MRPB: Memory Request Prioritization for Massively Parallel Processors

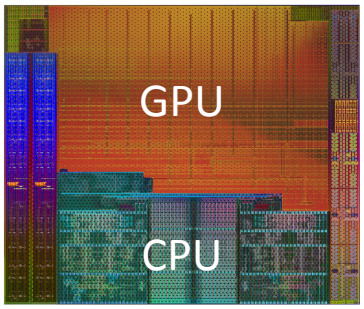
Wenhao Jia, Princeton University
Kelly A. Shaw, University of Richmond
Margaret Martonosi, Princeton University

PRINCETON UNIVERSITY

Benefits of GPU Caches

- Heterogeneous parallelism:
  - High performance per watt
  - Prominent example: CPU-GPU pairs

AMD Kaveri



- GPU memory hierarchy evolution:
  - SW-managed scratchpads → general-purpose caches
- Why GPU caches?
  - Reduce memory latency: esp. for irregular accesses
  - Memory bandwidth filter: reduce bandwidth demand
  - Programmability: easier to use than scratchpads

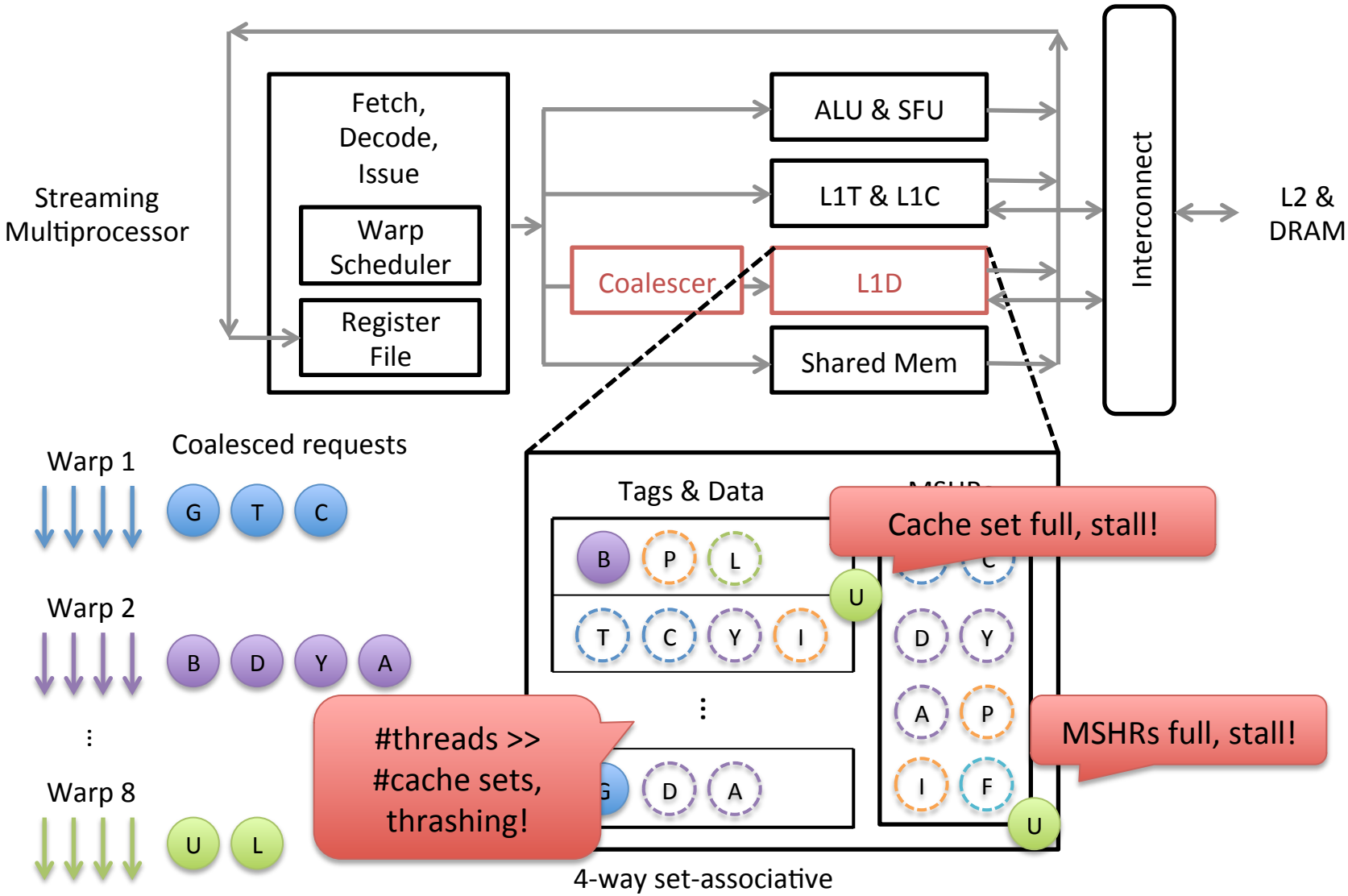
GPU Caches: Usage Issues

