Harnessing GPU Computing in System Level Software

Weibin Sun

Software stack needs
fast system level base!

Software stack needs
fast system level base!

System functionality
can be expensive!

Software stack needs fast system level base!

System functionality can be expensive!

- **Crypto**: AES, SHA1, MD5, RSA
- **Lookup**: IP routing, DNS, key-value store, DB
- **Erasure coding**: RAID, distributed storage, wireless
- **Pattern matching**: NIDS, virus, searching
- **Compression**: storage, network
- …

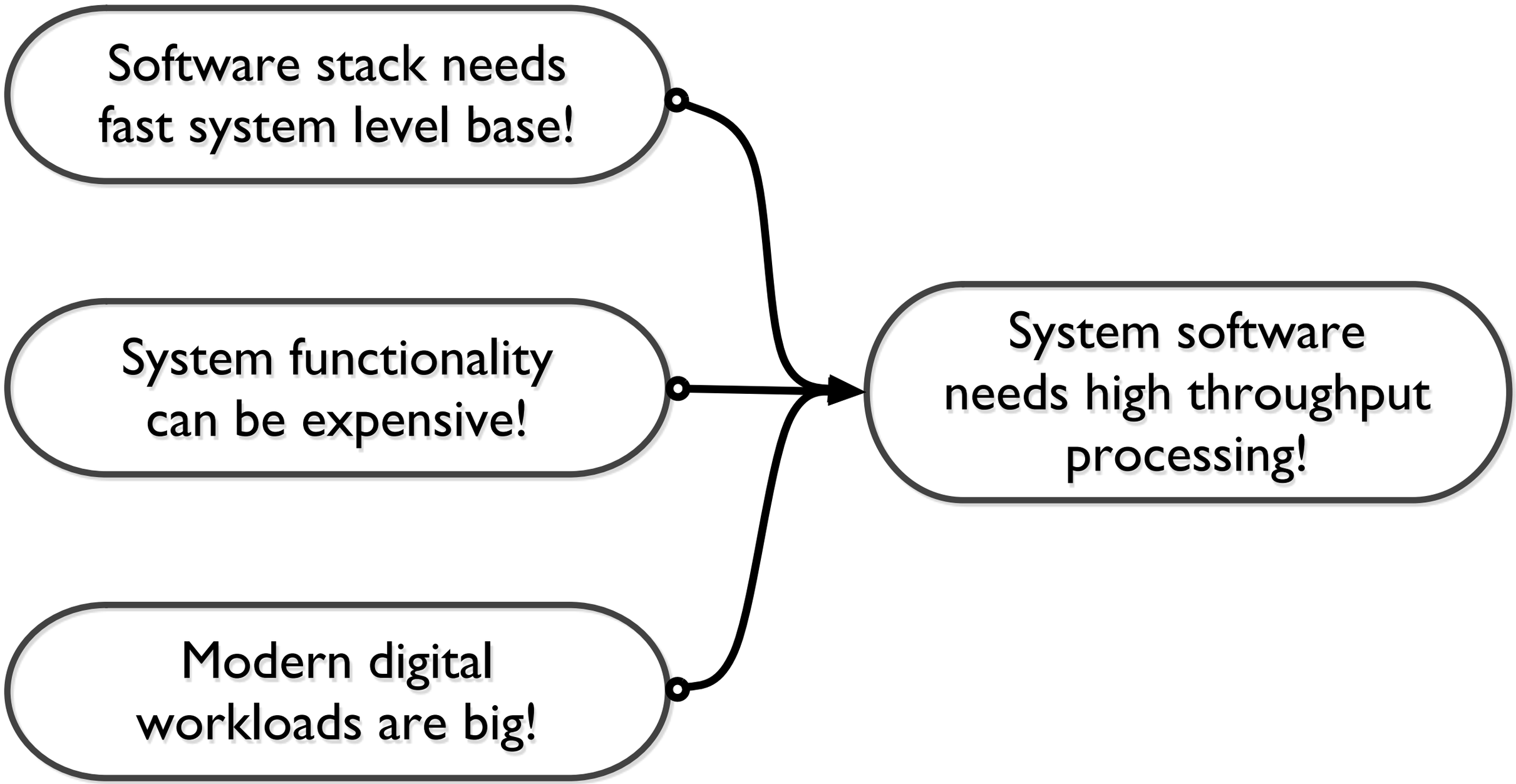
Software stack needs
fast system level base!

System functionality
can be expensive!

Software stack needs
fast system level base!

System functionality
can be expensive!

Modern digital
workloads are big!

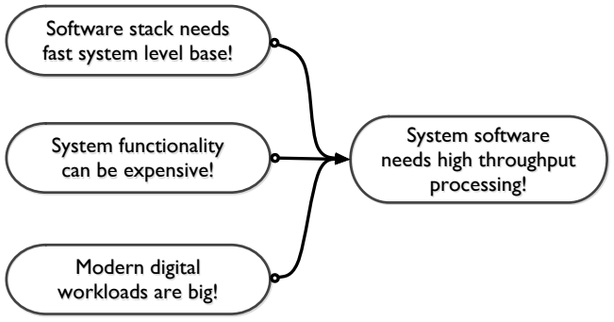

Software stack needs fast system level base!
System functionality can be expensive!
Modern digital workloads are big!
System software needs high throughput processing!

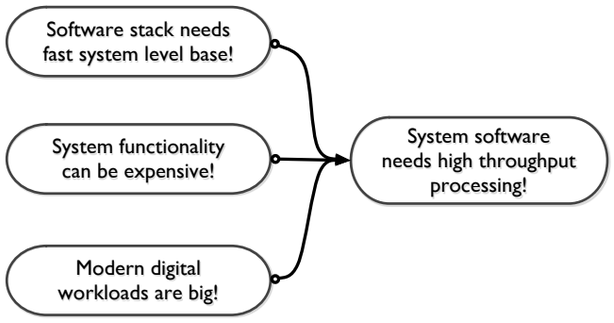
Software stack needs fast system level base!

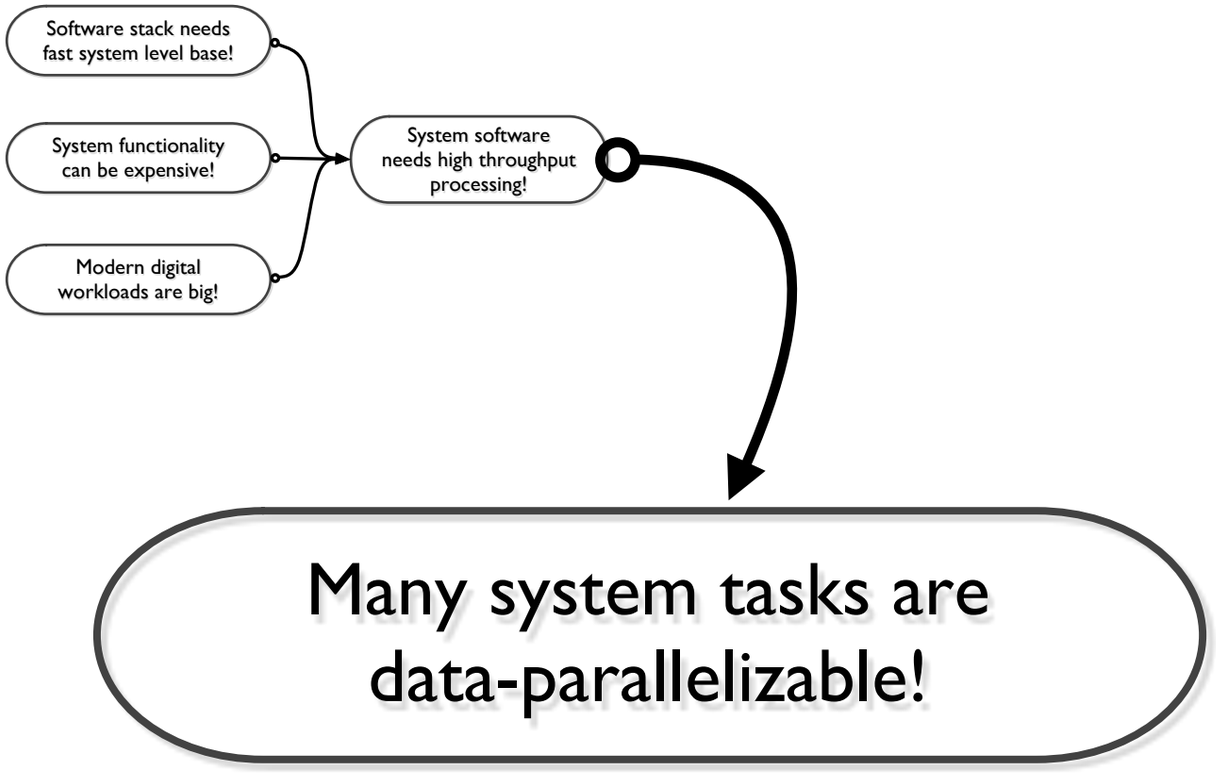
System functionality can be expensive!

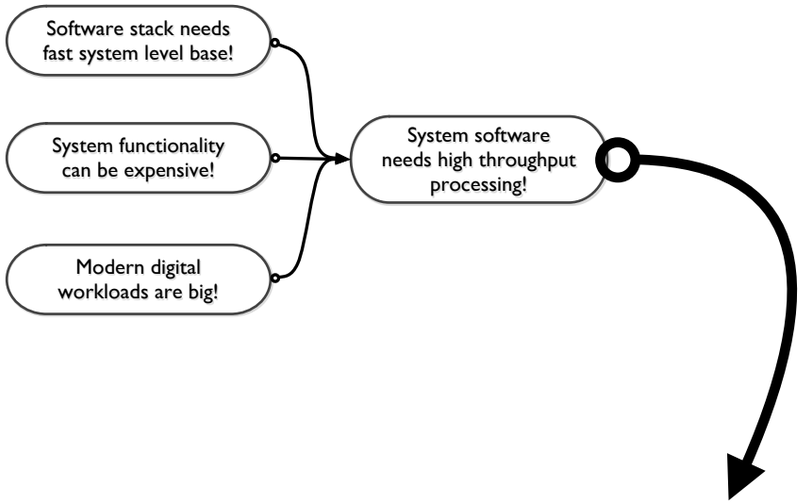
Modern digital workloads are big!

System software needs high throughput processing!


Software stack needs fast system level base!
System functionality can be expensive!
Modern digital workloads are big!
System software needs high throughput processing!

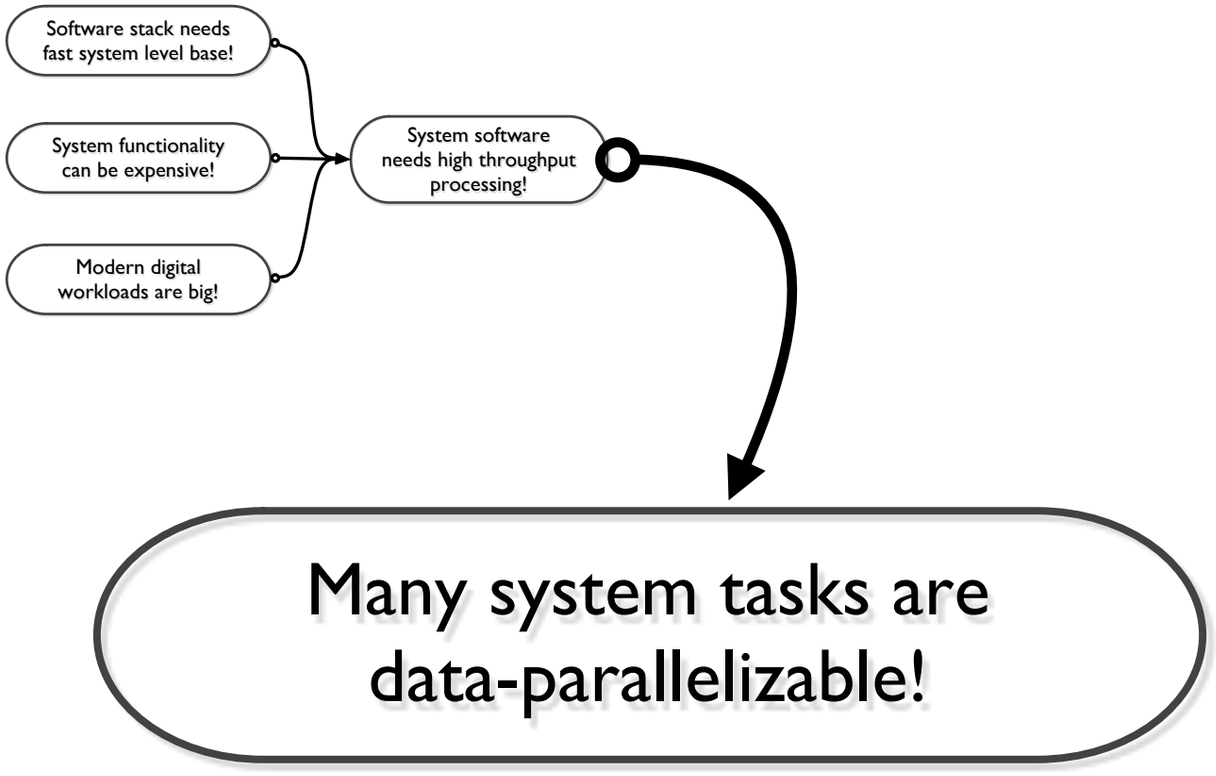

Software stack needs fast system level base!
System functionality can be expensive!
Modern digital workloads are big!
System software needs high throughput processing!

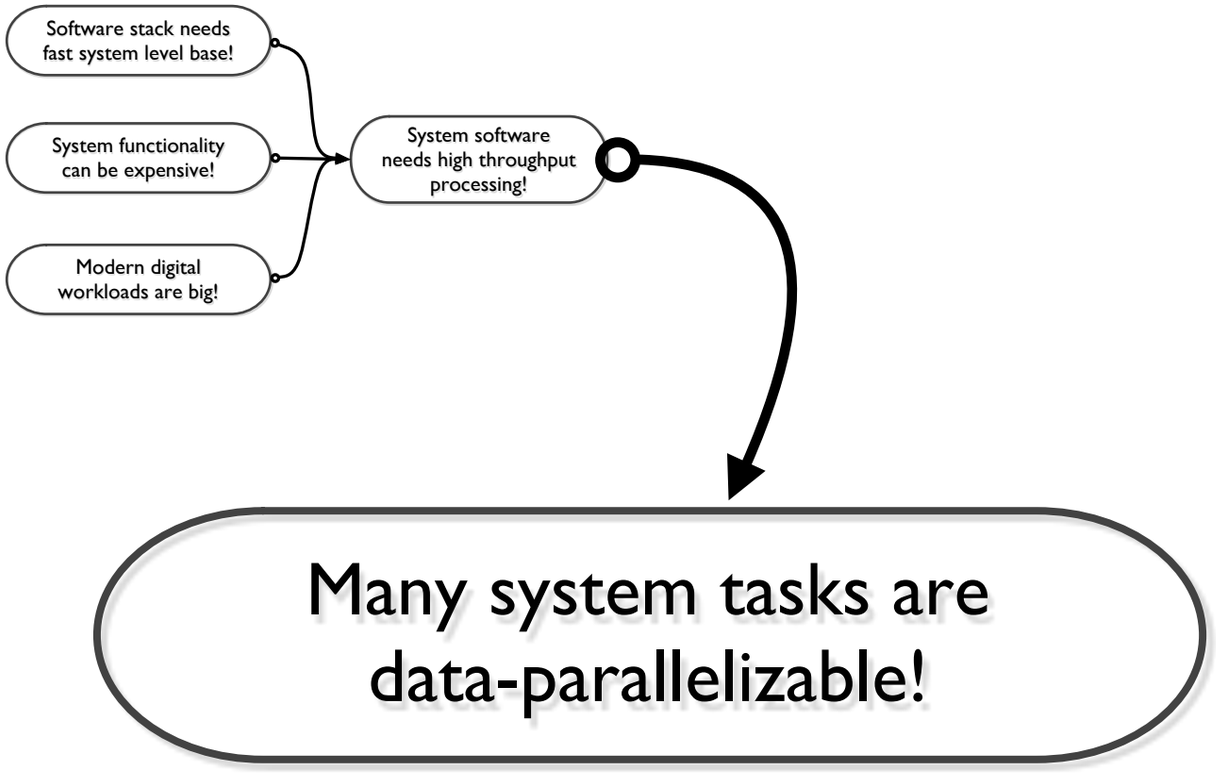

Software stack needs fast system level base!
System functionality can be expensive!
Modern digital workloads are big!
System software needs high throughput processing!
Many system tasks are data-parallelizable!



Many system tasks are data-parallelizable!

packets, sectors, data blocks, …


Software stack needs fast system level base!
System functionality can be expensive!
Modern digital workloads are big!
System software needs high throughput processing!
Many system tasks are data-parallelizable!



GPUs are the de-facto standard parallel processors!

Software stack needs fast system level base!

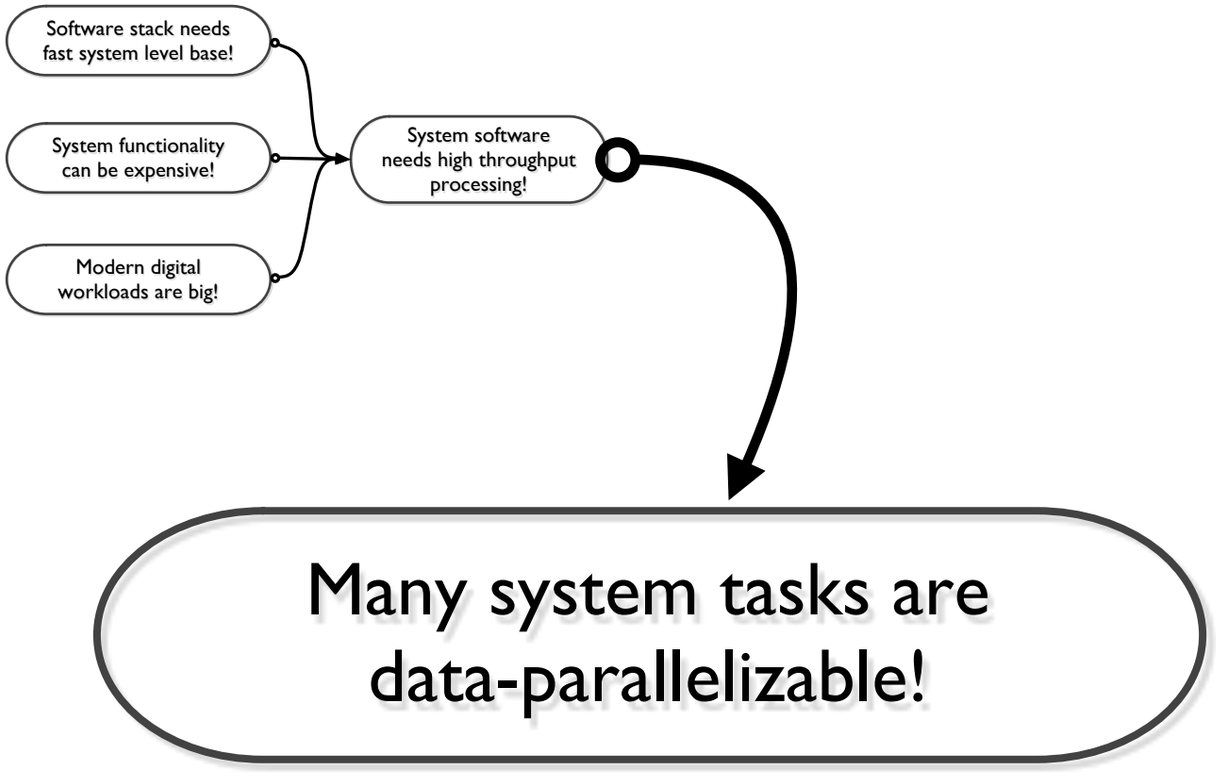
System functionality can be expensive!

Modern digital workloads are big!

System software needs high throughput processing!

Many system tasks are data-parallelizable!

GPUs are the de-facto standard parallel processors!



GPUs are the de-facto standard parallel processors!

Software stack needs fast system level base!

System functionality can be expensive!

Modern digital workloads are big!

System software needs high throughput processing!

Many system tasks are data-parallelizable!

GPUs are the de-facto standard parallel processors!

Using GPUs in system software!

Thesis statement

The throughput of system software with parallelizable, computationally expensive tasks can be improved by using GPUs and frameworks with memory efficient and throughput oriented designs.

Thesis statement

The throughput of system software with parallelizable, computationally expensive tasks can be improved by using GPUs and frameworks with memory efficient and throughput oriented designs.

Thesis statement

The throughput of system software with parallelizable, computationally expensive tasks can be improved by using GPUs and frameworks with memory efficient and throughput oriented designs.

Thesis statement

The throughput of system software with parallelizable, computationally expensive tasks can be improved by using GPUs and frameworks with memory efficient and throughput oriented designs.

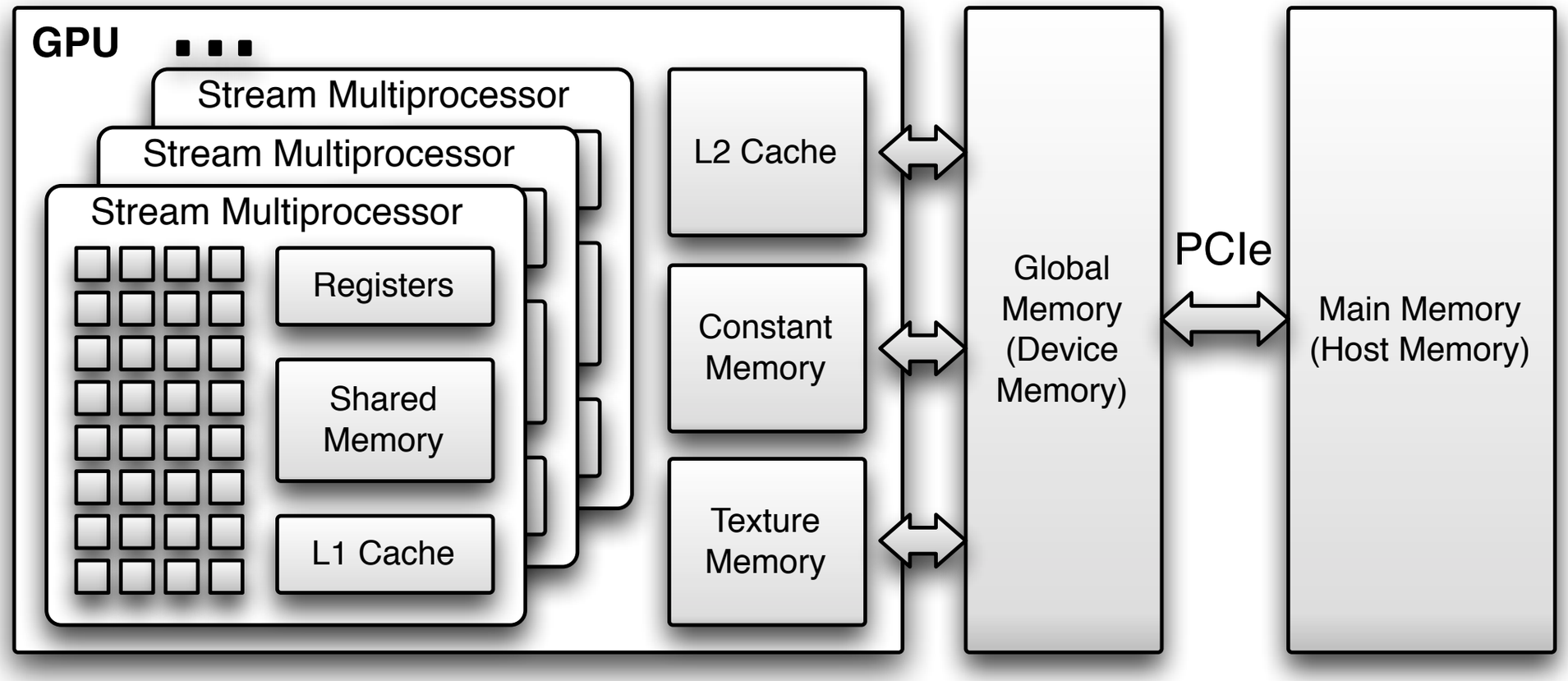
Thesis statement

The throughput of system software with parallelizable, computationally expensive tasks can be improved by using GPUs and frameworks with memory efficient and throughput oriented designs.

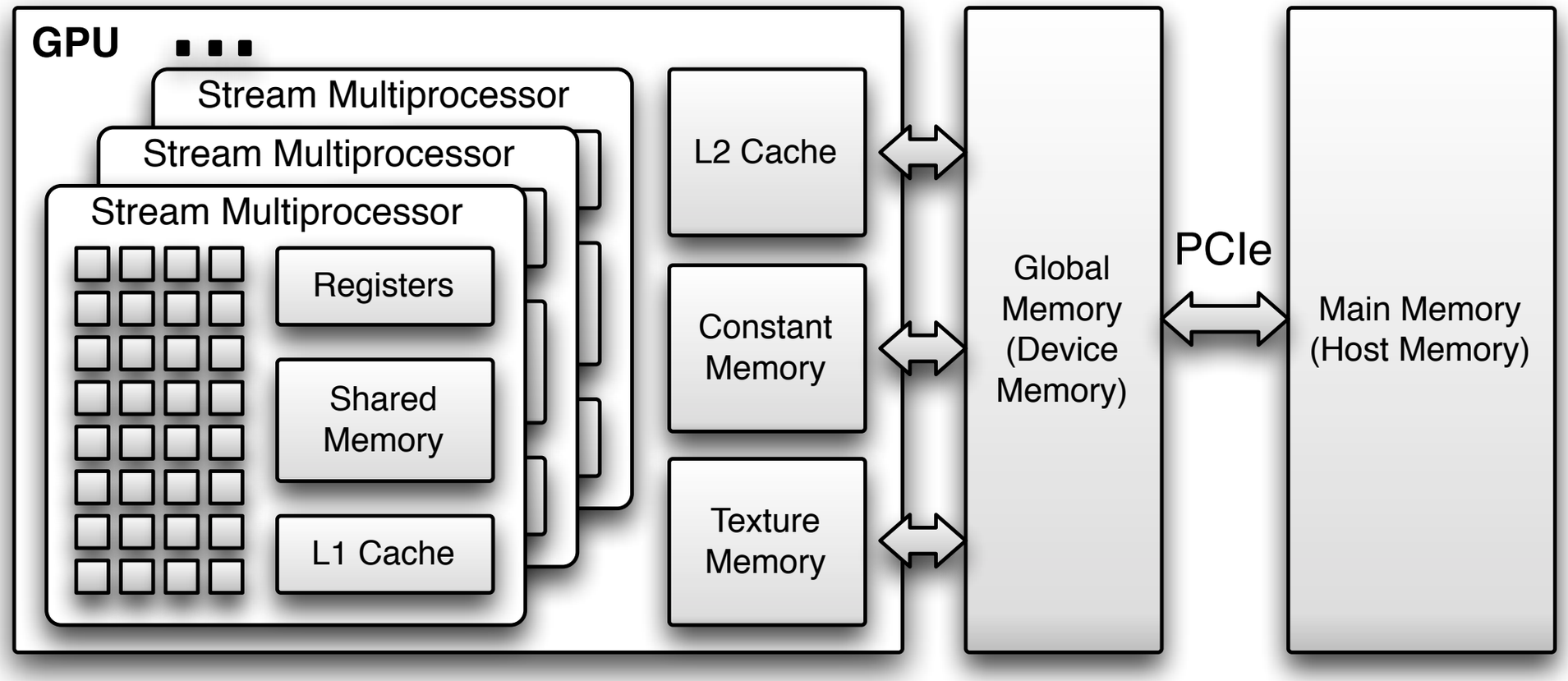
Thesis work

- Generic principles
- Concretely:
  - Two generic frameworks for representative system tasks: storage and network
  - Example applications on top of both frameworks
- Literature survey *

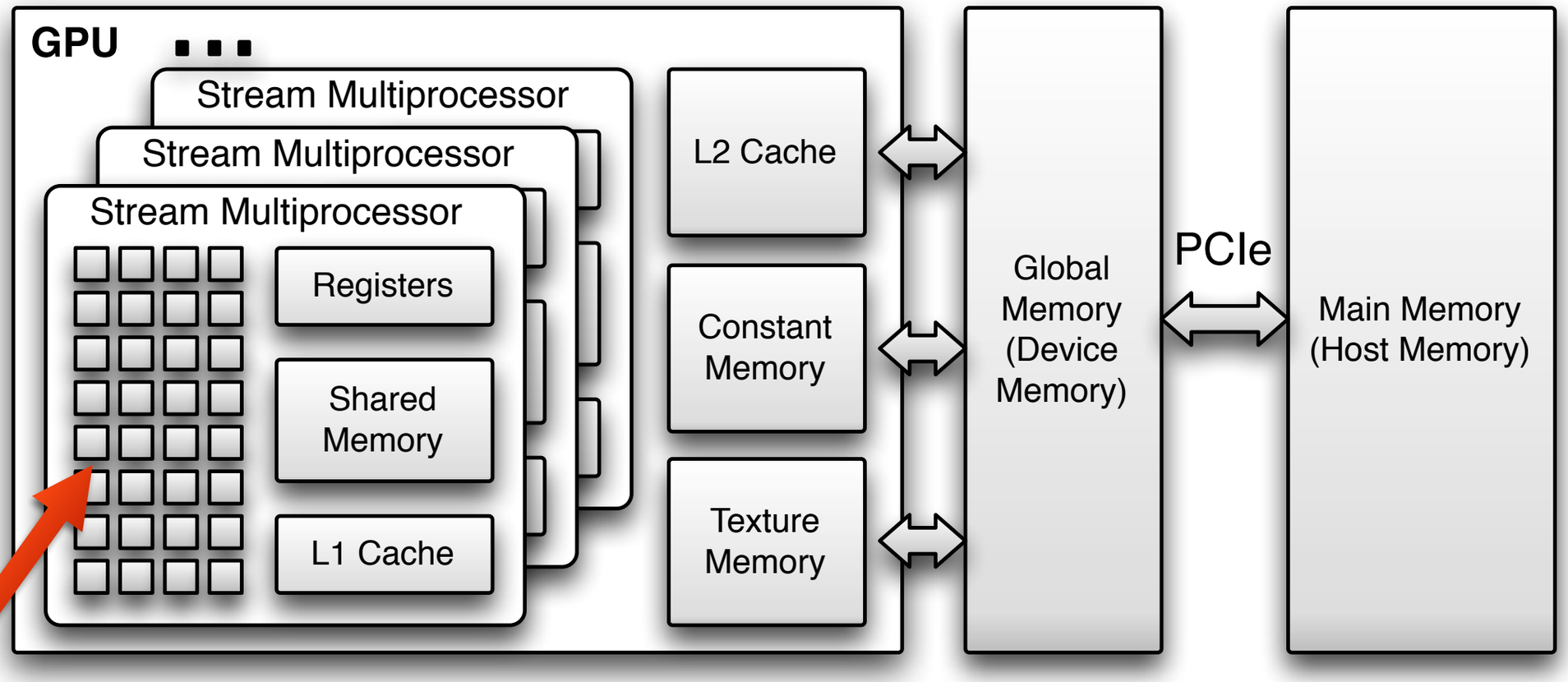
GPU essentials



CUDA GPU essentials

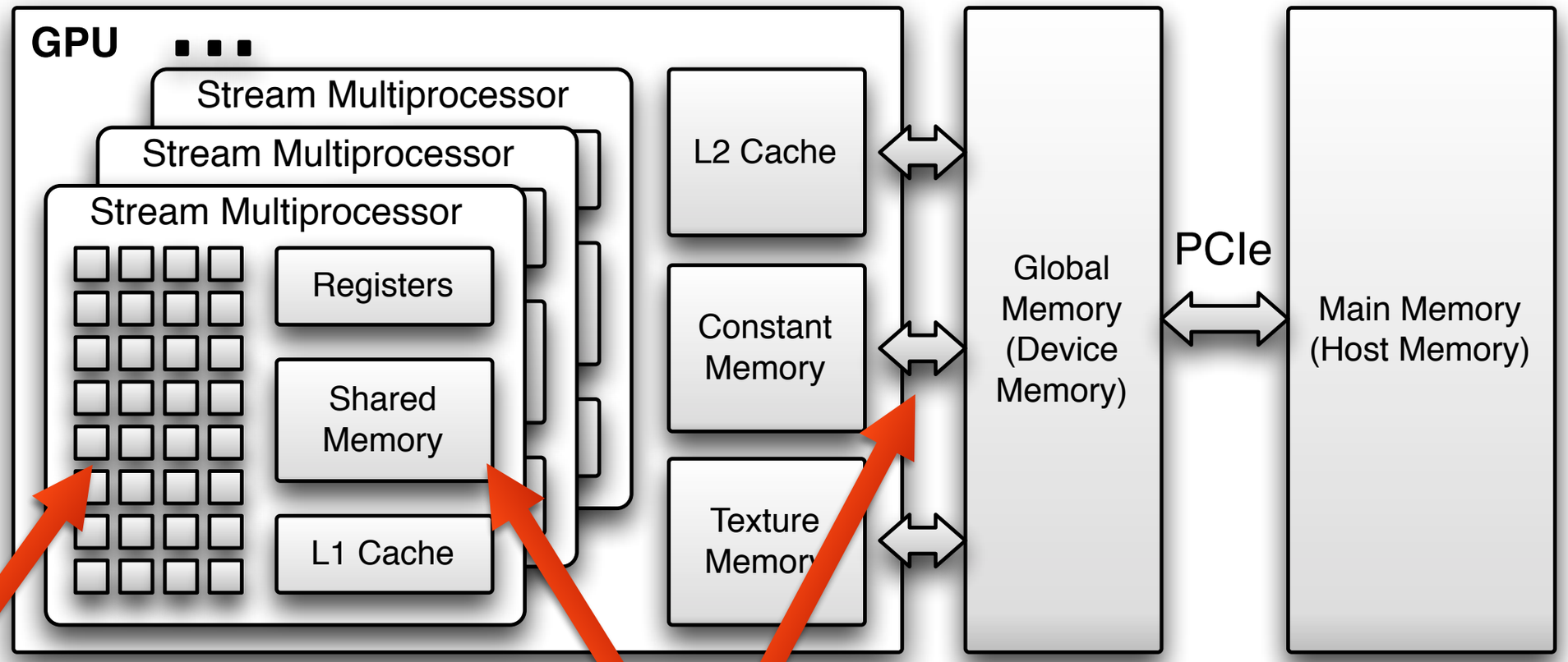


CUDA GPU essentials



SIMT wimpy cores:
32-core warp

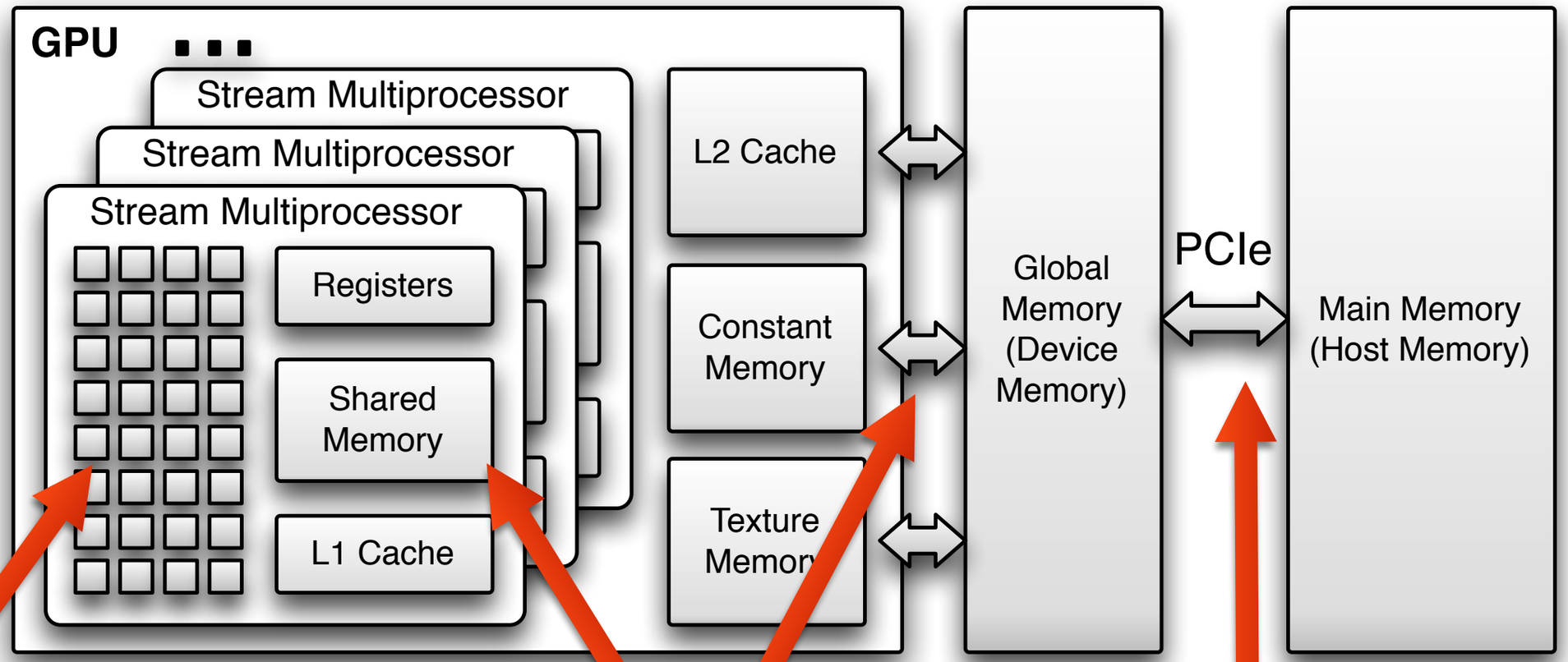
CUDA GPU essentials



SIMT wimpy cores:
32-core warp

wide memory interface: 384bits!

