Coloring Smart Contracts and Other Musings About Efficient Blockchain Execution

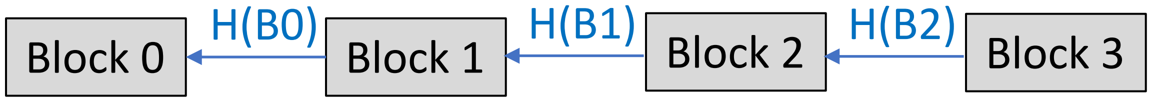
Roy Friedman

Technion

A (Distributed) Systems View of Blockchains

Blockchains implement a distributed replicated ledger abstraction

- Ledger $\triangleq$ a log of transactions
- The ledger is divided into blocks of transactions
- Each block includes a cryptographic hash of its predecessor, thereby creating a tamper-proof chain



Loosely speaking, blockchains consist of the following aspects

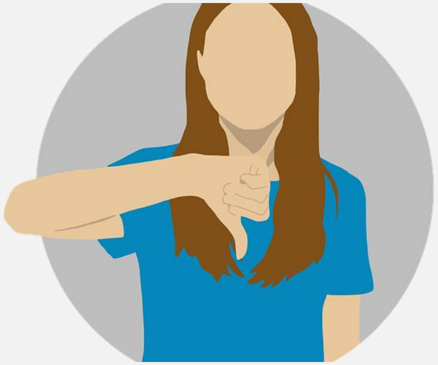
- Crypto
- Agreement/consensus on the blockchain content despite Byzantine (malicious) failures
- P2P dissemination of transactions and blocks
- Transactions and smart contract validation and execution

In This Talk...

Since I am not a crypto expert, I will focus on the other aspects

From PoW to BFT Consensus

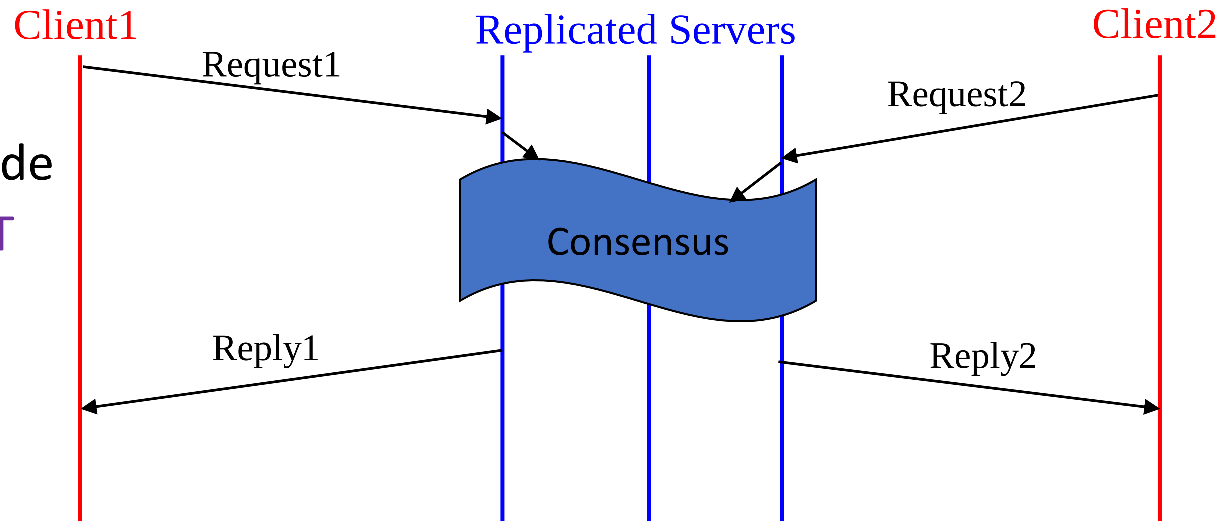
- First blockchains were based on PoW
- The good
  - Zero trust, fully decentralized, claimed to be censorship resistance, “scalable”
- The bad
  - Inherently low transactions rate, probabilistic finality, most hash rate is concentrated in a few mining pools (it is enough to attack the code base of a few mining pools to takeover the system), easy to cheat in (new) coins with low compute power
- The ugly
  - Consumes too much energy: more resources translate into more power consumption per TX, but do not improve the throughput of the system
    - The energy required for a single transaction could power dozens of US households for a day



