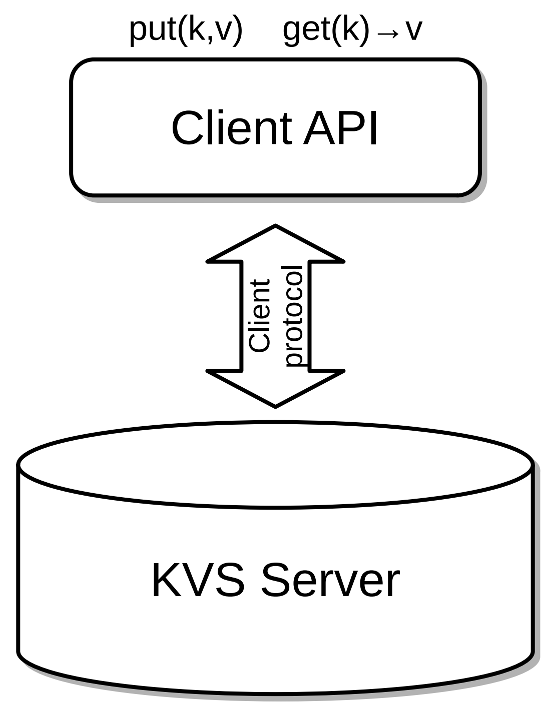
Large Scale Distributed Systems

José Orlando Pereira

Departamento de Informática
Universidade do Minho

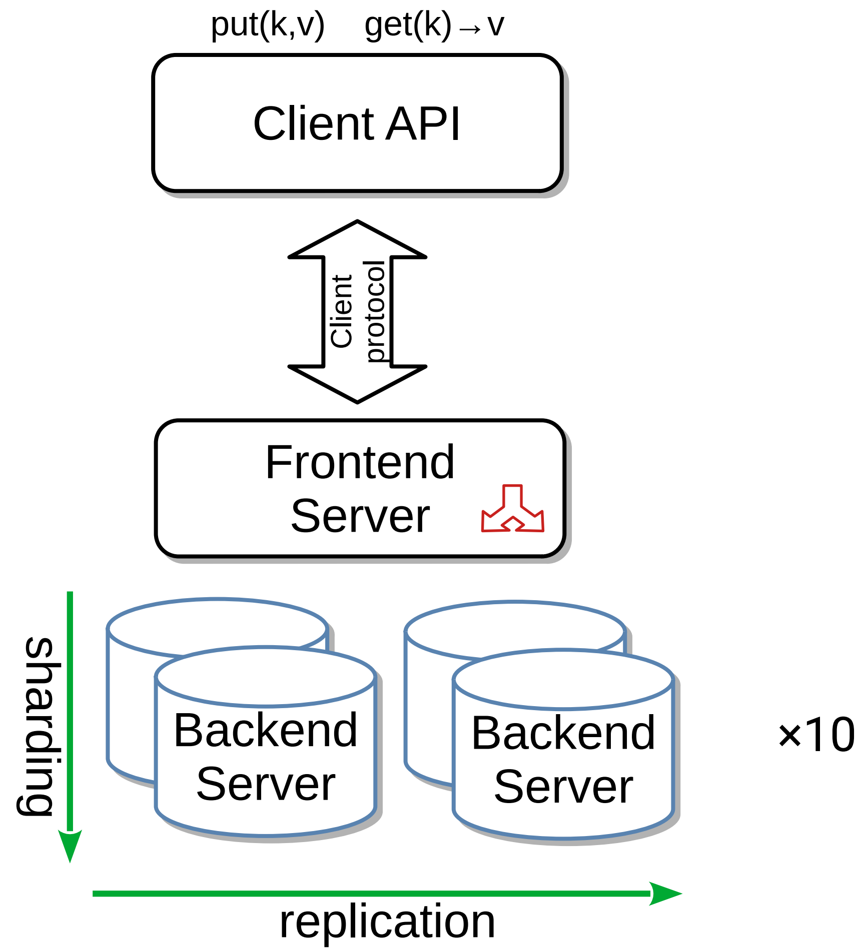
Key-Value Store

- Simple data model:
  - Map<K,V>
- Simple interface:
  - get(k) → v
  - put(k,v)
- <u>Avoids session state</u> in server:
  - No multi-item transactions
  - No long lived operations



