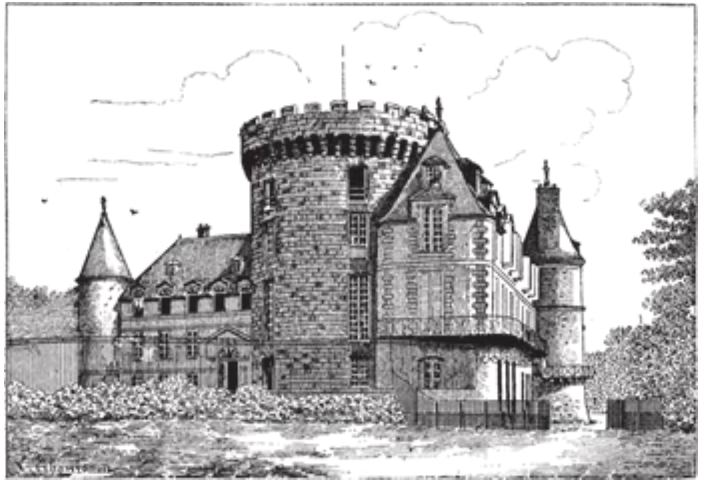
Blockchains & Distributed Ledgers

Lecture 05

Aggelos Kiayias

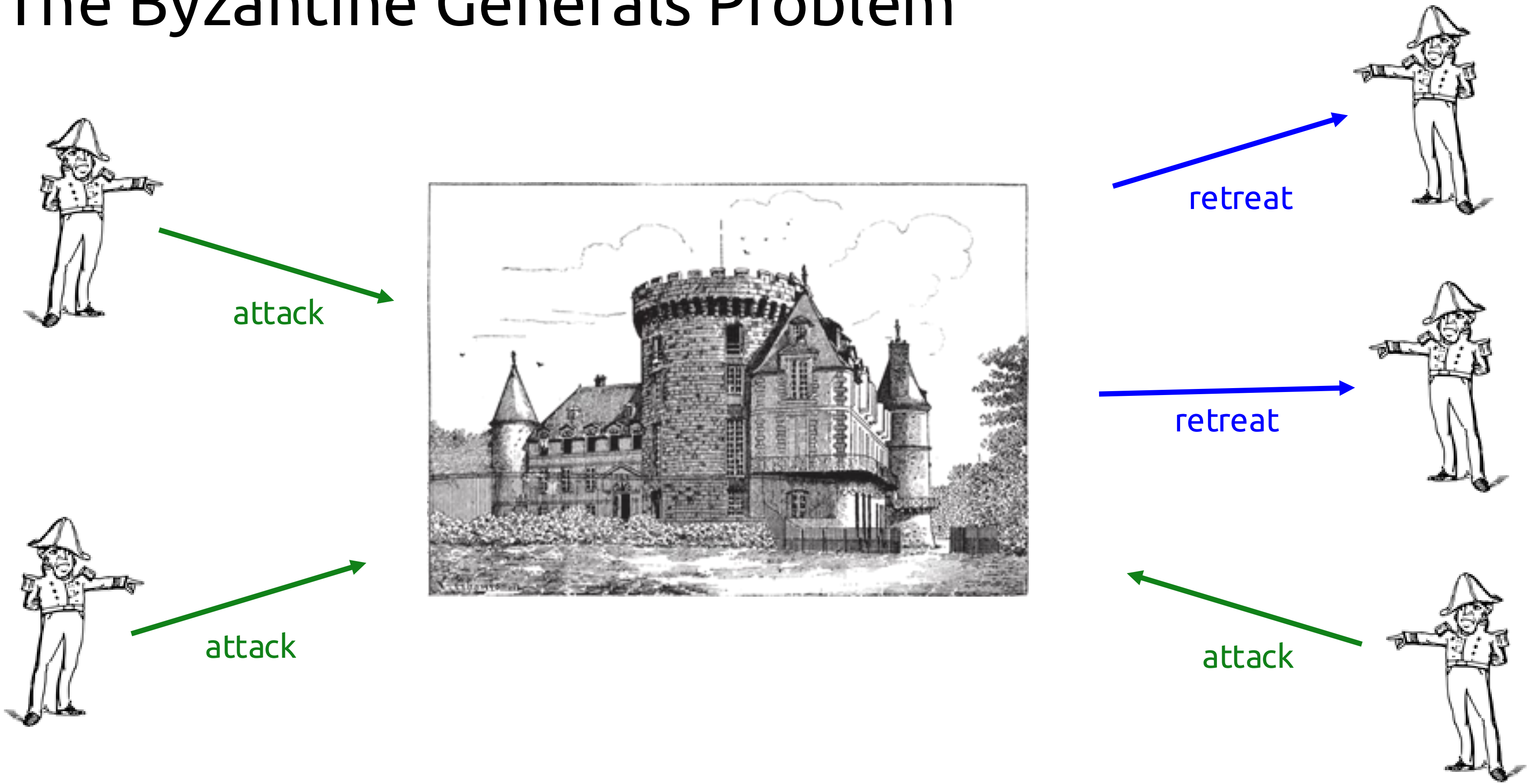
Slide credits: AK, Dimitris Karakostas, Dionysis Zindros, Christos Nasikas

