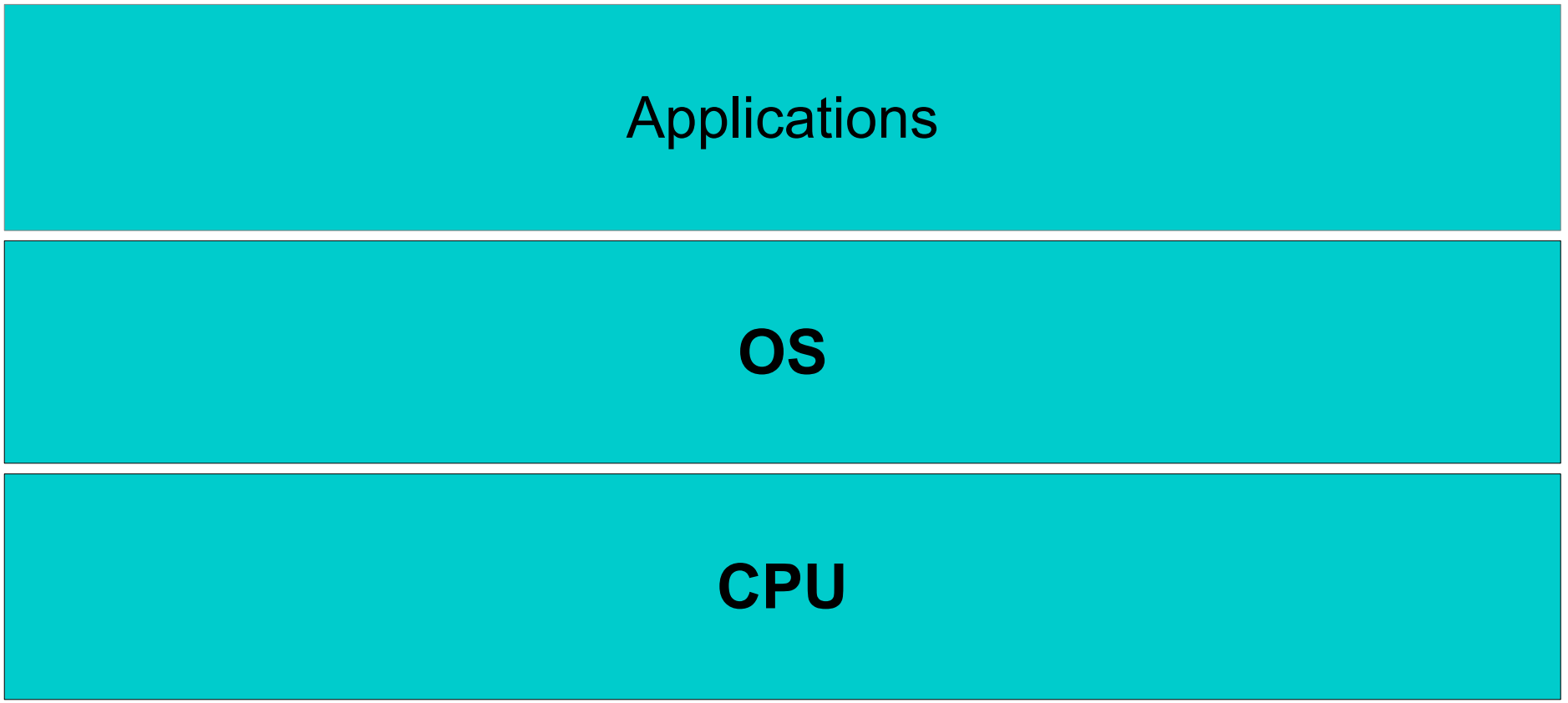
GPUfs: Integrating a file system with GPUs

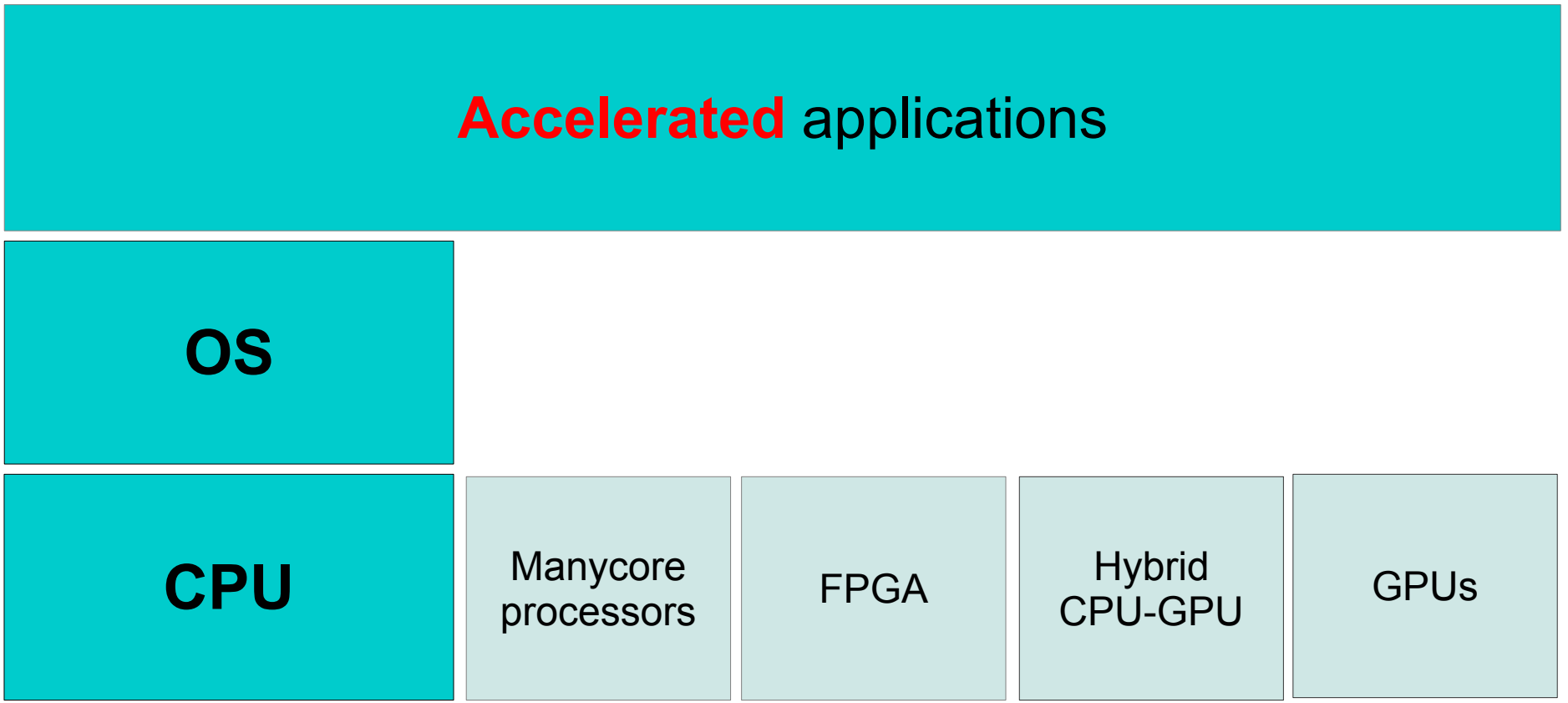
Mark Silberstein
(UT Austin/Technion)

Bryan Ford (Yale), Idit Keidar (Technion)
Emmett Witchel (UT Austin)

