GPUstore: Harnessing GPU Computing for Storage Systems in the OS Kernel

Weibin Sun*, Robert Ricci*, Matthew L. Curry†

Presented by *Weibin Sun*

* University of Utah, Flux Group

† Sandia National Lab

Motivation

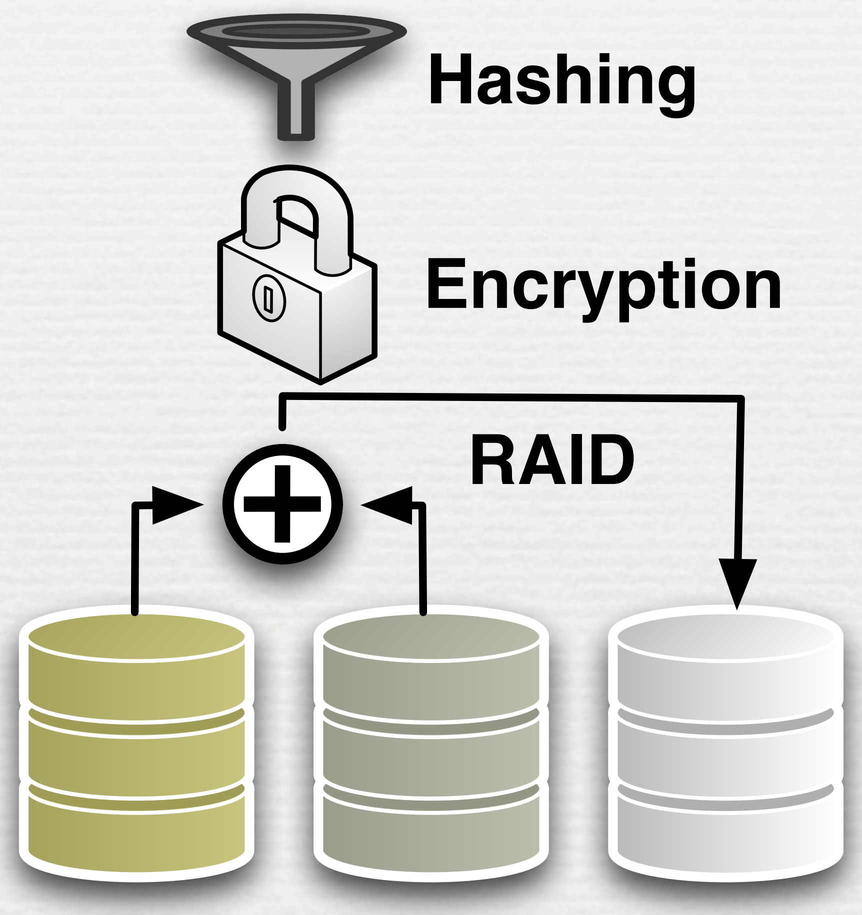
- Storage system performance decided by not only I/O, sometime but also computations
  - e.g. Intel X25-E SSD (256KB I/O size)

	Raw	**dm-crypt**
Read	~250MB/s	~103MB/s
Write	~170MB/s	~95MB/s

Motivation

- Storage system performance decided by not only I/O, sometime but also computations
  - e.g. Intel X25-E SSD (256KB I/O size)

	Raw	dm-crypt
Read	~250MB/s	~103MB/s
Write	~170MB/s	~95MB/s



Checksum, hashing, encryption, RAID, semantic fs search, in-kernel DB ...

Motivation

- Storage system performance decided by not only I/O, sometime but also computations
  - e.g. Intel X25-E SSD (256KB I/O size)

	Raw	dm-crypt
Read	~250MB/s	~103MB/s
Write	~170MB/s	~95MB/s

Checksum, hashing, encryption, RAID, semantic fs search, in-kernel DB ...

Speedup computations for storage

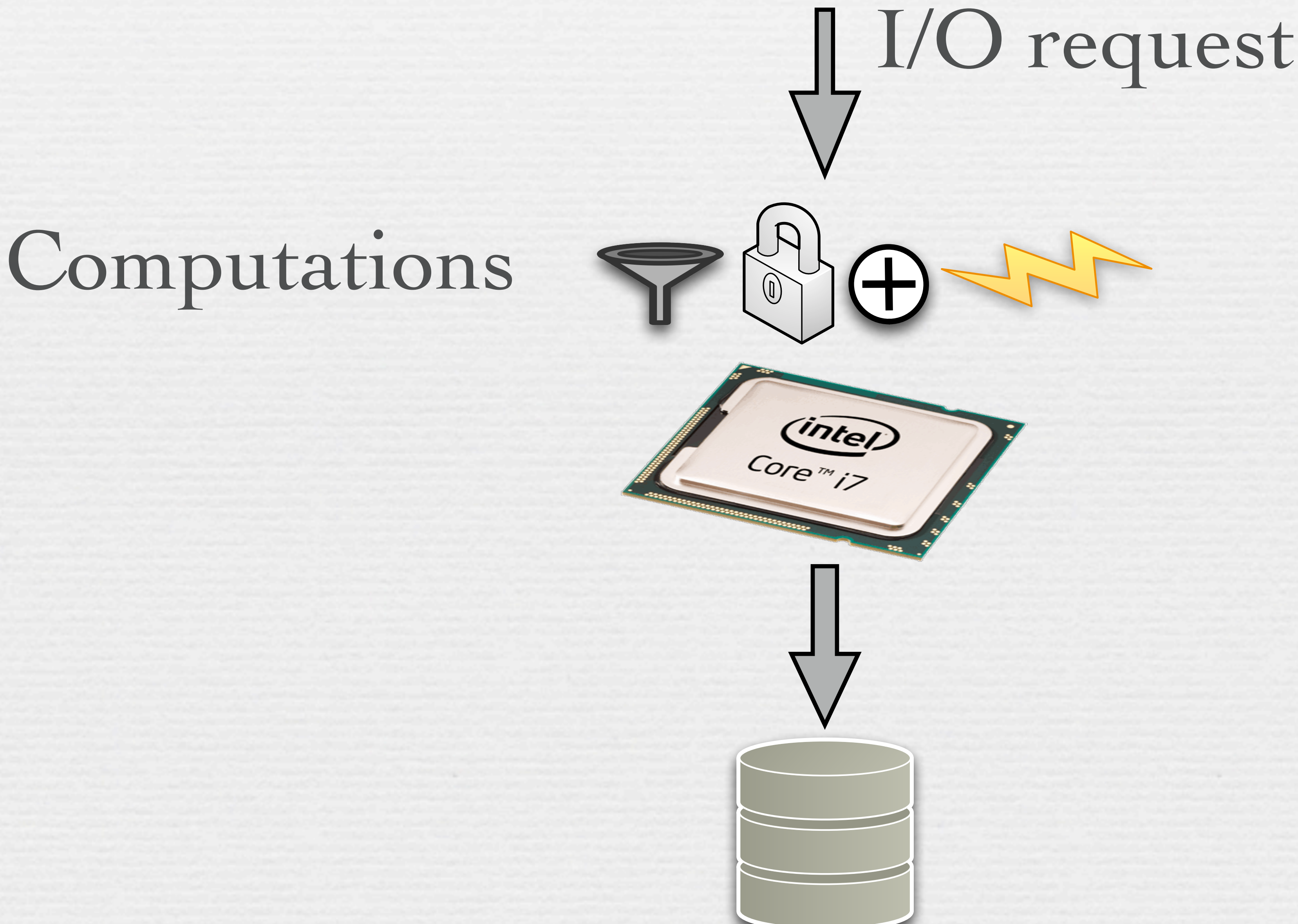
- Parallel at different scales
  - File, Block, Trunk, Sector, and Row
- Mostly same operations on different data

Speedup computations for storage

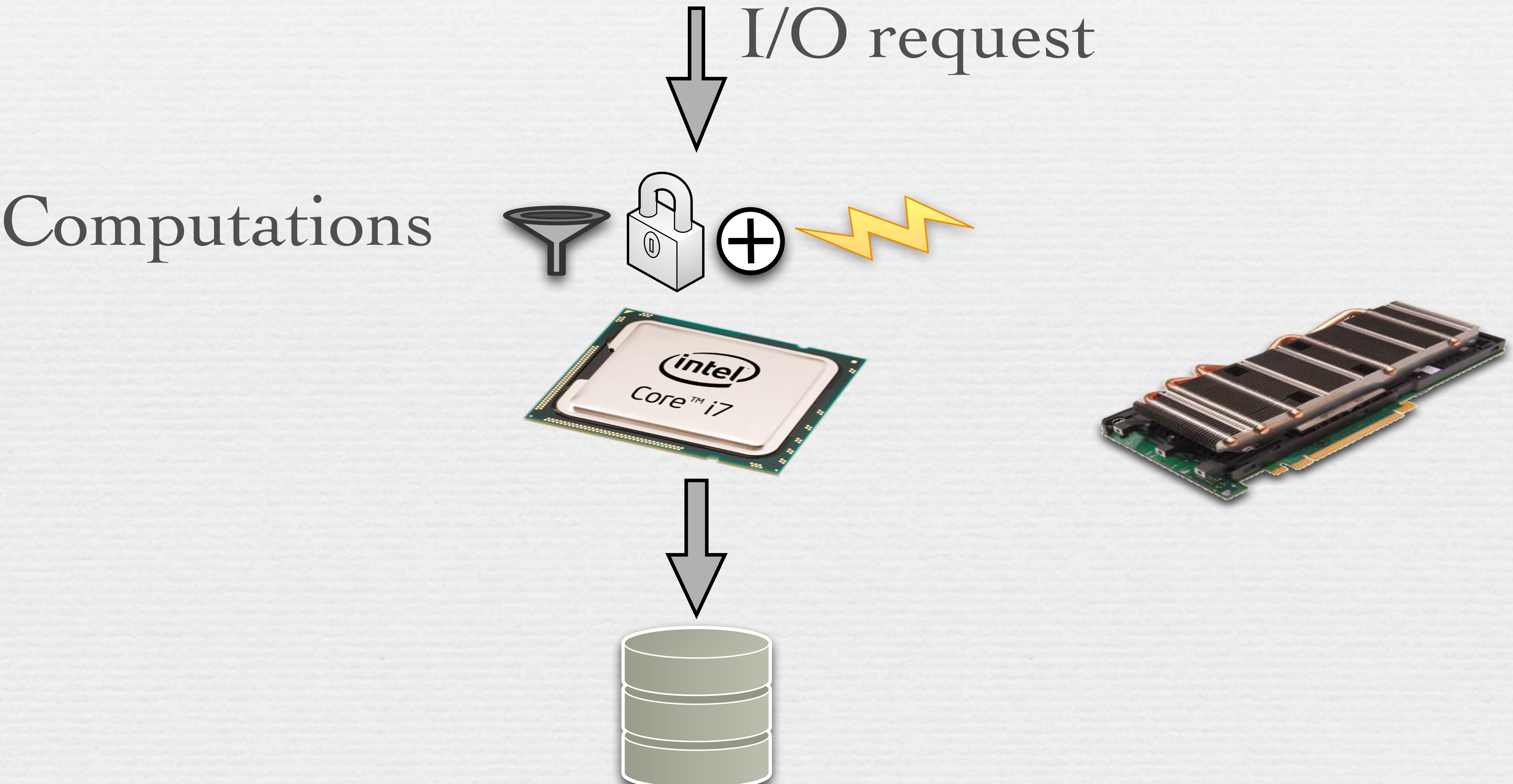
- Parallel at different scales
  - File, Block, Trunk, Sector, and Row
- Mostly same operations on different data

SIMD/STMD
GPU

The idea of GPUstore



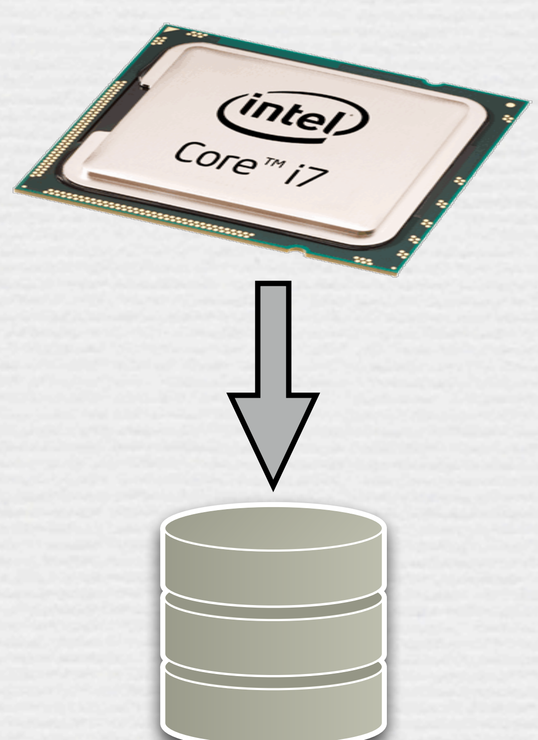
The idea of GPUstore

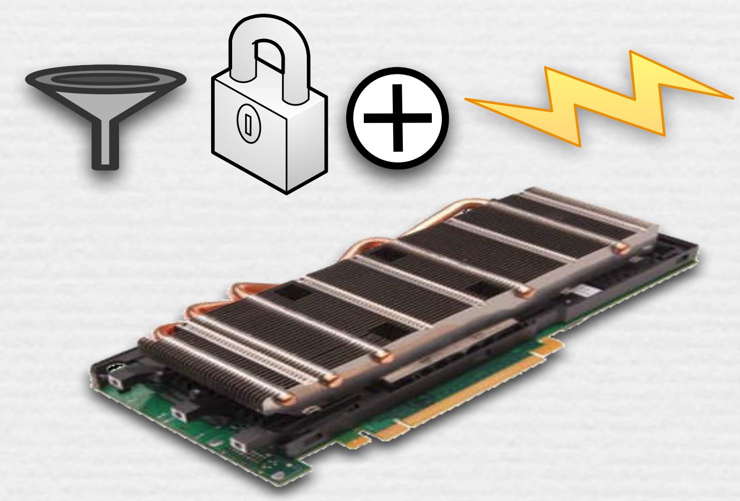


The idea of GPUstore

I/O request

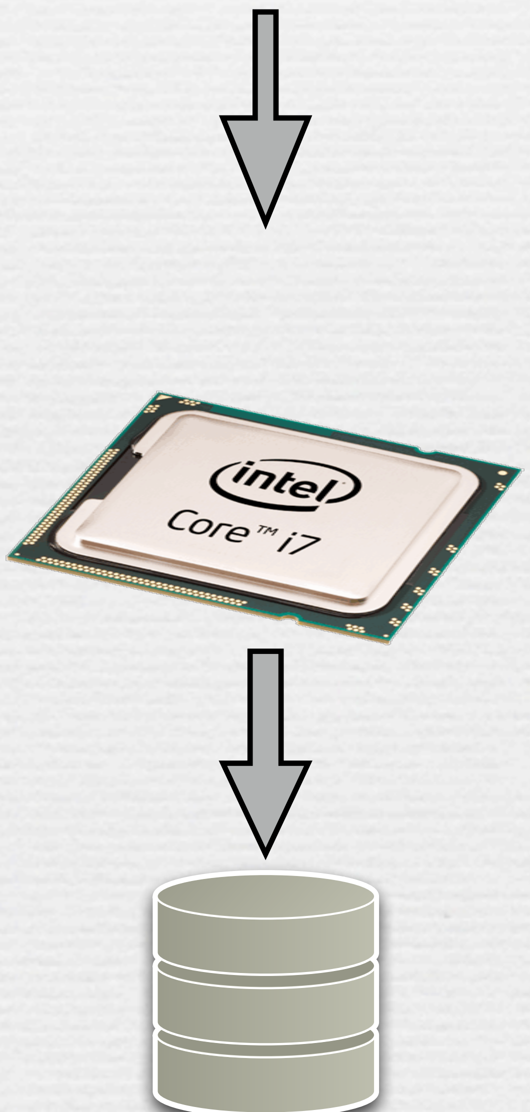
Computations



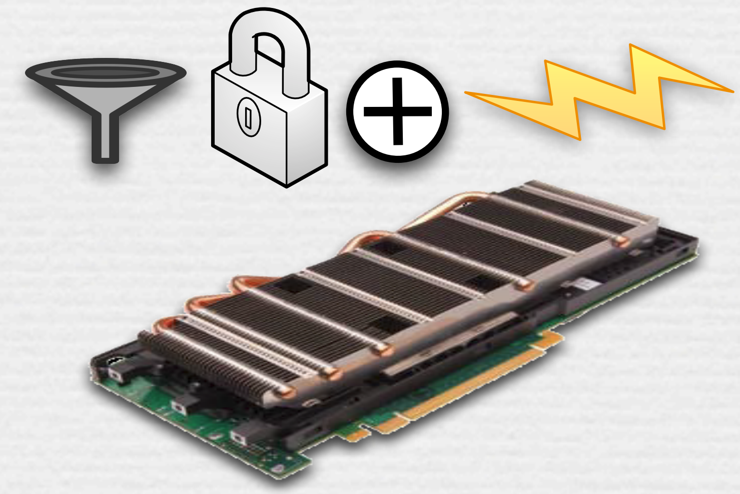


The idea of GPUstore

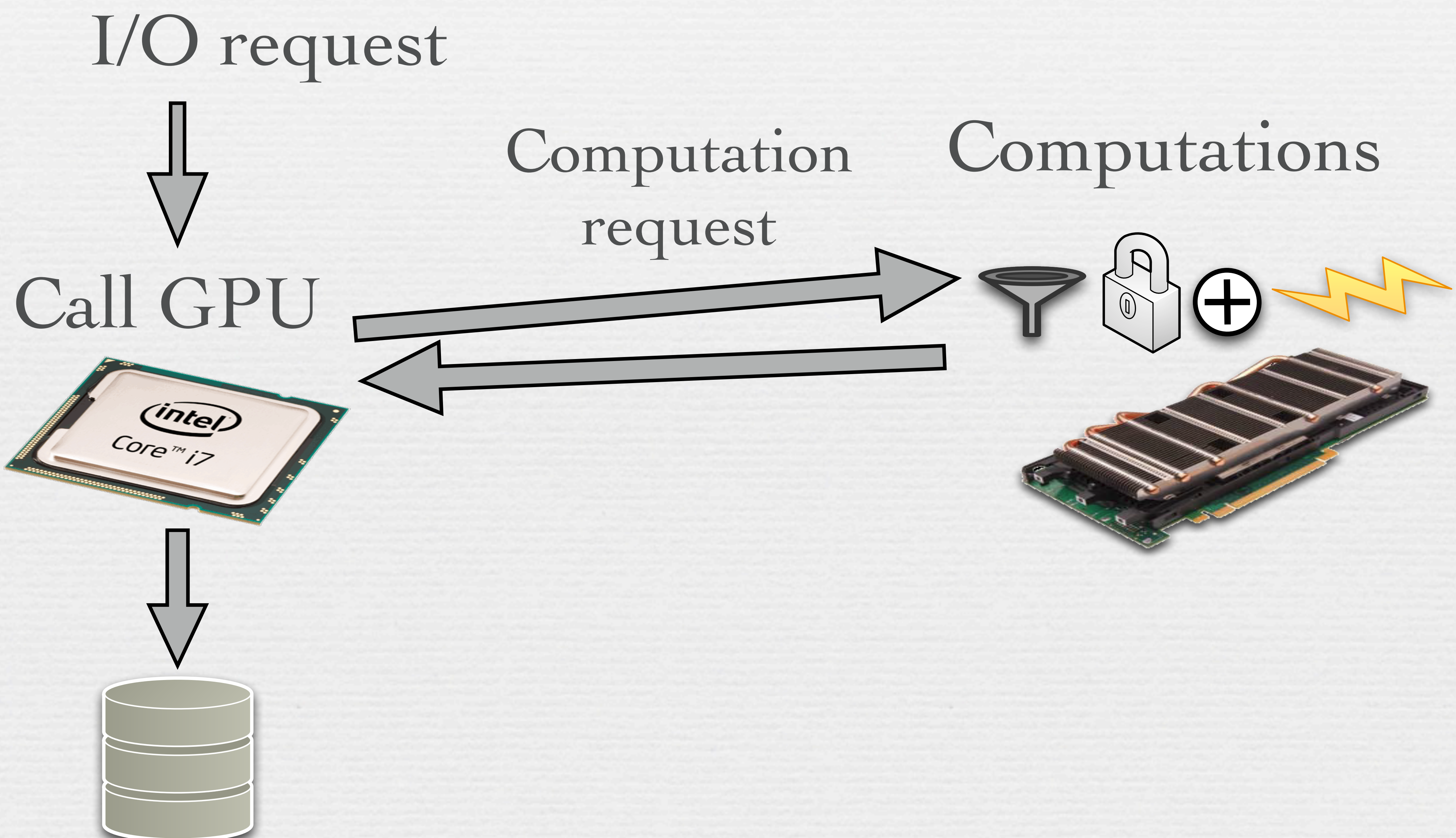
I/O request



Computations



The idea of GPUstore



But it's not that easy...

But it's not that easy...

- Storage challenges:
  - Make existing optimizations **just** work
    - Page cache, Read ahead, I/O scheduler ...
    - Leave them there, no big change, but still fast

But it's not that easy...

- Storage challenges:
  - Make existing optimizations **just** work
    - Page cache, Read ahead, I/O scheduler ...
    - Leave them there, no big change, but still fast
  - I/O request size decides computation request size
    - Too small/Too large requests are not preferred
    - GPU wants proper size, enough threads, full utilization

But it's not that easy...

- Storage challenges:
  - Make existing optimizations **just** work
    - Page cache, Read ahead, I/O scheduler ...
    - Leave them there, no big change, but still fast
  - I/O request size decides computation request size
    - Too small/Too large requests are not preferred
    - GPU wants proper size, enough threads, full utilization
  - Memory management in OS kernel is complicated

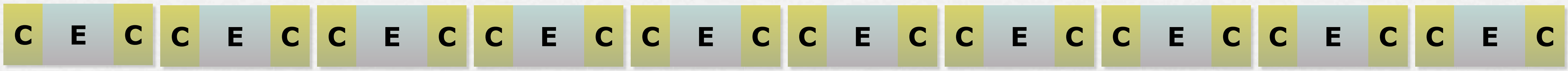
But it's not that easy...

- Storage challenges:
  - Make existing optimizations **just** work
    - Page cache, Read ahead, I/O scheduler ...
    - Leave them there, no big change, but still fast
  - I/O request size decides computation request size
    - Too small/Too large requests are not preferred
    - GPU wants proper size, enough threads, full utilization
  - Memory management in OS kernel is complicated
- Existing framework/API not for storage
  - TimeGraph, PTask, Sponge, Gdev ...

