Cosh

Clear OS data sharing in an incoherent world

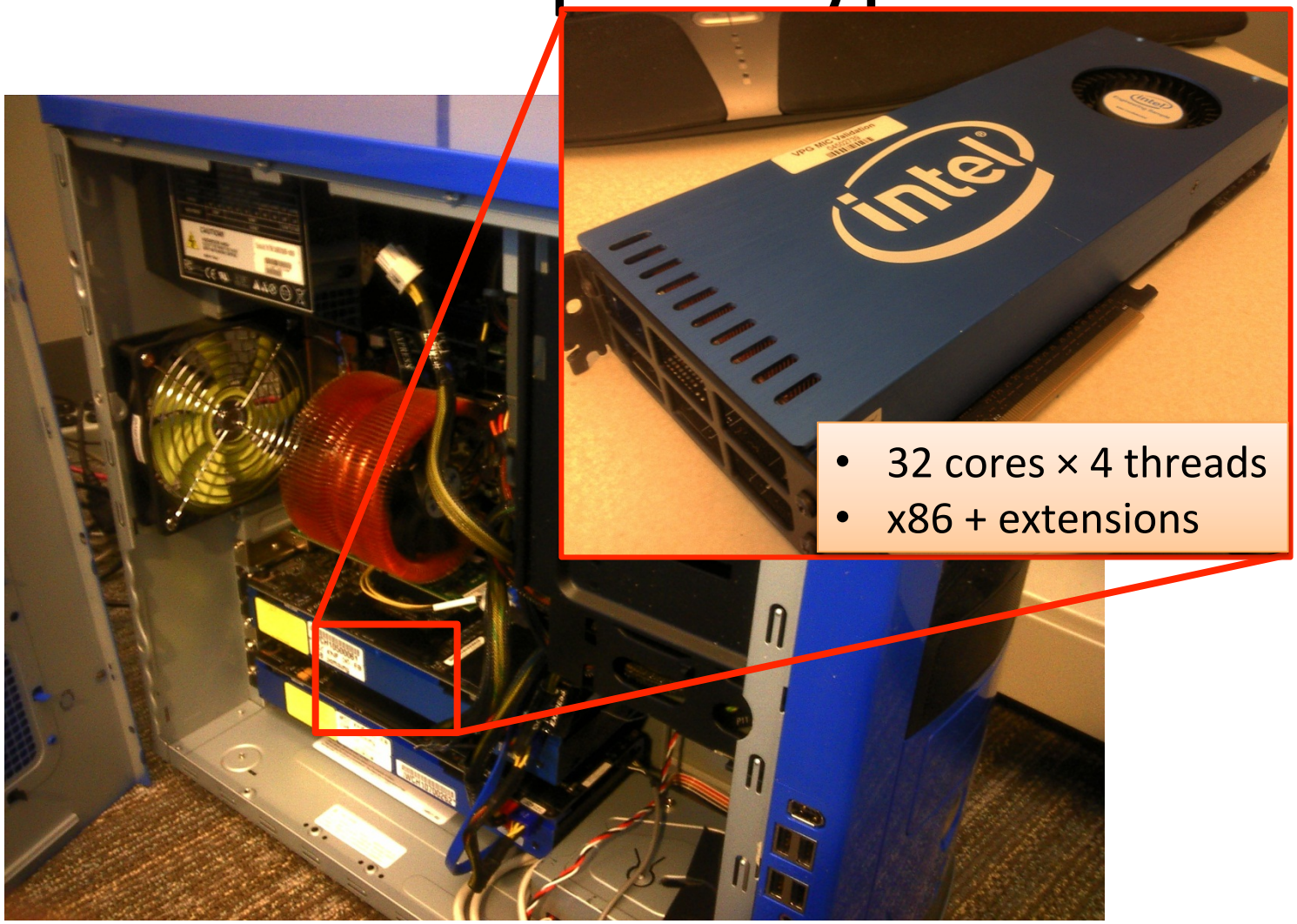
Andrew Baumann[†] Chris Hawblitzel[†] Kornilios Kourtis[§]
Tim Harris[‡] Timothy Roscoe[§]

