




# **Consensus Mechanisms**

# What is a consensus mechanism?

A protocol enabling nodes in a blockchain system to find an agreement on the content of the distributed ledger



The data in the ledger are not really important (as long as they are carried by valid transactions), what matters is that all nodes agree on the content (state) of the ledger:

which transactions (or states), possibly organized into blocks, are included and in which order



Every consensus mechanism is unique

There is no “best” mechanism

Choice depends on application scenario

# What do we know already?

Proof-of-Work (Bitcoin)

Proof-of-Stake (Ethereum)

# PoW v. PoS

- Change the nature of “punishment” for non-successful miners
- “Extrinsic punishment” of PoW
  - Miners waste the money paid for the electricity required for mining
- “Intrinsic punishment” of PoS
  - Validators waste ETH (i.e., a “stake” that they have in the network)

# How to choose a consensus mechanism?

When do we have certainty that a block is final?

How long does it take for a block to become final?

How many messages must be exchanged to reach consensus?

Can we handle nodes that connect/disconnect to/from the network while reaching consensus?

What does it take to “control” the consensus protocol?

# “Finality” of blocks

When does a block becomes final?



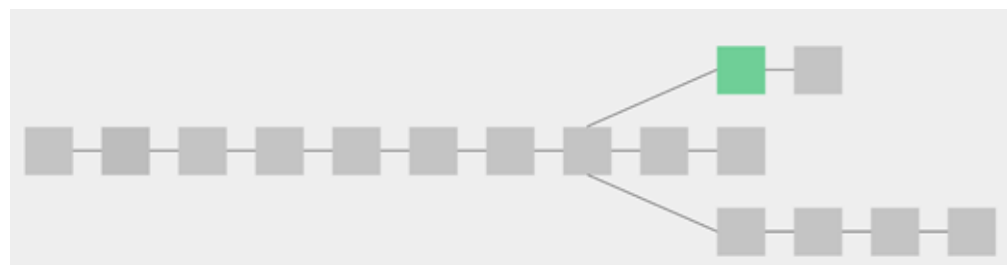
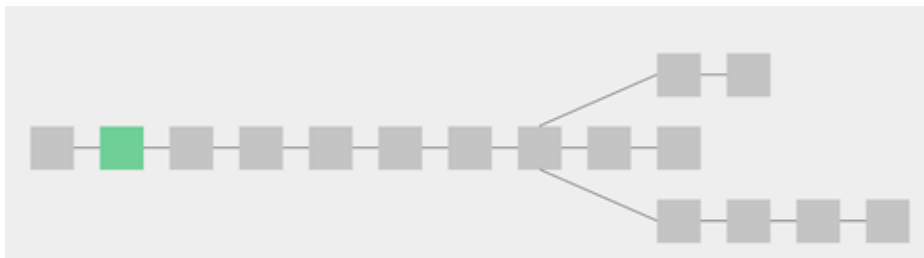
# Probabilistic finality (emergent consensus)

A block is final when “deep enough” in the main chain

Does not prevent forks

Affects performance after protocol execution  
(most time is spent waiting for a block to become final, not to “mine” a block)

It works! (Bitcoin, PoW in Ethereum until September 15th)



# Absolute finality

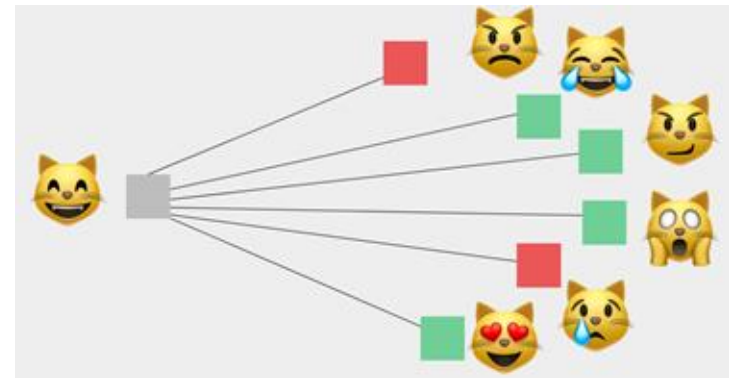
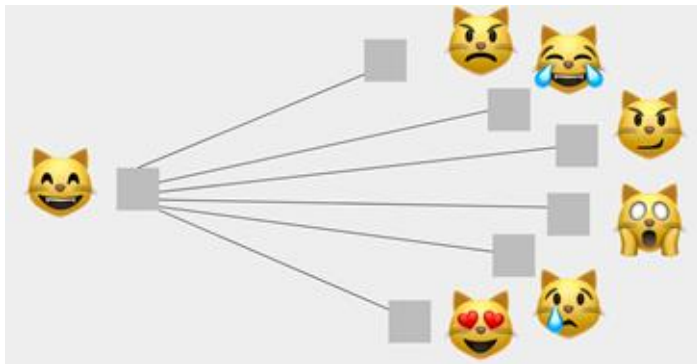
“Leader” proposes a block, finality is reached once a sufficient fraction of a “committee” members approves (validates) the new block

