CPElide: Efficient Multi-Chiplet GPU Coherence

Preyesh Dalmia[*,^], Rajesh Shashi Kumar[#,^], and Matthew D. Sinclair[^]
* NVIDIA #ARM ^University of Wisconsin-Madison

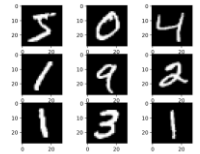
pdalmia@nvidia.com

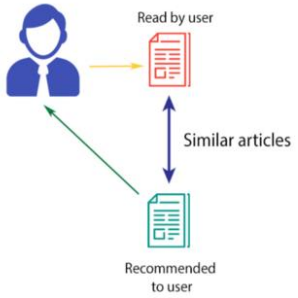
Outline

- **Introduction**
- **Motivation**
- **Contributions**
- **Background**
- **Hardware Design for CPElide**
- **Software Changes for CPElide**
- **Results and Analysis**
- **Conclusion**

Emerging GPGPU applications

DNN Training

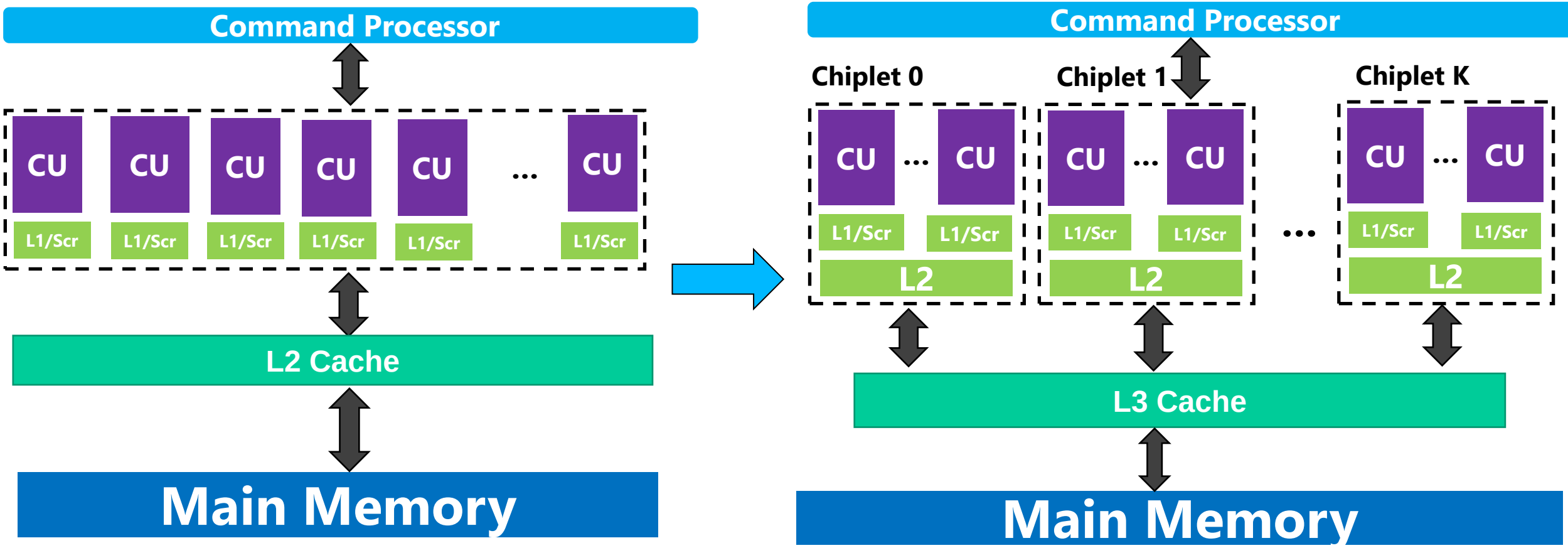




Graph Analytics

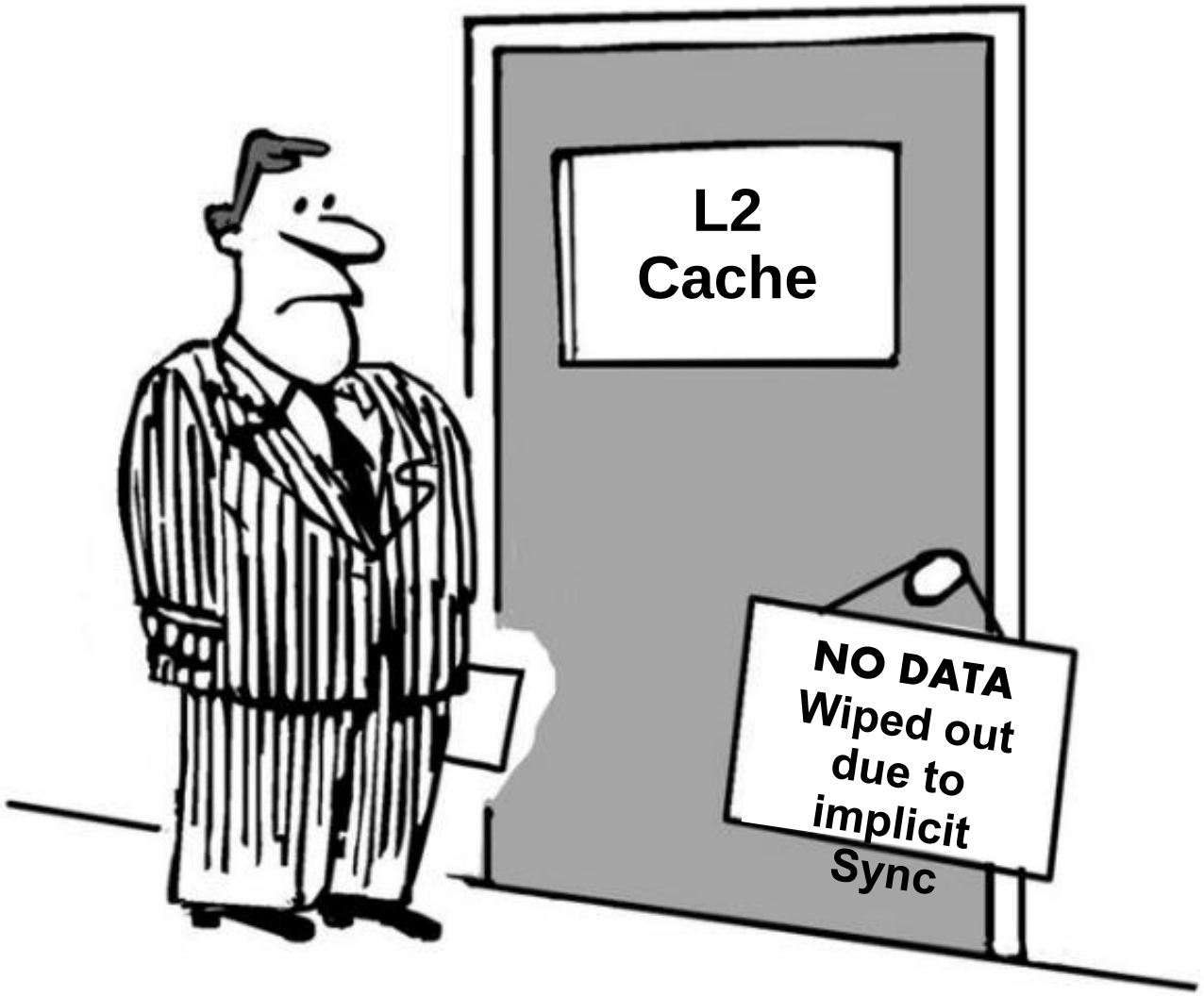
Modern GPU apps often use data sharing, reuse and fine-grained sync

