Blockhains and Distributed Ledgers
Lecture 02

Aggelos Kiayias

THE UNIVERSITY of EDINBURGH

Lecture 02

- The consensus layer.
- Basic Properties.
- Proof of work.

Motivation for the Consensus Layer, I

- A transaction history needs to be agreed by all participants.

- Participants may have diverging interests in terms of the history of transactions.

Motivation for the Consensus Layer, II

I prefer this!

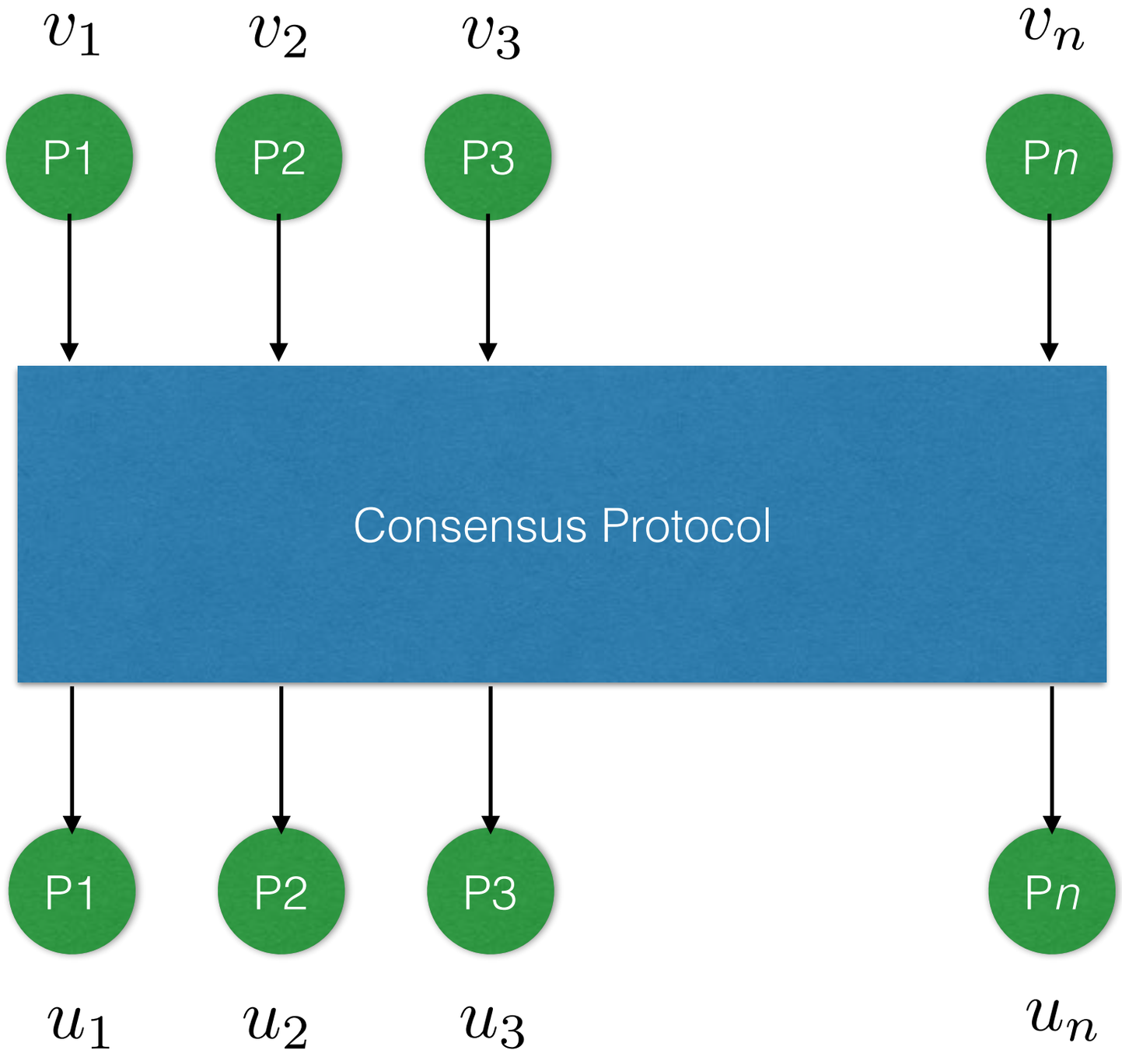
I prefer this!

