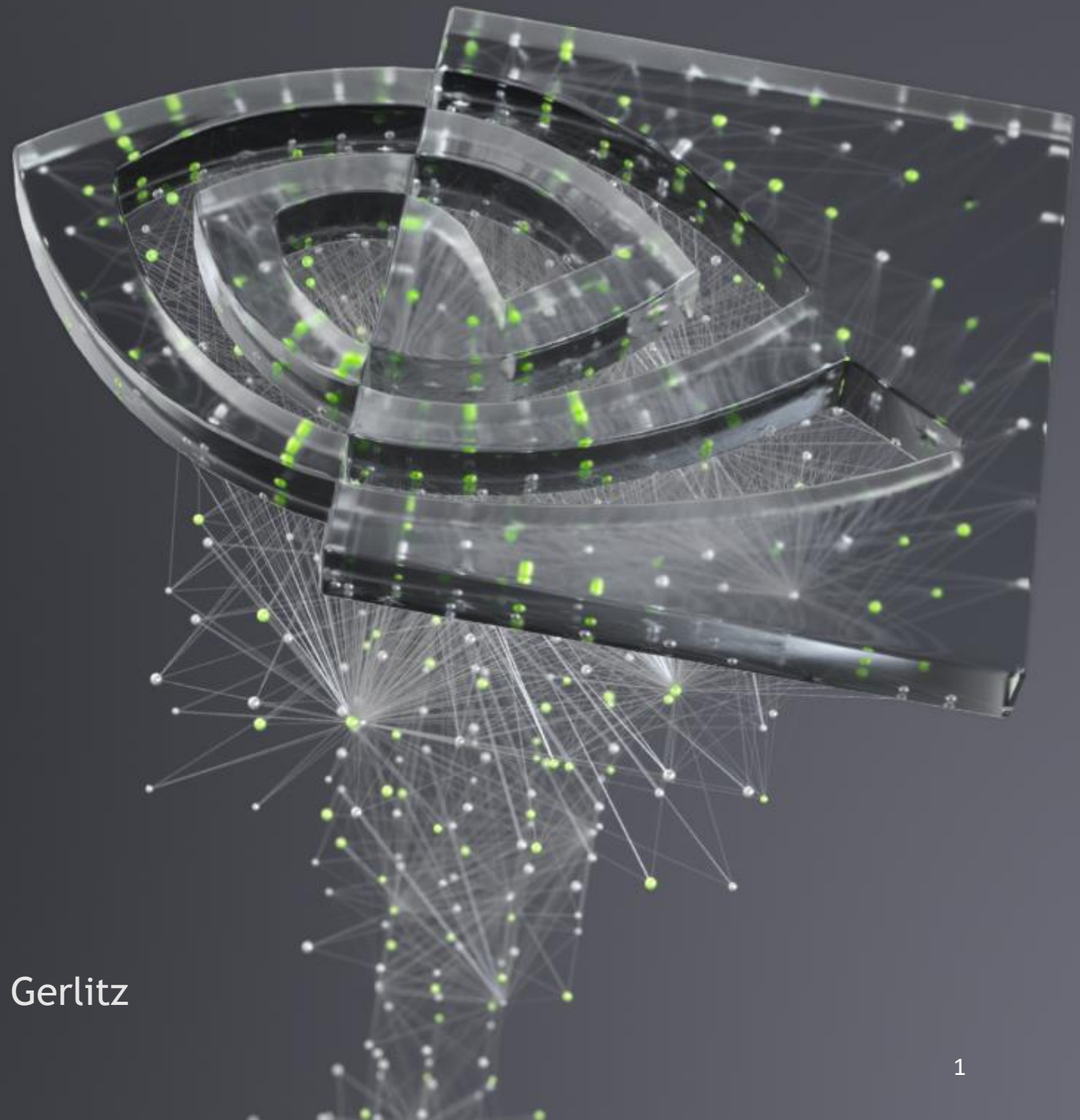


AUTONOMOUS NVME-TCP OFFLOAD

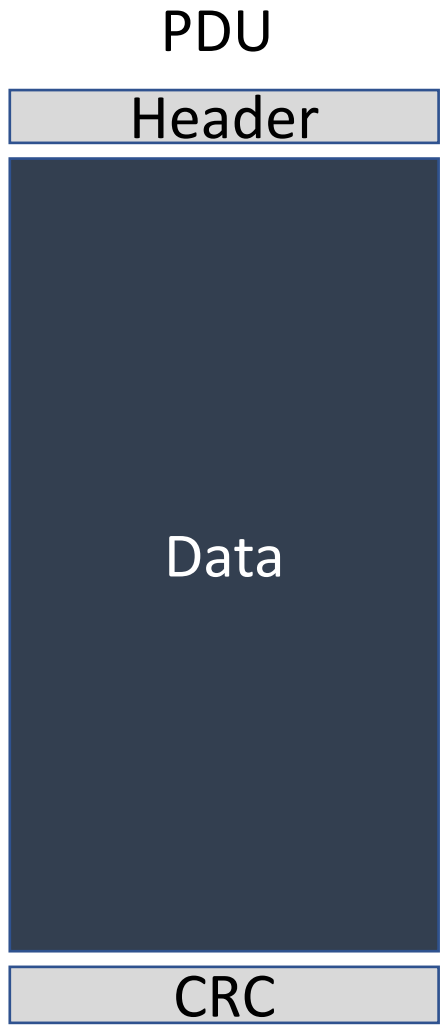
Boris Pismenny, Yoray Zack, Ben Ben-Ishay and Or Gerlitz

Overview

- Motivation
- Storage protocol offload
- Seamless integration
- APIs and implementation

