QoS for Accelerator-Rich “Fat” Nodes

Mattan Erez

The University of Texas at Austin

Two types of tasks on nearly all systems

- **Best effort** (BE)
- **Latency critical** (LC)

- Latency critical
  - User-facing: UI, games, video, …
  - System-facing: sync, comm, …
- Best-effort
  - User background tasks
  - Learning, optimizations, …
  - System management
- Latency sensitive
  - CPU-oriented tasks
  - Tightly-iterating algorithms
- Latency insensitive
  - Graphics
  - Big DNNs
  - …

Warehouse-scale computers (WSCs)

- User-facing (eventually) tasks are latency sensitive
- But WSC is wasted if no background tasks

http://e.huawei.com/uk/case-studies/global/2016/201609271019

What does **QoS** mean?

Traditional QoS goals:

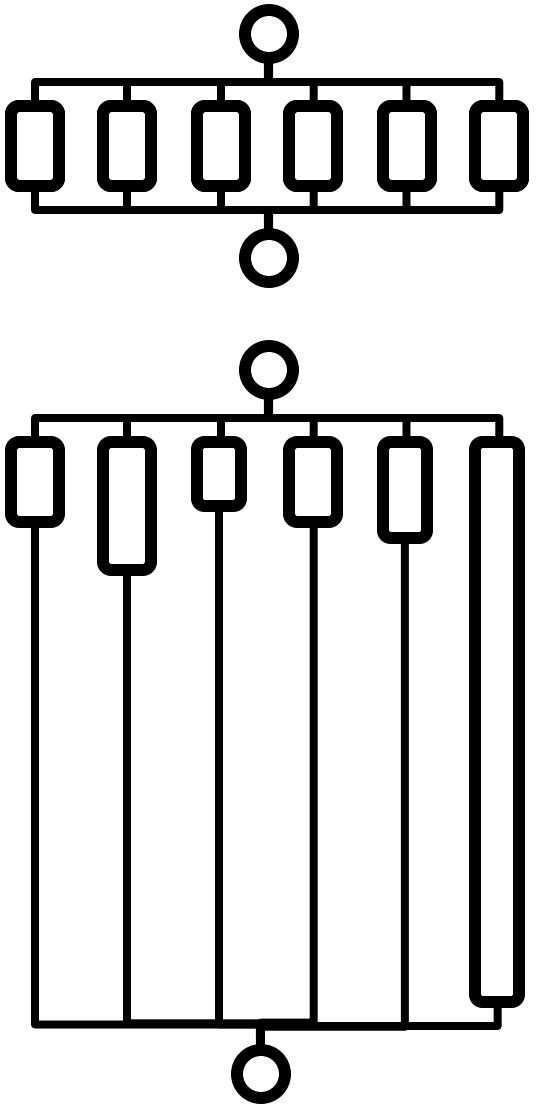
- Fairness
- Completion-time targets
  - No missed deadlines (hard realtime)
  - Often translated to strict scheduling constraints

Better QoS goals:

- Unfairness (priority)
- Completion-time targets
  - Percentile deadline targets
  - Tail statistics

Better QoS goals:

- Unfairness (priority)
- Completion-time targets
  - Percentile deadline targets
  - **Tail statistics**



Context: single node within a WSC

- Shared between LC and BE tasks
- Accelerators
- Lots of cores
- Heterogeneous memories

Goals:

(1) Meet completion-time percentile targets

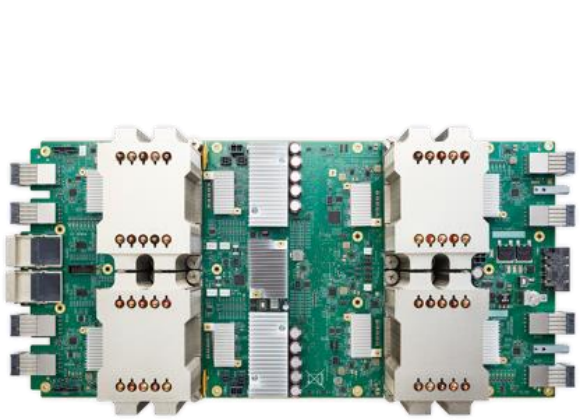
(2) No significant tail expansion

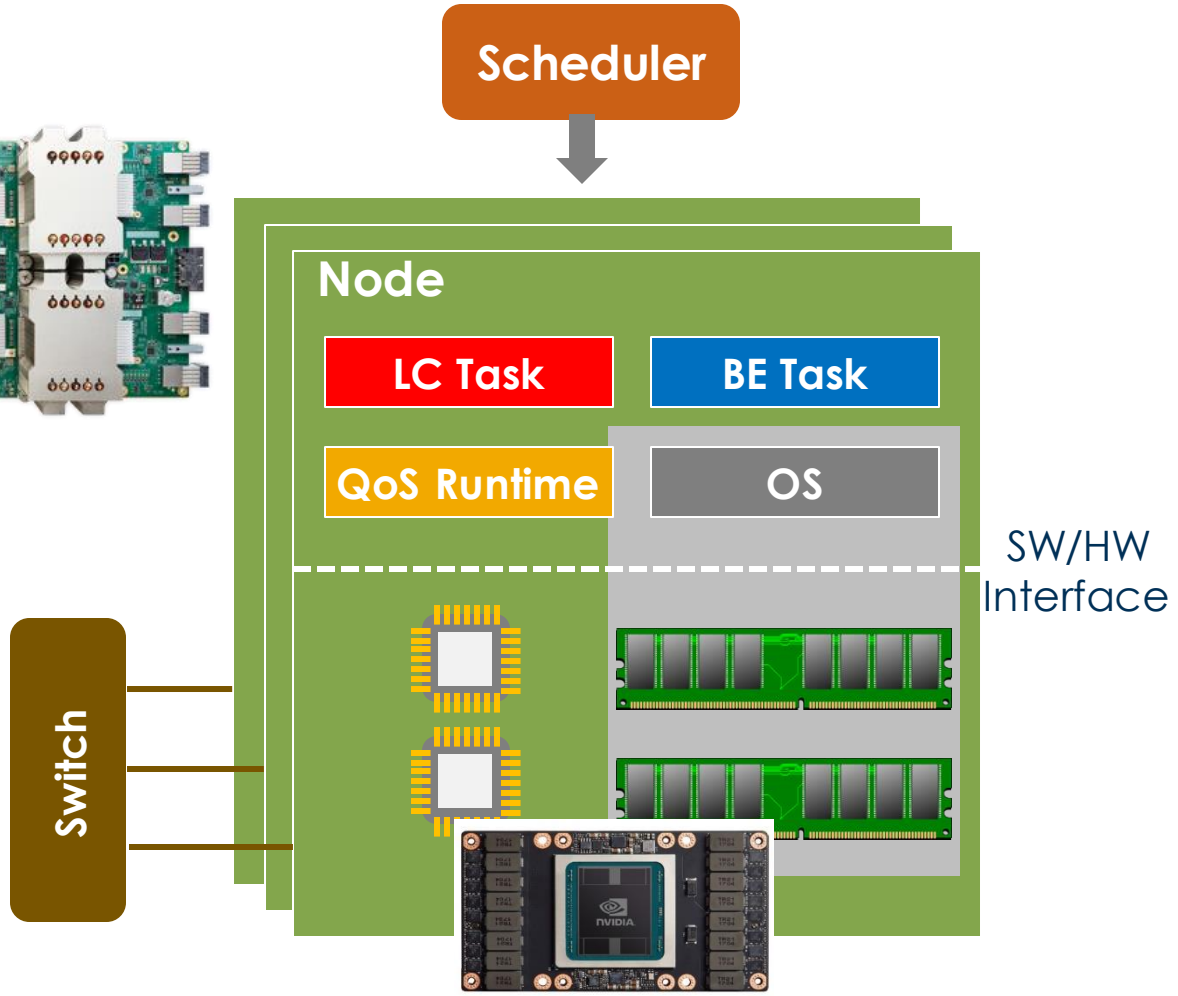
(3) Maximize best-effort task throughput

