Consensus in Distributed Systems

Course: Distributed Computing

Faculty: Dr. Rajendra Prasath

Spring 2019

About this topic

This course covers various concepts in **Consensus in Distributed Systems.** We will also focus on the essential aspects of consensus protocols in distributed systems.

RECAP

What did you learn so far?

- Challenges in Message Passing systems
- Distributed Sorting
- Space-Time Diagram
- Partial Ordering / Causal Ordering
- Concurrent Events
- Local Clocks and Vector Clocks
- Distributed Snapshots
- Termination Detection
- Topology Abstraction and Overlays
- Leader Election Problem in Rings
- Message Ordering / Group Communications
- Distributed Mutual Exclusion Algorithms

Topics to focus on …

- Distributed Mutual Exclusion
- Deadlock Detection
- Check Pointing and Rollback Recovery
- Self-Stabilization

Self-Study:

- Distributed Consensus

- Peer – to – peer computing and Overlays
- Authentication in Distributed Systems

Definition

A system is self-stabilizing if and only if:

- ➔ Convergence: Starting from any state, it is guaranteed that the system will eventually reach a correct state
- ➔ Closure: Given that the system is in a correct state, it is guaranteed to stay in a correct state, provided that no fault happens

- ➔ A system is said to be randomized self-stabilizing if and only if it is self-stabilizing and the expected number of rounds needed to reach a correct state is bounded by some constant k

Dijkstra's Algorithm

- For any machine:
  - S – State of its own
  - L – State of the left neighbor and
  - R - State of the right neighbor on the ring

- The exceptional machine:
  - If L = S then S = (S+1) mod K;

- All other machines:
  - If L = S then S = L;

Dijkstra's Algorithm

- A Privilege of a machine is able to change its current state on a Boolean predicate that consists of its current state and the states of its neighbors
- When a machine has a privilege, it is able to change its current state, which is referred to as a move.

Second solution (K = 3)

- The bottom machine, machine 0:
  - If (S+1) mod 3 = R then S = (S−1) mod 3;
- The top machine, machine n−1:
  - If L = R and (L+1) mod 3 = S then S = (L+1) mod 3;
- The other machines:
  - If (S+1) mod 3 = L then S = L;

Consensus in Distributed Systems

Let us explore distributed consensus algorithms in Distributed Systems

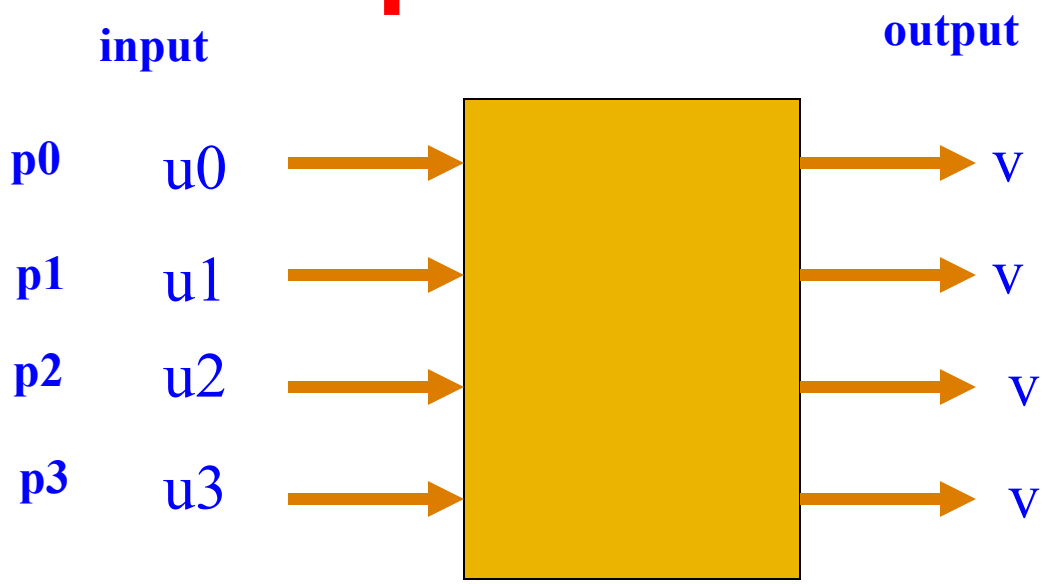
Distributed Consensus

Reaching agreement is a fundamental problem in distributed computing

Examples:

- Leader election / Mutual Exclusion
- Commit or Abort in distributed transactions
- Reaching agreement about which process has failed
- Clock phase synchronization
- Air traffic control system: all aircrafts must have the same view

Problem Specification



- Each process p_i has an input value u_i
- These processes exchanges their inputs, so that the outputs of all non-faulty processes become identical, even if one or more processes fail at any time
- Output v must be equal to the value of at least one process

Problem Specification

Termination	Every non-faulty process must eventually decide.
Agreement	The final decision of every non-faulty process must be identical.
Validity	If every non-faulty process begins with the same initial value v, then their final decision must be v

Observation

- If there is no failure, then reaching consensus is trivial. All-to-all broadcast followed by a applying a choice function …

- Consensus in presence of failures can however be complex. The complexity depends on the system model and the type of failures

Asynchronous Consensus

Seven members of a busy household decided to hire a cook, since they do not have time to prepare their own food

Each member *separately interviewed* every applicant for the cook's position. Depending on how it went, each member voted "yes" (means "hire") or "no" (means "don't hire").