Move to Multi-Chiplets



Key Takeaway: L2 caches have now become private -> Implicit kernel boundary sync impacts them

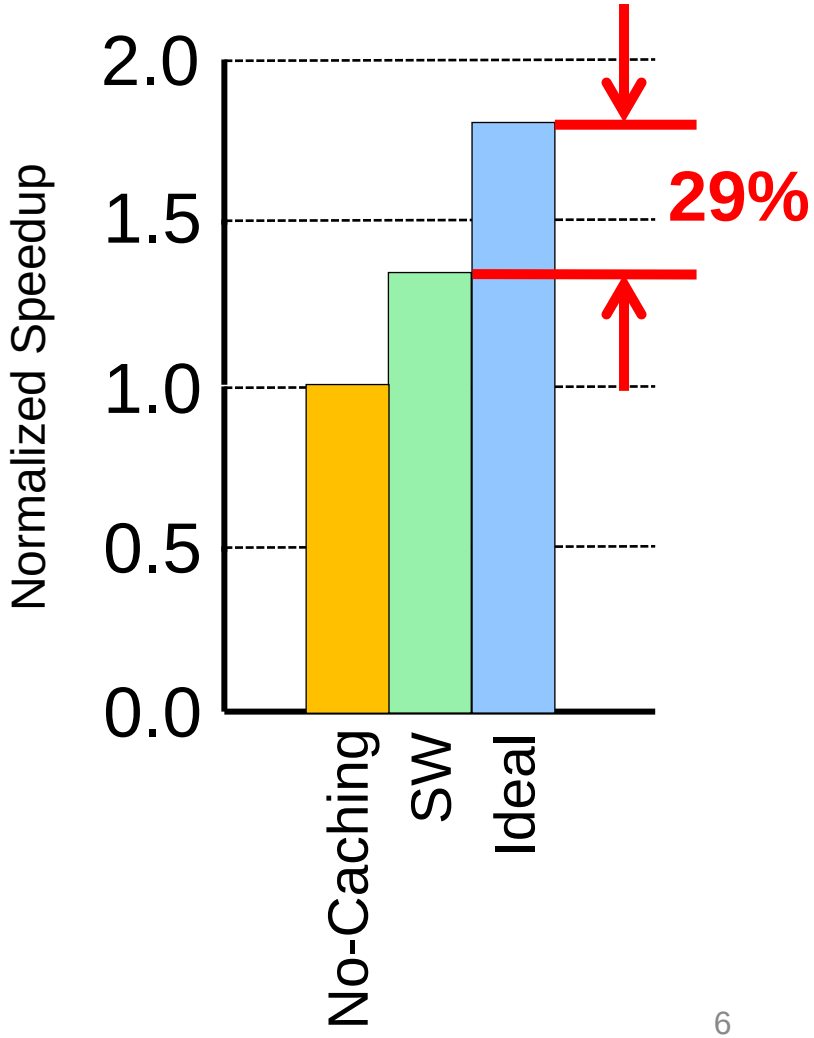
Why do we care about this?



Inter-kernel Reuse Matters

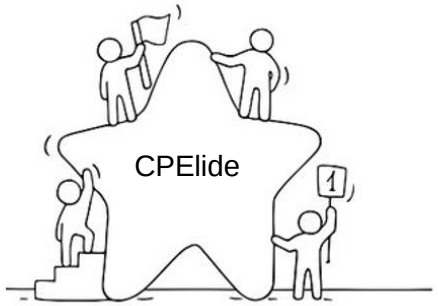
- [HMG] Ren, et al. [HPCA 2020]
  - 4-GPU system (4 chiplets each)
  - Study impact of existing coherence schemes
  - Entire L2 cache is invalidated at sync points
  - Perf 29% worse than idealized caching!
- [LADM] Khairy, et al. [Micro 2020]
  - Best thread-block management design had a 18% slowdown relative to a monolithic GPU of same size
  - Reason: Inter-kernel locality loss due to implicit global sync
- [CARVE] Jaleel, et al. [Micro 2018]
  - Study Impact of coherence on performance gains in NUMA-GPU
  - Findings: Important to retain data across kernel boundaries. Losing this inter-kernel shared L2 reuse hurts performance by 45% across HPC and ML applications in a NUMA-GPU system

Graph from Ren, et al. [HPCA 2020]

