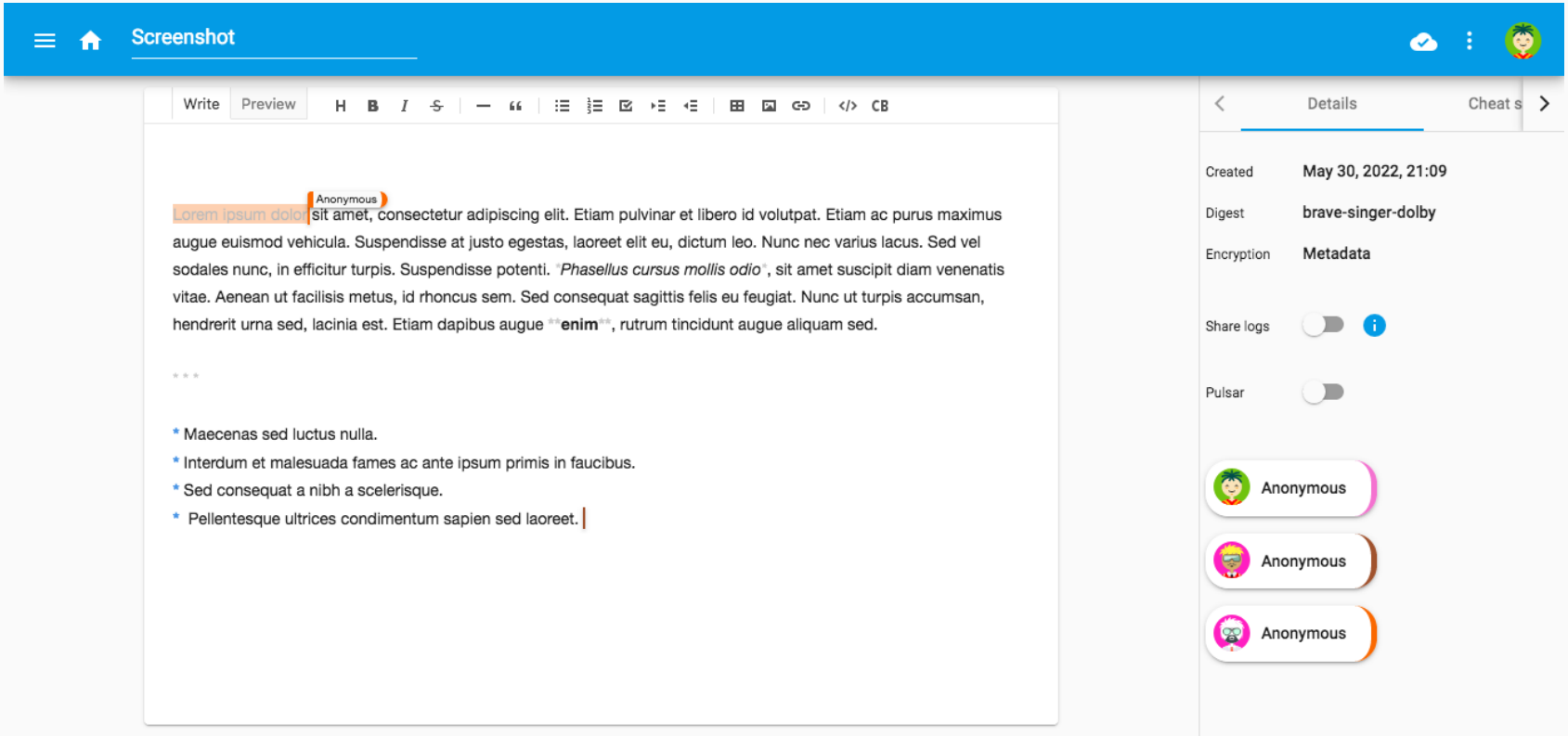
Conflict-free Replicated Data Types (CRDTs)

An Overview

Matthieu Nicolas (matthieu.nicolas@inria.fr)

02/05/2024

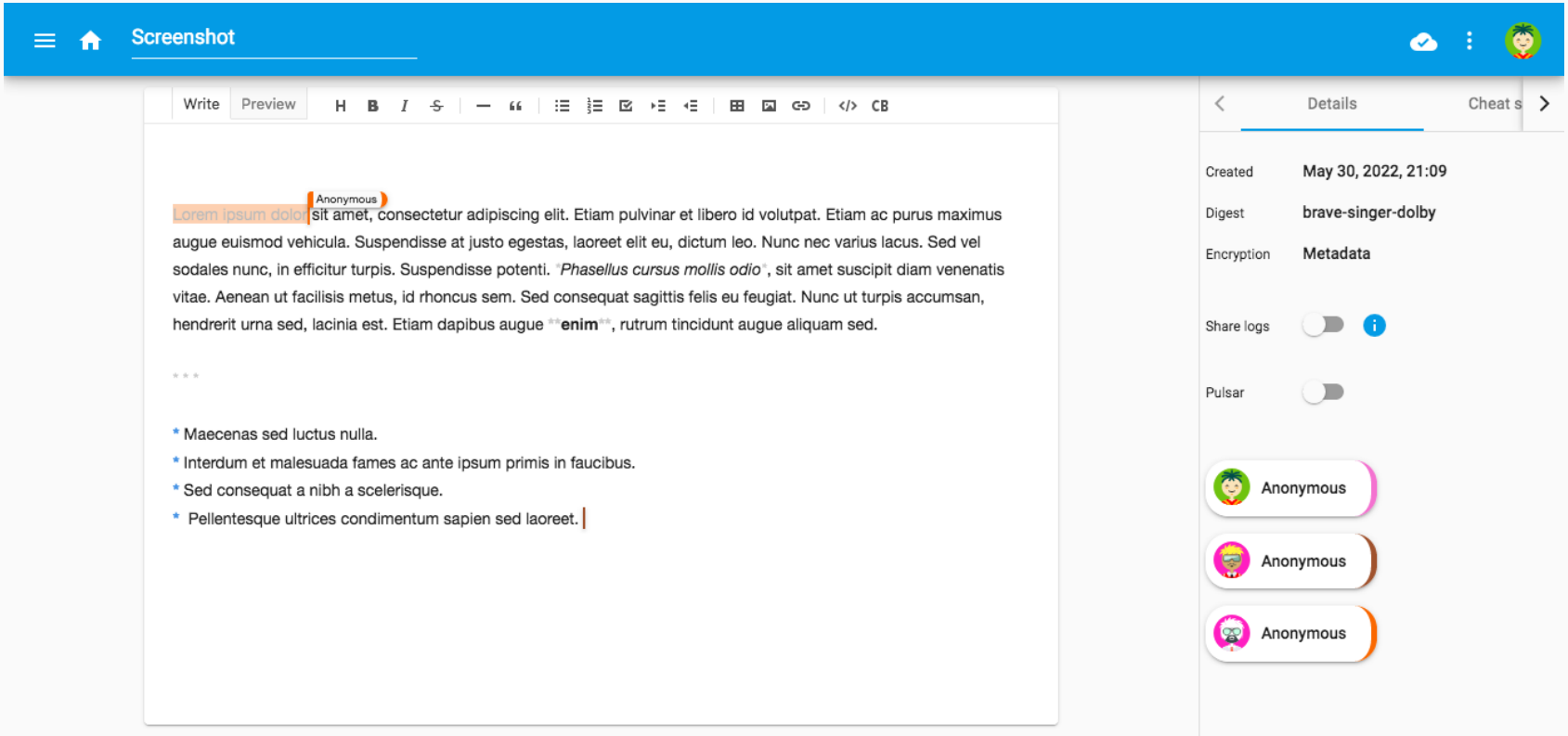
MUTE [Nic+17]



Local First Software [Kle+19]

- Peer-to-Peer (P2P) collaborative applications

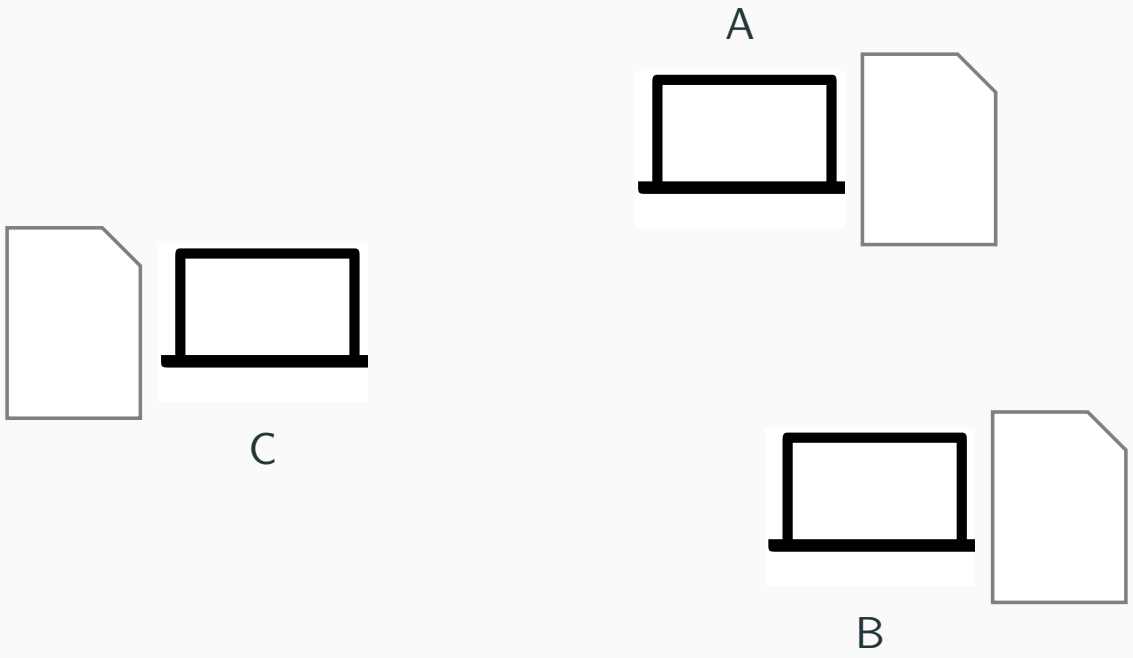
MUTE [Nic+17]



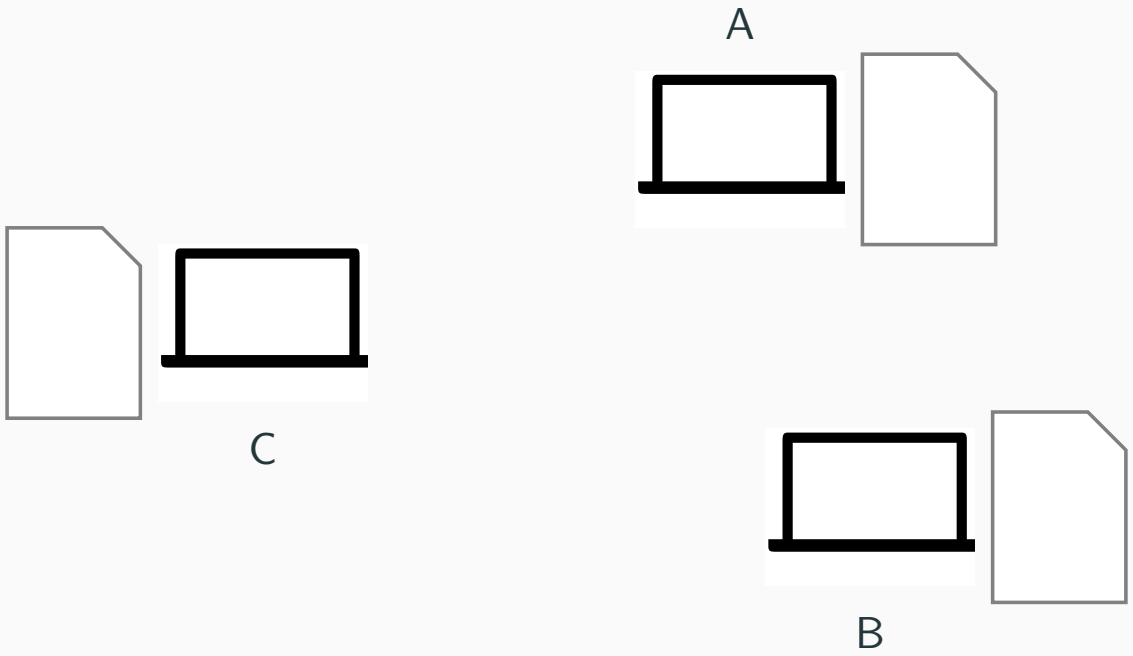
Local First Software [Kle+19]

