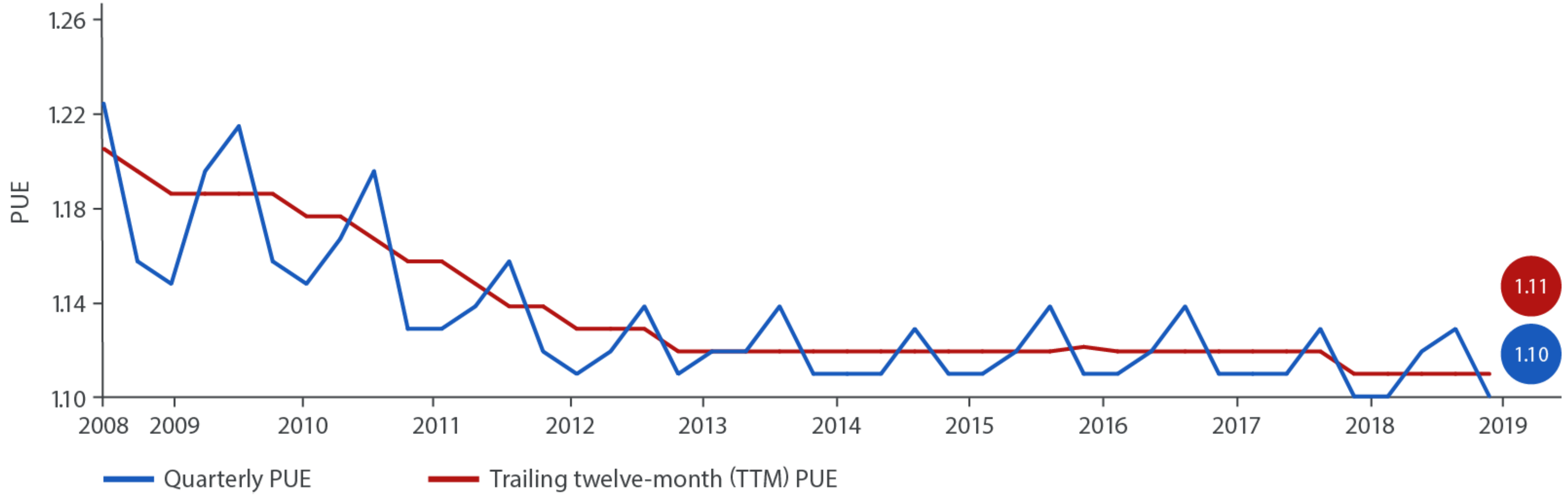
Kelp: QoS for Accelerated Machine Learning Systems

Haishan Zhu[1,3], David Lo[2], Liqun Cheng[2], Rama Govindaraju[2], Parthasarathy Ranganathan[2], and Mattan Erez[1]

[1] The University of Texas at Austin [2]Google [3]Microsoft

Cost-Efficiency Drives WSCs

- Cost Amortization
  - Power, cooling, resource management, etc.

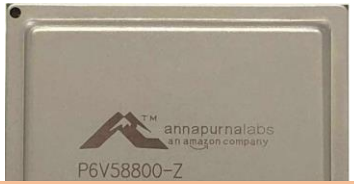


https://www.google.com/about/datacenters/efficiency/internal/

- Resource Utilization
  - "Backfill" hardware resources causes interference
  - Literatures report average between 10% to 50%

Accelerated ML Systems in WSCs

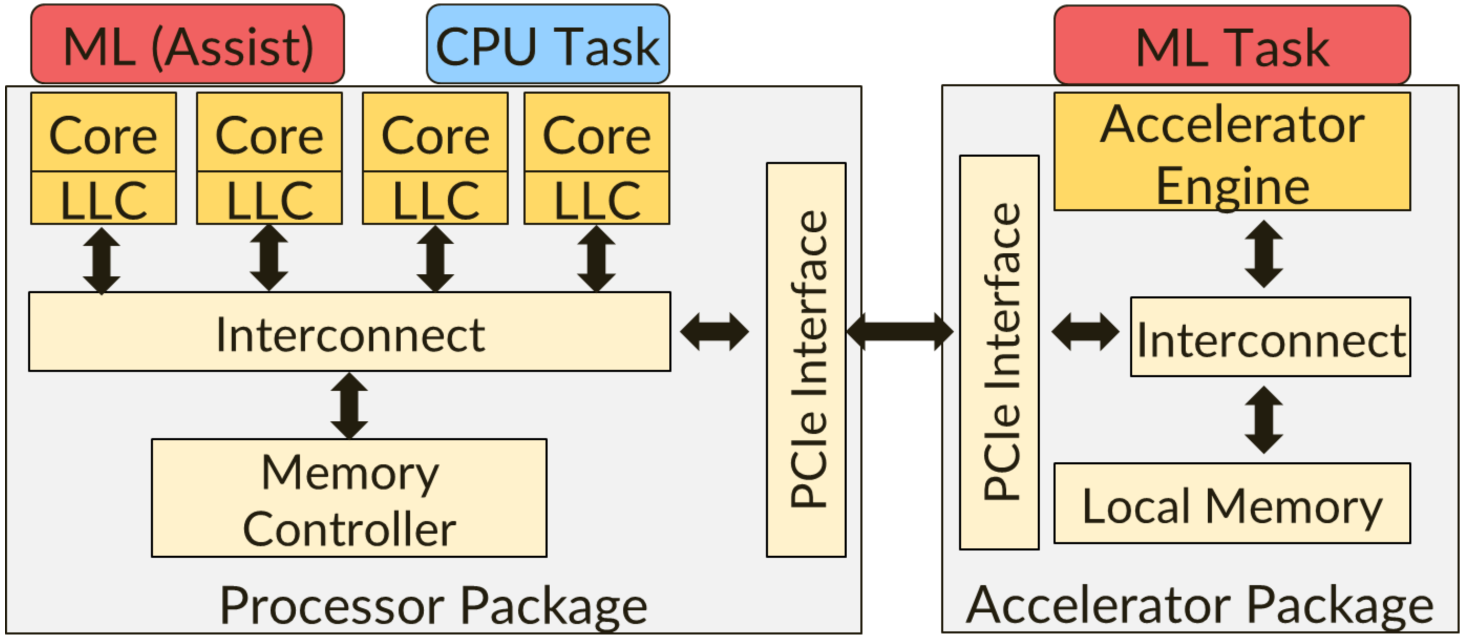
- Wide adoption of accelerators in production WSCs
  - GPU platforms are already popular in ML community
  - ASIC and FPGA based solutions have been released and deployed



Balance the tradeoff between performance of accelerated workload and hardware resource utilization

