Fault Tolerance

Distributed Systems [8]

殷亚凤

Email: yafeng@nju.edu.cn

Homepage: http://cs.nju.edu.cn/yafeng/

Room 301, Building of Computer Science and Technology

Review

- Replication
- Data-Centric Consistency Models
- Client-Centric Consistency Models
- Replication Management
- Consistency Protocols

This Lesson

- Concepts about faults
- How to improve dependability
- Two-army problem
- Byzantine agreement problem

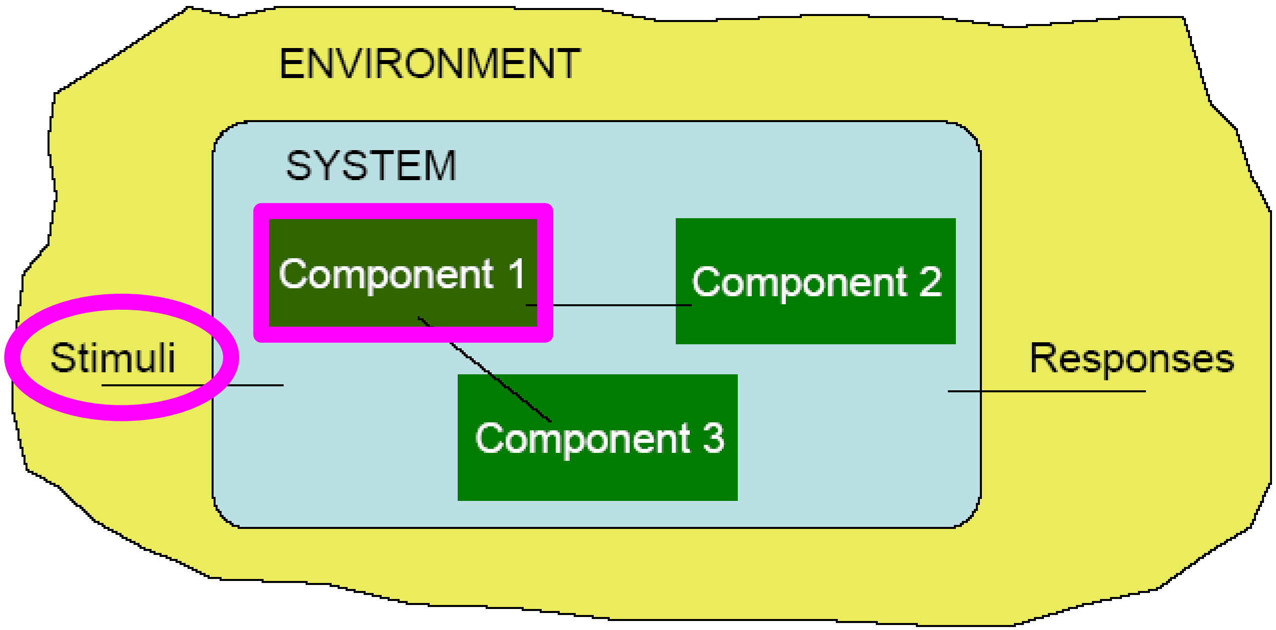
Basic Concepts

- Dependability Includes
  - Availability
  - Reliability
  - Safety
  - Maintainability

Dependability

- Availability
  - The fraction of the time that a system meets its specification.
  - The probability that the system is operational at a given time t.
- Reliability
  - A measure of success with which a system conforms to some authoritative specification of its behavior.
  - Probability that the system has not experienced any failures within a given time period.
  - Typically used to describe systems that cannot be repaired or where the continuous operation of the system is critical.
- Safety
  - When the system temporarily fails to conform to its specification, nothing catastrophic occurs.
- Maintainability
  - Measure of how easy it is to repair a system.