- Peer-to-Peer (P2P) collaborative applications
- Always available
- Ensure ownership and privacy of data

Data replication in P2P systems

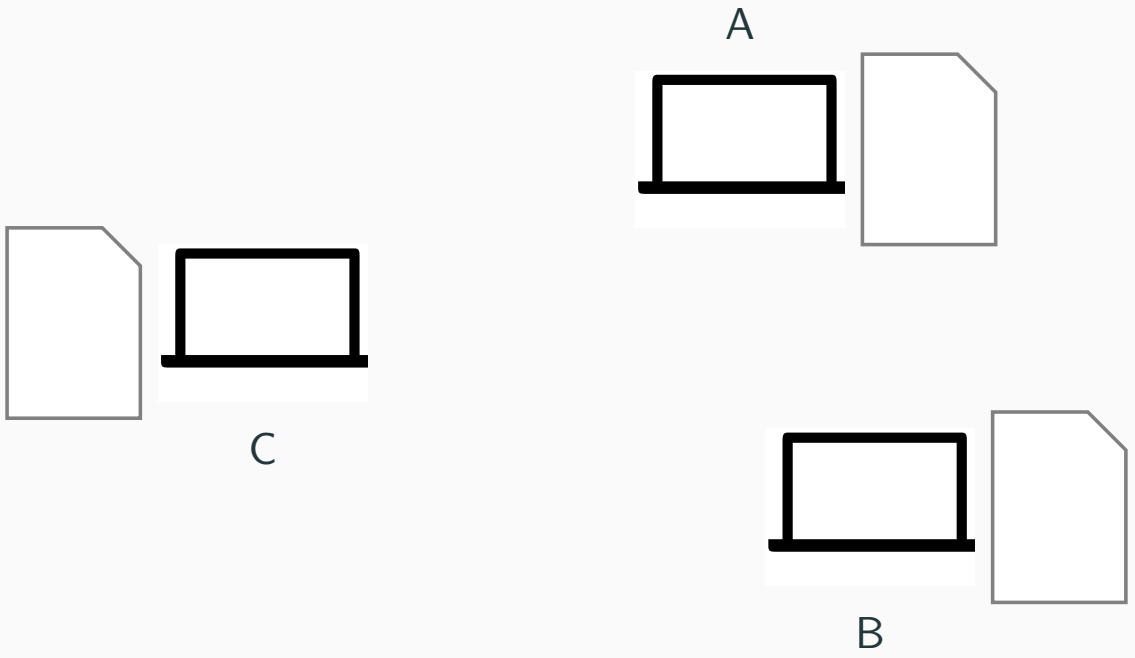


Data replication in P2P systems



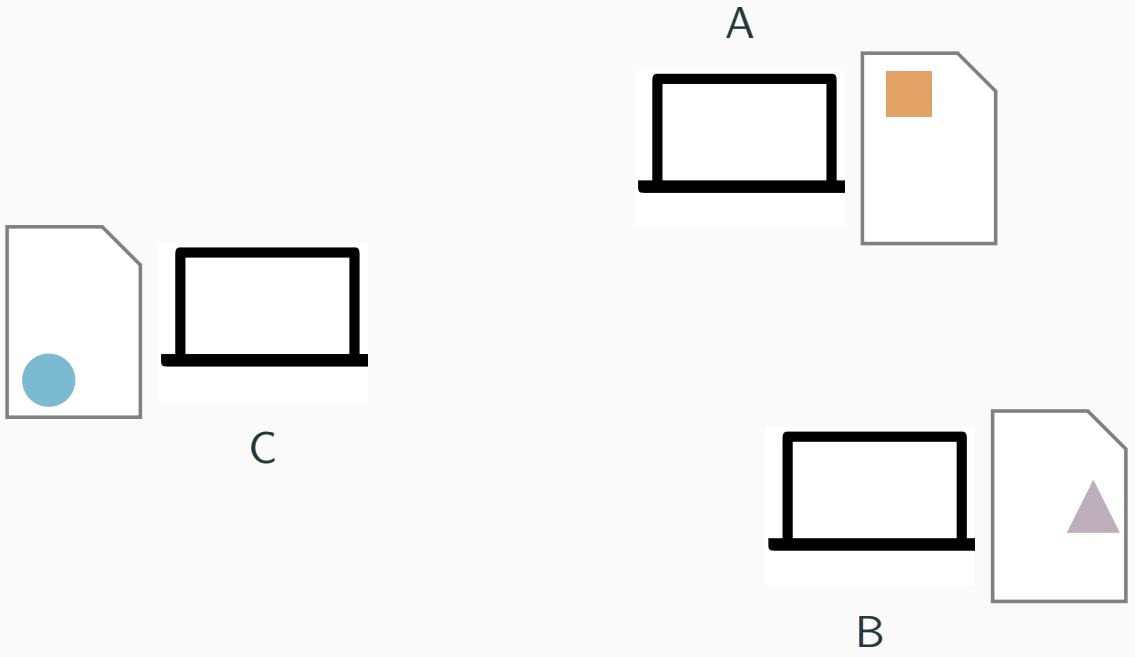
- Nodes may be disconnected

Data replication in P2P systems



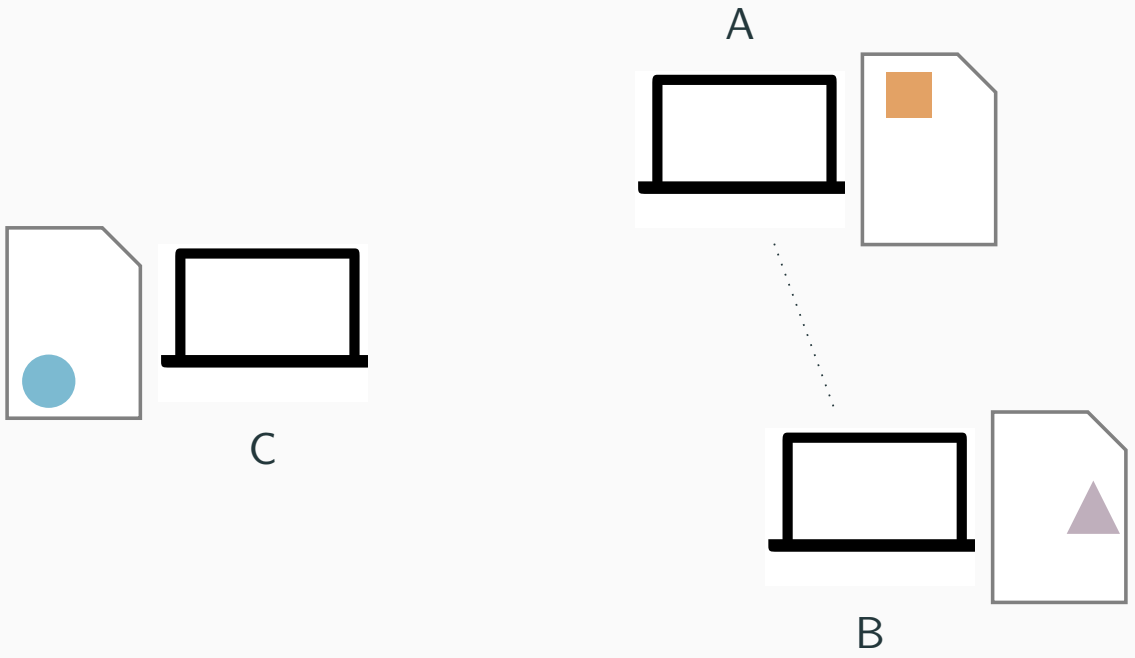
- Nodes may be disconnected
- Enable nodes to work without prior or current synchronous coordination (i.e. consensus)

Data replication in P2P systems



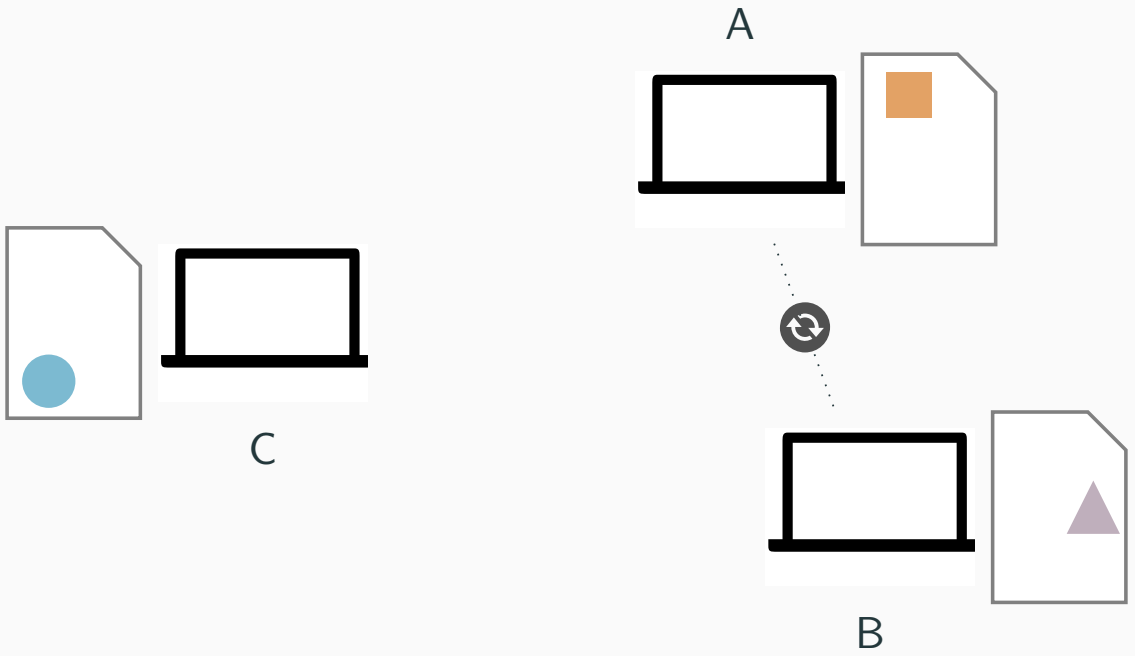
- Nodes may be disconnected
- Enable nodes to work without prior or current synchronous coordination (i.e. consensus)

