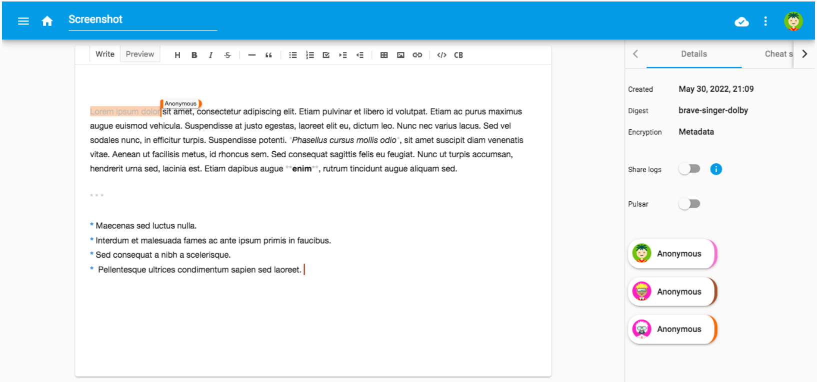
# Conflict-free Replicated Data Types (CRDTs)

An Overview

**Matthieu Nicolas** (`matthieu.nicolas@inria.fr`)

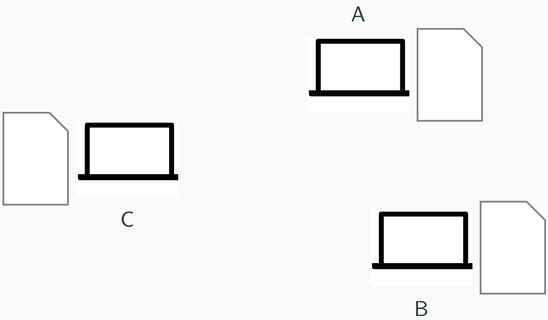
02/05/2024

# MUTE

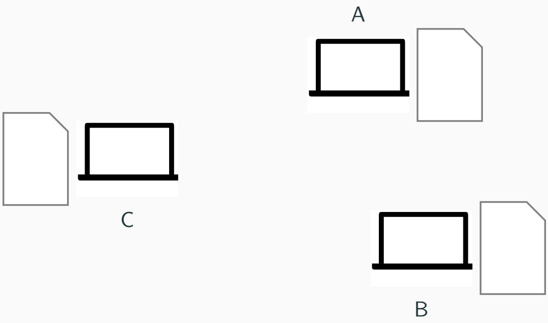



- Peer-to-Peer (P2P) application
- Allow to edit collaboratively text documents
- Ensure ownership and privacy of data
- Part of the Local-First Software [Kle+19] trend

# Data replication in P2P systems

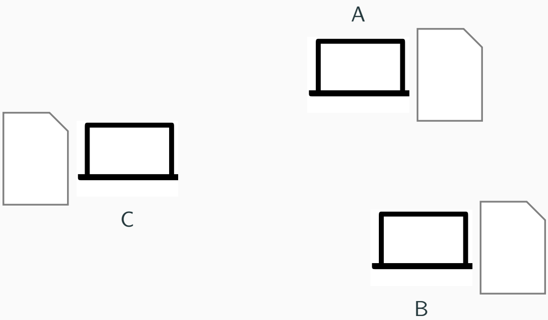


# Data replication in P2P systems



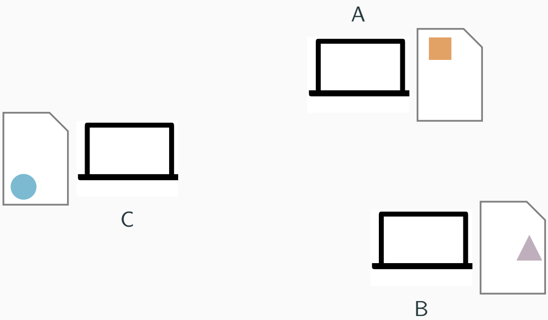

- Nodes may be disconnected

# Data replication in P2P systems



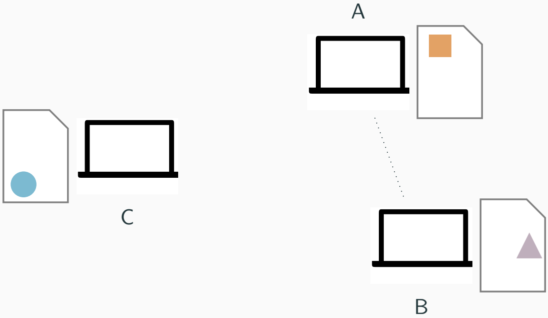

- Nodes may be disconnected
- Have to be able to work without prior synchronous coordination (i.e. consensus)

