Proof of Luck:

an Efficient Blockchain Consensus Protocol

Mitar Milutinovic, Warren He, Howard Wu, Maxinder Kanwal

Outline

Background: blockchains, consensus, and SGX

Existing consensus mechanisms

Our paper:

- 3 *basic* consensus primitives
- Proof of Luck

Conclusion

Outline

Background: **blockchains**, consensus, and SGX

Existing consensus mechanisms

Our paper:

- 3 *basic* consensus primitives
- Proof of Luck

Conclusion

Background: blockchains

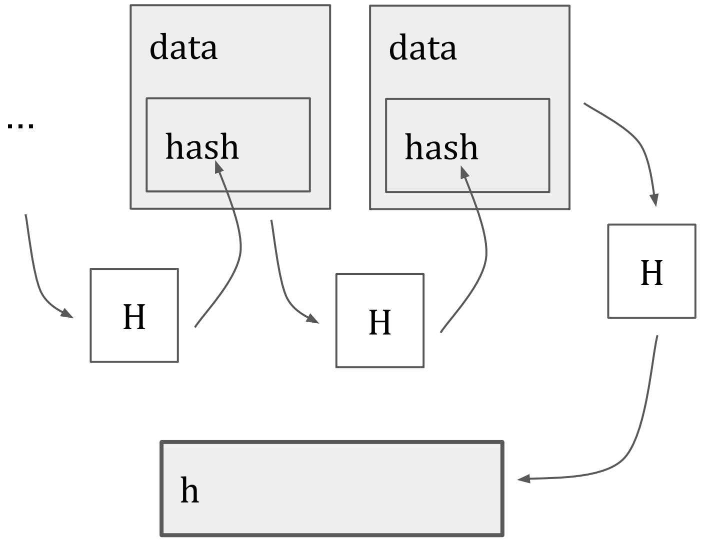
block = (data, H(previous block))

1 hash protects integrity of entire chain

Efficient to append

Efficient to verify recent blocks

Use case: append-only log



Background: blockchains

Use case: append-only *transaction* log

Remember previous payments
to know who has how much money

Still something missing:
What if you know multiple valid blockchains?

Outline

Background: blockchains, **consensus**, and SGX

Existing consensus mechanisms

Our paper:

3 *basic* consensus primitives

Proof of Luck

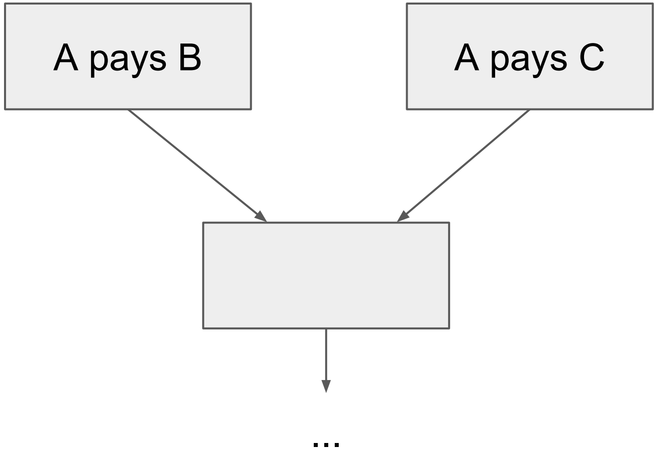
Conclusion

Background: consensus

Two valid chains, same ancestry

Whom has A paid?

Has A even paid anyone?



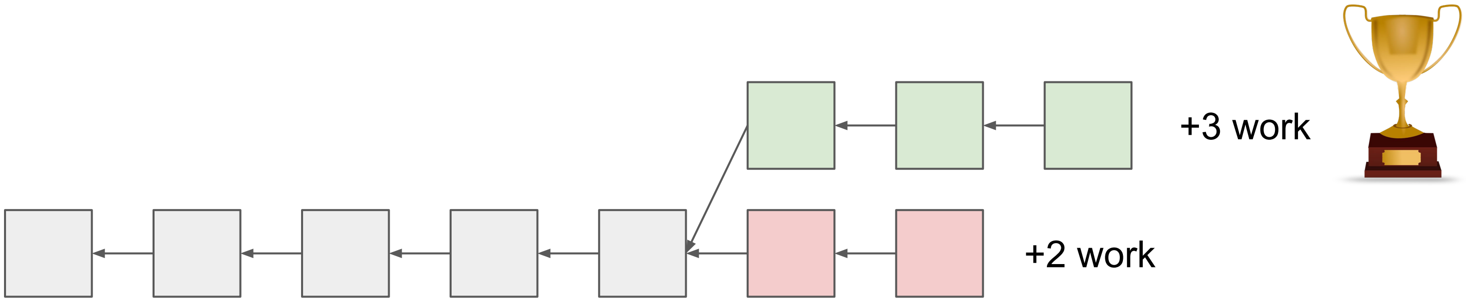
Background: consensus

One approach: *proof of work*

Each block must contain a proof of work

Bitcoin uses a partial hash preimage problem

Prefer the chain with the most work



Background: consensus

Issues with Bitcoin's consensus mechanism:

- To prevent ties, it's slow—10 minutes per block on average
- Time per block varies by chance
- Takes a lot of energy to do the work

Motivation: could do better with trusted execution
SGX is available in consumer CPUs

Outline

Background: blockchains, consensus, and **SGX**

Existing consensus mechanisms

Our paper:

- 3 *basic* consensus primitives
- Proof of Luck

Conclusion

Background: SGX

A *trusted execution environment*

Remote attestation: one can verify* that
a **specific computation**
ran on **suitable hardware** and
produced a **specific result**.

*Provided they trust in the platform vendor, Intel in the case of SGX

Outline

Background: blockchains, consensus, and SGX

Existing consensus mechanisms

Our paper:

3 *basic* consensus primitives

Proof of Luck

Conclusion

Existing consensus mechanisms

Proof of work - variations for useful work

Proof of Stake / Proof of Burn - depends on specific incentives

Byzantine fault tolerance - fast, participants known, adversary < ⅓

Intel Sawtooth Lake - developed concurrently, simulates Bitcoin mining,
more mature analysis of compromised CPUs

Outline

Background: blockchains, consensus, and SGX

Existing consensus mechanisms

Our paper:

- **3 *basic* consensus primitives**
- Proof of Luck

Conclusion

TEE Proof of Work 💪

Nonce to prevent replay, as usual

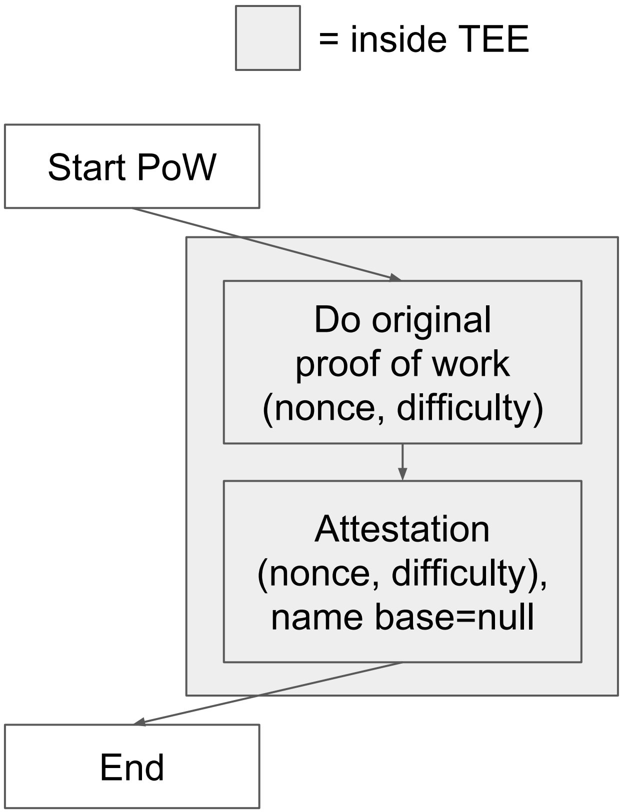
Null *name base*: anonymous proof (more later)