CUDA GPU essentials



SIMT wimpy cores: 32-core warp

wide memory interface: 384bits!

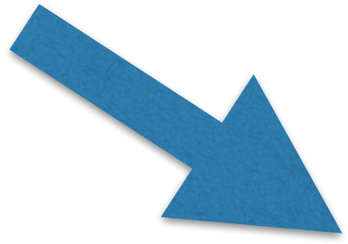
necessary memory copy

Related work: pioneers

- Gnort [RAID '2008]
- PacketShader [SIGCOMM '2010]
- SSLShader [NSDI '2011]
- EigenCFA [POPL '2011]
- Gibraltar [ICPP '2010]
- CrystalGPU [HPDC '2010]
- …

Related work: pioneers

- Gnort [RAID '2008]
- PacketShader [SIGCOMM '2010]
- SSLShader [NSDI '2011]
- EigenCFA [POPL '2011]
- Gibraltar [ICPP '2010]
- CrystalGPU [HPDC '2010]
- …



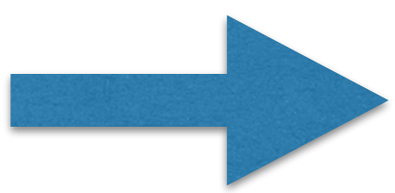
Specialized and Ad-hoc solutions to one particular application!

GPU computing frameworks/models

- Hydra [ASPLOS'08]
- CUDA-Lite [LCPC'08]
- hiCUDA [GPGPU'09]
- ASDSM [ASPLOS'10]
- Sponge [ASPLOS'11]
- CGCM [PLDI'11]
- PTask [SOSP'11]
- …

GPU computing frameworks/models

- Hydra [ASPLOS'08]
- CUDA-Lite [LCPC'08]
- hiCUDA [GPGPU'09]
- ASDSM [ASPLOS'10]
- Sponge [ASPLOS'11]
- CGCM [PLDI'11]
- PTask [SOSP'11]
- …



- Simple workflow
- Long computation, PCIe copy and synchronization not matter
- No batched data divergence

GPU computing frameworks/models

- Hydra [ASPLOS'08]
- CUDA
- …
- …
- …
- P…
- …

System level GPU computing needs its own generic frameworks!

No batched data divergence

Rest of this talk

- Generic principles
- GPUstore : for storage systems
- Snap : for packet processing

System level GPU computing:

generic principles

Problems

- Latency-oriented system code V.S. throughput-oriented GPU
- CPU-GPU synchronization hurts asynchronous systems
- Wasted PCIe data transfer
- Double buffering for GPU DMA

Problems

- Latency-oriented system code V.S. throughput-oriented GPU
- CPU-GPU synchronization hurts asynchronous systems
- Wasted PCIe data transfer
- Double buffering for GPU DMA

Latency V.S. throughput

Latency V.S. throughput

Batched processing: wait for enough workload

Latency V.S. throughput

Batched processing: wait for enough workload

- how much is enough?

Latency V.S. throughput

Batched processing: wait for enough workload

- how much is enough?
- increased latency

Problems

- Latency-oriented system code V.S. throughput-oriented GPU
- CPU-GPU synchronization hurts asynchronous systems
- Wasted PCIe data transfer
- Double buffering for GPU DMA

Problems

- Batched processing
- CPU-GPU synchronization hurts asynchronous systems
- Wasted PCIe data transfer
- Double buffering for GPU DMA

Problems

- Batched processing
- CPU-GPU synchronization hurts asynchronous systems
- Wasted PCIe data transfer
- Double buffering for GPU DMA

CPU-GPU synchronization

Blocking:

CPU: 

GPU:

CPU-GPU synchronization

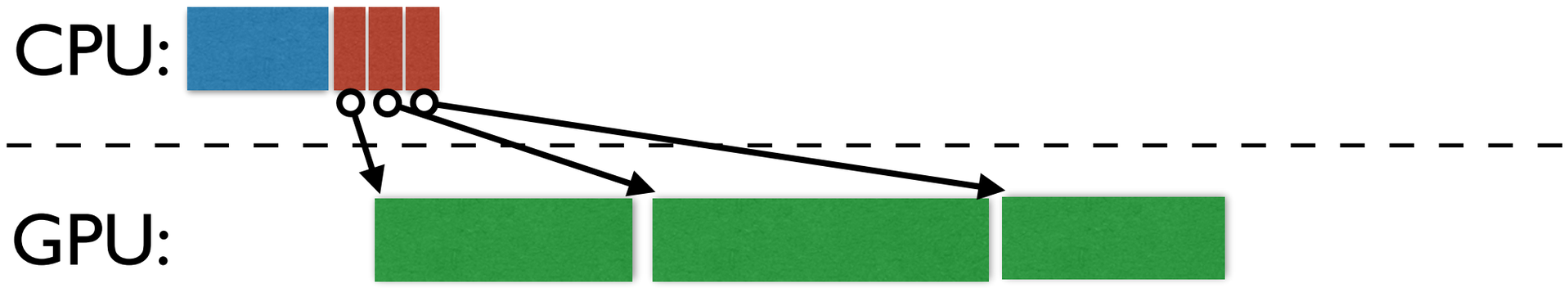
Blocking:

CPU: 

GPU:

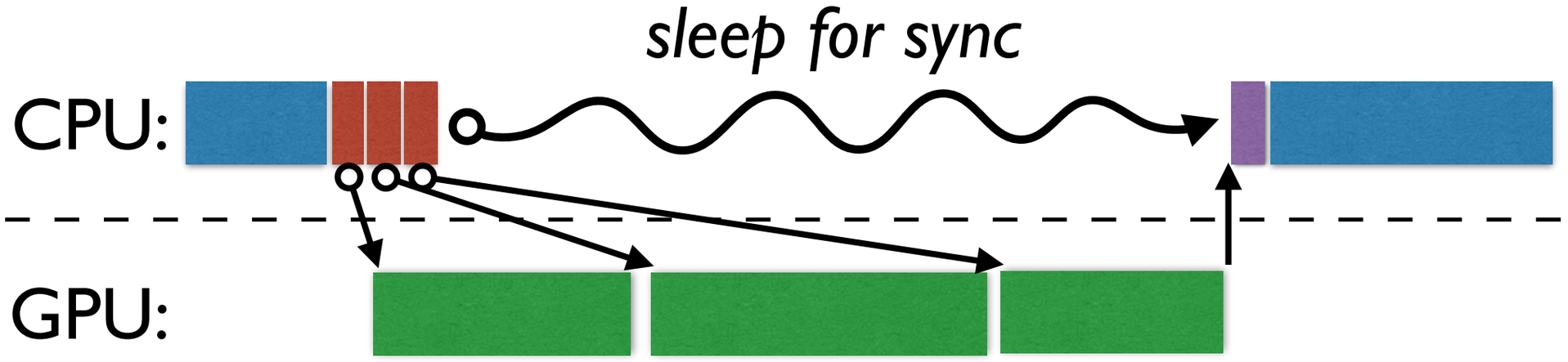
CPU-GPU synchronization

Blocking:



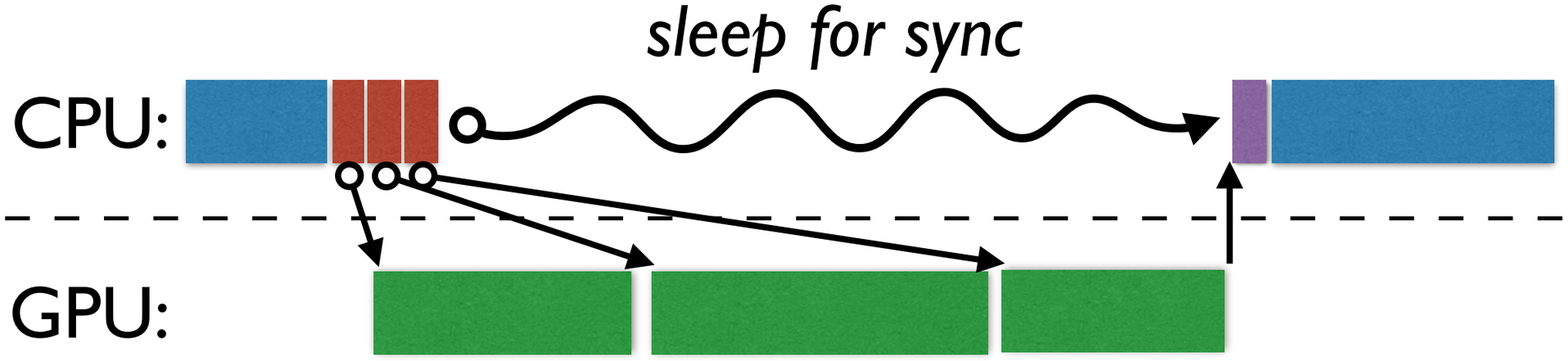
CPU-GPU synchronization

Blocking:

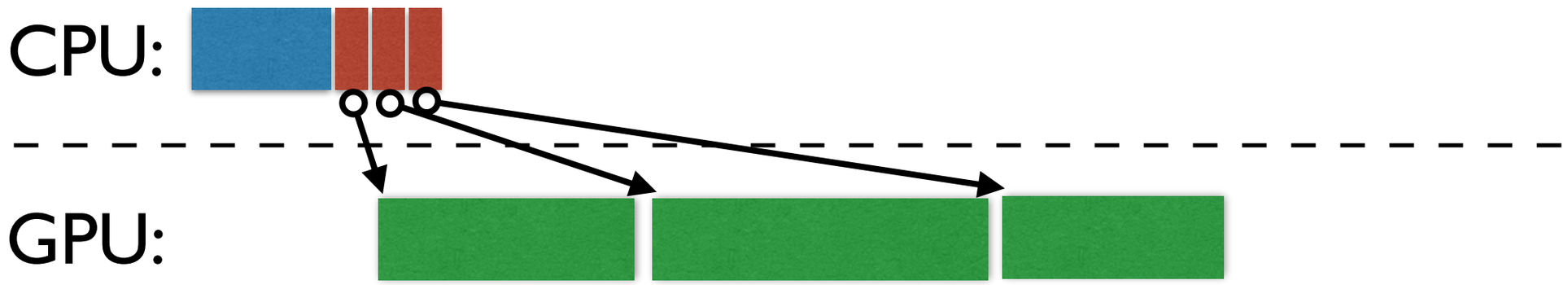


CPU-GPU synchronization

Blocking:

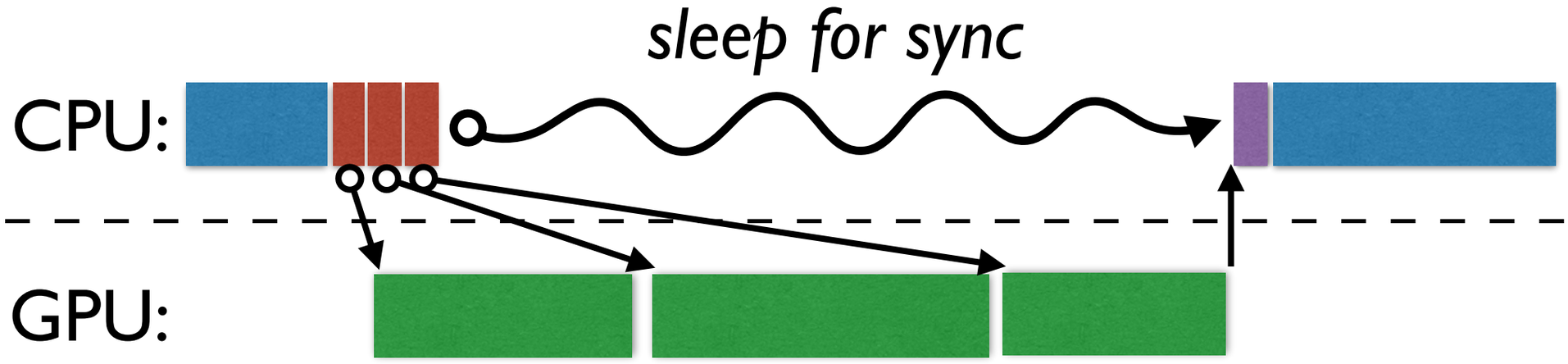


Non-blocking:

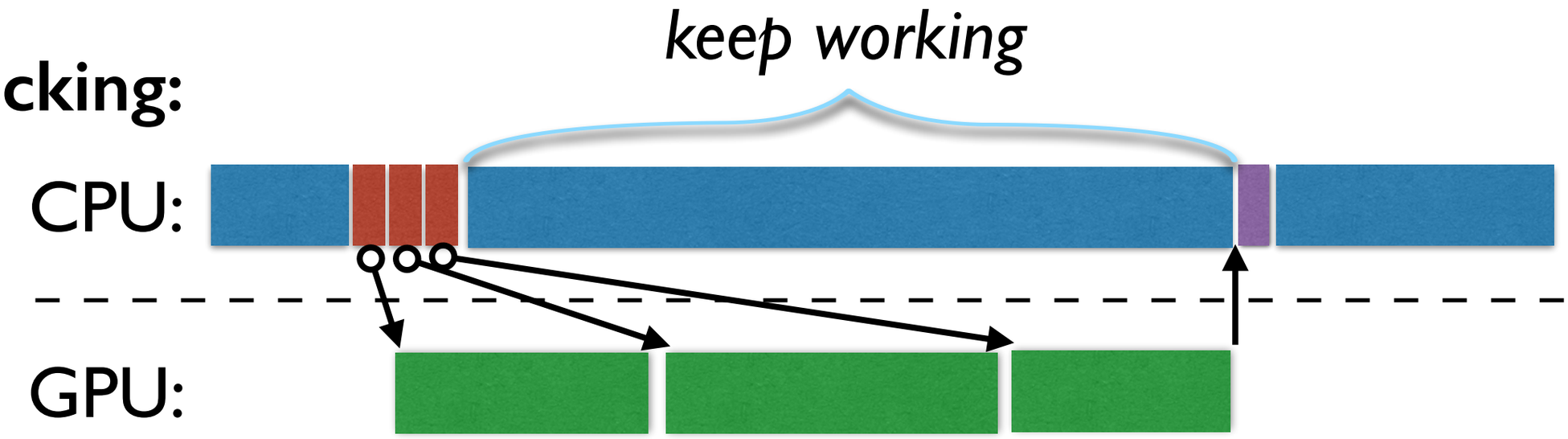


CPU-GPU synchronization

Blocking:



Non-blocking:



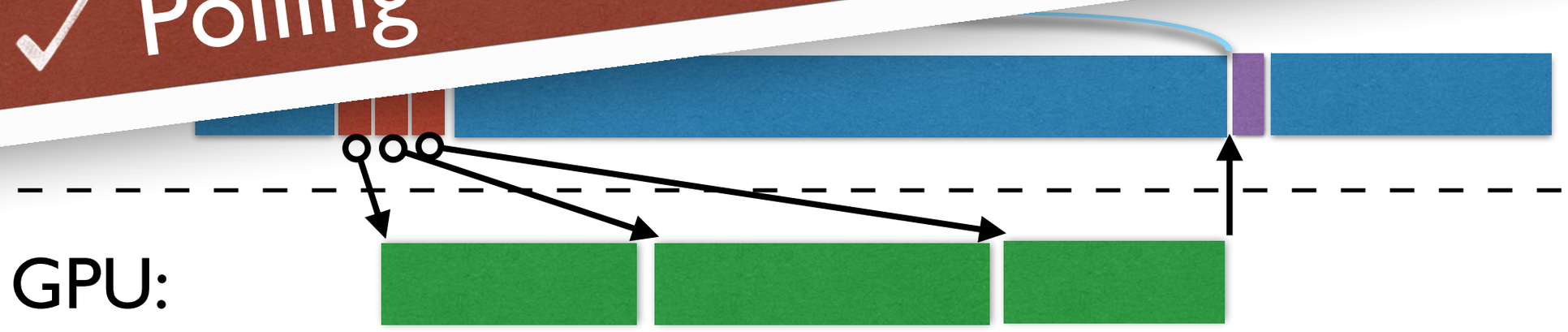
CPU-GPU synchronization

Blocking:

System code (network, disk, etc.) likes:

- ✓ Callback
- ✓ Polling

Non



Problems

- Batched processing
- CPU-GPU synchronization hurts asynchronous systems
- Wasted PCIe data transfer
- Double buffering for GPU DMA

Problems

- Batched processing
- Asynchronous non-blocking GPU programming
- Wasted PCIe data transfer
- Double buffering for GPU DMA

Problems

- Batched processing
- Asynchronous non-blocking GPU programming
- Wasted PCIe data transfer
- Double buffering for GPU DMA

How PCIe transfer wasted?

How PCIe transfer wasted?

Do you need the entire packet to do routing lookup?

How PCIe transfer wasted?

Do you need the entire packet to do routing lookup?

What you need:

How PCIe transfer wasted?

Do you need the entire packet to do routing lookup?

What you need:

4 bytes

How PCIe transfer wasted?

Do you need the entire packet to do routing lookup?

What you need:

4 bytes

What you may transfer:

How PCIe transfer wasted?

Do you need the entire packet to do routing lookup?

What you need:

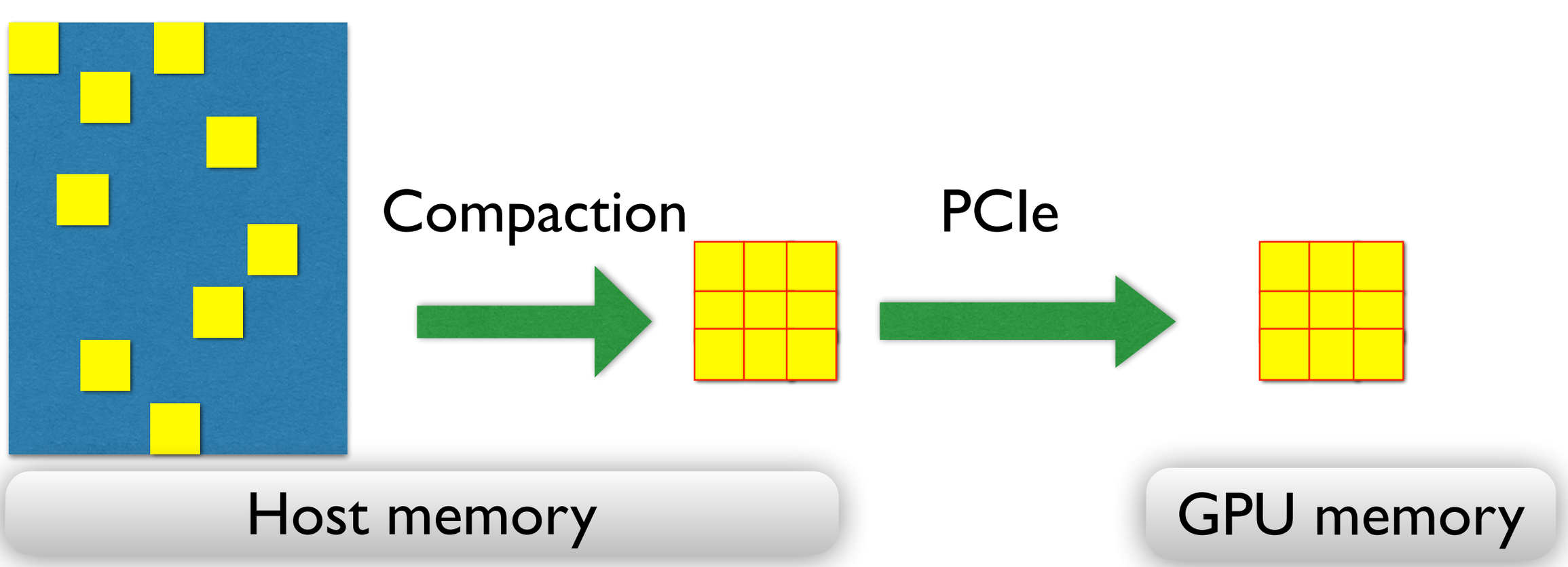
4 bytes

What you may transfer:

≥ 64 bytes

Compact your workload data

For ``use-partial'' data usage pattern



Problems

- Batched processing
- Asynchronous non-blocking GPU programming
- Wasted PCIe data transfer
- Double buffering for GPU DMA

Problems

- Batched processing
- Asynchronous non-blocking GPU programming
- Reduce PCIe data transfer with compacted workload
- Double buffering for GPU DMA

Problems

- Batched processing
- Asynchronous non-blocking GPU programming
- Reduce PCIe data transfer with compacted workload
- Double buffering for GPU DMA

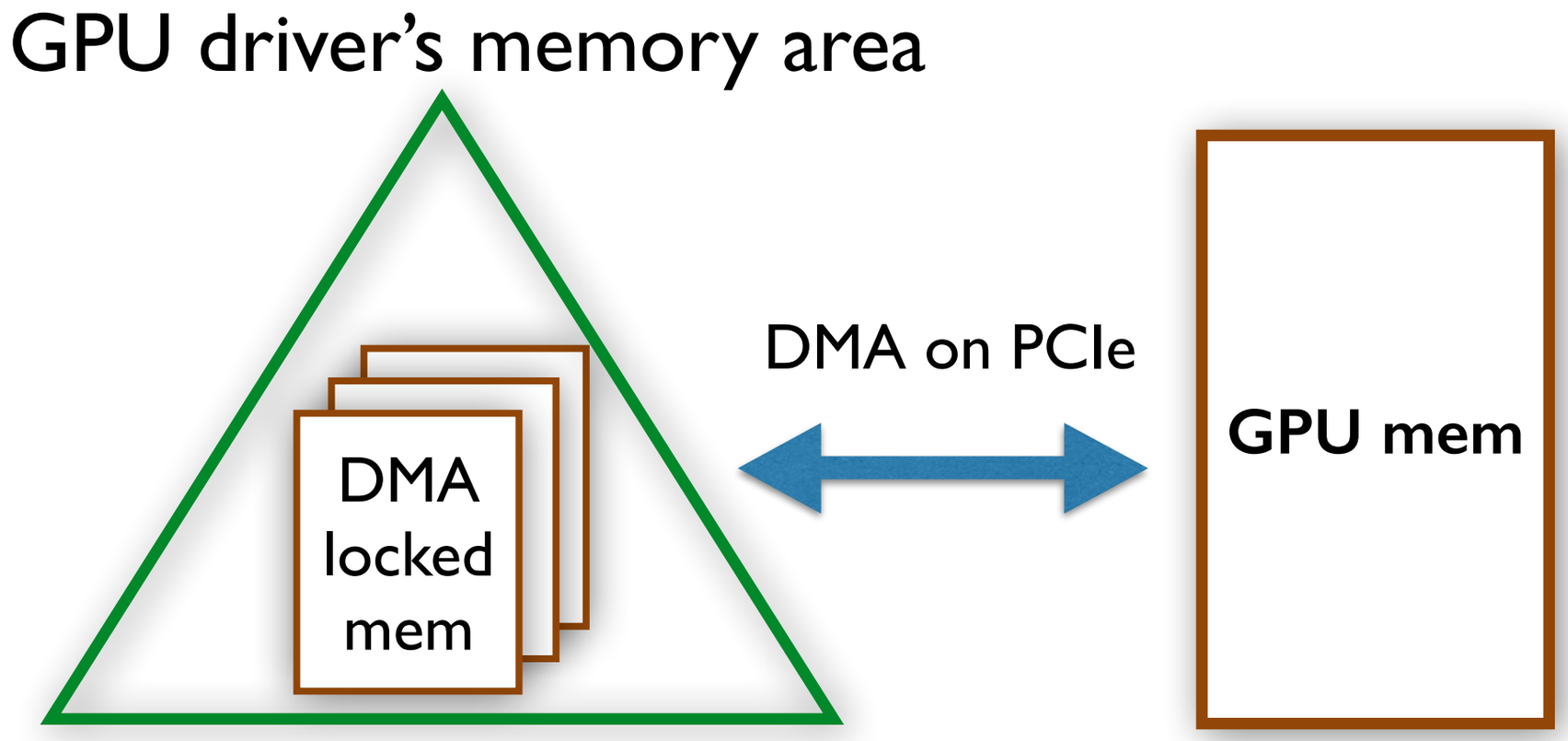
GPU DMA contraints

GPU DMA contraints

- DMA only on locked memory pages

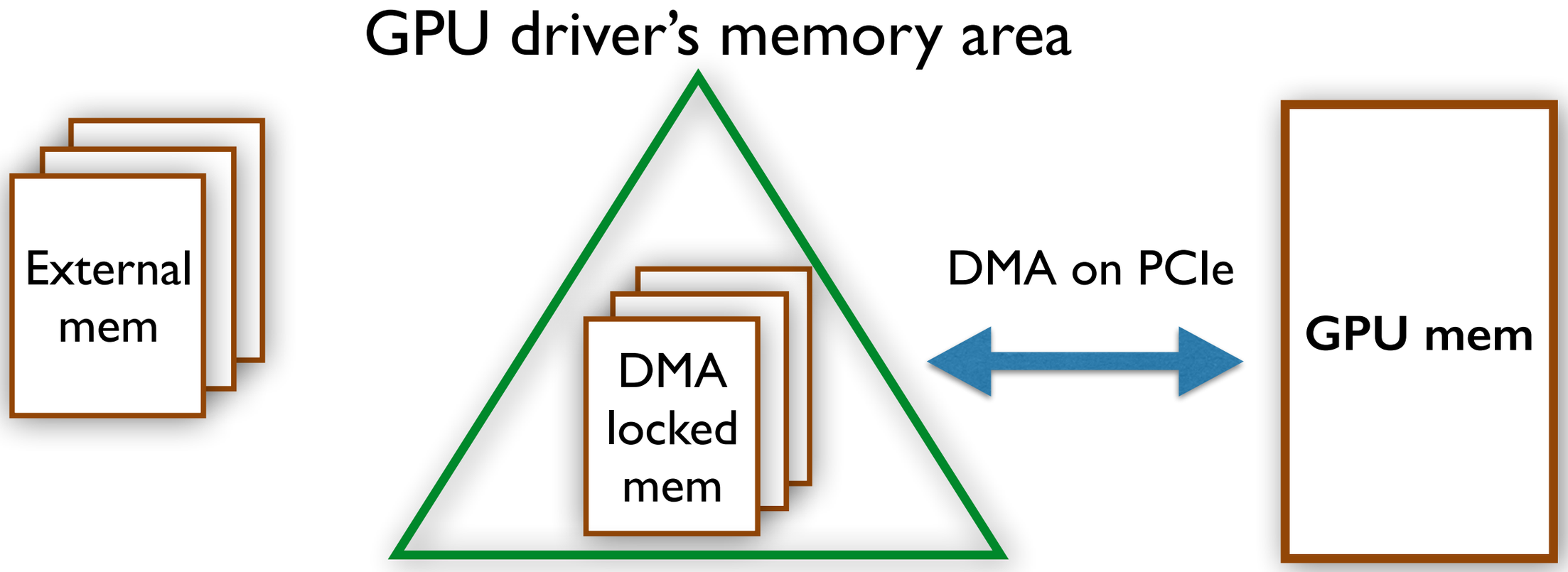
GPU DMA contraints

- DMA only on locked memory pages
- GPU drivers only recognize their own locked memory pool



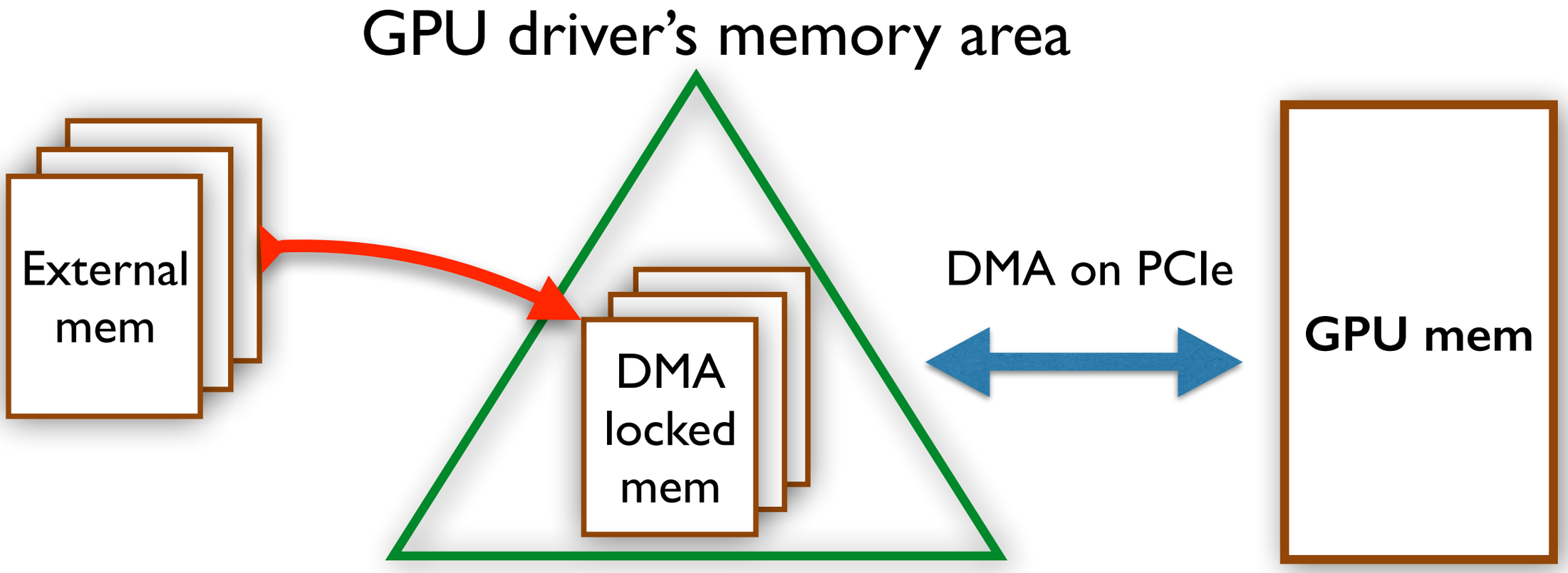
GPU DMA contraints

- DMA only on locked memory pages
- GPU drivers only recognize their own locked memory pool



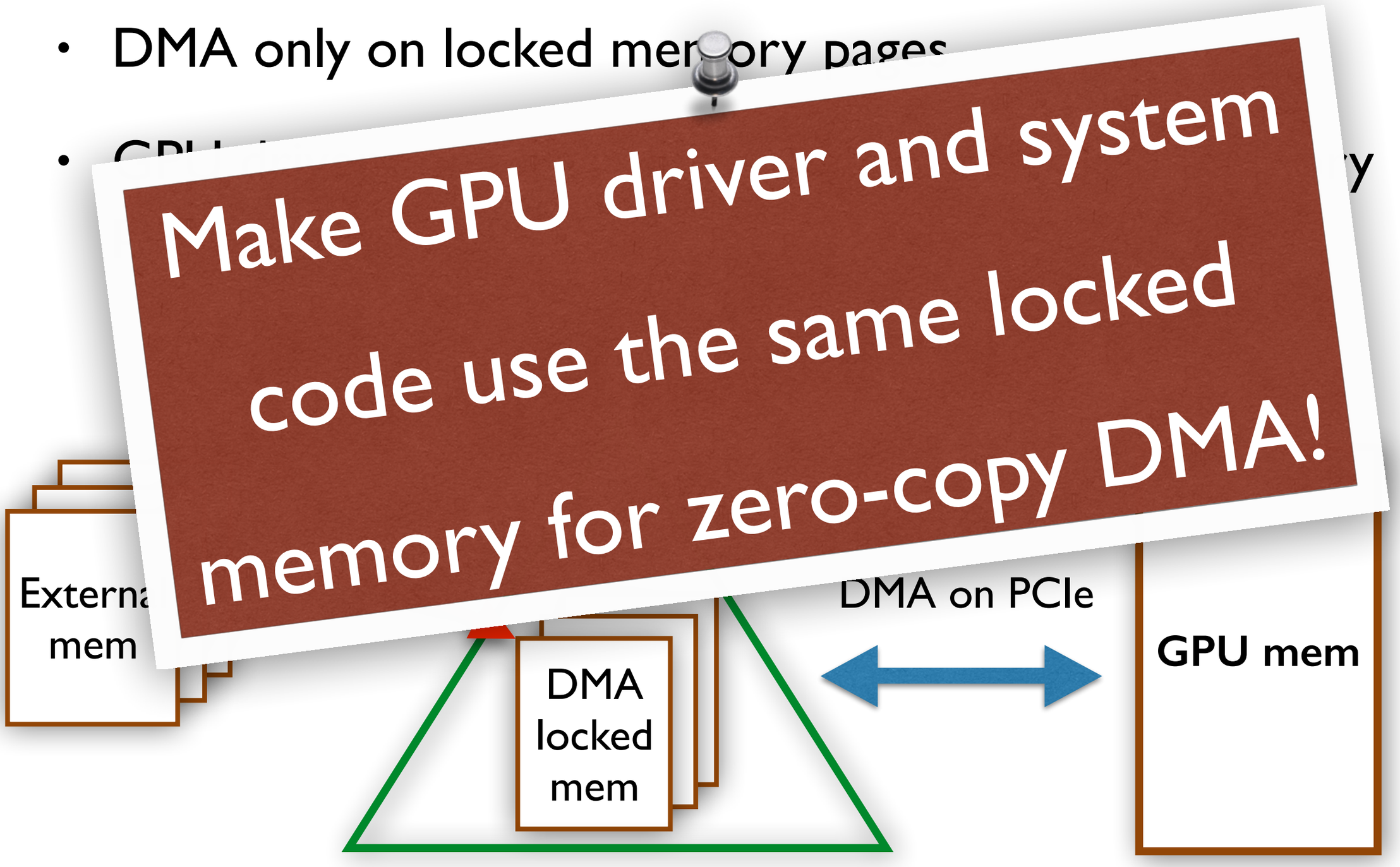
GPU DMA contraints

- DMA only on locked memory pages
- GPU drivers only recognize their own locked memory pool



GPU DMA contraints

- DMA only on locked memory pages
- GPU kernel code use the same locked memory for zero-copy DMA!