The Byzantine Generals Problem

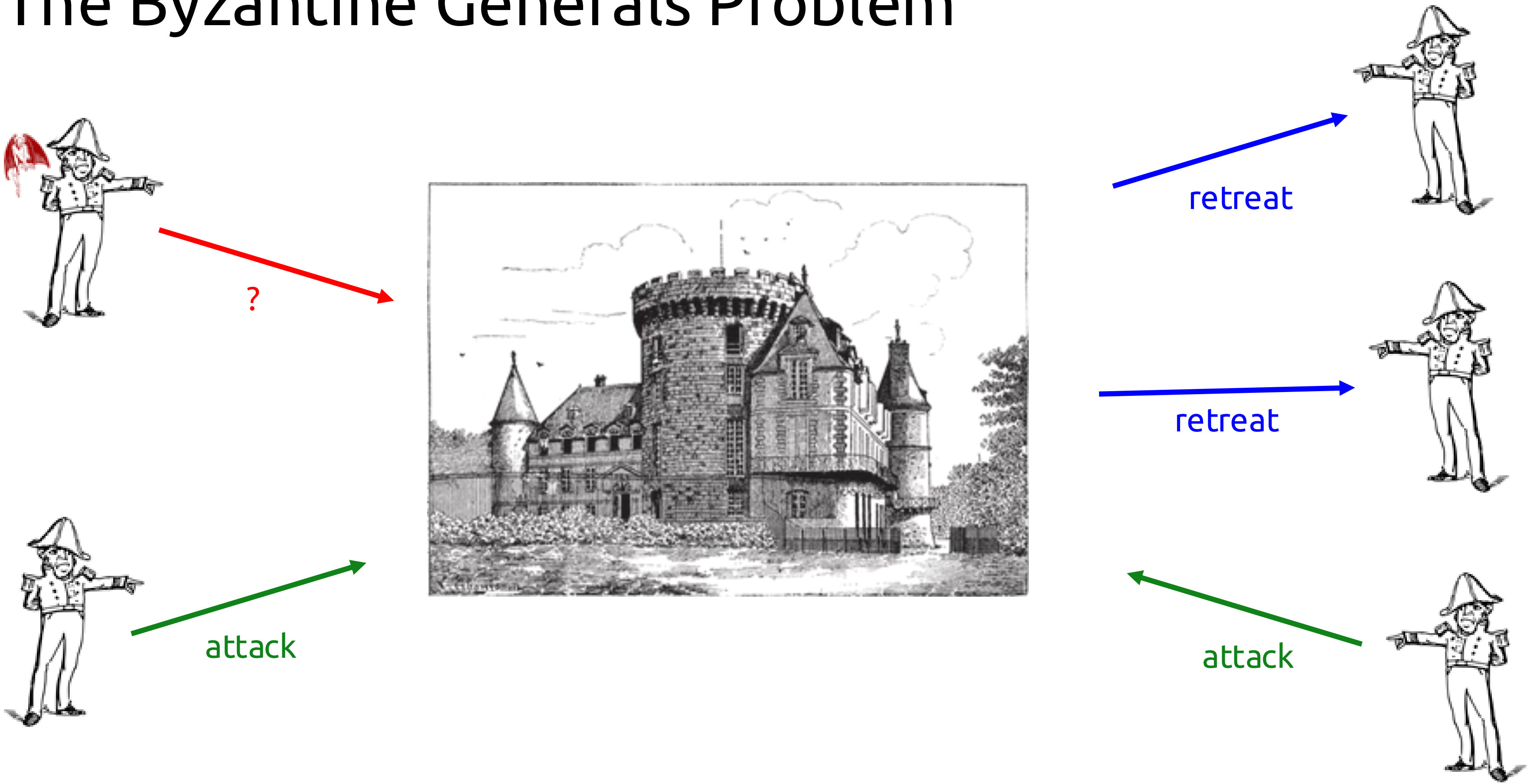


[LSP82] Leslie Lamport, Robert Shostak, and Marshall Pease.
The byzantine generals problem

The Byzantine Generals Problem



The Byzantine Generals Problem

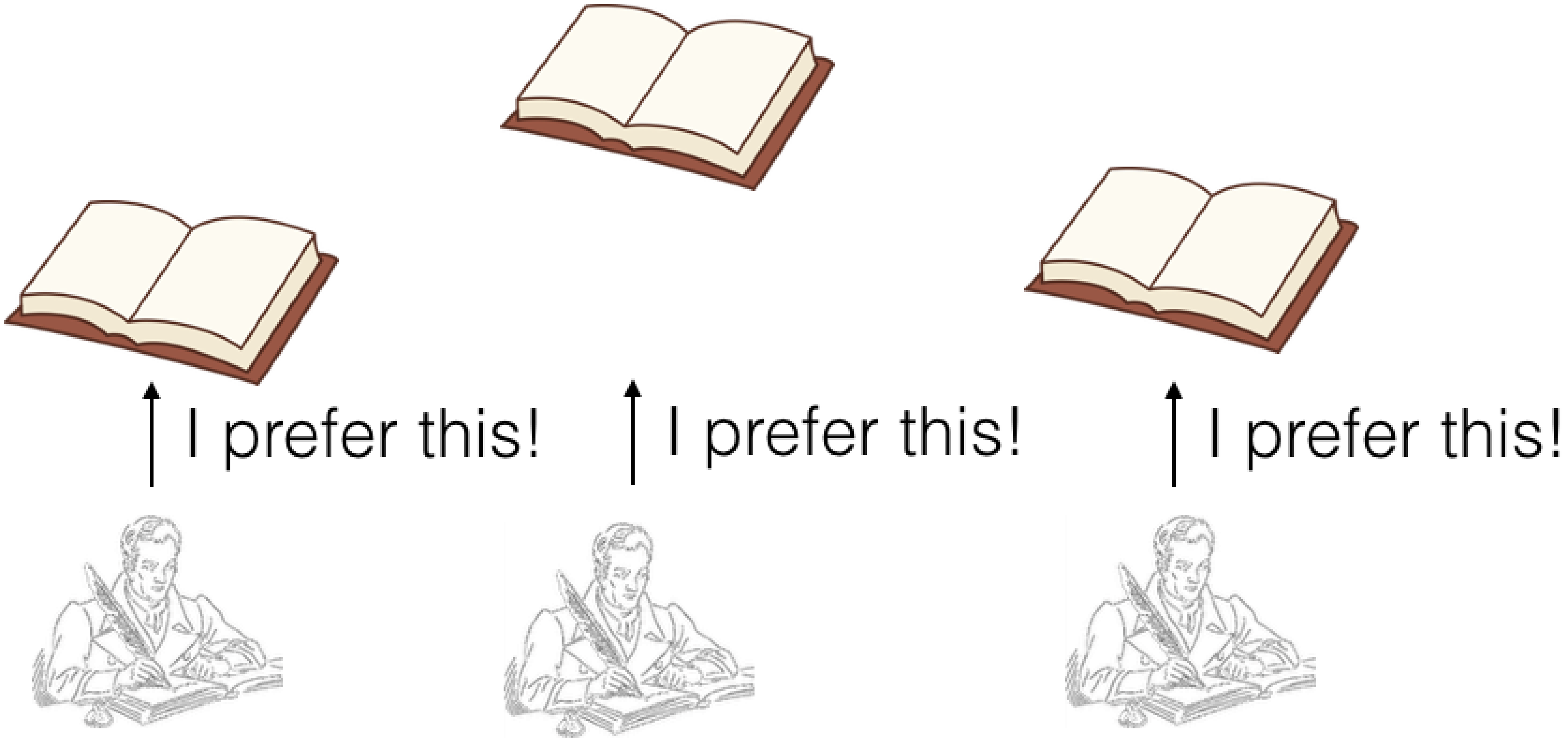


The Consensus Problem

Motivation for the Consensus Layer, I

- A transaction history and/or state of the service needs to be **agreed** by all servers.
- Servers may be operated by participants with **diverging interests**, in terms of the history of transactions and/or state of the service.