TEXAS

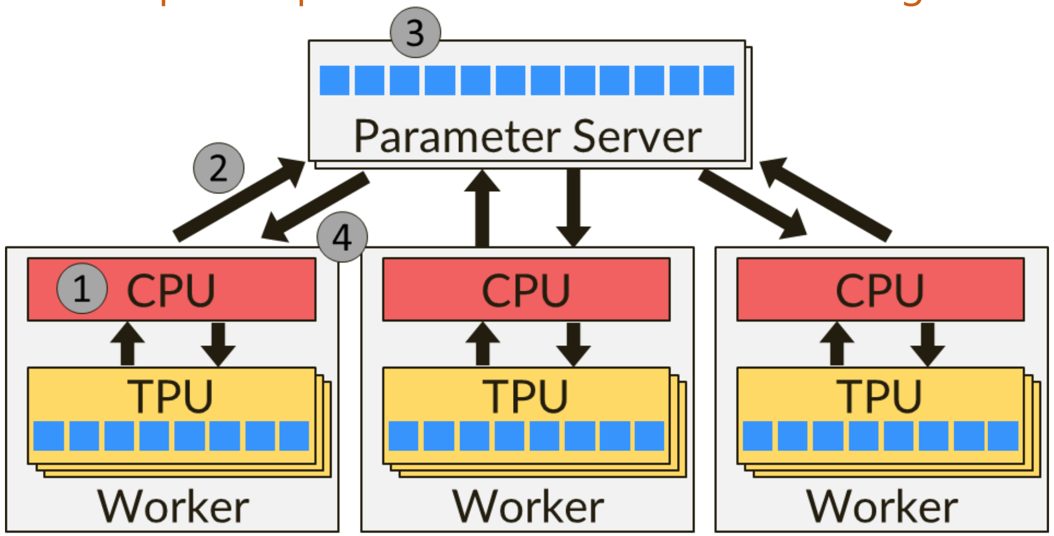
Target Architecture



- Accelerator is used by a single ML task
    - Prior work assumes time multiplexing accelerators [Chen, ASPLOS'16]
    - Performance is mostly bottlenecked by accelerator memory BW
- ML task also occupies multiple CPU cores
    - CPUs often in charge of assisting tasks that can't be easily mapped to accelerators

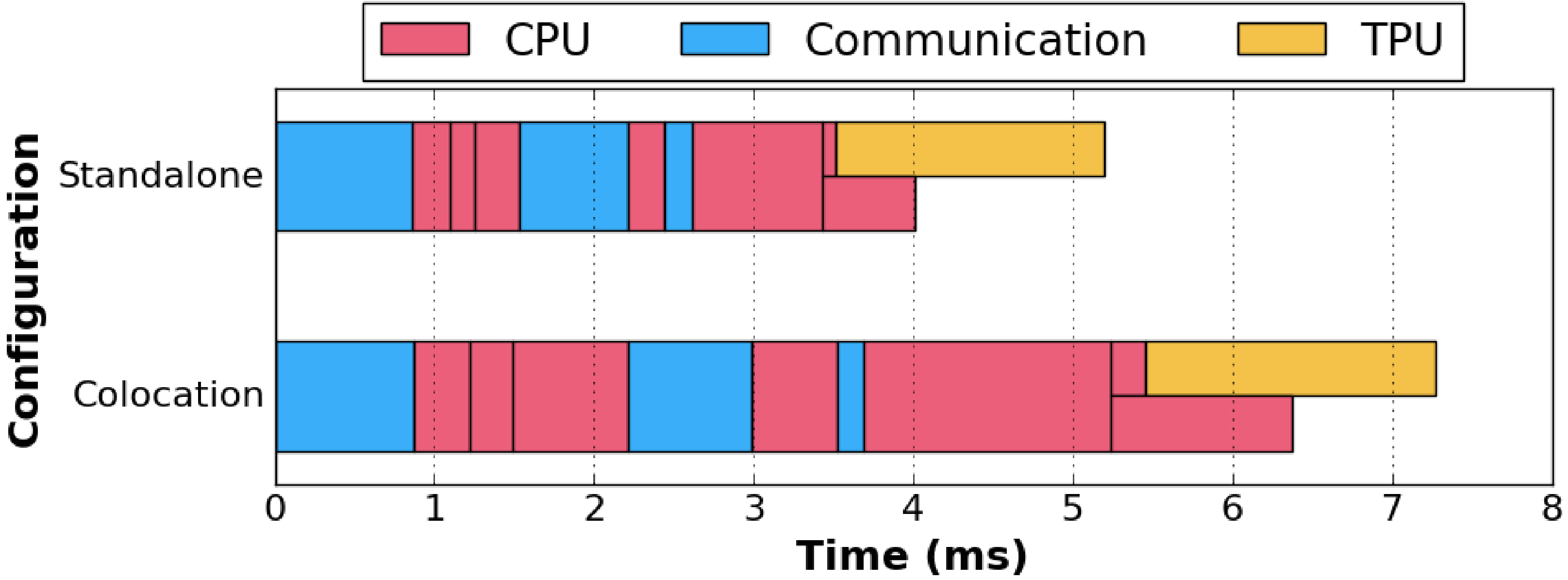
CPU-Accelerator Interaction

- Examples of assisting computation
  - Beam search
    - Sorts partial solutions and expand on a subset of best candidates
  - Data pre-processing (in-feed)
    - Interprets and reshapes data to enable efficient processing by the accelerators
  - Parameter server
    - Broadcasts and updates parameters in distributed training



- Training workloads can scale out to multiple nodes
  - Susceptible to resource interference due to “tail amplification” [Dean, 2013]

RNN Inference Server on TPU



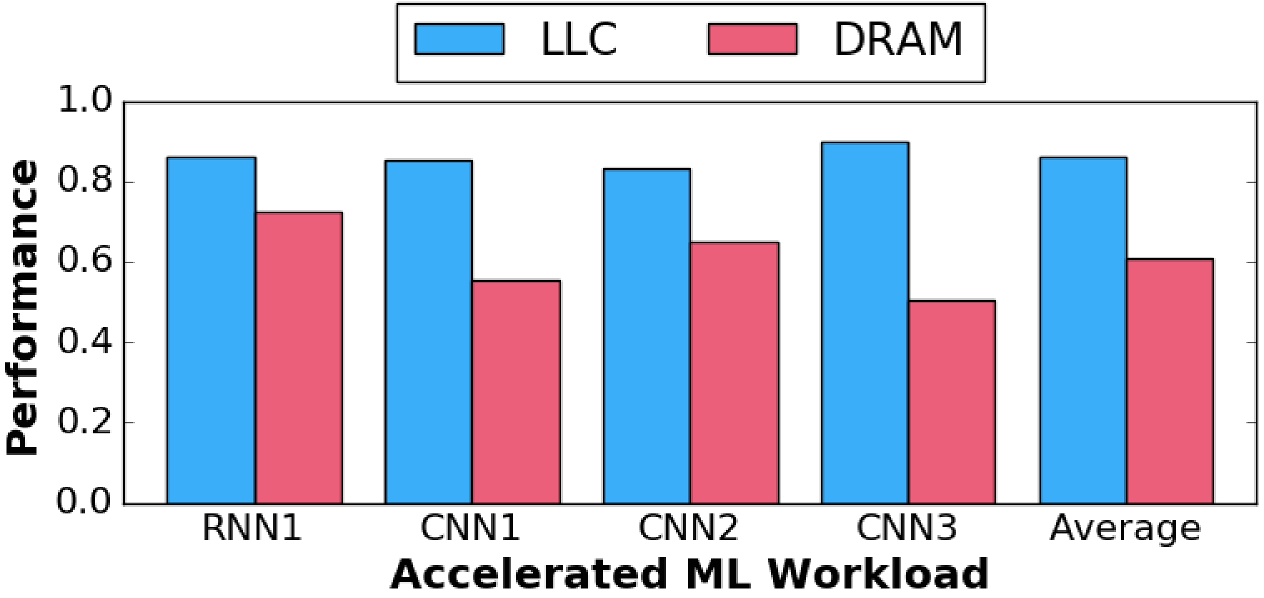
- High performance sensitivity to DRAM interference
  - Execution time for CPU-intensive phases increases significantly by 51%
  - Service-level tail latency increase by over 70%
- Sub-millisecond interleaving among different steps
  - Too fine-grained for polling-based reactive throttling
  - Highlight the needs for robust hardware performance isolation mechanism