I prefer this!

The Consensus Problem

High level requirement

agree on the output



Study initiated by Lamport, Pease, Shostak 1982

Problem Statement

- A number (say, *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. $\mathsf{H}, |\mathsf{H}| = n - t$

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

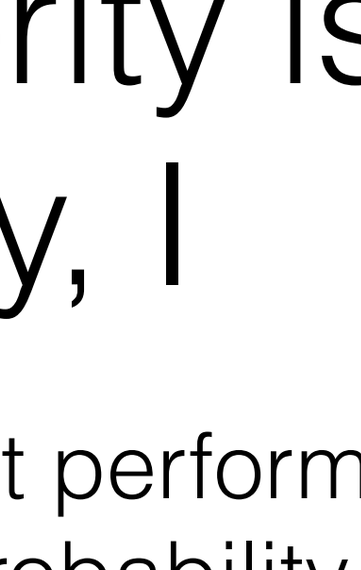
Consider an adversary that performs one of the following with probability 1/3

$\emptyset$

$n/2$

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

$n/2$



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

Honest Majority is Necessary, II

- If the adversary corrupts A0, then output of honest parties (that belong to A1) should be 1.

- If the adversary corrupts A1, then output of honest parties (that belong to A0) should be 0.

- If the adversary corrupts no-one, then the output of all parties should be equal.

Is Honest Majority Sufficient?

- Two important scenarios have been considered in the consensus literature.

- Point to point channels. **No setup.** $t < \frac{n}{3}$

- Point to point channels. **With setup.** $t < \frac{n}{2}$

The setup enables provides a correlated private initialization string to each participant. It is assumed to be honestly produced.

Setup in Consensus

- A "public-key infrastructure" (PKI).
  - parties have signing and verification keys for a digital signature scheme.
  - Each party knows every other party’s verification key.

Enter Bitcoin

- (2008-2009)
- Important concepts used
  - blockchain data structure.
  - proof of work (POW).
- both known and studied earlier, but put in combination for a novel application.

Blocks

$$B = \langle s, x, ctr \rangle$$

pointer
to previous

content

counter
(proof of work witness)

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

hash functions

target

upper bound

Blockchain

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

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

$$\vdots$$

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

genesis block

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

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

head

$$\mathbf{x}_{\mathcal{C}} = \langle x_0, x_1, \ldots, x_n \rangle$$

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

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.”

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 $\mathcal{C}$.

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

Algorithm 2 The function that finds the "best" chain, parameterized by function max(·). The input is $\{\mathcal{C}_1, \ldots, \mathcal{C}_k\}$.

```
1: function maxvalid(C_1, ..., C_k)
2:     temp ← ε
3:     for i = 1 to k do
4:         if validate(C_i) then
5:             temp ← max(C_i, temp)
6:         end if
7:     end for
8:     return temp
9: end function
```

Algorithm 3 The *proof of work* function, parameterized by q, T and hash functions $H(\cdot), G(\cdot)$. The input is $(x, \mathcal{C})$.

1: **function** pow$(x, \mathcal{C})$
2: **if** $\mathcal{C} = \varepsilon$ **then** ▷ Determine proof of work instance
3: $s \leftarrow 0$
4: **else**
5: $\langle s', x', ctr' \rangle \leftarrow \text{head}(\mathcal{C})$
6: $s \leftarrow H(ctr', G(s', x'))$
7: **end if**
8: $ctr \leftarrow 1$
9: $B \leftarrow \varepsilon$
10: $h \leftarrow G(s, x)$
11: **while** $(ctr \leq q)$ **do**
12: **if** $(H(ctr, h) < T)$ **then** ▷ This $H(\cdot)$ invocation subject to the q-bound
13: $B \leftarrow \langle s, x, ctr \rangle$
14: **break**
15: **end if**
16: $ctr \leftarrow ctr + 1$
17: **end while**
18: $\mathcal{C} \leftarrow \mathcal{C}B$ ▷ Extend chain
19: **return** $\mathcal{C}$
20: **end function**

Algorithm 4 The Bitcoin backbone protocol, parameterized by the *input contribution function* $I(\cdot)$ and the *chain reading function* $R(\cdot)$.