Some problems

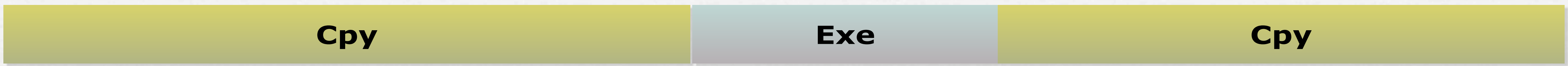
- Too large/too small I/O request
- Redundant buffering (mm problem)
- GPU execution resource abstraction for management

Small I/O => Small Computation



- Small request can't provide enough threads for GPU scheduling
  - e.g. encrypt 64KB disk block with AES CTR
    - 16B AES block size
    - 4K independent blocks = 4K threads
  - To hide mem latency, use all cores
- GPU kernel launch overhead not proportional to req size

Large I/O => Large Computation



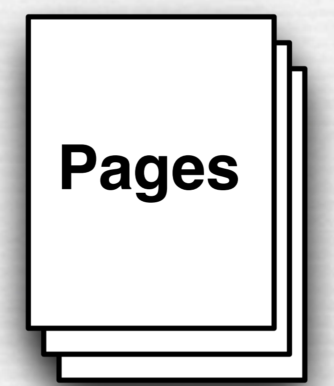
- Large request cause long-time waiting/blocking
  - Hard to avoid, e.g. `read(fd, buf, 1024*1024*128);`
    - but we can do better
  - Key to solution: GPU is “multi-task-able”
    - Multiple kernel execution
    - Execution and copy overlapping
    - Multiple DMA engines

8

Redundant buffering

Redundant buffering

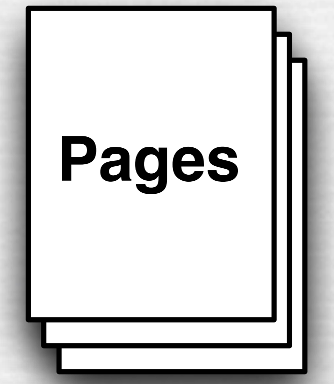
Memory pages allocated at somewhere else in existing code



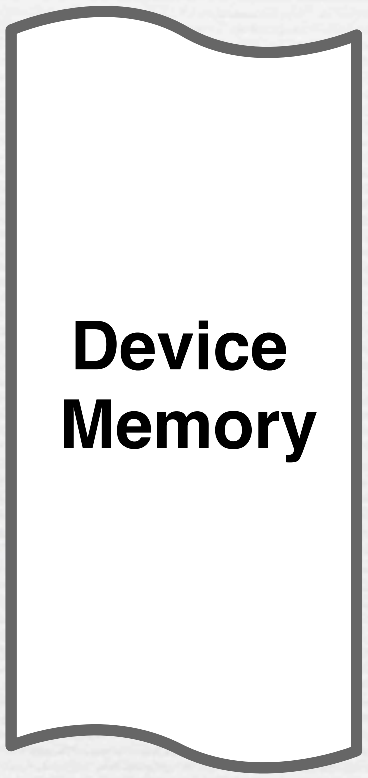
Storage code's pages

Redundant buffering

Memory pages
allocated at
somewhere else
in existing code

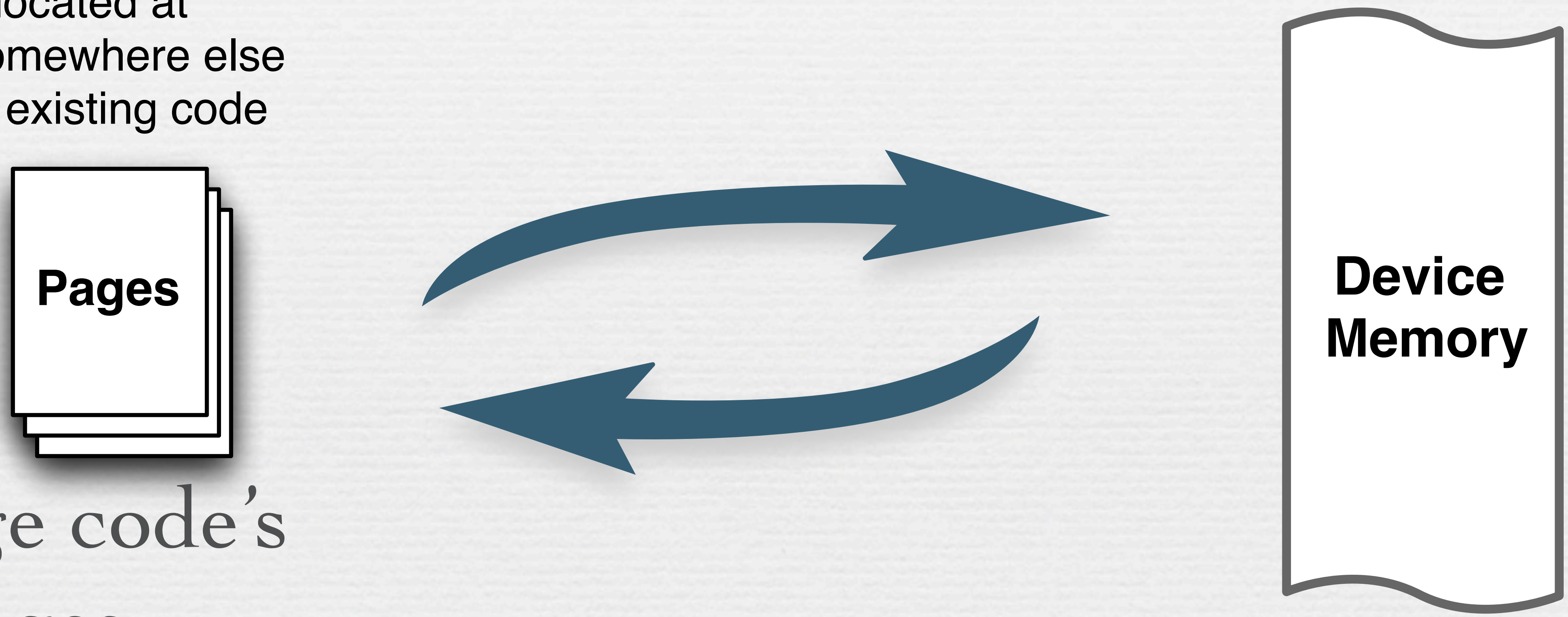


Storage code's
pages



Redundant buffering

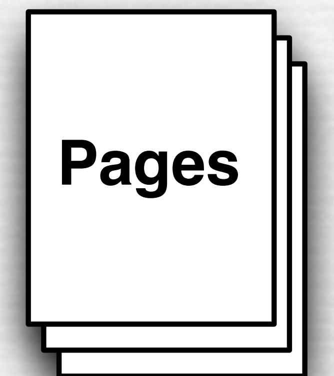
Memory pages allocated at somewhere else in existing code



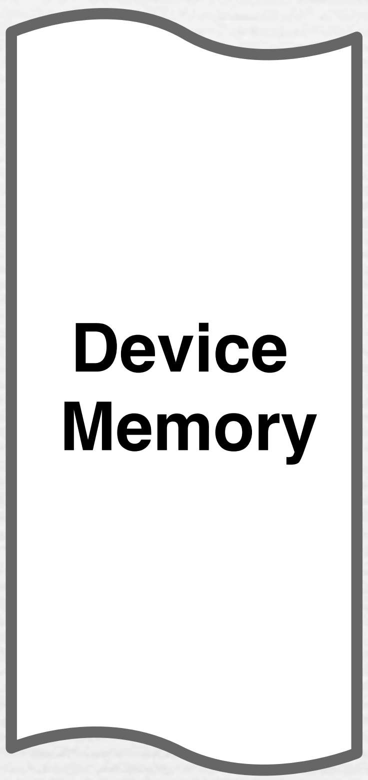
Storage code's pages

Redundant buffering

Memory pages
allocated at
somewhere else
in existing code

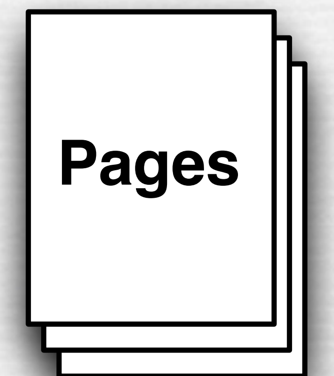


Storage code's
pages

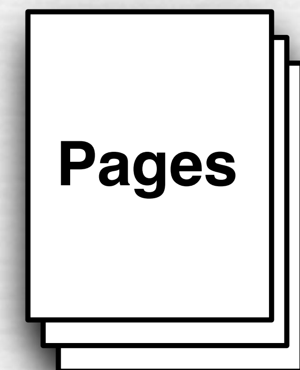


Redundant buffering

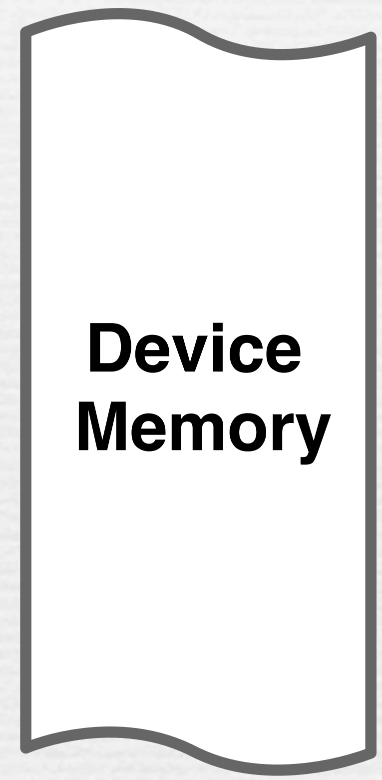
Memory pages
allocated at
somewhere else
in existing code



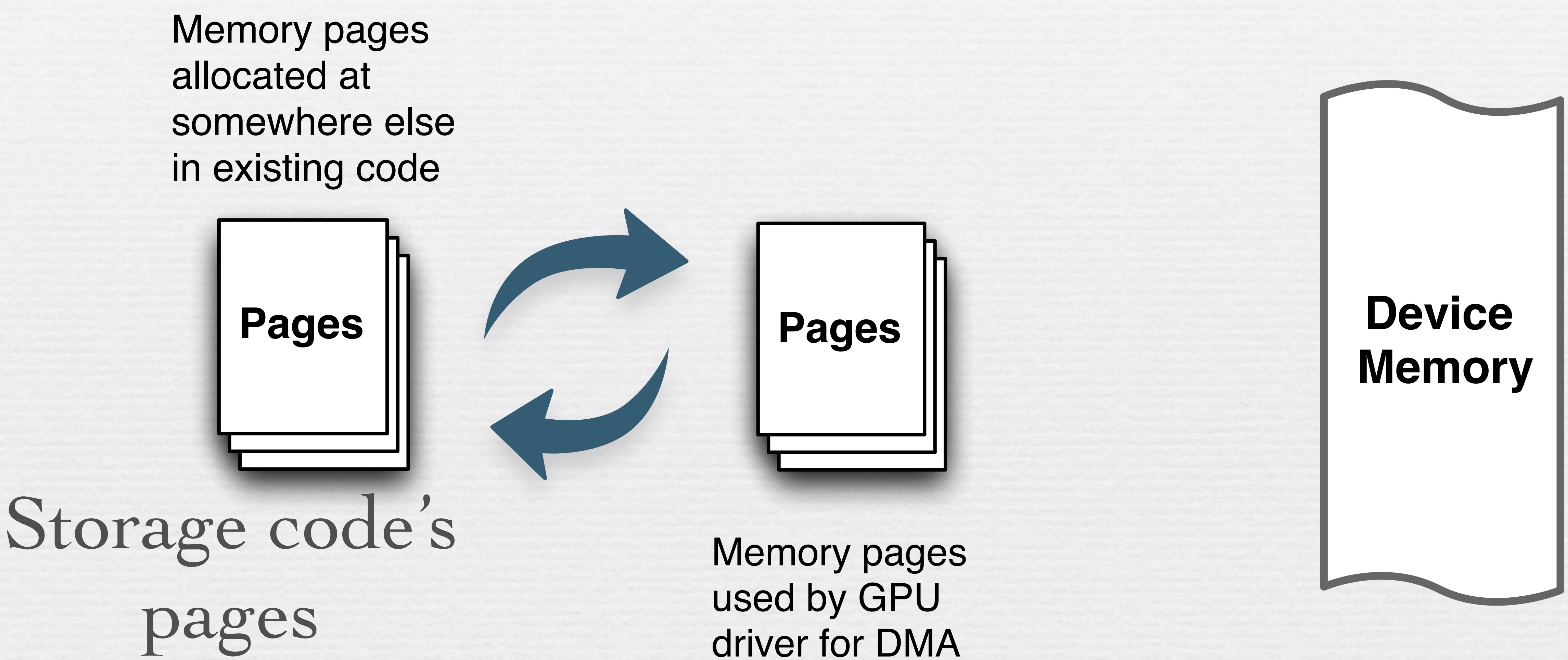
Storage code's
pages



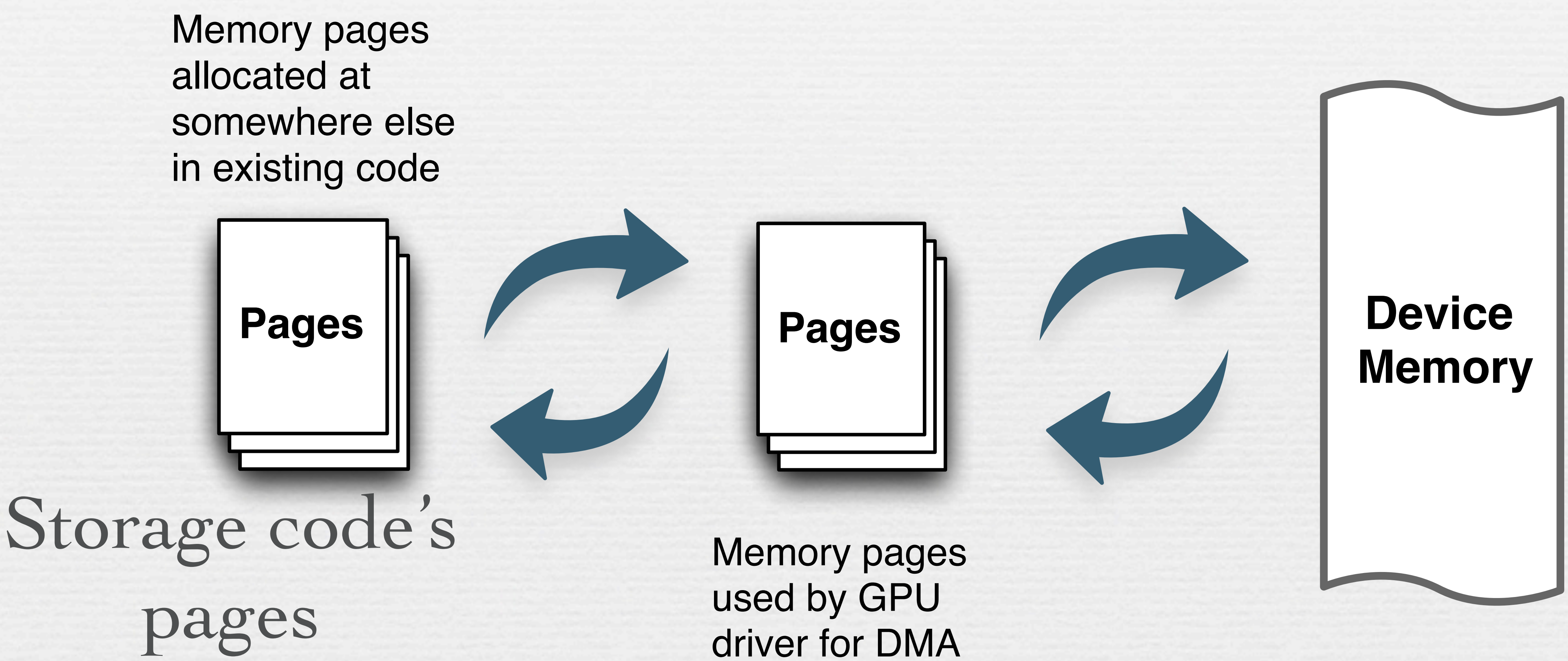
Memory pages
used by GPU
driver for DMA



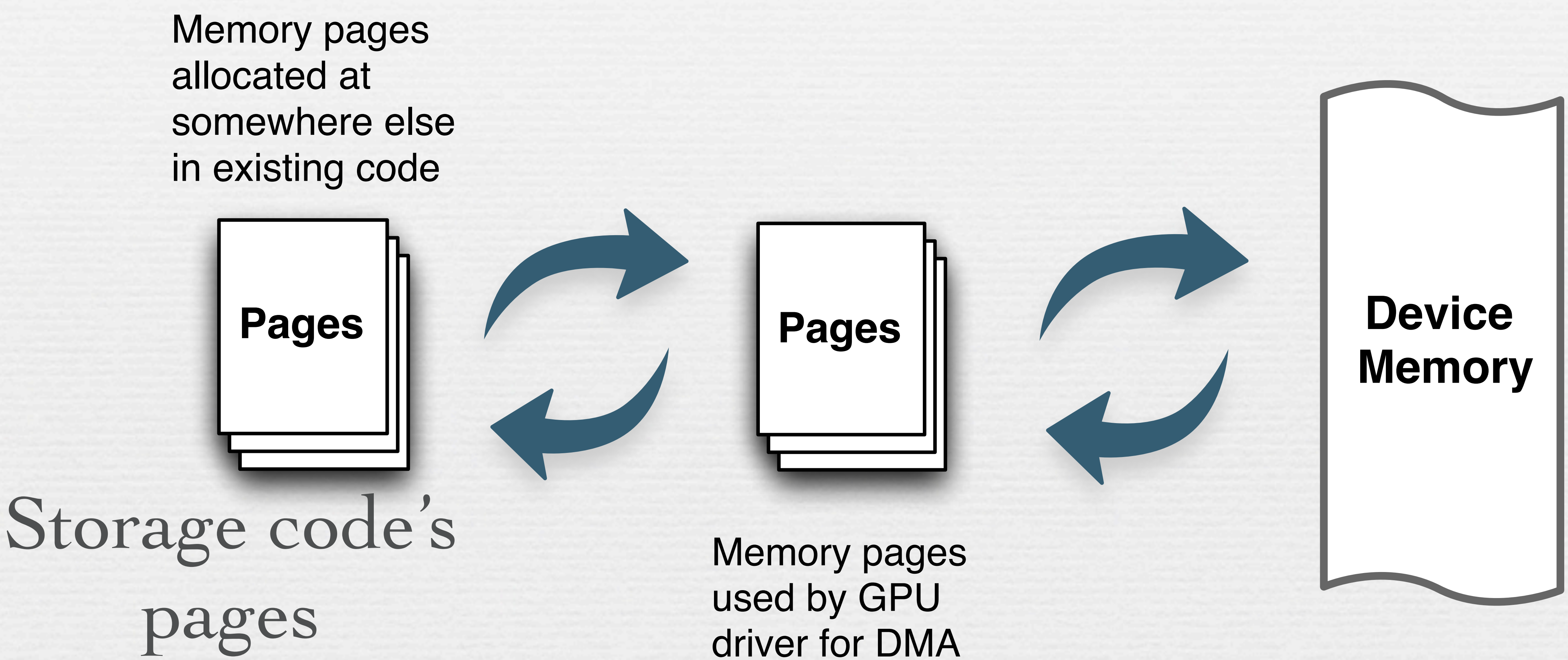
Redundant buffering



Redundant buffering



Redundant buffering



Make existing mechanisms just work: pages from page-cache, scheduled I/O request pages

GPU Execution Resources

- DMA engines
- Ability to run multiple kernels
- Ability to overlap execution and copy
- Multiple GPUs

GPU Execution Resources

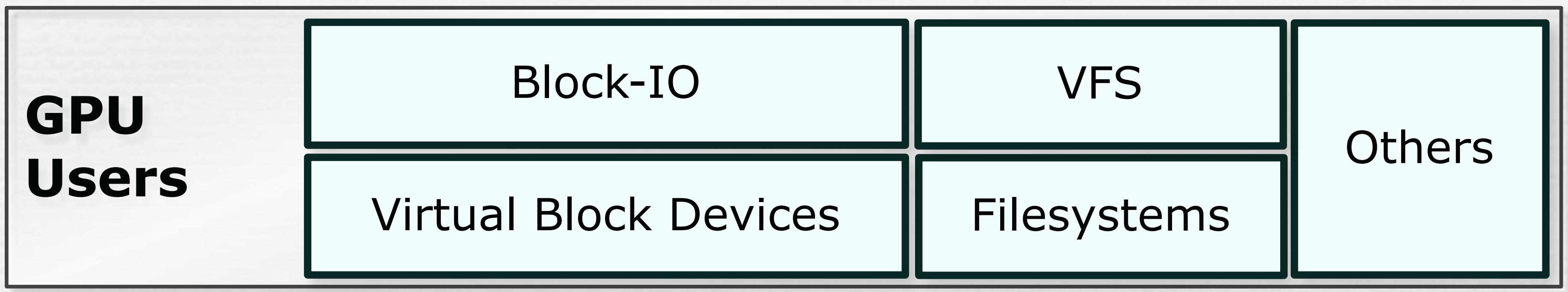
- DMA engines
- Ability to run multiple kernels
- Ability to overlap execution and copy
- Multiple GPUs

