Barrelfish Capabilities

Ross McIlroy

Systems and Networking Group
Microsoft Research Cambridge

Overview

- Give a taste of how Barrelfish manages resources using a capability-based model
- Much of the details are hidden from user-level apps by libbarrelfish
- At the end of the talk, I will outline the ways in which you are most likely to interact with capabilities

What are Capabilities

- Owning a capability gives a domain (application) the ability to access a particular resource
  - Physical RAM
  - Page table entries
  - CPU time (dispatcher)
  - Communication channel endpoints
- A domain starts with a small number of capabilities, get more from other services
  - Ram Caps from the memserv
  - Endpoints from the other domain you wish to talk to

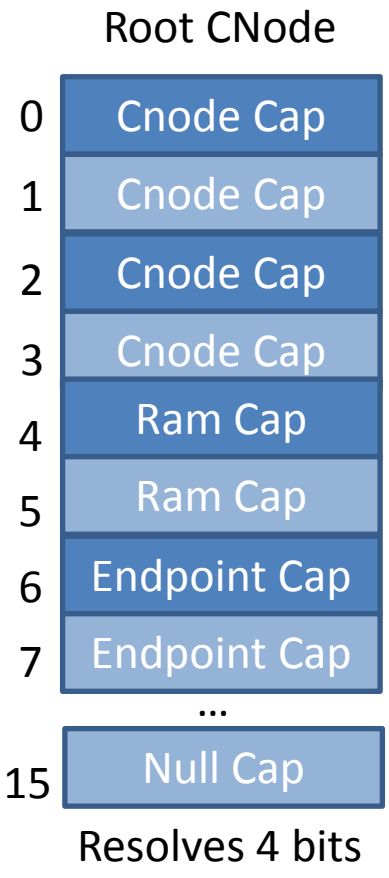
Capability Types

Null	Empty slot
PhysAddr	Physical address range
RAM	Physical address range in memory
Frame	Mappable address range of memory
DevFrame	Mappable address range of device memory
Vnode_*	Page table node
CNode	Node to store other capabilities
FCNode	Foreign CNode – Cnode on a different core
Dispatcher	A dispatcher's control block
Endpoint	Endpoint of an IDC channel
Kernel	Special syscall privileges (monitor)
IO	Legacy IO range
IRQ	Capability to handle interrupts
Others…	Notify, BMPEndpoint, BMPTable, Domain, etc.

Capability Address Space

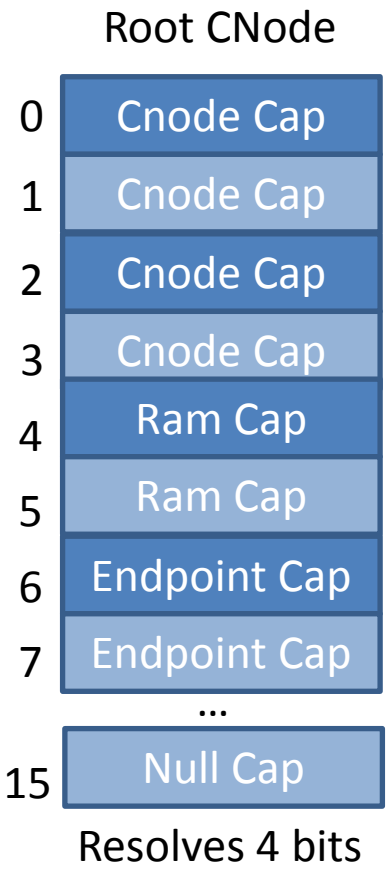
- Each domain's capabilities are stored in a guarded capability table, known as its **cspace**
- The cspace consists of a set of **cnodes**, each of which holds a power-of-two number of capability slots
- The cspace is opaque to user-level code
  - User level code gets a **capref** pointing to the capability's slot
  - The kernel uses this capref to traverse the cspaces guarded capability table

Cspace Lookup Example



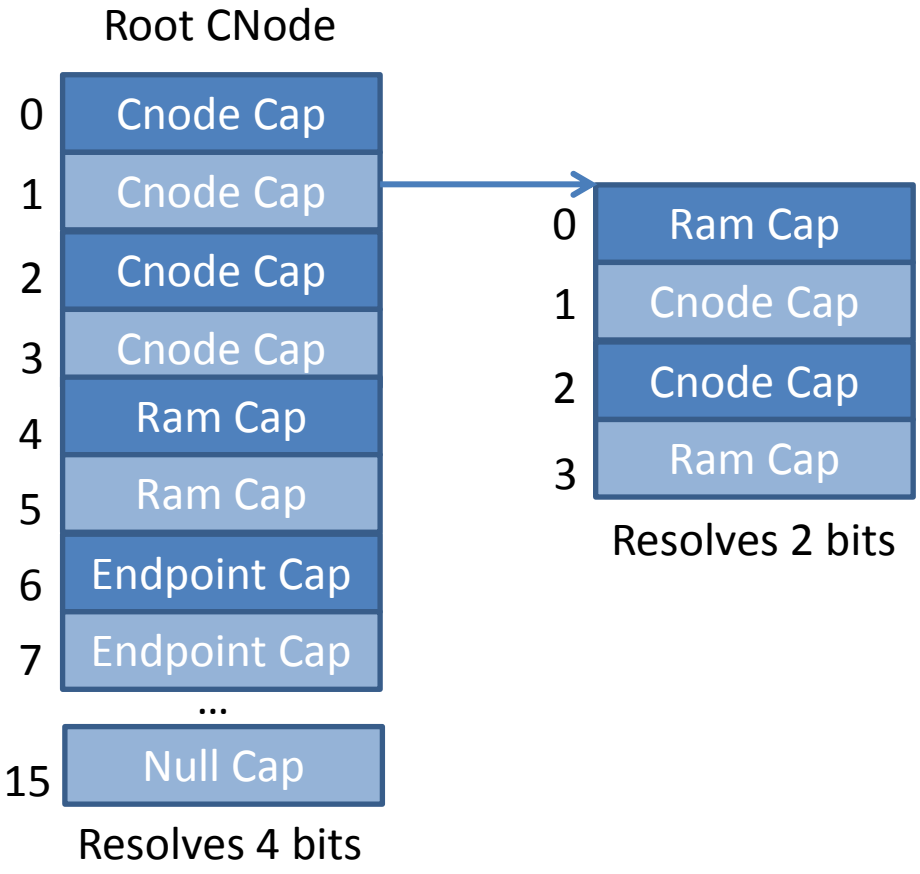
Cspace Lookup Example

Capability Address = 0x2A21

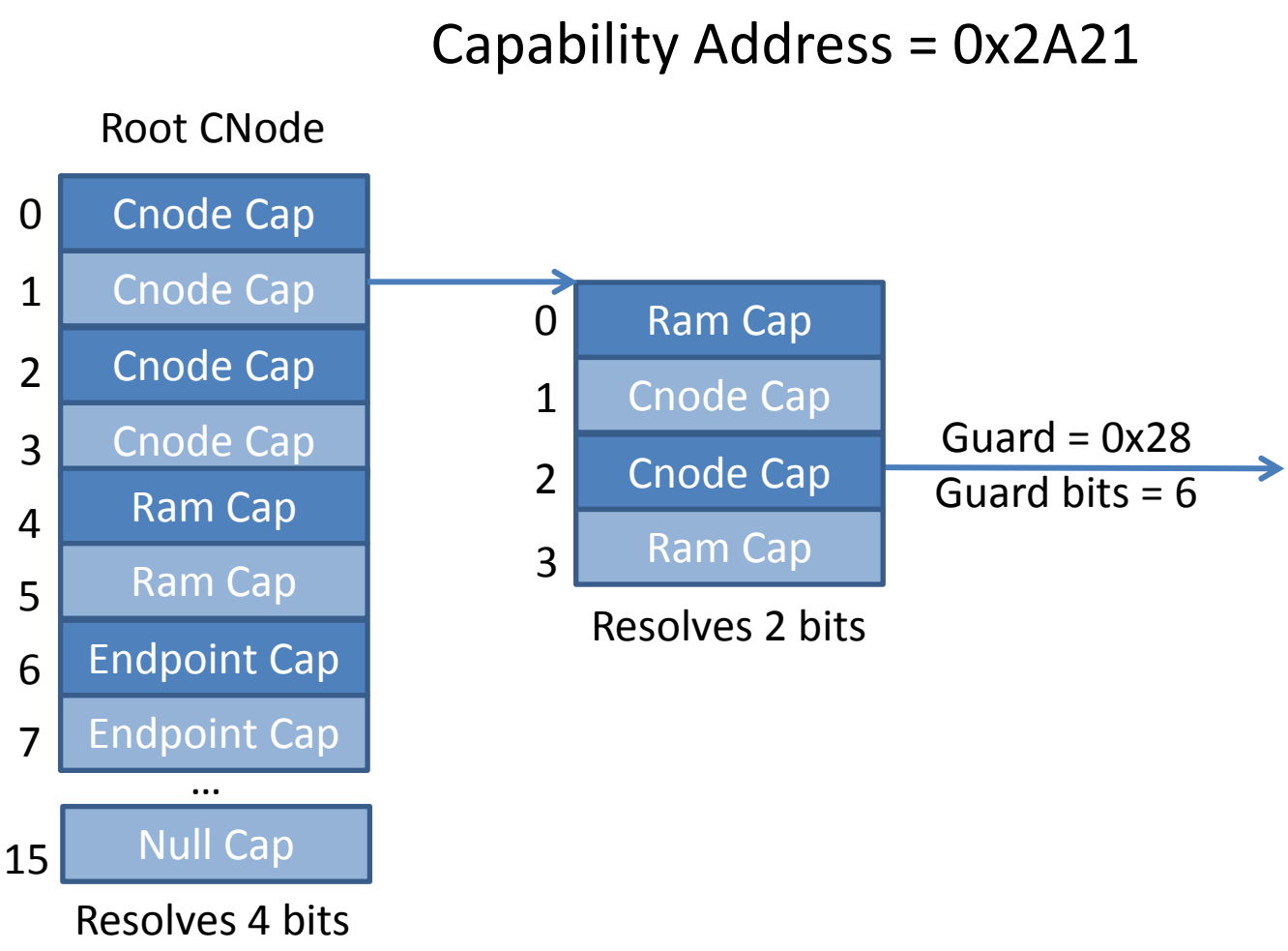


Cspace Lookup Example

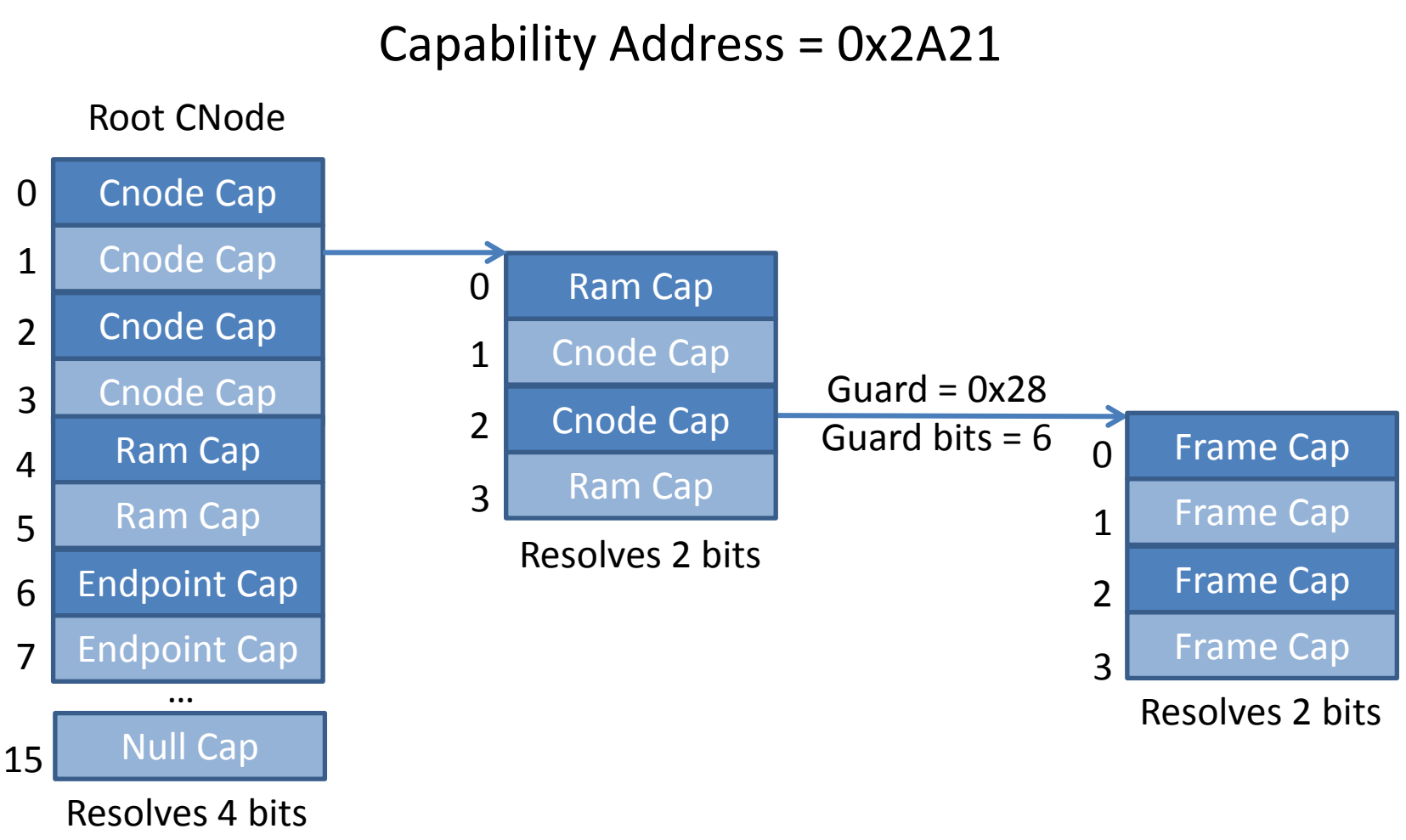
Capability Address = 0x2A21



Cspace Lookup Example



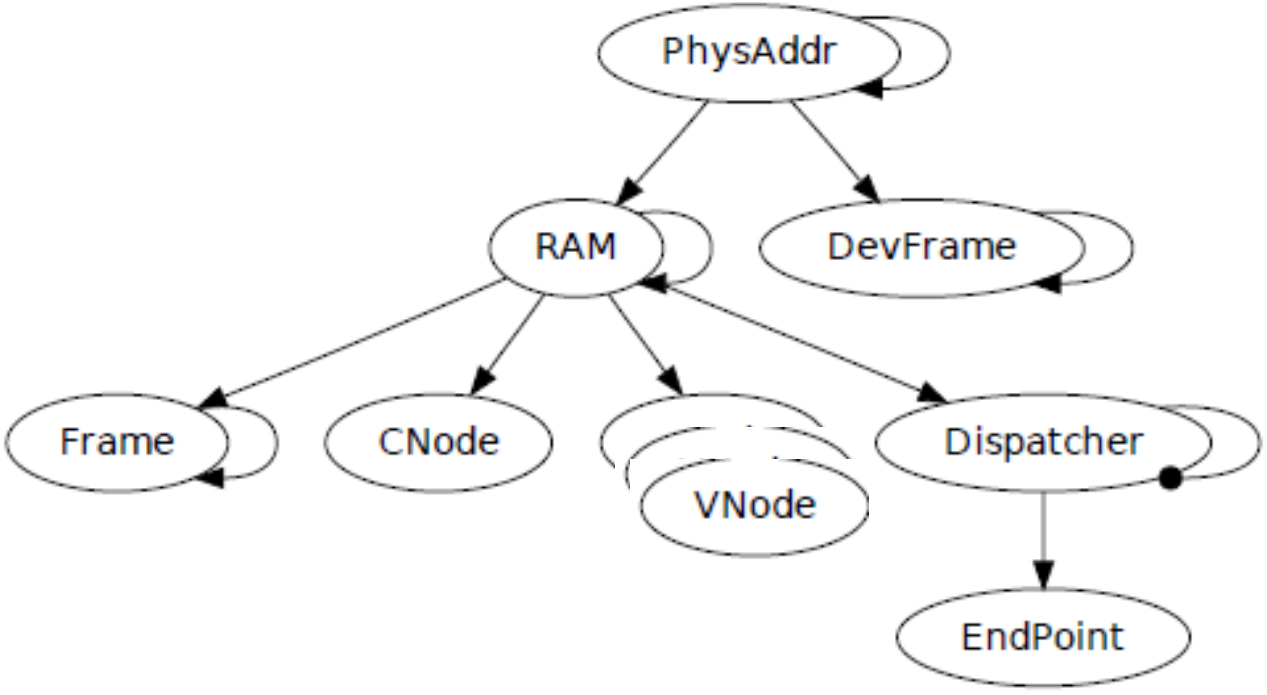
Cspace Lookup Example



Operations on Capabilities

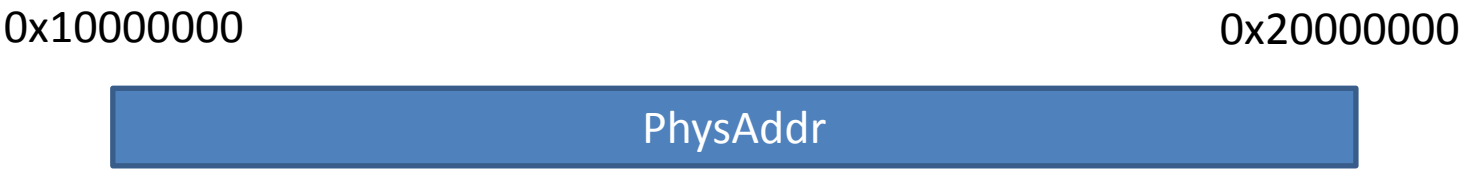
- `slot_alloc()` – Allocate a free slot in my cspace
  - Calls `cnode_create` if required to create another cnode
- `cap_copy()` - Create a new copy of capability (in a new slot)
- `cap_mint()` - Copy capability, changing type specific parameters
- `cap_retype()` - Create one or more descendent capabilities
  - These could be of a different type (e.g., RAM to Frame)
- `cap_delete()` - Delete this cap, but leave the slot for reuse
- `cap_destroy()` - Delete this cap and free the slot
- `cap_revoke()` - Delete all copies and decedents, but not this cap

Retyping Capabilities

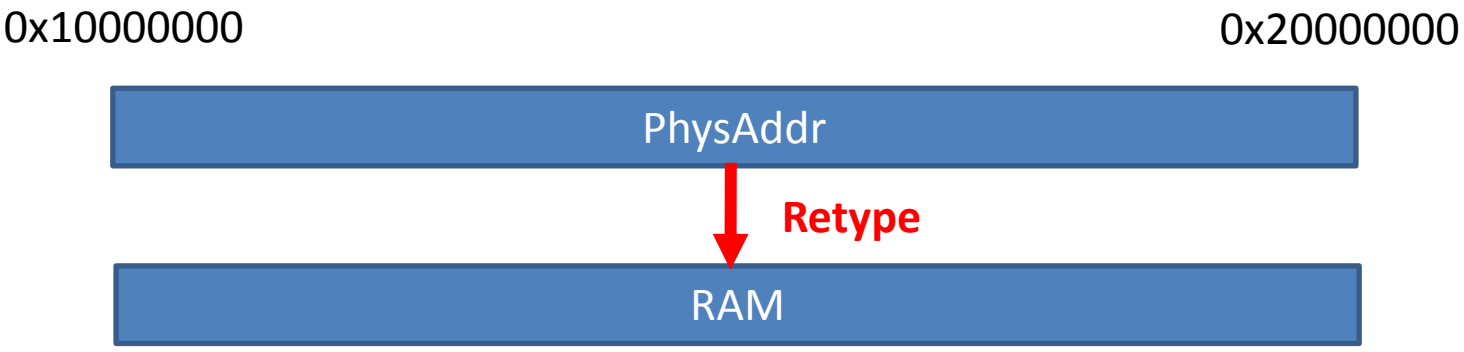


- Rules are specified using the Hamlet DSL (capabilities/caps.hl)

