Programmable Accelerators

Jason Lowe-Power

powerjg@cs.wisc.edu

cs.wisc.edu/~powerjg

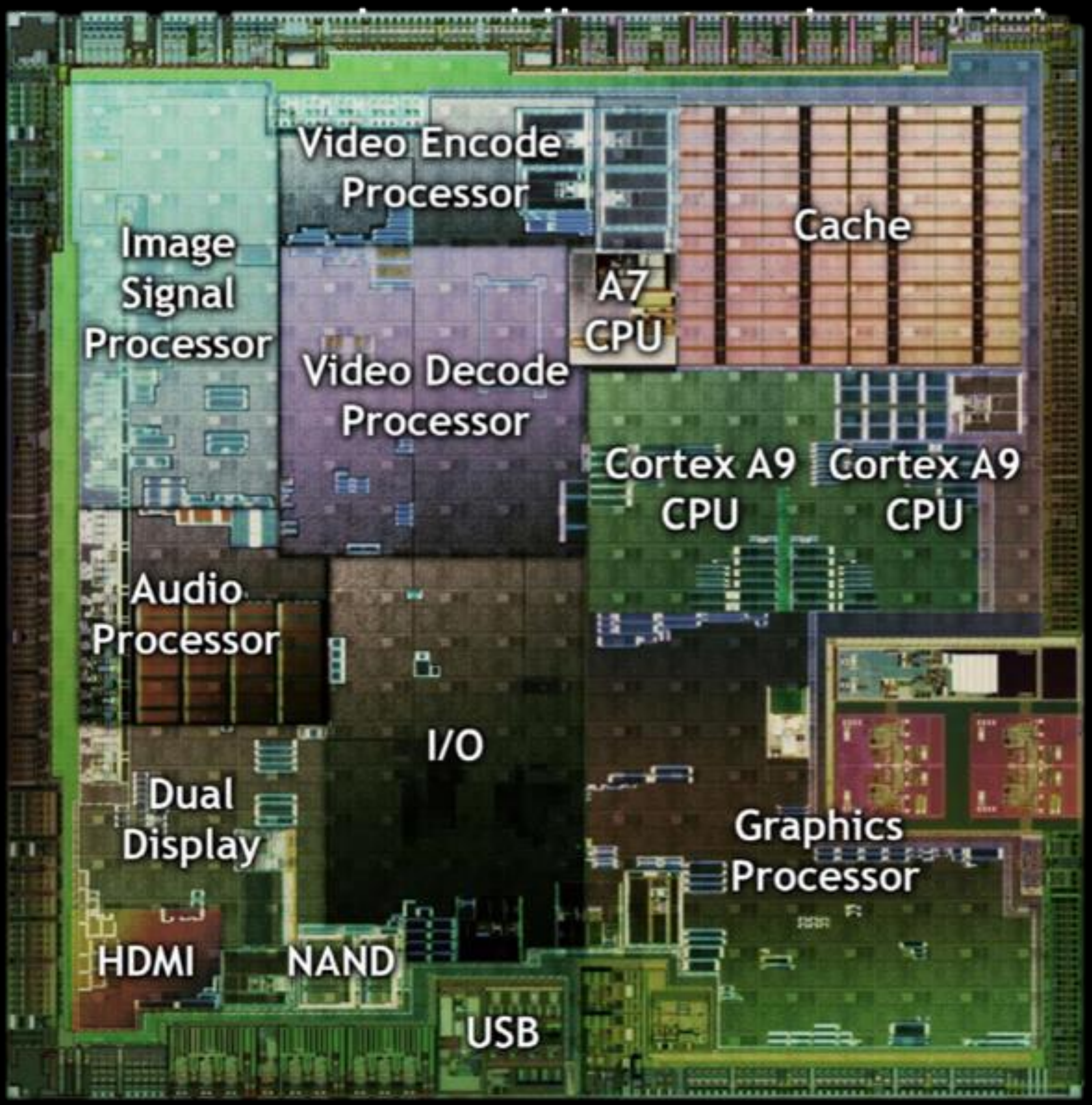
WISCONSIN

Increasing specialization

Need to ***program***
these accelerators

Challenges
1. Consistent pointers
2. Data movement
3. Security (Fast)

This talk: GPGPUs



*NVIDIA via anandtech.com

Programming accelerators (baseline)

```
void add(int*a, int*b, int*c) {
    for (int i = 0;
         i < N;
         i++)
    {
        c[i] = a[i] + b[i];
    }
}
```

Accelerator-side code

```
int main() {
    int a[N], b[N], c[N];

    init(a, b, c);

    add(a, b, c);

    return 0;
}
```

CPU-Side code

Programming accelerators (GPU)

```
void add_gpu(int*a, int*b, int*c) {
    for (int i = get_global_id(0);
         i < N;
         i += get_global_size(0))
    {
        c[i] = a[i] + b[i];
    }
}
```

Accelerator-side code

```
int main() {
    int a[N], b[N], c[N];

    init(a, b, c);

    add(a, b, c);

    return 0;
}
```

CPU-Side code

Programming accelerators (GOAL)

```
void add_gpu(int*a, int*b, int*c) {
    for (int i = get_global_id(0);
         i < N;
         i += get_global_size(0))
    {
        c[i] = a[i] + b[i];
    }
}
```

Accelerator-side code

```
int main() {
    int a[N], b[N], c[N];
    int *d_a, *d_b, *d_c;

    cudaMalloc(&d_a, N*sizeof(int));
    cudaMalloc(&d_b, N*sizeof(int));
    cudaMalloc(&d_c, N*sizeof(int));

    init(a, b, c);

    cudaMemcpy(d_a, a, N*sizeof(int),
               cudaMemcpyHostToDevice);
    cudaMemcpy(d_b, b, N*sizeof(int),
               cudaMemcpyHostToDevice);

    add_gpu(a, b, c);

    cudaMemcpy(c, d_c, N*sizeof(int),
               cudaMemcpyDeviceToHost);

    cudaFree(d_a); cudaFree(d_b);
    cudaFree(d_c);

    return 0;
}
```

CPU-Side code

Programming accelerators (GOAL)

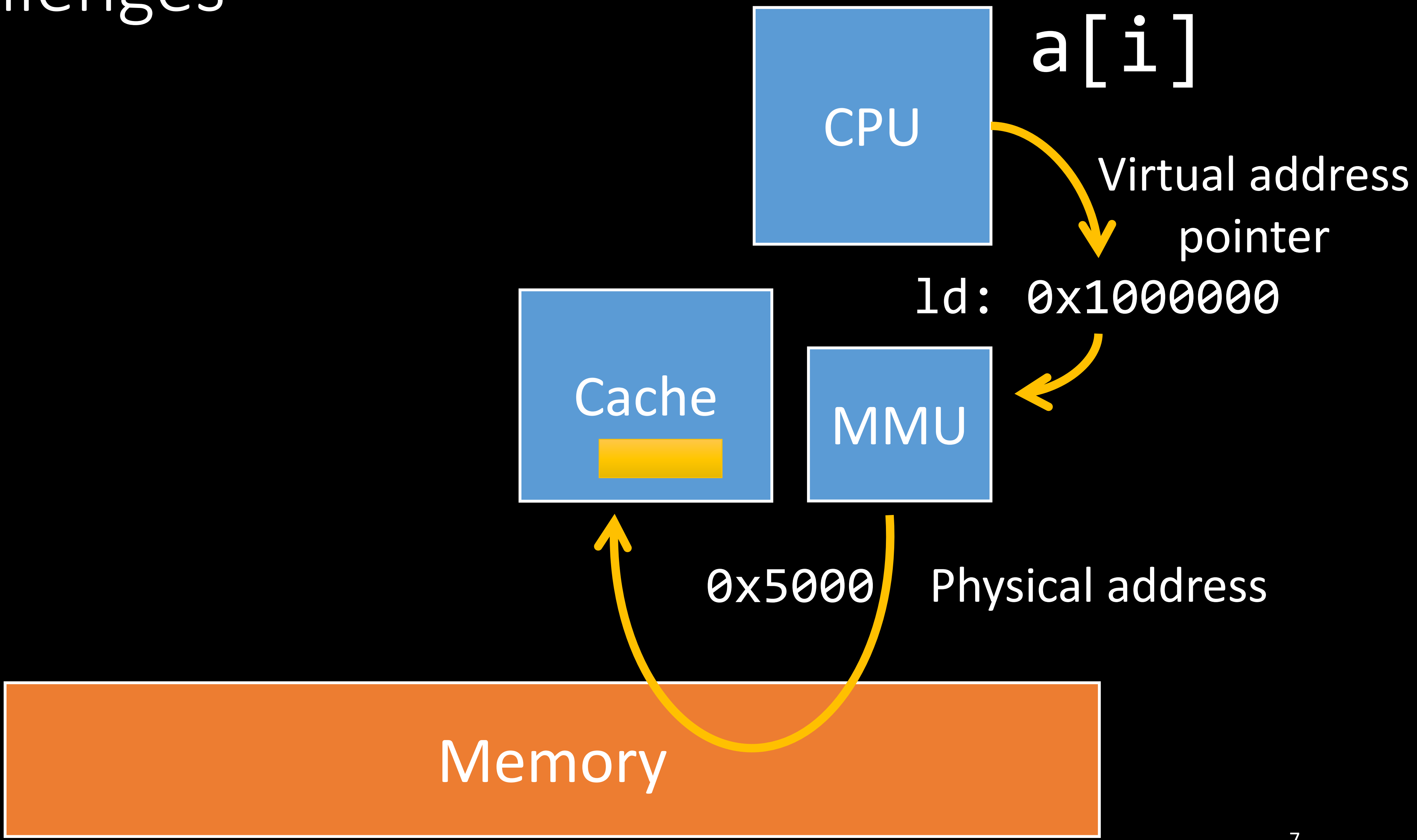
```
void add_gpu(int*a, int*b, int*c) {
    for (int i = get_global_id(0);
         i < N;
         i += get_global_size(0))
    {
        c[i] = a[i] + b[i];
    }
}
```

Accelerator-side code

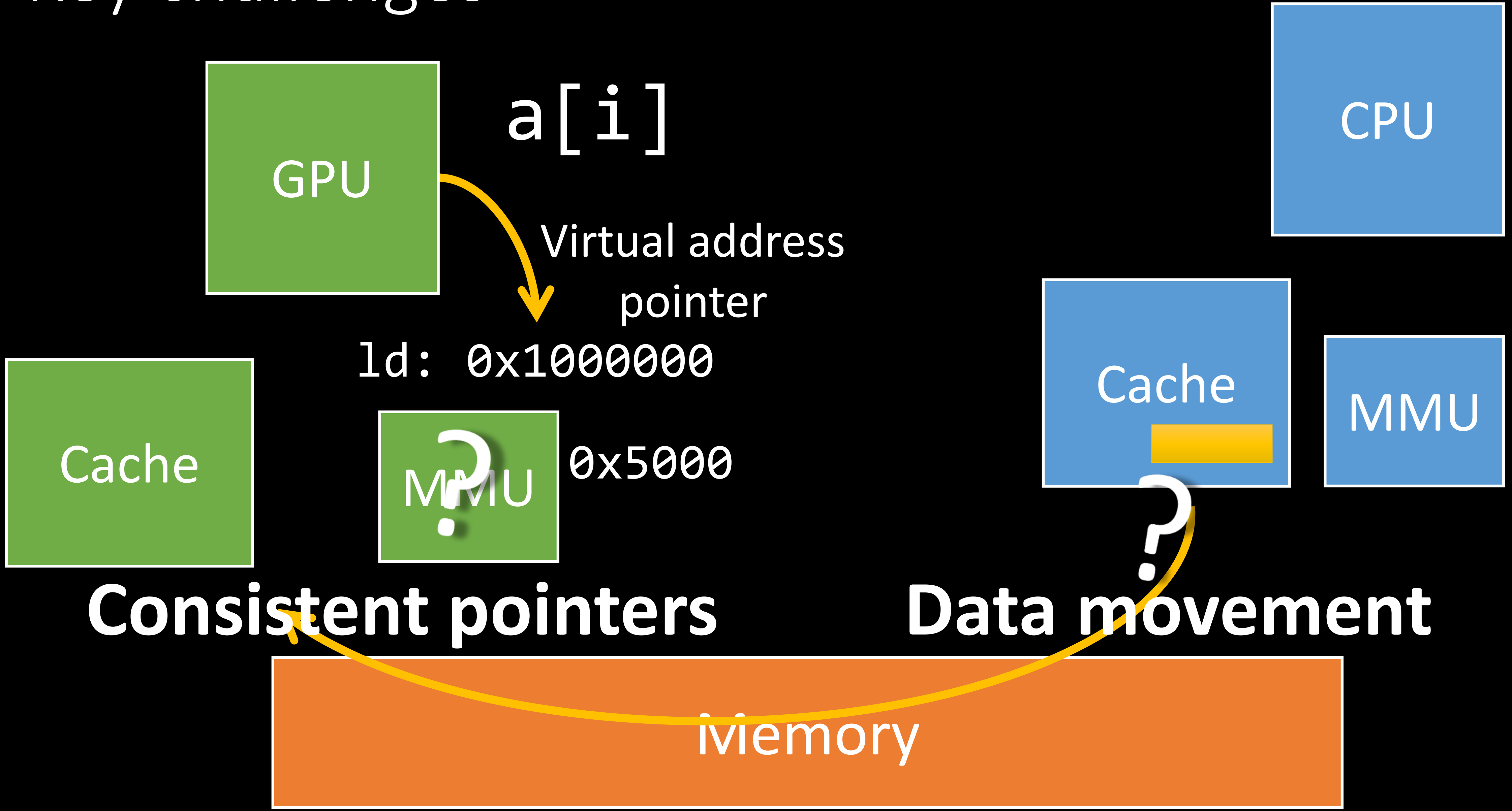
```
int main() {
    int a[N], b[N], c[N];

    init(a, b, c);

    add_gpu(a, b, c);

    return 0;
}
```

