Bubble-Flux: Precise Online QoS Management for Increased Utilization in Warehouse Scale Computers

Hailong Yang, Alex Breslow
Jason Mars and Lingjia Tang

CLARity (Cross-layer Architecture and Runtime) Lab

University of California, San Diego

Warehouse Scale Computers

- Host large-scale Internet services

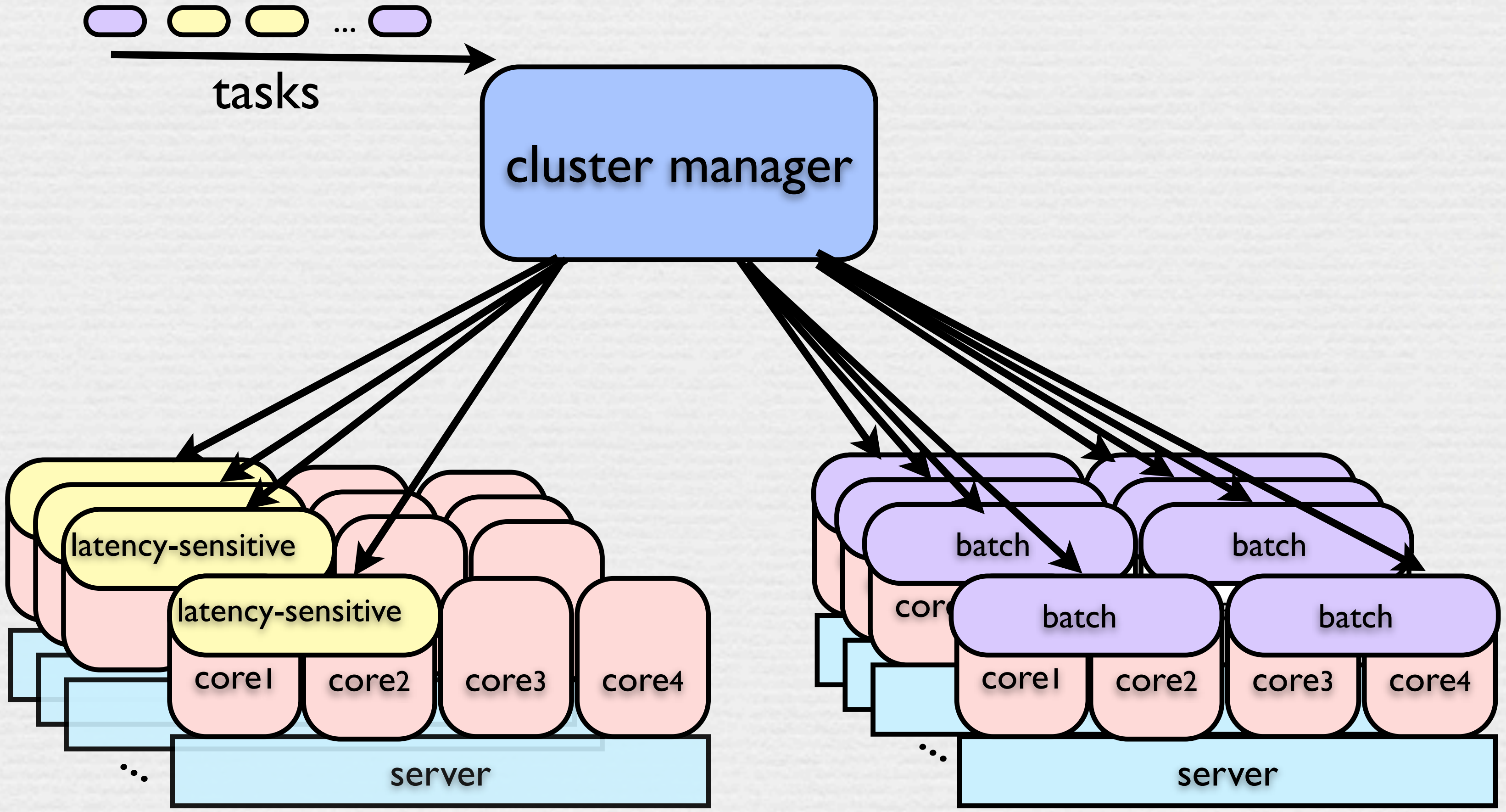
"Datacenters have become as vital to the functioning of society as power stations"
- *The Economist*