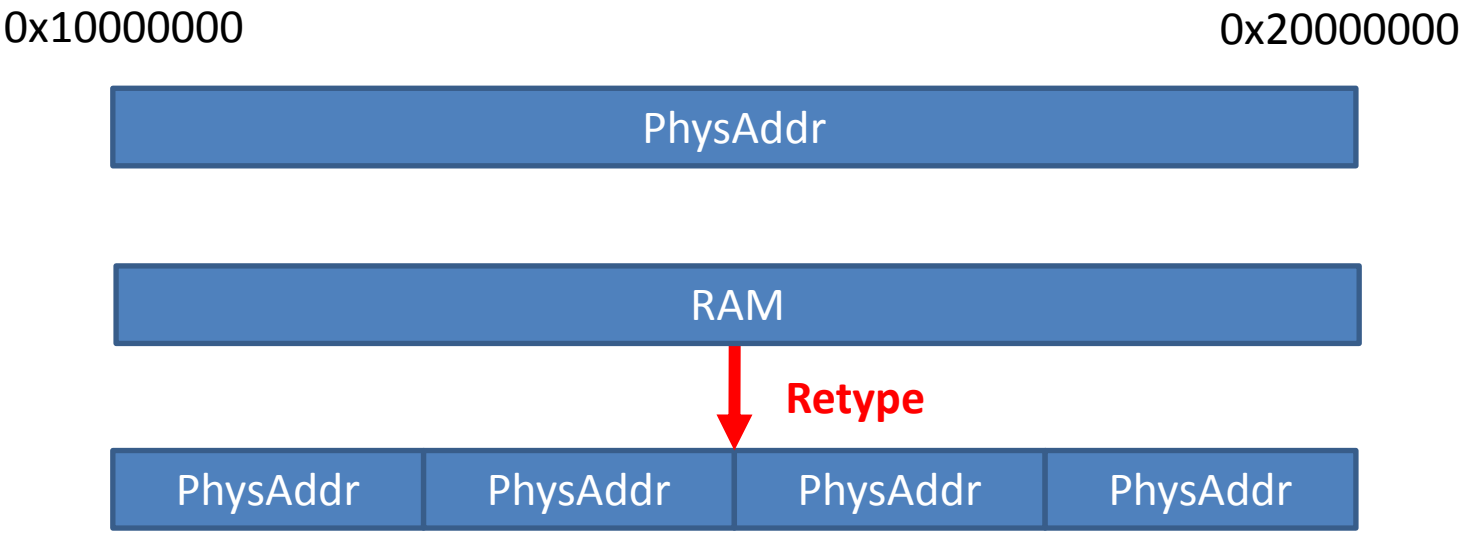
Operations on Capabilities



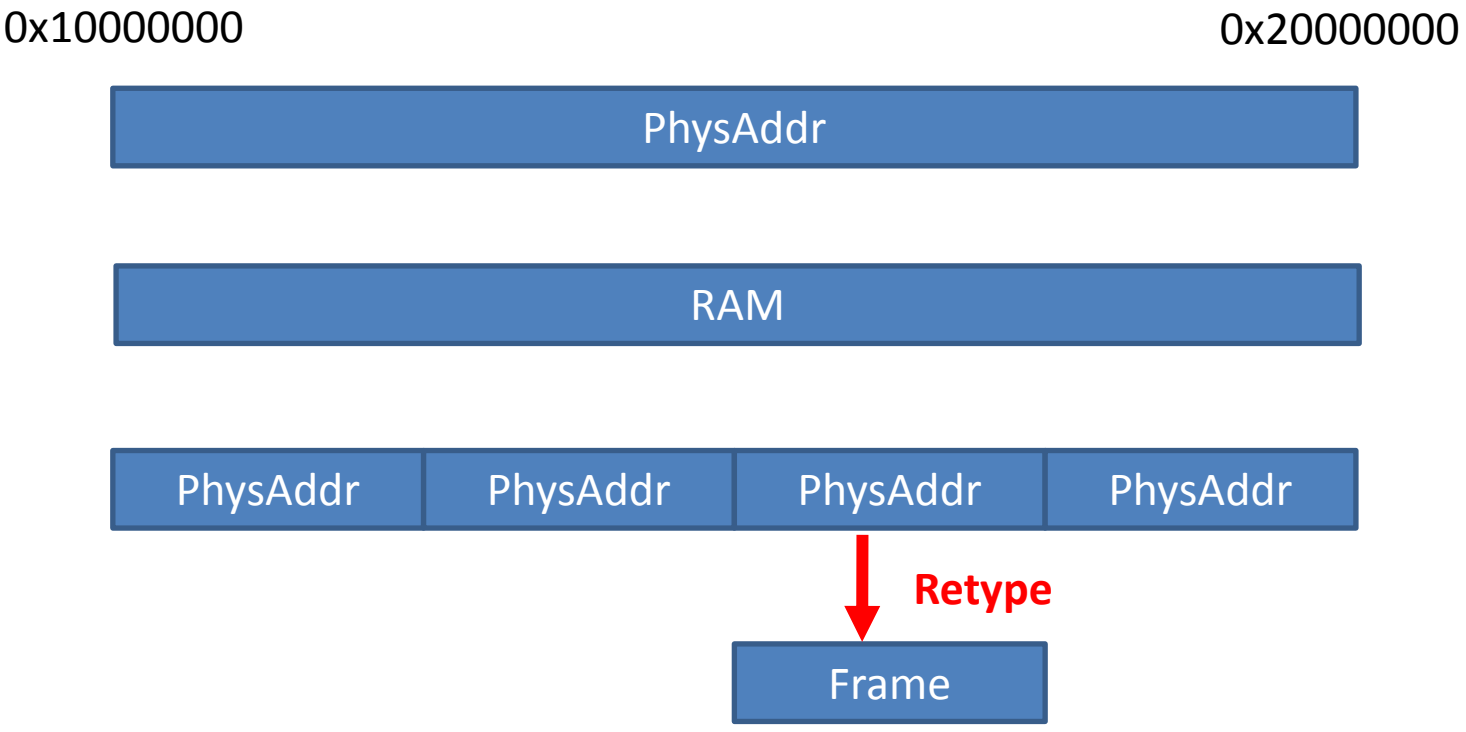
Operations on Capabilities



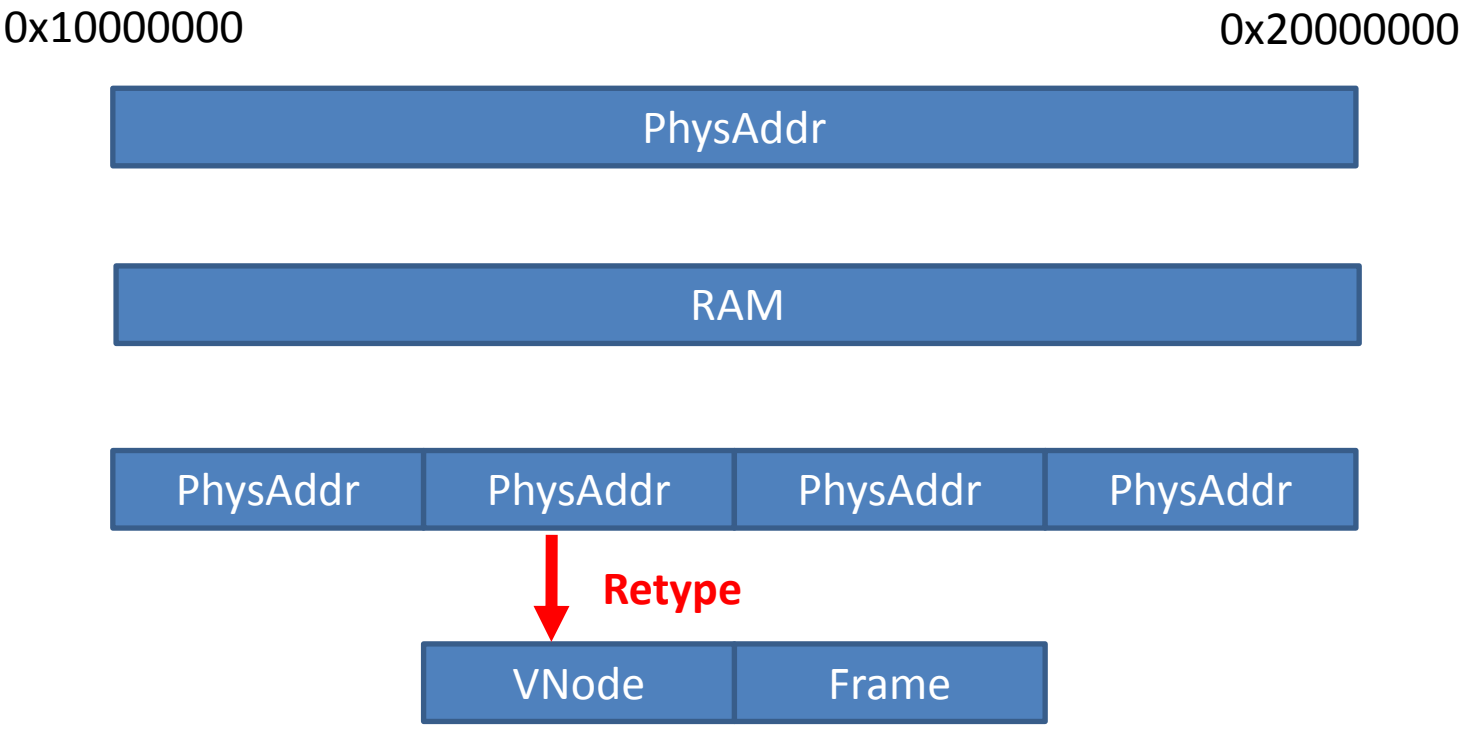
Operations on Capabilities



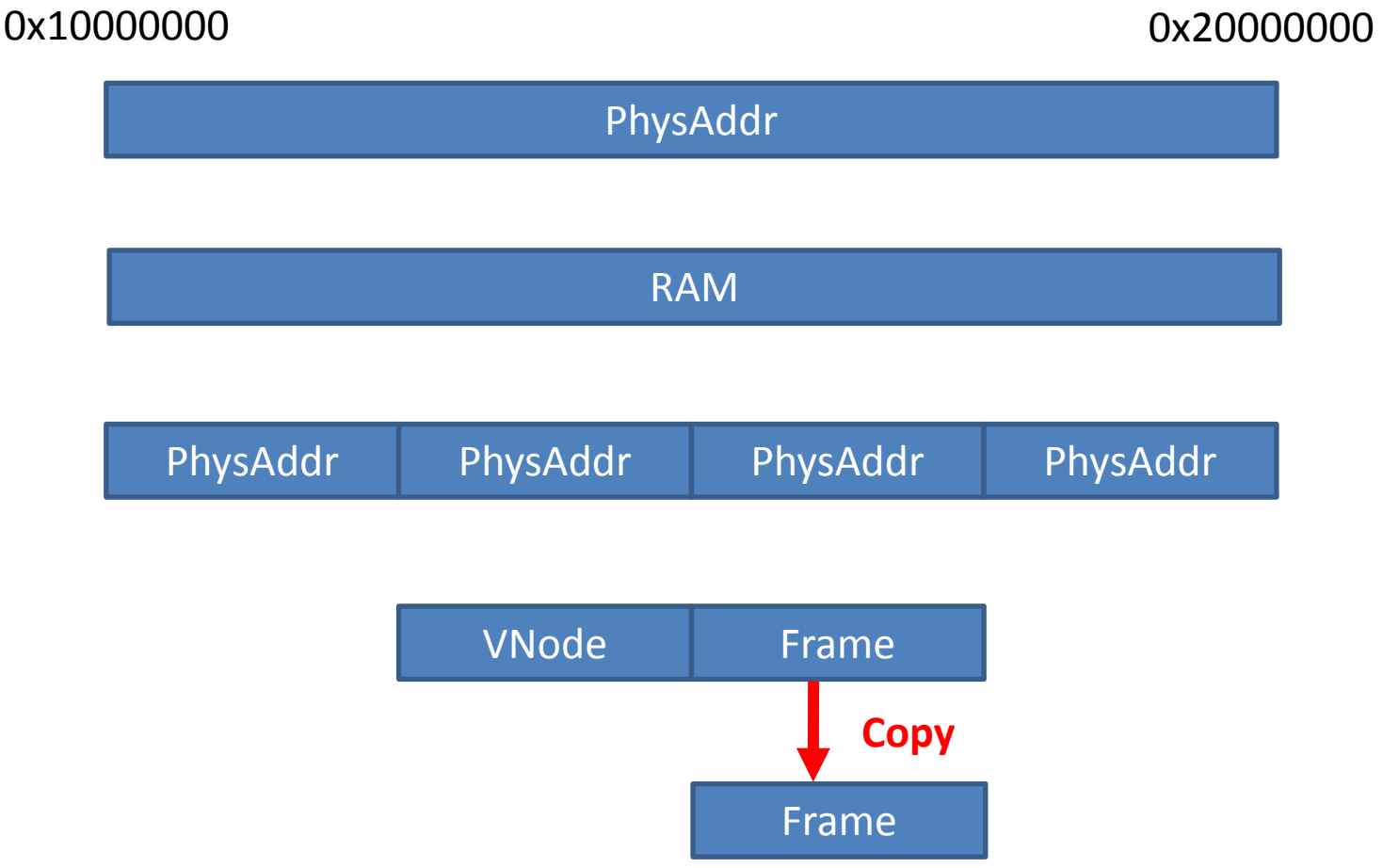
Operations on Capabilities



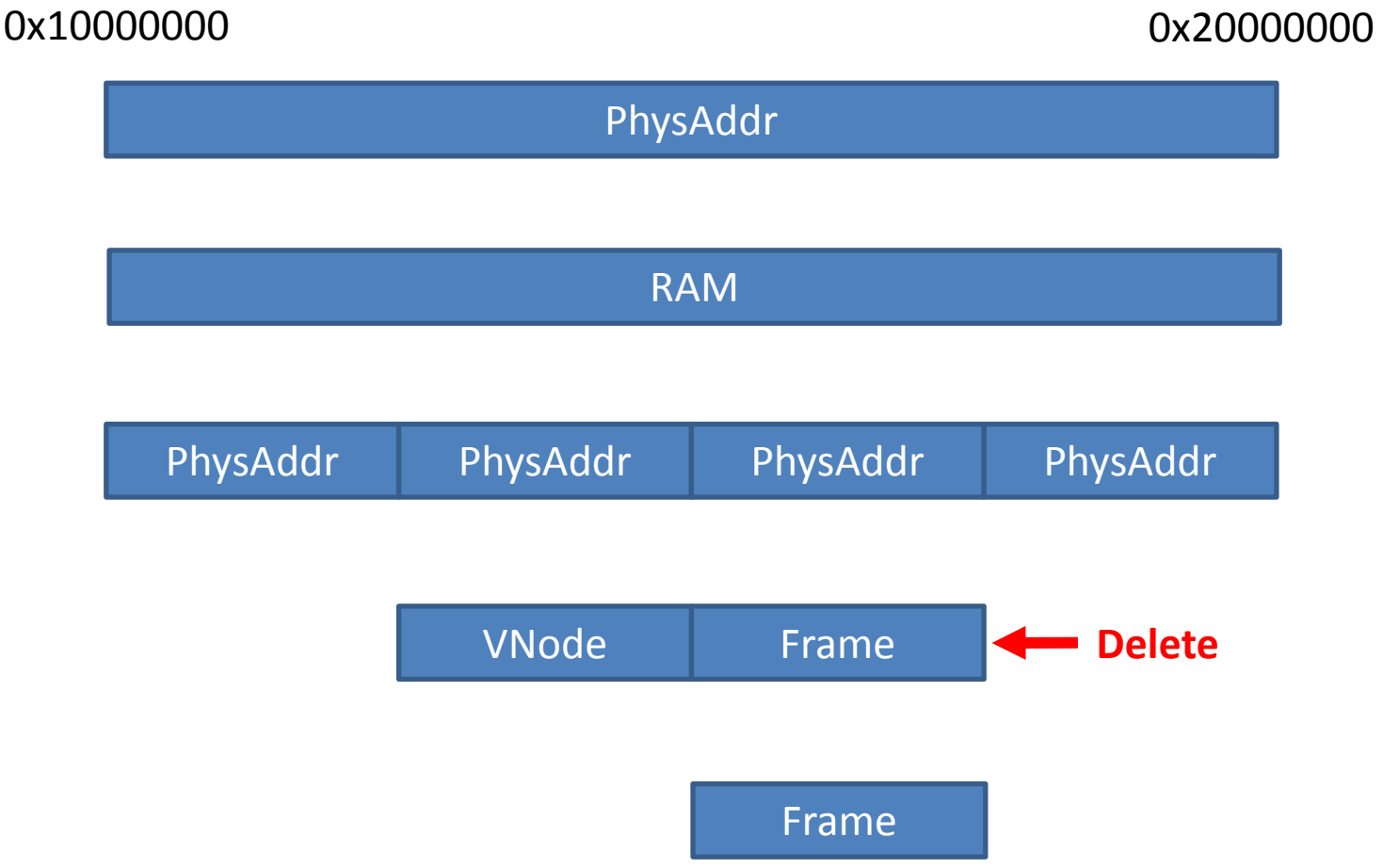
Operations on Capabilities



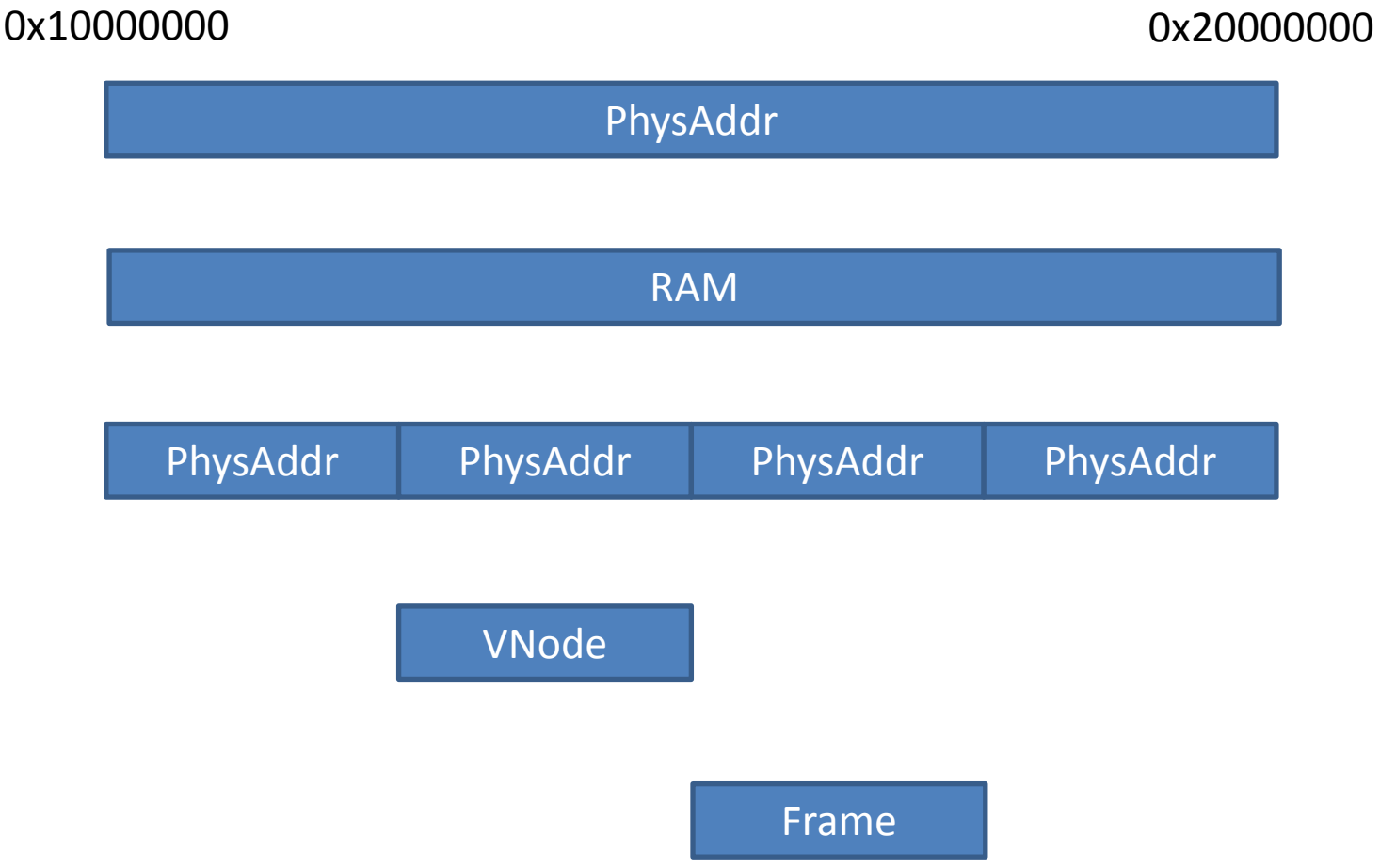
Operations on Capabilities



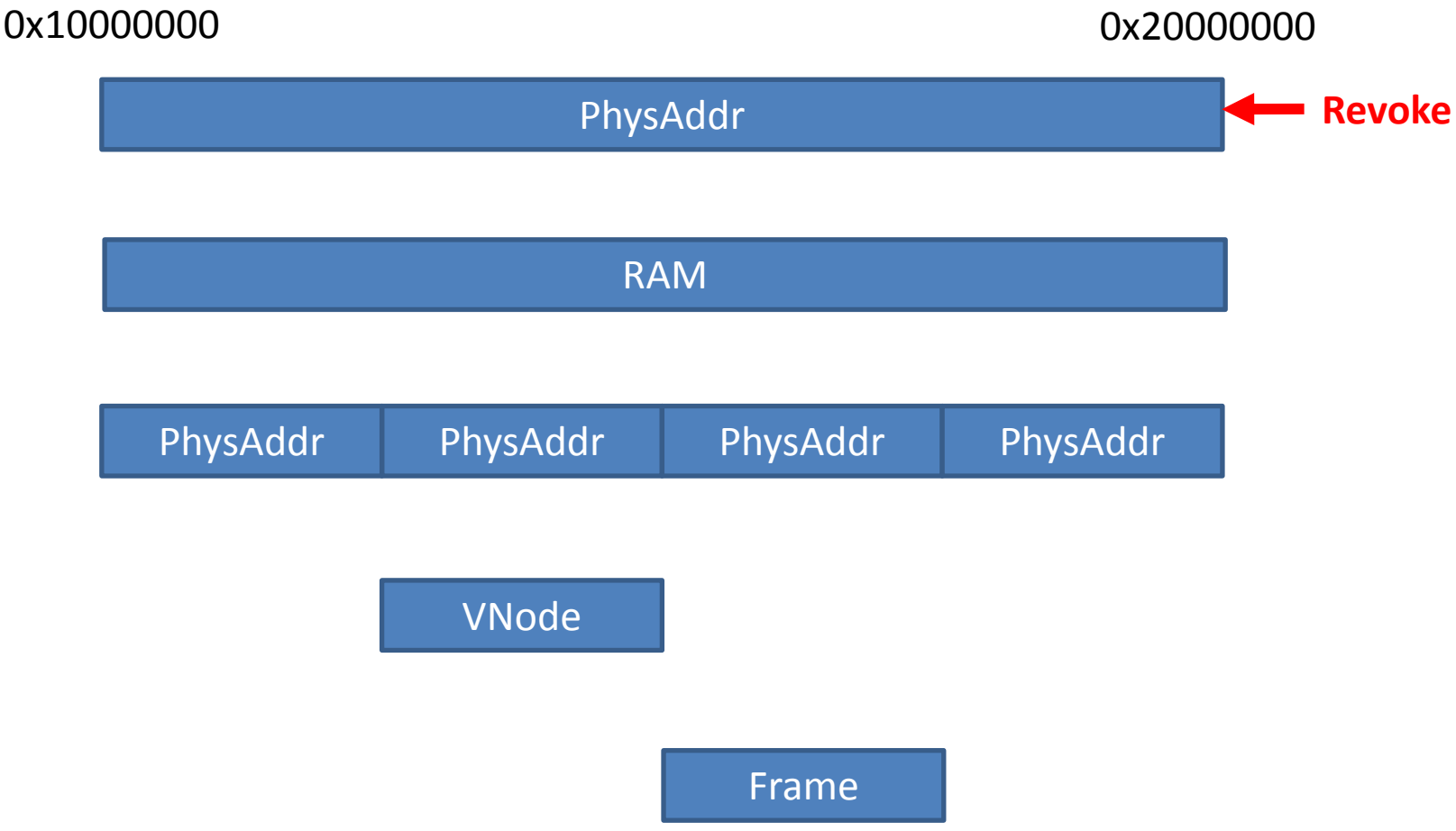
Operations on Capabilities



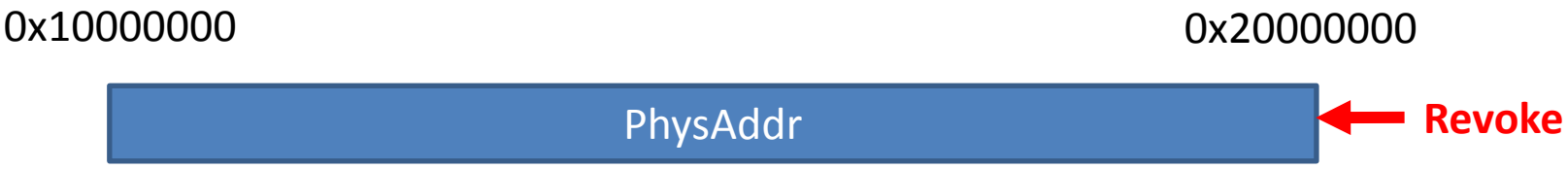
Operations on Capabilities



Operations on Capabilities



Operations on Capabilities



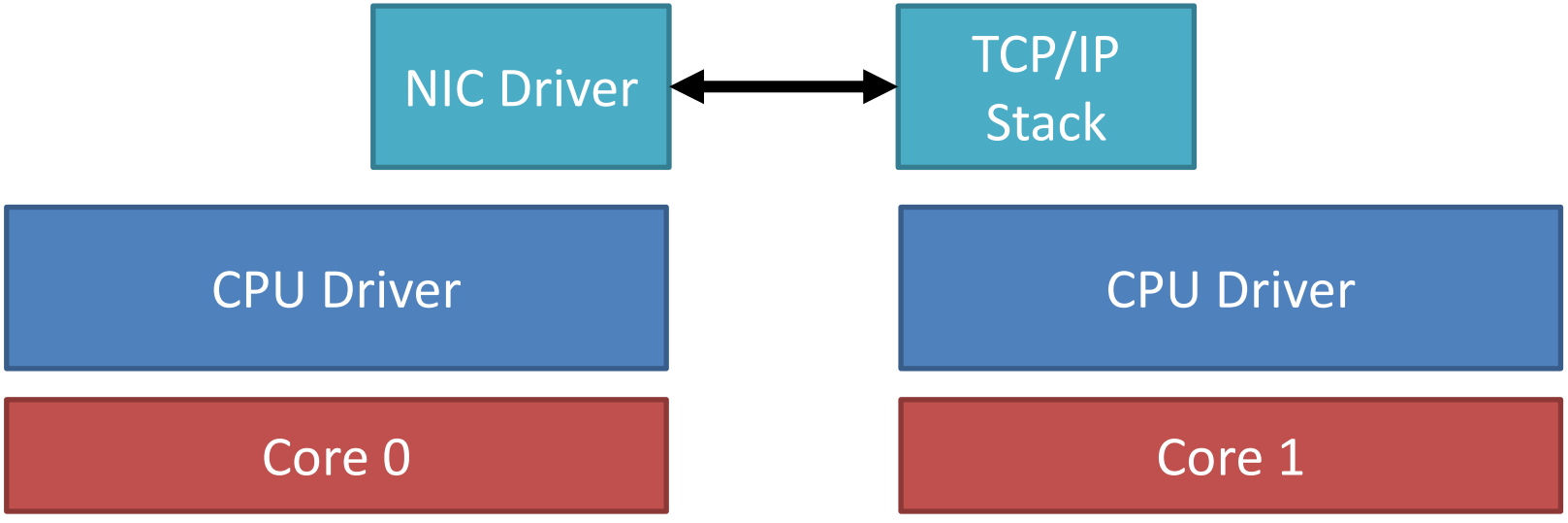
