NetCrafter

Tailoring Network Traffic for Non-Uniform Bandwidth Multi-GPU Systems

Amel Fatima[1]

Yang Yang[1], Yifan Sun[2], Rachata Ausavarungnirun[3], and Adwait Jog[1]

1 University of Virginia

2 William & Mary
CHARTERED 1693

3 King Mongkut's University of Technology North Bangkok

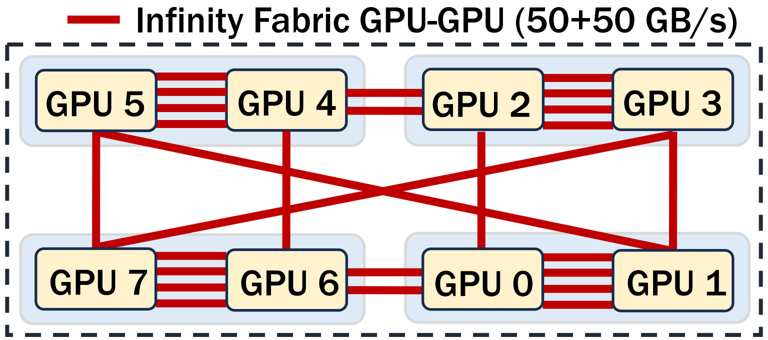
SUMMARY

Problem and Motivation

- Multi-GPU systems accelerate workloads by interconnecting **multiple GPUs**
- These interconnects often exhibit **non-uniform bandwidth** as systems scale
- **Slower links** become performance bottlenecks, limiting scalability

Key Ideas

- Target slower bandwidth links with two key strategies:
  - **Reduce** network traffic
  - **Manage** network traffic more efficiently
- We propose **NetCrafter**, a network crafting engine that:
  - **Stitches** partially empty flits to improve flit utilization
  - **Trims** network packets with redundant data
  - **Sequences** network traffic to prioritize latency-critical packets



Performance

- Up to 64% faster, and 16% average speedup over baseline non-uniform multi-GPU setups

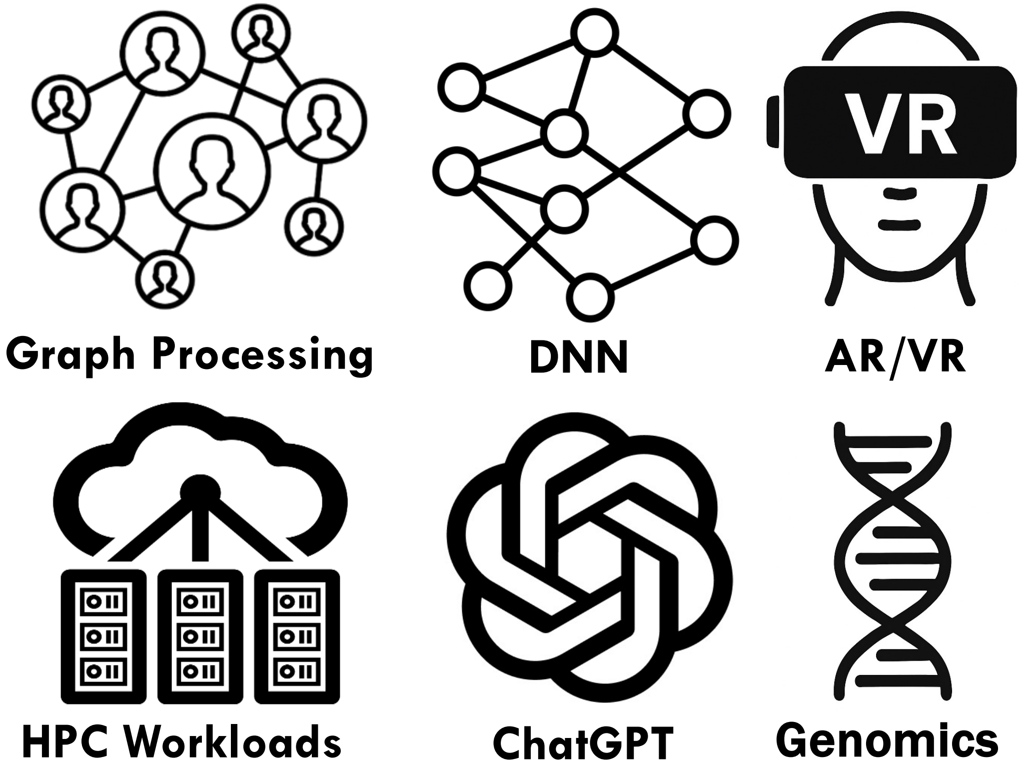
OUTLINE

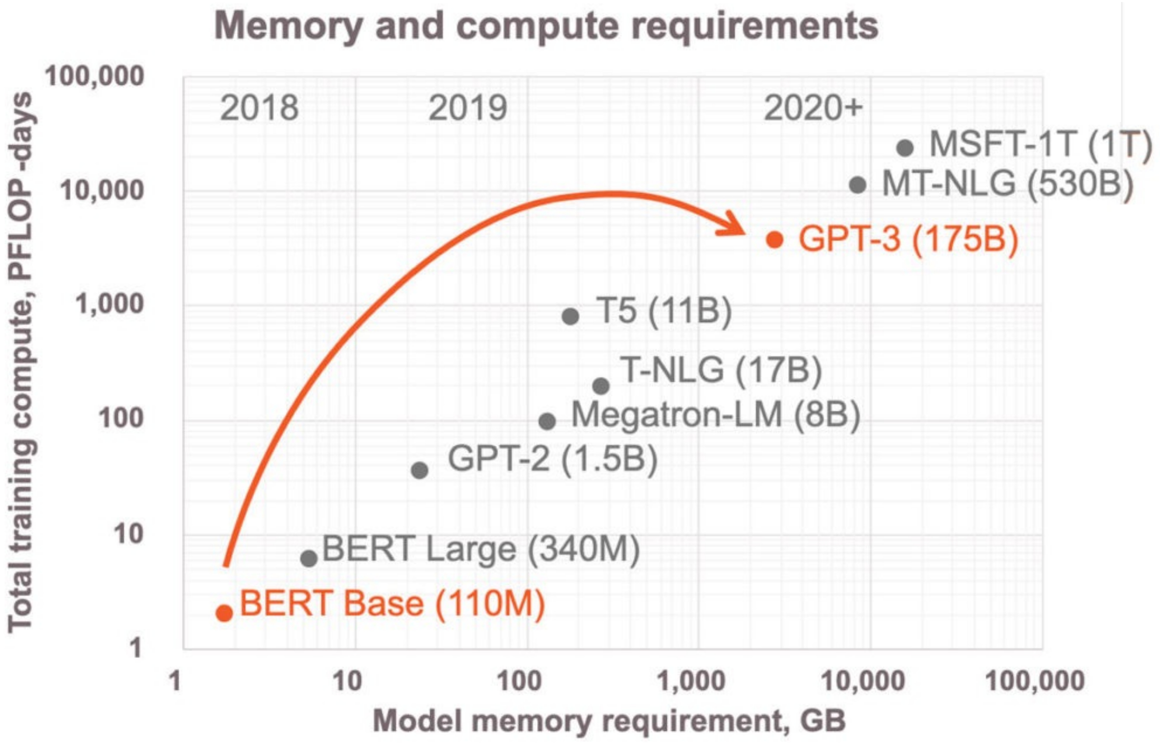
OUTLINE

- BACKGROUND AND MOTIVATION
- GOAL
- OBSERVATIONS & KEY IDEAS
- NETCRAFTER DESIGN
- EVALUATION
- CONCLUSION

BACKGROUND

GPUs are used to accelerate a wide range of applications



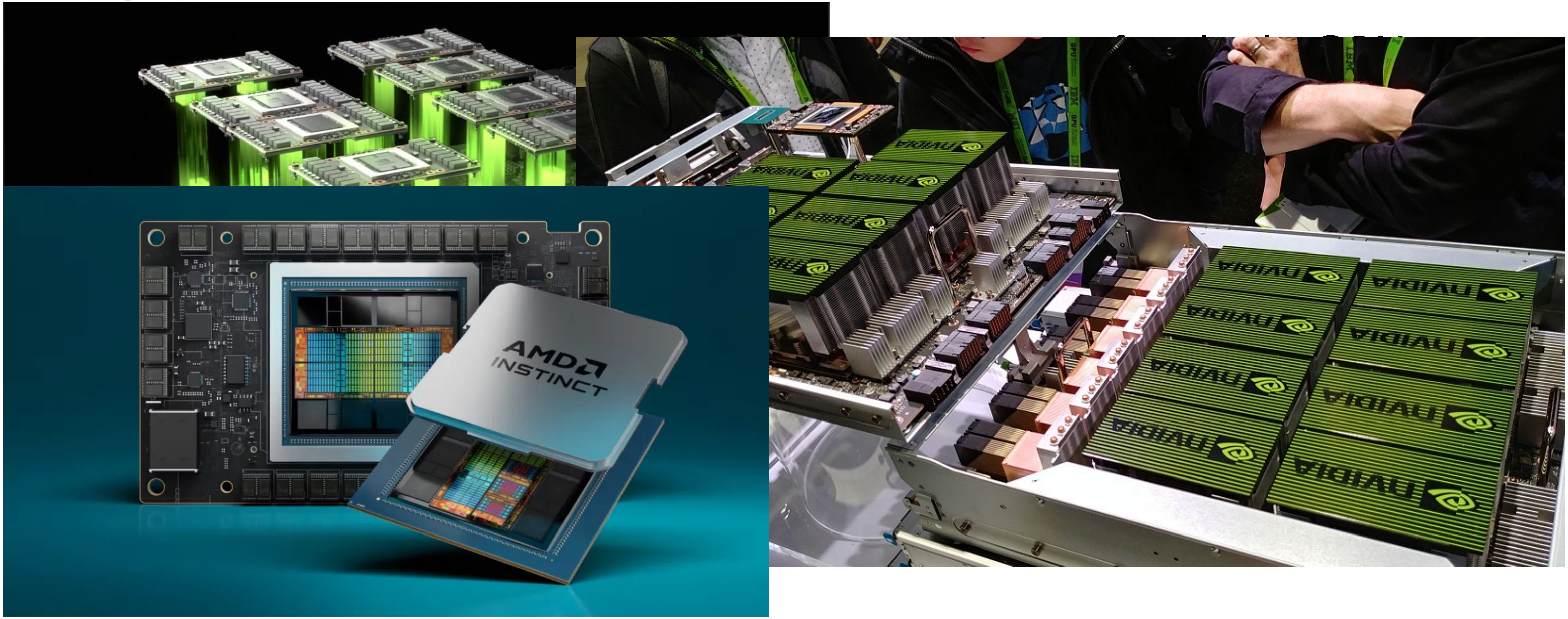


Rising application demands outgrow **single-GPU** capabilities

https://www.cerebras.net/blog/harnessing-the-power-of-sparsity-for-large-gpt-ai-models

MULTI-GPU SYSTEMS TO THE RESCUE

Multiple GPUs are connected over high bandwidth interconnects to scale:

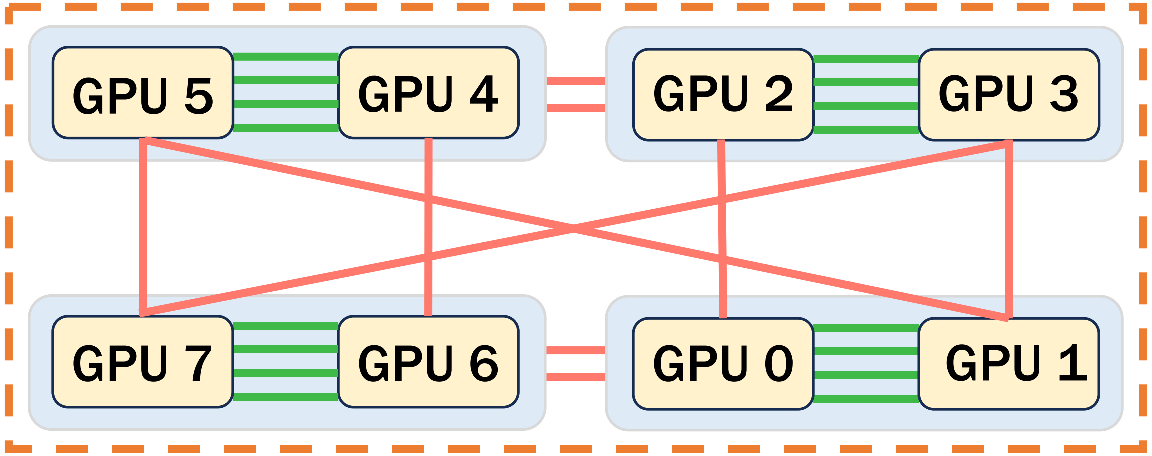


SCALING MULTI-GPU SYSTEMS

Multi-GPU systems must scale with growing application demands

These systems typically scale in a **hierarchical** manner

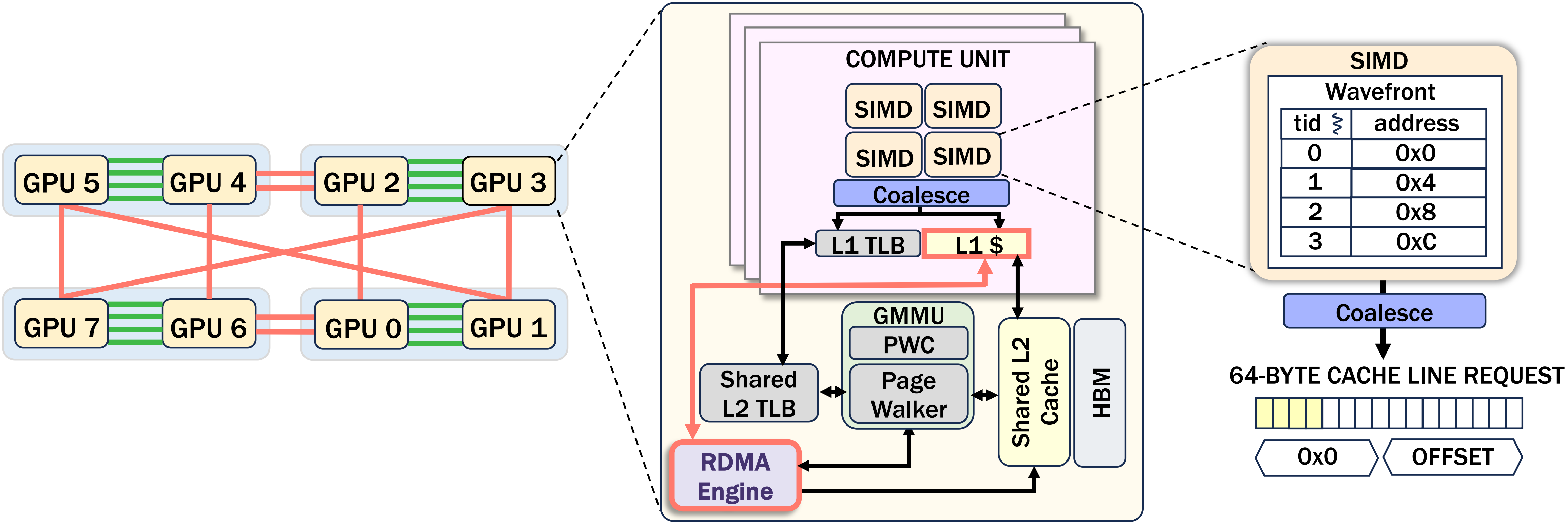
- Tightly coupled (e.g., MCM style): Linked via **higher-bandwidth** interconnects
- Loosely coupled (e.g., Multi-GPU style): Connected over **lower-bandwidth** interconnects



Multi-GPU scaling typically introduces bandwidth **non-uniformity** (e.g., Frontier, Summit, Aurora)

NON-UNIFORM BANDWIDTH -> NUMA

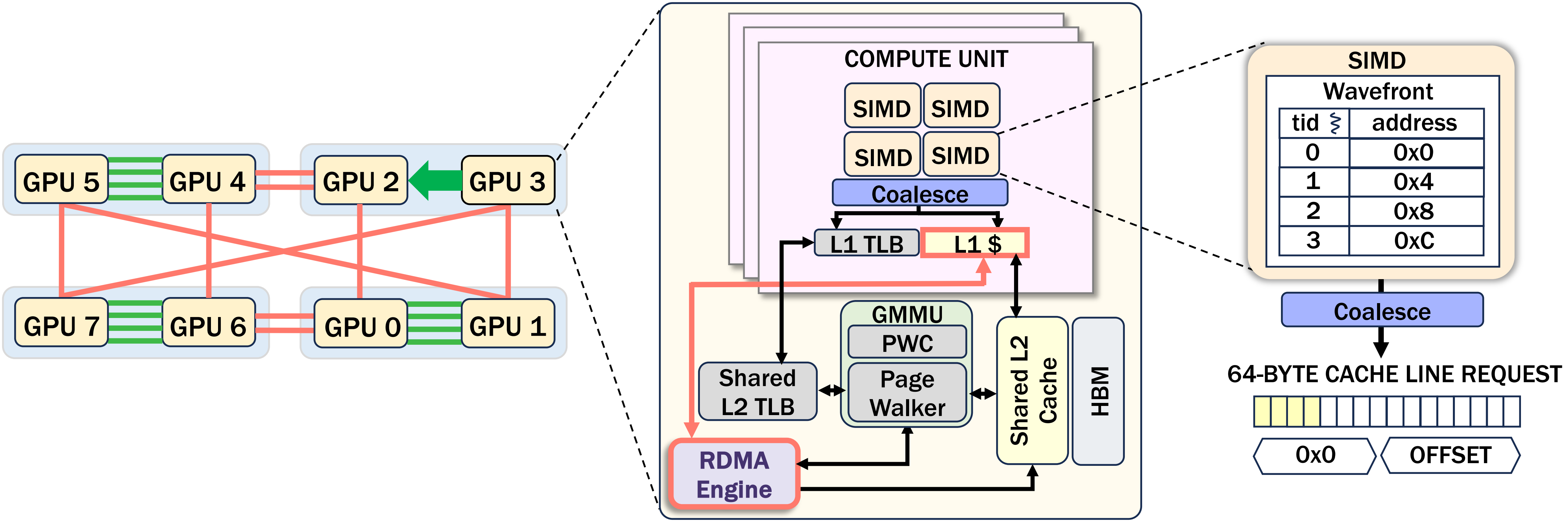
Memory access bandwidth varies across the system



NON-UNIFORM BANDWIDTH -> NUMA

Memory access bandwidth varies across the system

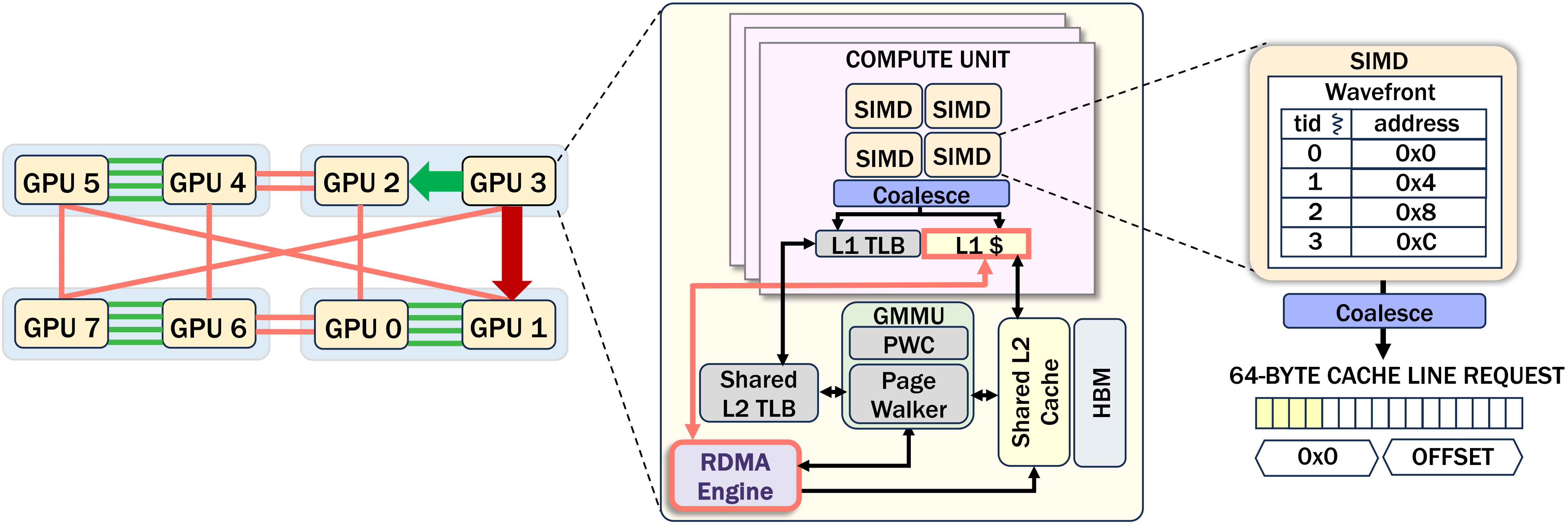
Nearby GPUs enjoy higher bandwidth and lower latency, and vice versa



NON-UNIFORM BANDWIDTH -> NUMA

Memory access bandwidth varies across the system

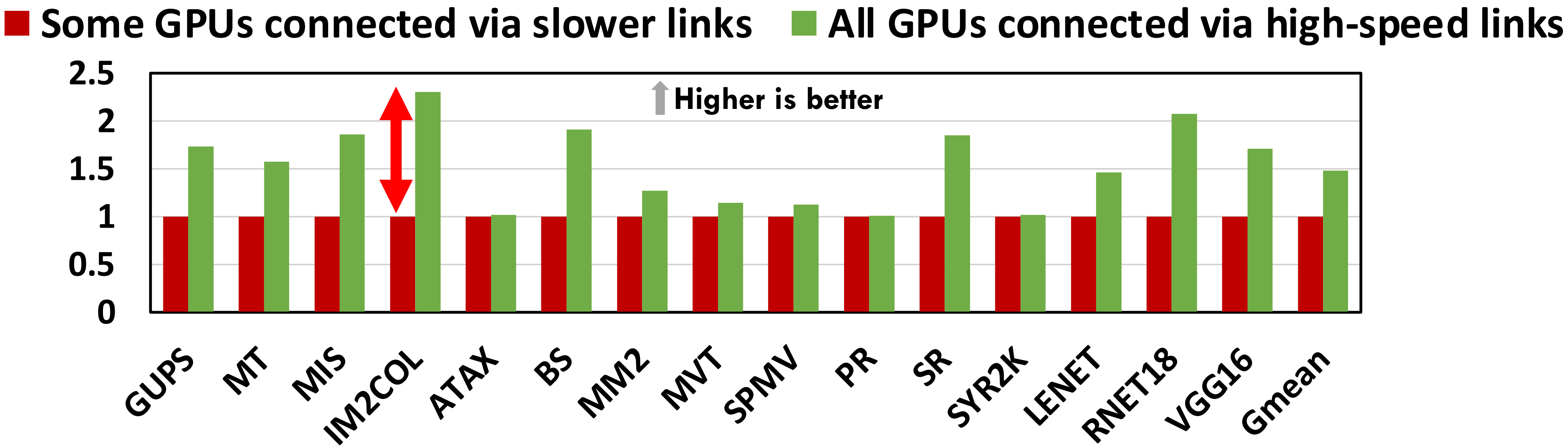
Nearby GPUs enjoy higher bandwidth and lower latency, and vice versa



NON-UNIFORM BANDWIDTH -> NUMA

Non-uniform bandwidth multi-GPU systems are constrained by:

Slower bandwidth interconnects connecting the GPUs



Non-uniform bandwidth causes up to **2.3×** slowdown in GPU performance

Slower interconnects lead to performance bottlenecks. How can we minimize the impact of **non-uniform bandwidth** in emerging **multi-GPU systems**?

OUTLINE

BACKGROUND AND MOTIVATION

GOAL

OBSERVATIONS & KEY IDEAS

NETCRAFTER DESIGN

EVALUATION

CONCLUSION

GOAL

Optimize **slower bandwidth links**:

1. **Reducing network traffic** across these links

2. **Efficiently managing network traffic** across these links

OUTLINE

BACKGROUND AND MOTIVATION

GOAL

OBSERVATIONS & KEY IDEAS

NETCRAFTER DESIGN

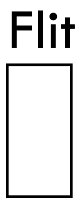
EVALUATION

CONCLUSION

Transmitting packets across the network

Network Transfer through Flits

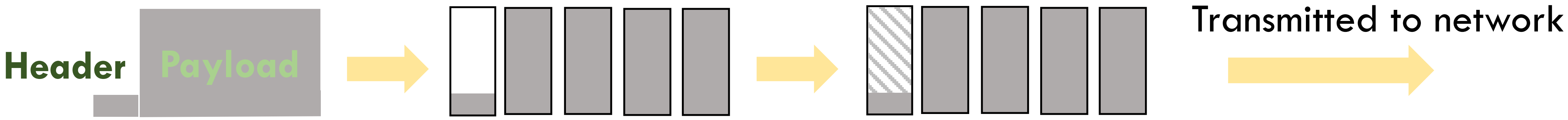
- Flit = fixed-size unit (x **bytes**), sent per cycle
- **Basic granularity** of data transfer in interconnects



GPU packet breakdown

- Packet = **Header** + **Payload**
- Split into flits (fixed-size units)
- Empty bytes padded to **align** the last flit with flit size

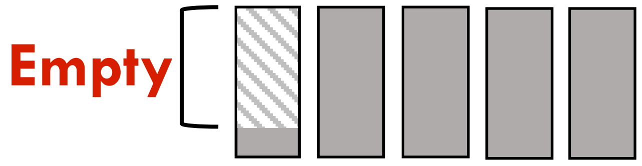
GPU request Packet



OBSERVATION #01

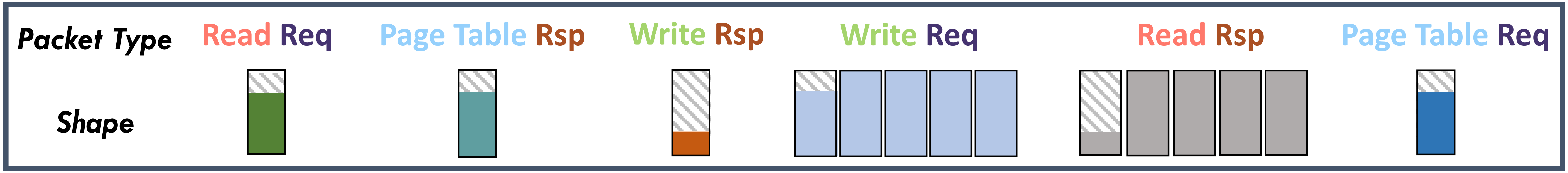
Inefficient Flit Utilization Due to Padding