Over-provisioning Leads to Low Utilization

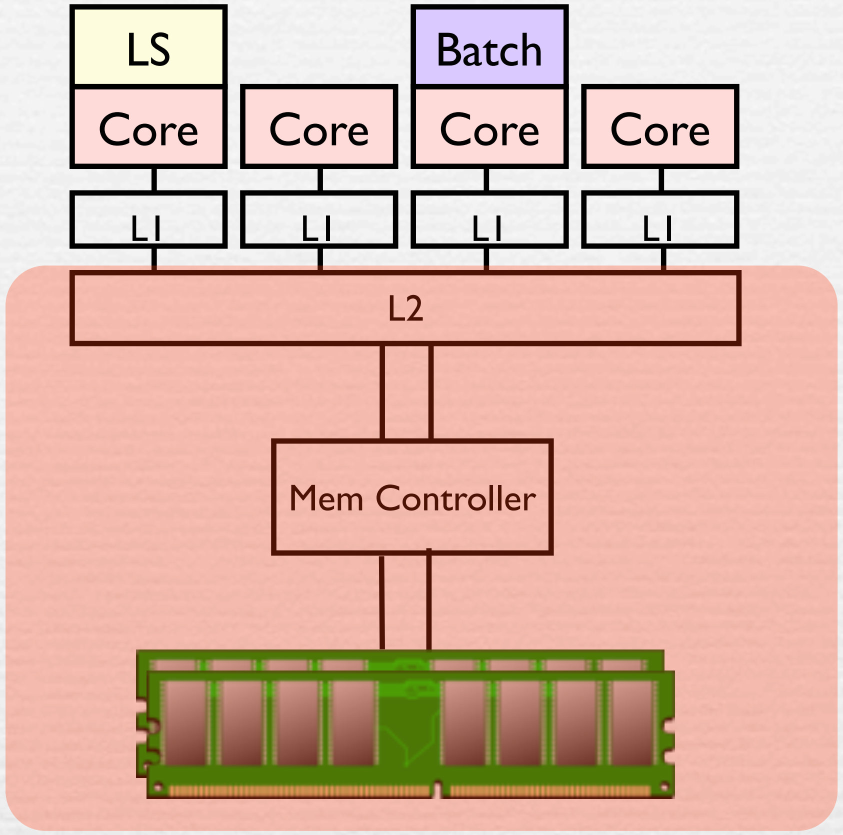
- Status quo
  - Over-provisioning to ensure quality of service for latency-sensitive applications
  - Low machine utilization

Over-provisioning Leads to Low Utilization

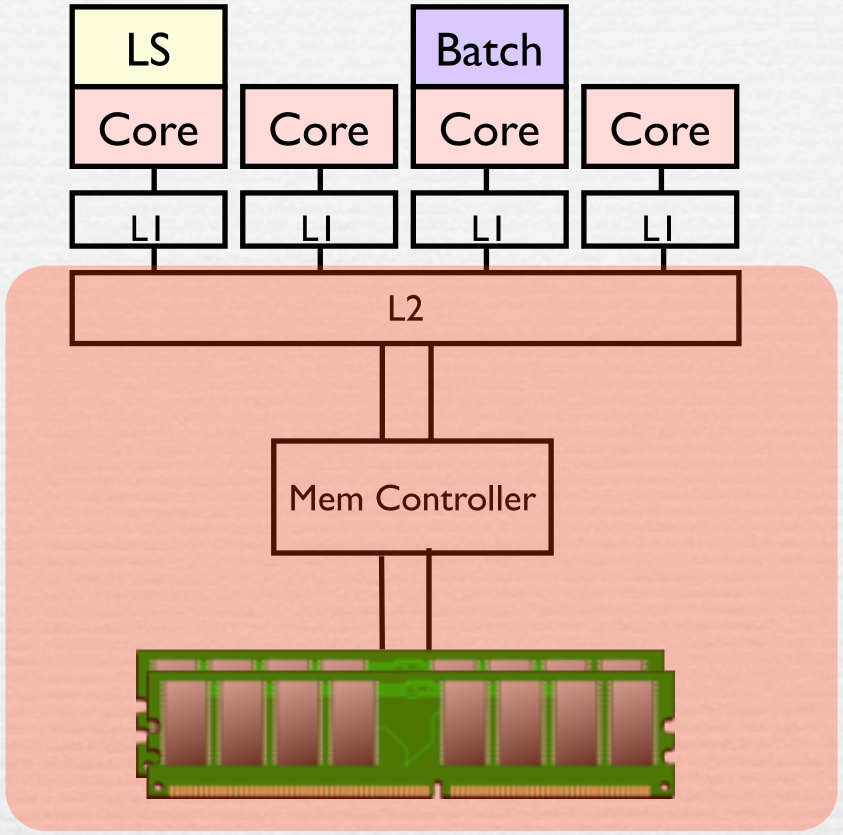
- Status quo
  - Over-provisioning to ensure quality of service for latency-sensitive applications
  - Low machine utilization