These members will now have to communicate with one another to reach a uniform final decision about whether the applicant will be hired

The process will be repeated with the next applicant, until someone is hired. Consider various modes of communication like shared memory or message passing Also assume that one process may crash at any time

Asynchronous Consensus

Theorem:

In a **purely asynchronous** distributed system, the consensus problem is **impossible** to solve if even a **single process crashes**

Fischer, Lynch, Patterson (commonly known as FLP 85). Received the most influential paper award of ACM PODC in 2001

Proof

Bivalent and Univalent states

A decision state is **bivalent**, if starting from that state, there exist two distinct executions leading to two distinct decision values 0 or 1

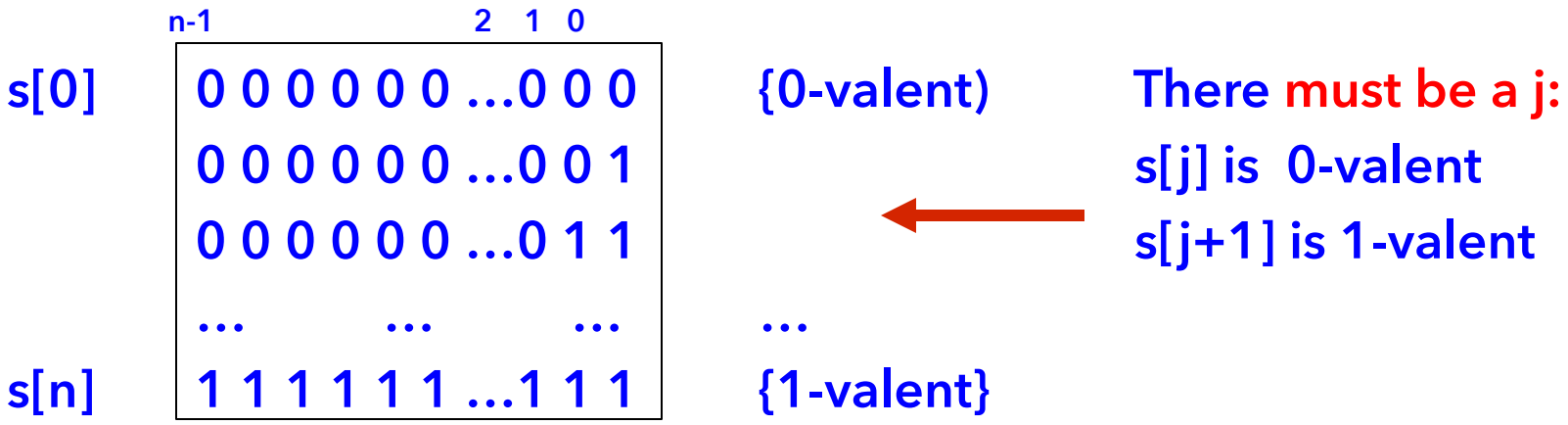
Otherwise it is **univalent**

An **univalent** state may be either **0-valent** or **1-valent.**

Proof

Lemma: Every consensus protocol must have a **bivalent initial state.**

Proof by contradiction: *Suppose not. Then consider the following input patterns:*



What if process j crashes at the first step?

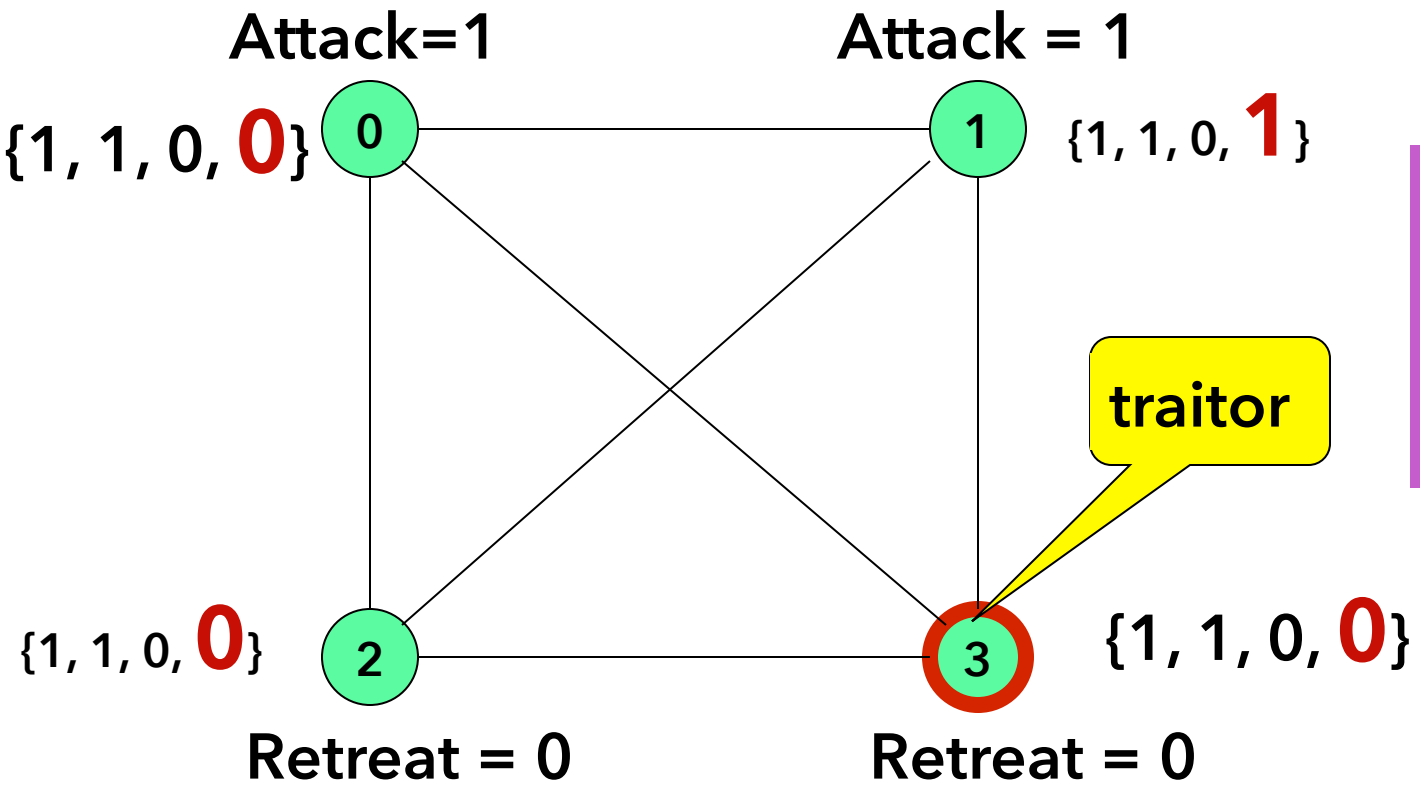
Consensus in Synchronous Systems: Byzantine Generals Problem