- Flits contain substantial empty bytes
- These empty bytes are padded with redundant data
- Redundant data increases unnecessary network load



Each packet type contributes differently to this network load

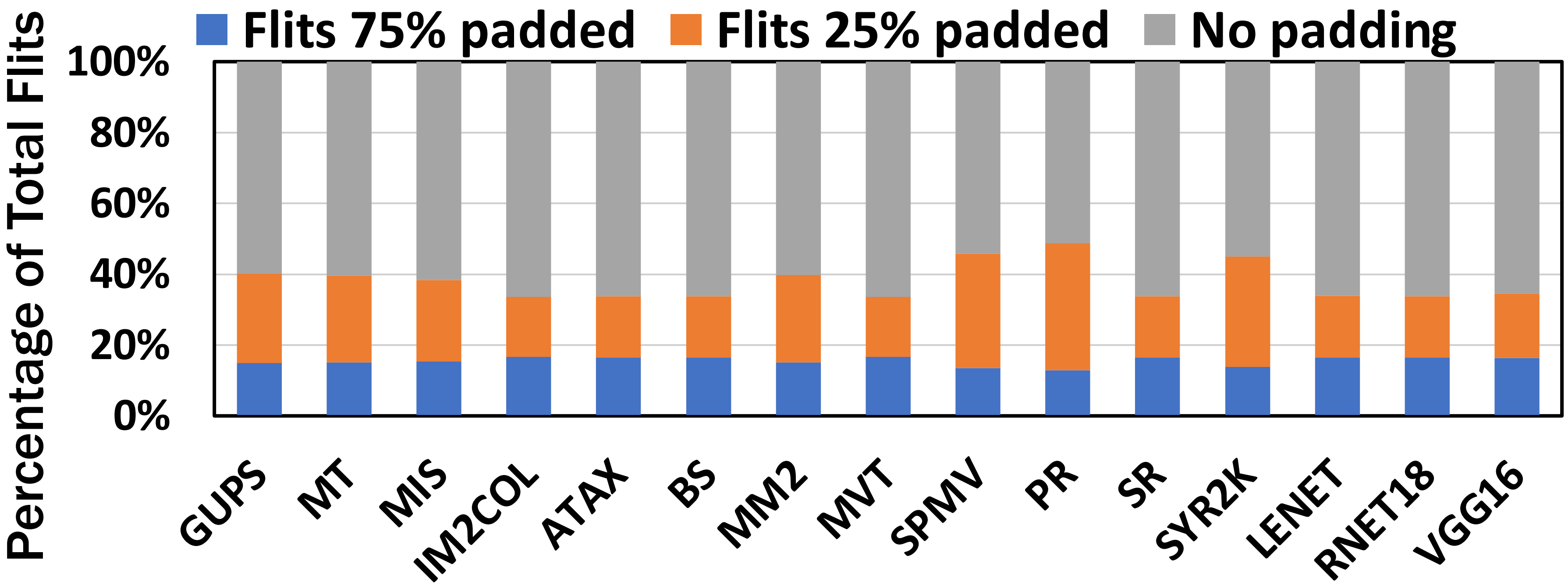
- Read Req (Header + Address)
- Write Req (Header + Data + Address)
- Page Table Req (Header +Address)
- Read Rsp (Header + Data)
- Write Rsp (Header)
- Page Table Rsp (Header + Address)



OBSERVATION #01

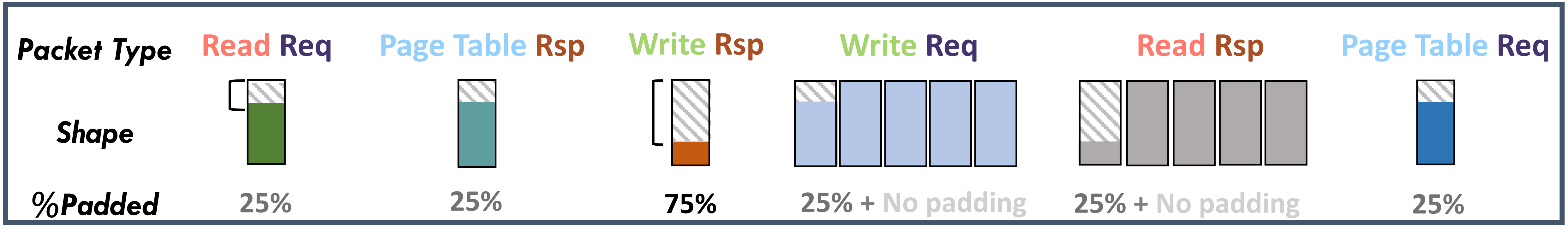
Different packet types introduce varying padding overhead

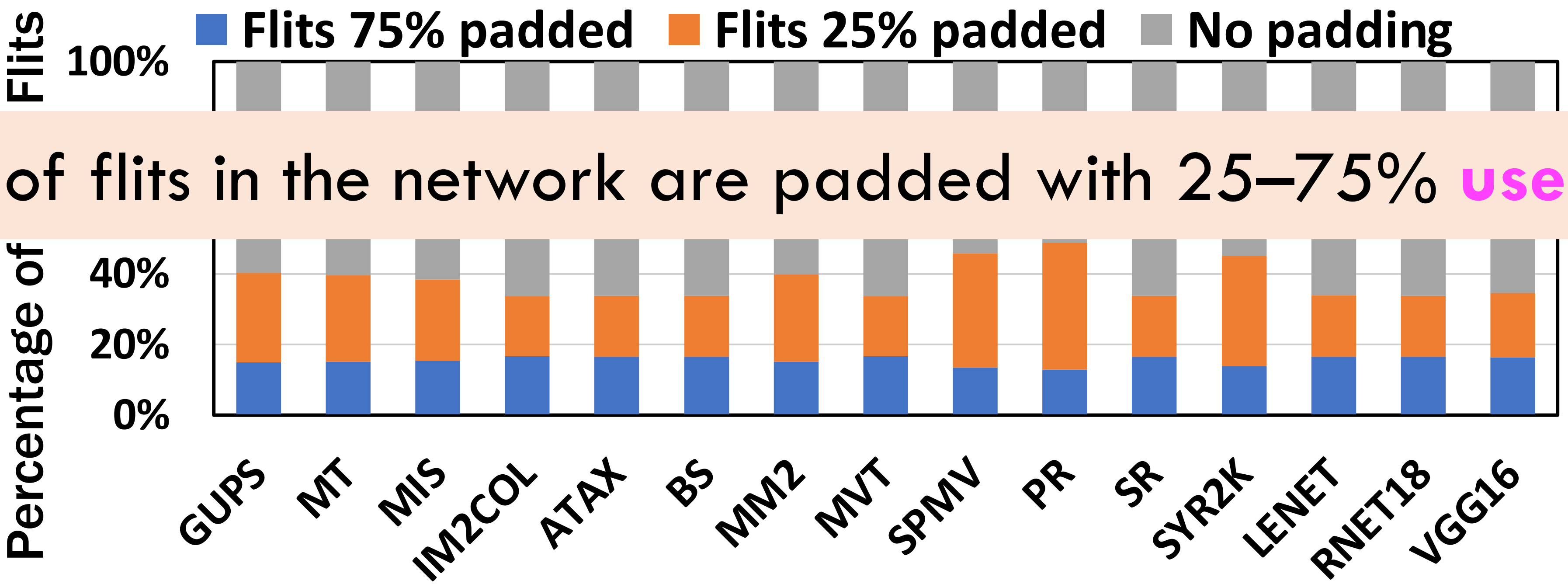
Packet Type	Read Req	Page Table Rsp	Write Rsp	Write Req	Read Rsp	Page Table Req
Shape						
%Padded	25%	25%	75%	25% + No padding	25% + No padding	25%



OBSERVATION #01

Different packet types introduce varying padding overhead





~40% of flits in the network are padded with 25–75% **useless data**

KEY IDEA I: STITCHING

Stitch flits across request categories to reduce network traffic overhead

KEY IDEA I: STITCHING

Stitch flits across request categories to reduce network traffic overhead

KEY IDEA I: STITCHING

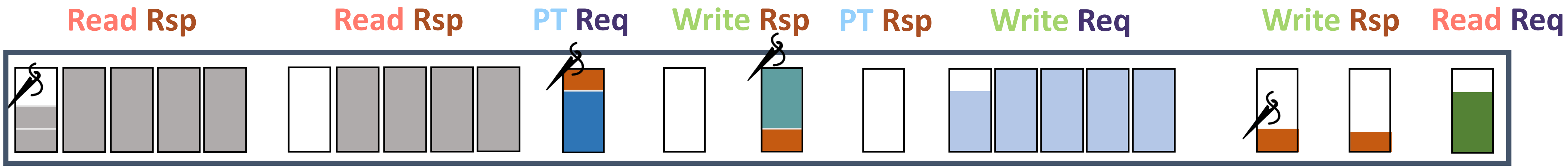
Stitch flits across request categories to reduce network traffic overhead