[†] Microsoft Research [§] ETH Zurich [‡] Oracle Labs

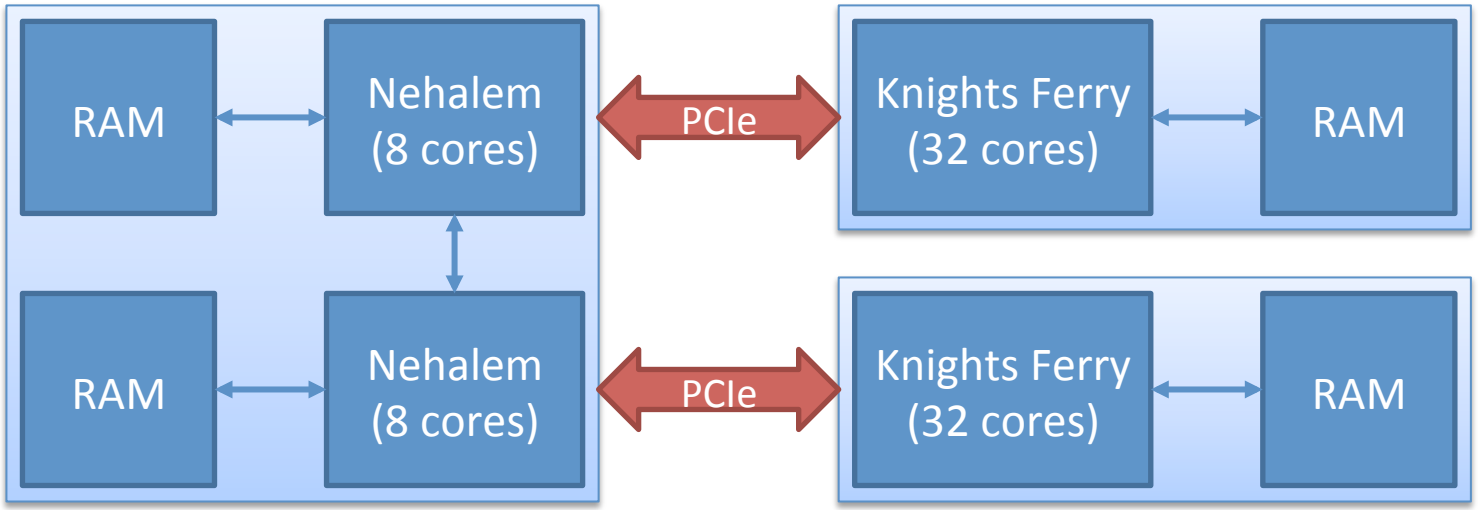
Motivation: heterogeneous multicores

- Accelerators, co-processors, offload engines
  - Discrete GPGPUs
  - AMD Fusion
  - Intel MIC
  - ARM big.LITTLE
  - …

Intel MIC prototype



Target platform



- 4 NUMA domains
- 3 “islands” of cache-coherence

Breaks existing OS assumptions

- Core uniformity
  - Performance properties
  - Instruction set architecture
- Global, cache-coherent shared memory
  - Used for OS data structures

- How can we extend one OS across all cores?

Possible OS models

Cluster-on-a-chip

- Avoid changing the OS
- Treat as "devices", hide behind driver API
- One OS per island
- Paper over the gaps at application/runtime level
  - CUDA, OpenCL, etc.

Multikernel

- It's a distributed system!
- Base OS on message passing
  - Don't assume shared memory
- Single OS across all cores
  - Process management
  - File system
  - Networking
  - Inter-process communication
- e.g. Barrelfish

