# Lecture 5
# Part 2: Proof-of-Work and Other Types of Consensus

Yury Yanovich

Skoltech, Bitfury

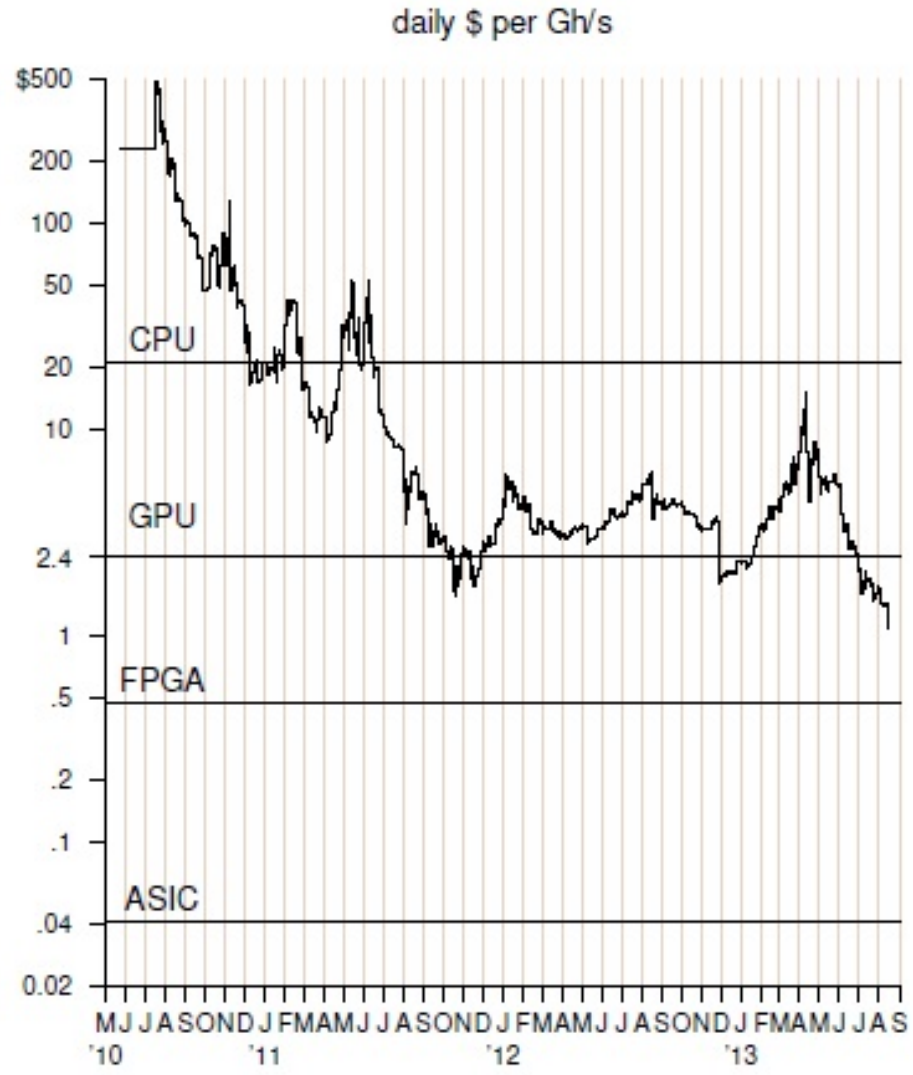
November 13, 2018

# Table of Contents

# Proof-of-Work History

- Dwork&Naor – Pricing via Processing or Combatting Junk Mail, 1992: function calculation to get access and functions properties
- Back – hashcash, 1997: $\texttt{SHA}(x)$ starts with $N$ zero bits. Find $x$
- Abadi&Burrows&Manasse&Wobber – Moderately Hard, Memory-bound Functions, 2005: memory-bound $\rightarrow$ Scrypt algorithm
- Nakamoto – Bitcoin: A Peer-to-Peer Electronic Cash System, 2008: one-CPU-one-vote and adaptive target value.

# Proof-of-Work Mining Hardware



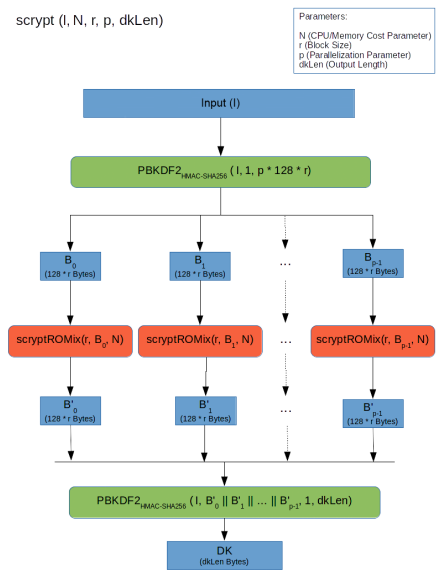

- FGPA: field-programmable gate array
- ASIC: application-specific integrated circuit.

# ASIC-resistance

- **Promotion:** every computer (theoretically) provides approximately similar hashrate, more people will join the mining $\rightarrow$ more protection
- **Decentralization:**
  - hardware manufacturers. The majority of ASICs are made on a few factories in China, and very few people possess the knowledge required to produce them
  - geographical location of the miners. If we do not consider pools, the versatile CPU-friendly algorithm is wider spread because everyone can afford to launch a miner.
- **High cost of 51% attack:** it is possible to design a specialized device that would solve a specific task more efficiently (time- or powerwise) than your average PC. Cost?

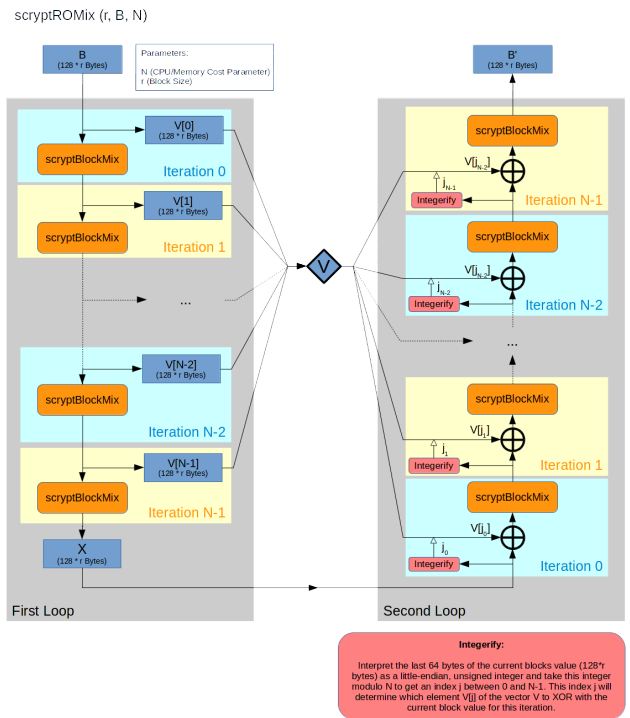
# Scrypt operating principle. Layer 1

1. Input data is processed through PBKDF2
2. It is divided into $p$ blocks, each processed by SMix function
3. The resultant data is pieced together and processed through PBKDF2 again.



