Fault-Tolerant Distributed Transactions on Blockchain

Introduction

Suyash Gupta

Jelle Hellings

Mohammad Sadoghi

UCDAVIS
UNIVERSITY OF CALIFORNIA

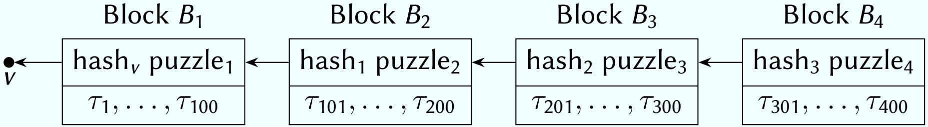
ExpoLab
Creativity Unfolded

McMaster University

A Quick Overview of Bitcoin

What is Bitcoin?

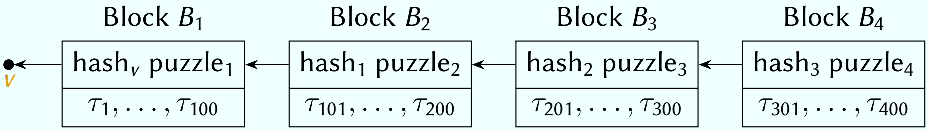
- A *monetary token*, the Bitcoin.
- An open and decentralized way to *transfer* these tokens.
- A representation of the history of these transfers: the *blockchain*:



A Quick Overview of Bitcoin

What is Bitcoin?

- A *monetary token*, the Bitcoin.
- An open and decentralized way to *transfer* these tokens.
- A representation of the history of these transfers: the *blockchain*:

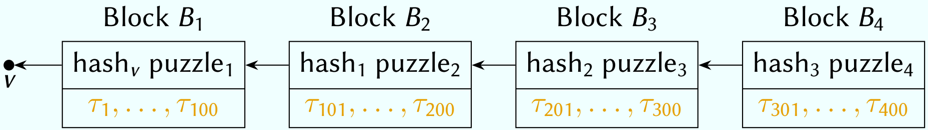


  - A chain of blocks starting at a predefined initial *genesis block* v.

A Quick Overview of Bitcoin

What is Bitcoin?

- A *monetary token*, the Bitcoin.
- An open and decentralized way to *transfer* these tokens.
- A representation of the history of these transfers: the *blockchain*:

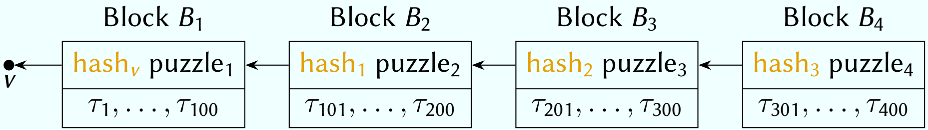


  - Each block holds a *list of transactions* transferring Bitcoins.

A Quick Overview of Bitcoin

What is Bitcoin?

- A *monetary token*, the Bitcoin.
- An open and decentralized way to *transfer* these tokens.
- A representation of the history of these transfers: the *blockchain*:

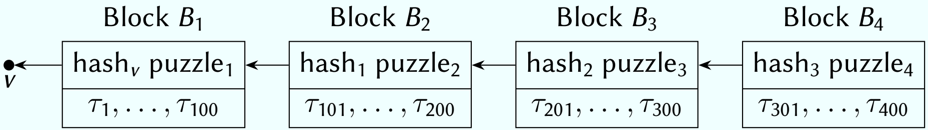


  - Each block holds the *hash* of a previous block.

A Quick Overview of Bitcoin

What is Bitcoin?

- A *monetary token*, the Bitcoin.
- An open and decentralized way to *transfer* these tokens.
- A representation of the history of these transfers: the *blockchain*:

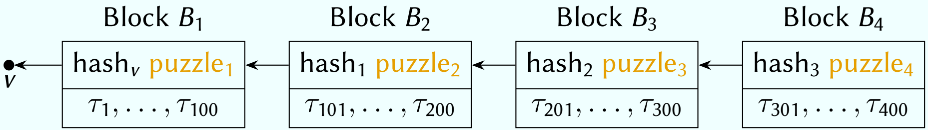


- Incentive: The creator of a block that is part of the chain gets a reward.

A Quick Overview of Bitcoin

What is Bitcoin?

- A *monetary token*, the Bitcoin.
- An open and decentralized way to *transfer* these tokens.
- A representation of the history of these transfers: the *blockchain*:

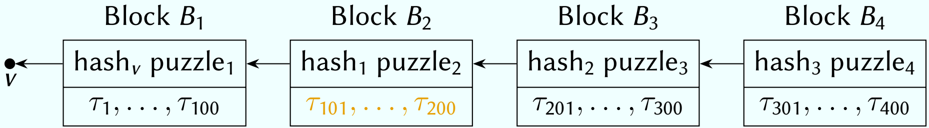


  - Each block contains contains a solution to a computational *complex puzzle*, used to make the blockchain *tamper-proof*.

The Tamper-Proof Design of Bitcoin

Incentive: The creator of a block that is part of the chain gets a reward.

Assumption: A malicious participant P wants to replace τ_{123} by τ'

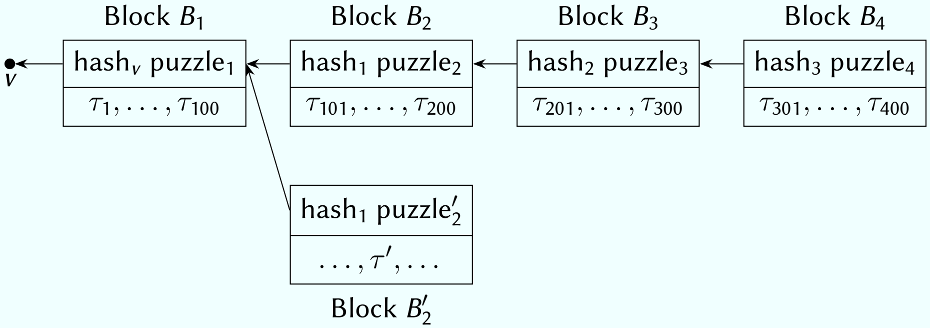


The Tamper-Proof Design of Bitcoin

Incentive: The creator of a block that is part of the chain gets a reward.

Assumption: A malicious participant P wants to replace τ_{123} by τ'

1. P creates a new block B_2' with τ'.

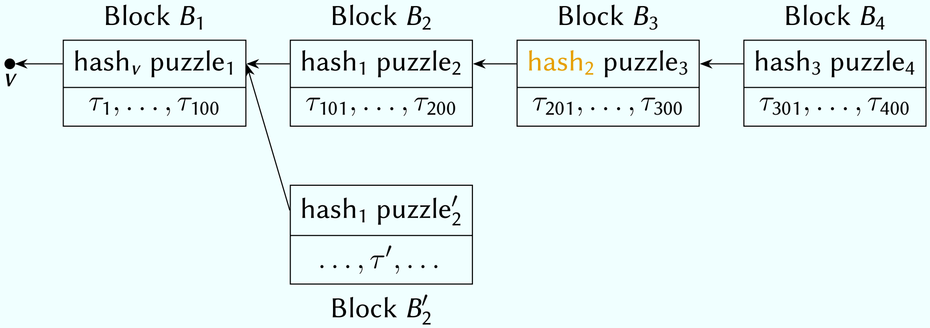


The Tamper-Proof Design of Bitcoin

Incentive: The creator of a block that is part of the chain gets a reward.

Assumption: A malicious participant P wants to replace τ_{123} by τ'

2. $B_2 \neq B_2'$. Hence, *$hash_2$* still points to B_2.

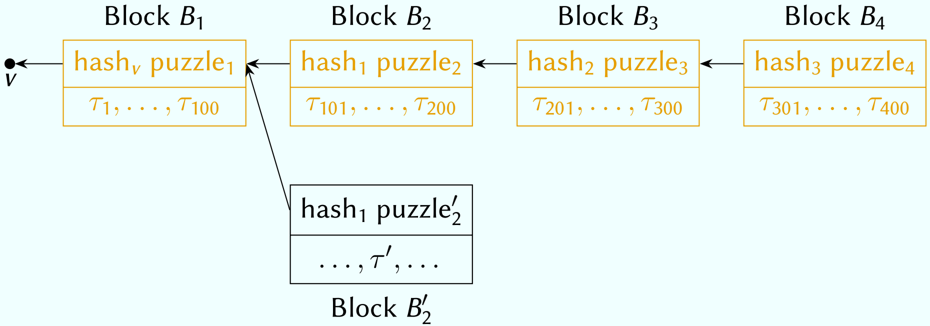


The Tamper-Proof Design of Bitcoin

Incentive: The creator of a block that is part of the chain gets a reward.

Assumption: A malicious participant P wants to replace τ_{123} by τ'

3. Good participants prefer to work on *long chains* over *short chains*.



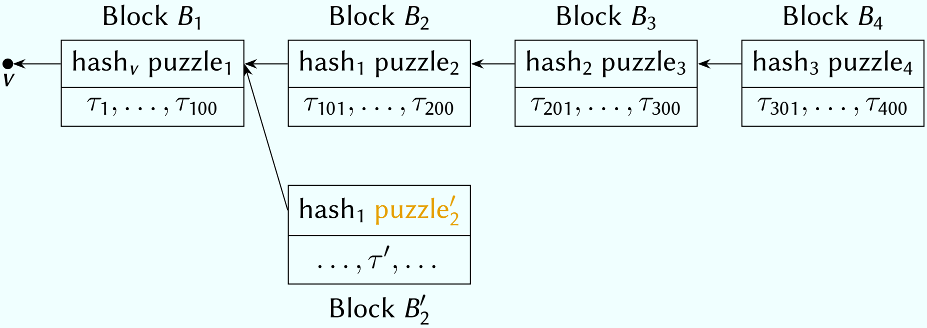
The Tamper-Proof Design of Bitcoin

Incentive: The creator of a block that is part of the chain gets a reward.

Assumption: A malicious participant P wants to replace τ_{123} by τ'

3. Good participants prefer to work on *long chains* over *short chains*.

   Long chains give rewards to *more participants*: incentive to work on long chains.

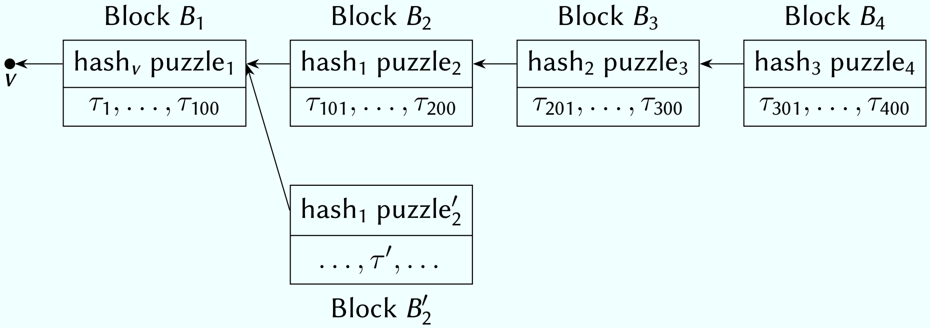


The Tamper-Proof Design of Bitcoin

Incentive: The creator of a block that is part of the chain gets a reward.