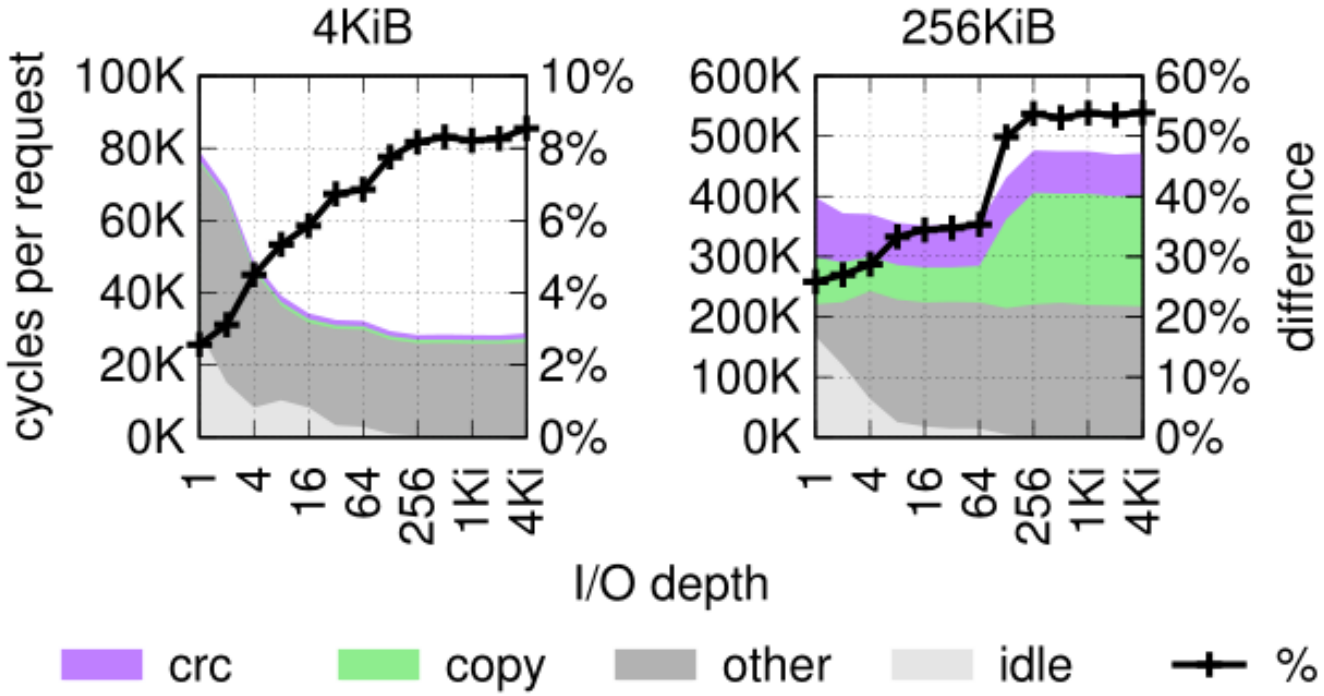
Motivation: offload opportunities

- Transmit side data checksum calculation
  - PDU data CRC calculation
- Receive side data checksum validation
  - PDU data CRC verification
- Receive side copy
  - Need to place data at destination buffers
  - But TCP receives data in anonymous unaligned buffers
  - Data is copied from TCP to destination buffers



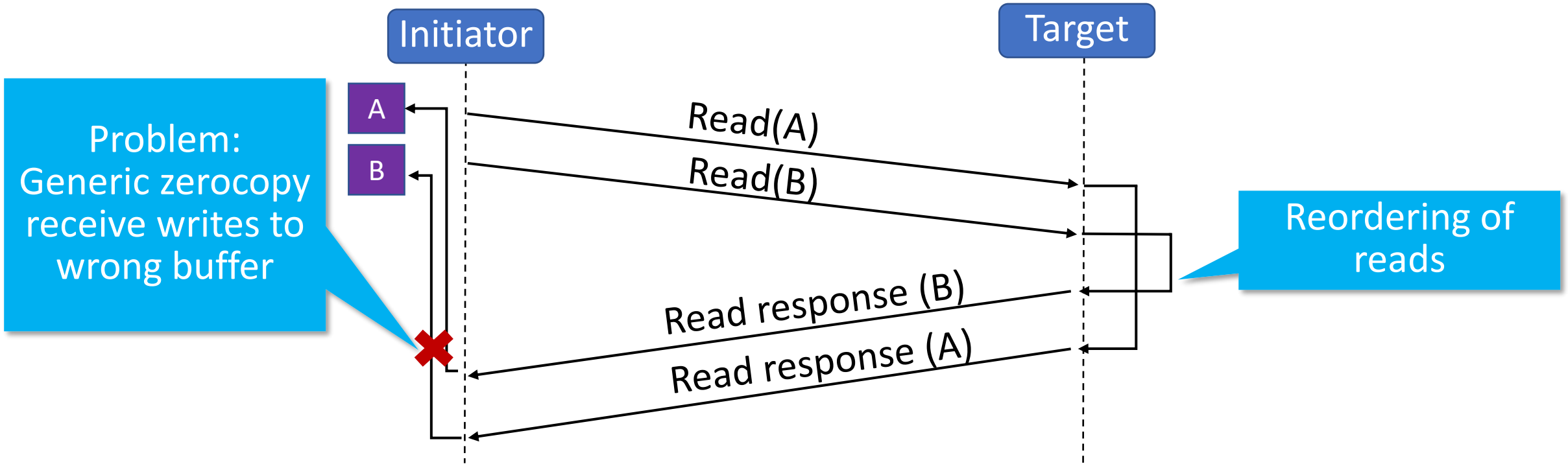
Motivation: offload opportunities

- Copy and CRC consume up to 50% per IO cycles



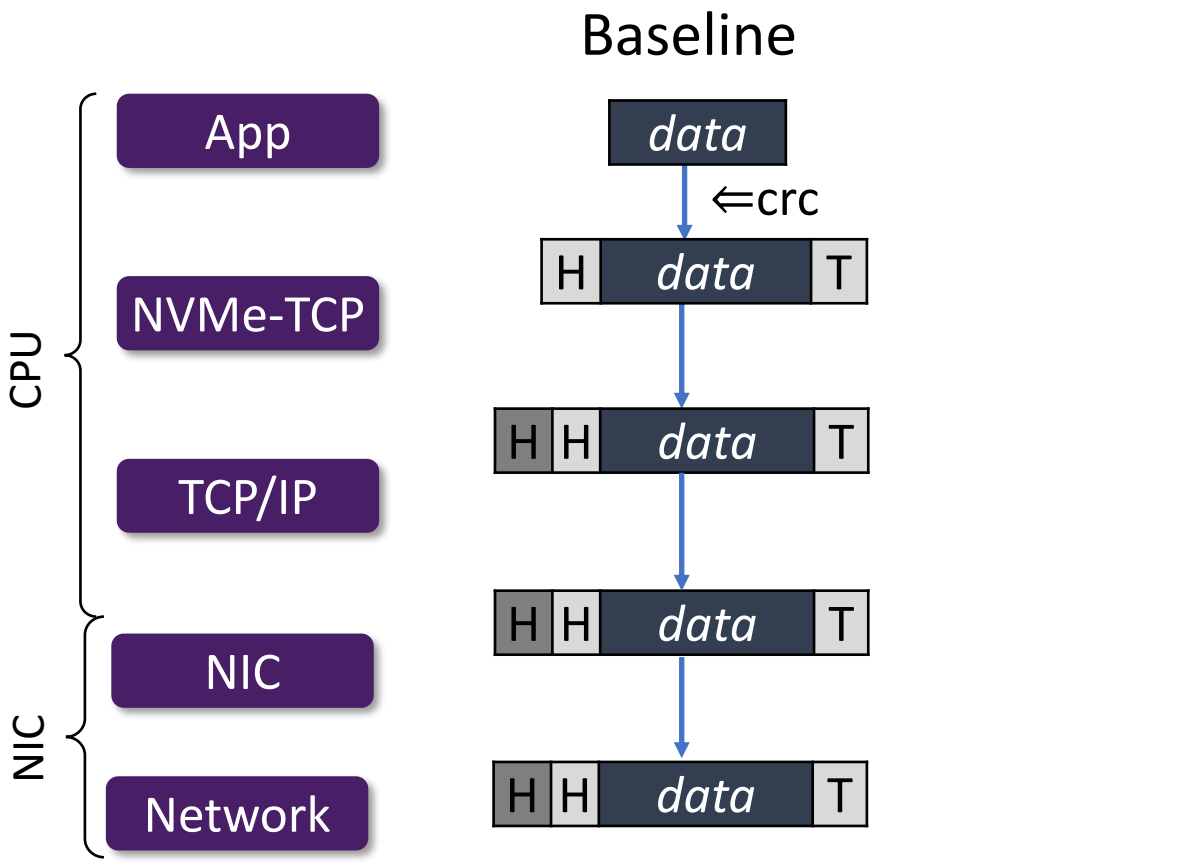
Motivation: NVMe out-of-order processing

- Generic zerocopy receive does not work
- NVMe supports reordering of storage read/write operations



To solve this problem, we need upper layer protocol awareness!

