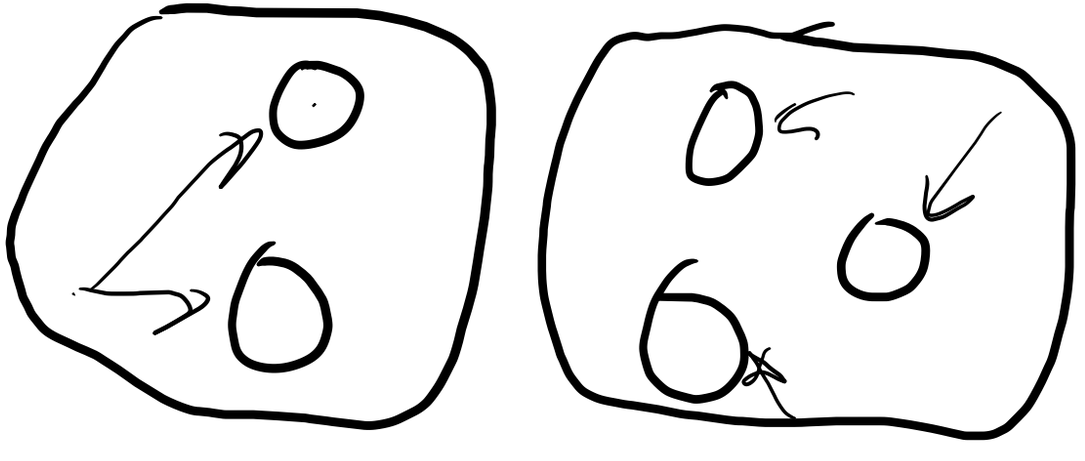
CS244b - COPS

- 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 | 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_i highest ts received from i
  ts_{min} lowest of ts_i

Lamport clocks

Calculate timestamps s.t.

if A caused B, the B.ts > A.ts

happens before $\rightsquigarrow$, $A \rightsquigarrow B$ iff

- A precedes B in same thread ($A \to B$)
- A "puts into" B ($A \mapsto B$), or
- $A \rightsquigarrow \subset \rightsquigarrow B$

T_i : local op $T_i \leftarrow \max(\text{realtime}, T_i + 1)$

receive m $T_i \leftarrow \max(T_i, m.ts)$

$vts = \langle 1\text{-}ts_1, 2\text{-}ts_2, \ldots \rangle$

$\langle k, val, vts, id \rangle$

$\forall_i \; t_i > vts[i]$

Shard have local serve for shard
talk to equivalent nodes

keep track of dependencies per op.
e.g. ctx arg.

