

# Consensus problem

# “Byzantine Fault Tolerance for security in distributed systems”

Traitor / Adversary

Byzantine failure: no assumption on the failure mode

L. Lamport, R. Shostak, M. Pease

The Byzantine Generals Problem

ACM Trans. on Progr. Languages and Systems, 4(3),1982

# Consensus problem

The Consensus problem can be stated informally as:

how to make a set of distributed processors achieve agreement on a value sent by one processor despite a number of failures

“Byzantine Generals” metaphor used in the classical paper by [Lamport et al.,1982]

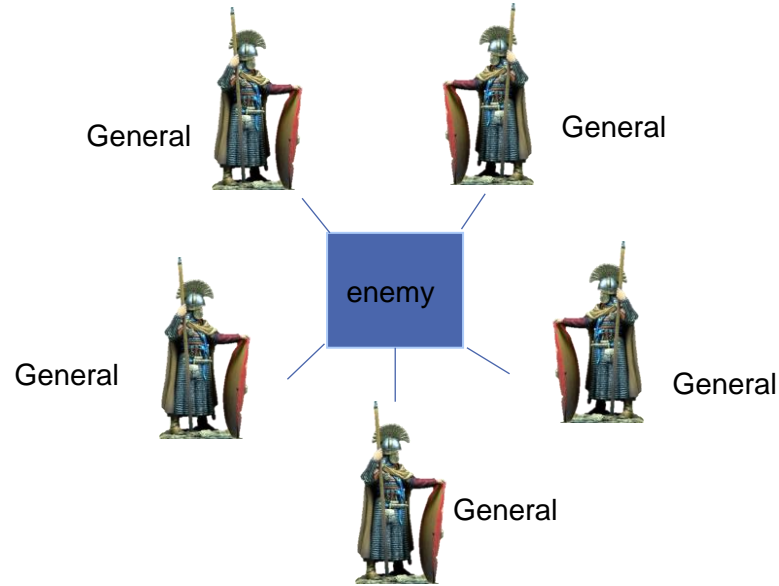
The problem is given in terms of generals who have surrounded the enemy.

Generals wish to organize a plan of action to attack or to retreat. They must take the same decision.

Each general observes the enemy and communicates his observations to the others.

Unfortunately there are traitors among generals and traitors want to influence this plan to the enemy's advantage. They may lie about whether they will support a particular plan and what other generals told them.

# Byzantine Generals Problem



General: either a loyal general or a traitor

Consensus:

A: All loyal generals decide upon the same plan of actions

B: A small number of traitors cannot cause loyal generals to adopt a bad plan

# Byzantine Generals Problem

Assume

- $n$  be the number of generals
- $v(i)$  be the opinion of general  $i$  (attack/retreat)
- each general  $i$  communicate the value  $v(i)$  by messangers to each other general
- each general final decision obtained by:  
majority vote among the values  $v(1), \dots, v(n)$

Absence of traitors:  
generals have the same values  $v(1), \dots, v(n)$  and they take the same decision

# Byzantine Generals Problem

Consensus:

A: All loyal generals decide upon the same plan of actions

B: A small number of traitors cannot cause loyal generals to adopt a bad plan

In presence of traitors:

to satisfy condition A

every general must apply the majority function to the same values  
 $v(1), \dots, v(n)$

to satisfy condition B

for each  $i$ , if the  $i$ -th general is loyal, then the value he sends must  
be used by every loyal general as the value  $v(i)$

# Interactive Consistency

Simpler situation:

- 1      Commanding general (C)
- n-1    lieutenant generals (L1, ..., L<sub>n-1</sub>)

The Byzantine commanding general C wishes to organize a plan of action to attack or to retreat; he sends the command to every lieutenant general  $L_i$

