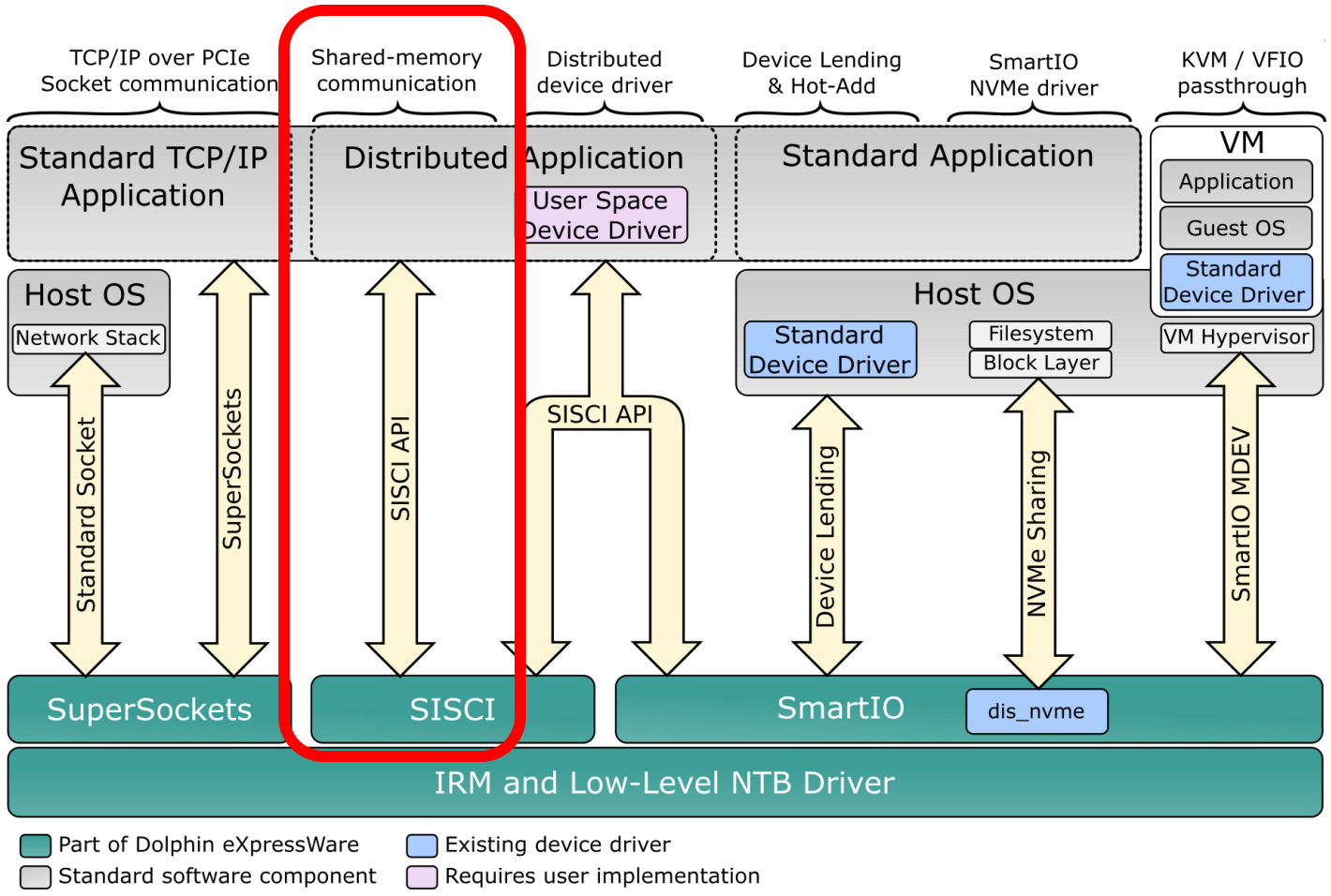
IN5050: SISCI API

Lars Bjørlykke Kristiansen

Dolphin Interconnect Solutions

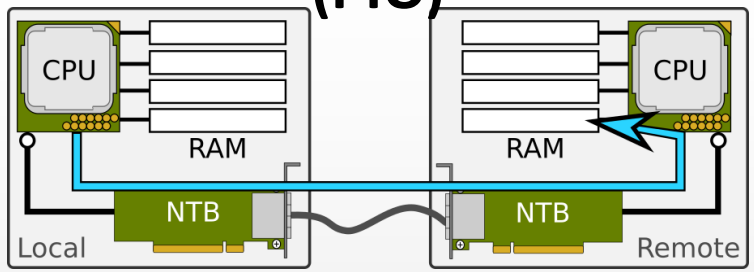
28th April 2026




Transp-02
Mex5-1
LarsCH
MXSW-03
PXMXS-40
MXMXS-20
MXMXS-10
PXMXS-31
PXMXS-45

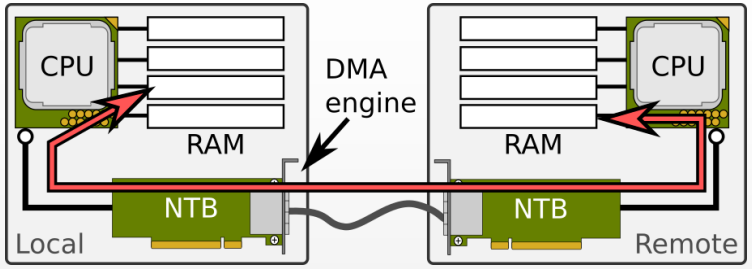
Dolphin provides a shared-memory API on top of PCIe NTBs called SISCI

Memory-mapped access (PIO)



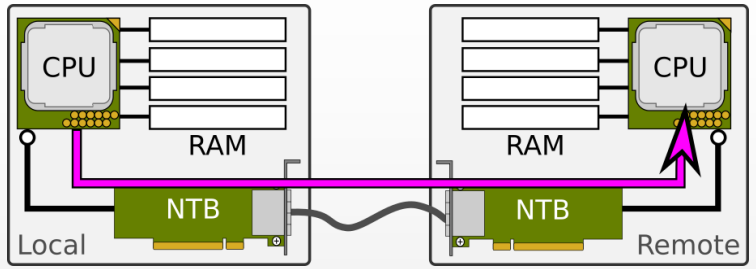
- Remote memory with pointers
- Medium data throughput
- Sub-microsecond latency (100s of nanoseconds)
- CPU Heavy

Adapter DMA



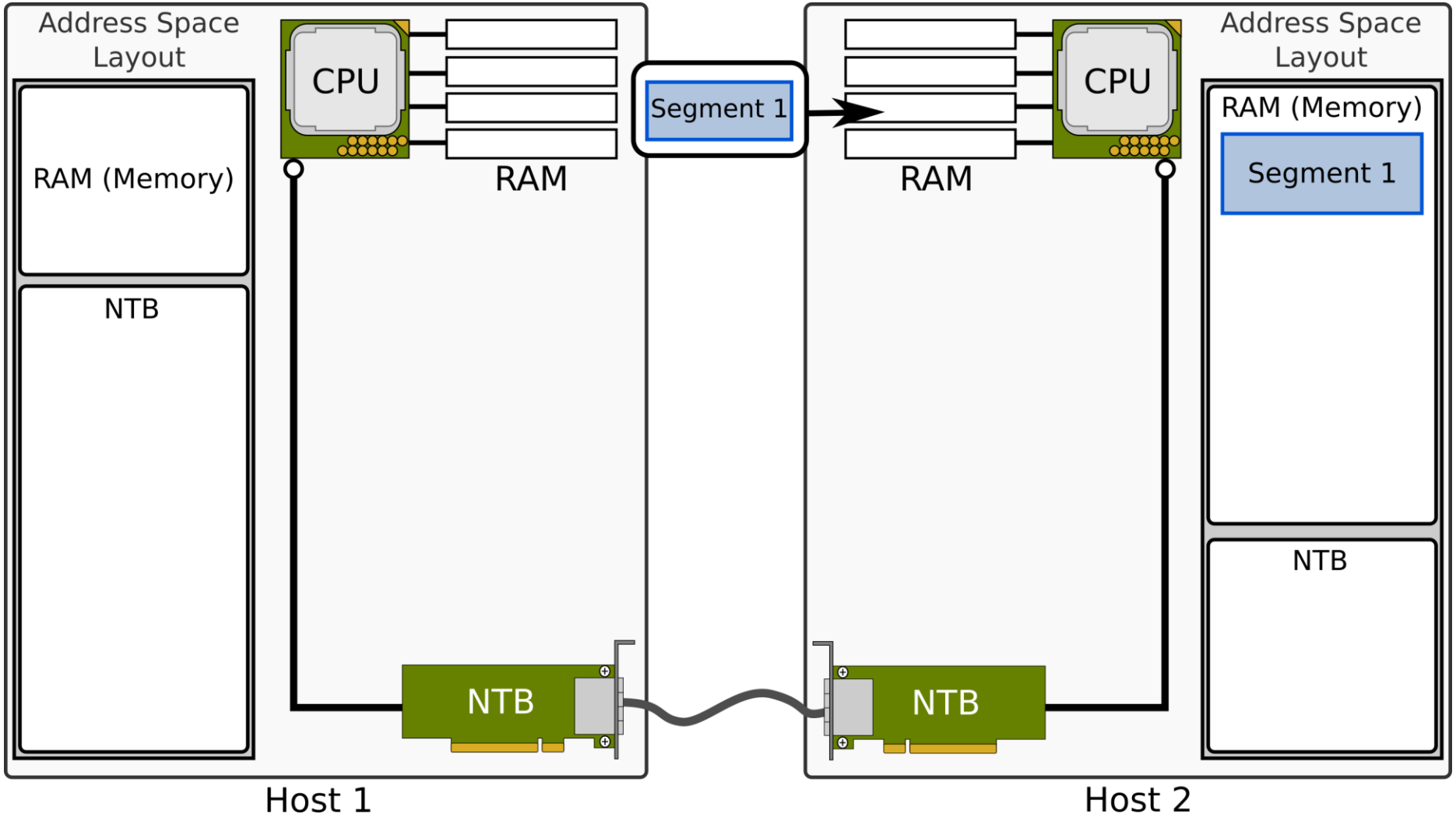
- Copy data with API call
- High data throughput
- Offload data transfer to adapter -> Low CPU load
- Can be used for pipelining data streaming