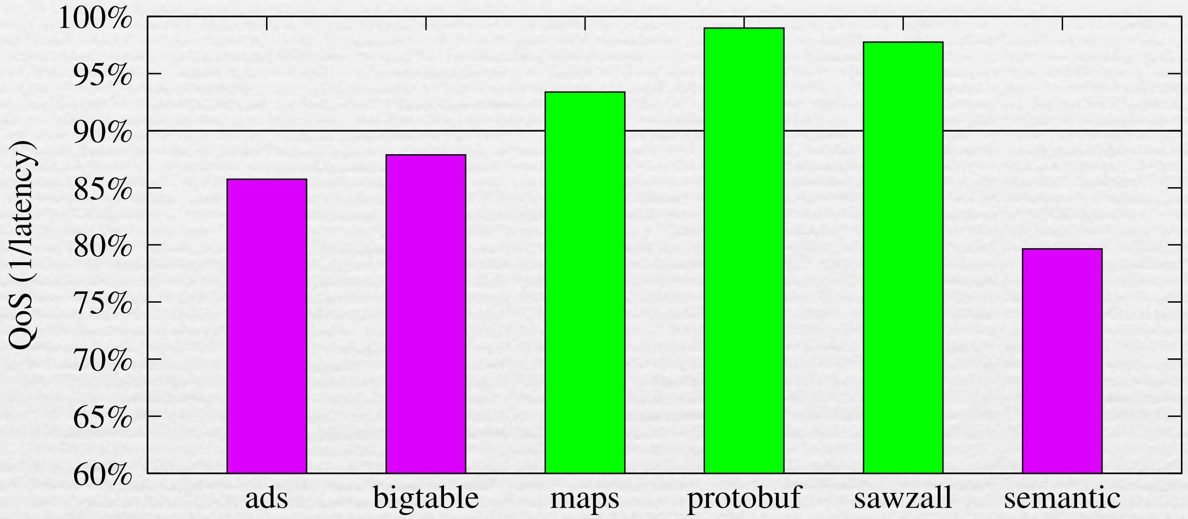
Why Over-provisioning?



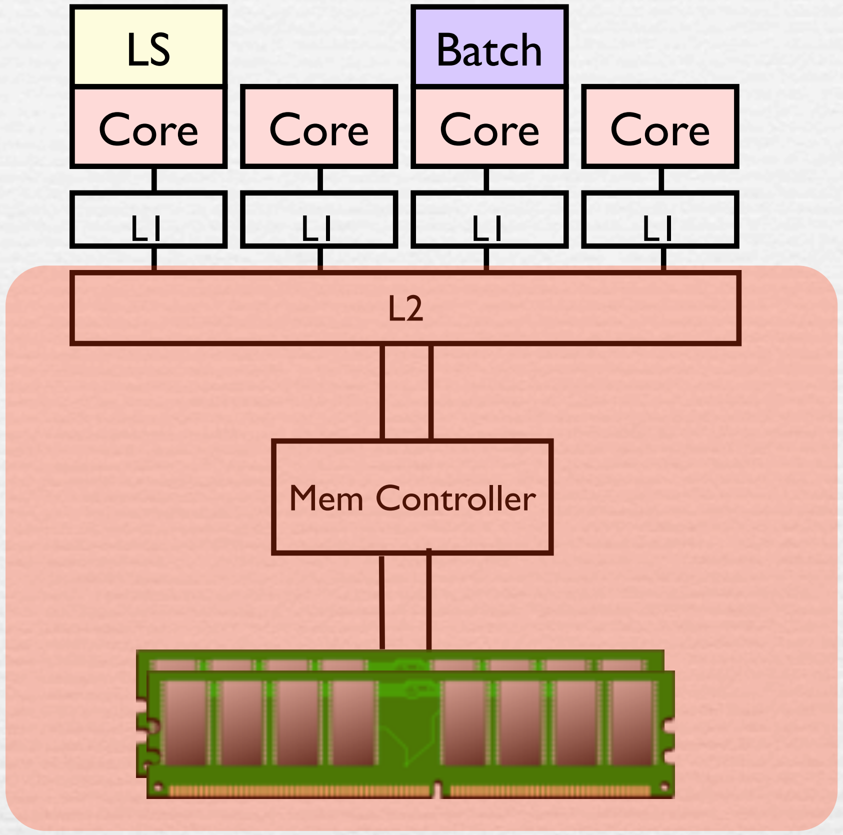
Why Over-provisioning?