KEY IDEA I: STITCHING

Stitch flits across request categories to reduce network traffic overhead

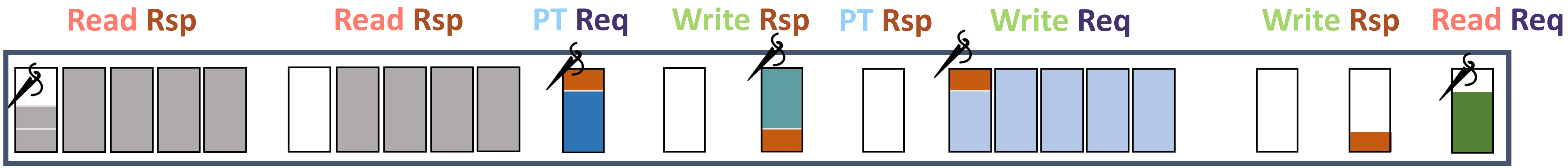
KEY IDEA I: STITCHING

Stitch flits across request categories to reduce network traffic overhead



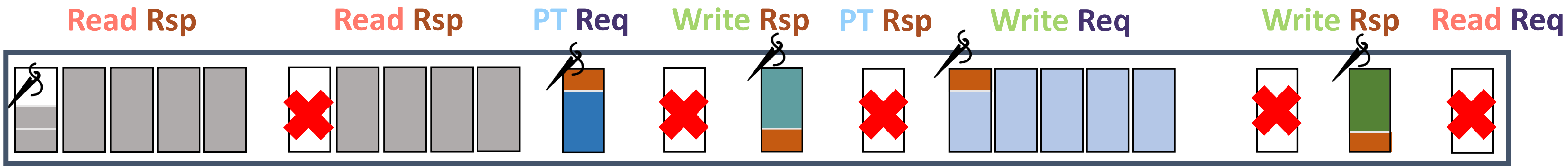
KEY IDEA I: STITCHING

Stitch flits across request categories to reduce network traffic overhead



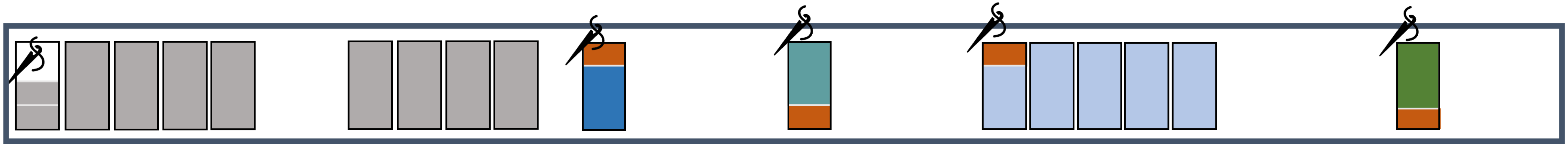
KEY IDEA I: STITCHING

Stitch flits across request categories to reduce network traffic overhead



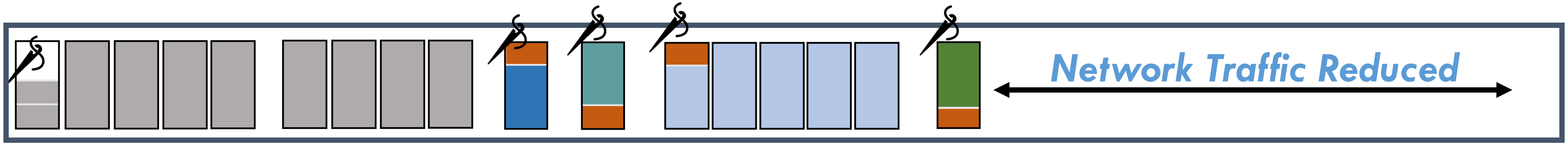
KEY IDEA I: STITCHING

Stitch flits across request categories to reduce network traffic overhead



KEY IDEA I: STITCHING

Stitch flits across request categories to reduce network traffic overhead

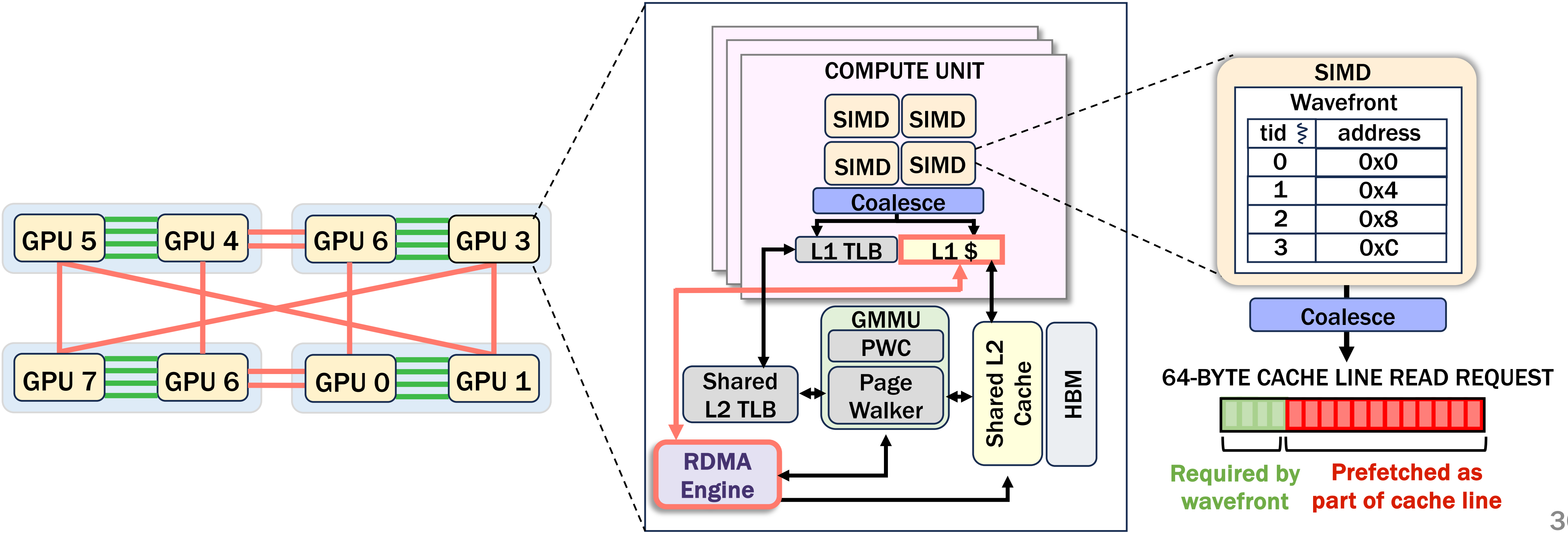


OBSERVATION #02

Prefetched unused data wastes bandwidth

OBSERVATION #02

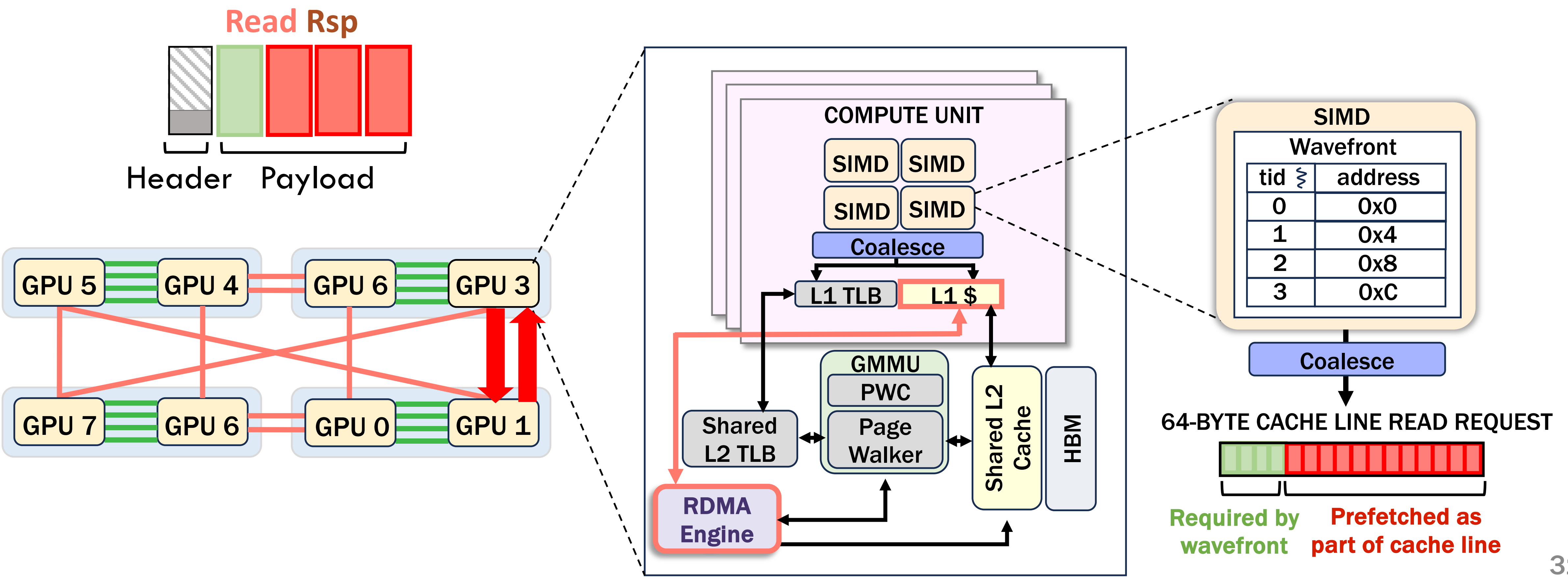
Prefetched unused data wastes bandwidth



OBSERVATION #02

Prefetched unused data wastes bandwidth

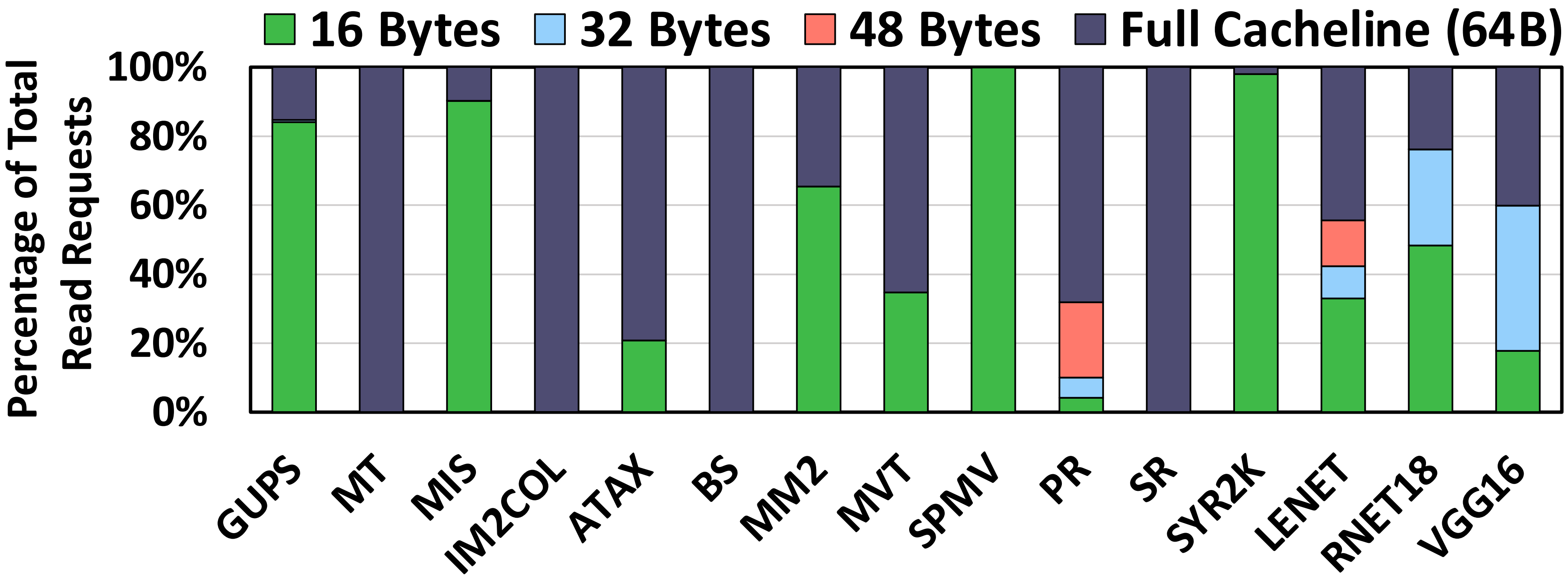
- Read response includes more data than requested



OBSERVATION #02

Prefetched unused data wastes bandwidth

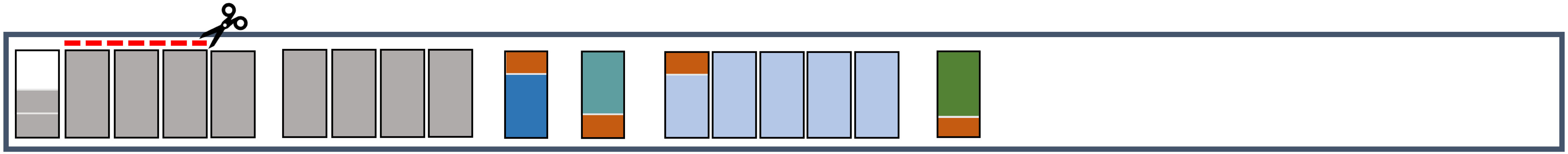
- Read response includes more data than requested