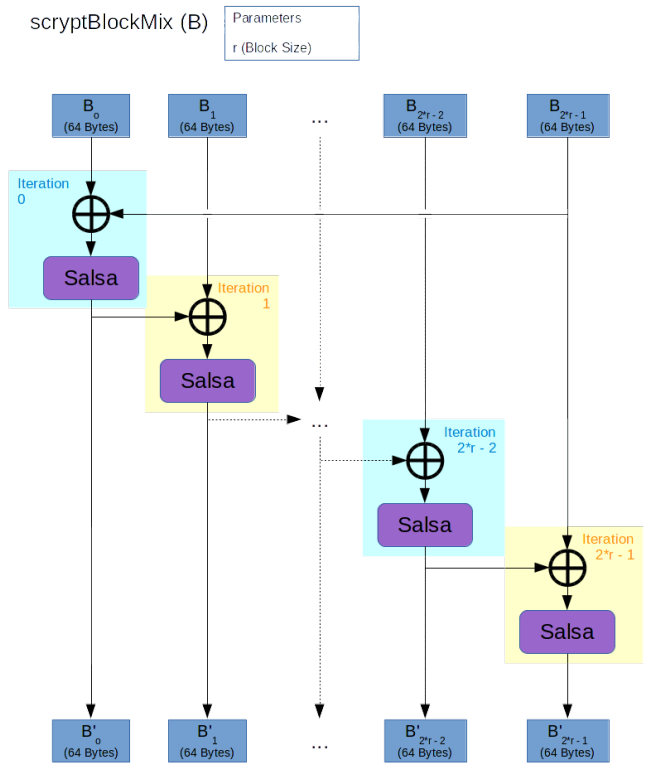
# Scrypt operating principle. Layer 2

1. Input block is enhanced to an array of $N$ blocks by sequential processing by BlockMix pseudo-random function
2. Block $X$ resultant from the last component is interpreted as an integer $j$ and component $V_j$ from the array is read
3. $X = \texttt{BlockMix}(X \texttt{xor} V_j)$
4. Repeat steps 2-3 $N$ times
5. The result is the current value of $X$ block.



# Scrypt operating principle. Layer 3

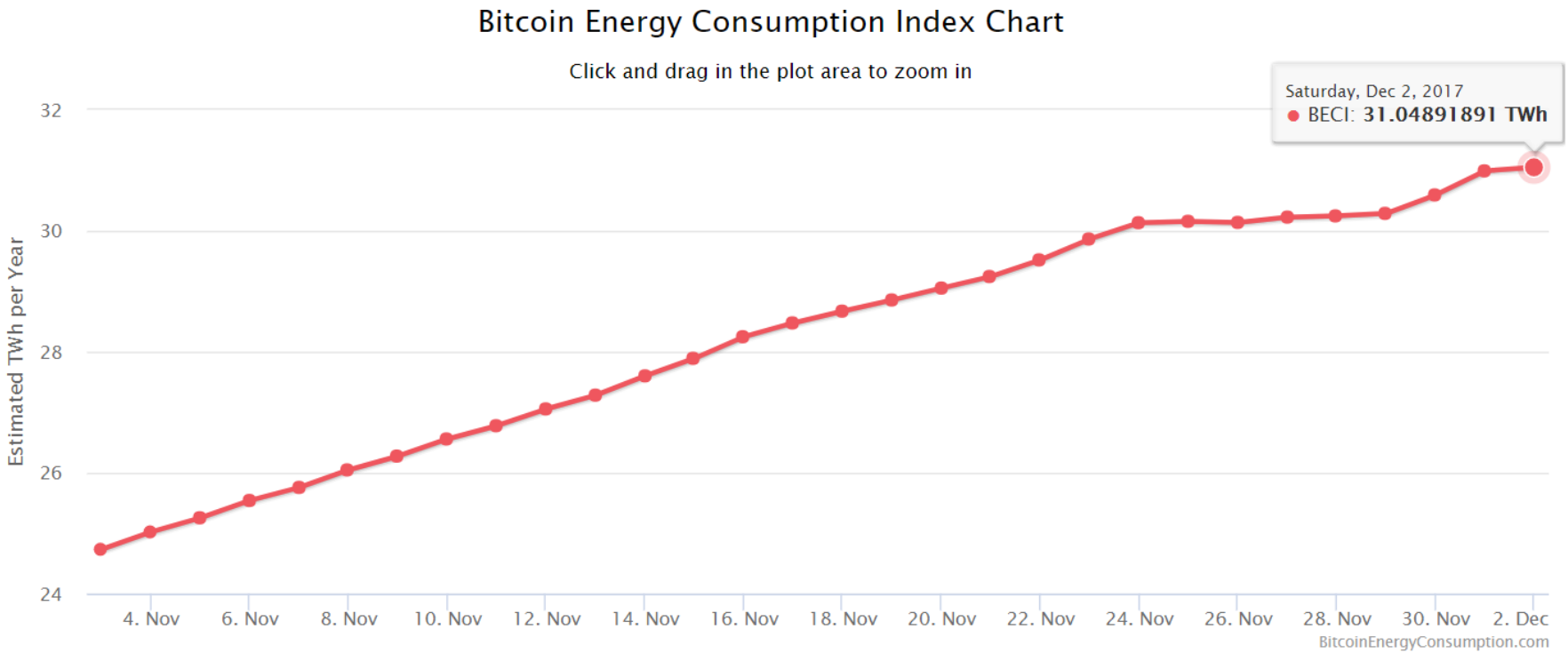
1. Input is divided into $2r$ blocks
2. Each block is xor-ed with the previous one and hashed by $H$ function
3. The result is displayed in displaced order.