Data replication in P2P systems



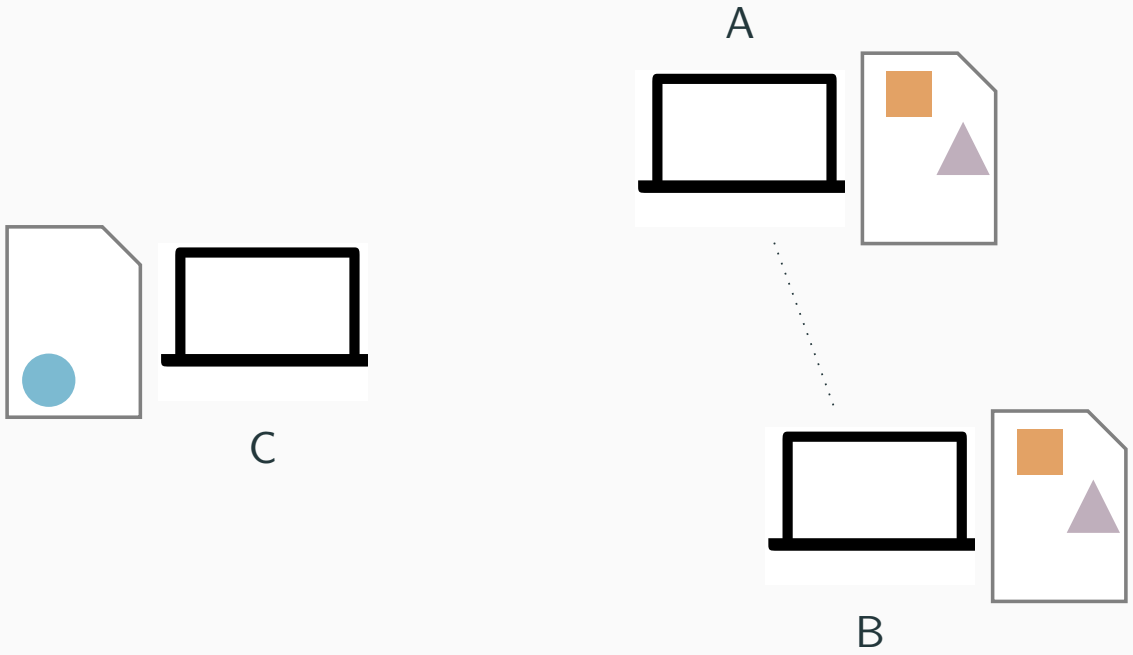
- Nodes may be disconnected
- Enable nodes to work without prior or current synchronous coordination (i.e. consensus)

Data replication in P2P systems



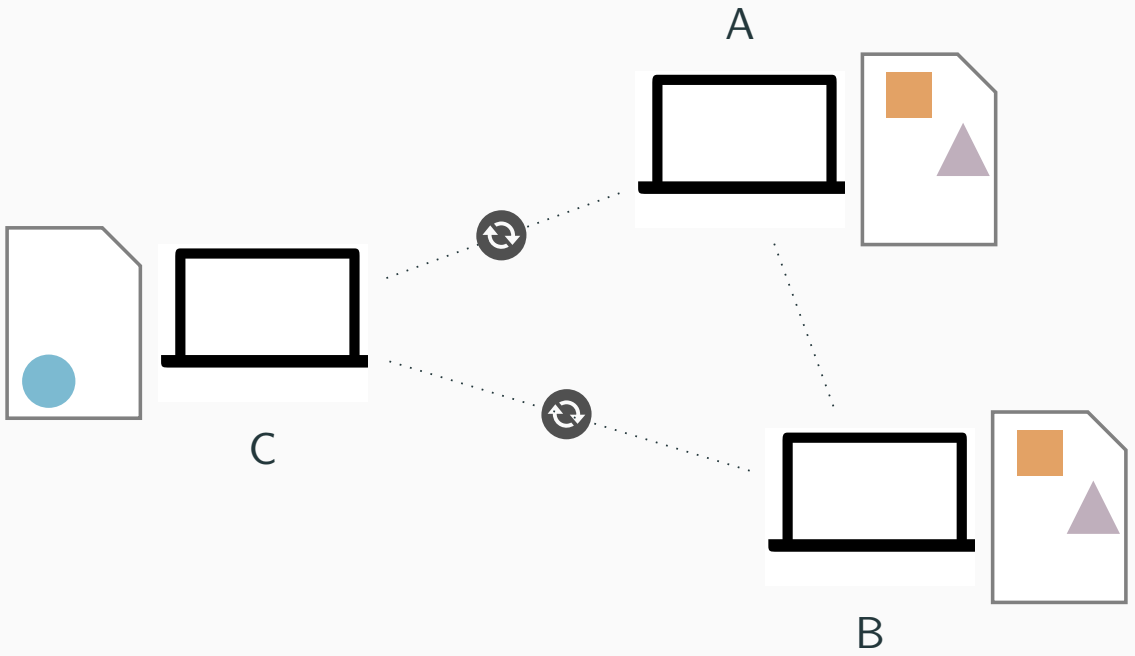
- Nodes may be disconnected
- Enable nodes to work without prior or current synchronous coordination (i.e. consensus)

Data replication in P2P systems



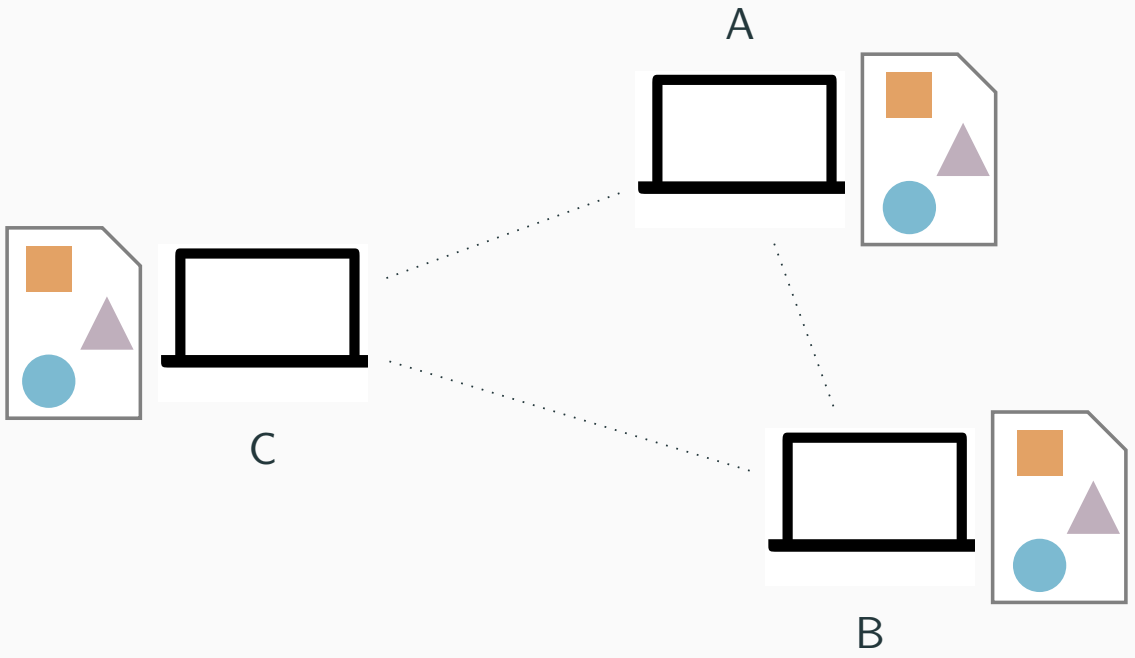
- Nodes may be disconnected
- Enable nodes to work without prior or current synchronous coordination (i.e. consensus)

Data replication in P2P systems



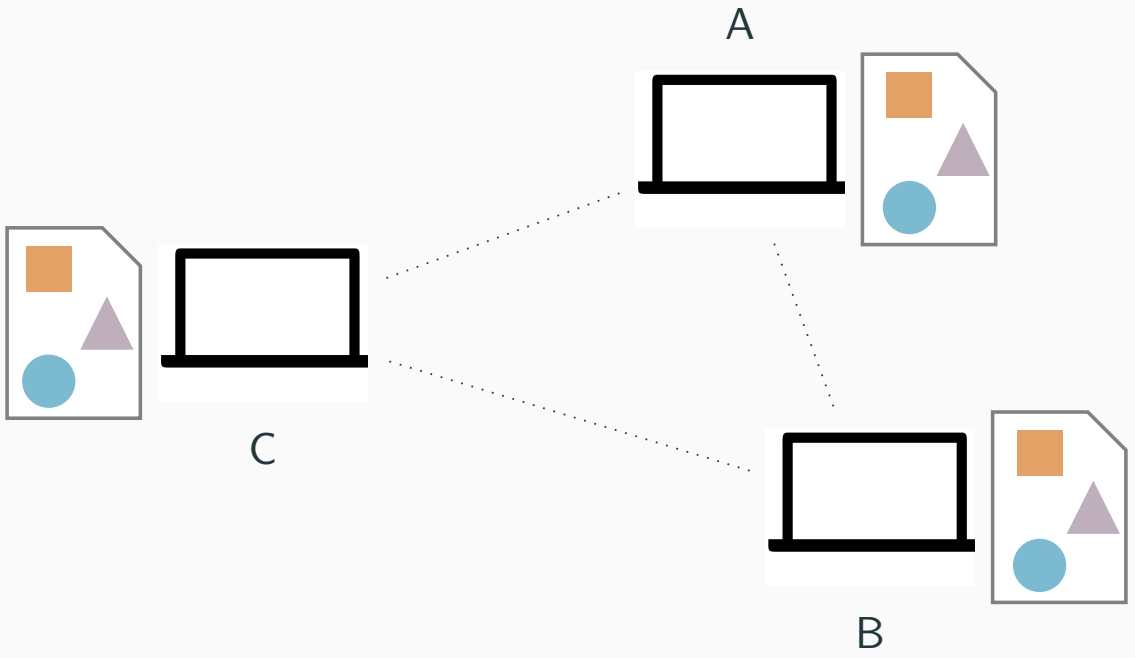
- Nodes may be disconnected
- Enable nodes to work without prior or current synchronous coordination (i.e. consensus)

Data replication in P2P systems



- Nodes may be disconnected
- Enable nodes to work without prior or current synchronous coordination (i.e. consensus)
- Must ensure Eventual Consistency [Ter+95]...
- ...Despite different integration orders of updates

Data replication in P2P systems



- Nodes may be disconnected
- Enable nodes to work without prior or current synchronous coordination (i.e. consensus)
- Must ensure Eventual Consistency [Ter+95]...
- ...Despite different integration orders of updates

Require *conflict resolution mechanisms*

Conflict-free Replicated Data Types (CRDTs)
are a family of conflict resolution mechanisms
for P2P systems

Conflict-free Replicated Data Types (CRDTs) [Sha+11]

- New specifications of existing Data Types, e.g. *Set* or *Sequence*
- Embed natively conflict resolution mechanisms

Conflict-free Replicated Data Types (CRDTs) [Sha+11]

- New specifications of existing Data Types, e.g. *Set* or *Sequence*
- Embed natively conflict resolution mechanisms

Properties of CRDTs

- Enable modifications without coordination
- Ensure Strong Eventual Consistency

Conflict-free Replicated Data Types (CRDTs) [Sha+11]

- New specifications of existing Data Types, e.g. *Set* or *Sequence*
- Embed natively conflict resolution mechanisms

Properties of CRDTs

- Enable modifications without coordination
- Ensure Strong Eventual Consistency

Strong Eventual Consistency

Nodes that integrate the same set of updates reach equivalent states, without additional actions or messages

Conflict-free Replicated Data Types (CRDTs) [Sha+11]

- New specifications of existing Data Types, e.g. *Set* or *Sequence*
- Embed natively conflict resolution mechanisms

Properties of CRDTs

- Enable modifications without coordination
- Ensure Strong Eventual Consistency

Strong Eventual Consistency

Nodes that integrate the same set of updates reach equivalent states, without additional actions or messages

- Rely on the lattice theory ...
- ... More specifically, CRDTs are join-semilattices

Lattice theory for CRDTs

Properties of join-semilattices

Lattice theory for CRDTs

Properties of join-semilattices

- States of the join-semilattice are partially ordered according to relation $\leq$

Lattice theory for CRDTs

Properties of join-semilattices

- States of the join-semilattice are partially ordered according to relation $\leq$
- Updates produce new states by inflation, i.e. greater to previous ones according to $\leq$

Lattice theory for CRDTs

Properties of join-semilattices

- States of the join-semilattice are partially ordered according to relation $\leq$
- Updates produce new states by inflation, i.e. greater to previous ones according to $\leq$
- Exists a function *join* that, given any pair of states, generates the minimal state greater or equal to both given states according to $\leq$

Lattice theory for CRDTs

Properties of join-semilattices

- States of the join-semilattice are partially ordered according to relation $\leq$
- Updates produce new states by inflation, i.e. greater to previous ones according to $\leq$
- Exists a function *join* that, given any pair of states, generates the minimal state greater or equal to both given states according to $\leq$
- Exists a set of minimal states of the join-semilattice, the *irreducible elements*