Most applications require ≤16B from each 64B GPU cache line

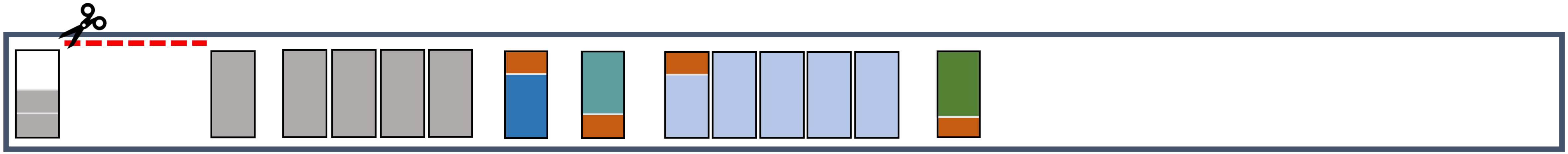
KEY IDEA II: TRIMMING

Trim redundant flits to cut network overhead



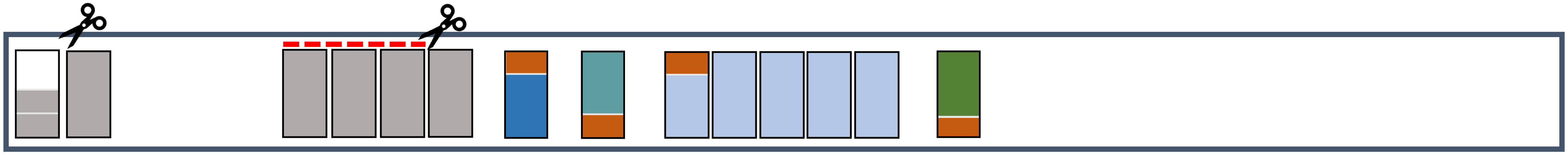
KEY IDEA II: TRIMMING ✂

Trim redundant flits to cut network overhead



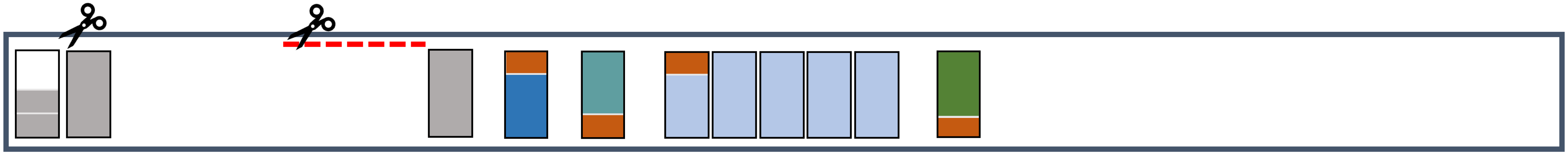
KEY IDEA II: TRIMMING

Trim redundant flits to cut network overhead



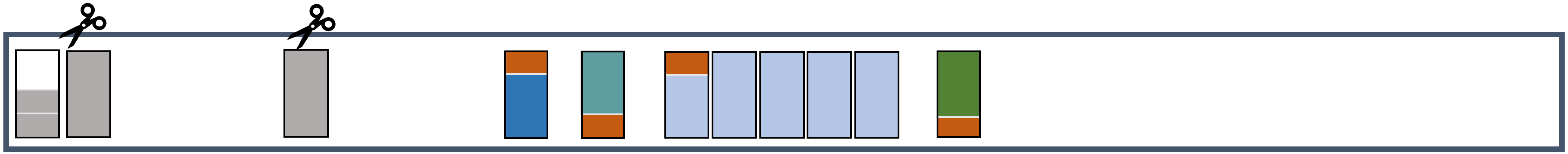
KEY IDEA II: TRIMMING

Trim redundant flits to cut network overhead



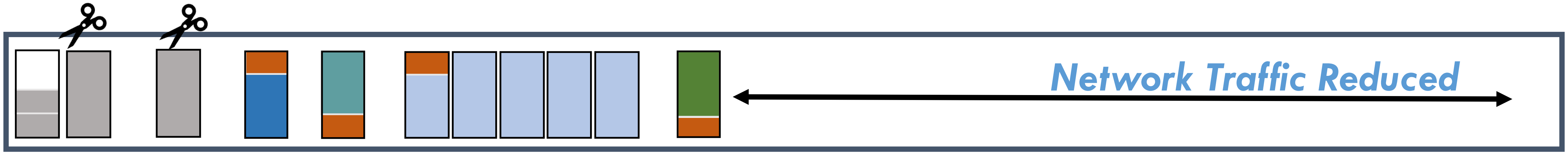
KEY IDEA II: TRIMMING

Trim redundant flits to cut network overhead



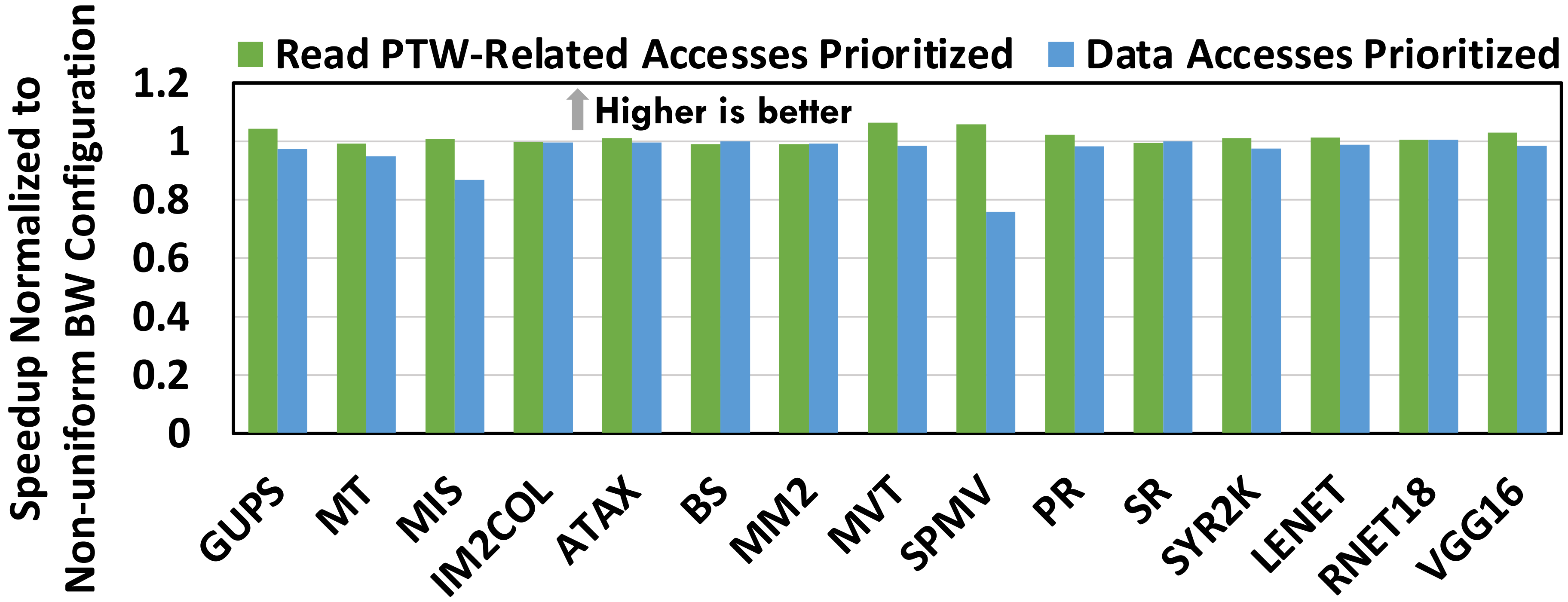
KEY IDEA II: TRIMMING

Trim redundant flits to cut network overhead



OBSERVATION#03

PTW-related flits are more latency critical than other flits

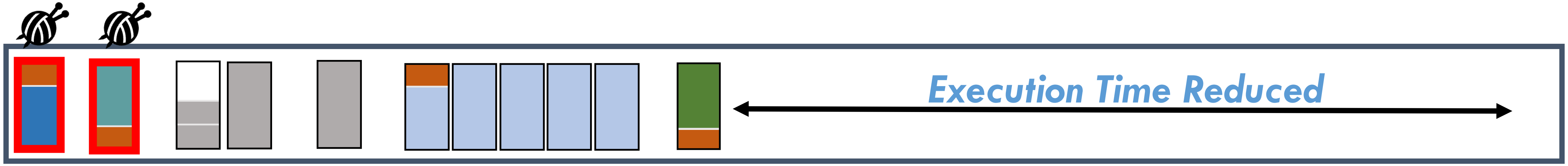


KEY IDEA III: SEQUENCING

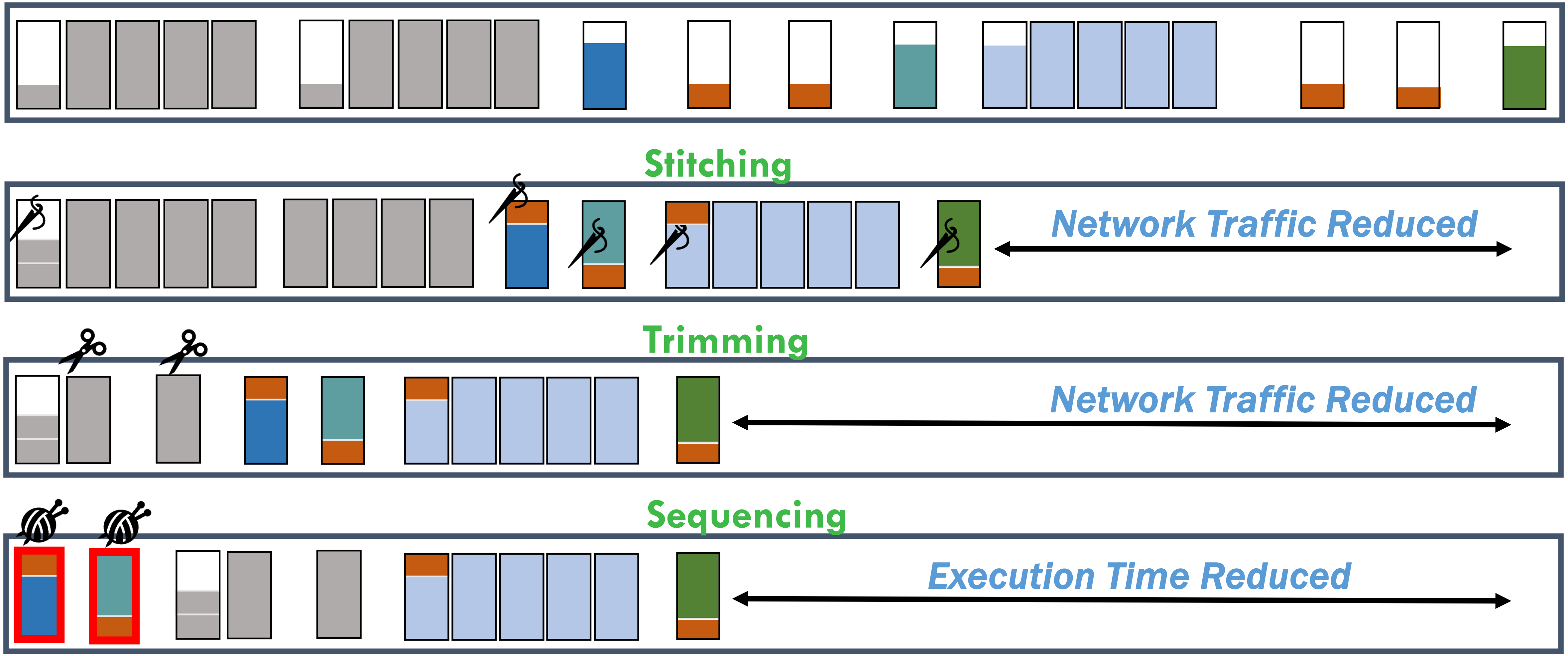
Prioritize flits in flight so latency-critical packets lead the line on slow links

KEY IDEA III: SEQUENCING

Prioritize flits in flight so latency-critical packets lead the line on slow links



