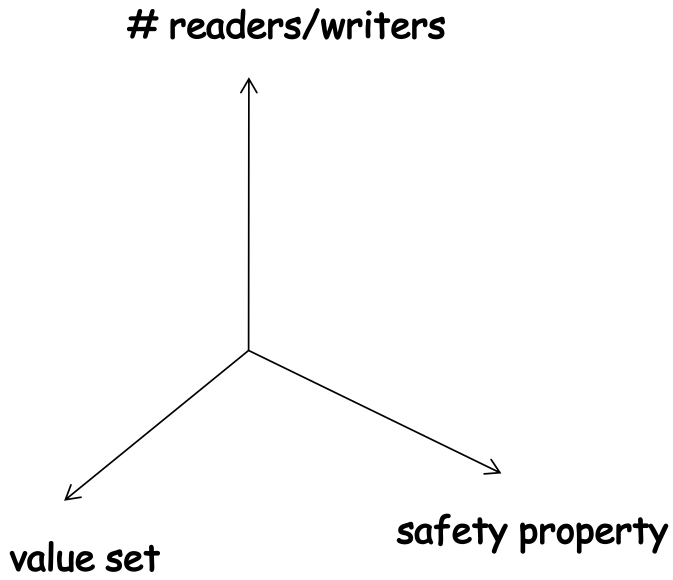
Atomic snapshot

SLR206, P2

The space of registers

- Nb of writers and readers: from 1W1R to NWNR
- Size of the value set: from binary to multi-valued
- Safety properties: safe, regular, atomic



All registers are (computationally) equivalent!

Transformations

From 1W1R binary safe to 1WNR multi-valued atomic

I. From safe to regular (1W1R)
II. From one-reader to multiple-reader (regular binary or multi-valued)
III. From binary to multi-valued (1WNR regular)
IV. From regular to atomic (1W1R)
V. From 1W1R to 1WNR (multi-valued atomic)
VI. From 1WNR to NWNR (multi-valued atomic)
VII. From safe bit to atomic bit (optimal, coming later)

This class

- Atomic snapshot: reading multiple locations atomically
  - ✓Write to one, read *all*

Atomic snapshot: sequential specification

- Each process p_i is provided with operations:
  ✓update$_i$(v), returns ok
  ✓snapshot$_i$(), returns $[v_1,\ldots,v_N]$

- In a sequential execution:
  For each $[v_1,\ldots,v_N]$ returned by snapshot$_i$(),
  v_j ($j=1,\ldots,N$) is the argument of the last update$_j$(.)
  (or the initial value if no such update)

Snapshot for free?

Code for process p_i:

initially:

shared 1WNR *atomic* register $R_i := 0$

upon snapshot()