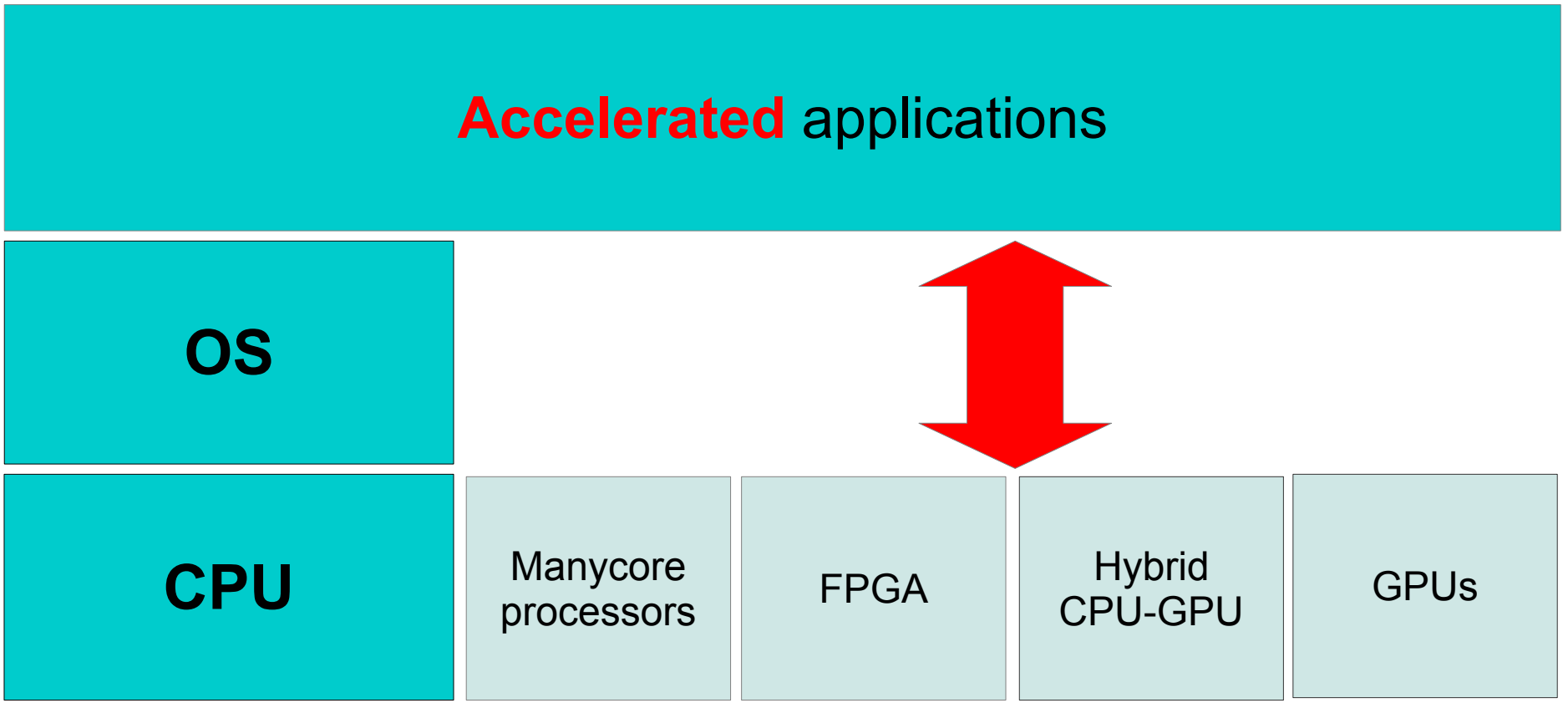
Traditional System Architecture



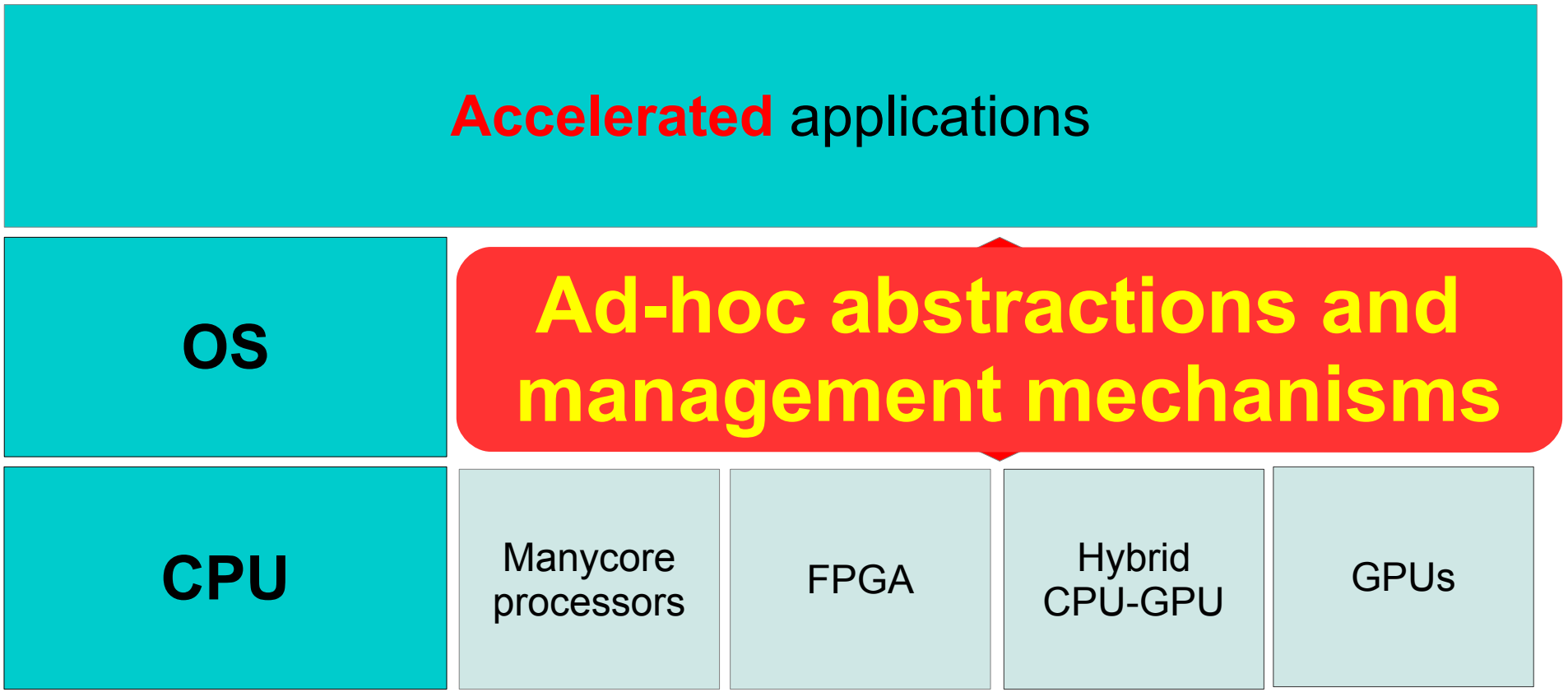
Modern System Architecture



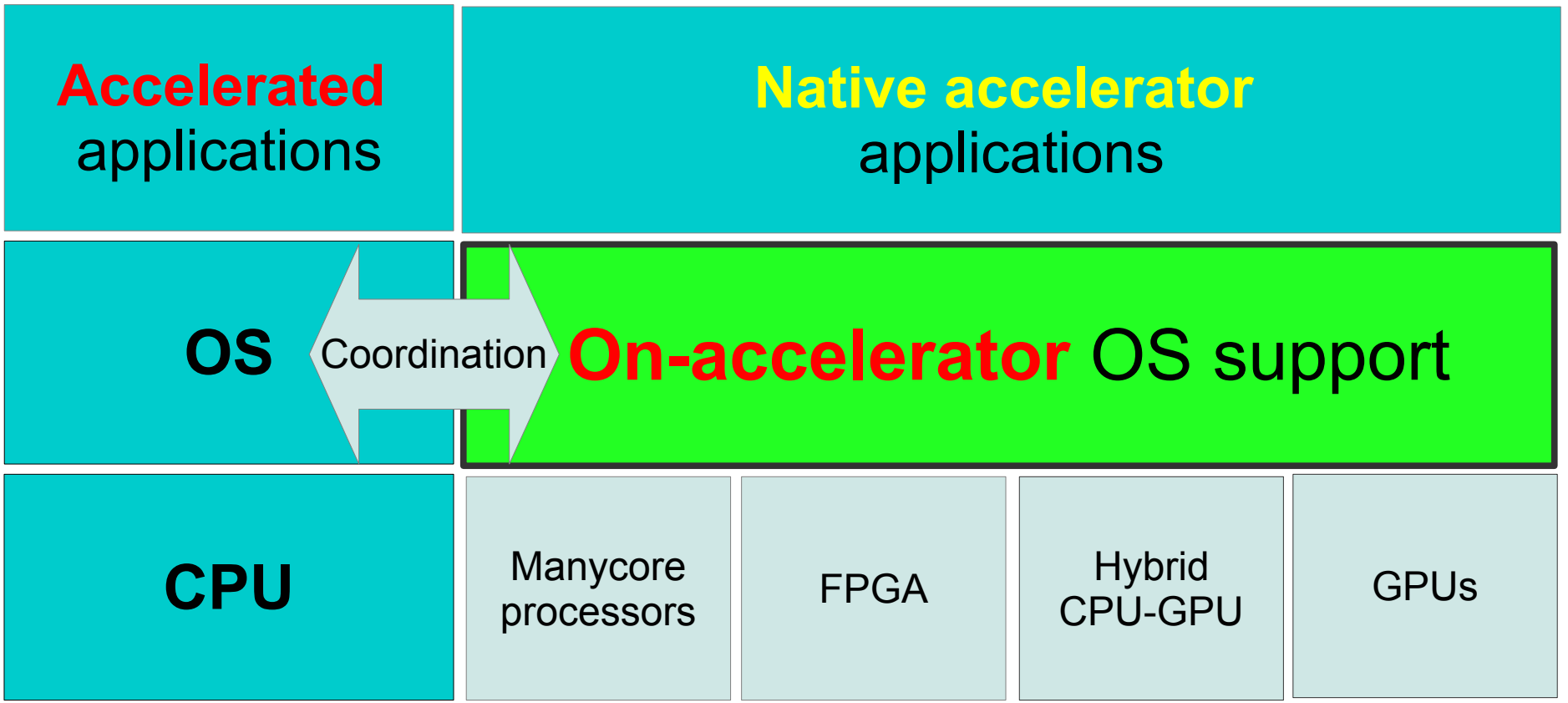
Software-hardware gap is widening



Software-hardware gap is widening



On-accelerator OS support closes the programmability gap

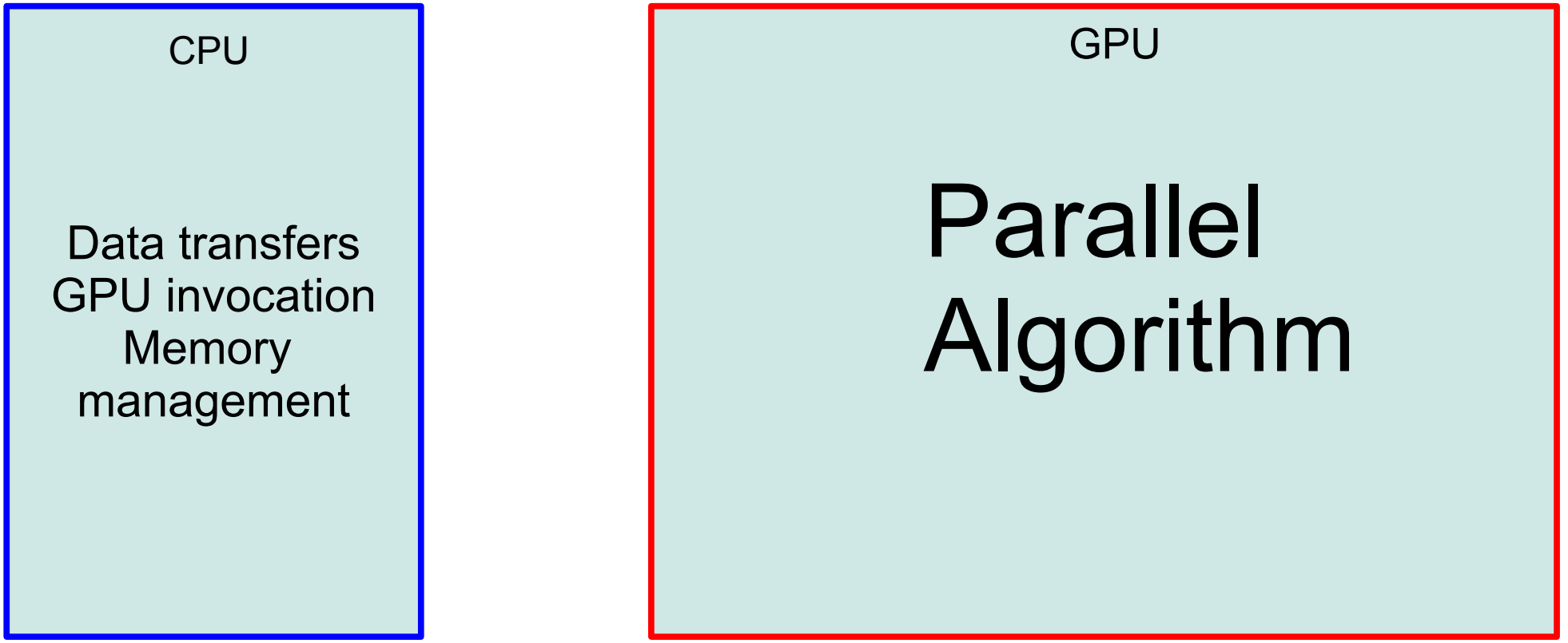


- **GPUfs: File I/O support for GPUs**
  - Motivation
  - Goals
  - Understanding the hardware
  - Design
  - Implementation
  - Evaluation

Building systems with GPUs is hard.

Why?

Goal of GPU programming frameworks



Headache for GPU programmers

CPU

Data transfers
Invocation
Memory management

GPU

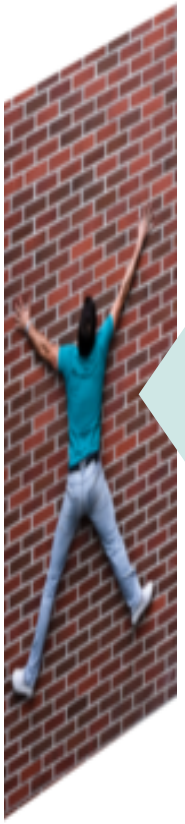
Parallel
Algorithm

Half of the CUDA SDK 4.1 samples:
at least **9 CPU LOC per 1 GPU LOC**

GPU kernels are isolated

CPU

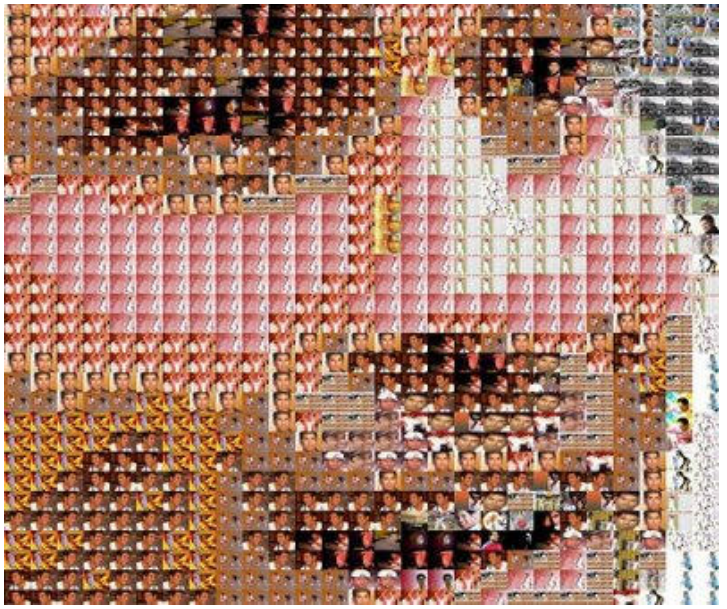
Data transfers
Invocation
Memory management



GPU

Parallel
Algorithm

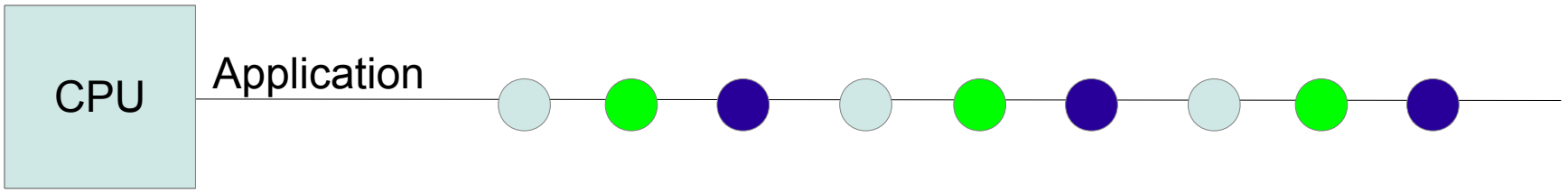
Example: accelerating photo collage



http://www.codeproject.com/Articles/36347/Face-Collage

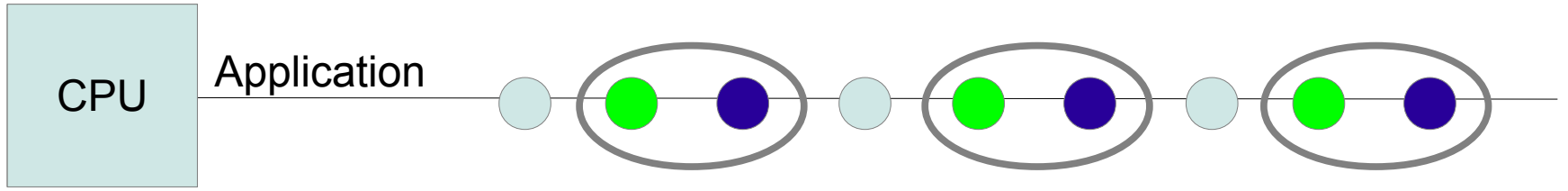
```
While(Unhappy()){
   Read_next_image_file()
   Decide_placement()
   Remove_outliers()
}
```

CPU Implementation



```
While(Unhappy()){
   Read_next_image_file()
   Decide_placement()
   Remove_outliers()
}
```

Offloading computations to GPU

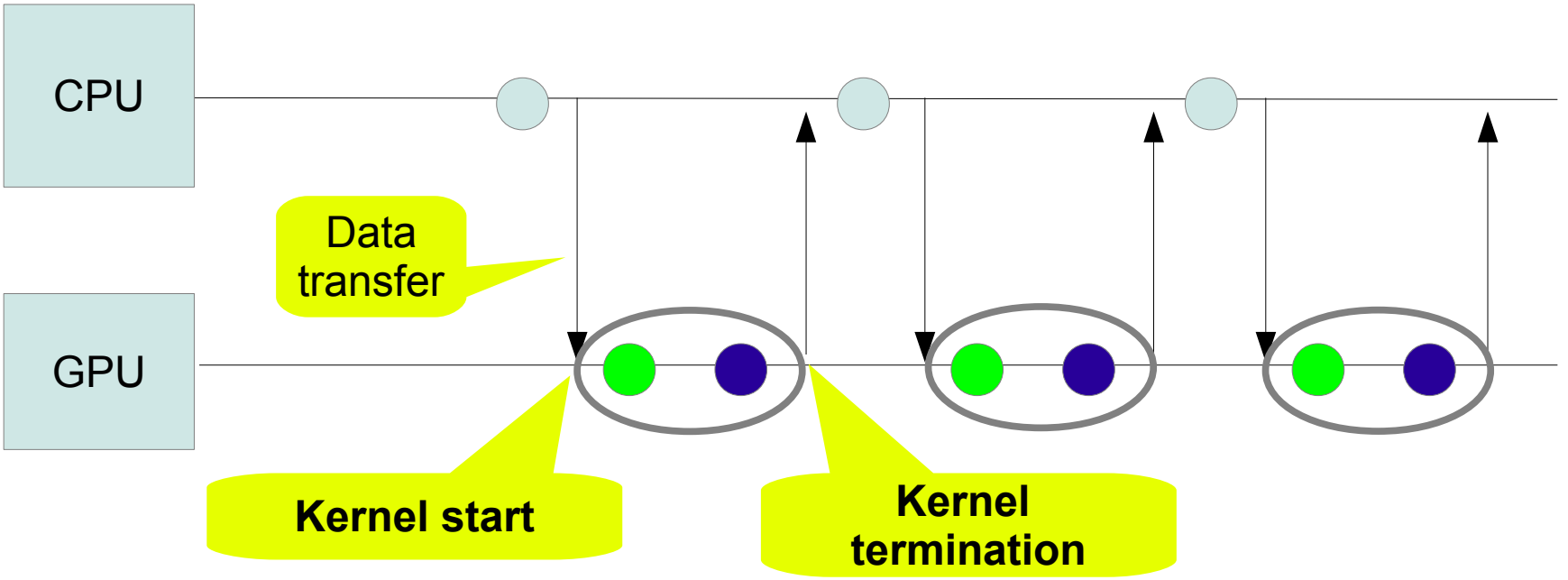


Move to GPU

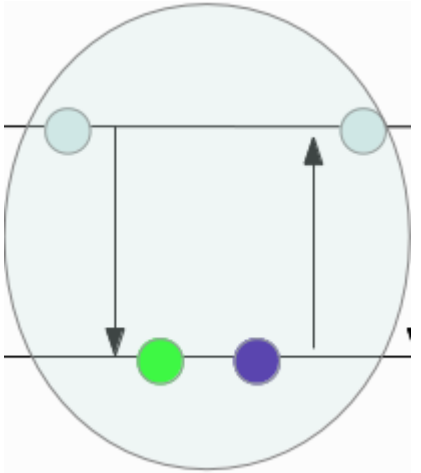
```
While(Unhappy()){
   Read_next_image_file()
   Decide_placement()
   Remove_outliers()
}
```

