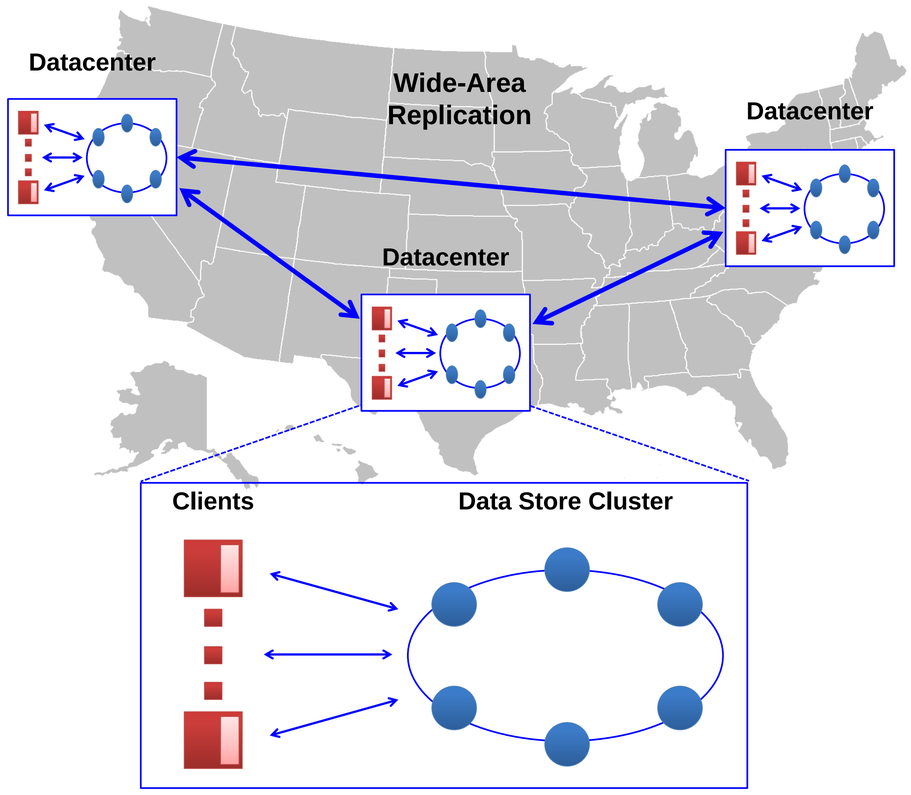
SCALABLE CAUSAL CONSISTENCY FOR WIDE-AREA STORAGE WITH COPS

Wyatt Lloyd (Princeton), Michael J. Freedman (Princeton), Michael Kaminsky (Intel Labs), David G. Andersen (CMU)

MOTIVATION

- Distributed data stores support complex online applications
  - e.g. social networks
- Theory constrains properties
  - CAP Theorem
  - Seq Consistency || Low Latency
- Most practical systems adopt eventual consistency
  - Complicates application logic
  - Exposes inconsistencies to users



IDEAL PROPERTIES

- Availability
- Low Latency
- Partition Tolerance
- Scalability
- Stronger Consistency
- Systems with the first four properties are ALPS systems

CAUSAL+ CONSISTENCY

- Causal consistency
  - Related ops appear in the correct order
- Plus convergent conflict handling
  - Conflicting puts are handled identically in each DC
- Spectrum of Consistency Models:

Linearizability > Seq. > Causal+ > Causal > FIFO > PK Seq. > Eventual
(Impossible with ALPS)

CAUSAL+ EXAMPLES

1) Alice uploads photo
2) Alice adds photo to album

Causal+: Referential integrity. Photo always exists before album.

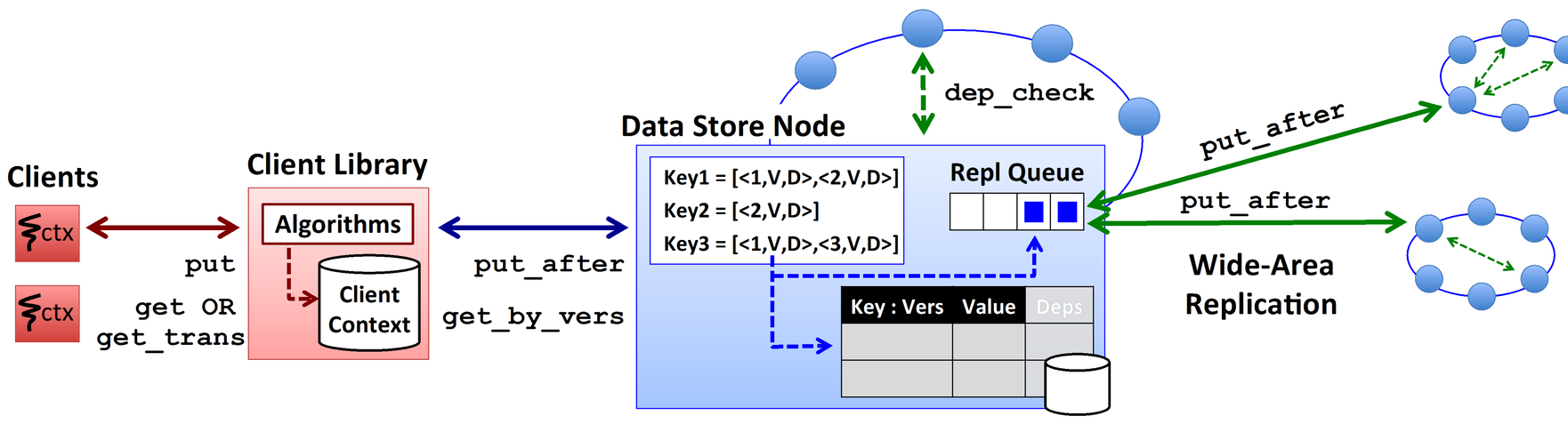
Eventual: Broken reference in album is possible.

A) Carol sets coffee.time = 8am
B) Dave sets coffee.time = 10am

Causal+: One time will be agreed upon. Either 8am, 10am, or something fancier.

Causal: Forever divergent times are possible.

CLUSTERS OF ORDER PRESERVING SERVERS



Client Library

- Interface hides complexity from programmer
- Calls include a context that tracks causality
- Get transactions provide a consistent view of multiple keys, even from diff. nodes

Key-Value Store

- Client ops are local, replication occurs in the background
  - Provides availability, low latency, partition tolerance
- Lamport timestamps version writes
  - Used to enable get transactions and in the default last-writer-wins conflict handler
- Put_after and dep_check operations order replication between clusters and nodes
  - Provides causal consistency

Challenges

- Minimize space footprint
  - Garbage collect old state
- Minimize overhead of consistent replication
  - Leverage transitivity of causality
- Ensure fast get transactions: Worst-case 2 rounds under concurrent writes
  - Get_by_version

Implementation

- Built on top of FAWN-KV
- ~13,000 LOC
- Latency < 1ms
- Throughput similar to weaker systems
- Scales linearly

Intel Science & Technology Center for Cloud Computing

Carnegie Mellon University · Georgia Tech · intel · Princeton University · UC Berkeley