

Why graphs need special memory management

EAGER: completion proves reuse

record_stream tracks cross-stream uses. Event polling proves GPU completion before the allocator recycles storage.

CAPTURE: recorded work has not run

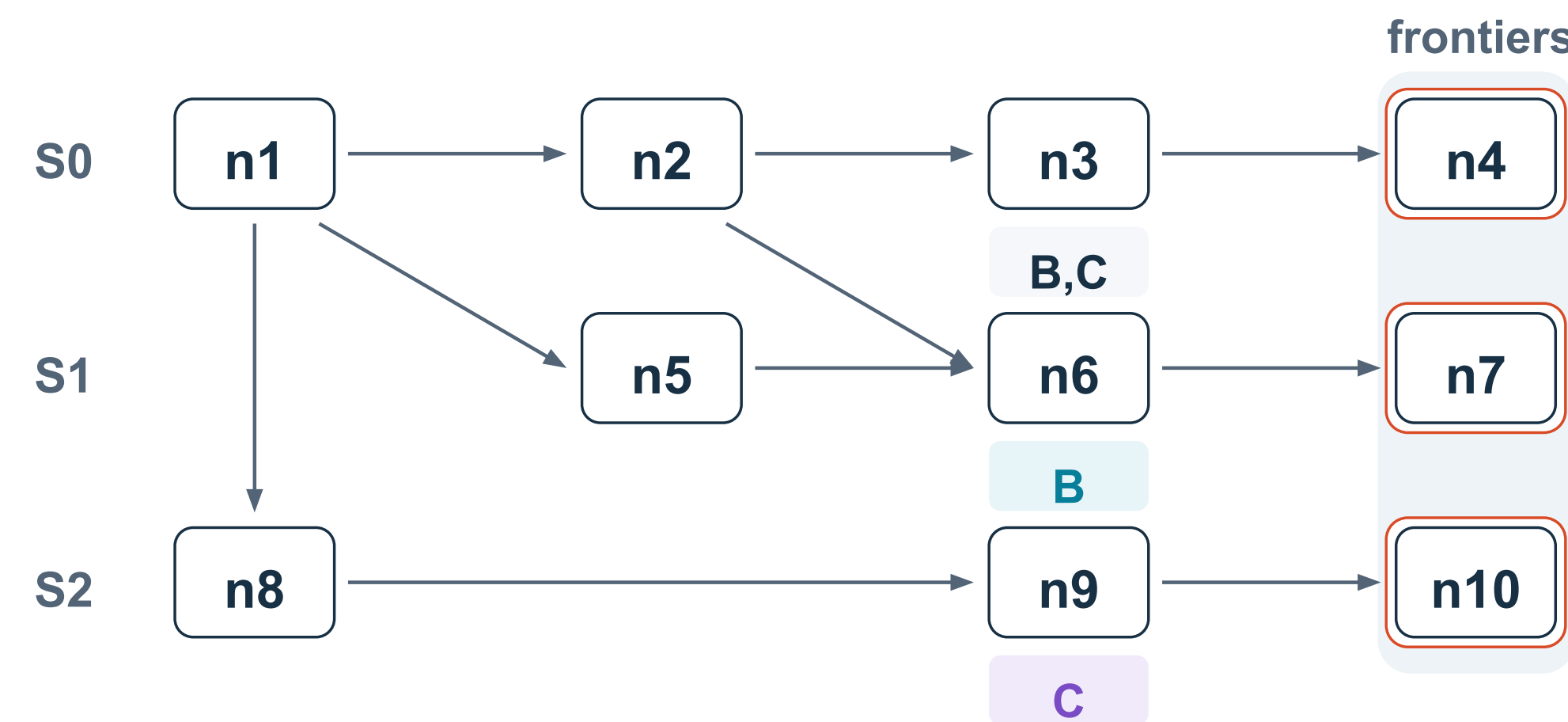
Event polling cannot prove reuse safe. Freed cross-stream blocks stay deferred, so reserved memory can grow.

FIX: prove reuse from graph dependencies

Keep captured addresses valid; reuse only after all recorded uses.

3 / At free: save stream frontiers

B and C are device blocks allocated on S0. Labels below n3/n6/n9 show which operations use them.



Frontier = last captured node on each stream.

B: save {n4, n7}

Used on S0/S1.

C: save {n4, n10}

Used on S0/S2.

At free, store frontiers from the allocation and use streams. No marker nodes are inserted.

Device reuse benchmark · True versus False

`graph_capture_record_stream_reuse:True`

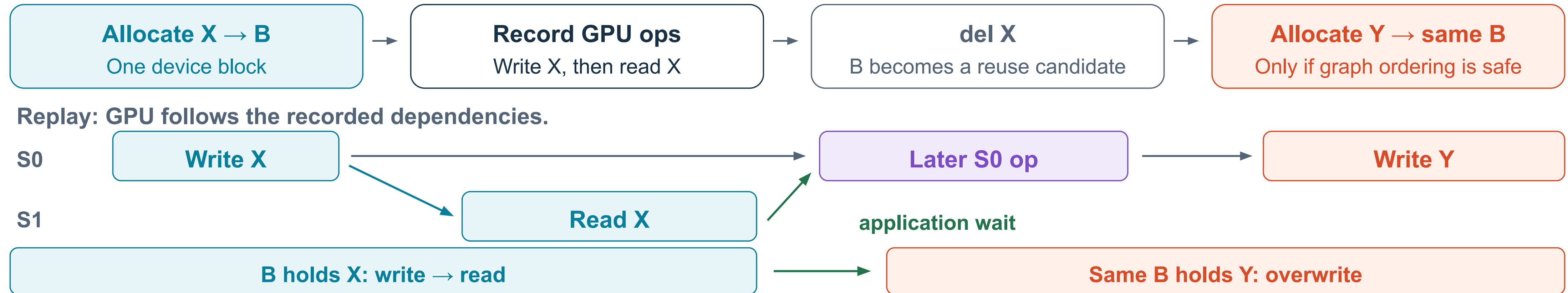
24–98% less reserved memory; peak allocated unchanged

Capture 0.96–1.05×; replay 0.90–1.03× (True / False)

H100; native allocator; 24 configs; 5-run medians. Historical; build/variance unreported; no output validation.