No need to wait for block to go deep – performance affected only by consensus protocol execution

Not scalable – a lot of messages are required among committee members in practice to reach consensus

Example:

PoS, Networks of notaries in Corda changing a state to “historic”; Ordering services in HLF



# Economic finality

Is it possible to “revert” a block and how much would it cost?

(Similarly, is it possible to control which transactions/blocks/states are recorded?)

# Economic Finality

With probabilistic finality, blocks may be reverted

How much does it cost to revert a block?  
(or to control which blocks go in the blockchain?)

PoW

- Cost of acquiring 51% mining power (HW/SW) and running miners (electricity)

# Economic Finality

With absolute finality, an attacker may obtain control of a sufficient number of approval votes and force invalid/malicious blocks in the blockchain

## PoS

- Cost of “paying” a sufficiently large stake (51% of current sum of all stakes)

See PBFT later

# PoW consensus (Bitcoin)

When do we have certainty that a block is final?

After ~6 blocks

How long does it take for a block to become final?

~60 minutes

How many messages must be exchanged to reach consensus?

None (new blocks are broadcasted when mined)

Can we handle nodes that connect/disconnect to/from the network while reaching consensus?

Yes, complete openness

What does it take to “control” the consensus protocol?

Controlling 51% of mining power

# PoS consensus (Ethereum)

When do we have certainty that a block is final?

Immediately after proposed block is validated

How long does it take for a block to become final?

Only time required to broadcast it to all nodes

How many messages must be exchanged to reach consensus?

Messages for block validation

Can we handle nodes that connect/disconnect to/from the network while reaching consensus?

Yes (validators should participate to consensus, they loose stake if they don't)

What does it take to “control” the consensus protocol?

Controlling 51% of voting power (stakes)

# Other consensus mechanisms

Proof-of-Authority (PoA)

Delegated Proof-of-Stake (DPoS)

Practical Byzantine Fault Tolerance (PBFT) – many implementations



# Proof-of-Authority (PoA)

Nodes can be nominated by showing a proof of authority

- Length of stay in the system, number of coins owned, ...

Each new block proposed by a nominated chosen randomly at each round (proportionally to their “level of authority”)

Basically, PoS without block validation

No actual consensus needed

Highly centralised, validators should always be online and uncompromised

# Delegated Proof-of-Stake (DPoS)

Nodes vote to elect a set of “witness” nodes who validate transactions and assemble blocks

- And collect rewards

Voting weight proportional to amount of currency held by node

Democratic witness election system

Witnesses have an incentive to behave correctly to be elected again in the future

# PBFT-like consensus

A different approach to consensus mechanism

Used by Hyperledger Fabric and Corda

Nodes can propose a change of the ledger (new block, transaction, or state changes)

Ordering services in HLF

Notaries of a state in Corda

A communication protocol is executed by the nodes (or a subset of them) in the network to reach consensus on the validity of the proposed change

Based on the outcome of protocol execution, the proposed change is recorded (or not) by every node

- Absolute finality

## TWO GENERALS' PROBLEM



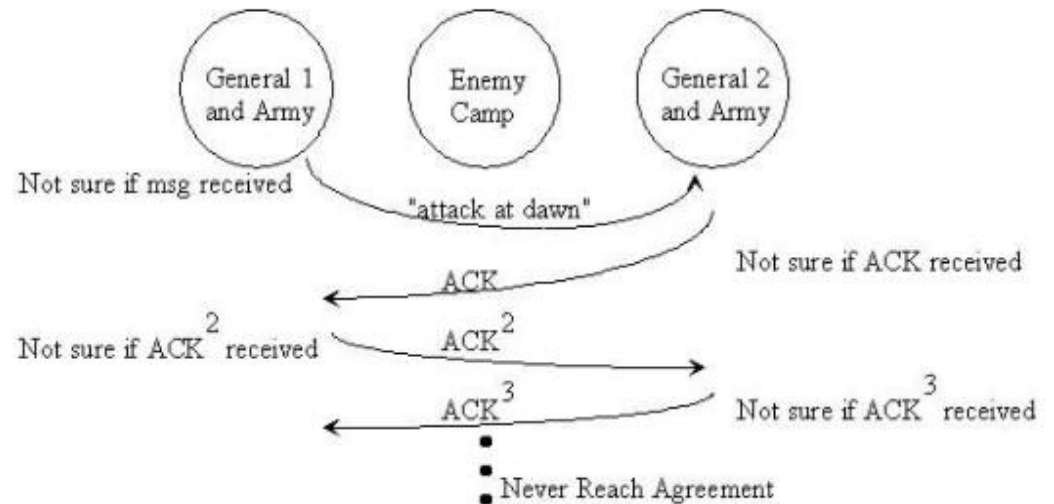
Two generals must agree on whether to attack the enemy