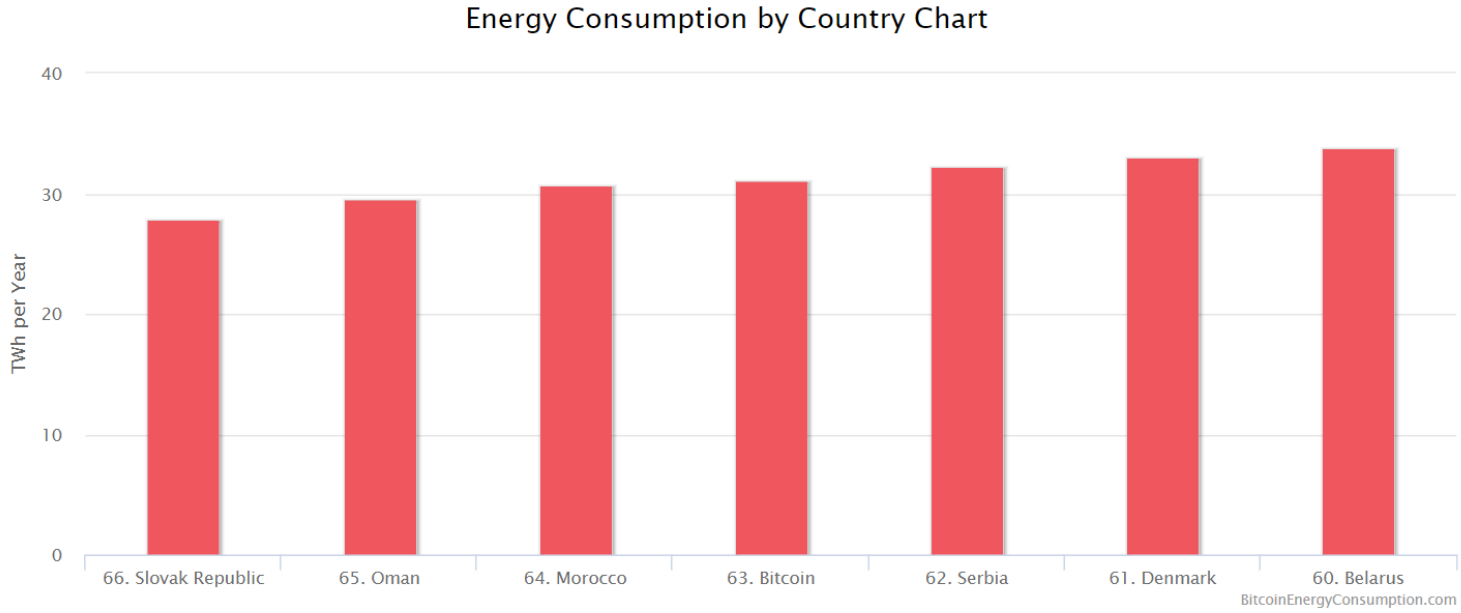
# Litecoin and other Scrypt-based Cryptocurrencies

- Litecoin introduced: 7 October 2011
- 100+ forks
  - Sifcoin, 2013: sequential hashing with 6 different hash functions
  - Momentum, 2013: to sign data $D$
    1. get $H = \texttt{Hash}(D)$, where $\texttt{Hash}()$ is some cryptographic hash-function.
    2. find such values $A$ and $B$ that $\texttt{BirthdayHash}(A + H) = \texttt{BirthdayHash}(B + H)$, with $\texttt{BirthdayHash}()$ being a memory-bound function, as scrypt.
    3. Now, if $\texttt{Hash}(H + A + B) < \texttt{TargetDifficulty}$, than it is finished. Victory! Otherwise, go back to step 2.

