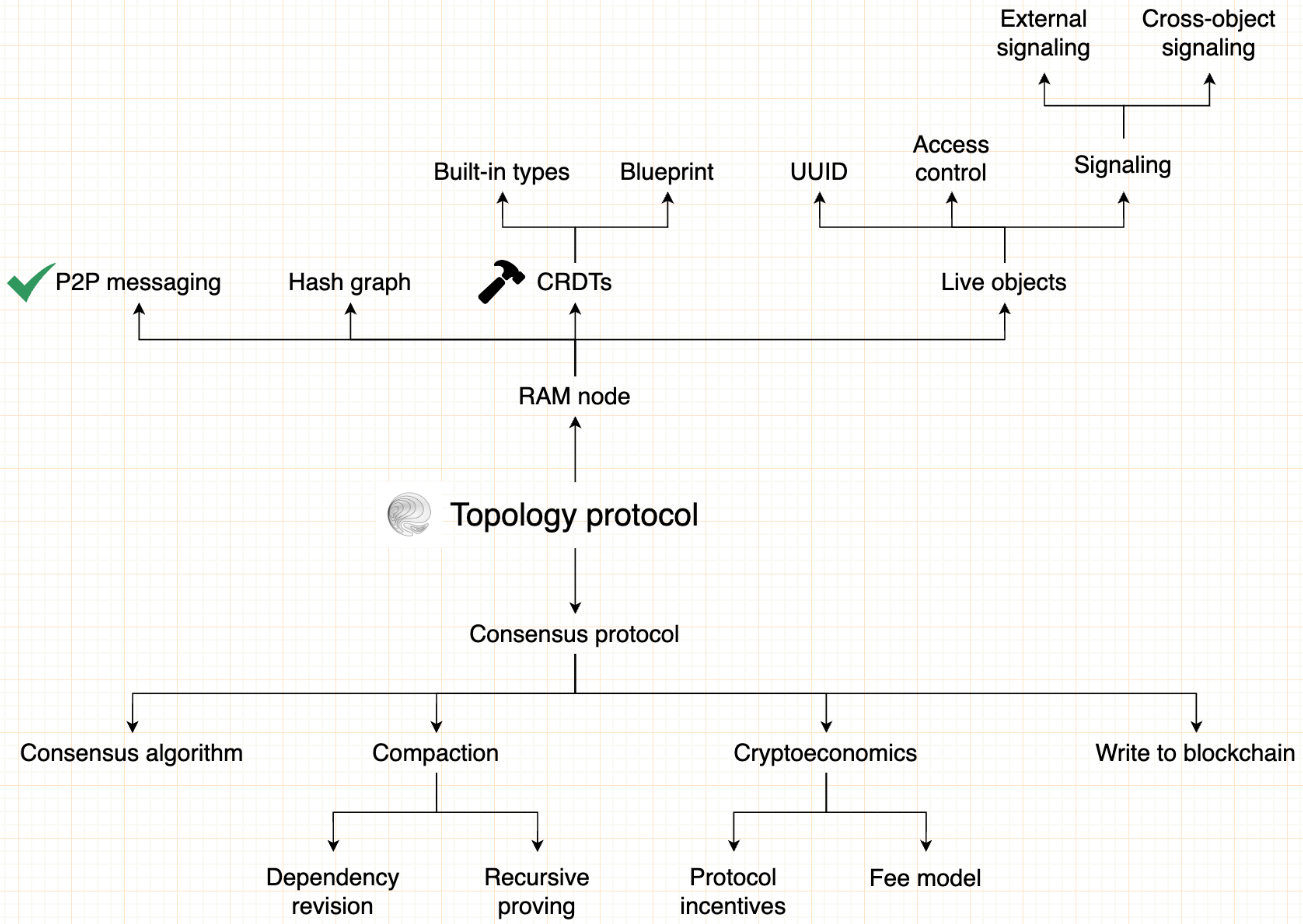
Topology protocol research

Thomas (guiltygyoza)
Mar. 28, 2024

Concept map


External signaling
Cross-object signaling
Built-in types
Blueprint
UUID
Access control
Signaling
✓ P2P messaging
Hash graph
CRDTs
Live objects
RAM node
Topology protocol
Consensus protocol
Consensus algorithm
Compaction
Cryptoeconomics
Write to blockchain
Dependency revision
Recursive proving
Protocol incentives
Fee model

