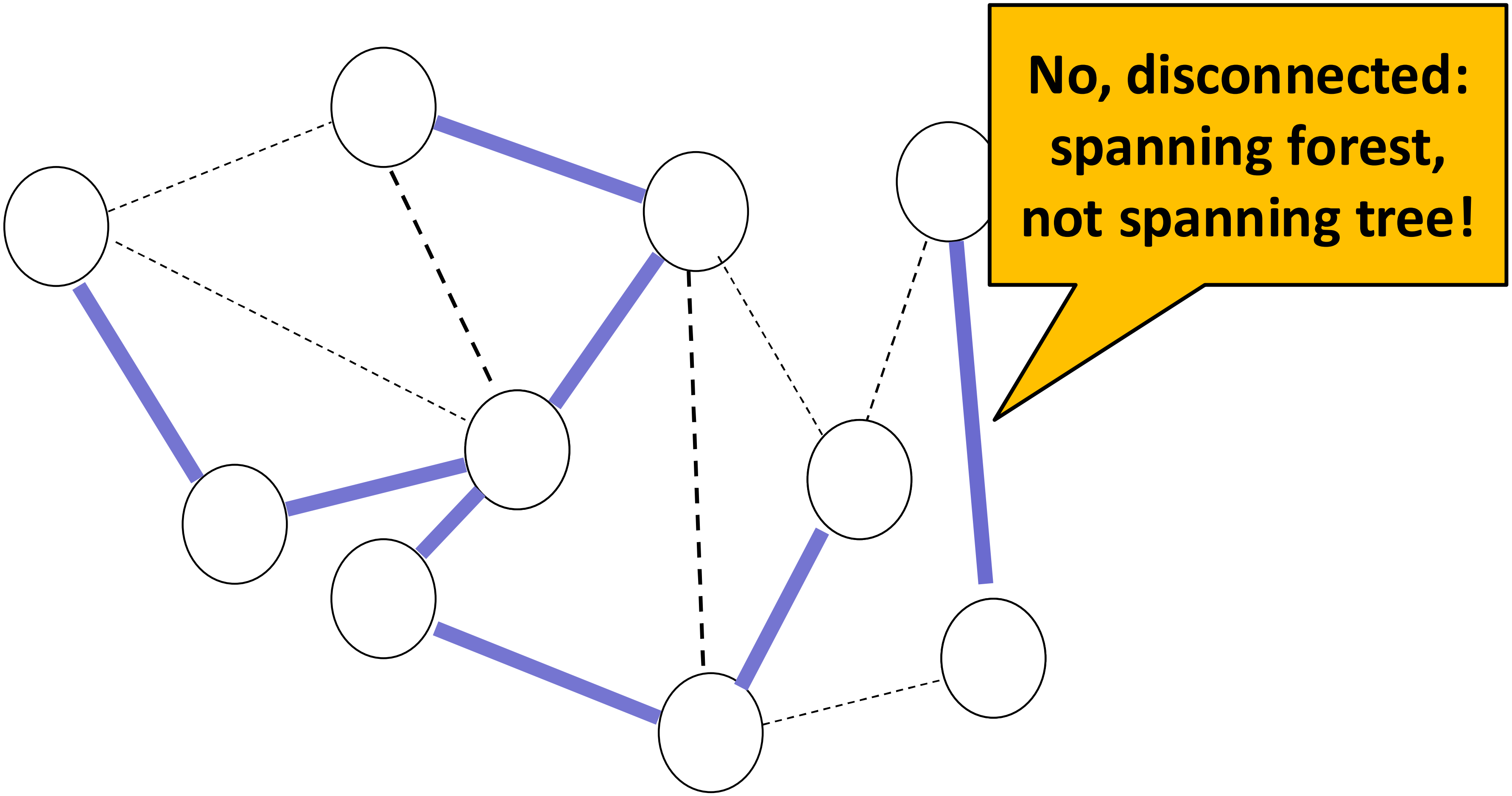


Spanning Trees

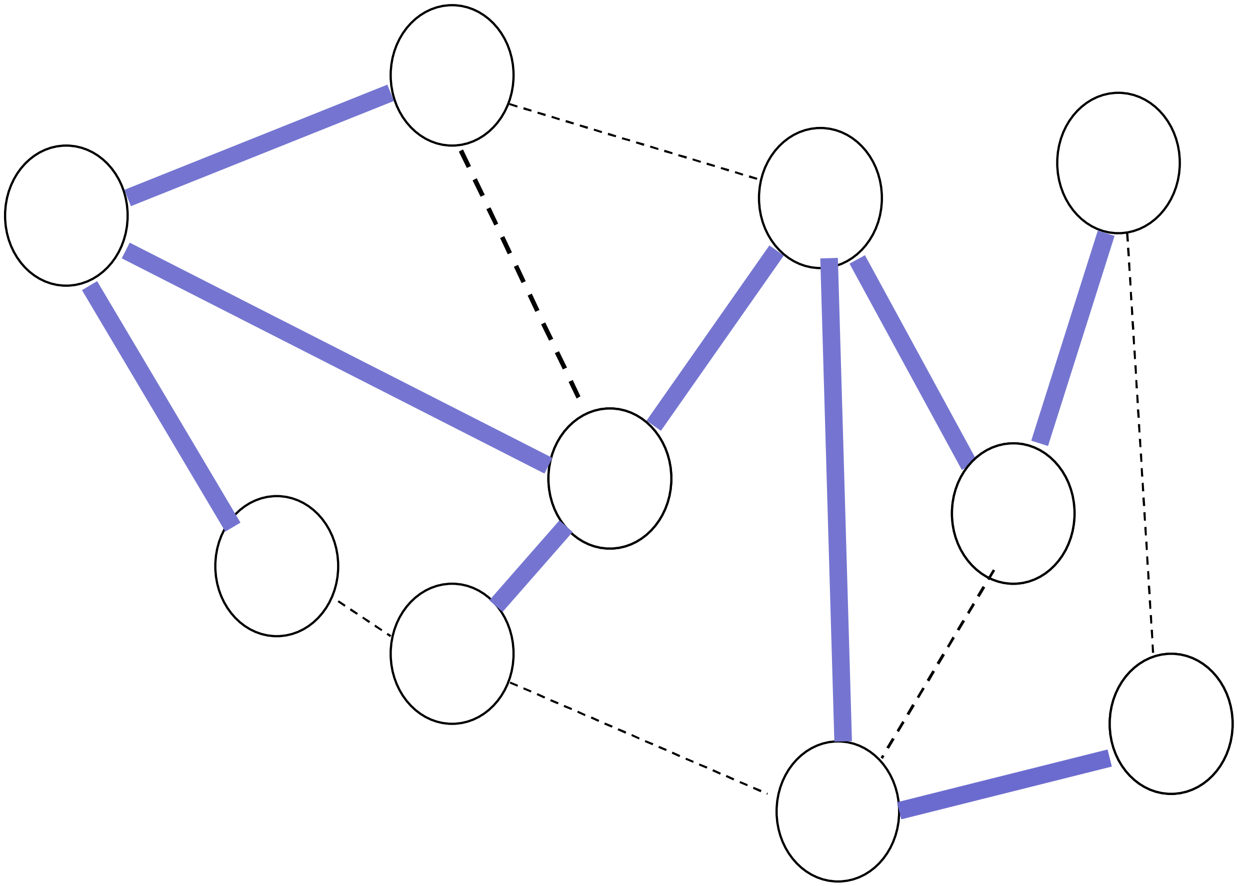


Is this a spanning tree?

Spanning Trees

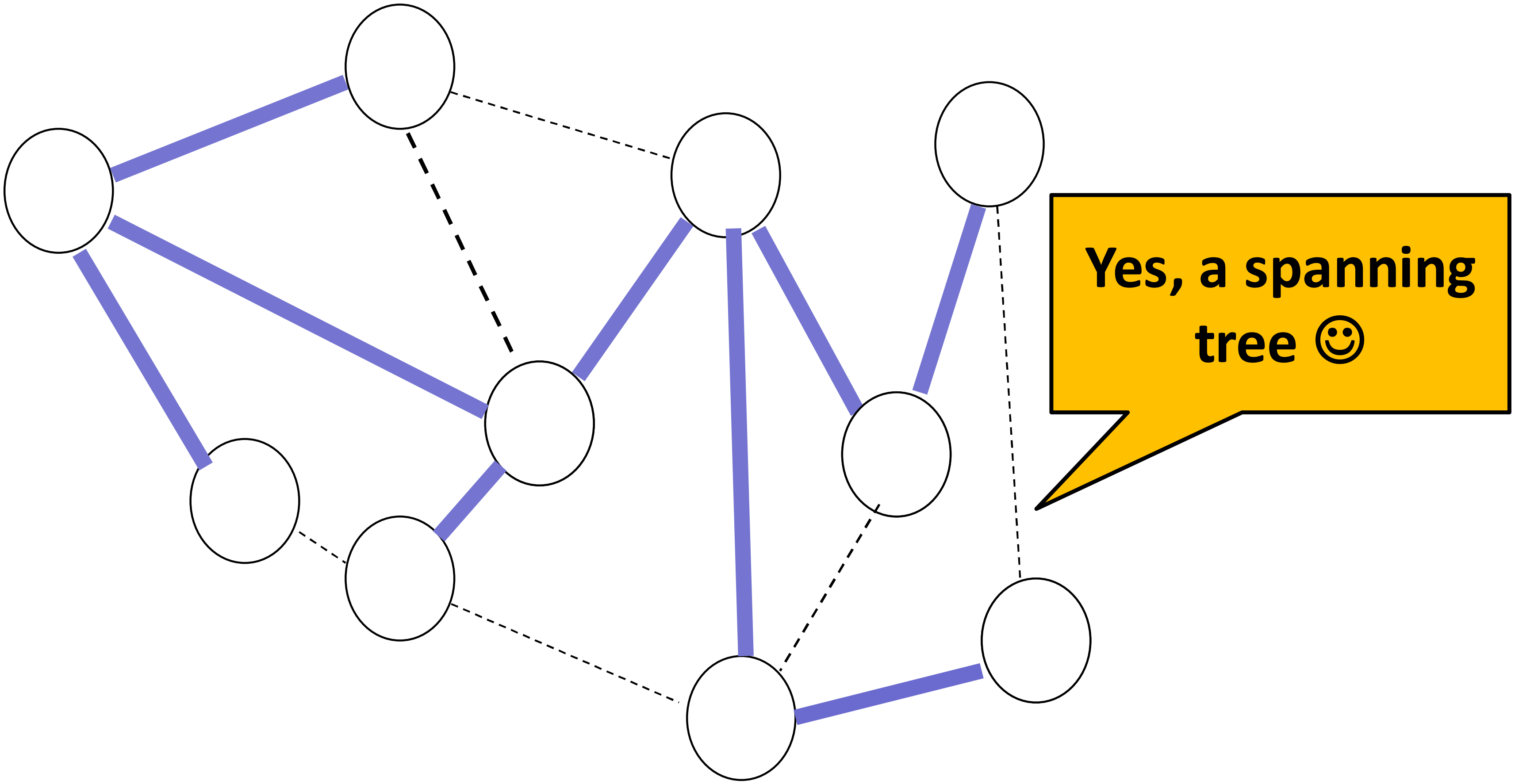


Spanning Trees



Is this a spanning tree?

Spanning Trees



Applications

Efficient Broadcast and Aggregation

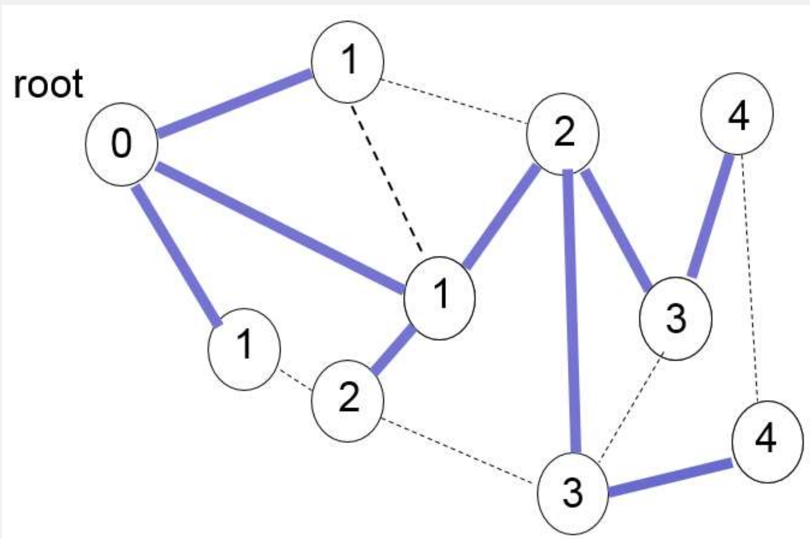
- ❑ Used in Ethernet network to avoid Layer-2 forwarding loops: Spanning Tree Protocol
- ❑ In ad-hoc networks: efficient backbone: broadcast and aggregate data using a linear number of transmissions

Algebraic Gossip

- ❑ Disseminating multiple messages in large communication network
- ❑ Random communication pattern with neighbors
- ❑ Gossip: based on local interactions

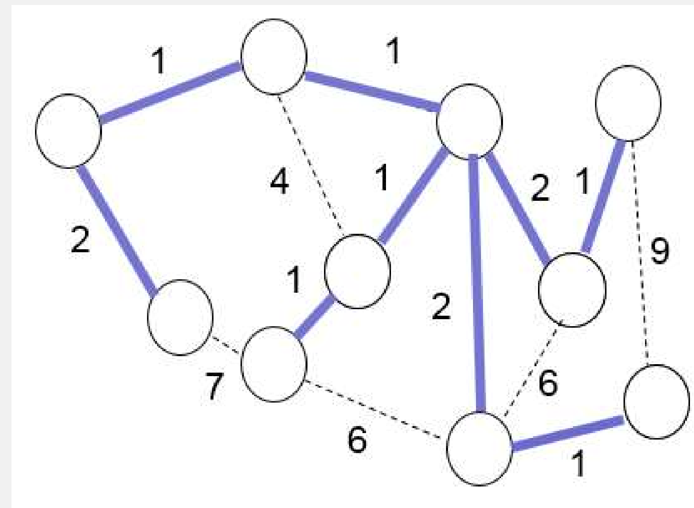
Types of Spanning Trees

BFS



- a.k.a. shortest distance spanning tree (may also be weighted)
- Spanning tree includes shortest paths from **a given root** to all nodes
- Interesting e.g. for fast broadcast

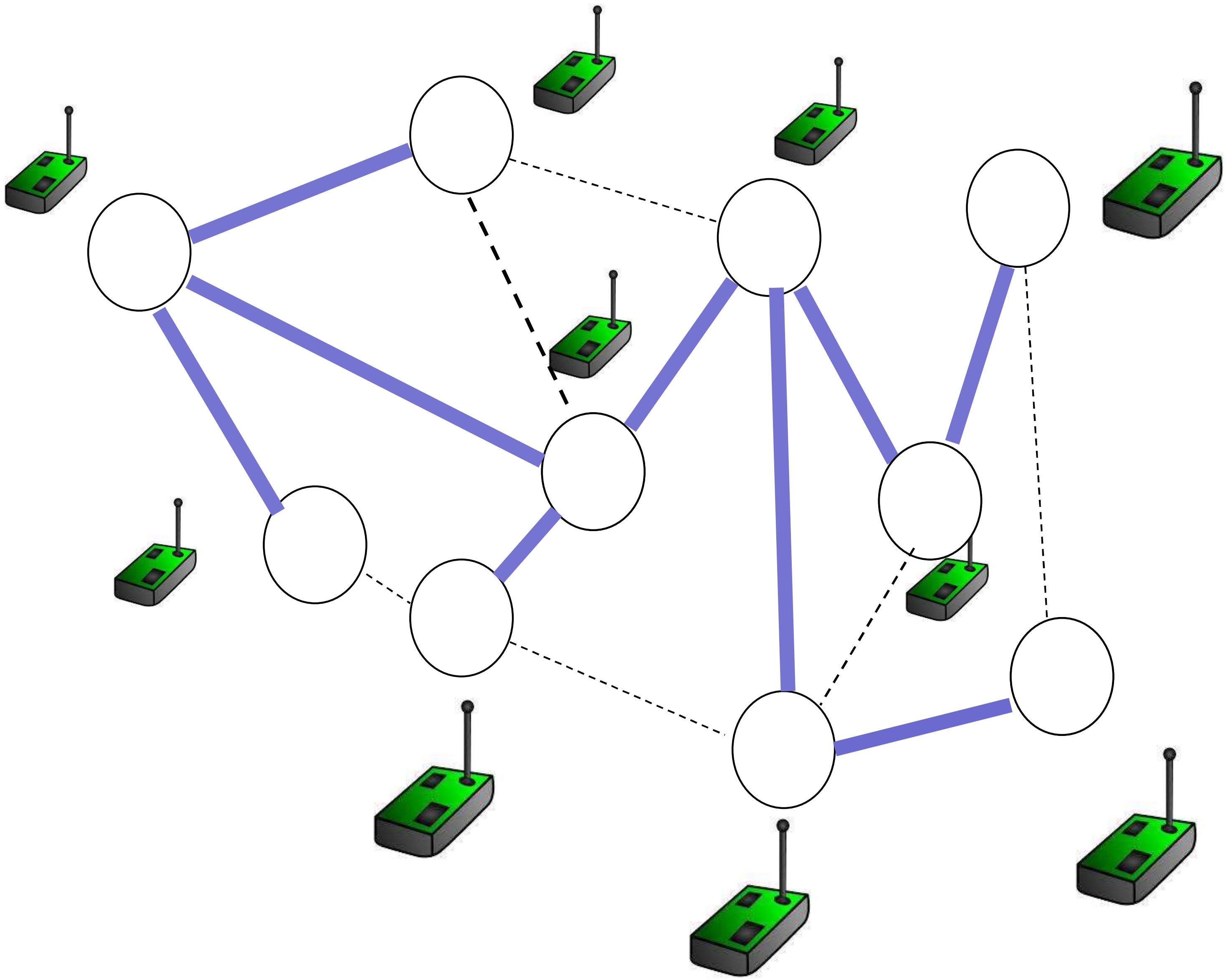
MST



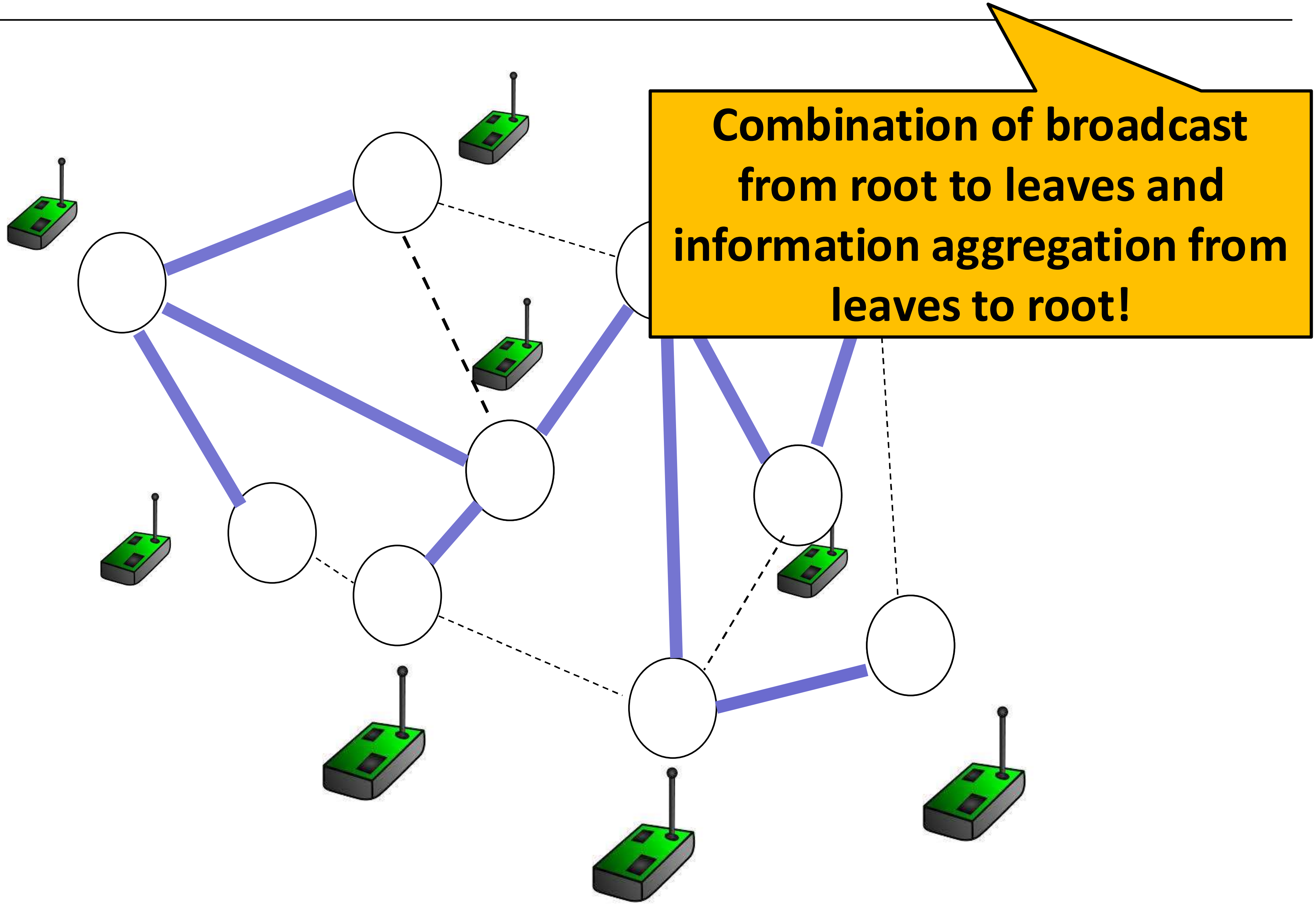
- Minimum link cost spanning tree
- Interesting, e.g., for least routing cost or energy cost

How to compute a spanning tree in the LOCAL model?

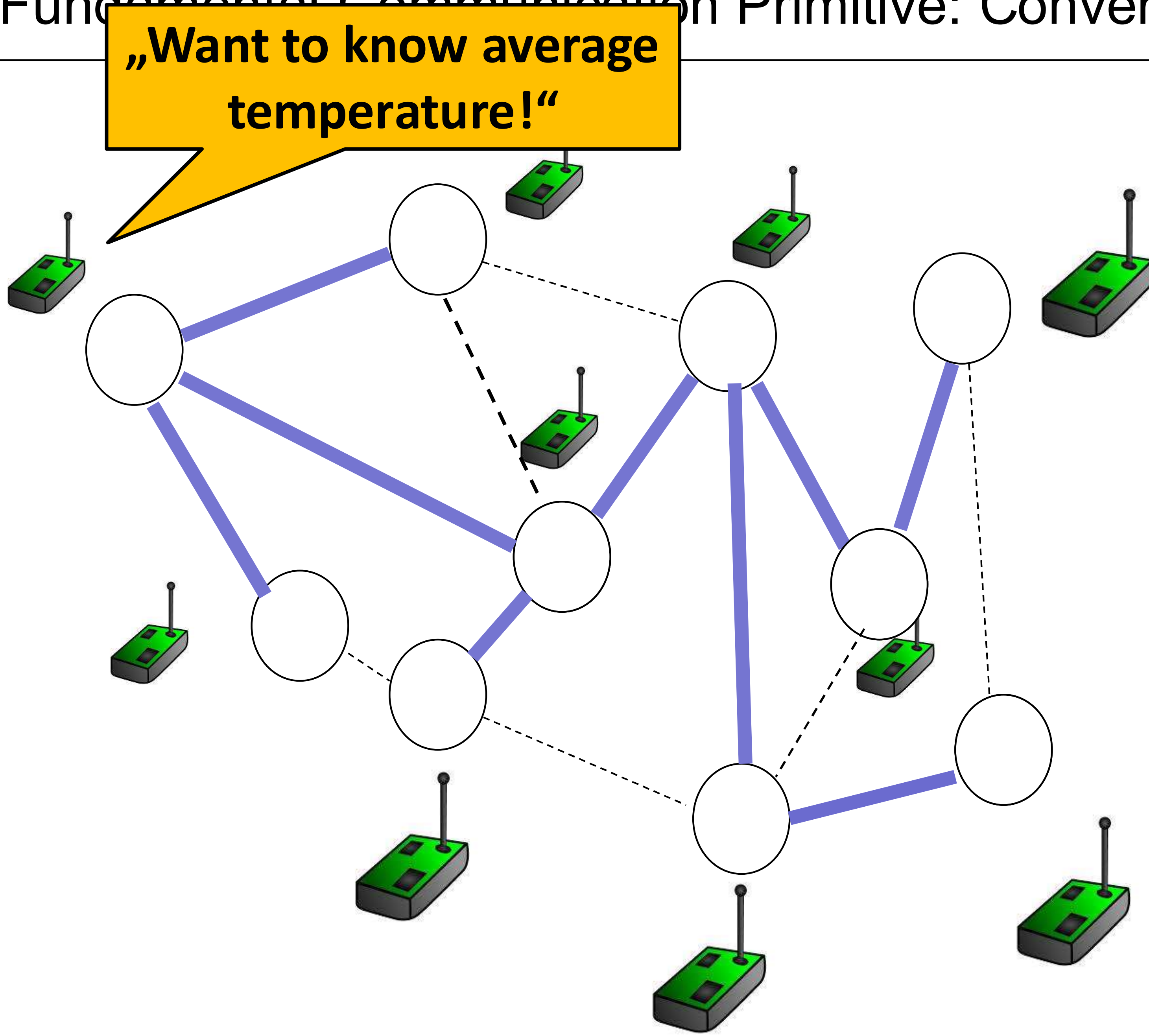
A Fundamental Communication Primitive: ConvergeCast



A Fundamental Communication Primitive: ConvergeCast

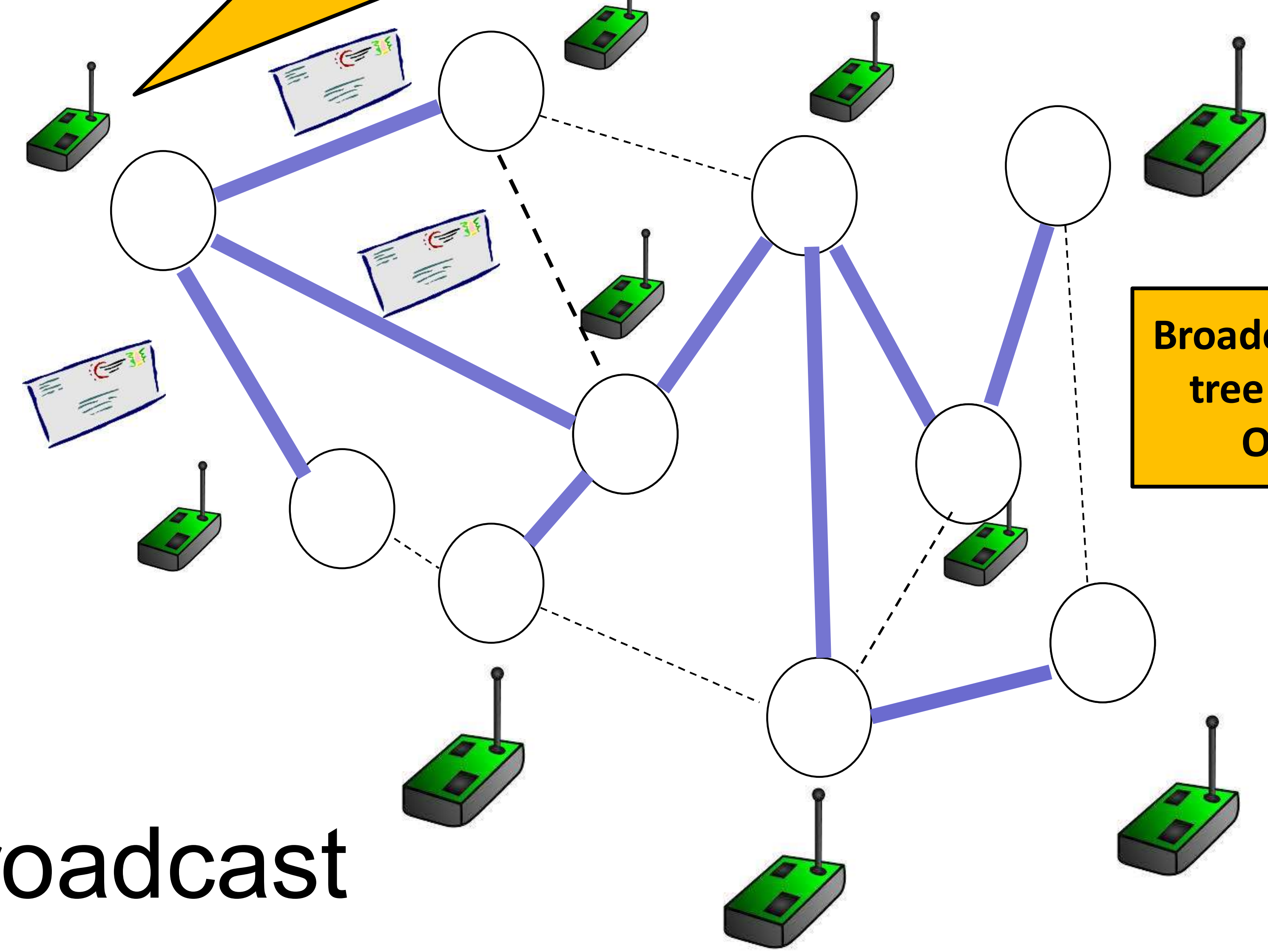


A Fundamental Communication Primitive: ConvergeCast



A Fundamental Communication Primitive: ConvergeCast

„Want to know average temperature!“



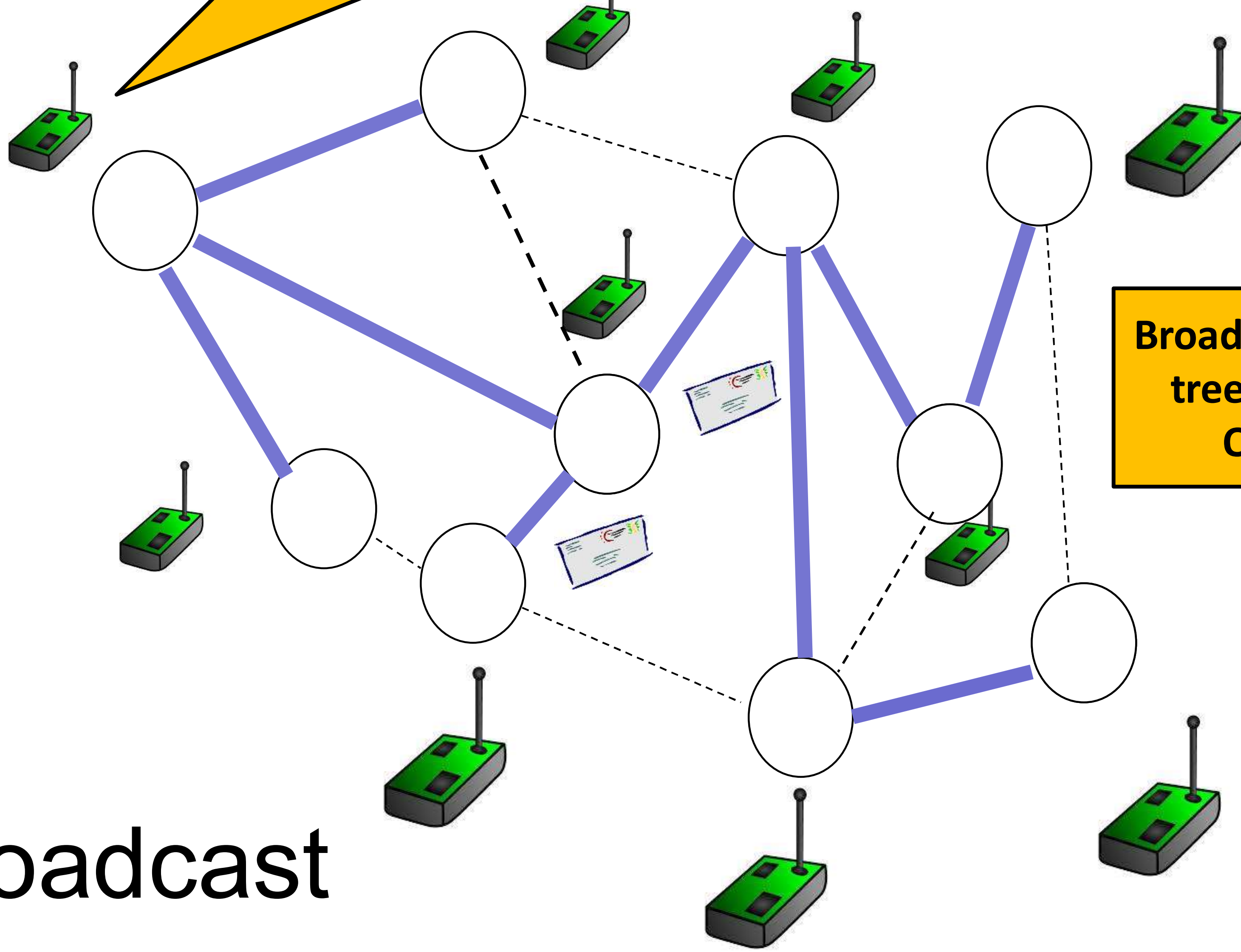
Broadcast along spanning tree: O(n) rather than O(m) messages!