(4) Prioritize best-effort tasks as needed

BE interference with LC tasks on an accelerator

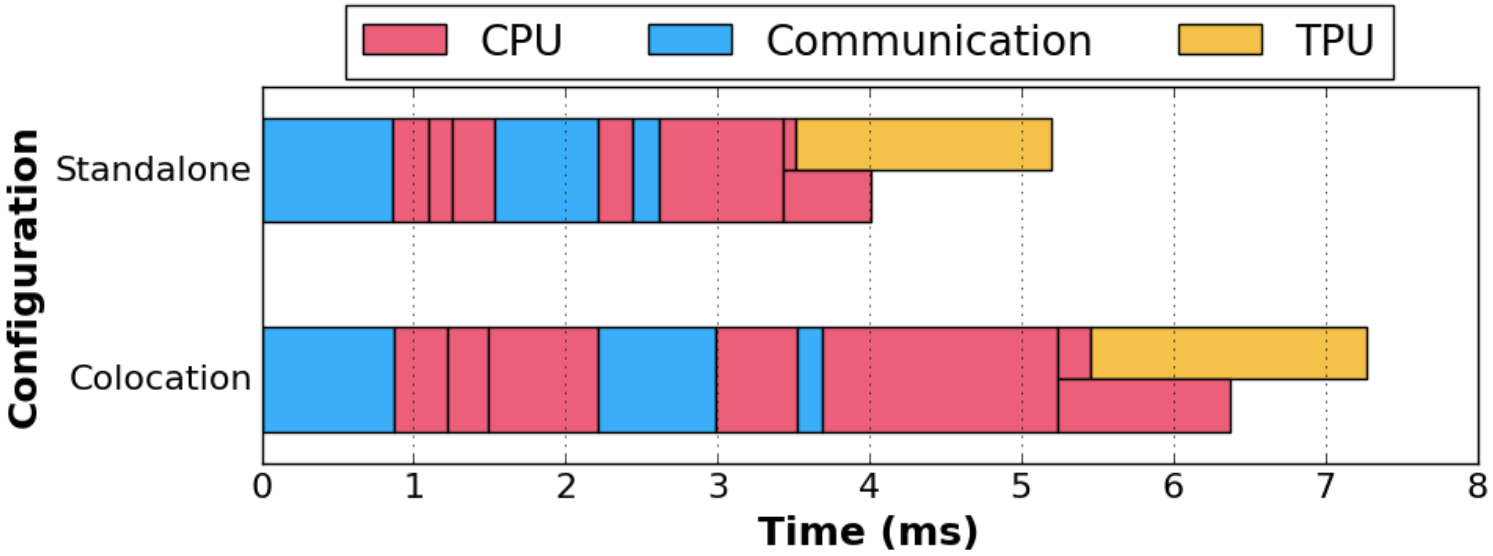
- From: Haishan Zhu et al., “Kelp” HPCA 2019
- Many thanks to Google collaborators





Performance interference with a TPU

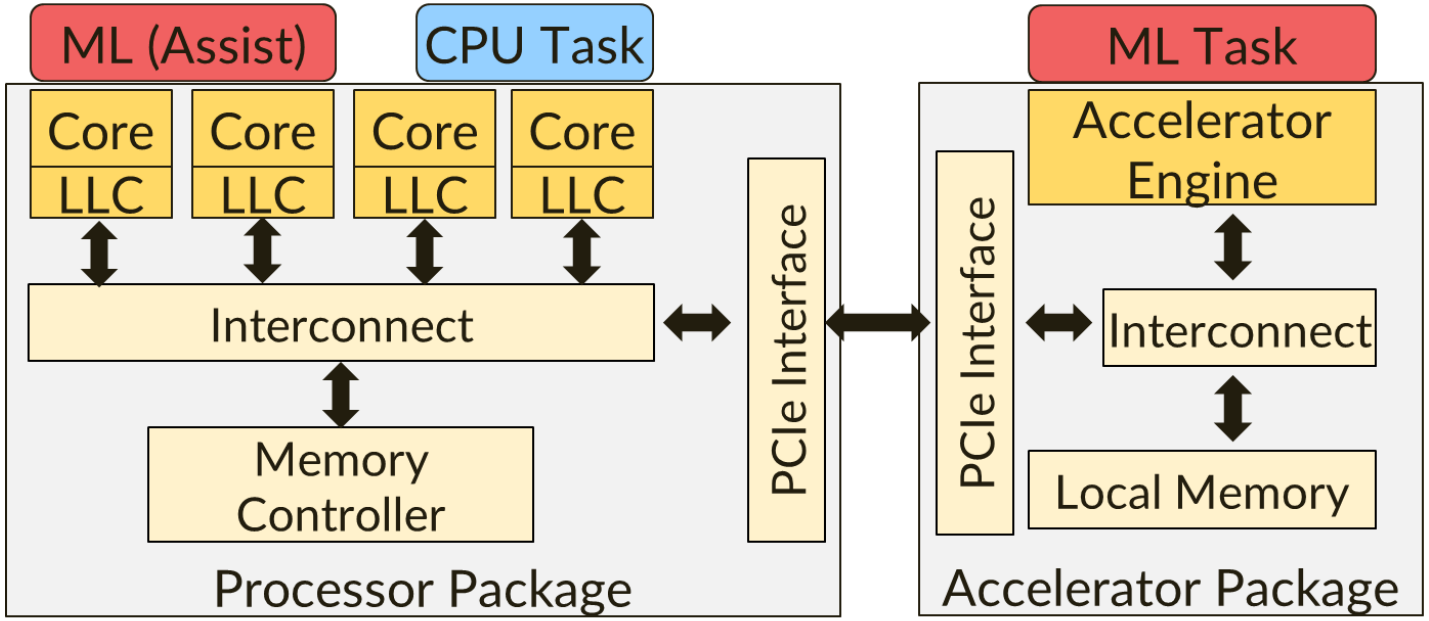
- High sensitivity to DRAM interference
  - Despite TPU having own DRAM and PCIe traffic too low
- Need hardware solutions
  - Sub-millisecond interleaving of task steps