Assumption: A malicious participant P wants to replace τ_{123} by τ'

4. Complex puzzles prevent P from easily adding blocks to B_2'.



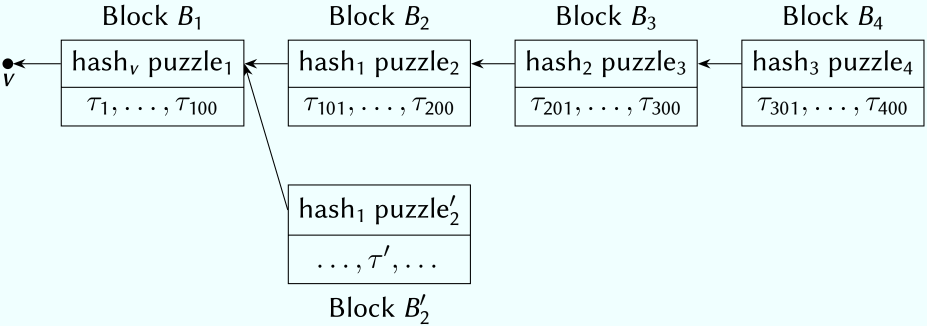
The Tamper-Proof Design of Bitcoin

Incentive: The creator of a block that is part of the chain gets a reward.

Assumption: A malicious participant P wants to replace τ_{123} by τ'

4. Complex puzzles prevent P from easily adding blocks to B_2'.

   More incentives to continue from block B_4 than block B_2'!



Permissioned Blockchains

Permissioned blockchains have a well-defined set of *vetted* participants: the identities of participants is known to all!

Permissioned Blockchains

Permissioned blockchains have a well-defined set of *vetted* participants: the identities of participants is known to all!

- ✖ No easy way to change the participants.
- ✖ Dozens of participants.
- ✔ Many thousands of transactions per second with low latencies.
- ✔ Strong consensus guarantees: no data inconsistencies.

Permissioned Blockchains

Permissioned blockchains have a well-defined set of *vetted* participants:
the identities of participants is known to all!

- ✖ No easy way to change the participants.
- ✖ Dozens of participants.
- ✔ Many thousands of transactions per second with low latencies.
- ✔ Strong consensus guarantees: no data inconsistencies.

Example: A e-health system managed by a consortium of health-care providers

- Each *vetted participant* manages their own systems (e.g., different software).
- Failure of individual participants should not break the system!
- Blockchains: *federated data management* and *resilience*.

Permissioned Blockchains

Permissioned blockchains have a well-defined set of *vetted* participants:
the identities of participants is known to all!

- ✖ No easy way to change the participants.
- ✖ Dozens of participants.
- ✔ Many thousands of transactions per second with low latencies.
- ✔ Strong consensus guarantees: no data inconsistencies.

Example: A e-health system managed by a consortium of health-care providers

- Each *vetted participant* manages their own systems (e.g., different software).
- Failure of individual participants should not break the system!
- Blockchains: *federated data management* and *resilience*.

Can be implemented using PBFT-style consensus.

Permissionless Blockchains

Permissionless blockchains have *open membership*:
anyone can participate, at any time, for any duration.

Permissionless Blockchains

Permissionless blockchains have *open membership*:
anyone can participate, at any time, for any duration.

- ✔ Participants are not known and can change.
- ✔ Thousands of participants.
- ✘ Few transactions per second with very high latencies.
- ✘ Weak consensus guarantees: data can be inconsistent (*forks*).

Permissionless Blockchains

Permissionless blockchains have *open membership*:
anyone can participate, at any time, for any duration.

- ✔ Participants are not known and can change.
- ✔ Thousands of participants.
- ✘ Few transactions per second with very high latencies.
- ✘ Weak consensus guarantees: data can be inconsistent (*forks*).

Example: Cryptocurrencies such as Bitcoin and Ethereum

Permissionless Blockchains

Permissionless blockchains have *open membership*:
anyone can participate, at any time, for any duration.

- ✔ Participants are not known and can change.
- ✔ Thousands of participants.
- ✘ Few transactions per second with very high latencies.
- ✘ Weak consensus guarantees: data can be inconsistent (*forks*).

Example: Cryptocurrencies such as Bitcoin and Ethereum

Challenge: *Sybil attacks*

A single entity can impersonate *many* participants to gain unfair control over the system.

Permissionless Blockchains

Permissionless blockchains have *open membership*:
anyone can participate, at any time, for any duration.

- ✔ Participants are not known and can change.
- ✔ Thousands of participants.
- ✖ Few transactions per second with very high latencies.
- ✖ Weak consensus guarantees: data can be inconsistent (*forks*).

Example: Cryptocurrencies such as Bitcoin and Ethereum

Challenge: *Sybil attacks*

A single entity can impersonate *many* participants to gain unfair control over the system.

Requires a consensus protocol that does not rely on *identities*.

Distributed Systems

Definition[1]

A distributed system is a collection of autonomous computing elements that appears to its users as a single coherent system.

[1] M. van Steen & A.S. Tanenbaum, *Distributed Systems*, 3rd ed., `distributed-systems.net`, 2017.

Distributed Systems

Definition[1]

A distributed system is a collection of autonomous computing elements that appears to its users as a single coherent system.

- Autonomous computing elements: nodes, replica, ...

[1] M. van Steen & A.S. Tanenbaum, *Distributed Systems*, 3rd ed., `distributed-systems.net`, 2017.

Distributed Systems

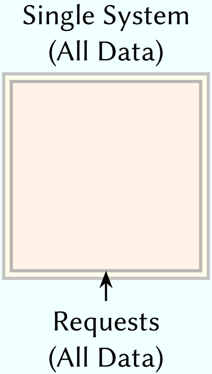
Definition[1]

A distributed system is a collection of autonomous computing elements that appears to its users as a single coherent system.

- Autonomous computing elements: nodes, replica, ...
- Single coherent system: "should behave as a single system".
  - Users should see a *single* system.
  - Nodes must *collaborate* to provide that system.

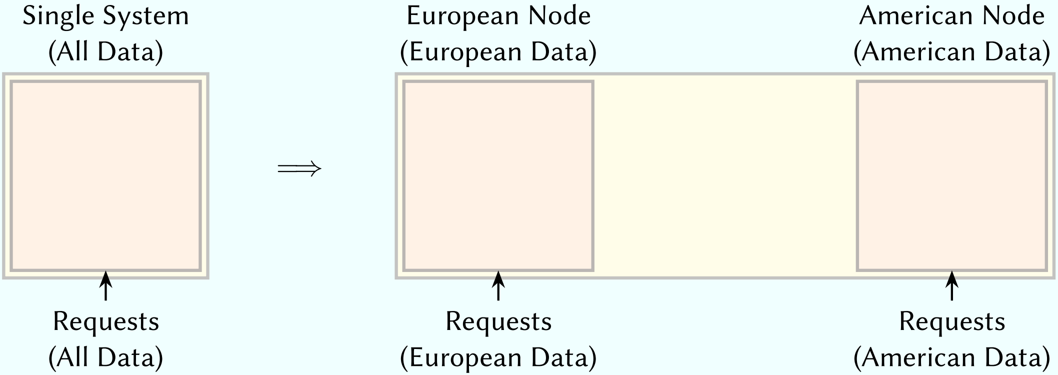
[1] M. van Steen & A.S. Tanenbaum, *Distributed Systems*, 3rd ed., distributed-systems.net, 2017.

Distributed Systems: Scalability



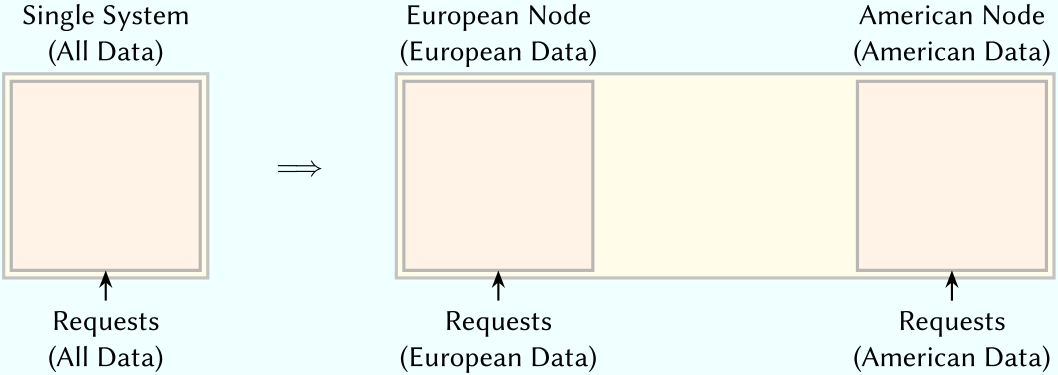
Single system: Storage and performance *bounded* by hardware.

Distributed Systems: Scalability



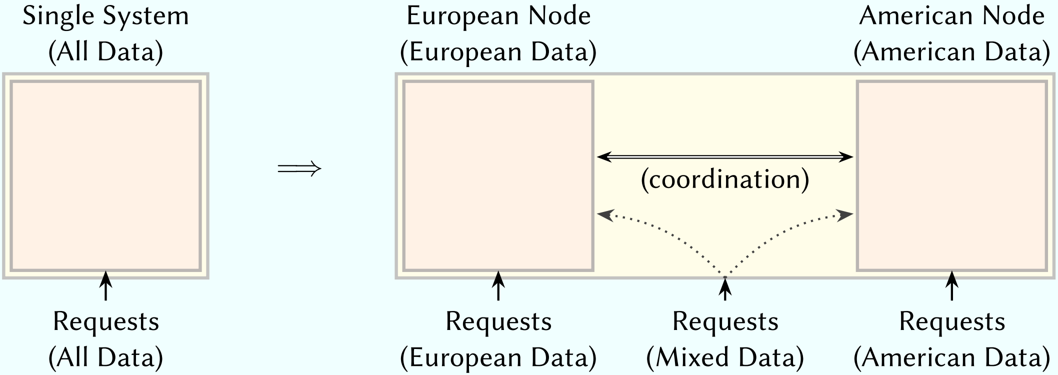
Partition the system: More storage and *potentially* more performance.

Distributed Systems: Scalability



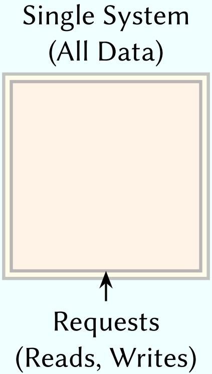
Partition the system: More storage and *potentially* more performance.
Potentially *lower latencies* if data ends up closer to users.

Distributed Systems: Scalability



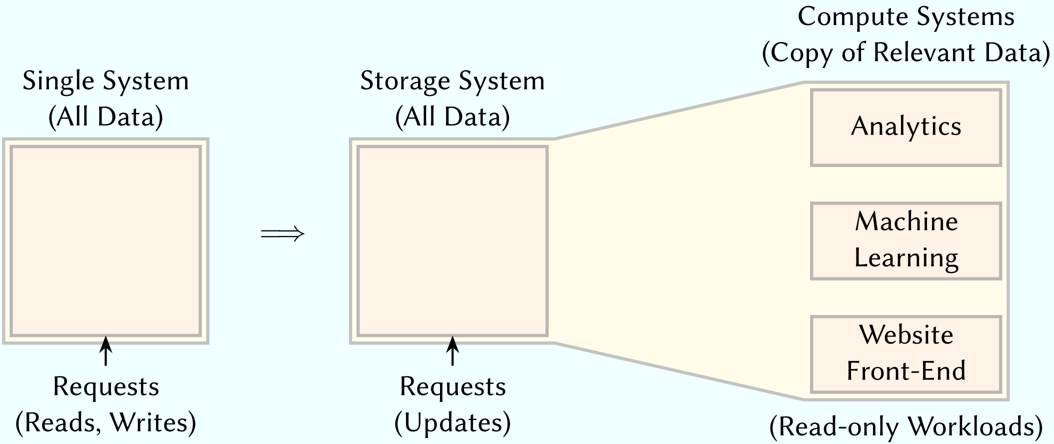
Complex requests: become more *costly* to answer!