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

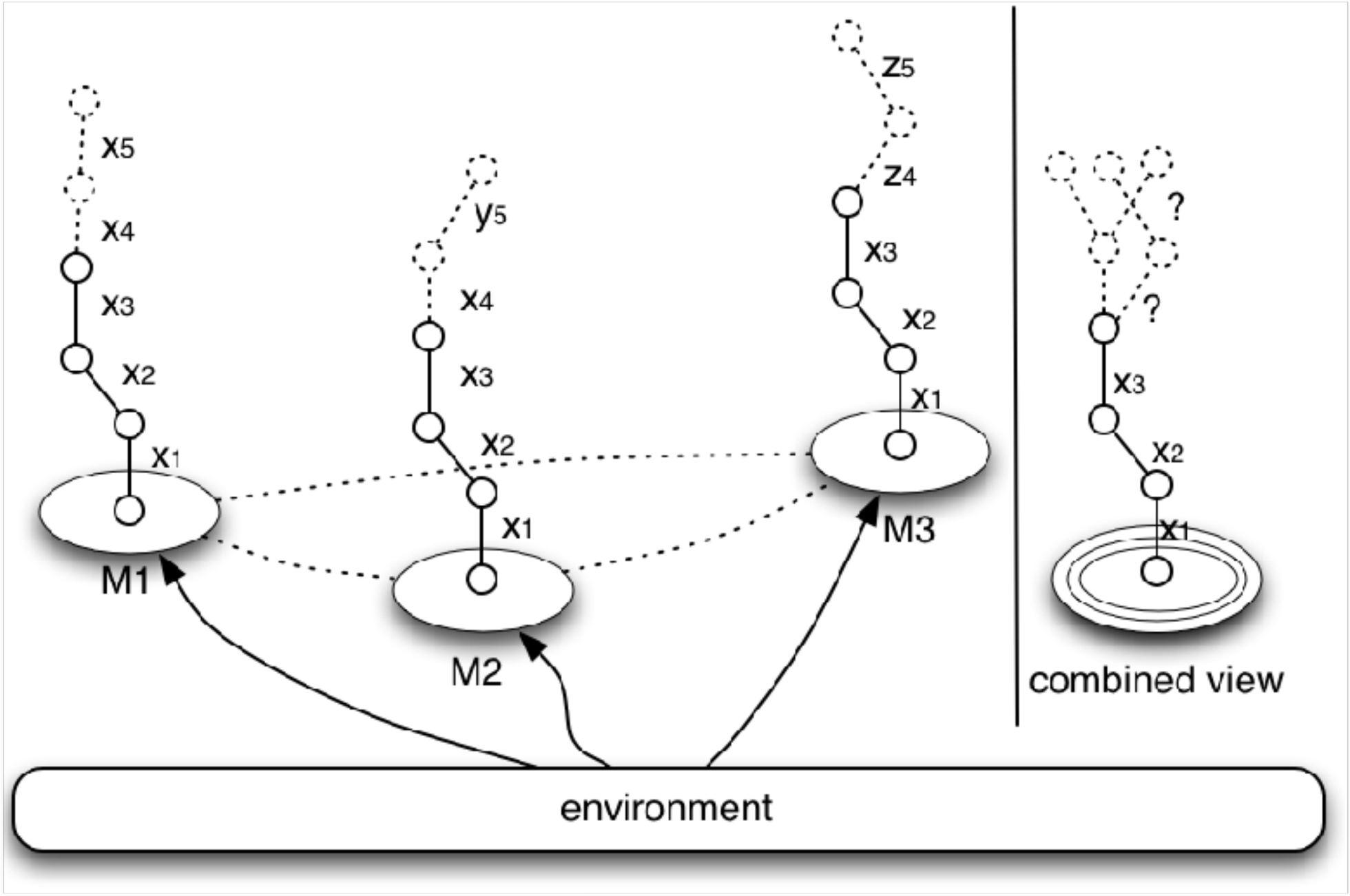
Adversary Model

- Assume there are *n* parties running of the protocol
  - synchronously.
    - each one 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).