They use messengers that have to travel through the enemy line to communicate (= messages can get lost)

Can we design a protocol that lets the two generals reach an agreement?

No! It's **impossible**

- The generals will always be waiting for a last ack, which however may get lost



# Byzantine Generals Problem

Many generals ( $>2$ ) generals must agree on whether to attack the enemy or wait

Note: attack or wait is not important, it is only important to agree on one of the options!

Let's assume there is a Commander (who suggests what to do) and Lieutenants (who should follow the order)

Can we design a protocol to let the Commander and Lieutenants agree on what to do?

## Assumptions

- Some generals may be "traitors" (they lie or behave arbitrarily), messages among generals and lieutenants can be lost
- All "non-lying" generals and lieutenants must agree on a message sent by the commander

## The Byzantine Generals Problem



# Why is this relevant?

Blockchain is a distributed system of nodes

Generally, there are 2 ways in which a node can fail in a distributed system

- **Fail-stop** (a node stops receiving, sending, and processing messages)
  - Easy to manage
- **“Byzantine failure”** - nodes can behave in any arbitrary way, e.g., they stop working, they send contradicting messages on purpose, they send random messages,...
  - The worst possible node failure
  - Fail-stop is a subset of Byzantine failures

Solving the Byzantine General Problem means finding a way to build distributed systems that can handle byzantine failing nodes

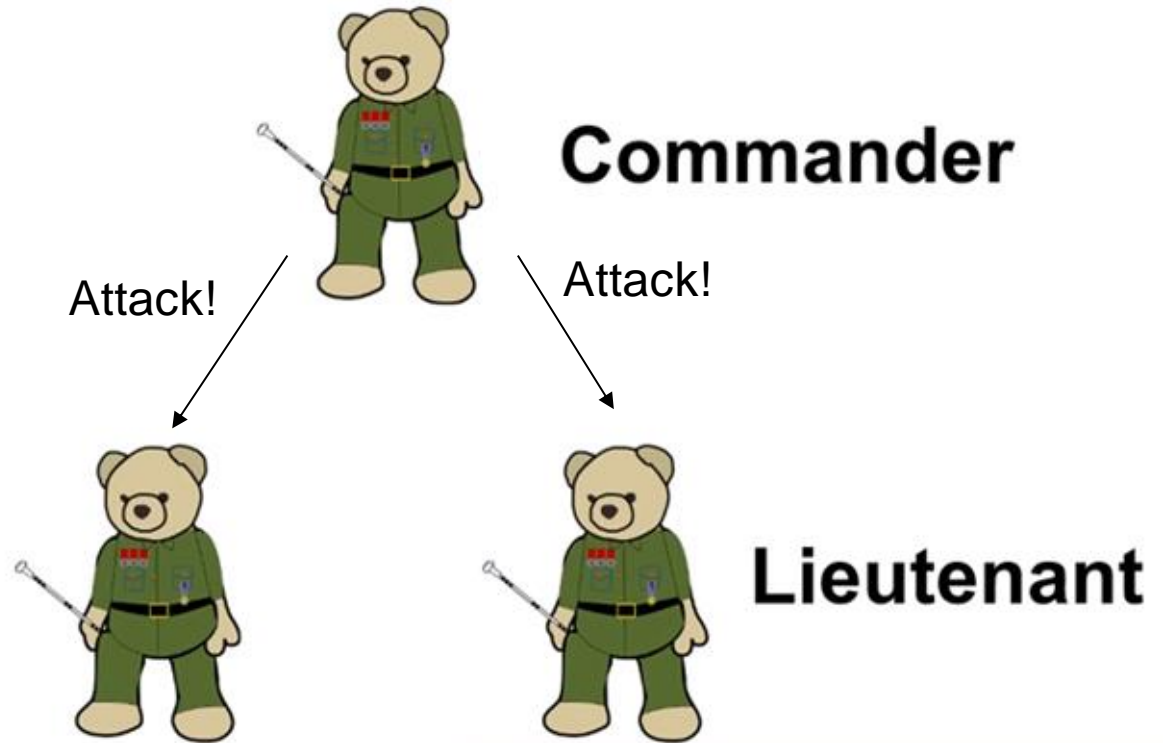
- Handling malfunctioning sensors on a plane
- Handling nodes who want to collude to take control of a blockchain!  
Example: notaries in Corda