Permissioned, PoA, and PoS Consensus

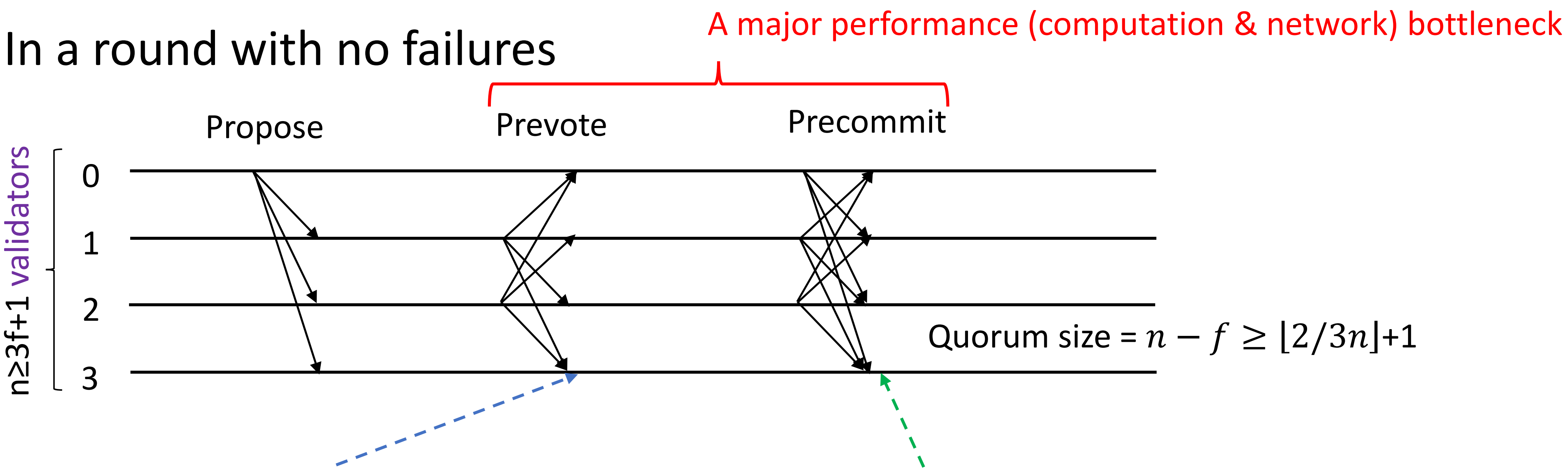
A simple replication protocol

- Clients can send requests to any replica
- All replicas repeatedly run consensus to decide what should be the next transaction (or next batch of transactions = block)
- Seminal PBFT published in 1999
- Famous adaptations to blockchain include
  - Tendermint/Cosmos and IBFT/QBFT



Tendermint/Cosmos (in a Nutshell)

- In a round with no failures



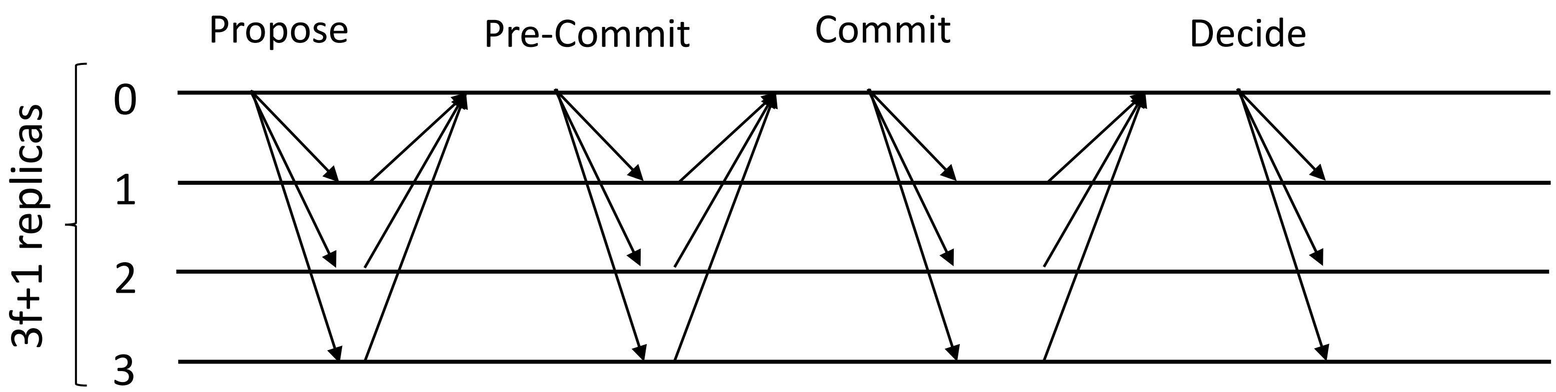
The goal is to prevent equivocation.
That is, ensure that the proposer did not send different blocks to different validators

Here we discover that a quorum knows that the proposer proposed the same valid block to a quorum, so we can accept and finalize the block

