



# Chapter 2

# Broadcast, Convergecast, and Spanning Trees

**Distributed Systems**

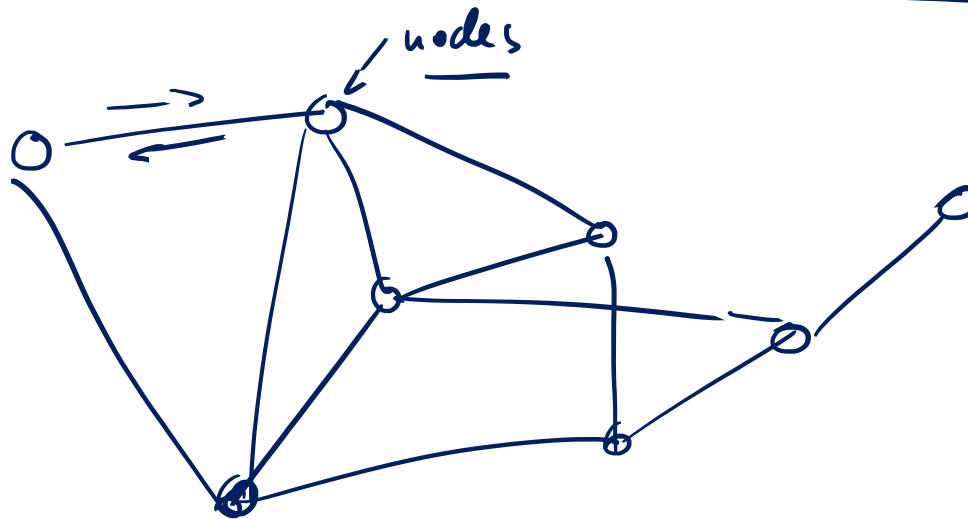
**SS 2019**

**Fabian Kuhn**

# Message Passing in Arbitrary Topologies

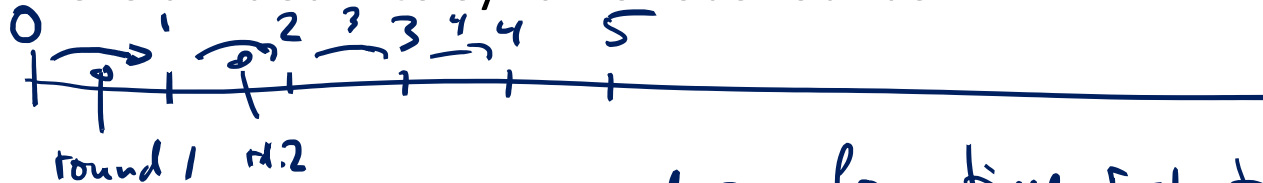
## Assumption for this chapter:

- Network: message passing system with arbitrary topology
- network topology is given by an undirected **graph  $G = (V, E)$**



# Synchronous Message Passing

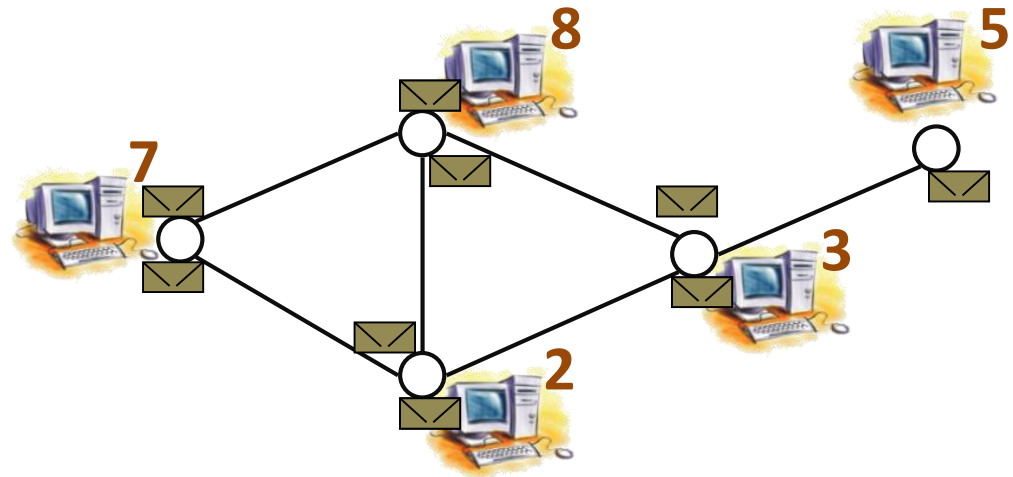
- Time is divided into synchronous rounds



round  $r$ : from time  $r-1$  to time  $r$

## In each synchronous round:

1. Each node does some internal computation
2. Send a message to each neighbor
3. Receive message from each neighbor



**time complexity = number of rounds**

# Asynchronous Message Passing

In this chapter: No failures, but **asynchrony**

## Asynchronous message passing:

- messages are always delivered in finite time
  - cf.: finite time  $\rightarrow$  admissible schedule
- message delays are completely unpredictable
- algorithms are **event-based**:

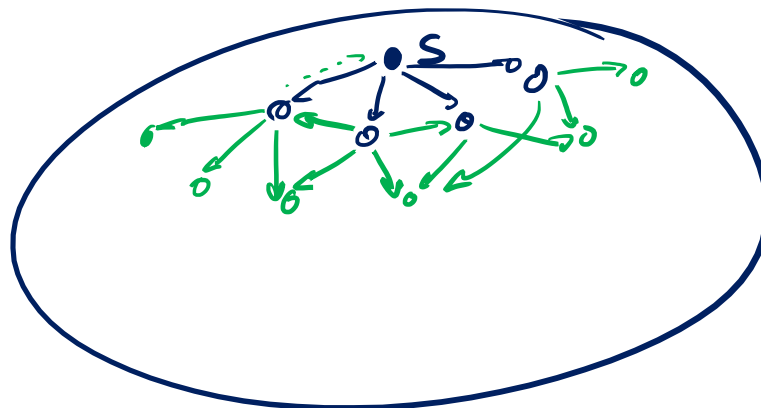
**upon receiving message from neighbor ..., do**  
**some local computation steps**  
**send message(s) to neighbor(s) ...**