- Unpredictable performance impact
  - Real-system characterization [ICS'12 Jia]: caches helpful for some, ineffective for other kernels
  - Occasionally harmful: long cache line sizes causing excessive over-fetching
- Even some vendor uncertainty
  - NVIDIA Fermi L1: "users should experimentally determine on/off and size configurations"

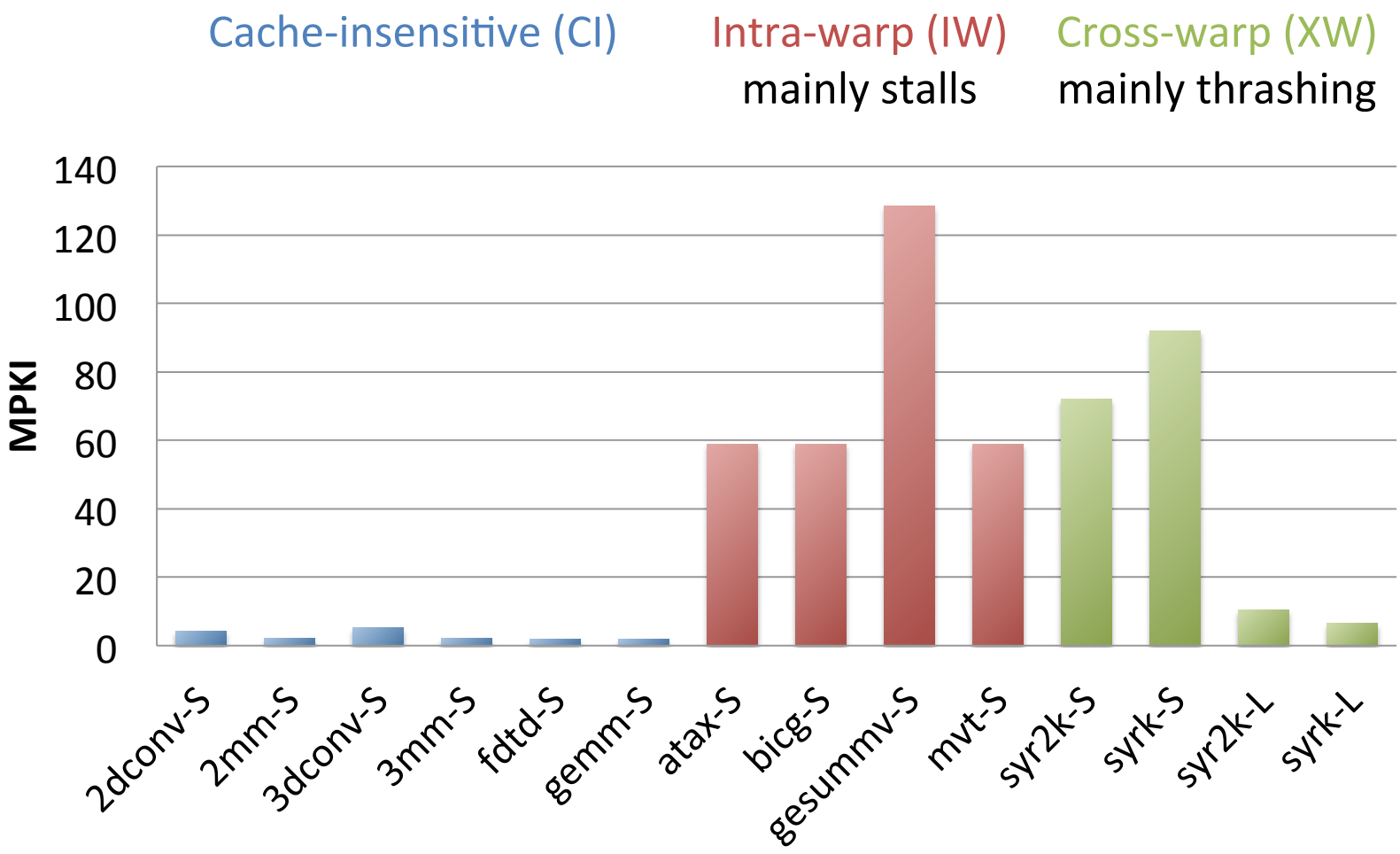
GPU Caches: Research Challenges

1. Thrashing due to low per-thread capacity
   - CPU L1: ~8–16 kilobytes per thread
   - GPU L1: ~8–16 bytes per thread
2. Resource conflict stalls due to bursts of concurrent requests from active threads
   - 100s of memory requests from 1000s of threads
   - GPU L1: ≤ 64-way & dozens of MSHRs