Remote interrupts

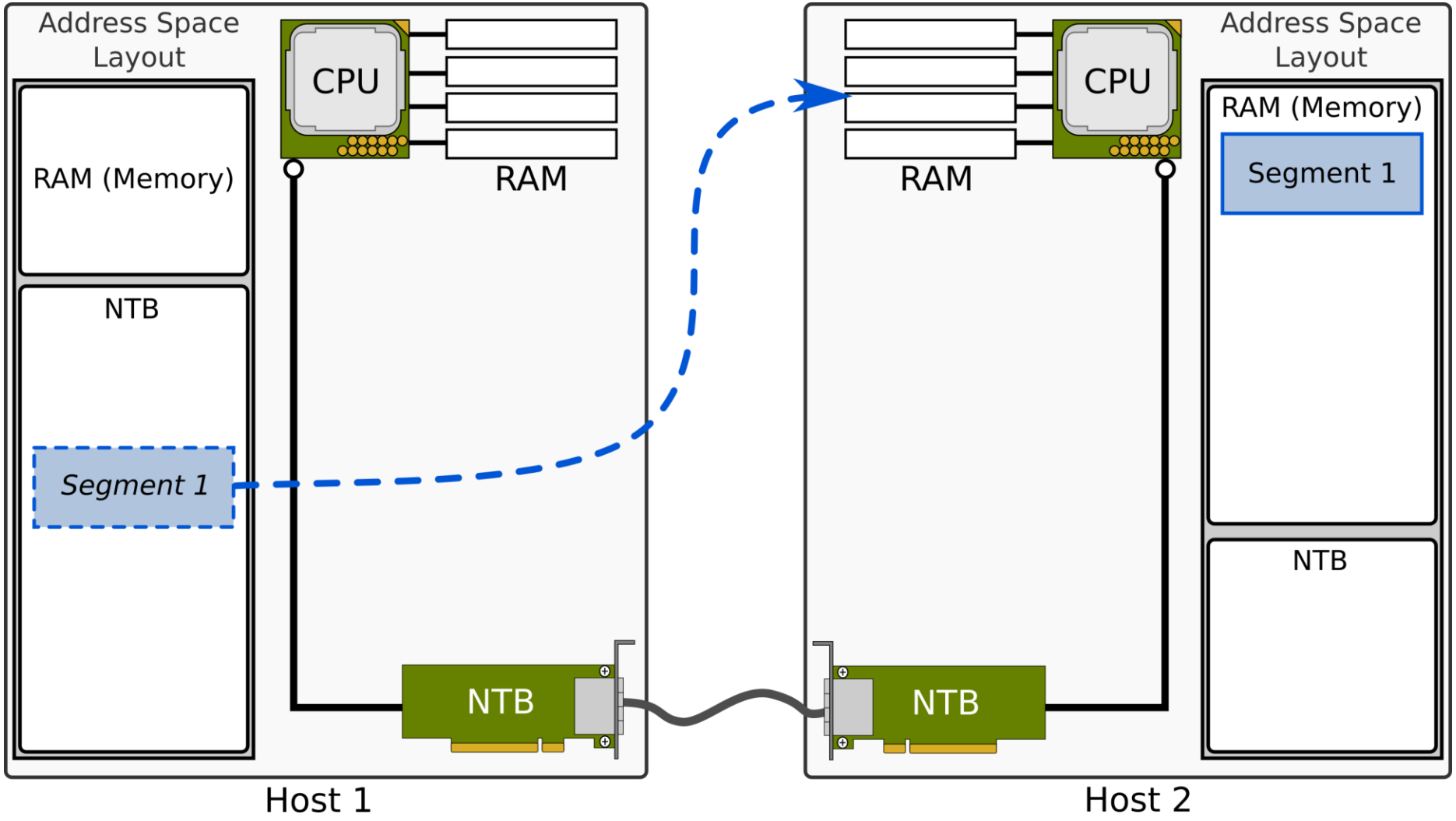


- Trigger callback function on remote node with API call
- Medium latency
- Can send small message with data interrupts

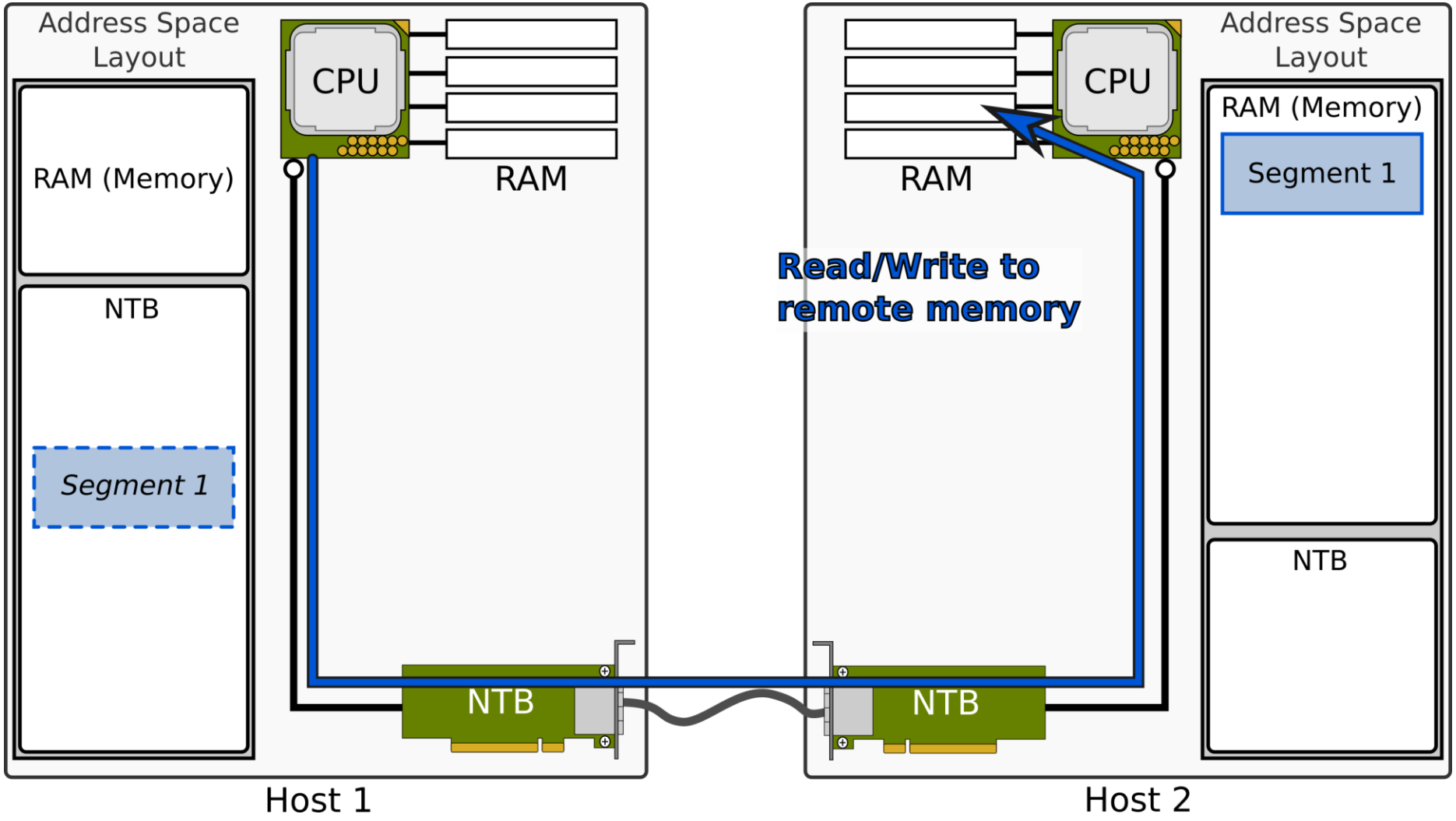
PIO: SCICreateSegment + SCIPrepareSegment + SCISetSegmentAvailable