PUTTING IT TOGETHER



1. Reducing network traffic across these links
2. Efficiently managing network traffic across these links

OUTLINE

BACKGROUND AND MOTIVATION

GOAL

OBSERVATIONS & KEY IDEAS

NETCRAFTER DESIGN

EVALUATION

CONCLUSION

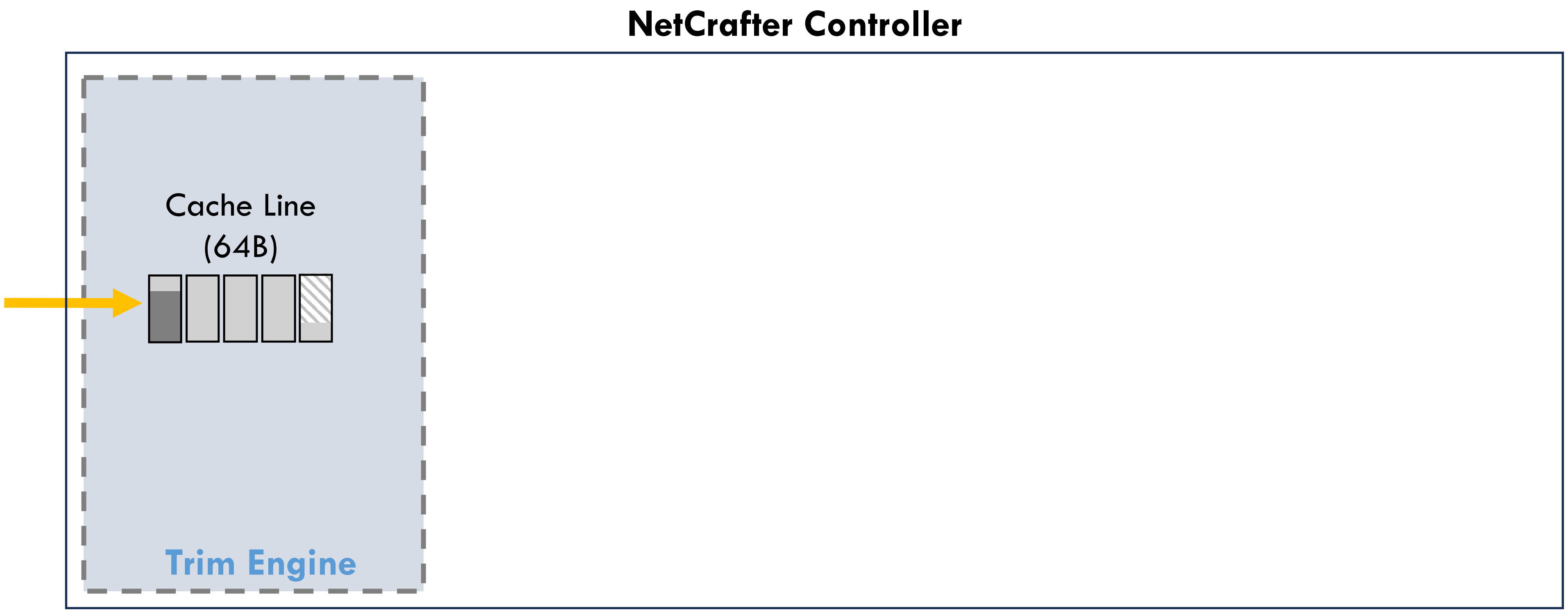
Mechanism: NetCrafter

Each cluster switch includes an integrated NetCrafter controller

NetCrafter Controller

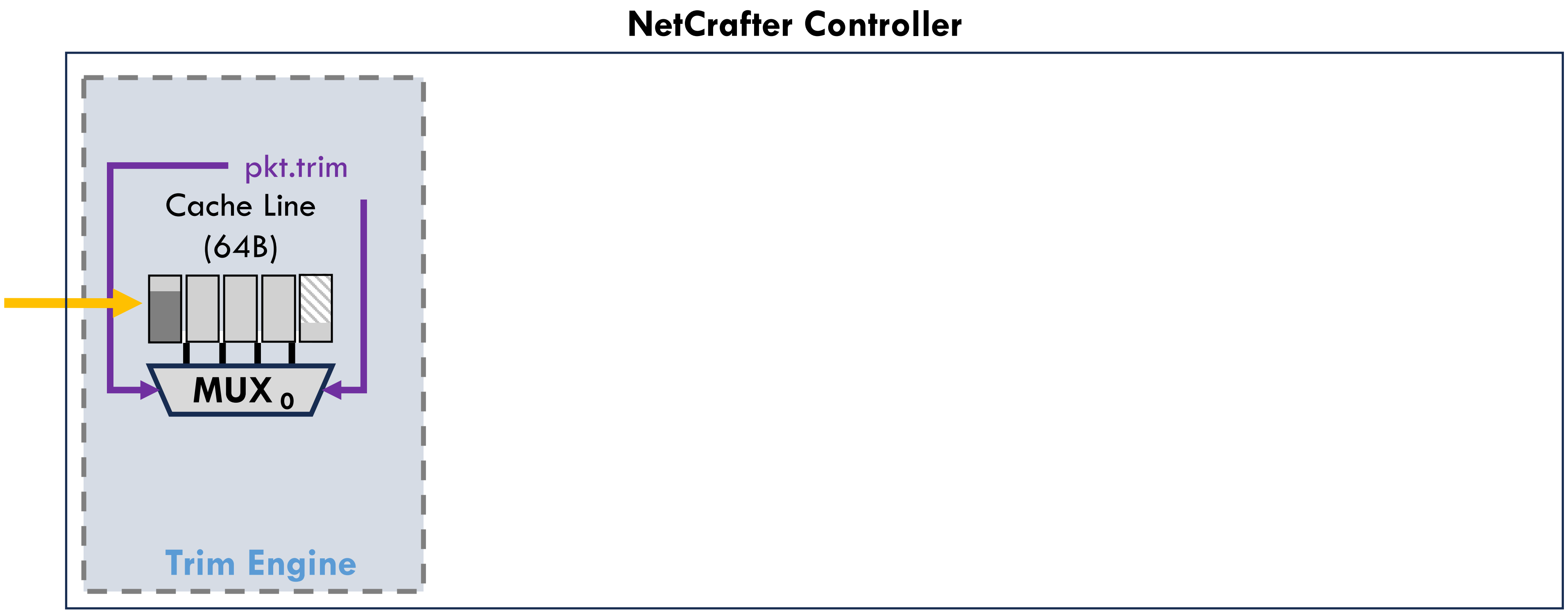
Mechanism: NetCrafter

Each cluster switch includes an integrated NetCrafter controller



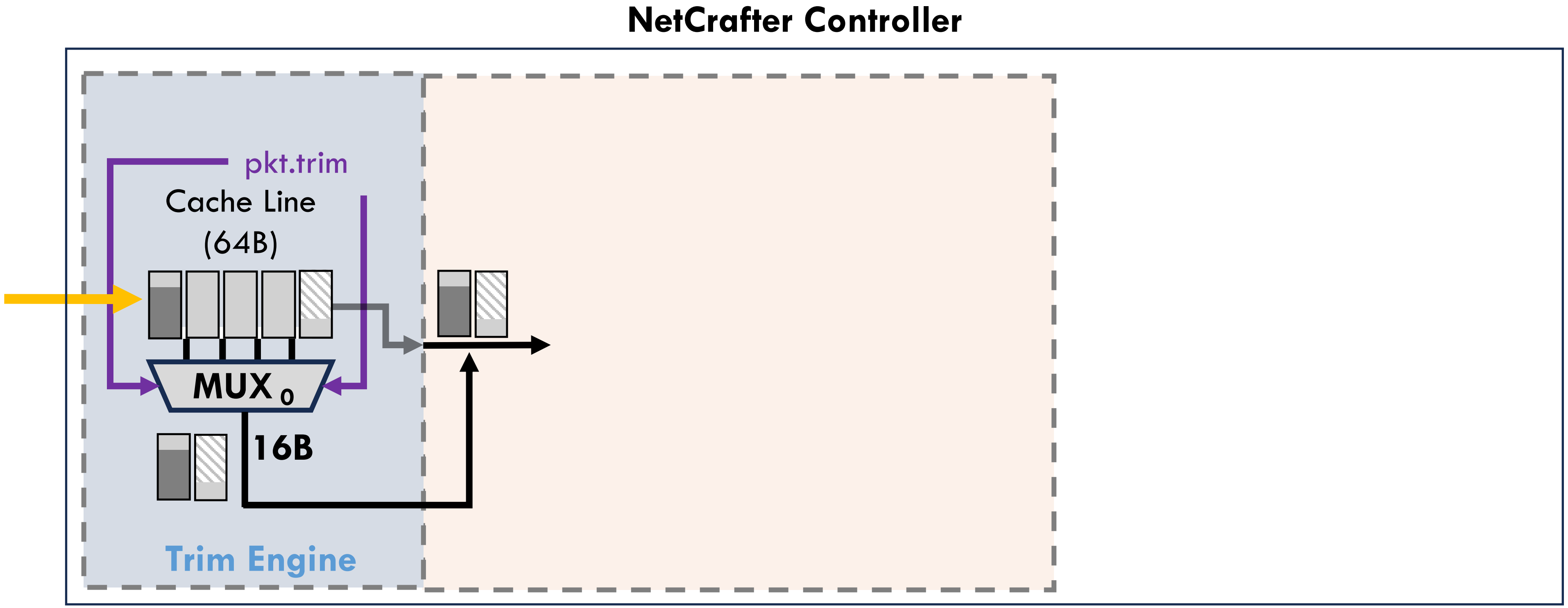
Mechanism: NetCrafter 🪡✂️🧶

Each cluster switch includes an integrated NetCrafter controller



Mechanism: NetCrafter 🪡✂️🧶

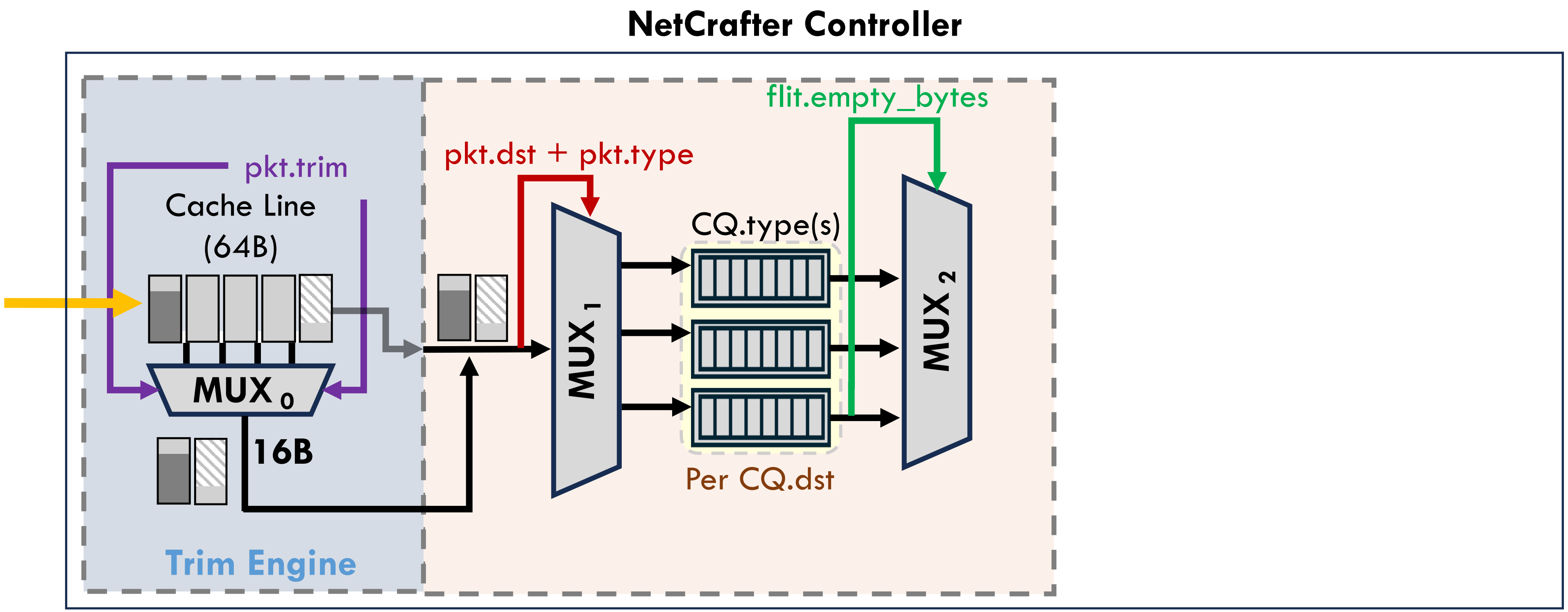
Each cluster switch includes an integrated NetCrafter controller