Abstract them for management
like CPU cores for CPU execution

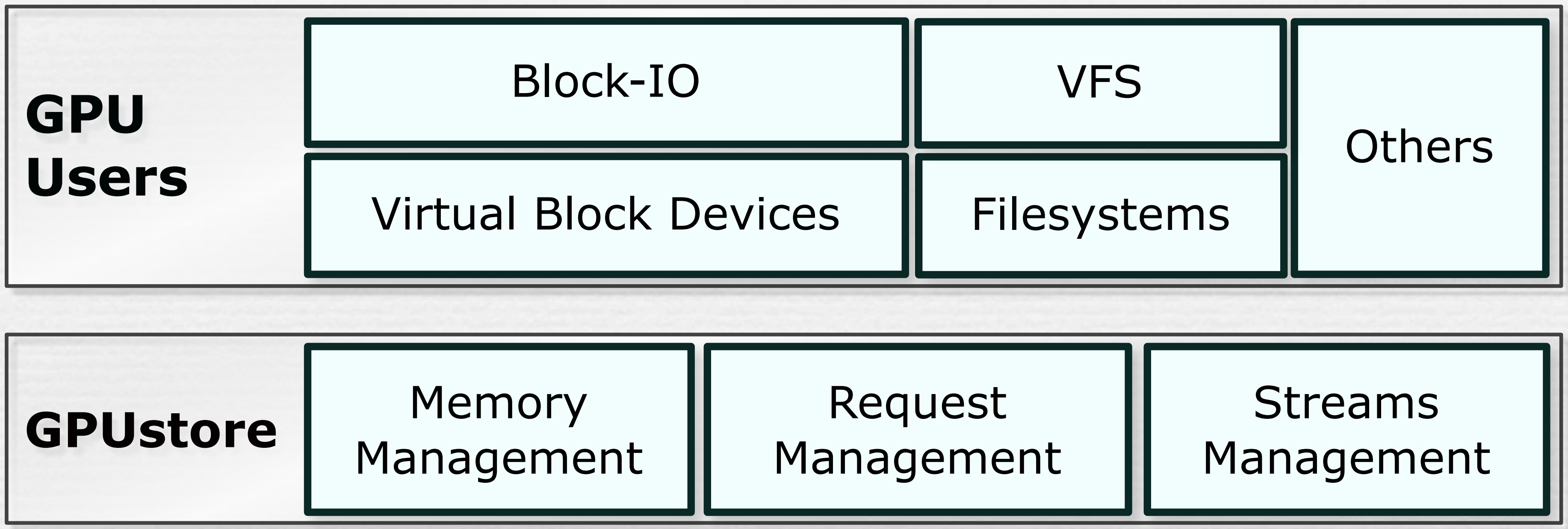
GPUstore

11

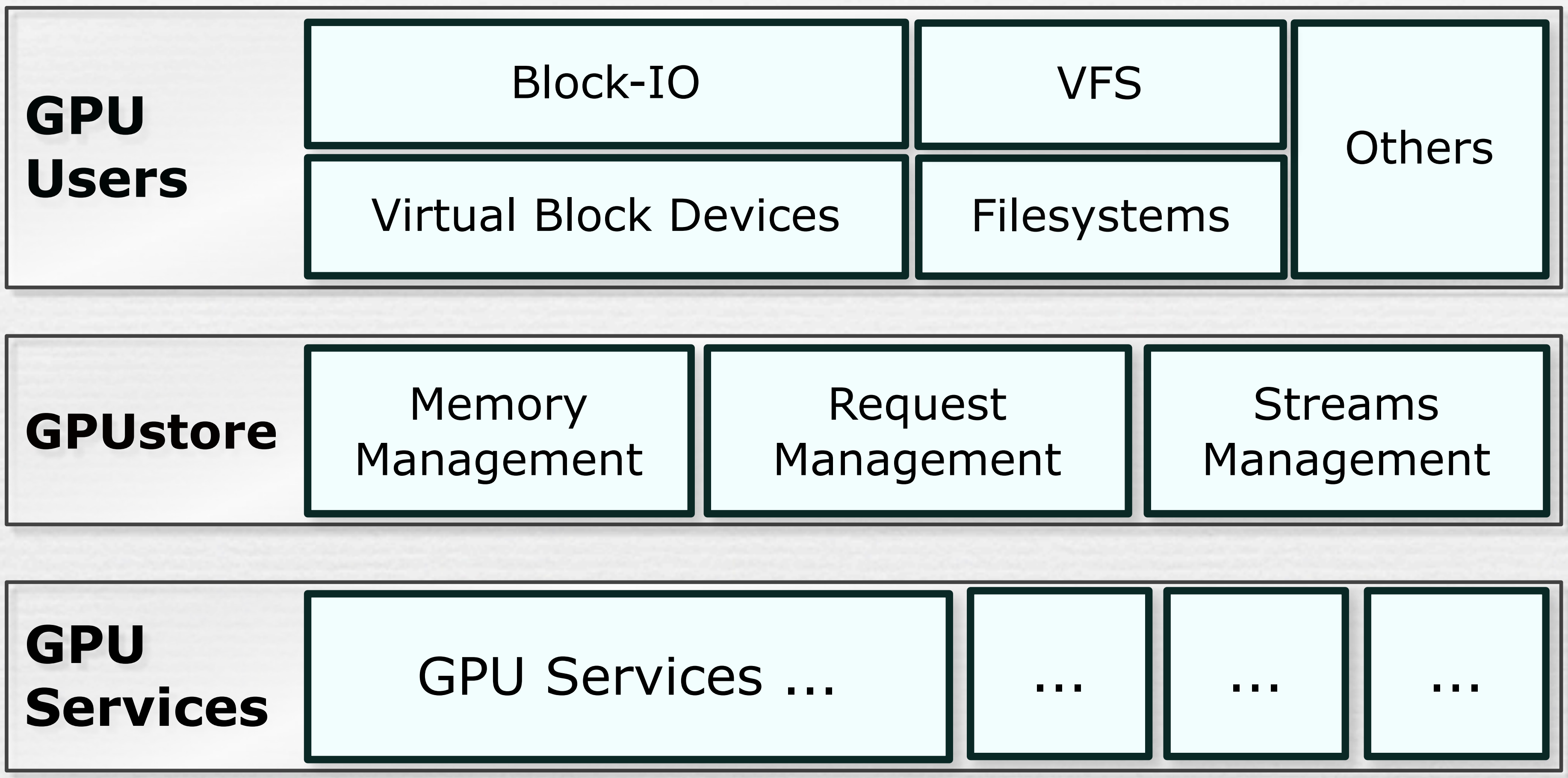
GPUstore


GPU Users
Block-IO
VFS
Virtual Block Devices
Filesystems
Others

GPUstore

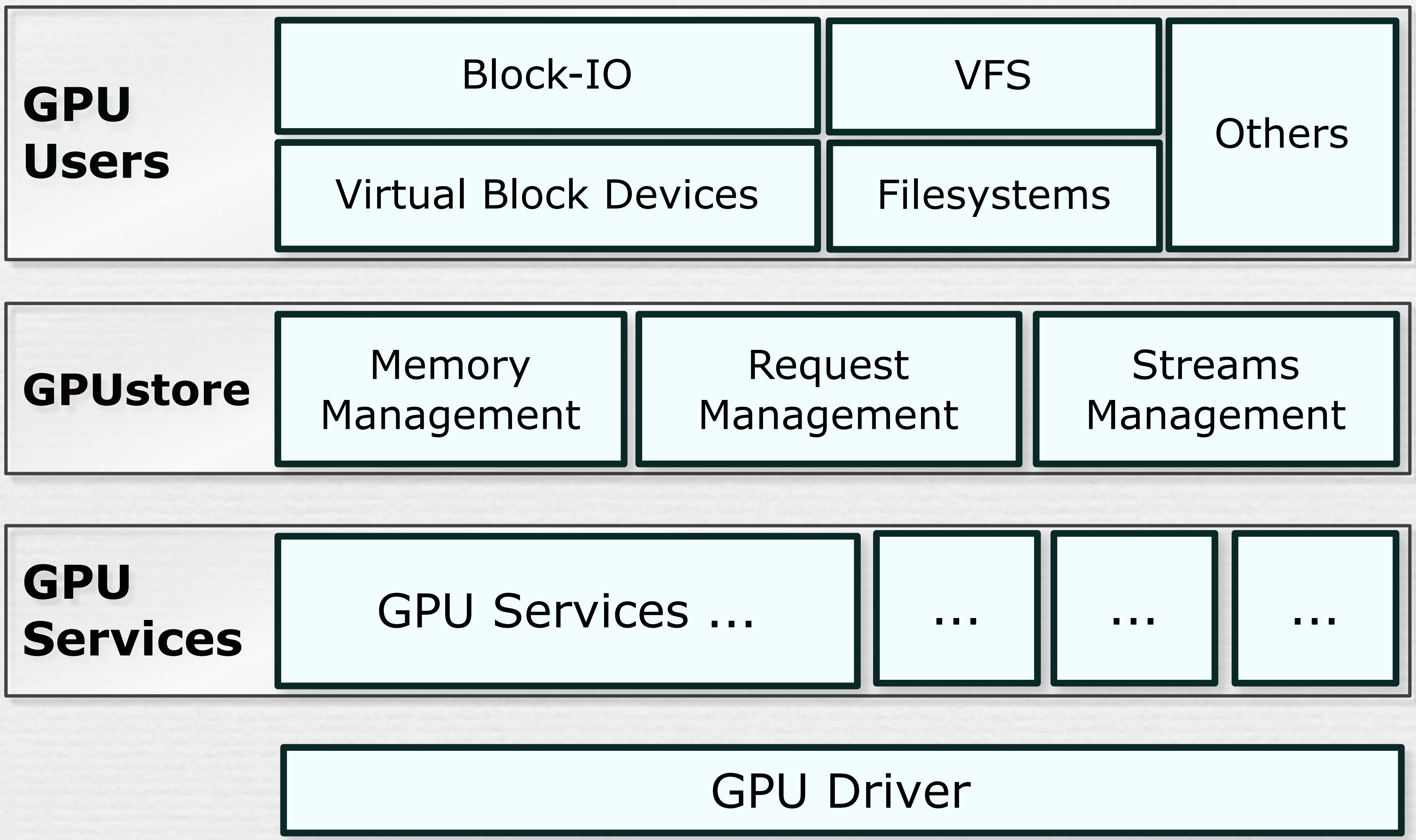

GPU Users
Block-IO
VFS
Others
Virtual Block Devices
Filesystems
GPUstore
Memory Management
Request Management
Streams Management

GPUstore


GPU Users
Block-IO
VFS
Others
Virtual Block Devices
Filesystems
GPUstore
Memory Management
Request Management
Streams Management
GPU Services
GPU Services ...
...
...
...

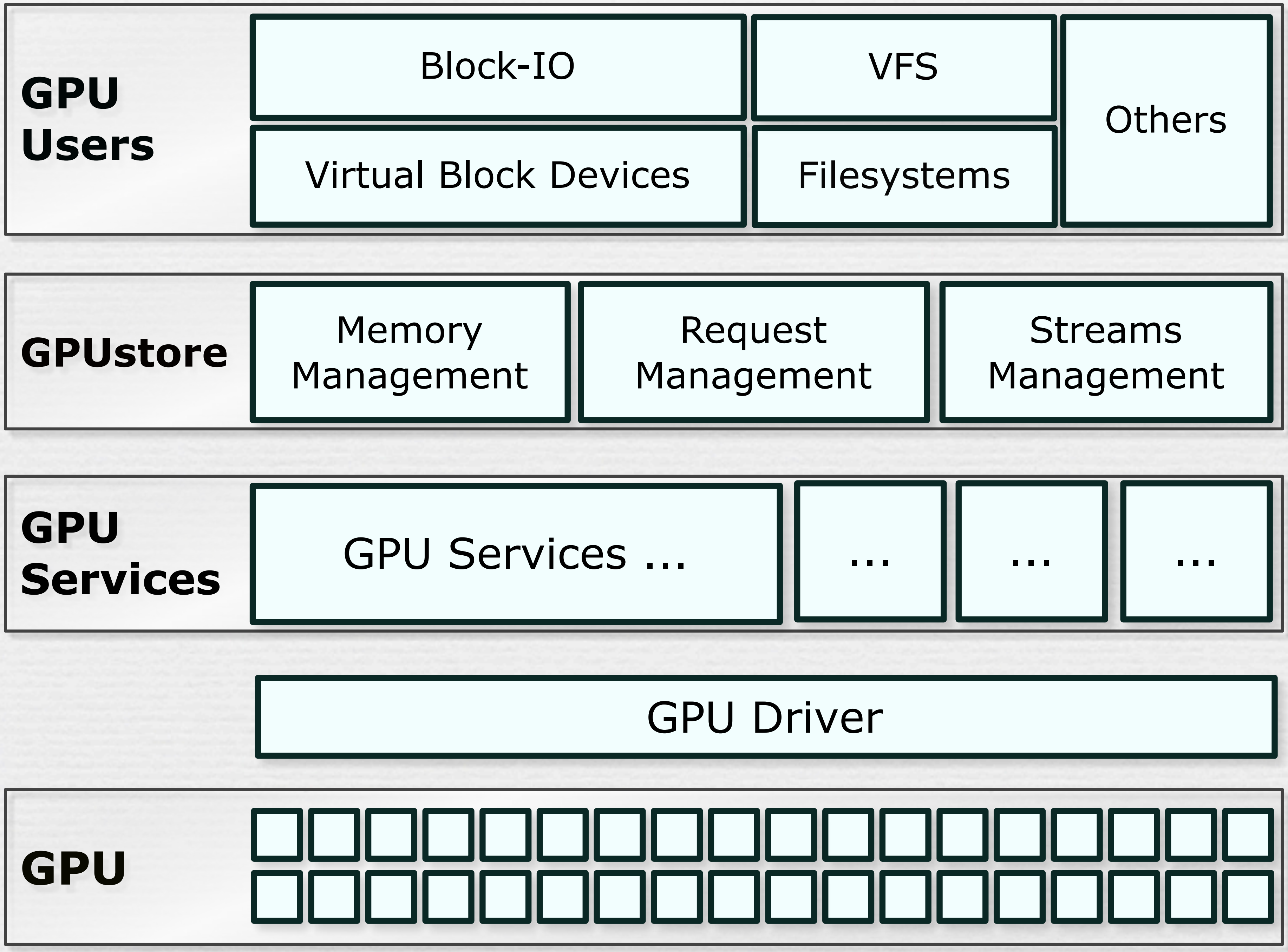
11

GPUstore


GPU Users
Block-IO
VFS
Others
Virtual Block Devices
Filesystems
GPUstore
Memory Management
Request Management
Streams Management
GPU Services
GPU Services ...
...
...
...
GPU Driver

11

GPUstore


GPU Users
Block-IO
VFS
Others
Virtual Block Devices
Filesystems
GPUstore
Memory Management
Request Management
Streams Management
GPU Services
GPU Services ...
...
...
...
GPU Driver
GPU

GPUstore

- Stream Management
- Request Management
- Memory Management

"Stream" for resource abstraction

- Term borrowed from CUDA
- A **stream** is an abstract execution pipeline including:
  - DMA engine
  - GPU cores
- Hide real # DMA engines, # cores, # GPUs
- Streams scheduled for request processing
  - First come, first serve now

GPUstore

- ~~Stream Management~~
- Request Management
- Memory Management

Request Management

Request Management

- Computation request scheduling behind all existing mechanisms
  - Just before invoking GPU operations
  - Different from I/O scheduler
    - Considering computing speed, not read/write speed, sectors' locations...

Request Management

- Computation request scheduling behind all existing mechanisms
  - Just before invoking GPU operations
  - Different from I/O scheduler
    - Considering computing speed, not read/write speed, sectors' locations...
- Too small - Merge

Request Management

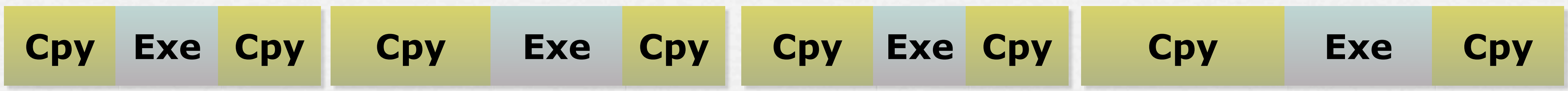
- Computation request scheduling behind all existing mechanisms
  - Just before invoking GPU operations
  - Different from I/O scheduler
    - Considering computing speed, not read/write speed, sectors' locations...
- Too small - Merge
- Too large - Split

Request Management

- Computation request scheduling behind all existing mechanisms
  - Just before invoking GPU operations
  - Different from I/O scheduler
    - Considering computing speed, not read/write speed, sectors' locations...
- Too small - Merge
- Too large - Split
- Right size?

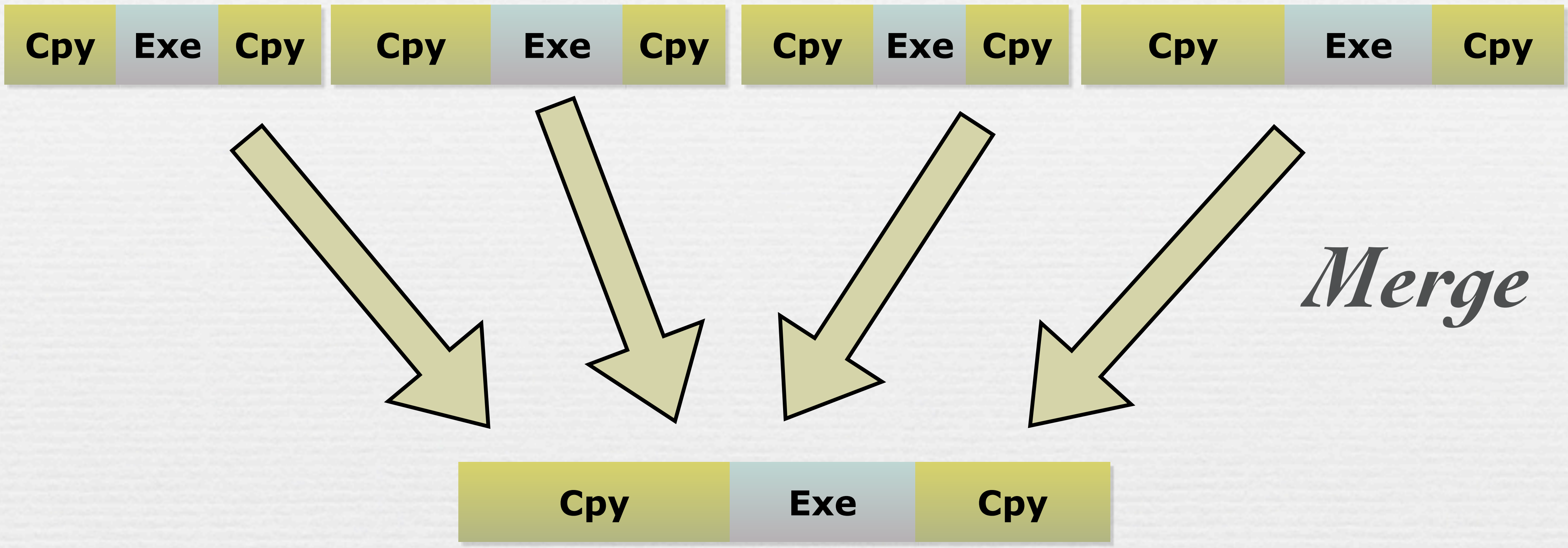
Too small I/O cause small computation

Too small I/O cause small computation

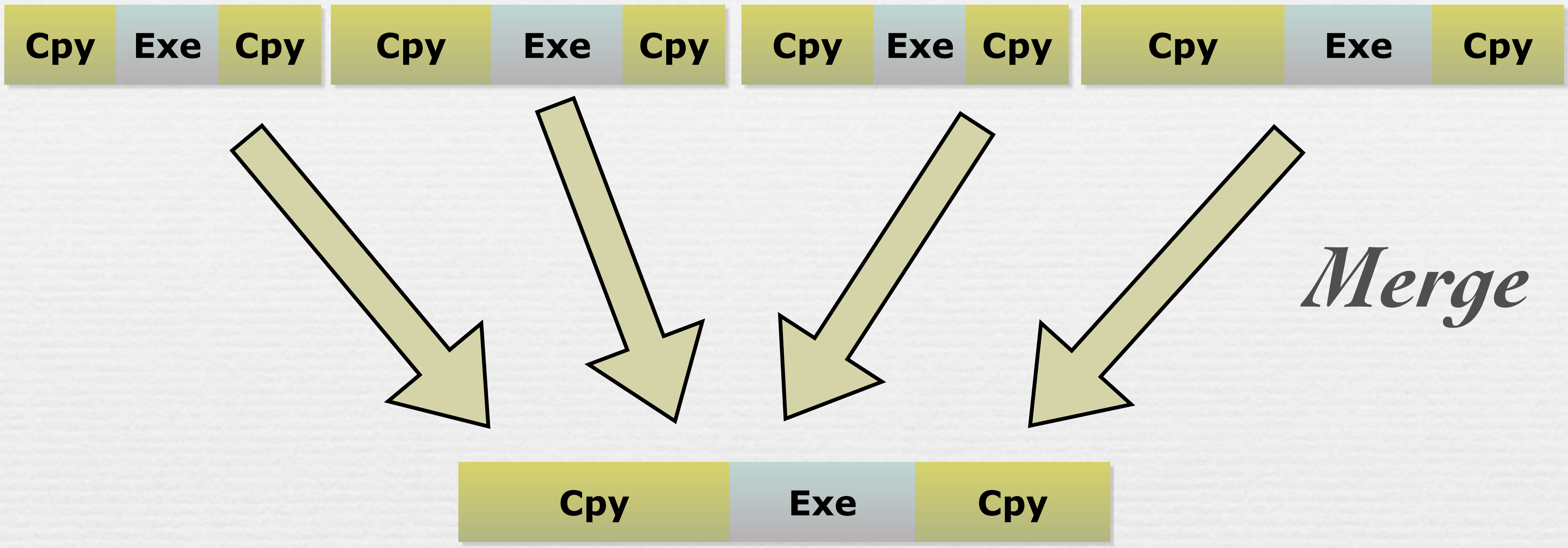


16

Too small I/O cause small computation

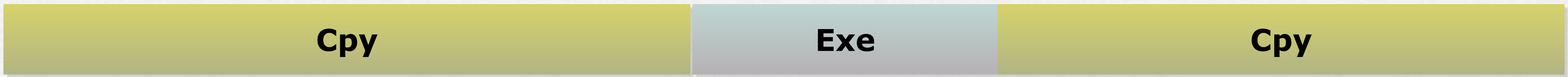


Too small I/O cause small computation



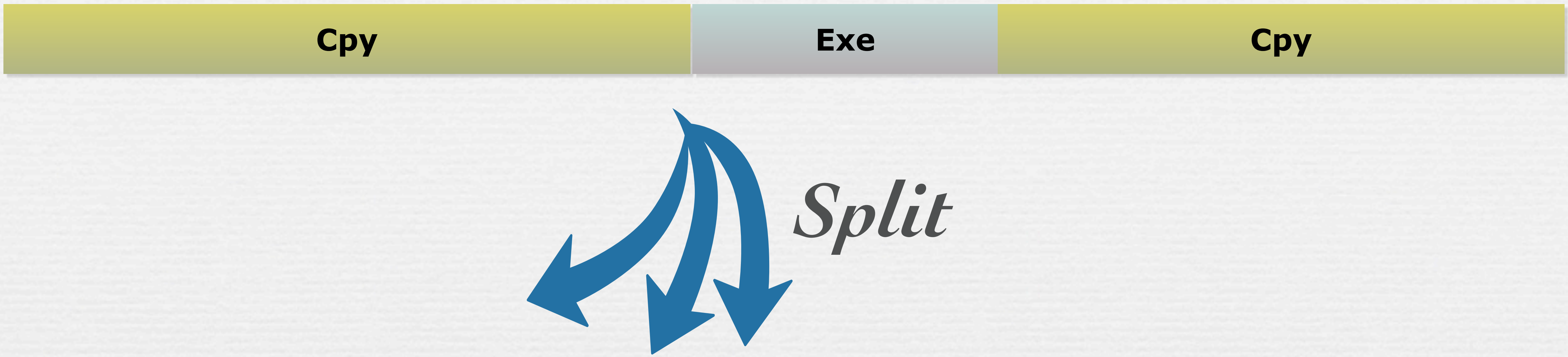
- Merge is not linear addition, total time is not sum of all original ones.
  - GPU utilization, mem latency, launch overhead

