Operating Systems must support GPU abstractions

Chris Rossbach, Microsoft Research
Jon Currey, Microsoft Research
Emmett Witchel, University of Texas at Austin
HotOS 2011

GPU Haiku (apropos 10 min talks)

Lots of GPUs
Must they be so hard to use?
We need dataflow…

GPU Haiku (apropos 10 min talks)

Lots of GPUs
Must they be so hard to use?
We need dataflow…

…support in the OS

Motivation and Agenda

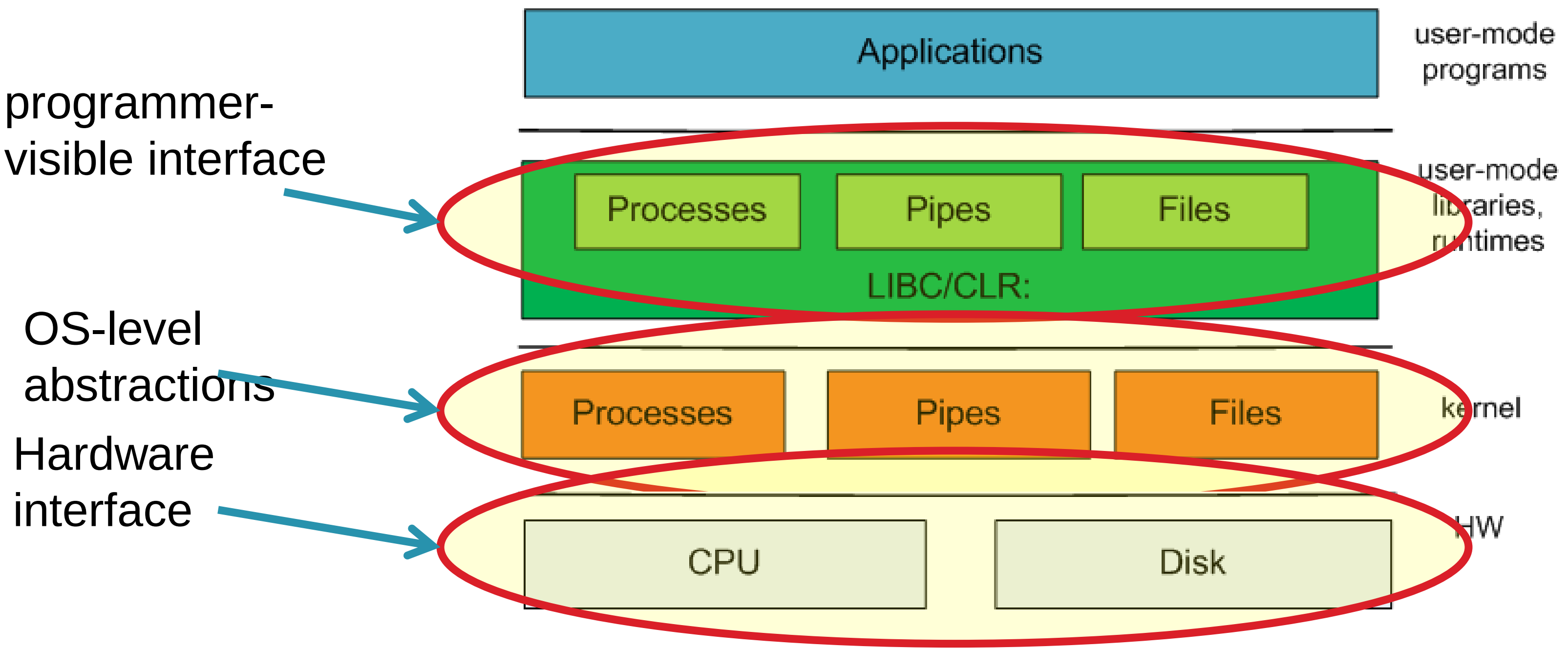
- There are lots of GPUs!
  - ~ more powerful than CPUs
  - Great for Halo <X> and HPC, but little else
  - Underutilized
- GPUs are difficult to program
  - SIMD execution model
  - Cannot access main memory
  - Treated as I/O device by OS

Motivation and Agenda

- There are lots of GPUs!
  - ~ more powerful than
  - Great for Halo <X> a
  - **Underutilized**
- GPUs are difficult to program
  - SIMD execution model
  - Cannot access main memory
  - **Treated as I/O device by OS**