Platforms and Workloads

Platform	Workload	Description	CPU-Accelerator Interaction
TPU	RNN1 Inference	Natural language processing	Beam search
CloudTPU	CNN1 Training	Image recognition	Data in-feed
CloudTPU	CNN2 Training	Image recognition	Data in-feed
GPU	CNN3 Training	Image recognition	Parameter server

- Requests for RNN1 inference are processed in pipeline
  - Target throughput is the knee of the tail latency curve
- CNN1 and CNN2 training on CloudTPU
  - Both rely on host CPU for data infeed operations
- GPU platforms are widely used in ML community
  - CNN3 uses distributed Tensorflow with host handling parameter server

Interference Sensitivity

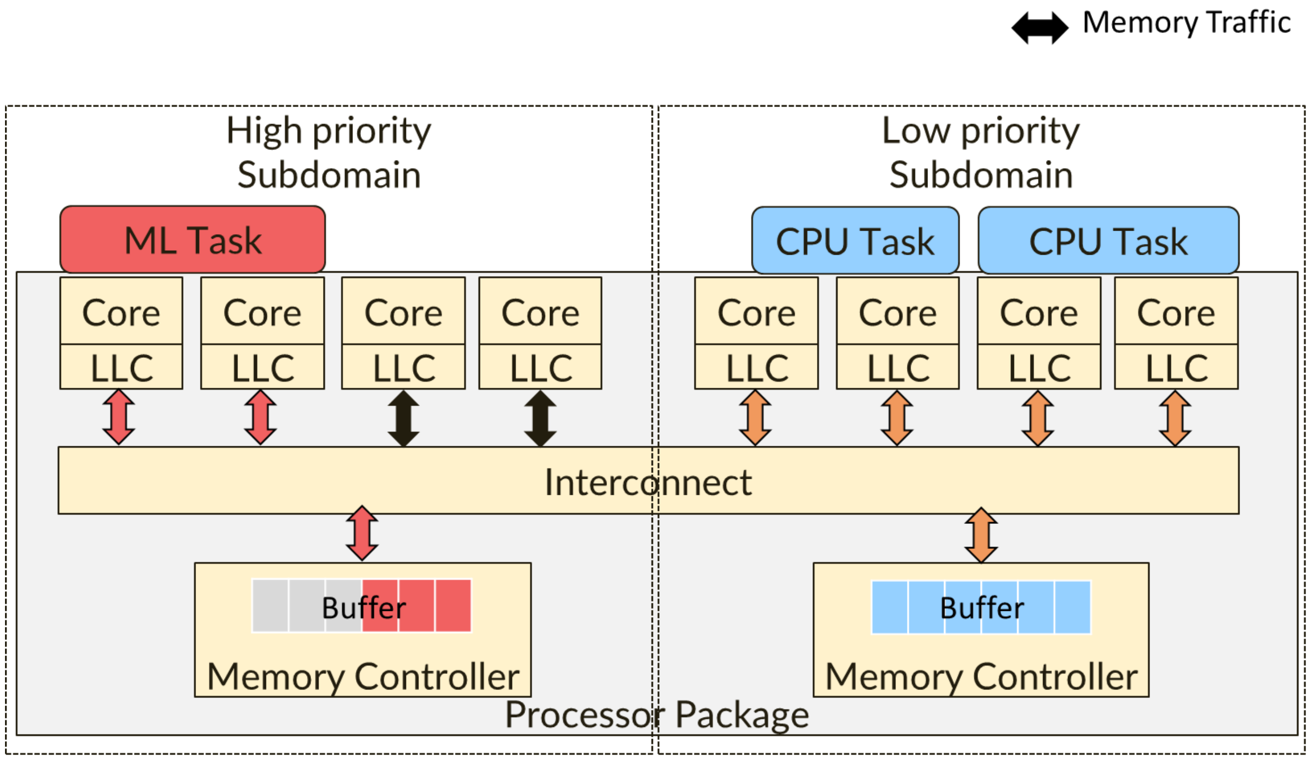


- Two types of aggressors
  - **LLC** contends for in-pipeline resources, private caches, and LLC
  - **DRAM** contends for host DRAM BW by traversing a large array
- ML workloads show higher sensitivity to **DRAM** aggressor
  - **LLC** causes 14% performance degradation
  - **DRAM** causes a dramatic 40% performance degradation

Performance interference caused by DRAM BW contention dominates the performance degradation

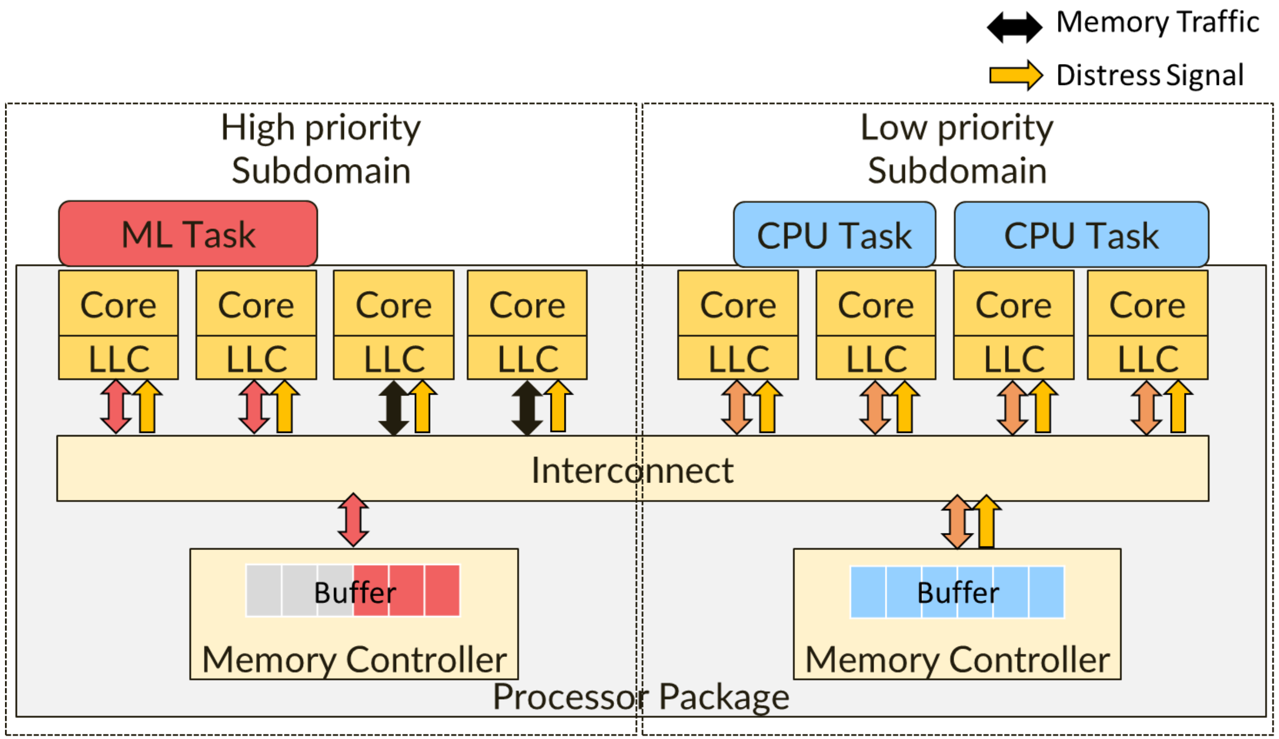
KELP: A PERFORMANCE ISOLATION RUNTIME SYSTEM