Design of CRDTs

- Several CRDTs may be designed for a given data type . . .
- . . . Each offering different trade-offs

Design of CRDTs

- Several CRDTs may be designed for a given data type ...
- ... Each offering different trade-offs

What impact the design of a given CRDT [Pre18]

- Conflict Resolution Semantics
- Synchronisation Model

Design of CRDTs

- Several CRDTs may be designed for a given data type . . .
- . . . Each offering different trade-offs

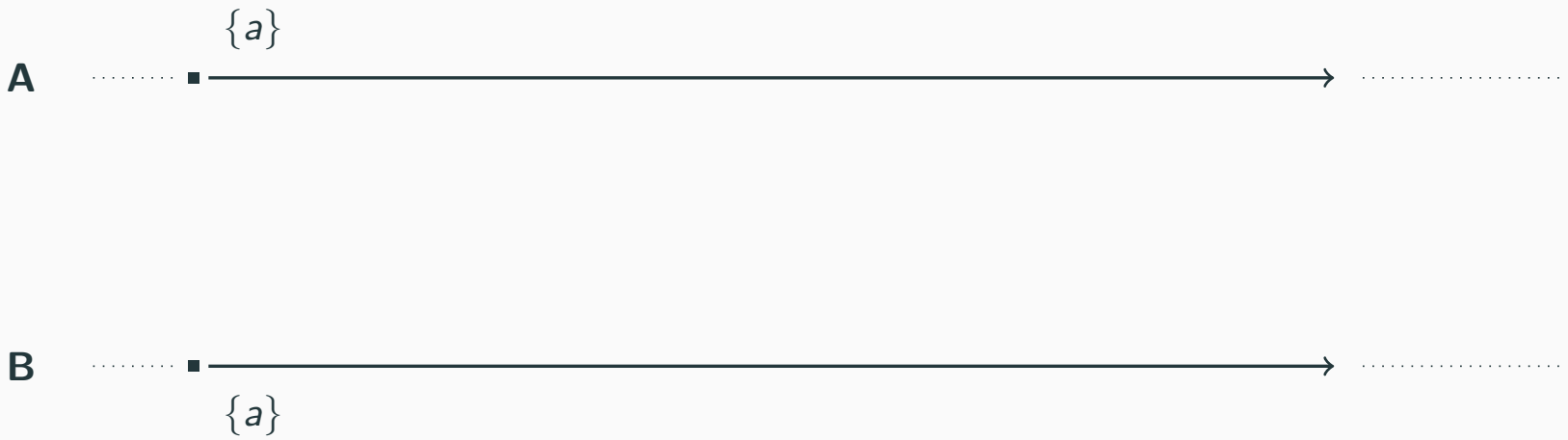
What impact the design of a given CRDT [Pre18]

- Conflict Resolution Semantics
- Synchronisation Model

- Impact their overhead in terms of computation, memory and bandwidth

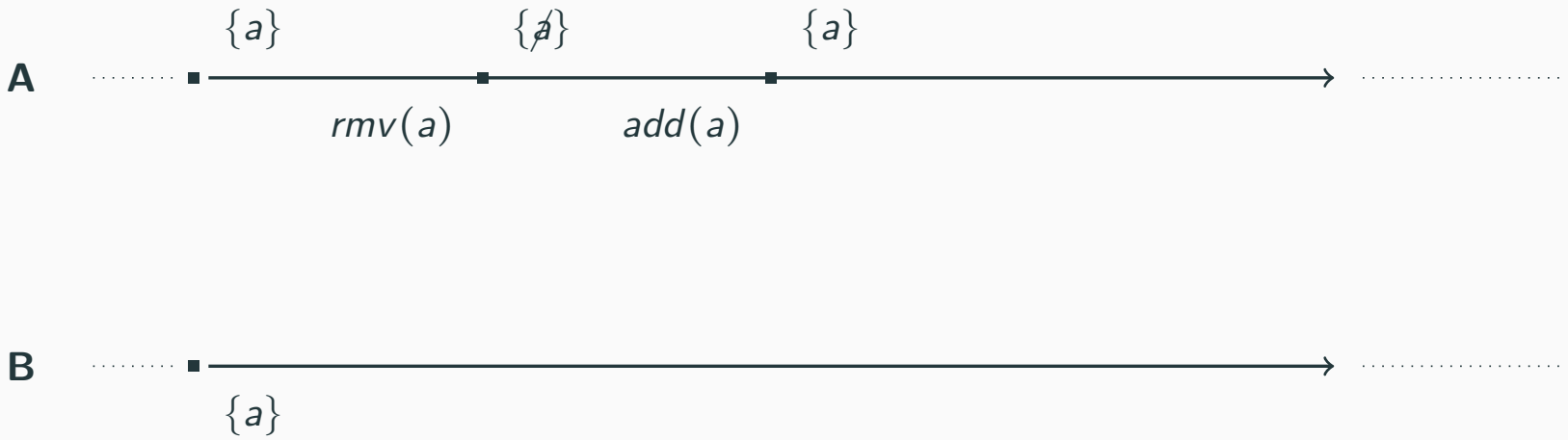
Conflict Resolution Semantics

- Distributed setting allows new scenarios



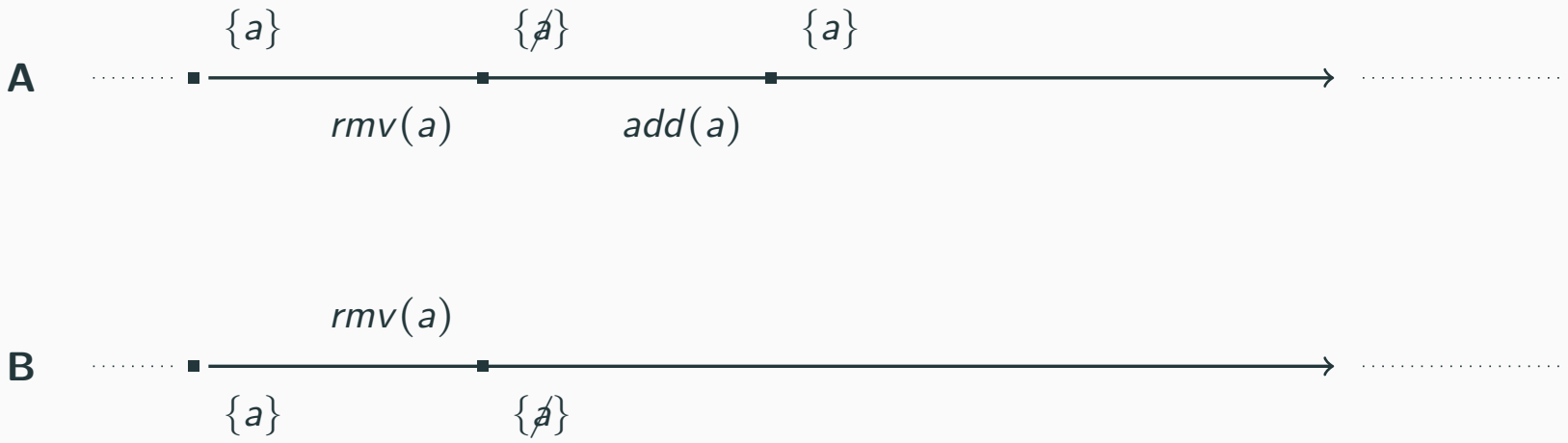
Conflict Resolution Semantics

- Distributed setting allows new scenarios



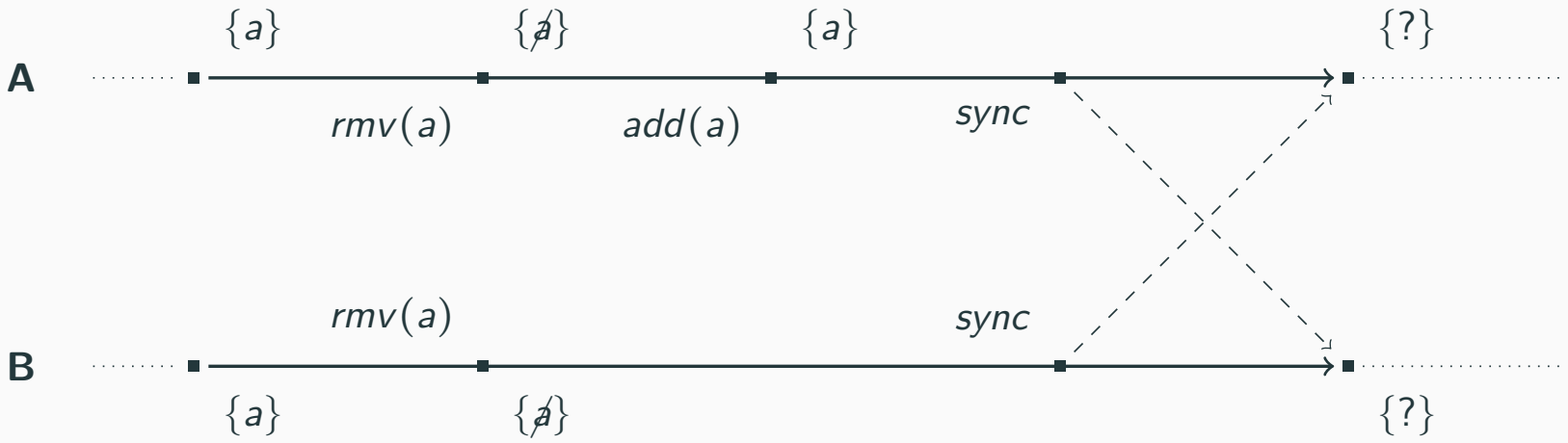
Conflict Resolution Semantics

- Distributed setting allows new scenarios



Conflict Resolution Semantics

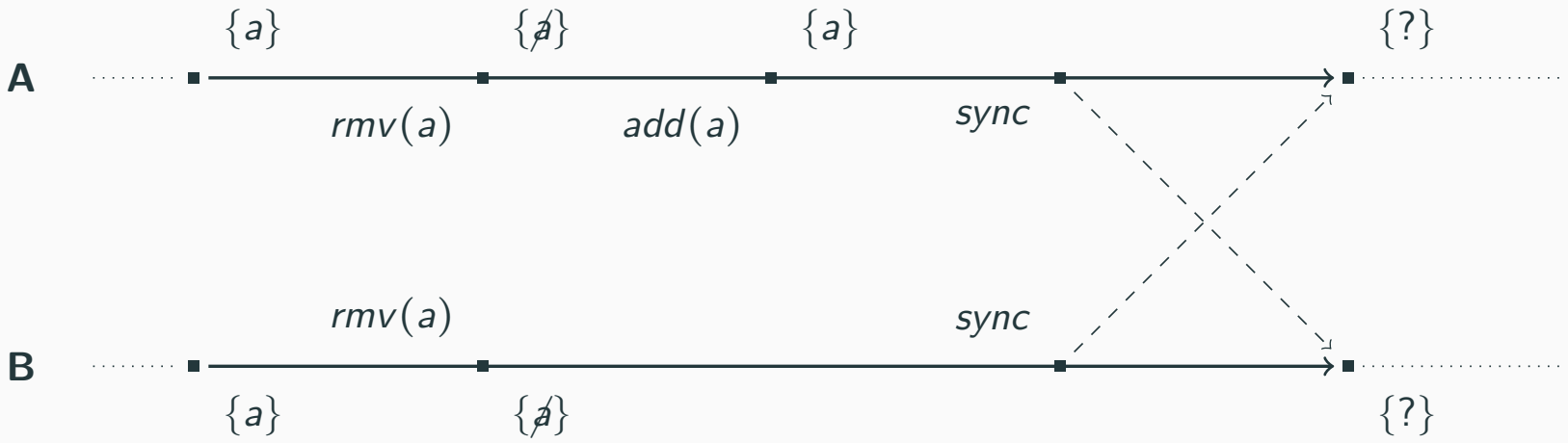
- Distributed setting allows new scenarios



- What should be the final state in this scenario ?