Single ML task occupies accelerator

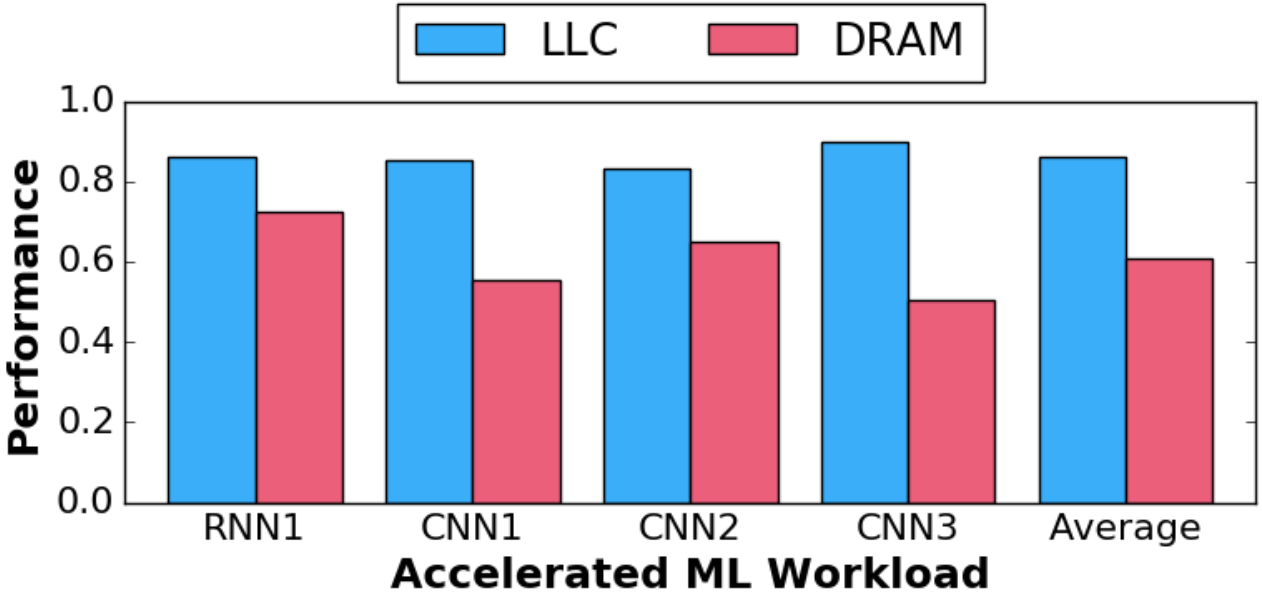
ML task also occupies multiple CPU cores

- Preprocessing
- Beam search
- Parameter servers
- …



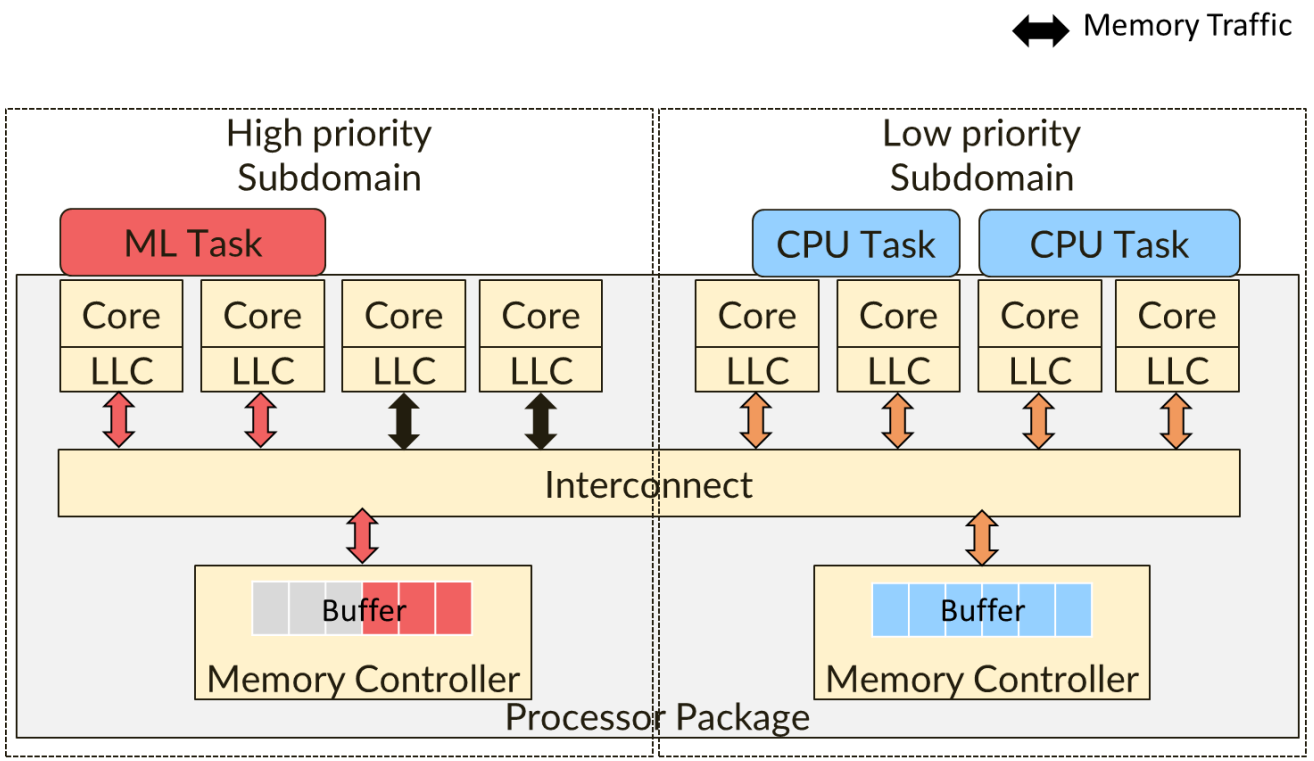
DRAM interference is dominant

– Aggressor microbenchmarks



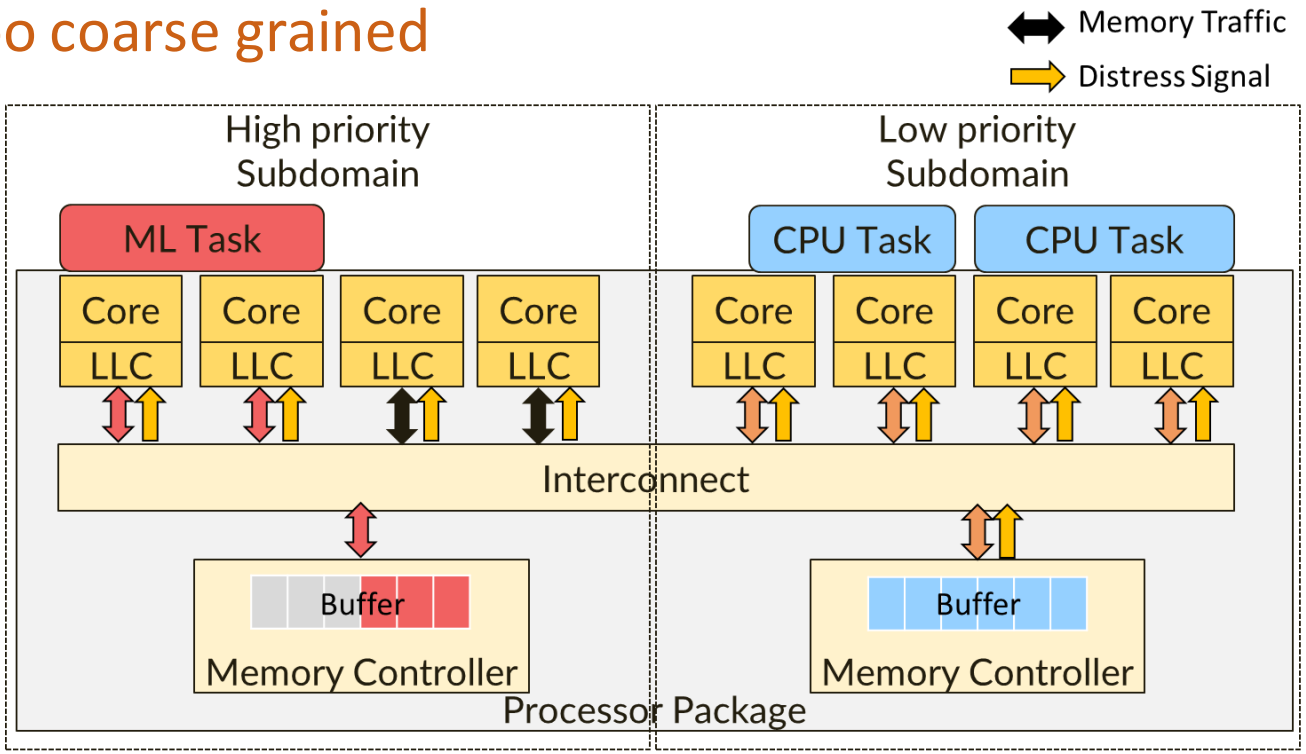
How to control DRAM interference?