Kelp Mechanism: NUMA Subdomain



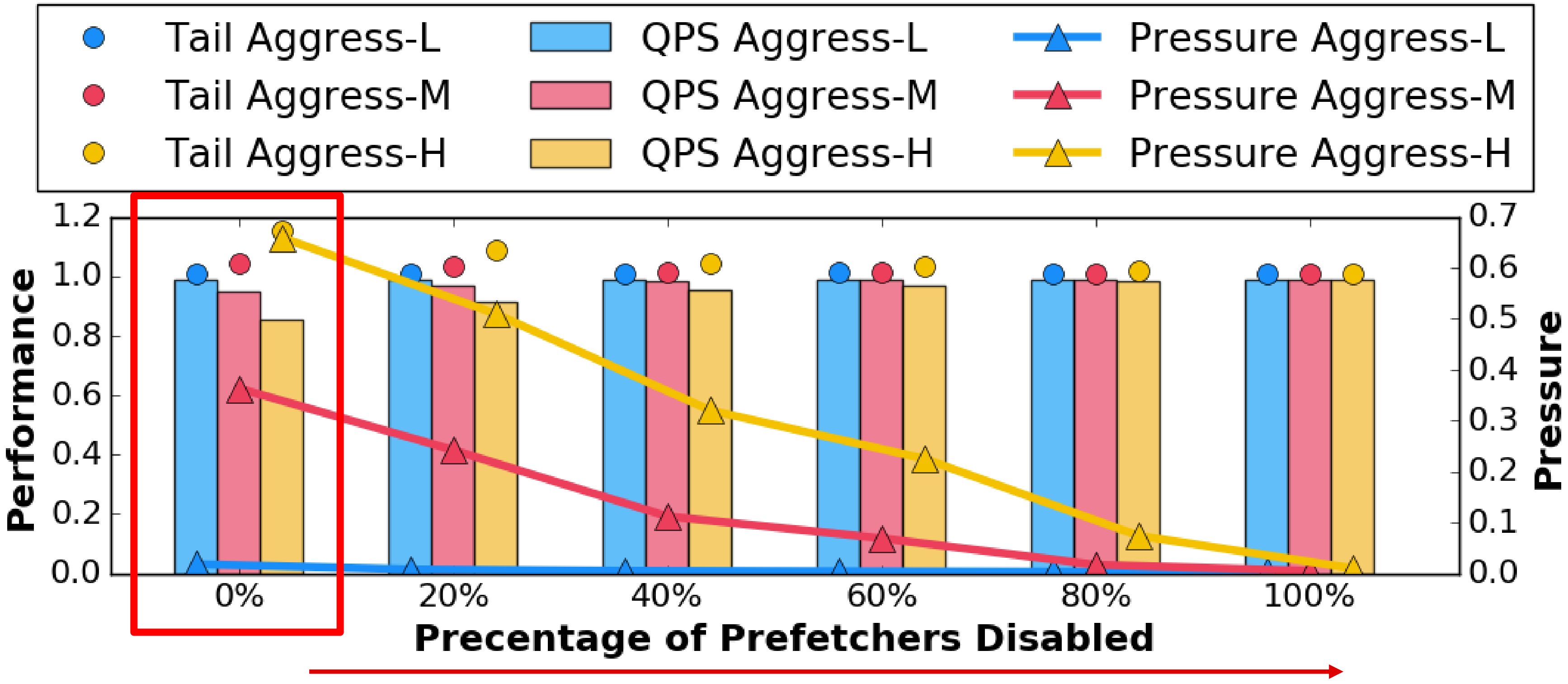
- Existing feature in Intel processors
  - Sub-NUMA Clustering (Skylake) or Cluster-on-Die (Haswell)
- Expose two NUMA domains from each socket
  - Memory traffic within a NUMA subdomain handled by its own memory controller
  - Dedicate separate subdomain to ML and CPU tasks
  - Achieves memory isolation through channel partitioning