- Describes and solves the consensus problem on the synchronous Communication model
- Processor speeds have lower bounds and communication delays have upper bounds.
- The network is completely connected
- Processes undergo byzantine failures, the worst possible kind of failure

Byzantine Generals Problem

- n generals $\{0, 1, 2, \ldots, n-1\}$ decide about whether to "attack" or to "retreat" during a particular phase of a war. The goal is to agree upon the same plan of action
- Some generals may be "traitors" and therefore send either no input, or send conflicting inputs to prevent the "loyal" generals from reaching an agreement
- Devise a strategy, by which every loyal general eventually agrees upon the same plan, regardless of the action of the traitors

Byzantine Generals



The traitor may send out conflicting inputs

Every general will broadcast his judgment to everyone else. These are inputs to the consensus protocol.

Byzantine Generals

We need to devise a protocol so that every peer (call it a **lieutenant**) ***receives the same value*** from any given general (call it a **commander**)

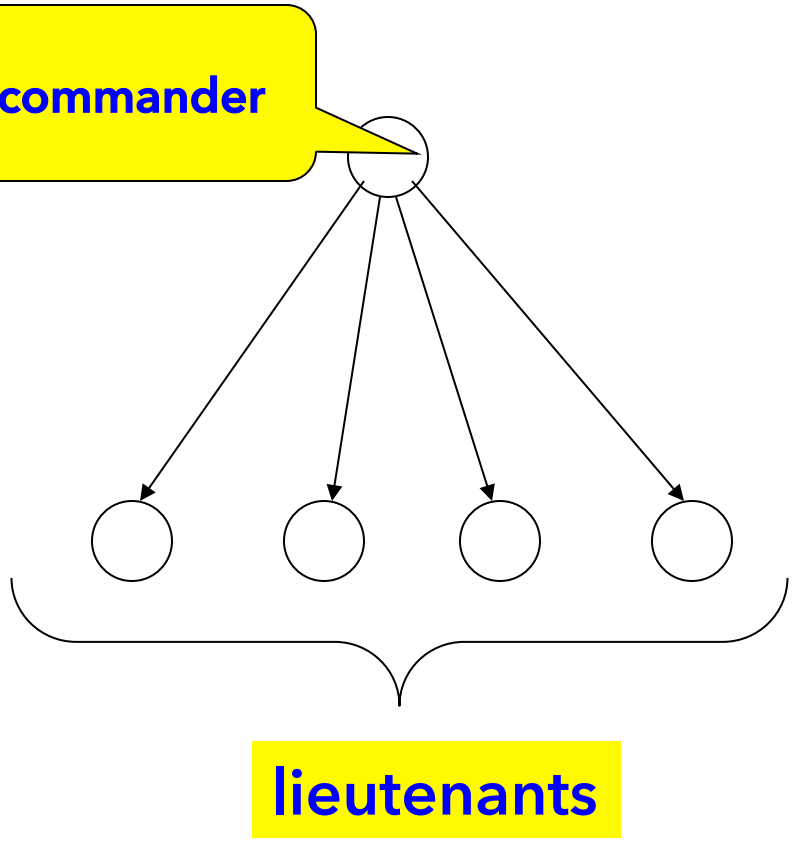
Clearly, the lieutenants will have to use **secondary information**

Note that the roles of the **commander** and the **lieutenants** will rotate among the generals

Interactive consistency specifications

IC1: Every loyal lieutenant receives the same order from the commander

IC2: If the commander is loyal, then every loyal lieutenant receives the order that the commander sends