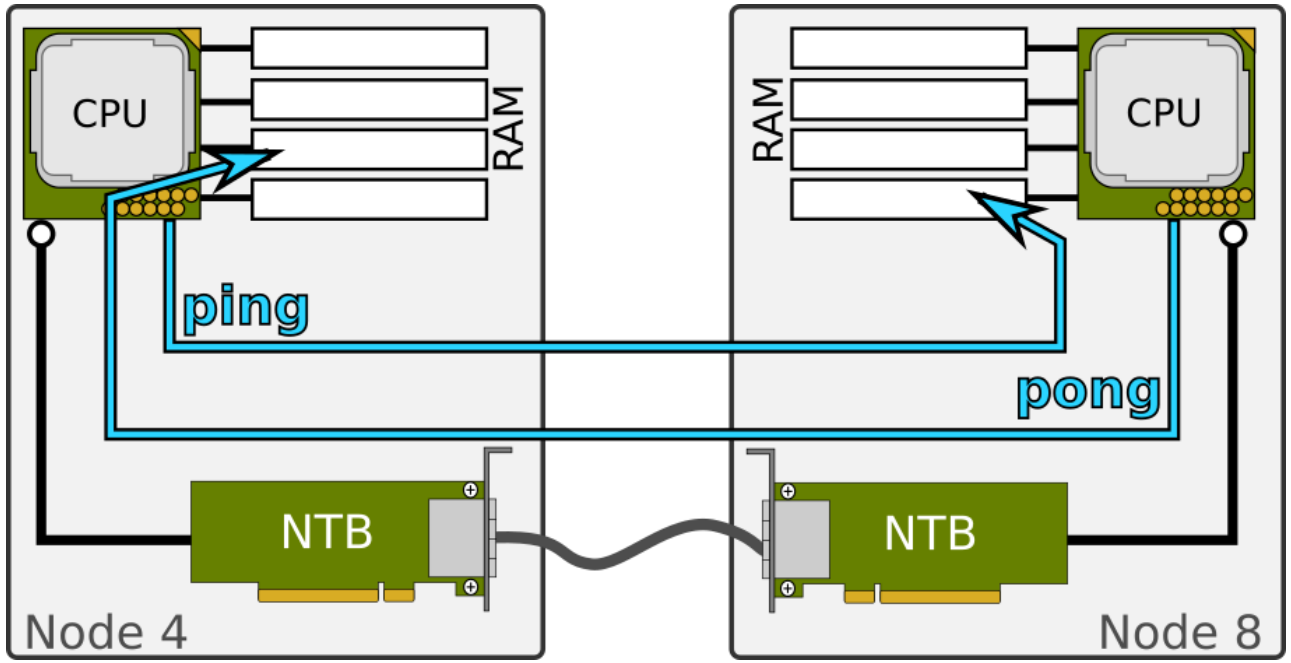
PIO: SCIConnectSegment + SCIMapRemoteSegment



PIO: remote_segment[0] = 0x00112233



Demo: PIO latency with scipp




```
for (i=1; i < 1000; i++) {
 remote_seg2[0] = i;  // ping
 while(local_seg1[0] != i) {} // busy wait
}
```

ping

```
for (i=1; i < 1000; i++) {
 while(local_seg2[0] != i) {} // busy wait
 remote_seg1[0] = i; // pong
}
```

pong

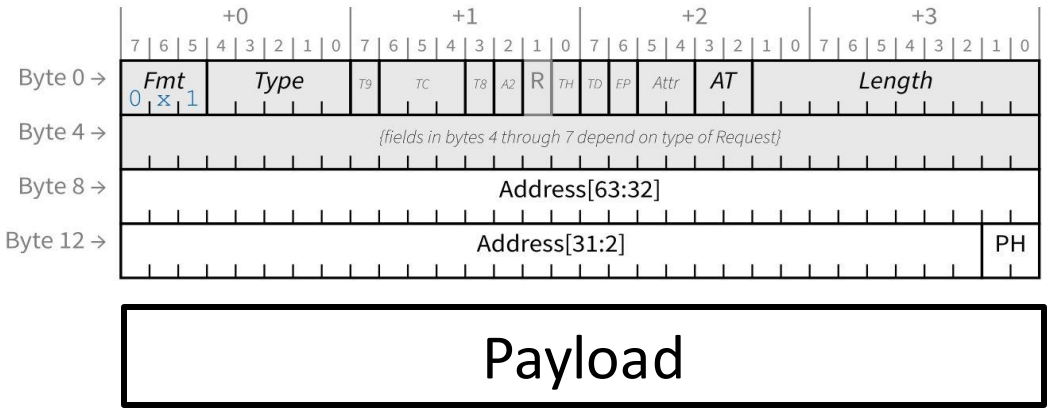
Problem: PCIe uses packets (TLPs) with headers, but this can lead to performance overhead for small reads/writes

If each memory write generates a TLP…

```
uint32_t* remote = SCIMapRemoteSegment();

for (i = 0; i < 4; i++) {
    remote[i] = i;
}
```





16 bytes header per TLP

4 bytes data payload per TLP

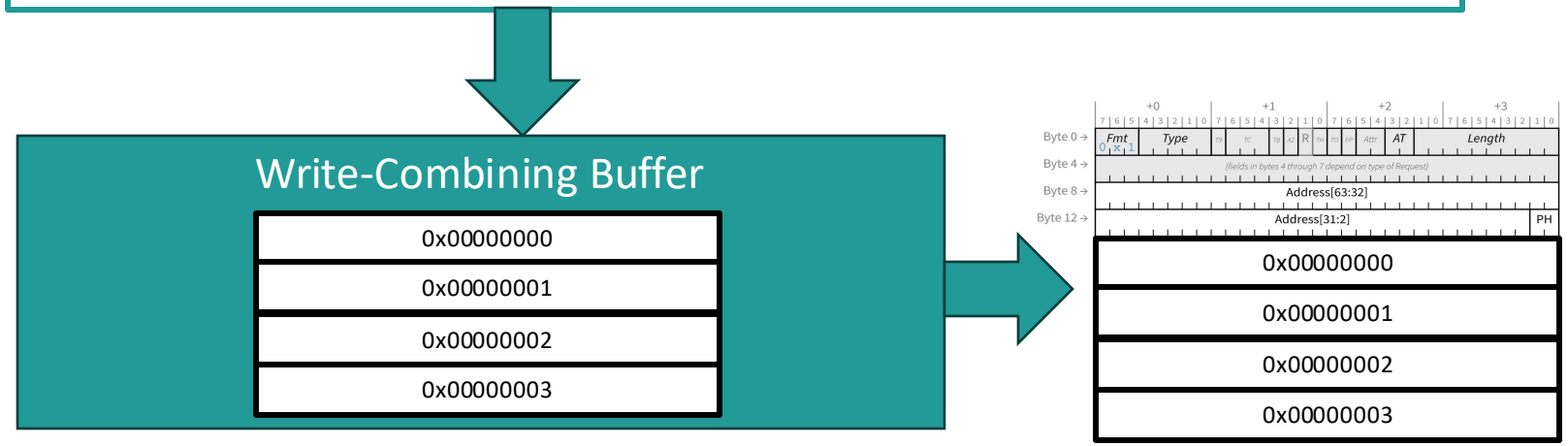
…64 bytes of headers vs 16 bytes of data

Solution: Write Combining

```
uint32_t* remote = SCIMapRemoteSegment();

for (i = 0; i < 4; i++) {
    remote[i] = i;
}

SCIFlush();
```



Write Combining buffer combines writes to a single 64 bytes TLP

Solution: Write Combining

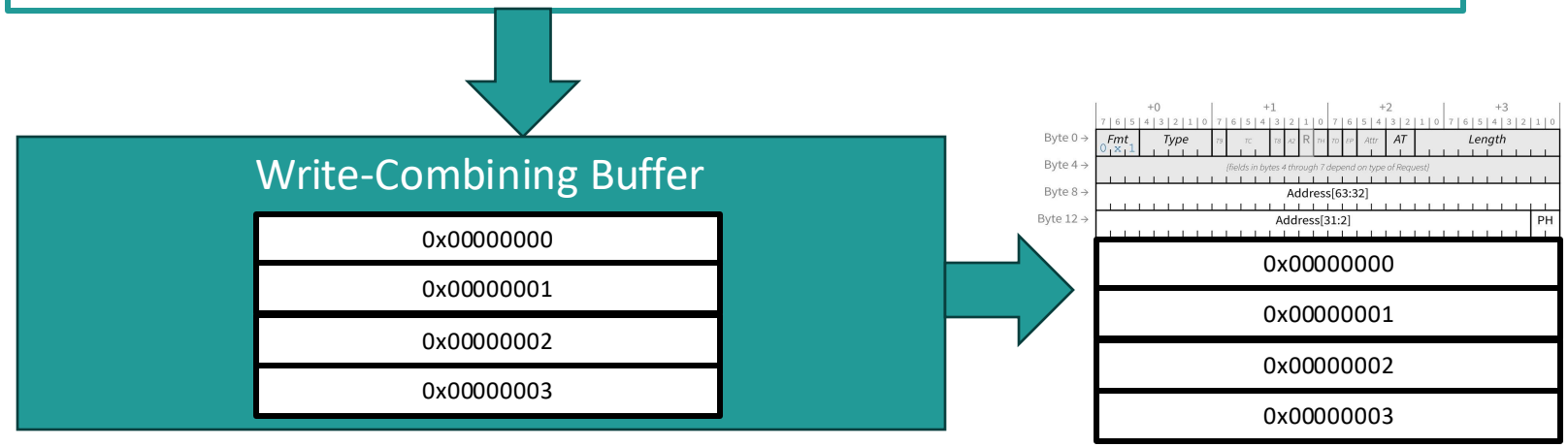
```
uint32_t* remote = SCIMapRemoteSegment();

for (i = 0; i < 4; i++) {
    remote[i] = i;
}

SCIFlush();
```

