Basic Principles of Security and Blockchain

Spring 2021,

Instructor: Fabrizio Baiardi, Laura Ricci
f.baiardi@unipi.it
laura.ricci@unipi.it

Lesson 2: Distributed Consensus for blockchain

26/2/2021

CONSENSUS DECLINATIONS

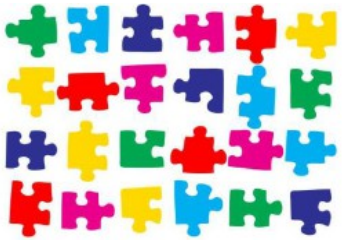
- agreement between multiple agents (processes) on some value that is needed for the evolution of the system, in an environment where some process may be unreliable or fail. The consensus protocol must be fault/attack resistant.

- different kind of consensus
  - Lottery based protocols: Proof of Work
  - Byzantine Fault Tolerance
    - synchronous environment
    - oral messages algorithm
  - Practical Byzantine fault tolerance
    - Asynchronous/Partially synchronous environment
  - Pure/Delegated Proof of Stake

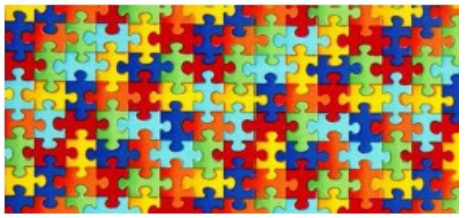
NAKAMOTO CONSENSUS

- a disruptive solution with respect to the previous based on byzantine agreement
  - no collective distributed algorithm executed by the nodes
  - no voting
  - based on Proof of Work that implements a lottery
    - who wins is "elected" as the node which decides the next state of the replicated ledger
    - is responsible of adding a block to the blockchain
- eventual consensus
  - sometimes there are inconsistencies in the replicated ledgers
  - but, evetually, all the replica converge toward a consistent state
  - valid if if the majority of the nodes are honest

RANDOM NODE SELECTION



Hard to find solution



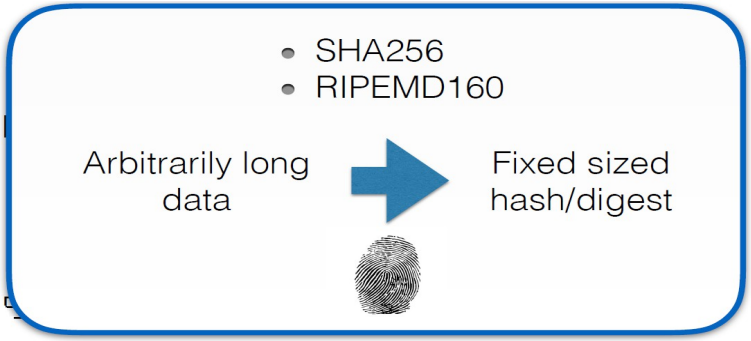
Easy to verify

- how to select a random node at each round?
- the key idea: select a node in proportion to a resource that it is hard to monopolize
- in Bitcoin this resource is computational power and selection is done on the basis of the Proof of work
- nodes which try to solve the proof of work are called miners and the whole validation process is called mining

PROOF OF WORK (POW)

- a mechanism that allows a party to prove to another party that a certain amount of computational resources has been utilized for a period of time.
- based on cryptographic puzzles that
  - can be solved
  - require a considerable effort which cannot be short-circuited
- verification of PoW requires less time with respect to the time needed to conduct the PoW
- the only tool needed: cryptographic hash functions

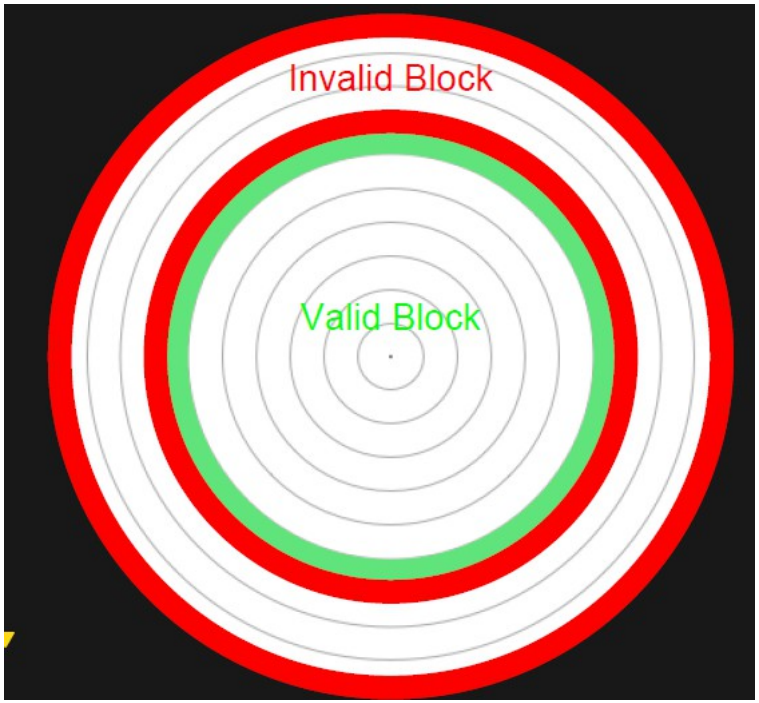
$$\mathcal{H} : \{0,1\}^* \rightarrow \{0,1\}^\lambda$$



PROOF OF WORK: A METAPHOR

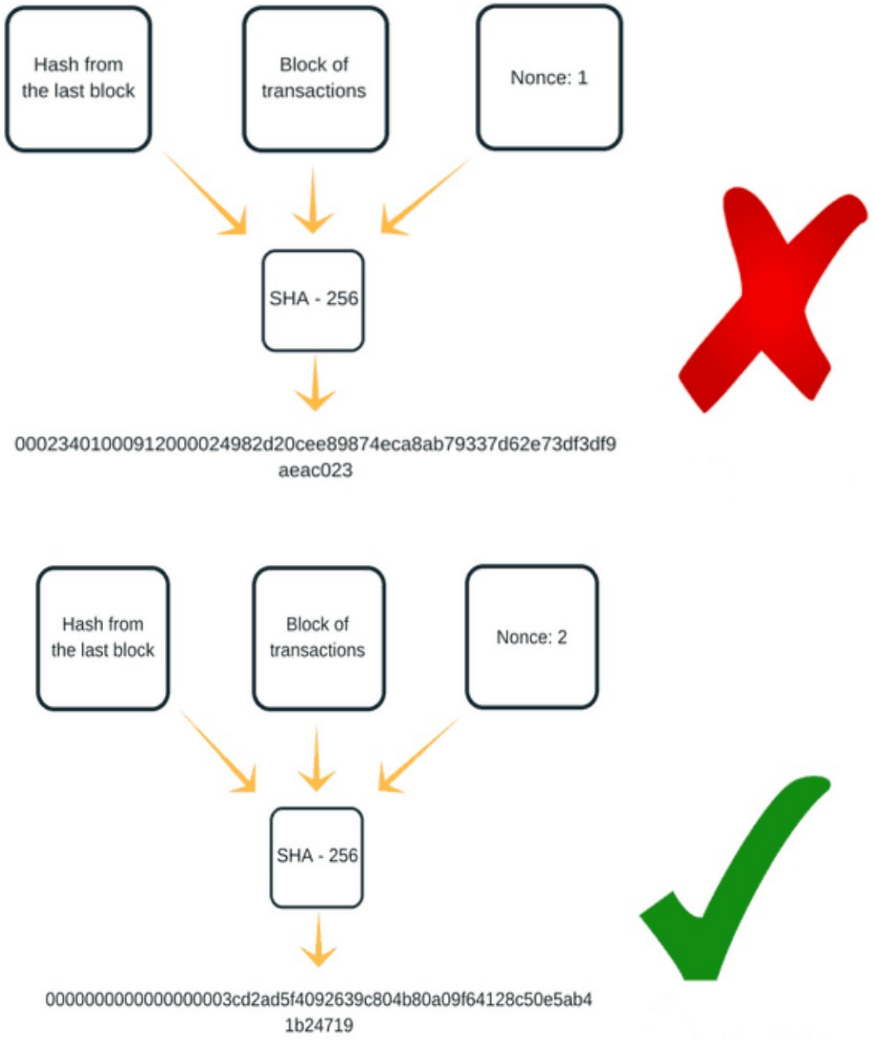
is like "throwing darts at a target while blindfolded"

- target: within the green ring
- equal likelihood of hitting any ring
- faster throwers: more hits per seconds
- difficulty: inversional proportional to green ring size
  - it can be tuned to obtain the desired difficulty.
  - if people get better at throwing darts, green circle needs to get smaller



POW IN BITCOIN

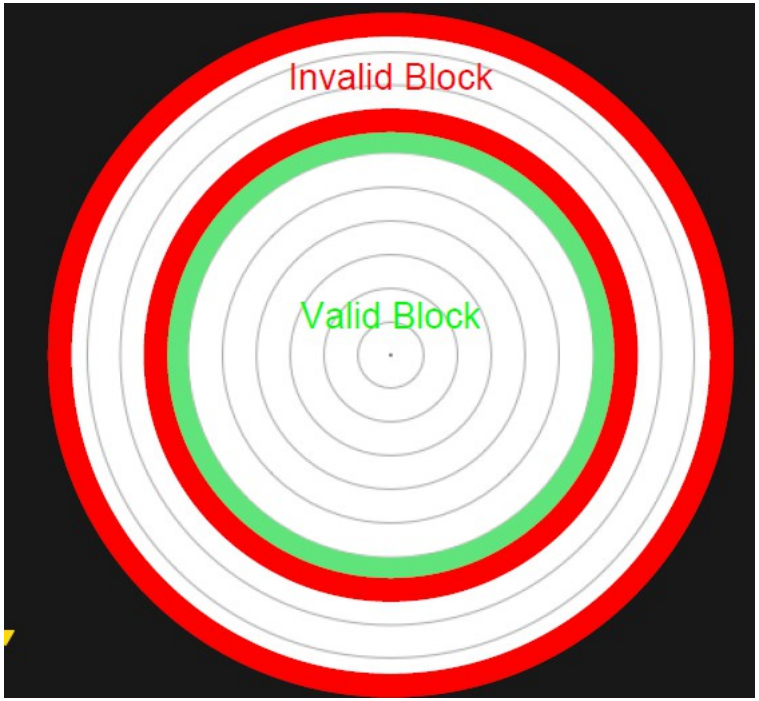
- concatenate the hash of the previous block, the content of the block and a nounce
  - nonce: a number which is used only once
- hash the resulting string
- if the output of the hashing function is a string starting with x zeros, the PoW is done
- X allows to tune the difficulty of the PoW