The Communication Model

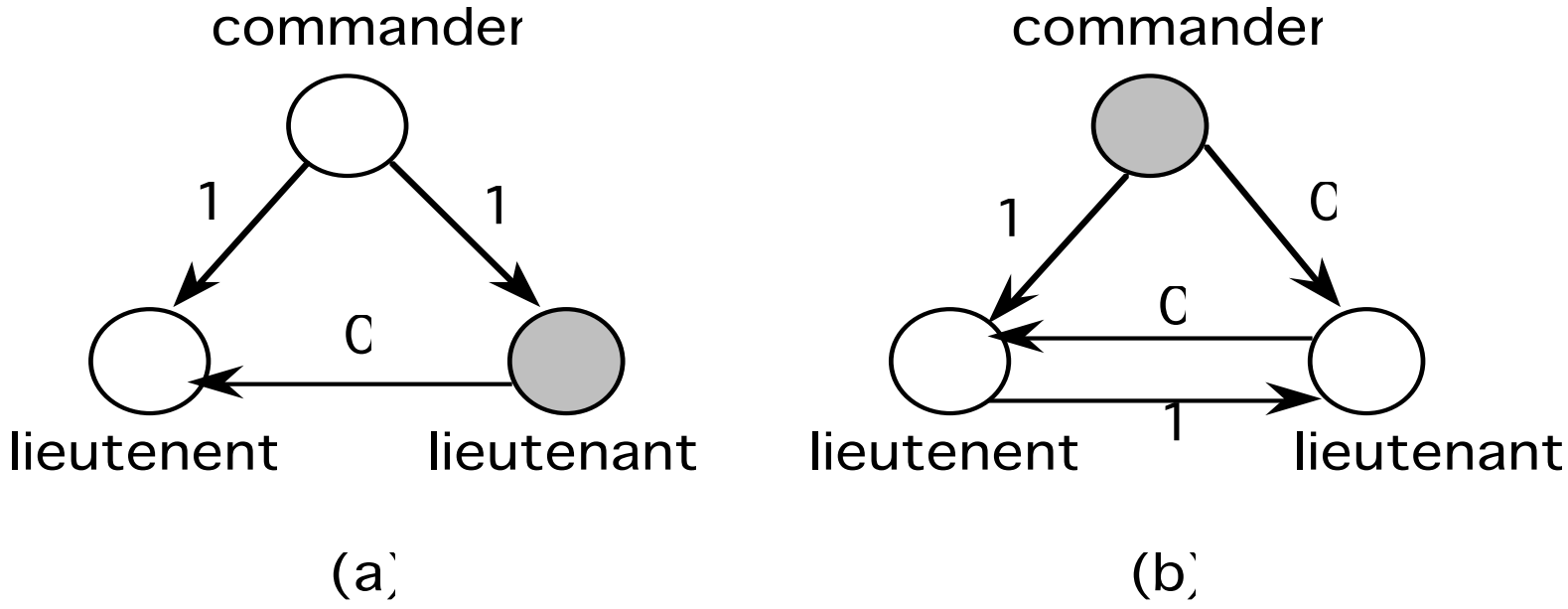
Oral Messages (OM)

1. Messages are not corrupted during the transit
2. Messages can be lost, but the absence of message can be detected
3. When a message is received (or its absence is detected), the receiver knows the identity of the sender (or the defaulter)

OM(m) represents an *interactive consistency protocol* in presence of at most **m** traitors.

An Impossibility Result

Using oral messages, no solution to the Byzantine Generals problem exists with three or fewer generals and one traitor. Consider the two cases:



In (a), to satisfy IC2, lieutenant 1 must trust the commander, but in IC2, the same idea leads to the violation of IC1.

Impossibility result

Using oral messages, no solution to the Byzantine Generals problem exists with 3m or fewer generals and m traitors (m > 0)

The proof is by contradiction: Assume that such a solution exists. Now, divide the 3m generals into three groups of m generals each, such that all the traitors belong to one group. Let one general simulate each of these three groups. This scenario is equivalent to the case of three generals and one traitor. We already know that such a solution does not exist.

Note: Be always suspicious about such an informal reasoning (refer to Lamport's original paper)

The OM(m) algorithm

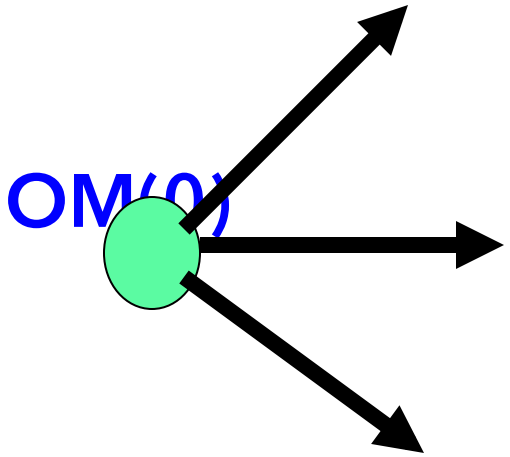
Recursive algorithm

OM(m)

OM(m-1)

