Cornell University
Computer Systems Laboratory

PLIANT: LEVERAGING APPROXIMATION TO IMPROVE RESOURCE EFFICIENCY IN DATACENTERS

Neeraj Kulkarni, Feng Qi, Christina Delimitrou

Cloud Computing

- **Resource Flexibility**
  - Users can elastically scale their resources on-demand
- **Cost Efficiency**
  - Sharing resources between multiple users and applications

Latency-critical Interactive apps

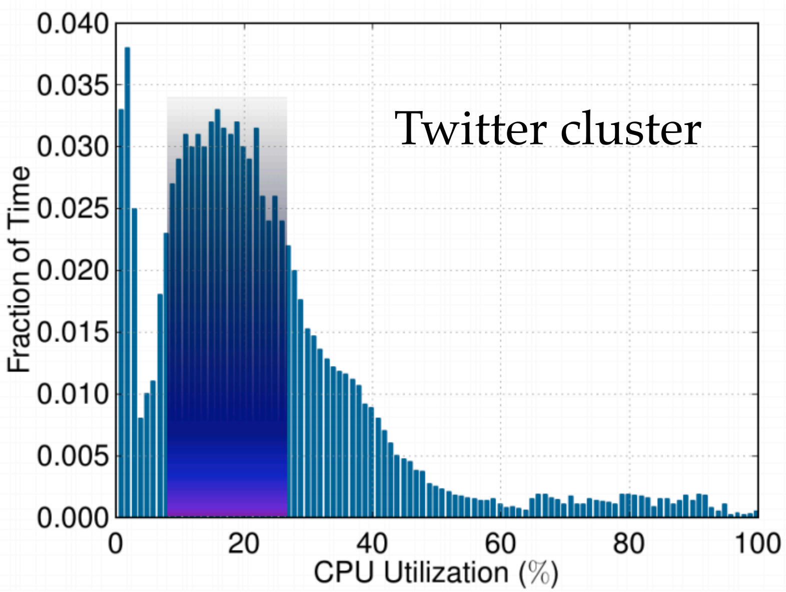
QoS: tail latency

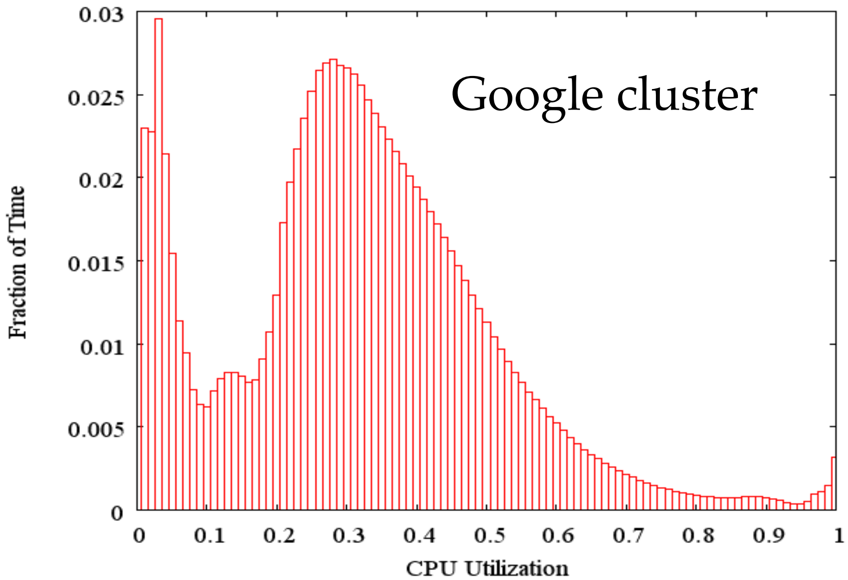
Batch applications

QoS: throughput

Low Utilization!