Motivation for the Consensus Layer, II

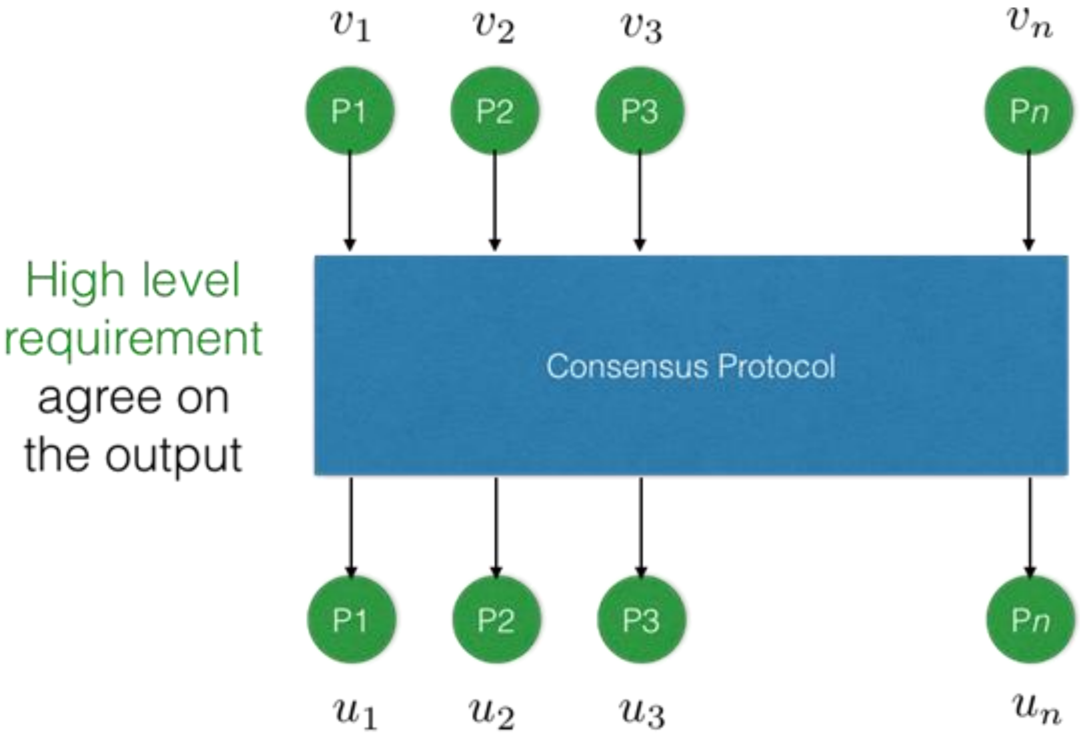


Consensus : Problem Statement

- A number (*t*) of the participating entities can diverge from the protocol.
- This has been called Byzantine behaviour in the literature.
- The properties of the protocol are defined in the presence of this “malicious” coalition of parties that attempts to disrupt the process for the “honest” parties.

$$\mathsf{H}, |\mathsf{H}| = n - t$$

The consensus problem



Study initiated by Lamport, Pease, Shostak 1982

Consensus Properties

- Termination $\forall i \in \mathsf{H}(u_i \text{ is defined})$

Consensus Properties

- Termination $\forall i \in \mathsf{H}(u_i \text{ is defined})$

- Agreement $\forall i, j \in \mathsf{H}\,(u_i = u_j)$

Consensus Properties

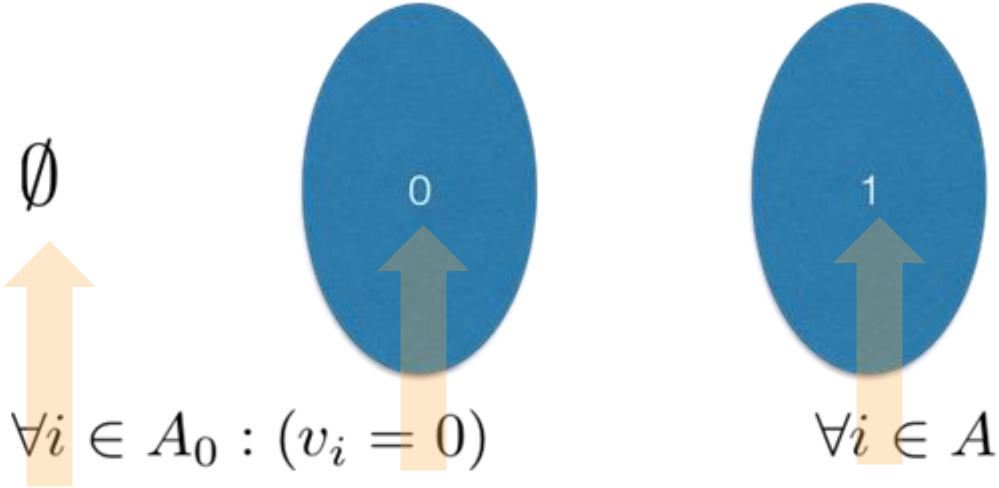
- Termination $\forall i \in \mathsf{H}(u_i \text{ is defined})$
- Agreement $\forall i, j \in \mathsf{H}\,(u_i = u_j)$
- Validity $\exists v(\forall i \in \mathsf{H}\,(v_i = v)) \implies (\forall i \in \mathsf{H}\,(u_i = v))$

Consensus Properties

- Termination $\forall i \in \mathsf{H}(u_i \text{ is defined})$
- Agreement $\forall i, j \in \mathsf{H}\,(u_i = u_j)$
- Validity $\exists v(\forall i \in \mathsf{H}\,(v_i = v)) \implies (\forall i \in \mathsf{H}\,(u_i = v))$
  - Strong Validity $\forall i \in \mathsf{H}\,\exists j \in \mathsf{H}\,(u_i = v_j)$

Honest Majority is Necessary, I

$\frac{n}{2}$ $\frac{n}{2}$



$\emptyset$

$\forall i \in A_0 : (v_i = 0)$ $\forall i \in A_1 : (v_i = 1)$

Consider an adversary that performs one of the following with probability ⅓

Honest Majority is Necessary, II

- If consensus protocol secure:
  - Adversary corrupts A_0: output of honest parties (that belong to A_1) should be 1.
  - Adversary corrupts A_1: output of honest parties (that belong to A_0) should be 0.
  - Adversary corrupts no-one: output of all parties should be the same.
- Adversary corrupts each set with prob. ⅓ and instructs corrupted parties to follow the protocol
  - honest parties cannot distinguish between honest/corrupted parties
- If all parties output same value: validity is violated with prob. at least ⅓
- If all parties output different value: consistency is violated with prob. at least ⅓

Is Honest Majority Sufficient?

- Two important scenarios have been considered in the consensus literature.
    - Point to point channels. **No setup.**
    - Point to point channels. **With setup.**

- The setup provides a correlated private initialization string to each participant; *it is assumed* to be honestly produced.