# Broadcast

- Simple, basic communication problem

## Problem Description:

- A source node  $s$  needs to broadcast a message  $M$  to all nodes of the system (network)
- Each node has a unique ID
- Initially, each node knows the IDs of its neighbors
  - or it can count / distinguish its neighbors by individual communication ports to the pairwise communication links



# Flooding

- One of the simplest distributed (network) algorithms

## Basic idea:

- When receiving  $M$  for the first time, forward to all neighbors

## Algorithm:

- Source node  $s$ :

**initially do**

send  $M$  to all neighbors

- All other nodes  $u$ :

**upon receiving  $M$  from some neighbor  $v$  (for the first time)**

**if  $M$  has not been received before then**

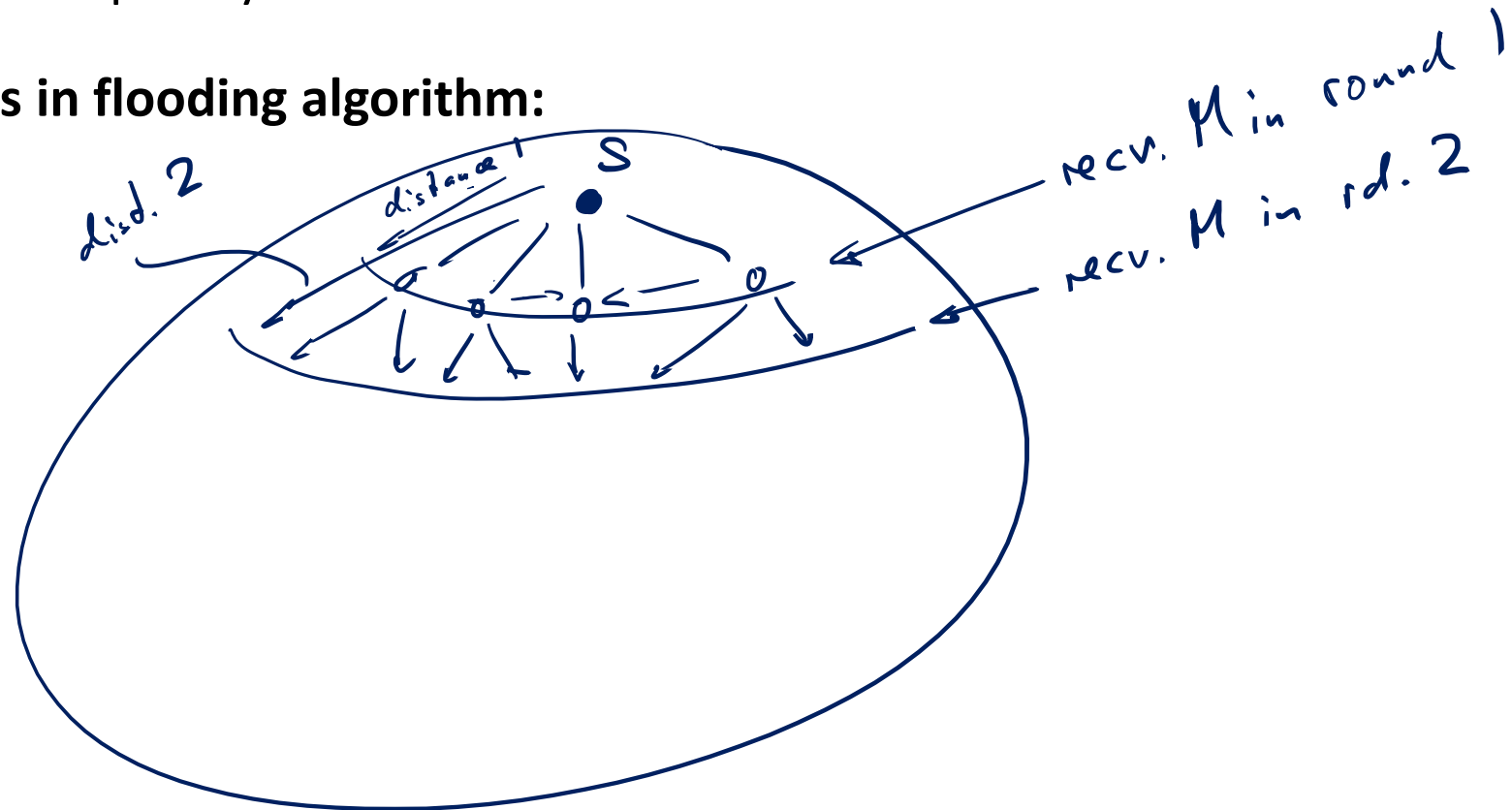
send  $M$  to all neighbors except  $v$

# Flooding in Synchronous Systems

## Synchronous systems:

- time divided into synchronous rounds, msg. delay = 1 round
- time complexity: number of rounds

## Progress in flooding algorithm:



# Flooding in Synchronous Systems

## Synchronous systems:

- time divided into synchronous rounds, msg. delay = 1 round
- time complexity: number of rounds