## Interactive Consistency

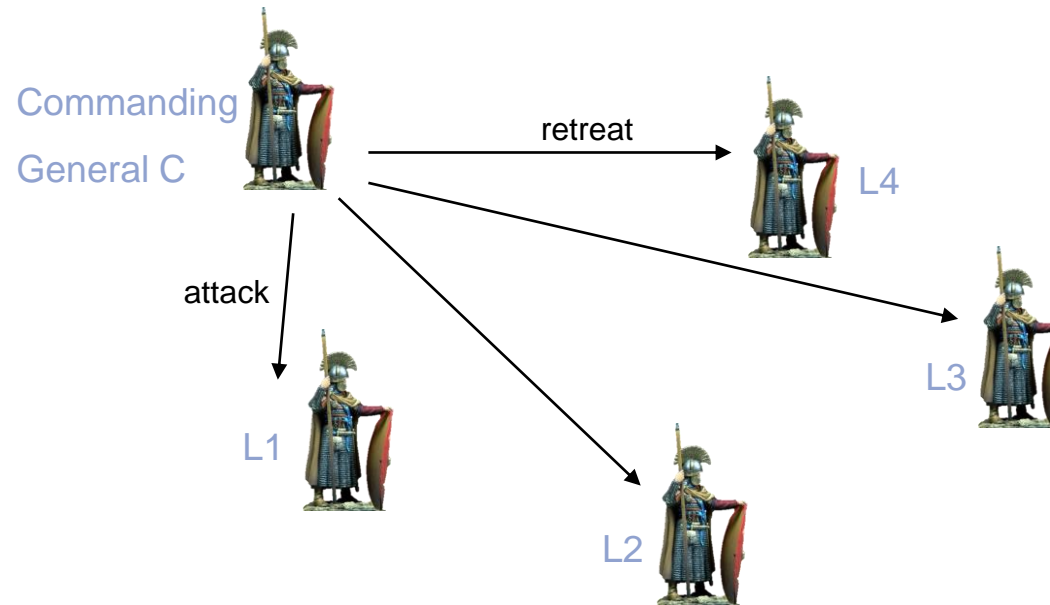
IC1:

All loyal lieutenant generals obey the same command

IC2:

The decision of loyal lieutenants must agree with the commanding general's order if he is loyal

# Byzantine Generals Problem



Commanding general is loyal: IC1 and IC2 are satisfied

Commanding general lies but sends the same command to lieutenants: IC1 and IC2 are satisfied

Commanding general lies and sends  
- attack to some lieutenant generals  
- retreat to some other lieutenant generals

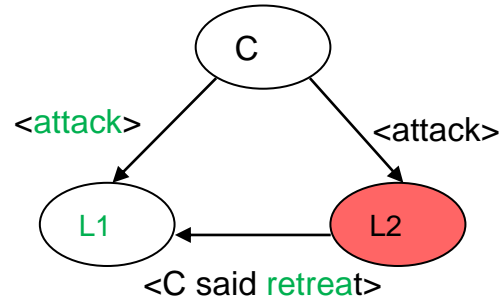
How loyal lieutenant generals may all reach the same decision either to attack or to retreat ?



# 3 Generals: one lieutenant traitor

no solution exists for 3 generals and one traitor

Case 1: L2 traitor



In this situation (two different commands, one from the commanding general and the other from a lieutenant general), assume L1 must obey the commanding general.

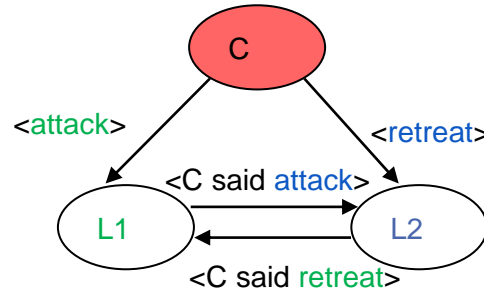
If L1 decides attack, IC1 and IC2 are satisfied.

If L1 must obey the lieutenant general, IC2 is not satisfied, therefore we fix the following rule:

if Li receives different messages, Li takes the decision he received by the commander