Setup and Network

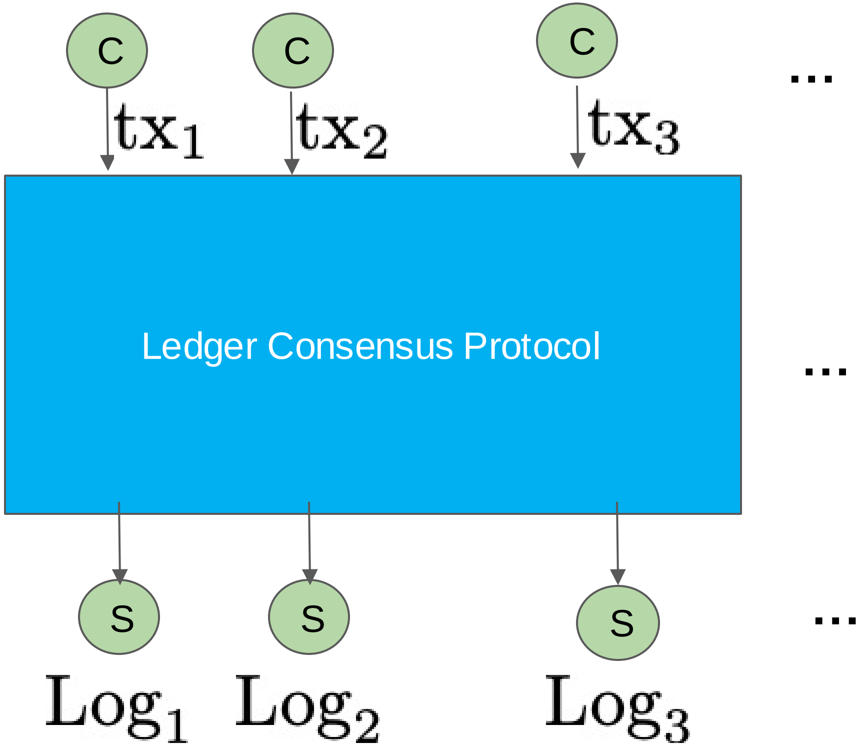
Setup/Network	Synchrony	Partial Synchrony
No Setup	$t < n/3$	$t < n/3$
With Setup	$t < n/2$	$t < n/3$

We know consensus can be achieved, assuming the above bounds on adversarial parties.

The typical setup and network configuration in classical consensus protocols

- Setup: a public-key directory
  - Parties have signing and verification keys for a digital signature scheme.
  - Each party knows every other party’s verification key.
- Network: point-to-point channels
  - Synchronous, partially synchronous, or asynchronous

Ledger Consensus



Related to state machine replication [Sch90]

Ledger Consensus Properties

- Consistency:

$$\forall i, j \in \mathbf{H}, t, t' : (\mathrm{Log}_i[t] \not\preceq \mathrm{Log}_j[t']) \rightarrow (\mathrm{Log}_j[t'] \preceq \mathrm{Log}_i[t])$$

- Liveness:

$$(\forall i \in \mathbf{H} : \mathrm{tx} \in I_i[t]) \rightarrow (\forall i \in \mathbf{H} : \mathrm{tx} \in \mathrm{Log}_i[t+u])$$

$I_j[t] =$ Transaction Input of party *j* at time *t* consistent with its log

$\mathrm{Log}_j[t] =$ Log of party *j* at time *t*

[.. and there are more properties of interest here.. e.g., fairness of transaction serialization, exporting a clock,...]

Solving ledger consensus

(solving SMR, actually)

Practical Byzantine Fault Tolerance

Miguel Castro and Barbara Liskov
Laboratory for Computer Science,
Massachusetts Institute of Technology,
545 Technology Square, Cambridge, MA 02139
{castro,liskov}@lcs.mit.edu

arbitrary behavior. Whereas previous algorithms assumed a synchronous system or were too slow to be used in practice, the algorithm described in this paper is practical: it works in asynchronous environments like the Internet and incorporates

- Its operation & threat model features:
  - Asynchronous communication, point-to-point channels.
  - Authenticated setting: parties are able to identify message sources
  - Fixed number of participants
  - Computationally bounded adversary in the sense it cannot forge digital signatures.
  - Setup assumption: PKI
  - Adversary controls less than ⅓ of participants (required)
  - No privacy properties for servers
  - communication complexity $O(n^2)$ in best case, where n is the number of participants
  - No protocol incentives

Solving ledger consensus, II

Practical Byzantine Fault Tolerance

Miguel Castro and Barbara Liskov
Laboratory for Computer Science,
Massachusetts Institute of Technology,
545 Technology Square, Cambridge, MA 02139
{castro,liskov}@lcs.mit.edu

arbitrary behavior. Whereas previous algorithms assumed a synchronous system or were too slow to be used in practice, the algorithm described in this paper is practical: it works in asynchronous environments like the Internet and incorporates

Bitcoin: A Peer-to-Peer Electronic Cash System

Satoshi Nakamoto
satoshin@gmx.com
www.bitcoin.org

Abstract. A purely peer-to-peer version of electronic cash would allow online payments to be sent directly from one party to another without going through a financial institution. Digital signatures provide part of the solution, but the main benefits are lost if a trusted third party is still required to prevent double-spending. We propose a solution to the double-spending problem using a peer-to-peer network. The network timestamps transactions by hashing them into an ...oing chain of hash-based proof-of-work, forming a record that cannot be chang... ...thout redoing

runs uninterruptedly over the Internet
since 2009 assuming bounded delays

Bitcoin Blockchain

- A **ledger consensus** protocol, where
  - participants called “miners” run an intensive proof of work (PoW) algorithm at every step.
- Its threat model features:
  - Bounded delay communication by ‘diffusion’ of messages (no point-to-point channels)
  - No ability to identify message sources - unauthenticated setting
  - Unknown and fluctuating number of participants … **dynamic availability**
  - Computationally bounded adversary possessing ~< **50%** of resources
  - Setup: public random string (no PKI)
  - Tolerates potential spikes of adversarial majority… **self-heals**
  - Privacy for maintainers… protocol transcript maintains anonymity/unlinkability of participants
  - Optimal amortized communication complexity of single multicast per block.
  - Protocol ince

[Nakamoto2009]

When bitcoin appeared the above threat model was an unexplored territory for consensus

The threat model was also not spelled out anywhere in the original works — protocol was given as is

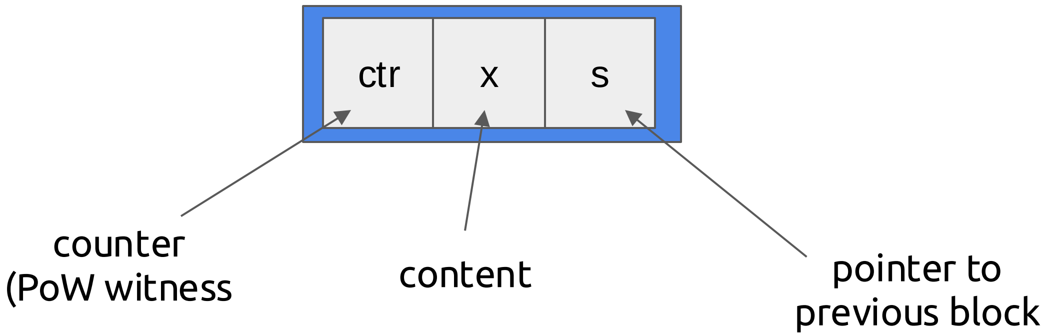
The Bitcoin "backbone"

- The core of the bitcoin protocol
  - The chain validation predicate.
  - The chain selection rule (max-valid)
  - The proof of work function.
  - The main protocol loop
- Protocol is executed by "miners"

[GKL2015] Garay, Kiayias, Leonardos. The Bitcoin Backbone Protocol: Analysis and Applications.

