Blockchain

… & cryptocurrencies & IoT

References

- “From blockchain consensus back to Byzantine consensus”, V. Gramoli, Future Generation Computer Systems 107, 2020
- “Blockchains and Smart Contracts for the Internet of Things”, K. Christidis, M. Devetsikiotis, IEEE ACCESS 2016
- Ethereum guide: https://ethereum.org/it/learn/
- Ethereum yellow paper, Gavin Wood, 2020, https://ethereum.github.io/yellowpaper/paper.pdf
- Bitcoin developers guide in http://www.bitcoin.org
- “An Overview of Blockchain Technology: Architecture, Consensus, and Future Trends”, Z. Zheng et Al., IEEE 6th International Congress on Big Data, 2017

Introduction

Distributed Ledger Technology (DLT)

- The Distributed Ledger Technology (DLT) is a database over a set of peer-to-peer (P2P) nodes (computers), where each node keeps an entire replica of the database
  - each update to the DB must be applied to all nodes
  - nodes need to reach a consensus on the update
- pros: no centralized control, no single point of failure
- cons: extra complexity to manage the redundancy and the consensus

Blockchain and DLT

- blockchain is an example of implementation of a DLT
- in a blockchain the DLT is composed by a sequence of blocks chained together
- each block contains data and it also guarantees for the authenticity of the previous blocks
- to append a new block is the only admitted operation
  - in some implementations, appending a block is called ***mining***

Blockchain and DLT

- adding a block is critical, only one node at a time can call for this operation
  - else the blockchain may become inconsistent
- selecting the node that has the right to make the update requires consensus of all nodes in the P2P network

Remember that:

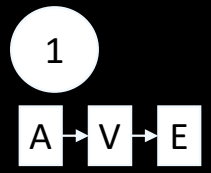
- each node keeps a copy of the entire blockchain
- each node should have **the same copy** of the blockchain

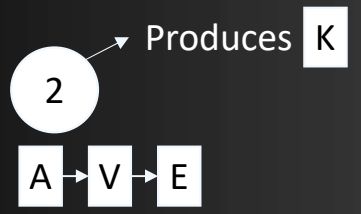
Consensus in DLT

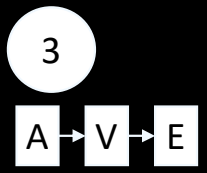
How to make sure that each node has the same copy of the blockchain?

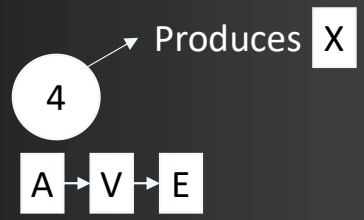
Consensus in DLT

Nodes 2 and 4 produce a new block to append to the blockchain



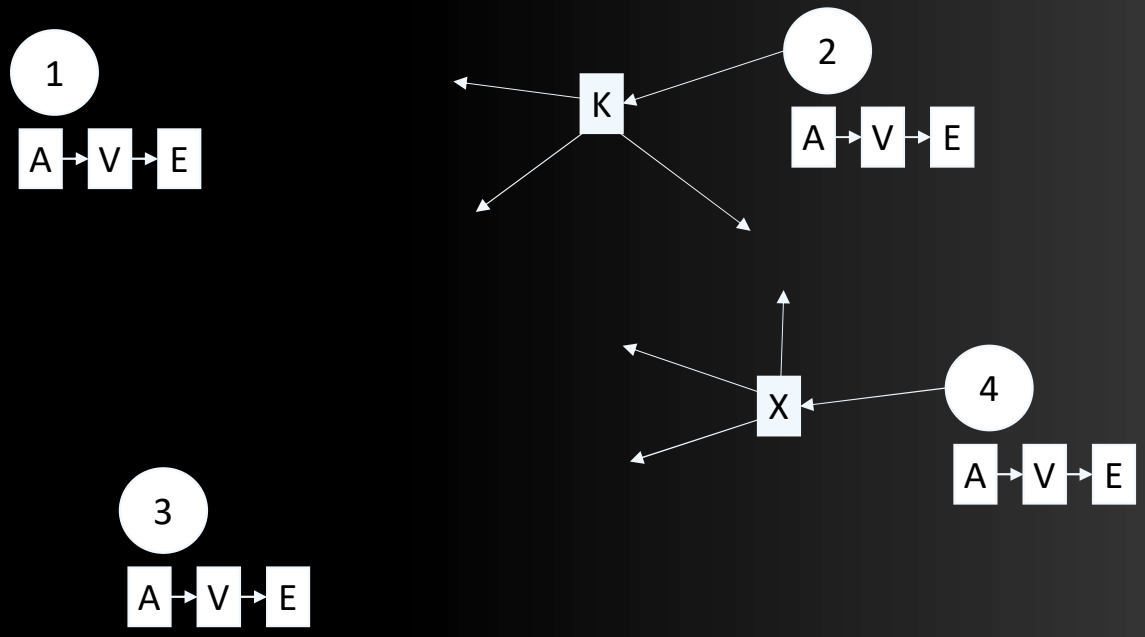






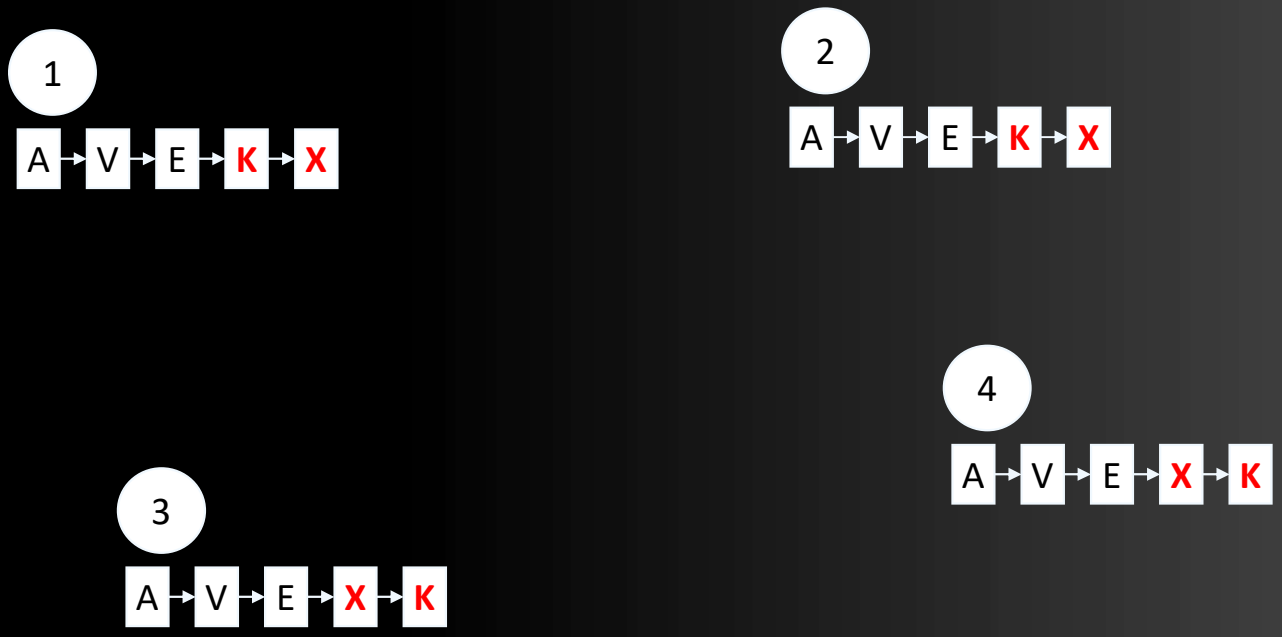
Consensus in DLT

Nodes 2 and 4 send the respective blocks to the other nodes, but transmission may take time…



Consensus in DLT

… and if blocks K and X arrive in different order at different nodes, then the blockchain becomes inconsistent!



Consensus in DLT

Furthermore…

- What if some blocks are lost during transmission?
- What if some node maliciously send different blocks to different nodes?
- …

This problem has been studied extensively in the literature, under the name of Bizantine Generals Problem (AKA consensus problem)

Bizantine Generals Problem

The consensus problem

Byzantine Generals Problem

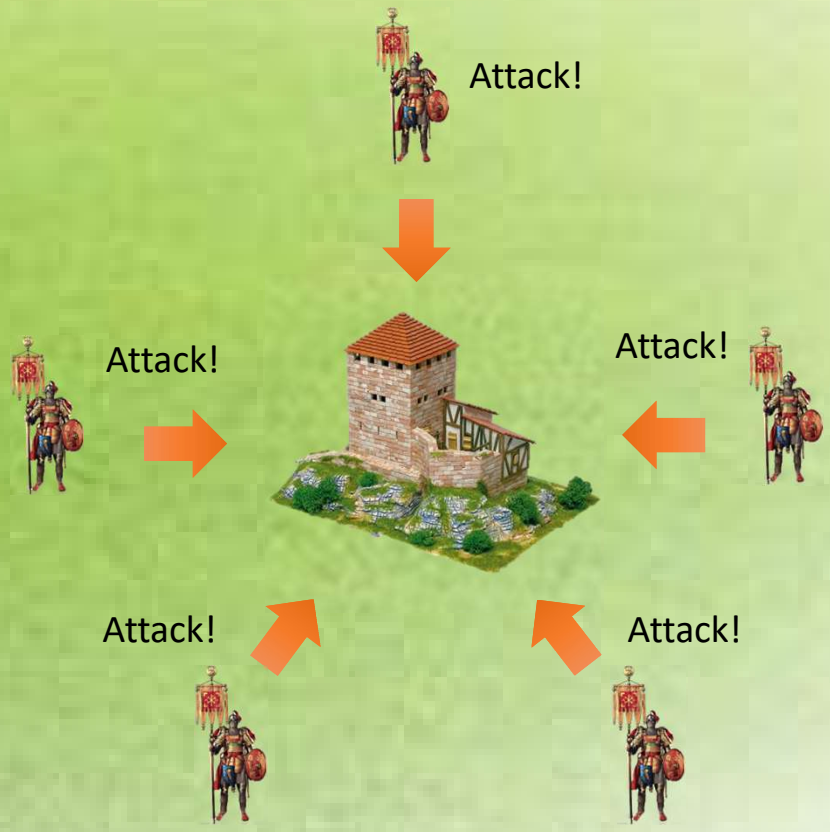
“several divisions of the Byzantine army are camped outside an enemy city, each division commanded by its own general.

The generals can communicate with one another only by messenger.
After observing the enemy, they must decide upon a common plan of action.”

Byzantine Generals Problem

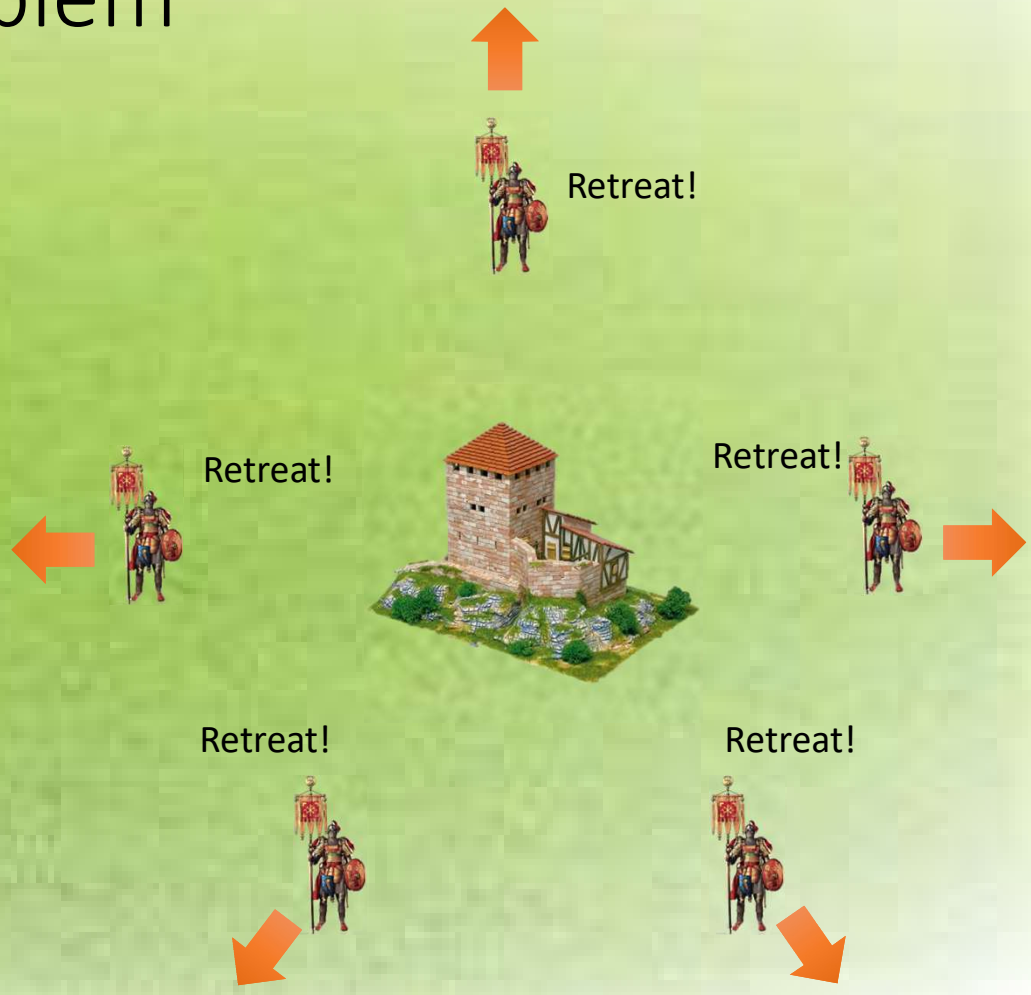
Generals should reach a consensus on the plan.

It could be ATTACK…



Byzantine Generals Problem

… or RETREAT.

