Introduction to Distributed Transactions and Blockchain

ECS 165a: Winter 2022

Slides are adopted from Gupta, Hellings, Sadoghi.

"Fault-tolerant Distributed Transactions on Blockchain". Morgan & Claypool. 2021

UCDAVIS
UNIVERSITY OF CALIFORNIA

ExpoLab
Creativity Unfolded

ResilientDB
Security, Privacy Reloaded

Exploratory Systems Lab at UC Davis

Mission: *To pioneer a resilient data platform at scale, a distributed ledger centered around a democratic and decentralized computational model (ResilientDB Fabric) that further aims to unify secure transactional and real-time analytical processing (L-Store).*

- 3 Ph.D. students, 5 M.Sc. and B.Sc. students.
- Recent papers at VLDB, ICDE, ICDCS, ICDT, DISC, EDBT, Middleware and more.
- Crossroad of distributed databases and blockchains .

Goal: Pioneering Resilient Data Platform at Scale.

Questions

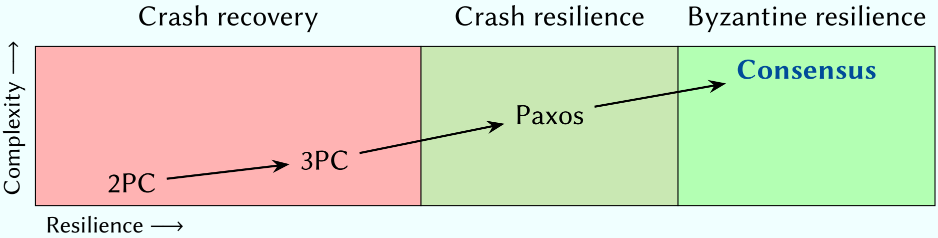
1. Why?
2. What is the relation with blockchains?
3. What do we already have?
4. Where can we improve?
5. What new tools do we need?

Towards high-performance resilient data processing:

Why?

Why resilient data processing?

Go beyond assumptions of traditional transaction processing!

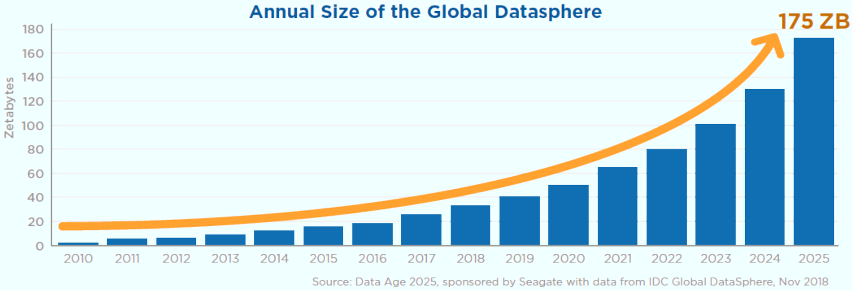


Example

- Provide continuous services during failures.
- Provide services in federated environments.

Why high-performance?

Support requirements of future applications!



- Ever-growing volumes of data (e.g., sensor networks).
- Ever-growing demands of applications (e.g., machine learning).

Towards high-performance resilient data processing:

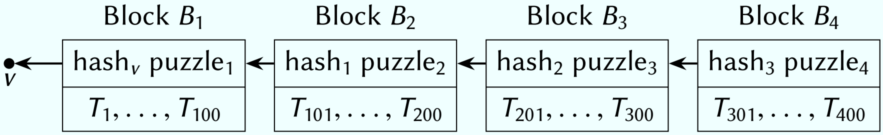
What is the relation with blockchains?

What is a blockchain?

What is a blockchain?

Bitcoin: Management of monetary tokens (Bitcoins)

- Open and decentralized transfer of tokens (*transactions*).
- History of transactions (*ledger*) stored in the blockchain.

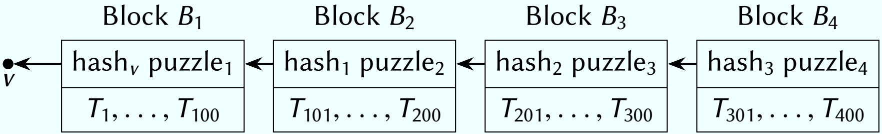


- *Many participants* hold a copy of the blockchain.
- Blockchain structure is *tamper-proof* by design.