CPU-Side code

Key challenges

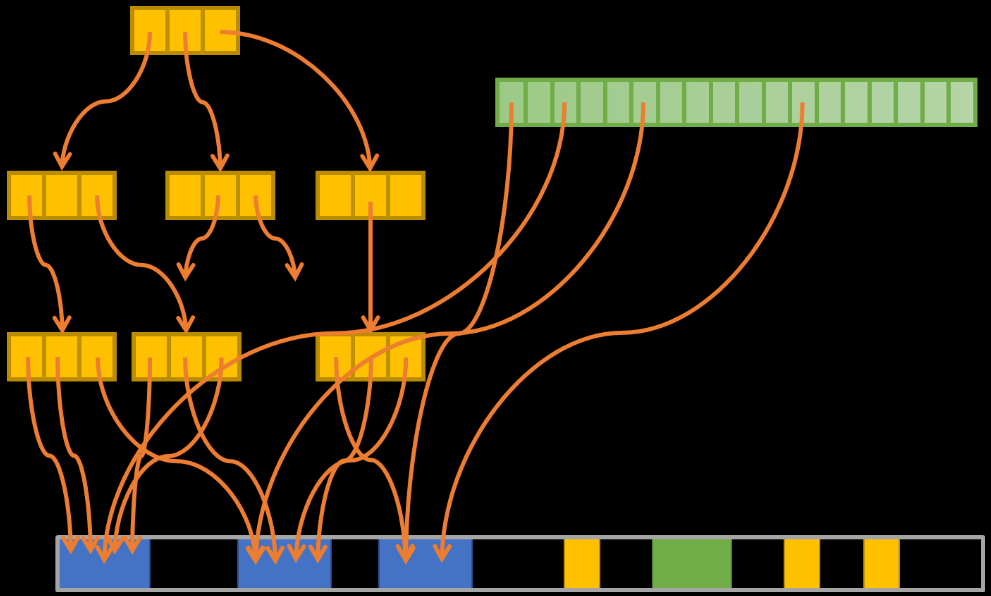


Key challenges



Consistent pointers

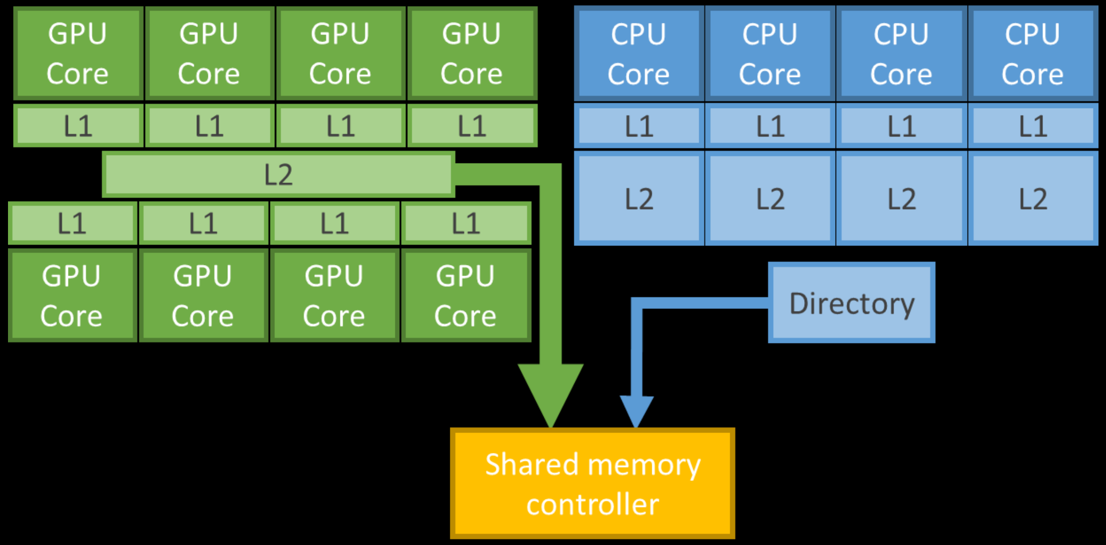
Supporting x86-64 Address Translation for 100s of GPU Lanes



[HPCA 2014]

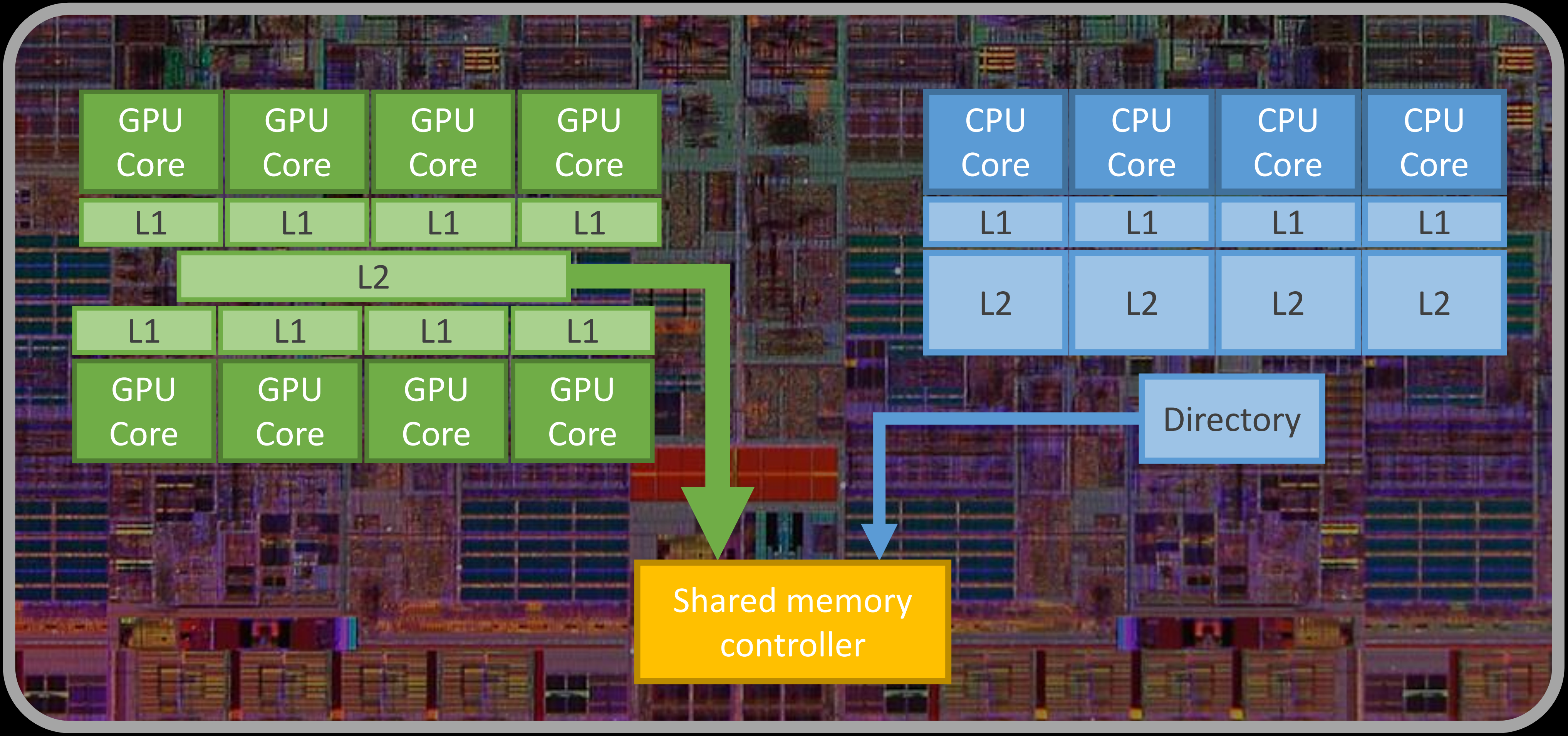
Data movement

Heterogeneous System Coherence



[MICRO 2014]

Heterogeneous System

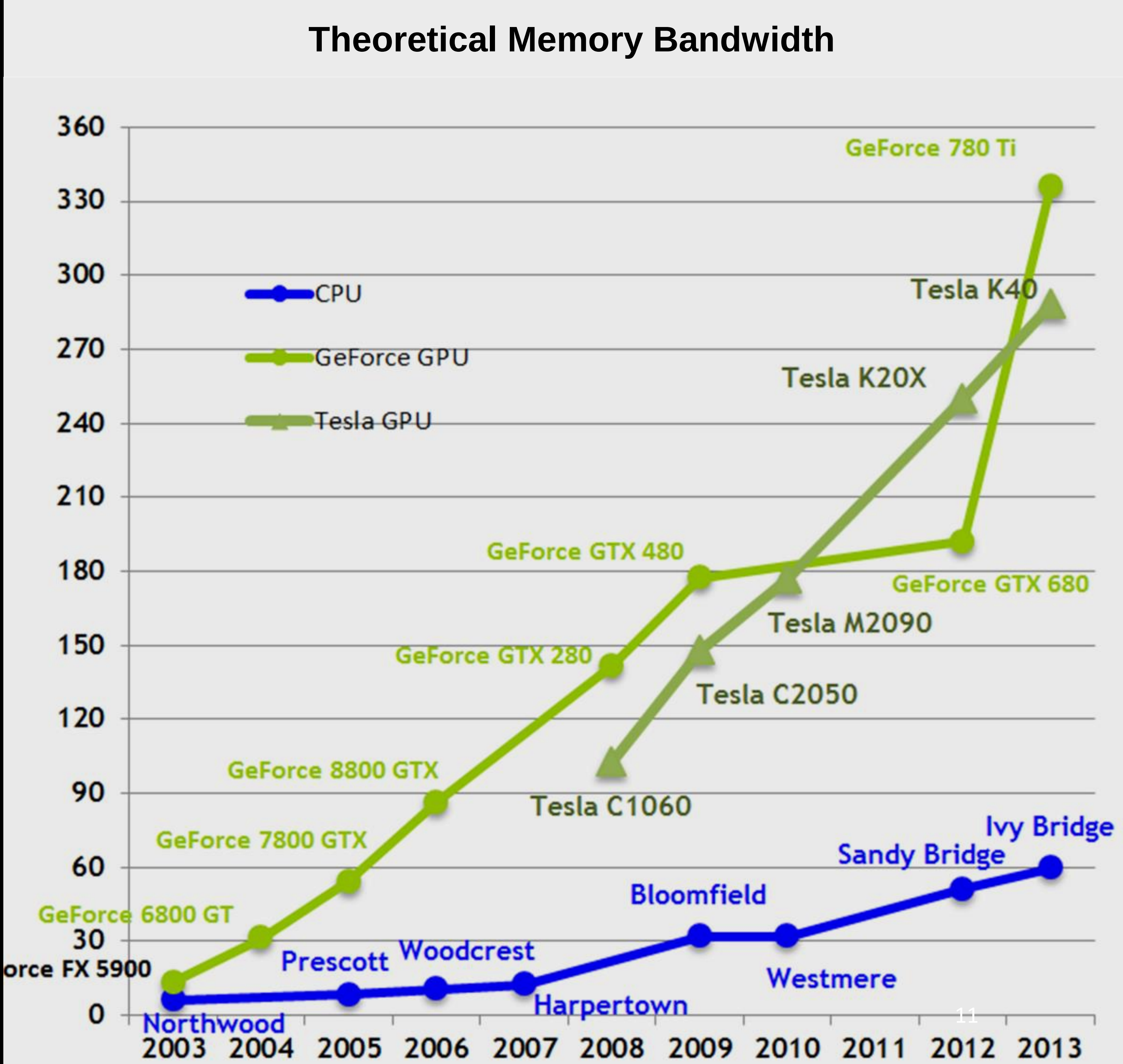


Why not CPU solutions?

It’s all about bandwidth!

Translating 100s of addresses

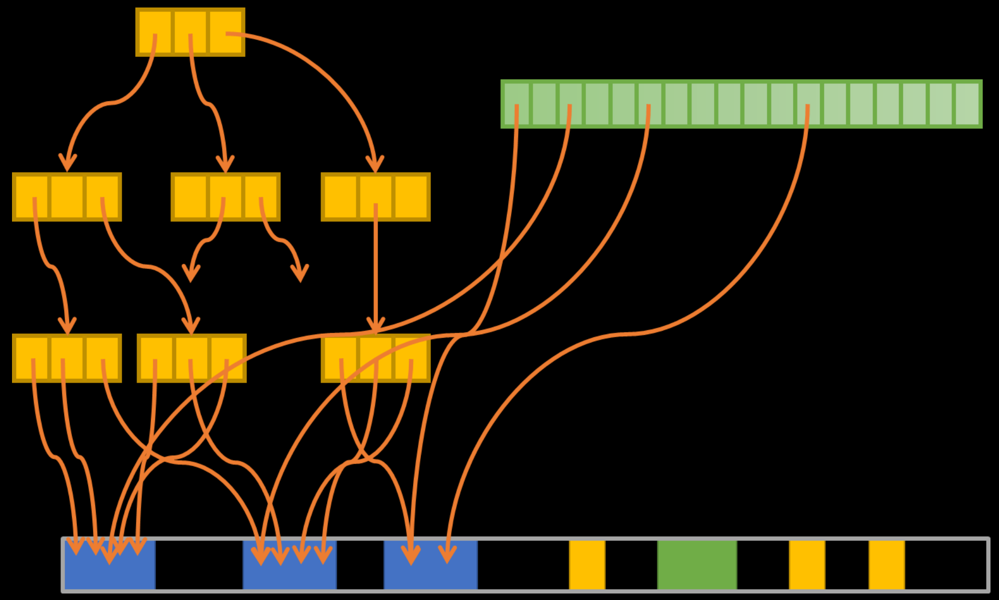
500 GB/s at the directory (many accesses per-cycle)



*NVIDIA via anandtech.com

Consistent pointers

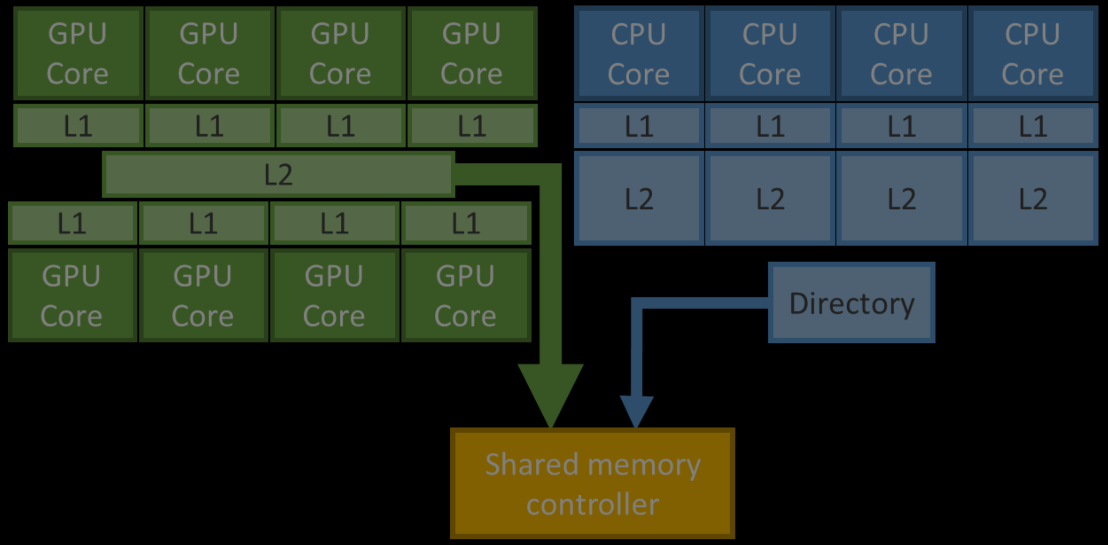
Supporting x86-64 Address Translation for 100s of GPU Lanes