## Progress in flooding algorithm:

- after 1 round:
  - all neighbors of  $s$  know  $M$
  - nodes at distance  $\geq 2$  from  $s$  do not know  $M$
- after 2 rounds:
  - exactly nodes at distance  $\leq 2$  from  $s$  know  $M$
- ...
- after  $r$  rounds:
  - exactly nodes at distance  $\leq r$  from  $s$  know  $M$

total time:  
max. distance  
of any node from  $s$



# Flooding in Synchronous Systems

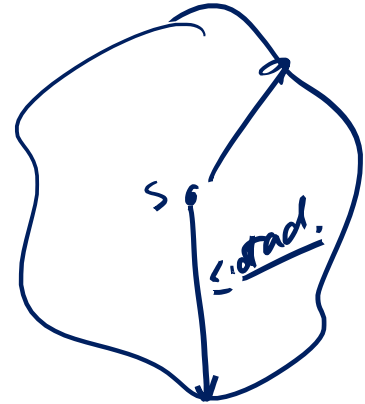
**Radius:** (Graph  $G = (V, E)$ )

- Given a node  $s \in V$ , **radius of  $s$  in  $G$ :**

$$rad(G, s) := \max_{v \in V} dist_G(s, v)$$

- radius of  $G$ :

$$rad(G) := \min_{s \in V} rad(G, s)$$



**Diameter of  $G$ :**

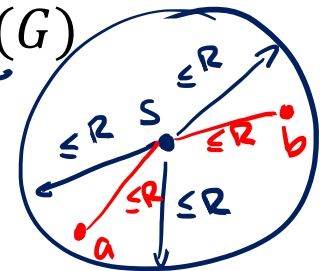
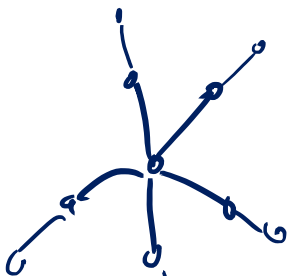
$$diam(G) := \max_{u, v \in V} dist_G(u, v) = \max_{s \in V} rad(G, s)$$

$$\frac{diam}{2} \leq rad \leq diam$$

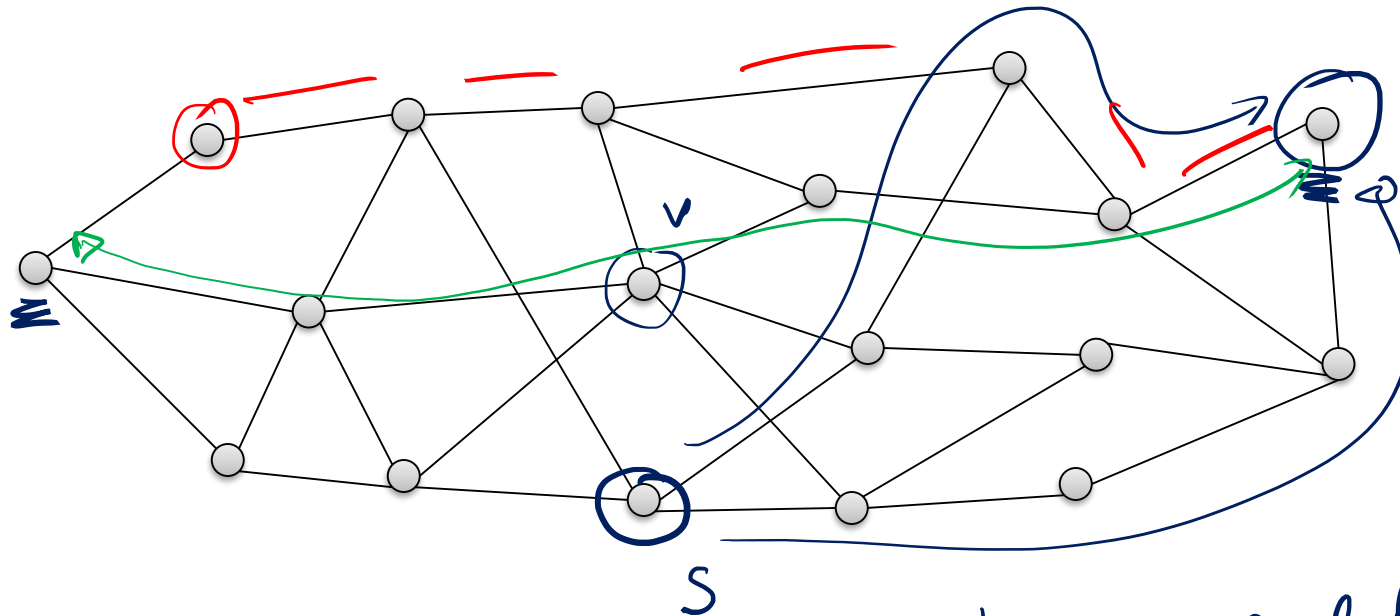
**Time complexity of flooding** in synchronous systems:  **$rad(G, s)$**

$$\frac{diam(G)}{2} \leq \underbrace{rad(G)}_R \leq rad(G, s) \leq \underbrace{diam(G)}_D$$

$$D \leq 2R$$



# Radius and Diameter



$$\text{rad}(G, s) = 4$$

$$\text{diam}(G) = \color{green}{5}$$

$$\text{rad}(G) = \text{rad}(G, v) = 3$$

time compl. of flooding

$$\text{rad}(G, s) \leq \text{diam}(G)$$

$$O(\text{diam}(G))$$

# Asynchronous Time Complexity

- Time complexity of flooding in asynchronous systems?
- How is time complexity in asynchronous systems defined?