SCIMapRemoteSegment()
enables write combining by default
(special flag required to disable it)

Flushes write combining buffer

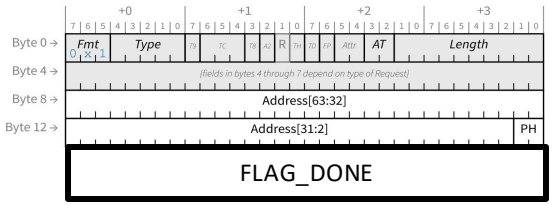


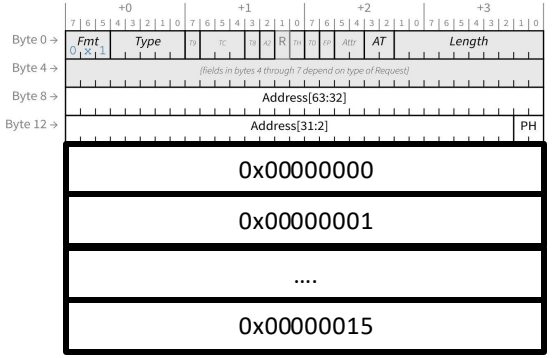
Write Combining buffer combines
writes to a single 64 bytes TLP

Beware write-ordering when using Write Combining, as Write Combining does not provide any ordering guarantees

```
for (i = 0; i < 16; i++) {
    remote[i] = i;
}
remote[16] = FLAG_DONE;
```

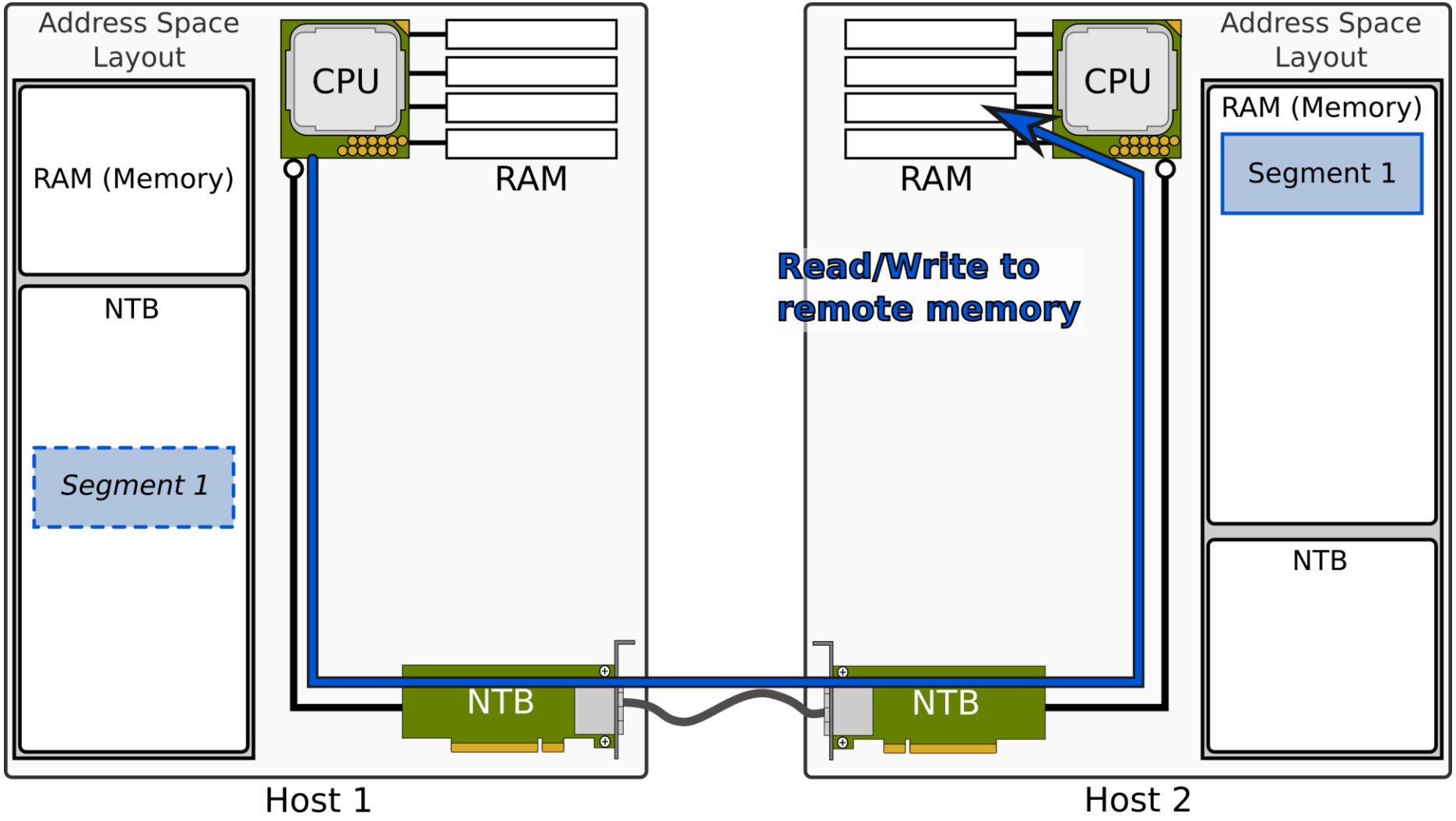
Enough data to generate 2 TLPs





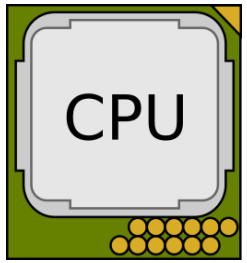
"FLAG_DONE" may arrive before data

Demo: PIO bandwidth with scibench2

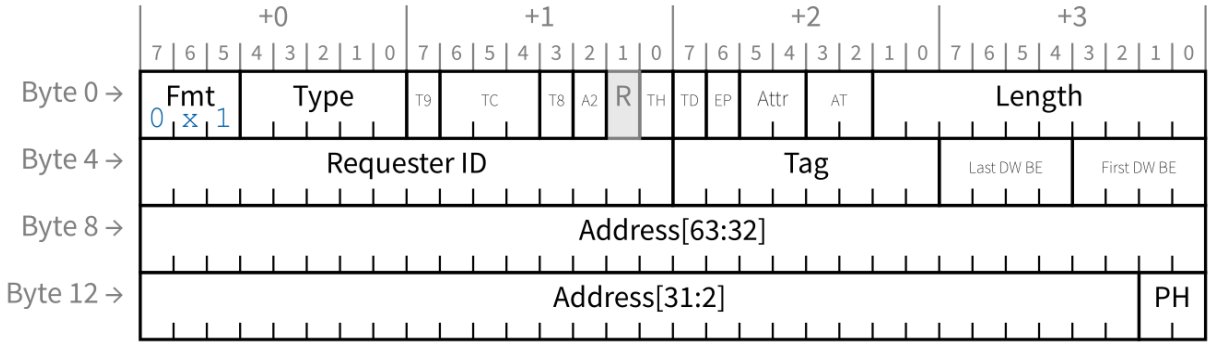


While memory writes are posted, reading from remote memory is slow as the distance between requester and completer increases