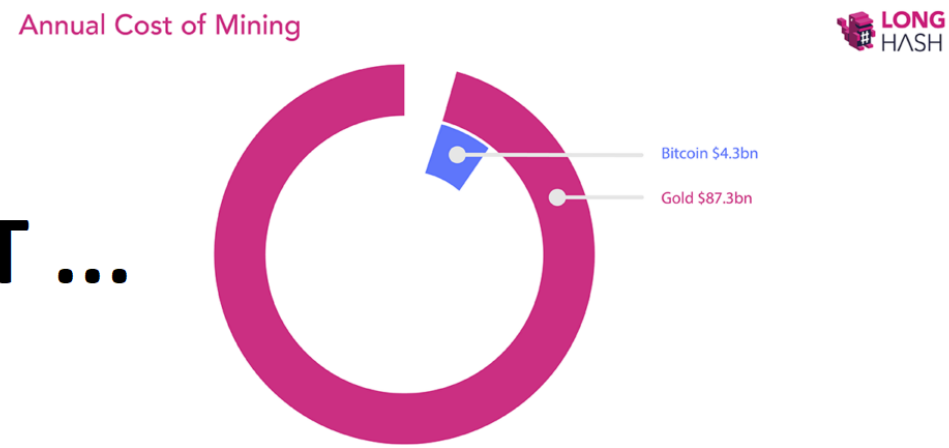
# Energy Consumption



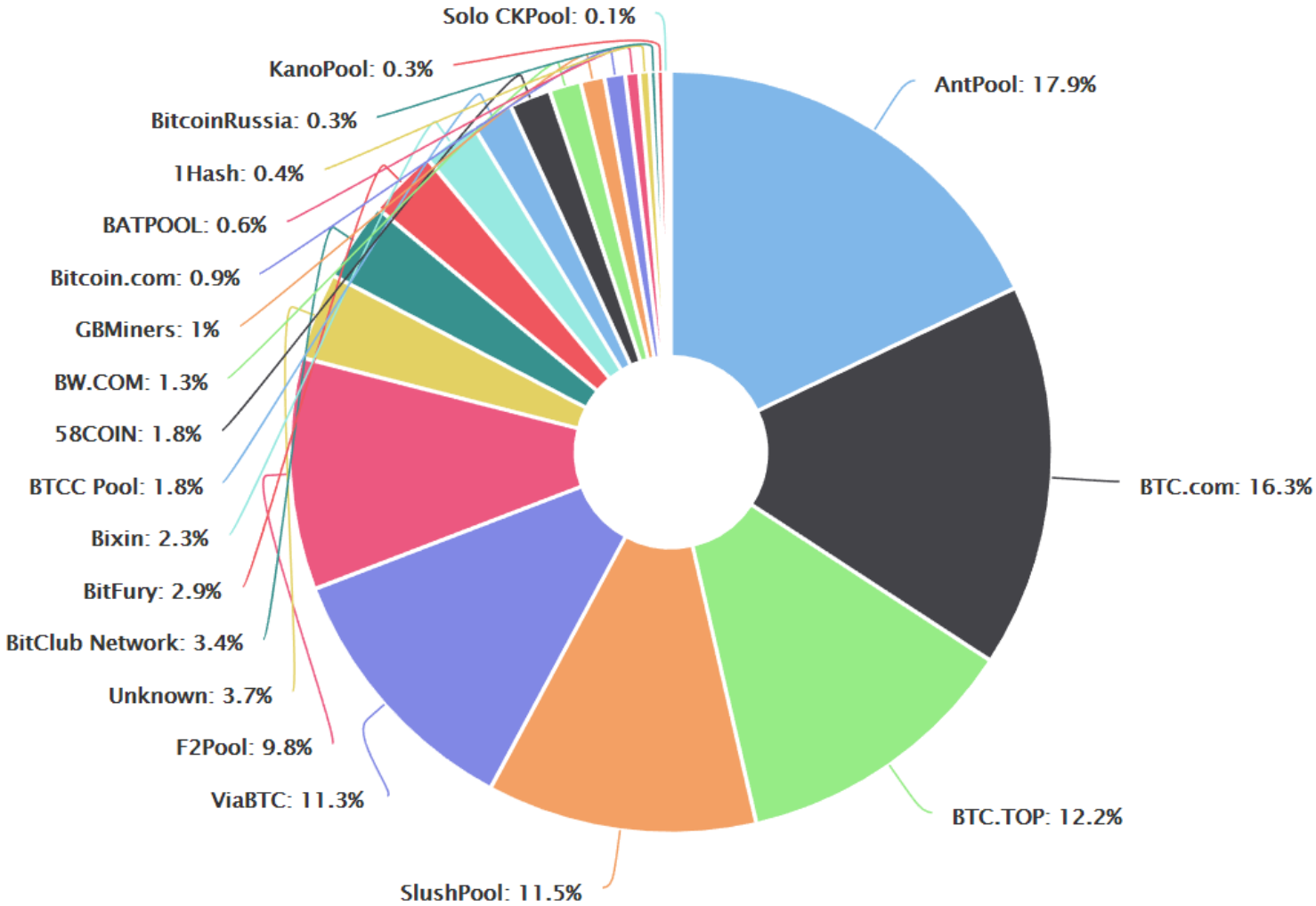
# Energy Consumption (2)