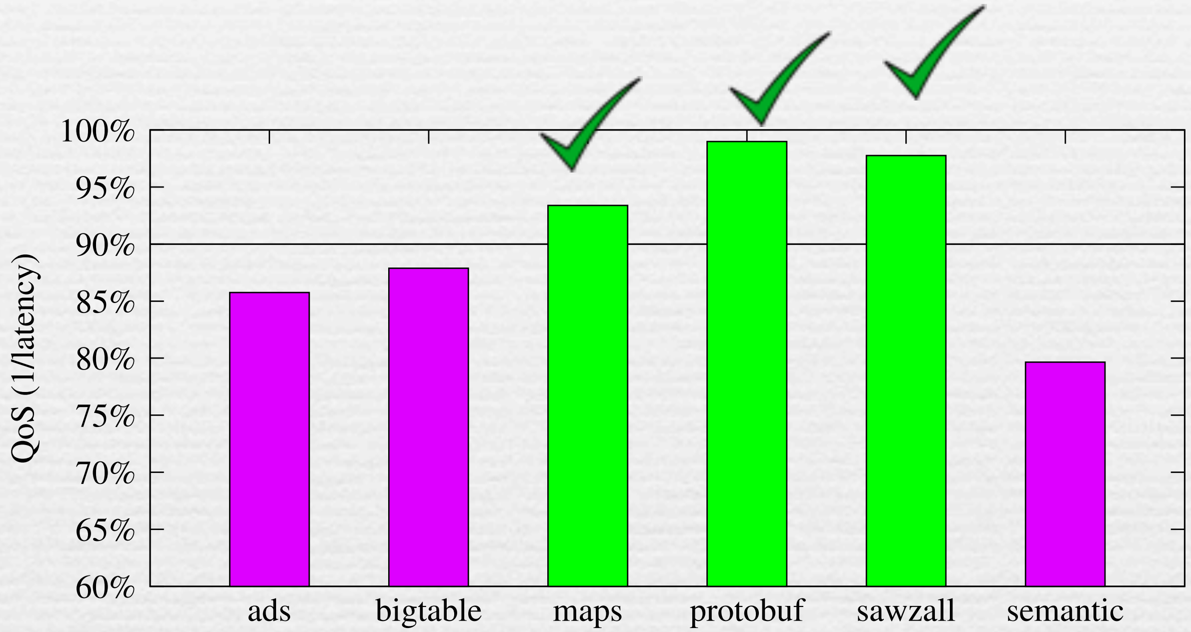
Performance of Search Render as Co-Runner Changes