## Assumptions:

- Message delays, time for local computations are arbitrary
  - Algorithms cannot use any timing assumptions!
- **For analysis:**
  - message delays  $\leq$  1 time unit
  - local computations take 0 time

## Determine asynchronous time complexity:

1. determine running time of a given execution
2. asynch. time complexity = max. running time of any exec.

## Running time of an execution:

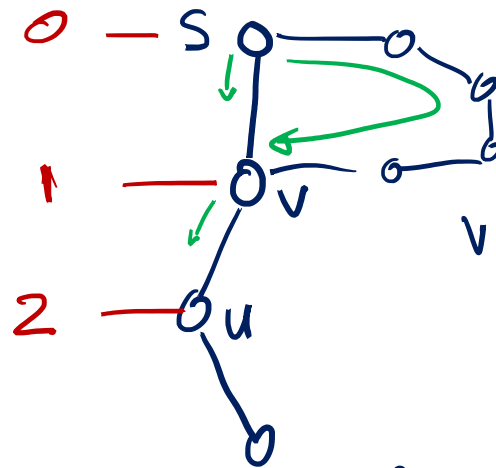
- assign times to send and receive events such that
  - order of all events remains unchanged
  - local computations take 0 time
  - message delays are at most 1 time unit
  - first send event is at time 0
  - time of last event is maximized
- essentially: normalize message delays such that the maximum delay is 1 time unit

## **Definition Asynchronous Time Complexity:**

**Total time of a worst-case execution in which local computations take time 0 and all message delays are at most 1 time unit.**

# Flooding in Asynchronous Systems

**Theorem:** The time complexity of flooding from a source  $s$  in an asynchronous network  $G$  is  $rad(G, s)$ .



$v$  recr.  $M$  at the latest at time 1

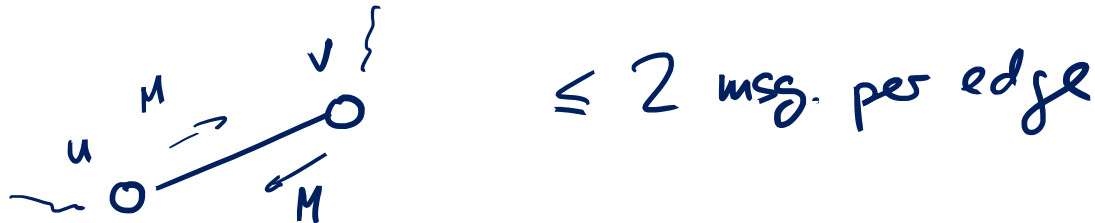
generally:

node  $v$  is at dist.  $r$  from  $s$ ,  
then  $v$  receives  $M$  at the latest  
at time  $r$

# Message Complexity

**Message Complexity:** Total number of messages sent (over all nodes)

What is the message complexity of flooding?



**Theorem:** The message complexity of flooding is  $O(|E|)$ .

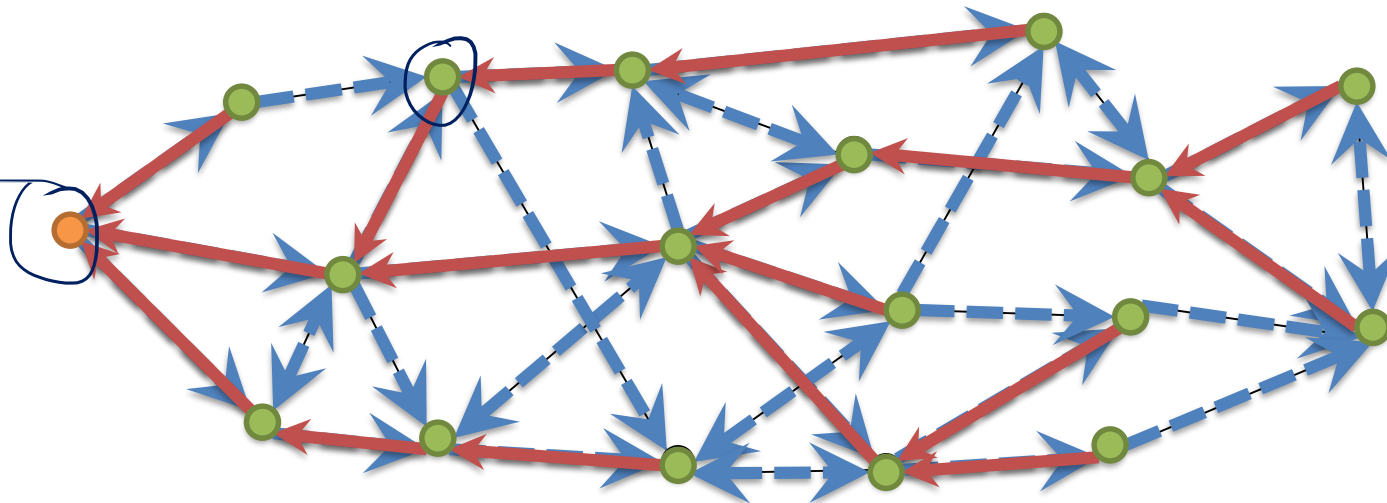
– on graph  $G = (V, E)$

# Flooding Spanning Tree

- The flooding algorithm can be used to compute a spanning tree of the network.

## Idea:

- Source  $s$  is the root of the tree
- For all other nodes, neighbor from which  $M$  is received first is the parent node.