Improvements

HotStuff (DiemBFT): BFT Consensus with Linearity and Responsiveness [Yin, Malkhi, Reiter, Gueta, Abraham - PODC 2019]

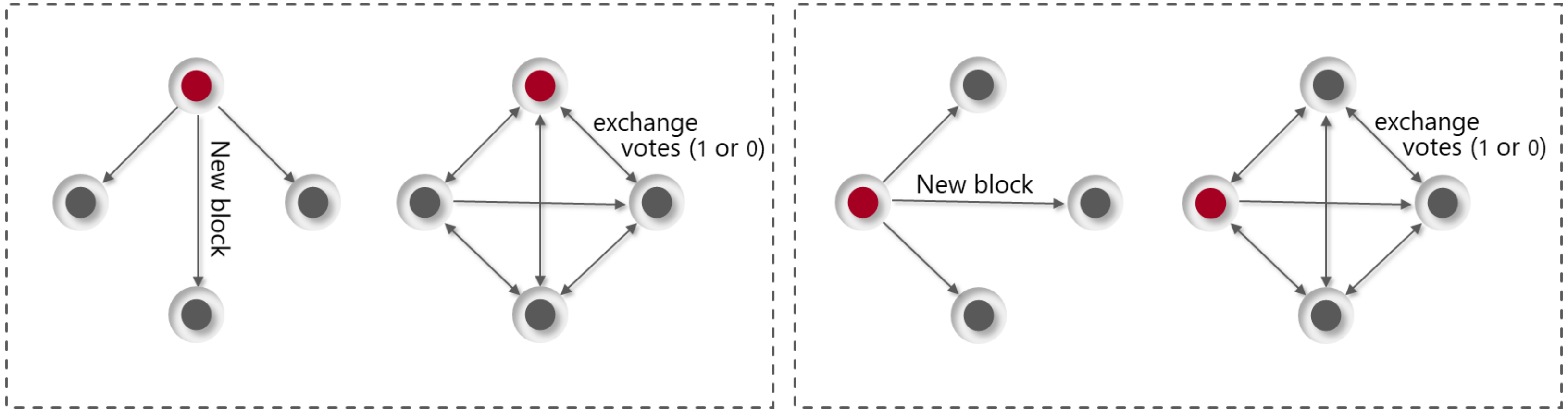
- Optimization 1: one-to-all and all-to-one communication with signatures aggregation
- Optimization 2: a third phase to eliminate timeout when no 2/3+ acks
  - Finality in 3 phases
- Optimization 3: pipelining



Improvements

FireLedger: A High Throughput Blockchain Consensus Protocol [Buchnik, Friedman – VLDB 2020]

- Idea 1: Rotating proposer + do not immediately mask Byzantine attacks
  - The concept of Weak Reliable Broadcast (WRB)

