fastos.org/molar

National Science Foundation
WHERE DISCOVERIES BEGIN

Office of Science
U.S. DEPARTMENT OF ENERGY

Scalable, Fault-Tolerant Membership for MPI Tasks on HPC Systems

Jyothish Varma[1], Chao Wang[1], Frank Mueller[1], Christian Engelmann[2], Stephen L. Scott[2]

[1]North Carolina State University, Department of Computer Science

[2]Oak Ridge National Laboratory, Computer Science and Mathematics Division

NC STATE UNIVERSITY

OAK RIDGE NATIONAL LABORATORY
MANAGED BY UT-BATTELLE FOR THE DEPARTMENT OF ENERGY

Process Group Membership for MPI

- Objective: To tolerate faults in an MPI job in a scalable fashion
- Group Membership
  - Domain where members can join / leave
    - Associate ID w/ every member in domain
- Group Membership Service
  - Tracks active tasks (processes/nodes)
    - Tasks communicate, coordinate execution & termination
  - Inform group members of
    - departure of failed nodes
    - arrival of new/revived nodes
  - View := Set of active and connected processes
  - Used by application layer that relies on this service

Our Approach

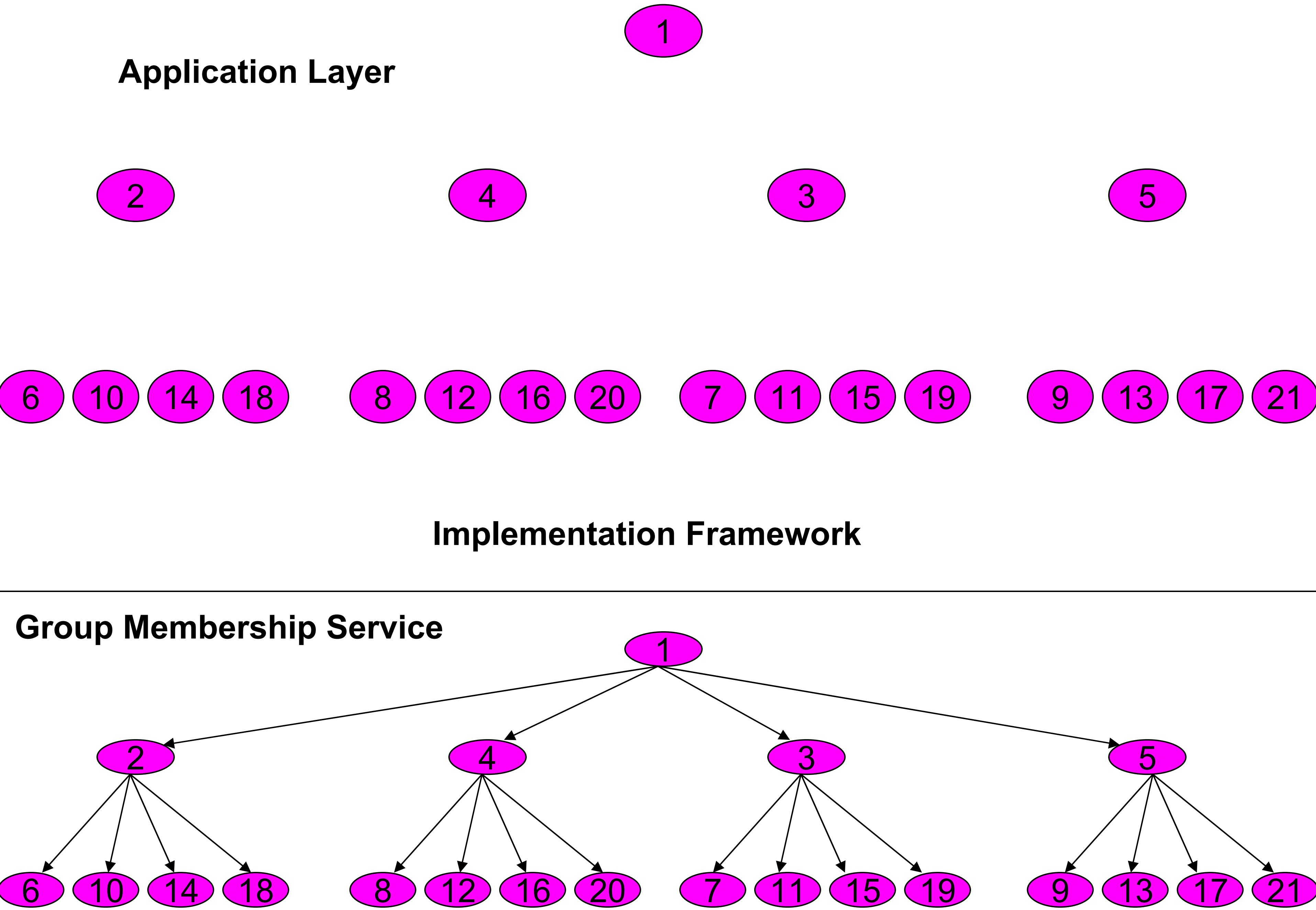
- Implemented group membership within runtime layer as service
  - Why ?
    - Modification to application is minimal
    - Application layer can be captured adequately
- Integrating Membership Service w/ BLCR (Berkeley Lab Checkpoint/Restart Mechanism)
  - Benefit: Node failure now handled w/o restarting MPI job
- Membership service maintains a consistent view of system.
- Communication only b/w processes that share same view

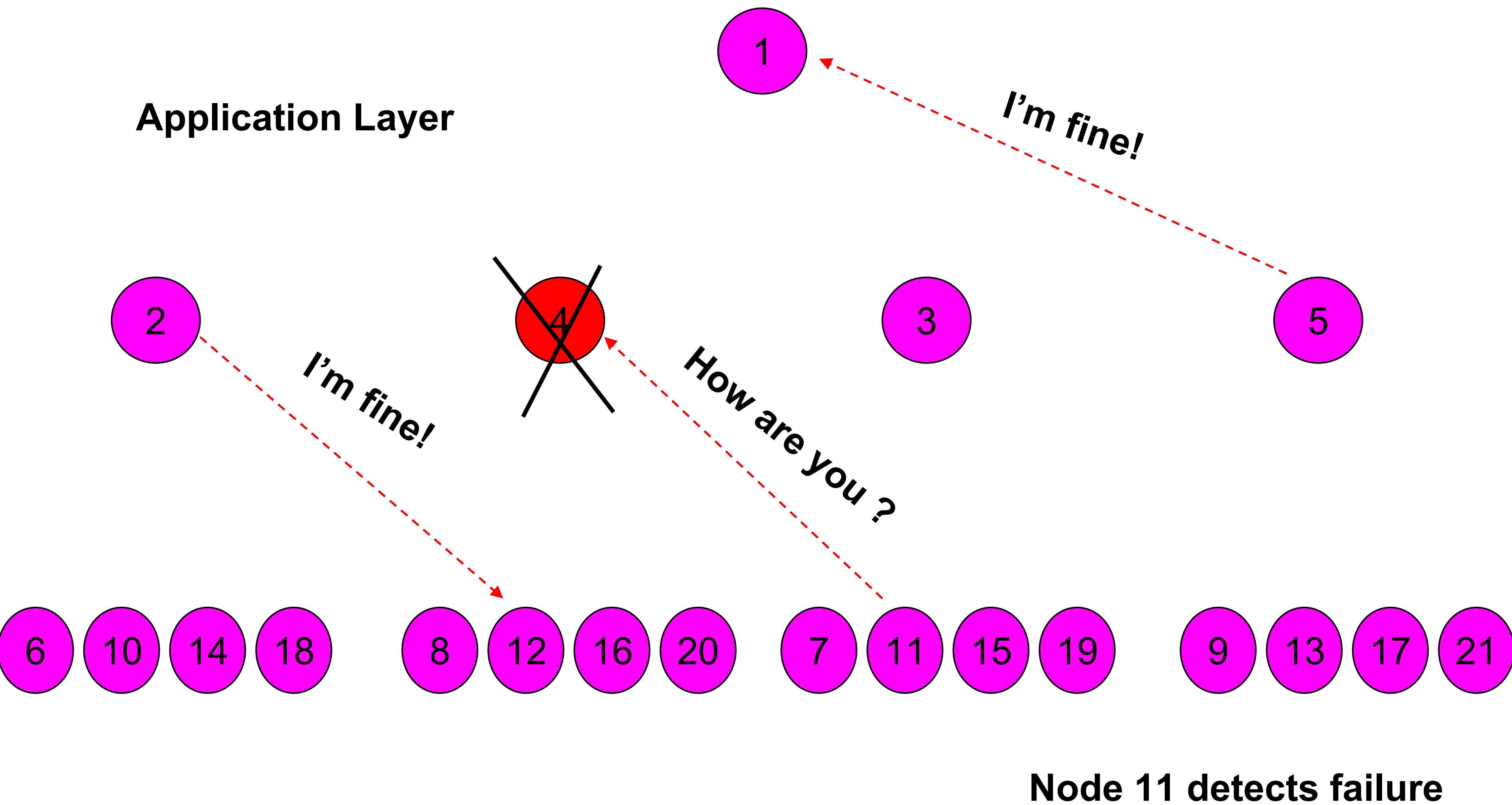
Assumptions and Fault Handling

- Assumptions
  - Execution Integrity
  - Message Uniqueness
  - Delivery Integrity
  - Same view delivery
- Fault Detection
  - External detection mechanism
    - Hardware health monitoring (e.g., IPMI)
    - Software health monitoring (e.g., heartbeat/timeouts)
- Our fault detection model
  - Fault detector based on time out mechanism
    - Link failure handled like a node failure