# 1 Commander, 2 Lieutenants, no traitors

No problems if all  
generals are not  
traitors

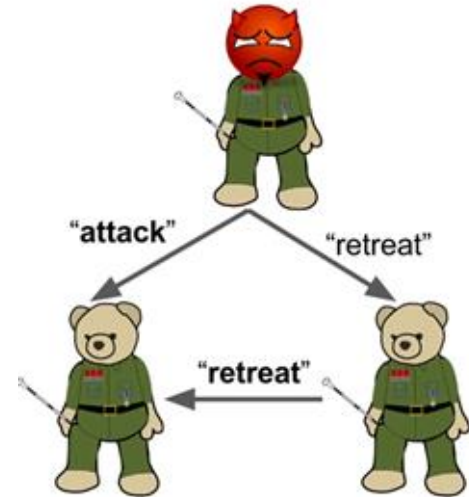
- L receive message
- L follow order

But what if one of  
them is a traitor?



# Traitor commander

- Lieutenants can share the message received to verify whether the commander is sending contradicting messages
- Seems like they can agree that commander is a traitor
- Is it that easy?



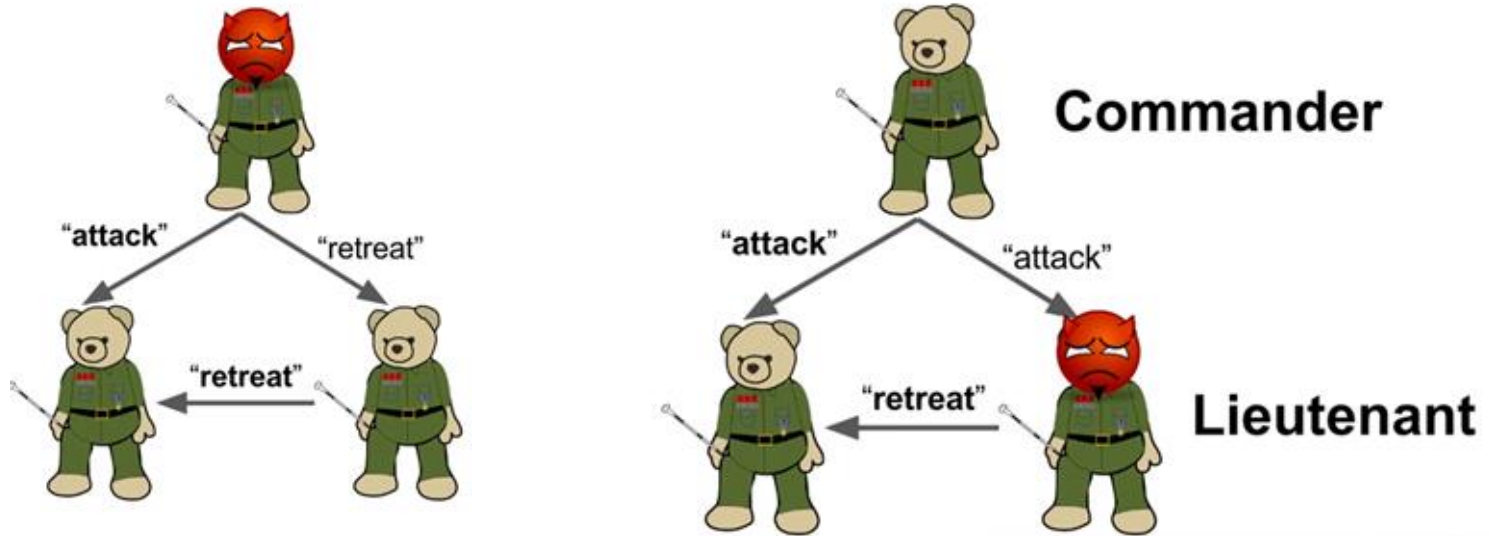



# 3 generals, 1 traitor

Lieutenant should share the message received to reach a consensus

The left-hand lieutenant receives the same messages in both cases and cannot take a decision

There is no solution for 3 Generals with 1 Traitor





How many generals do we need before we can tolerate 1 traitor?

How many generals before we can tolerate “m” traitors?