Distributed Systems: Specialization



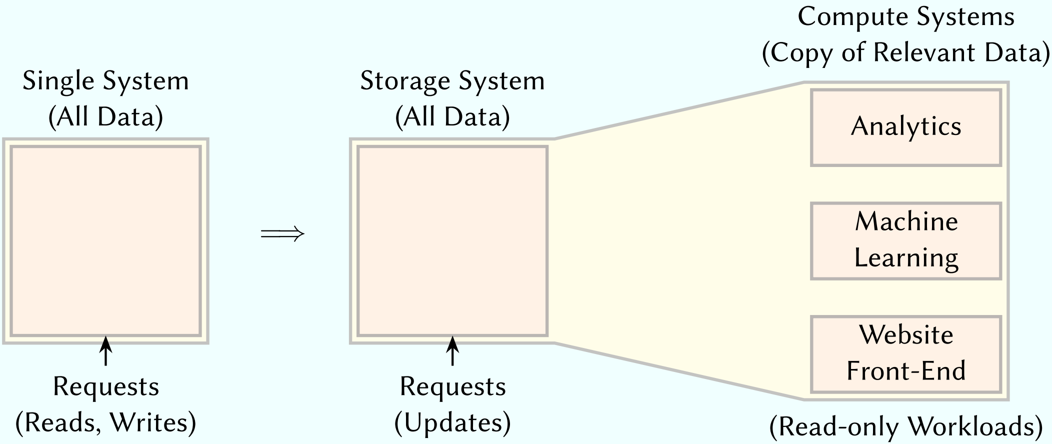
Single system: Compromise—cannot be optimized for *all* tasks.

Distributed Systems: Specialization



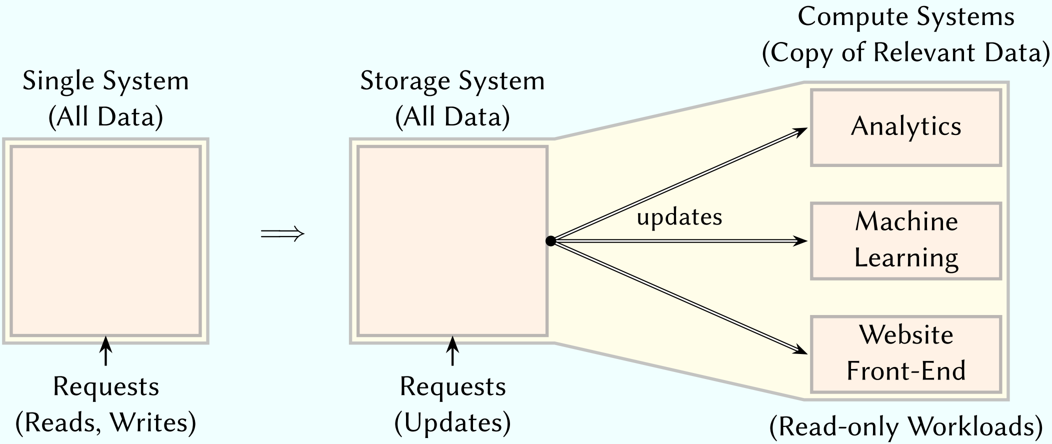
Specialize the system: Different nodes have distinct tasks.

Distributed Systems: Specialization



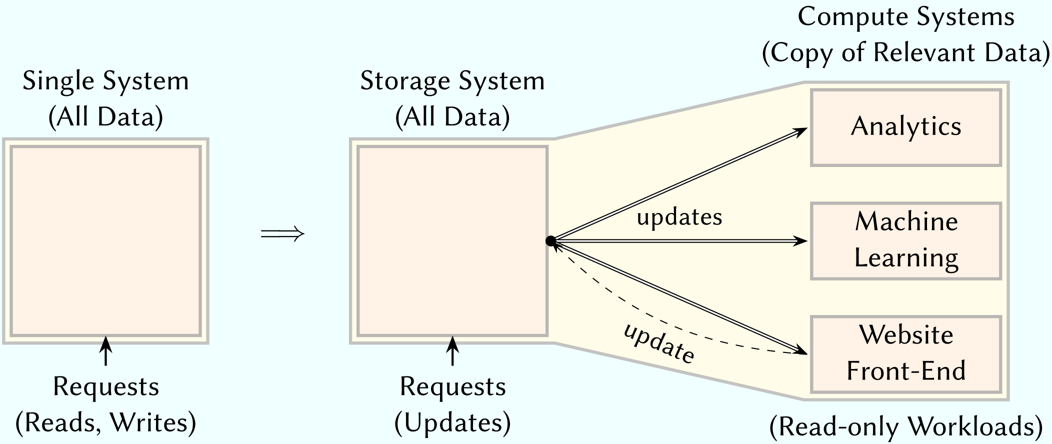
Specialize the system: Different nodes have distinct tasks.
Specialized hardware and software *per* task.

Distributed Systems: Specialization



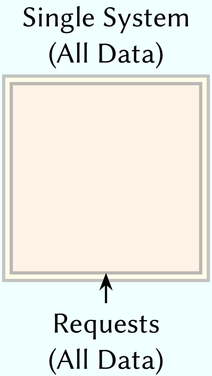
Added cost: Keeping the compute systems *up-to-date*.

Distributed Systems: Specialization



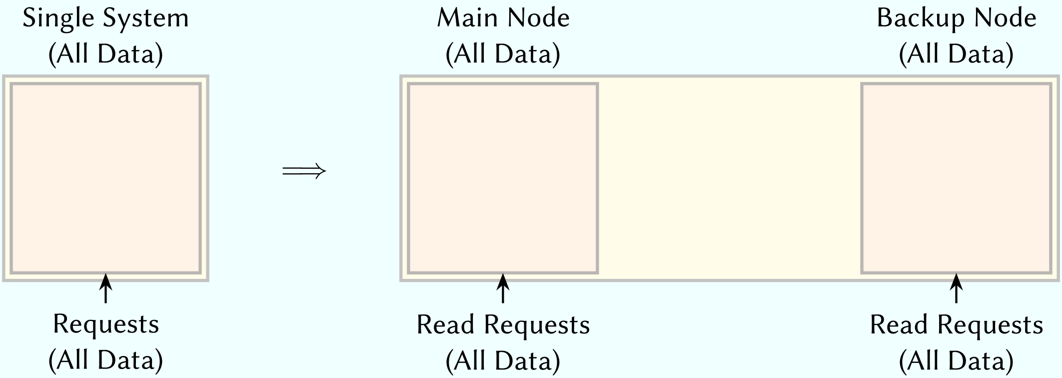
Design complexity: Updates from the compute systems?

Distributed Systems: Reliability (Primary-Backup)



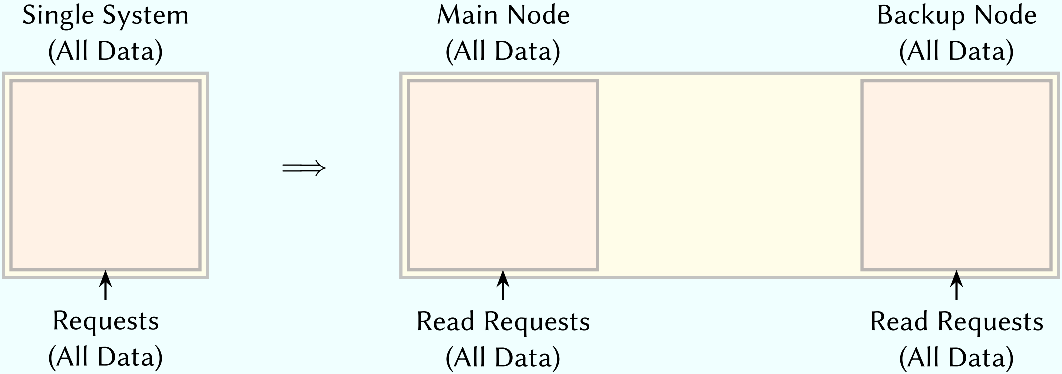
Single system: Single point of *failure*.

Distributed Systems: Reliability (Primary-Backup)



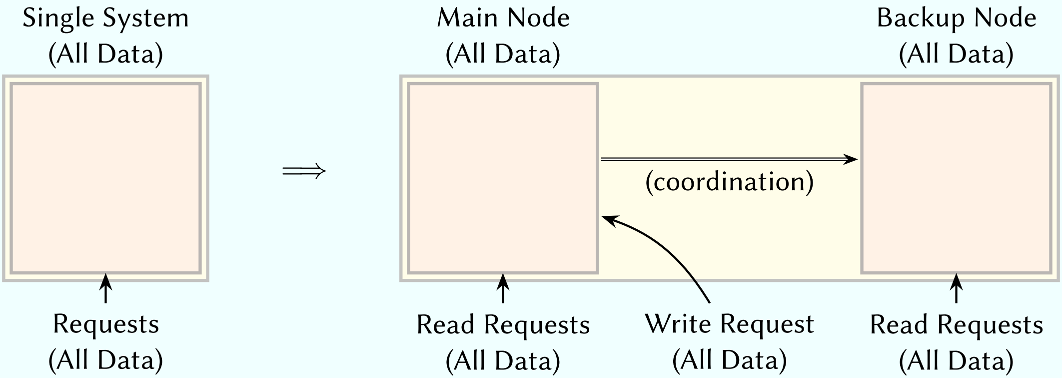
Multiple copies: Copies available after single failure.

Distributed Systems: Reliability (Primary-Backup)



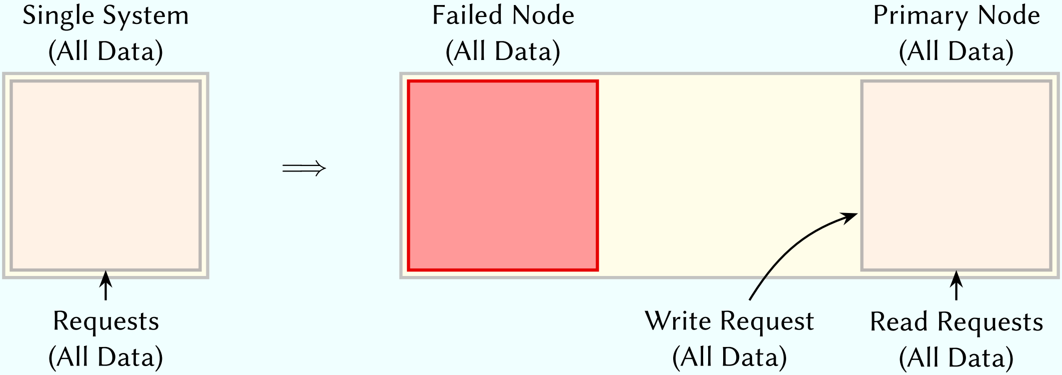
Multiple copies: Copies available after single failure.
Potentially *lower latencies & more performance* when users read data.