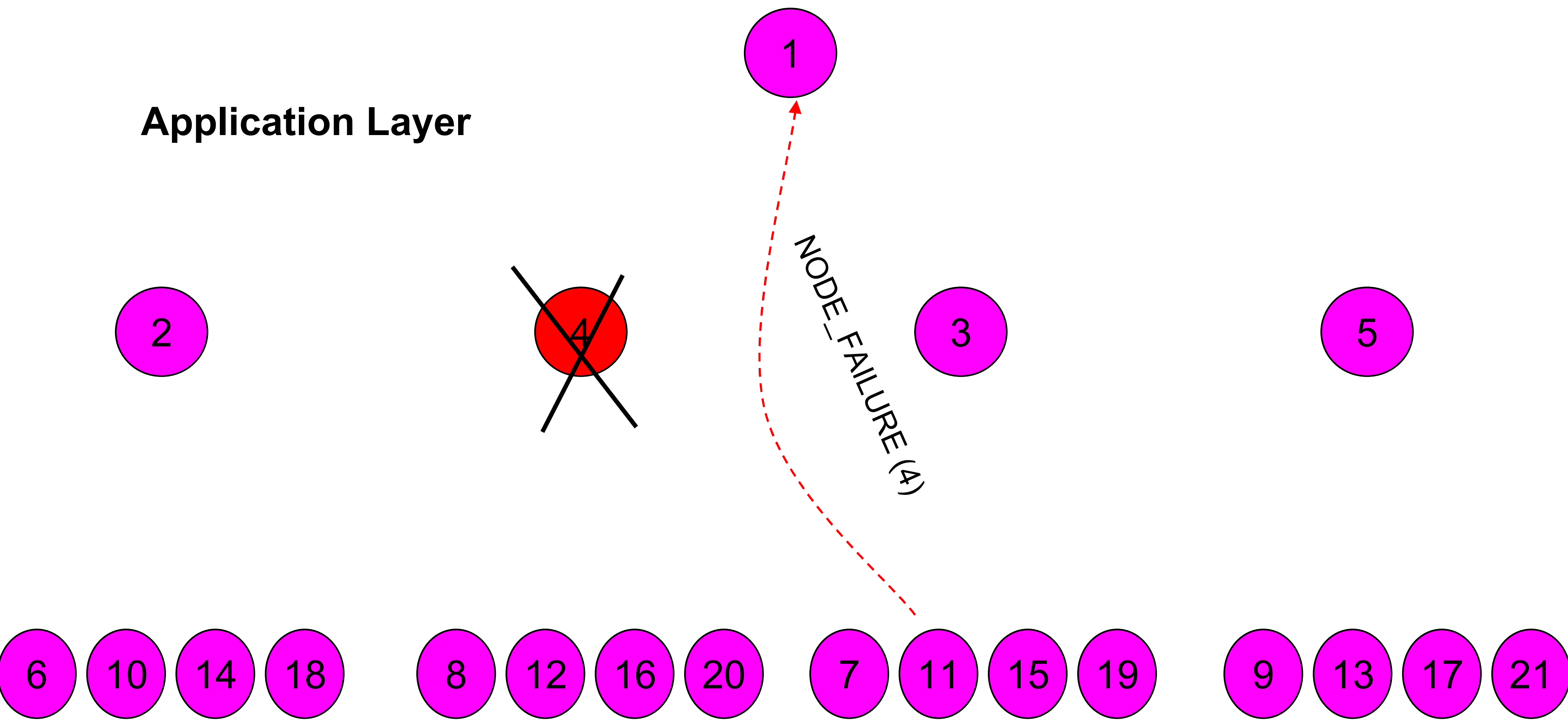
Group Membership Implementation

- Application Layer
  - applications communicate through simple message exchange
  - application may be MPI layer application
- Service Layer
  - Keeps group members up to date when *view* changes
  - Installs new *view* when *view* change message arrives
  - Protocols are pluggable
- Implementation details
  - Utilizes radix tree, default view on startup
  - Configurable
  - Extremely scalable
  - Fully decentralized


Application Layer
1
2
4
3
5
6
10
14
18
8
12
16
20
7
11
15
19
9
13
17
21
Implementation Framework
Group Membership Service
1
2
4
3
5
6
10
14
18
8
12
16
20
7
11
15
19
9
13
17
21

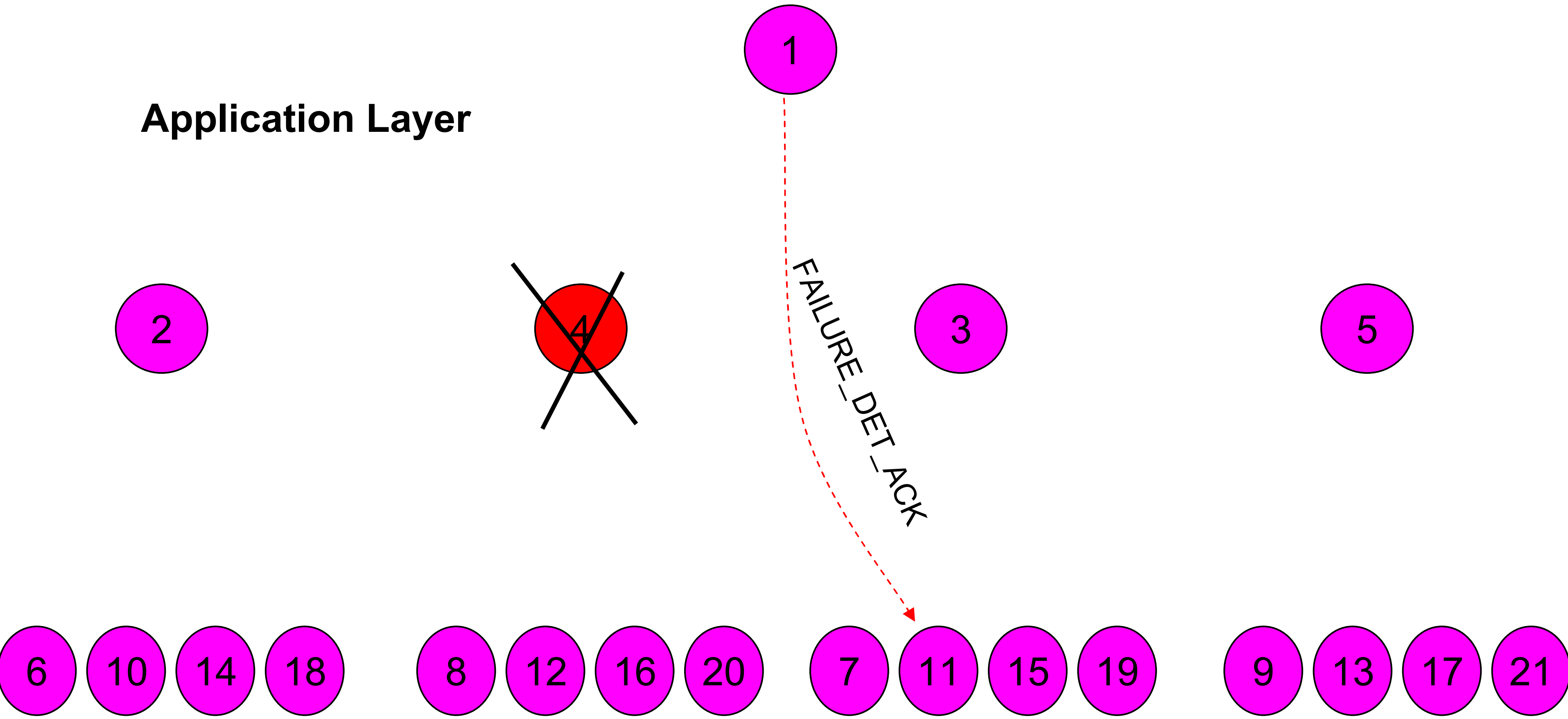


Application Layer
1
I'm fine!
2
4
3
5
I'm fine!
How are you ?
6
10
14
18
8
12
16
20
7
11
15
19
9
13
17
21
Node 11 detects failure



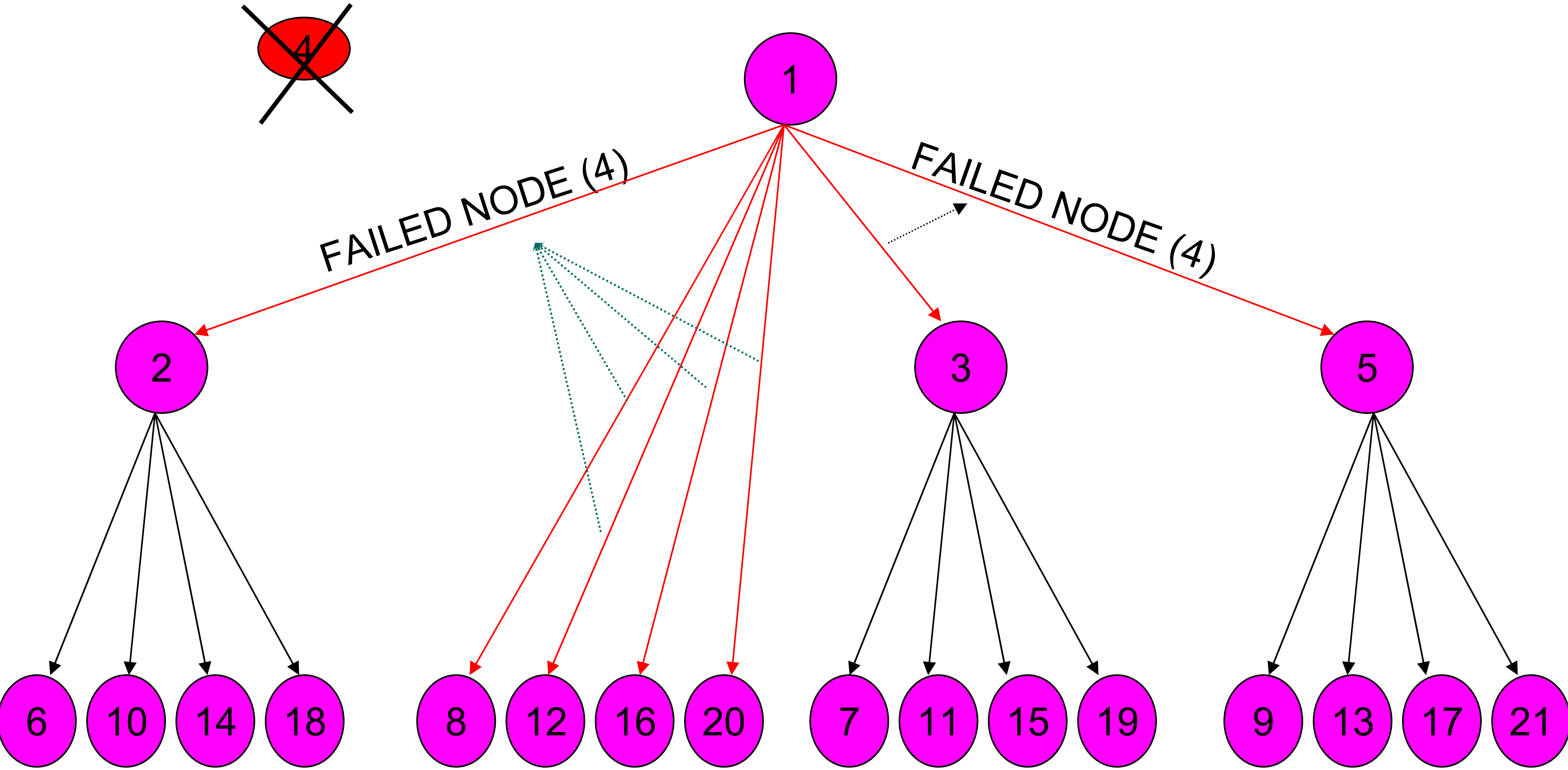
Application Layer
1
2
4
3
5
NODE_FAILURE (4)
6
10
14
18
8
12
16
20
7
11
15
19
9
13
17
21