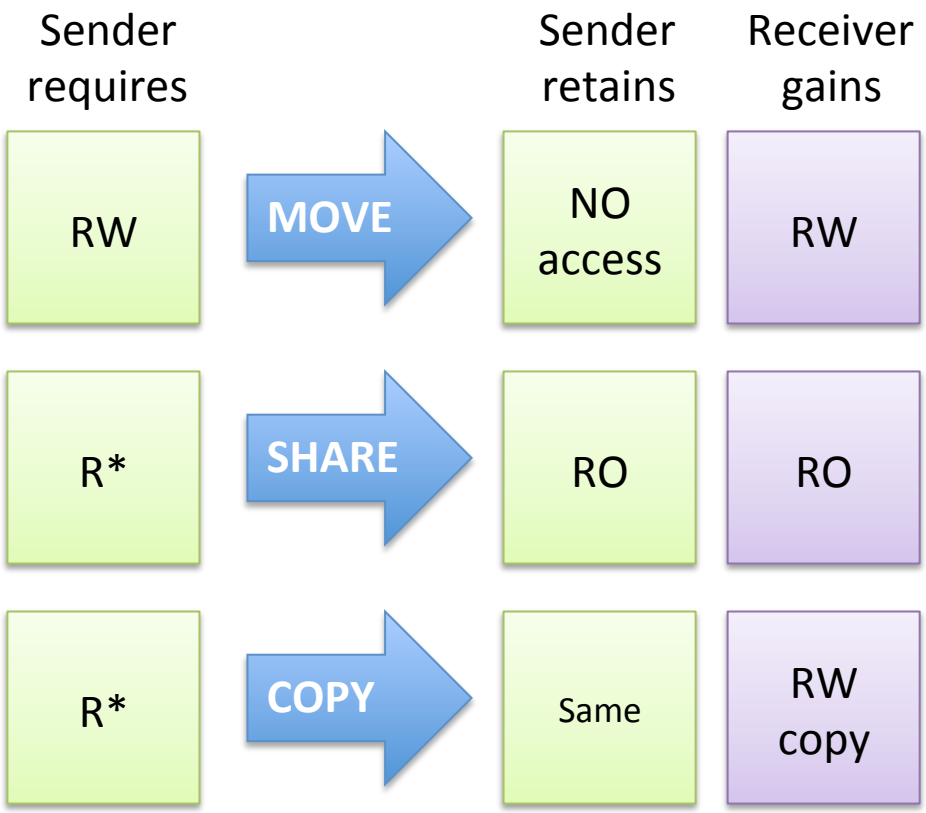
… but sharing data is useful!

- Particularly for bulk data
  - I/O buffers, networking, etc.
  - Computations on large data
- Platform specific – don't want to expose it
- Need an abstraction: **Cosh**

Why is this hard?

- Between any pair of cores, may have:
  - Cache-coherent shared memory
  - Non-cache-coherent shared memory
  - No shared memory
- Different mechanisms to transfer data
  - Page remapping (sharing)
  - DMA controllers (copies)

Transfers



RW: Read/Write, RO: Read-Only, R*: RW or RO

Design Principles:

- Rights never upgraded
  - No RW sharing
- All transfers can be implemented as copies
  - Permits DMA
  - Fast small transfers
- Page mappings established through transfer
  - Permits optimizations

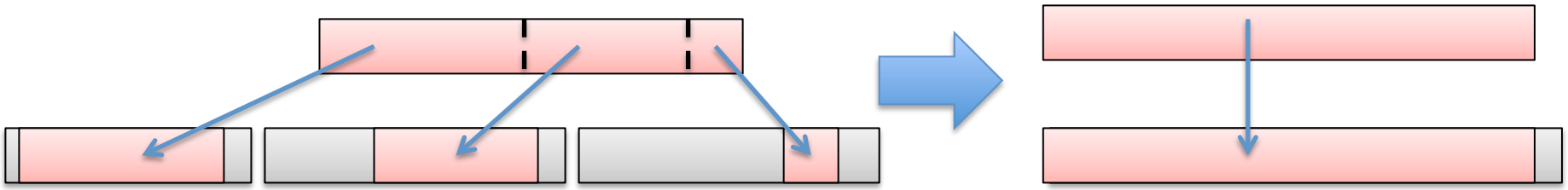
Making it practical

1. Weak transfers
   – Permit efficient use of shared memory
2. Aggregates
   – Avoid page-granularity restrictions

Weak transfers

- Changing memory permissions is costly
  - Update page tables, TLB shootdown, etc.
- Not always necessary (e.g., trusted services)
- Weak transfers permit implementation to defer permission changes
  - e.g. sender of a weak share may retain write permissions, but is trusted not to do so

Aggregates



- Page-granularity doesn't work for everything
  - Byte-oriented APIs (e.g., POSIX read/write)
  - Differing page sizes
- Cosh adds high-level *aggregate* abstraction
  - Byte-granularity buffer access, transfers
  - Derived from IO-Lite
- Aggregate structure is **not** maintained across transfers

