PTask: Dataflow programming for Heterogeneous Platforms

Emmett Witchel

CS380L

Motivation

- There are lots of GPUs
  - 3 of top 5 supercomputers use GPUs
  - In all new PCs, smart phones, tablets
  - Great for gaming and HPC/batch
  - Unusable in other application domains
- GPU programming challenges
  - GPU+main memory disjoint
  - Treated as I/O device by OS
  - Imperative code
  - Data movement, algorithms coupled

Motivation

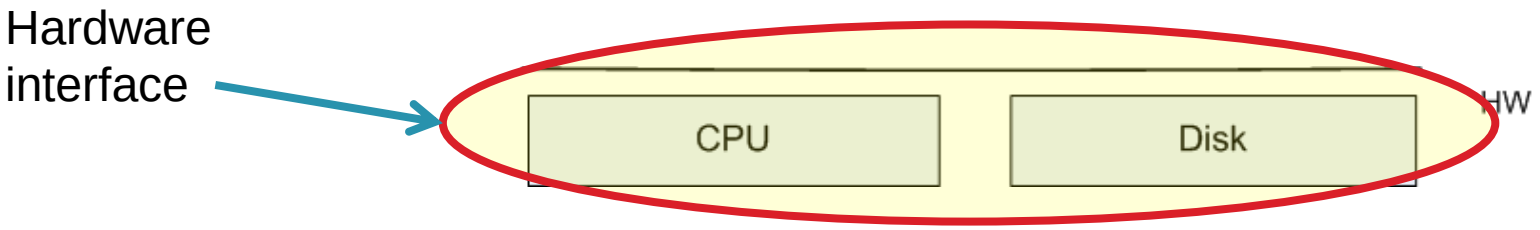
- There are lots of GPUs
  - 3 of top 5 supercomputers use GPUs
  - In all new PCs, smart
  - Great for gaming and
  - *Unusable in other ap*
- GPU programing challenges
  - GPU+main memory disjoint
  - *Treated as I/O device by OS*
  - *Imperative code*
  - *Data movement, algorithms coupled*

These things are related:
We need better abstractions, better programming models

Outline

- The case for OS support
- The case for dataflow abstractions
- PTask: Dataflow for GPUs
- Related and Future Work
- Conclusion

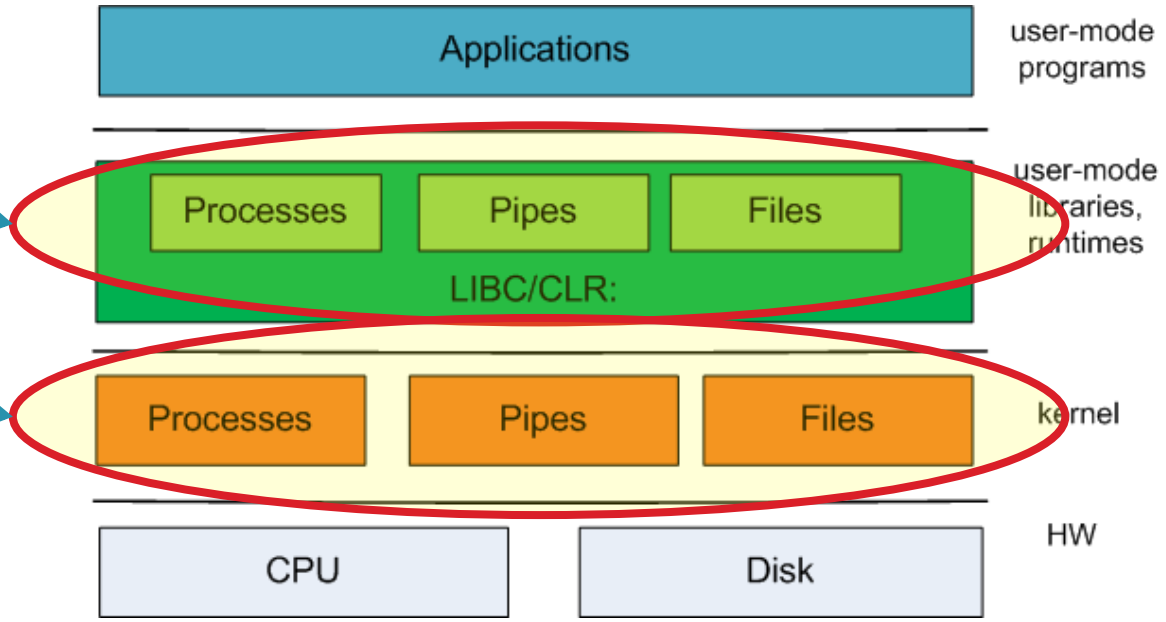
Déjà vu: OS-Level abstractions



Déjà vu: OS-Level abstractions

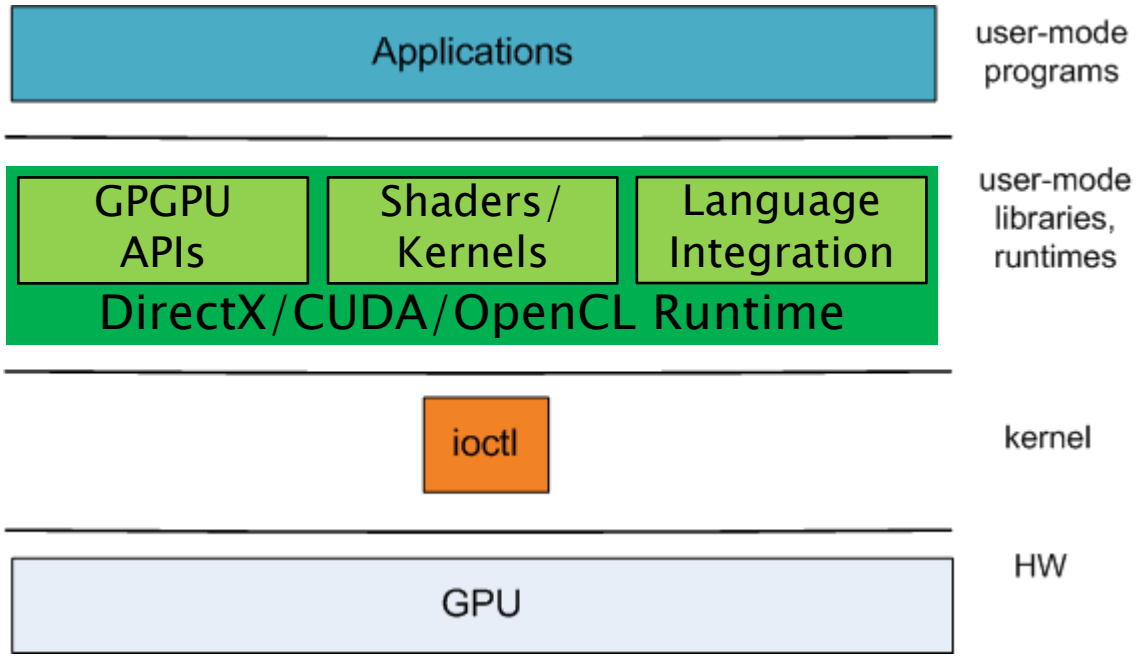
programmer-visible interface

OS-level abstractions



1:1 correspondence between OS-level and user-level abstractions

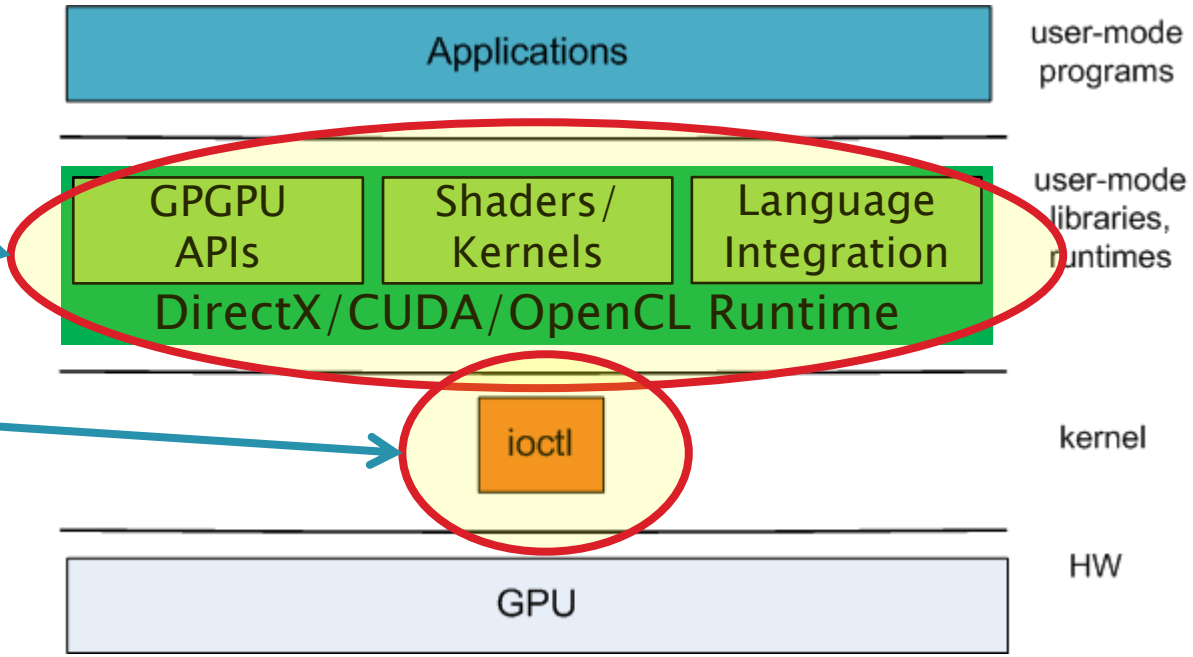
GPU Abstractions



GPU Abstractions

programmer-visible interface

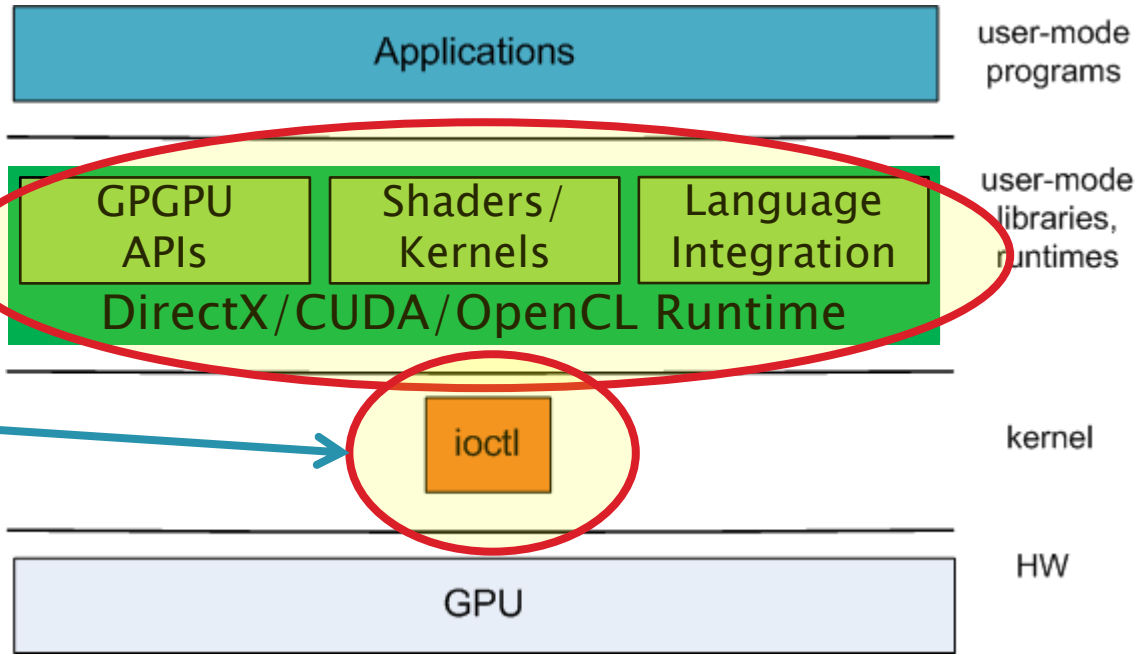
1 OS-level abstraction!



GPU Abstractions

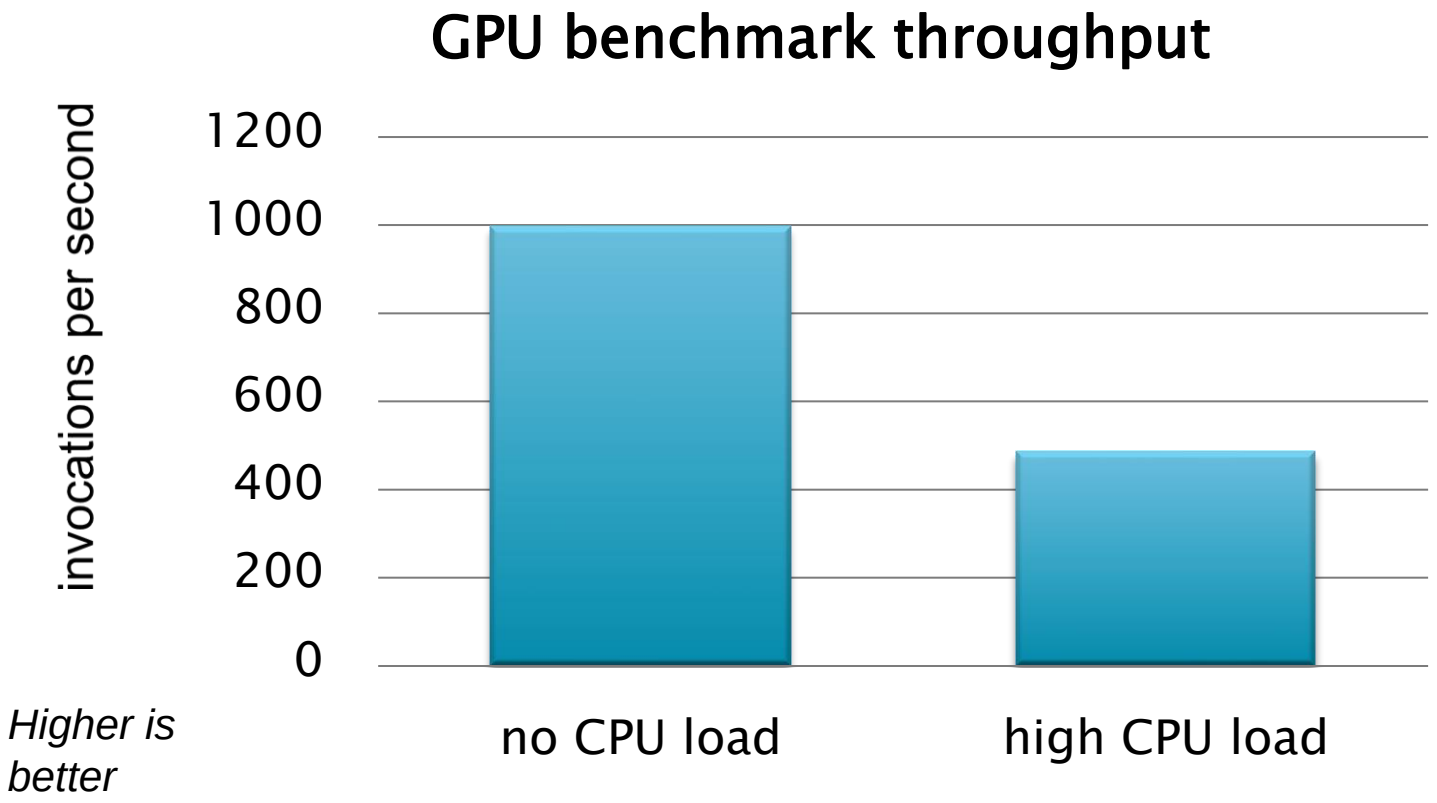
programmer-visible interface

1 OS-level abstraction!



1. No kernel-facing API
2. No OS resource-management
3. Poor composability

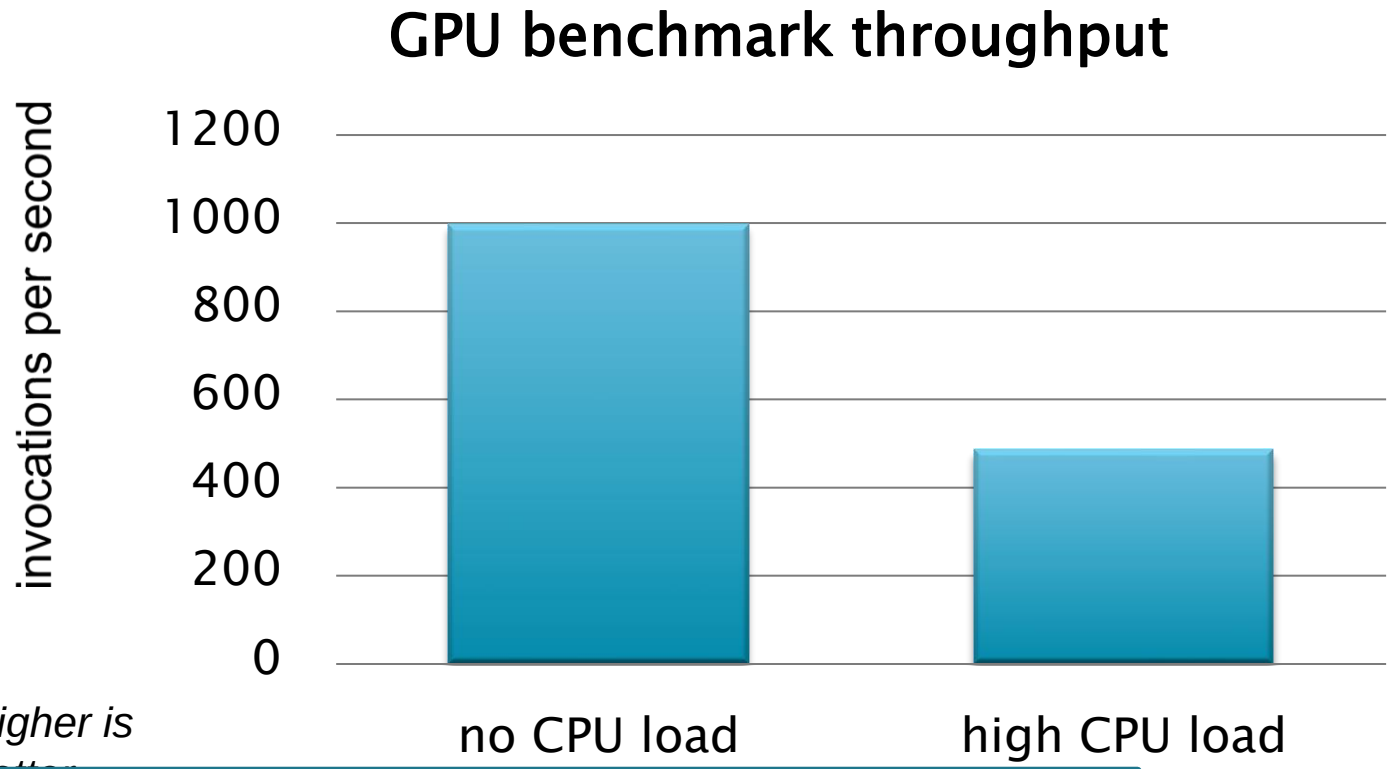
CPU-bound processes hurt GPUs



Higher is better

- Image-convolution in CUDA
- Windows 7 x64 8GB RAM
- Intel Core 2 Quad 2.66GHz
- nVidia GeForce GT230

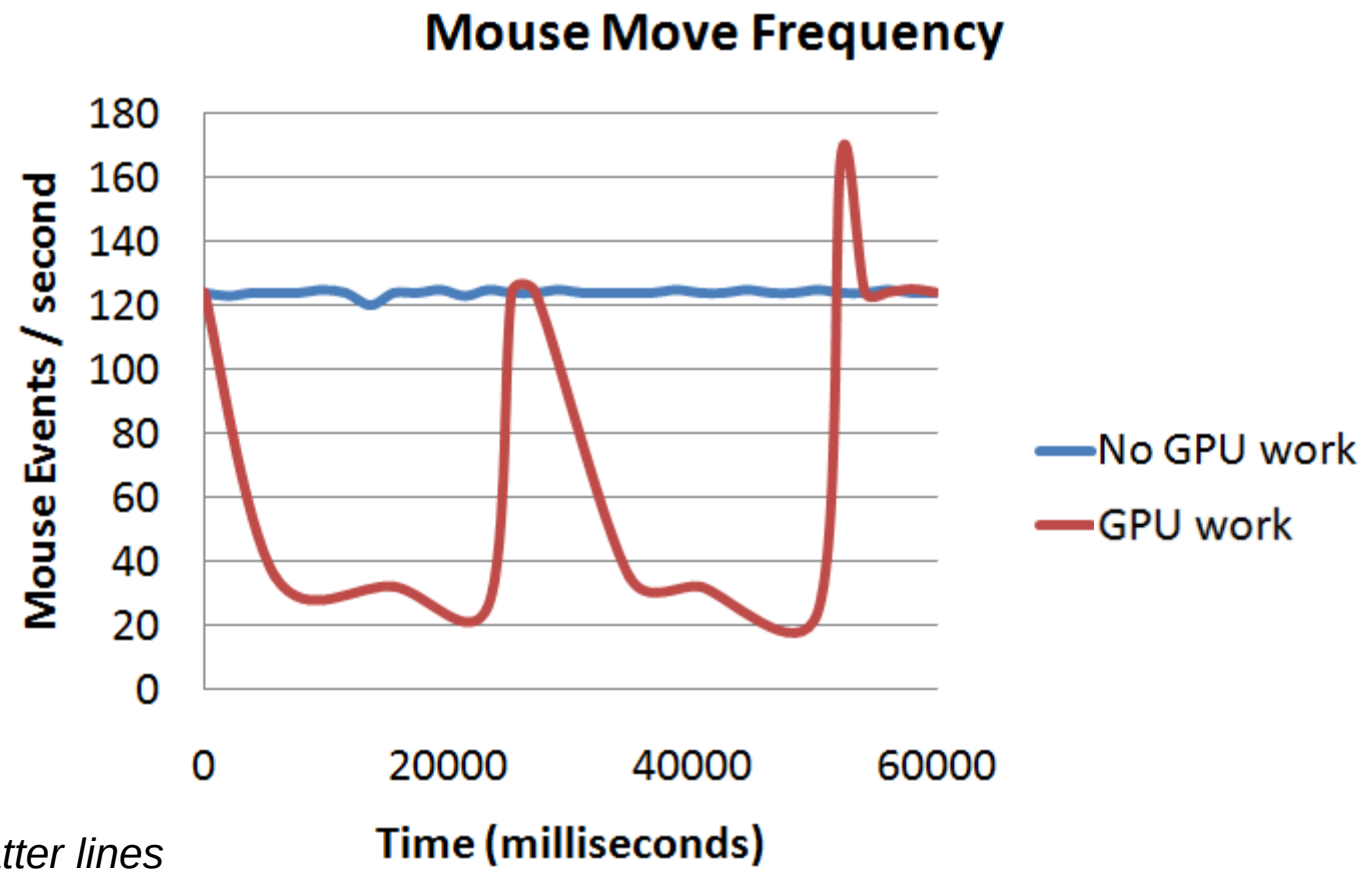
CPU-bound processes hurt GPUs



CPU scheduler and GPU scheduler not integrated!
…many other pathologies arise

age-convolution in CUDA
ndows 7 x64 8GB RAM
el Core 2 Quad 2.66GHz
dia GeForce GT230

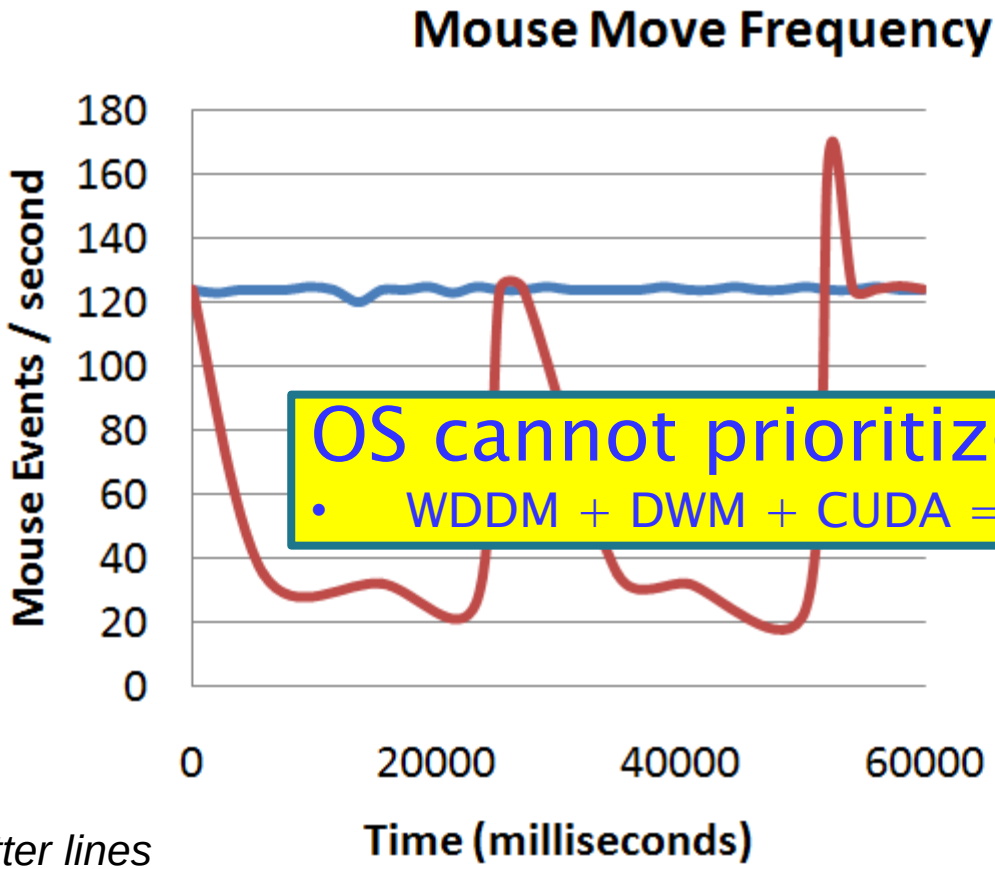
GPU-bound processes hurt CPUs



Flatter lines
Are better

- Windows 7 x64 8GB RAM
- Intel Core 2 Quad 2.66GHz
- nVidia GeForce GT230

GPU-bound processes hurt CPUs



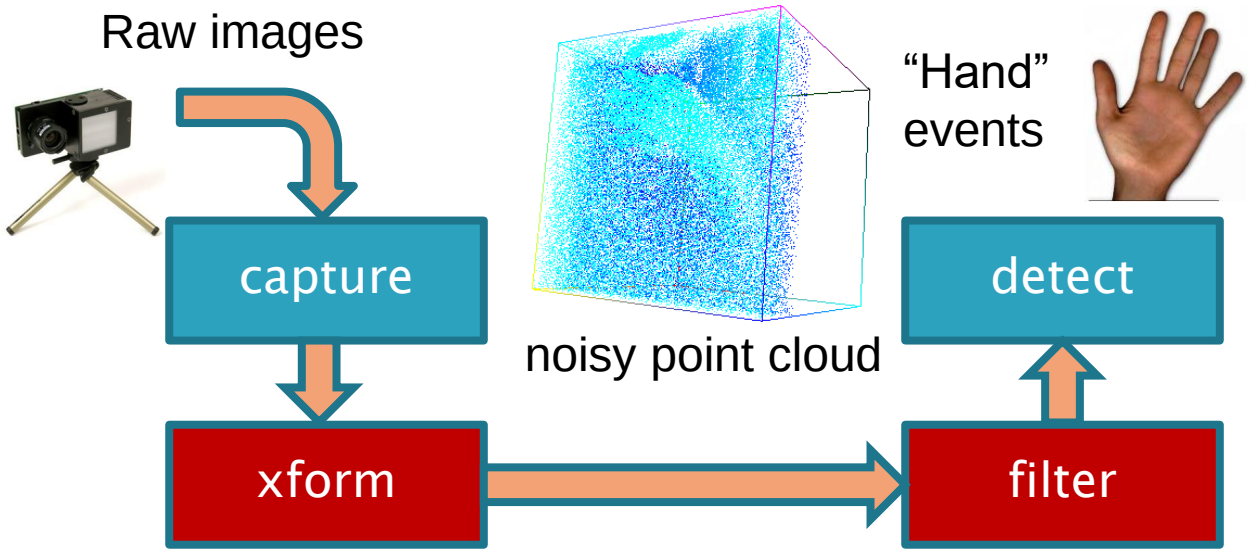
OS cannot prioritize cursor updates

- WDDM + DWM + CUDA == dysfunction

Flatter lines
Are better

- Windows 7 x64 8GB RAM
- Intel Core 2 Quad 2.66GHz
- nVidia GeForce GT230

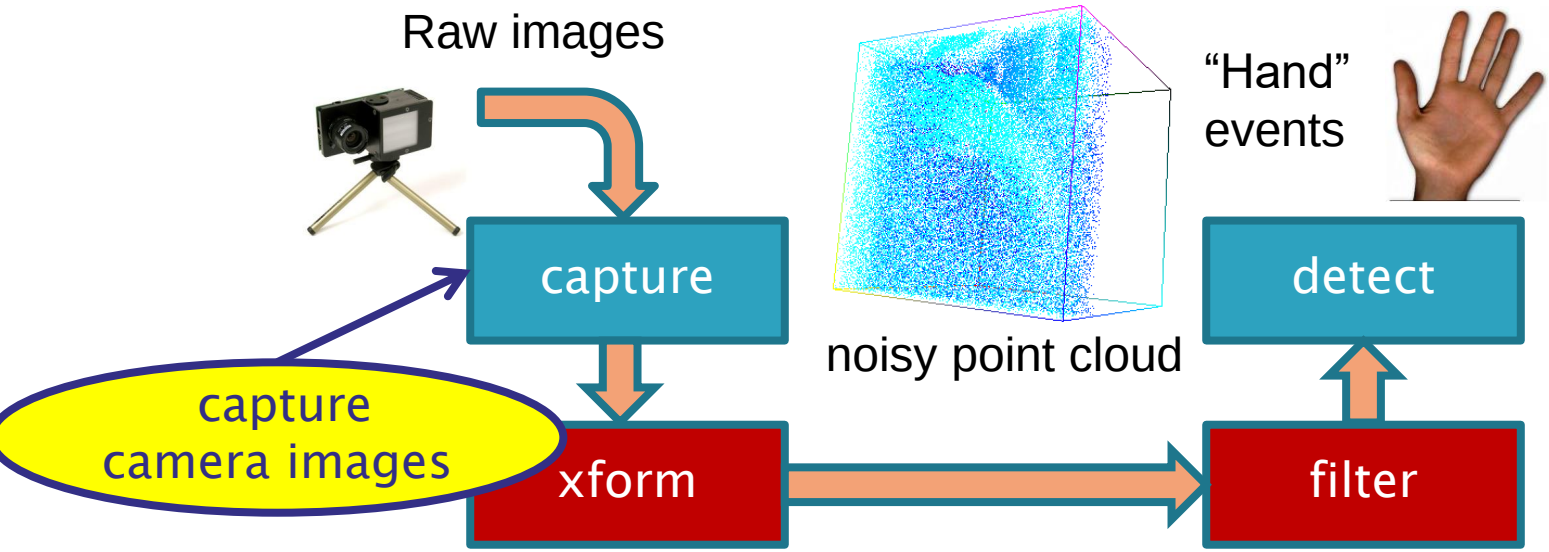
More déjà vu: Gestural Interface



- High data rates
- Data-parallel algorithms
  … good fit for GPU

NOT Kinect: this is a harder problem!

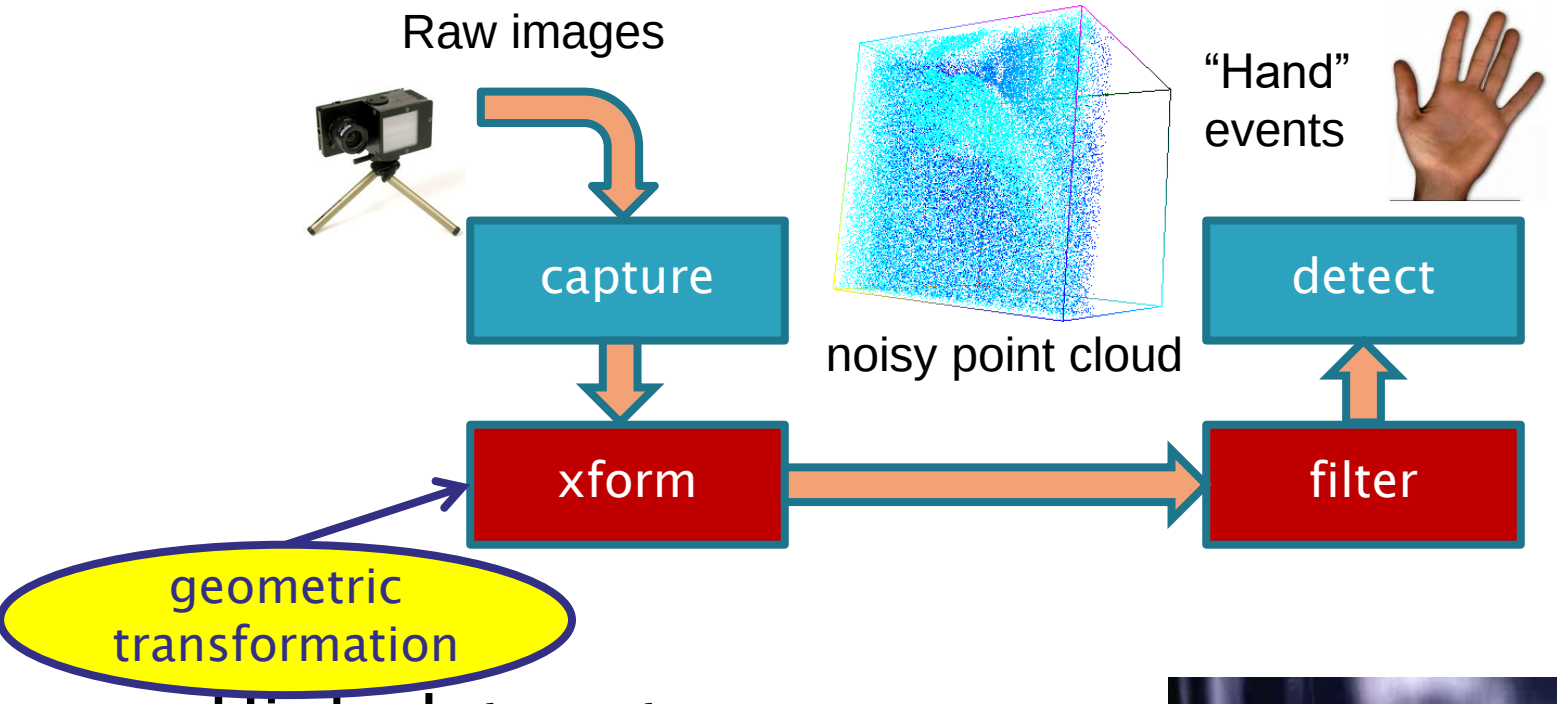
More déjà vu: Gestural Interface



- High data rates
- Data-parallel algorithms
  … good fit for GPU

NOT Kinect: this is a harder problem!

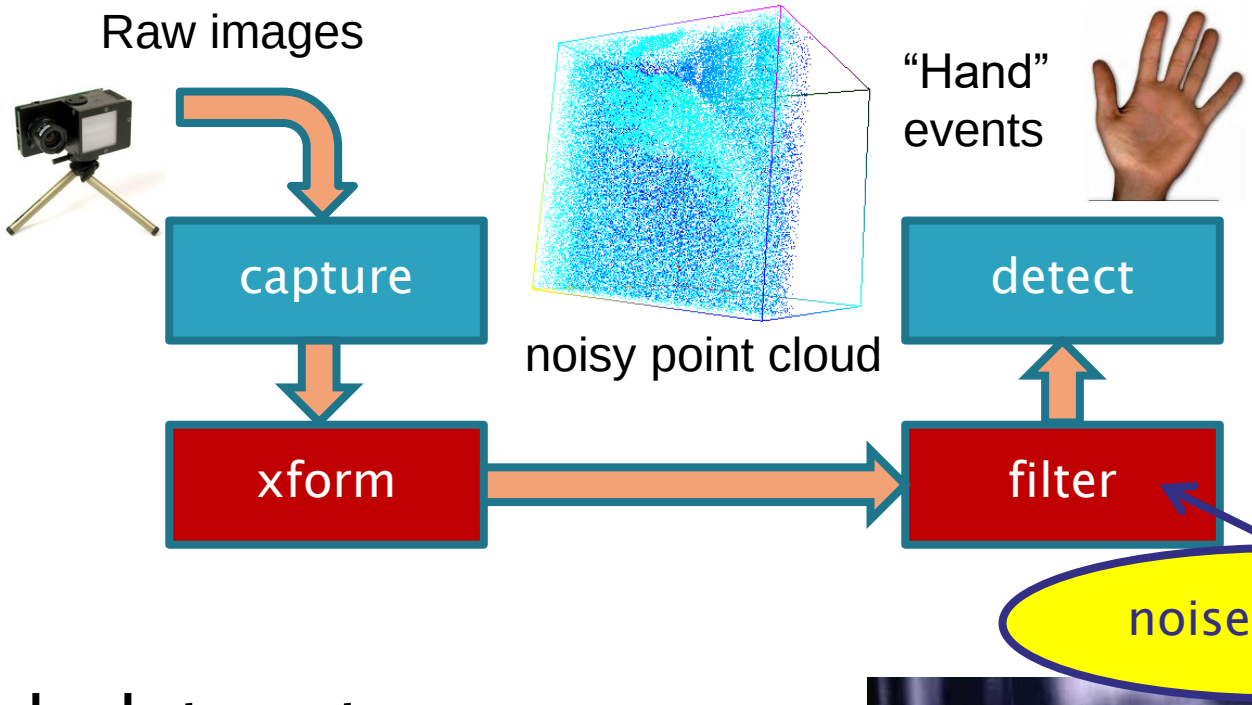
More déjà vu: Gestural Interface



- High data rates
- Data-parallel algorithms … good fit for GPU

NOT Kinect: this is a harder problem!