Too large I/O causes large computation

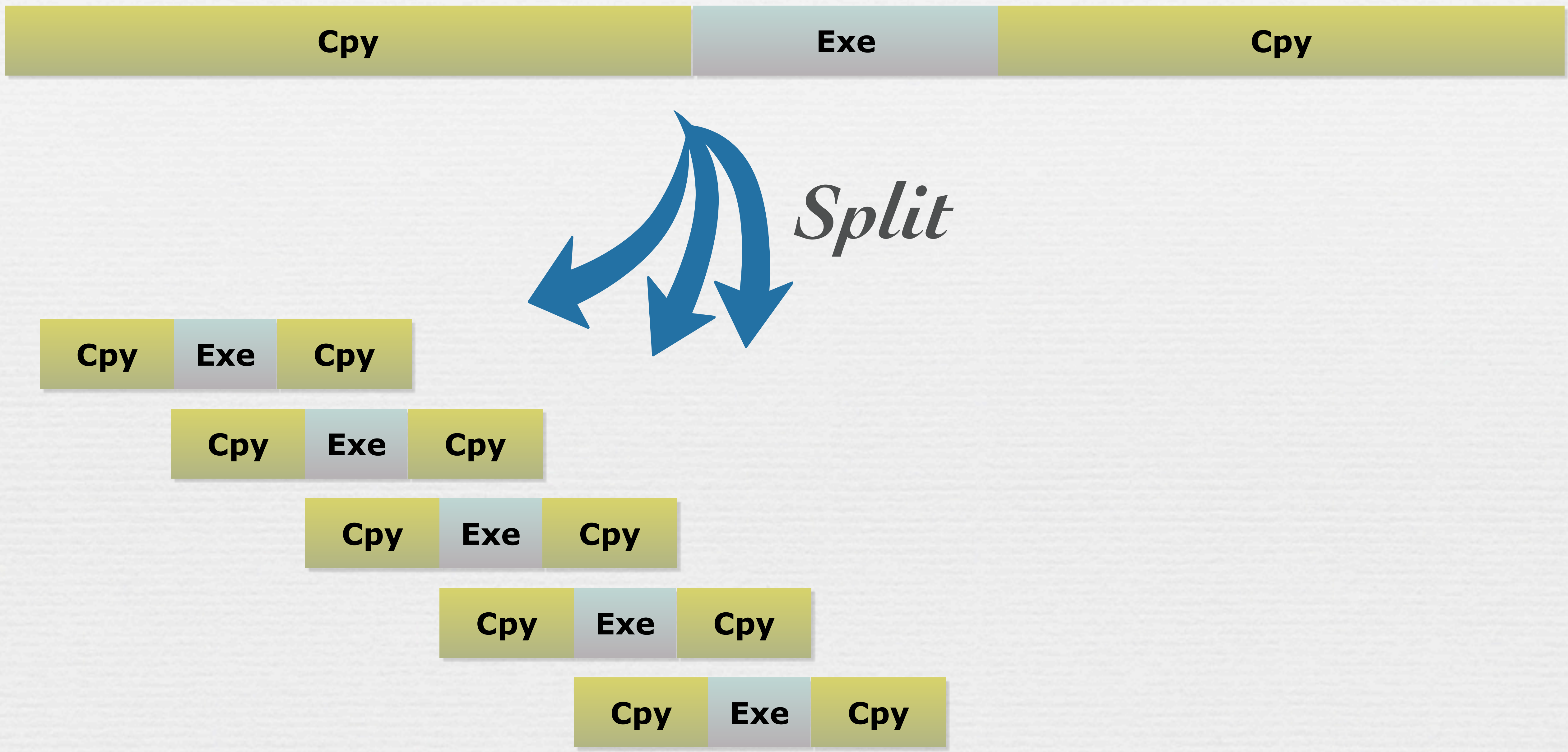


17

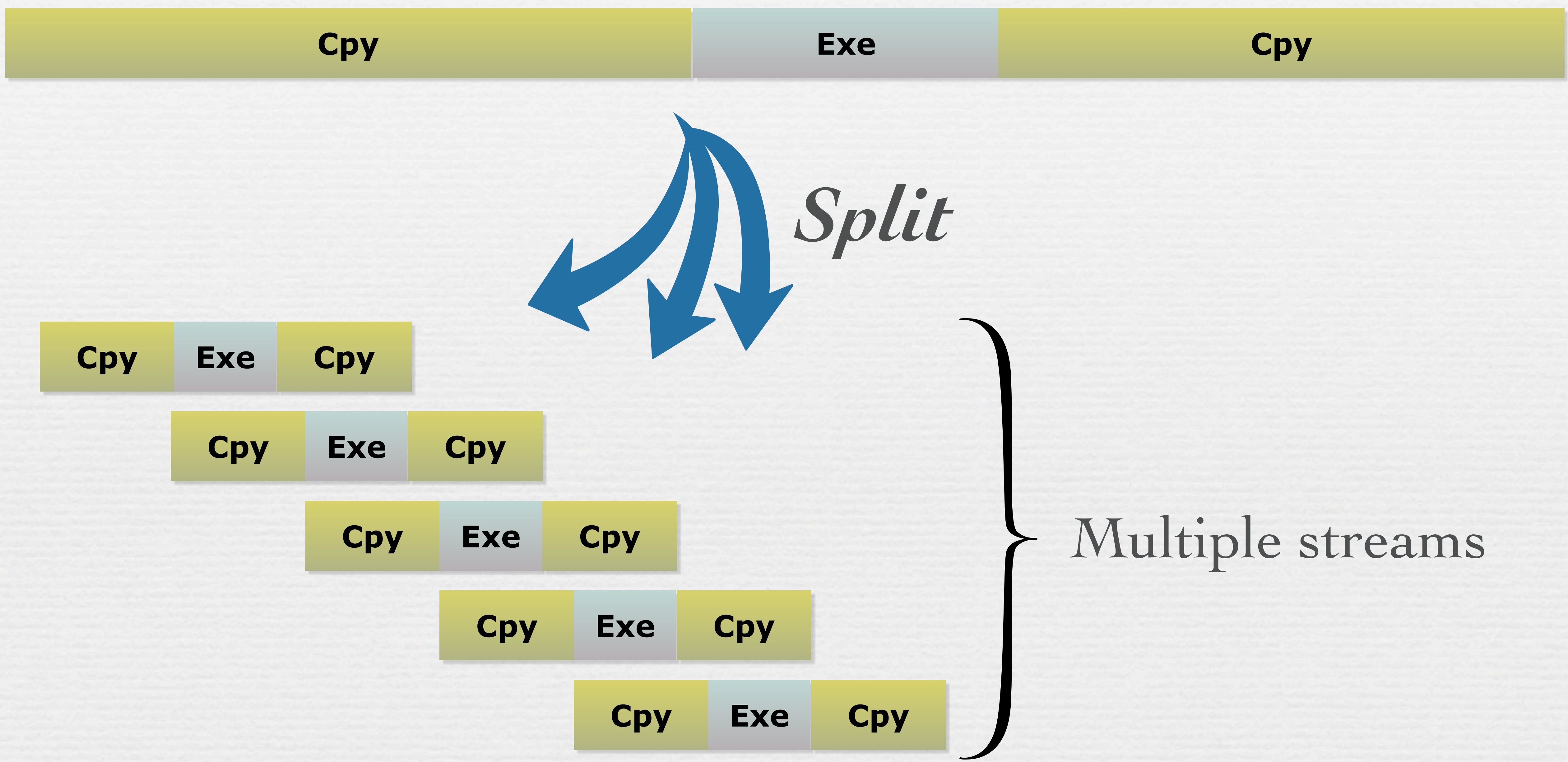
Too large I/O causes large computation



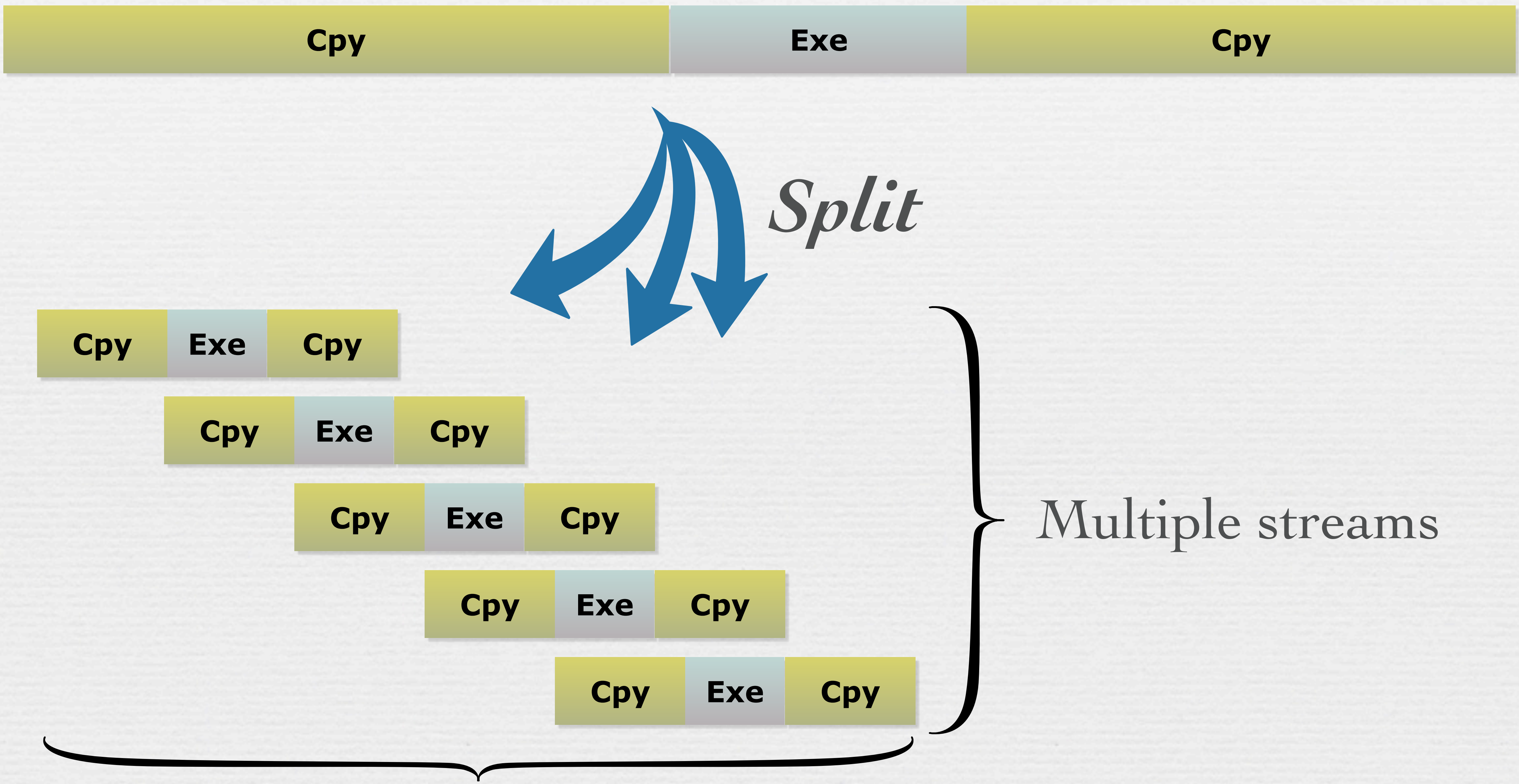
Too large I/O causes large computation



Too large I/O causes large computation



Too large I/O causes large computation



GPU overlapped copy & execution (Multiple DMA engine)

Service-specific request scheduling

Service-specific request scheduling

- Not too large, not too small, what's the RIGHT size:
  - Decided by service itself
    - Boot-time benchmark, service logic, computing features...

Service-specific request scheduling

- Not too large, not too small, what's the RIGHT size:
  - Decided by service itself
    - Boot-time benchmark, service logic, computing features…
- Make sure correctness: Merge/Split logic:
  - Done by service too
  - Simple ones may use common split/merge

GPUstore

- ~~Stream Management~~
- ~~Request Management~~
- Memory Management

19

GPUstore MM

GPUstore MM

- Remap pages for GPU DMA
  - Similar to cudaRegisterHost, but for scattered pages in kernel mode
  - To use existing pages

GPUstore MM

- Remap pages for GPU DMA
  - Similar to cudaRegisterHost, but for scattered pages in kernel mode
  - To use existing pages
- Allocate(and remap) GPU driver's pages directly
  - CUDA Page-locked memory in kernel mode
  - For easily changeable code/new code

20

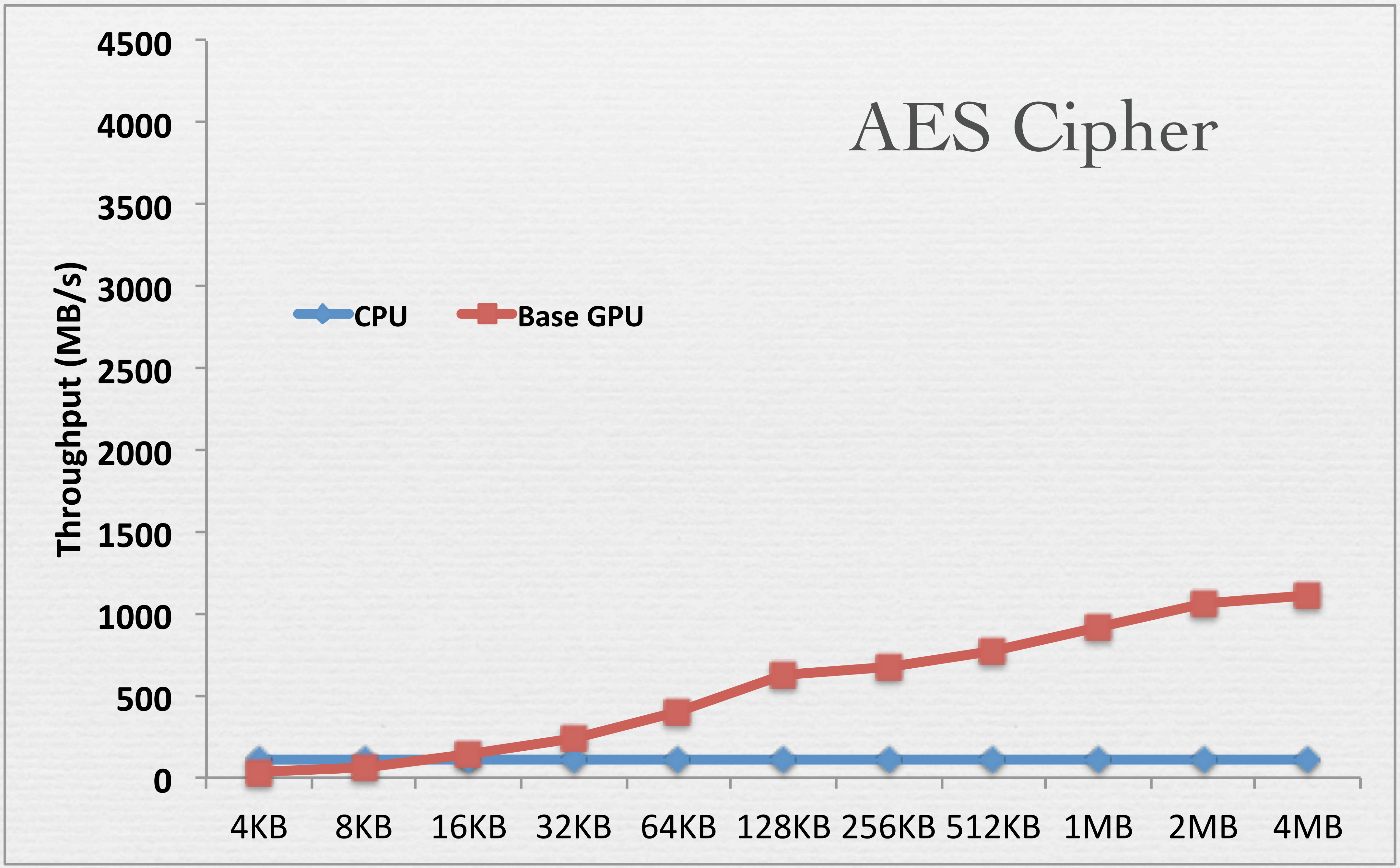
Evaluation

Applications: (Approximated LOC modification)

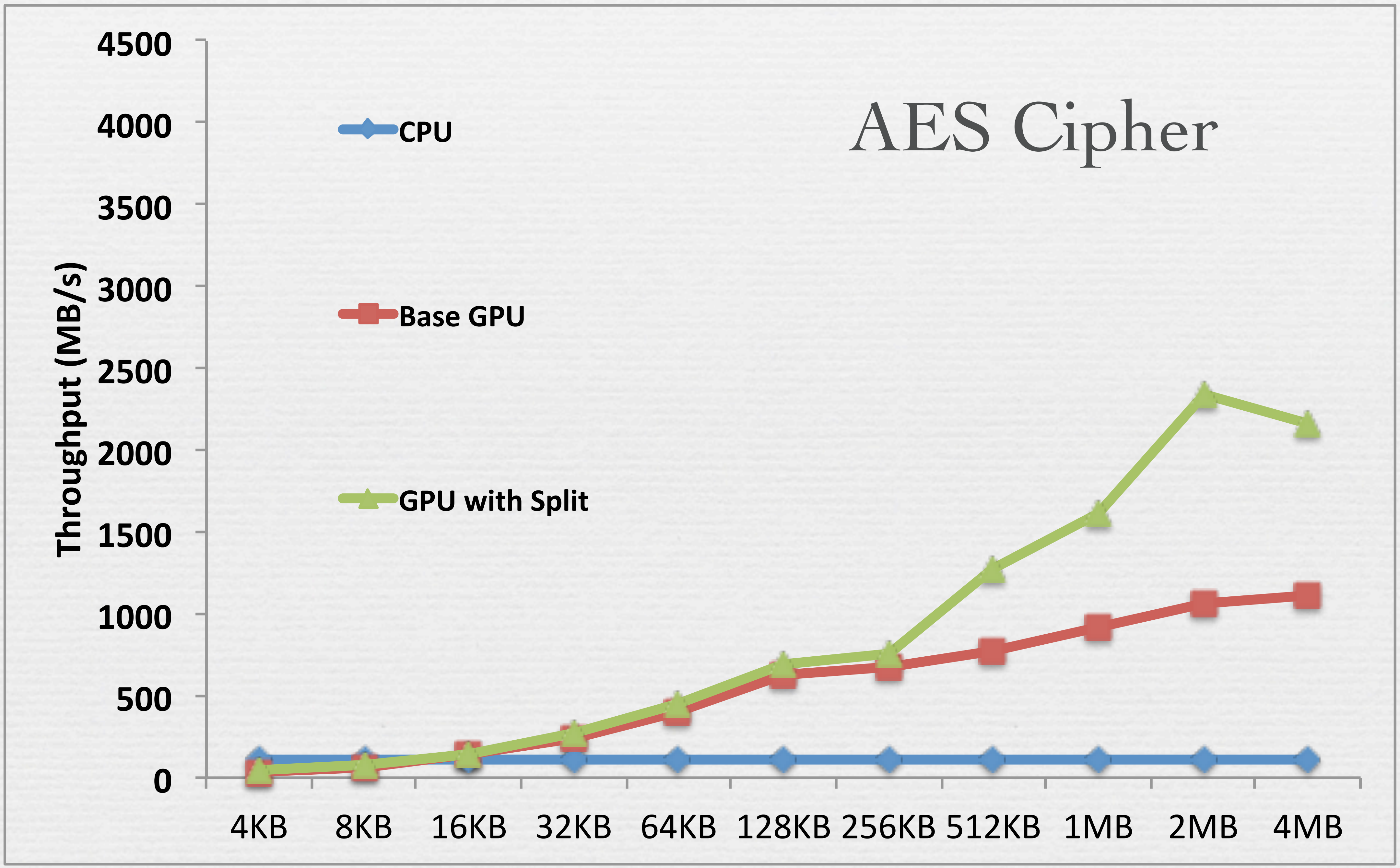
App.	Modified LOC	%	Request scheduling	MM
eCryptfs	200	2%	Merge/Split	Remapping
dm-crypt	50	3%	Merge/Split	No remapping
MD RAID6	20	0.3%	Merge/Split	No remapping