Trivial example: pipes

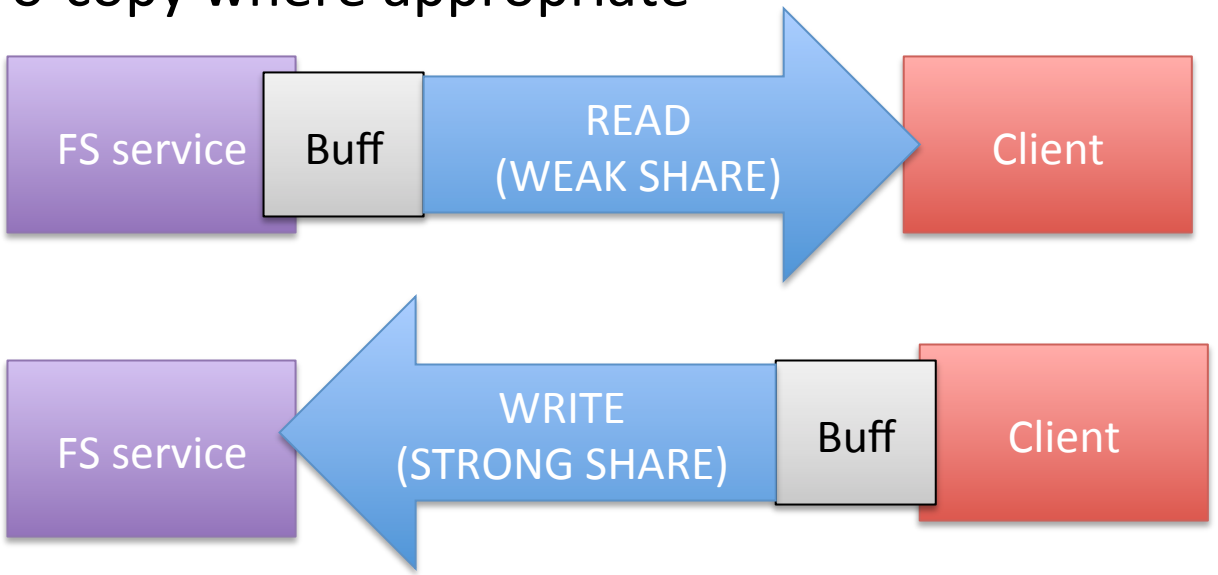
```
void pipe_write(wpipe *pipe, cosh_agg *agg)
{
  cosh_agg_transfer(agg, pipe->dest,
                    COSH_MOVE,
                    COSH_TRANSFER_STRONG);
  cosh_agg_decref(agg);
}
```

```
cosh_agg *pipe_read(rpipe *pipe)
{
  ...
  cosh_agg_receive(pipe->src, &agg,
                   &mode, &flags);

  if (mode != COSH_MOVE
      || (flags & COSH_TRANSFER_WEAK)) {
    // protocol error by sender
    ...
  }

  return agg;
}
```

- Untrusted sender: strong transfer
- Zero-copy (where permitted)