Research topics

1. Compaction & consensus algorithm
2. Fee model
3. Writing to blockchain

1. Compaction & consensus algorithm

Recap: unbounded memory

- Recap: transacting with a live object is default free (no need to pay for consensus for consistency). However, grow-only hash graph requires unbounded memory, which is impractical.
- Estimation of memory consumption:
  - Each message has the structure: *(event, dependencies, signature)*
  - Assume event = 4-byte function signature + N 32-byte arguments
  - Assume M users accessing the object at frequency F (Hz)
  - Assume 32 bytes per causal dependency hash (eg Keccak256)
  - Assume 64 bytes per signature (eg. Ed25519)

Recap: unbounded memory (cont.)

- After initial network delay (ie everyone starts hearing from everyone else):
  - The size of each message is the size of event (4 + 32*N) plus the size of dependencies (32 * M) plus the size of signature (64) = 68 + 32*N + 32*M (bytes)
  - a hash graph replica grows by M new messages per 1/F second (1 message generated locally, M-1 received from others)
  - Graph growth rate: M * (68 + 32*N + 32*M) * F bytes/second
- Assume: M = 10 (CSGO), N = 4 (blind guess), F = 64 (default tick rate of CSGO) → 330 KB/sec, or 19.8 MB/min

Recap: time-out

- Recap: besides the problem of unbounded memory, there is the problem of late arrival of concurrent messages (effectively requiring application to rollback and recompute current state), plus the possibility of old-dependency attack increases.

- It helps to introduce a notion of time for the purpose of timing-out old messages

  - Timed-out messages should not get dropped; they are simply “disqualified” from their concurrent message group

  - How to introduce time?

Intuition: need for consensus

- Every replica of a live object holds a differing hash graph. Snapshot precedes compaction.

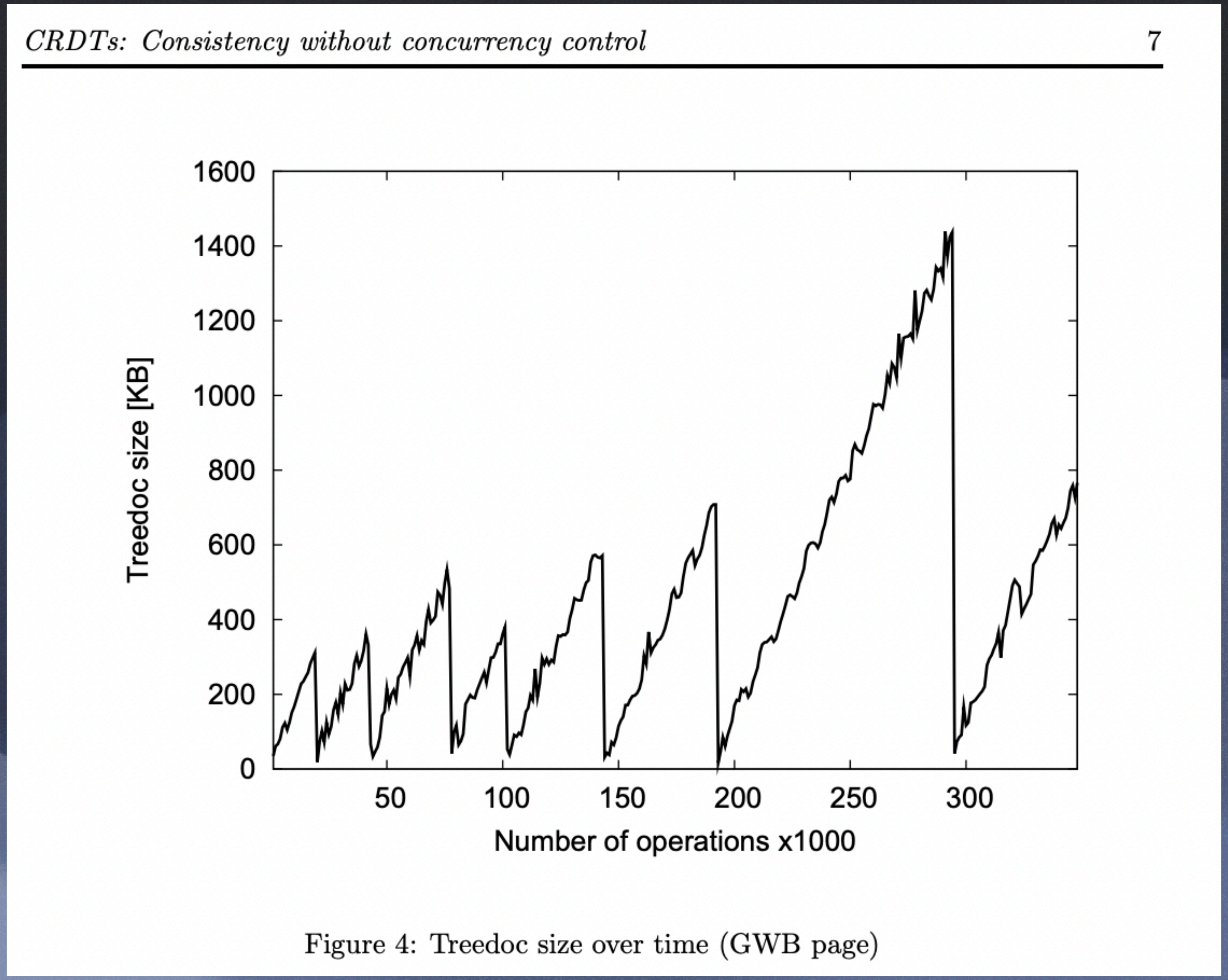
- For a permissioned live object, it makes sense to designate a specific replica as leader, and ask the leader to take a global snapshot (eg chandy-lamport), perform compaction, and sends out the compacted graph to other replicas