PROOF OF WORK: DIFFICULTY

- adjusts target depending on average time employed to produce valid results

- tune the computational effort
- increase or decrease the number of leading zeros required in the hash prefix.
  - double the computational effort, on the average: add a zero
  - reduce the computational effort of one half, on the average: remove one leading zero



PROOF OF WORK: CHARACTERISTICS

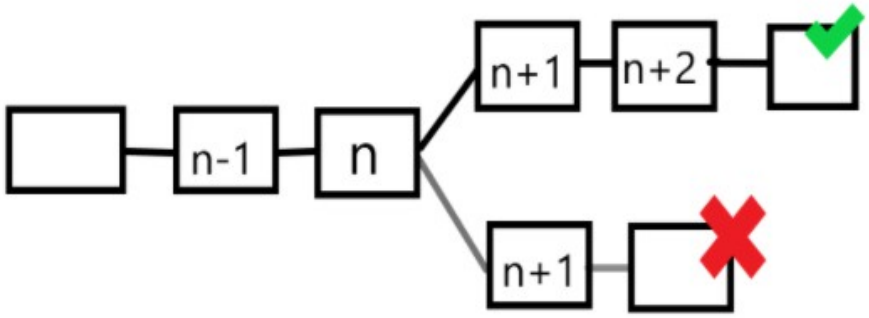
- what makes proof of work hard to solve?
  - output: like a a random 256-bit string where each bit is equally likely to be 0 or 1 independently of the other bits
  - each output bit looks like coin flips (0/1)
- no better way of finding the correct output than trying by brute force
- the probability p that the block hash falls below the target threshold T

$$p = \frac{T+1}{2^{256}}$$

- and the average number of trials required to find a block hash below the threshold is 1/p
- for instance, looking to the nBits field of the transactions on January 1st 2017, the average number of trails was about 2^{70}

FORKS IN LOTTERY BASED CONSENSUS

- two nodes win the lottery at the same time
  - a temporary fork occurs in the blockchain
  - each branch may grow independently



- Bitcoin consensus rules
  - the longest chain wins, the shorter is abandoned
  - a transaction may be inserted in a block that falls in the pruned branch
  - 6 confirmation rule:
    - a transaction is approved only when there are at least 5 following blocks in the blockchain

BITCOIN CONSENSUS: OTHER ISSUES

- lottery winner are rewarded
  - coinbase transactions
- rewarding correspond to minting bitcoin
- rewards are halved every four year
  - non inflationary approach
- Proof of work difficulty is tuned according to the mining power of the network
  - an example of self-organization of the system

BYZANTINE CONSENSUS

- "Byzantine generals": a well known problem already "active" from the 80's
  - n Byzantine generals surrounding a city
  - they must decide whether to attack or retret
  - generals may be traitors
  - nobody knows which generals are traitors

- find an algorithm to reach an agreement (consensus) such that
  - all the loyal generals decide on the same plan of action
  - a small number of traitors cannot cause the loyal generals to adopt different plans

PRELIMINARY NOTIONS

- computation
  - processes deterministic, vs probabilistic behaviour
    - probabilistic use random oracles
- interaction: processes interact through messages, and interaction is used to
  - communicate information to be shared
  - coordinate and synchronize the activities of the processed
- failures: several kind of failures may occur in the system
  - benign vs malicious (Byzantine)
  - failure in process vs communication
- time
  - determine whether we can make any assumptions on time bounds and relative computation speed

PRELIMINARY NOTIONS: COMPUTATION

- processes
  - unit of computation in a distributed system
- typical assumptions
  i. the set is static, and the number of processes is known
  ii. process know each other
  iii. all processes run a copy of the same algorithm: the sum of all these copies constitues the distributed algorithm
- assumption i. and ii. may be reasonable for a permissioned blockchain, not for a permissioned one
  - in a permissionless blockchain the set of peers is dynamic
  - they are too many and too dynamic to know them all
  - multiple algorithms

PRELIMINARY NOTIONS: PROCESS FAILURES

- both processes and communication channels may fail, depart from what is considered its correct behaviour
- benign failures
  - fail-stop a process stops executing events and other processes may detect this fact
  - crash a process stops executing events
- malicious failures: arbitrary failures, or byzantine
  - malicious attacks
    - processes may lie
    - processes may collude: malicious behaviours inspired by an intelligent attack
  - software errors
    - arbitrary states, arbitrary messages

PRELIMINARY NOTIONS: PROCESS FAILURES

- some of byzantine behaviours examples previous to blockchain
  - Amazon outage (2008)

    https://status.aws.amazon.com/s3-20080720.html
  - Shuttle Mission STS-124 (2008)

    https://c3.nasa.gov/dashlink/resources/624/
- previous are examples of software errors producing byzantine behavious
- several protocols proposed to reach consensus in presence of byzantine failures
  - several of them are designed to work correctly if the number of failures is bounded
  - for instance more than 2/3 are correct processes

PRELIMINARY NOTIONS: COMMUNICATION FAILURES

- bening communication failures
  - messages are omitted
  - messages lost from the network, problem storing the message in a buffer and so on
- malign communication failures
  - duplicated messages, created out of nothing,...
  - encryption techniques may help in this case
- minimal assumption:
  - the channels cannot systemitacally drop a specific language
  - UDP you send an infinite amount of messages

PRELIMINARY NOTIONS: TYPE OF SYSTEMS

- in distributed system
  - it is difficult to reason about time
  - lack of clock synchronization
  - difficult to pose time bounds on events and communication
- different models, according to the assumption about the time
  - asynchronous distributed systems
  - synchronous distributed systems
  - partially synchronous distributed systems.

ASYNCHRONOUS VS SYNCHRONOUS SYSTEMS

- asynchronous distributed system: the worst scenario. No bounds on:
  - the relative speed of process execution
  - message delays
  - clock drift
- unfortunately, a realistic scenario
  - several sources of asynchrony are present in large-scale network, like Internet
  - the worst possible model: lack of assumptions
- synchronous distributed systems: known upper bound
  - on the relative speed of process execution
  - on message trasmission delays
  - on clock drift
  - is not the Internet, can be built with specialized hardware

PARTIALLY SYNCHRONOUS SYSTEMS

- for most system it is relatively easy to define physical time bounds that are respected most of the times.
- there are however periods where the timing assumptions do not hold
- delays on processes
  - machine may run out of memory, slowing down processes
  - a typical case of "no bound on relative speeds of processes"
- delays on processes
  - network may be congested, and message may be dropped
  - re-trasmission protocols can ensure reliability, but at the price of asynchrony
  - messages may be re-trasmitted an arbitrary number of times

PARTIALLY SYNCHRONOUS SYSTEMS

- how to express partial asynchrony?
  - timing assumptions only hold eventually
  - the system is not always synchronous
  - there is no known bound to the period in which it is asynchronous
  - we expect that there are period during which the system is synchronous
  - some of these period are long enough to terminate the execution of the protocol

PROPERTY OF THE SYSTEM

- safety
  - something bad will never happen
  - a distributed protocol never enter an unacceptable state
- liveness
  - something good eventually does happen

THE MUDDY CHILDREN

- a problem showing you the kind of reasoning we will make in distributed consensus algorithms

