Don’t Settle for Eventual: Scalable Causal Consistency for Wide-Area Storage with COPS

Authors: Lloyd et al.
Presenter: Abhishek A. Singh

Ideas

- ALPS – **A**vailability, Low **L**atency, **P**artition Tolerance and **S**calability
- Causal+,
- COPS GT
- Design of COPS/COPS GT key-value store

What is Causal?

Causal + X = Causal+

- X = *convergent conflict handling*

- Implemented via **COPS**
  - **C**lusters of **O**rder-**P**reserving **S**ervers

System properties desired

- Availability
  - Ops. do not fail; do not block indefinitely; no error
- Low latency
  - Ops. complete “quickly” (in milliseconds)
- Partition tolerance
  - Data Store continues to operate under partitions
- High scalability
  - Linear increase in system performance on addition to resources
- Stronger consistency
  - Causal+ consistency. Weaker than linearizability.

Causal+ consistency

- Potentially causal → (in the paper a squiggly arrow)
  - If *a* and *b* are two operations in a single thread of execution, then *a*→*b*
  - If *a* := *puts()* operation, and *b* := *gets(a)*, then *a*→*b*
    - *gets(a)* returns value put by *a*
  - If *a*→*b* and *b*→*c*, then *a*→*c*

Define causal+ consistency

- Causal consistency + convergent conflict handling
- Convergent conflict handling
  - all conflicting puts should be handled in the same way across all replicas using some handler function *h*
  - Problem to be solved: avoid state divergence across the network.
    - Eg. *last-writer-wins*

Consistency foodchain

Linearizability > **Sequential** > Causal+ > Causal > FIFO
> Per-Key Sequential > Eventual

Causal+ in COPS

- Remember COPS?
- **C**lusters of **O**rder-**P**reserving **S**ervers
- Defines 2 abstractions:
  - *Versions*
    - If $x_i \rightarrow y_j$, then $i < j$
    - Only return causally later version of a key (progressing property)
  - *Dependencies*
    - *If $x_i \rightarrow y_j$, then x_i must be written before y_j*
    - *y_j **depends on** $x_i \leftrightarrow put(x_i) \rightarrow put(y_j)$*

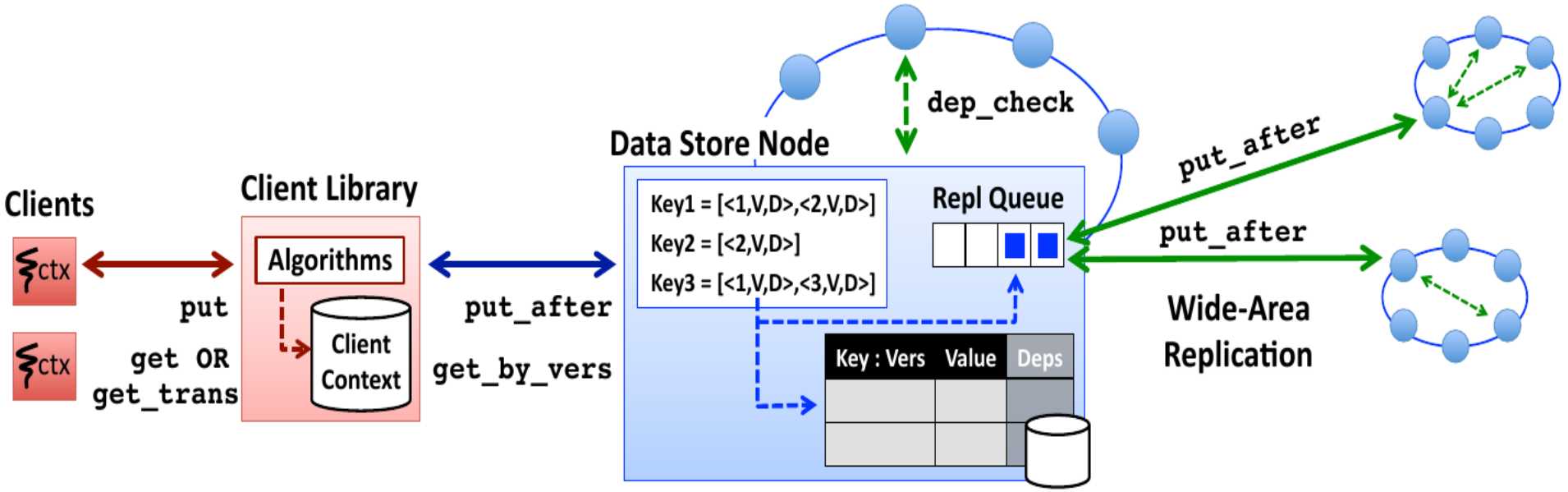
Scalable causality?

