Rethinking the Control Plane for Chiplet-Based Heterogeneous Systems

Matthew D. Sinclair

University of Wisconsin-Madison

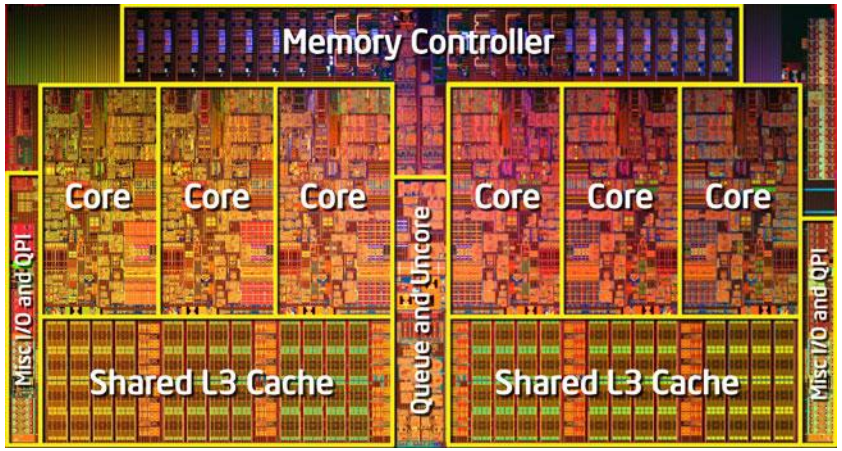
sinclair@cs.wisc.edu

Collaborators: Brad Beckmann (AMD RAD), Dan Negrut, Mike Swift, Shivaram Venkataraman, Zhao Zhang (TACC/Rutgers), and others

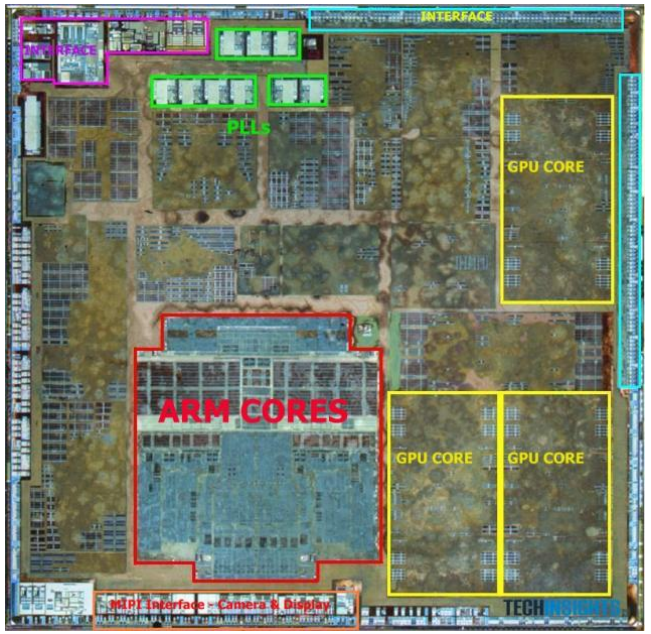
Students: Tanmay Anand, Aatman Borda, Preyesh Dalmia, Akhil Guliani, Jeremy Intan, Rutwik Jain, Yiwei Jiang, Rajesh Shashi Kumar, Rohan Mahapatra, Suchita Pati, Vishnu Ramadas, Kyle Roarty, Prasoon Sinha, Neeraj Surawar, Brandon Tran, Bobbi Yogatama, and others

Silver Bullet for Moore's Law? (CRNCH: Origins?)

Parallelism



Specialization

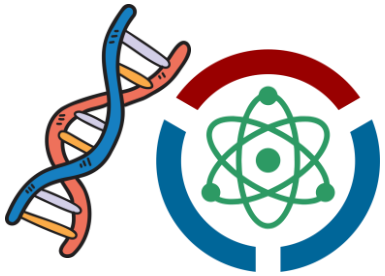


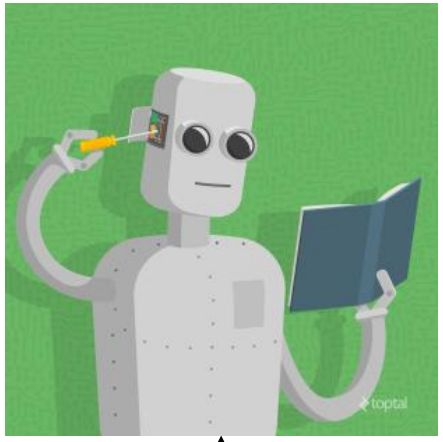
BUT even these systems are reaching die and yield limits