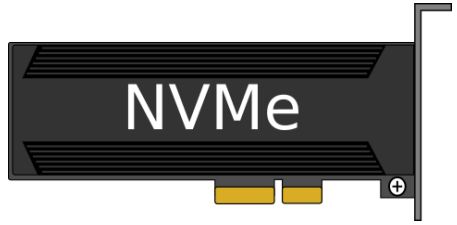
Requester



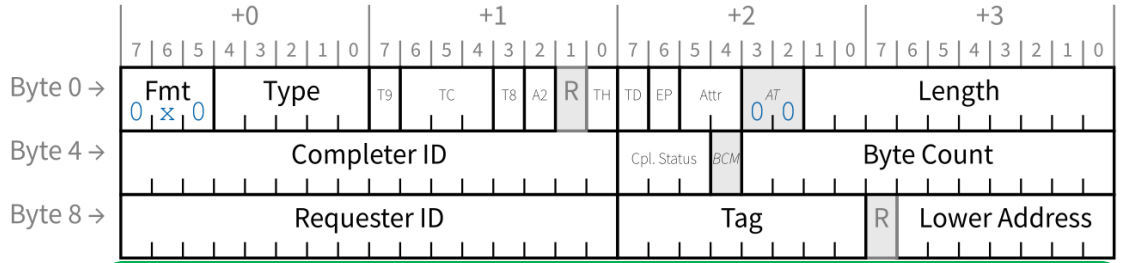
Memory Read request



Completer

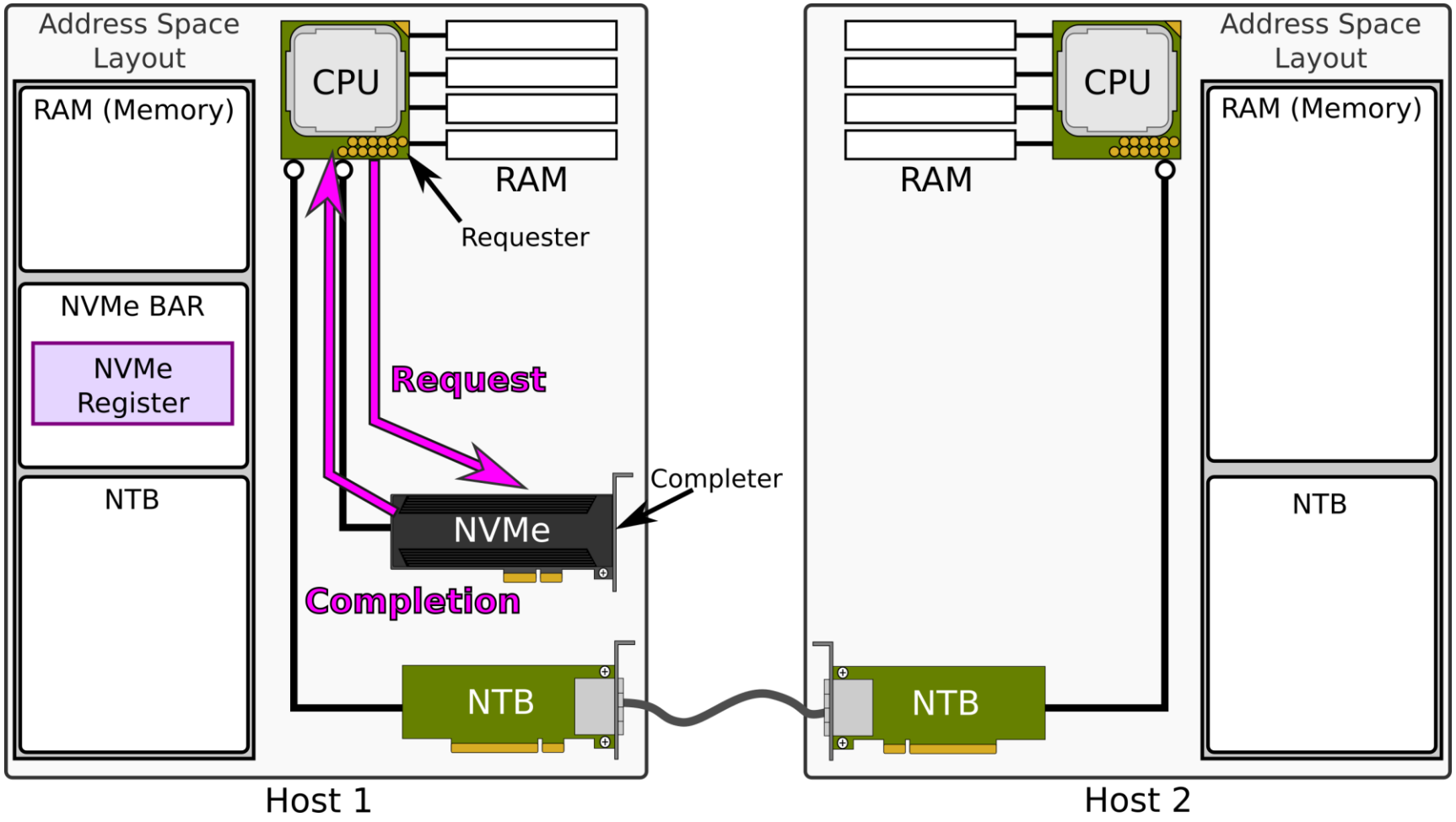


Completion with payload



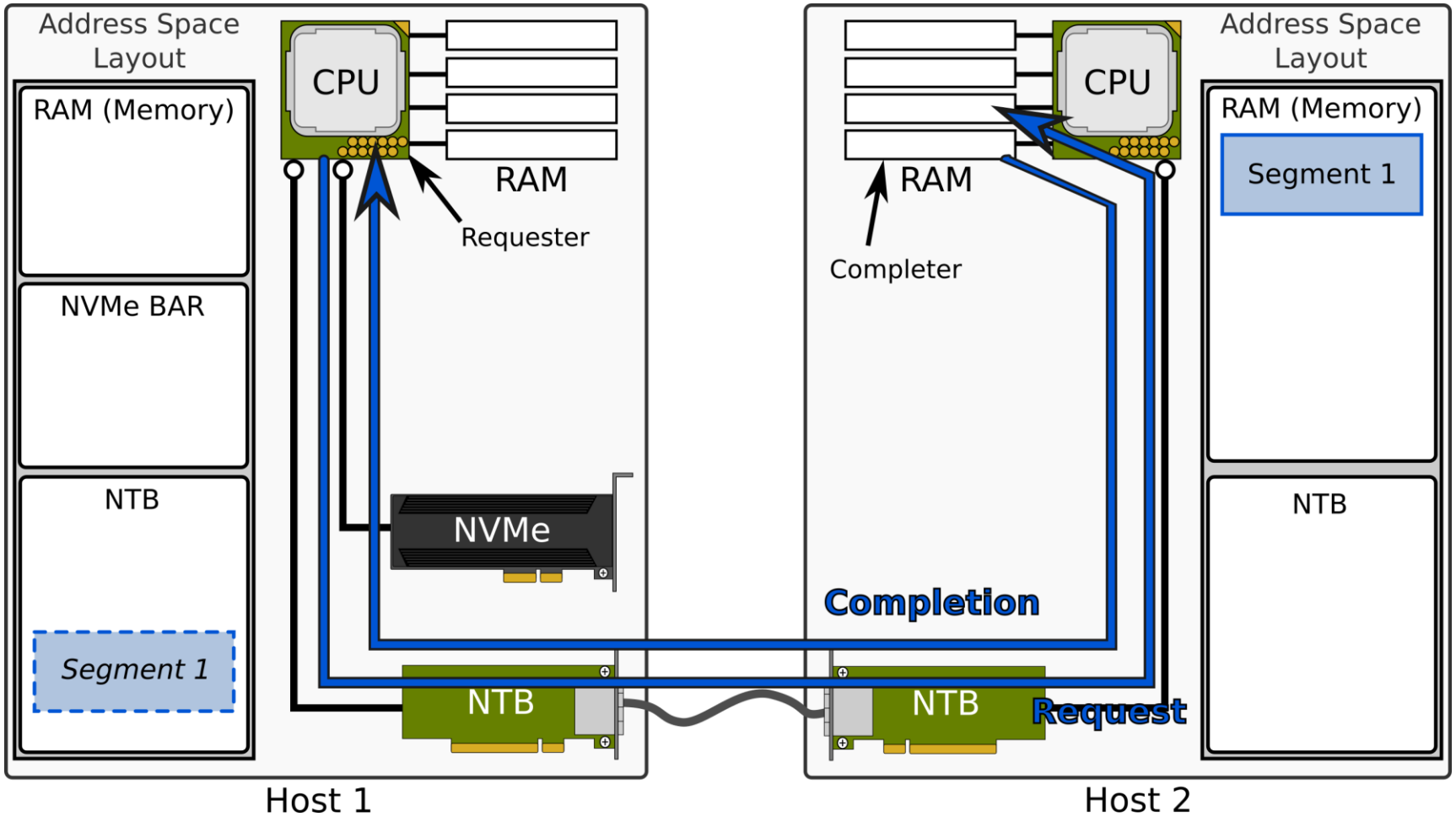
0x01020304

While memory writes are posted, reading from remote memory is slow as the distance between requester and completer increases



Reading from a local device's register

While memory writes are posted, reading from remote memory is slow as the distance between requester and completer increases



Reading from remote memory

PIO uses 100% CPU when transferring and polling which is good for small messages that need low latency, but bad for transferring larger sizes

```
size_t size = 1000000000;
for (i=0; i < size; i++) {
  remote_segment[i] = local_segment[i];
}

do_stuff();
```