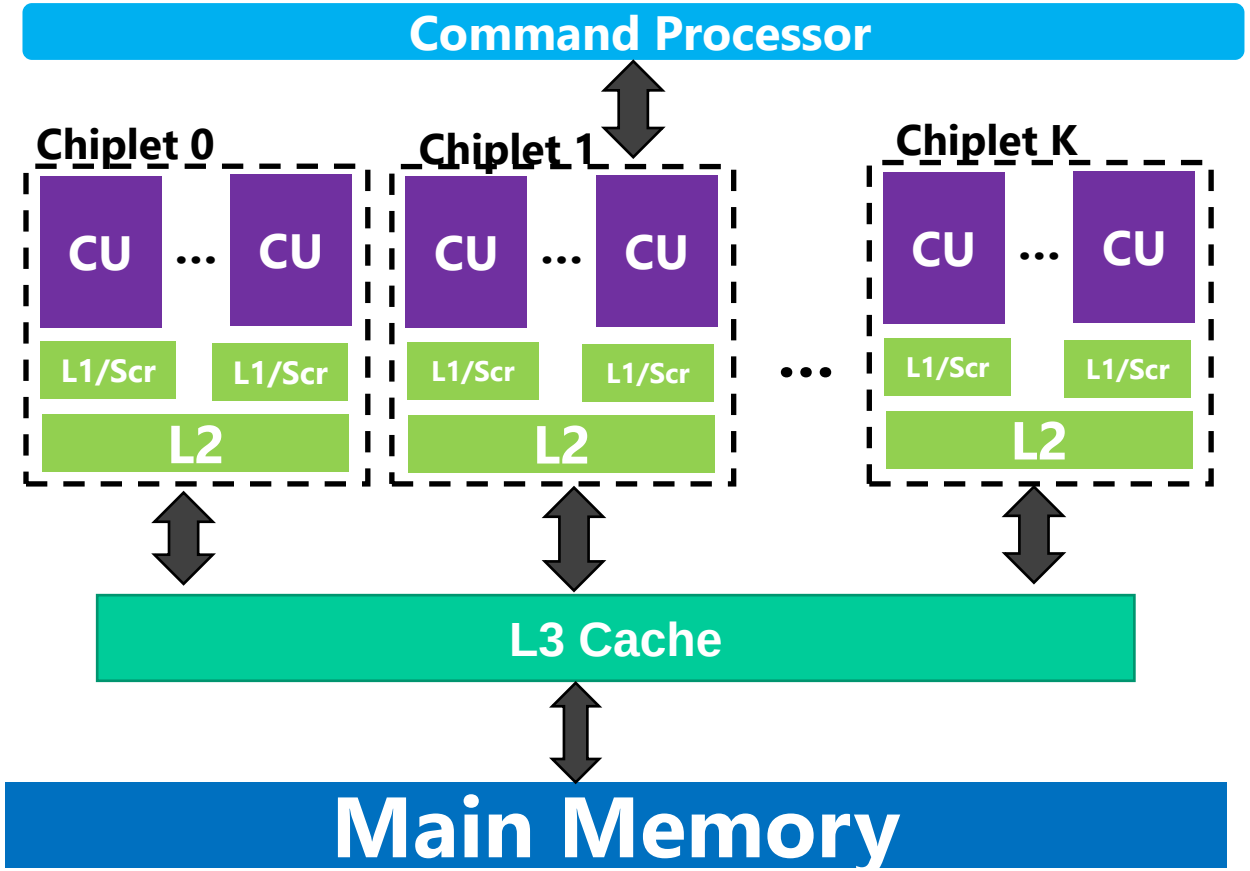


Contributions

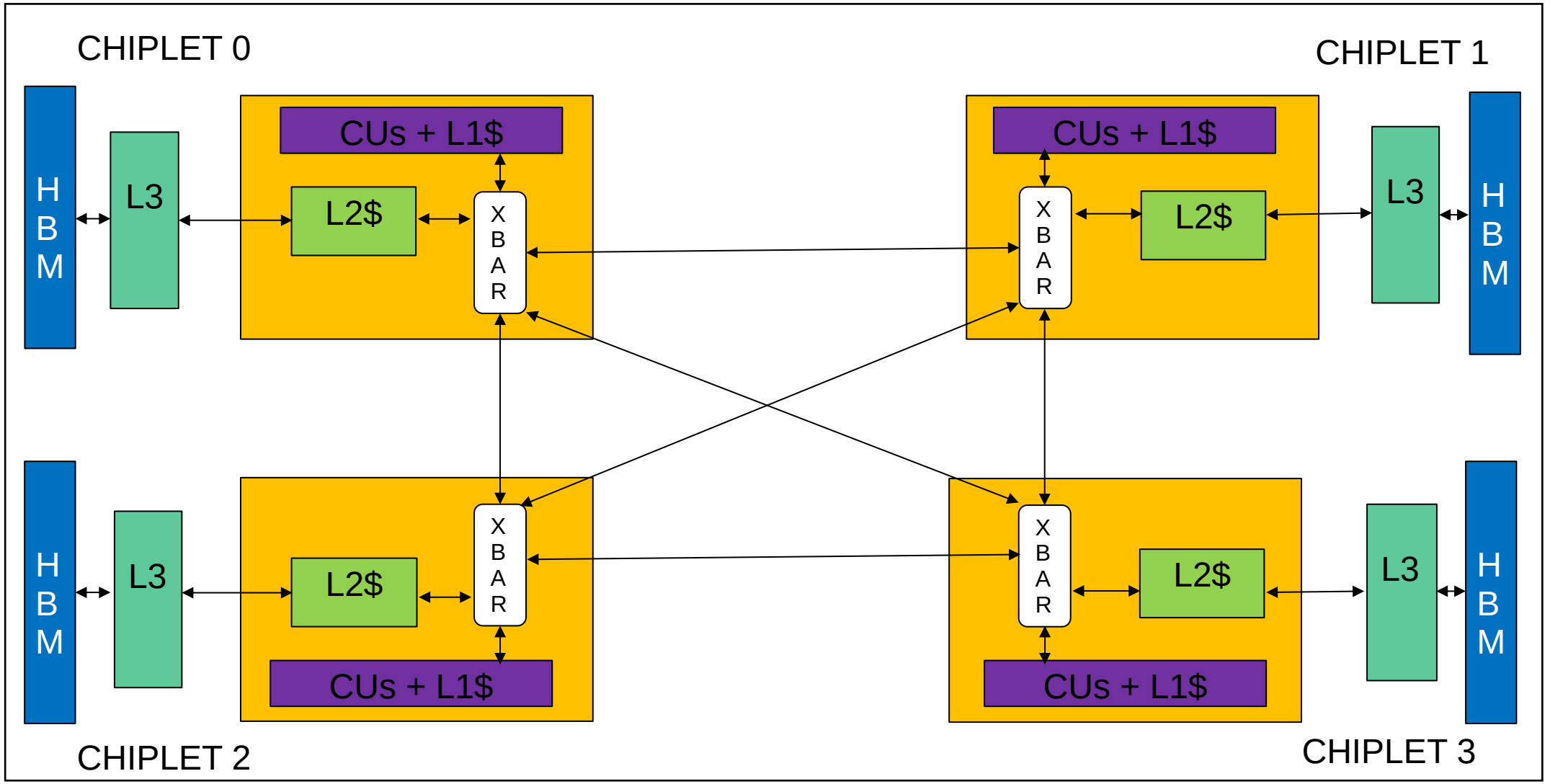
- Insight: Tracking inter-kernel dependencies inside the Command Processor (CP) can elide acquire/release at kernel boundaries
  - CP has dynamic scheduling information available
  - Programmers/compilers can identify mode of access/ranges for data structures
- CPElide adds dependency tracking table inside CP
  - Leverages information available in CP to conservatively store state of all data structures accesses for every 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
  - Reprogrammable to account for future trend changes



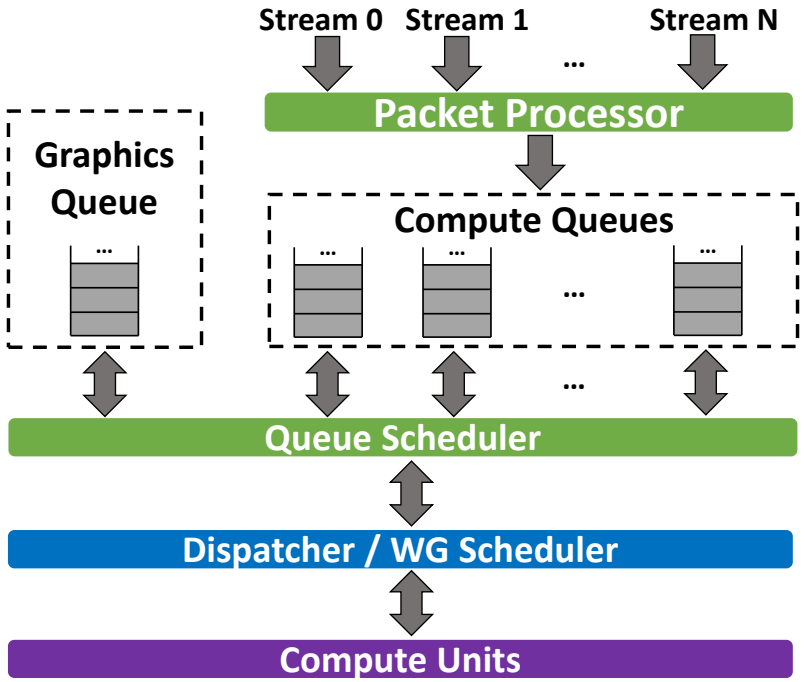
Background: Multi-Chip Architecture



- 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 memory (preceding WB req)