What is a blockchain? - Malicious behavior

Bitcoin: Preventing malicious behavior

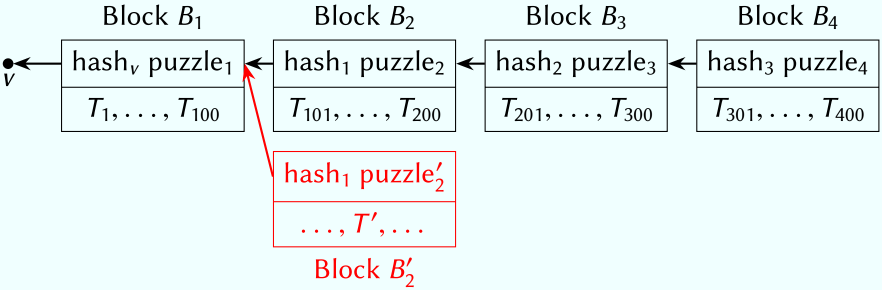
- Malicious attempts to change a chain.



What is a blockchain? - Malicious behavior

Bitcoin: Preventing malicious behavior

- Malicious attempts to change a chain.



- Longest chain has highest incentives.
- Making blocks (solving puzzles) is very costly.
- Malicious attempt leads to a *dead end*.

What is a blockchain? - A definition

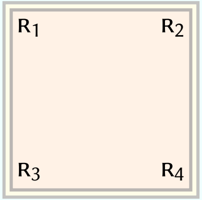
A resilient tamper-proof ledger maintained by many participants.

- *Ledger*.
  Append-only sequence of transactions.
  In database terms: a journal or log.
- *Resilient*.
  High availability via full replication among participants.
- *Tamper-proof*.
  Changes can only be made with majority participation.

Blockchains are *distributed fully-replicated systems*!

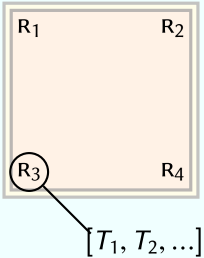
Components of blockchain systems

1. Replicas.



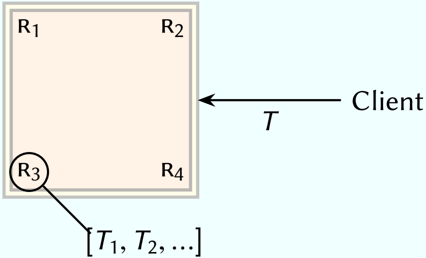
Components of blockchain systems

1. Replicas.
2. Holding the ledger of transactions.



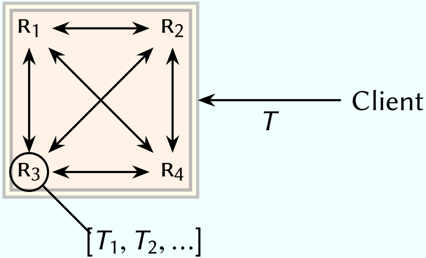
Components of blockchain systems

1. Replicas.
2. Holding the ledger of transactions.
3. Clients with new transactions.



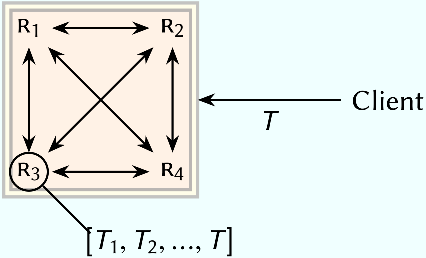
Components of blockchain systems

1. Replicas.
2. Holding the ledger of transactions.
3. Clients with new transactions.
4. Transaction agreement via consensus.



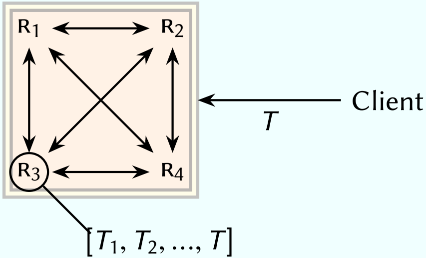
Components of blockchain systems

1. Replicas.
2. Holding the ledger of transactions.
3. Clients with new transactions.
4. Transaction agreement via consensus.
5. Append-only updates to ledger.



Components of blockchain systems

1. Replicas.
2. Holding the ledger of transactions.
3. Clients with new transactions.
4. Transaction agreement via consensus.
5. Append-only updates to ledger.
6. Cryptography.



Bitcoin: A permissionless blockchain

The participants are not known and can change.

Rationale: Fully decentralized and open cryptocurrencies

- Bitcoin, Ethereum, ...
- Scale to thousands of participants.
- Low transaction processing throughput.
- Very high transaction latencies.

