Chris Rossbach and Jon Currey
Microsoft Research Silicon Valley
NVIDIA GTC 5/17/2012

PTASK + DANDELION:
DATA-FLOW PROGRAMMING SUPPORT FOR HETEROGENEOUS PLATFORMS

Motivation/Overview

- GPU programming is super-important
  - (dissenters: are you at the right conference?)
  - still difficult despite amazing progress
- Technology stacks (at least partly) to blame:
  - OS thinks GPU is I/O device
  - Data movement + algorithms coupled
  - High level language support too low level

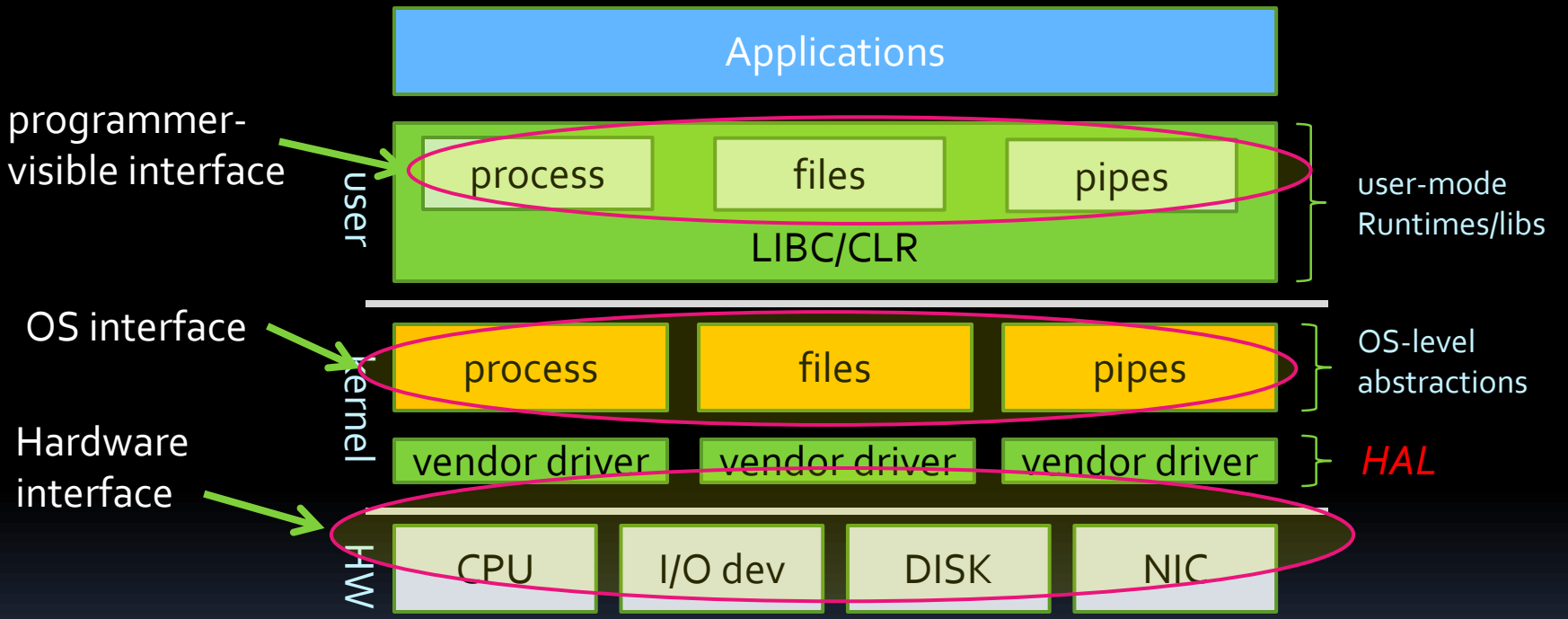
Motivation/Overview

- GPU programming is super-important
  - (dissenters: are you at the right conference?)
  - still difficult despite amazing progress
- GPU technology stacks (at least partly) to blame:
  - *OS thinks GPU is I/O device*
  - *Data movement + algorithms coupled*
  - *High level language support too low level*

Outline

- The case for OS support
- The case for dataflow abstractions
- PTask: better OS support for GPUs
- Dandelion: language+runtime support
- Related and Future Work
- Conclusion