Sending Capabilities

- Capabilities can be sent between domains over IDC channels
  - `message send_cap(cap sent_cap);`
- Local Message Passing (LMP)
  - Copy capability into the destination's cspace
- Cross-Core Message Passing (e.g., UMP)
  - Capabilities are held by the CPU driver (kernel)
  - Barrelfish has one CPU driver per core (multikernel)
  - Need to transfer capability to the destination core's CPU driver

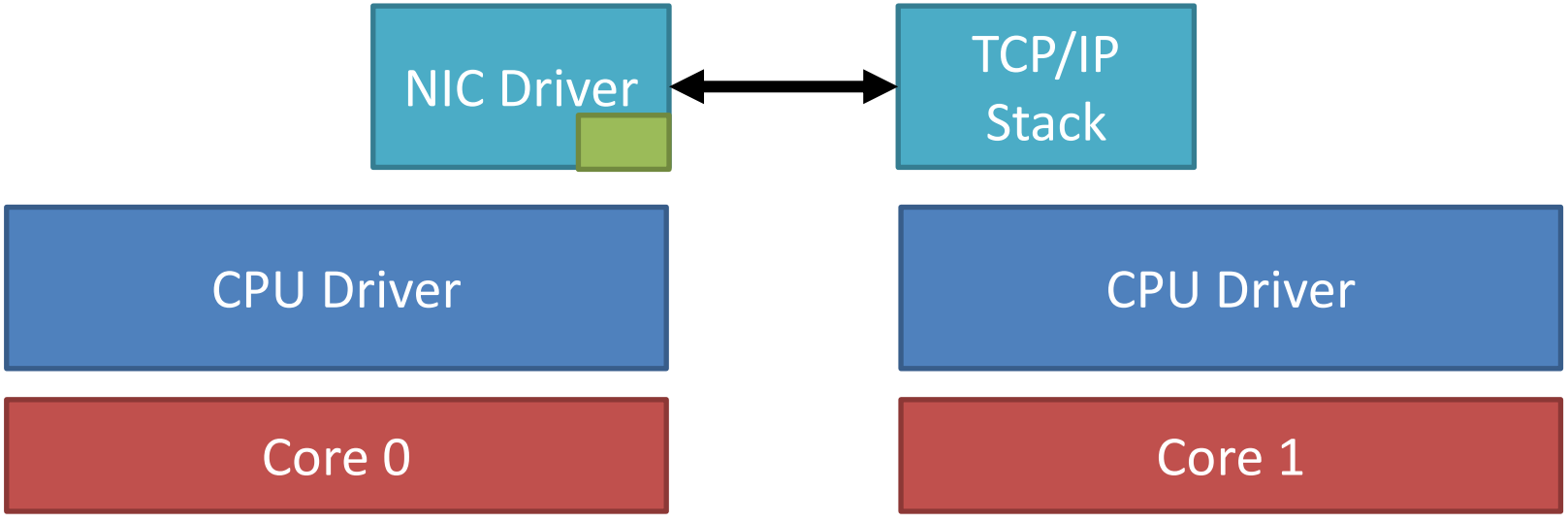
Cross-Core Capabilities

- Some caps cannot be sent cross-core
  - dispatcher, endpoint
- Some are converted to a different type when sent
  - cnode -> foreign cnode