- For an open-access live object, “taking a snapshot” mostly likely requires consensus — “a snapshot is the merging of correct hash graphs from a majority of nodes”

- The introduction of time also requires consensus — “lock-step synchrony, requiring a majority of nodes to agree to advance time”

Intuition: succinctness

- With periodic compaction, the memory footprint of a replica fluctuates like this: (Treedoc paper, Fig. 4)


CRDTs: Consistency without concurrency control 7

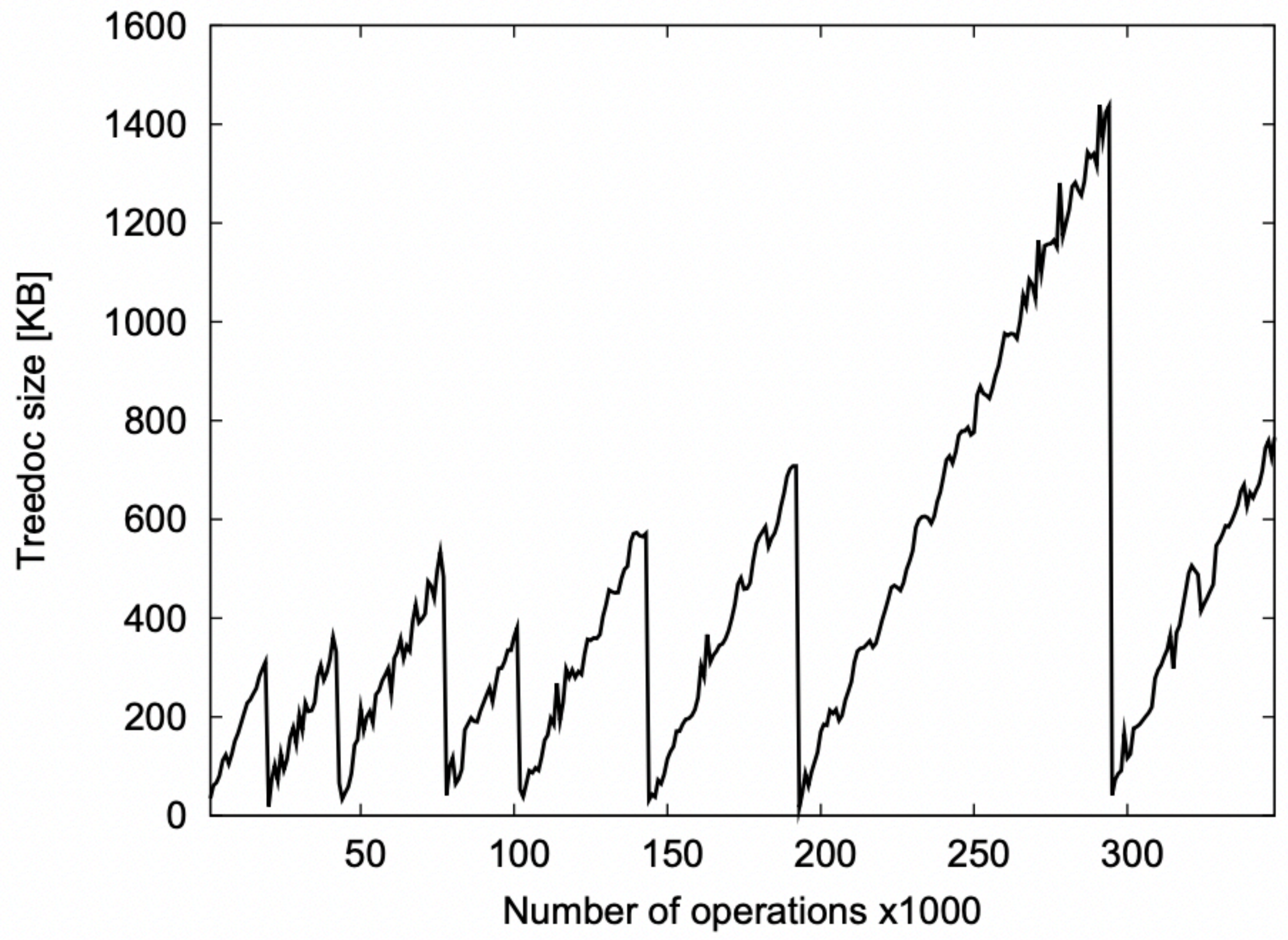


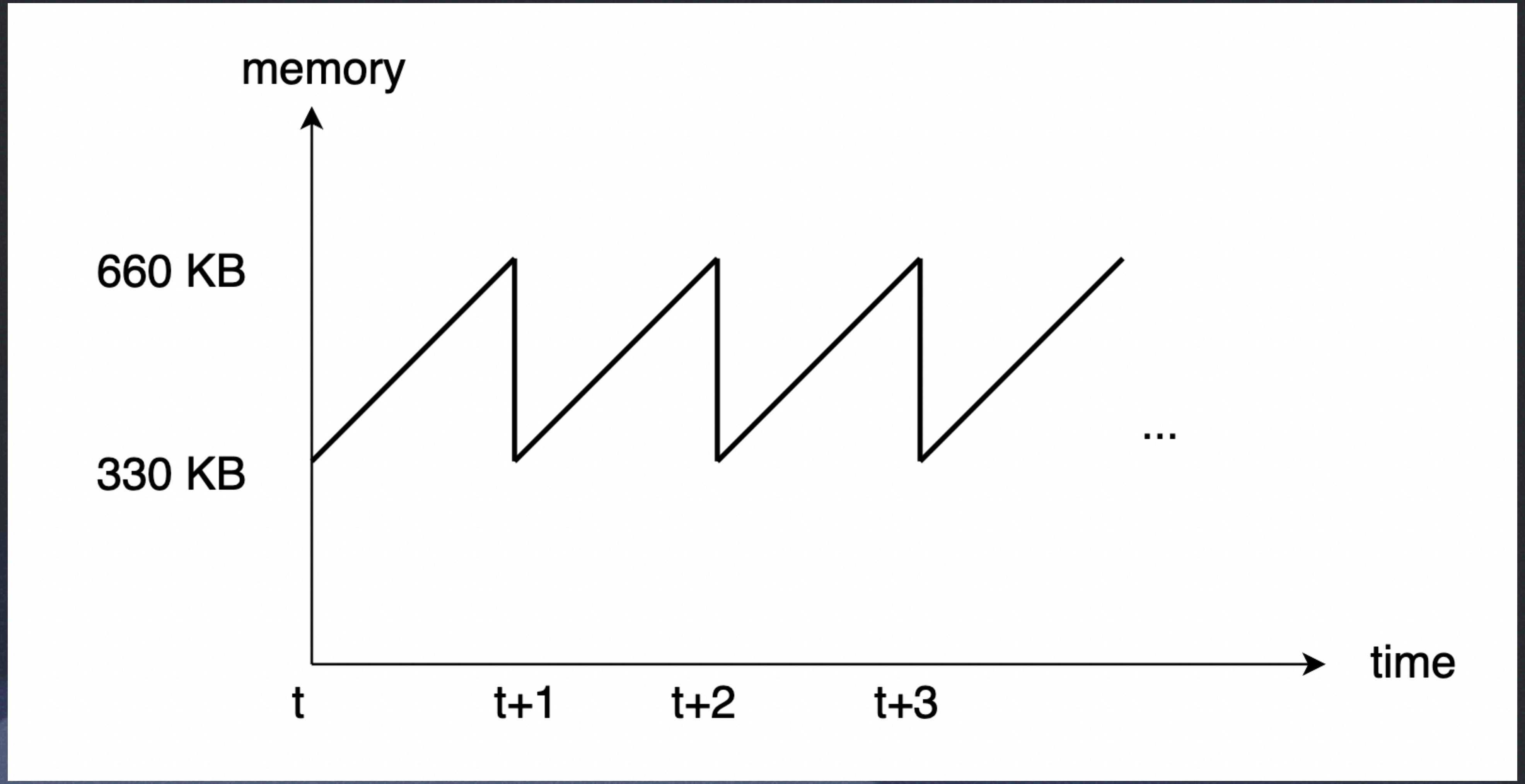
Figure 4: Treedoc size over time (GWB page)