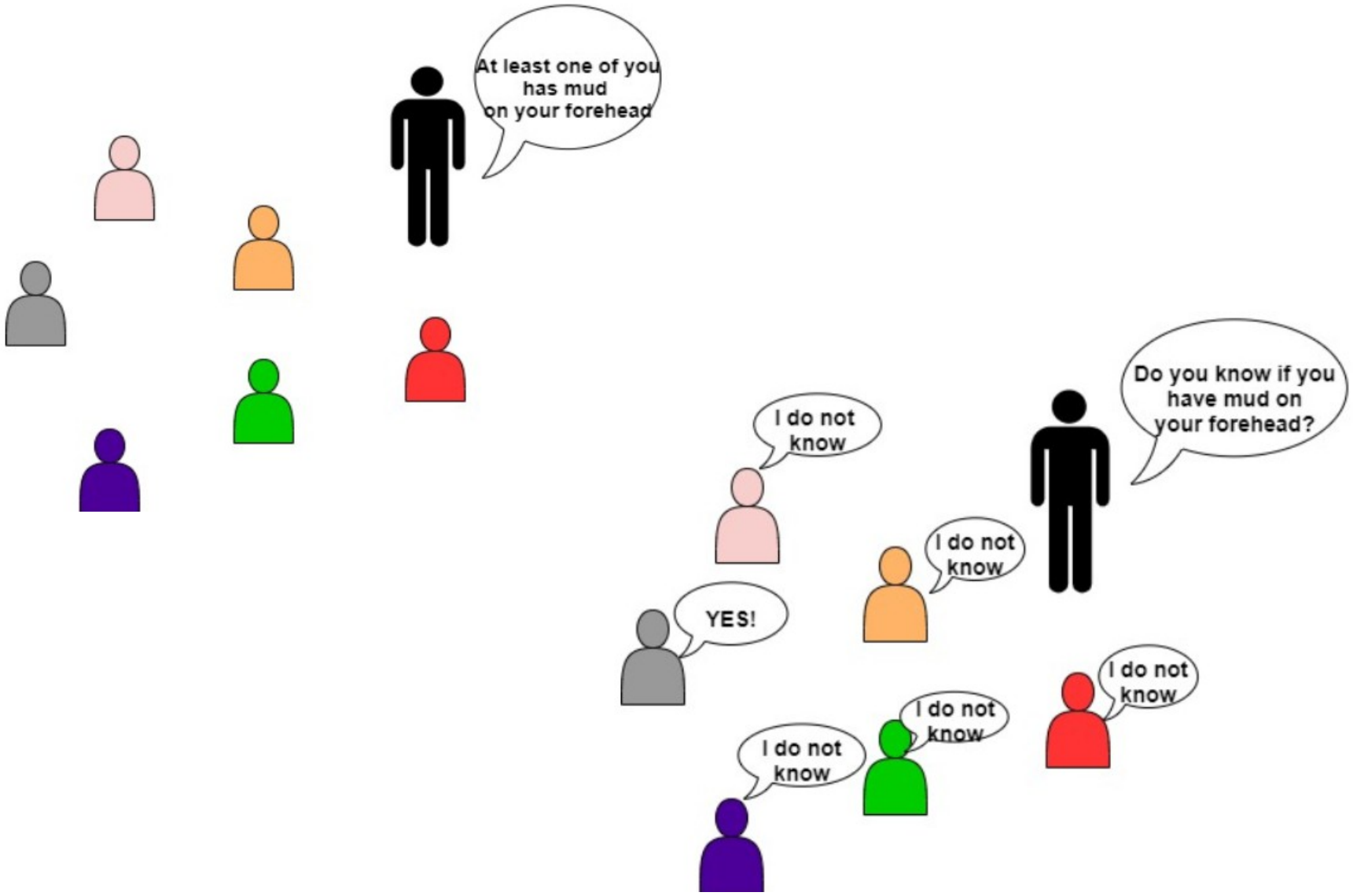
THE MUDDY CHILDREN PROBLEM

- The problem:
  - n children go playing
  - children are truthful, perceptive and intelligent
  - mom says: “Don't get muddy!”
  - a bunch (say, k) get mud on their forehead
  - daddy comes, looks around, and says “Some of you got a muddy forehead”
  - daddy repatedly ask: “Do you know whether you have a muddy forehead?
- What happens?

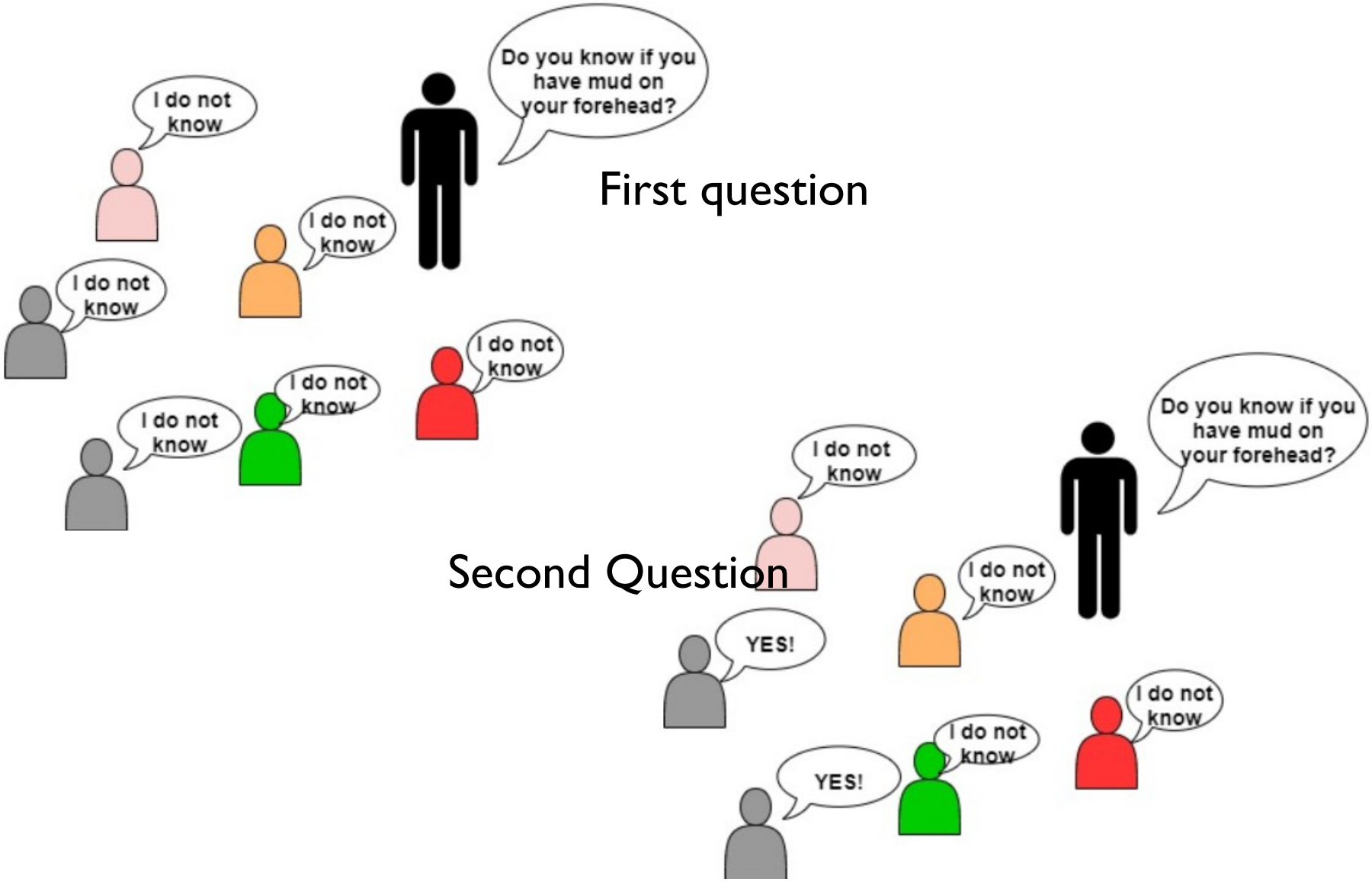
Claim

- if there are n children, k of whom are muddy, then the k children will announce that they are muddy after the father repeats his question k times.
- a single assumption: $n \geq k$
- It can be proved by induction

THE MUDDY CHILDREN PROBLEM, K=1



THE MUDDY CHILDREN PROBLEM, K=2



THE MUDDY CHILDREN PROBLEM, K=2

- suppose 2 children are muddy
- basic intuitions
- after the father announces "at least one of you has mud on your forehead"
  - the two muddy children consider possible either 1 or 2 muddy children in total
  - the clean children consider possible either 2 or 3 muddy children in total
- when the father asks for the first time "do you know if you have mud on your forehead?"
  - everyone answers "I do not know"
  - everyone knows that only a muddy child is not the possibility
  - the two muddy children know they are muddy

WHY MUDDY CHILDREN?

- an example of distributed system
  - agents: nodes, machines, people
  - they have some common knowledge on the state of the system
  - some of the knowledge is visual, you can see the state of another agent
- in a real system ask for the state of the other node
  - common knowledge: everybody has received the same information
  - everyone knows that everyone knows, and so on...
- many consensus algorithms we will present are based on the construction of a common knowledge among the nodes
- a famous paper,
  - J. Halpern, Y. Moses "*Knowledge and Common Knowledge in a Distributed Environment*". E.W. Djikstra Prize 2009.

THE BYZANTINE HISTORY

- State-of-art at the end of the 90's
  - theoretically feasible algorithms to tolerate Byzantine failures, very inefficient in practice, can be used only for small size systems
  - assume synchrony – known bounds for message delays and processing speed
  - synchrony assumption needed for correctness: what about DoS?
- the main result
  - L.Lamport, R. Shostak, M. Pease, *The Byzantine General problem*, ACM Transaction on Programming Language and Systems (TOPLAS), 4(3): 382-401, 1982.
  - the "oral messages" algorithm
  - we will start from this, whish is simpler with respect to algorithms for asynchronous system

THE BYZANTINE "REINASSANCE"

- M.Castro, B. Liskov, *Practical Byzantine fault tolerance and proactive recovery,* ACM Trans. Computer Systems, 20:398-461, November 2002
- contributions
  - first state machine replication protocol that survives Byzantine faults in asynchronous networks
  - live under weak assumptions
  - implementation of a byzantine, fault tolerant distributed file system
  - experiments measuting the cost of the replication techniques
- the real renaissance was in 2015
  - the algorithm has been used to define consensus protocol for many blockchain
  - *Hyperledger*, *Corda*, *Facebook Libra*, part of the consensus protocol of *Algorand*,...