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

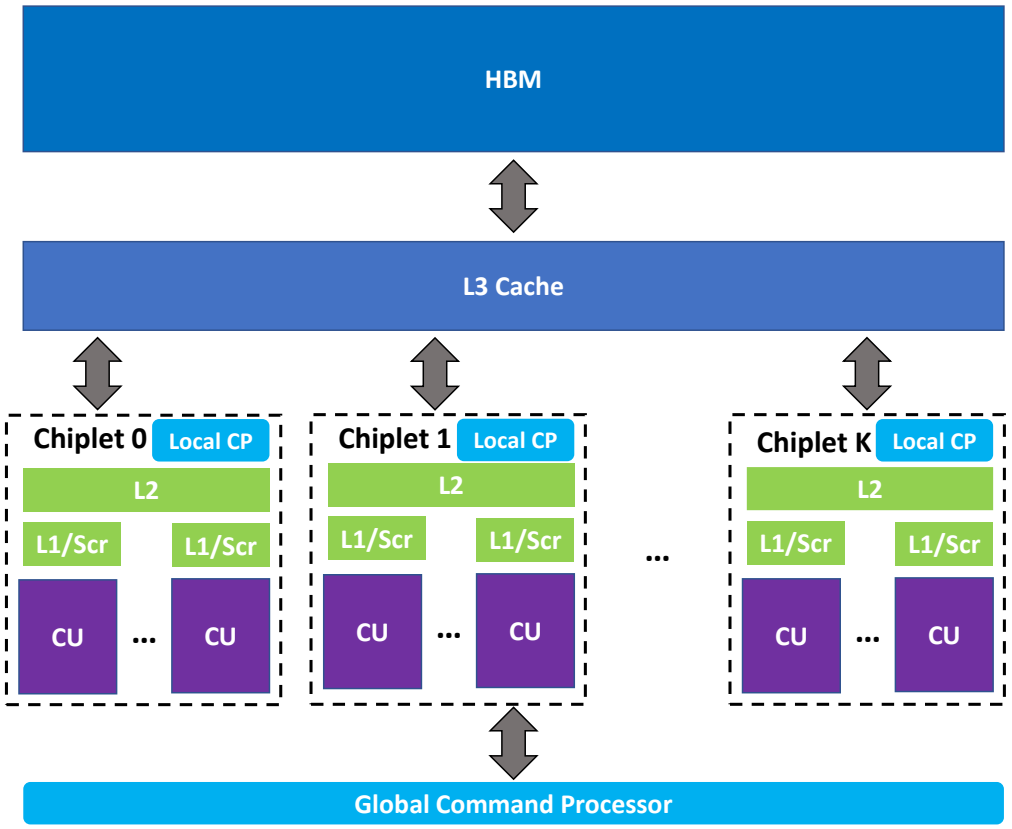
Background: CP



CP Memory

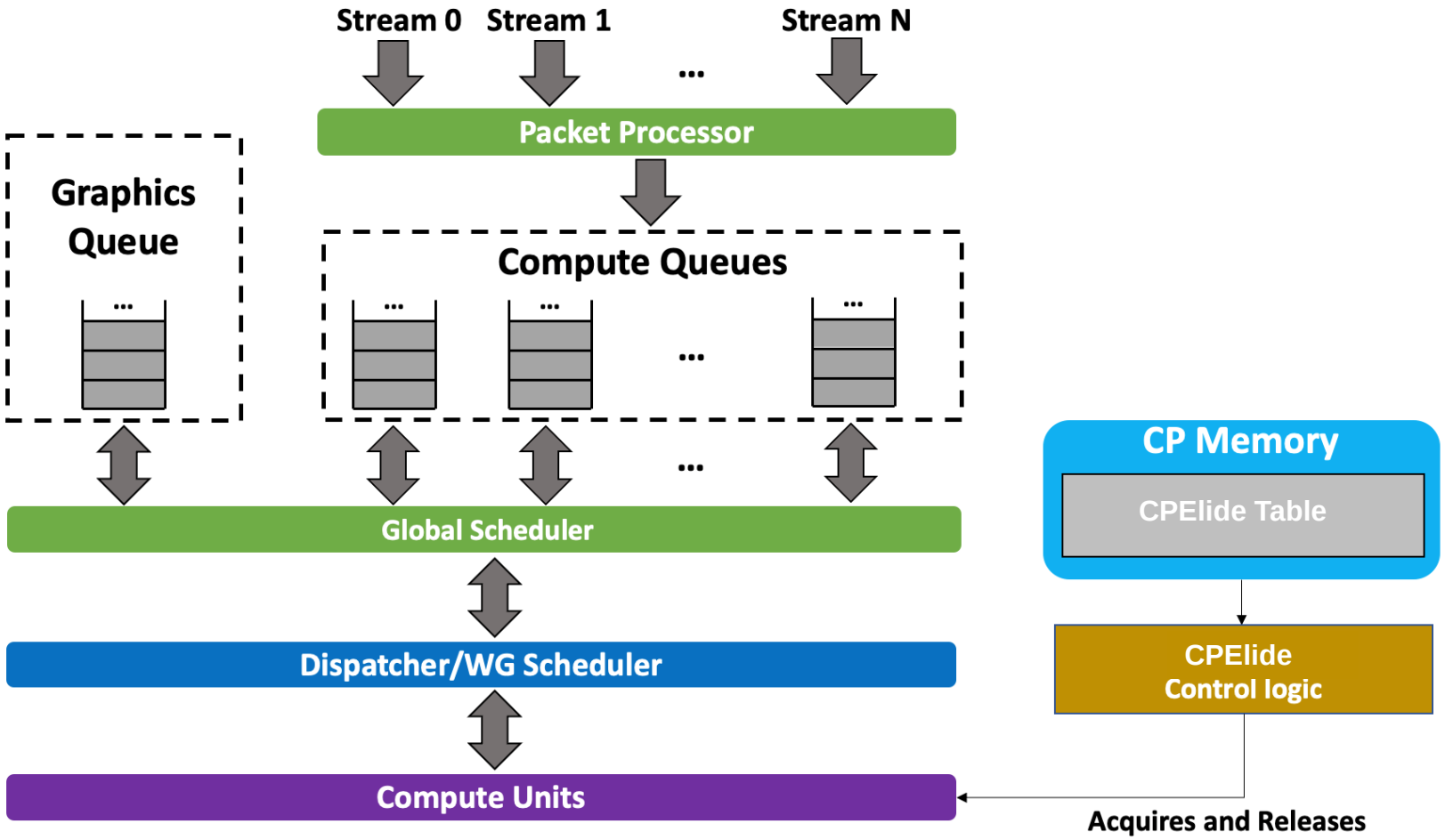
- ❑ CP has two primary components
  - Packet Processor
  - Workgroup (WG) Dispatcher
- ❑ CP responsible for dispatching WG's to CU's has dynamic scheduling info
- ❑ CP receives the kernel packet that has info the starting addr of DS accessed
  - It also initialize the RF and extracts meta data from binary