We focus on permissioned blockchains

All participants are known.

Rationale: Data processing in managed environment

- Support different attack models than cryptocurrencies.
- Easier to support low latencies and high throughputs.
- Downside: changing participants is hard.

Many ideas also apply to permissionless blockchains.

Towards high-performance resilient data processing:

What do we already have?

We have consensus: PBFT, PAXOS, POW, ...

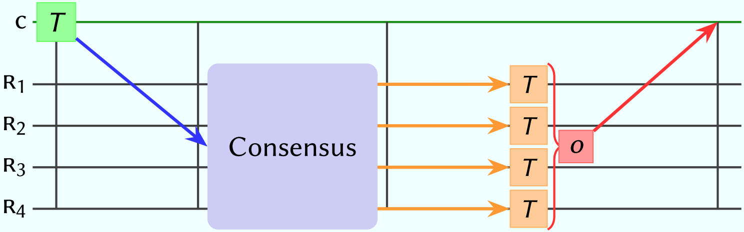
Termination	Each non-faulty replica decides on a transaction.
Non-divergence	Non-faulty replicas decide on the same transaction.

We have consensus: Pbft, Paxos, PoW, ...

Termination	Each non-faulty replica decides on a transaction.
Non-divergence	Non-faulty replicas decide on the same transaction.
Validity	Every decided-on transaction is a client request.
Response	Clients learn about the outcome of their requests.
Service	Every client will be able to request transactions.

We have consensus: PBFT, PAXOS, POW, …

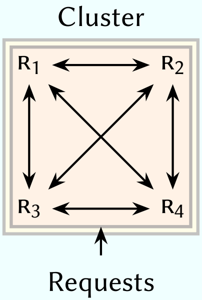
Termination	Each non-faulty replica decides on a transaction.
Non-divergence	Non-faulty replicas decide on the same transaction.
Validity	Every decided-on transaction is a client request.
Response	Clients learn about the outcome of their requests.
Service	Every client will be able to request transactions.



Operating a fully-replicated ledger using consensus

Each replica maintains a copy of the ledger:
Append-only sequence of transactions.

1. Use consensus to select the ρ-th client request T.
2. Append T as the ρ-th entry to the ledger.
3. Execute T as the ρ-th entry, inform client.



Consistent state: Linearizable order and deterministic execution

On identical inputs, execution of transactions at all non-faulty replicas *must produce identical outputs*.

Distributed fully-replicated systems: The CAP Theorem

Consistency Does every participant have exactly the same data?

Availability Does the system continuously provide services?

Partitioning Can the system cope with network disturbances?

Theorem (The CAP Theorem)

Can provide at most two-out-of-three of these properties.

Distributed fully-replicated systems: The CAP Theorem

Consistency Does every participant have exactly the same data?

Availability Does the system continuously provide services?

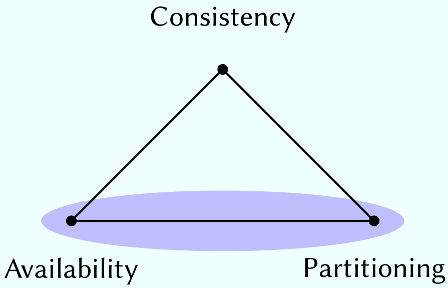
Partitioning Can the system cope with network disturbances?

Theorem (The CAP Theorem)

Can provide at most two-out-of-three of these properties.

CAP Theorem uses narrow definitions!

The CAP Theorem and Blockchains

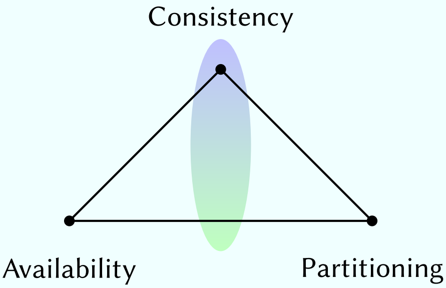


Permissionless Blockchains

Open membership focuses on Availability and Partitioning.

$\implies$ Consistency not guaranteed (e.g., forks).

The CAP Theorem and Blockchains



Permissioned Blockchains

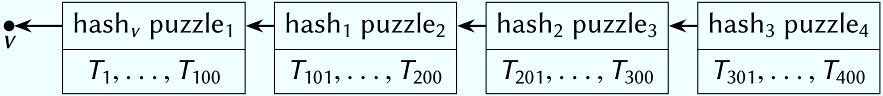
Consistency at all costs.