– NUMA subdomains (i.e., channel partitioning)

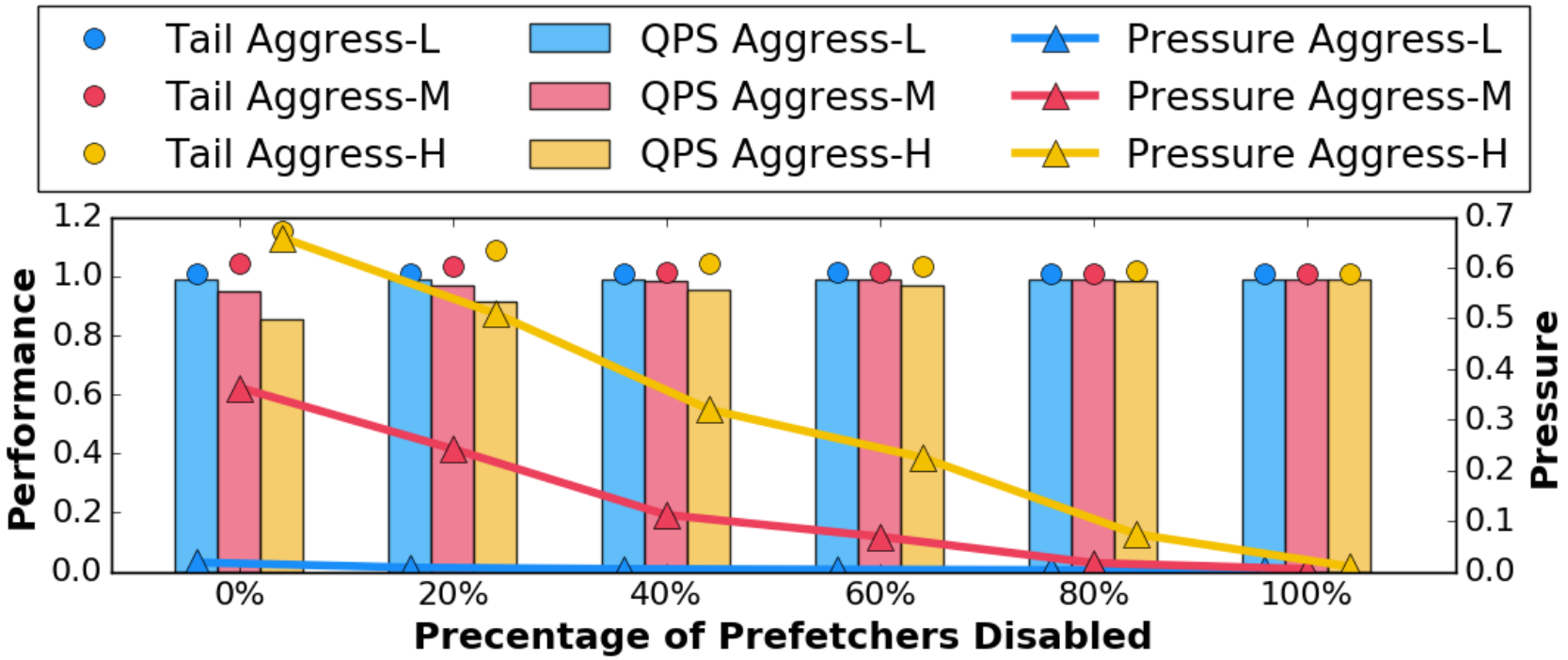


How to control DRAM interference?

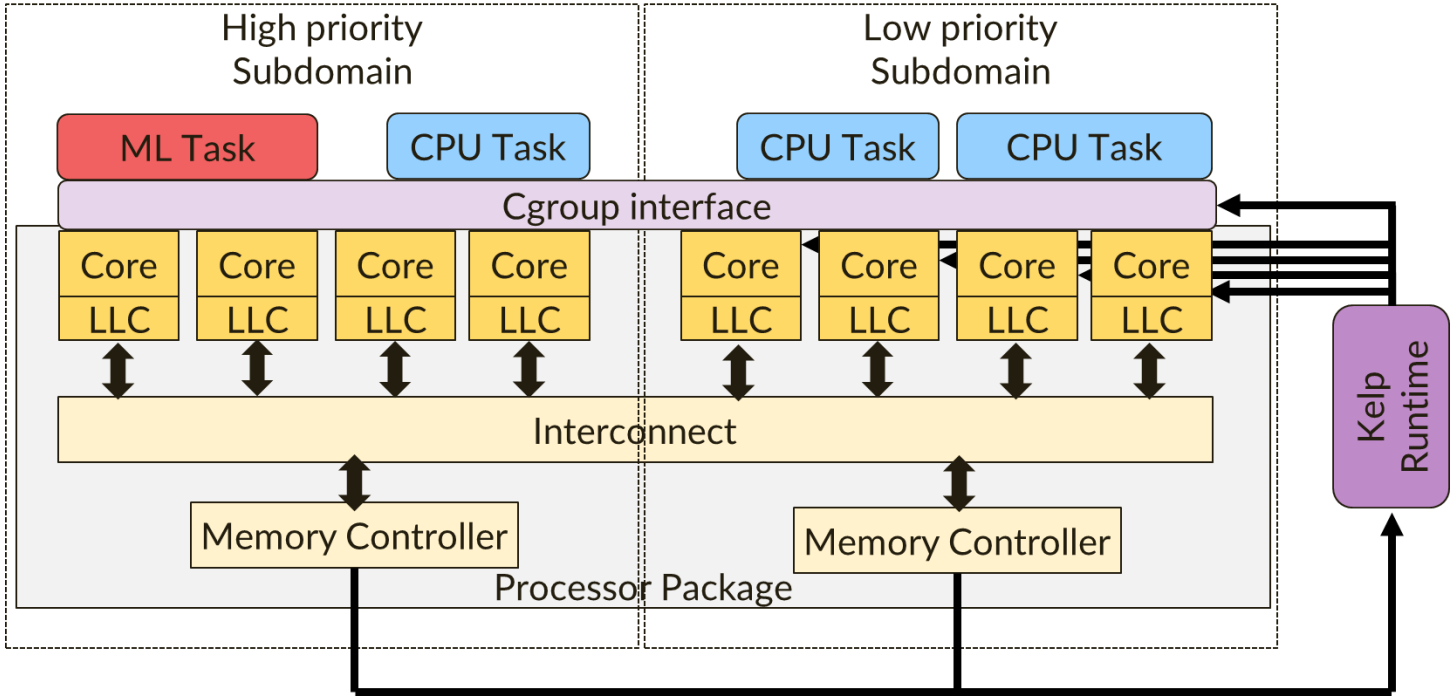
- NUMA subdomains (i.e., channel partitioning)
  - On Xeon we tried, occasionally failed to control interference (quite surprising)
  - Too coarse grained



