CS244b - COPS

- Logical clocks
- Causal consistency
- Get transactions
ALPS

CAP theorem

Consistency
Availability
Partition-tolerance
Linearizability > **Sequential** > **Causal**+ > **Causal** > **FIFO**
> **Per-Key Sequential** > **Eventual**

**Strict serializability**

Linearizability - strict serializability 1 key
Seq. Cst. - 1. Program Order, 2. Write atomicity
A posts photo
B comments on photo
C reads comment

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

`sync()` - flushes all pending writes
Writing B

everyone sees what you saw (A) before seeing B

1. Apply writes to key in same order or converge on same result

2.

Clusters, shards
complete get/put just talking to local
\[ \langle k, v, ts, id \rangle \] (strawman)

- log server per cluster
- keep tuple w. highest ts.
  break ties by id
- at each cluster: 
  \[ ts_{\text{min, lowest of } ts_i} \]
Lamport clocks

Calculate timestamps s.t.
if A caused B, the B.ts > A.ts
happens before \( \rightarrow \), \( A \rightarrow B \) iff
- A precedes B in same thread (\( A \rightarrow B \))
- A "puts into" B (\( A \leftarrow B \)), or
- \( A \rightarrow C \rightarrow B \)

\( T_i \) local op \( T_i \leftarrow \max(\text{realtime}, T_i+1) \)
receive m \( T_i \leftarrow \max(T_i, m.ts) \)
\[ vts = \langle 1-t_1, 2-t_s, \ldots \rangle \]

\[ \langle k, val, vts, id \rangle \]

\[ \forall i \in t_i \geq vts[i] \]

Shard have local serve for shard talk to equivalent nodes

Keep track of dependencies per op.

E.g. ctx arg
Figure 2: Graph showing the causal relationship between oper-
Causal vs. Causal+

- Reconciliation associative & commutative
- Causal order: see only increasing vers.
  see versions at least as recent as putters
- Causal+: can't see conflicting vers.

A₁ adds a photo

A₂ adds another photo

B comments on A₁

C sees comment

A₃ merges A₁ + A₂
① set (ACC)  \lor  ② set (photo-list)
change ACC, post photo

① set (photos)  \lor  ② set (ACC)
delete photo, change ACC

COPS GT
1. `ctx_id ← createContext()`
2. `bool ← deleteContext(ctx_id)`
3. `bool ← put (key, value, ctx_id)`
4. `value ← get (key, ctx_id) [In COPS]`
   or
4. `<values> ← get_trans ((keys), ctx_id) [In COPS-GT]`
Figure 6: A sample graph of causal dependencies for a
(bool, vers) < - put_after(k, val, [deps], nearest, vers=0)

bool < - dep_check(key, vers)

(val, vers, deps) < - get_by_version(k, vers=LATEST)

\text{put}(k, v, ctx)

\text{put\_after}(v, vers=0, nearest from ctx)

Local server assigns vers, commit immediately
Local server forwards to other clusters

\text{put\_after}(w, vers)

equiv. nodes dep\_check — never depend

get(k, ctx) in lib.

\text{values get\_by\_vers}(k, LATEST)

add(k, vers) to dependencies in ctx

in ops\_gt record (k, vers, deps)
# @param keys list of keys
# @param ctx_id context id
# @return values list of values

function get_trans(keys, ctx_id):
    # Get keys in parallel (first round)
    for k in keys
        results[k] = get_by_version(k, LATEST)

    # Calculate causally correct versions (ccv)
    for k in keys
        ccv[k] = max(ccv[k], results[k].vers)
        for dep in results[k].deps
            if dep.key in keys
                ccv[dep.key] = max(ccv[dep.key], dep.vers)

    # Get needed ccvs in parallel (second round)
    for k in keys
        if ccv[k] > results[k].vers
            results[k] = get_by_version(k, ccv[k])

    # Update the metadata stored in the context
    update_context(results, ctx_id)

    # Return only the values to the client
    return extract_values(results)
A, B, C

A, B, C

A_1 \rightarrow B_2 \rightarrow C_2