Basic System Concepts



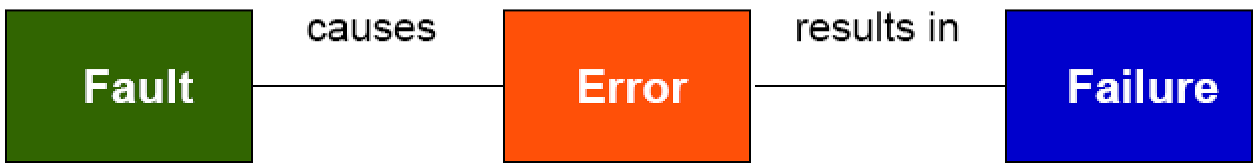
External state

Internal state

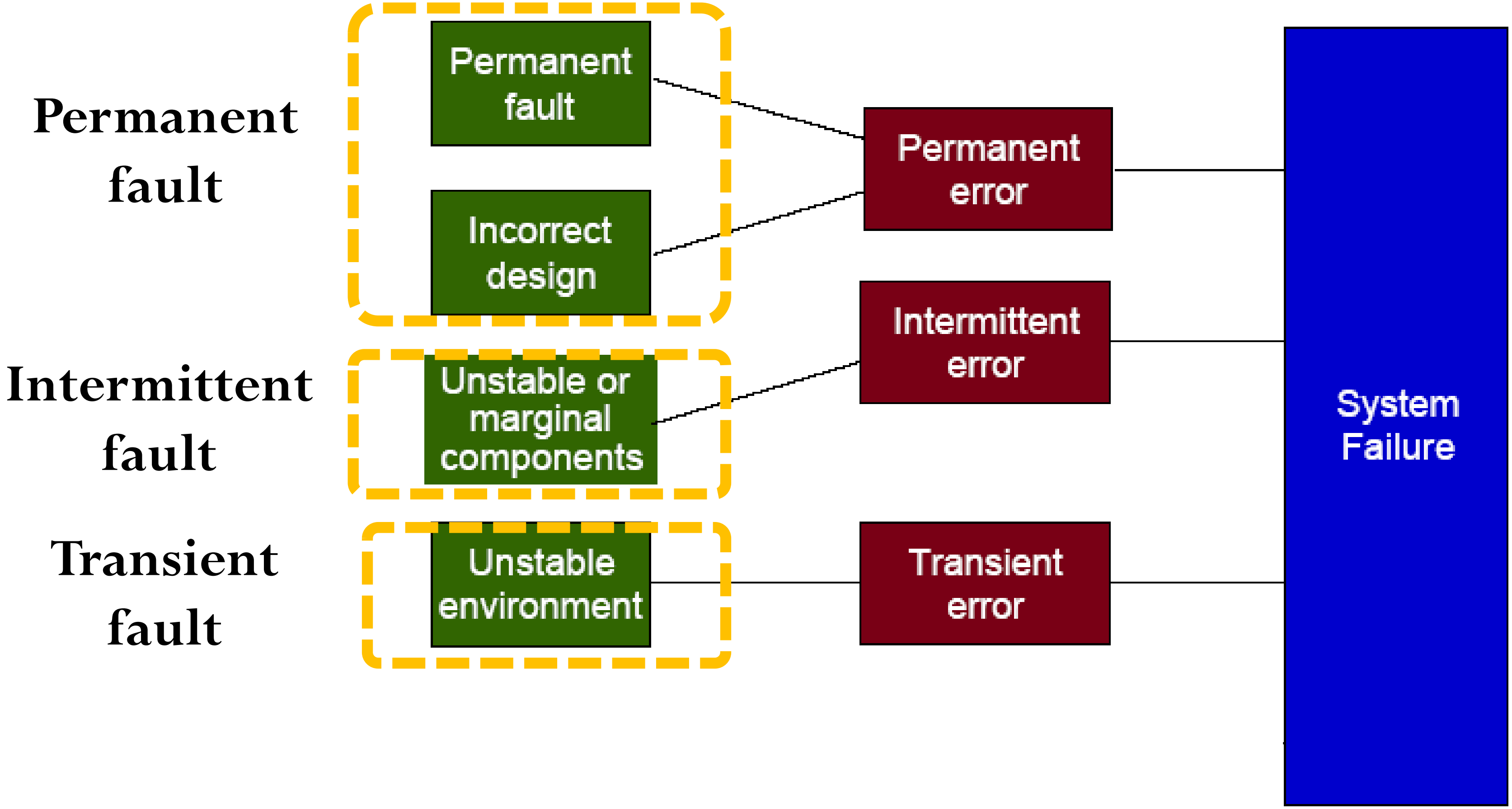
Fundamental Definitions

- Failure
  - A system is said to fail when it cannot meet its promises.
- Error
  - The part of the a system's state that may lead to a failure.
- Fault
  - The cause of an error is called a fault.

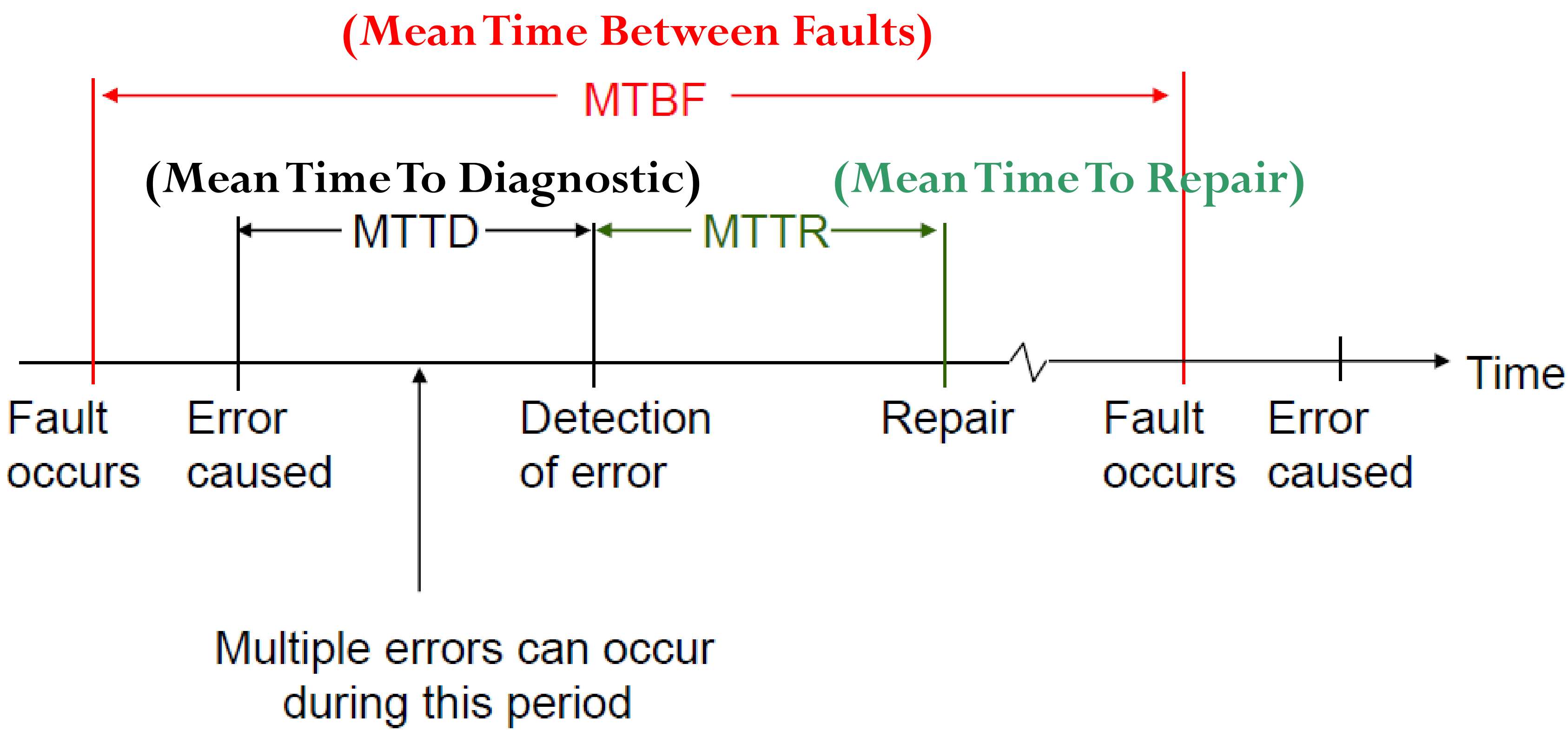
Faults to Failures



Fault Classification



Faults


(Mean Time Between Faults)
MTBF
(Mean Time To Diagnostic)
MTTD
(Mean Time To Repair)
MTTR
Time
Fault occurs
Error caused
Detection of error
Repair
Fault occurs
Error caused
Multiple errors can occur during this period

Faults Models

- Different types of faults.