# 3 Generals: Commander traitor

## Case 2: C traitor



The situation is the same as before, and the same rule is applied

L1 must obey the commanding general and decides attack

L2 must obey the commanding general and decides retreat

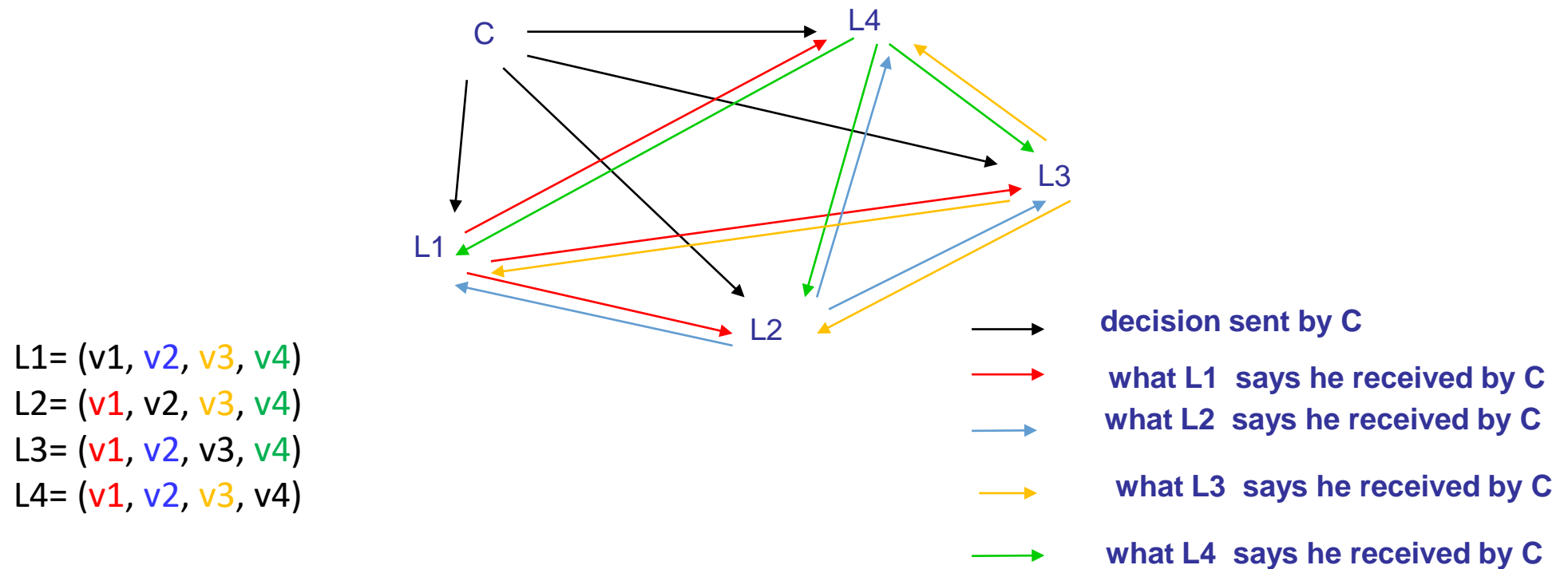
IC1 is violated

IC2 is satisfied (the commanding general is a traitor)

**To cope with 1 traitor, there must be at least 4 generals**

# Byzantine Generals Problem

Lieutenant generals send messages back and forth among themselves reporting the command received by the Commanding General.



If no traitors exist, each  $L_i$  receives three copies of the same order he has received by the Commander

# Oral Message (OM) algorithm

## Assumptions

1. the system is synchronous
2. any two processes have direct communication across a network *not prone to failure itself* and *subject to negligible delay*
3. *the sender of a message can be identified by the receiver*

In particular, the following assumptions hold

- A1. Every message that is sent by a non faulty process is correctly delivered
- A2. The receiver of a message knows who sent it
- A3. The absence of a message can be detected

# Oral Message (OM) algorithm

The Oral Message algorithm  $OM(m)$  by which a commander sends an order to  $n-1$  lieutenants, solves the Byzantine Generals Problem for  $n = (3m + 1)$  or more generals, in presence of at most  $m$  traitors.

---

majority( $v_1, \dots, v_{n-1}$ )

if a majority of values  $v_i$  equals  $v$ ,

then

majority( $v_1, \dots, v_{n-1}$ ) equals  $v$

else

majority( $v_1, \dots, v_{n-1}$ ) equals retreat

---

Deterministic majority vote on the values

A traitor commander may decide not to send any order. In this the default order “retreat” is assumed.

The function majority( $v_1, \dots, v_{n-1}$ ) returns “retreat” if there not exists a majority among values

# Oral Message (OM) algorithm

---

## Algorithm OM(0)

1. C sends his value to every  $L_i$ ,  $i \in \{1, \dots, n-1\}$
2. Each  $L_i$  uses the received value, or the value retreat if he receives no value

## Algorithm OM(m), $m > 0$

1. C sends its value to every  $L_i$ ,  $i \in \{1, \dots, n-1\}$
  2. Let  $v_i$  be the value received by  $L_i$  from C  
( $v_i = \text{retreat}$  if  $L_i$  receives no value)  
 $L_i$  acts as C in OM(m-1) to send  $v_i$  to each of the  $n-2$  other lieutenants
  3. For each  $i$  and  $j \neq i$ , let  $v_j$  be the value that  $L_i$  received from  $L_j$  in step 2 using Algorithm OM(m-1) ( $v_j = \text{retreat}$  if  $L_i$  receives no value).  
 $L_i$  uses the value of majority( $v_1, \dots, v_{n-1}$ )
- 

OM(m) is a recursive algorithm that invokes  $n-1$  separate executions of OM(m-1), each of which invokes  $n-2$  executions of OM(m-2), etc..