- **Servers operate at 10% - 40% utilization most of the time**





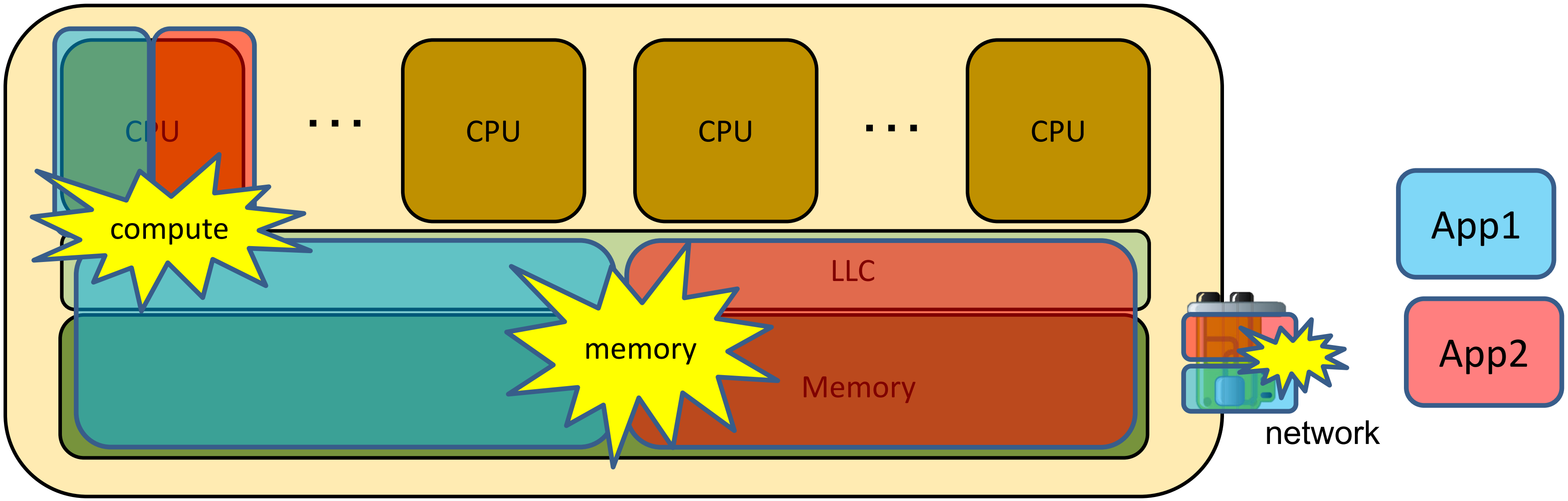
- **Major reasons:**
  - Dedicated servers for interactive services
  - Resource over-provisioning – conservative reservations

C. Delimitrou and C. Kozyrakis, "Quasar: Resource-Efficient and QoS-Aware Cluster Management," in *ASPLOS*, 2014

L. Barroso et. al., "The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines", Second edition, 2013

Multi-tenancy

- **Scheduling multiple jobs on the same server**
  - Increases server utilization and cost efficiency
  - Interference in shared resources



- **Interference → Unpredictable performance**
- **Difficult with interactive services**

Previous Solutions

1. **Allow co-scheduling of apps that would not violate QoS**
   - Bubble-Up, Bubble-Flux, Paragon and Quasar
2. **Partition shared resources at runtime to reduce interference**
   - Heracles, Ubik, Rubik
3. **Reduce interference by throttling applications at runtime**
   - Bubble-Flux, ReQoS, Protean Code

- **But …**
  - Server utilization by disallowing certain co-locations
  - Performance of batch applications by treating them as low-priority

Break utilization vs performance trade-off

- **Approximate computing applications**
  - Tolerate some loss in output accuracy in return for
    - » Improved performance, or
    - » Same performance with reduced resources

- **Cloud workloads suitable for approximation**

  - Performance can be more important than highest output quality

- **Co-locate approximate batch apps with interactive services**
  - Meet performance for both applications at the cost of some inaccuracy

Leveraging Approximation

1. Mitigate interference:

- Approximation can reduce # of requests to memory system & network
- Approximation may not be always sufficient

2. Meet performance of approximate applications:

- When approximation is not enough, employ resource partitioning:
  - » Core relocation
  - » Cache partitioning
  - » Memory partitioning
- Provide more resources to interactive service to meet its QoS
- Approximation preserves the performance of batch applications

Approximation Techniques

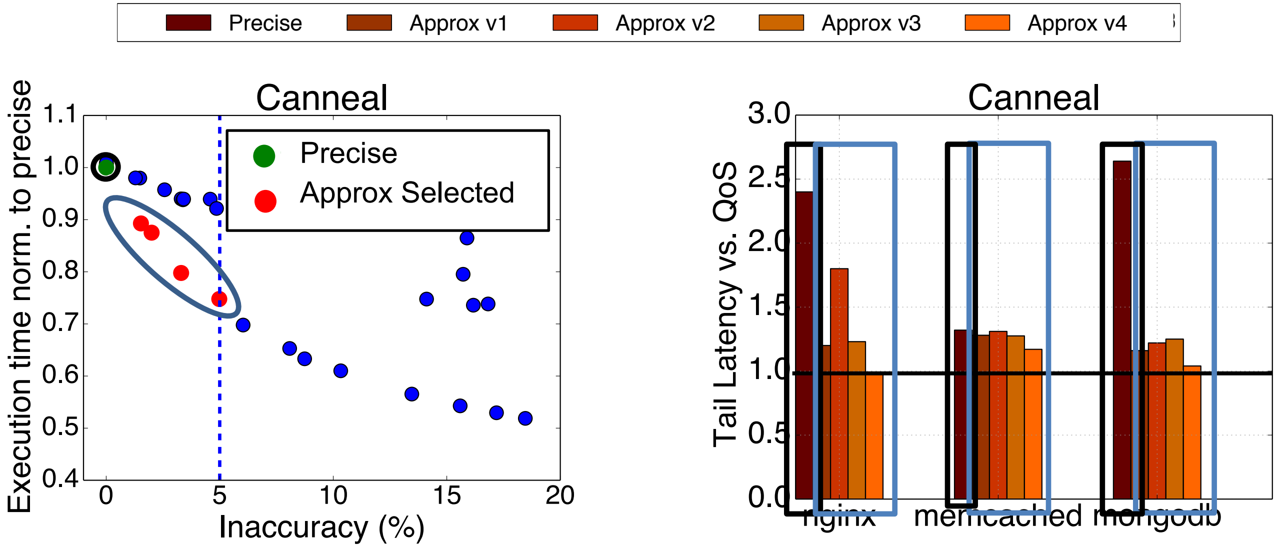
- **Loop perforation:** Skip fraction of iterations
  - Fewer instructions & data accesses → exec time ⇩ & cache interference ⇩
- **Synchronization elision:** Barriers, locks elided
  - Threads don't wait for sync → exec time ⇩
  - Reduces memory accesses for acquiring locks
- **Lower precision:** Reduce precision of variables
  - e.g., replace 'double' with 'float' or 'int'
  - Reduces memory traffic
- **Tiling:** Compute 1 element & project onto neighbors
  - Fewer instructions & data accesses → exec time ⇩ & cache interference ⇩

```
            :
double l_c = 1.0
float l_c = 1.0

for i = 1 to N:
  if i % 2 != 0:
     .....
            :
lock()
  g_c = g_c + l_c
unlock()
            :
BARRIER()
            :
for i = 1 to M:
  A[i,2] = F(i,2)
  for j = 1 to 3:
    A[i][j] = F(i,j)
    A[i][j] = A[i][2]
```

