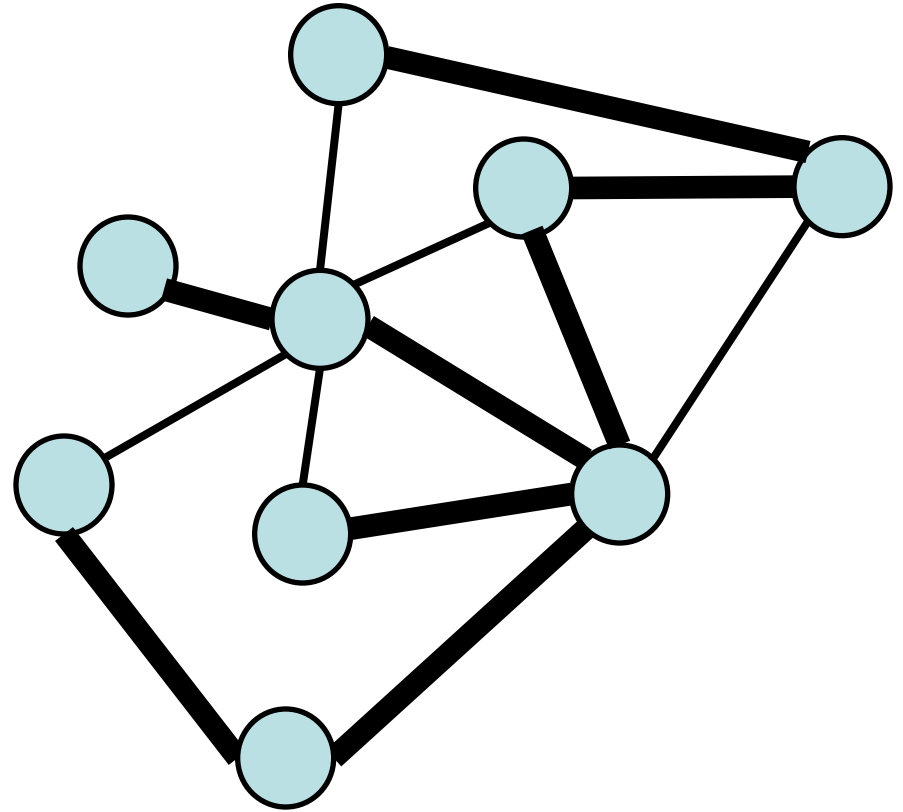


Network Algorithms

Tree Algorithms

Why (spanning) trees?

E.g., efficient broadcast, aggregation, routing, algebraic gossip...



Important trees?

E.g., breadth-first trees (BFS), minimal spanning trees (MST), ...

Spanning Tree

Cycle-free subgraph spanning all nodes.

In this lecture:

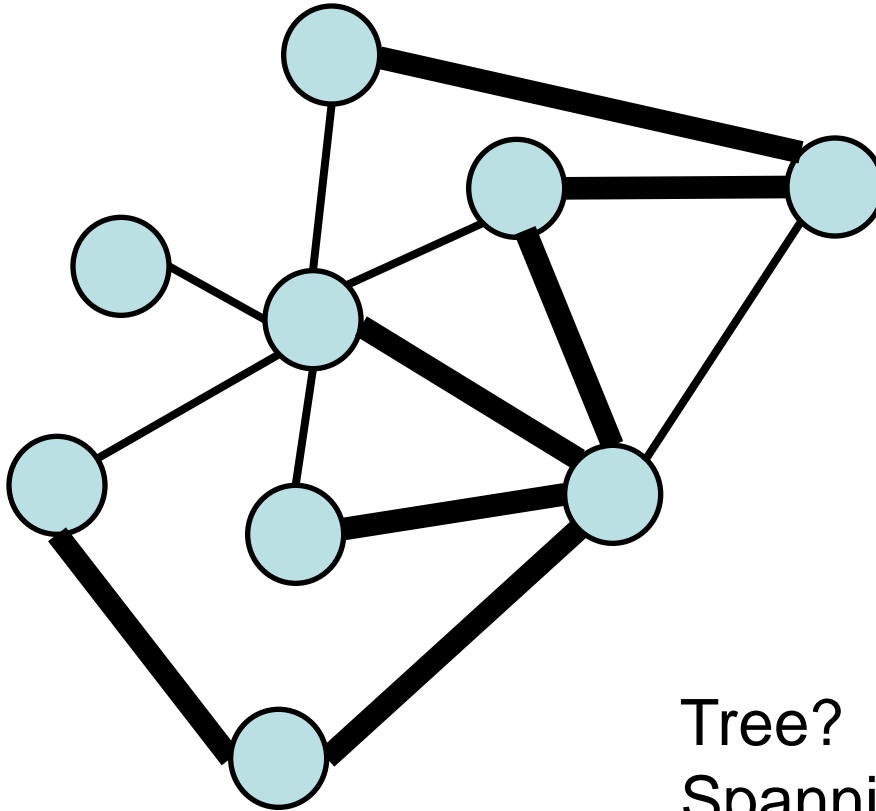
BFS

**Shortest path spanning tree (unweighted)
from given source.**

MST

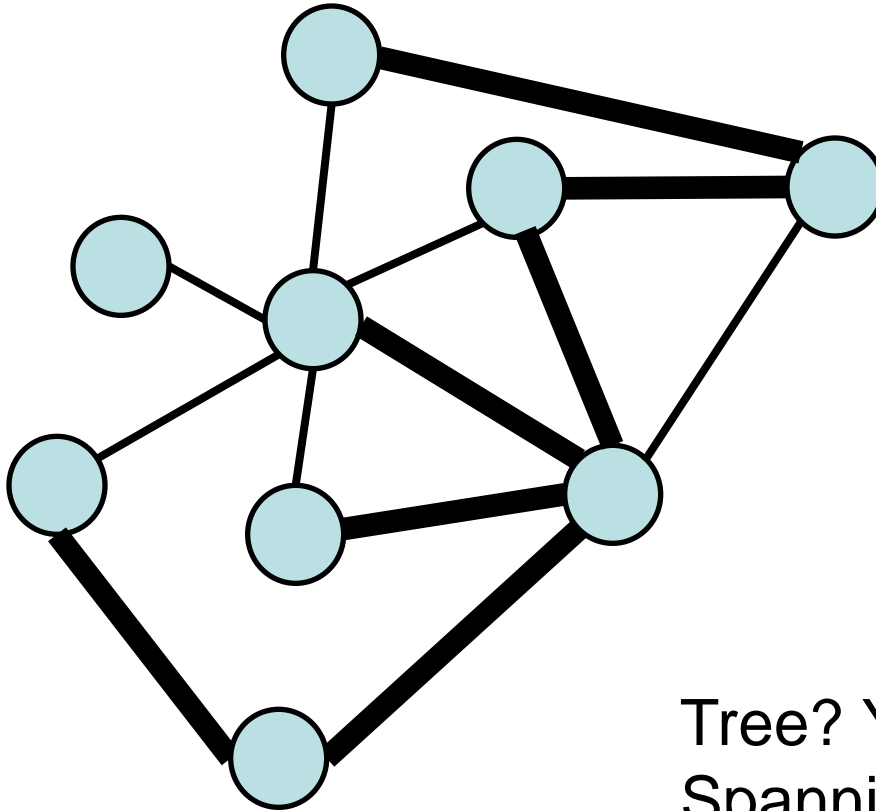
Spanning tree of minimal weight.

Definitions



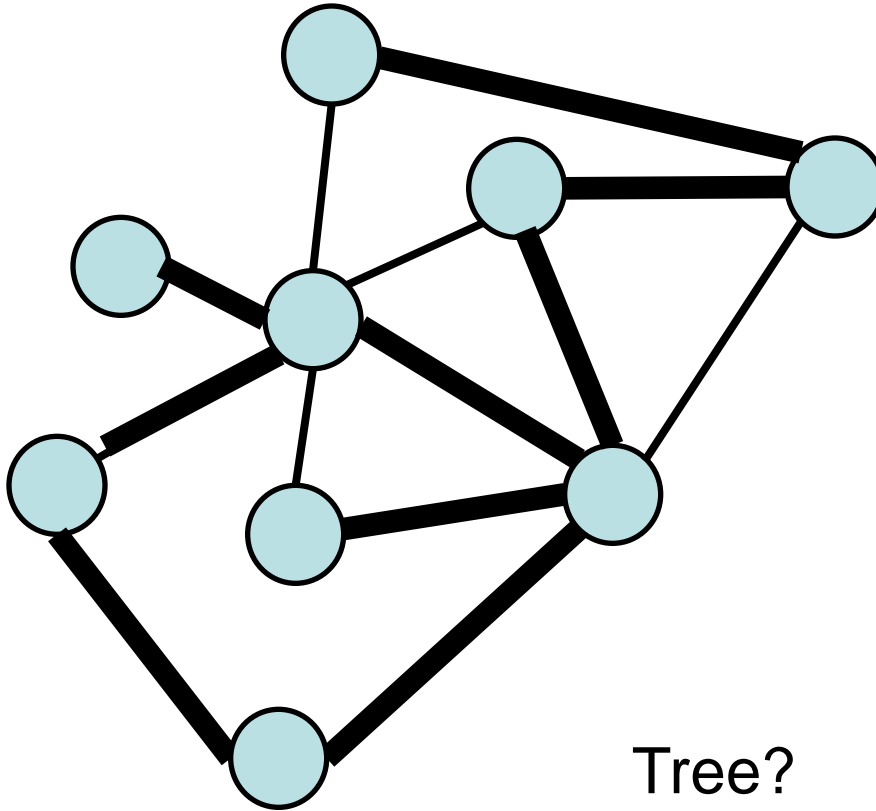
Tree?
Spanning tree?

Definitions



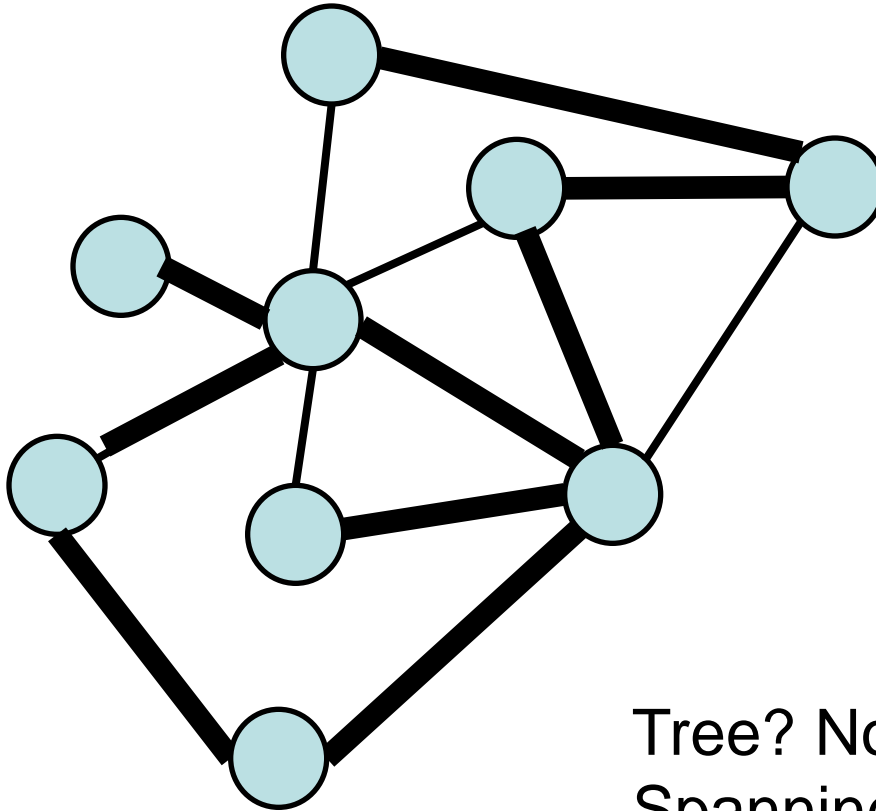
Tree? Yes
Spanning tree? No

Definitions



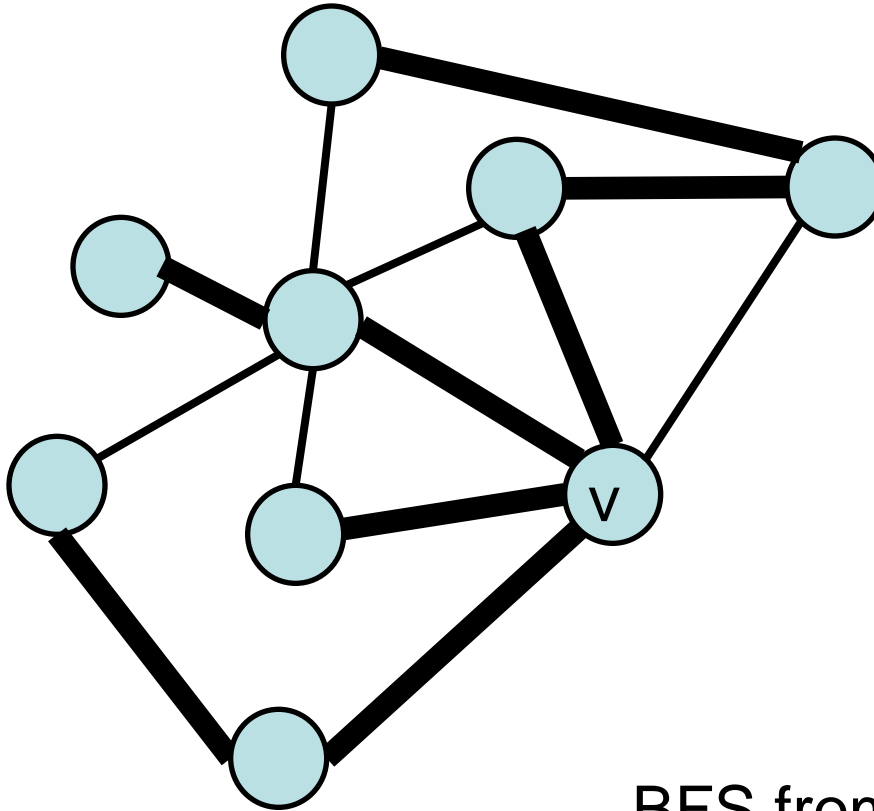
Tree?
Spanning tree?