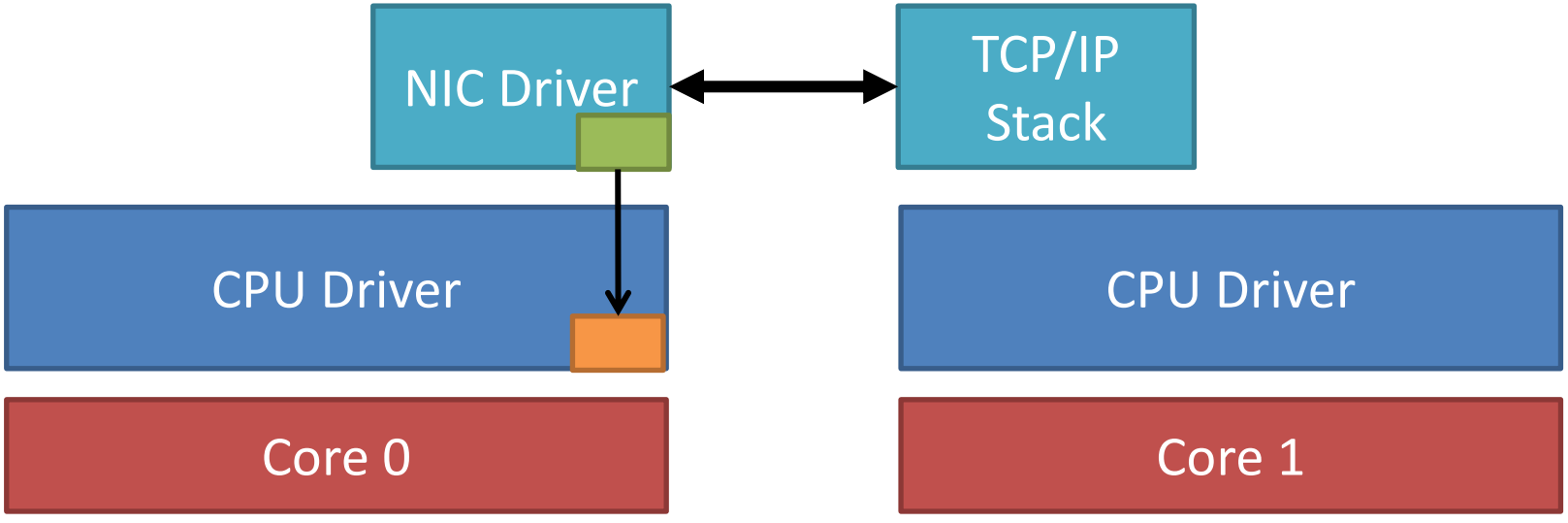
Cross-Core Capabilities



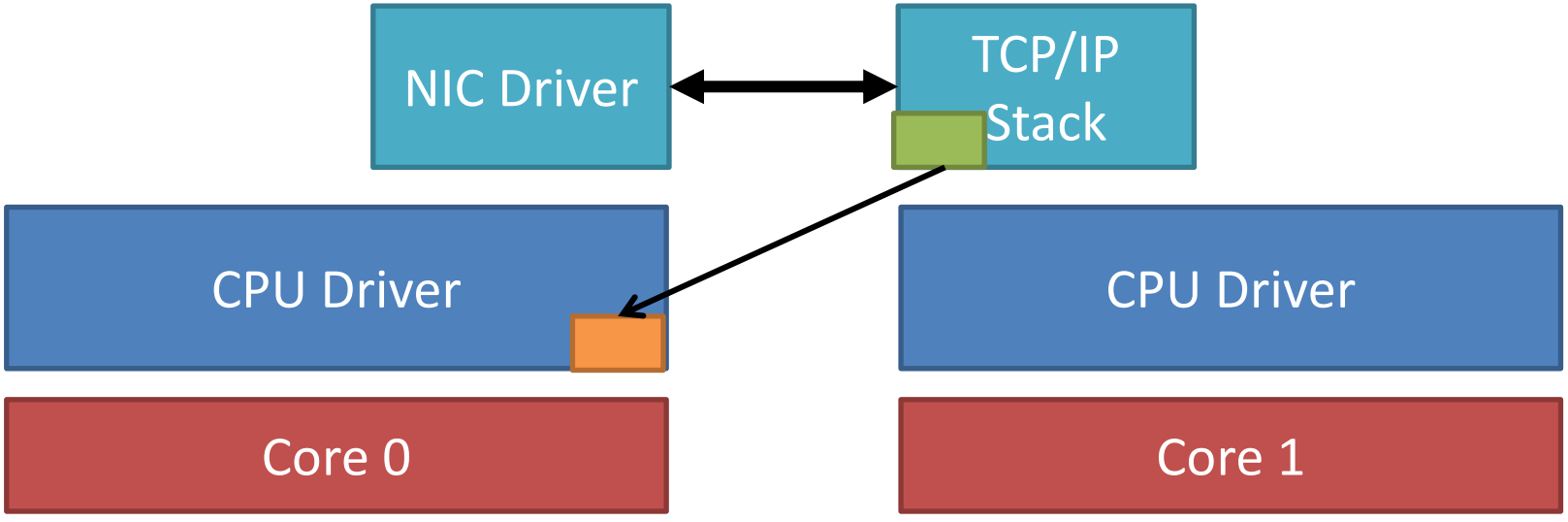
Cross-Core Capabilities



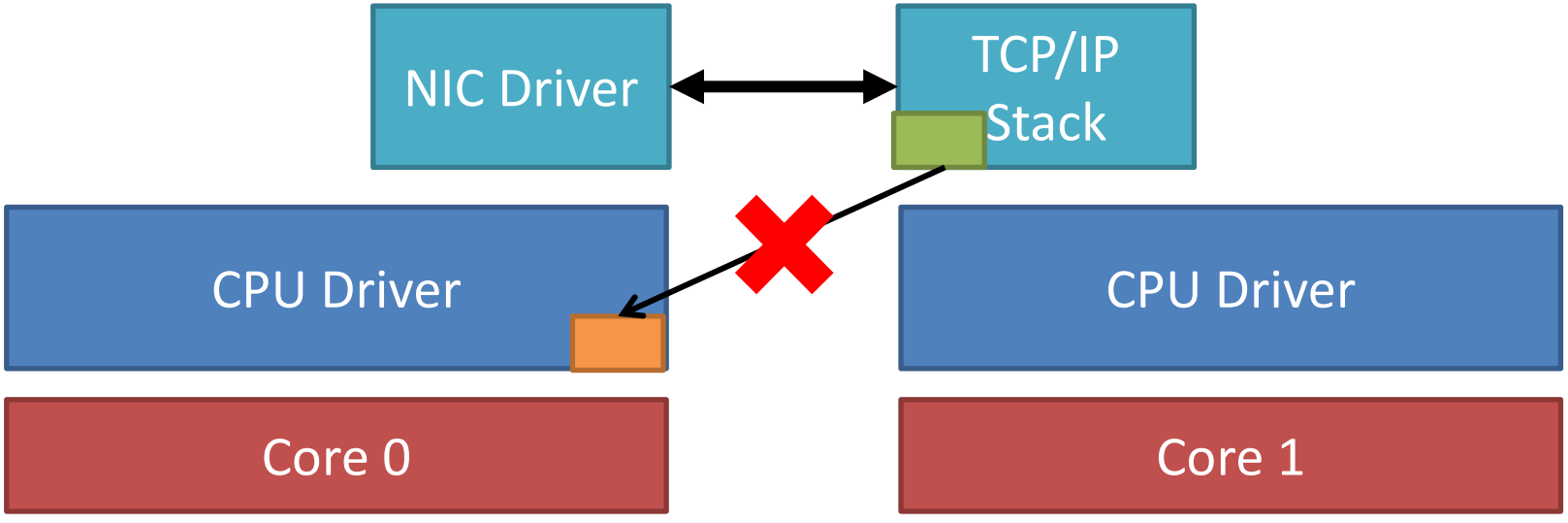
Cross-Core Capabilities



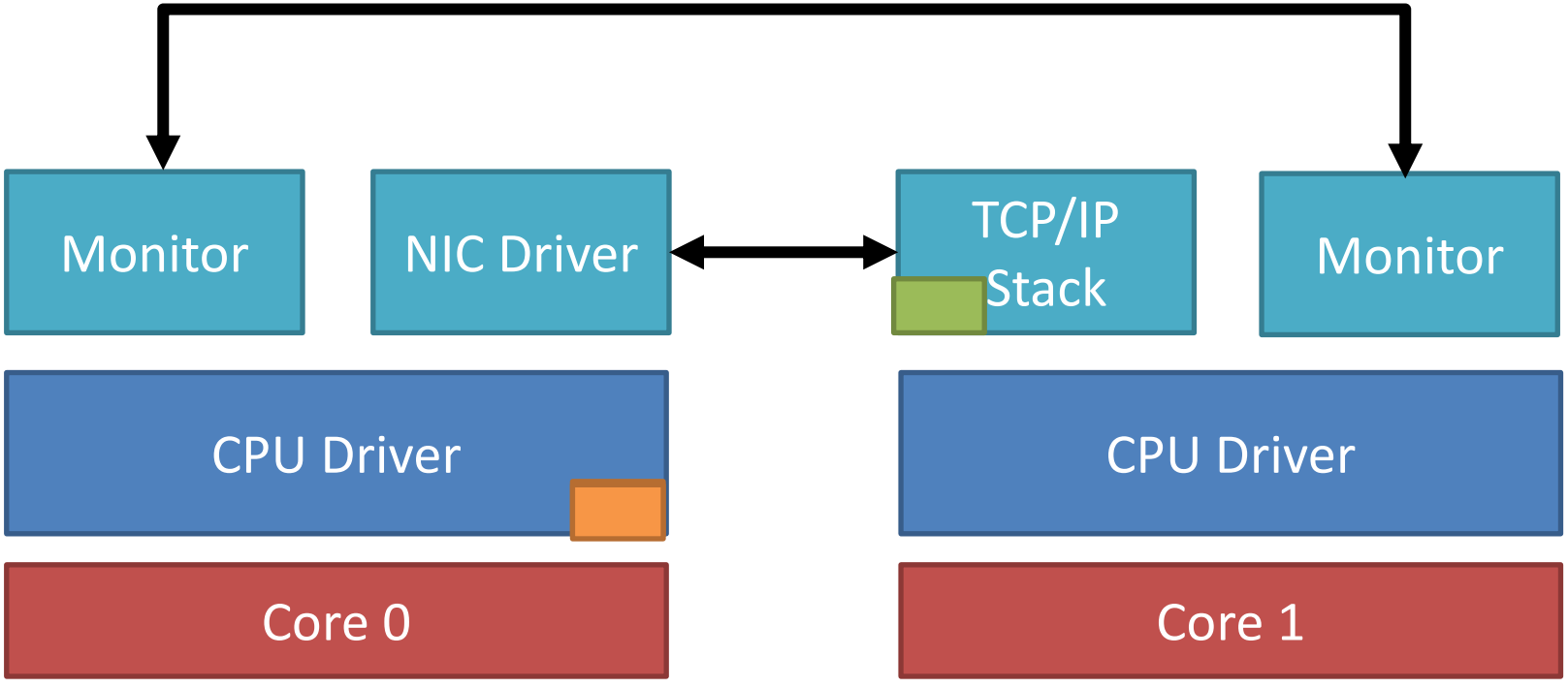
Cross-Core Capabilities



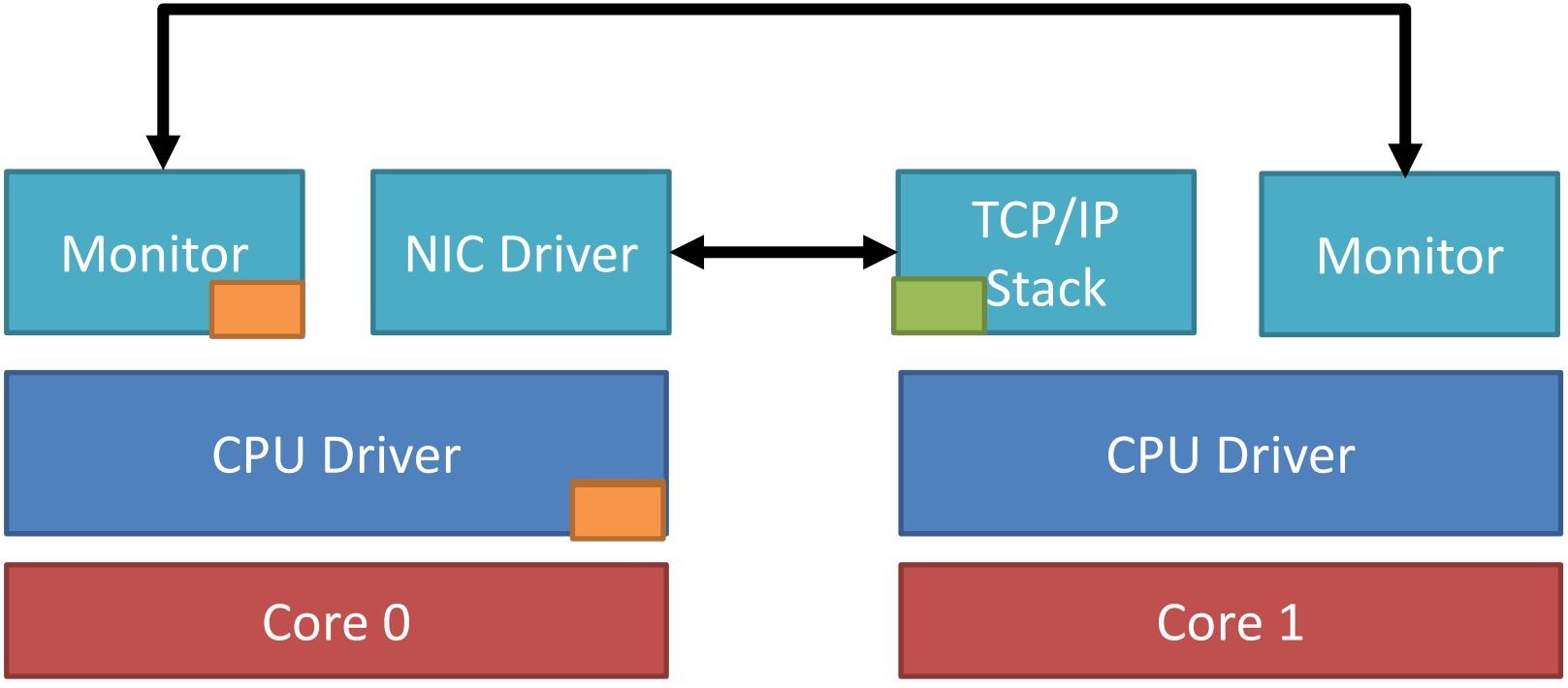
Cross-Core Capabilities



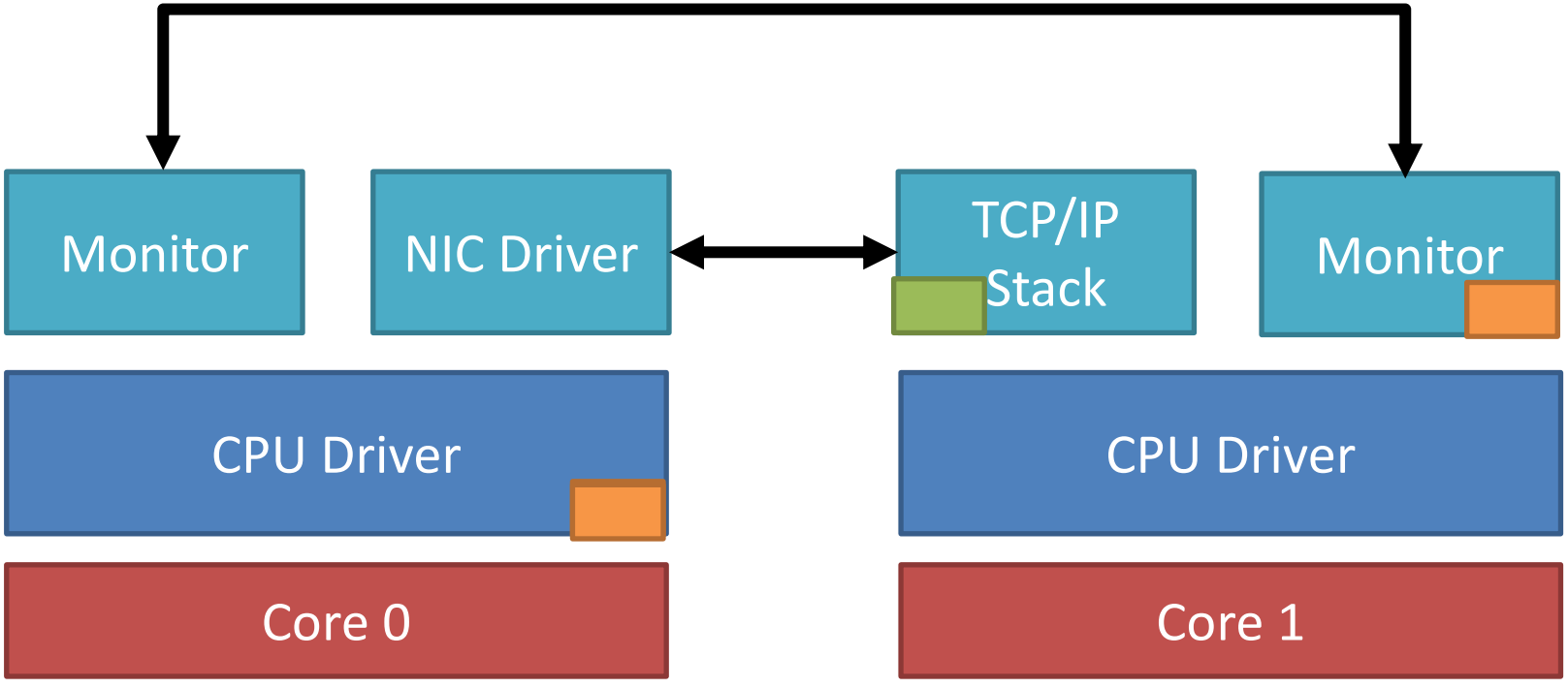
Cross-Core Capabilities



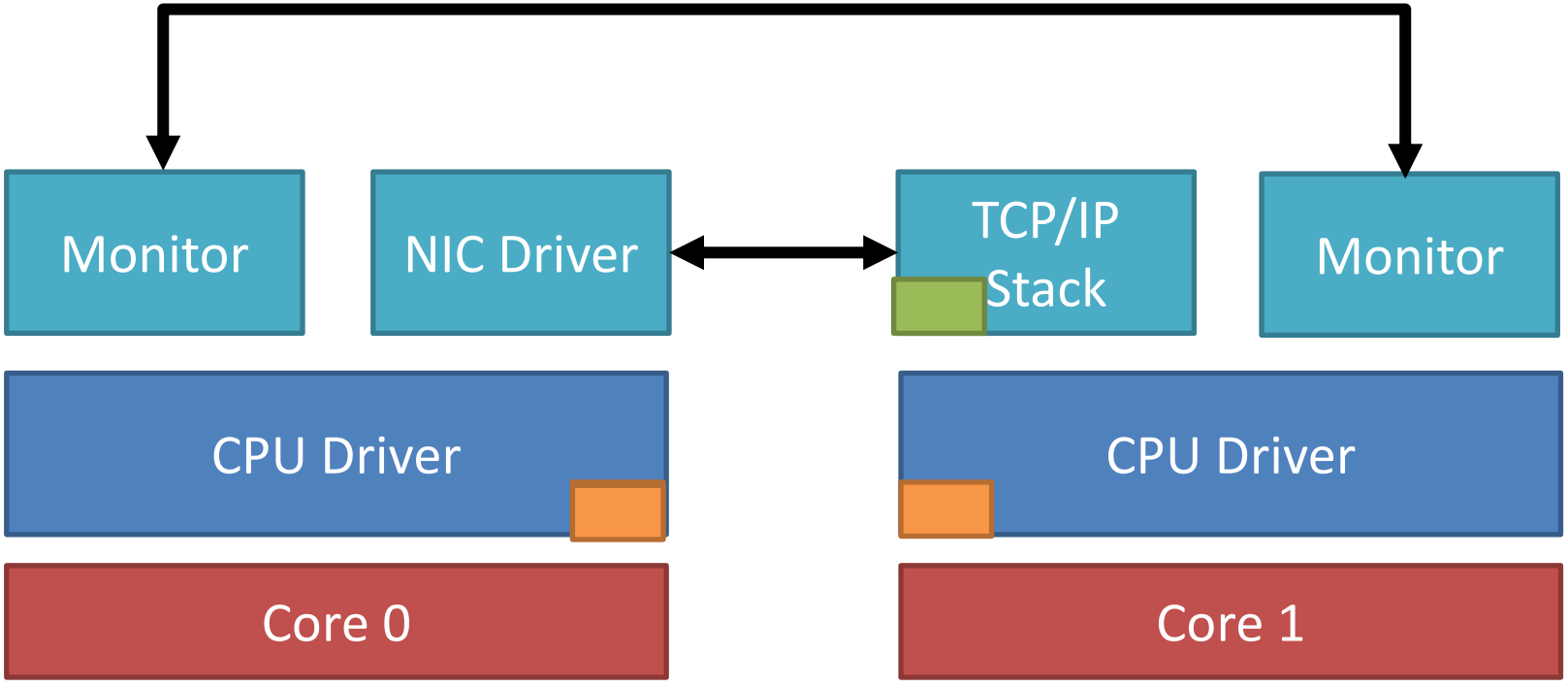
Cross-Core Capabilities



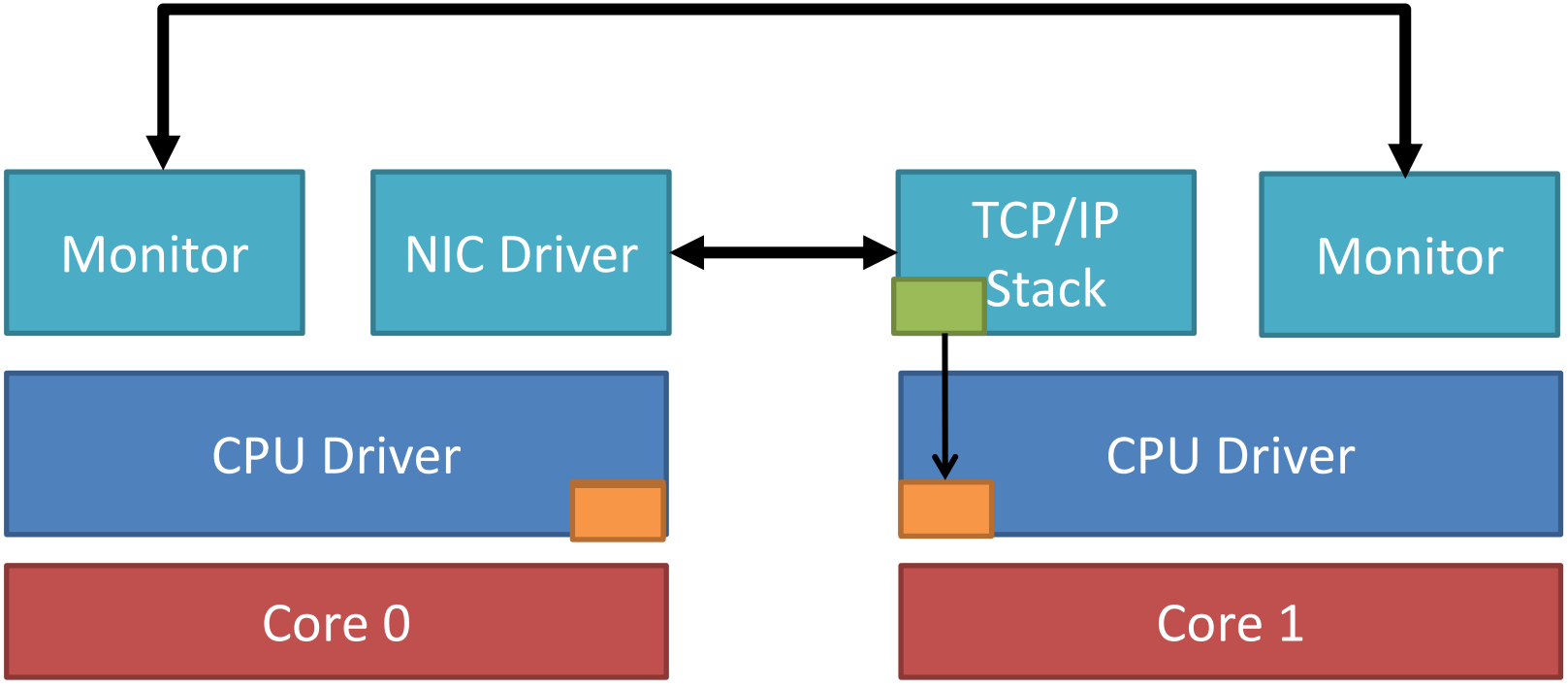
Cross-Core Capabilities