Approximation Trade-offs

- **100s of approximate variants**
- **Pruning design space:**
  - Hint-based:
    - » Employ approximations hinted by ACCEPT* tool
  - Profiling-based (gprof):
    - » Approximate in functions which contribute most to execution time



*ACCEPT: A Programmer-Guided Compiler Framework for Practical Approximate Computing, A. Sampson et. al.

PLIANT: GOALS

- **High utilization**
  - Co-schedule interactive services with approximate applications

- **High QoS**
  - Satisfy QoS of all co-scheduled jobs at the cost of some accuracy loss

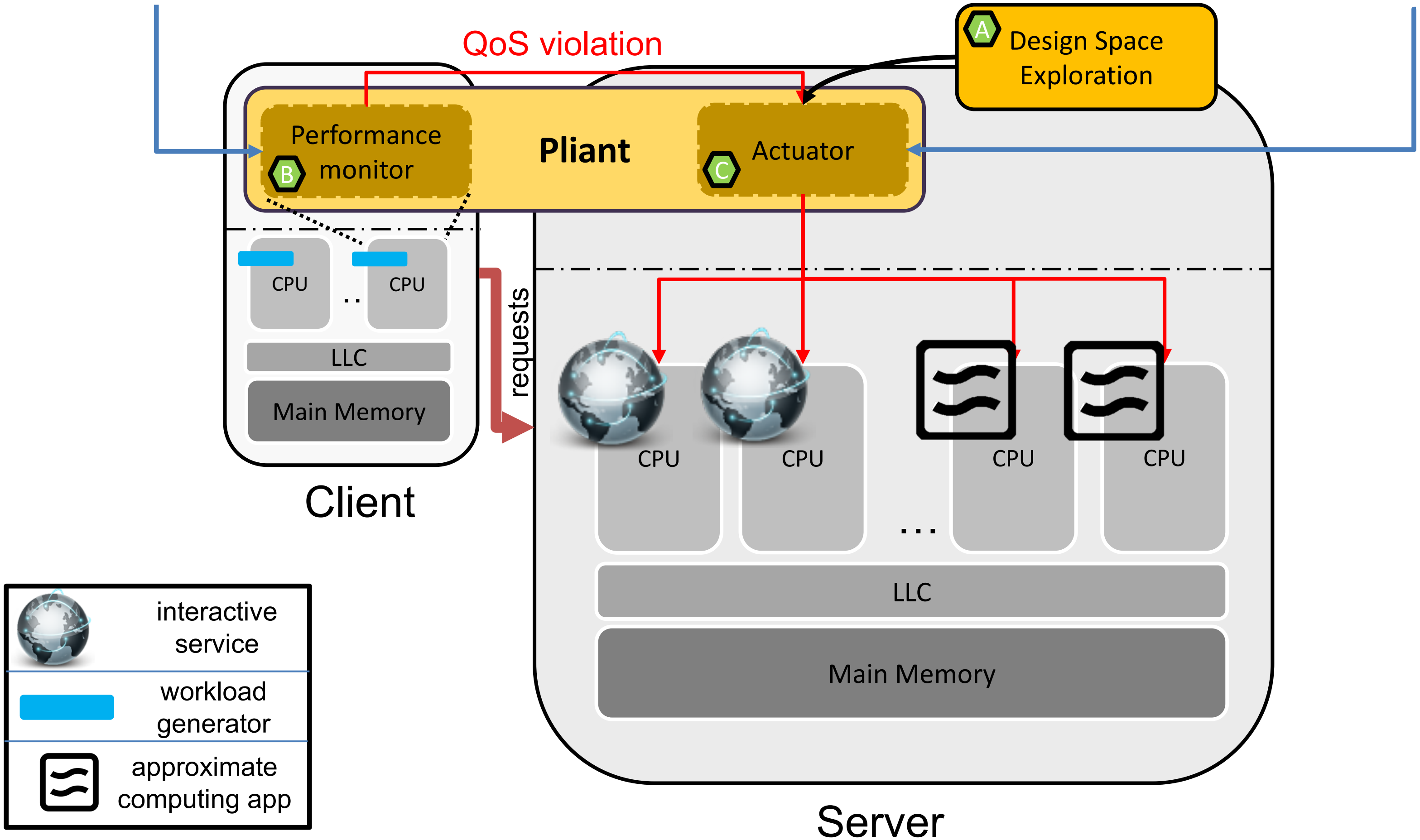
- **Minimize accuracy loss**
  - Adjust approximation at runtime using slack in tail latency