[HPCA 2014]

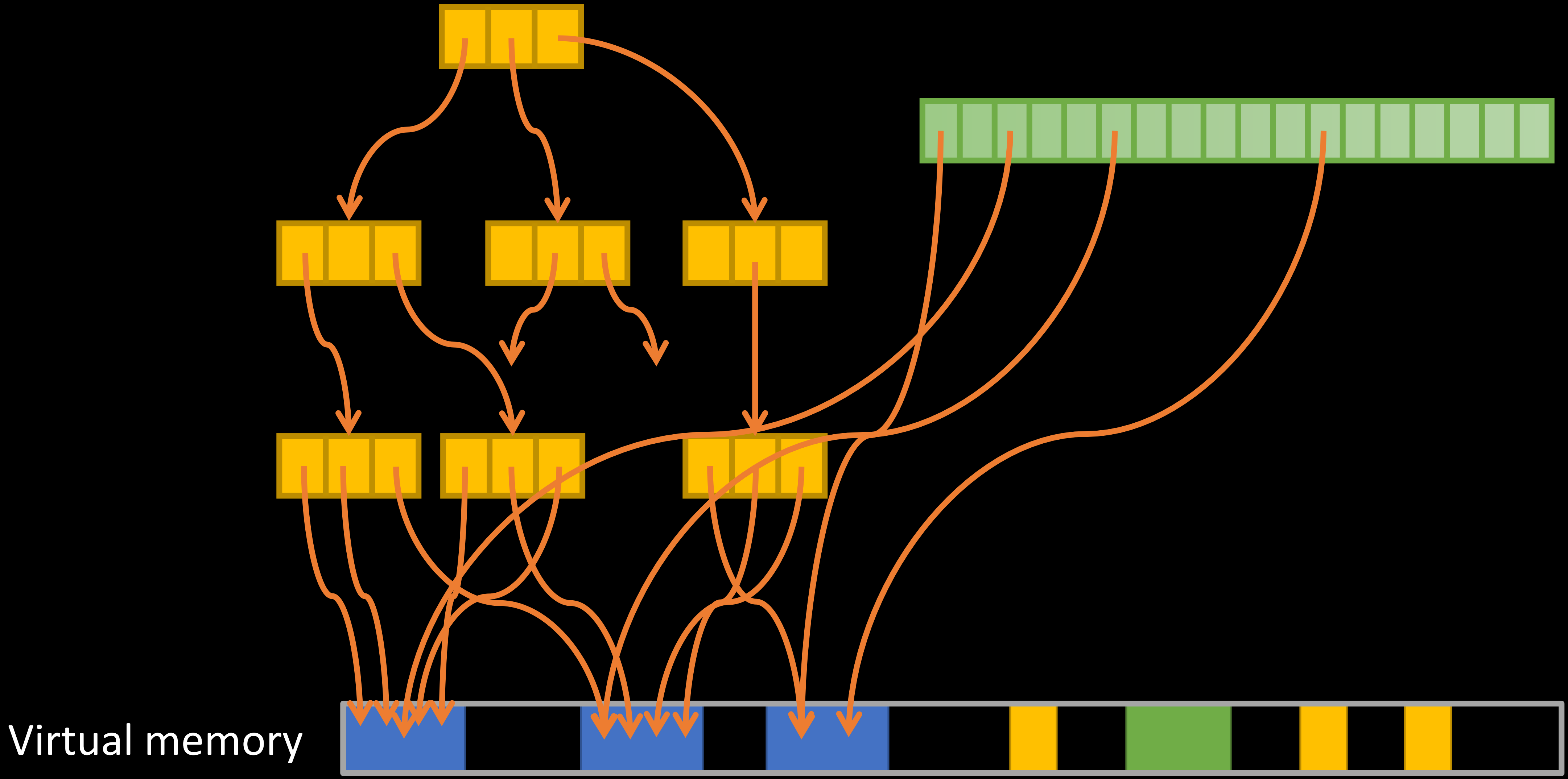
Data movement

Heterogeneous System Coherence

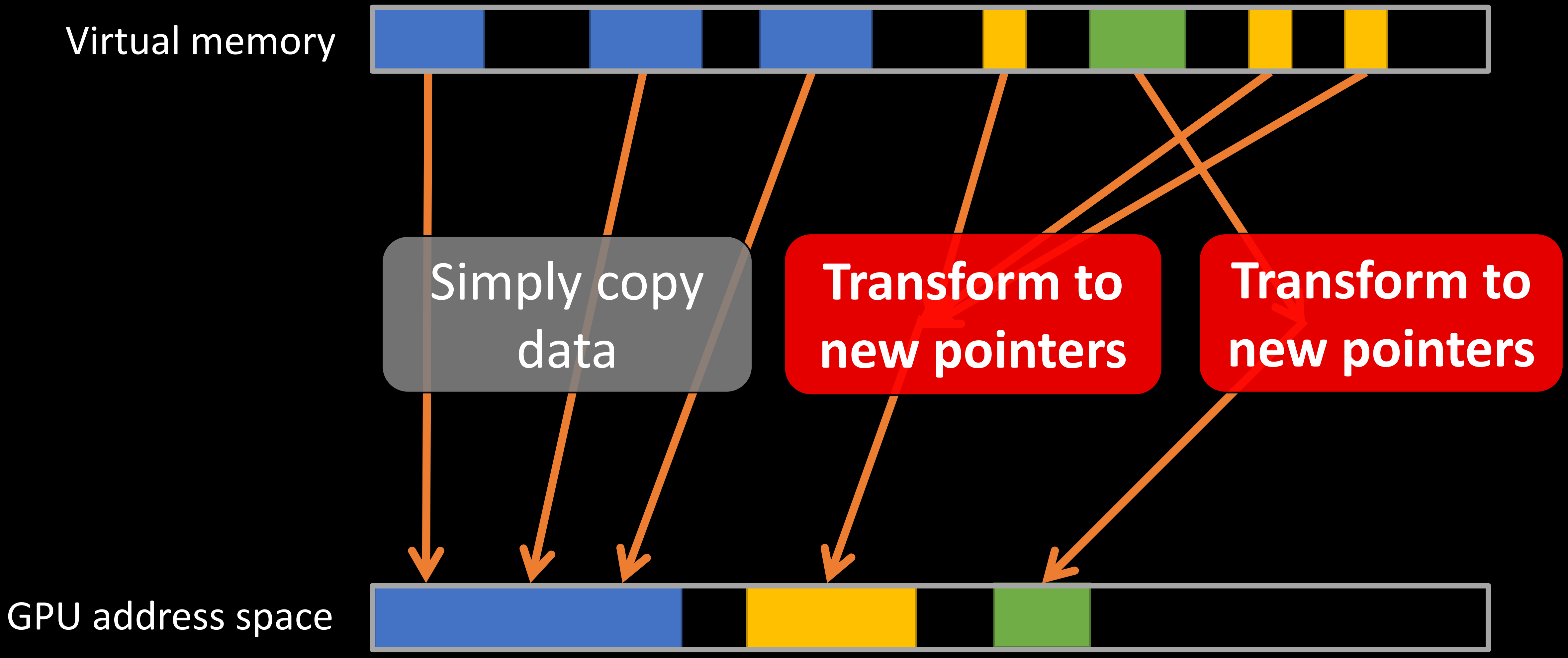


[MICRO 2014]

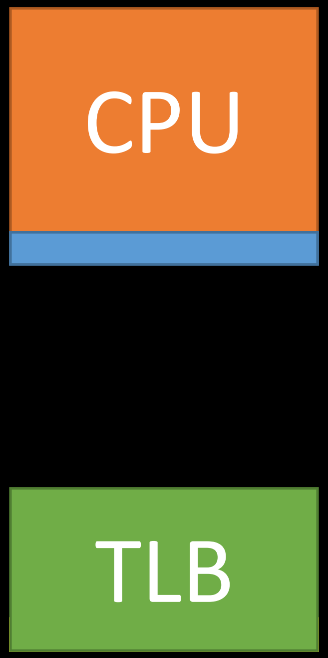
Why virtual addresses?



Why virtual addresses?



Bandwidth problem

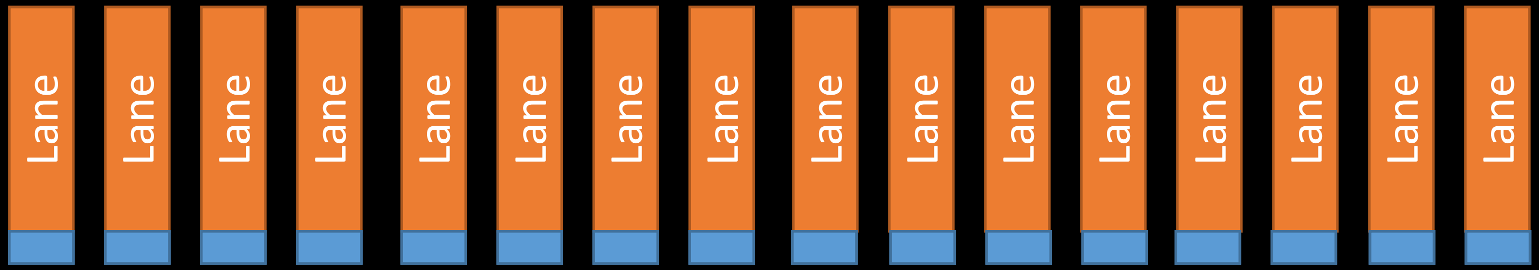


Virtual memory requests

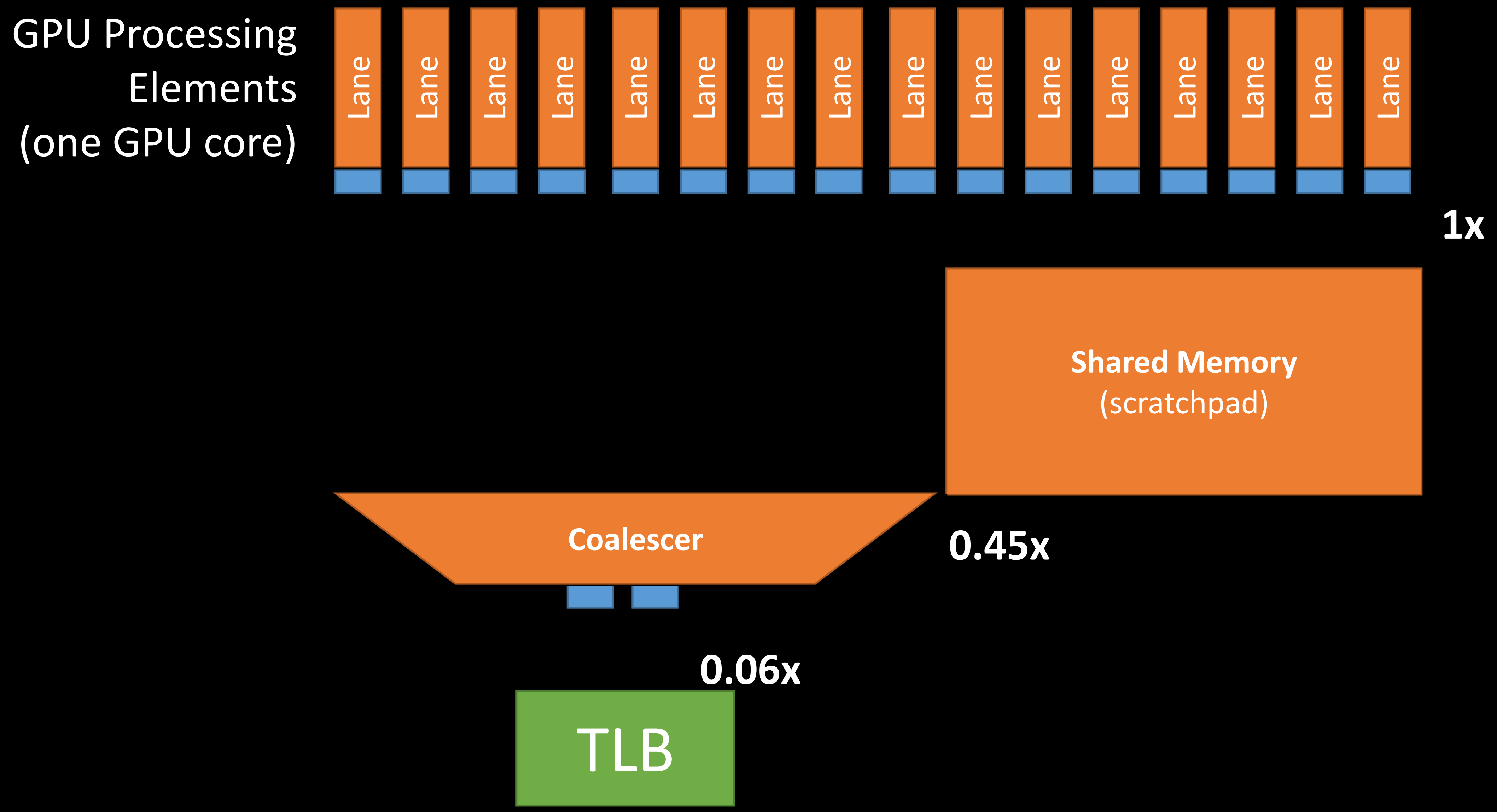
Physical memory requests

Bandwidth problem

GPU Processing Elements (one GPU core)