Problems

- Batched processing
- Asynchronous non-blocking GPU programming
- Reduce PCIe data transfer with compacted workload
- Double buffering for GPU DMA

Problems

- Batched processing
- Asynchronous non-blocking GPU programming
- Reduce PCIe data transfer with compacted workload
- All use the same locked memory for zero-copy DMA

Generic principles

- Batched processing
- Asynchronous non-blocking GPU programming
- Reduce PCIe data transfer with compacted workload
- All use the same locked memory for zero-copy DMA

GPUstore: Harnessing GPU Computing in OS Storage Systems

[Weibin Sun, Robert Ricci, Matthew L. Curry @ SYSTOR'12]

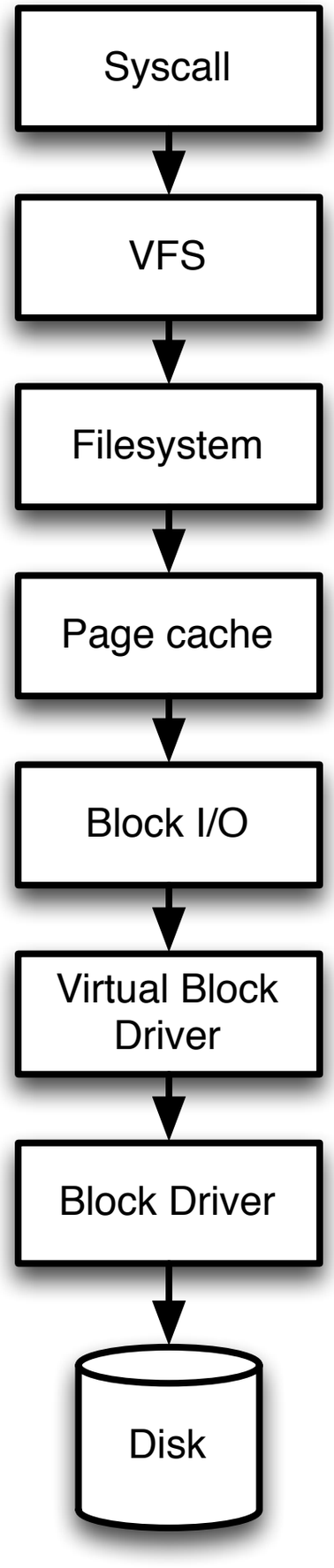
GPUstore overview

GPUstore overview

- A storage framework for Linux kernel to use CUDA GPUs in filesystems, block drivers, etc.

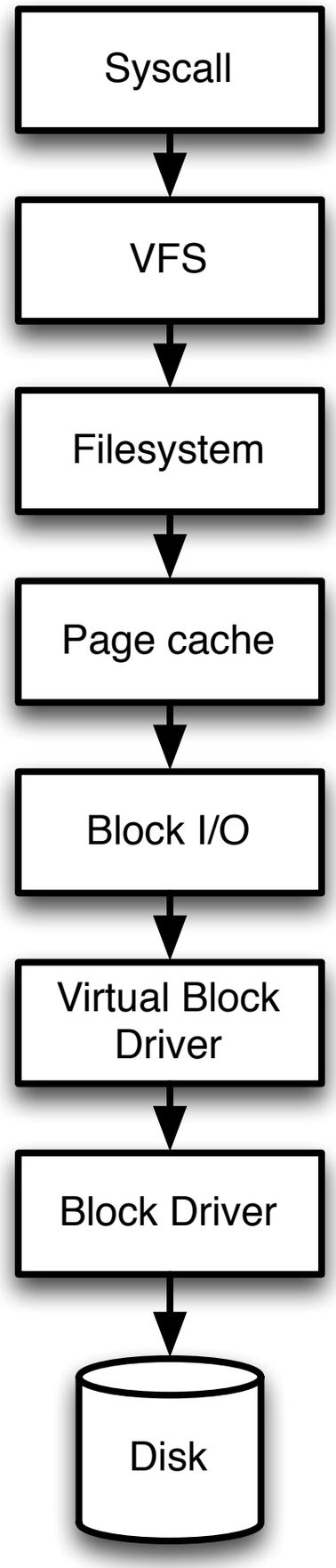
GPUstore overview

- A storage framework for Linux kernel to use CUDA GPUs in filesystems, block drivers, etc.



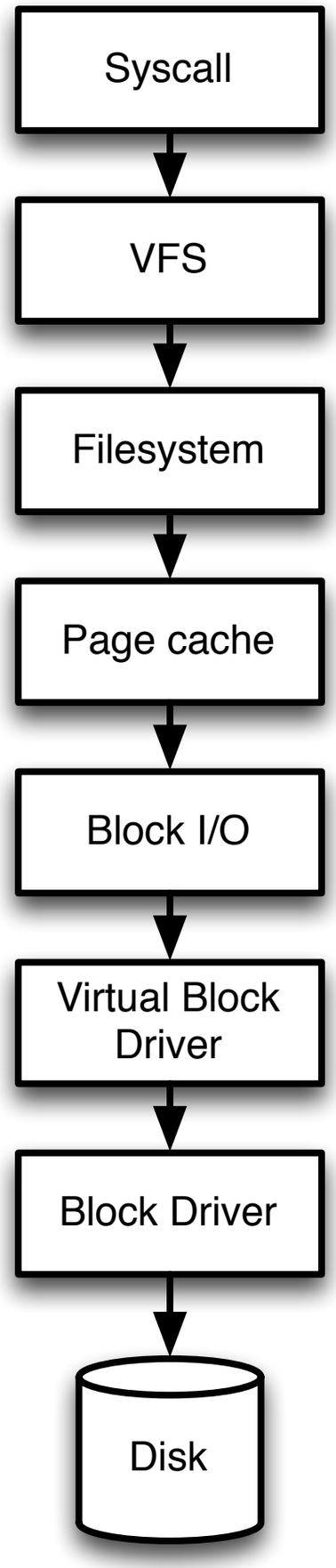
GPUstore overview

- A storage framework for Linux kernel to use CUDA GPUs in filesystems, block drivers, etc.
- Minimally invasive GPU integration



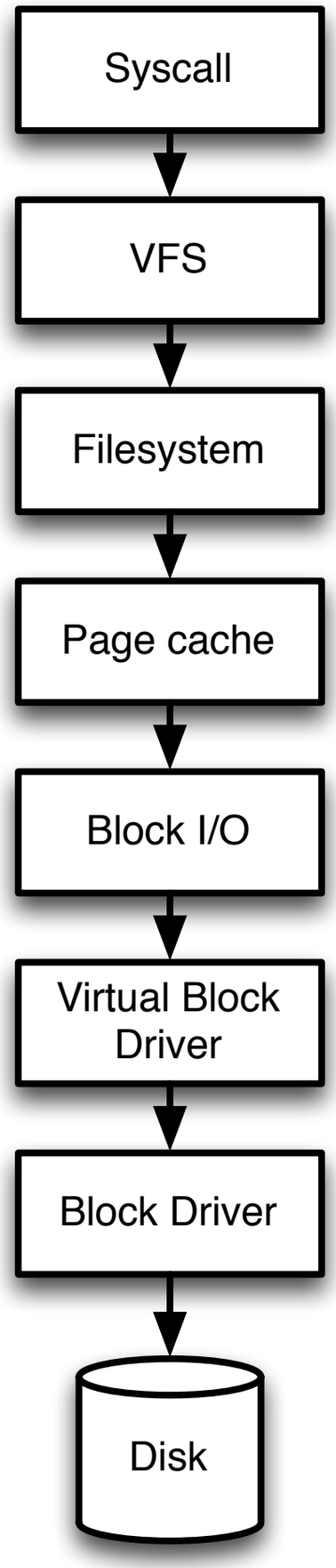
GPUstore overview

- A storage framework for Linux kernel to use CUDA GPUs in filesystems, block drivers, etc.
- Minimally invasive GPU integration
- Small changes



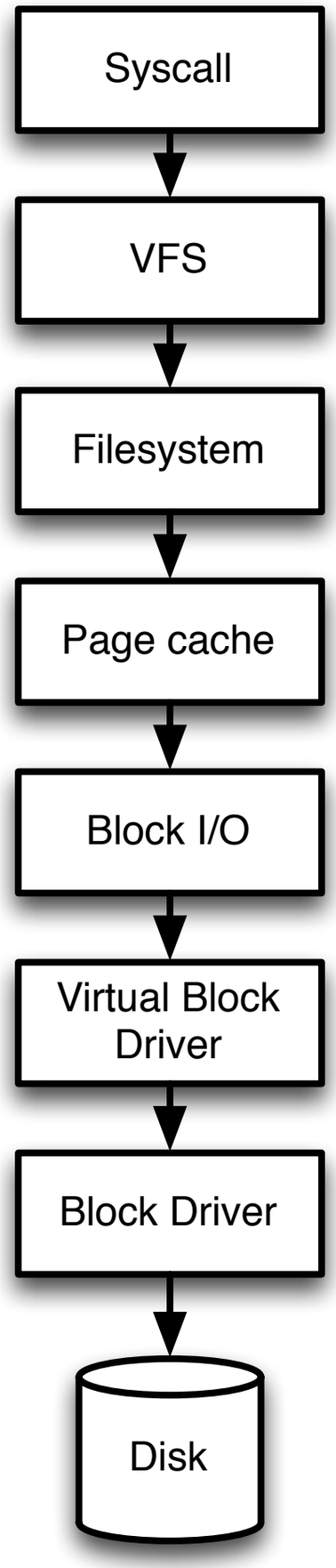
GPUstore overview

- A storage framework for Linux kernel to use CUDA GPUs in filesystems, block drivers, etc.
- Minimally invasive GPU integration
- Small changes
- Preserve interface and semantics

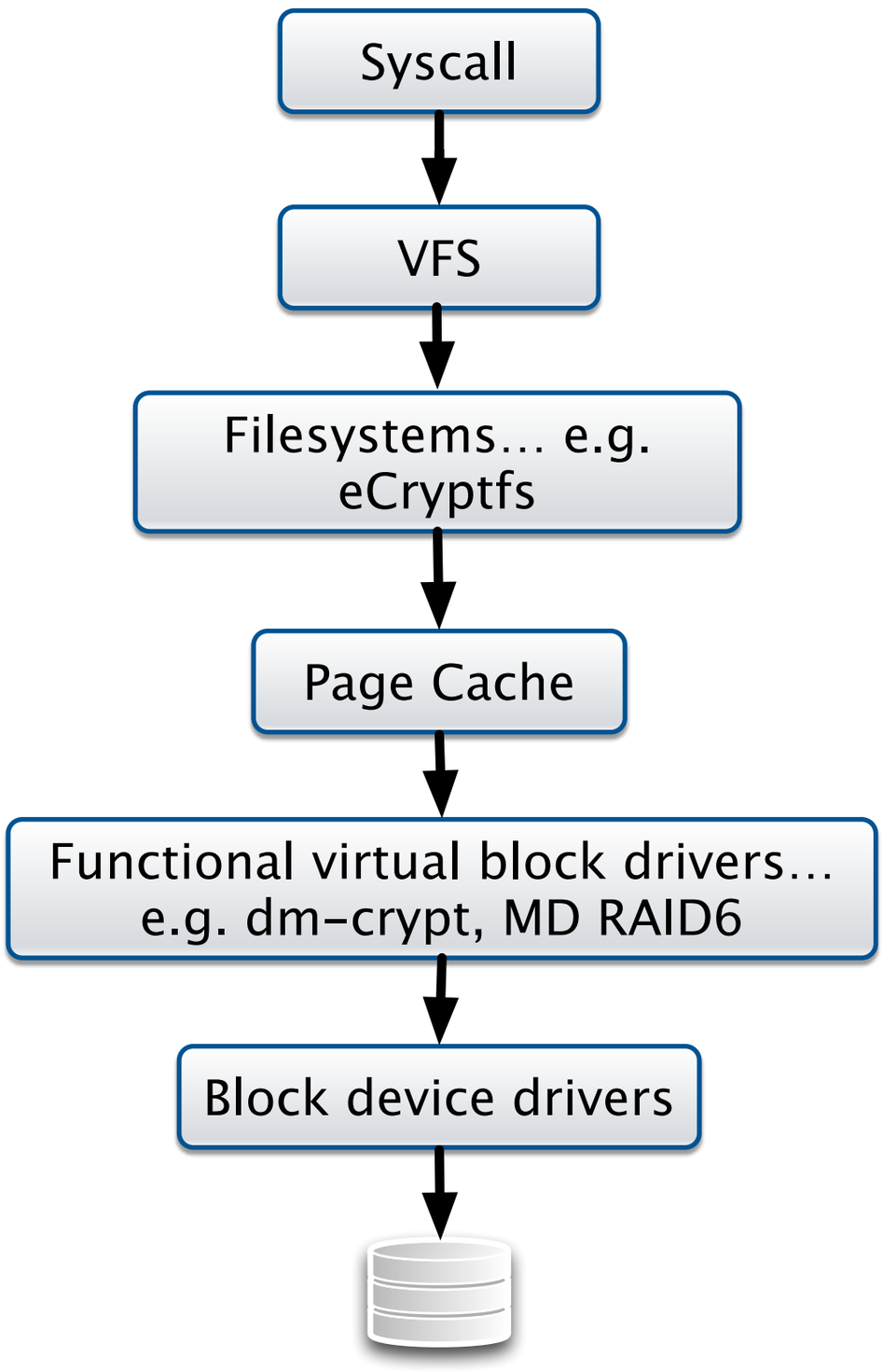


GPUstore overview

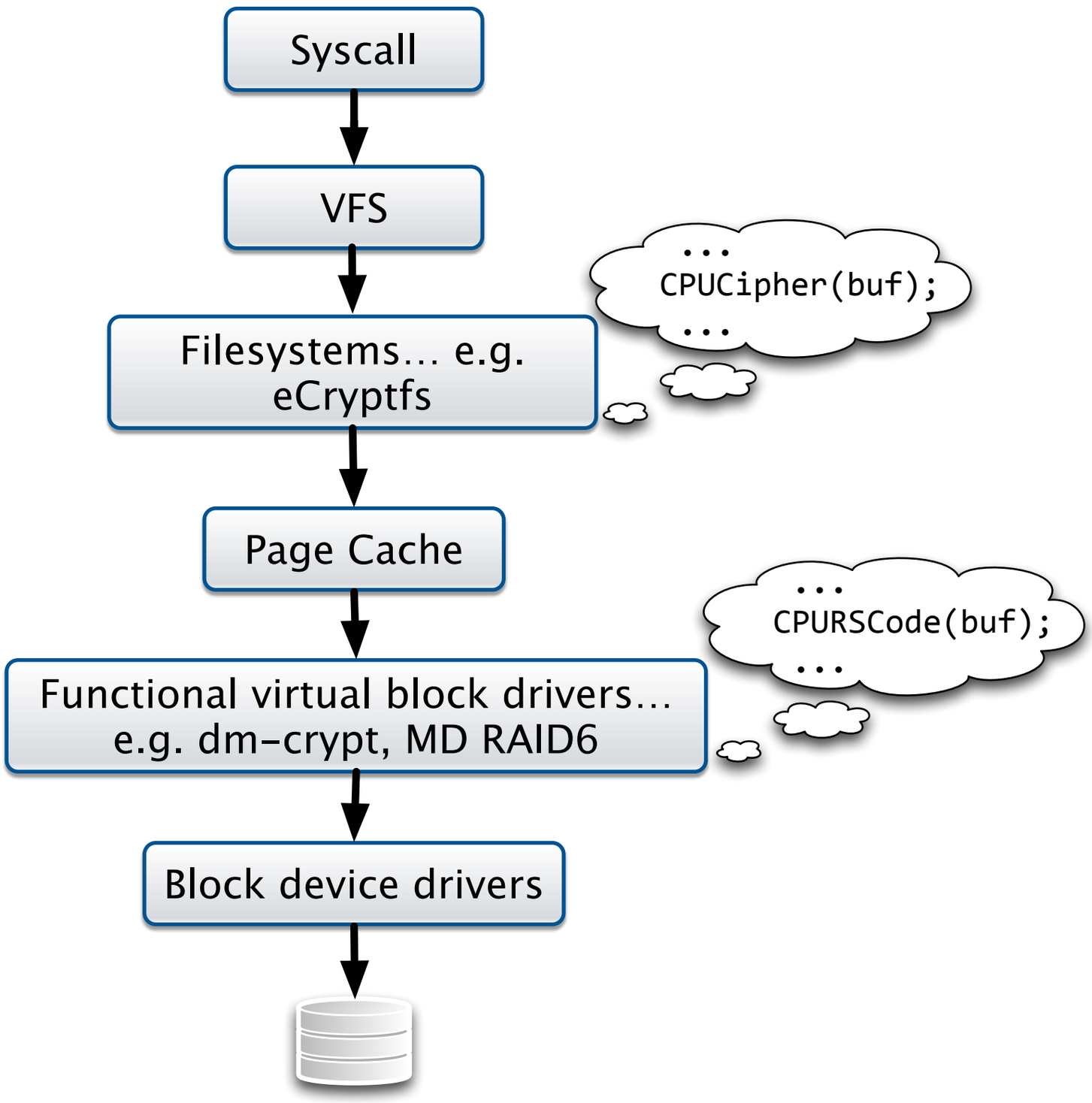
- A storage framework for Linux kernel to use CUDA GPUs in filesystems, block drivers, etc.
- Minimally invasive GPU integration
- Small changes
- Preserve interface and semantics
- Keep efficient



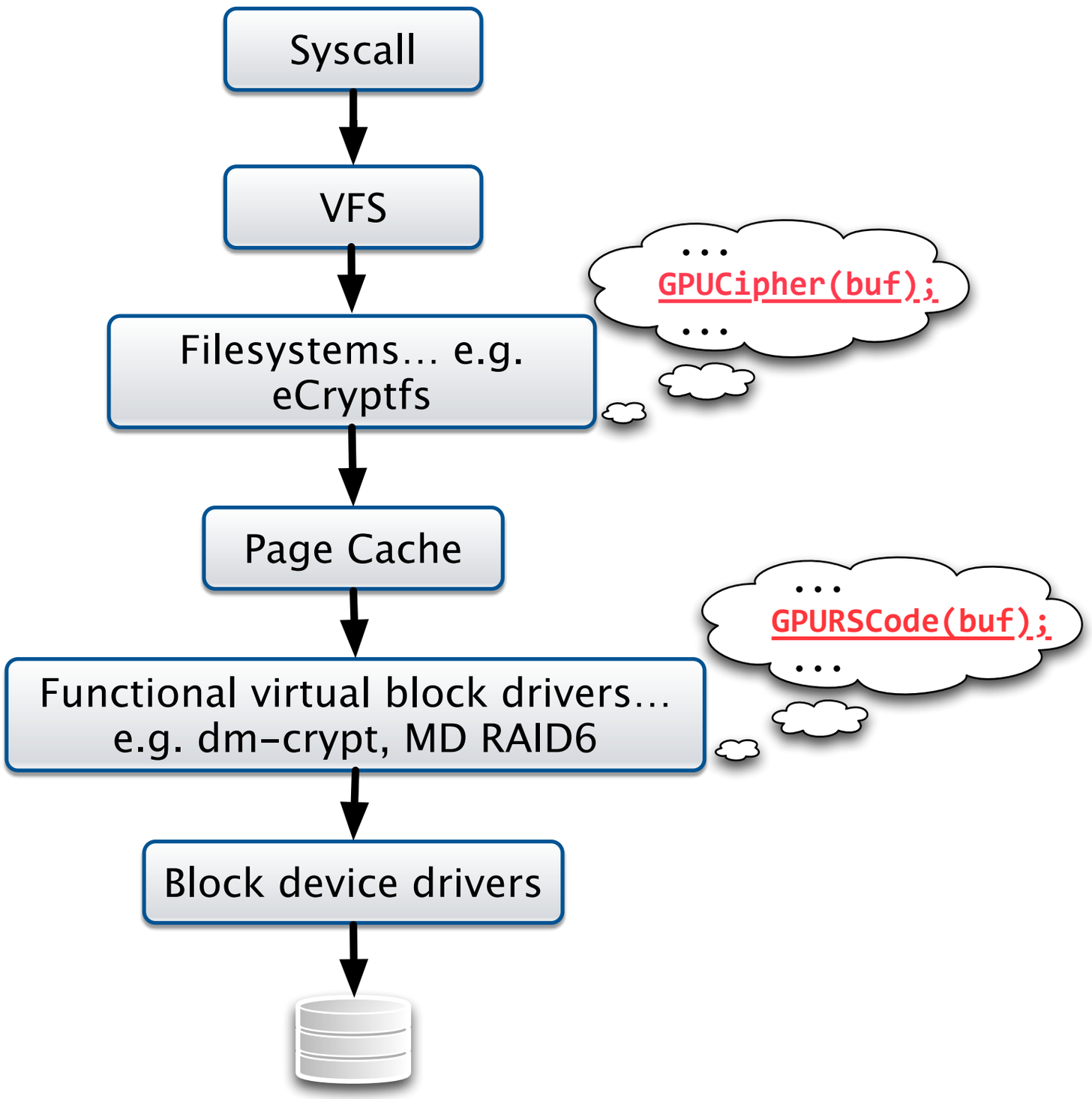
GPUstore integration



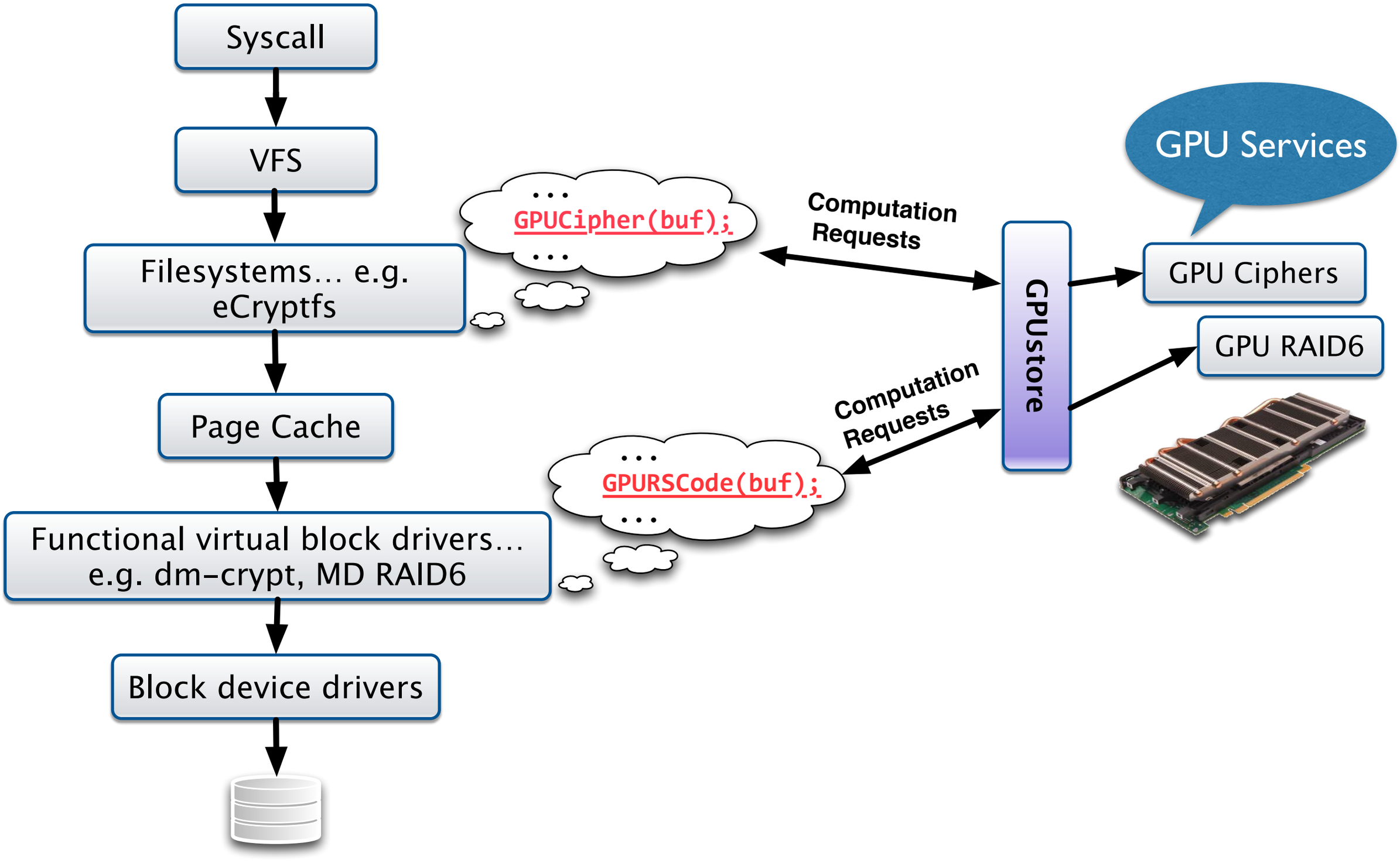
GPUstore integration



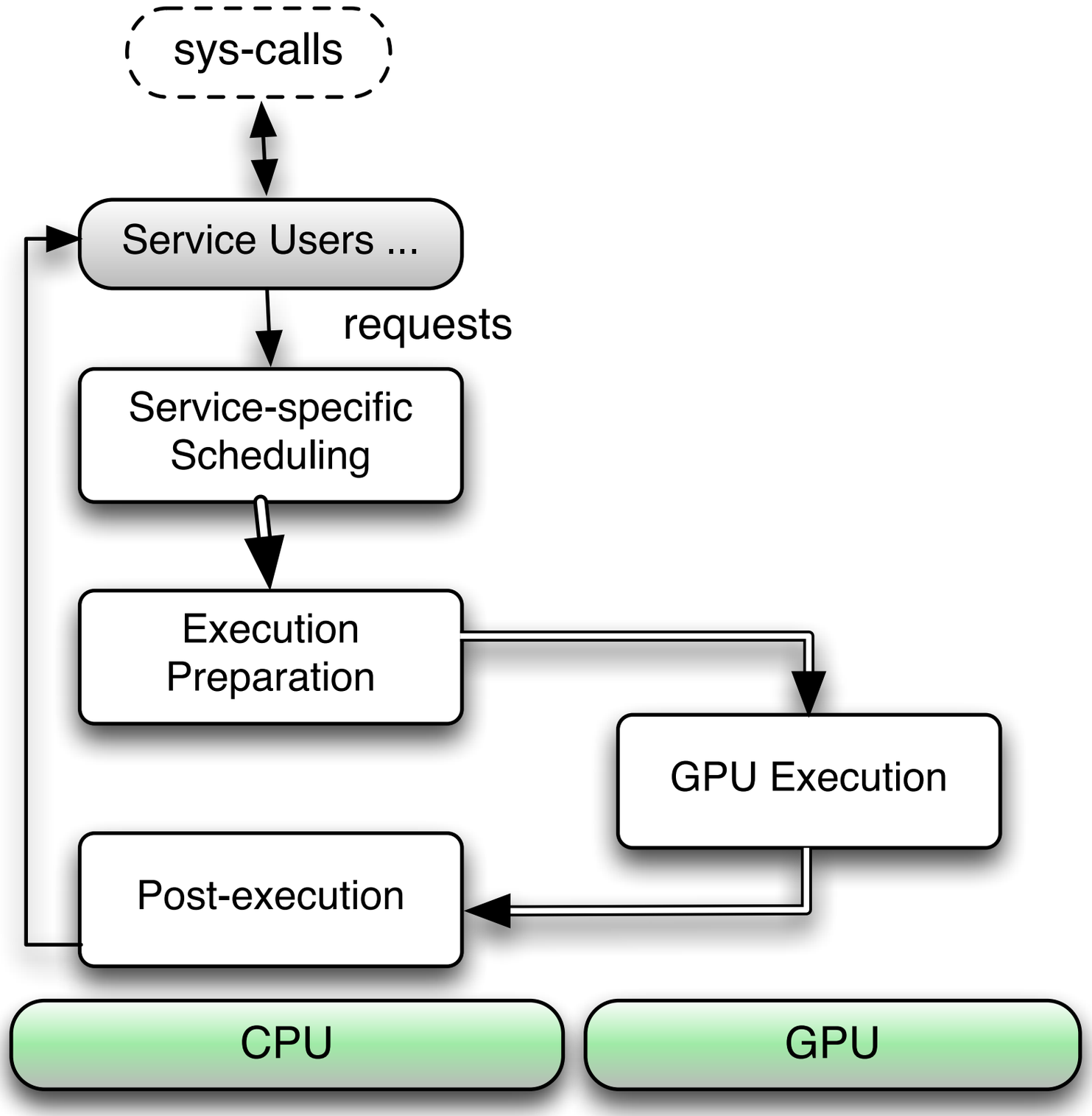
GPUstore integration



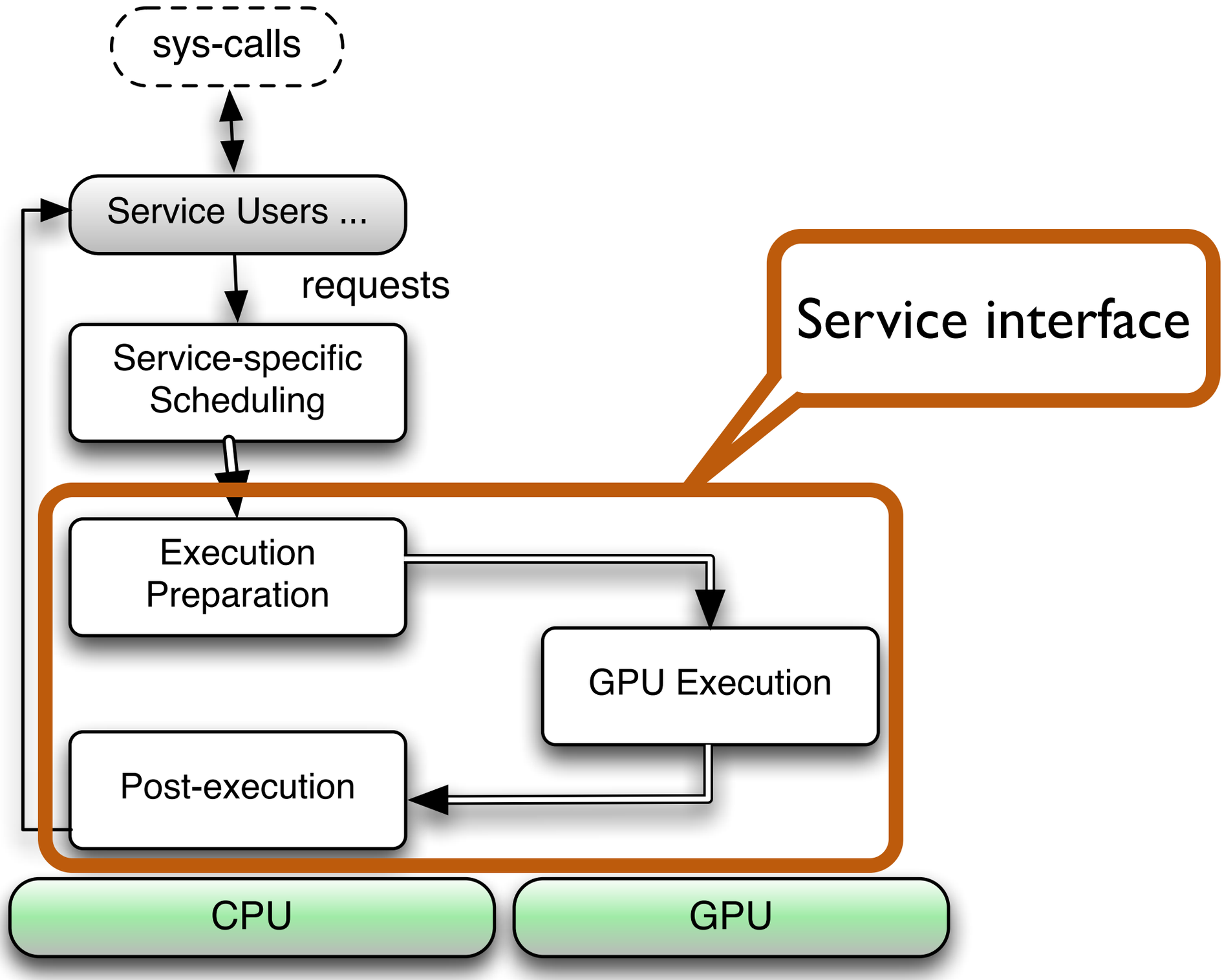
GPUstore integration



GPUstore request workflow



GPUstore request workflow



Apply generic principles

- Batched processing
- Asynchronous non-blocking GPU programming
- Reduce PCIe data transfer with compacted workload
- All use the same locked memory for zero-copy DMA

Apply generic principles

- Batched processing

- Asynchronous non-blocking GPU programming

Apply generic principles

- Batched processing
  - Merge small requests
- Asynchronous non-blocking GPU programming

Apply generic principles

- Batched processing
  - Merge small requests
- Asynchronous non-blocking GPU programming
  - Split large requests

Apply generic principles

- Batched processing
  - Merge small requests
- Asynchronous non-blocking GPU programming
  - Split large requests
  - Callback-based request processing

Request scheduling

- Merge small requests
- Split large requests

Request scheduling

- Merge small requests
- Split large requests
  - How to define “small” and “large”?