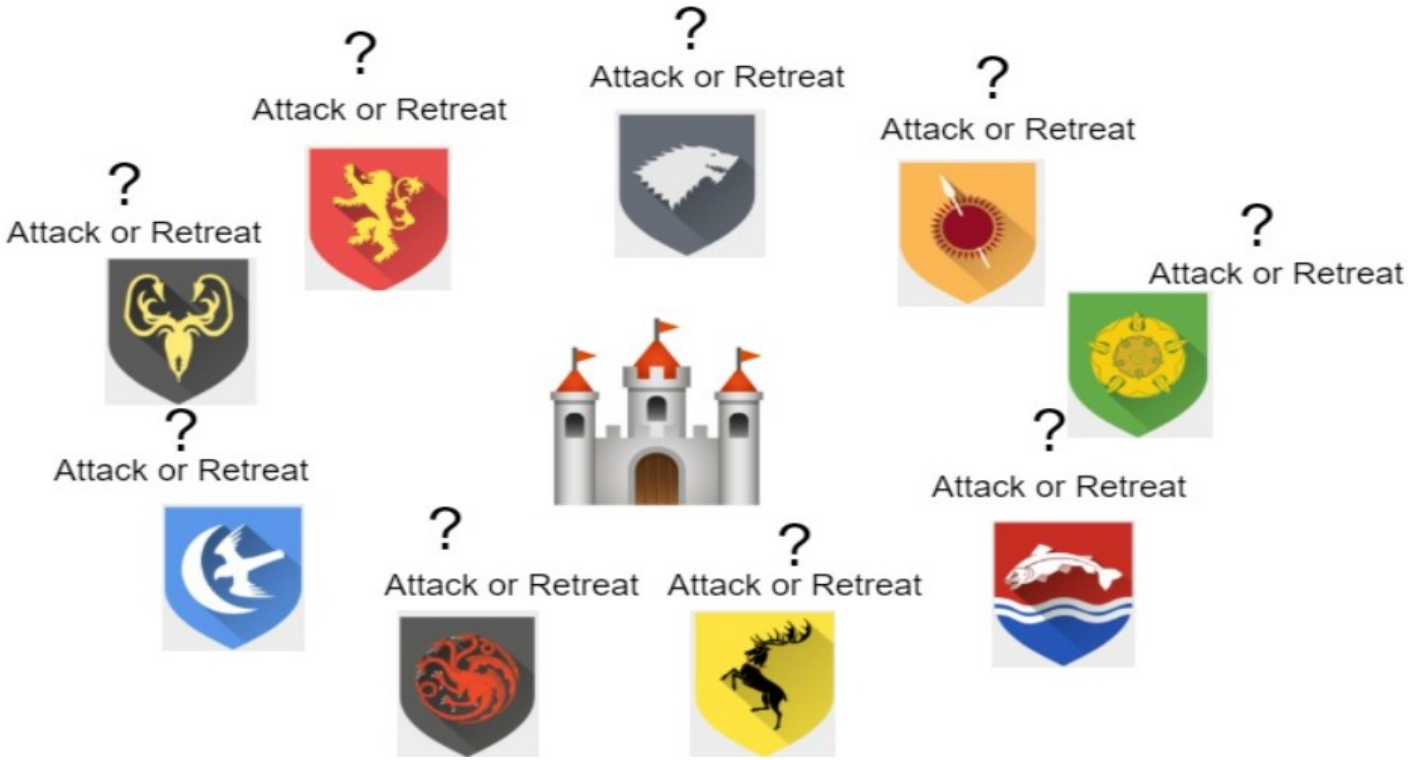
THE BYZANTINE GENERALS PROBLEM

- all the GOT houses (Lannister, Stark, ….) are allied for once to conquer a new kingdom and are camped outside the enemy castle
- each house has one commander (Tyrion, Jon Snow,...)

THE BYZANTINE GENERALS PROBLEM

- generals wish to organize a plan of action to attack or to retreat
- each general observer the enemy and communicates his decision to the other generals
- all loyal generals must decide a common plan, i.e. reach an agreement or consensus

THE BYZANTINE GENERALS PROBLEM

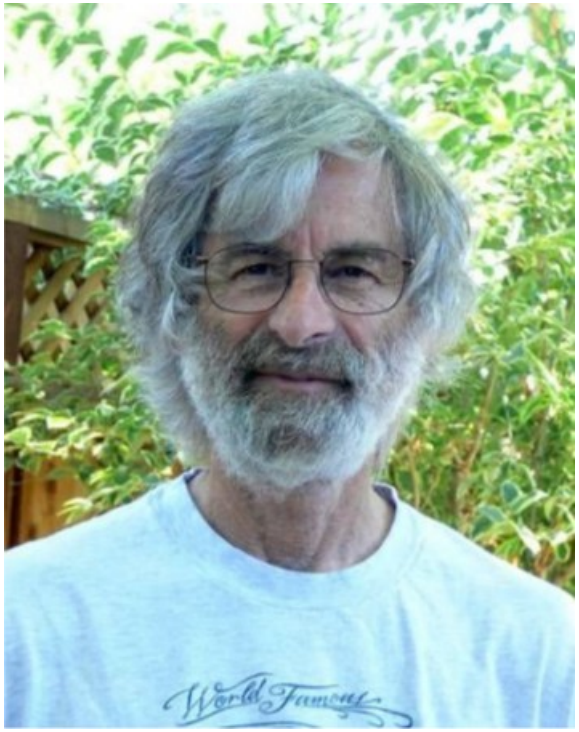


- unfortunately, there some generals are traitors (probably the Lannister...)
- traitors want to influence the plan to the enemy's advantage: they may lie
- may lie about whether they will support a particular plan, giving different information to different generals
- may lie on what other generals told them

BYZANTINE IN LESLIE LAMPORT'S VIEW

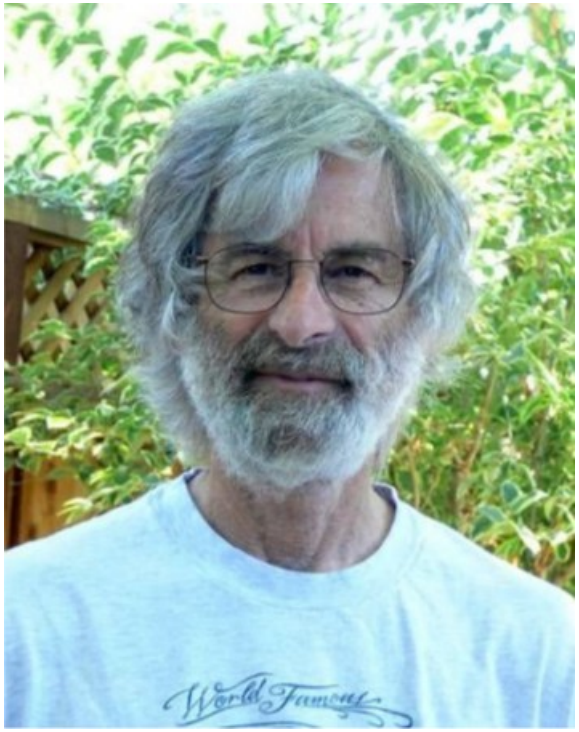
- "Byzantine Generals" metaphor introduced in the classical paper by Lamport [Lamport et al., 1982]
- Leslie Lamport, Robert Shostak, Marshall Pease, *The Byzantine Generals Problem*

  ACM Transactions on Programming Languages and Systems, vol. 4, n. 3, luglio 1982, pp. 382-401
- to the right, Leslie Lamport



BYZANTINE IN LESLIE LAMPORT'S VIEW

- "I have long felt that, because it was posed as a cute problem about philosophers seated around a table, Dijkstra's dining philosopher's problem received much more attention than it deserves."
- "the popularity of the dining philosophers problem taught me that the best way to attract attention to a problem is to present it in terms of a story"
- "there is a problem in distributed computing that is sometimes called the Chinese Generals Problem, in which two generals have to come to a common agreement on whether to attack or retreat, but cancommunicate only by sending messengers who might never arrive."
- "I stole the idea of the generals and posed the problem in terms of a group of generals, some of whom may be traitors, who have to reach a common decision.



THE BYZANTINE GENERALS PROBLEM

Consensus

A) all the loyal generals decide upon the same plan of action

- plan of action maybe attack or retreat
- this does not matter, as long as all adopt the same plan

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

- a few traitors can affect the decision only if the loyal general are almost equally divides between the two possibilities, in this case neither decision is bad.
- but they cannot force generals to adopt different plan of actions

which algorithm for decision making should the generals use to reach consensus?

what percentage of liers can the algorithm tolerate and still correcly determine consensus?

A NAIVE IMPLEMENTATION

- let
  - n be the number of generals
  - v(i) the opinion of general i (attack/retreat)
  - each general i communicates the value v(i) by messangers to each other general j
  - each general final decision is obtained by the majority function of the values v(1),...,v(n)
- But this does not work!
  - n=10, 2 traitors, 7 loyal generals, traitors send different messages to loyal
  - 4 loyal generals choose retreat (R), 3 choose attack(A)
  - general 1 (loyal) receives RRRRAAA from the loyal generals, RA from the traitors
  - general 2 (loyal) receives RRRRAAA from the loyal generals, AA from the traitors
  - majority is different for the two generals , even if they are both loyal

A NAIVE IMPLEMENTATION

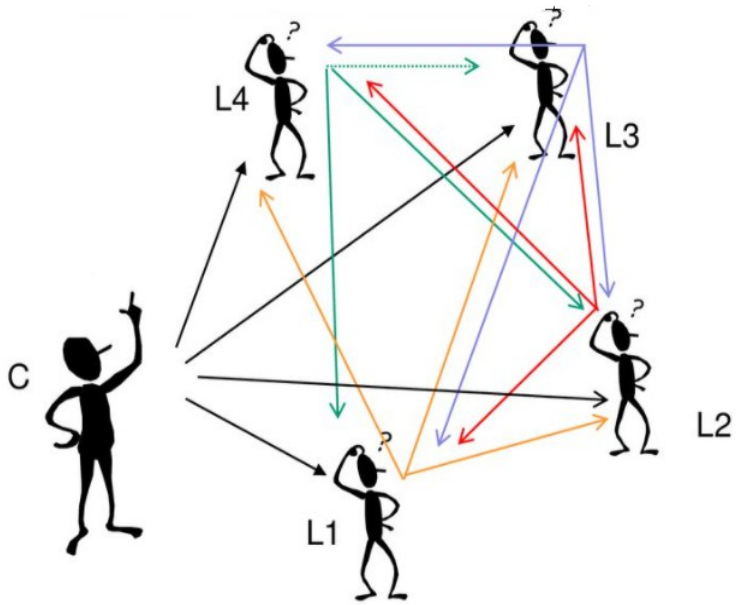
- why the naive algorithm does not work?
- to satisfy condition A) every general must apply the majority function to the same values v(1),...,v(n)
  - but a traitor may send different values to different generals
  - a general need to use a value of v(i) different with respect to that directly received from the i-th general
- to satisfy condition B)
  - a small number of traitors does not "interfere" with the majority computation: they do not disrupt the consensus
  - for each i, if the i-th general is loyal, then the value v(i) it sends must be used (and not changed) by every general to compute majority