Application Layer

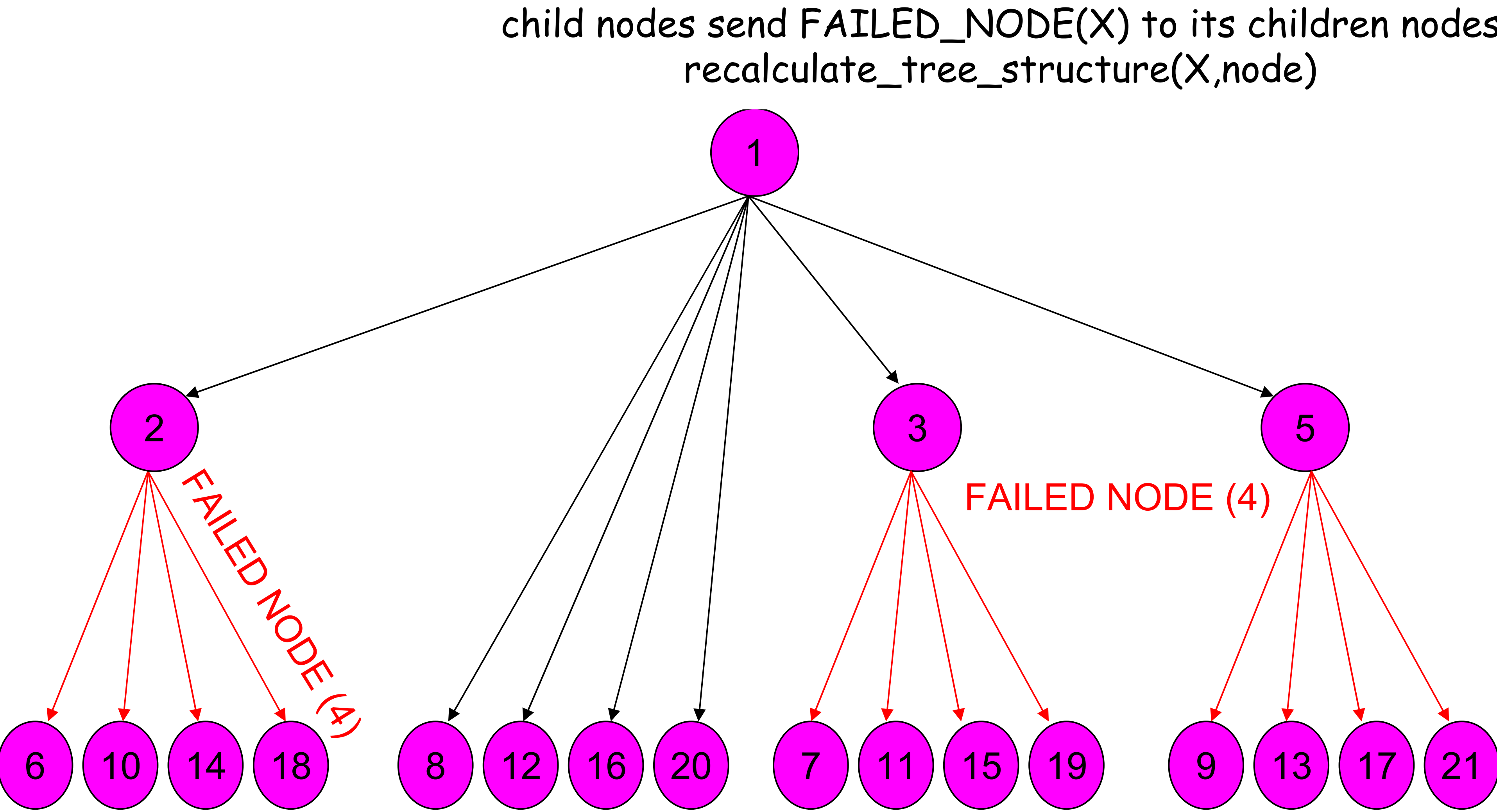


Group Membership Service

root sends FAILED_NODE(X) to children nodes
recalculate_tree_structure(X,node)