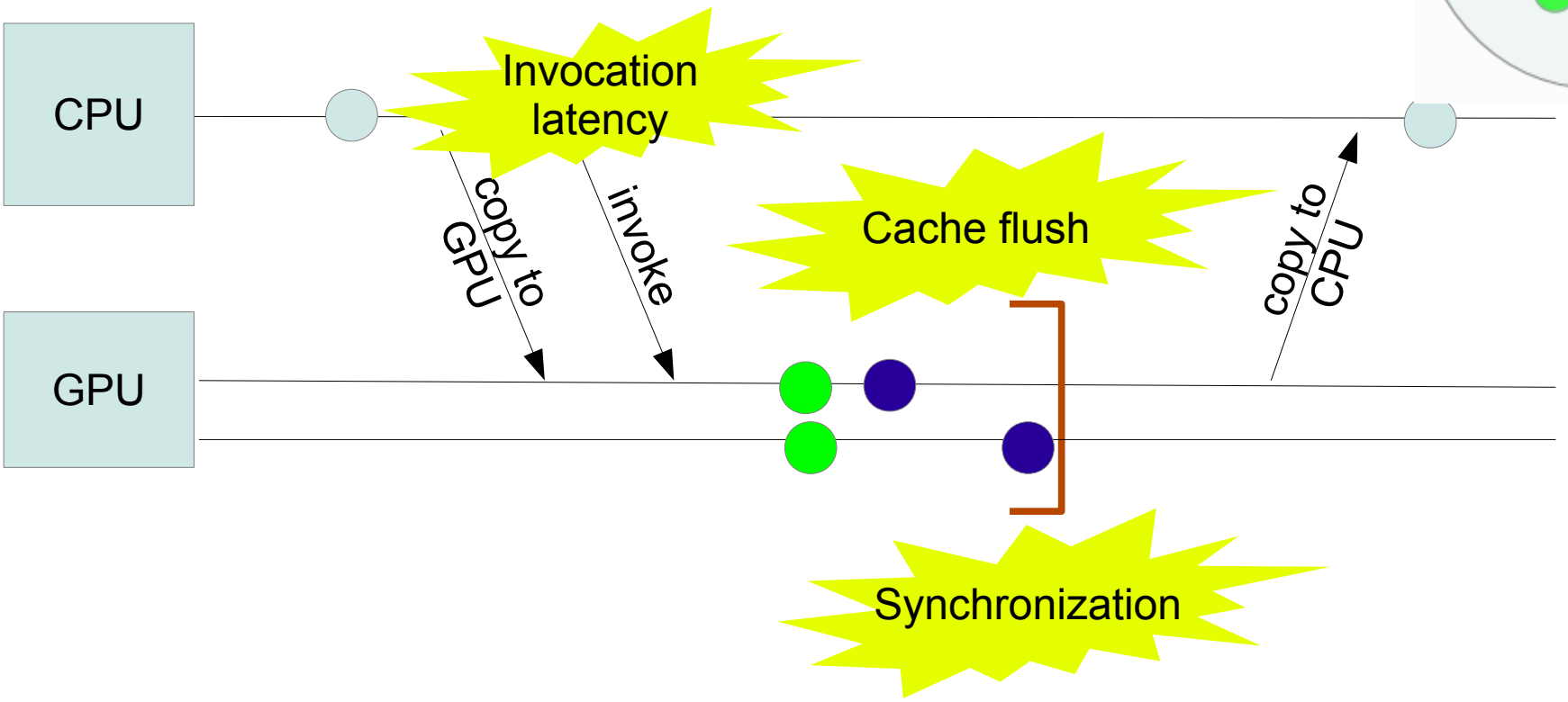
Offloading computations to GPU

Co-processor programming model

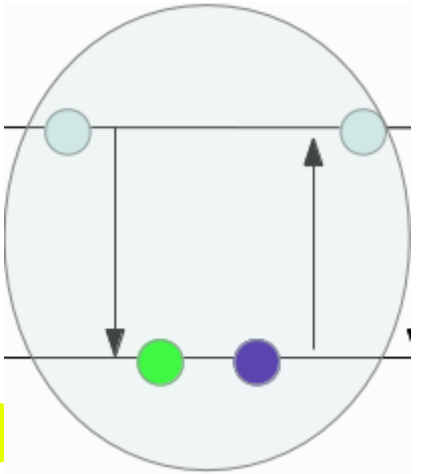


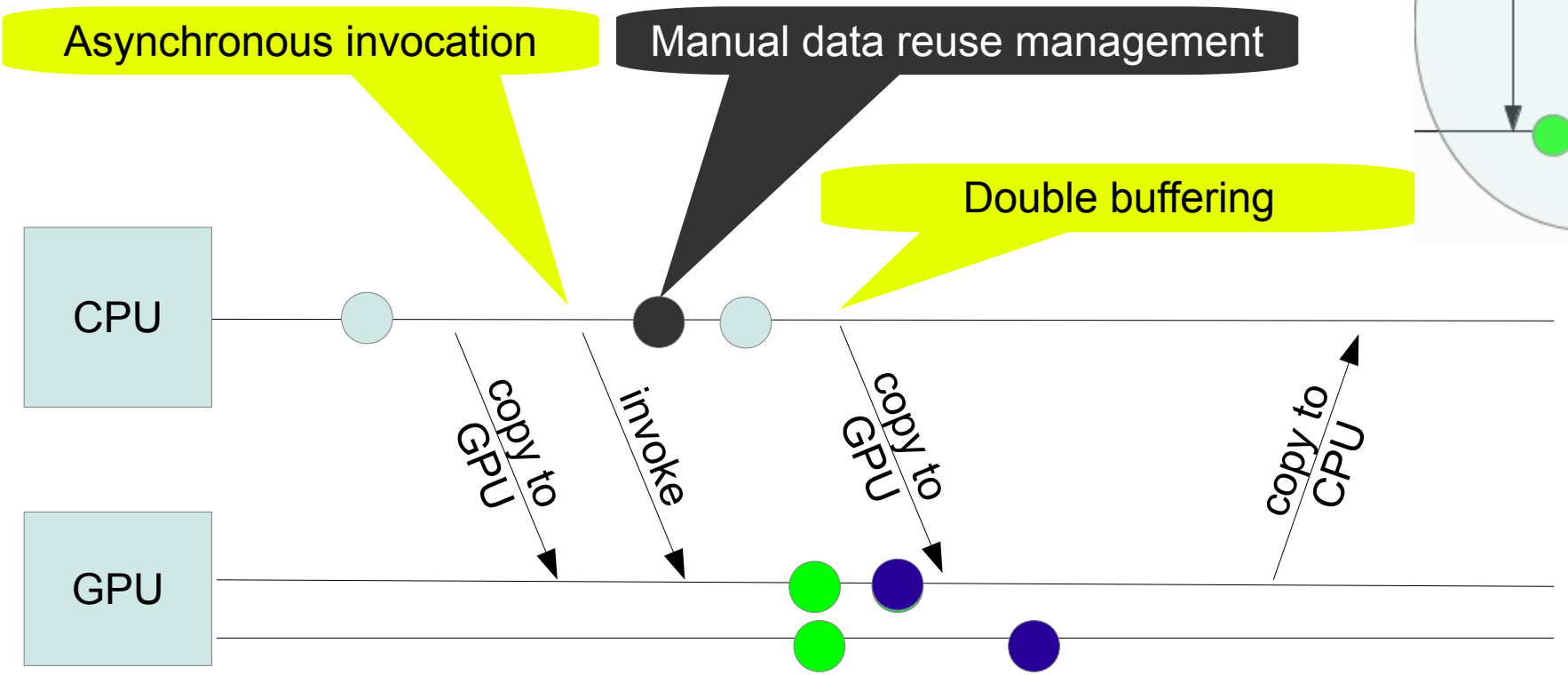
Kernel start/stop overheads



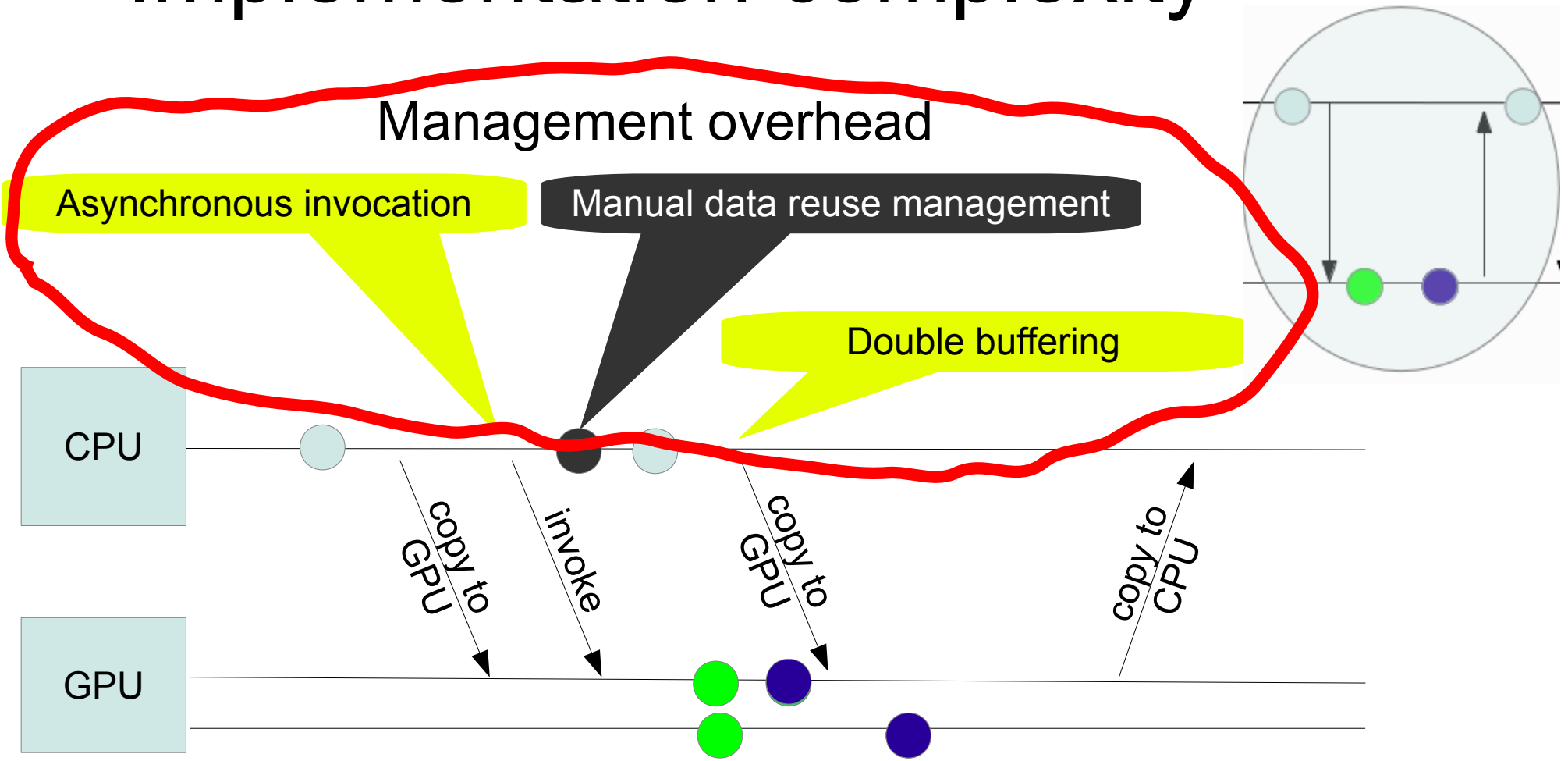


Hiding the overheads

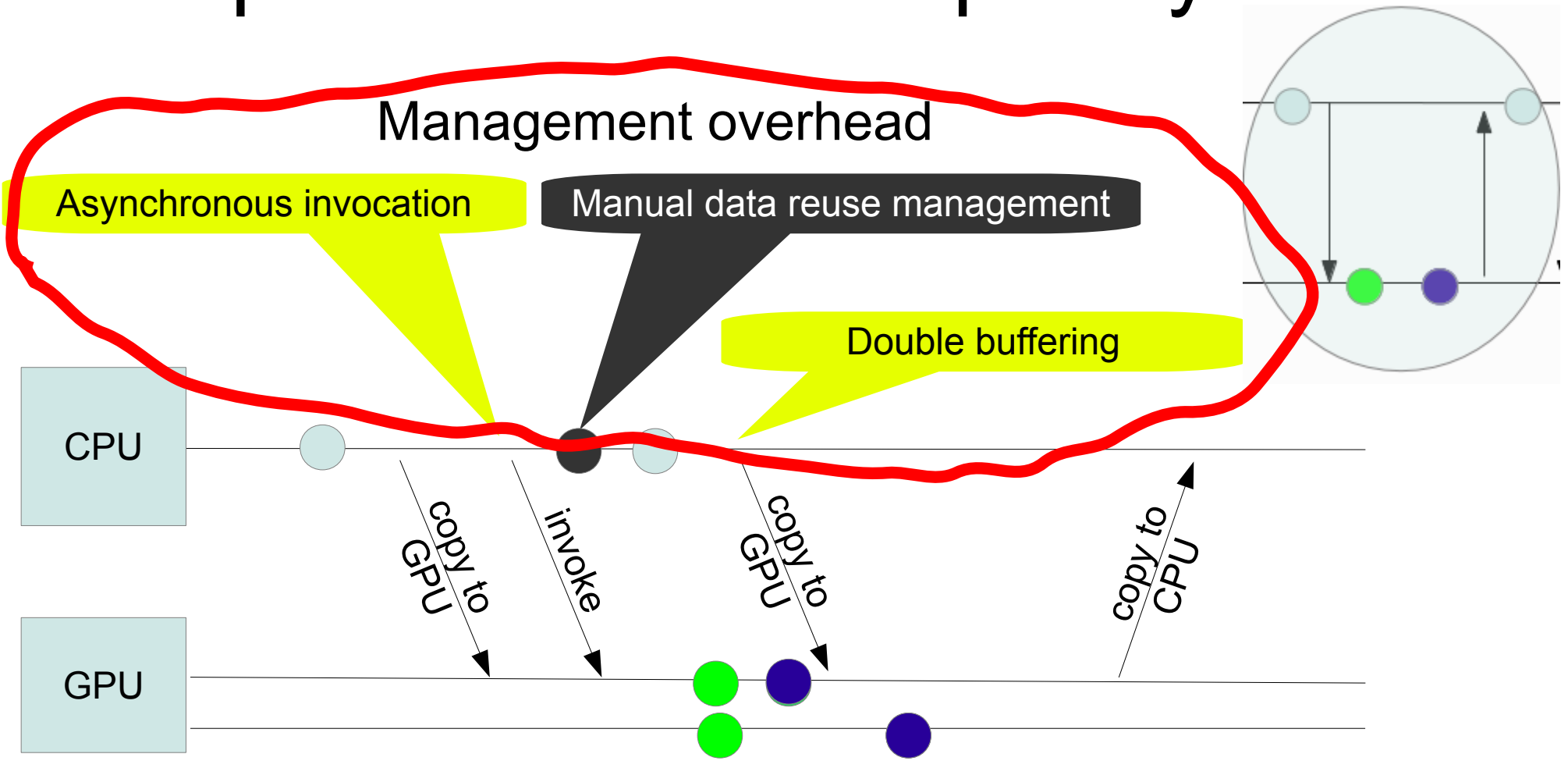




Implementation complexity



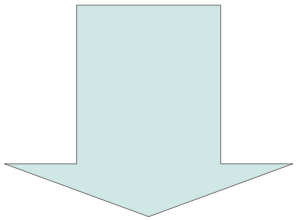
Implementation complexity



Why do we need to deal with low-level system details?