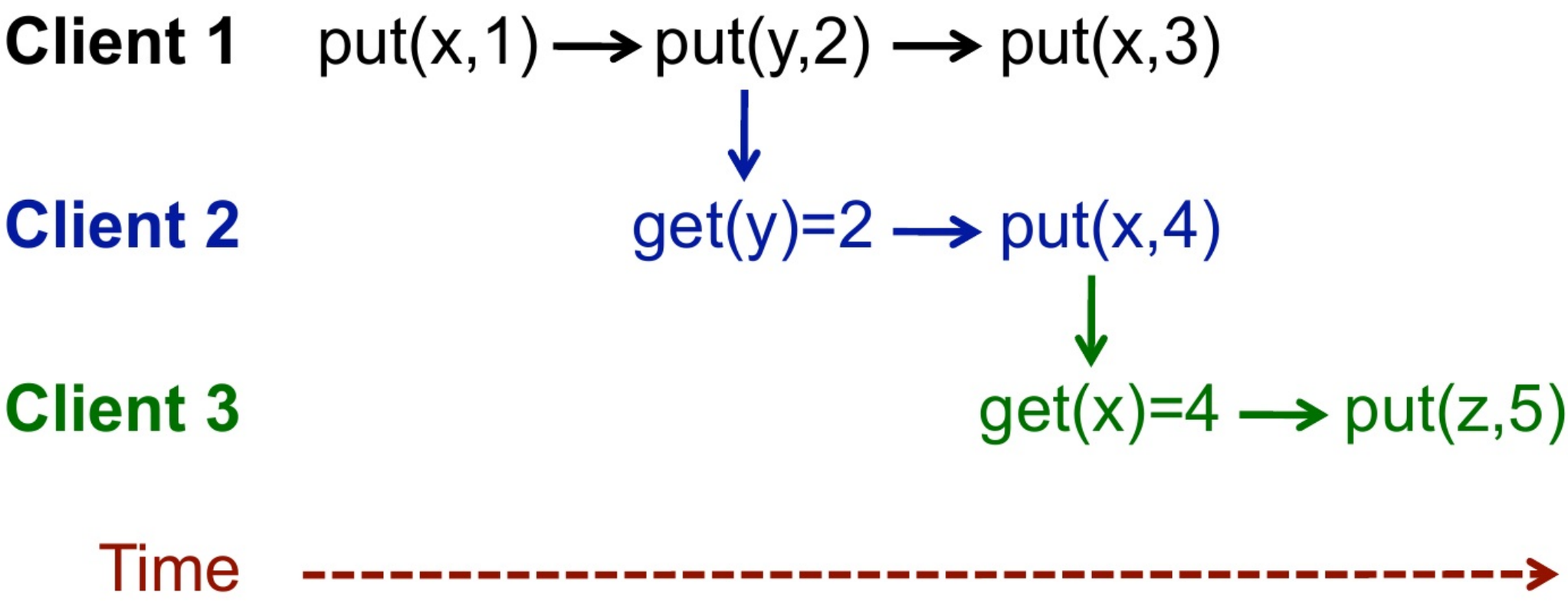
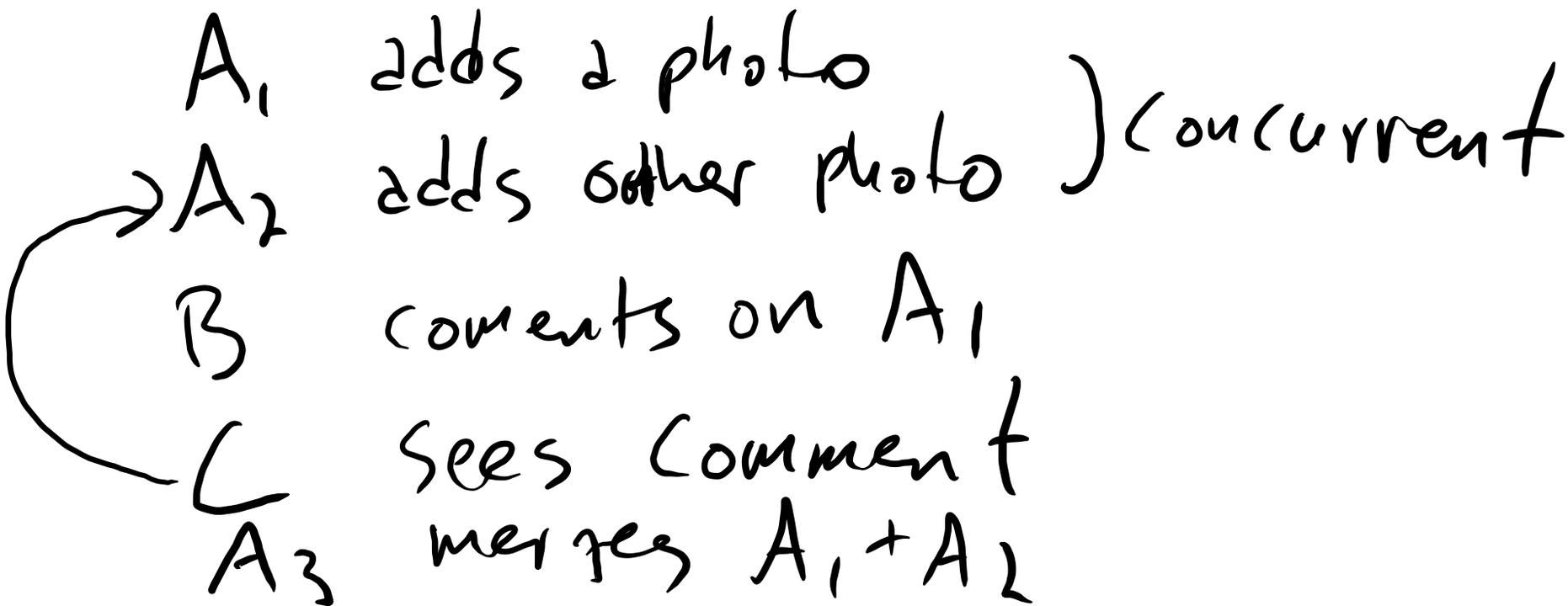
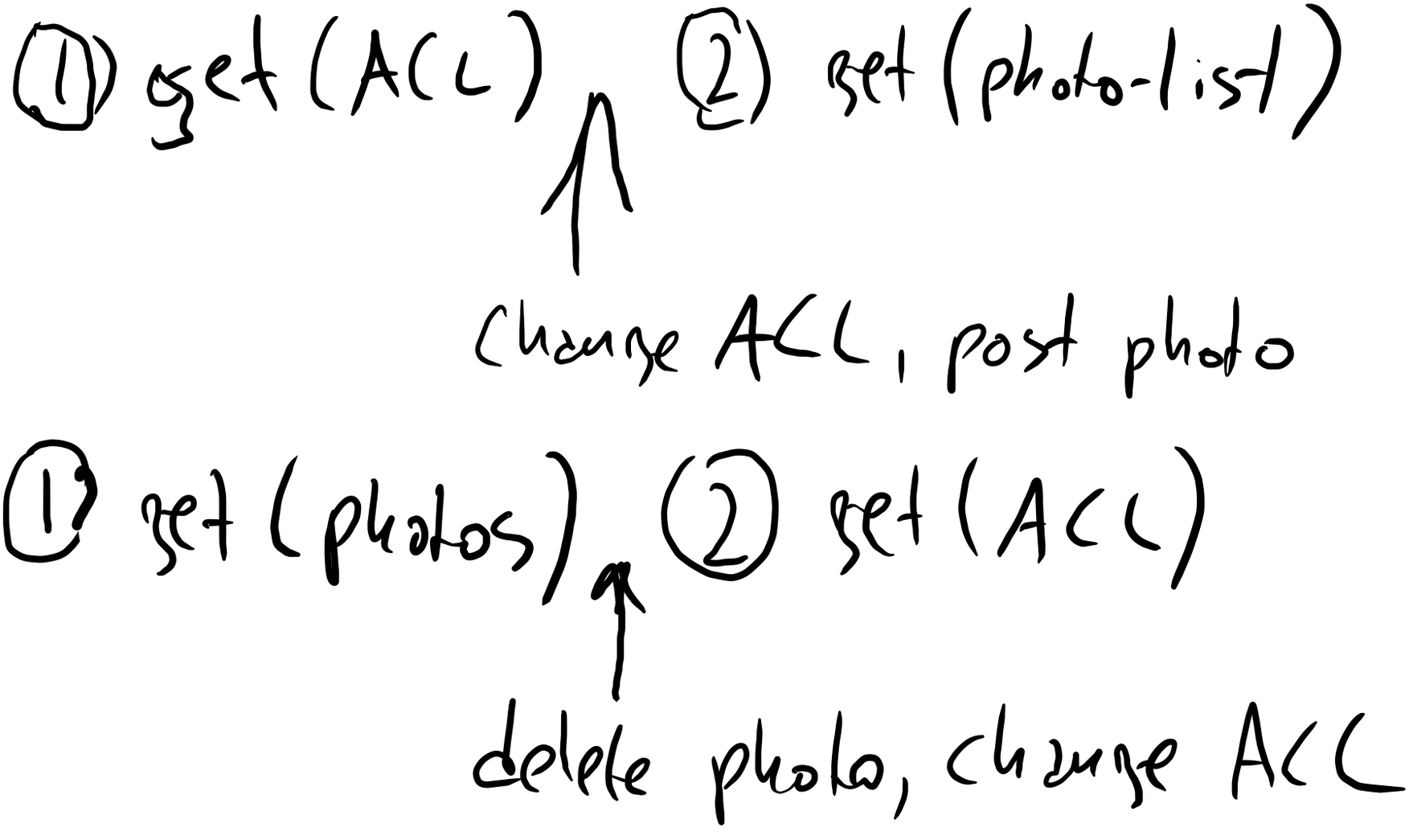