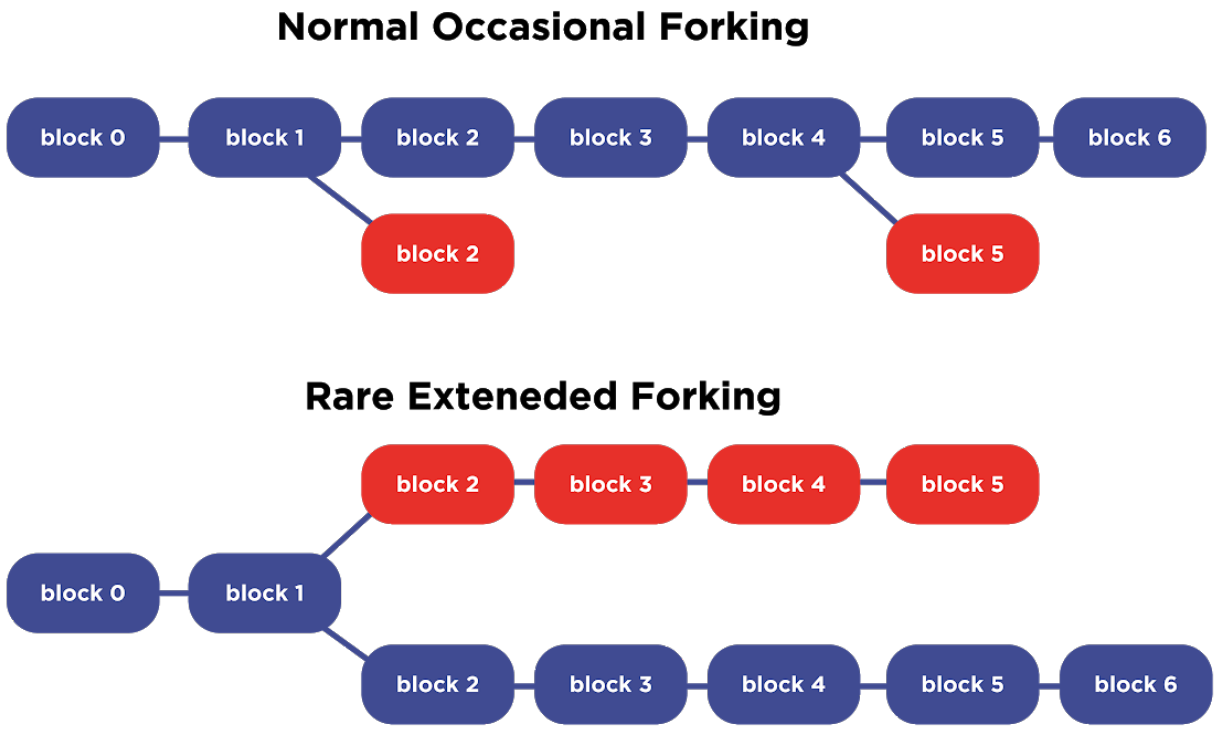

**BUT ...**



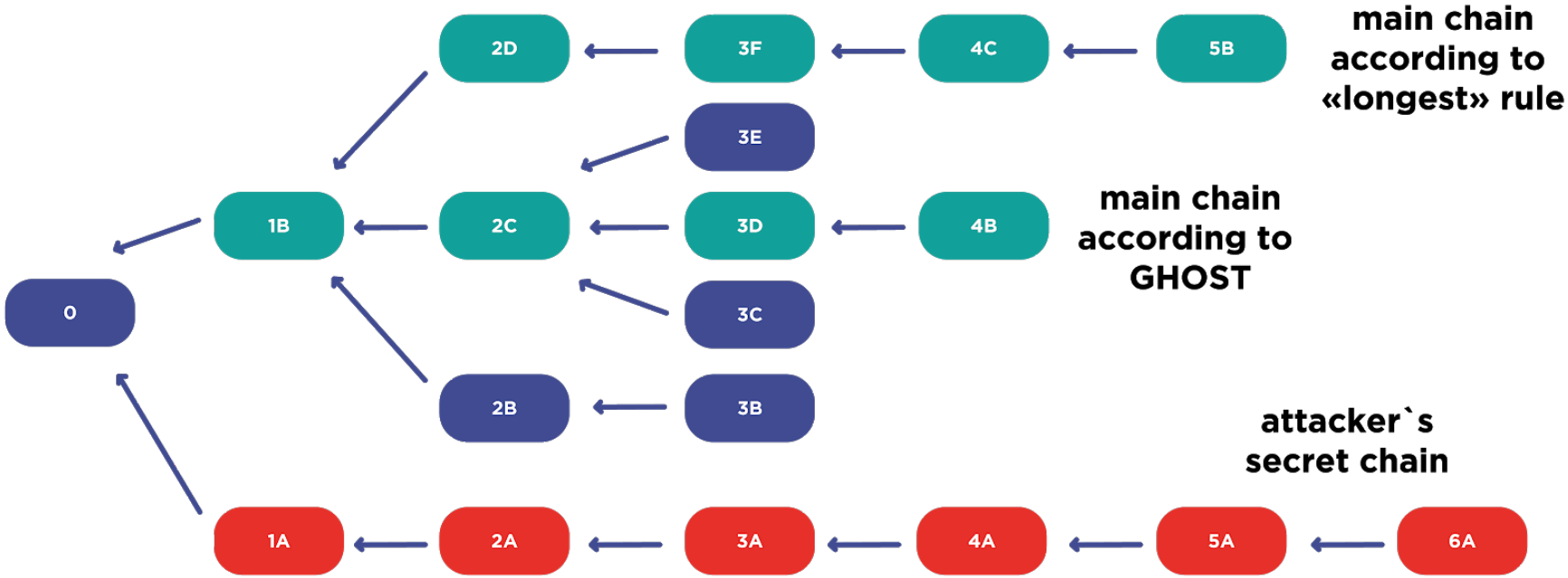
# Hashrate Distribution



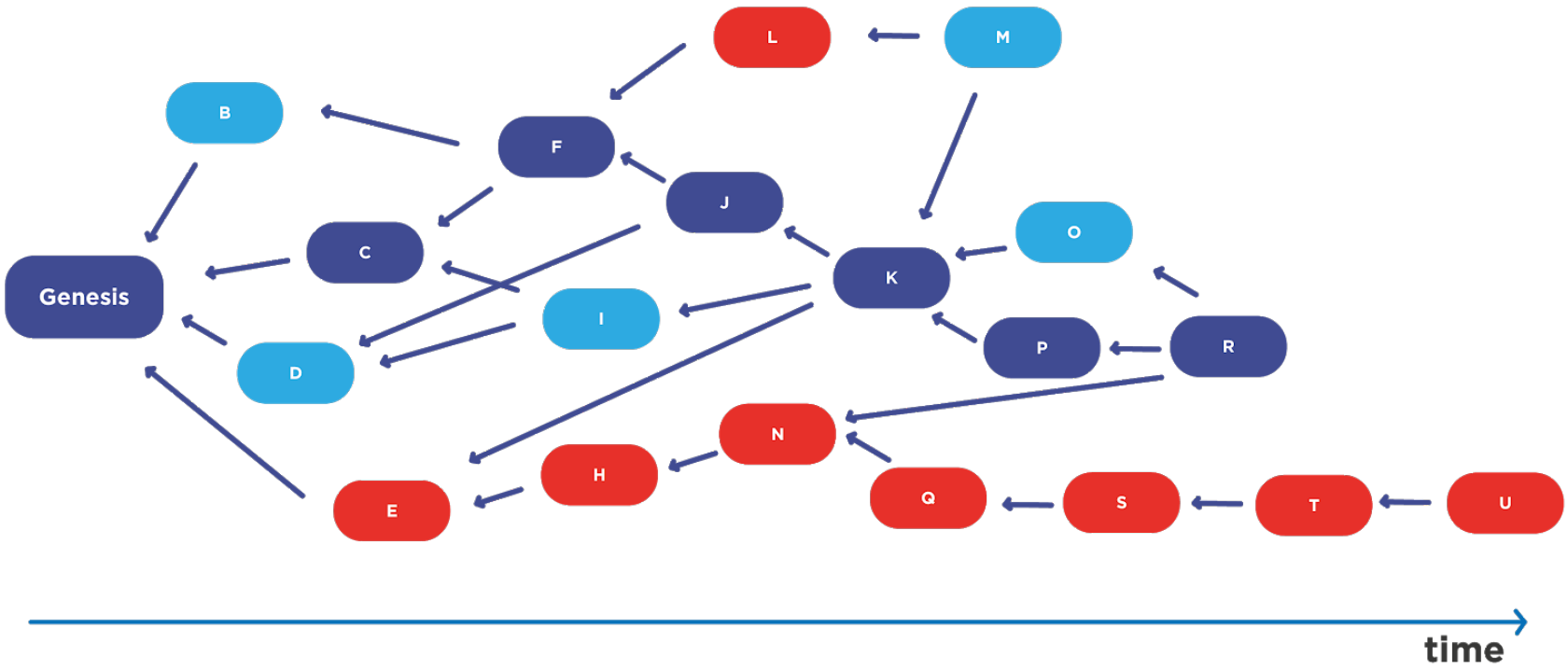
# Forking


Normal Occasional Forking
block 0
block 1
block 2
block 3
block 4
block 5
block 6
block 2
block 5
Rare Exteneded Forking
block 2
block 3
block 4
block 5
block 0
block 1
block 2
block 3
block 4
block 5
block 6

# 51% and GHOST


2D
3F
4C
5B
main chain according to «longest» rule
3E
1B
2C
3D
4B
main chain according to GHOST
0
3C
2B
3B
attacker`s secret chain
1A
2A
3A
4A
5A
6A

# SPECTRE and PHANTOM



# GHOST, SPECTRE and PHANTOM Summary

Pros

- higher performance
- higher level of decentralization.

Cons

- large amount of stored information.

# Consensus Problem

A fundamental problem in distributed computing is how to achieve system reliability in the presence of faulty processes – to get consensus.

The consensus problem requires agreement among a number of processes for a single data value, for example, about a new bit value.

The correct consensus classically means that three conditions hold for every execution of the algorithm

- *termination*: eventually each correct process sets its decision variable
- *agreement*: the decision value of all correct processes is the same
- *integrity*: if the correct processes all proposed the same value, then any correct process in the decided state has chosen that value.

