Consensus problem

“Byzantine Fault Tolerance for security in distributed systems”

Traitor / Adversary

Byzantine failure: no assumption on the failure mode

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.

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]

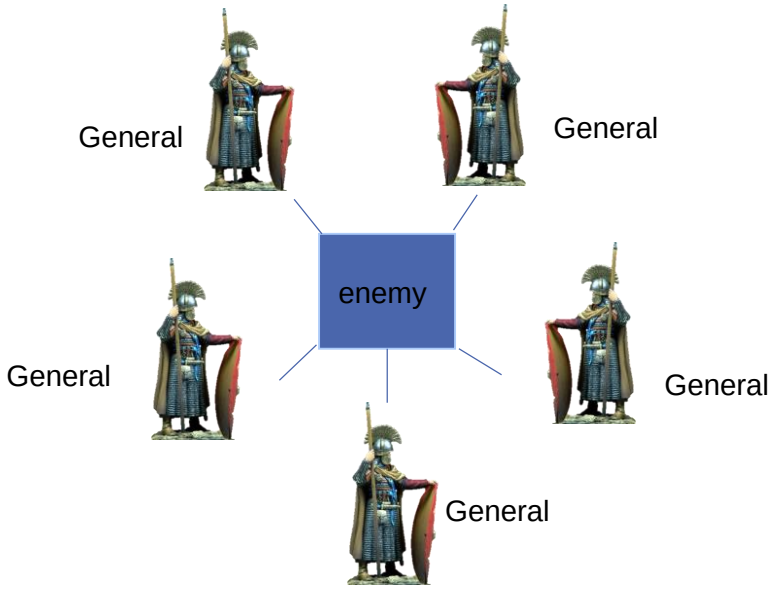
The problem is given in terms of generals who have surrounded the enemy.

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), ..., v(n)*

Absence of traitors:
generals have the same values v(1), ..., 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),...,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, ..., Ln-1)

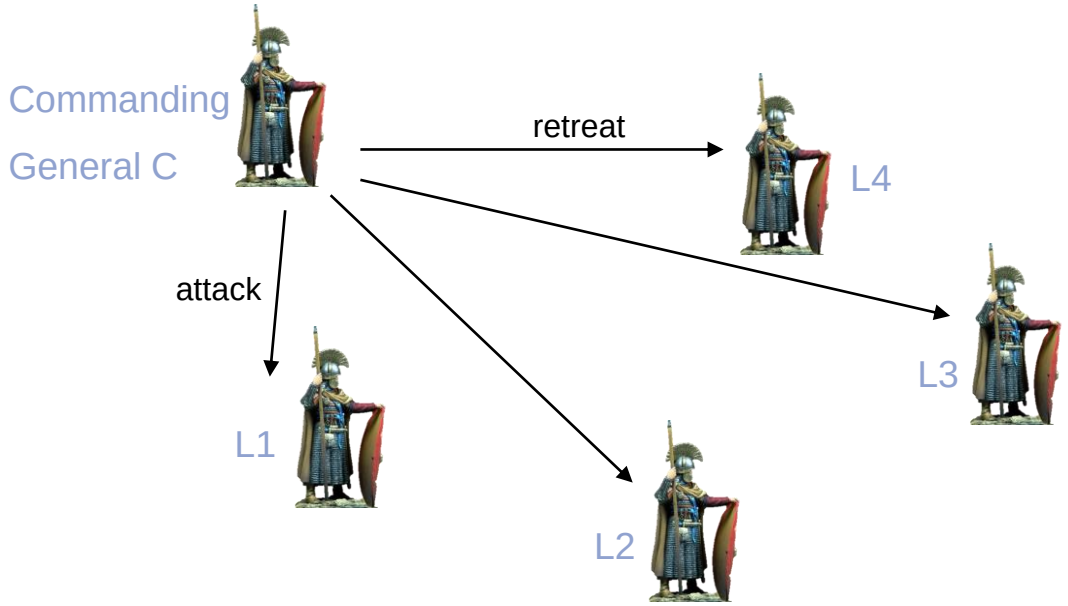
The Byzantine commanding general C wishes to organize a plan of action to attack or to retreat; he sends the command to every lieutenant general Li