Request scheduling

- Merge small requests
- Split large requests
  - How to define “small” and “large”?
    - Up to specific services.

Apply generic principles

- Batched processing
- Asynchronous non-blocking GPU programming
- Reduce PCIe data transfer with compacted workload
- All use the same locked memory for zero-copy DMA

Apply generic principles

- Reduce PCIe data transfer with compacted workload

- All use the same locked memory for zero-copy DMA

Apply generic principles

- All use the same locked memory for zero-copy DMA

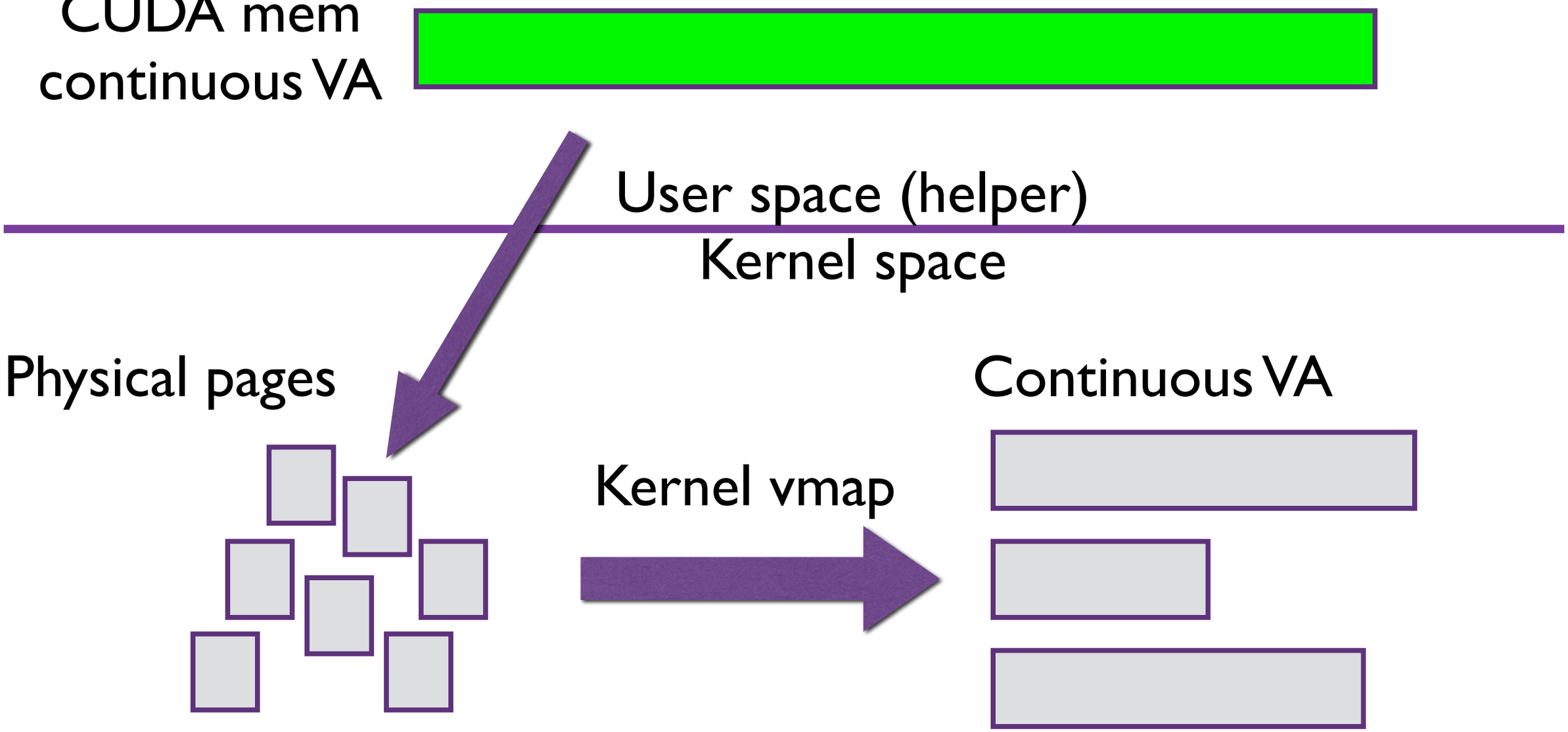
Apply generic principles

- All use the same locked memory for zero-copy DMA
- User space CUDA library dependency

Apply generic principles

- All use the same locked memory for zero-copy DMA
  - User space CUDA library dependency
    - In-kernel CUDA's locked memory allocator

In-kernel CUDA locked memory allocator



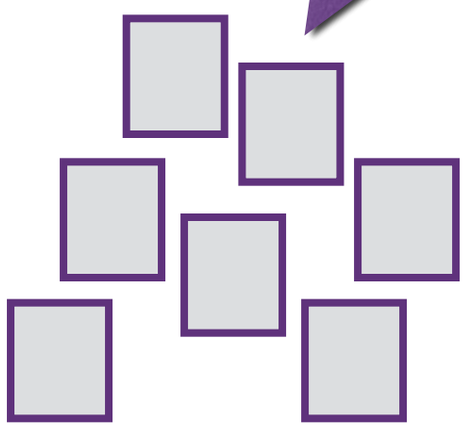
In-k ator

CUD
conti

What's next?

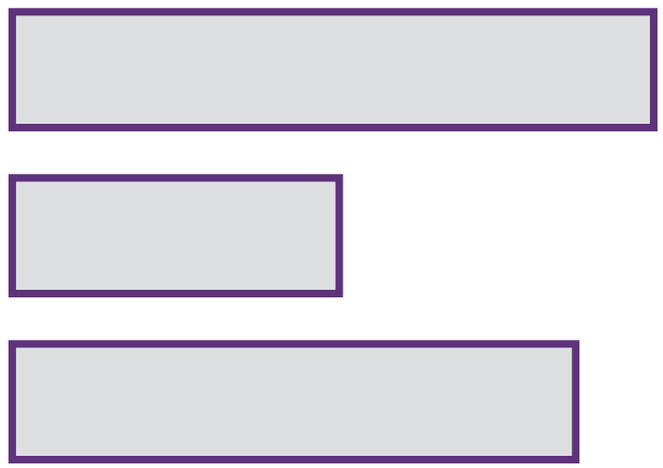
Replace *kmalloc()/vmalloc()/ get_free_pages()* with *GPUstoreMalloc()*?

Physical pages



Kernel vmap

Continuous VA



In-k ator

CUD
conti

Ph

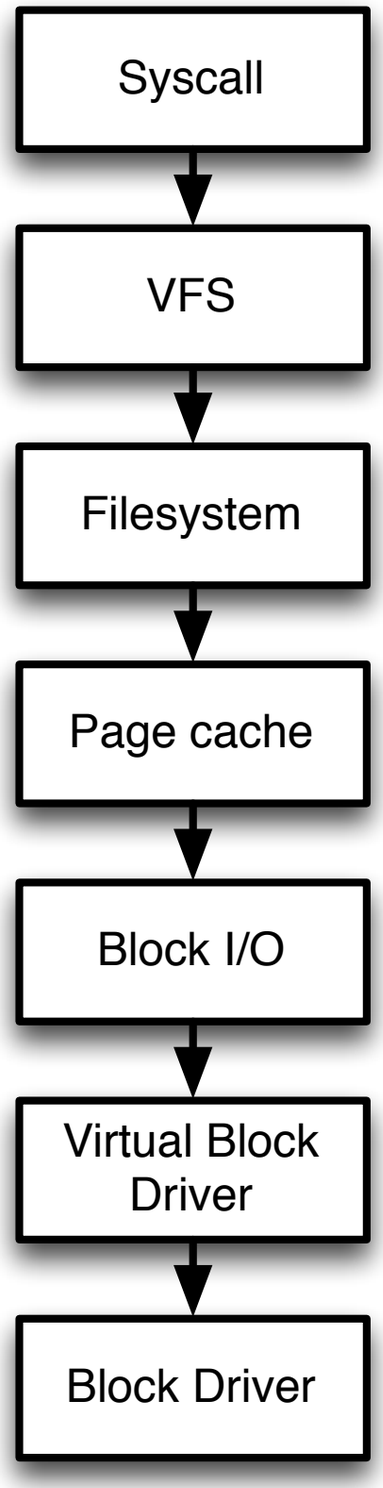
What's next?

Replace *kmalloc()/vmalloc()/ get_free_pages()* with *GPUstoreMalloc()*?

Sometimes it is infeasible!

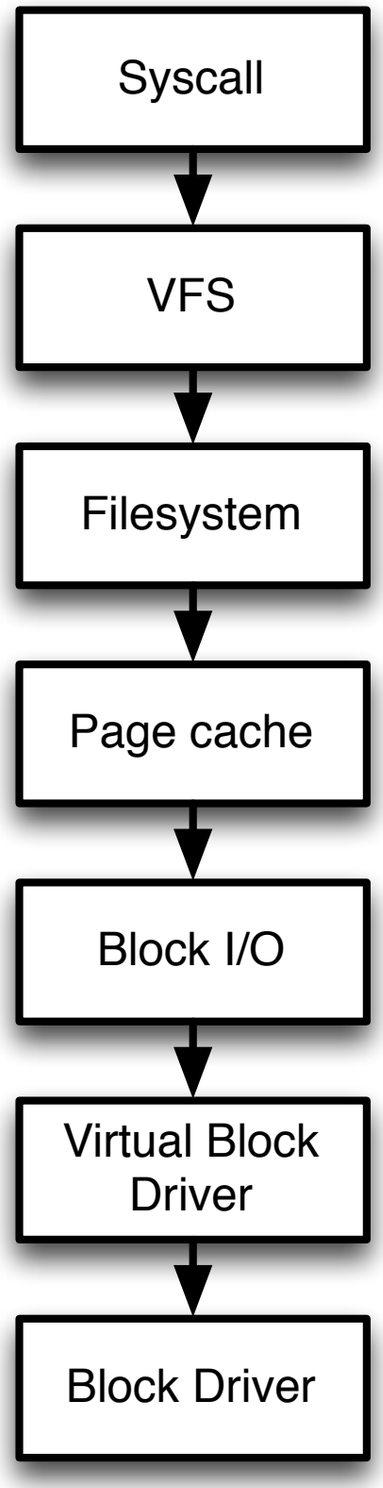
Why infeasible?

Why infeasible?



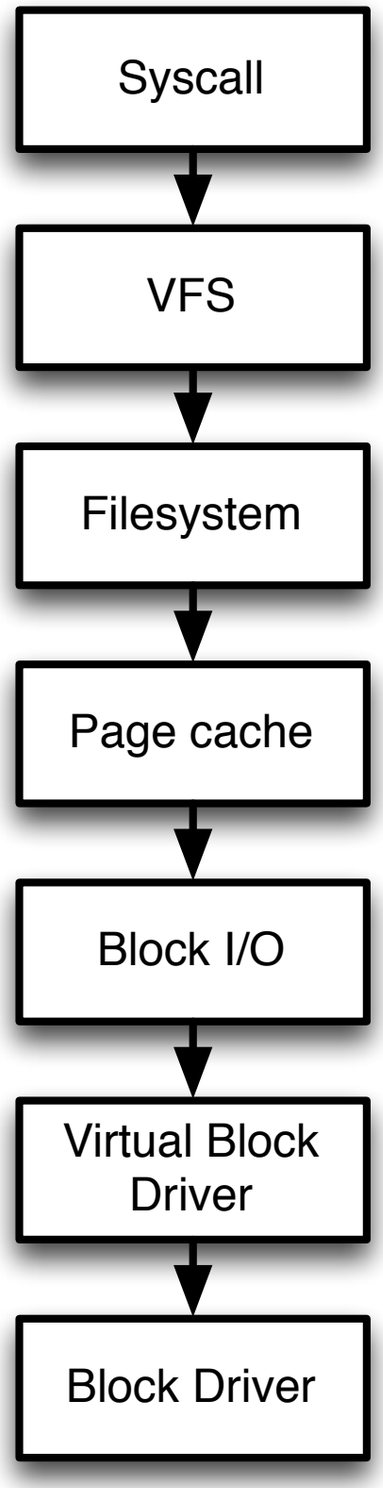
Why infeasible?

➡ Who allocates the memory?



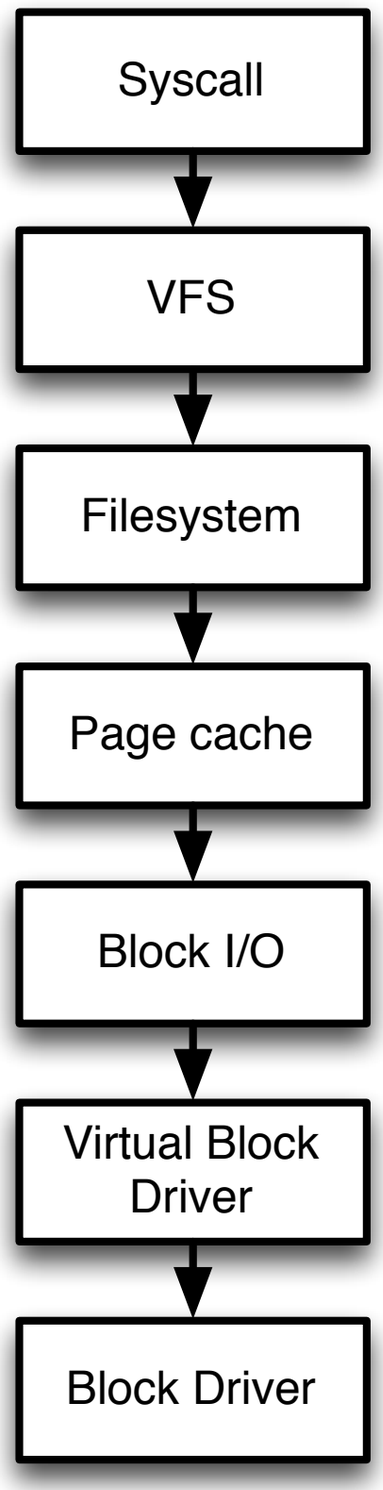
Why infeasible?

- Who allocates the memory?
  - "pass-by-ref" interface



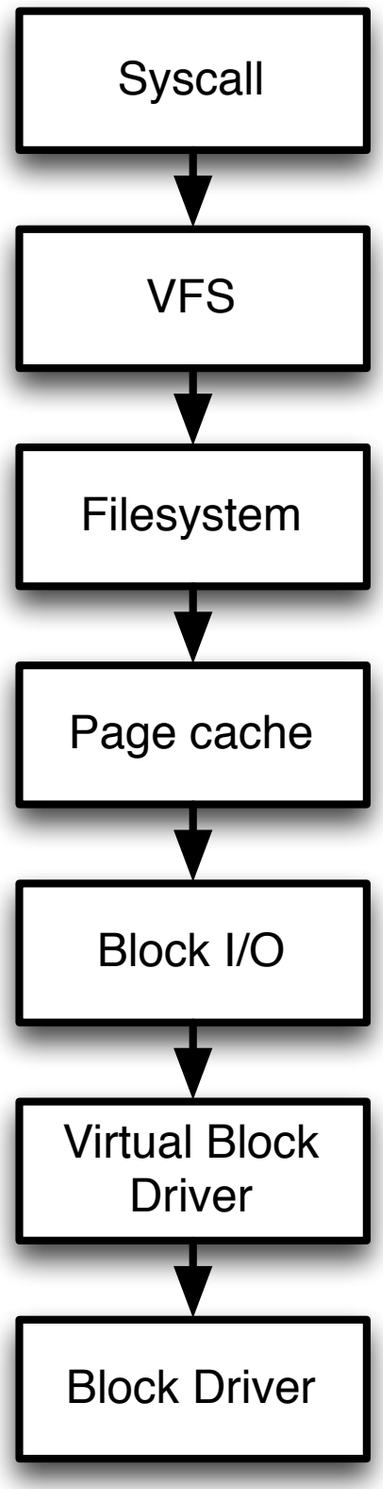
Why infeasible?

- Who allocates the memory?
  - “pass-by-ref” interface
- Can NOT modify highly-depended cache allocators:



Why infeasible?

- Who allocates the memory?
  - "pass-by-ref" interface
- Can NOT modify highly-depended cache allocators:
  - page cache, buffer cache
  - object cache
  - packet pool
  - …



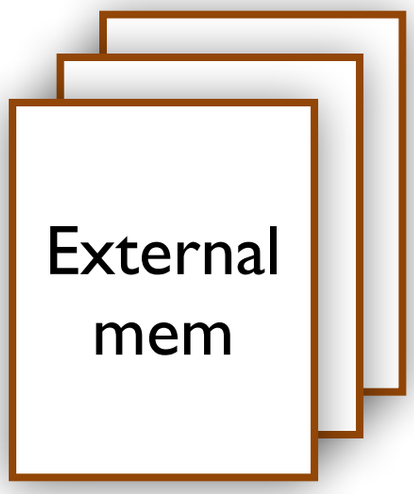
Why infeasible?

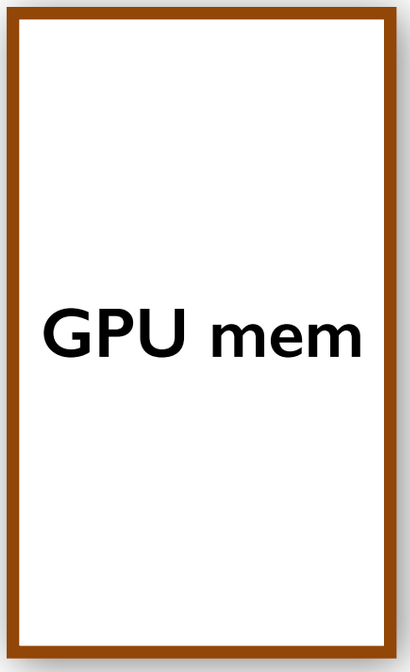
- Who allocates the memory?
  - "
  - packet pool
  - …

Can we use arbitrary memory for GPU DMA?

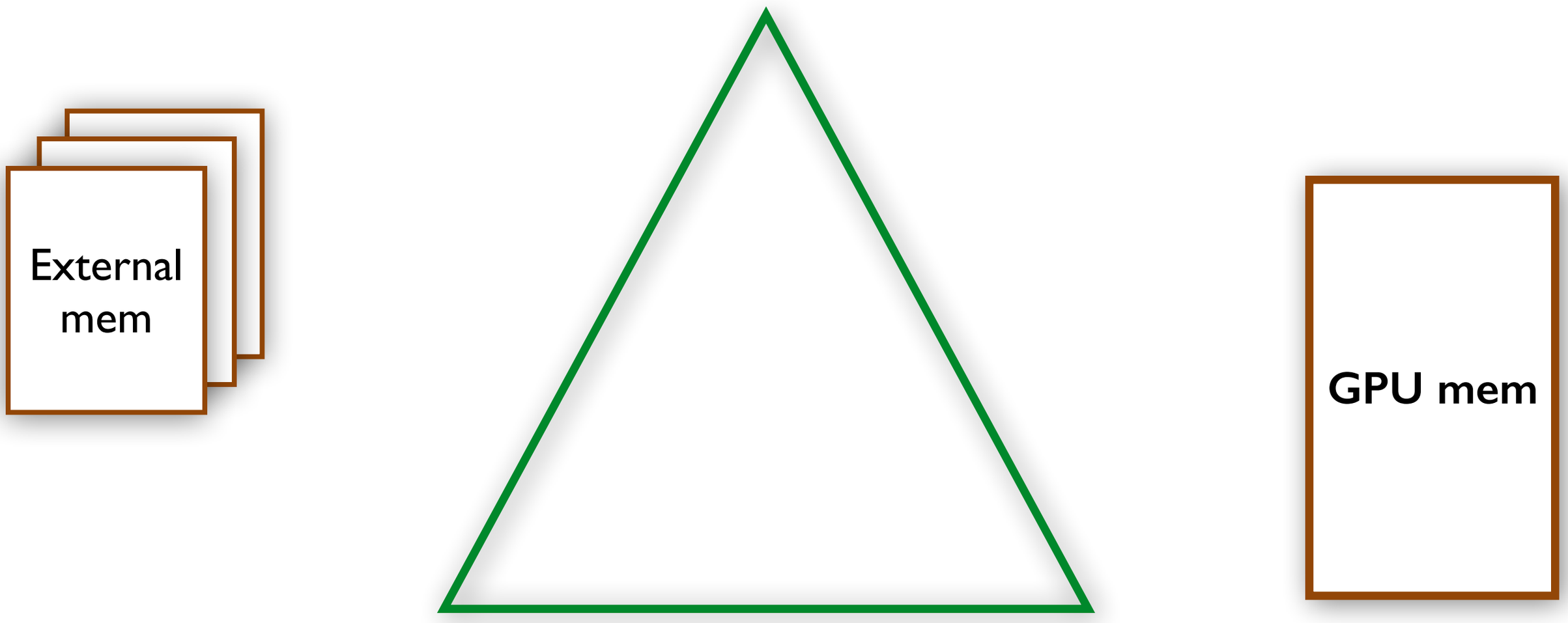
Remap external memory for GPU DMA

Remap external memory for GPU DMA

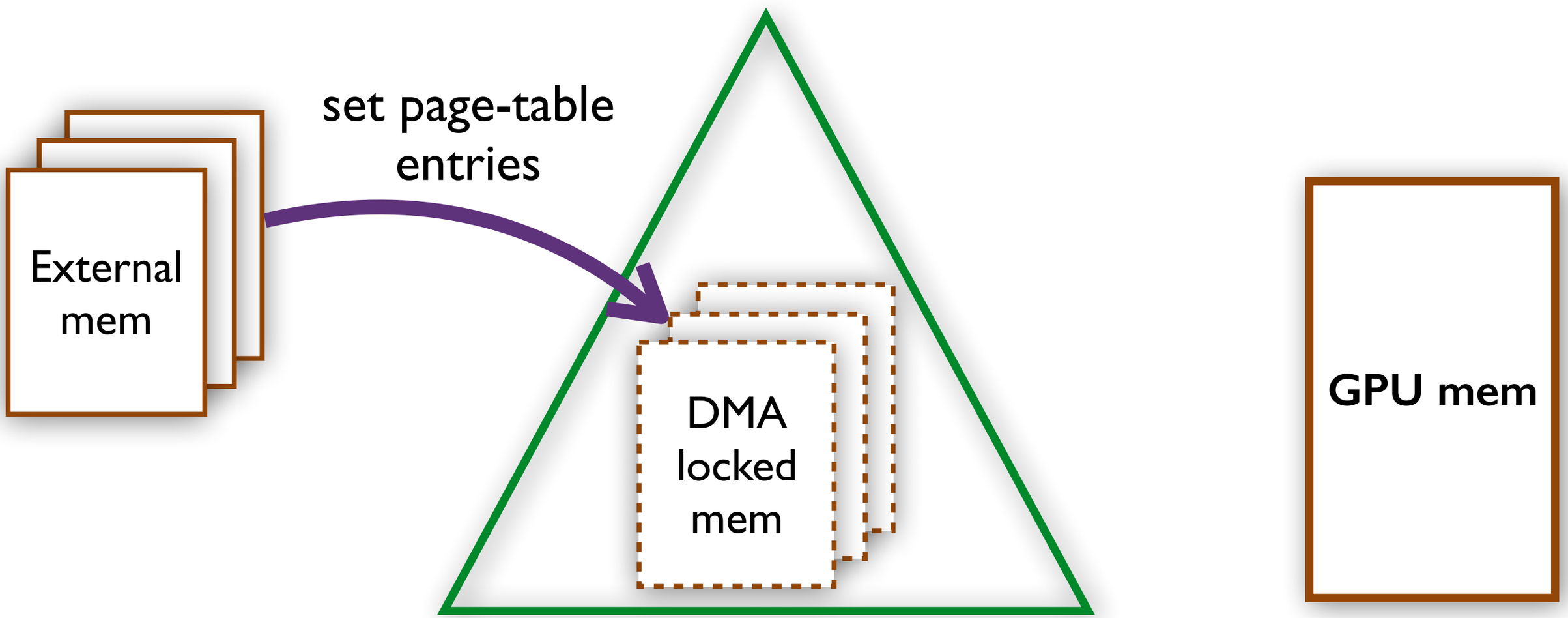




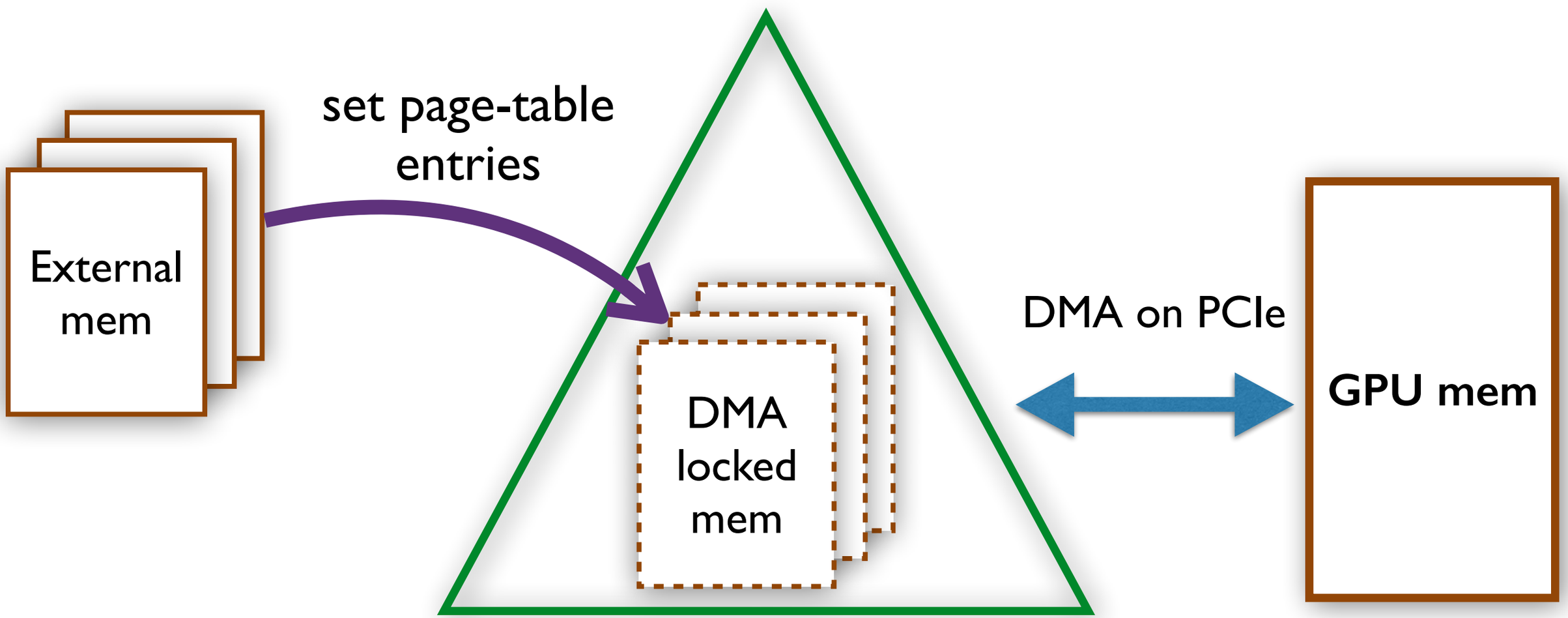
Remap external memory for GPU DMA



Remap external memory for GPU DMA



Remap external memory for GPU DMA



Apply generic principles

- All use the same locked memory for zero-copy DMA

Apply generic principles

- All use the same locked memory for zero-copy DMA
  - In-kernel CUDA locked memory allocator

Apply generic principles

- All use the same locked memory for zero-copy DMA
  - In-kernel CUDA locked memory allocator
  - Remap external memory into GPU driver's locked memory area

Implementation and evaluation

Implementation and evaluation

- Case studies of major storage residents:
  - **dm-crypt**: disk encryption layer
  - **eCryptfs**: encrypted filesystem
  - **md**: software RAID6