For  $m > 1$ , a lieutenant sends many separated messages to the other lieutenants.

# The algorithm OM(1)

4 generals, 1 traitor

Point 1

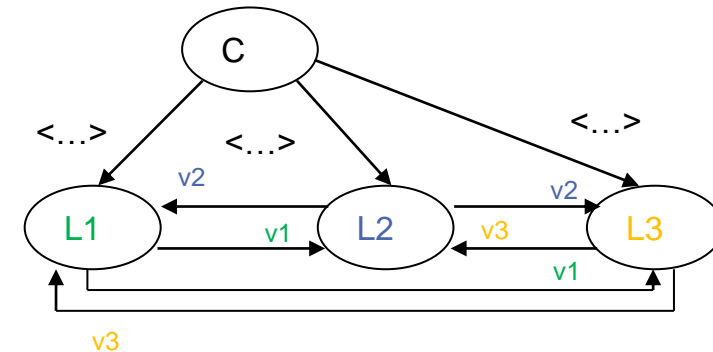
- C sends the command to L1, L2, L3.

Point 2

- L1 applies OM(0) and sends the command he received from C to L2 and L3
- L2 applies OM(0) and sends the command he received from C to L1 and L3
- L3 applies OM(0) and sends the command he received from C to L1 and L2

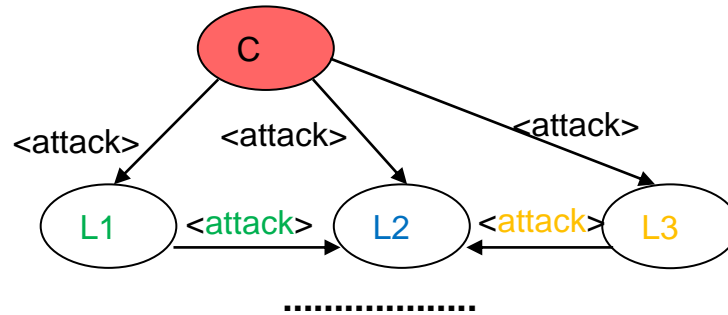
Point 3

- L1: majority(v1, v2, v3)  
//v1 command L1 says he received
- L2: majority(v1, v2, v3)  
//v3 command L3 says he received
- L3: majority(v1, v2, v3)



# 4 Generals: Commander traitor

C is a traitor but sends the same command to L1, L2 and L3



$L_i: v_1 = \text{attack}, v_2 = \text{attack}, v_3 = \text{attack}$   
 $\text{majority}(\dots) = \text{attack}$

L1, L2 and L3 are loyal. They send the same command when applying OM(0)  
IC1 and IC2 are satisfied

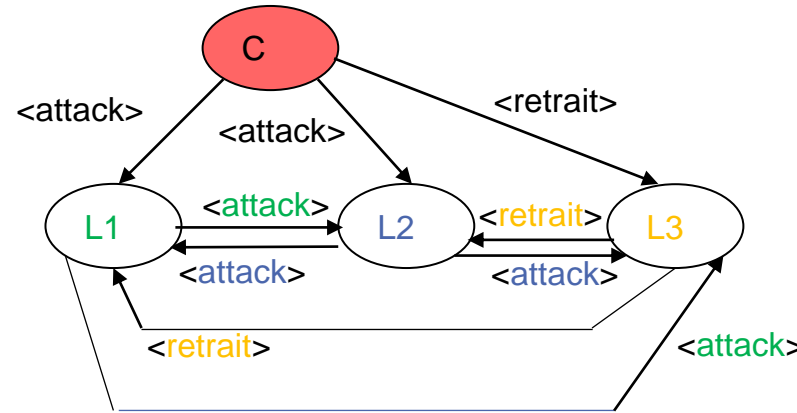


# 4 Generals: Commander traitor

C is a traitor and sends:

- attack to L1 and L2
- retrait to L3

L1, L2 and L3 are loyal.



L1: v1 = attack, v2 = attack, v3 = retrait      majority(...)= attack

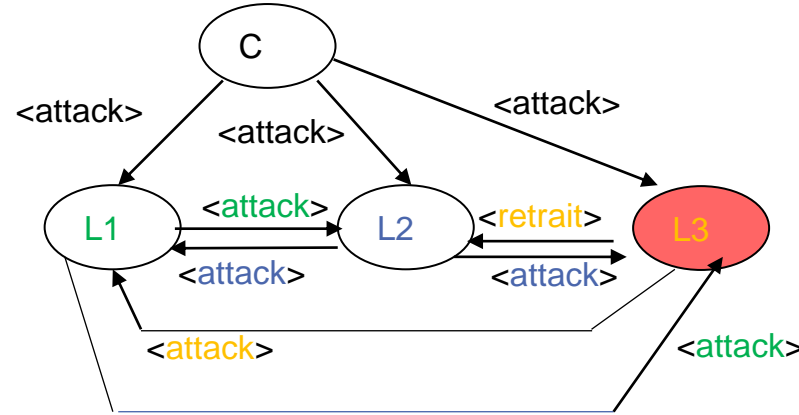
L2: v1 = attack, v2 = attack, v3 = retrait      majority(...)= attack