Broadcast

Round 1

A Fundamental Communication Primitive: ConvergeCast

„Want to know average temperature!“

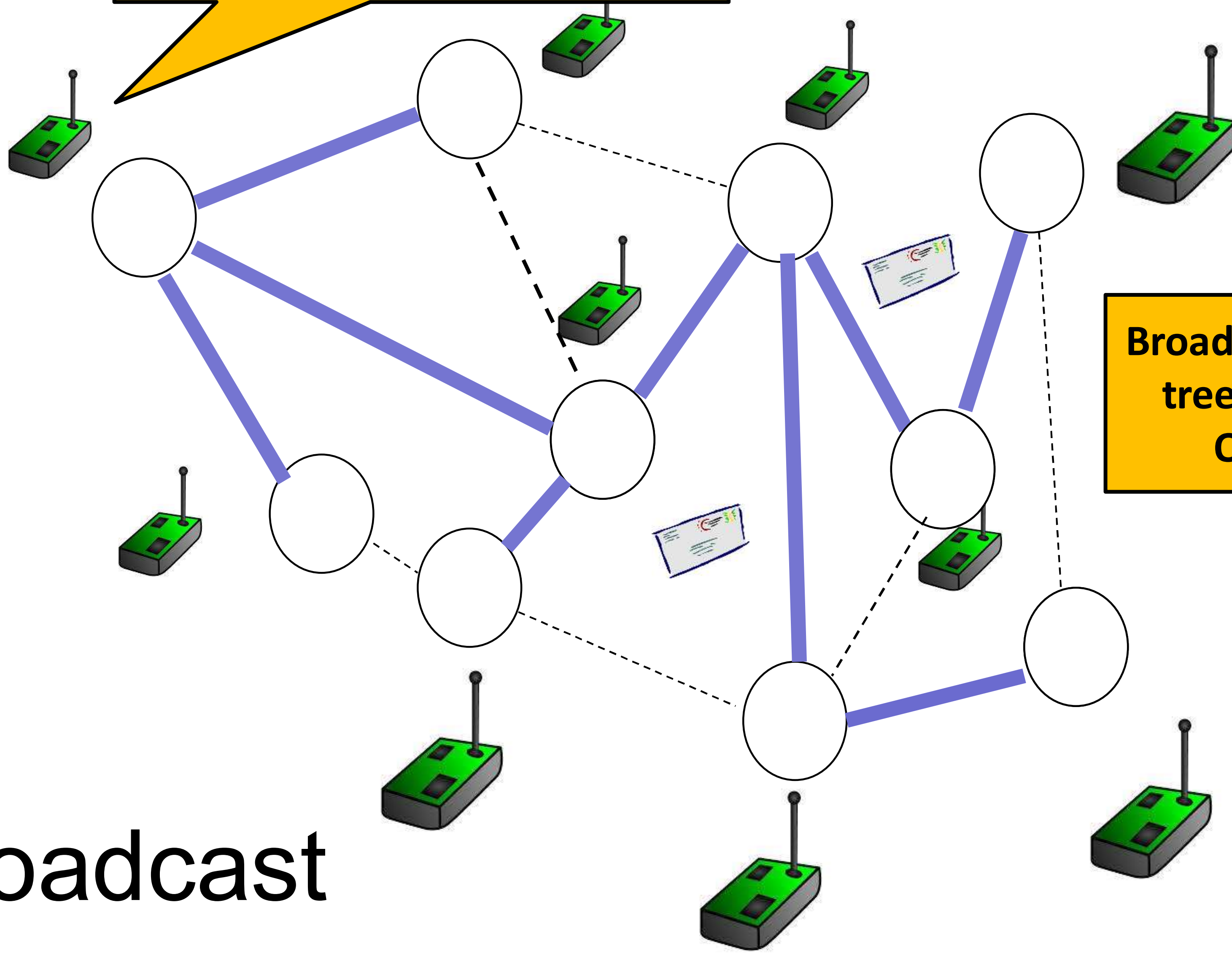


Broadcast along spanning tree: O(n) rather than O(m) messages!

Broadcast
Round 2

A Fundamental Communication Primitive: ConvergeCast

„Want to know average temperature!“

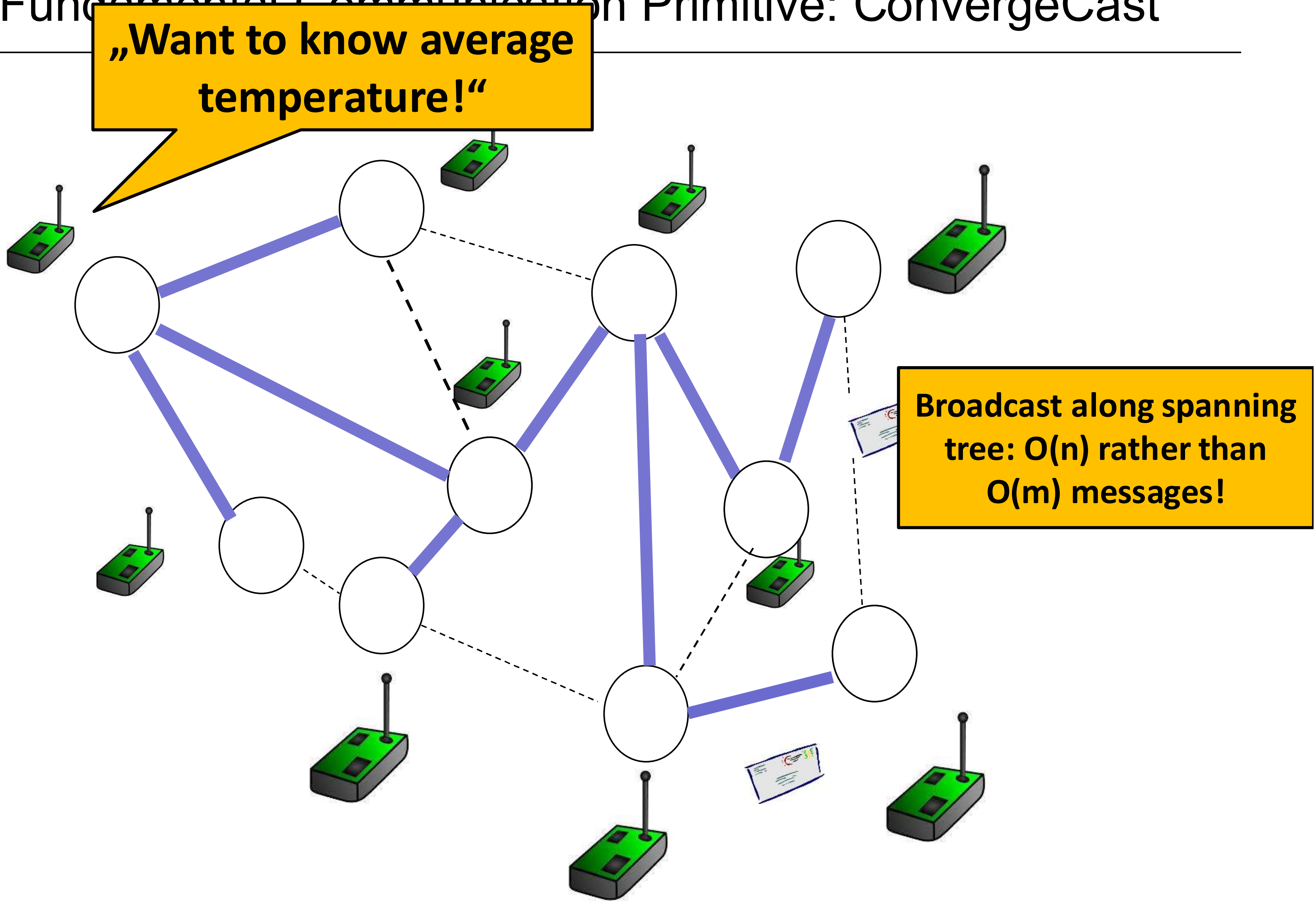


Broadcast along spanning tree: O(n) rather than O(m) messages!

Broadcast

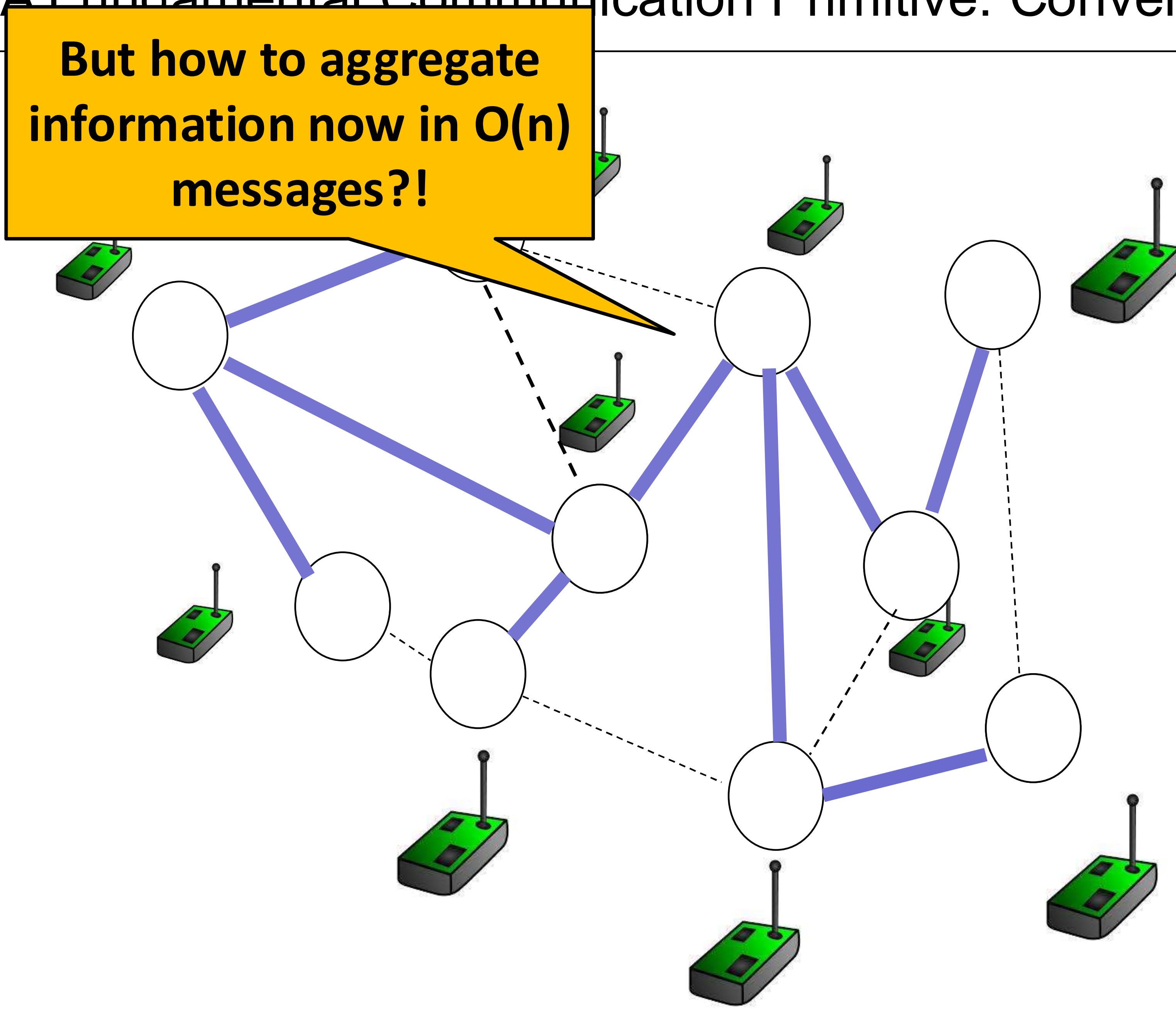
Round 3

A Fundamental Communication Primitive: ConvergeCast

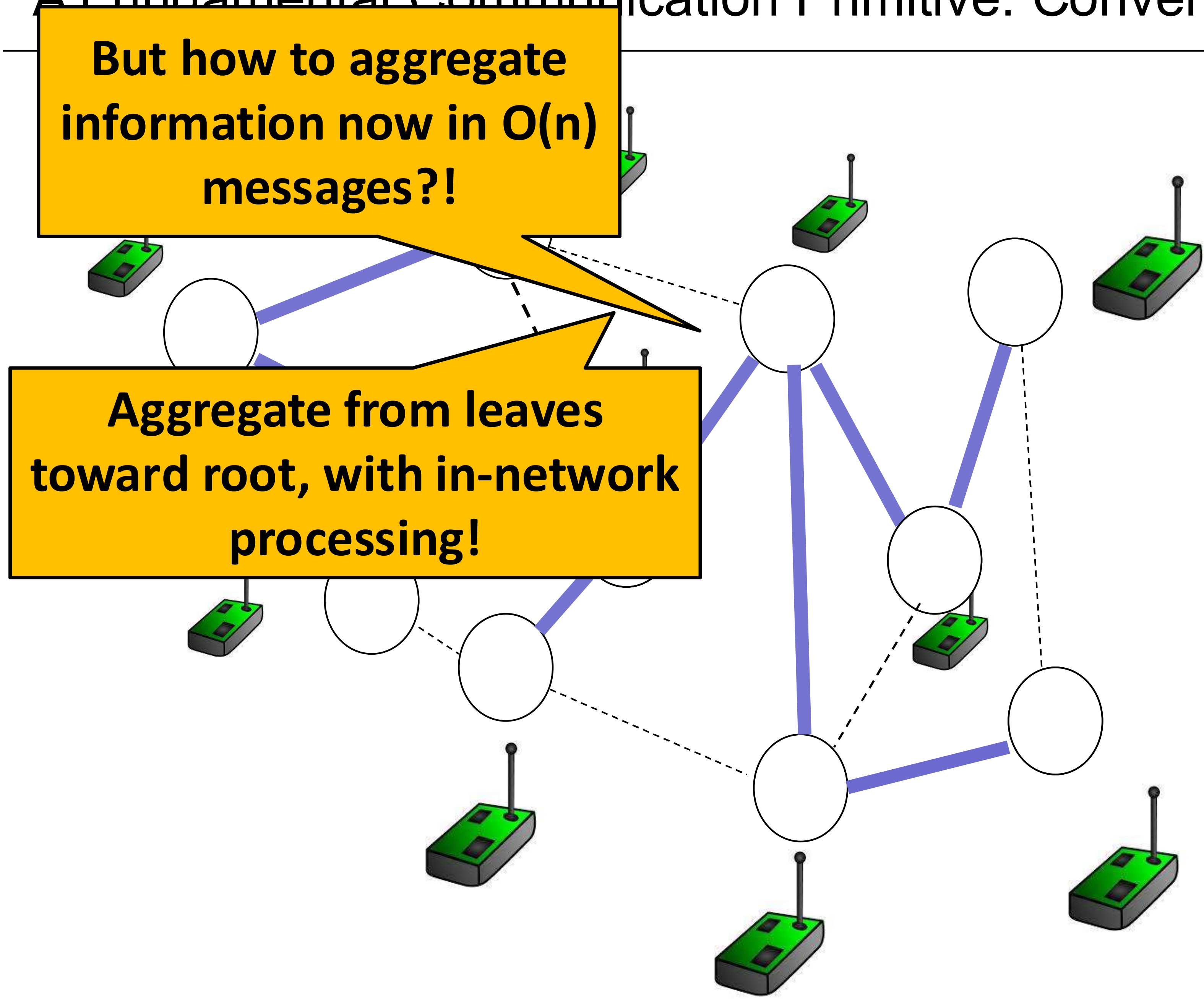


O(n) messages for broadcast!

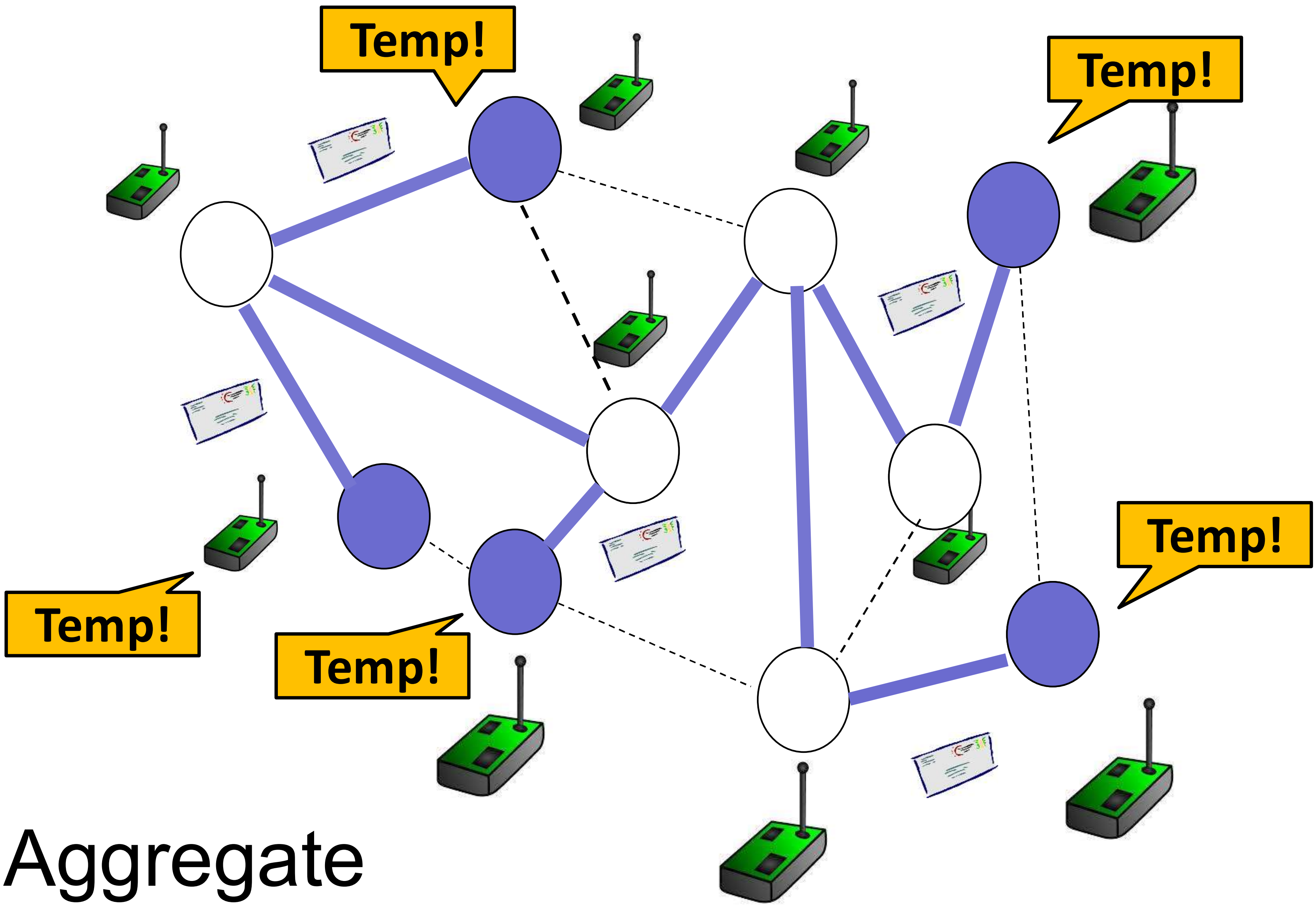
A Fundamental Communication Primitive: ConvergeCast



A Fundamental Communication Primitive: ConvergeCast



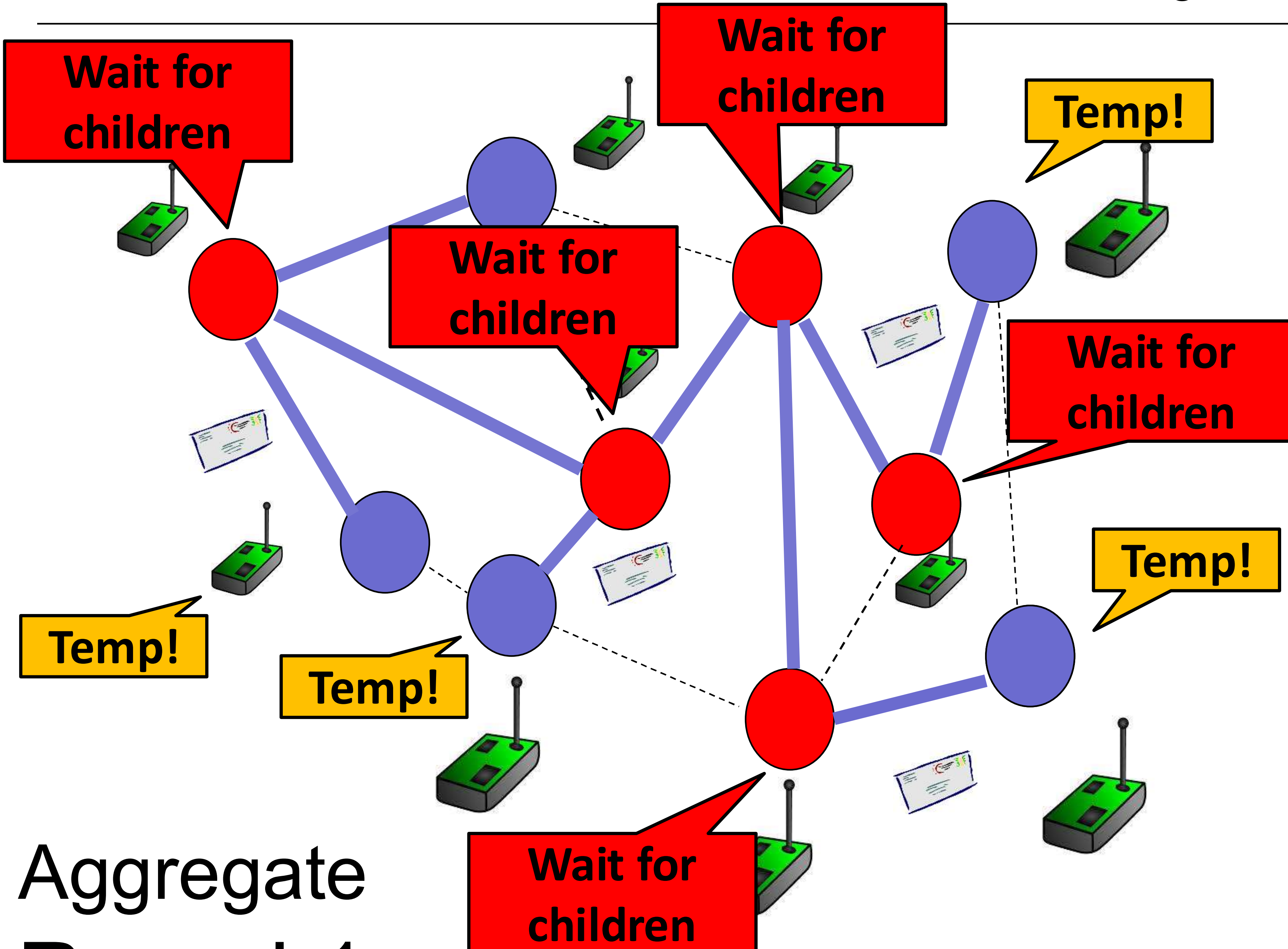
A Fundamental Communication Primitive: ConvergeCast



Aggregate

Round 1

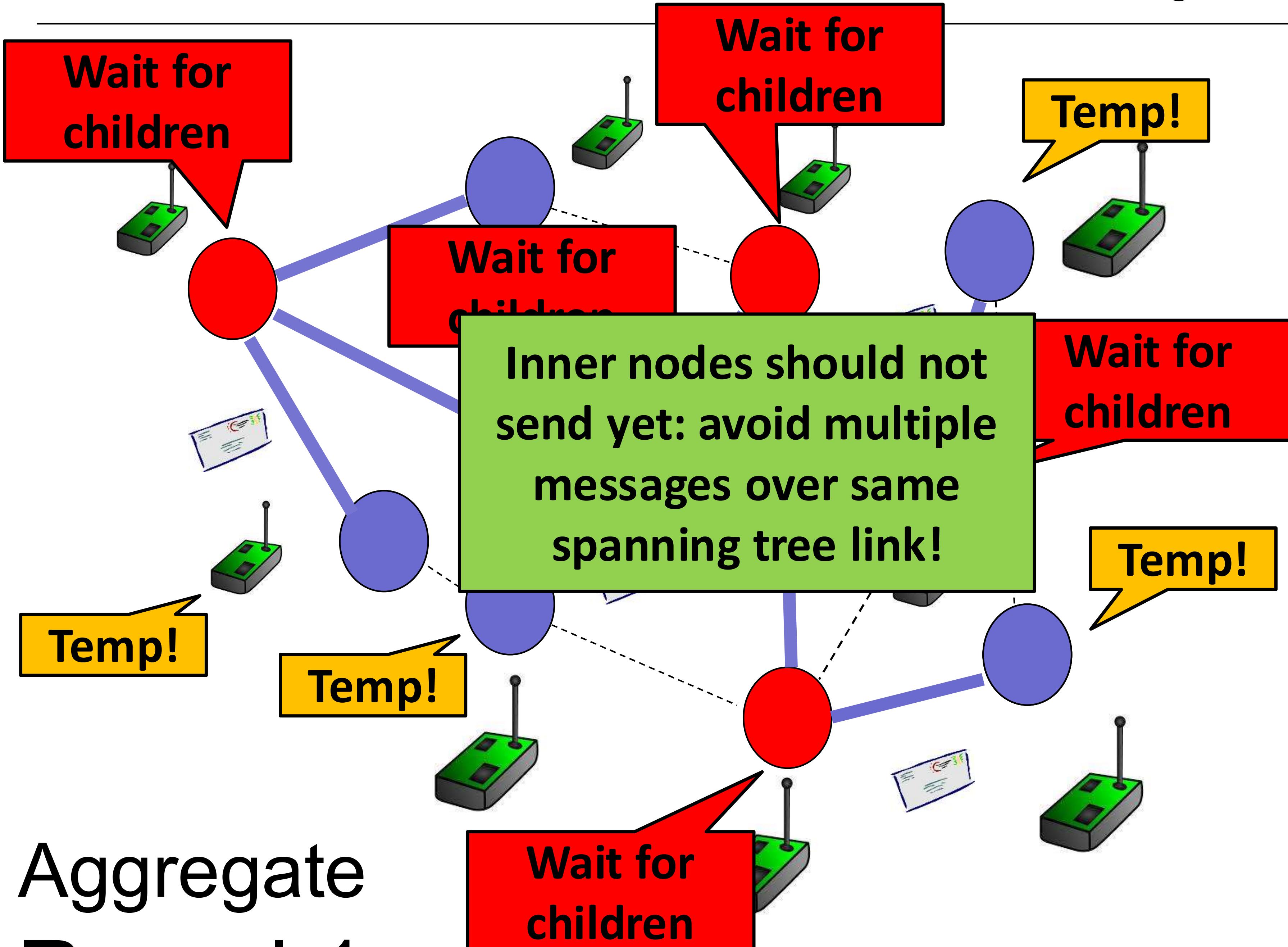
A Fundamental Communication Primitive: ConvergeCast



Aggregate

Round 1

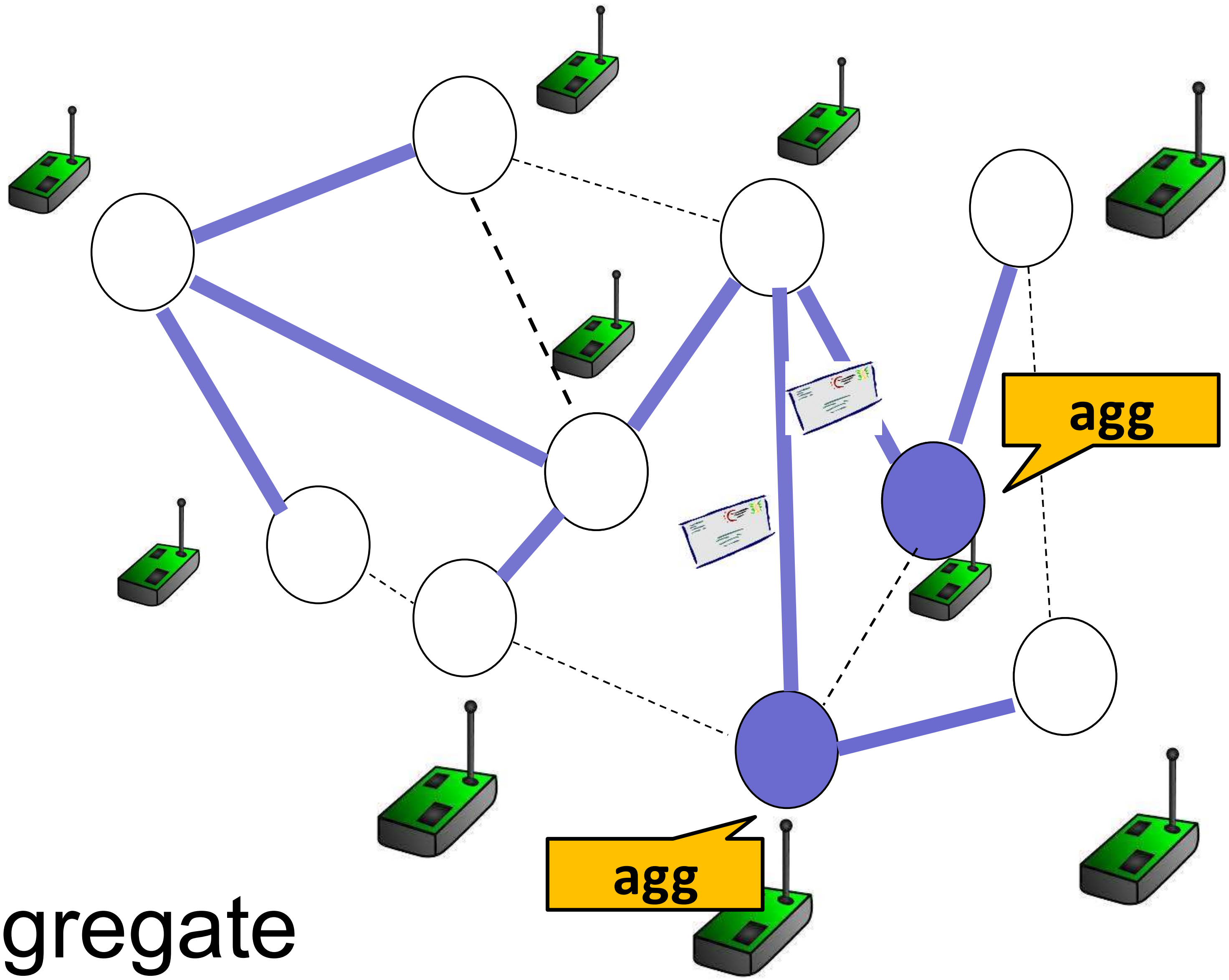
A Fundamental Communication Primitive: ConvergeCast



Aggregate

Round 1

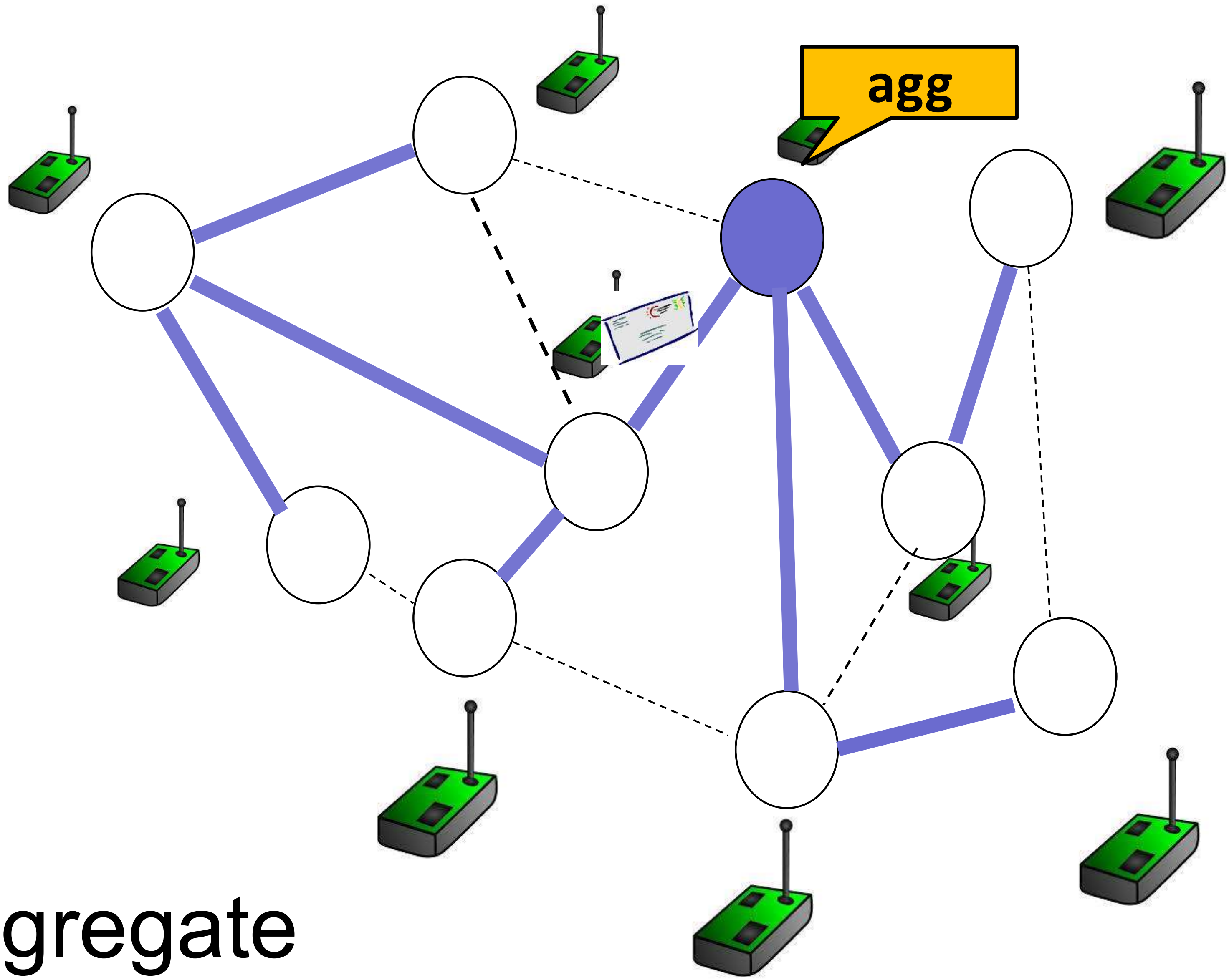
A Fundamental Communication Primitive: ConvergeCast