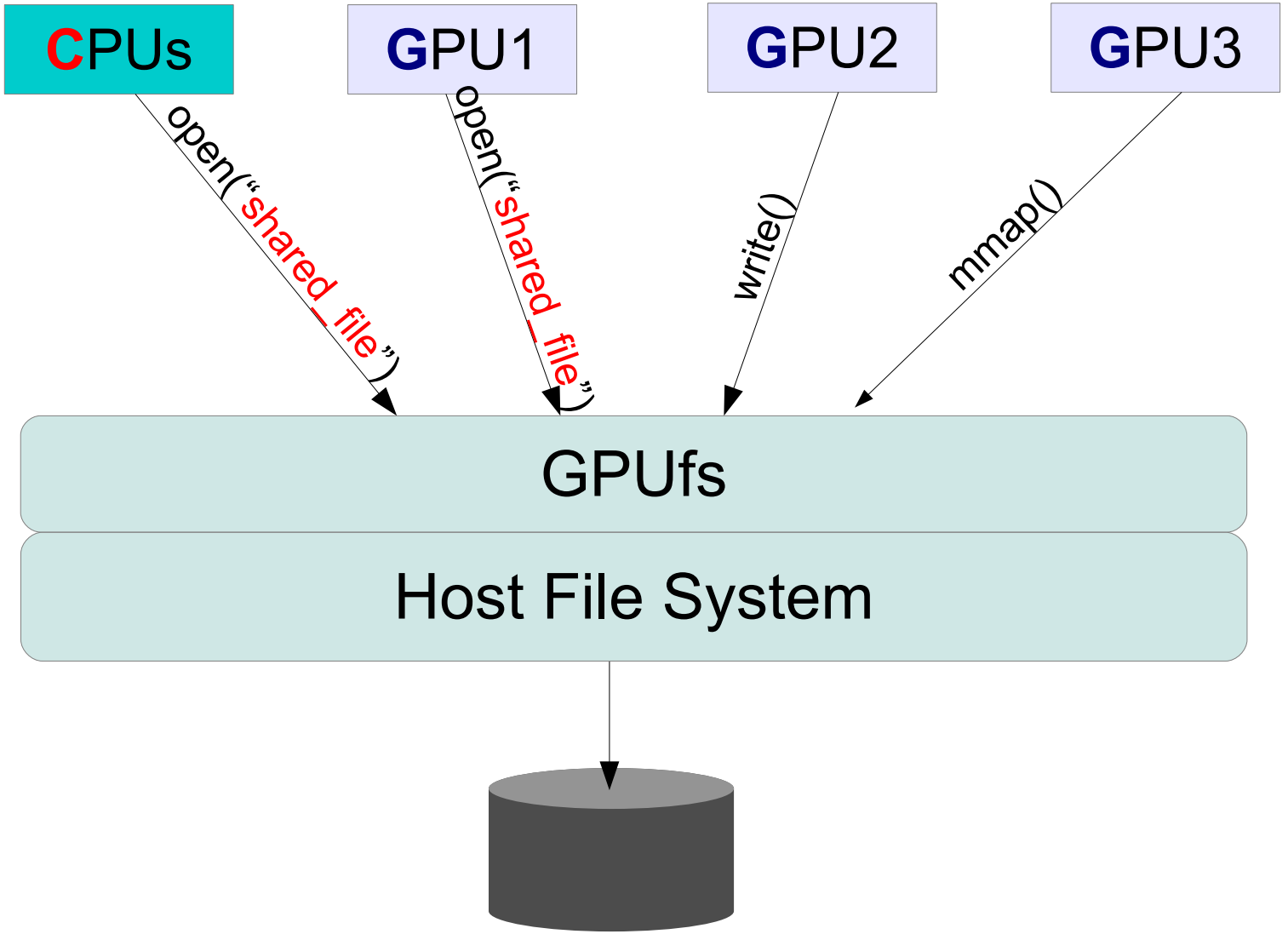
The reason is....