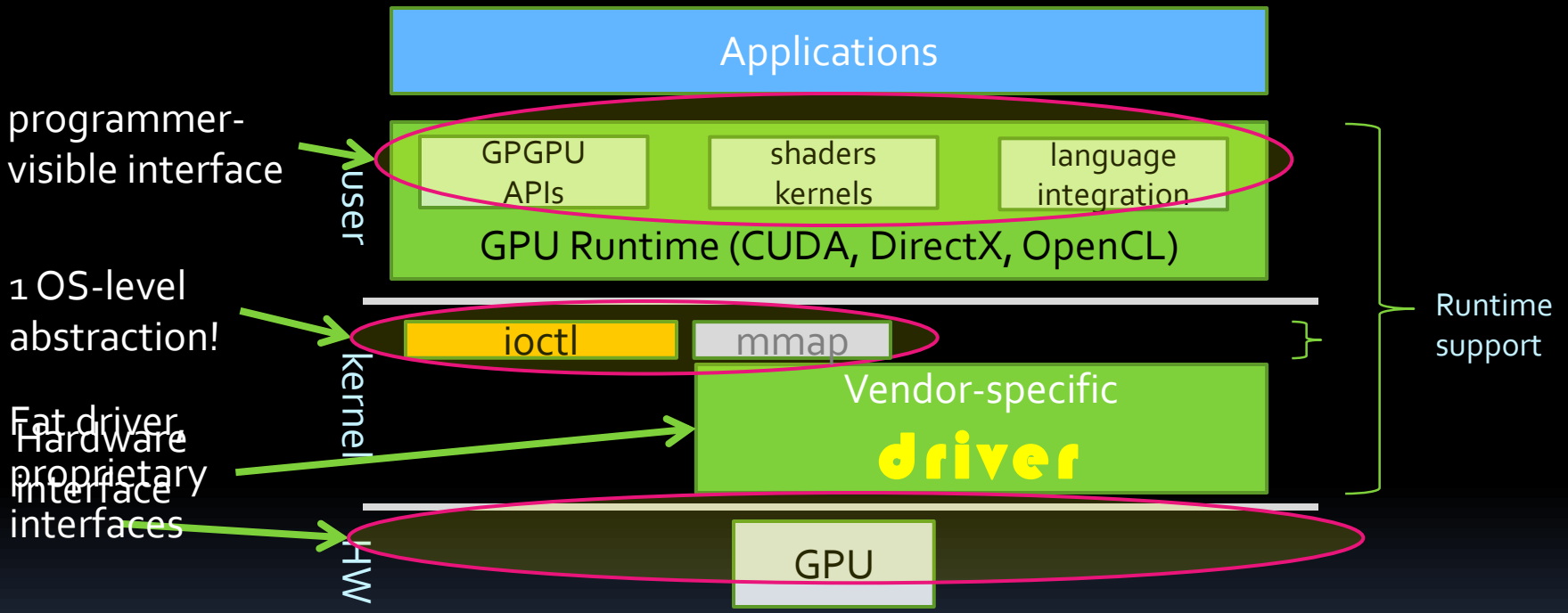
Layered abstractions + OS support



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

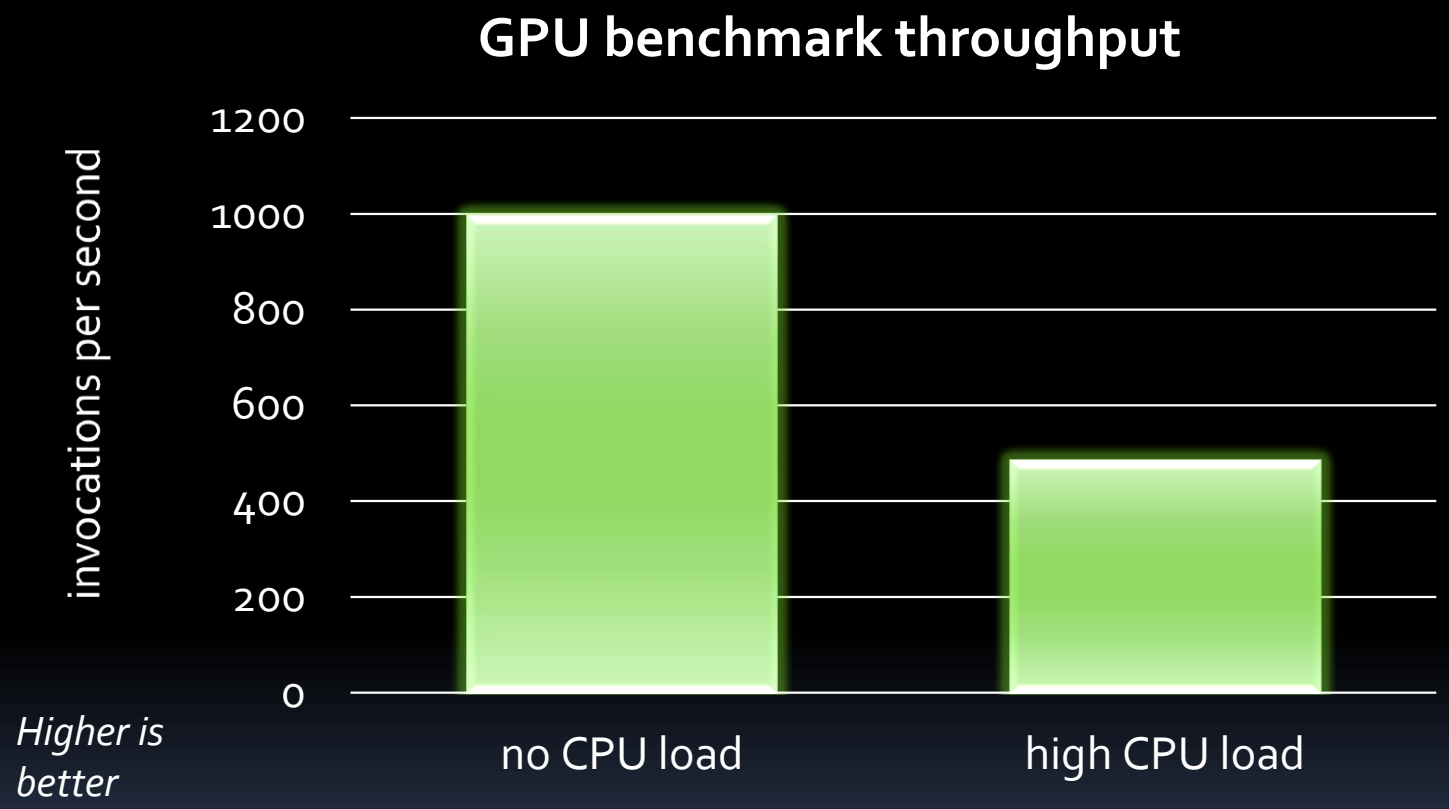
**** Diverse HW support enabled HAL***

GPU abstractions



1. No kernel-facing API
2. Too much runtime support in OS
3. Poor composability

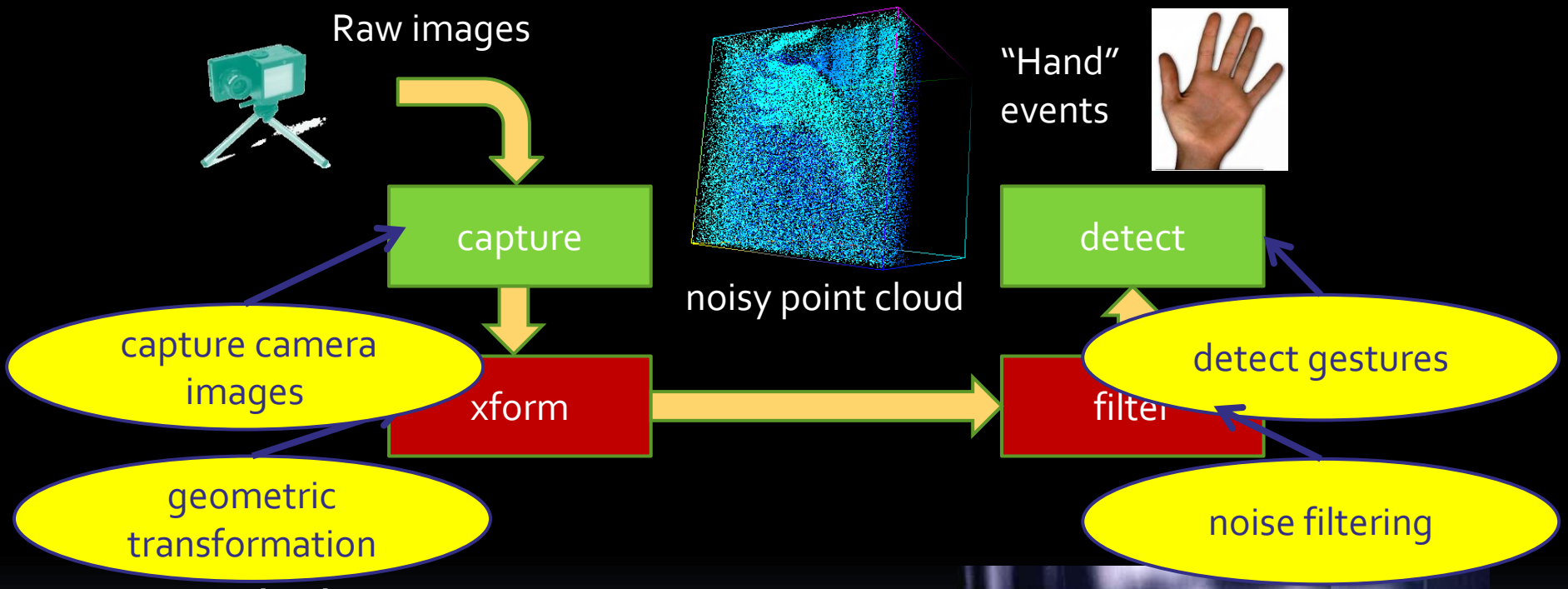
No OS support → No isolation



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

CPU+GPU schedulers not integrated!
…other pathologies abundant

Composition: Gestural Interface



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

What We'd Like To Do

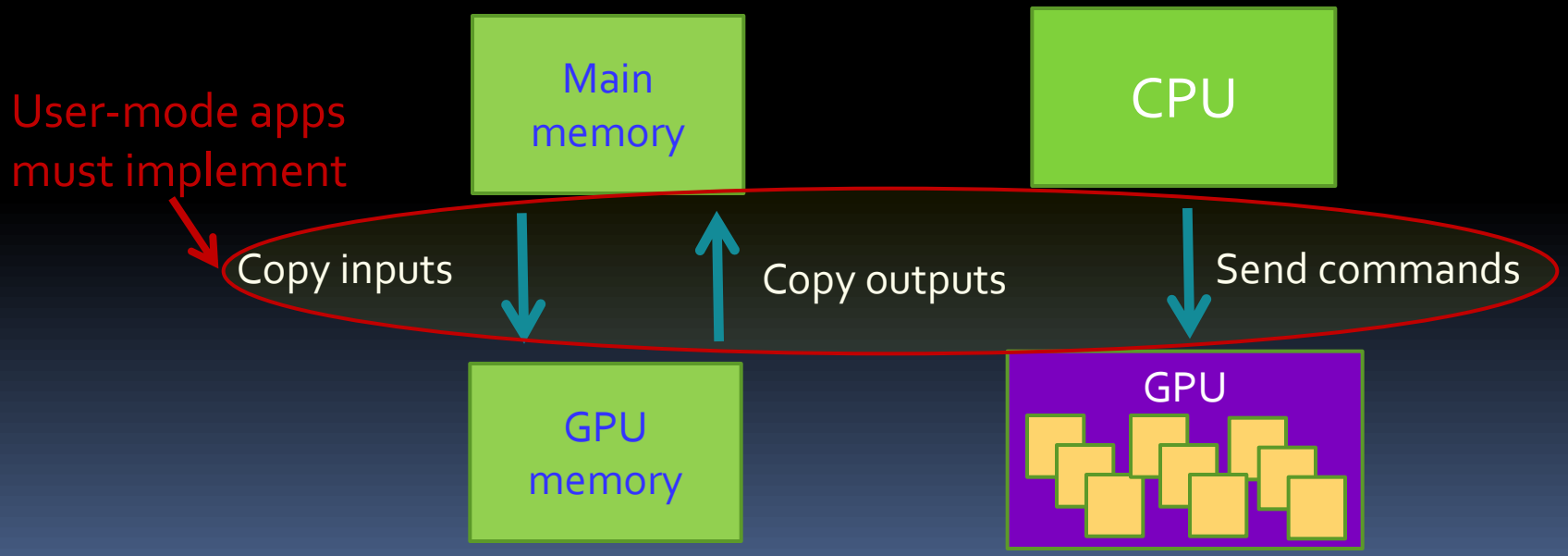
```
#> capture | xform | filter | detect &
```