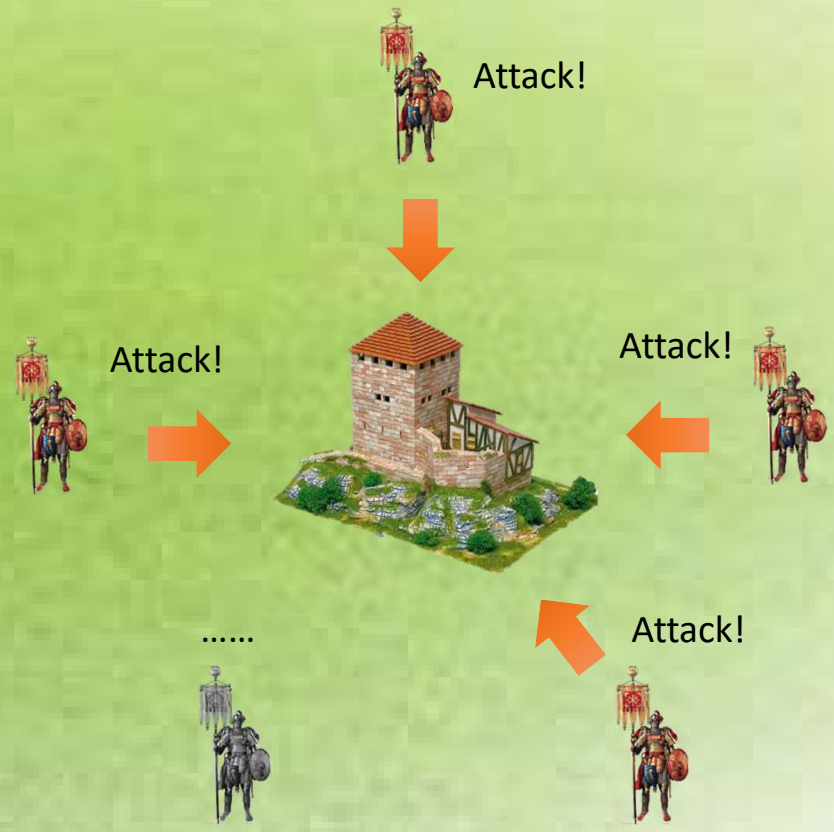


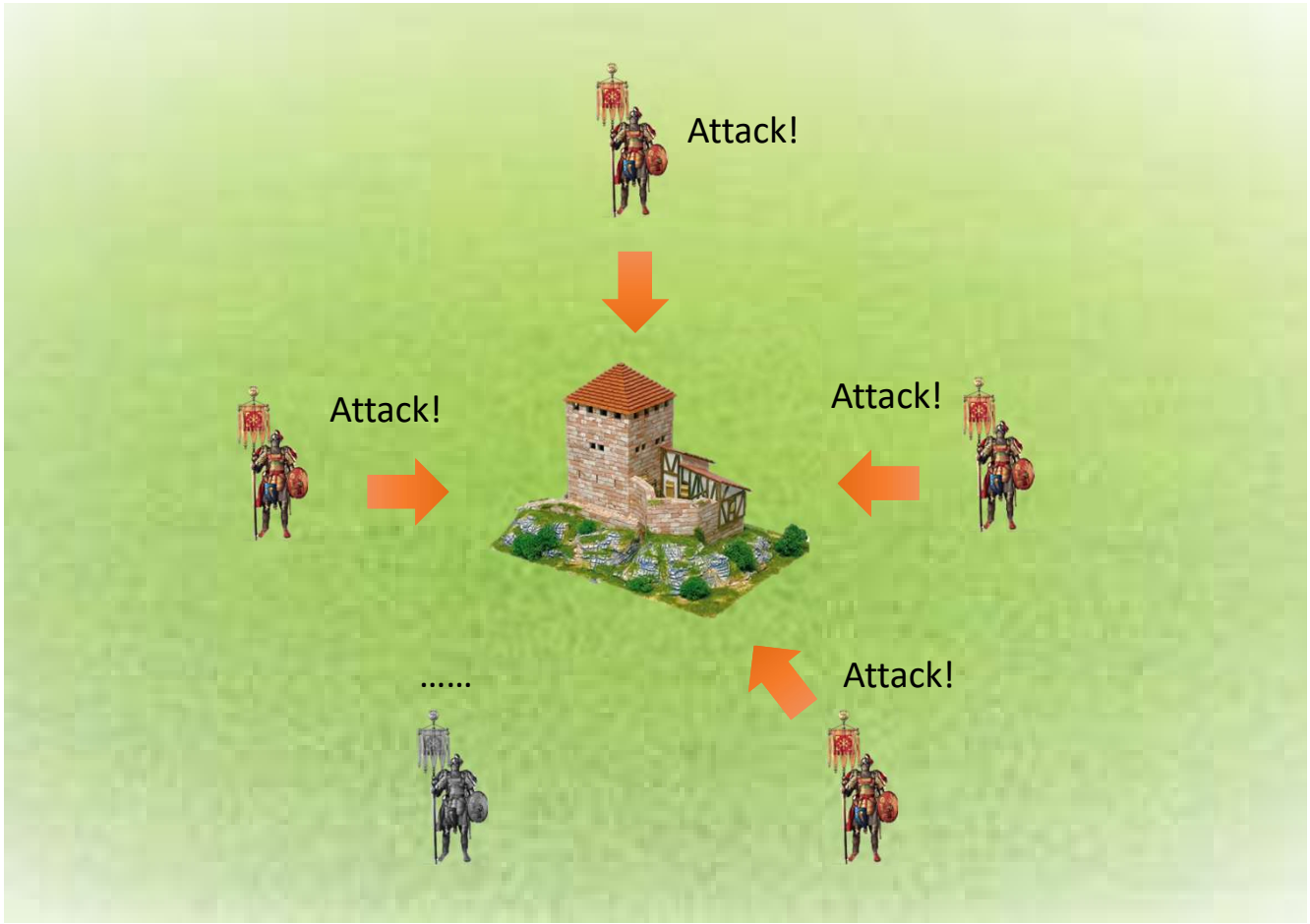
Byzantine Generals Problem

If all loyal generals reach a consensus, either attack or retreat, the battle plan succeeds.

Else the battle plan will fail.

but there may be traitors…





Question…

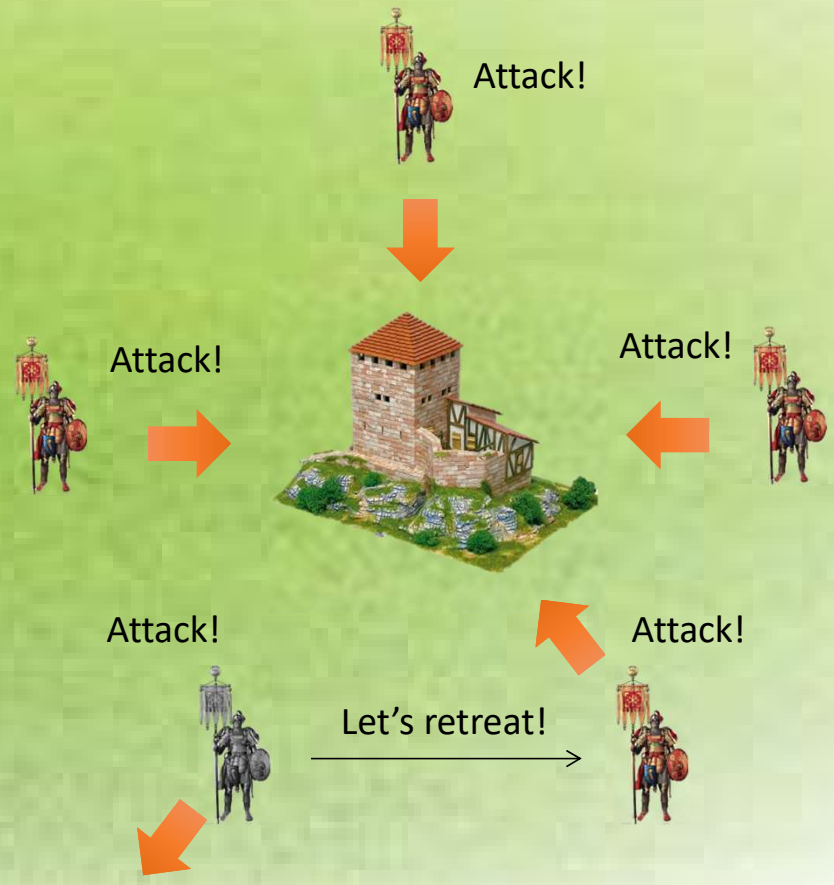
What does a traitor can do to make the plan fail?

Byzantine Generals Problem

If all loyal generals reach a consensus, either attack or retreat, the battle plan succeeds.

Else the battle plan will fail.

but there may be traitors…

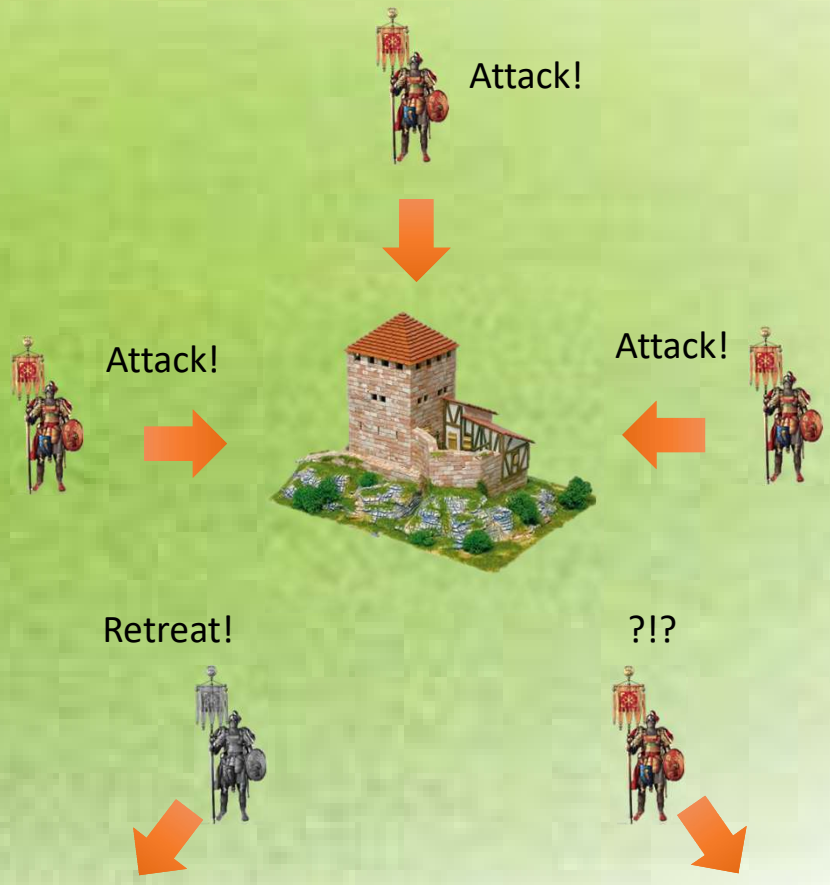


Byzantine Generals Problem

Traitors can act arbitrarily,

and if not all loyal generals take the same decision…

… the army is defeated!



Byzantine Generals Problem

There's also a simplified version of the problem:

"A commanding general sends an order to his $n - 1$ lieutenant generals such that:

a. all loyal lieutenants obey the same order

b. if the commanding general is loyal, then every loyal lieutenant obeys the order he sends"

Byzantine Generals Problem

consensus/agreement: “all loyal lieutenants obey the same order”

validity: “if the commanding general is loyal, then every loyal lieutenant obeys the order he sends”

Impossibility conditions

Assumptions:

- there are n generals and m traitors
- generals send dispatches through messengers:
  1. each dispatch that is sent is delivered correctly (reliable communications)
  2. the receiver knows the general who sent it (authenticated dispatches)
  3. loss of dispatches can be detected (synchronous communications)

No solution exists for $n \leq 3m$

Notes about the assumptions

1. each dispatch is delivered correctly (reliable communications)
2. the receiver knows the general who sent it (authenticated dispatches)
3. loss of dispatches can be detected (synchronous communications)

They are all necessary. Without these no solution is possible!

- 1) and 2) prevent the traitor from interferring with the communications of the other generals
  - If dispatches can be altered the traitor may send fake dispatches pretending he is the commanding general or anyboby else… nobody will be able to take the right decision or to isolate the traitor.
  - If the dispatches are blocked the traitor may prevent any decision
- 3) foils a traitor who tries to prevent a decision by not sending messages
  - With asynchronous communications there's no solution anyway

Notes about the assumptions

Still about synchronous & reliable communications:

- assume there is a solution for unreliable, asymmetric communications that lets the loyal generals to take a common decision (in a finite time *t*)…
- … and consider the last message *msg* exchanged:
  - *msg* may be delayed arbitrarily or even lost…
  - If *msg* is *necessary* to take a decision:
    if it does not arrives at time *t* then a decision at time *t* is impossible (for any *t*)
  - If *msg* is *not necessary* to take a decision:
    then you can avoid it and produce a new protocol that does not use it… in the end you will have an empty protocol…

Impossibility of solution – proof sketch

consider first the case $m = 1$,

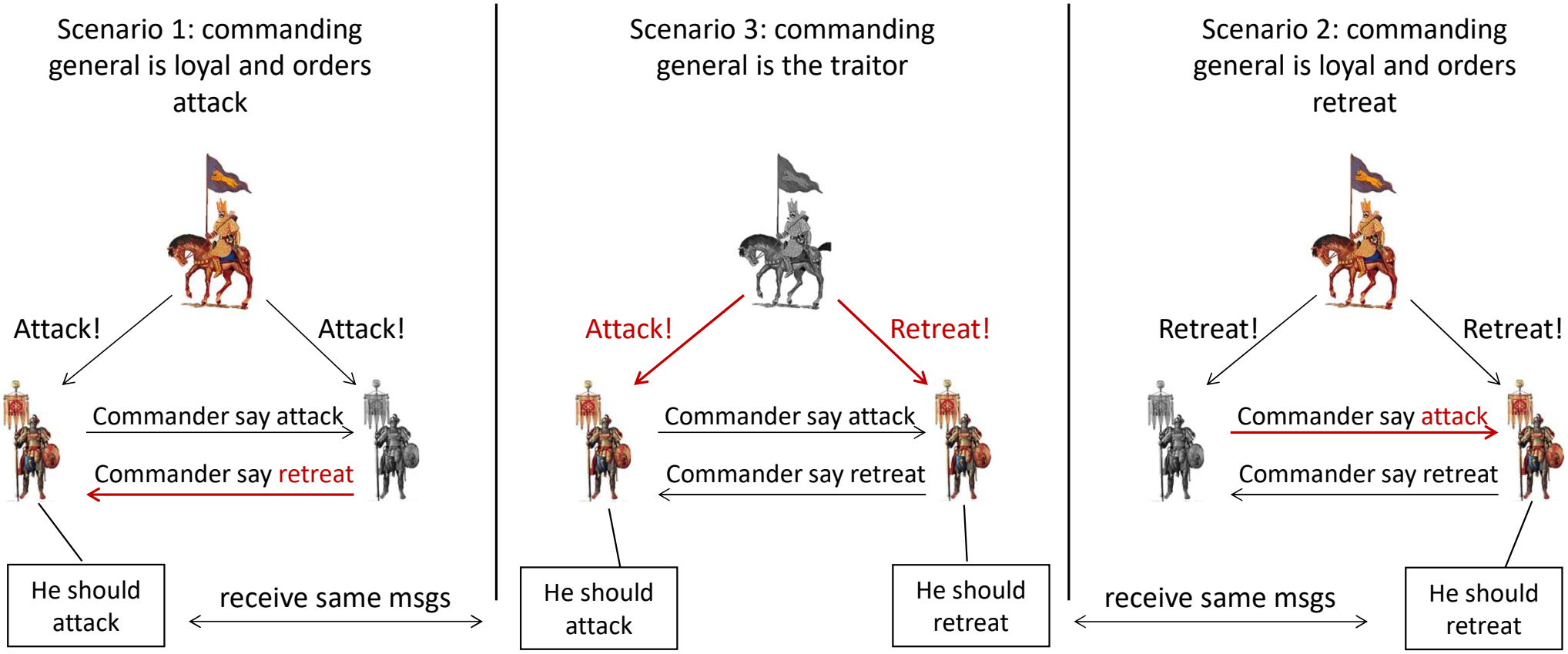
assume that there are $n = 3$ generals

- one commanding general and two lieutenant generals
- can be generalized to arbitrary n

there are three scenarios:

1. The commanding general is loyal and commands attack
2. The commanding general is loyal and commands retreat
3. The commanding general is the traitor and he commands attack to one lieutenant and retreat to the other
   - If he gives the attack command (or retreat) to both lieutenant, the attack (or the retreat) will succeed

Impossibility of solution – proof sketch



Impossibility of solution – generalization

The proof can be generalized to an arbitrary *n*:

Say that an algorithm *f* solves the problem for *n*=3*m* generals with *n*>3:

We show that *f* also solves the problem with *n*=3, which is a contradiction.

Let us consider three generals: the commanding general X and the liutenant generals Y, Z

Given the solution *f* for an arbitrary *n*=3*m*, with *m*>1, X, Y and Z use *f* in this way:

- X simulates himself (the commanding general) and a group of *m*-1 simulated liutenant generals
- Y simulates himself (a liutenant general) and another group of *m*-1 simulated liutenant generals
- Z simulates himself (a liutenant general) and the remaining group of *m*-1 simulated liutenant generals

Impossibility of solution – generalization

- X simulates himself (the commanding general) and a group of *m*-1 simulated liutenant generals
- Y simulates himself (a liutenant general) and another group of *m*-1 simulated liutenant generals
- Z simulates himself (a liutenant general) and the remaining group of *m*-1 simulated liutenant generals

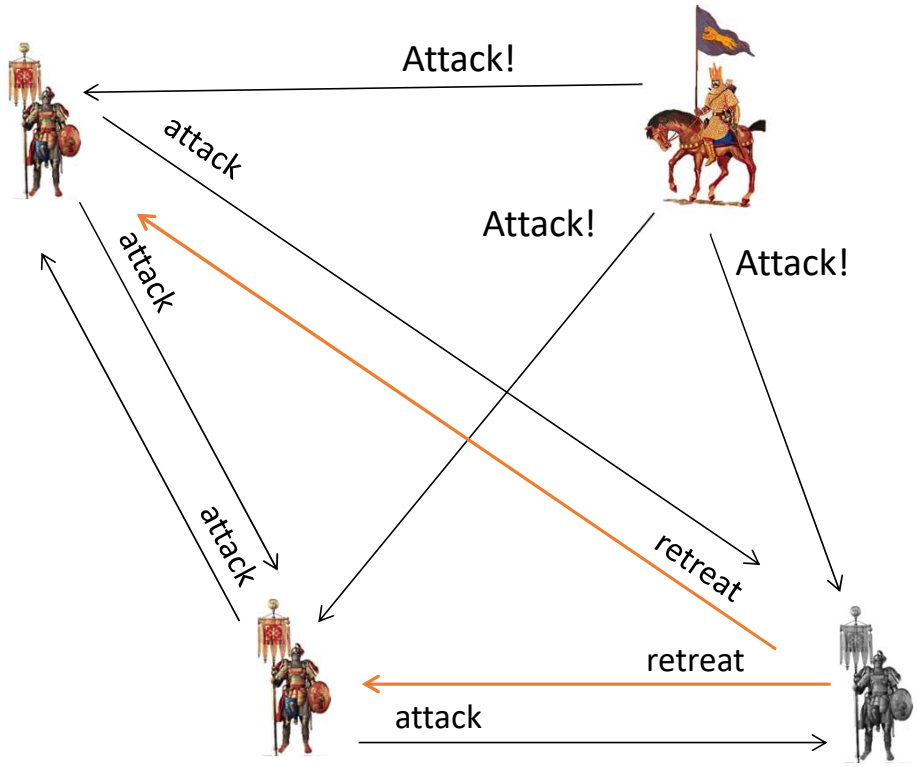
The conclusion:

- Since only one in {X, Y, Z} is a traitor, the traitors are exactly *m* in this construction
- Hence, using *f* over the *n*=3*m* simulated generals lets the loyal generals in {X, Y, Z} to reach a consensus.
- This is a contradiction because this construction provides a solution of the problem for *n*=3, which is known to be not possible.