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

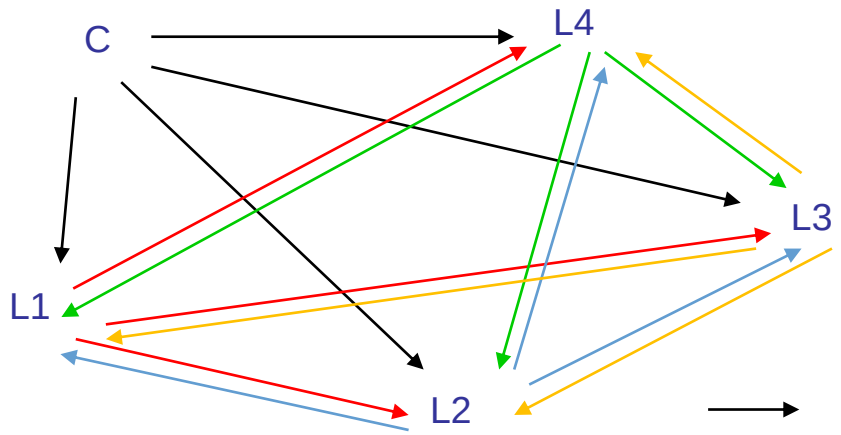
Commanding general lies but sends the same command to lieutenants:
IC1 and IC2 are satisfied

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 ?

Byzantine Generals Problem

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



L1= (v1, v2, v3, v4)
L2= (v1, v2, v3, v4)
L3= (v1, v2, v3, v4)
L4= (v1, v2, v3, v4)

decision sent by C

what L1 says he received by C

what L2 says he received by C

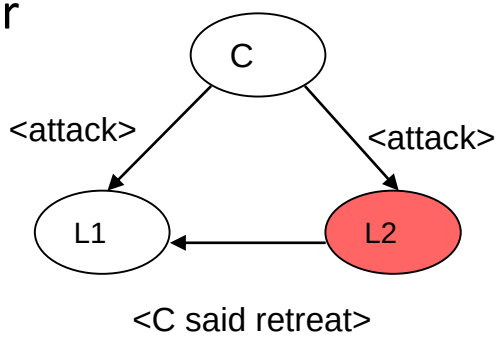
what L3 says he received by C

what L4 says he received by C

3 Generals: one lieutenant traitor

n = 3
no solution exists

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.

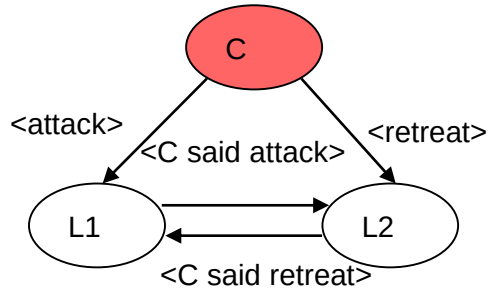
If L1 decides attack, IC1 and IC2 are satisfied.

If L1 must obey the lieutenant general, IC2 is not satisfied

RULE: if Li receives different messages, L1 takes the decision he received by the commander

3 Generals: Commander traitor

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 comanding general is a traitor)

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

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

Moreover, a traitor commander may decide not to send any order. In this case we assume a default order equal to “retreat”.

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(v1, ..., vn-1)

if a majority of values vi equals v,

then

majority(v1, ..., vn-1) equals v

else

majority(v1, ..., vn-1) equals retreat

Deterministic majority vote on the values

The function majority(v1, ..., vn-1) returns “retrait” if there not exists a majoirity among values

The algorithm

Algorithm OM(0)

1. C sends its value to every Li, $i \in \{1, ..., n-1\}$
2. Each Li uses the received value, or the value retreat if no value is received

Algorithm OM(m), m>0

1. C sends its value to every Li, $i \in \{1, ..., n-1\}$
2. Let vi be the value received by Li from C
   (vi = retreat if Li receives no value)
   Li acts as C in OM(m-1) to send vi to each of the n-2 other lieutenants
3. For each i and $j \neq i$, let vj be the value that Li received from Lj in step 2 using Algorithm OM(m-1) (vj = retreat if Li receives no value).
   Li uses the value of majority(v1, ..., vn-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 O(m-2), etc..
For m >1, a lieutenant sends many separated messages to the other lieutenants.