21

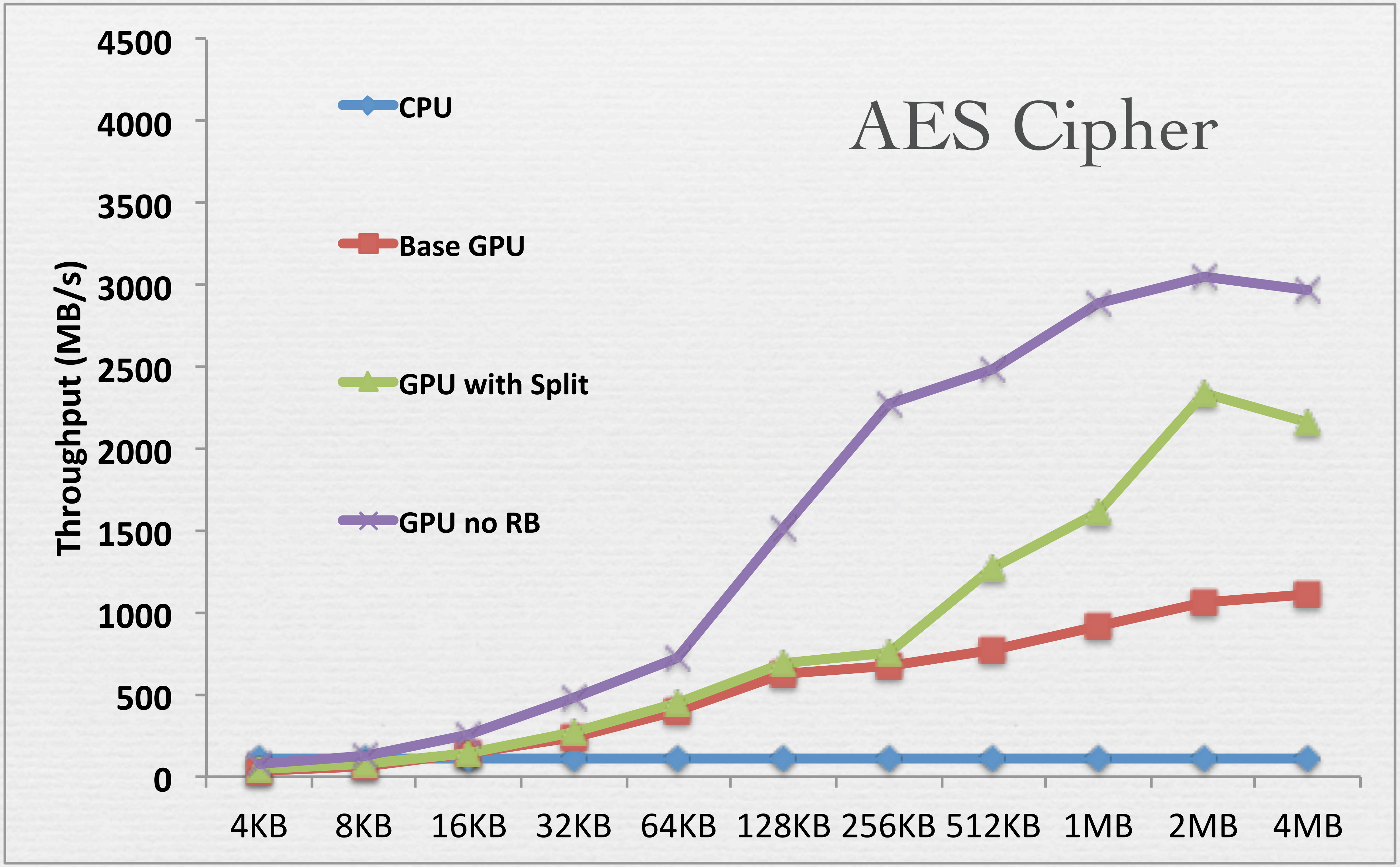
Effectiveness of optimizations



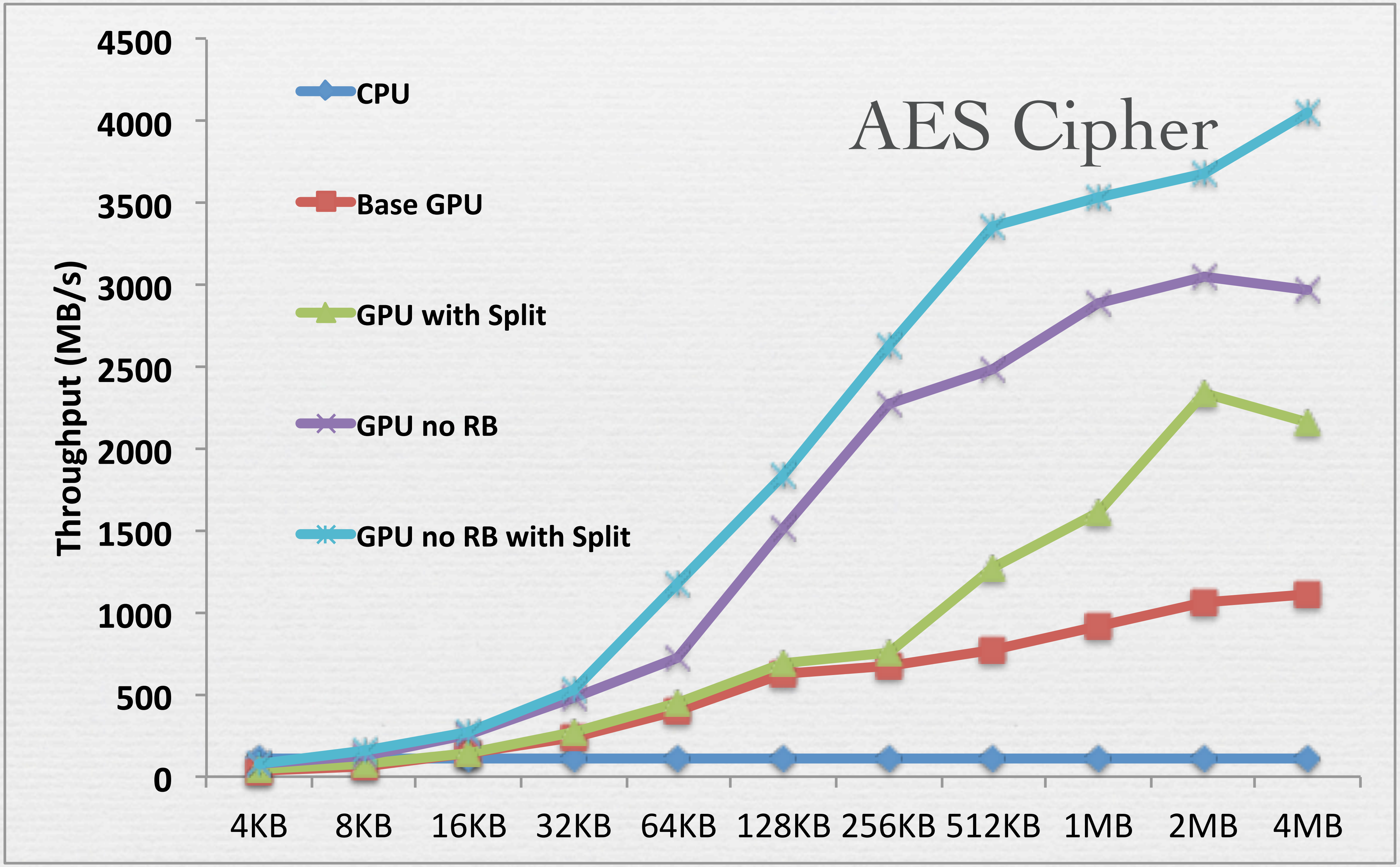
Effectiveness of optimizations



Effectiveness of optimizations

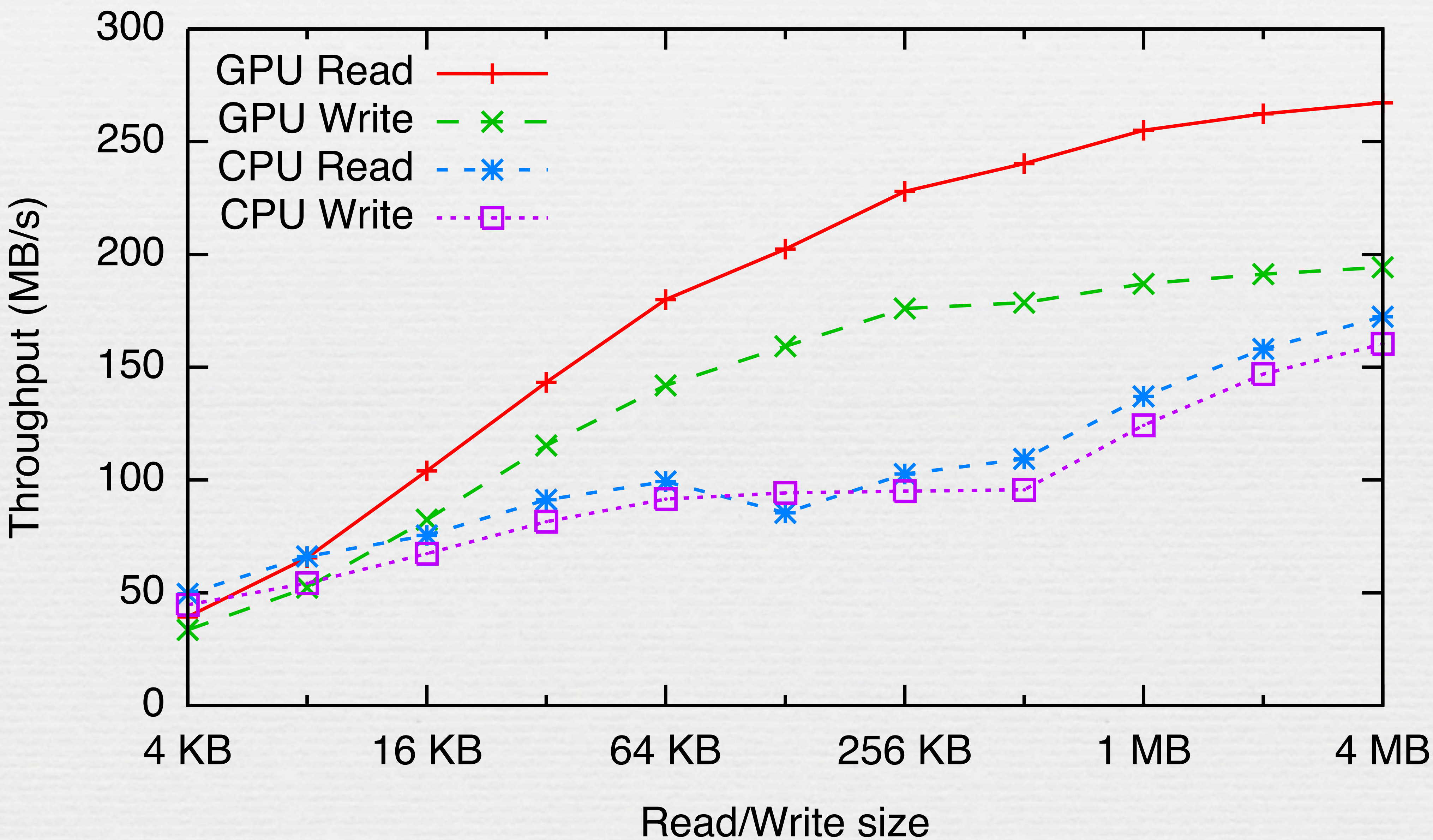


Effectiveness of optimizations



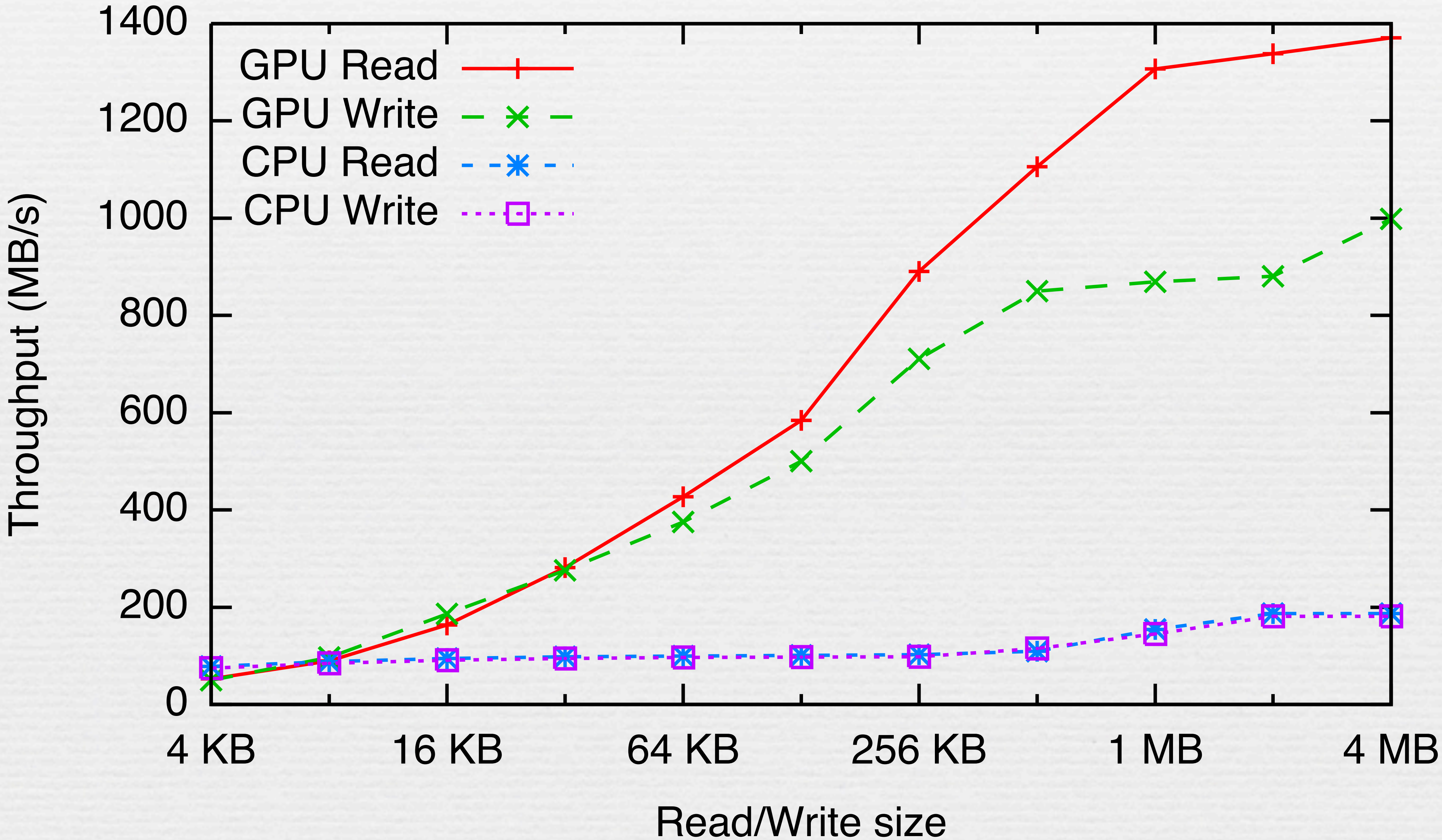
25

dm-crypt SSD

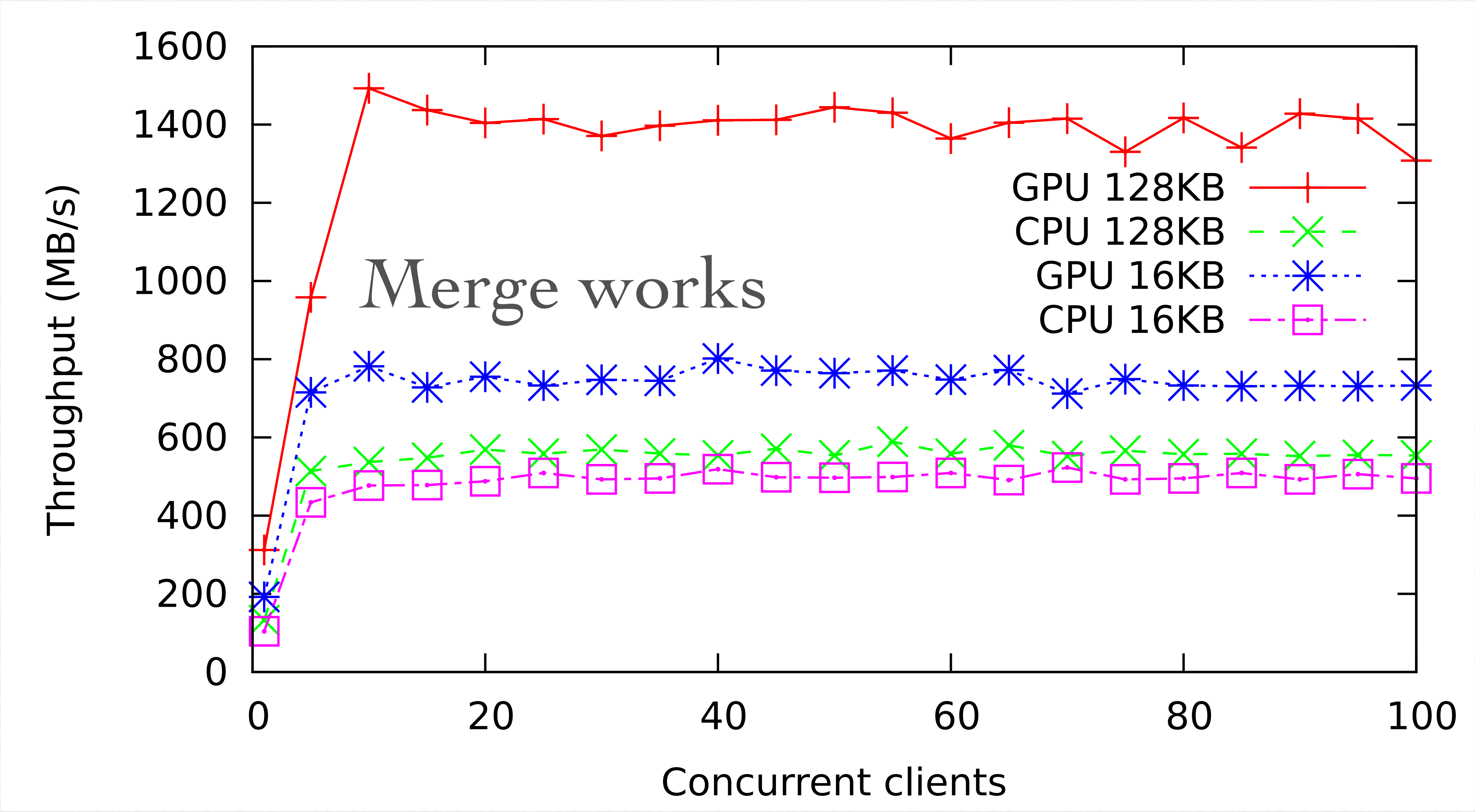


GPU Read
GPU Write
CPU Read
CPU Write
Throughput (MB/s)
300
250
200
150
100
50
0
4 KB
16 KB
64 KB
256 KB
1 MB
4 MB
Read/Write size

Better dm-crypt on RAM disk

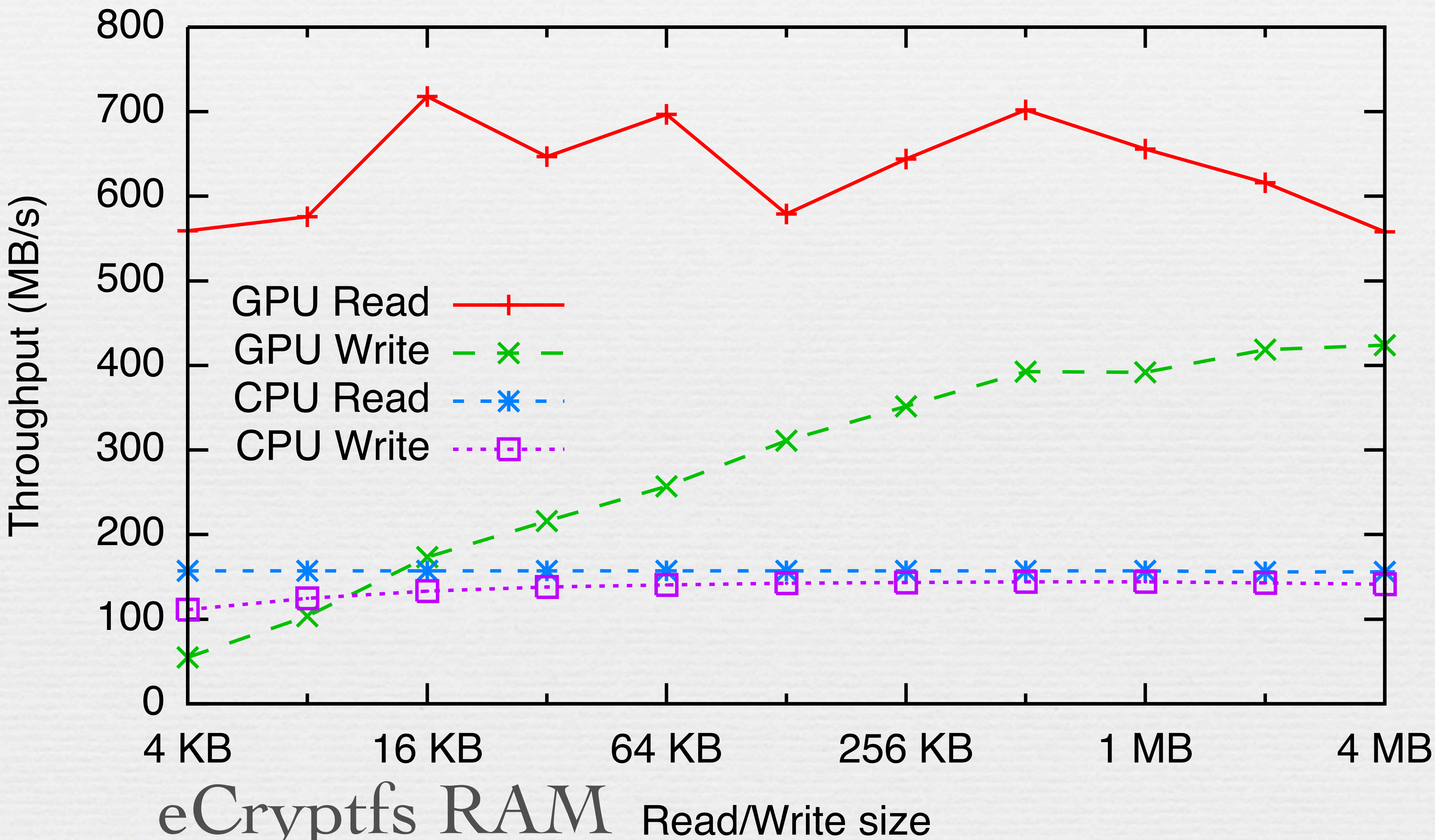


eCryptfs Concurrent clients



RAM Disk, Write only

Existing optimizations still work



eCryptfs RAM

More results in paper

- Upper-bound of best GPUstore framework performance
- eCryptfs on SSD
- RAID6 recovery algorithm
- RAID6 on HDs/RAM disks
- ...

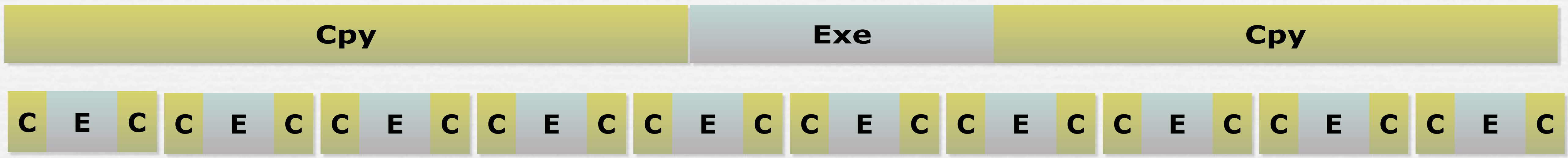
URLs

- Google Code: http://code.google.com/p/kgpu
- Github: https://github.com/wbsun/kgpu
- Being refactored
- Will use Gdev for kernel-level CUDA access

Thanks!
Q&A

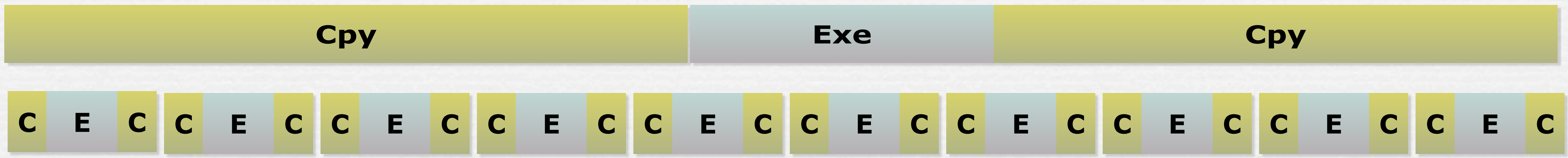
32

Hate too large/small I/O requests?



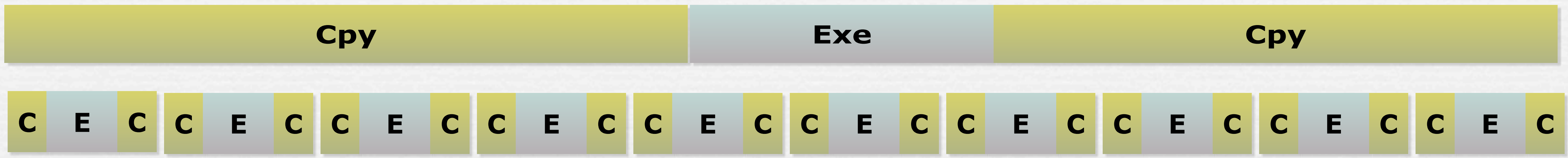
33

Hate too large/small I/O requests?



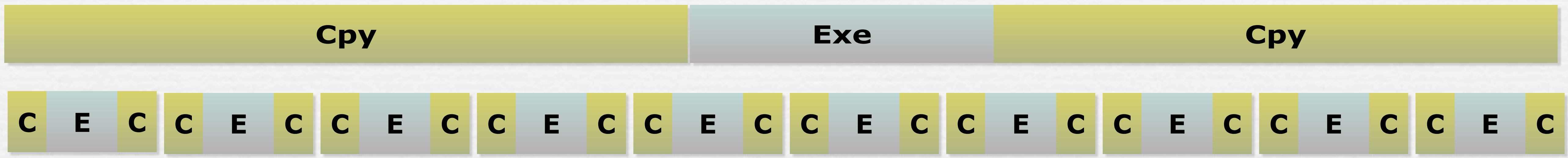
- Small request can't provide enough threads for GPU scheduling
  - To hide mem latency, use all cores

Hate too large/small I/O requests?