Too Many Threads, Too Few Resources



Cache Miss Categorization



Based on relationships between evicting & evicted threads

Related Work

- Warp schedulers [Two-Level, CCWS]
  - Indirectly manage caches, can't target IW contention
  - MRPB outperforms a state-of-the-art scheduler, CCWS
- Software optimization [Throttling, Dymaxion, DL]
  - Significant user effort, platform-dependent
- Hardware techniques [RIPP, Tap, DRAM-Sched]
  - Not directly applicable to GPU characteristics

Research Challenges → Solutions

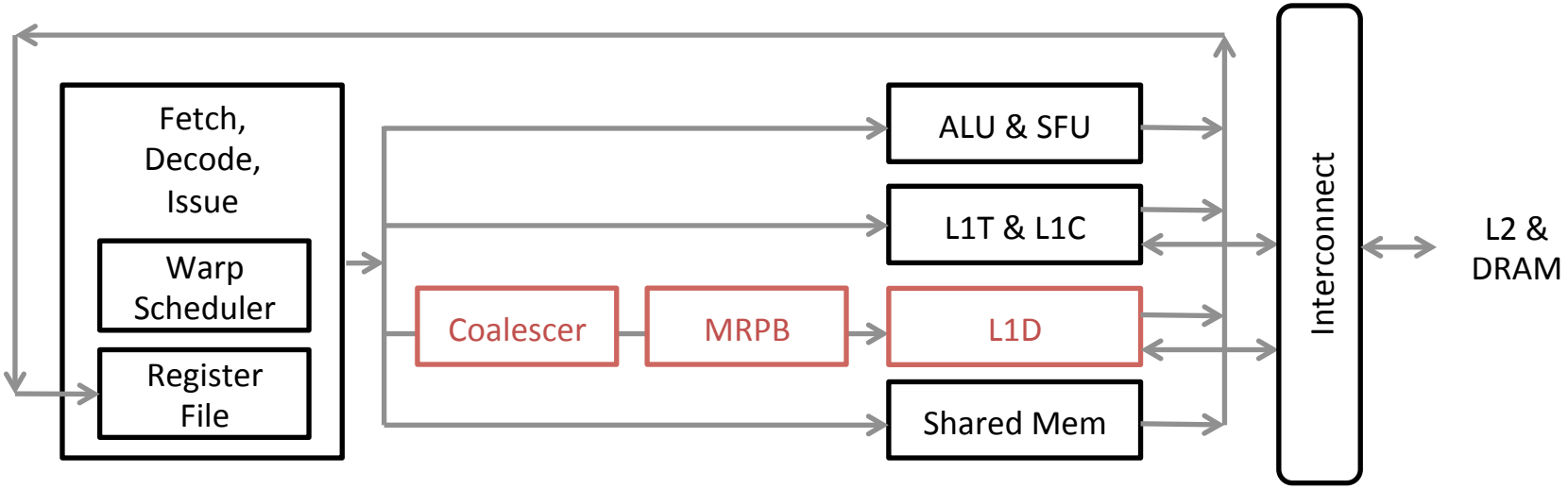
1. Thrashing due to low per-thread capacity
   - CPU L1: ~8–16 kilobytes per thread
   - GPU L1: ~8–16 bytes per thread

   Solution: reorder reference streams to group related requests
2. Resource conflict stalls due to bursts of requests
   - 100s of requests from 1000s of threads
   - GPU L1: ≤ 64-way & dozens of MSHRs