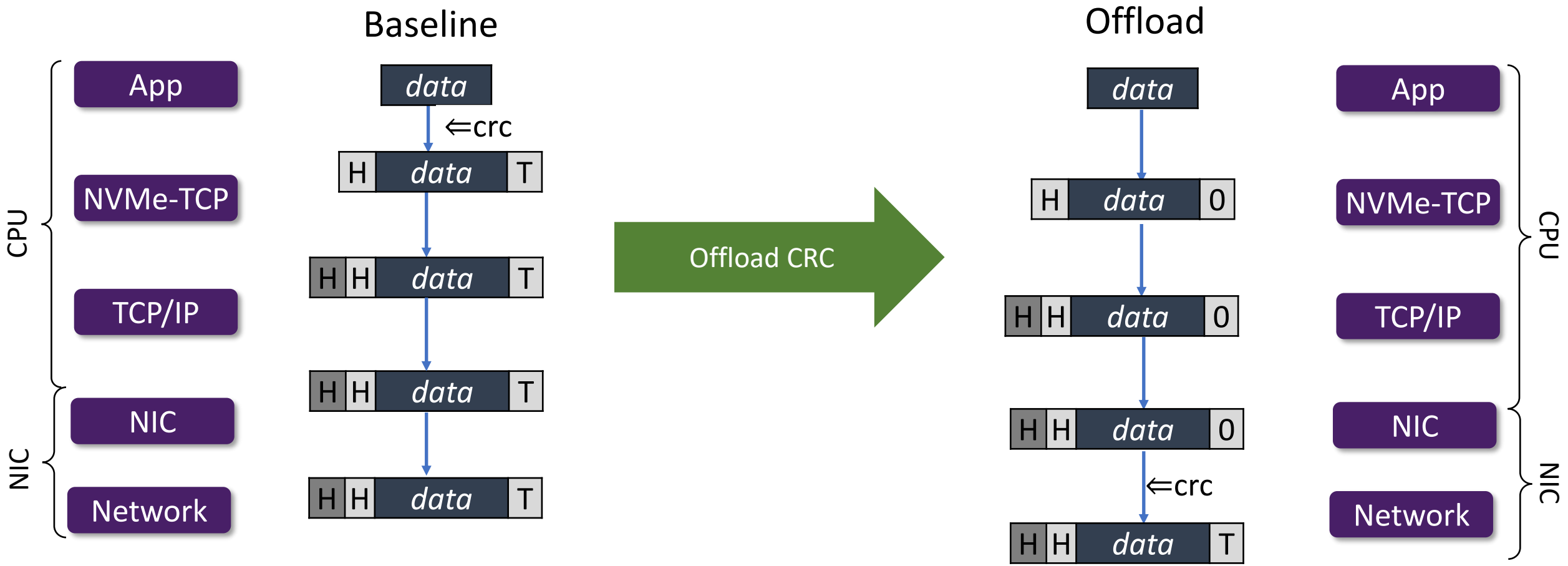
Transmit offload overview



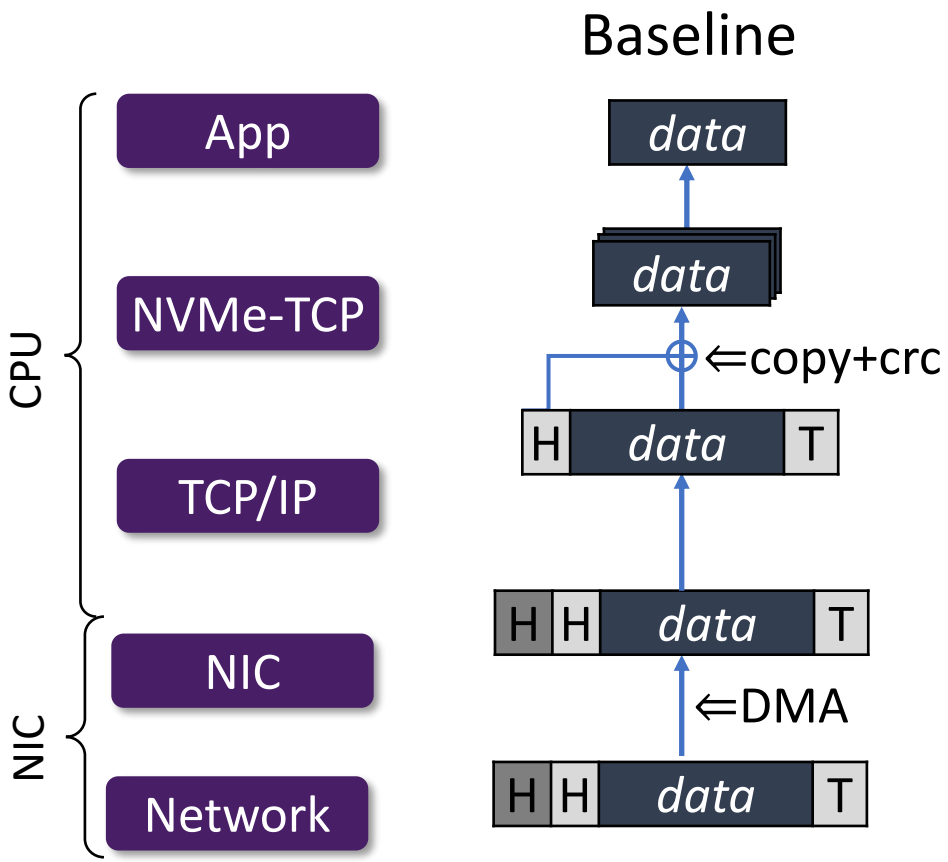
TCP header

NVMe-TCP PDU header/trailer

Transmit offload overview



Receive offload overview

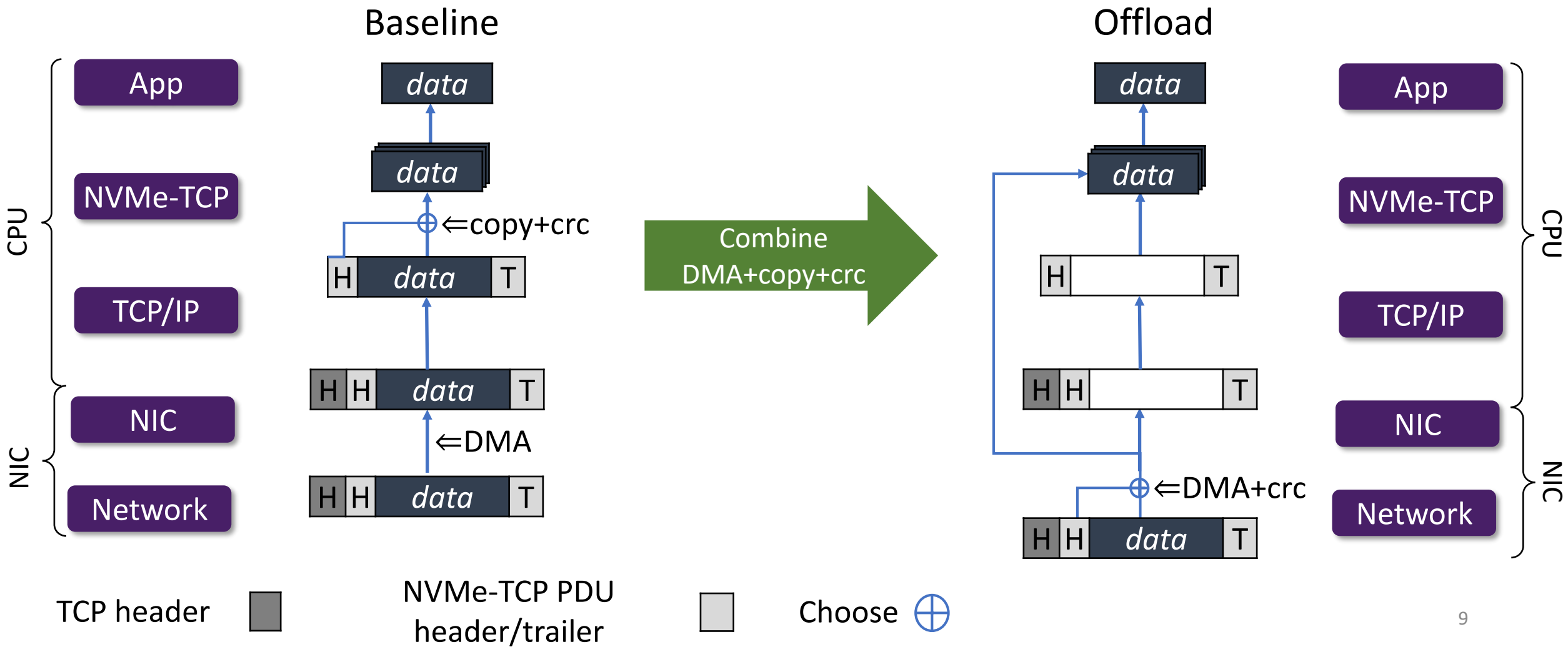


TCP header

NVMe-TCP PDU header/trailer

Choose ⊕

Receive offload overview

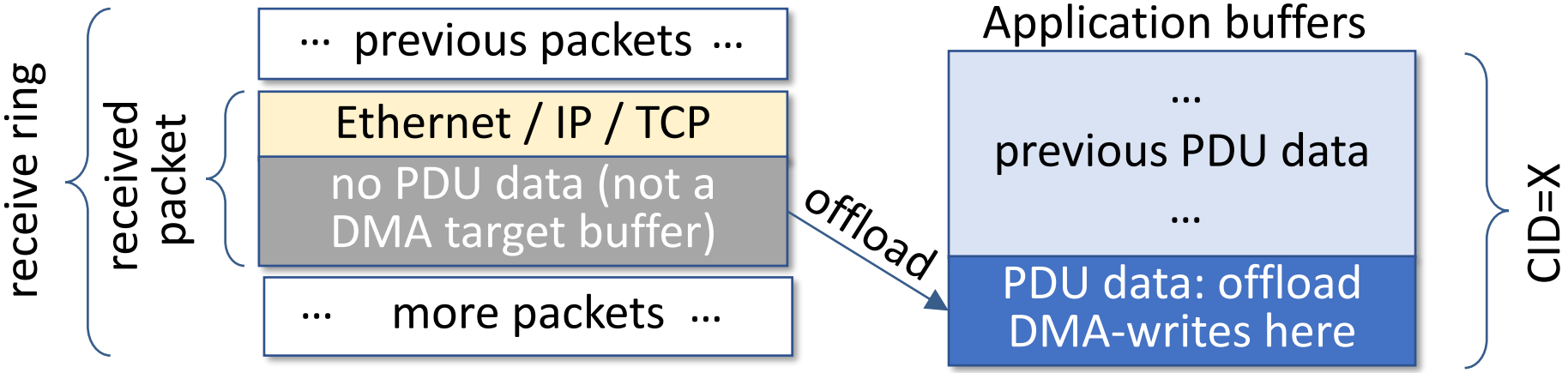


Seamless integration: crc

- New SKB bit skb->ddp_crc
  - Used similarly to TLS's skb->decrypted