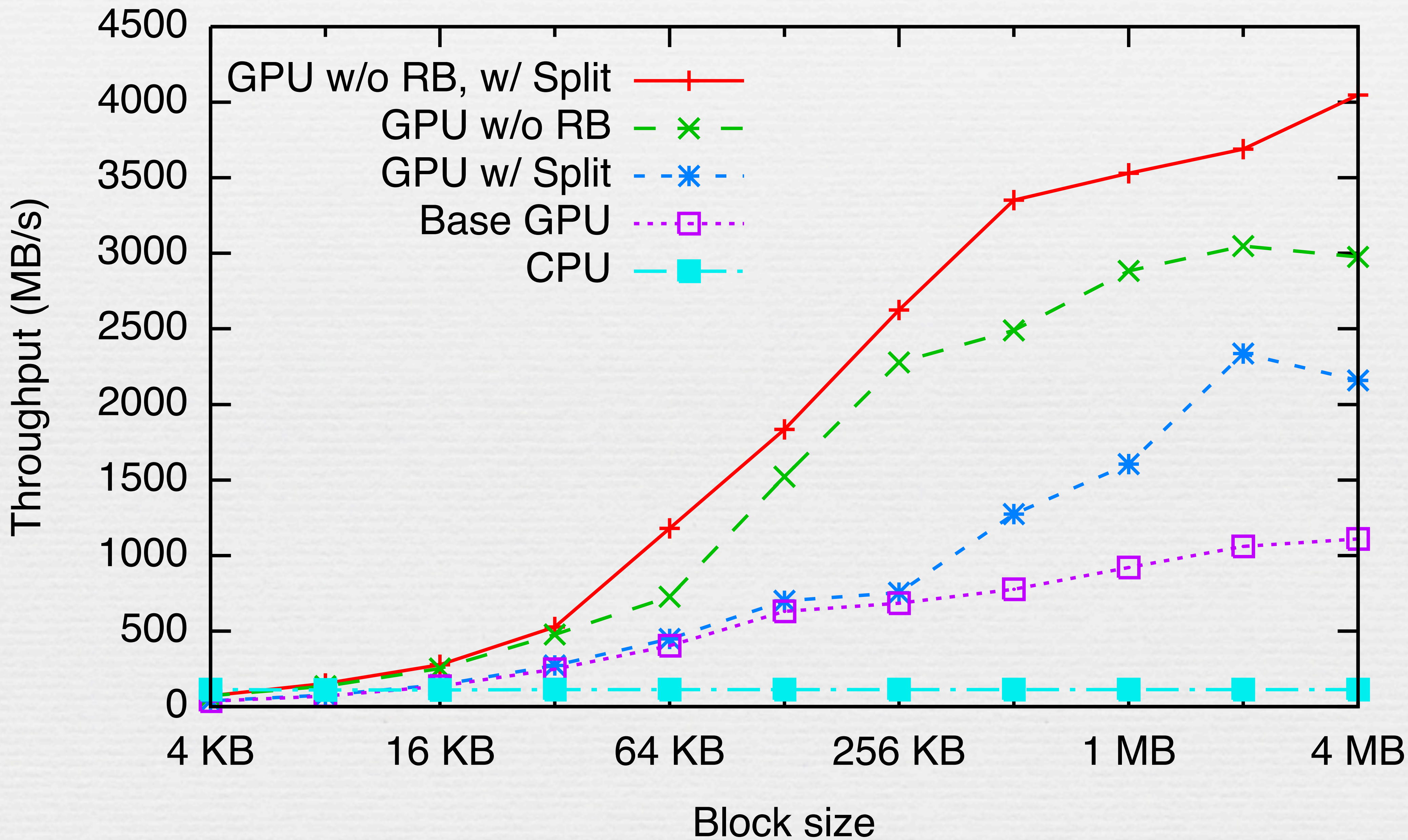
- Small request can't provide enough threads for GPU scheduling
    - To hide mem latency, use all cores
- GPU kernel launch overhead not proportional to req size

Hate too large/small I/O requests?



- Small request can't provide enough threads for GPU scheduling
  - To hide mem latency, use all cores
- GPU kernel launch overhead not proportional to req size
- Large request cause long-time waiting/blocking
  - Hard to avoid, but we can do better
  - GPU is "multi-task-able"
    - Multiple kernel execution
    - Execution and copy overlapping
    - Multiple DMA engines

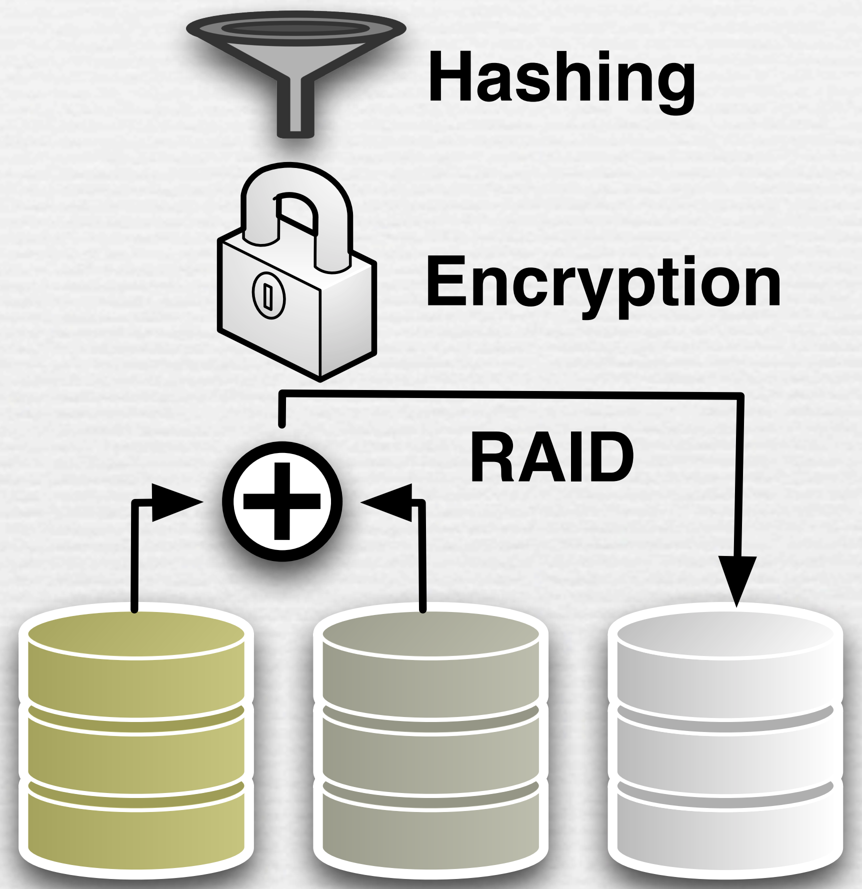
Effectiveness of optimizations



Why GPUstore?

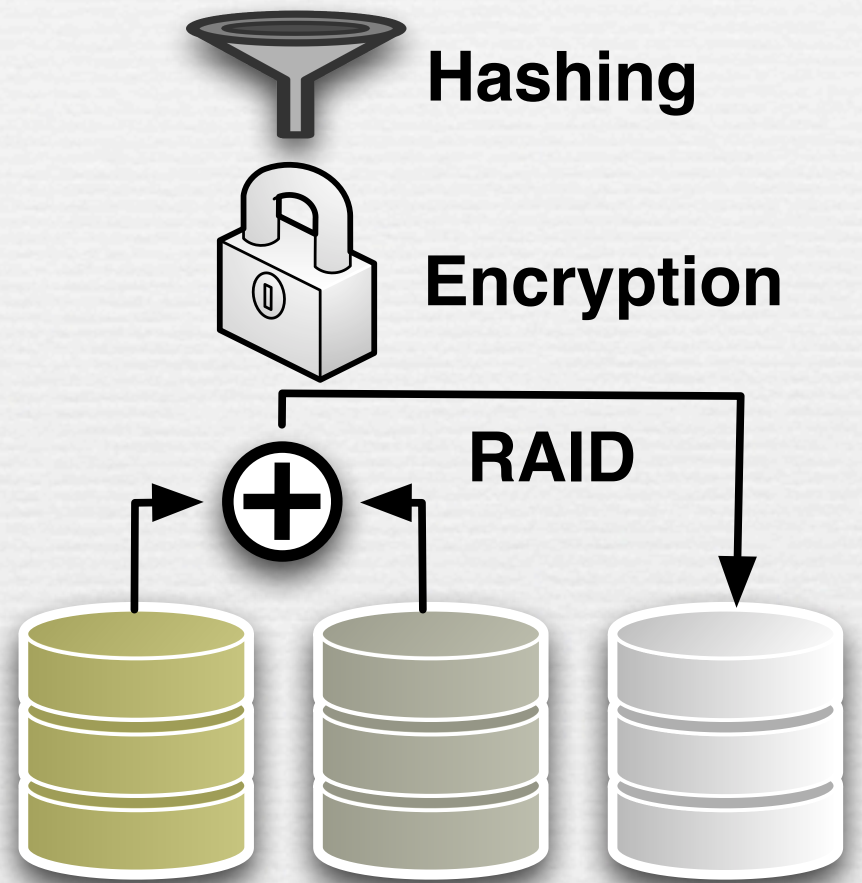
35

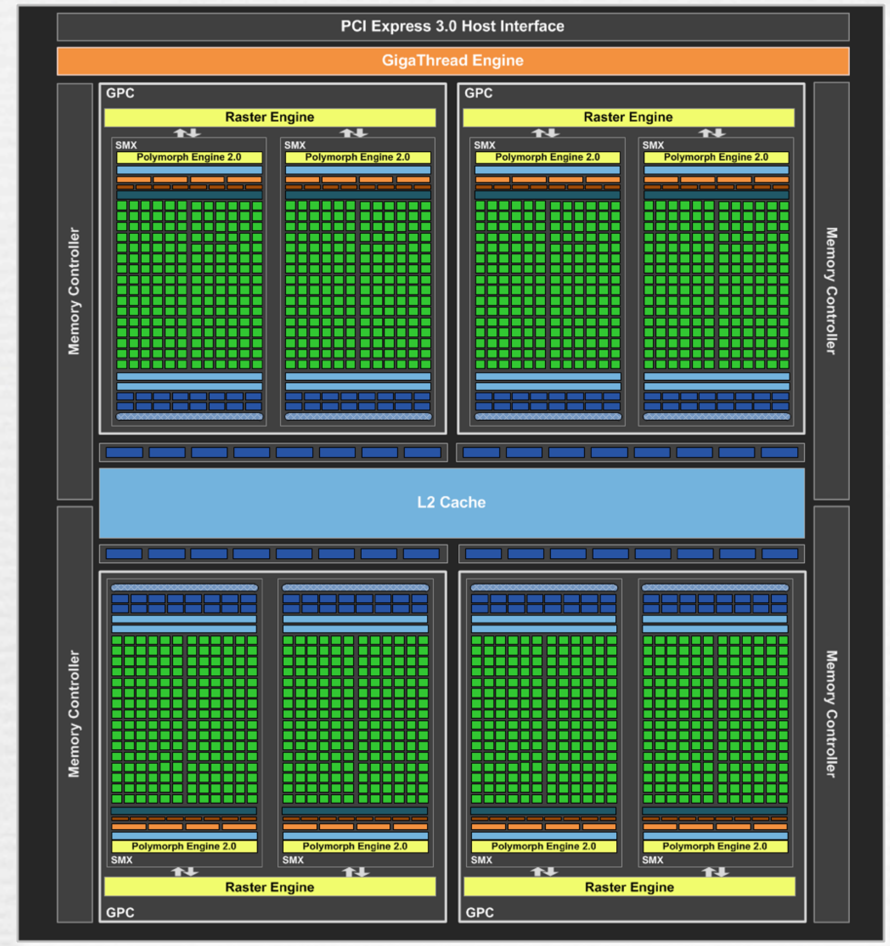
Why GPUstore?



35

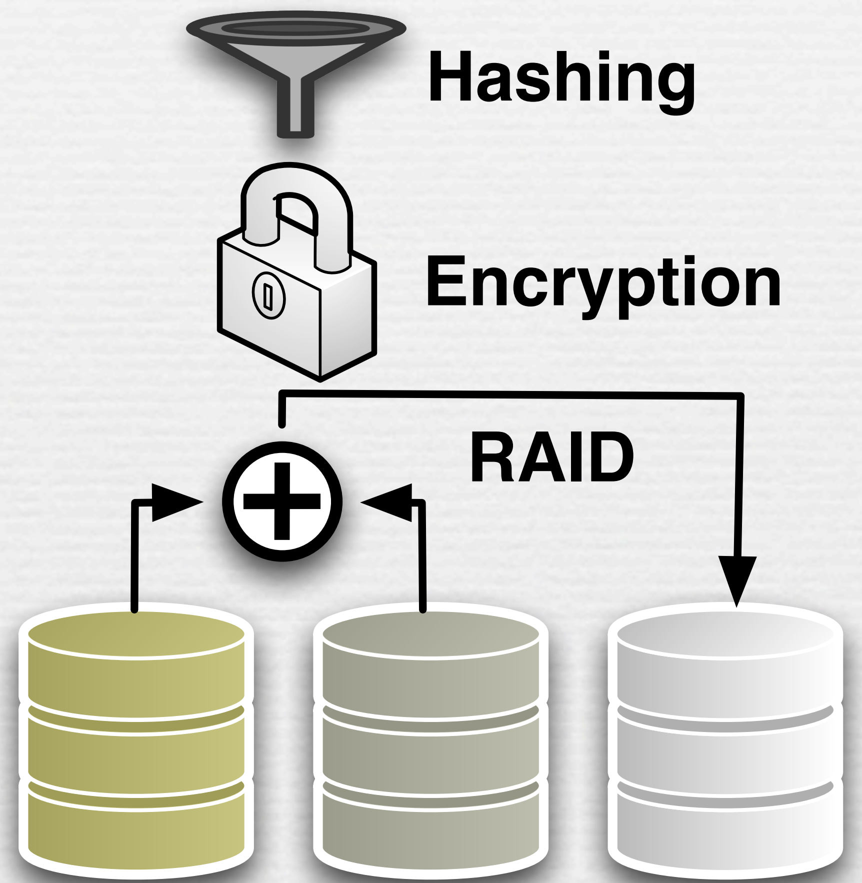
Why GPUstore?

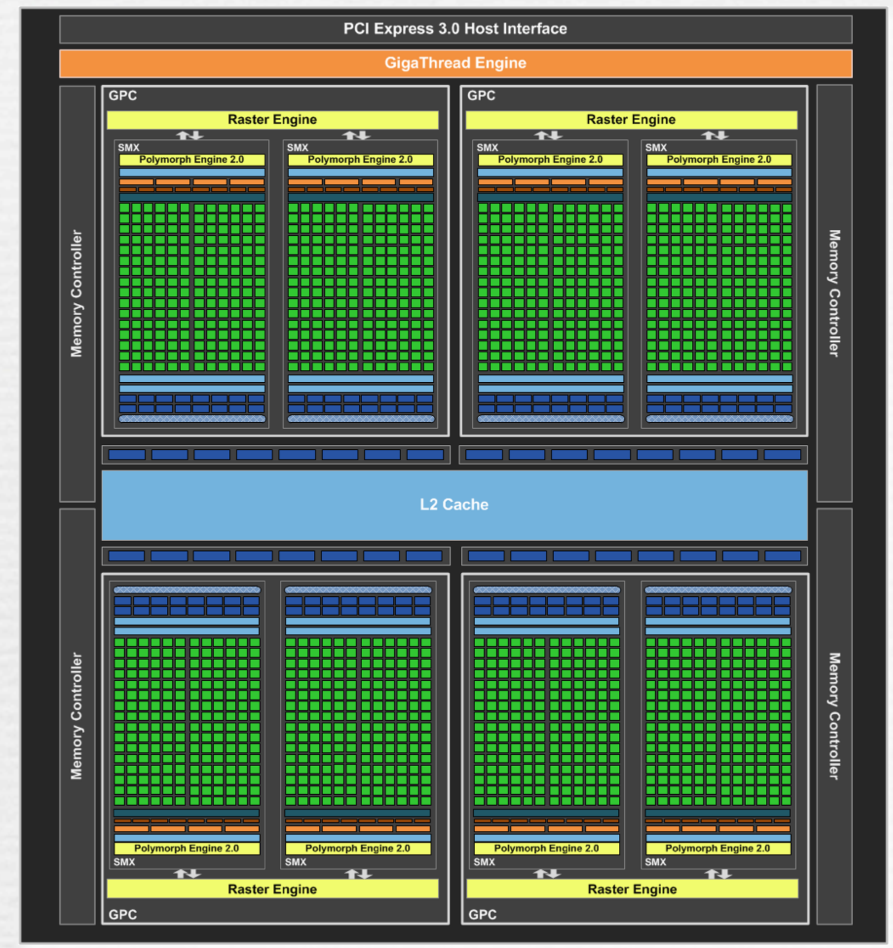




35

Why GPUstore?





Obstacles

- Computation sizes vary a lot, depend I/O sizes
  - Compared with: long-run, large dataset GPGPU
- No optimizations concerning task sizes in CUDA/OpenCL
- Resources (e.g. memory) management - (because storage systems in kernel is special)

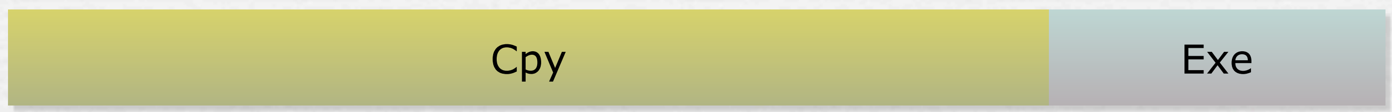
Contribution

- Identified problems in...
  - GPU computing for storage
- A framework:
  - eases integration into existing code
  - REALLY cares about OS kernel

Design - Challenges

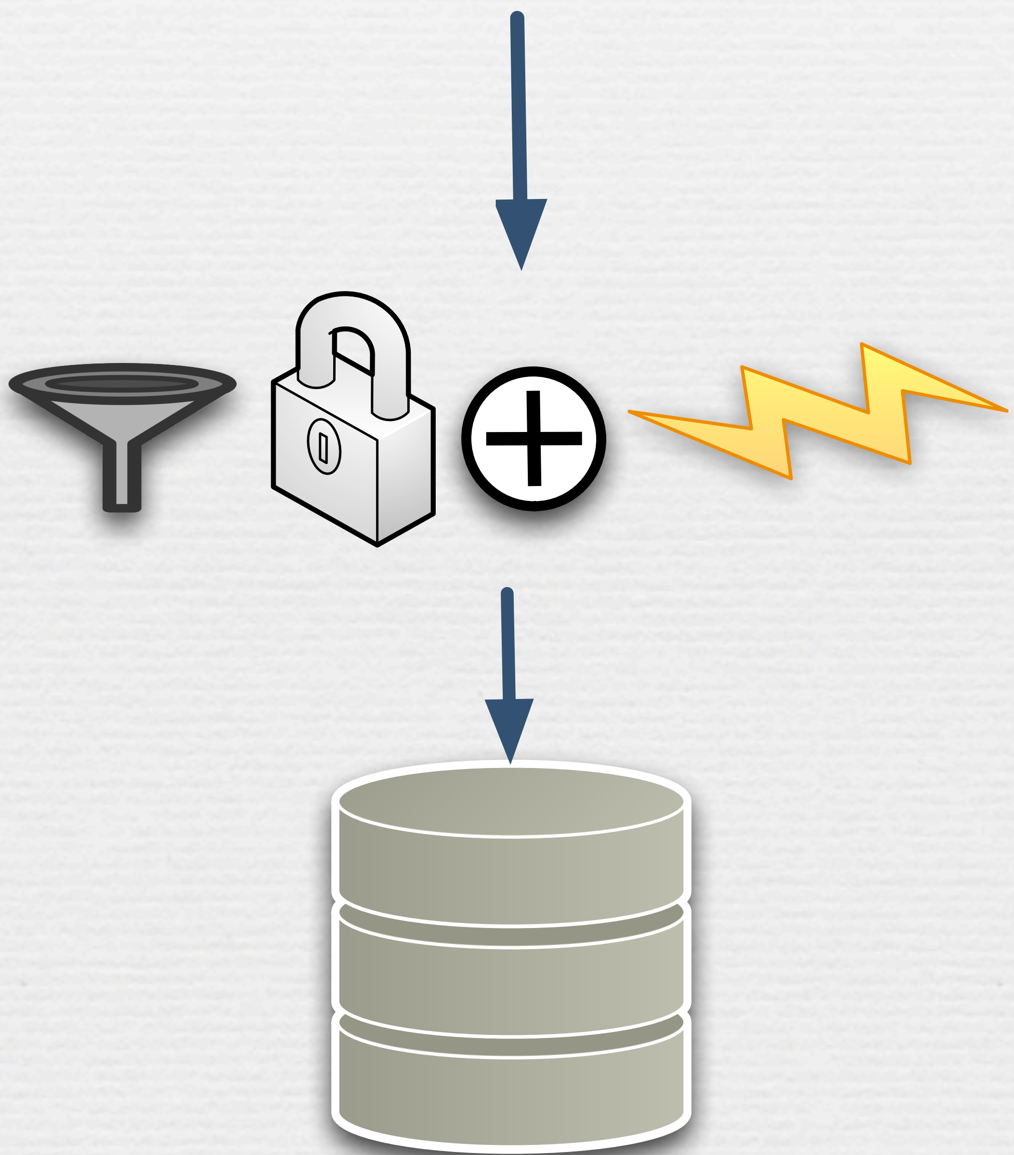
- Asynchrony (maybe not a big deal)
- Redundant buffering
- GPU memcpy and access overhead
- Large dataset
- Managing Resources