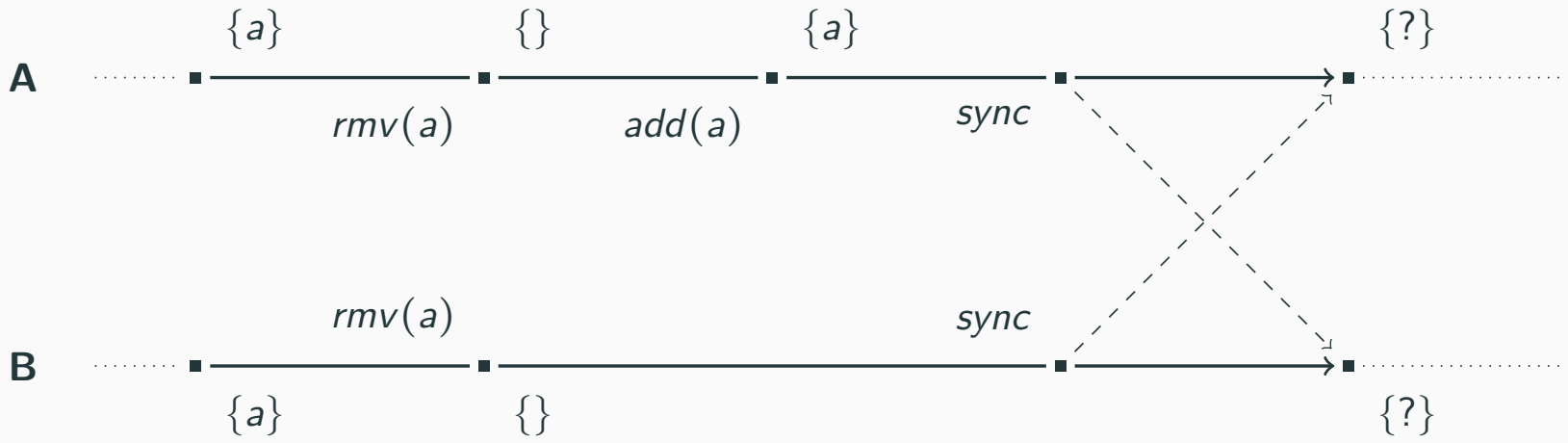
Conflict Resolution Semantics

- Distributed setting allows new scenarios



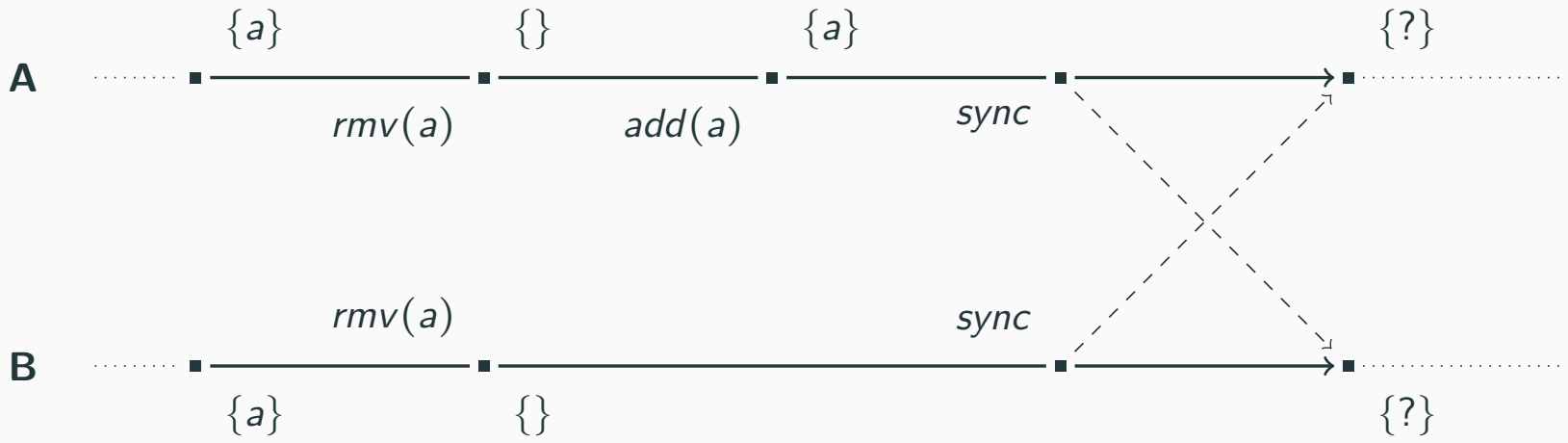
- What should be the final state in this scenario ?
- Designing a CRDT consists in defining its behaviour in such cases

Conflict Resolution Semantics - Case study of the Set



Several semantics proposed:

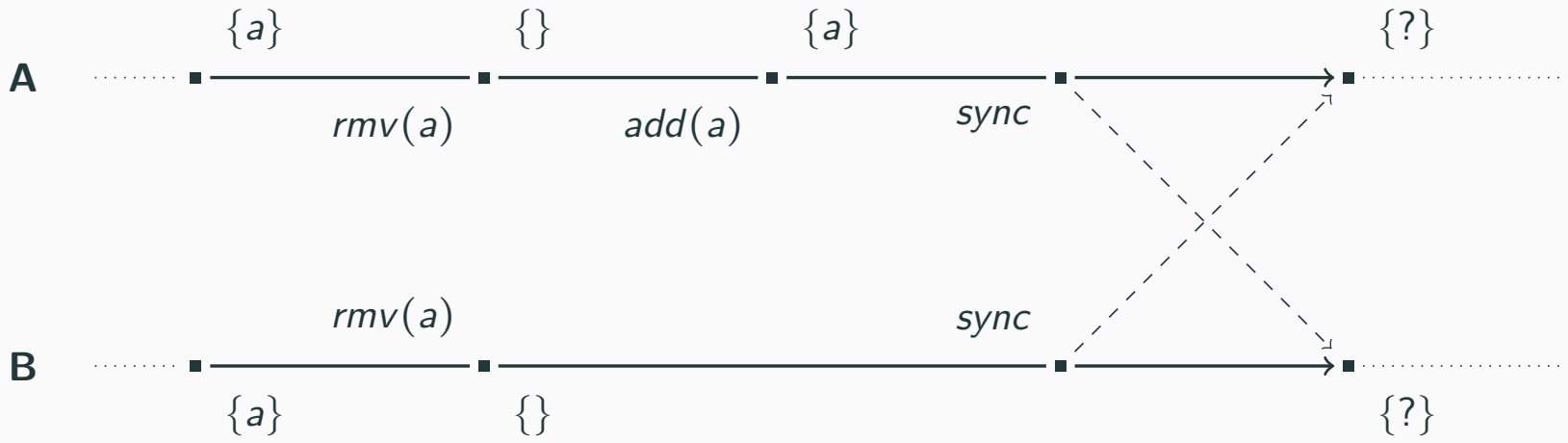
Conflict Resolution Semantics - Case study of the Set



Several semantics proposed:

- *Add-Wins*: *add*(*a*) has priority over concurrent *rmv*(*a*) $\Longrightarrow \{a\}$

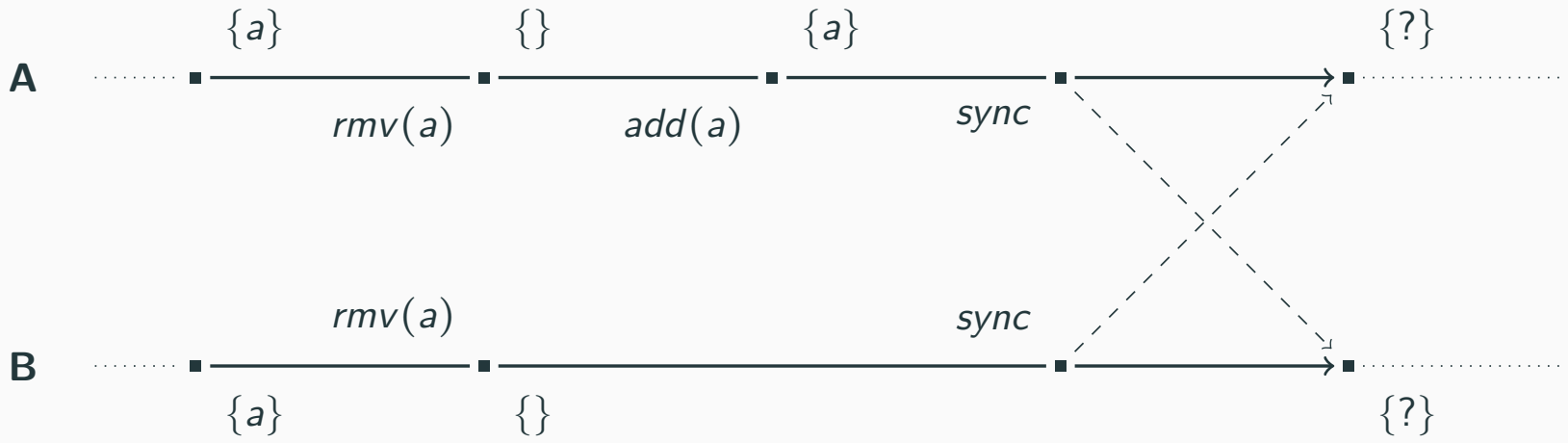
Conflict Resolution Semantics - Case study of the Set



Several semantics proposed:

- *Add-Wins*: $add(a)$ has priority over concurrent $rmv(a) \implies \{a\}$
- *Remove-Wins*: $rmv(a)$ has priority over concurrent $add(a) \implies \{\not{a}\}$

Conflict Resolution Semantics - Case study of the Set



Several semantics proposed:

- *Add-Wins*: *add*(*a*) has priority over concurrent *rmv*(*a*) $\Longrightarrow \{a\}$
- *Remove-Wins*: *rmv*(*a*) has priority over concurrent *add*(*a*) $\Longrightarrow \{\not{a}\}$
- *Causal-Length* [YR20]: The last action of the longuest chain of updates determines the presence (or not) of the element $\Longrightarrow \{a\}$

Synchronisation Models

To converge

- Nodes have to propagate changes . . .
- . . . And integrate those of others

Synchronisation Models

To converge

- Nodes have to propagate changes . . .
- . . . And integrate those of others

Several approaches proposed [Sha+11]

- State-based synchronisation
- Operation-based synchronisation

Synchronisation Models

To converge

- Nodes have to propagate changes ...
- ... And integrate those of others

Several approaches proposed [Sha+11]

- State-based synchronisation
- Operation-based synchronisation
- Delta-based synchronisation [ASB18]
    - "Best of the two worlds" approach

Synchronisation Models

To converge

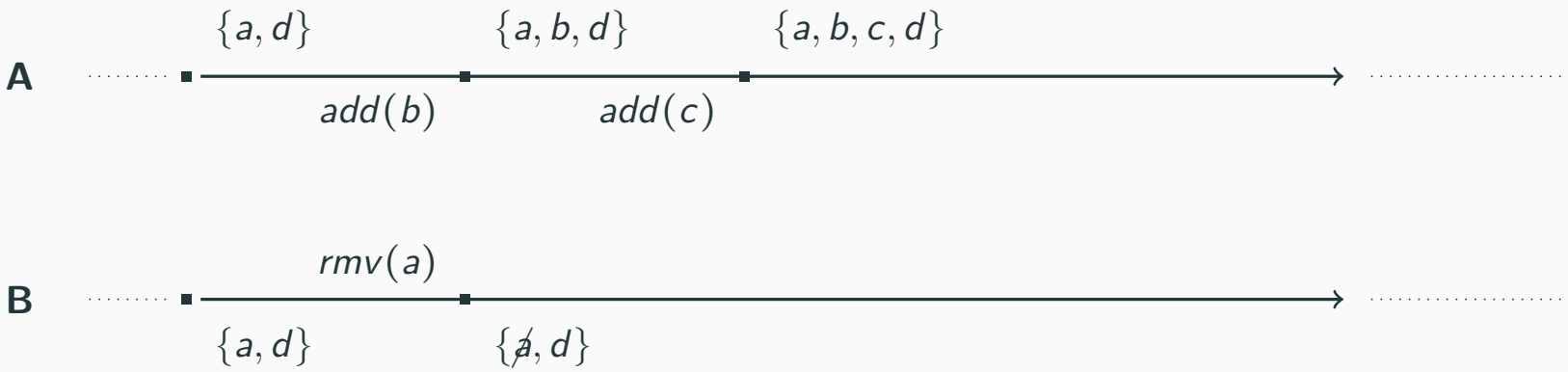
- Nodes have to propagate changes . . .
- . . . And integrate those of others