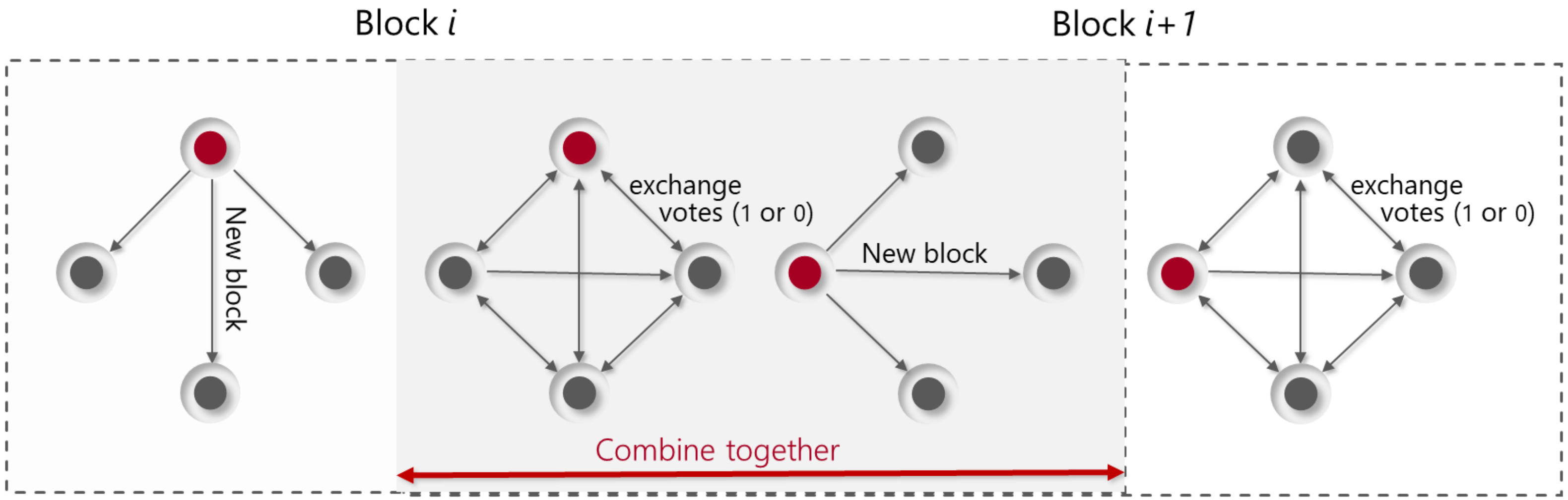


If a Byzantine proposer sends different blocks to different nodes, it is discovered within at most f+1 blocks
=> Run full BFT consensus (e.g., BFT-SMaRt, HotStuff, Asyc) only then
=> Transactions finality takes f+1 rounds (blocks)

Improvements

FireLedger: A High Throughput Blockchain Consensus Protocol [Buchnik, Friedman – VLDB 2020]

- Idea 2: Pipelining - overlap the exchange of block i with proposal of block i+1

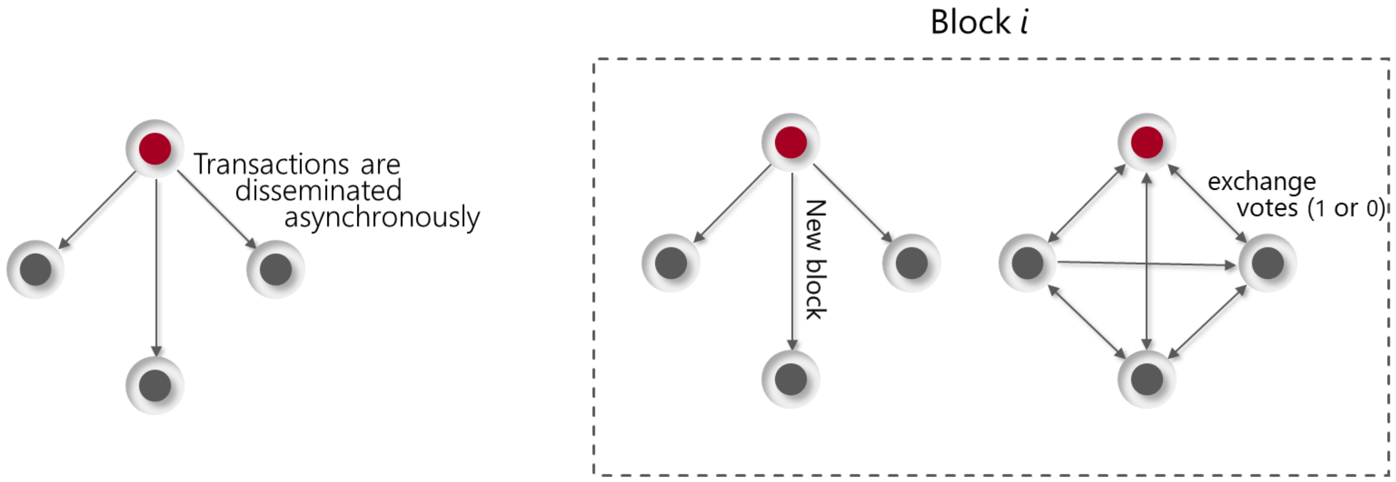


Benefit: Amortized 1-round consensus

Improvements

FireLedger: A High Throughput Blockchain Consensus Protocol [Buchnik, Friedman – VLDB 2020]

- Idea 3: Separating blocks' headers from blocks' data (transactions) dissemination