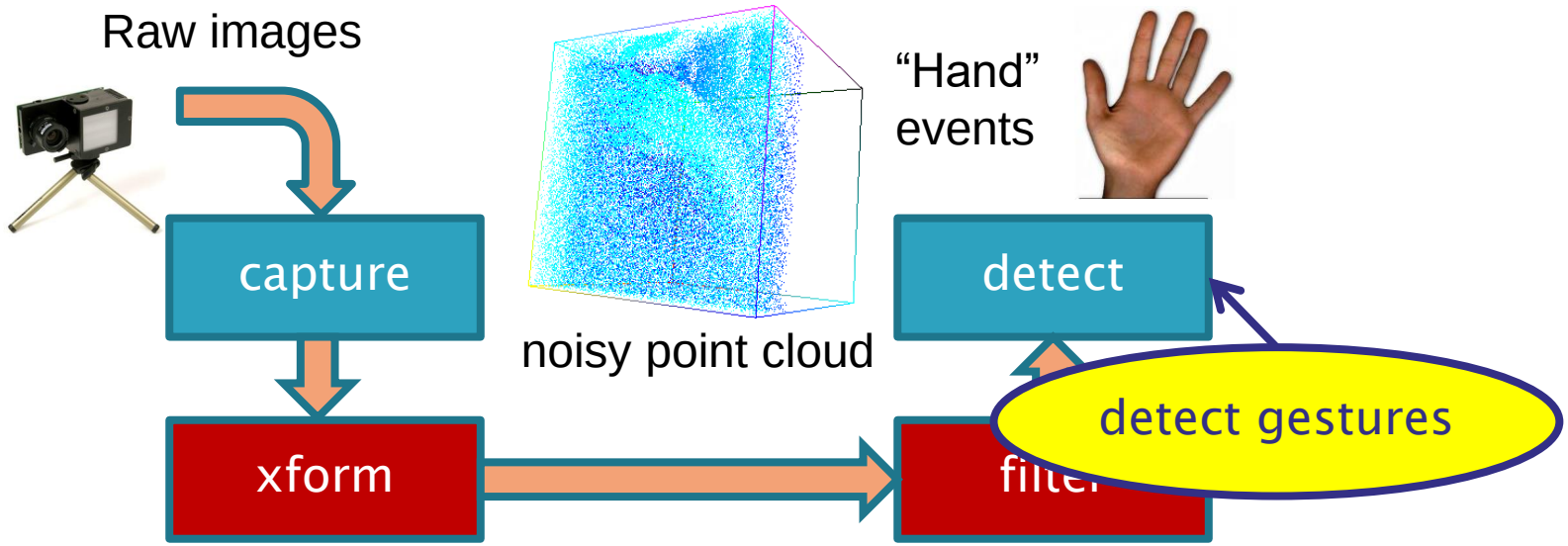
More déjà vu: Gestural Interface



- High data rates
- Data-parallel algorithms … good fit for GPU

NOT Kinect: this is a harder problem!

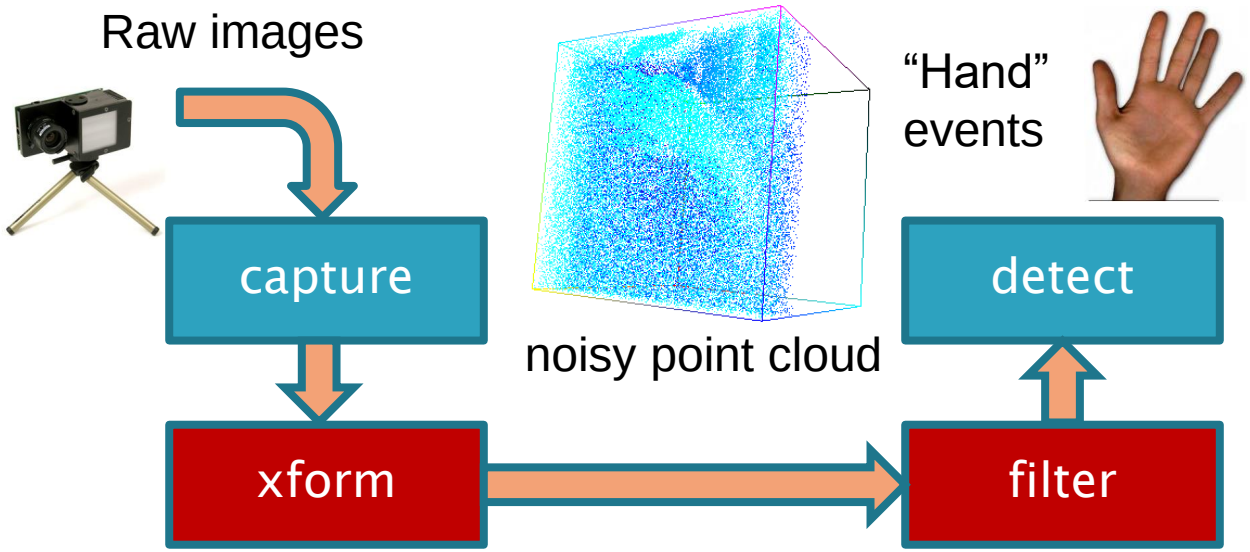
More déjà vu: Gestural Interface



- High data rates
- Data-parallel algorithms … good fit for GPU

NOT Kinect: this is a harder problem!

More déjà vu: Gestural Interface



- High data rates
- Data-parallel algorithms … good fit for GPU

NOT Kinect: this is a harder problem!

What We'd Like To Do

```
#> capture | xform | filter | detect &
```

- Modular design
  - flexibility, reuse
- Utilize heterogeneous hardware
  - Data-parallel components → GPU
  - Sequential components → CPU
- Using OS provided tools
  - processes, pipes

What We'd Like To Do

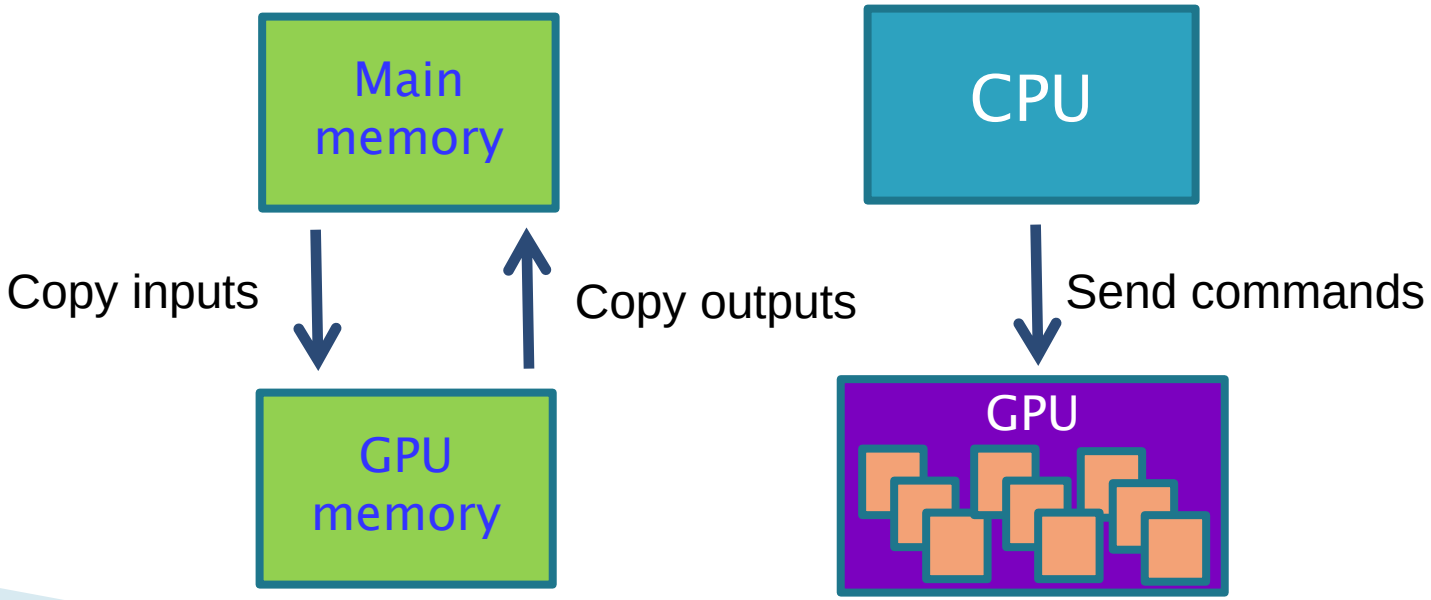
```
#> capture | xform | filter | detect &
```

CPU GPU GPU CPU

- Modular design
  - flexibility, reuse
- Utilize heterogeneous hardware
  - Data-parallel components → GPU
  - Sequential components → CPU
- Using OS provided tools
  - processes, pipes

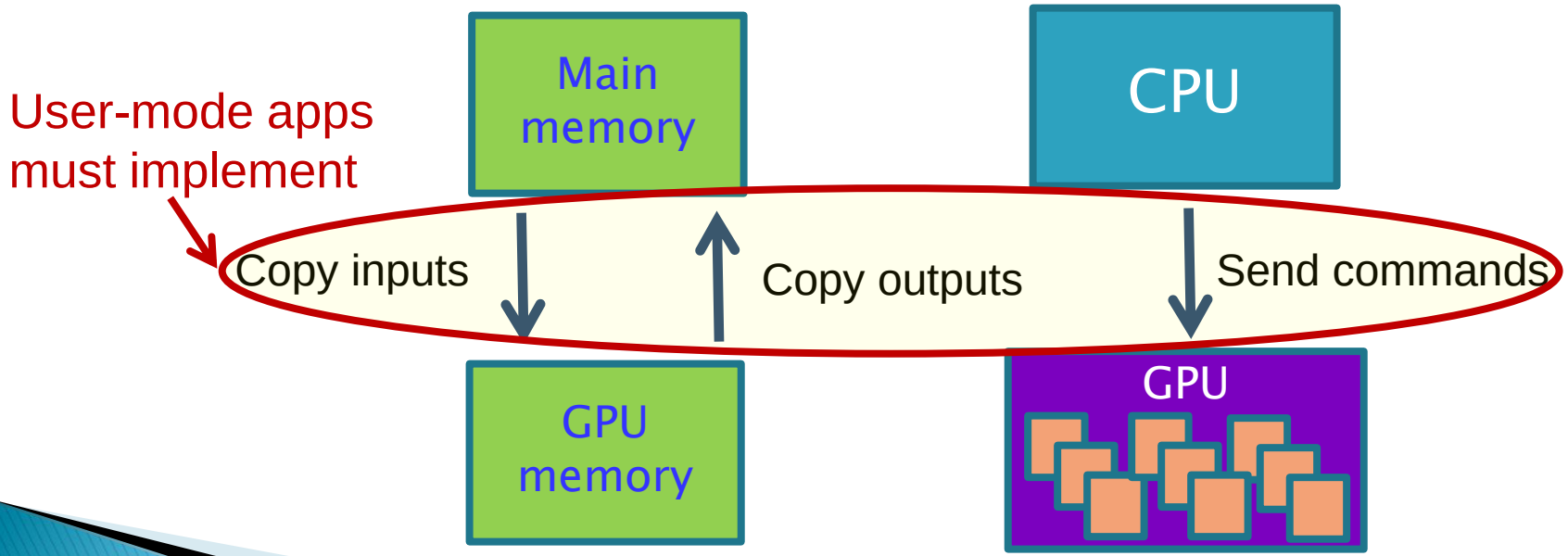
GPU Execution model

- GPUs cannot run OS: different ISA
- Disjoint memory space, no coherence
- Host CPU must manage GPU execution
  - Program inputs explicitly transferred/bound at runtime
  - Device buffers pre-allocated



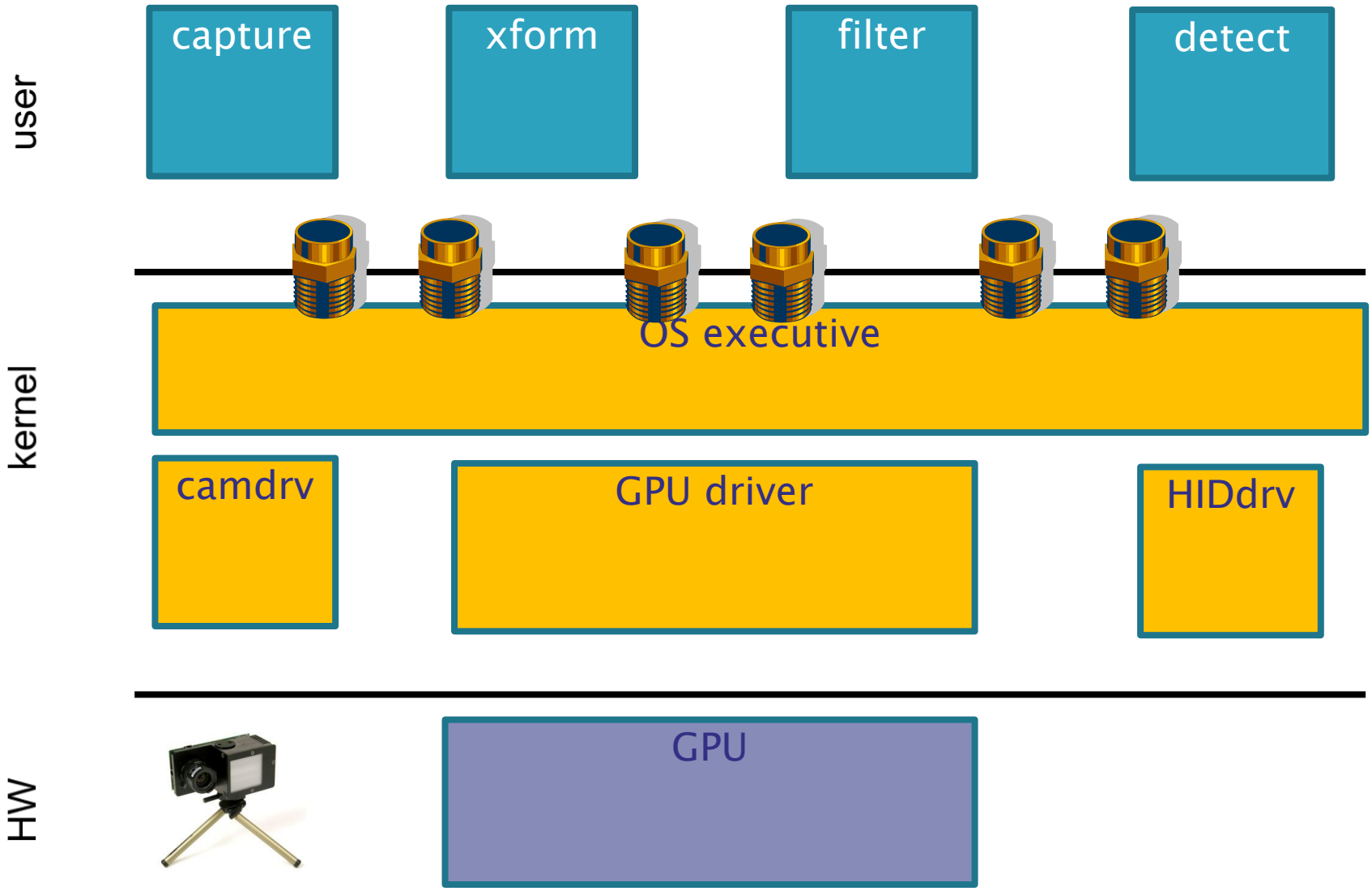
GPU Execution model

- GPUs cannot run OS: different ISA
- Disjoint memory space, no coherence
- Host CPU must manage GPU execution
  - Program inputs explicitly transferred/bound at runtime
  - Device buffers pre-allocated



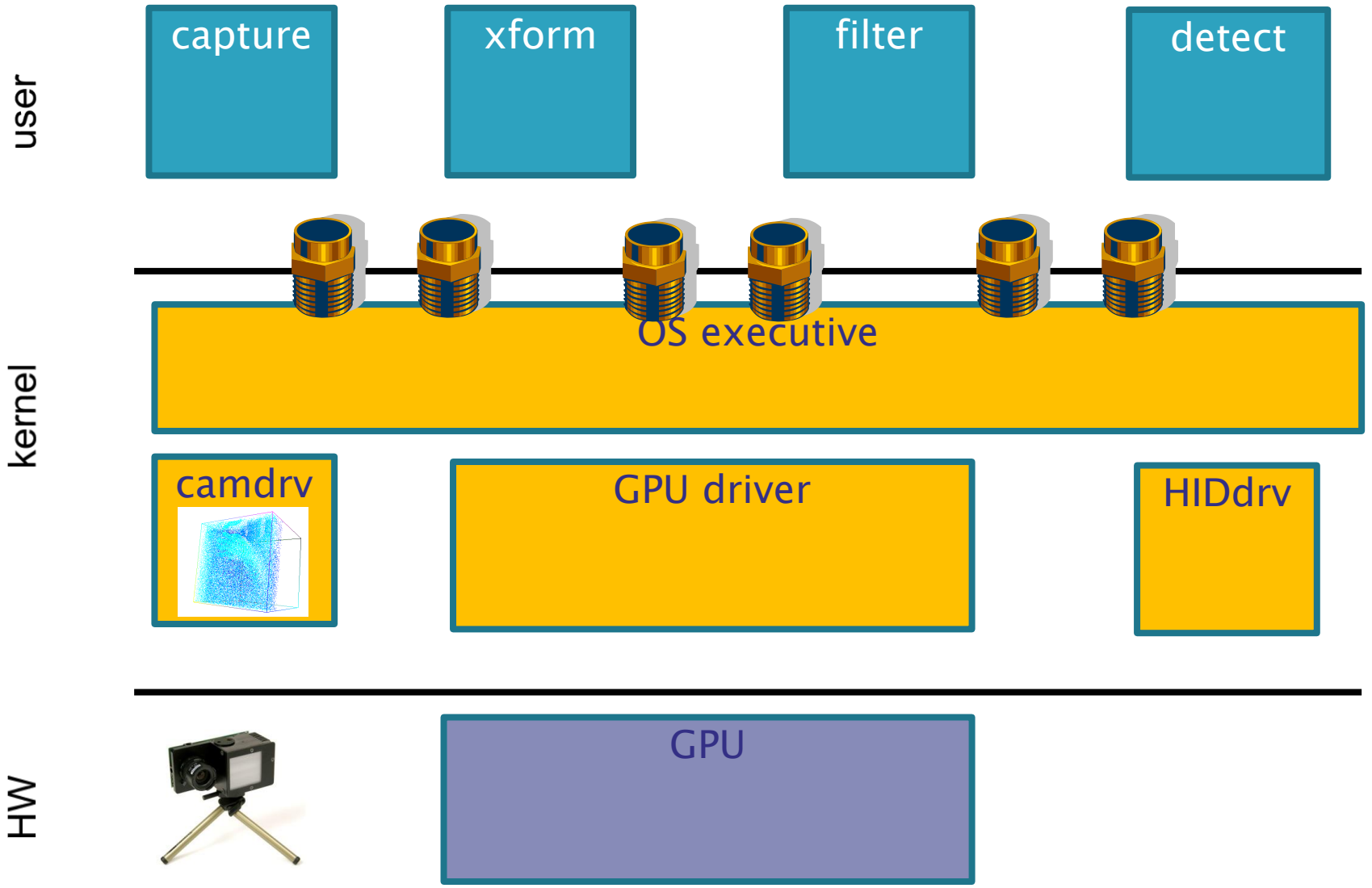
Data migration

```
#> capture | xform | filter | detect &
```



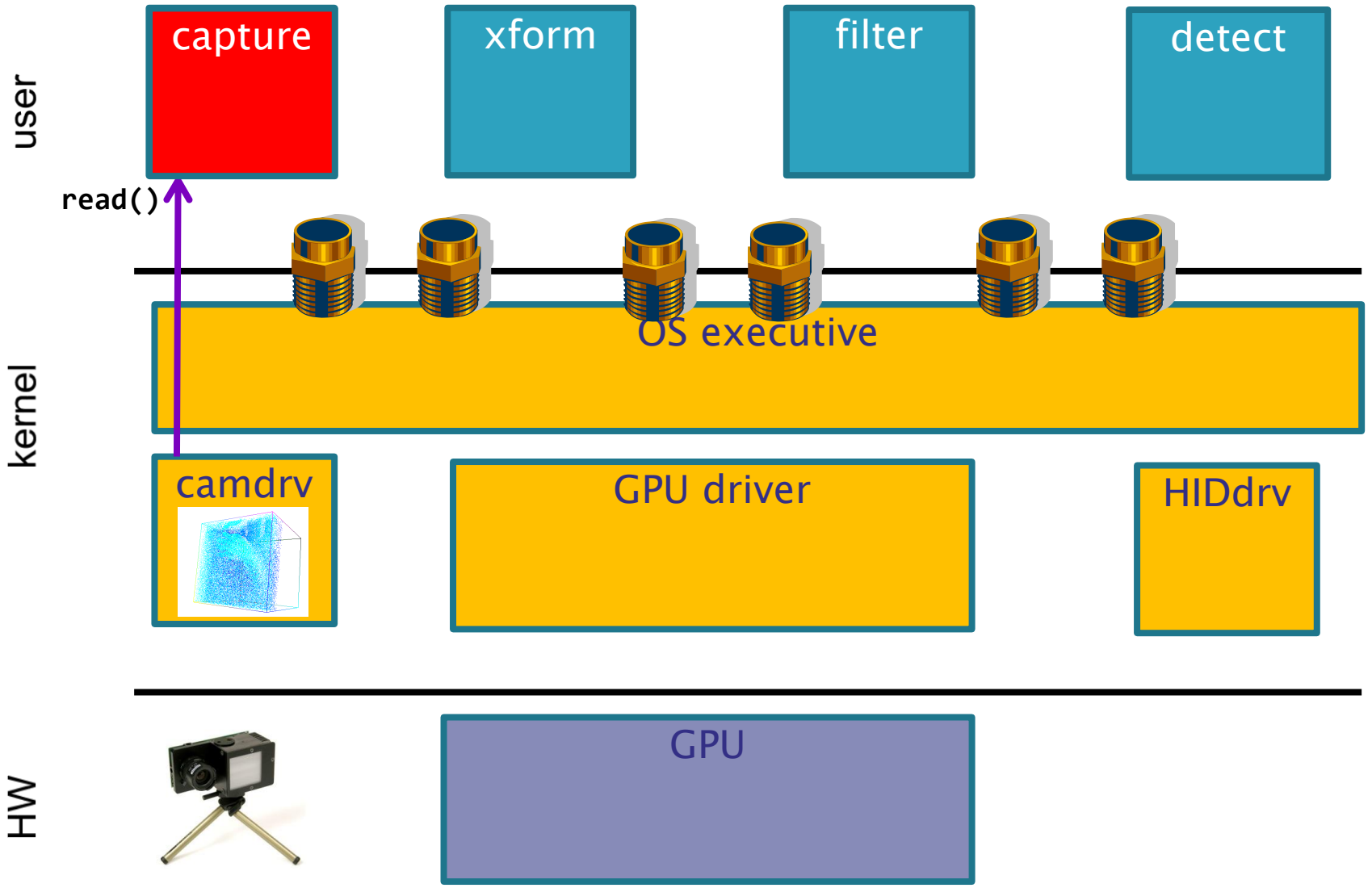
Data migration

```
#> capture | xform | filter | detect &
```



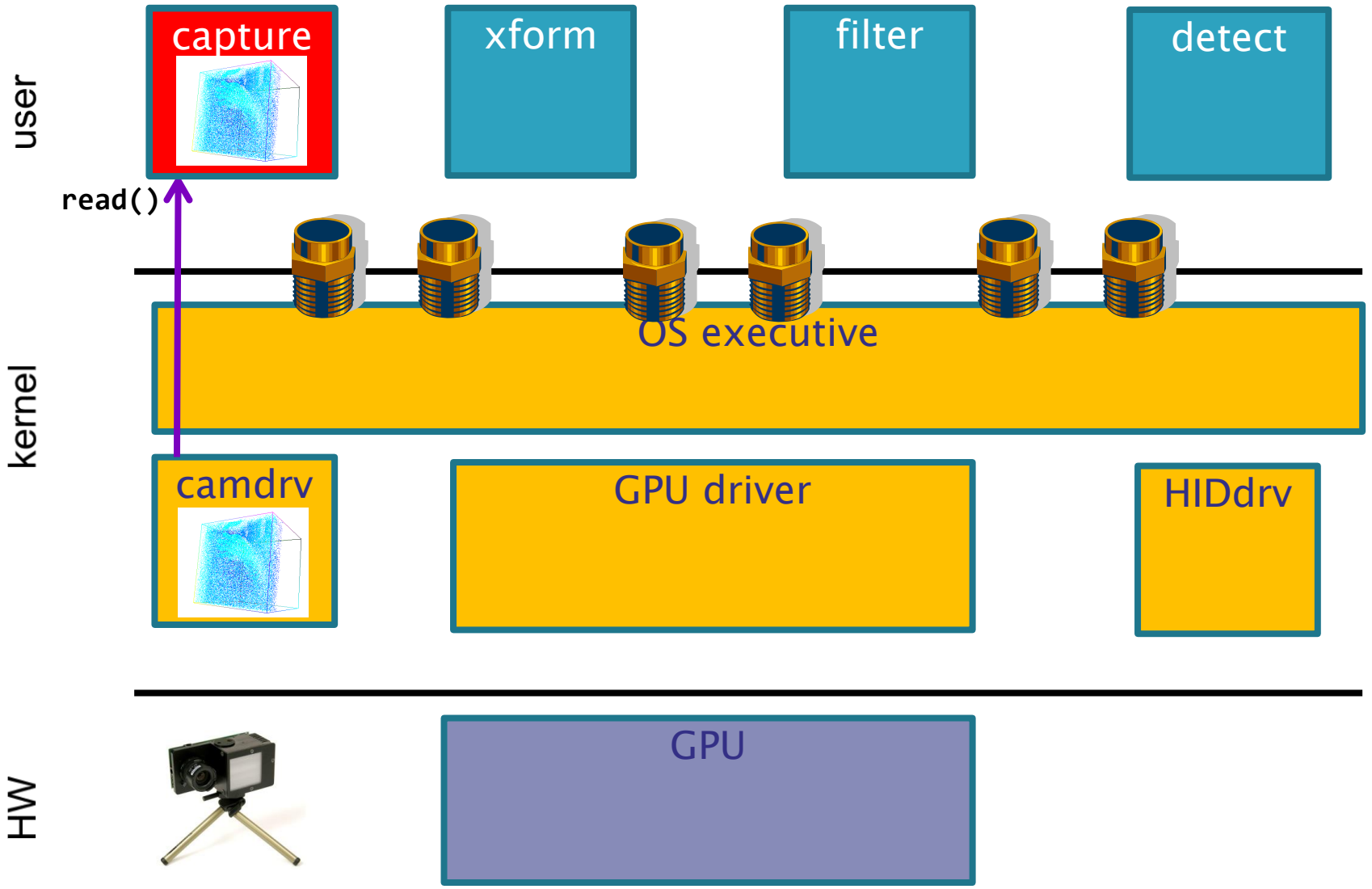
Data migration

```
#> capture | xform | filter | detect &
```



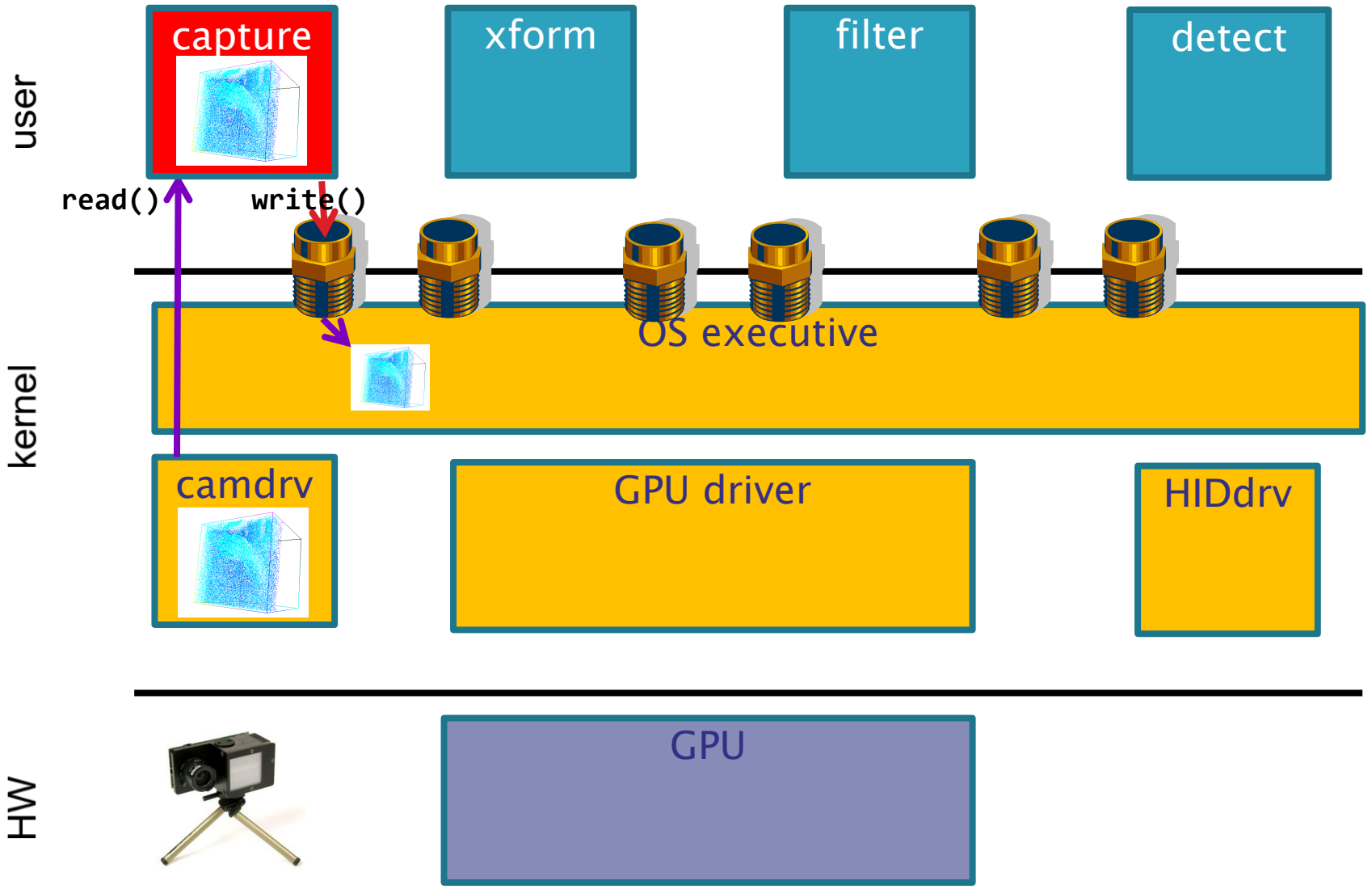
Data migration

```
#> capture | xform | filter | detect &
```



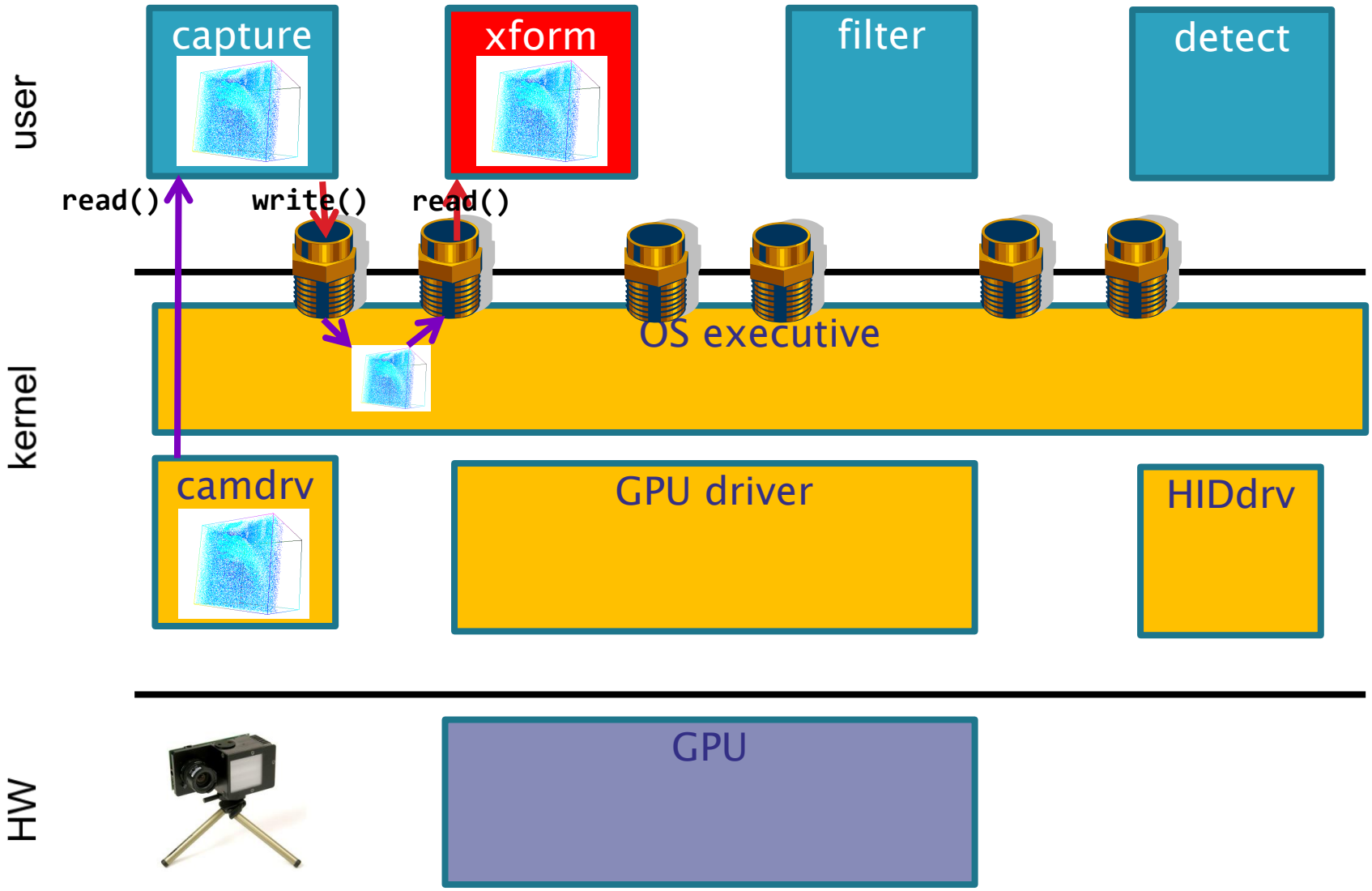
Data migration

```
#> capture | xform | filter | detect &
```