# Consensus for Blockchains

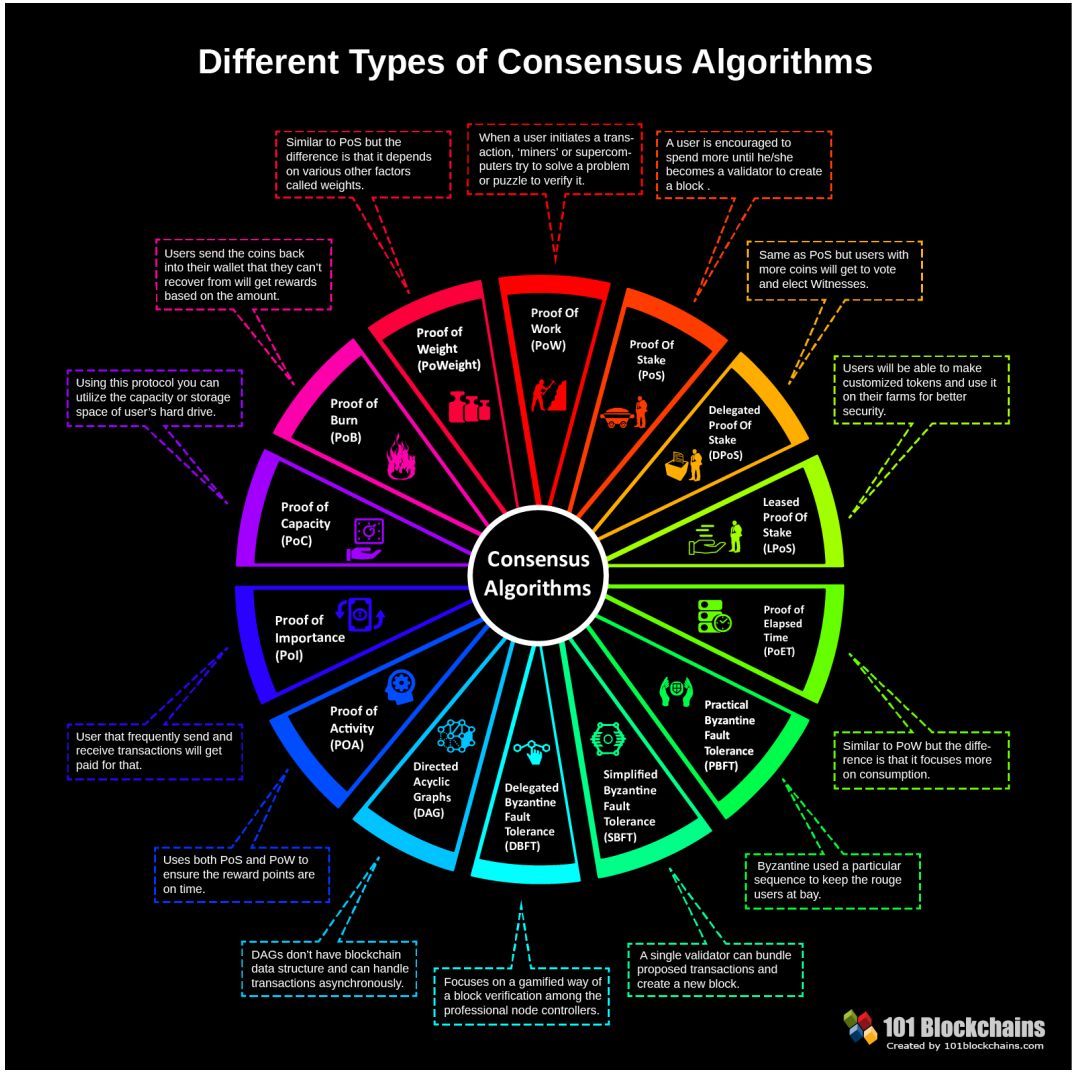
In case of blockchain, the processes have to get an agreement for a new block of transactions. This introduces specific limitations:

- should have high performance regarding transactions per second (TPS) and block acceptance time
- should be *censorship resistant*.

Other properties

- **persistence**
- **liveness**
- common prefix
- chain quality
- chain growth.

# Are there any blockchains beyond Proof-of-Work?




See `https://hackernoon.com/consensuspedia-an-encyclopedia-of-29-consensus-algorithms-e` for 15 minutes introduction.

# Table of Contents

# Proof of Stake

The **limited resource** used for voting can be found not only on the outside (consumed power and hardware), but also inside the system itself: the **digital coins**.

- bitcointalk.org, July 2011: idea
- PPCoin blockchain, August 2012: white paper and implementation
- Now Peercoin in Top150 by Cryptocurrency Market Capitalization.

# Peercoin Mining Process

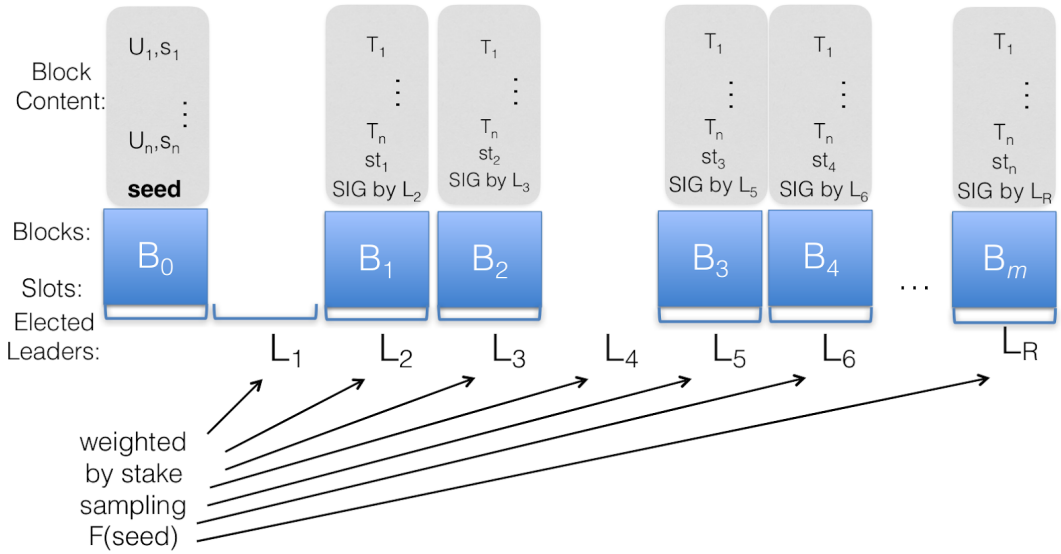
1. Selecting an output received at least 30 days ago.
2. Forming the Kernel structure, which includes: determined data from the output (time of the block of its origin, its number in the block etc.), current time, and so called nStakeModifier (periodically re-computed block of pseudorandom bits).
3. Hashing Kernel and verifying the resultant value against the current target, which depends on the current network complexity (higher complexity => lower target), "age" of the output ("older" output => larger target) and its sum (larger sum => larger target).
4. If hash is greater than the target, return to 1), i.e. take next output.
5. If output is "successful", we spend it in coinbase transaction (by sending to ourselves), add block reward and fees for included transactions and sign the whole block using the key related to the spent output.
6. The block is done. Start searching the next one.