   Solution: let requests bypass caches to avoid resource stalls

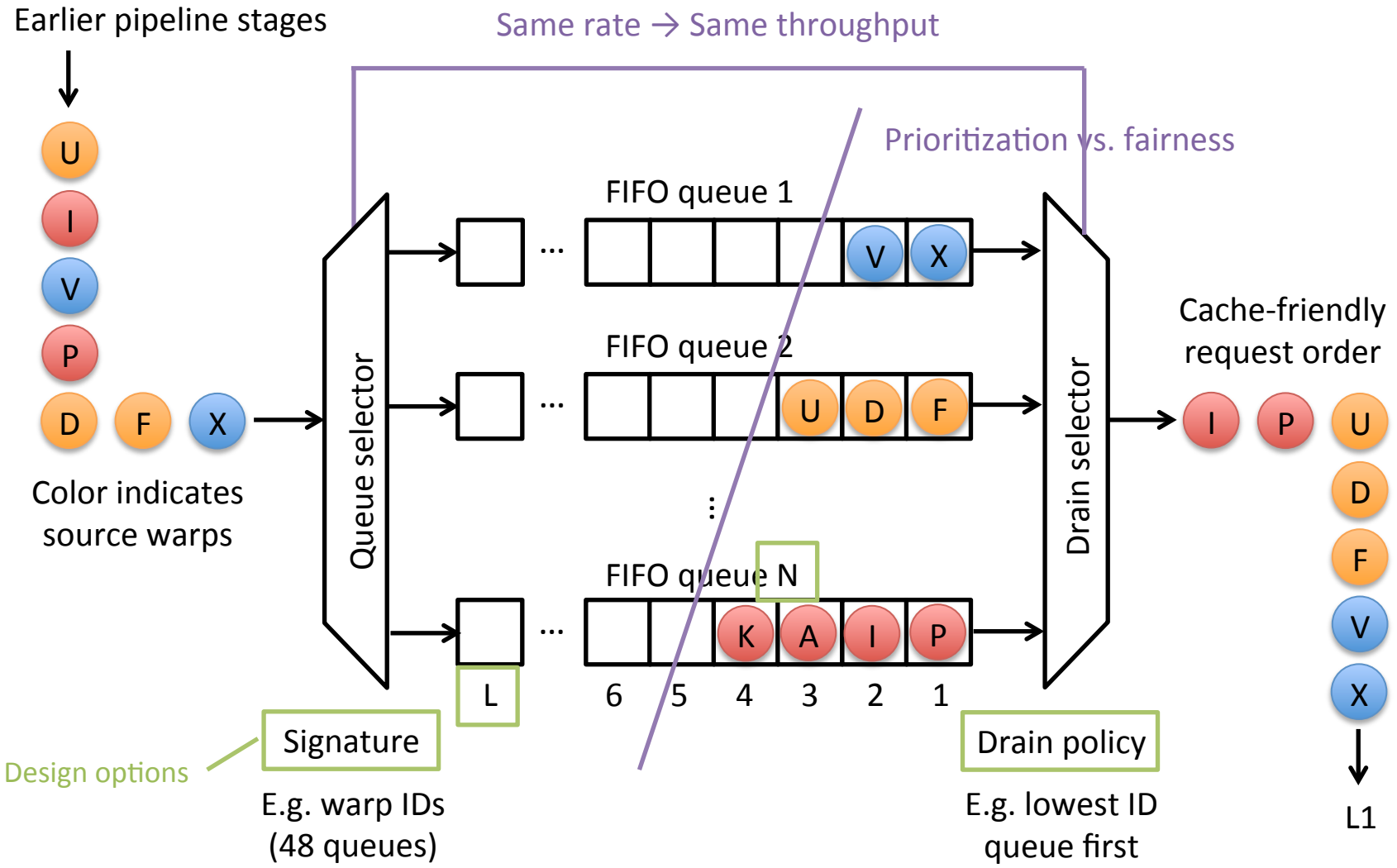
- Our approach: request prioritization = reorder + bypass
  - Prioritized cache use → effective per-thread cache resources ↑
  - 2.65× and 1.27× IPC improvements for PolyBench and Rodinia

