
Self-stabilizing systems



Professor Seif Haridi

Cosmin Arad, Tallat M. Shafaat

{haridi, icarad, tallat}@kth.se

Self-stabilization

- Self-stabilization first introduced in 1973 by Edsger W. Dijkstra
- A system is called **self-stabilizing** if it can converge from an **arbitrary** state to a **desired** state in **finite** number of steps

Motivation

- Fault-tolerance; especially transient faults
 - Corrupted state (code?)
 - Initialization errors
 - Processor failure
 - Loss of synchrony
 - ...

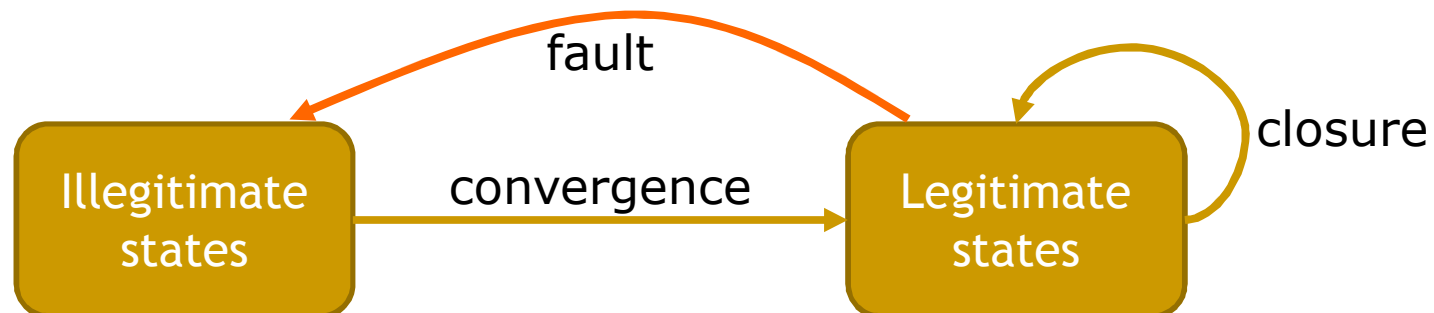
- Applications in computer science
 - Mutual exclusion, routing algorithms, resource allocation.

Definitions

- Legitimate state
 - When the system state satisfies a desired property P
 - E.g., P =there should be a single leader
- Illegitimate state
 - When P is violated (un-safe)
 - E.g., multiple leaders exist
- Starting from an illegitimate state, if things are well-behaved for long enough, the system will converge to a legitimate state

Formal definition

- A system S is self-stabilizing w.r.t. predicate P (defining legitimacy) iff:
 - **Convergence**
 - Starting from any arbitrary state, S will satisfy P within finite state transitions
 - **Closure**
 - Starting from a state that satisfies P , any computation will lead to states that satisfy P



Pros/Cons

- Advantages

- Fault tolerance
- No need for proper system initialization

- Disadvantages

- Initial inconsistencies violate safety
- Highly complex
- Detection of stabilization

Examples

- Routing
 - During convergence, build the routing tables
 - After convergence, use the routing tables
- Leader election
 - During convergence, multiple leaders
 - After convergence, single leader
- Requirement:
 - Termination detection

Model

- Processes are connected in a graph
- At each **step**, a process looks at its own and neighbour's state, and changes its own state
- Processes are scheduled to take steps in a **fair** manner

- Usually implemented as a do-forever loop
- Shared memory between neighbours

Impossibility

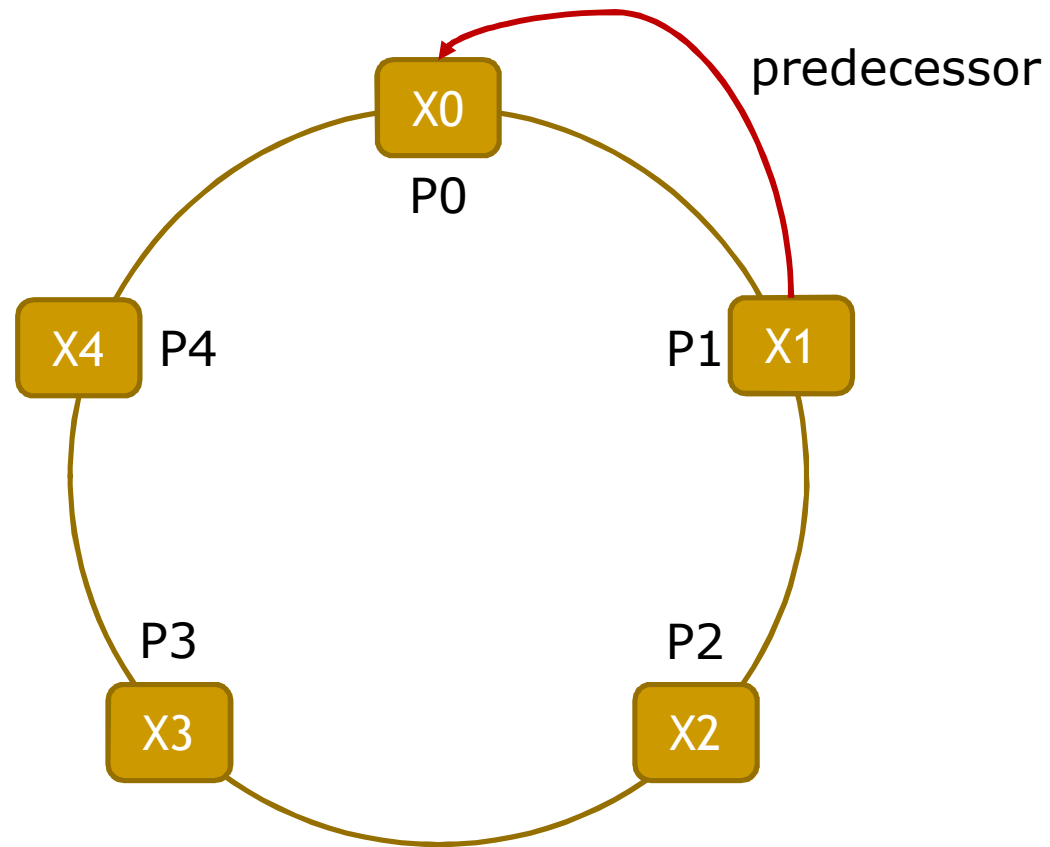
- If the processes are identical (symmetric), then stabilization is impossible
 - There has to be at least one different process (a.k.a 'privileged')
- Later, shown that with prime number of processes, self-stabilization with symmetric processes is possible [Burns and Pachl]

Lecture's focus

- Two examples
 - Dijkstra's token ring algorithm
 - A self-stabilizing spanning tree algorithm

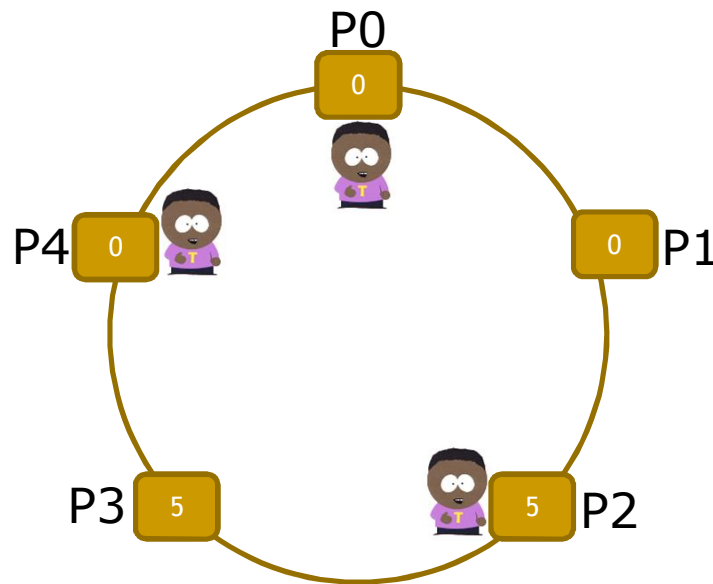
Token ring

- Mutual exclusion problem
- N processes connected in a ring
- Each talks to (can see state of) its counter-clockwise neighbour (a.k.a predecessor)
- State
 - local variable X
 - $0 \leq X < K$, where $K > N$
- Legitimate state of S:
 - There is only one token in the system



Token ring

- Privileged node: P0
- Do I have the token?
 - For P0, if $X = \text{predecessor}.X$
 - For P1 to PN, if $X \neq \text{predecessor}.X$

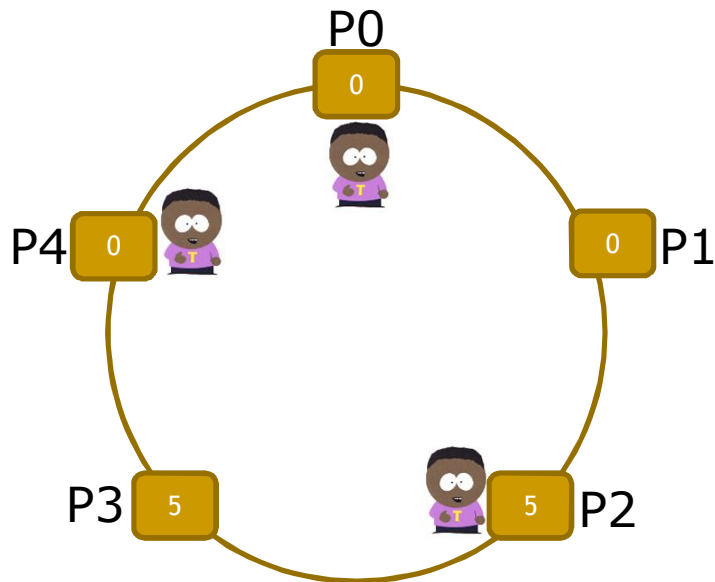


Token ring

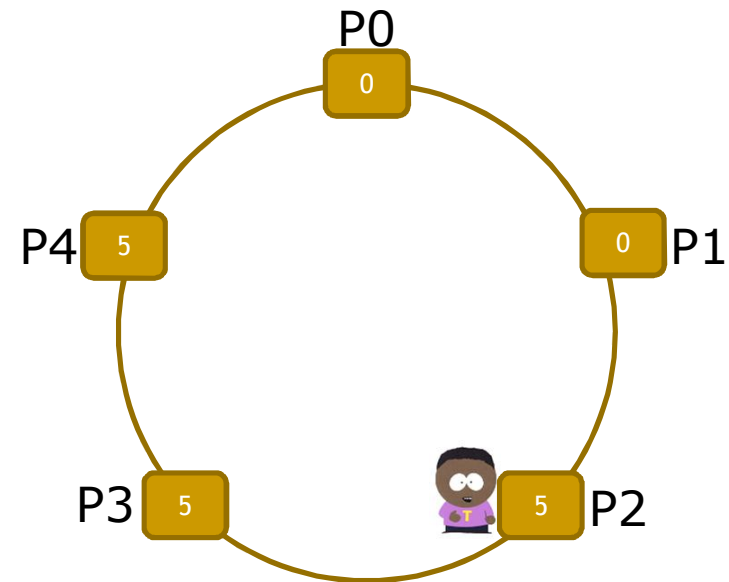
- Predicate P
 - Exactly one token in S

Token rule

For P0:
if $X = \text{predecessor}.X$
For P1 to PN:
if $X \neq \text{predecessor}.X$



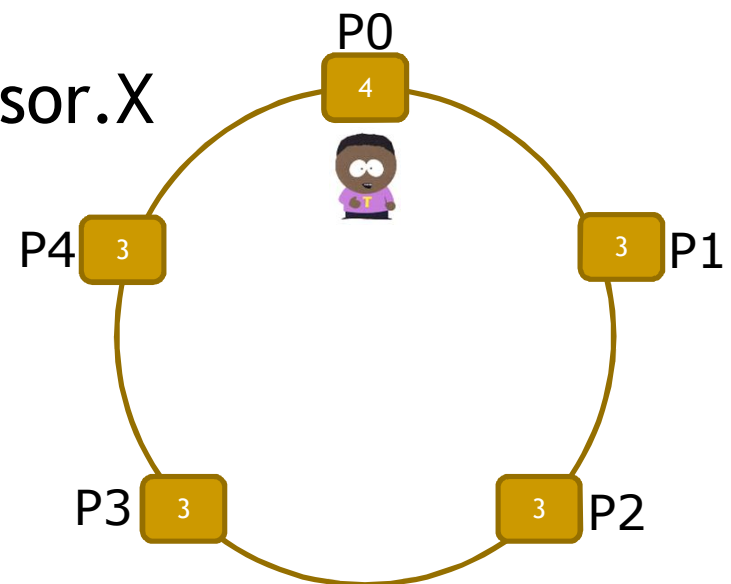
Illegitimate



Legitimate

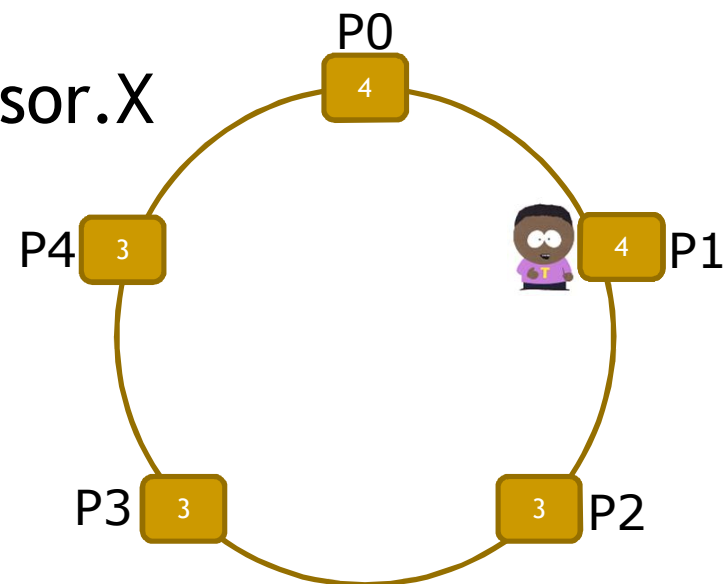
Circulating the token

- After holding the token, the process updates its local state to pass on the token clockwise
- Token condition
 - For P0, if $X = \text{predecessor}.X$
 - $X = (X+1) \bmod K$
 - For P1 to PN, if $X \neq \text{predecessor}.X$