Definitions



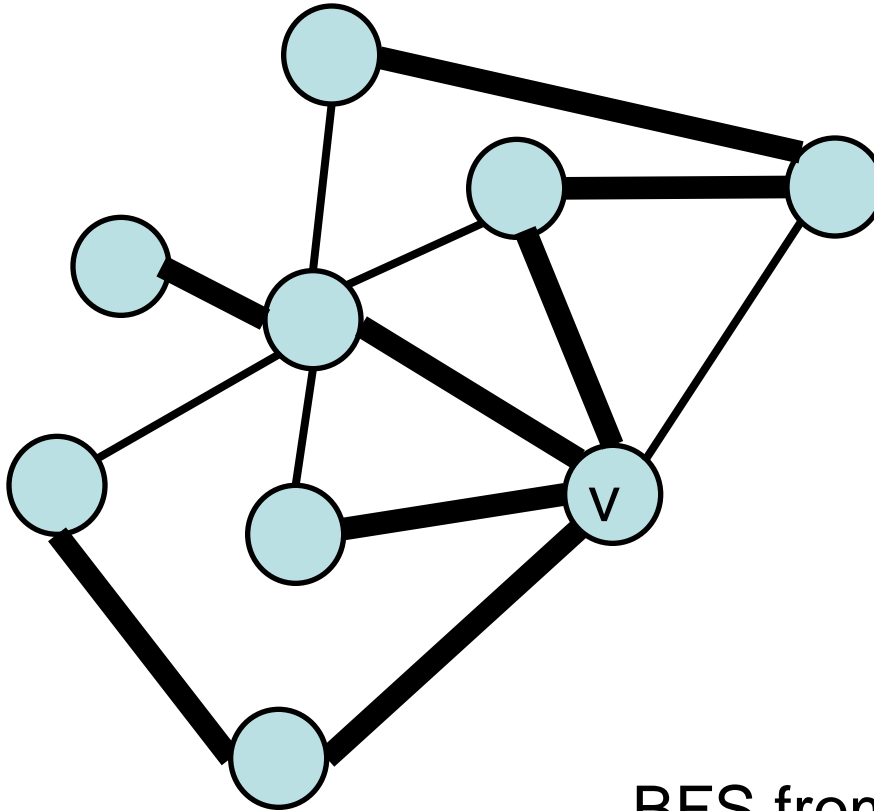
Tree? No
Spanning tree?
No, but **spanning subgraph!**

Definitions



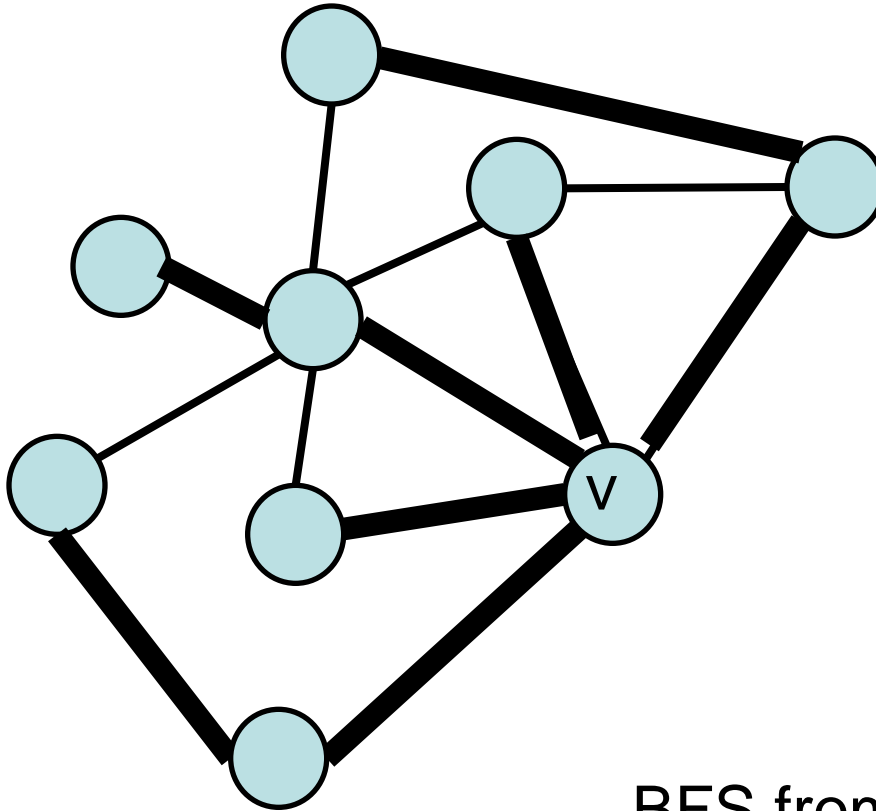
BFS from v?

Definitions



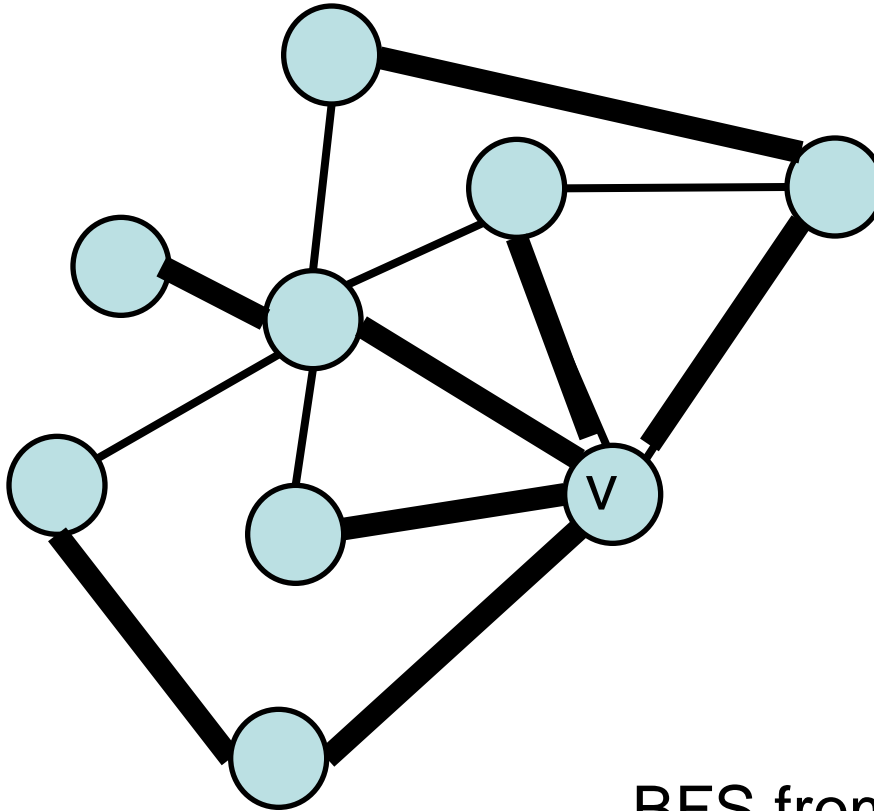
BFS from v? No.

Definitions



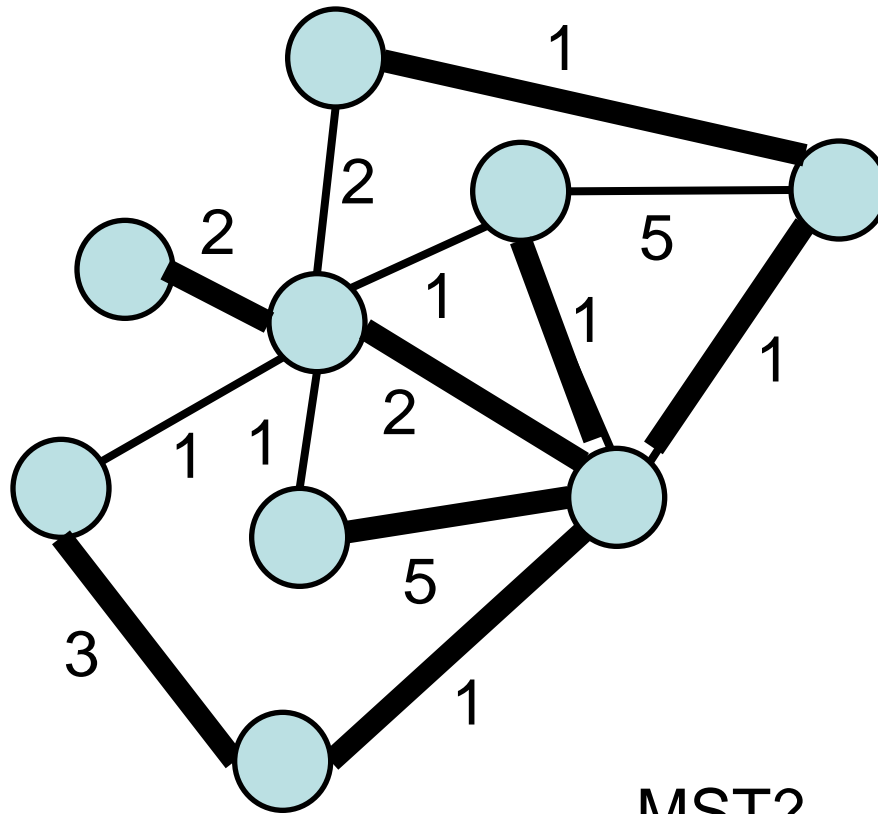
BFS from v?

Definitions

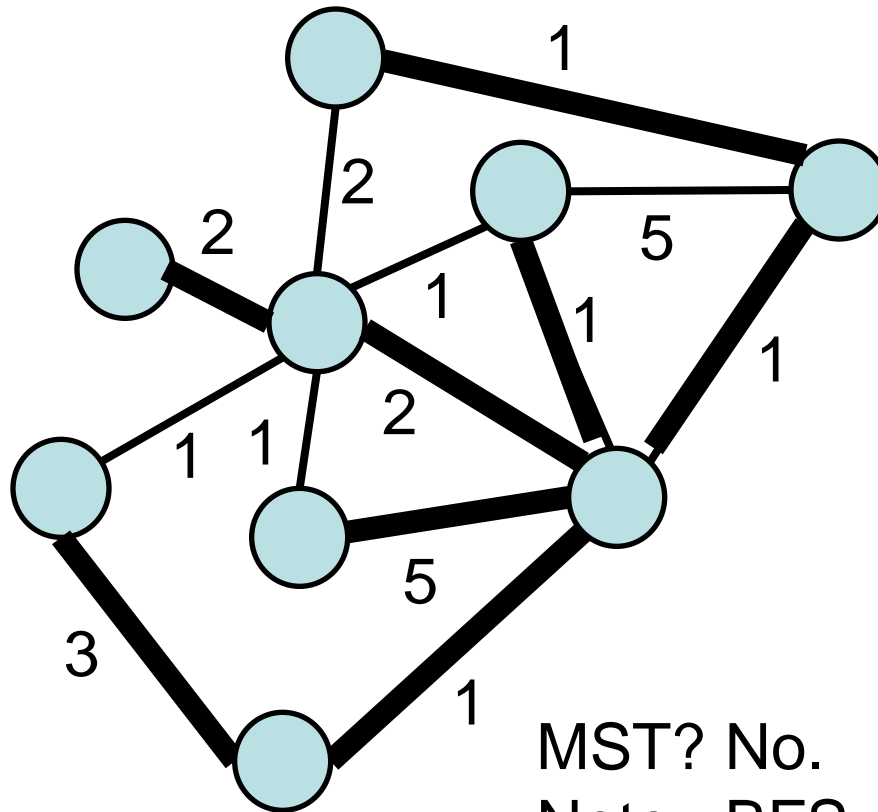


BFS from v? Yes.