GPUs are peer-processors

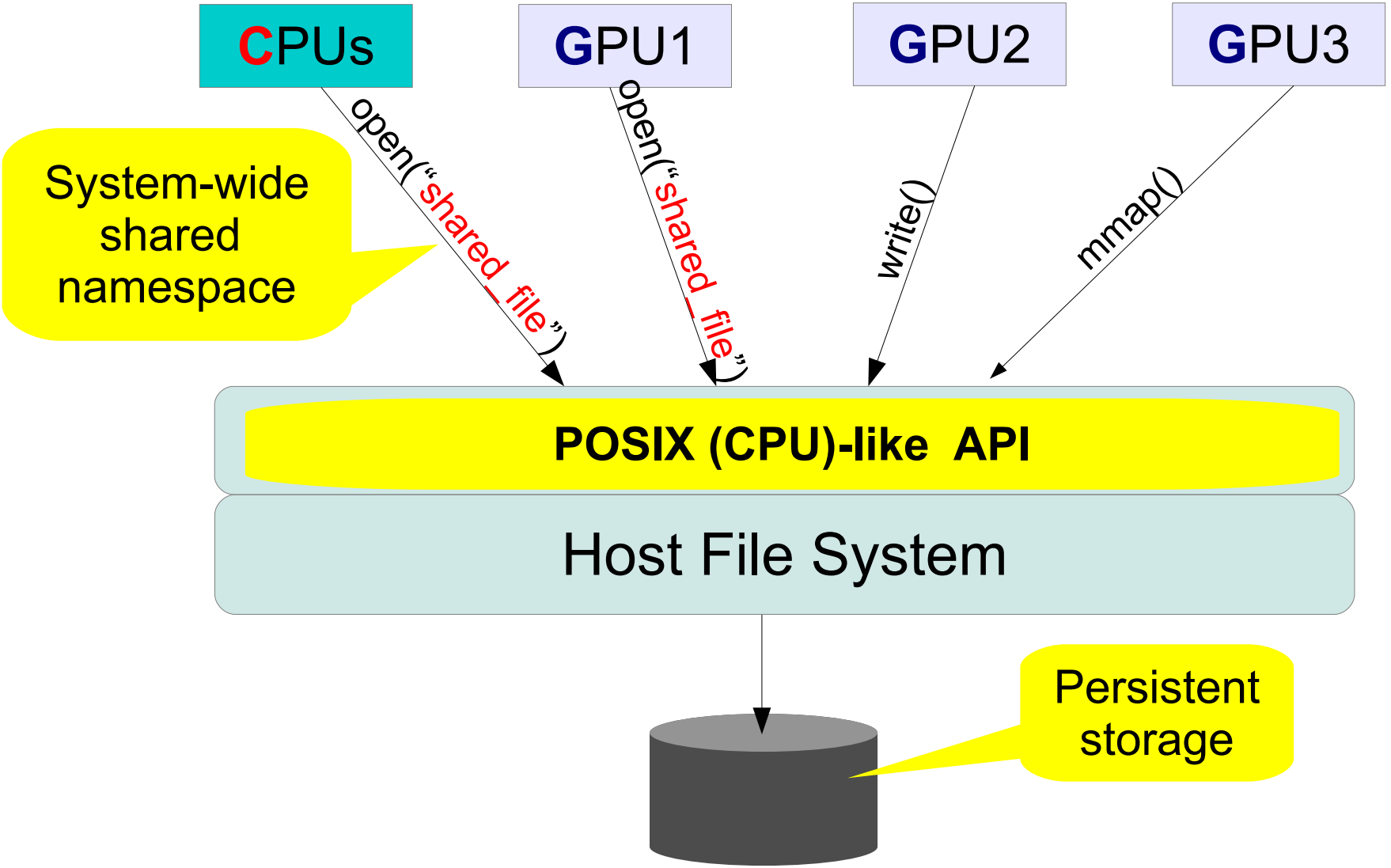


They need I/O OS services

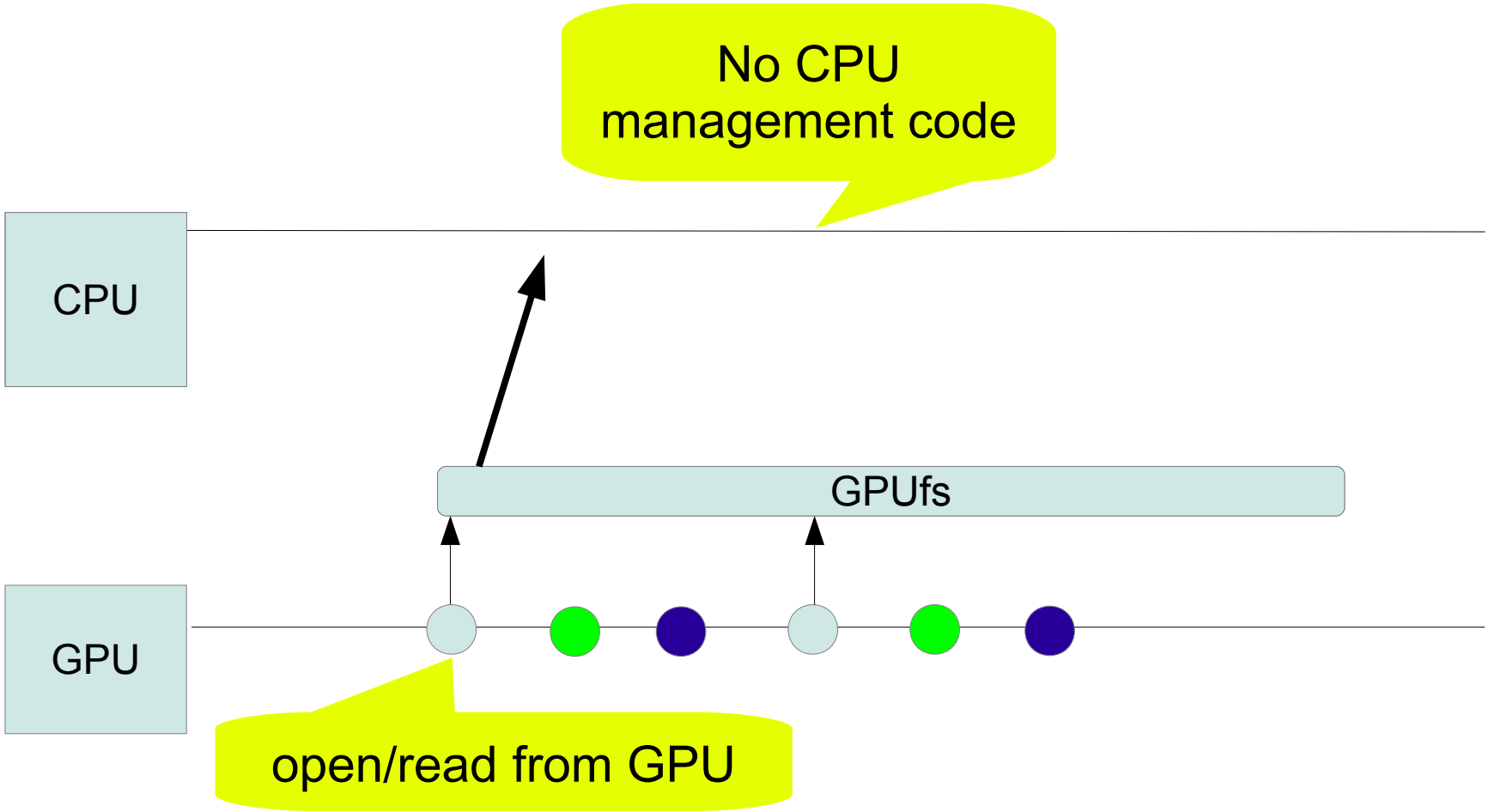
GPUfs: application view



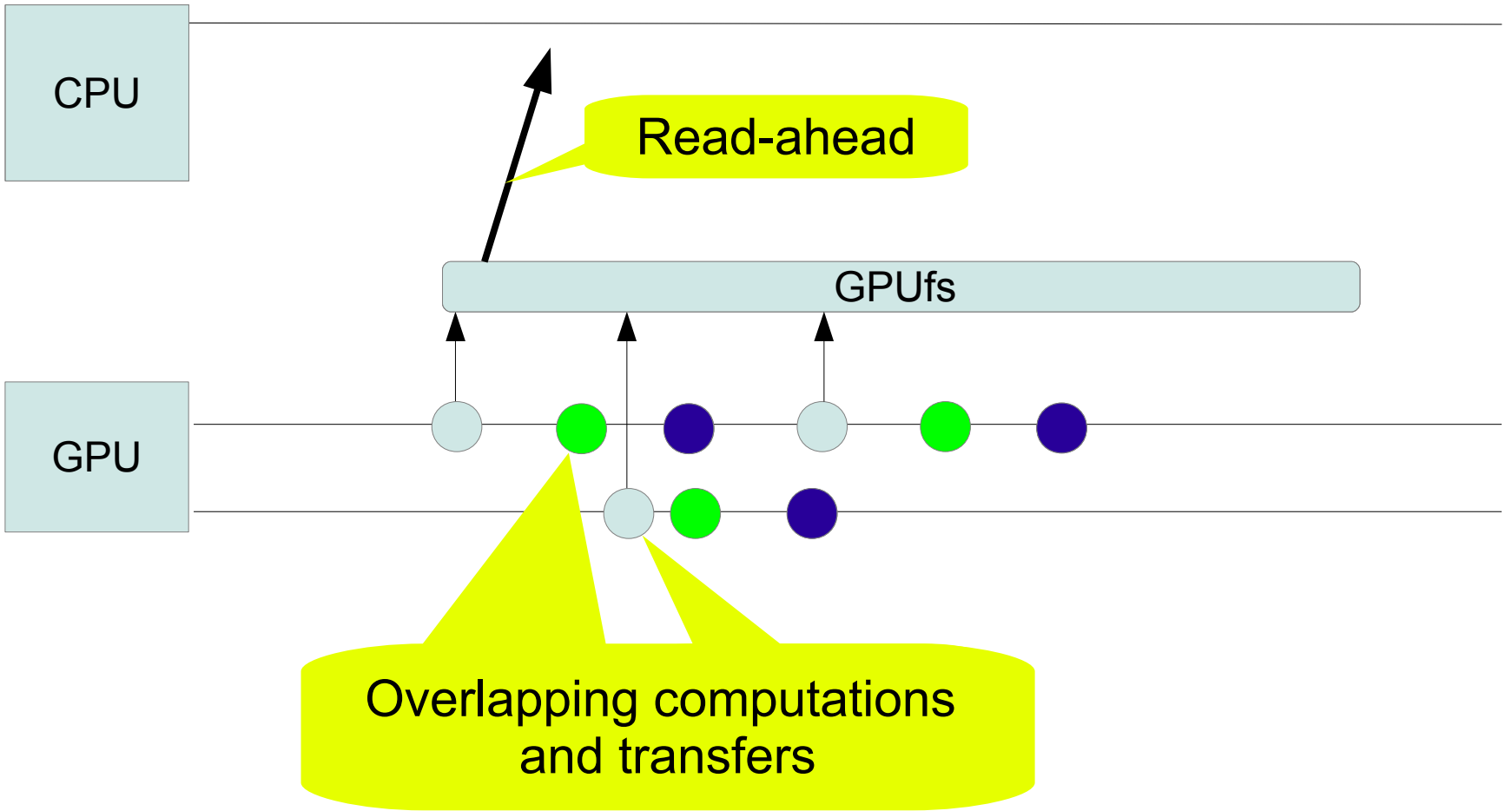
GPUfs: application view



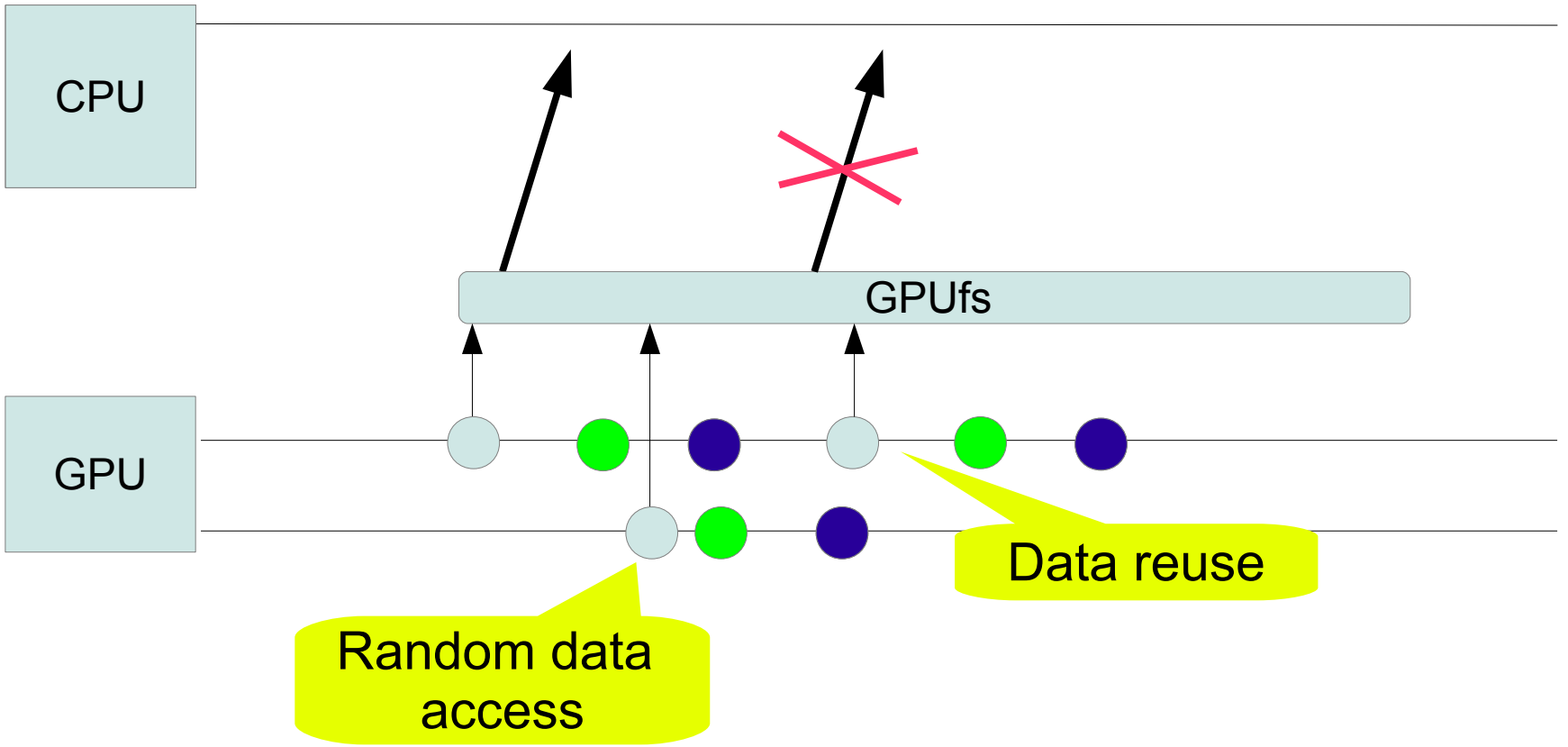
Accelerating collage app with GPUfs



Accelerating collage app with GPUfs



Accelerating collage app with GPUfs



Challenge

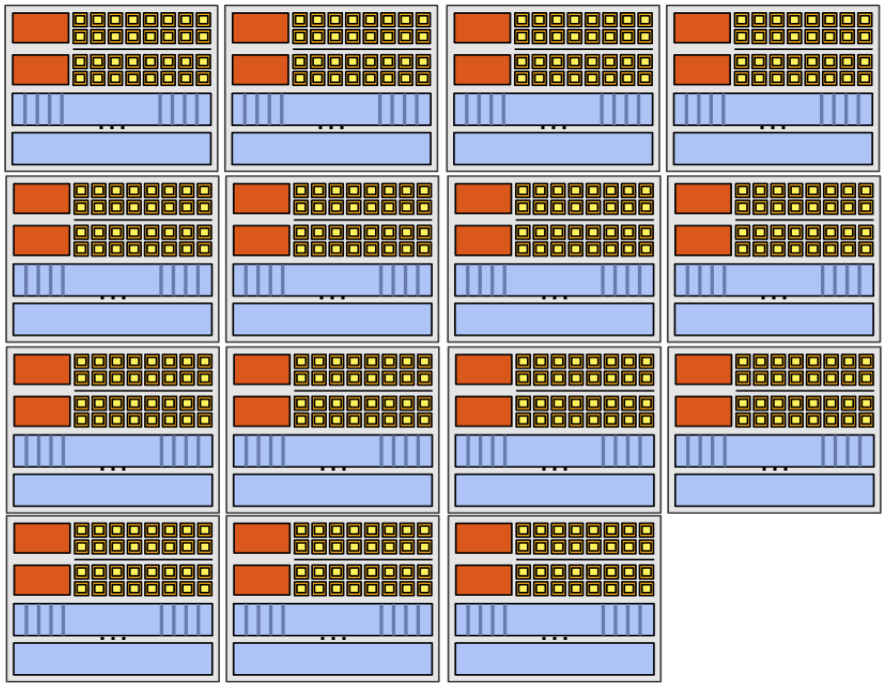
GPU ≠ CPU

Massive parallelism

Parallelism **is essential** for performance in deeply multi-threaded wide-vector hardware

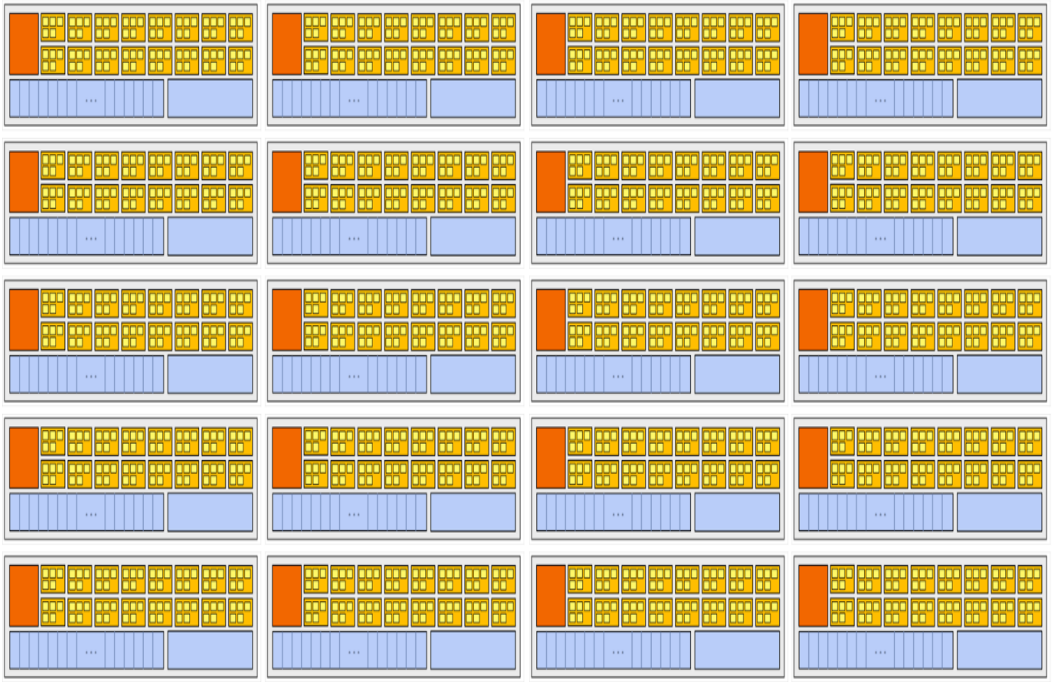
NVIDIA Fermi*

23,000
active threads



AMD HD5870*

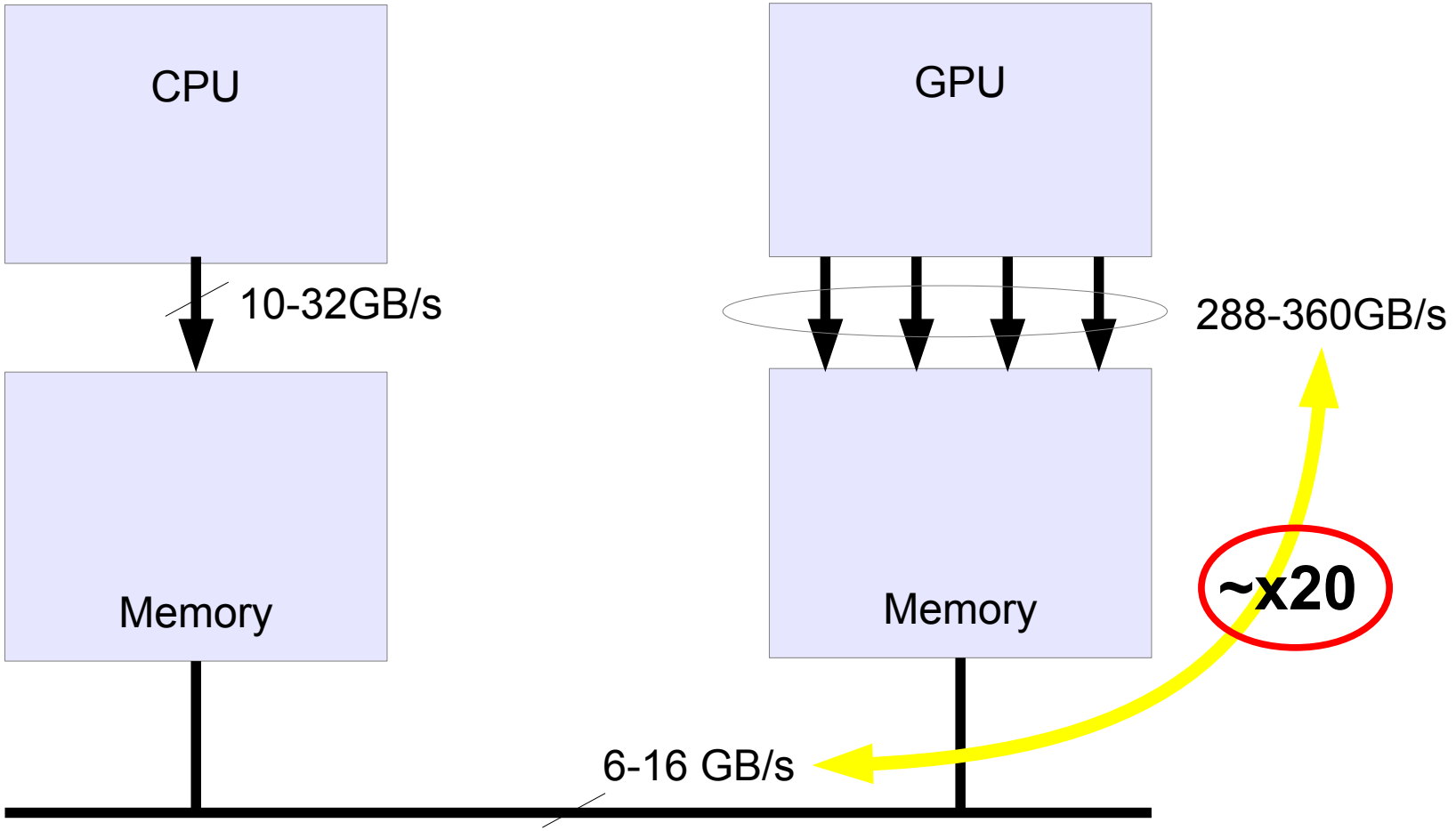
31,000
active threads



From M. Houston/A. Lefohn/K. Fatahalian – A trip through the architecture of modern GPUs*