Design: Two Level CP

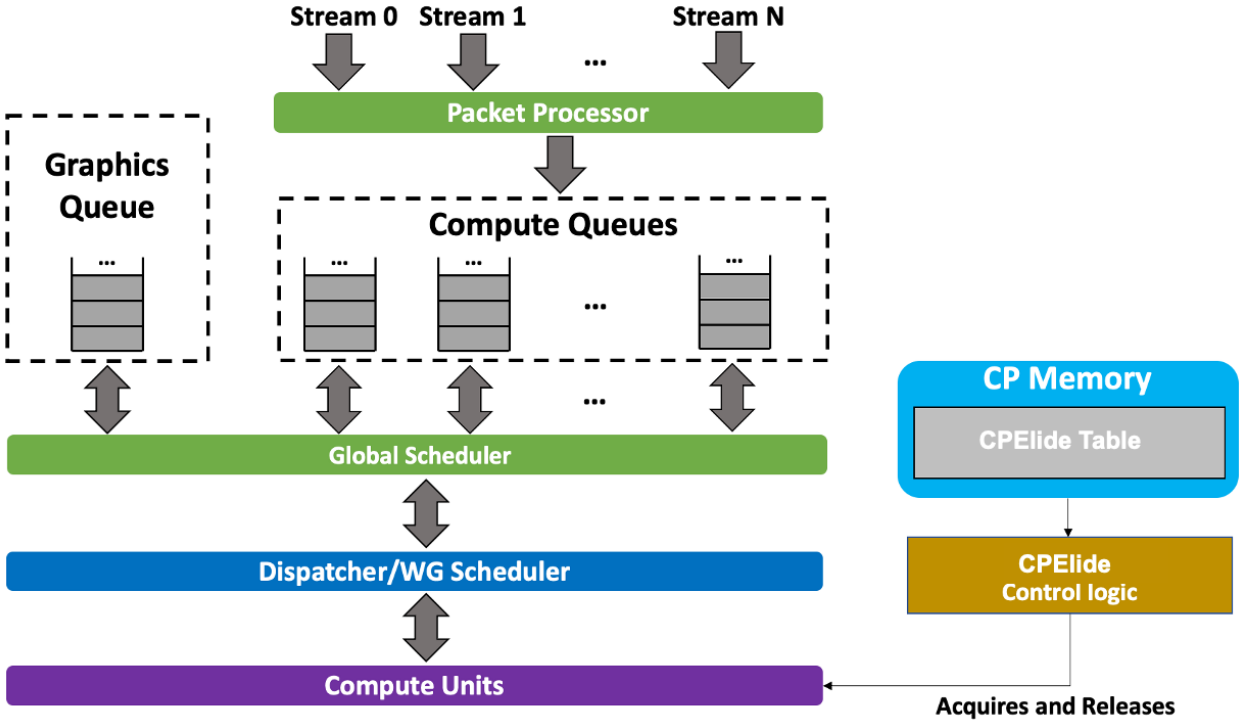


- Multi level Command Processor:
  - a global CP
  - a local CP per chiplet
- Local CP operation similar to monolithic GPU: controls local scheduling decisions
- Local CP passes runtime information back to global CP
- Global CP decides the distribution of work across chiplets and also houses CPElide