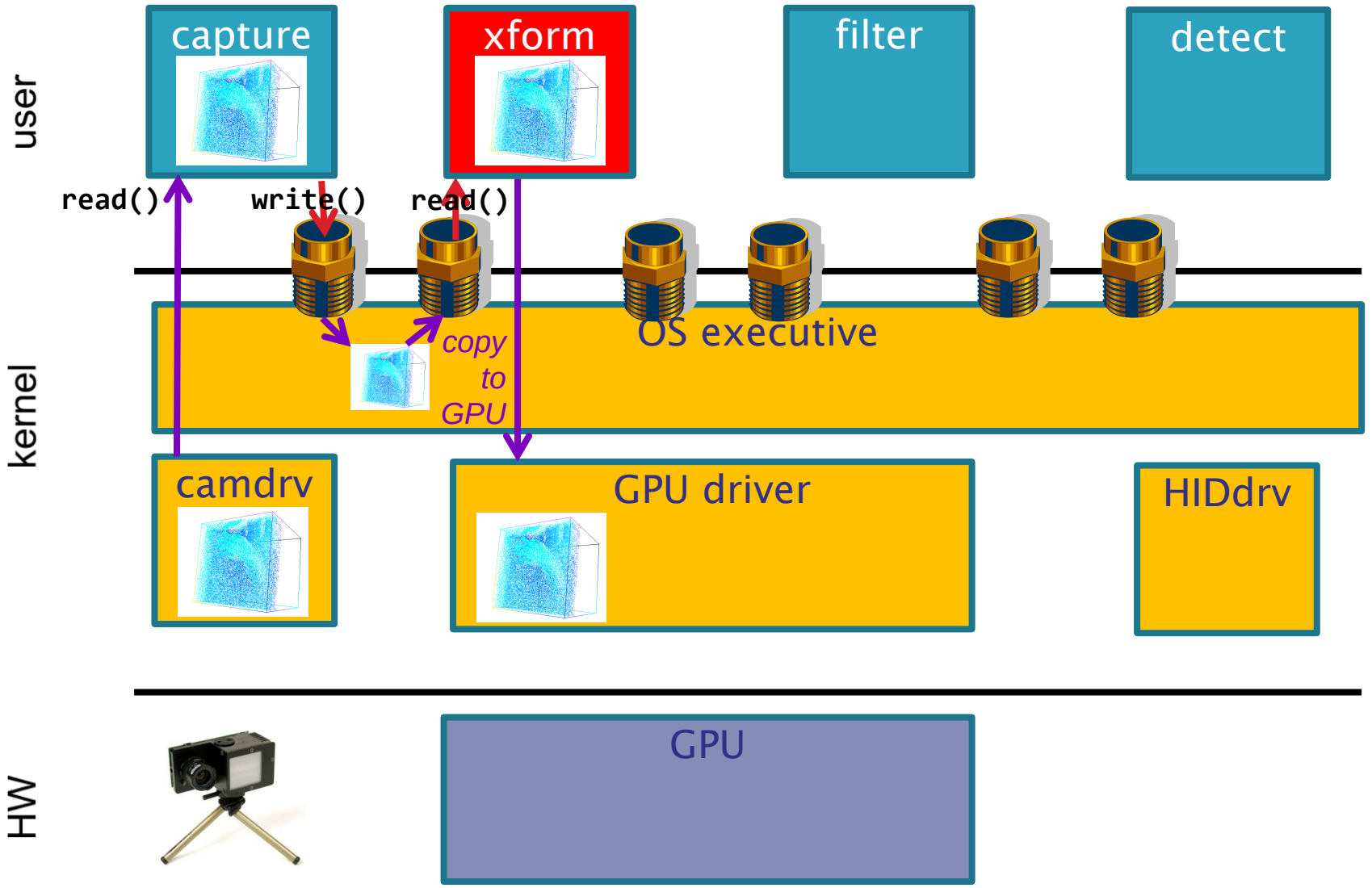
Data migration

```
#> capture | xform | filter | detect &
```



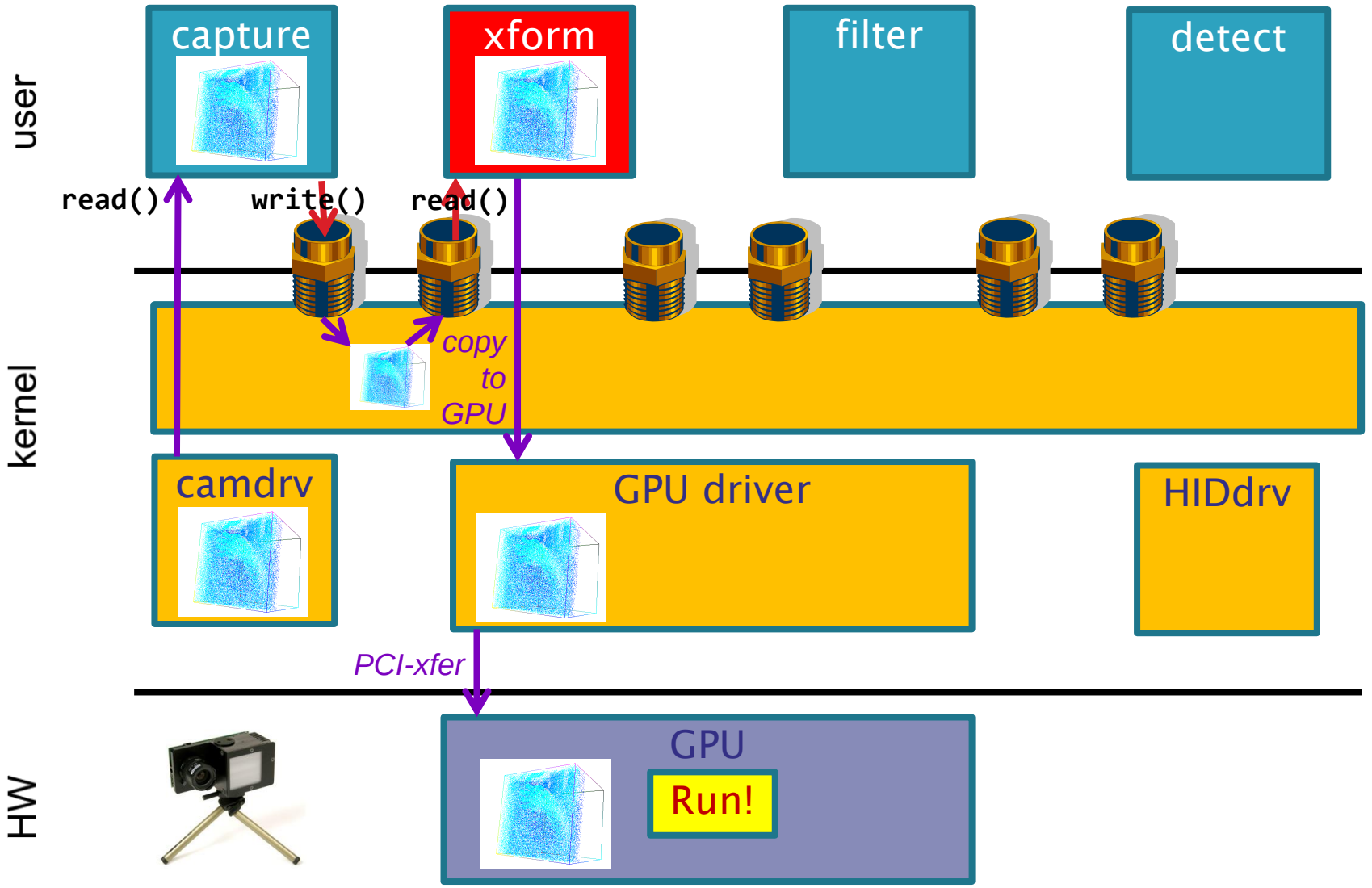
Data migration

```
#> capture | xform | filter | detect &
```



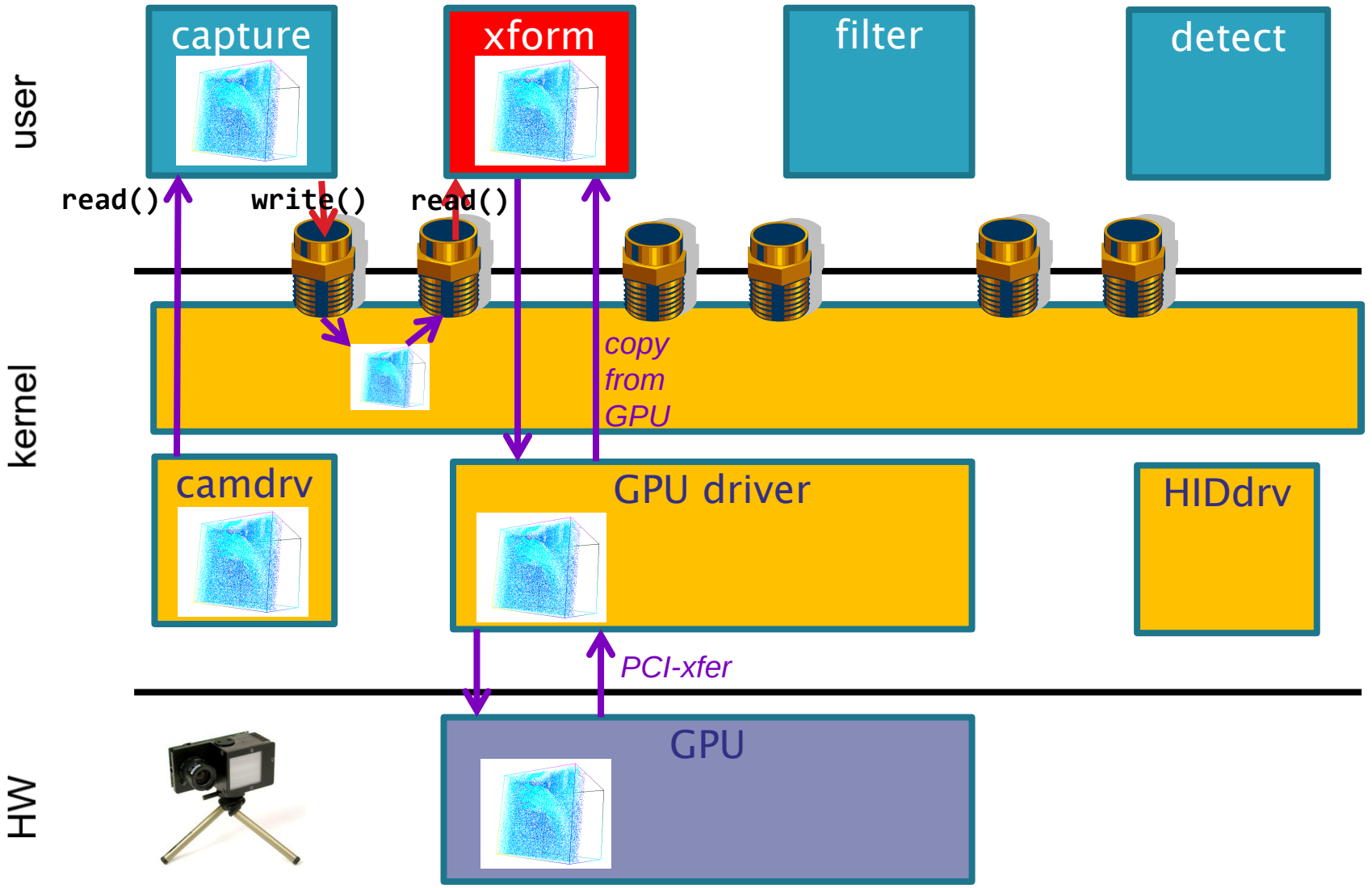
Data migration

```
#> capture | xform | filter | detect &
```



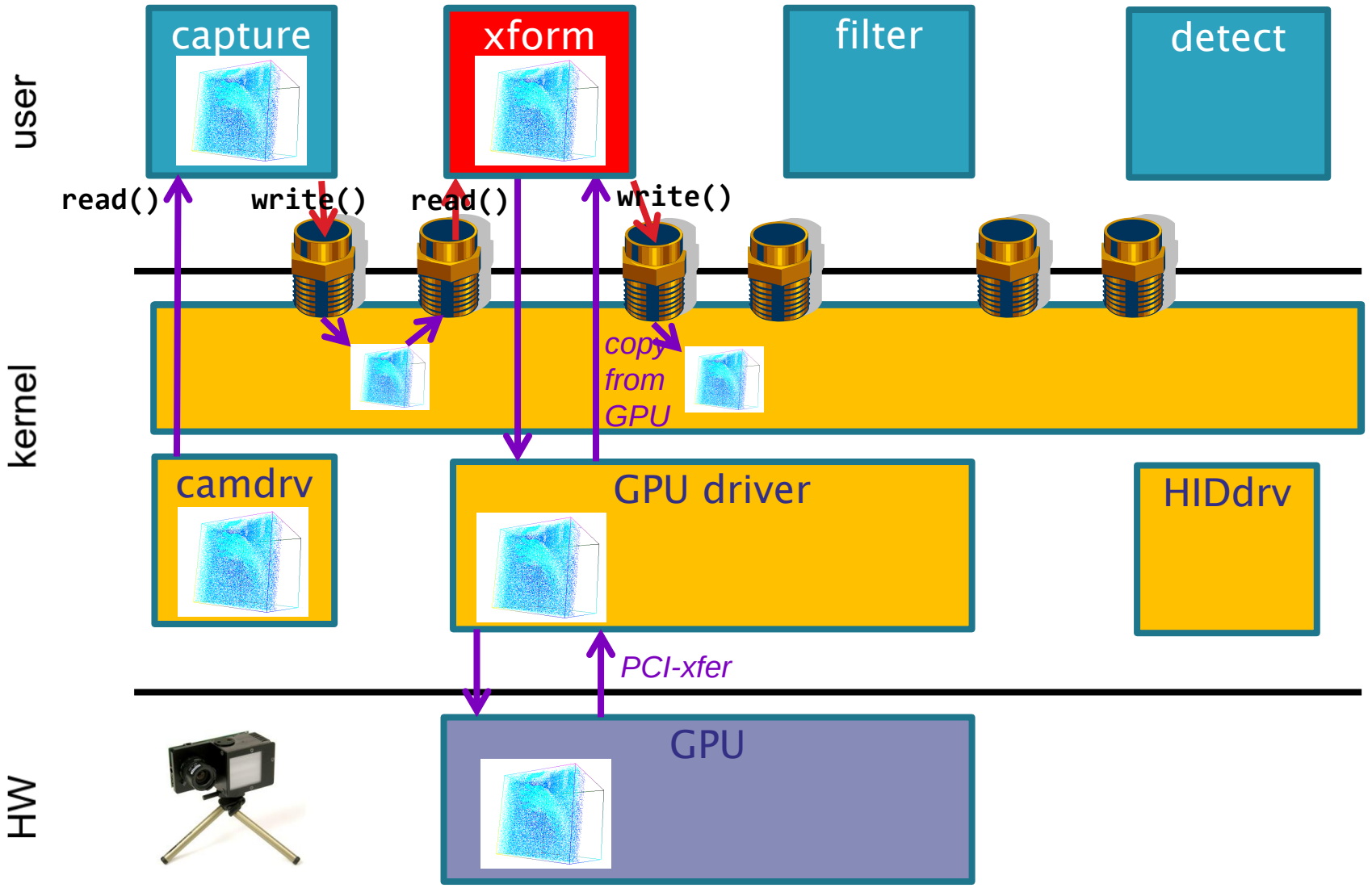
Data migration

```
#> capture | xform | filter | detect &
```



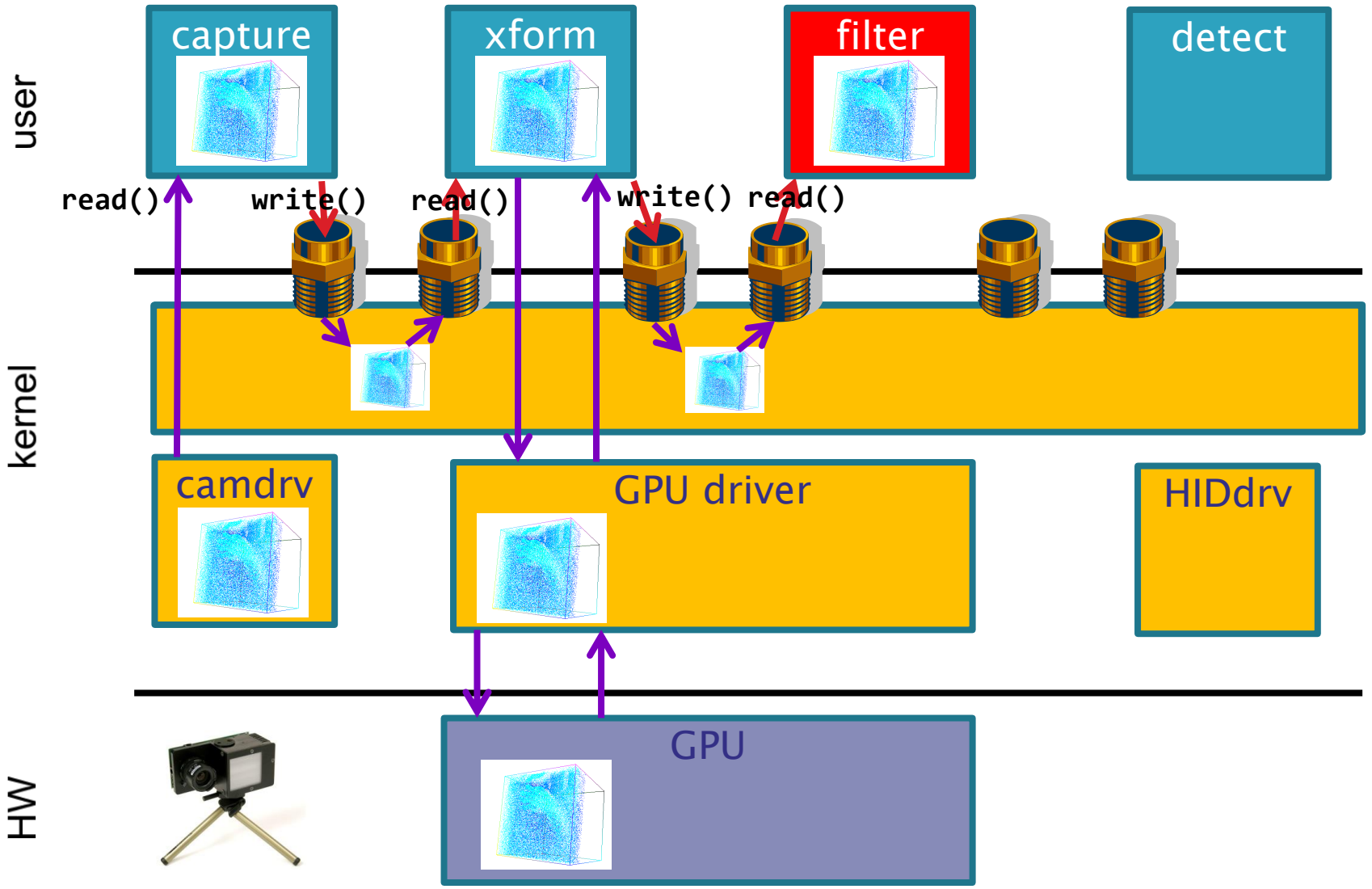
Data migration

```
#> capture | xform | filter | detect &
```



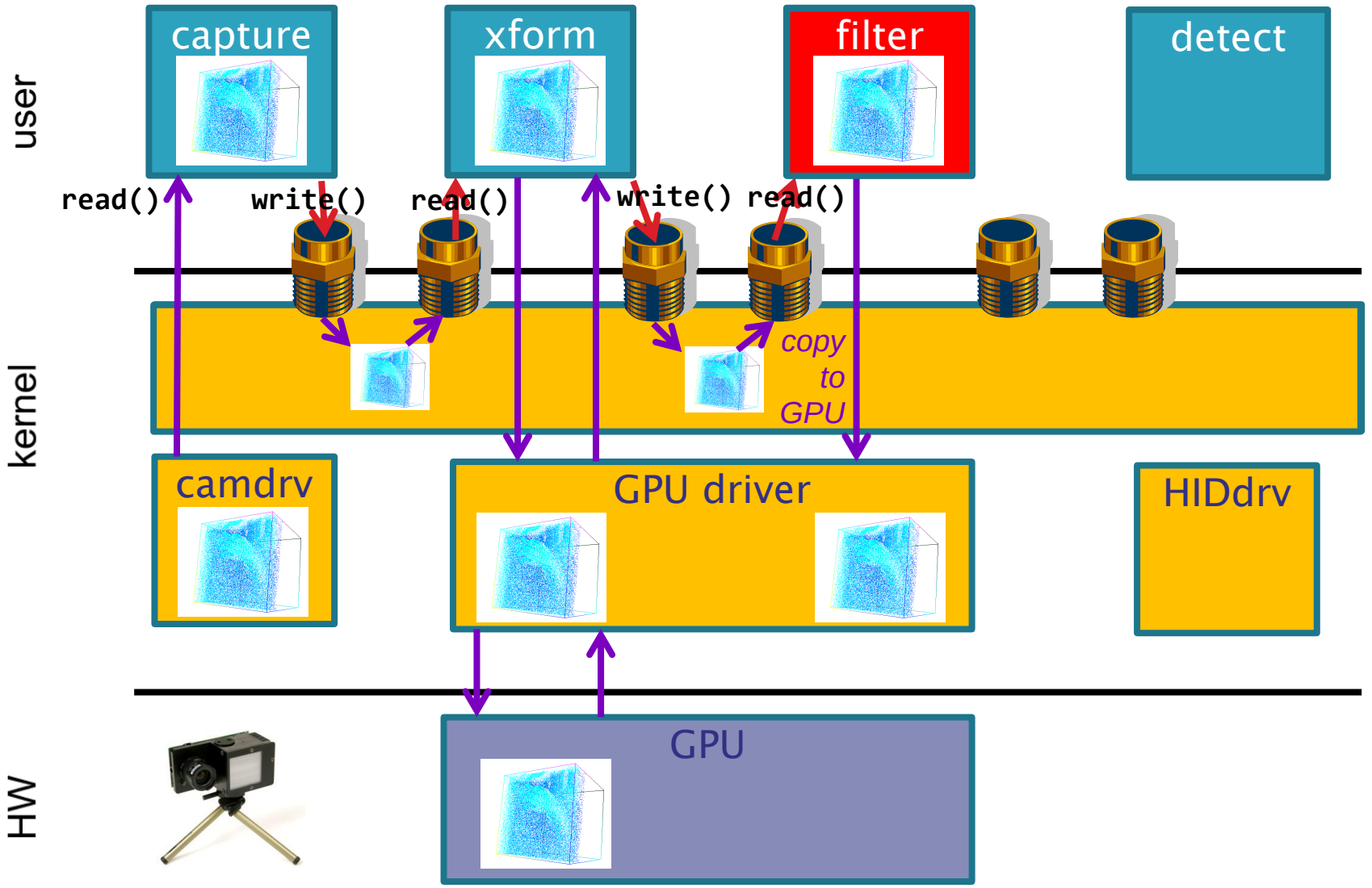
Data migration

```
#> capture | xform | filter | detect &
```



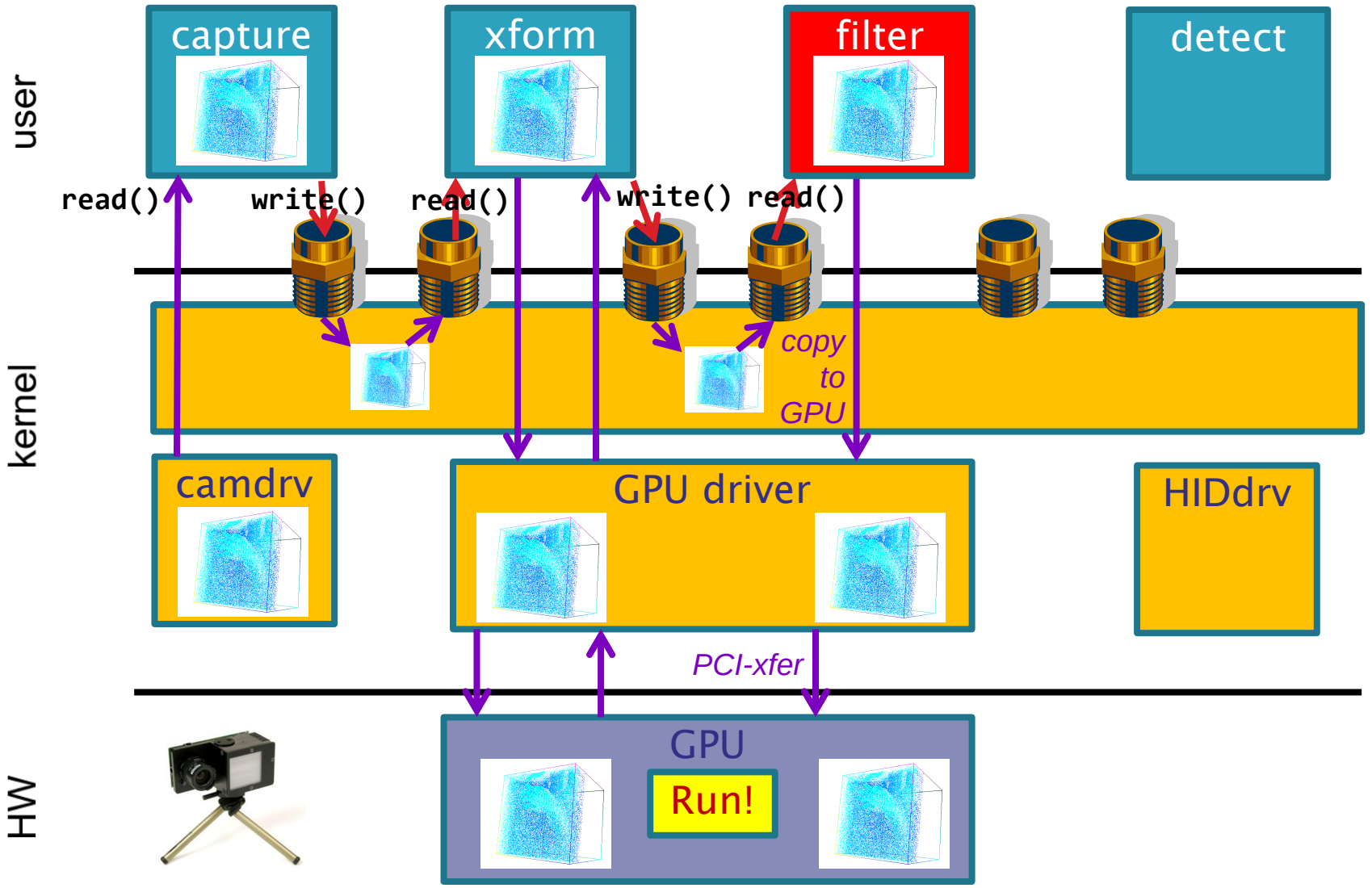
Data migration

```
#> capture | xform | filter | detect &
```



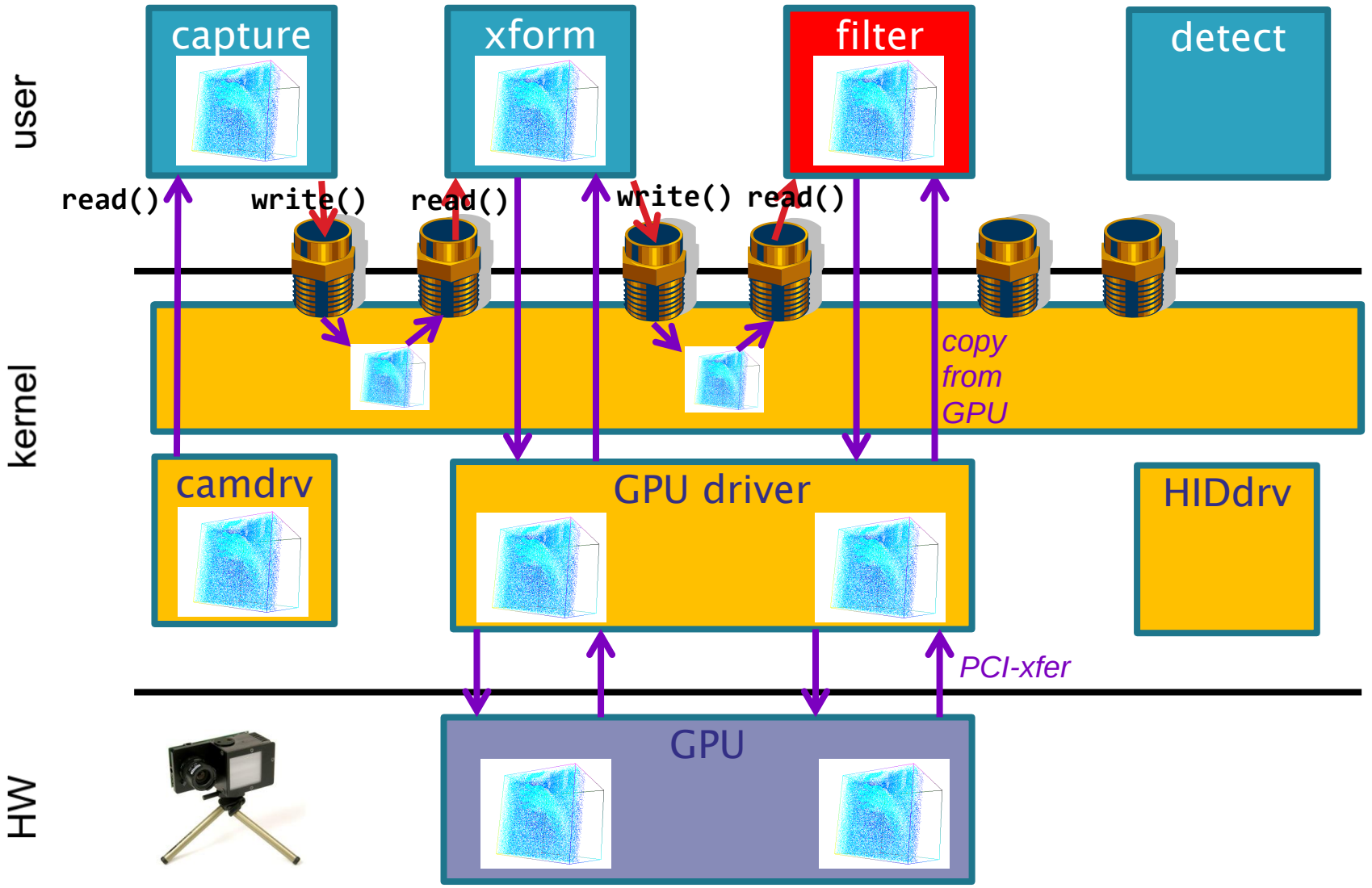
Data migration

```
#> capture | xform | filter | detect &
```



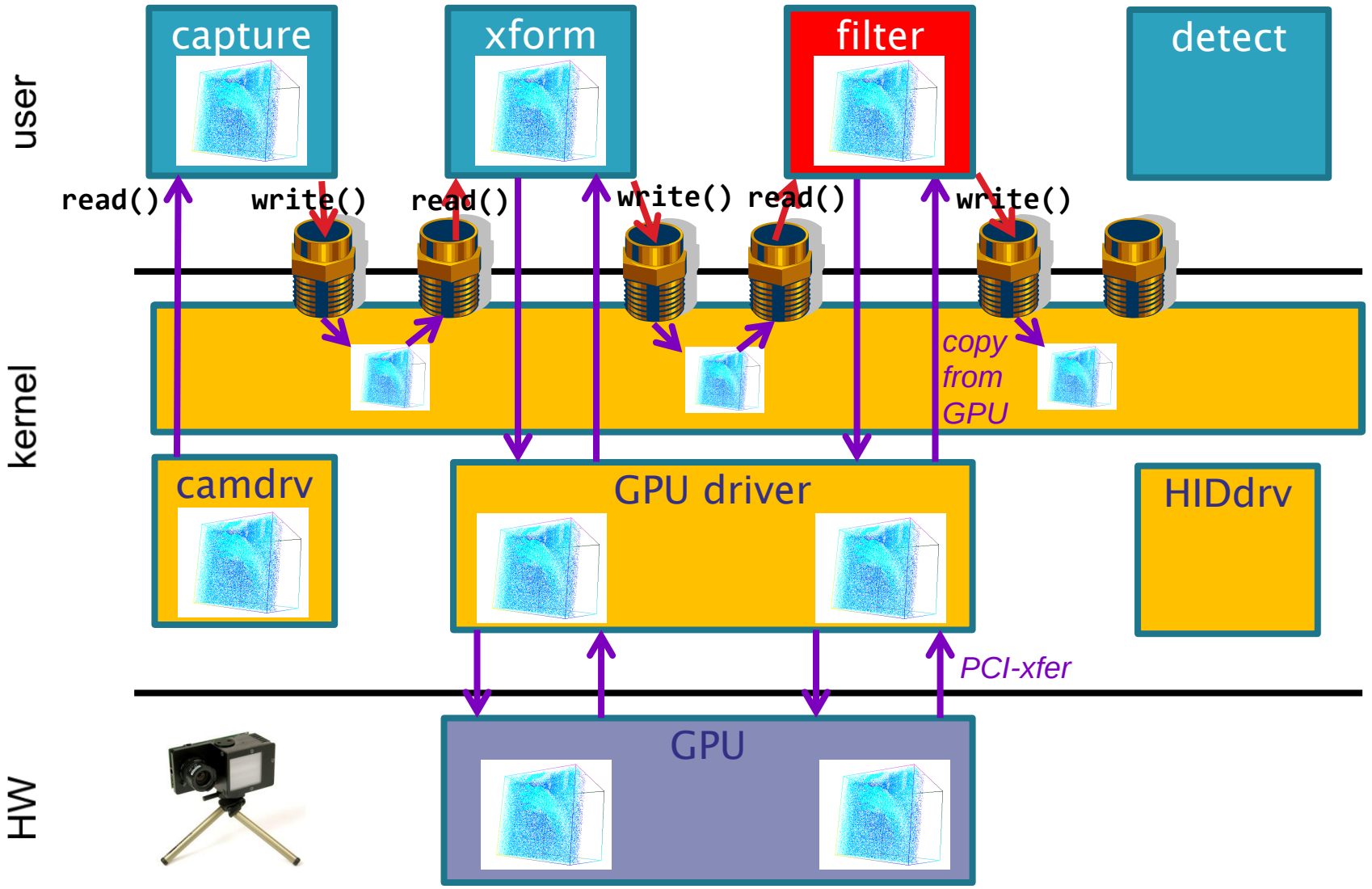
Data migration

```
#> capture | xform | filter | detect &
```



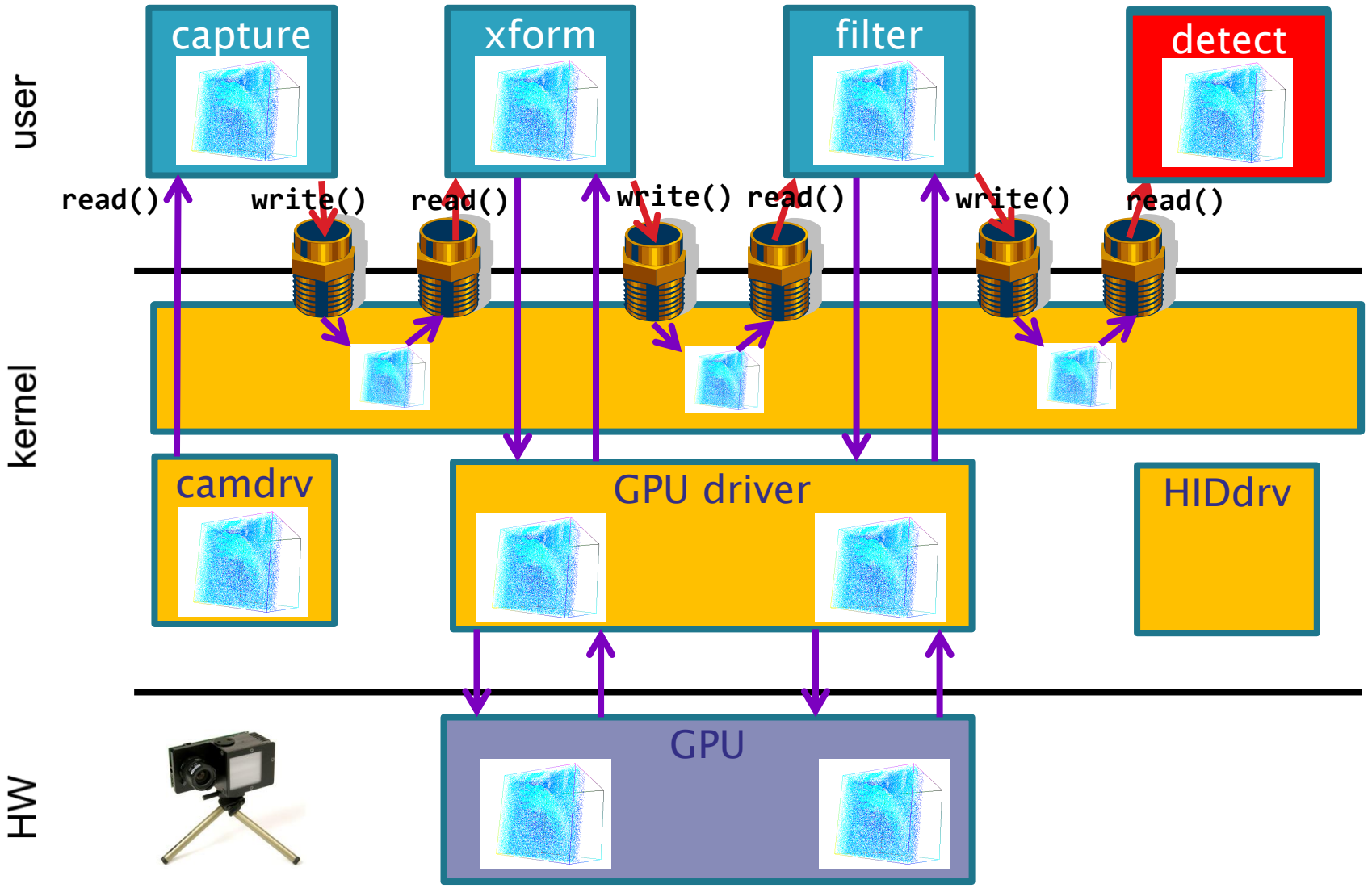
Data migration

```
#> capture | xform | filter | detect &
```



