Decentralized Software Updates for Stake-Based Ledger Systems

A proposal and prototype implementation from the PRIViLEDGE Project

Nikos Karayannidis, IOHK

Damian Nadales, Michele Ciampi, Aggelos Kiayias, Dionysis Zindros

Software updates for blockchains are **important!**

Software update => Change => **Evolution**

- Software systems that don’t evolve will become out of date and redundant
- Blockchains evolve via software updates
- Change for
  - New features
  - Bug fixing
  - Change requests
  - Security reasons
  - Optimizations
  - **Protocol upgrade**

Updates for blockchains are **hard**

- Blockchains depend on consensus
- A protocol upgrade changes the way consensus is achieved
- … while consensus is working!
- What may go wrong?
  - Chain splits into two - parties upgrade with no synchronization
  - Chain is controlled by the adversary - security assumptions dont hold anymore
  - Protocol upgrade is blocked by the adversary
  - Ledger history is disrupted
  - A malicious update is approved

Decentralized updates for blockchains are **harder**!

It is all about **Governance!**

- **Who** decides?
- Decides on **What**?
- **How** is this achieved (off-chain / on-chain)?

And don't forget...

- Who pays? :-)

Disagreement hurts!

- Ethereum  Ethereum Classic
- Bitcoin  Bitcoin Cash

Problem Statement

Software updates on public blockchains today are neither **secure** nor **decentralized**

Our research main goals:

- **Decentralize the whole lifecycle** (end-to-end from ideation to activation) of a software update and not just a part
- Provide a **secure update mechanism** for a public ledger after defining what “secure” means in this context
- Build a prototype and **integrate it with the Cardano node**

Type of updates supported

- **Consensus-related / chain-split risk** (e.g., Tx validation rules, chain selection rules, consensus protocol per se etc.)
  - **Protocol parameters** - don't require installation (dynamic changes)
  - **Protocol upgrades** - require installation
- **Consensus-unrelated / no chain-split risk** (dont require synchronization upon activation)
  - Protocol parameters - don't require installation (dynamic changes)
  - Software updates that don’t affect the consensus protocol, e.g., optimizations, refactoring etc.
- **Cancellation**
  - Special type of update that cancels other approved proposals in case of an emergency

Our proposal

What does it take to decentralize software updates?

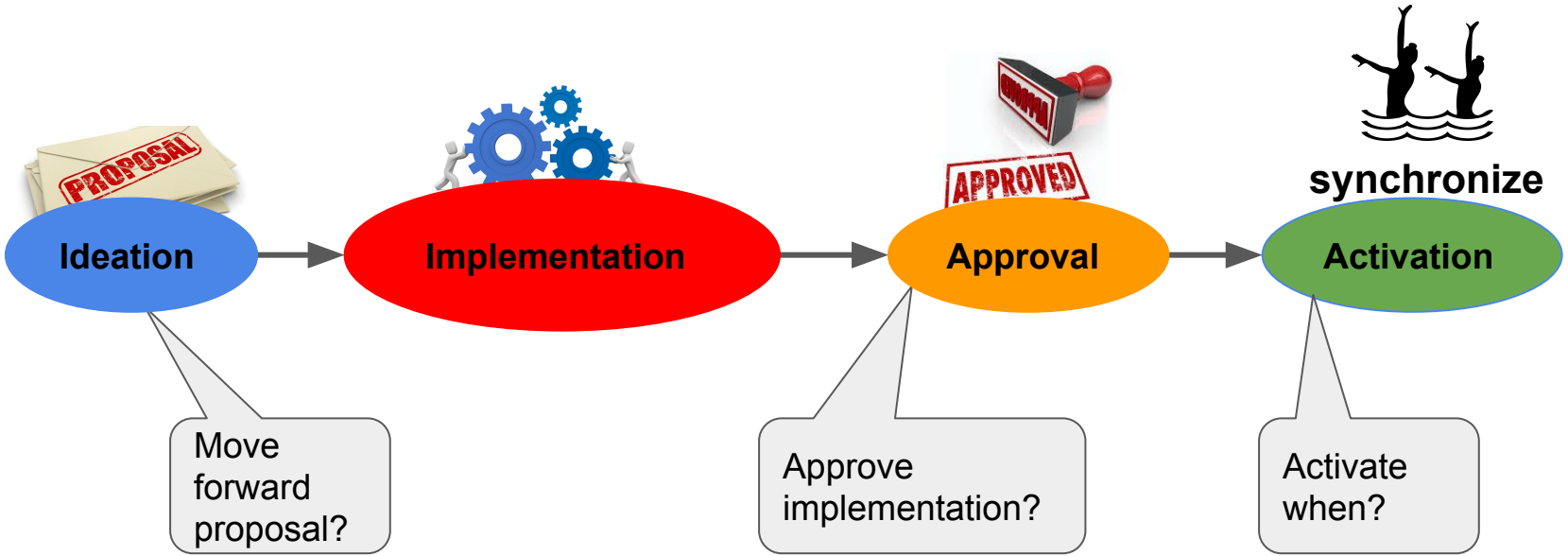
- **Open Participation Enabled** Anyone who can submit a common transaction should be able to submit an update proposal and vote for an update proposal.
- **Decentralized Decision-Making Enabled** All major decisions typically made by central authorities (e.g., code maintainer) should be made collectively by the community
- **On-chain and Protocol-Driven** The update mechanism should be an on-chain protocol driven process: Not based on informal discussions (e.g., social consensus by forums), but based on a protocol with specific security guarantees; a protocol that will leave behind an immutable (on-chain) trace of events.
- **Transparent and Auditable** Anybody should be able to answer WHEN, WHY and HOW the system has evolved the way it has. The evolution history of the system should be open to everyone and form a *globally-consistent tamper-evident public release log of the changes*.
- **Secure Activation Enabled** Define what is a *secure activation* and propose protocols that enable such an activation.
- **Performant and Scalable** The update mechanism should not impact the transaction throughput and size of the underlying blockchain. Moreover, the update mechanism should be scalable to thousands (or even millions) of participants.
- **Metadata-Driven** All updates are not the same (criticality, impact on the system, time to deploy etc.). We should enable a metadata-driven update policy (voting periods length, activation priority, deployment window etc.)
- **Consistent Update Logic Enabled** Efficiently handle priorities, version dependencies update conflicts and emergencies in order the system to always be at a consistent state