Aggregate

Round 2

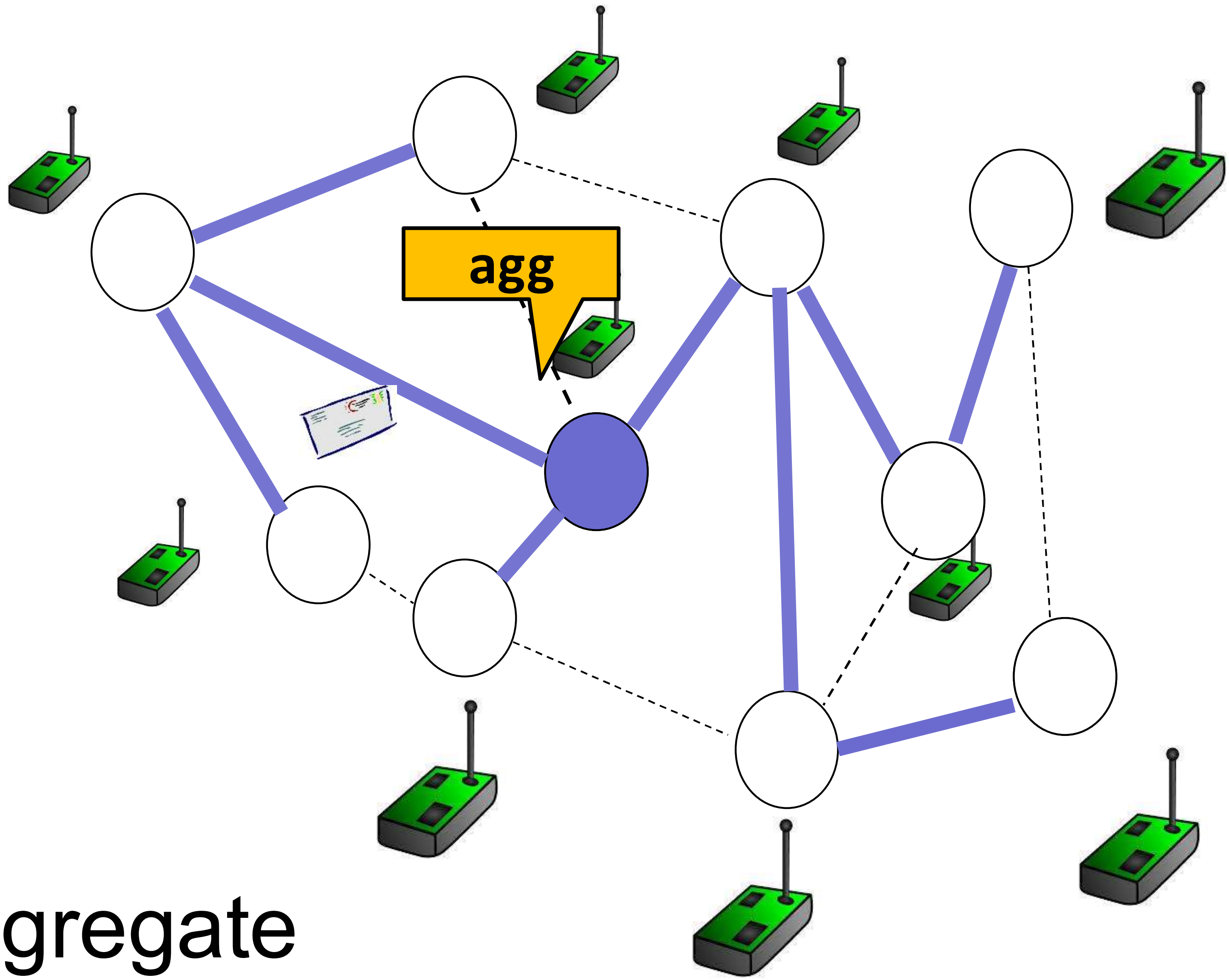
A Fundamental Communication Primitive: ConvergeCast



Aggregate

Round 3

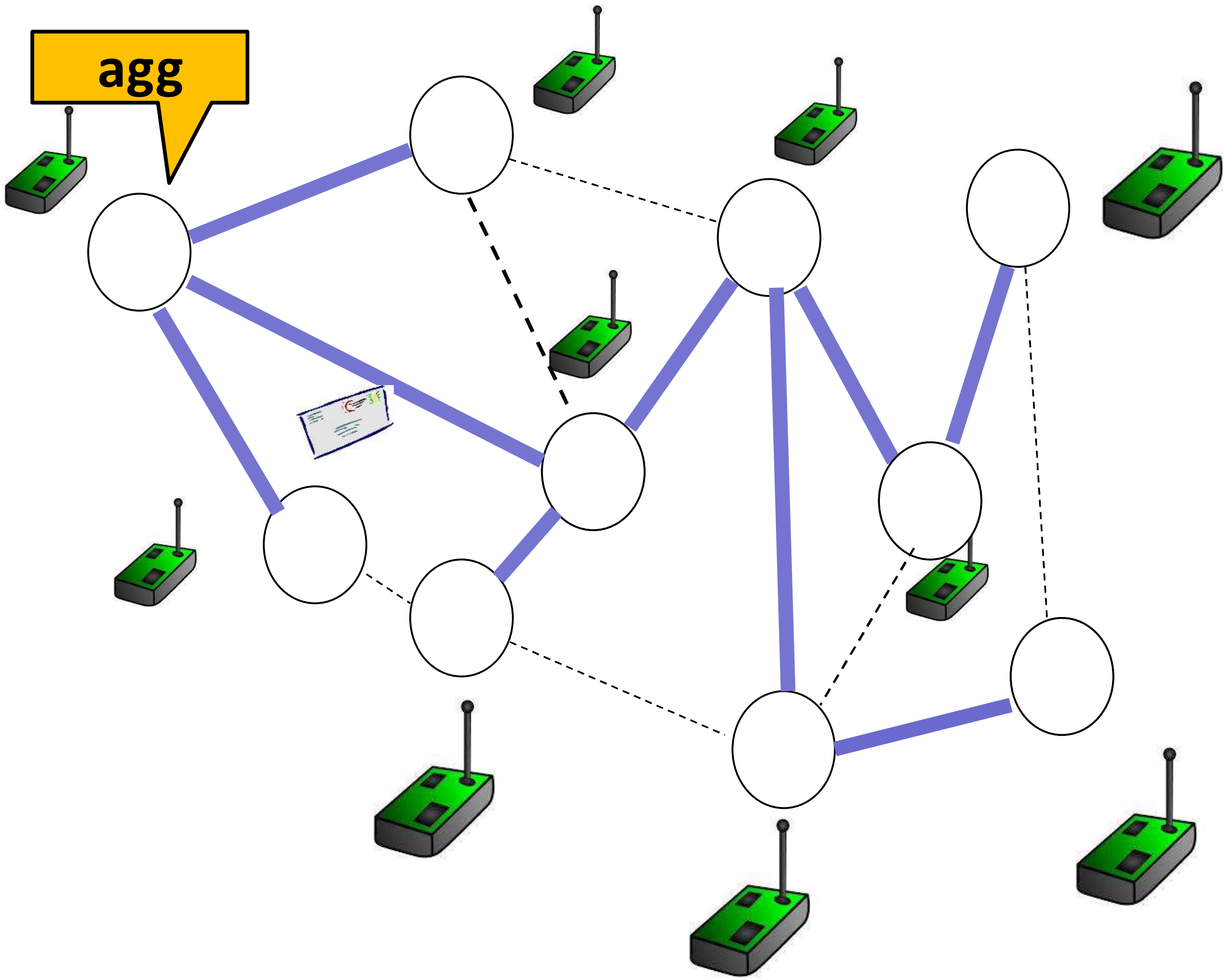
A Fundamental Communication Primitive: ConvergeCast



Aggregate

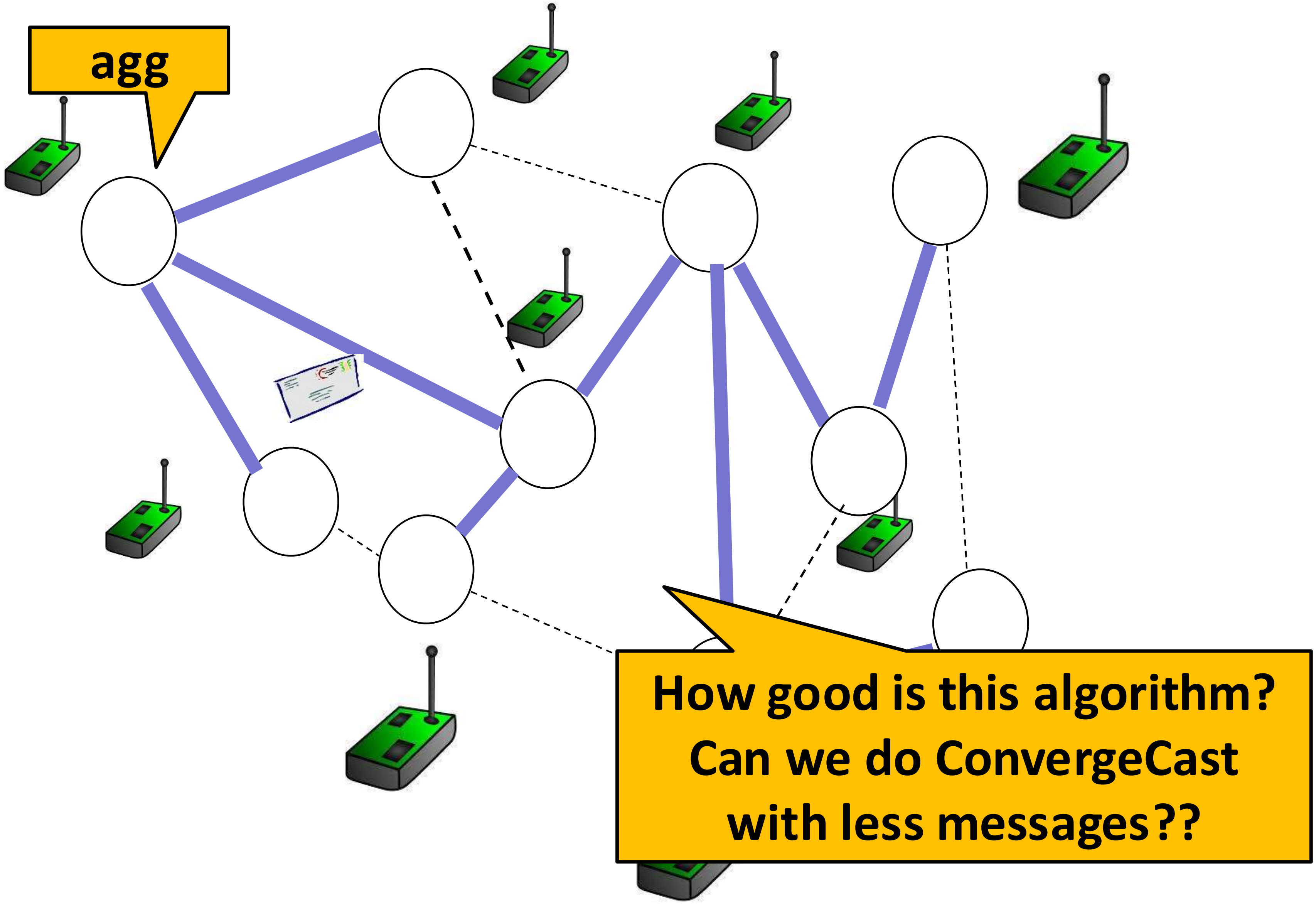
Round 4

A Fundamental Communication Primitive: ConvergeCast



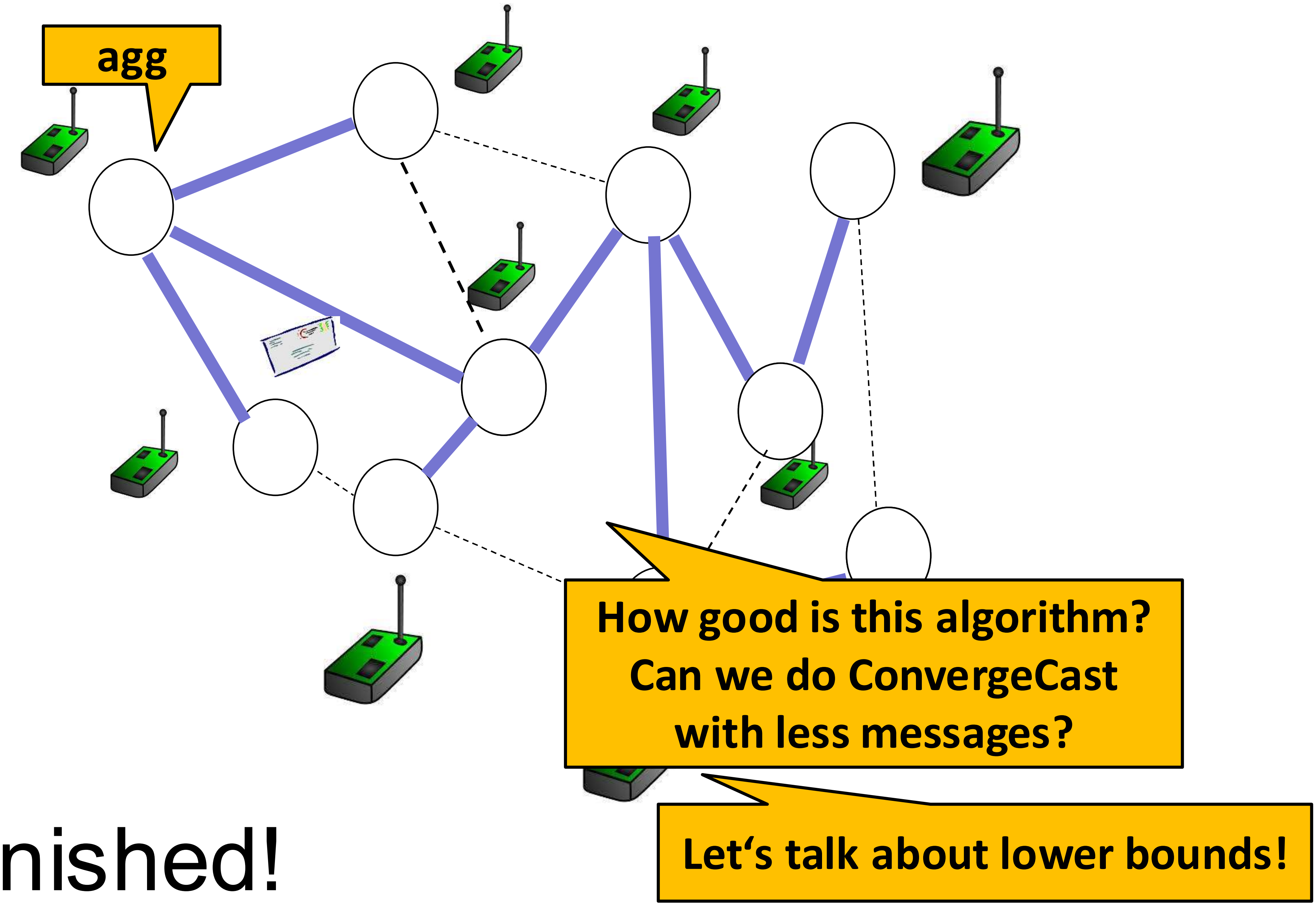
Finished!

A Fundamental Communication Primitive: ConvergeCast



Finished!

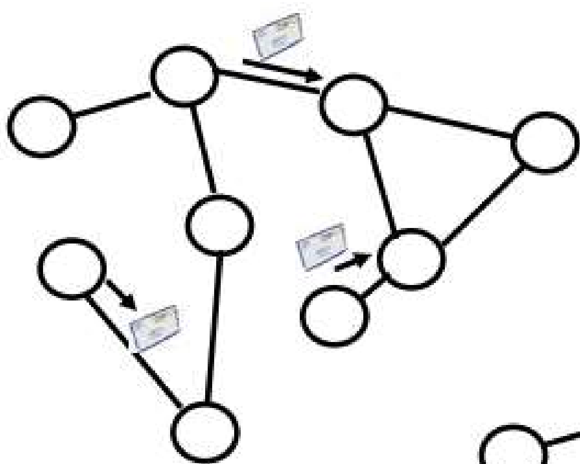
A Fundamental Communication Primitive: ConvergeCast



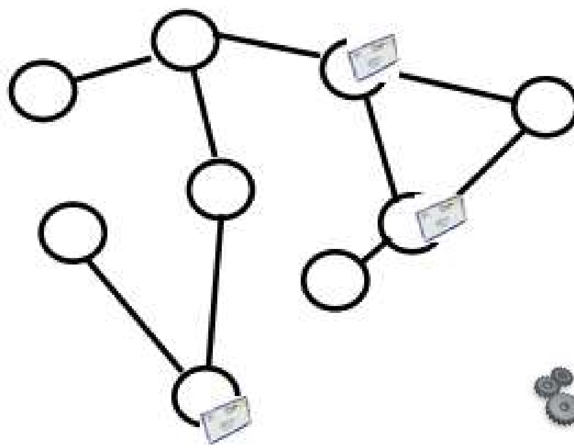
Finished!

Recall: Local Algorithm

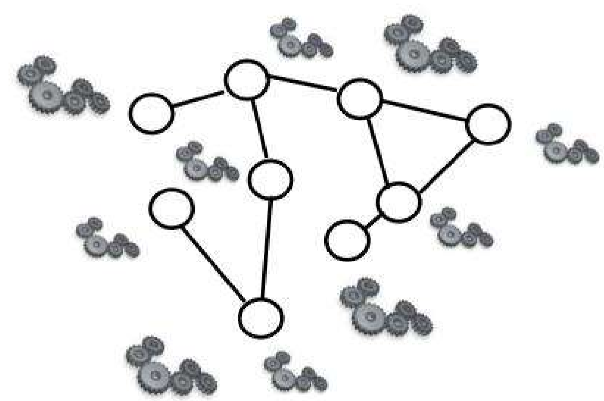
Send...



... receive...



... compute.



Let us introduce some definitions

Distance, Radius, Diameter

Distance between two nodes is # hops.
Radius of a node is max distance to any other node.
Radius of graph is *minimum* radius of any node.
Diameter of graph is *max* distance between any two nodes.

Relationship between R and D?

L[...]ons

In general: R ≤ D ≤ 2R.
max distance cannot be longer than going through this node.

Distance, Radius, Diameter

Distance between two nodes is # hops.
Radius of a node is max distance to any other node.
Radius of graph is *minimum* radius of any node.
Diameter of graph is *max* distance between any two nodes.

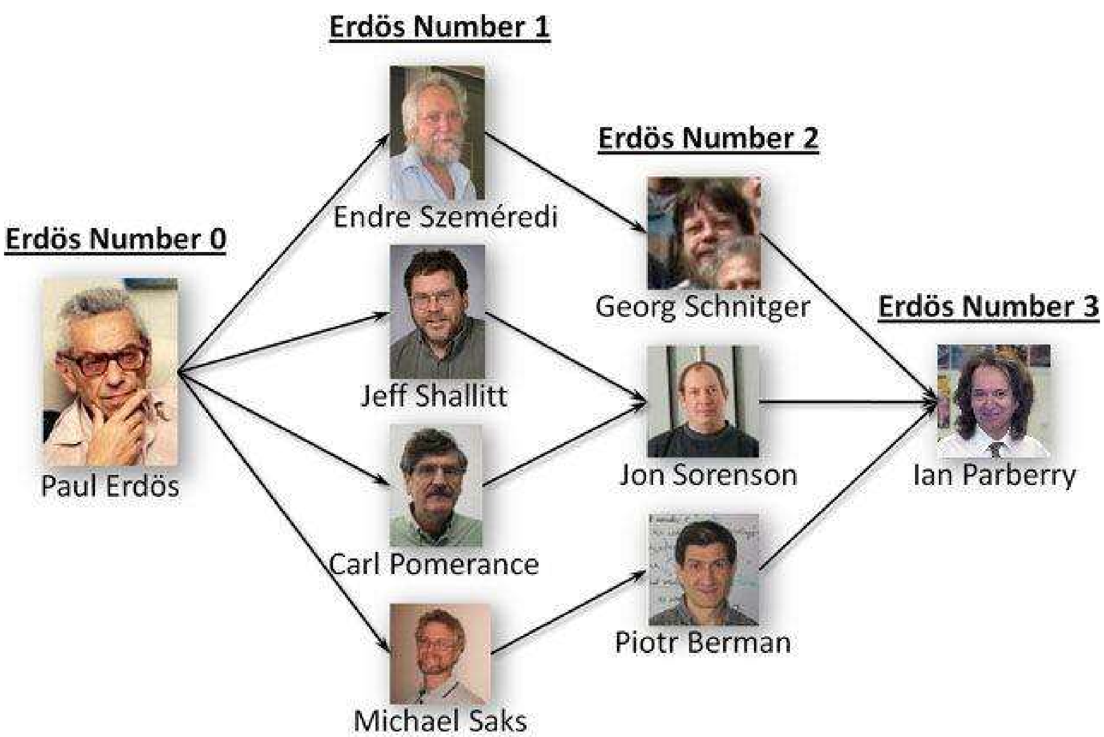
In the complete graph, for all nodes: R=D.

On the line, for broder nodes: 2R=D.

Relevance: Radius

People enjoy identifying nodes of small radius in a graph!

E.g., Erdös number, Kevin Bacon number, joint Erdös-Bacon number, etc.

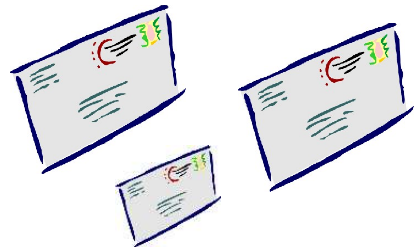


Kevin Bacon Number	# of People
0	1
1	3211
2	376831
3	1359872
4	347806
5	29593
6	3496
7	515
8	102
9	8
10	1

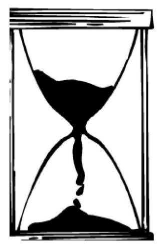
Total number of linkable actors: 2121436
Weighted total of linkable actors: 6401157
Average Kevin Bacon number: 3.017

Lower Bounds for Broadcast

Message complexity?

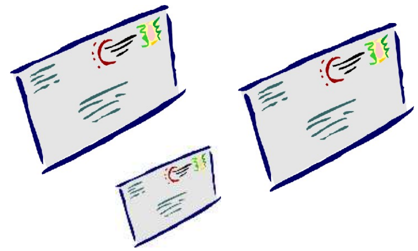


Time complexity?



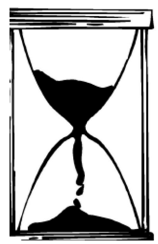
Lower Bounds for Broadcast

Message complexity?



Each node must receive message: so at least n-1.

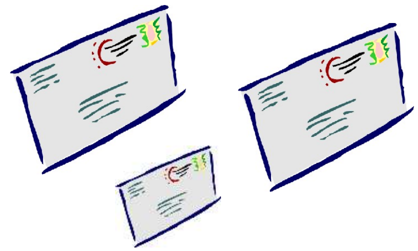
Time complexity?



The radius of the source: each node needs to receive message.

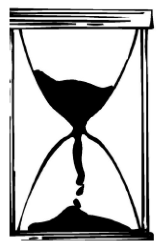
Lower Bounds for Broadcast

Message complexity?



Each node must receive message: so at least n-1.

Time complexity?

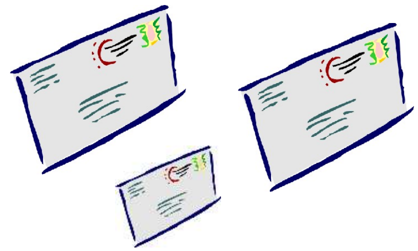


The radius of the source: each node needs to receive message.

How to achieve this?

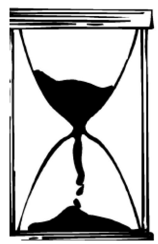
Lower Bounds for Broadcast

Message complexity?



Each node must receive message: so at least n-1.

Time complexity?

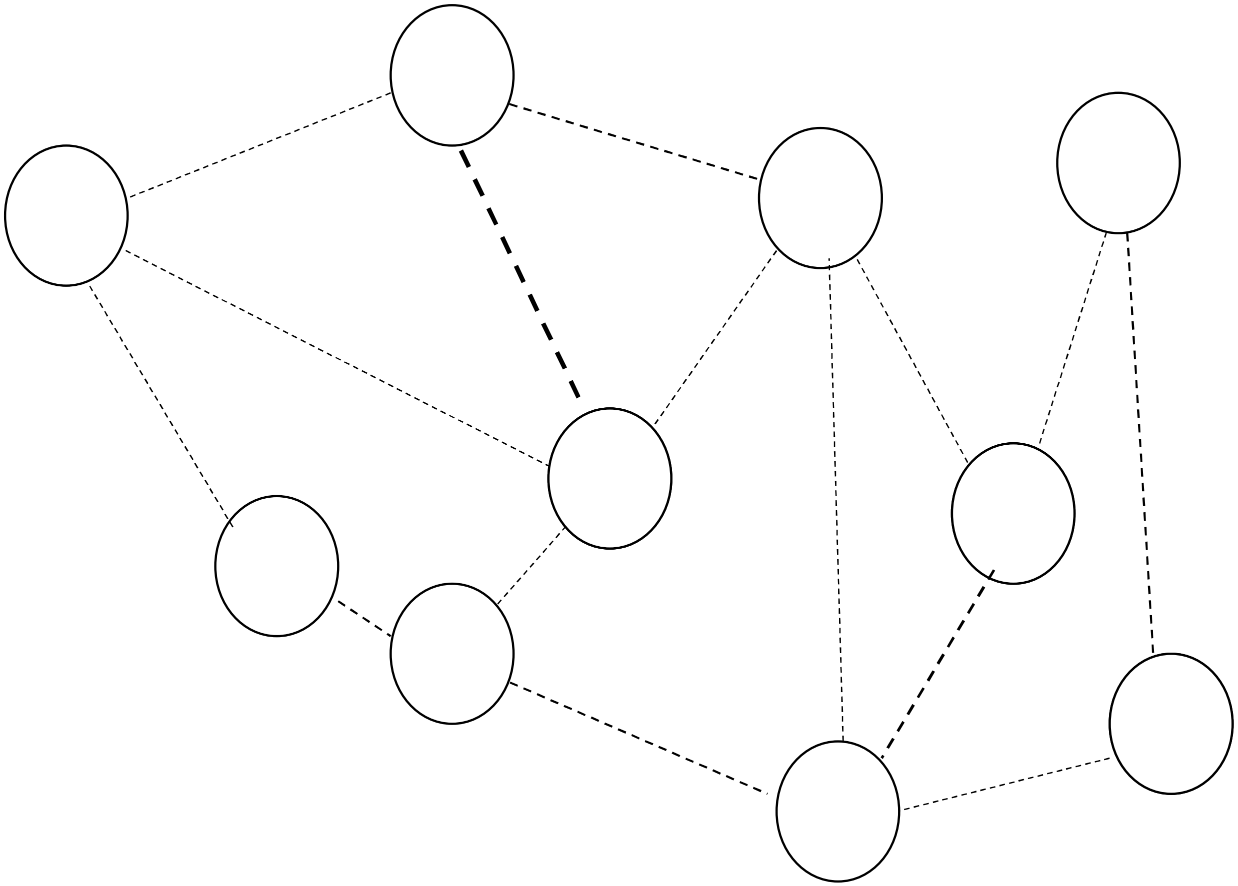


The radius of the source: each node needs to receive message.

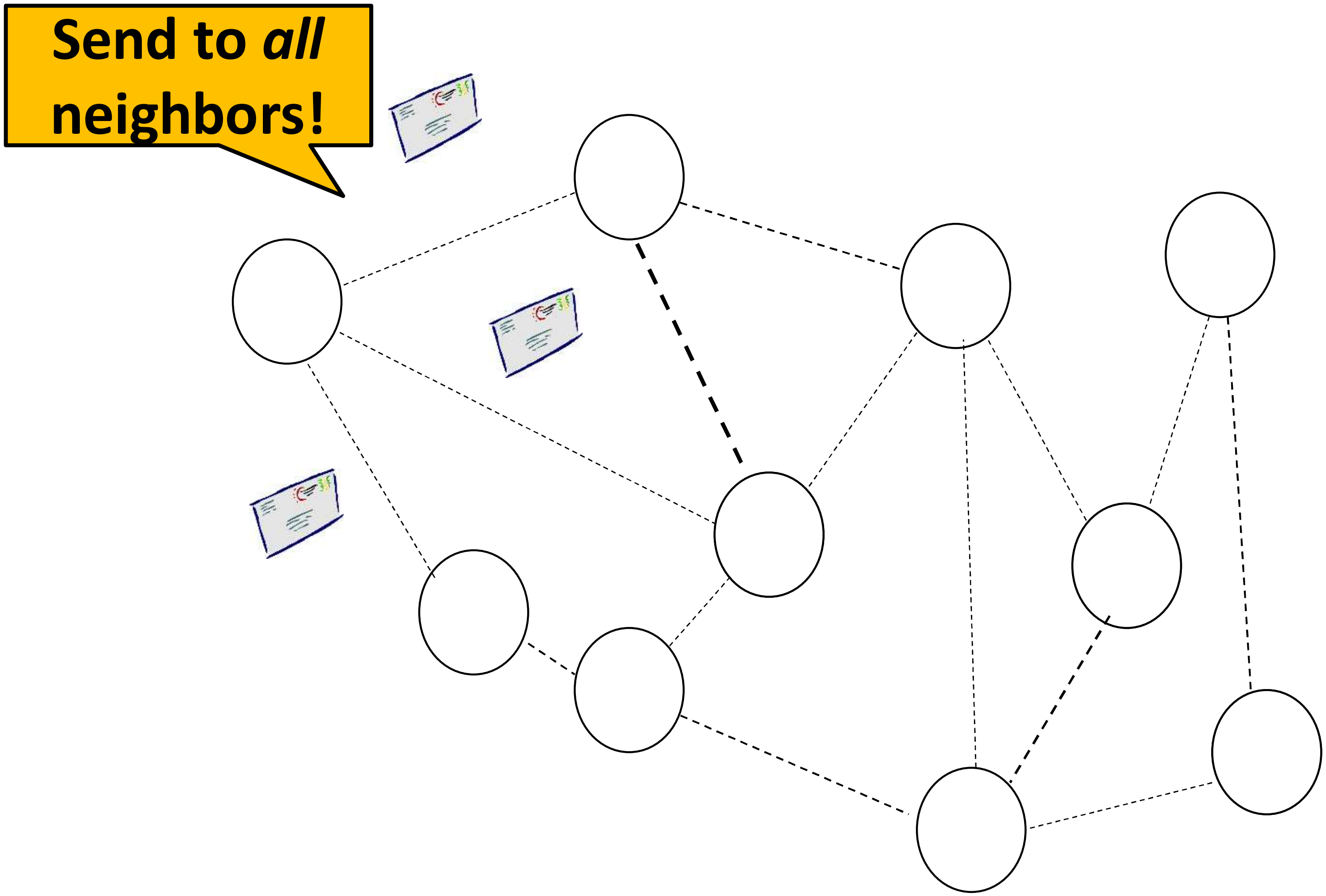
How to achieve this?

Compute a breadth first spanning tree! ☺ But how?

Idea: Compute BFS using Flooding!

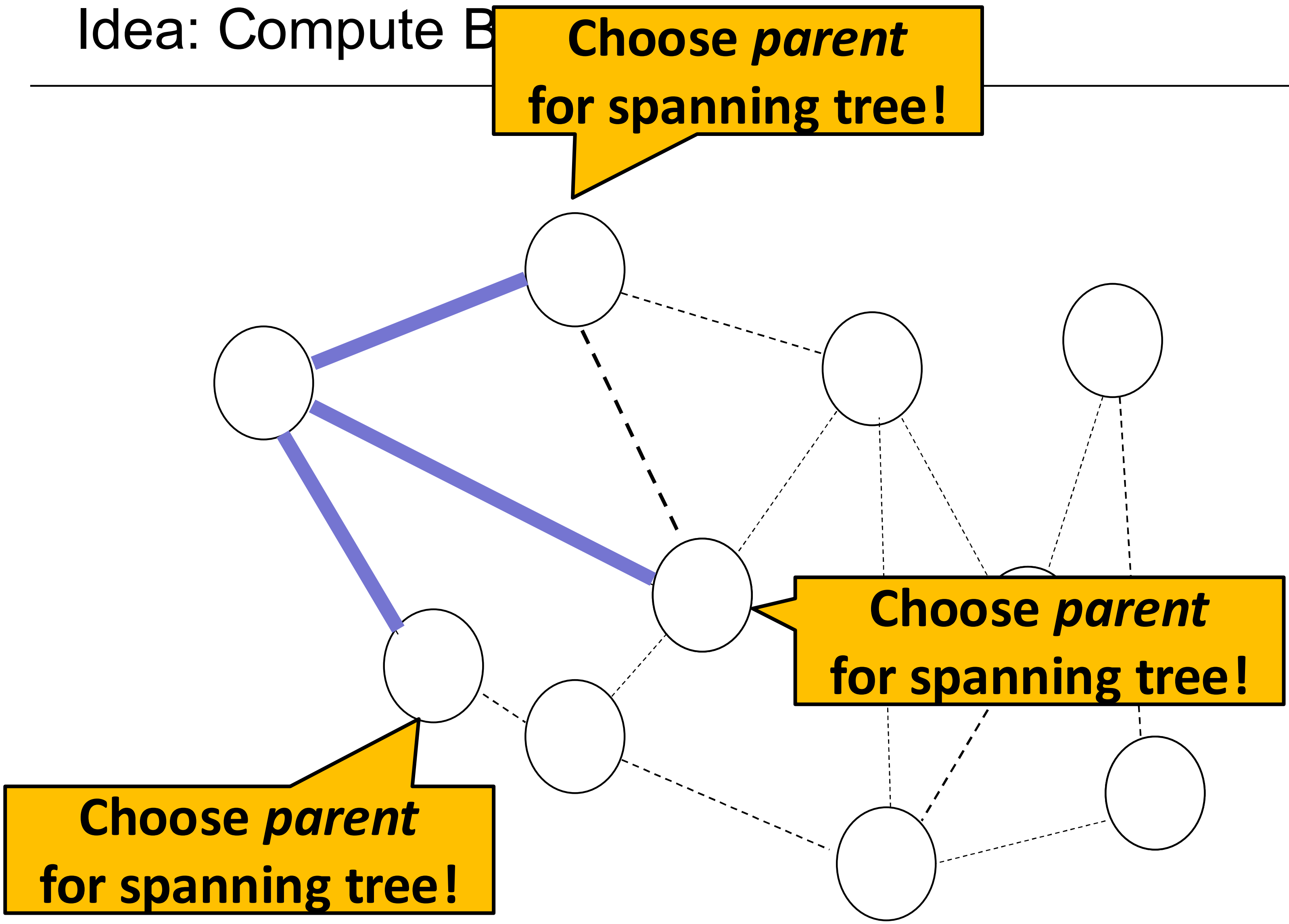


Idea: Compute BFS using Flooding!