Model

- Assume there are *n* parties running the protocol
- Synchronous
- Each party has a quota of *q* queries to the function H(.) in each round
- A number of *t* parties are controlled by an adversary (a malicious coalition)
    - Security arguments are *for any adversary*


ctr
x
s
counter
(PoW witness
content
pointer to
previous block

ALGORITHM 3: The *PoW* function, parameterized by q, T, s_{init} and hash functions $H(\cdot), G(\cdot)$. The input is (x, C).

function pow(x, C)
  if $C = \varepsilon$ **then** ▹ Determine PoW instance
    $s \leftarrow s_{\text{init}}$ ▹ Genesis block initialization
  else
    $(\langle s', x', ctr' \rangle, s) \leftarrow \text{head}(C)$
  end if
  $ctr \leftarrow 1$
  $B \leftarrow \varepsilon$
  $h \leftarrow G(s, x)$
  while $(ctr \leq q)$ **do**
    $s_{\text{new}} \leftarrow H(ctr, h)$ ▹ This $H(\cdot)$ invocation subject to the q bound
    if $(s_{\text{new}} < T)$ **then**
      $B \leftarrow \langle s, x, ctr \rangle$
      break
    end if
    $ctr \leftarrow ctr + 1$
  end while
  if $B \neq \varepsilon$ **then**
    $C \leftarrow C || (B, s_{\text{new}})$ ▹ Extend chain
  end if
  return C
end function

Blockchain

$B_0 = \langle \perp, x_0, ctr_0 \rangle$ genesis block

$B_1 = \langle s_1, x_1, ctr_1 \rangle$

…

$B_n = \langle s_n, x_n, ctr_n \rangle$

$\mathcal{C} = \langle B_0, B_1, \ldots, B_n \rangle$

chain head

$$s_1 = H(ctr_0, G(x_0, \text{“label”})) = s_{init}$$

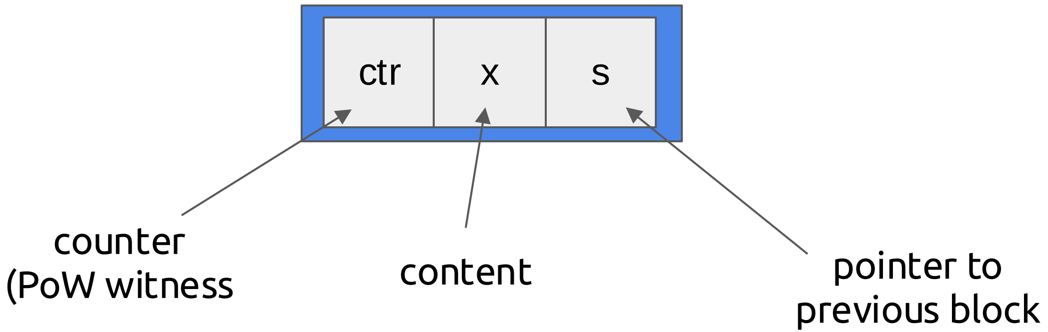
$$s_i = H(ctr_{i-1}, G(x_{i-1}, s_{i-1}))$$

$$x_{\mathcal{C}} = \langle x_0, x_1, \ldots, x_n \rangle$$

$$\mathcal{C}^{\lceil k} = \langle B_0, B_1, \ldots, B_{n-k} \rangle$$

ALGORITHM 1: The *chain validation predicate*, parameterized by q, T, the hash functions $G(\cdot), H(\cdot)$, and the *content validation predicate* $V(\cdot)$. The input is C.

function validate(C)
    $b \leftarrow V(\mathrm{x}_C)$
    if $b \wedge (C \neq \varepsilon)$ **then** ▷ The chain is non-empty and meaningful w.r.t. $V(\cdot)$
        $(\langle s, x, ctr \rangle, s') \leftarrow \mathrm{front}(C)$
        $s_{\mathrm{front}} \leftarrow s'$
        repeat
            if $\mathrm{validblock}_q^T(\langle s, x, ctr \rangle) \wedge (H(ctr, G(s, x)) = s') \wedge (s_{\mathrm{front}} = s')$ **then**
                $s_{\mathrm{front}} \leftarrow s$ ▷ Retain hash value
                $C \leftarrow C^{\lceil 1}$ ▷ Remove the head from C
                $(\langle s, x, ctr \rangle, s') \leftarrow \mathrm{head}(C)$
            else
                $b \leftarrow$ False
            end if
        until $(C = \varepsilon) \vee (b = \text{False})$
    end if
    return $(b \wedge (s_{\mathrm{front}} = s_{\mathrm{init}}))$
end function



validblock predicate:

$$(H(ctr, G(x, s)) < T) \wedge (ctr \leq q)$$

hash functions

PoW target

upper bound
(no of hashes per round)

ALGORITHM 2: The function that finds the "best" chain, parameterized by function $\max(\cdot)$. The input is $\{C_1, \ldots, C_k\}$.

function maxvalid($C_1, \ldots, C_k$)
    $temp \leftarrow \varepsilon$
    for $i = 1$ to k **do**
        if validate(C_i) **then**
            $temp \leftarrow \max(C_i, temp)$
        end if
    end for
    return $temp$
end function

ALGORITHM 4: The Bitcoin backbone protocol's main loop, executed every round when a party is activated by the environment, parameterized by the *input contribution function* $I(\cdot)$ and the *chain reading function* $R(\cdot)$. At the onset it is assumed "init= True".