Heterogeneous memory

GPUs **inherently impose** high bandwidth demands on memory

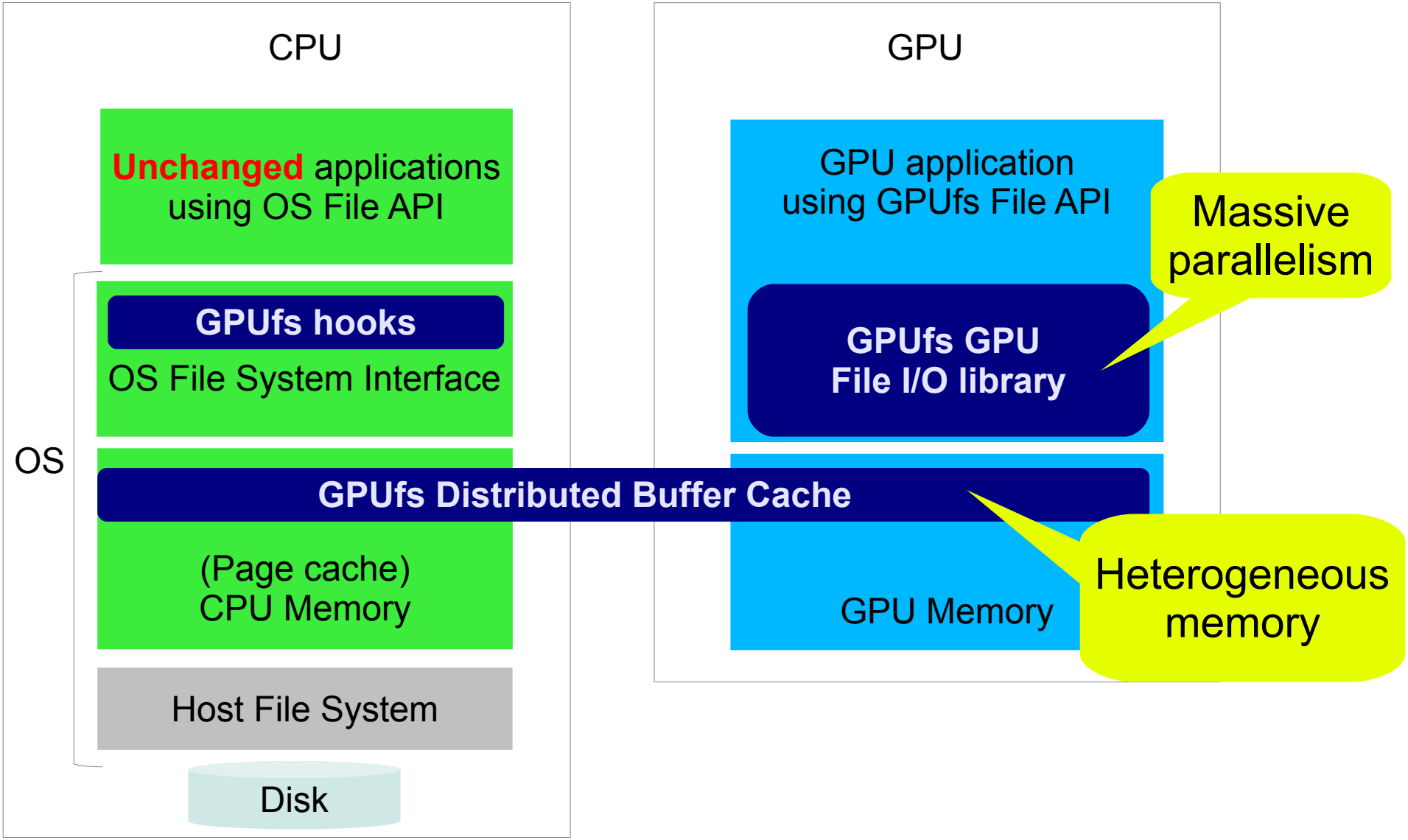


How to build an FS layer on this hardware?

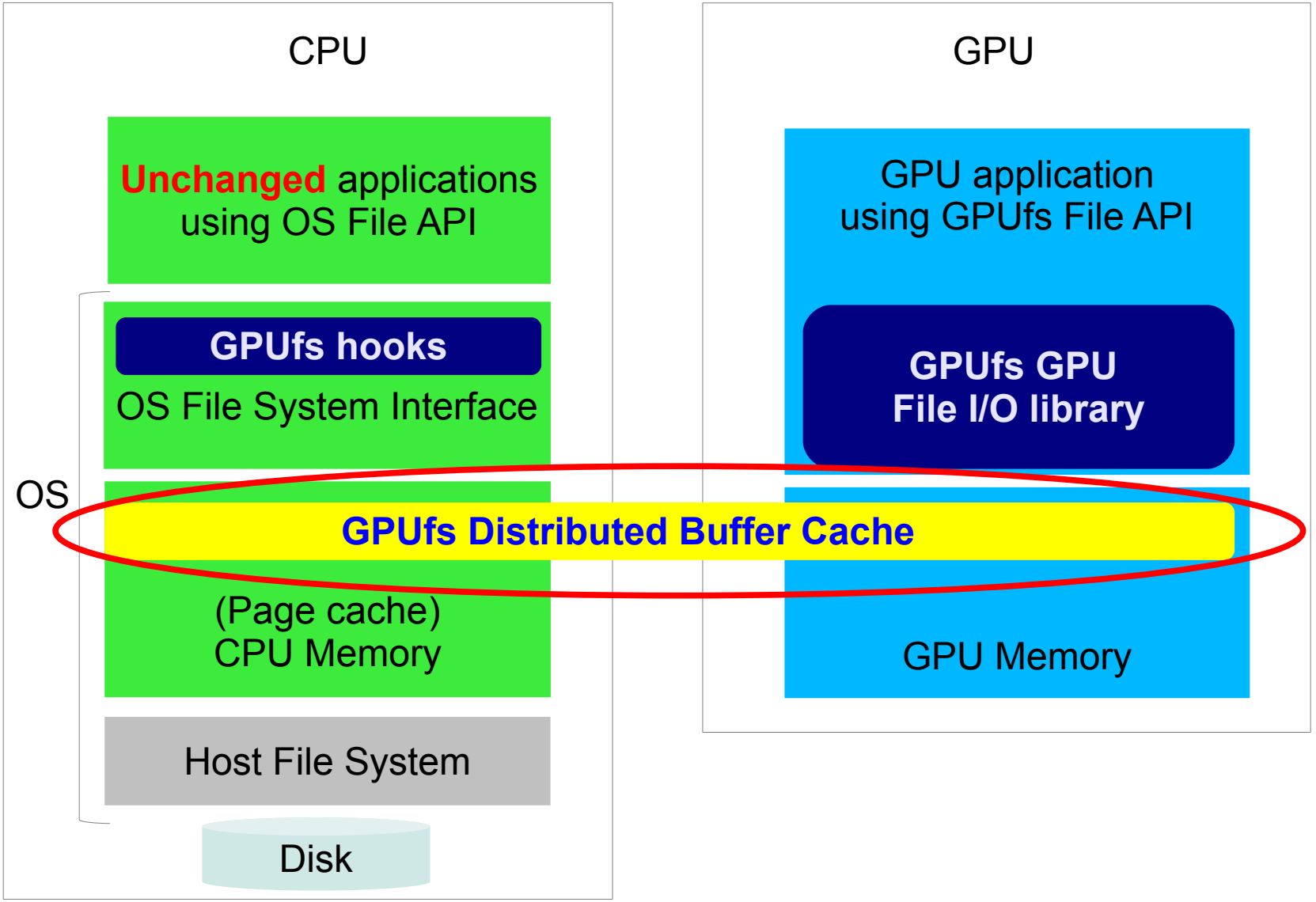
GPUfs: principled **redesign** of the **whole** file system stack

- **Relaxed FS API semantics** for parallelism
- **Relaxed FS consistency** for heterogeneous memory
- **GPU-specific implementation** of synchronization primitives, lock-free data structures, memory allocation, ….

GPUfs high-level design



GPUfs high-level design

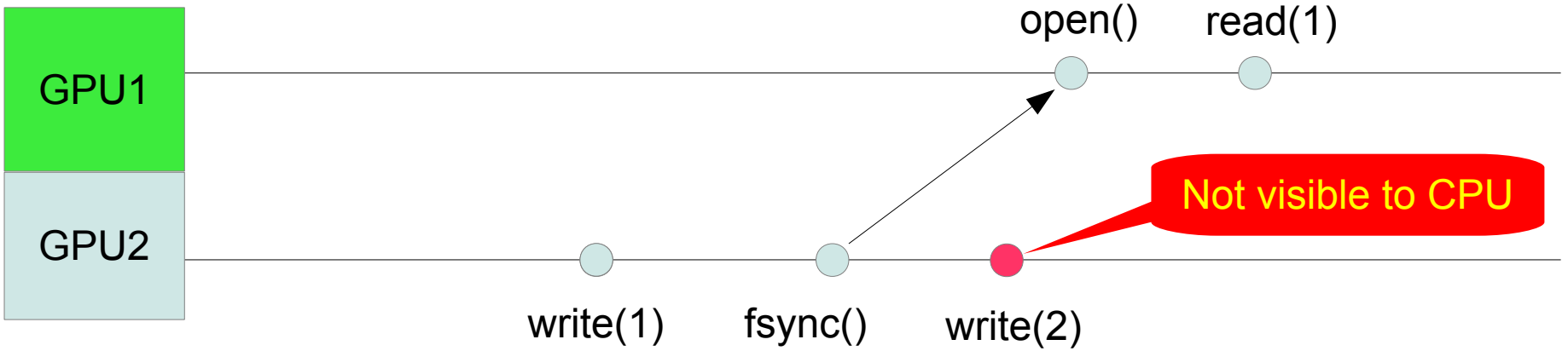


Buffer cache semantics

Local or Distributed file system data consistency?

GPUfs buffer cache
Weak data consistency model

- close(sync)-to-open semantics (AFS)



Remote-to-Local memory performance ratio is similar to **a distributed system**

On-GPU File I/O API

In the paper

open/close	→	gopen/gclose
read/write	→	gread/gwrite
mmap/munmap	→	gmmap/gmunmap
fsync/msync	→	gfsync/gmsync
ftrunc	→	gftrunc

Changes in the semantics are crucial

Implementation bits

In the paper

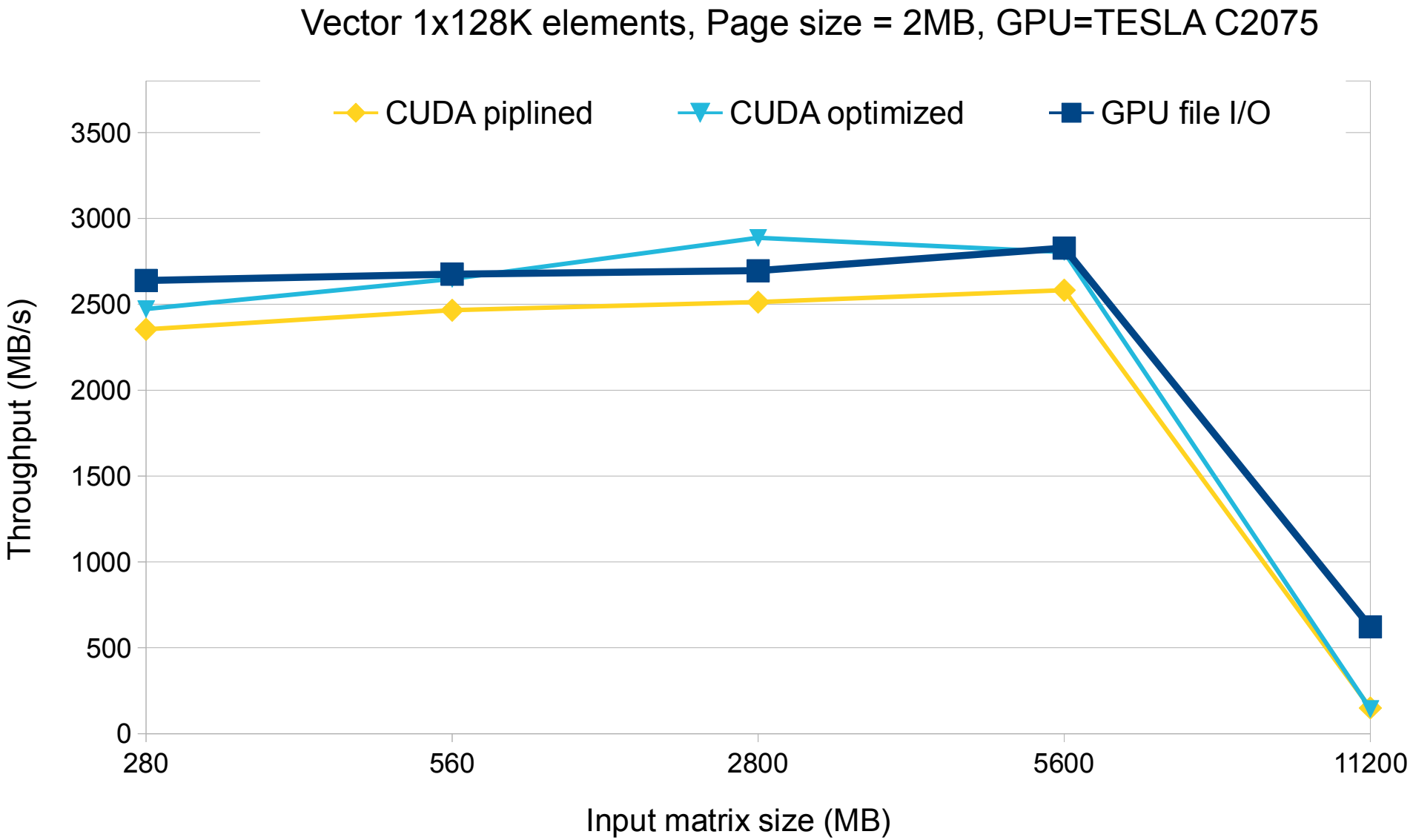
- Paging support
- Dynamic data structures and memory allocators
- Lock-free radix tree
- Inter-processor communications (IPC)
- Hybrid H/W-S/W barriers
- Consistency module in the OS kernel

~1,5K GPU LOC, ~600 CPU LOC

Evaluation

All benchmarks are written as a GPU kernel:
no CPU-side development

Matrix-vector product (Inputs/Outputs in files)



Word frequency count in text

- Count frequency of modern English words in the works of Shakespeare, and in the Linux kernel source tree

English dictionary: 58,000 words

Challenges

Dynamic working set
Small files
Lots of file I/O (33,000 files,1-5KB each)
Unpredictable output size