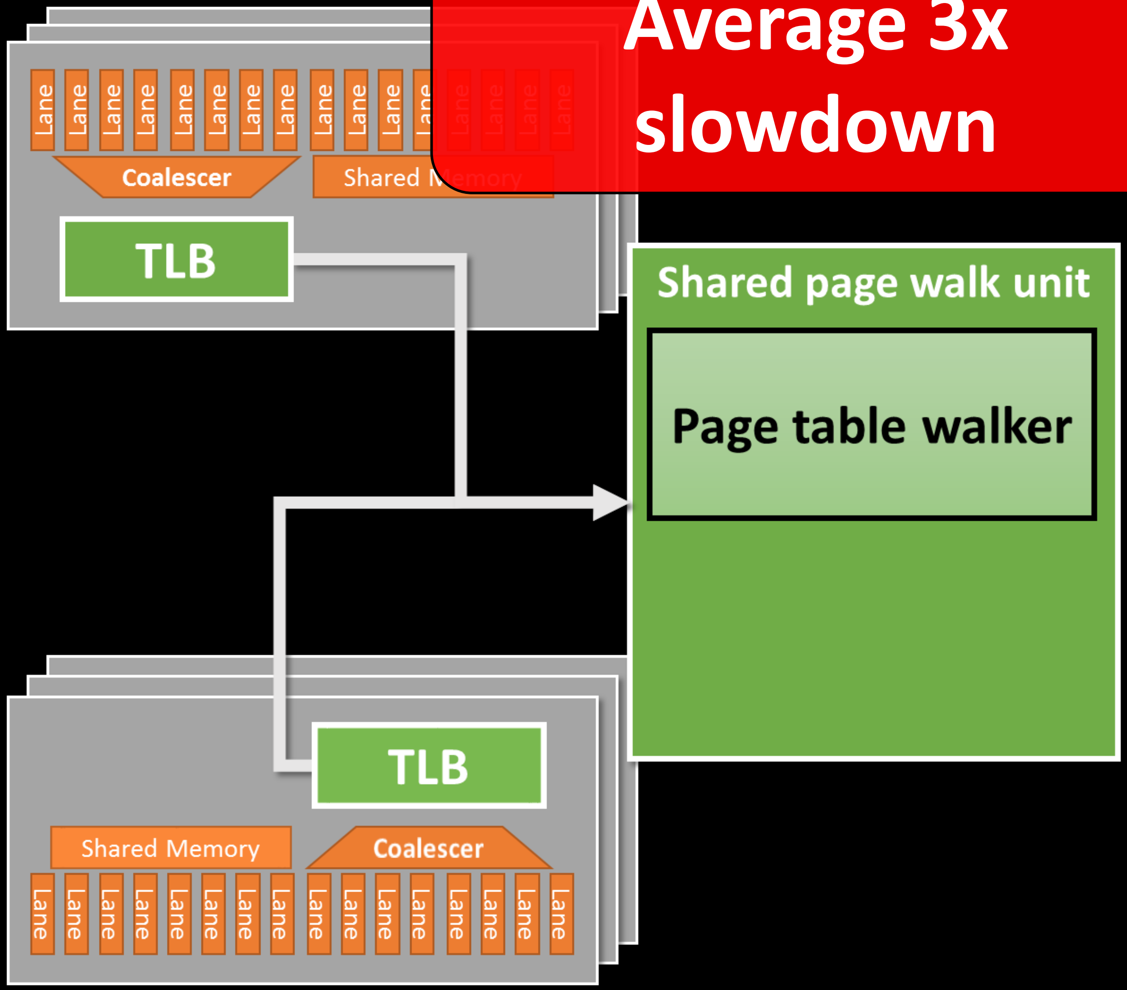


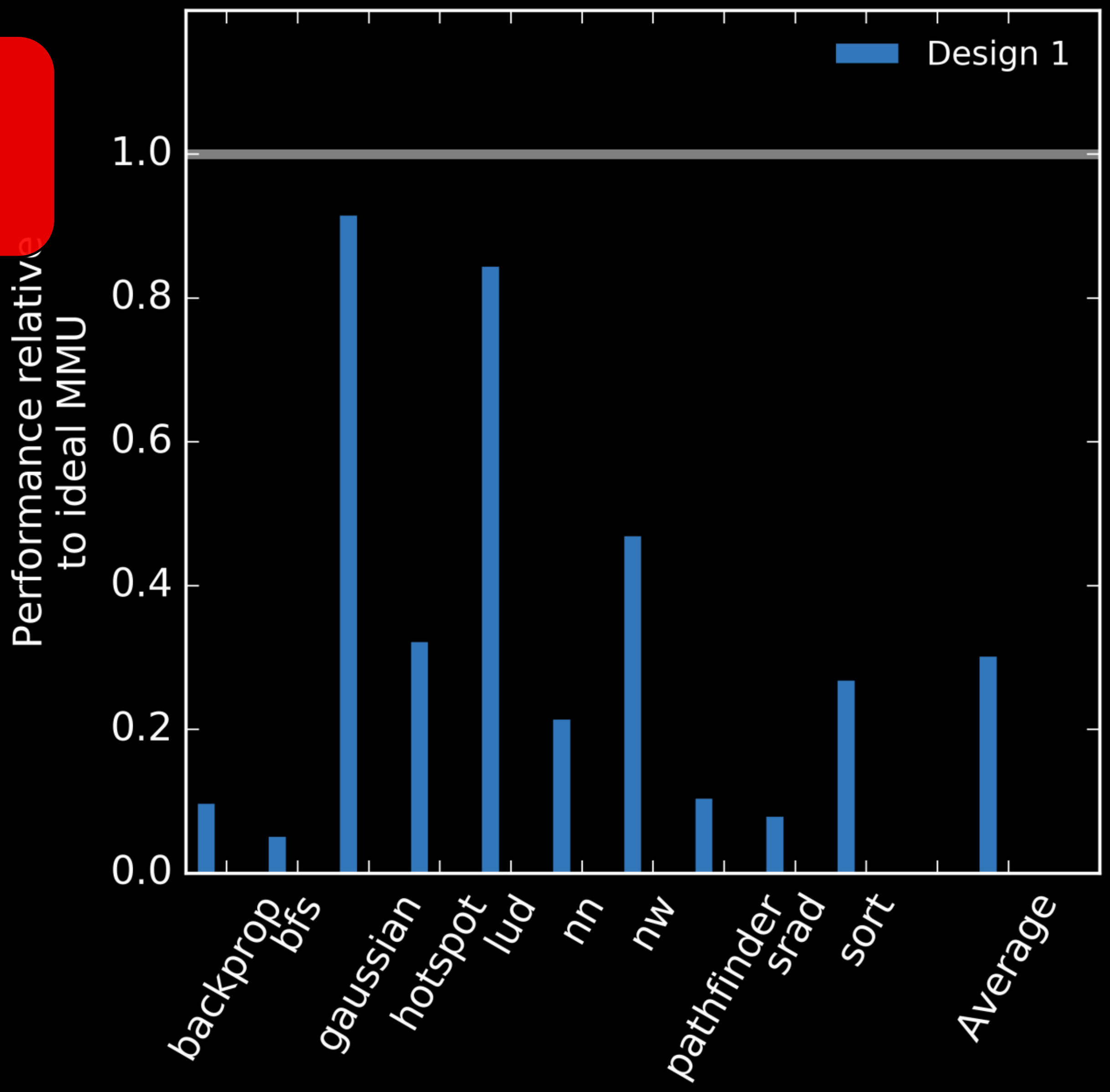
Solution: Filtering



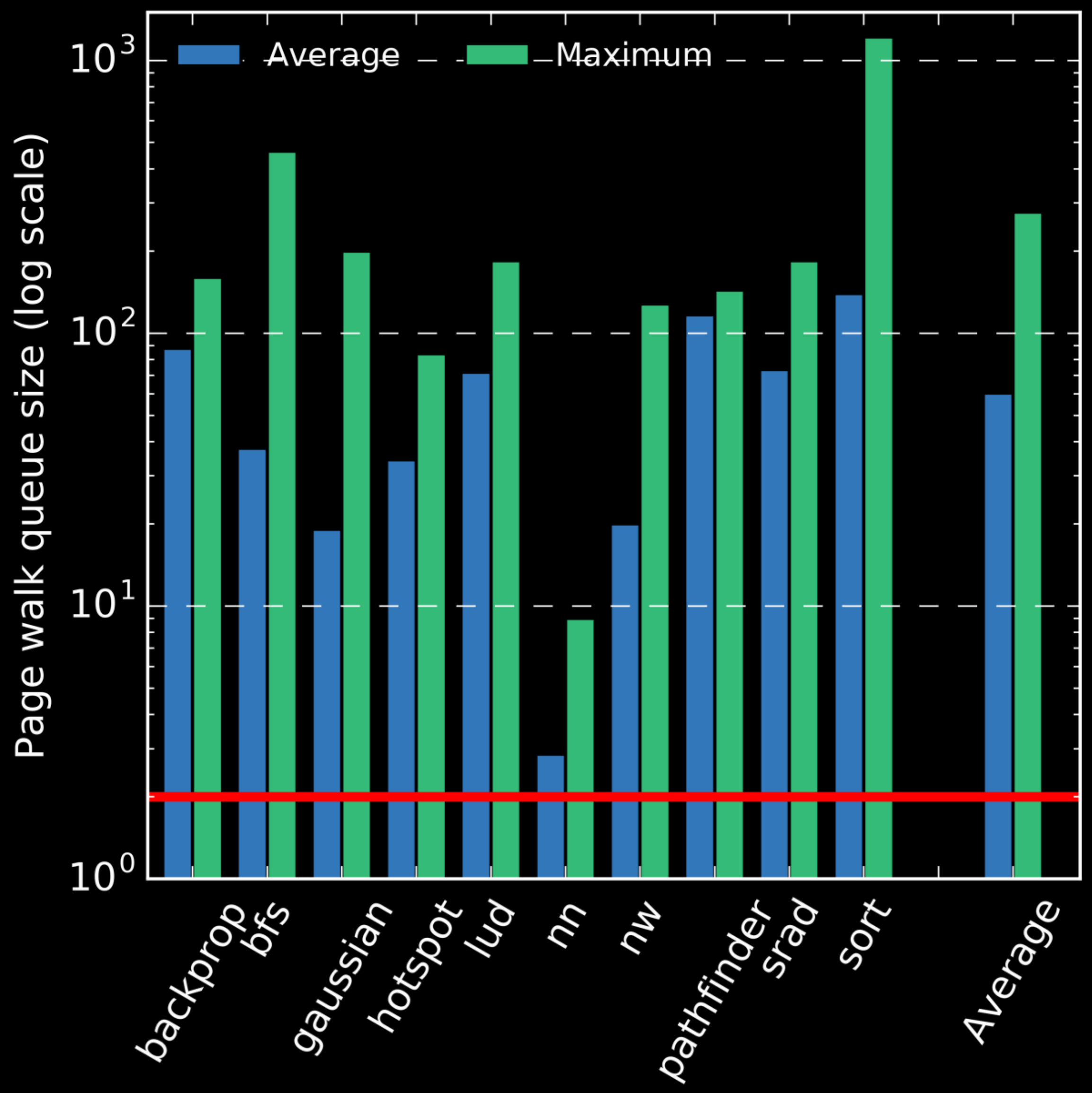
Poor performance

Average 3x slowdown





Bottleneck 1: Bursty TLB misses



Average: 60 outstanding requests

Max 140 requests

Huge queuing delays

Solution:

Highly-threaded pagetable walker

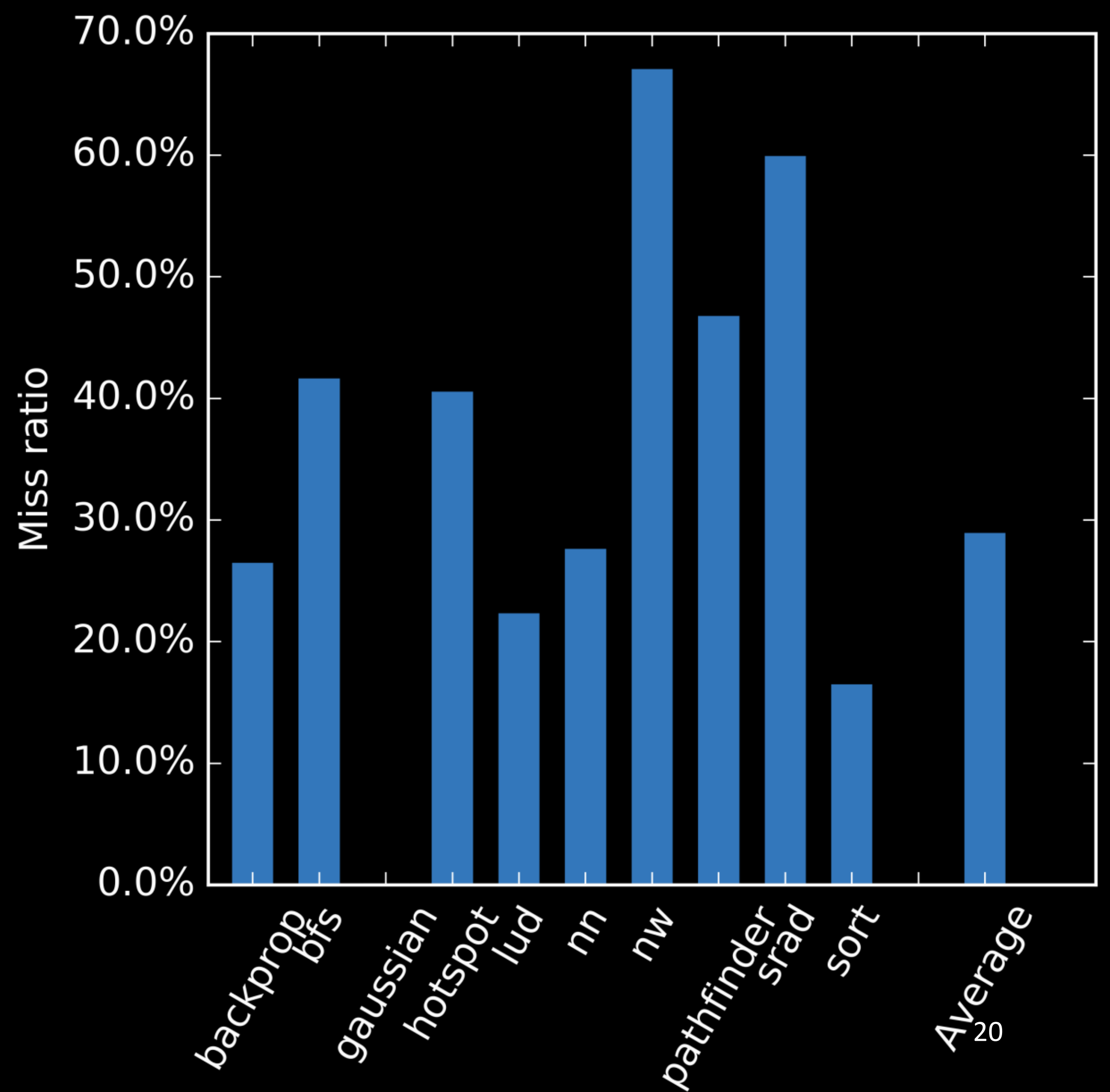
Bottleneck 2: High miss rate

Large 128 entry TLB doesn't help

Many address streams

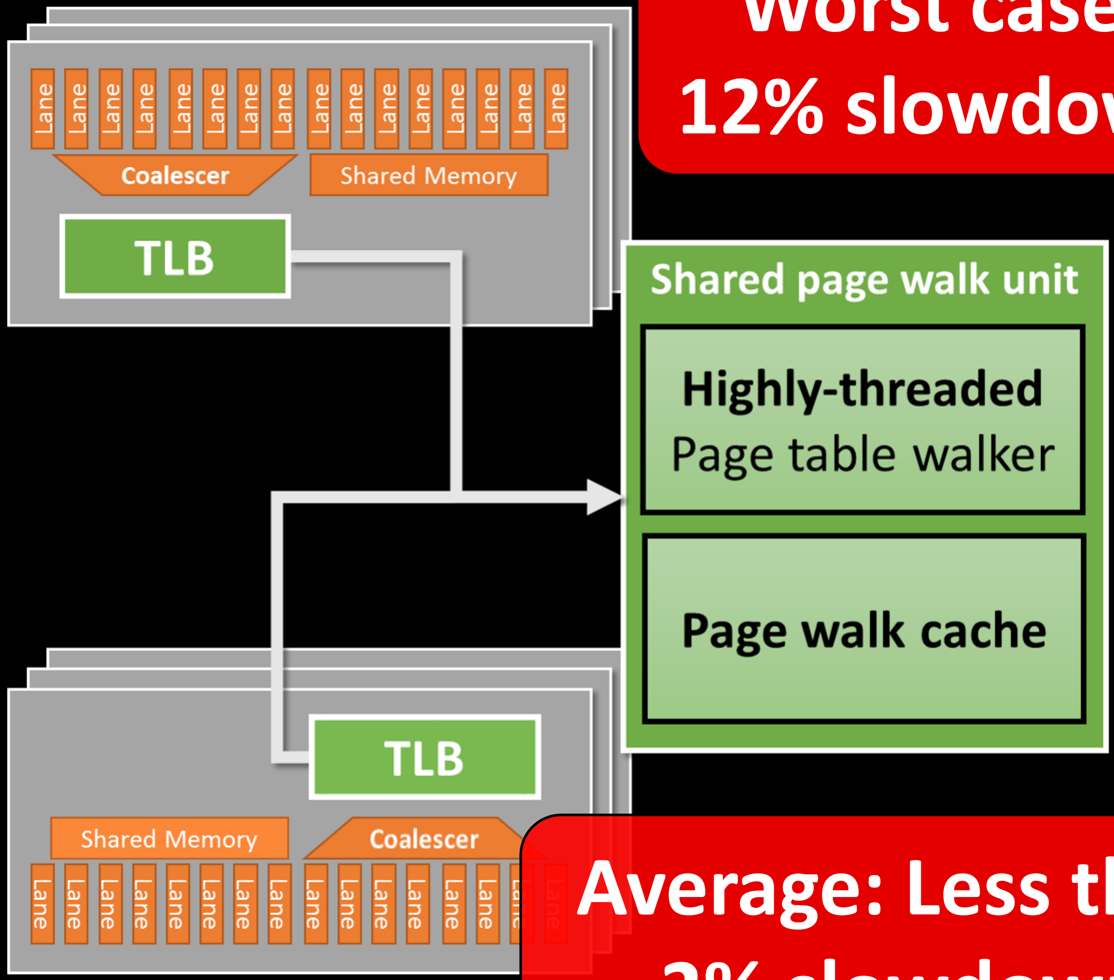
Need low latency

Solution:
Shared page-walk cache

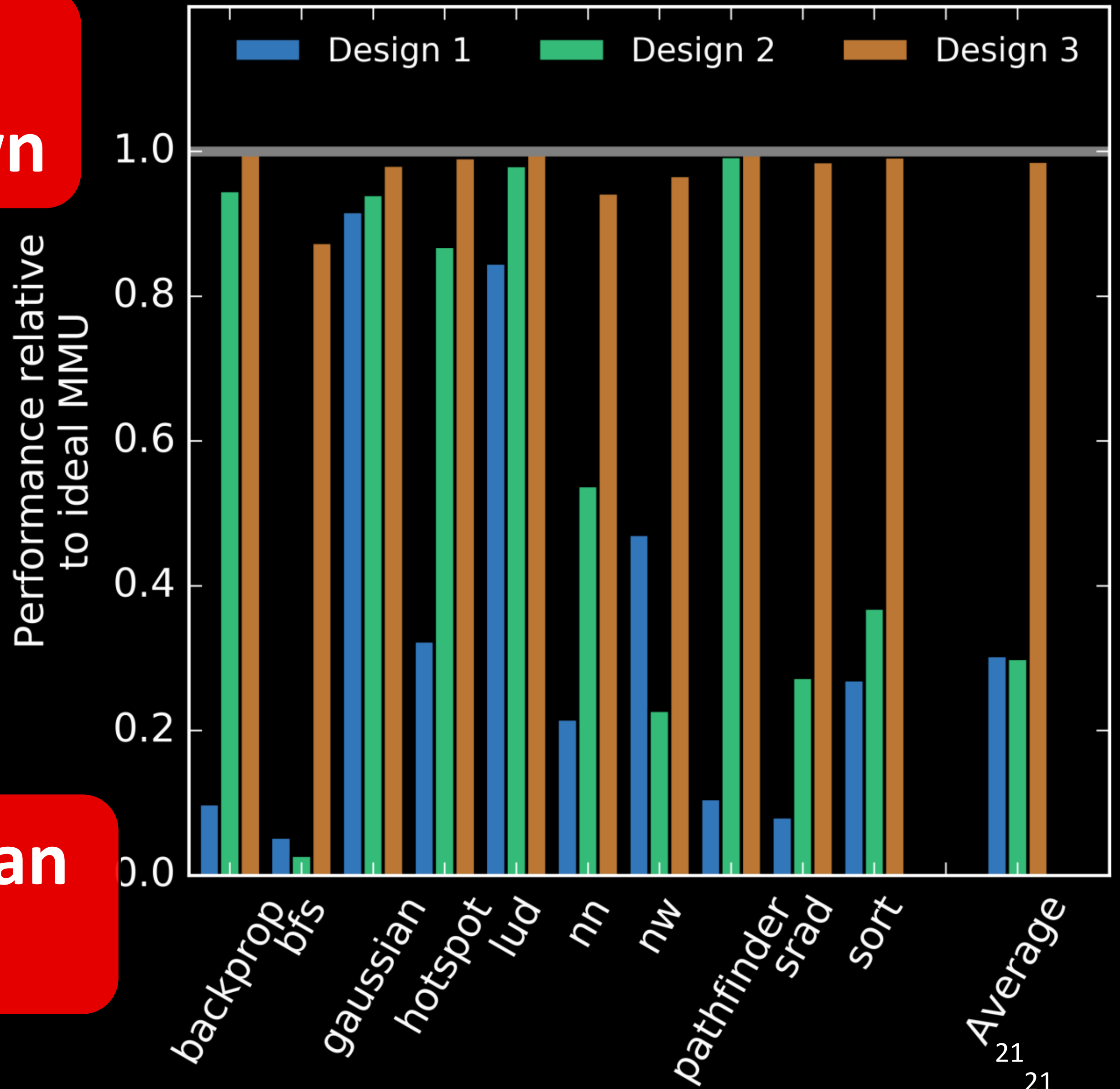


Performance: Low overhead

Worst case: 12% slowdown



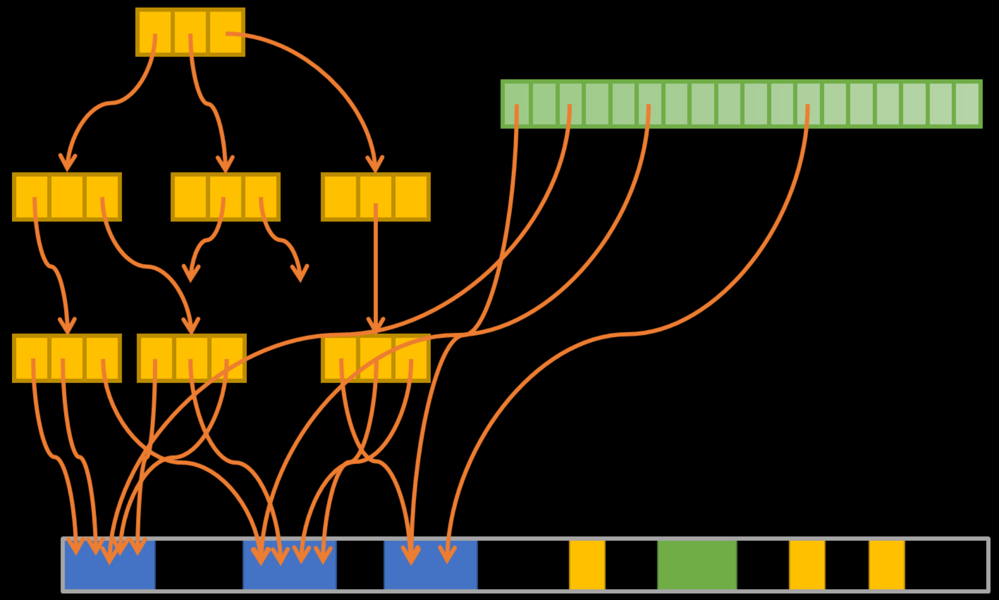
Average: Less than 2% slowdown



21

Consistent pointers

Supporting x86-64 Address Translation for 100s of GPU Lanes



Shared virtual memory is important

Non-exotic MMU design

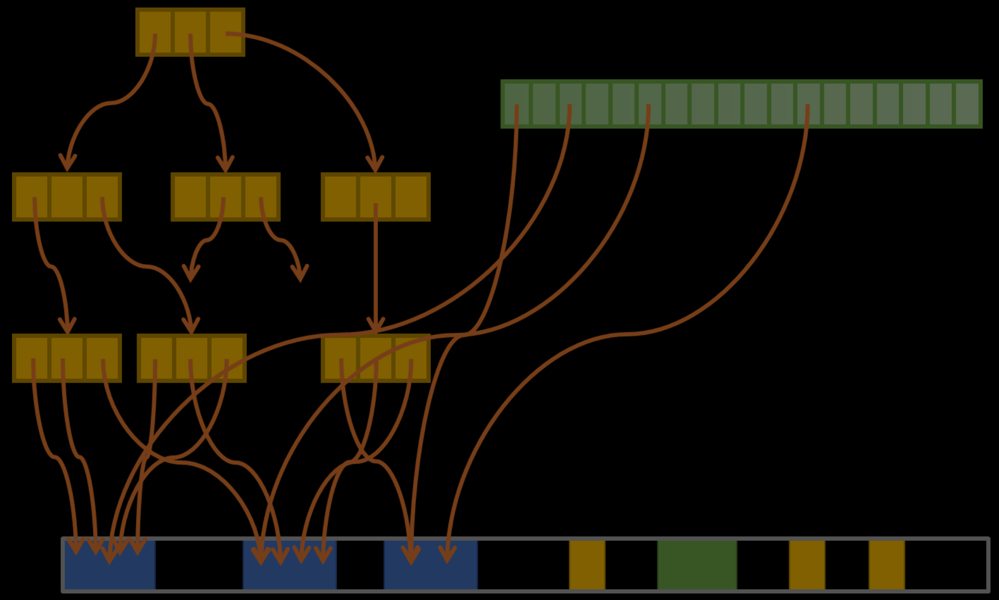
- Post-coalescer L1 TLBs
- Highly-threaded page table walker
- Page walk cache

Full compatibility with minimal overhead

Still room to optimize

Consistent pointers

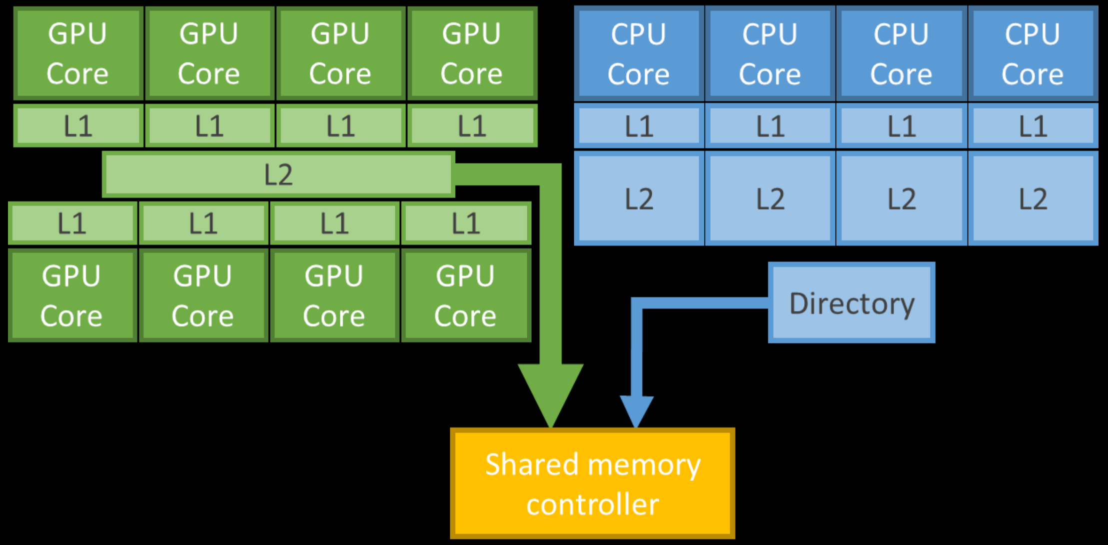
Supporting x86-64 Address Translation for 100s of GPU Lanes



[HPCA 2014]