Core design decisions

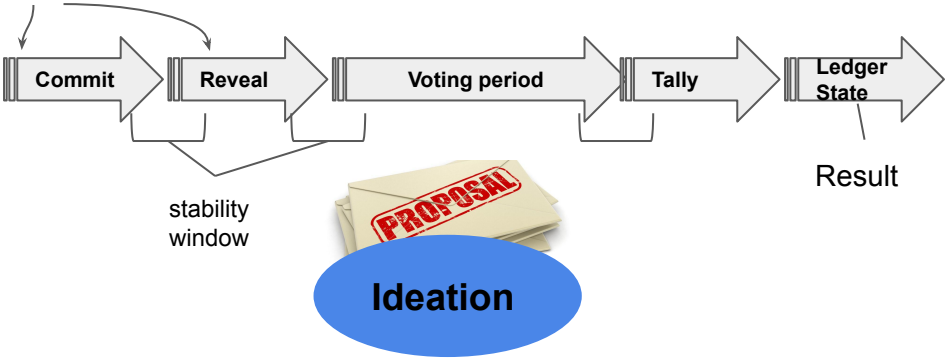
- Proposals and votes are just fee-based transactions with a special payload. Any stake owner can submit one (**open participation**)
- Replace all central authorities (e.g., code owner) with a voting process (**decentralized decision making**)
- The protocol runs **on-chain** and the update payload validation take place at the ledger layer as with common transactions
- No advanced crypto. **Simple, fast, secure and scalable** to millions of voters
- **Delegation of voting to Experts** to ensure technical expertise and participation.
- **No privacy (transparency)**. Only a commit-reveal scheme to ensure the rightful authorship.
- No one-size-fits all. **Metadata driven update context** (priorities, voting duration, deployment window etc). Voting applies to metadata also

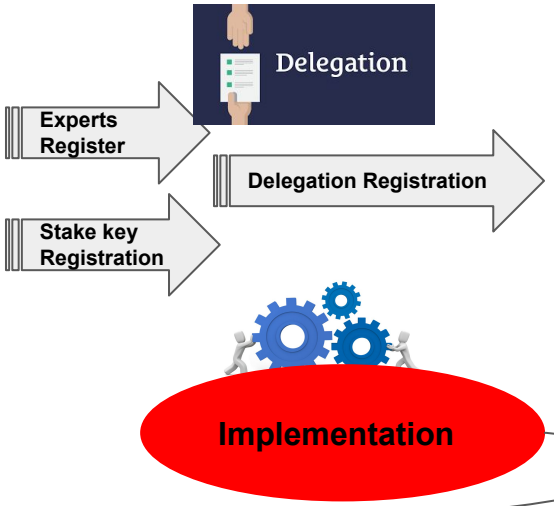
What are the critical decisions in the lifecycle of an update?

- Move forward or reject an update proposal
- Is the submitted code correct and safe to be activated?
- If and when should we activate?

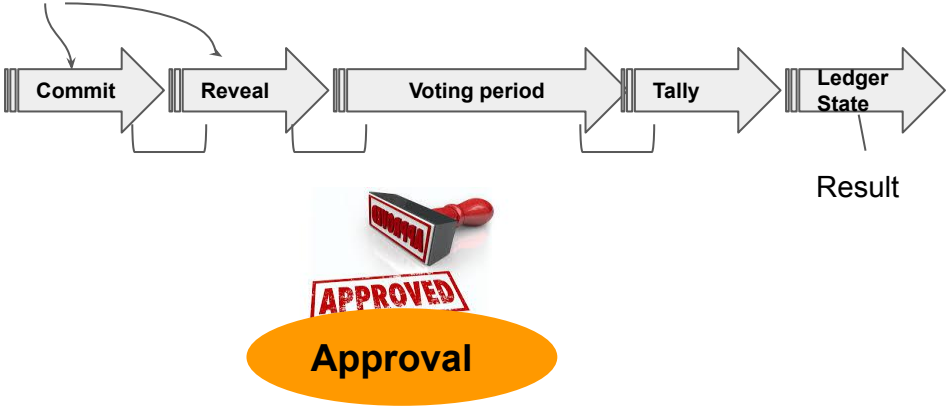


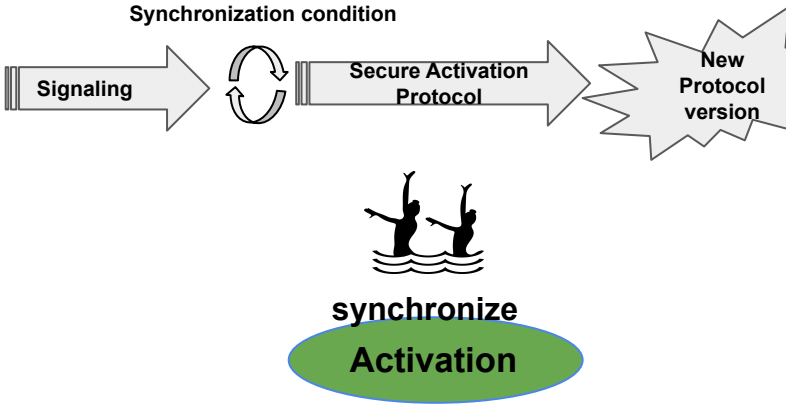
SIP (System Improvement Proposal)





UP (Update Proposal)





Update Governance: Voting

- A vote is a fee-based **transaction**
- Anyone who owns stake has a legitimate right to vote
- Voting power is proportional to owned stake
- Votes are accepted only within the **voting period** (metadata defined)
- Many proposals can compete in parallel
- A voter can change his/her mind and revote
- **3-value logic** for a vote (For, Against, Abstain)
- When an outcome has not been reached then a revoting period is possible until the proposal expires

Update Governance: Delegation

- Voters delegate their voting rights
- Why delegate?
- For 3 reasons: **Technical expertise / Specialization / Availability**
- Delegation for: **Specific / Category / Any** proposal
- A personal vote overrides a delegate's vote
- Default delegation to stakepools (to ensure participation)
- Delegation patterns
  - stake keys → stake pools
  - stake keys → stake pools → experts
  - stake keys → experts

Activation of changes: Security risks to consider ...