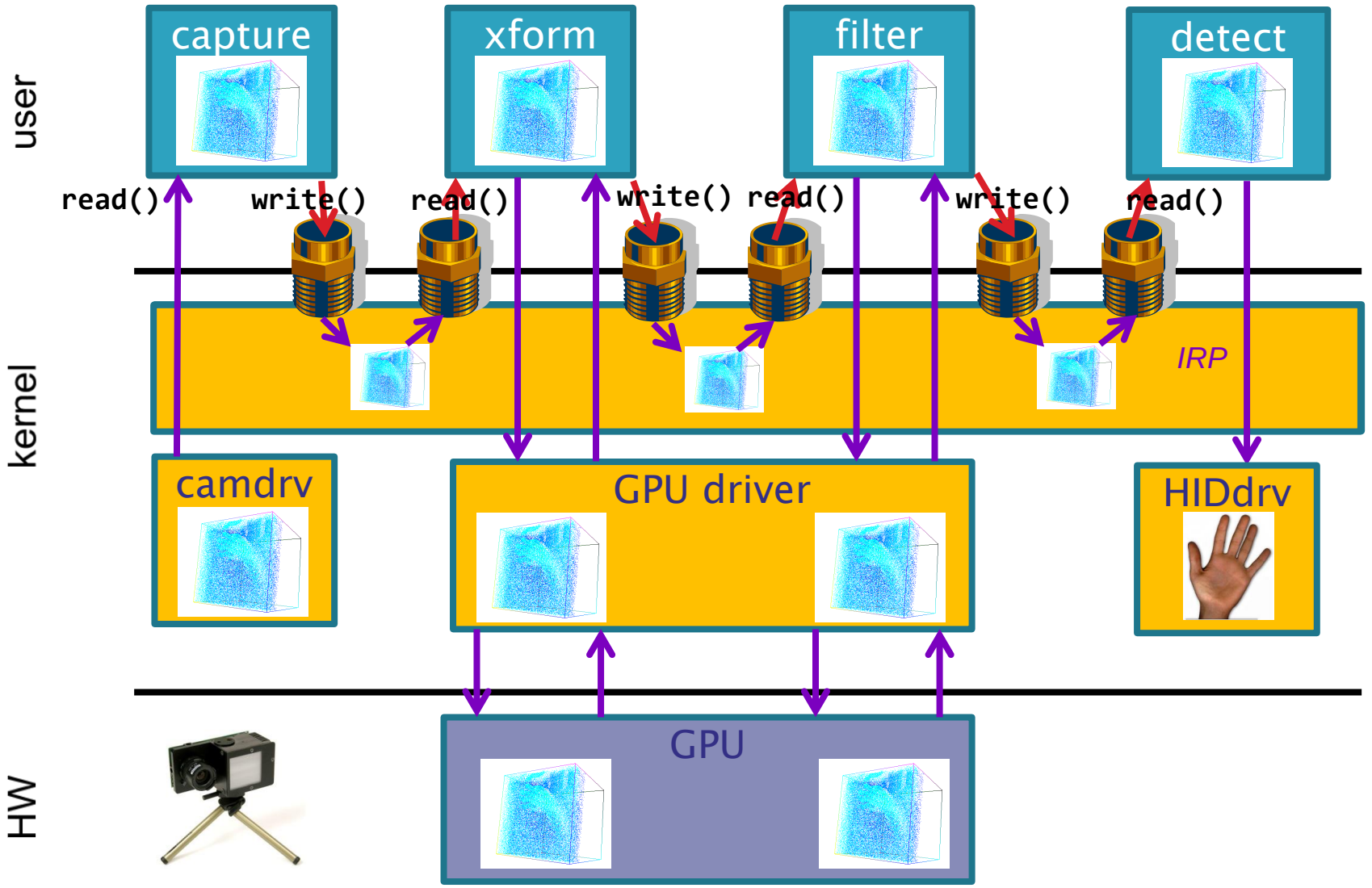
Data migration

```
#> capture | xform | filter | detect &
```

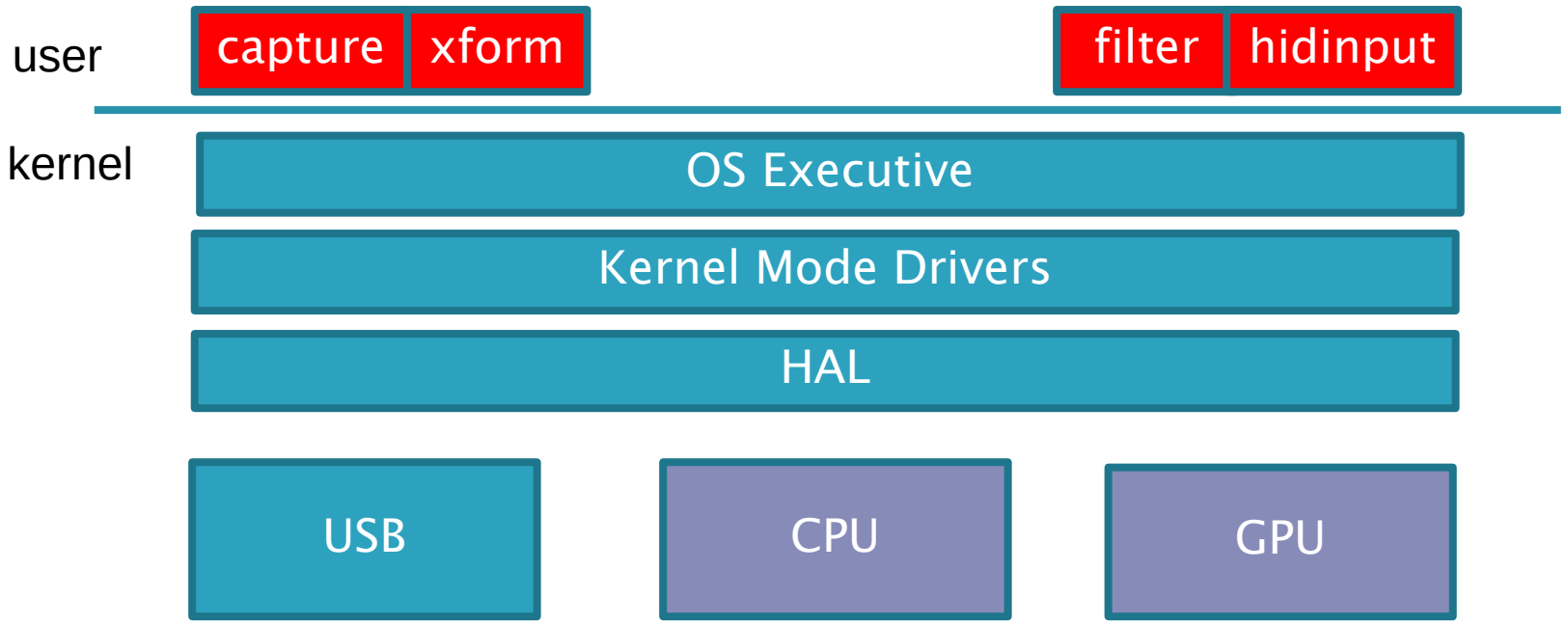


Data migration

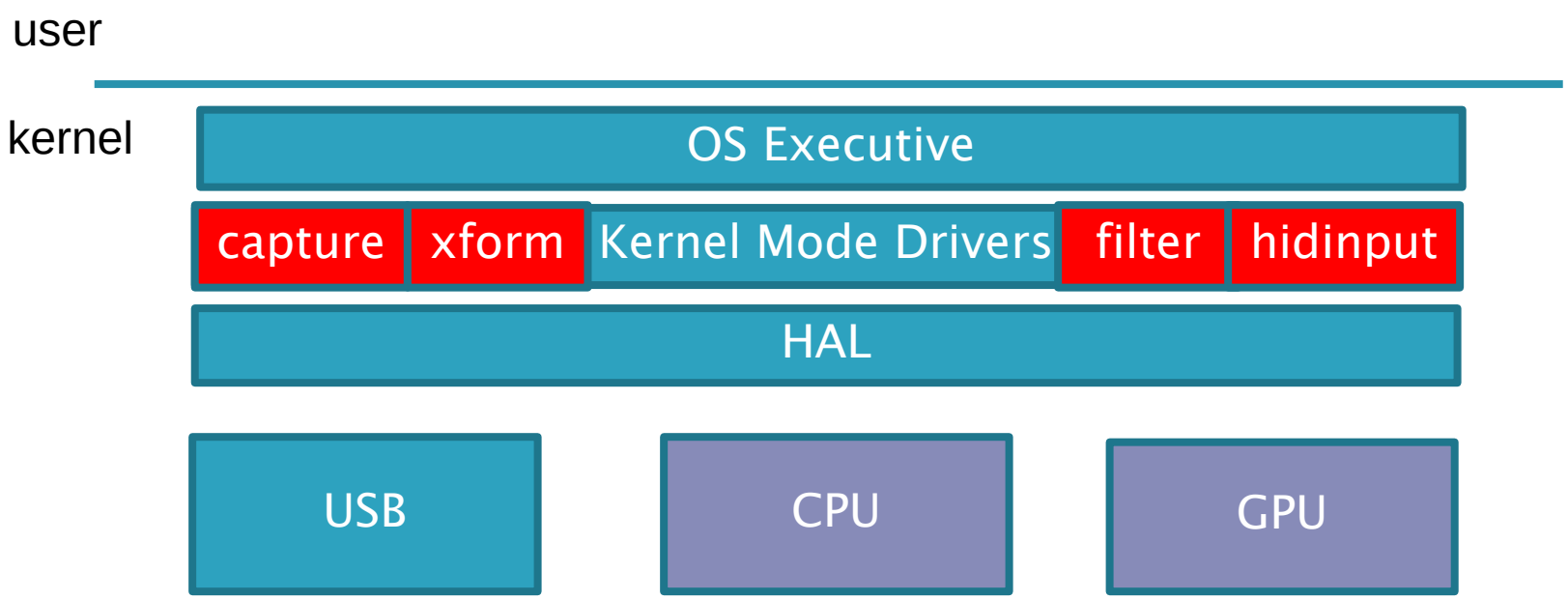
```
#> capture | xform | filter | detect &
```



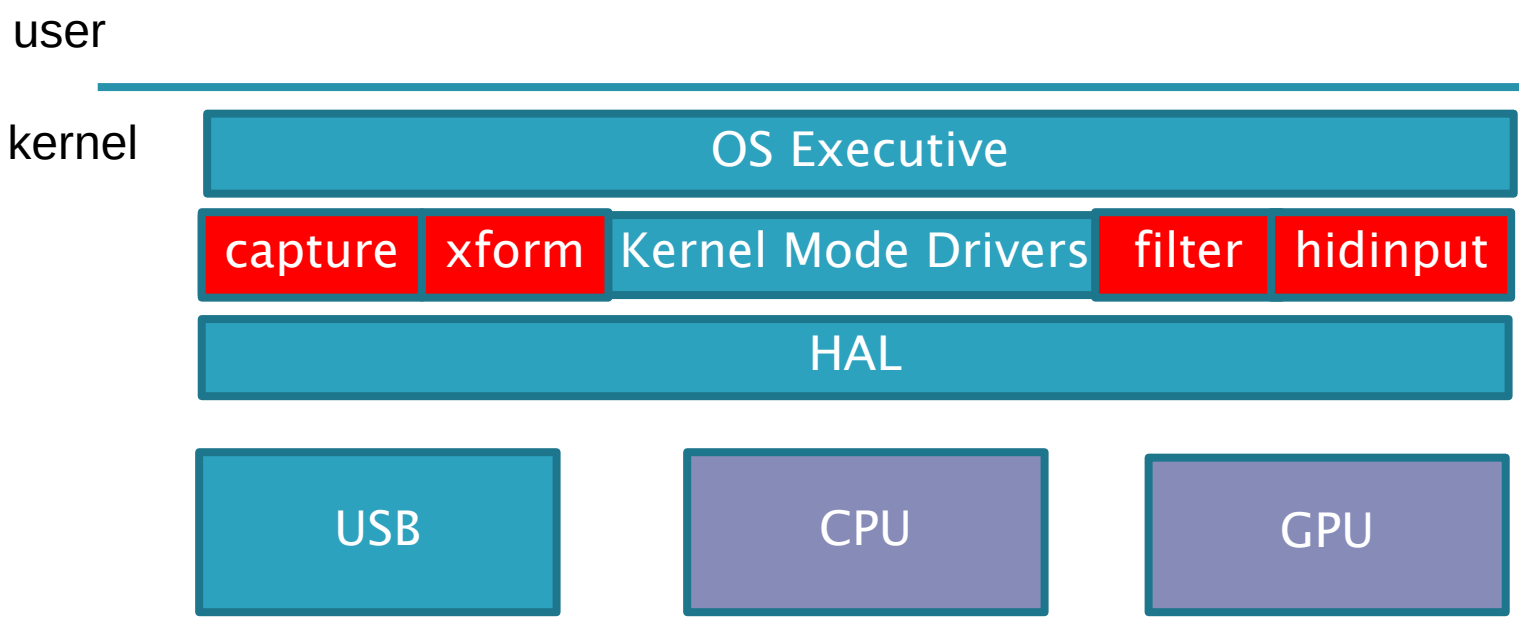
So, big deal…do it all in the kernel



So, big deal…do it all in the kernel



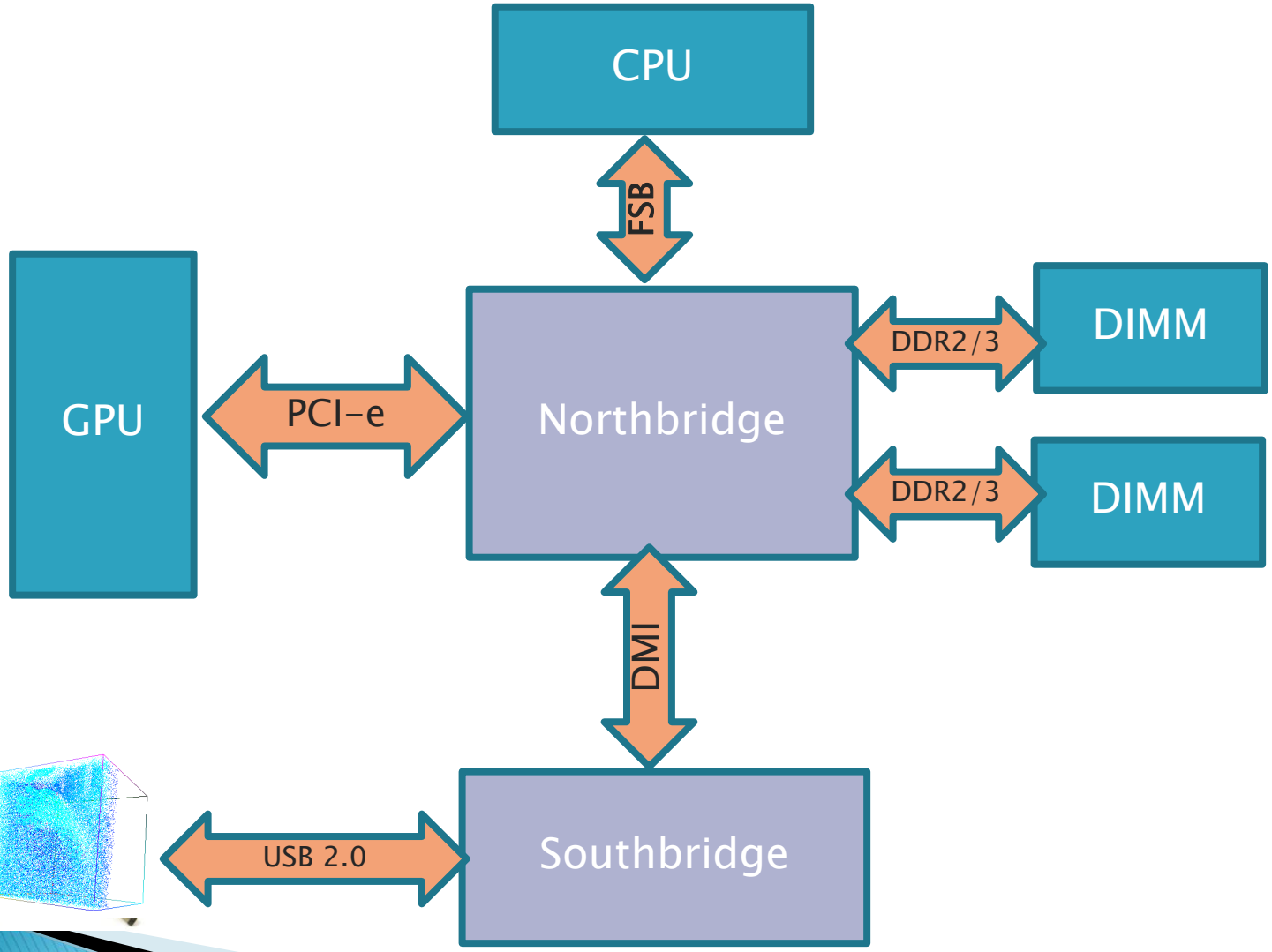
So, big deal…do it all in the kernel



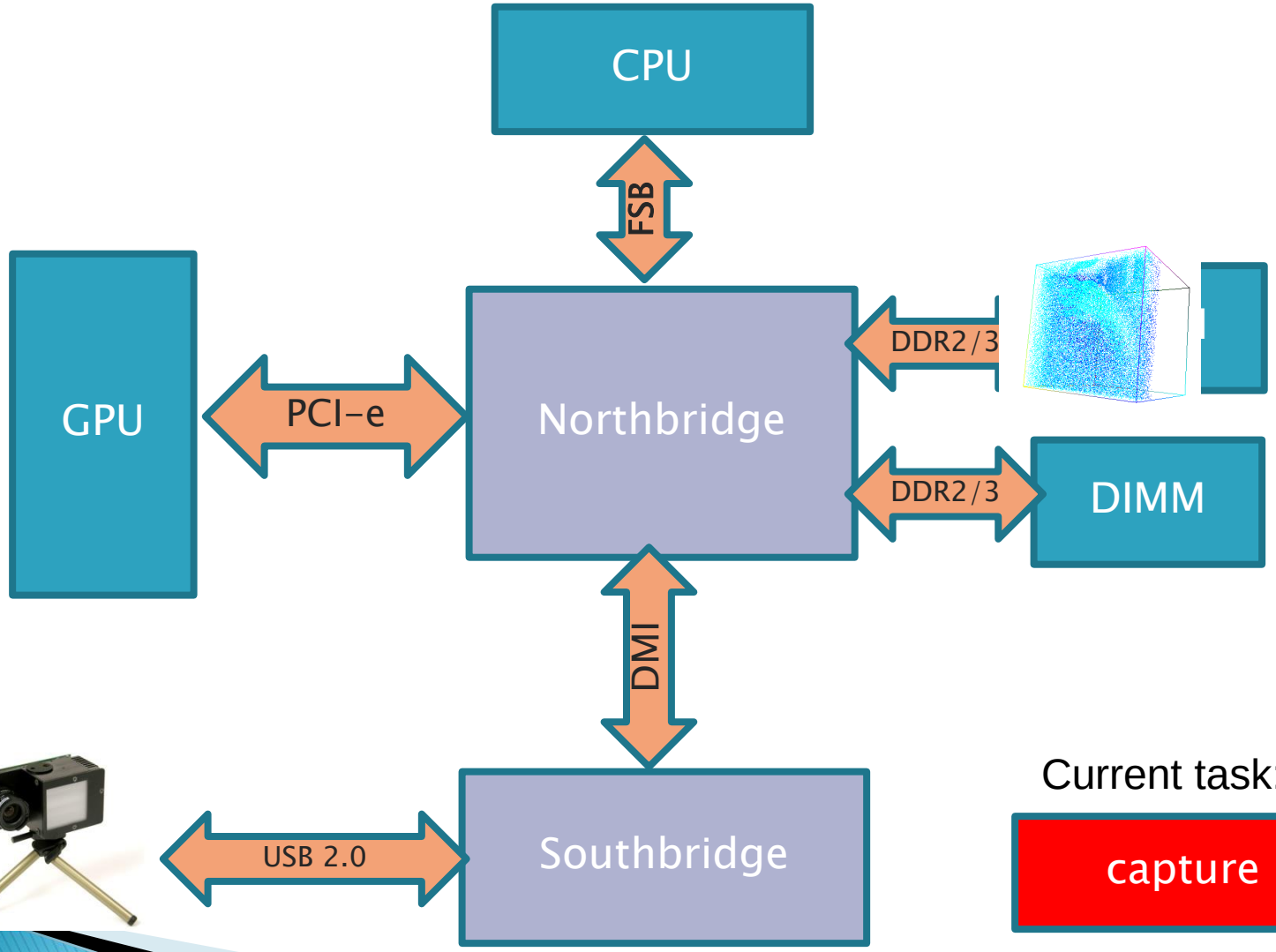
- *This is possible **if** you're MS and/or AMD,*
  - But that doesn't make it a good idea…
- No high level abstractions
- Solution is specialized

hardware data migration is still problematic…

Hardware View



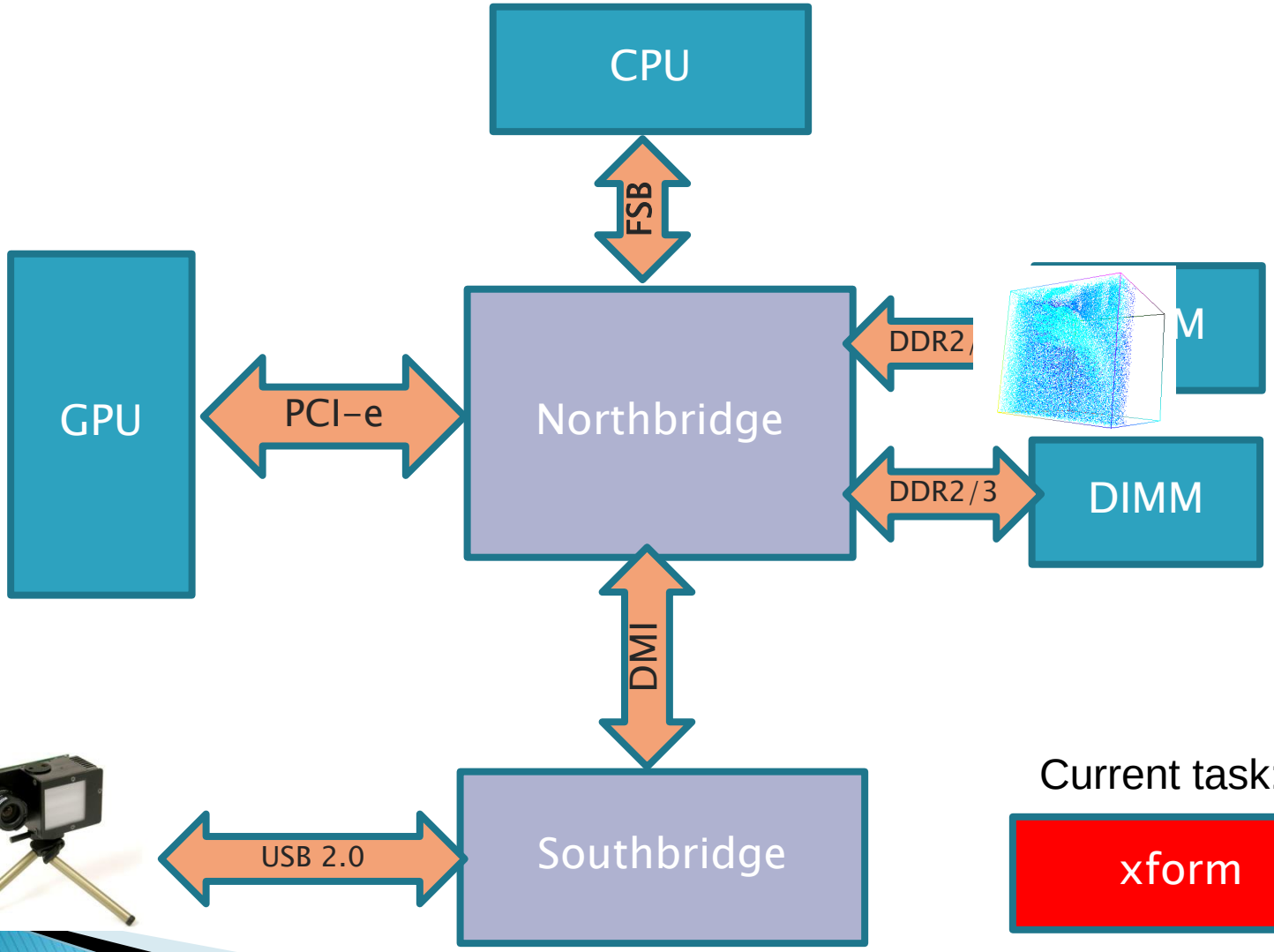
Hardware View



Current task:

capture

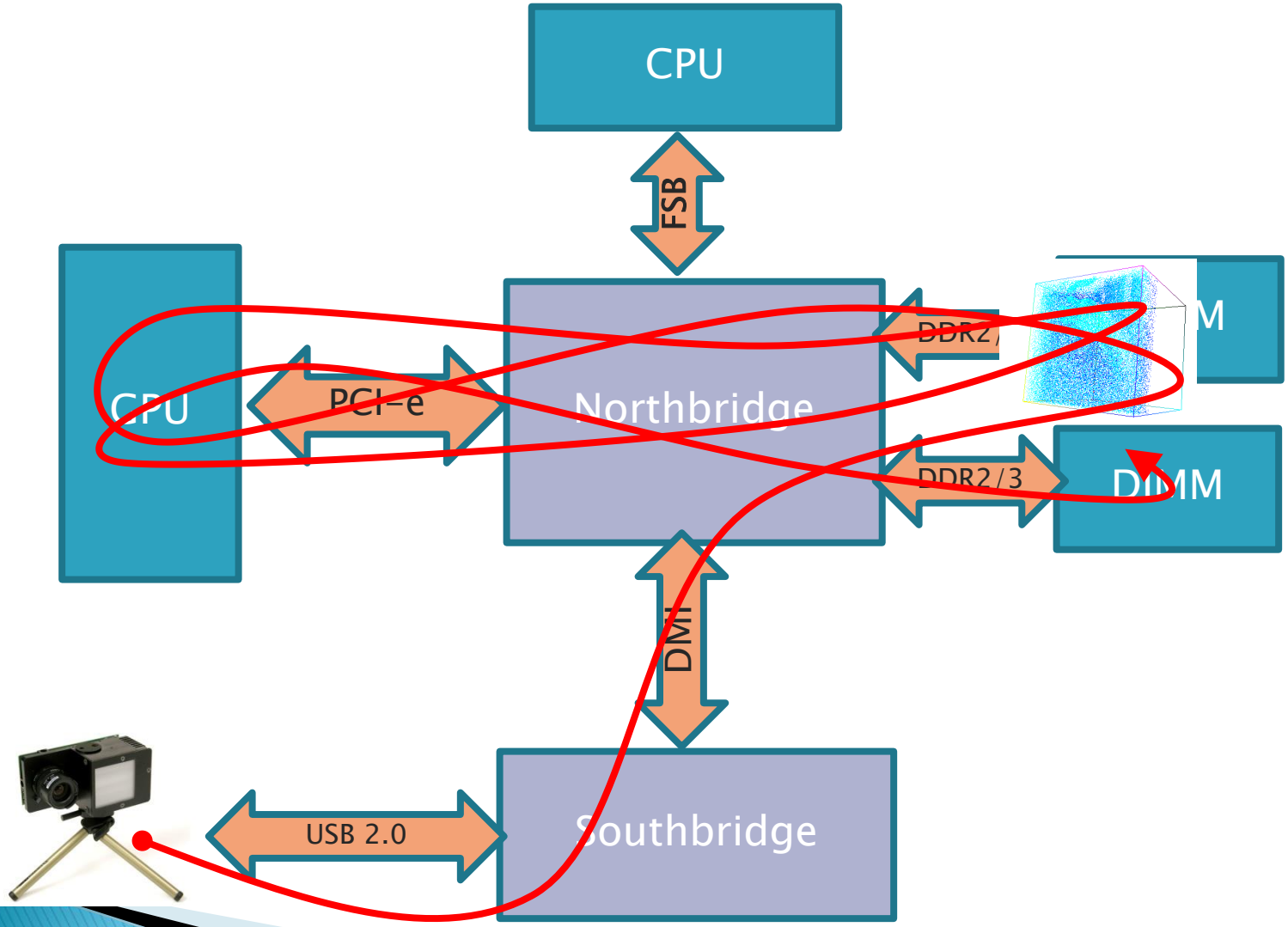
Hardware View



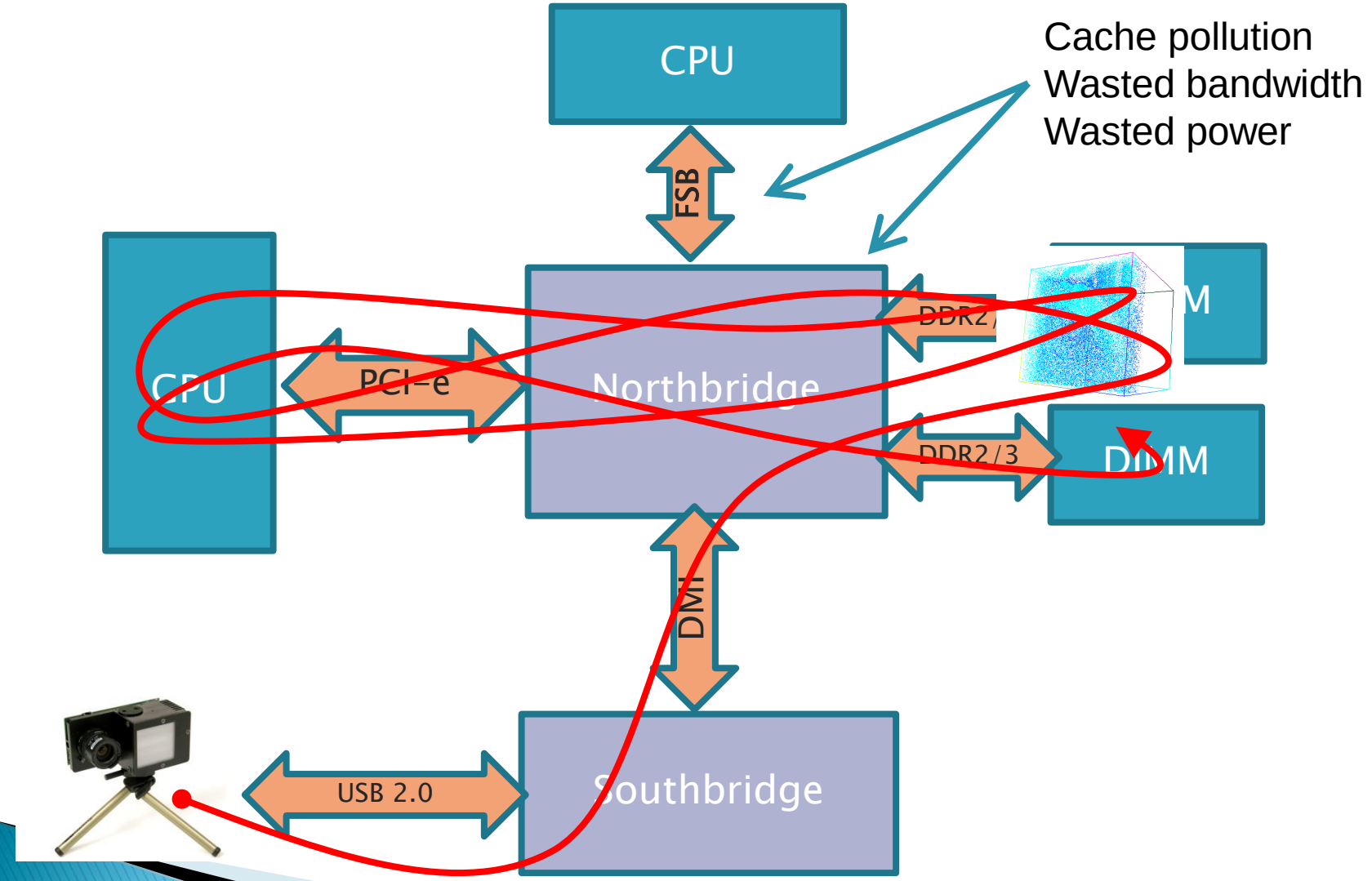
Current task:

xform

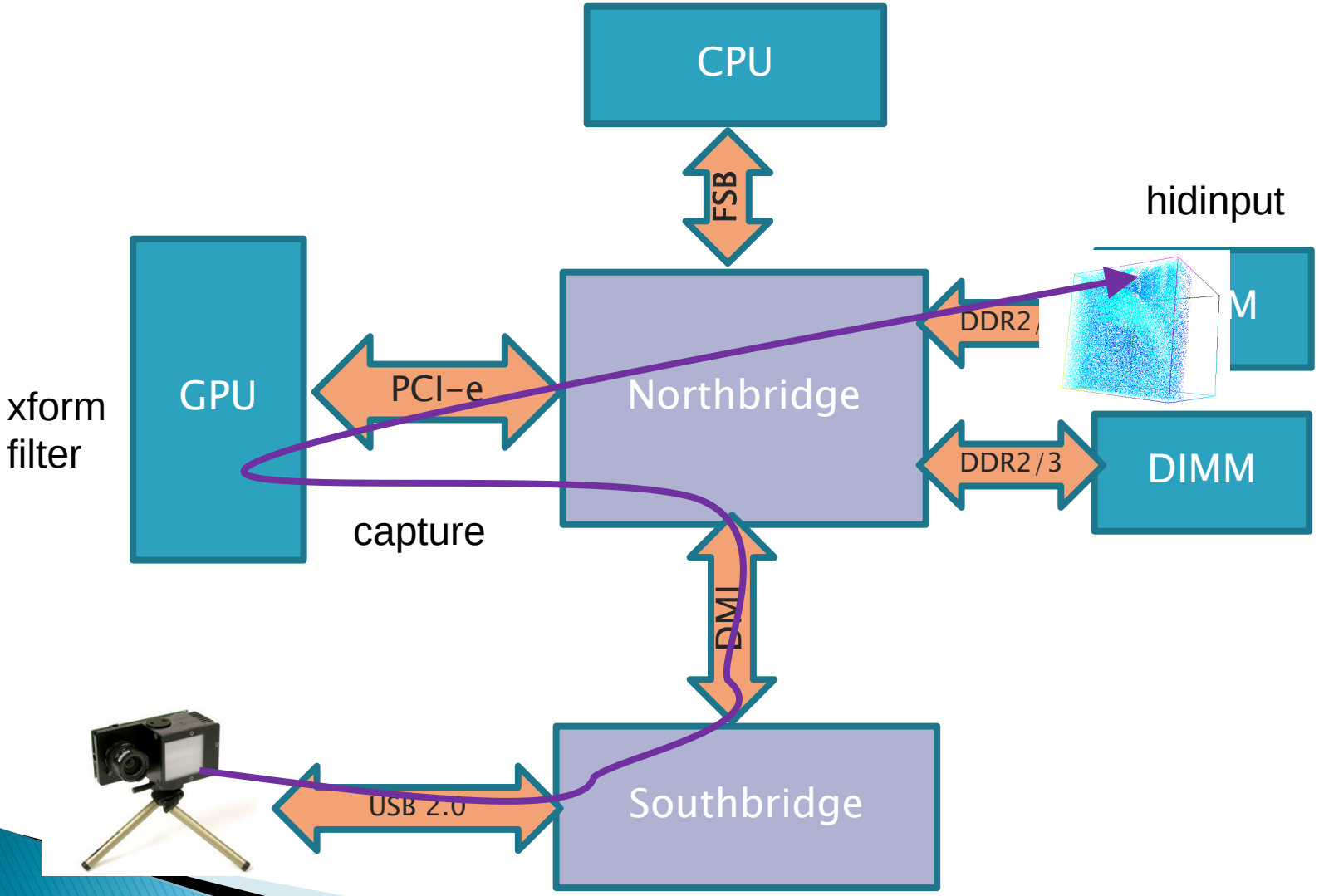
Hardware View



Hardware View



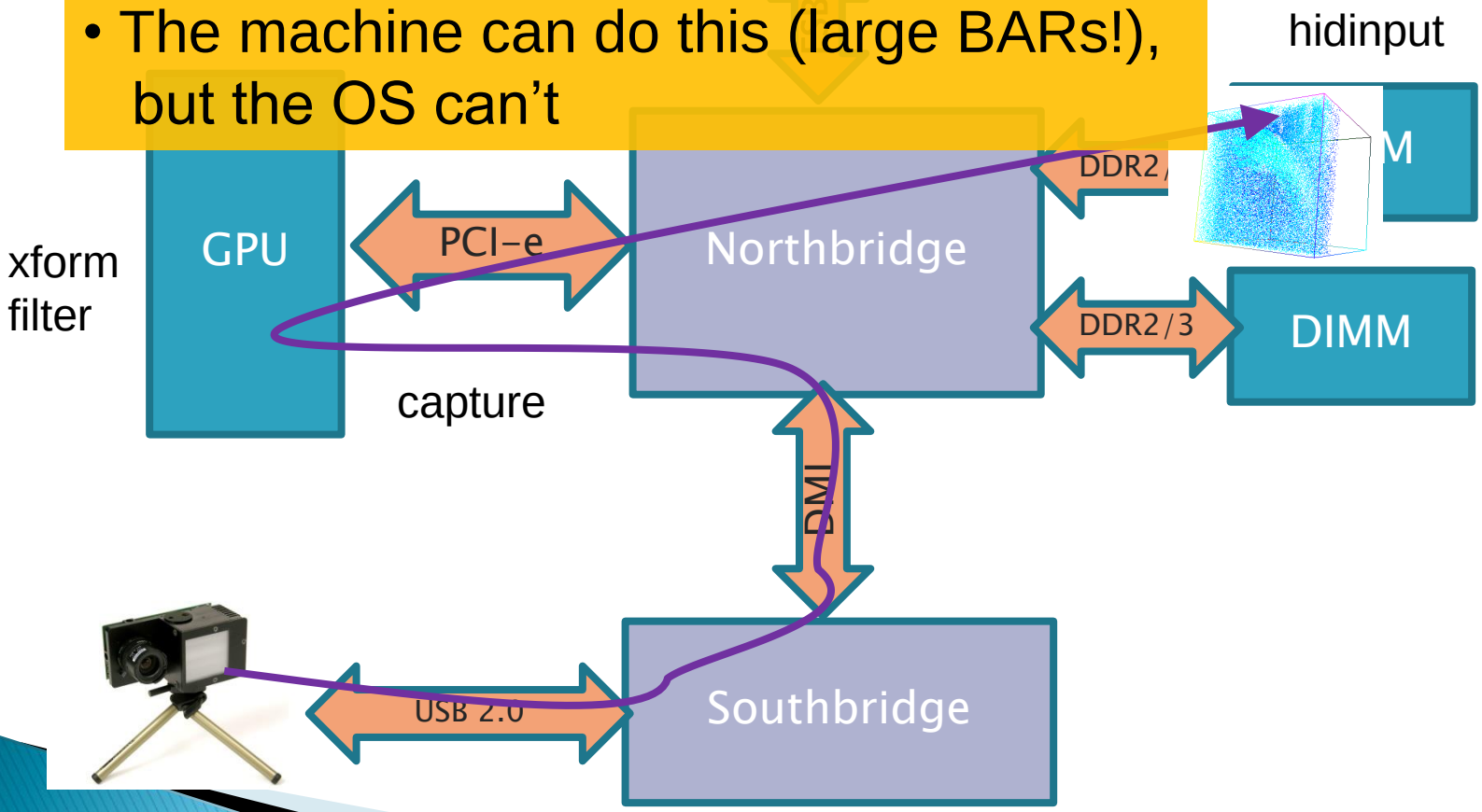
Hardware View



Hardware View

Why not:

- capture: USB bus → GPU memory
- xform, filter: **no** transfers
- The machine can do this (large BARs!), but the OS can't



Outline

- The case for OS support
- The case for dataflow abstractions
- PTask: Dataflow for GPUs
- Dandelion: LINQ on GPUs
- Related and Future Work
- Conclusion

Why dataflow?

```
Matrix
gemm(Matrix A, Matrix B) {
    copyToGPU(A);
    copyToGPU(B);
    invokeGPU();
    Matrix C = new Matrix();
    copyFromGPU(C);
    return C;
}
```

Why dataflow?

```
Matrix
gemm(Matrix A, Matrix B) {
    copyToGPU(A);
    copyToGPU(B);
    invokeGPU();
    Matrix C = new Matrix();
    copyFromGPU(C);
    return C;
}
```

What happens if I want the following?