Restricts ASIC use

Can do work that doesn't have efficient verification algorithm

Guaranteed to get a proof after doing work

Still uses lots of energy

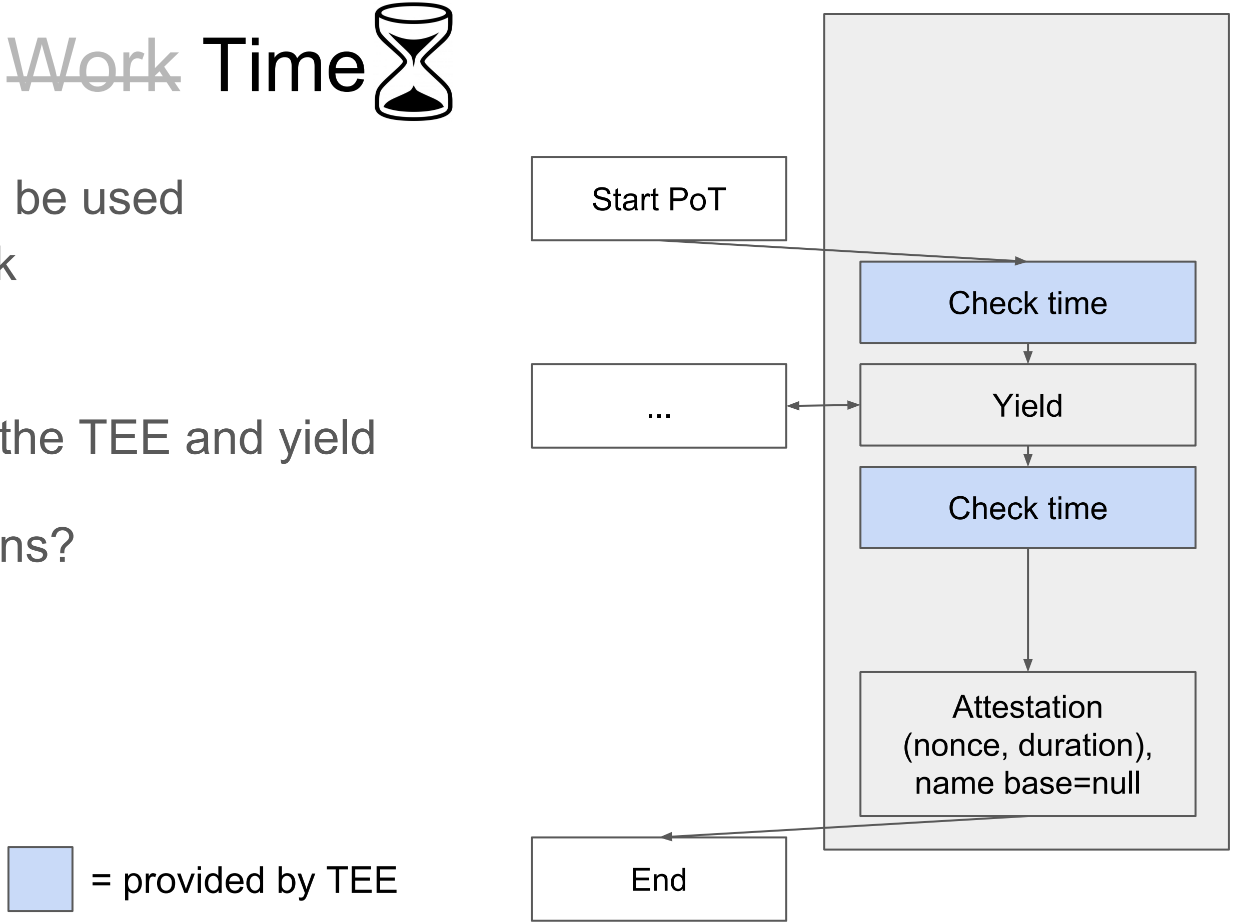


TEE Proof of ~~Work~~ Time ⌛

A busy-wait loop can be used
in TEE-Proof-of-Work

Even better:
just check time from the TEE and yield

Concurrent invocations?



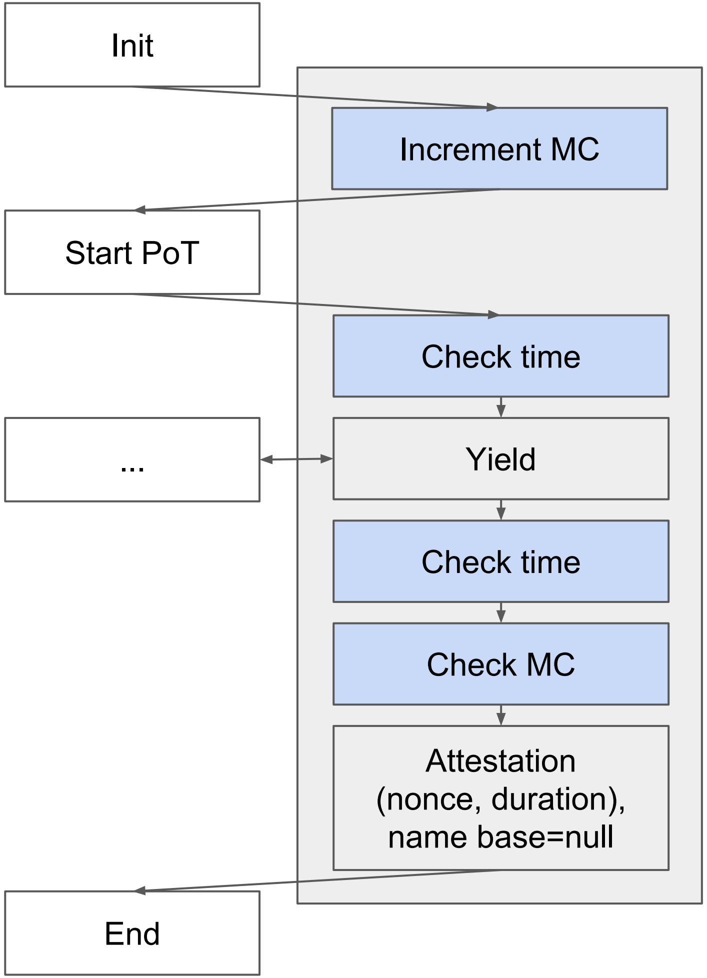
TEE Proof of ~~Work~~ Time ⌛

Concurrent invocations?

Prototype in SGX:
monotonic counters (MC) shared across instances of same enclave

Implement a mutex.