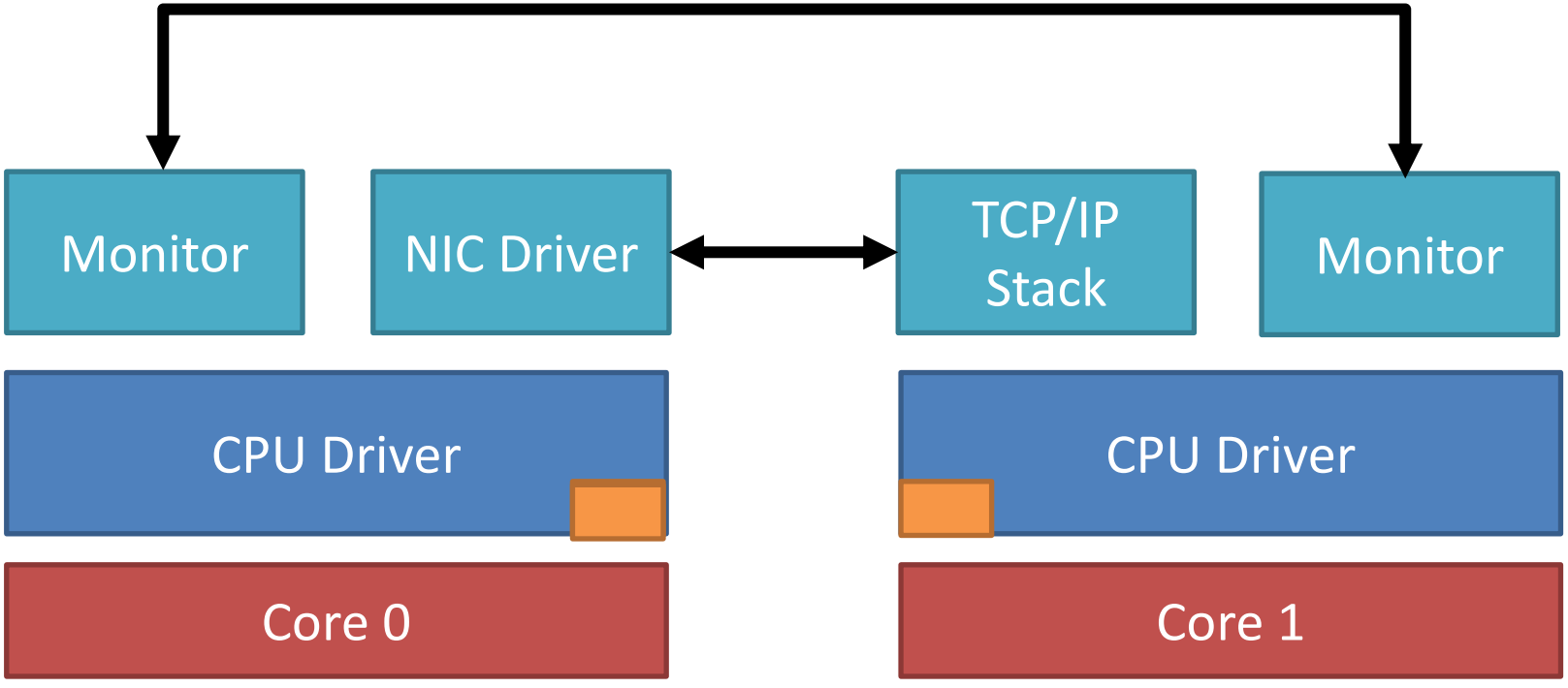
Cross-Core Capabilities



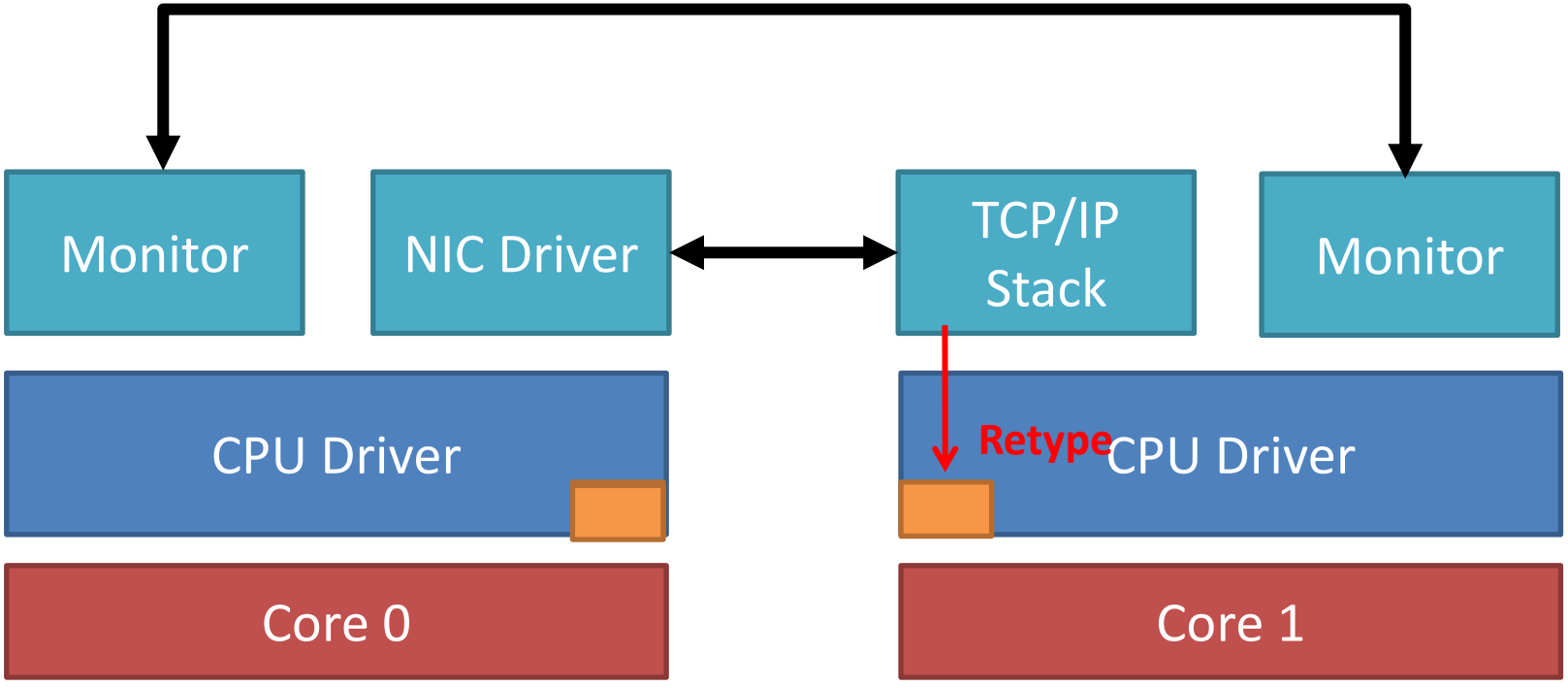
Cross-Core Capabilities



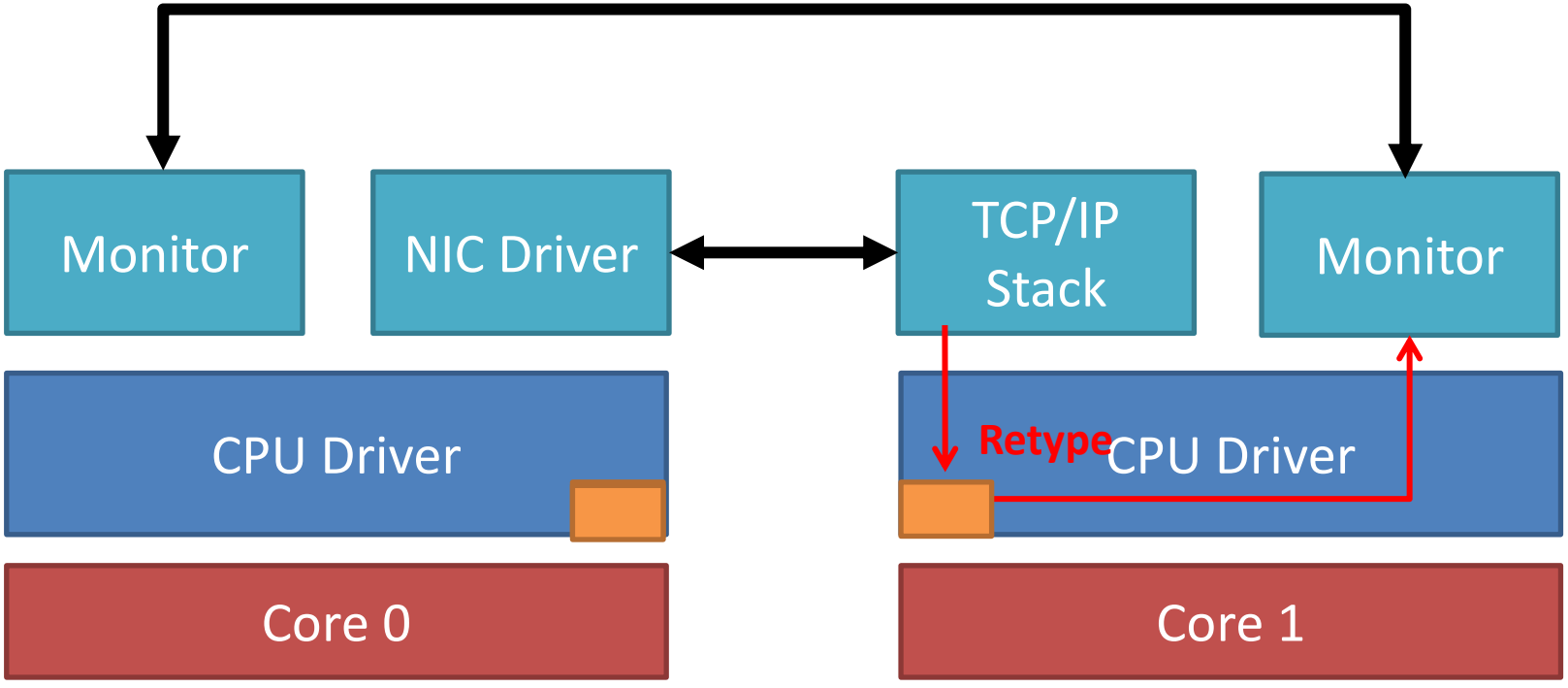
Cross-Core Capabilities



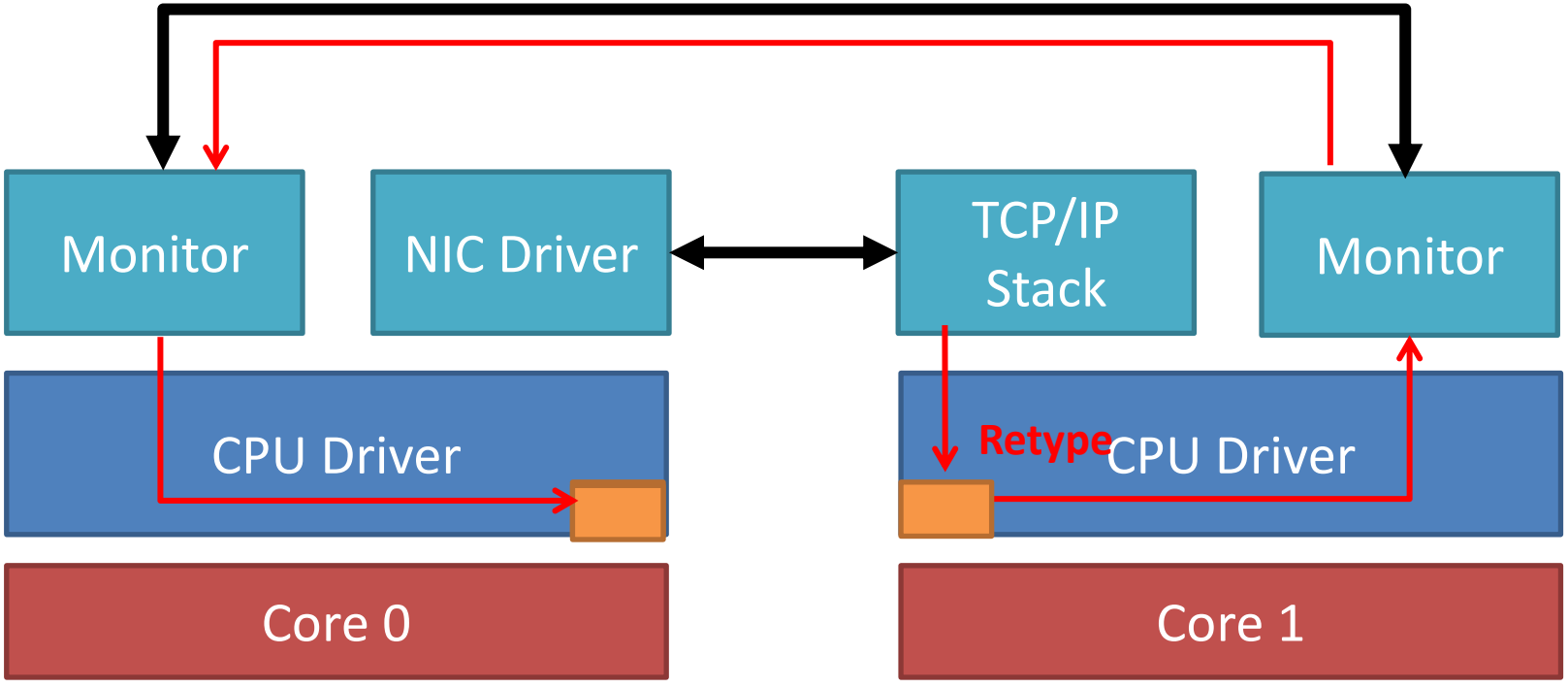
Cross-Core Capabilities



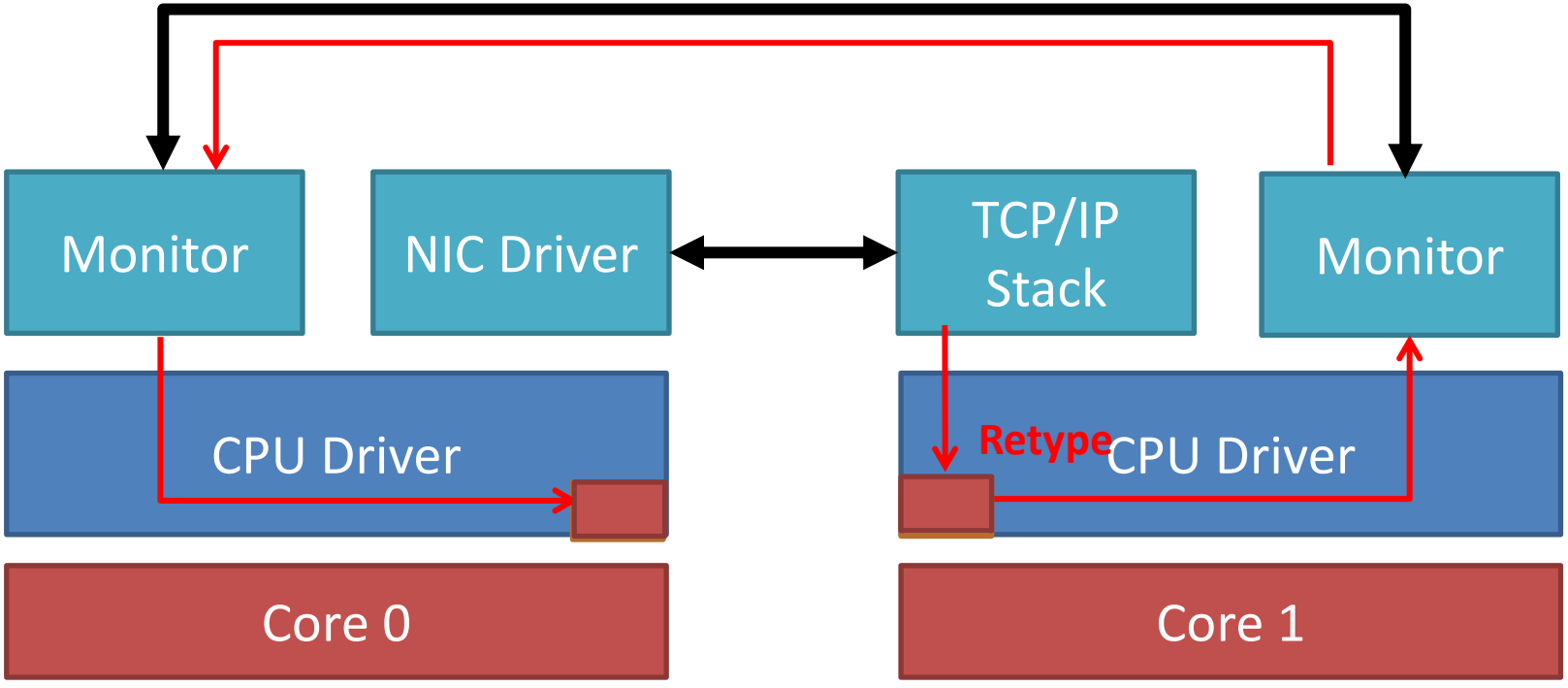
Cross-Core Capabilities



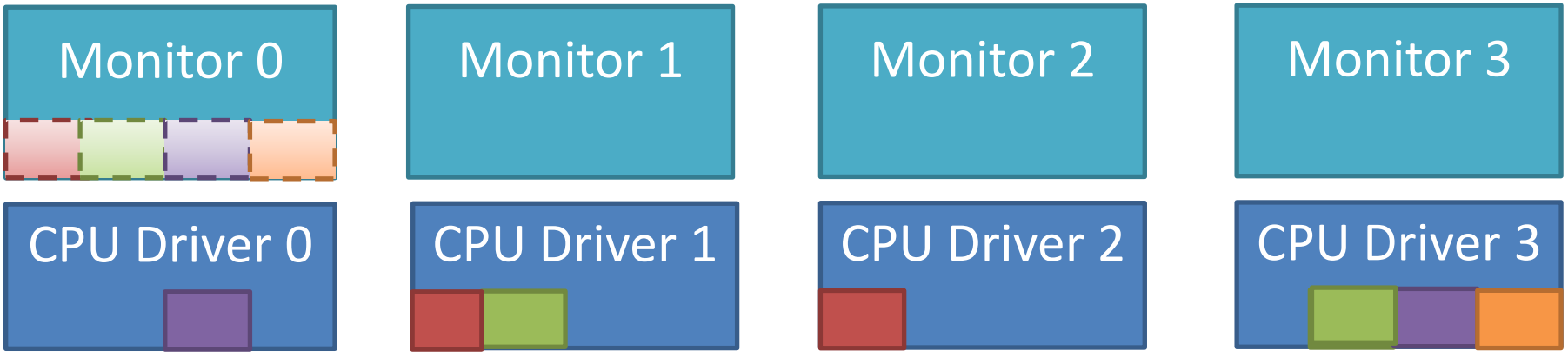
Cross-Core Capabilities



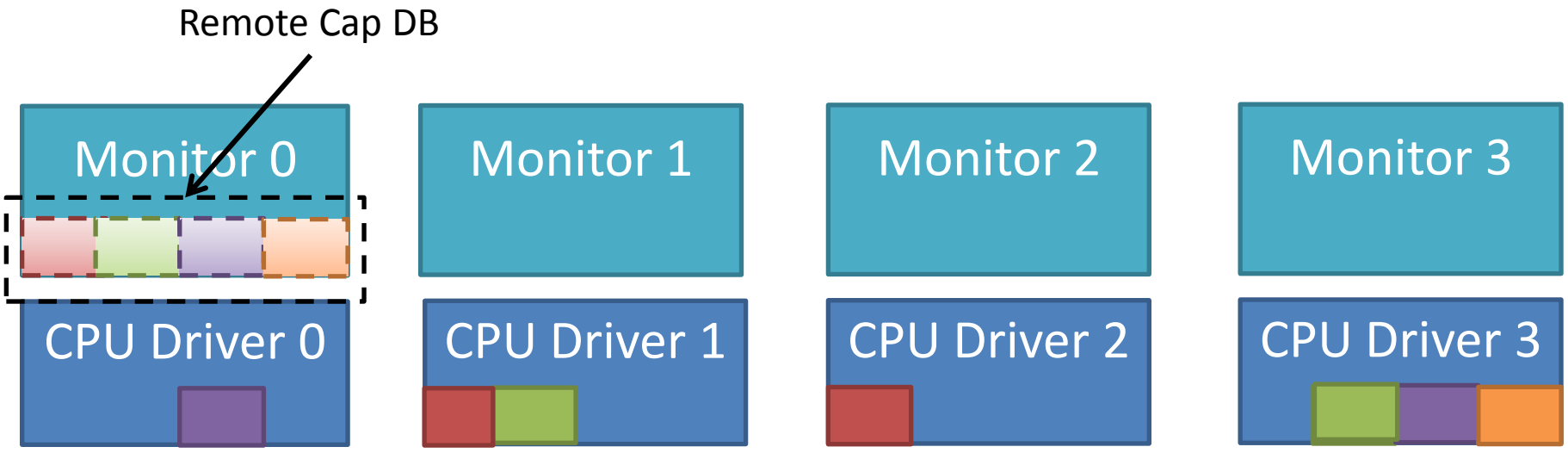
Cross-Core Capabilities



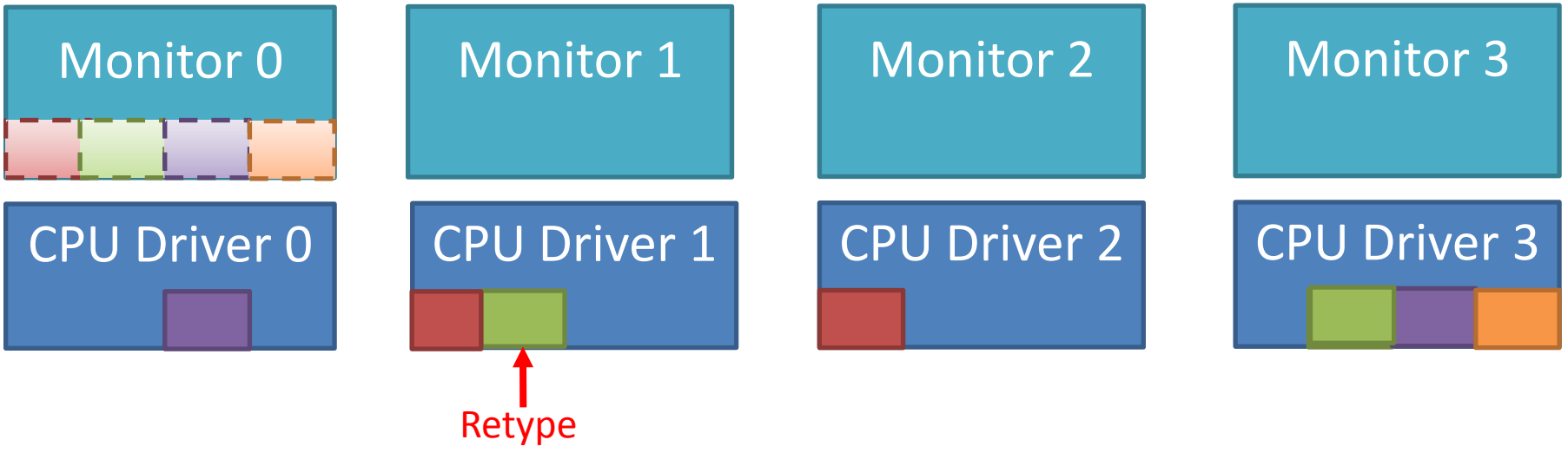
Centralised Consensus (Serializer)