- Activate too soon ⟶
  - **Chain split**
  - Honest majority does not hold ⟶ **Adversary takes over** the new chain
- Activate too late ⟶
  - Adversary blocks the activation of an update (**Denial of Activation**)
- Ledger history is disrupted

A secure activation protocol guarantees that

- Activation begins only when “**enough stake**” has signaled **upgrade readiness** (so that the security assumption of the new ledger holds)
- The new ledger is secure
  - It enjoys the properties of **consistency** and **liveness**
- The state of the old ledger has moved into the new ledger
- In a nutshell, **L1 -> L2 is secure iff**
  - L2 enjoys the properties of **consistency** and **liveness**
  - L1 is a prefix of L2

Secure Activation Protocols

- Formal definition of secure activation
- Two secure activation protocols with various tradeoffs

See

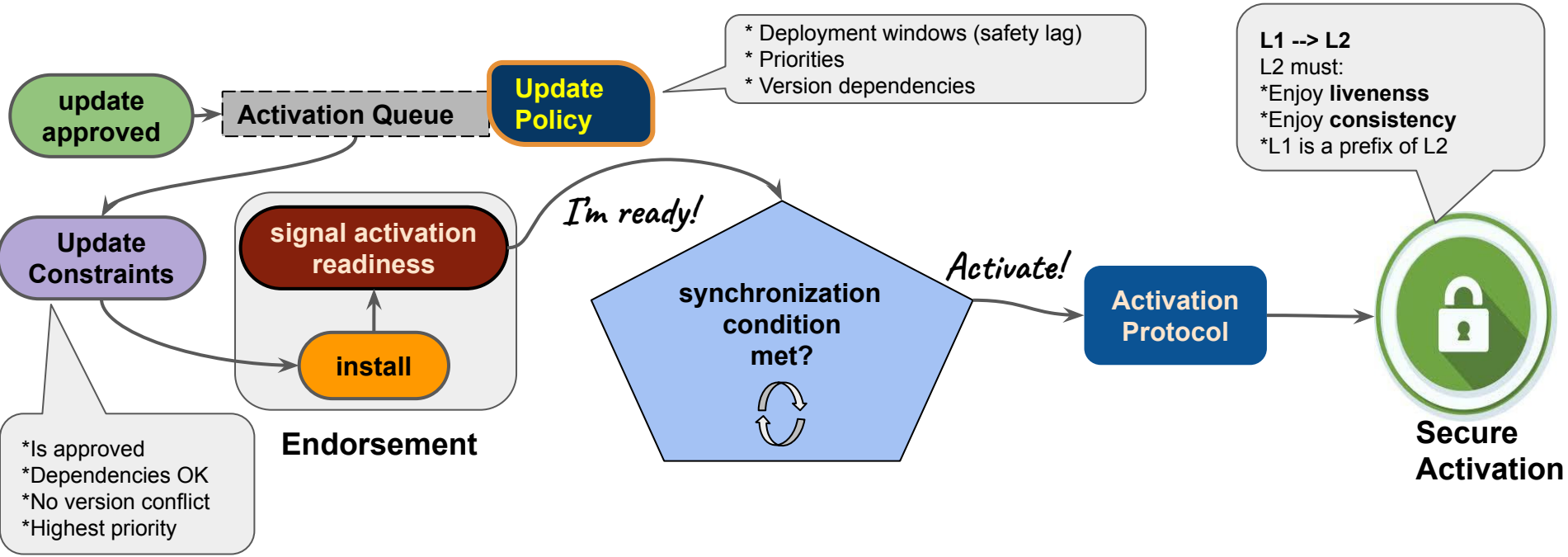
- Decrypto: Updatable Blockchains, Michele Ciampi
  https://www.youtube.com/watch?v=aYh7PsLlMNo
- Michele Ciampi, Nikos Karayannidis, Aggelos Kiayias, Dionysis Zindros: Updatable Blockchains. ESORICS (2) 2020: 590-609
  https://eprint.iacr.org/2020/887.pdf

In the real world an activation mechanism must also deal with

- Priorities
- Version dependencies
- Conflict resolution
- Emergency handling

The activation of changes

- Synchronize nodes via a **signaling mechanism**
- Use an appropriately chosen **adoption threshold**
- **Activation protocol** to ensure a **secure transfer** to the upgraded chain



Prototype implementation and integration with the Cardano node

Prototype Implementation

- Written in **Haskell**
- **Fully integrated** with the Cardano node (“**Mary” version** of the ledger)
- Extensively **property-based tested** (Quickcheck) via a rich set of validation criteria
- **Deployed** on a AWS **testnet** and have run successfully **end-to-end update scenarios**
- Cardano node now has a **pluggable update mechanism**
- Parametric on:
  - Who can vote (expert/stakeholder) (through stake distribution)
  - Proposal payload
  - Hashing algorithms
  - Crypto (pubkeys, signatures)