$\Longrightarrow$ Availability when communication is reliable.

$\Longrightarrow$ Partition-tolerance when network failure is limited and replicas are reliable.

What else do we have?

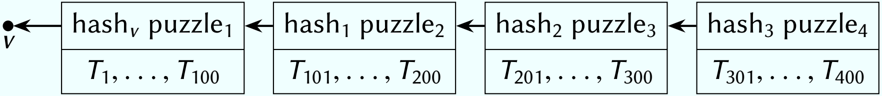
- A lot of *theory* on consensus: consensus is costly.
- PBFT: A practical Byzantine fault-tolerant consensus protocol.
- Tamper-proof *ledgers*.



  Exact details: depend on consensus, application, attack model, …
- Many *cryptographic tools*.

What else do we have?

- A lot of *theory* on consensus: consensus is costly.
- PBFT: A practical Byzantine fault-tolerant consensus protocol.
- Tamper-proof *ledgers*.



Exact details: depend on consensus, application, attack model, …

- Many *cryptographic tools*.

What about high-performance?

Theory on consensus: Summary

Limitations of practical consensus

- No asynchronous communication!
- Dealing with $\mathbf{f}$ malicious failures requires $\mathbf{n} > 3\mathbf{f}$ replicas.
- Worst-case: at least $\Omega(\mathbf{f} + 1)$ phases of communication.
- Worst-case: at least $\Omega(\mathbf{nf})$ signatures and $\Omega\left(\mathbf{n} + \mathbf{f}^2\right)$ messages.
- Network must stay connected when removing $2\mathbf{f}$ replicas.

Consensus in practice

Asynchronous communication, $\mathbf{n} > 3\mathbf{f}$, clique network:

$\implies$ termination only when communication is reliable.

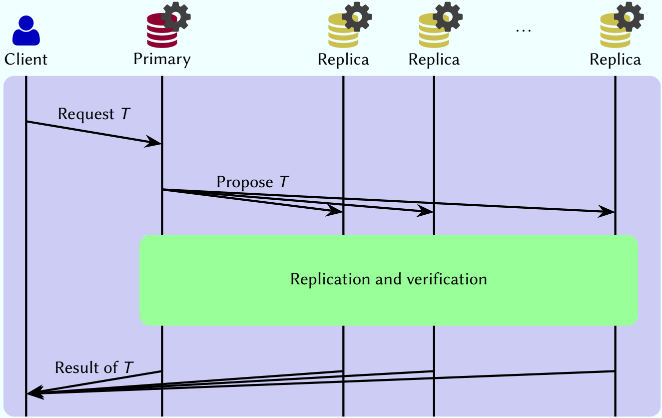
Towards high-performance resilient data processing:

What do we already have? PBFT

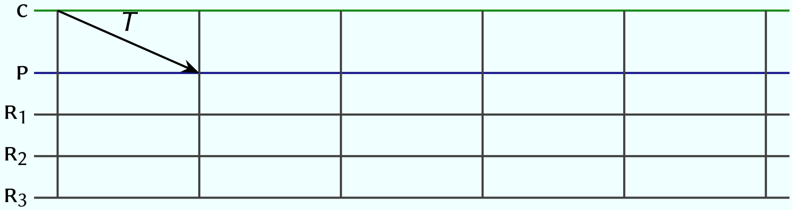
PBFT: Practical Byzantine Fault Tolerance

Primary Coordinates consensus: propose transactions to replicate.

Backup Accept transactions and verifies behavior of primary.