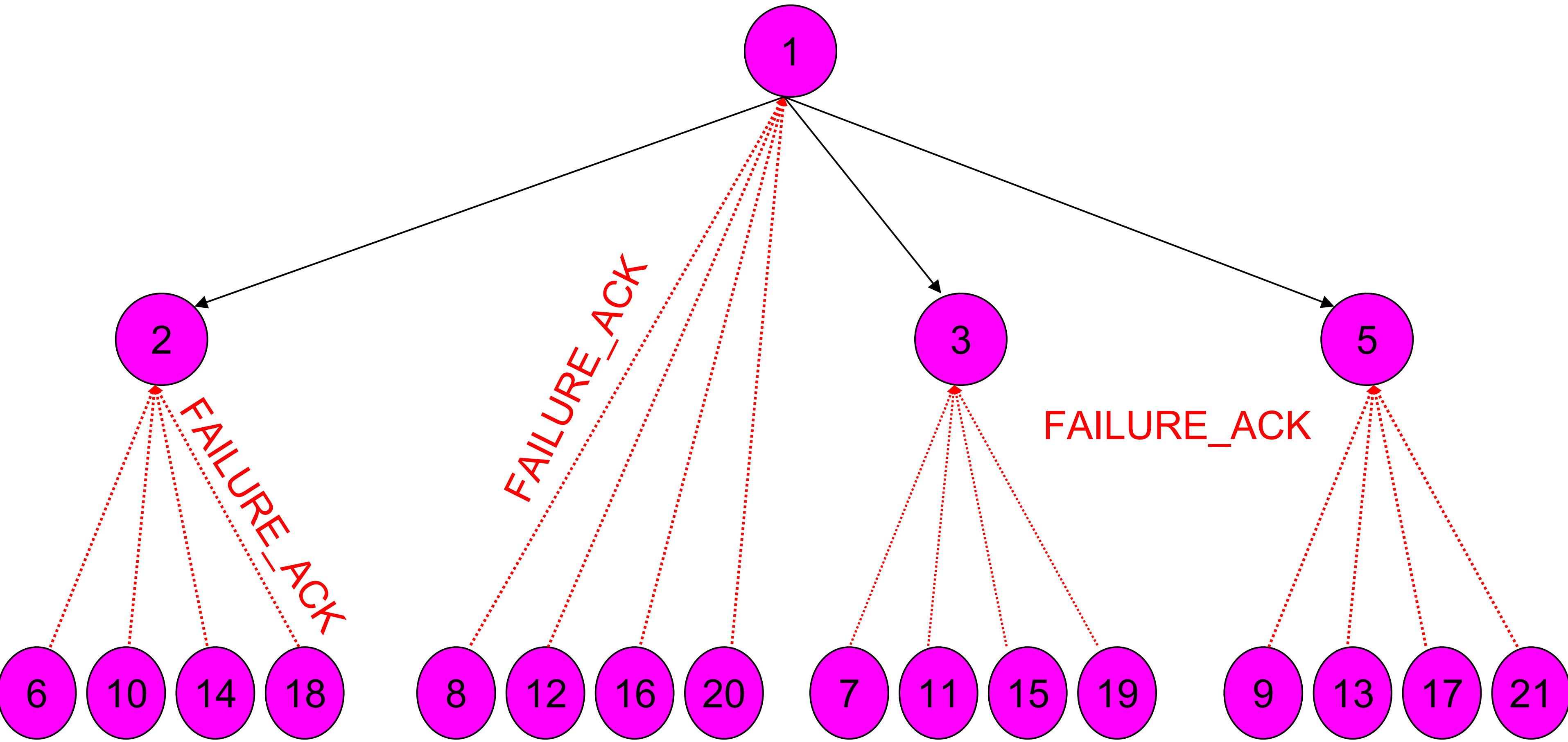


Group Membership Service



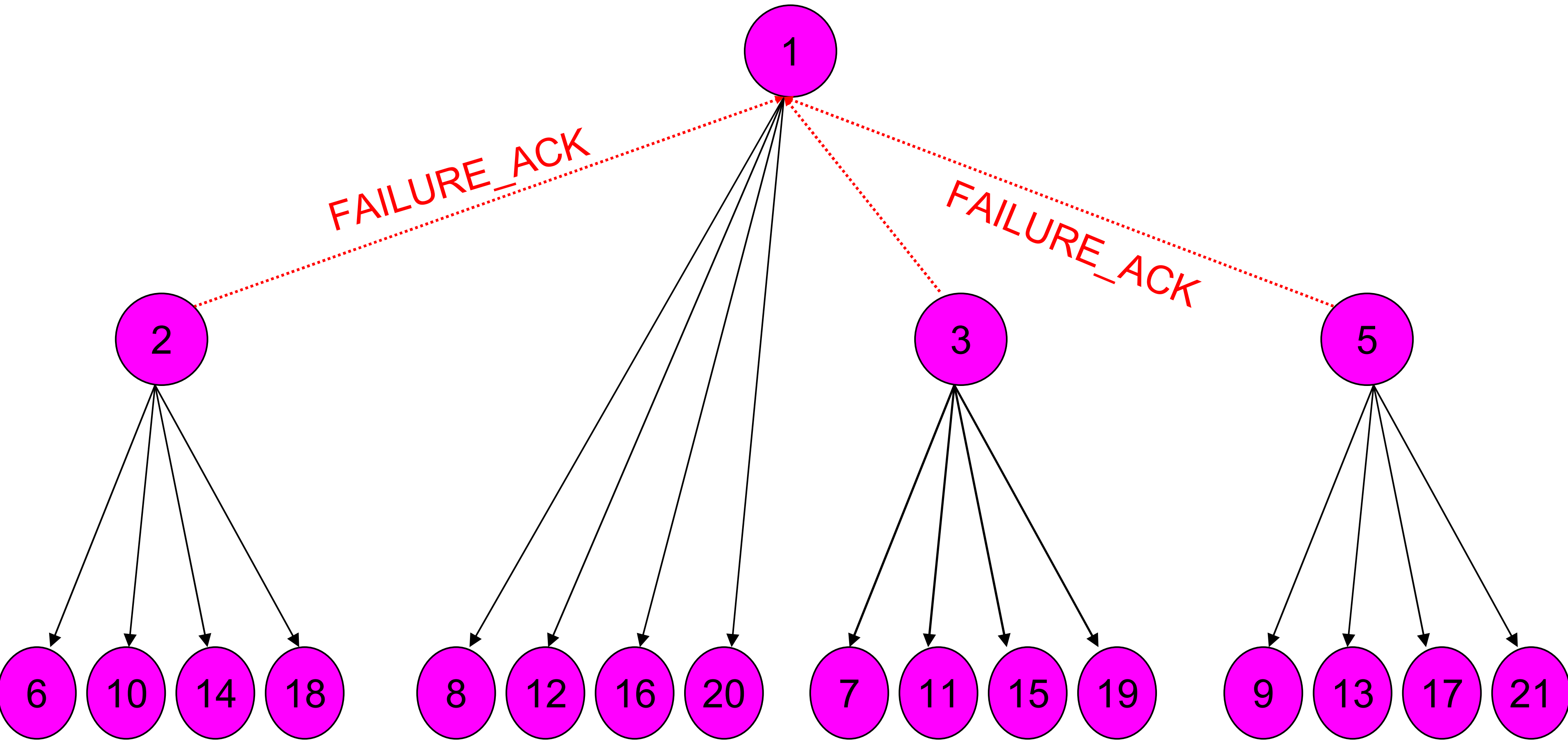
Group Membership Service

child nodes send FAILURE_ACK
to its root node



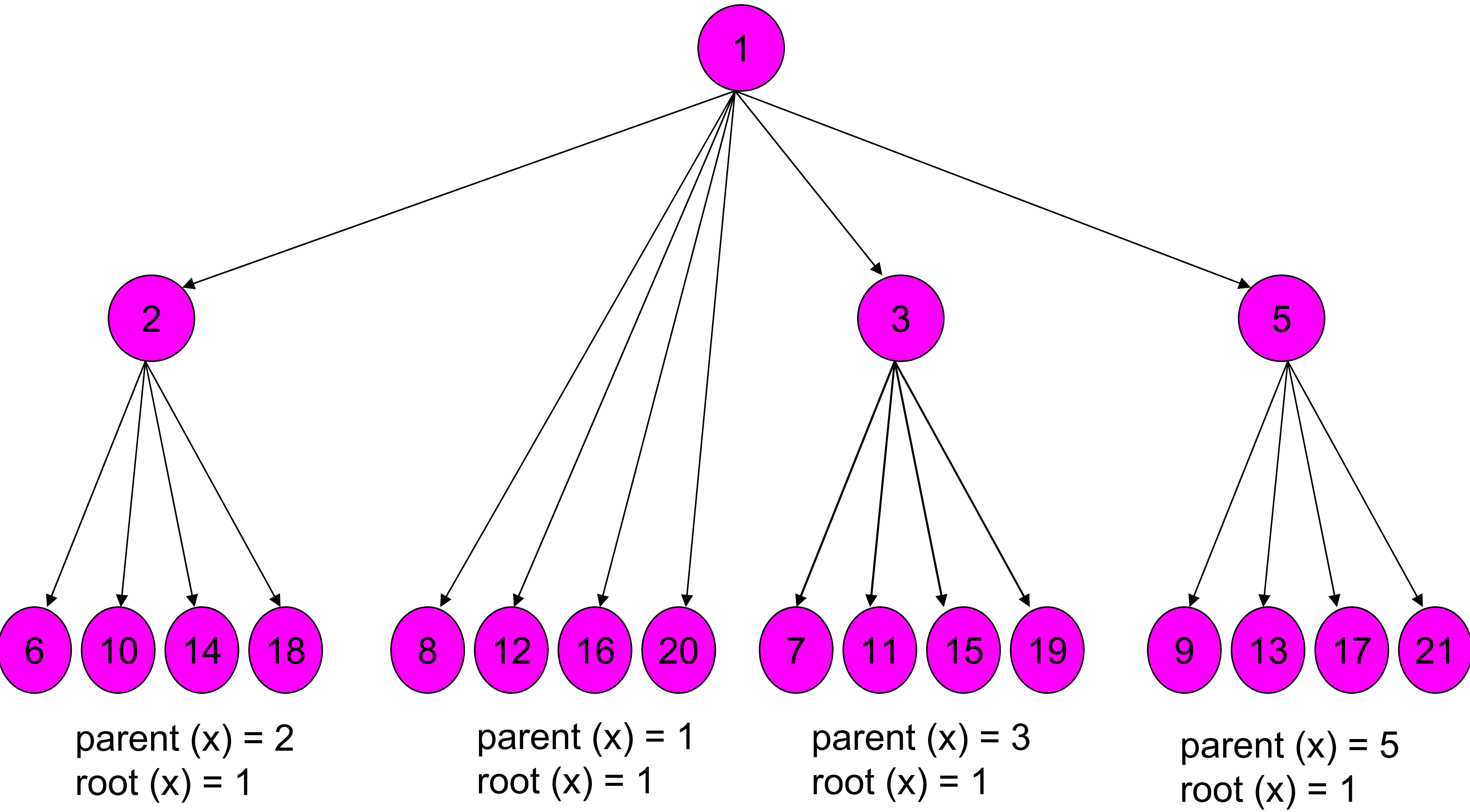
Group Membership Service

child nodes send FAILURE_ACK to root node



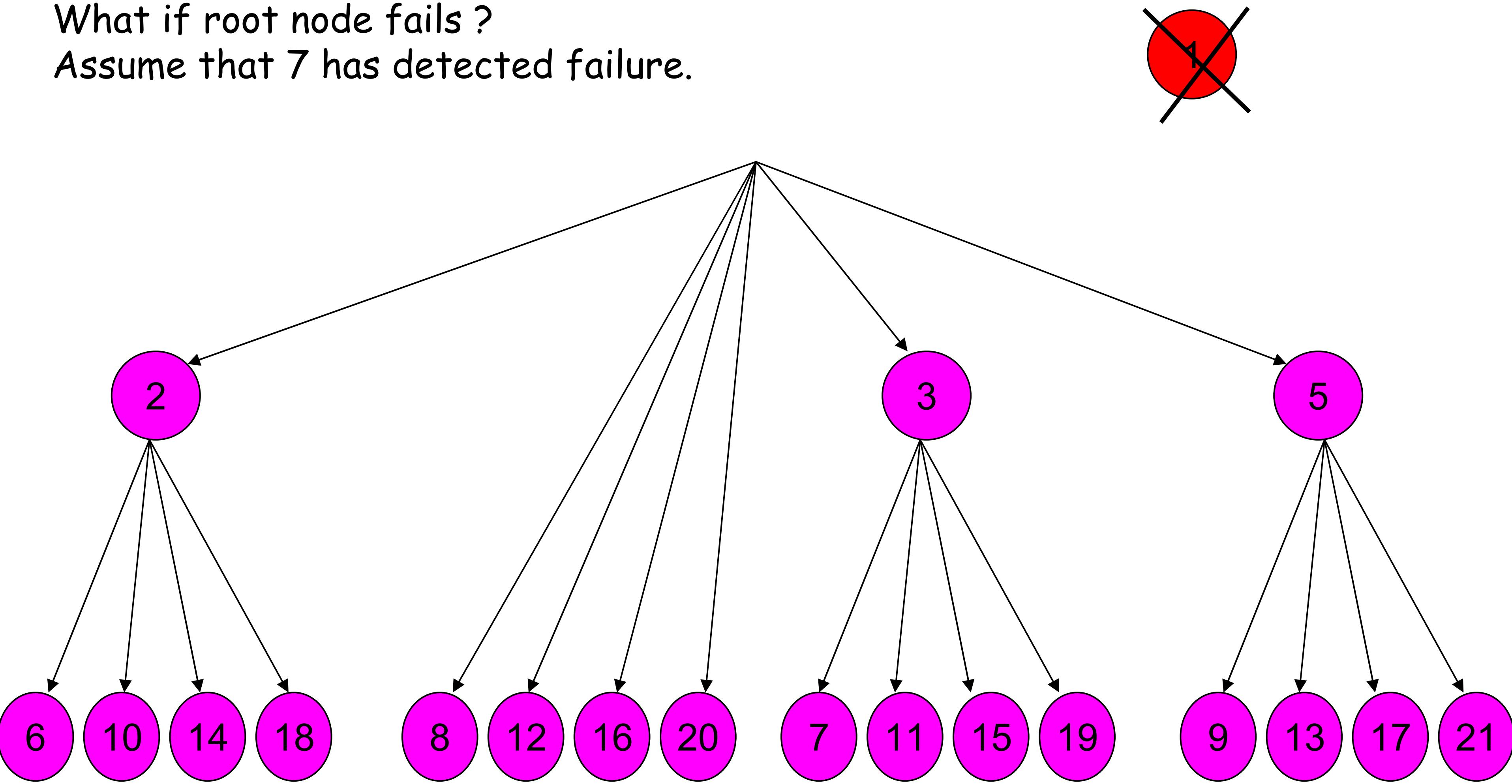
Group Membership Service

System restores to stable state when number of FAILURE_ACK received by each root node = number of its children



Group Membership Service

What if root node fails ?
Assume that 7 has detected failure.