Definitions



Definitions

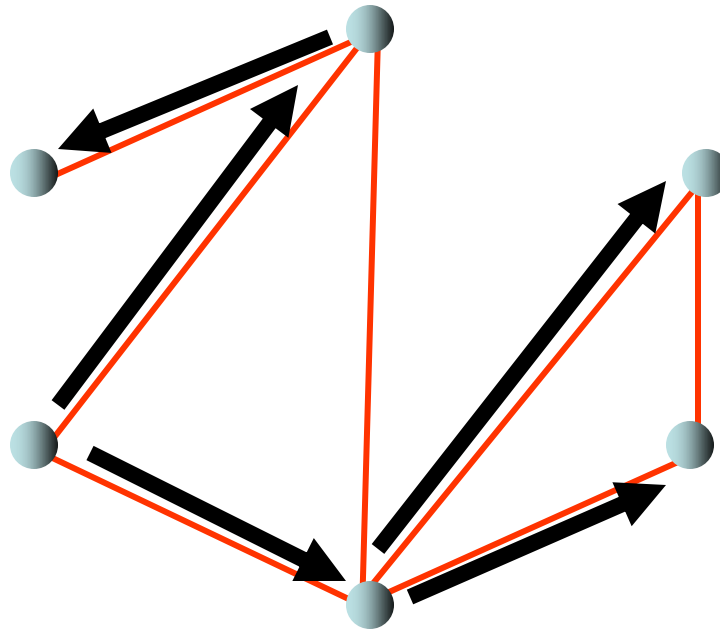


MST? No.

Note: BFS can also be defined wrt weights (shortest path spanning tree)!

Broadcast

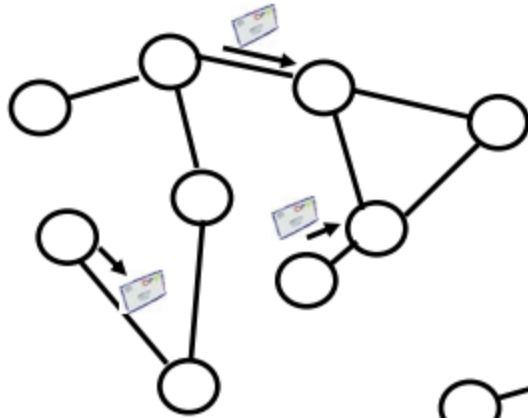
Task: Send one message to all nodes.



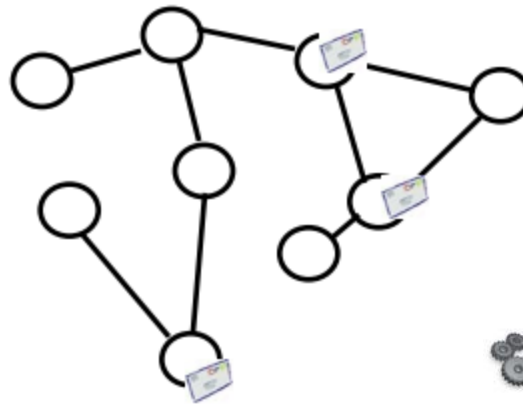
Lower bound for
time and messages?

Recall: Local Algorithm

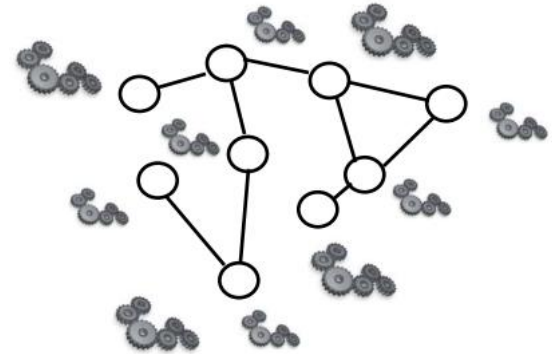
Send...



... receive...



... compute.



Broadcast

Broadcast

Message from one source to all other nodes.

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?

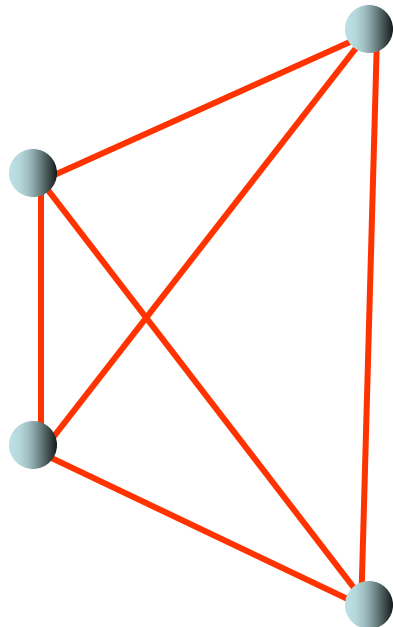
Examples....

Lemma (R, D)

$$R \leq D \leq 2R$$

Where $R=D$?

Complete graph:

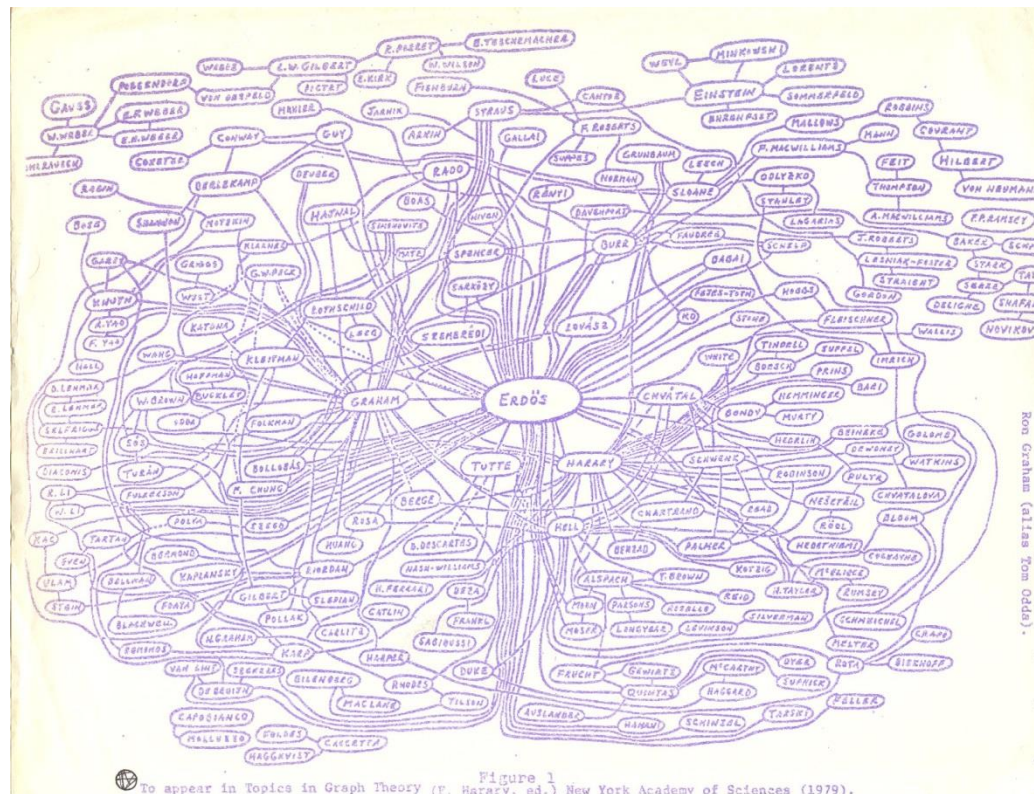


Where $2R=D$?



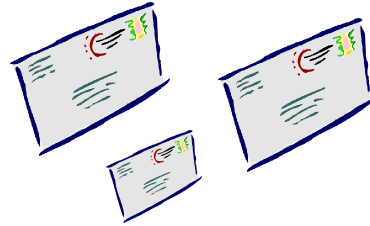
Kevin Bacon, Paul Erdős,

People like to find nodes of **small radius** in a graph! E.g., **movie** collaboration (link = act in same movie) or **science** (link = have paper together)!



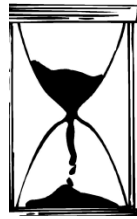
Lower Bound for Broadcast?

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 broadcast with $n-1$ messages and radius time?

Pre-computed **breadth-first spanning tree**...

Broadcast in Clean Networks?

Clean Graph

Nodes do not know topology.

Lower bound for clean networks?

Number of edges: if not every edge is tried, one might miss an entire subgraph!

How to do broadcast in clean network?

Flooding

1. Source sends message to all neighbors.
2. Each other node u when receiving the message for the first time from node v (called **u 's parent**), sends it to all (other) neighbors.
3. Later receptions are discarded.

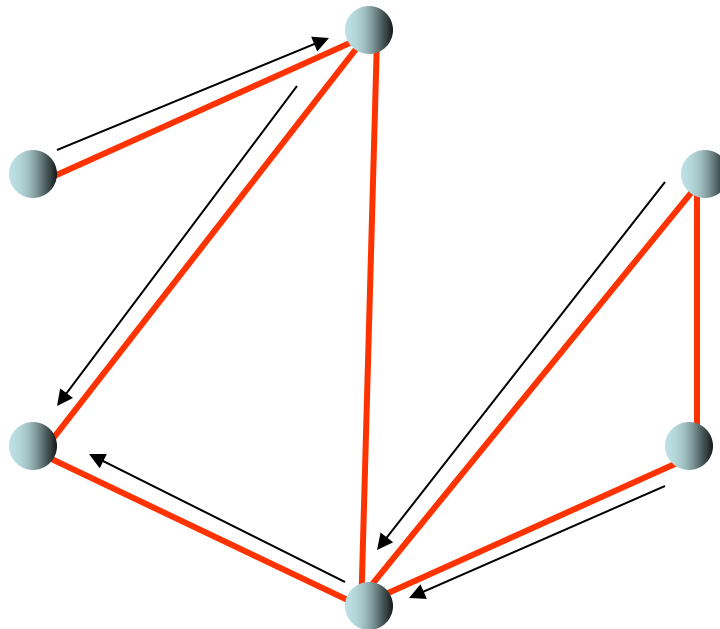
Note that parent relationship defines a **tree**!

In synchronous system, the tree is a **breadth-first search spanning tree**!

Convergecast

Convergecast

Opposite of broadcast: all nodes send message to a given node!



Purpose?

- E.g., for aggregation!
- E.g., find maxID!
- E.g., compute average!
- E.g., aggregate ACKs!

How to compute minimum efficiently?

Aggregation



Echo Algorithm

Echo Algorithm

0. Initiated by the **leaves** (e.g., of tree computed by **flooding algo**)
1. Leave sends message to its **parent**
2. If inner node has received a message *from each child*, it forwards message to parent

Application: convergecast to determine termination. How?

Have sub-trees completed?

Complexities?

Echo on tree, but complexity of flooding to build tree...

How to compute a breadth-first tree?

Flooding gives parent-relationship, but...

... breadth-first only if synchronous.

How to do it in asynchronous distributed system?

Dijkstra (‘link state’) or **Bellman-Ford** (‘distance vector’) style

Remember the ideas?

Bellman-Ford: BGP in the Internet!

Dijkstra: grow on the „**border**“

Bellman-Ford: **distances** (distance vector)...