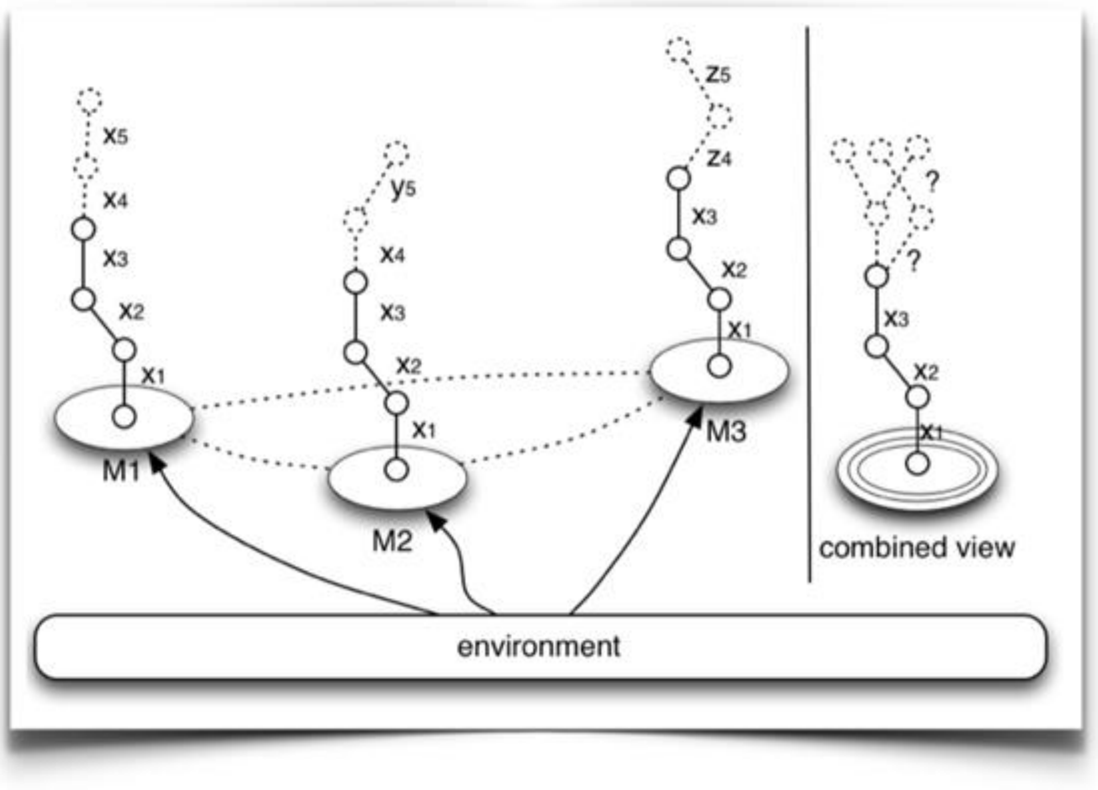
---

if (init) **then**
    $\mathcal{C} \leftarrow \varepsilon$
    $st \leftarrow \varepsilon$
    $round \leftarrow 1$
    $init \leftarrow$ False
else
    $\tilde{\mathcal{C}} \leftarrow$ maxvalid($\mathcal{C}$, any chain $\mathcal{C}'$ found in Receive)
    if Input contains Read **then**
        write $R(\tilde{\mathcal{C}})$ to Output   ▹ Produce necessary output before the PoW stage.
    end if
    $\langle st, x\rangle \leftarrow I(st, \tilde{\mathcal{C}}, round,$ Input, Receive$)$ ▹ Determine the x value.
    $\mathcal{C}_{\text{new}} \leftarrow \text{pow}(x, \tilde{\mathcal{C}})$
    if $\mathcal{C} \neq \mathcal{C}_{\text{new}}$ **then**
        $\mathcal{C} \leftarrow \mathcal{C}_{\text{new}}$
        Diffuse($\mathcal{C}$)   ▹ Send the chain in case of adoption/extension.
    else
        Diffuse($\perp$)   ▹ Signals the end of the round to the diffuse functionality.
    end if
    $round \leftarrow round + 1$
end if

Basic Properties

- Common Prefix
- Chain Quality
- Chain Growth

Common Prefix, I



Common Prefix, II

(strong common prefix / consistency)

$$\forall r_1, r_2, (r_1 \leq r_2), P_1, P_2, \text{ with } \mathcal{C}_1, \mathcal{C}_2 : \; \mathcal{C}_1^{\lceil k} \preceq \mathcal{C}_2$$

- The property holds true, in a probabilistic sense, with an error that decays exponentially in ***k***

Racing Attacks

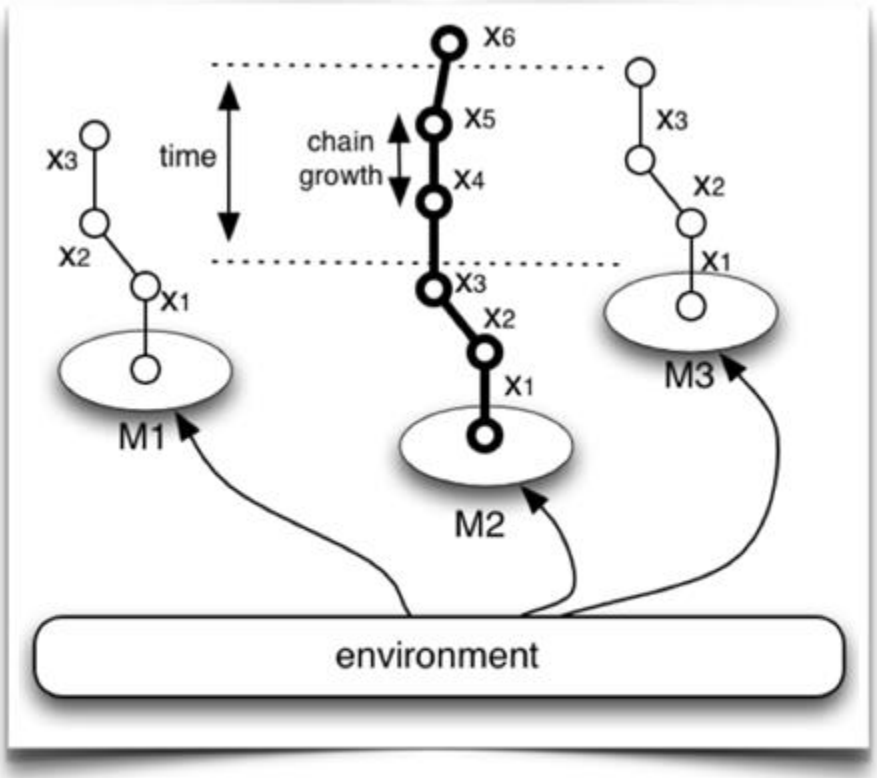
- Attacker splits from the main chain and tries to overtake the “honest chain”

  => Common prefix breaks

- <u>Intuition why the attack is a small probability event</u>:

  concentration bounds help honest parties

Chain Growth, I



Chain Growth, II

Parameters $\tau \in (0, 1), s \in \mathbb{N}$

In any period of s rounds at least τs blocks are added to the chain of an honest party P.

- The property holds true in a probabilistic sense with an error probability that exponentially decays in ***s***