parent (x) = 2
root (x) = 1

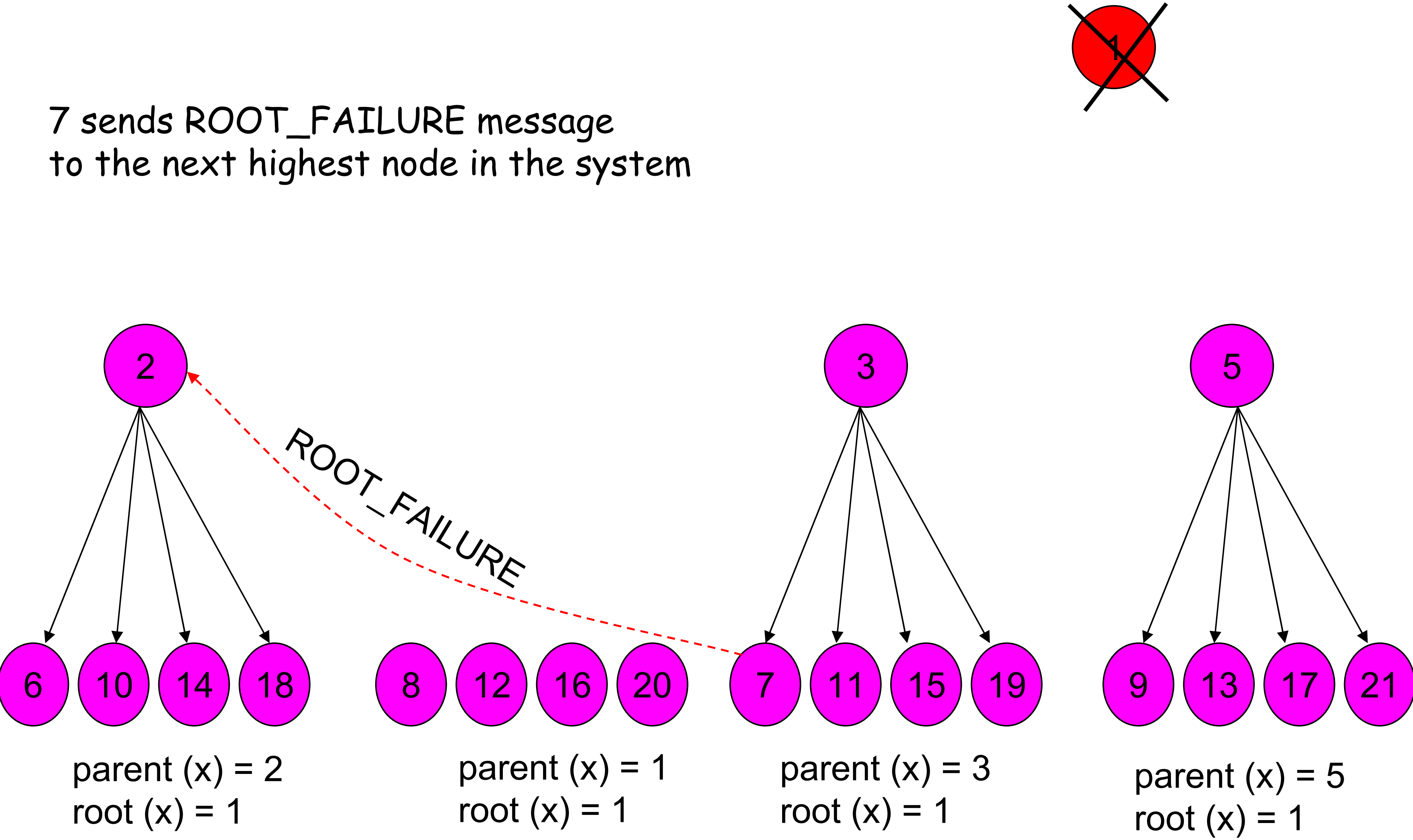
parent (x) = 1
root (x) = 1

parent (x) = 3
root (x) = 1

parent (x) = 5
root (x) = 1

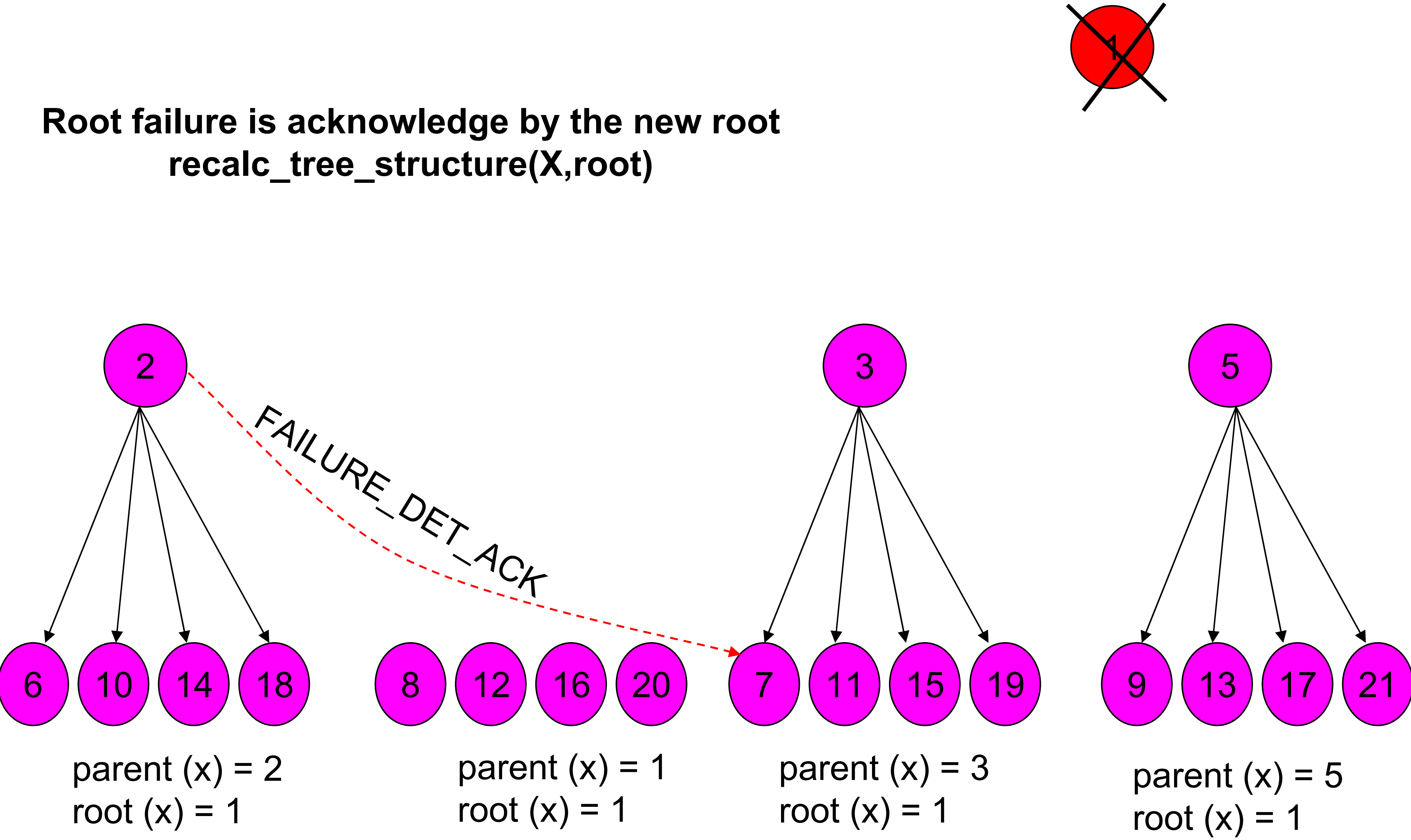
Group Membership Service

7 sends ROOT_FAILURE message
to the next highest node in the system



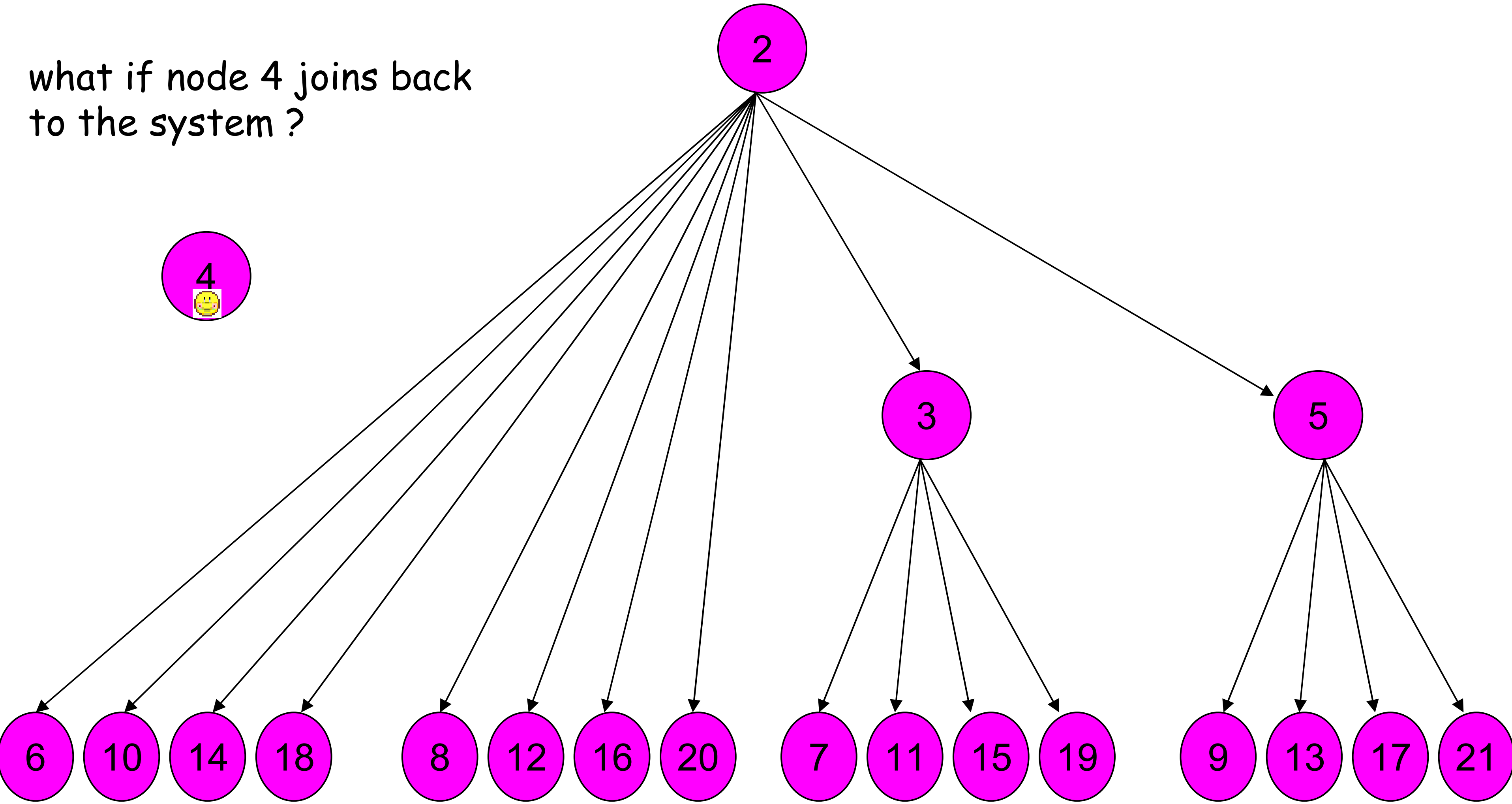
Group Membership Service

Root failure is acknowledge by the new root
recalc_tree_structure(X,root)



Group Membership Service

what if node 4 joins back
to the system ?