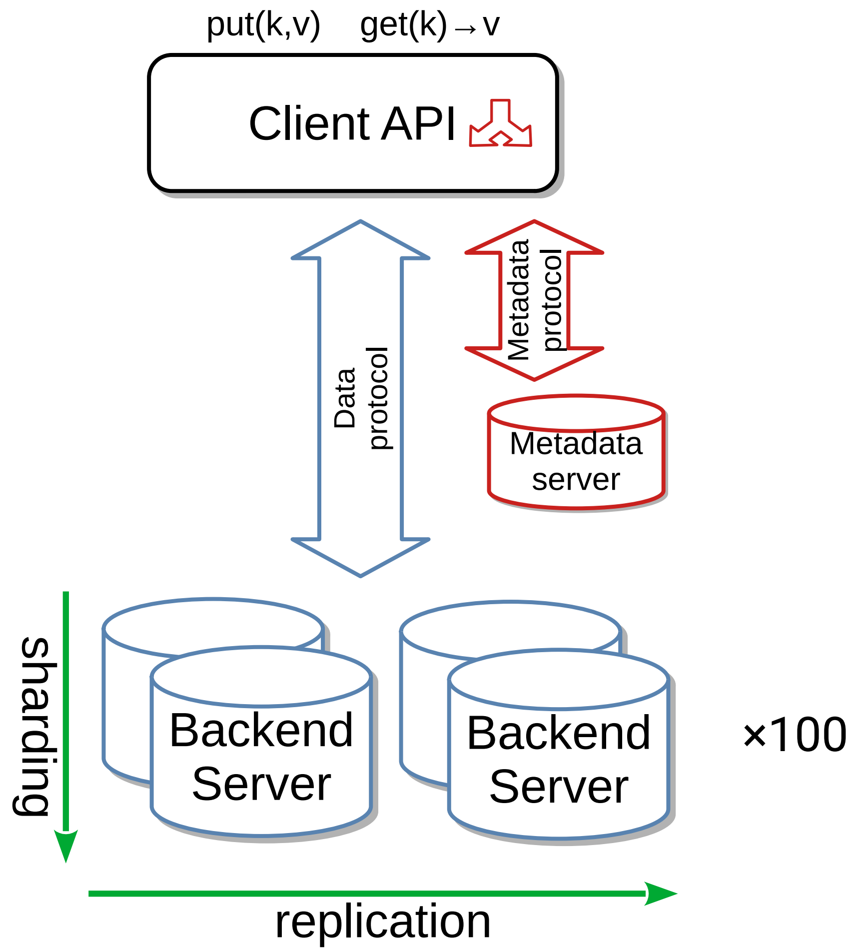
Centralized

- Replication for availability
- Sharding for scale-out
- Centralized architecture:
  - Frontend server with metadata store
  - Backend servers with data stores
- Frontend is a bottleneck (latency and bandwidth)
- Example: mongoDB



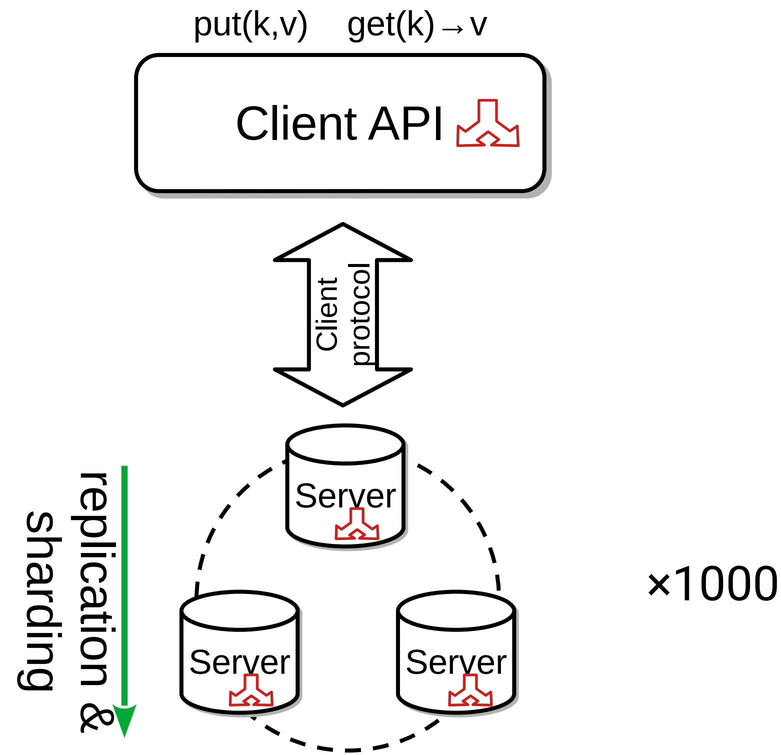
Decentralized data

- <u>Separate data and metadata</u> servers and protocols
  - Client-side caching of metadata
- Avoids data bottleneck
- Examples:
  - Cloud Bigtable
  - APACHE HBASE