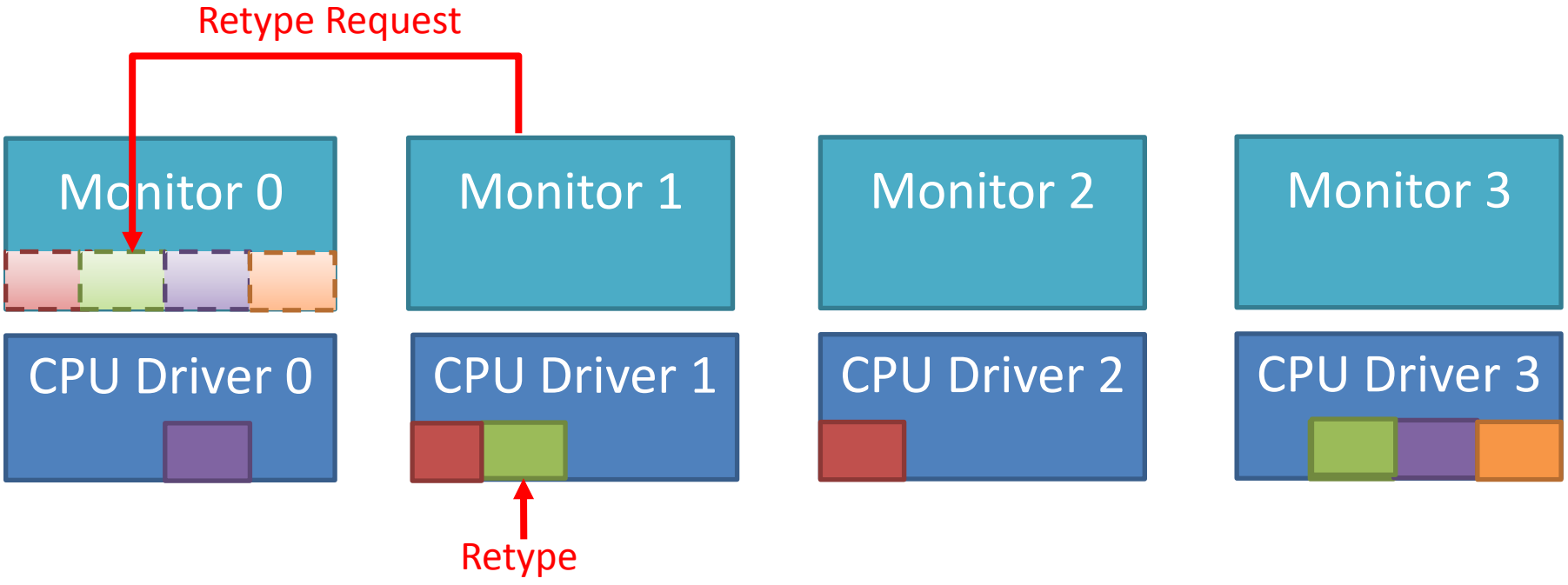
Centralised Consensus (Serializer)



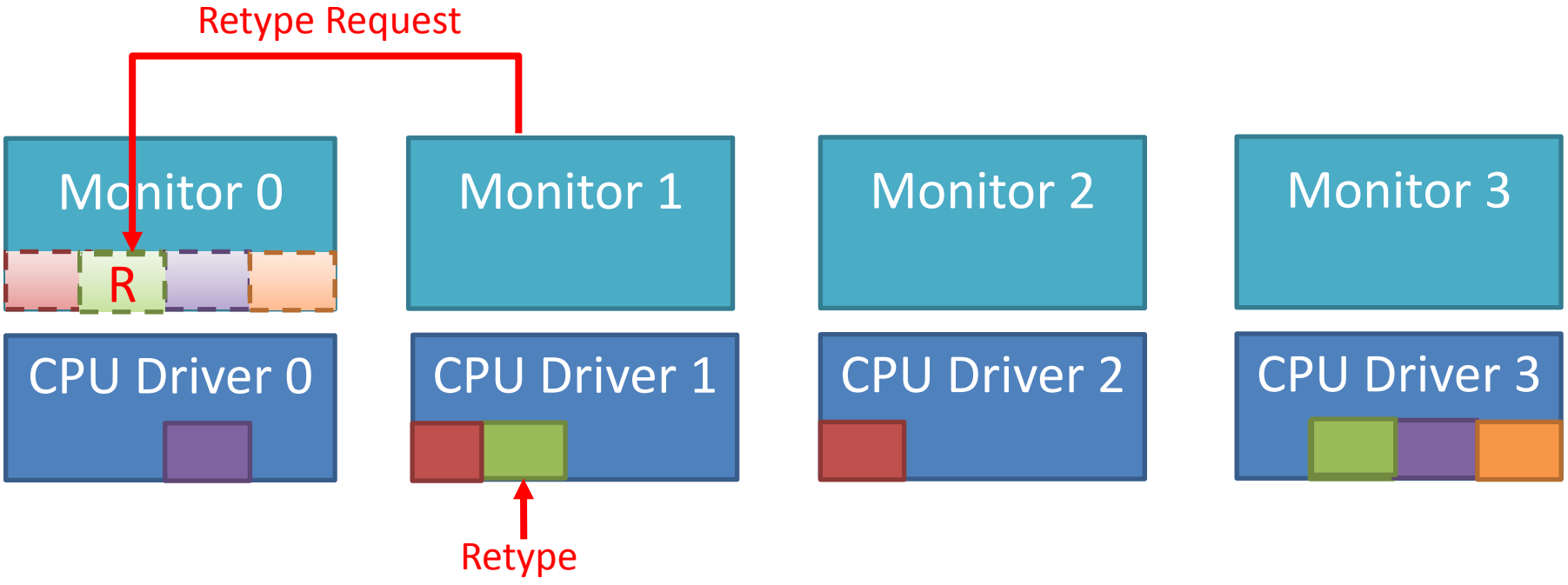
Centralised Consensus (Serializer)



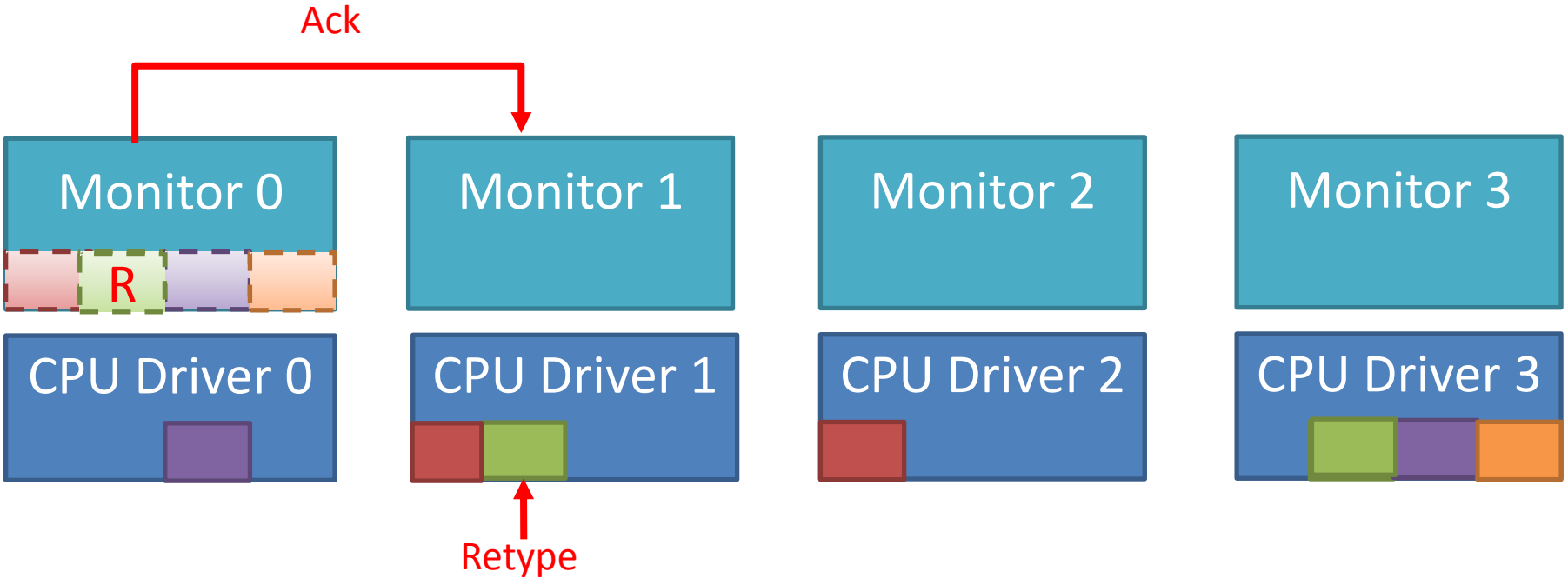
Centralised Consensus (Serializer)



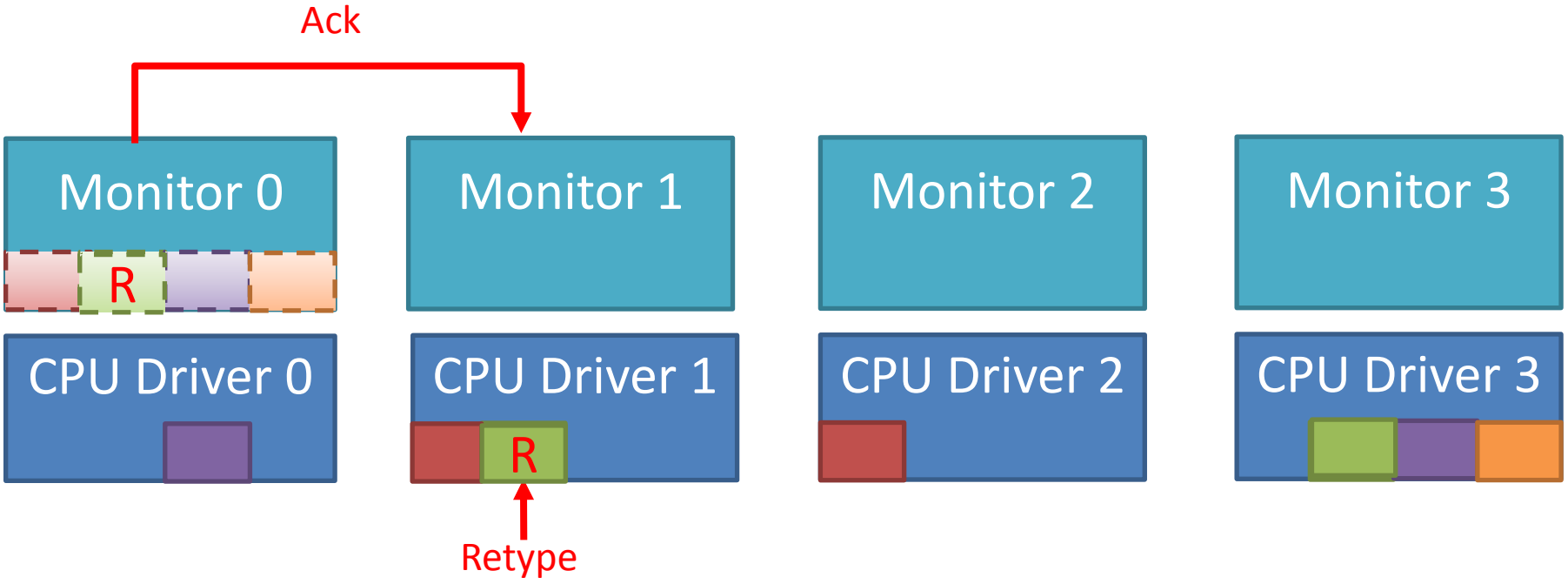
Centralised Consensus (Serializer)



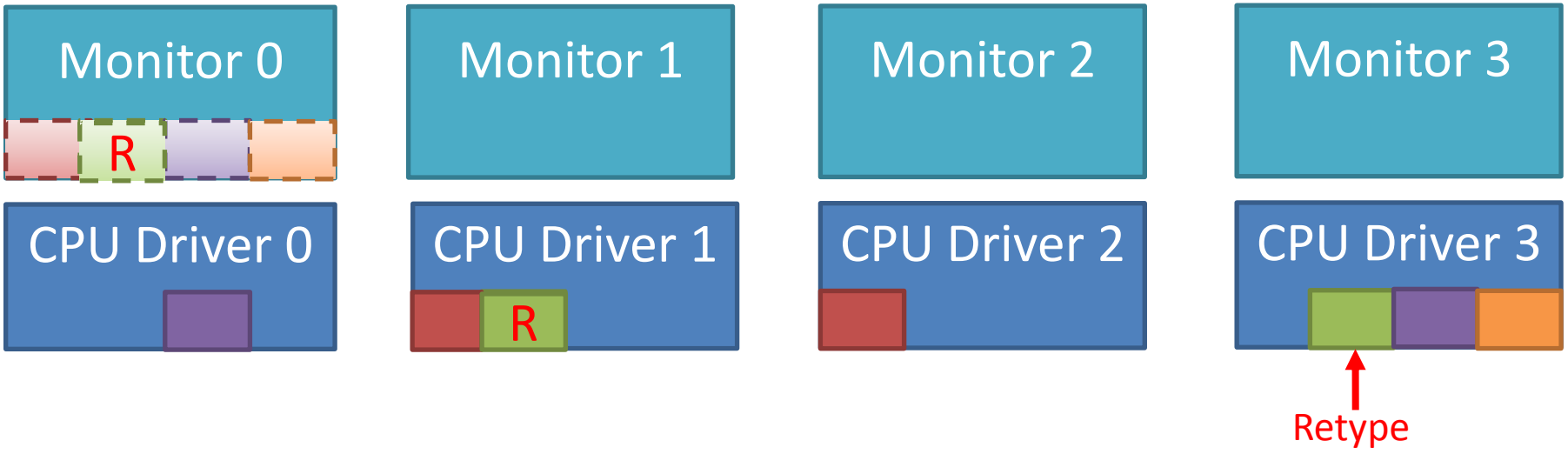
Centralised Consensus (Serializer)



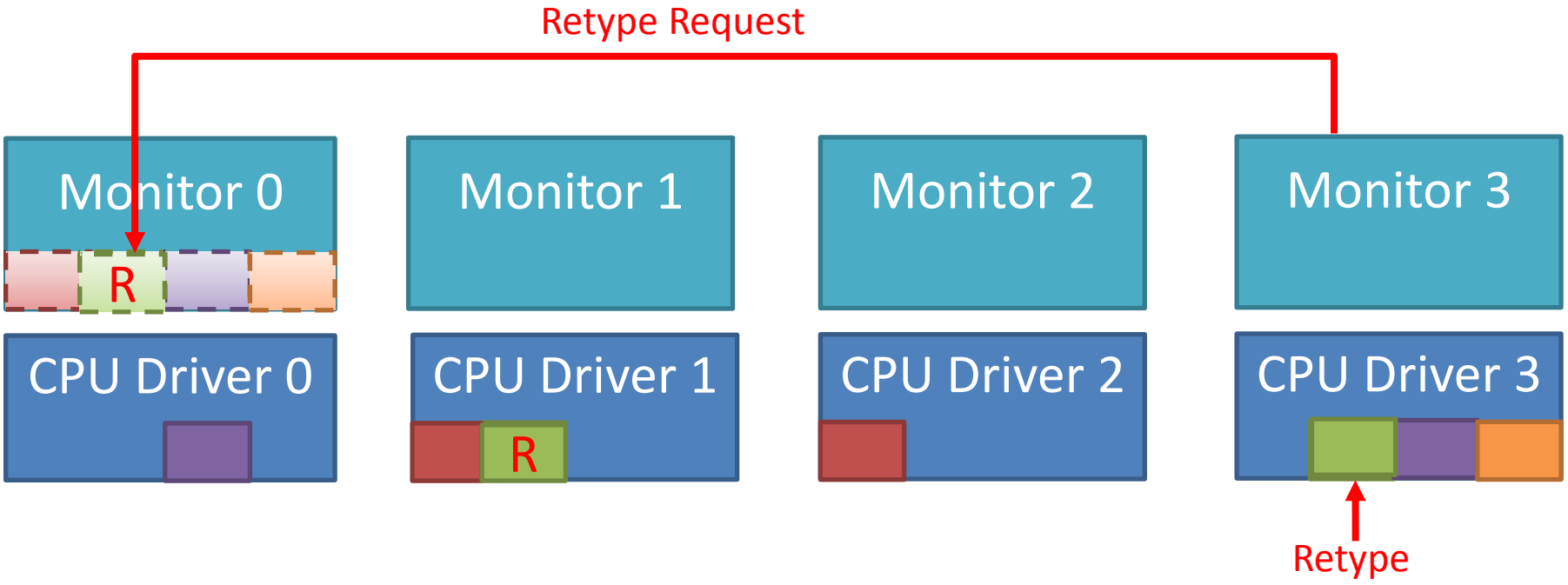
Centralised Consensus (Serializer)