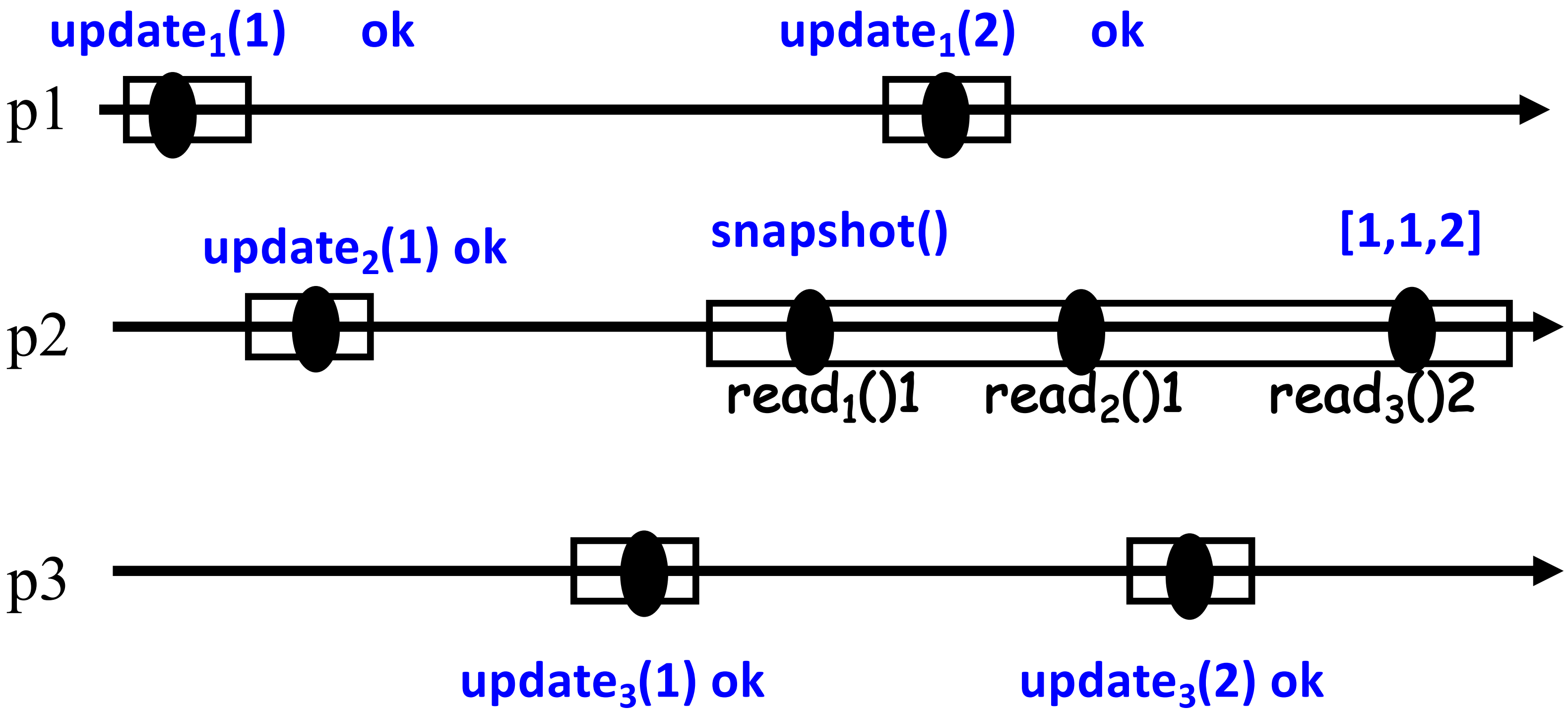
$[x_1,\ldots,x_N] := \text{scan}(R_1,\ldots,R_N)$ /*read $R_1,\ldots R_N$*/

return $[x_1,\ldots,x_N]$

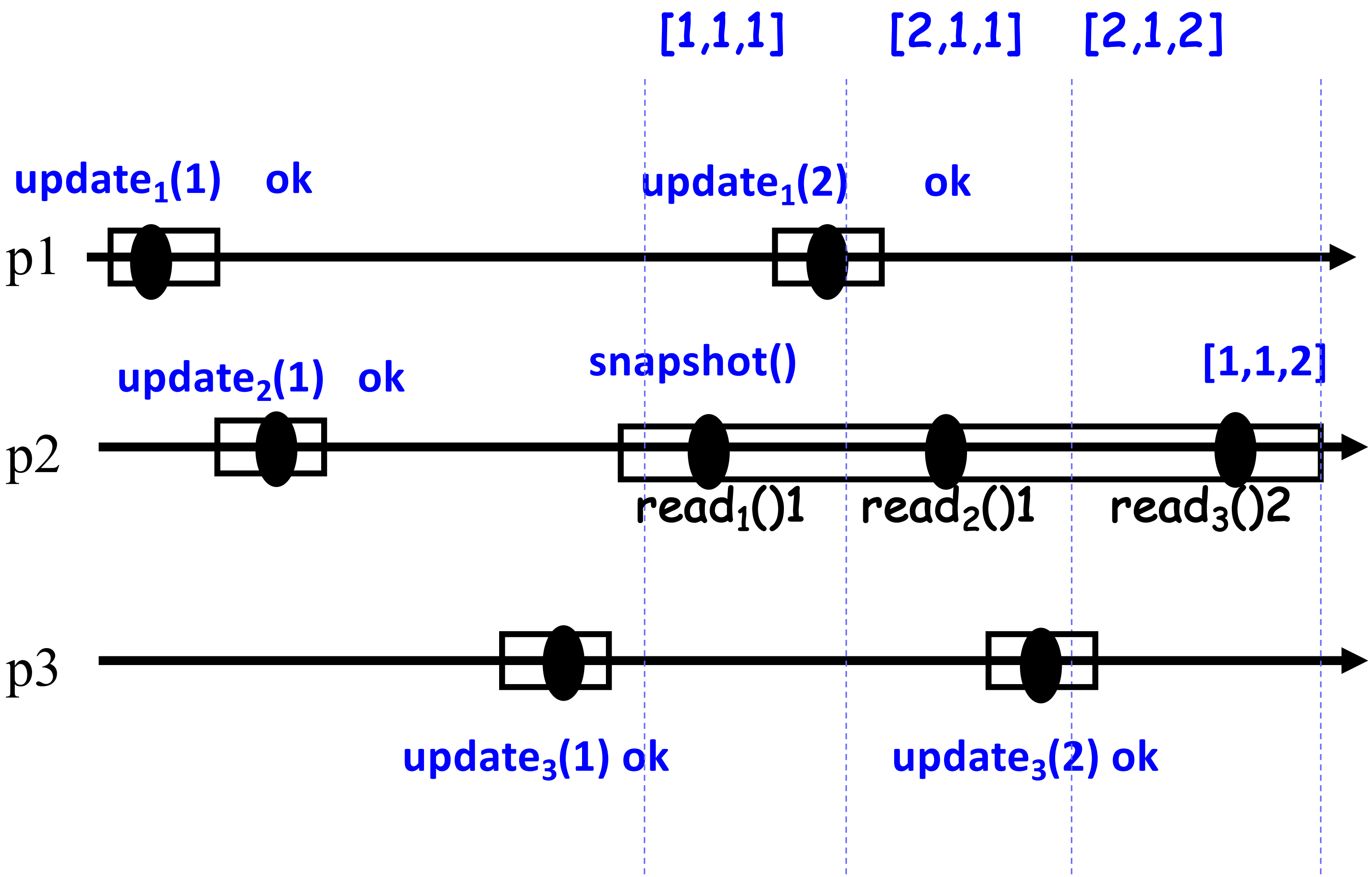
upon update$_i$(v)