Distributed Systems: Reliability (Primary-Backup)



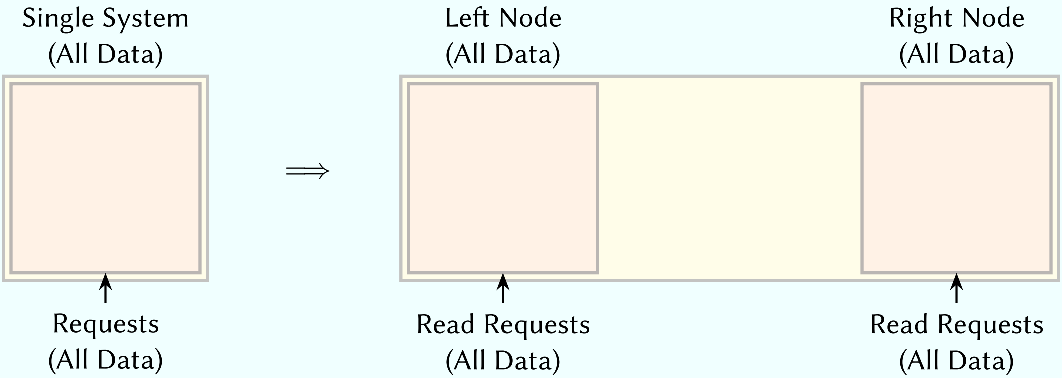
Changes to data requests: Primary *leads*, backups *follow*.

Distributed Systems: Reliability (Primary-Backup)



Failure: Recovery mechanisms—typically *complex*.

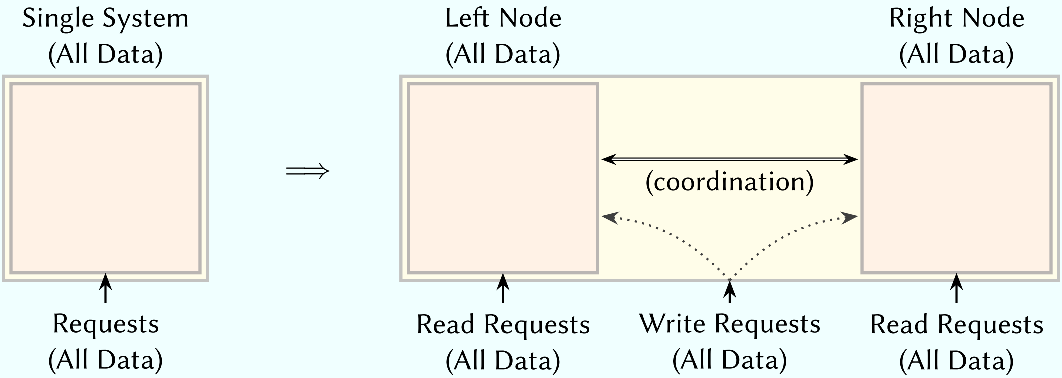
Distributed Systems: Reliability (Decentralized)



Multiple copies: Copies available after single failure.

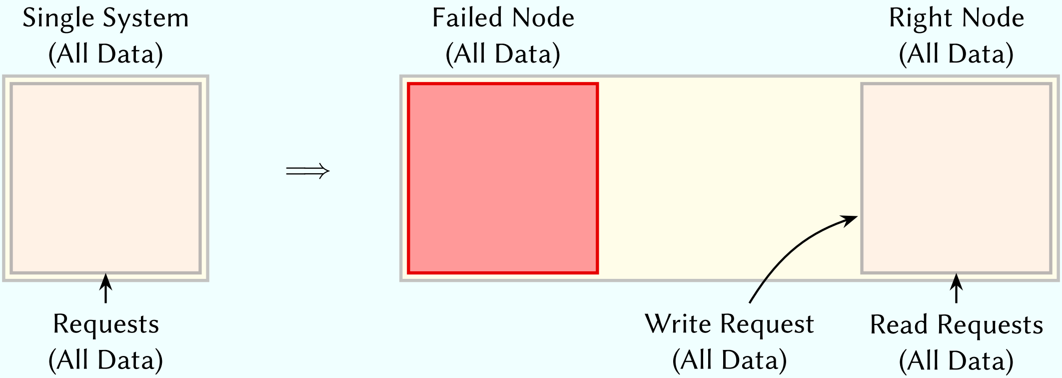
Potentially *lower latencies & more performance* when users read data.

Distributed Systems: Reliability (Decentralized)



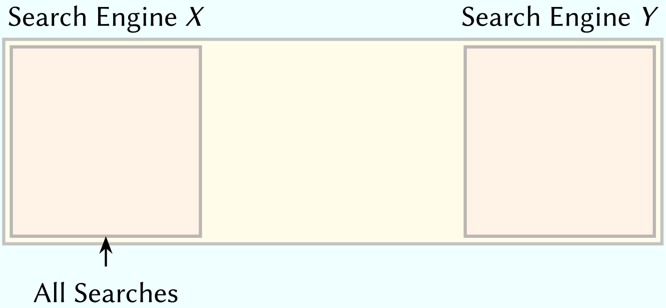
Changes to data requests: *Costly* coordinated decision among all active nodes.

Distributed Systems: Reliability (Decentralized)



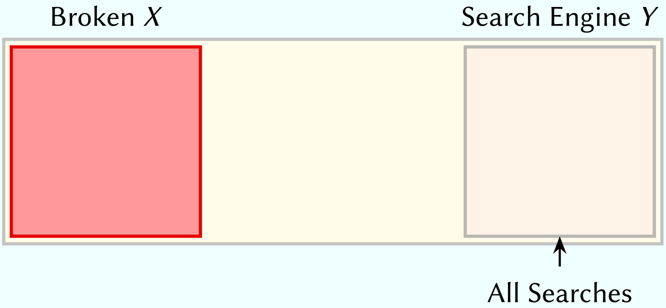
Failure: Recovery mechanisms—typically *easier* (or even *free*).

Distributed System?



I perform all my searches on the web with Search Engine X.

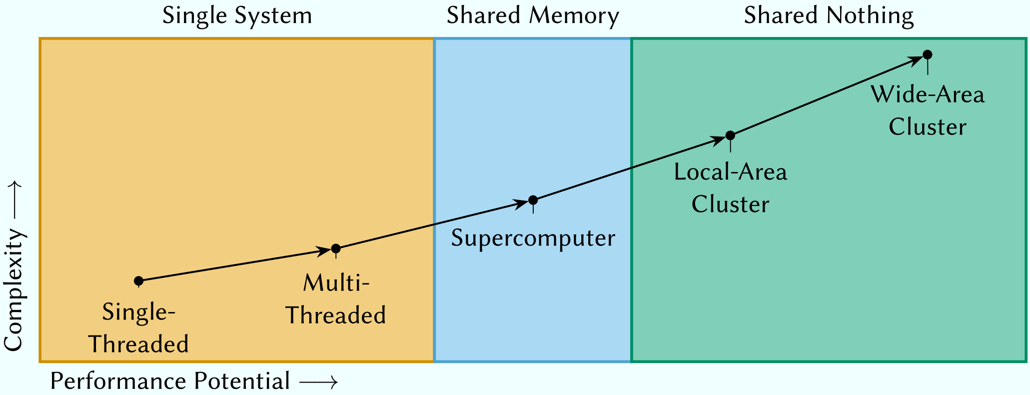
Distributed System?



Today Search Engine X failed—I remembered the alternative Search Engine Y.

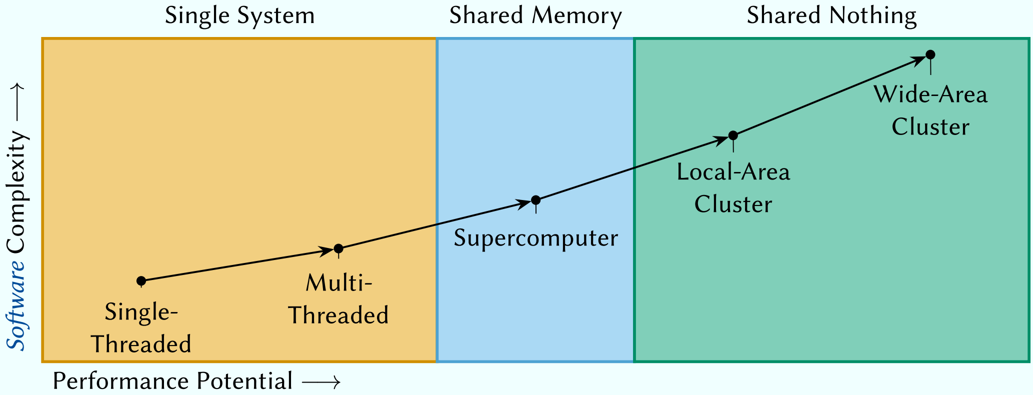
Do we need Distributed Systems?

Distributed computing is *complex*



Do we need Distributed Systems?

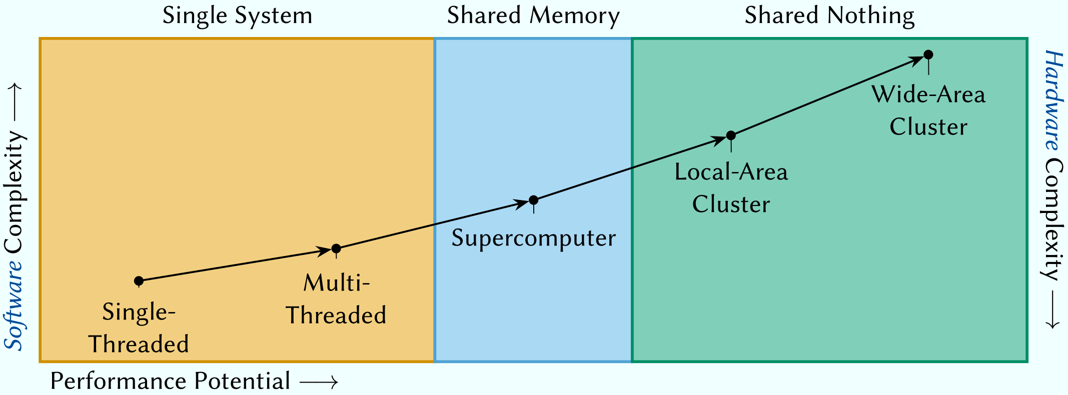
Distributed computing is *complex*



Complexity to write *efficient* software that solve a given problem.

Do we need Distributed Systems?

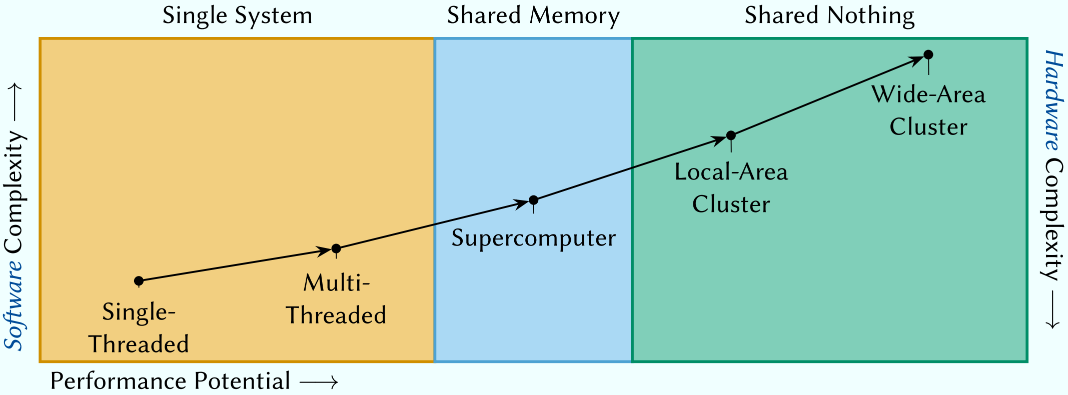
Distributed computing is *complex*



Complexity of *hardware* to solve a given problem in time t.