PROBLEM SPECIFICATION

- let us consider the consensus problem into a simpler situation in which we have
  - 1 commander general (C)
  - n-1 lieutenant generals (L1,..,Ln-1)
- consensus: Interactive consistency conditions
  - IC1:
    - all loyal lieutenants obey the same order
  - IC2:
    - the decision of loyal lietenants must agree with the commanding general's order if he is loyal
- if the commander is loyal IC1 follows from IC2

THE "ORAL MESSAGES" ALGORITHM: GENERAL IDEA

- assume only the general is faulty. Can send different messages to different liutenats
- liutenenat generals send message back and forth among themeselves reporting the command received by the commanding general
- each liutenent computes the majority of the messages
- same decision at each liutenant



L1: (v1, v2, v3, v4) majority(v1, v2, v3, v4)

L2:(v1, v2, v3, v4) majority(v1, v2, v3, v4)

L3: (v1, v2, v3, v4) majority(v1, v2, v3, v4)

L4: (v1, v2, v3, v4) majority(v1, v2, v3, v4)

THE "ORAL MESSAGES" ALGORITHM: ASSUMPTIONS

- every message that is sent by a non faulty node is received correctly
  - reliability
- only the receiver of a message knows who sent it
  - symmetric encryption
    - if A sends a message to B, this message is signed with an encryption key which is shared only between A and B
    - used to identify which has sent the message
    - the message received from B cannot be forwarded to another node C, to prove that A sent this message, because A and C share a different key
- the absence of a message can be detected
  - the system is synchronous

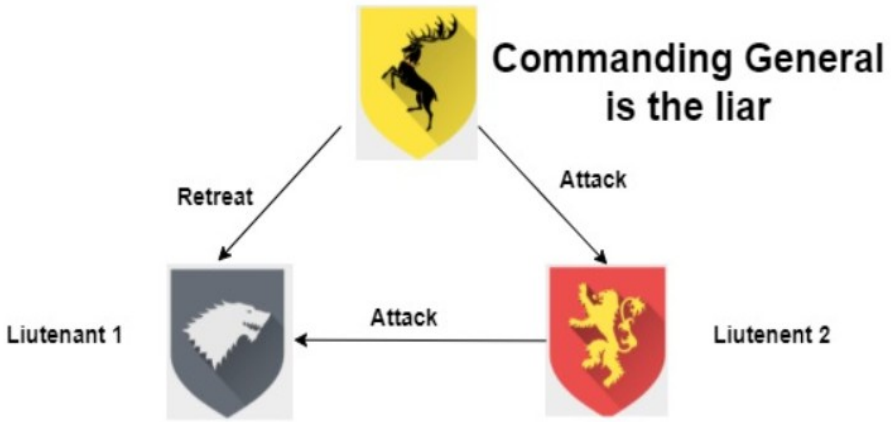
THE "ORAL MESSAGES" ALGORITHM: ASSUMPTIONS

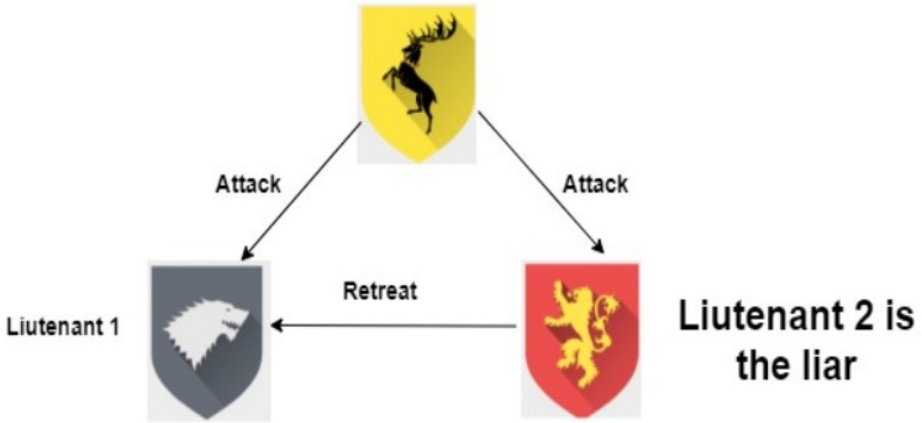
further assumptions:

- a traitor commander may decide not to send any order, in that case we assume a default order equal to "retreat"
- *majority (v1,...vn-1)* returns "retrait" if there is not a majority among nodes

IMPOSSIBILITY RESULT

- under the "Oral Message" Assumption, no solution exits with n=3 even with a single traitor

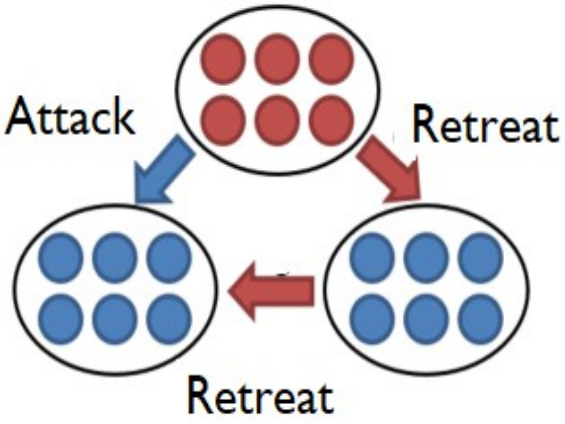


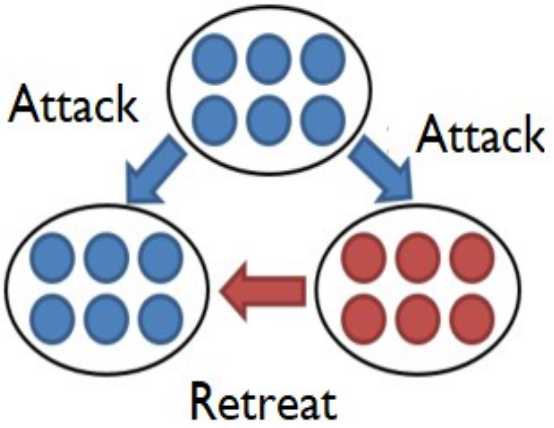


- majority is equal "retreat" in both cases.
- correct if the commanding is a liar, non correct if liutenant 2 is the liar
- impossibility to distinguish between the two scenarios

IMPOSSIBILITY RESULT

- can be generalized





- No solution with fewer than 3m+1 generals can cope with m traitors

THE O(M) ALGORITHM

- proposed in 1982 by Lamport, Pease, and Shostak
- n nodes
- m traitors
- the algorithm computes a correct agreement fulfilling condition IC1 and IC2 if $n > 3m$.
- in the paper: a succint and short, recursive
  - not an obvious definition
- we will give an intuitive presentation of the algorithm

"ORAL MESSAGES" ALGORITHM: OM(M)

- two stages
  - data gathering: m+1 rounds of messaging between the processors
  - majority computation
- data gathering
  - round 0: the general sends the order to all its lieutenants
  - round1 : each liutenant broadcasts the message it has received to all other liutenants, excluding the general (see previous slides)

    ……..
  - round i: each liutenant receives a message that is a pair (C, path)
  - a path $<ID_1, ID_2,...ID_k>$ means thet P_{IDk} was told that P_{IDk-1} was told that P_{IDk-2} was told ...that P_{ID1} was told by the commander that the command was C

"ORAL MESSAGES" ALGORITHM AS IN THE PAPER

Algorithm OM(0)

- commander C sends its value to very liutenant Li, i $\in$ {1,...n-1}
- each liutenant Li uses the received value, or the value retreat if no value is recieved

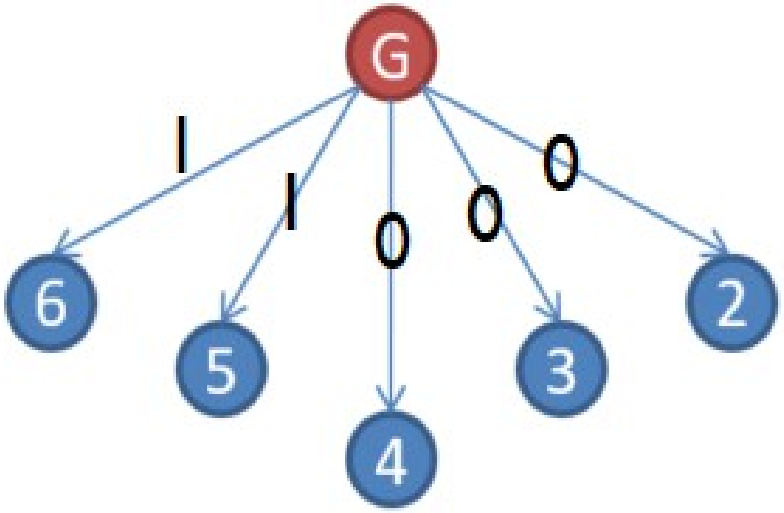
"ORAL MESSAGES" ALGORITHM AS IN THE PAPER

Algorithm OM(m), m>=0

- commander C sends its value to very liutenant Li, i $\in${1,...n-1}
- 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 liutenants
- 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).