Type of failure	Description
Crash fault	A server halts, but is working correctly until it halts
Omission fault *Receive omission* *Send omission*	A server fails to respond to incoming requests A server fails to receive incoming messages A server fails to send messages
Timing fault	A server's response lies outside the specified time interval
Response fault *Value failure* *State transition failure*	The server's response is incorrect The value of the response is wrong The server deviates from the correct flow of control
Arbitrary fault	A server may produce arbitrary responses at arbitrary times

How to Improve Dependability

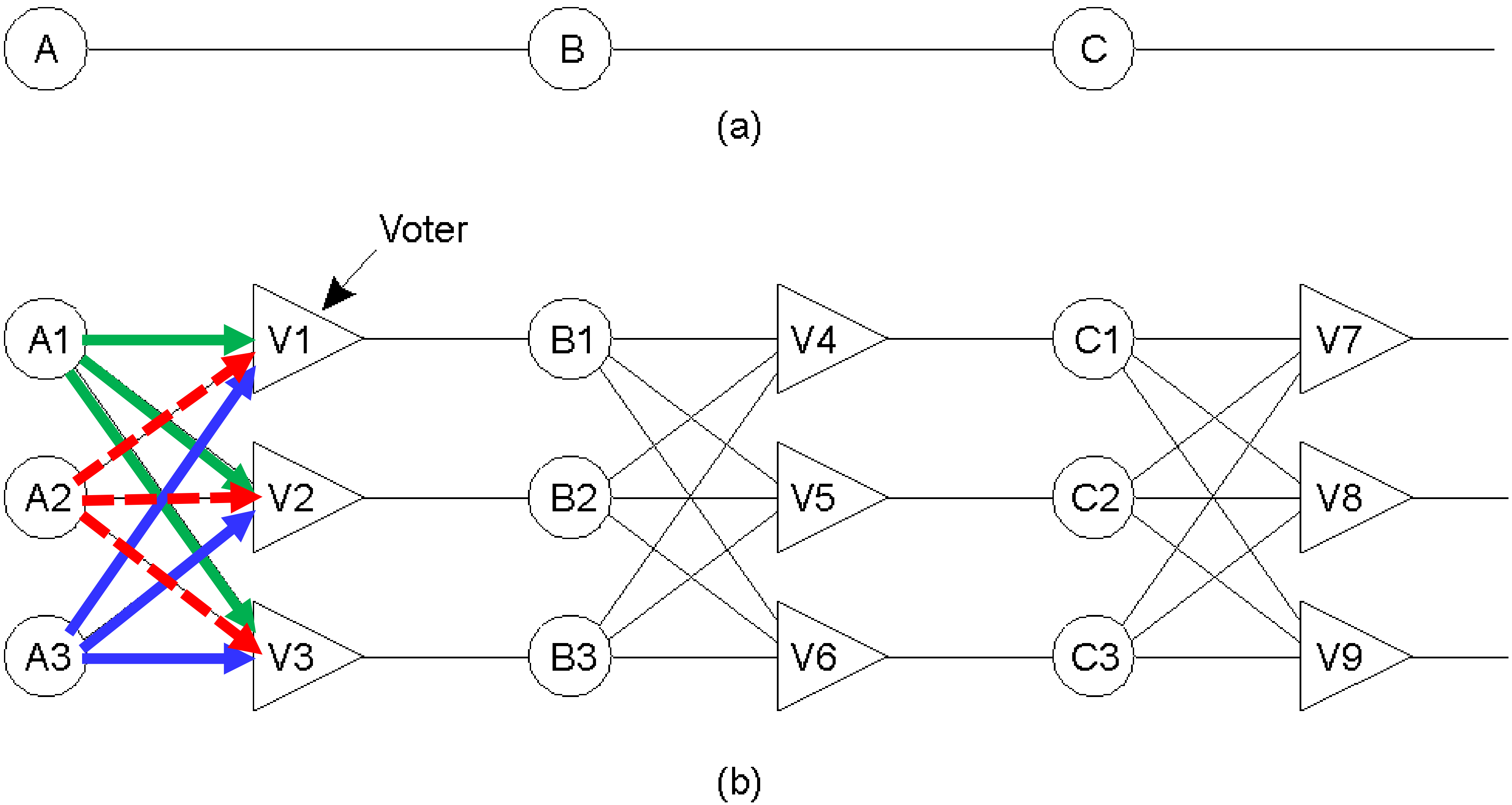
- Mask failures by redundancy
  - Information redundancy
    - E.g., add extra bits to detect and recovered data transmission errors
  - Time redundancy
    - Transactions; e.g., when a transaction aborts re-execute it without adverse effects.
  - Physical redundancy
    - Hardware redundancy
      - Take a distributed system with **4 file servers**, each with a 0.95 chance of being up at any instant
      - The probability of all 4 being down simultaneously is 0.054 = 0.000006
      - So the probability of at least one being available (i.e., the reliability of the full system) is 0.999994, far better than 0.95
      - If there are 2 servers, then the reliability of the system is (1-0.052) = 0.9975
    - Software redundancy
      - **Process redundancy** with similar considerations
- A design that does not require simultaneous functioning of a substantial number of critical components.

Hardware Redundancy

- Two computers are employed for a single application, one acting as a standby
  - Very costly, but often very effective solution
- Redundancy can be planned at a finer grain
  - Individual servers can be replicated
  - Redundant hardware can be used for non-critical activities when no faults are present
  - Redundant routes in network