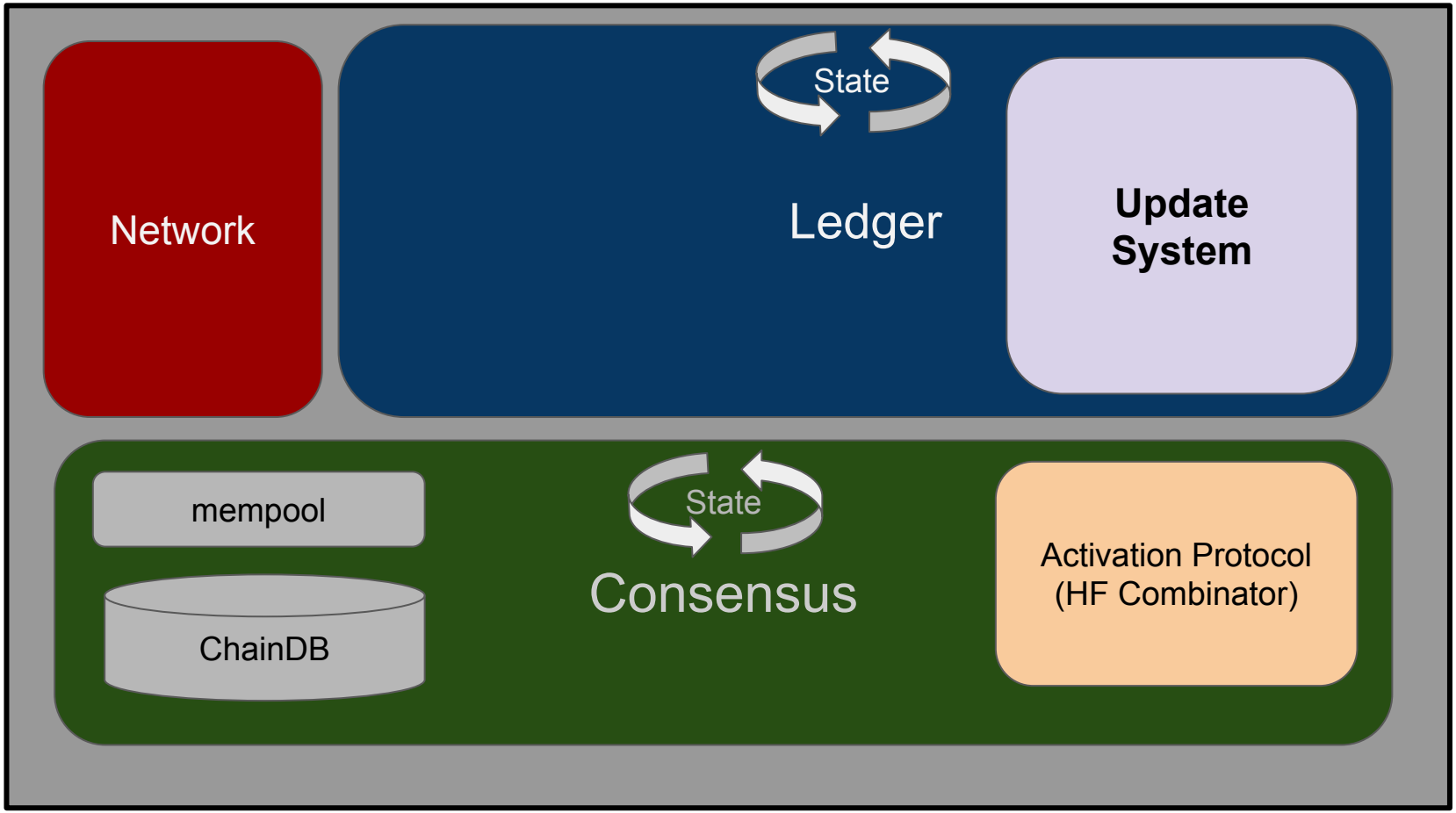
Prototype Approach

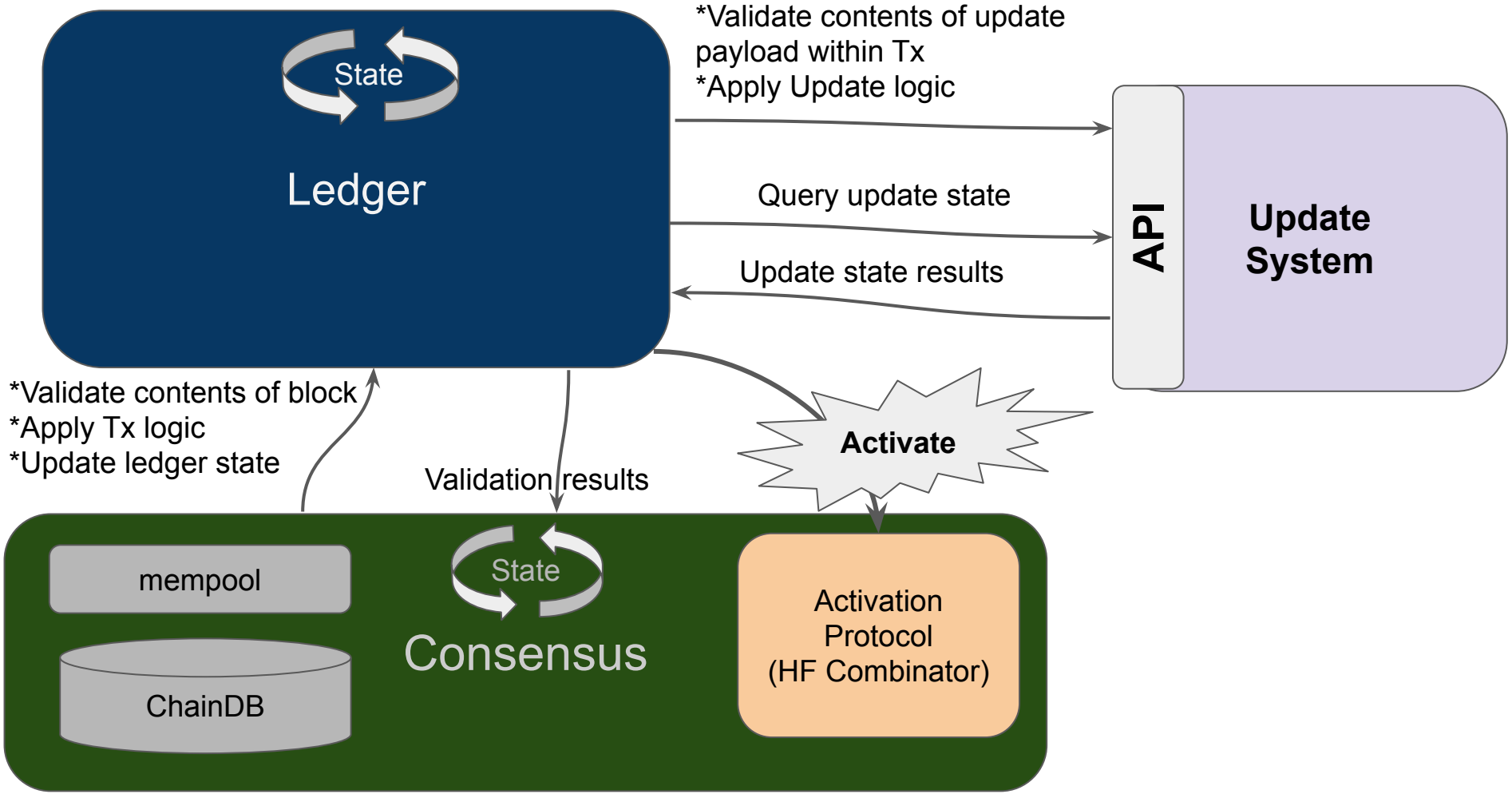
1st Step. Build a decentralized software updates system that materializes our design. Covers the full lifecycle of a software update from ideation to activation.