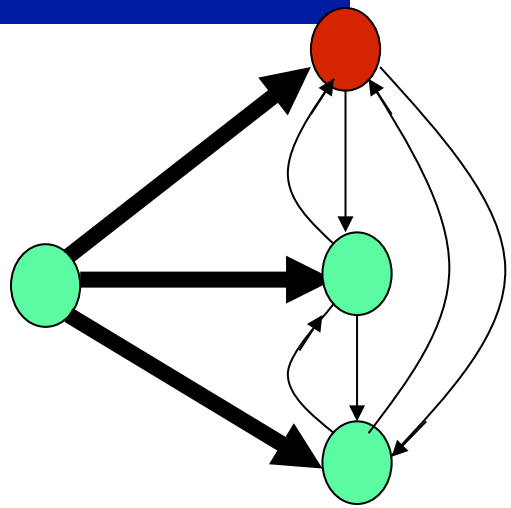
OM(m-2)

OM(0)



OM(0) = Direct broadcast

The OM(m) algorithm



1. Commander i sends out a value v (0 or 1)

2. If $m > 0$, then every lieutenant $j \neq i$, after receiving v , acts as a commander and initiates OM(m-1) with everyone except i

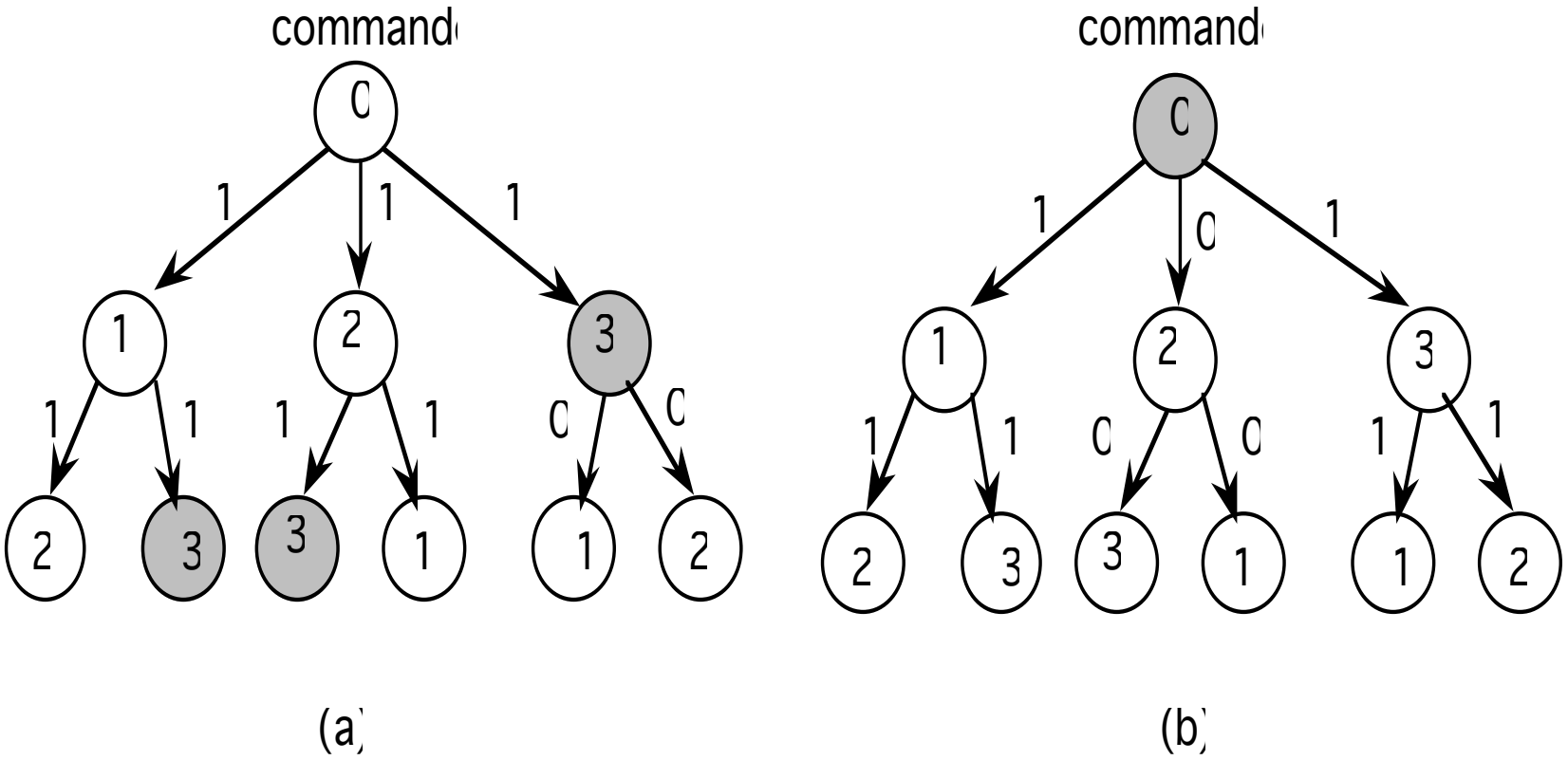
3. Every lieutenant, collects (n-1) values:

   → (n-2) values received from the lieutenants using OM(m-1) and

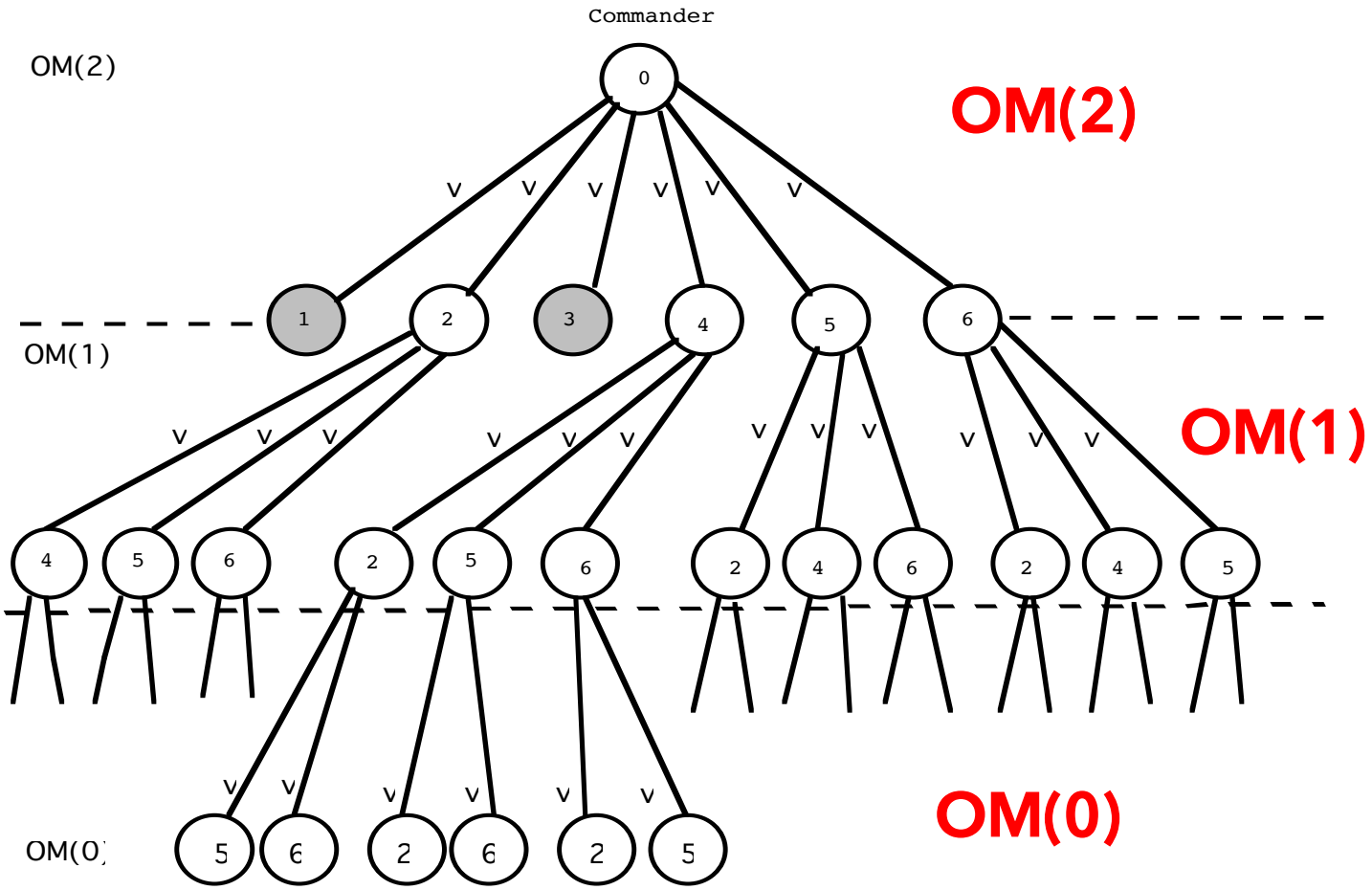
   → one direct value from the commander

   Then he picks the majority of these values as the order from i

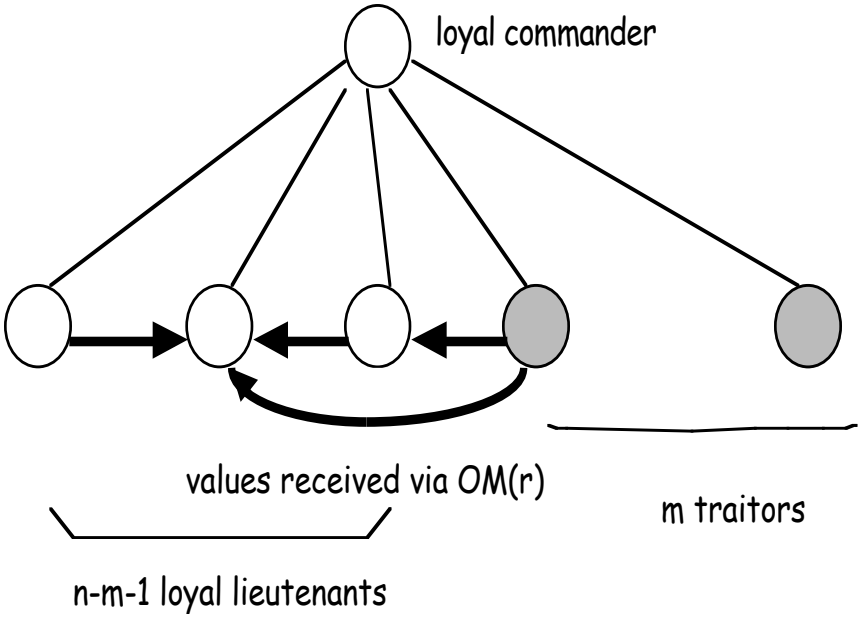
Example of OM(1)



Example of OM(2)



Proof of OM(m)



Lemma:

Let the commander be loyal, and n > 2m + k,

where m = maximum number of traitors.