Figure 2: Graph showing the causal relationship between oper-

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.



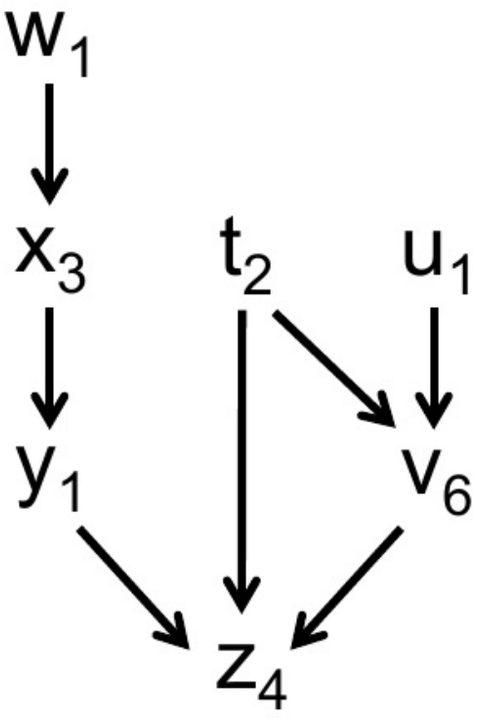

① get(ACL)
② get(photo-list)
change ACL, post photo
① get(photos)
② get(ACL)
delete photo, change ACL

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]