Decide the granularity

Mechanism: NetCrafter

Each cluster switch includes an integrated NetCrafter controller

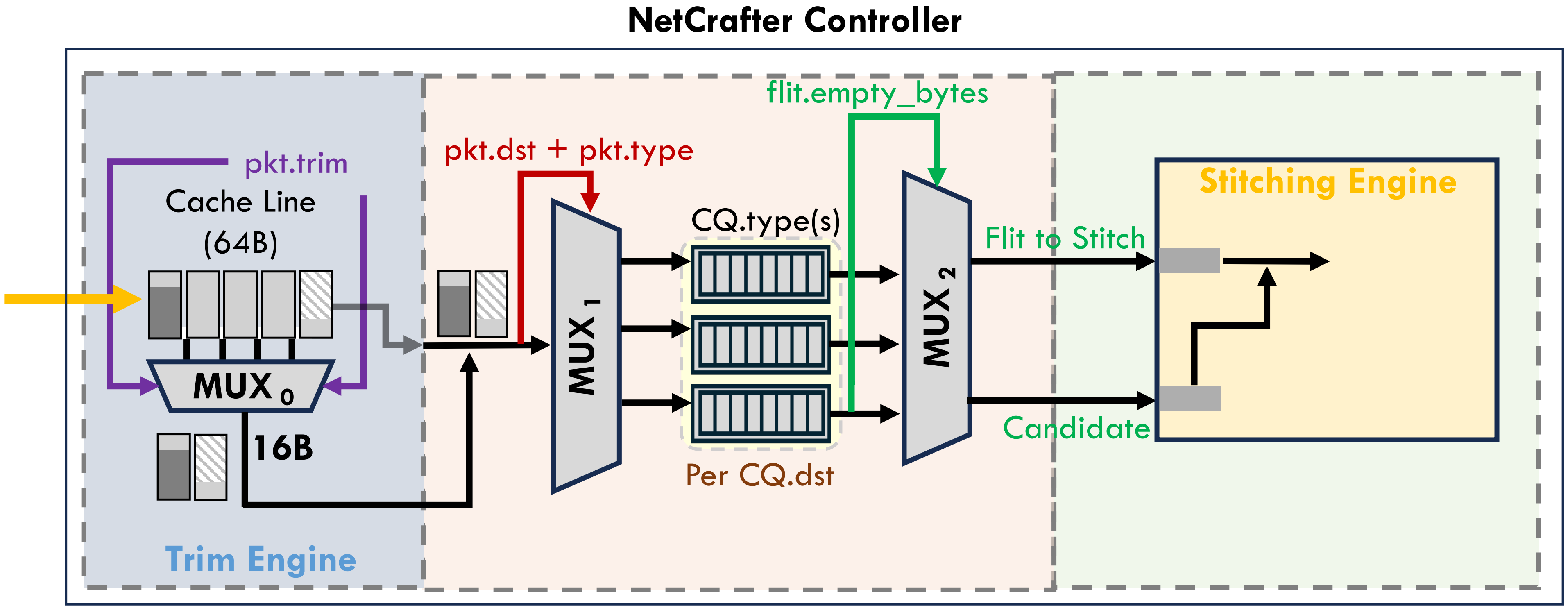


Decide the granularity

Choose the stitching candidates

Mechanism: NetCrafter 🪡✂️🧶

Each cluster switch includes an integrated NetCrafter controller

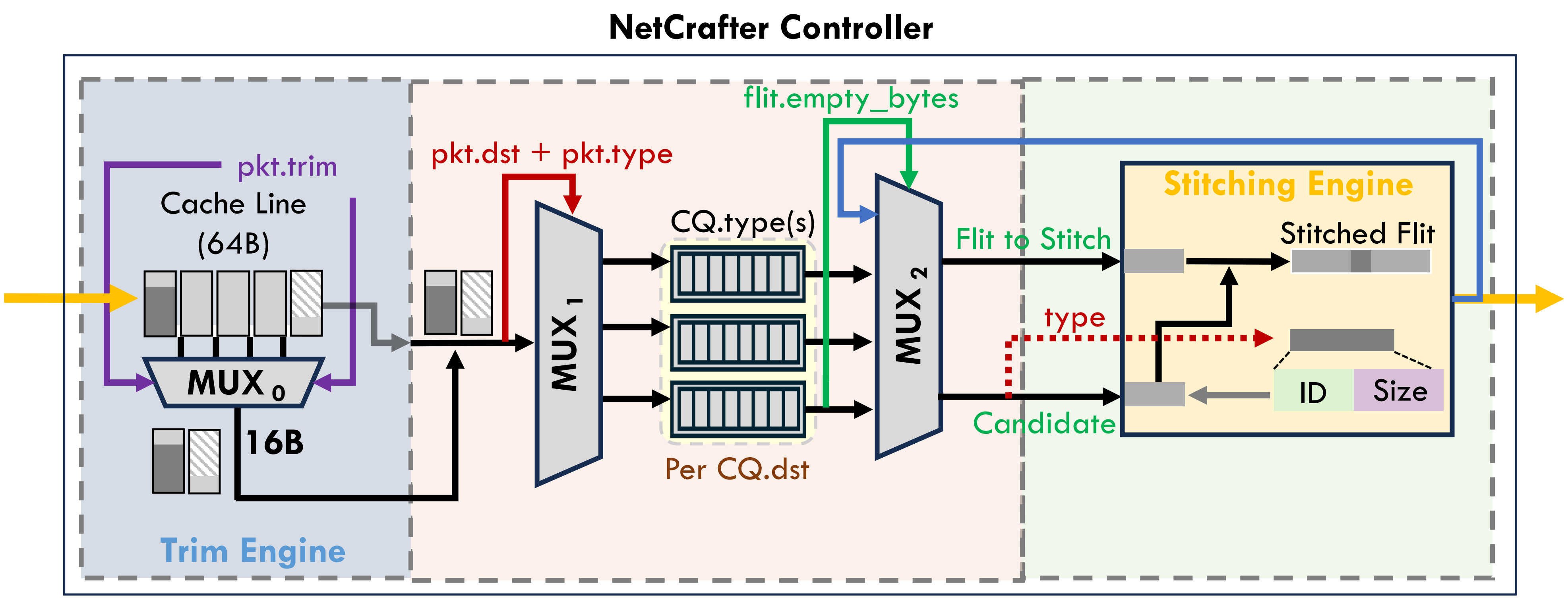


Decide the granularity

Choose the stitching candidates

Mechanism: NetCrafter

Each cluster switch includes an integrated NetCrafter controller



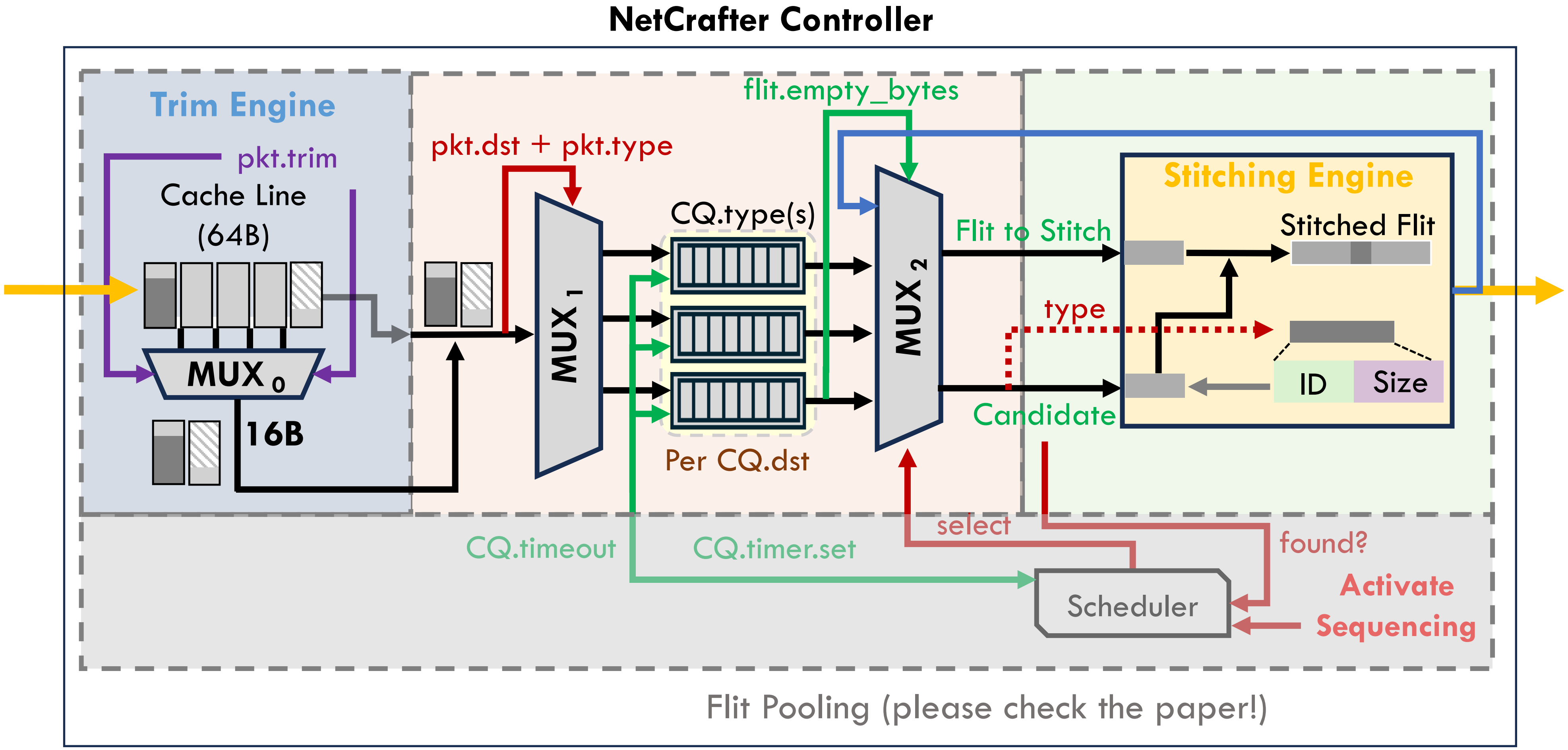
Decide the granularity

Choose the stitching candidates

Packet support + Sequence

Mechanism: NetCrafter

Each cluster switch includes an integrated NetCrafter controller



OUTLINE

- BACKGROUND AND MOTIVATION
- GOAL
- OBSERVATIONS & KEY IDEAS
- NETCRAFTER DESIGN
- EVALUATION
- CONCLUSION

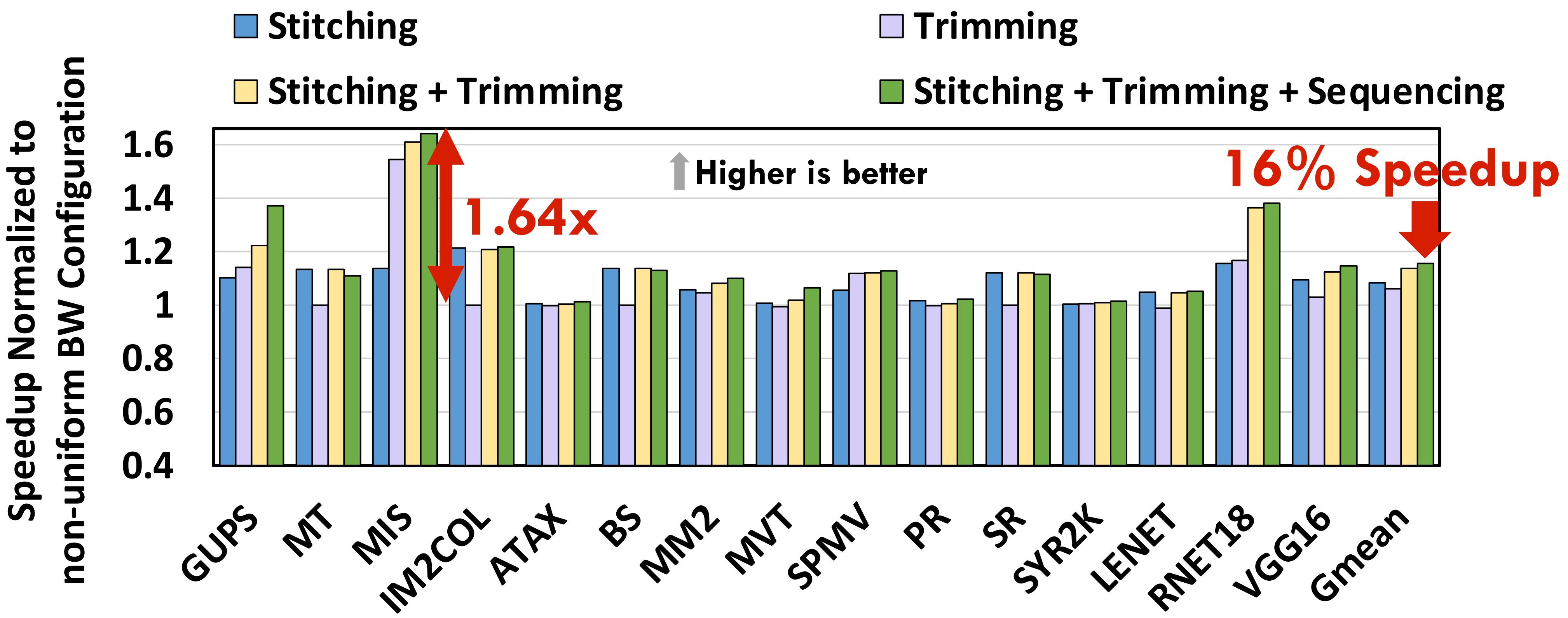
METHODOLOGY

- **MGPUSim Simulator:**

Compute Unit	1GHz, 64 per GPU (4 GPUs in total)
L1 D/I, L2 cache	64/32KB, 4MB per GPU (shared)
Heterogeneous Interconnect	Inter-GPU-cluster - 16GBps, bi-directional Intra-GPU-cluster - 128GBps, bi-directional
CTA/Page Scheduling	LASP (Locality-Aware Scheduling and Placement)
NetCrafter Parameters	Cluster Queue - 1024 Entries (16B each)

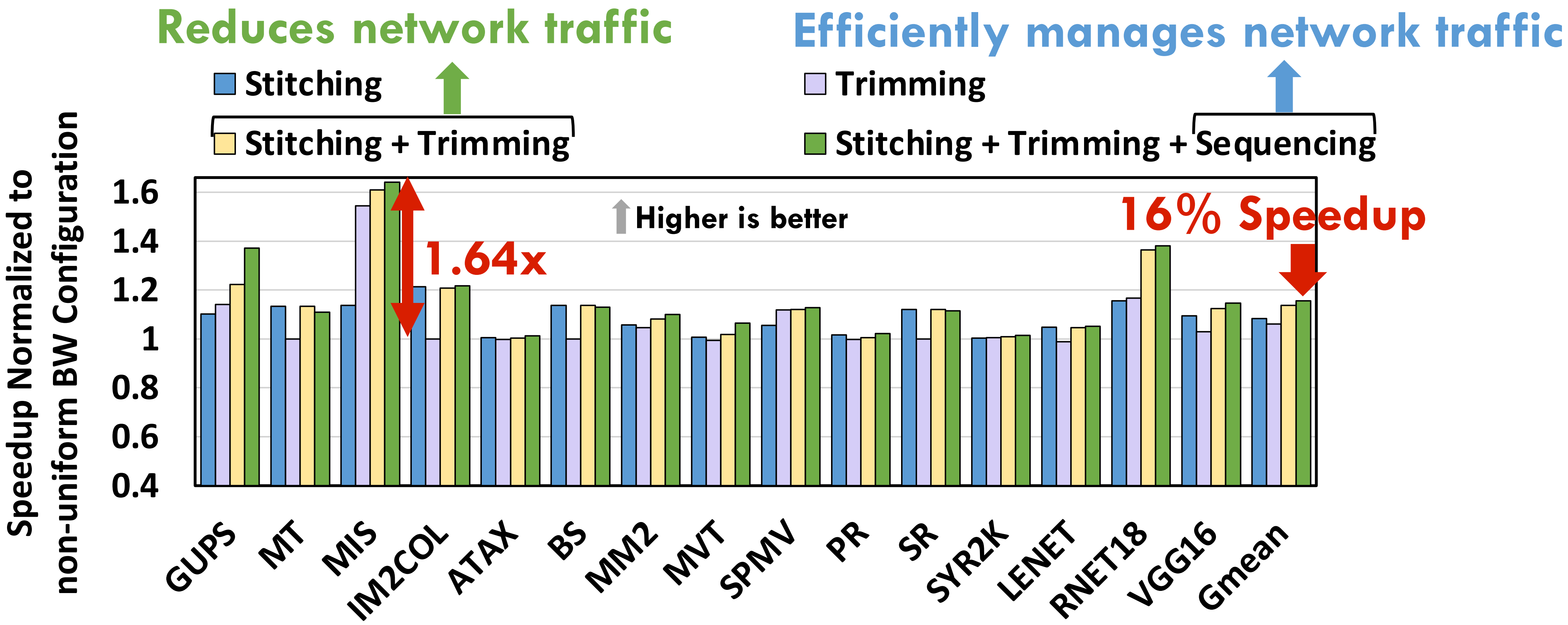
- **Design points:**
  - Baseline: non-uniform bandwidth configuration without NetCrafter
  - Sector Cache: baseline configuration with sector cache
  - NetCrafter: non uniform system with **Stitching, Trimming & Sequencing**

NetCrafter vs. Baseline



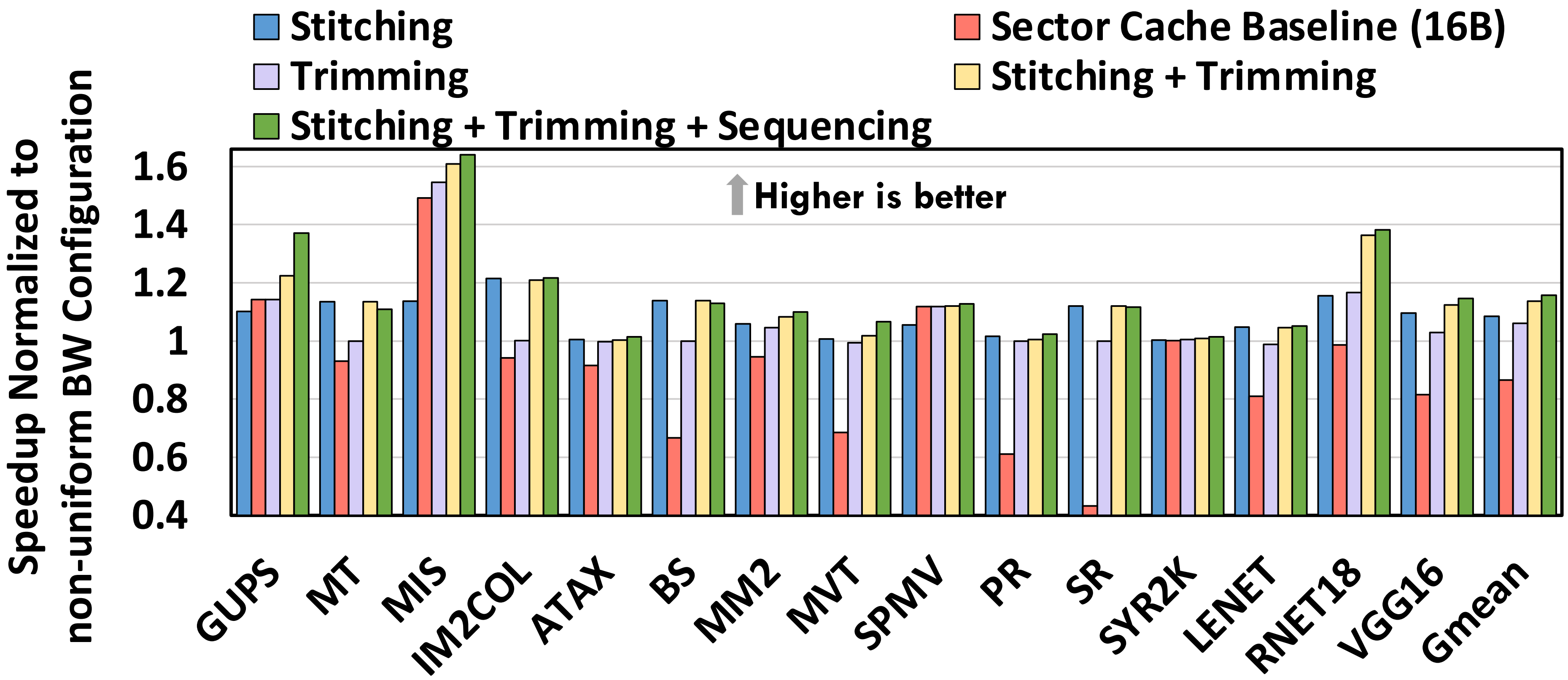
NetCrafter provides 16% speedup on average

NetCrafter vs. Baseline



NetCrafter provides 16% speedup on average

NetCrafter vs. Baseline + Sector Cache



NetCrafter provides 16% speedup on average

More Details…

- Flit Pooling and Selective Flit Pooling
- Impact of trimming on cache MPKI
- Modifications to the packet structure
- Hardware overhead
- Latency overhead
- CTA/data placement
- Coherence implications
- Handling deadlocks
- Packet unstitching details at the target
- Sensitivity studies with varied :
  - Flit-Pooling windows
  - Bandwidth numbers and ratios
  - Flit sizes

Please check our paper!