Round 1

Idea: Compute B

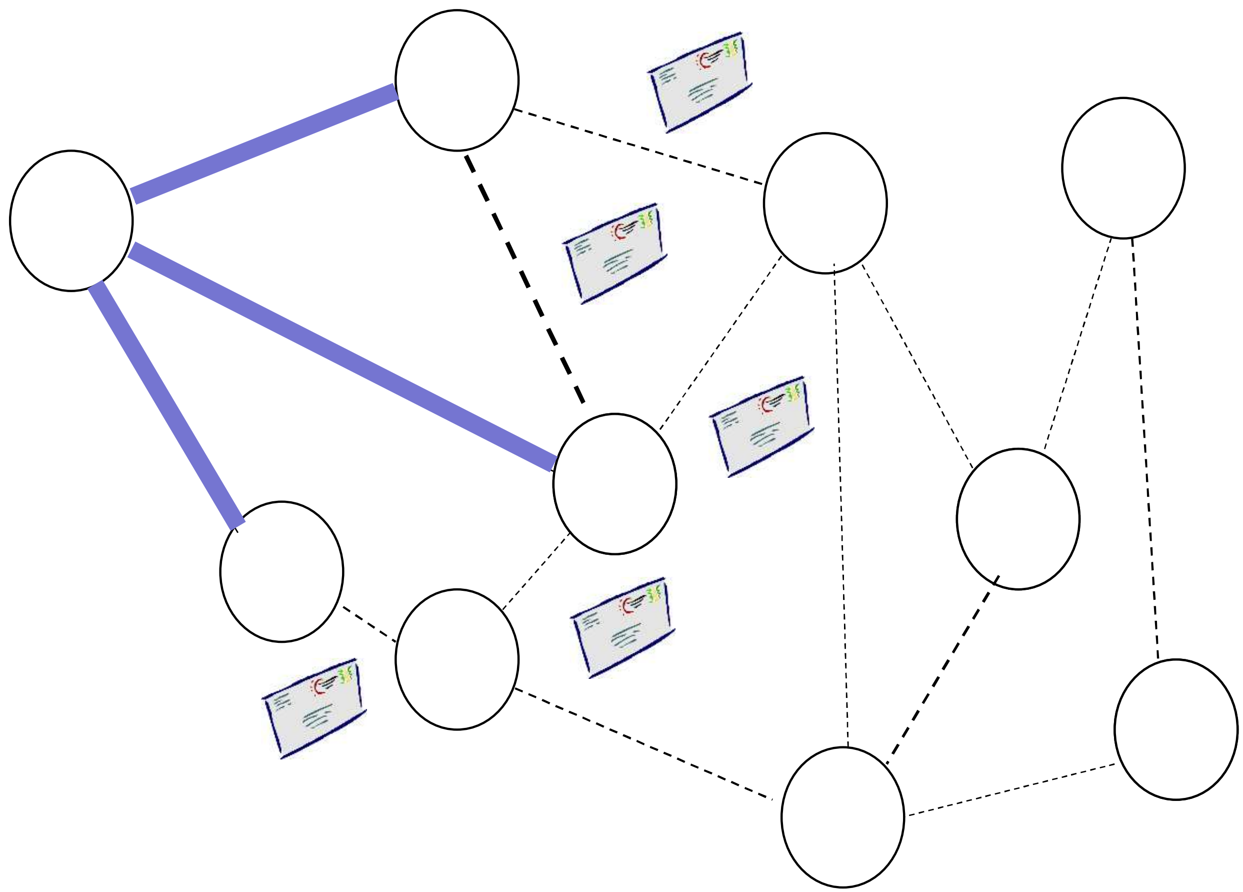


Round 1

Invariant: parent has shorter distance to root: loop-free!

Idea: Compute BFS using Flooding!

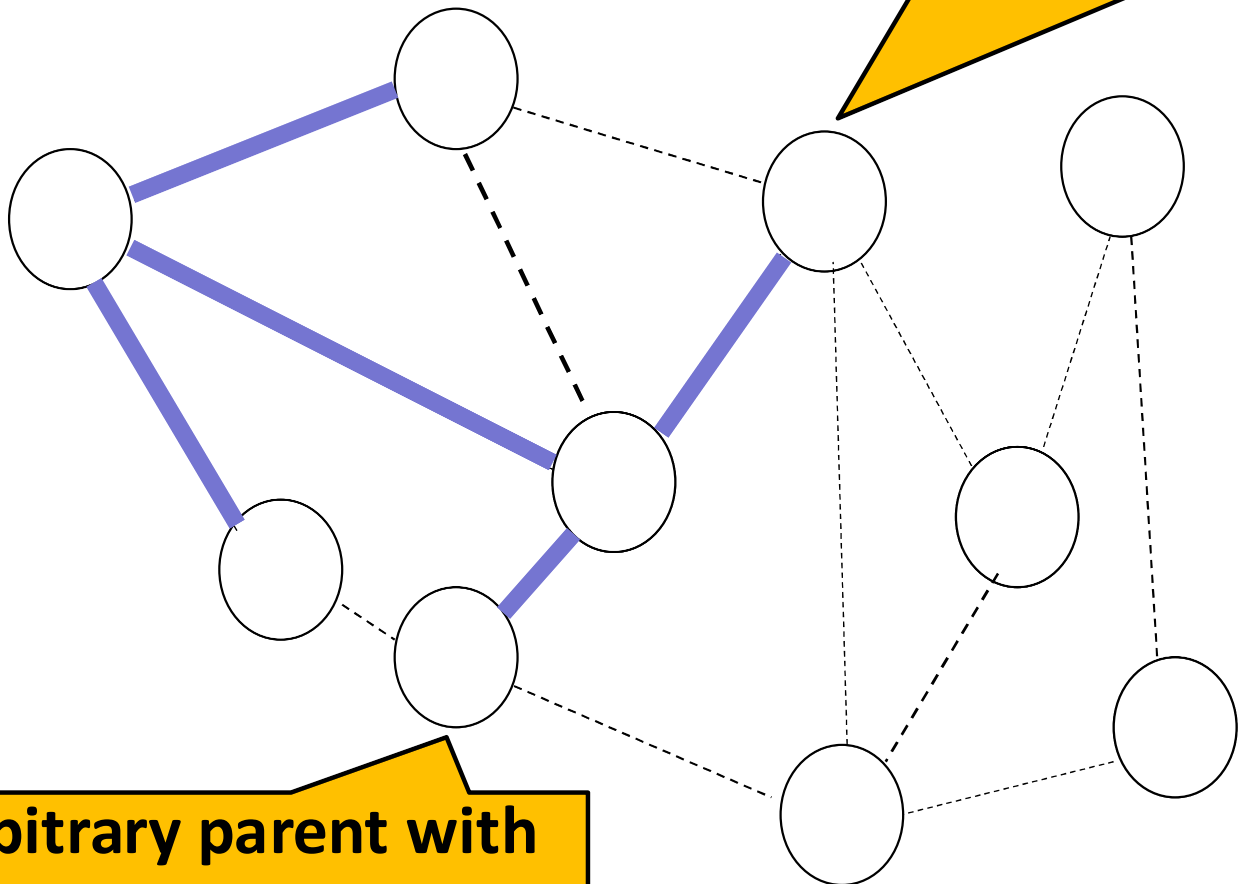
Send to *all* neighbors!



Round 2

Idea: Compute BFS using Flooding!

Choose a parent: if multiple arrive at same time, take *arbitrary*!

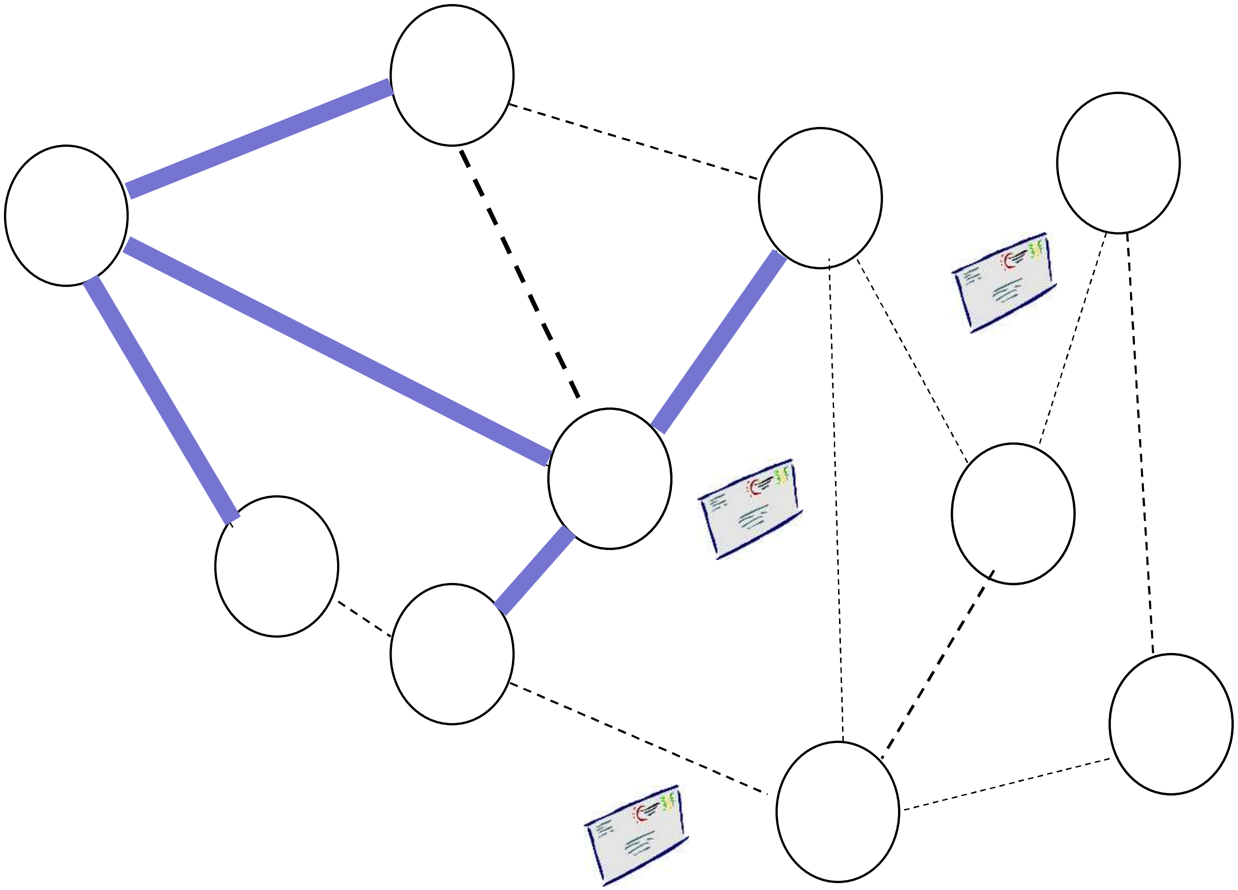


arbitrary parent with shorter distance!

Round 2

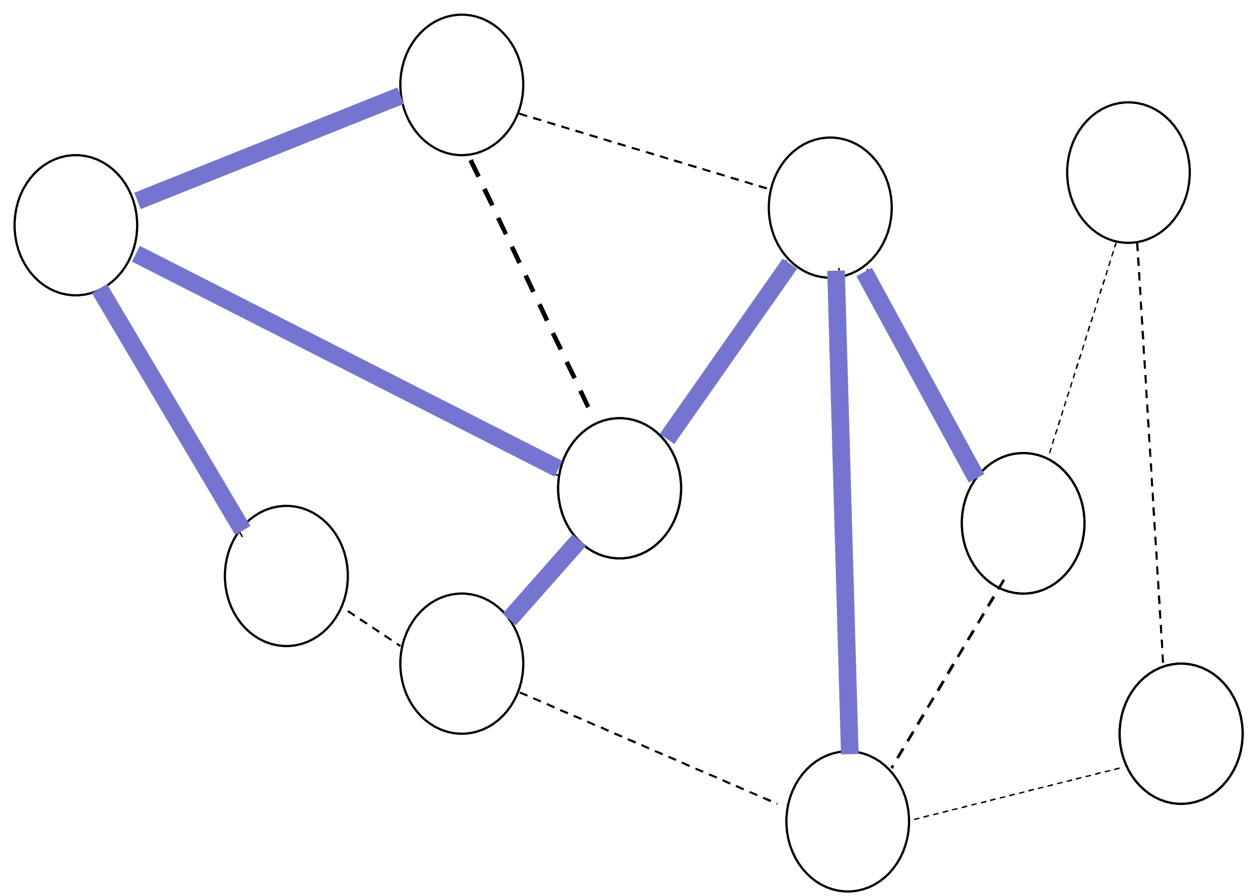
Invariant: parent has shorter distance to root: loop-free!

Idea: Compute BFS using Flooding!



Round 3

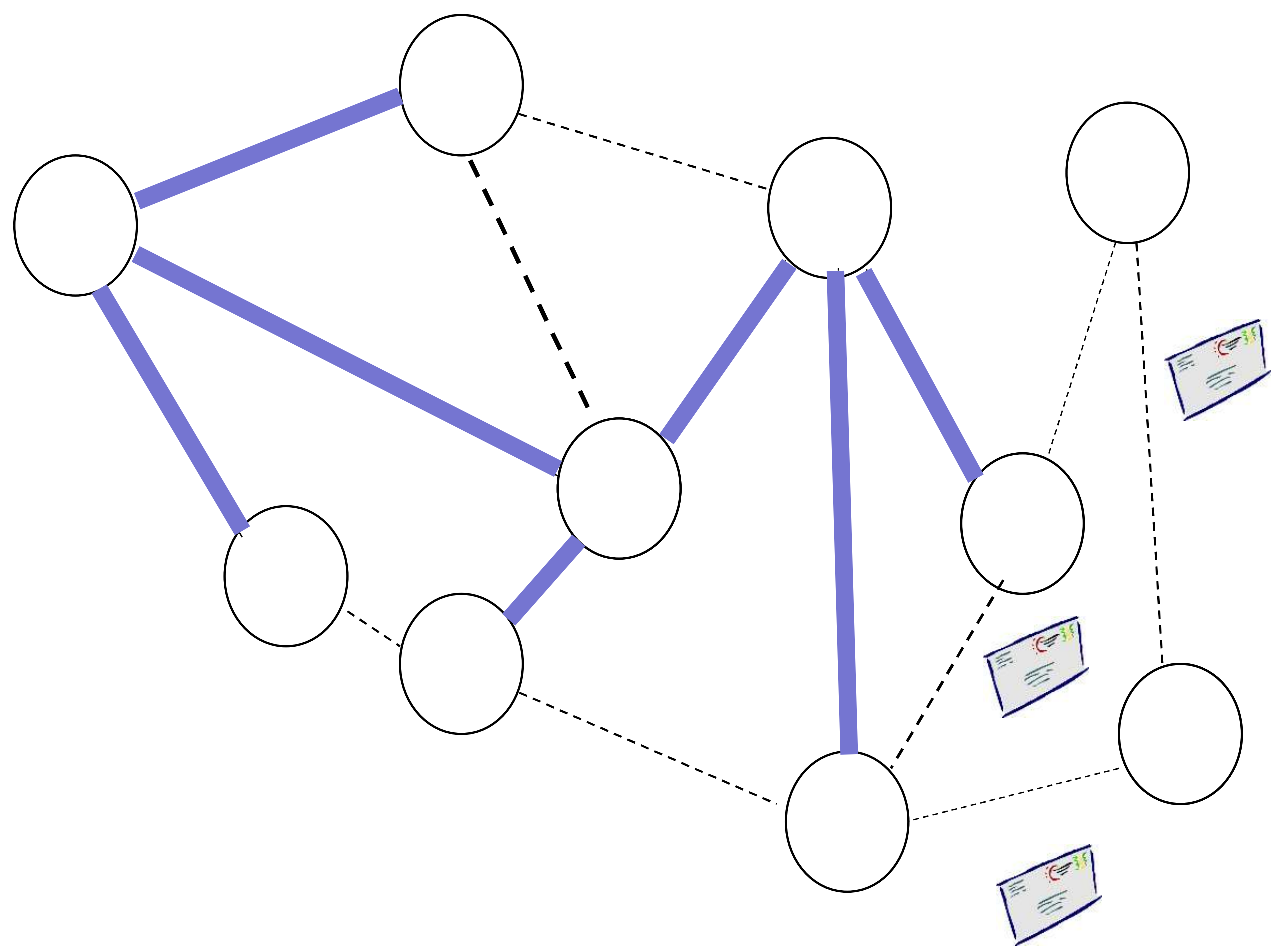
Idea: Compute BFS using Flooding!



Round 3

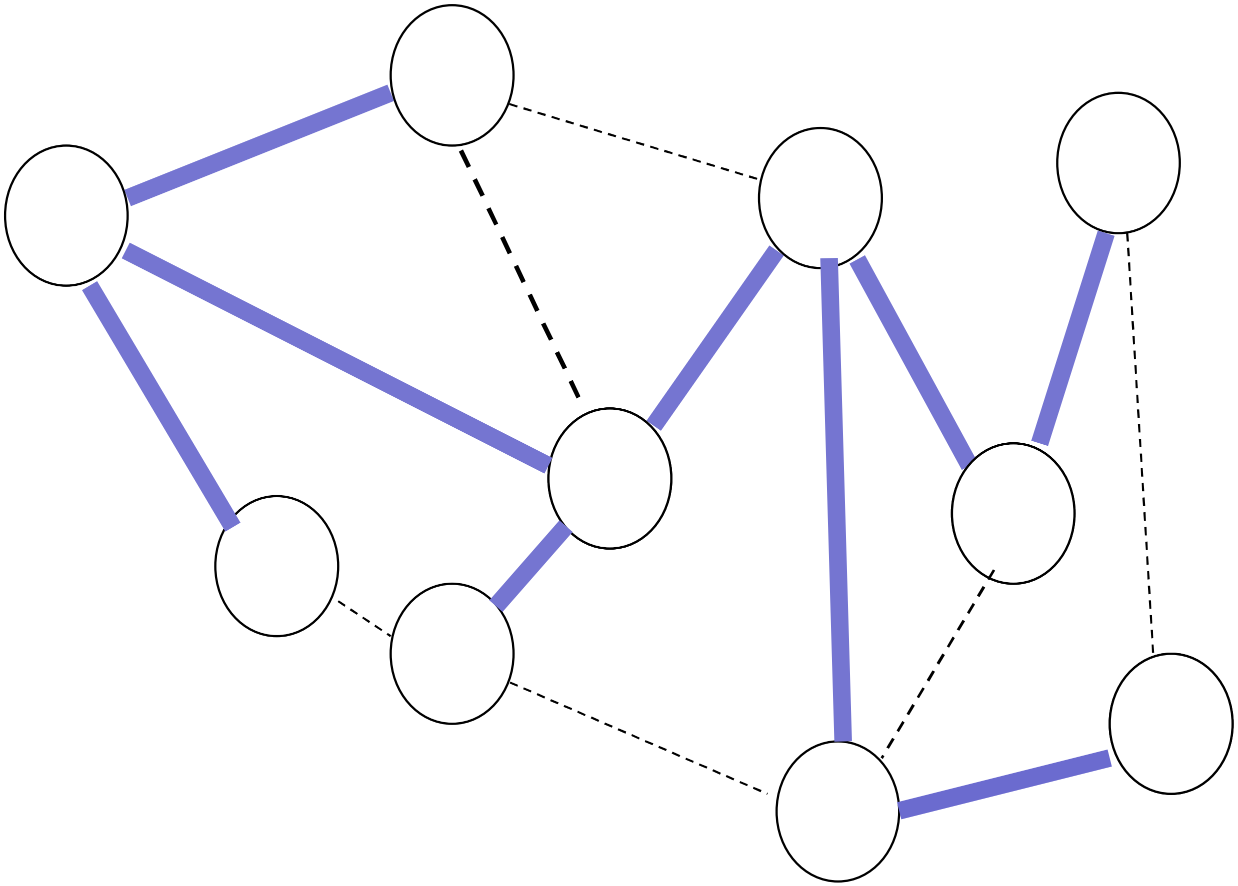
Invariant: parent has shorter distance to root: loop-free!

Idea: Compute BFS using Flooding!



Round 4

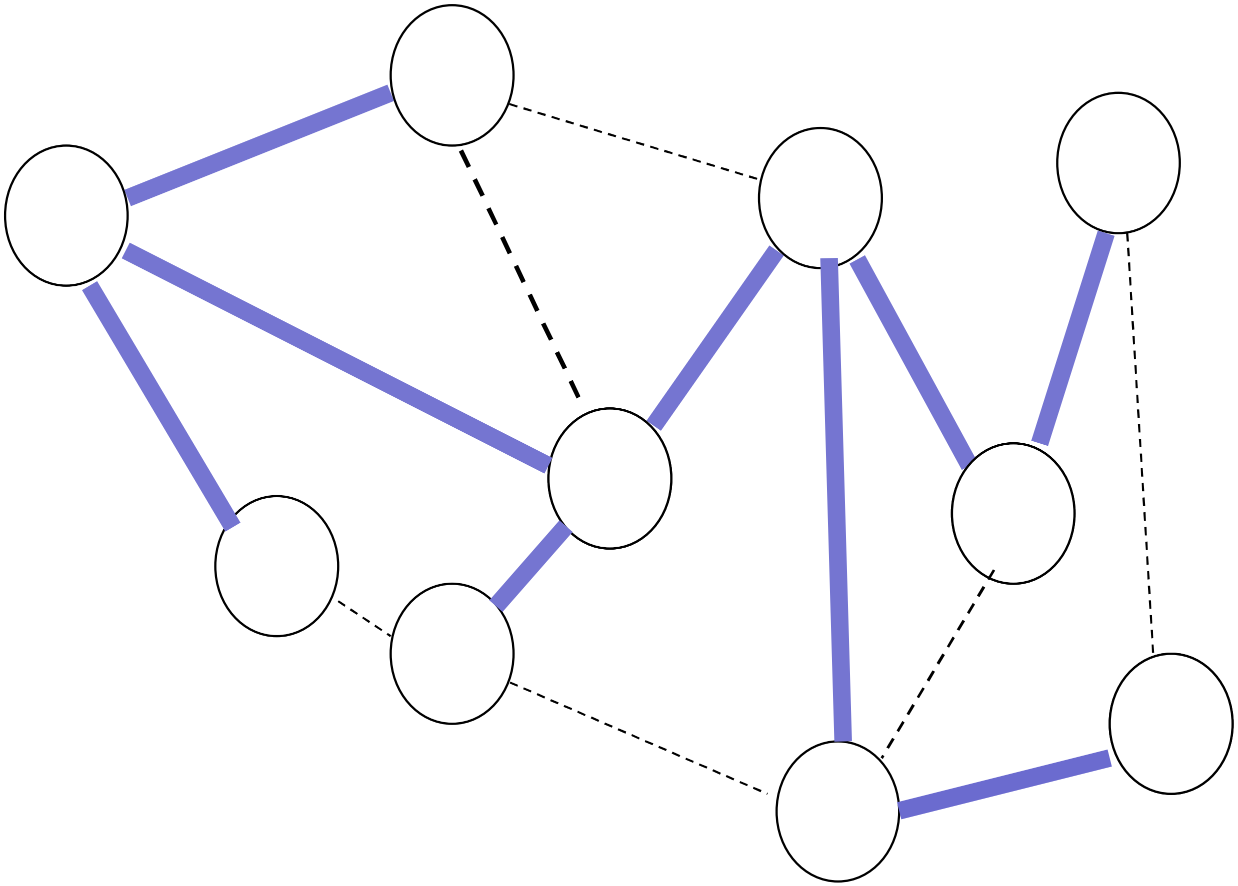
Idea: Compute BFS using Flooding!



BFS!

Invariant: parent has shorter distance to root: loop-free!

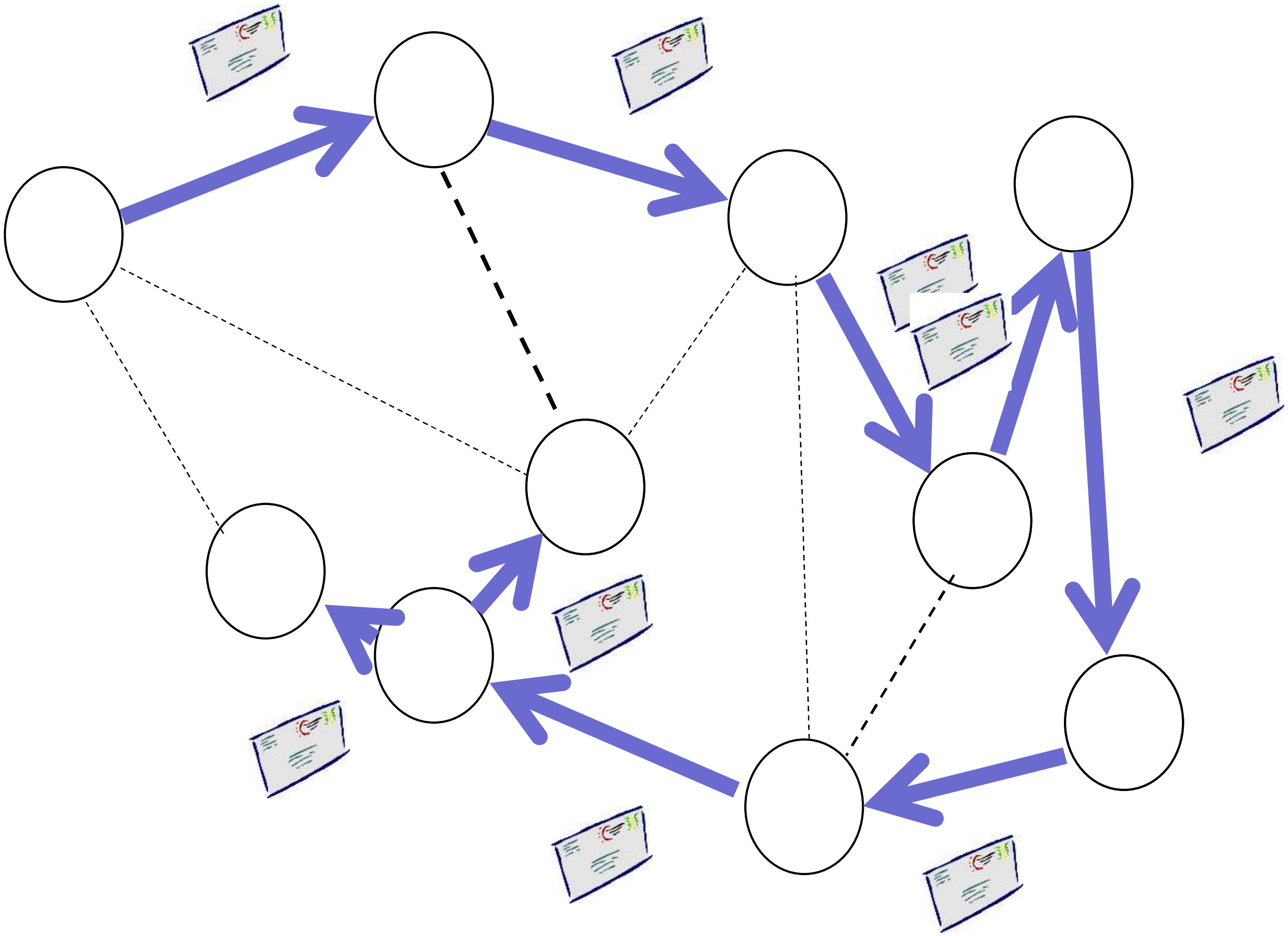
Idea: Compute BFS using Flooding!



BFS!

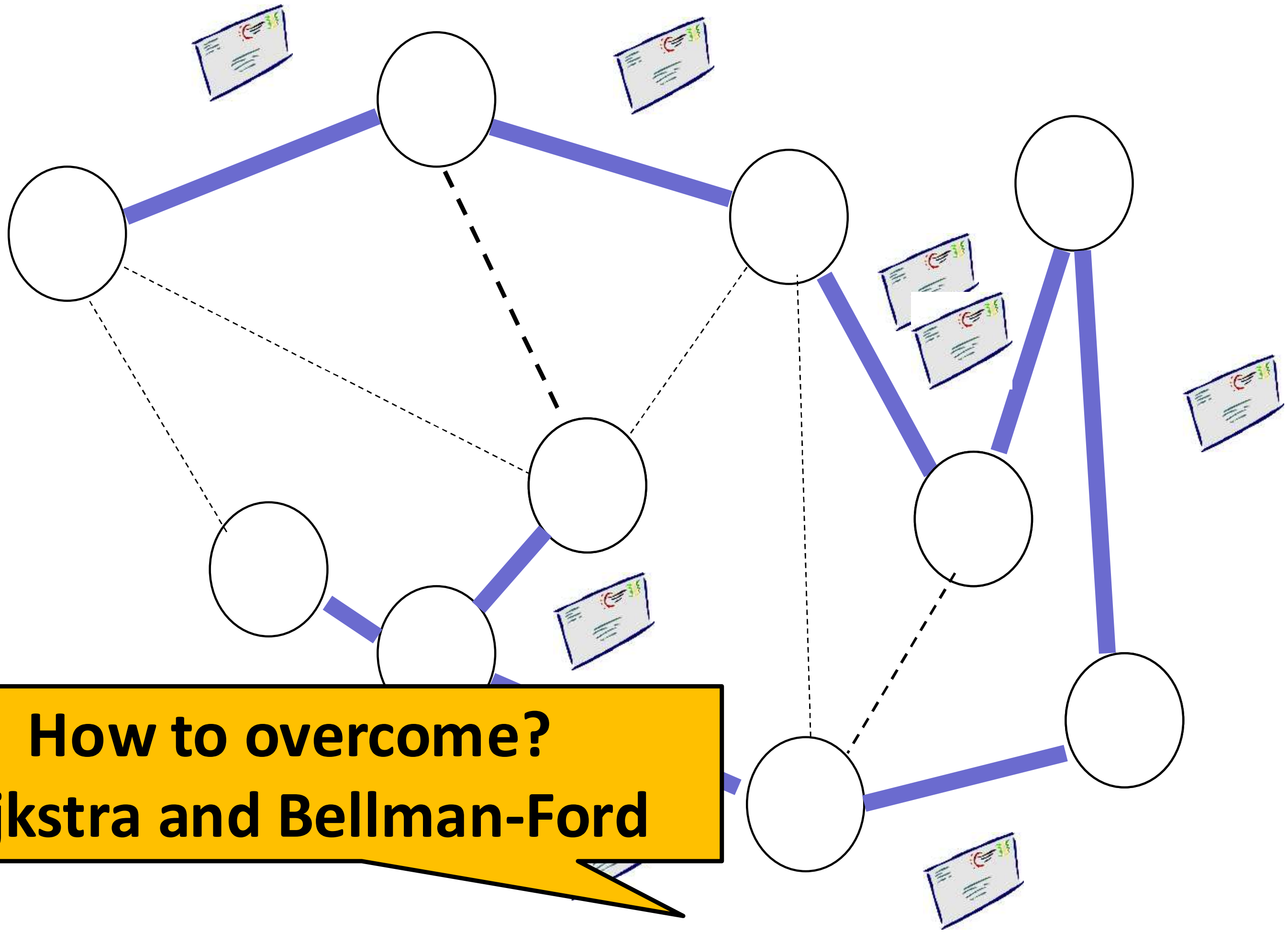
But careful! We assumed that messages propagate in synchronous manner! What if not?