Kelp Mechanism: Memory Pressure & Management



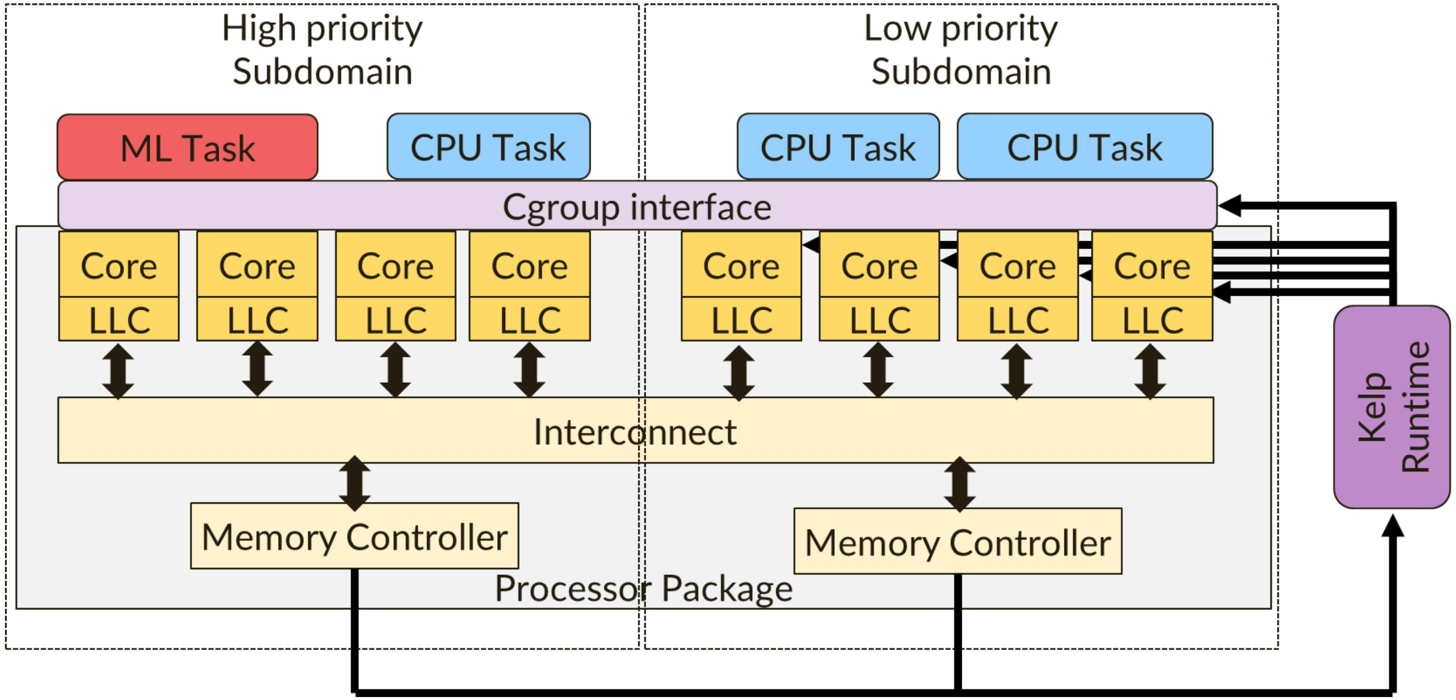
- Busy memory controller will broadcast a distress signal to all cores
  - Throttle cores to avoid unnecessary congestion in interconnect
  - Void benefits of memory isolation
- Manage memory pressure by toggling L2 prefetchers
  - Measure memory pressure using uncore performance counter
  - Toggle L2 prefetchers to keep memory pressure under threshold

Kelp Mechanism: Manage Memory Pressure



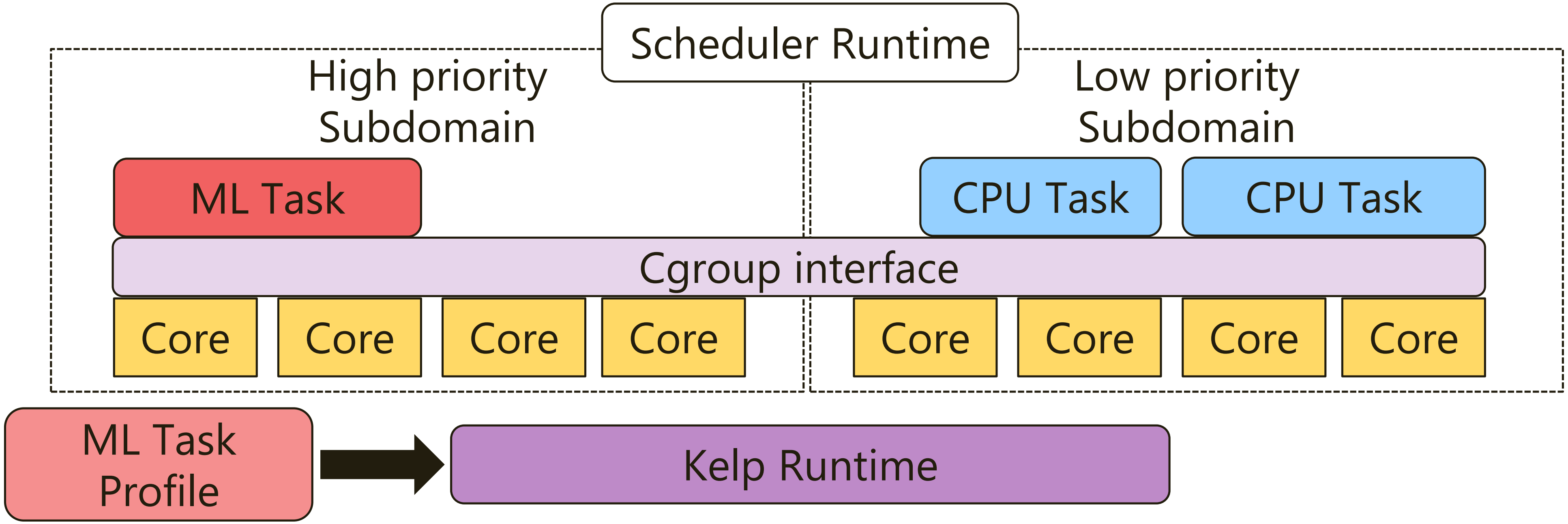
- Unmanaged memory pressure causes significant performance loss
  - Up to 14% QPS loss and 16% tail latency increase
- Turning prefetchers off effectively eliminate this effect in most cases
  - Less than 3% performance loss with 60% prefetchers turned off

Kelp Mechanism: Backfilling



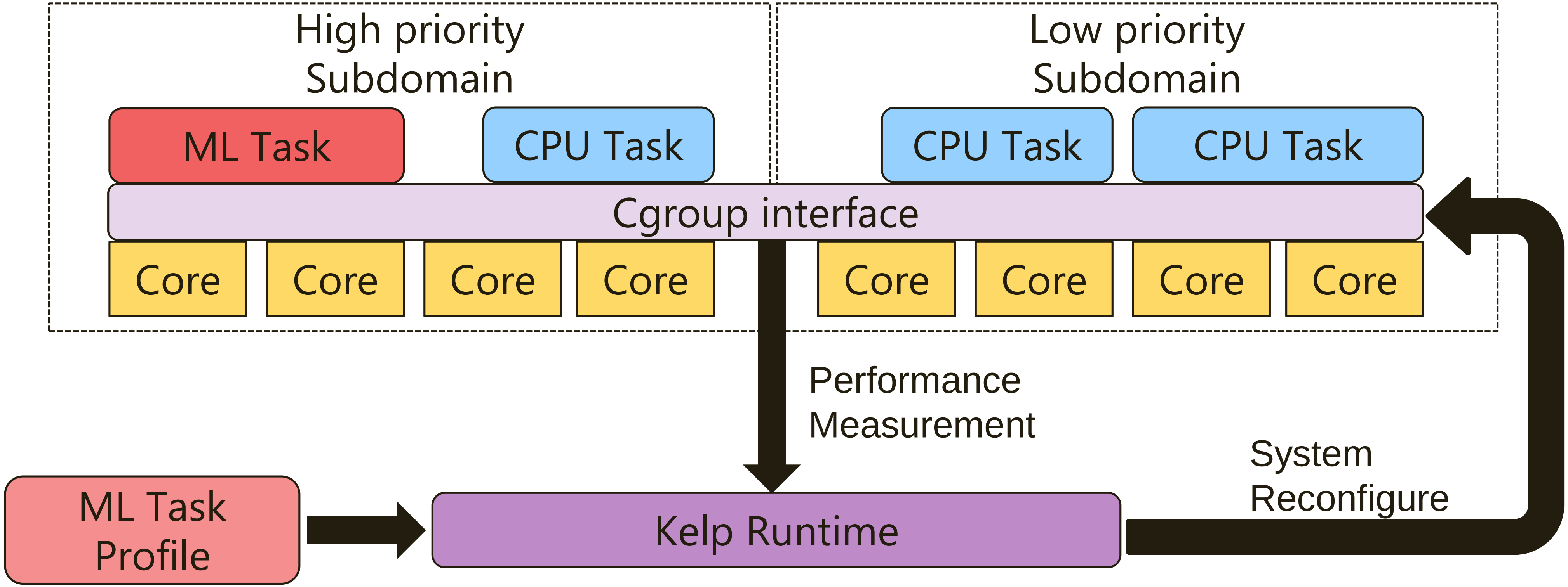
- NUMA subdomain coarsely segment CPU resources
  - Cores, Last-level cache, memory BW, etc.
- Backfill high priority subdomain with CPU tasks
  - Conservatively schedule jobs to limit the amount of interference on ML task