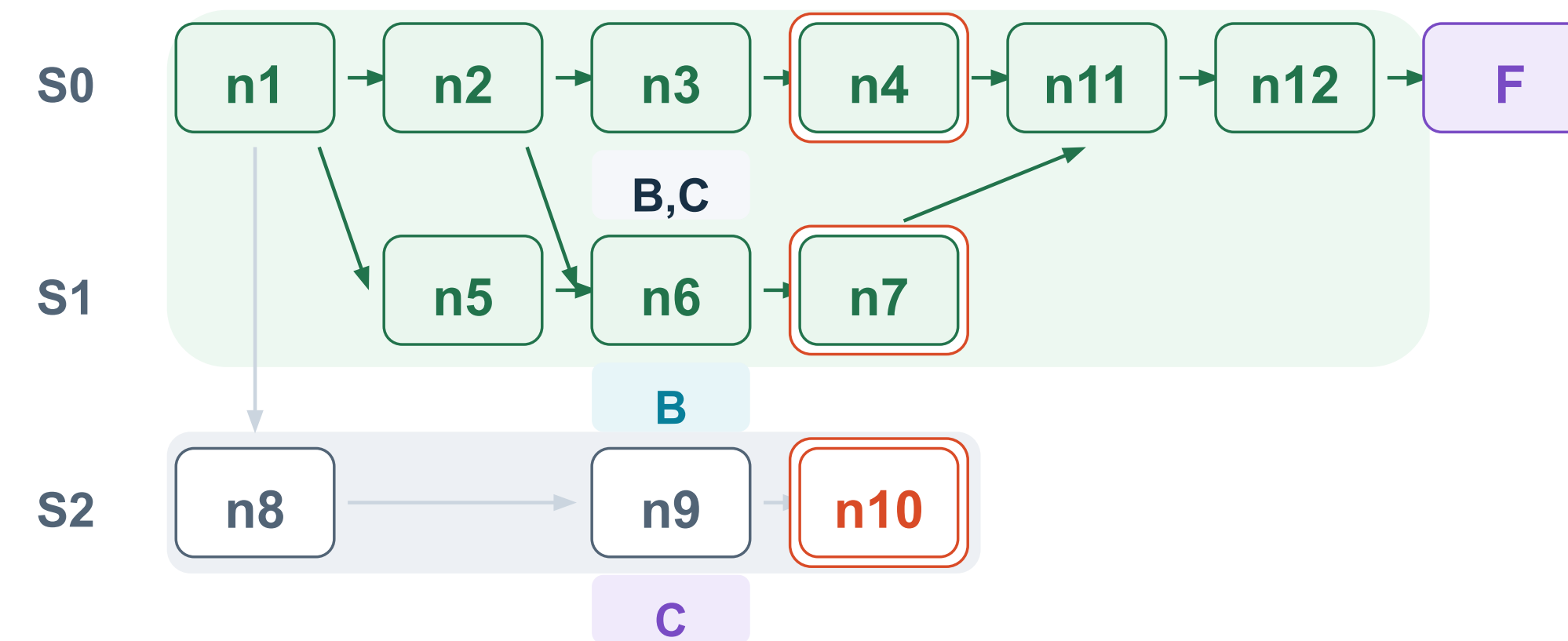
2 / One allocation serves X, then Y

Capture: CPU allocates now; GPU operations are recorded.



4 / At allocation: check predecessors

At allocation on S0, walk back from frontier F. Test every saved marker: B passes; C does not.



F = n13. Green precedes F; S2 cannot reach F.

B: {n4, n7} → reuse

Both markers precede F.

C: {n4, n10} → defer

n10 does not precede F.

Walk back from F; cache predecessors. Pool, size and allocation stream (S0) must also match.

Pinned host Allocator

Example: GPU adds loss terms 1 and 2. The correct loss is 3; recycled CPU staging can turn it into 4.

Pseudocode · pinned_tensor(v) writes v on the CPU

```
with capture(g):
    p = pinned_tensor(1)      # CPU writes 1
    L1 = copy_to_gpu(p)       # record H2D
    del p                     # release Python owner
    q = pinned_tensor(2)      # CPU writes 2
    L2 = copy_to_gpu(q)       # record H2D
    loss = L1 + L2            # record GPU addition
g.replay()                  # execute GPU work
```

CPU assignments run during capture; only GPU work runs on replay.

If q reuses p's address

L1 = L2 = 2 → loss = 4

HostPrivatePool retains p

L1 = 1, L2 = 2 → loss = 3

The graph cannot order CPU writes. Keep captured host buffers alive.

HostPrivatePool: retain for correctness

Retain capture-allocated pinned buffers used by recorded copies. Their addresses stay valid for the graph.

Unused buffers may recycle. CPU access still needs synchronization with replay.