L3: v1 = attack, v2 = attack, v3 = retrait      majority(...)= attack

IC1 and IC2 satisfied

# 4 Generals: one Lieutenant traitor

- A lieutenant is a traitor
- L3 is a traitor:  
sends retrait to L2 and attack to L1



L1: v1 = attack v2 = attack, v3 = attack

majority(...) = attack

L2: v1 = attack v2 = attack, v3 = retrait

majority(...) = attack

IC1 and IC2 satisfied

at least two of the values received by L1 (L2) must be equal to the value sent by C: the one received directly from C and the one relayed by another loyal general.

# Oral message (OM) Algorithm

The following theorem has been formally proved:

*Theorem:*

For any  $m$ , algorithm  $OM(m)$  satisfies conditions IC1 and IC2 if there are more than  $3m$  generals and at most  $m$  traitors. Let  $n$  the number of generals:

$$n \geq 3m + 1$$

4 generals are needed to cope with 1 traitor;

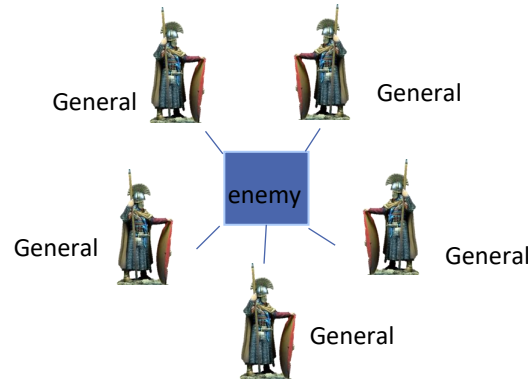
7 generals are needed to cope with 2 traitors;

10 generals are needed to cope with 3 traitors

.....

# Byzantine Generals Problem

## Original Byzantine Generals Problem



Solved assigning the role of commanding general to every lieutenant general, and running the algorithms concurrently

General agreement among  $n$  processors,  $m$  of which could be faulty and behave in arbitrary manners.

No assumptions on the characteristics of faulty processors

Conflicting values are solved taking a deterministic majority vote on the values received at each processor (completely distributed).

# Byzantine Generals Problem

Solutions of the Consensus problem are expensive

OM(m):

each  $L_i$  waits for messages originated at C and relayed via m others  $L_j$

OM(m) requires

$n = 3m + 1$  nodes

$m+1$  rounds

message of the size  $O(n^{m+1})$  - message size grows at each round

Algorithm evaluation using different metrics:

number of fault processors / number of rounds / message size

In the literature, there are algorithms that are optimal for some of these aspects.

# Byzantine Generals Problem

- The ability of the traitor to lie makes the Byzantine Generals problem difficult

Restrict the ability of the traitor to lie

A solution with signed messages:

allow generals to send unforgeable signed messages (authenticated messages)

Byzantine agreement becomes much simpler

Assumptions:

(a) The signature of a loyal general cannot be forged, and any alteration of the content of a signed message can be detected

(b) Anyone can verify the authenticity of the signature of a general

No assumptions about the signatures of traitor generals

# Signed messages

Let  $V$  be a set of orders. The function  $\text{choice}(V)$  obtains a single order from a set of orders

---

For  $\text{choice}(V)$  we require:

$\text{choice}(\emptyset) = \text{RETREAT}$

$\text{choice}(V) = v$                       if  $V$  consists of the single element  $v$

$\text{choice}(V) = \text{RETREAT}$       if  $V$  consists of more than 1 element

---

- $x:i$         denotes the value  $x$  signed by general  $i$
- $v:j:i$       denotes the value  $v$  signed by  $j$  ( $v:j$ ) and then  
                 the value  $v:j$  signed by  $i$

General 0 is the commander

For each  $i$ ,  
 $V_i$  contains the *set*  
*of properly signed orders* that  
lieutenant  $L_i$  has received so far