- On transmit skb->ddp_crc indicates CRC offload is expected
- On receive skb->ddp_crc indicates no CRC errors in packet's payload
  - skb->ddp_crc==0 triggers software PDU CRC calculation

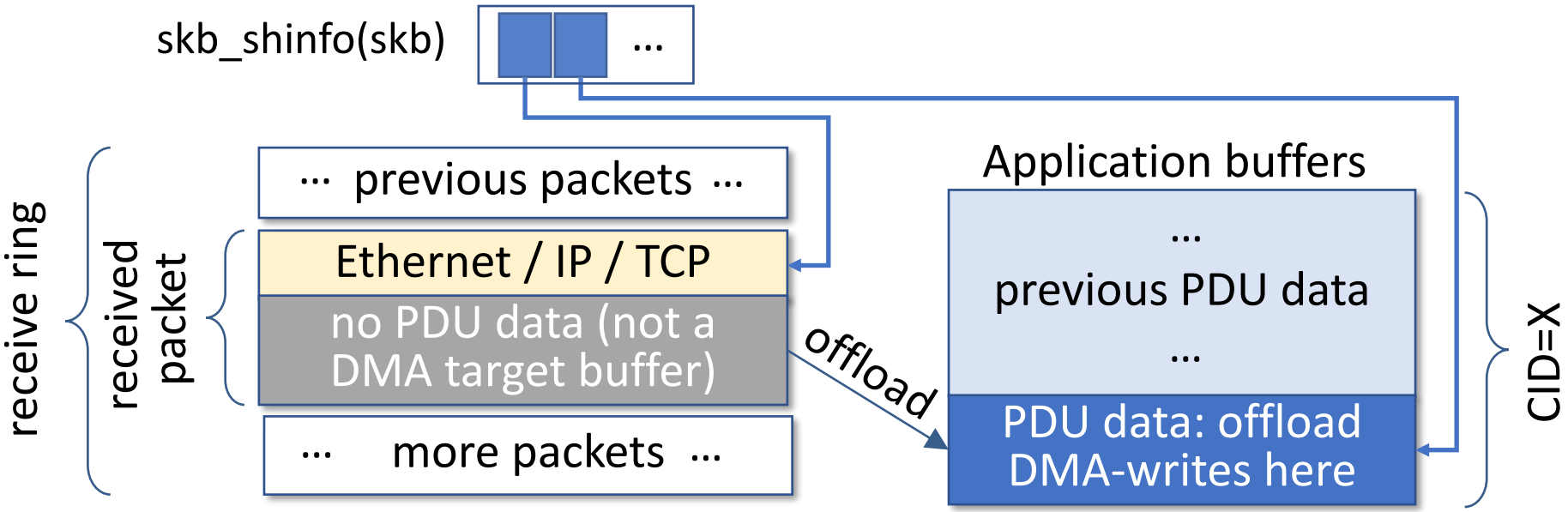
Seamless integration: copy

- NIC driver builds SKBs of packets on the wire
  - Packet headers from receive ring
  - Storage protocol headers/trailers from receive ring
  - Payload from destination buffers