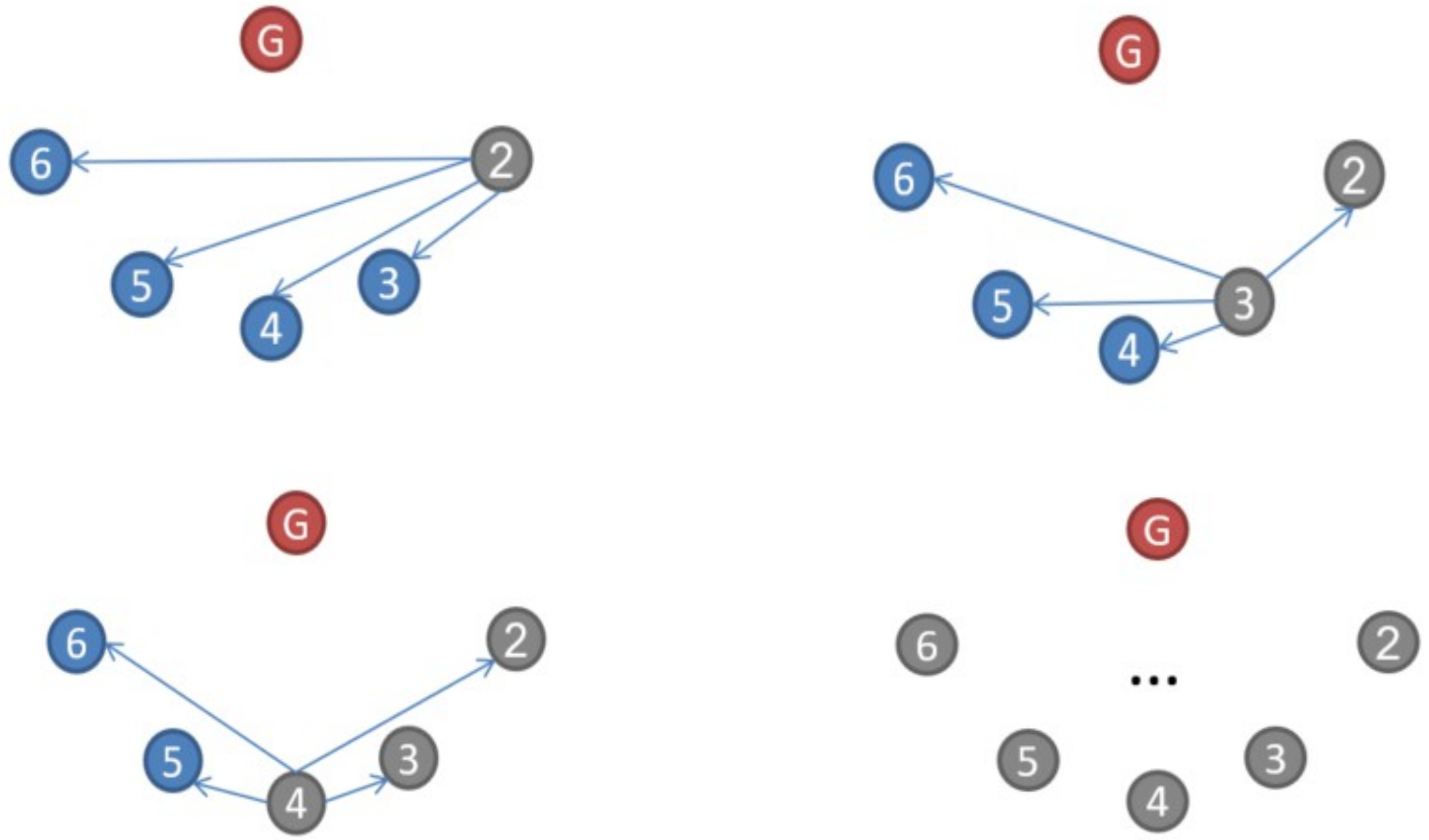
  Li uses the value of the majority (v1m..., vn-1).

"ORAL MESSAGES" ALGORITHM: OM(M)



- one faulty general: sends byzantine messages to the other peers
- m=1 (faulty general), n=6, n> 3*m, the nodes can reach a consensus
- first step of the algorithm: the faulty general sends a message to all the other nodes (liutenants)
  - it is faulty sends different messages to different peers

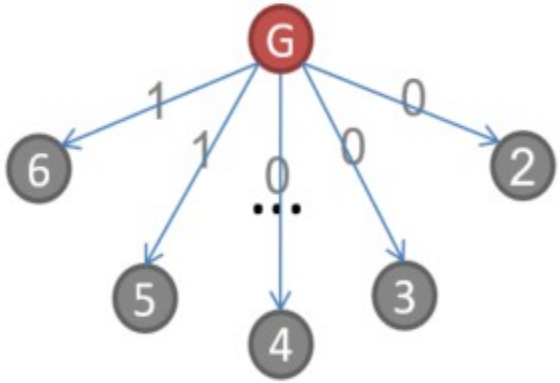
"ORAL MESSAGES" ALGORITHM: OM(M)



- each liutenant sends the message it has received to all other liutenant

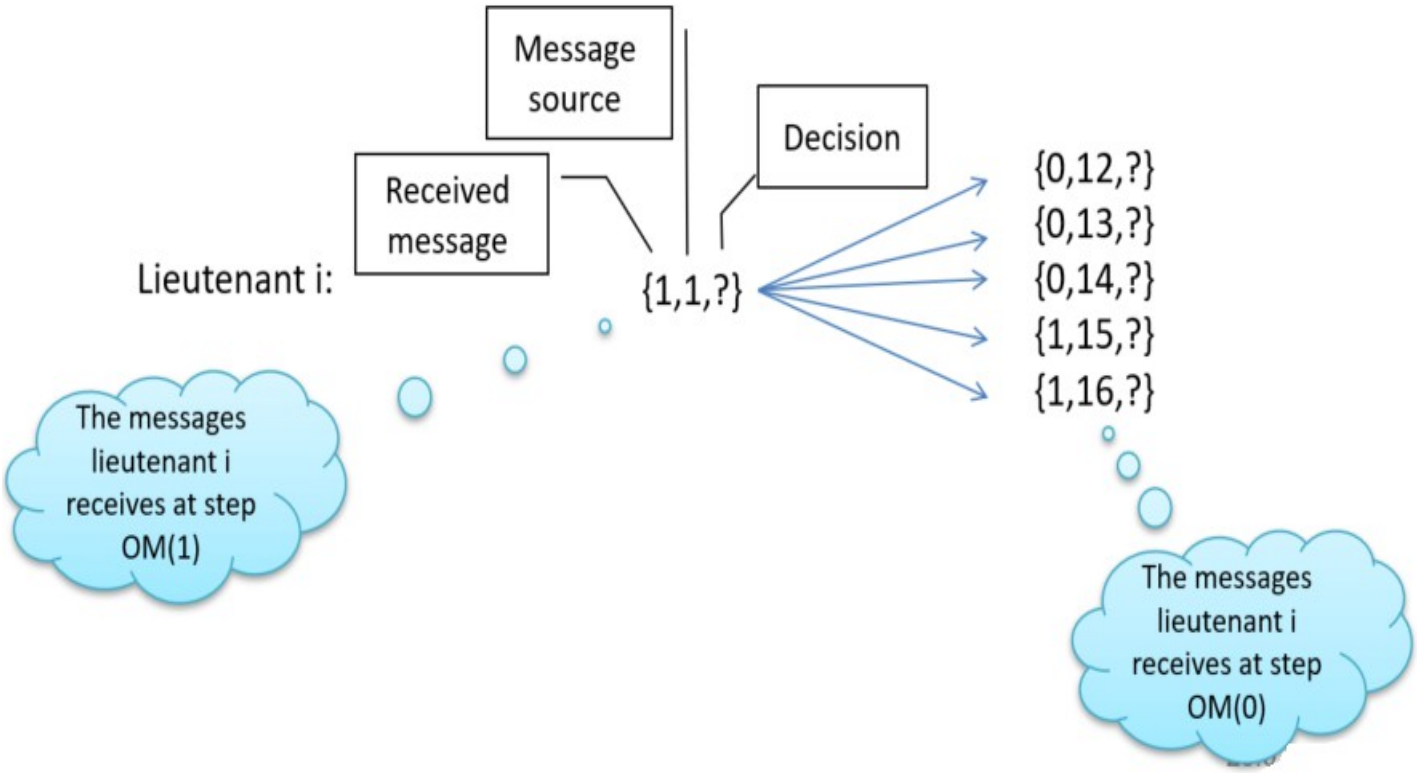
"ORAL MESSAGES" ALGORITHM: OM(M)

Sender=P_2		Sender=P_3		Sender=P_4		Sender=P_5		Sender=P_6	
Dest	Msg	Dest	Msg	Dest	Msg	Dest	Msg	Dest	Msg
P_2	{0,12}	P_2	{0,13}	P_2	{0,14}	P_2	{1,15}	P_2	{1,16}
P_3	{0,12}	P_3	{0,13}	P_3	{0,14}	P_3	{1,15}	P_3	{1,16}
P_4	{0,12}	P_4	{0,13}	P_4	{0,14}	P_4	{1,15}	P_4	{1,16}
P_5	{0,12}	P_5	{0,13}	P_5	{0,14}	P_5	{1,15}	P_5	{1,16}
P_6	{0,12}	P_6	{0,13}	P_6	{0,14}	P_6	{1,15}	P_6	{1,16}