Answer:

- Traitors must be  $< 1/3$  of the participants
- 1 traitor out of 3  $\rightarrow$  No solution
- 1 traitor out of 4  $\rightarrow$  Solution exists

# PBFT – Almost there...

How do we solve the problem with less than  $1/3$  traitor generals?

Let's see the solution for  $N = 4$  for generals and 1 traitor

(Solution illustrates the principles of PBFT consensus mechanism)

- Inductive solution

# N=4 (0 traitors)

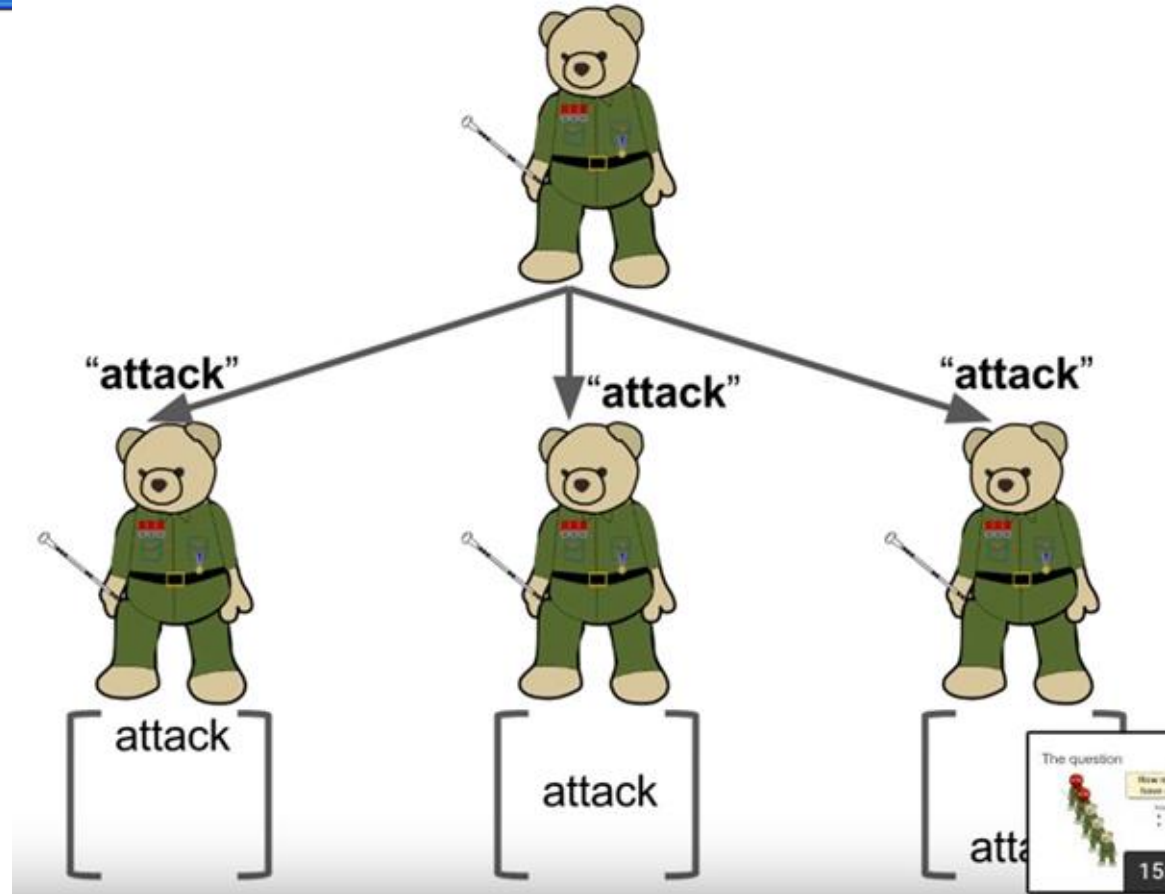
- PBFT(N),  $N > 0$  ::