Kelp mechanism 1: Manage prefetchers



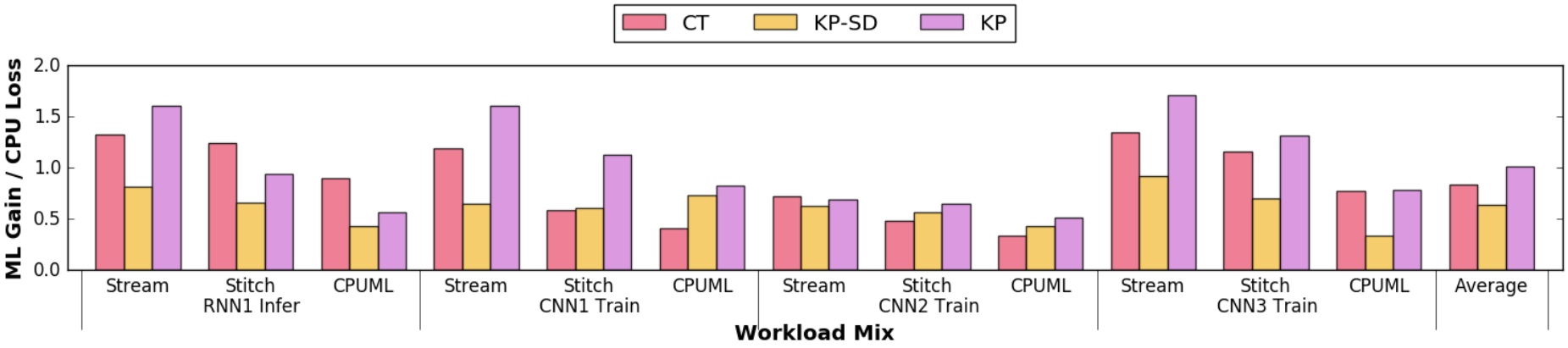
Kelp Mechanism 2: Backfilling



Result summary – Kelp works pretty well

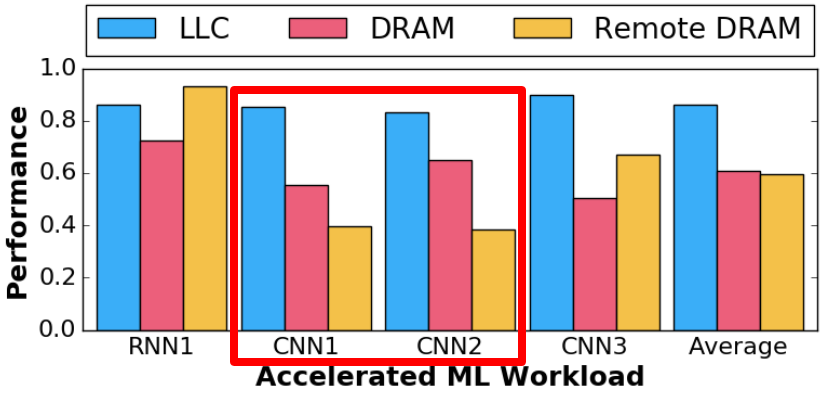
– Tradeoffs hard to visualize, so

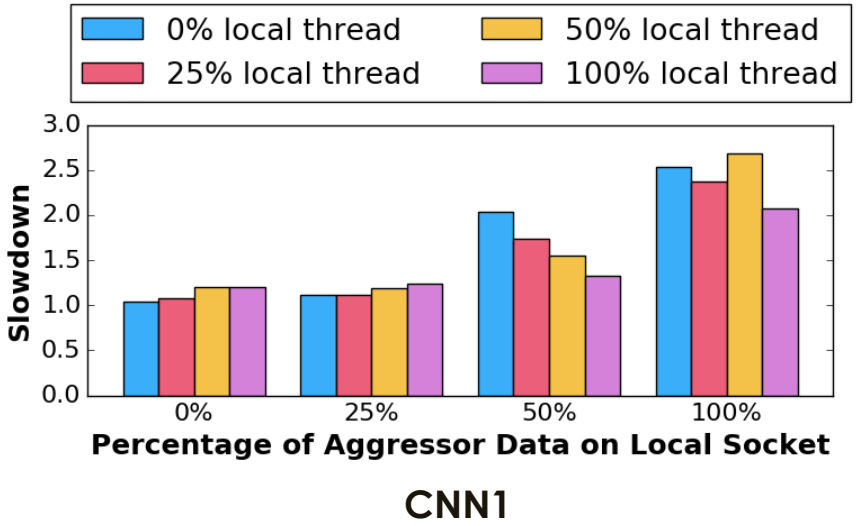
$$\text{Efficiency} = \frac{\text{Perf gain of ML tasks}}{\text{Perf loss of CPU tasks}}$$

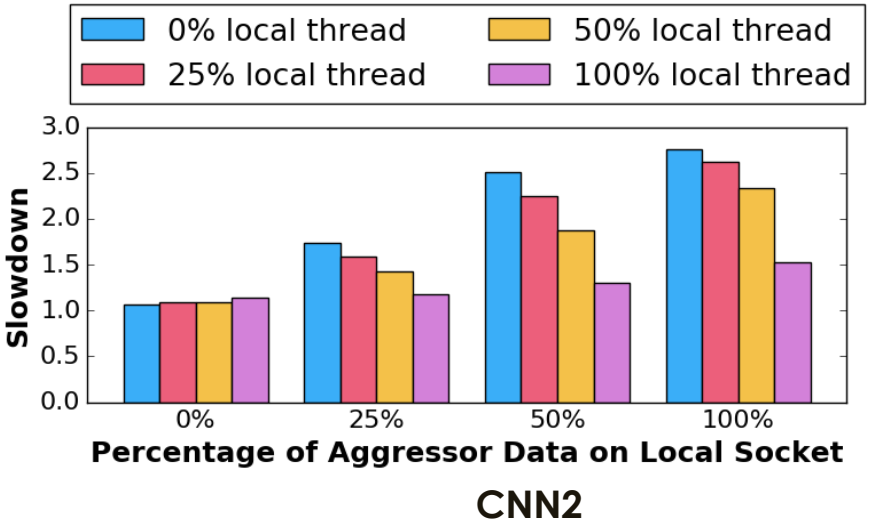


Challenges remain:

- Multi-socket can lead to worse inteference







Accelerators are challenging, but so are

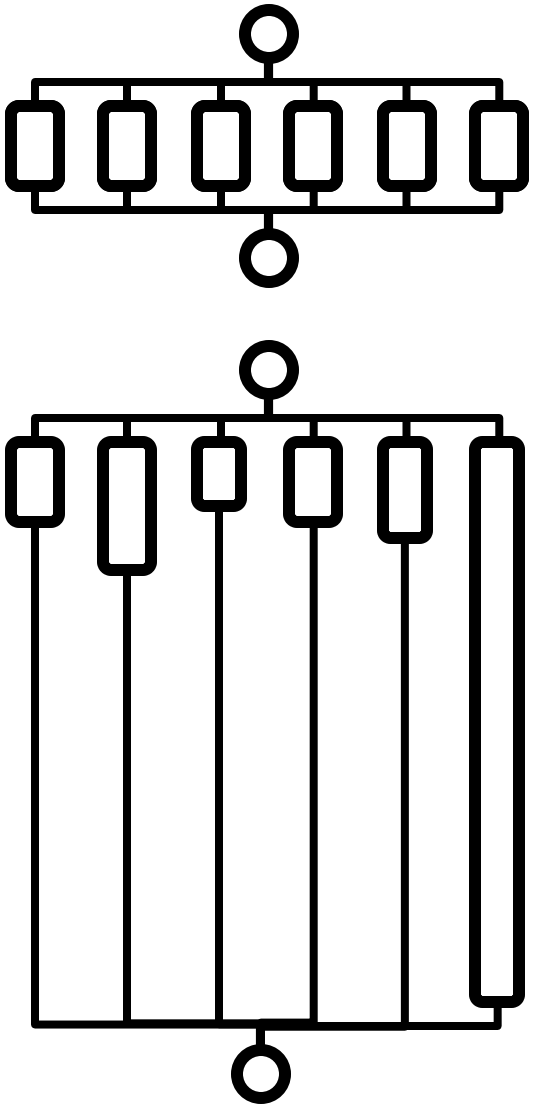
- Lots of cores
  - Completion time tracking to the rescue? (e.g, Zhu and Erez, “Dirigent”, ASPLOS 2016)
- Heterogeneous memories
  - Priority over partitioning really promising

Dirigent: Enforcing QoS for Latency-Critical Tasks on Shared Multicore Systems

Haishan Zhu and Mattan Erez **ASPLOS 2016**

How to ensure QoS?

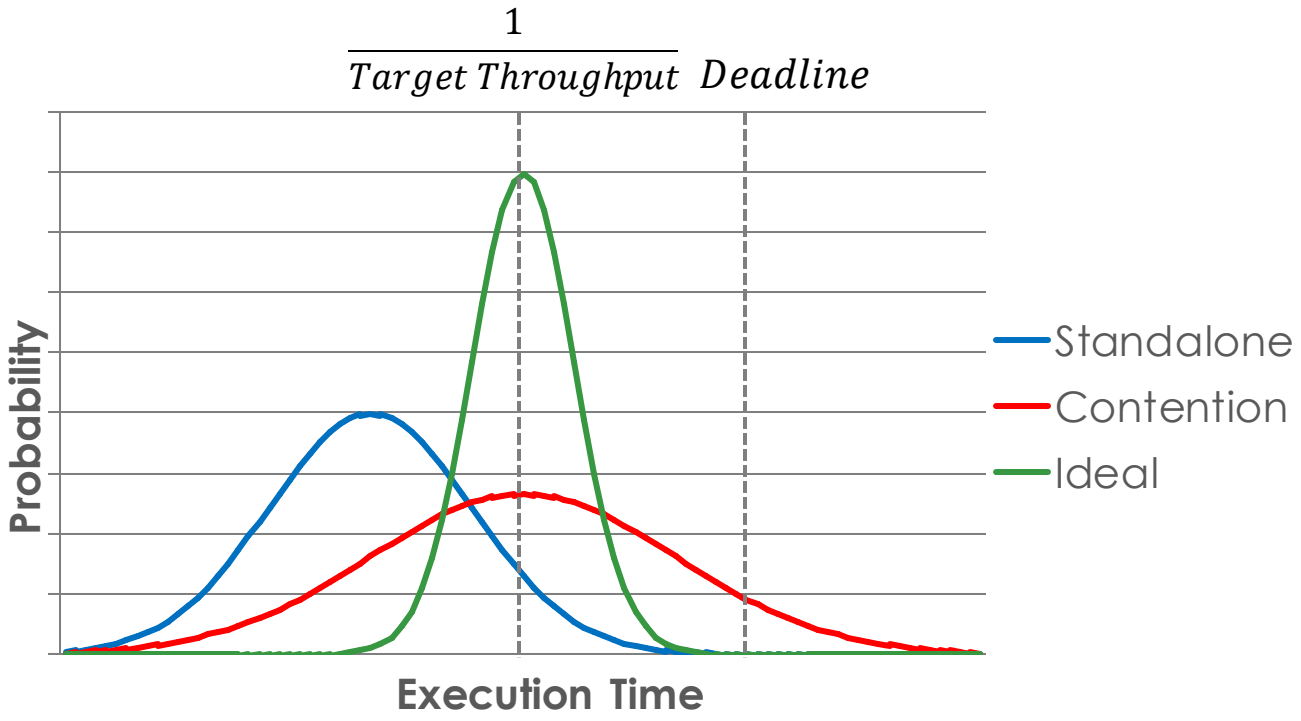
- Don't assign too much work
  - Shutdown unused resources
  - Very wasteful
- Backfill with "compatible" work
  - Not latency-critical
  - Hopefully not too conflicting



In practice severe waste

- Forced to schedule conservatively
- Performance variation a big concern

Performance variation causes wasted resources



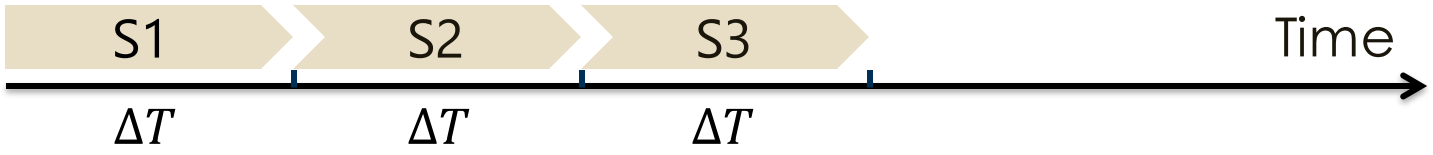
Performance variation of latency-sensitive tasks indicates the amount of resources wasted