Intuition: quantifying succinctness

- The maximum memory footprint for a live object replica is estimated as **2*ΔG**, where ΔG = compaction interval * ΔG/Δt
- Explaining the multiplier 2
  - While replicas are reaching agreement on what portion of the graph to be compacted, each replica will have grown the graph further.
  - The steady state looks like: from time t → t+1, each replica grows by roughly ΔG, while replicas come to agreement to compact the ΔG from t-1. Right before compaction occurs, a replica size ~2*ΔG.

Intuition: succinctness estimation

- Following our CSGO estimation, ΔG/Δt = 330 KB/sec
- If the protocol manages to compact every second, max memory footprint for this object is 2 * 1 sec * 330 KB/sec = 660KB.
- Compaction protocol speed impacts memory footprint, but not user transaction rate (user transacts at local speed), which is fundamentally different from blockchain

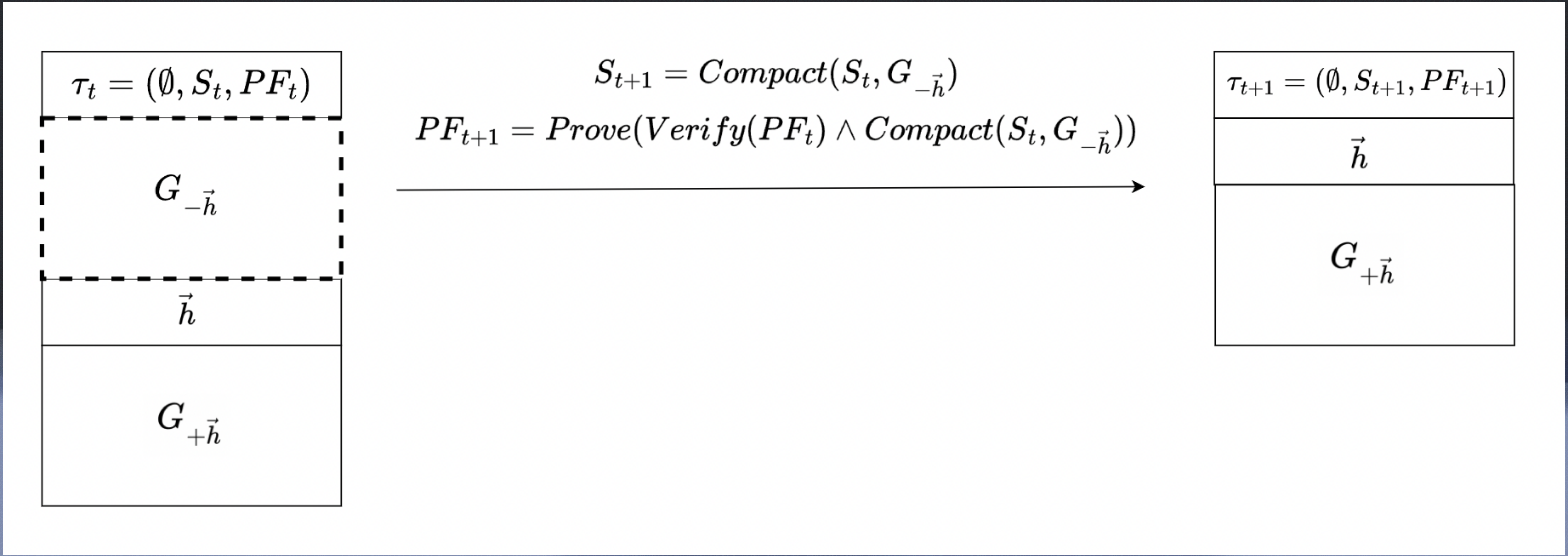


Intuition: trust-minimization

- However, we also want trust-minimization: assume a protocol node performs the compaction — how would other nodes trust or verify that the compaction was carried out correctly and the new state is correct?