CPU GPU GPU CPU

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

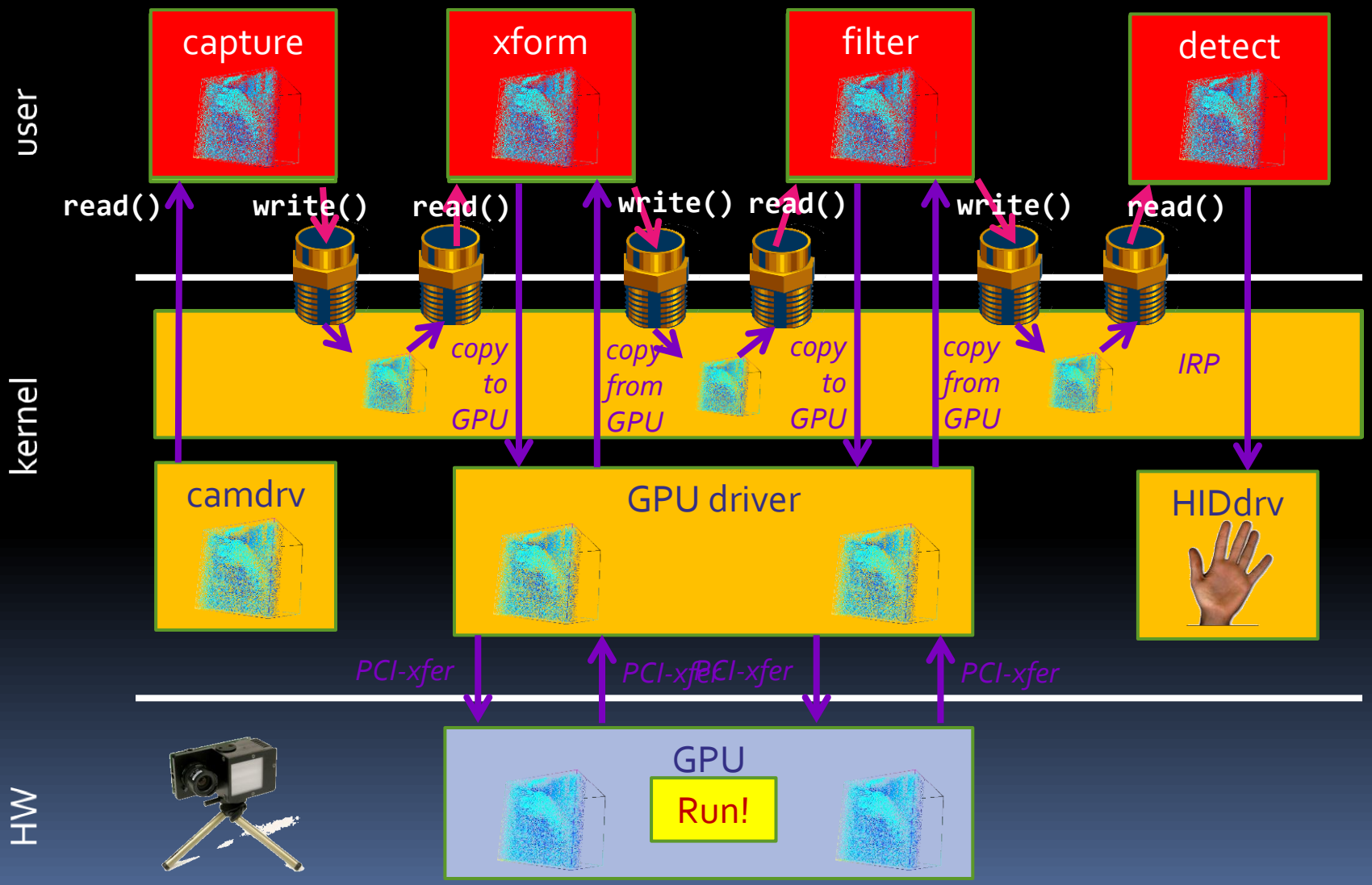
GPU Execution model

- GPUs cannot run OS:
  - different ISA
  - Memories disjoint, or have different coherence guarantees
- Host CPU must "manage" GPU execution
  - Program inputs explicitly transferred/bound at runtime
  - Device buffers pre-allocated



Data migration

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



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;
}
```

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;
}
```

```
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

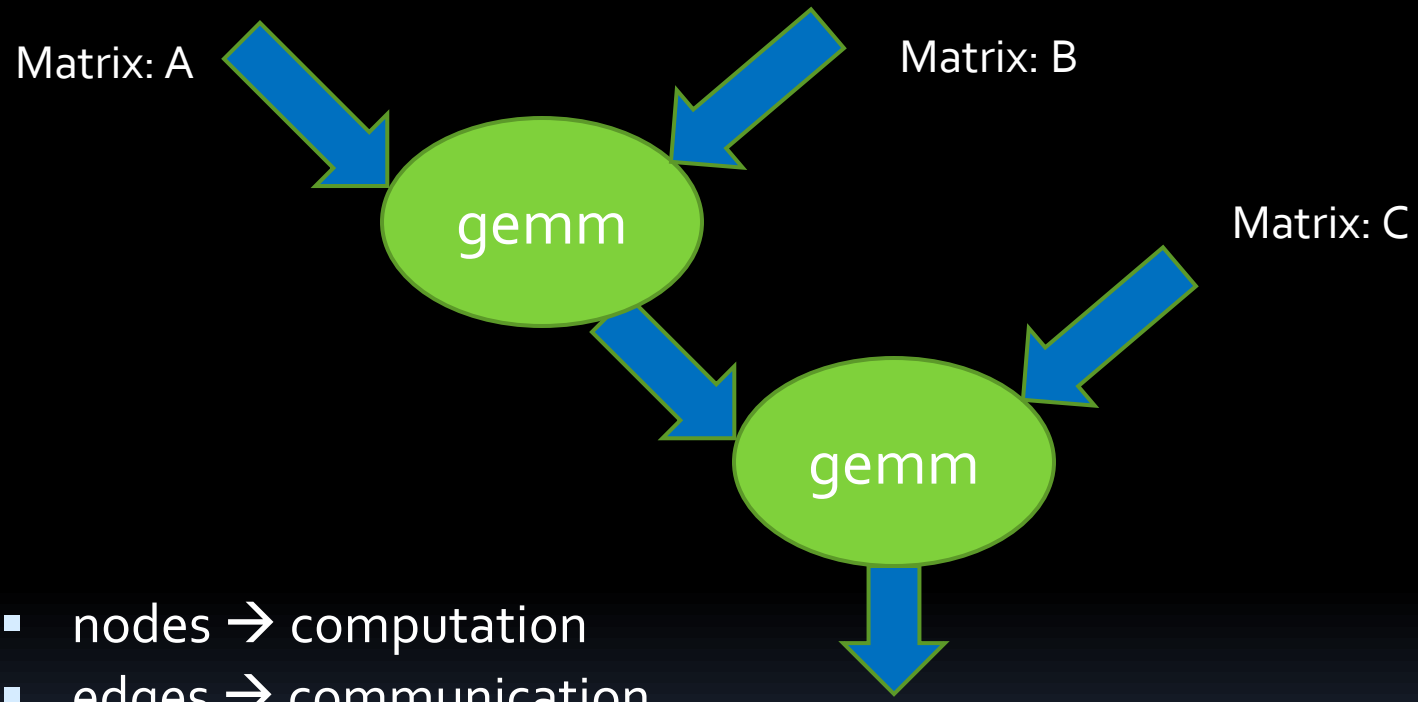
```
Matrix
AxBxC(Matrix A, B, C) {
    Matrix AxB = gemm(A,B);
    Matrix AxBxC = gemm(AxB,C);
    return AxBxC;
}
```

```
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!

We need different abstractions that help get around this…

Dataflow: runtime manages data movement



- nodes → computation
- edges → communication
- leaves flexibility for the runtime
- minimal specification of data movement
- asynchrony is a runtime concern (not programmer concern)

Outline

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

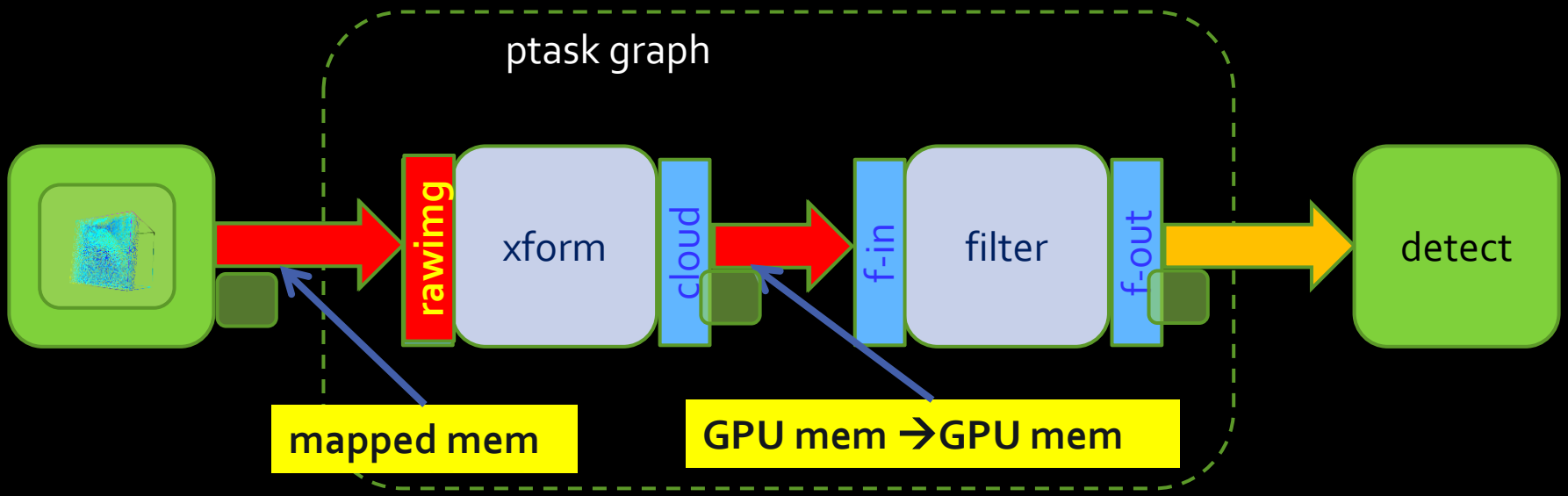
PTask OS abstractions: dataflow!