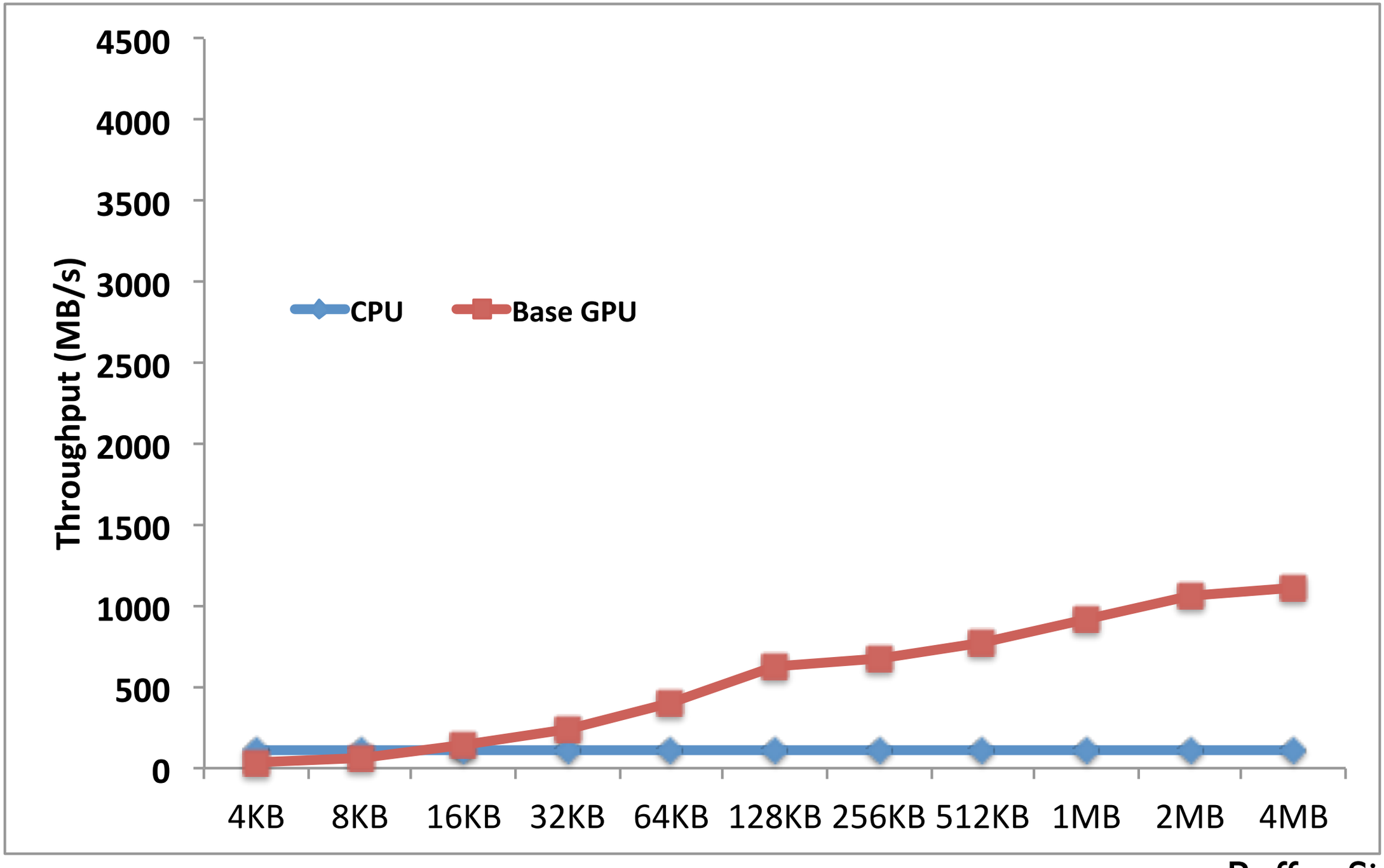
Implementation and evaluation

- Case studies of major storage residents:
  - **dm-crypt**: disk encryption layer
  - **eCryptfs**: encrypted filesystem
  - **md**: software RAID6

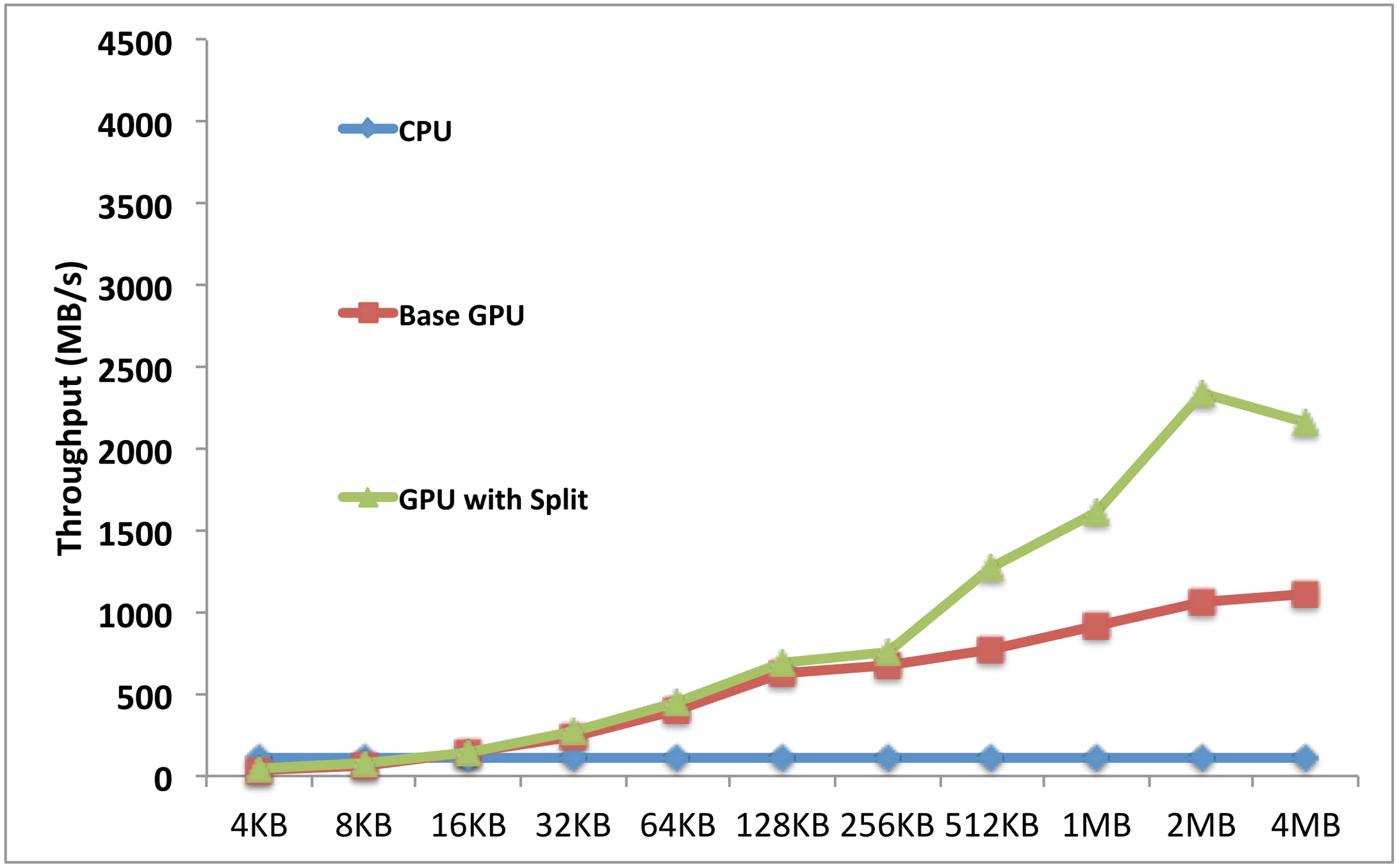
- Integration cost:

Subsystem	Total LOC	Modified LOC	Percent
`dm-crypt`	1,800	50	3%
`eCryptfs`	11,000	200	2%
`md`	6,000	20	0.3%

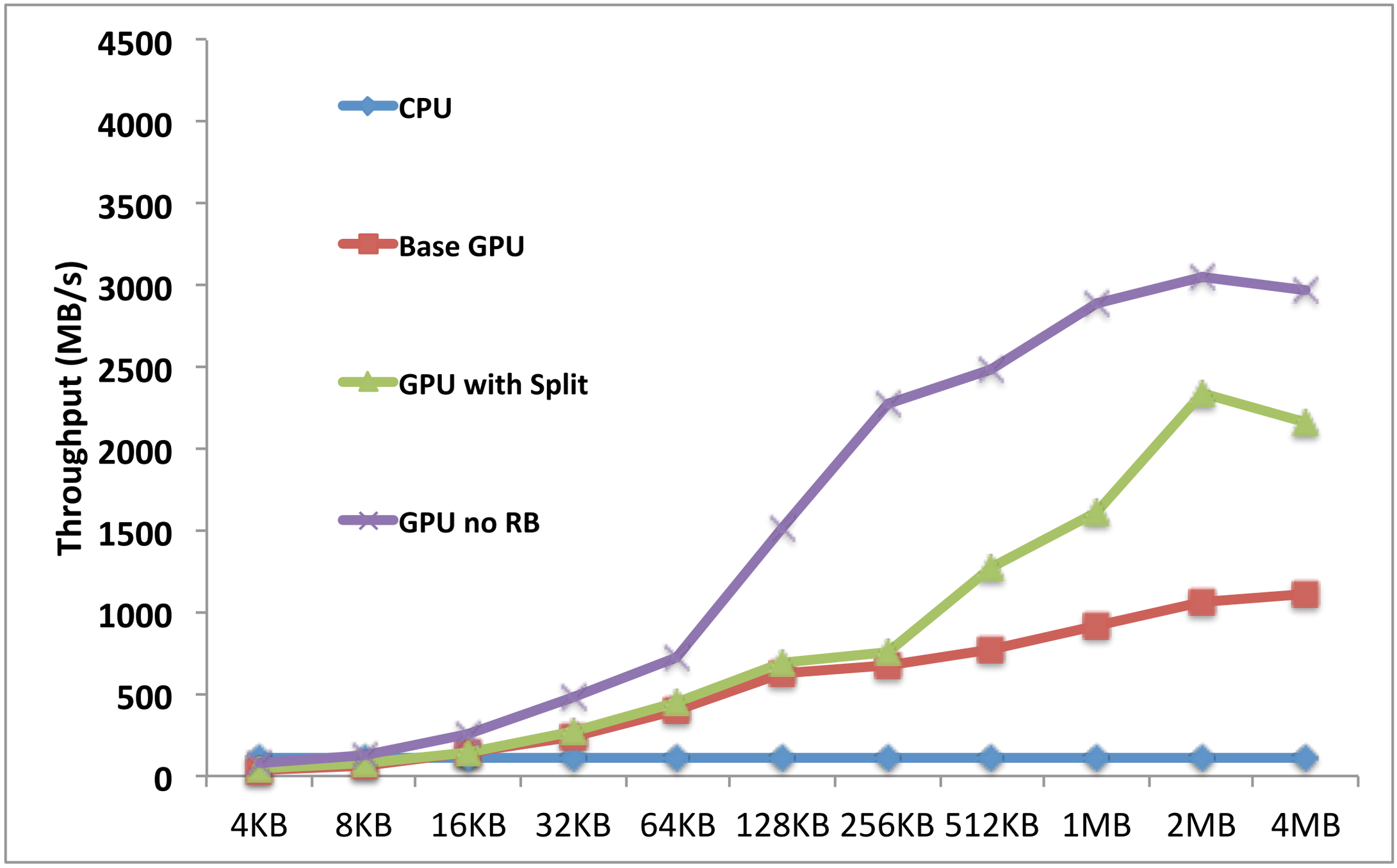
AES cipher performance



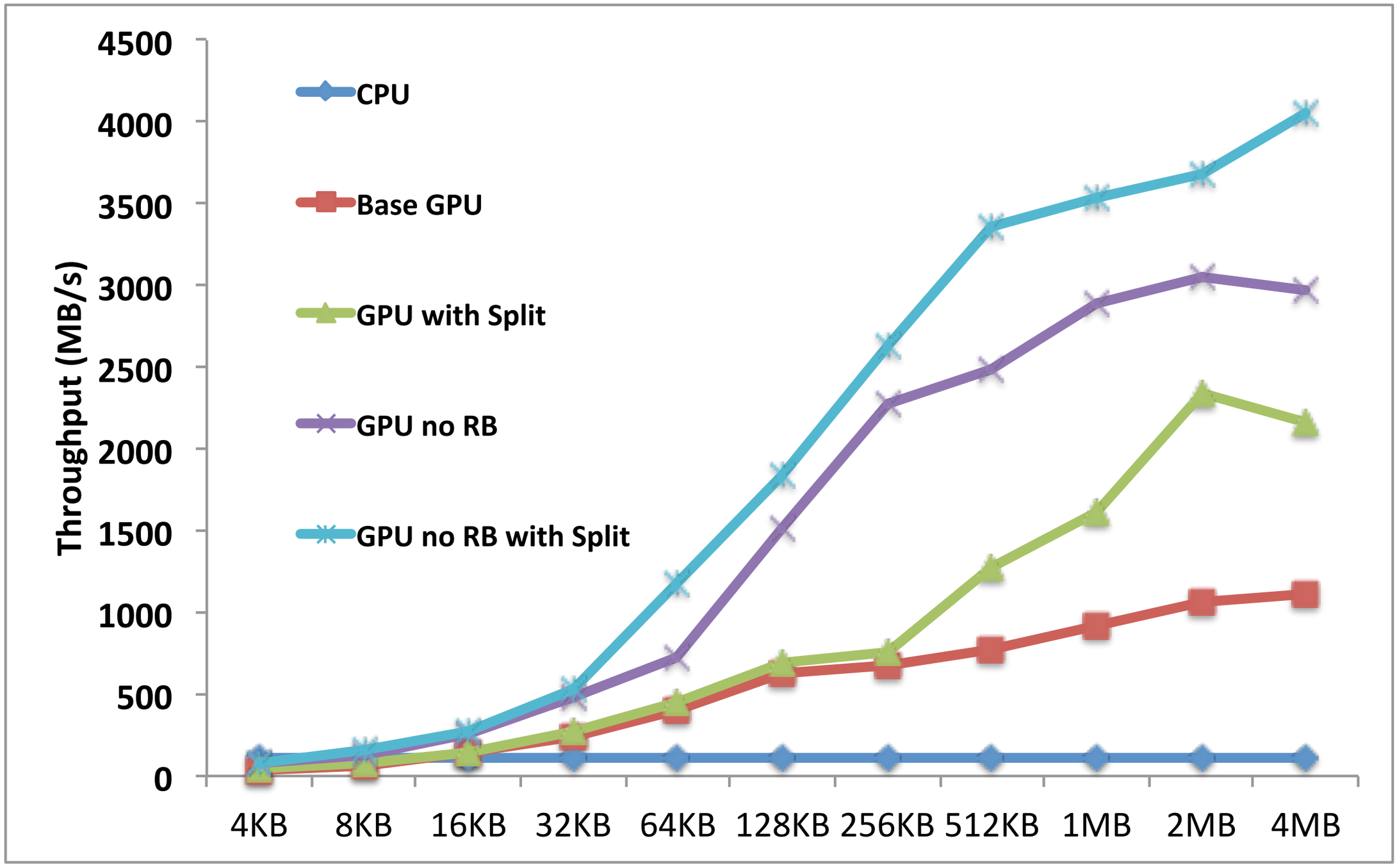
AES cipher performance



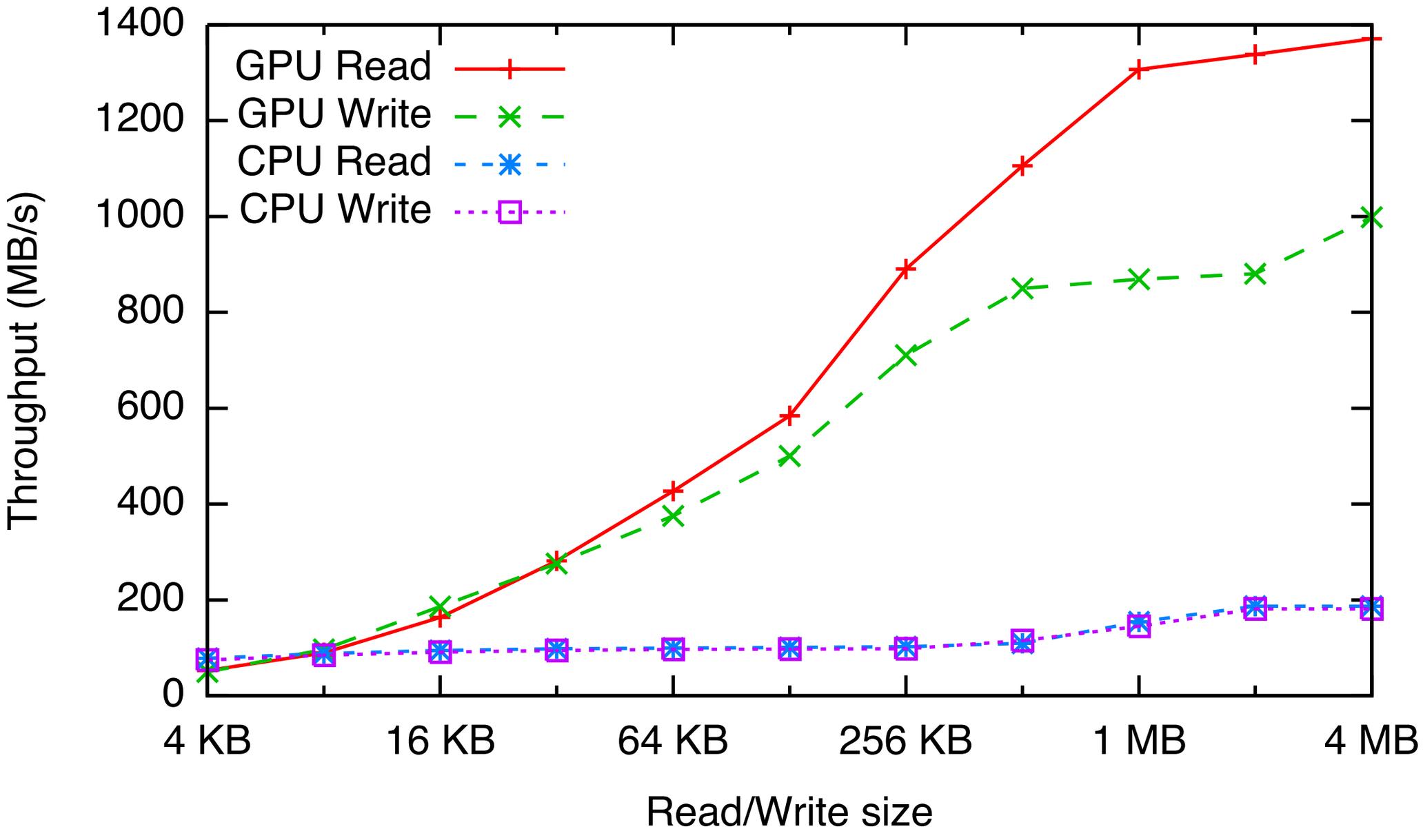
AES cipher performance



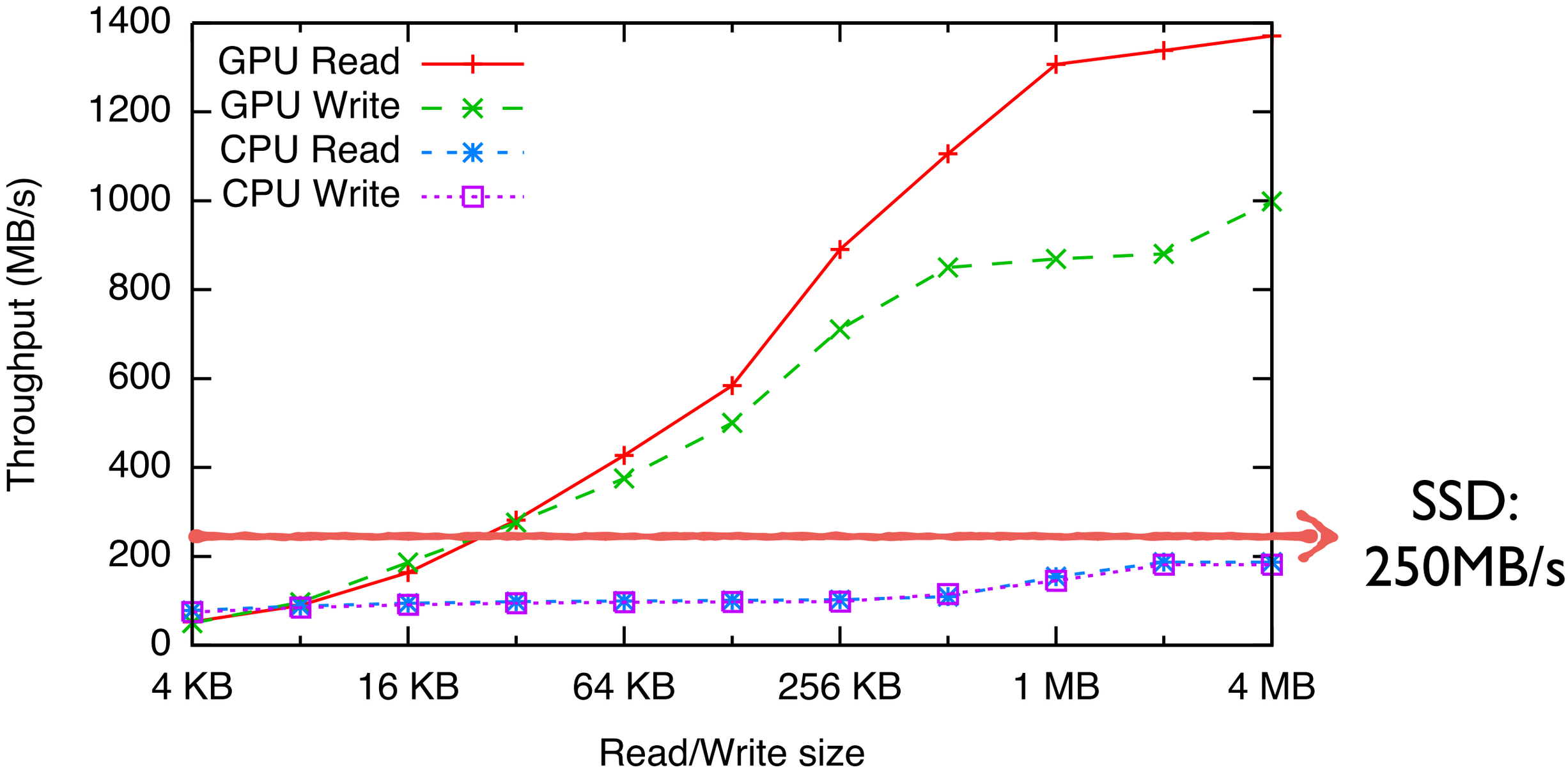
AES cipher performance



Faster than SSD: dm-crypt



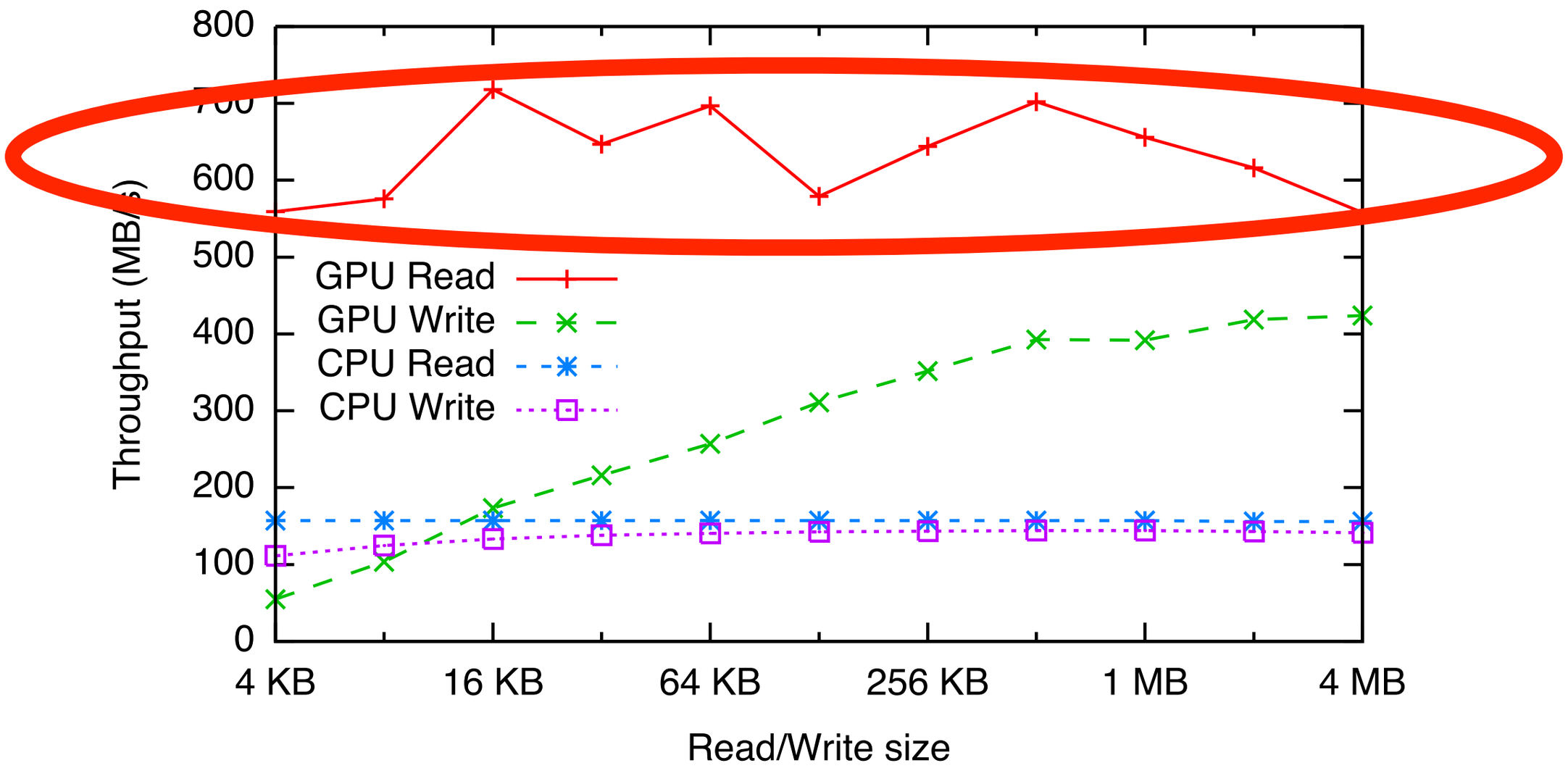
Faster than SSD: dm-crypt



Working with existing optimization

eCryptfs on RAM disks

Linux max 128KB read-ahead effects on read



GPUstore Summary

- Enables efficient GPU-accelerated storage in Linux kernel with small changes

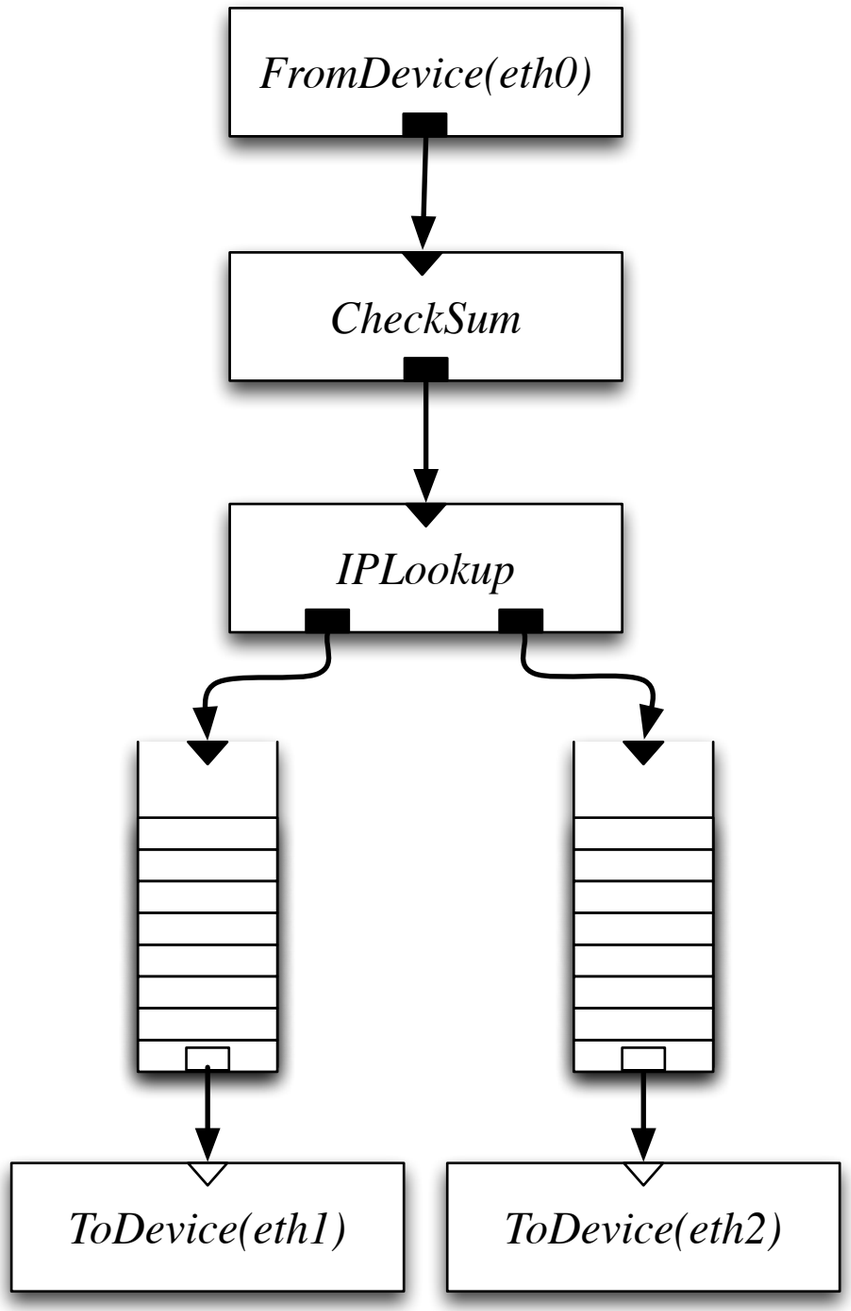
- https://github.com/wbsun/kgpu

- Details not presented: user-space helper, non-blocking K-U comm, more exp result

Snap:
Fast and Flexible Packet Processing With GPUs and Click

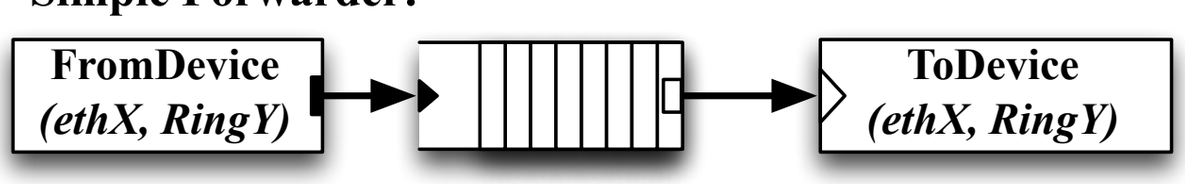
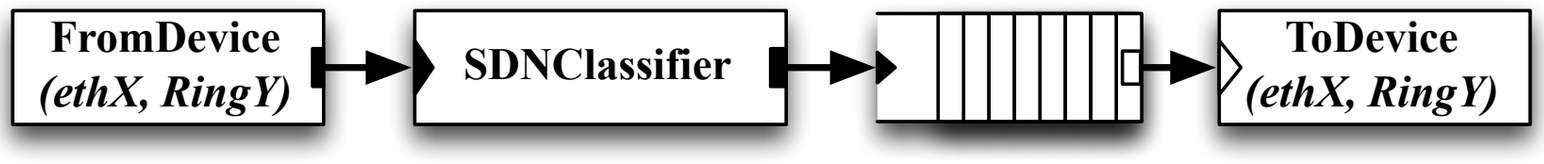
[Weibin Sun, Robert Ricci @ ANCS'13]

Click background [Kohler, et.al. TOCS'00]



- Elements
- Ports
- push/pull

Why using GPUs in Click?

Configurations	Throughput	Slow down
Simple Forwarder: 	30.97 Gbps	N/A
Simple SDN Forwarder: 	17.7 Gbps	42.7%

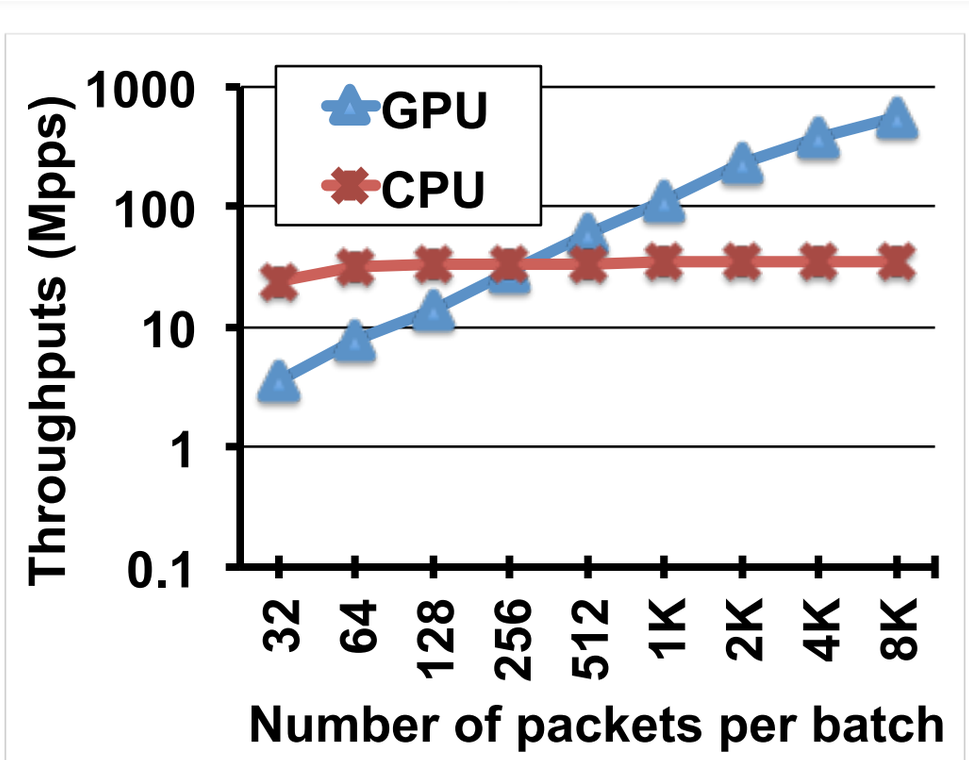
Why using GPUs in Click?