Several approaches proposed [Sha+11]

- State-based synchronisation
- Operation-based synchronisation
- Delta-based synchronisation [ASB18]
  - "Best of the two worlds" approach

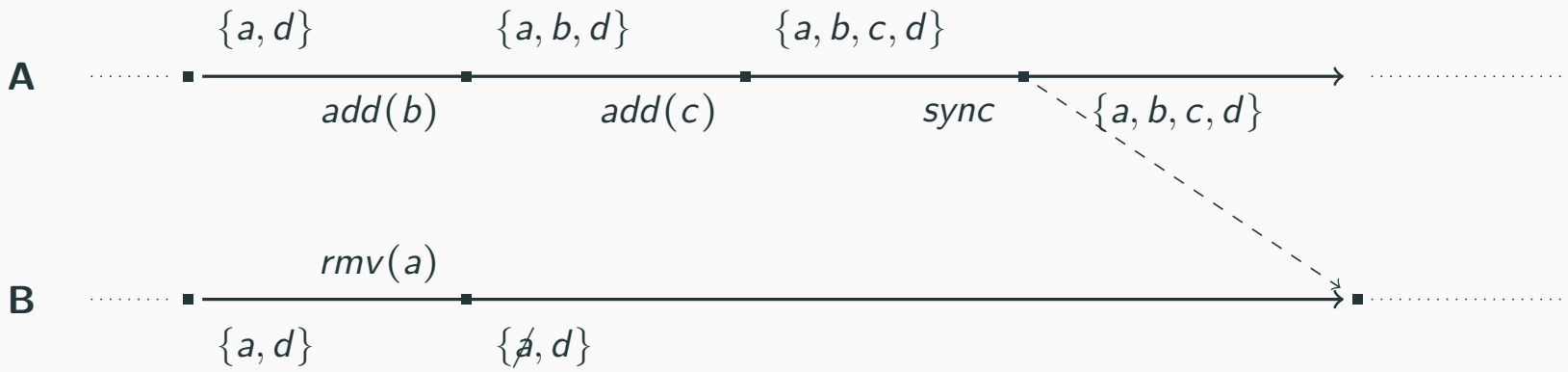
State-based synchronisation

- Send periodically current state to other nodes



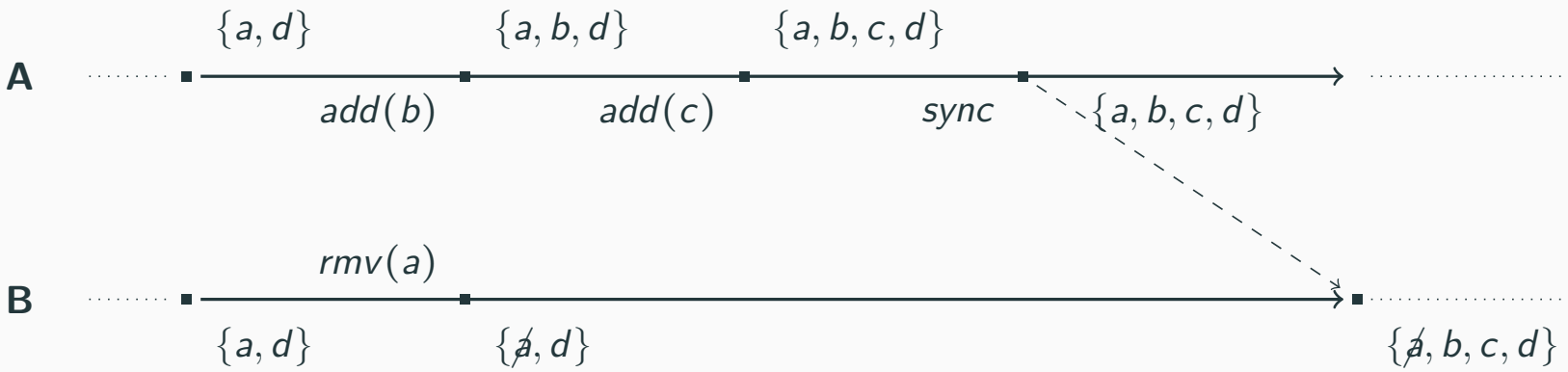
State-based synchronisation

- Send periodically current state to other nodes



State-based synchronisation

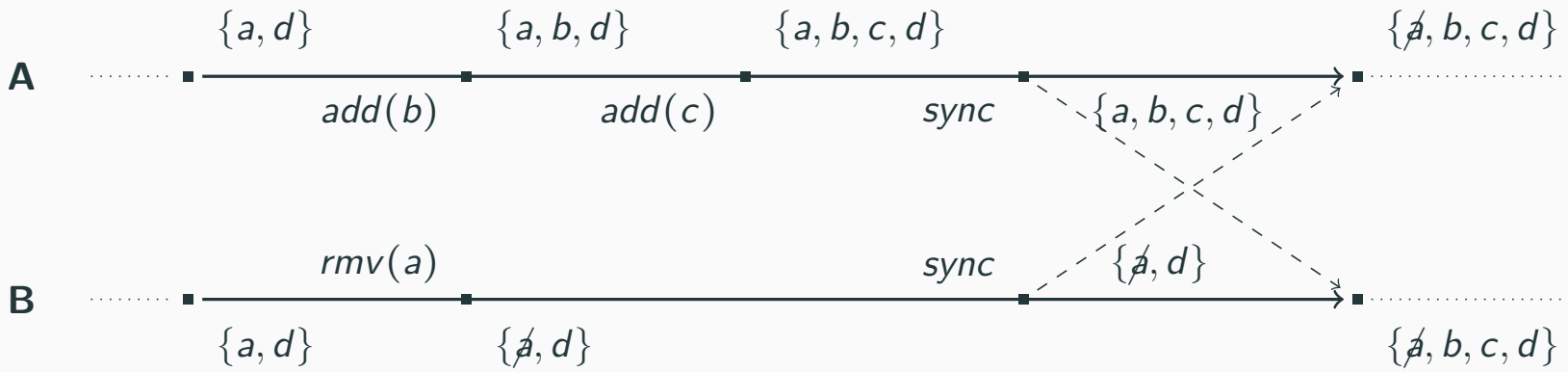
- Send periodically current state to other nodes



- Upon reception, computes new state by merging received state with current one using `merge` function (*join* in lattice theory)

State-based synchronisation

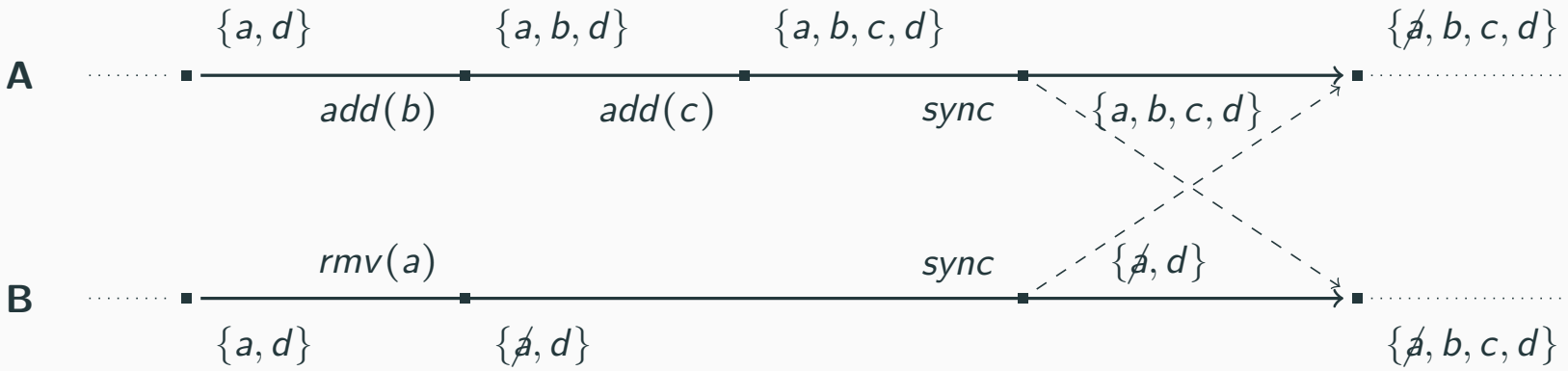
- Send periodically current state to other nodes



- Upon reception, computes new state by merging received state with current one using `merge` function (*join* in lattice theory)

State-based synchronisation

- Send periodically current state to other nodes



- Upon reception, computes new state by merging received state with current one using `merge` function (*join* in lattice theory)
- With `merge`, a commutative, associative and idempotent function

State-based synchronisation - Review

Strengths

- No assumptions on the network reliability
- i.e. messages may be lost, re-ordered or duplicated w/o impact

State-based synchronisation - Review

Strengths

- No assumptions on the network reliability
- i.e. messages may be lost, re-ordered or duplicated w/o impact

Limits

- States difficult to design
  - e.g. how to represent efficiently deletion of elements?

State-based synchronisation - Review

Strengths

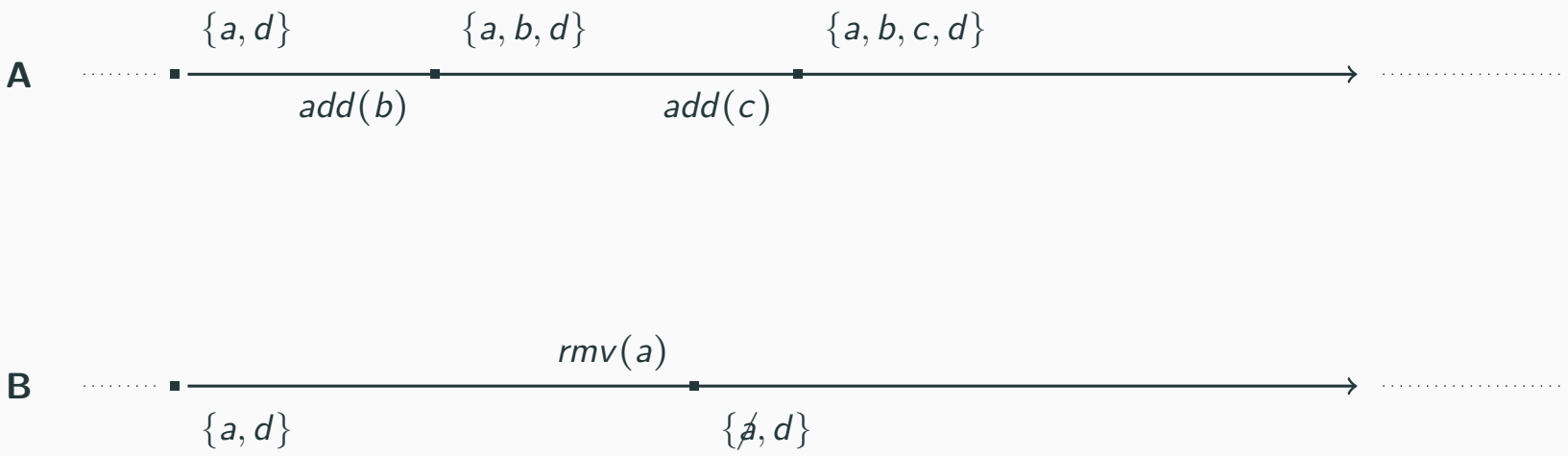
- No assumptions on the network reliability
- i.e. messages may be lost, re-ordered or duplicated w/o impact