Failure Masking by Redundancy

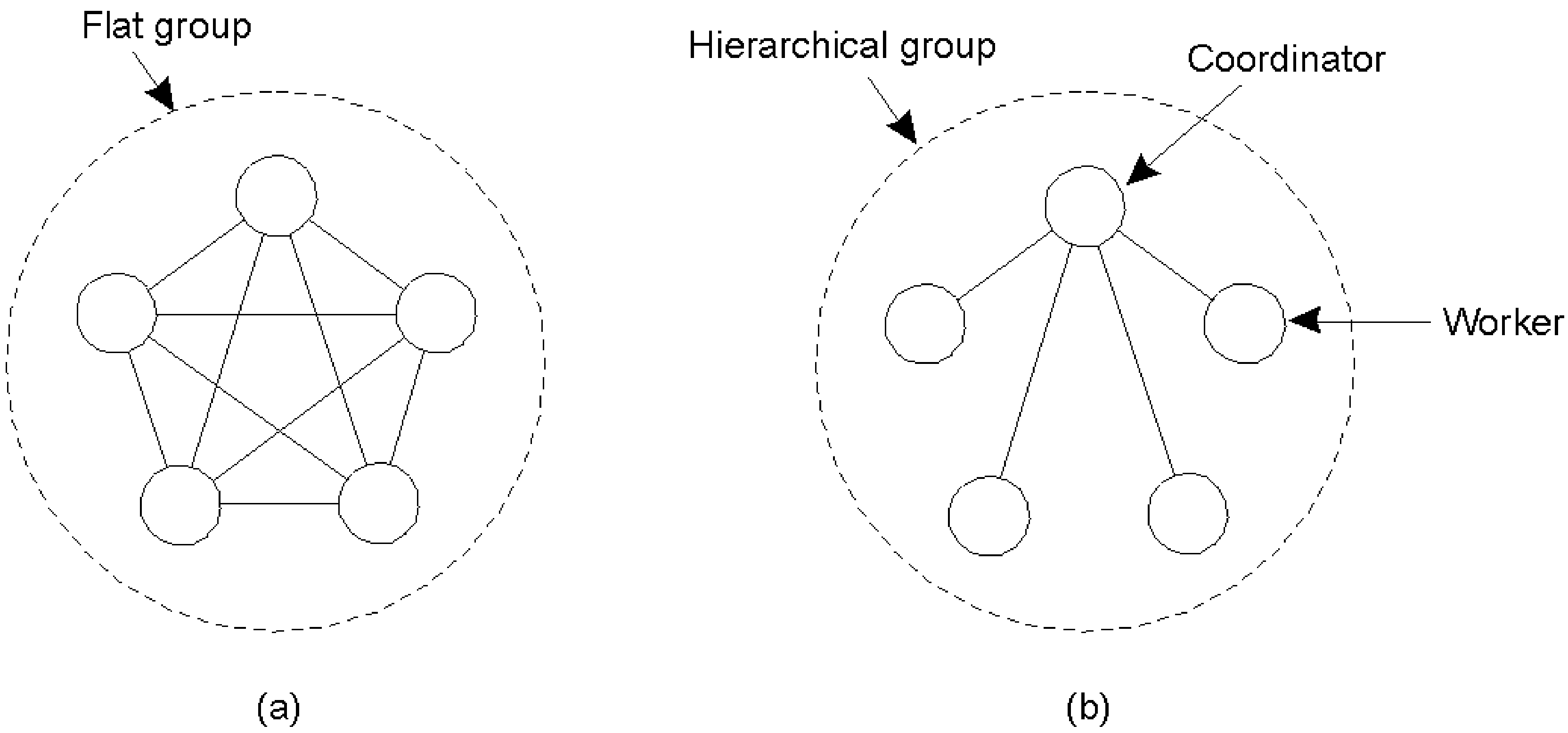
- Triple modular redundancy.



Process Recovery

- Process groups
  - All members of a group receive a message to the group
  - If one process fails, others can take over
  - Can be dynamic; processes can have multiple memberships
  - Flat vs. hierarchical groups

Flat Groups versus Hierarchical Groups



a) Communication in a flat group.
b) Communication in a simple hierarchical group

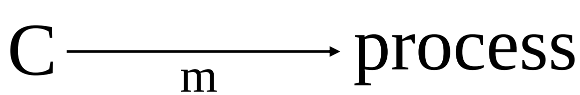
Management of Replicated Processes

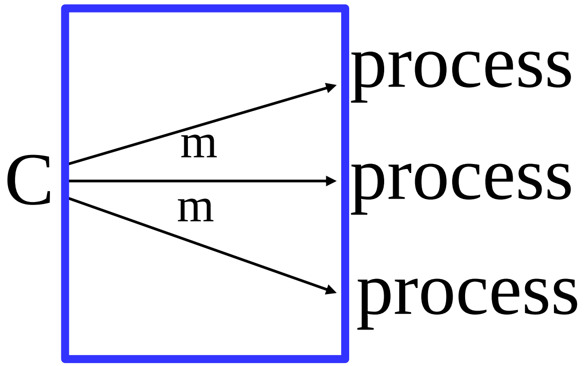
- Primary copy
  - Primary-backup setup
  - Coordinator is the primary that coordinates all updates
  - If coordinator fails, one backup takes over (usually through an election procedure)
  - Processes are organized hierarchically
- Replicated-writes
  - Active replication and quorum-based protocols
  - Flat group organization
  - No single points of failure

Process resilience

How can fault tolerance be achieved in distributed systems?

1. Key approach to tolerating a faulty process: replicate the process and organize these identical process into a group.





Some design issues

- Structure of group: flat/hierarchical
- Need for managing groups and group membership
  - centralized: group server
  - distributed: totally-ordered reliable multicast
- How many replicas? For K fault tolerant,