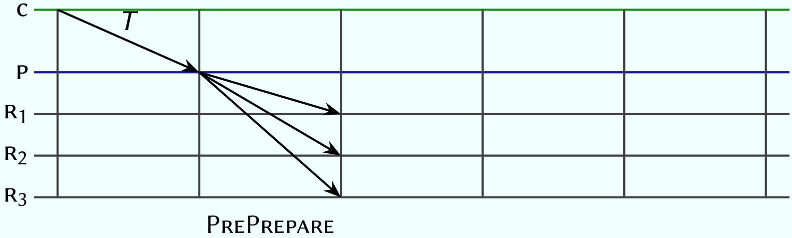


PBFT: Normal-case protocol in view v



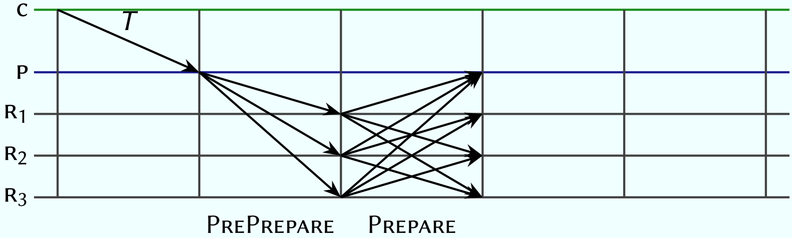
$$\langle T \rangle_{\mathrm{c}}.$$

PBFT: Normal-case protocol in view v



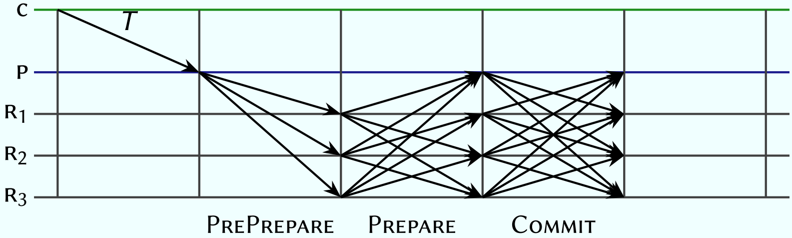
$$\text{PrePrepare}(\langle T \rangle_{c}, v, \rho).$$

PBFT: Normal-case protocol in view v



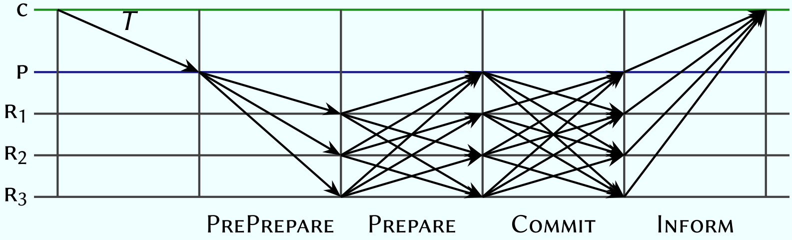
If receive PREPREPARE message m: PREPARE(m).

PBFT: Normal-case protocol in view v



If $\mathbf{n} - \mathbf{f}$ identical PREPARE(m) messages: COMMIT(m).

PBFT: Normal-case protocol in view v



If $\mathbf{n} - \mathbf{f}$ identical COMMIT(m) messages: execute, INFORM($\langle T \rangle_c$, ρ, r).

PBFT: Normal-case consensus

Theorem

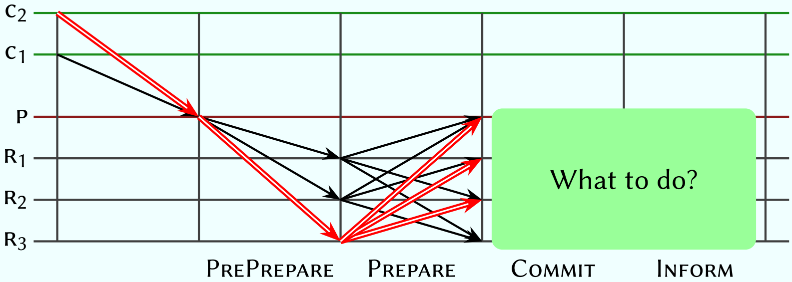
If the primary is non-faulty and communication is reliable,
then the normal-case of PBFT *ensures consensus on T in round ρ.*

PBFT: Normal-case consensus

Theorem

If the primary is non-faulty and communication is reliable, then the normal-case of PBFT *ensures consensus on T in round ρ.*

Example (Byzantine primary, $\mathbf{n} = 4$, $\mathbf{f} = 1$, $\mathbf{n} - \mathbf{f} = 3$)

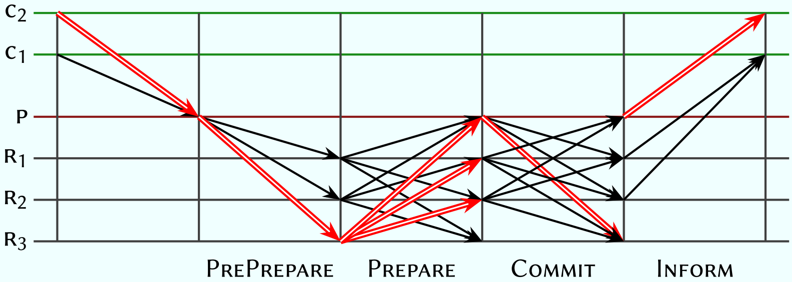


PBFT: Normal-case consensus

Theorem

If the primary is non-faulty and communication is reliable, then the normal-case of PBFT *ensures consensus on T in round ρ.*