Asynchronous BFS Tree

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

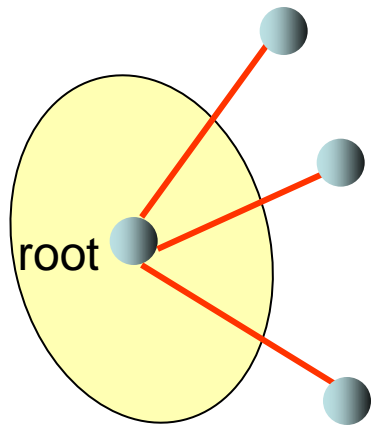
Dijkstra Style

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):

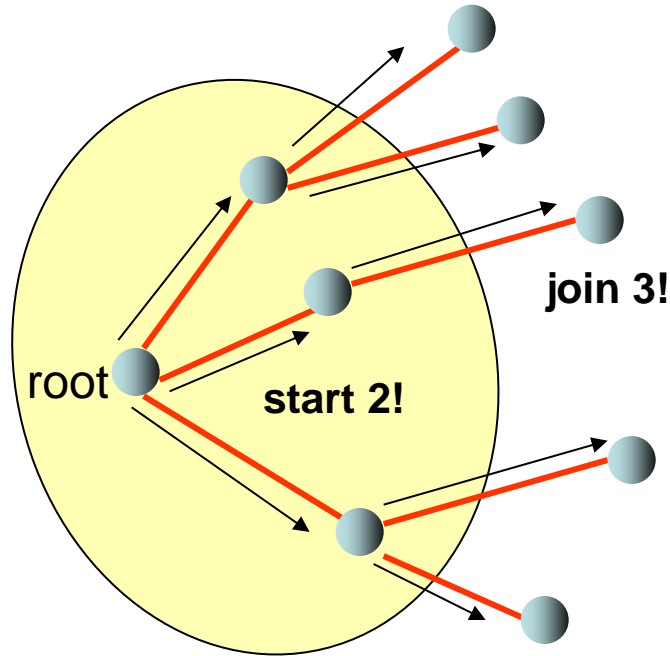
1. Root starts phase p by broadcasting „**start p**“ within T_p
2. A leaf u of T_p (= node discovered only in last phase) sends „**join p+1**“ to all **quiet neighbors** v (u has not talked to v yet)
3. Node v hearing „join“ for first time sends back „**ACK**“: it becomes leaf of tree T_{p+1} ; otherwise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

Asynchronous BFS Tree: Idea



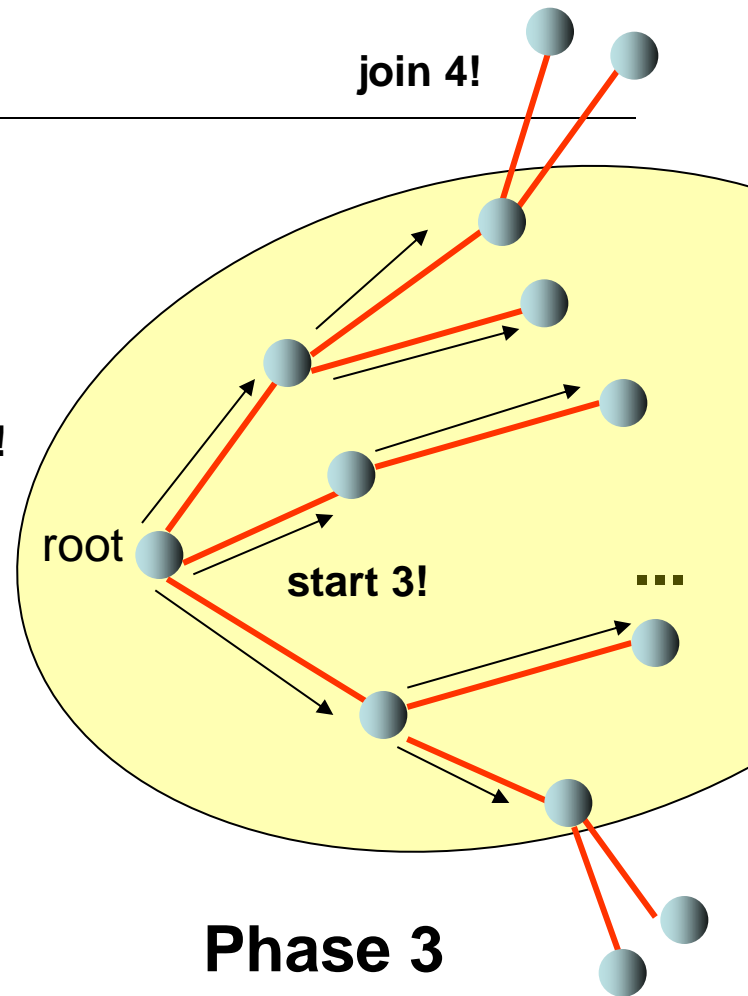
Phase 1

Wait until all next hops explored...



Phase 2

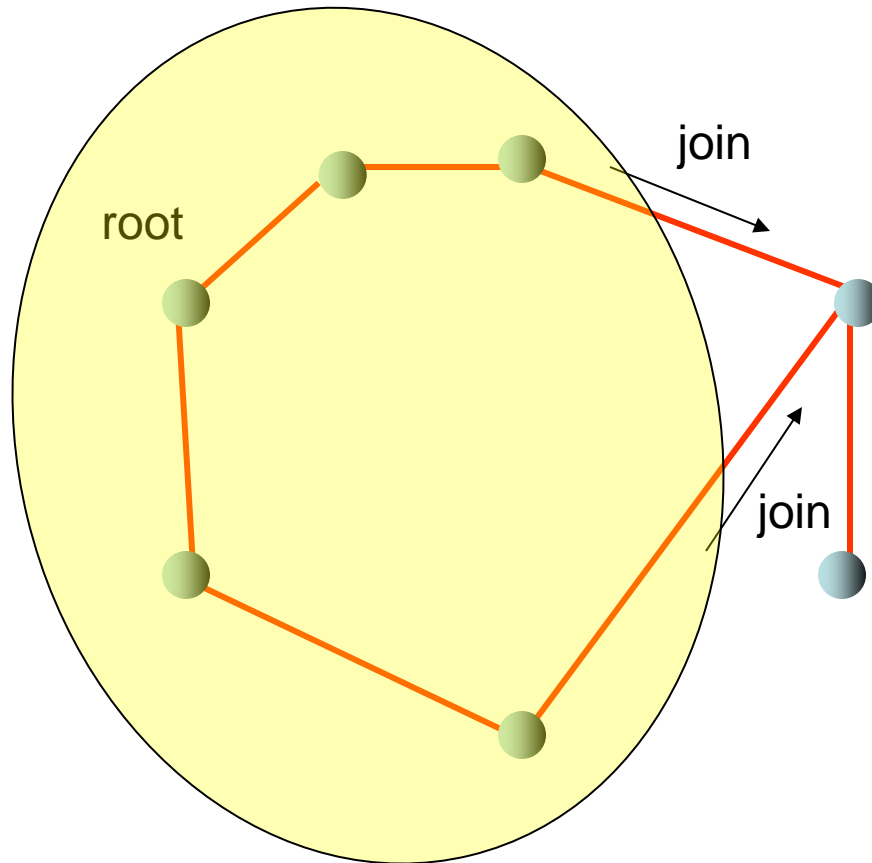
Wait until all next hops explored...



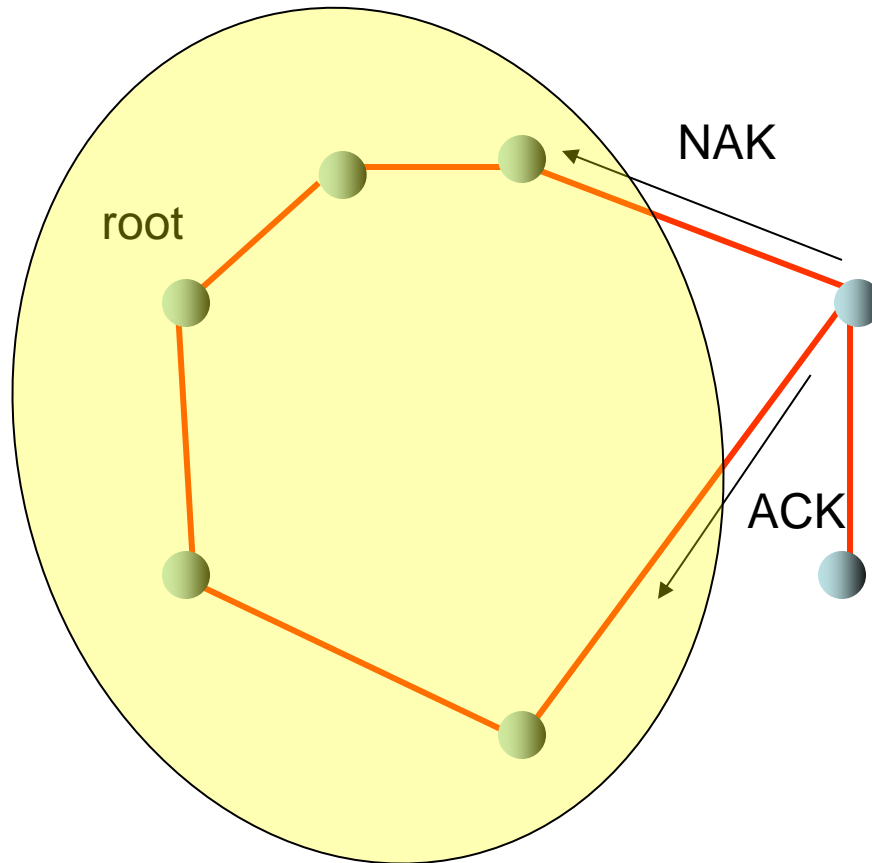
Phase 3

Wait until all next hops explored...

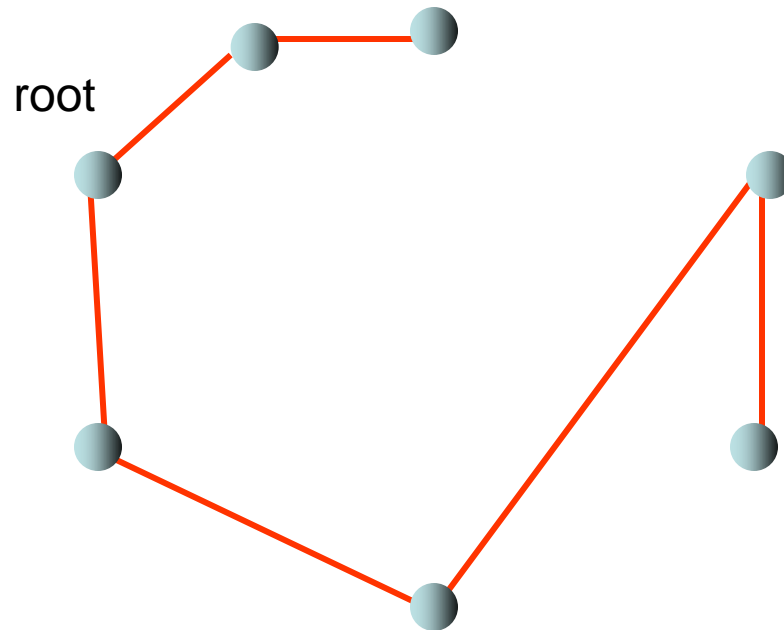
Asynchronous BFS Tree



Asynchronous BFS Tree



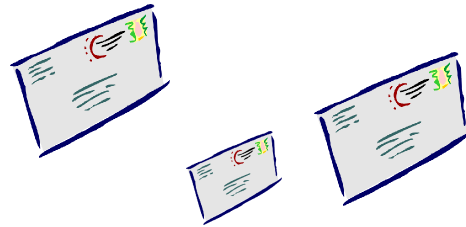
Asynchronous BFS Tree



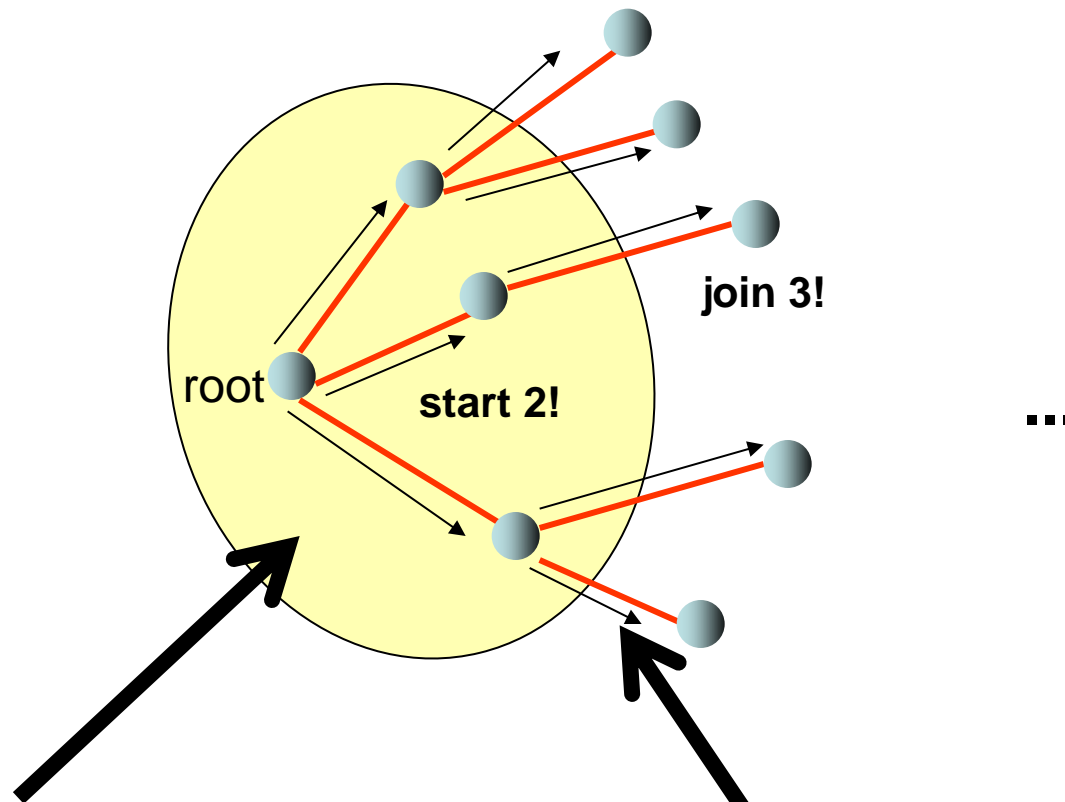
Time Complexity?