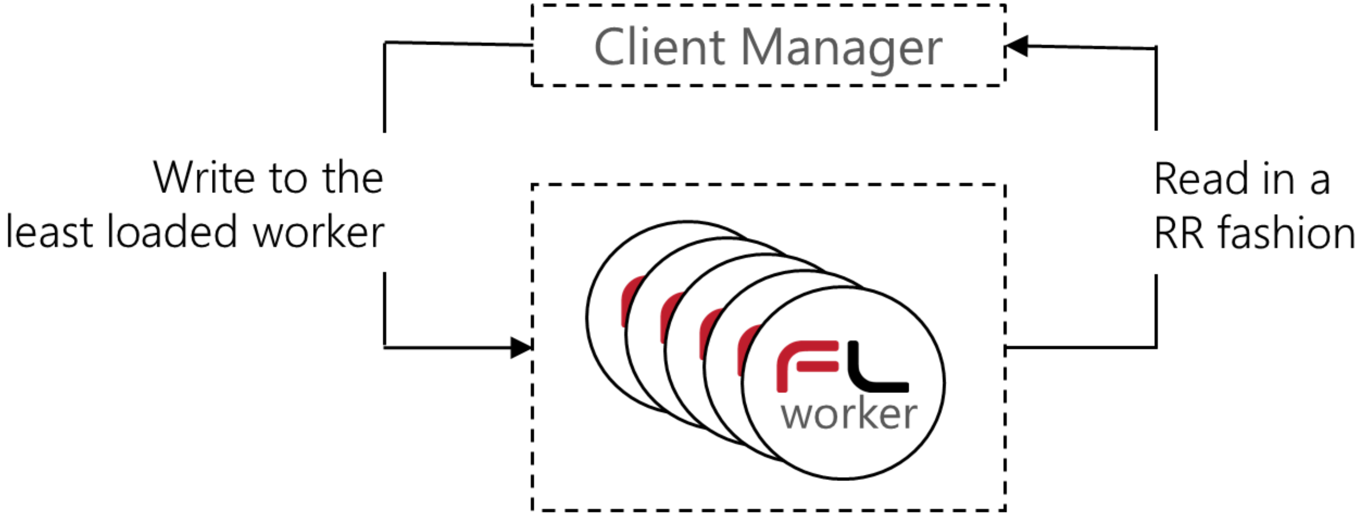


Benefits: ✓ Consensus executed on very small messages ✓ Low bandwidth utilization

Improvements

FireLedger: A High Throughput Blockchain Consensus Protocol [Buchnik, Friedman – VLDB 2020]

- Idea 4: Embrace parallelism
  - Run multiple instances of the protocol, but with a total order among them



Benefits: ✓ Fully utilize multicore machines ✓ Low variability of message size

Improvements

ResilientDB: Global Scale Resilient Blockchain Fabric [Gupta, Rahnama, Hellings, Sadoghi – VLDB 2020]

- Geo-Scale Byzantine FaultTolerant consensus protocol (GeoBFT):
- Scalability by using a topological-aware grouping of replicas in local clusters
- Parallelization of consensus at the local level
- Minimizing inter-cluster communication

10s to 100s of thousands of TPS in wide area deployments

Improvements

Dumbo: Faster Asynchronous BFT Protocols [Guo, Lu, Tang, Xu, Zhang – CCS 2020]

- In each round, every validator proposes a sub-block
- Use probabilistic asynchronous Byzantine consensus protocol to decide which sub-blocks should be accepted to form the next block
  - Does not depend on timing assumptions and is therefore very robust
- Throughput of ~20,000 TPS on 100 nodes committee

Improvements:
- HotStuff – PODC2019
- FireLedger – VLDB 2020
- ResilientDB – VLDB 2020
- Dumbo – CCS 2020

Improvements

Jolteon and Ditto: Network-Adaptive Efficient Consensus with Asynchronous Fallback
[Gelashvili, Kokoris-Kogias, Sonnino, Spiegelman, Xiang – FC 2022]

- A HotStuff like protocol when no failure occur that does not need the 3rd phase => 30% faster
- The recovery protocol is an asynchronous consensus protocol
  - If we had a failure, the system is probably in an unstable state, so better use an asynchronous protocol

Improvements:
- HotStuff – PODC2019
- FireLedger – VLDB 2020
- ResilientDB – VLDB 2020
- Dumbo – CCS 2020
- Jolton&Ditto – FC2022

Improvements

Narwhal and Tusk: A DAG-based Mempool and Efficient BFT Consensus [Danezis, Kogias, Sonnino, Spiegelman – EuroSys 2022]

- Narwhal: client transactions are disseminated using a scalable dissemination protocol while maintaining causality
- Tusk: an asynchronous consensus protocol that utilizes the fact that client transactions are already causally ordered
- Obtains throughput of hundreds of thousands TPS with geo-distributed committees

Improvements

Improvements:

- HotStuff – PODC2019
- FireLedger – VLDB 2020
- ResilientDB – VLDB 2020
- Dumbo – CCS 2020
- Jolton&Ditto – FC2022
- Narwhal&Tusk – EuroSys 2022

Kauri: Scalable BFT Consensus with Pipelined Tree-Based Dissemination and Aggregation
[Neiheiser, Matos, Rodrigues – SOSP 2021]

- Integrates a dissemination tree with the consensus protocol to enable scalability of the validators' committee

- Significantly improves HotStuff's throughput in very large clusters

Improvements

Improvements:
- HotStuff – PODC2019
- FireLedger – VLDB 2020
- ResilientDB – VLDB 2020
- Dumbo – CCS 2020
- Jolton&Ditto – FC2022
- Narwhal&Tusk – EuroSys 2022
- Kauri – SOSP 2021

- Numerous sharding ideas
  - Sharding reduces BFT resilience
  - Performance greatly depends on type of transactions and workload

➢ Summary: lots of ideas on how to order 10K-1M TPS
  - o But can we really obtain similar numbers on real blockchains?