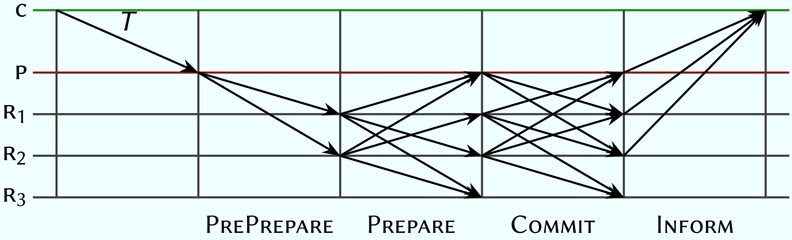
Example (Byzantine primary, $\mathbf{n} = 4$, $\mathbf{f} = 1$, $\mathbf{n} - \mathbf{f} = 3$)



PBFT: Primary failure versus malicious replicas

Primary P is faulty

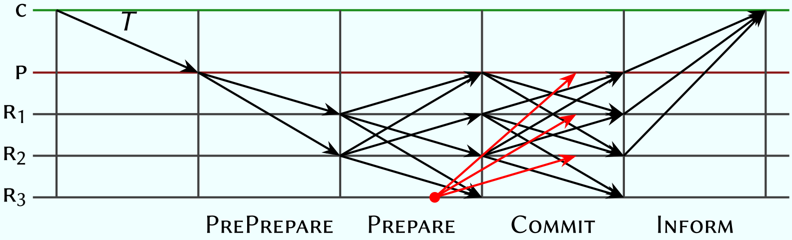
ignores R3



PBFT: Primary failure versus malicious replicas

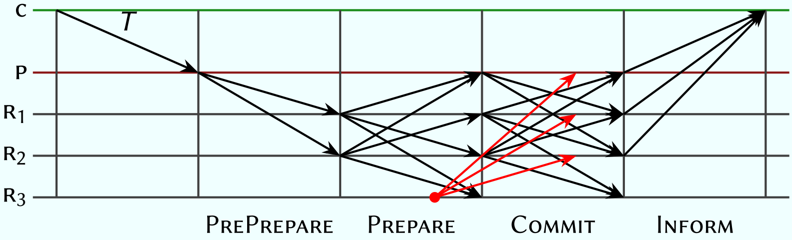
Primary P is faulty

ignores R3

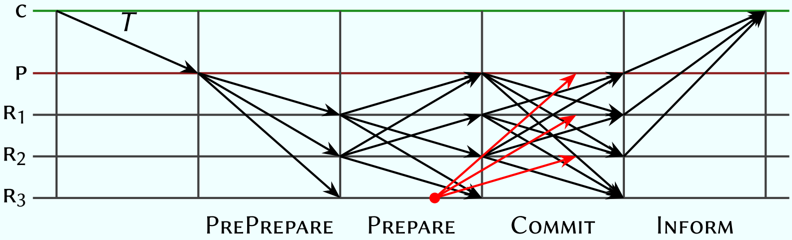


PBFT: Primary failure versus malicious replicas

Primary P is faulty
ignores R3

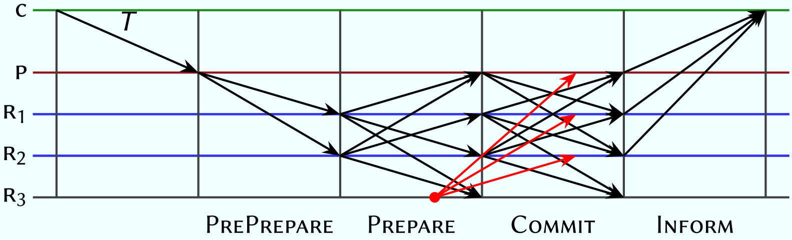


Replica R3 is malicious
pretends to be ignored

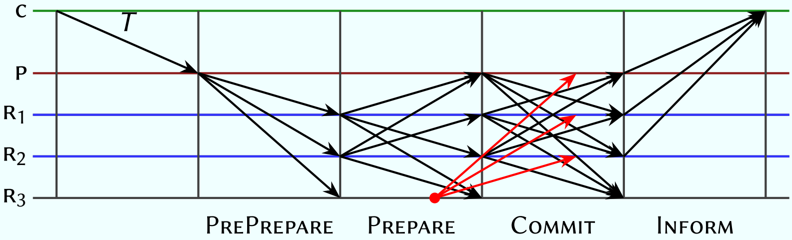


PBFT: Primary failure versus malicious replicas

Primary P is faulty
ignores R_3



Replica R_3 is malicious
pretends to be ignored



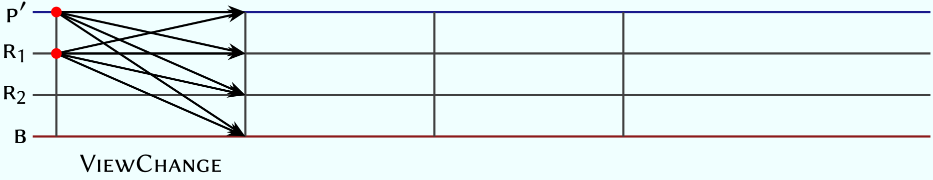
PBFT: Detectable primary failures

If the primary behaves faulty to > **f** non-faulty replicas, then failure of the primary is detectable.

Replacing the primary: View-change at replica R