Memory Request Prioritization Buffer

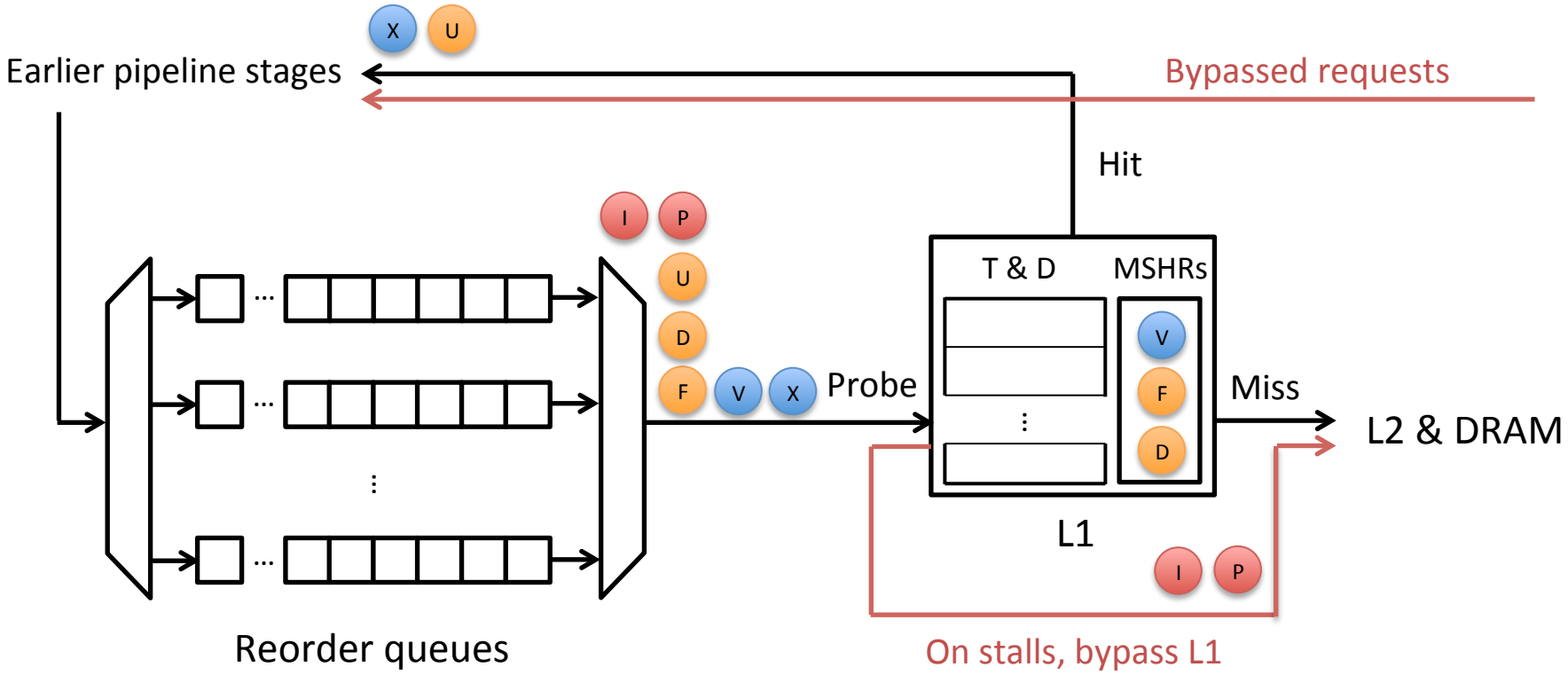


- Goal: make reference streams cache-friendly
- Key insight: delay some requests/warps to prioritize others → higher overall throughput

Request Reordering: Reduce Thrashing



Cache Bypassing: Reduce Stalls



- Bypassing condition modulates aggressiveness: bypass-on-all-stalls vs. bypass-on-conflict-stalls-only
- Exploit GPUs' weak consistency model for correctness