  Fail-silent faults : K+1

  Byzantine faults : 2K+1 majority

Agreement in faulty system

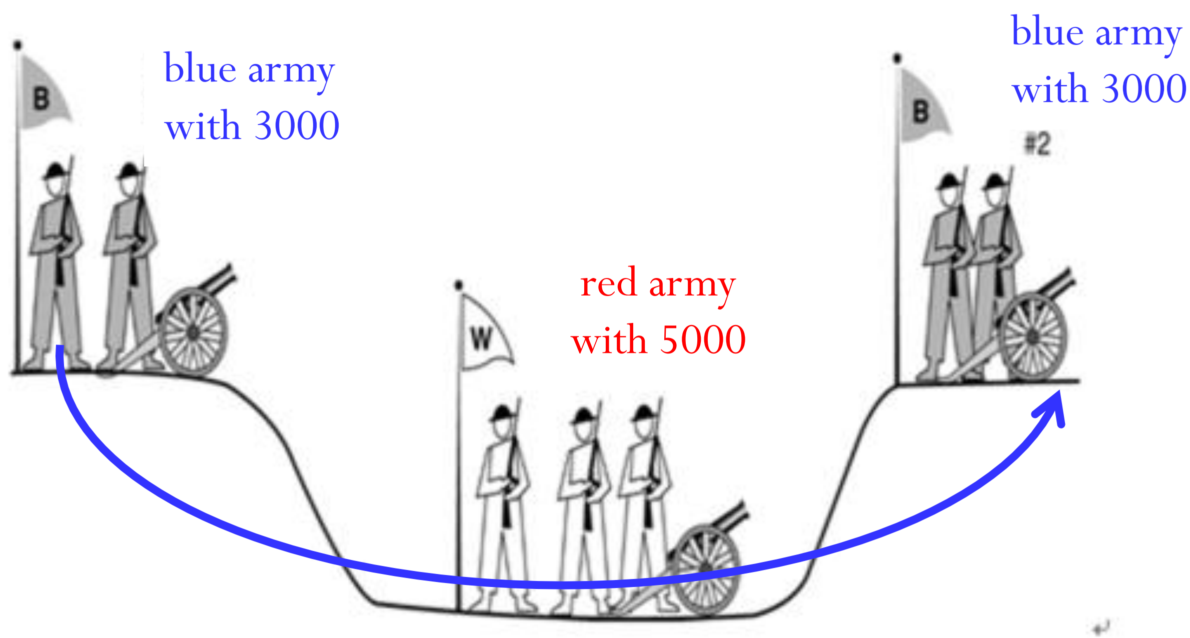
Introduction

There is a need to have processes agree on something.

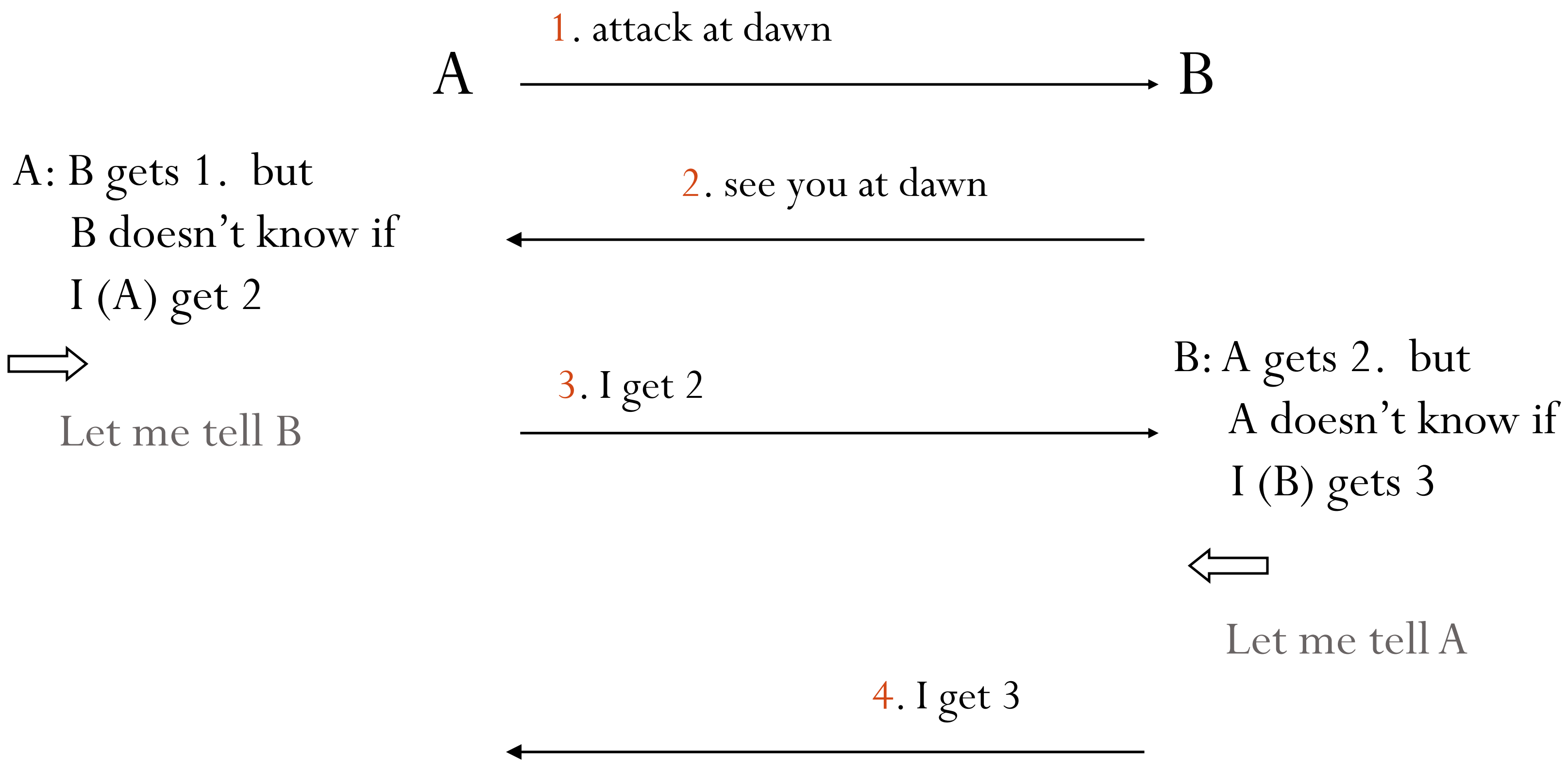
- Goal: all non-faulty processors reach consensus on some issue within a finite number of steps
- Two kinds of fault:
  - communication fault : two-army problem
  - processor fault : Byzantine generals problem

Two-army problem

- Problem:



Two blue armies want to coordinate their attacks on the red army. But they can only send messenger