Dirigent uses application information to **shape completion time distribution**

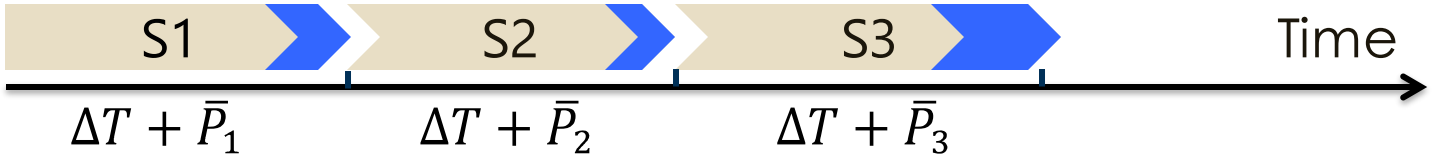
– Also improves scheduling quality

Execution time prediction

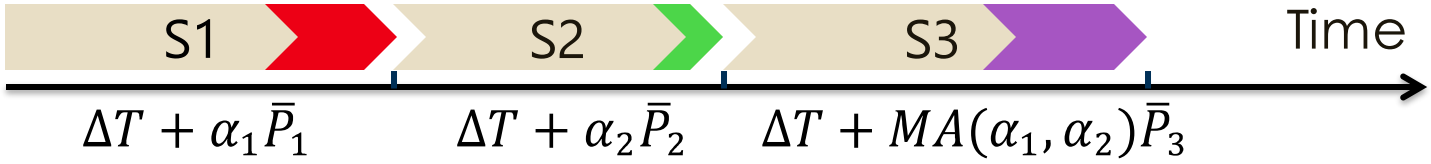
Profiling while Running Standalone



Average Execution Time Penalty



Execution Time Prediction



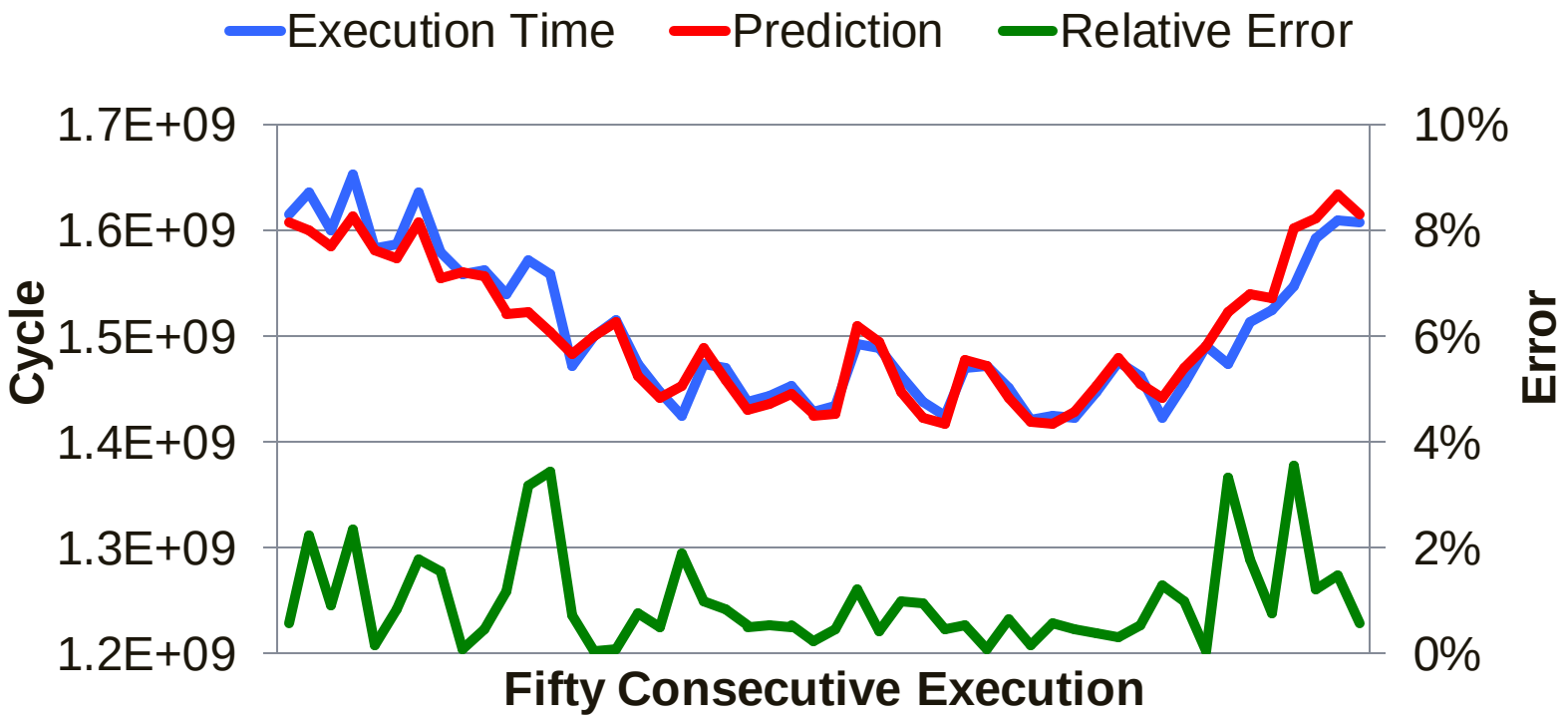
ΔT: *time duration of a sample segment*
Sn: *program progress during segment n*

$\bar{P}_i$: *average time penalty of the* i^{th} *segment*

$$\alpha_i = \frac{profiled\_progress_i}{measured\_progress_i}$$

$MA(\cdot)$: *moving average*

Execution time predictor accuracy



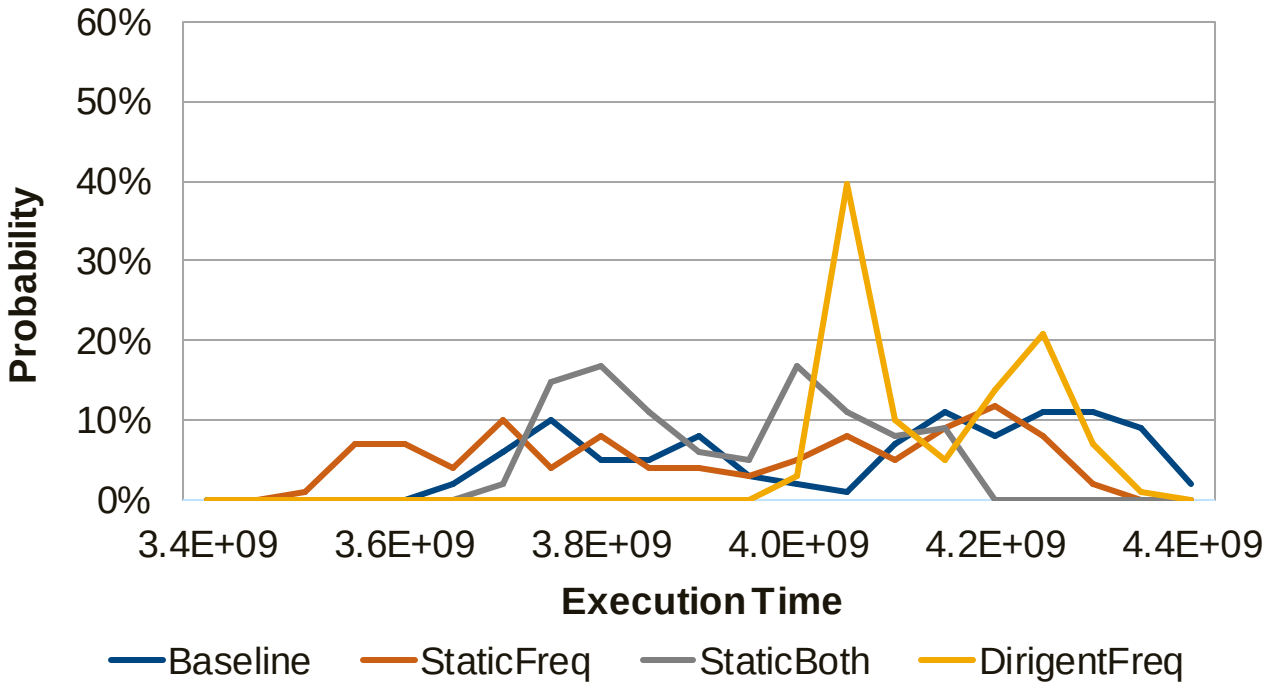
50 consecutive executions of raytrace (FG) and RS (BG)