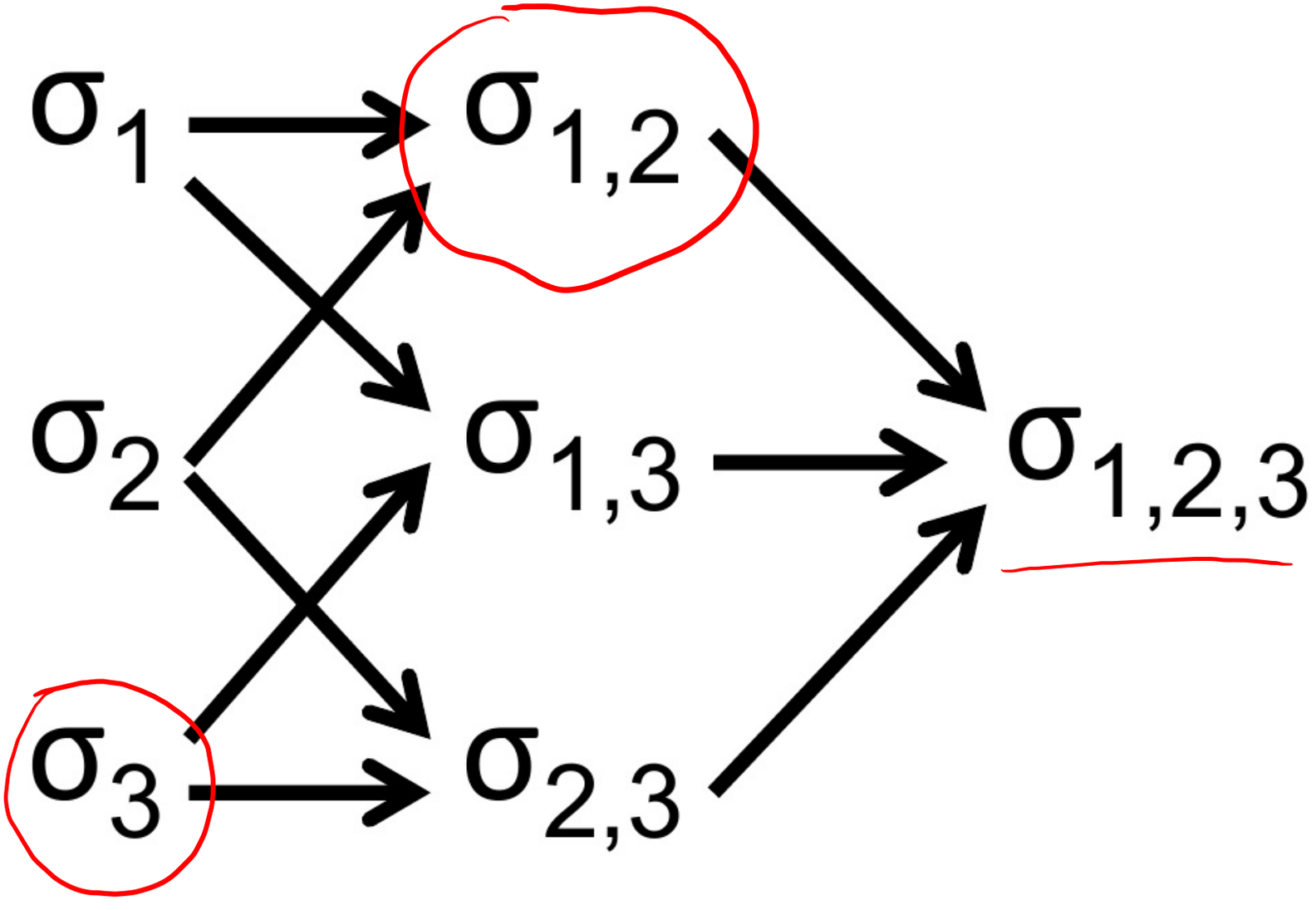
or

4. ⟨*values*⟩ ← `get_trans` (⟨*keys*⟩, *ctx_id*) [In COPS-GT]



Val	Nearest Deps	All Deps
t_2	-	-
u_1	-	-
v_6	t_2, u_1	t_2, u_1
w_1	-	-
x_3	w_1	w_1
y_1	x_3	x_3, w_1
z_4	y_1, v_6	$t_2, u_1, v_6, w_1, x_3, y_1$

Figure 6: A sample graph of causal dependencies for a


σ_1
σ_2
σ_3
$\sigma_{1,2}$
$\sigma_{1,3}$
$\sigma_{2,3}$
$\sigma_{1,2,3}$

(bool,vers) <- put_after(k,val,[deps],nearest,vers=0)

bool <- dep_check(key,vers)

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

put(k,v,ctx)

put_after vers=0, nearest from ctx

Local server assign vers, commit immediately

Local server forwards to other clusters

put_after w. vers

equiv. nodes dep_check —

never-depend

get(k,ctx) in lib.

v,vers get_by_vers(k, LATEST)

add(k,vers) to dependencies in ctx

in COPS-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_1 B_1 C_1 $\qquad$ $A_2 \mapsto B_2 \mapsto C_2$

A_1 B_1 C_2

~~A_1 B_2 C_2~~