Circulating the token

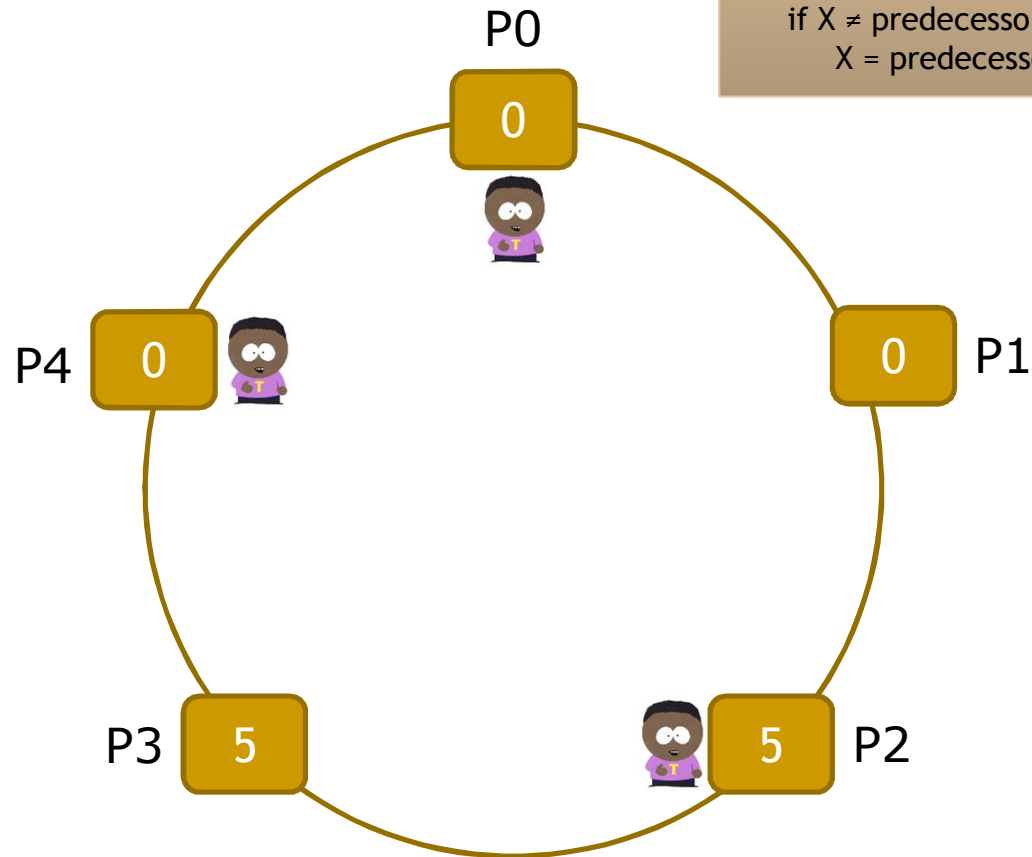
- After holding the token, the process updates its local state to pass on the token clockwise
- Token condition
 - For P0, if $X = \text{predecessor}.X$
 - $X = (X+1) \bmod K$
 - For P1 to PN, if $X \neq \text{predecessor}.X$
 - $X = \text{predecessor}.X$



Convergence

For P0:
if $X = \text{predecessor.X}$ ← token test
 $X = (X+1) \bmod K$ ← token passing

For P1 to PN:
if $X \neq \text{predecessor.X}$ ← token test
 $X = \text{predecessor.X}$ ← token passing



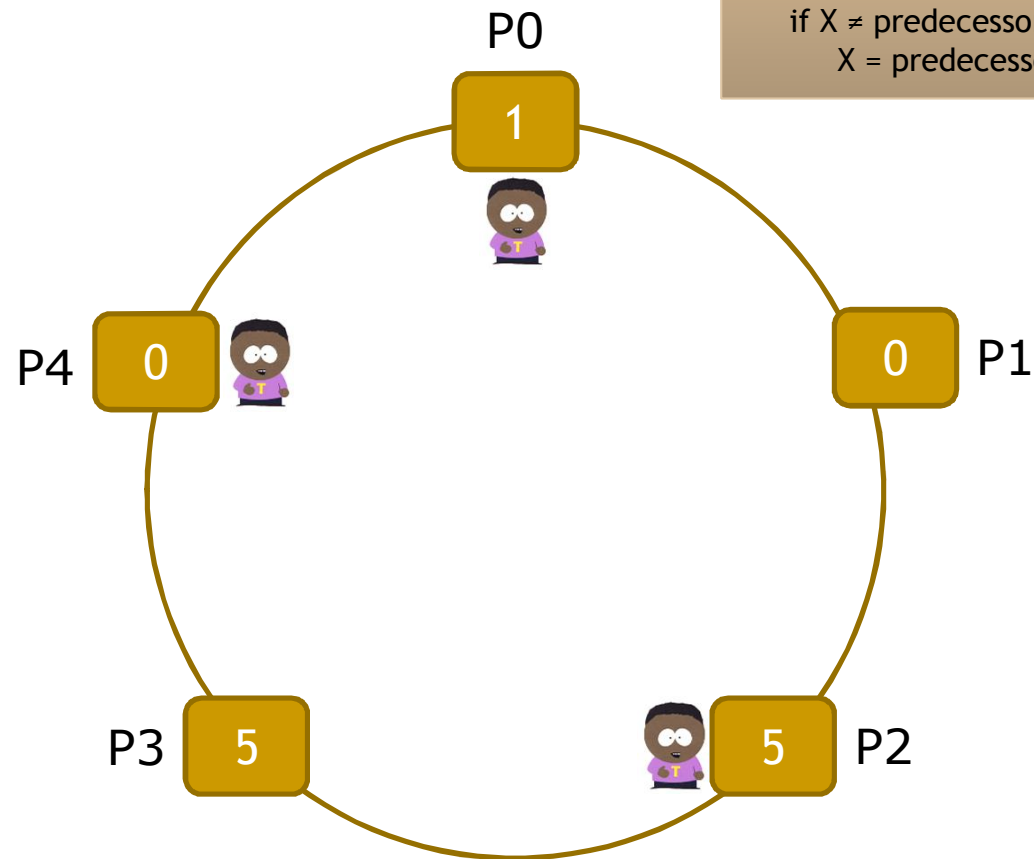
Say schedule selects **P0** to take a step

Illegitimate

Convergence

For P0:
if $X = \text{predecessor.X}$ ← token test
 $X = (X+1) \bmod K$ ← token passing

For P1 to PN:
if $X \neq \text{predecessor.X}$ ← token test
 $X = \text{predecessor.X}$ ← token passing



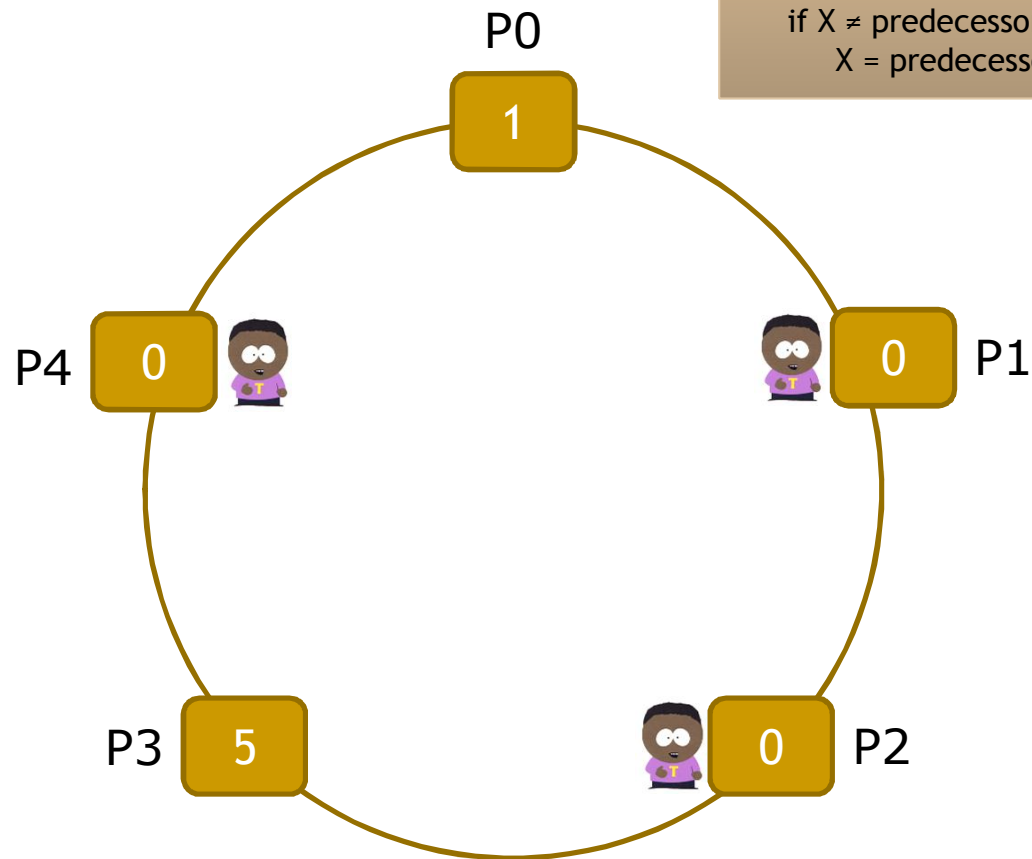
Say schedule selects **P2** to take a step

Illegitimate

Convergence

For P0:
if $X = \text{predecessor.X}$ ← token test
 $X = (X+1) \bmod K$ ← token passing

For P1 to PN:
if $X \neq \text{predecessor.X}$ ← token test
 $X = \text{predecessor.X}$ ← token passing



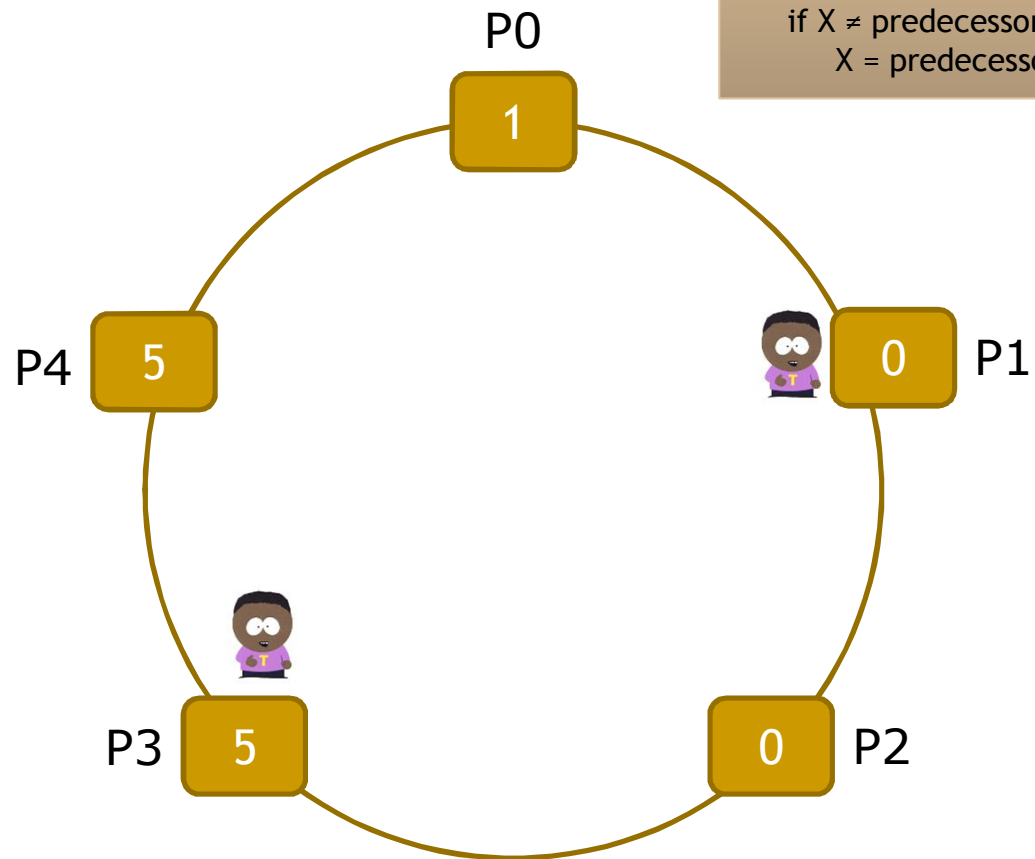
Say schedule selects **P4** to take a step

Illegitimate

Convergence

For P0:
if $X = \text{predecessor.X}$ ← token test
 $X = (X+1) \bmod K$ ← token passing

For P1 to PN:
if $X \neq \text{predecessor.X}$ ← token test
 $X = \text{predecessor.X}$ ← token passing



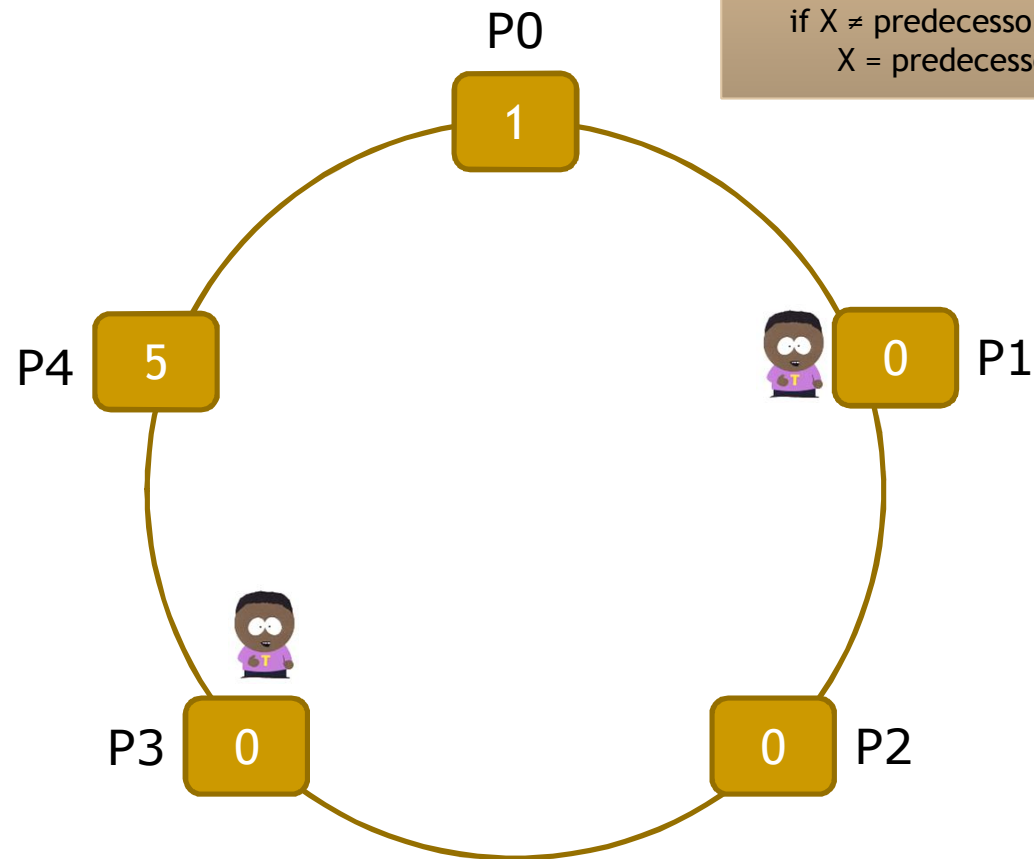
Say schedule selects **P3** to take a step

Illegitimate

Convergence

For P0:
if $X = \text{predecessor}.X$ ← token test
 $X = (X+1) \bmod K$ ← token passing

For P1 to PN:
if $X \neq \text{predecessor}.X$ ← token test
 $X = \text{predecessor}.X$ ← token passing