- **Techniques used to reduce interference at runtime**
  - Approximation
  - Resource relocation (core relocation, cache & memory partitioning)

PLIANT - OVERVIEW

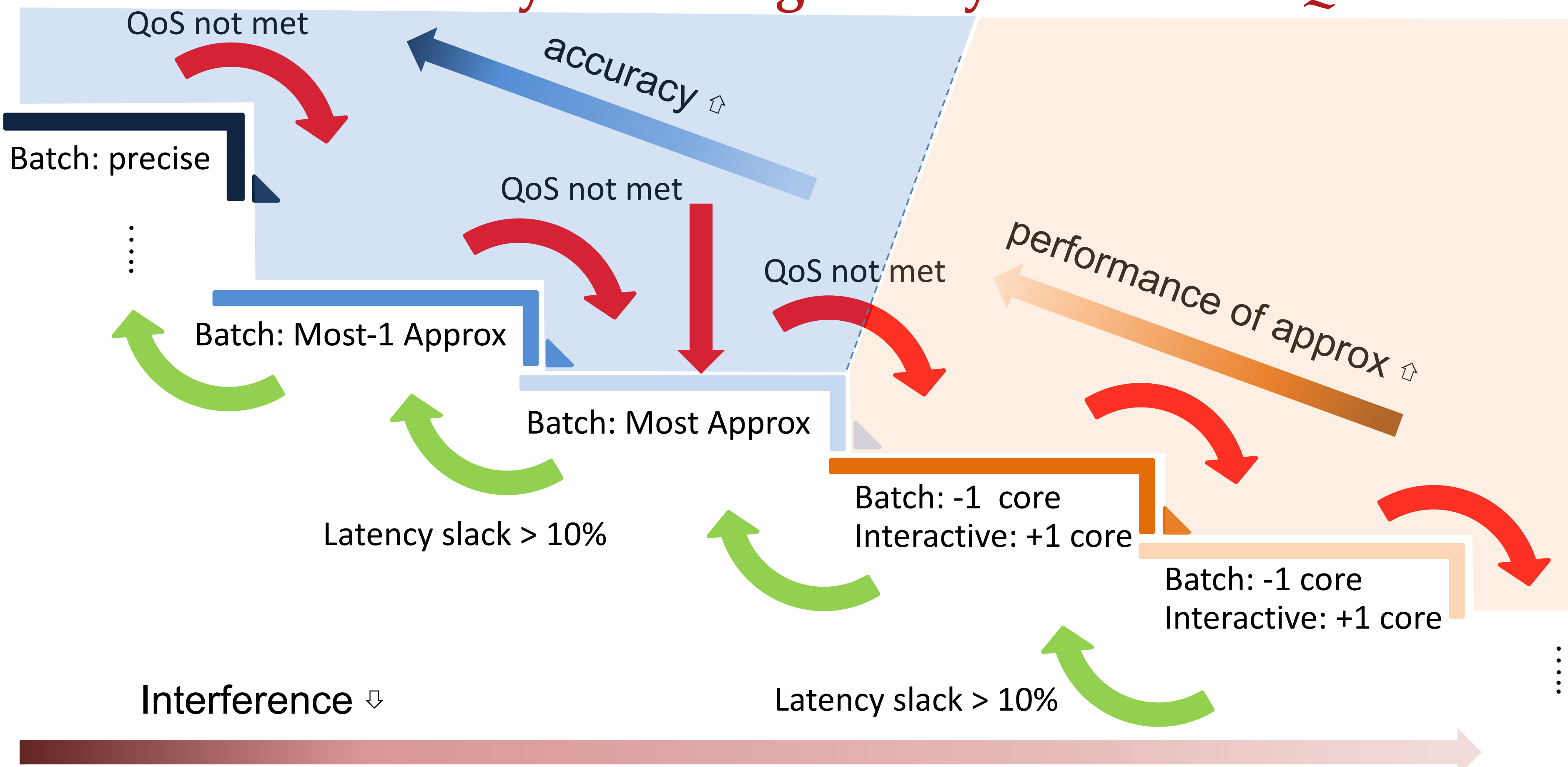
- Continuously monitors the tail latency