# Data replication in P2P systems



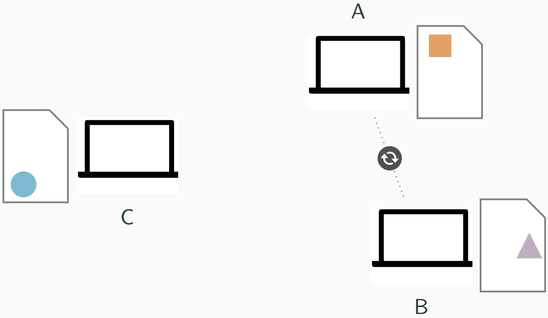

- Nodes may be disconnected
- Have to be able to work without prior synchronous coordination (i.e. consensus)

# Data replication in P2P systems



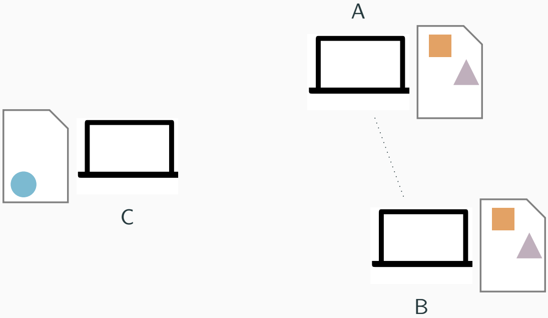

- Nodes may be disconnected
- Have to be able to work without prior synchronous coordination (i.e. consensus)

# Data replication in P2P systems



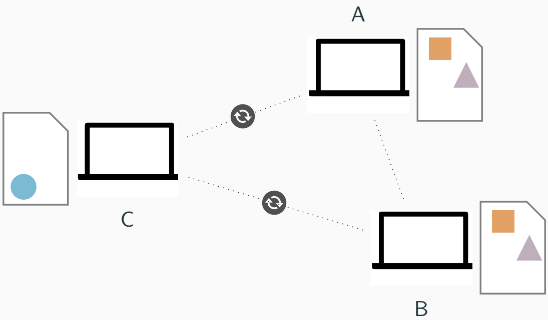

- Nodes may be disconnected
- Have to be able to work without prior synchronous coordination (i.e. consensus)

# Data replication in P2P systems



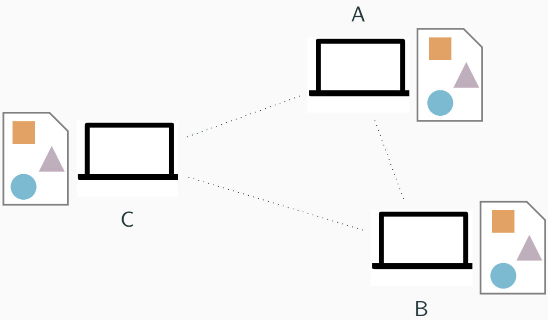

- Nodes may be disconnected
- Have to be able to work without prior synchronous coordination (i.e. consensus)

# Data replication in P2P systems




- Nodes may be disconnected
- Have to be able to work without prior synchronous coordination (i.e. consensus)

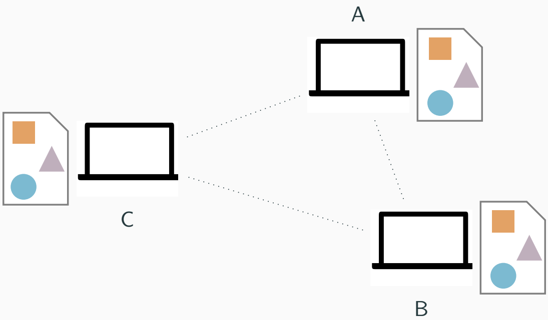
# Data replication in P2P systems




- Nodes may be disconnected
- Have to be able to work without prior 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
- Have to be able to work without prior 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**

# 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-semilattice

# Design of CRDTs

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

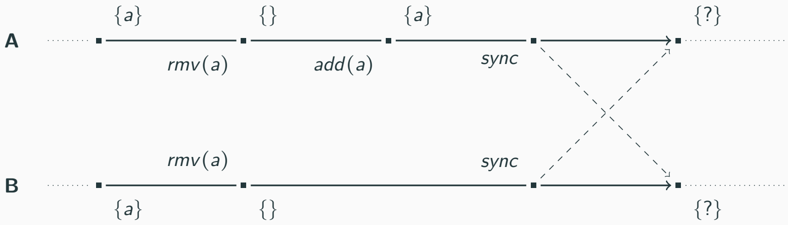
**What impact the design of a given CRDT are its**