Assumption:
TEE supports this use case



TEE Proof of ~~Work~~ Time

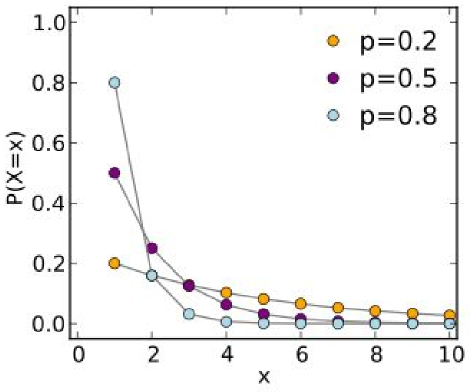
Related: Sawtooth Lake distributed ledger, *Proof of Elapsed Time*

Wait for a randomized amount of time—simulates partial preimage search

```
efc9a5df...
33bf7353...
31a75a03...
598fc24b...
c052d575...
d824325d...
fd3f6615...
f2c4d943...
d9799954...
fb2eb5e0...
439696f5...
c7882894...
00000000...
```

X ~ geometric distribution

$$\Pr[X = x] = (1-p)^{k-1}p$$



https://github.com/intelledger

TEE Proof of ~~Work~~ ~~Time~~ Ownership

Everyone has same amount of time

Boils down to owning capable CPUs

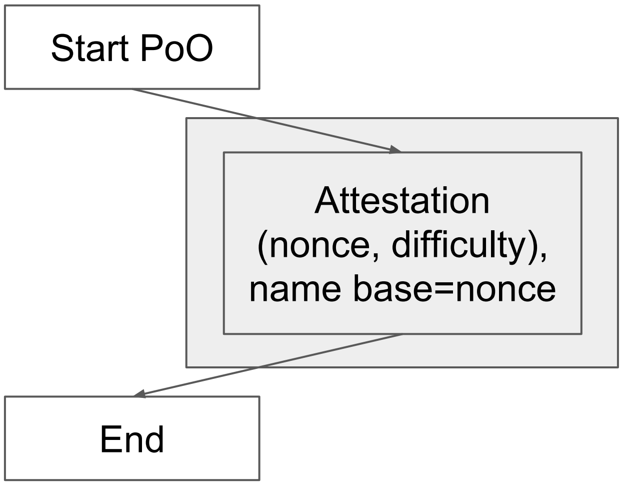
Don’t bother waiting

Name base:

attestation pseudonym = F(name base, CPU’s key)

CPUs vote with attestations

Scalability issue: need to collect all votes



Basic consensus primitives

	ASIC resistant	Energy efficient	Time efficient	Scalable
Bitcoin	no	no	no	yes
TEE Proof of work	yes	no	no	yes
TEE Proof of time	yes	yes	no	yes
TEE Proof of ownership	yes	yes	yes	no

Outline

Background: blockchains, consensus, and SGX

Existing consensus mechanisms

Our paper:

- 3 *basic* consensus primitives
- **Proof of Luck**

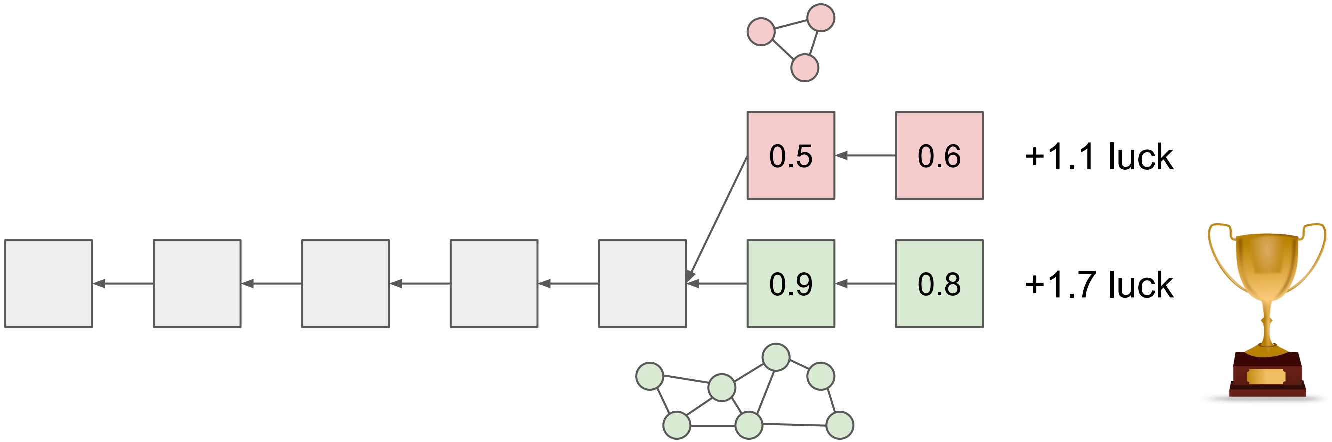
Conclusion

Proof of Luck

Idea: generate random number for each block (assumption: that a TEE can)

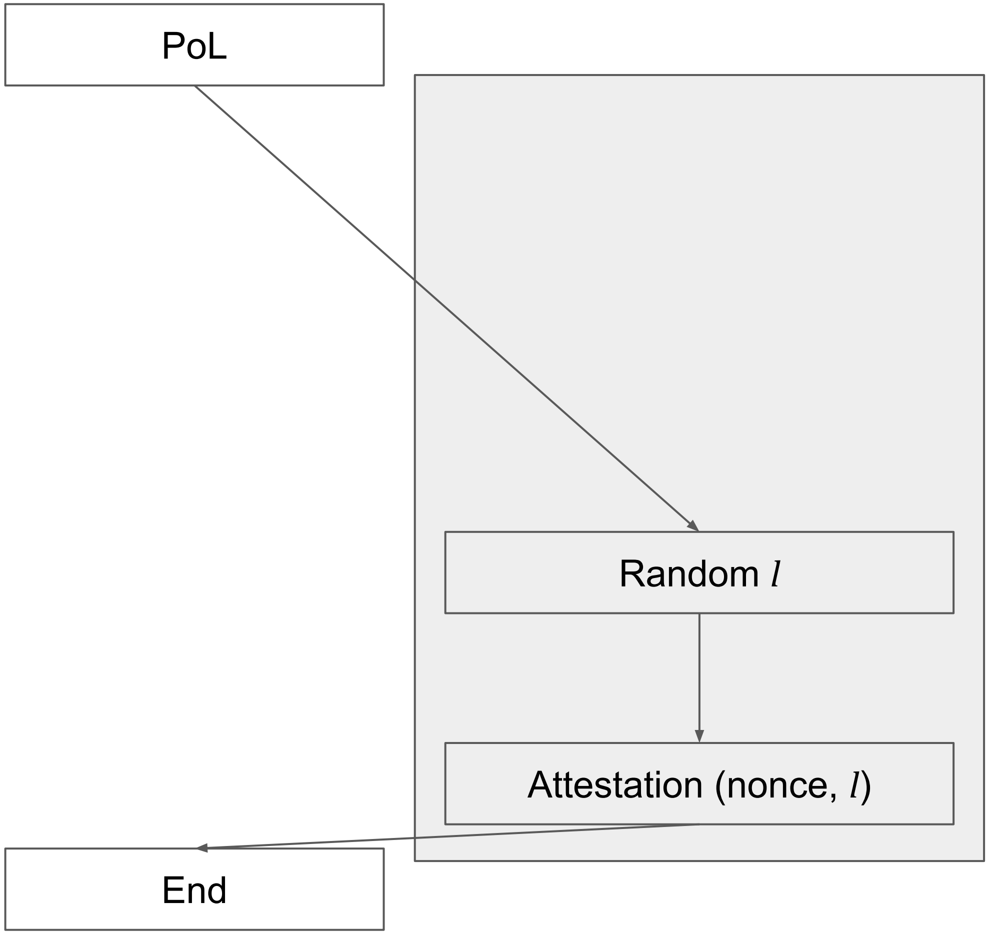
Extend block with highest number, prefer chain with highest total