A and B will never reach agreement

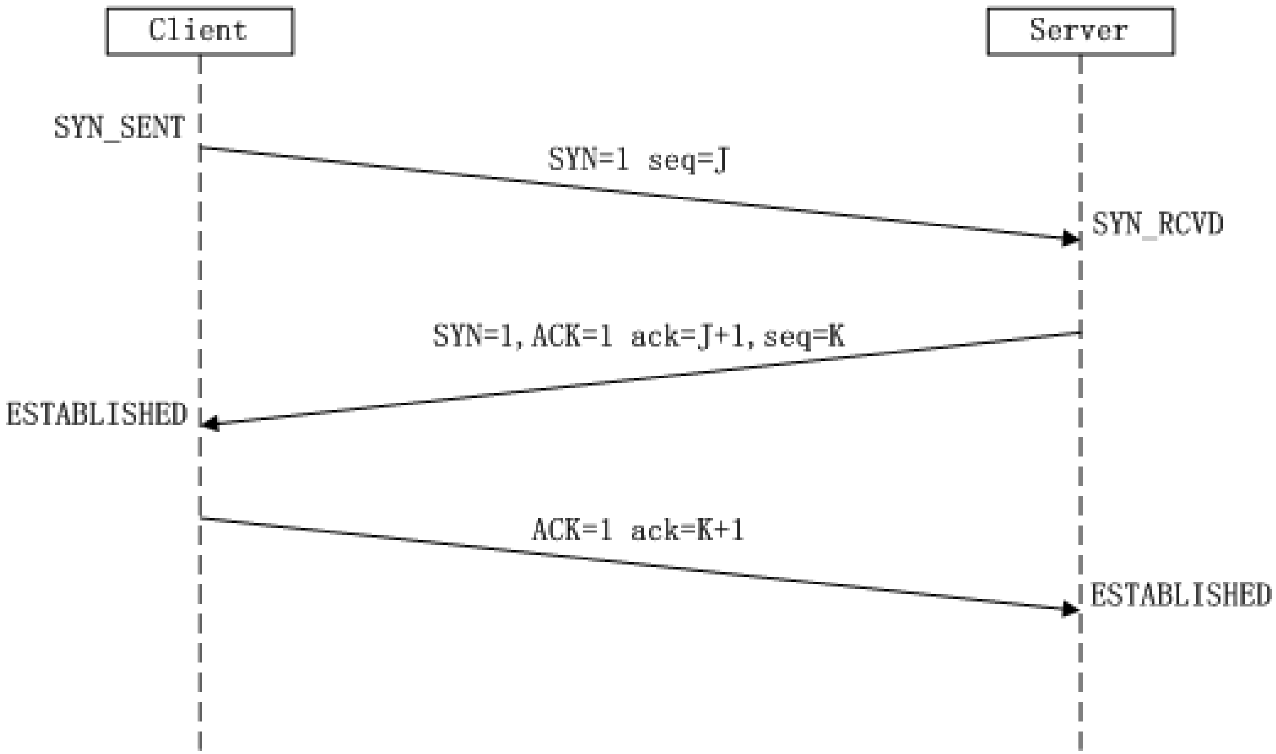
Because sender of the last message doesn’t know if the last message arrived.

- Conclusion

  agreement between two processes is not possible in the case unreliable communication

- How does TCP deal with this problem when two computer want to establish a TCP connection ?

Three way handshake



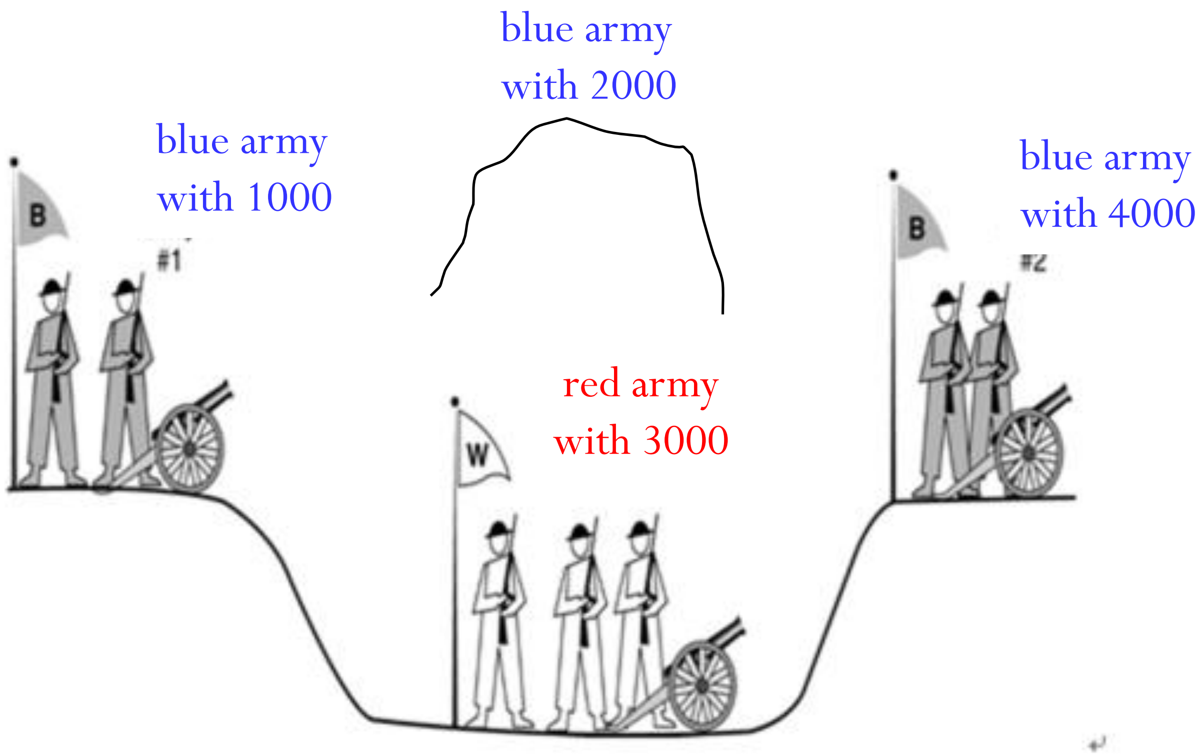
Byzantine agreement problem

- Communication is perfect but the processors are not (in Byzantine faults) .

- Problem:

  n blue generals want to coordinate their attacks on the red army. But *m* of them are traitors.

Byzantine agreement problem



Question:

Whether the loyal generals can still reach agreement?

Byzantine agreement problem

Generals exchange troop strengths, at the end, each general has a vector of length n corresponding to all the armies.