Large dataset

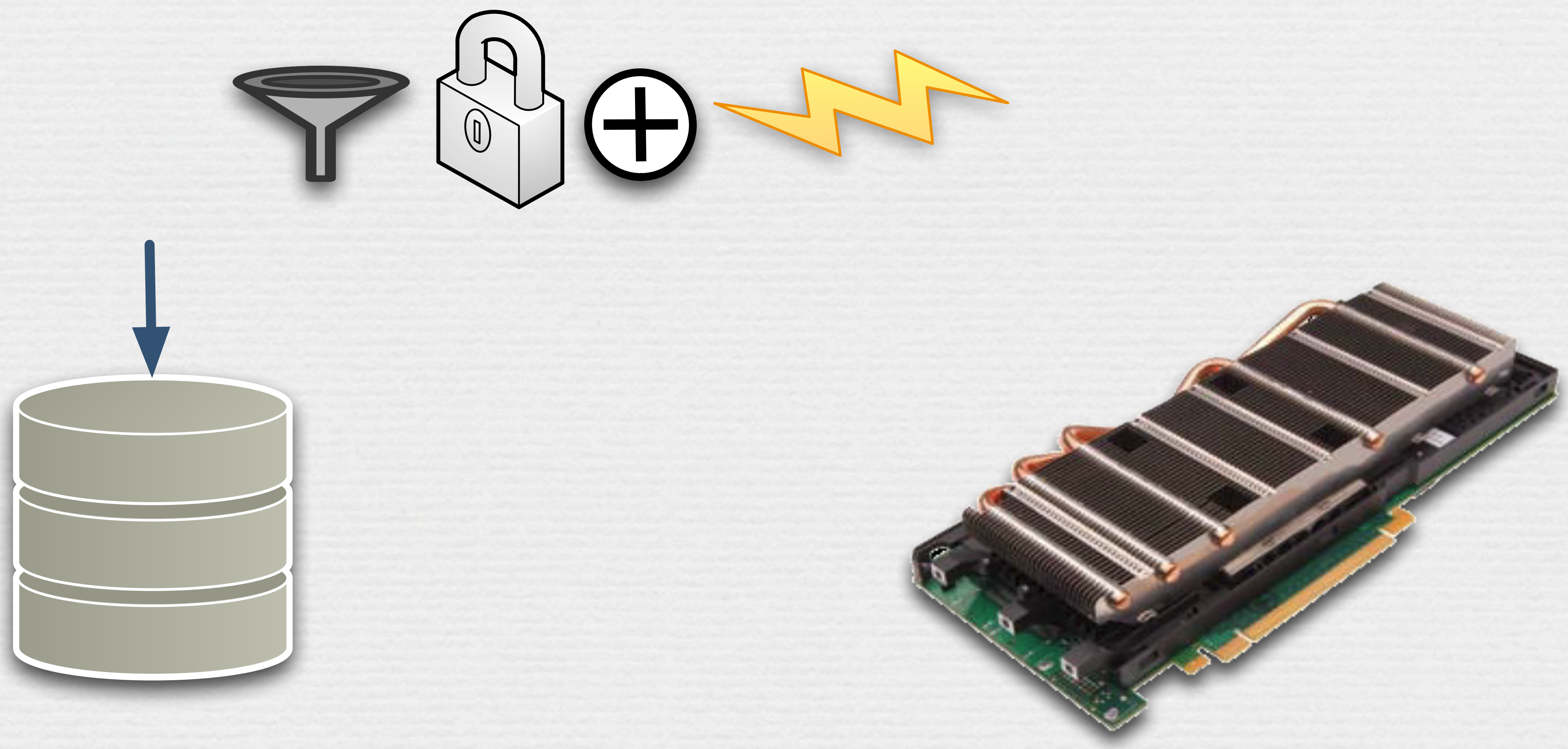


39

GPUstore

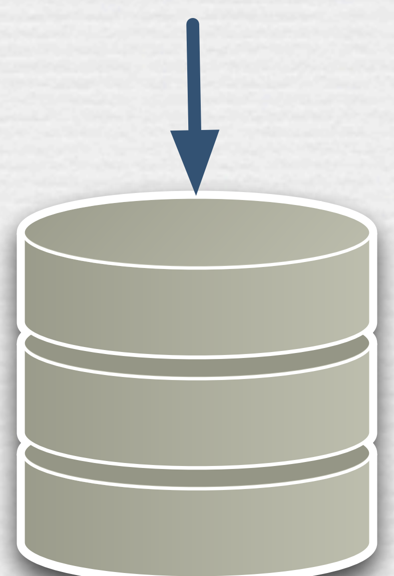


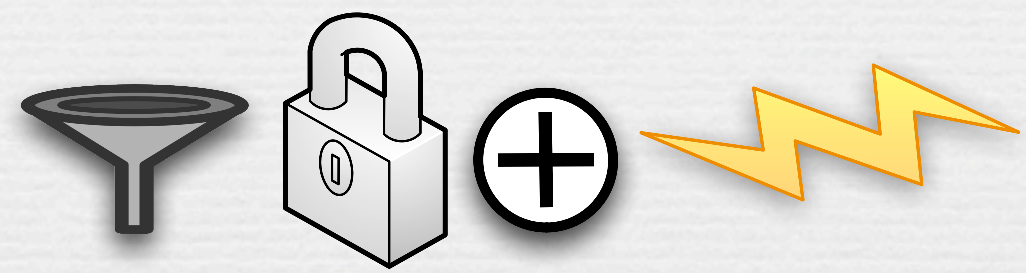
GPUstore

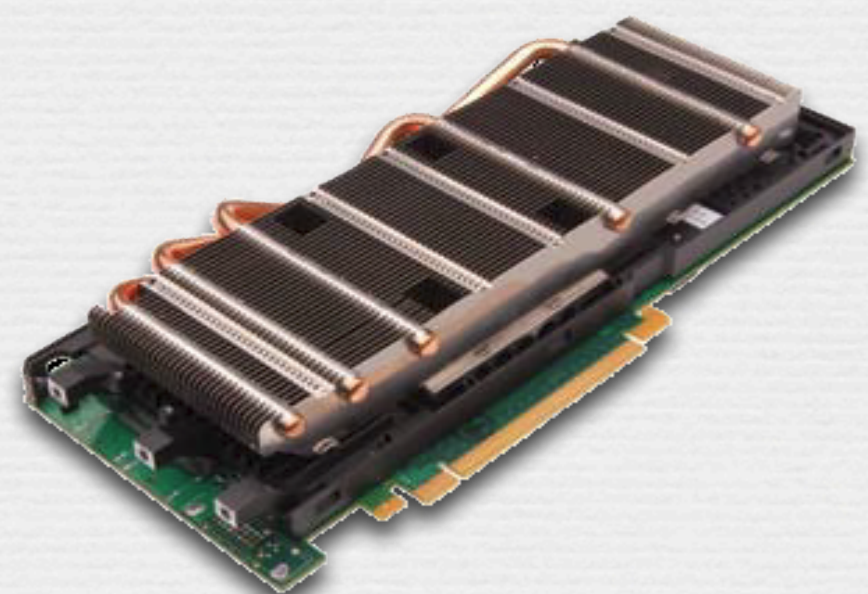


40

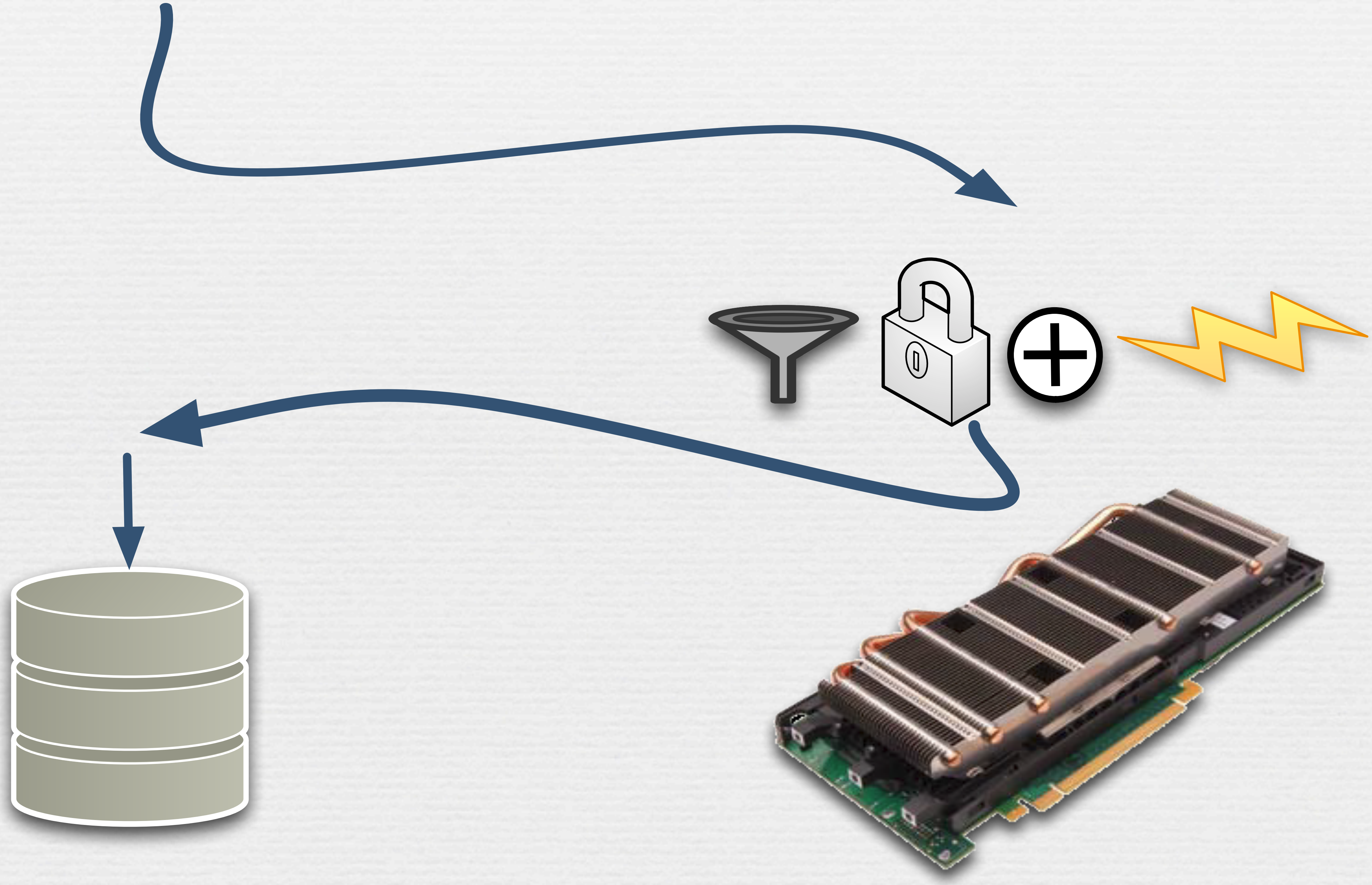
GPUstore







GPUstore



Design

- Request processor
- Functionality -> services

Design - Request Scheduling

- Merge
- Split

Design - Request Scheduling

- Merge
  - To reduce memcpy
  - Hide GPU memory access latency
  - Reduce GPU computing launch overhead

But ...

44

But ...

- Computation size matters

But ...

- Computation size matters
- We want to help existing storage code, in OS kernel, so:

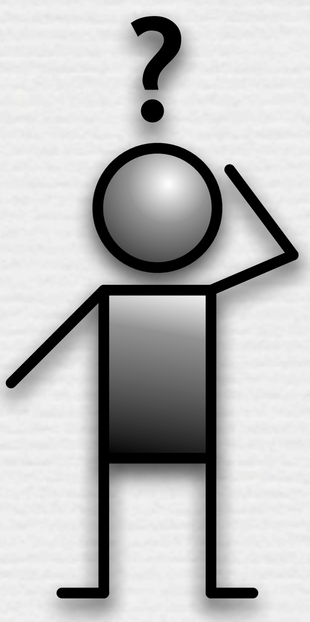
44

But ...

- Computation size matters
- We want to help existing storage code, in OS kernel, so:

Minimum change, maximum performance

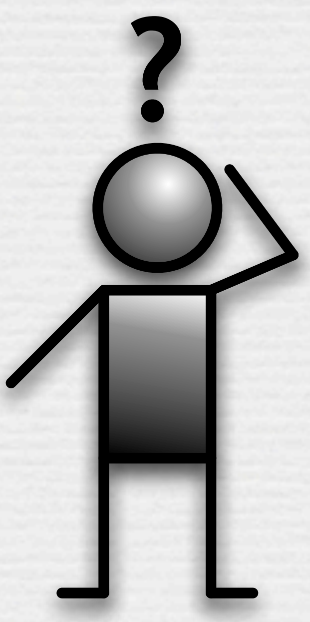
Why computation size matters?



45

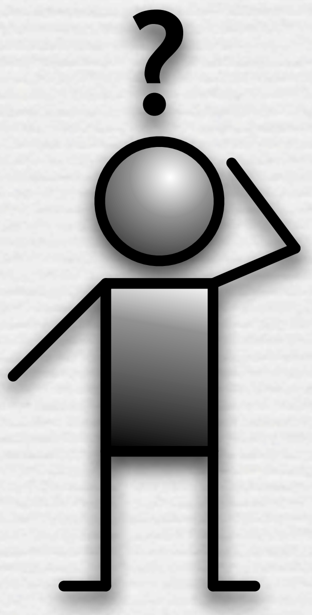
Why computation size matters?

- Size ∝ Time - so not too large



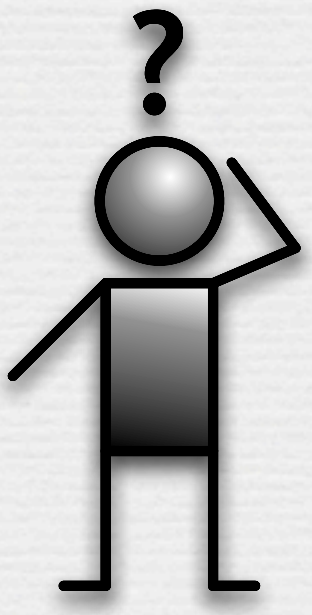
Why computation size matters?

- Size $\propto$ Time - so not too large
  - Frameworks of ***General***-purpose GPU not quite general



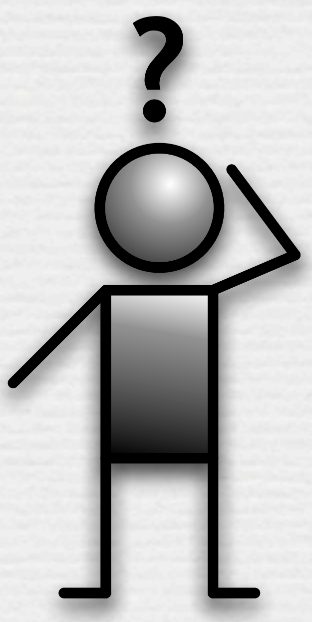
Why computation size matters?

- Size ∝ Time - so not too large
  - Frameworks of ***General***-purpose GPU not quite general
    - Large data/Long run scientific computing/ HPC

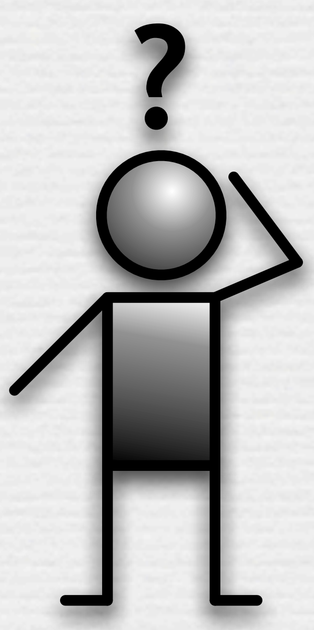


Why computation size matters?

- Size ∝ Time - so not too large
  - Frameworks of ***General***-purpose GPU not quite general
    - Large data/Long run scientific computing/ HPC
    - Kernel storage code should be latency-aware



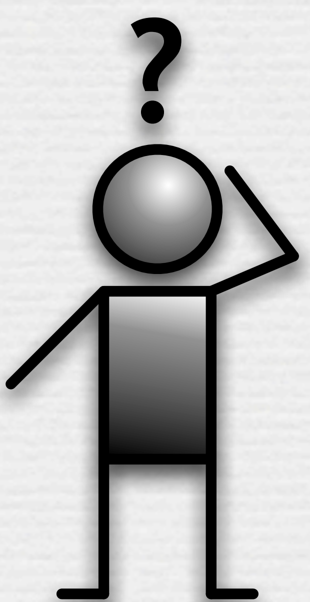
Why computation size matters?



46

Why computation size matters?

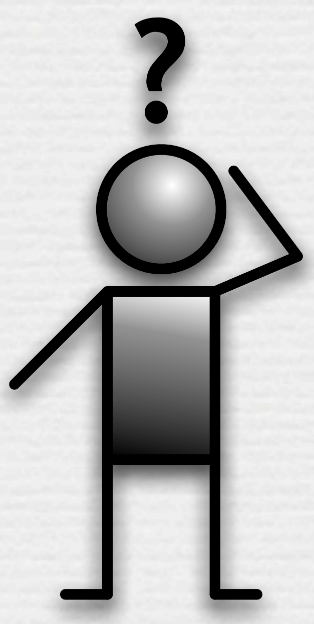
- Size ∝ Number of threads - so not too small



46

Why computation size matters?

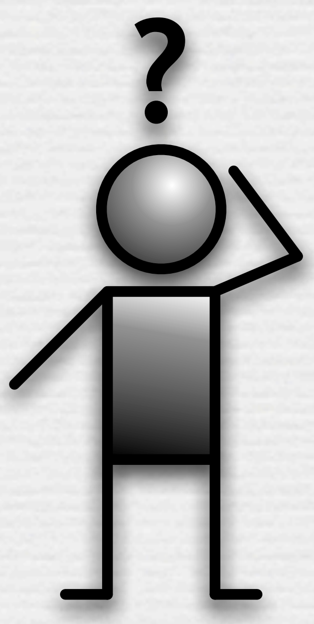
- Size ∝ Number of threads - so not too small
  - Why GPU faster?



46

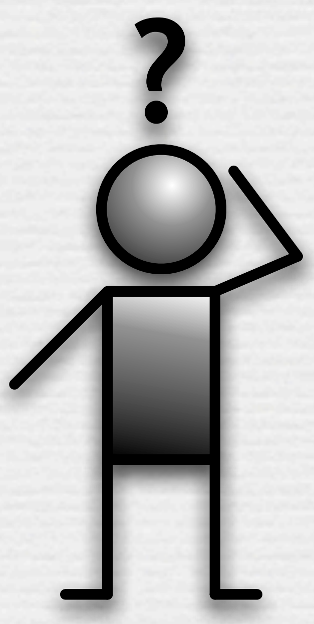
Why computation size matters?

- Size ∝ Number of threads - so not too small
  - Why GPU faster?
    - Single wimpy GPU core



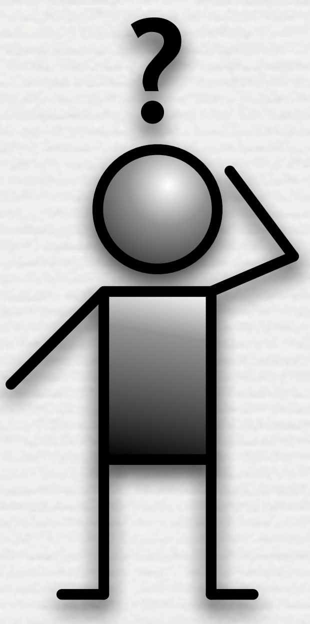
Why computation size matters?

- Size ∝ Number of threads - so not too small
  - Why GPU faster?
    - Single wimpy GPU core
    - But there could be more than 1000 cores



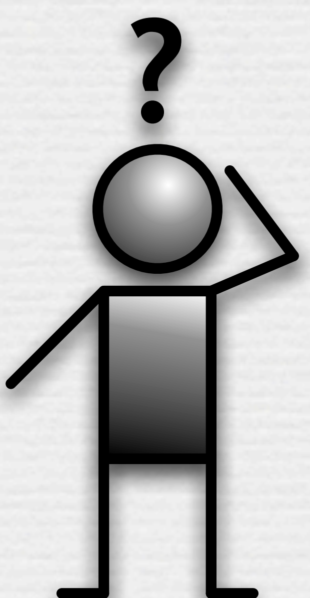
Why computation size matters?

- Size ∝ Number of threads - so not too small
  - Why GPU faster?
    - Single wimpy GPU core
    - But there could be more than 1000 cores
    - Long latency GPU memory (Global)



Why computation size matters?

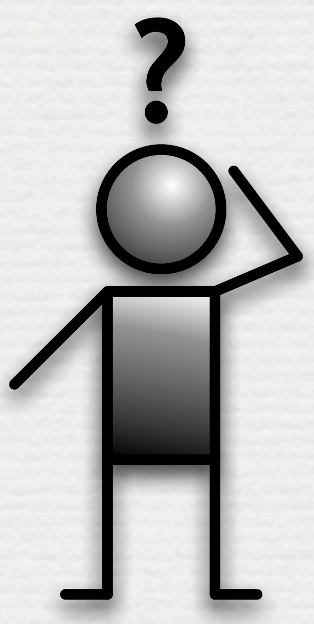
- Size ∝ Number of threads - so not too small
  - Why GPU faster?
    - Single wimpy GPU core
    - But there could be more than 1000 cores
    - Long latency GPU memory (Global)
    - But GPU hides mem access latency by scheduling ...



46

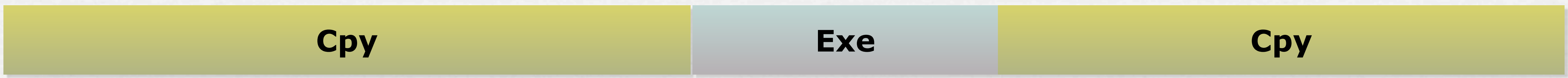
Why computation size matters?

- Size ∝ Number of threads - so not too small
  - Why GPU faster?
    - Single wimpy GPU core
    - But there could be more than 1000 cores
    - Long latency GPU memory (Global)
    - But GPU hides mem access latency by scheduling ...
  - GPU kernel launch overhead



46

Too large computation



47

Too large computation

Too large computation

- GPU can do:

Too large computation

- GPU can do:
  - Bidirectional DMA with multiple DMA engines

Too large computation

- GPU can do:
    - Bidirectional DMA with multiple DMA engines
    - Overlapped copy and execution

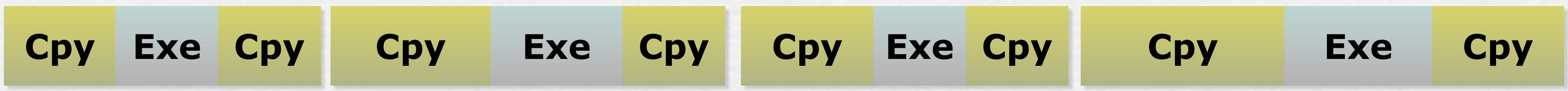
47

Too large computation

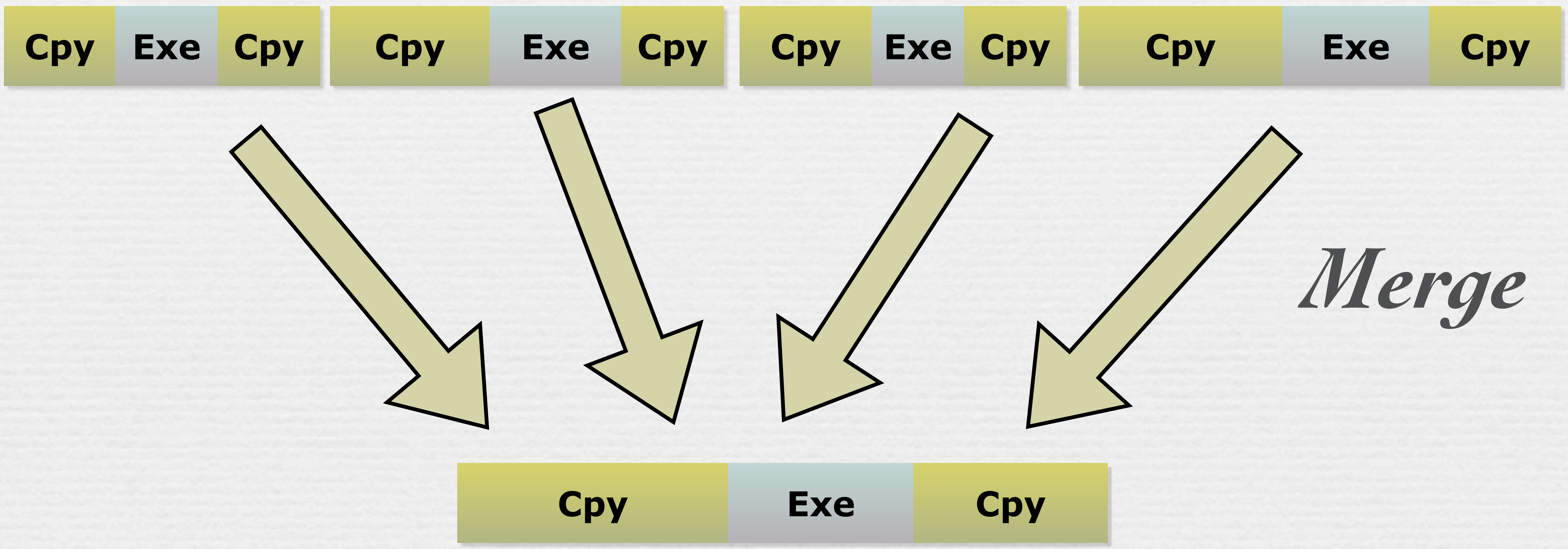
- GPU can do:
  - Bidirectional DMA with multiple DMA engines
  - Overlapped copy and execution
  - Do more than one computation at the same time