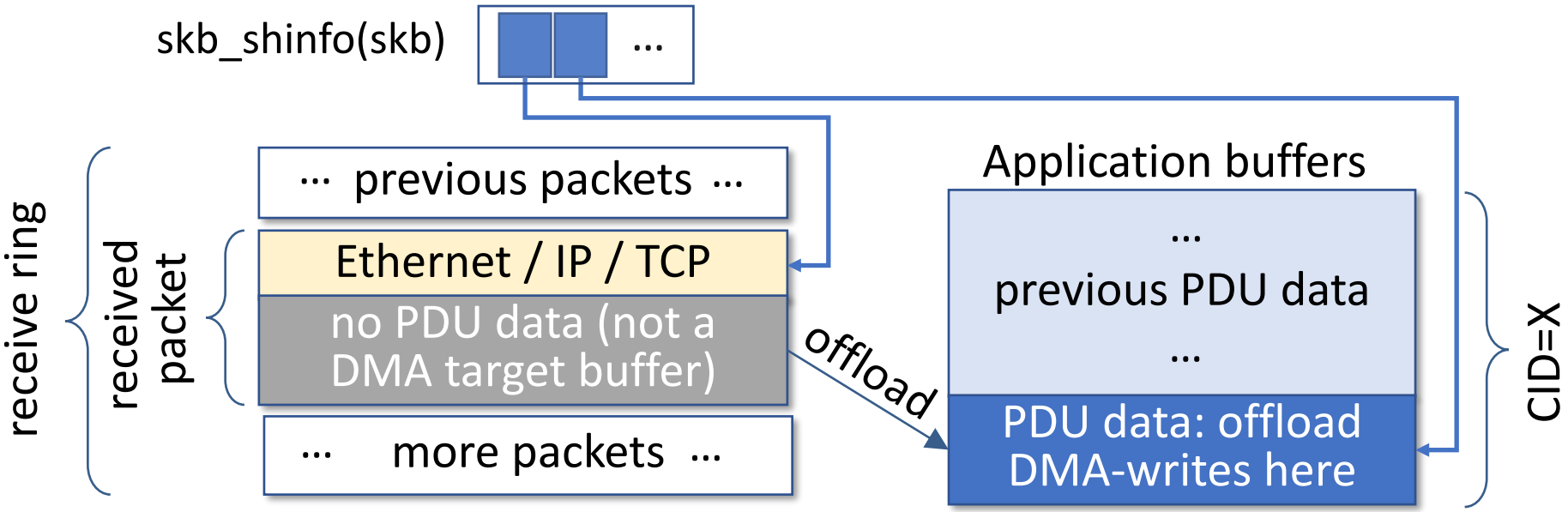
Seamless integration: copy

- NIC driver builds SKBs of packets on the wire
  - Packet headers from receive ring
  - Storage protocol headers/trailers from receive ring
  - Payload from destination buffers



Seamless integration: copy

- NIC driver builds SKBs of packets on the wire
  - Packet headers from receive ring
  - Storage protocol headers/trailers from receive ring
  - Payload from destination buffers
- Storage protocol skips copy
  - Iff (src == dst) before memcpy



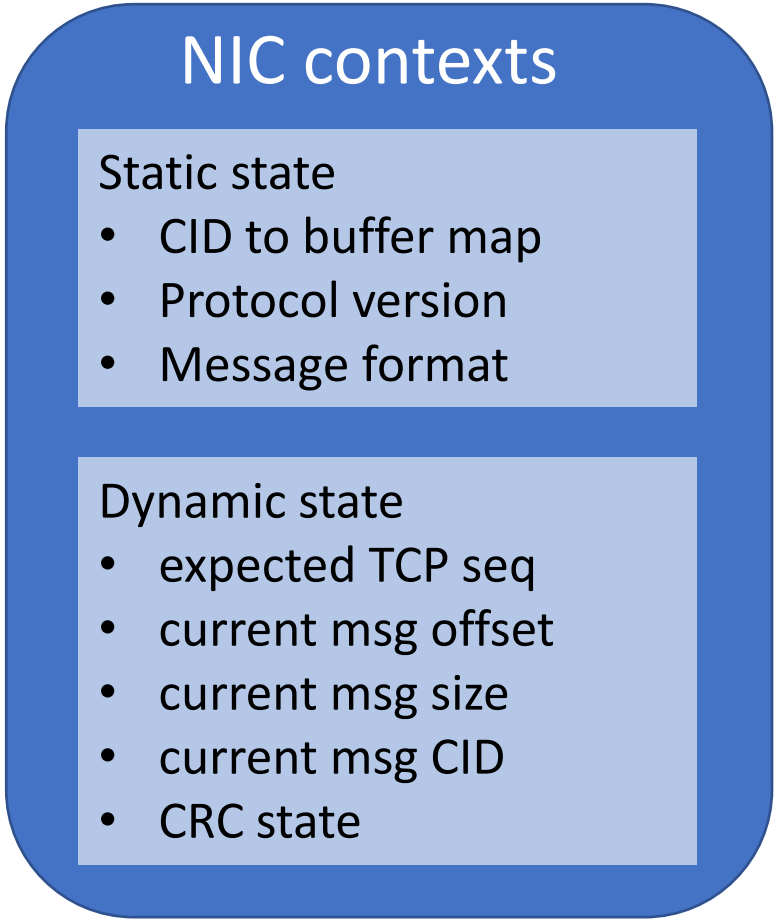
Seamless integration: copy

- Need to avoid network stack copies of data
  - Problem: skb_coalesce copies data from destination buffer back to SKB
  - Solution: Avoid it by reusing the skb->ddp_crc bit

- Need to map between destination pages and their identifiers
  - Upper layer protocol maintains mapping

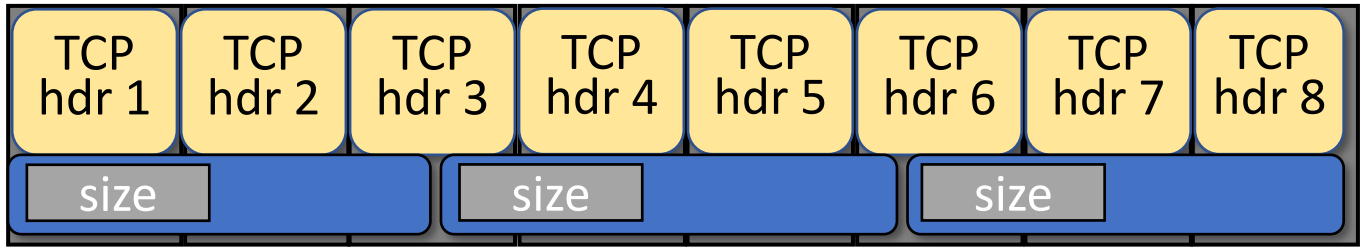
Hardware perspective

NIC contexts



Transmit offload in-sequence

- NIC offload Implementation is simple
  - Incrementally offload using NIC contexts