Basic Properties

- Common Prefix
- Chain Quality
- Chain Growth

Common Prefix, I



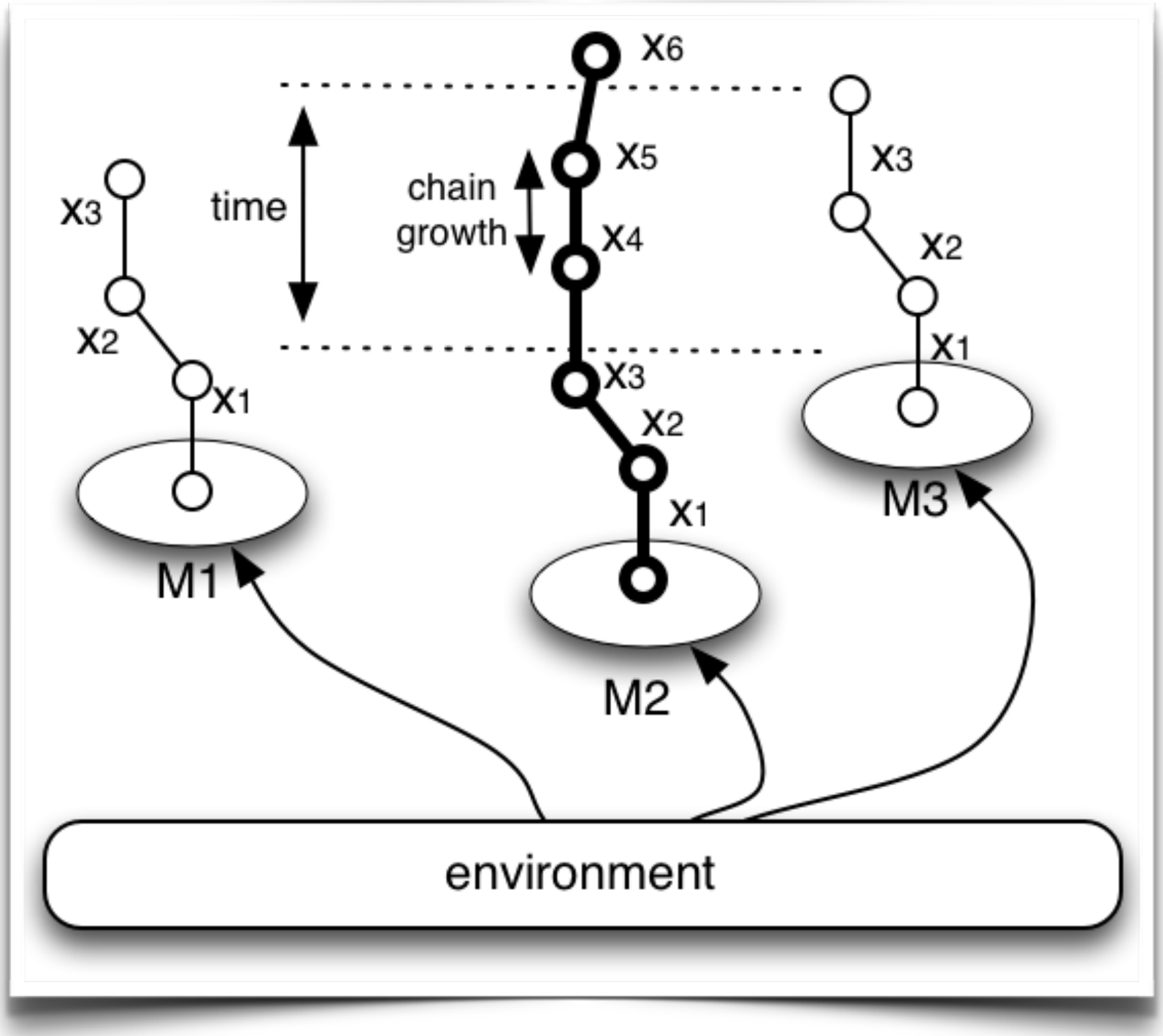
Common Prefix, II

(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*

Chain Growth, I



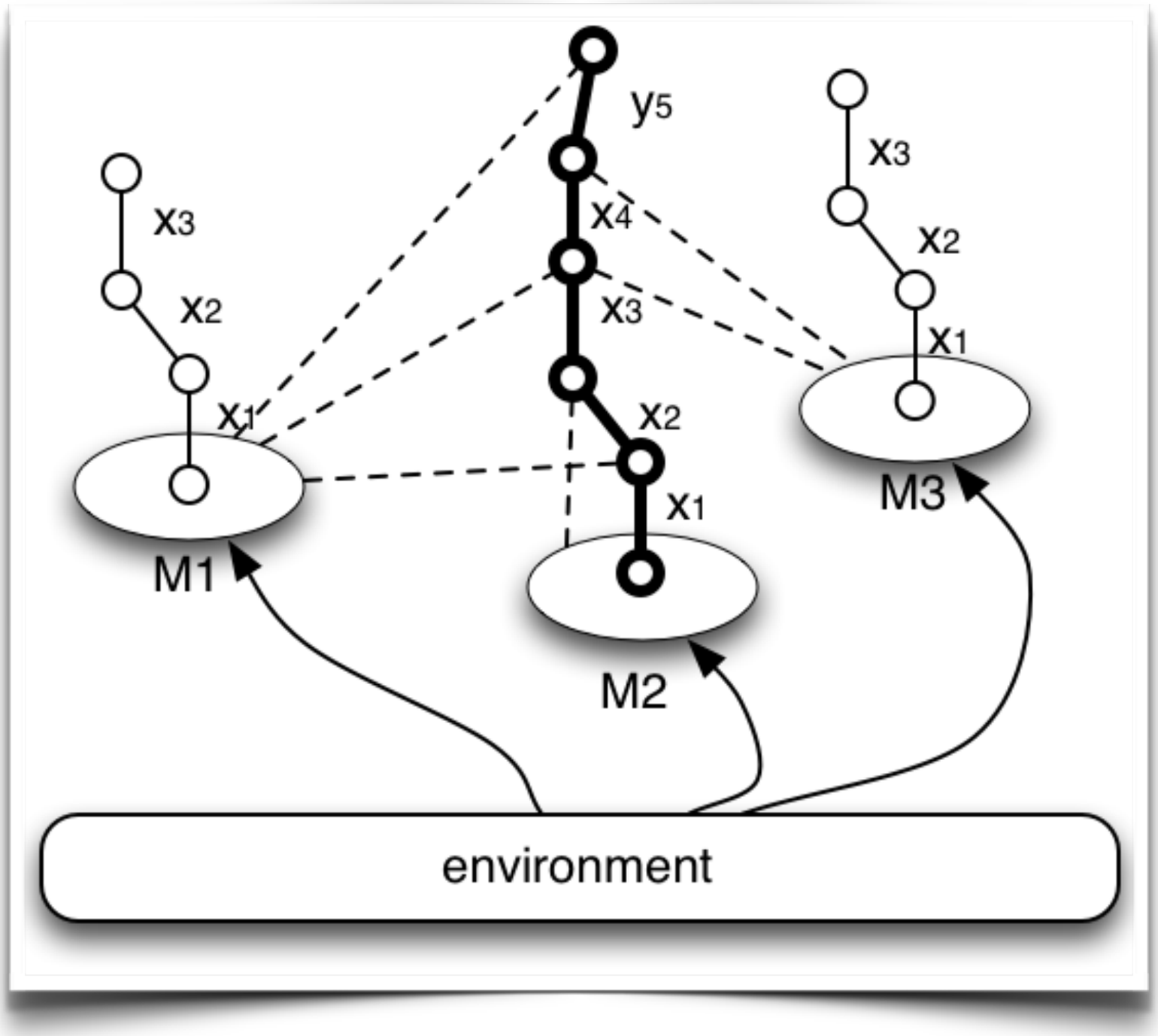
Chain Growth, II

Parameters $\tau \in (0,1), s \in \mathbb{N}$
$\forall r_1, r_2$ honest player P with chains $\mathcal{C}_1, \mathcal{C}_2$
$r_2 - r_1 \geq s \implies |\mathcal{C}_2| - |\mathcal{C}_1| \geq \tau s$

- 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

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 probability holds true probabilistically with an error that exponentially decays in ℓ

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

Robust Transaction Ledger

- The three properties can provide a ledger with the following characteristics.
  - persistence: Transactions are organized in a “log” and honest nodes agree on it.
  - liveness: New transactions are included in the log nodes, after a suitable period of time.