Bad example



Careful: in asynchronous environment, should not make first successful sender my parent!

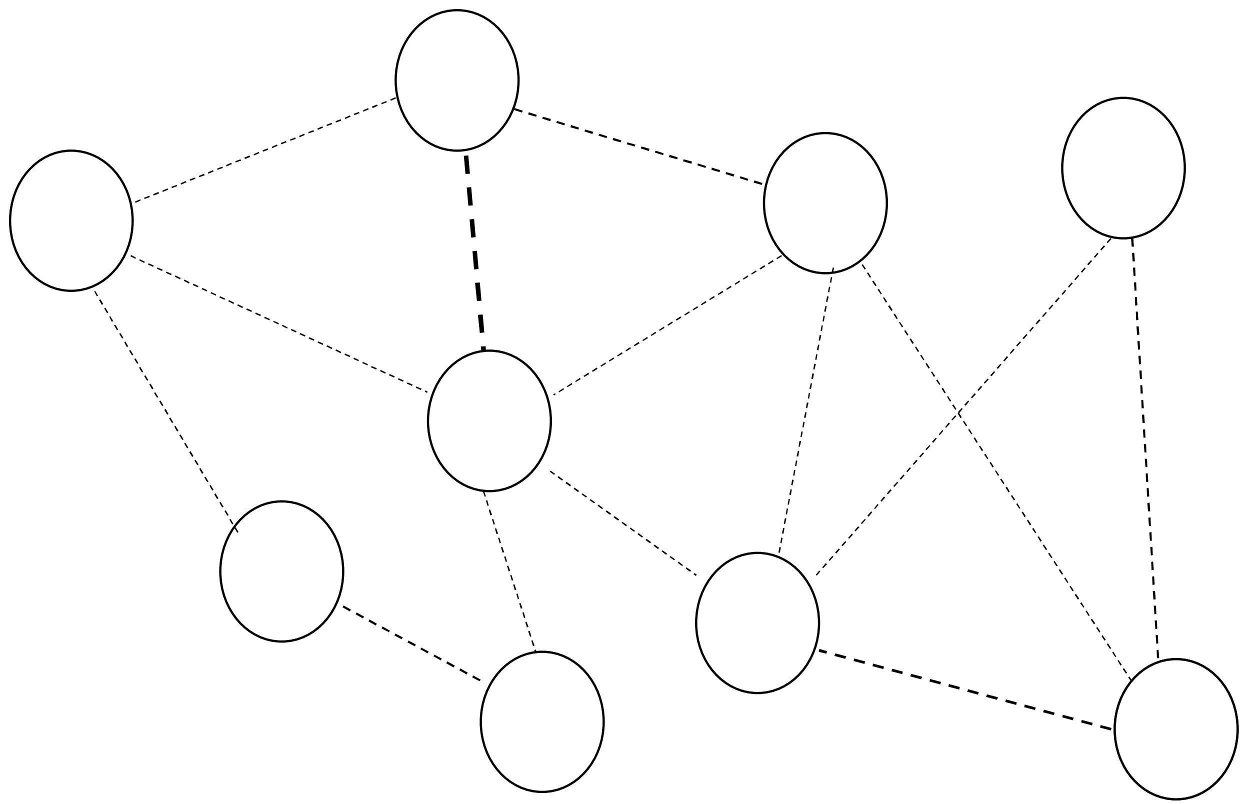
Bad example



How to overcome?
Dijkstra and Bellman-Ford

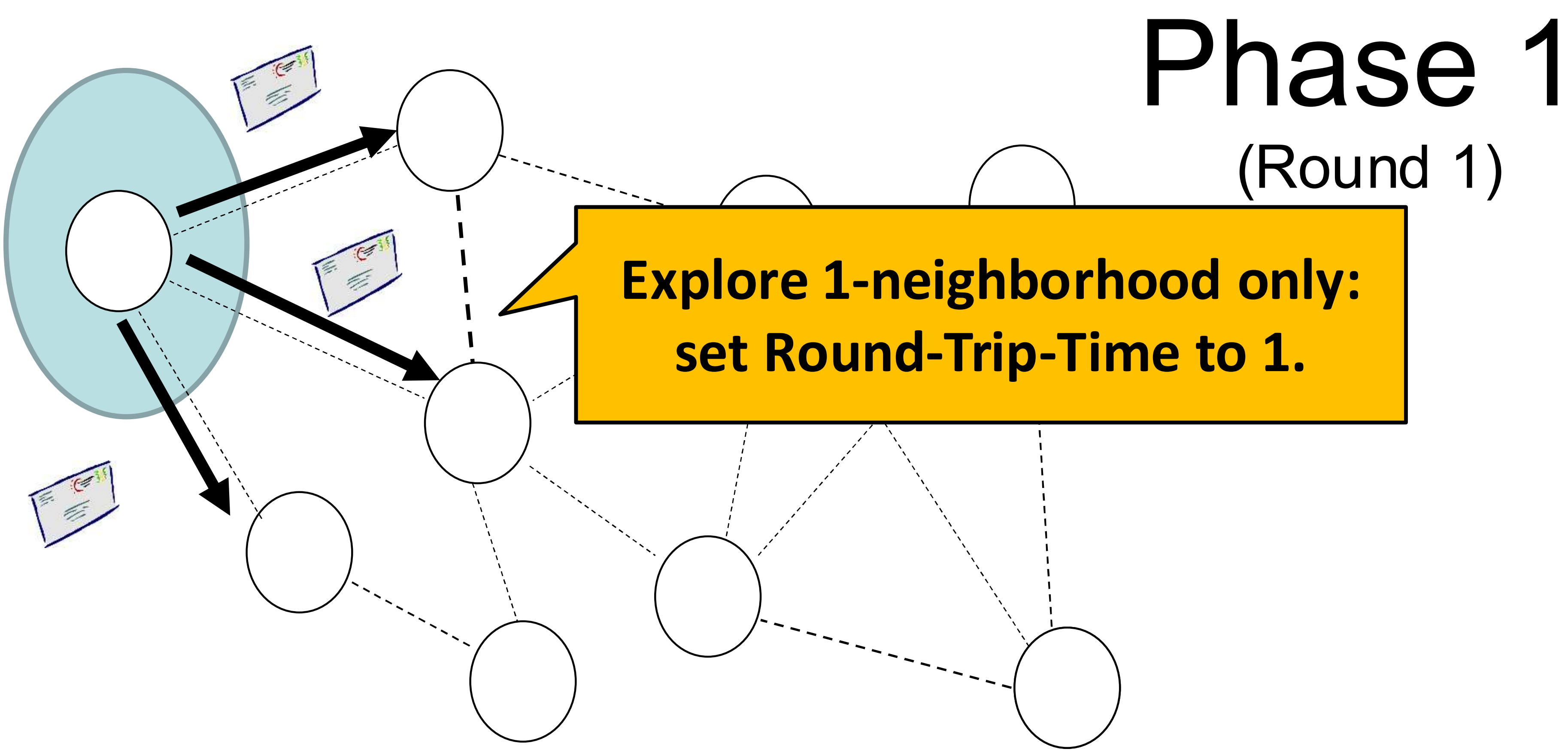
Careful: in asynchronous environment, should not make first successful sender my parent!

Distributed BFS: Dijkstra Flavor



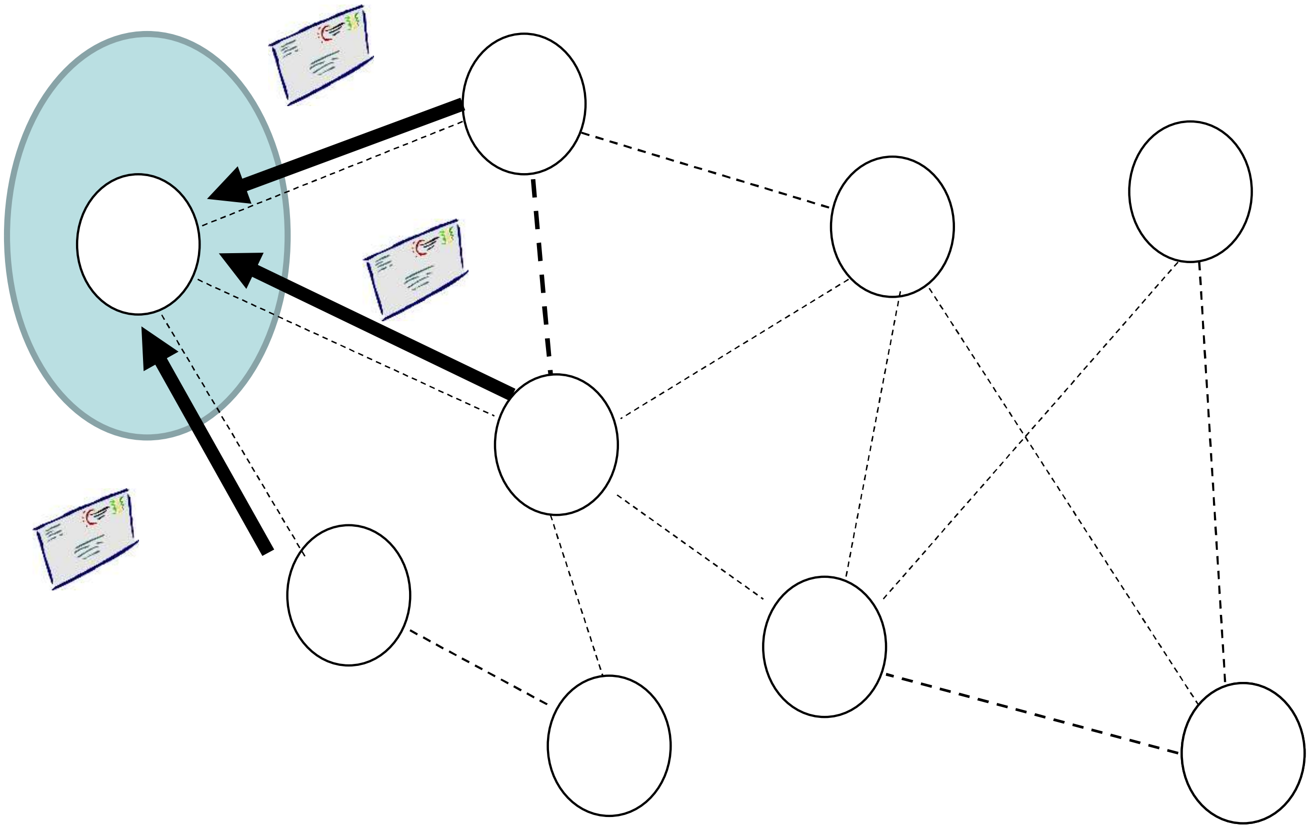
Idea: overcome asynchronous problem by proceeding in phases!

Distributed BFS: Dijkstra Flavor



Idea: overcome asynchronous problem by proceeding in phases!

Distributed BFS: Dijkstra Flavor



Phase 1

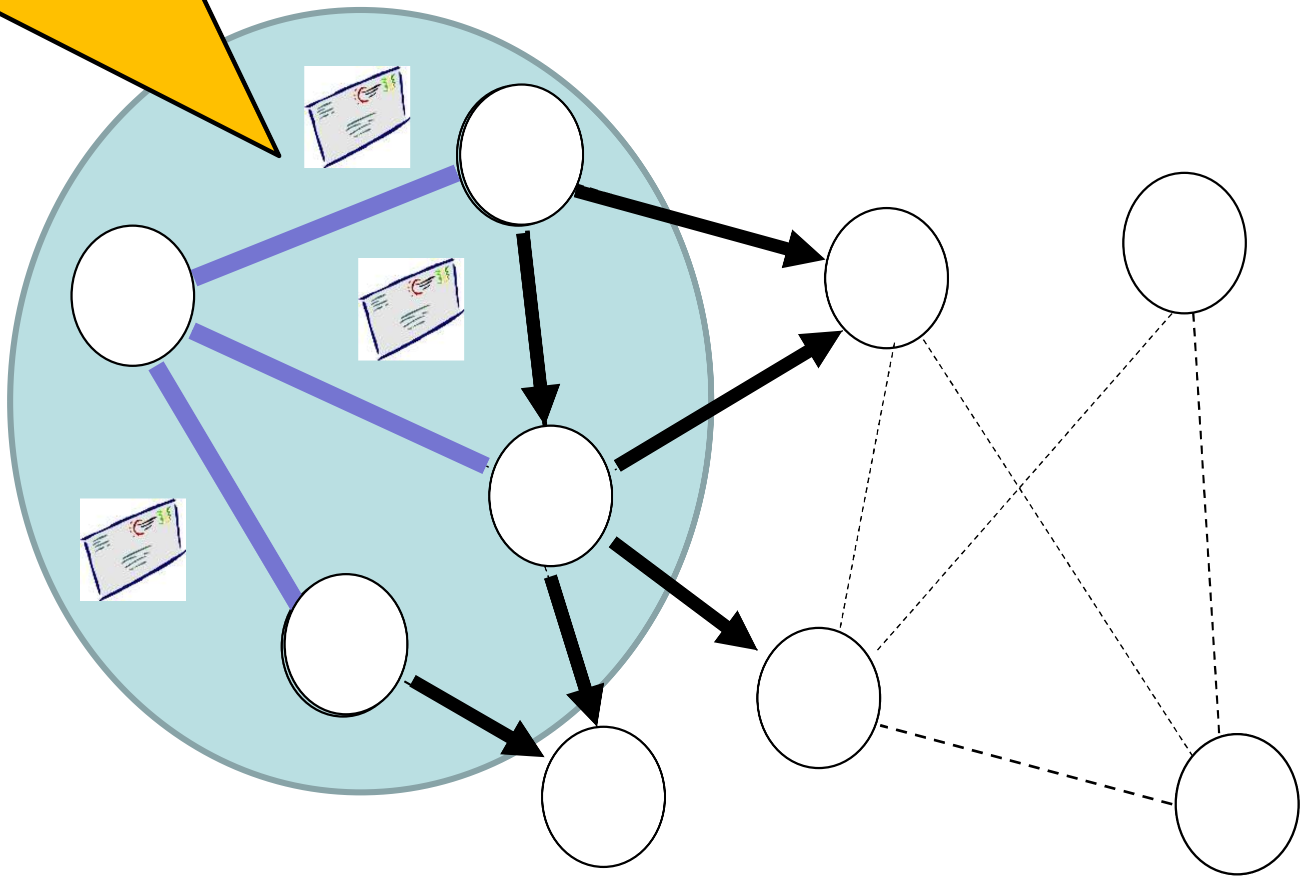
(Round 2)

Idea: overcome asynchronous problem by proceeding in phases!

Start Phase 2! (Propagate along existing spanning tree!)

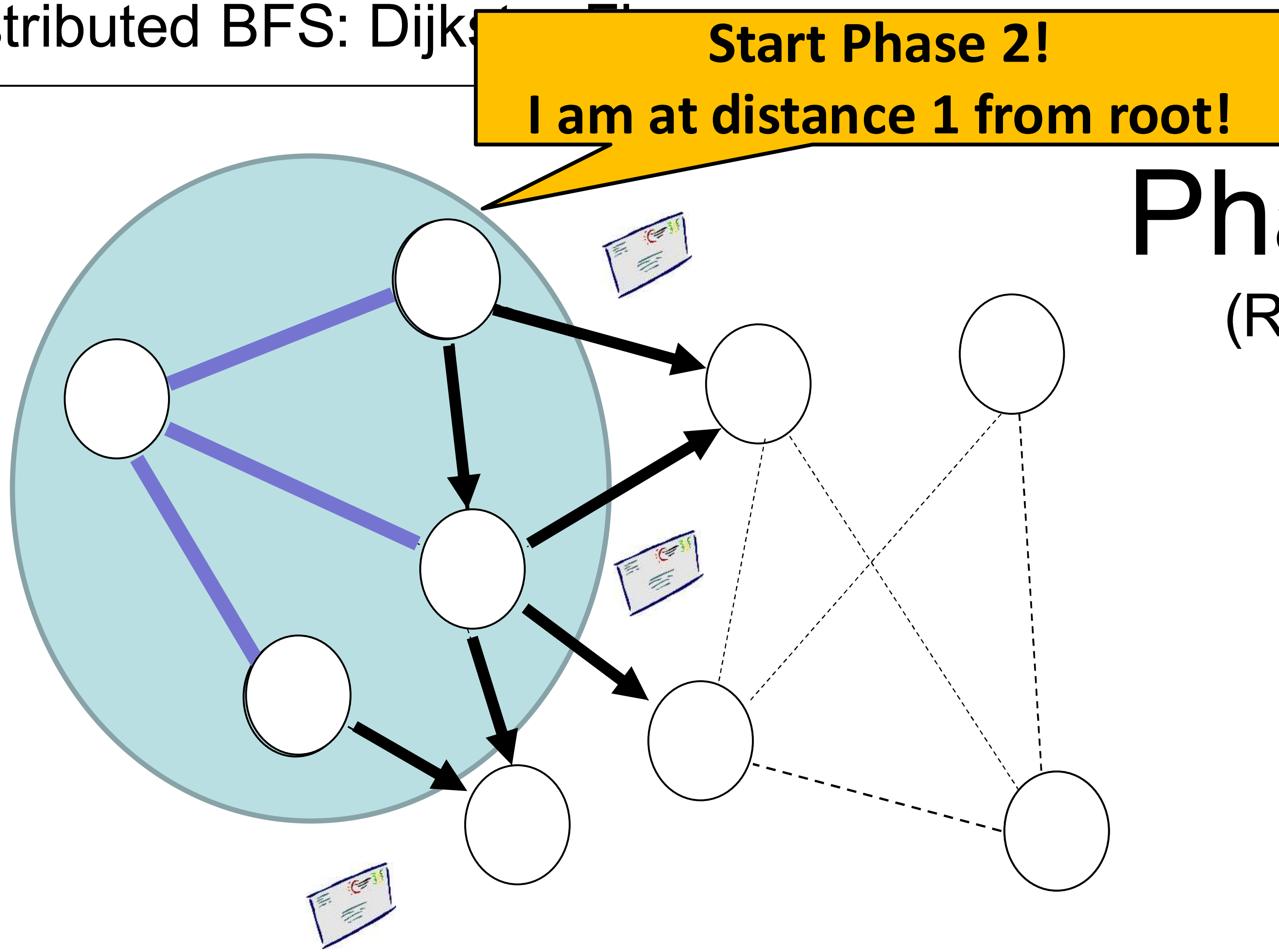
Phase 2

(Round 1)



Idea: overcome asynchronous problem by proceeding in phases!

Distributed BFS: Dijks[...]

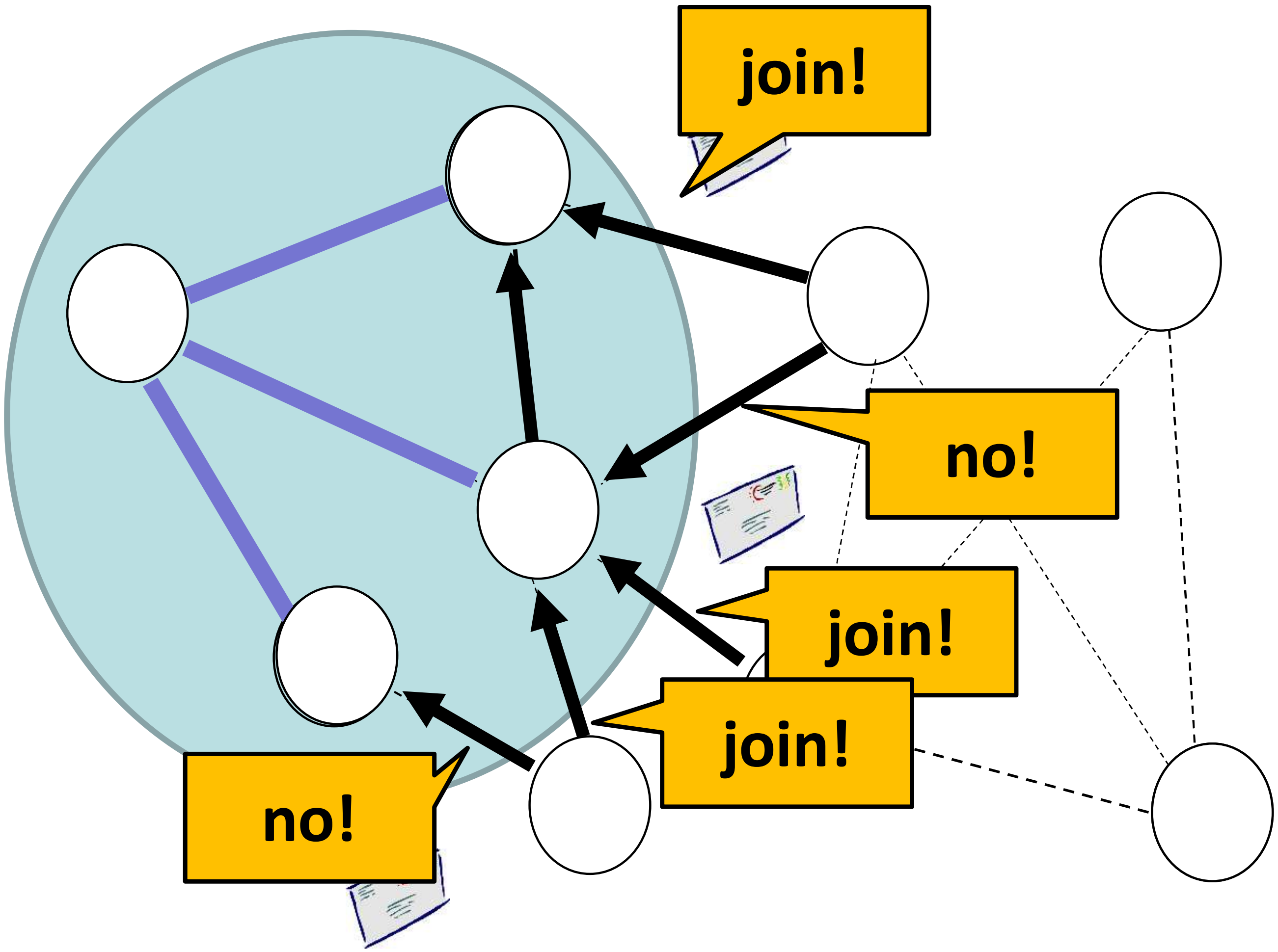


Phase 2

(Round 2)

Idea: overcome asynchronous problem by proceeding in phases!

Distributed BFS: Dijkstra Flavor



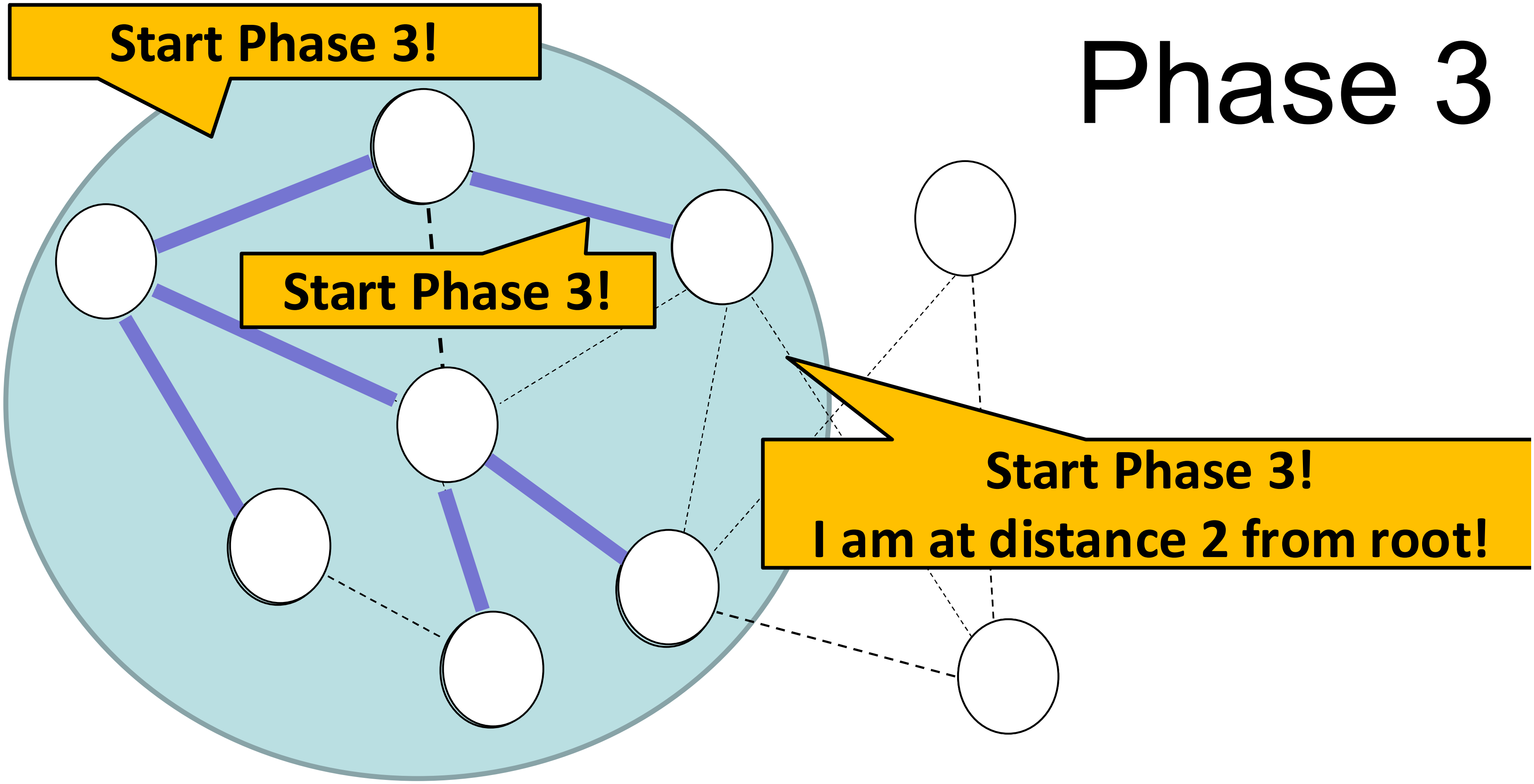
Phase 2

(Round 3)

Idea: overcome asynchronous problem by proceeding in phases!

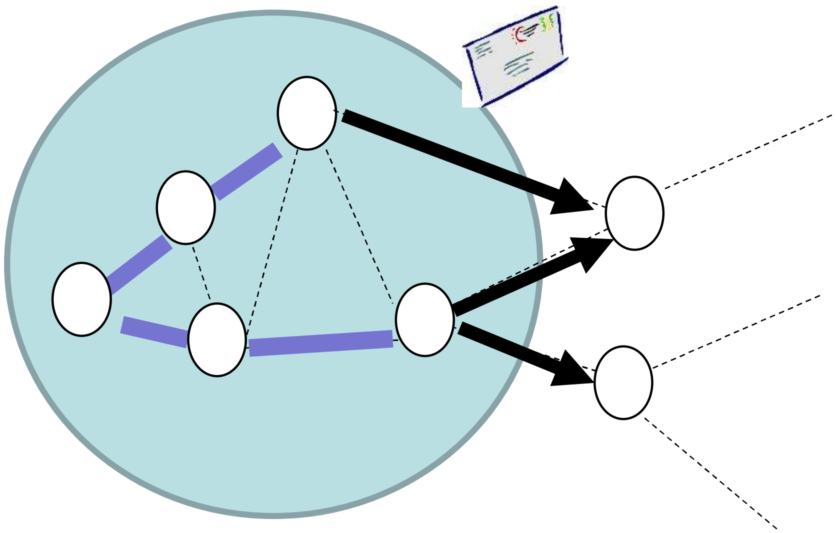
Choose parent with smaller distance!

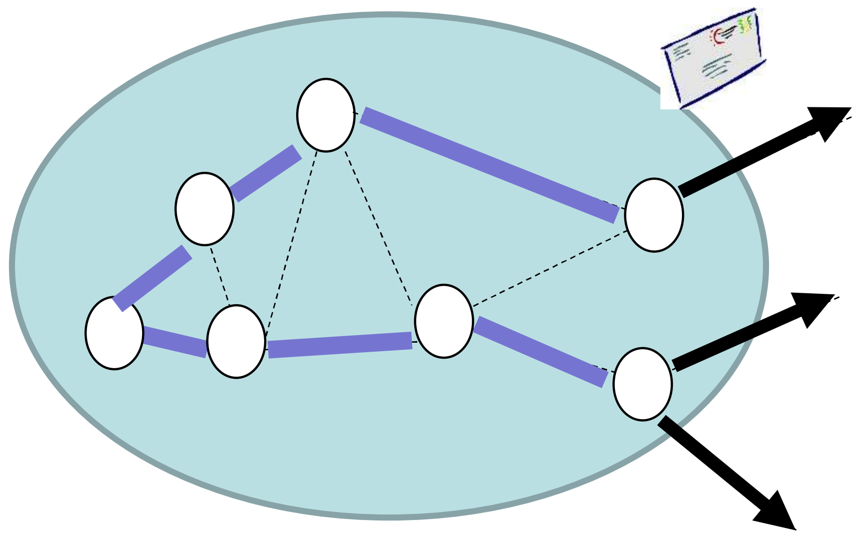
Distributed BFS: Dijkstra Flavor



Idea: overcome asynchronous problem by proceeding in phases!