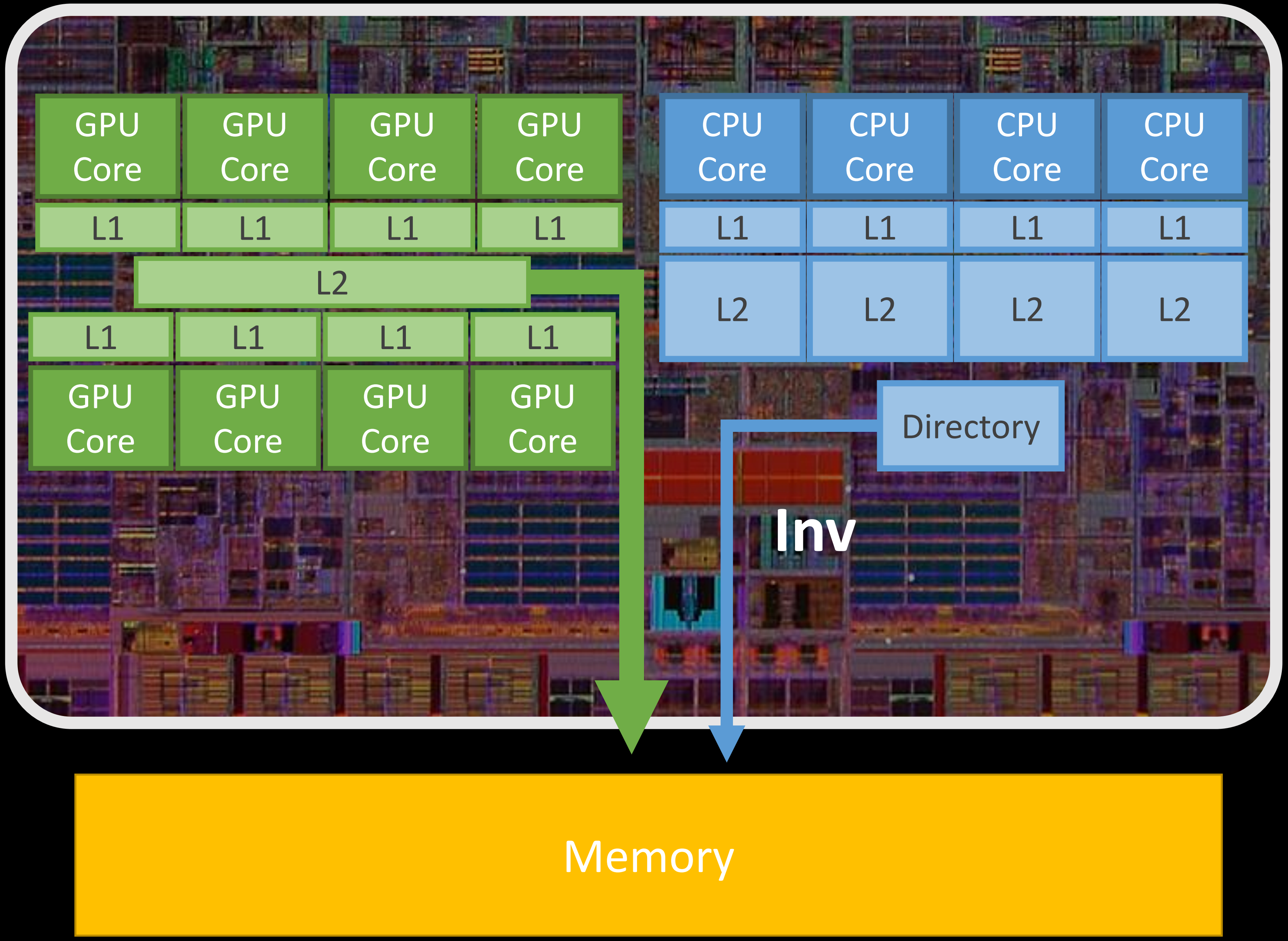
Data movement

Heterogeneous System Coherence



[MICRO 2014]