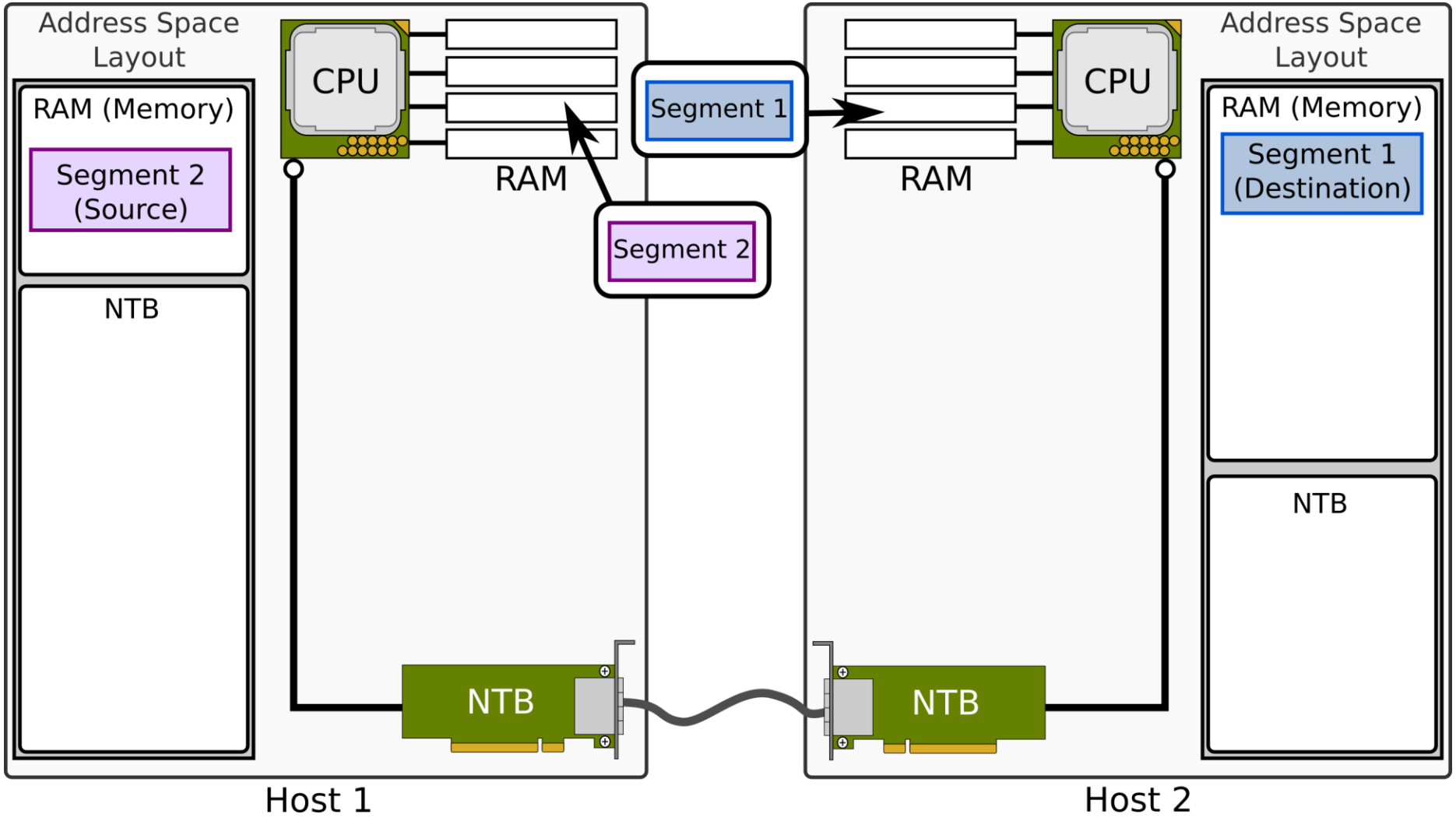
do_stuff() is delayed until CPU is done copying

Solution = Offload transfer to hardware by using DMA

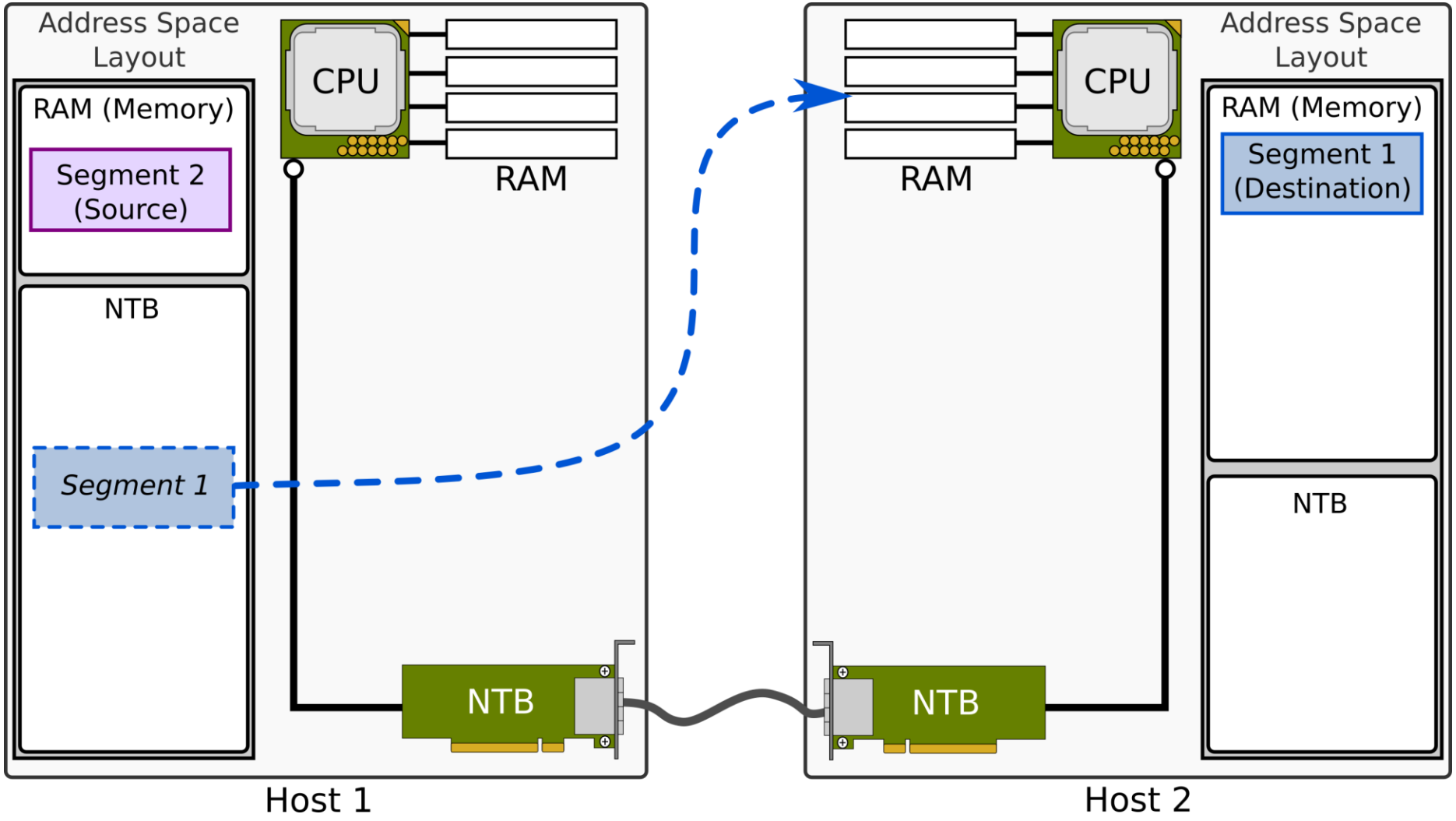
do_stuff() is can run in parallel while data is transferred

```
size_t size = 1000000000;
SCIStartDmaTransfer(local_segment,
                    remote_segment,
                    size);
do_stuff();
```

