PTask: Operating System Abstractions to Manage GPUs as Compute Devices

Chris Rossbach, Jon Currey, *Microsoft Research*
Mark Silberstein, *Technion*
Baishakhi Ray, Emmett Witchel, *UT Austin*
SOSP October 25, 2011

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

Motivation

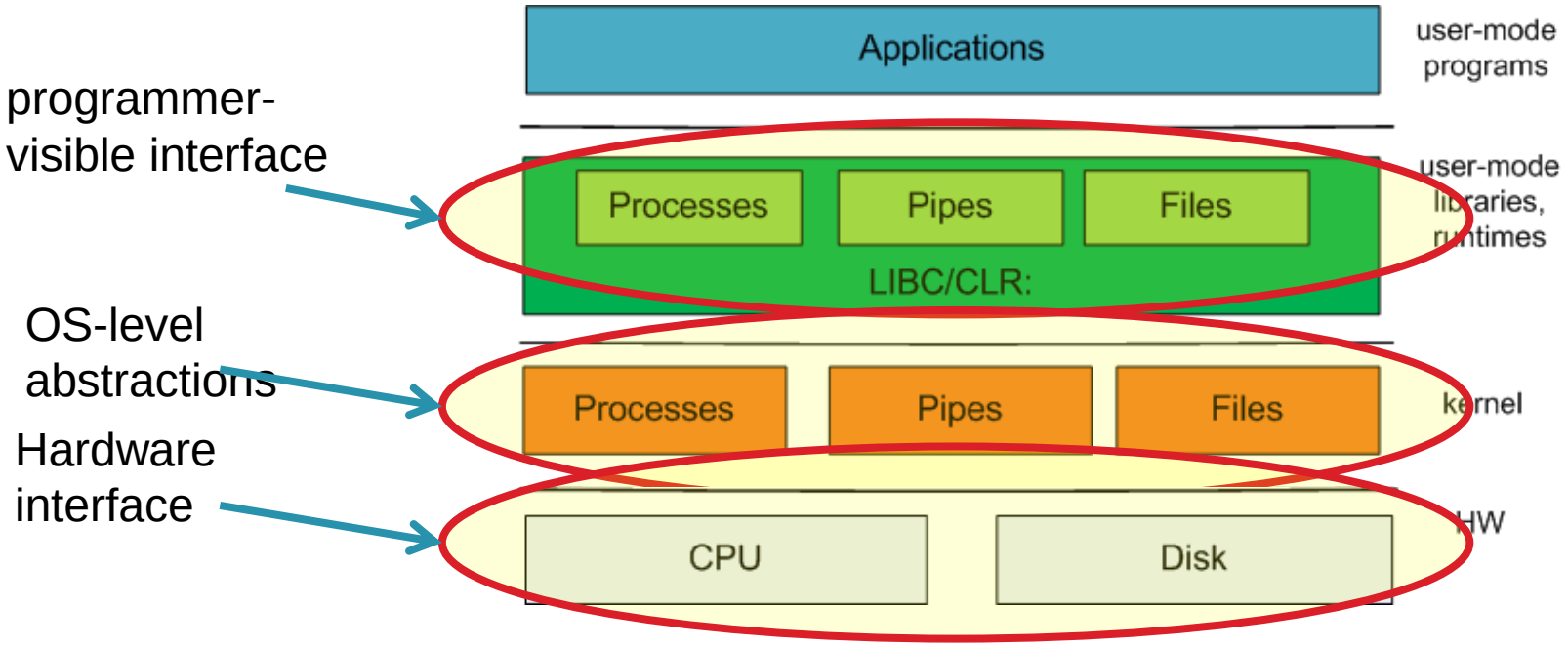
- There are lots of GPUs
  - 3 of top 5 supercomputers use GPUs
  - In all new PCs, smart
  - Great for gaming and
  - ***Unusable in other application domains***
- GPU programing challenges
  - GPU+main memory disjoint
  - ***Treated as I/O device by OS***

These two things are related:
We need OS abstractions

Outline

- The case for OS support
- PTask: Dataflow for GPUs
- Evaluation
- Related Work
- Conclusion