2nd Step. Validate the logic of the update system. Via a trace-enabled property-based testing framework

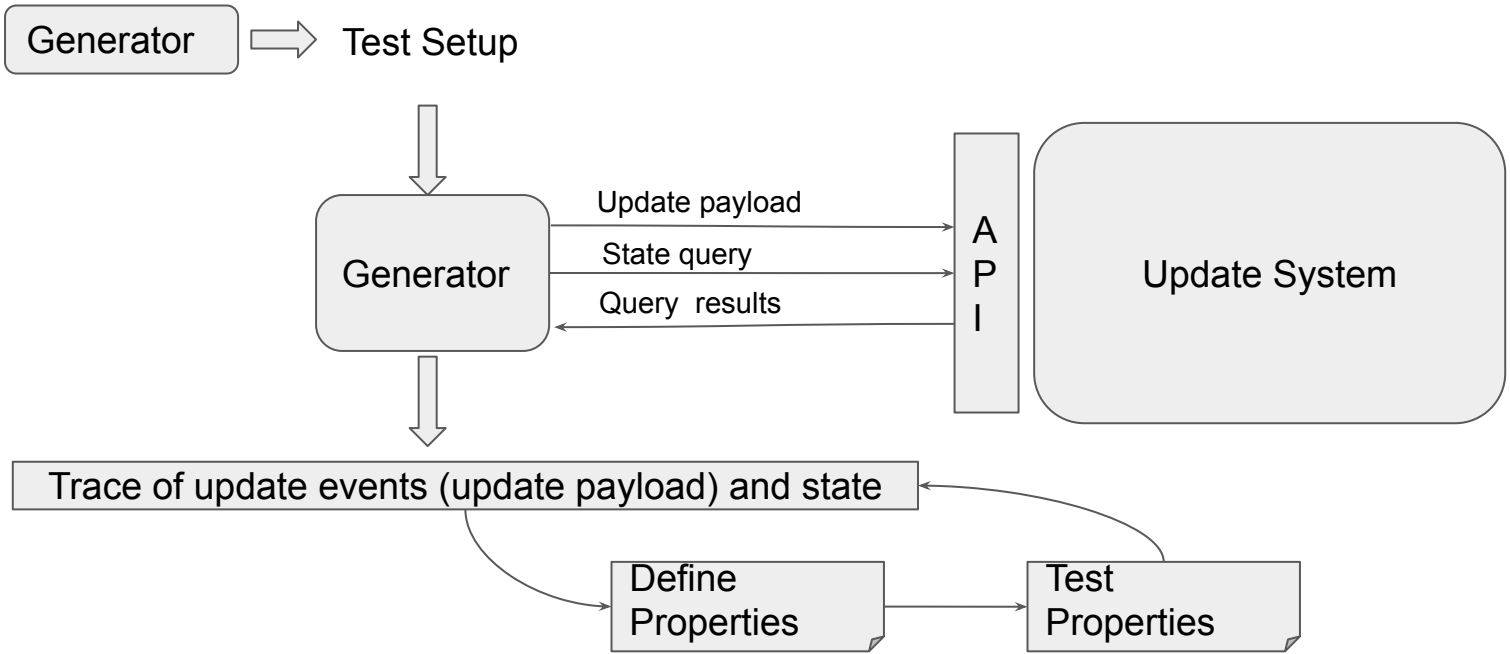
3rd Step. Integrate the update system to the Cardano blockchain.

Cardano Node

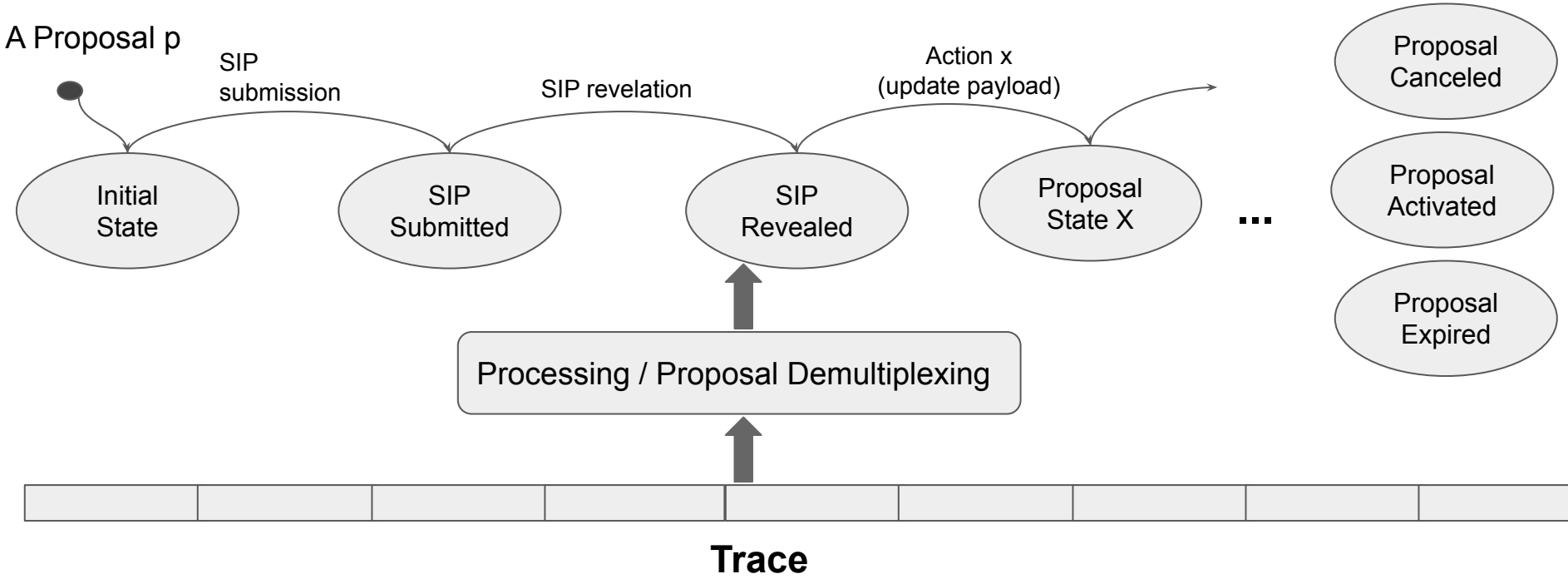



State
Ledger
*Validate contents of update payload within Tx
*Apply Update logic
Query update state
Update state results
API
Update System
*Validate contents of block
*Apply Tx logic
*Update ledger state
Validation results
Activate
mempool
State
Consensus
ChainDB
Activation Protocol (HF Combinator)