AND applications are evolving at a voracious pace

Applications are increasingly diverse

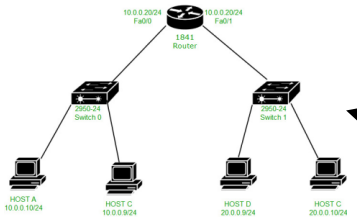
Scientific Computing





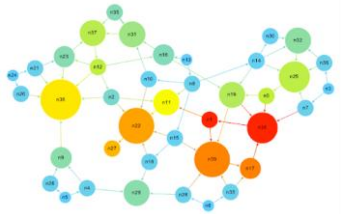
Machine Learning

Packet Processing



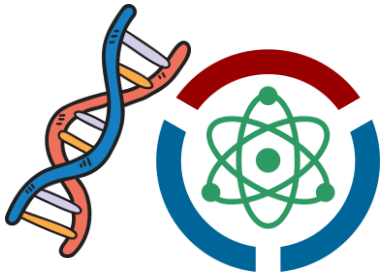
Raytracing

Graph Analytics

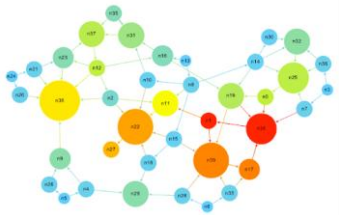


Intelligent Personal Assistants

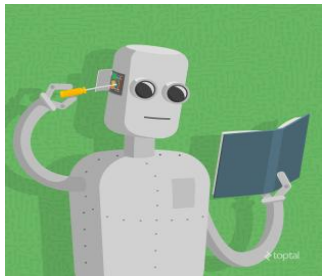
These Applications Drive System Reqs



Scientific Computing



Graph Analytics

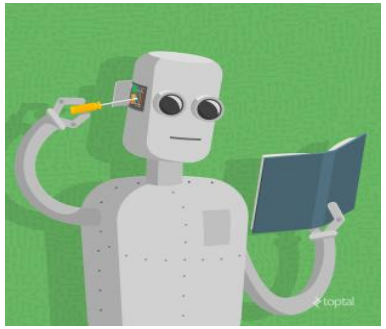


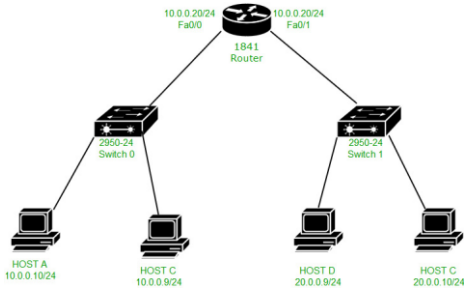
Machine Learning

Often reuse/share data, utilize fine-grained synchronization

Sensitive to NUMA effects in chiplet-based accelerators

These Applications Drive System Reqs





Intelligent Personal Assistants

Machine Learning

Packet Processing

Tight real-time deadlines

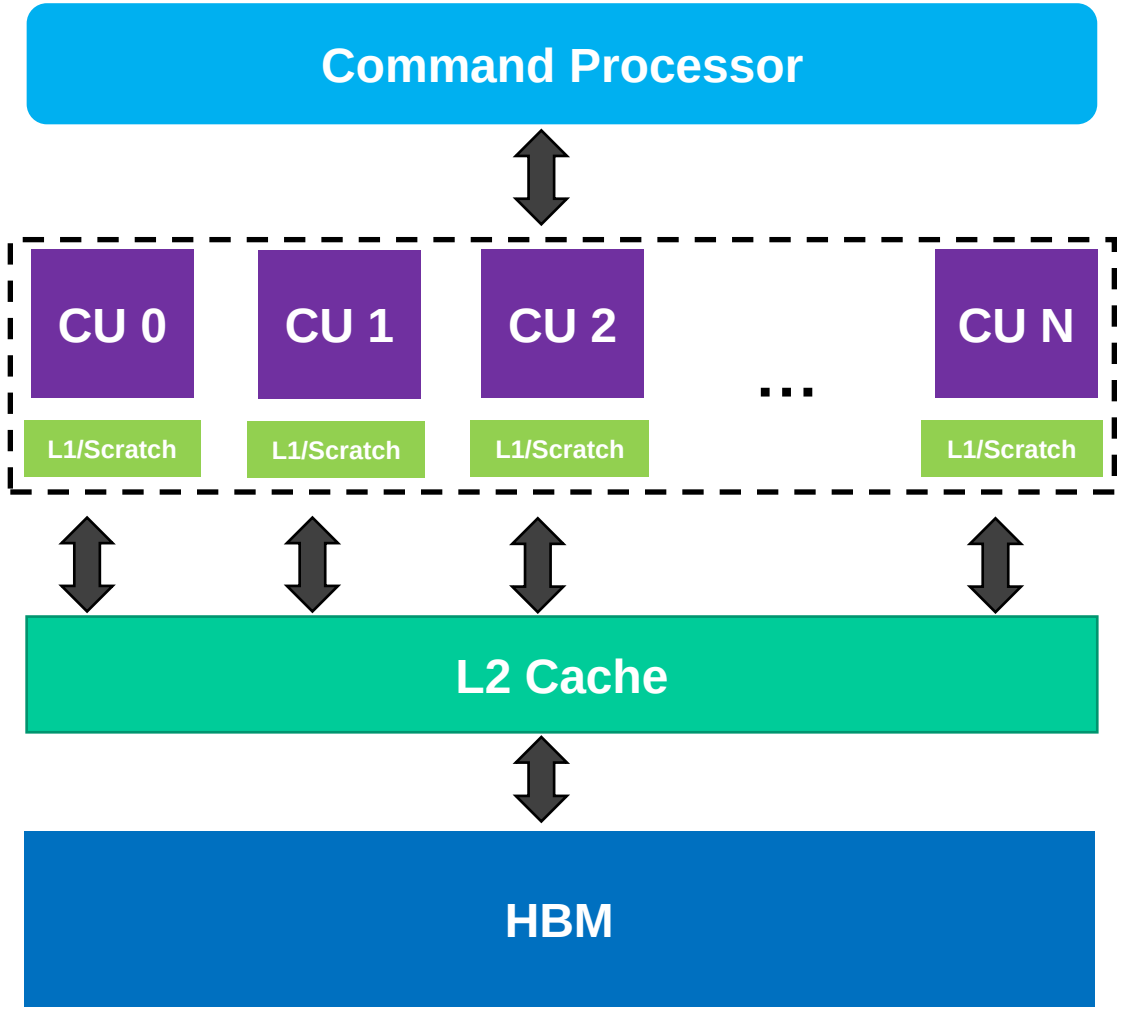
Varying amounts of parallelism: sometimes do not fully utilize accelerators

Multi-tenancy (e.g., datacenters): improved utilization but competing deadlines

Local-only power management

Many challenges!

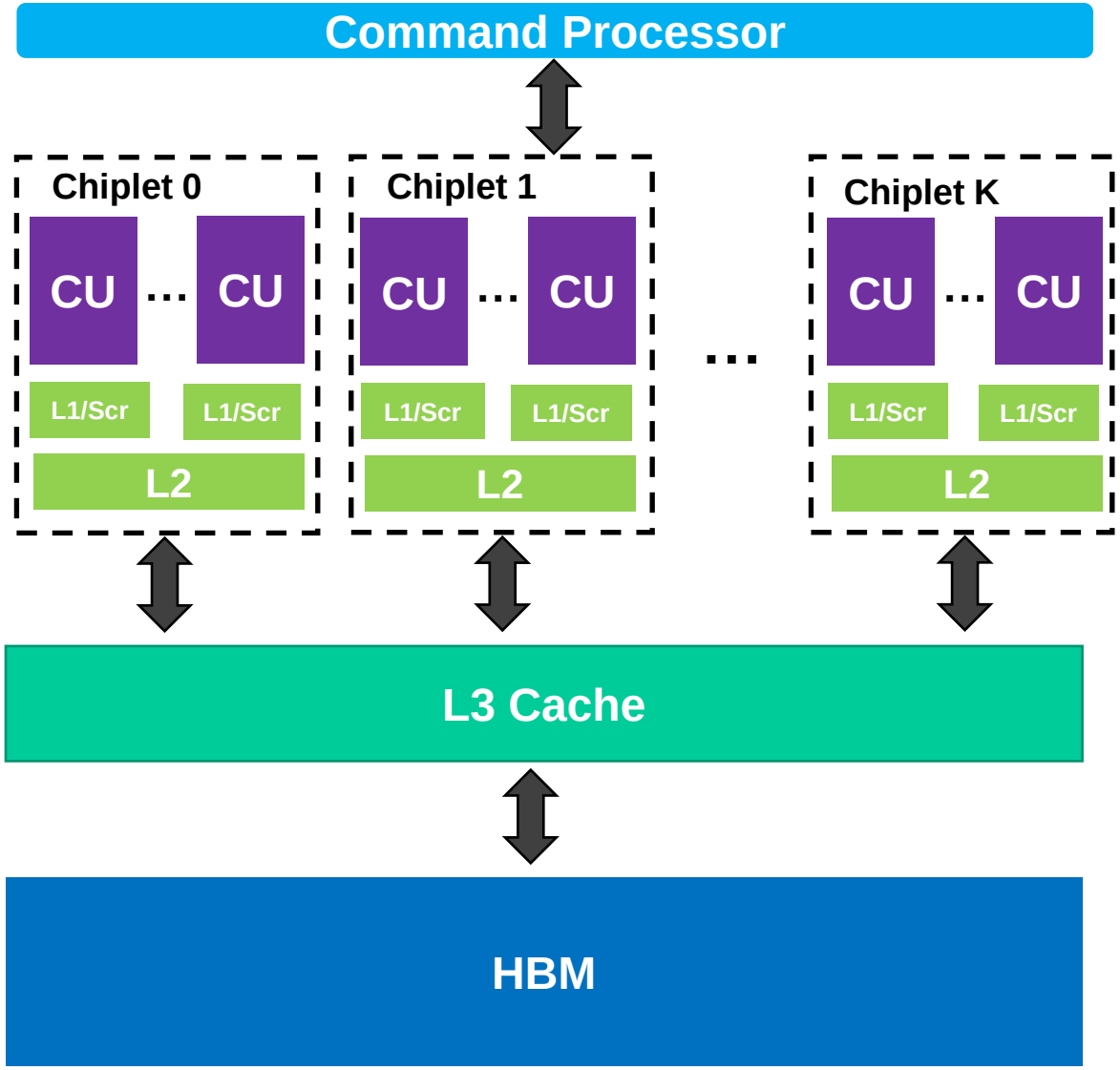
Monolithic Accelerators Reaching Size Limits