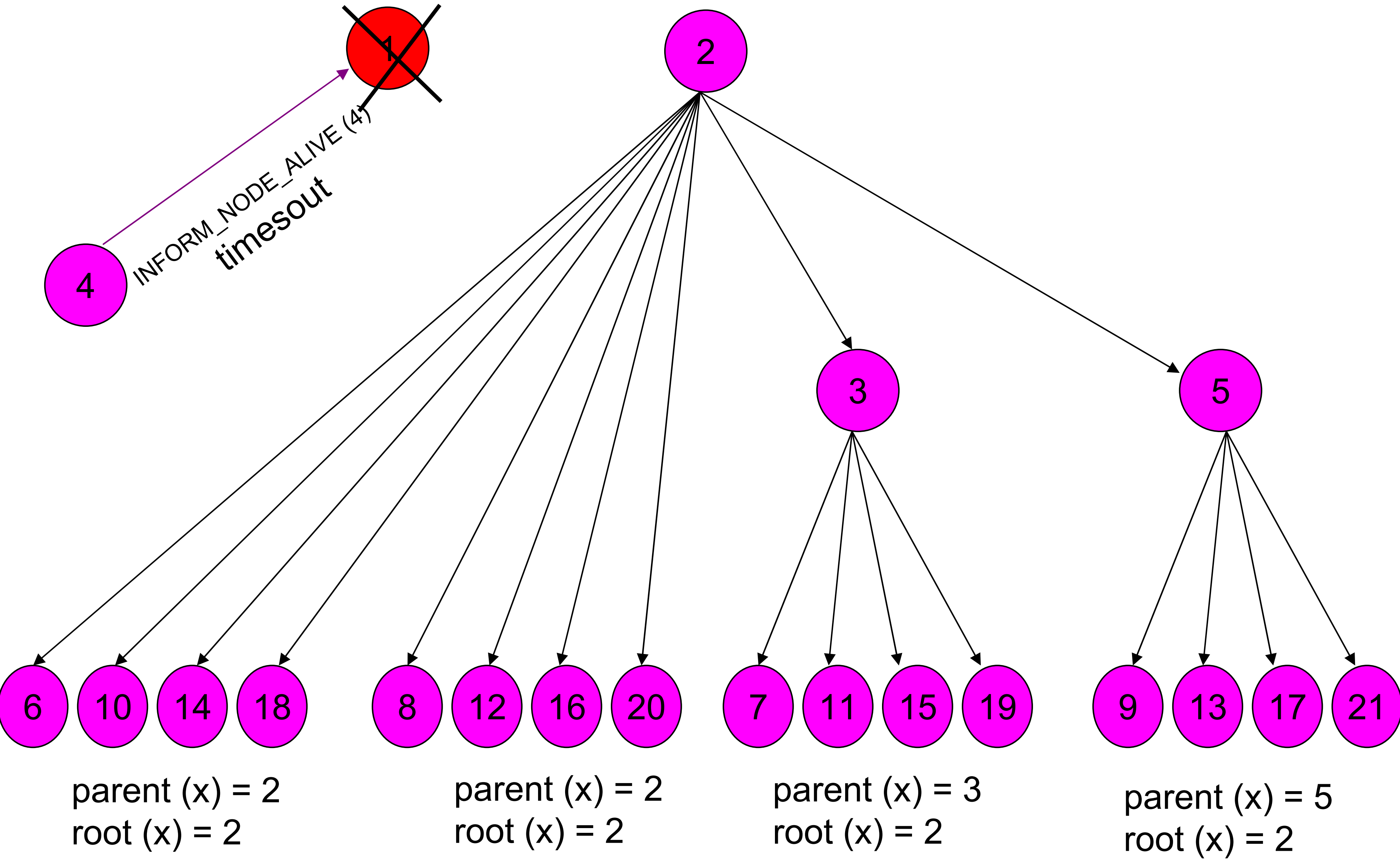
parent (x) = 2
root (x) = 2

parent (x) = 2
root (x) = 2

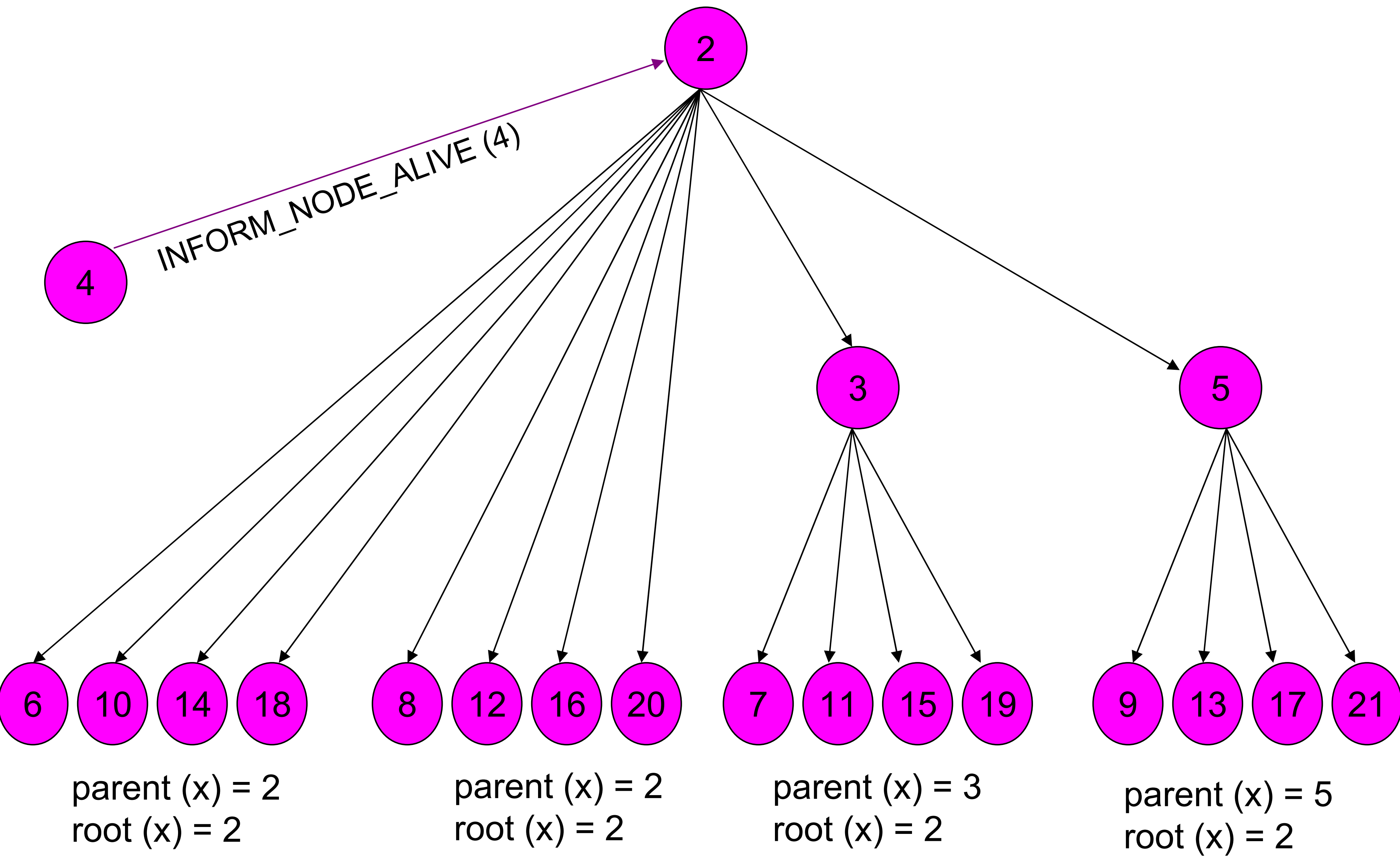
parent (x) = 3
root (x) = 2

parent (x) = 5
root (x) = 2

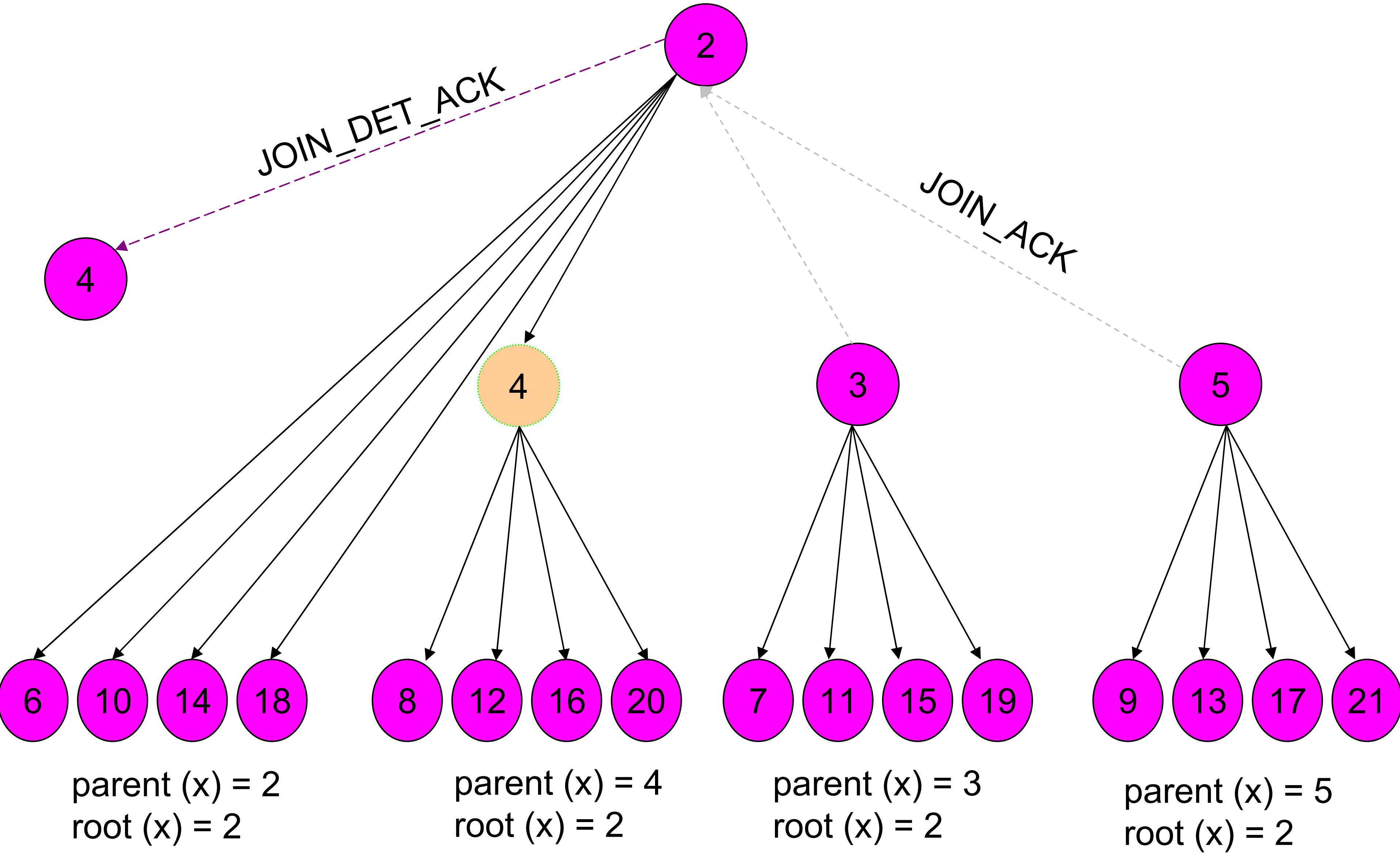
Group Membership Service



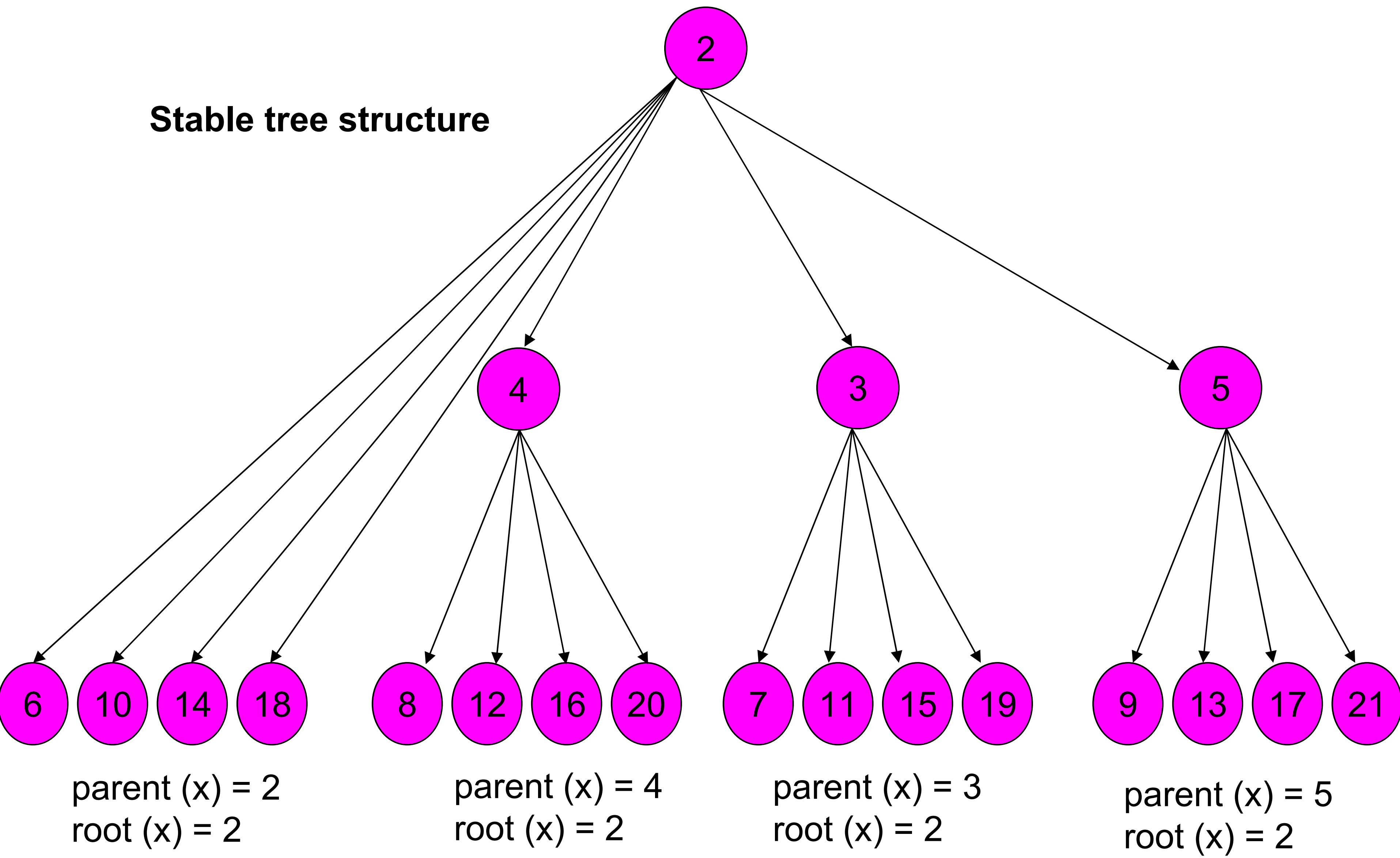
Group Membership Service



Group Membership Service



Group Membership Service



More failures !

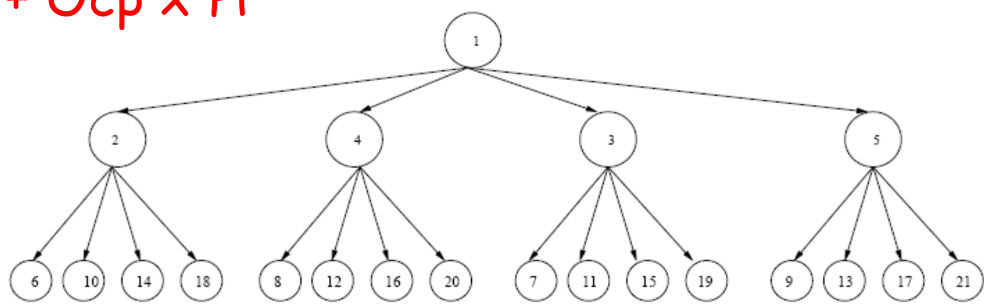
- Multiple Node Failures in parallel (before new view established)
  - Root node
    1. recalculating tree locally
    2. sends list of failed nodes
  - Steps may be repeated up to H-1 times, H=height of tree
- If a node fails at each level of tree structure →
  - H-1 initial tree stabilization phases for tree to stabilize
  - Lower height → lower complexity
    - increase branching factor "a"
    - but extremely low height reduces performance
    - trade-off

Experimental Framework