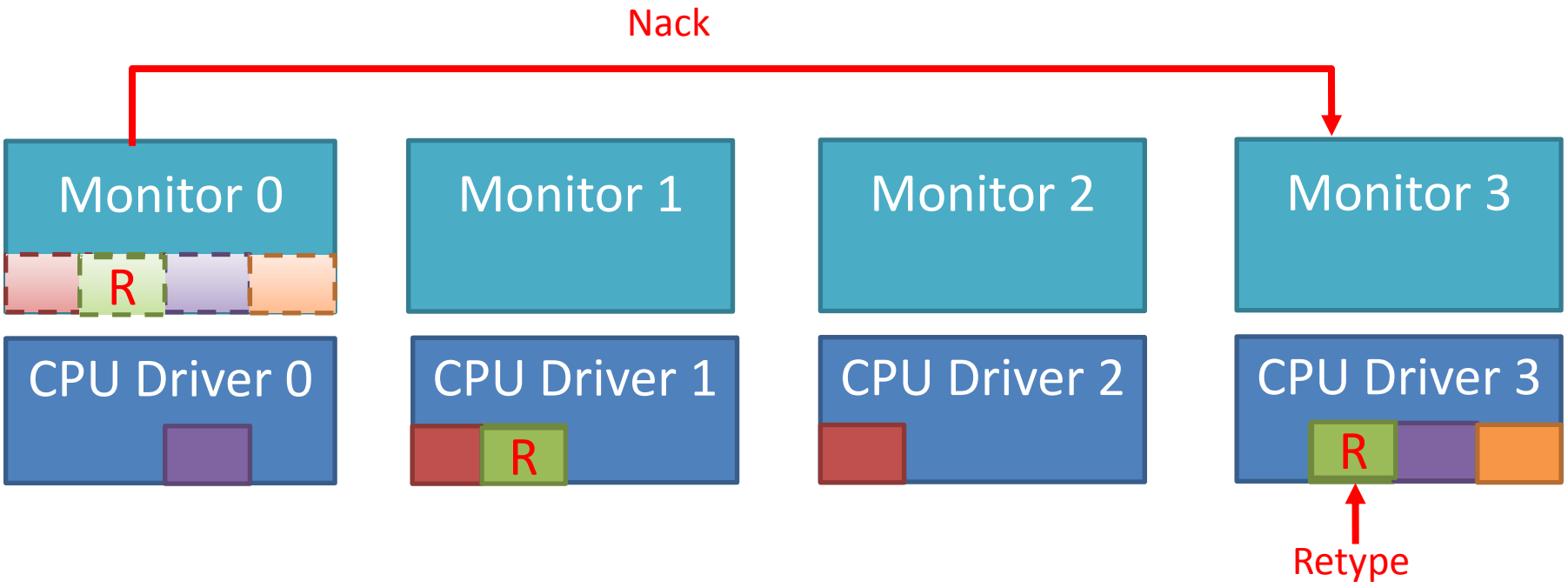
Centralised Consensus (Serializer)



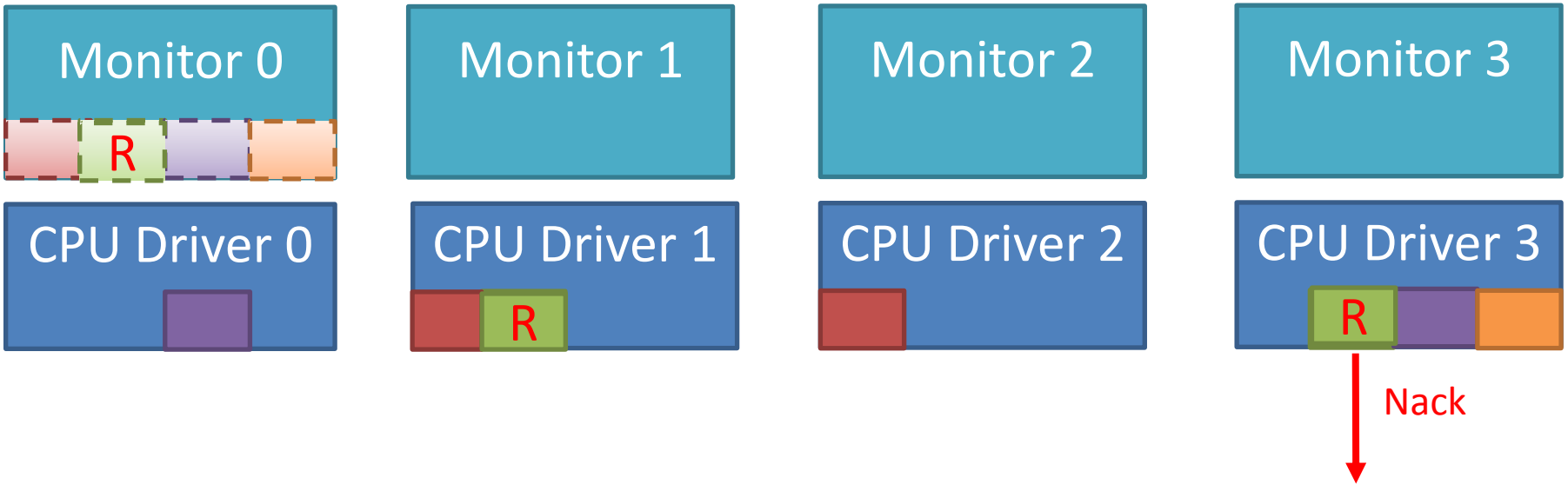
Centralised Consensus (Serializer)



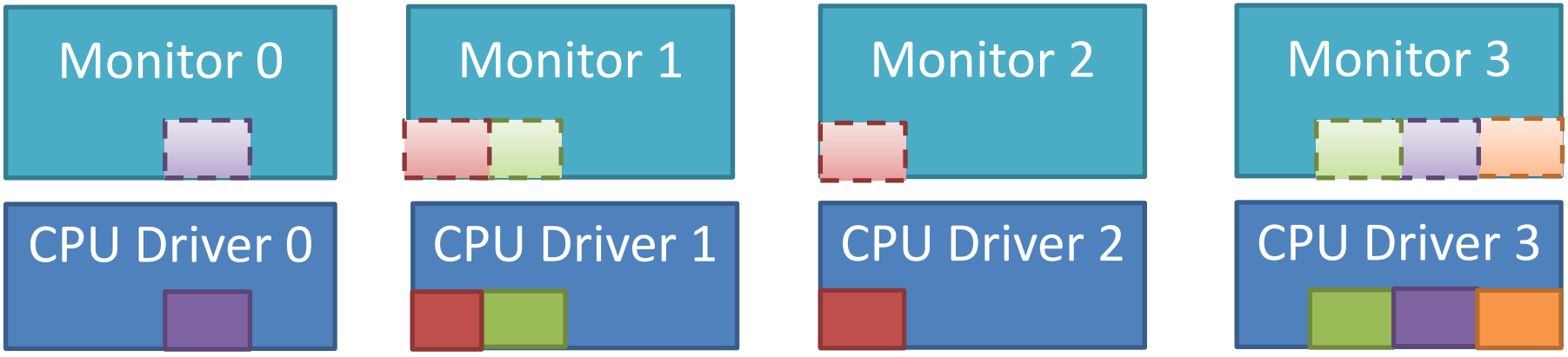
Centralised Consensus (Serializer)



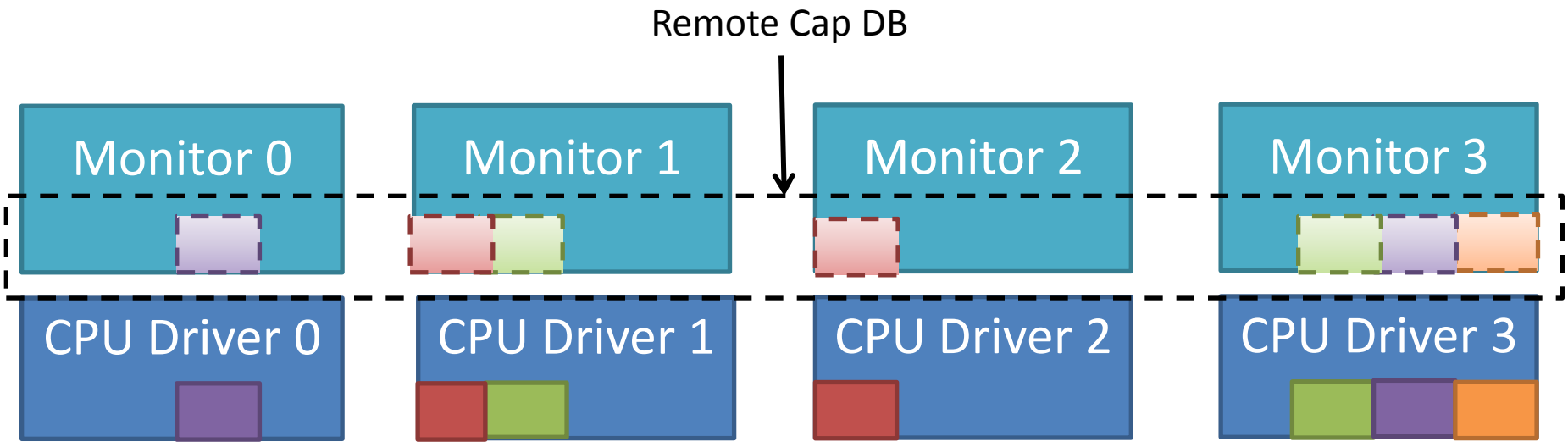
Centralised Consensus (Serializer)