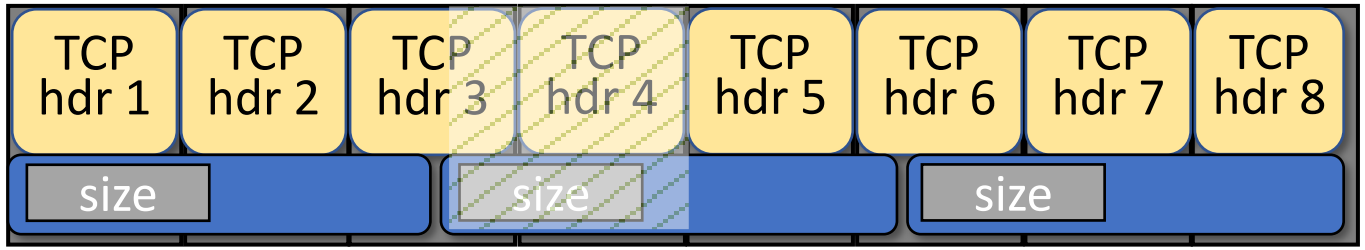
NIC contexts

Static state
- CID to buffer map
- Protocol version
- Message format

Dynamic state
- expected TCP seq
- current msg offset
- current msg size
- current msg CID
- CRC state

Transmit offload out-of-sequence

- Wrong dynamic NIC context state
- Context recovery needs only the message prefix
  - Driver can get the prefix from the storage protocol layer



- Reuse TCP transmit buffer for storing data
  - TCP ACKs release data in storage protocol PDU granularity

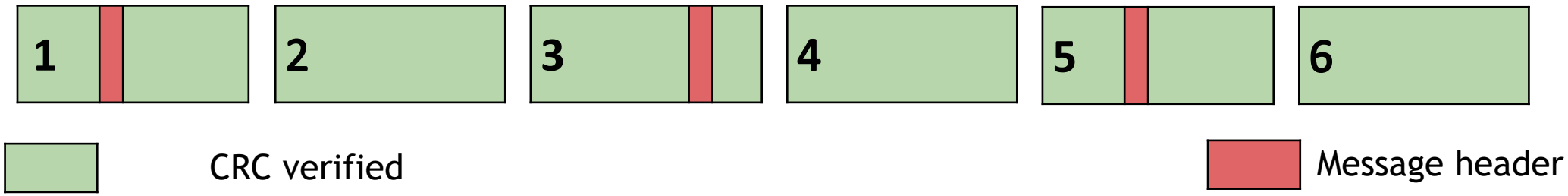
NIC contexts

Static state
- CID to buffer map
- Protocol version
- Message format

Dynamic state
- expected TCP seq
- current msg offset
- current msg size
- current msg CID
- CRC state

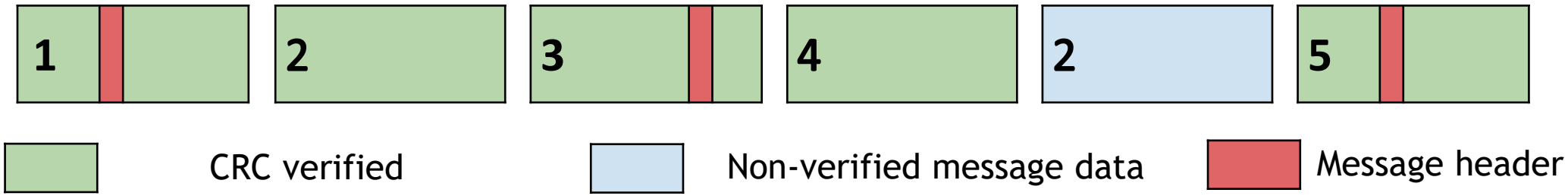
Receive offload in-sequence

- NIC offload Implementation is simple
  - Incrementally offload using NIC contexts
- Hardware reports one bit per packet
  - is packet CRC ok?



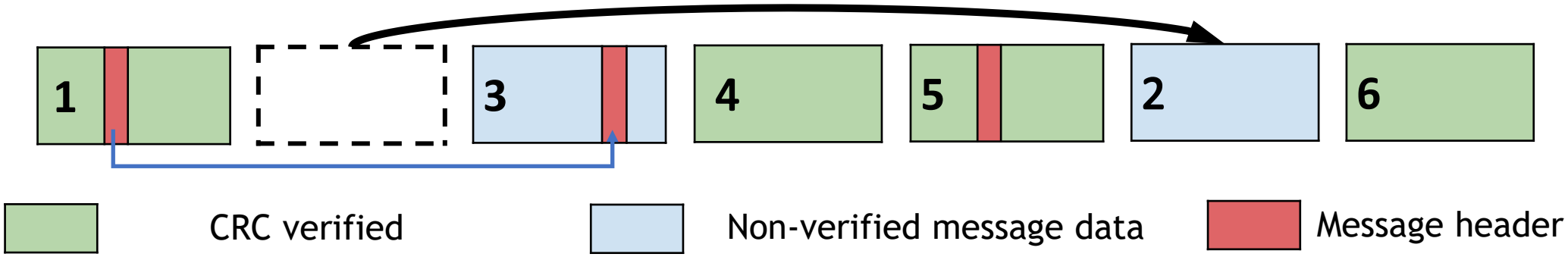
Receive offload retransmission

- Retransmissions bypass offload
  - Software fallback



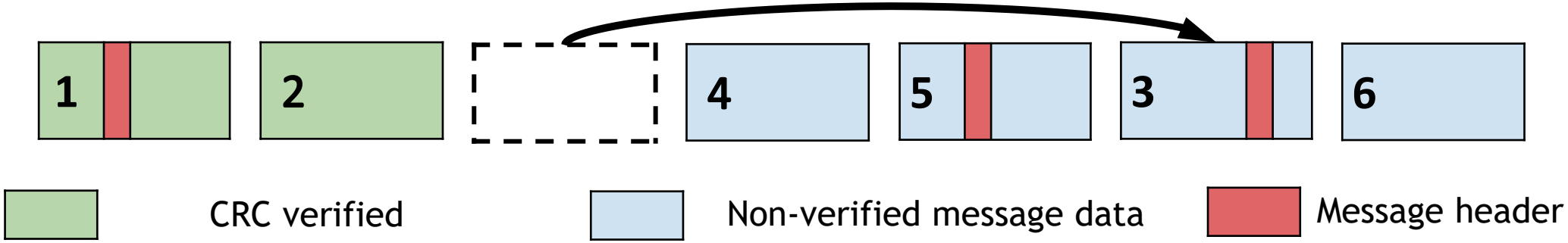
Receive offload data reordering

- PDU data reordering
  - Skip hardware to skip to the next record
  - Continue offloading