- Experiments conducted on
  - BlueGene/L
    - Two midplanes, each with 512 nodes nodes
    - 3D torus interconnect on each midplane
  - XTORC
    - 64 2 GHz P4 nodes (only 47 were available)
    - 1 Gb/s Ethernet
  - OS Cluster
    - 16 node dual processor AMD Athlon XP 1800+ machines
    - FastEther switch utilized through TCP/IP, MPICH over Myrinet GM
- Entire code written in C

Performance Modeling (Base Model)

- Total time for tree stabilization
  - Ts = Ocm + Ocp x H



- Communication overhead.
  - Ocm = 2 x L x (H-1)
    - L = point-to-point latency
- Computational overhead in each node
  - Ocp = 2.3 micro seconds

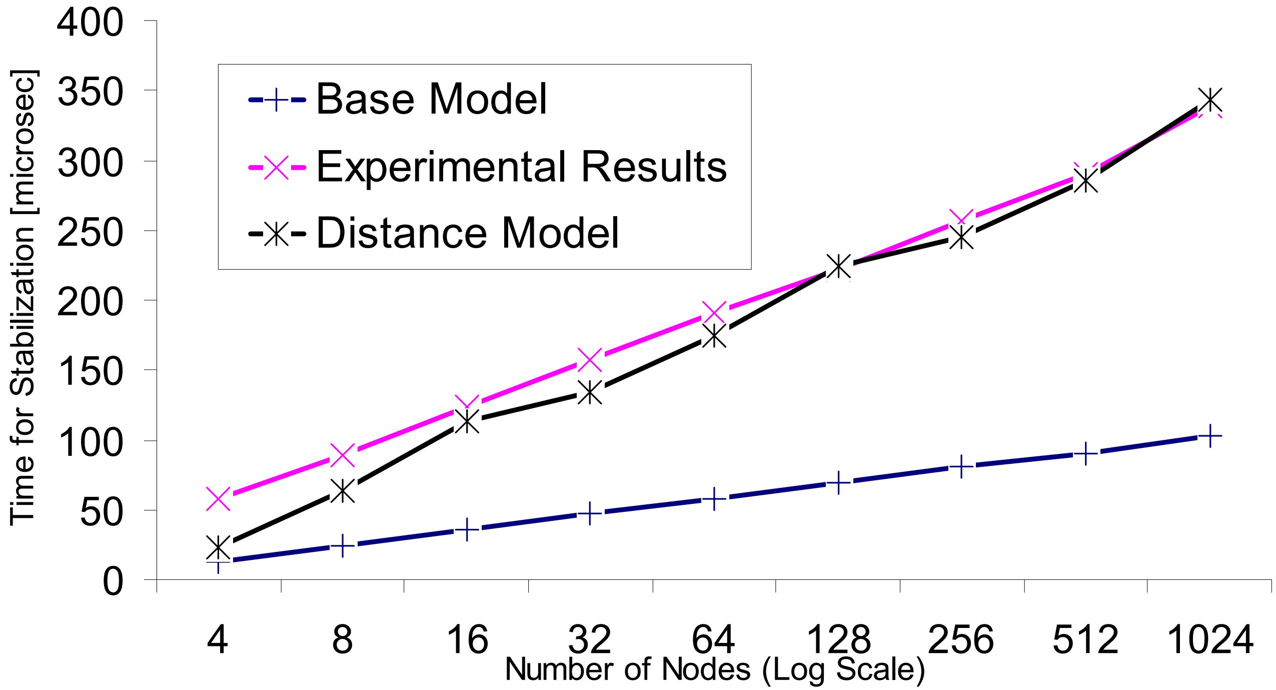
Performance Modeling (Distance Model)

- Distance model considers max. latency (L) b/w adjacent nodes (all parent/child pairs) at each level

$$Ocm = 2 \times \sum_{levels} \max(\text{hops b/w parent/child pairs at each level})] \times L \times (H - 1)$$