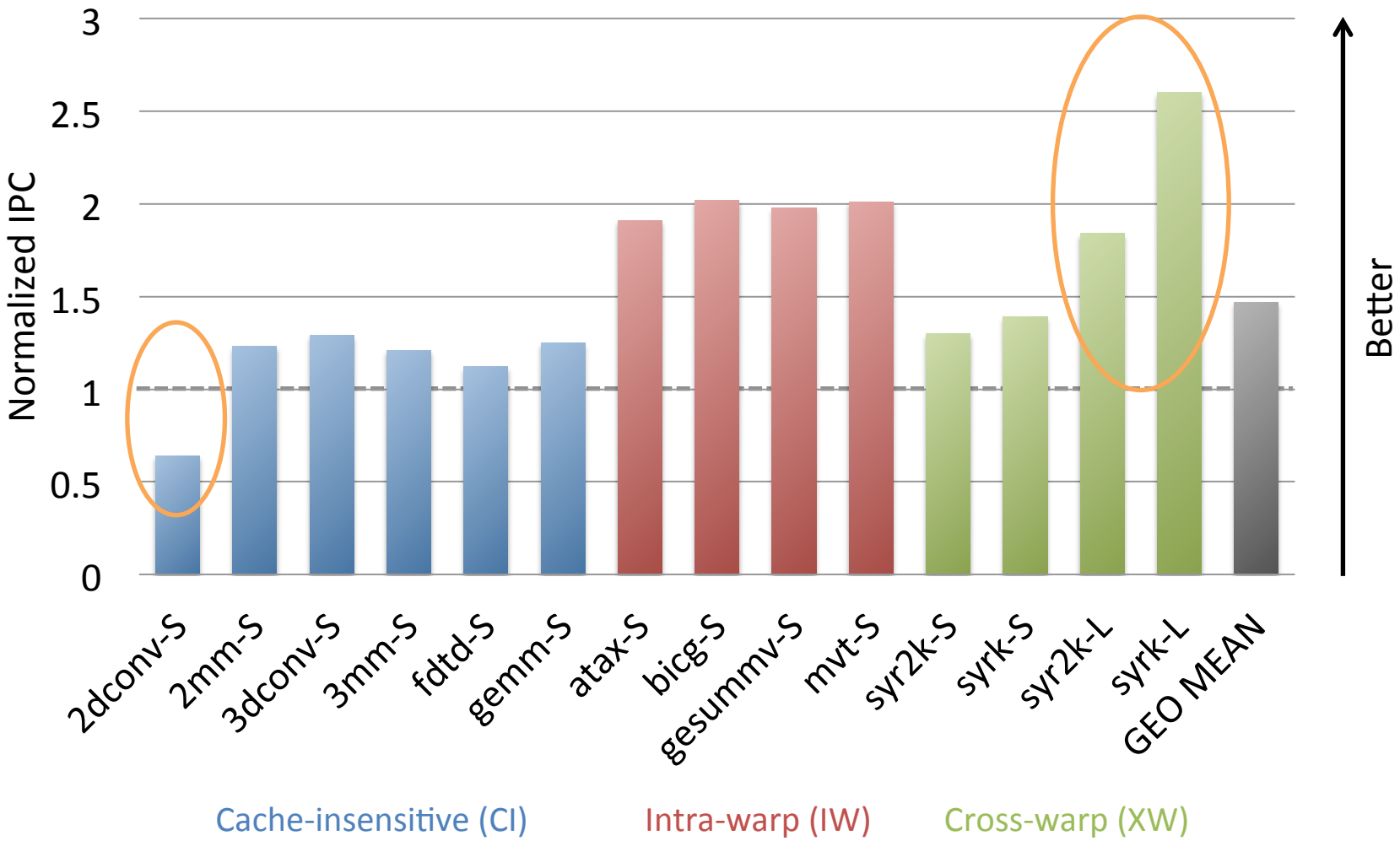
MRPB Design Summary

- Key observations:
  - Request reordering: longer timescale, reduces thrashing from cross-warp (XW) contention
  - Cache bypassing: more bursty, reduces resource conflict stalls from intra-warp (IW) contention
- Full design space exploration in the paper
  - Signature, drain policy, congestion/write flush, queue size and latency, bypassing policy