# Flooding Spanning Tree Algorithm

**Source node  $s$ :**

**initially do**

```
parent :=  $\perp$  //  $s$  is the root
send  $M$  to all neighbors
```

**Non-source node  $u$ :**

**upon receiving  $M$  from some neighbor  $v$**

**if  $M$  has not been received before then**

```
parent :=  $v$ 
send  $M$  to all neighbors except  $v$ 
```



# Spanning Tree: Synchronous Systems

- In tree: distance of  $v$  to root = round in which  $v$  is reached
- In synchronous systems, a node  $v$  are reached in round  $r$  if and only if  $dist_G(s, v) = r$

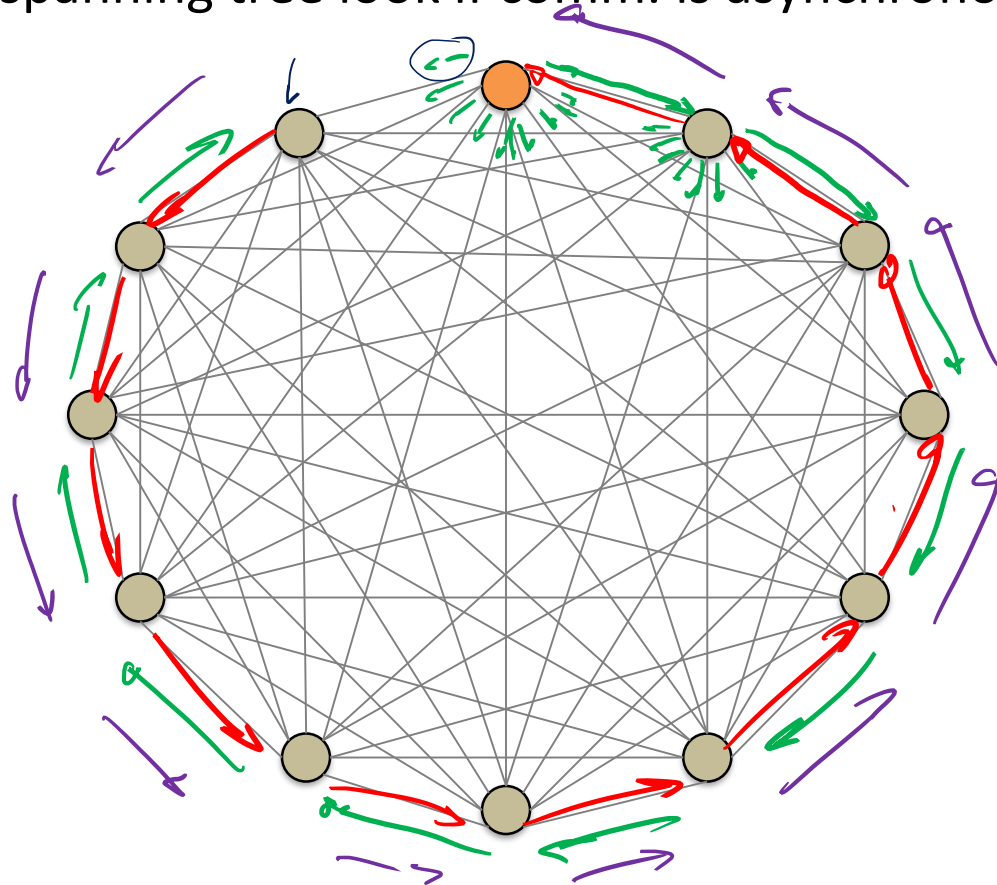
**Shortest Path Tree = BFS Tree** (BFS = breadth first search)

- tree which preserves graph distances to root node

**Theorem:** In synchronous systems, the flooding algorithm constructs a BFS tree.

# Spanning Tree: Asynchronous Systems

How does the spanning tree look if comm. is asynchronous?

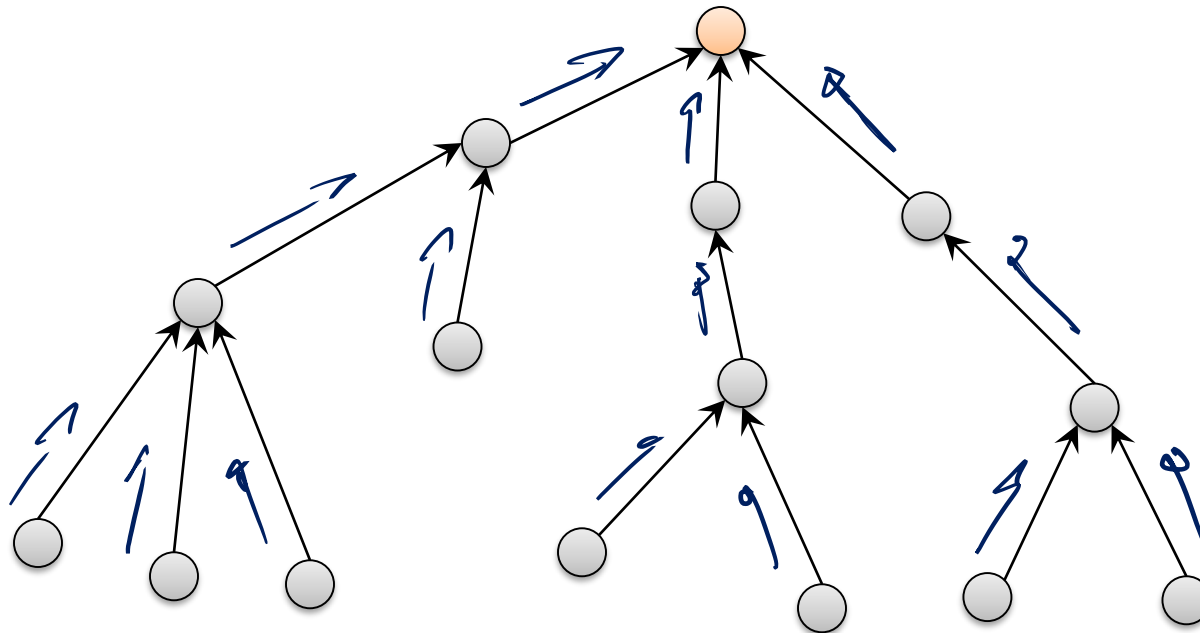


**Observation:** In asynchronous executions, the depth of the tree can be  $n - 1$  even if the radius/diameter of the graph is 1.