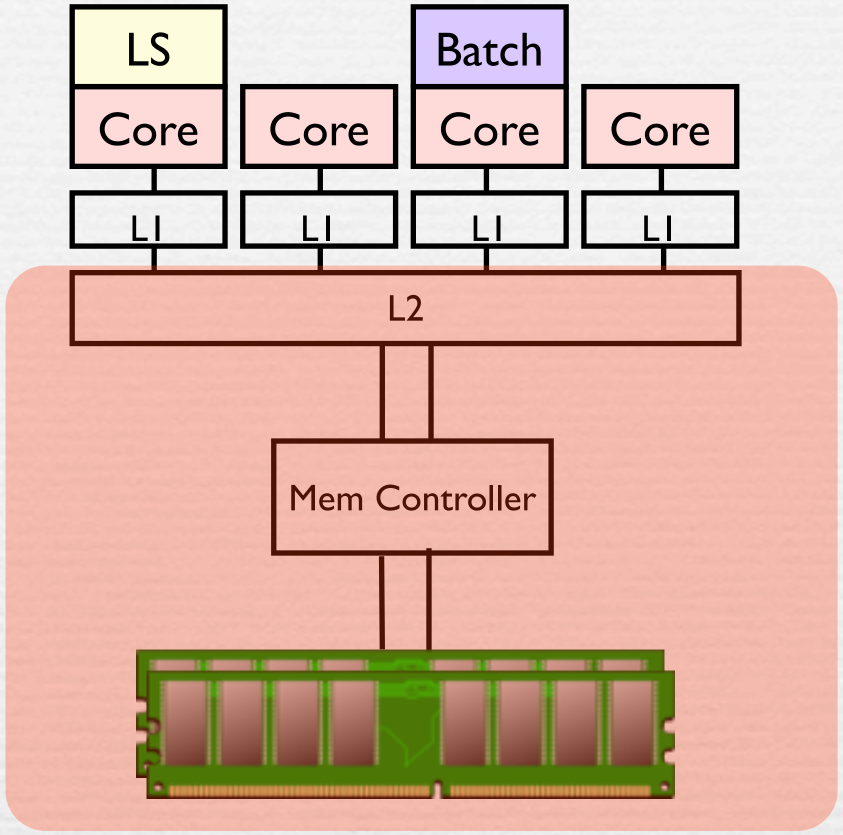
Why Over-provisioning?



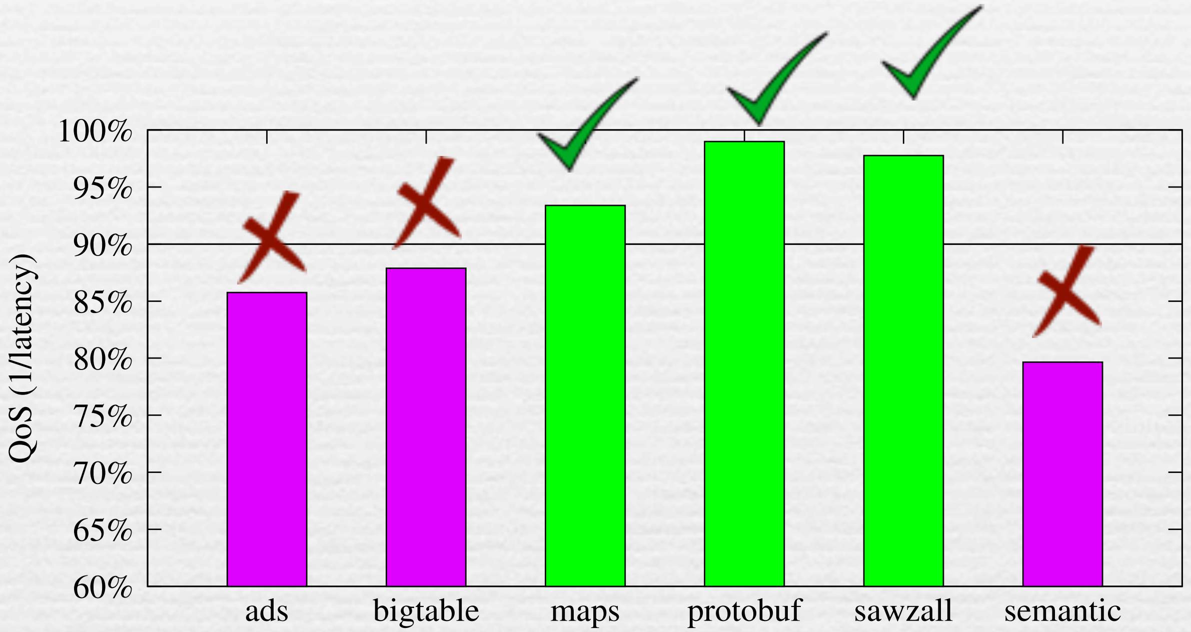
Performance of Search Render as Co-Runner Changes