R_i.write(v)

Snapshot for free?



Snapshot for free?



- What about 2 processes?

- What about lock-free snapshots?
  - ✓At least one correct process makes progress (completes infinitely many operations)

Lock-free snapshot

Code for process p_i (all written values, including the initial one, are unique, e.g., equipped with a sequence number)

Initially:

shared 1W1R atomic register $R_i := 0$

upon snapshot()

$[x_1,\ldots,x_N] := scan(R_1,\ldots,R_N)$

repeat

$[y_1,\ldots,y_N] := [x_1,\ldots,x_N]$

$[x_1,\ldots,x_N] := scan(R_1,\ldots,R_N)$

until $[y_1,\ldots,y_N] = [x_1,\ldots,x_N]$

return $[x_1,\ldots,x_N]$

upon update$_i$(v)

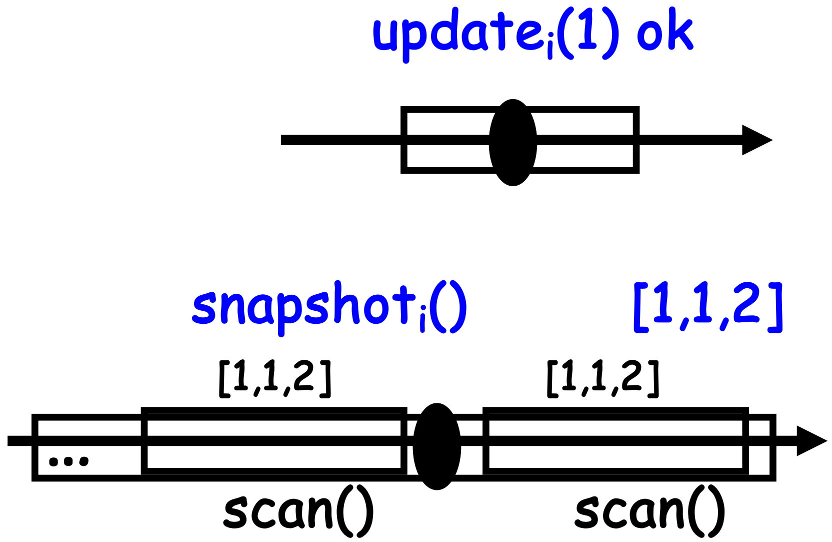
R_i.write(v)

Linearization

Assign a linearization point to each operation

- $update_i(v)$
  - ✓ R_i.write(v) if present
  - ✓ Otherwise remove the op
- $snapshot_i()$
  - ✓ if complete – any point between identical scans
  - ✓ Otherwise remove the op