# Notes

- The blockchain verification is determinate: current time is received from the block header, output data from the blockchain, nStakeModifier is also uniquely calculated for each block;
- Output should be "old" so that an attacker could not find a "good" output that allows for finding a block by transferring money between his wallets;
- nStakeModifier is calculated on the basis of the last blocks and is therefore incalculable. It makes mining even more unpredictable (and more resistant to possible attacks);
- The current timestamp from 2) can vary in very wide range: plus/minus an hour. Therefore, 7200 hashes should be actually checked for each output, not one;
- The "age" multiplier of the target has the upper limit of 90 days. Otherwise, an attacker could generate a number of blocks in a row with very high probability if he possessed only a few VERY old coins.
- Delegated Proof-of-Stake.

# Notes (2)

- Delegated Proof-of-Stake: the stake holders in the system can elect leaders(witnesses) who will vote in their behalf. This makes it faster than the normal PoS.
- Ouroboros: A Provably Secure Proof-of-Stake Blockchain Protocol [Kiayias2017].



# Proof of Stake. Pros and cons

Pros

1. Energy savings.
2. No endless "arms race"
3. Attack becomes more expensive

Cons

1. Attack with spent money $\rightarrow$ time stamping.
2. "Nothing at stake" problem: double-spend.
3. Who owns 51%?

# Proof of Activity

Combining Proof-of-Work and Proof-of-Stake (even in PeerCoin itself): blockchain includes both types of blocks.

1. First, the proof-of-work miner works, searching for the hash conforming with complexity... i.e. everything is as usual;
2. Then, when the hash is found, miner transmits all data to the network, but it is still not a block. It is but a template. A number of such templates might appear;
3. The block hash (256 pseudorandom bits) is interpreted as N numbers. Each number corresponds to some satoshi (e.g. if we number all emitted coins starting with the first block);

# Proof of Activity (2)

4. Each satoshi, in its turn, corresponds to a single public key of its current owner. This way we define N random owners;
5. The block "template" becomes a full-fledged block as soon as all N owners sign it (with the keys defined in step 4);
6. If one of the signers is unavailable (or does not mine on personal considerations) - no problem. Proof-of-work miners keep working, generating new "templates" with different sets of signers-candidates;
7. Sooner or later (depending on the percentage of users online) some block will be signed N times. The block-reward is shared between proof-of-work-miner and N signers.

# Notes

1. The protocol seems to require continuous data exchange. In order to reduce traffic, the block "template" does not include the transaction list. It is added by the last signer, when finalizing the block;
2. If $N = 3$ and only 10% of the users are online, then, obviously, proof-of-work miners will generate approximately $10 \cdot 10 \cdot 10 = 1000$ "templates" before one of them is signed. However, if message size is about a hundred bytes, it is insignificant.

# Proof of Burn

- Instead of burning electricity you should destroy your digital coins.
- You "burn" them by sending to such address where they cannot be redeemed. For example, to the address, which is a hash of a random number: chances for someone picking public and private keys for it are negligible.
- Thus, by 'throwing away' your coins, you get the rights for life-time mining, which is a lottery among the owners of burnt coins.
- Slimcoin (capitalization $\leq 20k$ USD).

# Proof of Space/Capacity/Storage

- allocate significant volume of the hard drive space to start mining
- repeatedly hashing your public key and nonces and the last block header gives us the index number
- provides botnet protection by design.

# Table of Contents

# Consensus Problem

A fundamental problem in distributed computing is how to achieve system reliability in the presence of faulty processes – to get consensus.

The consensus problem requires agreement among a number of processes for a single data value, for example, about a new bit value.

The correct consensus classically means that three conditions hold for every execution of the algorithm

- *termination*: eventually each correct process sets its decision variable
- *agreement*: the decision value of all correct processes is the same
- *integrity*: if the correct processes all proposed the same value, then any correct process in the decided state has chosen that value.

# Faulty Processes Models

- *fail-stop*: process stops and can not resume
- *omission*: process could not send some messages and it could delete some received messages without reading or even information about such a messages receiving
- *authorized Byzantine failures* stands for failures in which absolutely no conditions are imposed, but all the messages supposed to be cryptographically signed by author processes
- *Byzantine failures* stands for failures in which absolutely no conditions are imposed.

# Message Delivery and Processor Assumptions

- *asynchronous system*: any message could be delivered at any time after sending, or not be delivered at all
- *partially synchronous system*: some unknown time $T$ exists, such that each message is delivered faster than $T$.
- *synchronous system*: each message is delivered faster than a known finite latency $T$.

The processes also could be considered as *asynchronous*, *partially synchronous* or *synchronous* based on their relative computation speed.

# Classic BFT Results

- Fischer, Lynch and Patterson – FLP Impossibility – impossibility of distributed consensus with one faulty process: the correct consensus in the asynchronous system is impossible if a least one Byzantine process is presented
- the Byzantine fault tolerant consensus algorithm exists in partially synchronous and synchronous system if the total number of processes $N$ greater or equal than $3f + 1$, where $f$ is the number of faulty processes $f$.