Testing Framework for Software Updates



Valid State Transitions in the Software Update Lifecycle



```
OK (747.13s)
      +++ OK, passed 409600 tests:
      96.2341% Implementation is revealed when SIP commit is not stable
      96.2332% SIP is revealed when SIP commit is not stable
      96.2302% Implementation is voted when SIP commit is not stable
      96.1914% SIP is voted when SIP commit is not stable
      93.4824% SIP is revealed when SIP is unknown
      93.4661% Implementation is voted when SIP is unknown
      93.4565% SIP is voted when SIP is unknown
      93.4346% Implementation is revealed when SIP is unknown
      80.1011% Implementation is voted when the SIP is revealed
      80.0869% SIP is voted when the SIP revealed
      80.0598% SIP is revealed when the SIP is already revealed
      80.0547% Implementation is revealed when the SIP is revealed
      79.7781% SIP is voted when the SIP is not yet revealed
      79.7063% Implementation is revealed when the SIP is not yet revealed
      79.6580% Implementation is voted when the SIP is not yet revealed
      68.4961% SIP is revealed when the SIP is already stably revealed
      68.4775% Implementation is voted when the SIP is stably revealed
      68.4722% Implementation is revealed when the SIP is stably revealed
      33.3035% Implementation is voted when a verdict on its SIP was reached
      33.2842% SIP is submitted when a verdict on it was reached
      33.2676% SIP is revealed when a verdict on it was reached
      33.2668% Implementation is revealed when a verdict on its SIP was reached
      33.2610% SIP is voted when a verdict on it was reached
      26.6541% SIP is revealed when a verdict on it was stably reached
      26.6331% SIP is voted when a verdict on it was stably reached
      26.6248% SIP is submitted when a verdict on it was stably reached
      26.6140% Implementation is revealed when a verdict on its SIP was stably reached
      26.6116% Implementation is voted when a verdict on its SIP was stably reached
      12.8357% Implementation is submitted when it is already in the stably submitted state
```

Performance tests

Running a decentralized software update mechanism on a blockchain should result into a substantial performance degradation

Back-of-the-envelope calculations (throughput utilization)

- Real-world conditions (voting periods, TBPS etc)
- Worst-case scenarios to determine upper bounds of performance impact

Testnet runs

- 10 node set up, on AWS, geographically distributed (4 countries and 3 continents)
- Confirm calculations
- Observe impact on throughput and latency

Micro-benchmarking (tally phase) and Asymptotic Analysis

- Processing time
- Memory consumption

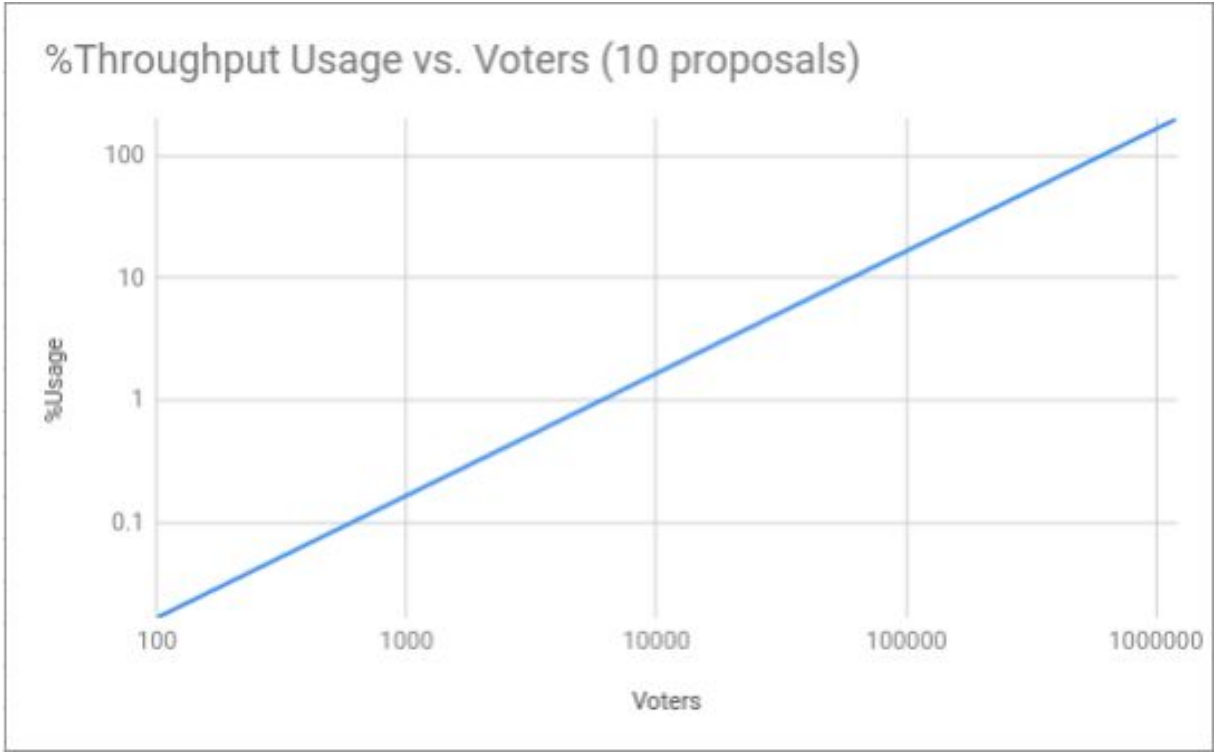
Throughput utilization (calculations)

$$usage_{pct} = 100\frac{psize}{TBPS\ duration_u}$$

$$psize_v = s_v n_p n_r n_c$$

* size(vote) = hash(proposal)+pub-key+signature+confidence = 164 bytes
* **Cardano TBPS = 3.2Kb/s**
* Max block body size = 64Kb
* 1 block every 20 sec

n_p	n_r	n_c	$duration_v$	$usage_{pct}$
1000	2	1	7	0.017
1000	2	5	7	0.083
1000	2	10	7	0.166
10000	2	1	7	0.166
10000	2	5	7	0.828
10000	2	10	7	1.655
100000	2	1	7	1.655
100000	2	5	7	8.277
100000	2	10	7	16.555
1000000	2	1	7	16.555
1000000	2	5	7	82.773
1000000	2	10	7	165.546
1000000	2	10	14	82.773
1000000	2	10	30	38.627
10000000	2	1	7	165.546
10000000	2	5	7	827.729
10000000	2	10	7	1655.458



TPS impact (Testnet)

- **10 node set up**
- AWS (4 countries and 3 continents)
- 30 min voting duration
- 50 threads per node submitting UTxO transactions
- 1 block every 20 sec

Voters per node	TPS	Usage %
1	16.64764543	0.036
10	16.5933518	0.362
100	16.04376731	3.662
250	15.14626039	9.051
500	13.64155125	18.087
750	12.13795014	27.115
1000	10.63822715	36.121
1250	9.127977839	45.189