Do we need Distributed Systems?

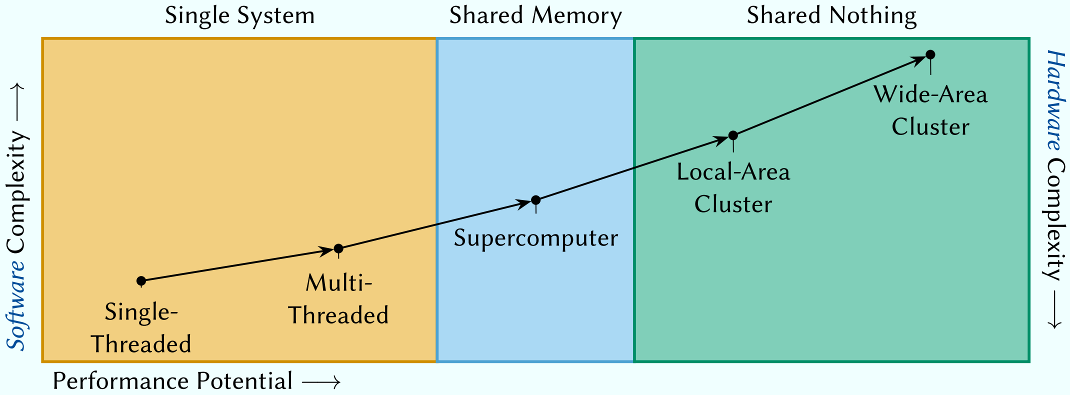
Distributed computing is *complex*



Increased software complexity is *fundamental*: Problems become theoretically harder.

Do we need Distributed Systems?

Distributed computing is *complex*



Do we really need distributed systems?

Distributed System Need: Resilience

Consider: a *service* with a very high SLA (may almost never stop).

Distributed System Need: Resilience

Consider: a *service* with a very high SLA (may almost never stop).

Causes of failure

- Software failure
- Deployment failure
- Hardware failure
- Network failure
- Datacenter failure
- Malicious attacks

Distributed System Need: Resilience

Consider: a *service* with a very high SLA (may almost never stop).

Causes of failure

- Software failure — Code practices? Regression tests? *Independent implementations?*
- Deployment failure
- Hardware failure
- Network failure
- Datacenter failure
- Malicious attacks

Distributed System Need: Resilience

Consider: a *service* with a very high SLA (may almost never stop).

Causes of failure

- Software failure — Code practices? Regression tests? *Independent implementations?*
- Deployment failure — Deployment testing? *Staged deployment?*
- Hardware failure
- Network failure
- Datacenter failure
- Malicious attacks

Distributed System Need: Resilience

Consider: a *service* with a very high SLA (may almost never stop).

Causes of failure

- Software failure — Code practices? Regression tests? *Independent implementations?*
- Deployment failure — Deployment testing? *Staged deployment?*
- Hardware failure — Use redundant, hot-swappable hardware? *Hot spares?*
- Network failure
- Datacenter failure
- Malicious attacks

Distributed System Need: Resilience

Consider: a *service* with a very high SLA (may almost never stop).

Causes of failure

- Software failure — Code practices? Regression tests? *Independent implementations?*
- Deployment failure — Deployment testing? *Staged deployment?*
- Hardware failure — Use redundant, hot-swappable hardware? *Hot spares?*
- Network failure — Network redundancy? *Remote hot spares?*
- Datacenter failure
- Malicious attacks

Distributed System Need: Resilience

Consider: a *service* with a very high SLA (may almost never stop).

Causes of failure

- Software failure — Code practices? Regression tests? *Independent implementations?*
- Deployment failure — Deployment testing? *Staged deployment?*
- Hardware failure — Use redundant, hot-swappable hardware? *Hot spares?*
- Network failure — Network redundancy? *Remote hot spares?*
- Datacenter failure — *Remote hot spares?*
- Malicious attacks

Distributed System Need: Resilience

Consider: a *service* with a very high SLA (may almost never stop).

Causes of failure

- Software failure — Code practices? Regression tests? *Independent implementations?*
- Deployment failure — Deployment testing? *Staged deployment?*
- Hardware failure — Use redundant, hot-swappable hardware? *Hot spares?*
- Network failure — Network redundancy? *Remote hot spares?*
- Datacenter failure — *Remote hot spares?*
- Malicious attacks — *Decentralized & independent implementations?*

Distributed System Need: Resilience

Consider: a *service* with a very high SLA (may almost never stop).

Causes of failure

- Software failure — Code practices? Regression tests? *Independent implementations?*
- Deployment failure — Deployment testing? *Staged deployment?*
- Hardware failure — Use redundant, hot-swappable hardware? *Hot spares?*
- Network failure — Network redundancy? *Remote hot spares?*
- Datacenter failure — *Remote hot spares?*
- Malicious attacks — *Decentralized & independent implementations?*

Distributed designs can provide resilience.

Desirable Properties of a Distributed System

Consider a simple & minimalistic distributed system

Fully-replicated: each node (replica) holds the *same data*.

[2] E. Brewer, *CAP Twelve Years Later: How the “Rules” Have Changed*, 2012.

Desirable Properties of a Distributed System

Consider a simple & minimalistic distributed system

Fully-replicated: each node (replica) holds the *same data*.

The CAP Properties[2]

[2] E. Brewer, *CAP Twelve Years Later: How the "Rules" Have Changed*, 2012.

Desirable Properties of a Distributed System

Consider a simple & minimalistic distributed system

Fully-replicated: each node (replica) holds the *same data*.

The CAP Properties[2]

Consistency All replicas have a single up-to-date copy of the data.
(All replicas always have the same date).

[2] E. Brewer, *CAP Twelve Years Later: How the "Rules" Have Changed*, 2012.

Desirable Properties of a Distributed System

Consider a simple & minimalistic distributed system

Fully-replicated: each node (replica) holds the *same data*.

The CAP Properties[2]

Consistency All replicas have a single up-to-date copy of the data.
(All replicas always have the same date).

Availability All data is always accessible & updateable.
(The system can always provide services to clients).

[2] E. Brewer, *CAP Twelve Years Later: How the "Rules" Have Changed*, 2012.

Desirable Properties of a Distributed System

Consider a simple & minimalistic distributed system

Fully-replicated: each node (replica) holds the *same data*.

The CAP Properties[2]

Consistency: All replicas have a single up-to-date copy of the data. (All replicas always have the same date).

Availability: All data is always accessible & updateable. (The system can always provide services to clients).

Partitioning: The system can tolerate network partitions. (The system can cope with network failures).

[2] E. Brewer, *CAP Twelve Years Later: How the "Rules" Have Changed*, 2012.

Desirable Properties of a Distributed System

Consider a simple & minimalistic distributed system

Fully-replicated: each node (replica) holds the *same data*.

The CAP Properties[2]

Consistency — All replicas have a single up-to-date copy of the data. (All replicas always have the same date).

Availability — All data is always accessible & updateable. (The system can always provide services to clients).

Partitioning — The system can tolerate network partitions. (The system can cope with network failures).

Theorem

A fully-replicated system can only privide two-out-of-three CAP properties.

[2] E. Brewer, *CAP Twelve Years Later: How the "Rules" Have Changed*, 2012.

The CAP Theorem: Revisited[2]

The CAP Theorem lacks nuance.

Consistency All replicas have a single up-to-date copy of the data.

Availability All data is always accessible & updateable.

Partitioning The system can tolerate network partitions.

[2] E. Brewer, *CAP Twelve Years Later: How the "Rules" Have Changed*, 2012.

The CAP Theorem: Revisited[2]

The CAP Theorem lacks nuance.

Consistency All replicas have a single up-to-date copy of the data.
Nuance: Perfect consistency is a very high bar & will limit performance.
Alternatives: eventual consistency & state convergence.

Availability All data is always accessible & updateable.

Partitioning The system can tolerate network partitions.

[2] E. Brewer, *CAP Twelve Years Later: How the "Rules" Have Changed*, 2012.

The CAP Theorem: Revisited[2]

The CAP Theorem lacks nuance.

Consistency All replicas have a single up-to-date copy of the data.
Nuance: Perfect consistency is a very high bar & will limit performance.
Alternatives: eventual consistency & state convergence.

Availability All data is always accessible & updateable.
Nuance: What about latency?
Alternatives: Read-only mode (e.g., offline mode), flexible response times.

Partitioning The system can tolerate network partitions.

[2] E. Brewer, *CAP Twelve Years Later: How the "Rules" Have Changed*, 2012.

The CAP Theorem: Revisited[2]

The CAP Theorem lacks nuance.

Consistency All replicas have a single up-to-date copy of the data.
Nuance: Perfect consistency is a very high bar & will limit performance.
Alternatives: eventual consistency & state convergence.

Availability All data is always accessible & updateable.
Nuance: What about latency?
Alternatives: Read-only mode (e.g., offline mode), flexible response times.

Partitioning The system can tolerate network partitions.
Nuance: For non-mobile systems, partition are rare.
Alternatives: aim at CA & compensation after partition recovery.

[2] E. Brewer, *CAP Twelve Years Later: How the "Rules" Have Changed*, 2012.

The CAP Theorem: Revisited[2]

The CAP Theorem lacks nuance.

Consistency All replicas have a single up-to-date copy of the data.
Nuance: Perfect consistency is a very high bar & will limit performance.
Alternatives: eventual consistency & state convergence.

Availability All data is always accessible & updateable.
Nuance: What about latency?
Alternatives: Read-only mode (e.g., offline mode), flexible response times.

Partitioning The system can tolerate network partitions.
Nuance: For non-mobile systems, partition are rare.
Alternatives: aim at CA & compensation after partition recovery.

There is a huge design space—CAP only covers a tiny part.

[2] E. Brewer, *CAP Twelve Years Later: How the "Rules" Have Changed*, 2012.

Distributed Databases

Definition[3]

Distributed database A collection of multiple, logically interrelated databases located at the nodes of a distributed system.

Distributed DBMS the software system that manages the distributed database & makes the distribution transparent to the users

[3] M.T. Özsu & P. Valduriez, *Principles of Distributed Database Systems*, 4th ed., 2020.

Resilient Systems

Definition

A resilient system is a system that can tolerate failures while continuously providing service.

Resilient Systems

Definition

A resilient system is a system that can tolerate failures while continuously providing service.

- *Failure*: hardware, software, network, malicious attacks, ...
  Crash fault tolerance versus Byzantine fault tolerance.

Resilient Systems

Definition

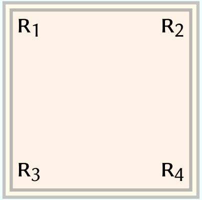
A resilient system is a system that can tolerate failures while continuously providing service.

- *Failure*: hardware, software, network, malicious attacks, ...
  Crash fault tolerance versus Byzantine fault tolerance.

- *Continuous services*: No downtime, manual intervention, restarts, ...