Traditional OS-Level abstractions

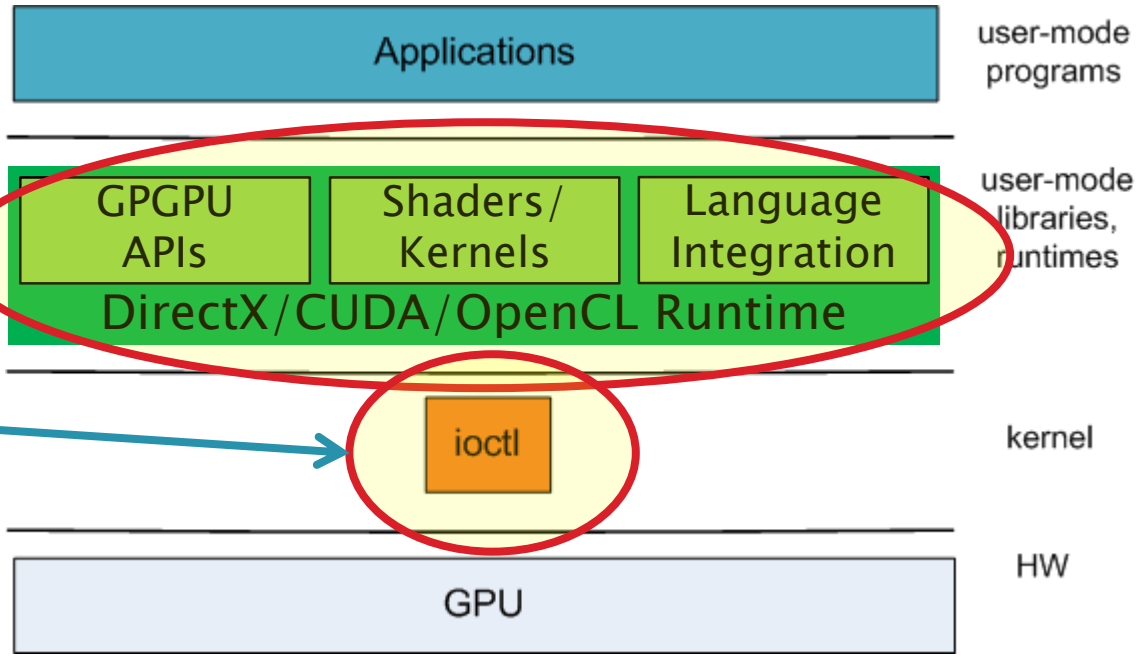


1:1 correspondence between OS-level and user-level abstractions

GPU Abstractions

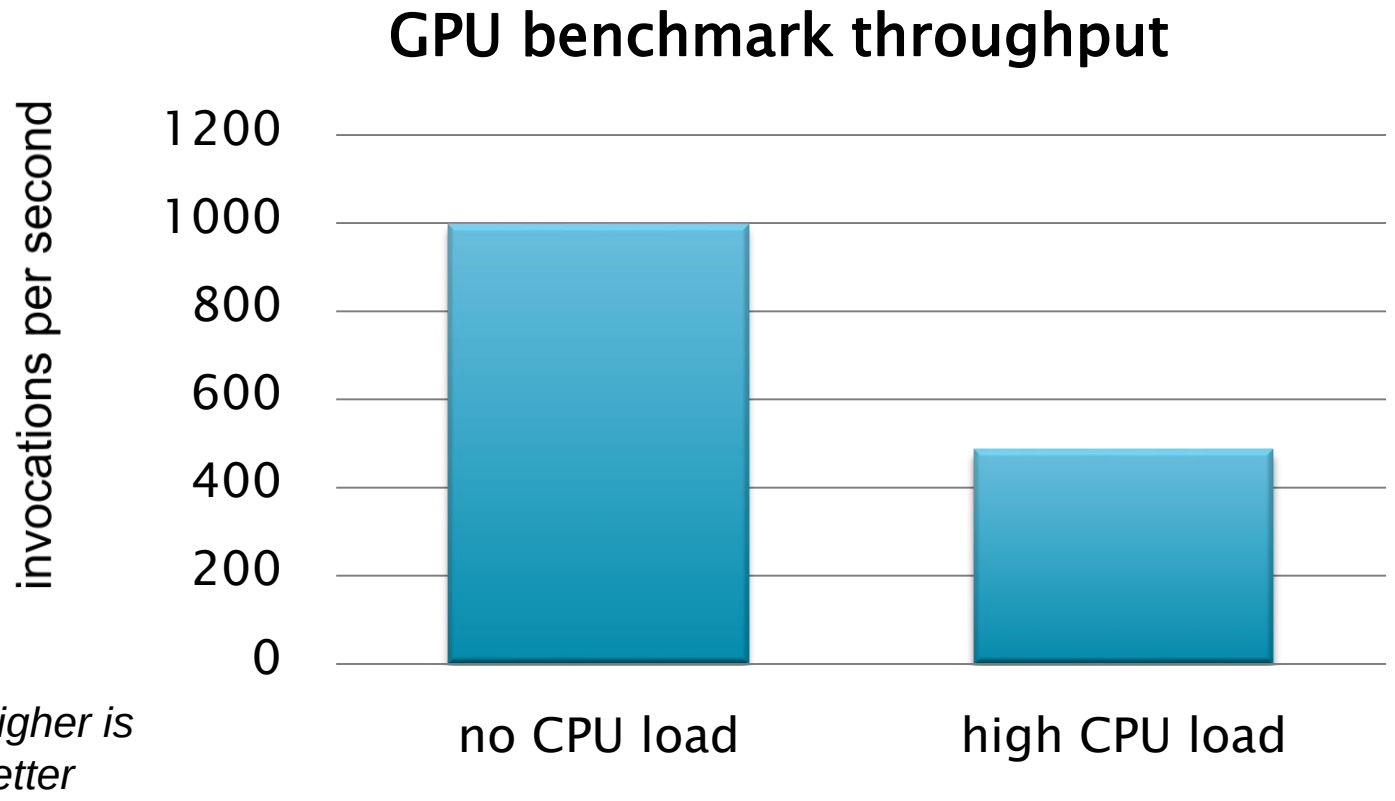
programmer-visible interface

1 OS-level abstraction!



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

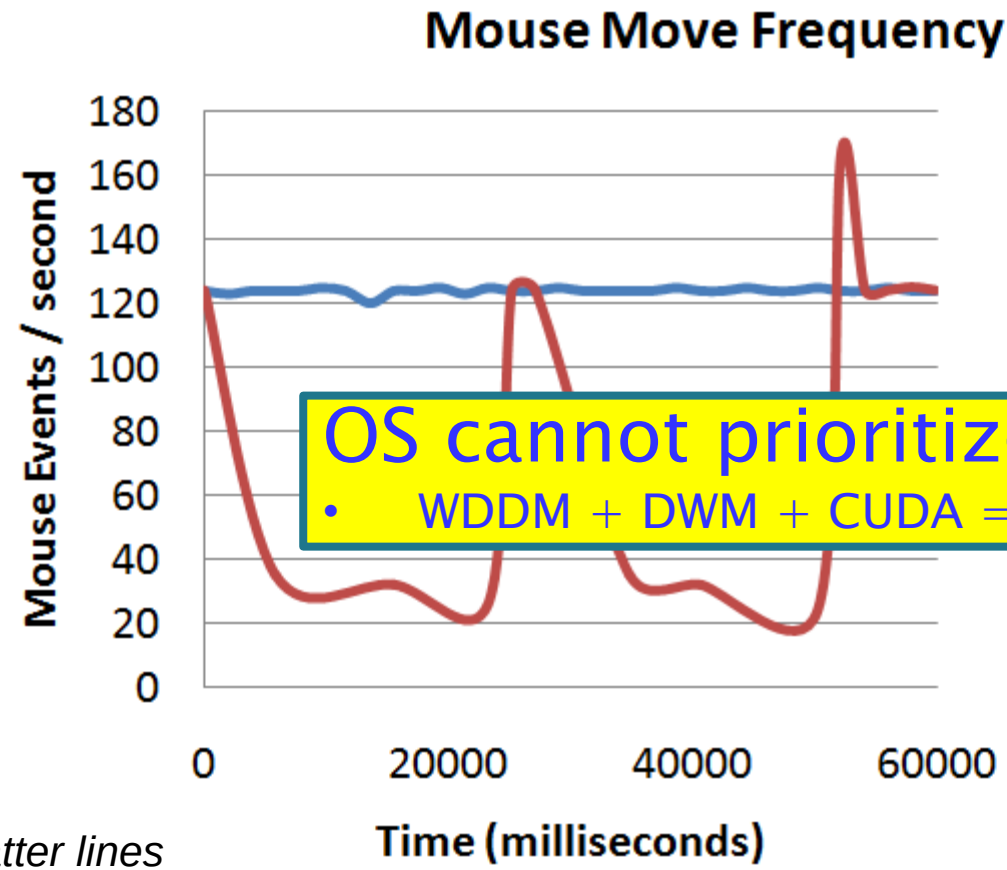
CPU-bound processes hurt GPUs



CPU scheduler and GPU scheduler not integrated!