- **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 dou
- **graph**
  - Directed, connected ptask
- **datablocks**
  - Memory-space transparent buffers

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

PTask Graph: Gestural Interface

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



process (CPU)

ptask (GPU)

port

channel

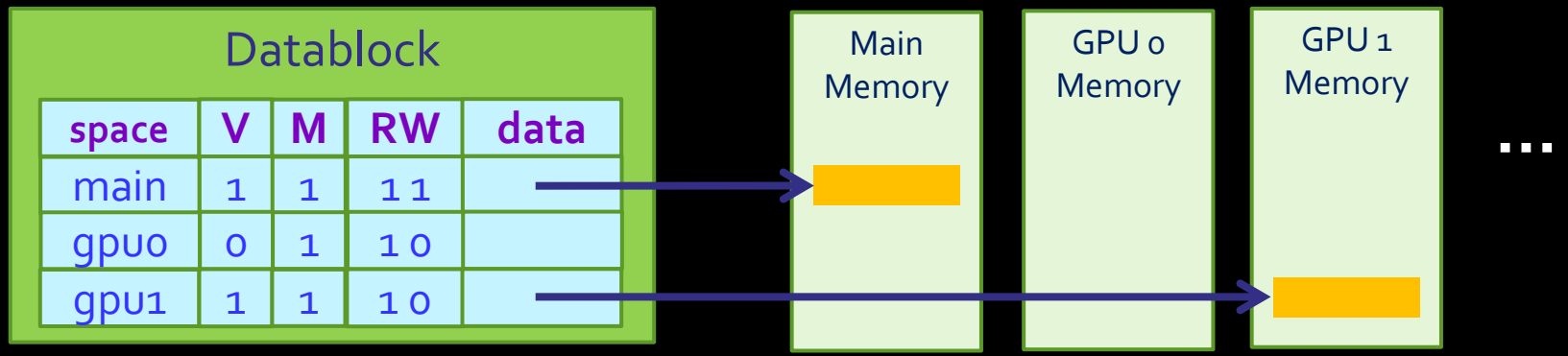
ptask graph

datablock

Optimized data movement

Data arrival triggers computation

Location Transparency: Datablocks

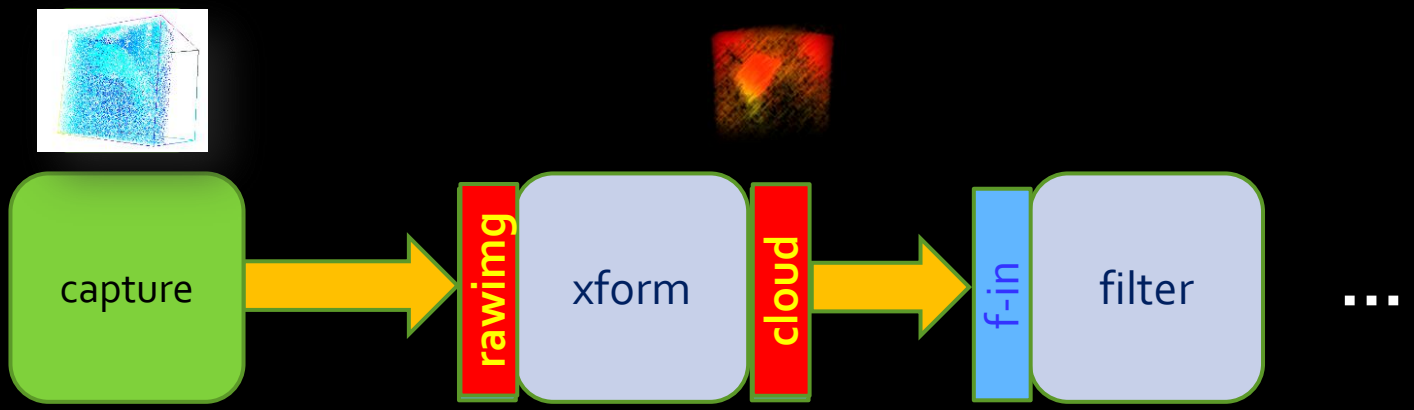


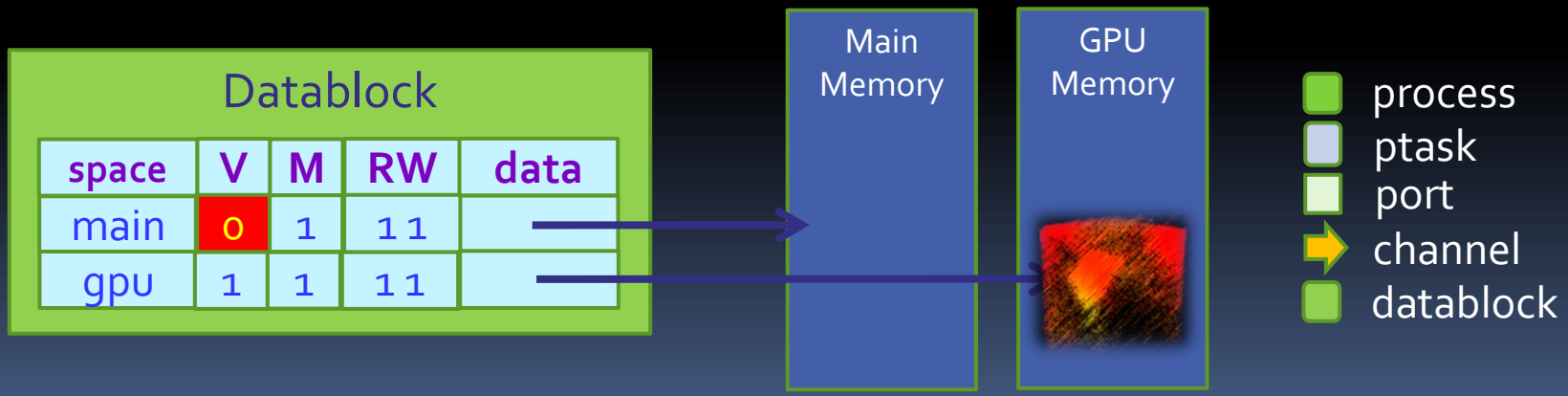
- Logical buffer
  - backed by multiple physical buffers
  - buffers created/updated lazily
  - mem-mapping used to share across process boundaries
- Track buffer validity per memory space
  - writes invalidate other views
- Flags for access control/data pl

Enables OS-mediated IPC ***through GPU memory***

Datablock Action Zone

```
#> capture | xform | filter …
```





Outline

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

Implementation

Windows 7

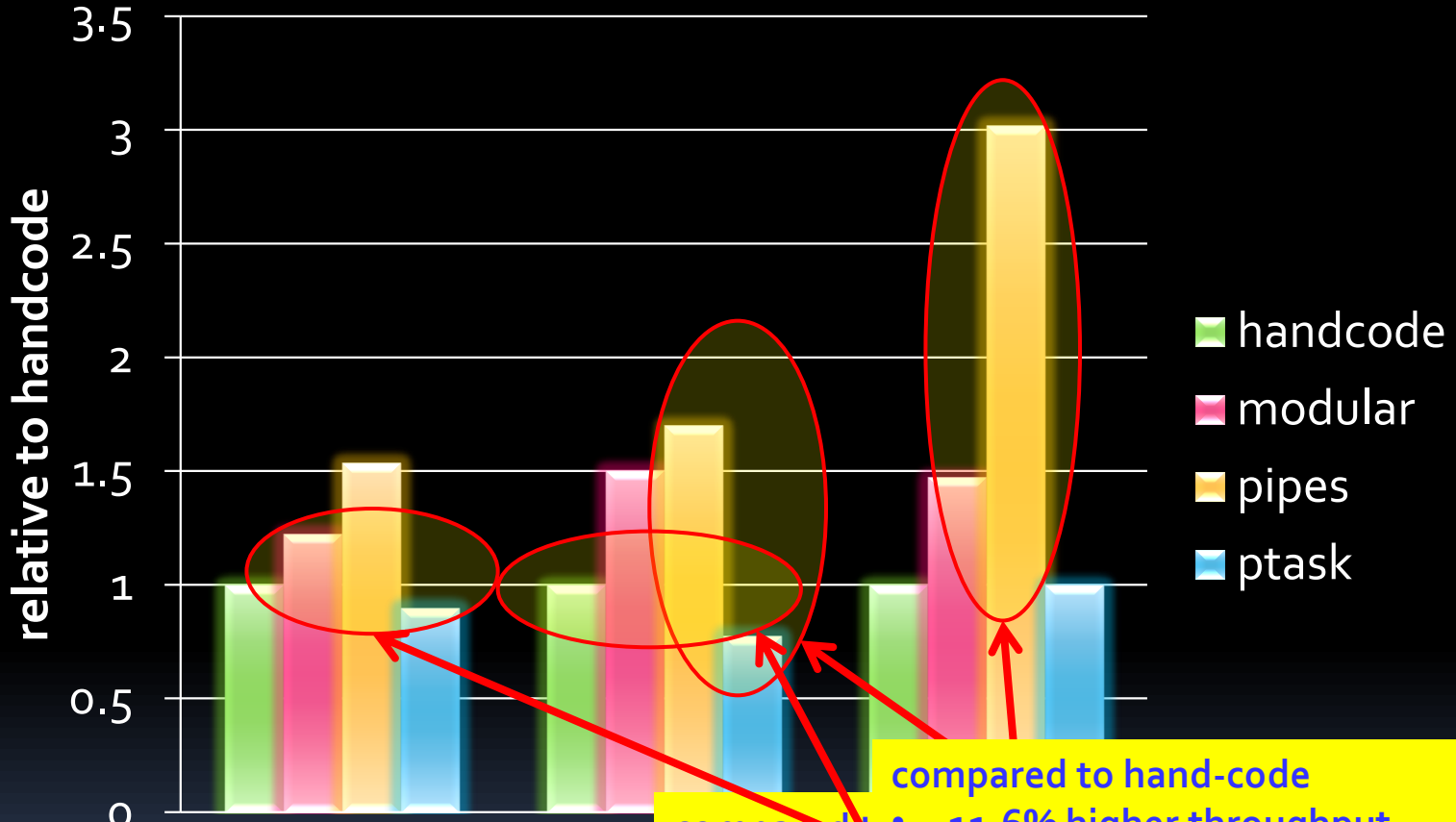
Two PTask API implementations:

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

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

compared t
- ~2.7x le
- 16x high
- ~45% less memory usage

2.66GHz

• GTX580 (EVGA)

Things I didn't show you

- PTask schedules GPUs like CPUs
- PTask provides performance isolation
- Locality-aware scheduling
- PTask transparently uses multiple GPUs
- Proof-of-concept in Linux
- Generality: micro-benchmarks
- …

Outline

- The case for OS support
- The case for dataflow abstractions
- PTask: better OS support for GPUs
- Dandelion: language+runtime support
- Related and Future Work
- Conclusion

Dandelion *Goal*

- A single programming interface for heterogeneous platforms:
  - Clusters comprising:
    - CPUs
    - GPUs
    - FPGAs
    - You name it…
- Programmer: writes sequential code once
- Runtime: partitions, runs in parallel on available resources
- …achievable?

(holy grail)

Dandelion goal

- Offload data-parallel code fragments
- Small cluster of multi-core + multi-GPU
- Starting point:
  - LINQ queries

(a less holy and attractive vessel: usually just as effective, depending on use case)

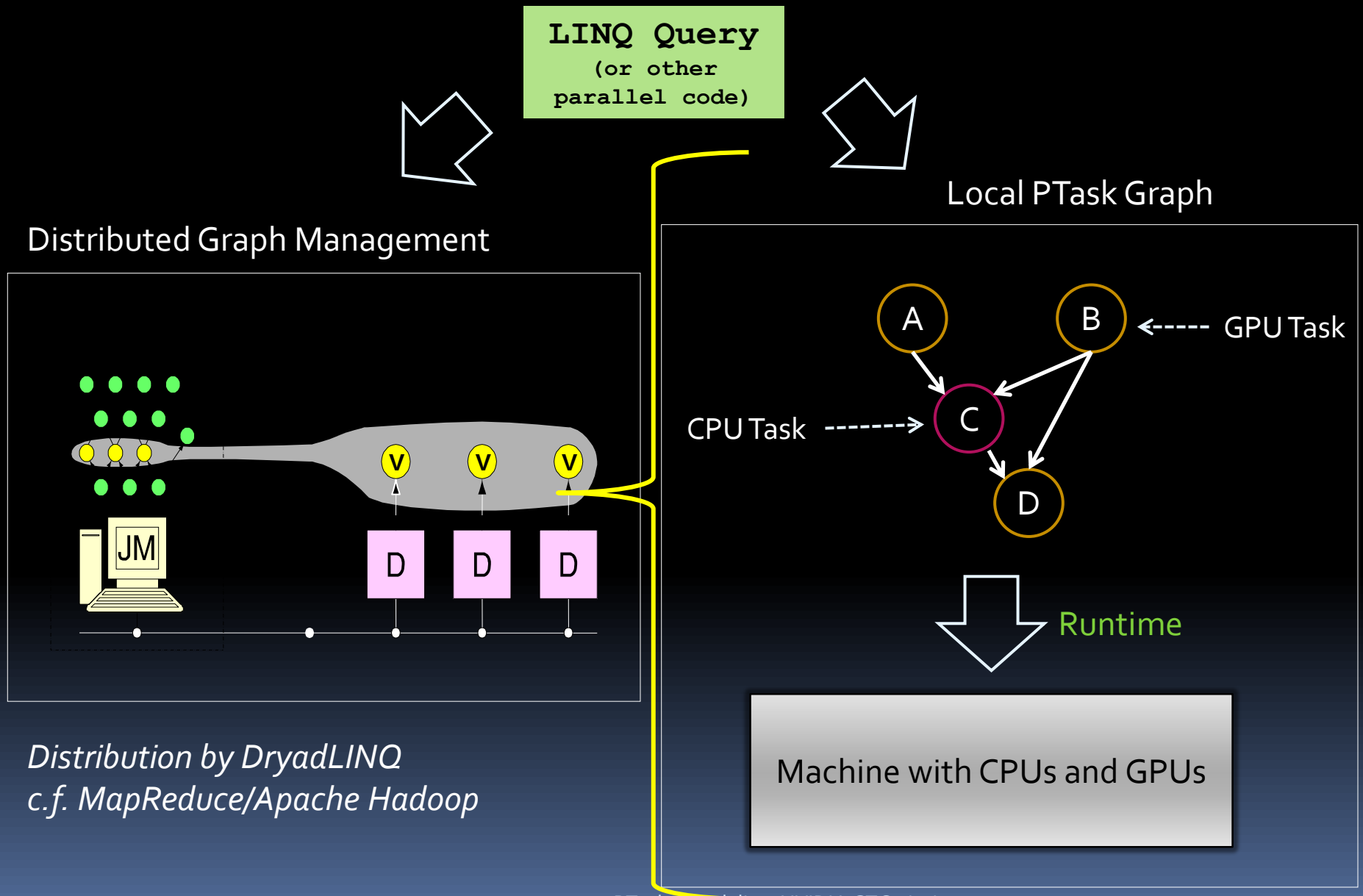
Wait! What's a LINQ Query?

- Language Integrated Query
- Relational operators over objects:

```
var res = collection
      .Where(c => c.hasSomeProperty())
       .Select(c => new {c, quack});
```

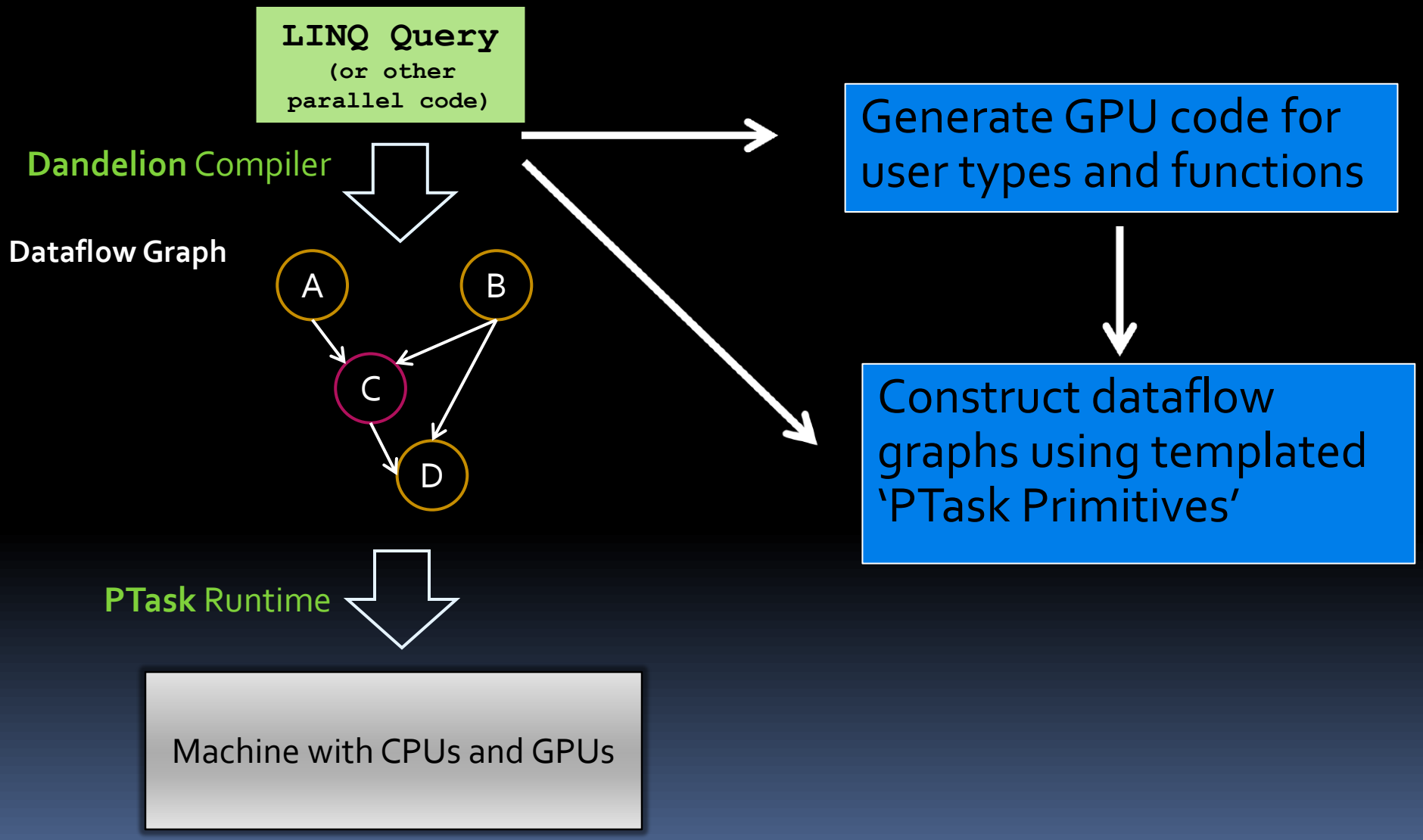
- Why is LINQ important?
  - Expresses many important workloads easily
    - K-Means, PageRank, Sparse Matrix SVD, …
  - LINQ Queries are **data-parallel**
  - Natural fit for data flow