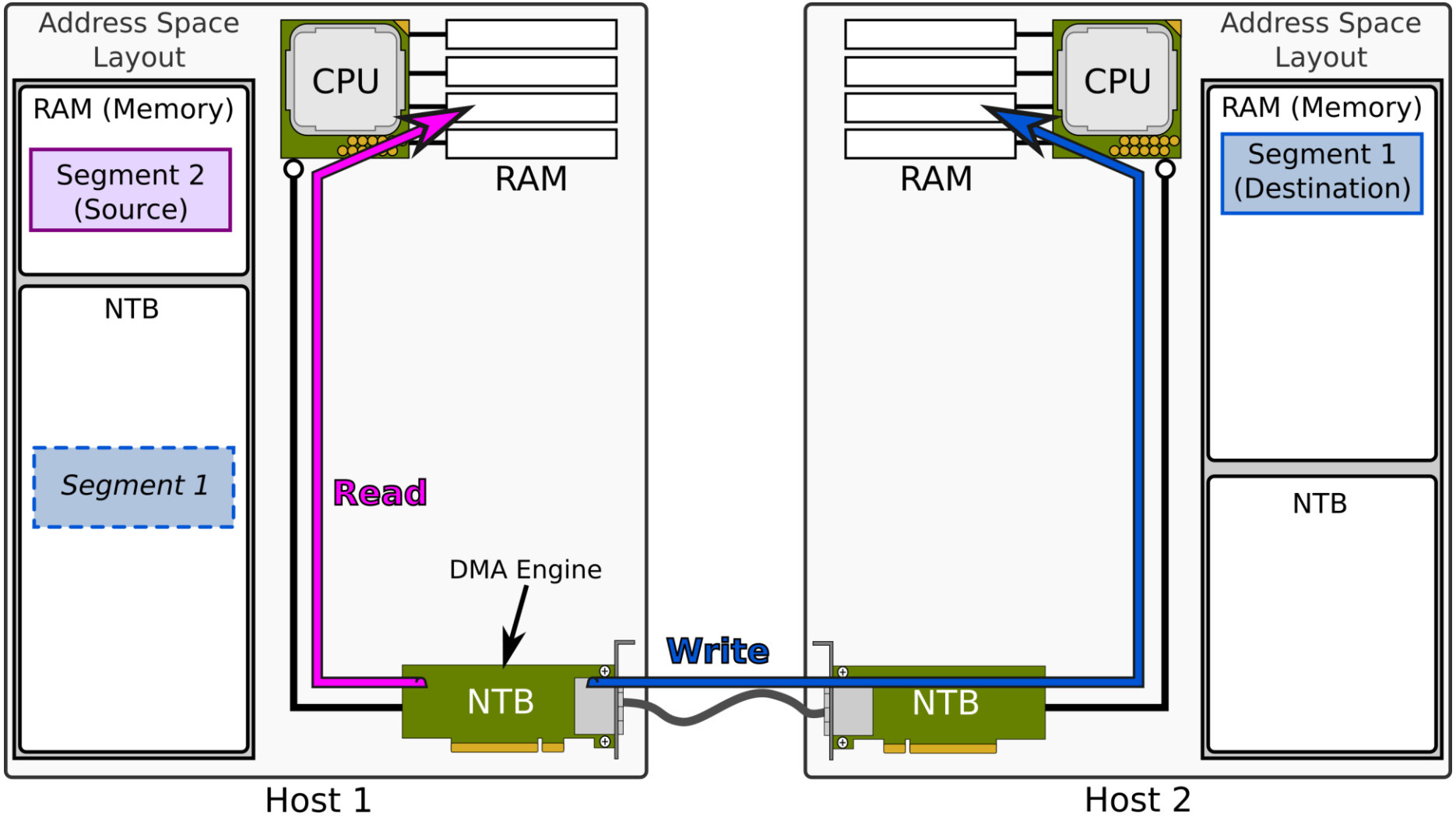
DMA: SCICreateSegment + SCIPrepareSegment + SCISetSegmentAvailable



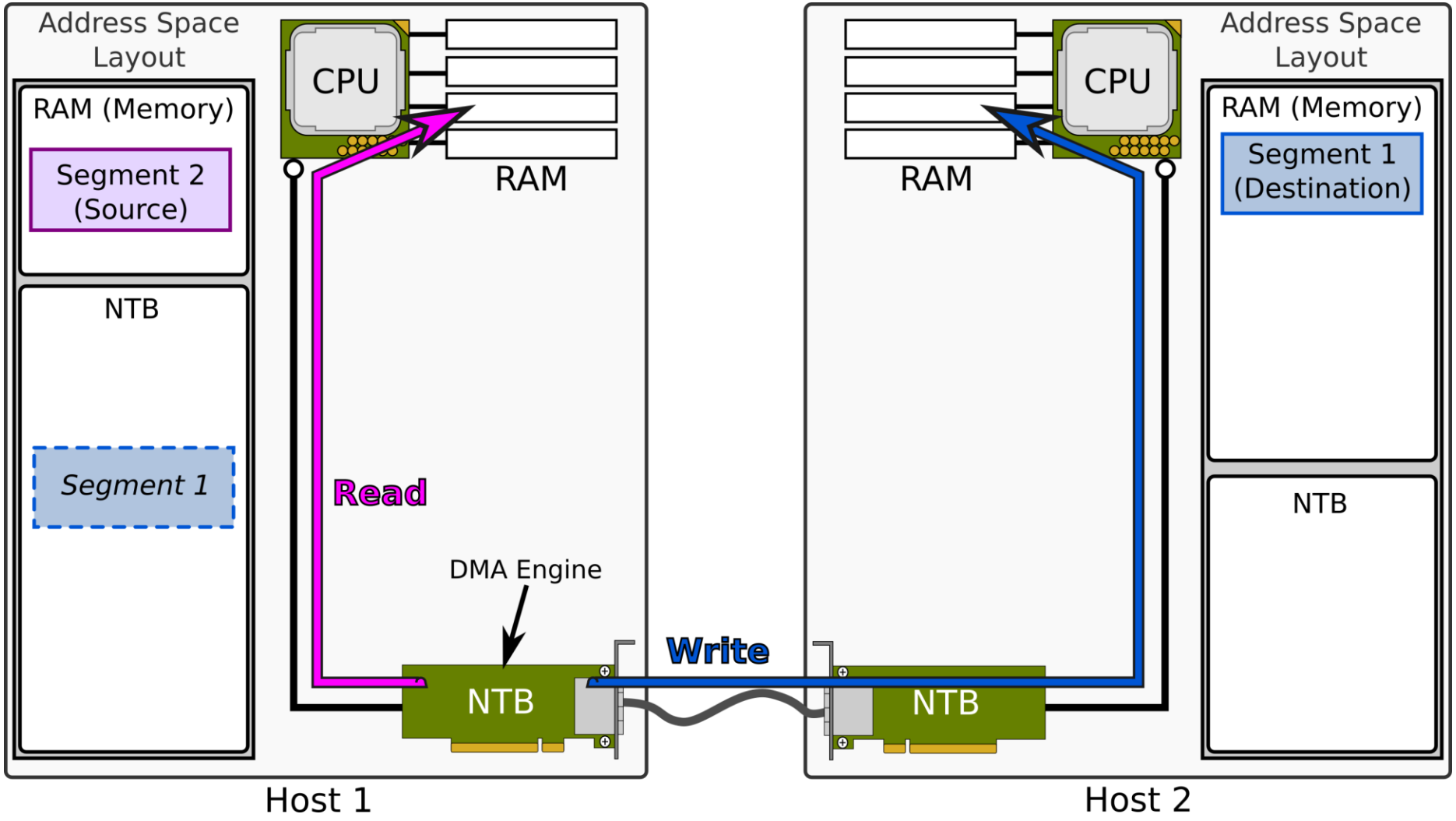
DMA: SCIConnectSegment + SCIMapRemoteSegment



DMA: SCIStartDmaTransfer

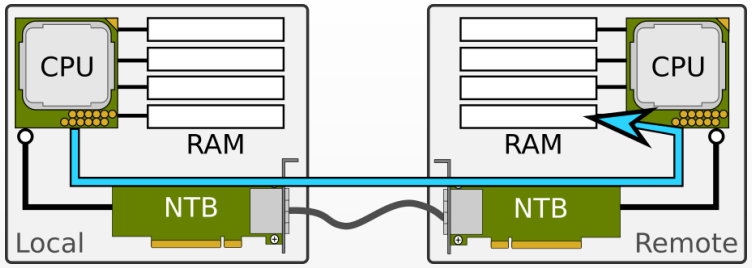


Demo: dma_bench



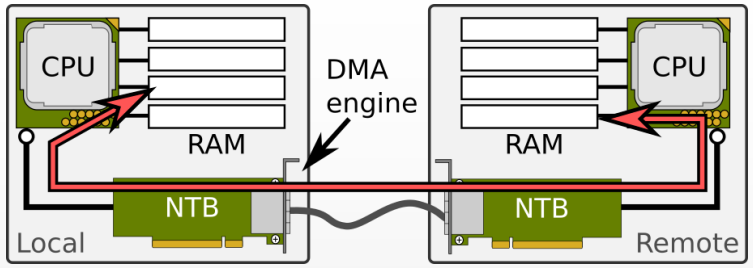
Which data transfer mechanism is "best" depends entirely on how you design your protocol

PIO



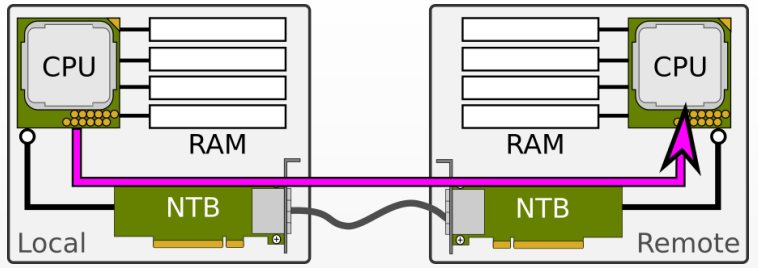
- Segments and PIO are good for control traffic, but stalls CPU when transferring a lot of data
- Reading from remote memory is significantly slower than writing to it
- Polling for changes in a local segment gives the lowest possible latency