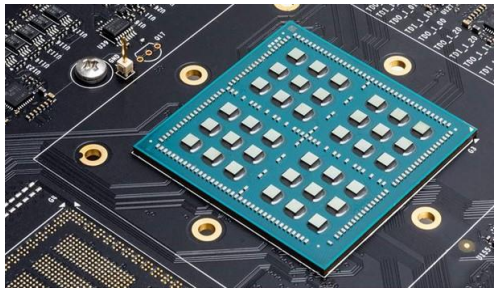


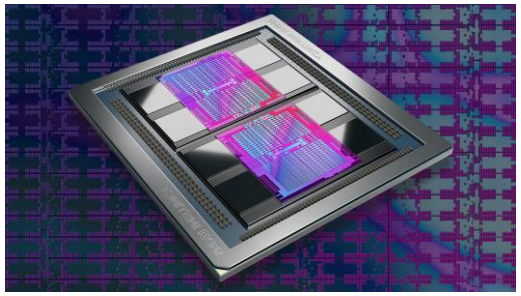
Issue: large dies have lower yields [Khairy MICRO '20]

How to continue scaling accelerator performance?

Chiplet-Based Systems to the Rescue?







AMD & NVIDIA chiplet-based GPUs

Better yield, continue scaling perf

… but new challenges:

How to schedule work efficiently?

How to avoid NUMA penalties?

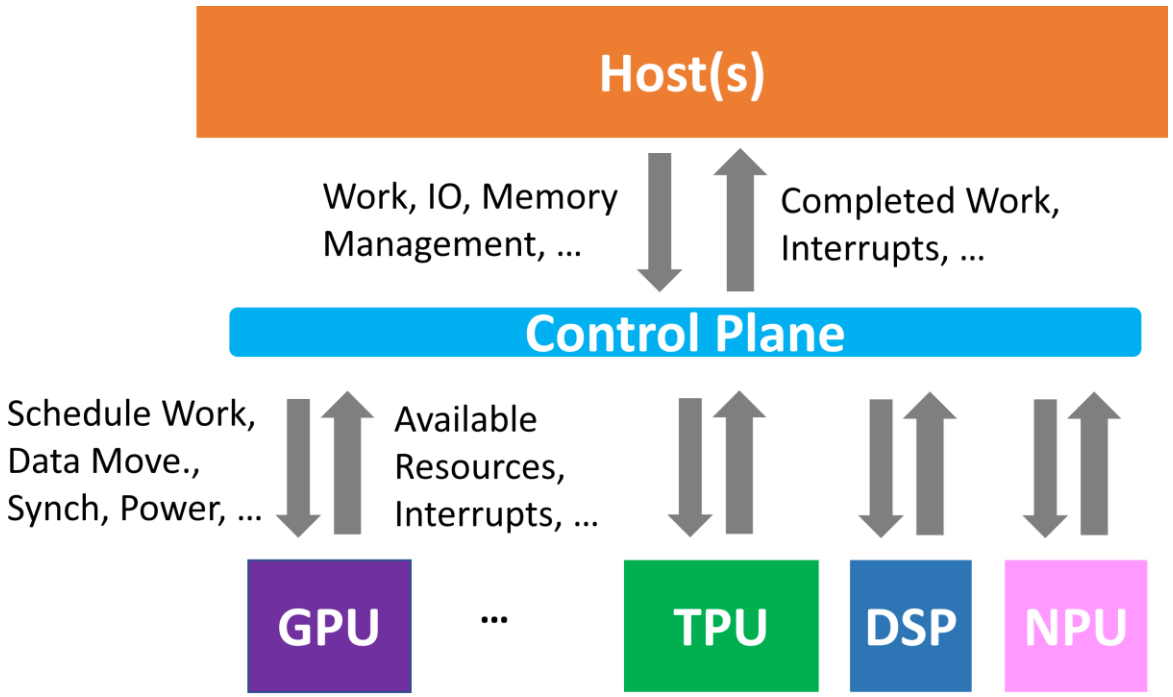
How to handle coh, consist, & synch?

…

Improving Scalability of Heterogeneous Systems

Challenges:	
Deadline-Aware	
Efficient Fine-Grained Synchronization	
Global Power Management	
Intelligent Data Movement	

Enter the Control Plane



- Control plane acts as interface
  - Many accelerators use **same interface style**
  - GPUs: Command Processor (CP)
  - **Has fine-grained info about what's happening …**
  - **… but current systems ignore much of it**
- Our approach:
  - **Rethink** CP design for multi-chiplet systems to improve scalability
  - Utilize CP's info, co-design

Better control plane solves many multi-chiplet heterogeneous system challenges

Improving Scalability of Heterogeneous Systems

Challenges:	Results: (+ tools)
Deadline-Aware	LAX [HPCA '21], SchedMC [in subm.]
Efficient Fine-Grained Synchronization	IFP [ISCA '20], LAB [HPCA '22], HS++ [TPDS '22], T3 [ASPLOS '24], CPElide [MICRO '24], CPElide++ [in subm.]
Global Power Management	SNL [TR '20], TACC [TR '20], ORNL [TR '21], NAGE [SC '22], PAL [SC '24], SpeedBump [in subm.]
Intelligent Data Movement	MI [IISWC '19], SeqPoint [ISPASS '20], Demyst. BERT [IISWC '22], DAB [MICRO '20], LAB [HPCA '22], 2C's [IISWC '23], GOLDYLOC [TACO '25], CAQS [in subm.]