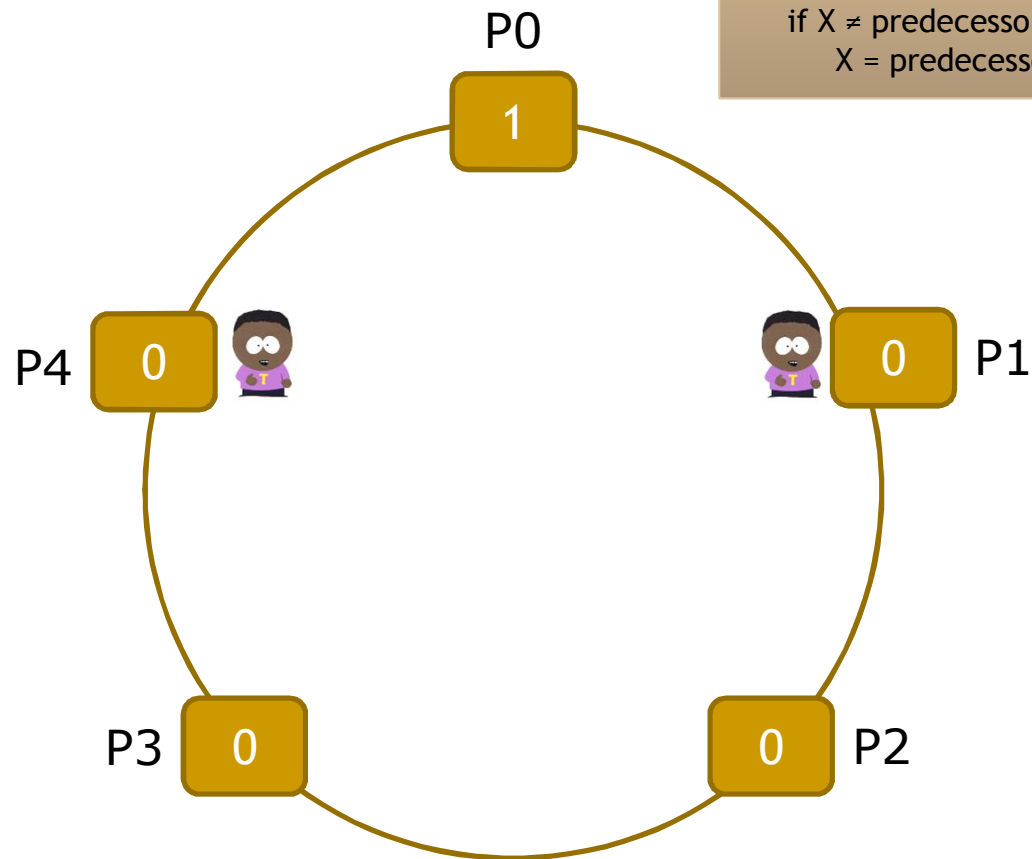
Say schedule selects **P4** to take a step

Illegitimate

Convergence

For P0:
if $X = \text{predecessor}.X$ ← token test
 $X = (X+1) \bmod K$ ← token passing

For P1 to PN:
if $X \neq \text{predecessor}.X$ ← token test
 $X = \text{predecessor}.X$ ← token passing

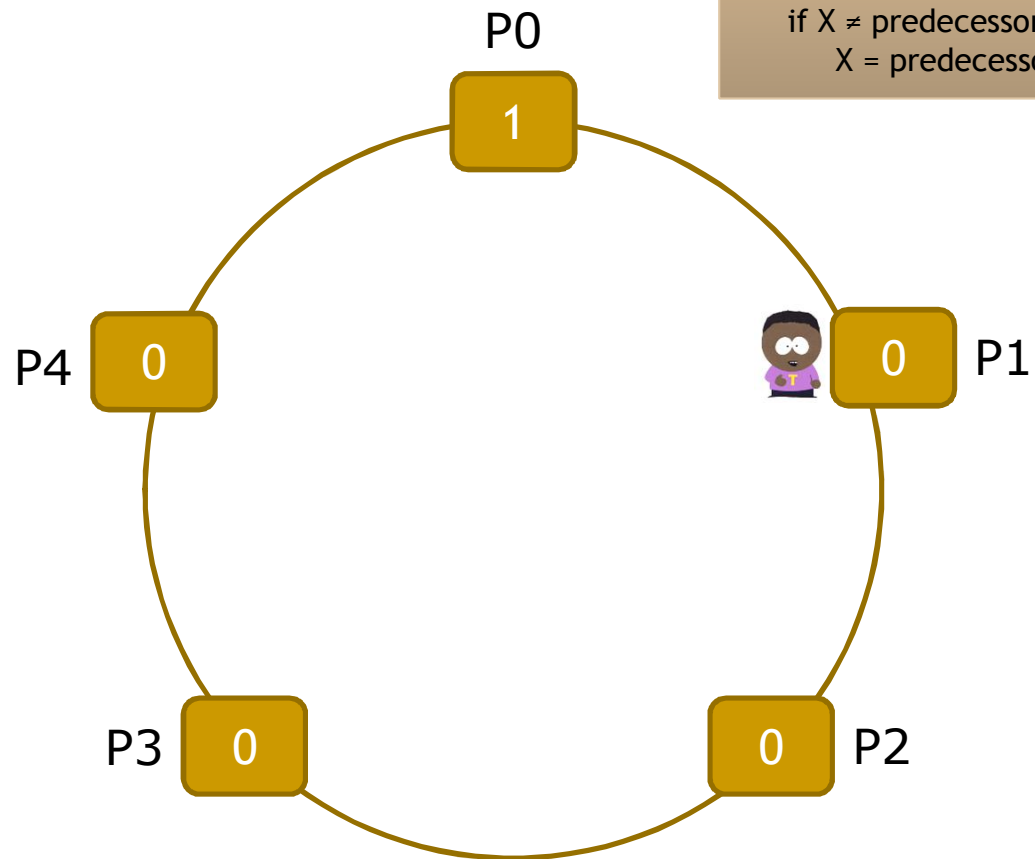


Legitimate

Closure

For P0:
if $X = \text{predecessor}.X$ ← token test
 $X = (X+1) \bmod K$ ← token passing

For P1 to PN:
if $X \neq \text{predecessor}.X$ ← token test
 $X = \text{predecessor}.X$ ← token passing

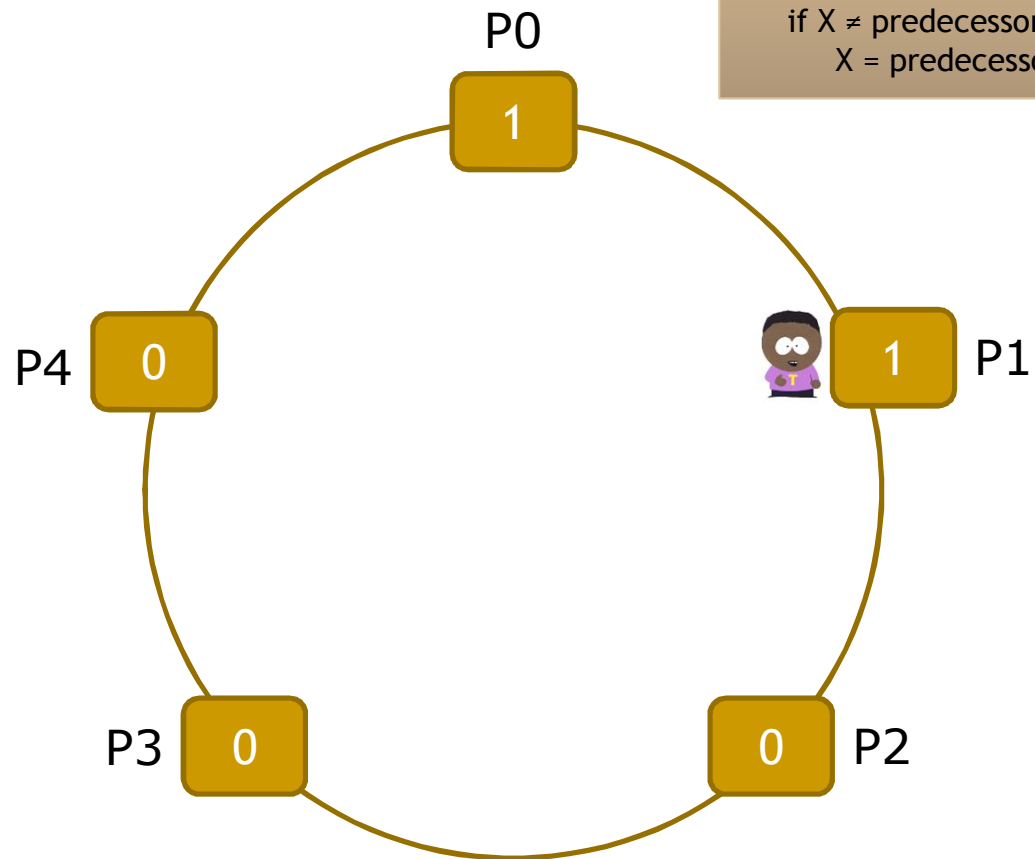


Legitimate

Closure

For P0:
if $X = \text{predecessor}.X$ ← token test
 $X = (X+1) \bmod K$ ← token passing

For P1 to PN:
if $X \neq \text{predecessor}.X$ ← token test
 $X = \text{predecessor}.X$ ← token passing

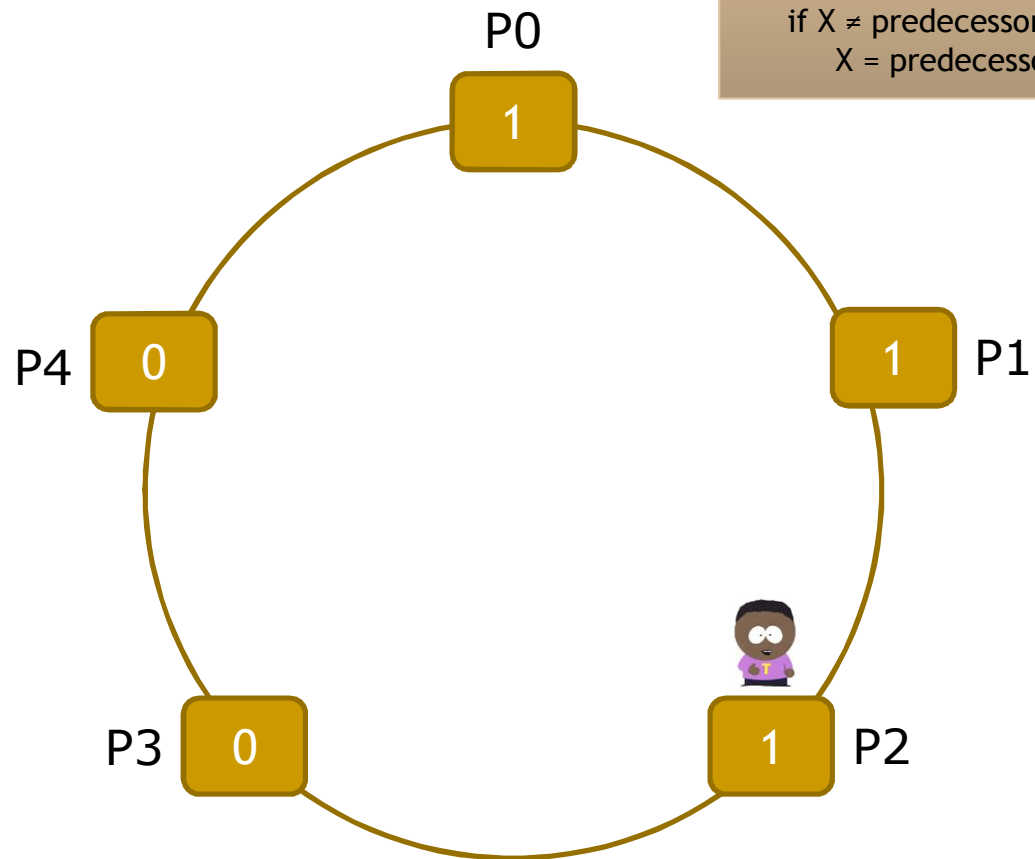


Legitimate

Closure

For P0:
if $X = \text{predecessor}.X$ ← token test
 $X = (X+1) \bmod K$ ← token passing

For P1 to PN:
if $X \neq \text{predecessor}.X$ ← token test
 $X = \text{predecessor}.X$ ← token passing



and so on ...

Legitimate

Proof of closure

- Only one token in S
- State changes only at the node that has the token
- In such as case, a state change only passes the token clockwise
 - Token moves clockwise along the ring
- Hence, no new token created

Informal proof of convergence (1/2)

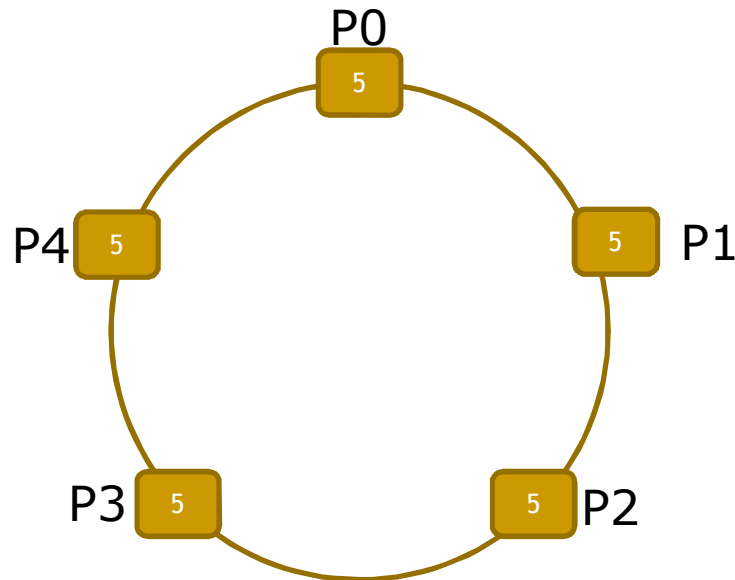
- As long as $K > N$, there is at least one value y ($0 \leq y < K$) that is **not** the initial state of any node (**pigeon hole principle**)
- There is no deadlock
- Number of tokens never increases (**closure**)

Informal proof of convergence (2/2)

- Processes $1, \dots, N-1$ acquire their states from counter-clockwise neighbour
- Hence, P_0 eventually attains the state y
- Thereafter, in $N-1$ steps, all processes attain the state y
- This is a legitimate configuration (only P_0 has a token)

Proof of convergence

- Lemma 1: P0 eventually receives the token
- Lemma 2: All states will be the same after P0 had the token **at most** N times

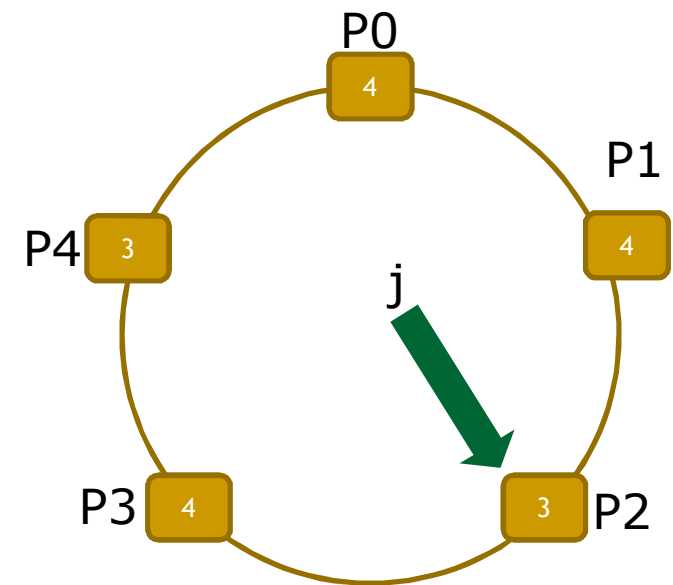


Proof of lemma 1

For P0:
 if $X = \text{predecessor}.X$ ← token test
 $X = (X+1) \bmod K$ ← token passing

For P1 to PN:
 if $X \neq \text{predecessor}.X$ ← token test
 $X = \text{predecessor}.X$ ← token passing

- Lemma 1: P0 eventually receives the token
 - If it doesn't have token, then $P0.X \neq P0.\text{pred}.X$
 - Let j be the minimum value such that $Pj.X \neq P0.X$
 - For all $i < j$, $Pi.X = P0.X$
 - $\rightarrow Pj.X \neq Pj-1.X$
 - $\rightarrow Pj$ is privileged
 - Pj will update state
 - Increases j if $j < N$, or
 - makes $P0.X = P0.\text{pred}.X (=PN.X)$ if $j = N$
 - P0 will receive the token



Proof

- Convergence + Closure proved
- Hence, the "Token Ring" algorithm is self-stabilizing

A self-stabilizing spanning tree algorithm

Problem Statement

- Given:
 - A connected graph $G=(V,E)$, and
 - A node r (root)
- Design a BFS self-stabilizing algorithm for maintaining a spanning tree
 - Transient failures corrupt local state
- Tree
 - Connected
 - Directed Acyclic Graph