Design: Global CP

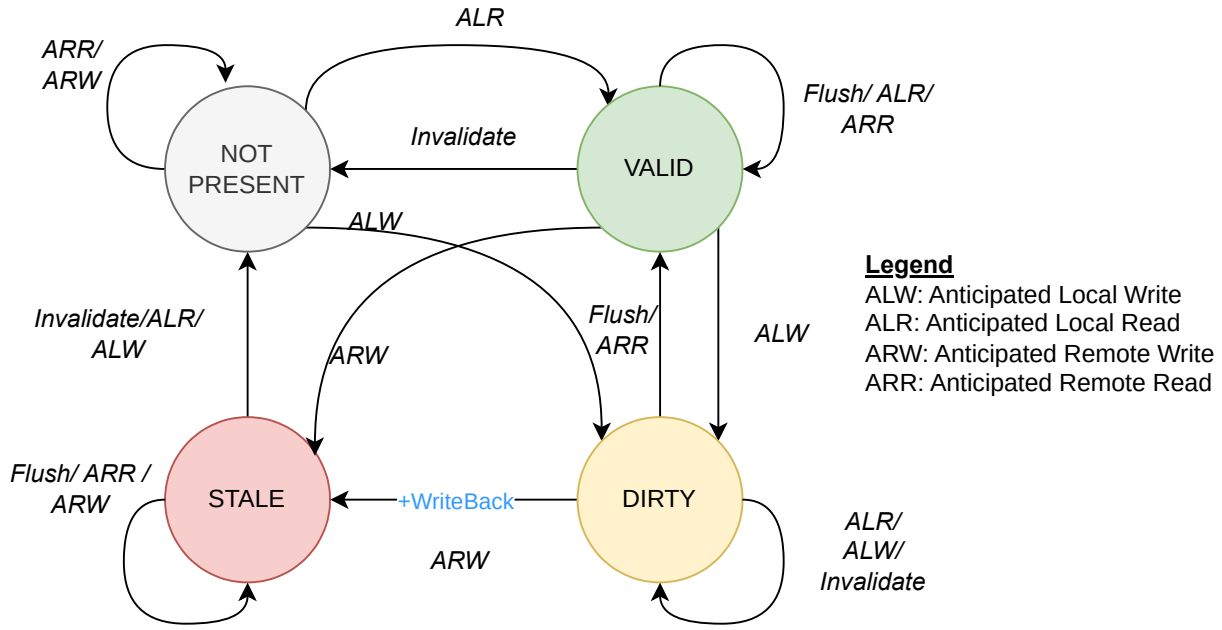


Design: CPElide



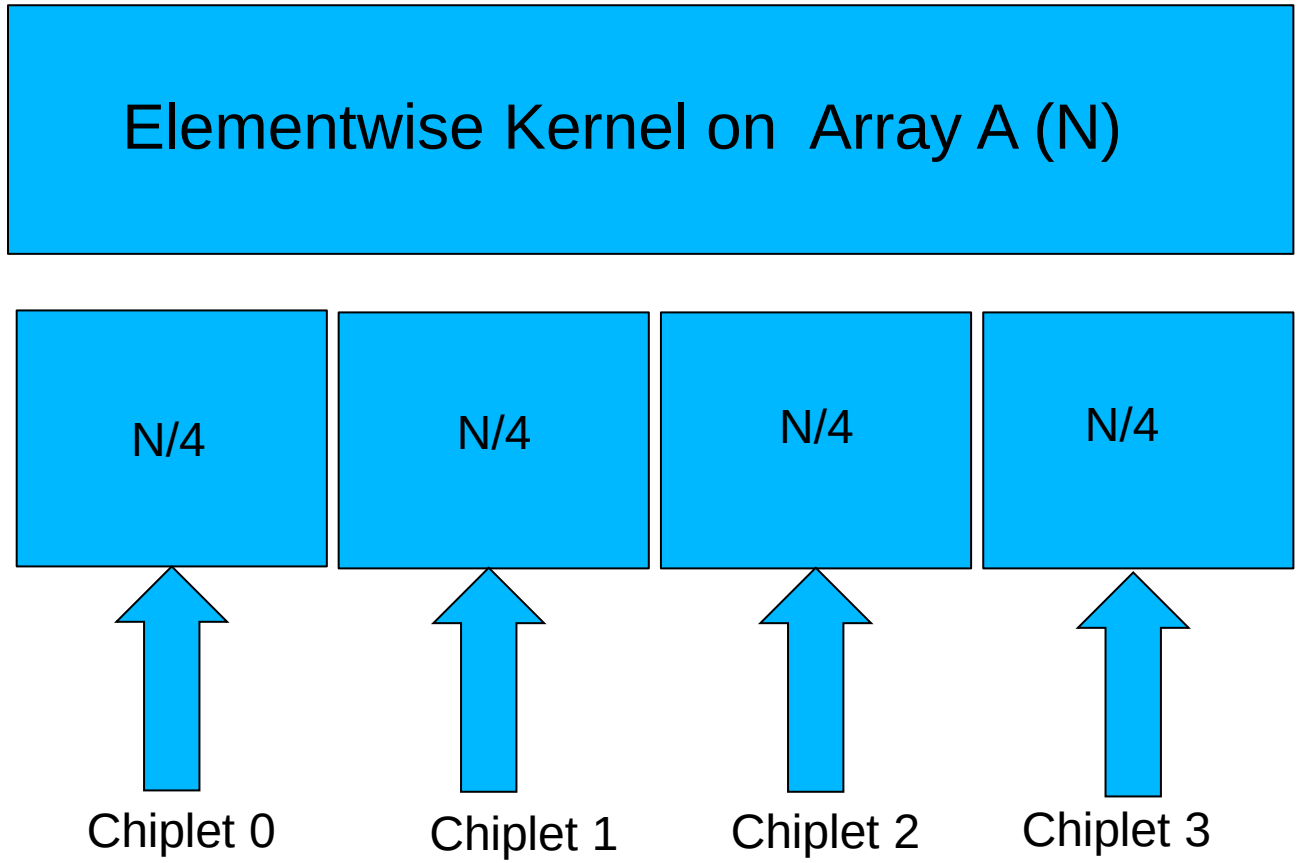
- CPElide updated after each kernel launch
  - Gets scheduling info from Global CP
  - Gets mode/range info from kernel packet
- This table has 4 fields per row:
  - DS identifier | Addr range(s) per chiplet | Access mode | bit vector

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
- State changes are triggered based on current state and access modes/ranges defined for the DS

Design: Range Tracking



- CPElide can track different address ranges for the same DS
- Chiplets commonly work on disjoint sets of data; range base tracking allows to elide rel/acq in these cases
- Range base tracking also helps in case of non disjoint sets e.g. later
- Range Based Tracking is software based still need to acq/rel complete cache