Limits

- States difficult to design
  - e.g. how to represent efficiently deletion of elements?
- Depending on data type, states expensive to broadcast...
- ...And `merge` expensive to execute

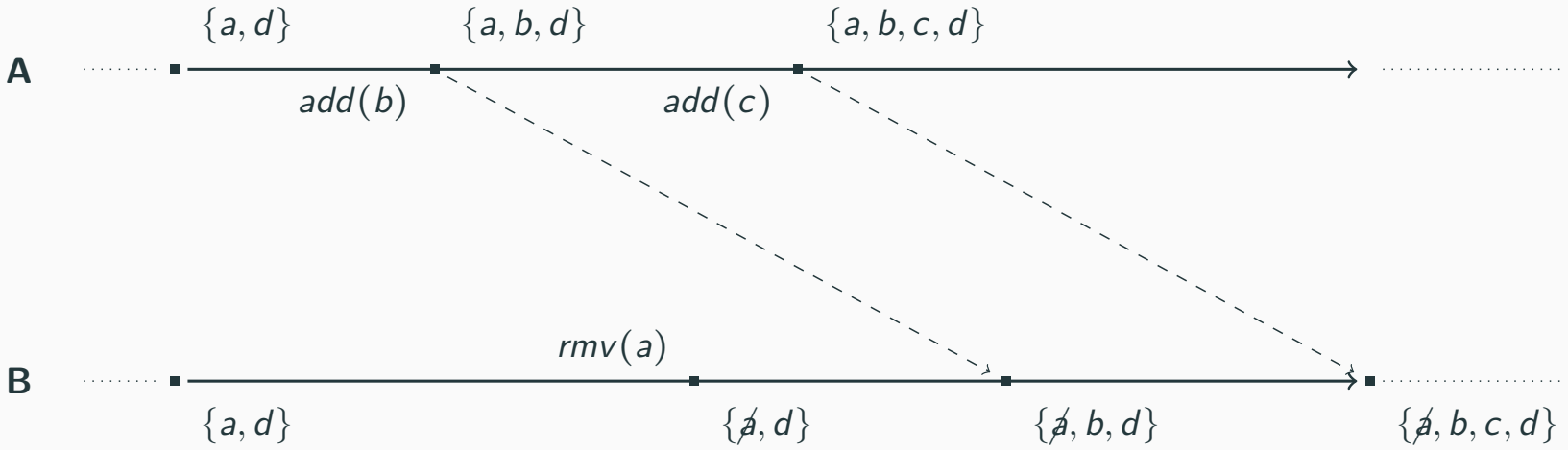
Operation-based synchronisation

- Encode updates as arbitrary messages, called *operations*
- An operation correponds to one or several *irreducible elements*



Operation-based synchronisation

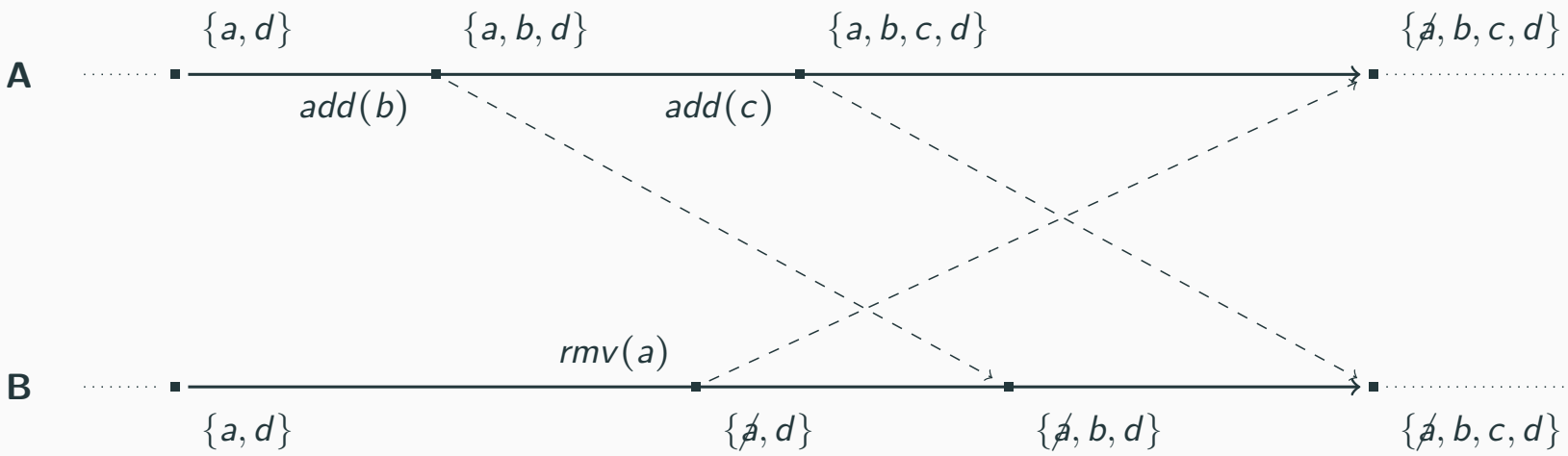
- Encode updates as arbitrary messages, called *operations*
- An operation correponds to one or several *irreducible elements*



- Upon reception, apply operations on current state

Operation-based synchronisation

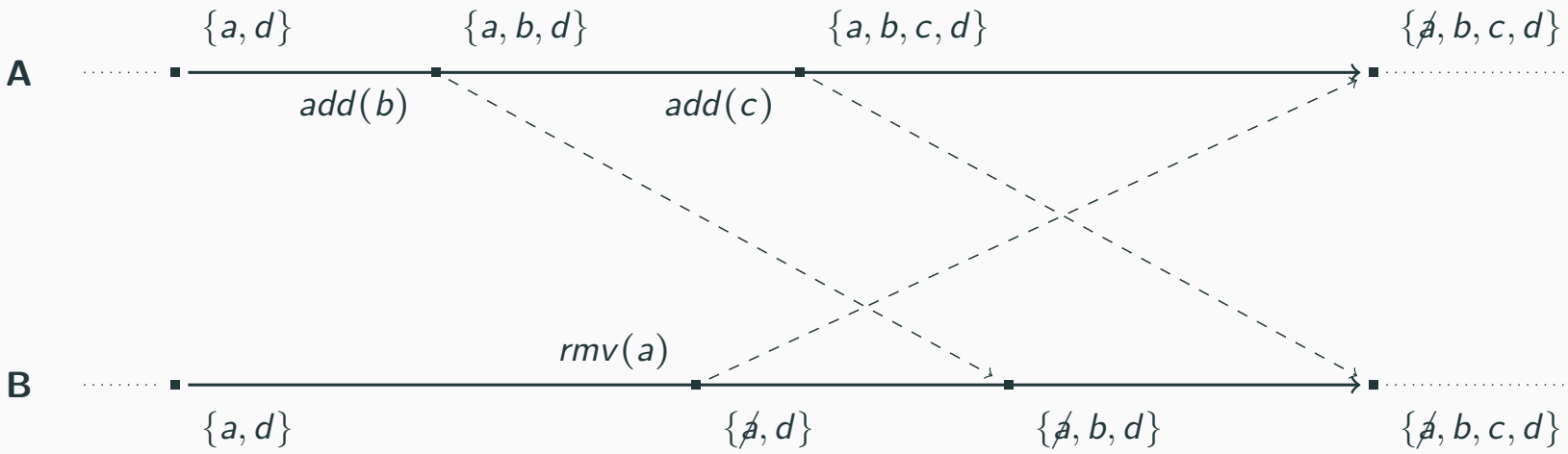
- Encode updates as arbitrary messages, called *operations*
- An operation correponds to one or several *irreducible elements*



- Upon reception, apply operations on current state

Operation-based synchronisation

- Encode updates as arbitrary messages, called *operations*
- An operation correponds to one or several *irreducible elements*



- Upon reception, apply operations on current state
- Concurrent operations must be commutative

Operation-based synchronisation - Review

Strengths

- Designing operations is straightforward
- Operations usually cheap to broadcast and apply

Operation-based synchronisation - Review

Strengths

- Designing operations is straightforward
- Operations usually cheap to broadcast and apply

Limits

- Hides/delegates complexity to delivery of operations
  - i.e. requires specific delivery order of operations
  - e.g. insertion of an element before its deletion

Operation-based synchronisation - Review

Strengths

- Designing operations is straightforward
- Operations usually cheap to broadcast and apply

Limits

- Hides/delegates complexity to delivery of operations
  - i.e. requires specific delivery order of operations
  - e.g. insertion of an element before its deletion
- Weak to network failures

Operation-based synchronisation - Review

Strengths

- Designing operations is straightforward
- Operations usually cheap to broadcast and apply

Limits

- Hides/delegates complexity to delivery of operations
  - i.e. requires specific delivery order of operations
  - e.g. insertion of an element before its deletion
- Weak to network failures
- Have to pair Op-based CRDTs to a message delivery service
  - To re-order and/or de-duplicate operations
  - To retrieve lost operations using anti-entropy mechanisms

To recap

CRDTs are new specifications of Data Types

- Enable nodes to collaboratively edit data w/o coordination
- Ensure Strong Eventual Consistency

To recap

CRDTs are new specifications of Data Types

- Enable nodes to collaboratively edit data w/o coordination
- Ensure Strong Eventual Consistency

Several CRDTs designed per data types

- With different behaviours in case of conflict...
- ...and different techniques to broadcast updates
- E.g. 10+ CRDTs for *Sequence*

State of the art

- CRDTs for many data types
  - *Register*, *Set* [Pre18], *Sequence* [Roh+11; WUM09], *JSON* [KB17], *Tree* [Kle+22] ...
- Libraries providing CRDTs to build new applications
  - Yjs [Yjs], Automerge [Aut]...

State of the art

- CRDTs for many data types
  - *Register*, *Set* [Pre18], *Sequence* [Roh+11; WUM09], *JSON* [KB17], *Tree* [Kle+22] ...
- Libraries providing CRDTs to build new applications
  - Yjs [Yjs], Automerge [Aut]...
- Used in collaborative applications
  - Teletype (Atom's pair-programming feature), Apple Notes...
- Used in multi-master replication for distributed databases
  - Redis [Lab], Microsoft Azure CosmoDB [DB]...

Ongoing researchs

- Designing CRDTs for new use cases
  - Rich Text [Lit+22], Access Control [RIP23]

Ongoing researchs

- Designing CRDTs for new use cases
  - Rich Text [Lit+22], Access Control [RIP23]
- Reducing CRDTs' overhead
  - Topic of my PhD, in the context of Sequence CRDTs [NOP22]

Ongoing researchs

- Designing CRDTs for new use cases
  - Rich Text [Lit+22], Access Control [RIP23]
- Reducing CRDTs' overhead
  - Topic of my PhD, in the context of Sequence CRDTs [NOP22]
- Designing CRDTs ensuring global invariants
  - Integrity constraints in relational databases [EI23; Lim]

Ongoing researchs

- Designing CRDTs for new use cases
  - Rich Text [Lit+22], Access Control [RIP23]
- Reducing CRDTs' overhead
  - Topic of my PhD, in the context of Sequence CRDTs [NOP22]
- Designing CRDTs ensuring global invariants
  - Integrity constraints in relational databases [EI23; Lim]
- Designing CRDTs for systems with malicious nodes

Thanks for your attention, any questions?

Bibliographie i

[Kle+19] Martin Kleppmann et al. **"Local-First Software: You Own Your Data, in Spite of the Cloud".** In: *Proceedings of the 2019 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*. Onward! 2019. Athens, Greece: Association for Computing Machinery, 2019, pp. 154–178. ISBN: 9781450369954. DOI: 10.1145/3359591.3359737. URL: https://doi.org/10.1145/3359591.3359737.