Definitions

- Size of the network, $n = |V|$
- $N(i)$: set of neighbours of node i
- $\delta_i = |N(i)|$
- $\Delta = \max(\delta_1, \delta_2, \dots, \delta_n)$
 - i.e. max degree

Local State

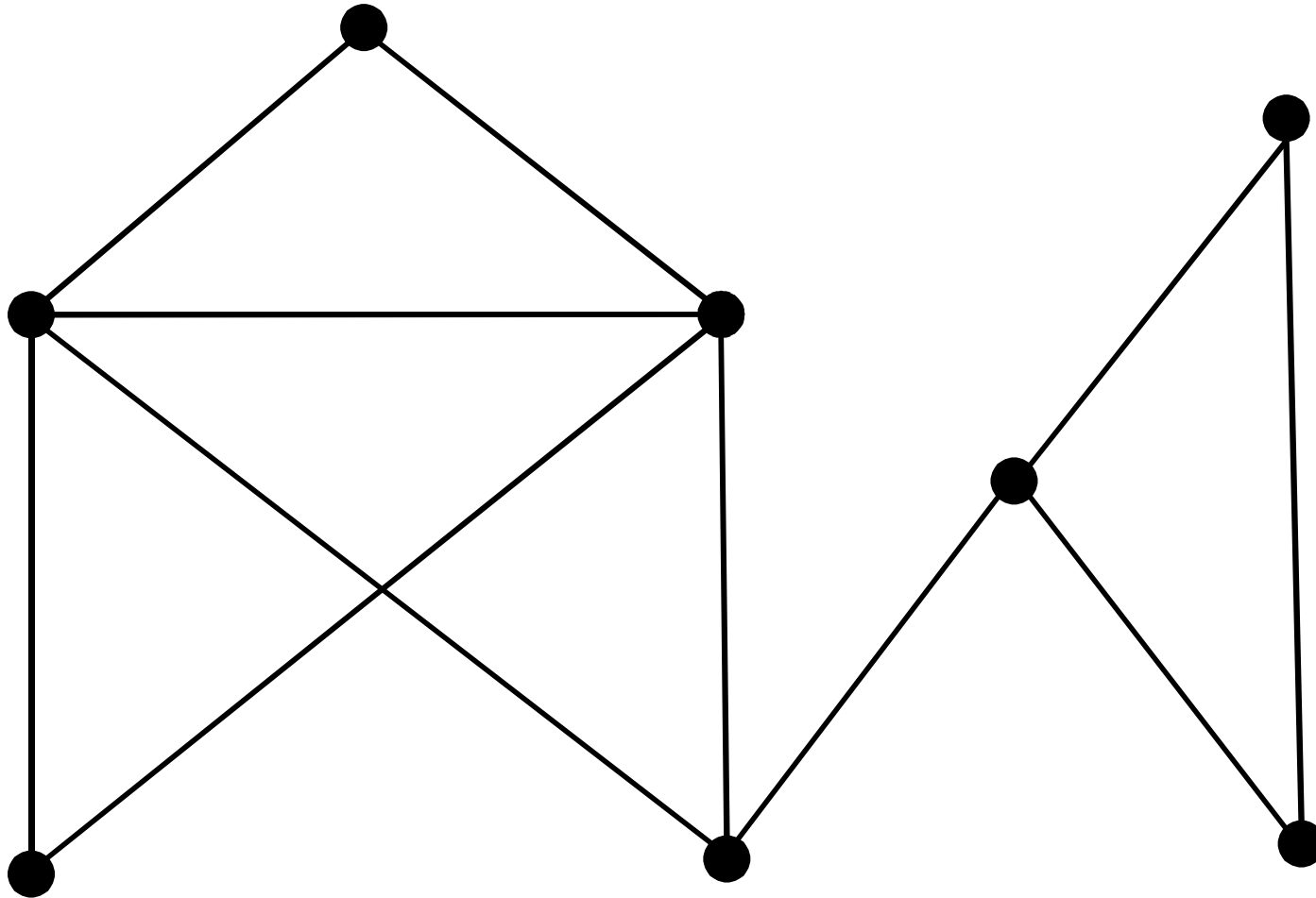
- Each node i stores
 - Distance to root, $L(i)$
 - Parent, $P(i)$
- For this talk, ‘distance’ means distance of the node to root
- By definition
 - $0 \leq L(i) < n$

Legitimate State

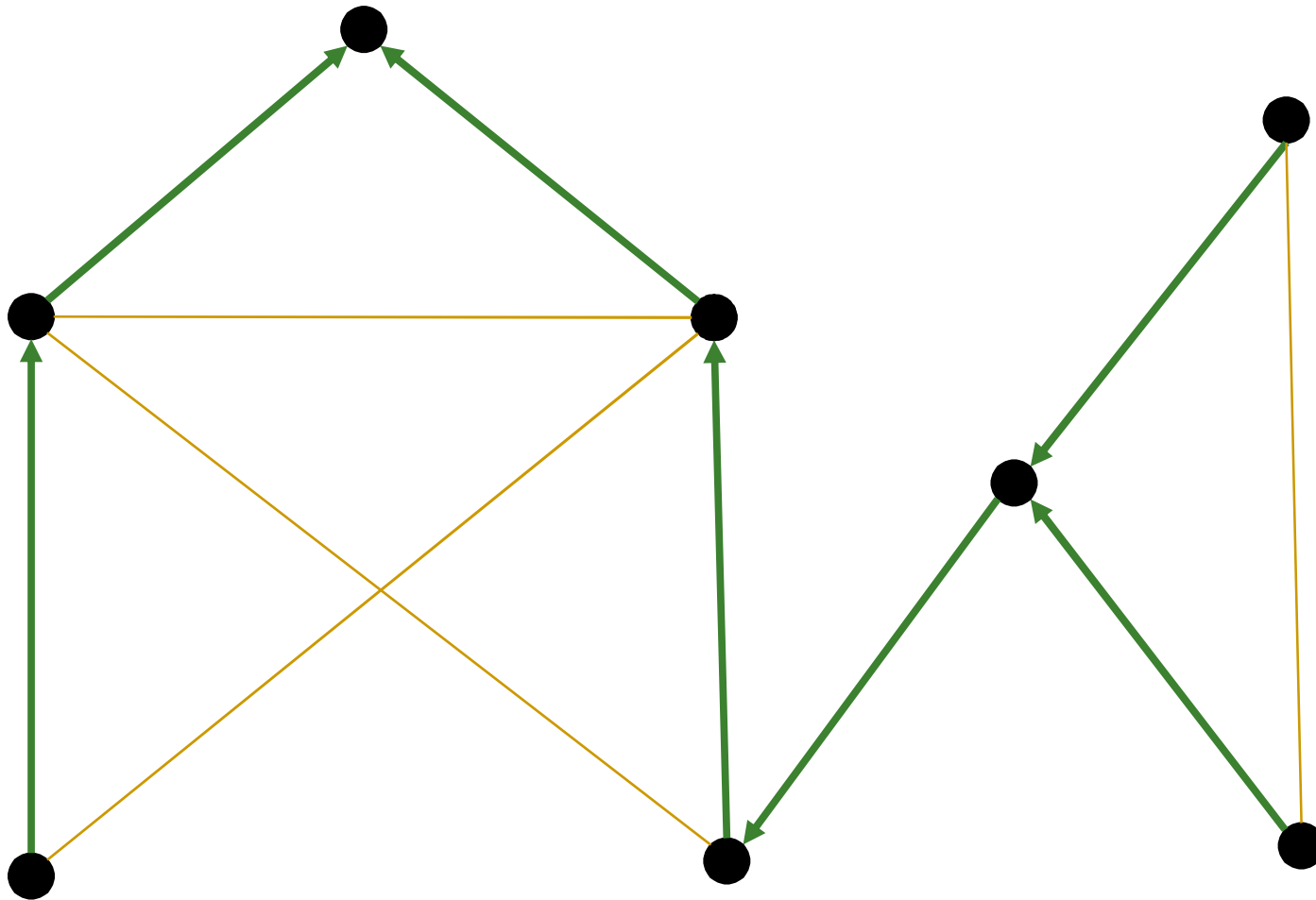
- For root
 - $L(r) = 0$
 - $P(r)$ is undefined

- For non-root nodes
 - $L(i) = L(P(i)) + 1$
 - Parent variables form spanning tree rooted at 'r'

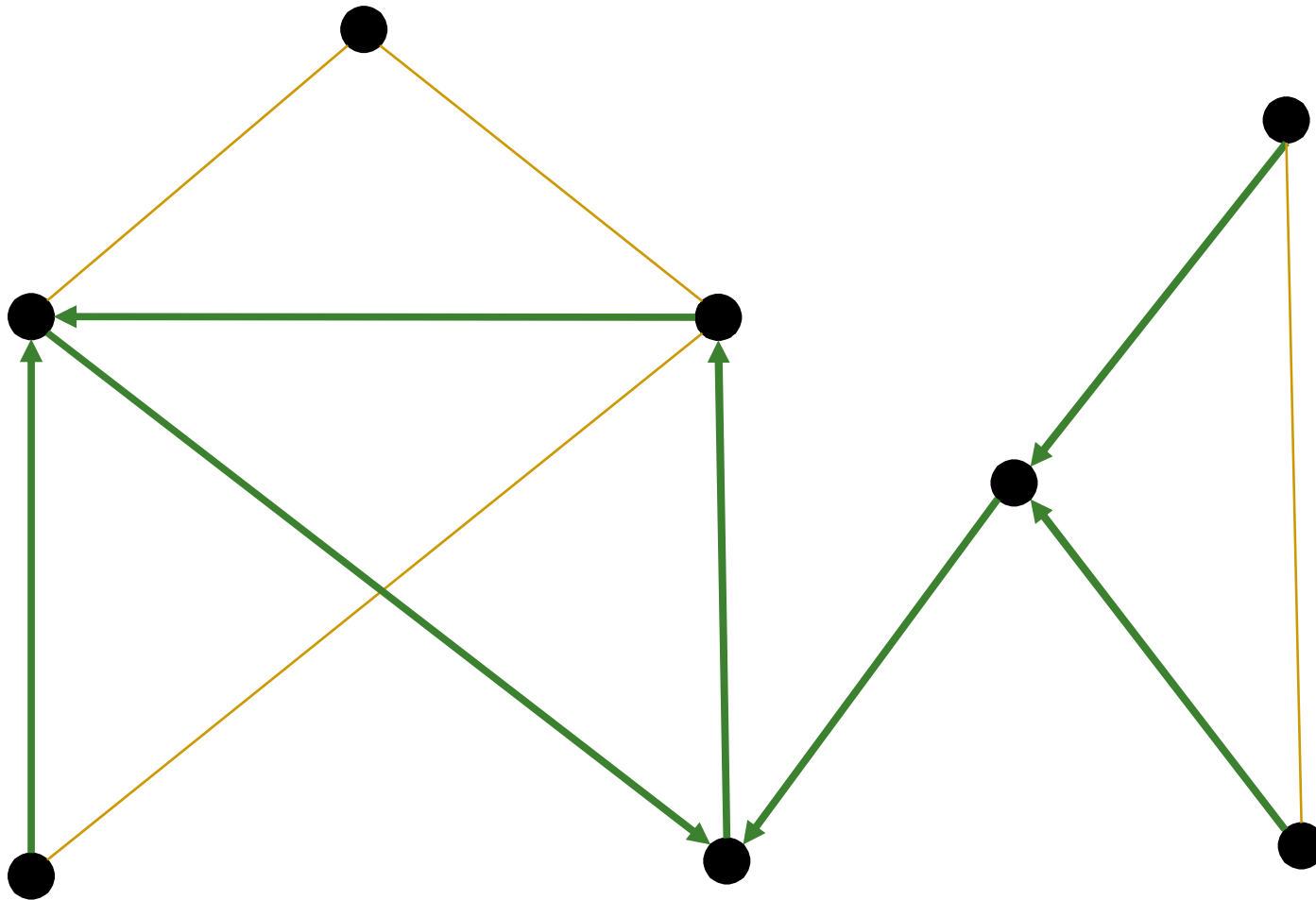
Example - Graph



Example - BFS Spanning Tree



Example - Illegitimate State



Algorithm

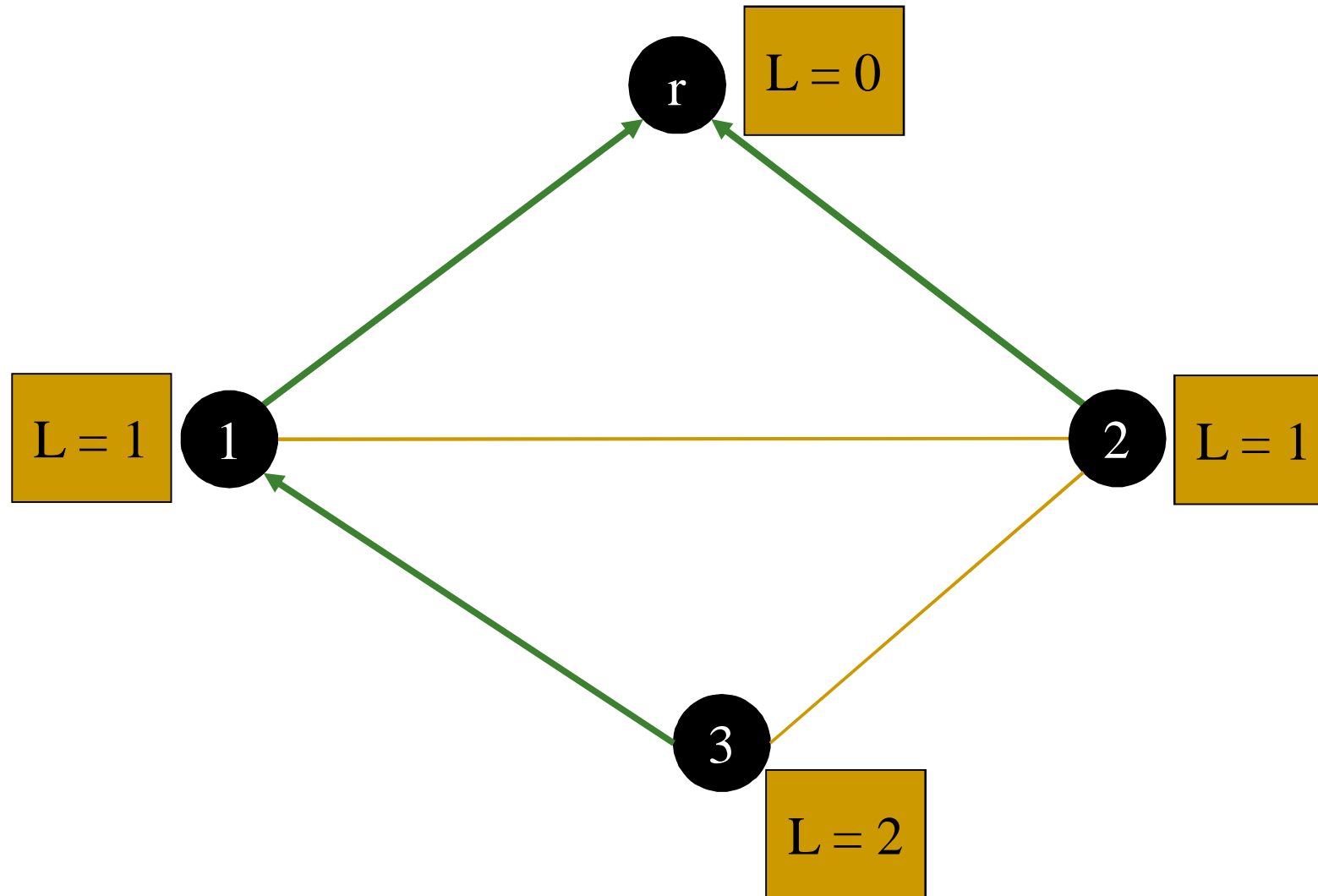
- Roles of nodes are asymmetric
 - Root runs a different algorithm
- Root, r (periodically)
 - Set *parent* as undefined and *distance* to 0
- All non-root nodes, $i \neq r$ (periodically)
 - From neighbours, choose as parent the closest node to root, i.e. which has minimum distance
 - If multiple at same distance, use a predefined criteria, e.g. min id node
 - Set *distance* = 1 + parent's distance to root