A. These two things are related
B. We need OS abstractions (dataflow)

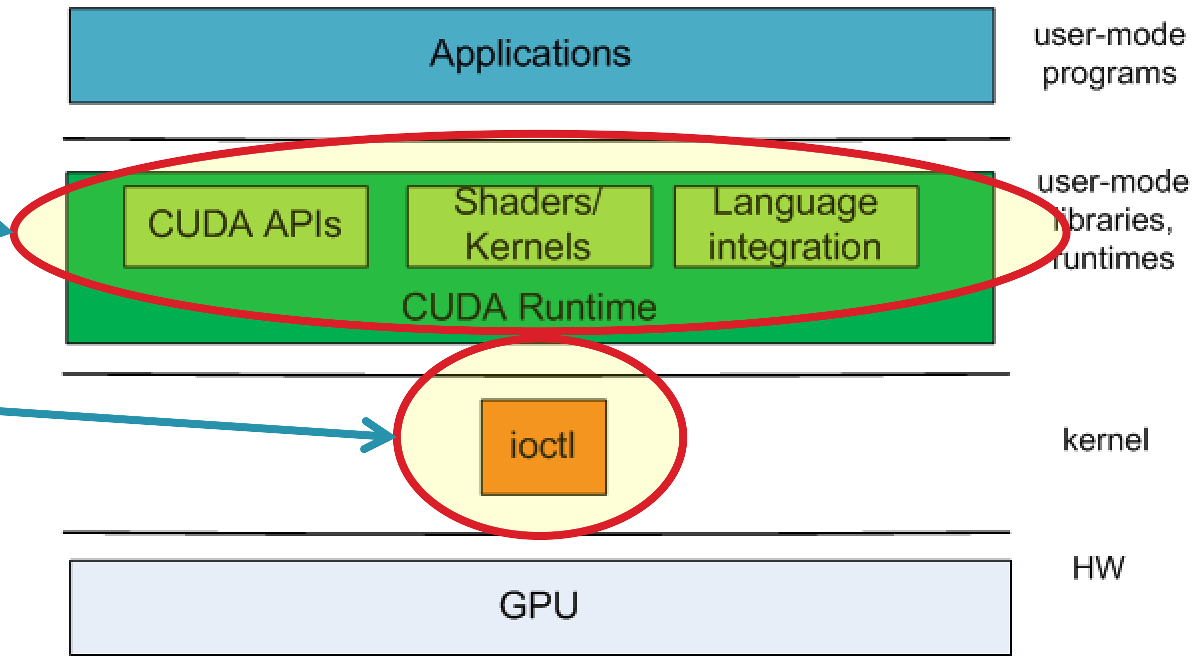
Traditional OS-Level abstractions



GPU Abstractions

programmer-visible interface

1 OS-level abstraction!



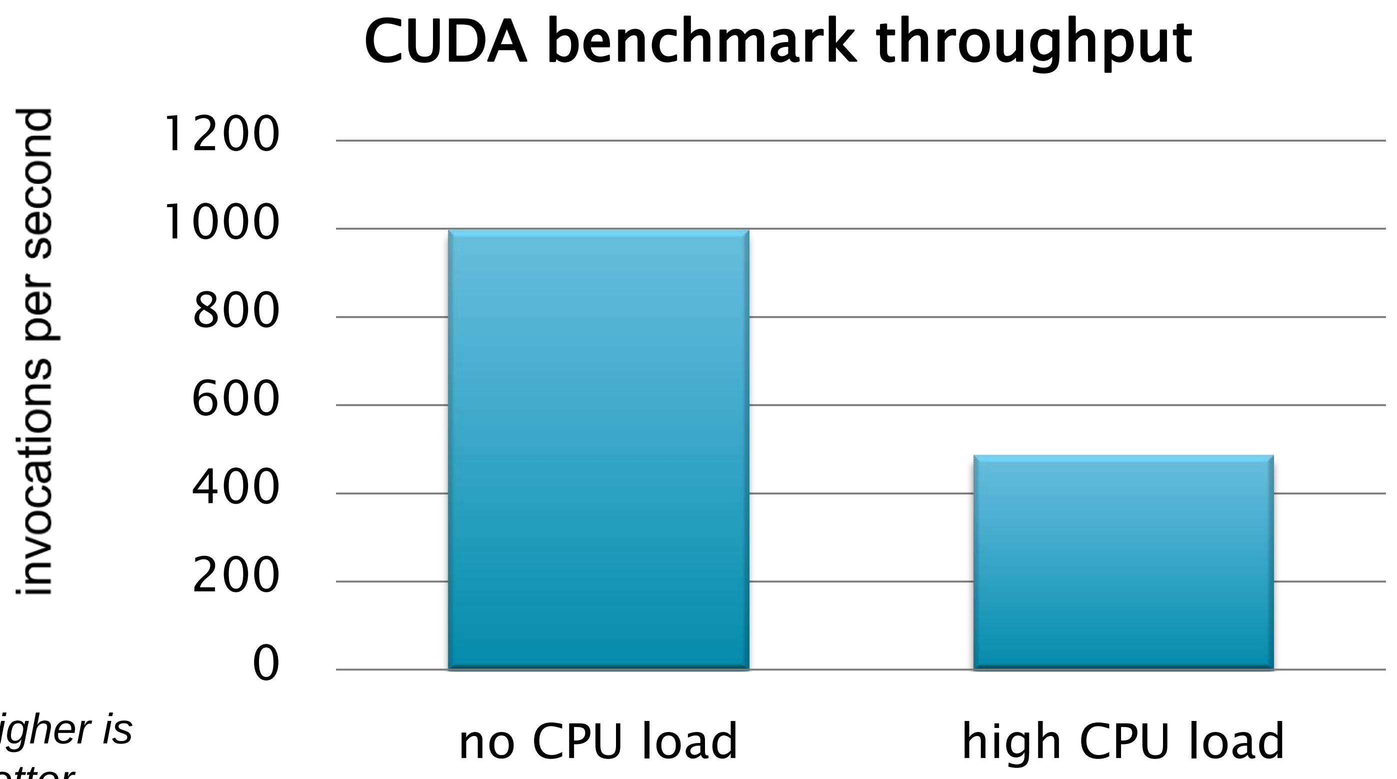
The programmer gets to work with great abstractions…
Why is this a problem?