Design: Software Changes

Labeling Access Mode and Access Ranges

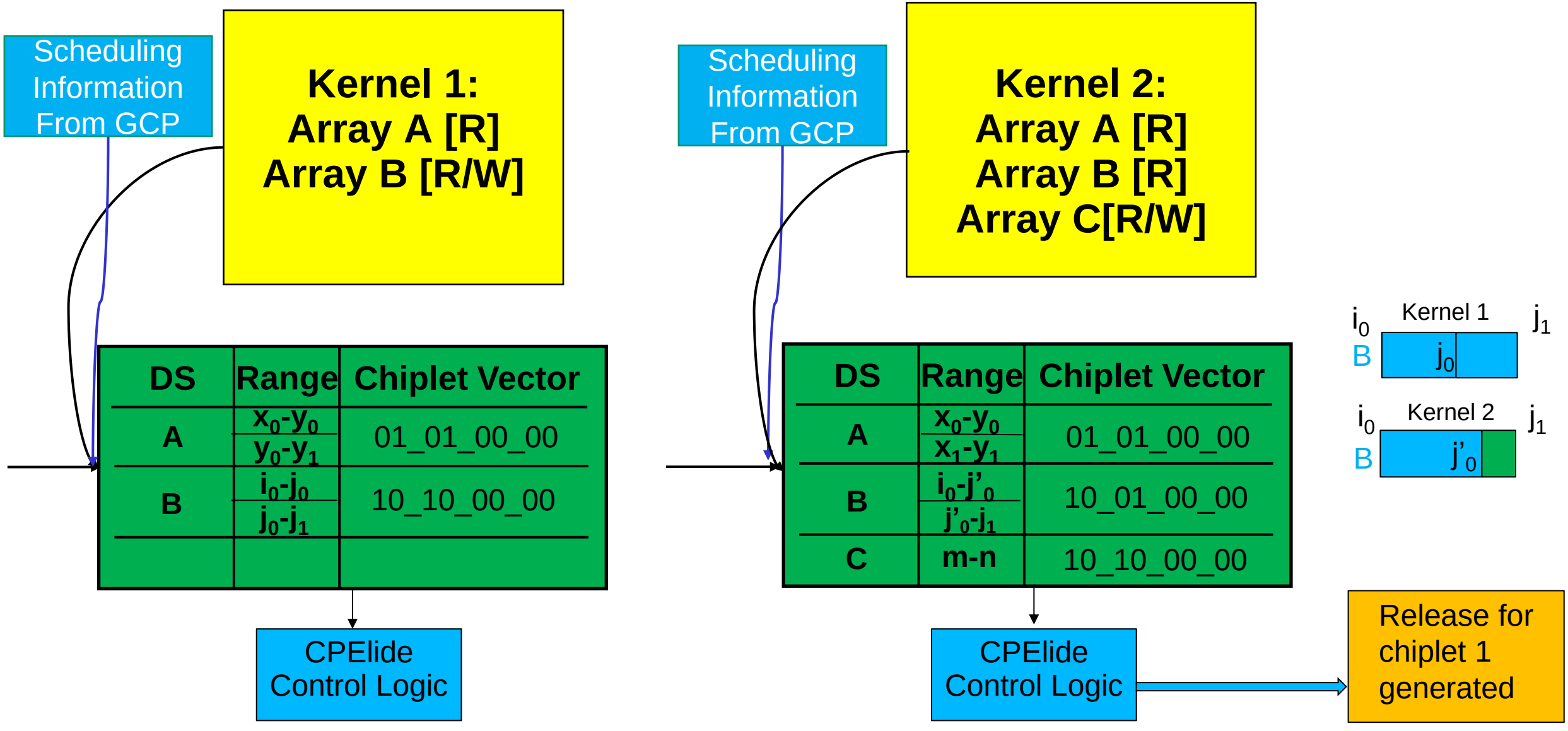
Labeling Access Mode

```
//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 can specify the access mode/range for a data structure before any kernel launch

CPElide Example



Evaluation Methodology

- System Simulated: AMD Radeon VII GPU

- Key Metrics: # chiplets: (2,4,6,7), 60 CUs per chiplet, 16 KB L1 Cache per CU, 8 MB L2 cache per chip, Inter-chiplet B/W 768 MB/S, L3 Size 16MB, 16 GB HBM2

- Simulation Environment: gem5 v21.1
  Evaluated Metrics: performance, network traffic, and energy consumption

- Configs: CPElide, Baseline system, HMG [Ren et al. HPCA 2022]
  - HMG Directory based cache coherence, keeps track of all sharers [Focus L2]

- Workloads
  - GPGPU benchmark Suite: Rodinia ; HPC: Lulesh, Pennant and HACC
  - Graph Analytics Benchmarks: Color , FW and SSSP
  - Machine Learning: 2 layer CNN, GRU, LSTM

CPElide VS HMG Design Choices

CPElide

01 Write Policy

- ❑ Write back cache with no sharer tracking

02 Allocation Policy

- ❑ Caches Local accesses but does not cache remote accesses

03 Coherence Protocol

- ❑ Uses GPU-VIPER coherence protocol
- ❑ Retains line in valid when a dirty line is written back to main memory

C H

HMG

01 Write Policy

- ❑ Write Through L2 with sharer tracking

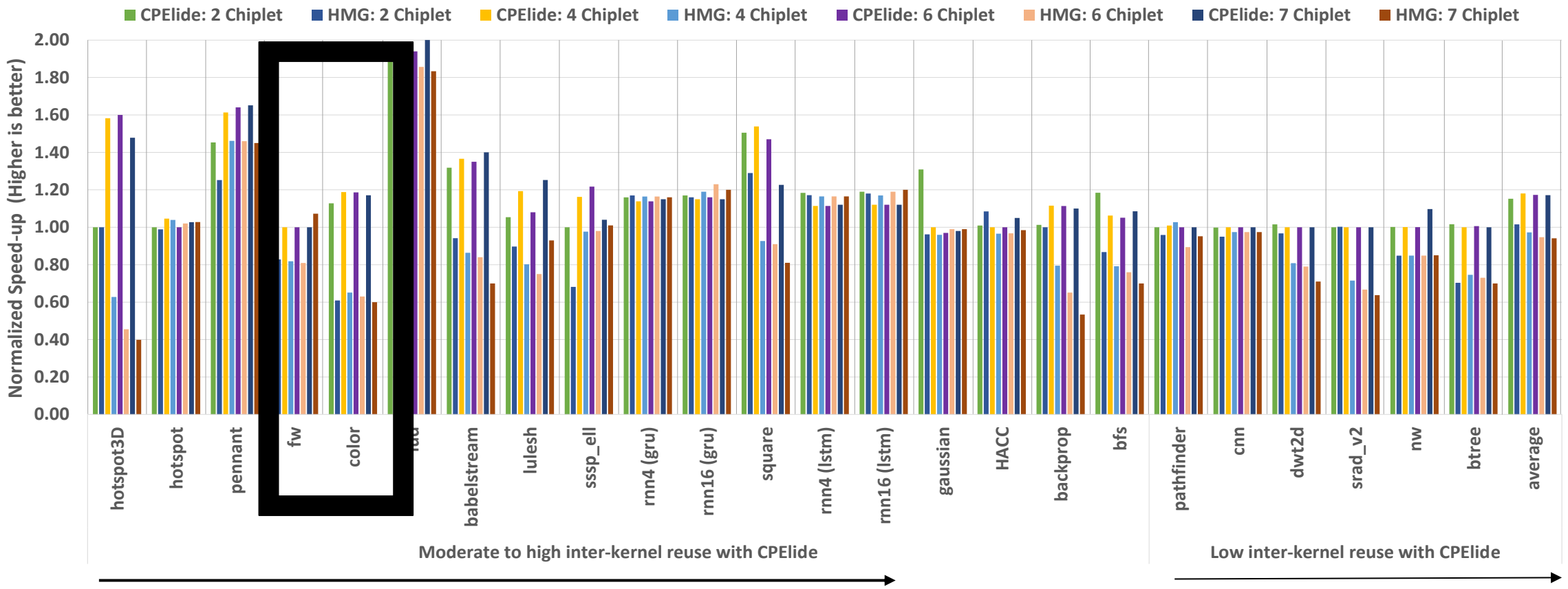
02 Allocation Policy

- ❑ Caches both local and remote accesses

03 Coherence Protocol

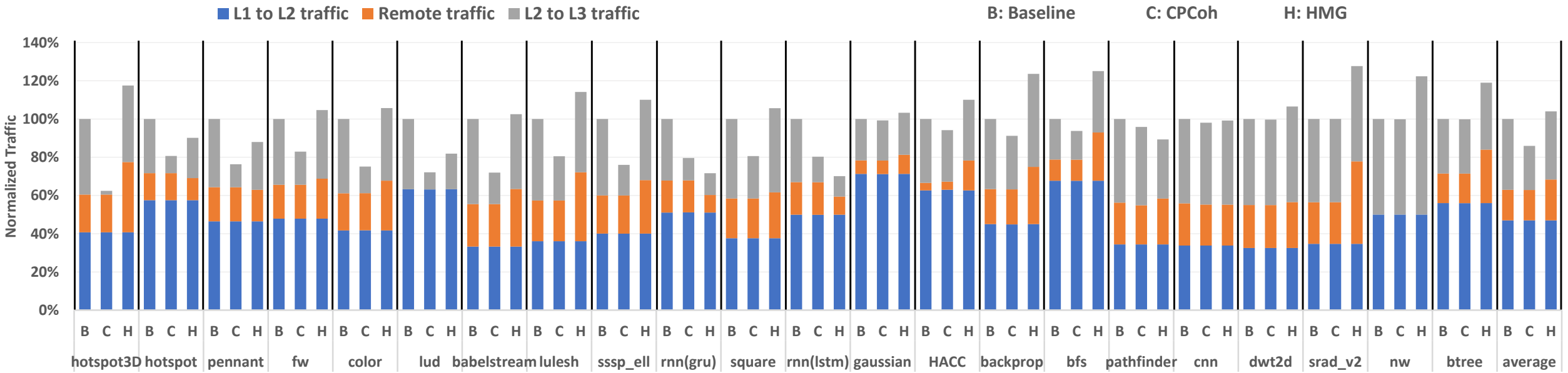
- ❑ A simple VI protocol, as updates always propagated to remote node
- ❑ Generates remote invalidation traffic in case of eviction or when some core writes to the block

Results: Normalized Performance



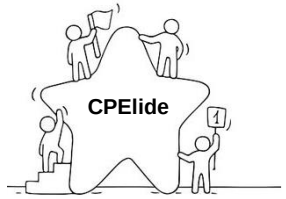
- CPElide outperforms both HMG (19%) and Baseline (13%)
  - Able to capture reuse from most apps with moderate-high inter-kernel reuse