Real example: file system

- Aggregates for POSIX read()/write()
  - Zero-copy where appropriate

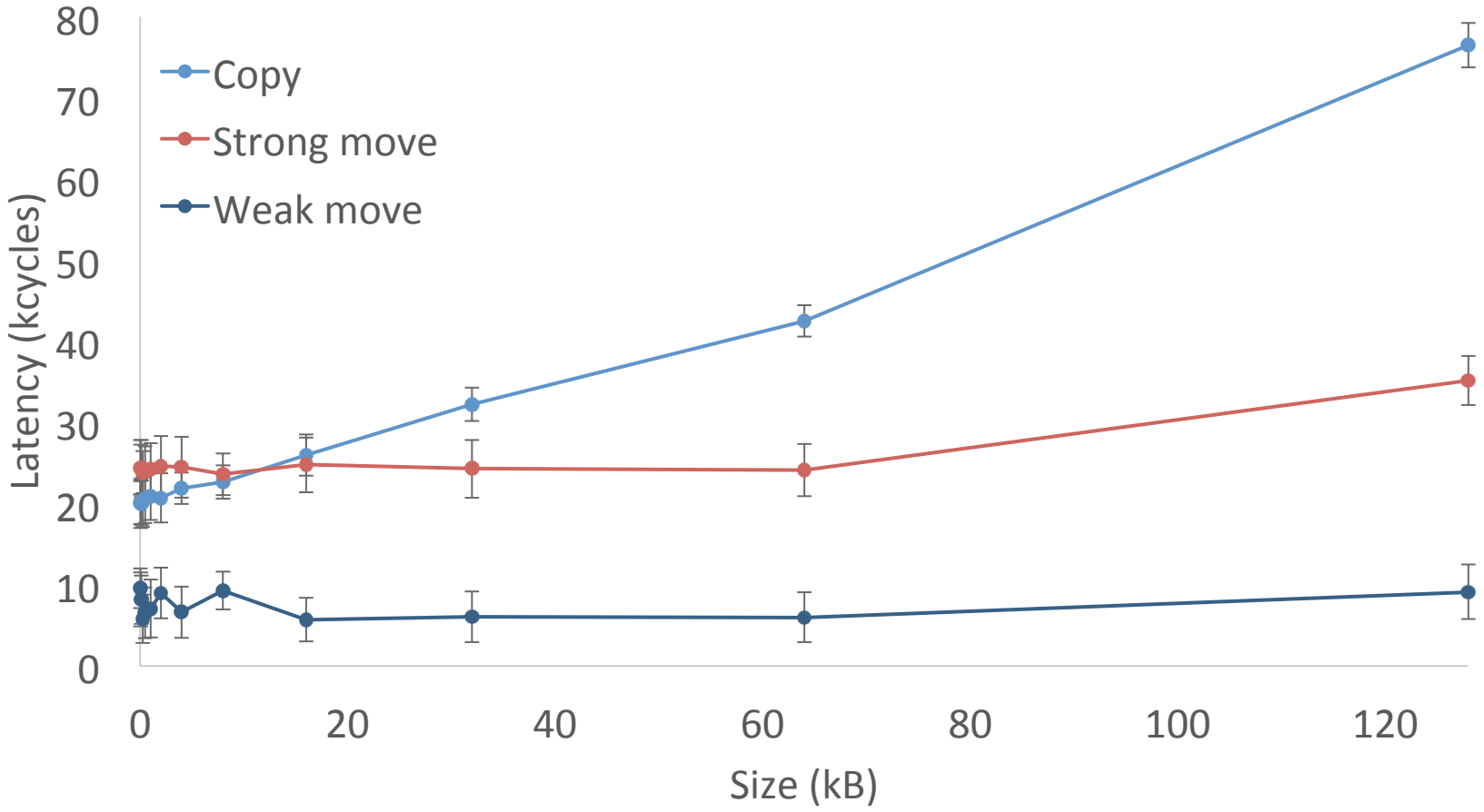


- Works exactly the same on MIC cores

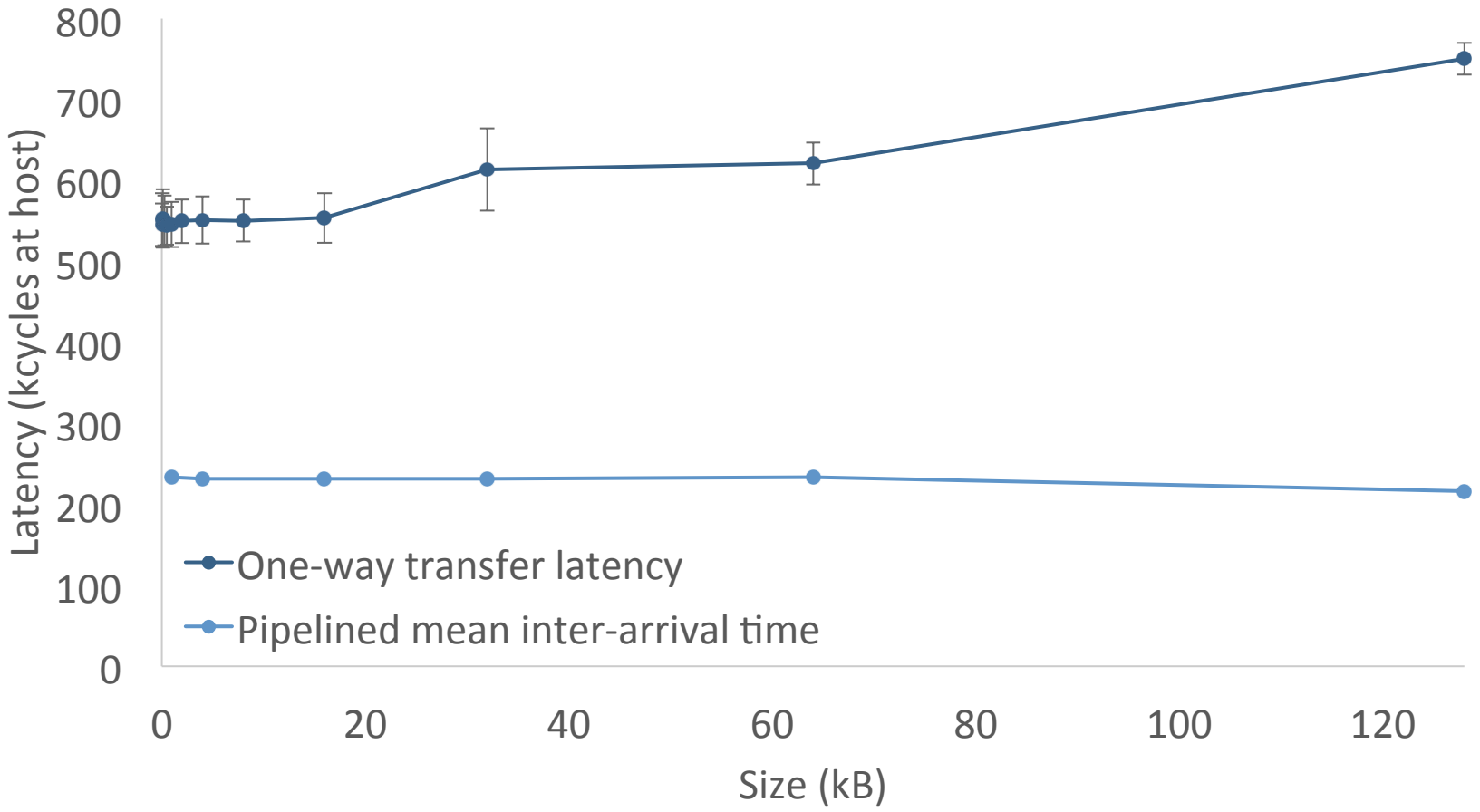
Prototype implementation

- Ported Barrelfish OS to MIC
- Heterogeneous system of x86 and MIC cores
  - Sharing where possible
  - DMA between MIC and PC
- "Asynchronous C" (AC) language [OOPSLA'11]
  - Lightweight extensions to C for asynchrony
- Simple user-space implementation
  - Not performant

Transfers between host cores



DMA transfer to Knights Ferry



Panorama stitching

- Capture trace on Linux, replay it on Barrelfish
  - RamFS
  - CoshFS

File System	Host (ms)	Co-processor (ms)
RamFS	145	-
CoshFS	144	49742
CoshFS + cache		2464

Related work

- High-performance I/O systems
  - Shared-memory optimisations are similar to previous bulk data transport systems
  - IO-Lite inspired our aggregate API
- OS support for specialised cores
  - Research OSes picked message-like semantics; e.g. copy [Hydra, Barrelfish] or move [Helios]
  - Other work has been driven by limitations of GPUs [PTask, GPUfs]

Conclusion

- **Cosh**: new abstraction for managing bulk data
- Used within OS for:
  - Shared file system
  - Inter-process communication
  - Networking, etc.
- Can exploit shared memory; doesn't rely on it

BACKUP SLIDES

Aggregate API

```
alloc(len, flags) -> agg
incref(agg)
decref(agg)
getlen(agg) -> length
getrights(agg) -> rights
iter_start(agg, read|write, offset) -> iter
iter_next(iter) -> addr, length
iter_end(iter)
concat(agg1, agg2) -> agg
select(agg, offset, length) -> agg
find_related(agg, minrights) -> [agg]
downgrade(agg, rights)
transfer(agg, dest, transfer_mode, flags)
receive(src) -> agg, transfer_mode, flags
```