Build a sequential history S in the order of linearization points

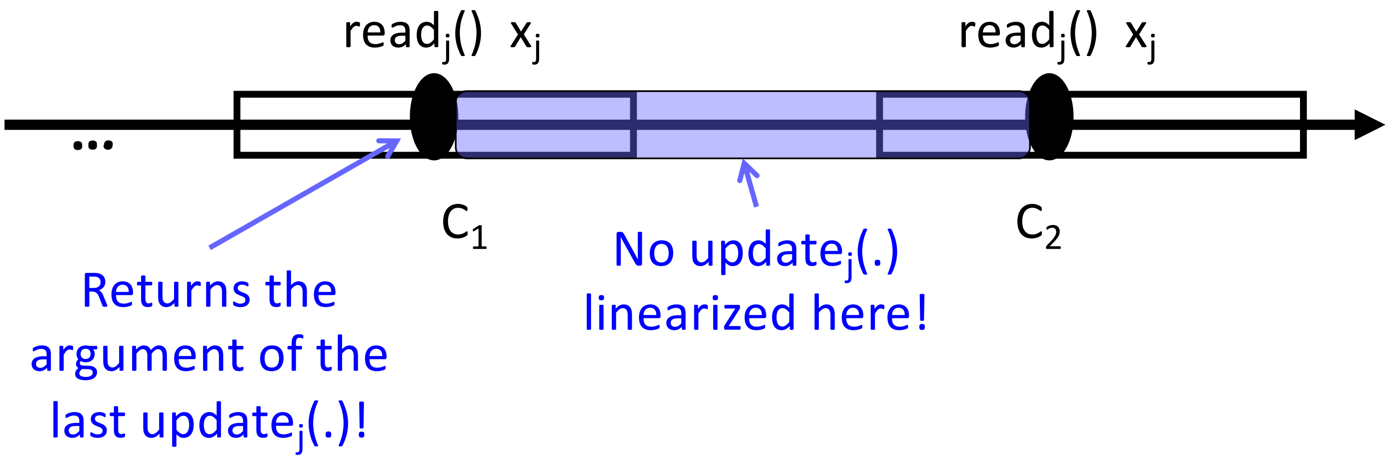


Correctness: linearizability

S is legal: every $\text{snapshot}_i()$ returns the last written value for every p_j

Suppose not: $\text{snapshot}_i()$ returns $[x_1,\ldots,x_N]$ and some x_j is not the the argument of the last $\text{update}_j(v)$ in S preceding $\text{snapshot}_i()$

Let C_1 and C_2 be two scans that returned $[x_1,\ldots,x_N]$



Correctness: lock-freedom

An $update_i()$ operation is wait-free (returns in a finite number of steps)

Suppose process p_i executing $snapshot_i()$ eventually runs in isolation (no process takes steps concurrently)

- All scans received by p_i are distinct
- At least one process performs an update between
- There are only finitely many processes => at least one process executes infinitely many updates

What if base registers are regular?

General case: helping?

What if an update interferes with a snapshot?

- Make the update do the work!

WF

upon snapshot()

$[x_1,\ldots,x_N] := \mathrm{scan}(R_1,\ldots, R_N)$
$[y_1,\ldots,y_N] := \mathrm{scan}(R_1,\ldots,R_N)$
if $[y_1,\ldots,y_N] = [x_1,\ldots,x_N]$ then
  return $[x_1,\ldots,x_N]$
else
  let j be such that
    $x_j \neq y_j$ and $y_j=(u,U)$
    return U

upon update$_i$(v)

S := snapshot()
R_i.write(v,S)