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

GPU-bound processes hurt CPUs



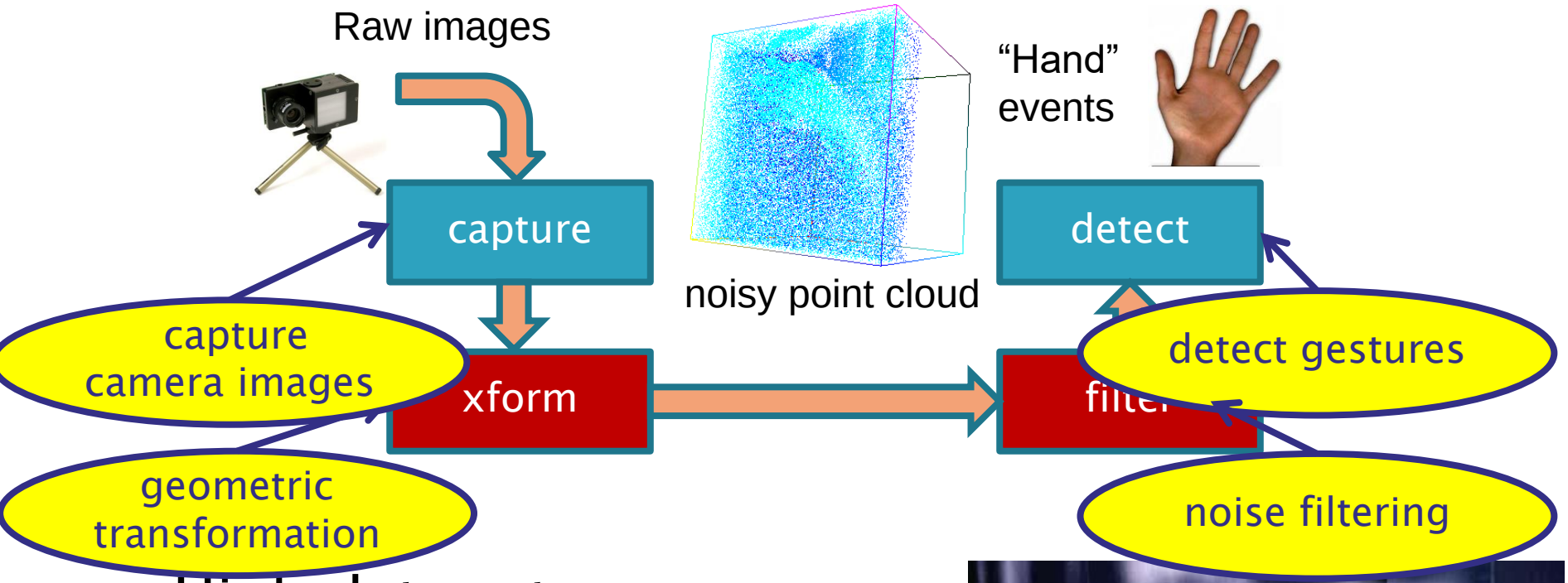
OS cannot prioritize cursor updates

- WDDM + DWM + CUDA == dysfunction

Flatter lines
Are better

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

Composition: 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

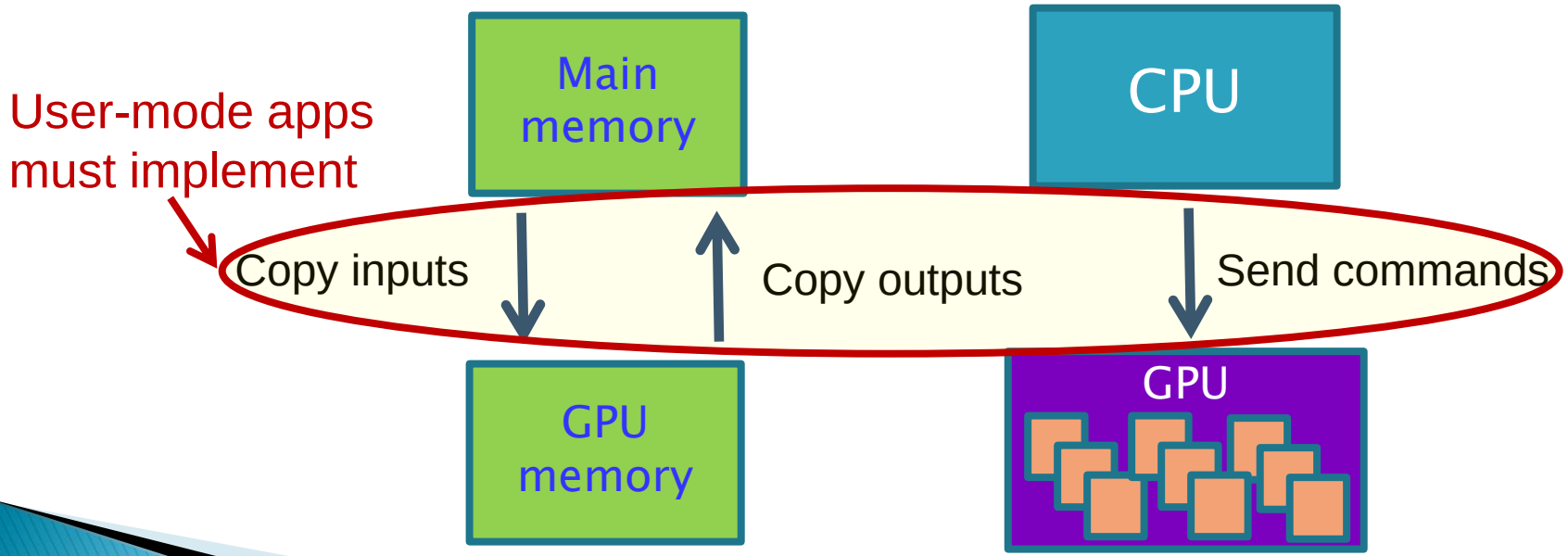
```
#> capture | xform | filter | detect &
```