Dandelion Overview



Distribution by DryadLINQ
c.f. MapReduce/Apache Hadoop

Dandelion: Local View



Example: K-Means Clustering

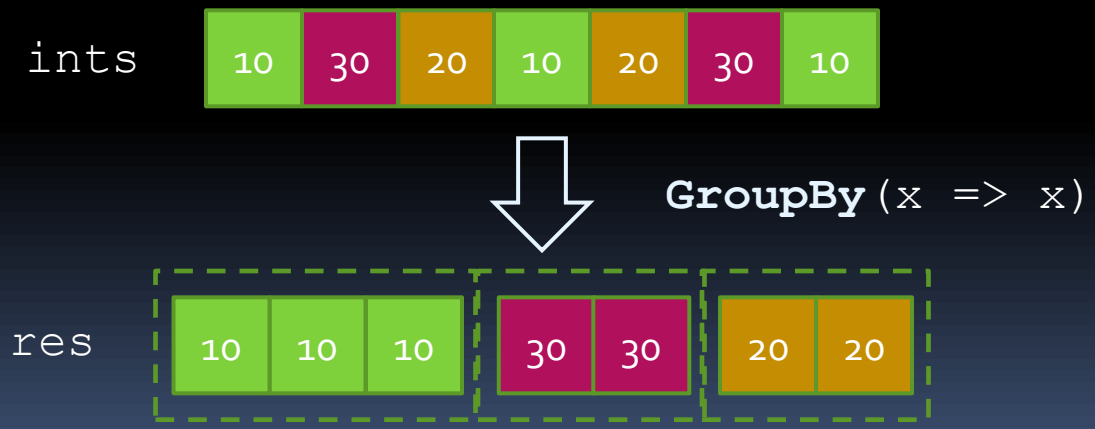
- Partition *n* points into *k* clusters
  - Step 0: Pick k initial cluster centers
  - Step 1: Each point → cluster with nearest center
  - Step 2: Recompute centers (calculate cluster means)
  - Iterate steps 1 & 2 until stable

```
centers = points
   .GroupBy(point => NearestCenter(point, centers))
   .Select(g => g.Aggregate((x, y) => x+y)/g.Count());
```

GroupBy

- Groups an input sequence by key
- Custom function maps input elements → key

```
List<Group<int>> res = ints.GroupBy(x => x);
```



Consider CPU Implementation


```
foreach(T elem in InputSequence)
{
  key   = KeySelector(elem);
  group = GetGroup(key);
  group.Add(elem);
}
```

- Mapping to GPUs is not obvious
  - How to assign work across 1000's of threads?
  - Synchronizing group creation is problematic
  - "Append" is problematic

Parallel GroupBy

Process each input element in parallel

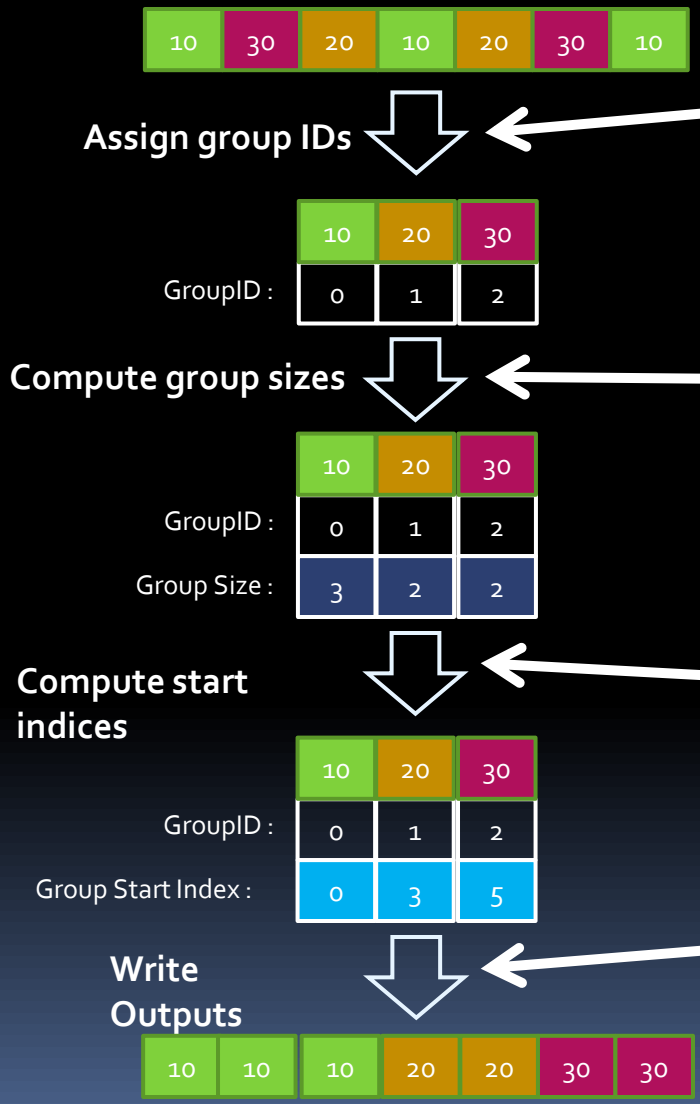
- grouping ~ shuffling
- item output offset = group offset + item number
- … but how to get the group offset?