Kelp Runtime



- Scheduler assign tasks to the node
  - High priority ML tasks are assigned to corresponding subdomain
  - CPU tasks are prioritized to low priority subdomain
- Application specific profile is loaded at runtime
  - Specify high and low water marks for memory bandwidth, latency, and pressure

Kelp Runtime



- System performance is periodically sampled
    - Socket-level memory bandwidth, latency, memory pressure
    - High-priority subdomain memory bandwidth
- Kelp runtime reconfigures the system
    - Measurements compared against watermarks specified in task profile
    - Kelp reconfigures CPU masks and toggles L2 prefetchers

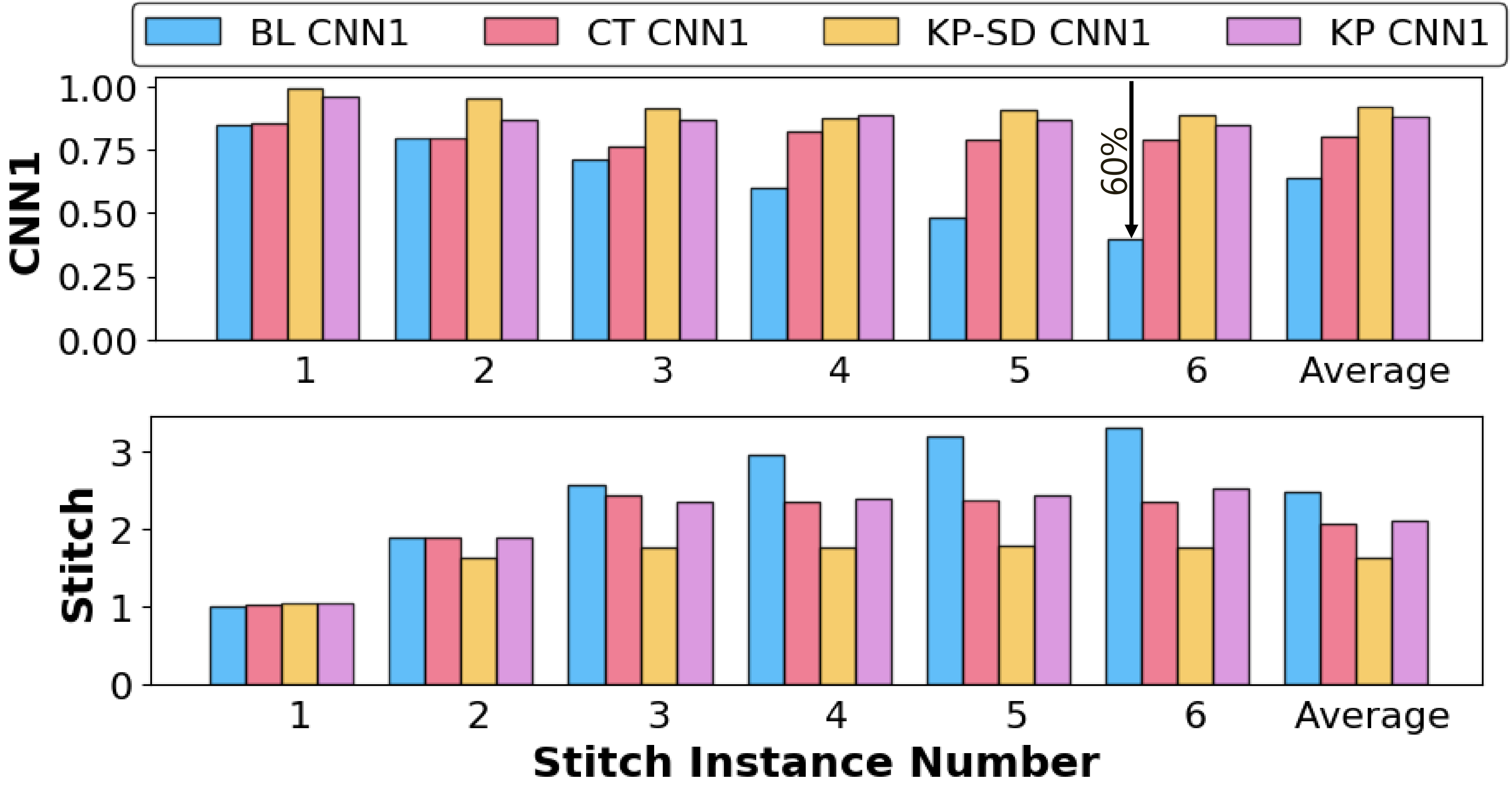
Evaluation Methodology

- CPU Workloads
  - Stream: artificial aggressor that iterate over a large array
  - Stitch: image stitching for Google street view
  - CPU ML: CPU-based training based on TensorFlow-Slim