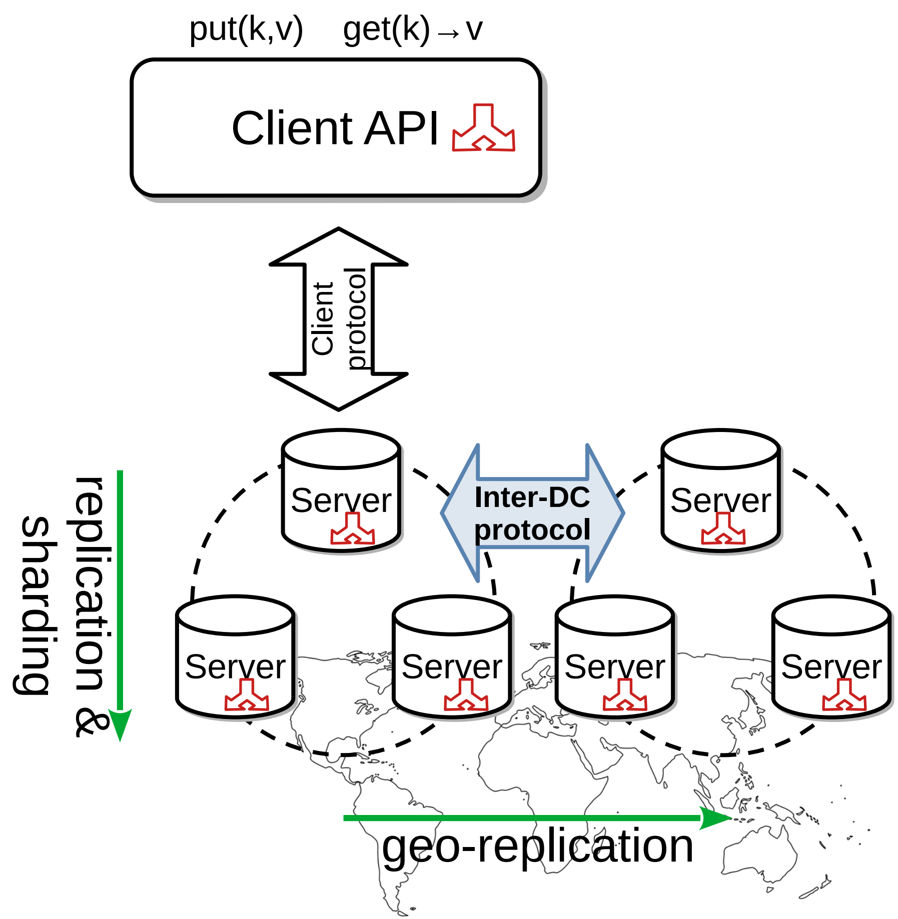
Fully decentralized

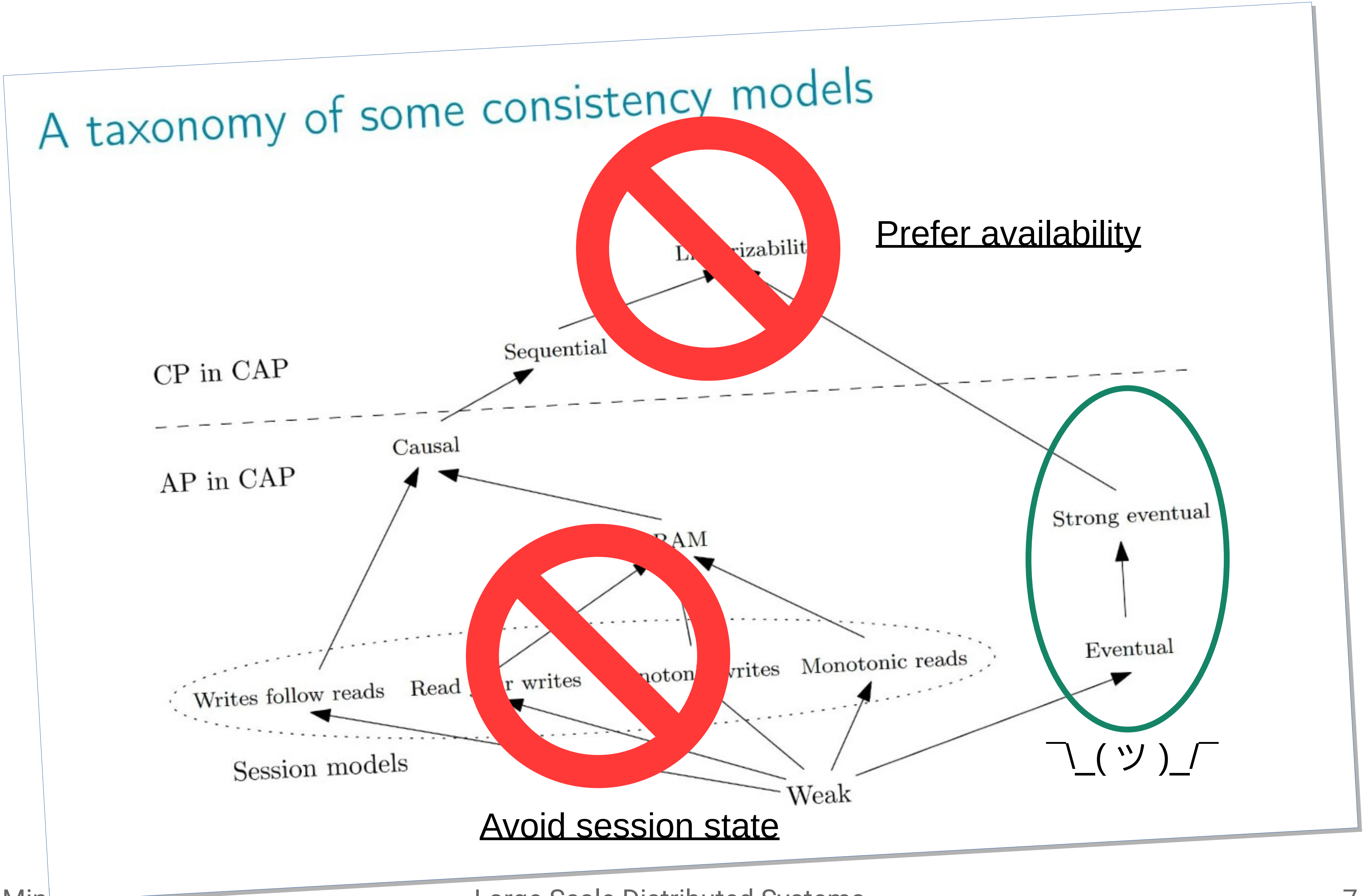
- Consistent hashing
- Epidemic dissemination
- Examples:
  - Dynamo (not DynamoDB!) aws
  - Apache CASSANDRA



Geo-replication

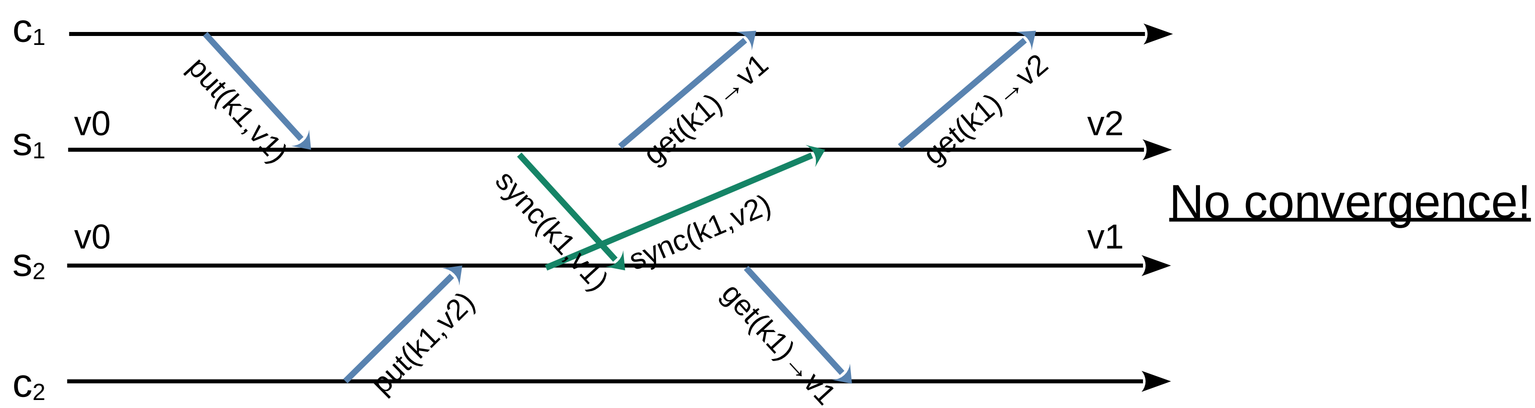
- Multi-data center replication
- <u>Main challenge</u>: How to shield clients from Inter-DC protocol?
  - Latency
  - Availability