- Dependencies encoded in each key’s metadata.
- During key replication each receiving data center performs dependency check before committing the incoming version

The design of COPS

- Local datacenter clusters implement linearizability
- Remote datacenters: causal replication

COPS Architecture



Client library interface

- ctx id ← createContext()
- bool ← deleteContext(ctx id)
- bool ← put (key, value, ctx id)
- value ← get (key, ctx id) [In COPS]

  OR
- [values] ← get trans (hkeysi, ctx id) [In COPS-GT]

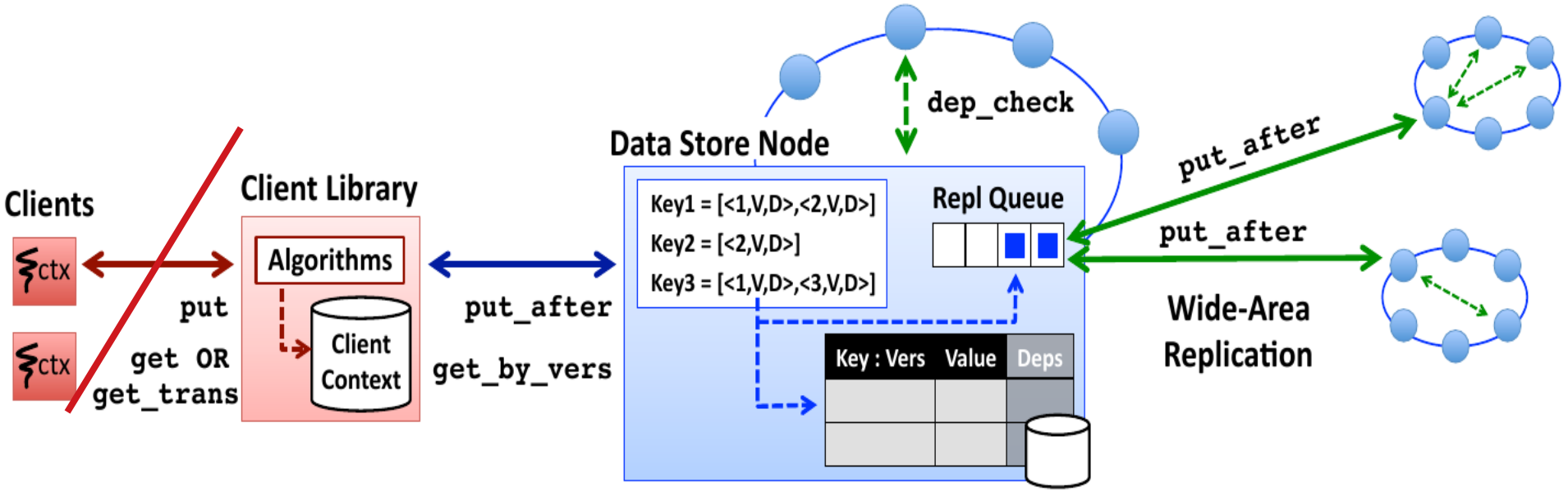
Read, write and replicate in COPS/COPS-GT

- [bool,vers] ← put after (key, val, [deps], nearest, vers=∅)
-
- bool ← dep check (key, version)
- [value, version, dep] ← get by version (key, version=LATEST)