Configurations	ut	Slow down
Simple Forwa **FromDevice** *(ethX, RingY)*		N/A
Simple SDN Forw **FromDevice** *(ethX, RingY)*	.7 Gbps	42.7%

Packet processing needs more computing power!

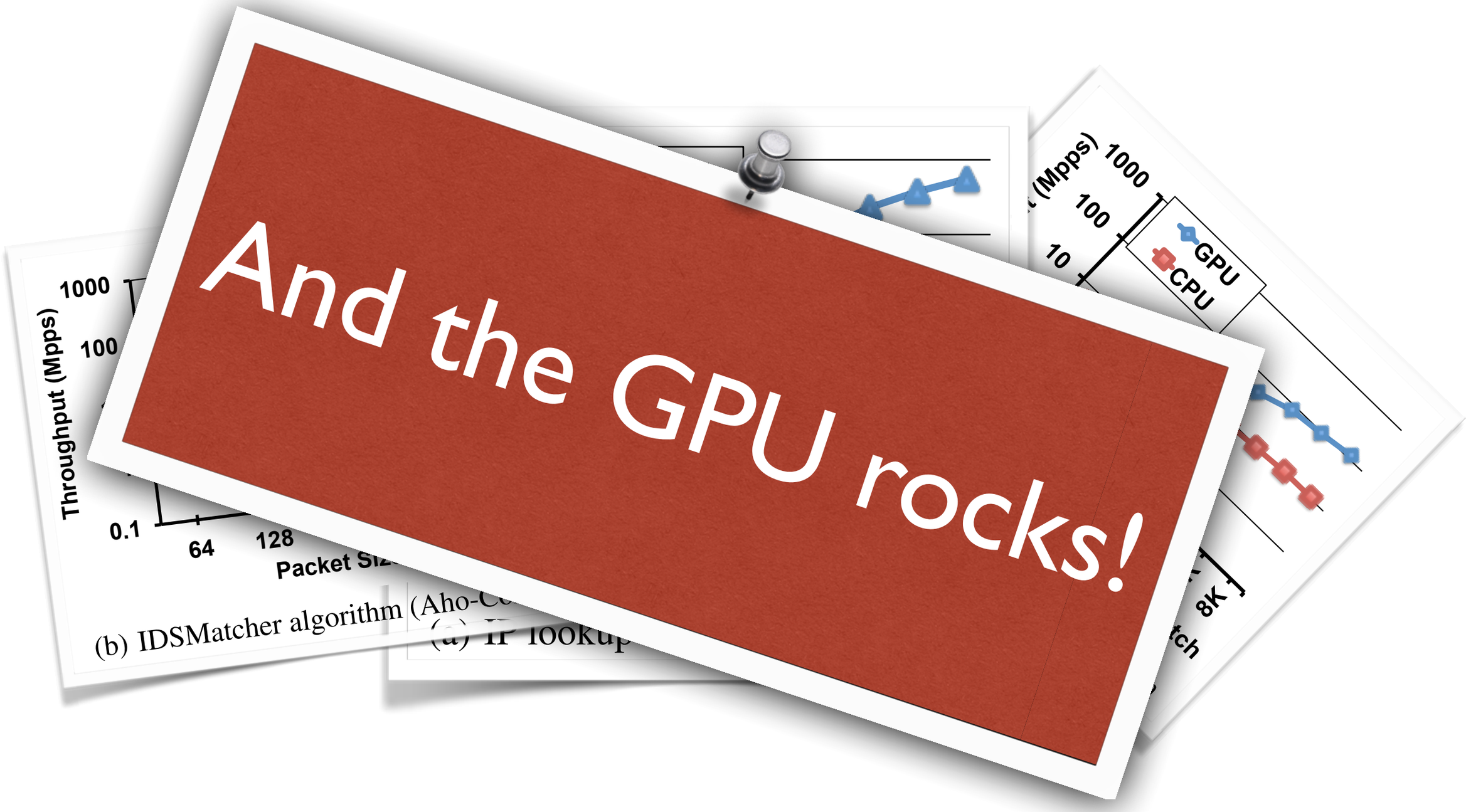
Why using GPUs in Click?

Why using GPUs in Click?



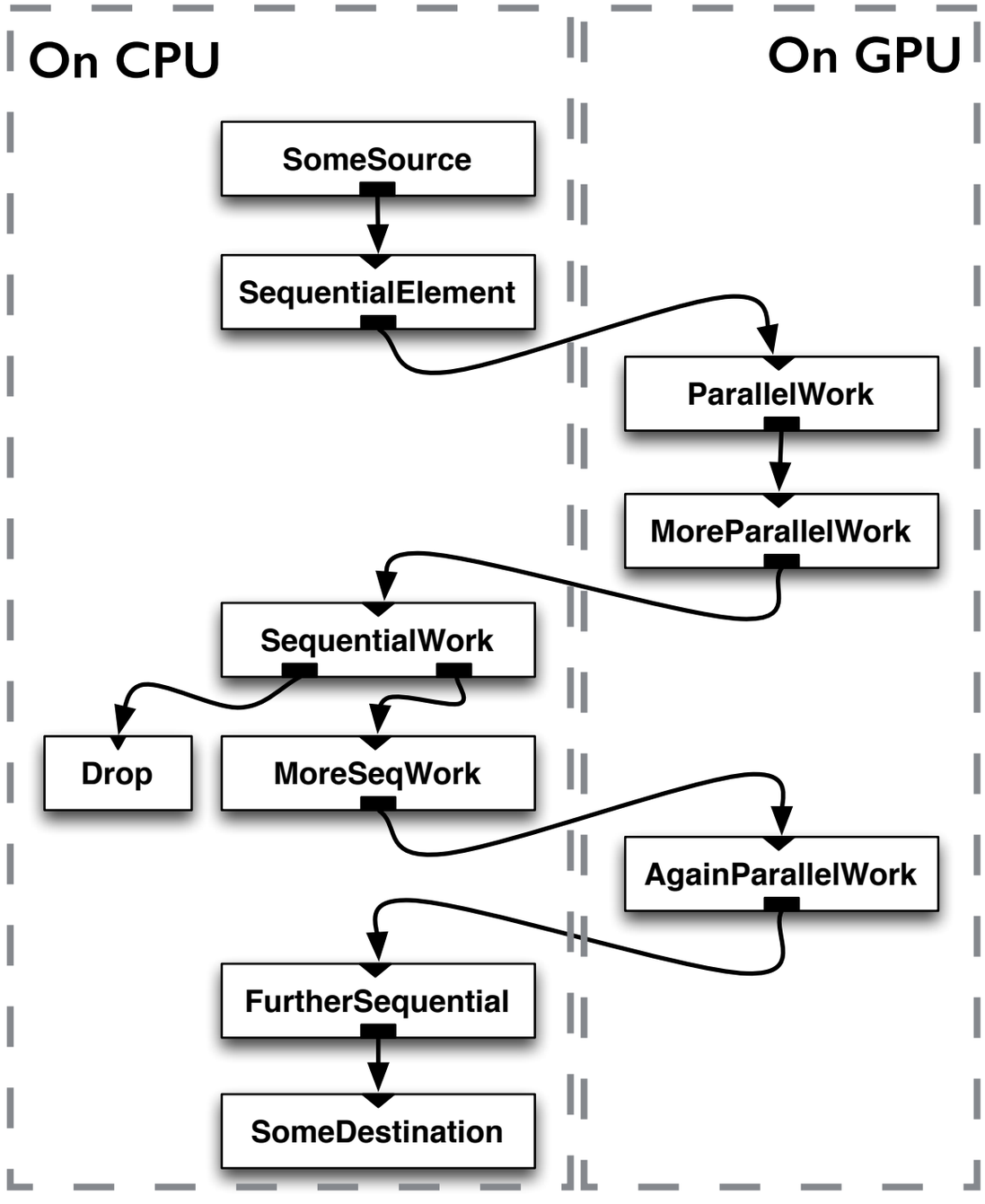
(a) IP lookup algorithm (radix tree)

Why using GPUs in Click?



Snap: the idea

- Moving parts of Click pipeline onto GPUs
  - CPU for sequential
  - GPU for parallel
- Keep Click's modular style



Apply generic principles

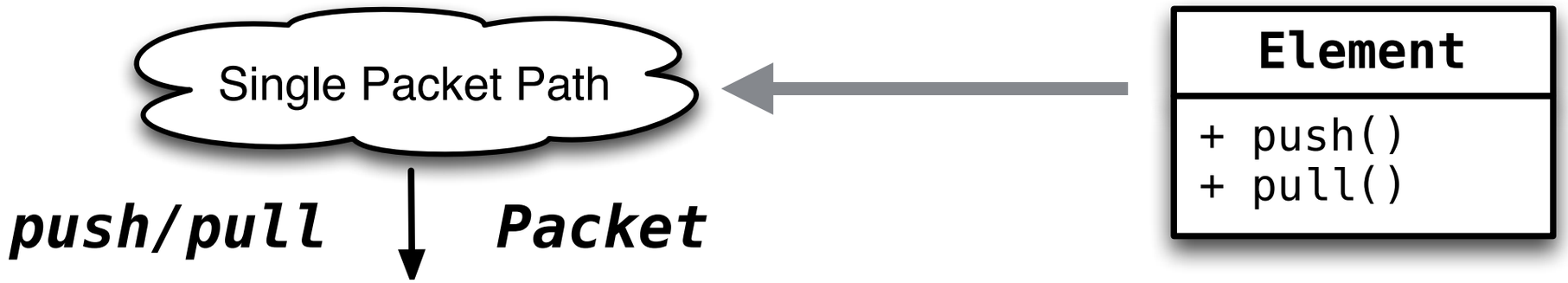
- Batched processing
- Asynchronous non-blocking GPU programming
- Reduce PCIe data transfer with compacted workload
- All use the same locked memory for zero-copy DMA

Apply generic principles

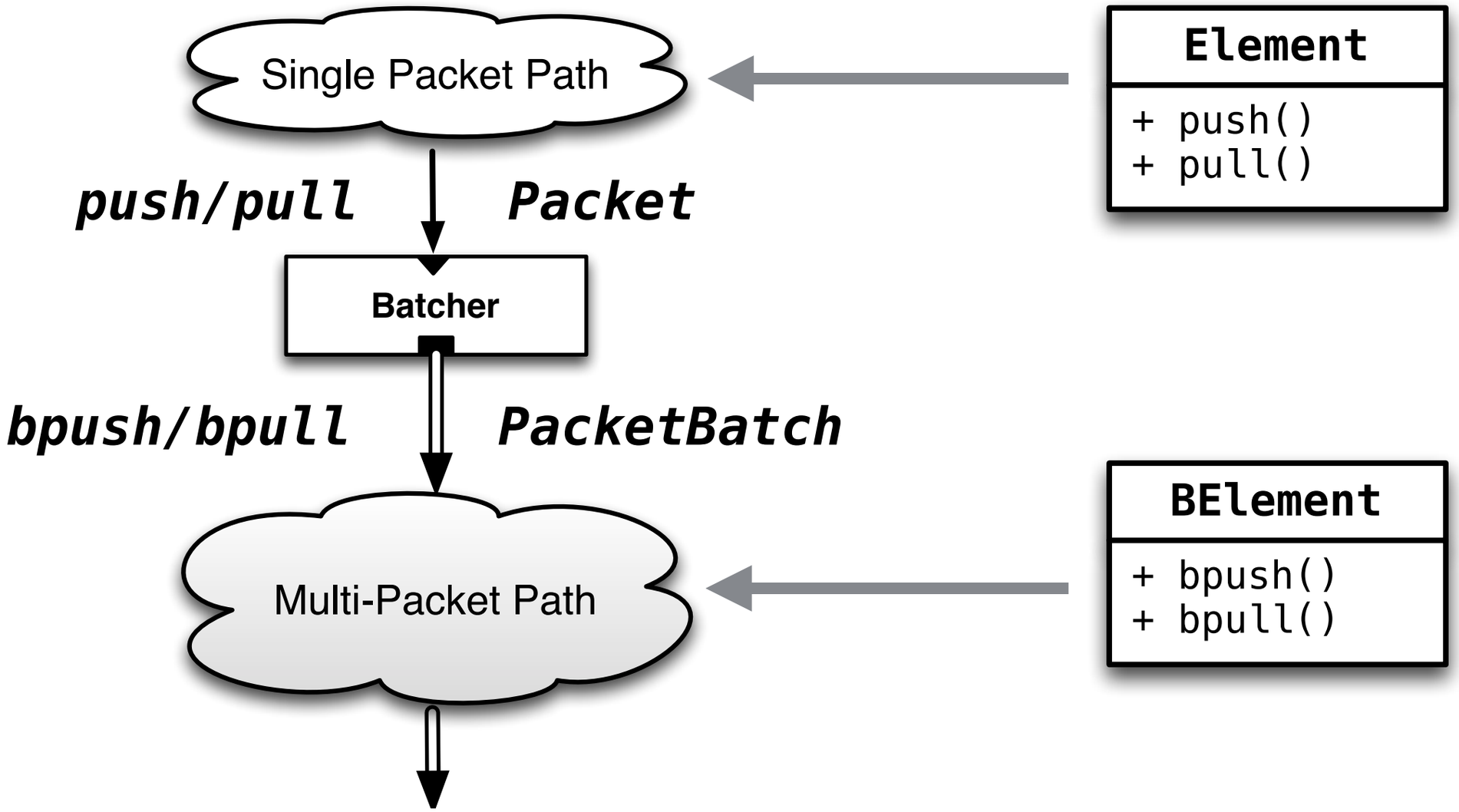
- Batched processing

Snap batched processing

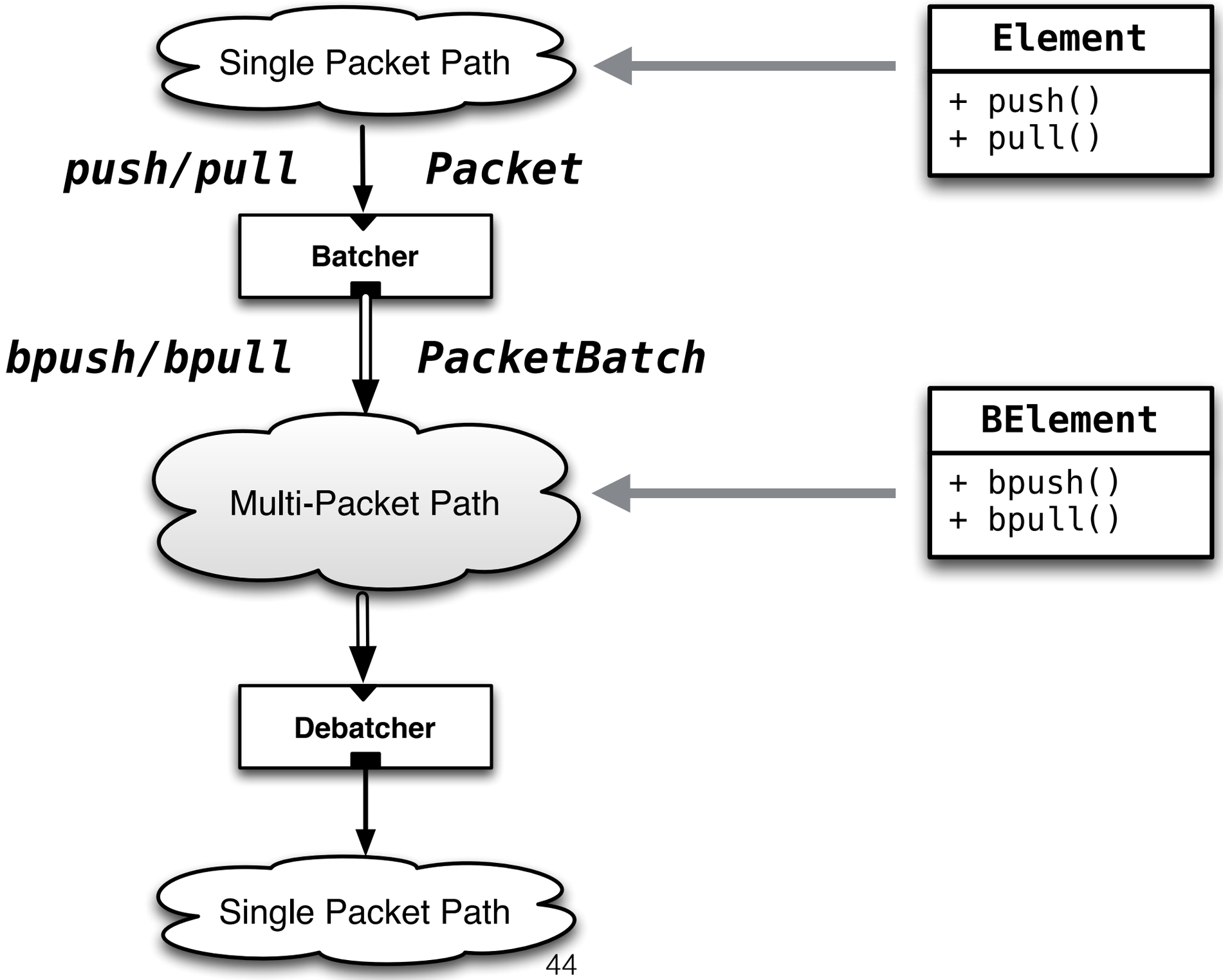
Snap batched processing



Snap batched processing

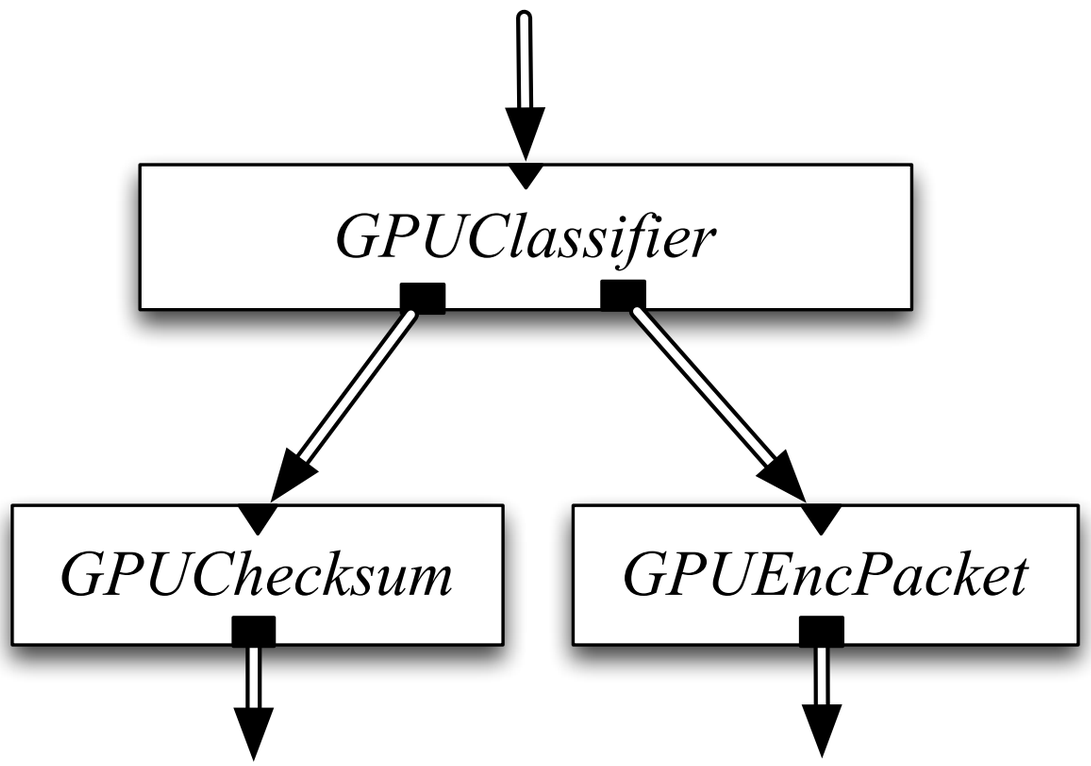


Snap batched processing



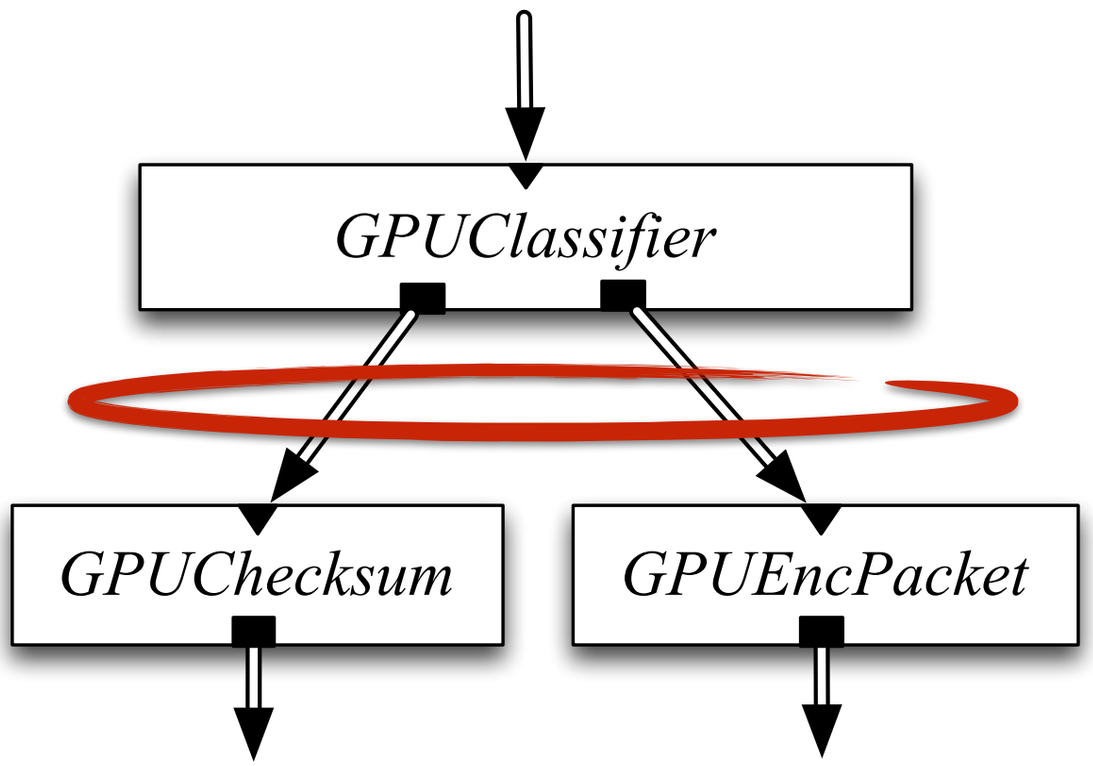
Batched packet divergence

- Packets in a batch go to different paths



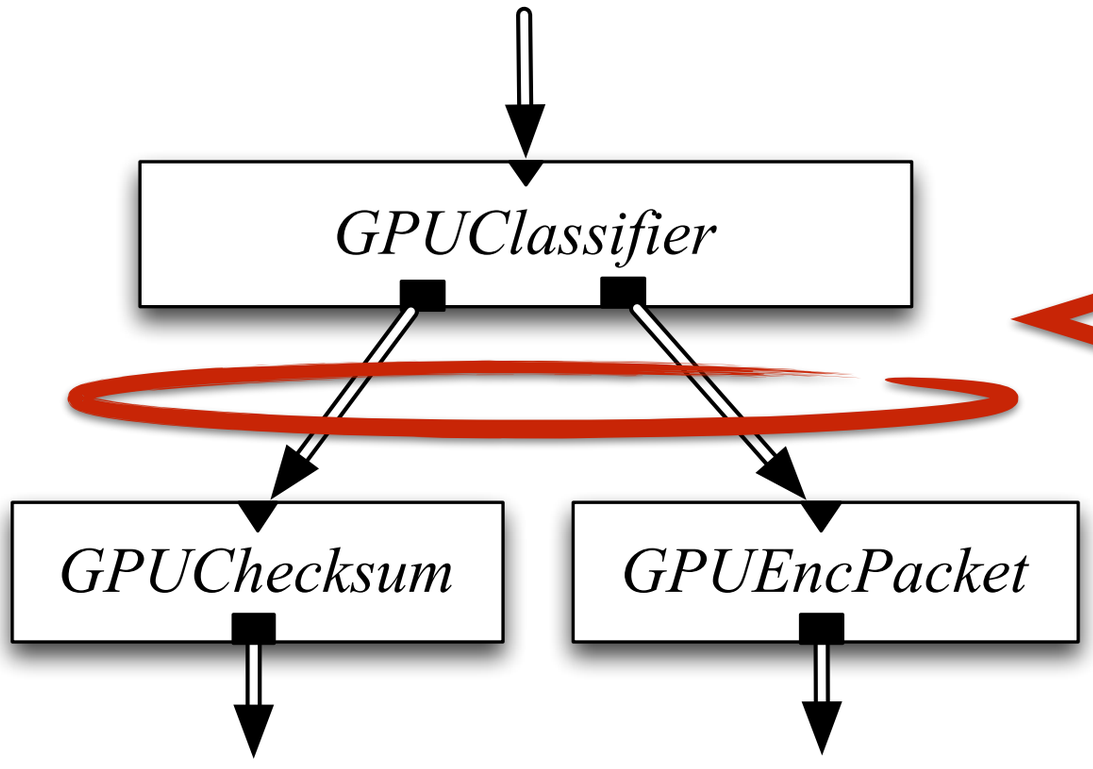
Batched packet divergence

- Packets in a batch go to different paths



Batched packet divergence

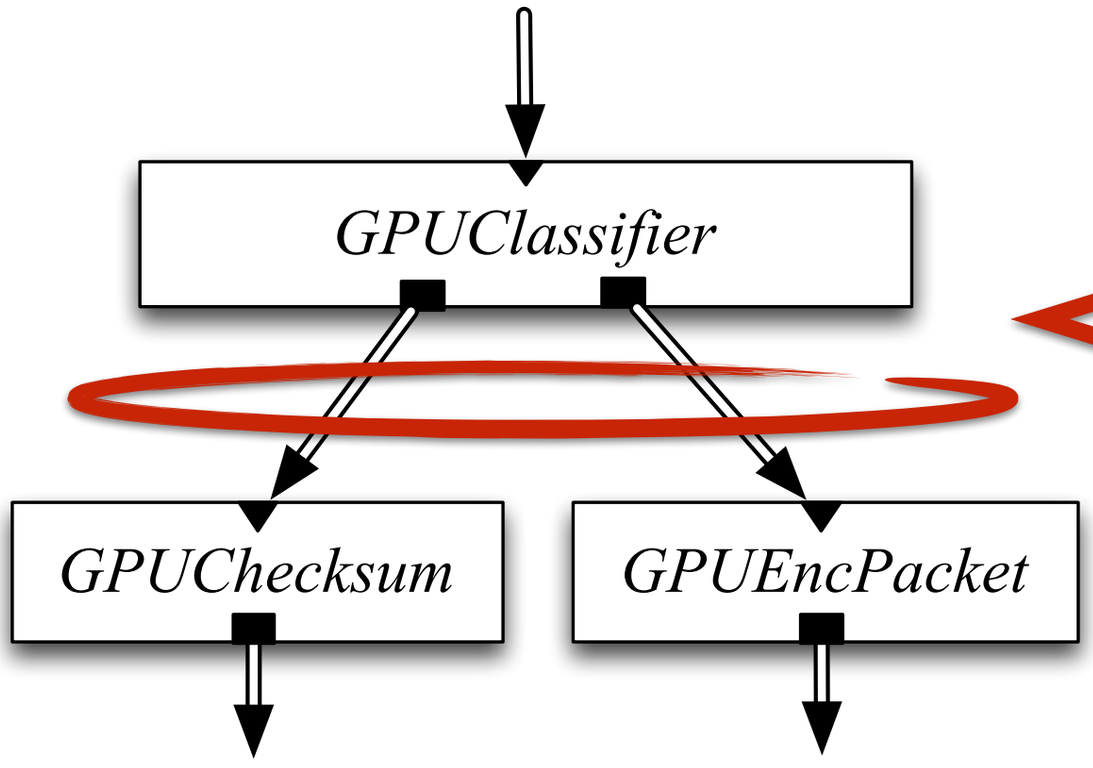
- Packets in a batch go to different paths



Two choices:

Batched packet divergence

- Packets in a batch go to different paths

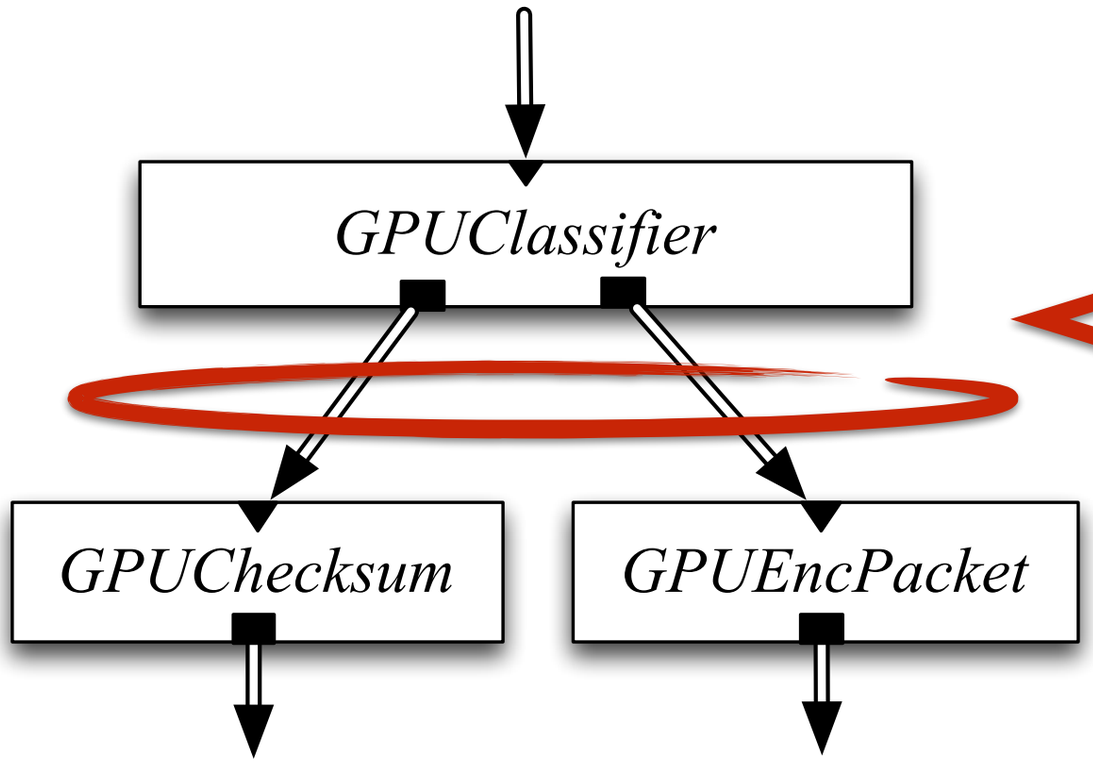


Two choices:

1. Split the batch into two copies

Batched packet divergence

- Packets in a batch go to different paths

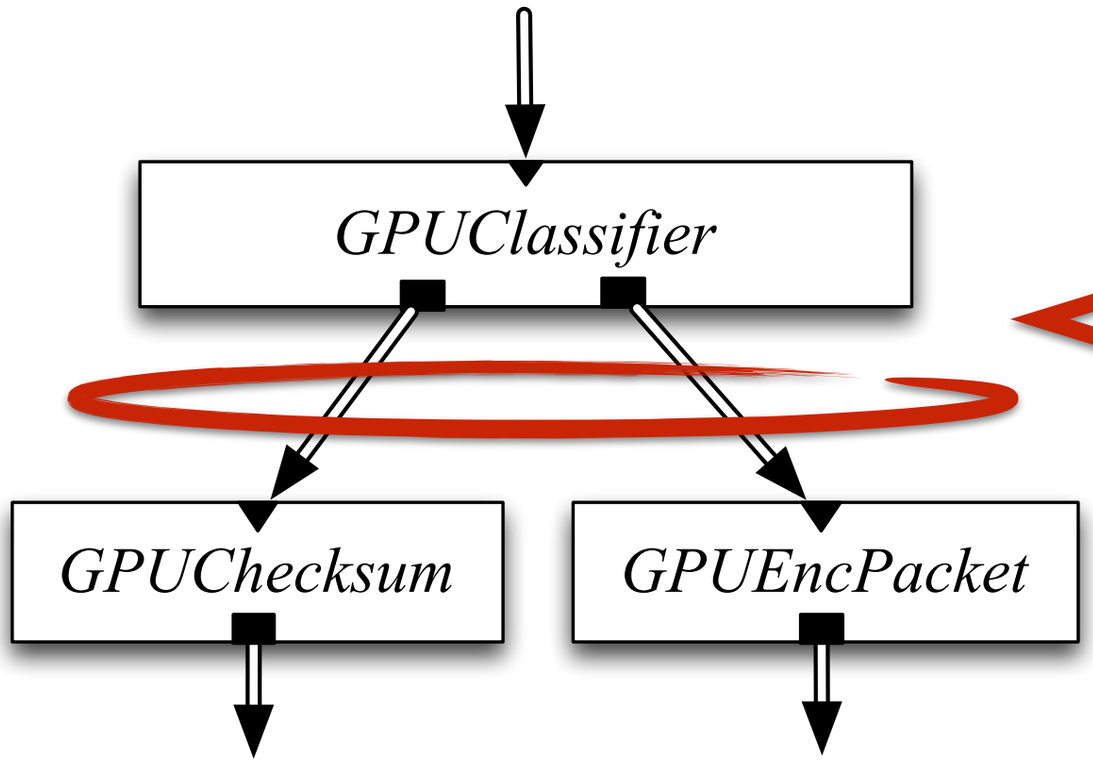


Two choices:

1. Split the batch into two copies
   - ➡ needs PCIe copy, sync!

Batched packet divergence

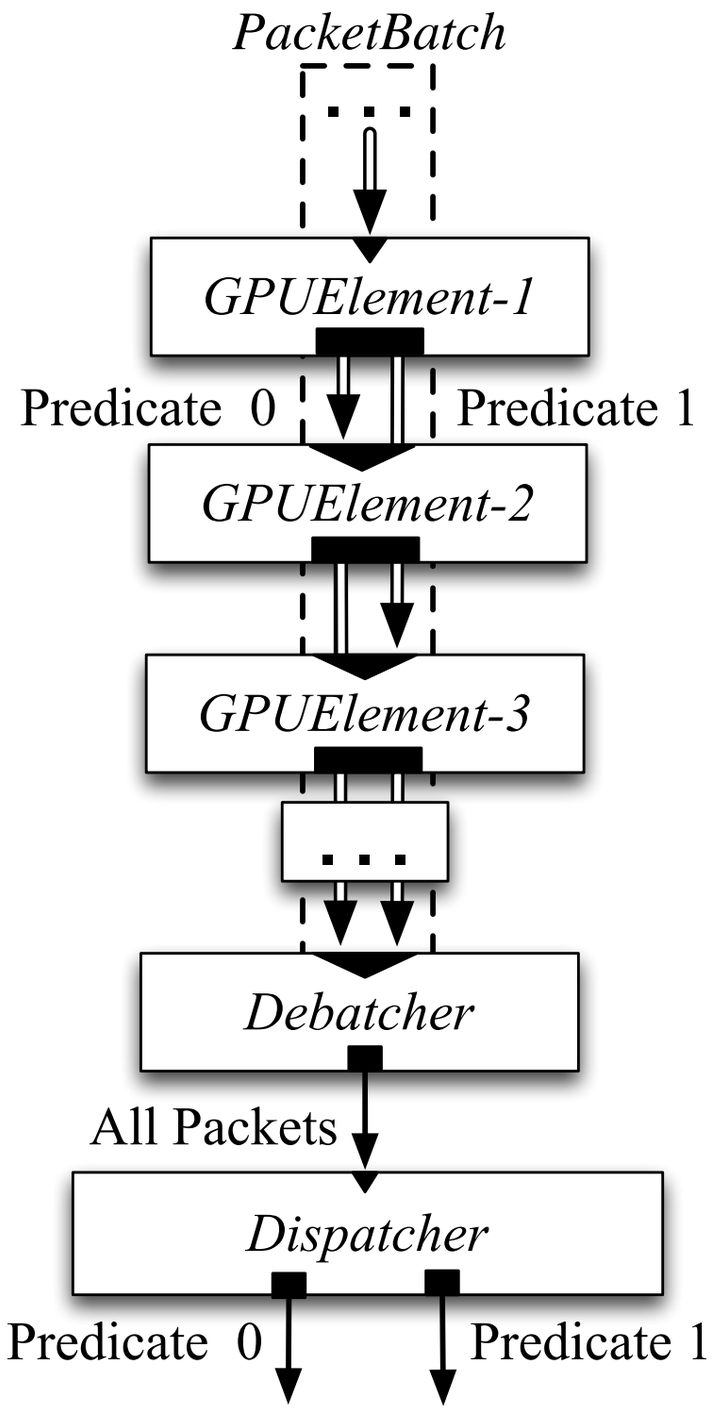
- Packets in a batch go to different paths



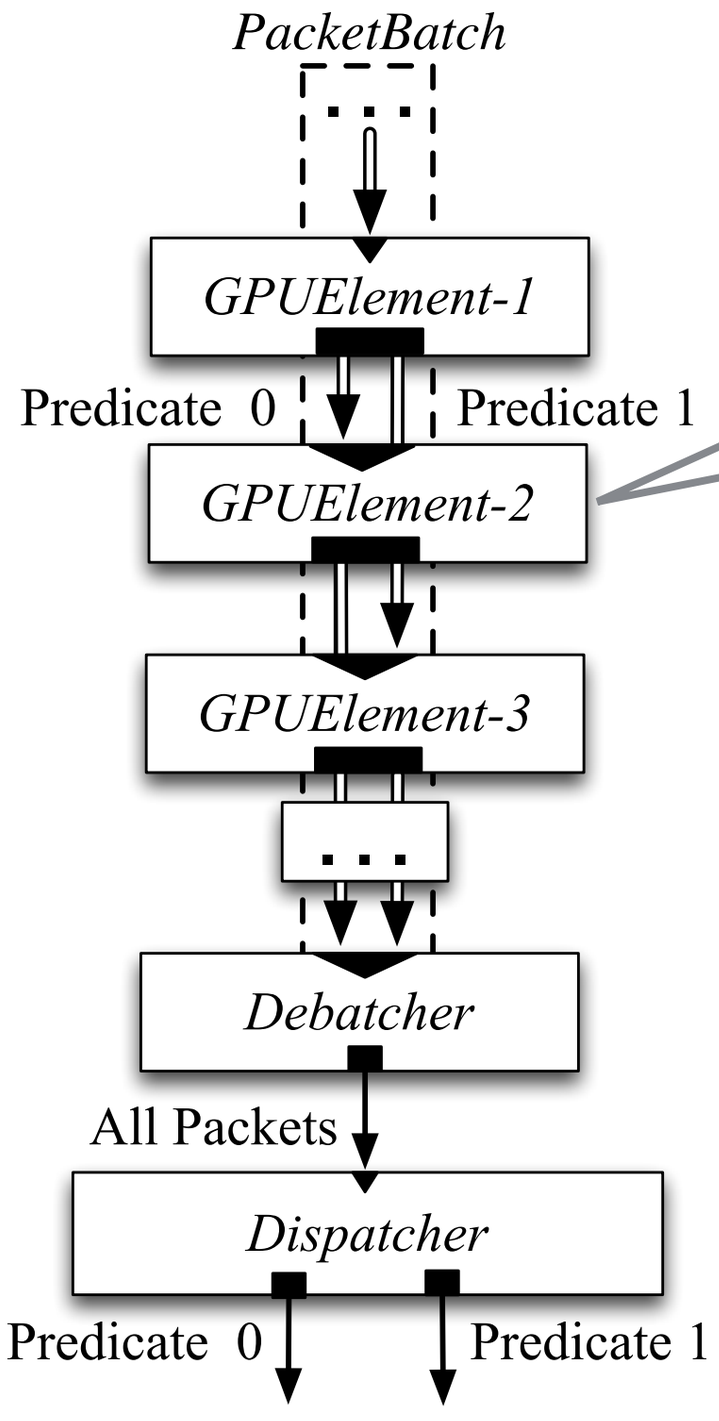
Two choices:

1. Split the batch into two copies
   ➡ needs PCIe copy, sync!
2. Keep the batch, skip some packets on GPUs

Predicated execution in a batch

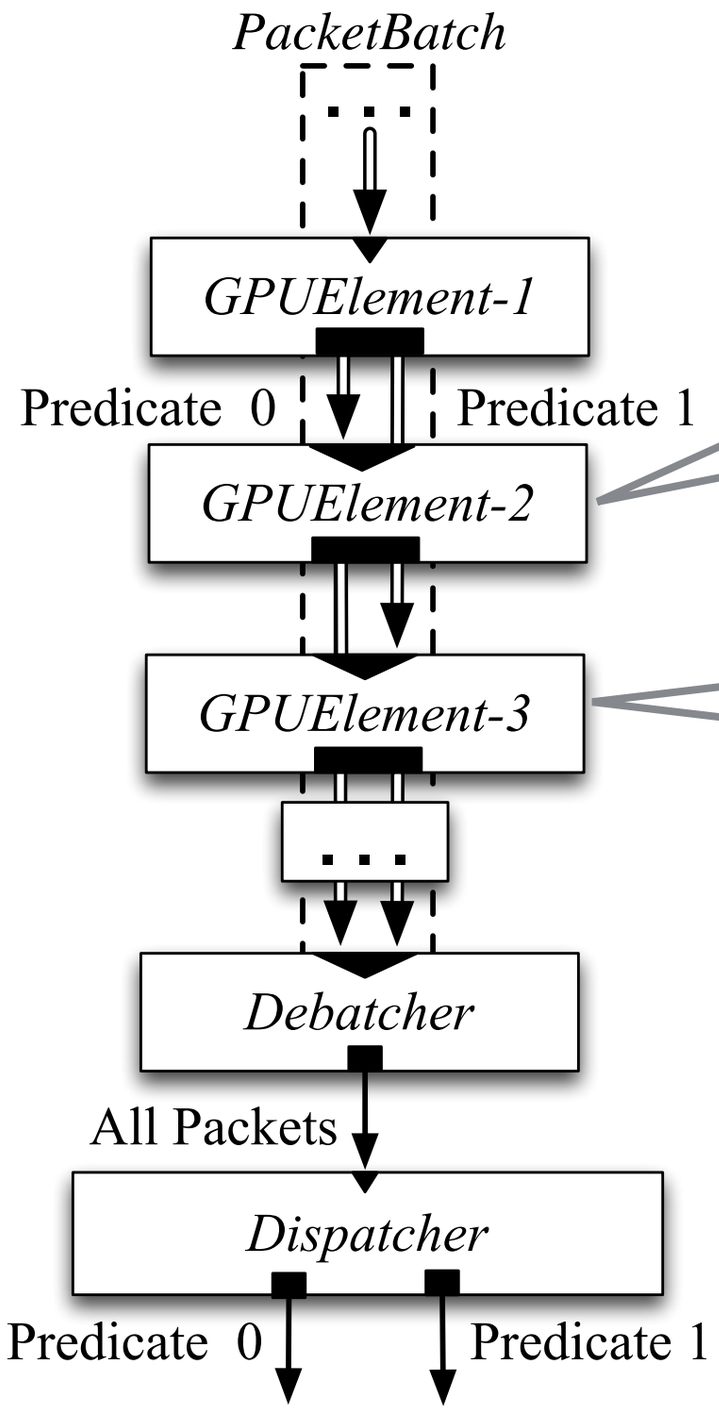


Predicated execution in a batch




```
void ge2kernel(…) {
  if (pkt.predicates[0]) {
    … // GPUElement-2 logic
  }
}
```

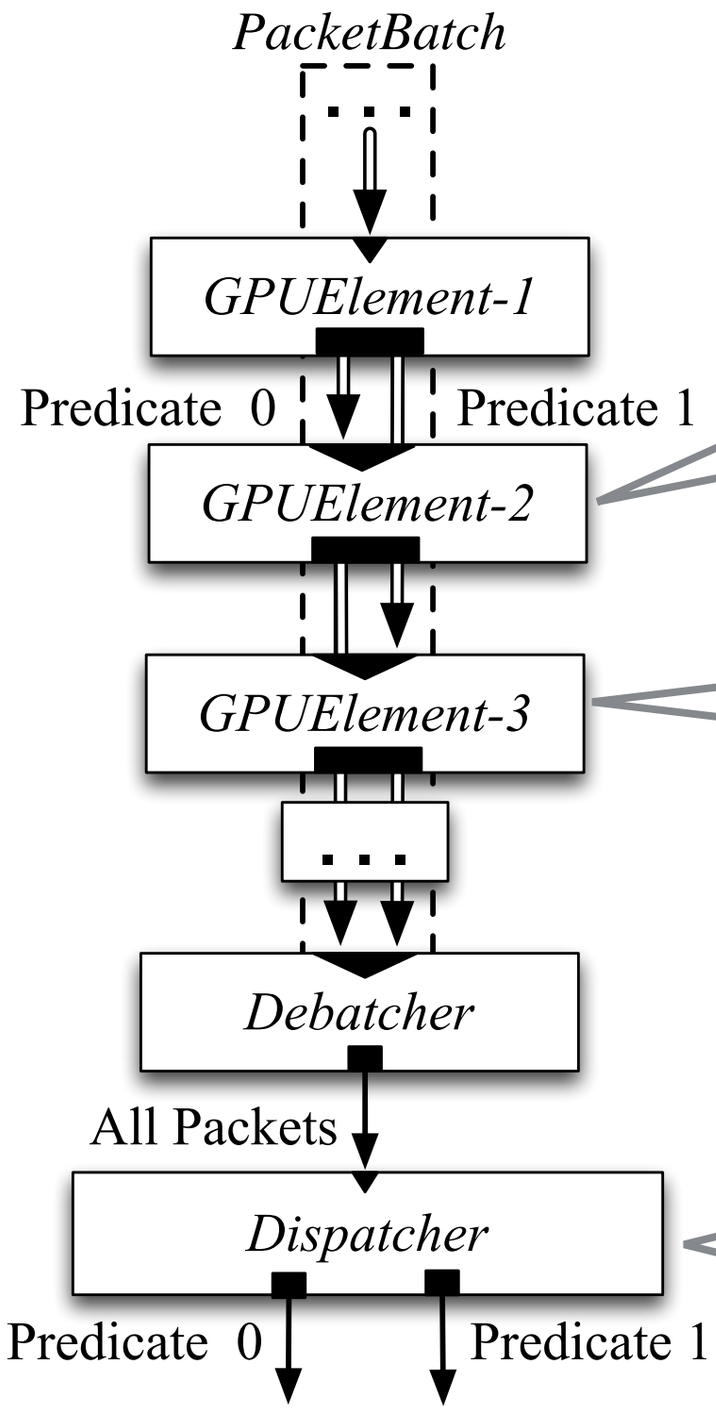
Predicated execution in a batch




```
void ge2kernel(…) {
  if (pkt.predicates[0]) {
    … // GPUElement-2 logic
  }
}
```

```
void ge3kernel(…) {
  if (pkt.predicates[1]) {
    … // GPUElement-3 logic
  }
}
```

Predicated execution in a batch




```
void ge2kernel(…) {
  if (pkt.predicates[0]) {
    … // GPUElement-2 logic
  }
}
```

```
void ge3kernel(…) {
  if (pkt.predicates[1]) {
    … // GPUElement-3 logic
  }
}
```

```
if (pkt.predicates[0])
  output(0).push(pkt);
else if (pkt.predicates[1])
  output(1).push(pkt);
…
```

Apply generic principles

- Batched processing
- Asynchronous non-blocking GPU programming
- Reduce PCIe data transfer with compacted workload
- All use the same locked memory for zero-copy DMA

Apply generic principles

- Asynchronous non-blocking GPU programming
- Reduce PCIe data transfer with compacted workload
- All use the same locked memory for zero-copy DMA

Apply generic principles

- Asynchronous non-blocking GPU programming
- Reduce PCIe data transfer with compacted workload
- All use the same locked memory for zero-copy DMA

Apply generic principles

- Asynchronous non-blocking GPU programming
  - Each batch binds a CUDA stream
- Reduce PCIe data transfer with compacted workload
- All use the same locked memory for zero-copy DMA

Apply generic principles

- Asynchronous non-blocking GPU programming
  - Each batch binds a CUDA stream
  - BElement only does async GPU ops
- Reduce PCIe data transfer with compacted workload
- All use the same locked memory for zero-copy DMA

Apply generic principles

- Asynchronous non-blocking GPU programming
  - Each batch binds a CUDA stream
  - BElement only does async GPU ops
  - Use GPUCompletionQueue to check stream status
- Reduce PCIe data transfer with compacted workload
- All use the same locked memory for zero-copy DMA

Apply generic principles

- Reduce PCIe data transfer with compacted workload

- All use the same locked memory for zero-copy DMA

Apply generic principles

- Reduce PCIe data transfer with compacted workload
  - "Region-of-interest" (ROI) based packet slicing
- All use the same locked memory for zero-copy DMA

Packet slicing for ``use-partial”

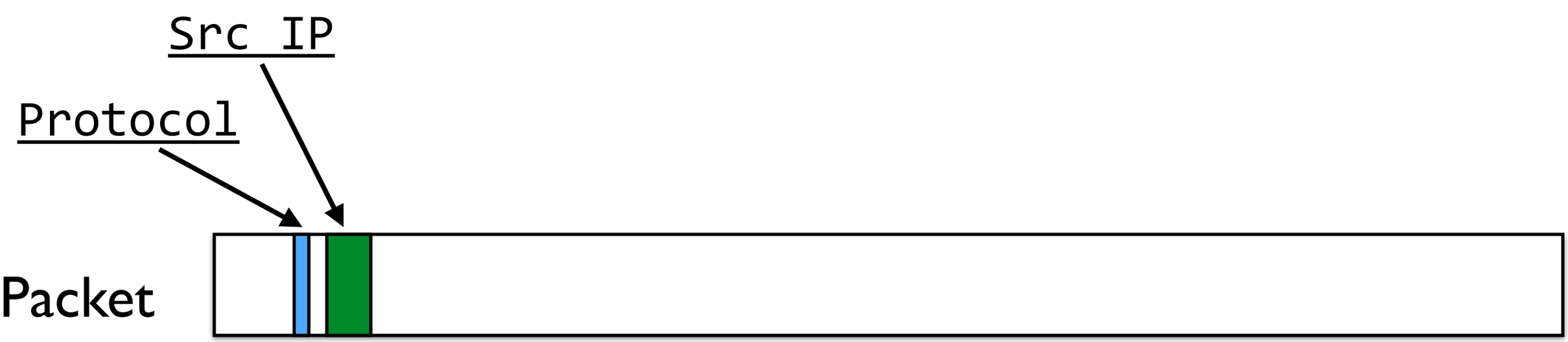
Packet Classification’s regions-of-interest (ROI):

Packet slicing for ``use-partial”

Packet Classification’s regions-of-interest (ROI):

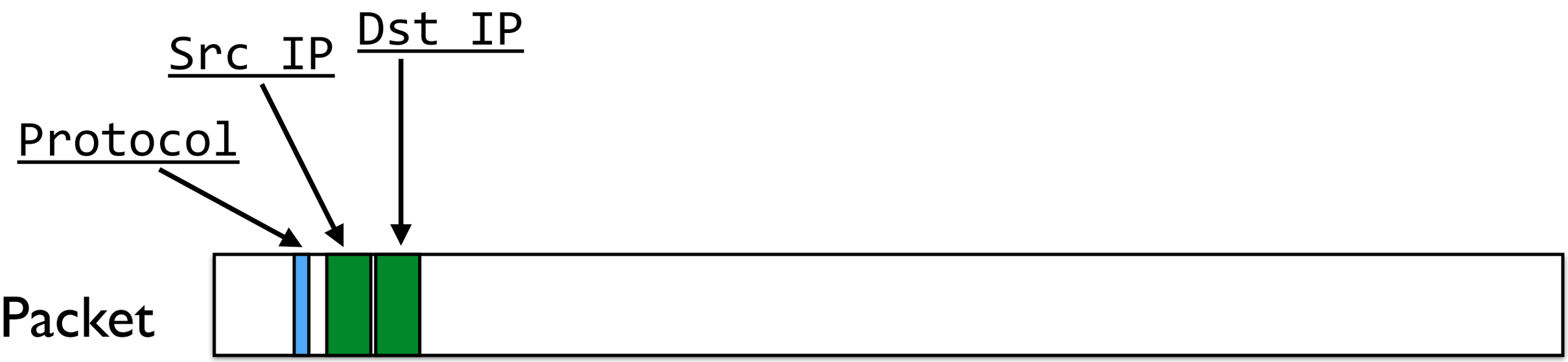
Packet slicing for ``use-partial”

Packet Classification’s regions-of-interest (ROI):



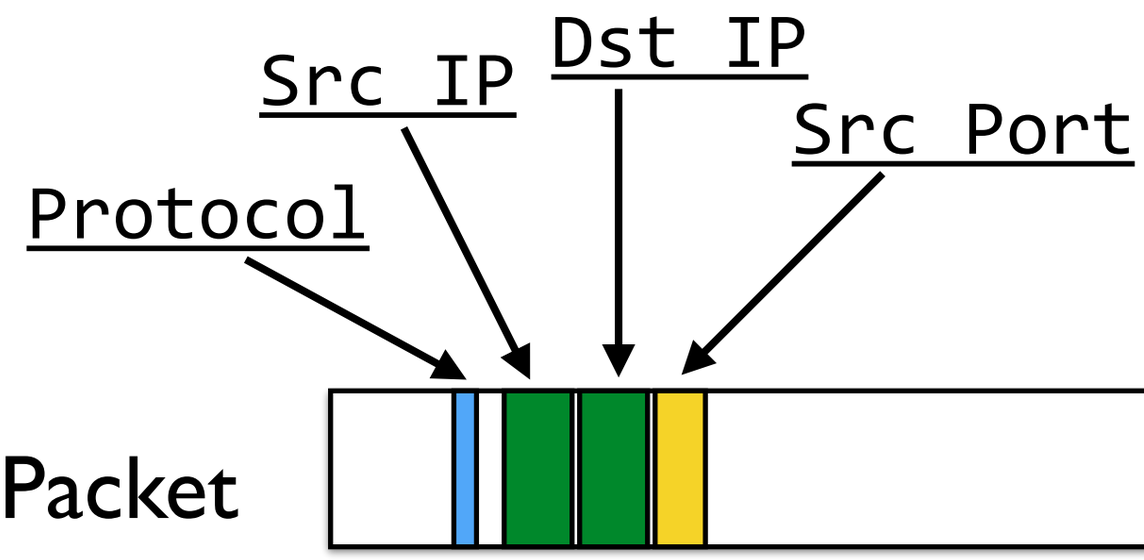
Packet slicing for ``use-partial”

Packet Classification’s regions-of-interest (ROI):



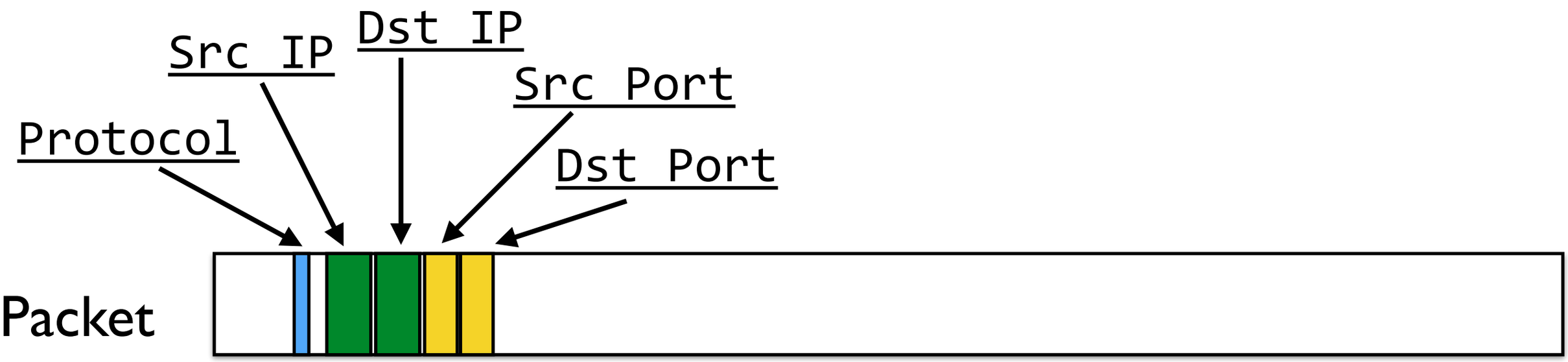
Packet slicing for ``use-partial”

Packet Classification’s regions-of-interest (ROI):



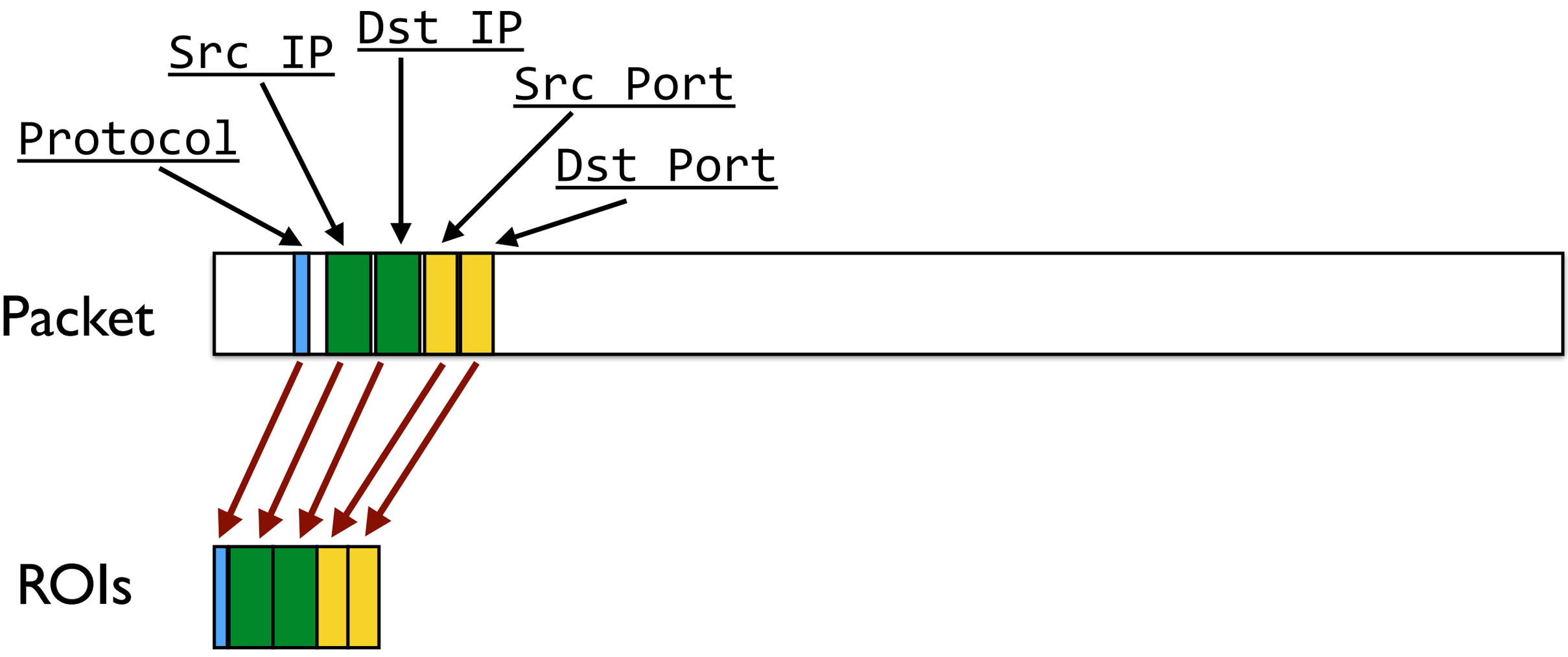
Packet slicing for ``use-partial”

Packet Classification’s regions-of-interest (ROI):



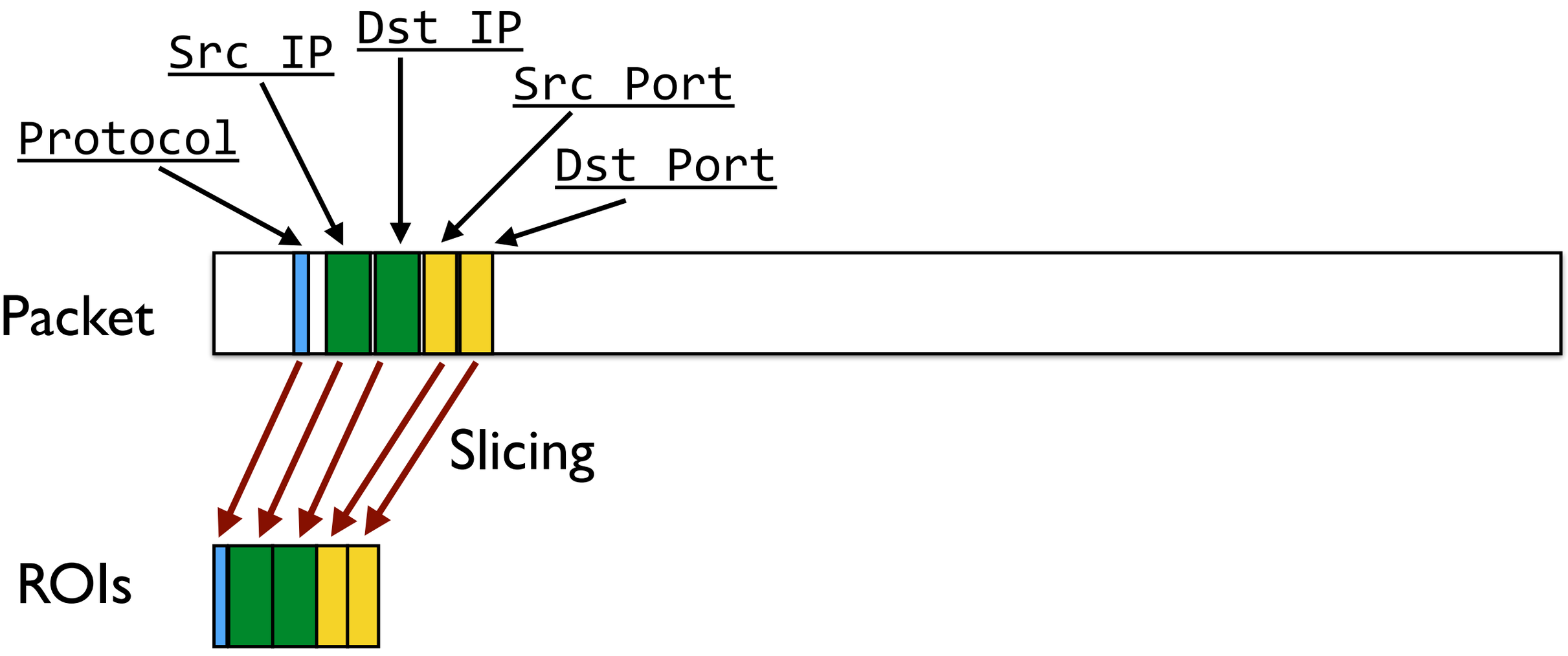
Packet slicing for ``use-partial”

Packet Classification’s regions-of-interest (ROI):



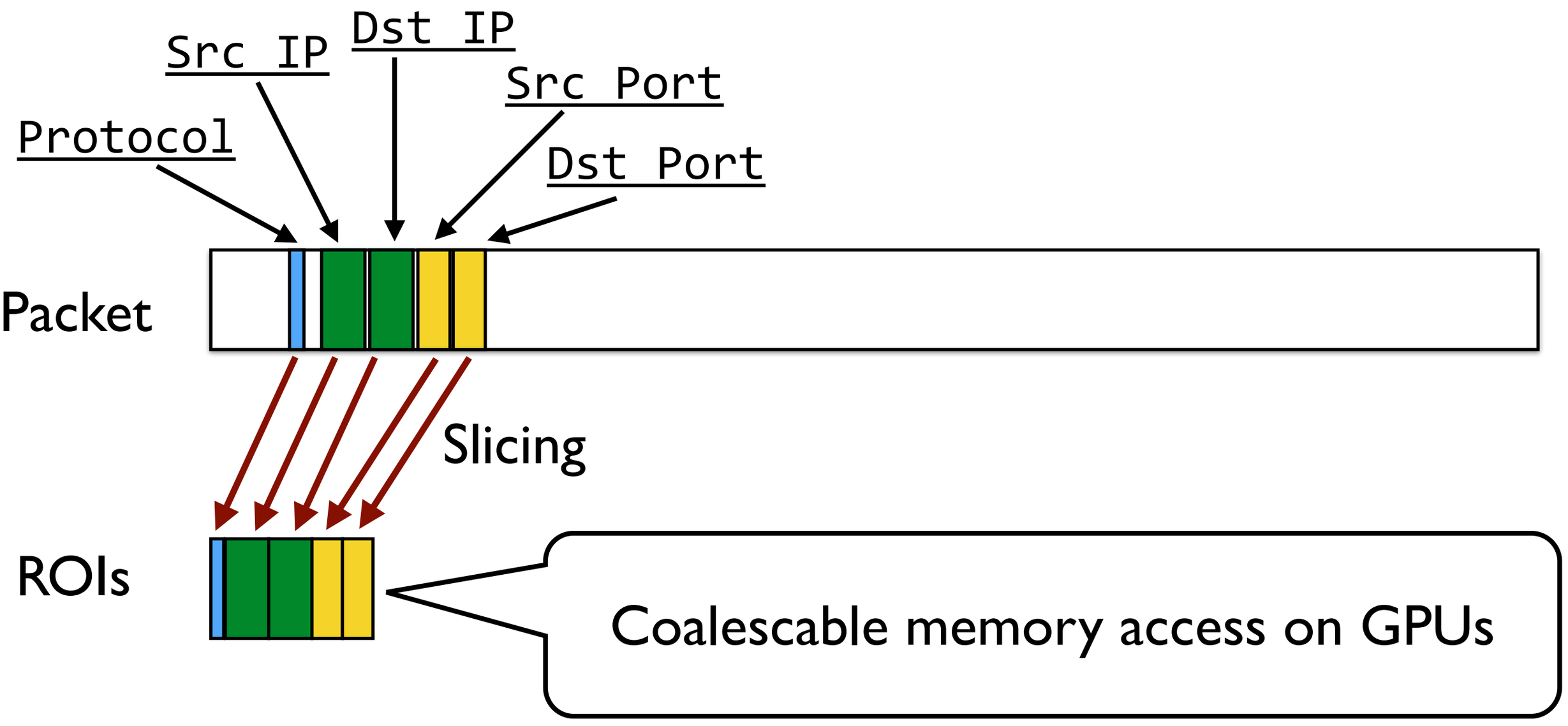
Packet slicing for ``use-partial”