A taxonomy of some consistency models
Prefer availability
Sequential
CP in CAP
Causal
AP in CAP
Strong eventual
Eventual
Writes follow reads
Read
Monotonic reads
Session models
Weak
¯_(ツ)_/¯
Avoid session state

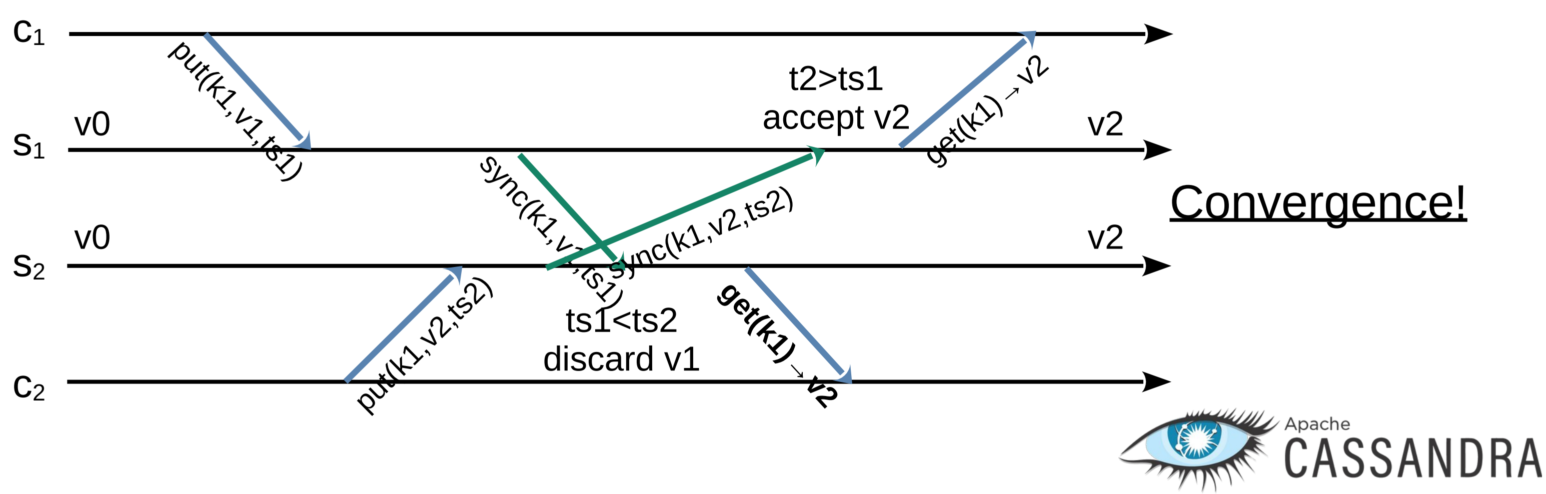
Local writes

- Writes are issued to one or a few servers in the local data center
- Writes are asynchronously propagated to other data centers



Convergence

- Simple reconciliation rule: Last Writer Wins (LWW)
  - Attach a timestamp to each data item
  - On conflict, keep the item with the latest timestamp



Convergence

- Generating the timestamp:
  - Physical: when the client is a Web application and the state partitioned by user
  - Logical?
- LWW Register is actually a State-based CRDT… and KVS servers could support a variety of CRDTs
  - Limits the range of possible data structures and operations
  - Pushes complexity / policy / computation into the server
  - Not worth it if divergence is rare (client affinity + network stability)

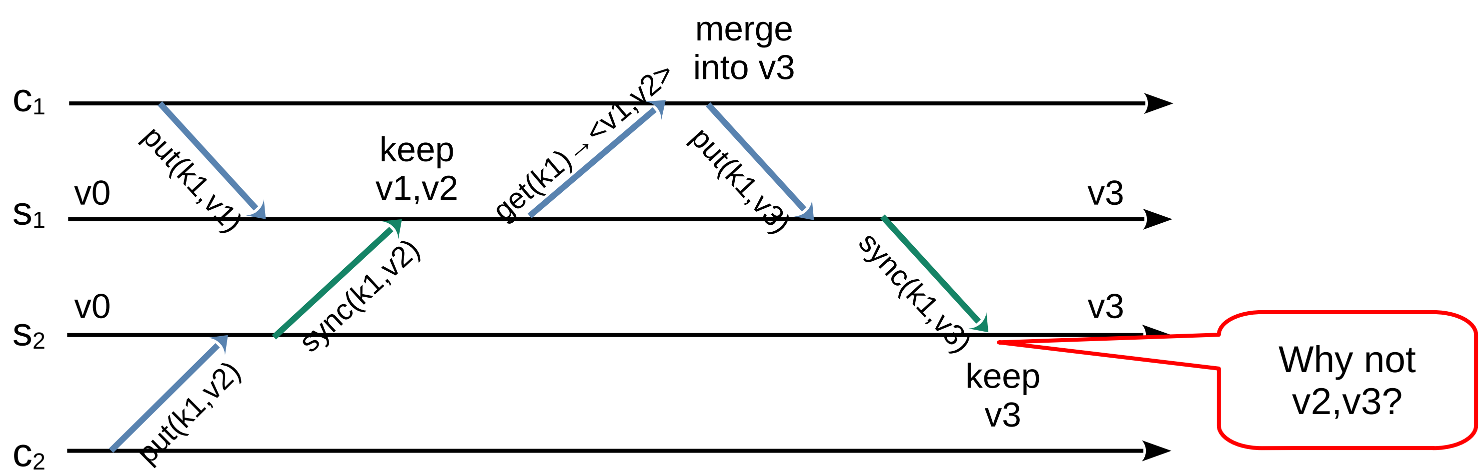
Convergence

- Idea: Delegate reconciliation to the client application
  - Simple LWW (Cassandra)
  - Other State-based CRDTs (Cure)
  - Custom code (Bayou)

- Servers cannot callback into the client application
- How to achieve this?