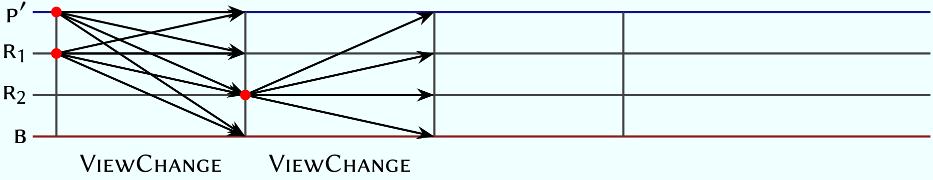
1. R detects *failure* of the current primary P.
2. R chooses a new primary P′ (the next replica).
3. R provides P′ with its *current state*.
4. P′ proposes a *new view*.
5. If the new view is valid, then R switches to this view.

Pbft: A view-change in view v



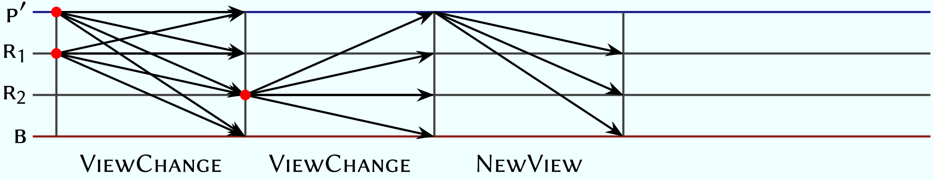
Send ViewChange(E, v) with E all prepared transactions.

PBFT: A view-change in view v



Indirect failure detection by R_2.

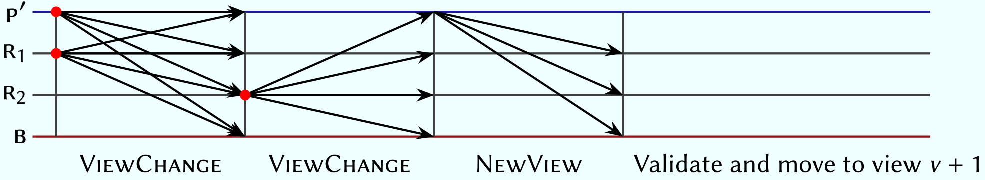
PBFT: A view-change in view v



If $\mathbf{n} - \mathbf{f}$ valid ViewChange(E, v) messages: NewView$(v + 1, \mathcal{E}, \mathcal{N})$.

- $\mathcal{E}$ contains $\mathbf{n} - \mathbf{f}$ valid ViewChange messages.
- $\mathcal{N}$ contains no-op proposals for *missing rounds*.

PBFT: A view-change in view v



Move to view $v + 1$ if NewView$(v + 1, \mathcal{E}, \mathcal{N})$ is valid.

- $\mathcal{E}$ contains $\mathbf{n} - \mathbf{f}$ valid ViewChange messages.
- $\mathcal{N}$ contains no-op proposals for *missing rounds*.