Then OM(k) satisfies IC2

Proof of OM(m)

Proof:

If k=0, then the result trivially holds.

Let it hold for k = r (r > 0) i.e. OM(r) satisfies IC2. We have to show that it holds for k = r + 1 too.

By definition, n > 2m+ r+1, so n-1 > 2m+ r

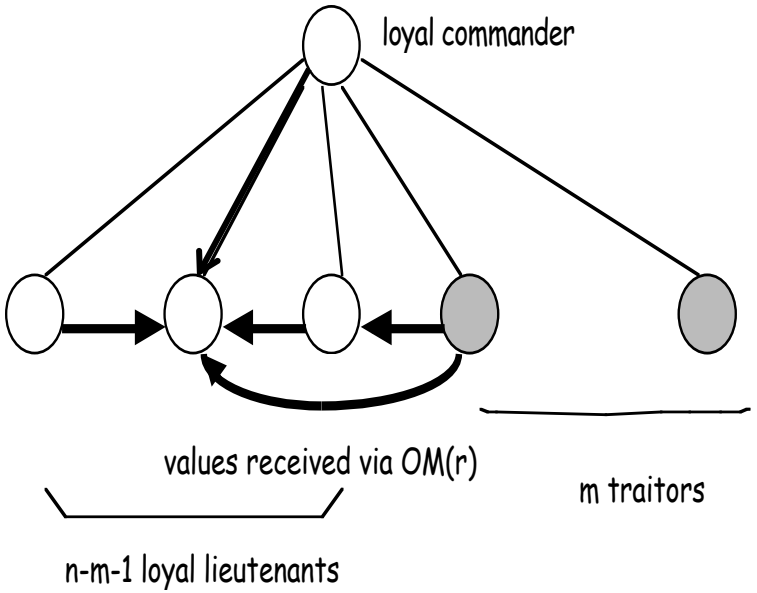
So OM(r) holds for the lieutenants in the bottom row.

Each loyal lieutenant

collects n-m-1 identical good values and m bad values.

So bad values are voted

out (n-m-1 > m+r implies n-m-1 > m)



"OM(r) holds" means each loyal lieutenant receives identical values from every loyal commander

The Final theorem

Theorem: If $n > 3m$ where m is the maximum number of traitors, then **OM(m)** satisfies both **IC1** and **IC2**

Proof: Consider two cases:

Case 1: Commander is loyal

The theorem follows from the previous lemma (substitute $k = m$).

Case 2: Commander is a traitor

We prove it by induction:

Base case: m=0 trivial. (*Induction hypothesis*)
Let the theorem hold for m = r

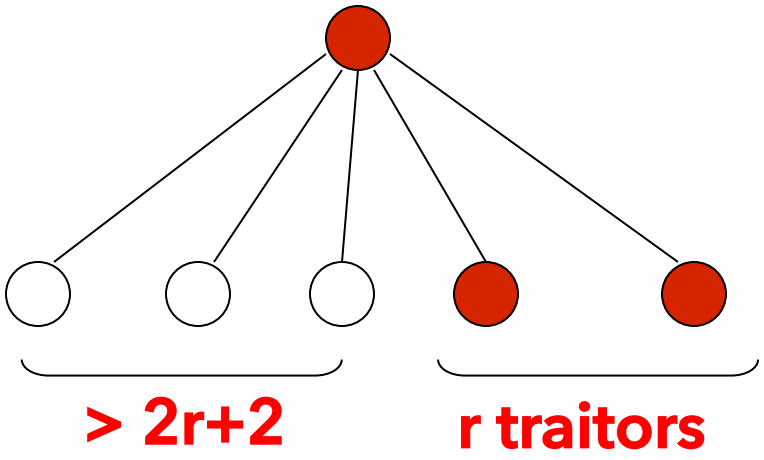
We have to show that it holds for m = r+1 too.

Proof (continued)

There are n > 3(r + 1) generals and r + 1 traitors

Excluding the commander, there are > 3r+2 generals of which there are r traitors. So > 2r+2 lieutenants are loyal.

Since 3r+ 2 > 3r ➔ OM(r) satisfies IC1 and IC2



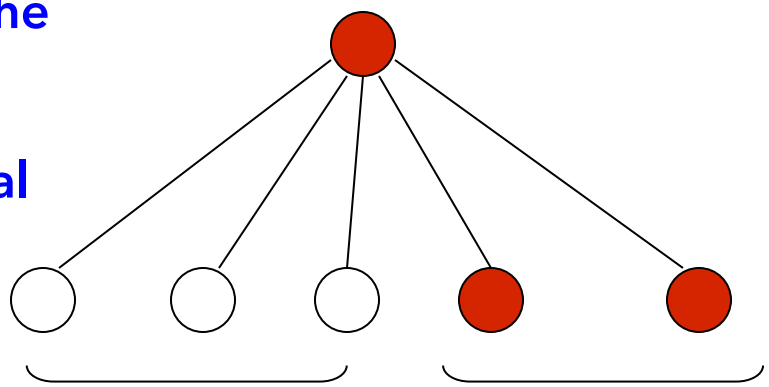
Proof (continued)

In OM(r+1), a loyal lieutenant chooses the majority from

(1) > 2r+1 values obtained from the loyal lieutenants via OM(r),

(2) the r values from the traitors, and

(3) the value directly from the commander.