Intuition: proof recursion

- Model: (Topology paper draft 7.3, Fig. 8). Tau is tail, \vec{h} is heads.


$\tau_t = (\emptyset, S_t, PF_t)$
$G_{-\vec{h}}$
$\vec{h}$
$G_{+\vec{h}}$
$S_{t+1} = Compact(S_t, G_{-\vec{h}})$
$PF_{t+1} = Prove(Verify(PF_t) \wedge Compact(S_t, G_{-\vec{h}}))$
$\tau_{t+1} = (\emptyset, S_{t+1}, PF_{t+1})$
$\vec{h}$
$G_{+\vec{h}}$

Topics to investigate

1. **Ad-hoc compaction:** In what scenarios would ad-hoc compaction procedure be suitable? An ad-hoc procedure might be: designate a leader replica; leader performs chandy-lamport snapshot regularly according to leader's machine clock. Describe the setup and step by step procedure in detail.

2. **Compaction inconsistency:** Topology paper section 9 intuits an algorithm for an asynchronous consensus protocol based on threshold logical clock [Ford]. The algorithm does not consider what happens when two nodes advance their logical time by compacting different graphs, making their tail node states different. How to fix?

3. **Who compacts:** assume a dynamic set of protocol nodes pace the compaction. Which node performs the compaction and produces the recursive proof?

4. **Consensus:** Devise a suitable consensus protocol. GG is investigating leader-less asynchronous consensus protocols: [QSC] & [Swirlds]. Also, prototype the protocol to study its characteristics around communication complexity, how many rounds/steps between a consensus agreement etc.

5. **Dependency revision:** When compacting graphs, we are throwing away causal information, which means we need to revise the dependencies of messages that causally depend on the compacted graph. Are protocol nodes responsible for this operation? Do protocol nodes perform signature over these reformed messages?

References

1. [Chandy & Lamport] https://lamport.azurewebsites.net/pubs/chandy.pdf
2. [Treedoc] https://arxiv.org/pdf/0907.0929.pdf
3. [Ford] https://arxiv.org/abs/1907.07010
4. [QSC] https://arxiv.org/abs/2003.02291 & https://github.com/dedis/tlc
5. [Swirlds] https://www.swirlds.com/downloads/SWIRLDS-TR-2016-01.pdf
6. Discussion threads under IPFS: https://github.com/ipfs/notes/issues/407 & https://github.com/ipfs-inactive/dynamic-data-and-capabilities/issues/2

2. Fee model

Intuition: rate-based fee

- Transacting with a live object at 60Hz makes it infeasible for user to review the fees of each transaction and manually sign each.

- At the same time, not all transactions are equal — they differ at least by their time complexity (e.g. WASM instruction count).

- For metaverse-like applications (continuous transactions), I believe user ultimately cares about "*how much do I need to pay per X seconds of login time?*"

- It would be nice to quote users a relatively fixed $TOP/s (assuming $TOP is the ticker of the native currency of this protocol)

Intuition: spam

- Consider a live object that expects each user to transact at 60Hz.
- By our current construction, there’s nothing to stop a user from transacting at 1000Hz, because we say transacting with live objects incur zero fee by default.
- Protocol nodes need to execute the transactions at compaction, and produce recursive proof. Complex transactions translate to more load on the protocol nodes, which should be compensated for.
- It would be nice to rate-limit each user at some frequency specified by a live object, where rate is not transaction/s but instruction/s.

Intuition: putting them together

- It would be nice to quote a fixed $TOP/s.
- It would be nice to rate-limit by instruction/s.
- If we can manage to have a stable $TOP/instruction denoted as **P**, consider a live object with rate limit **R** instruction/s, then the live object's interaction fee is:

P $TOP/instruction * R instruction/s = **P*R** $TOP/s.