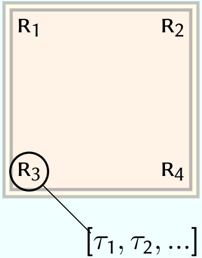
Outline of a Blockchain-Based Resilient System

1. Replicas.



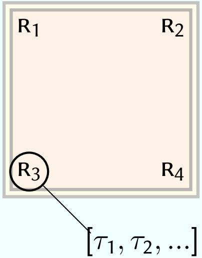
Outline of a Blockchain-Based Resilient System

1. Replicas.
2. Holding a *ledger* of transactions.



Outline of a Blockchain-Based Resilient System

1. Replicas.
2. Holding a *ledger* of transactions.

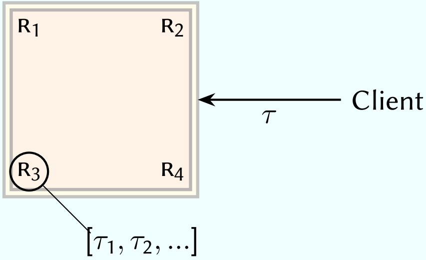


Assumption: Deterministic execution

The ledger provides *consistency*:
Good replicas execute the same operations in the same order and get the same results.

Outline of a Blockchain-Based Resilient System

1. Replicas.
2. Holding a *ledger* of transactions.
3. Clients request new transactions.

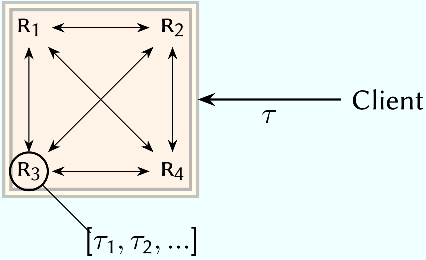


Assumption: Deterministic execution

The ledger provides *consistency*:
Good replicas execute the same operations in the same order and get the same results.

Outline of a Blockchain-Based Resilient System

1. Replicas.
2. Holding a *ledger* of transactions.
3. Clients request new transactions.
4. Transaction agreement via *consensus*.

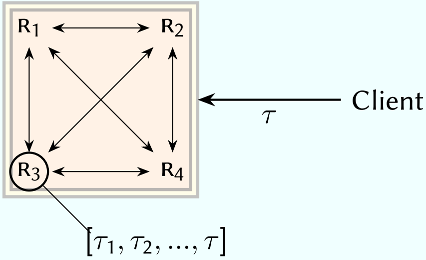


Assumption: Deterministic execution

The ledger provides *consistency*:
Good replicas execute the same operations in the same order and get the same results.

Outline of a Blockchain-Based Resilient System

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

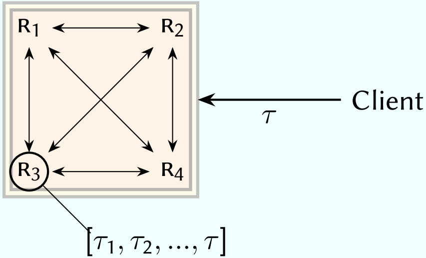


Assumption: Deterministic execution

The ledger provides *consistency*:
Good replicas execute the same operations in the same order and get the same results.

Outline of a Blockchain-Based Resilient System

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



Assumption: Deterministic execution

The ledger provides *consistency*:
Good replicas execute the same operations in the same order and get the same results.

Classifying Resilient Systems

The complexity of operating a resilient system depends on many factors:

Failures. In what ways can replicas fail?

Communication. What assumptions are made on communication?

Authentication. How are messages and their senders identified and verified?

Failure Models

We say that a replica is

initially crashed if it will never do anything;

crashed if it stops doing anything at some point;

omitting if it can omit coordination steps;

Byzantine if it can behave *arbitrary* (e.g., omitting steps, performing the wrong steps.).

Failure Models

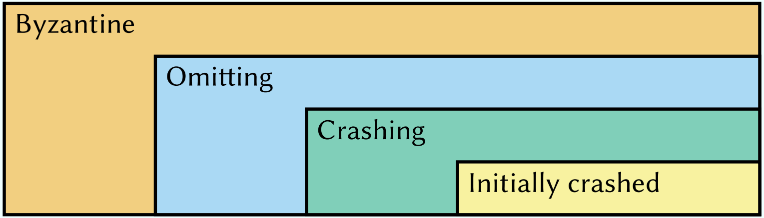
We say that a replica is

initially crashed if it will never do anything;

crashed if it stops doing anything at some point;

omitting if it can omit coordination steps;

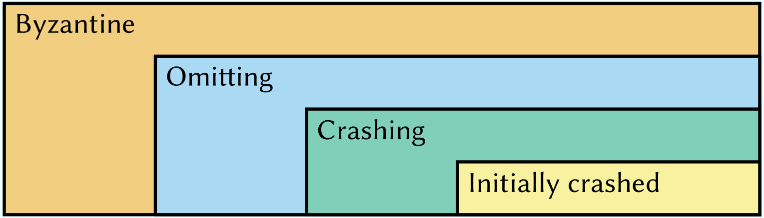
Byzantine if it can behave *arbitrary* (e.g., omitting steps, performing the wrong steps.).



Failure Models

We say that a replica is

- initially crashed — if it will never do anything;
- crashed — if it stops doing anything at some point;
- omitting — if it can omit coordination steps;
- Byzantine — if it can behave *arbitrary* (e.g., omitting steps, performing the wrong steps.).



Both *omitting* and *Byzantine* replicas can be **malicious**: these replicas can **coordinate** among themselves in attempts to disrupt the system.

Communication Models

We say that communication is

synchronous if every message sent will arrive only at its destination, will do so exactly once within some *known delay*. Communication can be modeled in *rounds*.

asynchronous if messages can be arbitrarily delayed, duplicated, or *dropped*. Arbitrary delay: message can arrive *out of order*.

Communication Models

We say that communication is

synchronous if every message sent will arrive only at its destination, will do so exactly once within some *known delay*. Communication can be modeled in *rounds*.

asynchronous if messages can be arbitrarily delayed, duplicated, or *dropped*. Arbitrary delay: message can arrive *out of order*.

- Synchronous: *assume* reliable communication, no partitions.

Communication Models

We say that communication is

synchronous if every message sent will arrive only at its destination, will do so exactly once within some *known delay*. Communication can be modeled in *rounds*.

asynchronous if messages can be arbitrarily delayed, duplicated, or *dropped*. Arbitrary delay: message can arrive *out of order*.

- Synchronous: *assume* reliable communication, no partitions. *unrealistic*

Communication Models

We say that communication is

synchronous
: if every message sent will arrive only at its destination, will do so exactly once within some *known delay*. Communication can be modeled in *rounds*.

asynchronous
: if messages can be arbitrarily delayed, duplicated, or *dropped*. Arbitrary delay: message can arrive *out of order*.

- Synchronous: *assume* reliable communication, no partitions. *unrealistic*
- Asynchronous: *wost-case* assumptions.

Communication Models

We say that communication is

- **synchronous**: if every message sent will arrive only at its destination, will do so exactly once within some *known delay*. Communication can be modeled in *rounds*.
- **asynchronous**: if messages can be arbitrarily delayed, duplicated, or *dropped*. Arbitrary delay: message can arrive *out of order*.

- Synchronous: *assume* reliable communication, no partitions. *unrealistic*
- Asynchronous: *wost-case* assumptions. *complex*

Byzantine Fault Tolerance: Authentication

Problem: Impersonation

Byzantine replicas can act arbitrarily and malicious:

they can try to impersonate many good replicas!

Byzantine Fault Tolerance: Authentication

Problem: Impersonation

Byzantine replicas can act arbitrarily and malicious:
they can try to impersonate many good replicas!

Solution: Authenticated communication

If replica Q determines that m was sent by replica R then:

- m must have been sent by R if R is good; and
- m must be sent by some faulty replica if R is faulty.
  Faulty replicas can only impersonate each other.

Byzantine Fault Tolerance: Authentication

Problem: Impersonation

Byzantine replicas can act arbitrarily and malicious:
they can try to impersonate many good replicas!

Solution: Authenticated communication

If replica Q determines that *m* was sent by replica R then:

- *m* must have been sent by R if R is good; and
- *m* must be sent by some faulty replica if R is faulty.
  Faulty replicas can only impersonate each other.

Implementation: *message authentication codes* (MACs).
(*cheap* symmetric-key cryptography).

Byzantine Fault Tolerance: Authentication

Problem: Impersonation

Byzantine replicas can act arbitrarily and malicious:
they can try to impersonate many good replicas!

Solution: Authenticated communication

If replica Q determines that *m* was sent by replica R then:

- *m* must have been sent by R if R is good; and
- *m* must be sent by some faulty replica if R is faulty.
  Faulty replicas can only impersonate each other.

Implementation: *message authentication codes* (MACs).
(*cheap* symmetric-key cryptography).

Second Problem: Message corruption

Faulty replicas can corrupt any *forwarded messages*.

Byzantine Fault Tolerance: Authentication

Problem: Impersonation

Byzantine replicas can act arbitrarily and malicious:
 they can try to impersonate many good replicas!

Second Problem: Message corruption

Faulty replicas can corrupt any *forwarded messages*.

Solution: Digital signatures

- Each replica R can *sign* any message m, yielding cert(m, R).
- A signed message cert(m, R) is non-forgeable without the help of R.
  Faulty replicas can only corrupt messages from each other when forwarding.

Byzantine Fault Tolerance: Authentication

Problem: Impersonation

Byzantine replicas can act arbitrarily and malicious:
they can try to impersonate many good replicas!

Second Problem: Message corruption

Faulty replicas can corrupt any *forwarded messages*.

Solution: Digital signatures

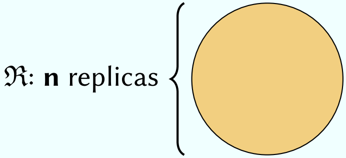
- Each replica R can *sign* any message m, yielding $\text{cert}(m, \textsc{r})$.
- A signed message $\text{cert}(m, \textsc{r})$ is non-forgeable without the help of R.
  Faulty replicas can only corrupt messages from each other when forwarding.

Implementation: *costly* public-key cryptography.

Notations for Resilient Systems

A resilient system is a *system* with

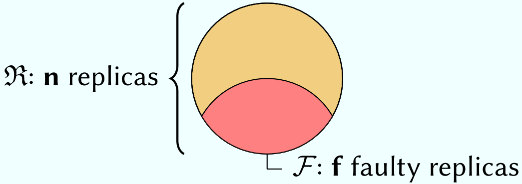
- A set of **n** *replicas* $\mathfrak{R}$.



Notations for Resilient Systems

A resilient system is a *system* with

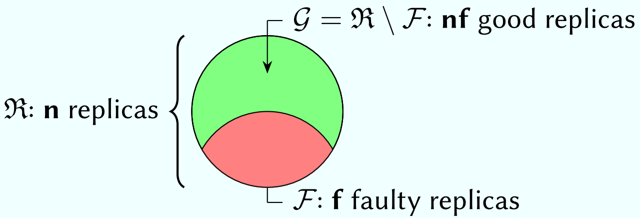
- A set of **n** *replicas* $\mathfrak{R}$.
- A set of **f** such replicas $\mathcal{F} \subseteq \mathfrak{R}$ are *faulty*.