Solution (must be $n >= 3m + 1$)

Case 1, the general is loyal and orders attack:

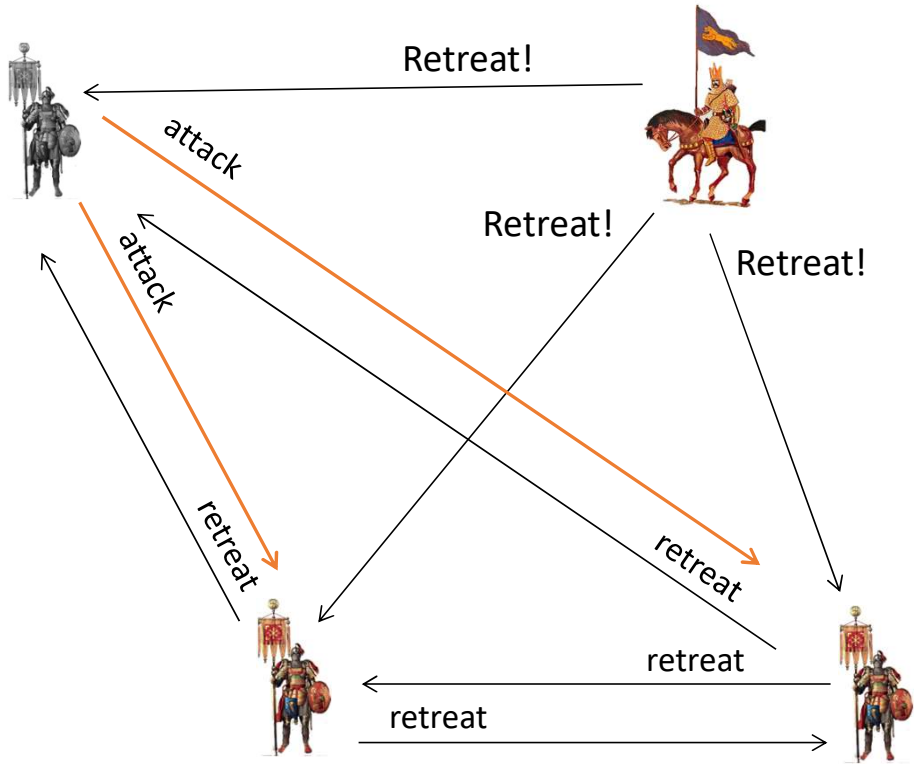
- the two loyal generals receive 2 attacks dispatches, at least one from a loyal general;
  - hence each loyal general receives a majority of attack dispatches;
- the commanding general and they both attack
- success!



Solution (must be $n >= 3m + 1$)

Case 2, the commanding general is loyal and orders retreat:

- the two loyal generals receive 2 retreat dispatches, at least one from a loyal general
  - hence each loyal general receives a majority of retreat dispatches;
- the commanding general and they both retreat
- success!

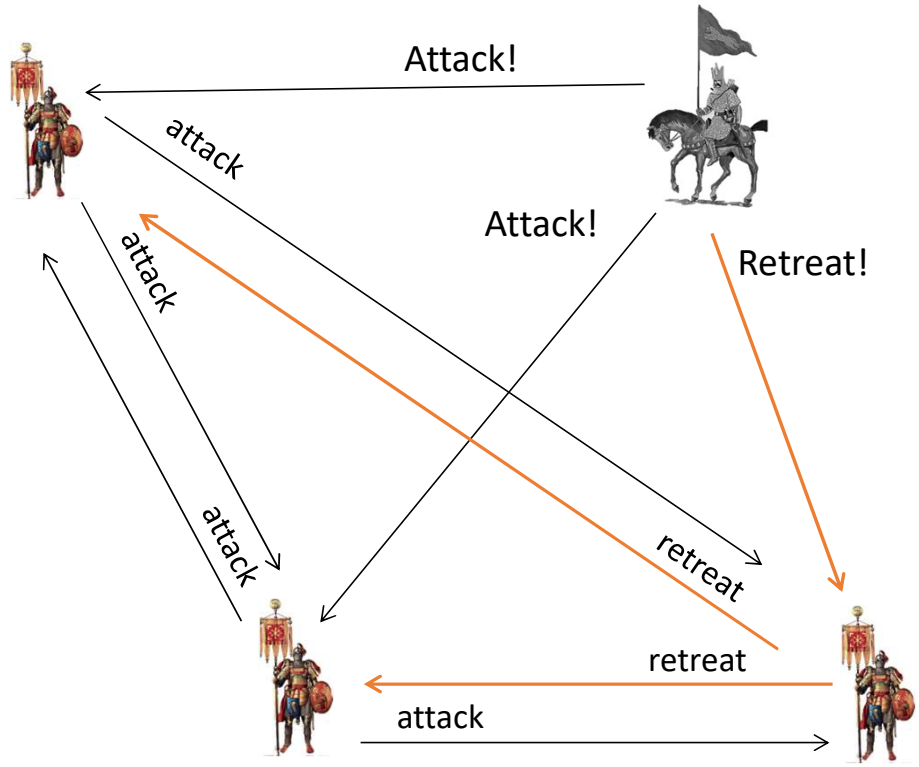


Solution (must be $n >= 3m + 1$)

Case 3, the commanding general is traitor:

- each of the three loyal generals receive 2 attacks dispatches, at least one from a loyal general
  - hence each loyal general receives a majority of attack dispatches;
- the three of them attack
- success!

Note: if majority of retreats they all agree on retreat, which is also a success



Algorithm – sketch

- The commanding general send his command to all lieutenant generals
- Each lieutenant general:
  - if receives no command then assumes the command received was retreat
  - forwards to all other $n-2$ liutenants the command received
  - receives the commands from the other $n-2$ liutenants
  - acts as the majority of commands received
    - i.e. in total received $n-1$ commands, one from the commanding general and $n-2$ from the lieutenant generals
    - if the majority is for retreat then retreat else attack

About solutions

- The previous solution requires the $O(n^2)$ dispatches
- Assuming messages can be signed it is possible to find a solution that works even $n \leq 3m$:

  add to the assumption a 4th requirement:

  4. a loyal general's signature cannot be forged and any alteration of his dispatches can be detected. Anyone can verify the authenticity of a message.

  Now a lieutenant can only forward the original message received from the commanding general (no way to alter it).

  Even if the traitors collude and forge the signature of each other, that's not a problem

  Note that the solution now work for any n, but of course it if vacuous if $n < 2m$

References

The original work (can be found on the internet):

L. Lamport, R. Shostak, M. Pease, “The Byzantine Generals Problem”, ACM transactions on programming Languages and Systems, 1982

Consensus in DLT

- we have seen the meaning of consensus in distributed systems
  - give rise to the Bizantine Generals Problem
- blockchains (and in DLT in general) have the same problem when they need to make an update
  - a large number of nodes in the P2P network may be unfaithful
  - need to assume that a “vast” majority of the nodes are faithful

Consensus in DLT

- In the blockchain context, the consensus problem is formulated as:

> "the non-faulty and faithful (correct) nodes should agree on one block of transactions at a given index of a chain of blocks"

- This consensus problem can be stated along three properties:

1. agreement: no two correct nodes decide different blocks;
2. validity: the decided block is a block that was proposed by one node;
3. termination: all correct nodes eventually decide.

Consensus in DLT

- A protocol solving the consensus problem is necessary to guarantee that blocks are totally ordered
  - prevents concurrently appended blocks containing conflicting transactions
- Different blockchains use consensus algorithms based on different principles:
  - Proof of work (PoW)
  - Proof of stake (PoS)
  - Proof of Authority (PoA)
  - and many others…

Proof of Work

- assume a node (a *miner* in this case) wants to make an update (add a new block)
- it computes first a number (nonce) that solves a very complex problem
  - assume that verifying the nonce is easy, i.e. the solution to the complex problem can be easily verified
- then it tells all other nodes the problem is solved:
  - the nonce is the proof it solved the problem (Proof of Work)
  - The PoW gives it the right to update the blockchain
- each node in the P2P network verifies the PoW and makes the update accordingly
  - since the problem is very difficult this reduces (makes negligible) the chance that two nodes request the update simultaneously

Proof of Work

- … but where to find a very complex problem?
- for example, invert a hash function by finding a nonce by which you can produce a hash with a specific property
  - the property could be any, for example that hash(nonce)=0
  - this requires a brute force computation
  - may take ages, but the complexity can be tuned by setting appropriately the length of the hash
- verification is easy: compute hash(nonce) and verify the result satisfy the given property

Proof of Stake

- to append a new block to the blockchain is called *forging* in this case
- the node that calls for the append is elected by a pseudo-random process
  - the main parameter of this process is the stake (a given amount of cryptocurrency given as guarantee for the transaction)
  - the larger is the stake the higher the probability of being selected
  - to avoid that only "rich" nodes update the DLT may also consider other parameters
  - E.g. something like time from last update (given by the time in which the stake of the node had remained in the deposit waiting for the right to update)

Proof of Authority

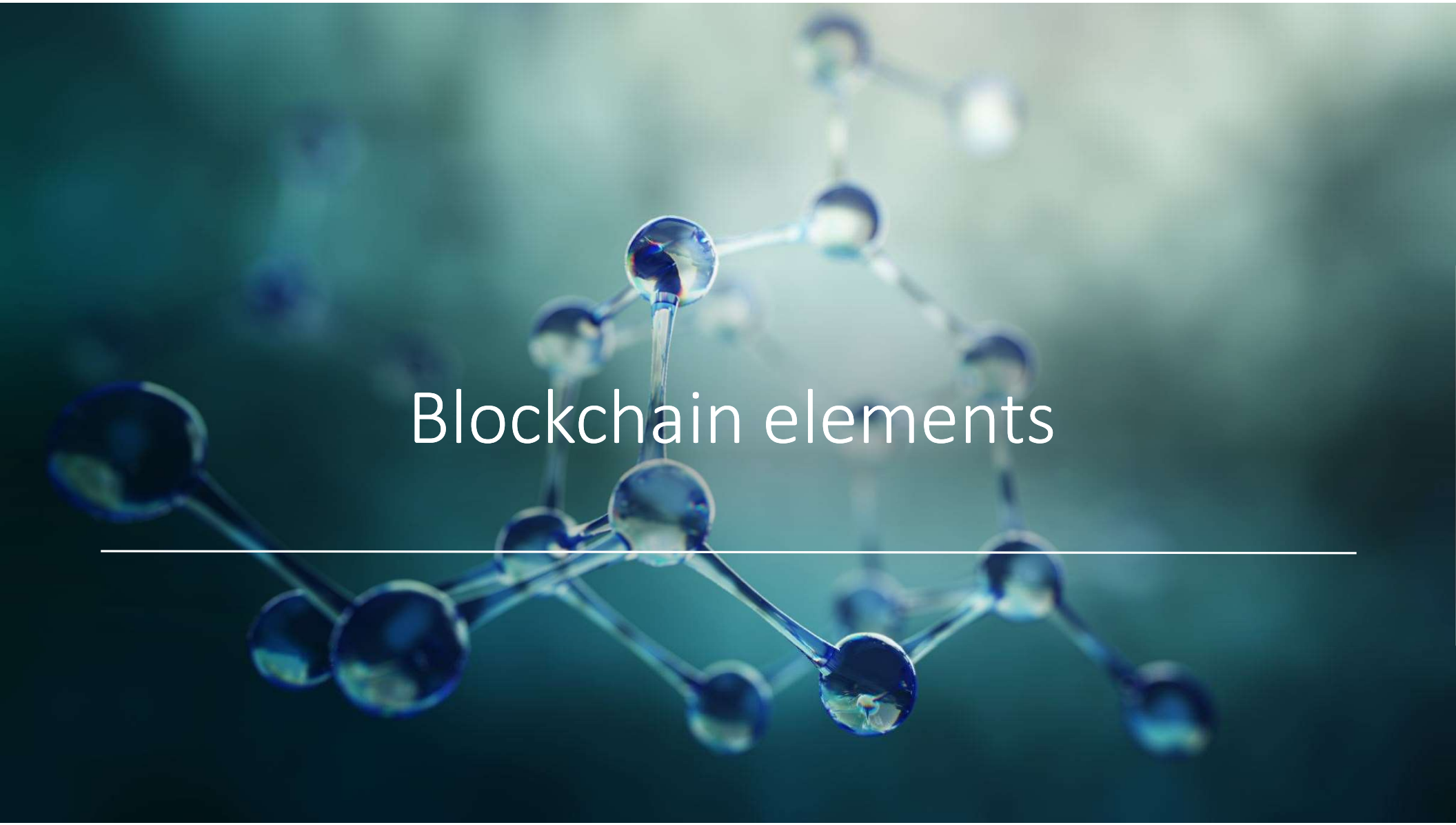
- the network is no longer a pure P2P
- two kind of nodes: ordinary and validators
- only validators can append
  - append only when a majority of validators reach the consensus
- need to ensure that:
  - validators have public and verifiable identities (all other nodes should trust them very much)
  - each validator has a reputation that is badly affected if it misbehaves

Proof of Authority

- with PoA the control of the blockchain is partially centralized
- reduces the complexity of consensus, scales better
- could be a solution for private / consortia blockchains

Review questions

- How does the Bizantine Generals Problem comes into the picture with blockchains?
- Can we fully trust the consensus mechanisms PoW, PoS, PoA? Explain…



Blockchain elements

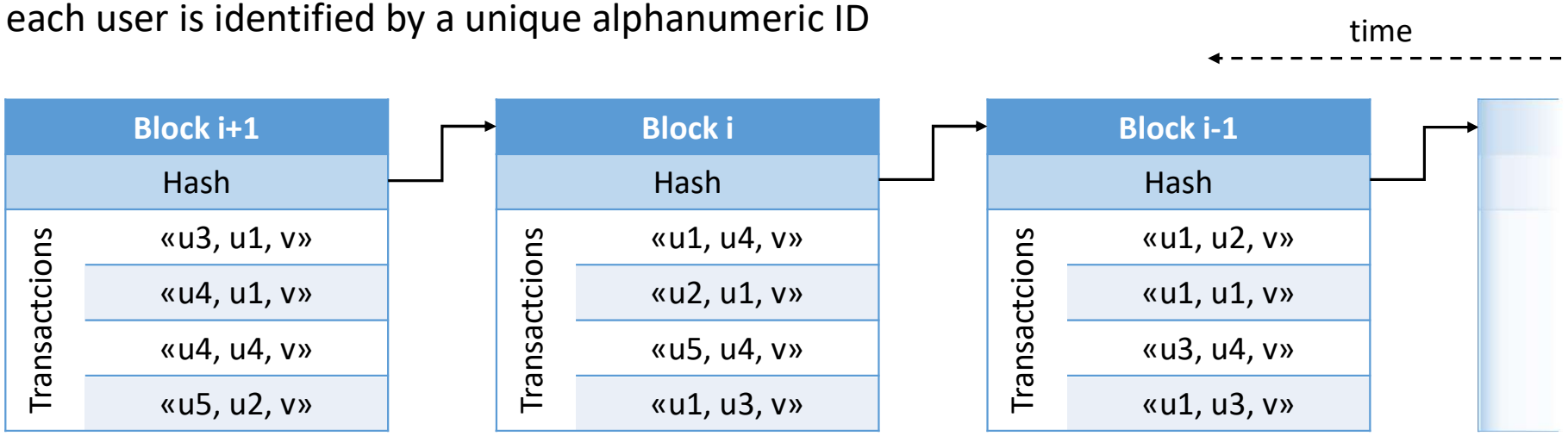
Blockchain

- the blockchain can be defined as a shared and trusted public ledger for making transactions
  - everybody can inspect it
  - nobody controls it
  - the transactions within cannot be altered
- the blockchain is shared and tamper-evident and it thus provides a single point of truth

- parties involved in business can use a blockchain to record the history of business transactions

Blockchain

- a blockchain contains a sequence of blocks in chronologic order
- each block contains:
  - data organized as a set of transactions
  - the cryptographic hash to the previous block
- each transaction is a transfer of value (for example cryptocurrencies, but not only) between users
- each user is identified by a unique alphanumeric ID



Blockchain

- The first block is called the *genesis block*, set at initialization of the blockchain
  - just for initialization, does not contain transactions!
- hash pointer is one-way:
  - guarantees that previous block cannot be altered…
  - … and by induction the entire blockchain cannot be altered
- the only way to alter the blockchain would be:
  - to replace a block but then need to change the hash of all subsequent blocks…
  - … or otherwise forge a new block to put in between (a previous and a next) but that must have the same hash of the previous block

Users of the blockchain

- Ethereum and Bitcoin (well-known cryptocurrencies) associate each user to:
  - public and private key of the user
  - the address (used to keep the account and to send or receive coins), which is just a short representation of the public key
- the private key guarantees the property of the address
  - … and thus, of the coins within
  - when two users make a transaction, they sign it with their private key