  - Apps with little/no reuse suffer perf loss with HMG->additional remote traffic from inv
  - HMG also does badly for applications with little to no locality in remote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

Results: Network Traffic



- HMG higher remote traffic than CPElide/baseline by 23% because of block invalidations and less reuse in remote traffic
- Due to no data retention baseline has rel. higher L2->L3 traffic over HMG/CPElide
- CPElide > HMG in reducing L2 to L3 traffic, no WT traffic to maintain sharer list

Conclusions



- Multi-chiplets: Add additional level of mem hierarchy L3 cache, 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 information's and mode/range infor from s/w to keep track of 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

BACKUP

Related Work: HMG Overview

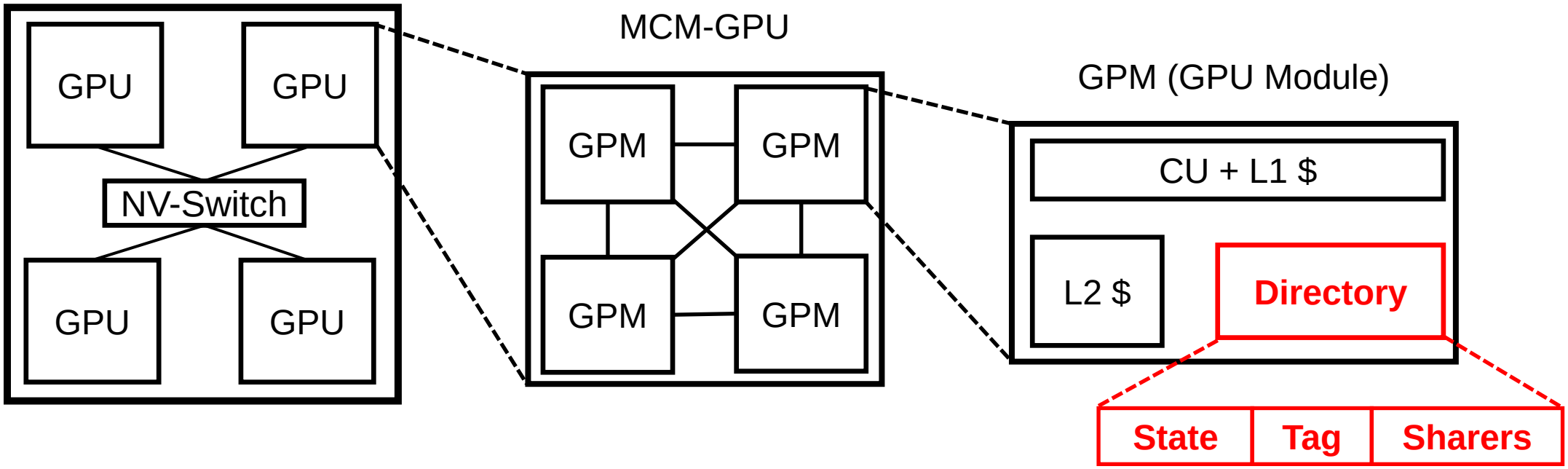
- Directory-based cache coherence, keep track of all sharers
- Map synchronization scopes to caches: .cta → L1 cache .gpu/.sys → L2 cache
- L1 cache coherence is software-maintained, they mainly focus on the L2 cache

Design choice Differences from CPElide

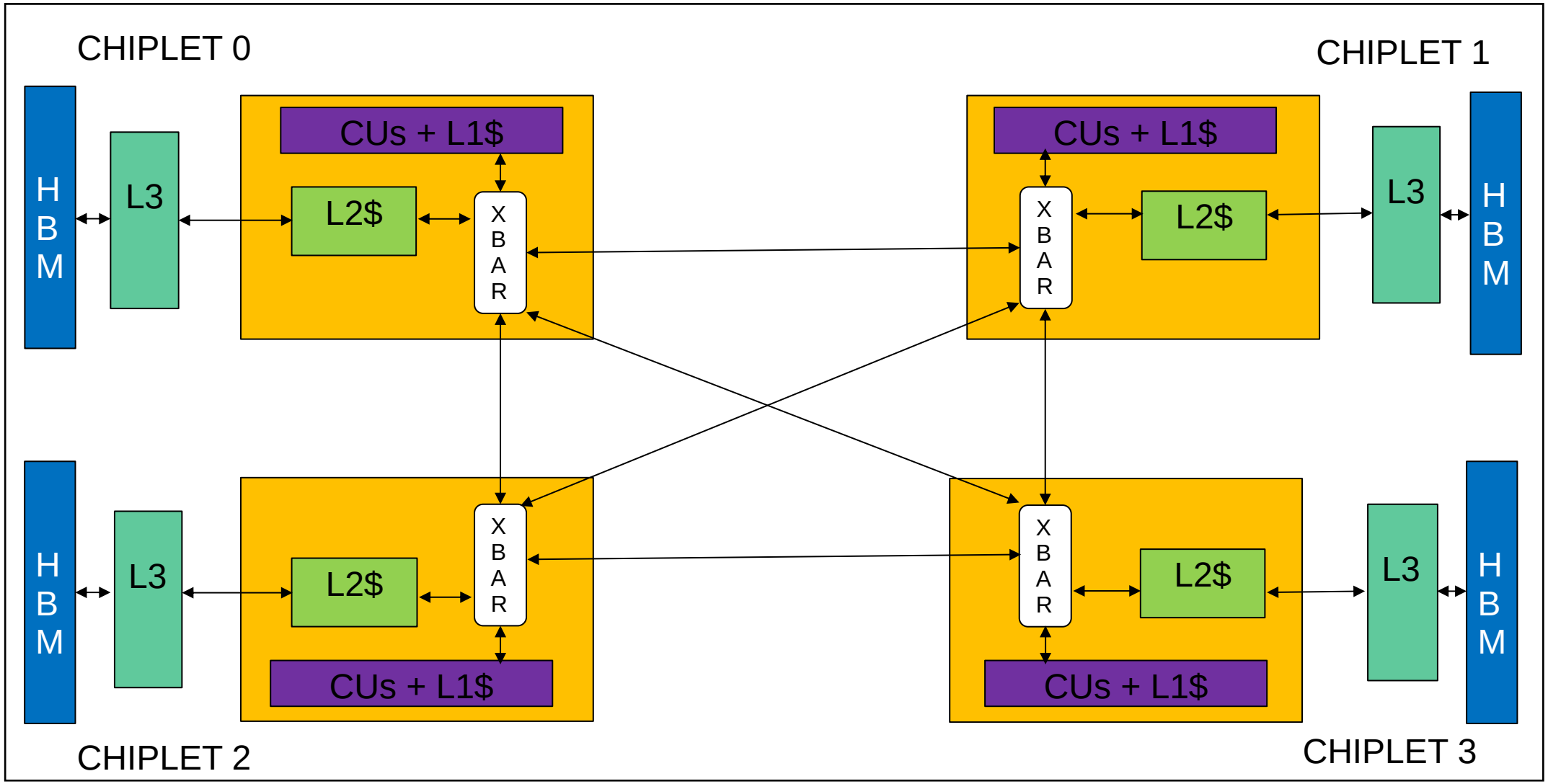
- Write Policy: HMG-> Writethrough, CPElide-> WriteBack
- Allocation Policy: HMG-> Cache local & remote , CPElide: Cache only local
- Coherence Protocol: HMG-> VI , CPElide: GPU-VIPER
- CPElide retains line in valid when a dirty line is written back to main mem
- HMG generates remote invalidations in case of eviction or when there is a write

BACKUP SLIDES

Related Work: HMG Overview



- Directory-based cache coherence, keep track of all sharers
- Map synchronization scopes to caches: .cta → L1 cache .gpu/.sys → L2 cache
- L1 cache coherence is software-maintained, they mainly focus on the L2 cache


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