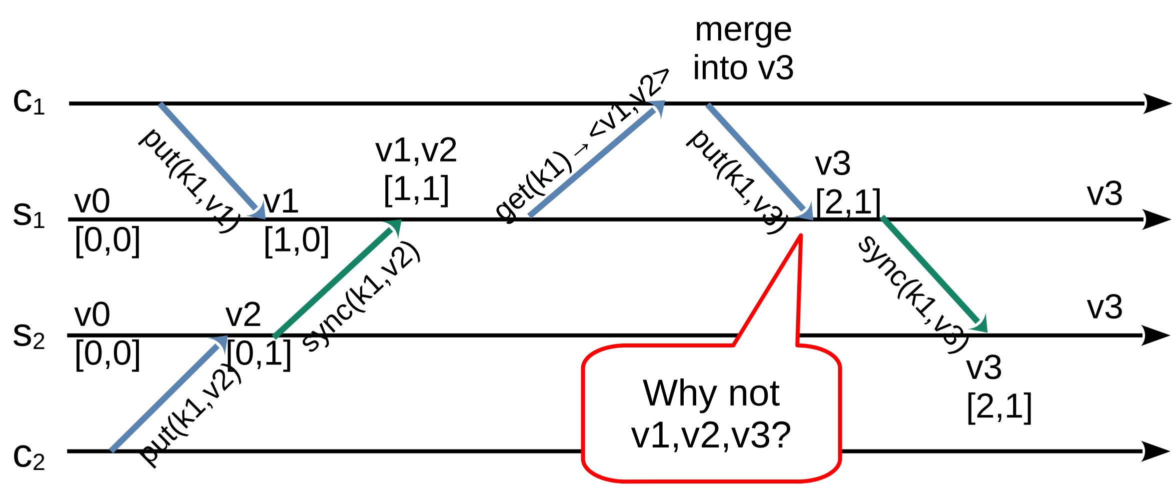
Convergence

- Keep list of conflicting values in each key and return them to the application: get(k)→<v1,...vn>
- Let the application merge them and replace the list with a single value