CPU GPU GPU CPU

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

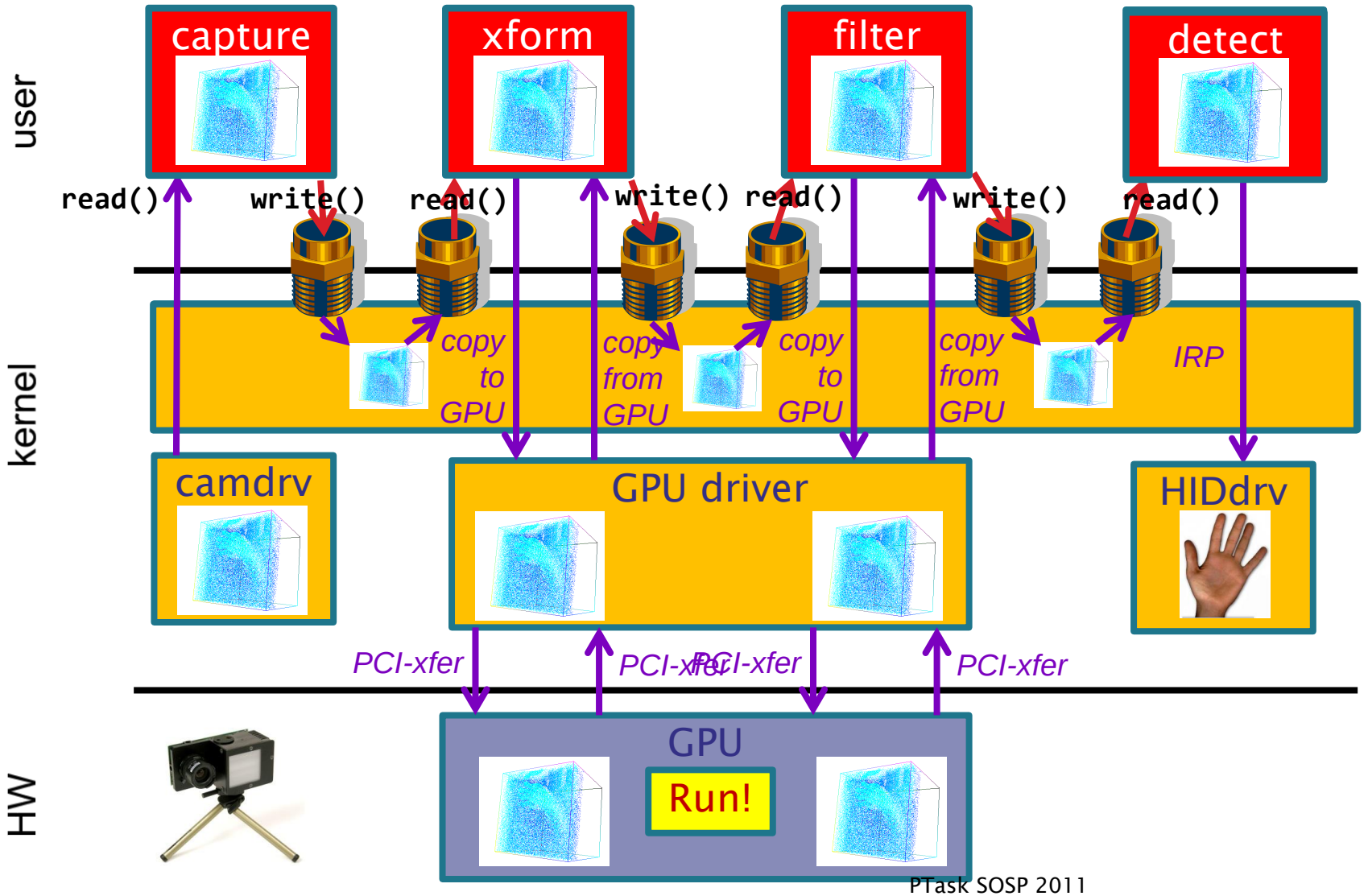
GPU Execution model

- 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



Data migration

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



GPUs need better OS abstractions

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

Outline

- The case for OS support
- PTask: Dataflow for GPUs
- Evaluation
- Related Work
- Conclusion

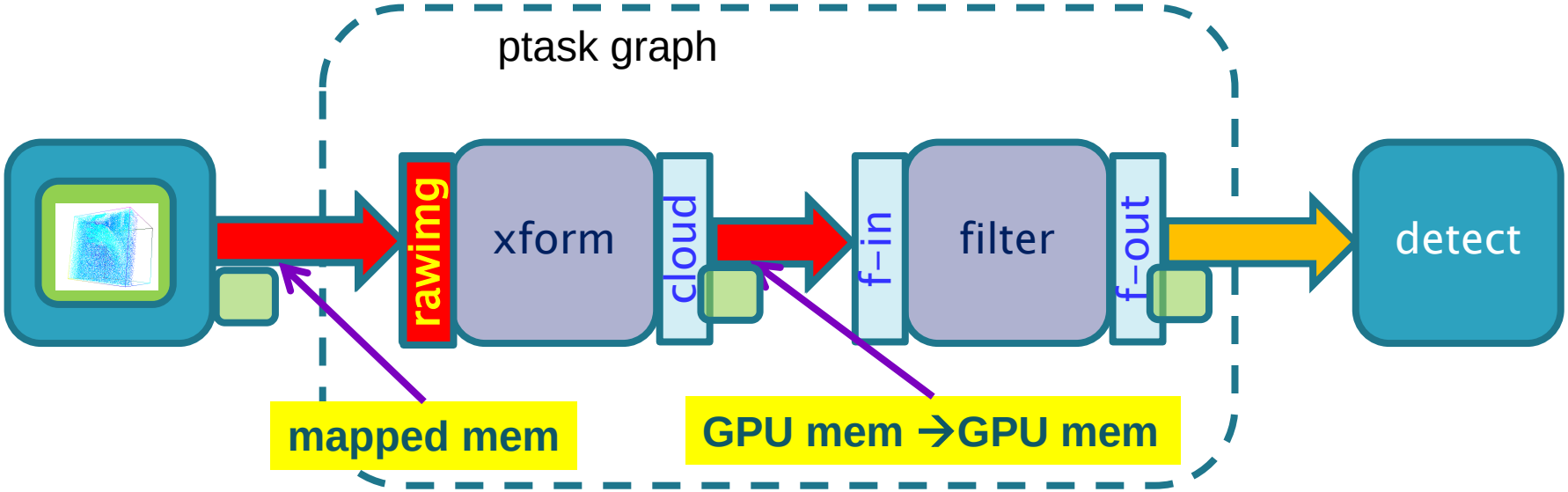
PTask OS abstractions: dataflow!