Table 5.3: Experimental results on TPS

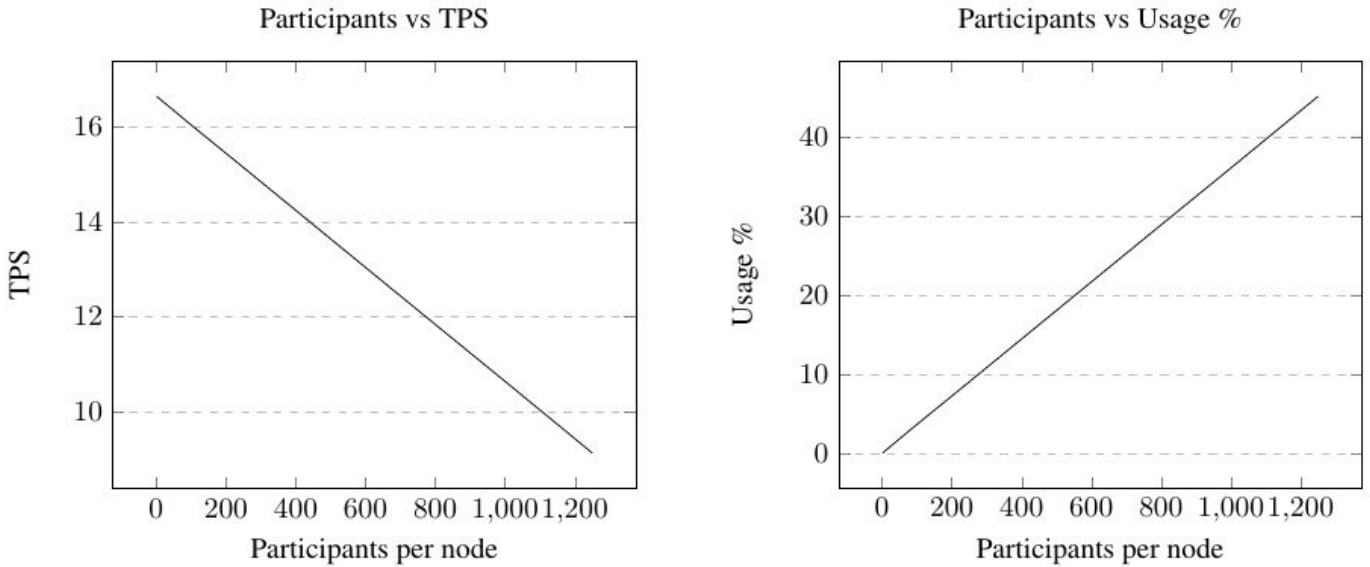
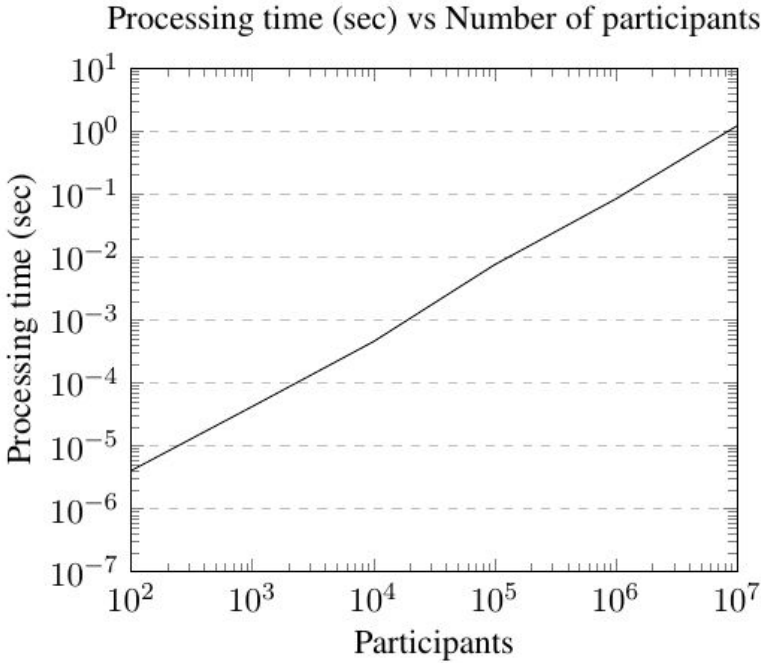
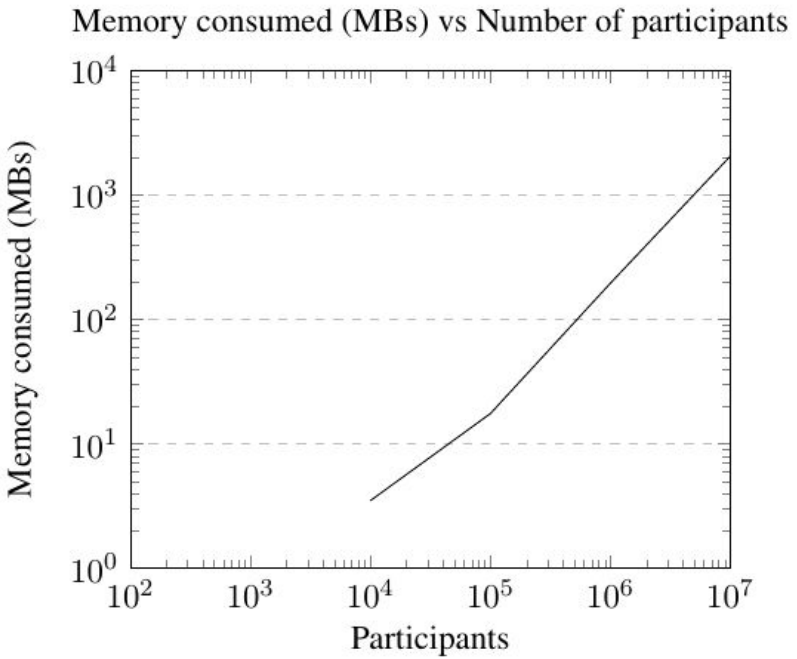


Figure 5.6: Experimental results on TPS

Processing time and Memory consumption (micro-benchmarking)





* Tally algorithm is **O(n)** where n is number of participants
* Tests on a i7, 32GB laptop
* Memory is consumed for storing the 256bit hashes of the pubkeys

Performance main conclusions

- The update mechanism does not impact the blockchain performance substantially
- It scales efficiently to the number of participants

Some performance boosters

- Expert pools vote instead of stakeholders
- Longer voting periods
- Greater max block body size (e.g., 1MB)

Threshold Analysis

2 processes and 2 attacks

- **Voting process**
  - **Malicious Proposal Attack:** The adversary manages with its own stake to approve a malicious proposal.
  - **Denial of Approval Attack:** The adversary manages to block a good proposal approved by the honest stake.
- **Activation process**
  - **Rushing Adversary Attack:** The adversary rushes, by using its own stake, to signal upgrade readiness and cause the activation of the changes, in order to take over the control of the updated blockchain, before enough honest parties manage to upgrade.
  - **Denial of Activation Attack:** The adversary manages to block the activation of a change signaled by the honest stake.