Packet Classification’s regions-of-interest (ROI):



Packet slicing for ``use-partial”

Packet Classification’s regions-of-interest (ROI):



Apply generic principles

- Batched processing
- Asynchronous non-blocking GPU programming
- Reduce PCIe data transfer with compacted workload
- All use the same locked memory for zero-copy DMA

Apply generic principles

- All use the same locked memory for zero-copy DMA

Apply generic principles

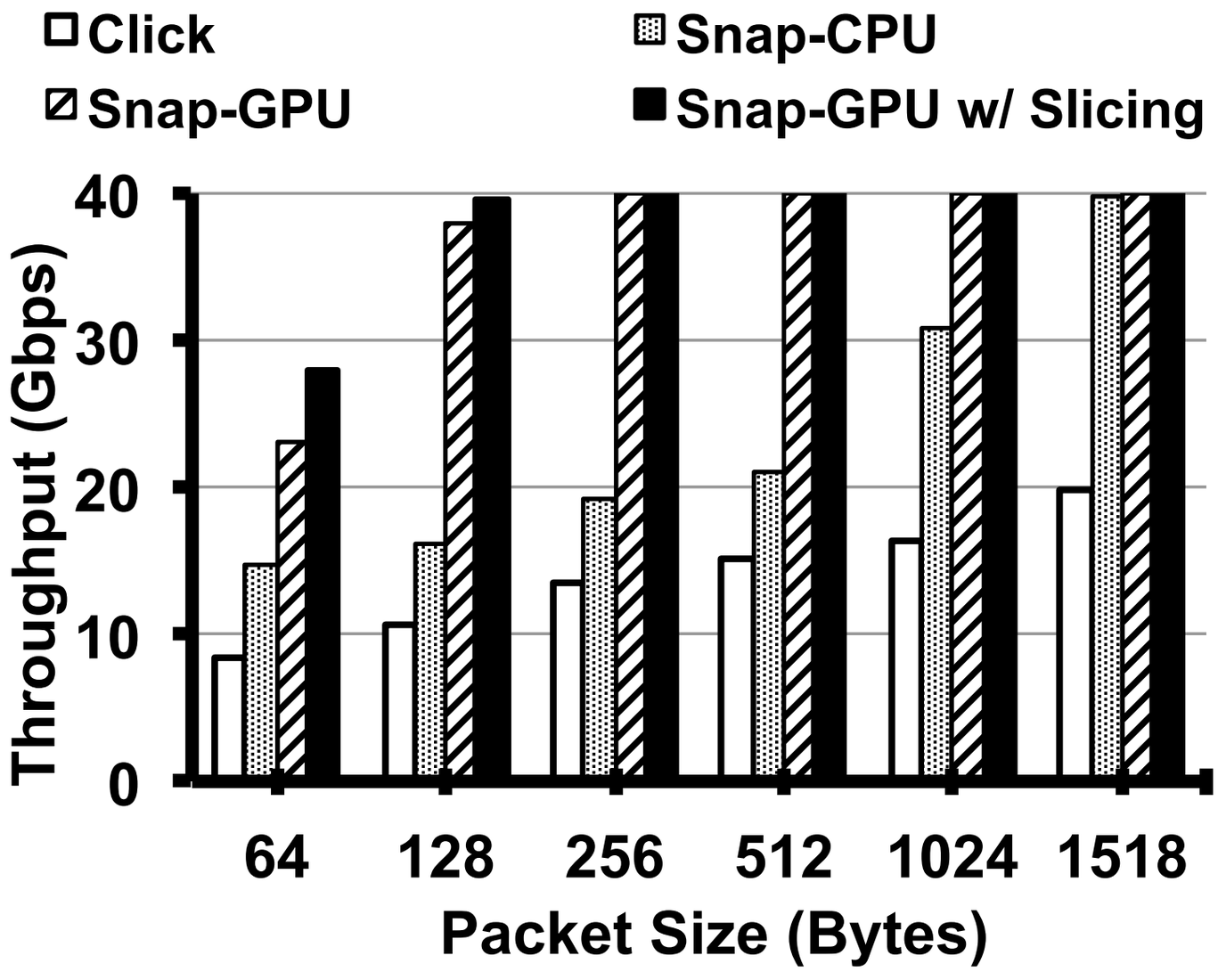
- All use the same locked memory for zero-copy DMA

- ROIs stored in CUDA locked memory

Evaluation

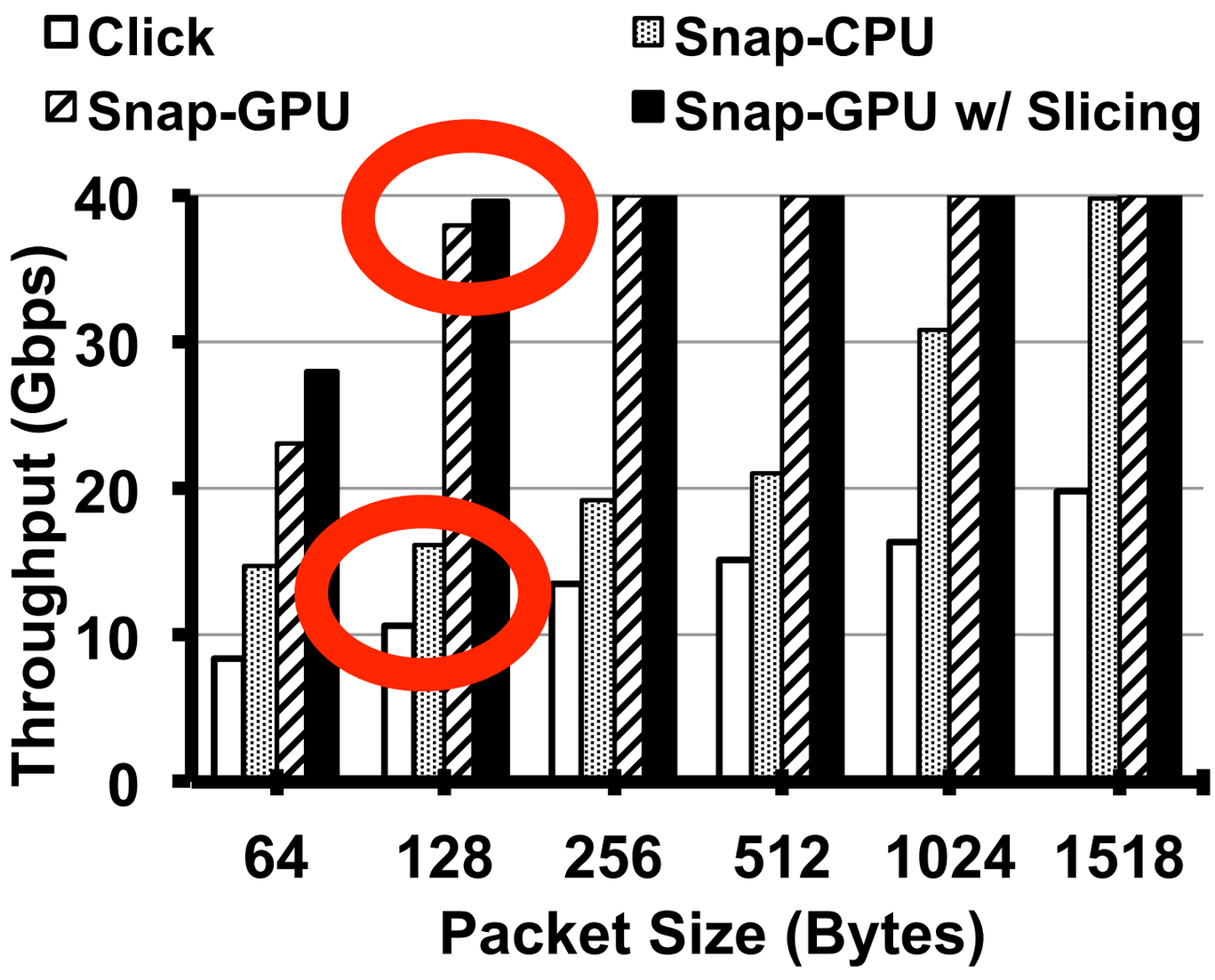
- Basic forwarding I/O improvement
- Example applications:
  - Classification
    - IP routing
    - Pattern matching for IDS
- Latency, re-order
- Flexibility and modularity

Classifier+routing+pattern matching



(c) IDS Router

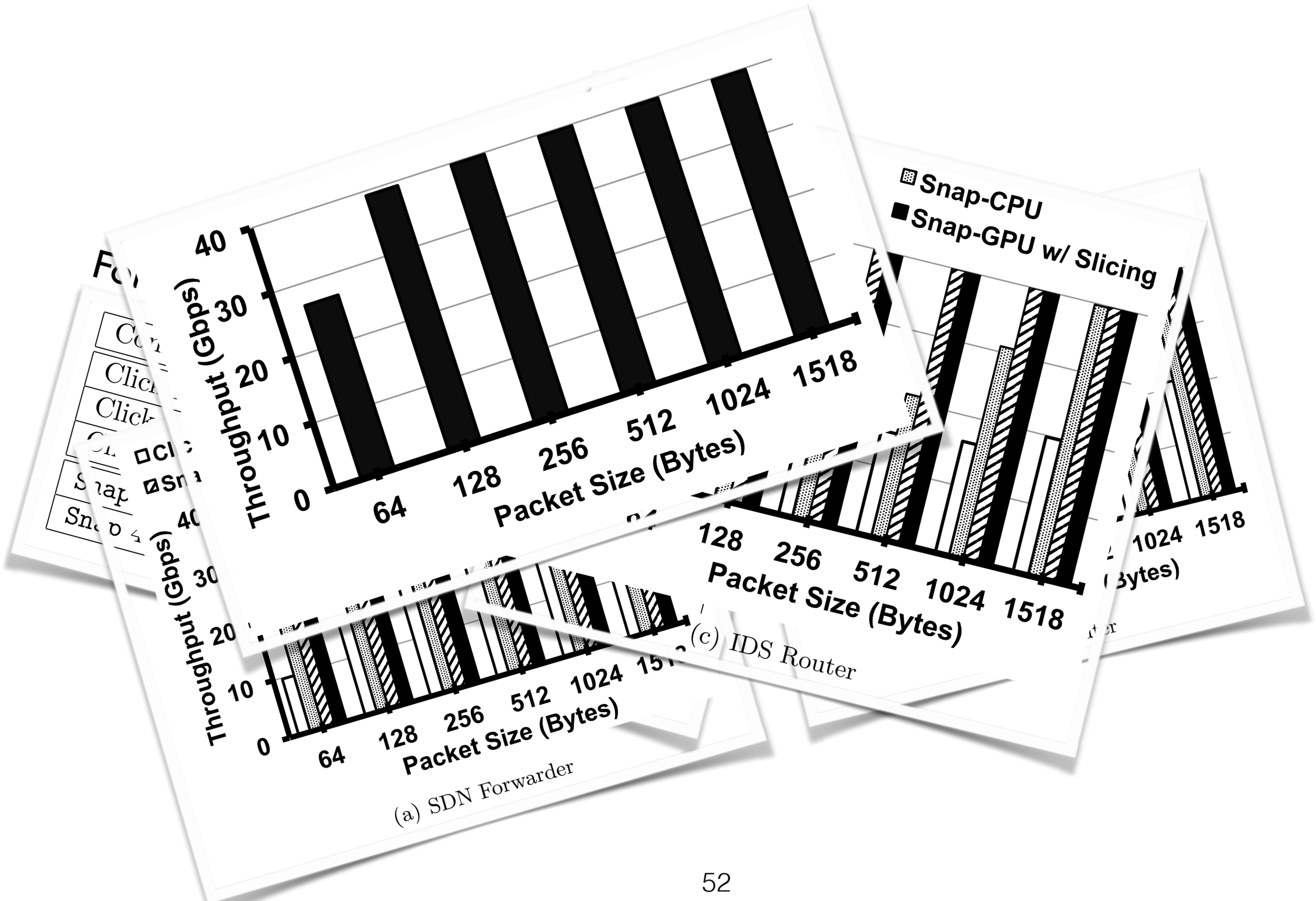
Classifier+routing+pattern matching



(c) IDS Router

Evaluation

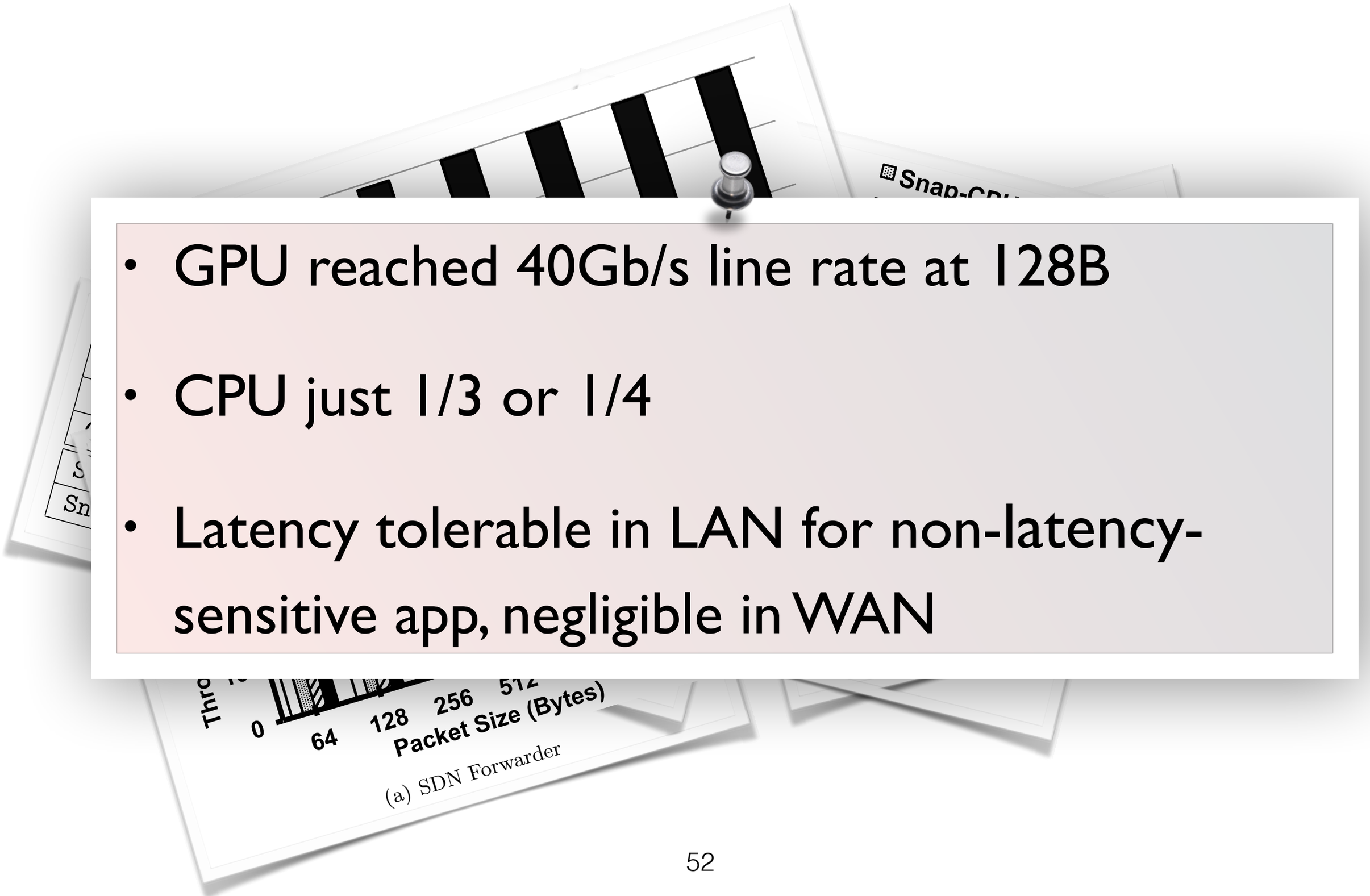
Evaluation



(a) SDN Forwarder

(c) IDS Router

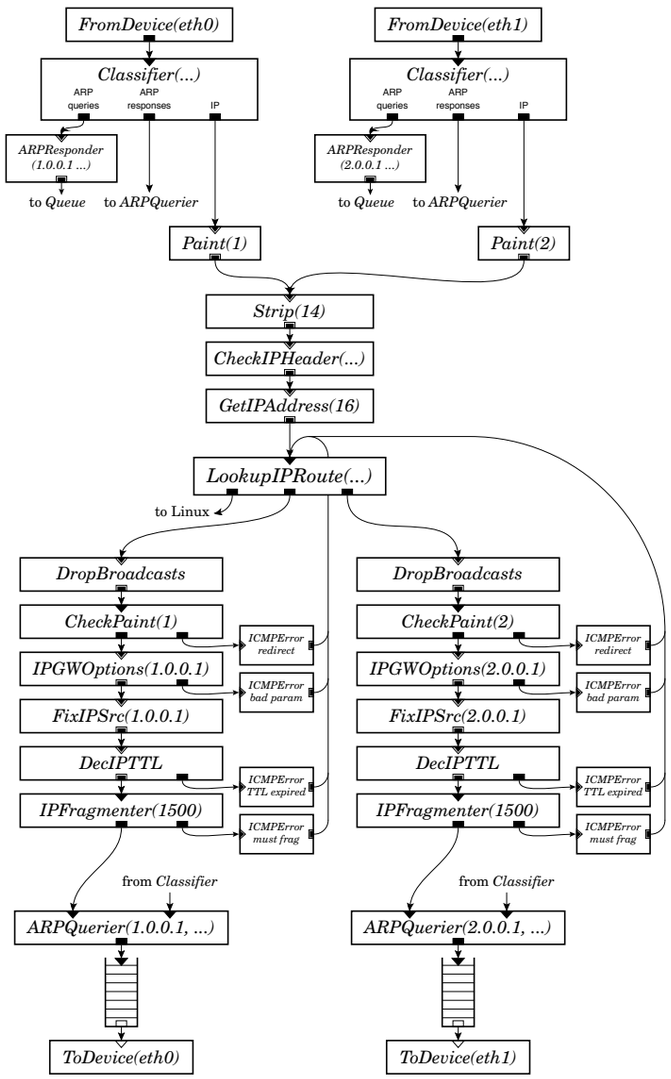
Evaluation



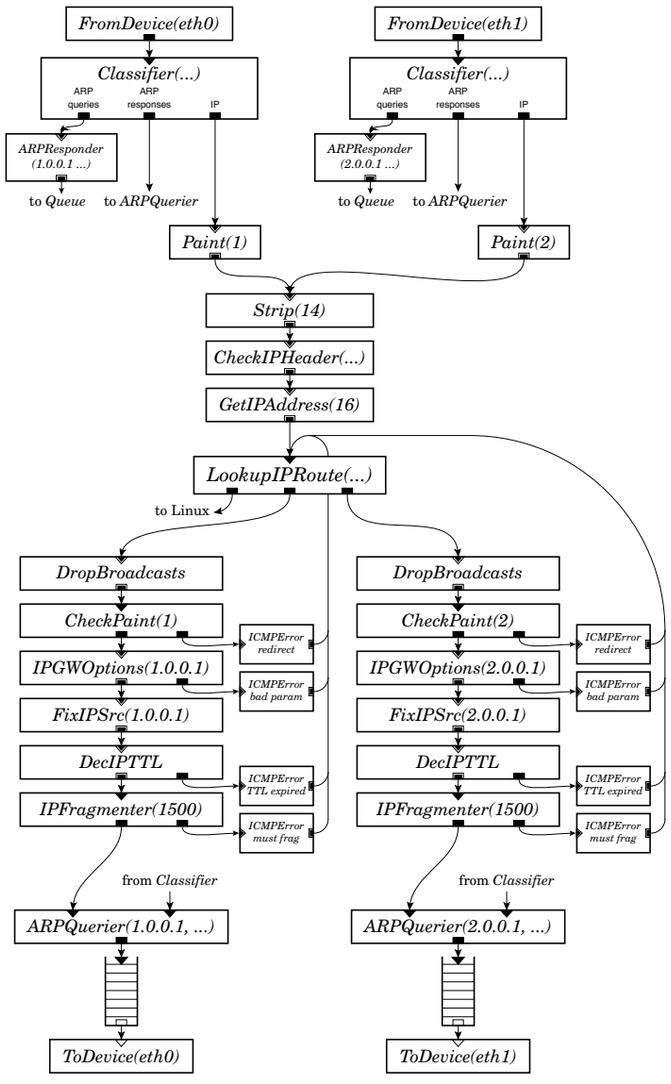
(a) SDN Forwarder

- GPU reached 40Gb/s line rate at 128B
- CPU just 1/3 or 1/4
- Latency tolerable in LAN for non-latency-sensitive app, negligible in WAN

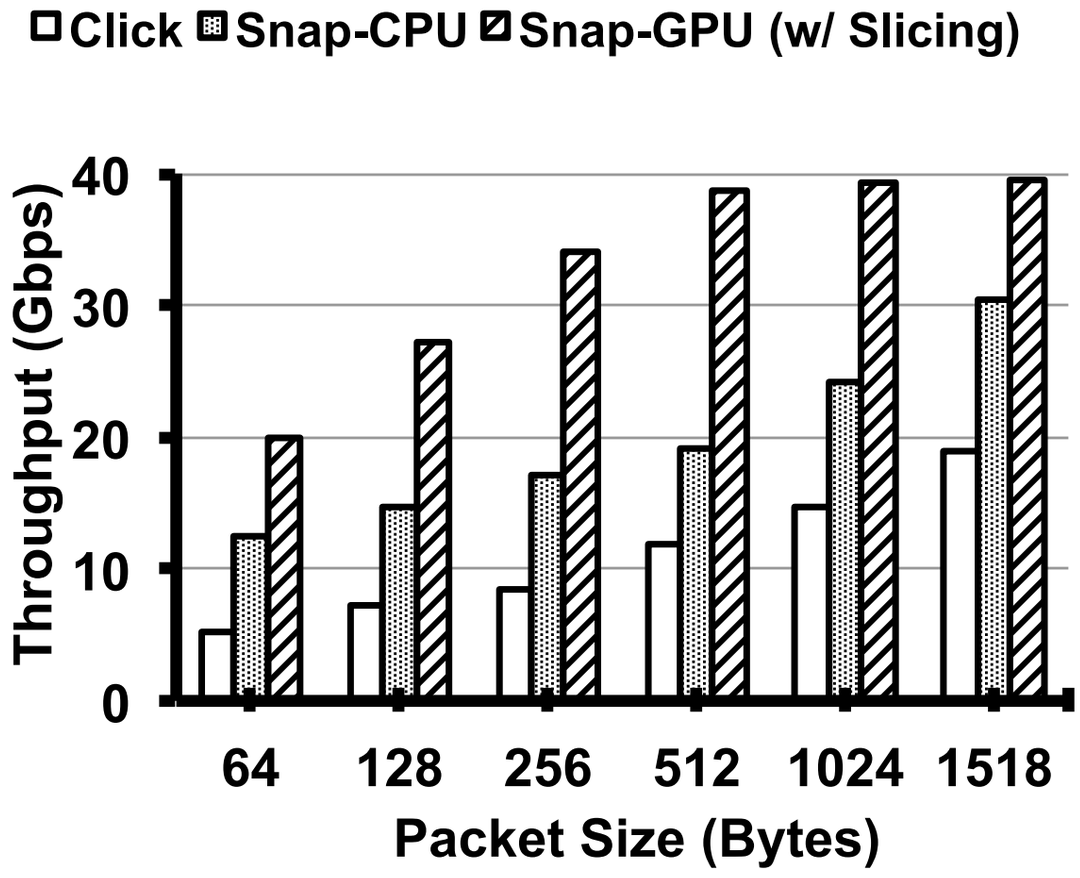
Preserving Click's flexibility: full IP router



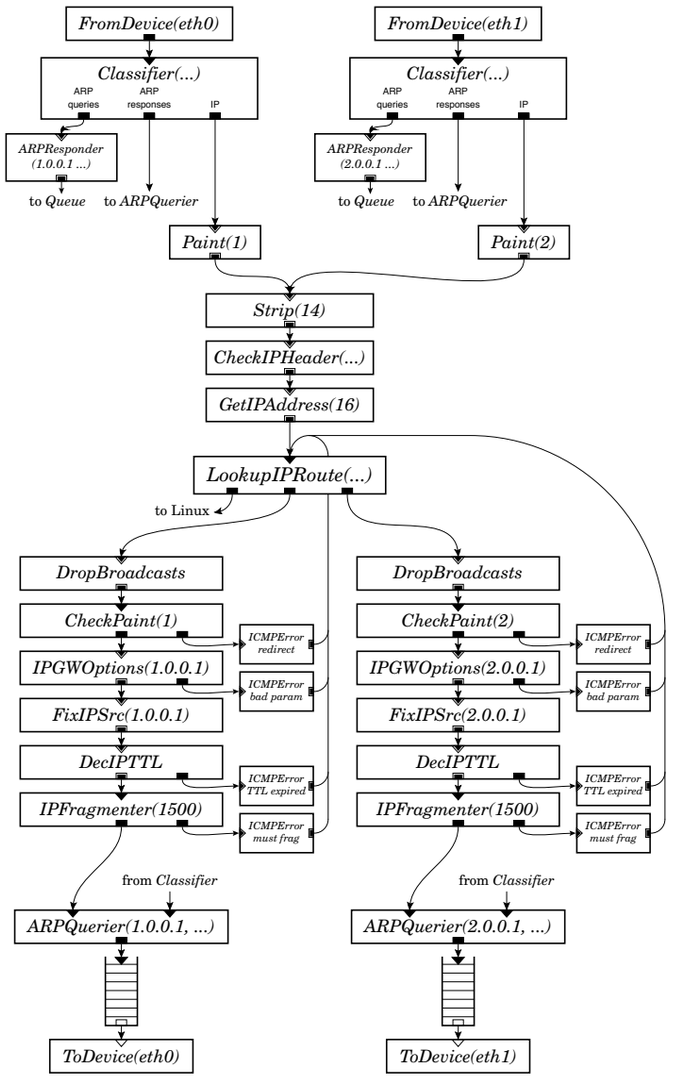
Preserving Click's flexibility: full IP router



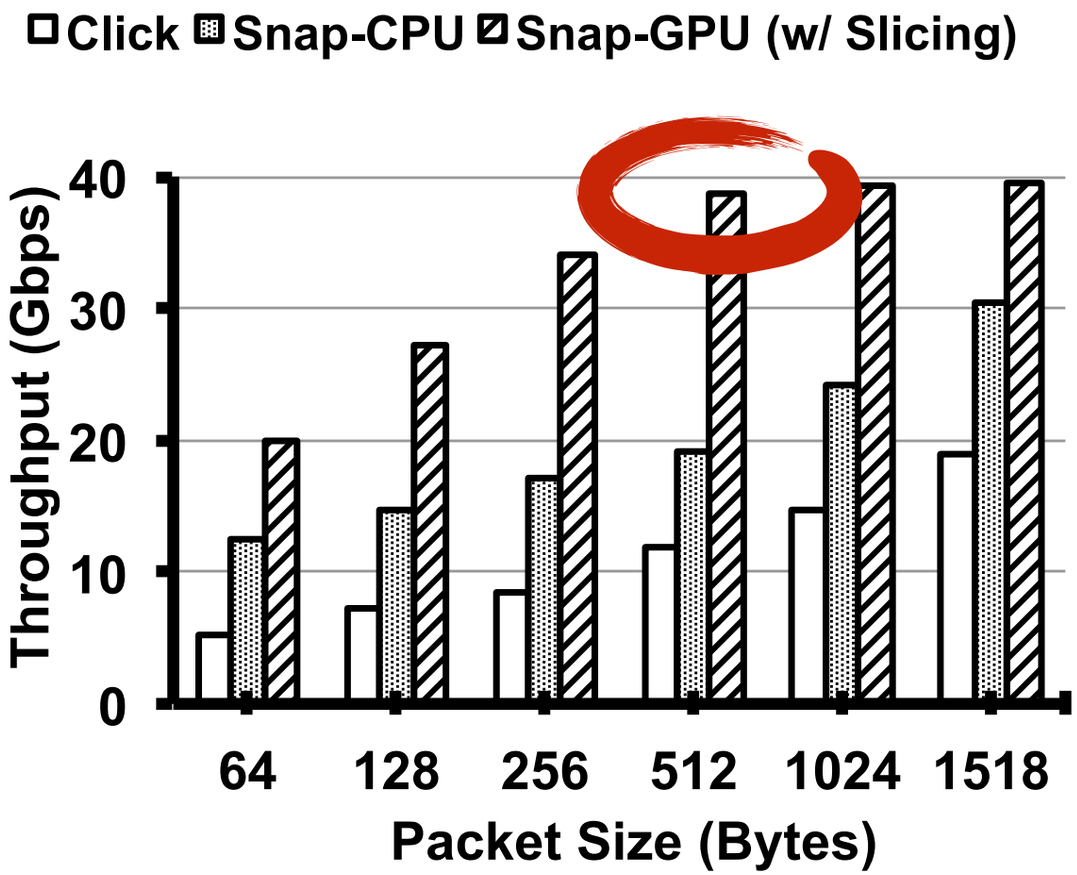
Add a pattern matching element



Preserving Click's flexibility: full IP router



Add a pattern matching element



Snap Summary

- A generic parallel packet processing framework
  - flexibility of Click
  - fast parallel power from GPUs
- https://github.com/wbsun/snap
- Details not presented: network I/O, async scheduling

Thesis statement

The throughput of system software with parallelizable, computationally expensive tasks can be improved by using GPUs and frameworks with memory-efficient and throughput-oriented designs.

Conclusion

Conclusion

Generic principles:

- Batched processing
- Asynchronous non-blocking GPU programming
- Reduce PCIe data transfer with compacted workload
- All use the same locked memory for zero-copy DMA

Conclusion

Generic principles:

- Batched processing
- Asynchronous non-blocking GPU programming
- Reduce PCIe data transfer with compacted workload
- All use the same locked memory for zero-copy DMA

Concrete frameworks:

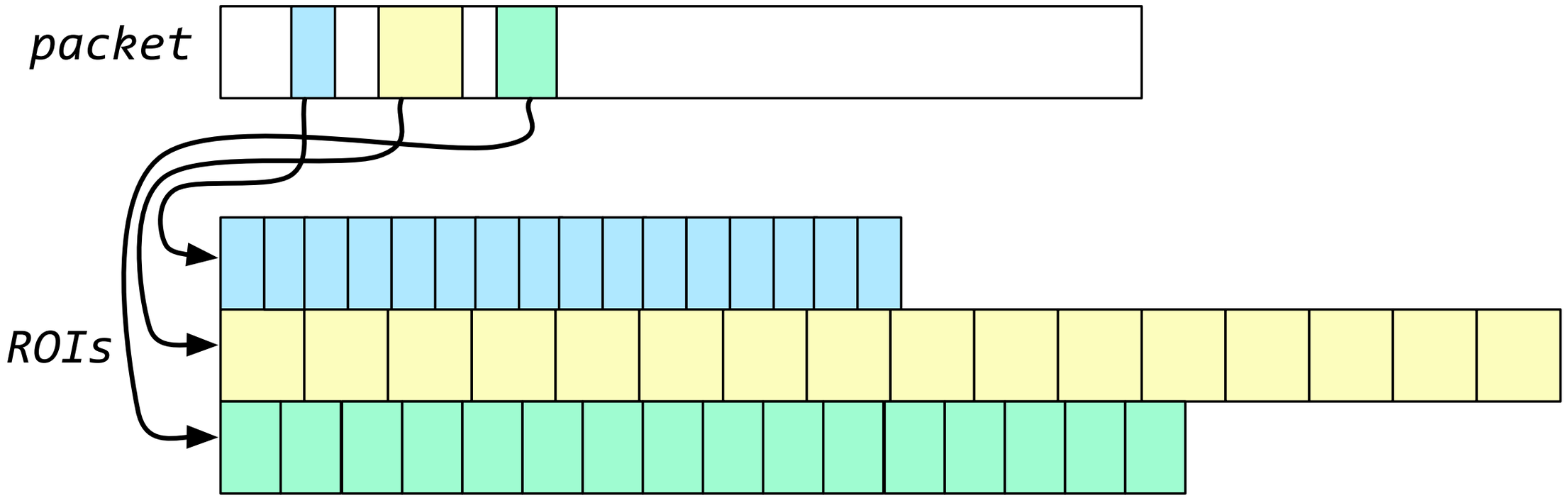
- GPUstore with high throughput storage applications
- Snap with high throughput network packet processing

Thanks!

Q&A

Backup slides

ROI for memory access coalescing



How ROI slicing works?

- To Click insiders:
  - Batcher accepts ROI requests from BElements
  - Batcher merges requested ROIs into result ROIs
  - Each BElement asks for its ROIs' offsets
  - GPU kernels invoked by BElements use variables for offsets

How predicated execution works?

- To Click insiders:
  - Manually assigned: where and what!

Path-encoded predicate