Start index of each group in the output sequence

Number of elements in each group

Number of groups and integer group IDs

GPU GroupBy: Multiple Passes



Uses a lock-free hash table

Looks up hash table to get group ID

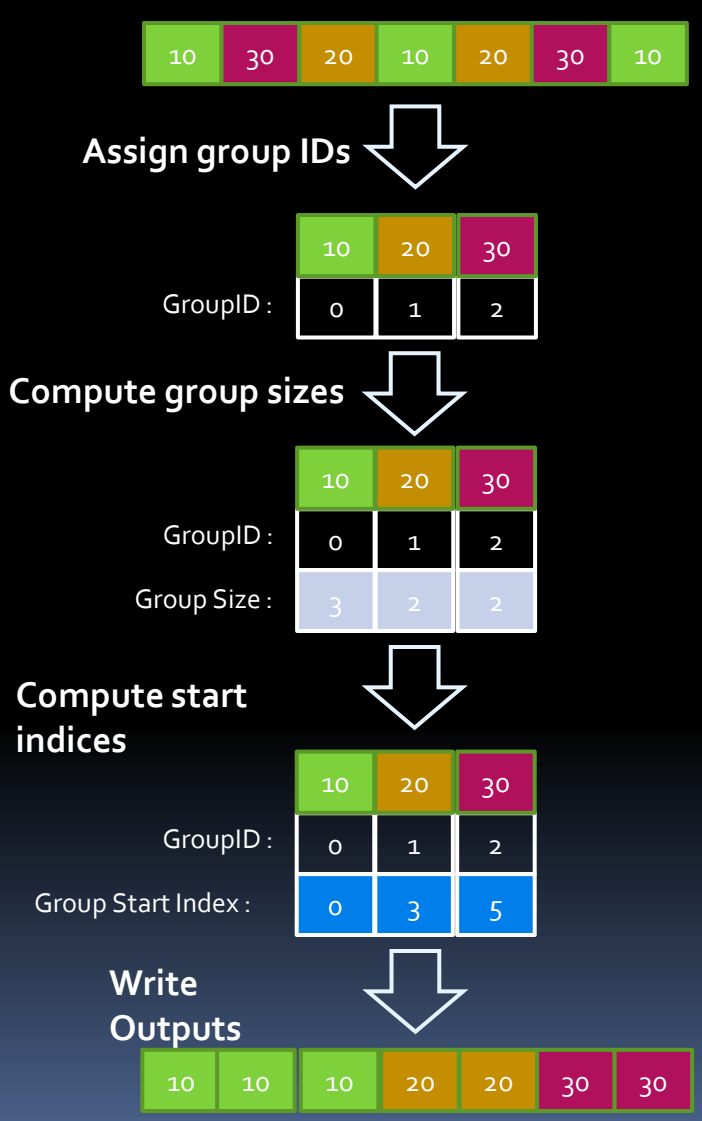
Atomically increments group counters

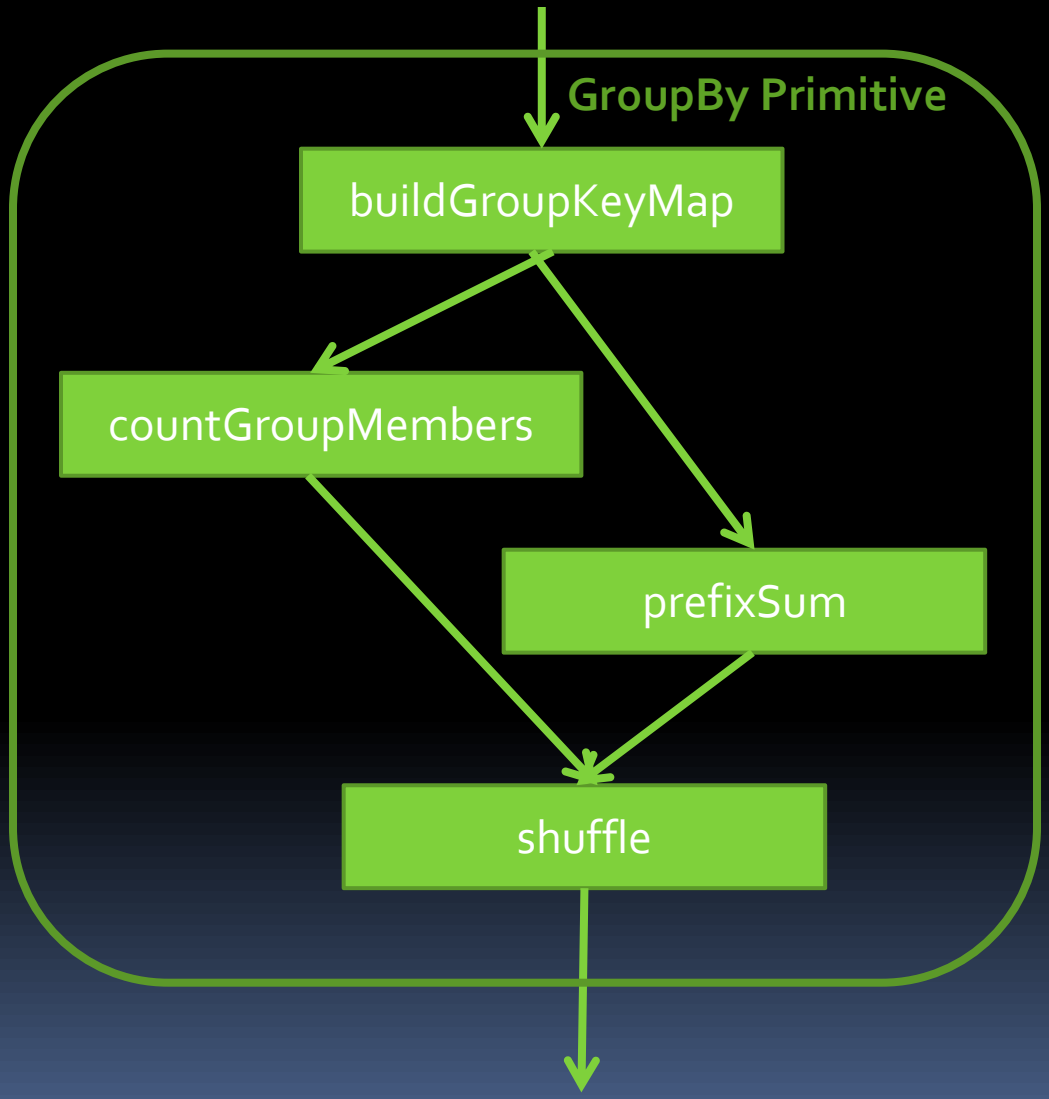
Exclusive prefix sum on group counts

Write output to correct location

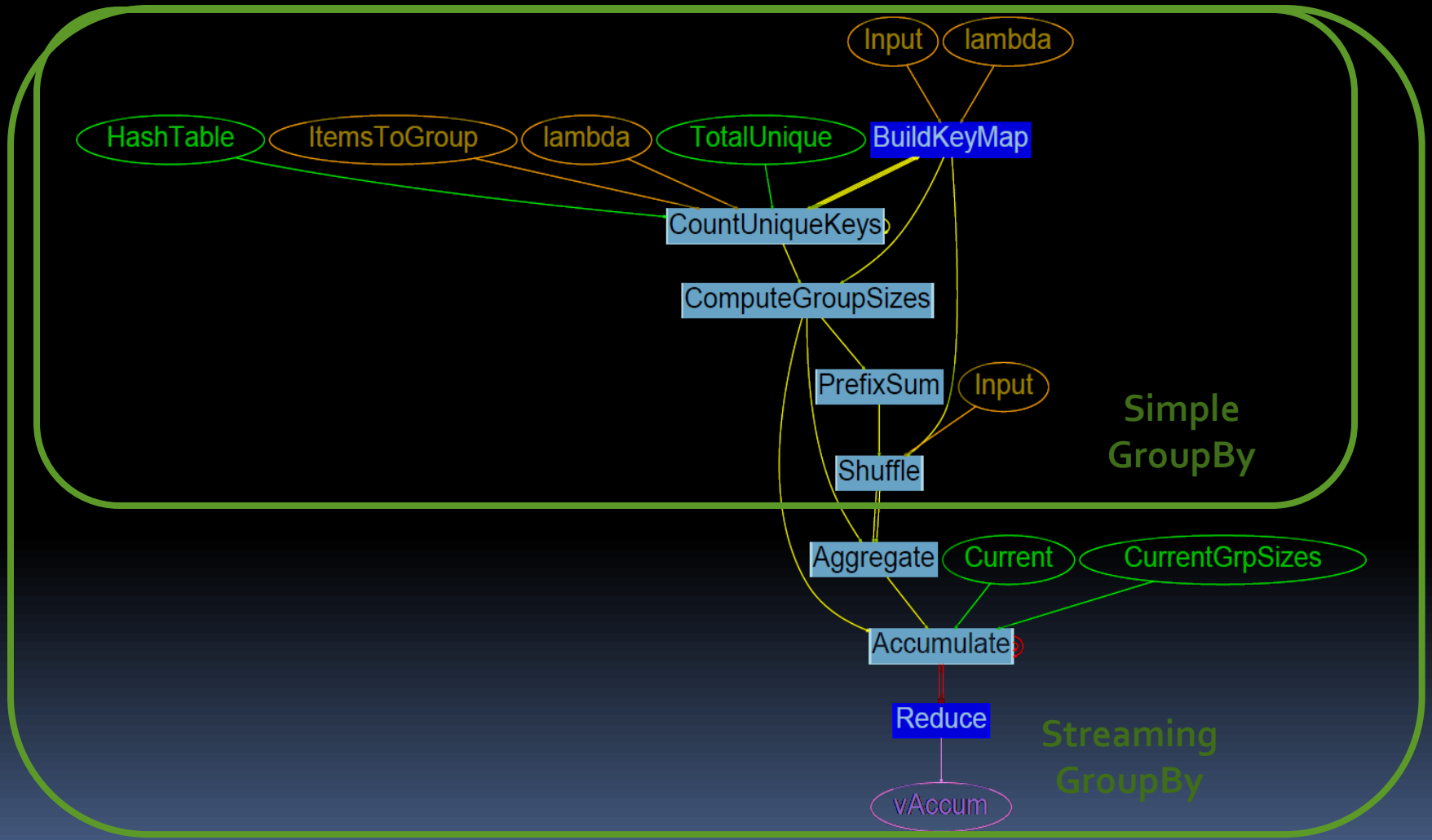
- Uses atomic increment

GroupBy As Composed Primitives





K-Means Dandelion graph



K-Means in C#/LINQ

```
class KMeans {
    int NearestCenter(Vector point, IEnumerable<Vector> centers) {
        int minIndex = 0, curIndex = 0;
        double minValue = Double.MaxValue;
        foreach (Vector center in centers) {
            double curValue = (center - point).Norm2();
            minIndex = (minValue > curValue) ? curIndex : minIndex;
            minValue = (minValue > curValue) ? curValue : minValue;
            curIndex++;
        }
        return minIndex;
    }

    IQueryable<Vector> Iteration(IQueryable<Vector> points,
                                 IQueryable<Vector> centers) {
        return points
            .GroupBy(point => NearestCenter(point, centers))
            .Select(g => g.Aggregate((x, y) => x + y) / g.Count());
    }
}
```