Establishing the ledger

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

Ledger and 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?

The Bitcoin Setting for Consensus

- The bitcoin setting is a different compared to what has been considered classically for the consensus problem.
    - Communication is by **diffusion**. (no point-to-point channels).
        - Message delivery is assumed but message origins and recipient list are not specified.
- The protocol setup is not a private correlated setup (digital signatures are not used to authenticate miners)
    - A public setup is assumed (in the form of the genesis block)

Consensus <= Ledger, I

- Using a ledger (or directly the underlying blockchain) to solve consensus.

- Let's focus on *binary* consensus for simplicity.

Consensus<= Ledger, II

Re: Bitcoin P2P e-cash paper

Satoshi Nakamoto | Thu, 13 Nov 2008 19:34:25 -0800

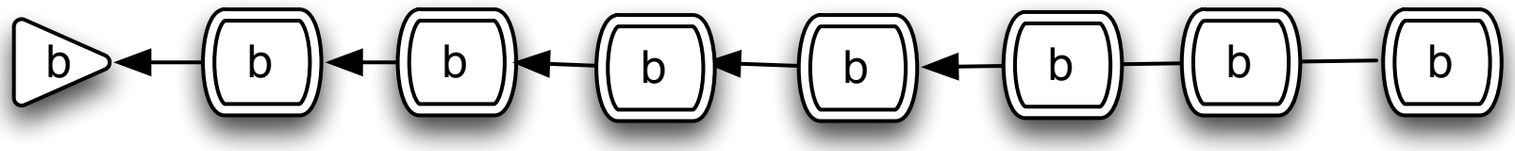
```
James A. Donald wrote:
> It is not sufficient that everyone knows X. We also
> need everyone to know that everyone knows X, and that
> everyone knows that everyone knows that everyone knows X
> - which, as in the Byzantine Generals problem, is the
> classic hard problem of distributed data processing.

The proof-of-work chain is a solution to the Byzantine Generals' Problem.  I'll
try to rephrase it in that context.

A number of Byzantine Generals each have a computer and want to attack the
King's wi-fi by brute forcing the password, which they've learned is a certain
number of characters in length.  Once they stimulate the network to generate a
packet, they must crack the password within a limited time to break in and
erase the logs, otherwise they will be discovered and get in trouble.  They
only have enough CPU power to crack it fast enough if a majority of them attack
at the same time.

They don't particularly care when the attack will be, just that they all agree.
 It has been decided that anyone who feels like it will announce a time, and
whatever time is heard first will be the official attack time.  The problem is
```

https://www.mail-archive.com/cryptography@metzdowd.com/msg09997.html

Consensus<=Ledger, III

Nakamoto's suggestion

Valid chains have a single bit

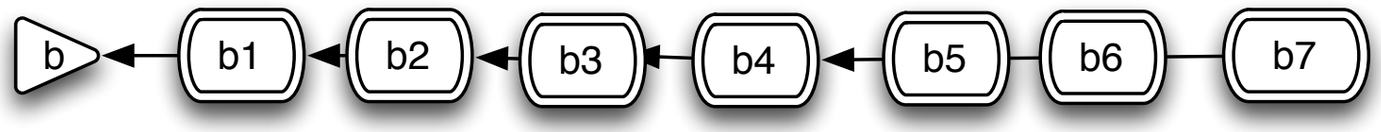


It works .. but only with constant probability of success
(not overwhelming)

Consensus<=Ledger, IV

A (1/3) “consensus protocol” (GKL protocol 1)

Valid chains have different bits with each block containing the bit of the block producer



It works .. but only up to 1/3 adversarial power.

Consensus<=Ledger, V

- Main obstacle (intuitively)
  the blockchain protocol does not provide sufficiently high chain quality.

- … we cannot guarantee that we have enough blocks originating from honest parties.

- How to fix this?

Consensus<=Ledger, VI

- The *n* parties build a ledger but now generate transactions based on POW that contain their inputs.