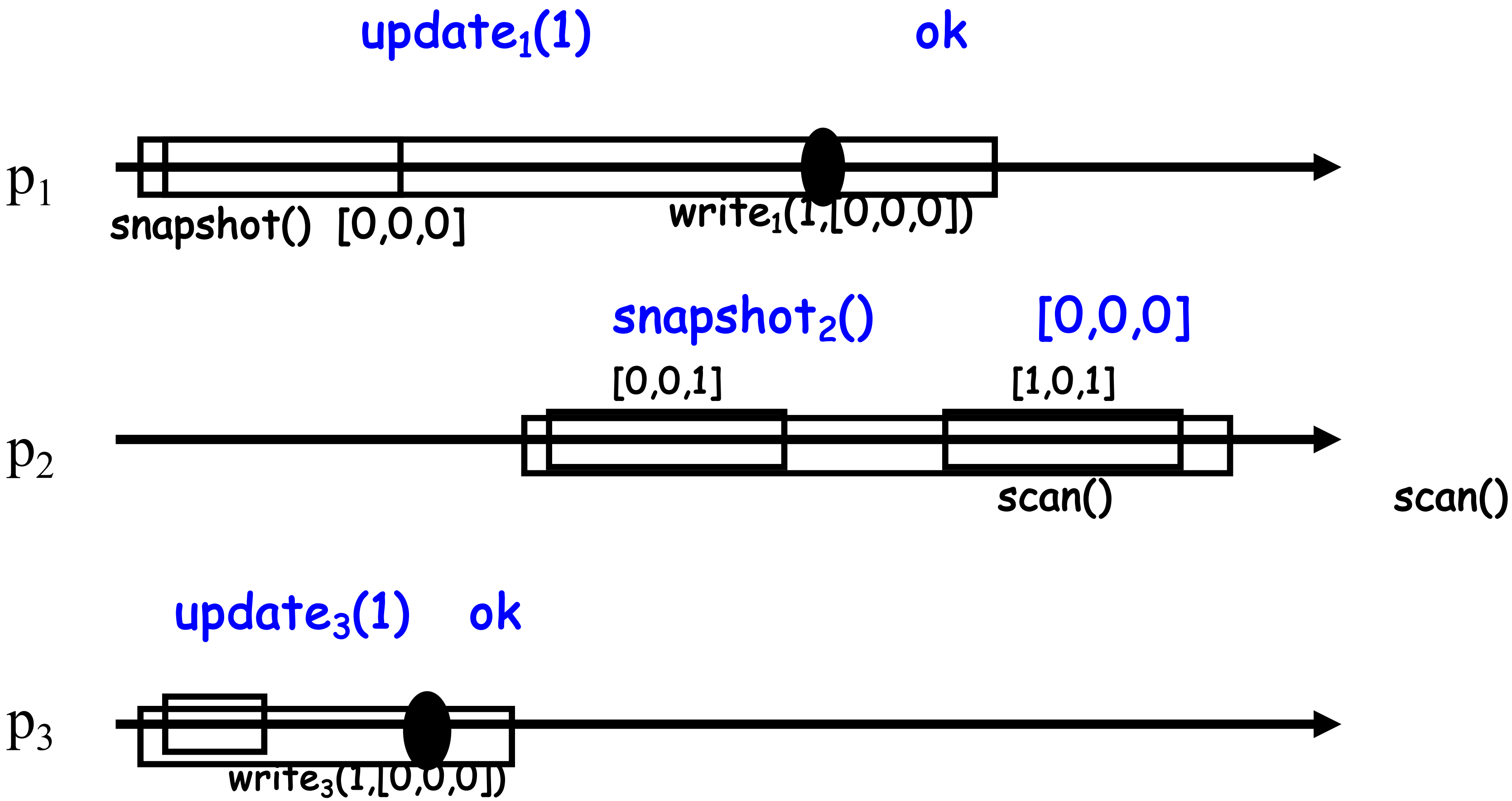
目的为了直接使用上次 update 的 snapshot.

If two scans differ - some update succeeded!

Would this work?

问题是不确定上一次 update 的 snapshot 在当前 snapshot 范围内

Not that easy!



General case: wait-free atomic snapshot

upon snapshot()

$[x_1,\ldots,x_N] := scan(R_1,\ldots,R_N)$

while true do

lock free

$[y_1,\ldots,y_N] := [x_1,\ldots,x_N]$

$[x_1,\ldots,x_N] := scan(R_1,\ldots,R_N)$

if $[y_1,\ldots,y_N] = [x_1,\ldots,x_N]$ then

return $[x_1,\ldots,x_N]$

else if $moved_j$ and $x_j \neq y_j$ then

let $x_j = (u,U)$

wait free

return U

for each j: $moved_j := moved_j \lor x_j \neq y_j$

upon update$_i$(v)

S := snapshot()

R_i.write(v,S)

If a process moved twice: its last snapshot is valid!

通过两个判断确定在当前snapshot范围内同一process发生了两次snapshot，那么最新的一次snapshot必发生在当前范围内 验证了snapshot的有效性

Correctness: wait-freedom

Claim 1 Every operation (update or snapshot) returns in $O(N^2)$ steps (bounded wait-freedom)

snapshot: does not return after a scan if a concurrent process moved and no process moved twice

- At most N-1 concurrent processes, thus (pigeonhole), after N scans:
  - ✓Either at least two consecutive identical scans
  - ✓Or some process moved twice!