Reality Check

When trying to run a full fledged blockchain, performance drops dramatically

- Discussions with industry
- HyperLedger fabric's limited throughput
- Diablo: A Benchmark Suite for Blockchains (EuroSys 2023)
- Smart Red Belly Blockchain

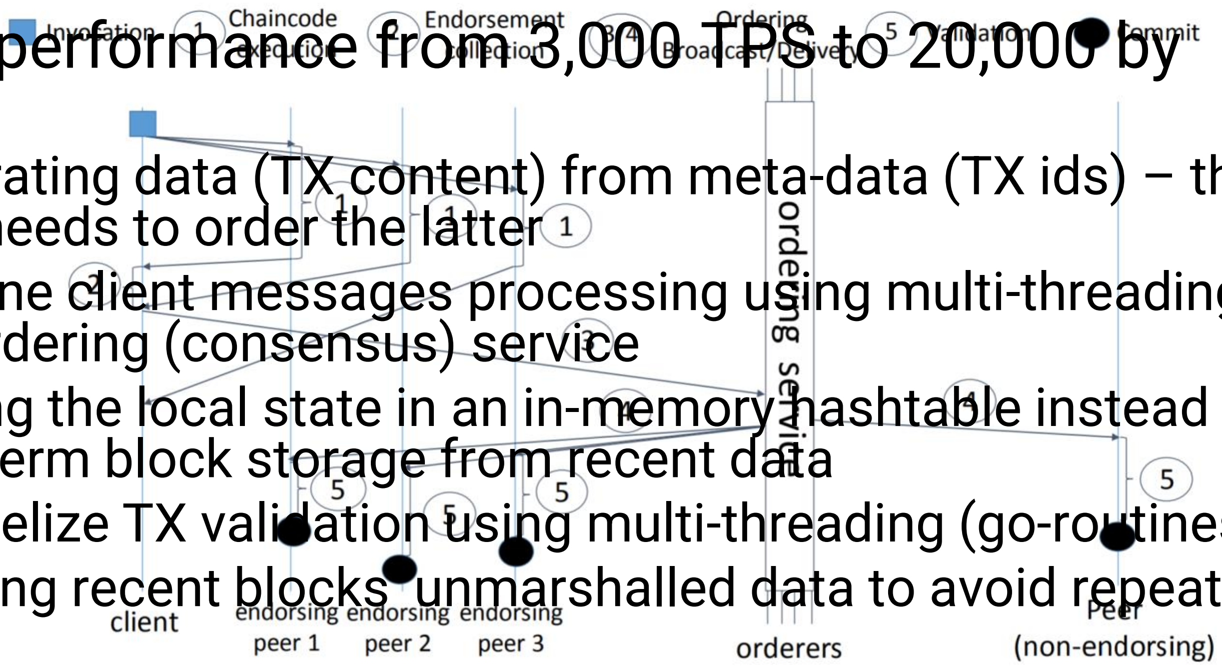
That is, consensus is no longer the performance bottleneck

Systems Aspects of Local BC Execution

FastFabric: Scaling Hyperledger Fabric to 20,000 Transactions per Second [Gorenflo, Lee, Golab, Keshav – ICBC 2019]

- Improved performance from 3,000 TPS to 20,000 by

1. Separating data (TX content) from meta-data (TX ids) – the ordering service only needs to order the latter
2. Pipeline client messages processing using multi-threading with forwarding to the ordering (consensus) service
3. Storing the local state in an in-memory hashtable instead of DB + separating long term block storage from recent data
4. Parallelize TX validation using multi-threading (go-routines)
5. Caching recent blocks unmarshalled data to avoid repeated deserialization



Systems Aspects of Local BC Execution

Smart Red Belly Blockchain: Enhanced Transaction Management for Decentralized Applications [Tennakoon, Gramoli – ArXiv 2022]

- Improved dApp performance from 1,000 TPS to 4,000 by

1. Perform deep validation only on a single validator (block proposer)
2. Divide each block into sub-blocks – process and store sub-blocks, resulting in better overlap of processing and I/O
3. Cache recent data
4. Changed the state data structure
5. Replace the hashing algorithm from Sha-256 to Blake3

What About Concurrent TX Execution?