# Signed messages SM(m) algorithm

---

## Algorithm SM(m)

- $V_i = \emptyset$
1. C signs and sends its value to every  $L_i$ ,  $i \in \{1, \dots, n-1\}$
  2. For each  $i$ :
    - (A) if ( $L_i$  receives  $v:0$  from the commander  
and he has not yet received any order)  
then  $V_i = \{v\};$   
sends  $v:0:i$  to every other  $L_j$
    - (B) if  $L_i$  receives  $v:0:j_1:\dots:j_k$  and  $v \notin V_i$   
then  $V_i = V_i \cup \{v\};$   
if  $k < m$  then  
sends  $v:0:j_1:\dots:j_k:i$  to every other  $L_j$ ,  $j \notin \{j_1, \dots, j_k\}$
  3. For each  $i$ : when  $L_i$  will receive no more msgs, he obeys the order  $\text{choice}(V_i)$

## Observations:

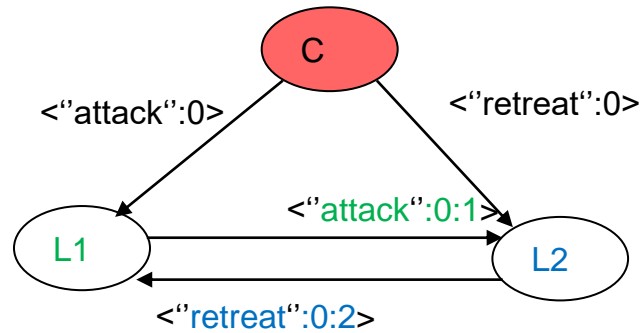
- $L_i$  ignores msgs containing an order  $v \in V_i$
- If  $L_i$  is the  $m$ -th lieutenant that adds the signature to the order, then the message is not relayed to anyone.
- Time-outs are used to determine when no more messages will arrive



# Signed messages

3 generals, 1 traitor

C is a traitor and  
sends:  
attack to L1 and L2  
retreat to L3



$V1 = \{\text{"attack"}, \text{"retreat"}\}$

$V2 = \{\text{"retreat"}, \text{"attack"}\}$

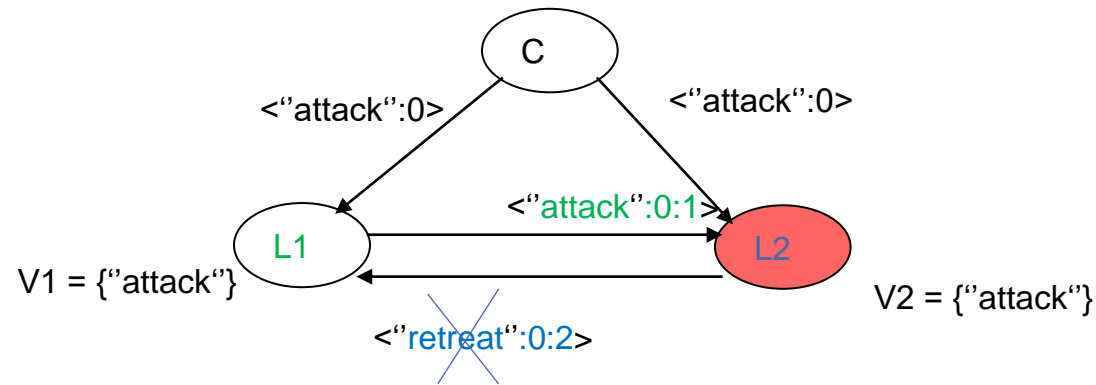
SM(1): Li stops re-sending  
messages after one round

- L1 and L2 obey the order **choice**({"attack", "retreat"})
- L1 and L2 know that **C** is a traitor because the signature of **C** appears in two different orders

# Signed messages

3 generals, 1 traitor

C is a traitor and  
sends:  
attack to L1 and L2  
retreat to L3



SM(1): Li stops re-sending  
messages after one round

- L2 obeys the order **choice({"attack"})**
- L1 knows that **L2** is a traitor because (i) the signature of **C** cannot be forged, and (ii) any alteration of the content of a signed message can be detected

The following theorem asserting the correctness of the algorithm has been formally proved.

*Theorem :*

For any  $m$ , algorithm  $SM(m)$  solves the Byzantine Generals Problem if there are at most  $m$  traitors.

# Remarks

Assumption A1.

Every message that is sent by a non faulty process is delivered correctly

Assumption A2.

The receiver of a message knows who sent it

Assumption A3:

The absence of a message can be detected

Assumption A4:

- (a) a loyal general signature cannot be forged, and any alteration of the content of a signed message can be detected
- (b) anyone can verify the authenticity of a general signature

# Impossibility result

Asynchronous distributed system:

no timing assumptions (no bounds on message delay, no bounds on the time necessary to execute a step)

Asynchronous model of computation is attractive.