Adapter DMA



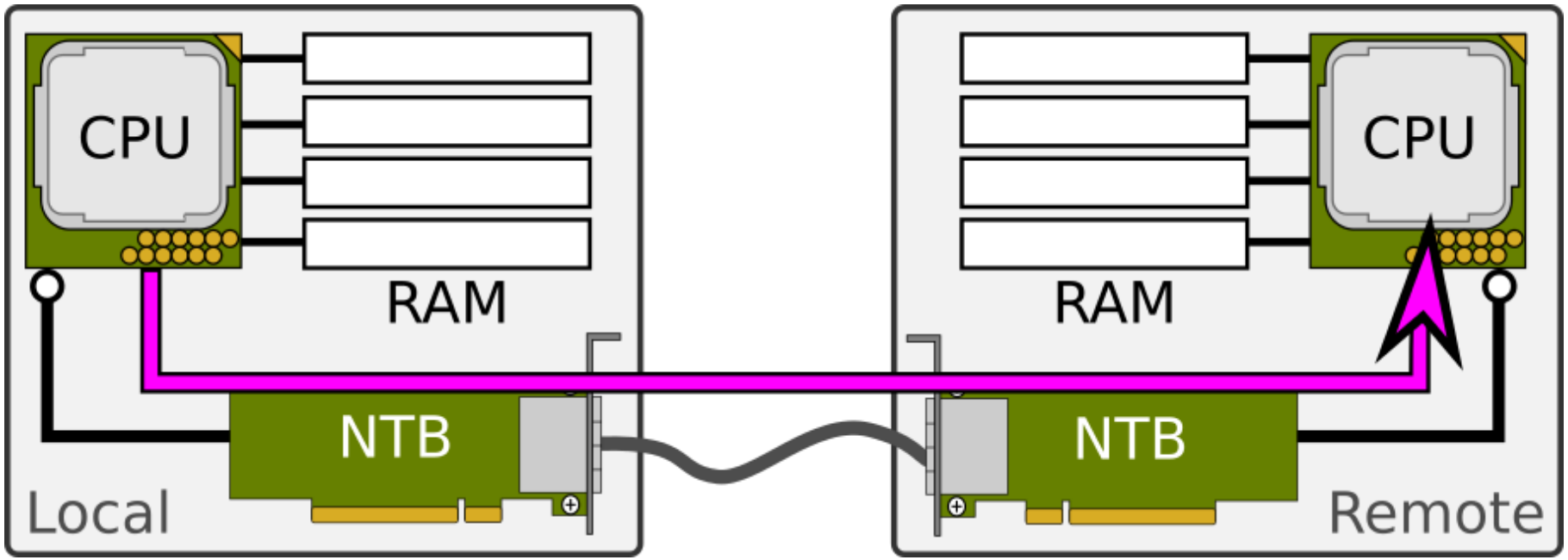
- DMA is good for bulk data transfers, but has an overhead for initiating transfer
- Many different functions for starting and waiting for DMA transfers (Tip: DMA vectors are transferred in-order, use last vector element for control data)
- Works best when you can pipeline computing and transfers

Remote Interrupts



- Remote interrupts are a decent method of triggering a callback function on a remote node, which can be used for signaling
- Remember to compile your SISCI program using _REENTRANT macro

Demo: intr_bench



SISCI API Documentation

SCIStartDmaTransfer()

```
SISCI_API_EXPORT void SCIStartDmaTransfer (
    sci_dma_queue_t       dq,
    sci_local_segment_t   localSegment,
    sci_remote_segment_t  remoteSegment,
    size_t                localOffset,
    size_t                size,
    size_t                remoteOffset,
    sci_cb_dma_t          callback,
    void *                callbackArg,
    unsigned int          flags,
    sci_error_t *         error
)
```

SCITriggerInterrupt()

```
SISCI_API_EXPORT void SCITriggerInterrupt (
    sci_remote_interrupt_t  interrupt,
    unsigned int            flags,
    sci_error_t *           error
)
```

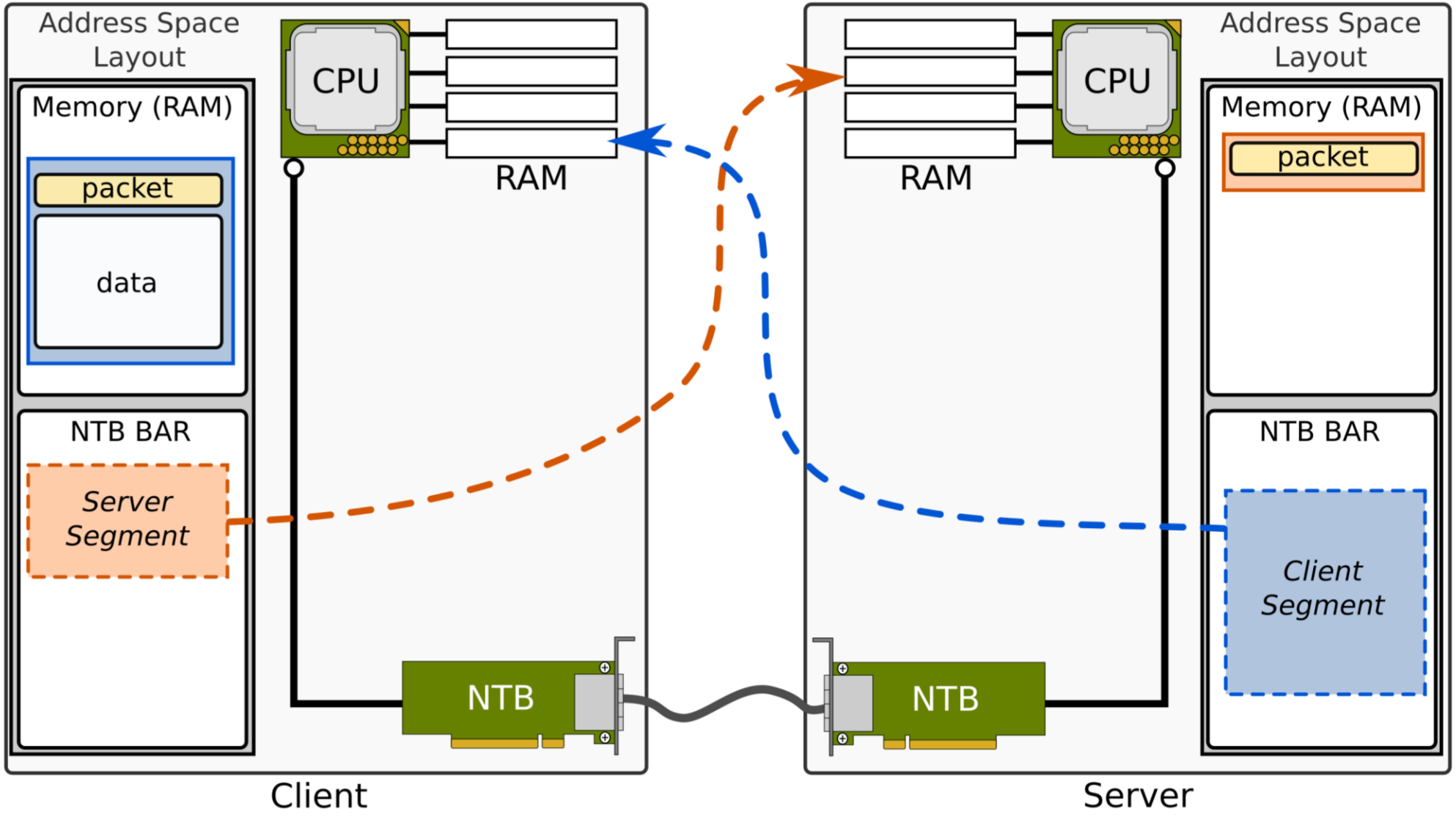
SCIMapRemoteSegment()

```
SISCI_API_EXPORT volatile void* SCIMapRemoteSegment (
    sci_remote_segment_t  segment,
    sci_map_t *           map,
    size_t                offset,
    size_t                size,
    void *                addr,
    unsigned int          flags,
    sci_error_t *         error
)
```

Dolphin

docs.dolphinics.com

Live Programming




Transp-02
Mex5-1
LarsCH
MXSW-03
PXMXS-40
MXMXS-20
MXMXS-10
PXMXS-31
PXMXS-45

The IN5050 Cluster



- 12 Nodes
- PCIe 3.0 x8 link
- Blocks of 4 nodes
  - 2 Intel i3 dual-core
  - 2 Intel i5 quad-core
- ./run.sh starts your program on 4 nodes
  - 1 server, 3 workers
- 512MB of segment memory
- 256MB of "prefetch" mapping space
  - Some space is used for communication with the other nodes
- 4 multicast groups, 2MB each