General Scheme

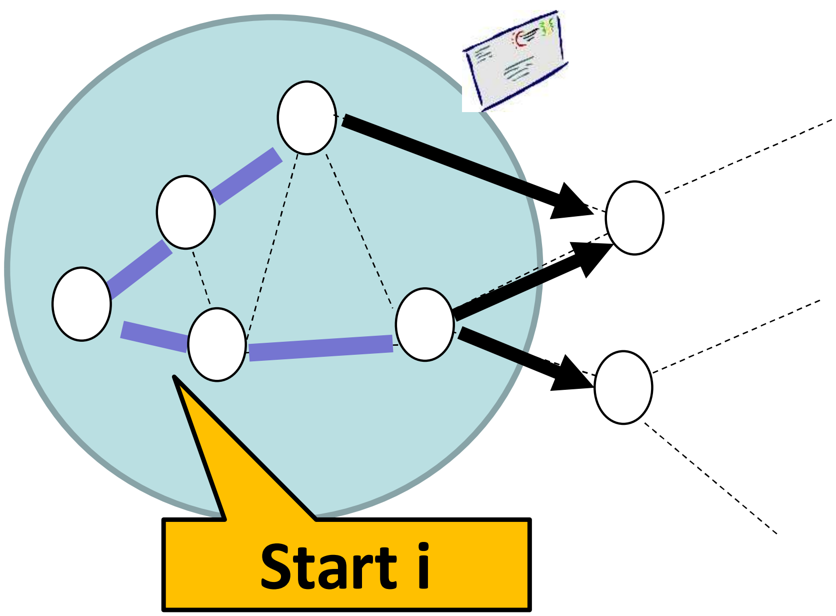


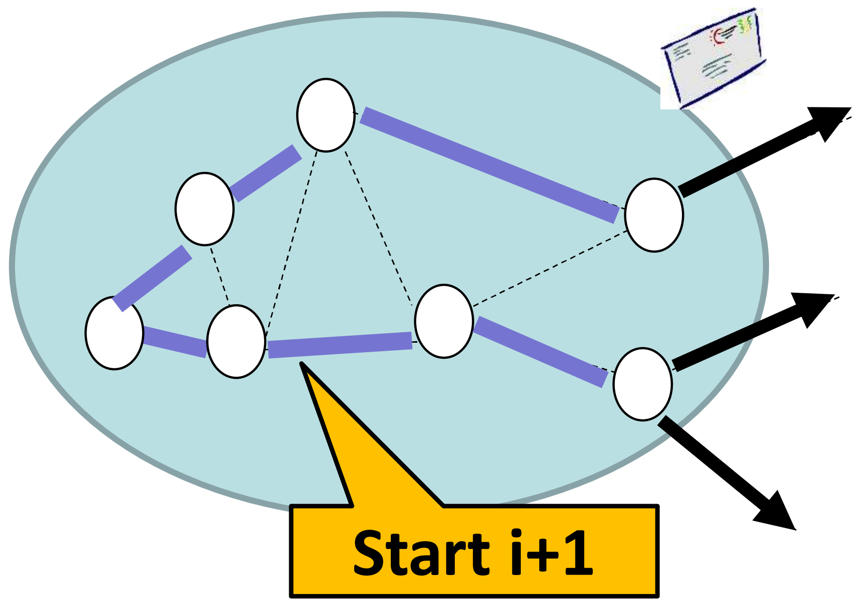


Phase i

Phase i+1

General Scheme

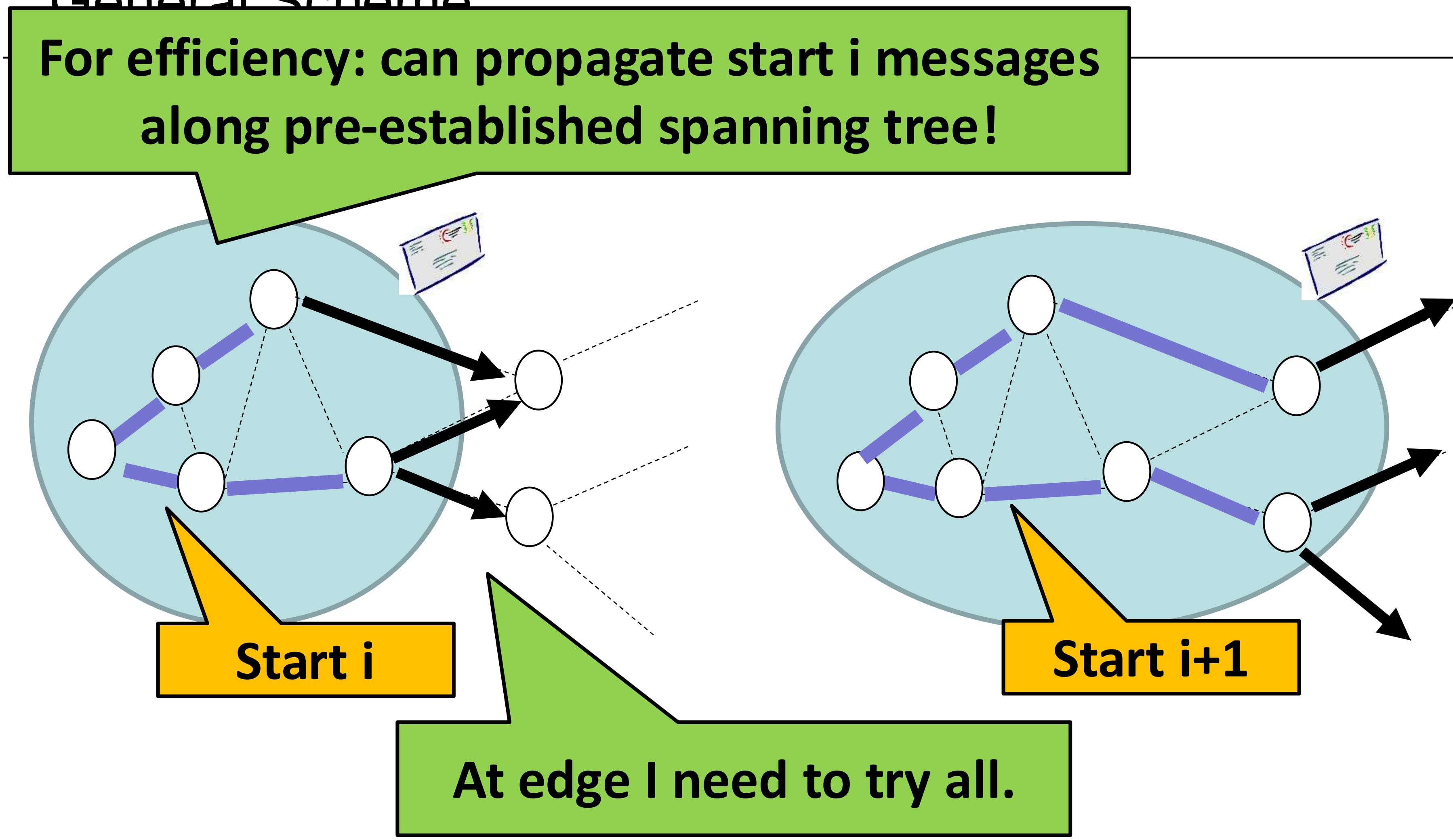




Phase i

Phase i+1

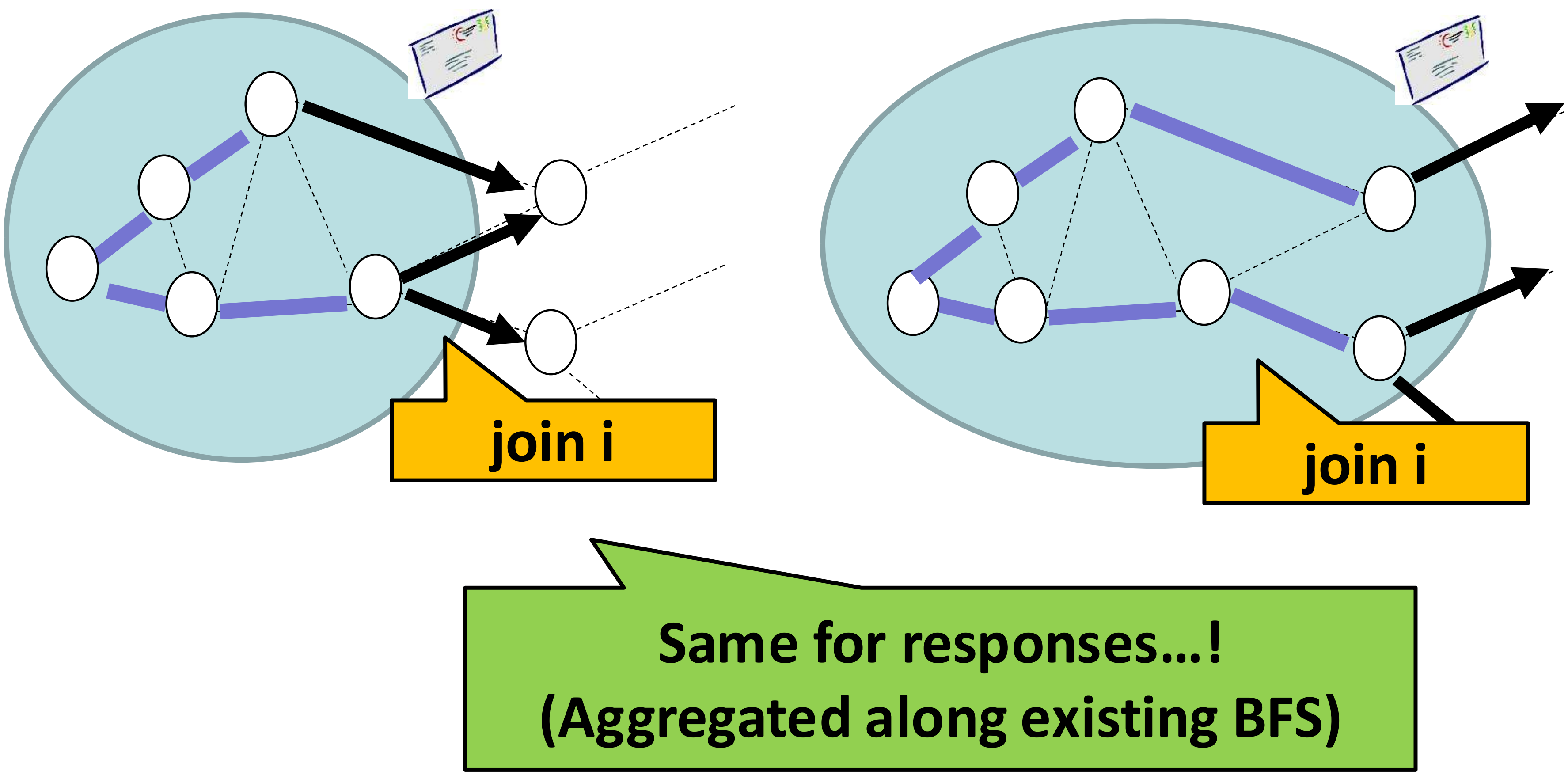
General Scheme



Phase i

Phase i+1

Distributed BFS: Dijkstra Flavor

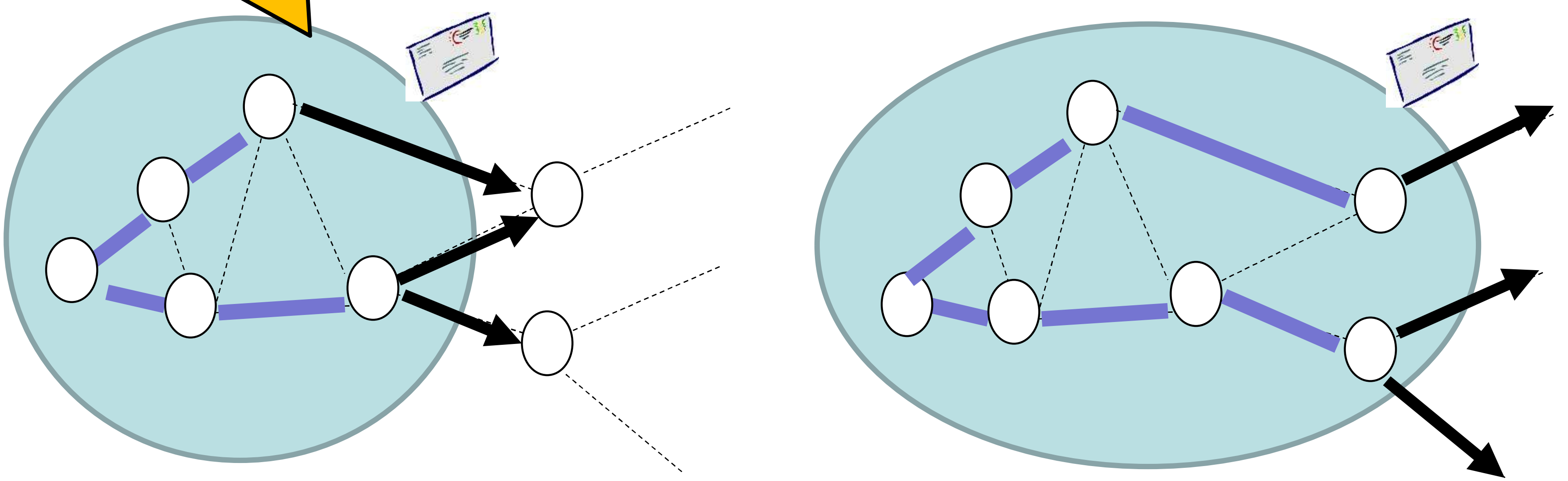


Phase i

Phase i+1

Time Complexity?

vor

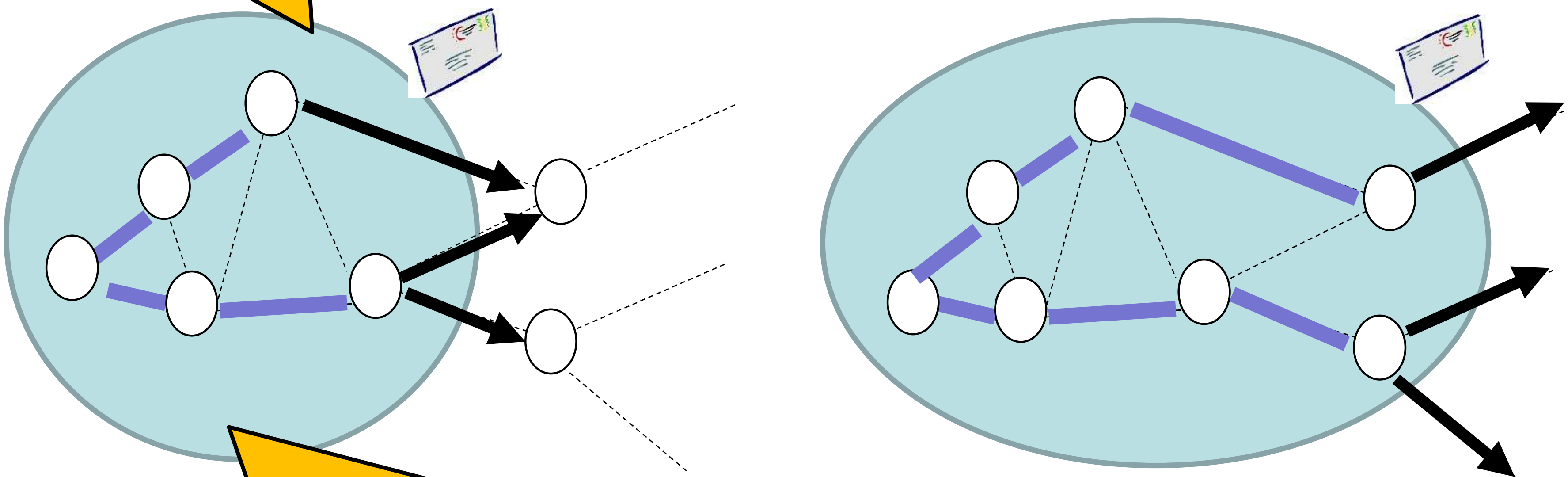


Phase i

Phase i+1

vor

Time Complexity?



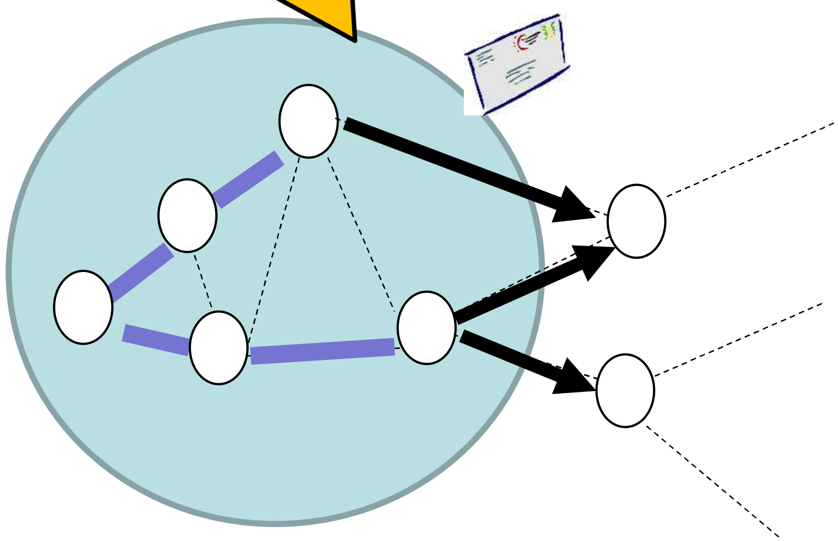
O(D) phases, take time O(D): $O(D^2)$ where D is the radius from the root.

Phase i

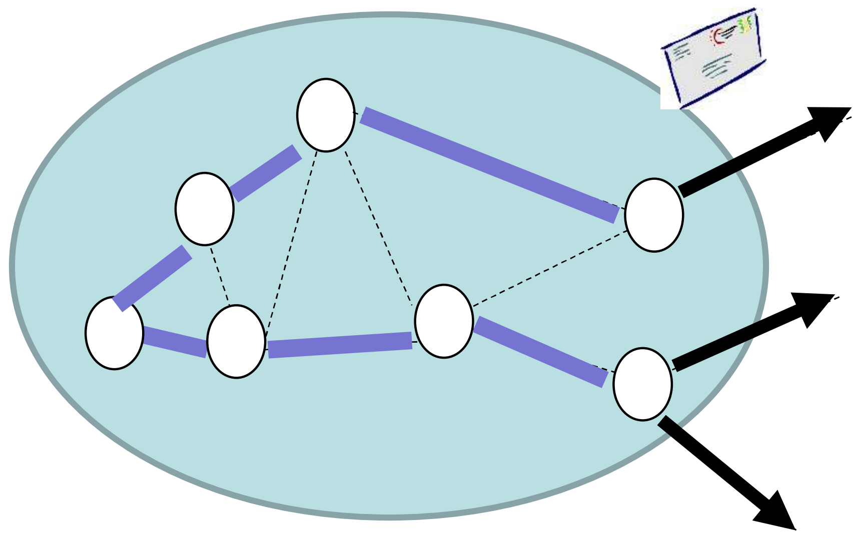
Phase i+1

Message Complexity?

vor



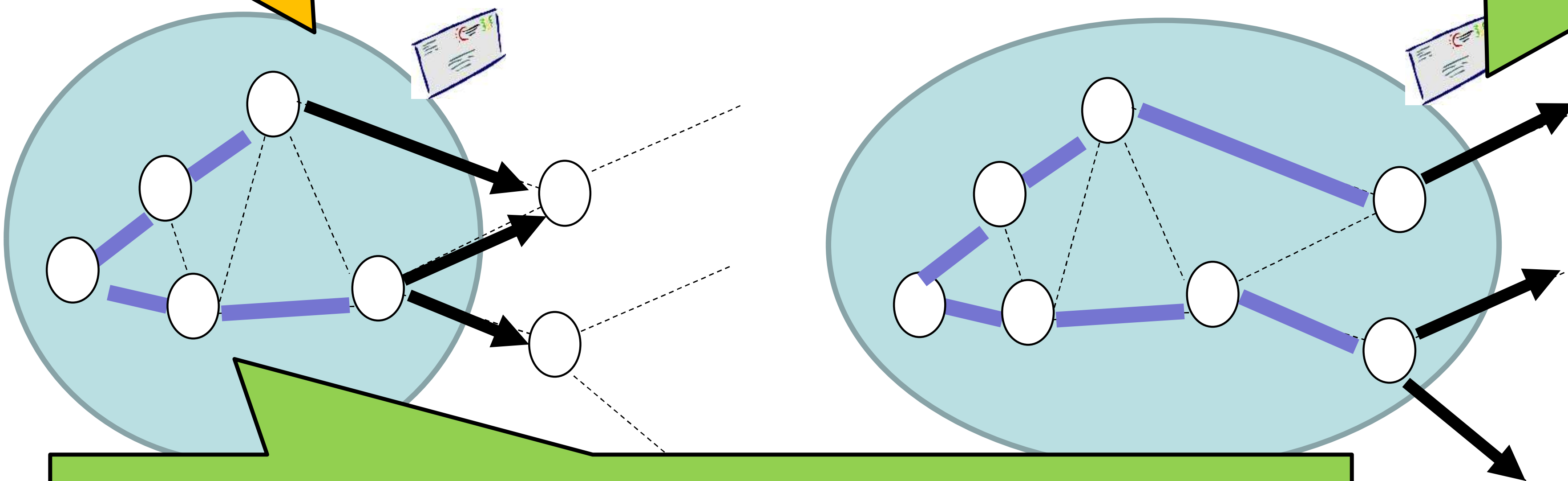
Phase i



Phase i+1

Message Complexity?

Plus: test each edge once: join, ACK/NAK at edge: total O(m).



„start“ and „join“ propagation inside spanning tree: O(n) per phase: O(nD) in total.

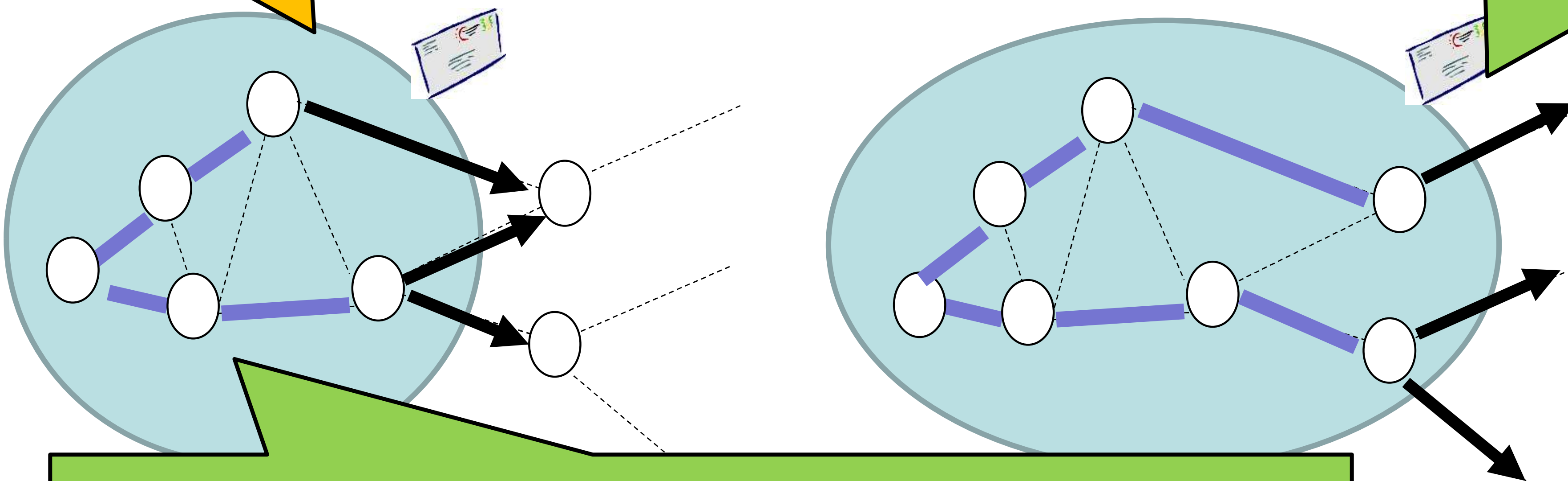
Phase i

Phase i+1

Message Complexity?

vor

Plus: test each edge once: join, ACK/NAK at edge: total O(m).



„start“ and „join“ propagation inside spanning tree: O(n) per phase: O(nD) in total.

Pha **O(nD+m)** e i+1

Distributed BFS: Dijkstra Flavor

Dijkstra: find next closest node („on border“) to the root

Dijkstra Style

Divide execution into *phases*. In phase p, nodes with distance p to the root are detected. Let T_p be the tree of phase p. T_1 is the root plus all direct neighbors.

Repeat (until no new nodes discovered):

1. Root starts phase p by broadcasting „**start p**“ within T_p
2. A leaf u of T_p (= node discovered only in last phase) sends „**join p+1**“ to all quiet neighbors v (u has not talked to v yet)
3. Node v hearing „join“ for first time sends back „**ACK**“: it becomes leave of tree T_{p+1}; otherwise v replied „**NACK**“ (needed since async!)
4. The leaves of T_p collect all answers and start Echo Algorithm to the root
5. Root initates next phase

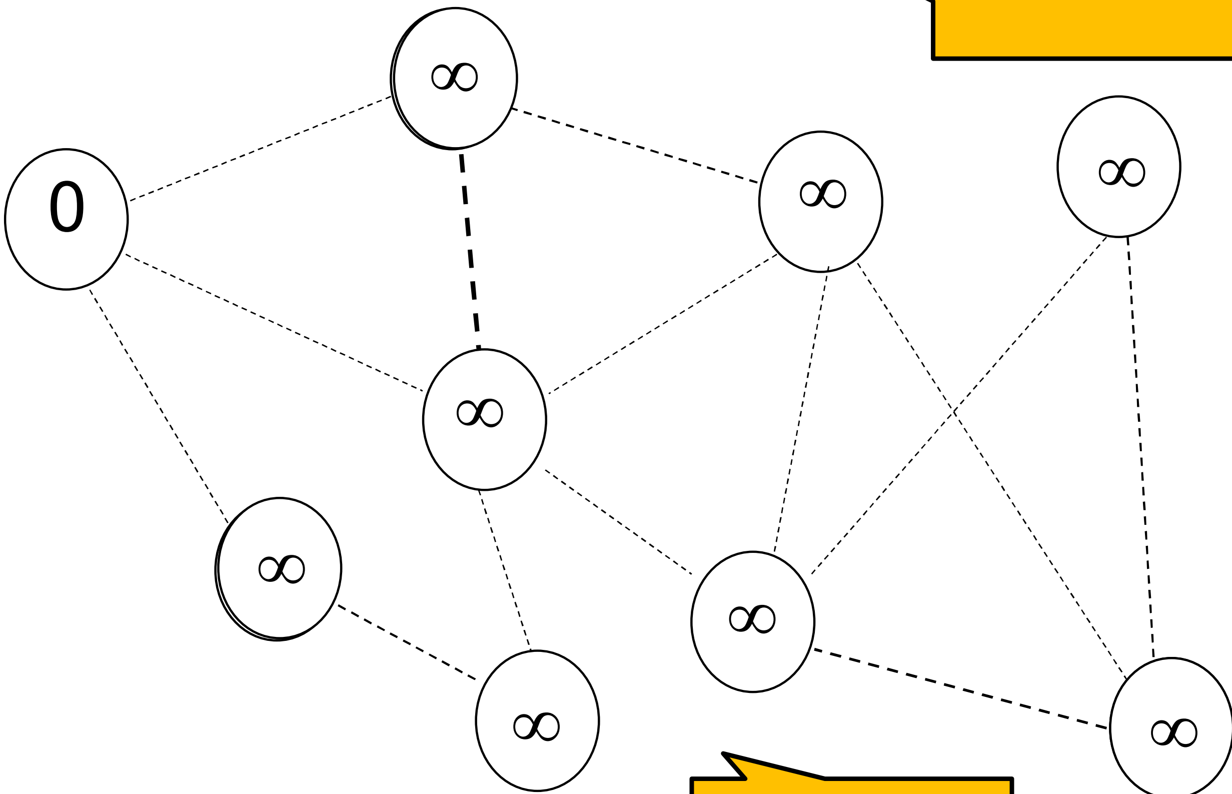
Distributed BFS: Bellman-Ford Flavor

Distributed BFS: Bellman-Ford Flav

Idea: Don't go through these time-consuming phases but blast out messages but with distance!

Distributed BFS: Bellman-Ford Flav

Idea: Don‘t go through these time-consuming phases but blast out messages but with distance!



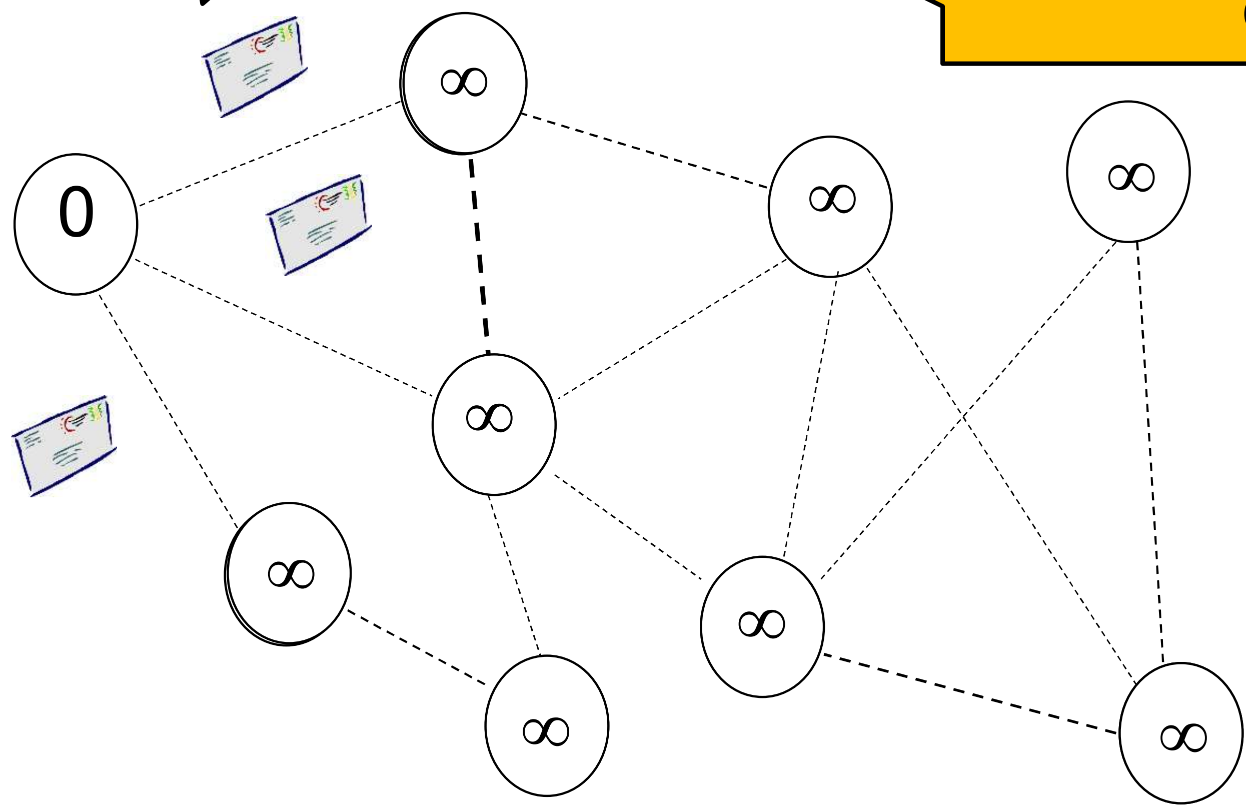
init to ∞

Initialize: root distance 0, other nodes ∞

Dis Bellman-Ford Flav

distance 1

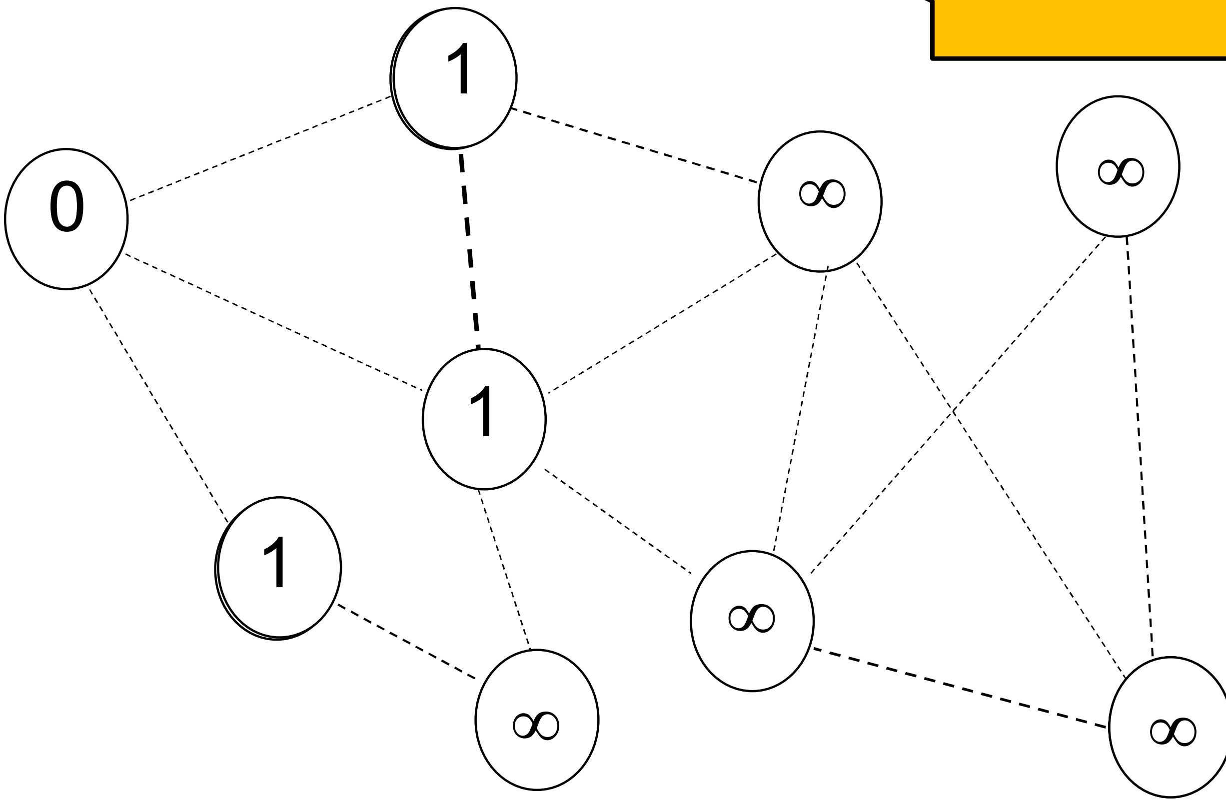
Idea: Don‘t go through these time-consuming phases but blast out messages but with distance!



Start: root sends distance 1 packet to neighbors

Distributed BFS: Bellman-Ford Flav

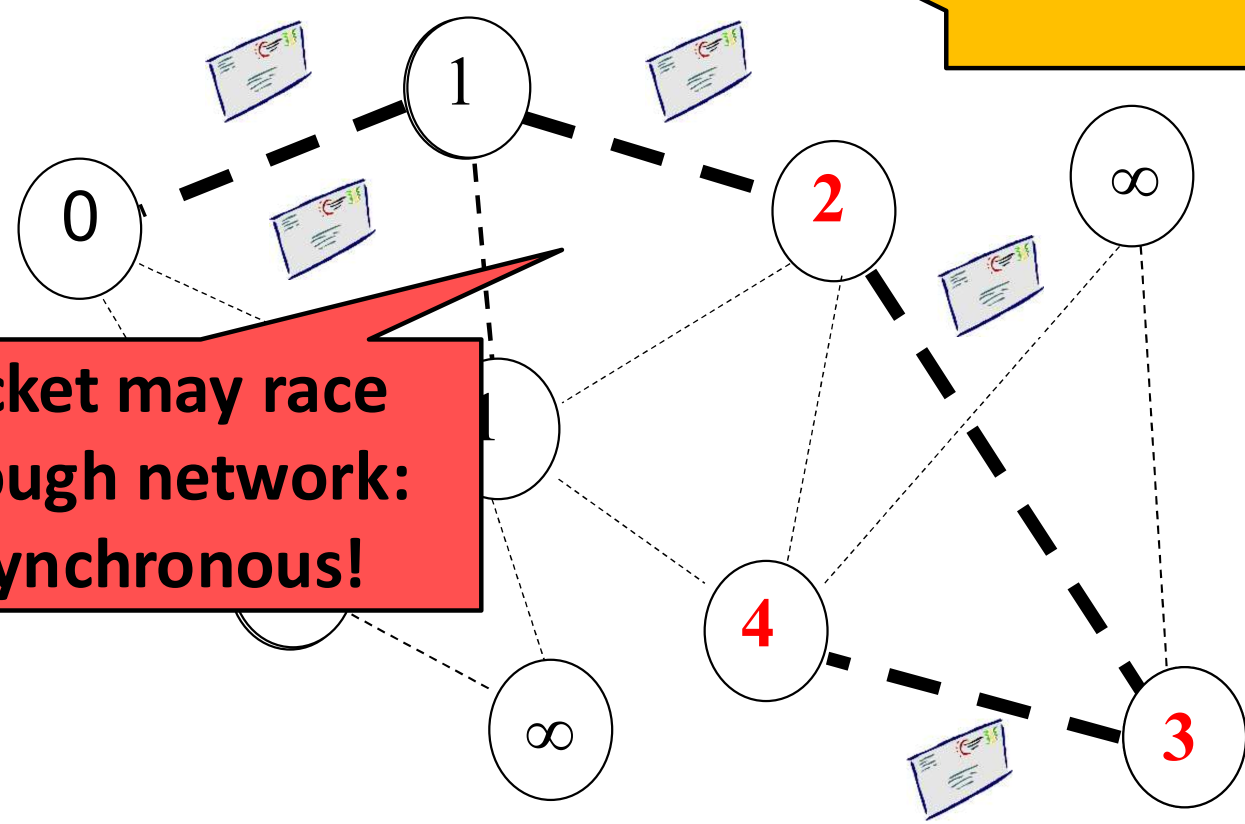
Idea: Don't go through these time-consuming phases but blast out messages but with distance!