Matrix D = A x B x C

Composed matrix multiplication

```
Matrix
AxBxC(Matrix A, B, C) {
    Matrix AxB = gemm(A,B);
    Matrix AxBxC = gemm(AxB,C);
    return AxBxC;
}
```

Composed matrix multiplication

```
Matrix
AxBxC(Matrix A, B, C) {
    Matrix AxB = gemm(A,B);
    Matrix AxBxC = gemm(AxB,C);
    return AxBxC;
}
```

```
Matrix
gemm(Matrix A, Matrix B) {
    copyToGPU(A);
    copyToGPU(B);
    invokeGPU();
    Matrix C = new Matrix();
    copyFromGPU(C);
    return C;
}
```

Composed matrix multiplication

AxB copied from GPU memory…

```
Matrix
AxBxC(Matrix A, B, C) {
    Matrix AxB = gemm(A,B);
    Matrix AxBxC = gemm(AxB,C);
    return AxBxC;
}
```

```
Matrix
gemm(Matrix A, Matrix B) {
    copyToGPU(A);
    copyToGPU(B);
    invokeGPU();
    Matrix C = new Matrix();
    copyFromGPU(C);
    return C;
}
```

Composed matrix multiplication

```
Matrix
AxBxC(Matrix A, B, C) {
    Matrix AxB = gemm(A,B);
    Matrix AxBxC = gemm(AxB,C);
    return AxBxC;
}
```

```
Matrix
gemm(Matrix A, Matrix B) {
    copyToGPU(A);
    copyToGPU(B);
    invokeGPU();
    Matrix C = new Matrix();
    copyFromGPU(C);
    return C;
}
```

…only to be copied right back!

Composed matrix multiplication

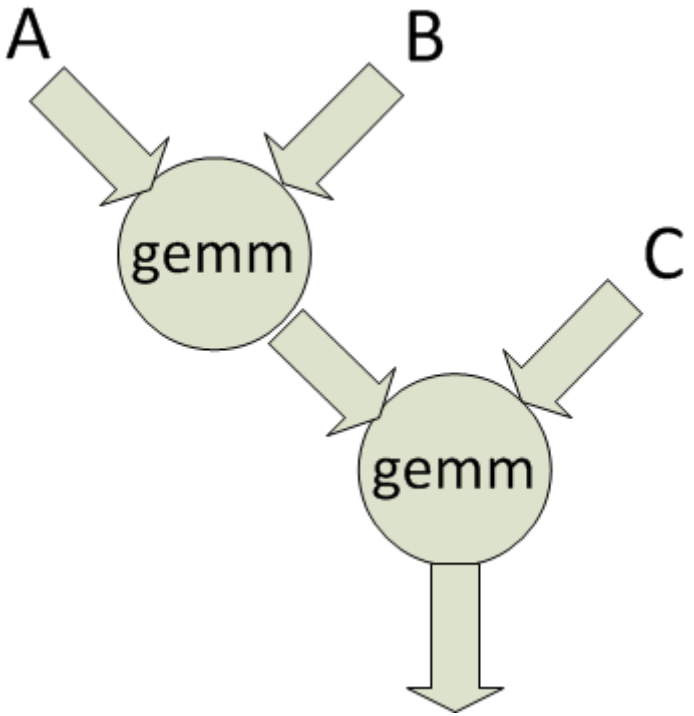
```
Matrix
AxBxC(Matrix A, B, C) {
    Matrix AxB = gemm(A,B);
    Matrix AxBxC = gemm(AxB,C);
    return AxBxC;
}
```

```
Matrix
gemm(Matrix A, Matrix B) {
    copyToGPU(A);
    copyToGPU(B);
    invokeGPU();
    Matrix C = new Matrix();
    copyFromGPU(C);
    return C;
}
```

…only to be copied right back!

We need abstractions that help get around this…

Decoupling data movement



- leaves flexibility for the runtime
- minimal specification of data movement
- runtime can be asynchronous

Outline

- The case for OS support
- The case for dataflow abstractions
- PTask: Dataflow for GPUs
- Dandelion: LINQ on GPUs
- Related and Future Work
- Conclusion

PTask OS abstractions: dataflow!

- **ptask** (parallel task)
  - Has ***priority*** for fairness
  - Analogous to a process for GPU execution
  - List of input/output resources (*e.g. stdin, stdout…*)
- **ports**
  - Can be mapped to ptask input/outputs
  - A data source or sink
- **channels**
  - Similar to pipes, connect arbitrary ports
  - Specialize to eliminate double-buffering
- **graph**
  - Directed, connected ptasks, ports, channels
- **datablocks**
  - Memory-space transparent buffers

PTask OS abstractions: dataflow!

- **ptask** (parallel task)
  - Has ***priority*** for fairness
  - Analogous to a process for GPU execution
  - List of input/output resources (*e.g. stdin, stdout…*)
- **ports**
  - Can be mapped to ptask input/outputs
  - A data source or sink
- **channels**
  - Similar to pipes, connect arbitrary ports
  - Specialize to eliminate
- **graph**
  - Directed, connected ptasks, ports, channels
- **datablocks**
  - Memory-space transparent buffers

- data: specify *where*, not *how*
- OS objects→OS RM possible

PTask Graph: Gestural Interface

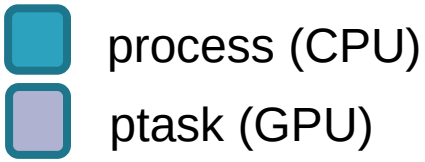
```
#> capture | xform | filter | detect &
```

PTask Graph: Gestural Interface

```
#> capture | xform | filter | detect &
```

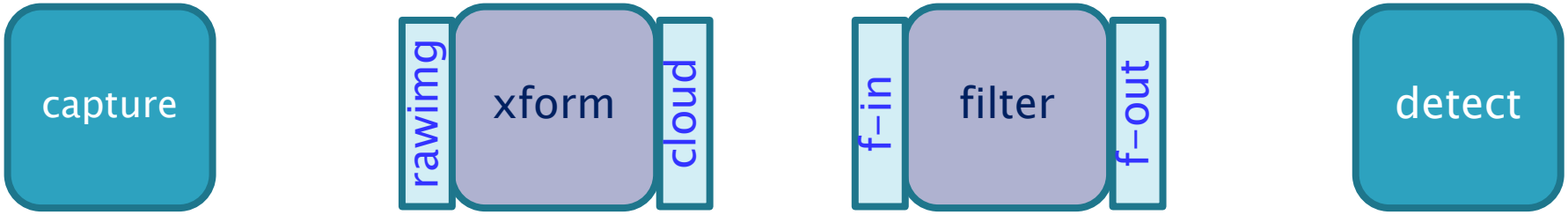
PTask Graph: Gestural Interface

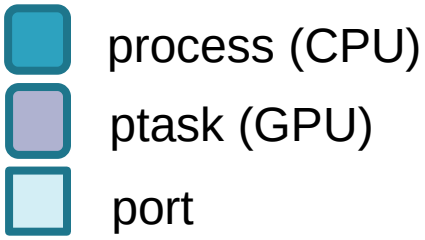
```
#> capture | xform | filter | detect &
```



PTask Graph: Gestural Interface

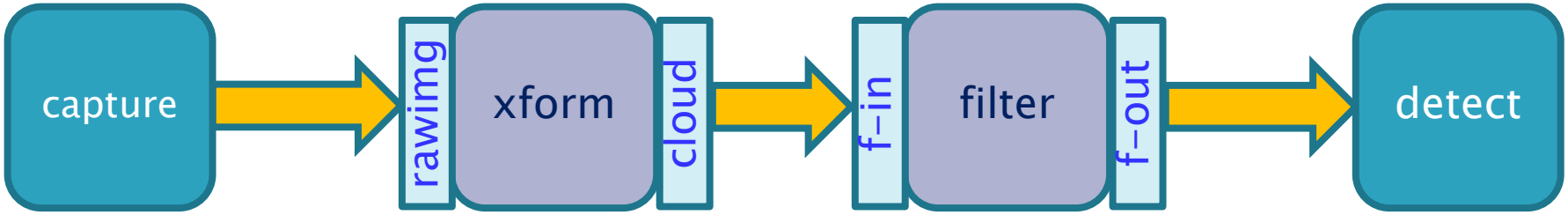
```
#> capture | xform | filter | detect &
```

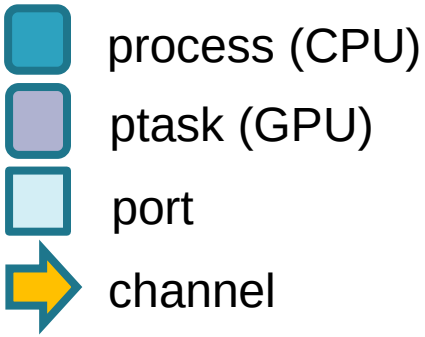




PTask Graph: Gestural Interface

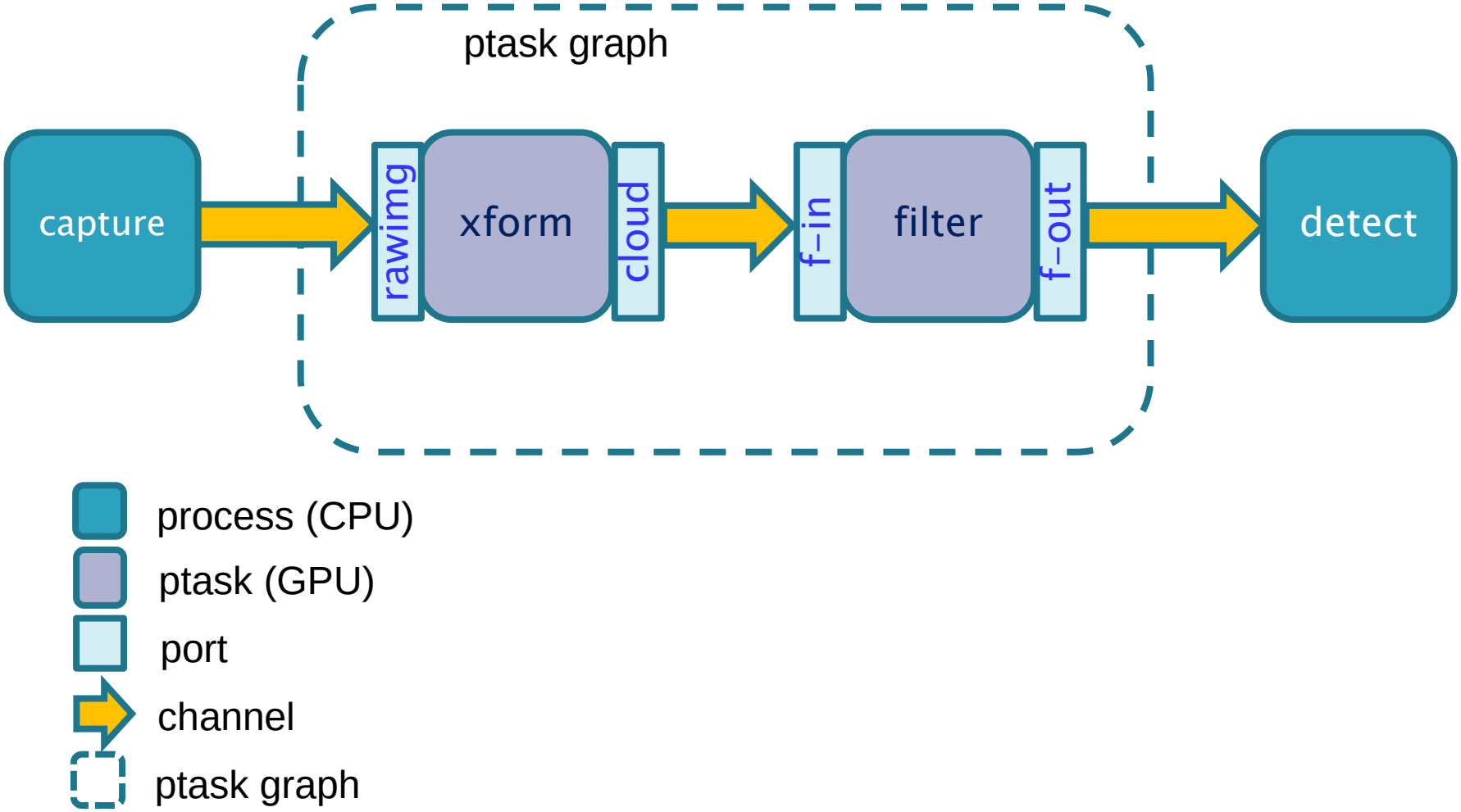
```
#> capture | xform | filter | detect &
```





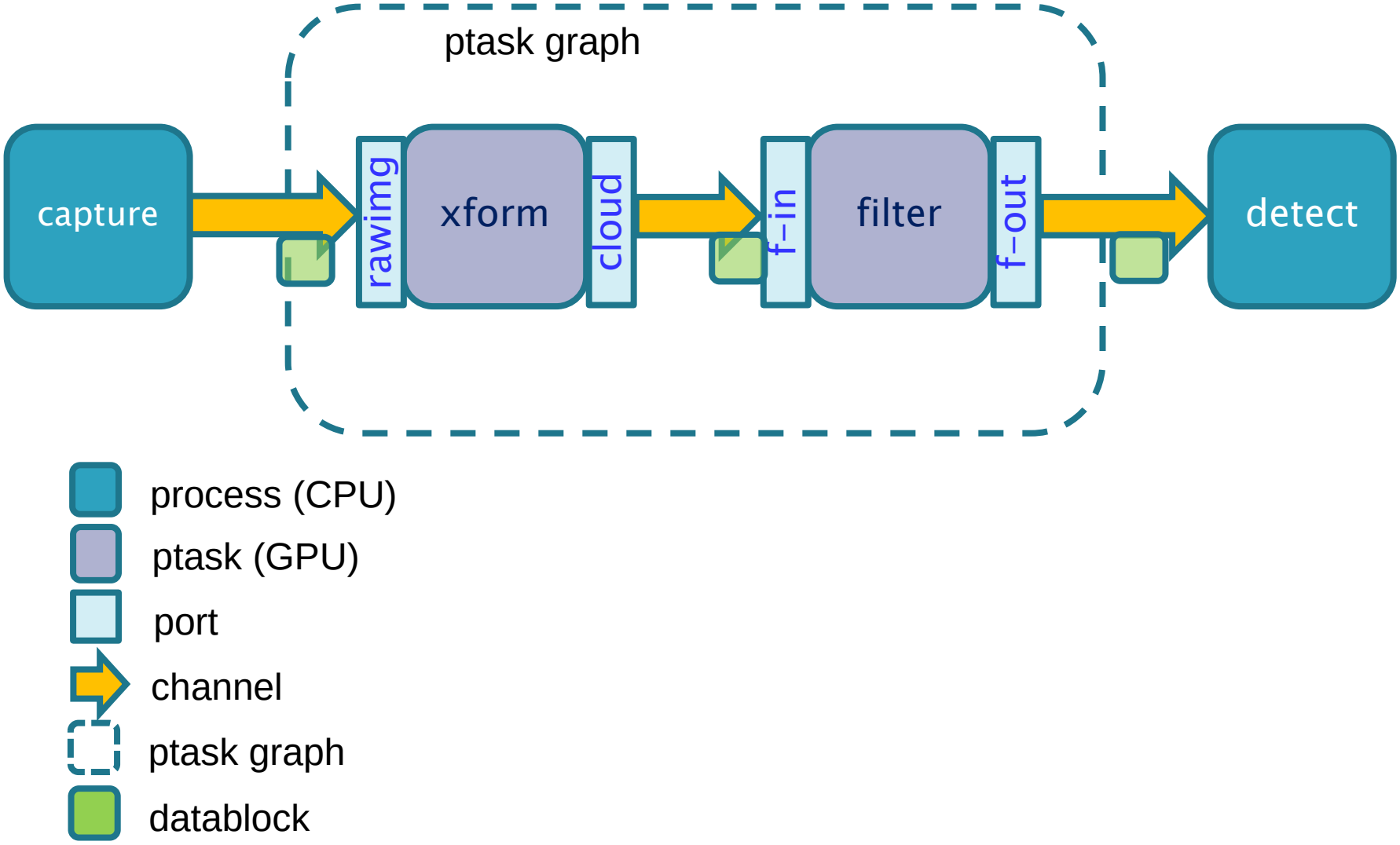
PTask Graph: Gestural Interface

```
#> capture | xform | filter | detect &
```



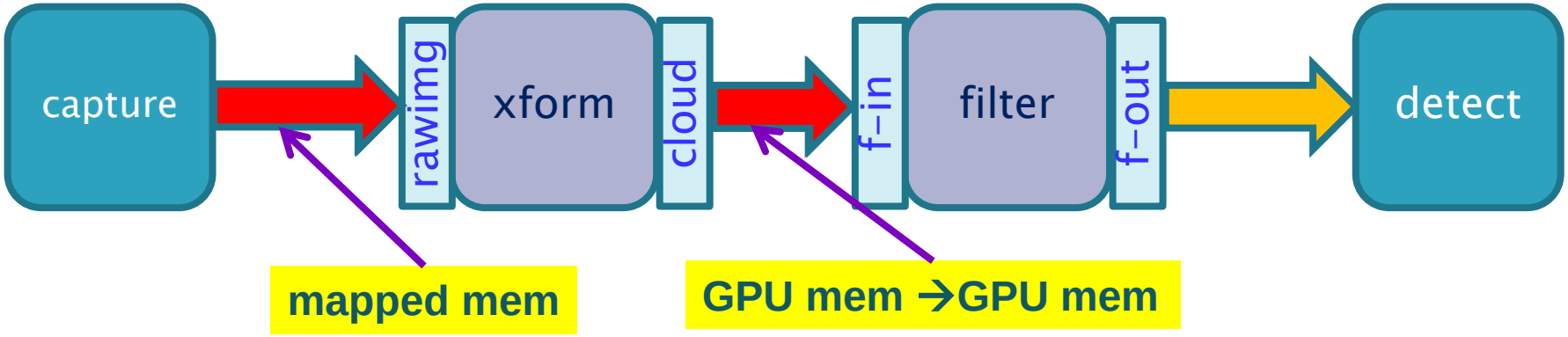
PTask Graph: Gestural Interface

```
#> capture | xform | filter | detect &
```



PTask Graph: Gestural Interface

```
#> capture | xform | filter | detect &
```



process (CPU)

ptask (GPU)

port

channel

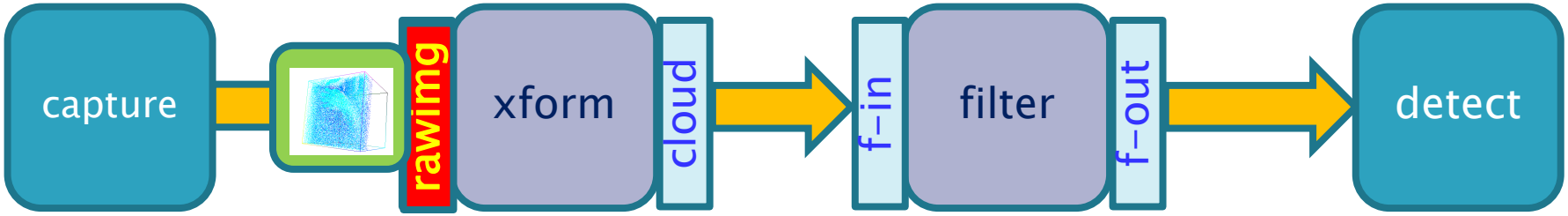
ptask graph

datablock

Optimized data movement

PTask Graph: Gestural Interface

```
#> capture | xform | filter | detect &
```



process (CPU)

ptask (GPU)

port

channel

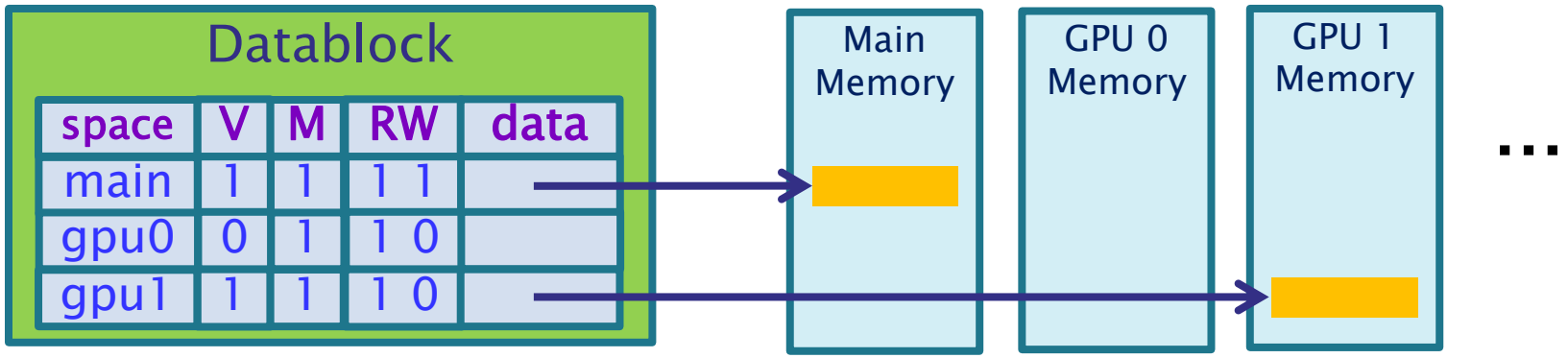
ptask graph

datablock

Optimized data movement

Data arrival triggers computation

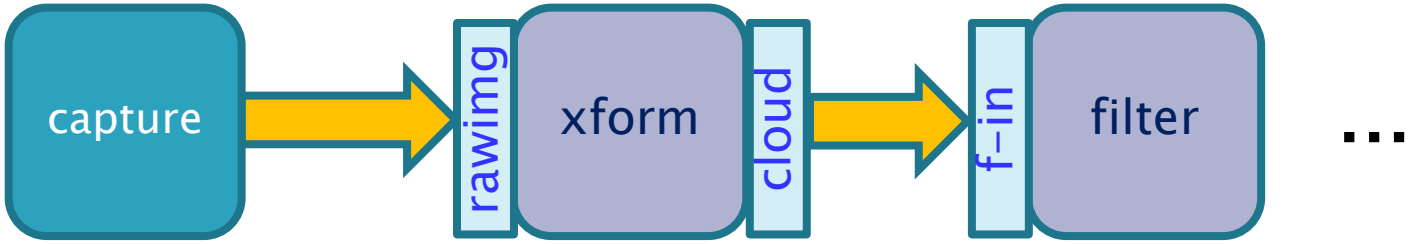
Location Transparency: Datablocks

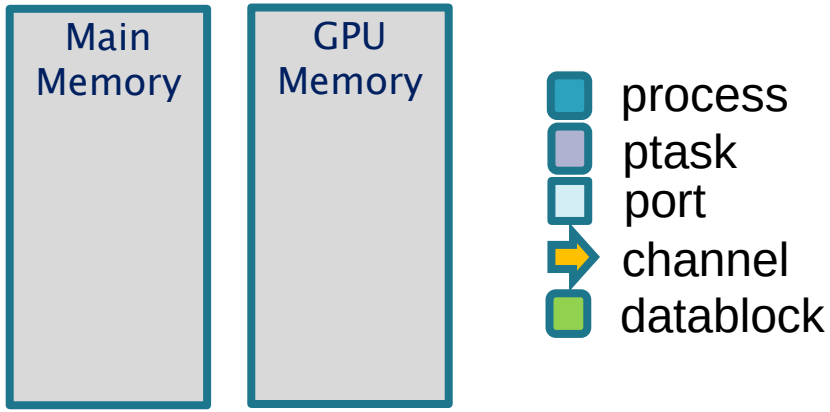


- Logical buffer
  - backed by multiple physical buffers
  - buffers created/updated lazily
  - mem-mapping used to share across process boundaries
- Track buffer validity per memory space
  - writes invalidate other views
- Flags for access control/data placement