Miners

- Miners are the entities that have the role of producing a new valid block to append to the blockchain. They:
  - maintain the blockchain
  - control the validity of the transactions to be inserted in a block
  - build blocks
  - implement the consensus algorithm (using PoW or any other method of consensus)
  - get the reward for their work each time they succeed in adding a valid block to the blockchain
- The activity of the miners is called *mining*

Reward for miners

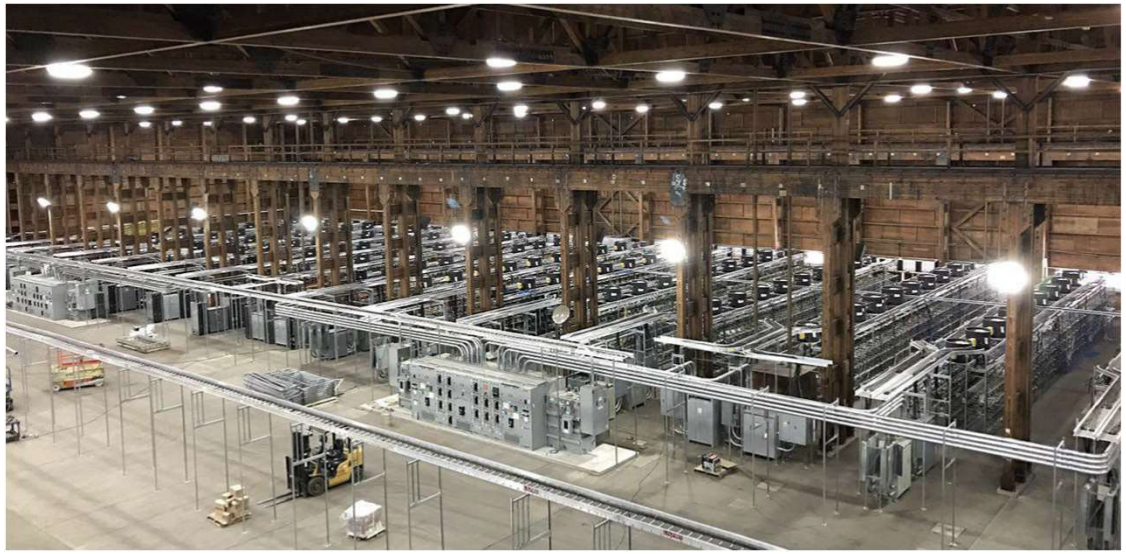
In Bitcoin:

- when the miner inserts a block in the blockchain, it adds a *coinbase transaction* to the block to obtain its own reward
- the coinbase transaction creates new cryptocurrency

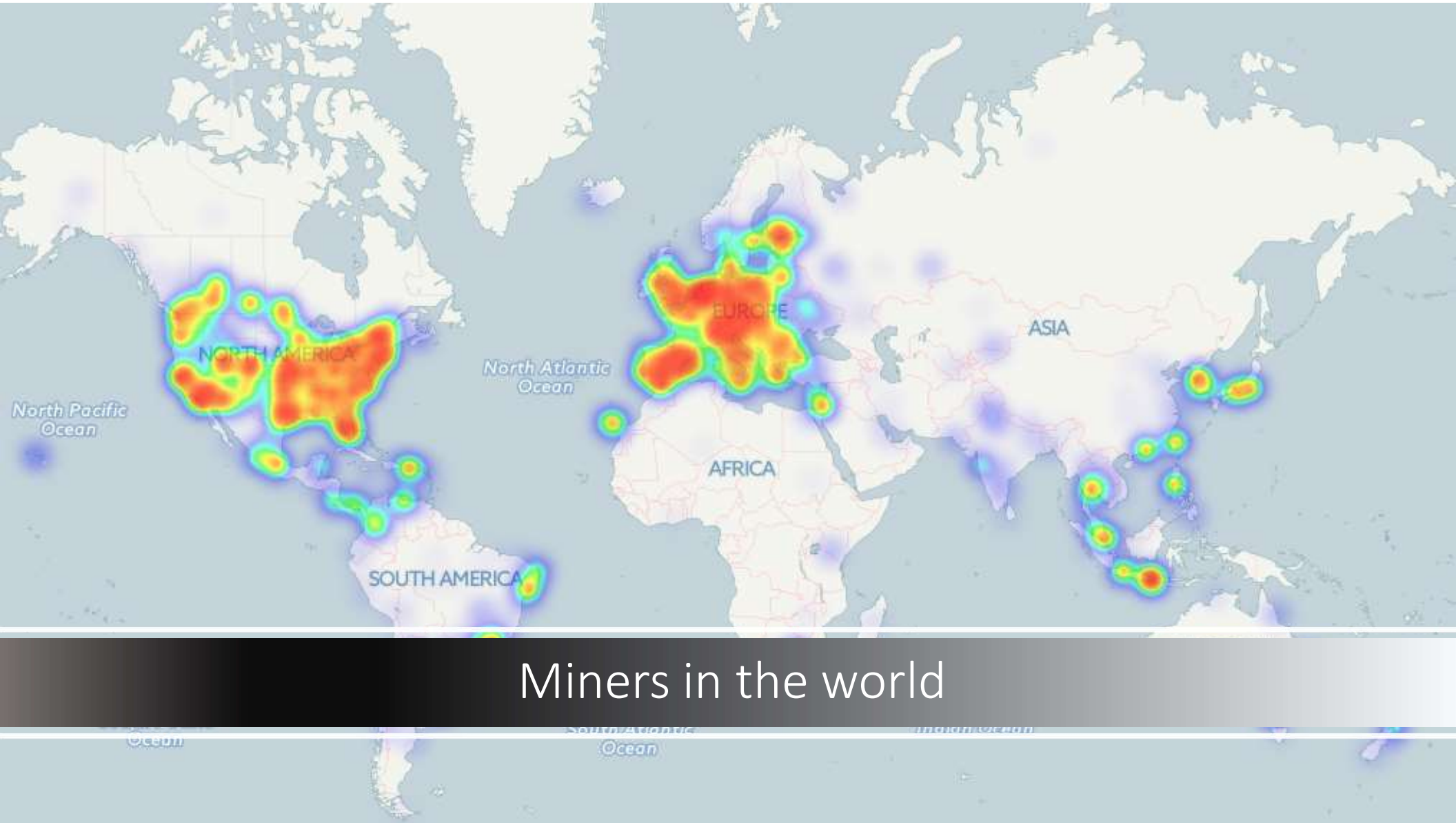
In Ethereum:

- does not have coinbase transactions, but there is an automatic constant reward for the activity of mining;
- originally, the user that requests to record a transaction pays a fee for the miner depending on the actual cost of the PoW (this also in Bitcoin)
  - Now Ethereum shifted to Proof of Stake (see later)

… considering the complexity, costs and rewards of mining, this gave rise to mining farms



Rewards for miners



Miners in the world

The New York Times

U.S. Seizes Share of Ransom From Hackers in Colonial Pipeline Attack

Investigators traced 75 Bitcoins worth more than $4 million through nearly two dozen cryptocurrency accounts.

The cyberattack on Colonial Pipeline last month shut down its computer systems, leading to soaring gas prices and panic buying. Shawn Thew/EPA, via Shutterstock

By Katie Benner and Nicole Perlroth

June 7, 2021

… and pitfalls…

Review questions

- How does one can alter a blockchain? Why does this is considered infeasible?
- Blockchains are not really a «green» technology… why?

Consensus with Proof of Work

Mining and consensus based on PoW

- Ethereum (in the past) and Bitcoin are the most used blockchains based on PoW
- consensus based on PoW – assumptions for next slides:
  - transactions are transfers of digital assets (coins) from a user u_i to a user u_j
  - transactions are sent in broadcast to all miners in best-effort: some miner may not receive a transaction, the broadcast is not reliable
  - if a client broadcasts twice the same transaction this will be considered as a new one
  - miners combine transactions into blocks and implement the PoW consensus

Mining and consensus based on PoW

- A miner M collects all received transactions and checks the validity of each one:
  - the account balances should remain non-negative: amount to be exchanged exists and it is not already spent
  - the signatures of the users making the transaction are valid
- Then it puts the valid transactions into its own *transaction pool*
- When the number of transactions in the pool is sufficient, the miner creates the new block b and sends the block to all other miners in the P2P network to be stored

```
function create_block() {
    // executed by miner m_i when the
    // transaction pool is full, to
    // propose the append of a new block

// let B_i be the blockchain at miner m_i
b = new block;
b.transactions = get_transactions(pool);
while true do
   nounce=local-random-coin()
   b.pow=nounce;
   b.parent=last_block(B_i);
   if solve_cryptopuzzle(b) {
       broadcast(b);     //included itself
       break;
   }
}
```

Consensus algorithm: creation of a new block

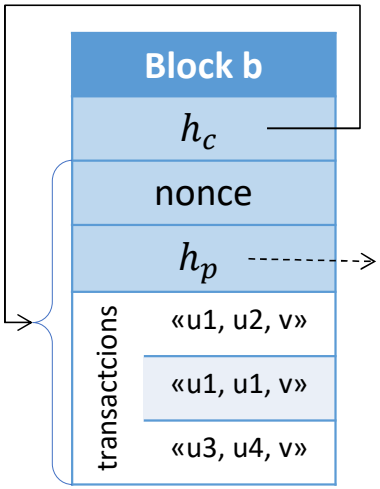
```
function update(b) {
    // executed by miner M_i upon
    // reception of block b to append

   // let B_i be the blockchain at miner M_i
   if !check_validity(b) { reject b; return; }
   B_i = B_i ∪ b;
}
```

Consensus algorithm: append a received block

Block validity

Formally a block is a tuple: $b = \langle nonce, h_c, h_p, T \rangle$, where:

- $nonce$ is the proof of work
- $h_c = hash(nonce \parallel h_p \parallel T)$ it is the *current hash* of b
- $h_p = h_c^i$, that is, it equals the current hash contained in the last block b_i in the blockchain (hash to the previous block)
- T is a set of transactions

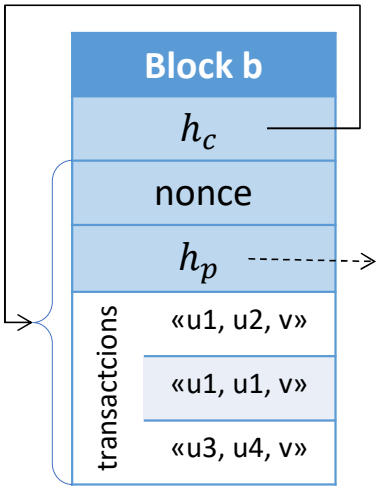


Block validity

the $nonce$ is computed by M in such a way that $h_c \leq L$ where L is a target value, common to all miners.

since the hash function is one-way, determining a $nonce$ such that $h_c \leq L$ forces the miner M to make a brute force search.

the complexity of this search depends on the value of L: the smaller it is the more complex is the work for the miner.

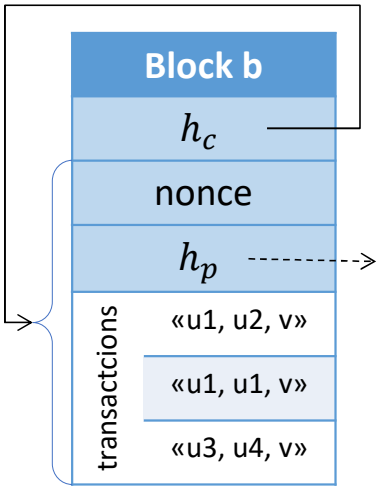


Block validity

concluding, block b is valid if:

- all transactions it contains are valid
- $h_p = h_c^i$
- $h_c = hash\big(nonce \parallel h_p \parallel T\big) \leq L$

Once added to the blockchain it becomes block b_{i+1}



Mining and consensus based on PoW

- the value of L is set depending on the computational power of the network of miners
- it is configured to ensure that new blocks are added to the blockchain regularly, at a given pace
  - in Bitcoin typically one new block every 10 minutes

Mining and consensus based on PoW

- Upon reception of block b, any other miner checks the validity of all the transactions contained in the block
  - independent check of signatures and amounts of the exchange for each transaction in the block
- if there are invalid transactions the miner may put M in its "black list" and never accept anymore new blocks from M
  - this prevent a miner from cheating by adding inappropriate data to the blockchain
- If the transactions are valid the block is then appended to the blockchain
  - remember this happens in each miner!

Review questions

- What happens if a miner receives a valid block while it is mining another block?
- Is it possible that two miners produce the same block independently?
- Is it possible that two miners produce two different valid blocks at the same time?

Failures of PoW consensus

The failure model

- the value of coins is an incentive to execute the "*double-spending attack*"
  - that is, spending the same coin in two different transactions
- this corresponds to a byzantine failure model, in which byzantine faulty node may follow any different protocol than non-faulty nodes
- in blockchain implementations (e.g. Bitcoin or Ethereum) it is assumed that attackers may own a relatively small fraction of the total mining power of the system (<0.5)

The consensus requirements

Requirements are the classical ones of byzantine agreement:

Agreement: no two correct miners decide different blocks

Termination: all correct miners eventually decide a block

Validity: the decided block was proposed by a miner

However, miners in the blockchain operate over Internet, in which the assumption of synchronous communications does not hold

In this conditions byzantine agreement is not possible.

The consensus requirements

To cope with this impossibility, existing blockchains relax the classic Byzantine consensus in favor of probabilistic guarantees by exploiting randomization.

An example is the Monte Carlo Byzantine Agreement:

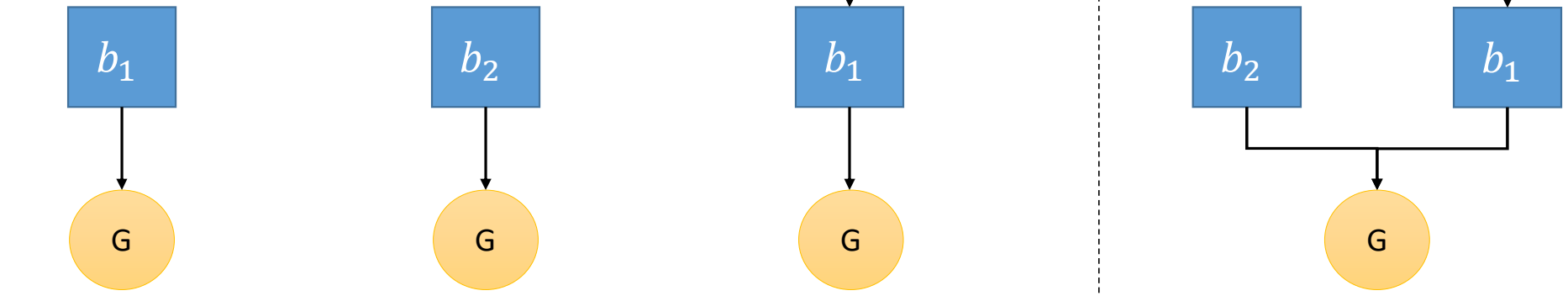
Probabilistic agreement: no two correct miners decide different blocks with probability at least δ

Termination: all correct miners eventually decide a block

Validity: the decided block was proposed by a miner

That's the case of PoW consensus

Local and global view of the blockchain



View of miner M_1

View of miner M_2

View of miner M_3

Global state of the blockchain

G Genesis block

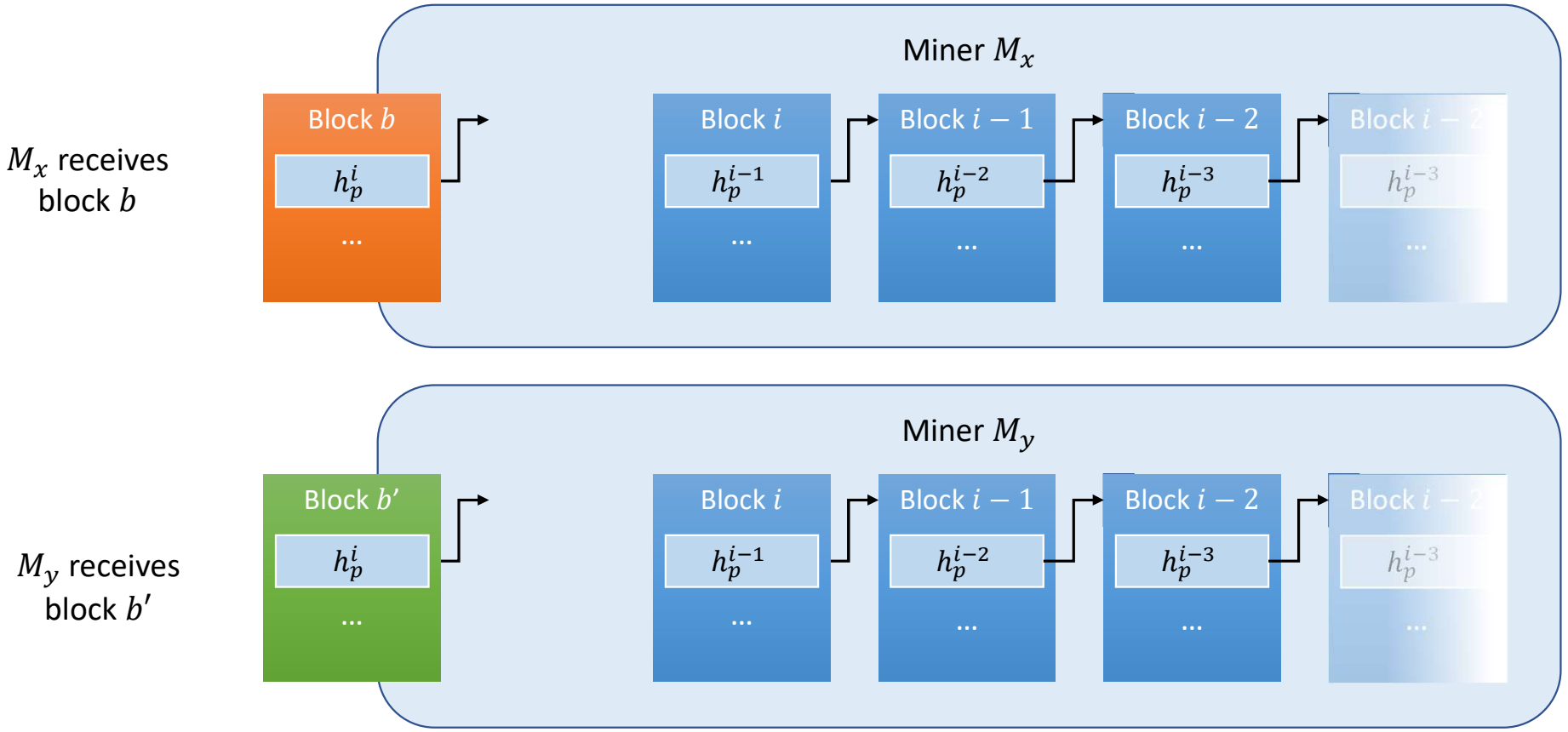
Forks in the blockchain

- if two valid blocks (say b and b') are produced at the same time…
- … the miners will receive the two blocks (in arbitrary order)…
- … each miner will append the two blocks to its local blockchain…
- … and the blockchain would fork into two branches…