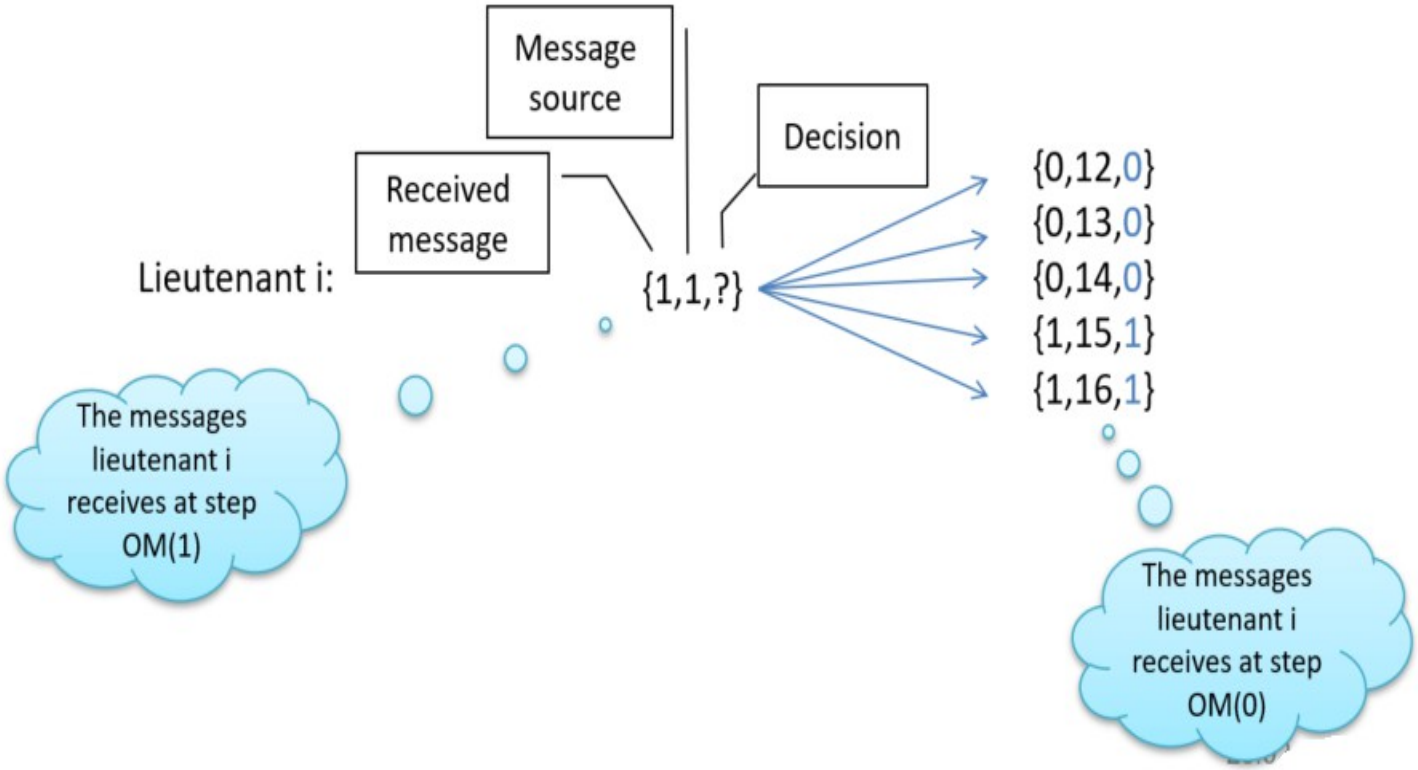
- we can organize the received messages in a table
  - each column reports the messages sent from each liutenant to all the others: for each message, the value and the path that the information has transversed
  - each row reports the messages received from a liutenant (node)

"ORAL MESSAGES" ALGORITHM: OM(M)



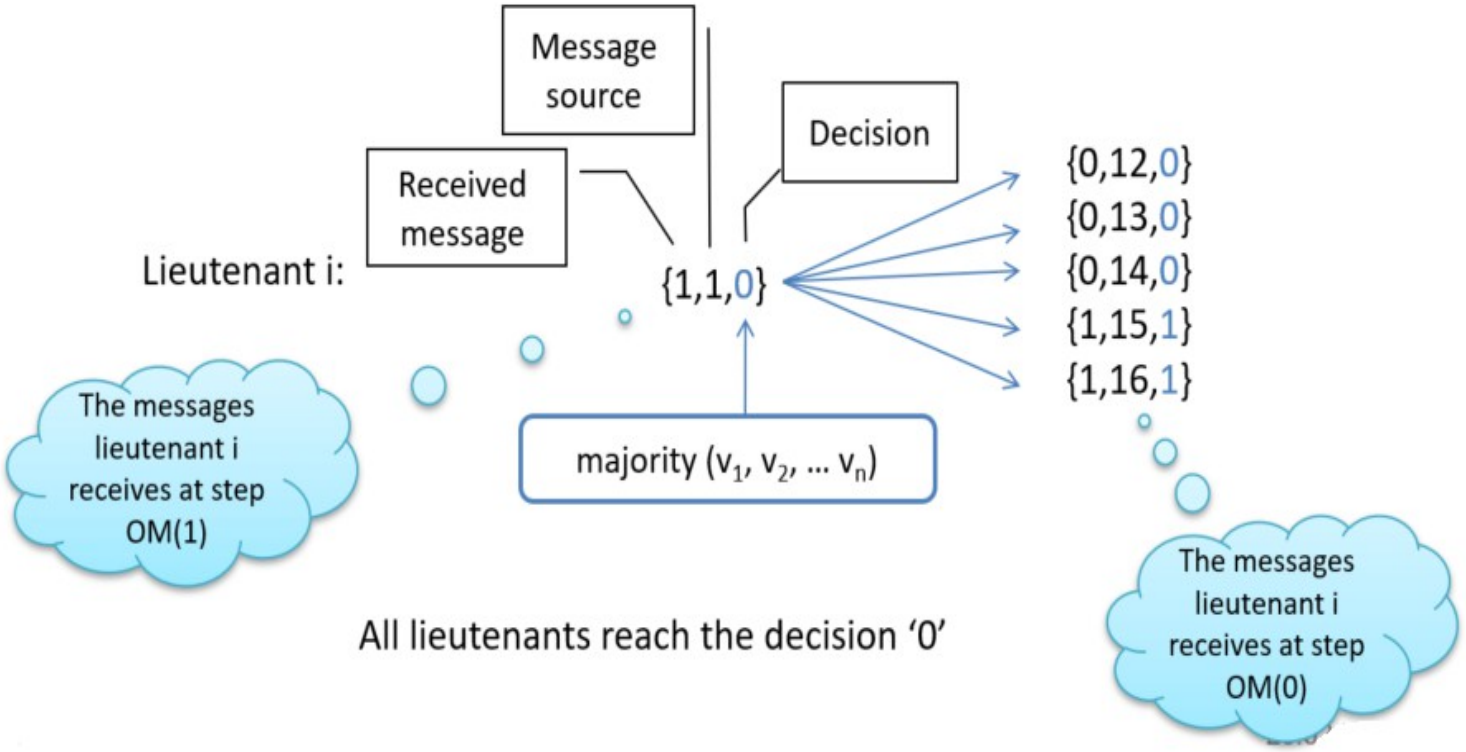
- you can build a recursion tree for the decision of each liutenant

"ORAL MESSAGES" ALGORITHM: OM(M)



- you can build a recursion tree for the decision of each liutenant

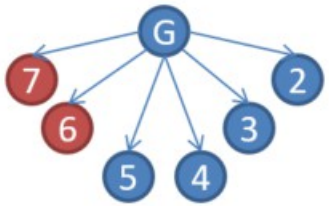
"ORAL MESSAGES" ALGORITHM: OM(M)