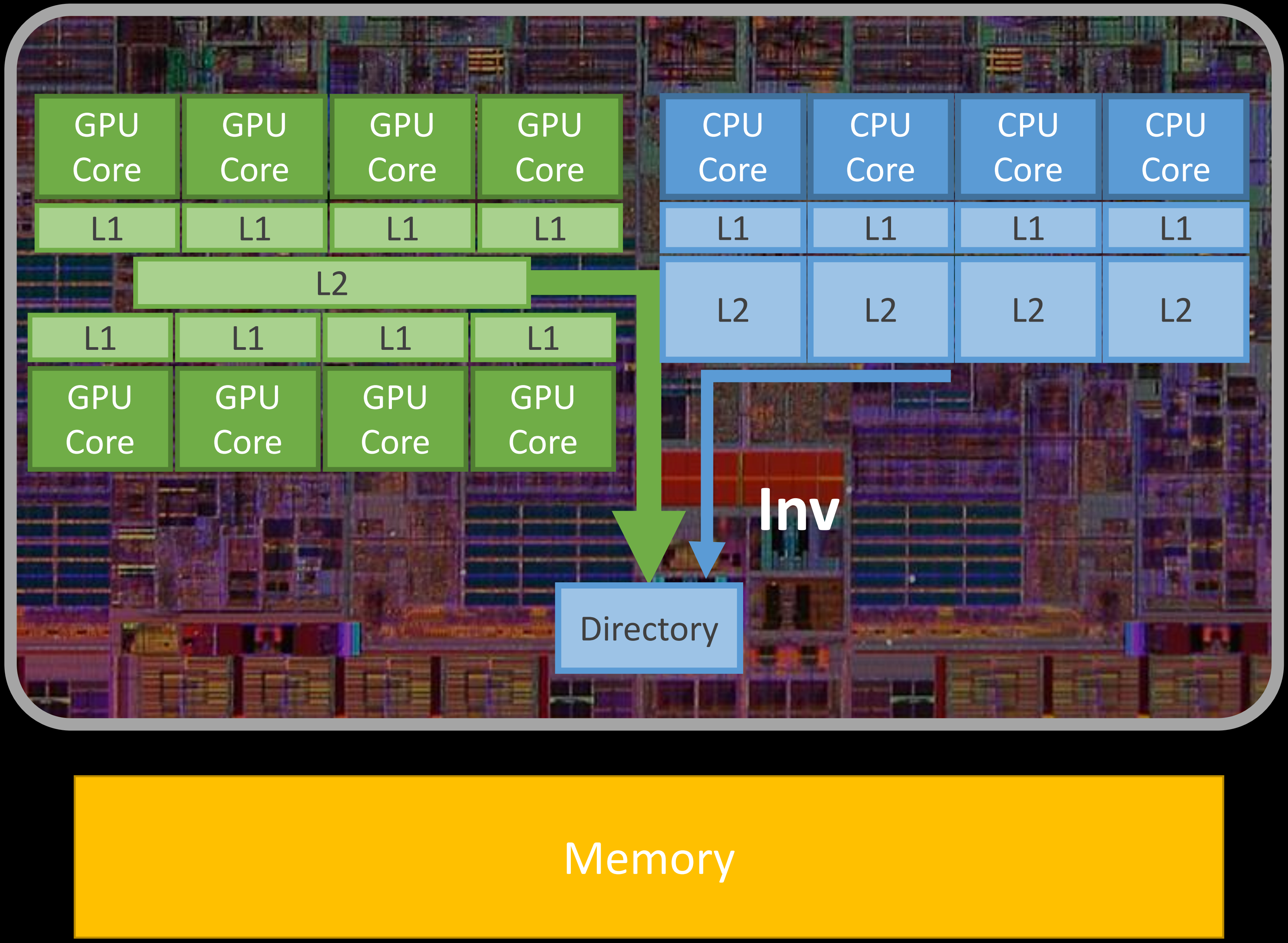
Legacy Interface



1. CPU writes memory
2. CPU initiates DMA
3. GPU direct access

High bandwidth
No directory access

CC Interface

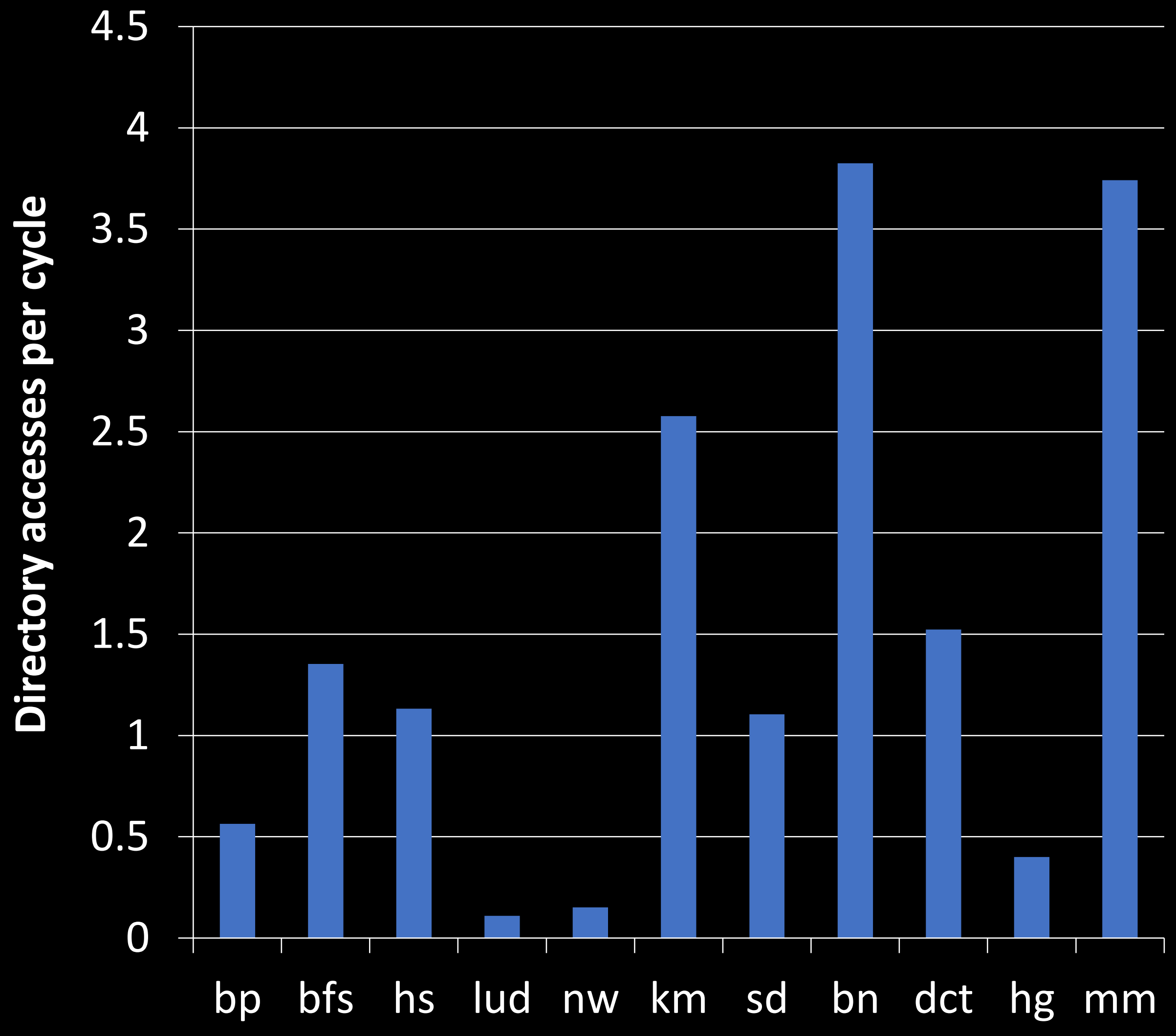


1. CPU writes memory
2. GPU access

Bottleneck: Directory

1. Access rate
2. Buffering

Directory Bottleneck 1: Access rate



Many requests per cycle

Difficult to design multi-ported directory

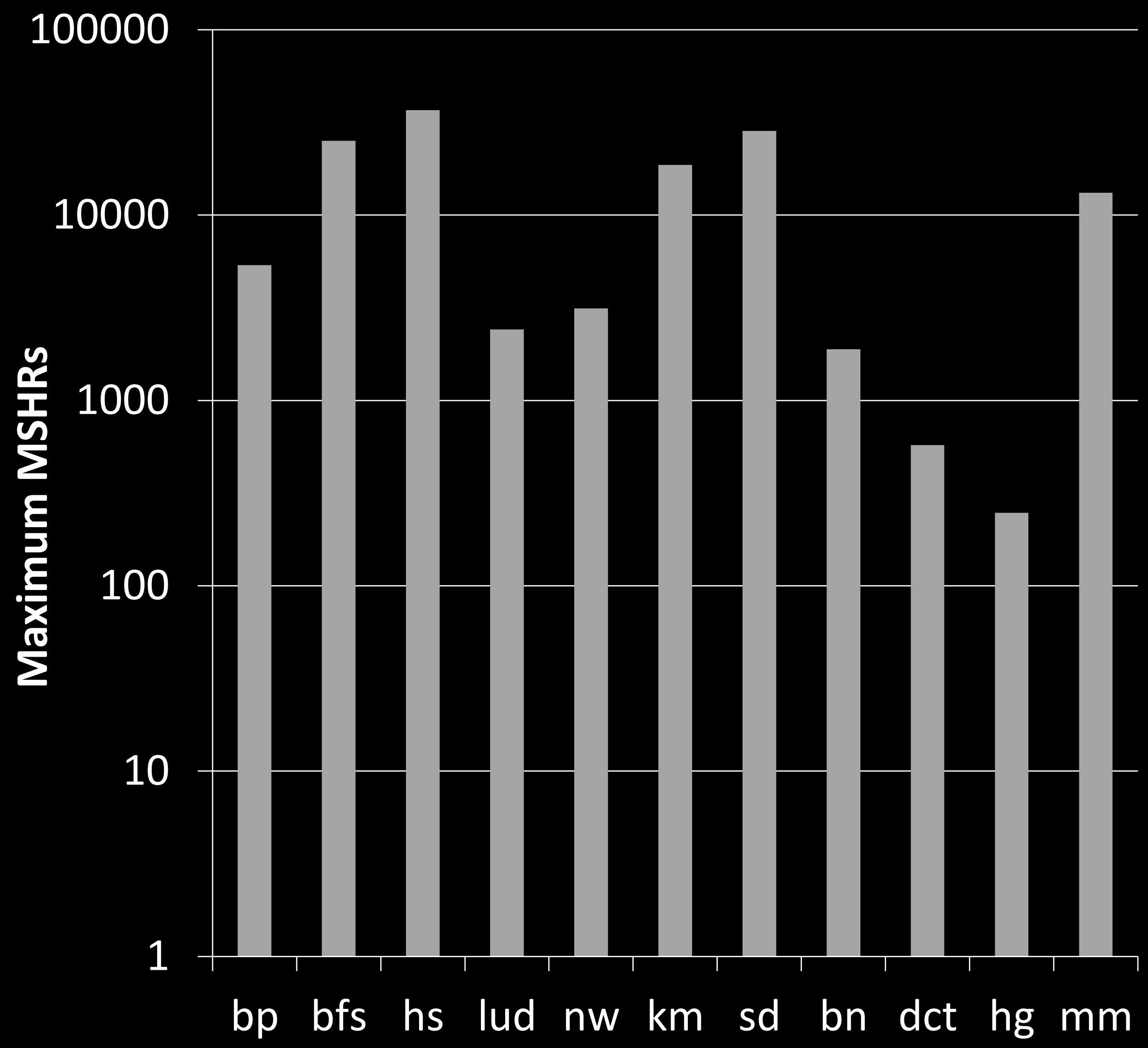
Directory Bottleneck 2: Buffering

Must track many outstanding requests

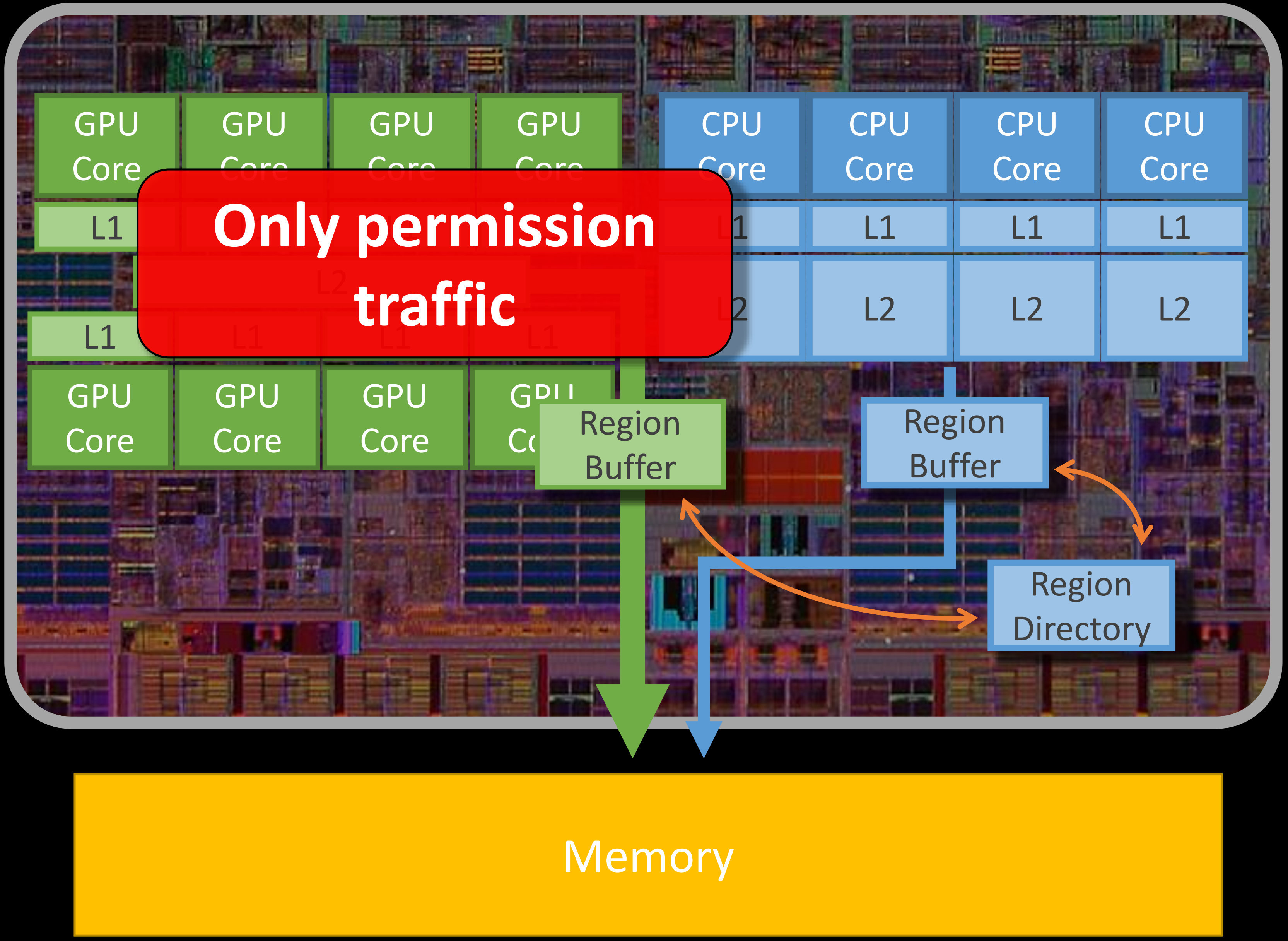
Huge queuing delays

Solution:

Reduce pressure on directory



HSC Design



Goal:

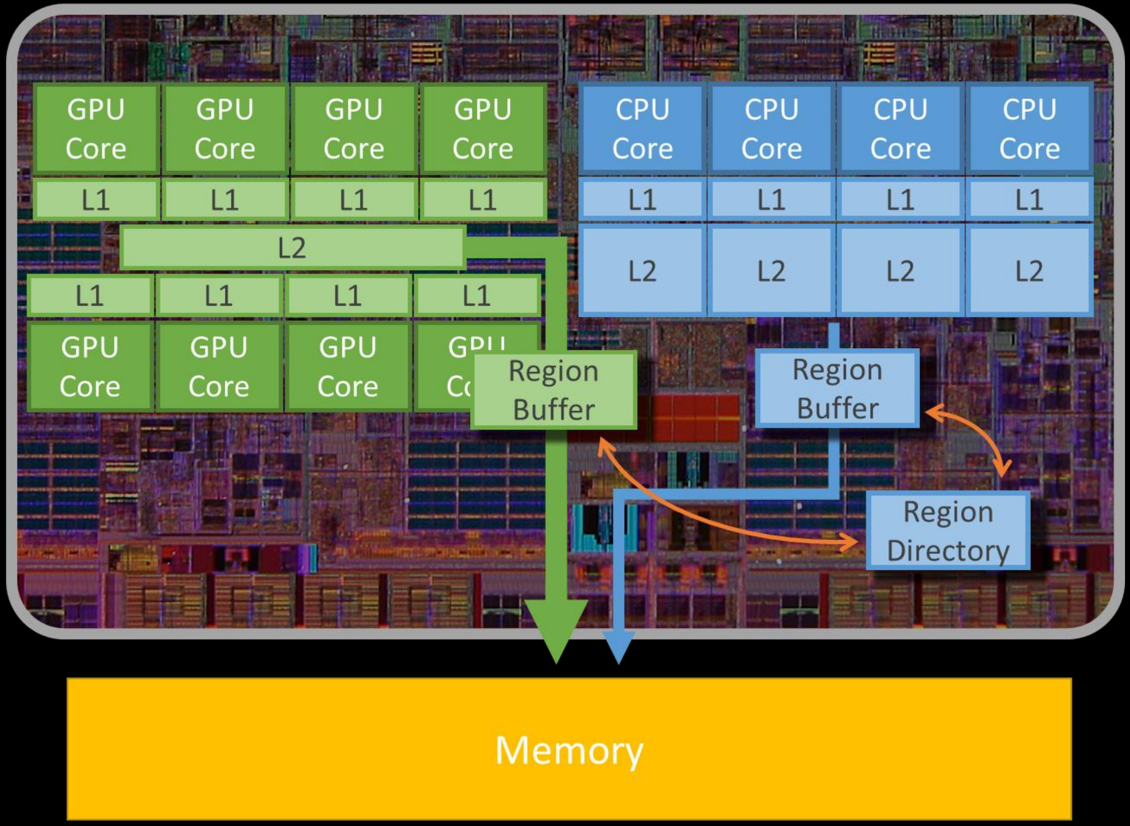
Direct access (B/W)
+ Cache coherence

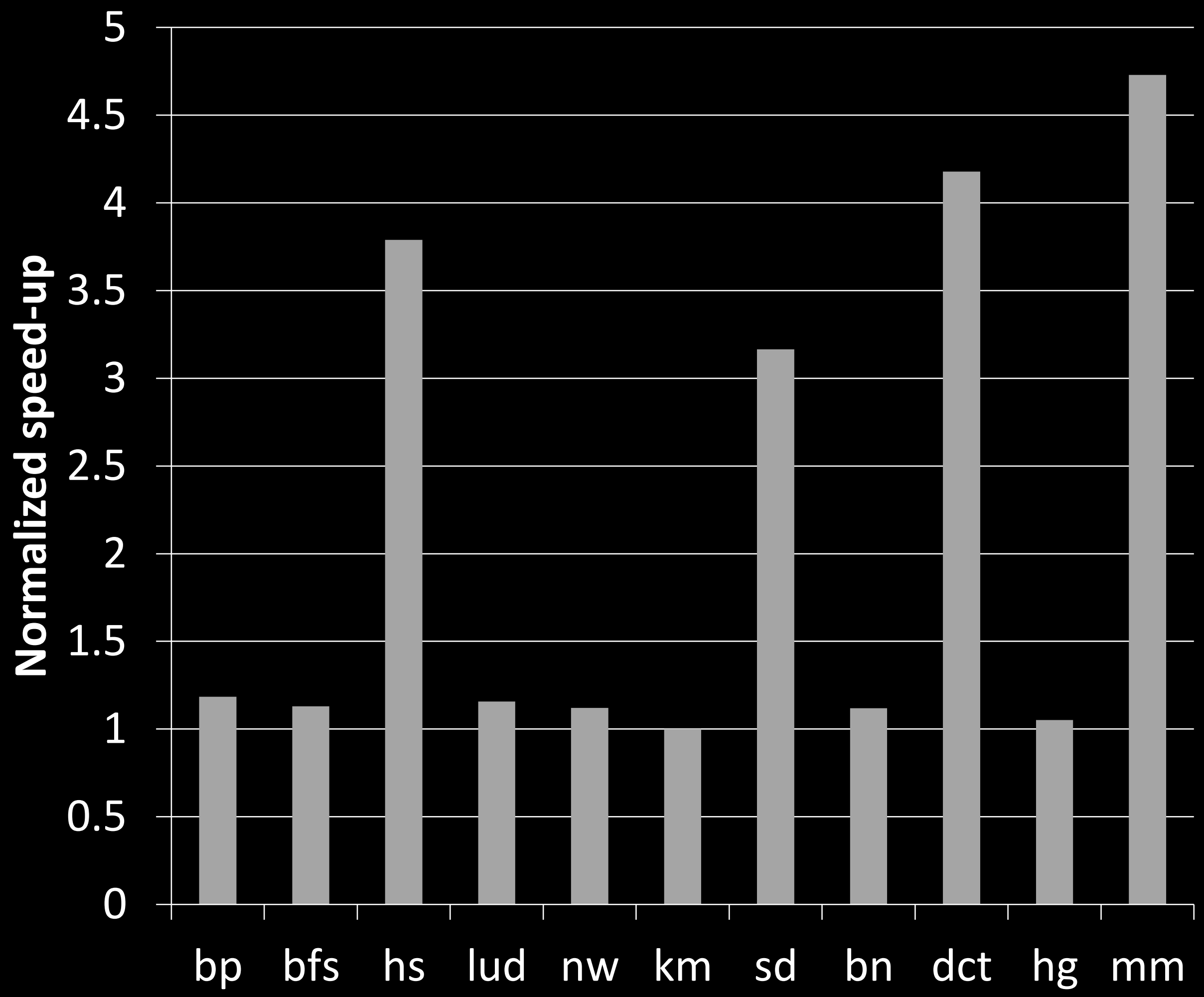
Add:

Region Directory
Region Buffers

Decouples permission from access

HSC: Performance Improvement





Data movement

Heterogeneous System Coherence

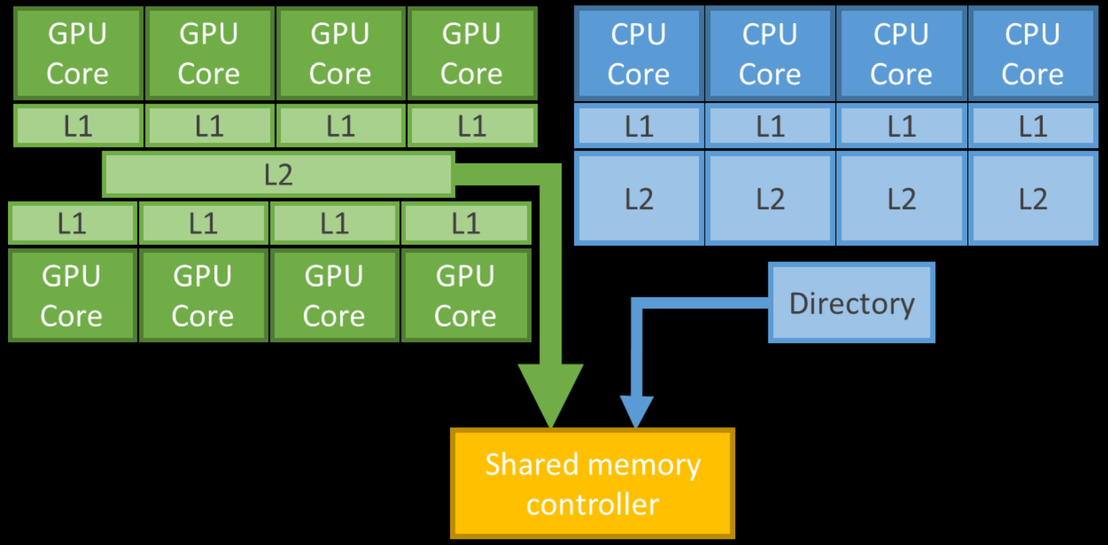
Want cache coherence without sacrificing bandwidth

Major bottlenecks in current coherence implementations

1. High bandwidth difficult to support at directory
2. Extreme resource requirements

Heterogeneous System Coherence

- Leverages spatial locality
- Reduces bandwidth and resource requirements by 95%