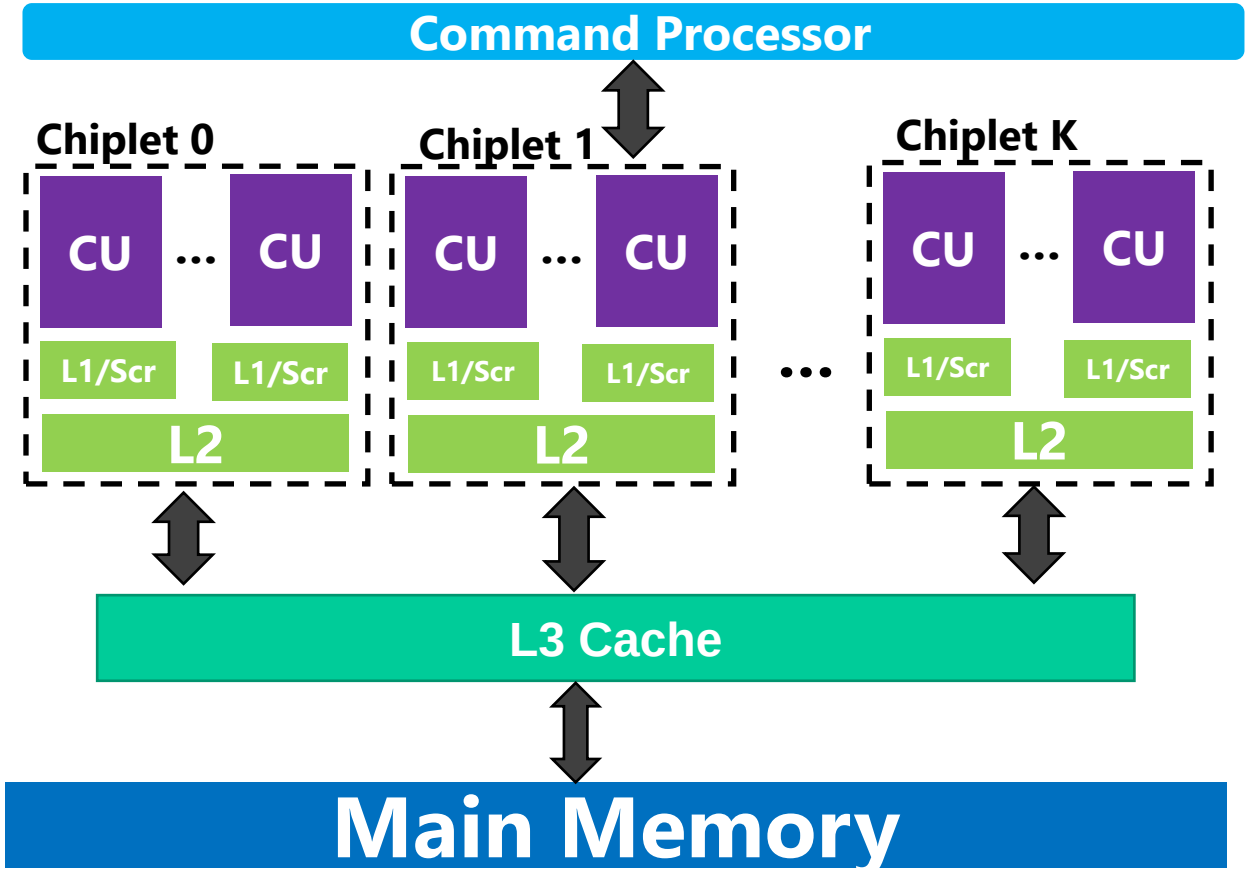
Approach: HW-SW co-design to improve efficiency (hopefully "right" codesign)

Today's Focus: using GPUs as exemplar

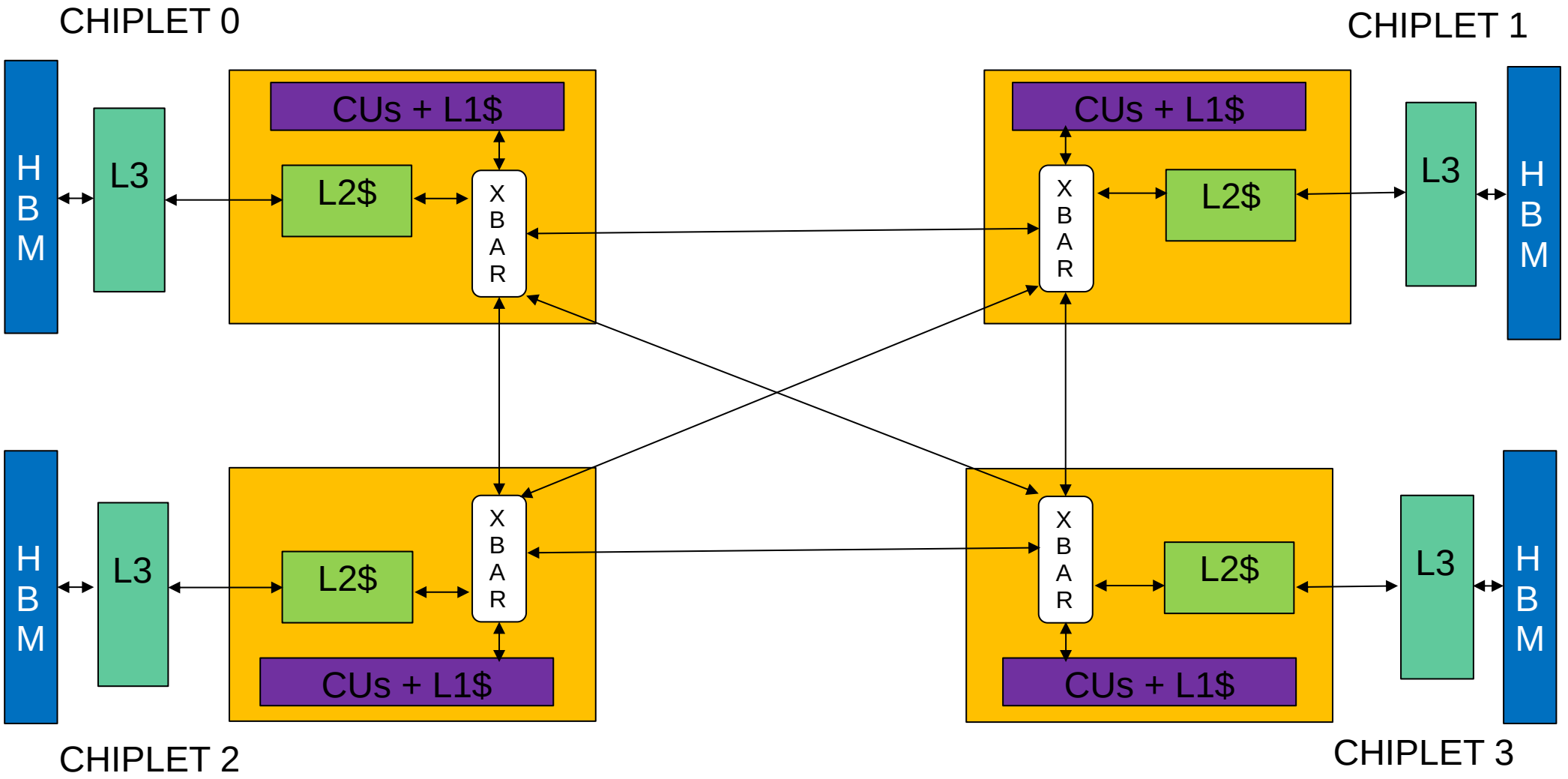
Outline

- Motivation
- **Background**
- CPElide: Efficient Multi-Chiplet GPU Implicit Synch [MICRO ‘24]
- The Next Steps: Building On CPElide
- Conclusion

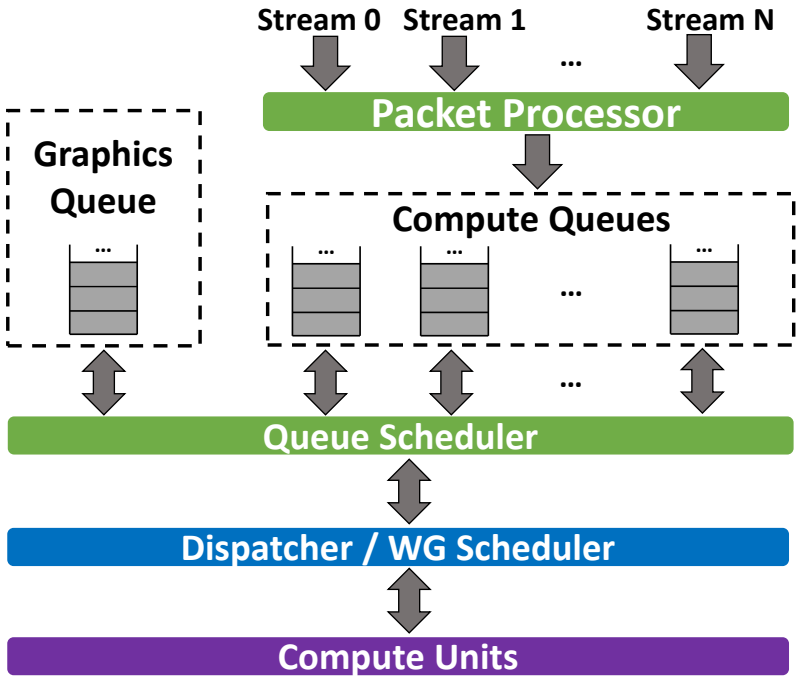
Multi-Chiplet Accelerator Architecture



- Chiplet-based GPUs add additional level to the mem hierarchy->L3 cache
- L2 cache private to a particular chiplet
- Access to data in another chiplet's L2
  - inter-chiplet link
  - Through mem (preceding WB req)


CHIPLET 0
CHIPLET 1
CHIPLET 2
CHIPLET 3
HBM
L3
CUs + L1$
L2$
XBAR

What Are Command Processors?