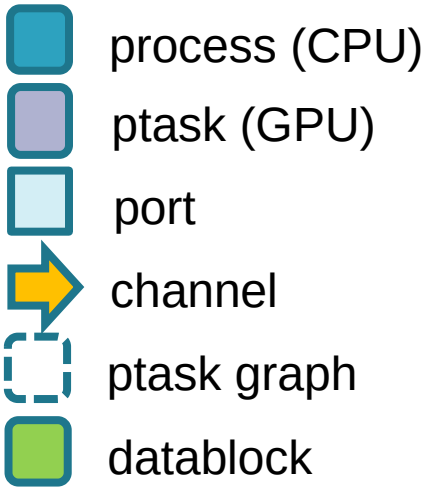
- **ptask** (parallel task)
  - Has ***priority*** for fairness
  - Analogous to a process for GPU execution
  - List of input/output resources (*e.g. stdin, stdout…*)
- **ports**
  - Can be mapped to ptask input/outputs
  - A data source or sink
- **channels**
  - Similar to pipes, connect ar
  - Specialize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

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

PTask Graph: Gestural Interface

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





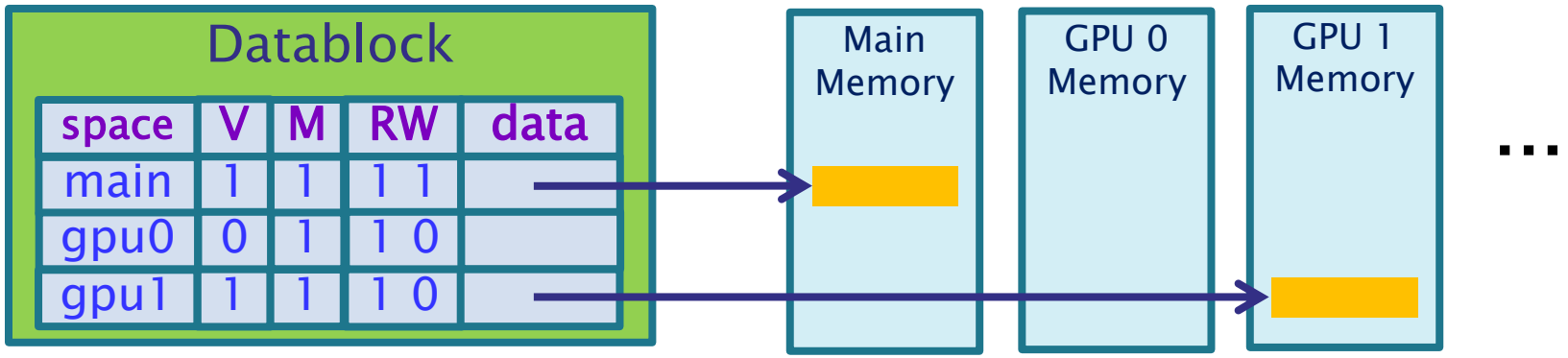
Optimized data movement

Data arrival triggers computation

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