- Dynamic recompilation
- Runtime allocation



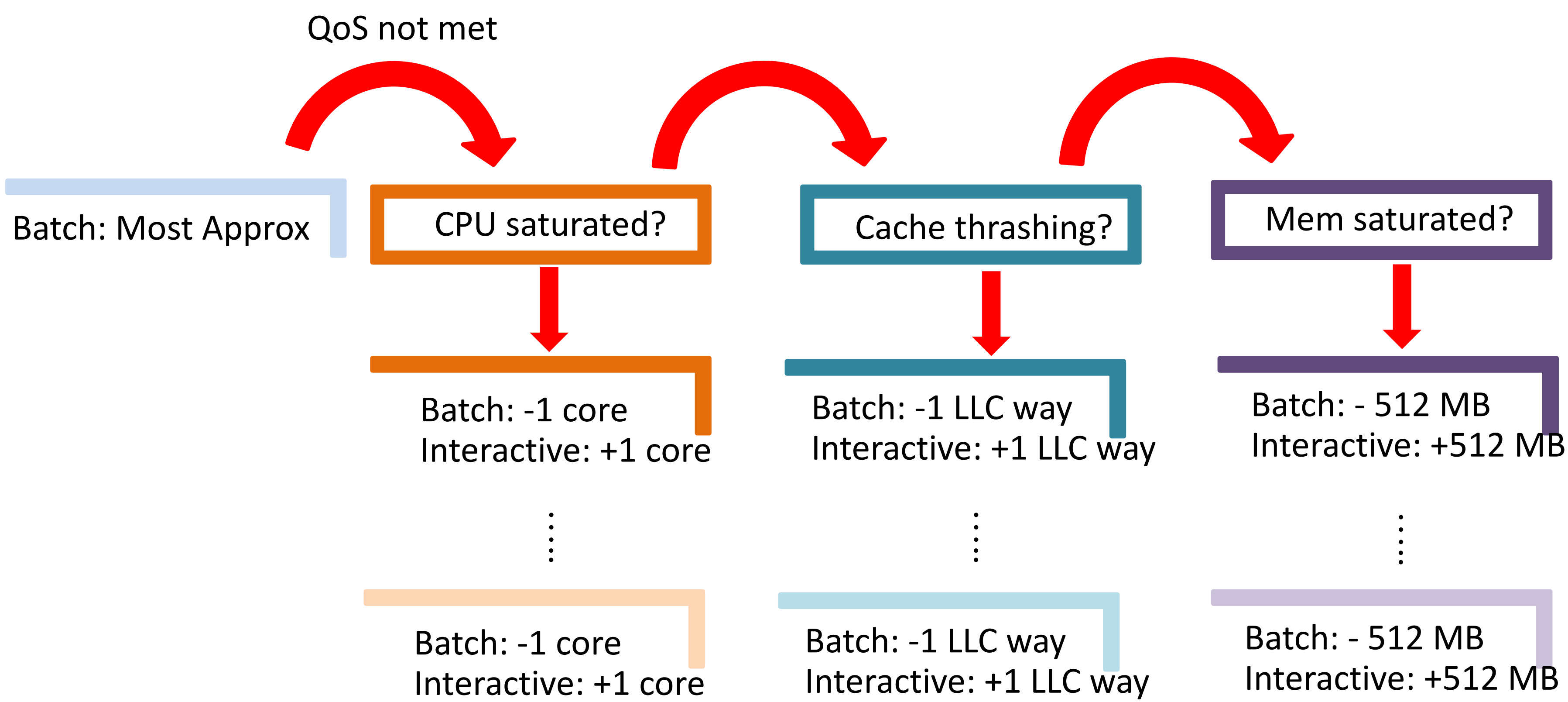
PLIANT – RUNTIME ALGORITHM

- **Meet QoS as fast as possible**
- **Minimize accuracy loss using latency slack when QoS met**