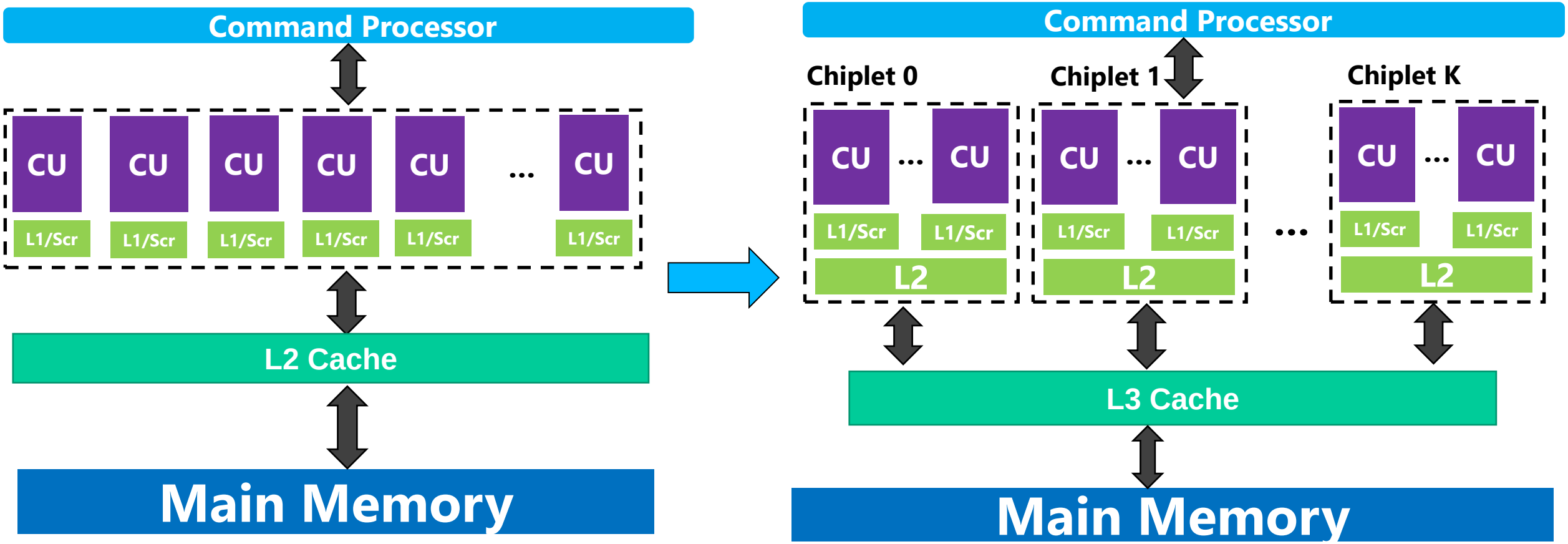
CP Memory

- Interface between host, accelerator
  - Scheduling, Synchronization, Address Translation, Power Management, …
  - Everything control plane is responsible for
- Programmable
- Two primary components:
  - Packet Processor
  - WG Dispatcher (Queue/Stream Scheduler)
- Dispatches WGs to CUs
  - Gets dynamic sched info from code object
  - Includes data structures, addresses
- Initializes RF state
  - Extracts metadata from code object
  - Initializes both scalar and vector regs.

Outline

- Motivation
- Background
- **CPElide: Efficient Multi-Chiplet GPU Implicit Synch [MICRO ‘24]**
- The Next Steps: Building On CPElide
- Conclusion

Move to Multi-Chiplets



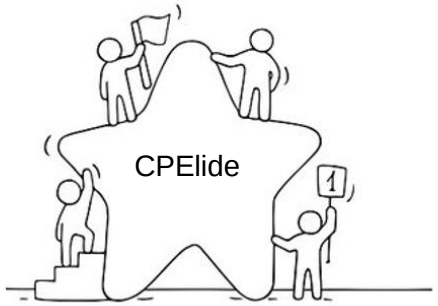
GPU coh./consist. performs implicit synch at kernel boundaries for correctness

Key Takeaway: L2 caches now private → Implicit kernel boundary sync impacts them

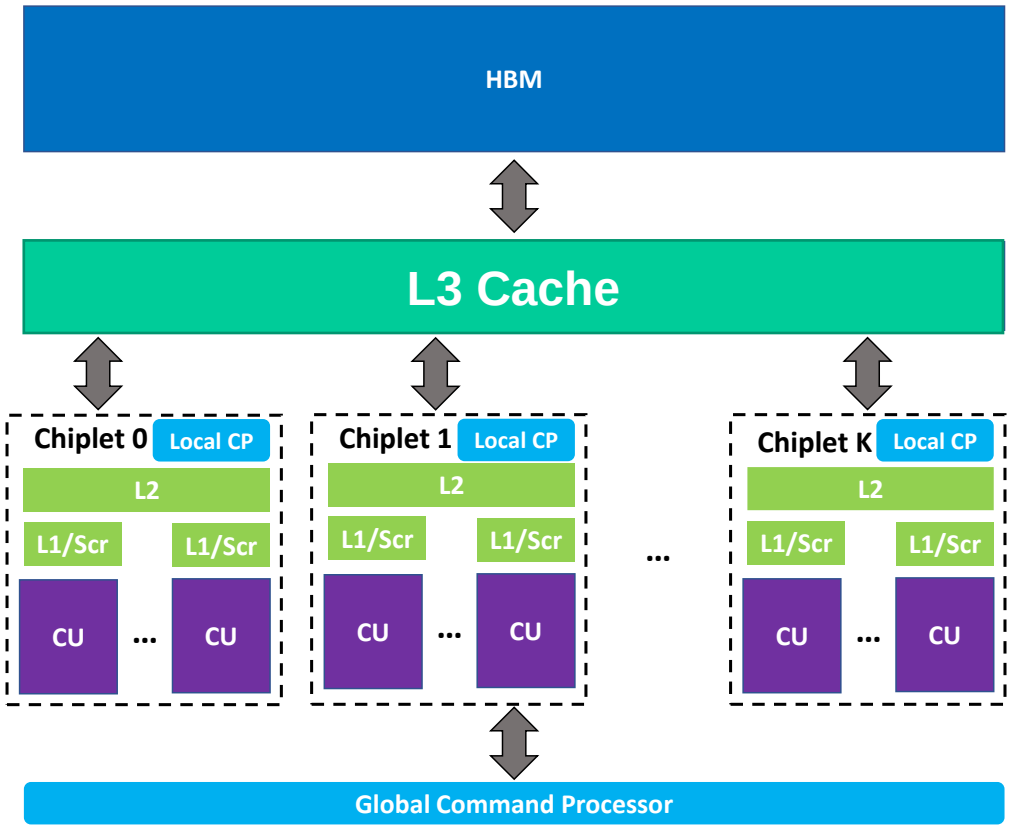
Loss of Inter-Kernel L2 Reuse: 18-54% average performance loss