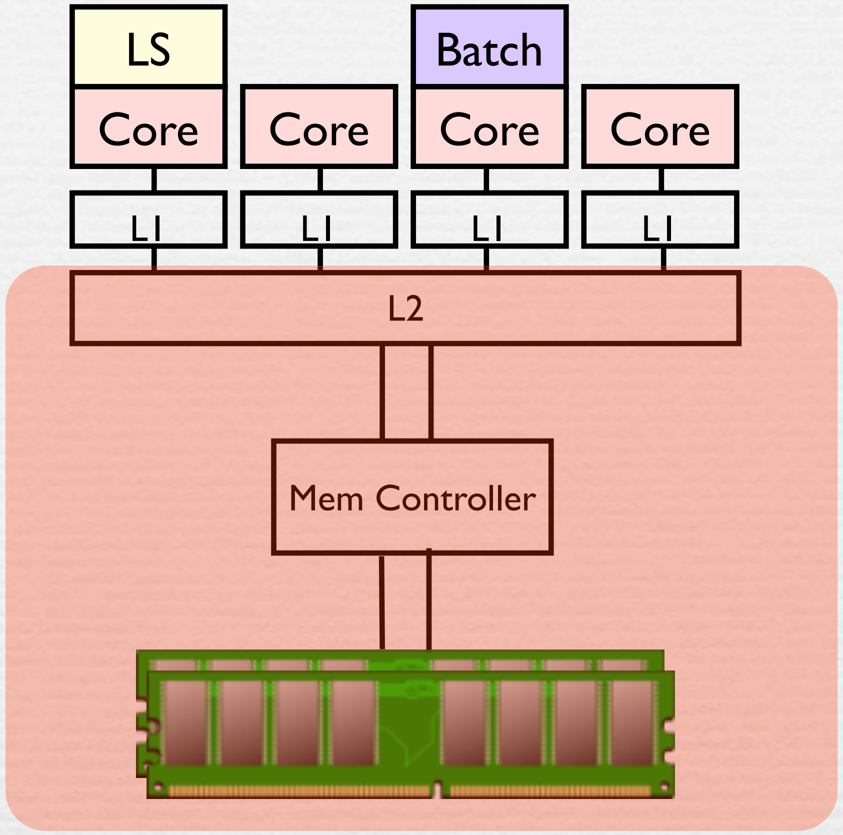
Why Over-provisioning?



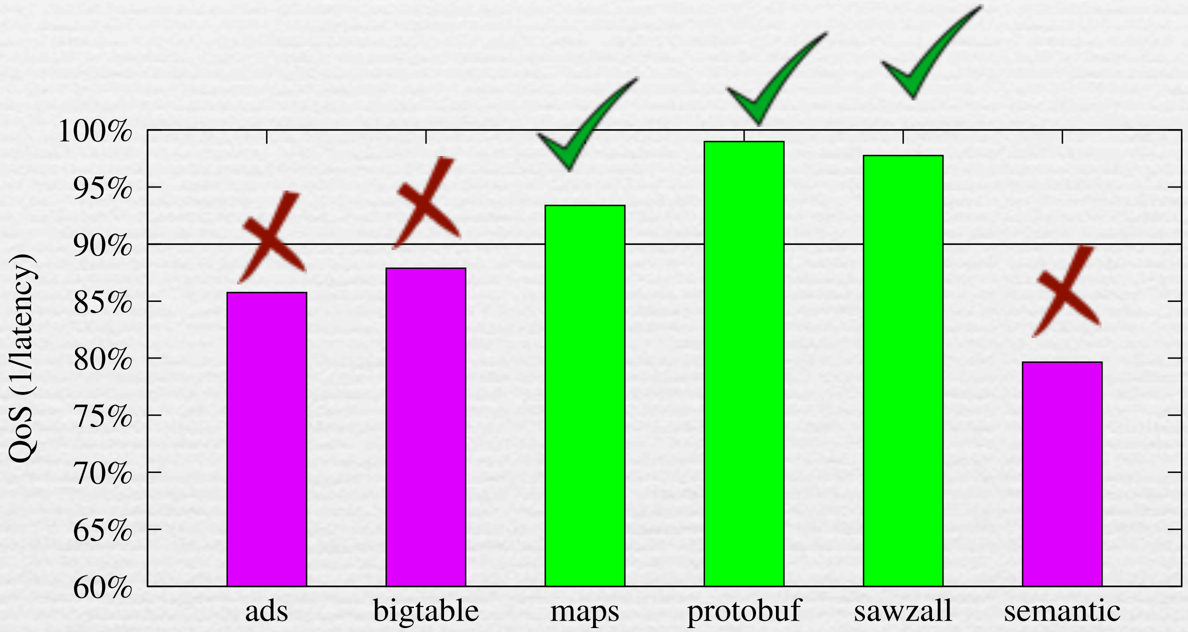
Performance of Search Render as Co-Runner Changes



Why Over-provisioning?