$\tau \approx$ probability at least one honest party finds a POW in a round

Abstention Attacks

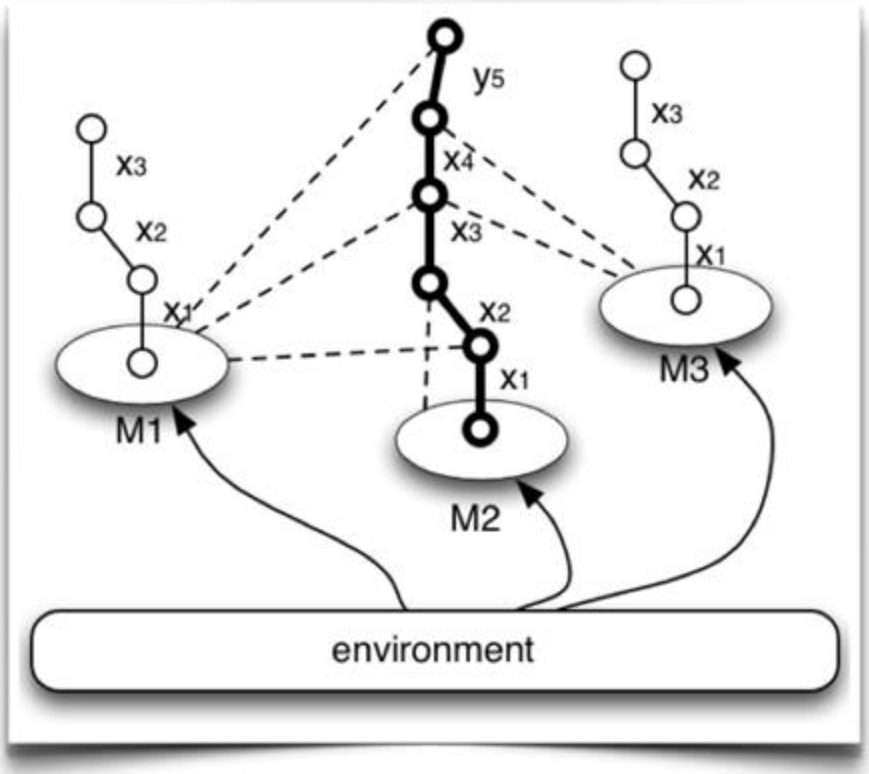
- Attacker stops producing blocks

  => Chain growth stops

- Intuition why the attack is a small probability event:

  honest parties will eventually issue blocks

Chain Quality, I



Chain Quality, II

Parameters $\mu \in \{0, 1\}, \ell \in \mathbb{N}$
The ratio of blocks of an ℓ-long segment of an honest chain produced by the adversary is bounded by $(1 - \mu)\ell$

- The property holds true probabilistically with an error that exponentially decays in **ℓ**

$$\mu \approx \frac{n - 2t}{n - t}$$

Block Withholding Attacks

- Attacker mines privately and releases their block at the same time an honest party releases its own block
- Assuming honest propagation favours the adversary, the honest block is dropped, reducing chain quality

- <u>Intuition why the attack is a small probability event:</u>

  over time the adversary cannot produce blocks at the same rate as honest parties (to compete with them)

Establishing a Ledger Consensus from a Blockchain

- Consistency ← *(strong) Common Prefix*
  - need to exclude *k* most recent blocks
- Liveness ← *Chain Growth* and *Chain Quality*
  - leave sufficient time for chain to grow
  - apply chain quality to ensure that at least one honest block is included

Ledger Consensus vs. Consensus

- What is the connection?
  - ledger is an ever-going protocol with inputs (e.g., transactions) continuously coming from also external sources
  - consensus is a one-shot execution
- Is it possible to reduce consensus to the ledger? Is it possible to reduce the ledger to consensus?
  - (See the GKL paper for more details)

Hash operations

- Consider a regular PC (30 MHash / sec)
- With expectation of 2^{78} hashing operations (current difficulty – 10/24), mining a block will require ~ 320 *million* years.

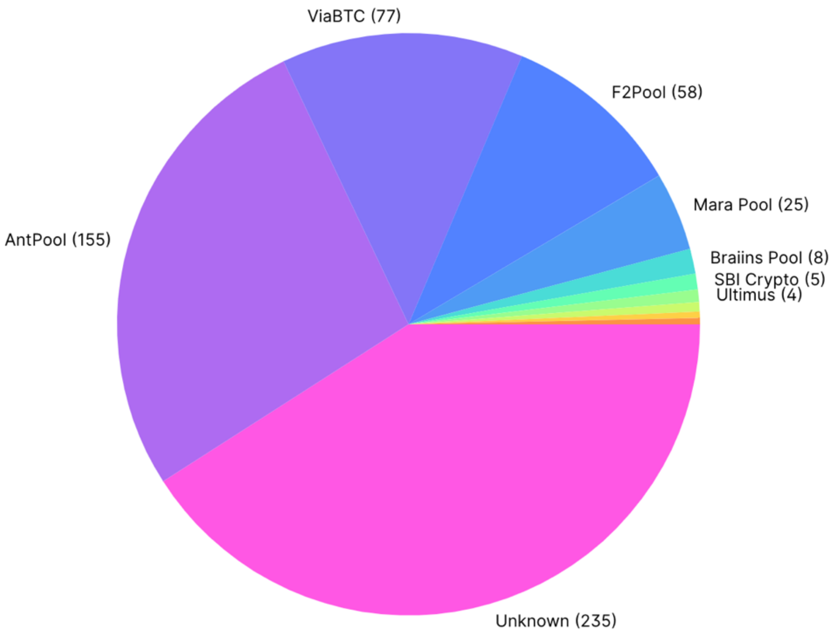
https://en.bitcoin.it/wiki/Non-specialized_hardware_comparison

https://www.coinwarz.com/mining/bitcoin/difficulty-chart

Parallelising mining

- Bitcoin's Proof of Work can be parallelized
- Parties tend to form mining *pools*
  - Instead of working separately, work **together** to solve PoW for the same block.
  - By collecting **"shares"** (small hashes of the block that are not quite as small as needed) one can prove how much they contributed.