CPElide Contributions

- Insight: Tracking inter-kernel dependencies inside CP can elide unnecessary acquire/release at kernel boundaries
  - CP has dynamic scheduling information available
  - Programmers/compilers can identify mode of access/ranges for data structures
- CPElide adds dependency tracking table inside CP
  - Leverages CP's available info: coarsely track state of all data structure's accesses per chiplet
  - Up to 39% less execution time (13% avg), 37% less energy (11% avg), 39% less N/W traffic (14% avg) across GPGPU, ML, graph analytics and HPC apps
  - Benefits continue to hold as chiplets scale
  - Reprogrammable to account for future trend changes

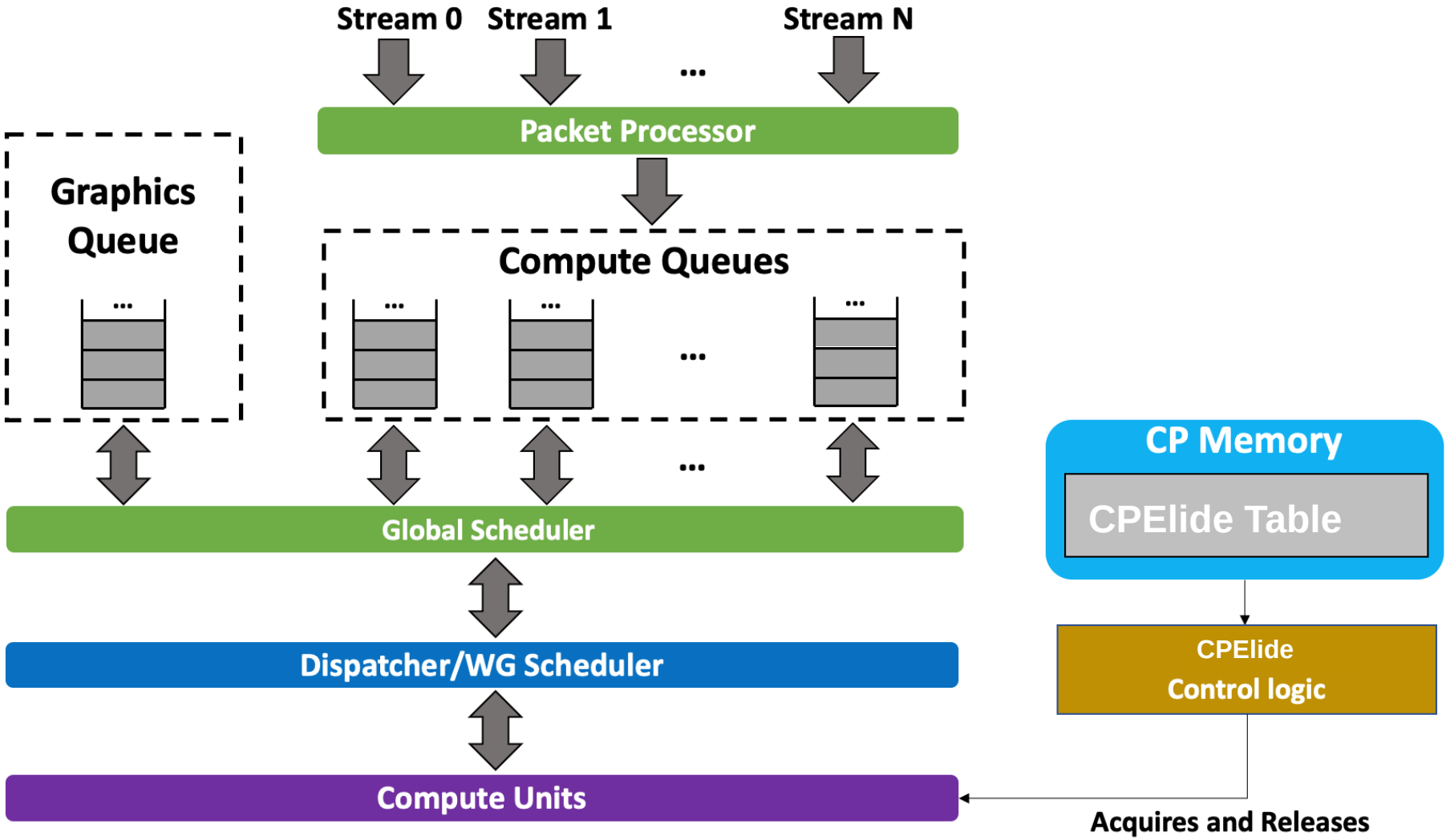


Design: Two-Level CP

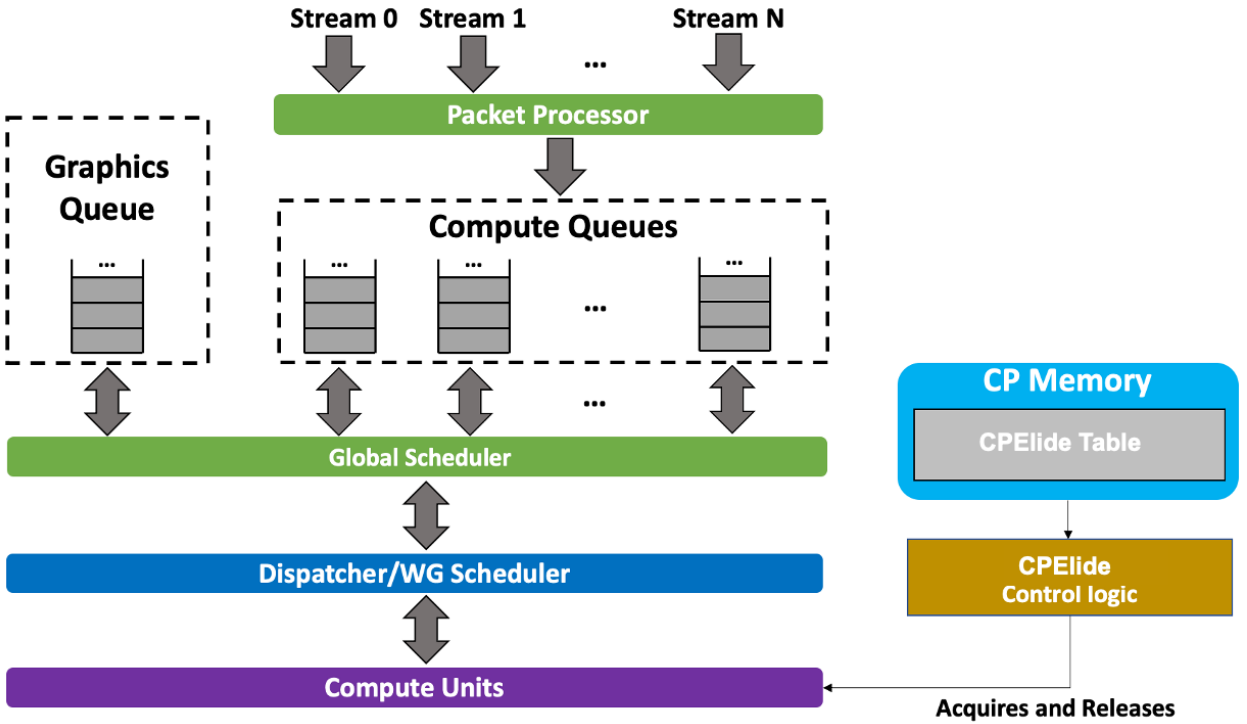


- ❑ Multi-level Command Processor:
  - a global CP
  - a local CP per chiplet
- ❑ Local CP:
  - Controls local scheduling decisions
  - Passes runtime info back to global CP
- ❑ Global CP:
  - Decides work distribution across chiplets
  - Houses CPElide