Repeat: whenever receive new packet: check whether new minimal distance (if so change parent), and propagate!

Distributed BFS: Bellman-Ford Flav

Idea: Don't go through these time-consuming phases but blast out messages but with distance!

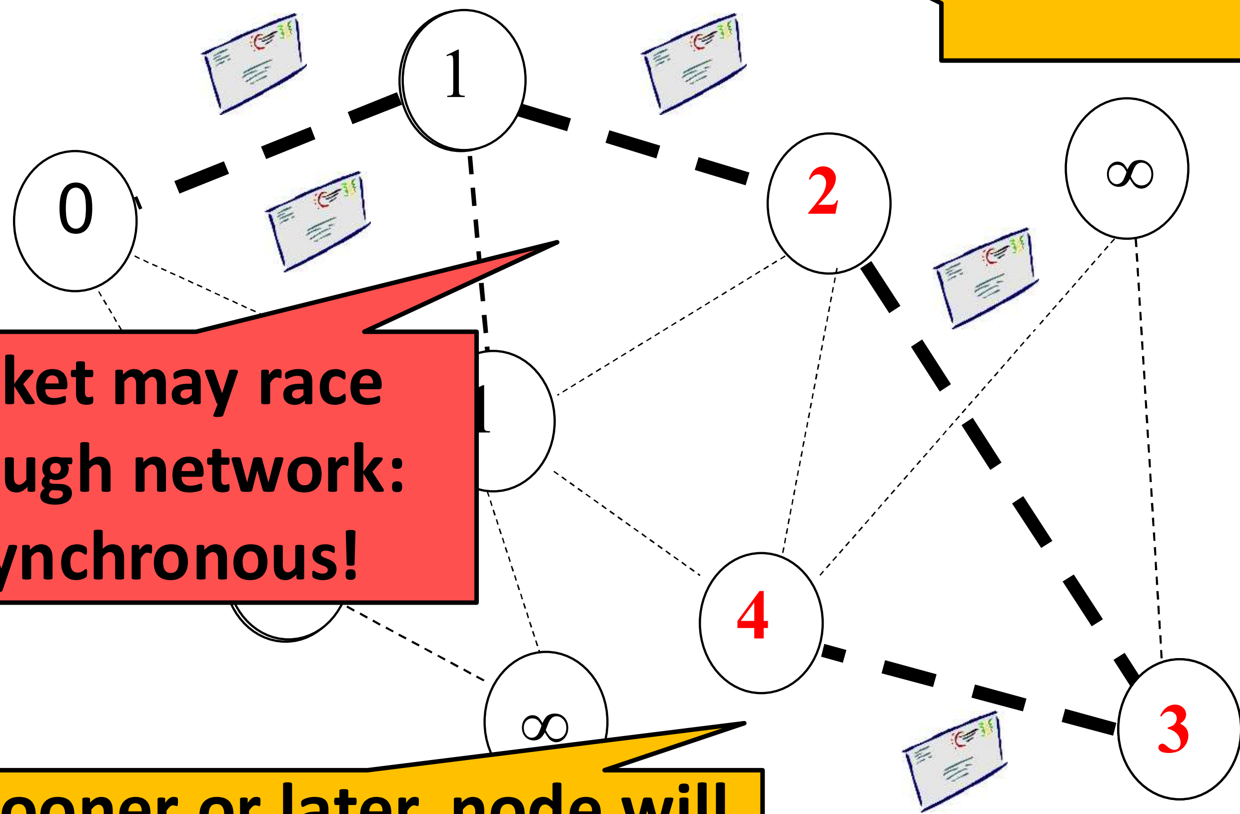


Packet may race through network: asynchronous!

Repeat: whenever receive new packet: check whether new minimal distance (if so change parent), and propagate!

Distributed BFS: Bellman-Ford Flav

Idea: Don't go through these time-consuming phases but blast out messages but with distance!



Packet may race through network: asynchronous!

But sooner or later, node will learn shorter distance!

Repeat: whenever receive new packet: check whether new minimal distance (if so change parent), and propagate!

Distributed BFS: Bellman-Ford Flavor

Bellman-Ford: compute shortest distances by flooding an all paths; best predecessor = parent in tree

Bellman-Ford Style

Each node u stores d_u, the distance from u to the root.
Initially, d_{root}=0 and all other distances are 1. Root starts algo by sending „1“ to all neighbors.

1. If a node u receives message „y“ with $y<d_u$

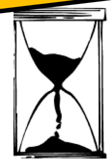
   d_u := y

   send „y+1“ to all other neighbors

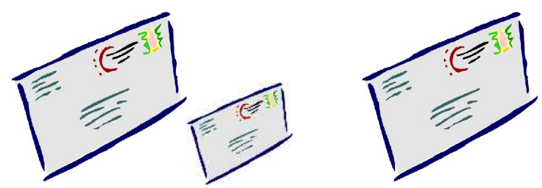
Analysis

How is this defined?! Assuming a unit upper bound on per link delay!

Time Complexity?



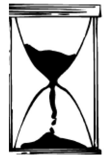
Message Complexity?



Analysis

Worst propagation time is simply the diameter.

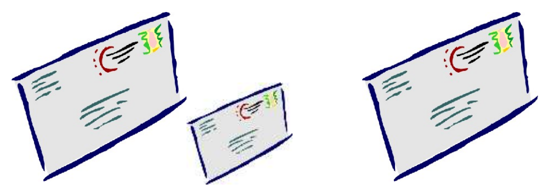
Time Complexity?



O(D) where D is diameter of graph. ☺

By induction: By time d, node at distance d got „d“.
Clearly true for d=0 and d=1.
A node at distance d has neighbor at distance d-1 that got „d-1“ on time by induction hypothesis. It will send „d“ in next time slot...

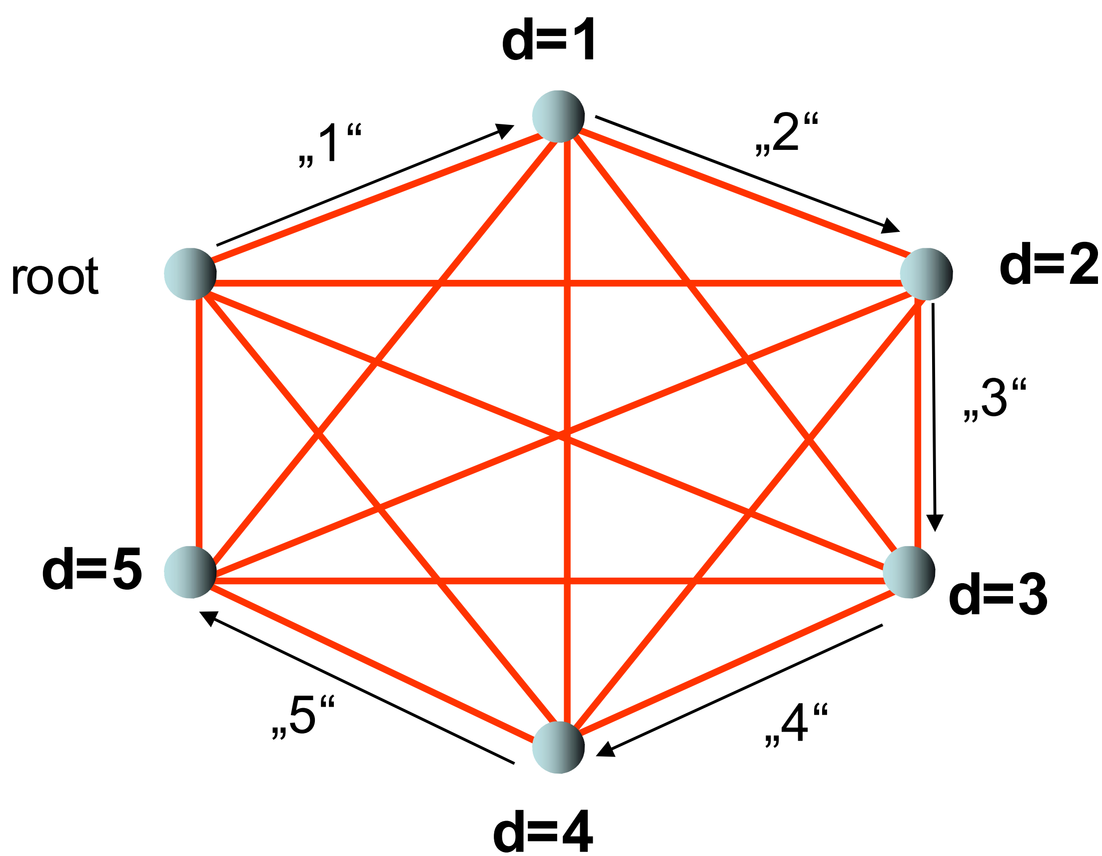
Message Complexity?



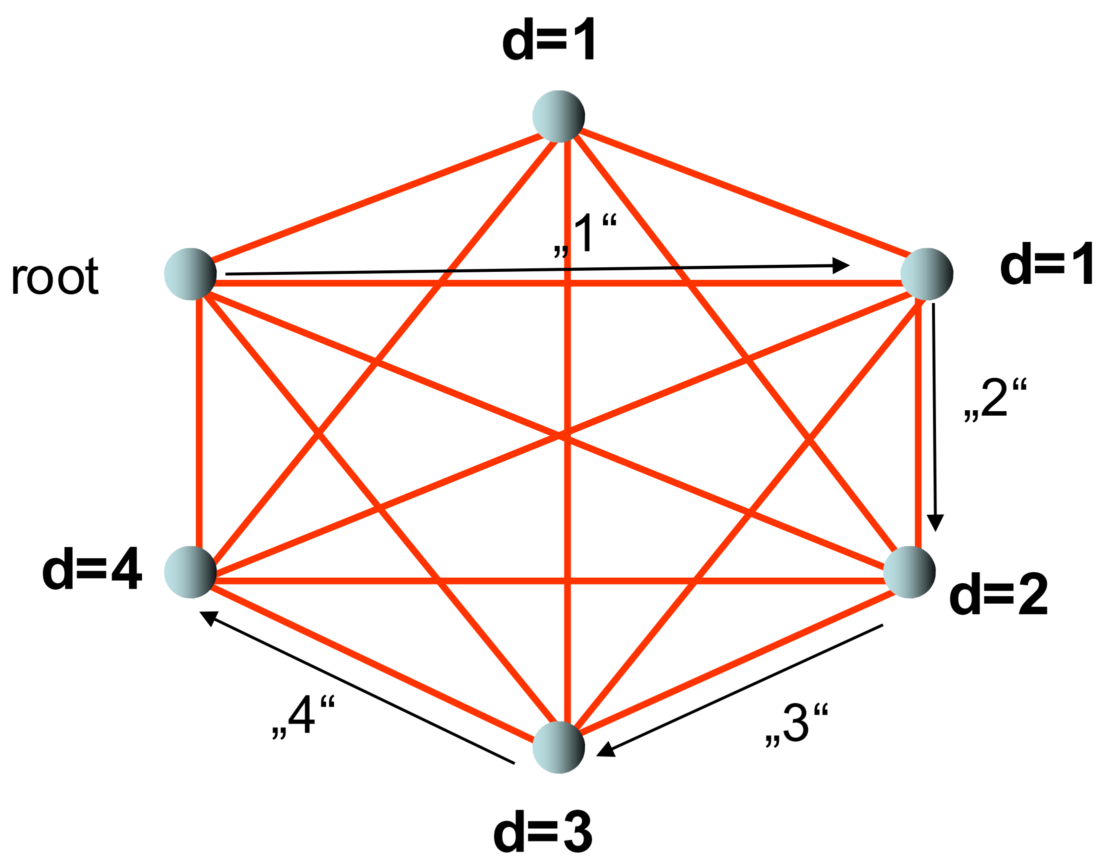
O(mn) where m is number of edges, n is number of nodes. ☹

Because: A node can reduce its distance at most n-1 times (recall: asynchronous!). Each of these times it sends an upate message to all its neighbors

Bellman-Ford with Many Messages

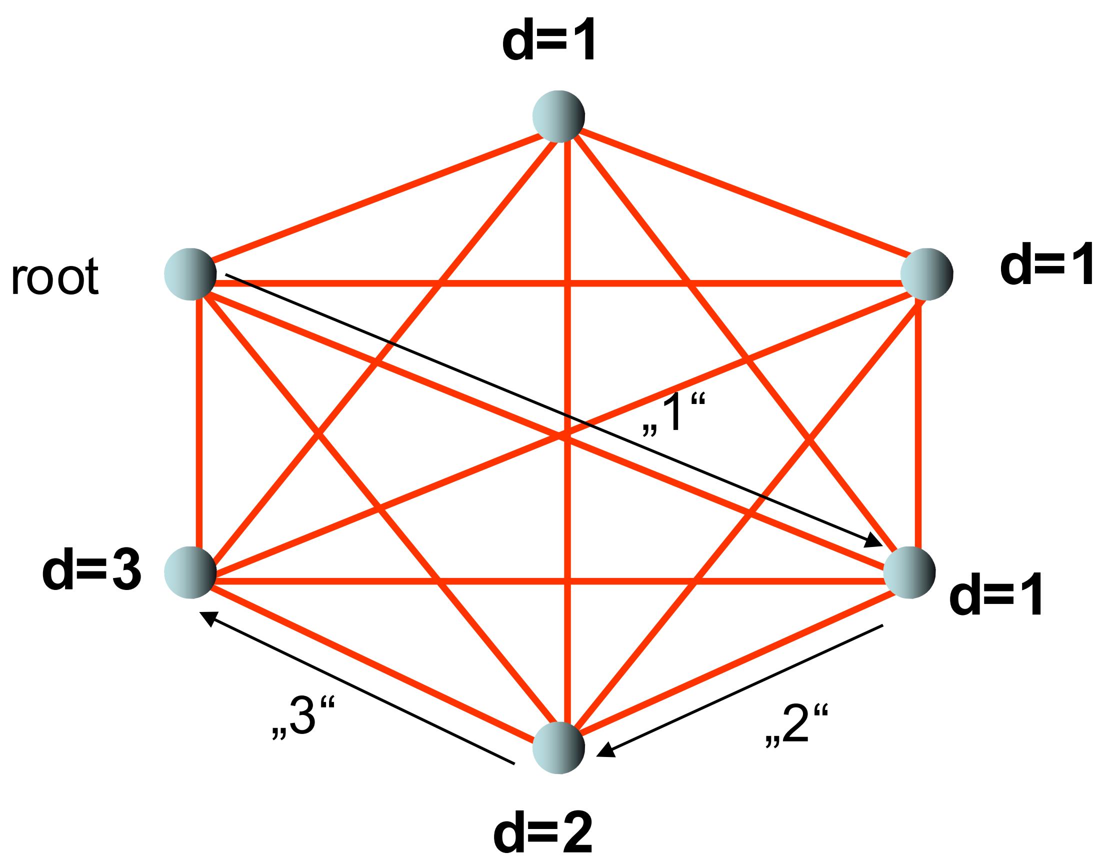


Bellman-Ford with Many Messages



Everyone has a new best distance and informs neighbors!

Bellman-Ford with Many Messages



Everyone has a new best distance and informs neighbors!

Discussion

Which algorithm is better?

Dijkstra has better message complexity, Bellman-Ford better time complexity.

Can we do better?

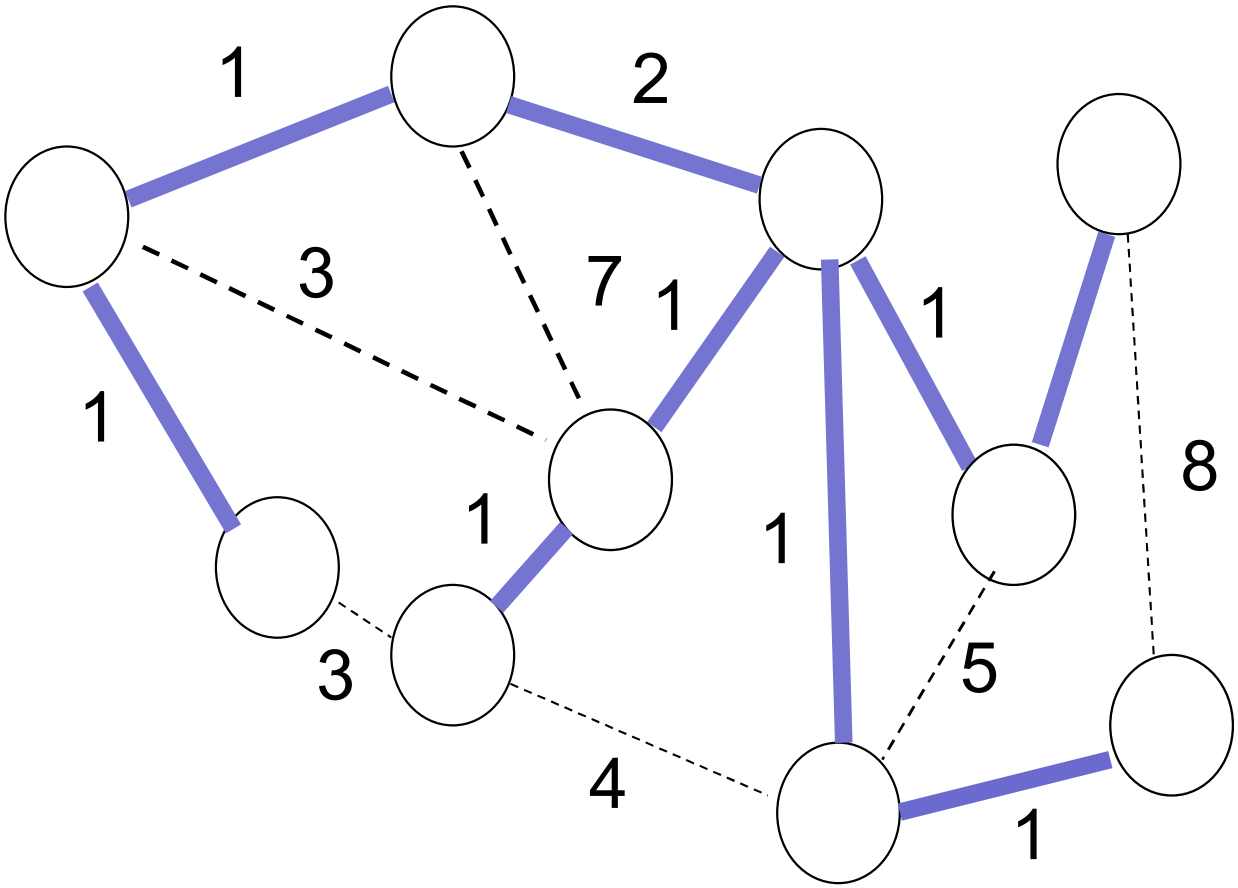
Yes, but not in this course... ☺

Remark: Asynchronous algorithms can be made synchronous... (e.g., by central controller or better: local synchronizers)

How to compute an MST?

MST

Tree with edges of minimal total weight.



Idea: Exploit Basic Fact of MST: Blue Edges

Blue Edge

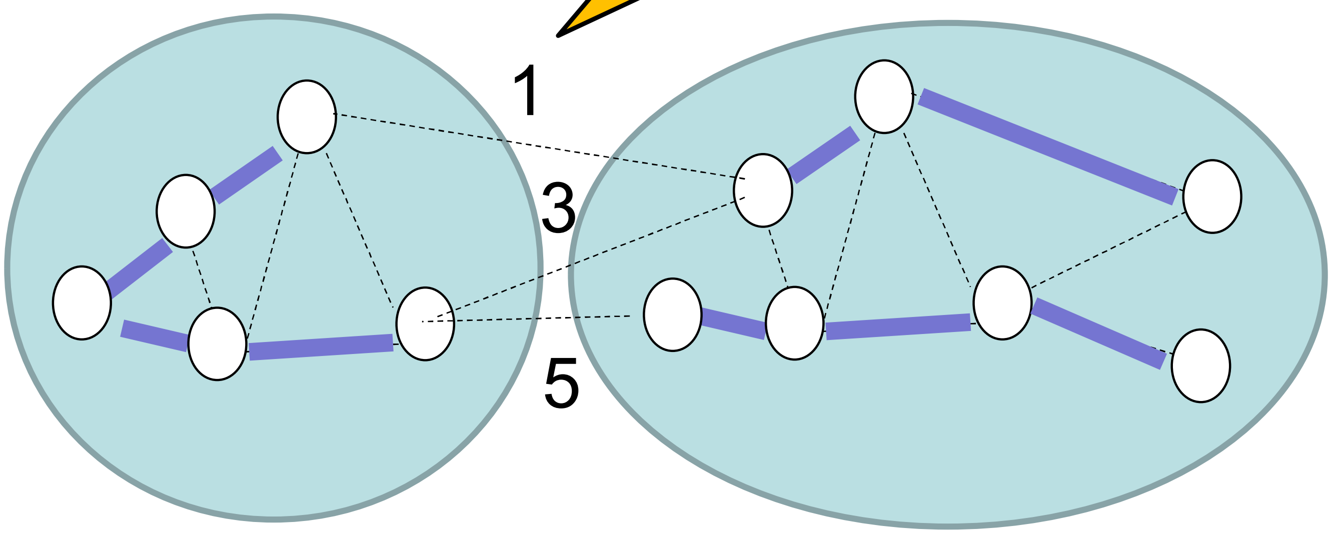
Let T be an MST and T' a subgraph of T.
Edge e=(u,v) is *outgoing edge* if u ∊ T' and v ∉ T'.
The outgoing edge of minimal weight is called *blue edge*.

Lemma

If T is the MST and T' a subgraph of T, then the blue edge of T' is also part of T.

It holds: the lightest edge across a cut must be part of the MST!

By contradiction: otherwise get a cheaper MST by swapping the two cut edges!



Gallager-Humblet-Spira

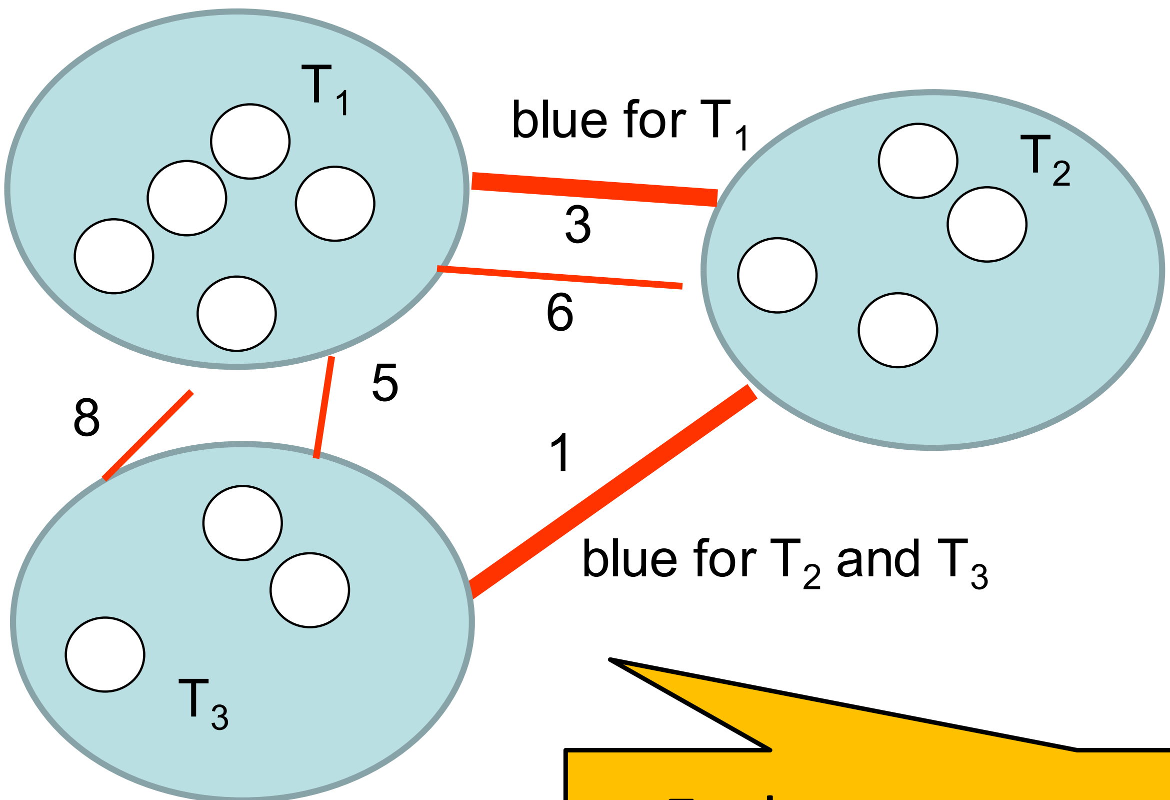
Gallager-Humblet-Spira

Basic idea: Grow components in parallel and merge them at the blue edge! Using Covergecast.

Gallager-Humblet-Spira

Basic idea: Grow components in parallel and merge them at the blue edge! Using Covergecast.

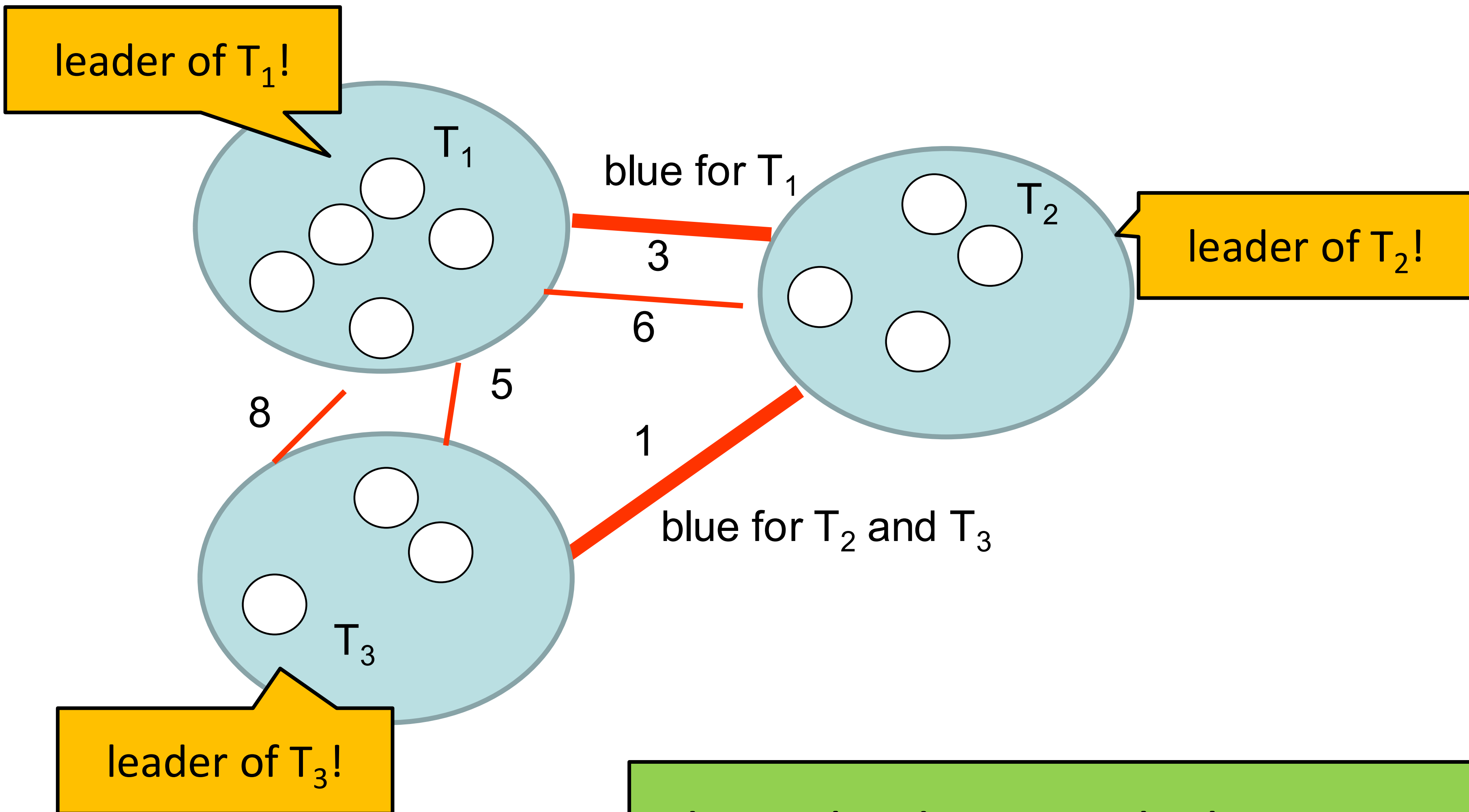
Assume some components have already emerged:



Each component has only one blue edge (cheapest outgoing): loops impossible, can take them in parallel!

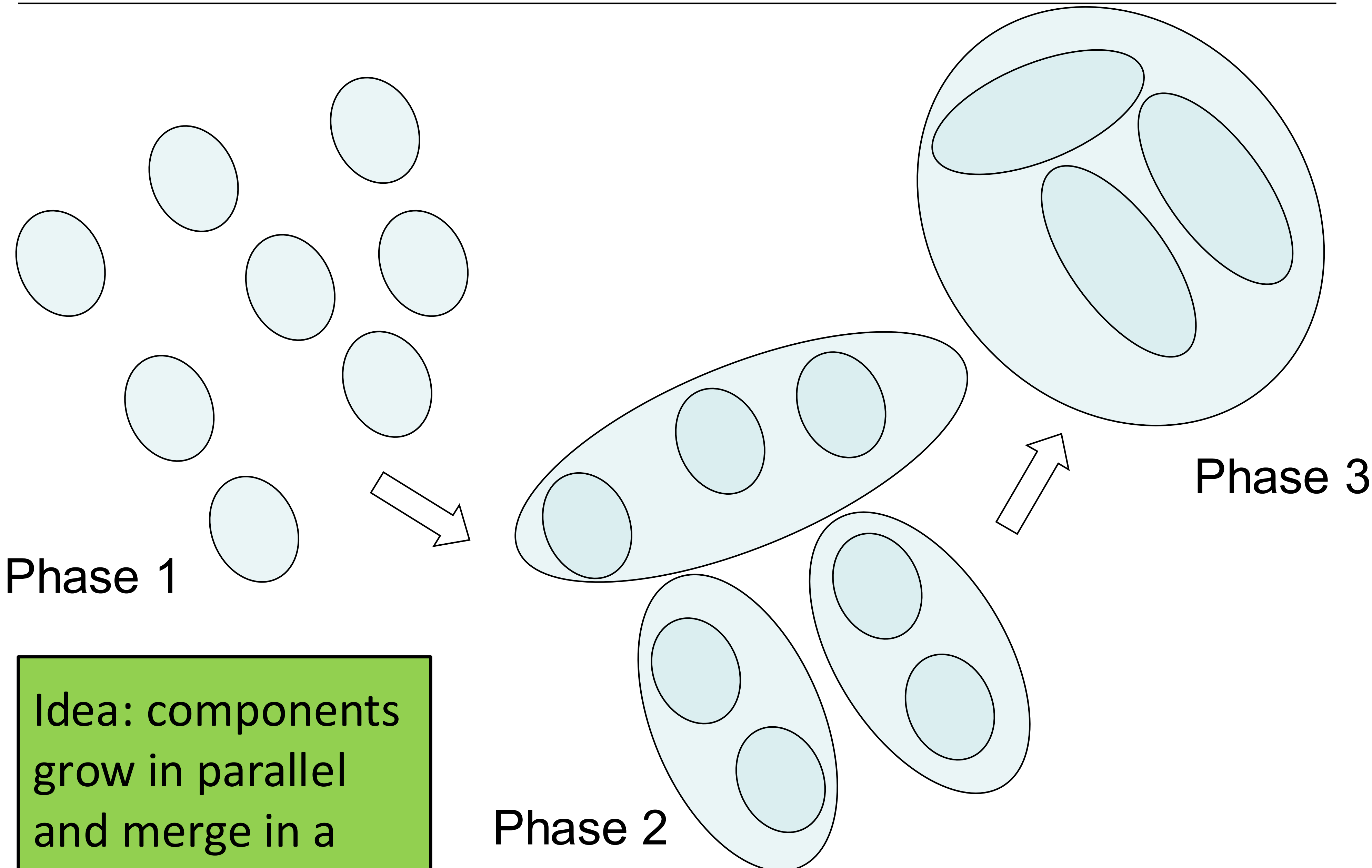
Gallager-Humblet-Spira

Basic idea: Grow components in parallel and merge them at the blue edge! Using Covergecast.