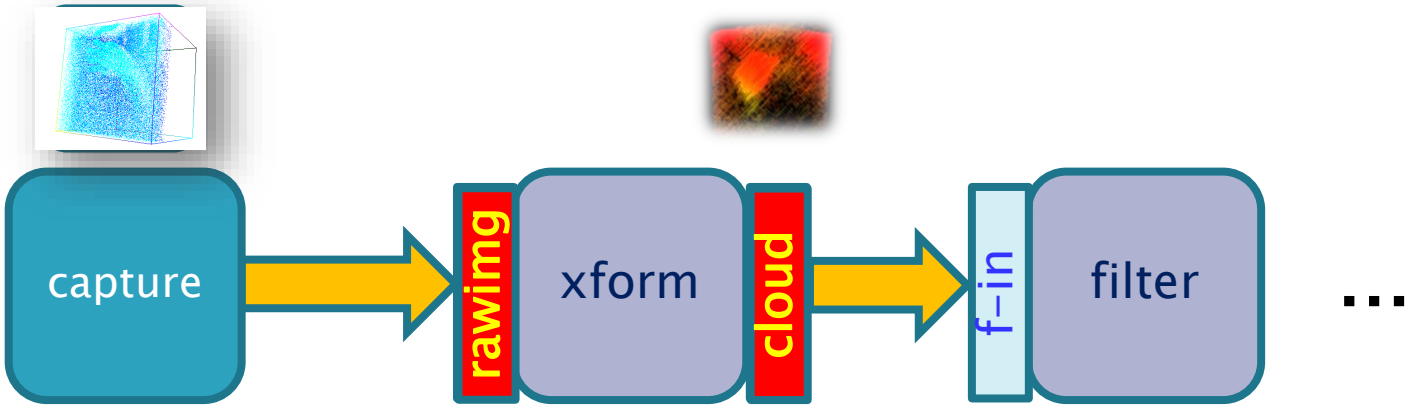
Location Transparency: Datablocks

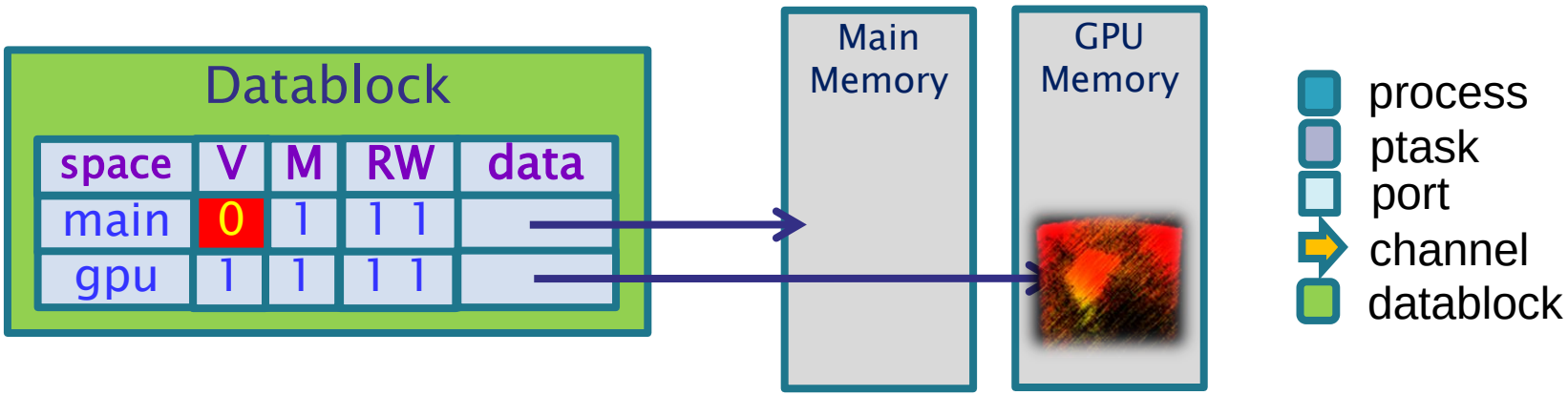


- Logical buffer
  - backed by multiple physical buffers
  - buffers created/updated lazily
  - mem-mapping used to share across process boundaries
- Track buffer validity per memory space
  - writes invalidate other views
- Flags for access control/data placement

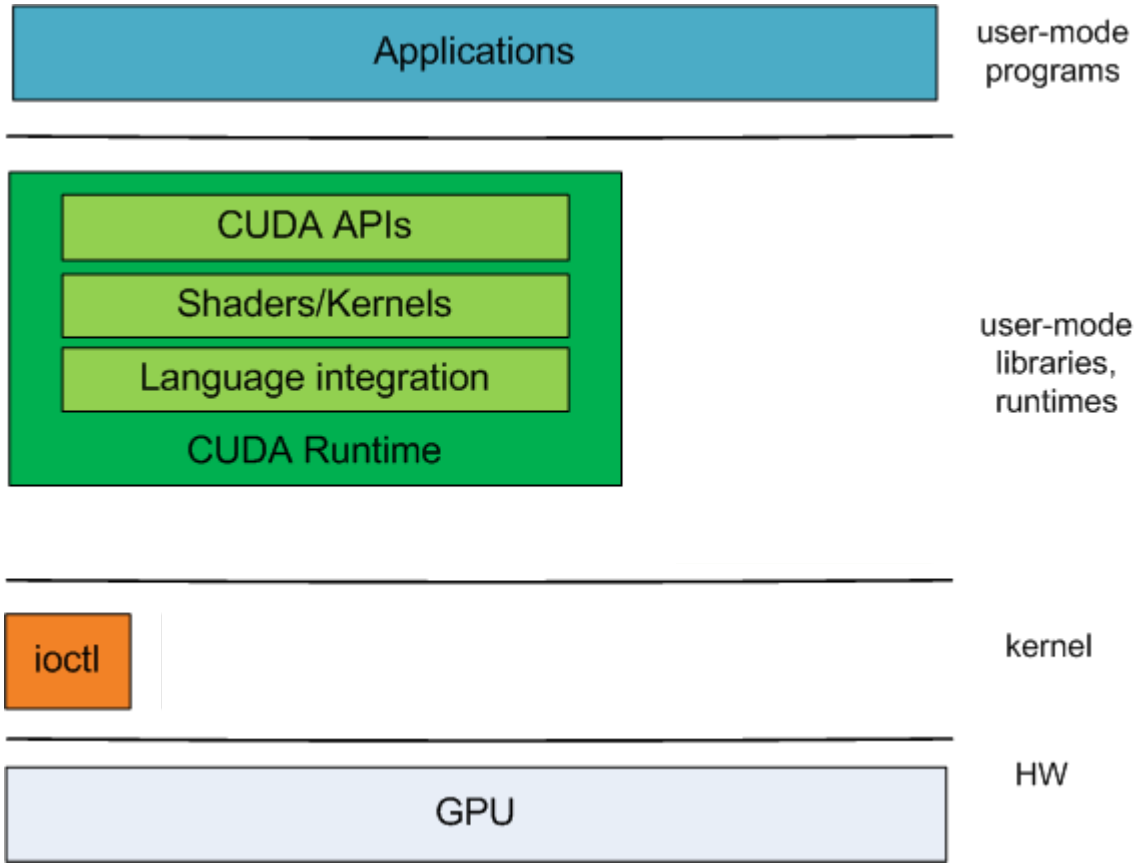
Datablock Action Zone

```
#> capture | xform | filter …
```





Revised technology stack



- 1-1 correspondence between programmer and OS abstractions
- GPU APIs can be built on top of new OS abstractions

Outline

- The case for OS support
- PTask: Dataflow for GPUs
- Evaluation
- Related Work
- Conclusion