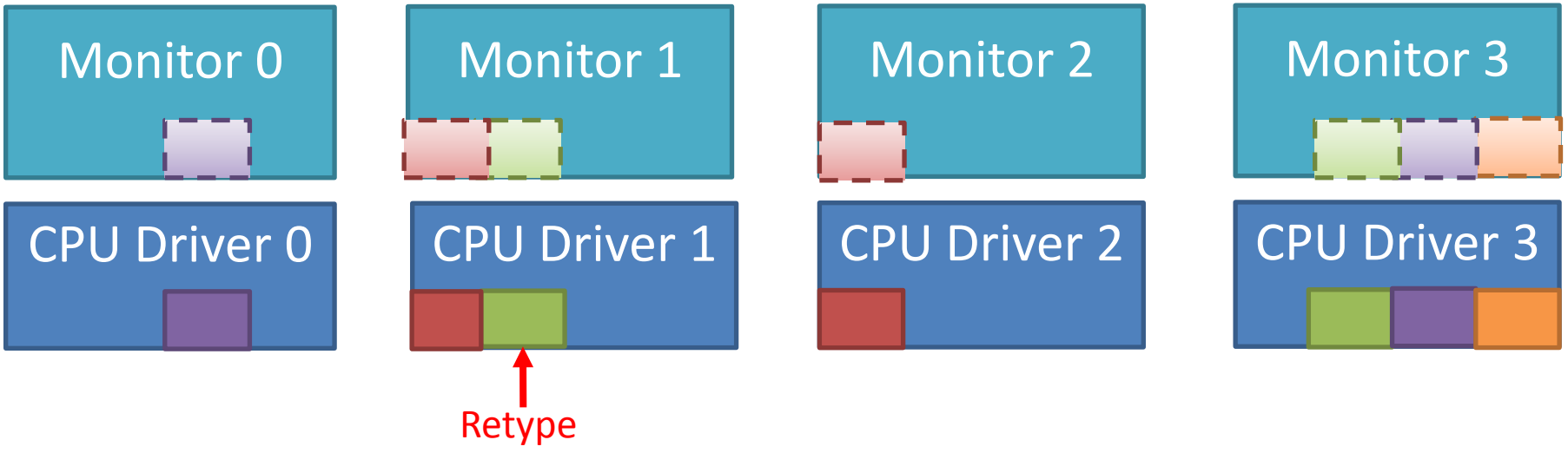
Two Phase Commit



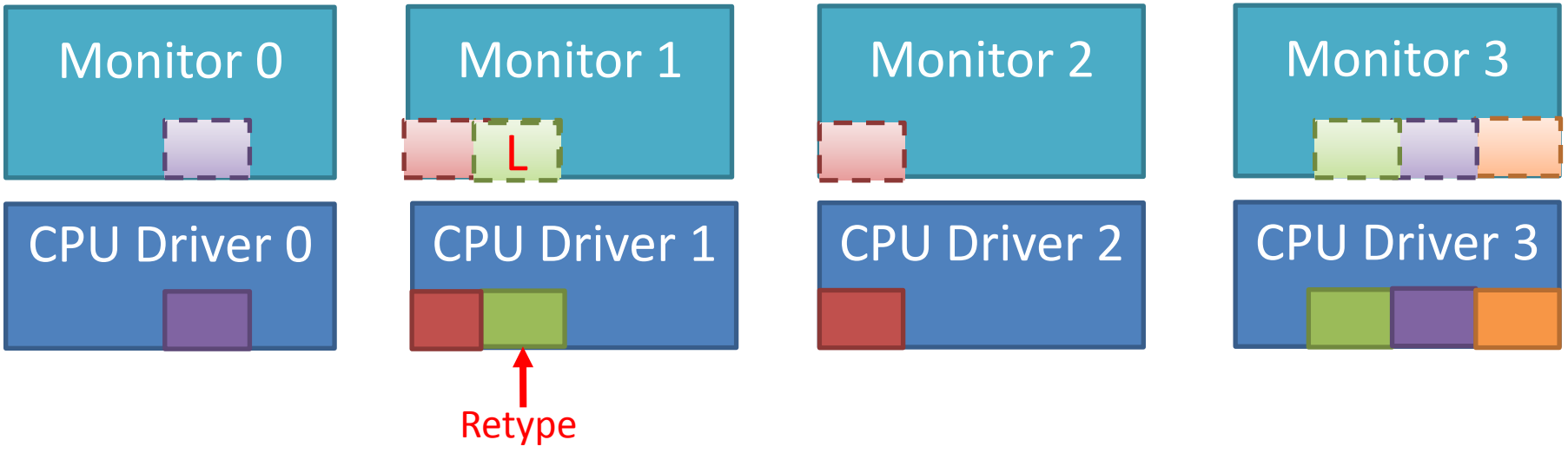
Two Phase Commit



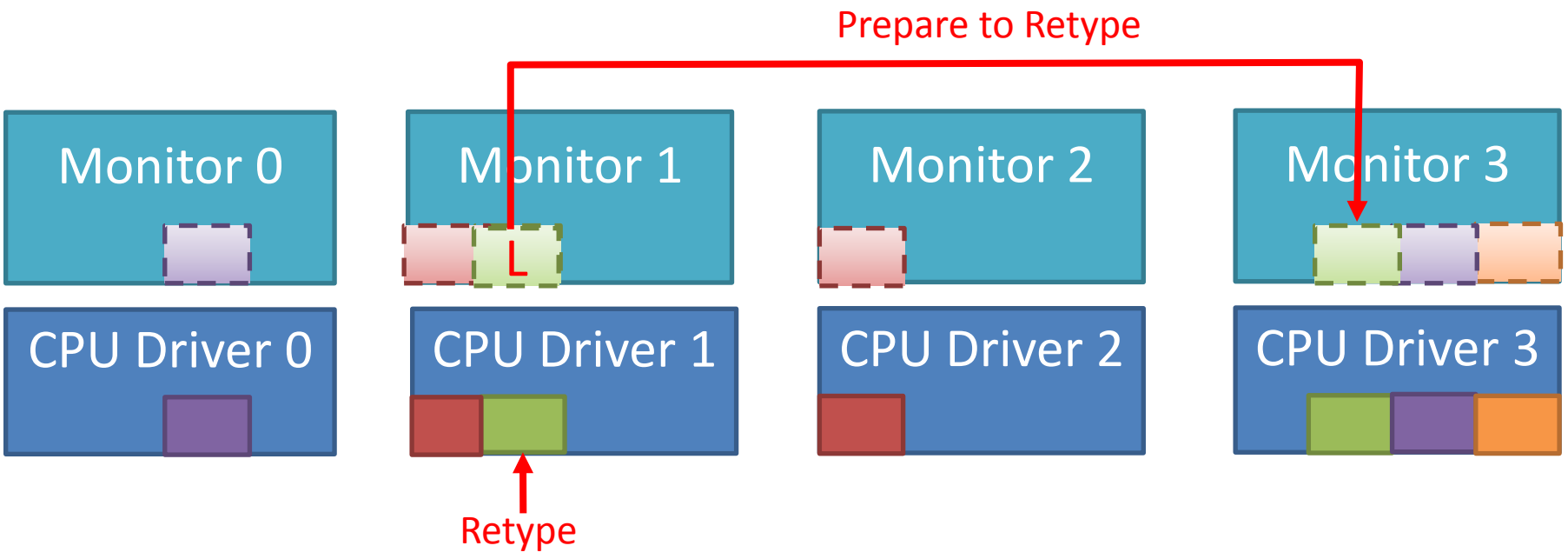
Two Phase Commit



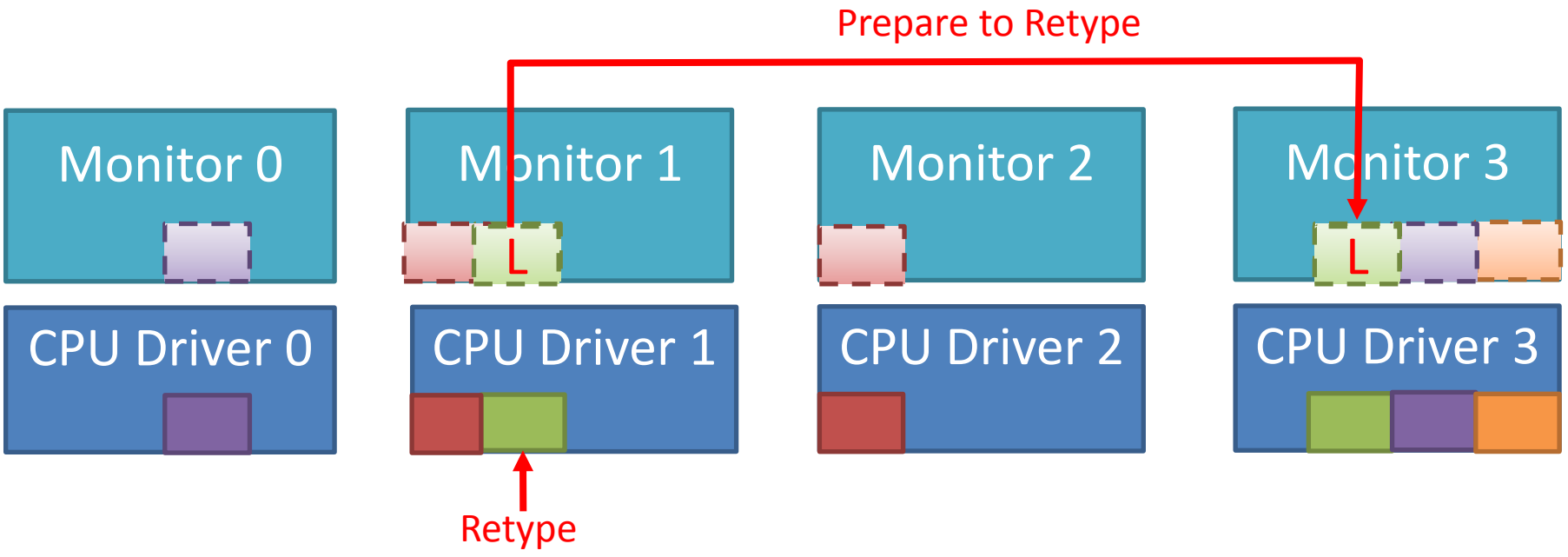
Two Phase Commit



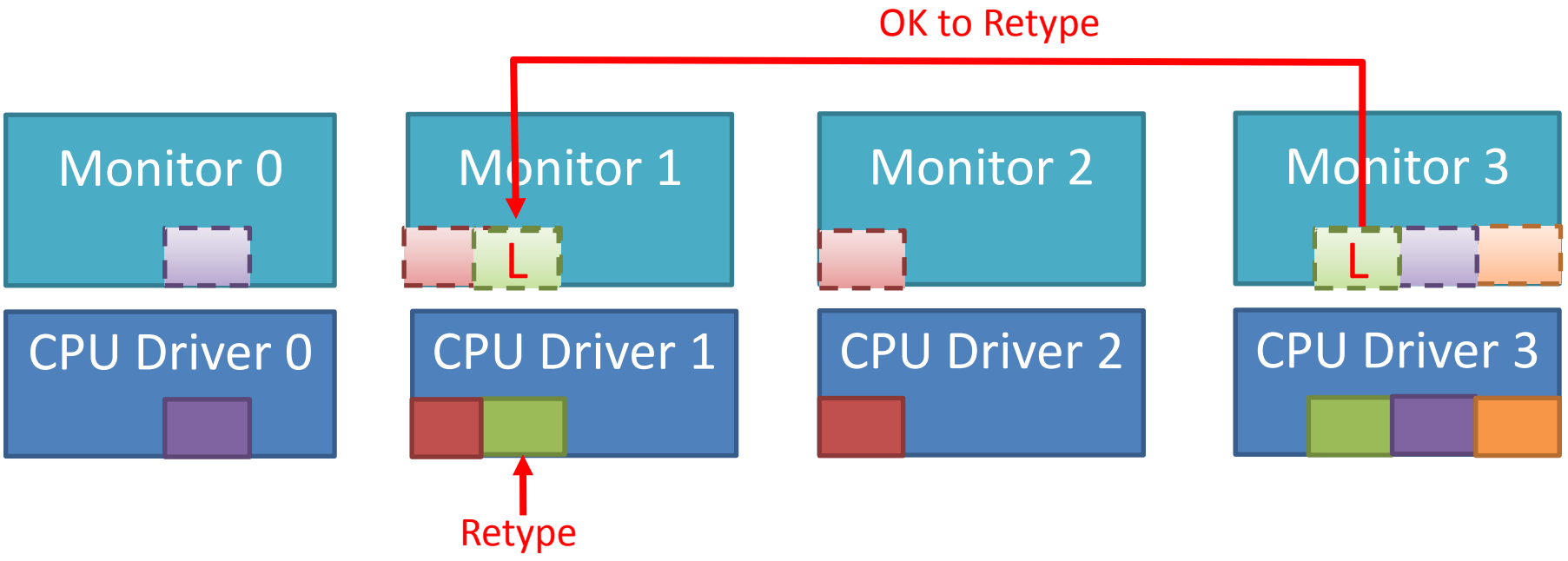
Two Phase Commit



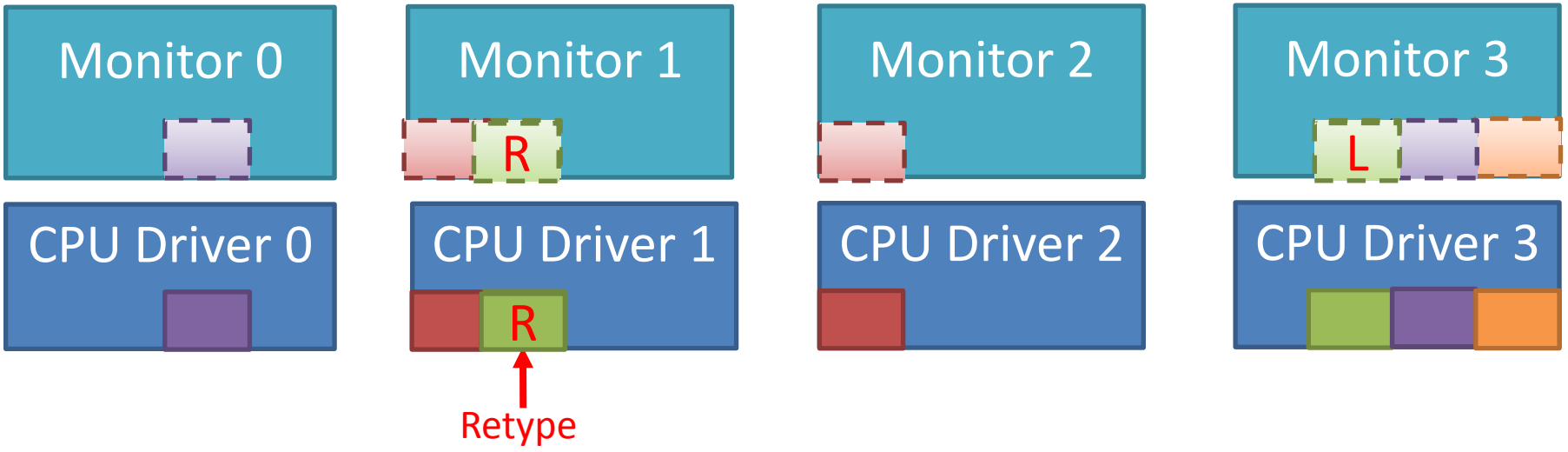
Two Phase Commit



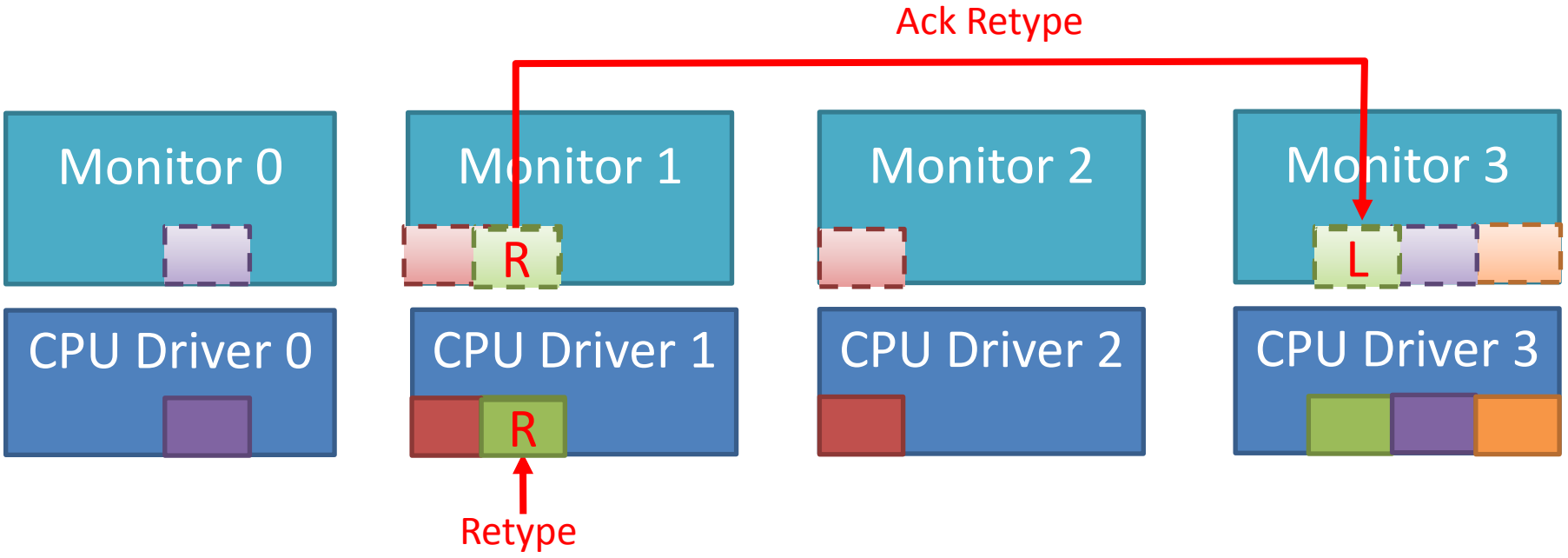
Two Phase Commit



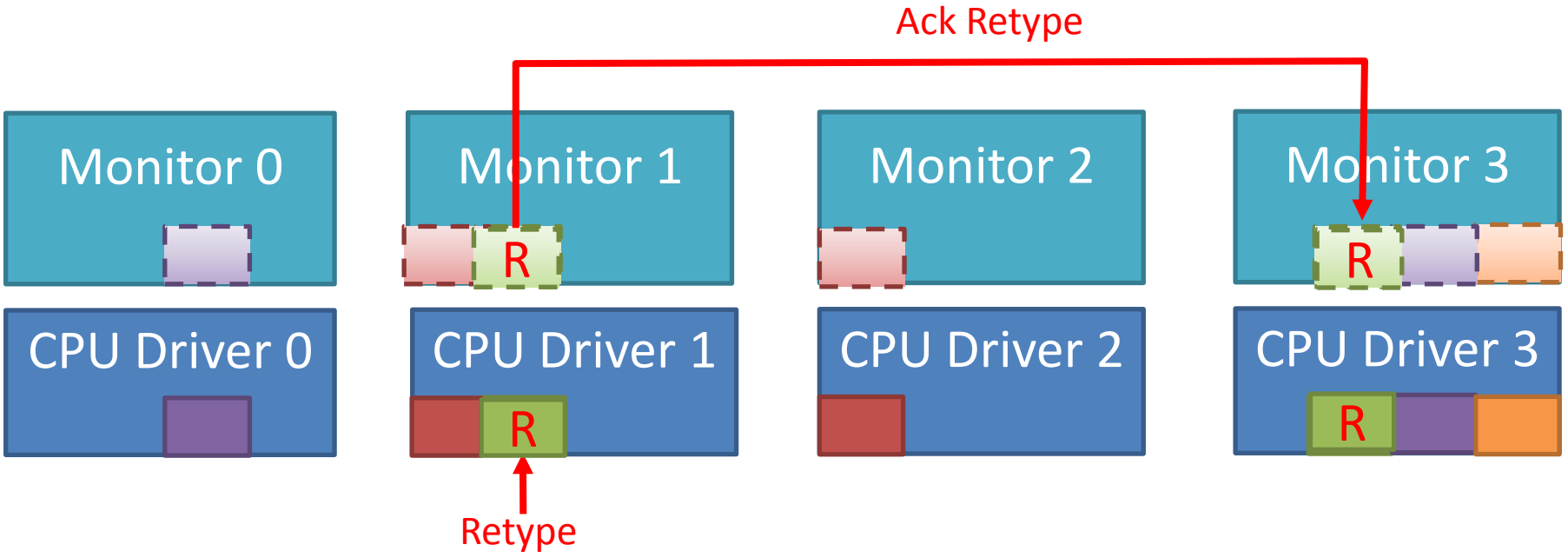
Two Phase Commit



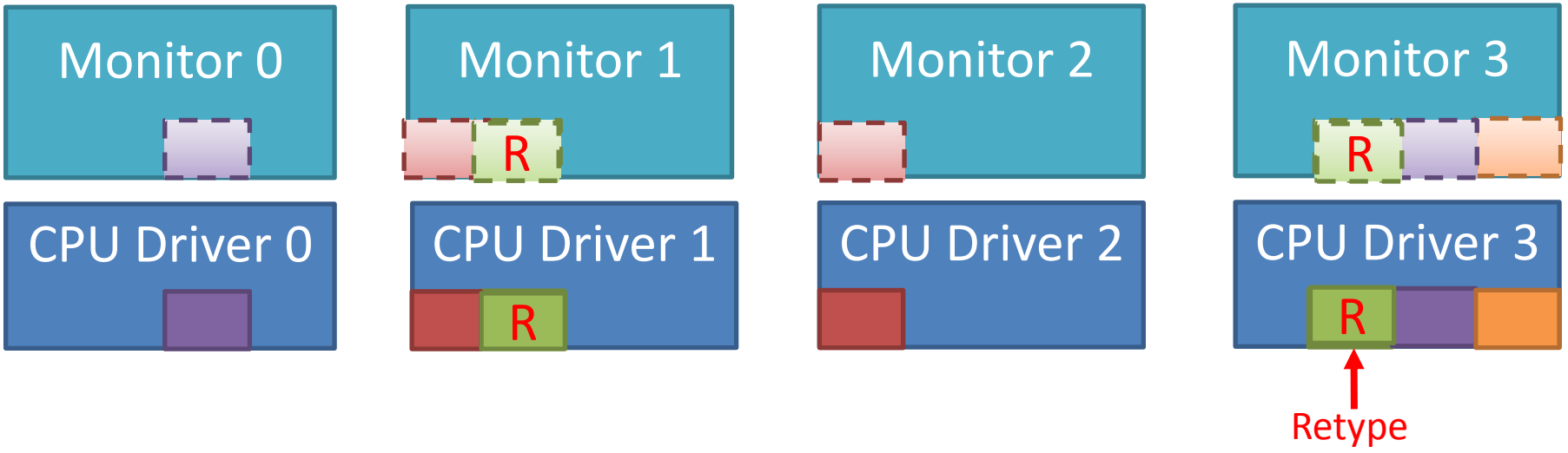
Two Phase Commit



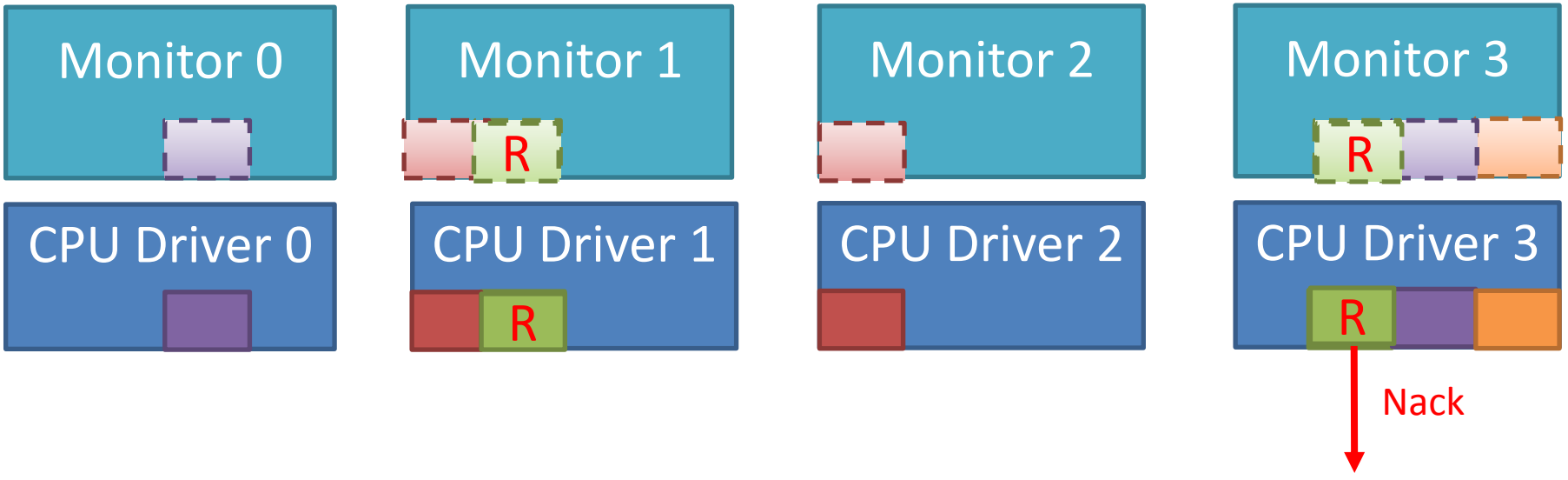
Two Phase Commit



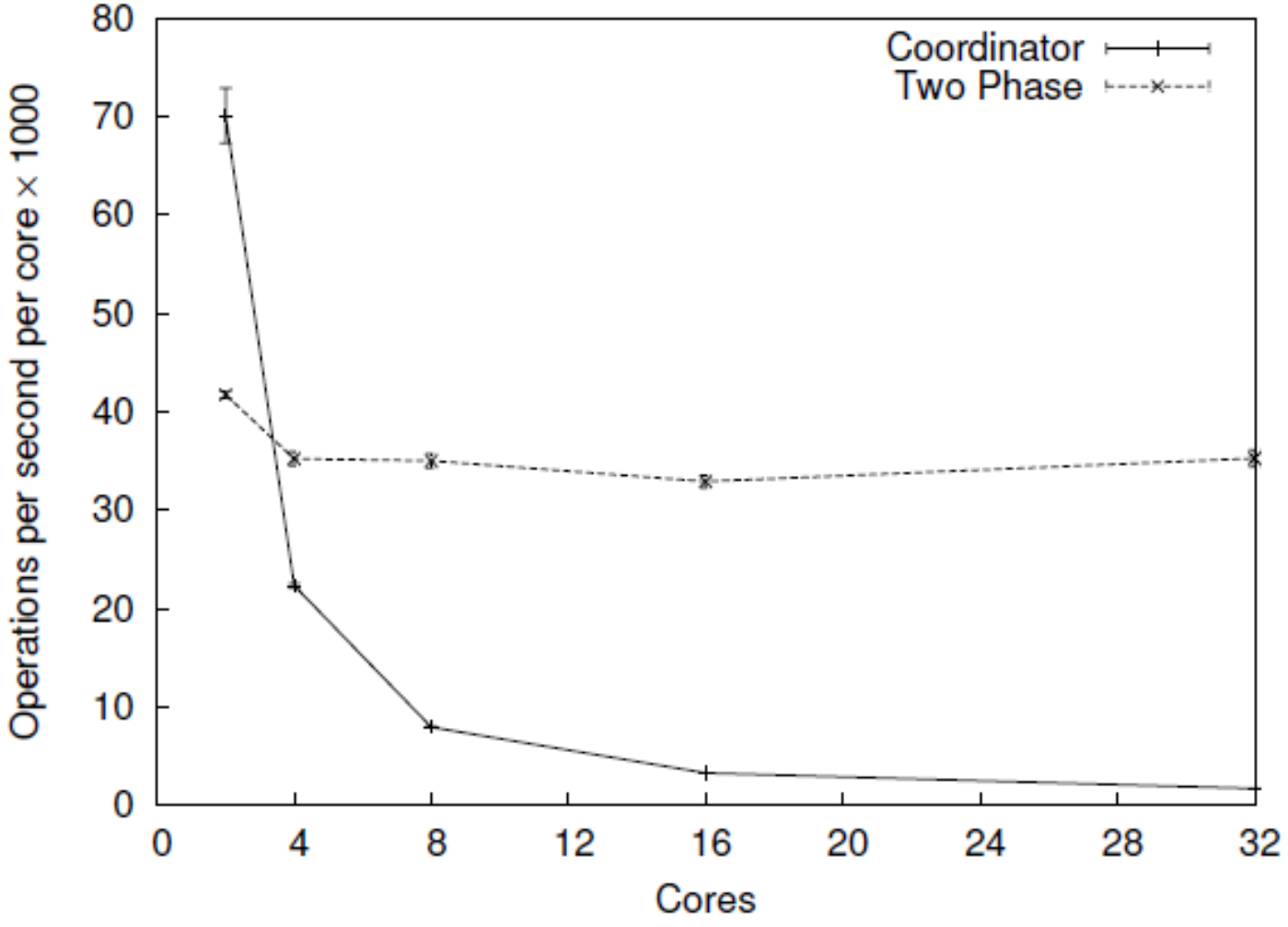
Two Phase Commit



Two Phase Commit



Scalability of Cross-Core Coordination



How You Might Interact With Caps

- Mapping a frame into the address space of two different domains:

```
frame_alloc(&framecap, size, &actual_size);
<transfer cap>
vspace_map_one_frame(&virAddr, size, framecap,
                     &memobj, &vregion);
```

How You Might Interact With Caps

- Map a buffer for a hardware device:

```
bufferAddr = alloc_map_frame(FLAGS, size, &framecap);
frame_identify(frame, &phys Addr);
```

Further Info

- Technical Note #10 – Spec
- Debugging:
  - debug_cspace()
  - print_cspace shell command in fish
- Code:
  - lib/barrelfish/capabilities.c
  - kernel/capabilities.c
  - usr/monitor/rcap_db_twopc.c
- Based upon seL4 capability model:
  - http://ertos.nicta.com.au/research/sel4/