Notations for Resilient Systems

A resilient system is a *system* with

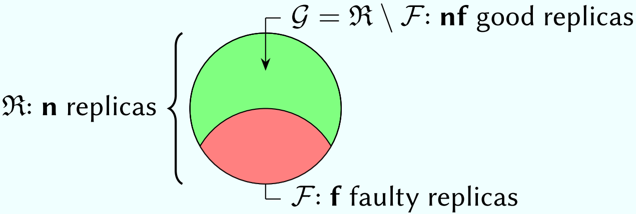
- A set of **n** *replicas* $\mathfrak{R}$.
- A set of **f** such replicas $\mathcal{F} \subseteq \mathfrak{R}$ are *faulty*.
- We have **nf** *good (non-faulty) replicas* $\mathcal{G} = \mathfrak{R} \setminus \mathcal{F}$.



Notations for Resilient Systems

A resilient system is a *system* with

- A set of $\mathbf{n}$ *replicas* $\mathfrak{R}$.
- A set of $\mathbf{f}$ such replicas $\mathcal{F} \subseteq \mathfrak{R}$ are *faulty*.
- We have $\mathbf{nf}$ *good (non-faulty) replicas* $\mathcal{G} = \mathfrak{R} \setminus \mathcal{F}$.
- Each replica $\textsc{r} \in \mathfrak{R}$ has a *unique identifier* $0 \leq \text{id}(\textsc{r}) < \mathbf{n}$.

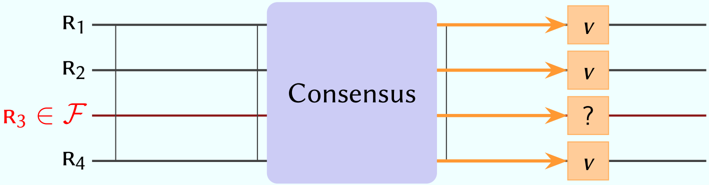


Coordination in Resilient Systems: Consensus

A protocol provides *consensus* if upon completion of the protocol:

Termination Eventually, each good replica $R \in \mathcal{G}$ decides on a value $v(R)$.

Non-divergence All good replicas decide on the same value.
Hence, if $R_1, R_2 \in \mathcal{G}$ decide $v(R_1)$ and $v(R_2)$, then $v(R_1) = v(R_2)$.



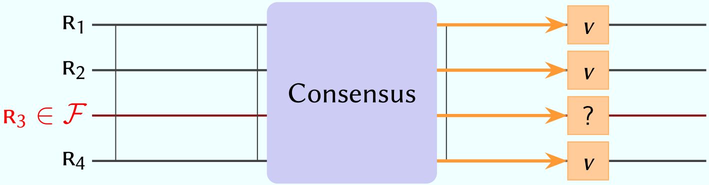
Coordination in Resilient Systems: Consensus

A protocol provides *consensus* if upon completion of the protocol:

Termination Eventually, each good replica $\text{R} \in \mathcal{G}$ decides on a value $v(\text{R})$.

Non-divergence All good replicas decide on the same value.
Hence, if $\text{R}_1, \text{R}_2 \in \mathcal{G}$ decide $v(\text{R}_1)$ and $v(\text{R}_2)$, then $v(\text{R}_1) = v(\text{R}_2)$.

Non-triviality In different runs of the protocol, replicas can decide on different values.

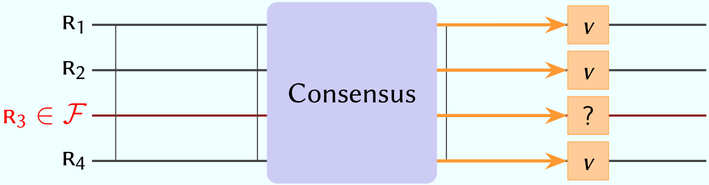


Excludes trivial solutions: e.g., good replicas always deciding a pre-defined value.

Coordination in Resilient Systems: Consensus

A protocol provides *consensus* if upon completion of the protocol:

Termination	Eventually, each good replica $\text{R} \in \mathcal{G}$ decides on a value $v(\text{R})$.	*Liveness*
Non-divergence	All good replicas decide on the same value. Hence, if $\text{R}_1, \text{R}_2 \in \mathcal{G}$ decide $v(\text{R}_1)$ and $v(\text{R}_2)$, then $v(\text{R}_1) = v(\text{R}_2)$.	*Safety*
Non-triviality	In different runs of the protocol, replicas can decide on different values.	



Excludes trivial solutions: e.g., good replicas always deciding a pre-defined value.

Consensus: From Formalization to Practice

Formal consensus is *abstract*: Where do the decided values come from?

Consensus: From Formalization to Practice

Formal consensus is *abstract*: Where do the decided values come from?

Towards consensus in practice

Expand *non-triviality* by putting application-specific requirements on decided values.

Consensus: From Formalization to Practice

Formal consensus is *abstract*: Where do the decided values come from?

Towards consensus in practice

Expand *non-triviality* by putting application-specific requirements on decided values.

Example: System processing client transactions

Decided-upon values that are not-yet executed client-requested transactions.

Coordination in Resilient Systems: Interactive Consistency

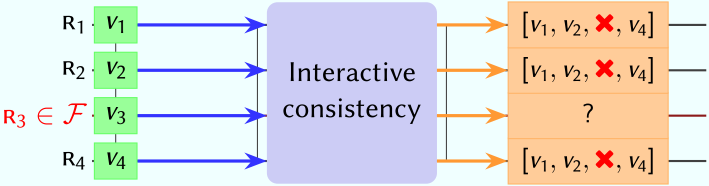
Assumption: Each replica R holds an initial value $v(\text{R})$

A protocol provides *interactive consistency* if upon completion of the protocol:

Termination Eventually, each good replica $\text{R} \in \mathcal{G}$ decides on a list $L(\text{R})$ of $\mathbf{n}$ values.

Non-divergence All good replicas decide on the same list.
Hence, if $\text{R}_1, \text{R}_2 \in \mathcal{G}$ decide $L(\text{R}_1)$ and $L(\text{R}_2)$, then $L(\text{R}_1) = L(\text{R}_2)$.

Dependence Let $\text{R} \in \mathcal{G}$. Good replicas will have $v(\text{R})$ as the $\text{id}(\text{R})$-th list value.
Hence, if replica $\text{Q} \in \mathcal{G}$ decided on $L(\text{Q})$, then $L(\text{Q})[\text{id}(\text{R})] = v(\text{R})$.



From Interactive Consistency to Consensus

Example: System processing client transactions

1. Each $\textsc{r} \in \mathcal{G}$ chooses as $v(\textsc{r})$ a new client transaction.

From Interactive Consistency to Consensus

Example: System processing client transactions

1. Each $\text{R} \in \mathcal{G}$ chooses as $v(\text{R})$ a new client transaction.
2. All replicas participate in *interactive consistency*.

From Interactive Consistency to Consensus

Example: System processing client transactions

1. Each $\textsc{r} \in \mathcal{G}$ chooses as $v(\textsc{r})$ a new client transaction.
2. All replicas participate in *interactive consistency*.

   Each good replica obtains the same list L of $\mathbf{n}$ values.

From Interactive Consistency to Consensus

Example: System processing client transactions

1. Each $\textsc{r} \in \mathcal{G}$ chooses as $v(\textsc{r})$ a new client transaction.
2. All replicas participate in *interactive consistency*.

   Each good replica obtains the same list L of $\mathbf{n}$ values.

3. At least $\mathbf{nf} = \mathbf{n} - \mathbf{f}$ values in L are *valid new client transactions*.

From Interactive Consistency to Consensus

Example: System processing client transactions

1. Each $\textsc{r} \in \mathcal{G}$ chooses as $v(\textsc{r})$ a new client transaction.
2. All replicas participate in *interactive consistency*.

   Each good replica obtains the same list L of $\mathbf{n}$ values.

3. At least $\mathbf{nf} = \mathbf{n} - \mathbf{f}$ values in L are *valid new client transactions*.
4. Good replicas use a *deterministic method* to choose a valid transaction from L. E.g.,
   - the first valid transaction in L; or
   - all valid transactions in L (optimization).

Coordination in Resilient Systems: Byzantine Broadcast

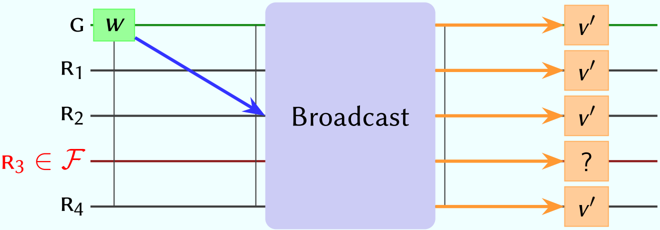
Assumption: Some replica G holds an initial value w

A protocol provides *byzantine broadcast* if upon completion of the protocol:

Termination Eventually, each good replica $R \in \mathcal{G}$ decides on a value $v(R)$.

Non-divergence All good replicas decide on the same value.
Hence, if $R_1, R_2 \in \mathcal{G}$ decide $v(R_1)$ and $v(R_2)$, then $v(R_1) = v(R_2)$.

Dependence If G is good, then good replicas decide w.
Hence, if $G \in \mathcal{G}$ and $Q \in \mathcal{G}$, then $v(Q) = w$.



($v' = w$ if $G \in \mathcal{G}$)

From Byzantine Broadcasts to Interactive Consistency

Assumption: Each replica R holds an initial value $v(\text{R})$

1. Each replica R performs Byzantine broadcast of $v(\text{R})$.

From Byzantine Broadcasts to Interactive Consistency

Assumption: Each replica R holds an initial value $v(\text{R})$

1. Each replica R performs Byzantine broadcast of $v(\text{R})$.
2. What about faulty replicas? They might not broadcast!

From Byzantine Broadcasts to Interactive Consistency

Assumption: Each replica R holds an initial value $v(\text{R})$

1. Each replica R performs Byzantine broadcast of $v(\text{R})$.
2. What about faulty replicas? They might not broadcast!
3. Each good replica $\text{Q} \in \mathfrak{R}$ constructs $L(\text{Q}) = [v_0, \dots, v_{\mathbf{n}-1}]$ in which v_i, $0 \leq i < \mathbf{n}$, is the value broadcasted by the i-th replica.

From Byzantine Broadcasts to Interactive Consistency

Assumption: Each replica $\textsc{r}$ holds an initial value $v(\textsc{r})$

1. Each replica $\textsc{r}$ performs Byzantine broadcast of $v(\textsc{r})$.
2. What about faulty replicas? They might not broadcast!
3. Each good replica $\textsc{q} \in \mathfrak{R}$ constructs $L(\textsc{q}) = [v_0, \ldots, v_{\mathbf{n}-1}]$
   in which v_i, $0 \leq i < \mathbf{n}$, is the value broadcasted by the i-th replica.

Interactive consistency and Byzantine broadcasts solve the same problem.

Both can be used to provide practical consensus.

On the Complexity of Consensus

Consider a system with **n** replicas of which **f** are faulty.

The consensus problem (informal)

Can the good replicas reach *agreement* on a value?

	Initially Crashed	Crashed	Omitting	Byzantine	Byzantine (with MACs)	Byzantine (with DSs)
Asynchronous						
Synchronous						

On the Complexity of Consensus

Consider a system with **n** replicas of which **f** are faulty.

The consensus problem (informal)

Can the good replicas reach *agreement* on a value?

	Initially Crashed	Crashed	Omitting	Byzantine	Byzantine (with MACs)	Byzantine (with DSs)
Asynchronous						
Synchronous	$\mathbf{n} > \mathbf{f}$					

On the Complexity of Consensus

Consider a system with **n** replicas of which **f** are faulty.

The consensus problem (informal)

Can the good replicas reach *agreement* on a value?