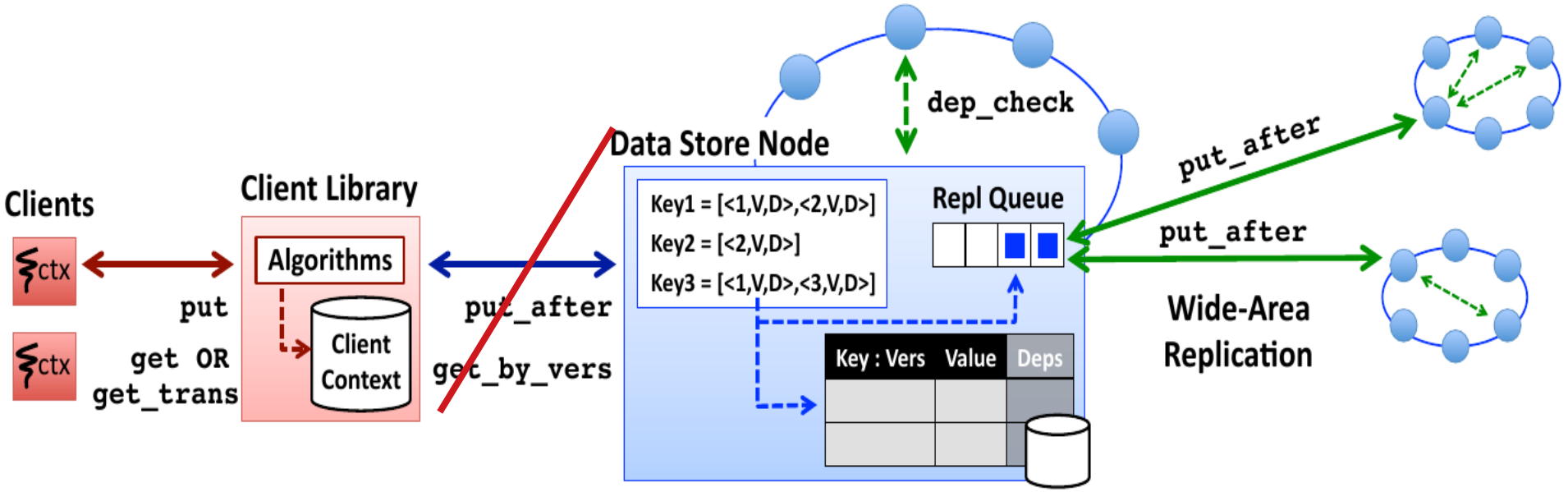
GT part of COPS-GT

- Get Transactions API
  - [values] ← get_trans(keys, ctx_id)
- Why does this have to be exposed to the client and not handled internally?
- Is COPS-GT really “stronger” than COPS? How?