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

CPU-bound processes hurt GPUs

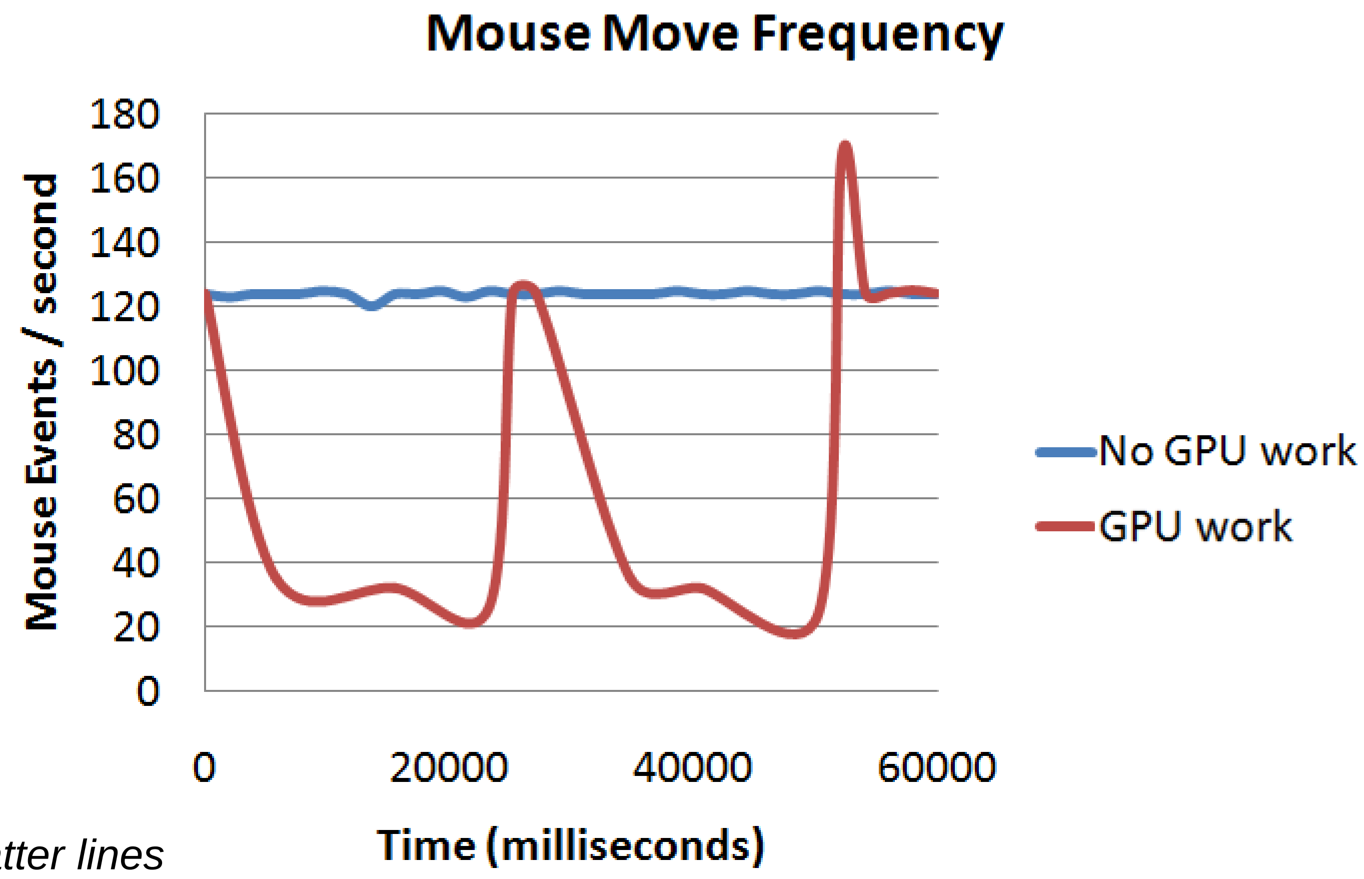


Higher is better

CPU scheduler and GPU scheduler not integrated!

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

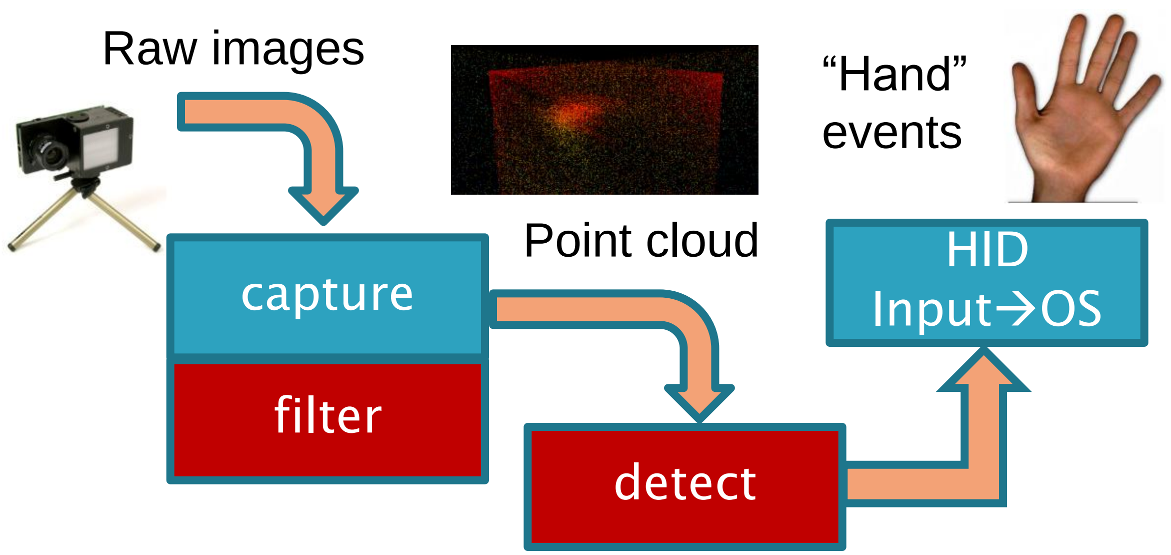
GPU-bound processes hurt CPUs



Flatter lines
Are better

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

Composability: Gestural Interface



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

- Data crossing u/k boundary
- Double-buffering between camera drivers and GPU drivers

Pipes between filter and detect move data to and from GPU even when it's already there