Too small computation

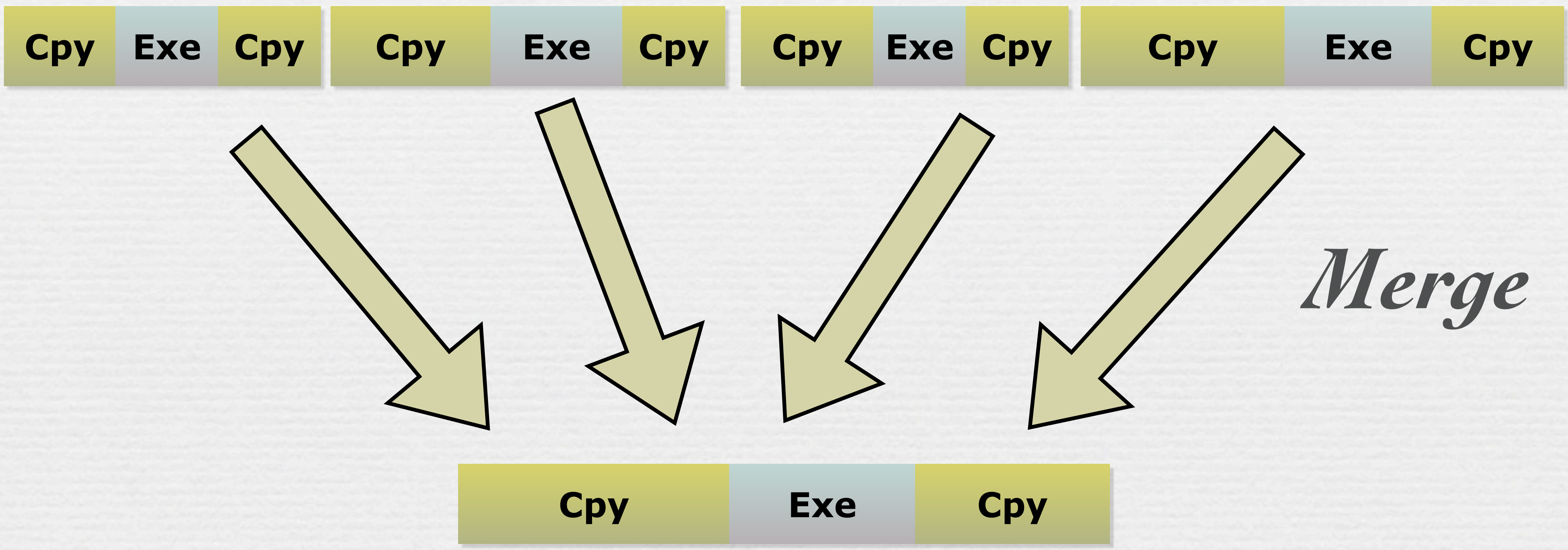
Too small computation



Too small computation



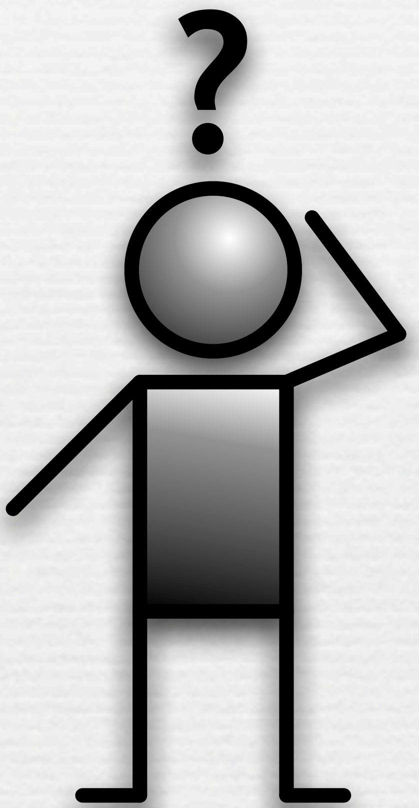
Too small computation



Merge is not linear addition

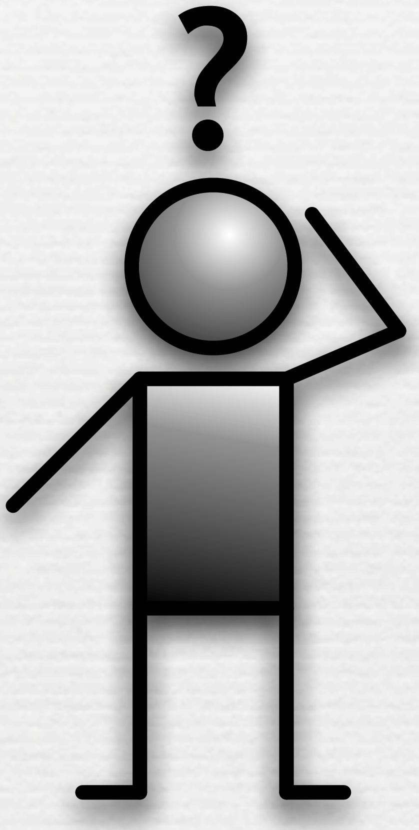
Okay, but

Okay, but



What is the RIGHT size?

Okay, but



What is the RIGHT size?

How to logical-correctly merge/split?

GPUstore design

- Functionality ➡ Service (simply, code on GPU)
- Using functionality ➡ Request to service

Service-specific request scheduling

Service-specific request scheduling

- Right size:

Service-specific request scheduling

- Right size:
  - Decided by service itself

51

Service-specific request scheduling

- Right size:
  - Decided by service itself
    - Boot-time benchmark, service logic, computing features...

Service-specific request scheduling

- Right size:
  - Decided by service itself
    - Boot-time benchmark, service logic, computing features...
- Merge/Split logic:

Service-specific request scheduling

- Right size:
  - Decided by service itself
    - Boot-time benchmark, service logic, computing features...
- Merge/Split logic:
  - Done by service too, or simply nothing

The other “But”

The other “But”

- Existing code - no big change

The other "But"

- Existing code - no big change
    - In OS kernel: not always be able to control mem allocation - redundant buffering

The other "But"

- Existing code - no big change
  - In OS kernel: not always be able to control mem allocation - redundant buffering
  - Use functionality as function call

The other "But"

- Existing code - no big change
  - In OS kernel: not always be able to control mem allocation - redundant buffering
  - Use functionality as function call
    - Simple: move complex work to service, service as a function call

One more thing

- Other GPU resources
  - Copy engine
  - Overlapping cpy/exec capability

One more thing

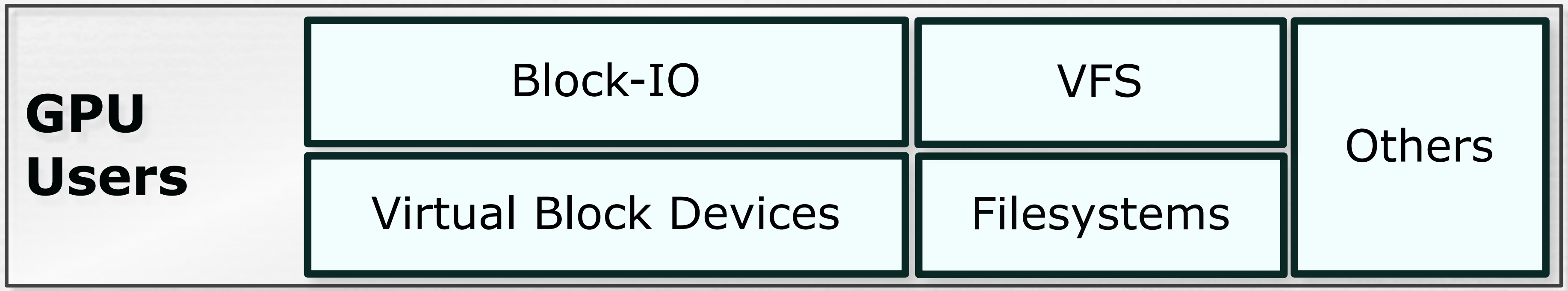
- Other GPU resources
  - Copy engine
  - Overlapping cpy/exec capability

Streams

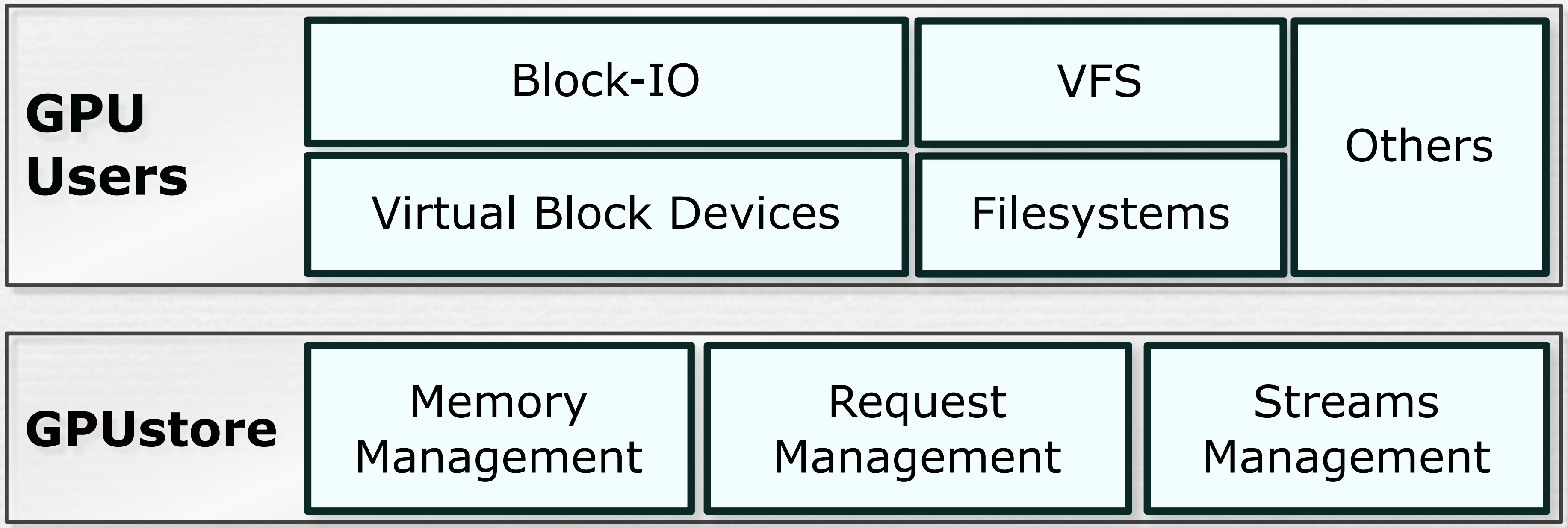
53

GPUstore

GPUstore

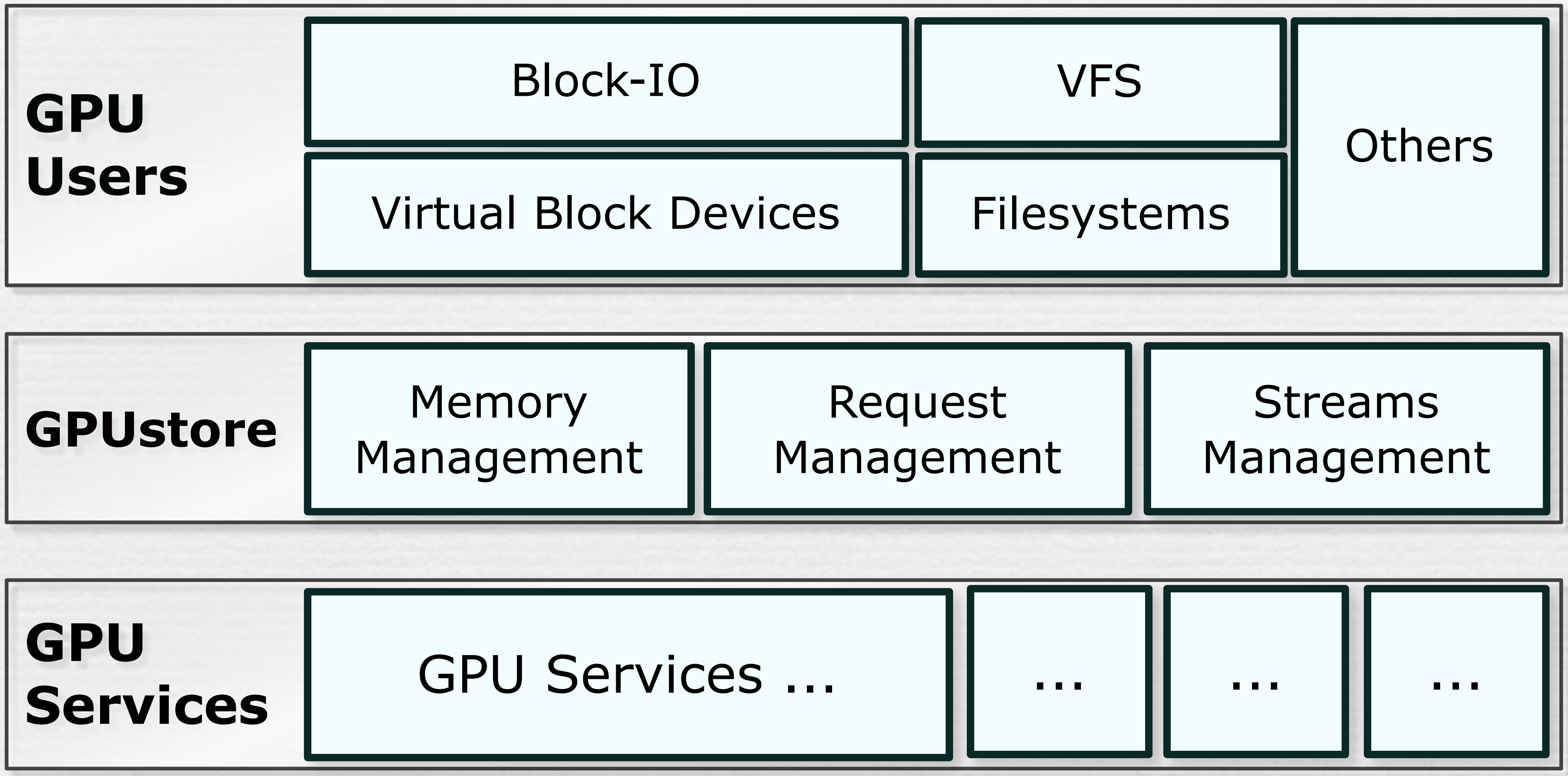

GPU Users
Block-IO
VFS
Virtual Block Devices
Filesystems
Others

GPUstore

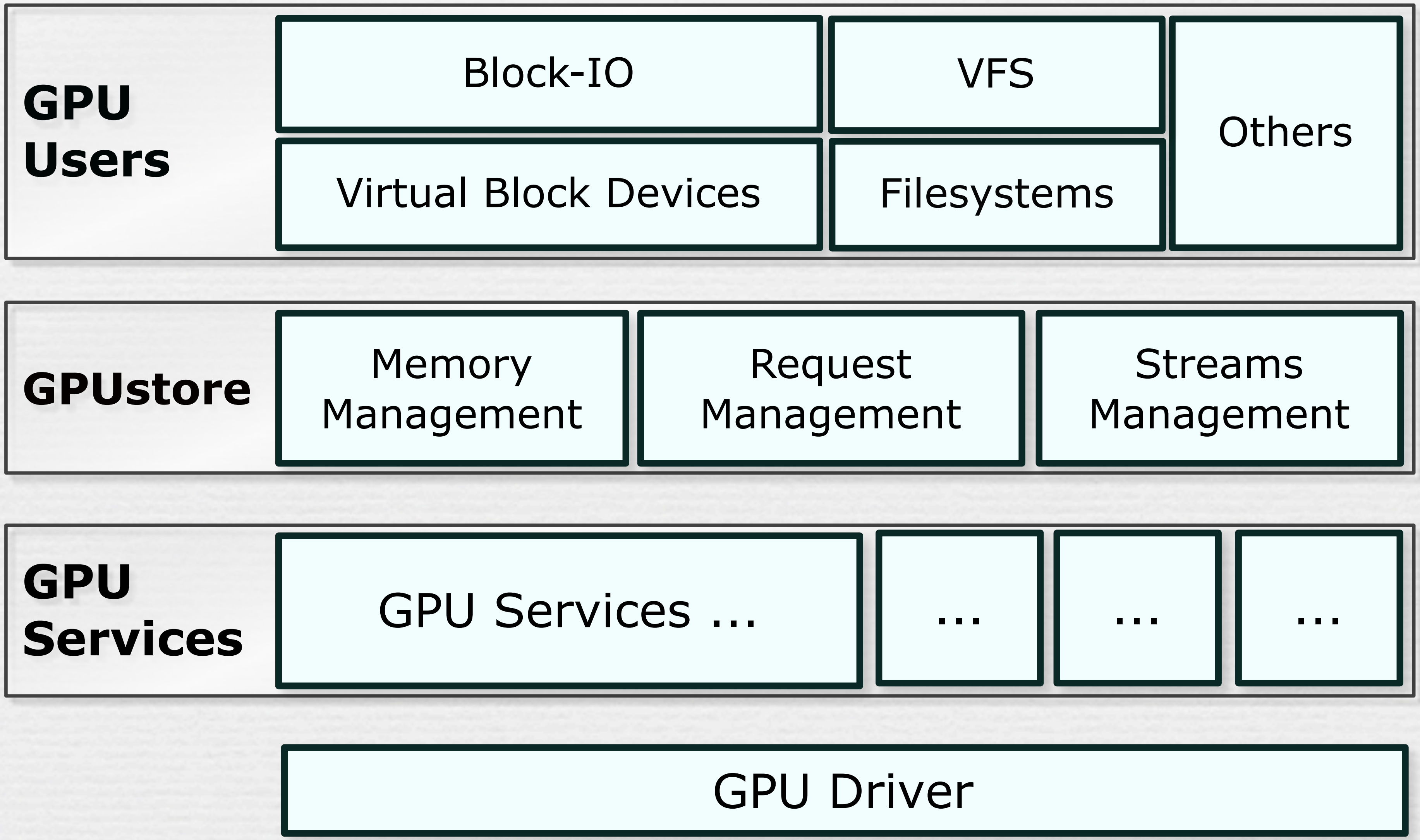


GPU Users
Block-IO
VFS
Others
Virtual Block Devices
Filesystems
GPUstore
Memory Management
Request Management
Streams Management

GPUstore

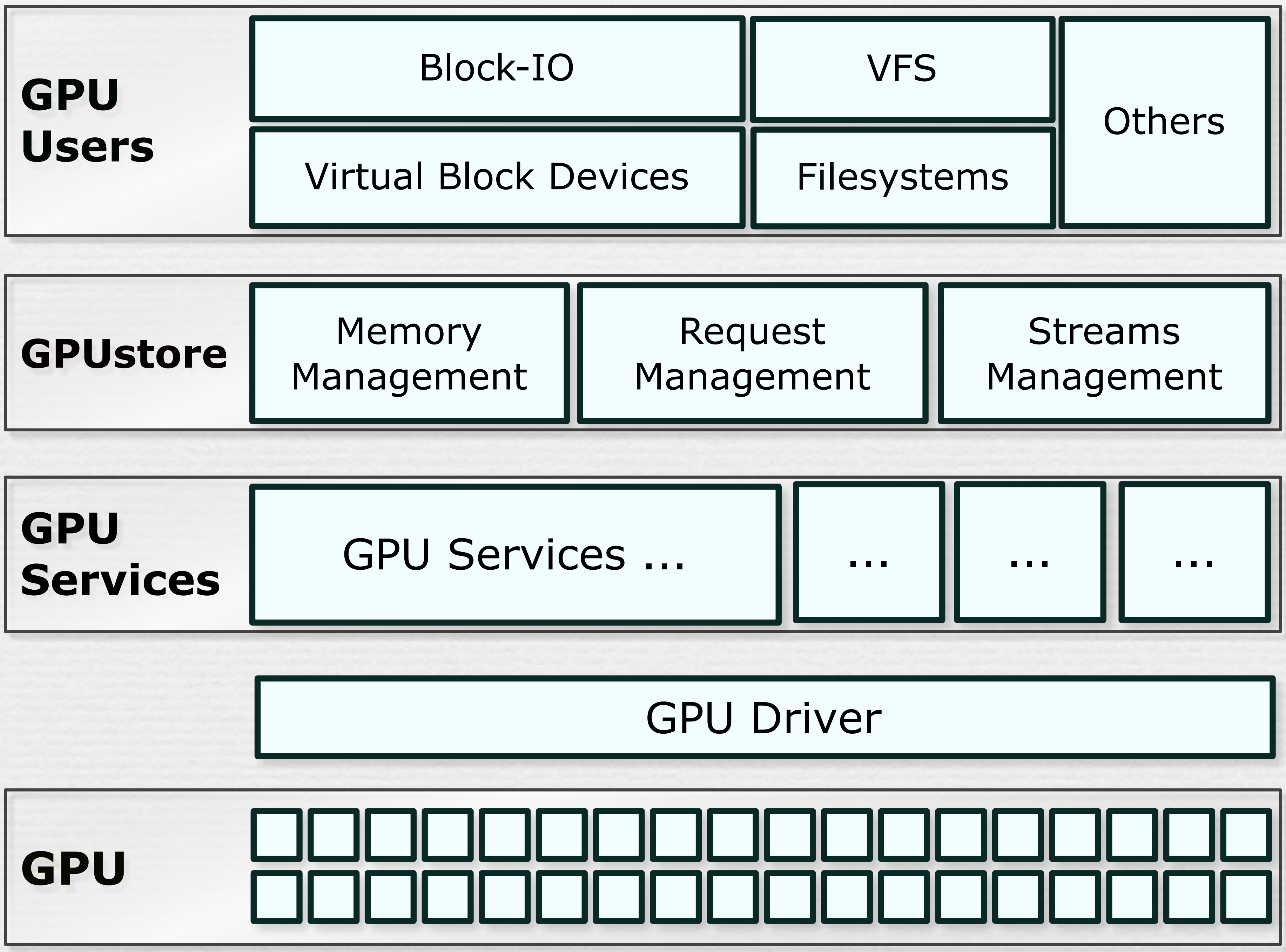

GPU Users
Block-IO
VFS
Others
Virtual Block Devices
Filesystems
GPUstore
Memory Management
Request Management
Streams Management
GPU Services
GPU Services ...
...
...
...

GPUstore



GPU Users
Block-IO
VFS
Others
Virtual Block Devices
Filesystems
GPUstore
Memory Management
Request Management
Streams Management
GPU Services
GPU Services ...
...
...
...
GPU Driver

GPUstore



GPU Users
Block-IO
VFS
Others
Virtual Block Devices
Filesystems
GPUstore
Memory Management
Request Management
Streams Management
GPU Services
GPU Services ...
...
...
...
GPU Driver
GPU

54

Design - Request Scheduling

- Split
  - Efficiently using DMA engines
  - Overlapping execution
  - Newer GPU (Fermi)/Multi-GPU preferred

Why merge and split?

- Better performance
- (more fundamental)Simplify GPU computing integration
  - Code deals with single page, single block, small buffers
  - Code accepts a buffer as whole and throws it onto GPU

Design - Memory Management

- Allocating memory from GPU computing runtime
- Using memory from existing code
  - Remapping them

Design - Resource Management

- Managing execution and copy through 'streams' in CUDA
  - First come first serve now
- No preemption currently

Evaluation

- eCryptfs
- dm-crypt
- md RAID 6 (see the paper)