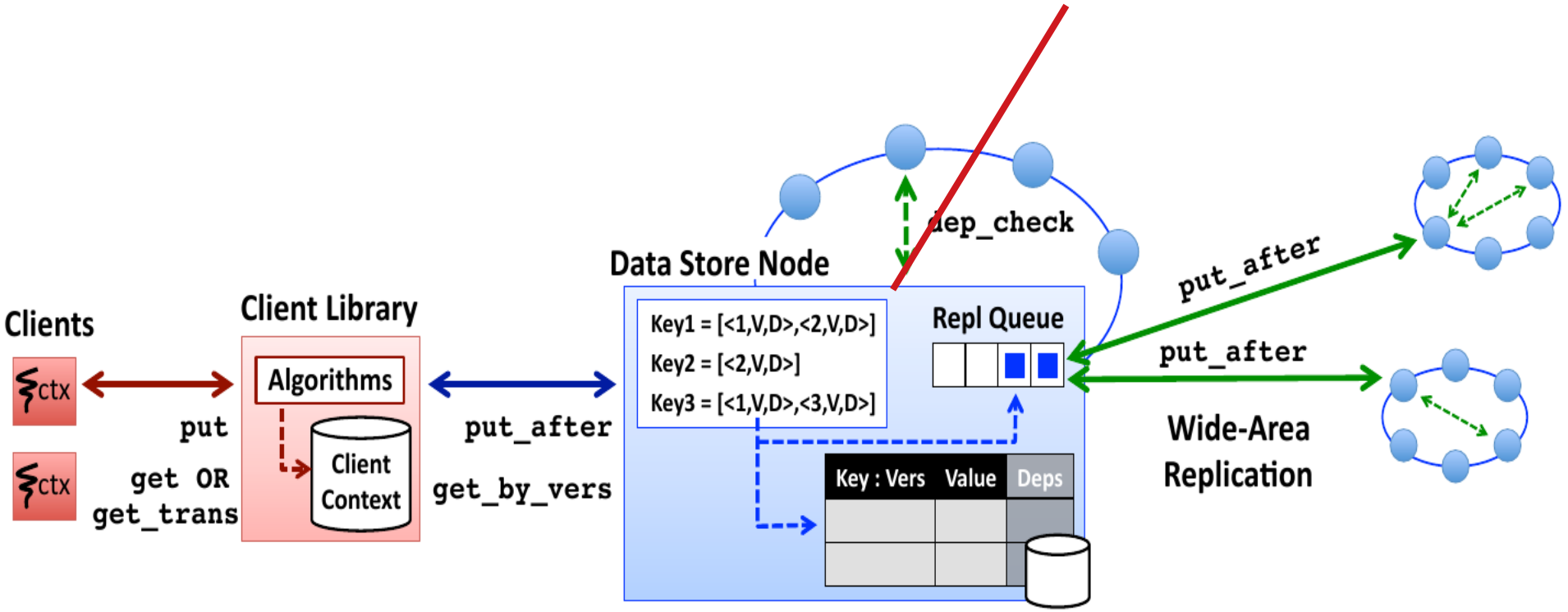
Faults: Scenario 1



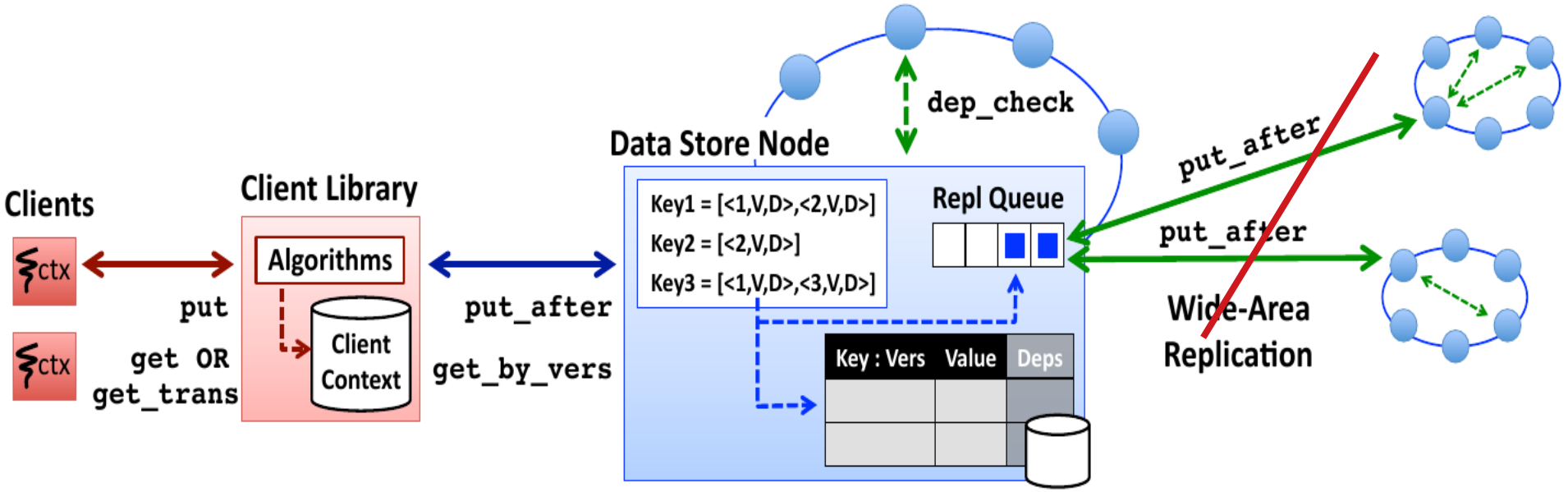
Faults: Scenario 2



Faults: Scenario 3



Faults: Scenario 4



Faults: All you can fault