PLIANT – RUNTIME ALGORITHM

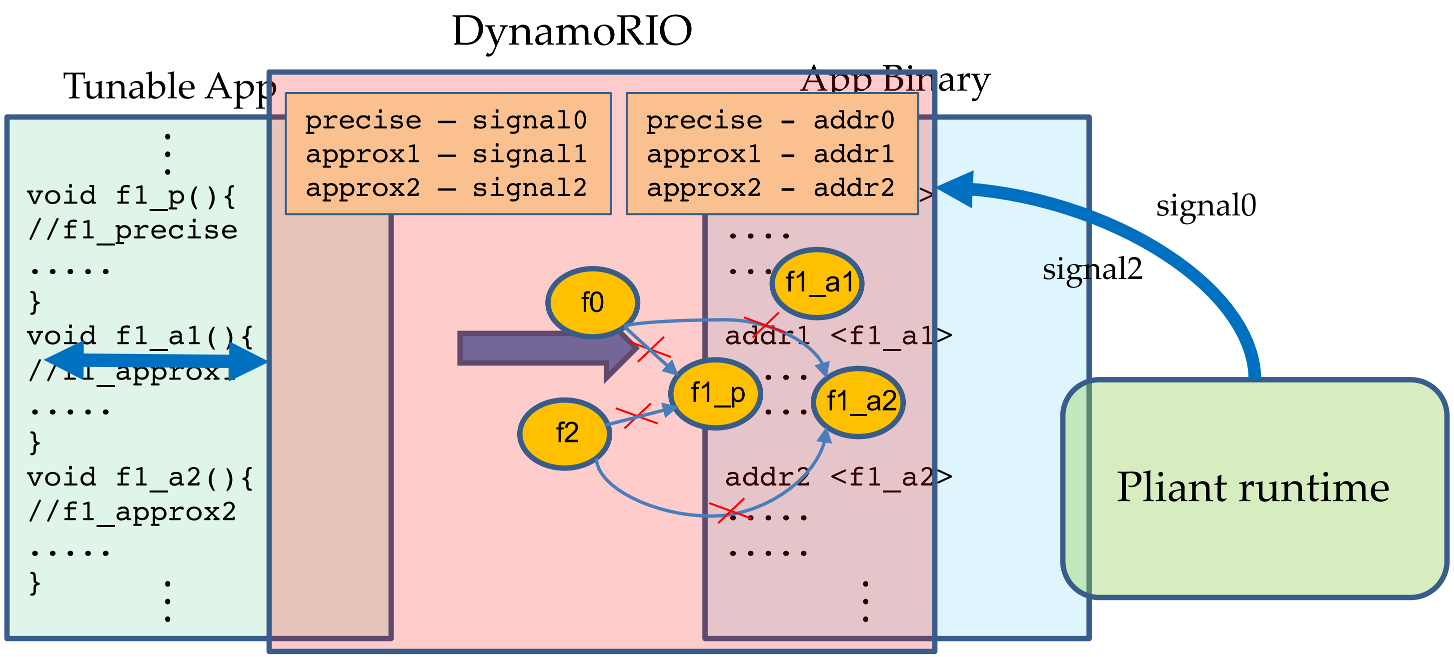
- Multiple resources: cores, LLC and memory



PLIANT – VARYING APPROXIMATION DEGREE

CSL

- **Dynamic recompilation system**
  - Aggregated approximate variants to construct tunable app
  - Linux signals for DynamoRIO to switch to an approximate variant
  - drwrap_replace() interface is used to replace functions
    - » Coarse granularity → low overheads