- CPUs are becoming increasingly parallel
  - An i9-13900K CPU has 24 cores and 32 hardware threads
  - AMD Ryzen™ Threadripper™ PRO 5995WX has 32 cores
  - Xeon CPUs with up to 56 cores (double threads)
  - 4th generation AMD EPYC with 96 cores (later this year, also 128 cores)

- Why is the problem different than standard DB parallelism?

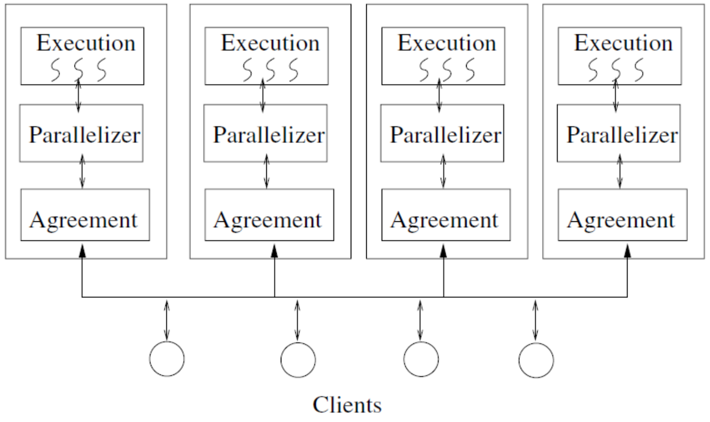
  - Because all validators must execute all transactions in the same (logical) order

Parallelizing BC Transactions' Execution

High Throughput Byzantine Fault Tolerance [Kotla and Dahlin – DSN 2004]

- Enables parallel execution of TXs as long as they maintain the *logical* consensus total order (the CBASE algorithm)

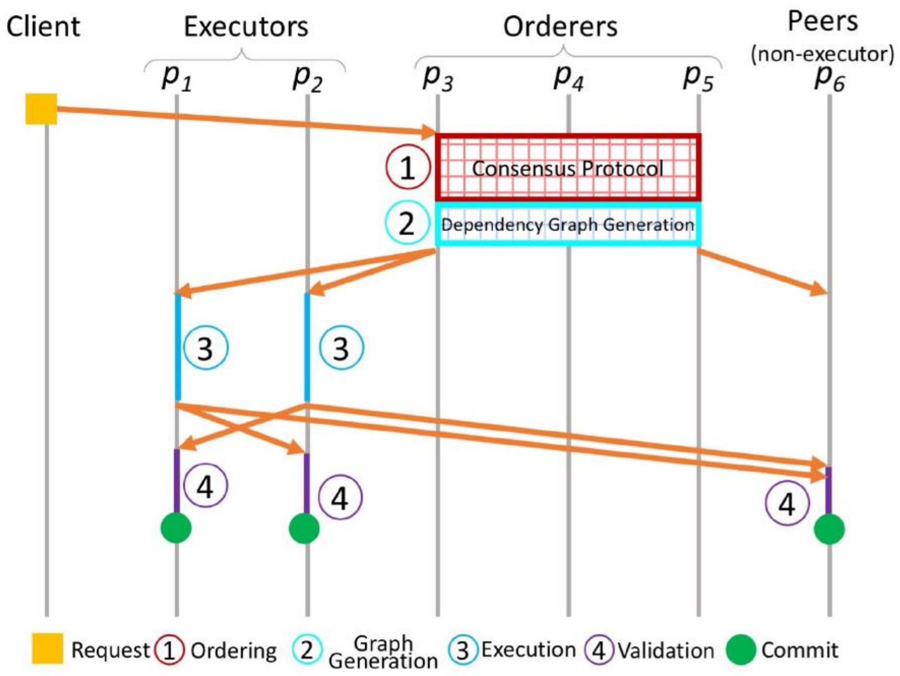
1. Assumes knowledge (possibly conservative) of read sets and write sets
2. Parallelizer ensures that if TX_1 conflicts TX_2 **and** $TX_1 \rightarrow TX_2$ in the Byzantine consensus order, then TX_2 only starts executing after TX_1 has terminated
   a. Maintains a dependency graph
      - The dependency graph is a DAG since the direction of edges is determined by the Byzantine consensus order
   b. A TX is scheduled as soon as all its dependencies in the DAG are removed



Parallelizing BC Transactions' Execution

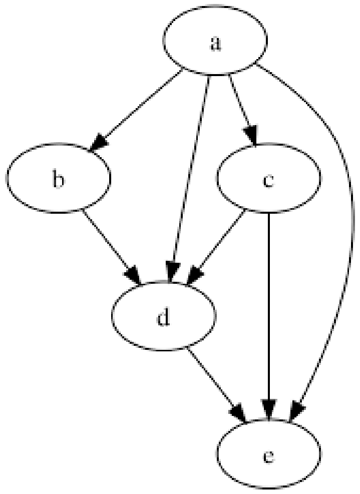
ParBlockchain: Leveraging Transaction Parallelism in Permissioned Blockchain Systems [Amiri, Agrawal, El Abbadi – ICDCS 2019]