- Applications programmed on this basis are easier to port than those incorporating specific timing assumptions.
- Synchronous assumptions are at best probabilistic: in practice, variable or unexpected workloads are sources of asynchrony

# Impossibility result

Consensus cannot be solved deterministically in an asynchronous distributed system that is subject even to a single crash failure [Fisher et al. 1985]

**difficulty of determining whether a process has actually crashed or is only very slow**

Stopping a single process at an inopportune time can cause any distributed protocol to fail to reach Consensus

Circumventing the problem: Adding Time to the Model (using the notion of partial synchrony), Randomized Byzantine consensus, Failure detectors, etc ...

# Example: processor synchronization

loosely synchronization

guarantee that different processors allocated to a task are executing the same iteration, do not need tight synchronization to the instruction or clock level

median clock algorithm

each clock observes every other clock and sets itself to the median of the values that he sees

**The median computed by two different processes will be the same if the set of clock values they obtain are the same.**

# Clock synchronization

Without loss of generality we can assume  $\text{Clock A} < \text{Clock B}$ . Let C faulty.

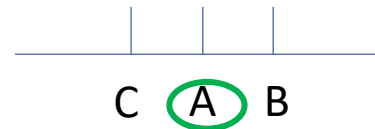
Assume A and B observe the same value for the clock C (no Byzantine faults)

Proc C  
faulty

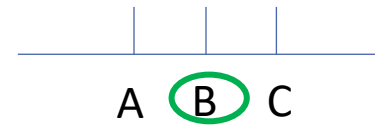
Proc A

Proc B

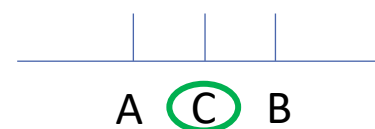
Case 1)  
 $\text{Clock C} < \text{A, B}$



Case 2)  
 $\text{Clock C} > \text{A, B}$



Case 3)  
 $\text{Clock A} < \text{C} < \text{B}$



The median clock algorithm  
satisfies the Assumption

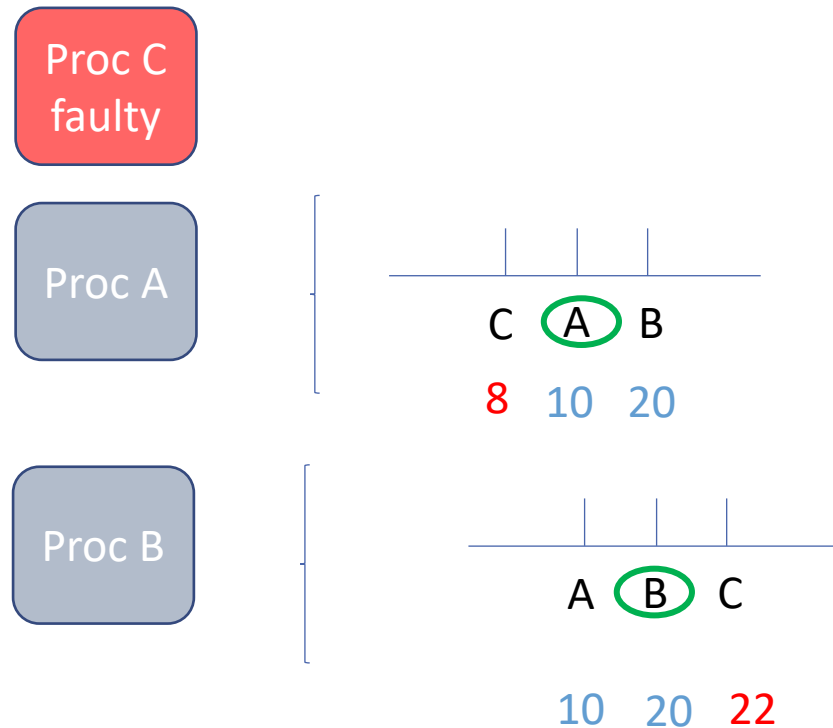
The weakness of this algorithm is  
the Byzantine fault.  
Processes may obtain different  
values for the clock of the faulty  
process.