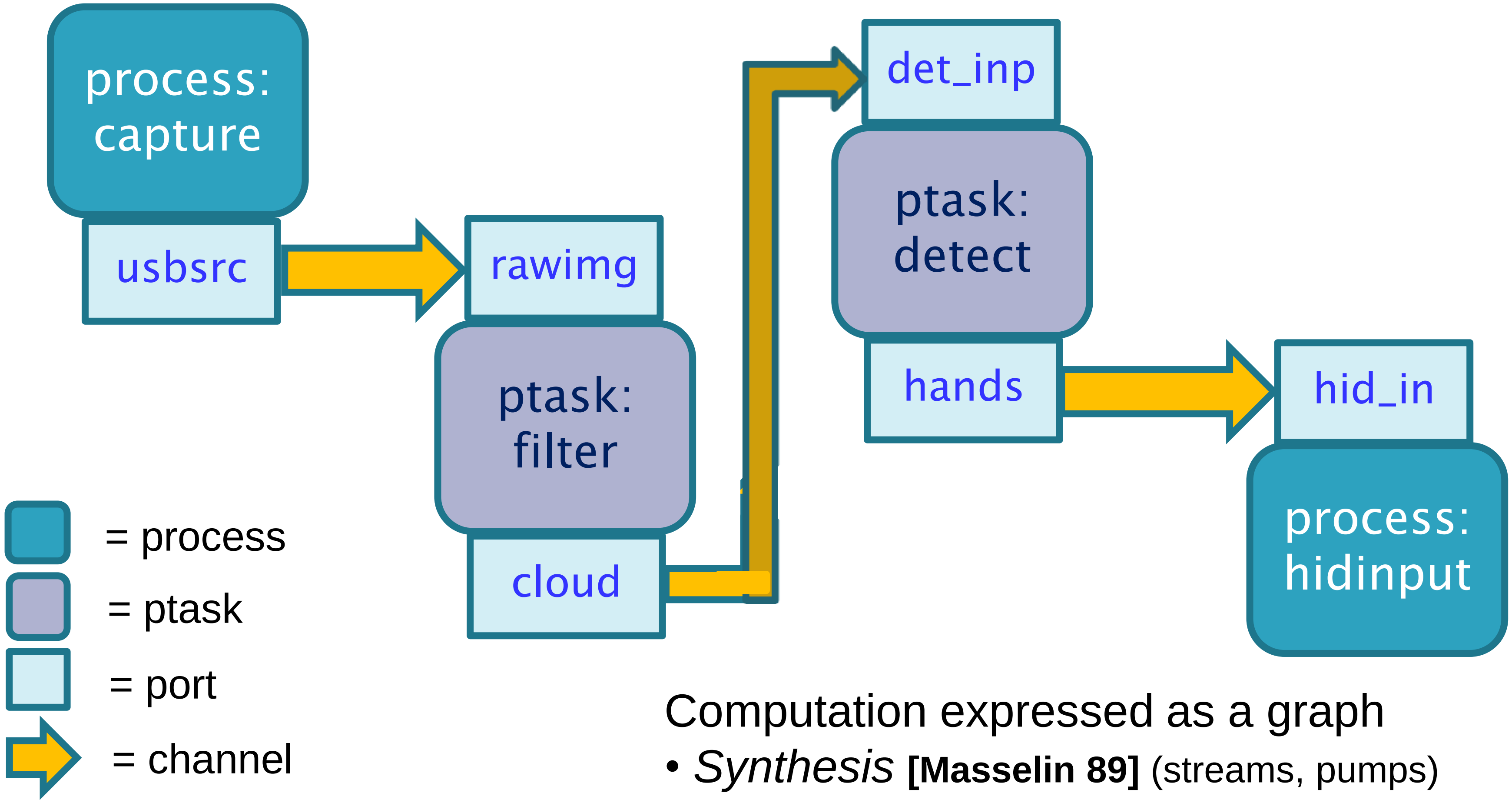
Meaningful GPGPU implies GPUs should be managed like CPUs

- Process API analogues
- IPC API analogues
- Scheduler hint analogues
- Abstractions that enable:
  - Composition
  - Data movement optimization
  - Easier programming

OS abstractions: dataflow!