Datablock Action Zone

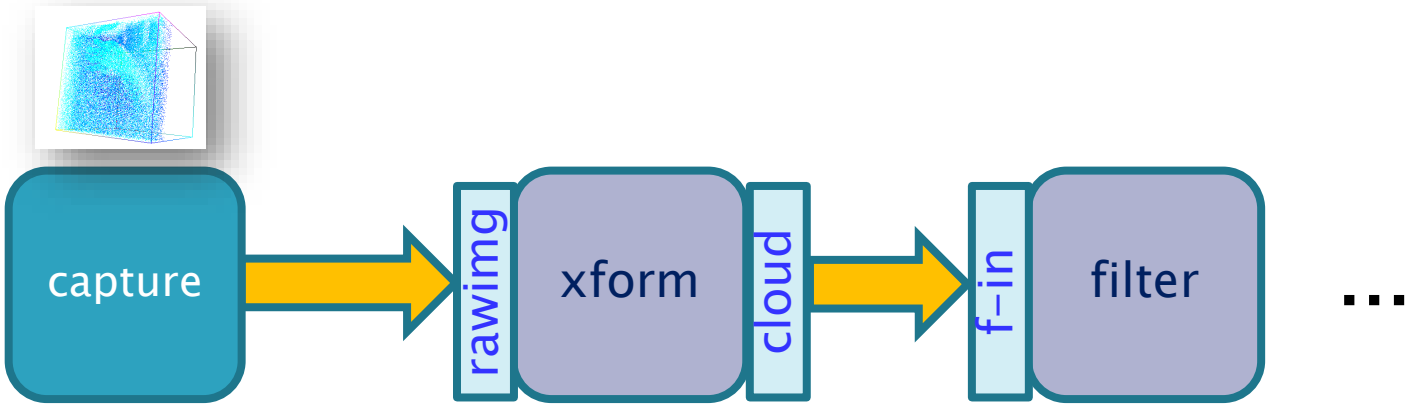
#> capture | xform | filter …

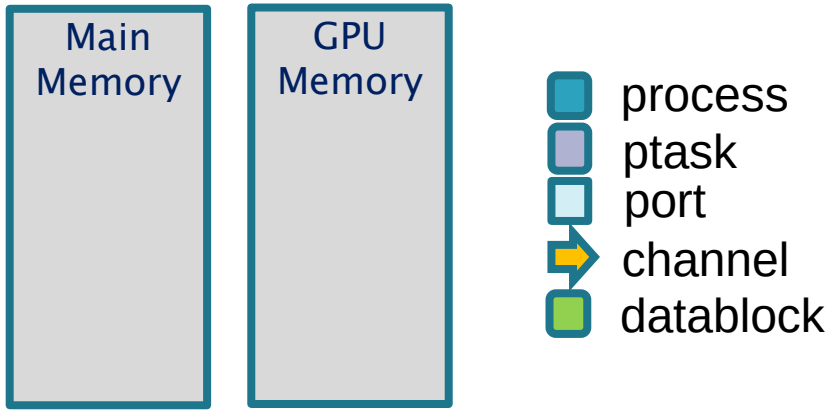




Datablock Action Zone

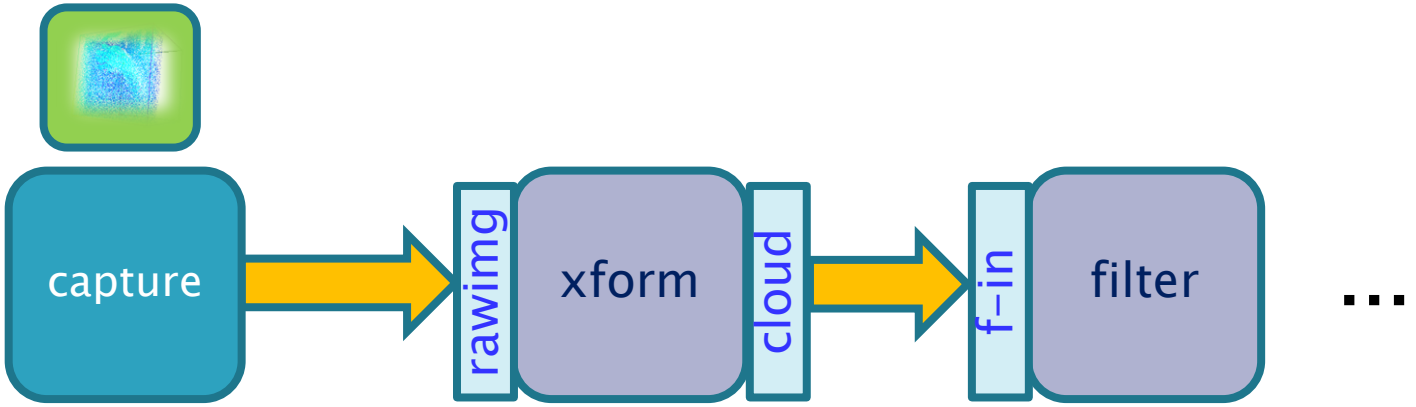
```
#> capture | xform | filter …
```

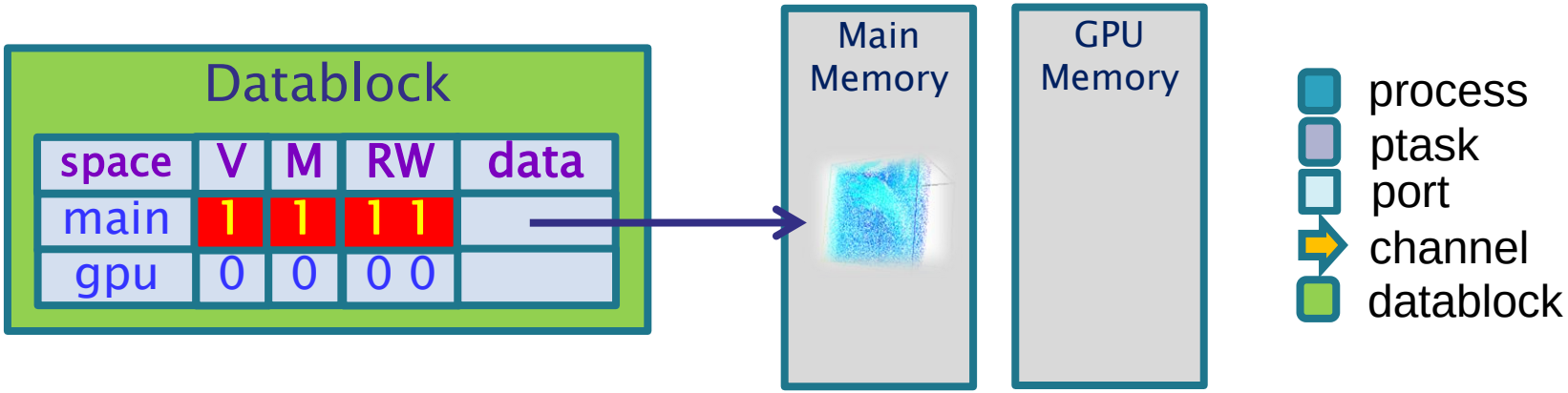




Datablock Action Zone

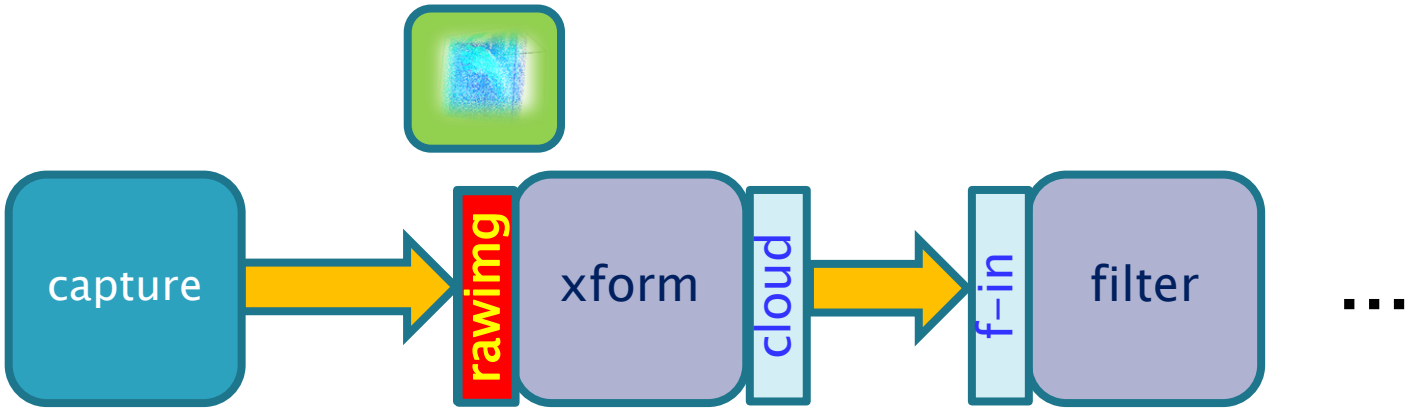
#> capture | xform | filter …

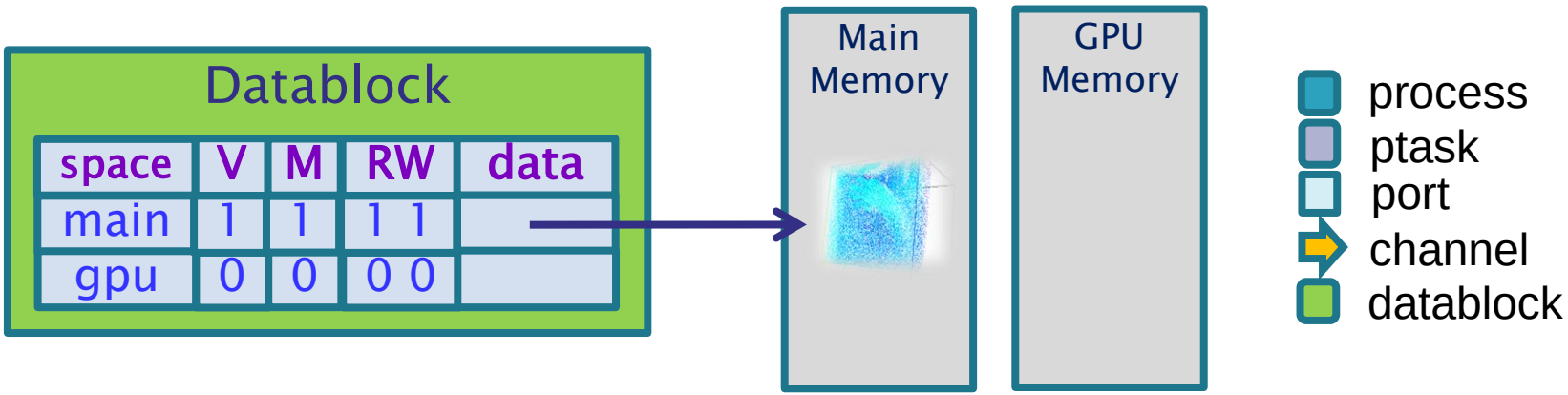




Datablock Action Zone

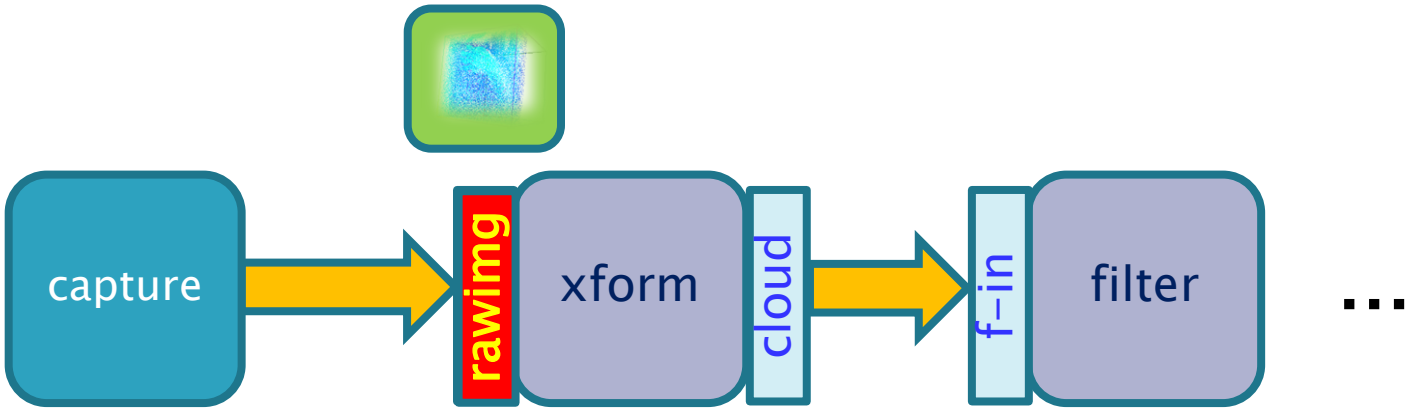
#> capture | xform | filter …

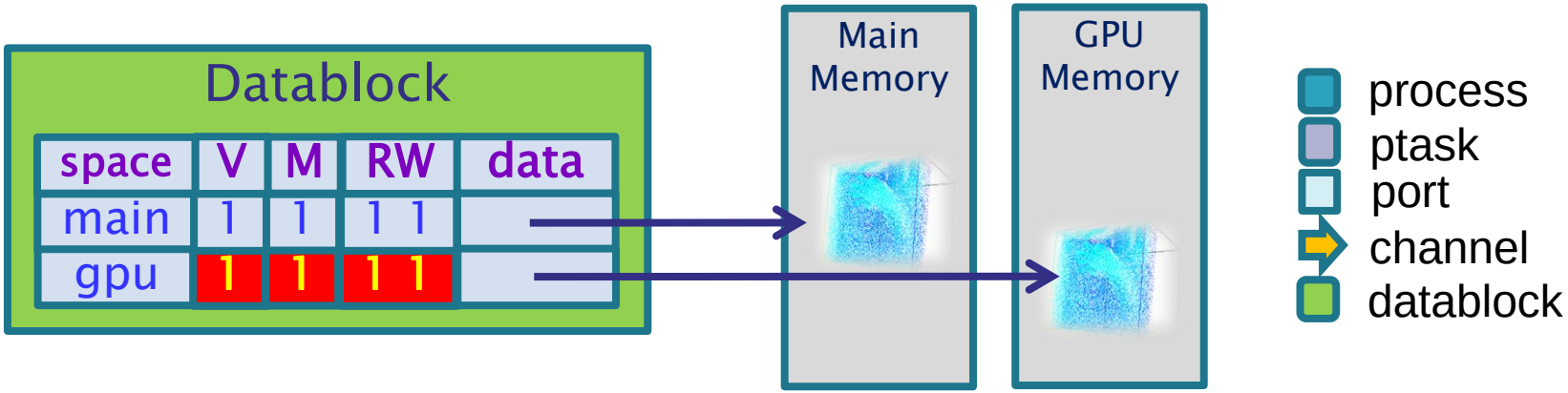




Datablock Action Zone

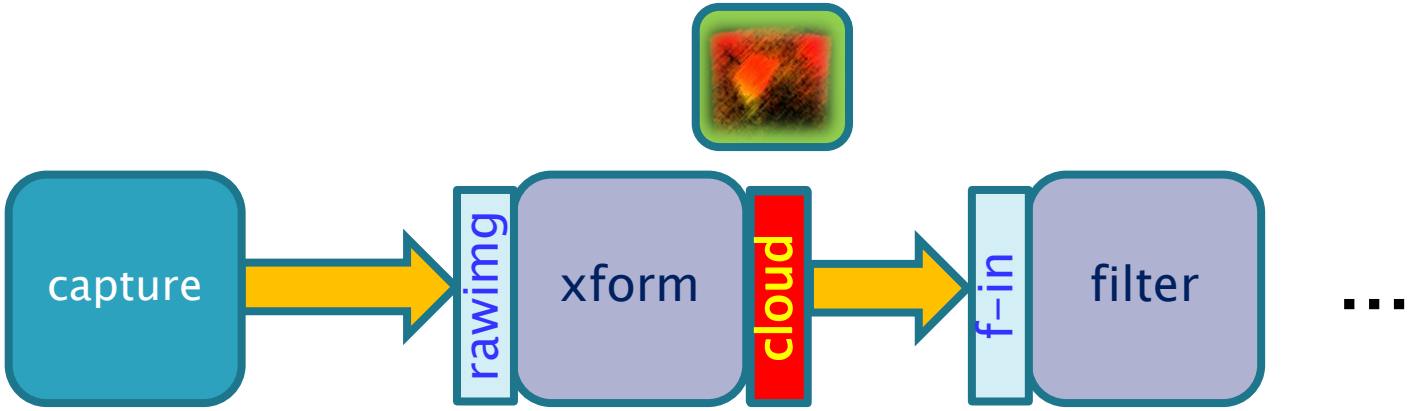
#> capture | xform | filter …

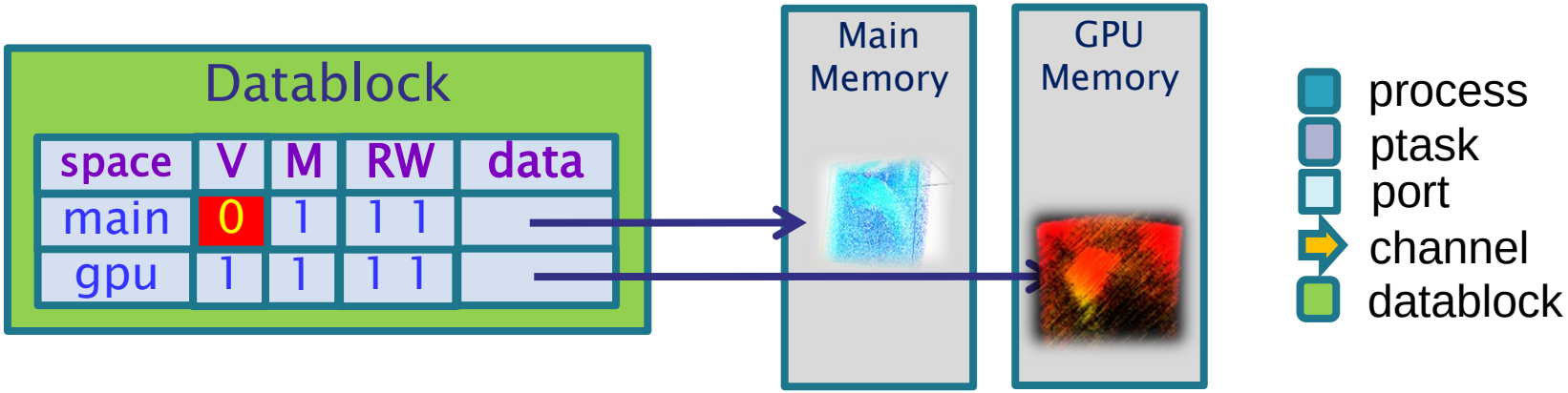




Datablock Action Zone

#> capture | xform | filter …





PTask Scheduling

- Graphs scheduled dynamically
  - ptasks queue for dispatch when inputs ready
- Queue: dynamic priority order
  - ptask priority user-settable
  - ptask prio normalized to OS prio
- Transparently support multiple GPUs
  - Schedule ptasks for input locality

Outline

- The case for OS support
- The case for dataflow abstractions
- PTask: Dataflow for GPUs
  - **Some numbers**
- Dandelion: LINQ on GPUs
- Related and Future Work
- Conclusion

Implementation

Windows 7

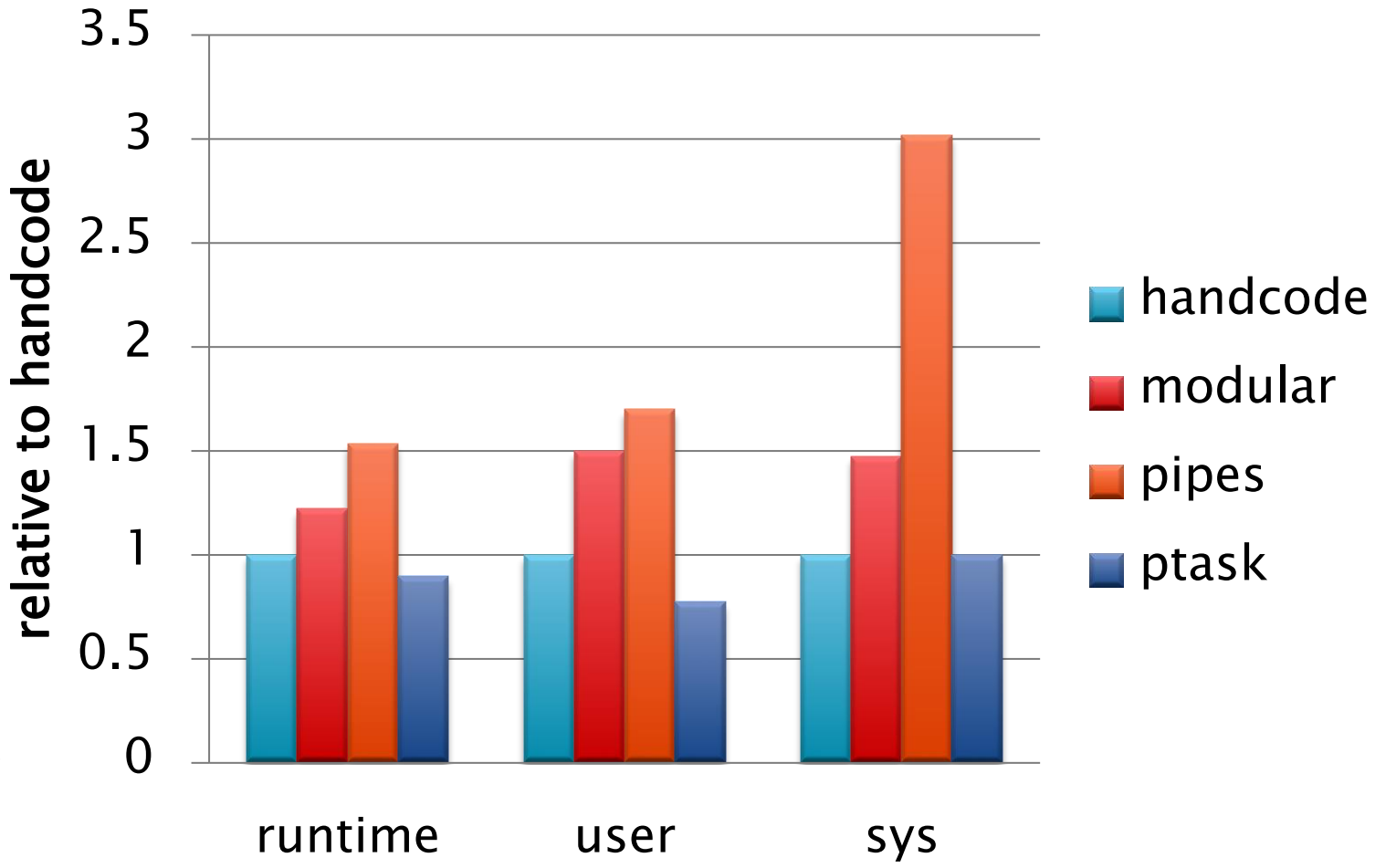
Two PTask API implementations:

1. Stacked UMDF/KMDF driver
   - Kernel component: mem-mapping, signaling
   - User component: wraps DirectX, CUDA, OpenCL
   - syscalls → DeviceIoControl() calls
2. User-mode library

Gestural Interface evaluation

- Implementations
  - **pipes**: capture | xform | filter | detect
  - **modular**: capture+xform+filter+detect, 1process
  - **handcode**: data movement optimized, 1process
  - **ptask**: ptask graph
- Configurations
  - **real-time**: driven by cameras
  - **unconstrained**: driven by in-memory playback

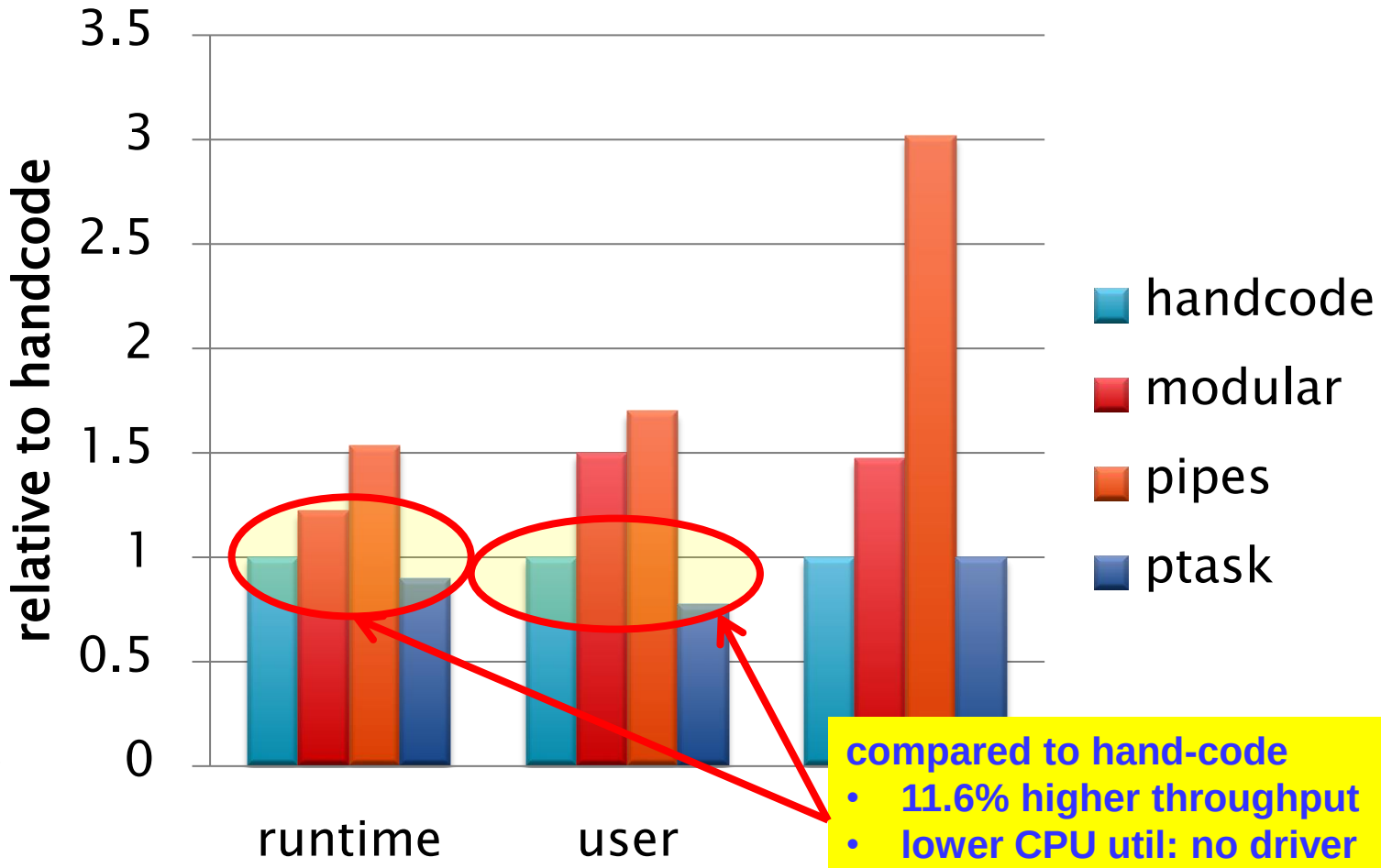
Gestural Interface Performance



lower is better

- Windows 7 x64 8GB RAM
- Intel Core 2 Quad 2.66GHz
- GTX580 (EVGA)

Gestural Interface Performance

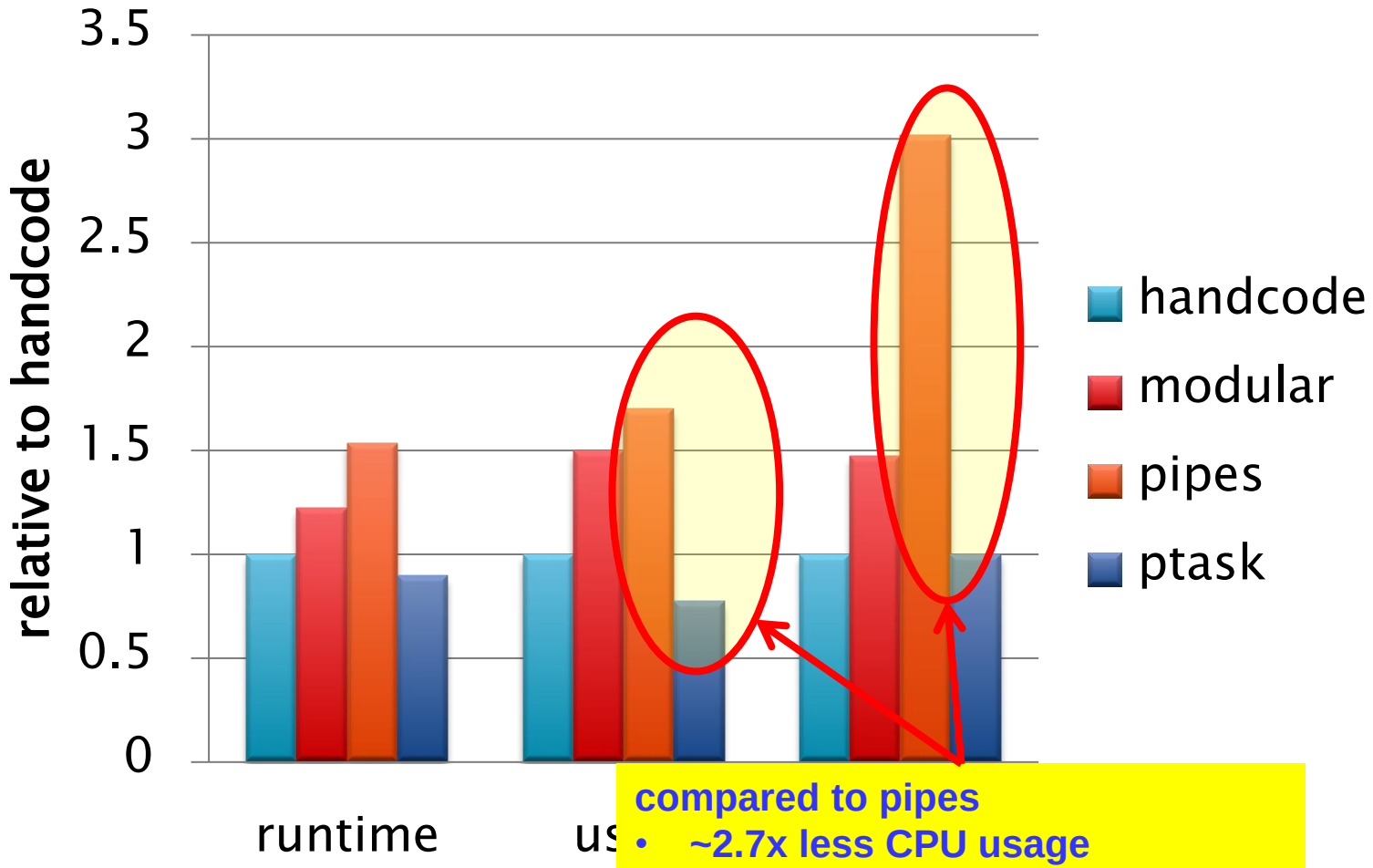


lower is better

compared to hand-code
- **11.6% higher throughput**
- **lower CPU util: no driver program**

- Intel Core 2 Quad 2.66GHz
- GTX580 (EVGA)

Gestural Interface Performance

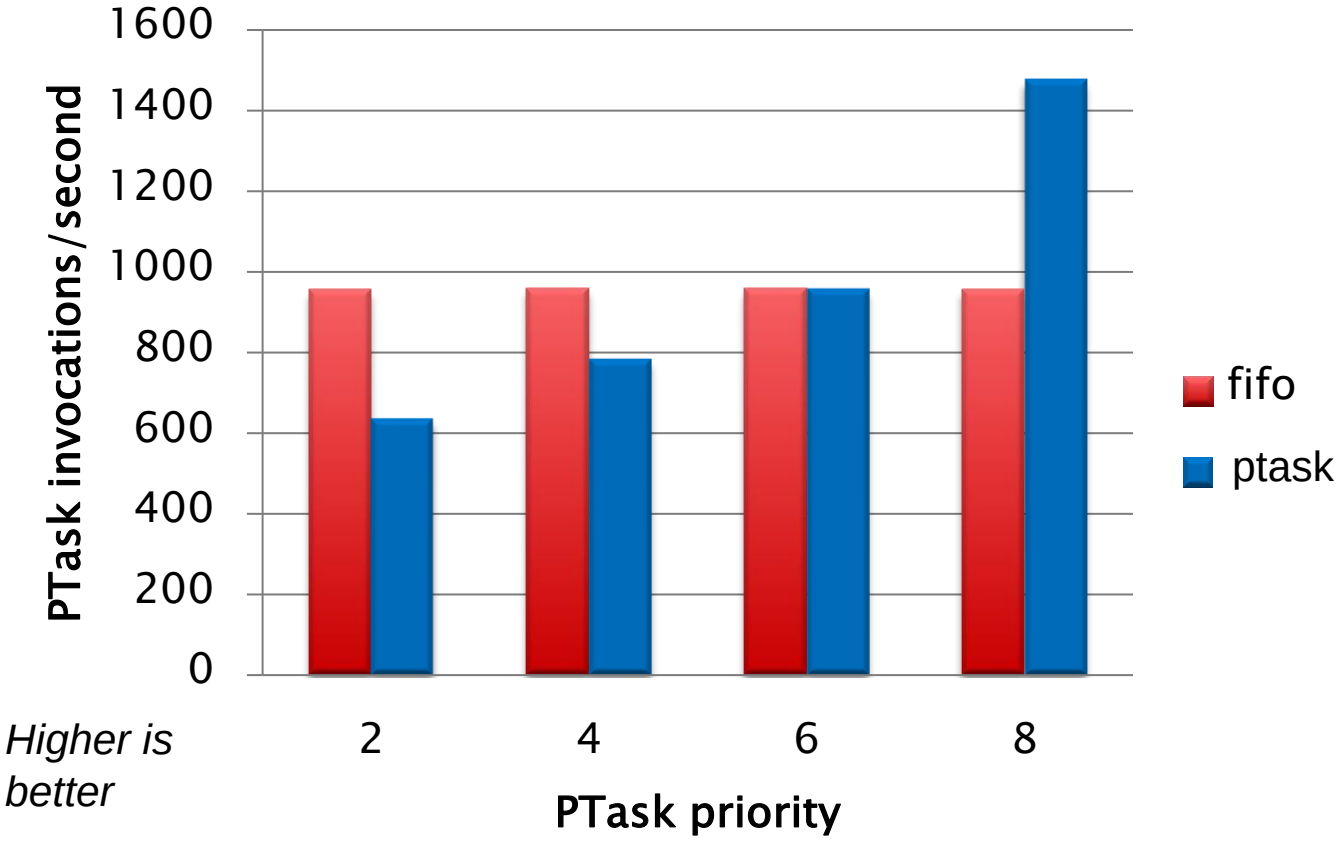


lower is better

compared to pipes
- **~2.7x less CPU usage**
- **16x higher throughput**
- **~45% less memory usage**

GB RAM
d 2.66GHz
• GTX580 (EVGA)

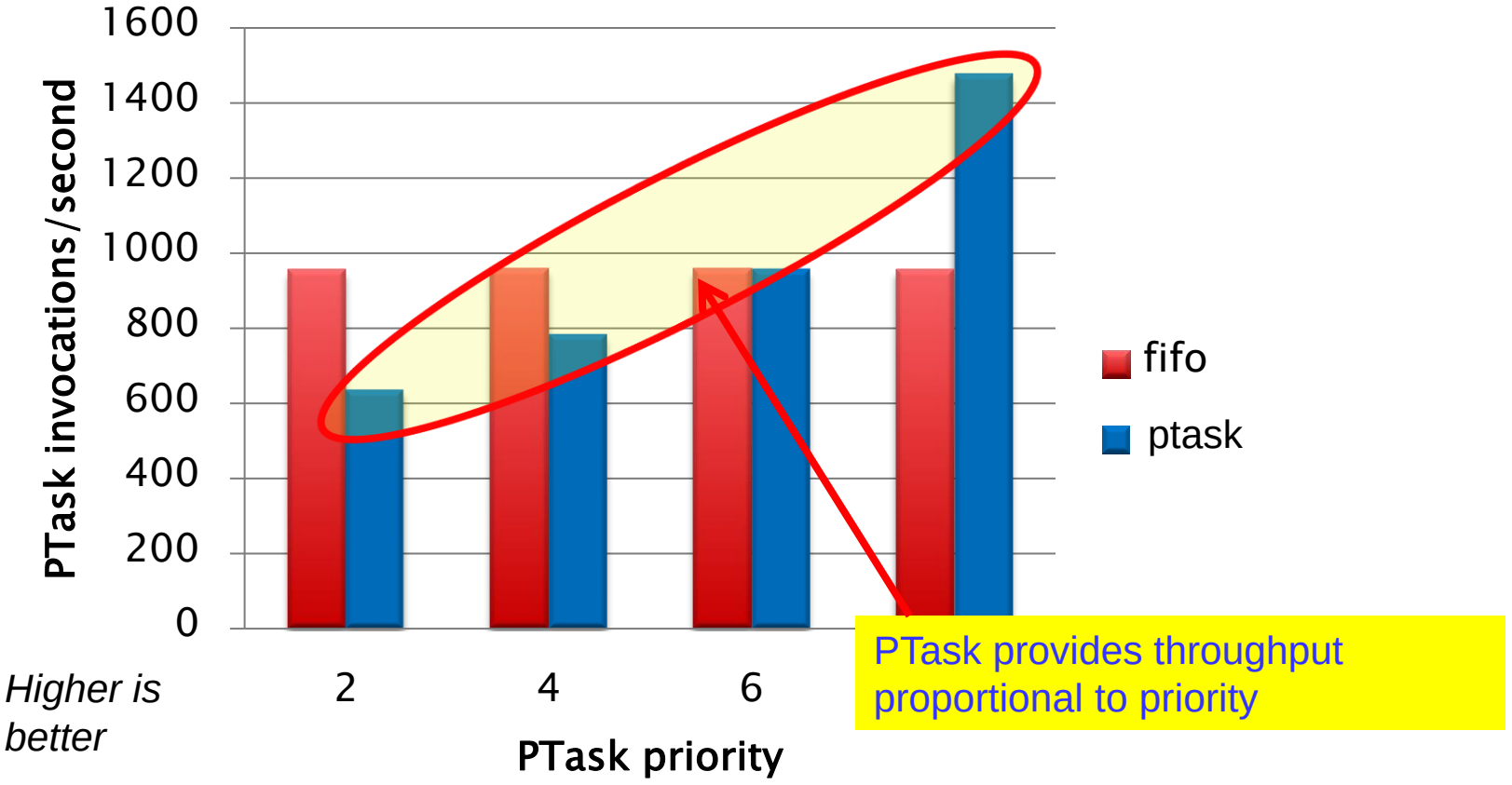
Performance Isolation



- FIFO – queue invocations in arrival order
- ptask – aged priority queue w OS priority
- graphs: 6x6 matrix multiply
- priority same for every PTask node

- Windows 7 x64 8GB RAM
- Intel Core 2 Quad 2.66GHz
- GTX580 (EVGA)

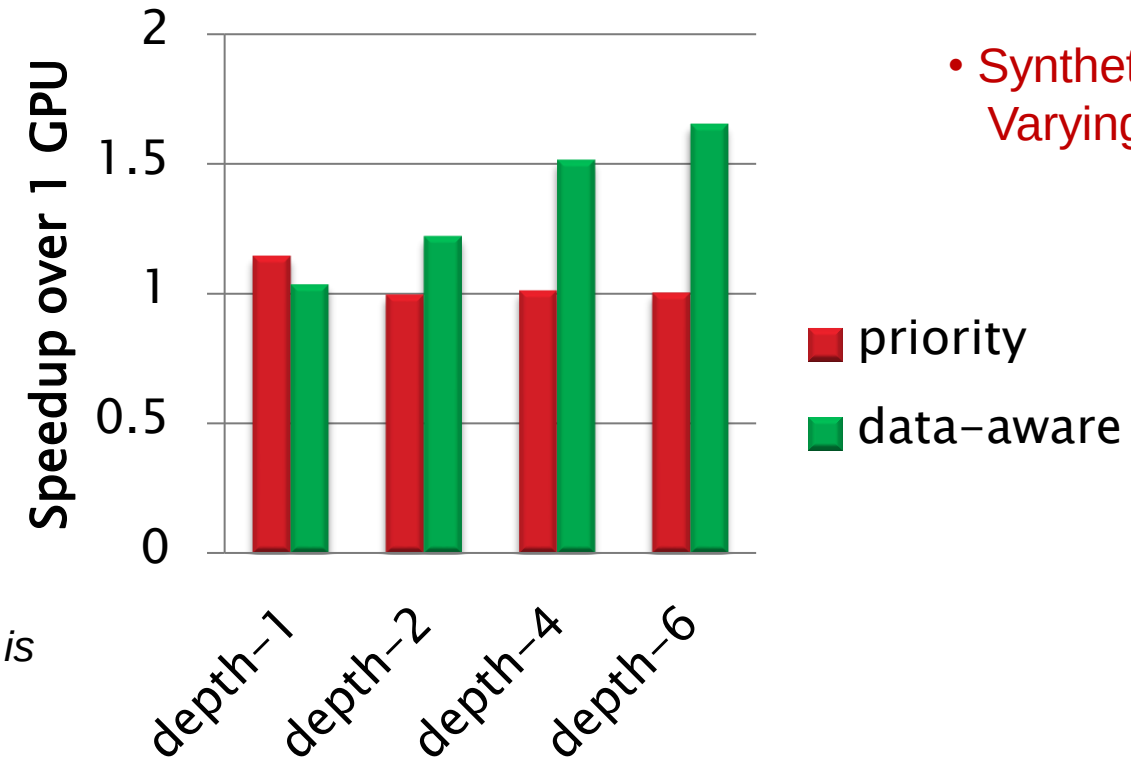
Performance Isolation



- FIFO – queue invocations in arrival order
- ptask – aged priority queue w OS priority
- graphs: 6x6 matrix multiply
- priority same for every PTask node

- Windows 7 x64 8GB RAM
- Intel Core 2 Quad 2.66GHz
- GTX580 (EVGA)

Multi-GPU Scheduling



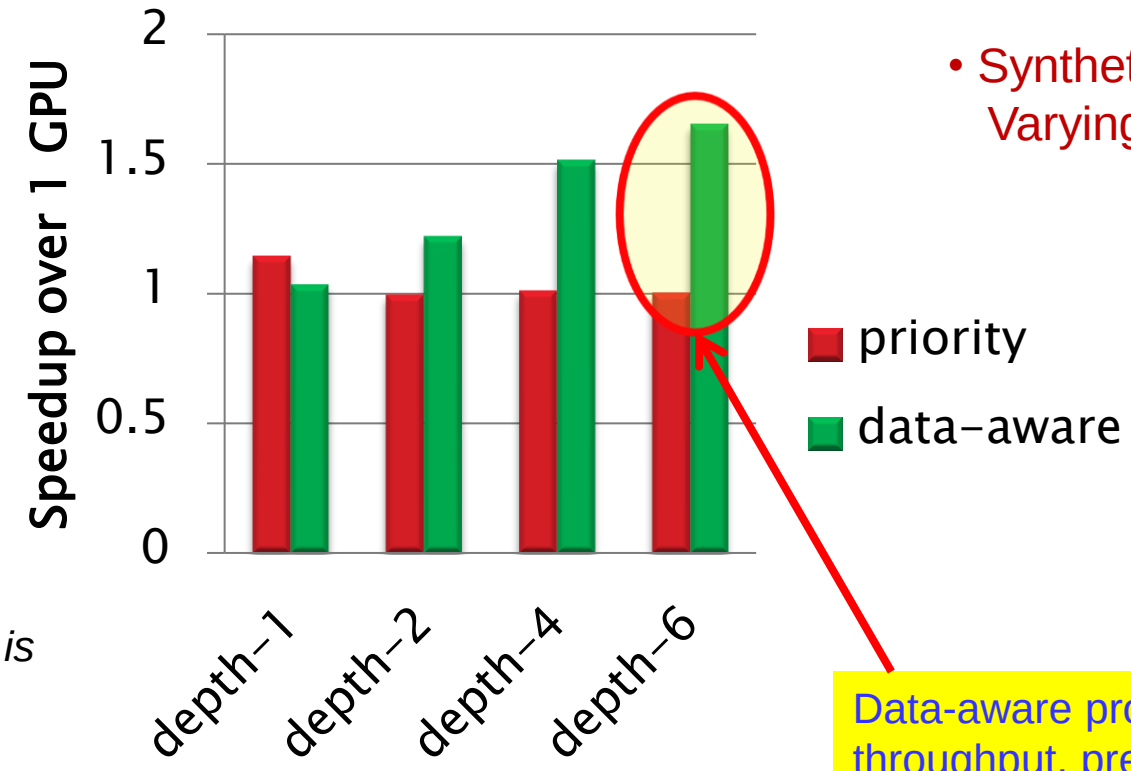
Higher is better

- Synthetic graphs: Varying depths

- Data-aware == priority + locality
- Graph depth > 1 req. for any benefit

- Windows 7 x64 8GB RAM
- Intel Core 2 Quad 2.66GHz
- 2 x GTX580 (EVGA)

Multi-GPU Scheduling



Higher is better

- Synthetic graphs: Varying depths

Data-aware provides best throughput, preserves priority

- Data-aware == priority + locality
- Graph depth > 1 req. for any benefit

- Windows 7 x64 8GB RAM
- Intel Core 2 Quad 2.66GHz
- 2 x GTX580 (EVGA)