- forks are a pathological situation and the PoW mechanism is used to reduce their chance
  - without PoW the forks would be rather frequent
  - … but PoW cannot avoid forks completely

- forks are the result of disagreements among miners
  - i.e failure of the consensus protocol
  - they may occur, fortunately not so frequently…

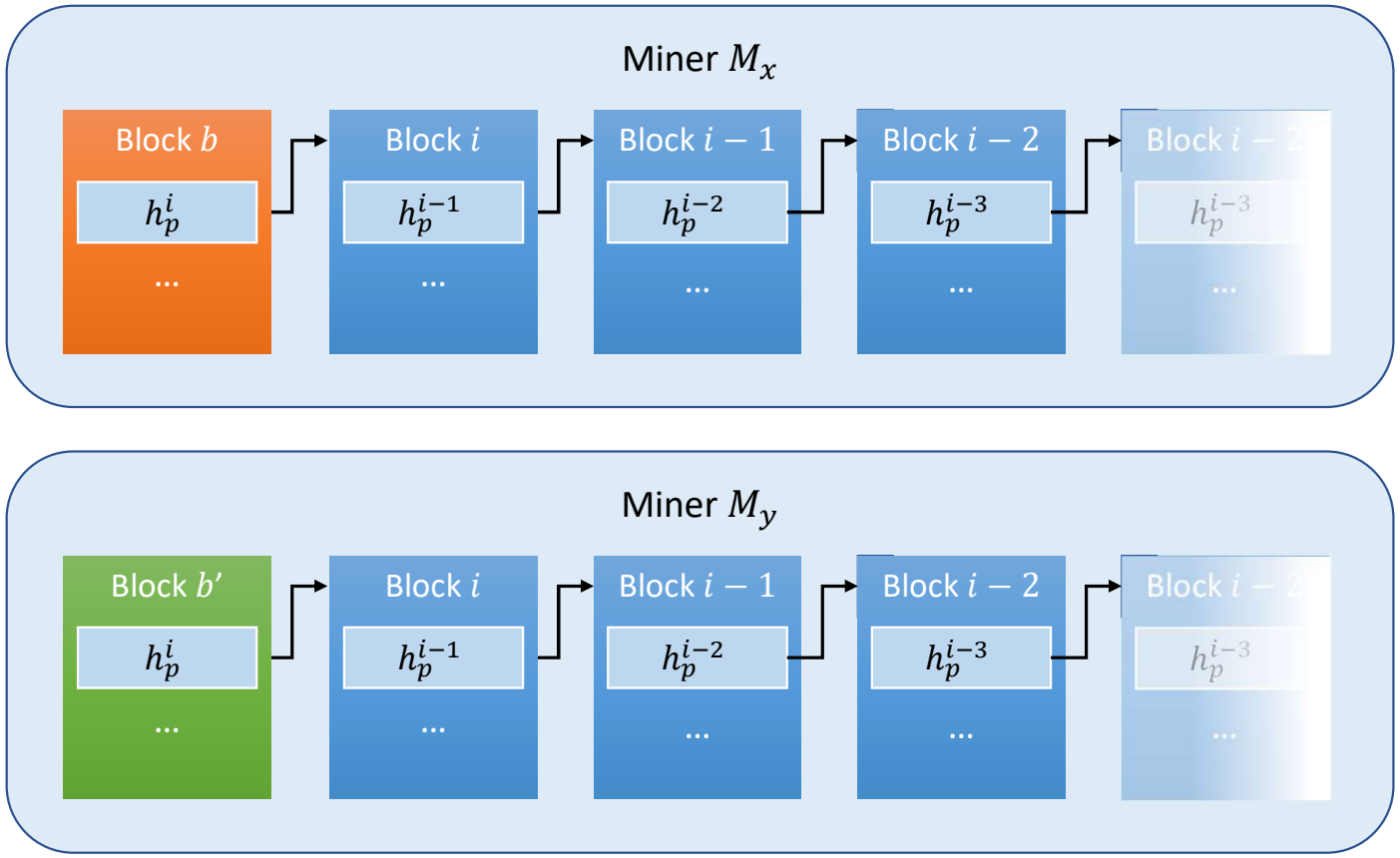
Forks in the blockchain



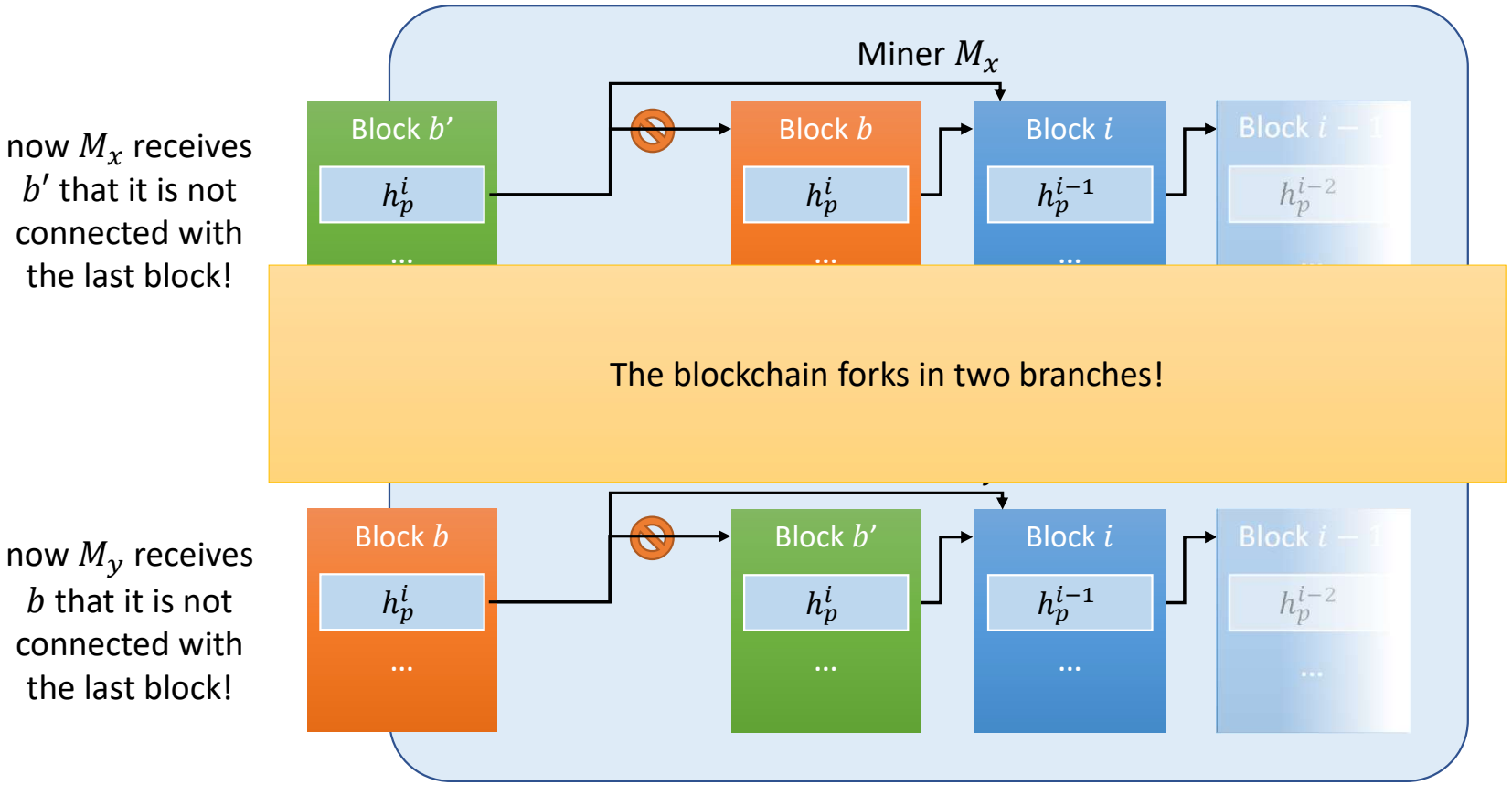
Forks in the blockchain

M_x appends b to its copy of the blockchain

M_y appends b' to its copy of the blockchain

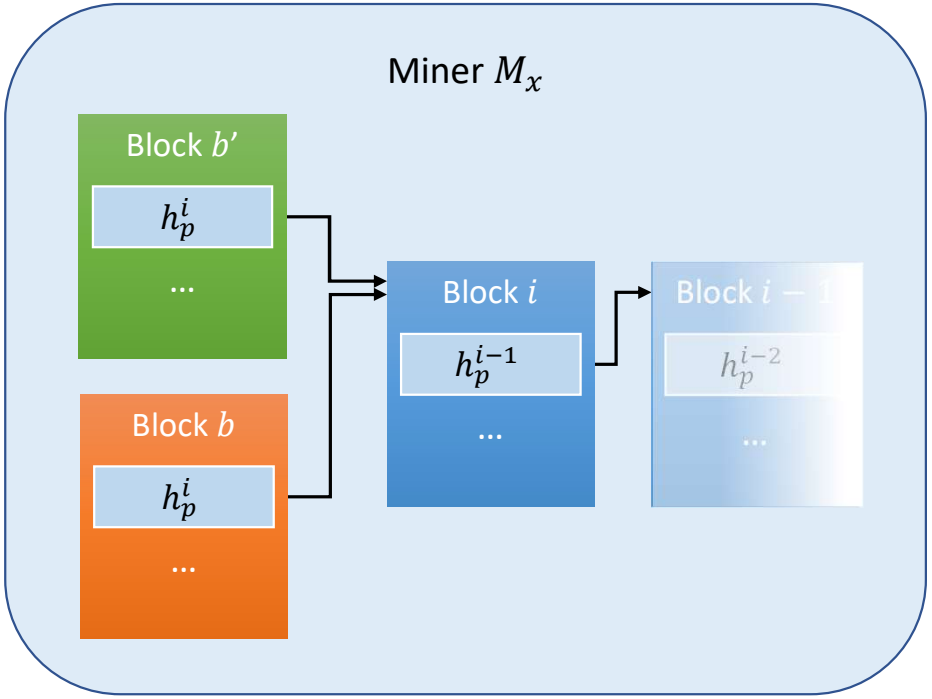


Forks in the blockchain



Forks in the blockchain

… and now, to which branch the new blocks should be appended?



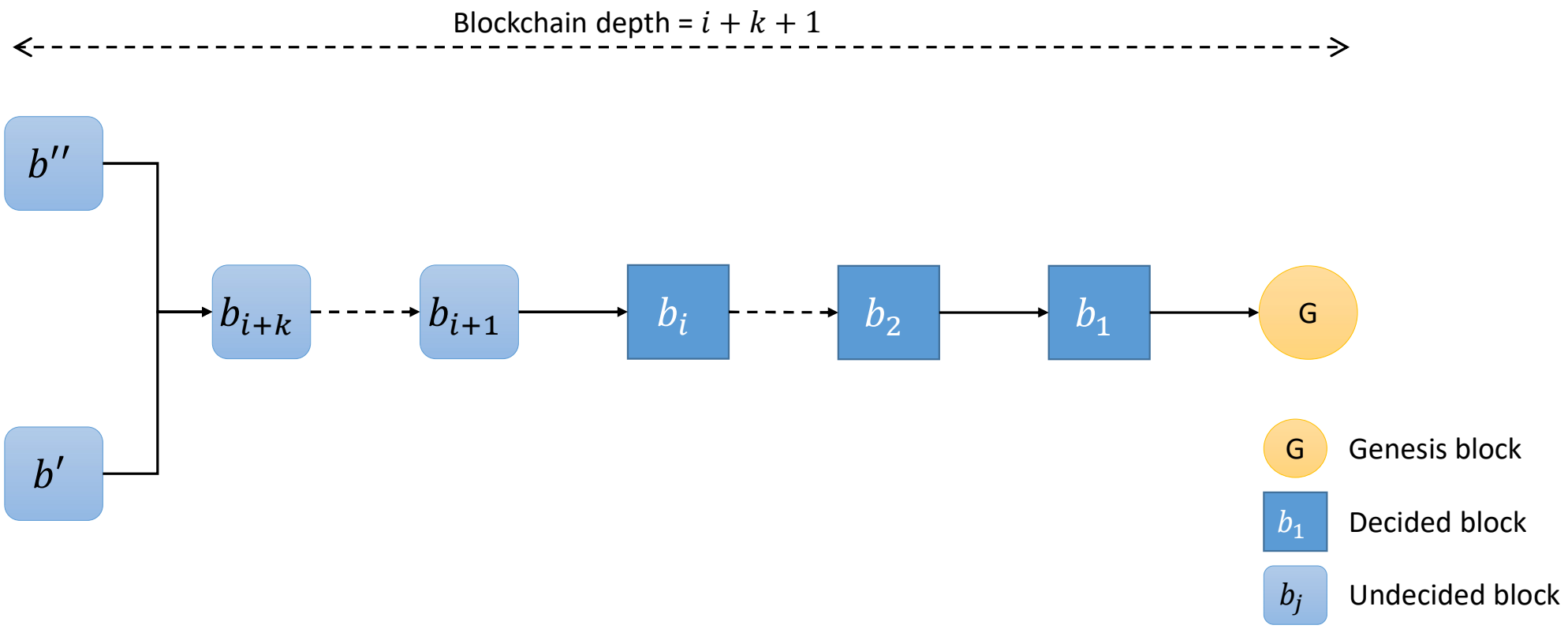
Forks in the blockchain

- due to forks the blockchain may become populated with several branches
- in addition, blockchains at different miners may also differ from each other
  - some miners may have not received a block (communications are unreliable)
- from this point on, a miner willing to produce a new block will append it to the ***main branch***
  - computed locally on its blockchain
- Main branch selection is necessary to resolve the forks and to define a deterministic state agreed by all miners
  - different approaches in Bitcoin and Ethereum

Main branch selection in Bitcoin

- When a fork occurs, Bitcoin selects ***the deepest branch*** as the main branch
  - The main branch is found by a pruning procedure on the blockchain executed by each miner
  - Each miner will then produce new blocks for the main branch
- The consequence of the selection of the main branch is that the blocks in the other branches are ***not committed***
  - hence they are not "valid"
  - the blocks attached to the dead branches will be disregarded by compliant miners, as if those blocks were never produced
  - … and the blockchain will thus converge again to the same branch
- To be considered committed, a block should have at least $m = 5$ other blocks appended to it
  - … this means that Bitcoin assumes that dead branches are cut before they become longer than 4 blocks

Forks in the blockchain



```
const m=5;
function update(b) {
    // executed by miner M_i upon
    // reception of block b to append

   // let B_i be the blockchain at miner M_i
   if !check_validity(b) { reject b; return; }
   B_i = B_i ∪ b;
   B_i'=get_main_branch()
   if b'∈ B_i' ∧ (∃ b_1,…,b_m ∈ B_i': b_1→b_2→…→b_m→b') {
       decide b'
}
```