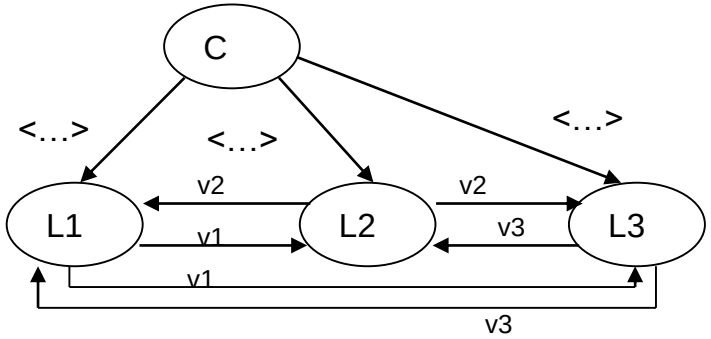
The algorithm OM(1)

4 generals, 1 traitor

Point 1

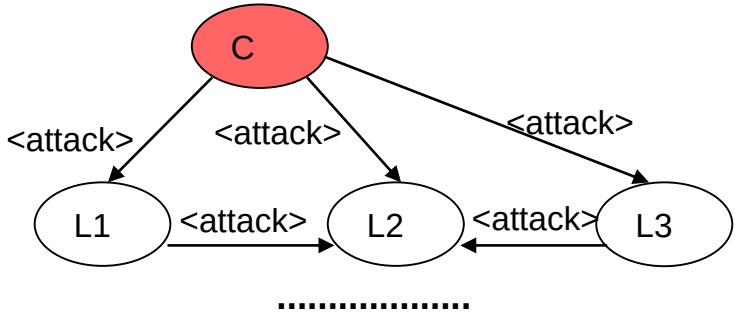
- C sends the command to L1, L2, L3.
- L1 applies OM(0) and sends the command he received from C to L2 and L3
- L2 applies OM(0) and sends the command he received from C to L1and L3
- L3 applies OM(0) and sends the command he received from C to L1 and L2

- Point 2
- L1: majority(v1, v2, v3)
- L2: majority(v1, v2, v3)
- //v1 command L1 says he received
- //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 ad L3



Li: v1 = attack, v2 =attack, v3 = attack
majority(....)= attack

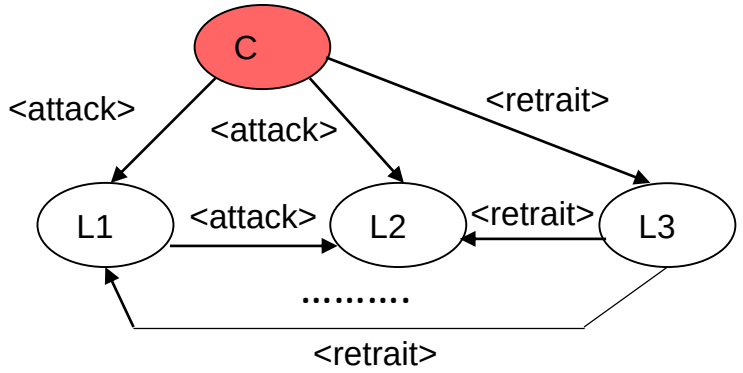
L1, L2 and L3 are loyal. They send the same command when applying OM(0)
IC1 and IC2 are satisfied

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

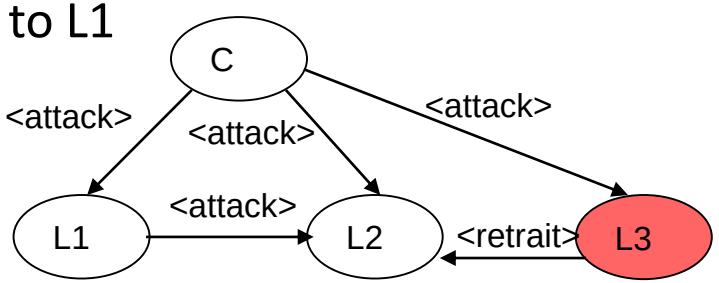
L2: v1 = attack, v2 =attack, v3 = retrait majority(...)= attack

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

IC1 and IC2 satisfied

4 Generals: one Lieutenant traitor

- A leutenant 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

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:

$$n >= 3m + 1$$

4 generals are needed to cope with 1 traitor;