# BFT for Blockchain

In case of blockchain, the processes have to get an agreement for a new block of transactions. This introduces specific limitations:

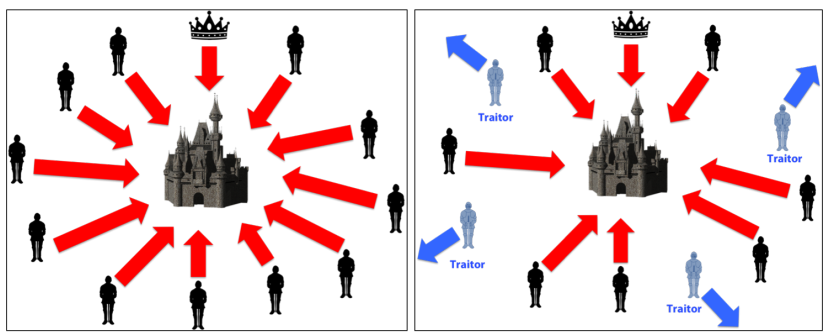
- the BFT protocol for blockchain should have high performance regarding transactions per second (TPS)
- should be *censorship resistant*.

There most popular are BFT protocols for partially synchronous systems [Kwon2014, Cachin2016].

Alternatives: probabilistic BFT [Zhao2007] and atomic broadcast [Defago2004, Miller2016].

# Byzantine Generals' Problem

- A group of generals, each commanding a portion of the Byzantine army, encircle a city. These generals wish to formulate a plan for attacking the city.
- The generals must only decide whether to attack or retreat.
- Some generals may prefer to attack, while others prefer to retreat.
- The important thing is that every general agrees on a common decision, for a halfhearted attack by a few generals would become a rout and be worse than a coordinated attack or a coordinated retreat.
- The problem is complicated by the presence of traitorous generals.




Coordinated Attack Leading to Victory | Uncoordinated Attack Leading to Defeat

# Why $N \geq 3f + 1$?

- $f$ faulty processes
- $h$ honest processes
- $N = h + f$
- $1 \geq \alpha > 1/2$: decision rule $\texttt{votes} \geq \alpha \cdot N$.

Honest processes can make a decision without faulty processes

$$\alpha \cdot N \leq h.$$

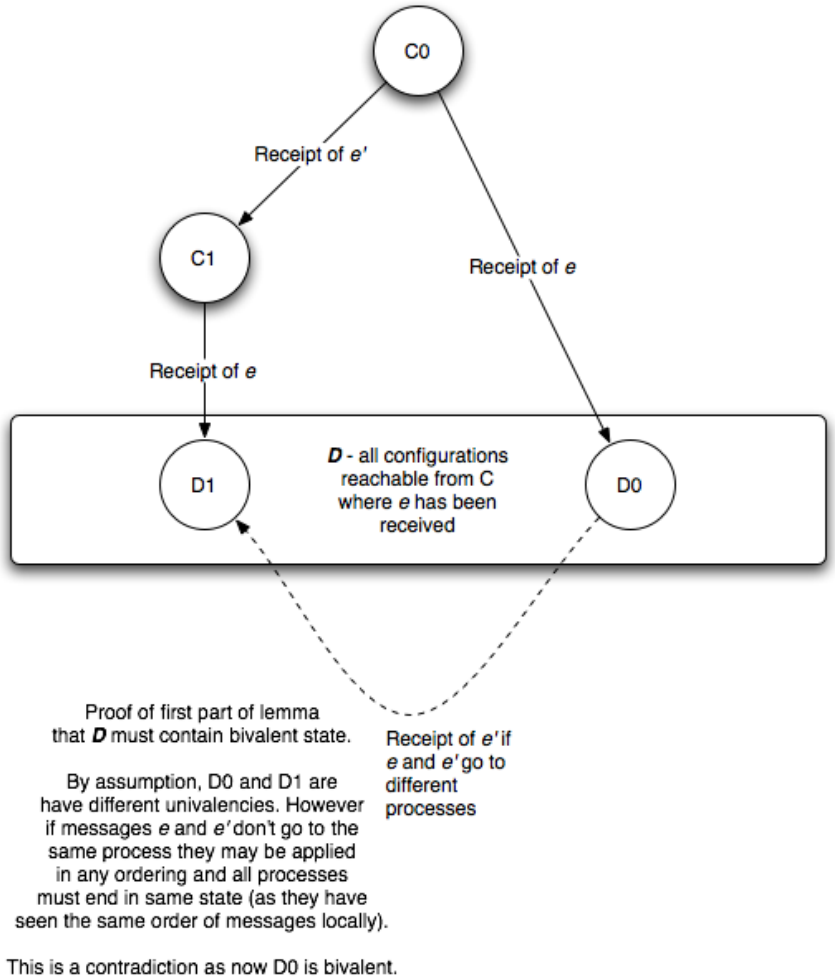
Each decision include more than a half of honest processes

$$\alpha \cdot N > [h/2] + f;$$

So $\alpha > 2/3$ and $N \geq 3f + 1$.

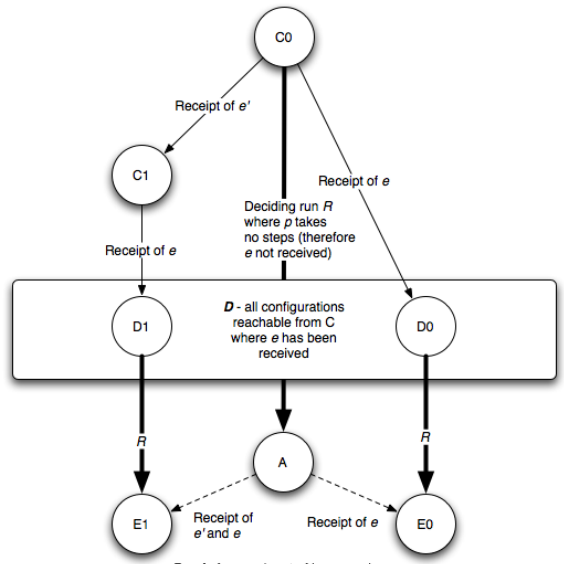
# FLP

There must be some initial configuration of the system in which the outcome of consensus is not already determined.



# FLP (2)

If you start in a bivalent configuration then it is always possible to reach another bivalent configuration by delaying a pending message.



# FLP (3)

The chance of entering that infinite undeciding loop can be made arbitrarily small – it's one thing for nondecision to be possible, another thing entirely for it to be guaranteed.

Still, there is much to be said for keeping this possibility in mind when your system livelocks during consensus and you don't know why. Also, in practice, real systems aren't always truly asynchronous.

# A bit more about BFT

- Lamport L., Smith P. M. Byzantine clock synchronization //ACM SIGOPS Operating Systems Review. – 1986.
- Leader Election and Censorship Resistance `https://exonum.com/blog/04-28-17-leader-election-roundrobin/`

Thank you for your attention!