Algorithm - Formal

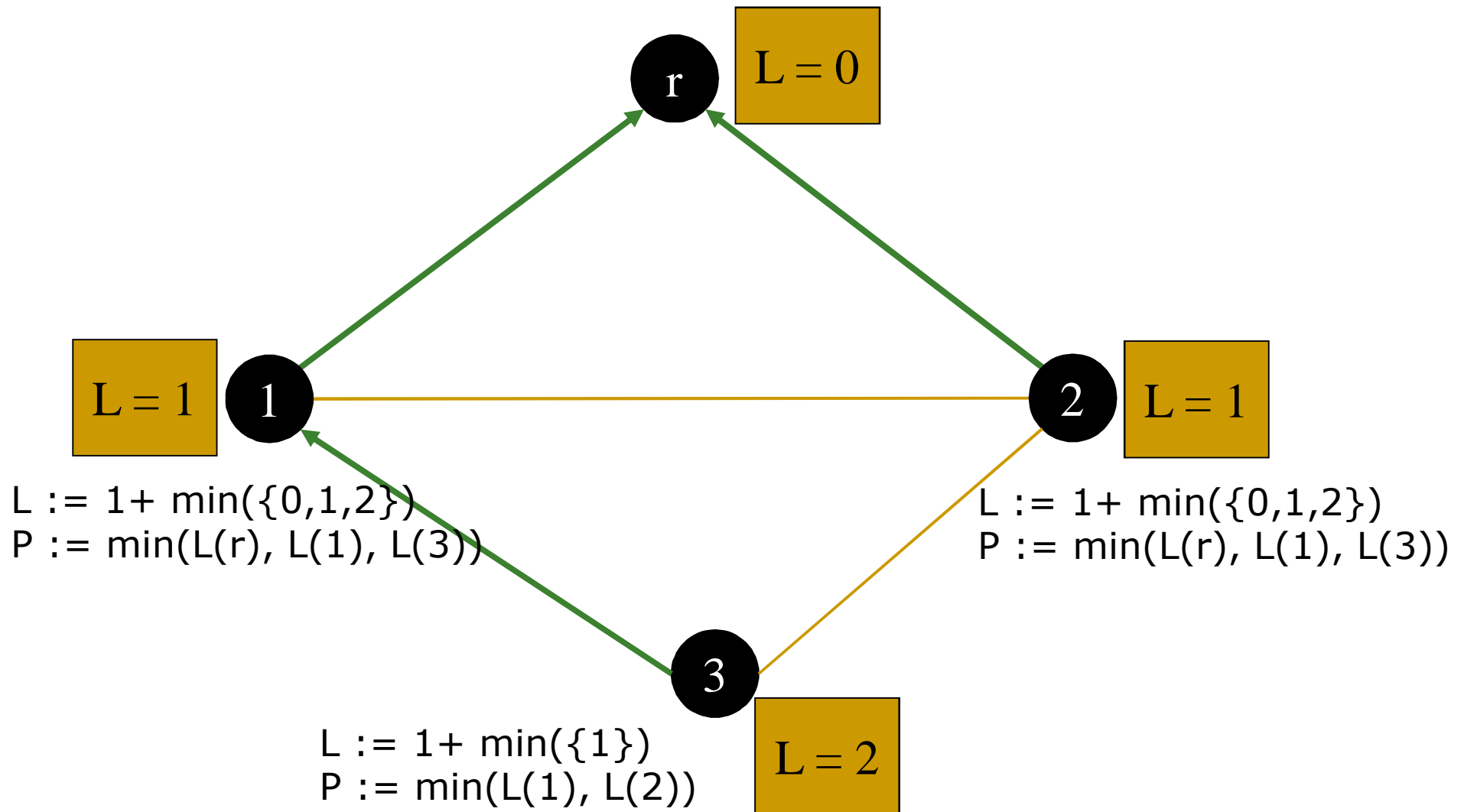
- Root, r
 - $P(r) := \text{undefined}$
 - $L(r) := 0$

- All non-root nodes, $i \neq r$
 - $P(i) := \text{minRank} (\{ j \mid j, k \in N(i) : L(j) \leq L(k) \})$
 - $L(i) := 1 + L(P(i))$