- A similar idea to the previous slide, applied to blockchains with a HyperLedger inspired architecture
  - Transactions are divided into blocks
  - Assumes each transaction exposes its read-sets and write sets
    - Known, static analysis, or speculative execution
  - Ordering nodes are separate from execution nodes
  - Ordering nodes repeatedly run consensus on the ordering of transactions
  - An ordering node batches groups of transactions obtained from consensus, according to a deterministic rule, computes their dependency DAG, and disseminates to executors
  - When using a multi-version KV, no need to maintain W-W conflicts in the graph



Parallelizing BC Transactions' Execution

Boosting Concurrency in Parallel State Machine Replication [Escobar, Dotti, Alchieri, Pedone – Middleware 2019]



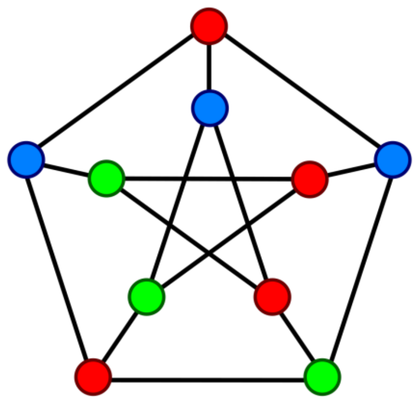
- BFT SMR – not specific to blockchains
- Focuses on how to maintain the DAG concurrently, when transactions arrive continuously
  - Specifically, defined Conflict-Ordered Set (COS) data type
    1. Coarse grain locking (entire structure)
    2. Fine grain (single node) locking through the hand-over-hand paradigm
    3. Lock-free implementation based on AtomicSet, AtomicRead, and CompareAndSwap (CAS)

    Relative results depend on percentage of writes (conflicts) and the execution time for a single TX

Parallelizing BC Transactions' Execution

Coloring Approach – work in progress

- Assume all TXs in the same block can be executed in any agreed upon permutation
- Calculate a minimal (possibly approximate) coloring of the dependency graph
- Ensure execution that obeys the coloring order



Parallelizing BC Transactions’ Execution

Benefits of the Coloring Approach

- If the execution time of all TXs is similar, then the scheduler can simply schedule TXs in phases based on their color
  - This is the fastest schedule and requires no synchronization

TX1: A→B

TX2: B→C

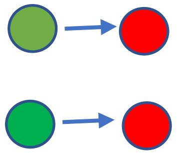
TX3: C→D

TX4: D→A

TO inspired DAG

Linear time

Coloring DAG



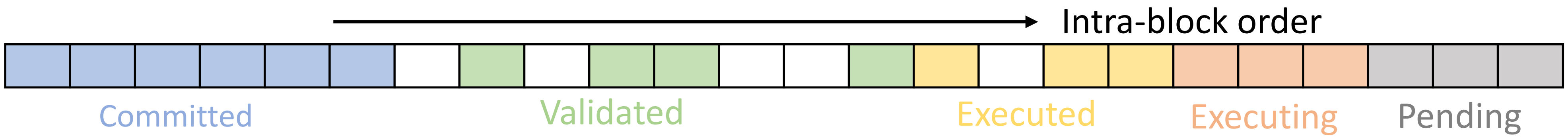
Contant time (2)

- Otherwise, schedule TXs in color order, and impose synchronization in color order
  - Reduces synchronization tracking and overheads
  - Can potentially obtain faster schedules than maintaining the Block order which is arbitrary

Parallelizing BC Transactions' Execution

Block-STM: Scaling Blockchain Execution by Turning Ordering Curse to a Performance Blessing [Gelashvili, Spiegelman, Xiang, Danezis, Li, Malkhi, Xia, Zhou – ArXiv 2022]

- Main goal is to solve the readset/writeset transparent discovery problem
  1. Transactions read and write from a multi-versioned DB
  2. Transactions are scheduled in their block order, but tentatively executed concurrently
  3. Validations occur concurrently, but a transaction only commits if all previous transactions (including itself) have passed validation successfully
  4. If a transaction aborts, it gets re-executed
  5. When a TX aborts, it uses the readset and writeset of the aborted execution as estimated readset and writeset for its re-execution phase



Conclusions

- To help realize blockchains' potential:
  - The research community should invest more efforts on improving the verification and execution time of smart contracts
  - Establish agreed upon benchmarks of dApps execution

Q&A

Thank you