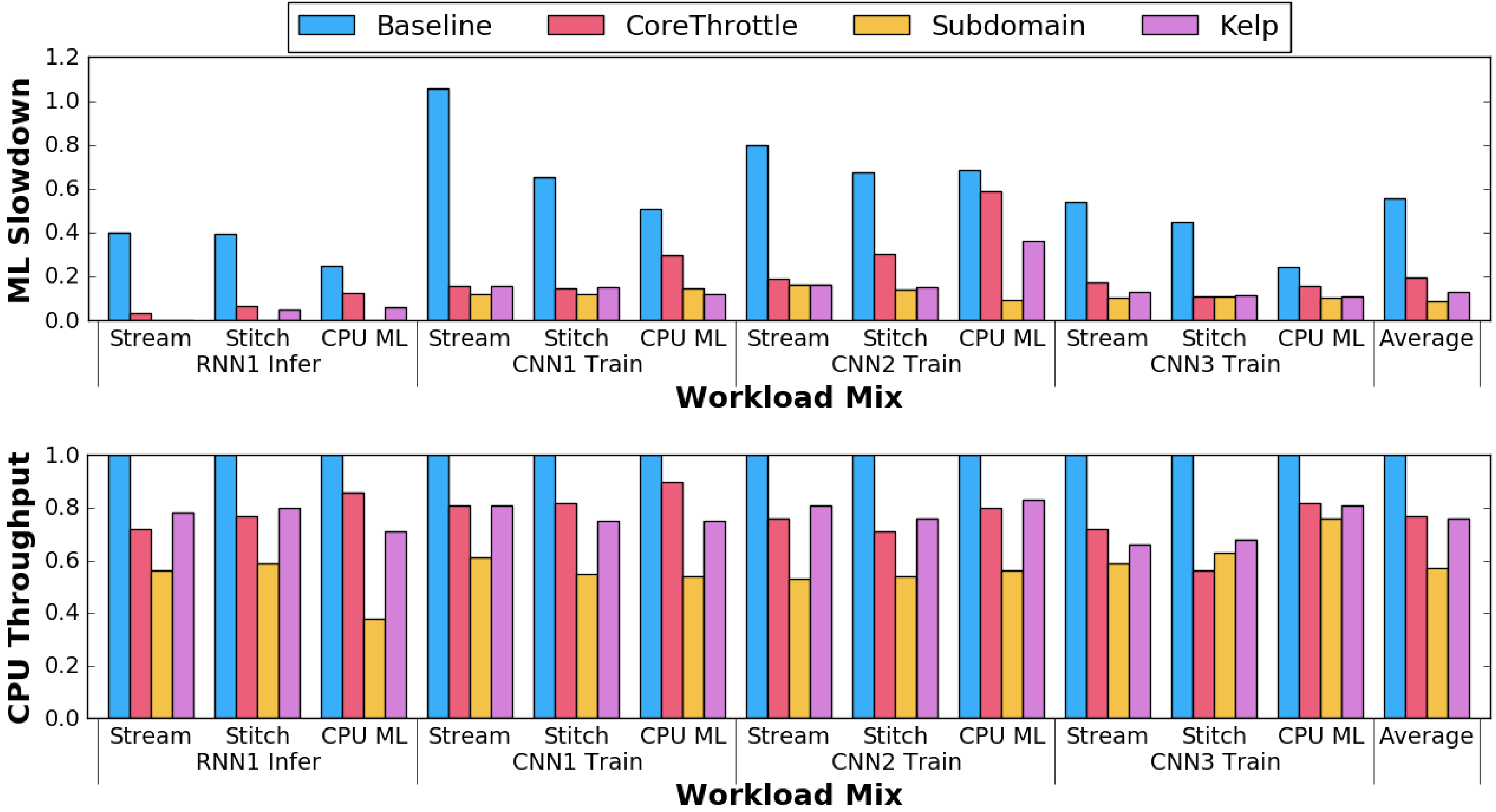
- Configurations
  - Baseline (BL)
    - Contention unmanaged except for priority maintained by WSC scheduler
  - CoreThrottle (CT)
    - Limit number of cores and LLC partitions available to the CPU tasks [Lo, ISCA'15]
  - Kelp Subdomain (KP-SD)
    - Use NUMA Subdomain to isolate performance and manage memory pressure
  - Kelp (KP)
    - Full Kelp implementation with backfilling high priority subdomain

Case Study

- Workload Mix: CNN1 + Stitch
  - CNN1 is highly sensitive to BW contention
  - Stitch heavily contends for DRAM BW