OUTLINE

- BACKGROUND AND MOTIVATION
- GOAL
- OBSERVATIONS & KEY IDEAS
- NETCRAFTER DESIGN
- EVALUATION
- CONCLUSION

Conclusion

NetCrafter, a combination of novel approaches (***Stitching***, ***Trimming*** and ***Sequencing***) to deal with the **multi-GPU** network traffic, which provides up to **64%** speedup and **16%** speedup on average.

Our proposed techniques are ***generic*** and can be applied to ***any*** network. They are especially useful in alleviating the bottlenecks presented by ***lower-bandwidth networks*** connecting multiple groups of GPUs.

Thanks!
Questions?

NetCrafter

Tailoring Network Traffic for Non-Uniform Bandwidth Multi-GPU Systems

Amel Fatima[1], Yang Yang[1], Yifan Sun[2], Rachata Ausavarungnirun[3], and Adwait Jog[1]

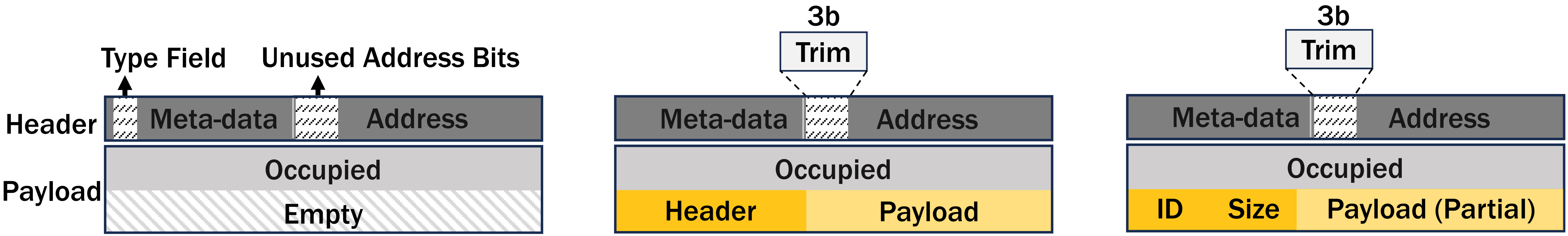
1 University of Virginia

2 William & Mary
CHARTERED 1693

3 King Mongkut's University of Technology North Bangkok

BACKUP SLIDES

Packet structure to support NetCrafter

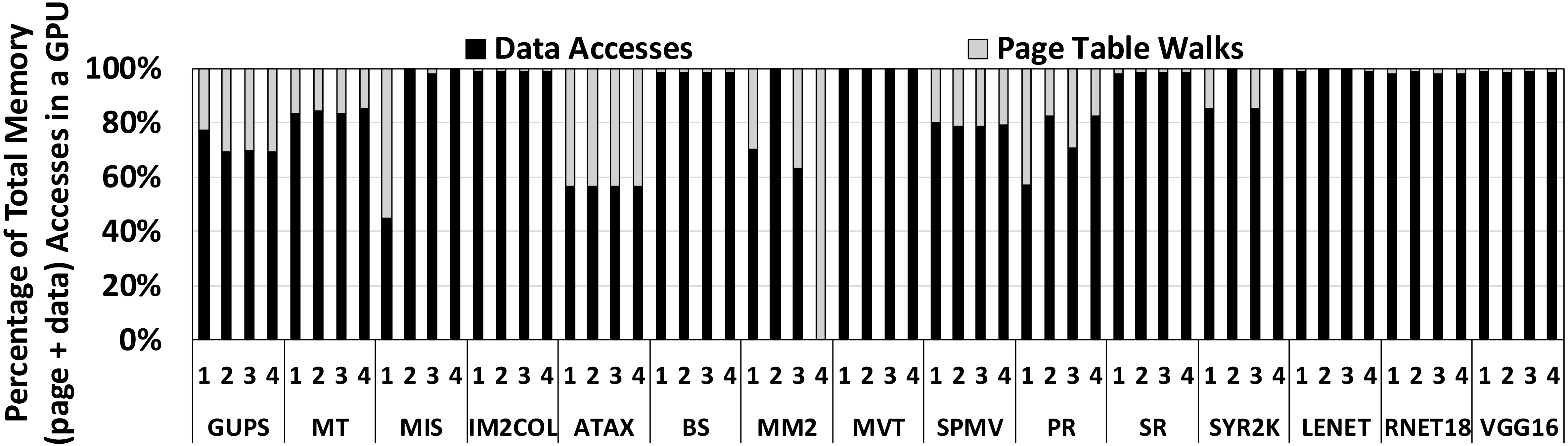


a) A typical PCIe Packet

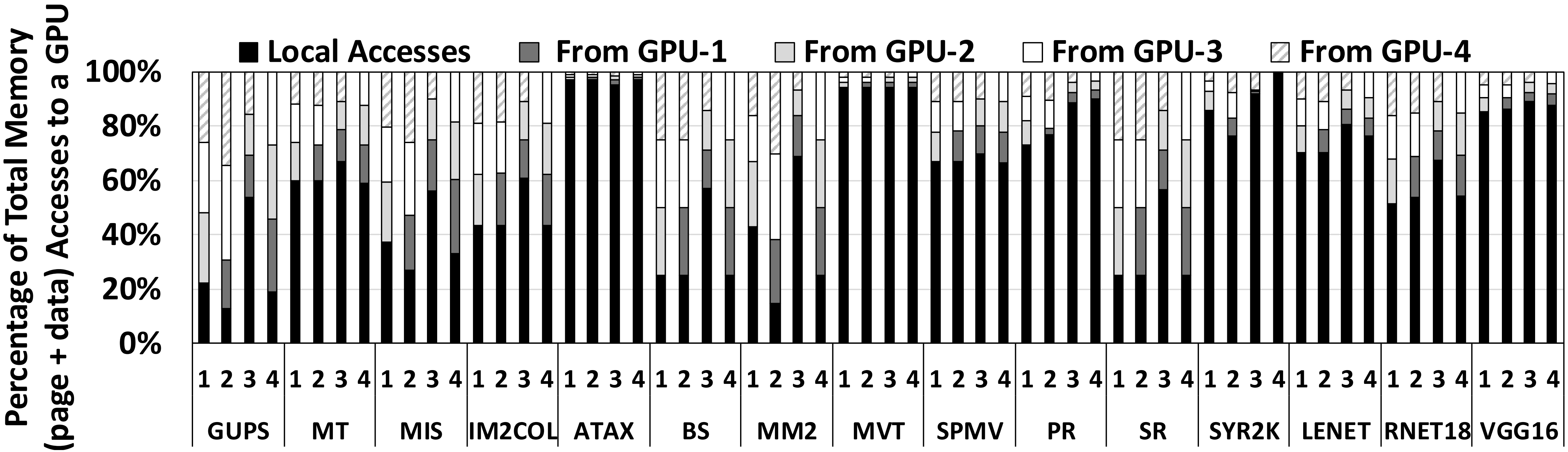
b) Stitching a complete packet

c) Stitching a partial packet

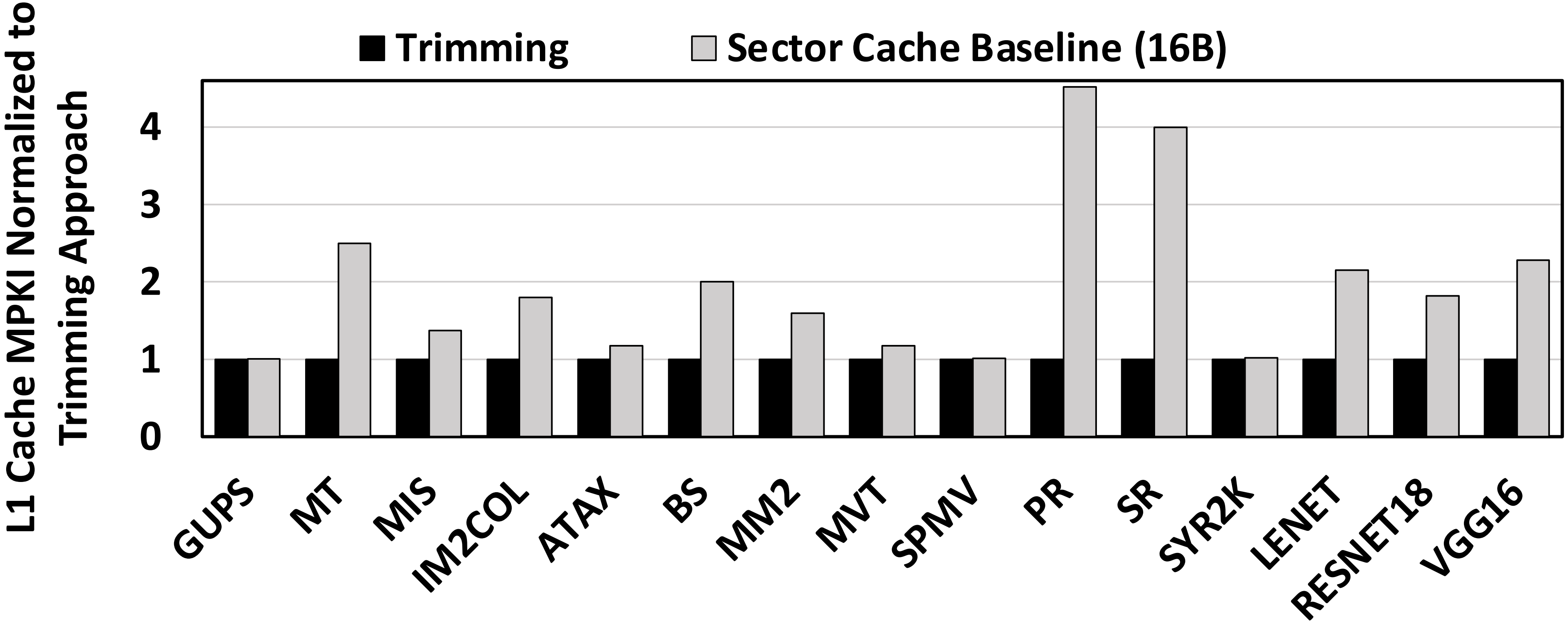
Ratio of Data vs Page Table Accesses per GPU



Distribution of Memory Access Requests/GPU



Cache MPKI Comparison with Sector Cache



```
Parameters      : Qx holding different packets
function STITCH ();
begin
  while True do
    /* select a valid queue through scheduler */
    while Qi.empty() or Qi.timer() do
      Qi ← Sched (QA, QB, QPTW);
    end
    pkt ← Qi[head];
    /* compute empty bytes in the last flit */
    sz ← SZflit-sizeof(pkt)%SZflit;
    /* last flit size equals sz in Qj */
    Qj ← PARTITION_QUEUE(sz);
    sc ← STITCH_CANDIDATE(sz, Qj);
    if sc == nil then
      /* prioritized packet */
      if pkt.type==latency_critical or
      sc.type==0 then
        Eject (pkt);
      end
      else if Qi.timer.timeout() then
        /* only wait for one period */
        Qi.timer.clear();
        Eject (pkt);
      end
      else
        /* wait for a period in wish to find a candidate */
        Qi.timer.set();
      end
    end
    else
      /* add sub-header for type-2 candidate */
      if sc.type==2 then
        pkt.add(sc.tag||sizeof(sc));
      end
      /* combine two flits */
      pkt.add(sc);
    end
  end
end
```

Algorithm 1: Stitching with Selective Flit Pooling

$$Q_i = Sched(Pkt_{dst}, Pkt_{type}) = P_{dst,type}$$
$$P_{dst} = P[dst]$$
$$P_{dst,type} = P_{dst}[Flit_{size} - (Pkt_{size} \ \% \ Flit_{size})]$$
$$where,$$
$$Q = Queue \ \& \ P = Partition \ \& \ Pkt = Network \ Packet$$
$$P : dst \rightarrow Q_f \ \& \ Q_f : size \rightarrow Q$$