1) C: send order

2) L: records order if received

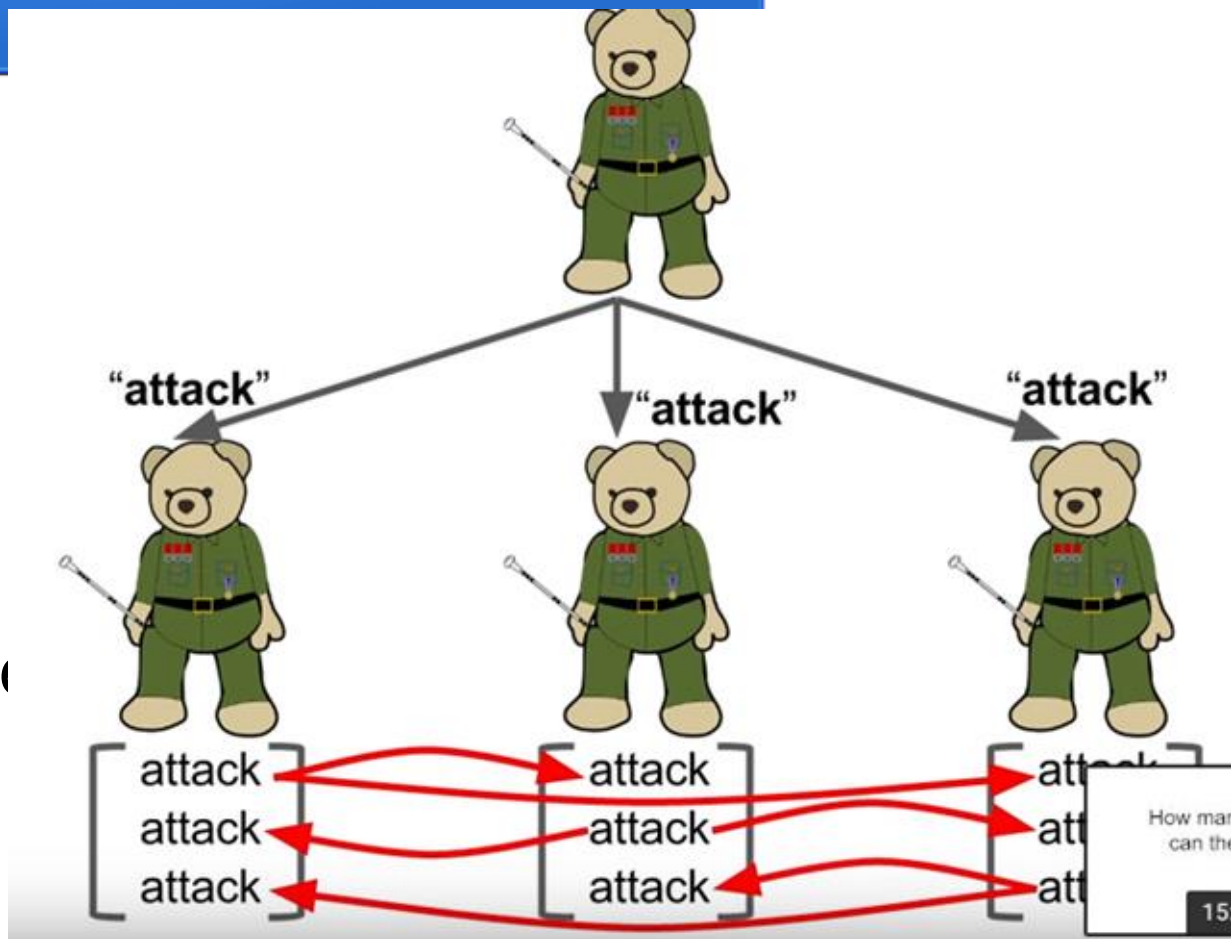
3) Use PBFT(N-1) to tell other Ls

4) Follow majority order



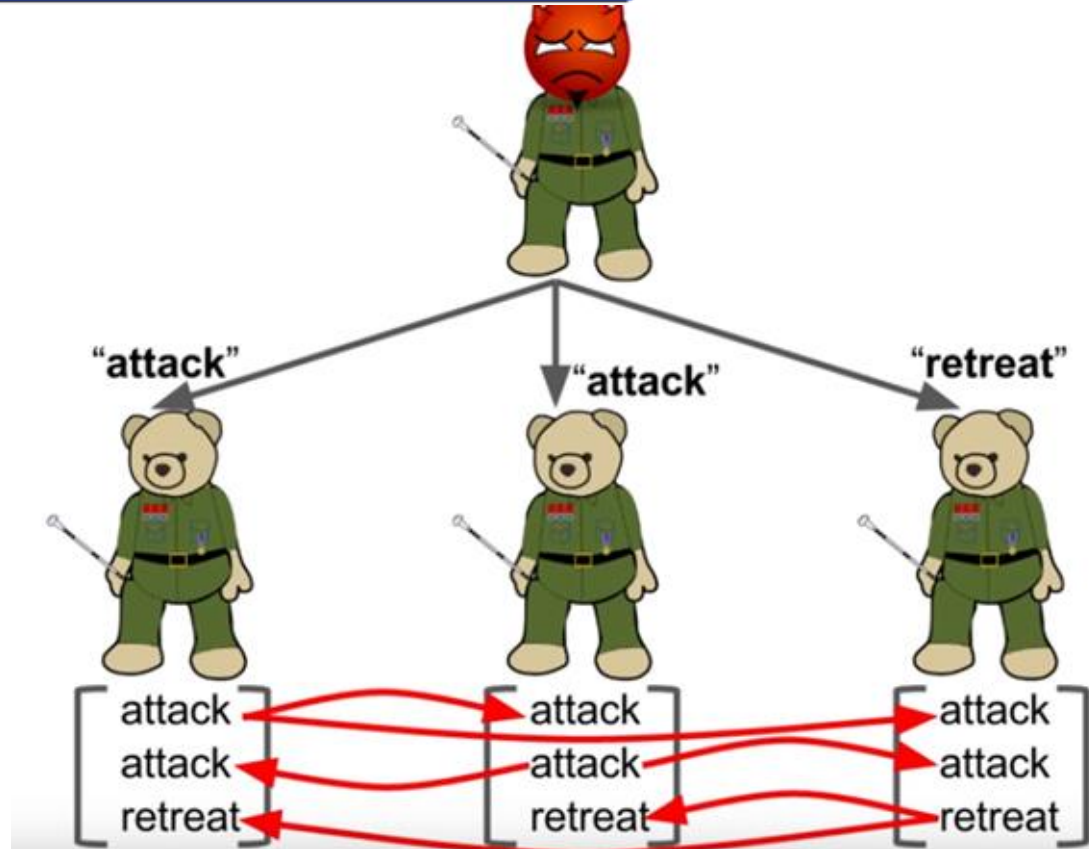
# N=4 (0 traitors)