Results

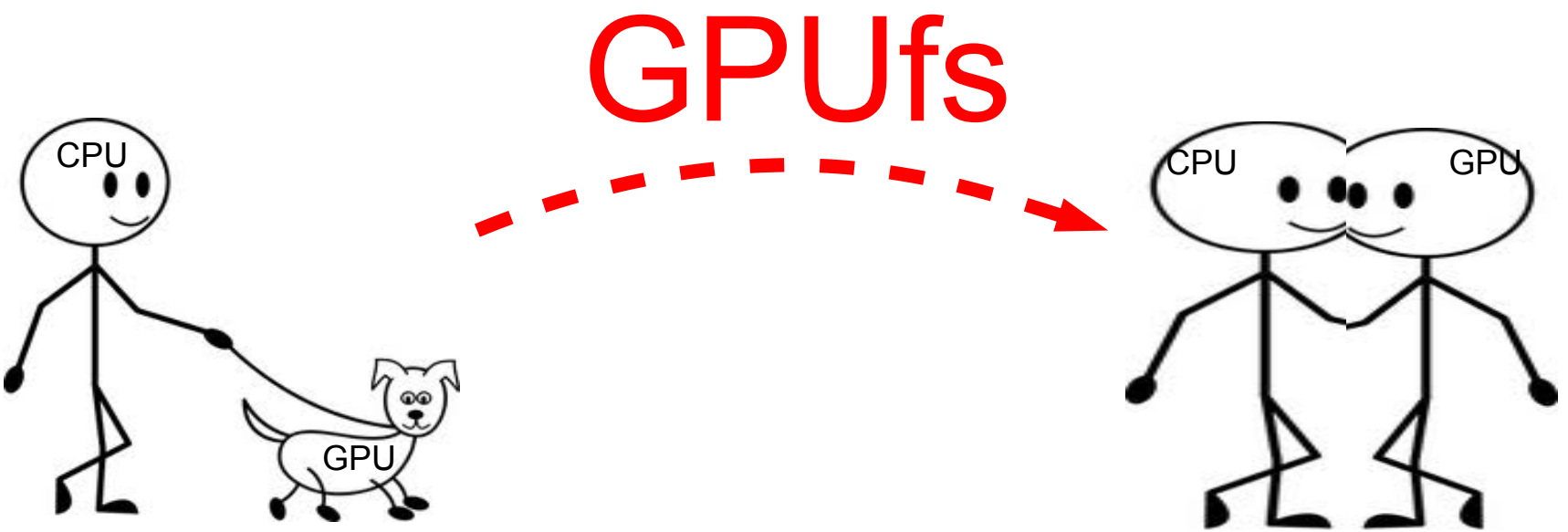
	8CPUs	GPU-vanilla	GPU-GPUfs
Linux source 33,000 files, 524MB	6h	50m (7.2X)	53m (6.8X)
Shakespeare 1 file, 6MB	292s	40s (7.3X)	40s (7.3X)

Results

	8CPUs	GPU-vanilla	GPU-GPUfs
Linux source 33,000 files, 524MB	6h	50m (7.2X)	53m (6.8X)
Shakespeare 1 file, 6MB	292s	40s (7.3X)	40s (7.3X)
Unbounded input/output size support	✓	✗	✓

8% overhead

GPUfs is the **first system** to provide native access to **host OS services** from **GPU programs**



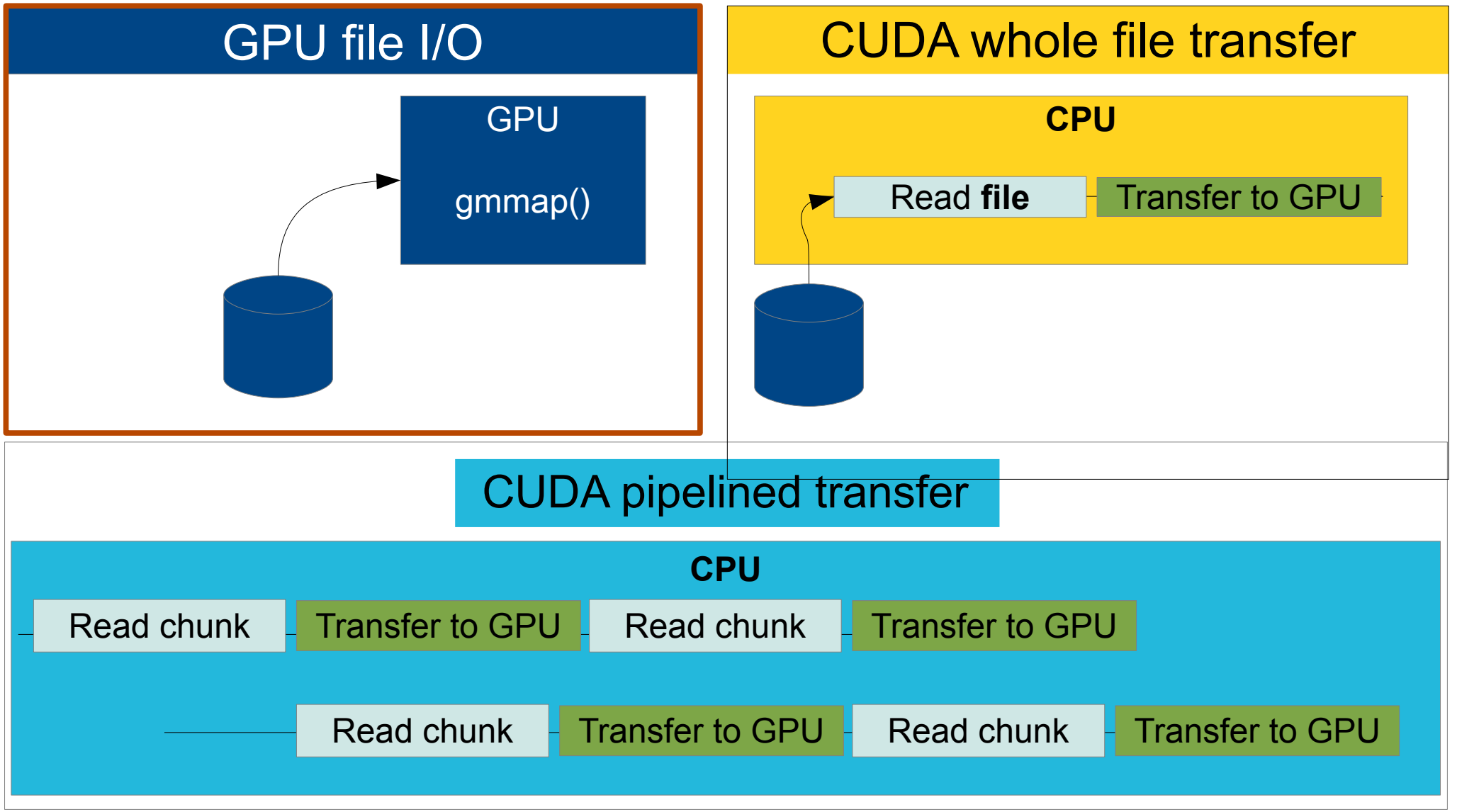
Code is available for download at:
https://sites.google.com/site/silbersteinmark/Home/gpufs

http://goo.gl/ofJ6J

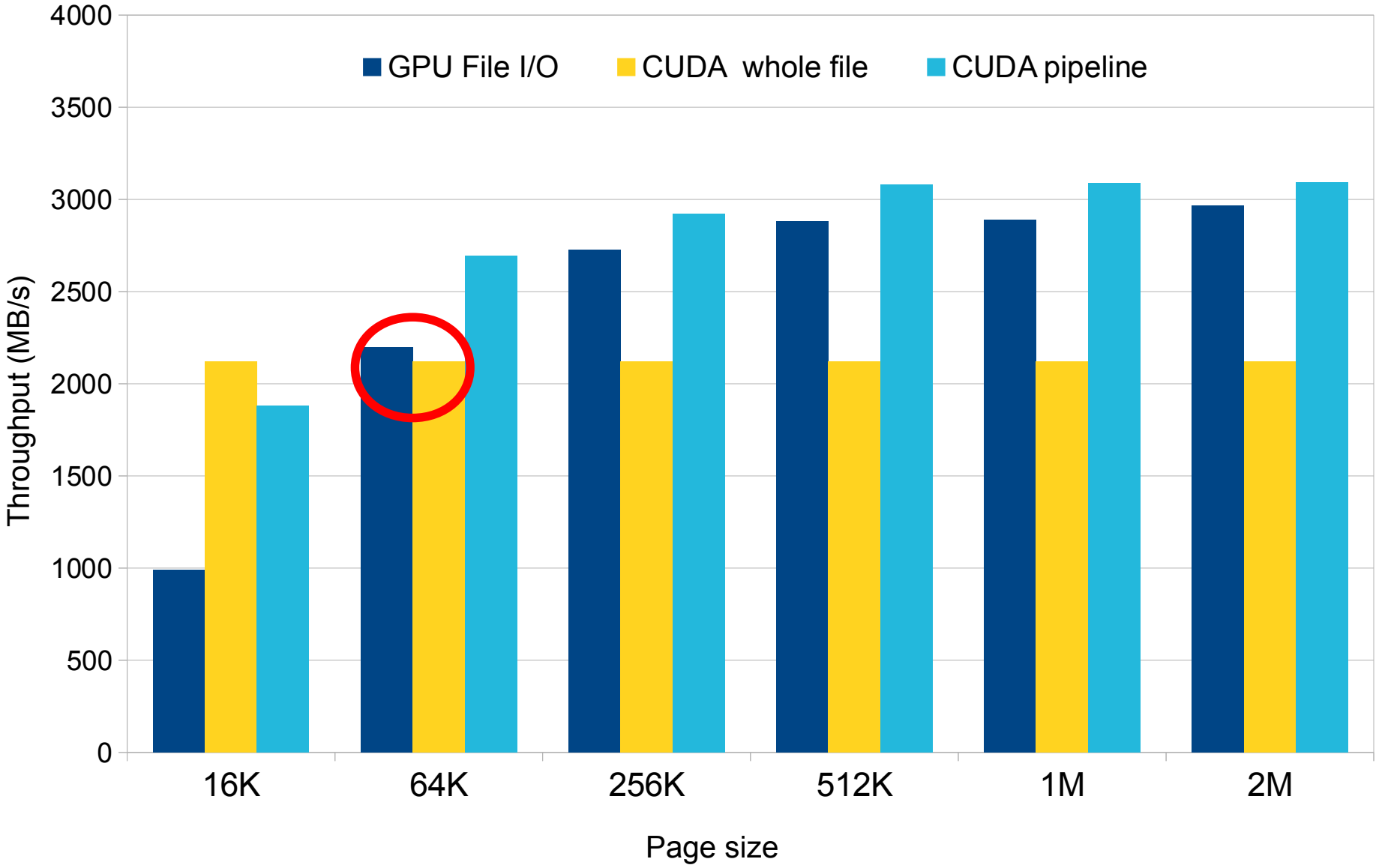
Our life would have been easier with

- PCI atomics
- Preemptive background daemons
- GPU-CPU signaling support
- In-GPU exceptions
- GPU virtual memory API (host-based or device)
- Compiler optimizations for register-heavy libraries
  - Seems like accomplished in 5.0