# Clock synchronization

Let clock A < clock B. (A=10, B=20)

Assume Byzantine failure mode of clock C. For example,

- proc A sees a value for clock C that is slightly earlier than its own value (8)
- proc B sees a value for clock C that is slightly later than its own value (22)



Processors A and B will both see their own value as the median value, and they do not change it.

median clock algorithm fails



# Consensus on the value of each Clock

**The median computed by two different processes will be the same if the set of clock values they obtain are the same.**

1. Processes reach the consensus on the clock value of the other processes.
2. Processes apply the median value algorithm

Reach the consensus on the clock value of the other processes:

apply the OM(m) algorithm assuming that

*processes send their clock and  
the median value is used as majority function*

# Consensus problem on a *number* sent by process *r*

$N=3+1$  processes. At most 1 faulty process.

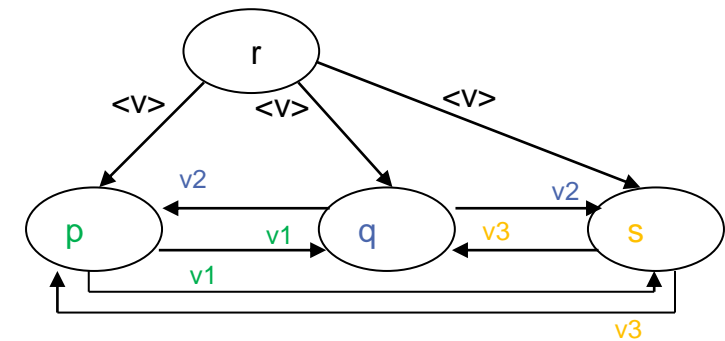
$OM(m)$  on a *number sent by a process*

Assume we use the *median value as majority* function.

**No faulty nodes.**

All processes receive three copies of the same number  $v$ , the one received directly by  $r$  and two other copies received by the other two nodes.

$$\text{median-value}(v, v, v) = v$$



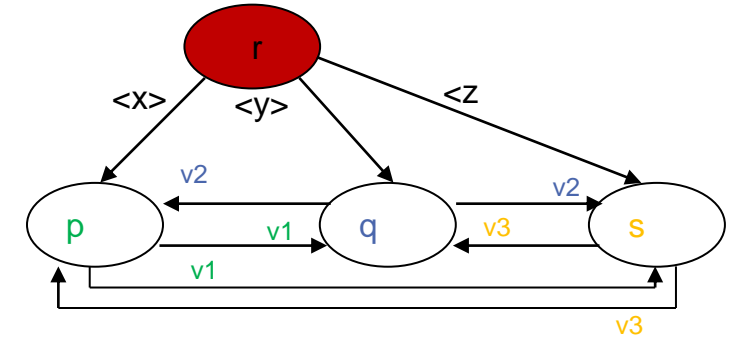
$$v1=v2=v3=v$$

# Consensus problem on a *number*

**Let  $r$  be the faulty node.**

The other three processes are not faulty. They send the value they received by  $r$ . All the node choose the median of the same set of numbers. All the nodes choose the same number:

$\text{median-value}(x, y, z)$

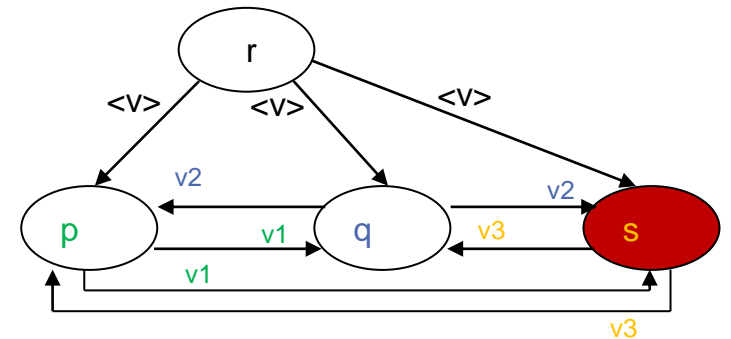


**Let  $s$  be the faulty process.**

At least two of the values received by the non faulty processes must be equal to the value sent by  $r$ : the one received directly from  $r$  and the one relayed by another non faulty node.

With three values, if at least two are equal to  $v$ , the median is  $v$ . All the nodes choose the same number:

$p: \text{median-value}(v, v, ?) = v$      $q: \text{median-value}(v, v, ?) = v$



# Other references

[Fisher et al., 1985] M. Fisher, N. Lynch, M. Paterson. Impossibility of Distributed Consensus with one faulty process.

Journal of the Ass. for Computing Machinery, 32(2), 1985.

[Chandra et al. 1996] T. D. Chandra, S. Toueg, Unreliable Failure Detectors for Reliable Distributed Systems. Journal of the Ass. For Computing Machinery, 43 (2), 1996.