Towards high-performance resilient data processing:

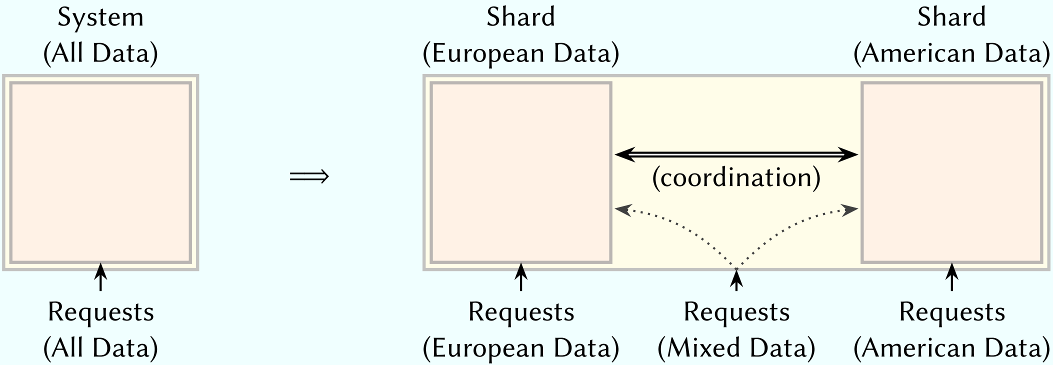
Where can we improve?

A look at high-performance data processing

Scalability: adding resources $\Longrightarrow$ adding performance.

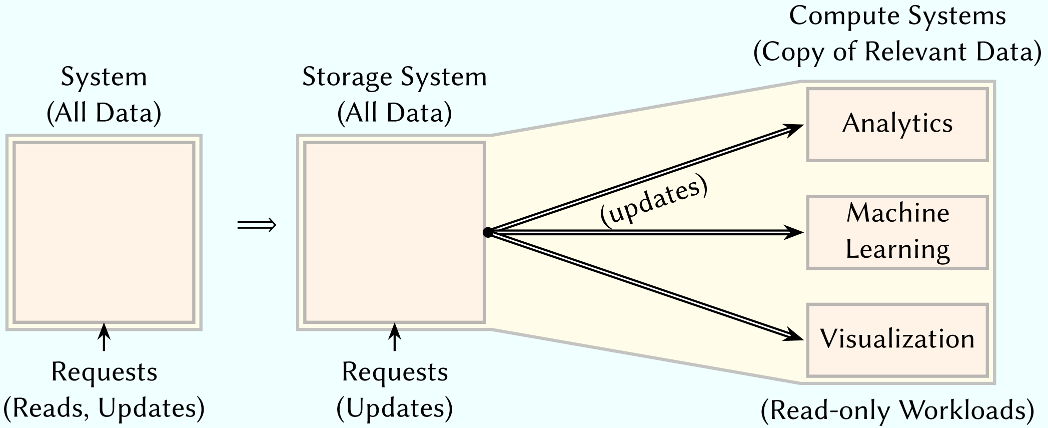
Full replication: adding resources (replicas) $\Longrightarrow$ less performance!

Sharding and Geo-scale aware sharding



Adding shards ⟹ adding throughput (parallel processing), adding storage.

Role Specialization: Read-only workloads



Specializing roles $\Longrightarrow$ adding throughput (parallel processing, specialized hardware, ...).

Towards high-performance resilient data processing:

What new tools do we need?

Central ideas for improvement

Reminder

We can make a resilient cluster that manages data: *blockchains*.

- **Sharding**: make each shard an independent blockchain.
  Requires: *reliable communication between blockchains.*
  Permissionless blockchains: relays, atomic swaps!

- **Role Specialization**: make the storage system a blockchain.
  Requires: *reliable read-only updates of the blockchain.*
  Permissionless blockchains: light clients!

Consensus is of no use here if we want efficiency.

Towards high-performance resilient data processing:

Concluding remarks

Conclusion

We need an extensive toolbox!

	(permissioned)	(permissionless)
▶ Consensus	PBFT, Paxos... GeoBFT, RCC, PoE...	PoW, PoS, ...
▶ Cross-blockchain communication	Cluster-sending... Cerberus...	Relays, atomic swaps
▶ Read-only participation	Byzantine learning	Light clients

High-performance resilient data processing is nearby.

Ongoing work

Initial results are available

- Cluster-Sending: DISC 2019, doi: PDF.
- Wait-free Consensus: DISC 2019, doi: PDF.
- Byzantine Learning: ICDT 2020, doi: PDF.
- Geo-aware Consensus: VLDB 2020, doi: PDF.
- Blockchain Architecture, ICDCS 2020, PDF.
- Concurrent Consensus: ICDE 2021, PDF.
- Proof-of-Execution: EDBT 2021, PDF.
- ByShard: VLDB 2021, PDF.
- RingBFT: EDBT 2022, PDF.

More about us and our work

- ExpoLab Creativity Unfolded `https://expolab.org/`.
- ResilientDB Security, Privacy Reloaded `https://resilientdb.com/`.