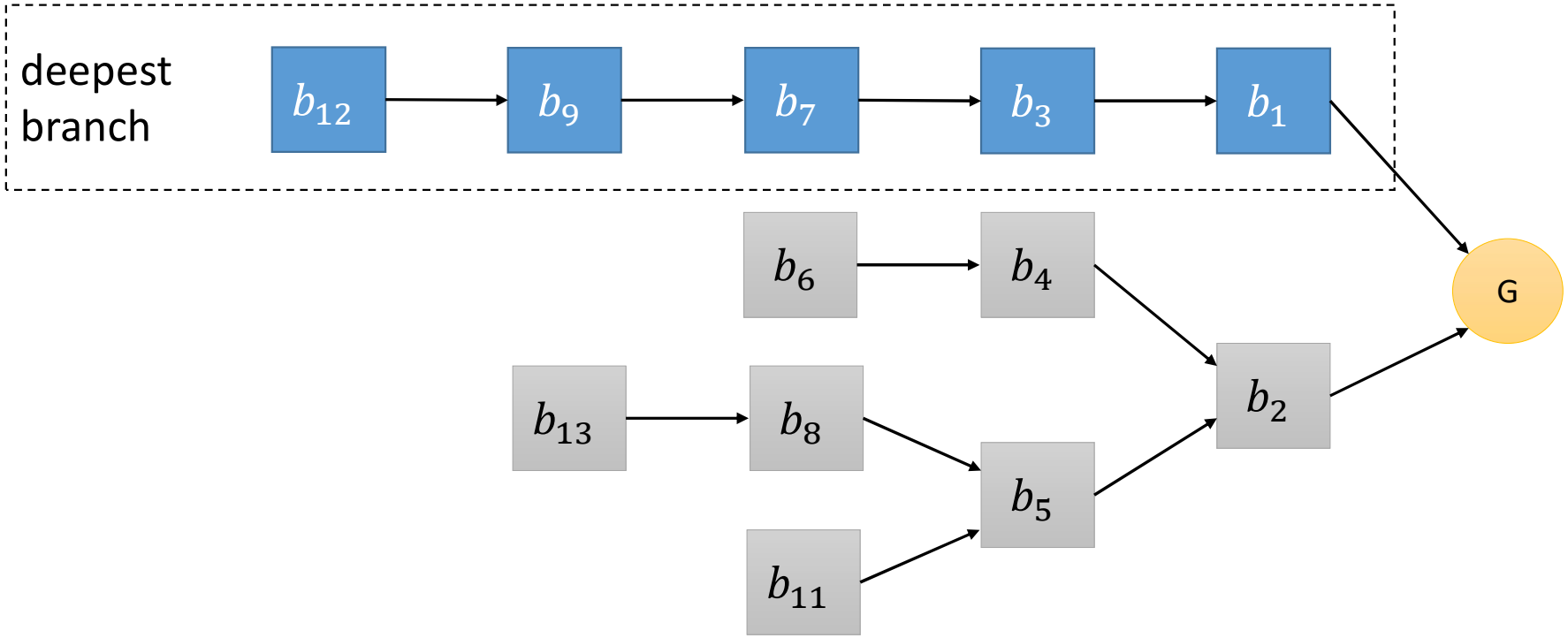
commits each block that has at least m other blocks appended to it

Consensus algorithm: append a received block (revisited)

```
function get_main_branch() {
    // let g be the genesis block
   b=g; D={g};
   while(children(b)≠∅) {// b has children
       b'=argmax c ∈ children(b){depth(c)}
       D = D ∪ b';
       b=b'
   }
   return(D);
}
function depth(b) {
   if (children(b)=∅) return 1;
   else return 1+max_{c∈children(b)}{depth(c)};
}
```

Main branch selection in Bitcoin

Deepest branch (Bitcoin)



Ethereum consensus algorithm

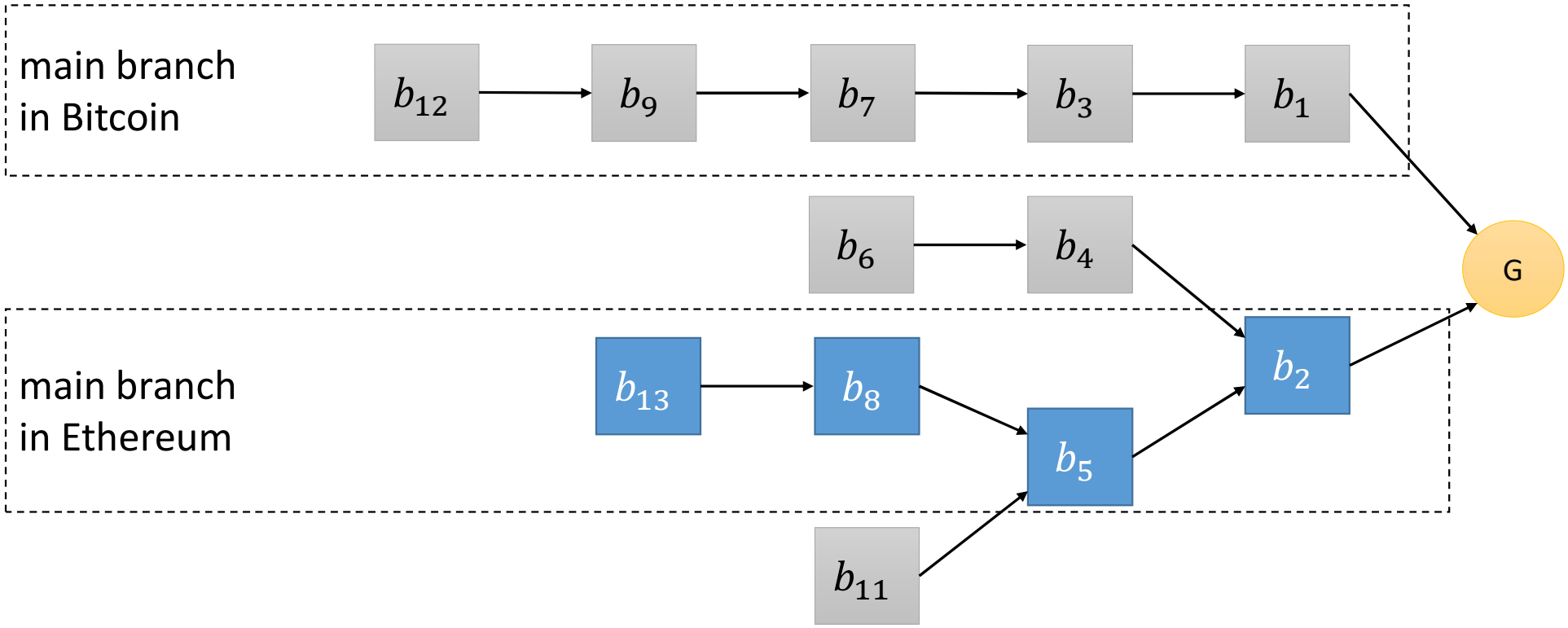
Ethereum uses a similar approach as Bitcoin, but:

- it produces a new block every 12/15 seconds (it was 10 minutes in Bitcoin)
- this favors transient forks as miners may likely propose new blocks without having heard about the last block yet
- to this purpose Ethereum used a variant of GHOST (Greedy Heaviest Observed Subtree)
  - a different method to compute the main branch
  - takes into account also non-committed blocks
  - (note however that this protocol has now changed)
- It also uses a different value of m (11) to commit blocks

```
function get_main_branch() {
    // let g be the genesis block
   b=g; D={g};
   while(children(b)≠∅) {// b has children
       b'=argmax c ∈ children(b){weight(c)}
       D = D ∪ b';
       b=b'
   }
   return(D);
}
function weigth(b) {
   if (children(b)=∅) return 1;
   else return 1+∑max_{c∈children(b)}weight(c);
}
```

Main branch selection in Ethereum

Main branch in Bitcoin and Ethereum



More about committing a block

- Due to the existence of forks, the blockchain should decide when to consider committed a block of a given index
  - committed blocks are called *decided*
  - in theory there cannot be consensus at all (the network is asynchronous)
  - in practice the blockchain looks for a reasonable guarantee
- This is done by parameter m ($m = 5$ for Bitcoin and $m = 11$ for Ethereum)
- Note however that these two parameters do not lead to the same probability of success, and their effect on performance is debated

Let B_i be the blockchain at node M_i in the system

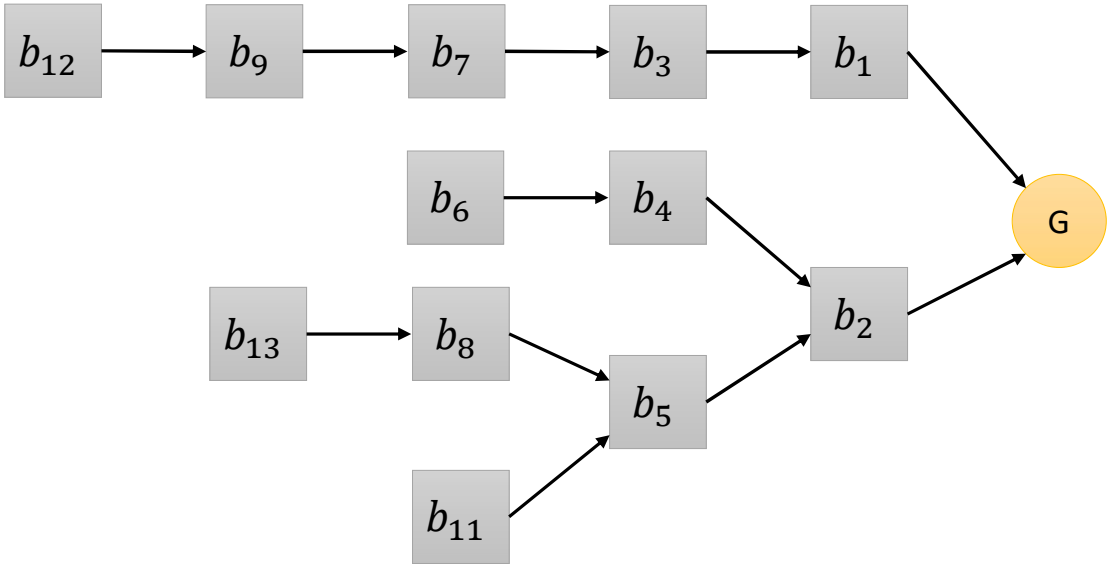
A transaction t_x is committed at M_i iff:

1. $t_x \in b_j$ for some block $b_j \in B_i$ that is in the main branch of B_i:
   $t_x \in b_j \wedge$
   $B' =$`get_main_branch()` $\wedge$
   $b_j \in B'$
2. There is a sequence of m blocks appended after b_j:
   $\exists b_1, \ldots, b_m \in B' : b_1 \longrightarrow b_2, \ldots, b_m \longrightarrow b_j$

More about committing a block

Question

what are the decided blocks in this blockchain assuming m=2?



Review questions

- I hence should wait for a block to be committed before assuming the transactions in it are valid. But is there still a possibility that I'm wrong and those transactions will be declared invalid later?

Main branch selection…

- This consensus mechanism follows a pragmatic approach, but has several drawbacks:
  - many miners would have spent resources uselessly to create blocks for dead branches
  - when a deeper branch emerges after a fork, all miners on dead branches will have to update their own blockchains by importing the blocks of the deeper branch
  - users that have executed transactions inserted in blocks on the dead branches will have to repeat their transactions again


Vulnerabilities of blockchains

Unsafety of PoW

- the randomized consensus adopted in PoW can result in violations of the safety property
- that is, the blockchain can decide for different, conflicting transactions, thus leading to potential double spending
- a solution against this problem could act on the tradeoff between termination and agreement
  - by taking into account network delays, variations in mining power
- but the current solutions are not flexible enough

Unsafety of PoW

- existing solutions adopt fixed parameters
  - like waiting for a fixed number m of blocks to be mined in order to reach termination.
- … but byzantine faults are subtle, they can vary delays and mining power to double spend in Bitcoin and Ethereum
  - some attacks had been successful and resulted in significant financial losses.
- Higher risks with consortium and private blockchains
  - due to misconfiguration of the blockchain

Attacks against Bitcoin

- Traditional attacks consist of waiting for some action (like shipping goods) in response to a transaction, and then discard the transaction from the main branch.
- As the transaction is revoked, the issuer of the transaction can reuse the coins of the transaction in another one.
- As the external action cannot be revoked, the second transaction appears as a ‘‘double spending’’.
  - if you feel lucky you may ask to have back the shipped goods or to reply the transaction…

Attacks against Bitcoin

Finney's attack is such an attack in the basic form:

1. Produce a "solo" block with a transaction that sends coins to yourself (but do not broadcast it yet)
2. Issue a transaction that double-spend bitcoins with a merchant
3. When the goods are delivered in exchange of the coins, broadcast the solo block to override the payment

vector76 attack is another one:

1. produce a "solo" block b_1 after a block b_0 , where b_1 contains a transaction to a merchant to purchase goods.
2. once another block b'_1 is mined after b_0, quickly send b_1 to the merchant for an external action to be taken.
3. If b'_1 is accepted by the blockchain, b_1 will be discarded and its coins will be available for another transaction

Attacks against Bitcoin

the previous attacks occur before a block is committed. However, the attack may become harder if done after…

Rosenfeld's attack:

1. Issue first a transaction to a merchant in a block b.
2. now the merchant waits for the block b to be committed (waits for m blocks to be appended after b), before taking an external action (e.g. ship goods)
3. In the mean time, build a "solo" blocks long branch, this will make block b to be discarded in the end

- The success depends on the value of m (a larger makes this attack more difficult) and on the mining power of the attacker

Attacks against Bitcoin

In any case, if the attacker has more mining power than the rest of the network, the attack will succeed regardless the value of m.

- this is called the majority hashrate attack or 51-percent attack

- The attacker may also incentivize other miners to make a coalition (to add to its long branch) to get more than half of the total mining power.

Selfish mining

- Selfish miners
  - keep their mined blocks without broadcasting
  - disclose their private branch only if some requirements are satisfied
  - as the private branch is longer than the current public chain, it would be admitted by all miners.
  - selfish miners would get more revenues
  - honest miners waste their resources
- Even less than 25% of the computational power may be successful to let selfish miners gain more revenues than honest miners

Attacks against Bitcoin

Studies on attack on block propagation delays have shown a major limit of Bitcoin:

- delaying propagation of blocks can waste the computational effort of correct processes by letting them mine blocks unnecessarily at the same index of the chain.
- In this case, the attacker does not need more mining power than the correct miners
- but simply needs to expand its local blockchain faster than the growth of the longest branch of the correct blockchain.

That’s why Ethereum introduced the GHOST protocol…

Blockchain anomaly in Ethereum

- The Blockchain anomaly prevents someone from executing dependent transactions
  - For example: ‘‘Bob transfers some coins to Carole only if it received coins from Alice’’
  - this anomaly allows an attacker to double spend.

- The anomaly occurs when the network has very long delays in messages delivery
- Two miners may proceed agreeing on different branches containing more than m blocks each
- When messages get finally delivered, the results of the disagreement creates inconsistencies
  - like the reordering or deletion of transactions from previously decided blocks

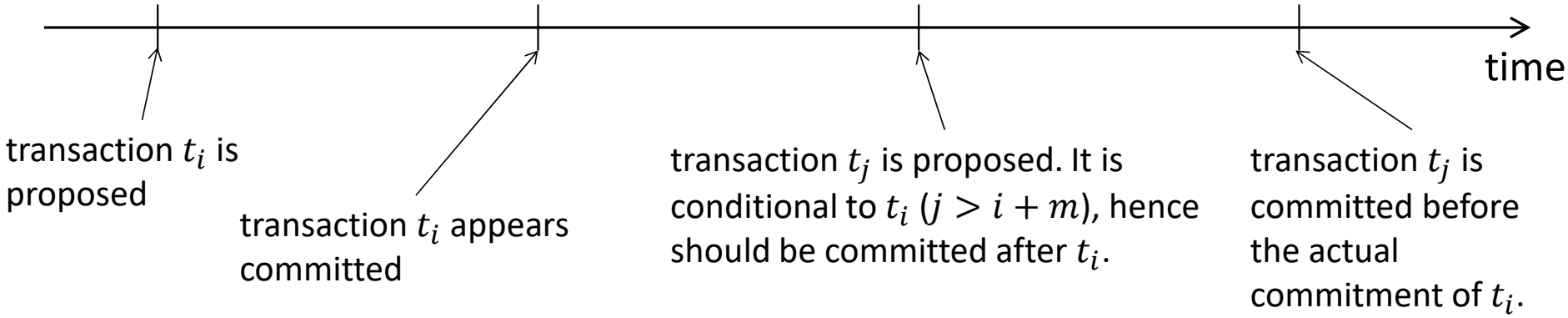
Blockchain anomaly

Two transactions are dependent: t_j should occur after commitment of t_i:

- a user first proposes t_i, then, once committed, another user proposes t_j.

However, these processes get notified of another branch of committed transactions, and they decide to reorganize the branch to resolve the fork.

- the reorganization removes the committed transaction t_i which is postponed.
- later, the transaction t_j is successfully committed in a block before that of t_i.