Design: Global CP

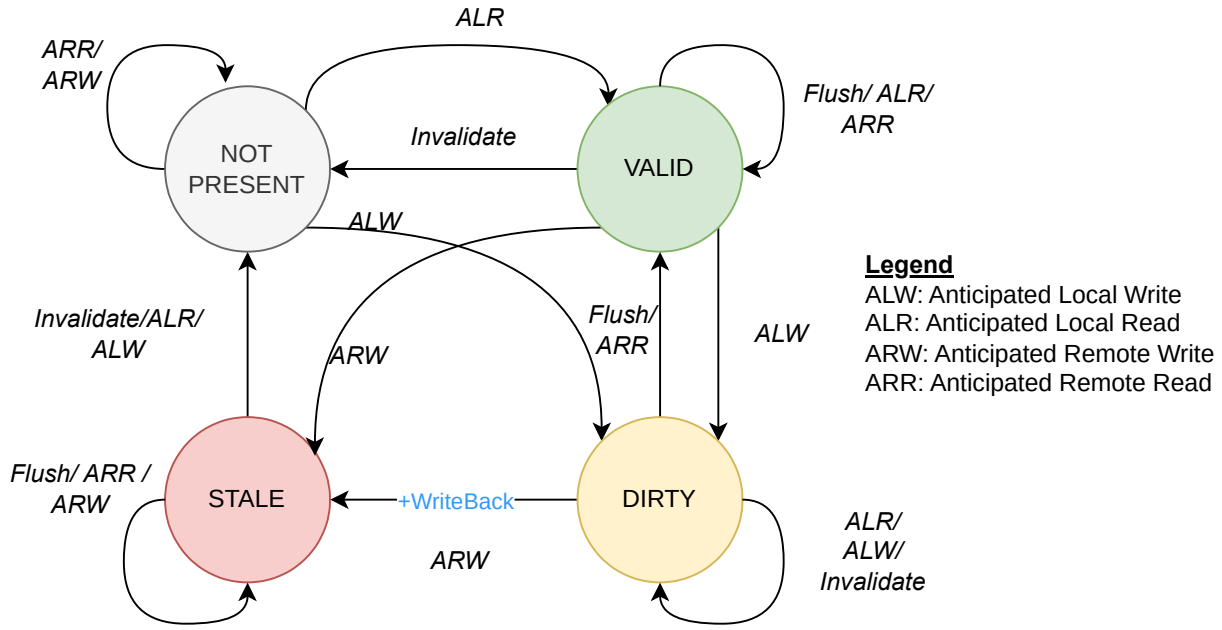


Design: CPElide



- CPElide updated after each kernel launch
  - Gets scheduling info from Global CP
  - Gets mode/range info from kernel packet
- Table has 4 fields per row:
  - DS identifier | Addr range(s) per chiplet | Access mode | bit vector

Design: CPElide State Diagram



- States Not Present (00), Dirty (10), Stale (11), Valid (01)
- State represents state of a data structure at the end of incoming kernel
- This state is conservative to ensure correctness
  - Invalidates/flushes entire L2 cache if a chiplet requires implicit sync
- State changes triggered based on current state & the DS's access modes/ranges

Design: Software Changes

Labeling Access Mode and Access Ranges

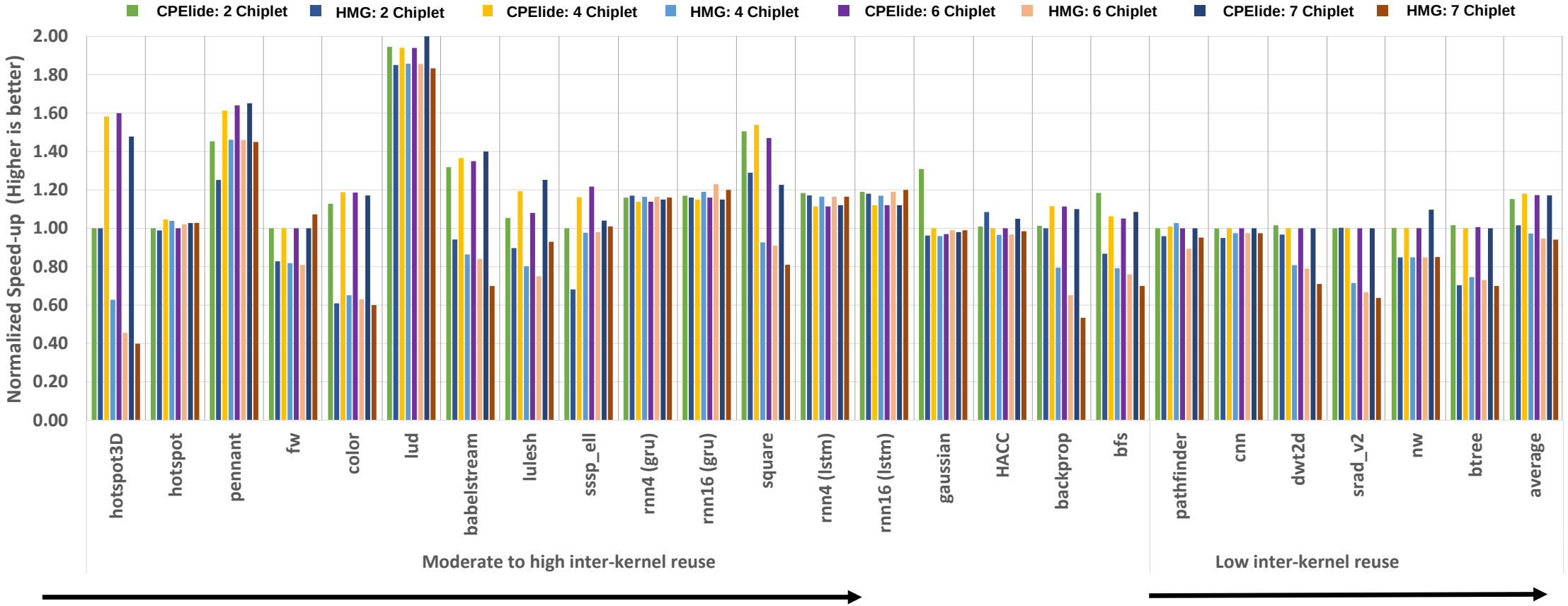
Labeling Access Mode

```
//Square Kernel with Array A (R) as
//input and Array C (R/W) as output
hipSetAccessMode(square, C_d, 'R/W');
hipSetAccessMode(square, A_d, 'R');
hipLaunchKernelGGL(square,..., C_d, A_d, N);
```

```
//numSchedChip: # chiplets to schedule kernel on
typedef tuple<Addr_t, Addr_t, LogicalchipletID>
rangeChiplet;
//Kernel to be launched on 2 chiplets
//Each chiplets works on half of input & output
vector<rangeChiplet> C_ranges(numSchedChip) =
  {make_tuple(C_d[start], C_d[mid], 0),
   make_tuple(C_d[mid+1], C_d[end], 1)};
vector<rangeChiplet> A_ranges(numSchedChip) =
  {make_tuple(A_d[start] , A_d[mid], 0),
   make_tuple(A_d[mid+1] , A_d[end], 1)};
hipSetAccessModeRange(square, C_d, 'R/W', C_ranges);
hipSetAccessModeRange(square, A_d, 'R', A_ranges);
hipLaunchKernelGGL(square,..., C_d,  A_d, N);
```