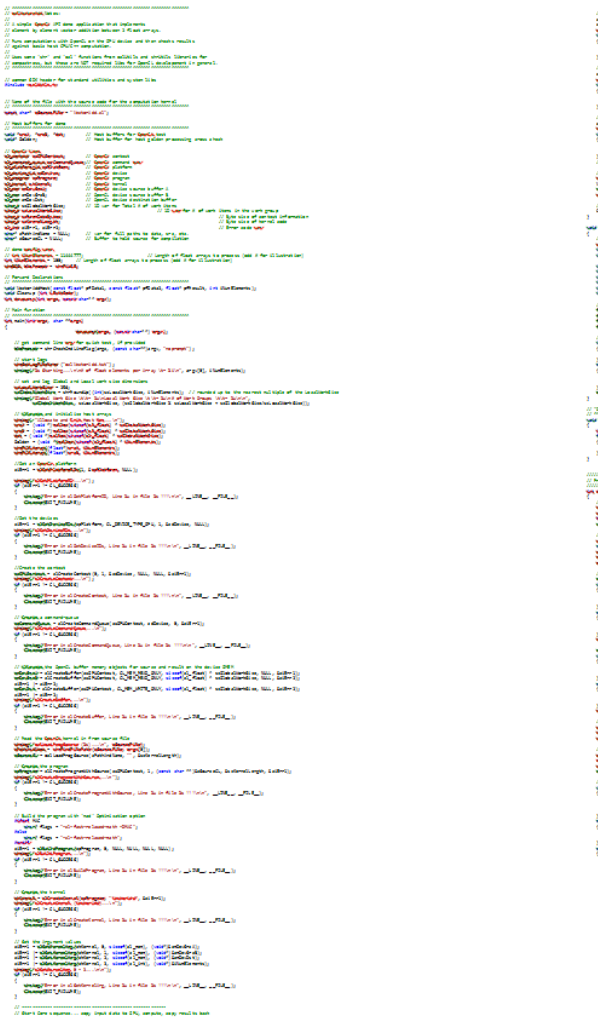
Things I didn't show you

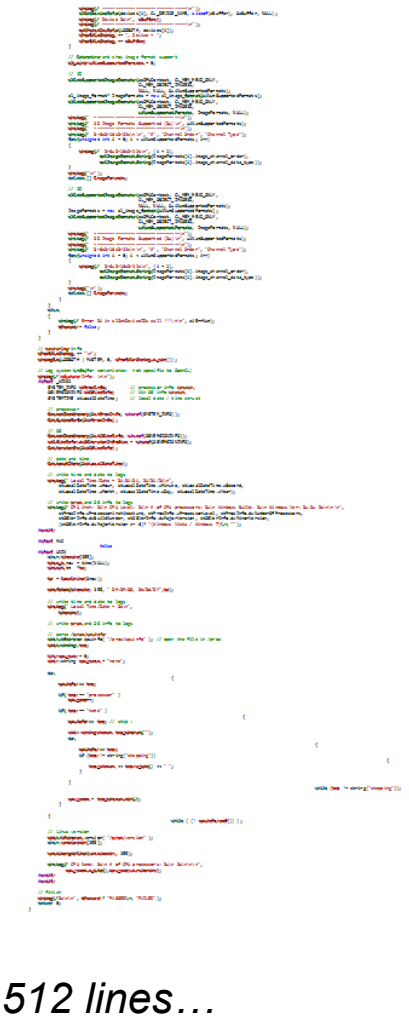
- How PTask schedules GPUs like CPUs
- Locality-aware scheduling
- Proof-of-concept in Linux
- Generality: micro-benchmarks
- …

A DirectCompute program

matrix multiplication: 961 lines

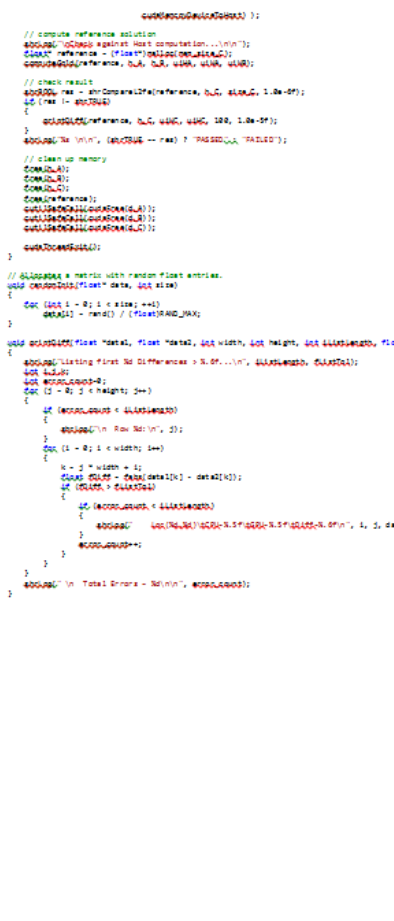
An OpenCL program





better--512 lines…

A CUDA program



even better--240 lines…

A PTask program

```
// matrix multiply
Matrix* gemm(Matrix* A, Matrix* B) {

  Graph * pGraph = new Graph();   // sys_open_graph
  DatablockTemplate * pTemplate   = GetDatablockTemplate(stride, cols, rows, 1);
  CompiledKernel * pMatmulKernel  = GetCompiledKernel("matmul.hlsl", "gemm");

  pAxBInputPorts[0]   = CreatePort(INPUT_PORT, pTemplate);
  pAxBInputPorts[1]   = CreatePort(INPUT_PORT, pTemplate);
  pAxBOutputPorts[0]  = CreatePort(OUTPUT_PORT, pTemplate);
  Task * pAxBTask = pGraph->AddTask(pMatmulKernel, pAxBInputPorts, pAxBOutputPorts);
  GraphInputChannel * pAInput = pGraph->AddInputChannel(pAxBInputPorts[0]);
  GraphInputChannel * pBInput = pGraph->AddInputChannel(pAxBInputPorts[1]);
  GraphOutputChannel * pOutput = pGraph->AddOutputChannel(pAxBOutputPorts[0]);

  pAxBTask->SetComputeGeometry(rows, cols, 1);
  pGraph->Run();   // sys_run_graph

  pAInput->Push(pTemplate, A->cells(), pAInput);
  pBInput->Push(pTemplate, B->cells(), pBInput);
  Datablock * pResultBlock = pOutput->Pull();

  // HLSL, tear-down omitted...
  return new Matrix(rows, cols, pResultBlock->GetDataPointer());
}
```

wow: 30-40 lines!

Related Work

- Prior work : regular (scientific, HPC) apps
  - Accelerator : Array computations **[ASPLOS '06]**
  - StreamIt → CUDA **[CGO '09, LCTES '09]**
  - MATLAB →CUDA compiler **[PLDI '11]**
- OS support for heterogeneous platforms:
  - Helios **[Nightingale 09]**, BarrelFish **[Baumann 09]** ,Offcodes **[Weinsberg 08]**
- GPU Scheduling
  - TimeGraph **[Kato 11]**, Pegasus **[Gupta 11]**
- Graph-based programming models
  - Synthesis **[Masselin 89]**, Monsoon/Id **[Arvind]**, Dryad **[Isard 07]**
  - StreamIt **[Thies 02]**, DirectShow, TCP Offload **[Currid 04]**
- Tasking
  - Tessellation, Apple GCD, …

Conclusion and Future Work

- Better abstractions for GPUs are critical
  - Enable fairness & priority
  - OS can use the GPU
- Dataflow: a good fit abstraction
  - system manages data movement
  - performance benefits significant
- Proof of concept: LINQ on GPUs
- Future work
  - GPU code generation
  - Automatic type inference
  - Automatic task partitioning across CPU and GPU
  - Finding *best* parallelization for a given LINQ query

Conclusion and Future Work

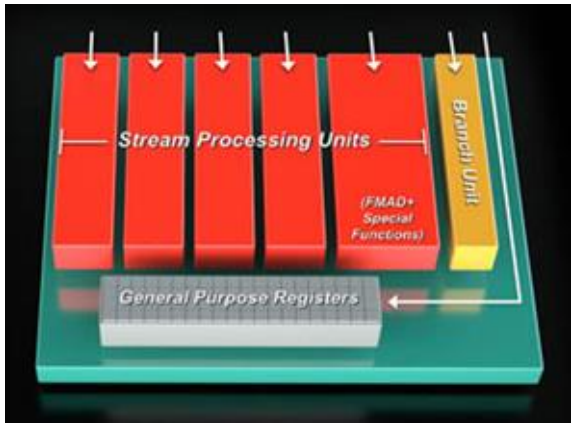
- Better abstractions for GPUs are critical
  - Enable fairness & priority
  - OS can use the GPU
- Dataflow: a good fit abstraction
  - system manages data movement
  - performance benefits significant
- Proof of concept: LINQ on GPUs
- Future work
  - GPU code generation
  - Automatic type inference
  - Automatic task partitioning across CPU and GPU
  - Finding *best* parallelization for a given LINQ query

Thank you! Questions?

Backup slides

GPU Architecture Trends

- What about
  - NVIDIA Tegra
  - AMD Fusion
  - Qualcomm Snapdragon
  - Intel Larrabee/Sandybridge
  - Apple Ax
  - Freescale i.MX
  - …
- GPUs on PCIe not going away
- PTask is about accelerators
  - interrupts/coherent mem on *everything?*

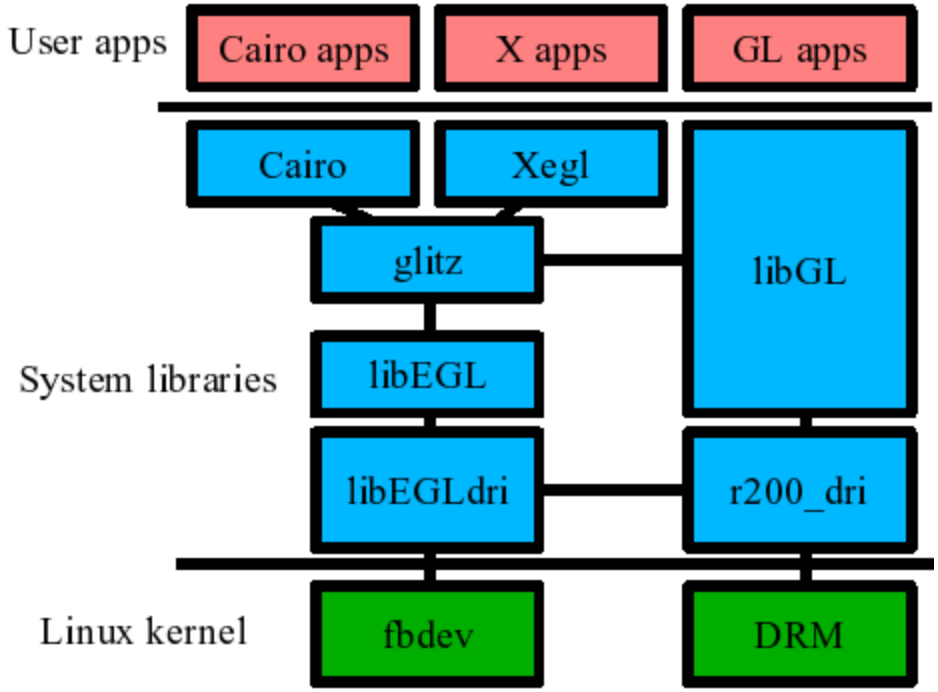


From: http://insidehpc.com/category/hpc-hardware/compute/page/2/

What about WDDM?

- Tightly coupled with display
  - Can't even open a Tesla card w/DirectX!
- (Non-functional) priority not exposed
- Preemption fails without GPU hw-support
  - and is the common case
- Interface cumbersome/slow
  - CUDA developers hacking around it to preserve perf
- What's right: a standard interface
  - Exposing some semantics of device
- I'd love to know more about WDDM 2.0!

Linux Graphics Stack



Gestural Interface Perf Shootout

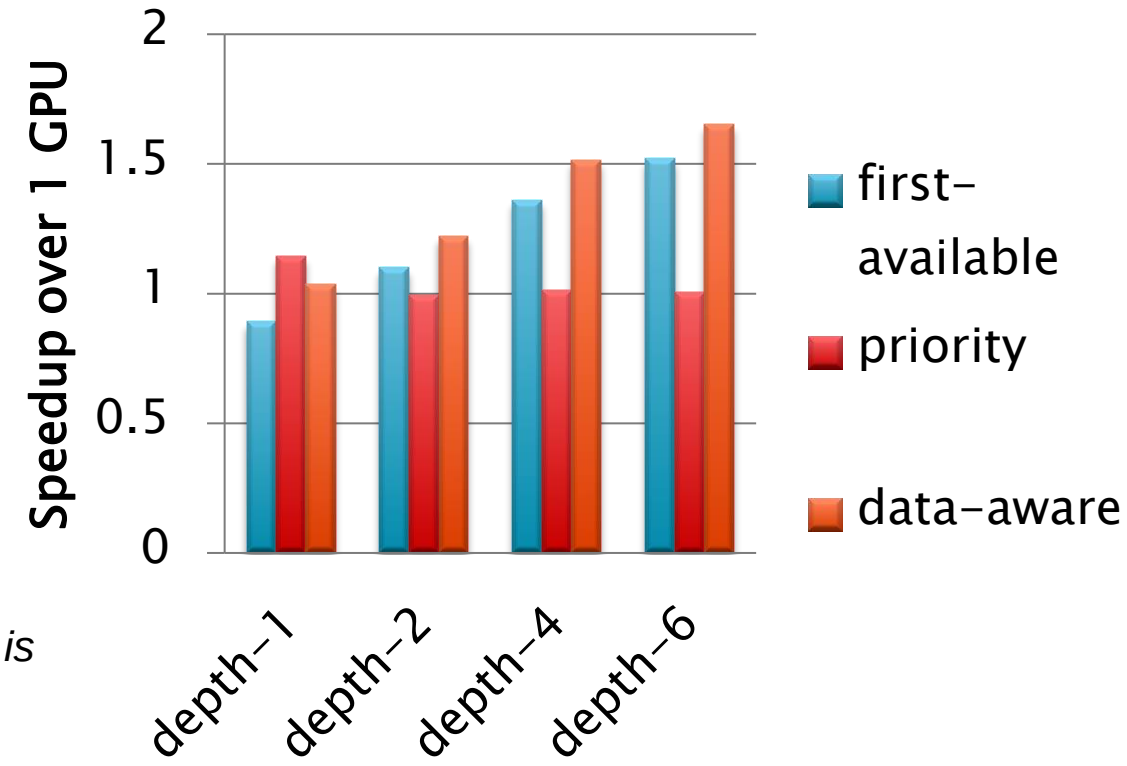
impl	bnc	fps	MB/s	lat(ms)	user	kern	gpu	thrds	ws+-
host-based	RT	20	35.8	36.6	78.1%	3.9%	0%	6	--
	max	20	35.8	36.6	78.1%	3.9%	0%	6	--
handcode	RT	30	53.8	13.0	0.5%	0.3%	4.5%	3	0%
	max	103.2	185.0	13.0	9.2%	5.7%	8.2%	3	0%
pipes	RT	30	53.8	15.1	3.1%	4.3%	5.3%	3	15.4%
	max	85.0	152.4	15.1	16.7%	17.8%	7.6%	3	15.5%
ptask	RT	30	53.8	12.5	0.2%	0.6%	3.5%	8	3.5%
	max	108.3	194.1	12.5	10.5%	6.2%	8.1%	8	6.7%

- Windows 7 x64 8GB RAM
- Intel Core 2 Quad 2.66GHz
- GTX580 (EVGA)

Offcodes [Weinsberg 08]

- Similarities:
  - OS + GPU
  - Motivated by similar data migration issues
  - Graph-based programming model
- Differences
  - Host OS +target device firmware must support the same offcode API/runtime: device must run offcode runtime
  - NIC: TCP-Offload focused
  - didn't evaluate GPU
  - no scheduler integration
  - GPU still not a first class resource

Data-aware scheduling

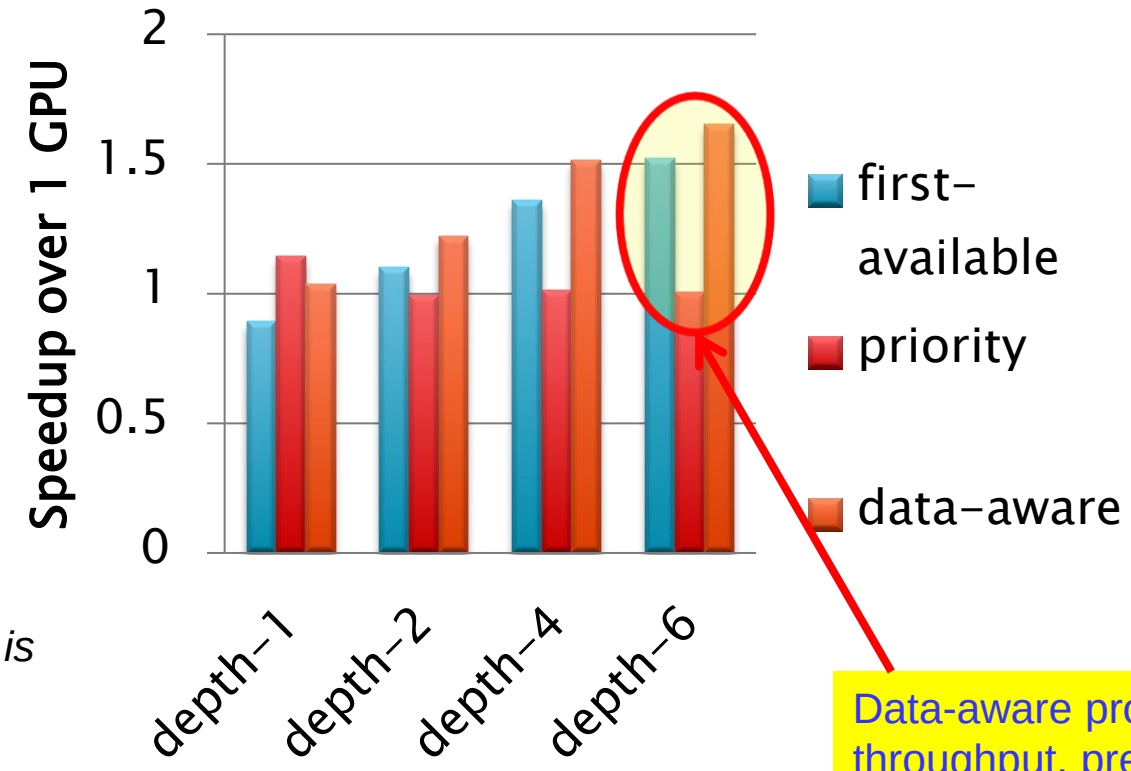


Higher is better

- Data-aware == priority + locality
- depth > 1 req. for any benefit
- priority – aged priority queue w OS prio
- PTask graphs: depth x 6 matrix multiply
- priority same for every PTask node

- Windows 7 x64 8GB RAM
- Intel Core 2 Quad 2.66GHz
- 2 x GTX580 (EVGA)

Data-aware scheduling



Higher is better

Data-aware provides best throughput, preserves priority

- Data-aware == priority + locality
- depth > 1 req. for any benefit
- priority – aged priority queue w OS prio
- PTask graphs: depth x 6 matrix multiply
- priority same for every PTask node

- Windows 7 x64 8GB RAM
- Intel Core 2 Quad 2.66GHz
- 2 x GTX580 (EVGA)

Linux+EncFS Throughput

	GPU/ CPU	1-cont Linux	1-cont PTask	2-cont Linux	2-cont PTask
Read	1.17x	-10x	1.16x	-30.9x	1.16x
Write	1.28x	-8.2x	1.21x	-10.0x	1.20x

- EncFS, running nice -20
- GPU contenders run nice +19
- Linux 2.6.33
- AES in XTS chaining mode
- "GPU" means AES → GTX470
- "CPU" means AES → SSL lib impl
- Core i5 3.2GHz, 12GB RAM
- 2x80GB SATA SSD, RAID (md)
- seq. read/write 200 MB file
- tput proportional to prio (see paper)

Linux+EncFS Throughput

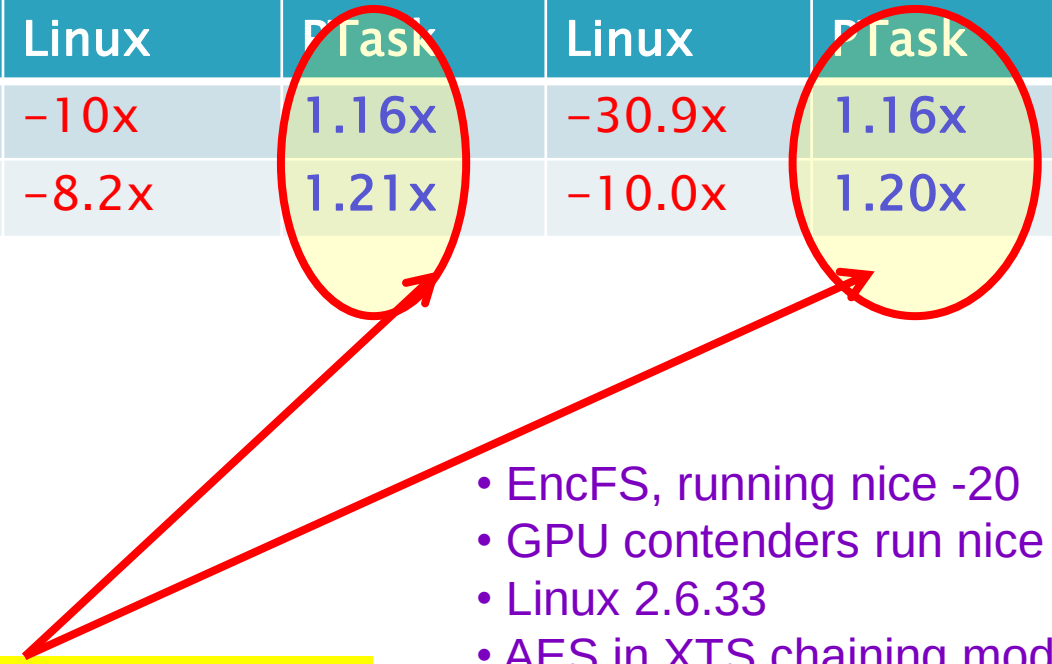
	GPU/ CPU	1-cont Linux	1-cont PTask	2-cont Linux	2-cont PTask
Read	1.17x	-10x	1.16x	-30.9x	1.16x
Write	1.28x	-8.2x	1.21x	-10.0x	1.20x

GPU defeats OS scheduler
- Despite EncFS nice -19
- Despite contenders nice +20

- EncFS, running nice -20
- GPU contenders run nice +19
- Linux 2.6.33
- AES in XTS chaining mode
- "GPU" means AES → GTX470
- "CPU" means AES → SSL lib impl
- Core i5 3.2GHz, 12GB RAM
- 2x80GB SATA SSD, RAID (md)
- seq. read/write 200 MB file
- tput proportional to prio (see paper)

Linux+EncFS Throughput

	GPU/ CPU	1-cont Linux	1-cont PTask	2-cont Linux	2-cont PTask
Read	1.17x	-10x	1.16x	-30.9x	1.16x
Write	1.28x	-8.2x	1.21x	-10.0x	1.20x



Simple GPU usage accounting
- Restores performance
- Does not require preemption!

- EncFS, running nice -20
- GPU contenders run nice +19
- Linux 2.6.33
- AES in XTS chaining mode
- "GPU" means AES → GTX470
- "CPU" means AES → SSL lib impl
- Core i5 3.2GHz, 12GB RAM
- 2x80GB SATA SSD, RAID (md)
- seq. read/write 200 MB file
- tput proportional to prio (see paper)

Why isn't `ioctl()` enough?

- We expect traditional OS guarantees:
  - Fairness
  - Isolation

  No user-space runtime can provide these!
- No kernel-facing interface
  - The OS cannot use the GPU
  - OS cannot manage the GPU
- Lost optimization opportunities
  - Suboptimal data movement
  - Poor composability

Location Transparency: Datablocks

- Map of memory domains -> Buffers
  - CPU memory domain
  - Memory domain per GPU instance
- Automatically track buffer invalidation
  - Optimize copies between domains
- Access flags (read/write; mutable/immutable)
  - Each GPU runtime has different flavors of buffer
  - Dynamically determine need at runtime
  - Hide complexities from programmer

System call interface

Ptask system call	Description
sys_open_graph(name)	Open or create a PTask graph
sys_open_port(name, type, template)	Open or create a port
sys_open_ptask(nm, krnl, graph, ports)	Open or create a new Ptask bound to a graph and ports
sys_open_channel(name, src, dst)	Open or create a channel bound to given ports
sys_open_template(name, dims)	Open/create datablock template
sys_push(channel|port)	Write to a channel or port
sys_pull(channel|port)	Read from a channel or port
sys_run_graph(graph)	Move graph to running state
sys_terminate_graph(graph)	Terminate graph execution
sys_set_ptask_prio(name, prio)	Set priority for a ptask or graph
sys_set_geometry(ptask, dims)	Set mapping from datablock to GPU threads

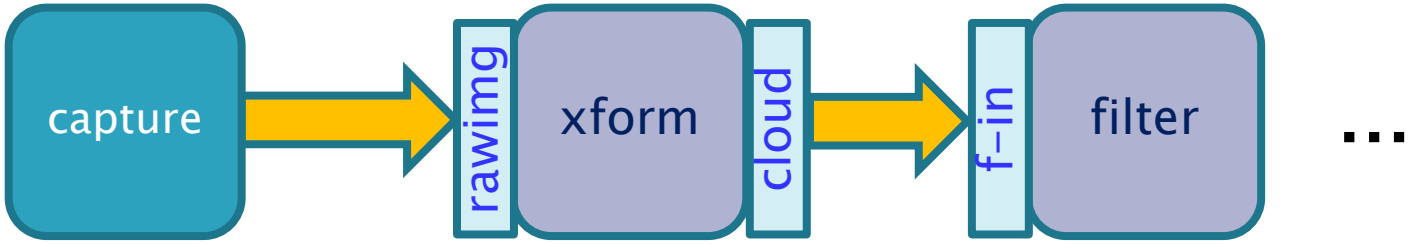
System call interface

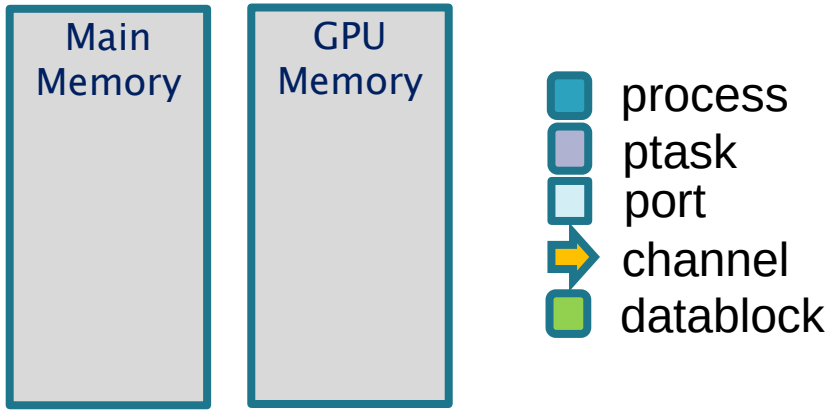
Ptask system call	Description
sys_open_graph(name)	Open or create a PTask graph
sys_open_port(name, type, template)	Open or create a port
sys_open_ptask(nm, krnl, graph, ports)	Open or create a new Ptask bound to a graph and ports
sys_open_channel(name, src, dst)	Open or create a channel bound to given ports
sys_open_template(name, dims)	Open/create datablock template
sys_push(channel|port)	
sys_pull(channel|port)	
sys_run_graph(graph)	Move graph to running state
sys_terminate_graph(graph)	Terminate graph execution
sys_set_ptask_prio(name, prio)	Set priority for a ptask or graph
sys_set_geometry(ptask, dims)	Set mapping from datablock to GPU threads