- PBFT(N),  $N > 0$  ::  
C send order
- L: records order if  
received  
    Use PBFT(N-1) to  
    tell other Ls  
    Follow majority order



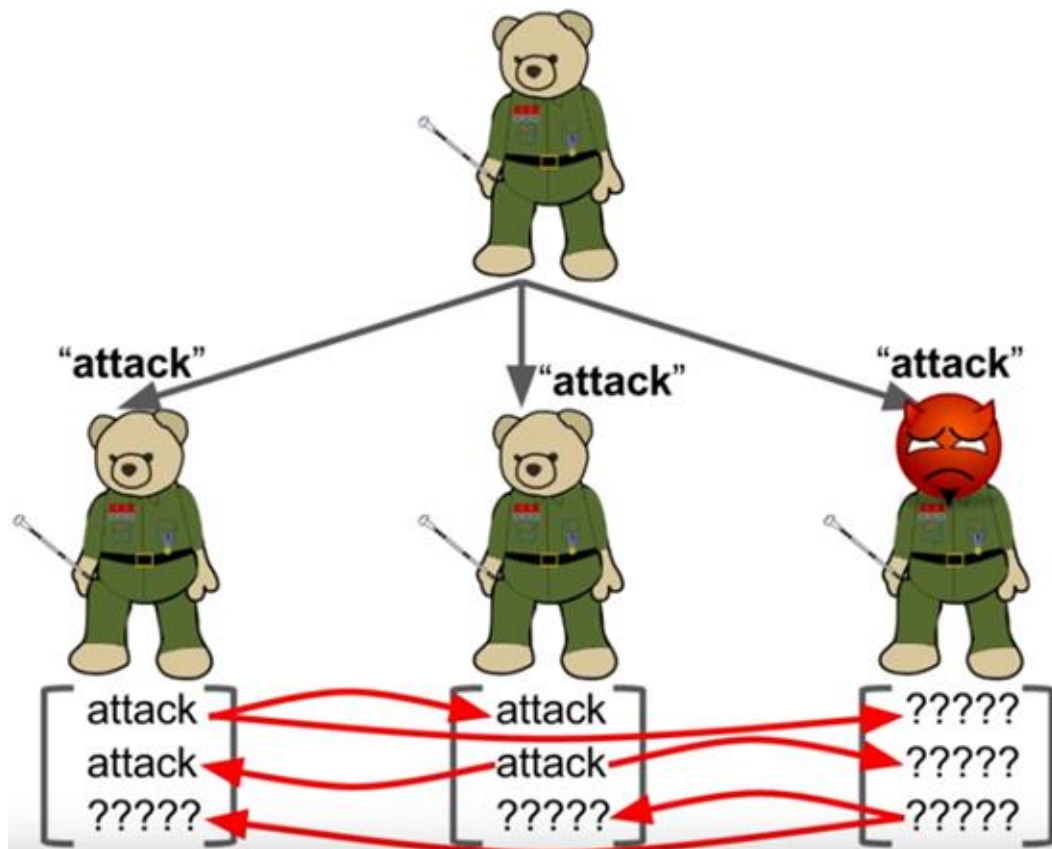
# N=4, 1 traitor (commander)