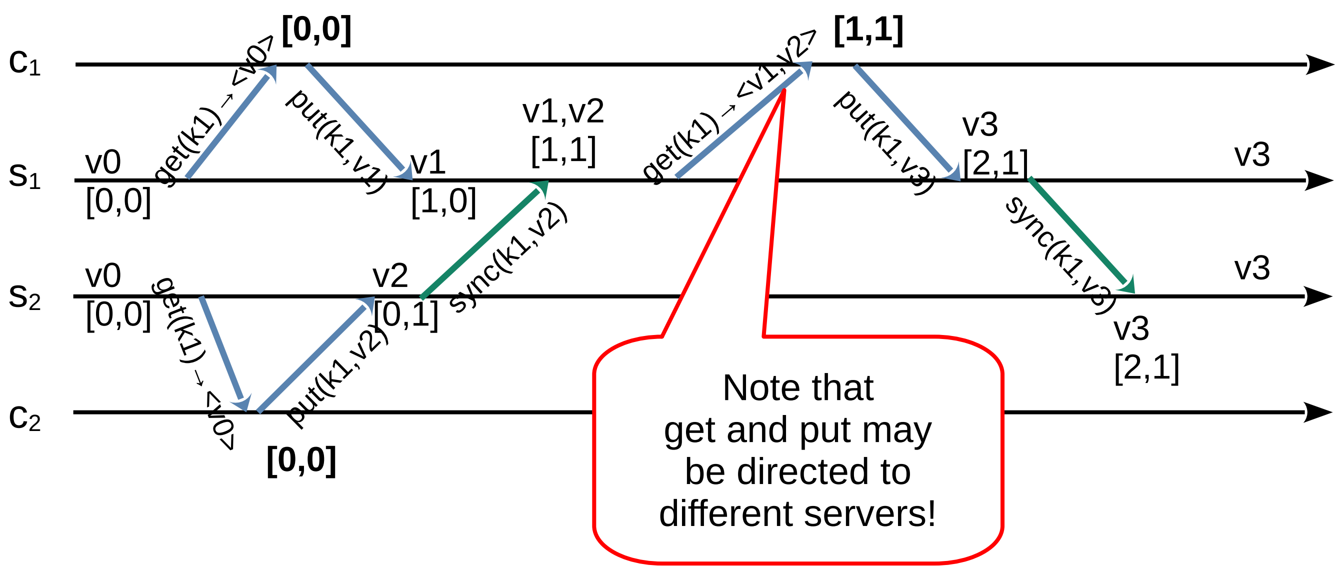
Ordering propagation requests

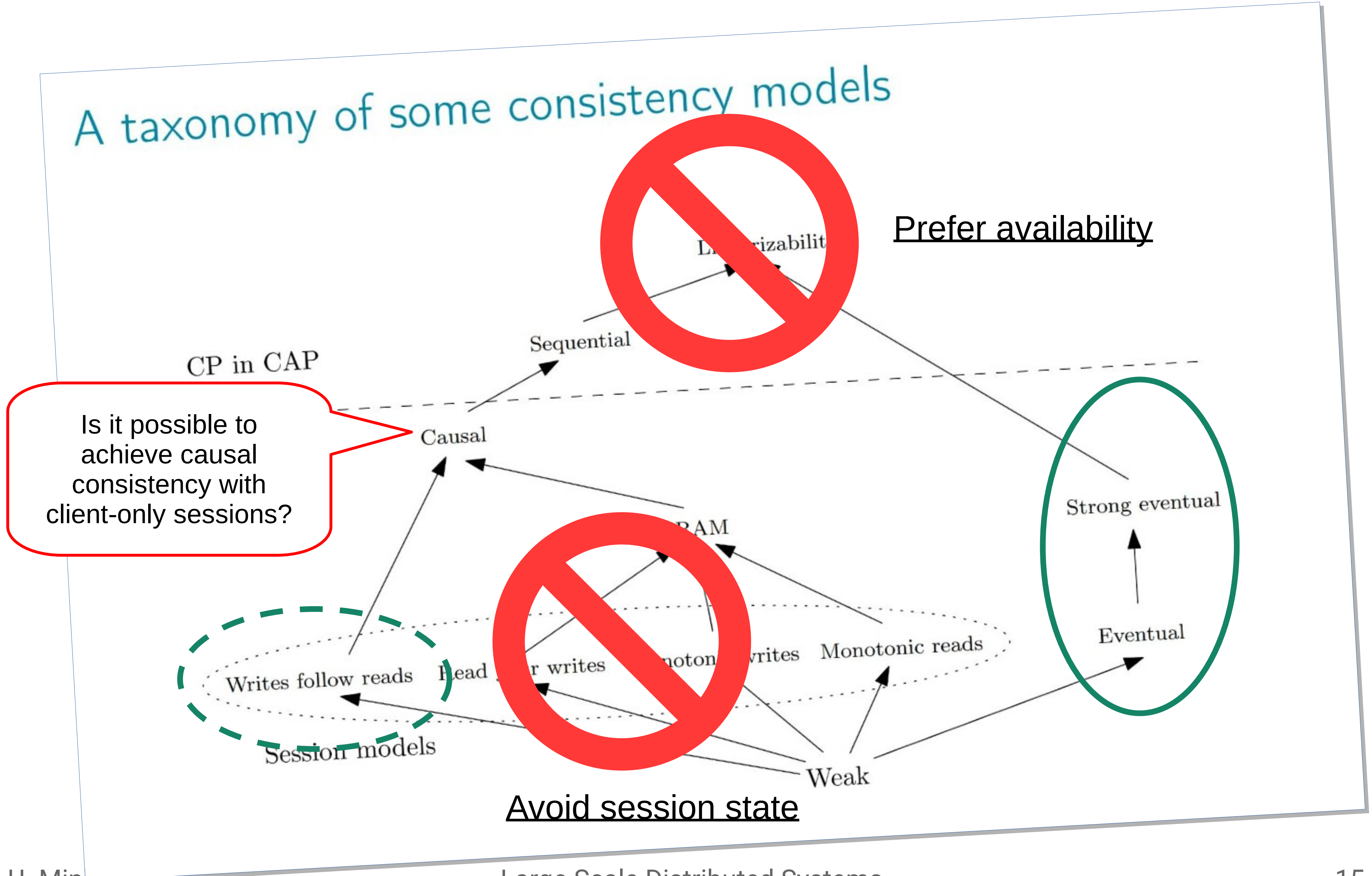
- Keep a vector timestamp with each data item
  - An element for each server
- When merging concurrent timestamps, keep distinct values



Ordering write requests

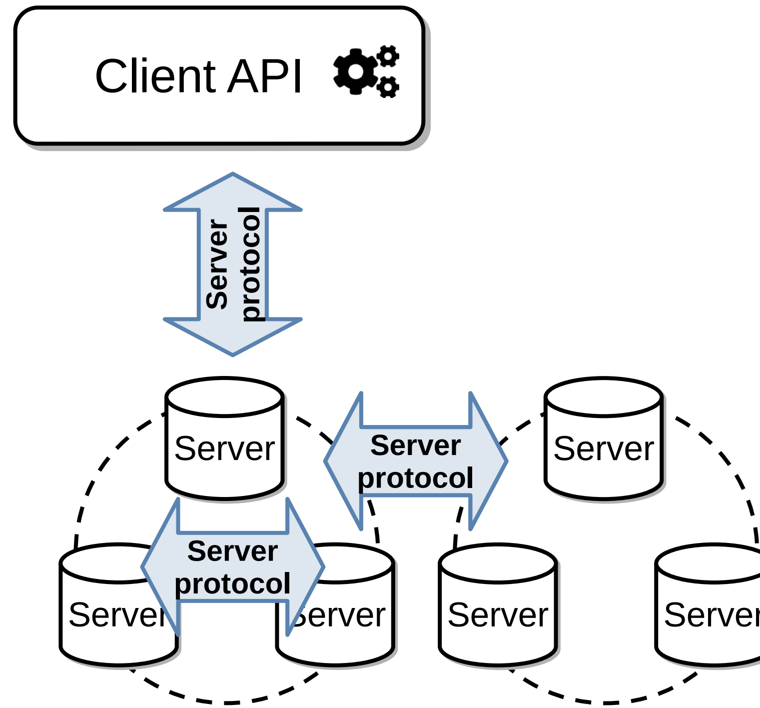
- Client requests are anonymous: How to tell what the client has read before?
  - Force the client to read before writing
  - Make it keep track of context




A taxonomy of some consistency models
Prefer availability
Linearizability
Sequential
CP in CAP
Is it possible to achieve causal consistency with client-only sessions?
Causal
Strong eventual
PRAM
Eventual
Writes follow reads
Read
writes
Monotonic reads
Session models
Weak
Avoid session state

Causal KVS (COPS)