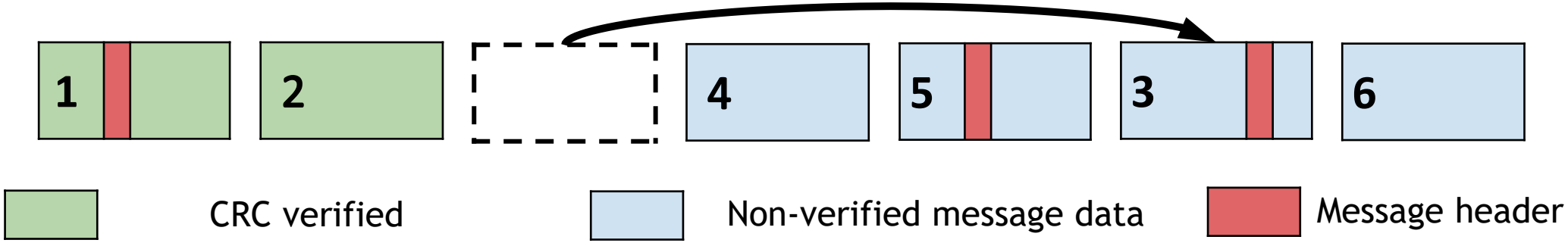
Receive offload header reordering

- PDU header reordering
  - Stops hardware NIC offloading
  - Software must recover NIC context to continue



Receive offload recovery problem

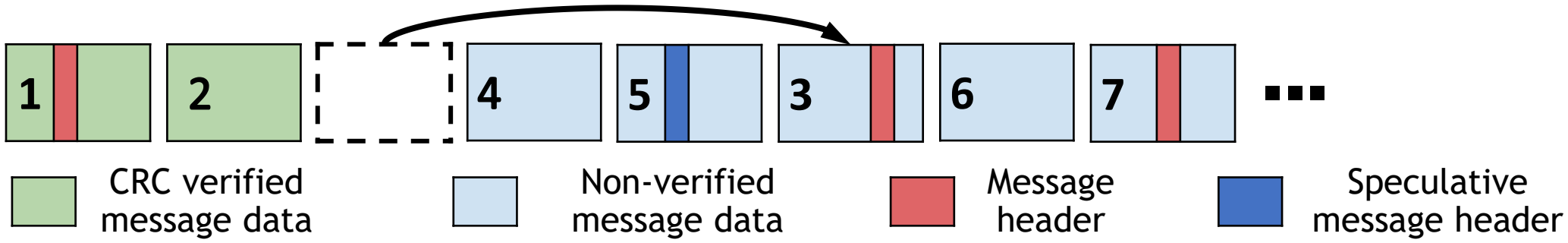
- NIC context recovery on receive is non-trivial:
  - Stopping packets to recover NIC context is impossible
    - Packets keep coming
  - Software alone cannot recover during traffic
    - Need to combine software and hardware



Receive offload recovery solution

NIC context recovery relies on:

(1) Speculatively finding PDU message header magic pattern

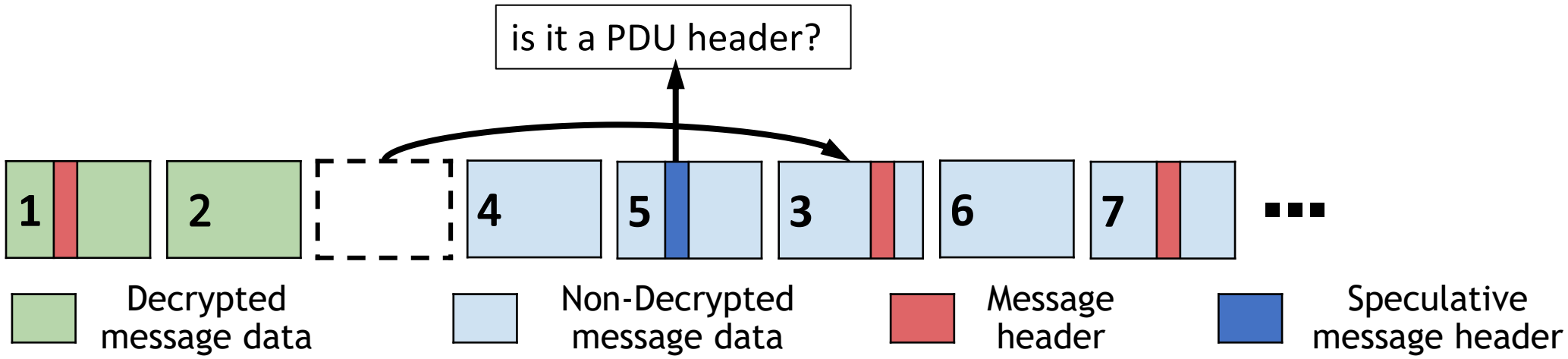


Receive offload recovery solution

NIC context recovery relies on:

(1) Speculatively finding PDU message header magic pattern

(2) Requesting software to confirm that this is indeed a PDU header, while



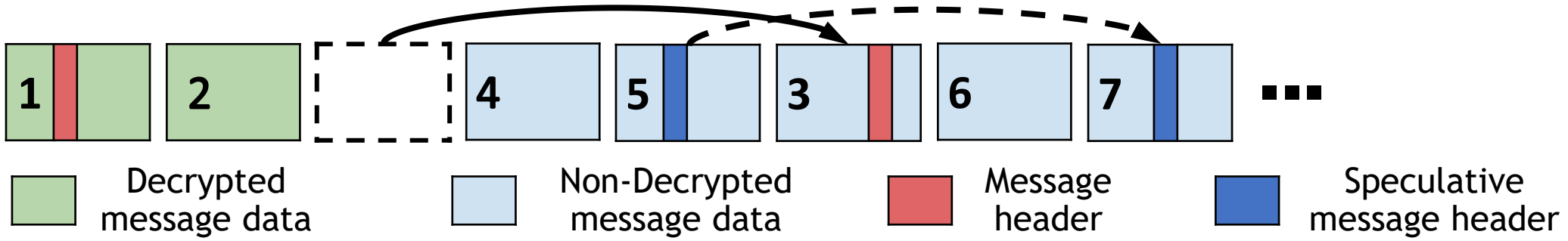
Receive offload recovery solution

NIC context recovery relies on:

(1) Speculatively finding PDU message header magic pattern

(2) Requesting software to confirm that this is indeed a PDU header, while

(3) Tracking subsequent messages using the message header's length field



Receive offload recovery solution

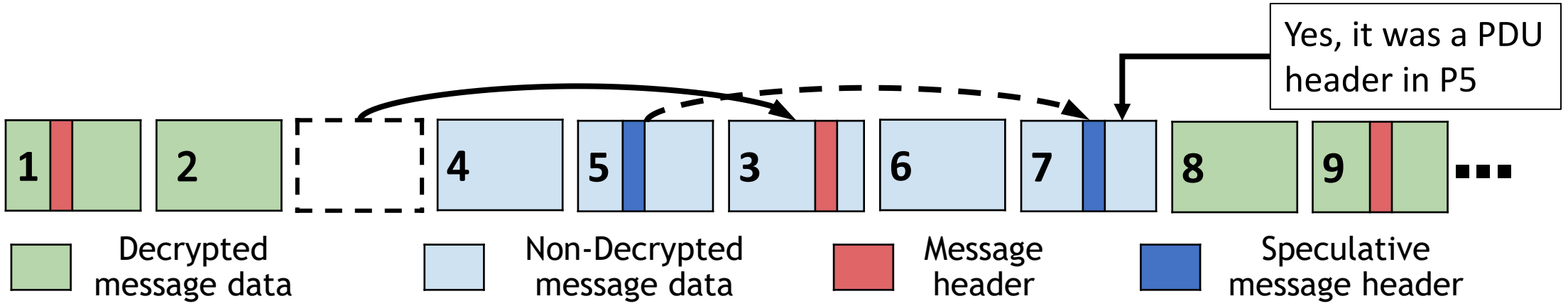
NIC context recovery relies on:

(1) Speculatively finding PDU message header magic pattern

(2) Requesting software to confirm that this is indeed a PDU header, while

(3) Tracking subsequent messages using the message header's length field

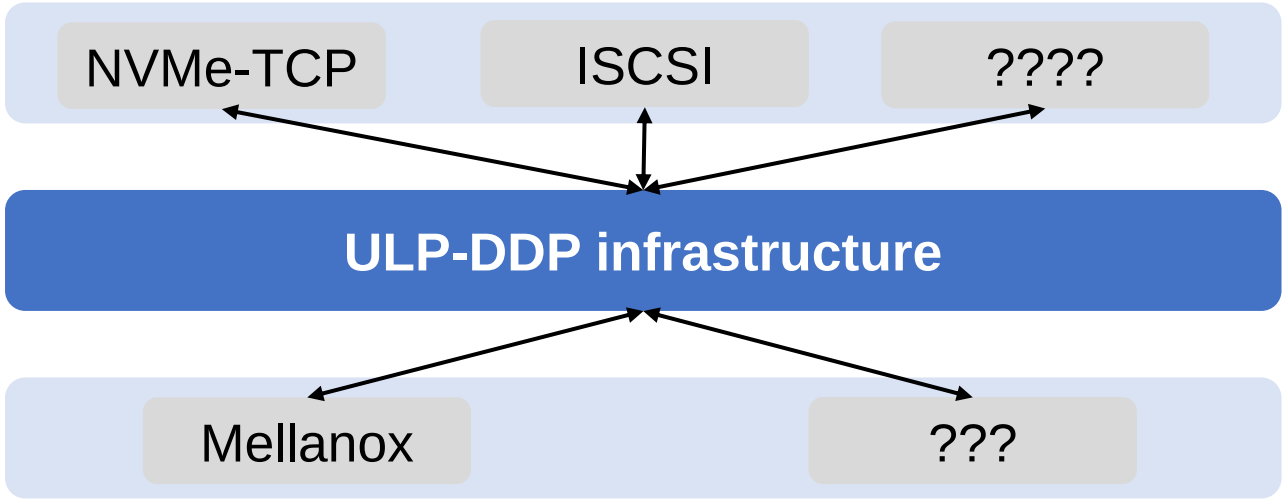
(4) Resuming offload if software confirms the HW speculation



APIs and implementation

ULP DDP infrastructure

- ULP DDP interposes between NIC drivers and storage protocols
- Protocol agnostic
- Vendor agnostic
- First users are NVMe-TCP and Mellanox drivers

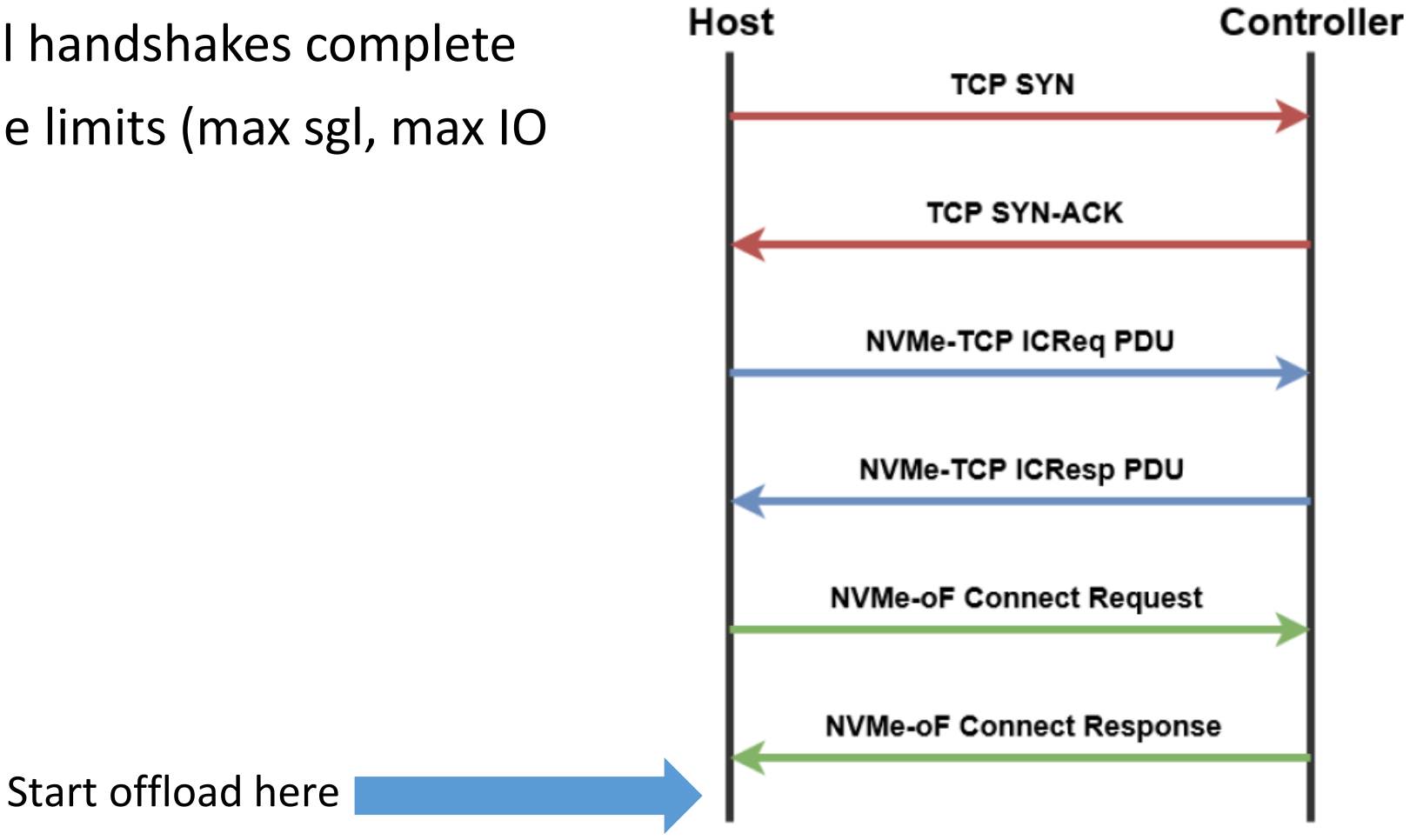


ULP DDP APIs

- Setup/teardown per-connection state
- Setup/teardown mapping between pages and their identifiers
- Protocol resynchronization

NVMe-TCP setup per-connection state

- Offload begins after all handshakes complete
- Configure NVME queue limits (max sgl, max IO size, etc.)



NVMe-TCP mapping pages

- Map buffers before IO send
- Unmap on IO completion
  - Added asynchronous unmap to improve performance

Netdev features

- We run out of netdev feature bits!
- Proposal: override __UNUSED_NETIF_F_1
  - Single bit for both receive and transmit

Future work

- Integration with TLS
  - Data-path POC working
  - Need a solution for the TLS handshake in NVMe-TCP