- PBFT(N),  $N > 0$  ::  
C send order
- L: records order if  
received
- Use PBFT(N-1) to tell  
other Ls
- Follow majority order



# N=4, 1 traitor (lieutenant)

- PBFT(m),  $m > 0$  ::  
C send order
- L: records order if  
received
- Use PBFT(m-1) to tell  
other Ls
- Follow majority order



# Notes

- The point is having agreement among non-traitor lieutenants, not really deciding what they agree on
  - Consensus!
- If commander is not traitor, then lieutenants agree on what he says
- Algorithm is recursive
  - Try  $N=7$ , 2 traitors
- Lot of messages must be exchanged
  - $n=0$ , ( $n$  messages)
  - $n=1$ , ( $n^2$  messages)
  - $n=2$ , ( $n^3$  messages)
  - ...
  - Number of messages is exponential with the size of the network



# PBFT and blockchain systems

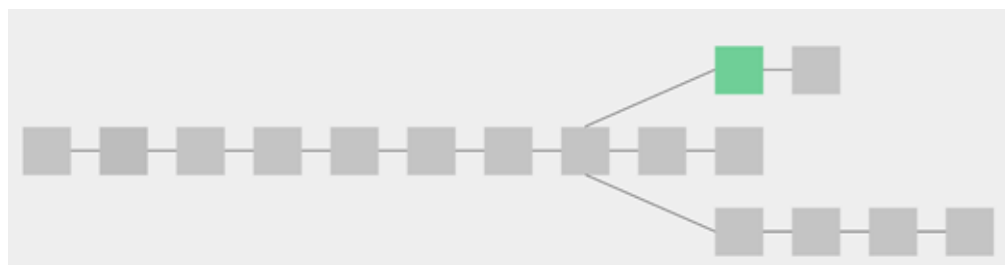
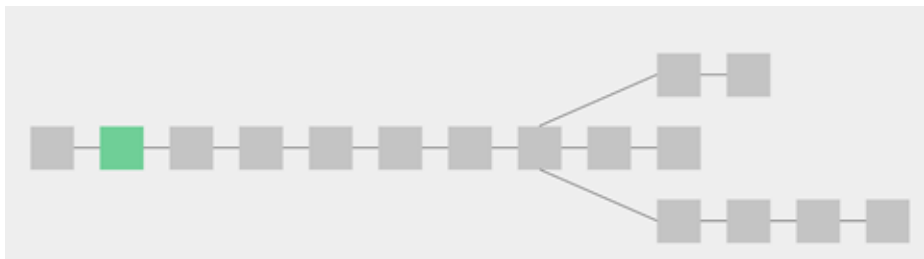
- Commander
  - A node proposing a state change (e.g. notary or ordering service)
- Lieutenants
  - Other nodes who must agree on the validity of the propose state change
- Commander lies
  - An invalid state change is submitted, other nodes must agree on the fact that it is invalid
- Lieutenant lies
  - Somebody has taken control of a validation node to prevent valid state changes or enforce invalid ones
- Consensus
  - All non-lying (=non-faulty/malicious) nodes will agree on a common decision regarding the proposed state change.

# Discussion

- BFT – Byzantine Fault Tolerance
  - Property of a consensus algorithm to be able to tolerate a number of “byzantine failing” nodes
- PBFT – Practical BFT
  - A particular algorithm to achieve BFT in a distributed systems
  - Used as consensus mechanism in private blockchains
- PoW, PoS, ...
  - Consensus mechanisms in use in public blockchains

# BFT and PoW

- PoW algorithm is also BFT (probabilistically)
- “Eventually” all nodes will agree on which blocks are (deep enough) in the main chain
  - ~60 minutes for Bitcoin
- Do not “trust” transactions in the last 6 blocks



# PBFT

Finality in PBFT is “absolute”

- After having executed the protocol, all nodes agree on whether to accept a block

Absolute finality comes at a cost

- High (exponential) number of messages
- Need to know who are the nodes before protocol begins
  - Disconnecting nodes would be “byzantine”

Makes sense in private blockchains

- Participants are known and cannot dynamically connect/disconnect

# PBFT: Evaluation

When do we have certainty that a block is final?

Immediately after a block is approved

How long does it take for a block to become final?

Immediate after execution of protocol

How many messages must be exchanged to reach consensus?

A lot! Impacts time for protocol execution

Can we handle nodes that connect/disconnect to/from the network while reaching consensus?

Node disconnecting in committee is treated as byzantine failure

What does it take to “control” the consensus protocol?

Control at least 1/3 of committee nodes to disrupt consensus