Message Complexity?



Asynchronous BFS Tree: Idea



Reuse shortest path infrastructure here!
Time $O(D)$ per phase,
 $O(D)=O(n)$ messages.

At most two messages per edge **overall**: one «join» and one «ACK/NAK»: $O(m)$

Time Complexity?



$O(D^2)$ where D is diameter of graph...

... as convergecast costs $O(D)$, and we have D phases.

Message Complexity?



$O(m+nD)$ where m is number of edges, n is number of nodes.

Because: **Convergecast** has cost $O(n)$, one per link in tree, so over all phases $O(nD)$. On each edge, there are **at most two join** messages (both directions), and there is at most an ACK/NAK answer, so $+m$...

Alternative algo?

Dijkstra Algorithm



Time Complexity?

$O(D^2)$ where D is diameter of graph...

... as convergecast costs $O(D)$, and we have D phases.

Can we do it faster?! Without «back and forth»?

Asynchronous BFS Tree

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

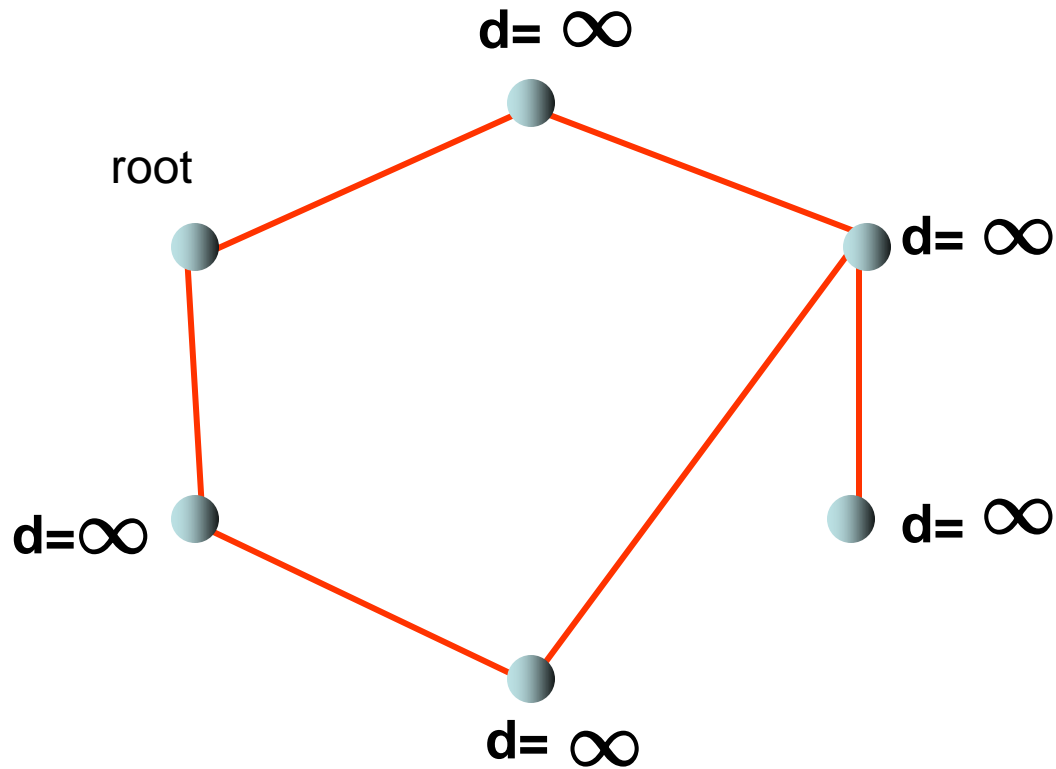
Bellman-Ford Style

Each node u stores d_u , the distance from u to the root.
Initially, $d_{\text{root}}=0$ and all other distances are ∞ . 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

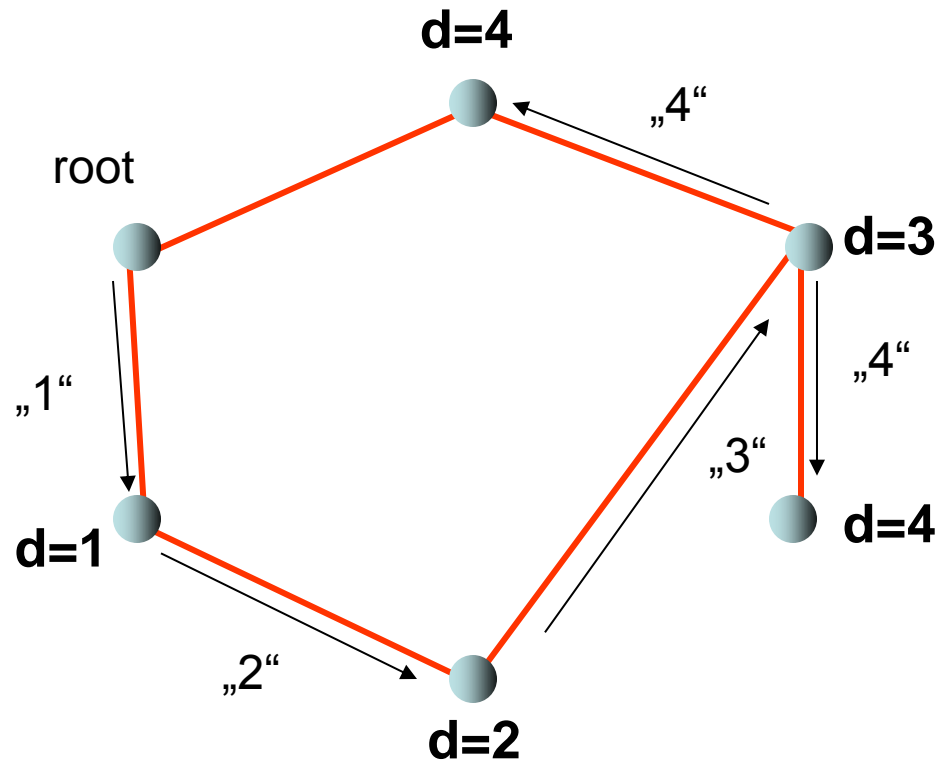
Asynchronous BFS Tree

Initially:



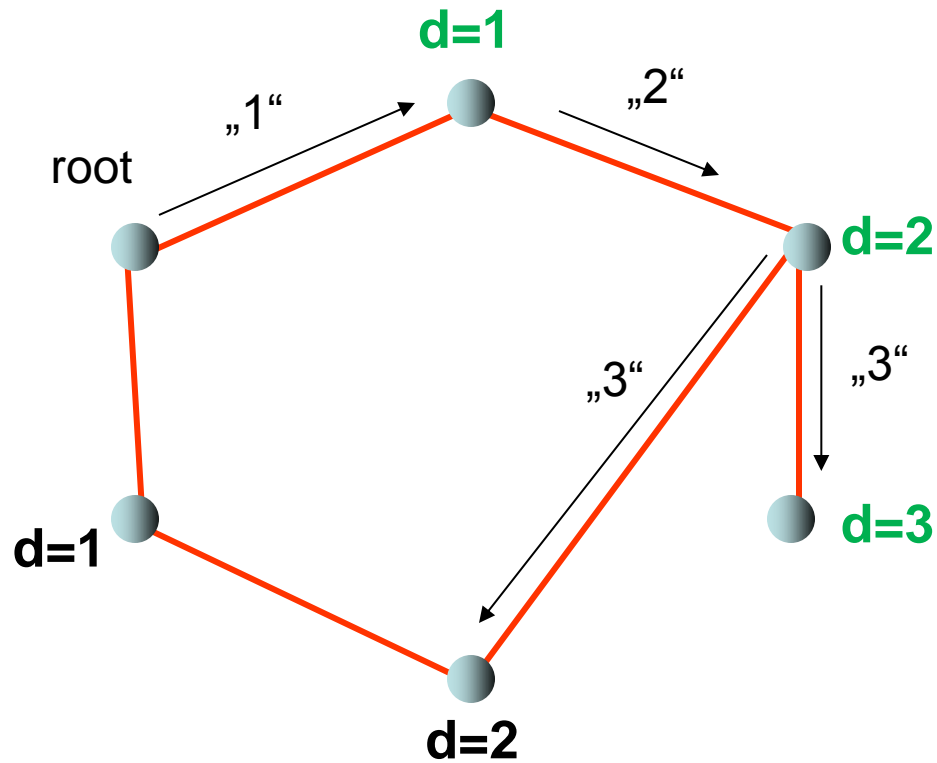
Asynchronous BFS Tree

Fast transmission:



Asynchronous BFS Tree

Slow transmission:





Time Complexity?

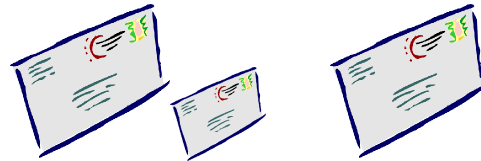
$O(D)$ where D is diameter of graph. 😊

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...

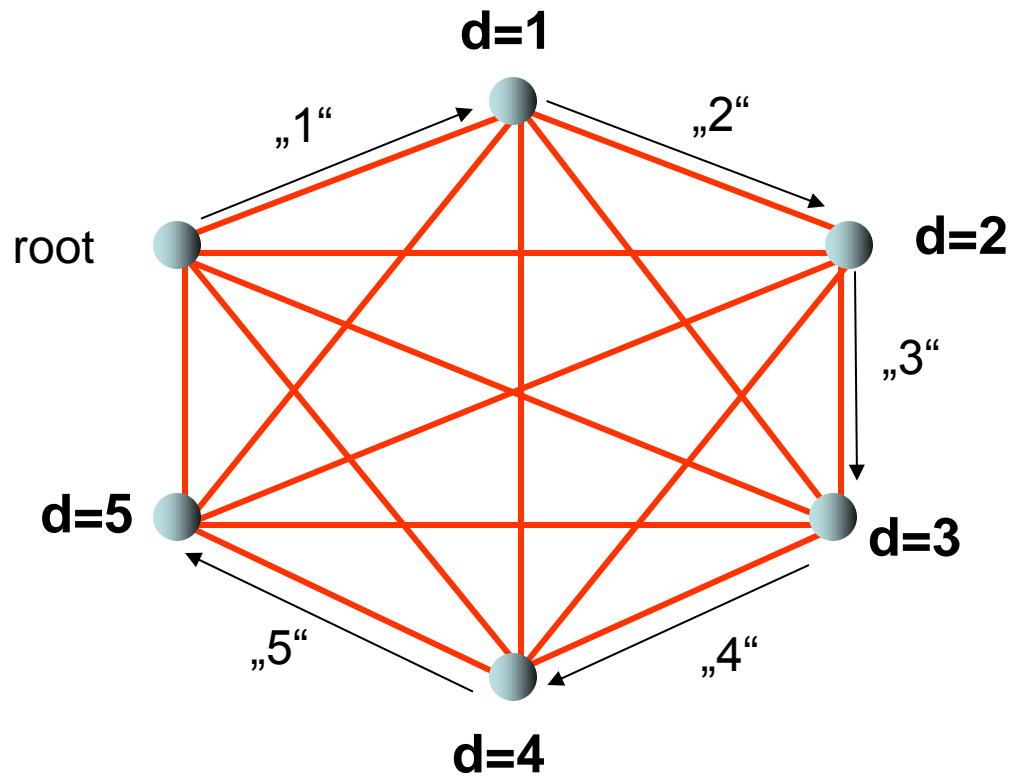
Message Complexity?