During network split, larger network will likely generate higher max block



Proof of Luck

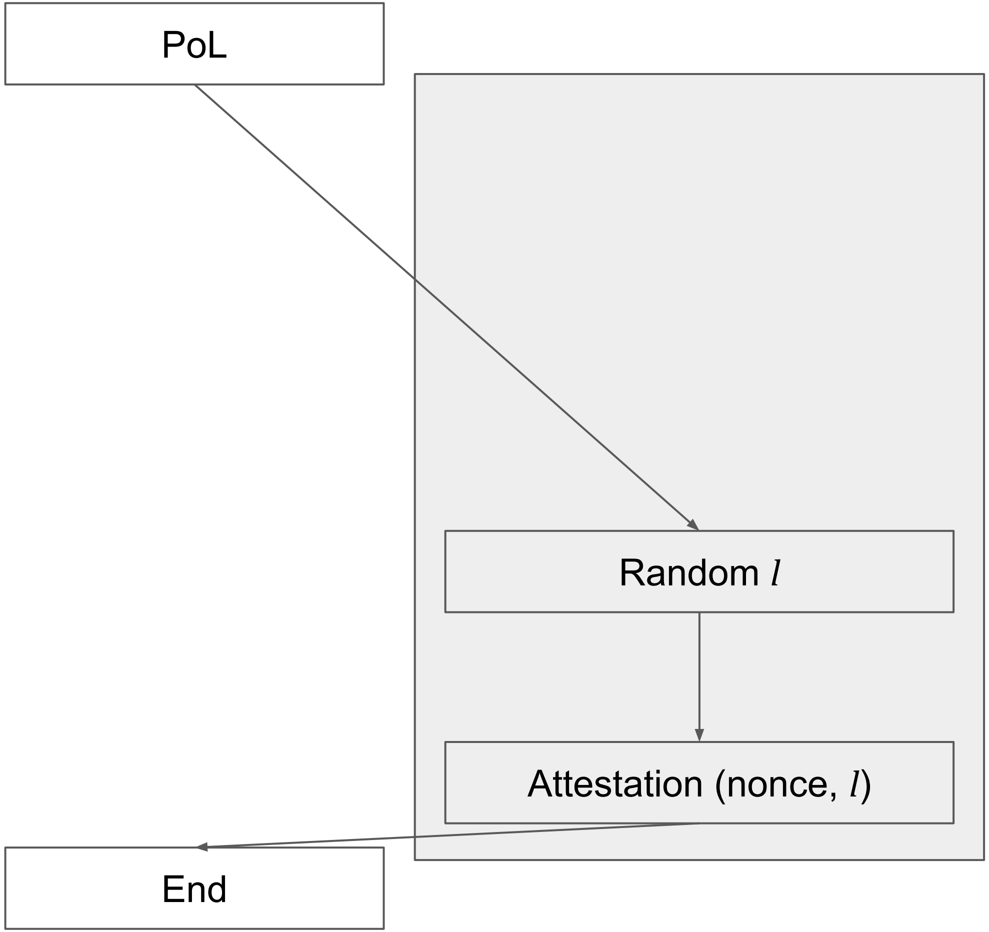
Strawman design:
generate random number,
generate attestation



Proof of Luck

Problem 1:
becomes proof of work

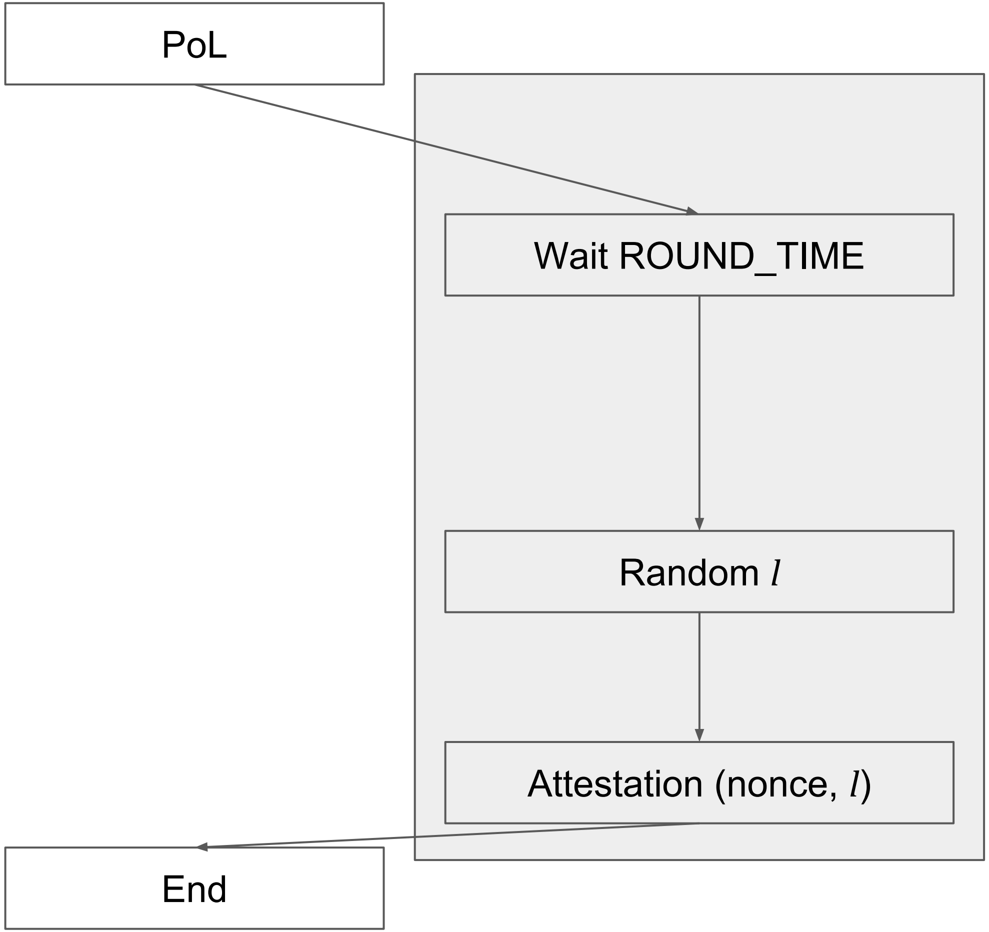
Low number? Restart



Proof of Luck

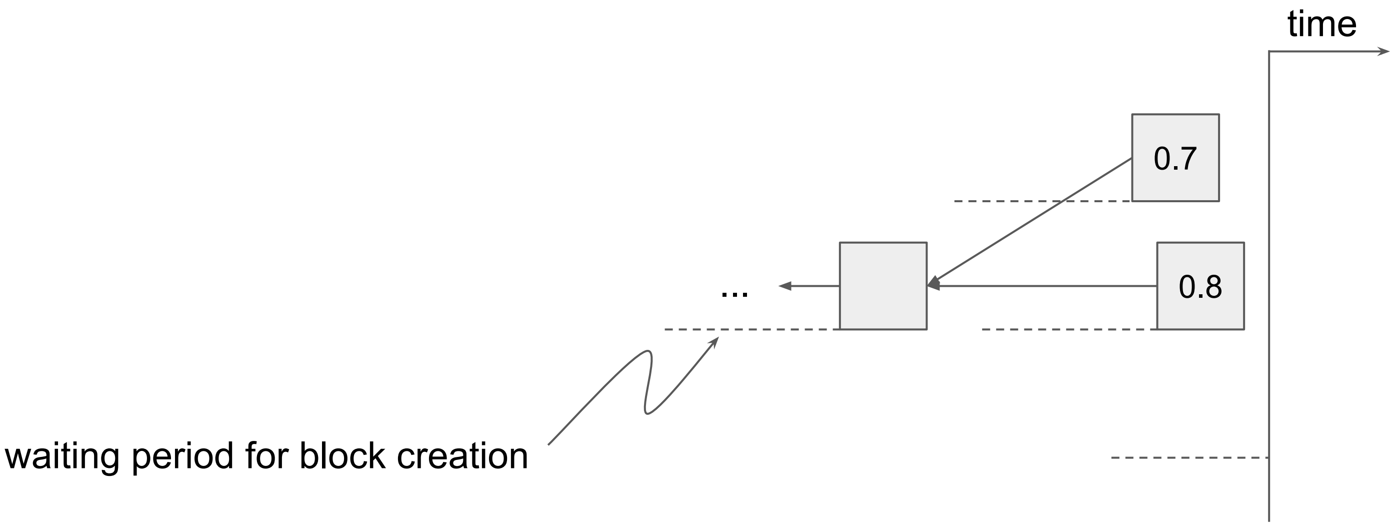
Problem 1:
becomes proof of work

Solution:
must wait for some time,
a “round time”