# Convergecast

- “Opposite” of broadcast
- Given a rooted spanning tree, communicate from all nodes to the root
  - starting from the leaves

Example: Compute sum of values in a rooted tree



# Convergecast Algorithm

**Leaf node  $v$ :**

**initially do**

send message to parent  
(e.g., send input value)

**Inner node  $u$ :**

**upon receiving message** from child node  $v$

**if**  $u$  has received messages from all children **then**  
send message to parent  
(e.g., send sum of all inputs in  $u$ 's subtree)

**Root node  $r$ :**

**upon receiving message** from child node  $v$

**if**  $r$  has received messages from all children **then**  
convergecast terminates

# Convergecast: Analysis & Remarks

## Time Complexity:

depth of tree

## Message Complexity:

#edges of tree =  $n - 1$

## Application of the convergecast algorithm:

- Computing functions, e.g.:
  - min, max, sum, average, median, ...
- Termination detection
  - inform parent as soon as all nodes in subtree have terminated
- ...

# Flooding/Echo Algorithm

- If a leader (root), but no spanning tree exists, flooding and convergecast can be used together for computing functions, ...
1. Use flooding to construct a tree
  2. Use convergecast (echo) to report back to the root when done

## Time Complexity of Flooding + Convergecast (Echo):

$O(\text{depth of constructed tree})$

synchronous:  $O(D)$  ← diameter

asynchronous:  $O(n)$

# Constructing Good Trees

- When combining flooding and convergecast, the time complexity is linear in the depth of the constructed tree.
- In synchronous systems, the tree is a BFS tree (shortest path tree), i.e., the depth of the tree is  $O(\text{diam}(G))$ 
  - optimal time complexity:  $O(\text{diam}(G))$
- In asynchronous systems, the time complexity can be  $\Omega(n)$ , even if the graph has a very small diameter!
- Convergecast / low diameter spanning trees are important!
- How can be construct a BFS tree in an asynchronous system?

# Constructing Shortest Path Tree

## Dijkstra

- Grow tree from source  $s$
- At intermediate step  $t$ , subtree of all nodes at distance  $\leq r_t$  from source node  $s$
- Next step: add node with min. distance to  $s$

## Bellman-Ford

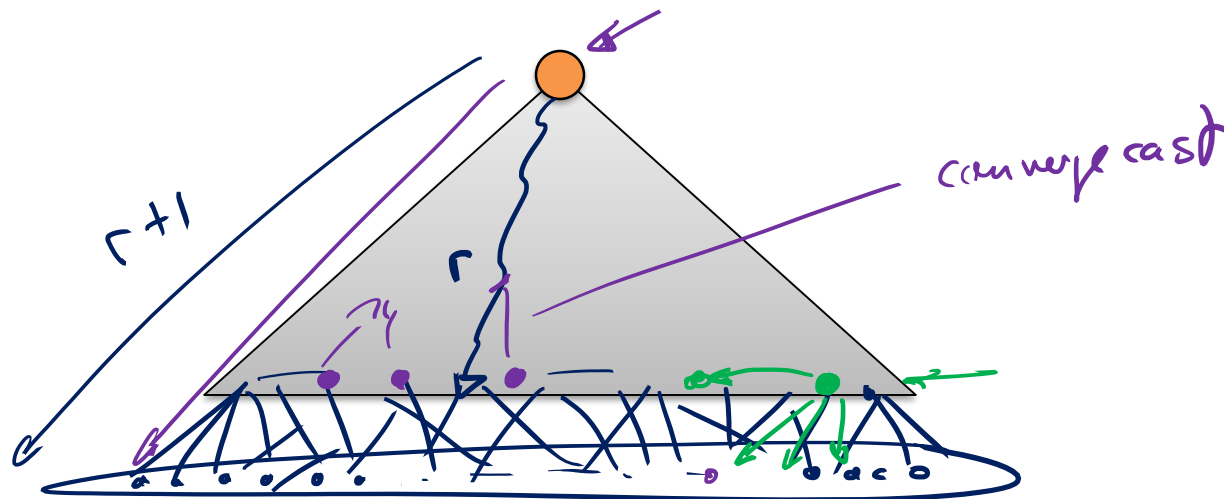
- Each node  $v$  keeps a distance estimate  $d_v$  to  $s$ 
  - initially:  $d_s = 0$ ,  $d_v = \infty$  (for all  $v \neq s$ )
- In each step, all nodes update their estimate based on neighbor estimates:

$$\underline{d_v} = \min \left\{ d_v, \min_{u \in N(v)} \{ \underline{d_u} + \underline{1} \} \right\}$$



# Distributed Dijkstra

- In our case, the graph is unweighted
- We can therefore grow the tree level by level
  - Essentially like in a synchronous execution
- Assume, the tree is constructed up to distance  $r$  from  $s$
- How can we add the next level?