Blockchain anomaly in Ethereum

- this scenario is counterintuitive for Ethereum users:
- a transaction may not persist even if it is committed
- However, in a scenario of a large public blockchain this scenario is rather unlikely
  - It is feasible mostly in private blockchains

Privacy leakage in blockchains

- Users transact with their private key and public key without any real identity exposure.
  - To some extent this preserves a certain degree of users' privacy
- However, all transactions and balances for each public key are publicly visible
  - This may lead to inference attacks, by analyzing the parties involved in common transactions
  - Some recent studies have shown that a user's Bitcoin transactions can be linked to reveal user's information.
- Hence, blockchains do not guarantee the so-called *transactional privacy*
  - There are ongoing studies to propose fully anonymous blockchains

States and Transactions

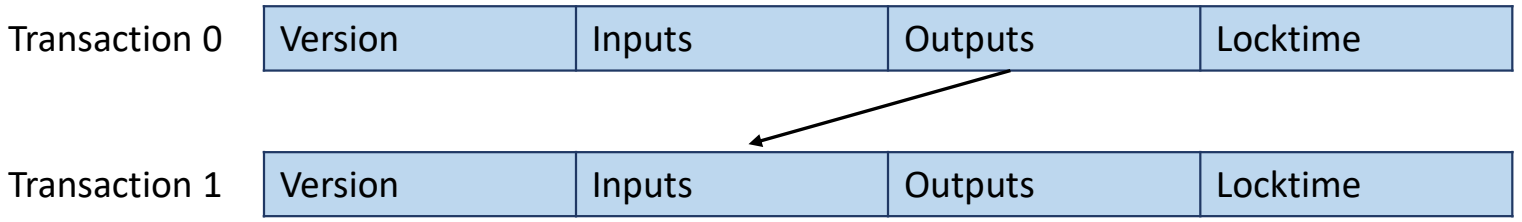
In Bitcoin and Ethereum

Bitcoin & Ethereum

- Both Ethereum and Bitcoin can be viewed as a transaction-based machine
  - characterized by a global state
  - each transaction changes the global state
  - transactions should be valid (as we have already seen)

Bitcoin transactions

- A transaction
  - version: 4 bytes, encodes the version of the blockchain in which the transaction is validated (to let future extensions)
  - Details of input, output and Locktime (see next slides)
- Each transaction spends the output of a previous transaction
- Pending outputs, that are not already spent, are called *Unspent Transaction Output (UTXO)*
  - All UTXO referable to a given address (and then a user) are the account of that user, the bitcoins he possesses



Bitcoin transactions

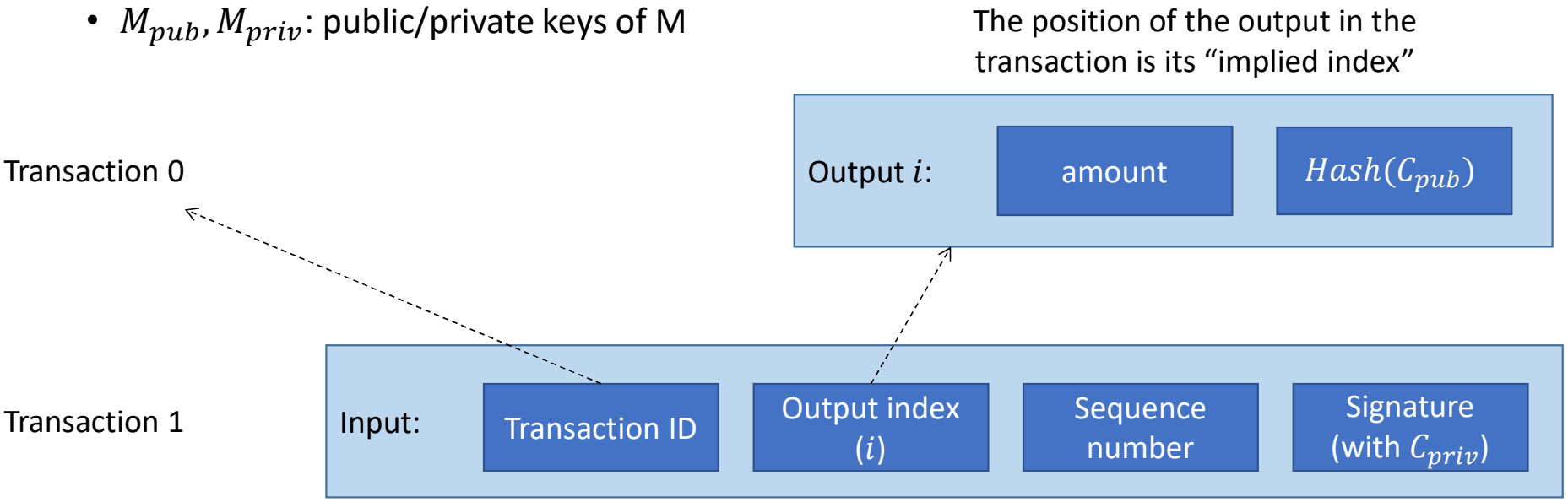
- a user does not own properly cryptocurrency, but he just owns the output of the transactions, which is called the UTXO
- with UTXO the currency cannot be spent partially but only entirely
- thus, a user can spend a part of his account by making two transactions:
  - one, of the amount established, to the merchant
  - one, of the remaining amount, to himself

Bitcoin transactions

- Locktime indicates the earliest time a transaction can be added to the block chain.
  - it allows signers to create time-locked transactions which will only become valid in the future
  - gives the signers a chance to change their minds.
  - it’s not fine grained, and can be set at up to two hours of current time
  - Locktime can be disabled

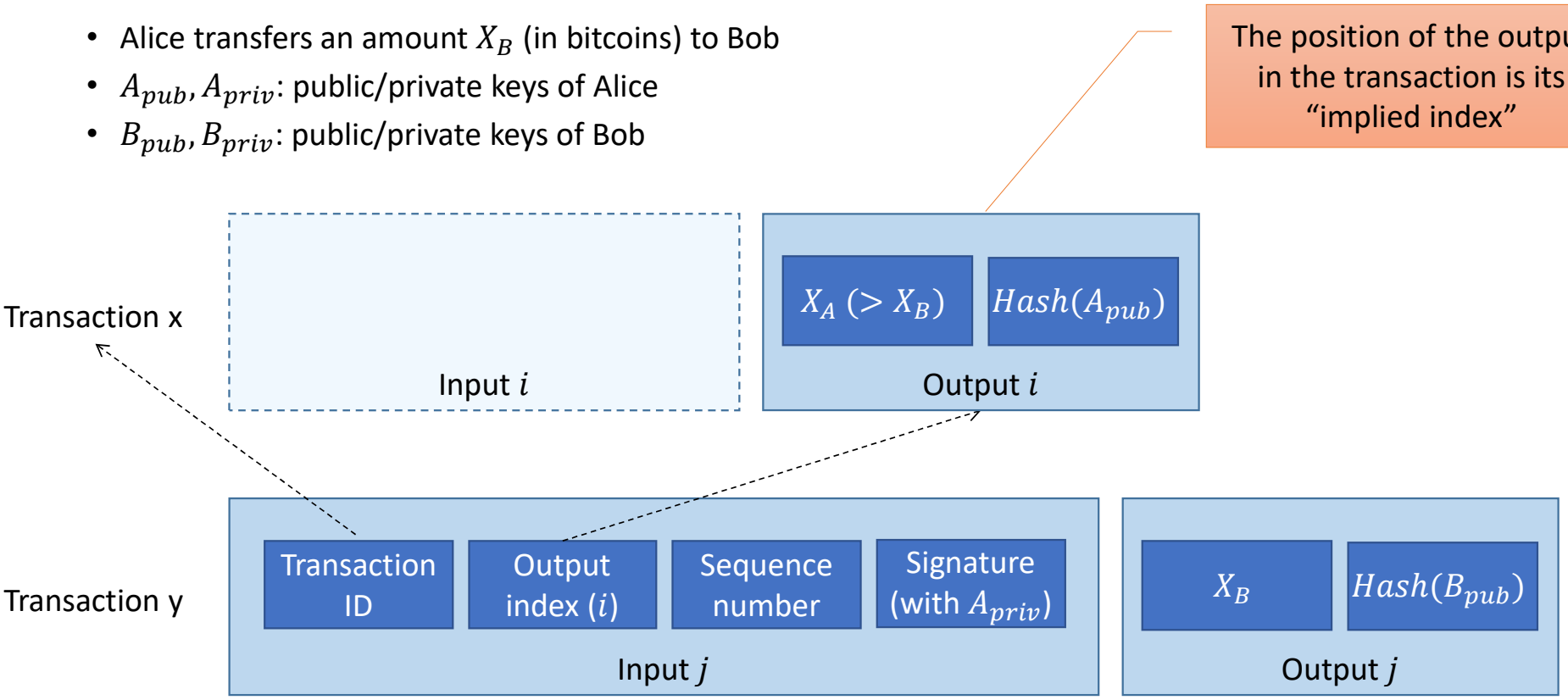
Bitcoin transactions

- A client C transfers value v to a merchant M
- C_{pub}, C_{priv}: public/private keys of C
- M_{pub}, M_{priv}: public/private keys of M



Bitcoin transactions

- Alice transfers an amount X_B (in bitcoins) to Bob
- A_{pub}, A_{priv}: public/private keys of Alice
- B_{pub}, B_{priv}: public/private keys of Bob



Bitcoin scripts

- The verification of the transactions in bitcoin is made by the execution of scripts
- Scripts are specified in an ad-hoc language
  - non Turing-complete, without loops or go-to statements
- Bitcoin already defines a number of scripts to manage some "standard" transactions
  - the example in the previous slides is a standard one (for which a script already exists)
  - called "Pay to public key hash" transaction

States and transactions in Ethereum

States and transactions in Ethereum can be modeled as:

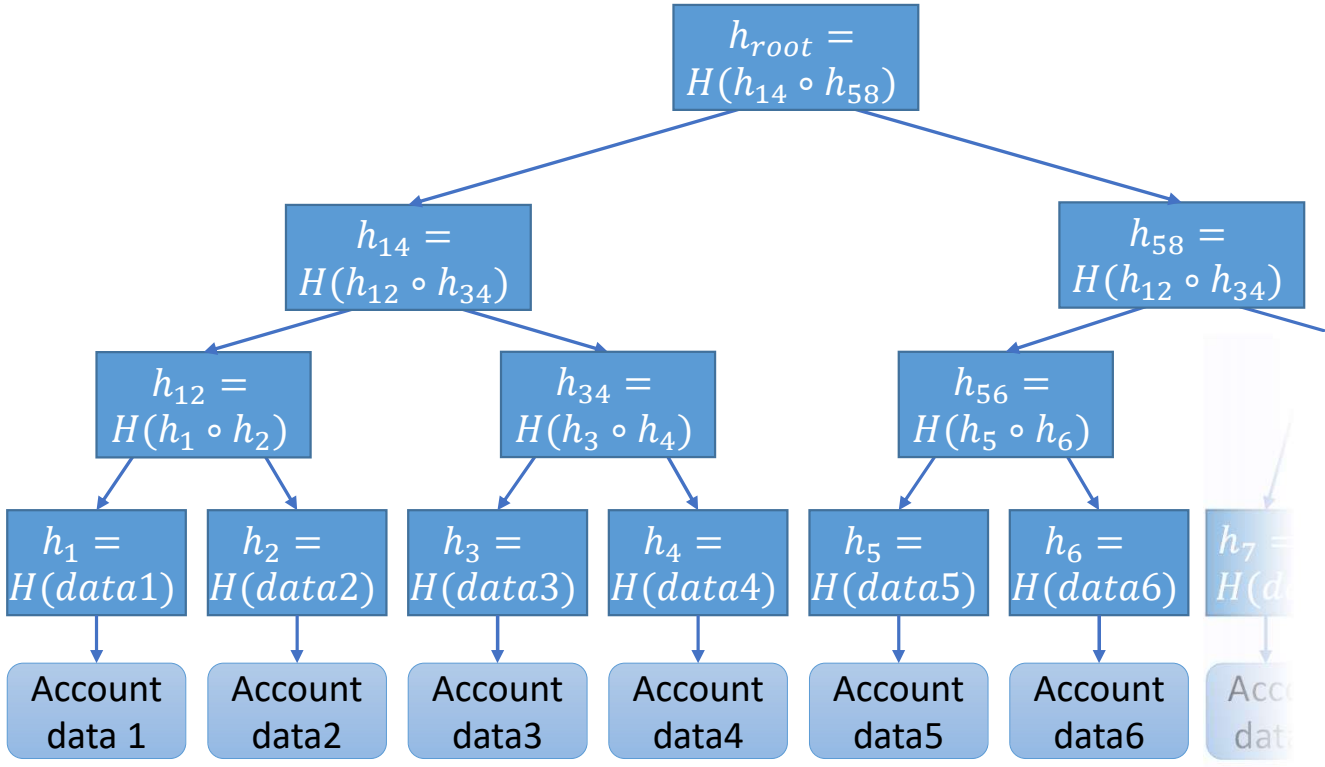
$$\sigma_{t+1} = \Upsilon(\sigma_t, T)$$

- σ_t is the global state of Ethereum at time t
- Υ is the transition function, that allows arbitrary computation
- T is the transaction

- Transactions are grouped in blocks to form the blockchain
  - For example, to scan blocks contents: https://etherscan.io/blocks
- Ethereum defines a "value" to incentivize the mining activity
  - the Ether, defined ad 10^{18} Wei

Accounts and state

- Users in Ethereum have an account
- The global state of Ethereum:
  - Account states
  - mapping between account addresses and account states
- The global state is kept in a special version of a *Merkle Tree* data structure
  - Called Merkle Patricia Tree, or *trie*



An example of a Merkle Tree

Accounts

- Each account is identified by a 160-bits identifiers and contains:
  - **nonce** – the number of transactions sent by this address or, if it the account is associated with code, the number of contracts made
  - **balance** – a number of Wei owned by this address
  - **storageRoot** – the hash of the trie that stores the content of the account
  - **codeHash** – the hash of the code (if it exists) that gets executed when this account receives a call.
- if the account is associated to a code it is a *contract*.
- otherwise it is a simple account (and the codeHash field is null).

Transactions

- Transactions are constructed by entities external to Ethereum
  - humans or software tools on their behalf
- Two types of transactions:
  - make a call to an account
  - create a new account (contract creation)

Transaction structure:

- **nonce** – number of transactions sent by the sender
- **gasPrice** – number of Wei to be paid for the computation cost incurred in the execution of the transaction (this cost is measured in *gas*)
- **gasLimit** – max. amount of gas to be used in executing the transaction (if execution exceeds this limit the transaction is discarded
- **to** – the 160-bit address of the account recipient of the call (it is null if the transaction is a contract creation)
- **value** – number of Wei to be transferred to the recipient or to the new account
- **sign** – signature of the sender

Transactions

- A call transaction may also contain data
  - Just an array of input data for the call
- A contract creation transaction may include an init code
  - Executed just once at the creation of the account

Gas and payment

- At each transaction Ethereum runs a script:
  - The initialization code in case of creation of a new contract
  - The code associated to the account recipient of a call transaction
  - *no script executed to call an account without code
- The script is formalized in terms of a Turing-complete language
  - That is, it can do anything a computer can do, even play a videogame…
- However, it makes no sense to run general purpose applications on Ethereum
  - It is extremely costly
  - Ethereum may become too busy to do actual work

Gas and payment

- every transaction specifies a gasLimit.
  - amount of gas which is implicitly purchased from the sender's account balance.
  - The transaction also specify the offered price for the gas (gasPrice)
  - the transaction is invalid if the account balance cannot make such a purchase.
- unused gas at the end of the transaction is refunded
- gas that is not refunded is delivered to the beneficiary address (typically of the miner).

Gas and payment

- transactions can specify any gasPrice, but miners may ignore the transaction if the gasPrice is not enough
  - A higher gas price on a transaction will cost the sender more in terms of Ether
  - … and deliver a greater value to the miner
  - Thus the transaction will more likely be selected for inclusion by more miners.
- miners may advertise the gas price they accept
- in the end the gasPrice is subject to a demand/offer model

Ethereum vs Bitcoin

Feature	Bitcoin	Ethereum
Blockchain	Yes	Yes
E-cash p2p payment system	Yes	Yes
Smart contract	No	Yes
Consensus algorithm	PoW	PoW
Ground data structure	Merkle Tree	Modified Merkle Patricia Tree
Transaction scheme model	UTXO	Account state
Turing complete	No	Yes
Block validation time	10 minutes	15 seconds

Review questions

- What does UTXO mean?
- What are the information included in a transaction in Bitcoin?
- What types of transactions are possible in Ethereum?
- What is the purpose of «Gas» in Ethereum?