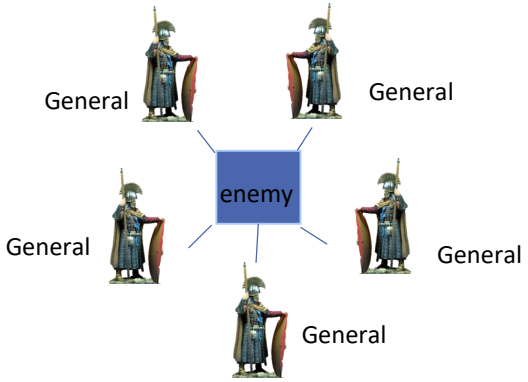
7 generals are needed to cope with 2 traitors;

10 generals are neede 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 arbirary 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

A message is authenticated if:

1. a message signed by a fault-free processor cannot be forged
2. any corruption of the message is detectable
3. the signature can be authenticated by any processors

Byzantine Generals Problem

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 choice(V) obtains a single order from a set of orders:

_____

For choice(V) we require:

choice(∅) = retreat

choice(V) = v if V consists of the single element v

choice(V) = retrait if V consists of more than 1 element

_____

General 0 is the commander
For each i, Vi contains the *set of properly signed orders* that lieutenant Li has received so far

- x:i denotes the message 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

Signed messages SM(m) algorithm

Algorithm SM(m)

$V_i = \varnothing$

1. C signs and sends its value to every Li, $i \in \{1, ..., n-1\}$
2. For each i:

(A) if Li receives v:0 and Vi is empty

then $V_i = \{v\}$;

sends v:0:i to every other Lj

(B) if Li receives v:0:j1:...:jk and $v \notin V_i$

then $V_i = V_i \cup \{v\}$;

if k < m then

sends v:0:j1:...:jk:i to every other Lj , $j \notin \{j1, ..., jk\}$

3. For each i: when Li will receive no more msgs, he obeys the order choice(Vi)

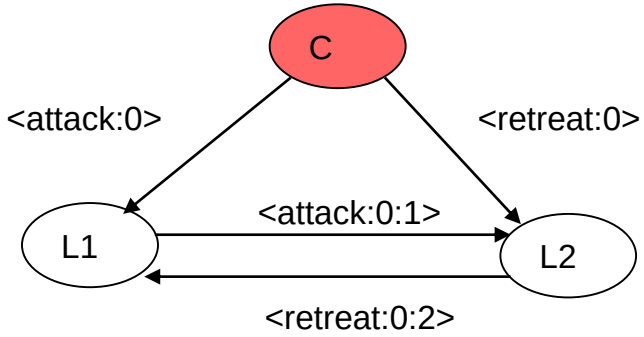
Observations:

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

Signed messages

3 generals, 1 traitor

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



- 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

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: 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