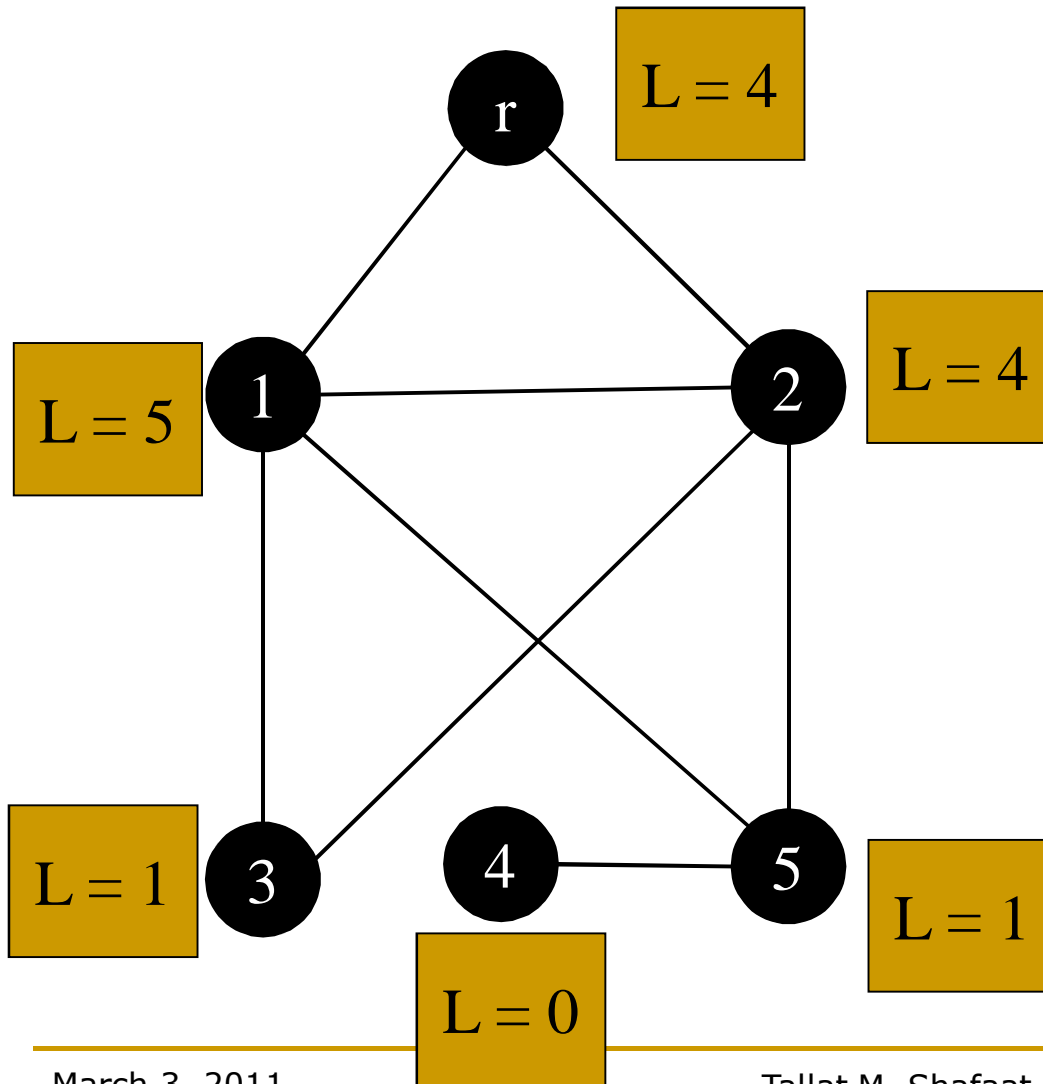
Example 1 - Sample Execution



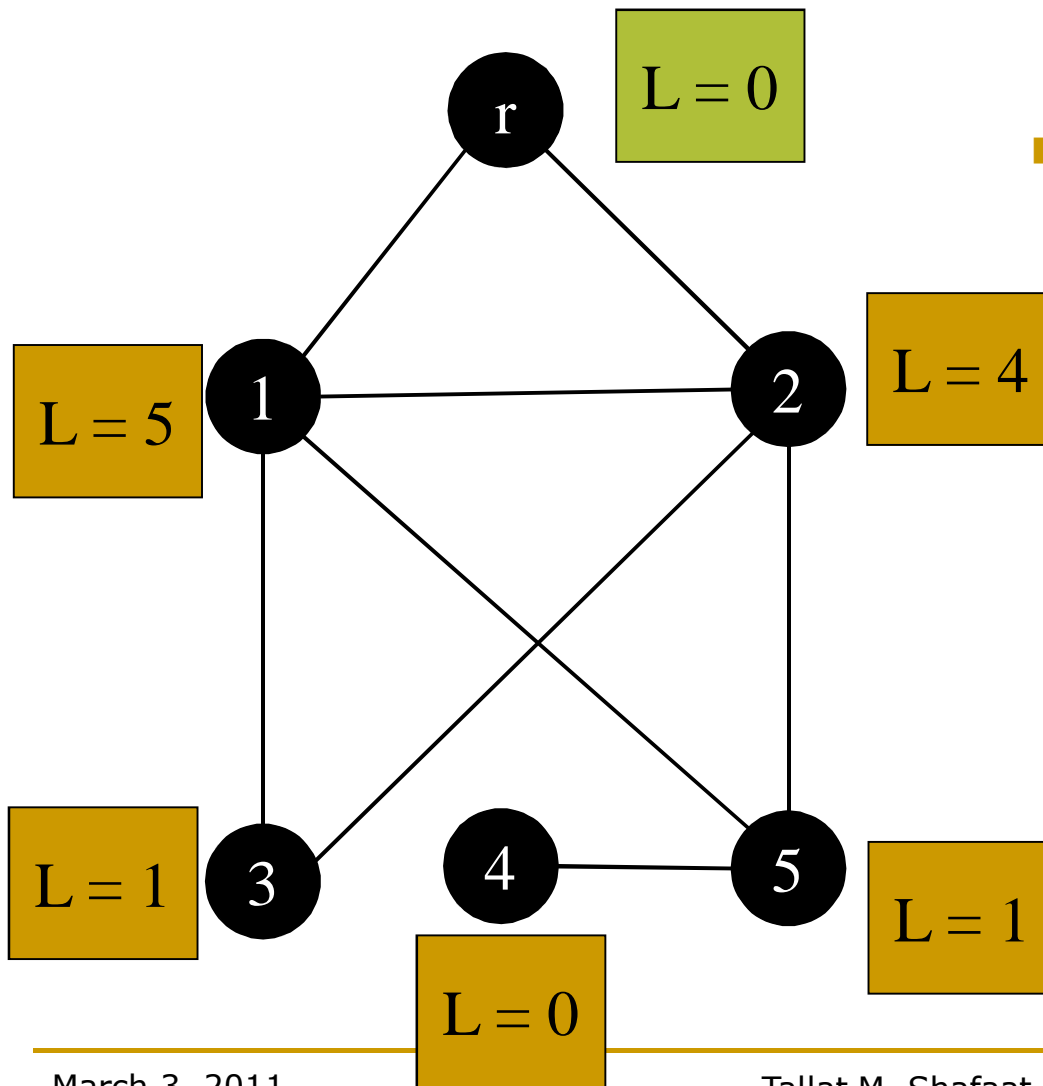
Example 1 - Sample Execution



Example 2 - Sample Execution

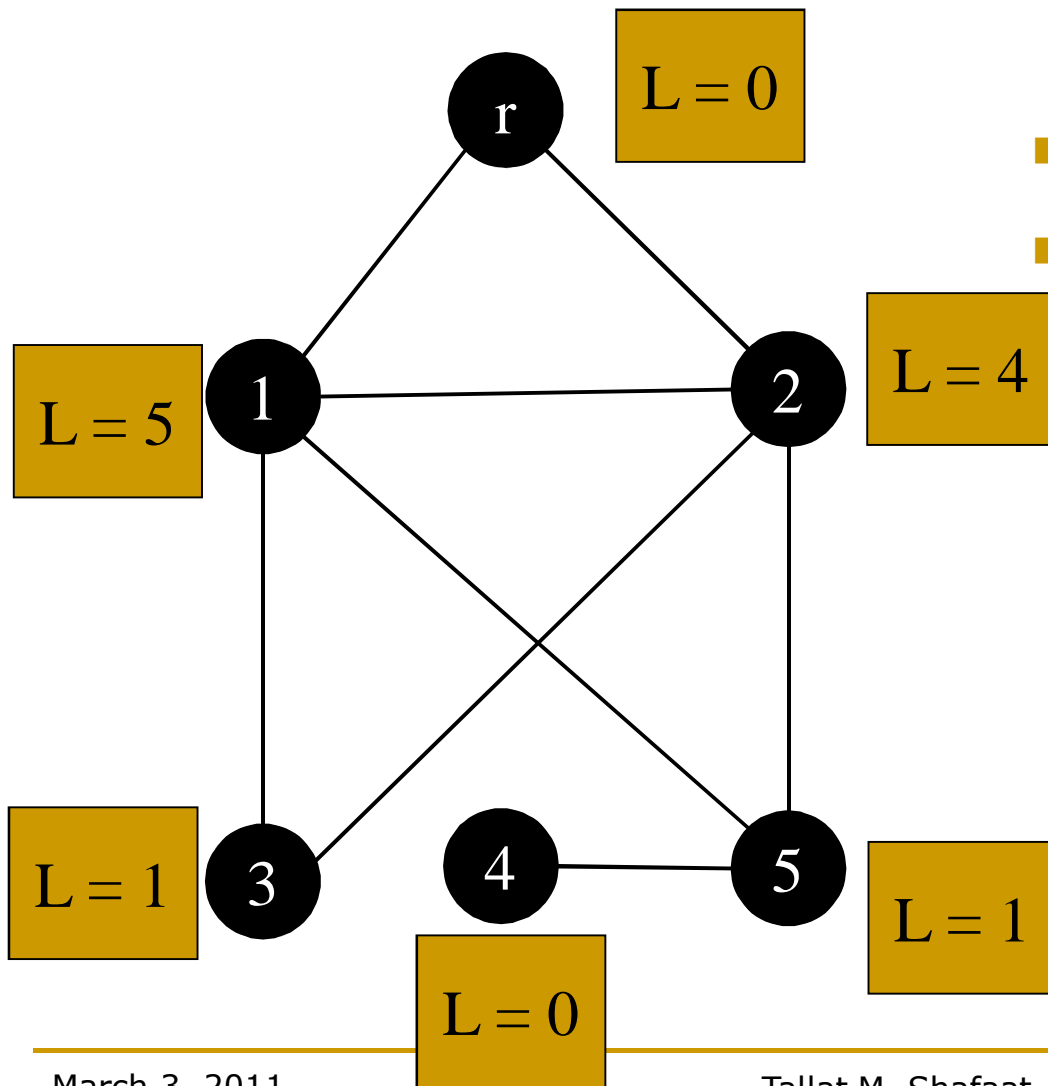


Example 2 - Sample Execution



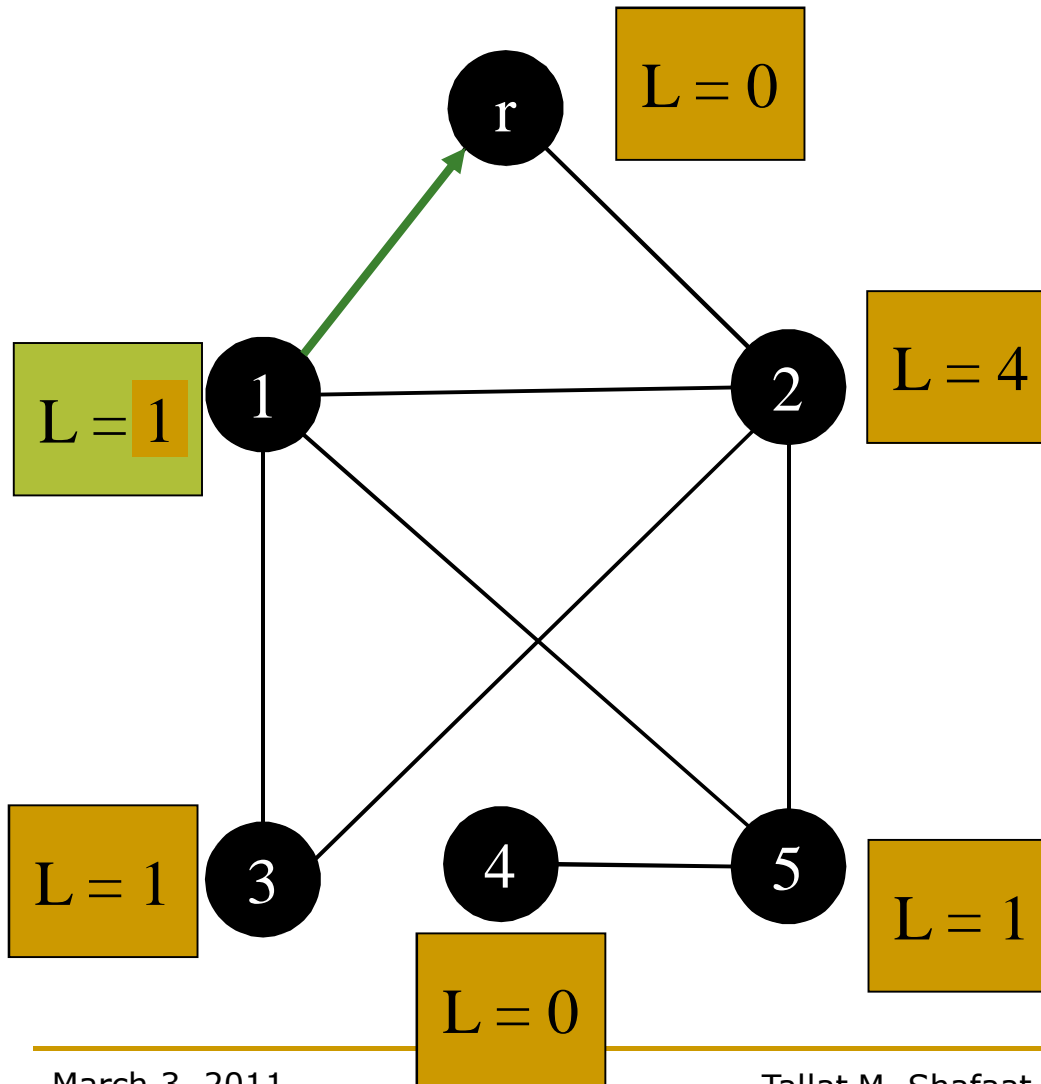
- Root: set $L=0$, $P=nil$

Example 2 - Sample Execution



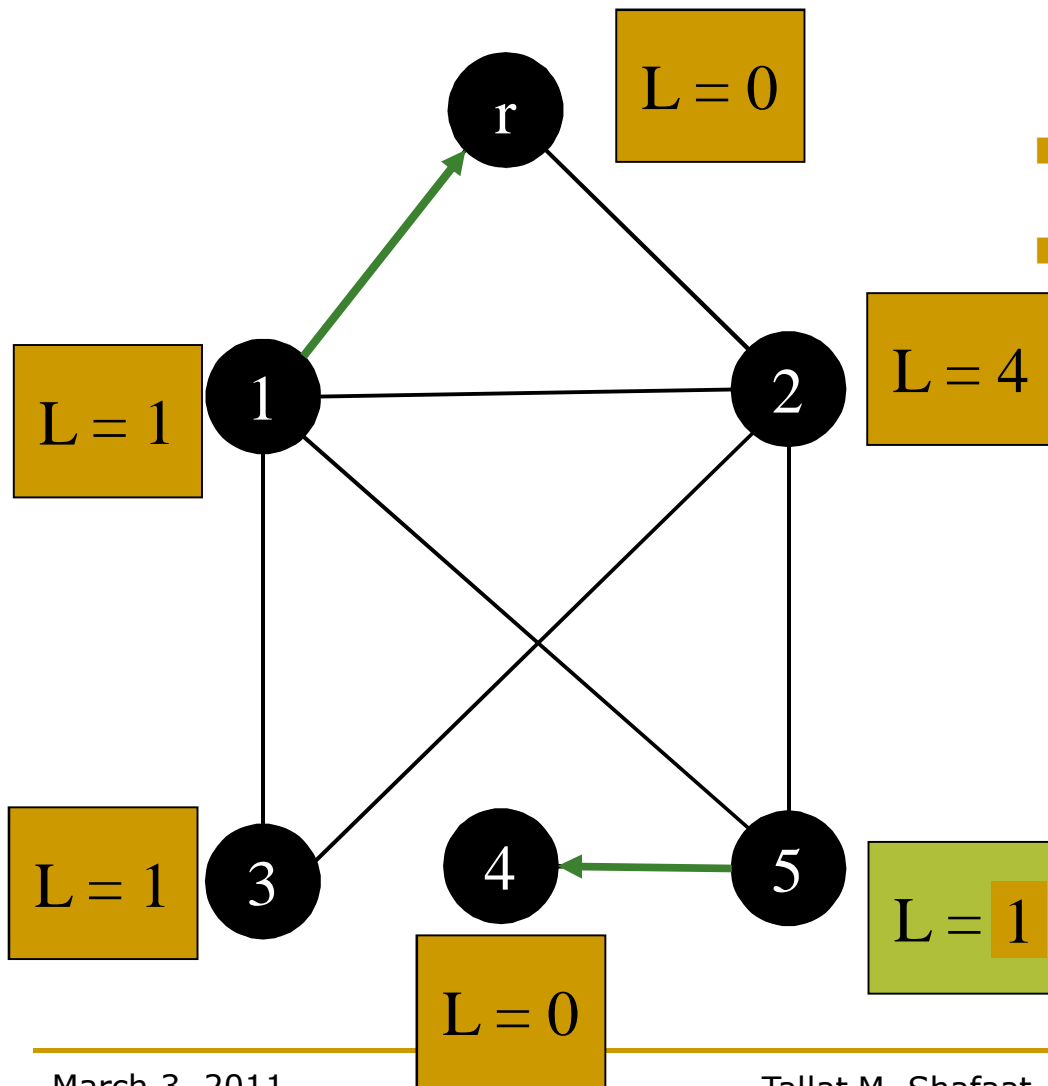
- Root: set $L=0$, $P=nil$
- Say nodes update in the following order
 - 1, 5, 2, 3, 4

Example 2 - Sample Execution



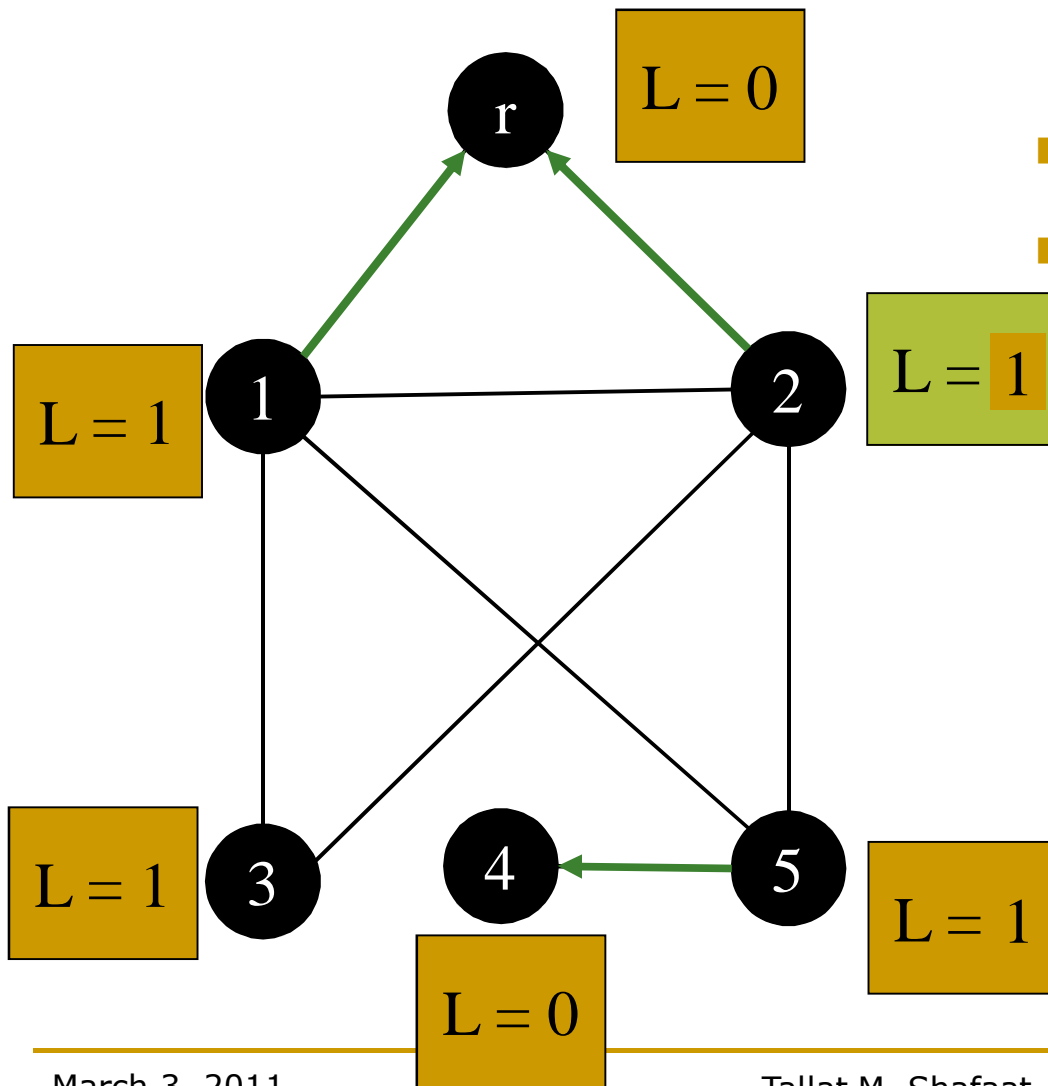
- Root: set $L=0$, $P=nil$
- Say nodes update in the following order
 - 1, 5, 2, 3, 4
 - $L=1+\min(\{0,1,4\})$

Example 2 - Sample Execution



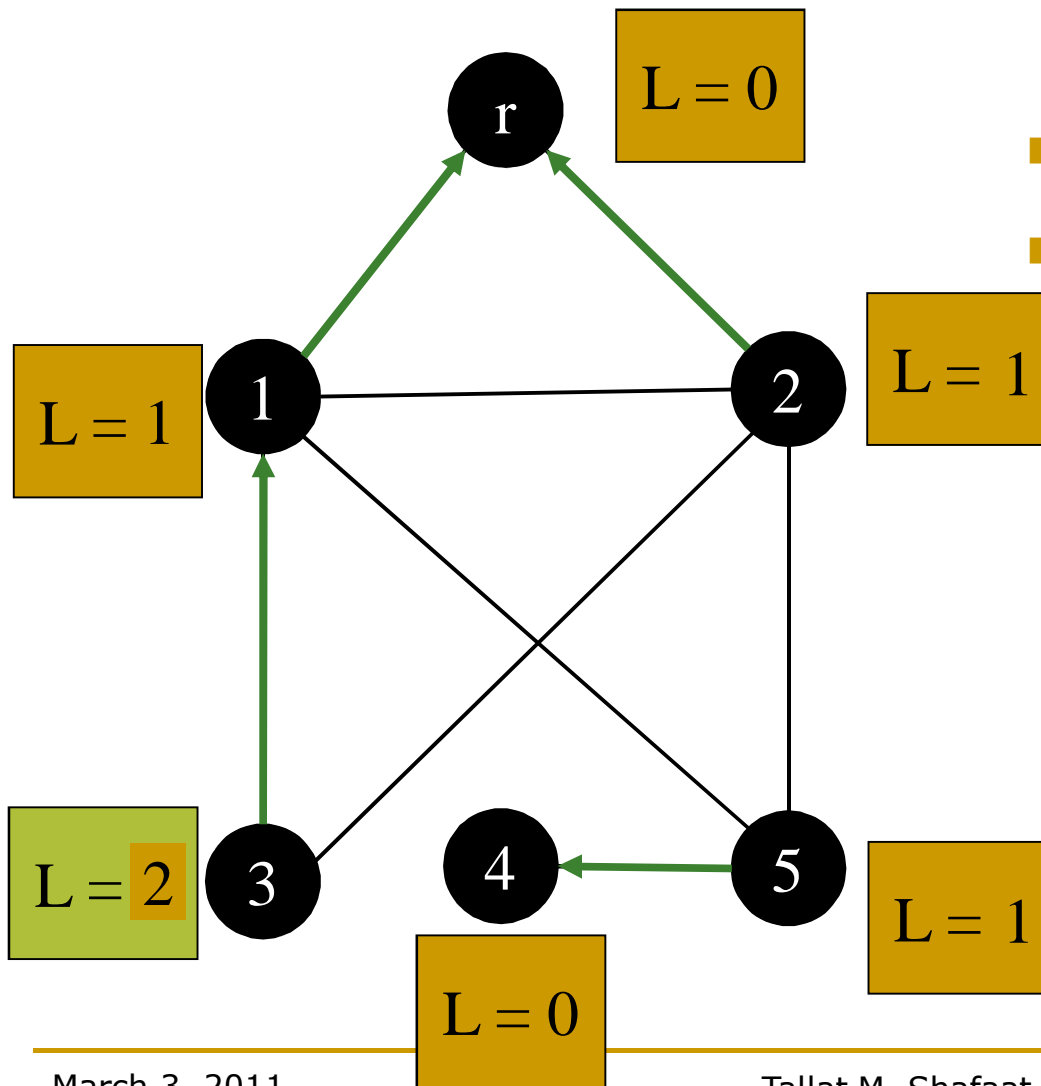
- Root: set $L=0$, $P=nil$
- Say nodes update in the following order
 - 1, 5, 2, 3, 4
 - $L=1+\min(\{0,1,4\})$