# Distributed Dijkstra

- Source/root node coordinates the phases

## Algorithm for Phase $r + 1$ :

1. Root node broadcasts “*start phase  $r + 1$* ” in current tree
2. Leaf nodes (level  $r$  nodes) send “*join  $r + 1$* ” to neighbors
3. Node  $v$  receiving “*join  $r + 1$* ” from neighbor  $u$ :
  1. First such message:  $u$  becomes parent of  $v$ ,  $v$  sends *ACK* to  $u$
  2. Otherwise,  $v$  sends *NACK* to  $u$
4. After receiving *ACK* or *NACK* from all neighbors, level  $r$  nodes report back to root by starting a convergecast
5. When the convergecast terminates at the root, the root can start the next phase

# Distributed Dijkstra: Analysis

**Time Complexity:**

$$O\left(\sum_{i=1}^D i\right) = \boxed{O(D^2)}$$

**Message Complexity:**

$$\underline{O(m + D \cdot n)}$$

$\swarrow$   
 $|E|$

# Distributed Bellman-Ford

## Basic Idea:

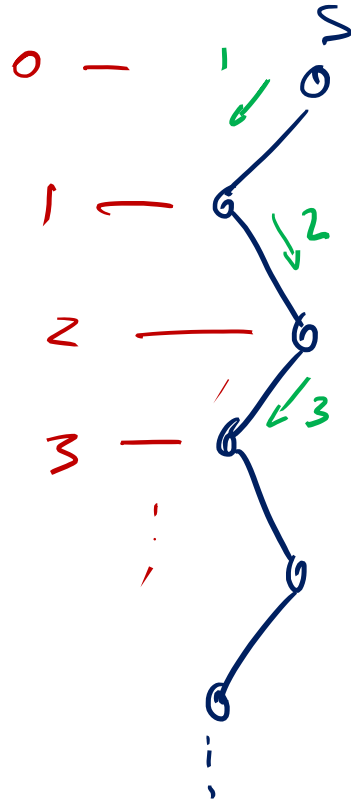
- Each node  $u$  stores an integer  $d_u$  with the current guess for the distance to the root node  $s$
- Whenever a node  $u$  can improve  $d_u$ ,  $u$  informs its neighbors

## Algorithm:

1. Initialization:  $d_s := 0$ , for  $v \neq s$ :  $d_v := \infty$ ,  $\text{parent}_v := \perp$
2. Root  $s$  sends “1” to all neighbors
3. For all other nodes  $u$ :  
**upon receiving message “ $x$ ” with  $x < \underline{d_u}$  from neighbor  $v$  do**  
    set  $d_u := x$   
    set  $\text{parent}_u := v$   
    send “ $x + 1$ ” to all neighbors (except  $v$ )