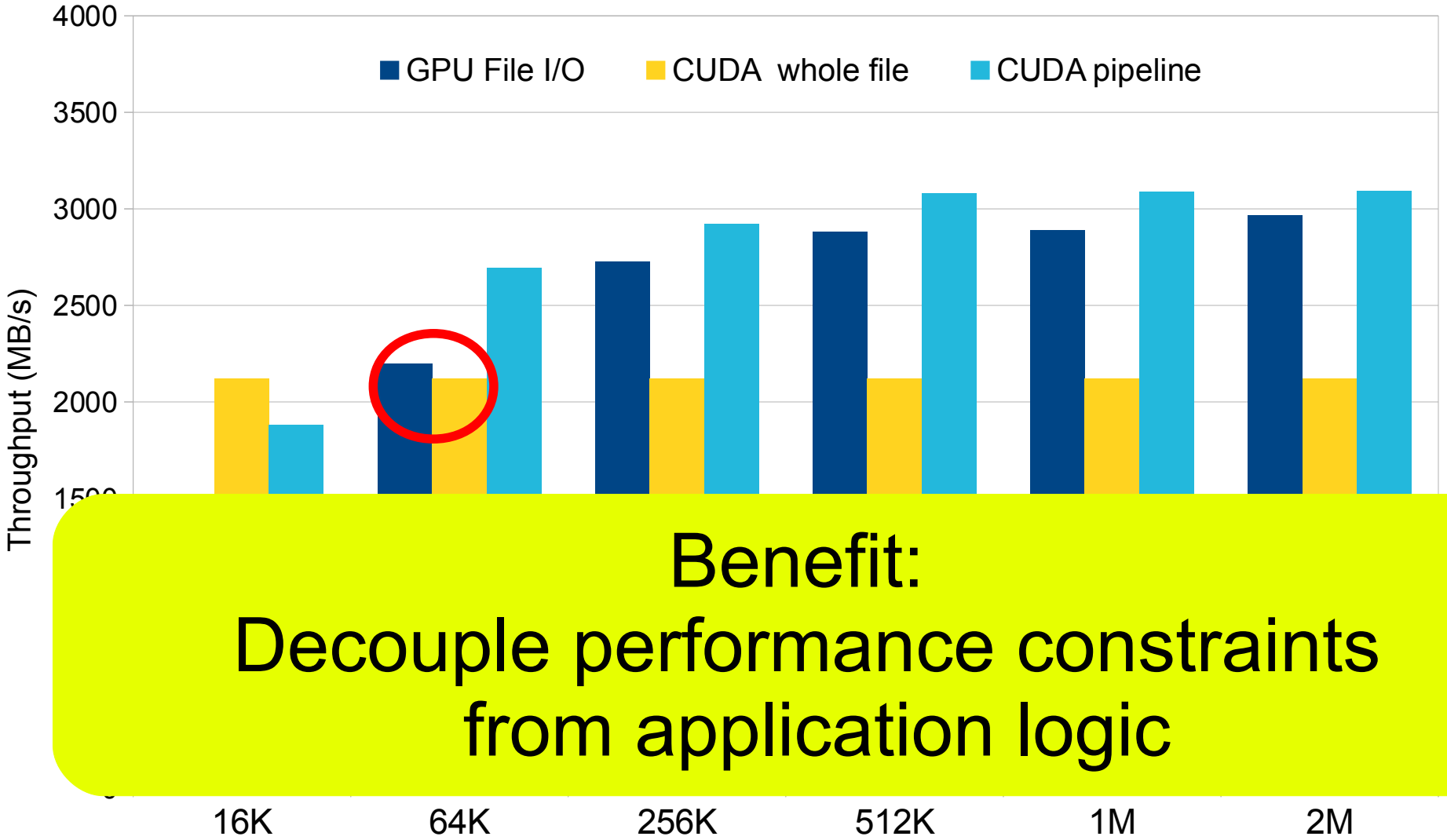
Sequential access to file: 3 versions



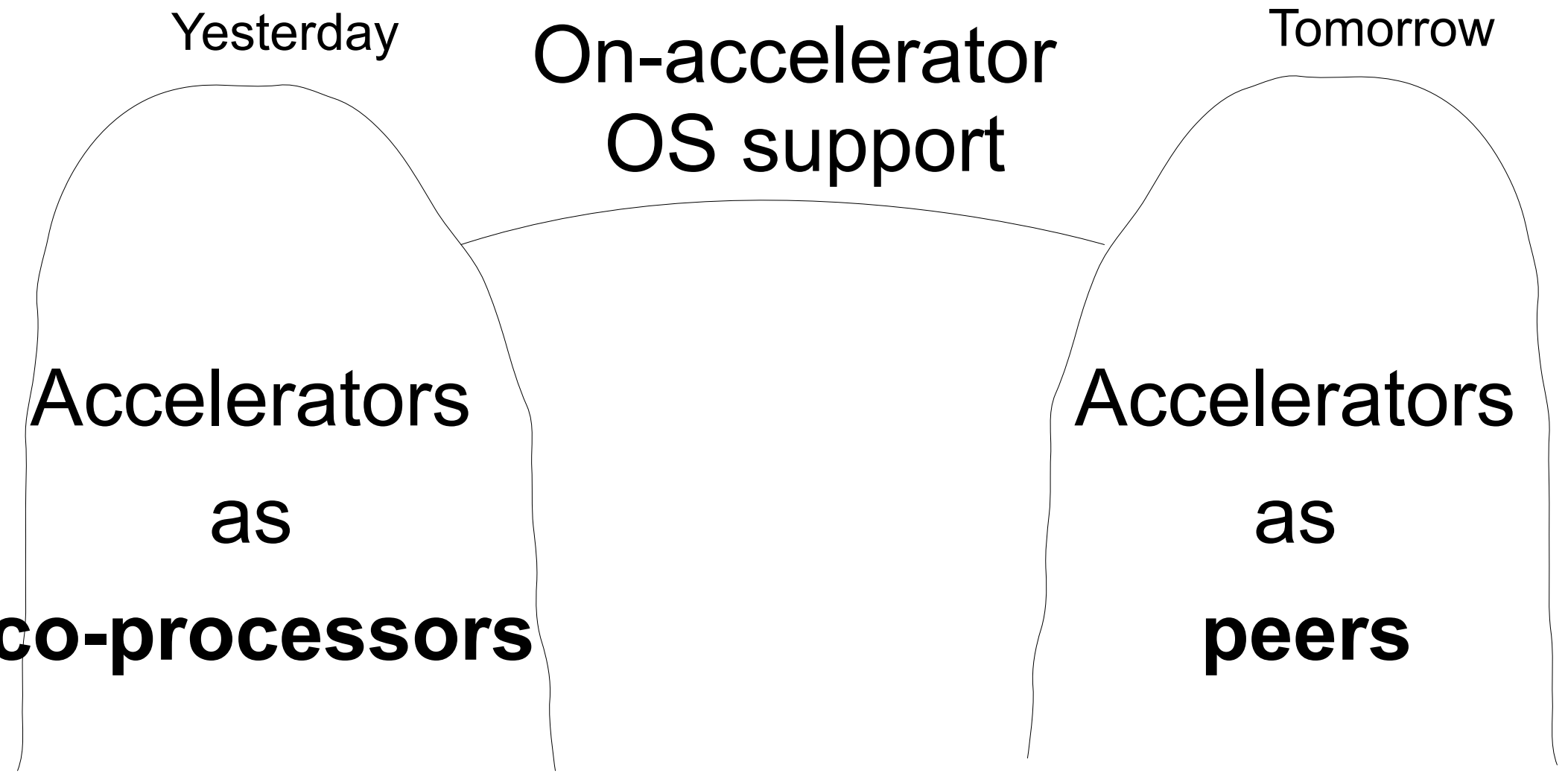
Sequential read
Throughput vs. Page size



Sequential read
Throughput vs. Page size

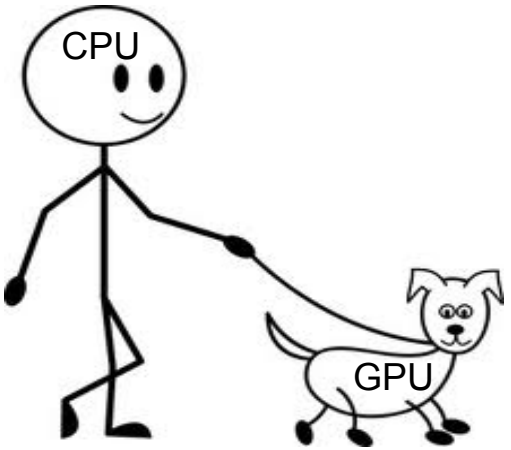


Benefit:
Decouple performance constraints from application logic


Yesterday
On-accelerator
OS support
Tomorrow
Accelerators
as
co-processors
Accelerators
as
peers

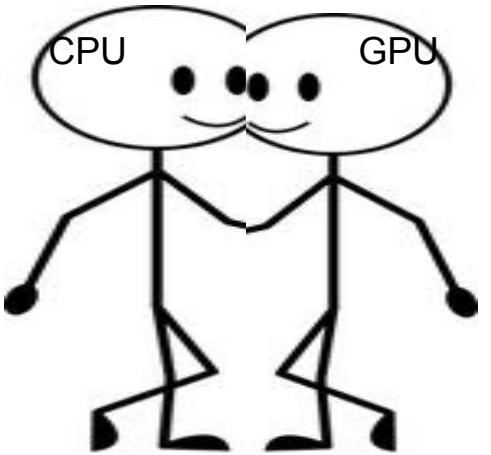
What about software?

Yesterday



?

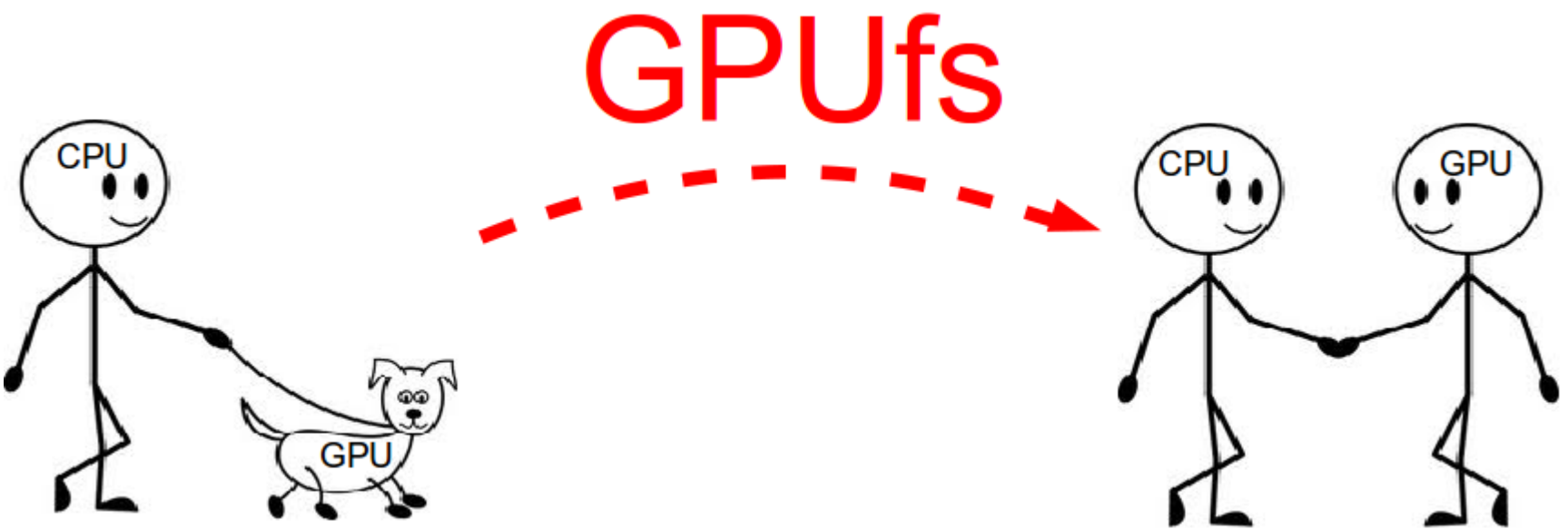
Tomorrow



Accelerators as **co-processors**

Accelerators as **peers**

Set GPUs free!



Parallel square root on GPU

```
gpu_thread(thread_id i){
 float buffer;
 int fd=gopen(filename,O_GRDWR);

 offset=sizeof(float)*i;
 gread(fd,sizeof(float),&buffer,offse
 buffer=sqrt(buffer);
 gwrite(fd,sizeof(float),&buffer,offs

 gclose(fd);
}
```

Same code will run in all thousands of the GPU threads

GPUfs impact on GPU programs


Memory overhead


Register pressure

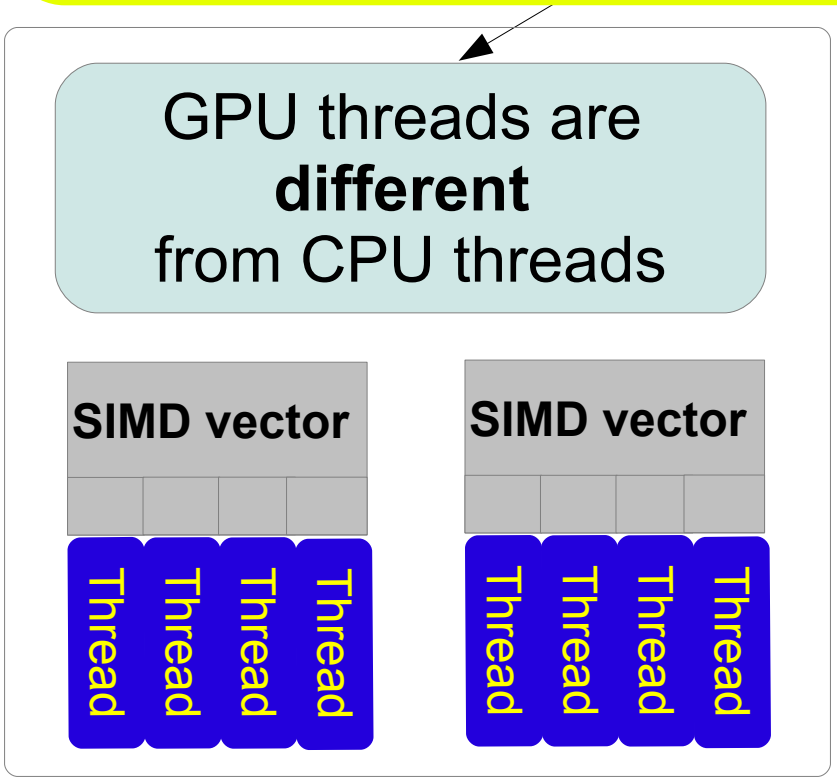

Very little CPU coding


Makes exitless GPU kernels possible

Pay-as-you-go design

Preserve CPU semantics?

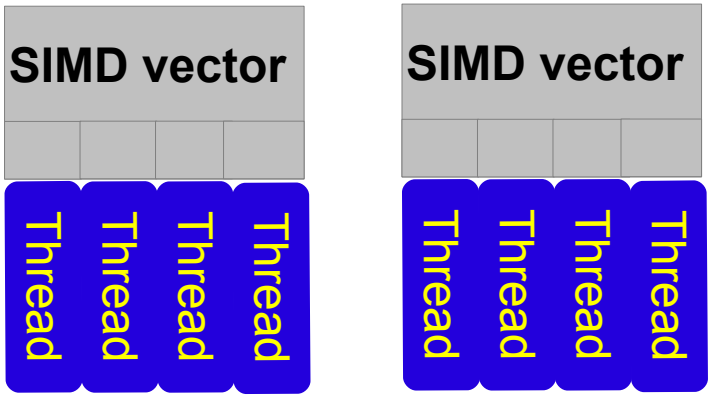
What does it mean to
open/read/write/close/mmap a file
in thousands of threads?



Preserve CPU semantics?

What does it mean to
open/read/write/close/mmap a file
in thousands of threads?

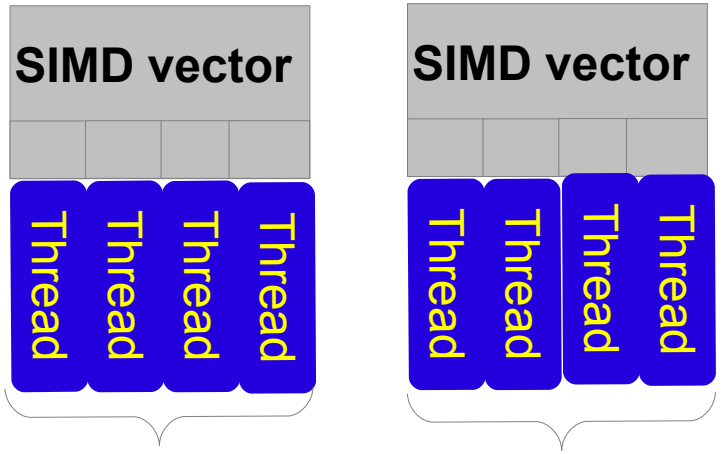
GPU threads are
different
from CPU threads



GPU kernel is a
single data-parallel
application

GPUfs semantics (see more discussion in the paper)

```
int fd=gopen("filename",O_GRDWR);
```



100,000
file descriptors
to the same file?

One call
per **SIMD vector:**
bulk-synchronous
cooperative execution

One file
descriptor per **file**:
open()/close()
cached on a GPU

GPU hardware characteristics

Parallelism

Heterogeneous memory

API semantics

```
int fd=gopen("filename",O_GRDWR);
```

This code runs in 100,000 **GPU** threads

API semantics

```
int fd=gopen("filename",O_GRDWR);
```

100,000
different calls

C
P
U

100,000
file descriptors

This code runs in 100,000 **GPU** threads

```
int fd=gopen("filename",O_GRDWR);
```

100,000 different calls

100,000 file descriptors

C P U ≠ G P U

1 call per set of threads

1 file descriptor per file