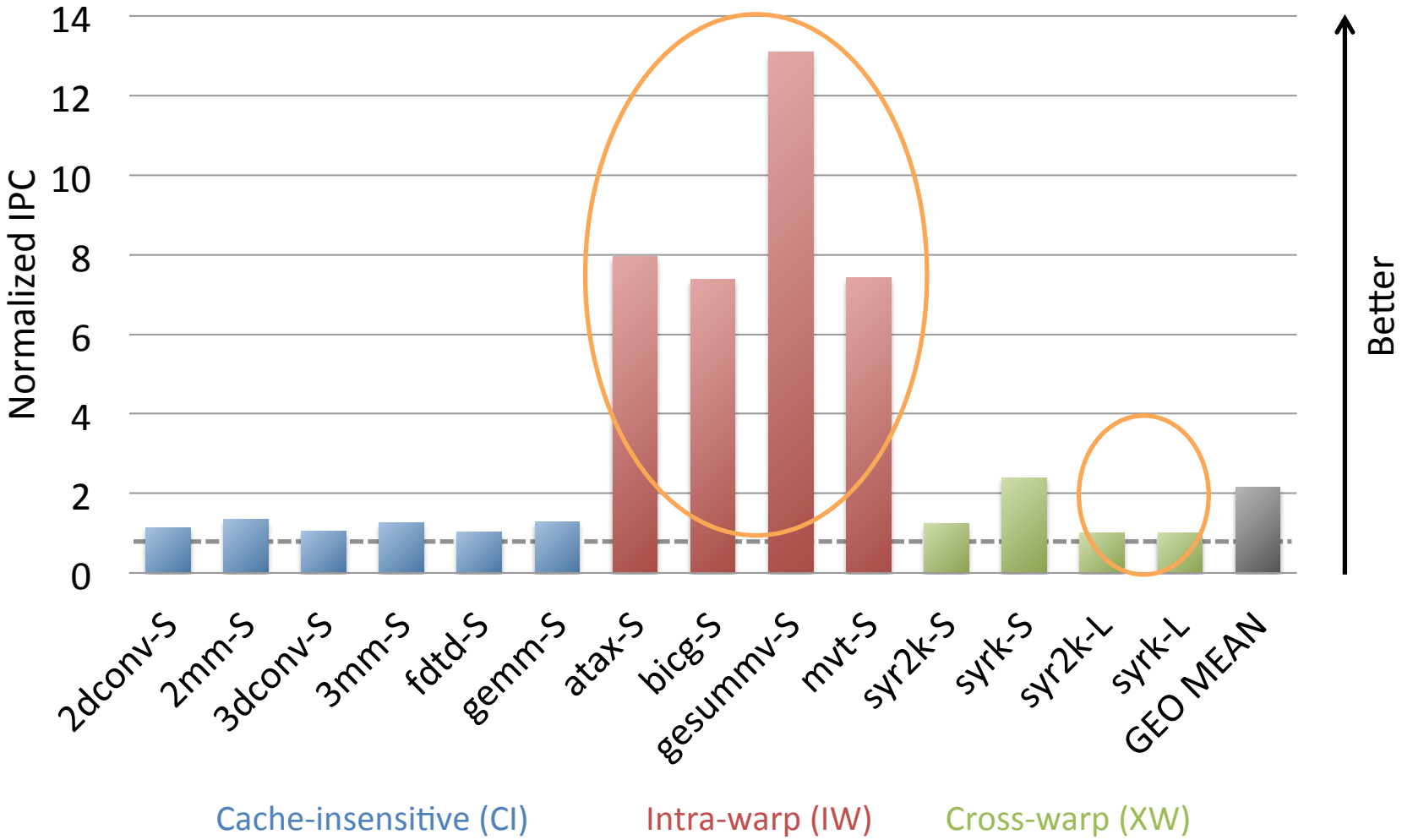
Experimental Methodology

- Simulated GPU: NVIDIA Tesla C2050
  - How does MRPB handle different cache sizes?
    - Baseline-S: 4-way 16KB L1 vs. Baseline-L: 6-way 48KB L1
- Benchmark suites: PolyBench & Rodinia
  - Different usage scenarios:
    - PolyBench: cross-platform vs. Rodinia: GPU-centric
  - Different optimization levels:
    - PolyBench: rely on caches vs. Rodinia: rely on scratchpads
  - Different purposes in our study:
    - PolyBench: exploration vs. Rodinia: evaluation