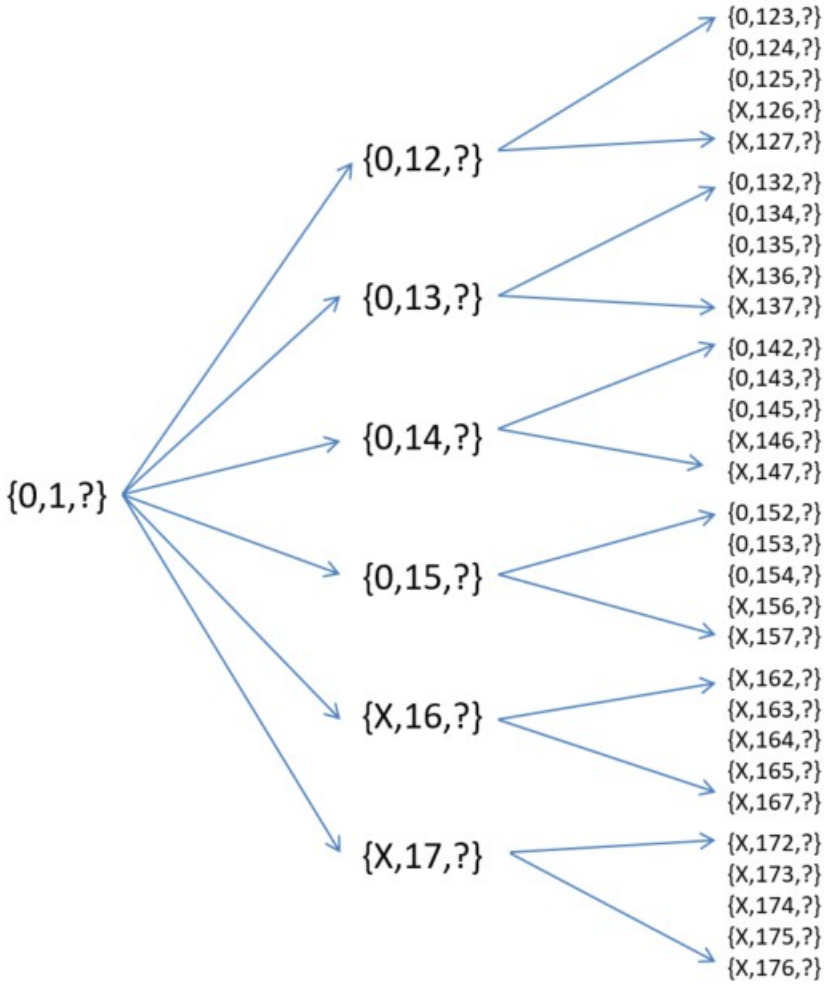


- you can build a recursion tree for the decision of each liutenant

A MORE COMPLEX EXAMPLE

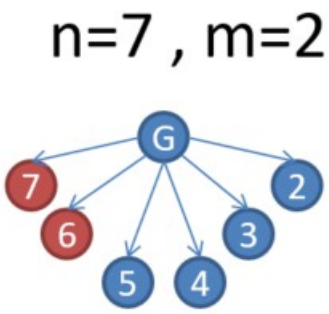
n=7 , m=2

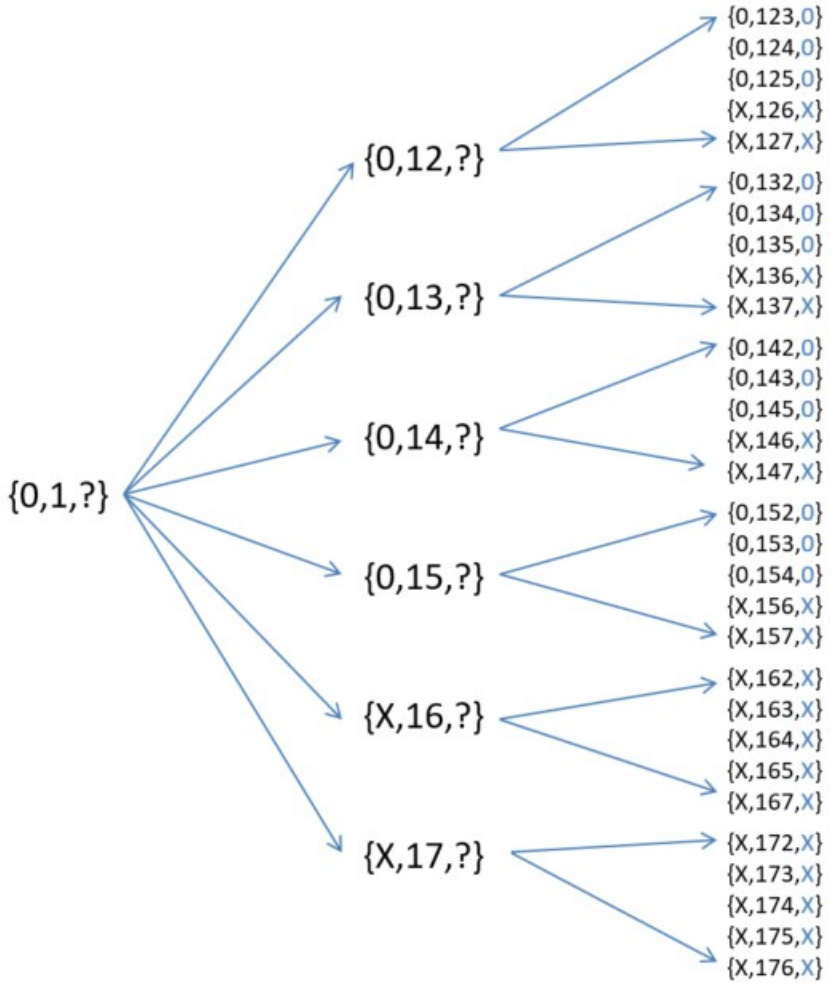




- you can build a recursion tree for the decision of each liutenant

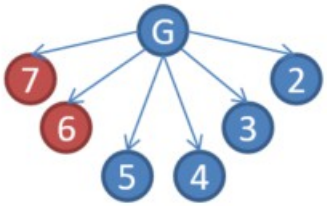
A MORE COMPLEX EXAMPLE

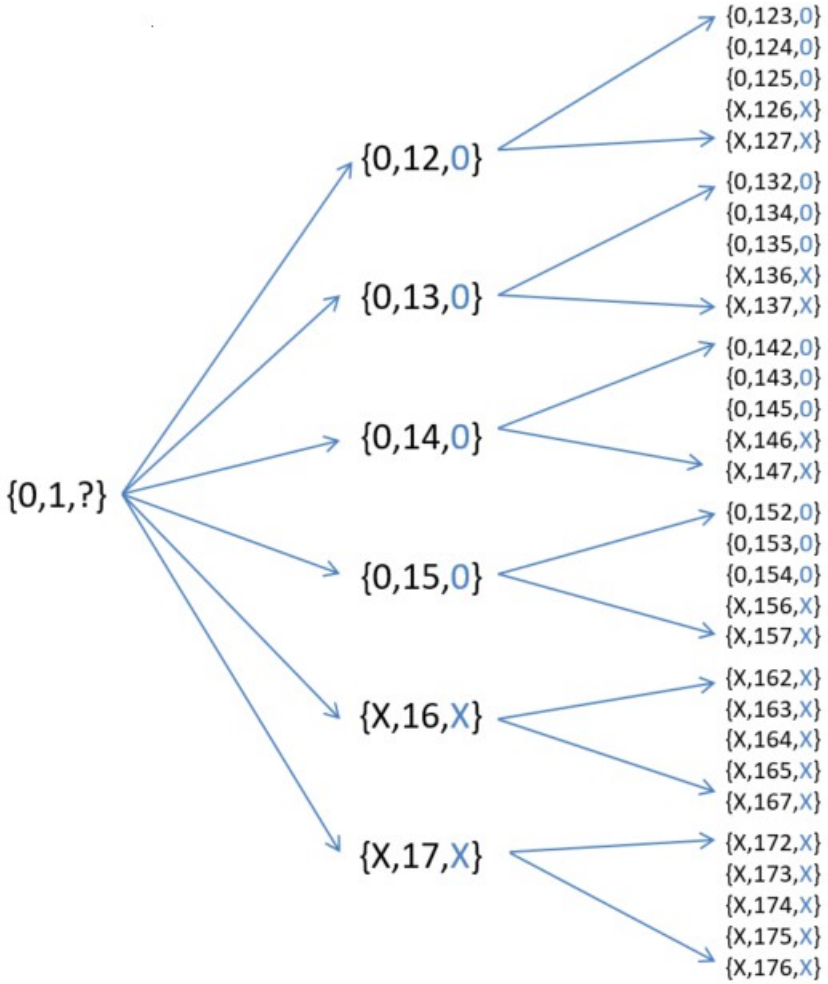




- you can build a recursion tree for the decision of each liutenant

A MORE COMPLEX EXAMPLE

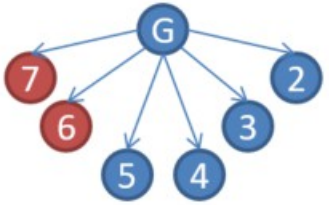


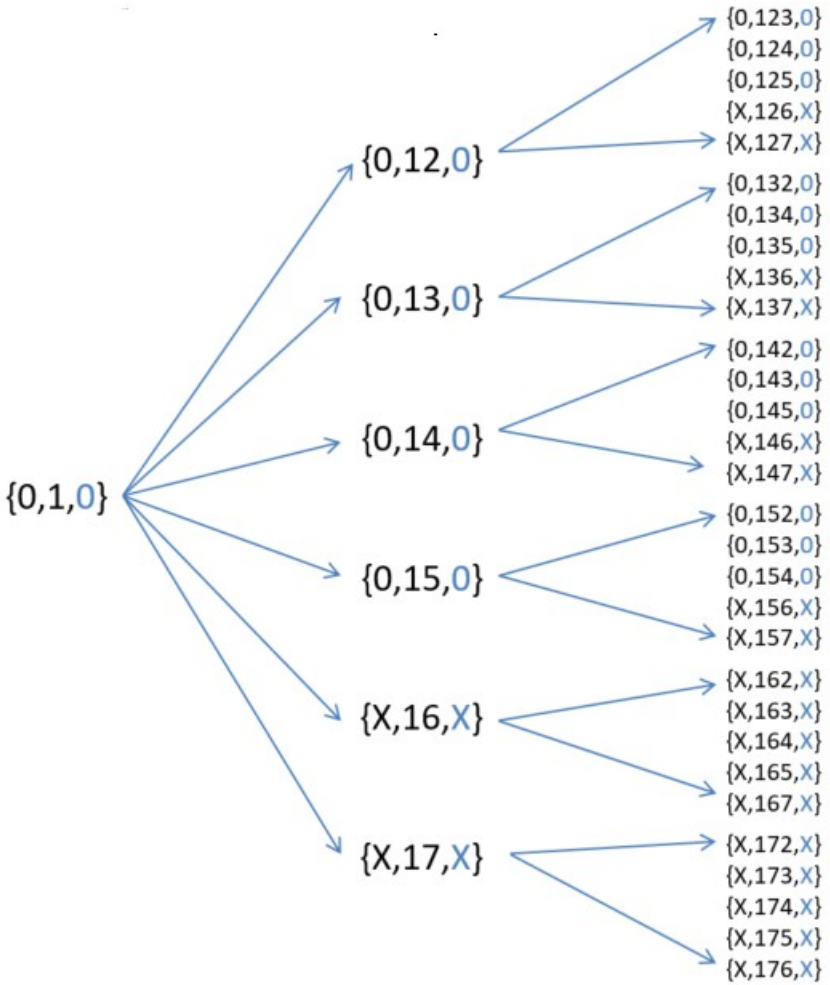


- you can build a recursion tree for the decision of each liutenant

A MORE COMPLEX EXAMPLE

n=7 , m=2





13

- you can build a recursion tree for the decision of each liutenant

DISTRIBUTED KNOWLEDGE THROUGH ORAL MESSAGES

- the system build a distributed knowledge
- everynode knows that every other node...and so on.
  - the reasoning is similar to that of the muddy children
- information is exchanged directly between nodes the nodes
- if Liutenent i receives information about Commanding General through Liutenant j, Liutenent i can only conclude that
  - Liutenant j has said thatthe commanding general had said,...and so on
  - more complex of the muddy system because of faulty nodes

"ORAL MESSAGES" ALGORITHM: FORMAL RESULTS

- Theorem: necessary condition

  for any m >1, no soultion with fewer than 3m +1 generals can cope with m traitors

- Theorem: correctness

  for any m, Algorithm OM(m) satisfies conditions IC1 and IC2 if there are 3m+1 generals or more and at most m traitors

"ORAL MESSAGES" ALGORITHM: PROBLEMS

- message paths of length m +1 (expensive)
- absence of messages msut be detected via time-outs (vulnerable to DoS)
- an attacker may compromise the safety of a service by delaying non-faulty nodes or the communication between them until they are tagged as faulty and excluded from the replica group
  - such a denial-of-service attack is generally easier than gaining control over a non-faulty node.
  - synchrony essential for correctness
    - what happens in DoS attacks scenarios?

"SIGNED MESSAGES" ALGORITHM

- use asymmetric cryptography
- a loyal general's signature cannot be forged, and any alteration of the contents of his signed messages can be detected
- anyone can verify the authenticity of a generl's signature
- algorithm SM(m)
  - For any m, SM(m) solves the byzantine Generals Problem if there are at most m traitors