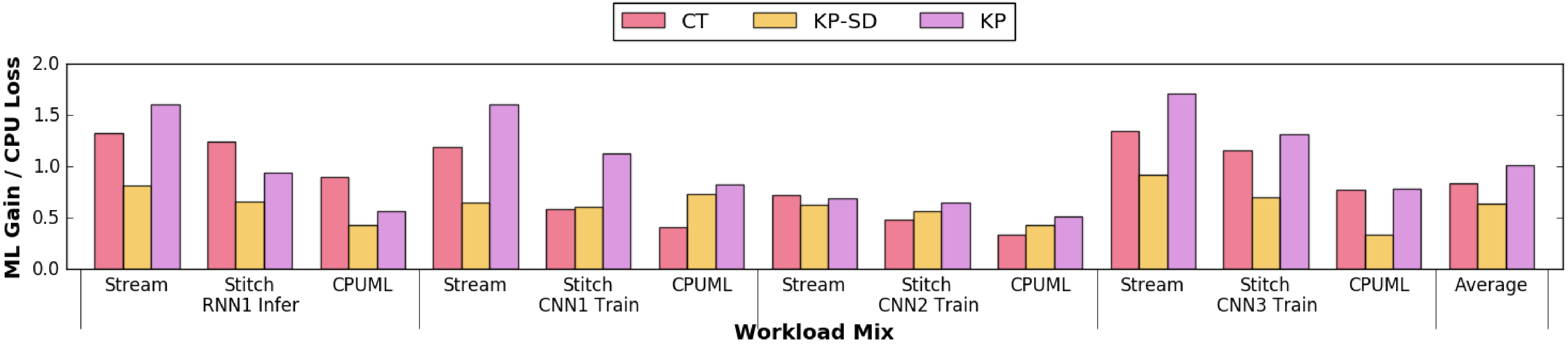


Evaluation Result Summary



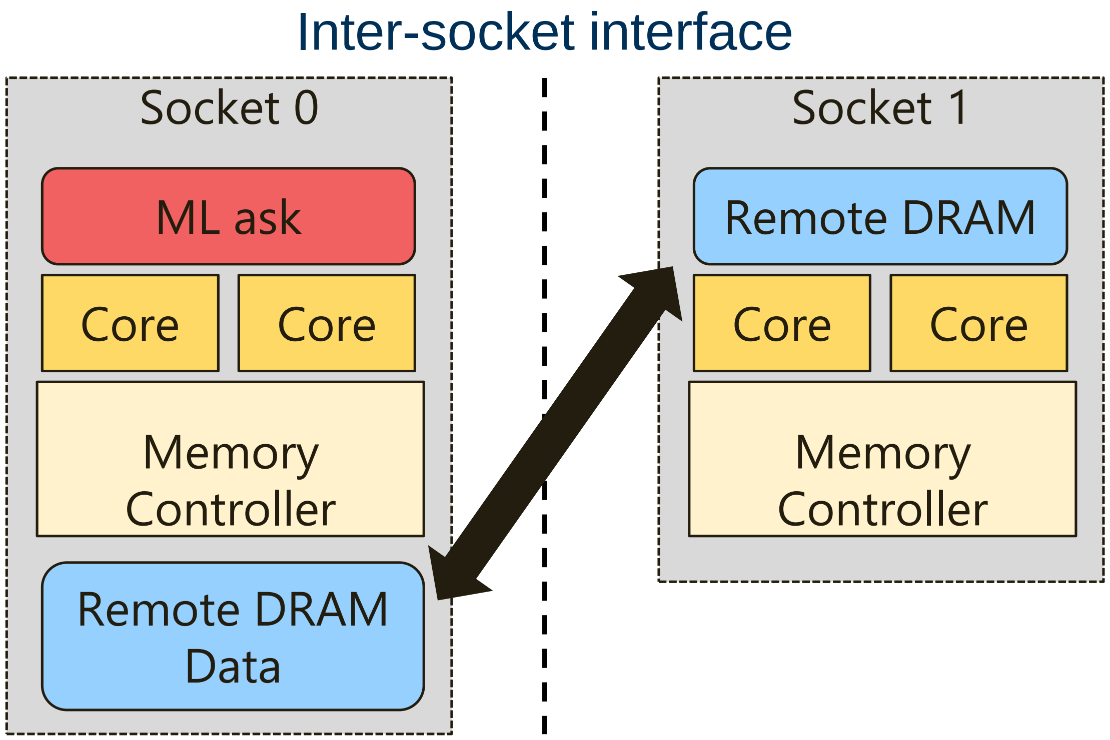
Result Summary

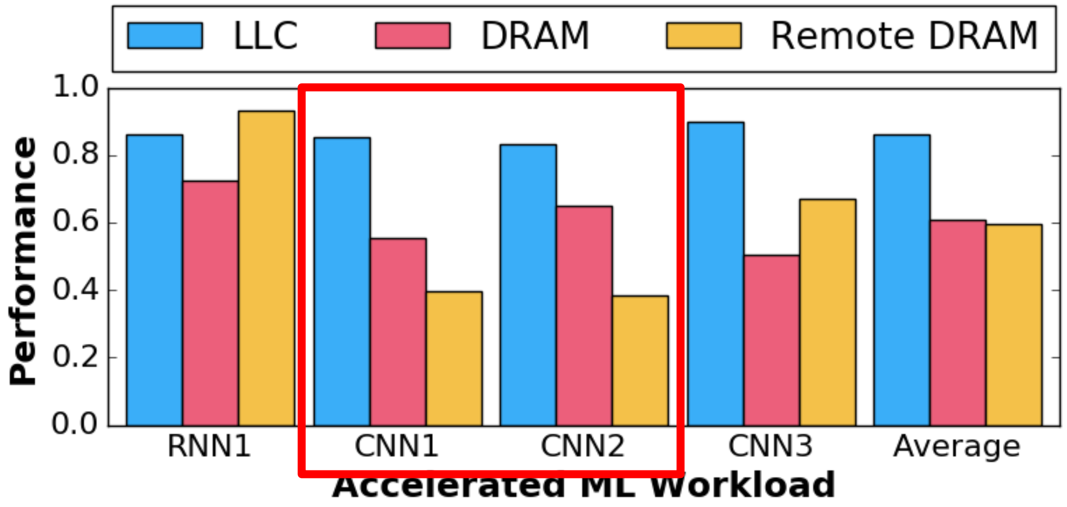
- Define "efficiency" to compare all configurations
  - $$\text{Efficiency} = \frac{\text{Perf gain of ML tasks}}{\text{Perf loss of CPU tasks}}$$



CPU Design Challenges

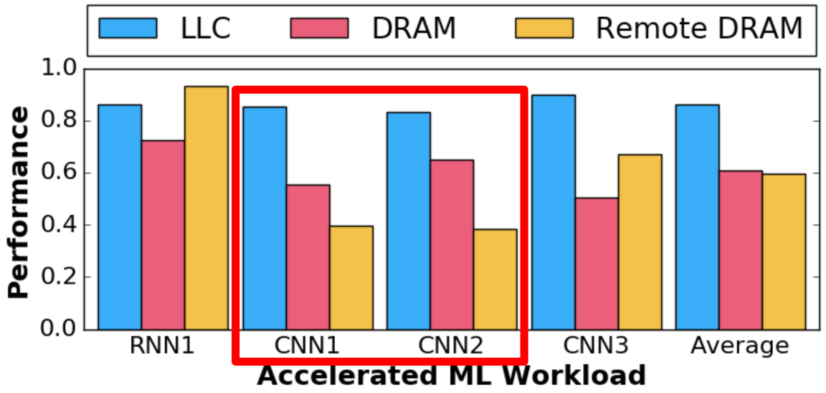
- Remote memory performance interference
  - Remote DRAM traffic can cause surprisingly large performance degradation

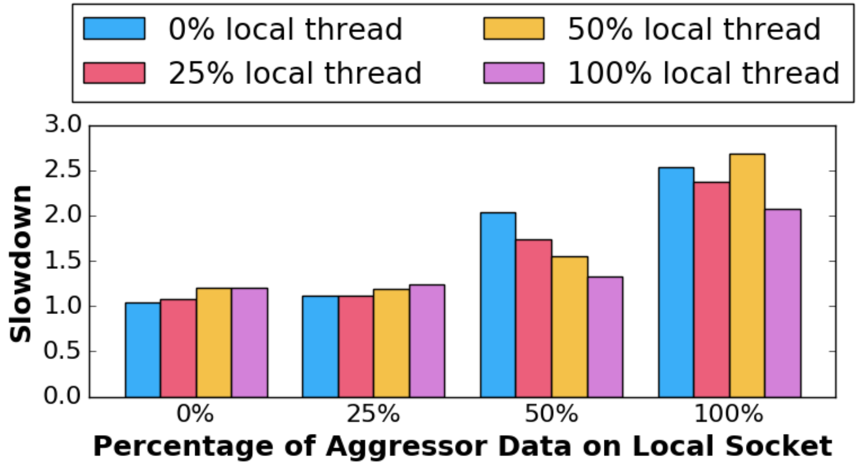




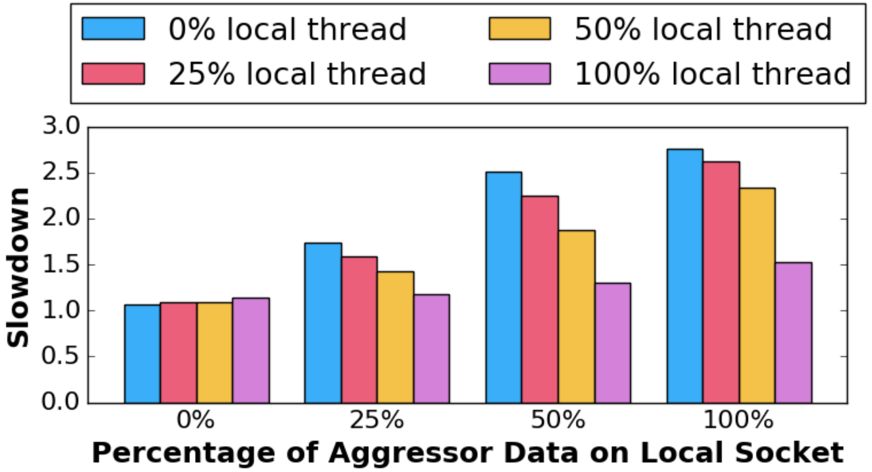
CPU Design Challenges

- Remote memory performance interference
  - Remote DRAM traffic can cause surprisingly large performance degradation





CNN1



CNN2

Conclusion

- Investigate memory resource interference on accelerated ML platforms
  - Identify fine time granular host-accelerator interaction
  - Show high sensitivity to CPU memory resource contention and low sensitivity to core resources contention
- Kelp: a runtime solution that mitigates performance interference using existing CPU features
  - NUMA subdomain and memory pressure monitoring to achieve performance isolation
  - Improve efficiency compared to previous work by 17%
- Demonstrate multiple challenges posed by high-performance accelerators
  - Fine-grained memory performance isolation can further improve system efficiency