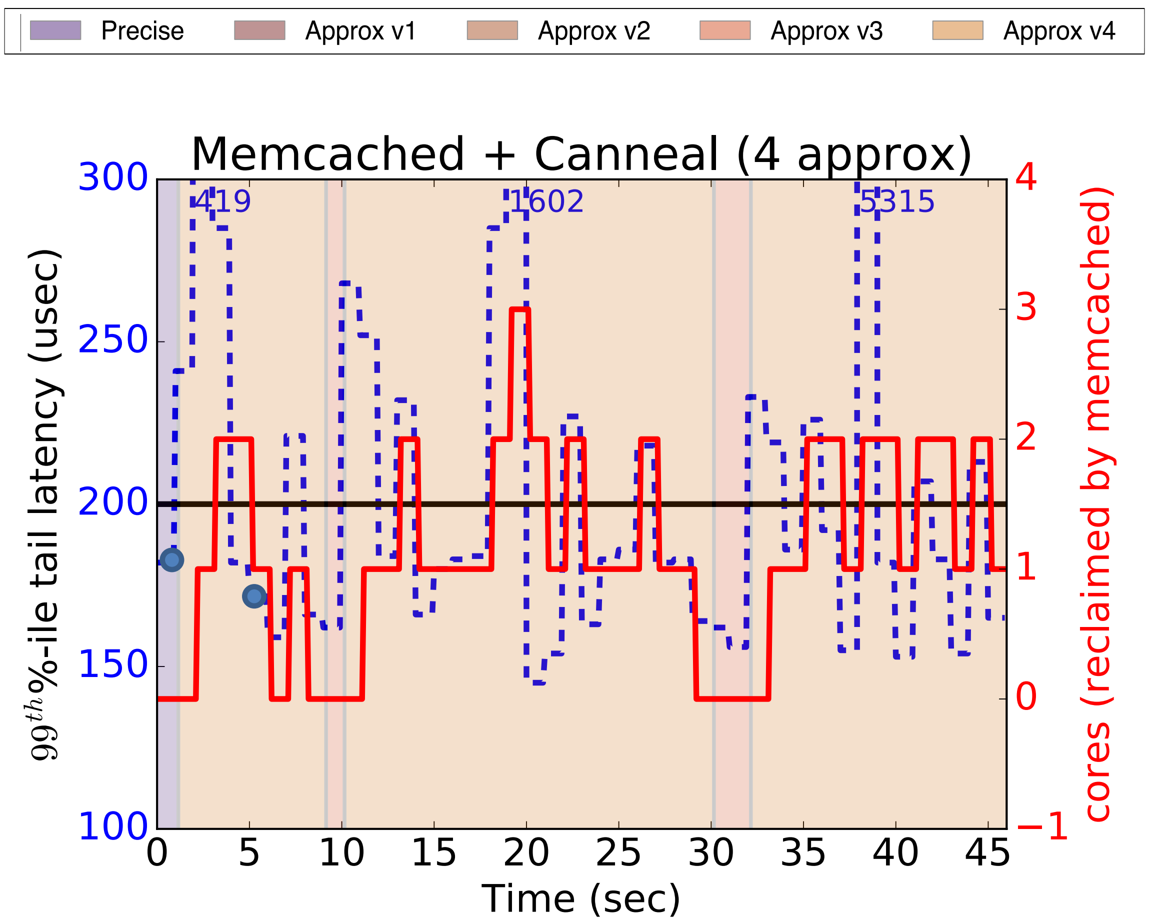
PLIANT – RUNTIME RESOURCE ALLOCATION

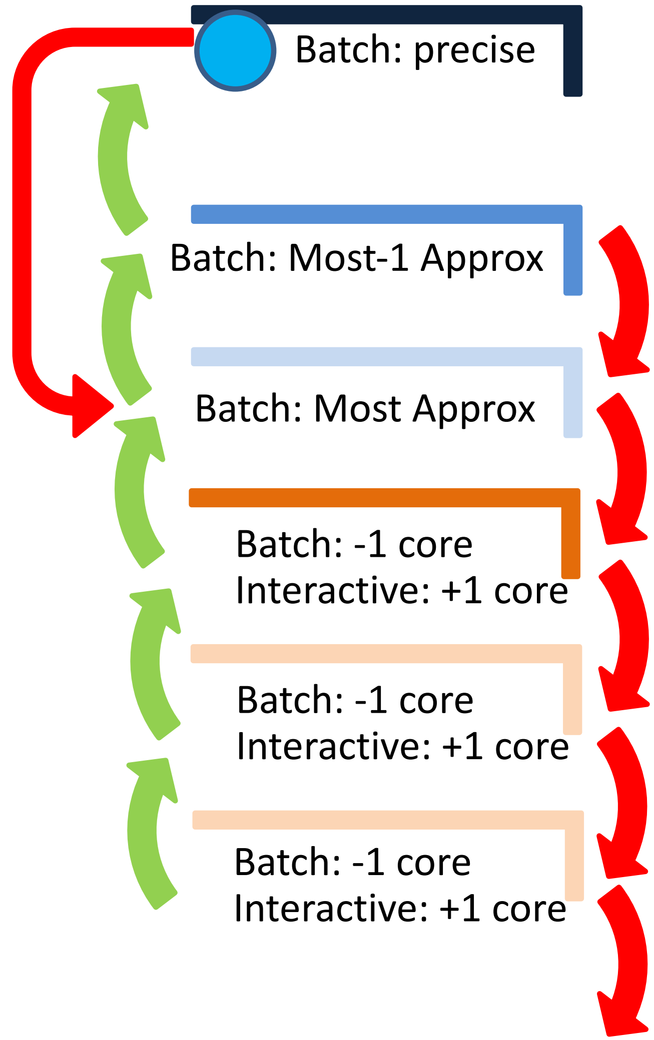
- **All applications run in Docker containers**
- **Core relocation**
  - Docker update interface to allocate cores to each container
- **Cache allocation**
  - Intel's Cache Allocation Technology (CAT) to allocate cache ways
- **Memory capacity**
  - Docker update interface to assign memory limits

Experimental Setup

- **Interactive services:** NGINX, memcached, MongoDB
- **24 approximate computing applications:**
  - PARSEC, SPLASH2x, MineBench, BioPerf benchmark suites
- **Systems**
  - 44 physical core dual-socket platform, 128 GB RAM, 56 MB LLC/socket
  - Interactive services & approximate applications pinned to different physical cores of same socket
- **Baseline**
  - Approximate application run in precise mode
  - Cores, cache, and memory shared fairly among the applications