- Conflict Resolution Semantics
- Synchronisation Model

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

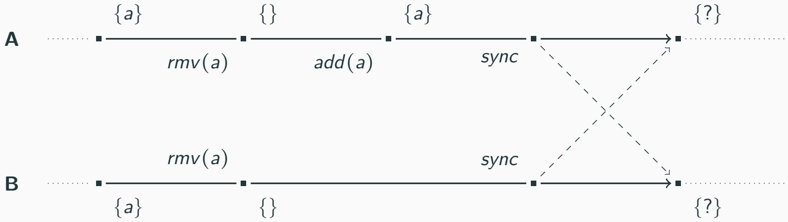
# Conflict Resolution Semantics

- Distributed setting allows new scenarios




- Results of these executions are undefined
- Designing a CRDT consists in defining its behaviour in such cases

# Conflict Resolution Semantics - Case study of the Set




**Several semantics were proposed:**

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

# Synchronisation Models

**To converge**

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

**Several approaches proposed**

- State-based synchronisation
- Operation-based synchronisation

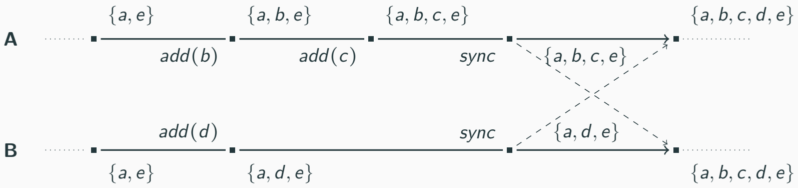
# An aparté about lattice theory

**Properties of join-semilattices**

- States of the join-semilattices are 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 $\leq$

# 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
- With `merge`, a commutative, associative and idempotent function

# State-based synchronisation - ???

**Strengths**

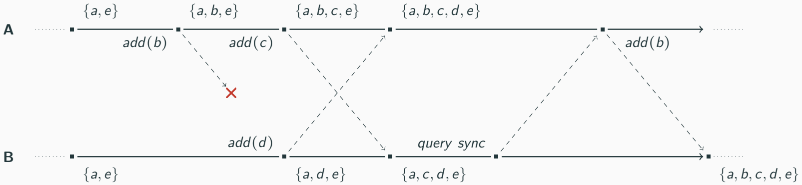
- No assumptions on the network reliability
- i.e. messages may be lost, re-ordered or duplicated w/o impact

**Limits**

- Sending states may be expensive
- `merge` may be expensive and difficult to design

# Operation-based synchronisation

- Encode updates as arbitrary messages, called *operations*




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

# Operation-based synchronisation - ???

**Strengths**

- Operations usually cheap to broadcast and apply

**Limits**

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

# Synchronisation Models - Recap

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

[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.

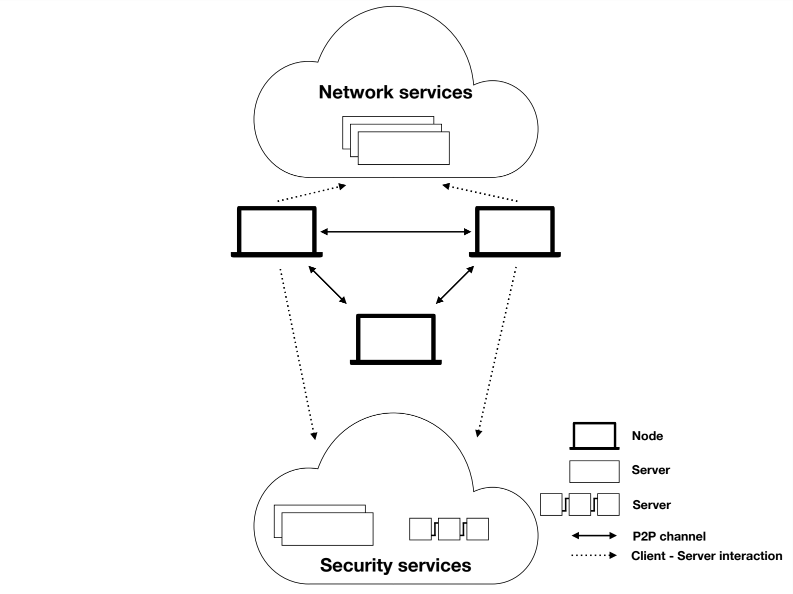
[Ter+95] Douglas B Terry et al. **“Managing Update Conflicts in Bayou, a Weakly Connected Replicated Storage System”.** In: *SIGOPS Oper. Syst. Rev.* 29.5 (Dec. 1995), pp. 172–182. ISSN: 0163-5980. DOI: 10.1145/224057.224070. URL: https://doi.org/10.1145/224057.224070.

[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.

[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.

# Back-up slides

# MUTE System Architecture



# MUTE Software Architecture