APIs for creating and managing dataflow graphs

Datablock Action Zone

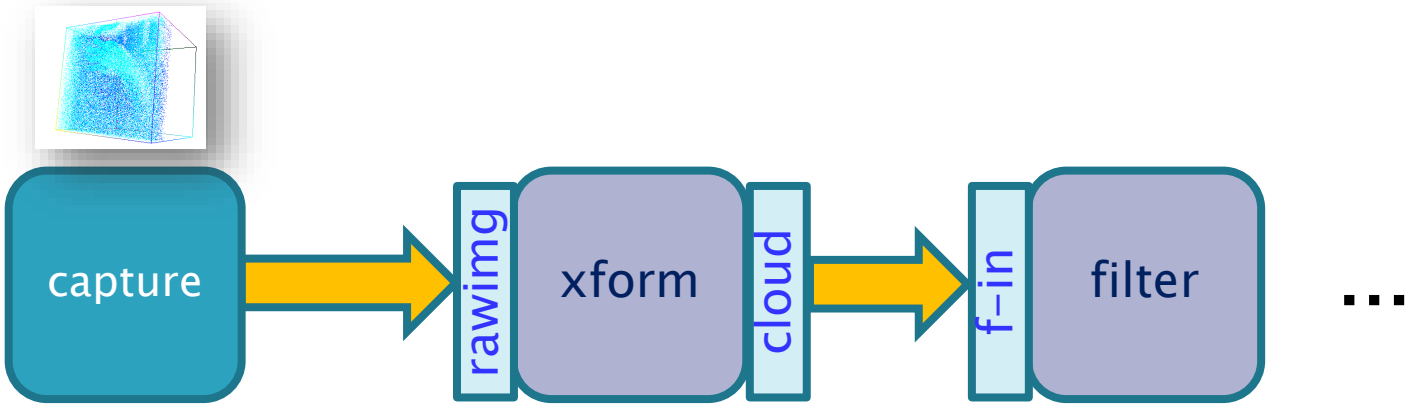
```
#> capture | xform | filter …
```

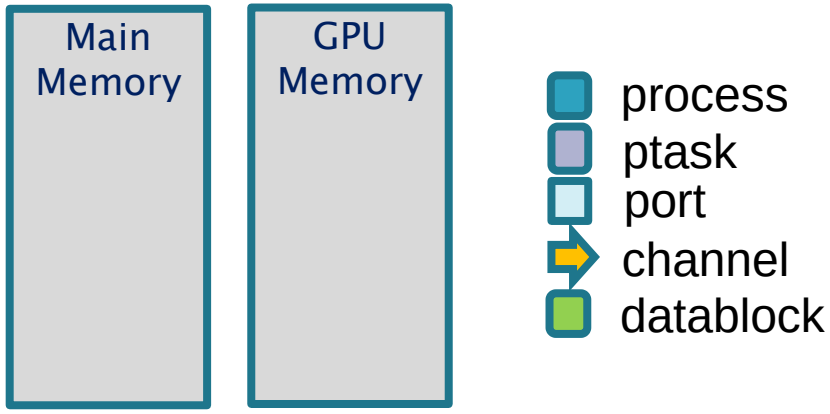




Datablock Action Zone

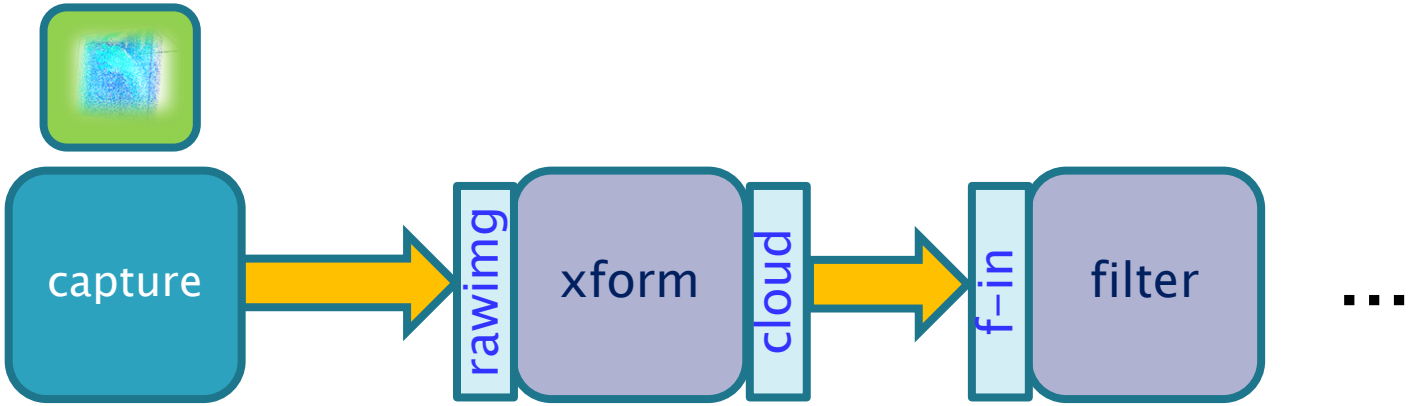
```
#> capture | xform | filter …
```

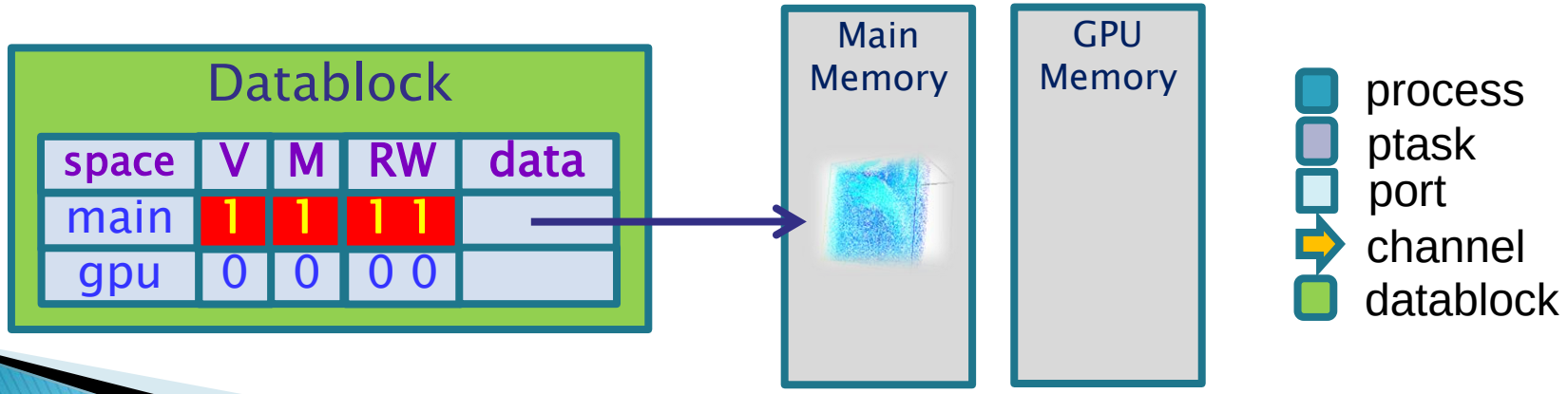




Datablock Action Zone

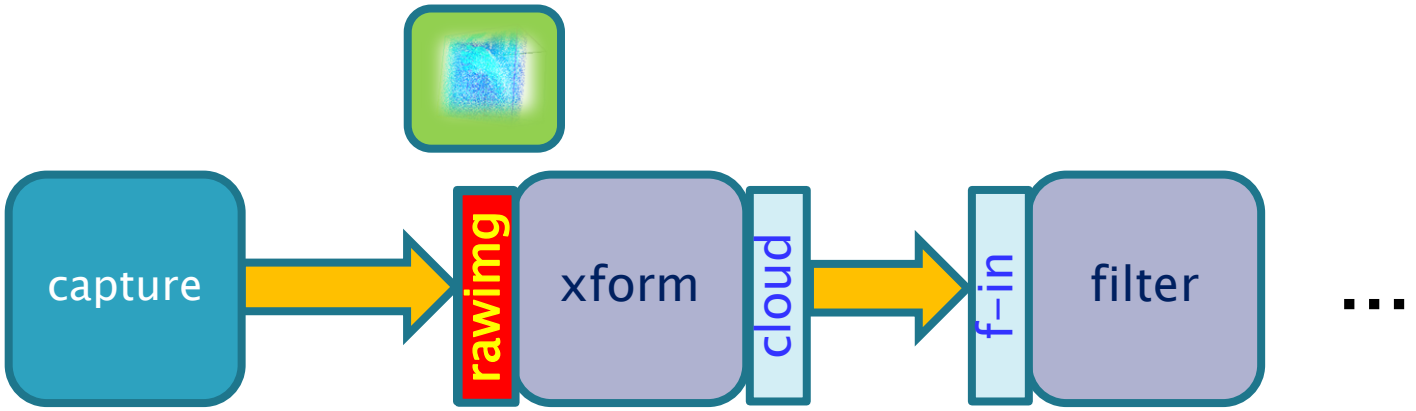
```
#> capture | xform | filter …
```

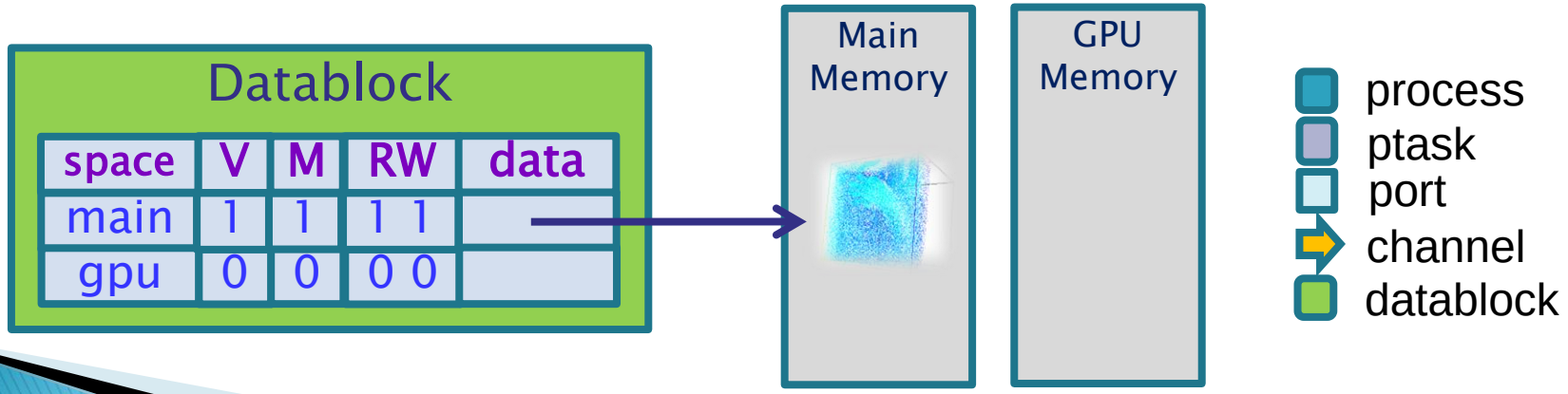




Datablock Action Zone

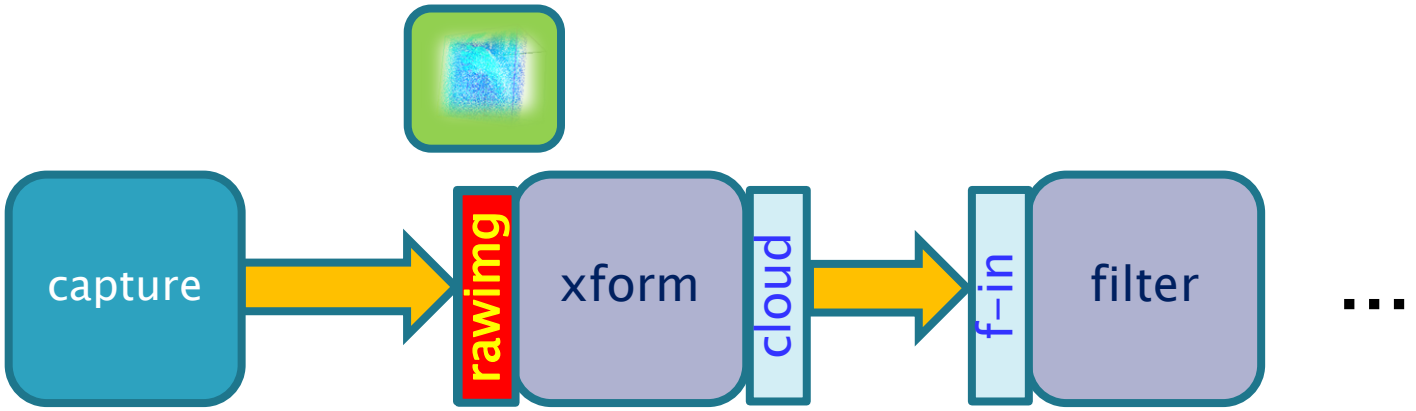
#> capture | xform | filter …

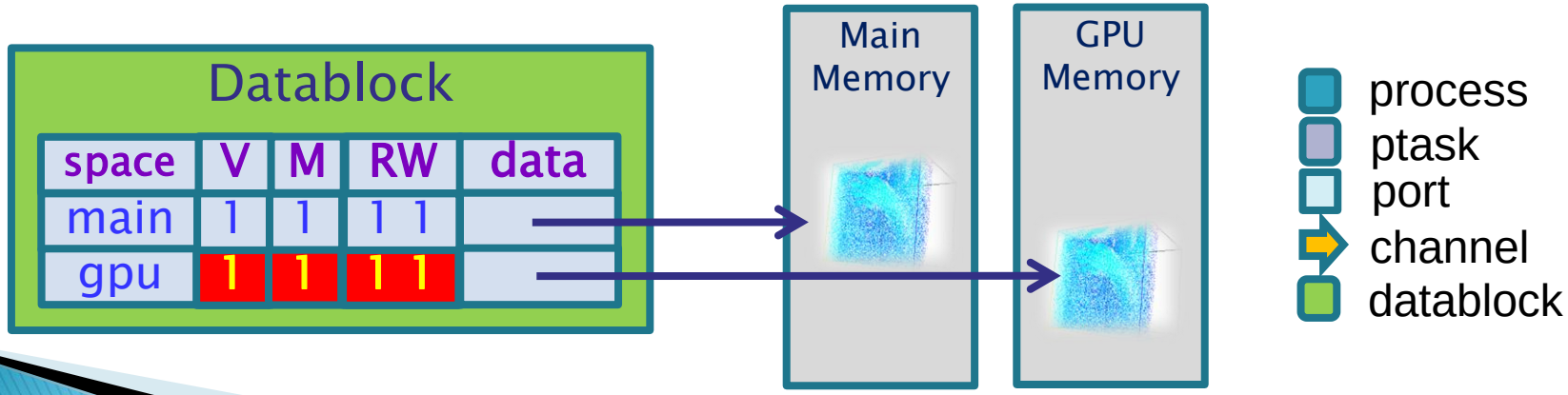




Datablock Action Zone

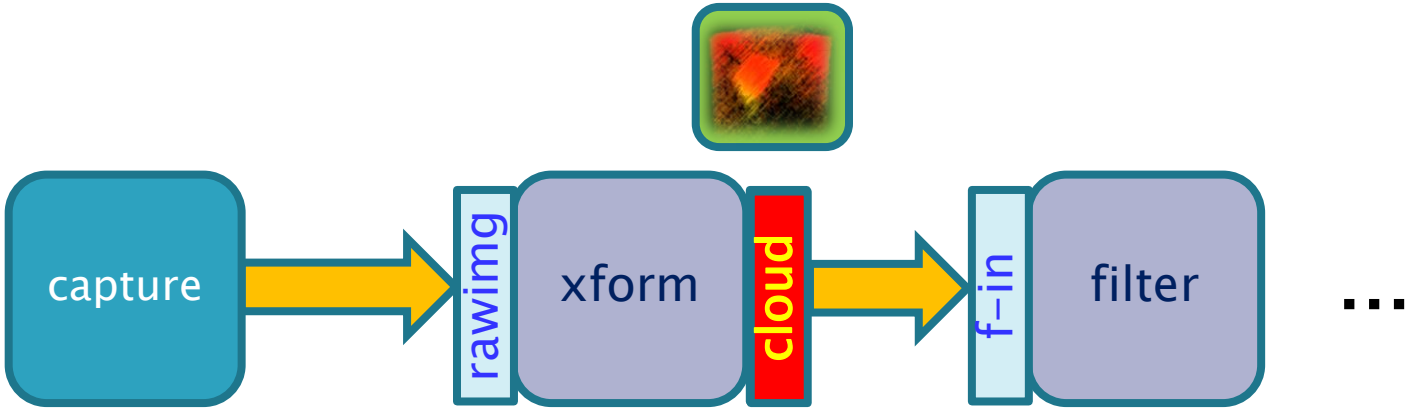
#> capture | xform | filter …

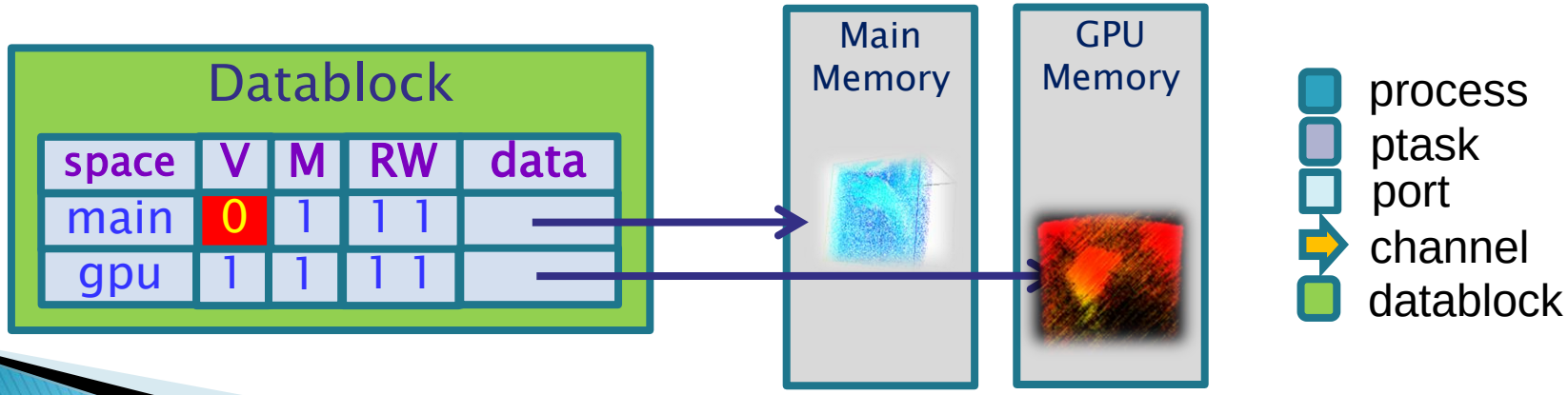




Datablock Action Zone

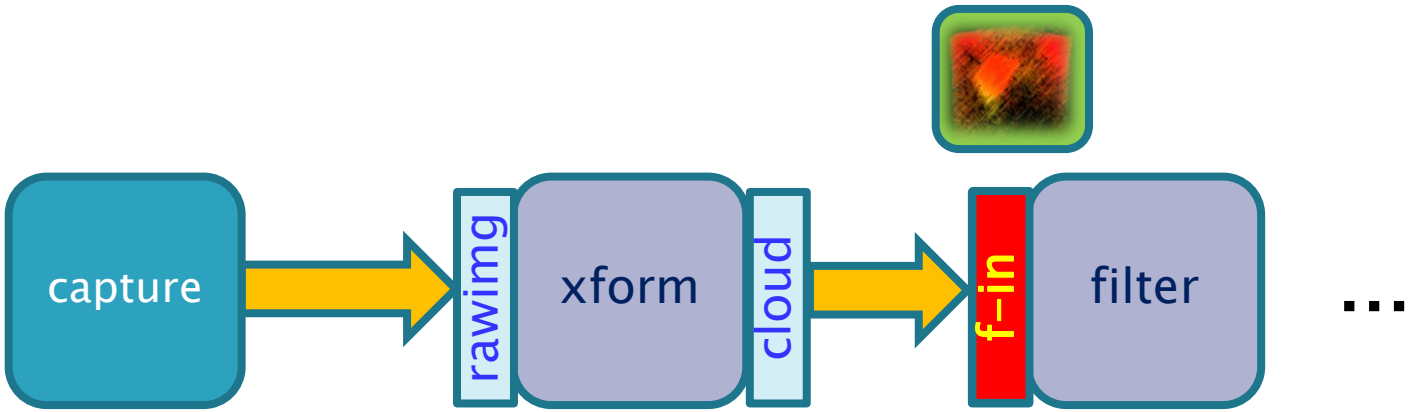
#> capture | xform | filter …

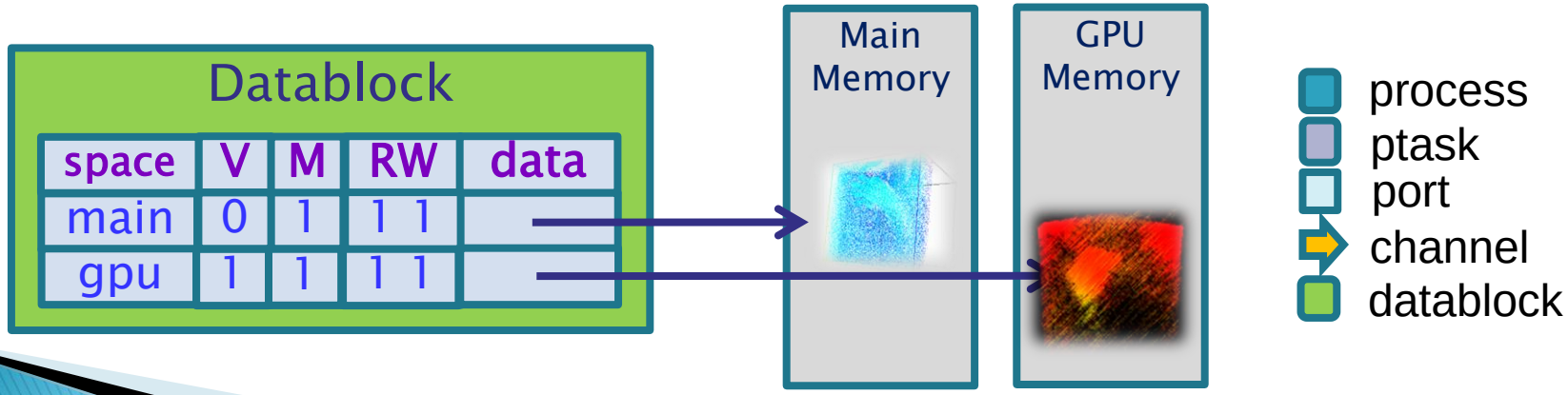




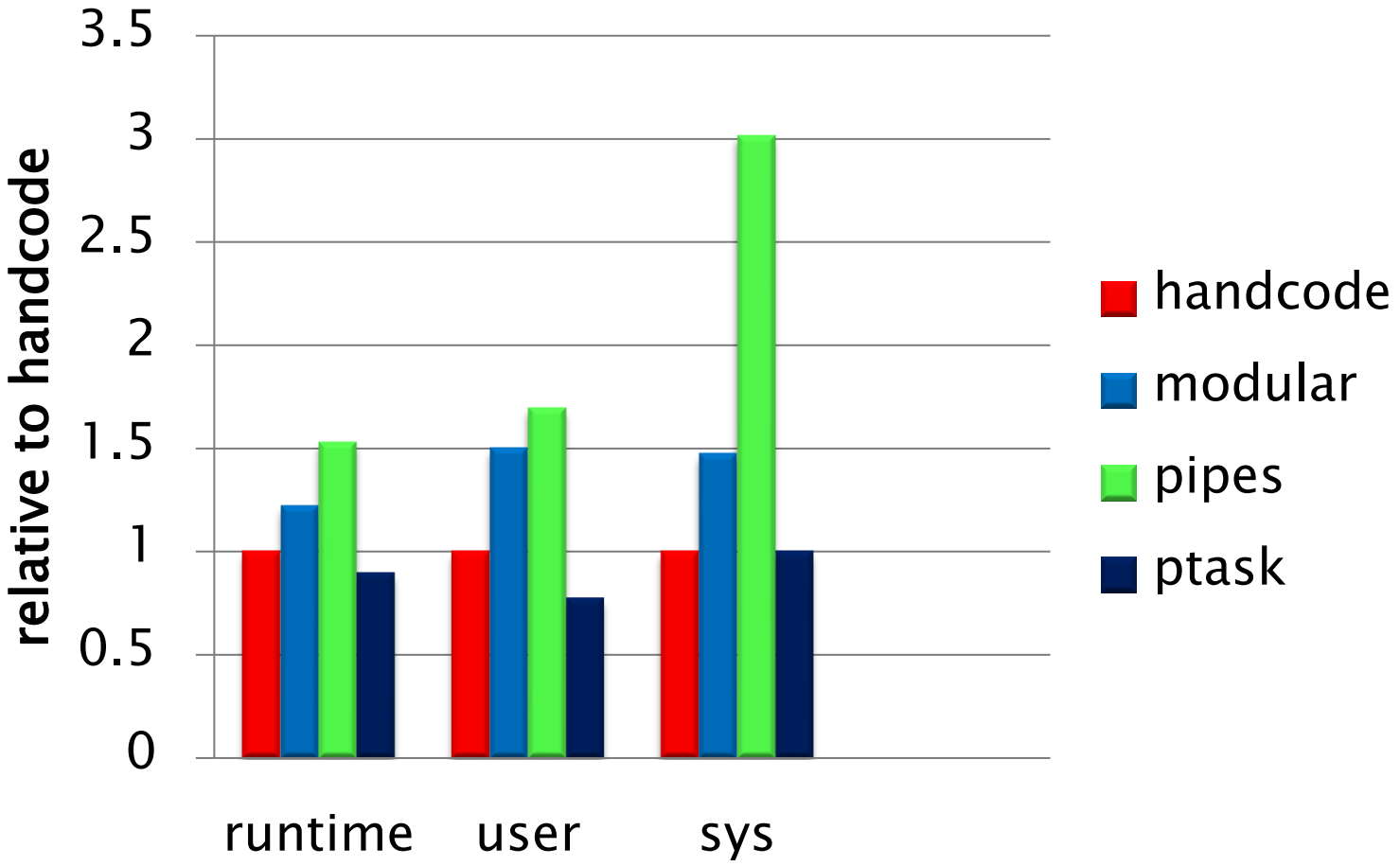
Datablock Action Zone

```
#> capture | xform | filter …
```





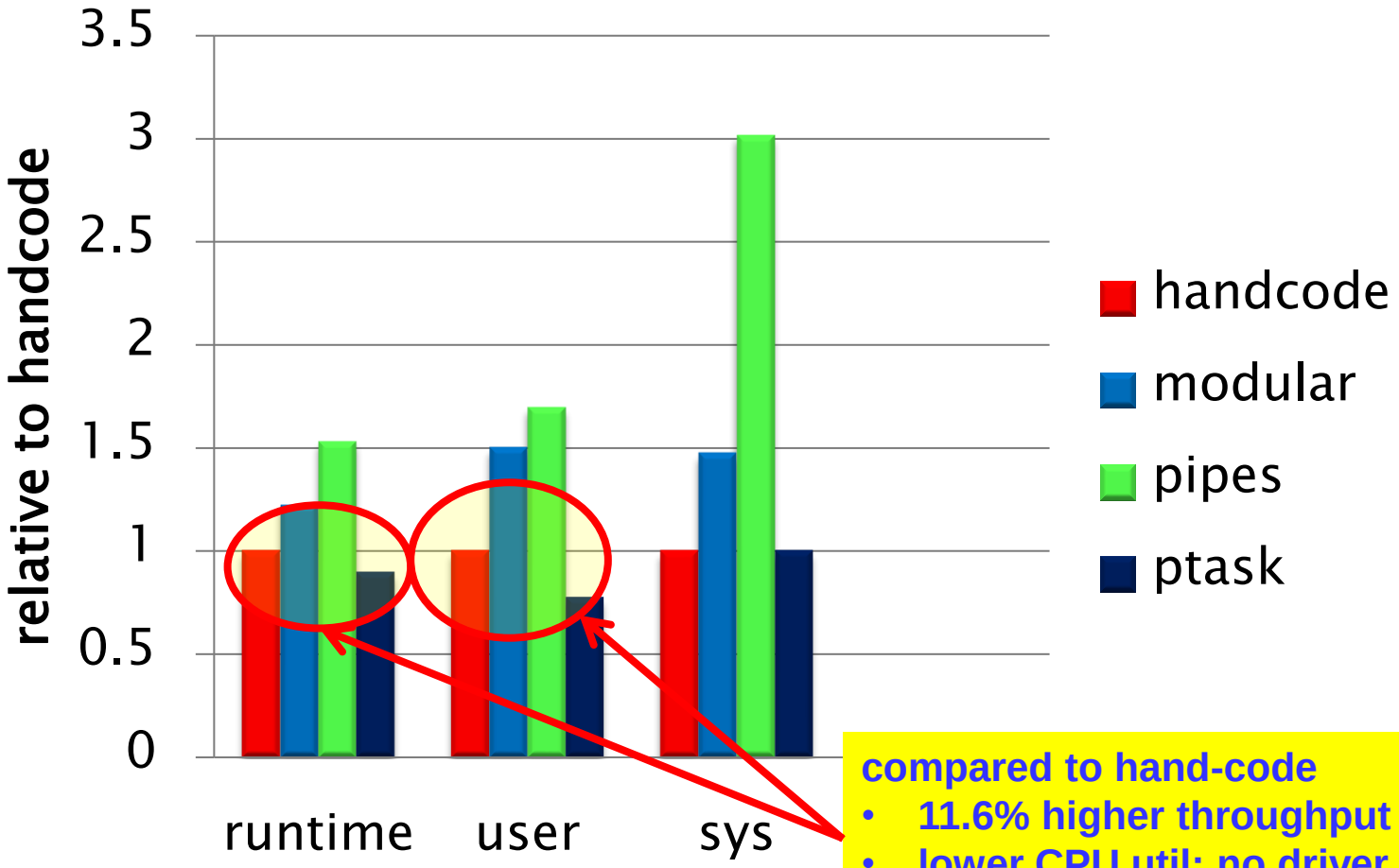
Gestural Interface Performance



lower is better

- Windows 7 x64 8GB RAM
- Intel Core 2 Quad 2.66GHz
- GTX580 (EVGA)

Gestural Interface Performance

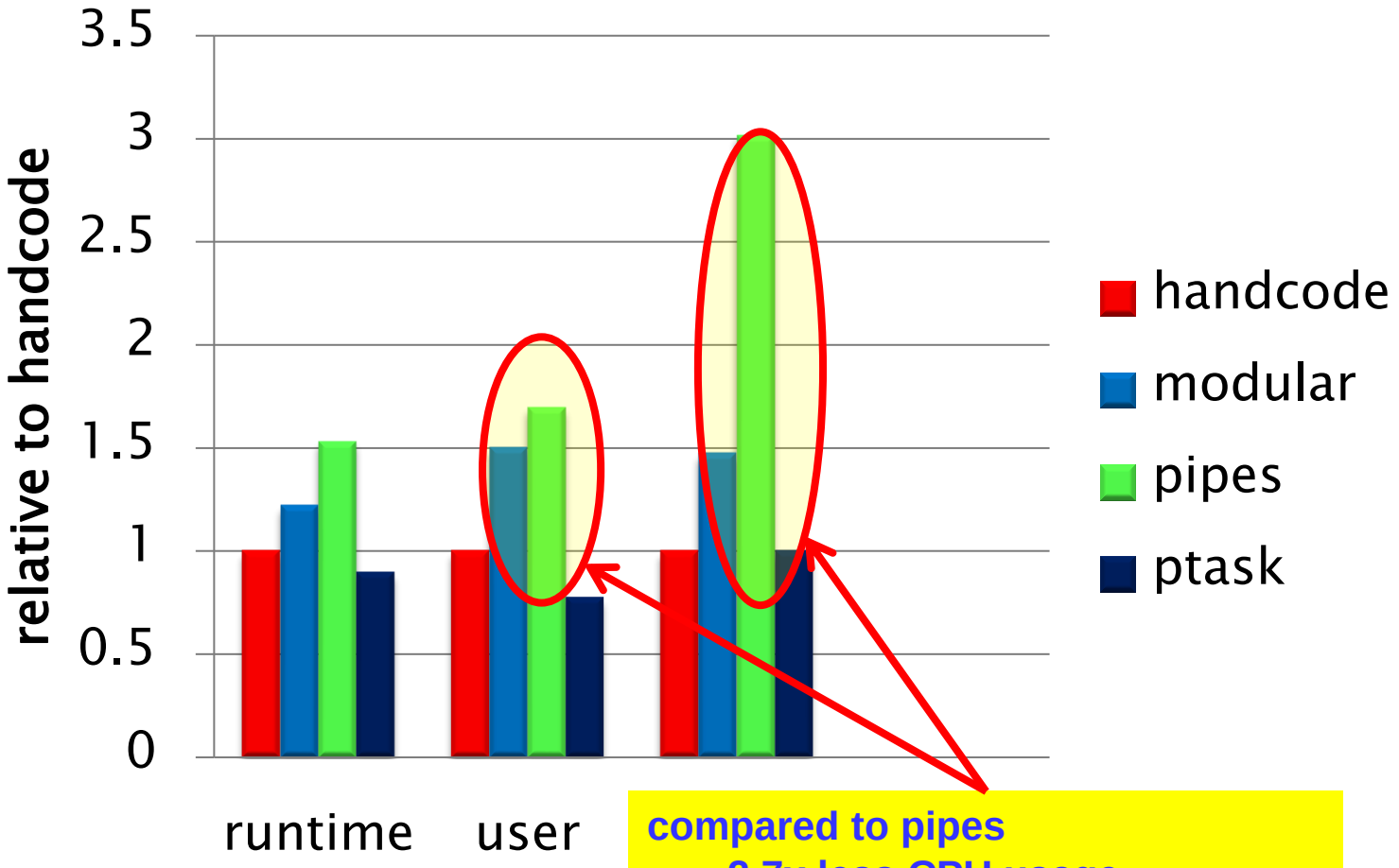


lower is better

compared to hand-code
- **11.6% higher throughput**
- **lower CPU util: no driver program**

- Intel Core 2 Quad 2.66GHz
- GTX580 (EVGA)

Gestural Interface Performance

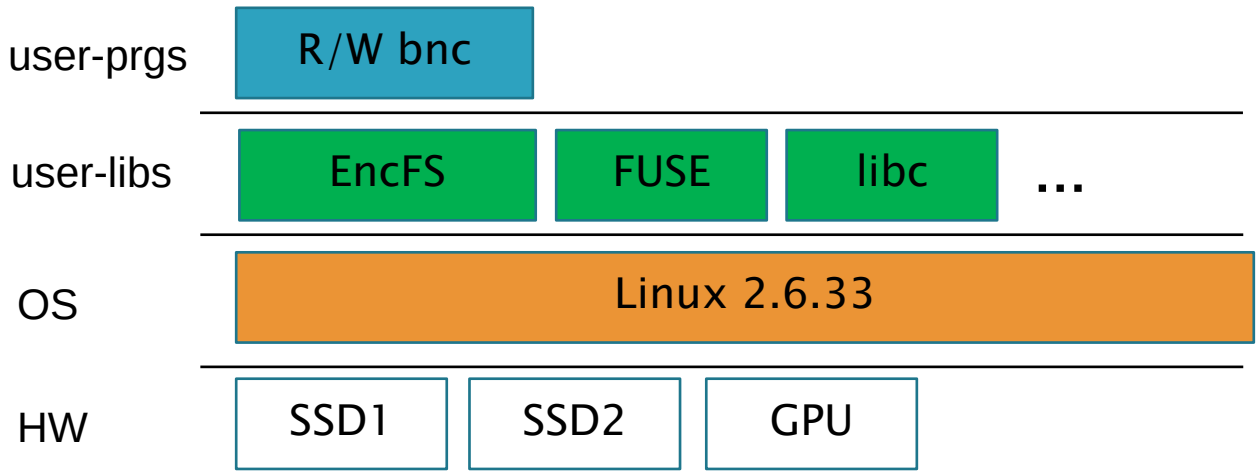


lower is better

compared to pipes
- **~2.7x less CPU usage**
- **16x higher throughput**
- **~45% less memory usage**

GB RAM
d 2.66GHz
• GTX580 (EVGA)

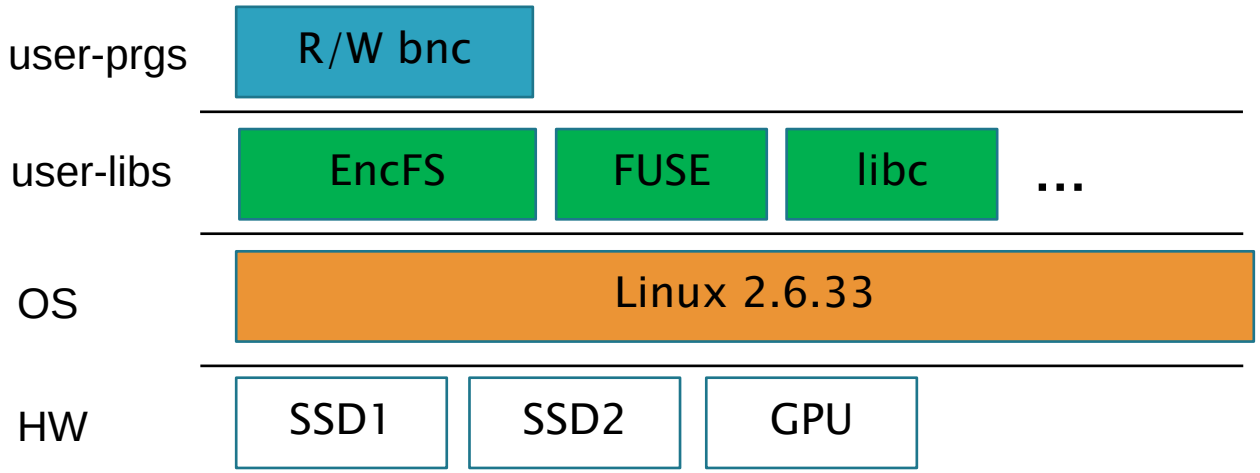
Linux+EncFS Throughput



	GPU/ CPU	cuda-1 Linux	cuda-2 Linux	cuda-1 PTask	cuda-2 Ptask
Read	1.17x	-10.3x	-30.8x	1.16x	1.16x
Write	1.28x	-4.6x	-10.3x	1.21x	1.20x

- EncFS: nice -20
- cuda-*: nice +19
- AES: XTS chaining
- SATA SSD, RAID
- seq. R/W 200 MB

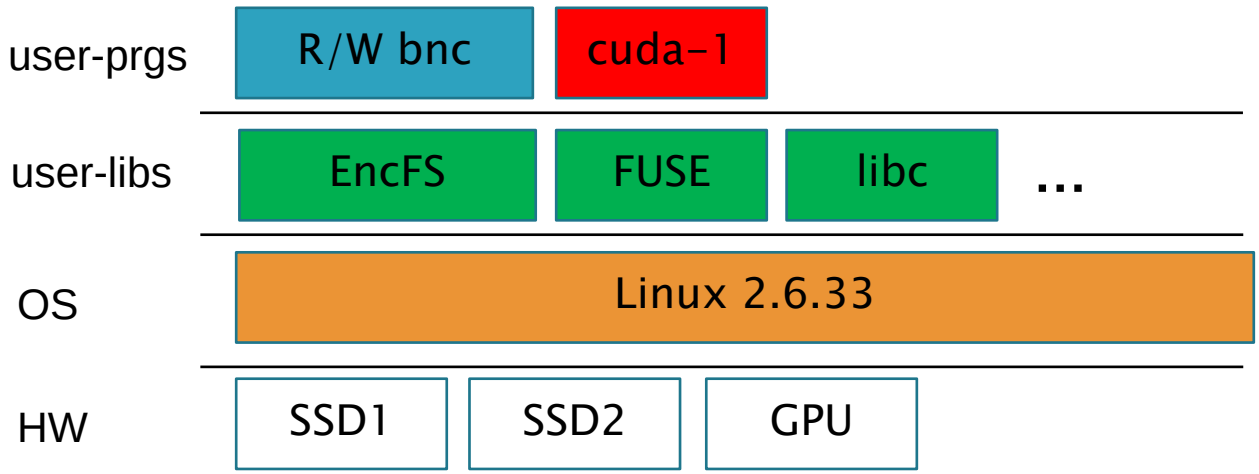
Linux+EncFS Throughput



	GPU/ CPU	cuda-1 Linux	cuda-2 Linux	cuda-1 PTask	cuda-2 Ptask
Read	1.17x	-10.3x	-30.8x	1.16x	1.16x
Write	1.28x	-4.6x	-10.3x	1.21x	1.20x

- EncFS: nice -20
- cuda-*: nice +19
- AES: XTS chaining
- SATA SSD, RAID
- seq. R/W 200 MB

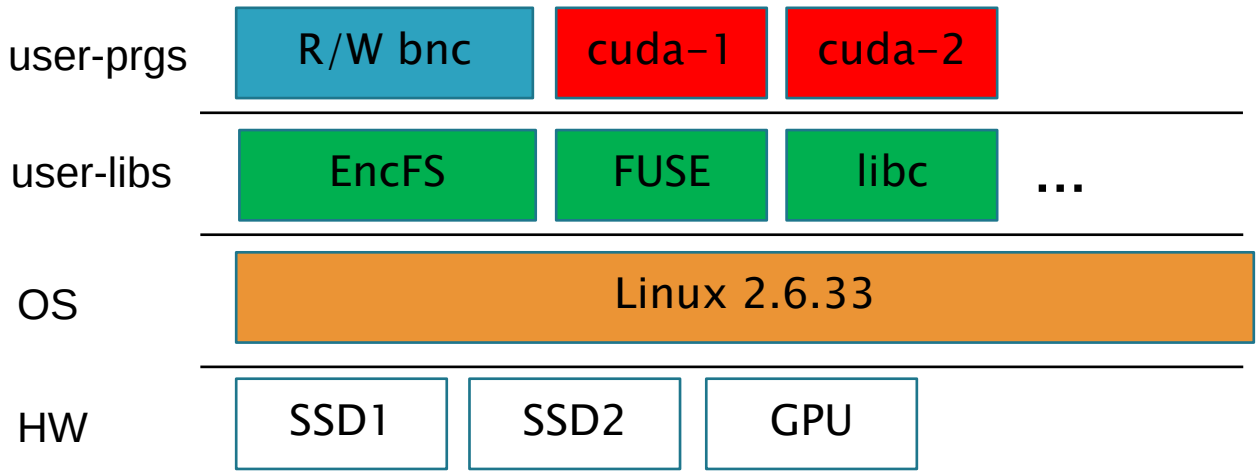
Linux+EncFS Throughput



	GPU/ CPU	cuda-1 Linux	cuda-2 Linux	cuda-1 PTask	cuda-2 Ptask
Read	1.17x	-10.3x	-30.8x	1.16x	1.16x
Write	1.28x	-4.6x	-10.3x	1.21x	1.20x

- EncFS: nice -20
- cuda-*: nice +19
- AES: XTS chaining
- SATA SSD, RAID
- seq. R/W 200 MB

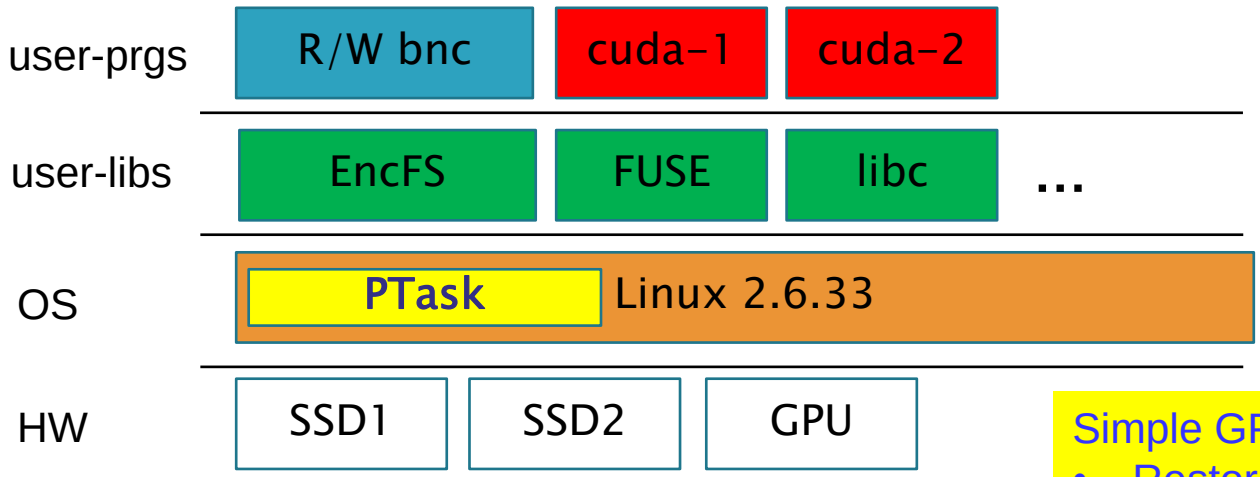
Linux+EncFS Throughput



	GPU/ CPU	cuda-1 Linux	cuda-2 Linux	cuda-1 PTask	cuda-2 Ptask
Read	1.17x	-10.3x	-30.8x	1.16x	1.16x
Write	1.28x	-4.6x	-10.3x	1.21x	1.20x

- EncFS: nice -20
- cuda-*: nice +19
- AES: XTS chaining
- SATA SSD, RAID
- seq. R/W 200 MB

Linux+EncFS Throughput



Simple GPU usage accounting
- Restores performance

	GPU/ CPU	cuda-1 Linux	cuda-2 Linux	cuda-1 PTask	cuda-2 Ptask
Read	1.17x	-10.3x	-30.8x	1.16x	1.16x
Write	1.28x	-4.6x	-10.3x	1.21x	1.20x

- EncFS: nice -20
- cuda-*: nice +19
- AES: XTS chaining
- SATA SSD, RAID
- seq. R/W 200 MB

GPUs need better abstractions

- GPU Analogues for:
  - Process API
  - IPC API
  - Scheduler hints
- Abstractions that enable:
  - Fairness/isolation
  - OS use of GPU
  - Composition/data movement optimization

GPU Execution Model

- Code executed on GPU as ***kernel*** calls
- Kernel calls specify number of threads
  - Thousands of threads execute in parallel
  - No guarantees on thread order/scheduling
  - Preferably all threads follow same control path
- Inter-thread communication support is weak
  - Simple global atomic primitives
  - Synchronization is super-expensive

K-Means in PTask

- PTask extensions:
  - Cycles in the graph
  - Control signals
  - Conditional ports
  - Conditional channels
  - Nodes on CPU
  - Special ports to size outputs dynamically