[Nic+17] Matthieu Nicolas et al. **"MUTE: A Peer-to-Peer Web-based Real-time Collaborative Editor".** In: *ECSCW 2017 - 15th European Conference on Computer-Supported Cooperative Work*. Vol. 1. Proceedings of 15th European Conference on Computer-Supported Cooperative Work - Panels, Posters and Demos 3. Sheffield, United Kingdom: EUSSET, Aug. 2017, pp. 1–4. DOI: 10.18420/ecscw2017_p5. URL: https://hal.inria.fr/hal-01655438.

Bibliographie ii

[Ter+95] Douglas B Terry et al. **"Managing Update Conflicts in Bayou, a Weakly Connected Replicated Storage System".** In: *SIGOPS Oper. Syst. Rev.* 29.5 (Dec. 1995), pp. 172–182. ISSN: 0163-5980. DOI: 10.1145/224057.224070. URL: https://doi.org/10.1145/224057.224070.

[Sha+11] Marc Shapiro et al. **"Conflict-Free Replicated Data Types".** In: *Proceedings of the 13th International Symposium on Stabilization, Safety, and Security of Distributed Systems*. SSS 2011. 2011, pp. 386–400. DOI: 10.1007/978-3-642-24550-3_29.

[Pre18] Nuno M. Preguiça. **"Conflict-free Replicated Data Types: An Overview".** In: *CoRR* abs/1806.10254 (2018). arXiv: 1806.10254. URL: http://arxiv.org/abs/1806.10254.

[YR20] Weihai Yu et al. **"A Low-Cost Set CRDT Based on Causal Lengths".** In: *Proceedings of the 7th Workshop on Principles and Practice of Consistency for Distributed Data*. New York, NY, USA: Association for Computing Machinery, 2020. ISBN: 9781450375245. URL: https://doi.org/10.1145/3380787.3393678.

Bibliographie iii

[ASB18] Paulo Sérgio Almeida et al. **"Delta state replicated data types".** In: *Journal of Parallel and Distributed Computing* 111 (Jan. 2018), pp. 162–173. ISSN: 0743-7315. DOI: 10.1016/j.jpdc.2017.08.003. URL: http://dx.doi.org/10.1016/j.jpdc.2017.08.003.

[Roh+11] Hyun-Gul Roh et al. **"Replicated abstract data types: Building blocks for collaborative applications".** In: *Journal of Parallel and Distributed Computing* 71.3 (2011), pp. 354–368. ISSN: 0743-7315. DOI: https://doi.org/10.1016/j.jpdc.2010.12.006. URL: http://www.sciencedirect.com/science/article/pii/S074373151000716.

Bibliographie iv

[WUM09] Stéphane Weiss et al. **"Logoot : A Scalable Optimistic Replication Algorithm for Collaborative Editing on P2P Networks".** In: *Proceedings of the 29th International Conference on Distributed Computing Systems - ICDCS 2009*. Montreal, QC, Canada: IEEE Computer Society, June 2009, pp. 404–412. DOI: 10.1109/ICDCS.2009.75. URL: http://doi.ieeecomputersociety.org/10.1109/ICDCS.2009.75.

[KB17] Martin Kleppmann et al. **"A Conflict-Free Replicated JSON Datatype".** In: *IEEE Transactions on Parallel and Distributed Systems* 28.10 (Oct. 2017), pp. 2733–2746. ISSN: 1045-9219. DOI: 10.1109/tpds.2017.2697382. URL: http://dx.doi.org/10.1109/TPDS.2017.2697382.

[Kle+22] Martin Kleppmann et al. **"A Highly-Available Move Operation for Replicated Trees".** In: *IEEE Transactions on Parallel and Distributed Systems* 33.7 (2022), pp. 1711–1724. DOI: 10.1109/TPDS.2021.3118603.

Bibliographie v

[Yjs] Yjs. **Yjs: A CRDT framework with a powerful abstraction of shared data.** Last Accessed: 2022-10-07. URL: https://github.com/yjs/yjs.

[Aut] Automerge. **Automerge: data structures for building collaborative applications in Javascript.** Last Accessed: 2022-10-07. URL: https://github.com/automerge/automerge.

[Lab] redis Labs. **Active-Active Geo-Distribution (CRDTs-Based).** Last Accessed: 2022-10-07. URL: https://www.redislabs.com/redis-enterprise/technology/active-active-geo-distribution/.

[DB] Azure Cosmos DB. **Azure Cosmos DB: Pushing the frontier of globally distributed databases.** Last Accessed: 2024-04-30. URL: https://azure.microsoft.com/en-us/blog/azure-cosmos-db-pushing-the-frontier-of-globally-distributed-databases/.

Bibliographie vi

[Lit+22] Geoffrey Litt et al. **"Peritext: A CRDT for Collaborative Rich Text Editing".** In: *Proceedings of the ACM on Human-Computer Interaction (PACMHCI)* 6.MHCI (Nov. 2022). DOI: 10.1145/3555644. URL: https://doi.org/10.1145/3555644.

[RIP23] Pierre-Antoine Rault et al. **"Access control based on CRDTs for Collaborative Distributed Applications".** In: *The International Symposium on Intelligent and Trustworthy Computing, Communications, and Networking (ITCCN-2023), in conjunction with the 22nd IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom-2023)*. Exeter, United Kingdom, Nov. 2023. URL: https://inria.hal.science/hal-04224855.

[NOP22] Matthieu Nicolas et al. **"Efficient Renaming in Sequence CRDTs".** In: *IEEE Transactions on Parallel and Distributed Systems* 33.12 (Dec. 2022), pp. 3870–3885. DOI: 10.1109/TPDS.2022.3172570. URL: https://hal.inria.fr/hal-03772633.

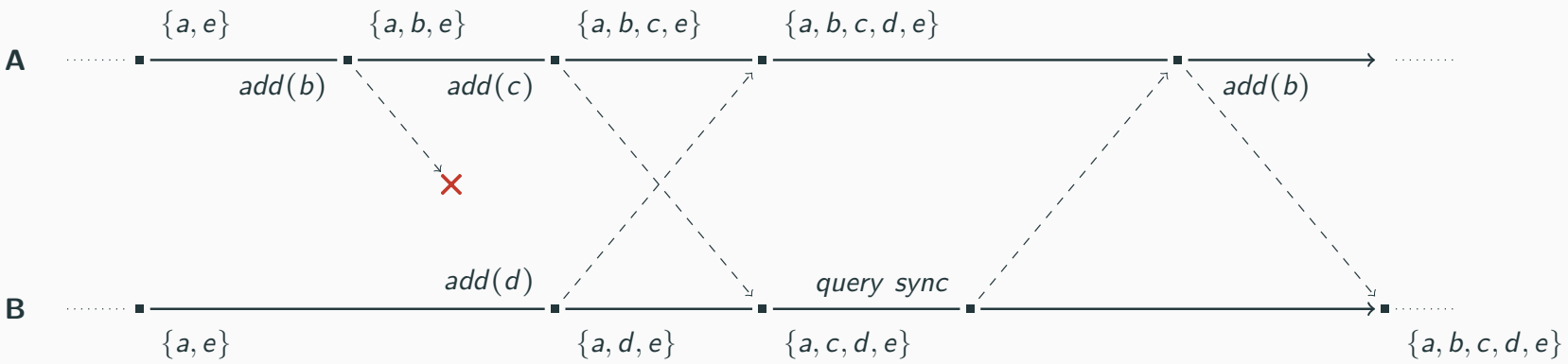
Bibliographie vii

[EI23] Victorien Elvinger et al. **Synql: Replicated Relations and Integrity Maintenance.** Tech. rep. Inria, Feb. 2023. URL: https://inria.hal.science/hal-03999168.

[Lim] Electric DB Limited. **ElectricSQL - Sync for modern apps.** Last Accessed: 2024-04-30. URL: https://electric-sql.com/.

Back-up slides

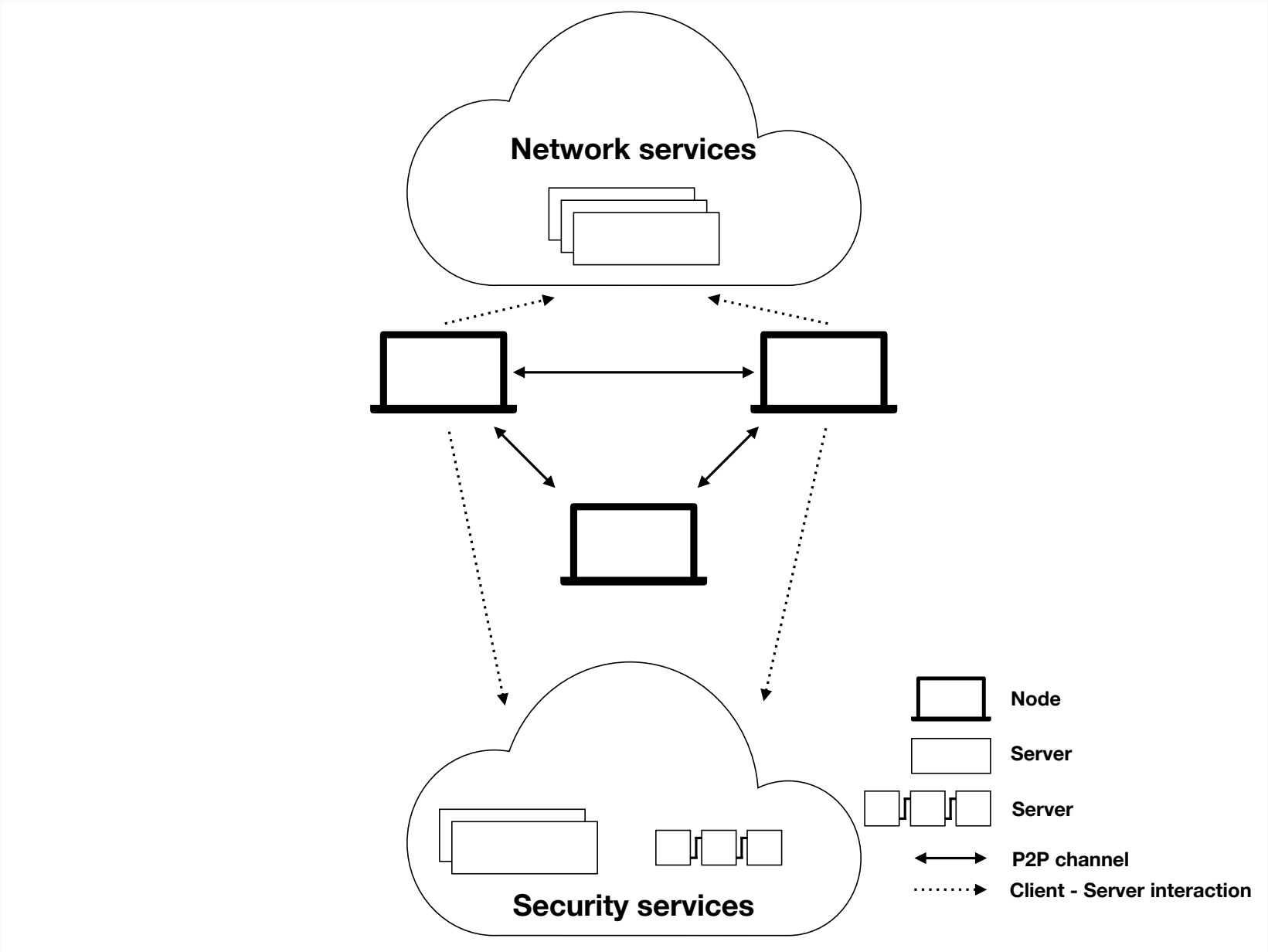
Operation-based synchronisation - network failure



Synchronisation Models - Summary

	State-based	Op-based	Delta-based
Integrate updates by merging states	✓	✗	✓
Integrate updates by irreducible elts	✗	✓	✓
Handle natively network failures	✓	✗	✓
Suited for real-time systems	✗	✓	✓

MUTE System Architecture



MUTE Software Architecture