	Initially Crashed	Crashed	Omitting	Byzantine	Byzantine (with MACs)	Byzantine (with DSs)
Asynchronous	$\mathbf{n} > 2\mathbf{f}$					
Synchronous	$\mathbf{n} > \mathbf{f}$					

On the Complexity of Consensus

Consider a system with **n** replicas of which **f** are faulty.

The consensus problem (informal)

Can the good replicas reach *agreement* on a value?

	Initially Crashed	Crashed	Omitting	Byzantine	Byzantine (with MACs)	Byzantine (with DSs)
Asynchronous	$\mathbf{n} > 2\mathbf{f}$					
Synchronous	$\mathbf{n} > \mathbf{f}$	$\mathbf{n} > \mathbf{f}$				

On the Complexity of Consensus

Consider a system with **n** replicas of which **f** are faulty.

The consensus problem (informal)

Can the good replicas reach *agreement* on a value?

	Initially Crashed	Crashed	Omitting	Byzantine	Byzantine (with MACs)	Byzantine (with DSs)
Asynchronous	$\mathbf{n} > 2\mathbf{f}$	Impossible[4]				
Synchronous	$\mathbf{n} > \mathbf{f}$	$\mathbf{n} > \mathbf{f}$				

[4] The FLP Impossibility Theorem.

On the Complexity of Consensus

Consider a system with **n** replicas of which **f** are faulty.

The consensus problem (informal)

Can the good replicas reach *agreement* on a value?

	Initially Crashed	Crashed	Omitting	Byzantine	Byzantine (with MACs)	Byzantine (with DSs)
Asynchronous	$\mathbf{n} > 2\mathbf{f}$	Impossible[4]	Impossible	Impossible	Impossible	Impossible
Synchronous	$\mathbf{n} > \mathbf{f}$	$\mathbf{n} > \mathbf{f}$				

[4] The FLP Impossibility Theorem.

On the Complexity of Consensus

Consider a system with **n** replicas of which **f** are faulty.

The consensus problem (informal)

Can the good replicas reach *agreement* on a value?

	Initially Crashed	Crashed	Omitting	Byzantine	Byzantine (with MACs)	Byzantine (with DSs)
Asynchronous	$\mathbf{n} > 2\mathbf{f}$	Impossible[4]	Impossible	Impossible	Impossible	Impossible
Synchronous	$\mathbf{n} > \mathbf{f}$	$\mathbf{n} > \mathbf{f}$	$\mathbf{n} > \mathbf{f}$			

[4] The FLP Impossibility Theorem.

On the Complexity of Consensus

Consider a system with **n** replicas of which **f** are faulty.

The consensus problem (informal)

Can the good replicas reach *agreement* on a value?

	Initially Crashed	Crashed	Omitting	Byzantine	Byzantine (with MACs)	Byzantine (with DSs)
Asynchronous	$\mathbf{n} > 2\mathbf{f}$	Impossible[4]	Impossible	Impossible	Impossible	Impossible
Synchronous	$\mathbf{n} > \mathbf{f}$	$\mathbf{n} > \mathbf{f}$	$\mathbf{n} > \mathbf{f}$	Impossible		

[4] The FLP Impossibility Theorem.

On the Complexity of Consensus

Consider a system with **n** replicas of which **f** are faulty.

The consensus problem (informal)

Can the good replicas reach *agreement* on a value?

	Initially Crashed	Crashed	Omitting	Byzantine	Byzantine (with MACs)	Byzantine (with DSs)
Asynchronous	$\mathbf{n} > 2\mathbf{f}$	Impossible[4]	Impossible	Impossible	Impossible	Impossible
Synchronous	$\mathbf{n} > \mathbf{f}$	$\mathbf{n} > \mathbf{f}$	$\mathbf{n} > \mathbf{f}$	Impossible	$\mathbf{n} > 3\mathbf{f}$	

[4] The FLP Impossibility Theorem.

On the Complexity of Consensus

Consider a system with **n** replicas of which **f** are faulty.

The consensus problem (informal)

Can the good replicas reach *agreement* on a value?

	Initially Crashed	Crashed	Omitting	Byzantine	Byzantine (with MACs)	Byzantine (with DSs)
Asynchronous	$\mathbf{n} > 2\mathbf{f}$	Impossible[4]	Impossible	Impossible	Impossible	Impossible
Synchronous	$\mathbf{n} > \mathbf{f}$	$\mathbf{n} > \mathbf{f}$	$\mathbf{n} > \mathbf{f}$	Impossible	$\mathbf{n} > 3\mathbf{f}$	$\mathbf{n} > \mathbf{f}$

[4] The FLP Impossibility Theorem.

On the Complexity of Consensus

Consider a system with **n** replicas of which **f** are faulty.

The consensus problem (informal)

Can the good replicas reach *agreement* on a value?

	Initially Crashed	Crashed	Omitting	Byzantine	Byzantine (with MACs)	Byzantine (with DSs)
Asynchronous	$\mathbf{n} > 2\mathbf{f}$	Impossible[4]	Impossible	Impossible	Impossible	Impossible
Synchronous	$\mathbf{n} > \mathbf{f}$	$\mathbf{n} > \mathbf{f}$	$\mathbf{n} > \mathbf{f}$	Impossible	$\mathbf{n} > 3\mathbf{f}$	~~$\mathbf{n} > \mathbf{f}$~~ $\mathbf{n} > 2\mathbf{f}$

[4] The FLP Impossibility Theorem.

Consensus in Practice: Impossible?

Theorem (FLP Impossibility Theorem)

The consensus problem cannot be solved for systems that

- *uses asynchronous communication; and*
- *allows faulty replicas to crash.*

Real-world networks: Asynchronous, but mostly well-behaved

Consensus in Practice: Impossible?

Theorem (FLP Impossibility Theorem)

The consensus problem cannot be solved for systems that

- *uses asynchronous communication; and*
- *allows faulty replicas to crash.*

Real-world networks: Asynchronous, but mostly well-behaved

Do not solve consensus: weaken the *termination* guarantee:

Each good replica $\textsc{r} \in \mathcal{G}$ decides on a value $v(\textsc{r})$.

Consensus in Practice: Impossible?

Theorem (FLP Impossibility Theorem)

The consensus problem cannot be solved for systems that

- *uses asynchronous communication; and*
- *allows faulty replicas to crash.*

Real-world networks: Asynchronous, but mostly well-behaved

Do not solve consensus: weaken the *termination* guarantee:

$$\text{Each good replica } \textsc{r} \in \mathcal{G} \text{ decides on a value } v(\textsc{r}).$$

Weak termination Good replicas decide *when communication is well-behaved*.

Probabilistic termination Good replicas decide *with high probability*.

On the Complexity of Consensus in Practice

Consider a system with **n** replicas of which **f** are faulty.

The consensus problem (informal)

Can the good replicas reach *agreement* on a value?

	Initially Crashed	Crashed	Omitting	Byzantine	Byzantine (with MACs)	Byzantine (with DSs)
Asynchronous	$\mathbf{n} > 2\mathbf{f}$			Impossible		
Synchronous	$\mathbf{n} > \mathbf{f}$	$\mathbf{n} > \mathbf{f}$	$\mathbf{n} > \mathbf{f}$	Impossible	$\mathbf{n} > 3\mathbf{f}$	~~$\mathbf{n} > \mathbf{f}$~~ $\mathbf{n} > 2\mathbf{f}$

On the Complexity of Consensus in Practice

Consider a system with **n** replicas of which **f** are faulty.

The consensus problem (informal)

Can the good replicas reach *agreement* on a value?

	Initially Crashed	Crashed	Omitting	Byzantine	Byzantine (with MACs)	Byzantine (with DSs)
Asynchronous	$\mathbf{n} > 2\mathbf{f}$	$\mathbf{n} > 2\mathbf{f}$ [5]		Impossible		
Synchronous	$\mathbf{n} > \mathbf{f}$	$\mathbf{n} > \mathbf{f}$	$\mathbf{n} > \mathbf{f}$	Impossible	$\mathbf{n} > 3\mathbf{f}$	~~$\mathbf{n} > \mathbf{f}$~~ $\mathbf{n} > 2\mathbf{f}$

[5] Weak or probabilistic termination.

On the Complexity of Consensus in Practice

Consider a system with **n** replicas of which **f** are faulty.

The consensus problem (informal)

Can the good replicas reach *agreement* on a value?

	Initially Crashed	Crashed	Omitting	Byzantine	Byzantine (with MACs)	Byzantine (with DSs)
Asynchronous	$\mathbf{n} > 2\mathbf{f}$	$\mathbf{n} > 2\mathbf{f}$ [5]	$\mathbf{n} > 2\mathbf{f}$ [5]	Impossible		
Synchronous	$\mathbf{n} > \mathbf{f}$	$\mathbf{n} > \mathbf{f}$	$\mathbf{n} > \mathbf{f}$	Impossible	$\mathbf{n} > 3\mathbf{f}$	~~$\mathbf{n} > \mathbf{f}$~~ $\mathbf{n} > 2\mathbf{f}$

[5] Weak or probabilistic termination.

On the Complexity of Consensus in Practice

Consider a system with **n** replicas of which **f** are faulty.

The consensus problem (informal)

Can the good replicas reach *agreement* on a value?

	Initially Crashed	Crashed	Omitting	Byzantine	Byzantine (with MACs)	Byzantine (with DSs)
Asynchronous	$\mathbf{n} > 2\mathbf{f}$	$\mathbf{n} > 2\mathbf{f}$ [5]	$\mathbf{n} > 2\mathbf{f}$ [5]	Impossible	$\mathbf{n} > 3\mathbf{f}$ [5]	
Synchronous	$\mathbf{n} > \mathbf{f}$	$\mathbf{n} > \mathbf{f}$	$\mathbf{n} > \mathbf{f}$	Impossible	$\mathbf{n} > 3\mathbf{f}$	~~$\mathbf{n} > \mathbf{f}$~~ $\mathbf{n} > 2\mathbf{f}$

[5] Weak or probabilistic termination.

On the Complexity of Consensus in Practice

Consider a system with **n** replicas of which **f** are faulty.

The consensus problem (informal)

Can the good replicas reach *agreement* on a value?

	Initially Crashed	Crashed	Omitting	Byzantine	Byzantine (with MACs)	Byzantine (with DSs)
Asynchronous	$\mathbf{n} > 2\mathbf{f}$	$\mathbf{n} > 2\mathbf{f}$ [5]	$\mathbf{n} > 2\mathbf{f}$ [5]	Impossible	$\mathbf{n} > 3\mathbf{f}$ [5]	$\mathbf{n} > 3\mathbf{f}$ [5]
Synchronous	$\mathbf{n} > \mathbf{f}$	$\mathbf{n} > \mathbf{f}$	$\mathbf{n} > \mathbf{f}$	Impossible	$\mathbf{n} > 3\mathbf{f}$	~~$\mathbf{n} > \mathbf{f}$~~ $\mathbf{n} > 2\mathbf{f}$

[5] Weak or probabilistic termination.

The Complexity and Cost of Consensus

Many other limitations on consensus are known

Communication phases	a worst-case approach in at-least $\mathbf{f} + 1$ phases; at-least $t + 2$ phases if $t \leq \mathbf{f}$ replicas behave faulty (optimistic);
Communication cost	an exchange of at-least $\mathbf{nf}$ signatures; an exchange of at least $\mathbf{n} + \mathbf{f}^2$ messages;
Network structure	at-least $2\mathbf{f} + 1$ disjoint communication paths between every pair of replicas.