The set of values collected in part (1) & (3) are the same for all loyal lieutenants - it is the same set of values that these lieutenants received from the commander

Also, by the induction hypothesis, in part (2) each loyal lieutenant receives identical values from each traitor. So every loyal lieutenant eventually collects the same set of values

Applications

The real world applications include

- Block Chain Transactions – Bit Coins
- Clock Synchronization
- PageRank
- Opinion Formation
- Smart Power Grids
- State Estimation
- Control of UAVs (and multiple robots / agents in general)
- Load Balancing and many other domains
- Block Chain
  - Distributed Consensus is really important

Summary

- State of Machines
- Legitimate / Illegitimate States

- Self-Stabilizing Algorithms
  - Dijkstra's Algorithm (token rings)
  - Constructing a Breadth First Tree

- Fault Tolerance
- Costs of self-stabilization
  - Stay tuned ... More to come up … !!

How to reach me?

➔ Please leave me an email:

rajendra [DOT] prasath [AT] iiits [DOT] in

➔ Visit my homepage @

➔ http://www.iiits.ac.in/FacPages/index-rajendra.html

OR

➔ http://rajendra.2power3.com

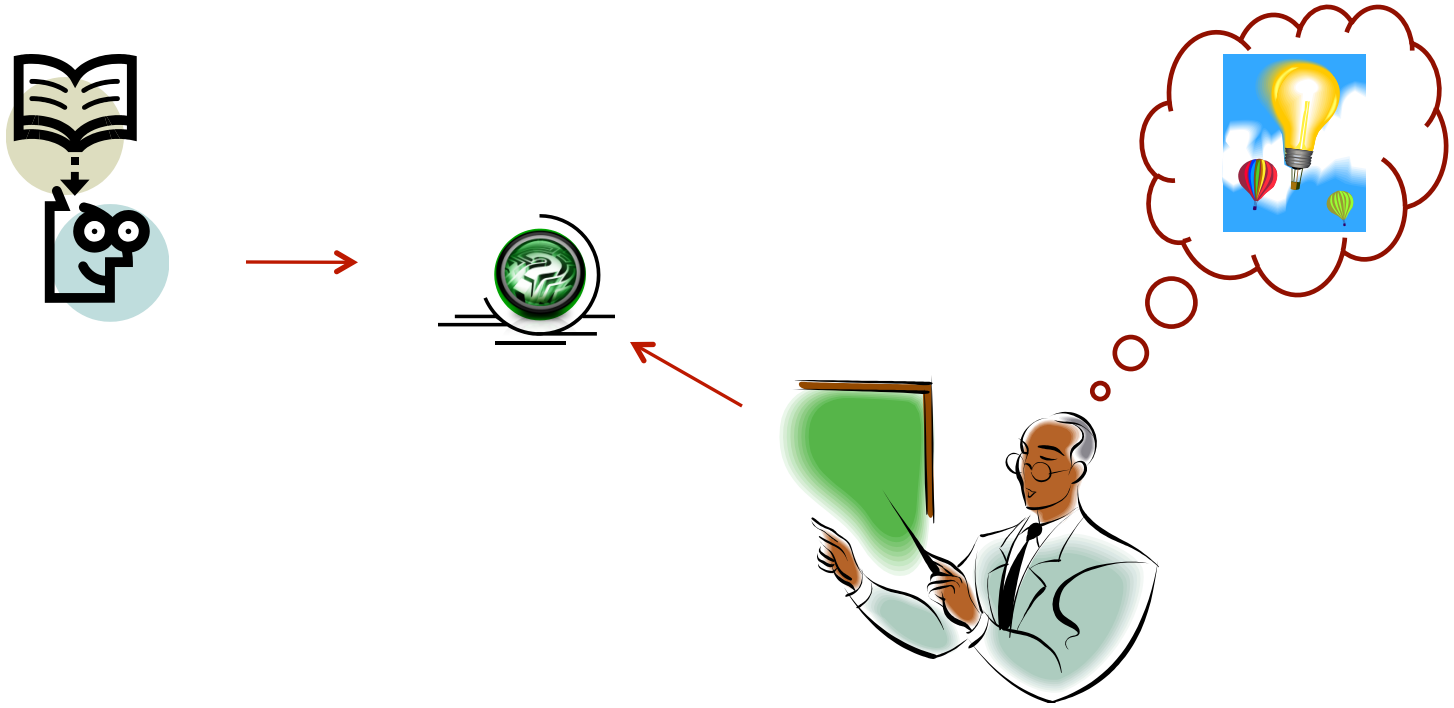
Help among Yourselves?

- **Perspective Students** (having CGPA above 8.5 and above)
- **Promising Students** (having CGPA above 6.5 and less than 8.5)

- **Needy Students** (having CGPA less than 6.5)
  - Can the above group help these students? (Your work will also be rewarded)

- You may grow a culture of **collaborative learning** by helping the needy students

Thanks …



… Questions ???