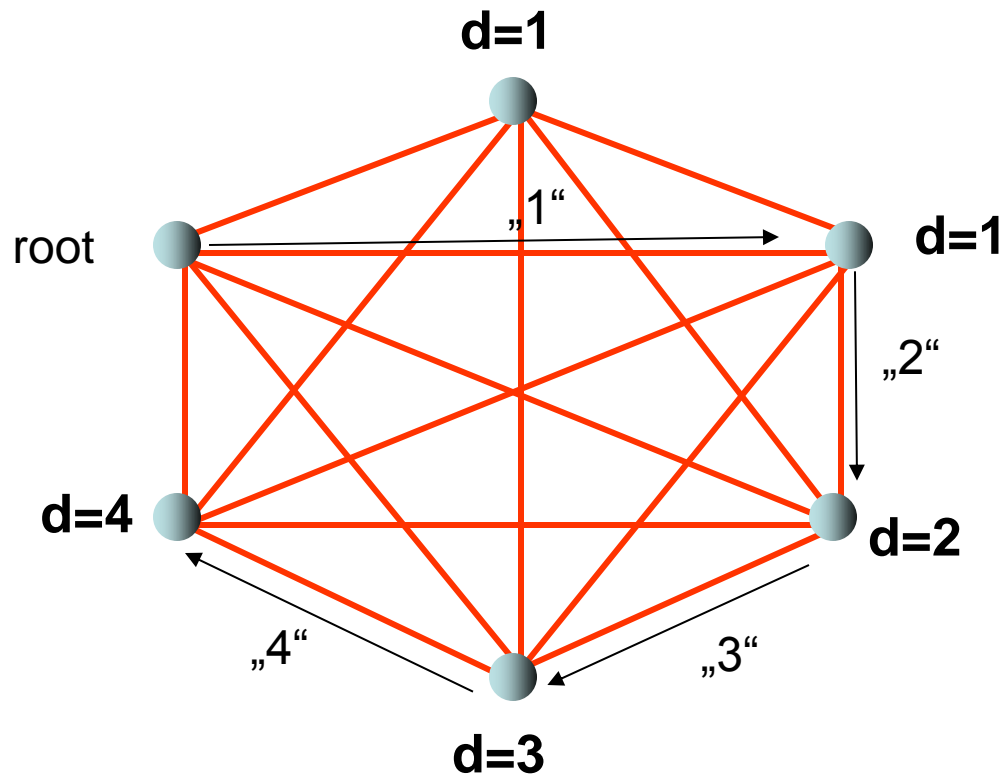
$O(mn)$ where m is number of edges, n is number of nodes. 😞

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

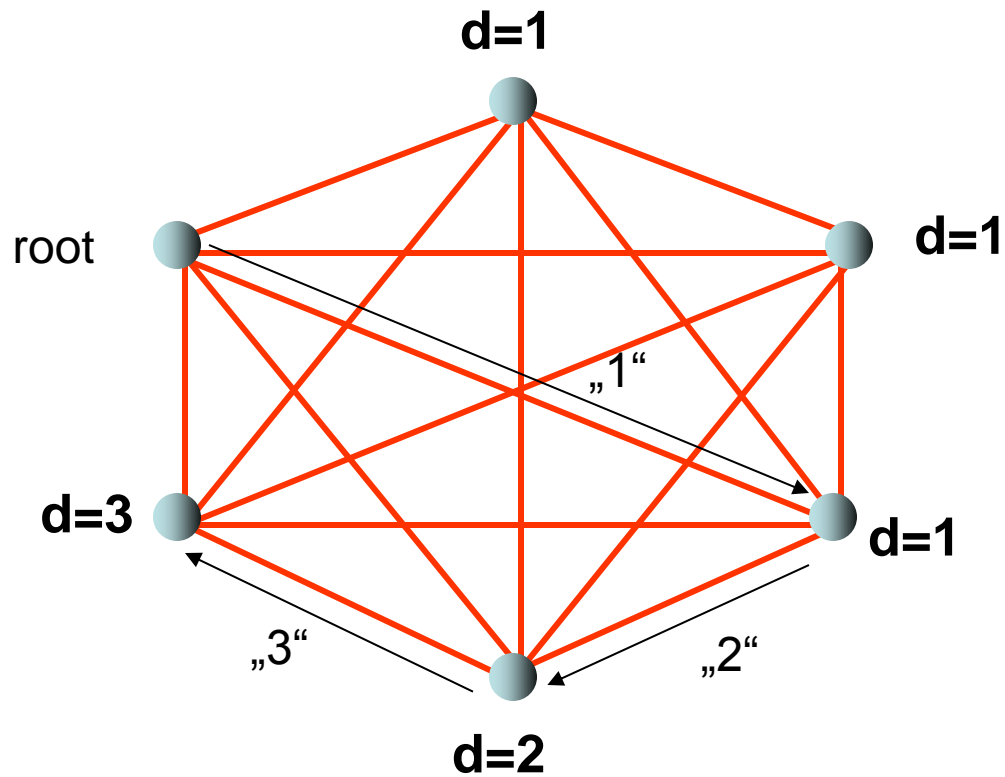
Bellman-Ford with Many Messages



Bellman-Ford with Many Messages



Bellman-Ford with Many Messages



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**)

MST Construction

MST

Tree with edges of minimal total weight.

Another spanning tree? Why?

For **weighted graphs**: tree of minimal costs...
useful building block (approximation algorithms etc.)!

Assume all links have different weights. So...

MST is **unique.**

How to compute in a **distributed manner
(synchronously...)?! How to do it classically?**

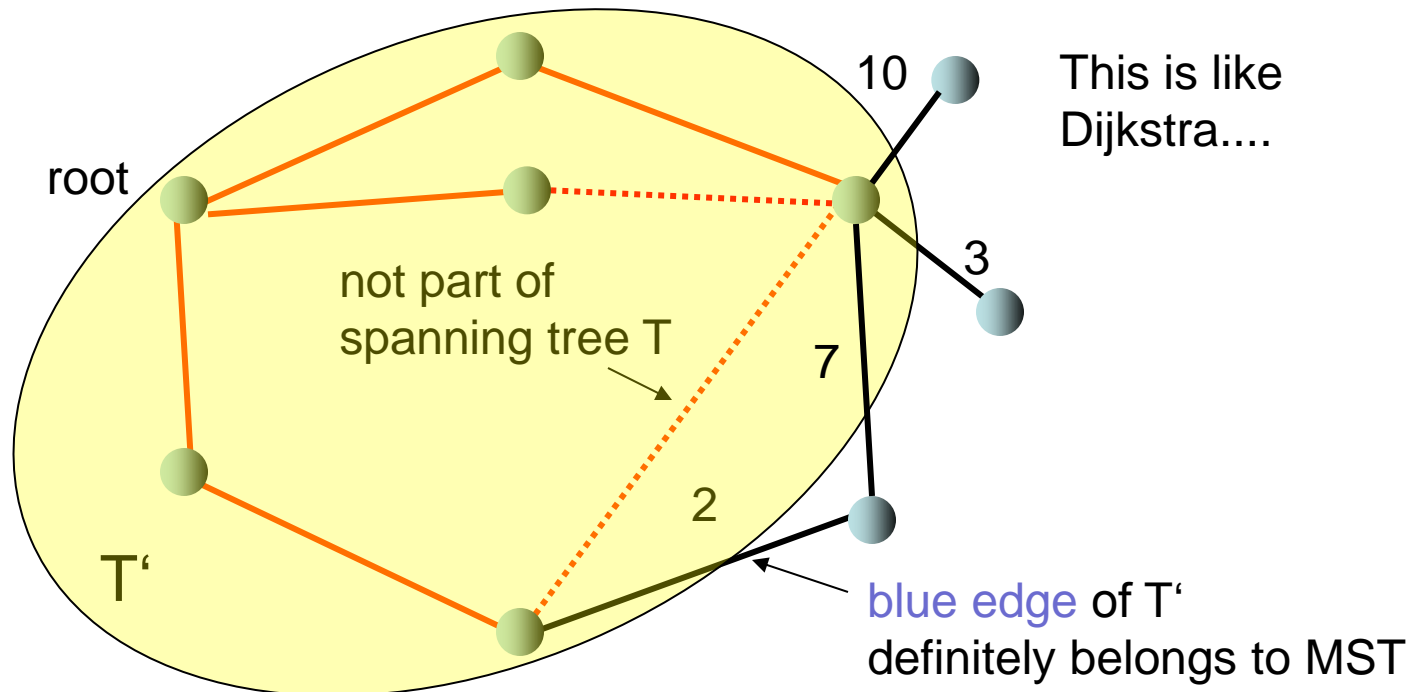
Kruskal (lightest non-cycle edge), Prim (lightest outward edge), ...

Guess: Faster or slower than BFS tree?

Idea

Blue Edge

Let T be a MST and T' a subgraph of T .
Edge $e=(u,v)$ is outgoing edge if $u \in T'$ but v is not.
The outgoing edge of minimal weight is called *blue edge*.



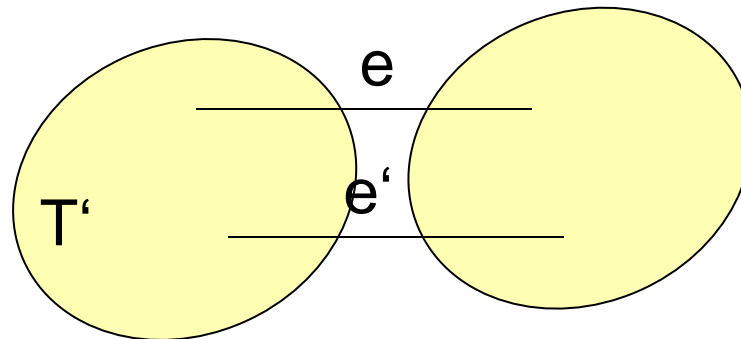
Lemma

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

Proof idea?

By contradiction! Suppose there is an other edge e' connecting T' to the rest of T . If we add the blue edge e and remove e' from the resulting cycle, we still have a spanning tree, but **with lower cost**...

T :



So what?!

Ideas:

- Grow MST component by learning blue edge!
- But do many **fragments in parallel!**
- Each component managed by its root (the «leader»)