Safety & Liveness

- **Safety property for voting:** a software update that has achieved only the adversary stake approval, will never be approved.
- **Liveness property for voting:** a software update that has achieved sufficient honest stake approval, will be eventually approved, i.e., cannot be blocked by the adversary stake.
- **Safety property for activation:** a software update that has not received signals of sufficient stake w.r.t. the security assumption of the upgraded protocol, will never be activated.
- **Liveness property for activation:** a software update that has received signals of sufficient honest stake, will be eventually activated, i.e., cannot be blocked by the adversary stake.

Condition for safety and liveness

For any threshold τ, **both Safety and Liveness hold** iff

$$T < \tau_V < H \quad \text{(Voting)}$$

$$1/r_{Th} T < \tau_A < H \quad \text{(Activation)}$$

- τ_V: Voting threshold
- τ_A: Adoption threshold
- H: % of honest stake
- T: % of adversary stake
- r_{Th} : the theoretical adversary tolerance of the consensus protocol e.g., $r_{Th} = ½$
- If **%signals > τ_A**, then **Adversary Stake / Total Stake < r_{Th}** holds on the upgraded protocol => t**he security assumption holds**

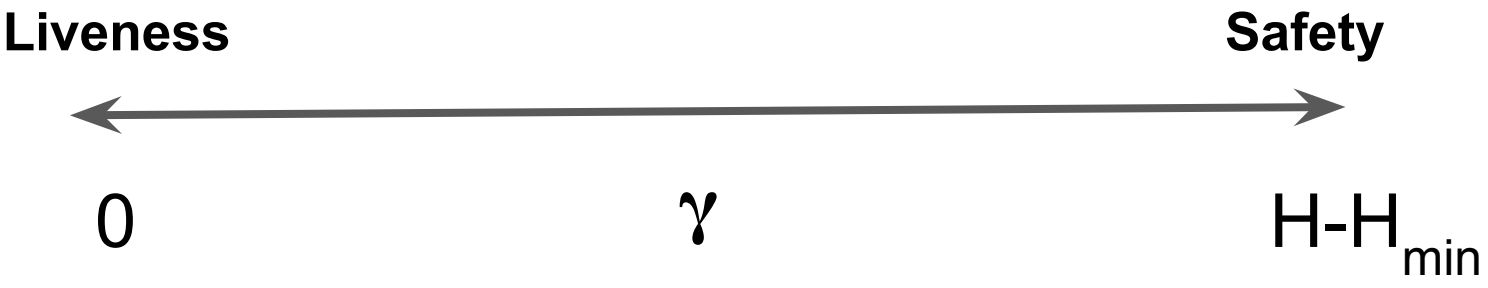
The threshold function $\tau(\gamma)$

$$\tau(\gamma) = H_{min} + \gamma$$

$$\wedge \quad T < H_{min} \text{ (voting)}$$

$$\wedge \quad 1/r_{Th} T < H_{min} \text{ (activation)}$$

$$0 \leq \gamma < H\text{-}H_{min}$$



H_{min}: determines the minimum percent of **honest stake** i.e., $0 \leq H_{min} \leq H$ that we wish the liveness property to hold and is the lower bound of our threshold function $\tau(\gamma) \geq H_{min}$

Examples of the threshold $\tau(\gamma)$

Phase	**H_{min}**	**Threshold function** $\tau(\gamma) = H_{min} + \gamma$ $0 \le \gamma < H\text{-}H_{min}$	**Constraint** **$T < H_{min}$** (voting) **$1/r_{Th}T < H_{min}$** (activation)
Voting	H/2	50(1-r) + γ	r < 1/3
	T + 1	100r + 1 + γ	Any r < r_{Th}
Activation	H/2	50(1-r) + γ	r < $r_{Th}/(2 + r_{Th})$ (e.g., r < 1/5 for r_{Th} = 1/2)
	$1/r_{Th}$ x T+1	$1/r_{Th}$ x 100r + 1 + γ	Any r < r_{Th}

Where

H: % of honest stake

T: % of adversary stake

H + T = 100

r = T/100: the adversary ratio

r_{Th} : the theoretical adversary tolerance of the consensus protocol e.g., r_{Th} = 1/2

Example for Voting process

$$\tau(\gamma) = H/2 + \gamma = 50(1-r) + \gamma \;,\; r < 1/3$$
$$0 \leq \gamma < H/2$$

Adversary ratio r (r < 1/3 ≈ 0.33)	**$\gamma = 0$**	**$\gamma = 100 \times r = T$**	**$\gamma = H/2 - 1$**
0.1	$T_V = 45\%$	$T_V = 55\%$	$T_V = 89\%$
0.2	$T_V = 40\%$	$T_V = 60\%$	$T_V = 79\%$
0.3	$T_V = 35\%$	$T_V = 65\%$	$T_V = 69\%$

H/2 is enough for liveness

Unanimity of honest stake required for liveness

[35% - 69%] threshold range for both safety & liveness for r = 0.3

Key takeaways

- Software updates for blockchains are important
- Software updates for blockchains are difficult
- Decentralized software updates for blockchains are even more difficult
- A decentralized software update mechanism should enable
  - Open participation
  - Decentralized decision making in all phases (ideation, approval, activation)
  - Protocol driven on-chain solution
  - Transparent and auditable
  - Secure activation, resilient to chain splits
  - Metadata-Driven and Consistent Update Logic
  - Scalable and efficient
- We have proposed such a mechanism
- We have implemented it and integrated it into Cardano as a proof of concept

More info

- PRIViLEDGE Decentralized Software Updates Design Spec https://github.com/input-output-hk/decentralized-software-updates/tree/master/design-spec
- Michele Ciampi, Nikos Karayannidis, Aggelos Kiayias, Dionysis Zindros: Updatable Blockchains. ESORICS (2) 2020: 590-609 https://eprint.iacr.org/2020/887.pdf
- Our repo https://github.com/input-output-hk/decentralized-software-updates
- The PRIViLEDGE project https://priviledge-project.eu/