Increasing specialization

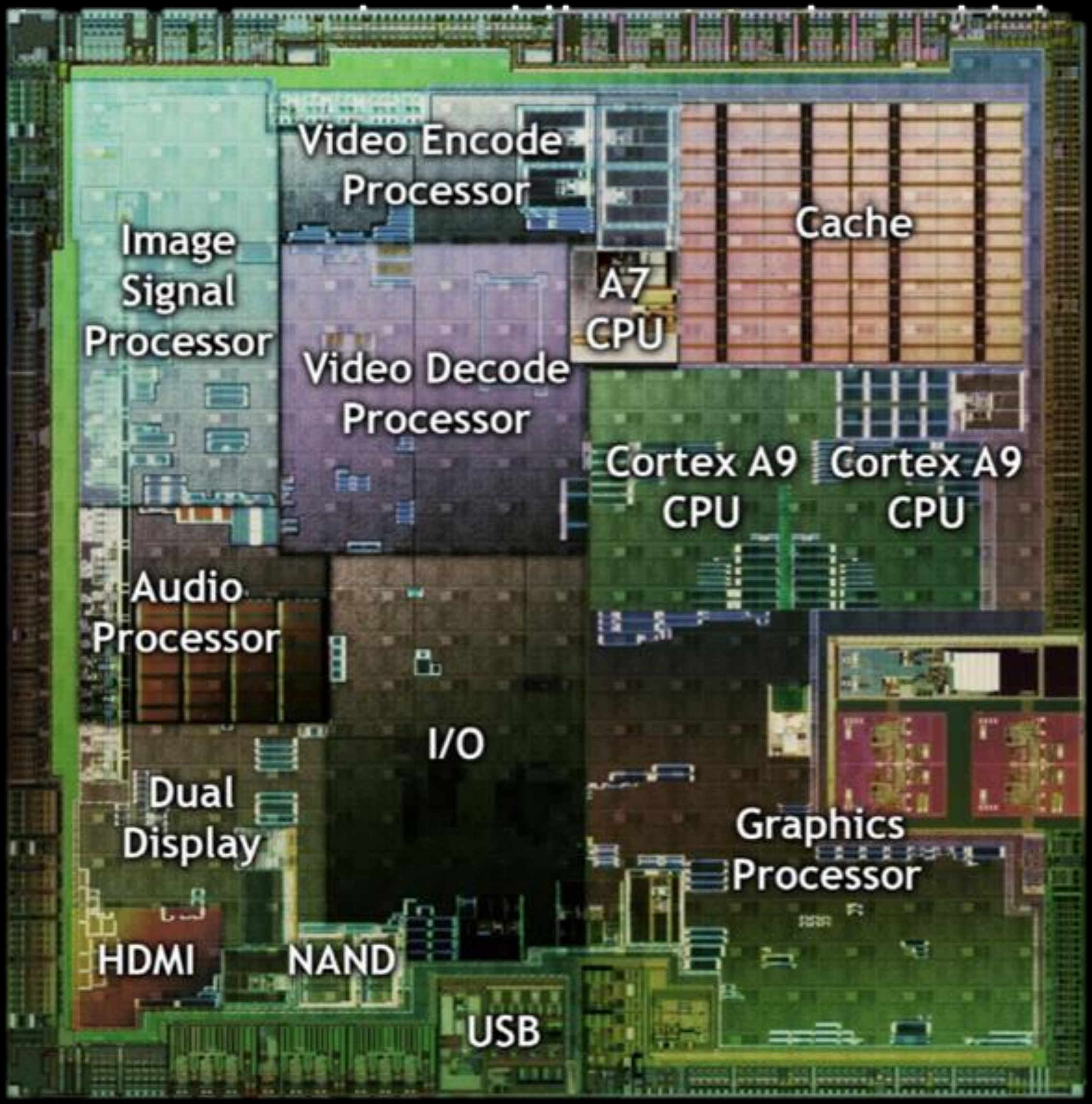
Need to ***program***

these accelerators

Challenges

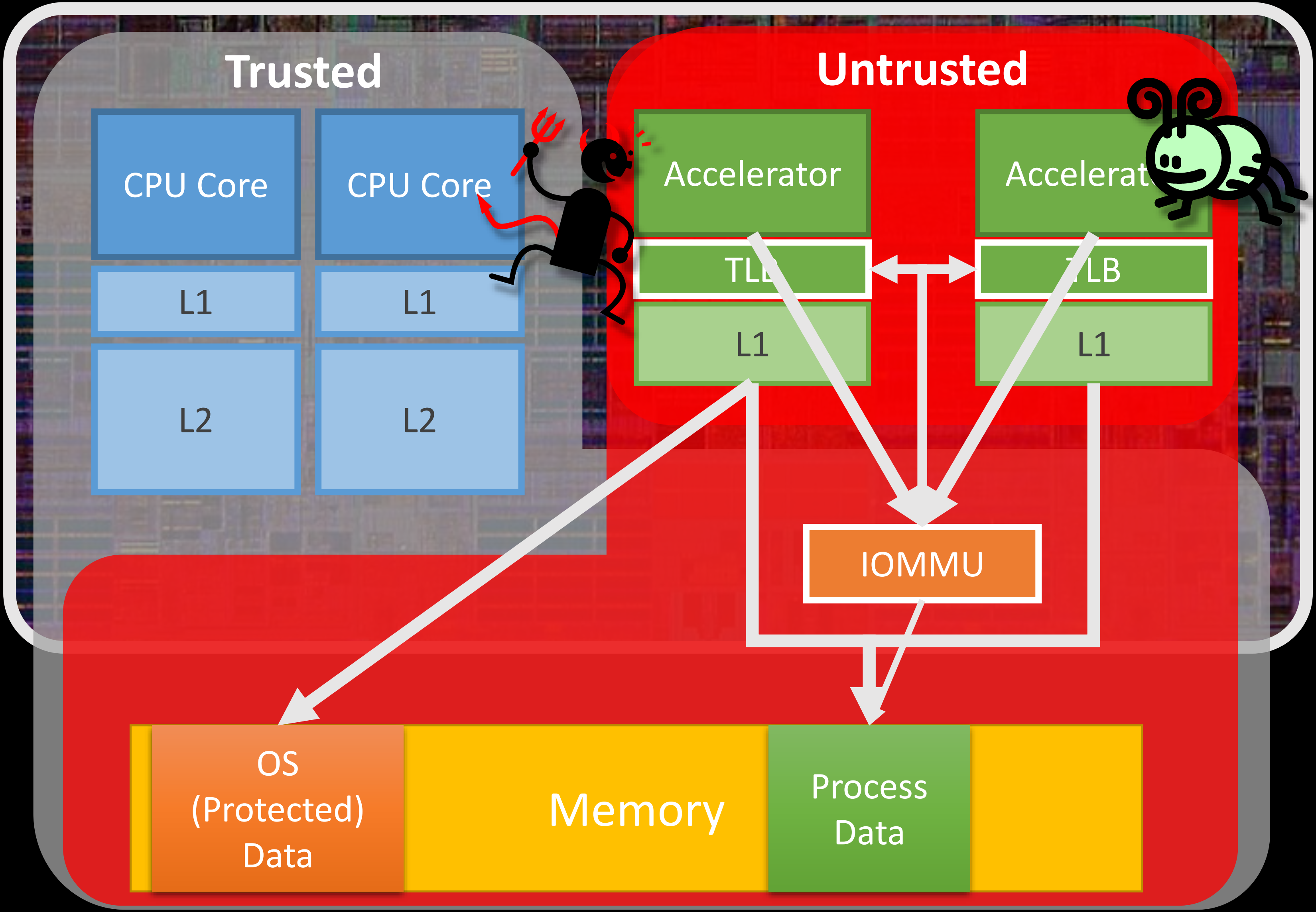
1. Consistent pointers
2. Data movement
3. **Security (Fast)**

This talk: GPGPUs



*NVIDIA via anandtech.com

Security & tightly-integrated accelerators



What if accelerators come from 3rd parties?

Untrusted!

All accesses via IOMMU

Safe

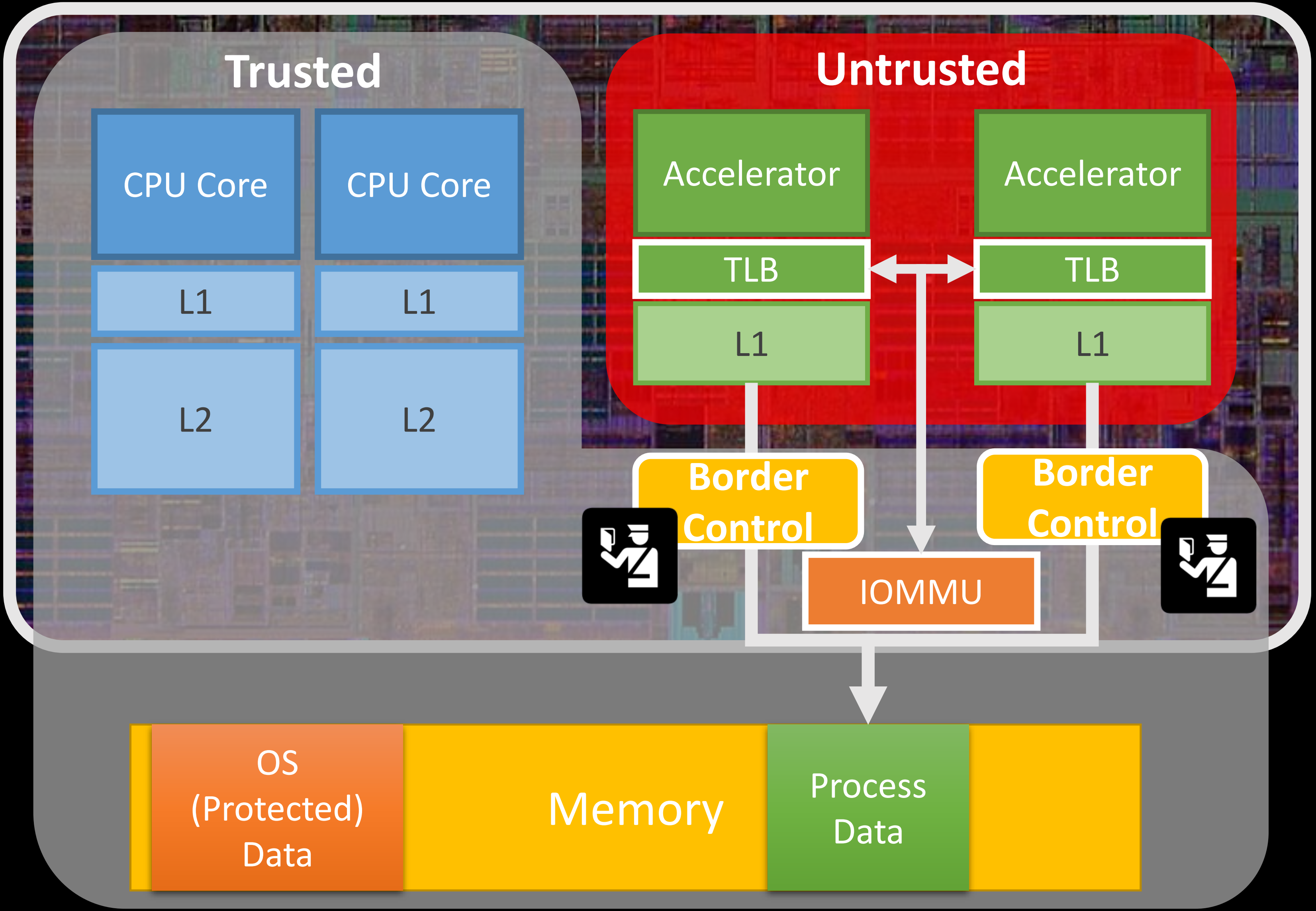
Low performance

Bypass IOMMU

High performance

Unsafe

Border control: sandboxing accelerators [MICRO 2015]

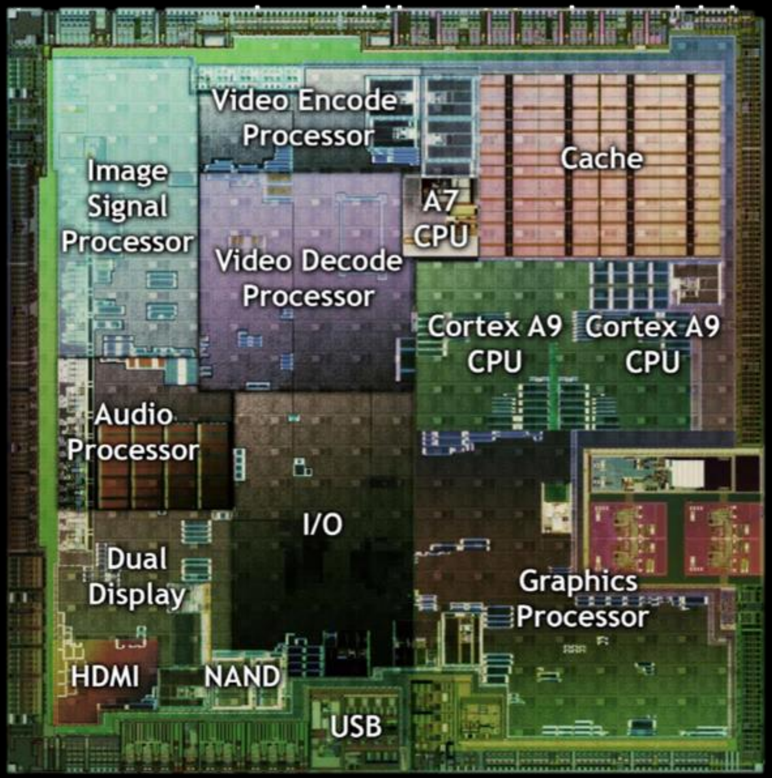


Solution:
Border control

Key Idea: Decouple translation from safety

Safety + Performance

Conclusions



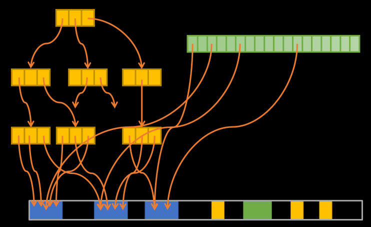
Goal: Enable programmers to use the whole chip

Challenges

1. Consistent addresses
   GPU MMU Design

2. Data movement
   Heterogeneous System Coherence

3. Security
   Border Control

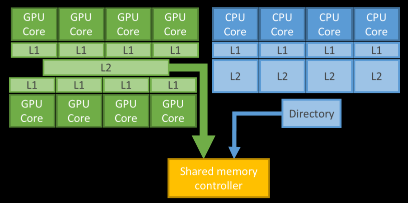


Consistent pointers

Supporting x86-64 Address Translation for 100s of GPU Lanes

[HPCA 2014]

Jason Power, Mark D. Hill, David A. Wood



Data movement

Heterogeneous System Coherence

[MICRO 2013]

Jason Power, Arkaprava Basu*, Junli Gu*, Sooraj Puthoor*, Bradford M. Beckmann*, Mark D. Hill, Steven K. Reinhardt*, David A. Wood

* AMD

Security

Border Control: Sandboxing Accelerators

[MICRO 2015]

Lena E. Olson, **Jason Power**, Mark D. Hill, David A. Wood

I'm on the job market this year! Graduating in Spring

Contact: **Jason Lowe-Power**
powerjg@cs.wisc.edu
cs.wisc.edu/~powerjg

WISCONSIN
UNIVERSITY OF WISCONSIN–MADISON

Other work

Analytic database + Tightly-integrated GPUs

[BPOE 2016]	When to use 3D Die-Stacked Memory for Bandwidth-Constrained Big-Data Workloads	**Jason Lowe-Power**, Mark D. Hill, David A. Wood
[DaMoN 2015]	Towards GPUs being mainstream in analytic processing	**Jason Power**, Yinan Li, Mark D. Hill, Jignesh M. Patel, David A. Wood
[SIGMOD Rec. 2015]	Implications of Emerging 3D GPU Architecture on the Scan Primitive	**Jason Power**, Yinan Li, Mark D. Hill, Jignesh M. Patel, David A. Wood

Simulation Infrastructure

[CAL 2014]	gem5-gpu: A Heterogeneous CPU-GPU Simulator	**Jason Power**, Joel Hestness, Marc S. Orr, Mark D. Hill, David A. Wood

Comparison to CAPI/OpenCAPI

	Same virtual address space	Cache coherent	System safety from accelerator	Assumes on-chip accel.	Allows accel. physical caches	Allows pre-translation
CAPI	Yes ✓	Yes ✓	Yes ✓	No ✓	No ✗	No ✗
My work	Yes ✓	Yes ✓	Yes ✓	Yes ✗	Yes ✓	Yes ✓

Allows for high-performance accelerator optimizations