Example 2 - Sample Execution



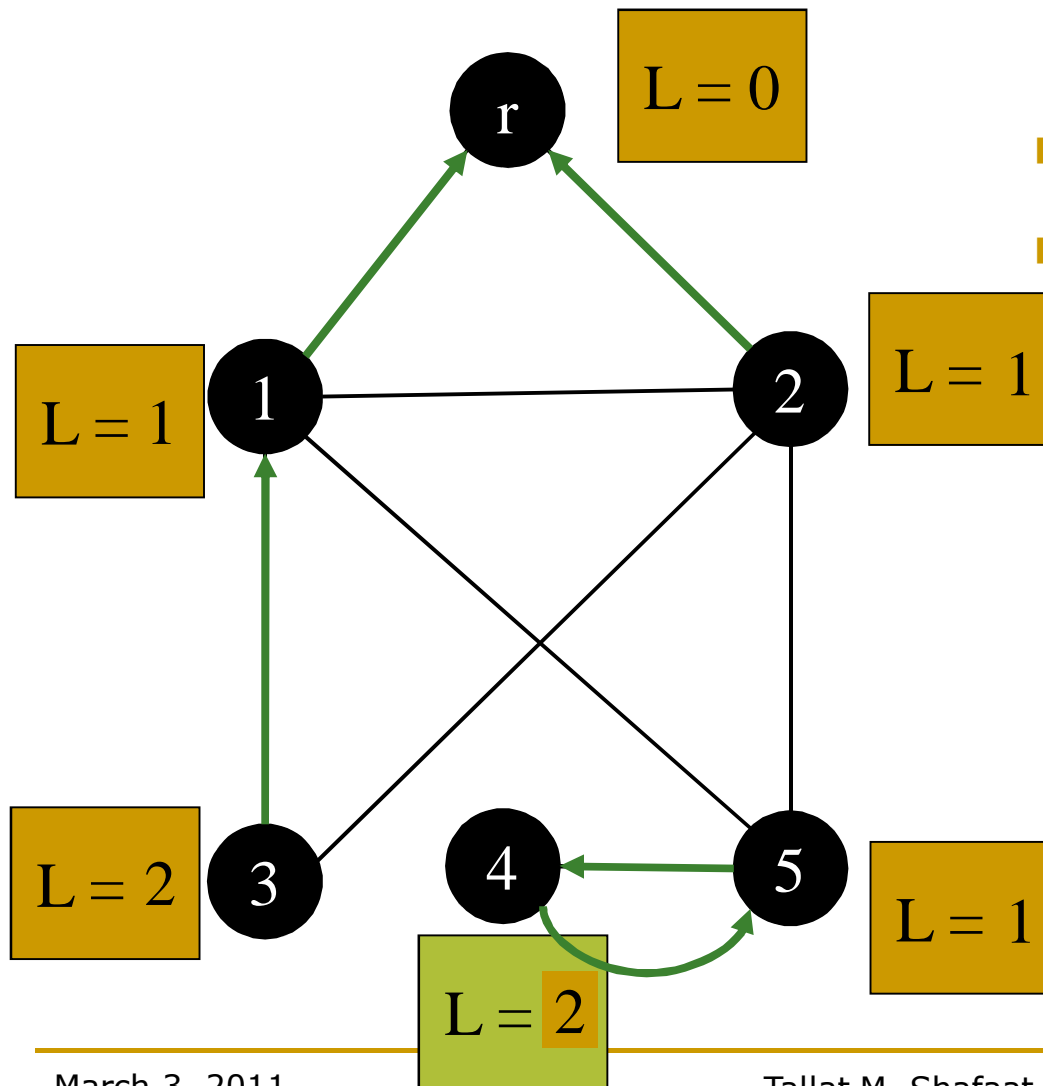
- Root: set $L=0$, $P=nil$
- Say nodes update in the following order
 - 1, 5, 2, 3, 4
 - $L=1+\min(\{0,1\})$

Example 2 - Sample Execution



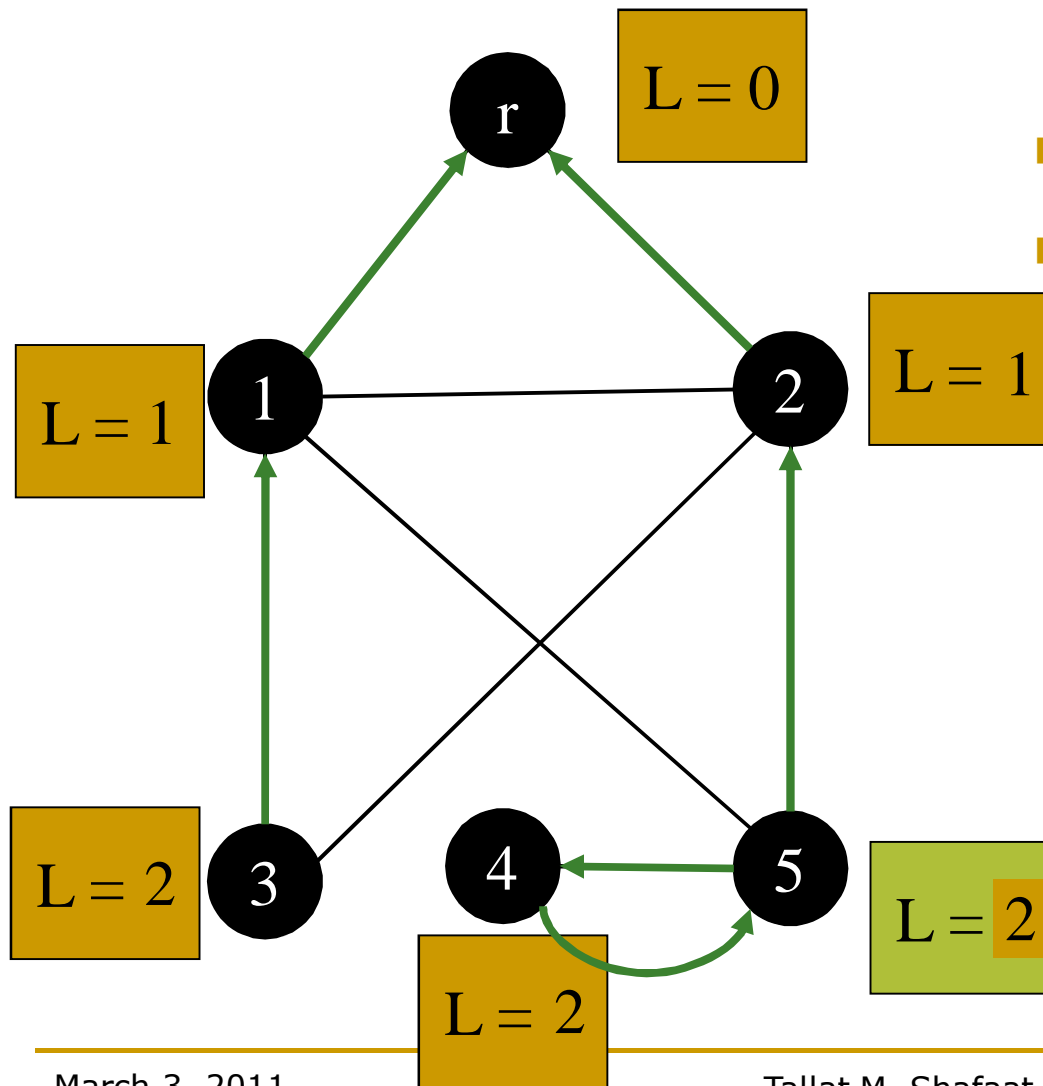
- Root: set $L=0$, $P=nil$
- Say nodes update in the following order
 - 1, 5, 2, 3, 4
 - $L=1+\min(\{1\})$
 - Multiple choices for parent. We use node with min id.

Example 2 - Sample Execution



- Root: set $L=0$, $P=nil$
- Say nodes update in the following order
 - 1, 5, 2, 3, 4
 - $L=1+\min(\{1\})$

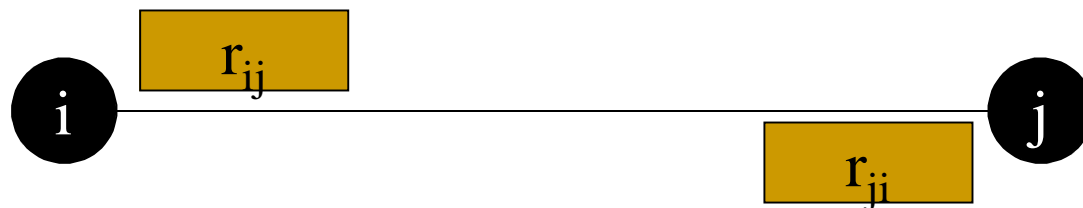
Example 2 - Sample Execution



- Root: set $L=0$, $P=nil$
- Say nodes update in the following order
 - 1, 5, 2, 3, 4
 - Later, **5** executes
 - $L=1+\min(\{2,1\})$

Model used

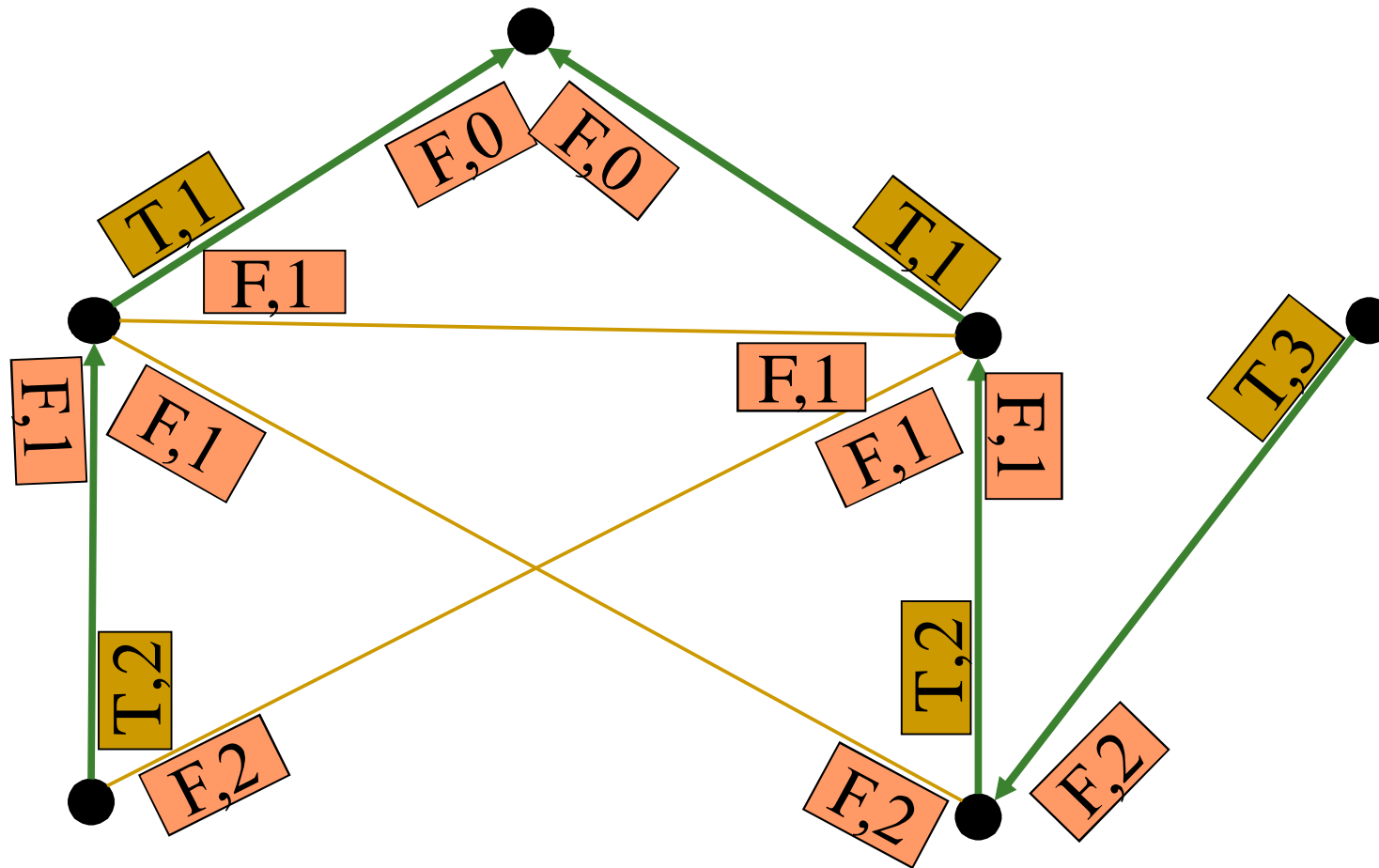
- Shared memory model
- A process i communicates with a neighbour process j by **writing** to a register r_{ij} and by **reading** r_{ji}
- For this talk, a register can store two values



Values in registers

- Each register r_{ij} stores two values:
 - Parent (boolean)
 - True if j is parent of i , false otherwise
 - Distance (integer)
 - $L(i)$, i.e. distance of i to root.
- Thus, the spanning tree is embedded in the register values

Example - BFS Spanning Tree

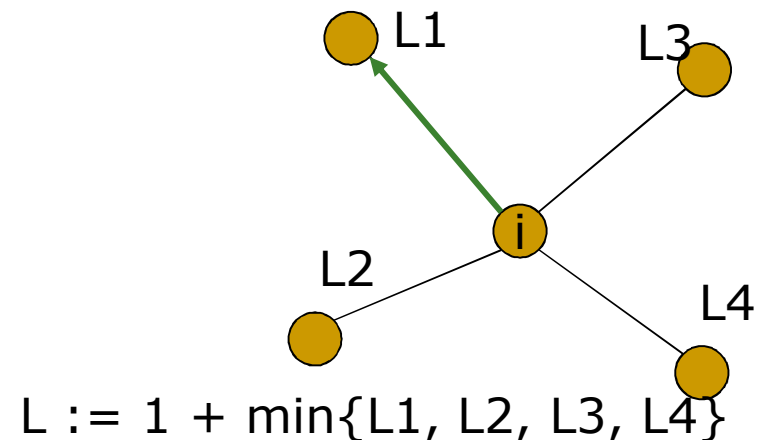


Algorithm - root

- while true
 - For each $m \in N(r)$
 - write $r_{rm} = [\text{false}, 0]$

Algorithm - $i \neq r$

- while true
 - $\forall m \in N(i)$ read distance in r_{mi} into lmidistance
 - $distance := 1 + \min(\{ lmidistance : \forall m \in N(i) \})$
 - foundParent := false
 - For each neighbour m
 - If not foundParent $lmidistance = distance - 1$
 - foundParent := true
 - Parent := m
 - write $r_{im} := [true, distance]$
 - Else
 - write $r_{im} := [false, distance]$



Definitions

- A **step** is a communication and computation event by a node
- Executions are divided into shortest **segments** such that in each segment
 - Each node performs at least one step
 - A segment is called a **round**
- Δ is the degree of the graph
 - maximum number of neighbours any node has
 - $\max(\delta_1, \delta_2, \dots, \delta_n)$
- **D** is diameter of the graph
 - i.e. maximum of shortest path distance between any pair of nodes