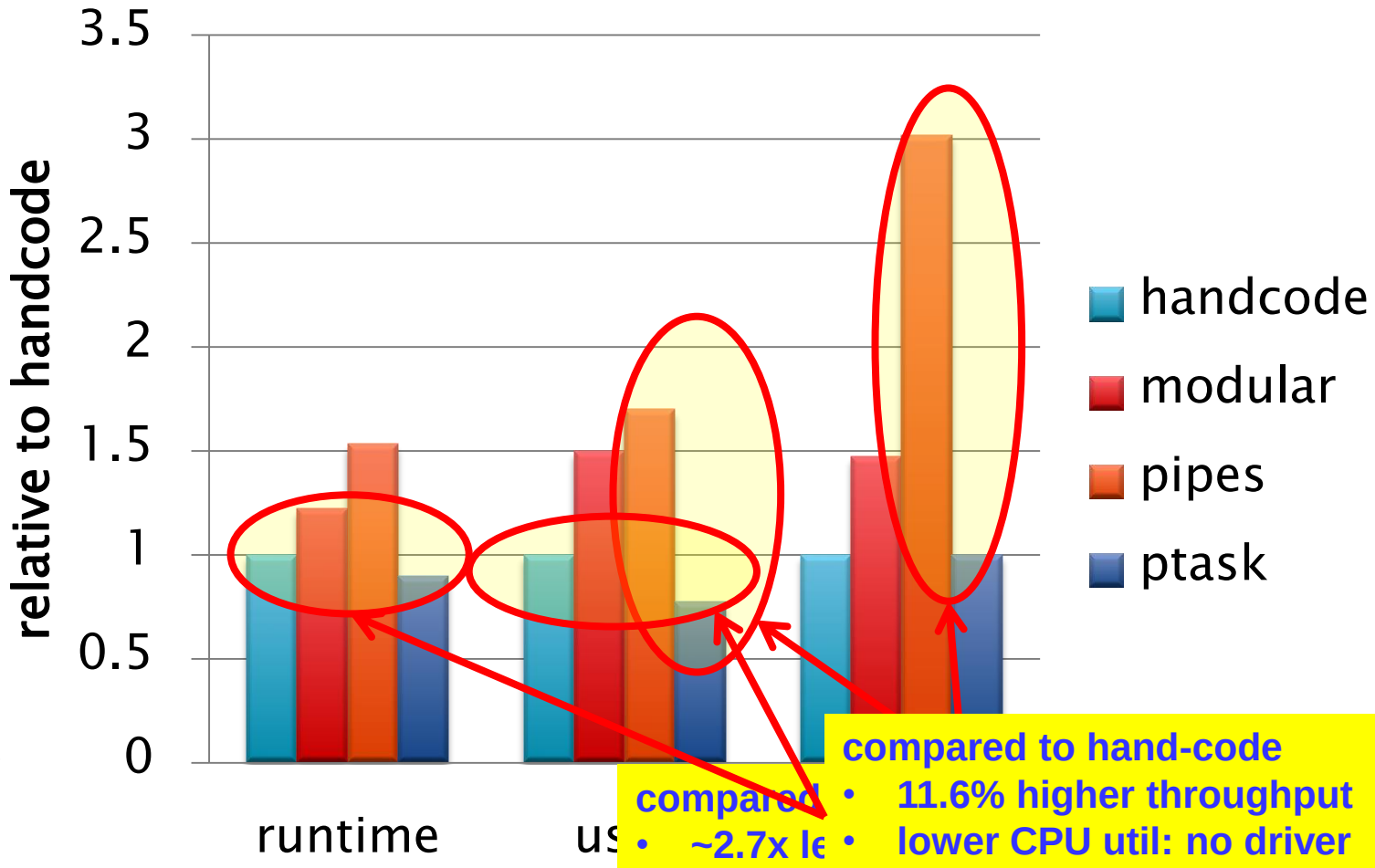
Implementation

- Windows 7
  - Full PTask API implementation
  - Stacked UMDF/KMDF driver
    - Kernel component: mem-mapping, signaling
    - User component: wraps DirectX, CUDA, OpenCL
  - syscalls → DeviceIoControl() calls
- Linux 2.6.33.2
  - Changed OS scheduling to manage GPU
    - GPU accounting added to task_struct

Gestural Interface evaluation

- Windows 7, Core2-Quad, GTX580 (EVGA)
- 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