Programmer/compiler specifies access mode/range for DS per kernel

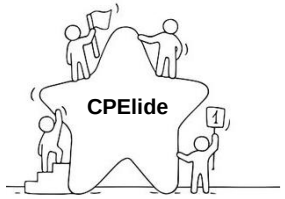
Results: Normalized Performance



- CPElide outperforms both HMG (19% geomean) and Baseline (13% geomean)
  - Able to capture reuse from most apps with moderate-high inter-kernel reuse
  - Apps with little/no reuse suffer perf loss with HMG → more remote traffic from inv
  - HMG also does badly for apps with little to no locality in remote traffic

Similar energy and network traffic benefits

CPElide Conclusions

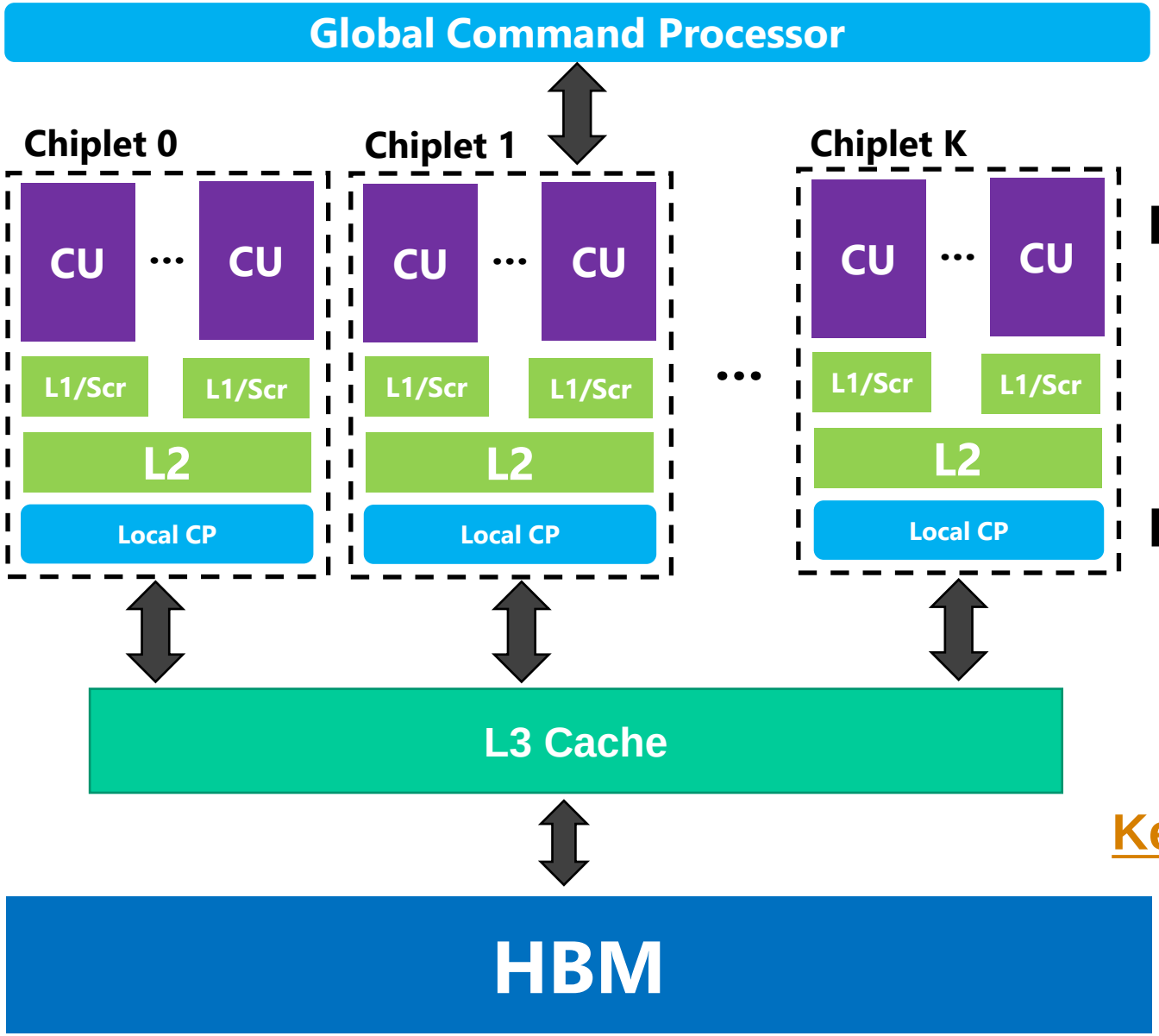


- ❑ Multi-chiplets: Add extra level of cache (L3), making L2 caches private to chiplets
- ❑ Implicit Sync at kernel boundaries leads loss of inter-kernel reuse from L2
- ❑ Insight:
  - Tracking producer consumer dependencies can reduce implicit sync penalty
- ❑ Solution:
  - Redesign CP hierarchy: Global CP houses a dependency tracking table
  - CPElide leverages runtime scheduling info and mode/range info from SW to track data state and elide acquires/releases
  - Effective solutions for kernel with mod/high inter-kernel use, outperforms baseline by 13% and state of the art schemes (HMG) by 19%
  - Scales well, can be reprogrammed to adapt to changing app trends

Outline

- Motivation
- Background
- CPElide: Efficient Multi-Chiplet GPU Implicit Synch [MICRO ‘24]
- **The Next Steps: Building On CPElide**
- Conclusion

Next Steps



Efficient inter-chiplet coh & consist

Chiplet-aware address translation

Rethink HW for global power man.

Handling inter-chiplet NUMA effects

...

Key Insight: Smarter CP can resolve challs.

Better Control Plane Design Can Enable:

- (Even) More efficient fine-grained synchronization
  - CPElide conservatively flushes/invalidates entire L2 on implicit sync
  - **Insight**: Use CP's fine-grained tracking info to perform page granularity implicit sync
  - **Prelim. Results**: 41% geomean energy reduction, same perf vs. CPElide
- Scheduler-Coherence co-design to combat NUMA effects
  - Global CP queue sched ignores CPElide's info → NUMA effects
  - **Insight**: Leverage Global CP's info to co-design queue scheduler and CPElide
  - **Prelim. Results**: geomean 6-30% perf, 19-36% energy, 61-80% NW traffic improvements
- Balancing Deadline-Awareness and Locality
  - Often run concurrent jobs with competing deadlines + switch chiplets → NUMA effects
  - **Insight**: CP info can be used to design queue schedulers that balance locality & deadlines
  - **Prelim. Results**: Completes latency-sensitive jobs up to 2.5X sooner

Outline

- Motivation
- Background
- CPElide: Efficient Multi-Chiplet GPU Implicit Synch [MICRO ‘24]
- The Next Steps: Building On CPElide
- **Conclusion**

Conclusion

- Need to rethink heterogeneous system design to continue scaling
  - Die and yield limitations → chiplet-based accelerators now mainstream
  - Applications increasingly diverse
  - NUMA effects and other challenges hamper efficient scaling
- Opportunity: rethink existing hardware and software support
- Heterogeneous systems must rethink control plane for chiplets
  - Utilizing existing information can significantly reduce overheads
  - **Broadly applicable** to many challenges: deadlines, NUMA effects, synch, …
  - Low area overhead & reprogrammable – potential “firmware” style patches
- **More efficient, intelligent, scalable heterogeneous systems**