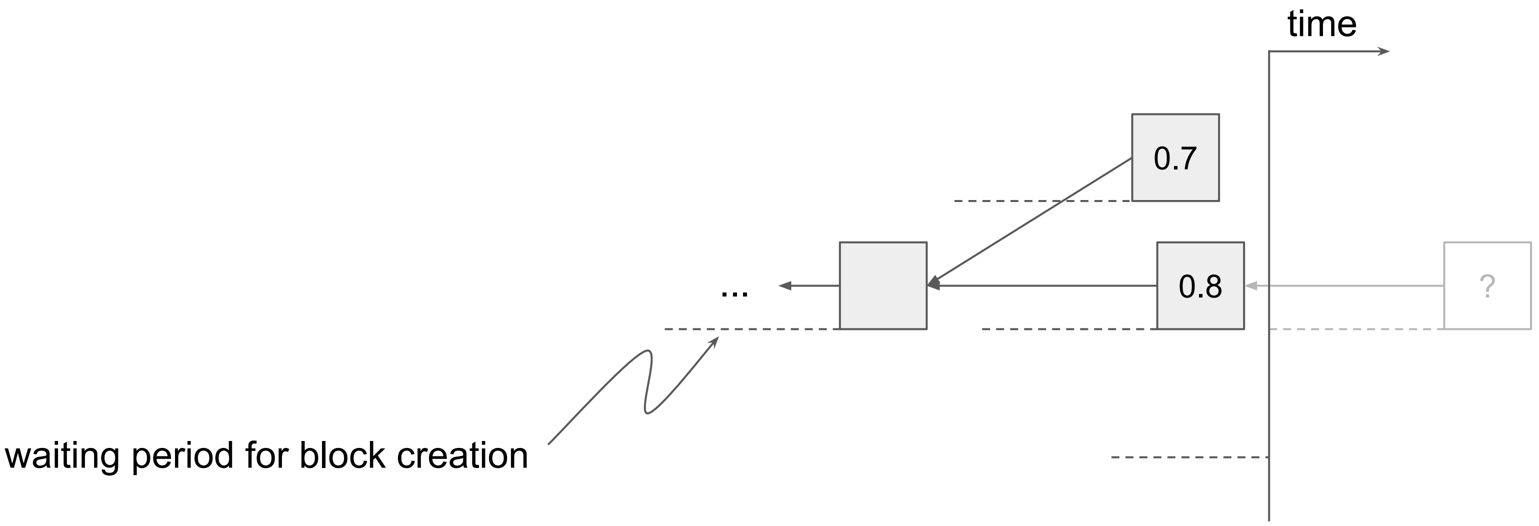
Proof of Luck

Problem 2:
unsynchronized clocks waste luck



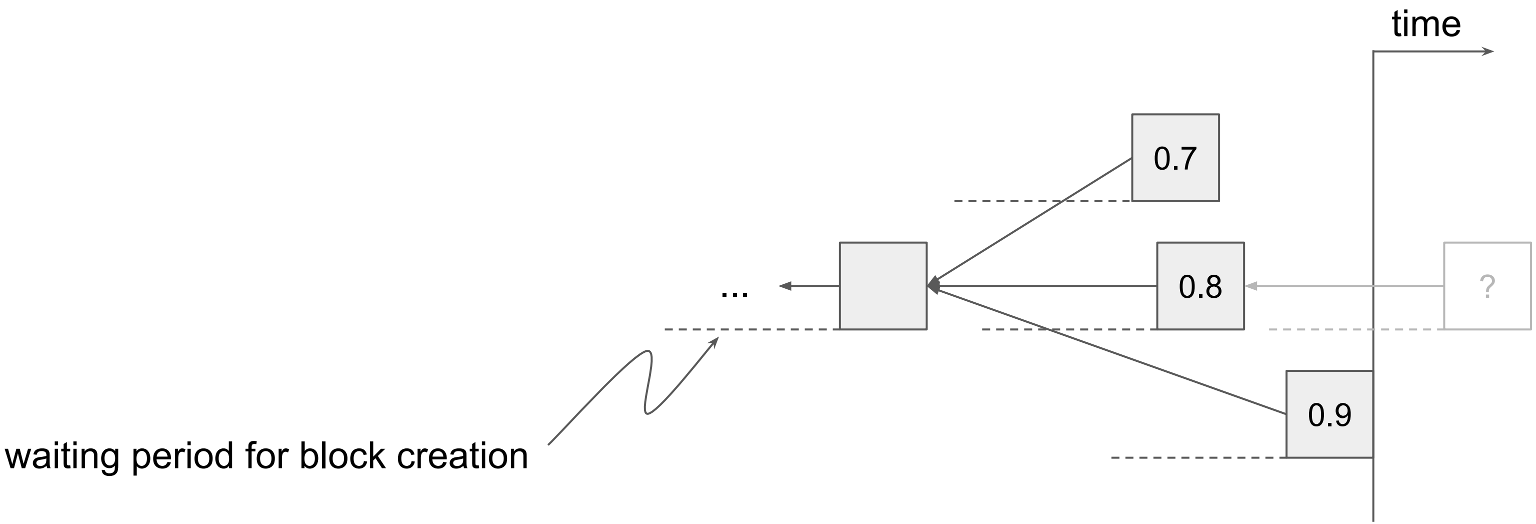
Proof of Luck

Problem 2:
unsynchronized clocks waste luck



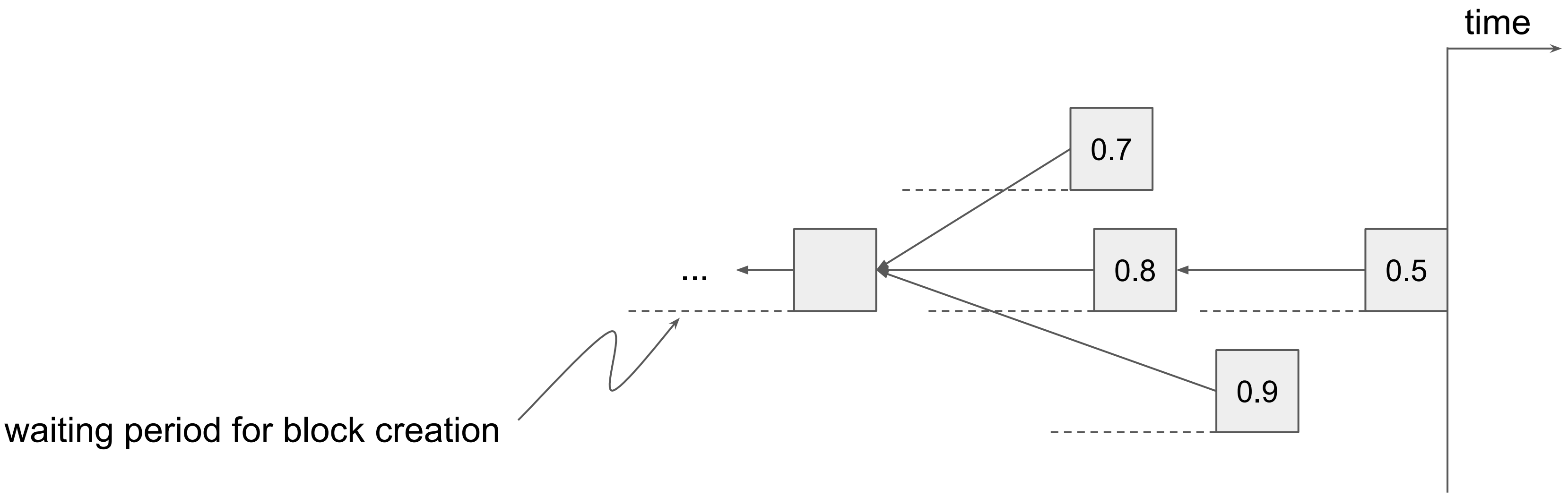
Proof of Luck

Problem 2:
unsynchronized clocks waste luck



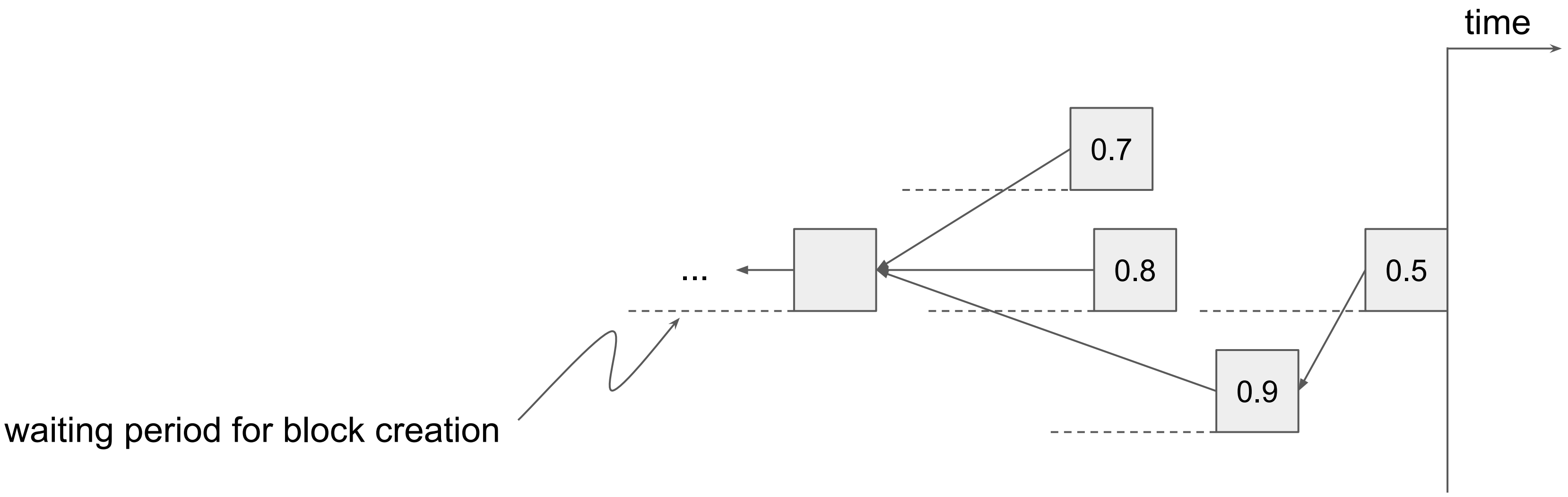
Proof of Luck