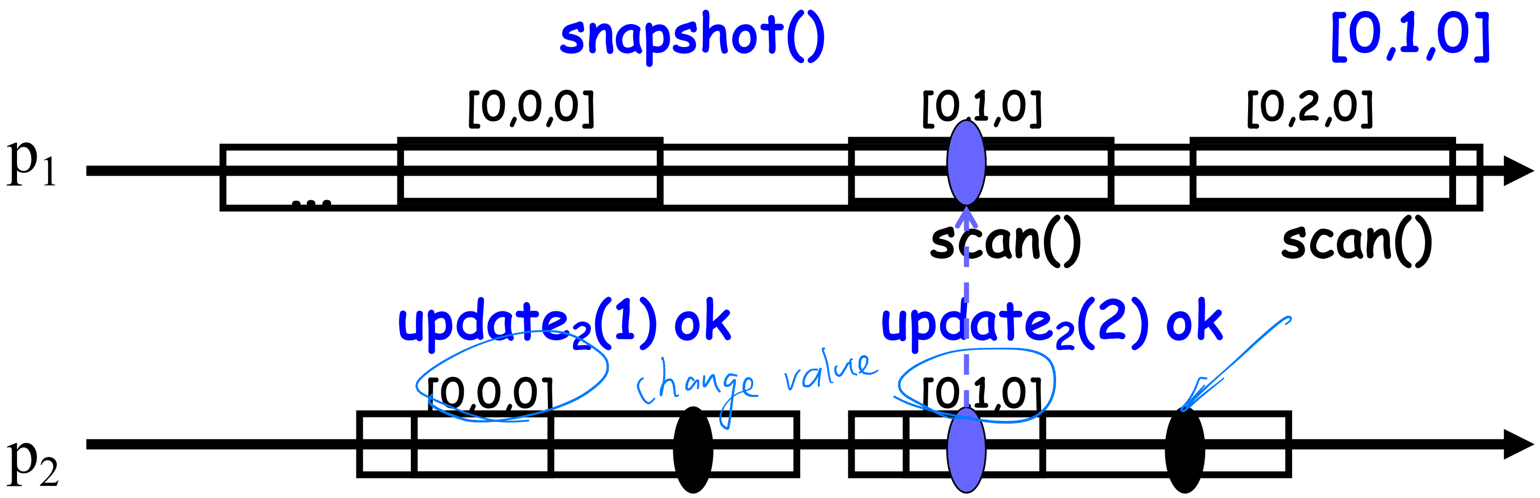
update: snapshot() + one more step

Correctness: linearization points

update$_i$(v): linearize at the R_i.write(v,S)

complete **snapshot()**

- If two identical scans: between the scans
- Otherwise, if returned U of p_j: at the linearization point of p_j's snapshot



The linearization is:

- Legal: every snapshot operation returns the most recent value for each process
- Consistent with the real-time order: each linearization point is within the operation's interval
- Equivalent to the run (locally indistinguishable)

(Full proof in the lecture notes, Chapter 6)

Atomic snapshot: sequential specification

- Each process p_i is provided with operations:
  - ✓$update_i(v)$, returns ok
  - ✓$snapshot_i()$, returns $[v_1,\ldots,v_N]$

- In a sequential execution:

  For each $[v_1,\ldots,v_N]$ returned by $snapshot_i()$,
  v_j $(j=1,\ldots,N)$ is the argument of the last $update_j(.)$
  (or the initial value if no such update)

One-shot atomic snapshot (AS)

Each process p_i:

update$_i$(v_i)

S_i := snapshot()

$S_i = S_i[1],\ldots,S_i[N]$

(one position per process)

Vectors S_i satisfy:

- Self-inclusion: for all i: v_i is in S_i
- Containment: for all i and j: S_i is subset of S_j or S_j is subset of S_i

$v_i \in S_j \Leftrightarrow S_j[i] = v_j$.

bottom

$S_i \subseteq S_j \Leftrightarrow \forall k,\ S_i[k] \neq \bot \Rightarrow S_i[k] = S_j[k]$

Quiz 5: atomic snapshots

1. Prove that one-shot atomic snapshot satisfies self-inclusion and containment:
   - ✓Self-inclusion: for all i: v_i is in S_i
   - ✓Containment: for all i and j: S_i is subset of S_j or S_j is subset of S_i
2. Show that the atomic snapshot is subject to the ABA problem (affecting correctness) in case the written values are not unique