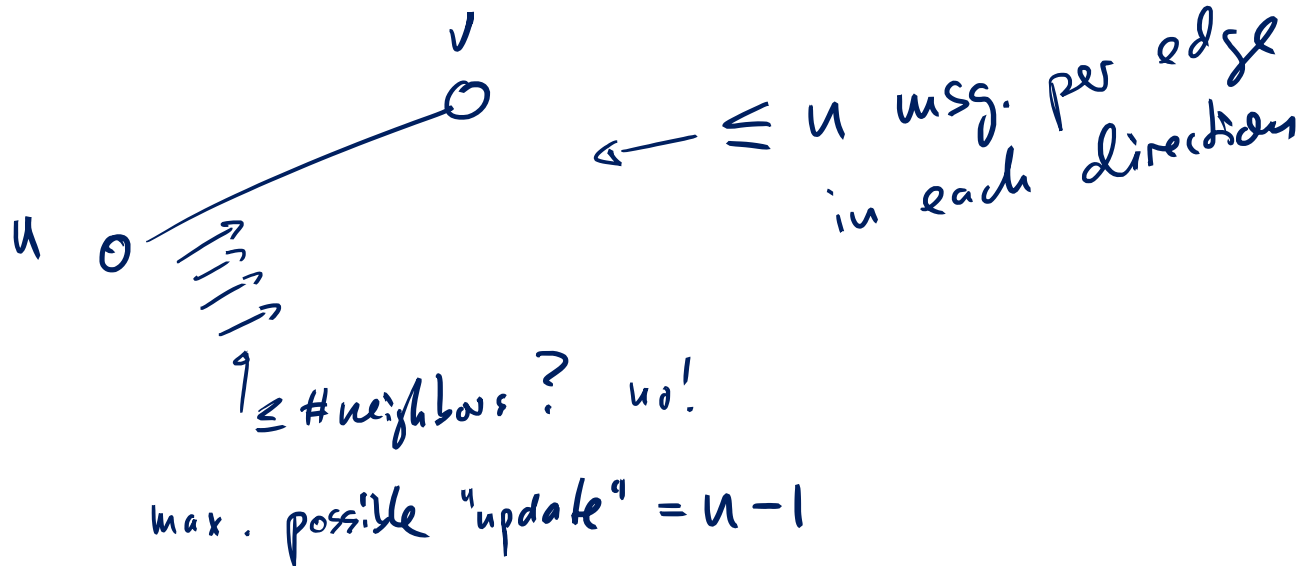
# Distr. Bellman-Ford: Time Complexity

**Theorem:** The time complexity of the distributed Bellman-Ford algorithms is  $\text{rad}(G, s) = O(D)$



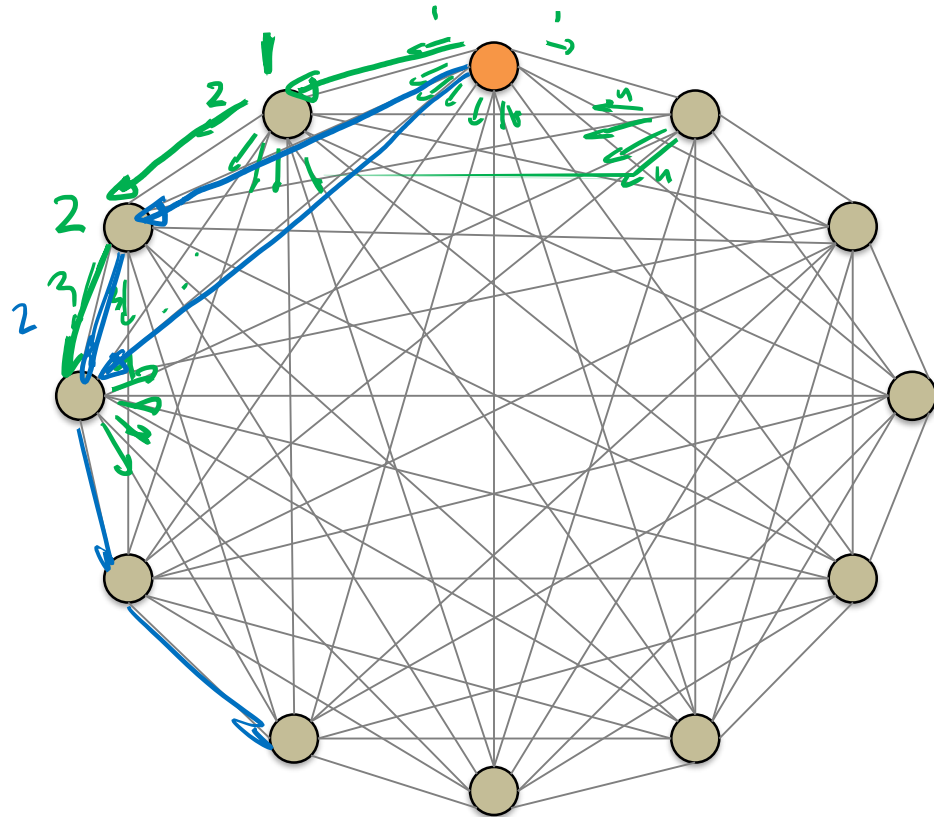
# Distr. Bellman-Ford: Message Complexity

**Theorem:** The message complexity of the distributed Bellman-Ford algorithms is  $O(|E| \cdot |V|) = O(m \cdot n)$  (right even if  $D=1$ )



# Distr. Bellman-Ford: Message Complexity

**Theorem:** The message complexity of the distributed Bellman-Ford algorithms is  $O(|E| \cdot |V|)$ .



# Distributed BFS Tree Construction

## Synchronous

- Time:  $O(\text{diam}(G))$ , Messages:  $O(|E|)$
- both optimal

## Asynchronous

- **Distributed Dijkstra:**  
Time:  $O(\text{diam}(G)^2)$ , Messages:  $O(|E| + |V| \cdot \text{diam}(G))$
- **Distributed Bellman-Ford:**  
Time:  $O(\text{diam}(G))$ , Messages:  $O(|E| \cdot |V|)$
- **Best known trade-off between time and messages:**  
Time:  $O(\text{diam}(G) \cdot \log^3 |V|)$ , Messages:  $O(|E| + |V| \cdot \log^3 |V|)$ 
  - based on **synchronizers**  
(generic way of translating synchronous algorithms into asynch. ones)
  - We will look at synchronizers in a later lecture...



# Leader Election

**Task:** Each node has an input value, compute sum of values

**Solution:** Compute spanning tree and use convergecast on spanning tree (i.e., flooding + convergecast)

**Problem:** What if we don't have a source/root node?

We need to choose a root node

- known as the *leader election problem*

## Solving leader election:

- E.g.: Choose node with smallest ID
- How to find node with smallest ID?

# Solving Leader Election

## Choose node with smallest ID

### Algorithm for node $u$ :

- Node  $u$  stores smallest known ID in variable  $x_u$ 
  1. Initially,  $u$  sets  $x_u := ID_u$  and sends  $x_u$  to all neighbors
  2. when receiving  $x_v < x_u$  from neighbor  $v$ :
    - $x_u := x_v$
    - send  $x_u$  to all neighbors (except  $v$ )

### Time Complexity:

$$O(D)$$

# Solving Leader Election

## Choose node with smallest ID

### Algorithm for node $u$ :

- Node  $u$  stores smallest known ID in variable  $x_u$
- 1. Initially,  $u$  sets  $x_u := ID_u$  and sends  $x_u$  to all neighbors
- 2. when receiving  $x_v < x_u$  from neighbor  $v$ :
  - $x_u := x_v$
  - send  $x_u$  to all neighbors (except  $v$ )

### Message Complexity:

$$O(m \cdot n)$$

# Solving Leader Election

## Choose node with smallest ID

### Algorithm for node $u$ :

- Node  $u$  stores smallest known ID in variable  $x_u$
- 1. Initially,  $u$  sets  $x_u := ID_u$  and sends  $x_u$  to all neighbors
- 2. when receiving  $x_v < x_u$  from neighbor  $v$ :
  - $x_u := x_v$
  - send  $x_u$  to all neighbors (except  $v$ )

### Termination?

# Leader Election

Simple leader election algorithm has time complexity  $O(\text{diam}(G))$  and message complexity  $O(|V| \cdot |E|)$ .

## Problems:

- While time compl. is optimal, msg. complexity is extremely high
- It is not clear when/how to terminate
- Like for BFS tree construction, there are **many possible trade-offs** between time and message complexity, e.g.:
  - Time Complexity:  $O(|V|)$ , Message Complexity:  $O(|E| + |V| \cdot \log|V|)$
  - Algorithm is based on constructing a spanning tree in a message-efficient way
- Termination can be solved
  - see exercises!