Problem 2:
unsynchronized clocks waste luck



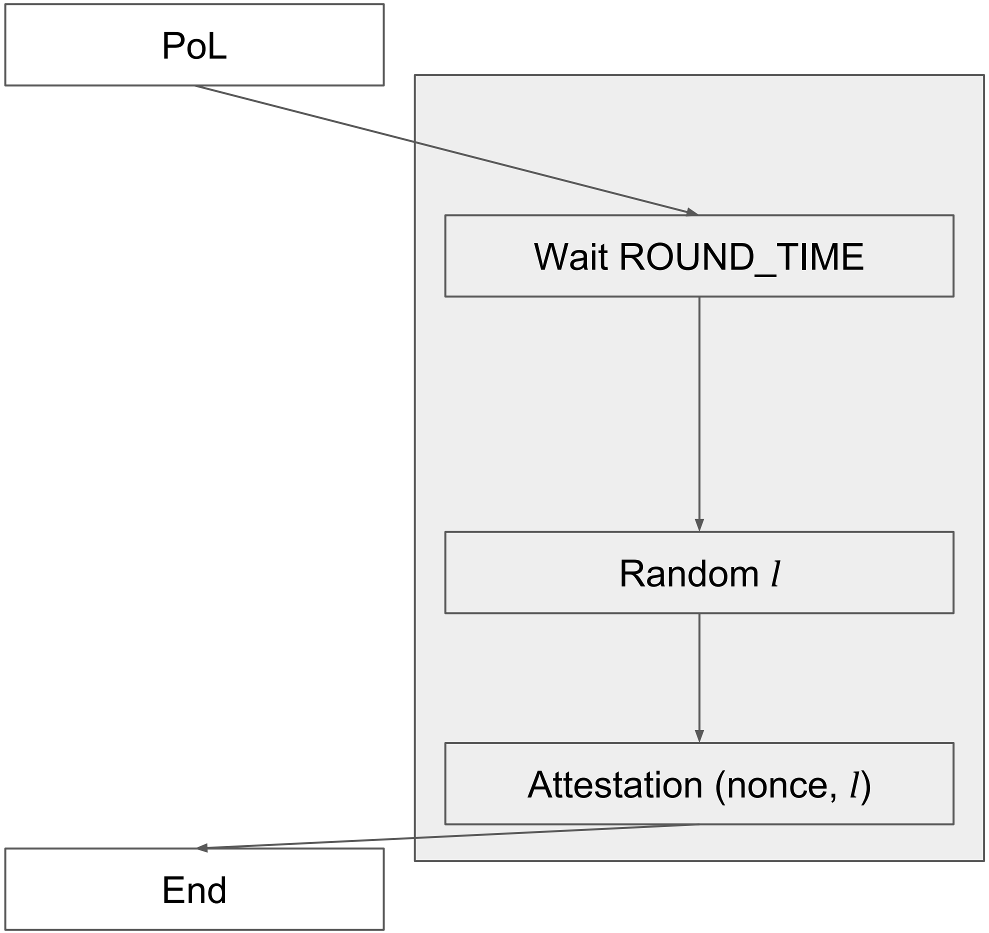
Proof of Luck

Problem 2:
unsynchronized clocks waste luck



Proof of Luck

Problem 2:
unsynchronized clocks waste luck

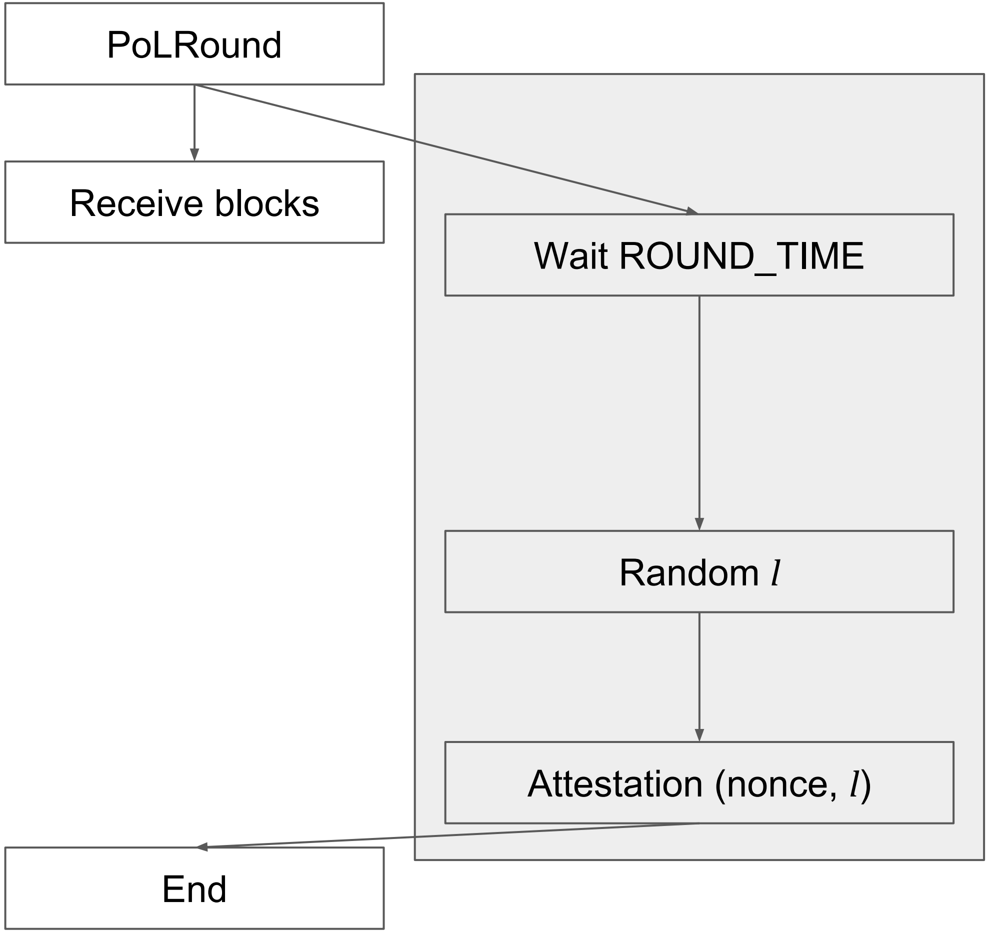


Proof of Luck

Problem 2:
unsynchronized clocks waste luck

Solution:

- Continue to receive competing blocks during ROUND_TIME

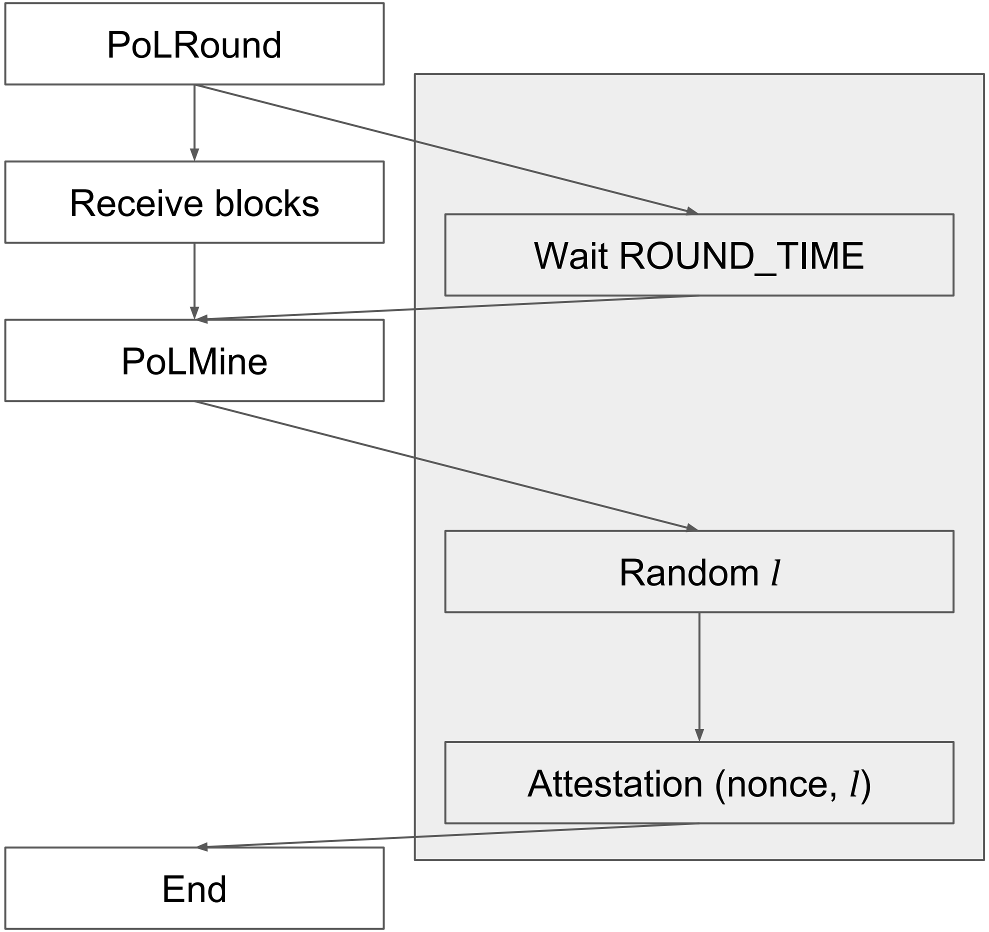


Proof of Luck

Problem 2:
unsynchronized clocks waste luck

Solution:

- Continue to receive competing blocks during ROUND_TIME
- After waiting, have a chance to switch

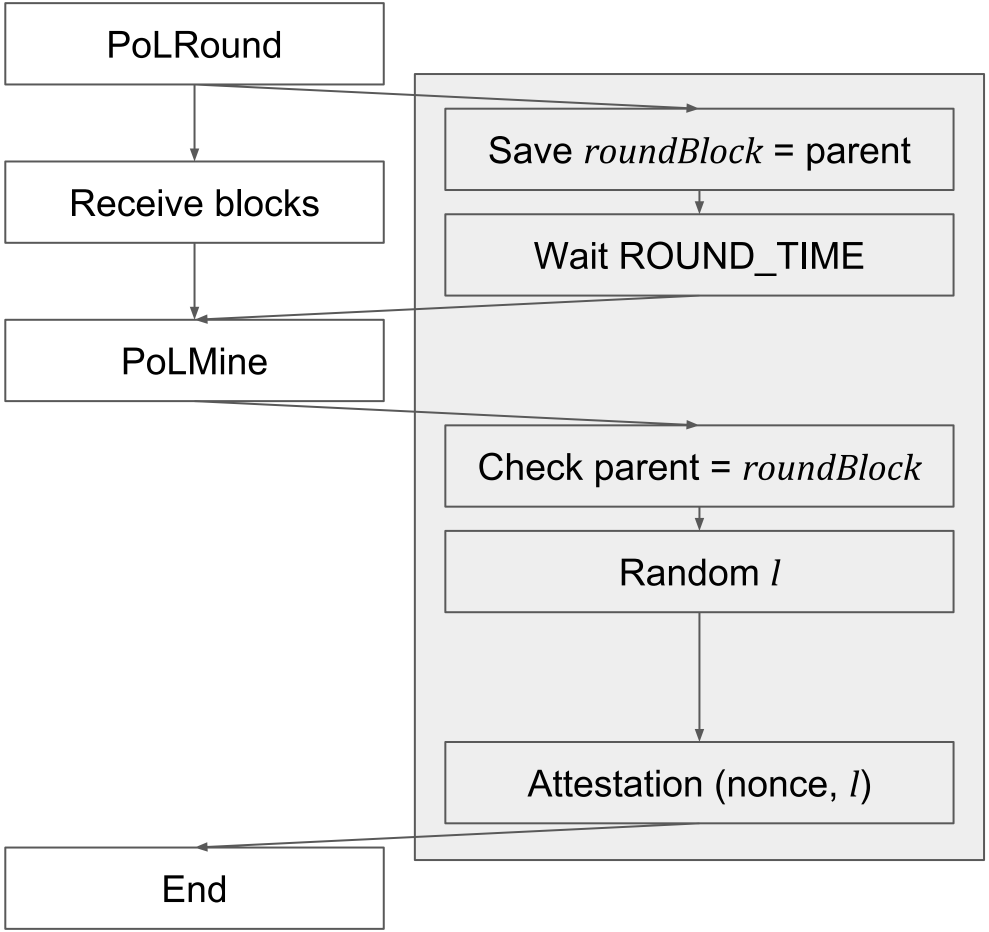


Proof of Luck

Problem 2:
unsynchronized clocks waste luck

Solution:

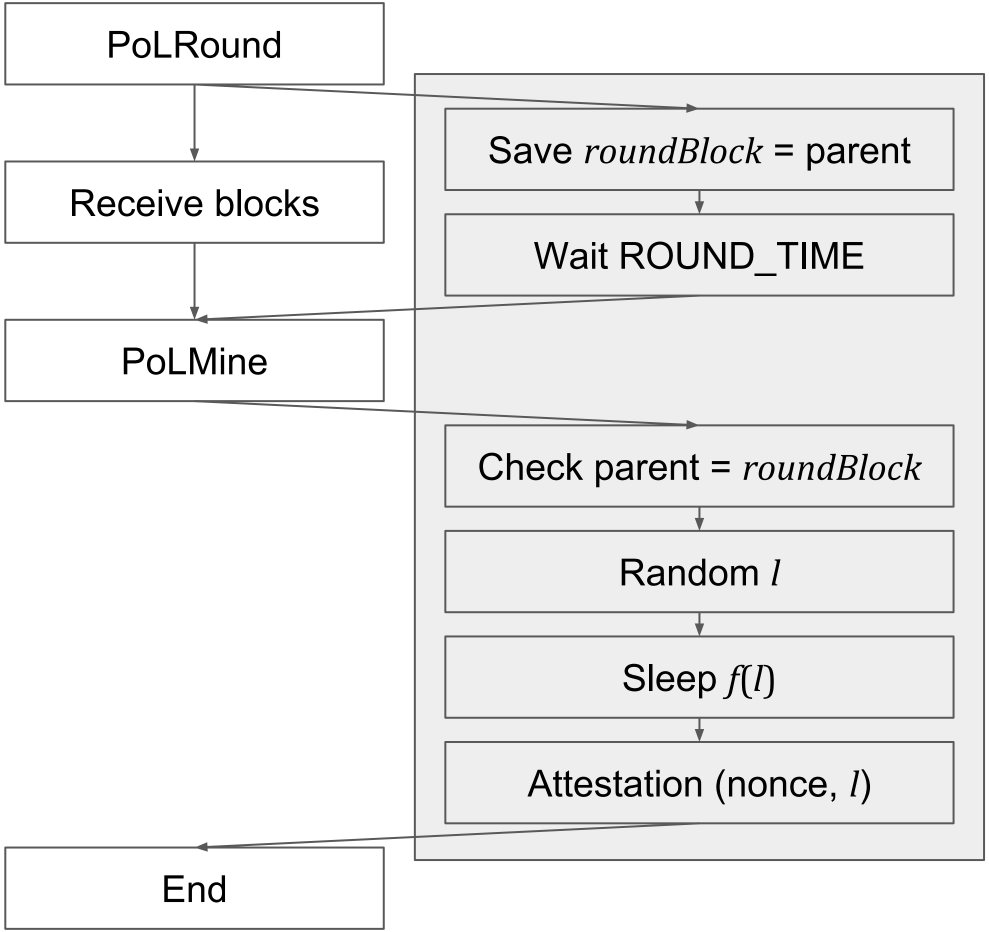
- Continue to receive competing blocks during ROUND_TIME
- After waiting, have a chance to switch
- Must have same parent as block chosen at beginning



Proof of Luck

Optimization:
slightly delay less-lucky blocks

Don’t broadcast if you’ve already received a luckier block



Analysis

Luck values: $l \sim \mathrm{Uniform}(0, 1)$

Scenario: attacker (m) splits itself from rest of network (M)

Threat model: attacker cannot compromise TEE, cannot split honest participants

h blocks after the fork, we have two chains with luck values:

$$1 \leq t \leq h \quad \begin{cases} l_M(t) \sim \text{max of } M\ \mathrm{Uniform}(0, 1) \\ l_m(t) \sim \text{max of } m\ \mathrm{Uniform}(0, 1) \end{cases}$$

All independent

Analysis

Scenario: attacker (m) splits itself from rest of network (M)

h blocks after the fork

$$L^{(h)} := \sum_{t=1}^{h} l_M(t) - l_m(t)$$

$$Pr\left(L^{(h)} \leq 0\right)$$?

Attacker's chain preferred

Analysis

$$L^{(h)} := \sum_{t=1}^{h} l_M(t) - l_m(t)$$

Chernoff bound

$$Pr\left(L^{(h)} \leq 0\right) \leq \min_{s>0} \mathbb{E}\left[e^{-sL^{(h)}}\right]$$

Expectation of product of independent variables

$$= \min_{s>0} \prod_{t=1}^{h} \mathbb{E}\left[e^{-sl_M(t)}\right] \mathbb{E}\left[e^{sl_m(t)}\right]$$

Identically distributed

$$= \min_{s>0} \left(\mathbb{E}\left[e^{-sl_M(t)}\right] \mathbb{E}\left[e^{sl_m(t)}\right]\right)^h$$

Analysis

Scenario: attacker (m) splits itself from rest of network (M)

Threat model: attacker cannot compromise TEE, cannot split honest participants

After the fork, **exponentially small probability that minority wins**

$$Pr\left(L^{(h)} \leq 0\right) \leq \min_{s>0} \Bigg(\underbrace{\mathbb{E}\left[e^{-sl_M(t)}\right] \mathbb{E}\left[e^{sl_m(t)}\right]}_{<1 \text{ for optimal } s \text{ if } M > m} \Bigg)^h$$

Compromised TEE

Scenario: attacker can compromise a few CPUs, not the whole platform

Approach: save top m luckiest numbers in each block,
only mth place (least lucky) one counts

Example (m = 5):

If attacker compromises fewer than m CPUs, they can't fully control block's luck

Needs further analysis

Outline

Background: blockchains, consensus, and SGX

Existing consensus mechanisms

Our paper:

3 *basic* consensus primitives

Proof of Luck

Conclusion

Conclusion

Properties of Proof of Luck:

- ASIC resistant
- Energy efficient
- Time efficient
- Permissionless and scalable

Summary of assumptions:

- Participants have access to suitable TEE hardware
- TEE programs can detect concurrent invocations
- TEE programs can generate unbiased random numbers

End of presentation.

Proof of time ⌛ - Implementation

Question: Which monotonic counter?

Monotonic counters accessed by random ID

Storage and communication must be done outside TEE

Proof of time ⌛ - Implementation

Question: Which monotonic counter?

Answer: All of them.

> **SGX_ERROR_MC_OVER_QUOTA**
>
> The enclave has reached the quota(256) of Monotonic Counters it can maintain

https://software.intel.com/sites/default/files/managed/d5/e7/Intel-SGX-SDK-Users-Guide-for-Windows-OS.pdf

Proof of time ⌛ - Implementation

Question: Which monotonic counter?

Answer: All of them.

- create 256 monotonic counters
- yield
- make sure all 256 still have correct value

Compromised TEE

Network may have slightly different blocks (e.g., due to latency)

Merge proofs of luck as long as blocks are “similar”

Similar blocks can be compressed

Analysis

Proportional control of blocks

Outline

Background: blockchains, consensus, and SGX

Existing consensus mechanisms

Our paper:

- 3 *basic* consensus primitives
- Proof of Luck

Conclusion