Distributed Kruskal

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

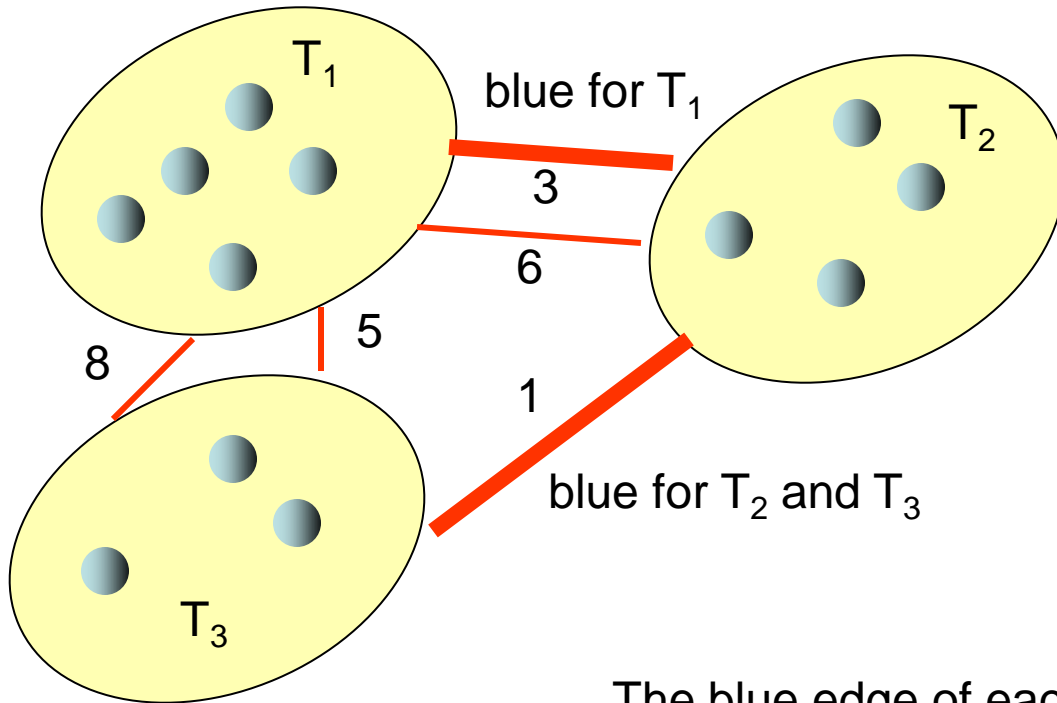
Gallager-Humblet-Spira

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

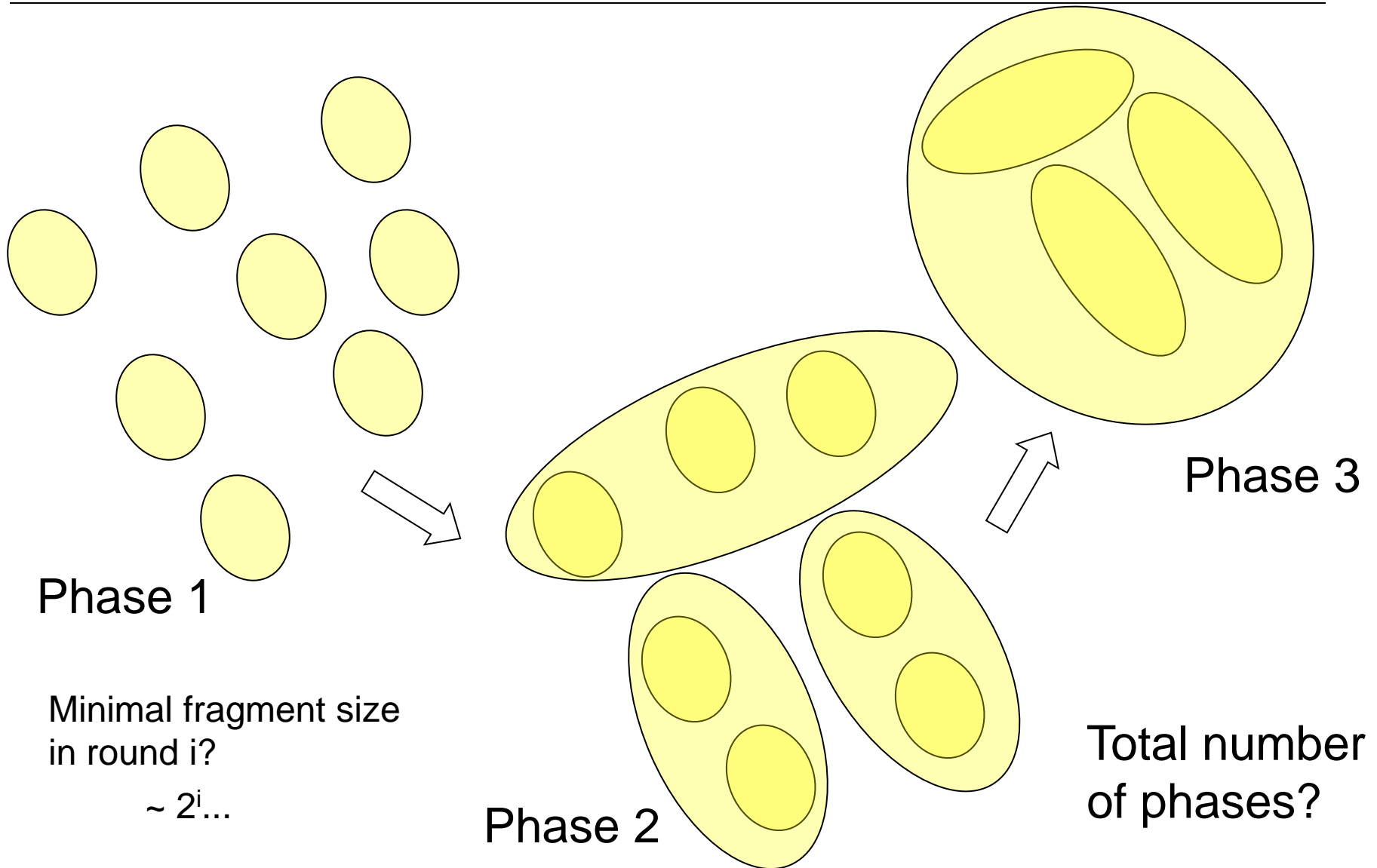
Idea: Merge Components



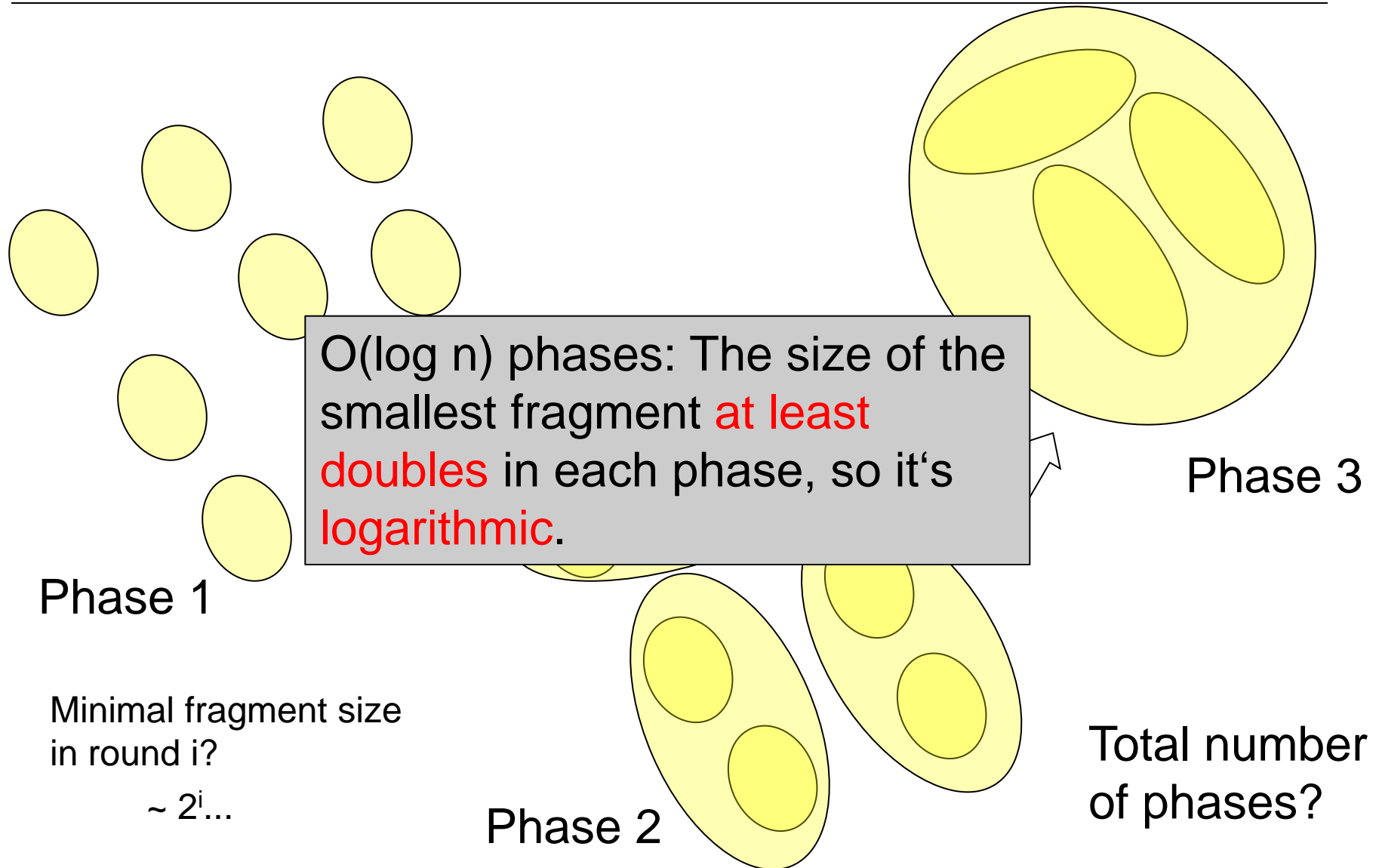
The blue edge of each fragment can be taken for sure: cycles not possible!
(Blue edge lemma!)