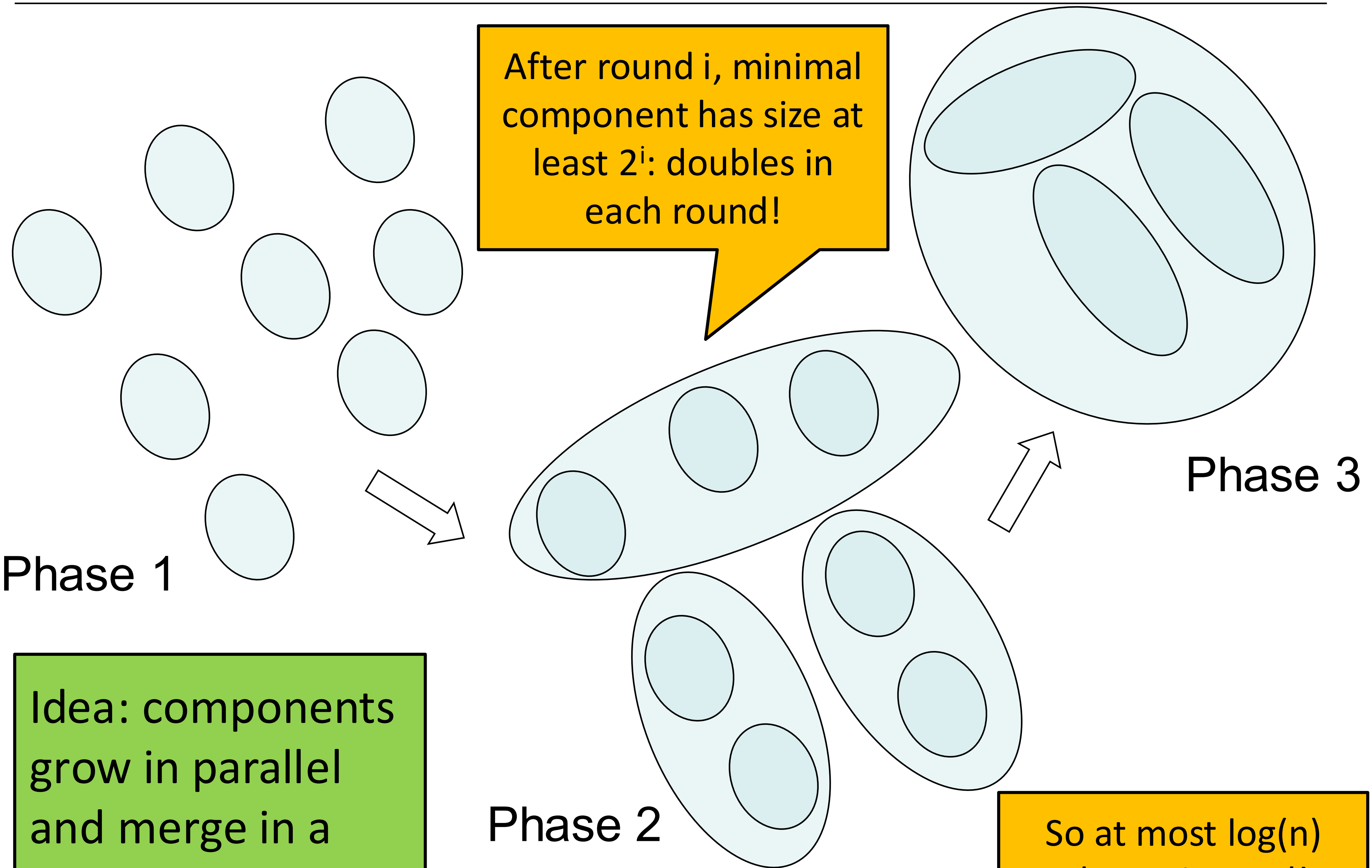
Idea: a leader in each cluster responsible of finding blue edge with convergecast!

Gallager-Humblet-Spira: High-level View



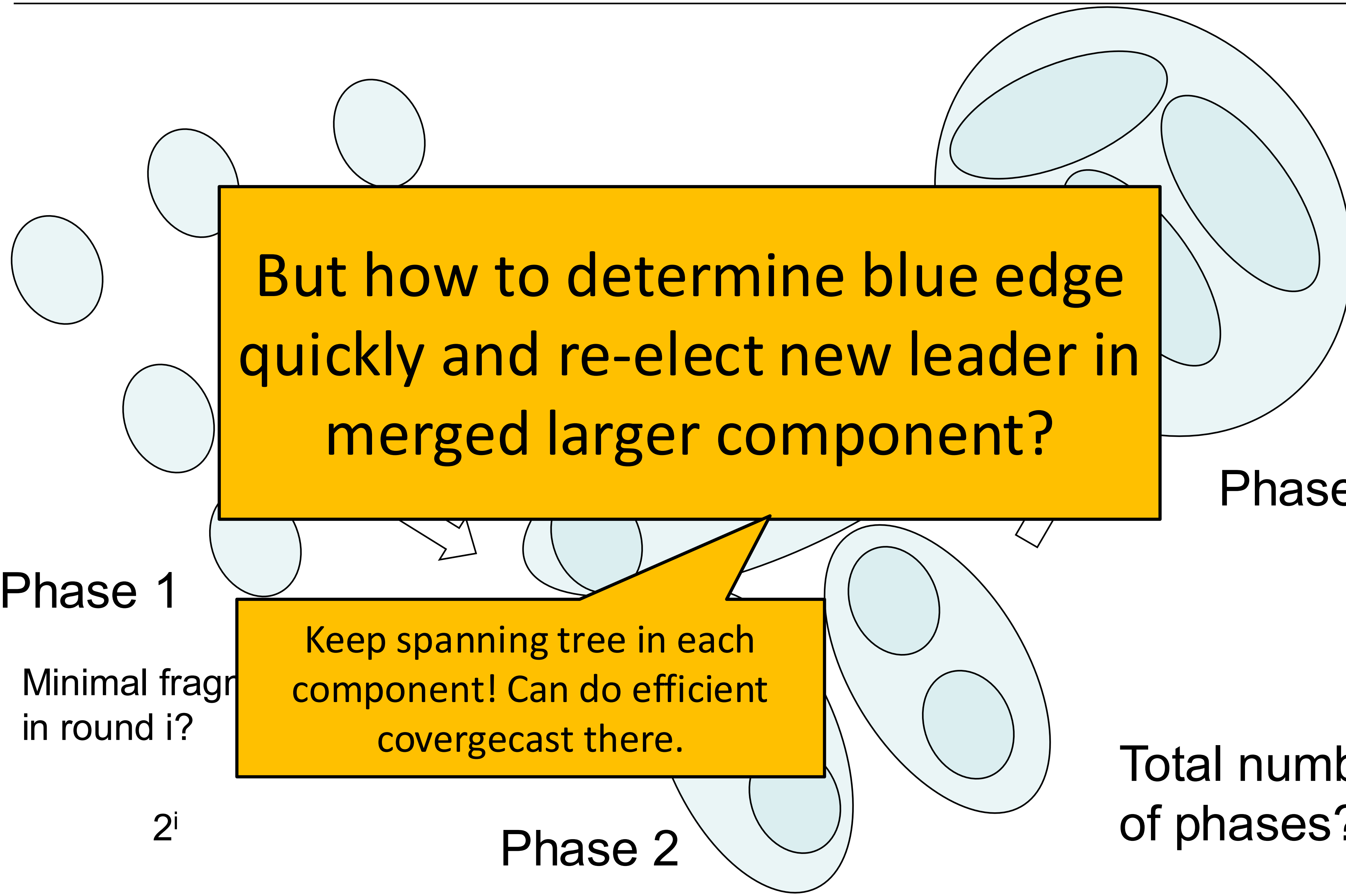
Idea: components grow in parallel and merge in a loop-free manner!

Gallager-Humblet-Spira: High-level View



Idea: components grow in parallel and merge in a loop-free manner!

Gallager-Humblet-Spira: High-level View



But how to determine blue edge quickly and re-elect new leader in merged larger component?

Keep spanning tree in each component! Can do efficient covergecast there.

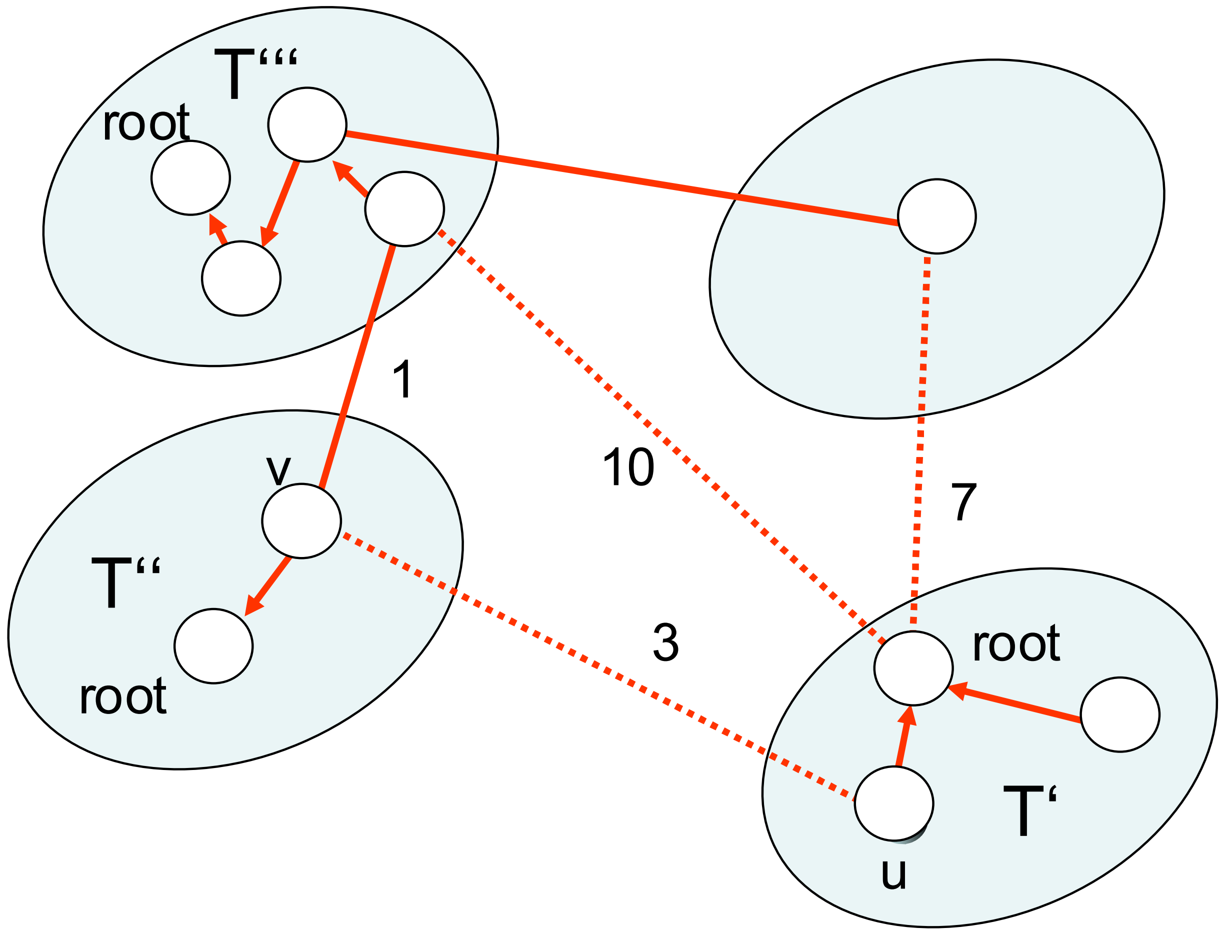
Minimal fragr
in round i?

2^i

Total number of phases?

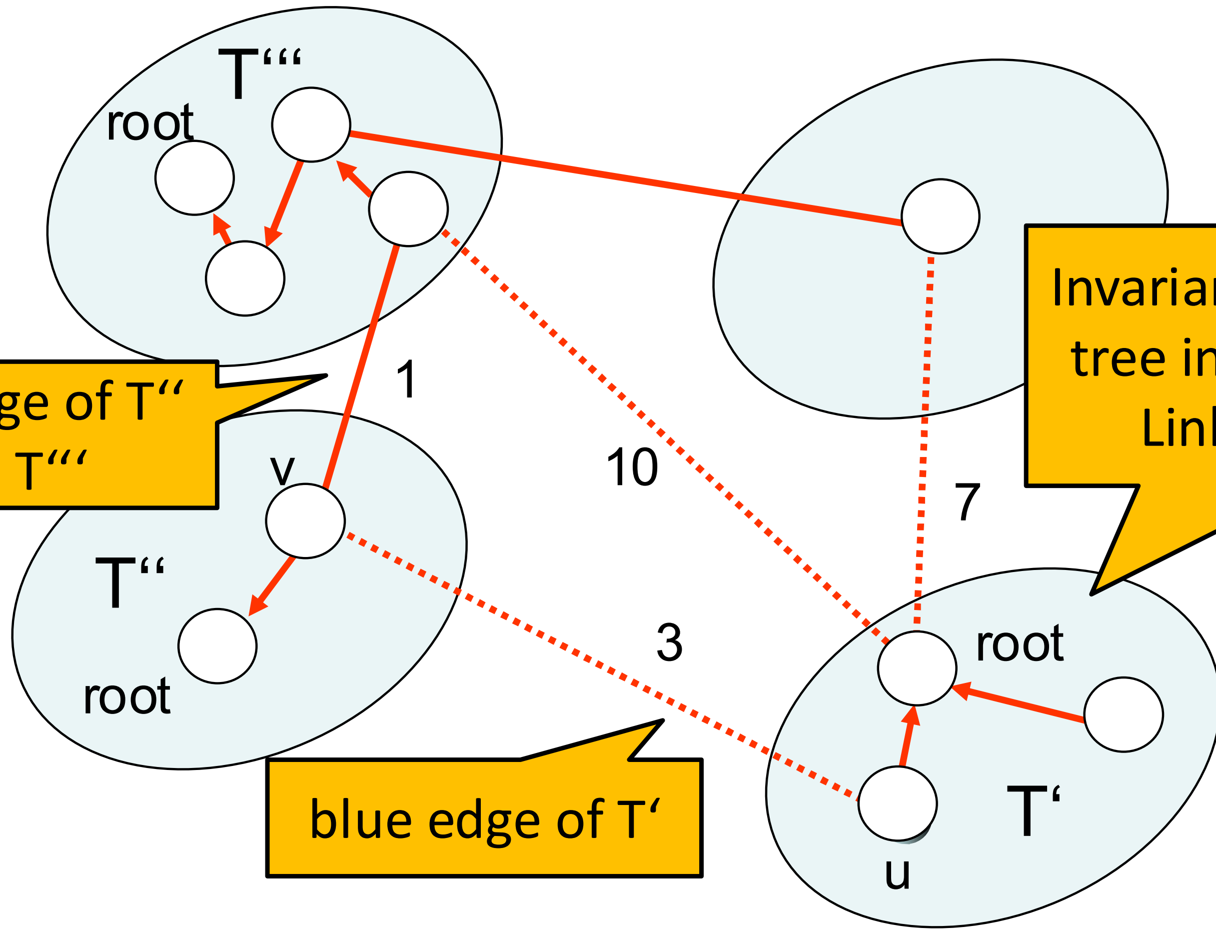
Example: Agree on a New Root

How to merge T‘ and T‘‘ across (u,v)?



Example: Agree on a New Root

How to merge T‘ and T‘‘ across (u,v)?



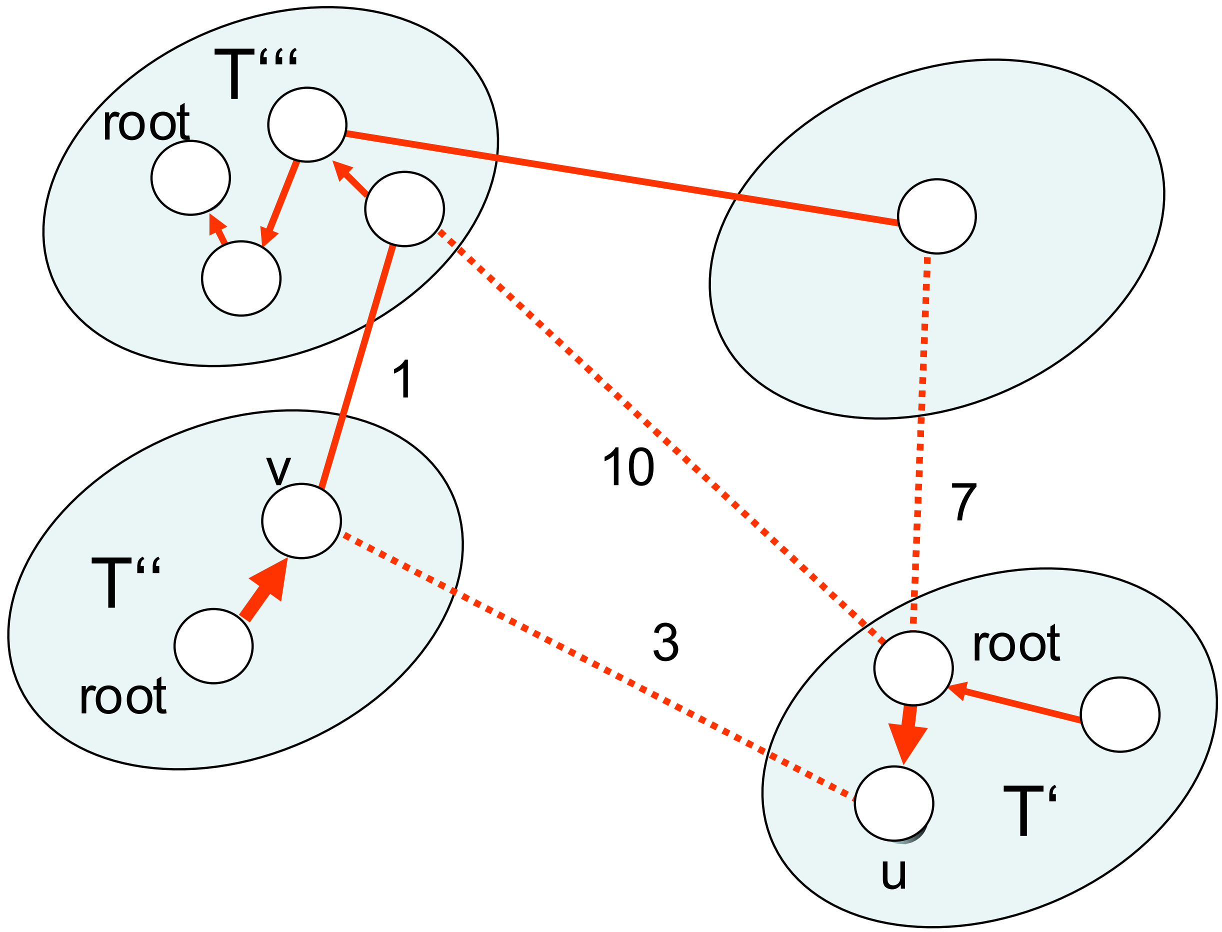
Invariant: rooted spanning tree in each component! Links point to root.

blue edge of T‘‘ and T‘‘‘

blue edge of T‘

Example: Agree on a New Root

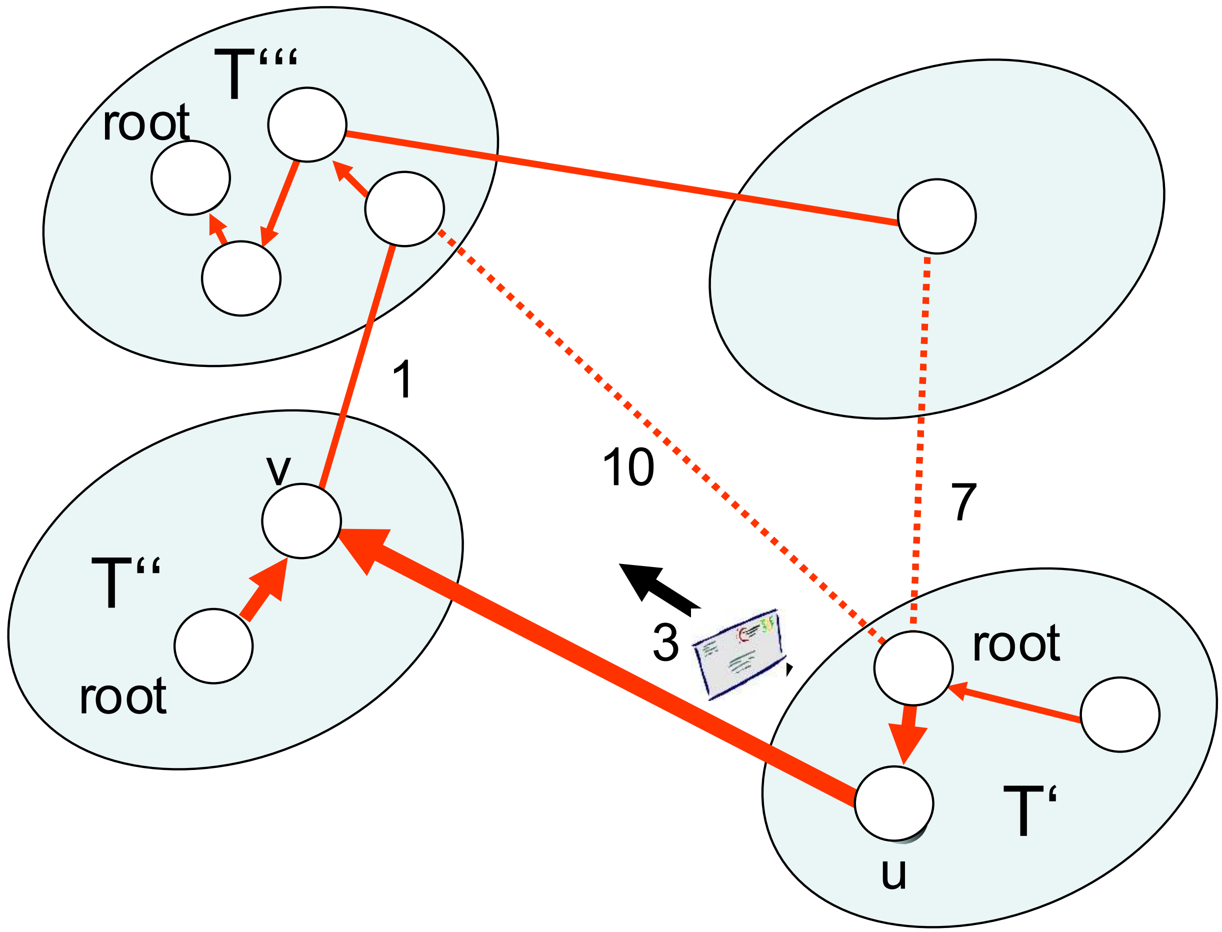
How to merge T' and T'' across (u,v)?



Step 1: invert path from root to u and v.

Example: Agree on a New Root

How to merge T‘ and T‘‘ across (u,v)?



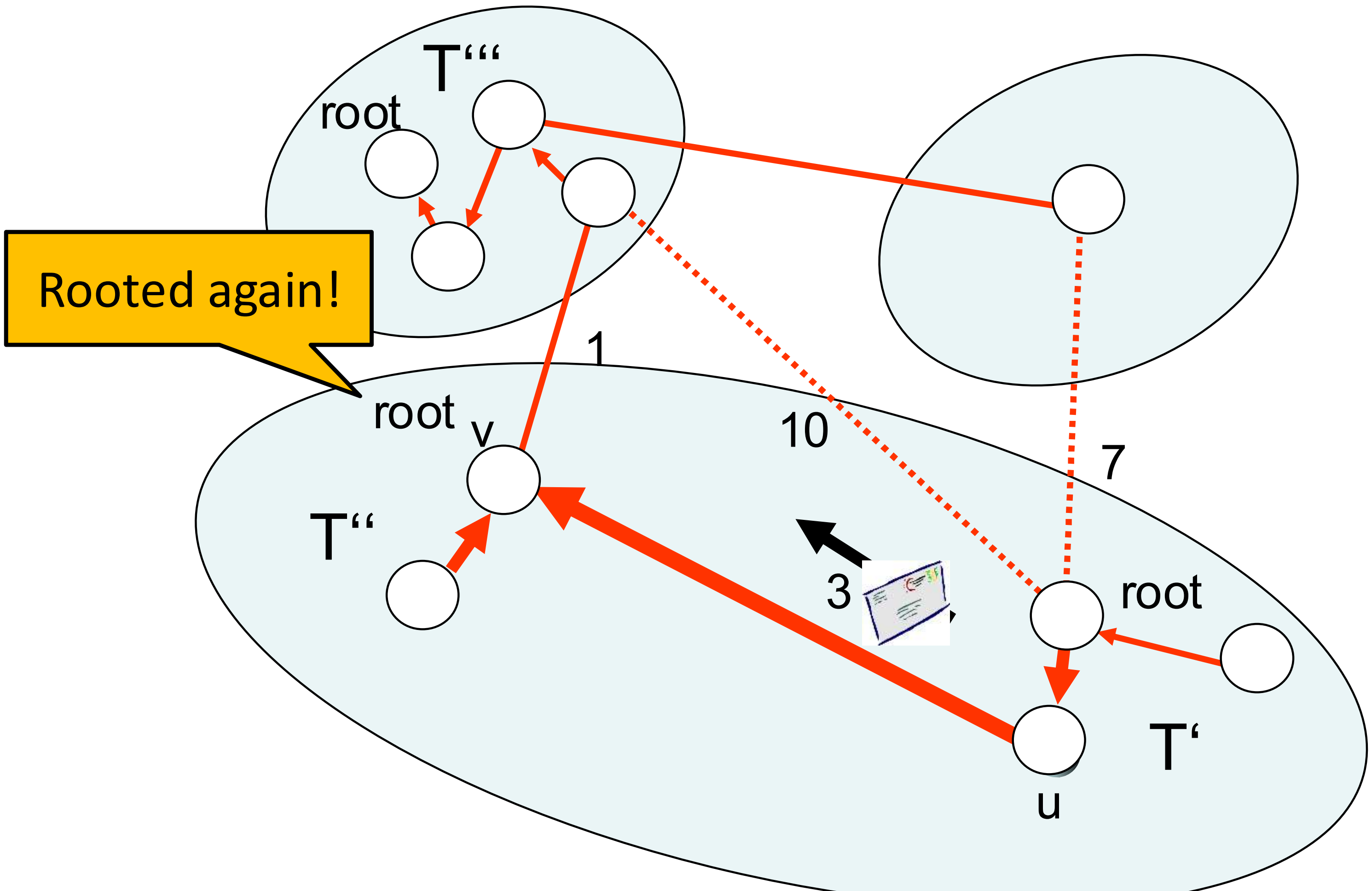
Step 1: invert path from root to u and v.

Step 2: send merge request across blue edge (u,v). Here only blue edge for T‘ so one message!

Step 3: v becomes new root overall!

Example: Agree on a New Root

How to merge T‘ and T‘‘ across (u,v)?



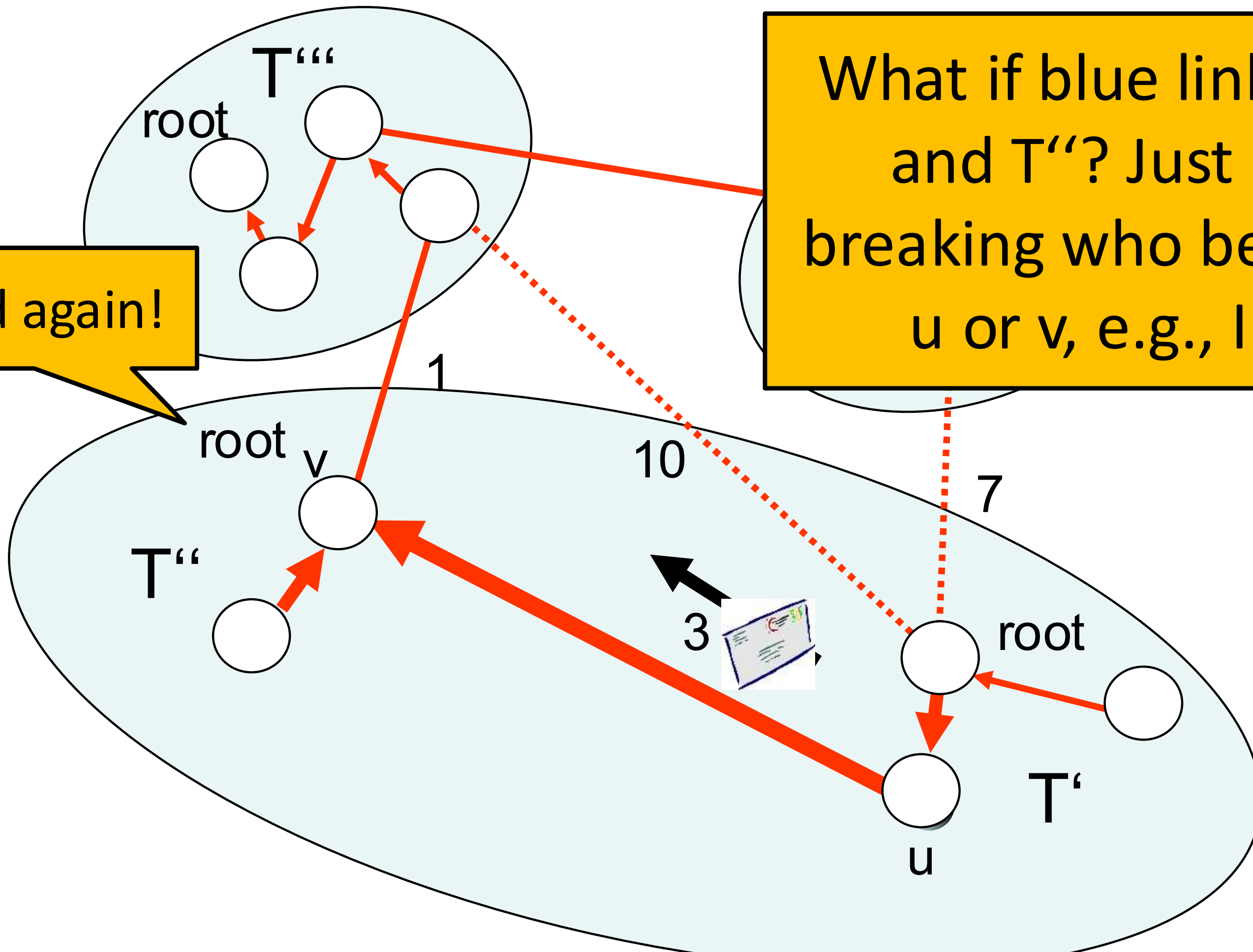
Step 1: invert path from root to u and v.

Step 2: send merge request across blue edge (u,v). Here only blue edge for T‘ so one message!

Step 3: v becomes new root overall!

Example: Agree on a New Root

How to merge T‘ and T‘‘ across (u,v)?



What if blue link for both T‘ and T‘‘? Just make tie-breaking who becomes root, u or v, e.g., ID based!

Rooted again!

Step 1: invert path from root to u and v.

Step 2: send merge request across blue edge (u,v). Here only blue edge for T‘ so one message!

Step 3: v becomes new root overall!

Distributed Kruskal

Idea: Grow components by learning blue edge!
But do many fragments in parallel!

Gallager-Humblet-Spira

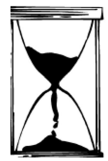
Initially, each node is root of its own fragment.

Repeat (until all nodes in same fragment)

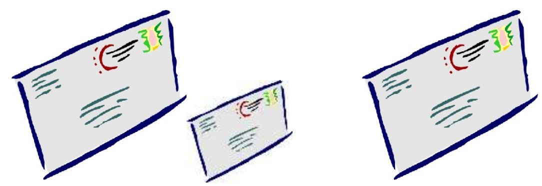
1. nodes learn fragment IDs of neighbors
2. root of fragment finds blue edge (u,v) by convergecast
3. root sends message to u (inverting parent-child)
4. if v also sent a merge request over (u,v), u or v becomes new root depending on smaller ID (make trees directed)
5. new root informs fragment about new root (convergecast on „MST“ of fragment): new fragment ID

Analysis

Time Complexity?



Message Complexity?

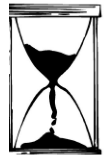


Each phase mainly consists of two convergecasts, so O(D) time and O(n) messages per phase?

Analysis

Log n phases with O(n) time convergecast: spanning tree is not BFS!

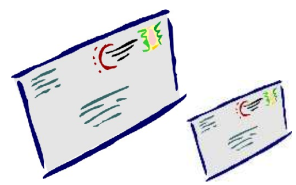
Time Complexity?



The size of the smallest fragment at least doubles in each phase, so it's logarithmic. But converge cast may take n hops.

O(n log n) where n is graph size.

Message Complexity?



Log n phases but in each phase need to learn leader ID of neighboring fragments, for *all* neighbors!

O(m log n) where m is number of edges: at most O(1) messages on each edge in a phase.

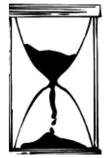
Really needed? Each phase mainly consists of two convergecasts, so O(n) time and O(n) messages. In order to learn fragment IDs of neighbors, O(m) messages are needed (again and again: ID changes in each phase).

Yes, we can do better. ☺

Analysis

Log n phases with O(n) time convergecast: spanning tree is not BFS!

Time Complexity?

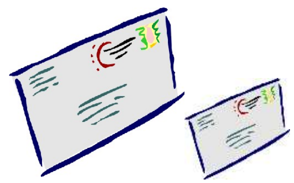


The size of the smallest fragment at least doubles in each phase, so it's logarithmic. But converge cast may take n hops.

Log n phases but in each phase need to learn leader ID of neighboring fragments, for *all* neighbors!

O(n log n) where n is graph size.

Message Complexity?



O(m log n) where m is number of edges: at most O(1) messages on each edge in a phase.

Really needed? Each phase mainly consists of two convergecasts, so O(n)
time … (m)
mess…

Note: this algorithm can solve leader election! Leader = last surviving root!

Literature for further reading:

- Peleg‘s book

End of lecture

Exercise 2: License to Match

In preparation of a highly dangerous mission, the participating agents of the gargantuan Liechtensteinian secret service (LSS) need to work in pairs of two for safety reasons. All members in the LSS are organized in a tree hierarchy. Communication is only possible via the official channel: an agent has a secure phone line to his direct superior and a secure phone line to each of his direct subordinates. Initially, each agent knows whether or not he is taking part in this mission. The goal is for each agent to find a partner.

a) Devise an algorithm that will match up a participating agent with another participating agent given the constrained communication scenario. A "match" consists of an agent knowing the identity of his partner and the path in the hierarchy connecting them. Assume that there is an even number of participating agents so that each one is guaranteed a partner. Furthermore, observe that[1] the phone links connecting two paired-up agents need to remain open at all times. Therefore, you cannot use the same link (i.e., an edge) twice when connecting an agent with his partner.

b) What are the time and message (i.e., "phone call") complexities of your algorithm?