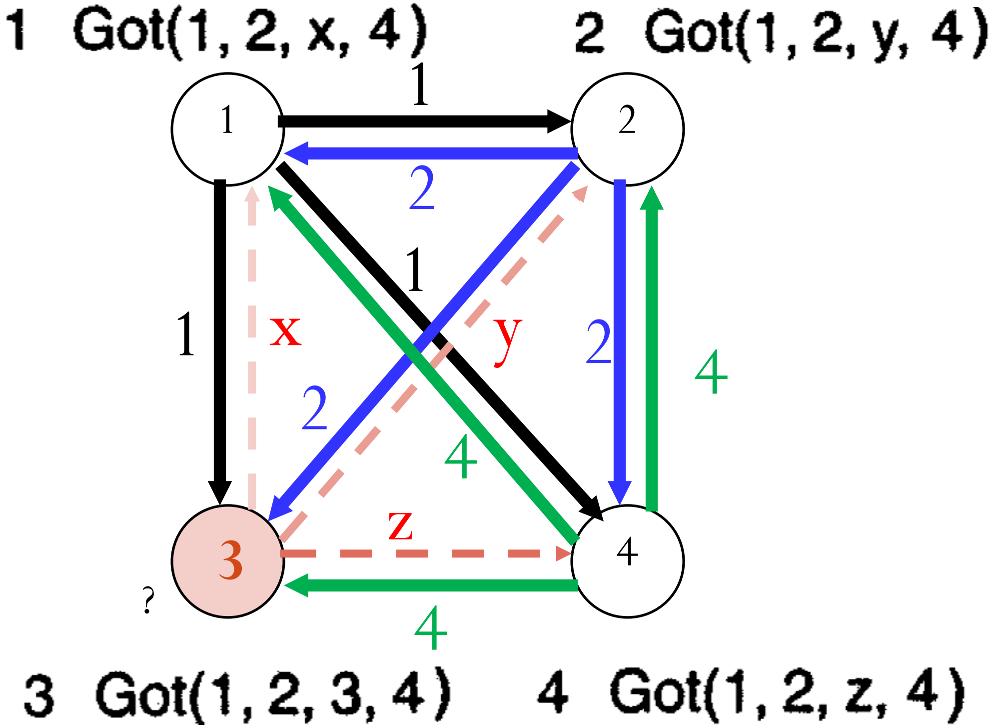
Let n=4, m=1

general 1 has 1K troop

general 2 has 2K troop

general 3 is traitor

general 4 has 4K troop



1 is not sure if 2 sends him true message

1 is not sure if 3 sends him true message

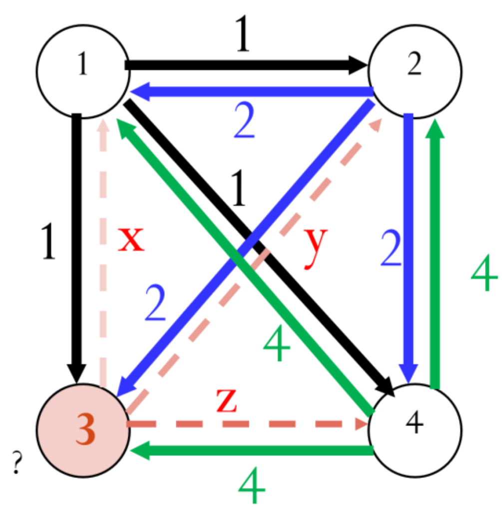
1 is not sure if 4 sends him true message

1 Got
(1, 2, y, 4)
(a, b, c, d)
(1, 2, z, 4)

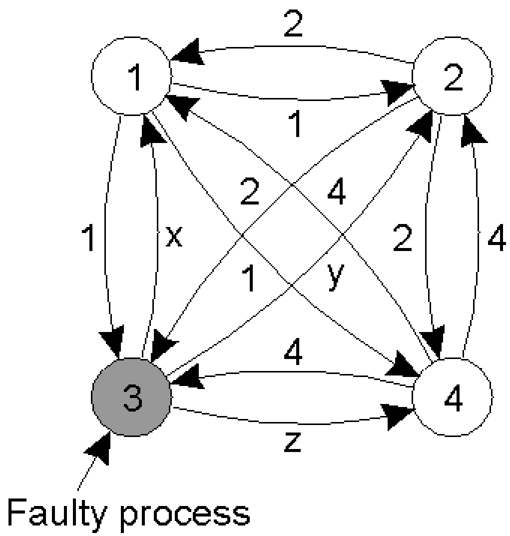
2 Got
(1, 2, x, 4)
(e, f, g, h)
(1, 2, z, 4)

4 Got
(1, 2, x, 4)
(1, 2, y, 4)
(i, j, k, l)

	P1	P2	P3	P4
step 2:	(<u>1</u>, 2, x, 4)	(1, <u>2</u>, y, 4)	(1, 2, <u>3</u>, 4)	(1, 2, z, <u>4</u>)
step 3:	(1, 2, y, 4)	(1, 2, x, 4)	(1, 2, x, 4)	(1, 2, x, 4)
	(a, b, c, d)	(e, f, g, h)	(1, 2, y, 4)	(1, 2, y, 4)
	(1, 2, z, 4)	(1, 2, z, 4)	(1, 2, z, 4)	(i, j , k, l)
step 4:	(1, 2, u, 4)	(1, 2, u, 4)	~~(1, 2, u, 4)~~	(1, 2, u, 4)



Agreement in Faulty Systems (1)



(a)

1	Got(1, 2, x, 4)
2	Got(1, 2, y, 4)
3	Got(1, 2, 3, 4)
4	Got(1, 2, z, 4)

(b)

1 Got	2 Got	4 Got
(1, 2, y, 4)	(1, 2, x, 4)	(1, 2, x, 4)
(a, b, c, d)	(e, f, g, h)	(1, 2, y, 4)
(1, 2, z, 4)	(1, 2, z, 4)	(i, j, k, l)

(c)

- The Byzantine generals problem for 3 loyal generals and 1 traitor.

a) The generals announce their troop strengths (in units of 1 kilosoldiers).

b) The vectors that each general assembles based on (a)

c) The vectors that each general receives in step 3.