K-Means in Dandelion

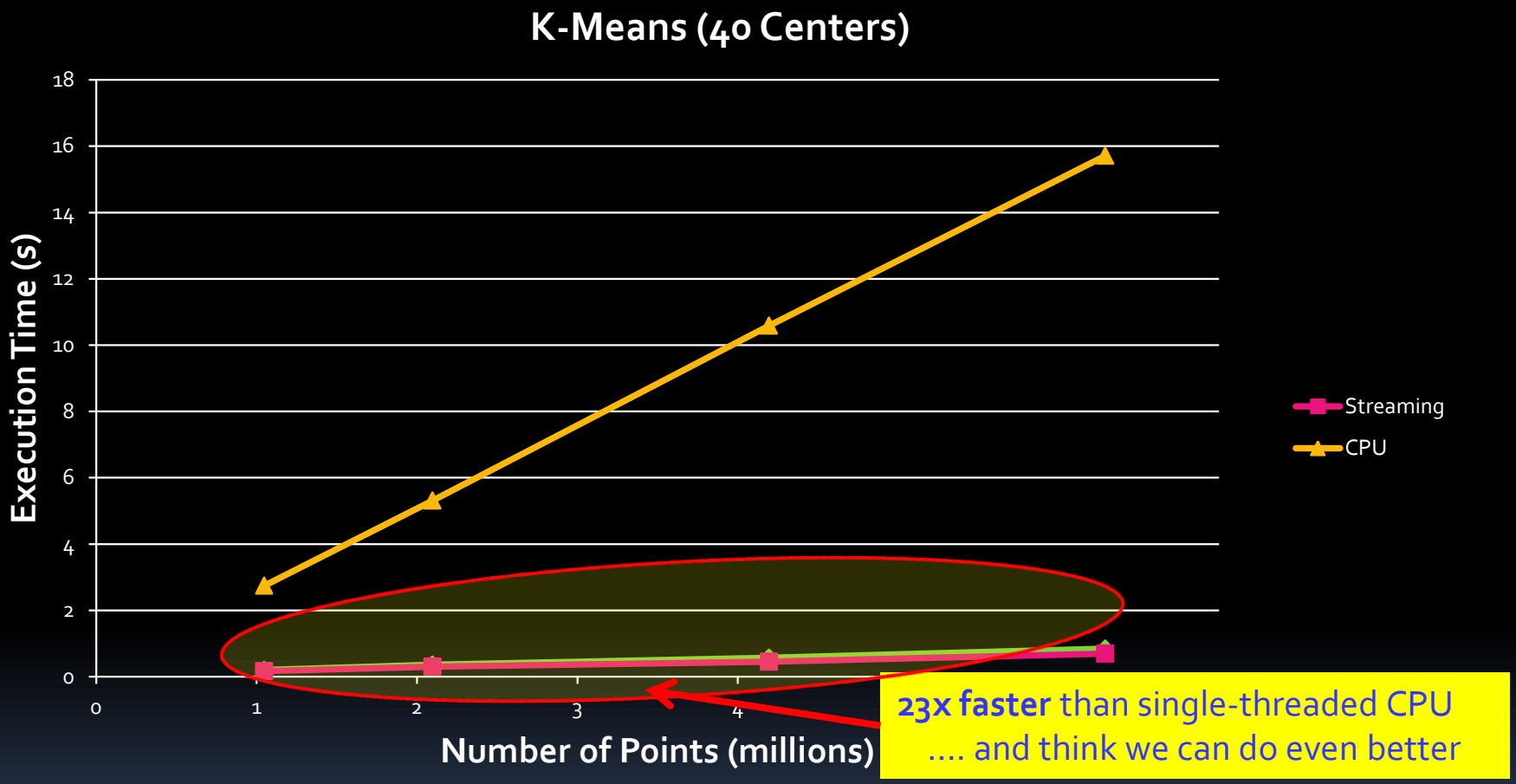
```
class KMeans {
    int NearestCenter(Vector point, IEnumerable<Vector> centers) {
        int minIndex = 0, curIndex = 0;
        double minValue = Double.MaxValue;
        foreach (Vector center in centers) {
            double curValue = (center - point).Norm2();
            minIndex = (minValue > curValue) ? curIndex : minIndex;
            minValue = (minValue > curValue) ? curValue : minValue;
            curIndex++;
        }
        return minIndex;
    }

    IQueryable<Vector> Iteration(IQu
                                 IQu
        return points
            .GroupBy(point => NearestCenter(point, centers))
            .Select(g => g.Aggregate((x, y) => x + y) / g.Count());
    }
}
```

This is kind of a big deal!

- managed sequential code
- running on GPUs
- zero programmer effort

K-Means Performance



- 4-core Intel Xeon W3550 @ 3.07 GHz, 6GB RAM
- NVIDIA GTX580 GPU with 1.5GB GDDR
- PTask on NVIDIA CUDA version 4.0

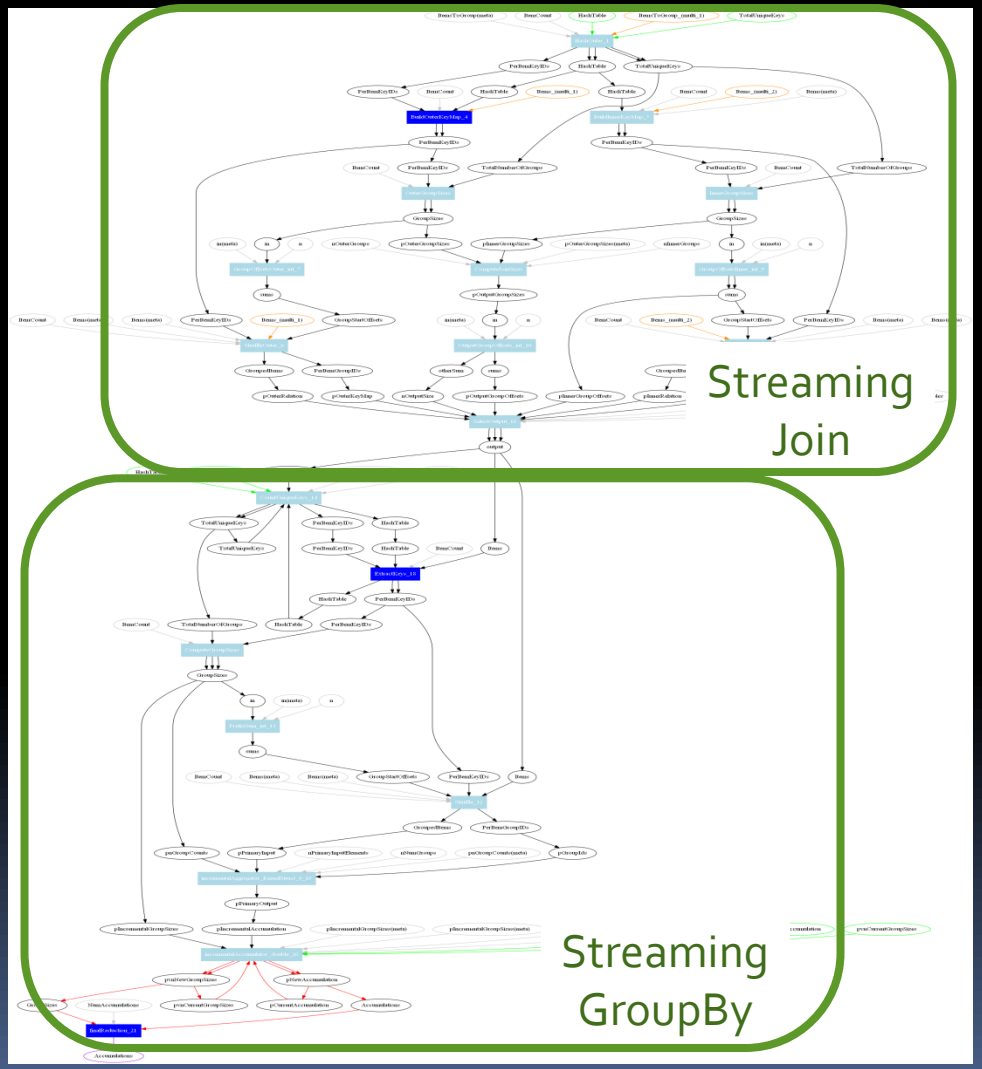
PageRank in Dandelion

```
public struct Rank {
   public  UInt64 PageID;
   public UInt64 PageRank;
...
}

public struct Page {
   public  UInt64 PageID;
   public UInt64 LinkedPageID;
   int numLinkedPages;
...
}

// Join Pages with Ranks, and disperse updates
var partialRanks =
    from rank in ranks
    join page in pages
       on rank.PageID equals page.PageID
    select new Rank(
       page.LinkedPageID,
       rank.PageRank / page.NumLinks);

// Re-accumulate Ranks
var newRanks =
   from partialRank in partialRanks
   group partialRank.PageRank
     by partialRank.PageID into g
   select new Rank(g.Key, g.Sum());
```



Status

- Dandelion is a research prototype
  - Working end to end … now add more workloads
  - Current results promising … but incomplete
- Many areas of research remain unexplored
  - Optimal work partitioning and scheduling
  - When to expose/hide architectural features
  - We *will* need programmer hints sometimes. When?

Outline

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

Related Work

- Prior work : regular (scientific, HPC) apps
  - Accelerator : Array computations **[ASPLOS '06]**
  - Amp/C++
  - 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
  - enables more flexible technology stack
- Proof of concept: LINQ on GPUs
- Future work
  - Automatic task partitioning across CPU and GPU
  - Finding *best* parallelization for a given LINQ query

Thank you! Questions?