Consensus with Proof of Stake

A critic to Proof of Work

- The mechanism of proof of work is computationally intensive
- Under this respect it is not really «green» and sustainable
- For this reason some blockchains are adopting (migrating) to Proof of Stake
  - Ethereum is a notable case

Proof of Work and Proof of Stake

- Often «Proof of Work» and «Proof of Stake» are referred to as consensus mechanisms
- However they are only part of it. The consensus mechanism includes combines «proofs» with other methods to decide what is the last block of the chain (hence including also branch selection)

Proof of Stake in Ethereum

In proof of stake, the validator (formely a miner) explicitly stake a capital into its Ethereum smart contract

- The stake is in the Ethereum cryptocurrency (the Ether – ETH)

This value can be lost all or in part if the validator acts disonestly

- for example, if the validator sends multiple blocks when it ought to send only one …
- … or if it sends conflicting attestations… (see below what an attestation is)

Specifically:

- Each validator may occasionally create and propagate a new block (whose production is rewarded in ETH)
- The validators also check for the validity of the new blocks propagated over the network and send a vote (called an attestation) in favor of that block across the network.
- Once the block has been attested by sufficient validators it is added to the head of the chain
  - With the property that the block with a higher stake has higher chance to be selected as next block in the chain

Proof of Stake in Ethereum – block creation

Time is divided into slots (12 seconds)

At each slot one validator is randomly selected to be a block proposer

- To this purpose Ethereum uses a pseudo-random number generator known to everybody in the network
- In this slot the selected validator creates and sends a new block to other validators

In the same slot, a set of validators randomly chosen (a committee) is responsible to check the validity of the block and to send their attestation

- This reduces the overall network load to makes the protocol manageable

References

More detailed information about Proof of Stake in Ethereum can be found here:

- “Casper the Friendly Finality Gadget” - https://arxiv.org/pdf/1710.09437.pdf - Casper is the protocol used by Ethereum to decide which block to add to the top of the chain
- “Combining GHOST and Casper” - https://arxiv.org/pdf/2003.03052.pdf - The combination of Casper and Ghost (called Gasper) is the proof of stake mechanism of Ethereum. Ghost is the fork-choice rule based on the heaviest subtree that we have seen before.

Consortium blockchains

Refining the blockchain model

- Consortium-model for blockchains:
  - a consortium is a pre-selected set of participants
  - the consortium fully controls the consensus protocol of the blockchain
- Relaxes the requirements on distributed consensus
- Leads to simpler and manageable solutions
- … but it requires trust of other users in the consortium participants

Features of consortium blockchains

Permissioned:

- only a specified set of institutions can participate in the consensus of the consortium blockchain *(permissioned* participants)*.*
  - the consensus is not centralized anyway (there's no a single leader as in fully private blockchains)
  - alleviates the problem of having an uncontrollable amount of nodes wasting resources.
- other users can inspect the blockchain content and can issue transactions
- In contrast, Bitcoin and Ethereum are permissionless
  - any participant connected to Internet can join at any time and fully participate to the blockchain

Features of consortium blockchains

Global knowledge:

- Since membership is predetermined, it is (or may be) easy to know the exact list of the participants of the consortium.
  - Consequently, any participant that lags behind, simply needs to contact a majority or a quorum of participants to catch up with the most up-to-date system size n.
  - Moreover, this fixed list of participants naturally prevents an attacker from executing a Sybil attack by forging multiple identities it can control

Features of consortium blockchains

Bound on the number of failures:

- Since participants are known:
  - a malicious participant cannot convince the consortium to introduce a large number of fake identities in comparison to the consortium size.
  - new participants go through a detailed KYC (know-your-customer) process before getting the permission to join the consortium.
- it is realistic to assume malicious participants are limited
  - this simplifies the consensus

Failure model

- The failure model is still that of Byzantine failures
  - the participating institutions can have conflicting interests
  - the blockchain should protect from the possible misbehavior of a participant.
- However, since the expected number of failures is low, it is possible to adopt practical consensus algorithms that do not require PoW
- Still, consensus may require:
  - a leader election, although this conflicts with the inherent decentralization aim of blockchains
  - or other complex techniques to circumvent the impossibility result of byzantine agreement

Communication model

- The communication model is also the same as for public blockchains
  - participants may be located in different regions of the globe
  - they typically communicate through internet when issuing transactions.
  - the Internet is unpredictable and the delay of a message cannot be known in advance.
  - Internet is a shared infrastructure, it is not possible to predict congestions, traffic disruptions or other disasters

Consortium blockchains

- Examples are:
  - Ripple – the third largest digital currency
    - D. Schwartz, N. Youngs, A. Britto, The Ripple Consensus Protocol, (1) Ripple Labs Inc., 2014, URL https://ripple.com/files/ripple_consensus_whitepaper.pdf
  - Hyperledger fabric – an industry-wide collaborative effort to develop an open-source blockchain
    - C. Cachin, Architecture of the hyperledger blockchain fabric, in: Workshop on Distributed Cryptocurrencies and Consensus Ledgers, DCCL'16, 2016. https://www.zurich.ibm.com/dccl/papers/cachin_dccl.pdf
  - R3 consortium – supported by financial institutions
    - https://www.r3.com/

Review questions

- When do you think it is better to use a blockchain and when it is more convenient the use of a conventional database?
- Can you give two use cases that motivate the use of a blockchain and a conventional DB?

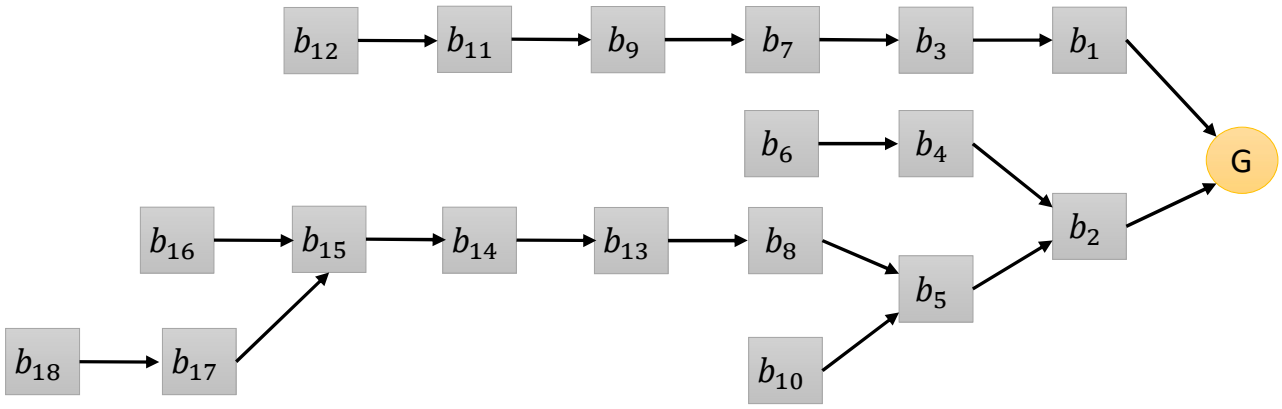
Summary

- A general introduction to blockchain
- How to achieve consensus with proof of work
- Failures of proof of work
- Vulnerabilities of proof of work
- Bitcoin and Ethereum
- Consortium blockchains
- Blockchain and IoT

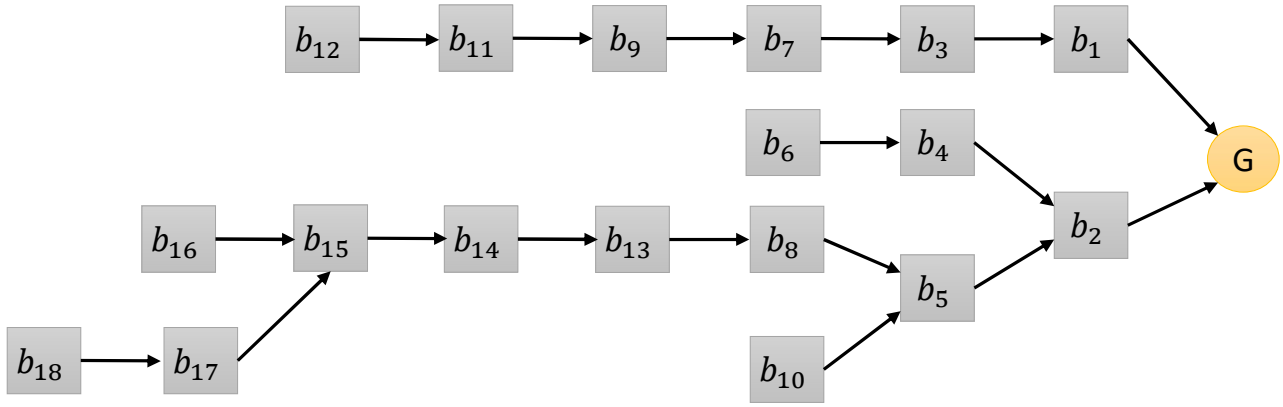
Exercise 1

Let's assume that, due to several forks, the Bitcoin blockchain takes the form in the figure.

1. What is the main branch?
2. To which block a new block should be appended?
3. What are the blocks that will be committed for m in {5, 7, 9}?



Solution

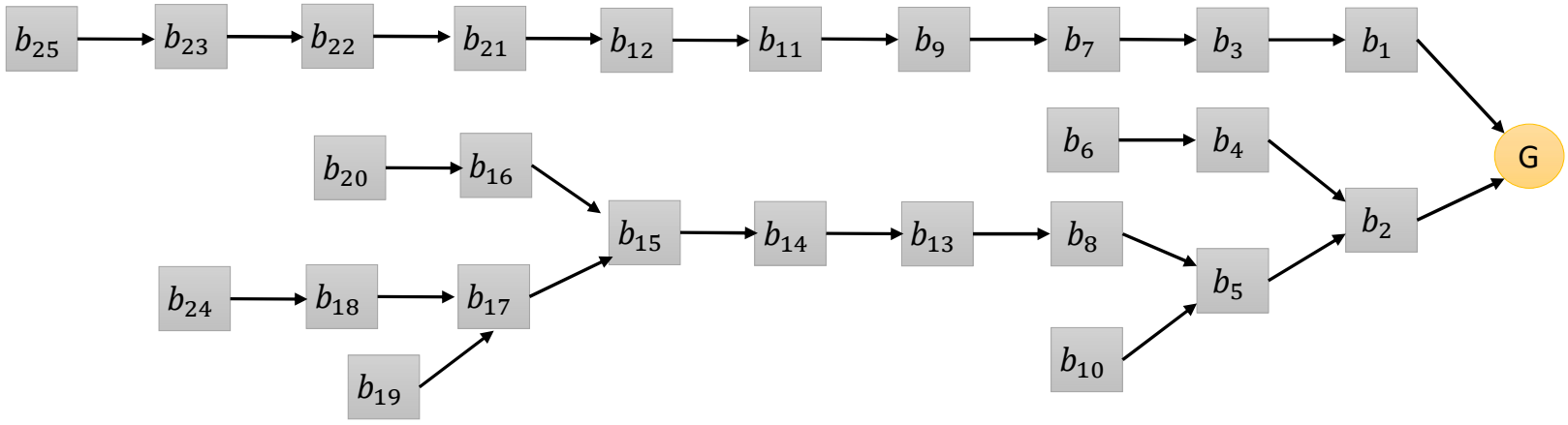


1. What is the main branch? _____
2. To which block the next blocks will be appended? _____
3. What are the blocks that will be committed if m=5? _____
4. What are the blocks that will be committed if m=7? _____
5. What are the blocks that will be committed if m=9? _____

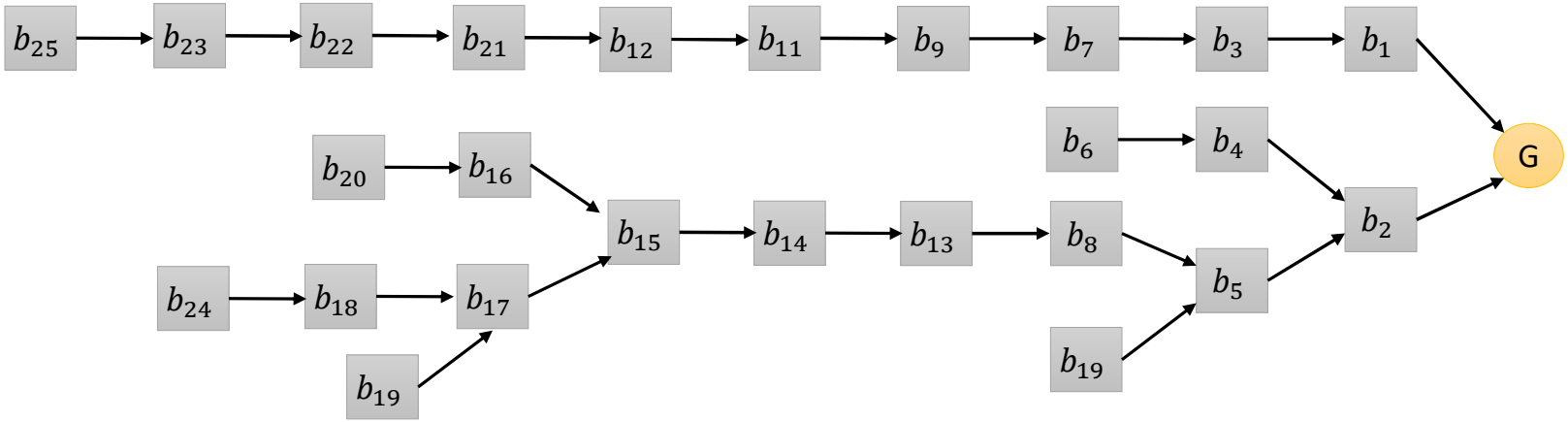
Exercise 2

Let's assume that, due to several forks, the Ethereum blockchain takes the form in the figure.

1. What is the main branch?
2. To which block a new block should be appended?
3. What are the blocks that will be committed for m in {5, 7, 9}?



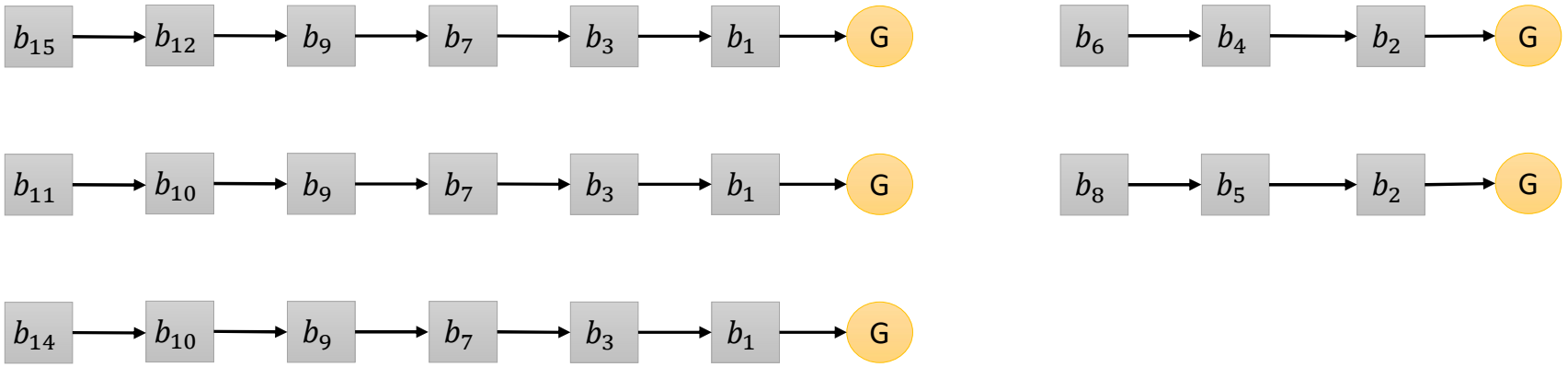
Solution



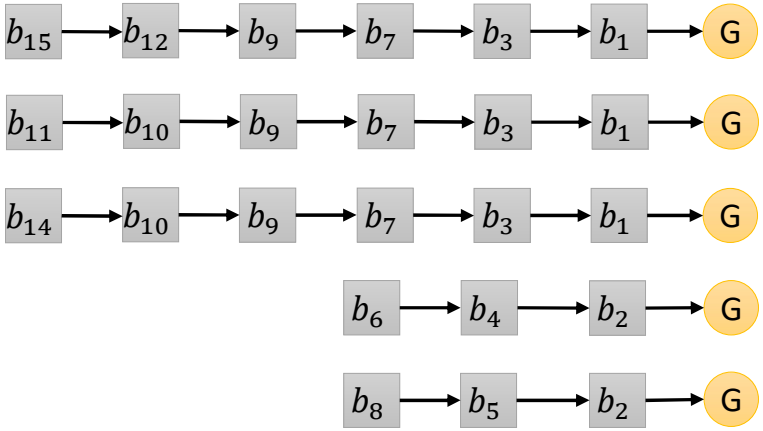
1. What is the main branch? _____
2. To which block the next blocks will be appended? _____
3. What are the blocks that will be committed if m=5? _____
4. What are the blocks that will be committed if m=7? _____
5. What are the blocks that will be committed if m=9? _____

Exercise 3

Let's assume that the local view of the blockchain at the miners is that depicted in the figure. Depict the global view of the blockchain.



Solution


b_{15} b_{12} b_9 b_7 b_3 b_1 G
b_{11} b_{10} b_9 b_7 b_3 b_1 G
b_{14} b_{10} b_9 b_7 b_3 b_1 G
b_6 b_4 b_2 G
b_8 b_5 b_2 G