So we can do it **in parallel!**

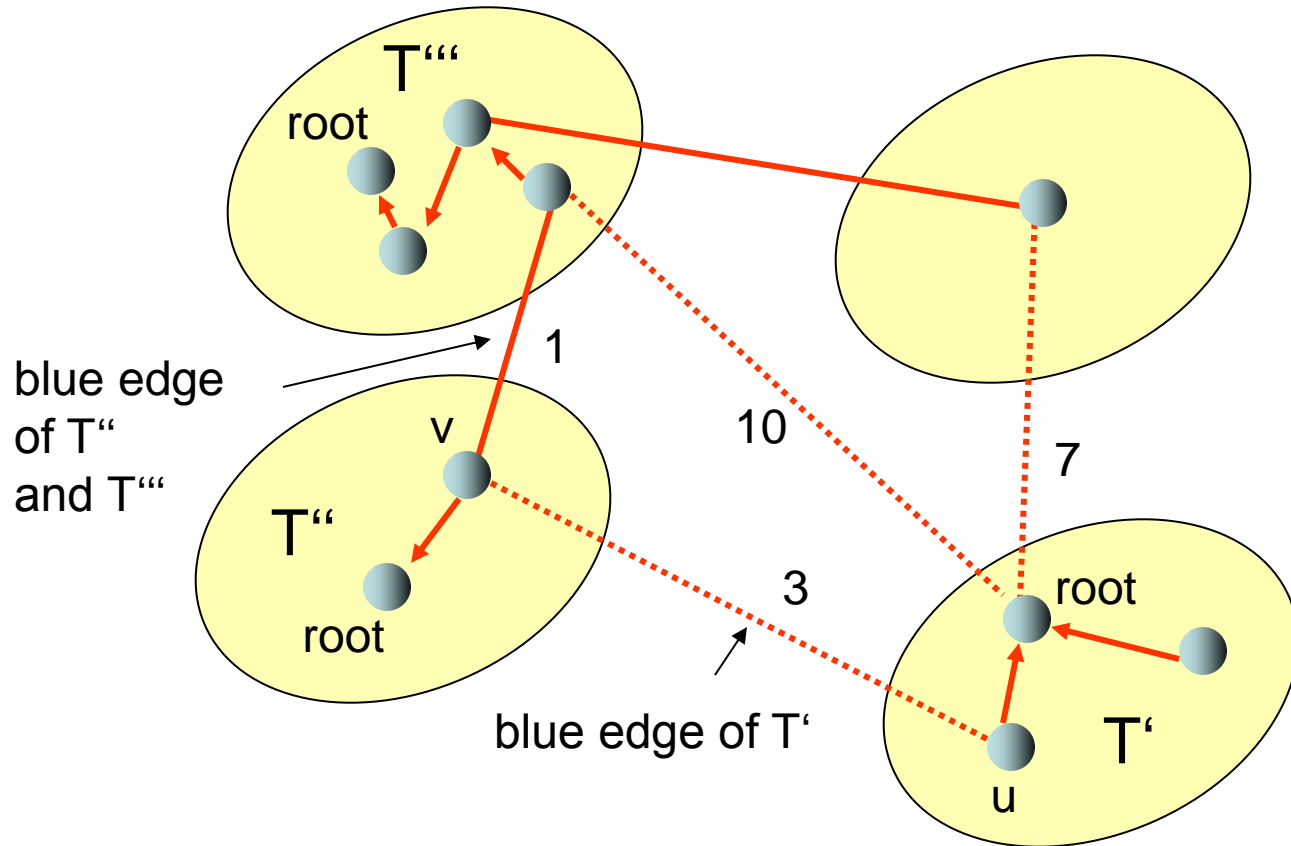
Idea: Components Grow Quickly



Idea: Components Grow Quickly

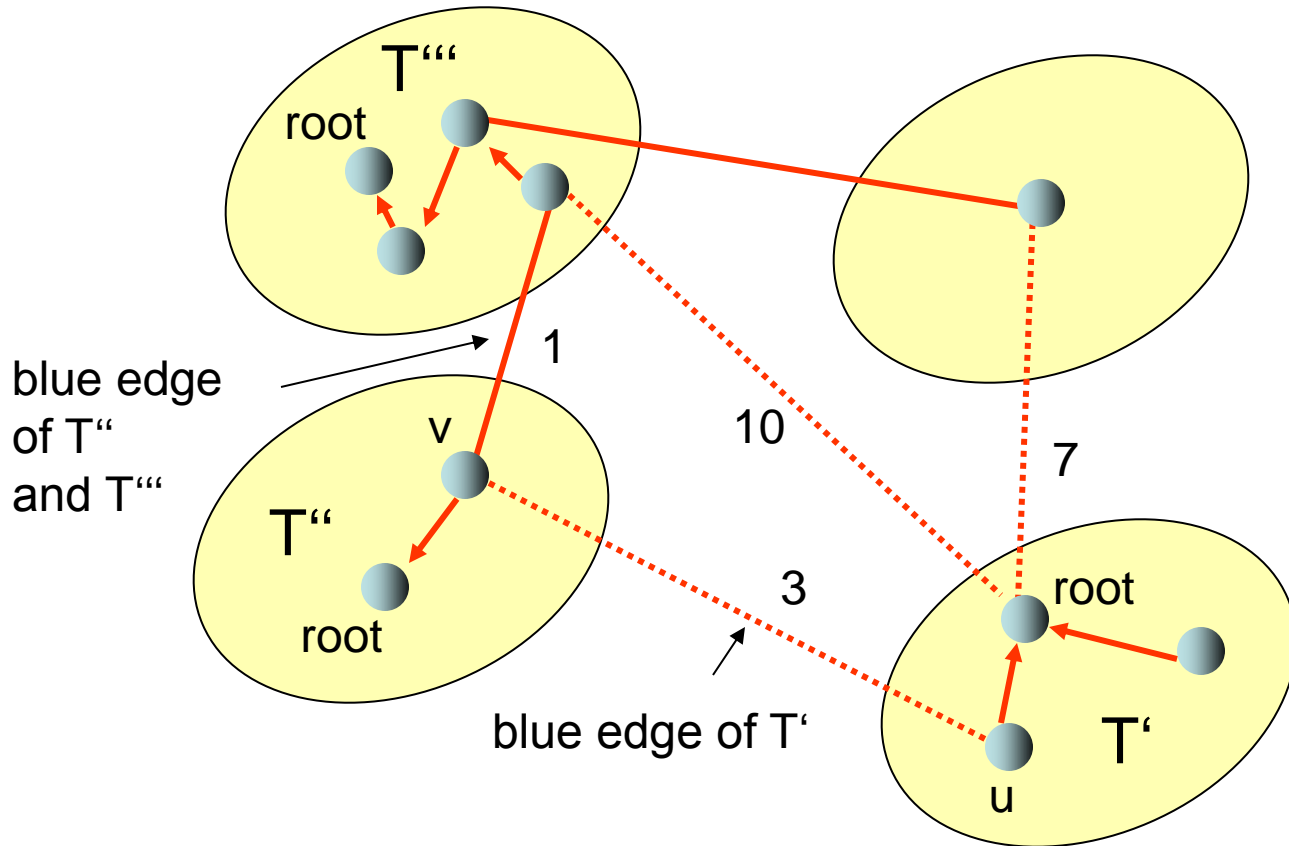


Idea: Agree on a New Root («Leader»)



Who becomes overall leader of T and T' ?
Make trees directed...

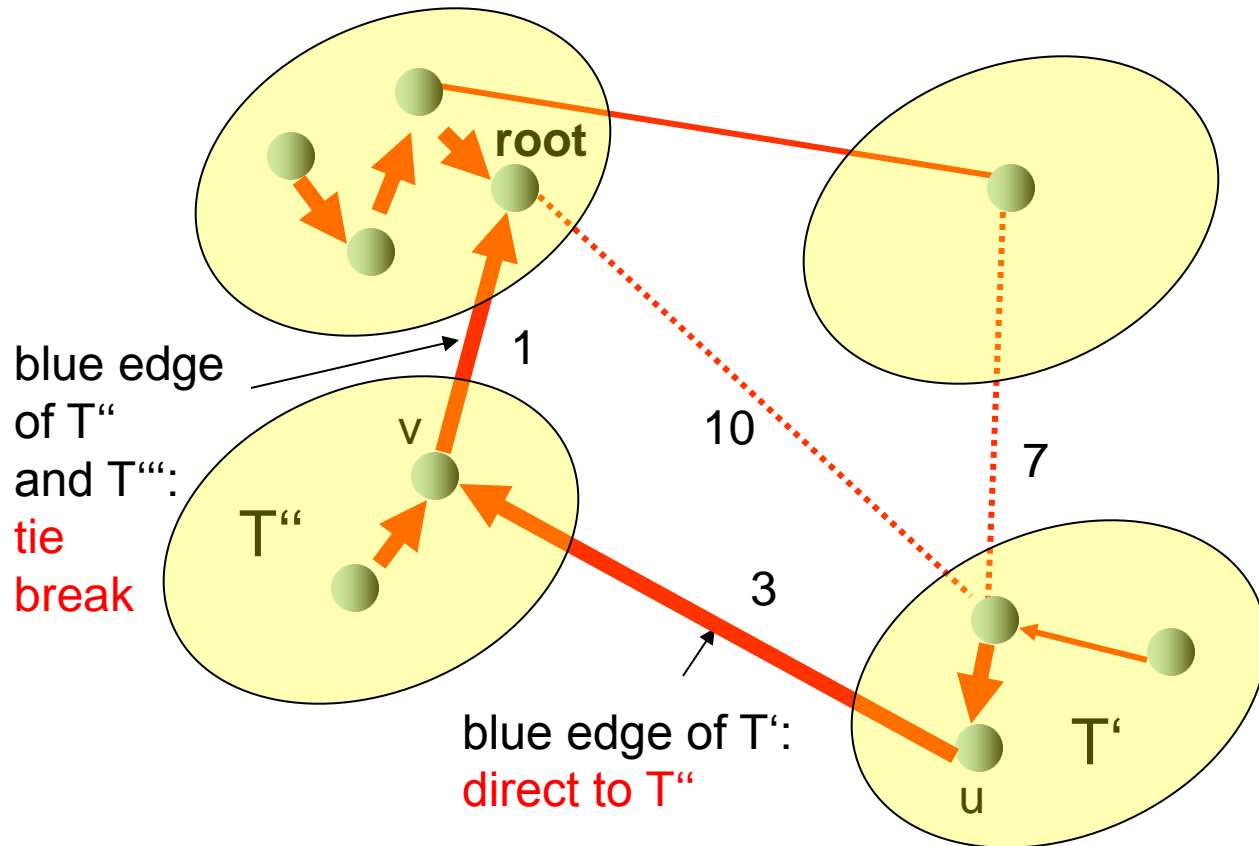
Idea: Agree on a New Root («Leader»)



All trees rooted! How to merge on **blue edge (u,v)**?

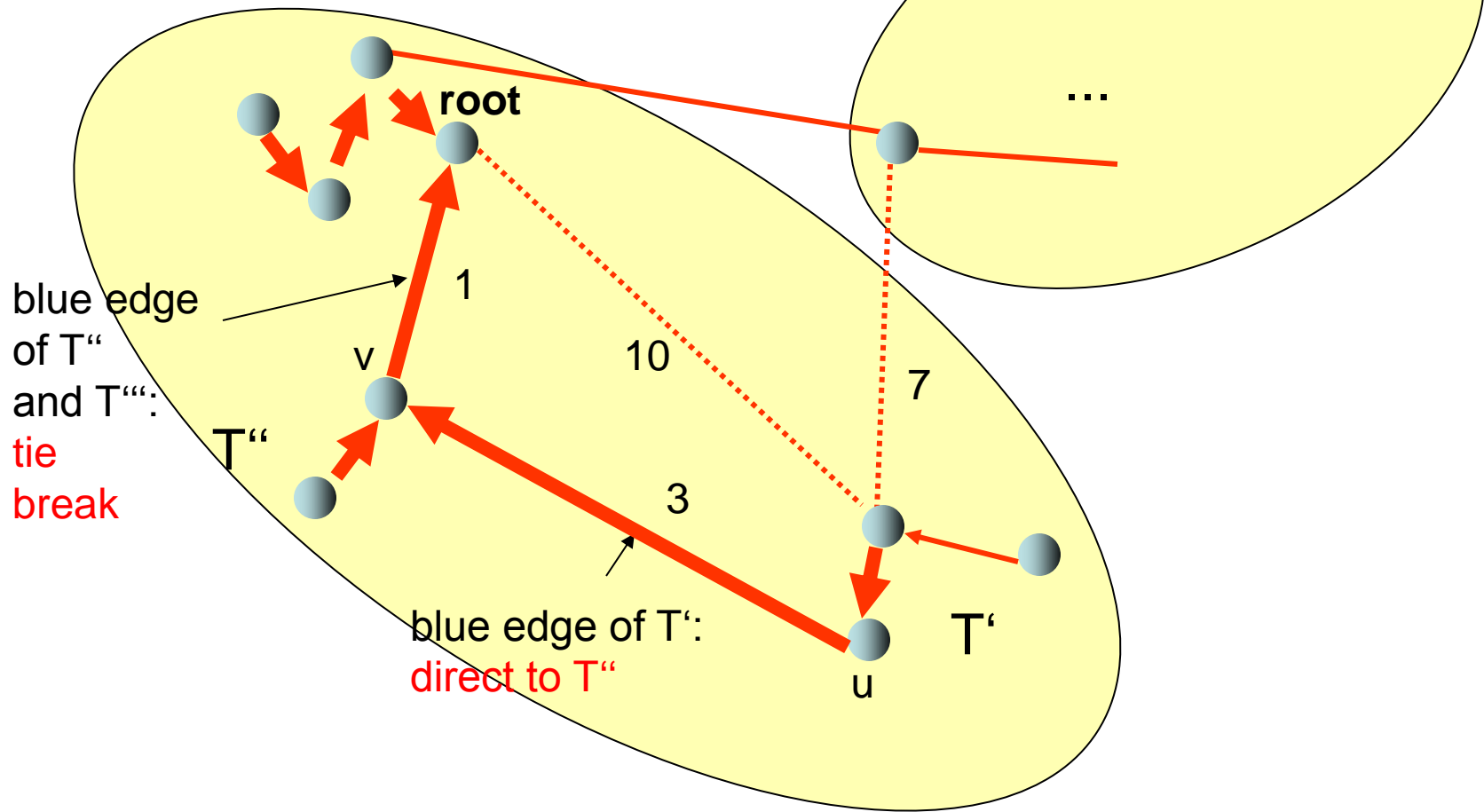
1. Invert path from root to u (u is temporary root)
2. If u sent merge request over blue edge, v becomes root; if u and v sent message over blue edge: **point blue edge to smaller ID**

Idea: Agree on a New Root («Leader»)



New directed tree with new root! 😊
 T''' connects somewhere else...

Idea: Agree on a New Root («Leader»)

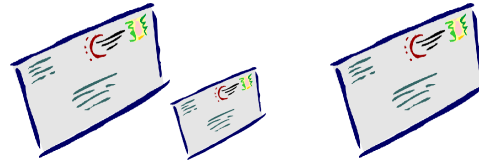


Merged fragments!

Time Complexity?



Message Complexity?

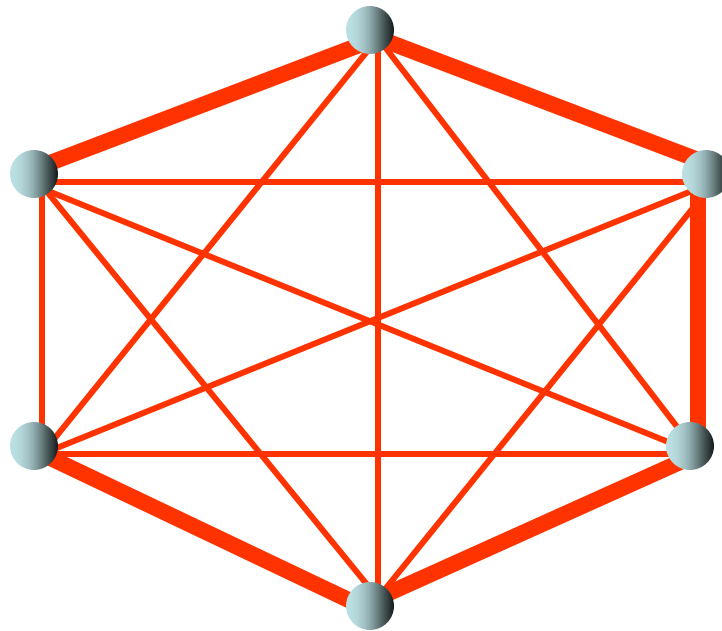


Each phase mainly consists of two convergecasts, so $O(D)$ time and $O(n)$ messages per phase?

Analysis

Careful:

- Convergecast on MST, not on BFS tree
- MST may be larger than diameter of graph!



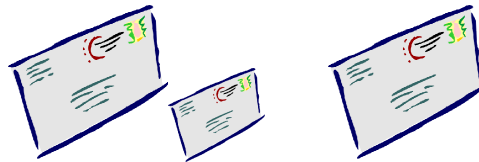
$O(n)$ time for convergecast, and not $O(1)$...



Time Complexity?

The size of the smallest fragment **at least doubles** in each phase, so it's **logarithmic**.

$O(n \log n)$ 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 and $O(n)$ messages**. In order to learn fragment IDs of neighbors, $O(m)$ messages are needed (in beginning of each phase: constant time).

Yes, we can do better. 😊

- GHS solves leader election in general graphs! How?

Last surviving root...

- Some details left out, e.g.:

if fragment larger than other, may need to wait to find out whether neighbor also wants to merge over this edge: could do in phases (like Dijkstra BFS)

Literature for further reading:

- Peleg's book

End of lecture