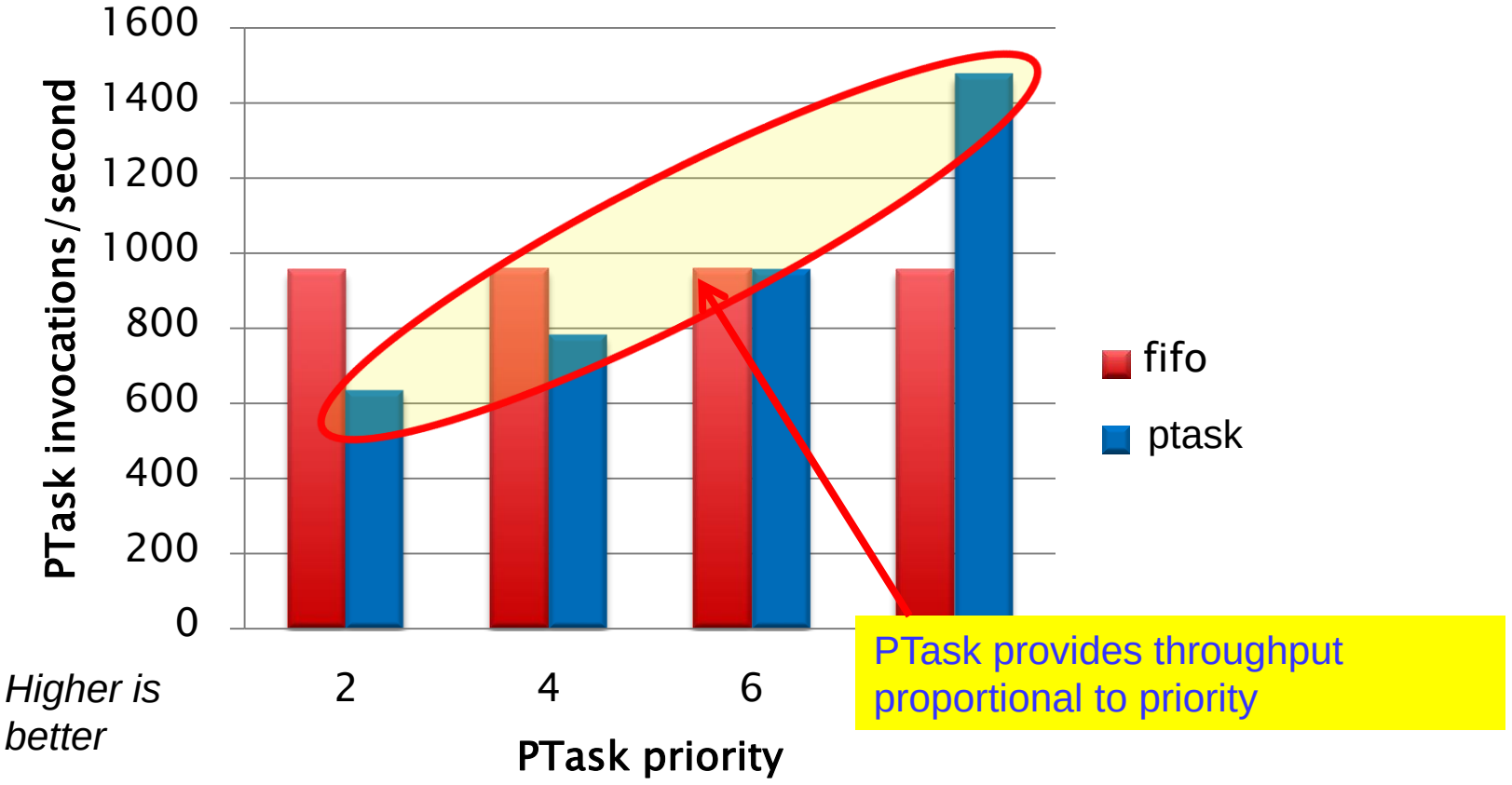
lower is better

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

compared
- **~2.7x le**
- **16x hig**
- **~45% less memory usage**

d 2.66GHz
- GTX580 (EVGA)

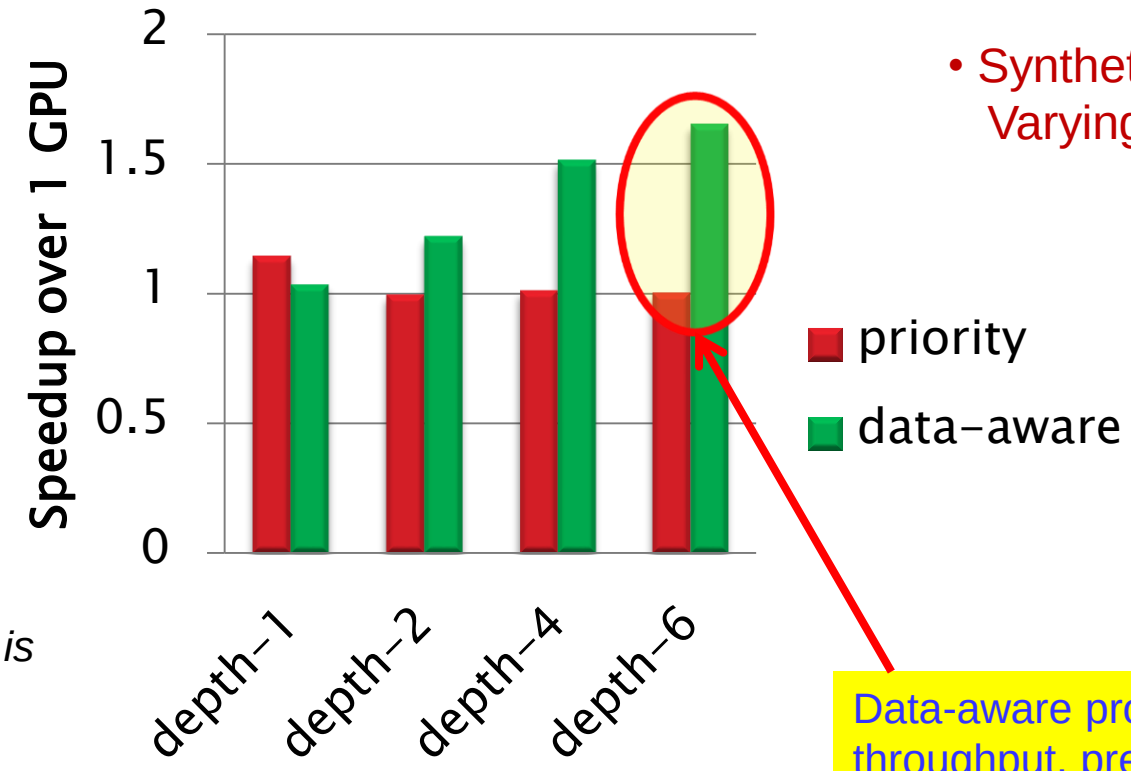
Performance Isolation



- 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)

Multi-GPU Scheduling



Higher is better

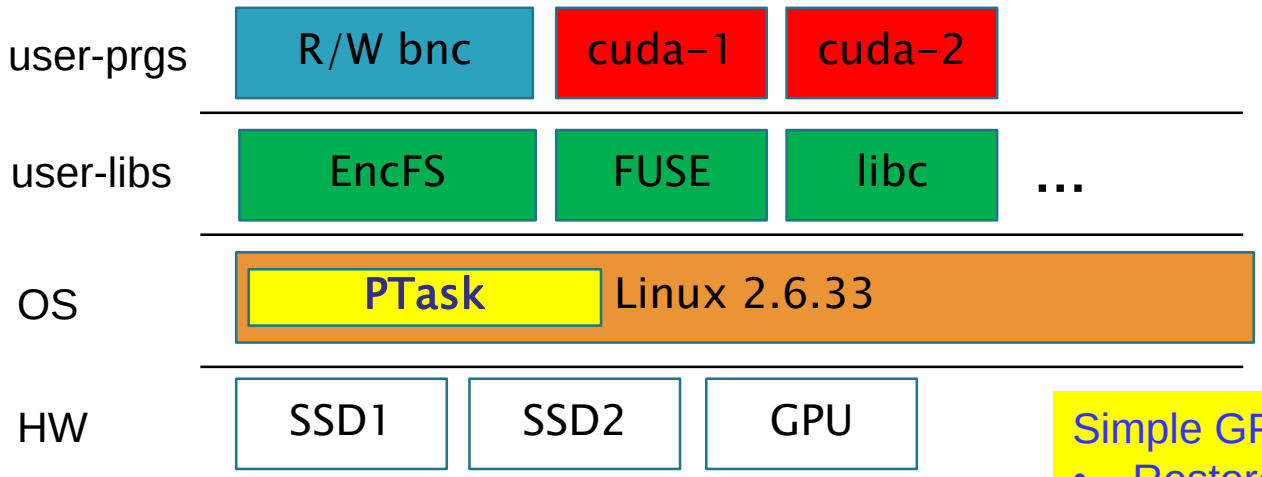
- Synthetic graphs: Varying depths

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)

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

Outline

- The case for OS support
- PTask: Dataflow for GPUs
- Evaluation
- Related Work
- Conclusion

Related Work

- 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, …

Conclusions

- OS 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

Thank you. Questions?