- Client API:
  - createCtx()→ctx
  - get(k,ctx)→v
  - put(k,v,ctx)
  - deleteCtx(ctx)
- Servers keep a scalar timestamp for each (k,v) item (including server id in lower bits)
- Server protocol:
  - get(k)→(v,vers)
  - put_after(k,v,deps,vers=0)→vers
  - dep_check(k,vers)



get operation is trivial

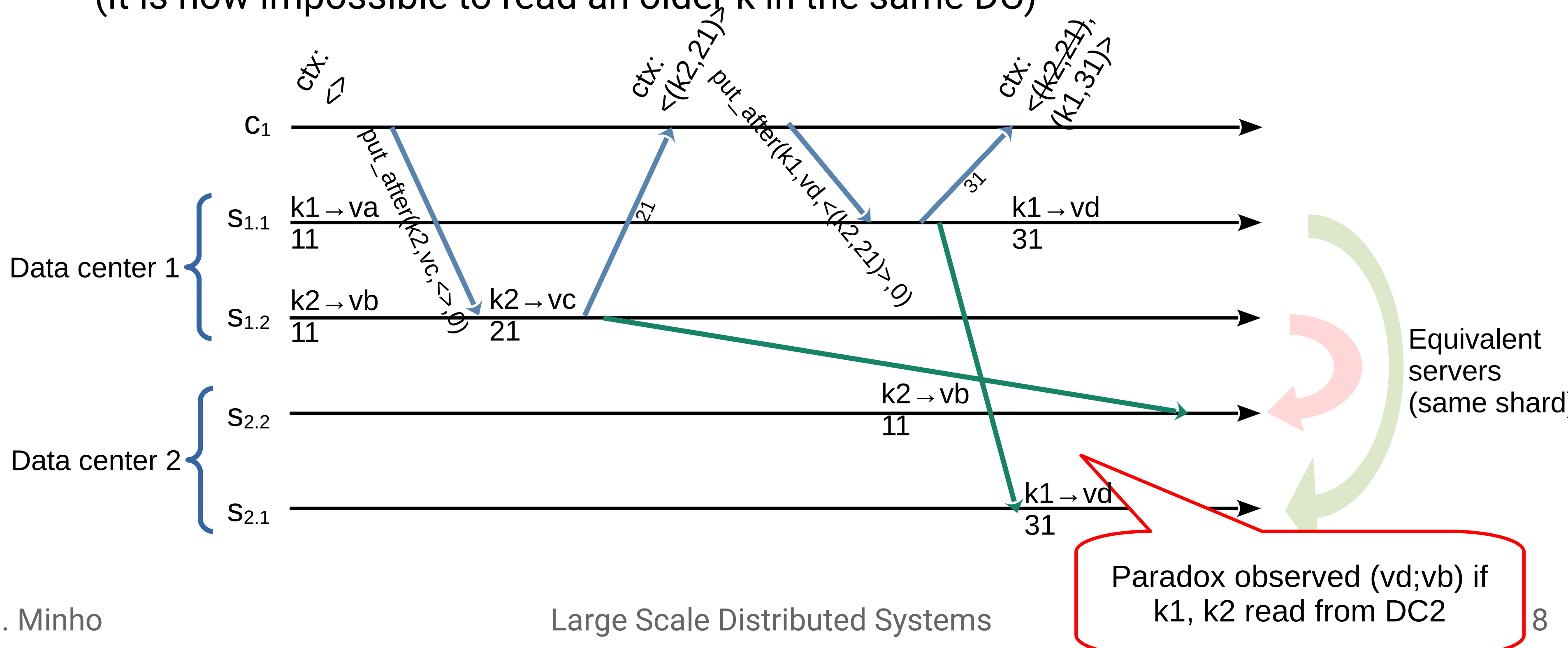
Managing client context

- Client context needs to keep nearest dependencies
- Simple (approximate) strategy:
  - After a put(k,...)→vers operation:
    - Clear all dependencies
    - Insert (k,vers)
  - After a get(k,...)→ (v,vers) operation:
    - If (k,...) in context, update to (k,vers)
    - If not, insert (k,vers)

(It is approximate because it does not recognize dependencies between separately read keys and keeps them both)

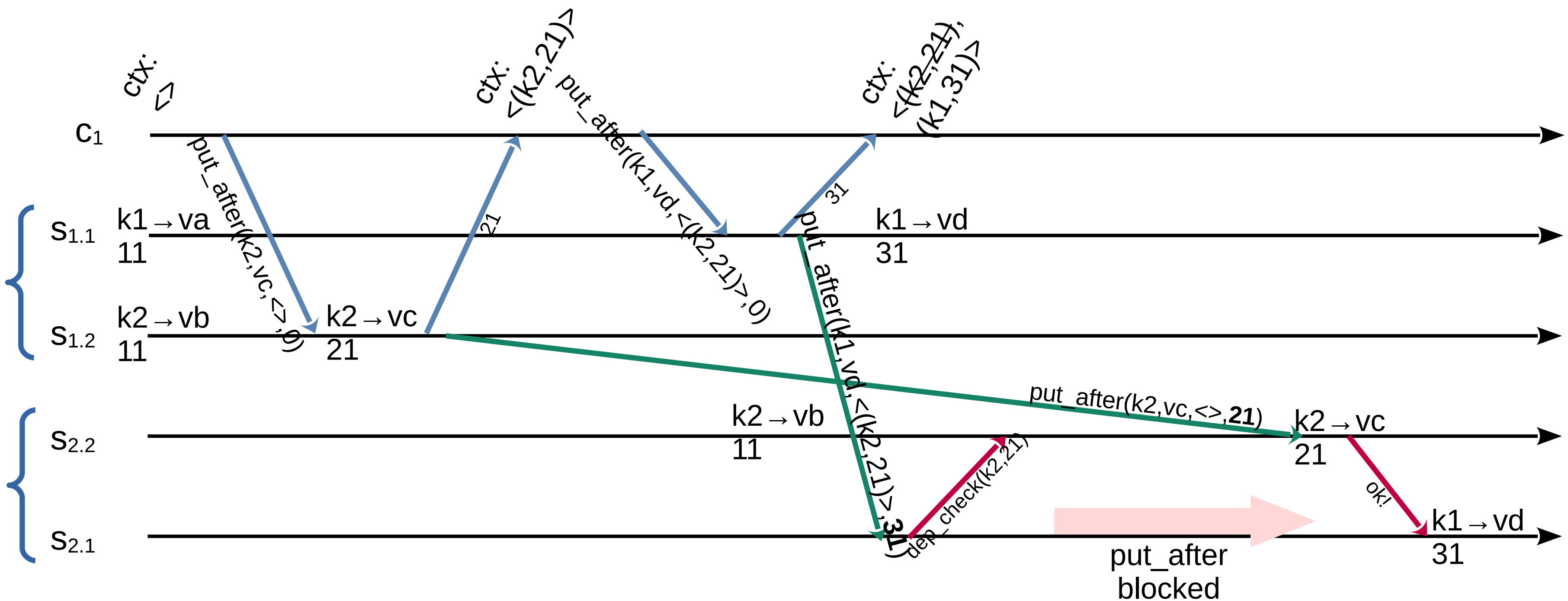
Write operation

- Synchronous write to server in local data center:
  - append dependencies in local context to request
  - wait for write to commit before returning local clock
    (it is now impossible to read an older k in the same DC)