- Once the blockchain is long enough the parties' prune the last *k* blocks and output the majority of the values drawn from the set of transactions in the ledger.

Beware! given that POW's are used for two different tasks how do we prevent the attacker from shifting its hashing power from the one to the other?

2-for-1 POWs

parallel composition of POW protocols

$h \leftarrow G(s, x)$
if $H(h, ctr) < T \ldots$

$h' \leftarrow G(s', x')$
if $H(h', ctr') < T' \ldots$

$h \leftarrow G(s, x)$
$h' \leftarrow G(s', x')$
$w \leftarrow H(h, h', ctr)$
if $w < T \ldots$
if $[w]^{\mathsf{R}} < T' \ldots$

Not Secure

given $((s, x), ctr)$
verify:
$H(G(s, x), ctr) < T$
given $((s', x'), ctr')$
verify:
$H(G(s', x'), ctr') < T'$

given
$((*, *), (s', x'), ctr')$
verify:
$[H(G(*, *), G(s', x'), ctr')]^{\mathsf{R}} < T'$

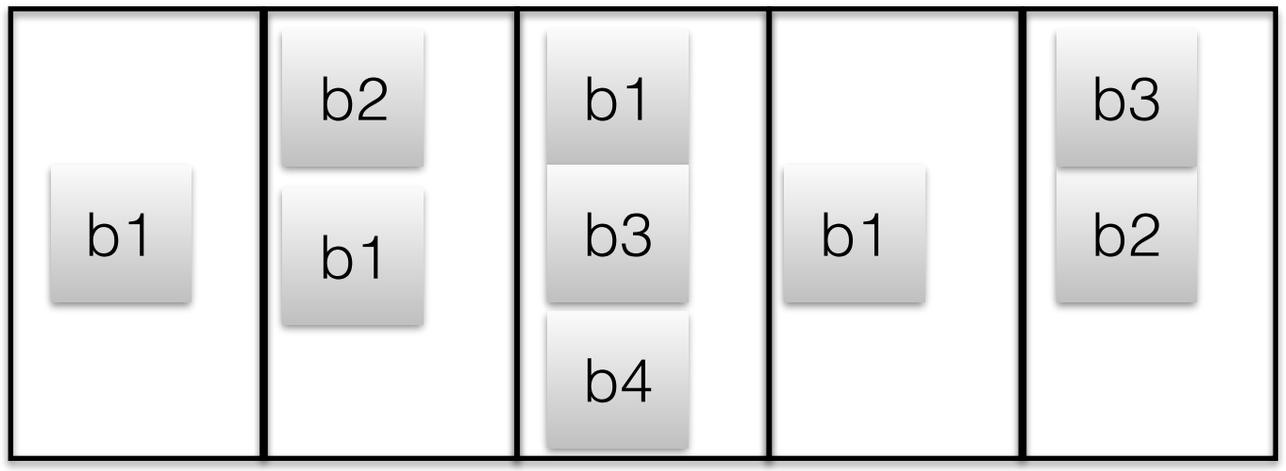
Key Observation

- **Lemma.** Finding POW solution for either “side” of the POW protocol is an independent event.

- [note that it works only for a suitable choice of T and T’]

GKL consensus protocol 2

parties **mine** POWs for each block (as in bitcoin backbone)



parties **mine** POWs for their input in {0,1} (input+"nonce")

they keep **transmitting** POW-inputs, until they are accepted.

Finally, after the blockchain **grows sufficiently**,
they **chop** the last *k* blocks and return the **majority among unique inputs** in the (common) prefix.

Why it works

Key proof idea

recall : the output of the protocol is the **majority** bit from the inputs in the **common prefix**

The **(minuscule) chain quality** of the protocol, given parties transmit inputs until they are accepted, it guaranteeing that they **will be included** eventually.

Moreover, because each input, has a POW, the **majority** of unique POW-inputs will be **originating from honest parties** in any **sufficiently long** part of the chain (such as the common prefix)

Ledger <= Consensus

- (Potentially) simpler:
  - run consensus for each set of transactions (blocks) with each party proposing a set of transactions.

- plain **validity** is insufficient.

End of lecture 02

- Next lecture:
  - The ledger as a platform.
  - Smart Contracts
  - Ethereum