Evaluation - Dynamic behavior

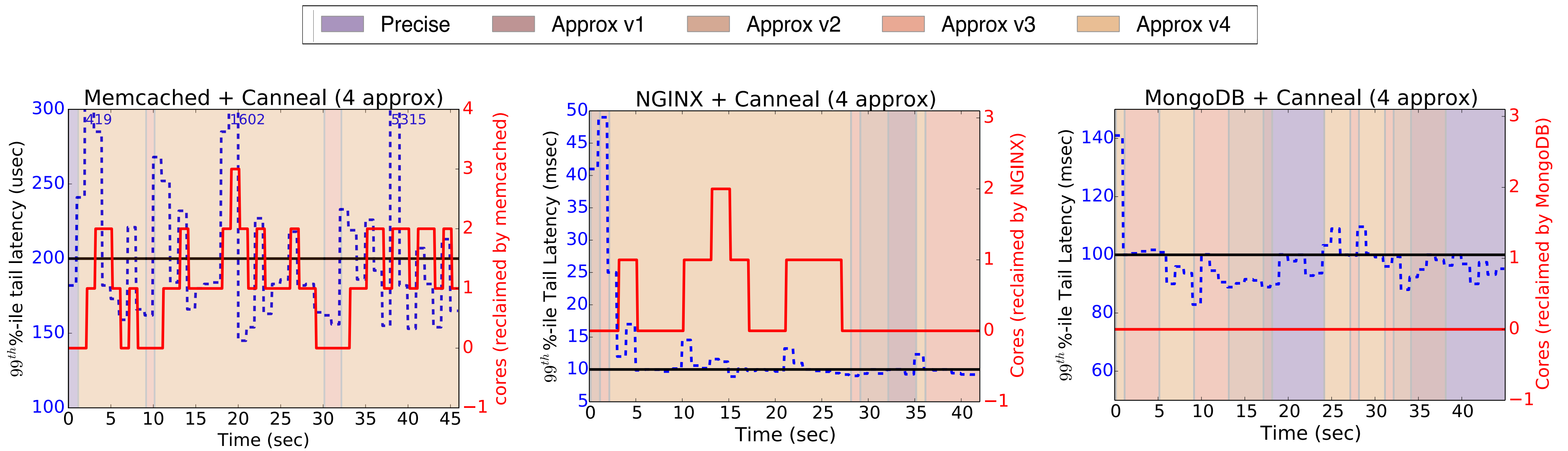




Evaluation – Dynamic Behavior

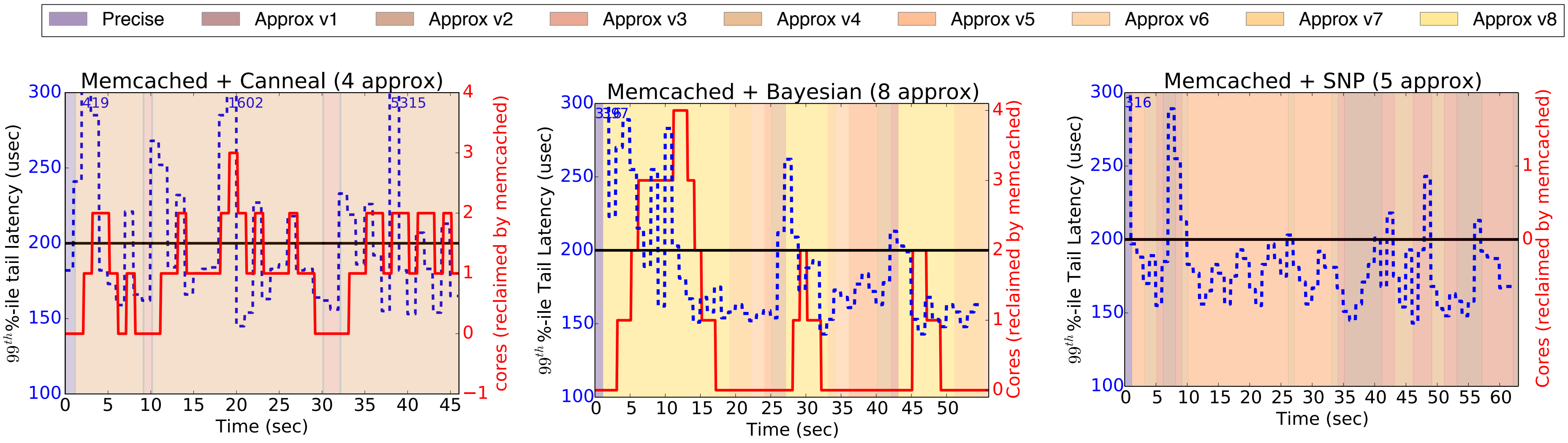
CSL

- **Across interactive services**
  - memcached and NGINX need to reclaim resources
  - In case of MongoDB, approximation is enough



Evaluation – Dynamic Behavior

- **Across approximate applications**
  - Bayesian shows bursty behavior - approximation usually enough
  - In case of SNP, no resource reclamation is required



- **For all co-schedulings, show QoS is met for all apps at an accuracy loss of up to 5% (2.8% on average)**

Summary - Pliant

- **Approximation can break performance vs utilization trade-off**
- **Many cloud applications can tolerate some loss of quality**

- **Pliant – practical runtime system**
  - Incremental approximation using dynamic recompilation
  - Dynamic allocation of shared resources

- **Achieves high utilization**
  - Enabled co-scheduling of approximate batch apps with interactive services

- **Achieves high QoS**
  - Meets QoS for all apps at cost of small accuracy loss (max 5%, avg 2.8%)

QUESTIONS?

CSL

- **Approximation can break performance vs utilization trade-off**
- **Many cloud applications can tolerate some loss of quality**

- **Pliant – practical runtime system**
  - Incremental approximation using dynamic recompilation
  - Dynamic allocation of shared resources

- **Achieves high utilization**
  - Enabled co-scheduling of approximate batch apps with interactive services

- **Achieves high QoS**
  - Meets QoS for all apps at cost of small accuracy loss (max 5%, avg 2.8%)

THANK YOU!