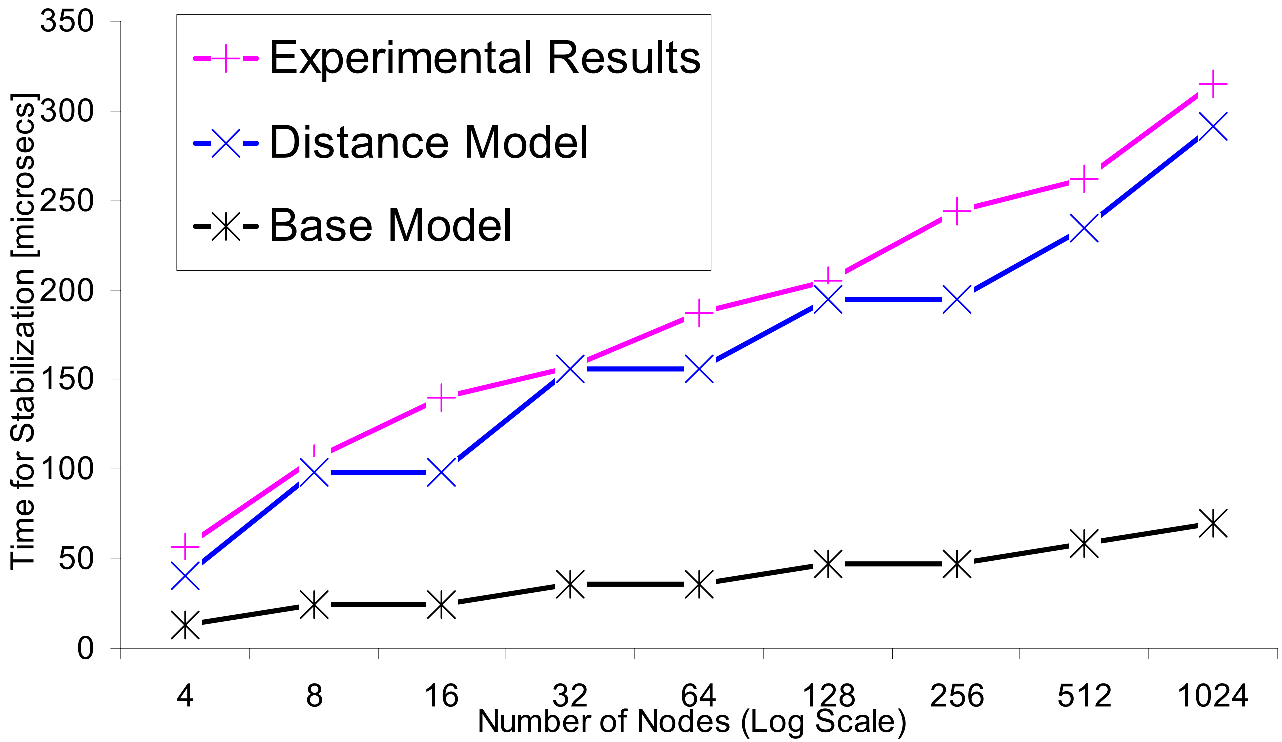
- Computational overhead in each node
  - Ocp = 2 micro seconds
- Total time for tree stabilization
  - Ts = Ocm + Ocp * H

Ts over MPI for a=2 on BG/L



1. Base model diverges from experimental results
    - Because of point to point communication topology in BG/L
2. Distance model matches observed results
3. Point-to-point latency = 4.6 micro sec
4. MPI tasks mapped to nodes → adjacent nodes in tree communicate over varying number of hop counts

Ts over MPI for a=4 on BG/L

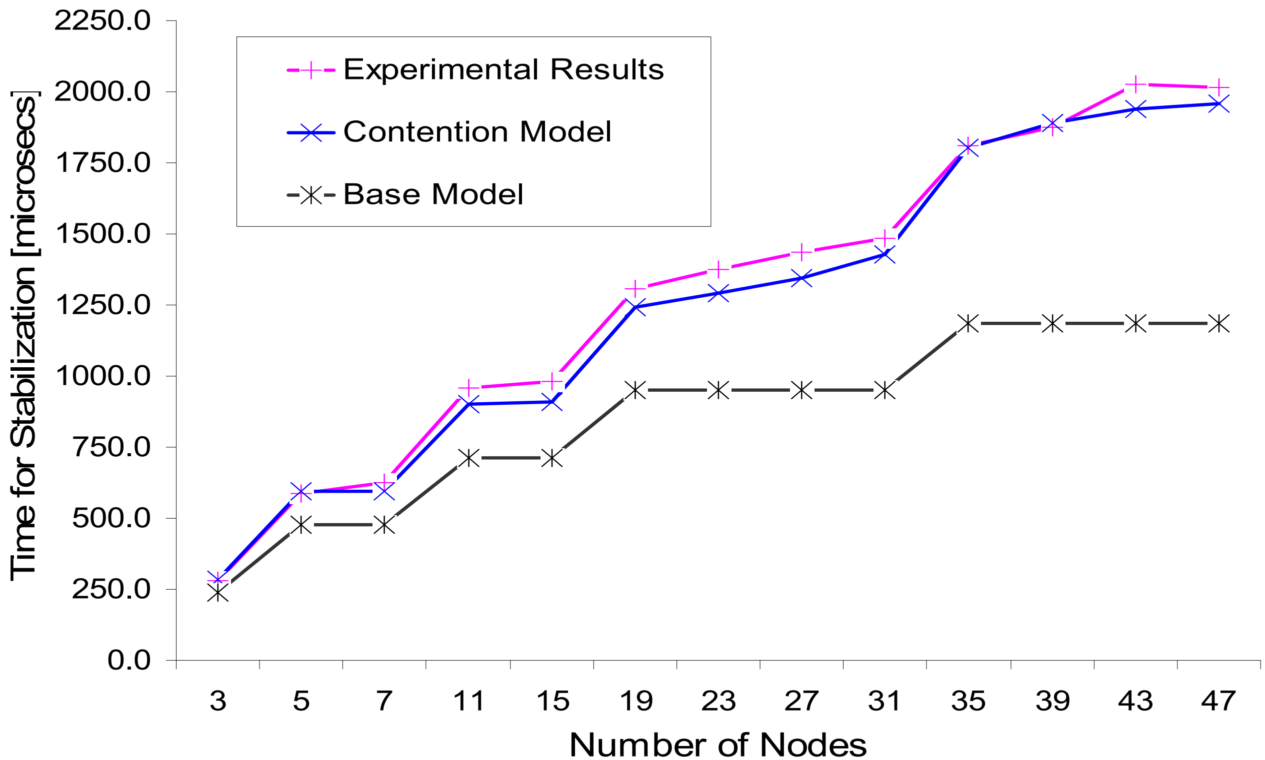


1. Model approximates observed performance w/ distance model
2. We have not considered system activity
3. Trend demonstrates scalability

Performance Modeling (Contention Model)

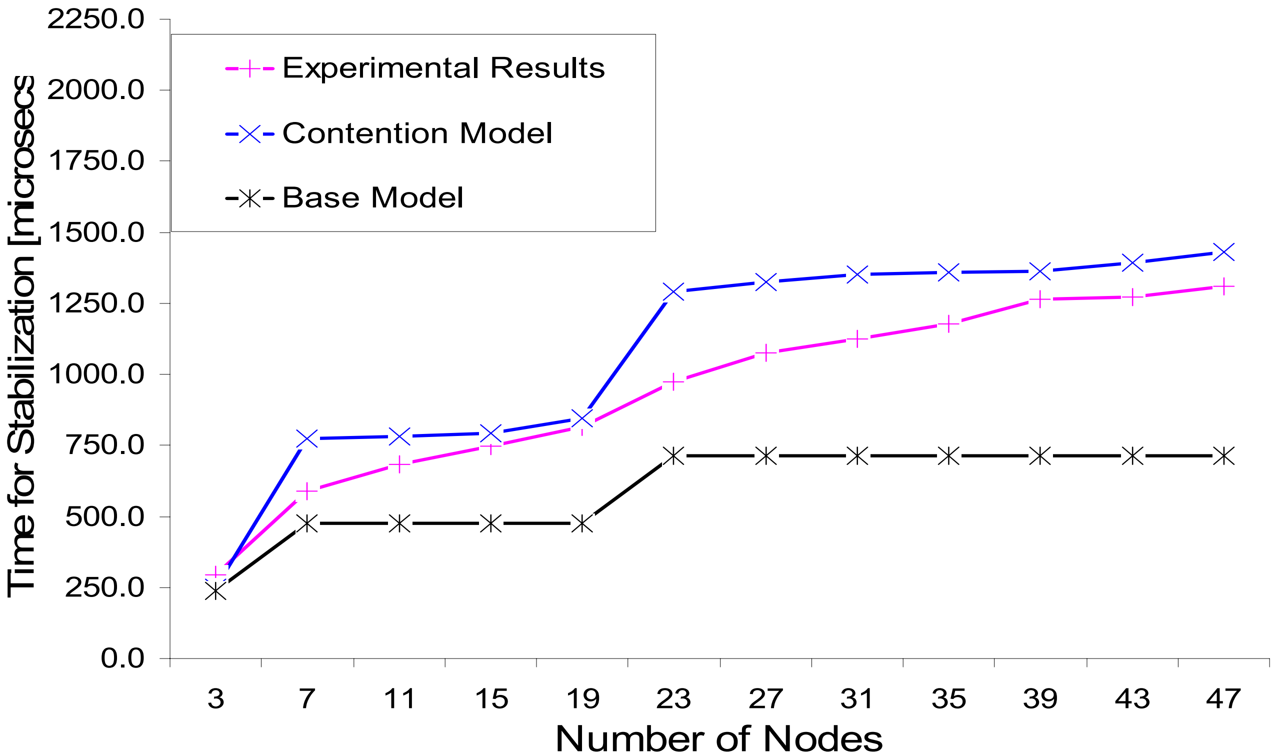
- Communication model similar to base model
  - $Ocm = 2 \times L(n) \times (H-1)$
  - where L(n) = latency as a function of # of nodes
- Computational overhead in each node
  - Ocp = 2.3 micro seconds
- Total time for tree stabilization
  - Ts = Ocm + Ocp * H

Ts over TCP for a=2 on XTORC



1. Base model shows step curve with increase in stabilization time
2. Contention model accurately reflects increased contention for large number of nodes
3. Close resemblance with experiments → extrapolate for large number of nodes

Ts over TCP for a=4 on XTORC



1. The model approximates the observed performance for a fully formed tree
2. Trend demonstrates scalability

Conclusion

Contributions:

- Scalable approach to reconfigure communication infrastructure
- Decentralized (peer-to-peer) protocol that maintains constant view of active nodes in the presence of faults
- Response time in order of hundreds of micro seconds and single-digit milliseconds over MPI on BG/L and TCP on Gigabit Ether, respectively.

Future Work:

- Performance evaluation for root/multiple node failure
- How to maintain a balanced tree even after a node failure?
- Integration into OpenMPI, LAM/MPI with BLCR to continue job execution in the presence of faults.

Questions or Comments?