Topics to investigate

- How to verifiably rate-limit by instruction/s? Consider Ethereum's RLN. Also consider this from the perspective of protocol node needing to know the instruction count of each transaction.

- What happens when a user hits the rate limit, as detected by a protocol node?

- How to be able to charge a stable $TOP/instruction? Assuming WASM instruction.

- Consider these topics together with the responsibility of protocol node to perform compaction and recursive proving.

3. Write to blockchain

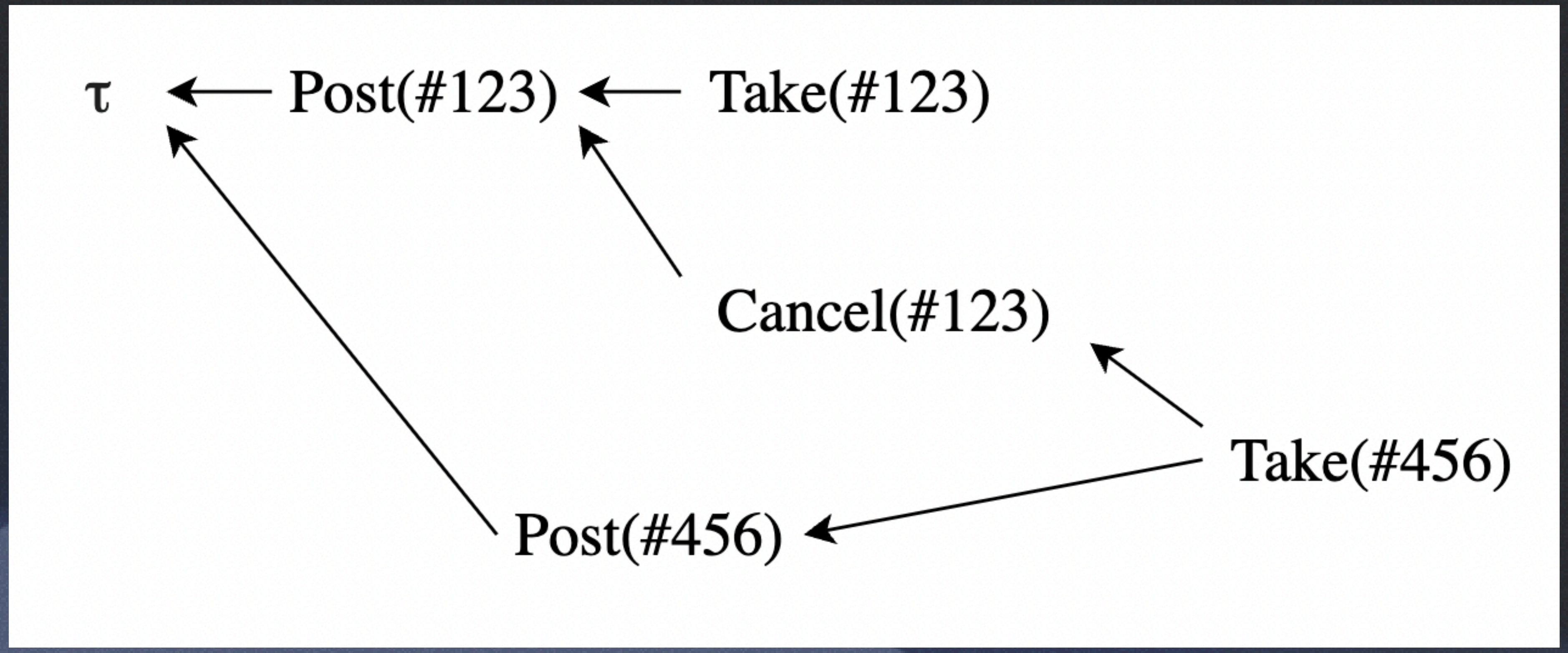
Intuition: snapshot

- A snapshot of a hash graph can be converted into a set of transactions to be submitted onchain.

- Example: orderbook DEX:

  - Consider the snapshot on the right

  - This snapshot should translate to a transaction that settles some payment between the poster and taker of order #456.



Topics to investigate

- Who performs the translation of snapshot to transactions? Who posts transactions onchain and pays for them?
- How does the smart contract onchain verify the correctness of transactions coming from a snapshot?
- How to prevent double-posting?