Algorithm - root

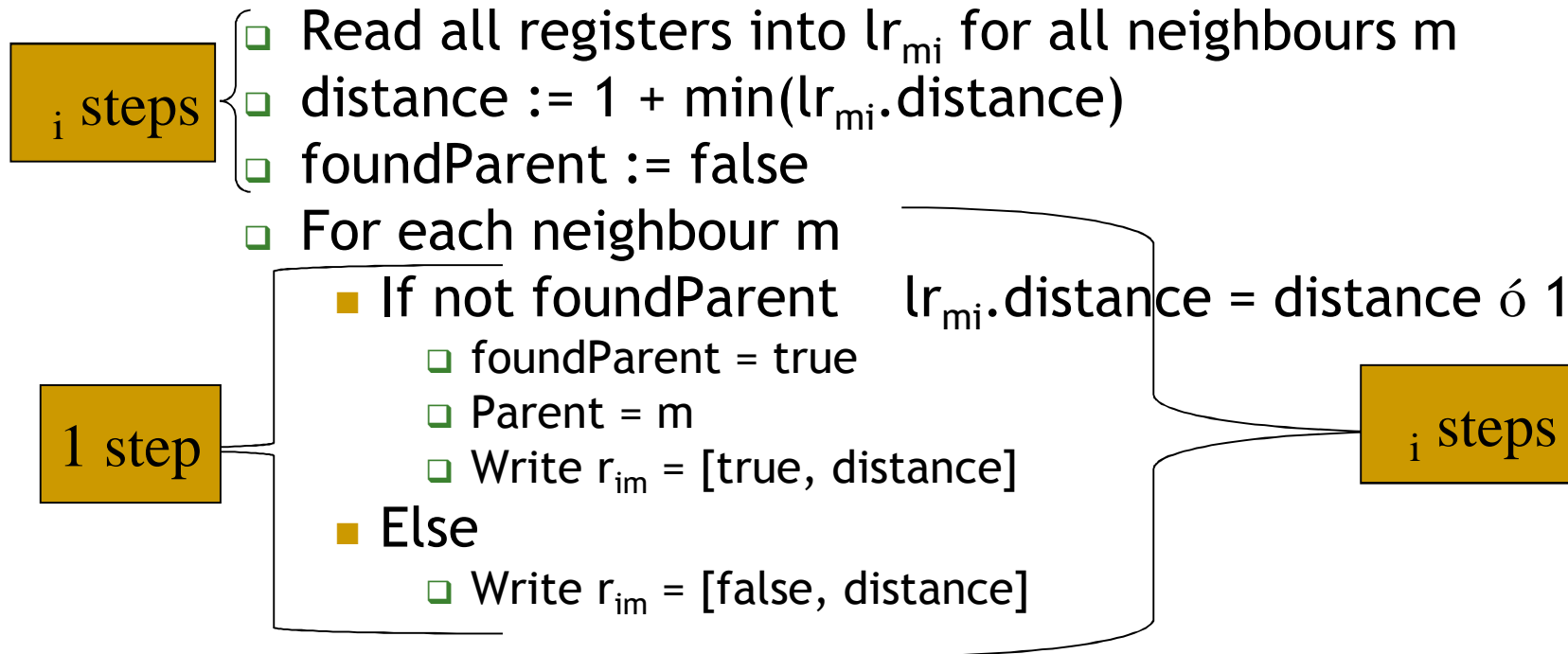
- while true

r steps { For each $m \in N(r)$
■ write $r_{rm} = [\text{false}, 0]$

- In $\Delta (\geq \delta_r)$ rounds, root must have executed an iteration at least once

Algorithm - $i \neq r$

- while true



- In 2Δ rounds, all nodes must have executed an iteration at least once

Proving Self-stabilization

- Prove that the algorithm, starting from an arbitrary state, produces a BFS spanning tree such that
 - $L(r) = 0$
 - $\forall i \in V \setminus \{r\}$ (non-root nodes)
 - $L(i) = L(P(i)) + 1$

Lemma

- In any execution, there exists a sequence

$$T_1 < T_2 < T_3 < \dots < T_D$$

such that after T_k ($1 \leq k \leq D$)

1. Every node i at distance $\leq k$ from root
 - Has $L(i) =$ shortest distance to root
 - Their parent variables form a BFS Tree
2. Every node i at distance $> k$ from root has $L(i) \geq k$

Proving Self-stabilization

- Distance of any node to root is at most D
- Thus, after $O(D\Delta)$ rounds, a BFS tree will be created

Proving Lemma

- By induction
- Prove the lemma for $k=1$
- Assume true for k , show that it will be true for $k+1$

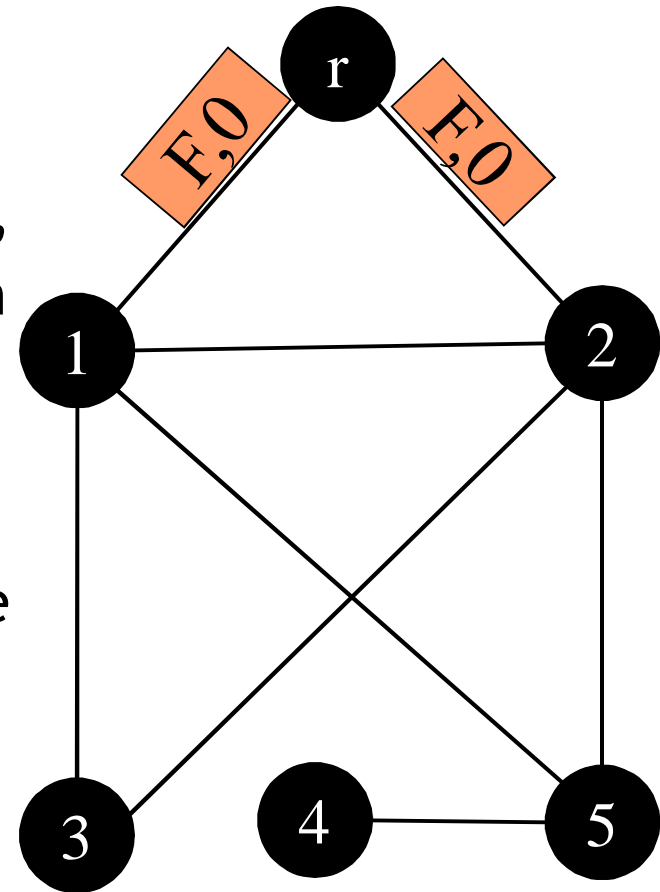
Base step (k=1)

- Show that after T_1 :
 1. Every node i at distance ≤ 1 from root will have
 - $L(i)$ = shortest distance to root
 - Parent variables form a BFS
 2. Every node i at distance > 1 from root has $L(i) \geq 1$

- Remember, $0 \leq L(i) < n$

Base step (k=1)

- Let $T_1 = \Delta + 4\Delta$ rounds
- Even if u start in any position in the iteration, after 2Δ , partial iteration will be done,
 - i.e. all nodes will be starting an iteration
- During this 2Δ , root will have wrote $[F, 0]$ in all registers
 - root requires Δ to complete one iteration

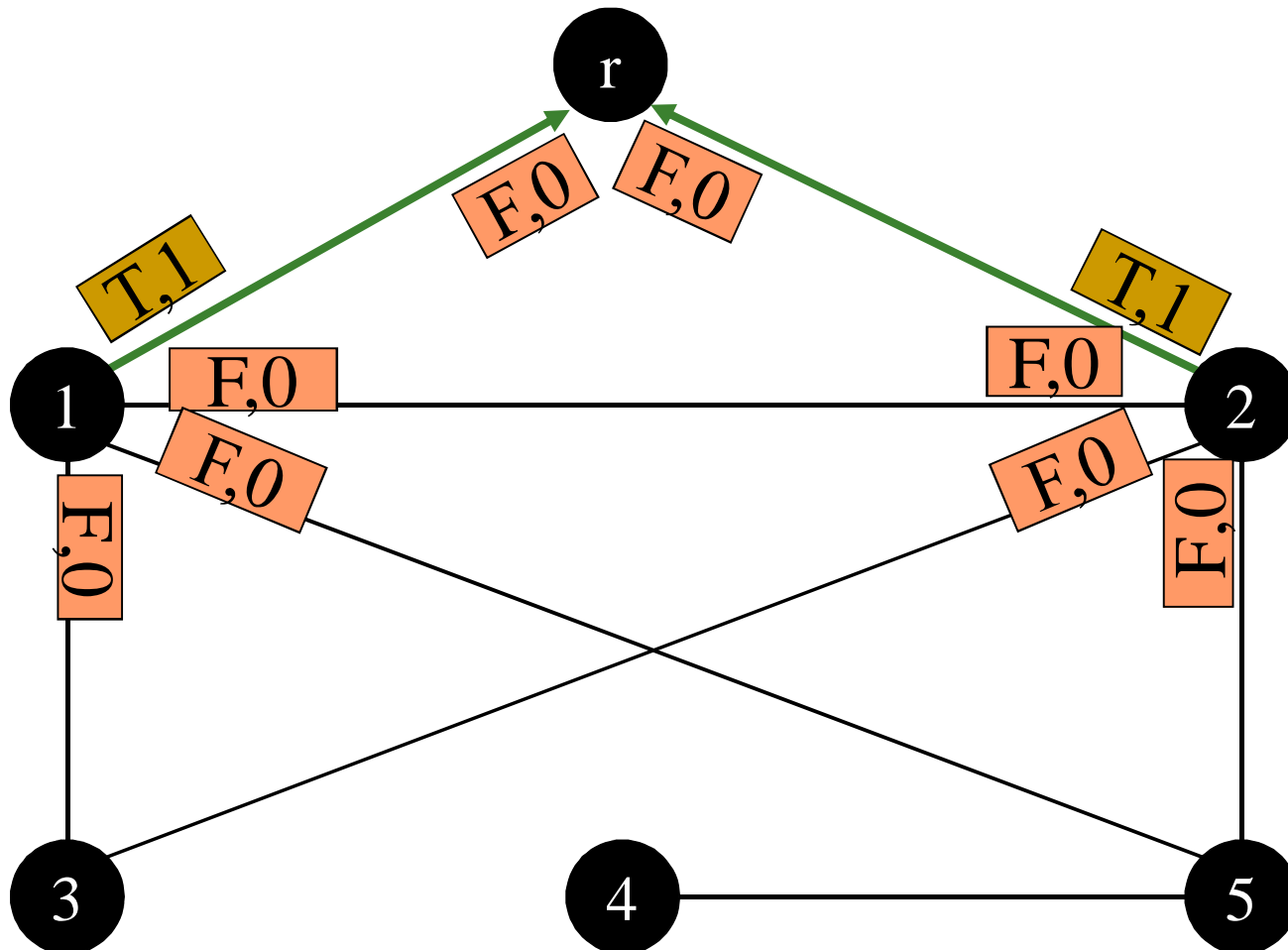


Base step ($k=1$)

distance = 1 + (some non negative number)

- After next Δ (3Δ),
 - all non-root nodes must have completed a read loop at least once and computed distance
 - Thus, all distances are greater than or equal to 1 (Assertion 2 proved)
- After next Δ (4Δ),
 - all non-root nodes must have completed a write loop at least once
 - Neighbours of root, must have wrote
 - $[T, 1]$ to register for root
 - $[F, 1]$ to register for all other neighbours
 - Assertion 1 proved
- After next Δ (5Δ),
 - all non-root nodes must have completed a read loop at least once

After base step / T_1



Induction step

- Assume for k , prove for $k+1$

- Assume its true at T_k that
 1. Every node i at distance $\leq k$ from root has
 - $L(i) = \text{shortest distance to root} \leq k$
 - Parent variables form a BFS
 2. Every node i at distance $> k$ from root has $L(i) \geq k$

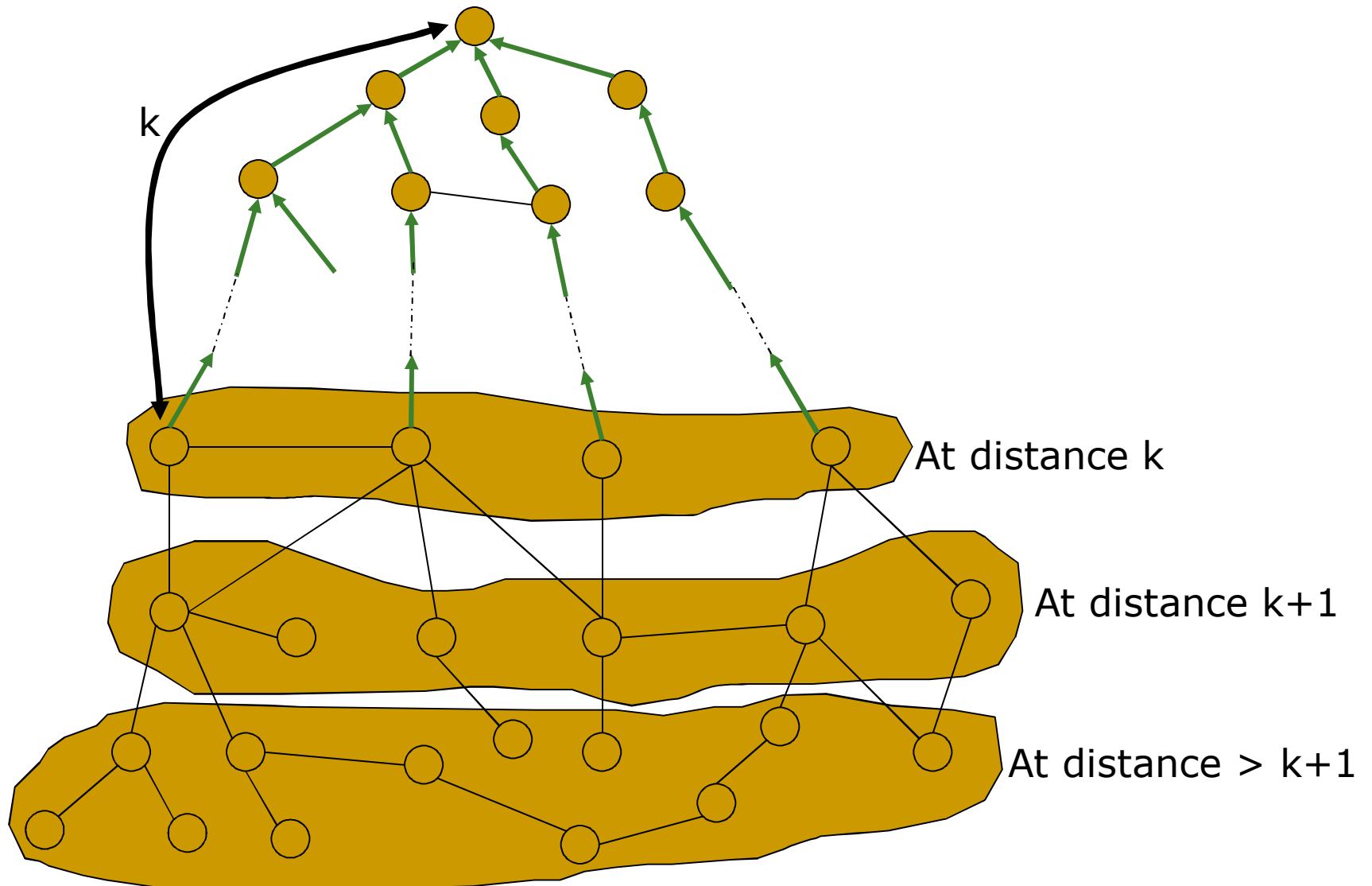
- This basically means BFS tree is correct till depth k

- Prove that at T_{k+1}
 1. Every node i at distance $\leq k+1$ from root will have
 - $L(i) = \text{shortest distance to root} \leq k+1$
 - Parent variables form a BFS
 2. Every node i at distance $> k+1$ from root has $L(i) \geq k+1$

Induction step

- Let $T_{k+1} = T_k + 2\Delta$
- Consider the execution after end of asynchronous round T_k
- After next Δ rounds, all non-root nodes must have
 - calculated their parent, and
 - executed write [parent, distance] to all neighbours at least once
- After next Δ rounds, all non-root nodes must have
 - executed read [parent, distance] from all neighbours at least once, and
 - computed their distance
- Now, consider two cases
 - nodes at distance $\leq k+1$
 - nodes at distance $> k+1$

Induction step - Induction



Induction step

- Consider node i at distance $\leq k+1$ from root
 - i has at least one neighbour j such that j 's distance to root $d \leq k$ from root and no neighbour closer to root
 - By induction hypothesis,
 - j has converged, i.e. j 's register has correct distance and parent values
 - All other neighbours of i have register values with distances $\geq d$
 - Thus, i will correctly compute its distance and parent

Induction step

- Consider node i at distance $> k+1$ from root
 - Every neighbour of i is at distance $\geq k+1$ from root
 - By induction hypothesis, all their registers have values $\geq k$
 - Thus, i computes distance $\geq k+1$

Proving Self-stabilization

- Distance of any node to root is at most D
- Thus, after $O(D\Delta)$ rounds, a BFS tree will be created