- **ptask** (parallel task)
  - Have ***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 (e.g. buffer in GPU memory)
- **channels**
  - Similar to pipes
  - Connect arbitrary ports
  - Specialize to eliminate double-buffering

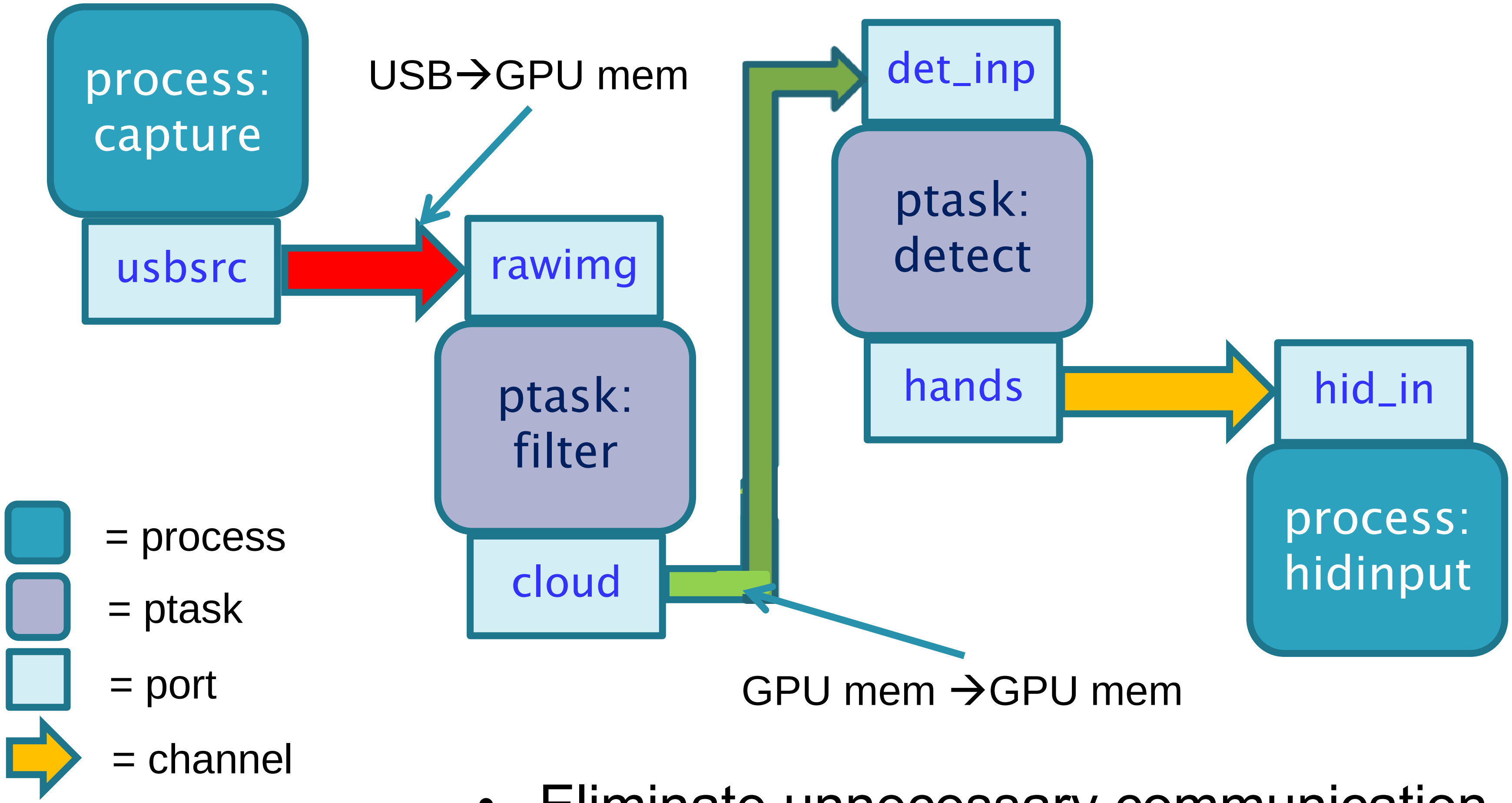
Gestural interface revisited



Computation expressed as a graph

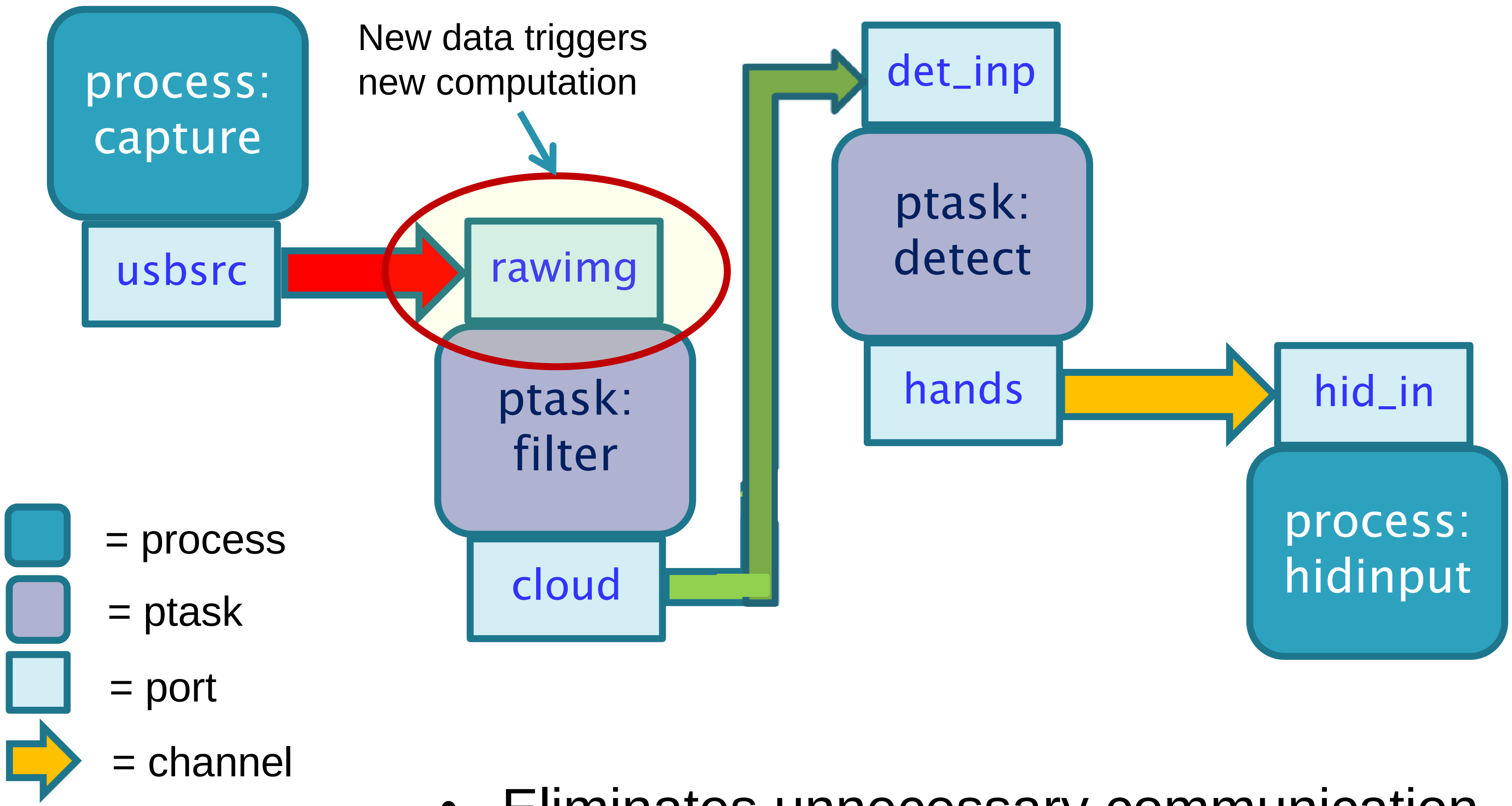
- *Synthesis* **[Masselin 89]** (streams, pumps)
- Dryad **[Isard 07]**
- SteamIt **[Thies 02]**
- Offcodes **[Weinsberg 08]**
- others…

Gestural interface revisited



- Eliminate unnecessary communication…

Gestural interface revisited



- Eliminates unnecessary communication
- Eliminates u/k crossings, computation

Conclusions

- OS must get involved in GPU support
- Current approaches:
  - Require wasteful data movement
  - Inhibit modularity/reuse
  - Cannot guarantee fairness, isolation
- OS-level abstractions are required

Questions?