- Algorithm to reach agreement. They perform the following :

step 1: every general sends a message to every other general telling his strength (true or lie)

step 2: each general collects received messages to form a vector

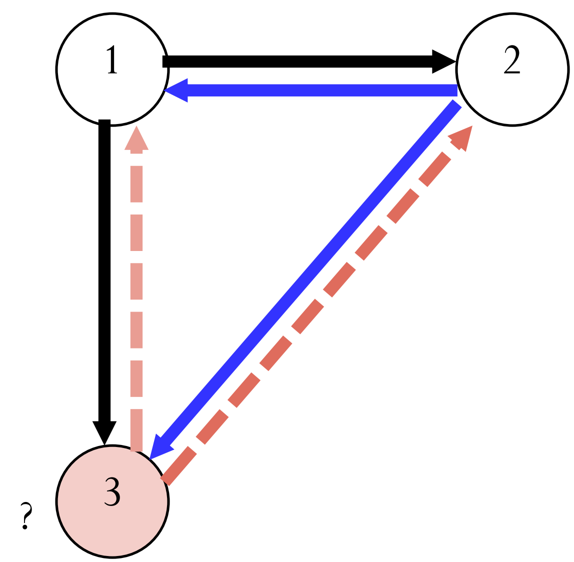
step 3: every general passes his vector to every other general

step 4: each general examines the ith element of each of the newly received vectors. If any value has a majority, that value is put into the result vector

- 3 is Byzantine faulty processor

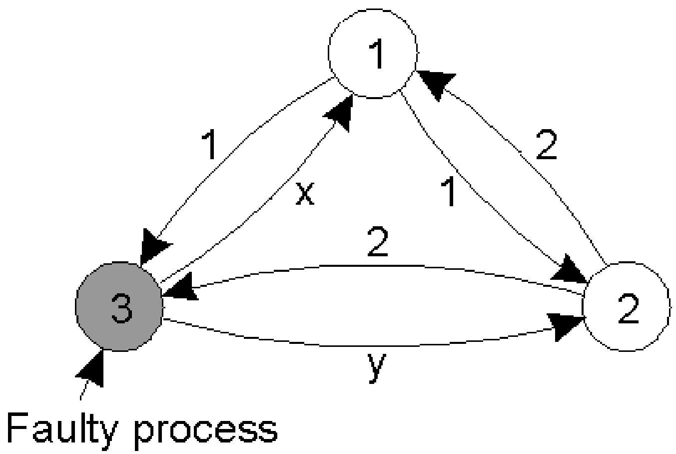
A system with m faulty processors, agreement can be achieved only if 2m+1 processors work properly, for a total of 3m+1. i.e. >2n/3

For example, let n=3, m=1



	P1	P2	P3
step 2:	($\underline{1}$, 2, x)	(1, $\underline{2}$, y)	(1, 2, $\underline{3}$)
step 3:	(1, 2, y)	(1, 2, x)	(1, 2, x)
	(a, b, c)	(d, e, f)	(1, 2, y)
step 4:	(u, u, u)	(u, u, u,)	~~(1, 2, u)~~

Agreement in Faulty Systems (2)



(a)

1 Got(1, 2, x)
2 Got(1, 2, y)
3 Got(1, 2, 3)

(b)

1 Got	2 Got
(1, 2, y)	(1, 2, x)
(a, b, c)	(d, e, f)

(c)

- The same as in previous slide, except now with 2 loyal generals and one traitor.

Interactive Consistency (IC)

- The question is whether for given $m, n > 0$, it is possible to devise an algorithm based on an exchange of messages that will allow each nonfaulty processors to compute a vector of values with an element for each of the n processors, such that

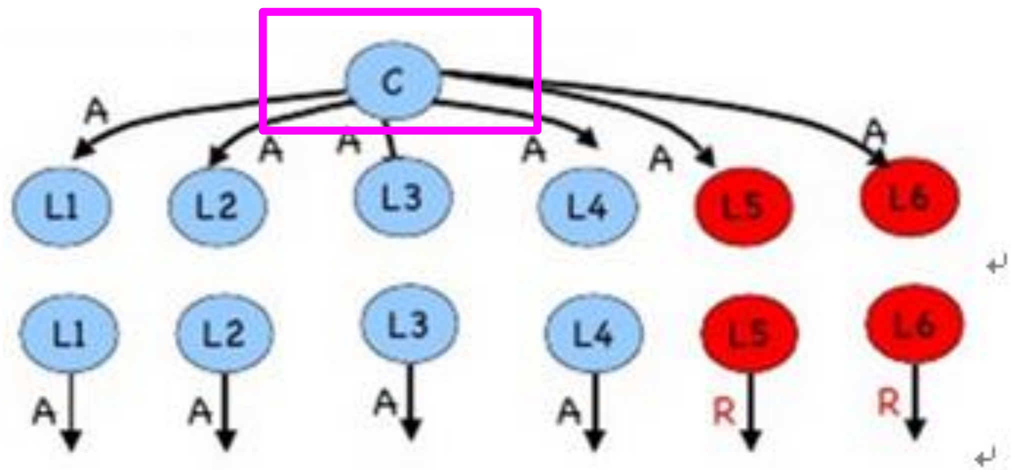
- (1) the nonfaulty processors compute exactly the ***same vector***;
- (2) the element of this vector corresponding to a given nonfaulty processor is the private value of that processor.

1 Got	2 Got	4 Got
(1, 2, y, 4)	(1, 2, x, 4)	(1, 2, x, 4)
(a, b, c, d)	(e, f, g, h)	(1, 2, y, 4)
(1, 2, z, 4)	(1, 2, z, 4)	(i, j, k, l)
(1, 2, u, 4)	(1, 2, u, 4)	(1, 2, u, 4)

Byzantine Generals Problem

A commanding general must send an order to his n - 1 lieutenant generals such that

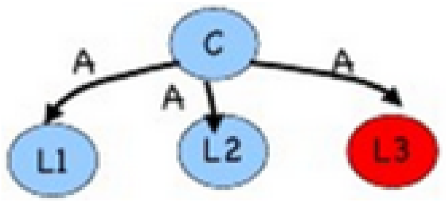
- IC1. All loyal lieutenants obey the same order.
- IC2. If the commanding general is loyal, then every loyal lieutenant obeys the order he sends.



Oral Message Algorithm

Algorithm OM(0):

1. Commander sends his value to every lieutenant
2. Each lieutenant uses the value received or "retreat" if no value received



Algorithm OM(m), m > 0:

1. Commander sends his value to every lieutenant
2. For each *i*, let v_i be the value that lieutenant *i* receives from the commander or "retreat". Lieutenant *i* acts as the commander in OM(m-1) to send the value v_i to each of the other n-2 other lieutenants
3. For each *i*, and each *j* <> *i*, let v_j be the value lieutenant *i* received from lieutenant *j* in step 2. Lieutenant *i* uses the value *majority* ($v_1, \ldots, v_{n-1}$)