Bitcoin mining pools



https://www.blockchain.com/pools

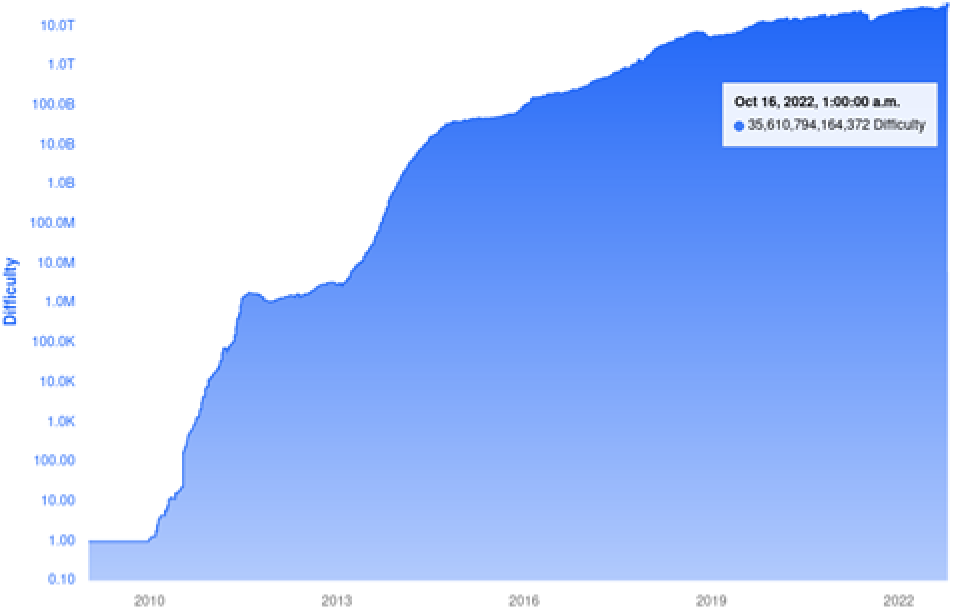
Recall: PoW algorithm

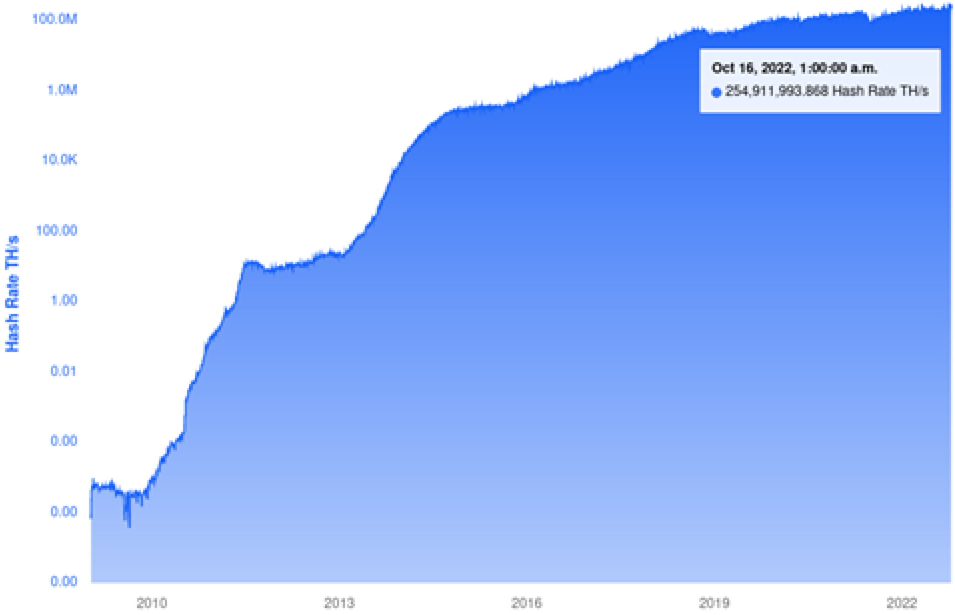
```
int counter;
counter = 0
while Hash(data, counter) > Target
        increment counter
return counter
```

Dynamic Availability

- So far: *n* nodes maintain the blockchain
- This number may change over time:
  - new users enter the system
  - existing users leave
- The change over time can be dramatic
- The Bitcoin blockchain handles this, by adjusting the target (difficulty) of the Proof of Work algorithm

Target difficulty / Total hash rate over time





Adjusting the difficulty

"maxvalid" rule is changed

s.t. parties adopt **chain with highest difficulty** linearly related to:

$$\sum_i \frac{1}{T_i}$$

The *f* parameter [GKL15]

f = probability of producing a block in a round of interaction

- f depends on:
  - target T
  - number of miners
  - duration of round
- If f becomes too small, parties do not progress
  - Chain growth slows
  - Liveness is hurt
- If f becomes too large, parties “collide” often
  - Attacker can exploit network scheduling of message delivery to create forks
  - Consistency is hurt
- To resolve this dynamically, Bitcoin **recalculates** T to keep f constant

Target recalculation

$$\text{next target} = \begin{cases} \frac{1}{\tau} \cdot T & \text{if } \frac{n_0}{n} \cdot T_0 < \frac{1}{\tau} \cdot T; \\ \tau \cdot T & \text{if } \frac{n_0}{n} \cdot T_0 > \tau \cdot T; \\ \frac{n_0}{n} \cdot T_0 & \text{otherwise} \end{cases}$$

- Recalculation occurs at the end of every “epoch”
  - m: epoch length in blocks (in Bitcoin: 2016)
- n_0: estimation of number of ready parties at the system’s onset (party=CPU)
- T_0: initial target
- τ: recalculation threshold parameter (in Bitcoin: 4)
- T: target in effect

- $n = m/(pTM)$: the “effective” number of parties in the epoch
  - M: last epoch’s duration based on block timestamps
  - pT: probability of a single party being successful in PoW in a round

Clay pigeon shooting game

Clay pigeon shooting game

- Suppose you shoot on targets successively against an opponent
  - your success probability: $p = 0.3$
  - your opponent's success probability: $q = 0.4$
  - you shoot in sequence 1000 targets
  - winner is the one that got the most hits
- What is your probability of winning?

Analysis, I

- Consider the random variable $Z_i = X_i + X_{i-1} + \ldots + X_1$
    - X_i=1 you get ahead, probability *p(1-q)*
    - X_i=-1 opponent gets ahead, probability *q(1-p)*
    - X_i=0 draw, probability *pq + (1-p)(1-q)*
- $E[X_i] = p(1-q) - q(1-p) = p - q < 0$
- $E[Z_n] = n(p - q)$
- Hoeffding inequality states: $\Pr[Z_n => \varepsilon + n(p - q)] <= \exp(- \varepsilon^2 / 2Bn)$, for $\varepsilon > 0$ and $-B <= X_i <= B$
- Set $\varepsilon = n(q - p)$, B=1, and we obtain $\Pr[Z_n => 0] <= \exp(- (q-p)^2 n / 2)$
- For our numbers (q=0.4, p=0.3, n=1000), $\Pr[Z_n => 0] <= 0.007 <= 1\%$

Analysis, II

- Now you are given a choice:
    - decrease the size of the clay pigeon target by a ratio β
    - augment your “kills” by multiplying with $1/\beta$
    - your accuracy is linear with β
    - your opponent will keep playing in the same way as before
- Do you prefer to play like this or it makes no difference?

Analysis, III

- Consider the random variable $Z_i = X_i + X_{i-1} + \ldots + X_1$
  - $X_i=1/\beta$ you get ahead, probability $\beta p(1-q)$
  - $X_i=-1$ opponent gets ahead, probability $q(1-\beta p)$
  - $X_i=1/\beta-1$ both win, probability βpq, and $X_i=0$ both lose, probability $(1-\beta p)(1-q)$
- $E[X_i] = (1/\beta)\beta p(1-q) - q(1-\beta p) + (1/\beta-1)\beta pq = p - q < 0$
- $E[Z_n] = n(p - q)$
- Hoeffding inequality states: $\Pr[Z_n => \varepsilon + n(p - q)] <= \exp(- \varepsilon^2 / 2Bn)$, for $\varepsilon>0$ and $-B <= X_i <= B$
- Set $\varepsilon = n(q - p)$, $B=1/\beta$, and we obtain $\Pr[Z_n => 0] <= \exp(- \beta(q-p)^2 n/ 2)$
- Observe: as β -> 0, tail bound deteriorates to <=1.

The Difficulty Raising Attack

- The recalculation threshold (τ) is essential
- Without it, an adversary that has a minority of hashing power:
    - Creates a private, artificially difficult chain
    - Similar to clay pigeon shooting game, this increases the variance in its block production rate
    - Overcoming the chain of the honest parties becomes a non-negligible event

[B13] Lear Bahack. Theoretical Bitcoin Attacks with less than Half of the Computational Power (draft)