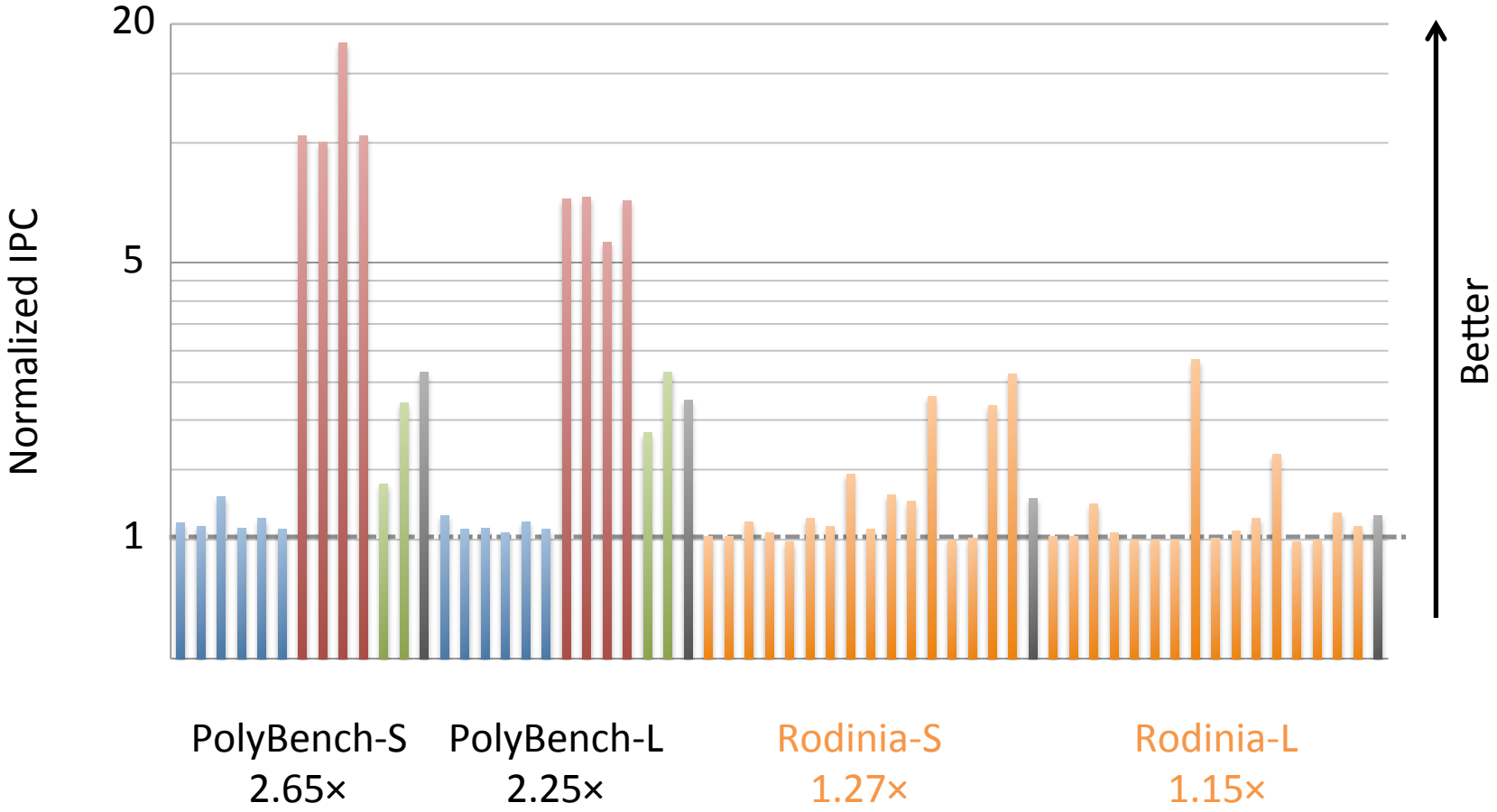
Reordering Benefits XW Applications



Bypassing Benefits IW Applications



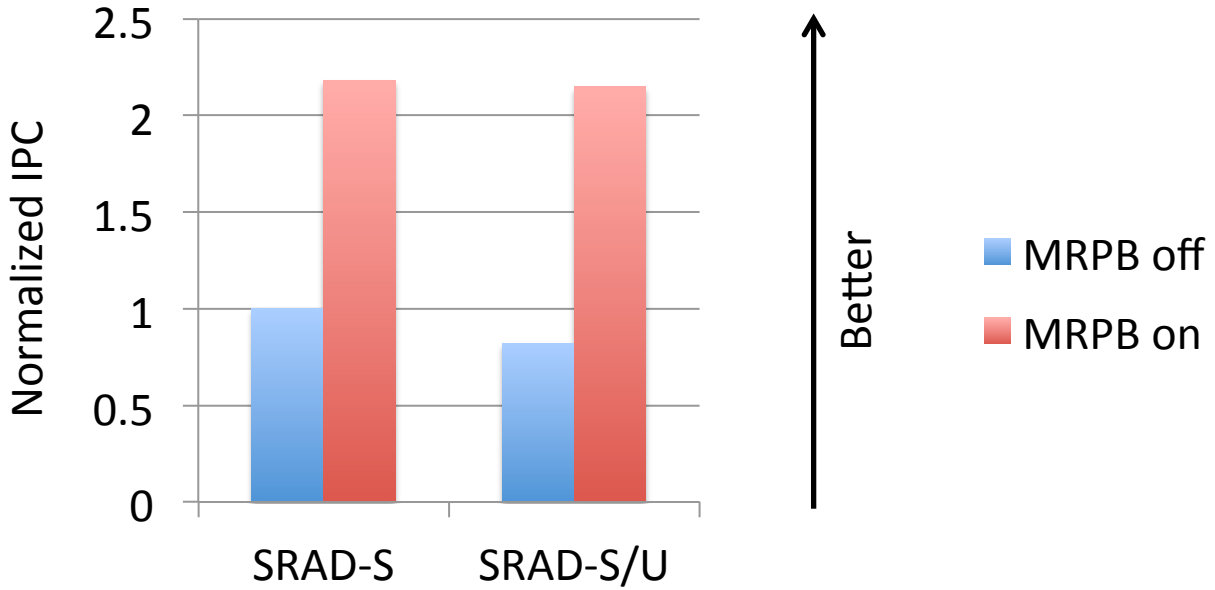
Final Design: Reordering + Bypassing



MRPB doesn't harm any app's performance!

Improving Programmability with MRPB

- Use caching + MRPB instead of shared memory
  - 6 “unshared” (/U) Rodinia apps: 37% slower → 9% with MRPB
- For SRAD-S, caching + MRPB outperforms shared version
  - Best of both worlds: better programmability & performance



Conclusion

- Highlighted and characterized how high thread counts often lead to thrashing- and stall-prone GPU caches
- MRPB: a simple HW unit for improving GPU caching
  - 2.65×/1.27× for PolyBench/Rodinia for 16KB L1
  - L1-to-L2 traffic reduced by 15.4–26.7%
  - Low hardware cost: 0.04% of chip area
- Future work and broader implications
  - Rethink GPU caches' primary role: latency → throughput
  - (Re-)design GPU components with throughput as a goal