Predictions are made halfway through each FG execution

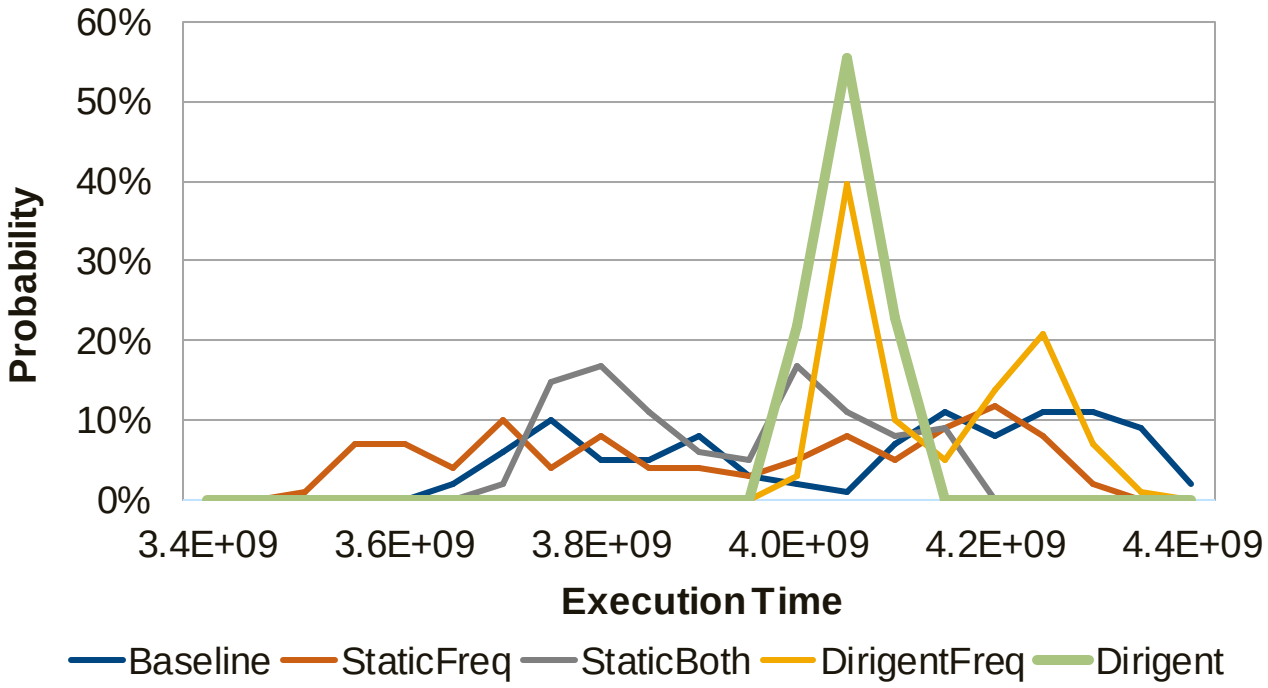
In future, should also use application-level hints

Dirigent results: single LC workloads

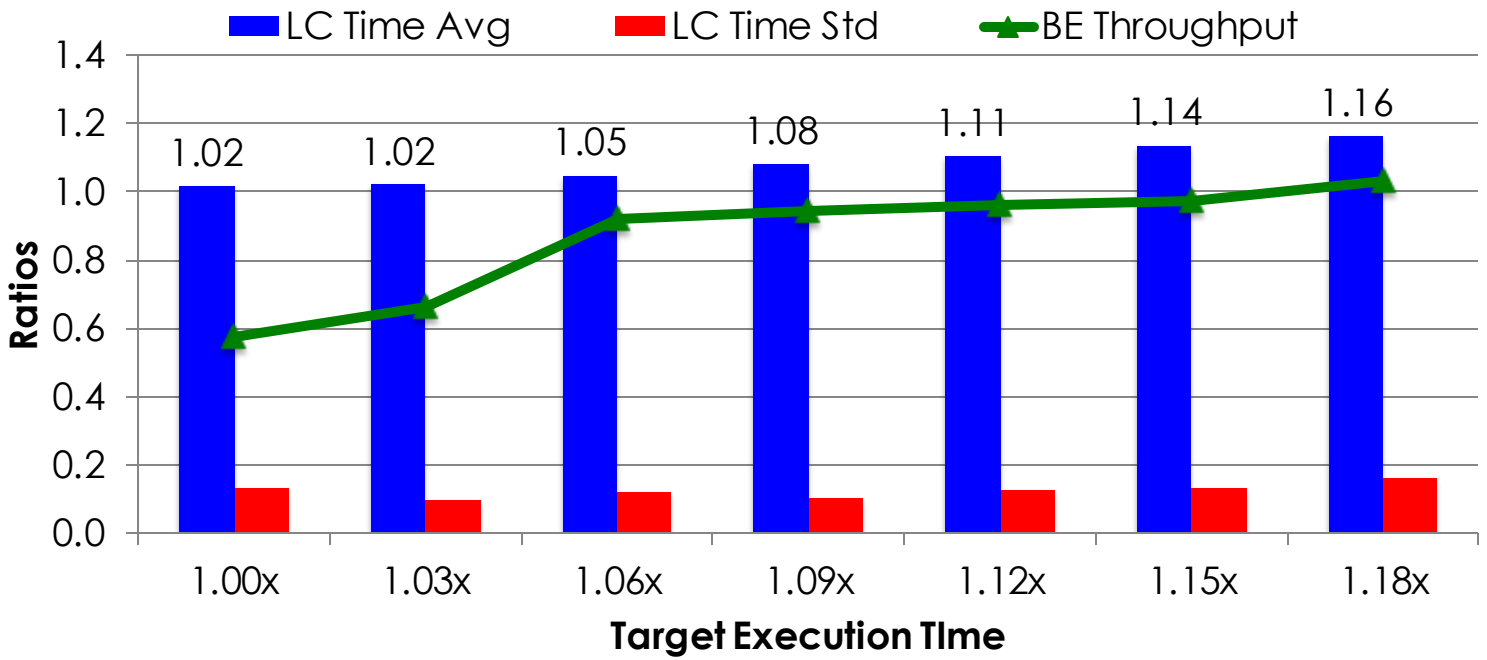
Dirigent results: single LC workloads



Dirigent results: single LC workloads



Tradeoff between LC throughput and BE performance



- Precise control over the range of target deadlines
- Convert LC time slack to BE performance
- Consistent QoS satisfaction rate