the traditional clock synchronization algorithm for reliable systems

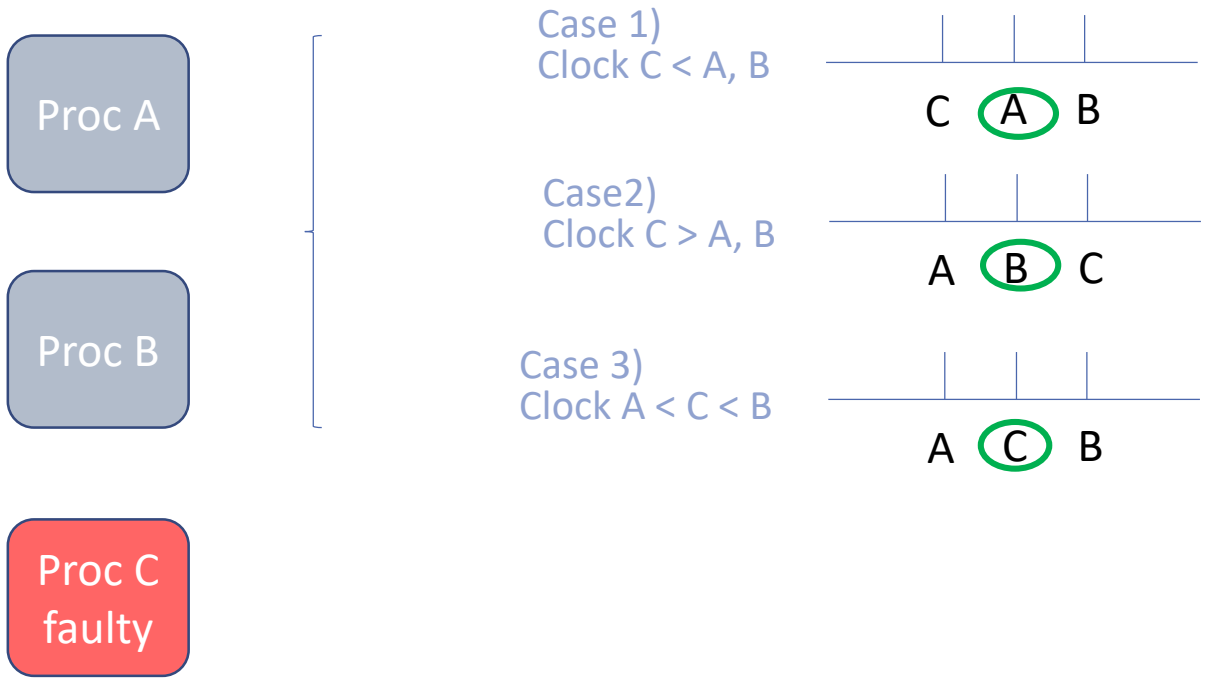
each clock observes every other clock and sets itself to the median of the values that it sees

Clock synchronization

Assumption:

in the presence of only a single fault, either the median value must be (i) the value of one of the valid clocks or else (ii) it must lie between a pair of valid clock values.

Let Clock A < Clock B.



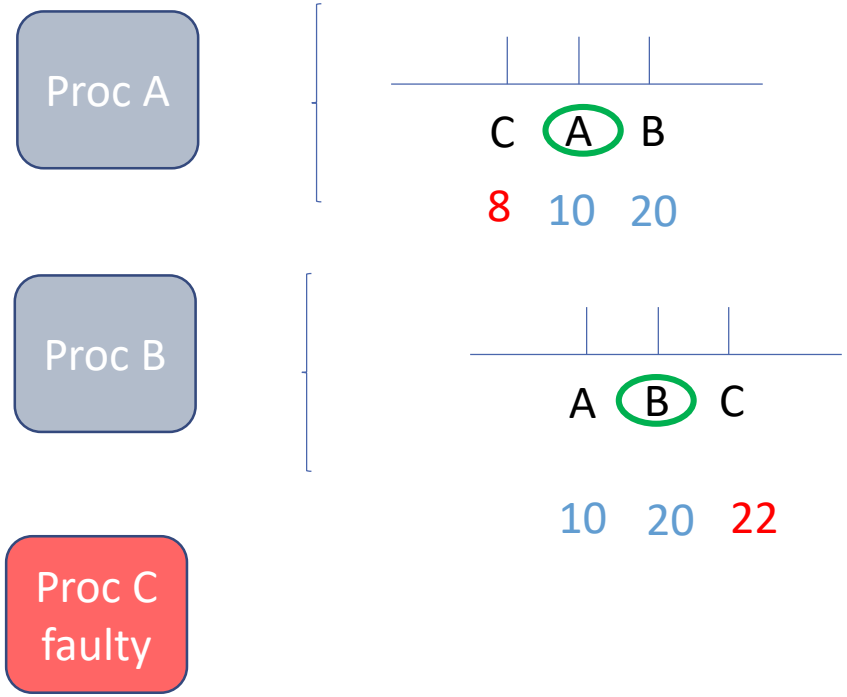
The weakness of this algorithm is the Byzantine fault, that may cause other processors to observe different values for the failing clock

Clock synchronization

Let clock A < clock B. A: B: 20 C -> Clock A=
A:10 B: C:

Assume failure mode of clock C is such that

- proc A sees a value for clock C that is slightly earlier than its own value, while
- proc B sees a value for clock C that is slightly later than its own value (Byzantine faults).



Assumption is violated

Processorss A and B will both see their own value as the median value, and therefore not change it.

To synchronise clocks SIFT applies a Consensus algorithm (5 processors)

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.