Write propagation

- Propagation is not ordered, as it is potentially from different data centers
- A value is committed remotely only after waiting for dependencies with the dep_check operation of corresponding servers

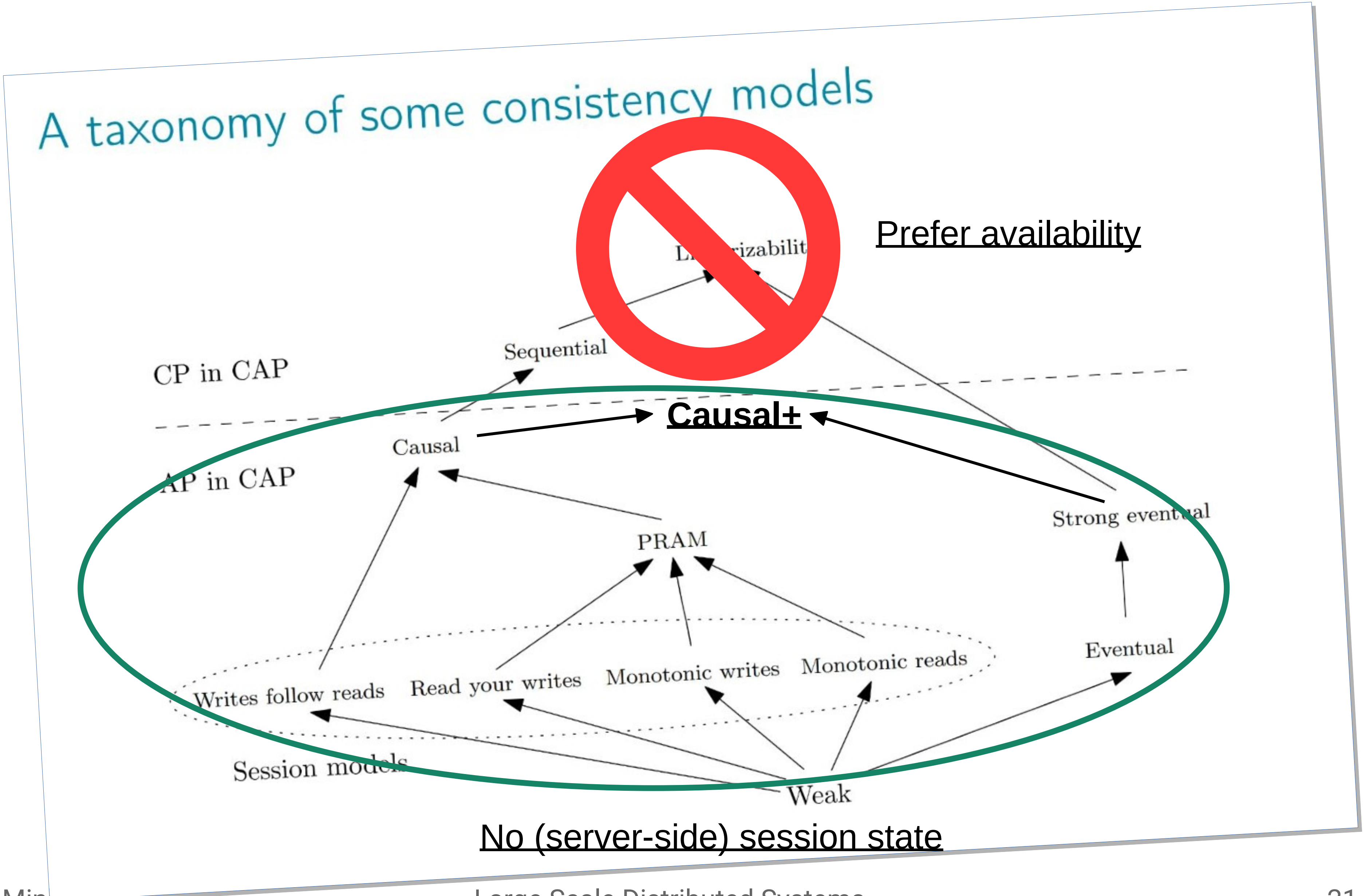


Consistency and convergence

- WFR: reads kept in local context and used in write
- RYW:
- MR:
- MW:

synchronous write and client affinity

- Strong Eventual:
  - Logical timestamps including global server mean that there are no ties between concurrent updates to same item
  - <u>Last Writer Wins</u> when propagating to remote DC


A taxonomy of some consistency models
Prefer availability
Linearizability
Sequential
CP in CAP
Causal+
Causal
AP in CAP
PRAM
Strong eventual
Eventual
Writes follow reads
Read your writes
Monotonic writes
Monotonic reads
Session models
Weak
No (server-side) session state

References

- G. DeCandia et al., “**Dynamo: Amazon’s highly available key-value store**,” Oper. Syst. Rev., vol. 41, no. 6, pp. 205–220, Oct. 2007 https://doi.org/10.1145/1323293.1294281
- W. Lloyd, M. J. Freedman, M. Kaminsky, and D. G. Andersen, “**Don’t settle for eventual: Scalable causal consistency for wide-area storage with COPS**,” in Proc. 23rd ACM Symposium on Operating Systems Principles, 2011 https://www.cs.cmu.edu/~dga/papers/cops-sosp2011.pdf