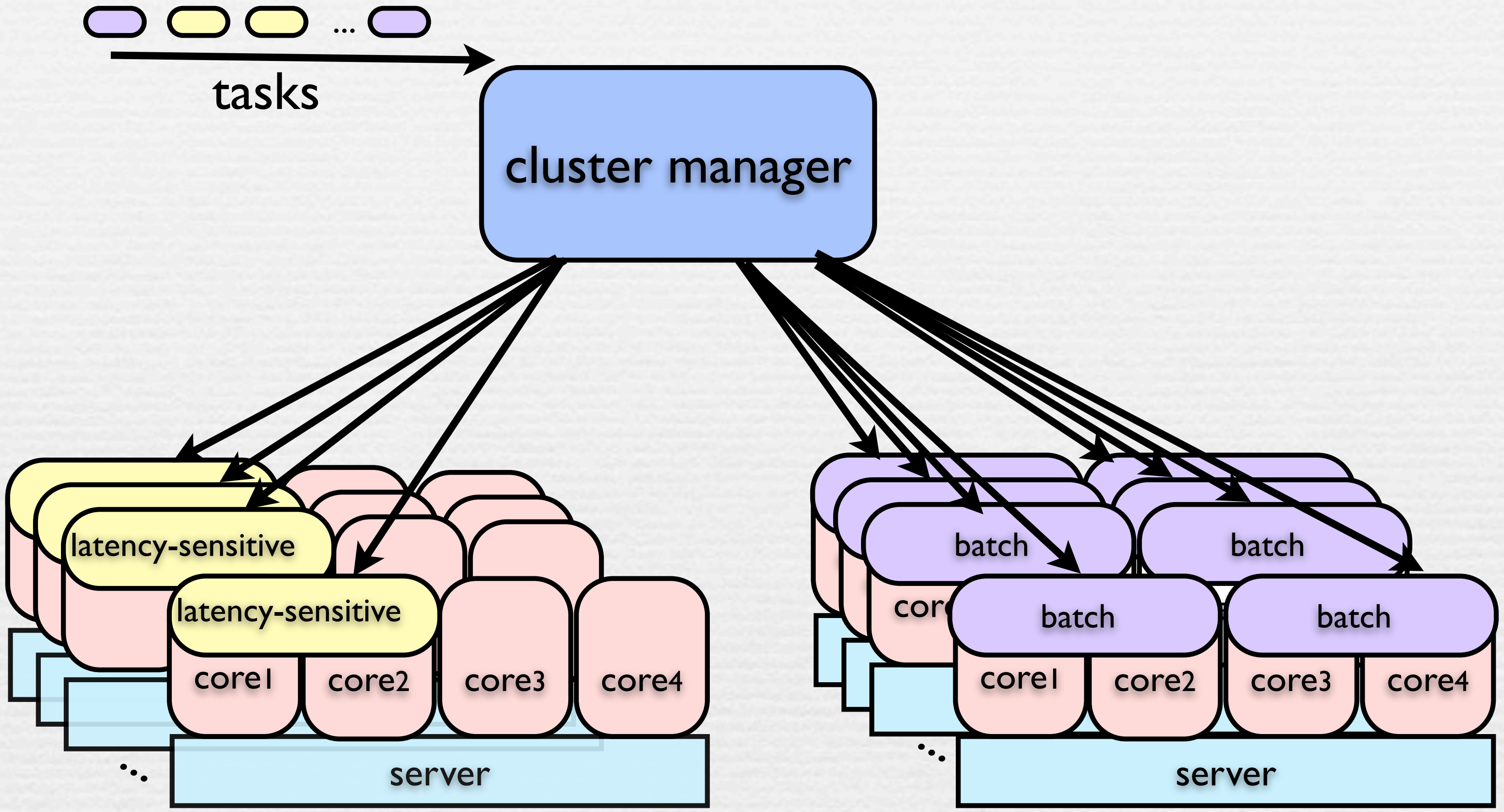


Performance of Search Render as Co-Runner Changes



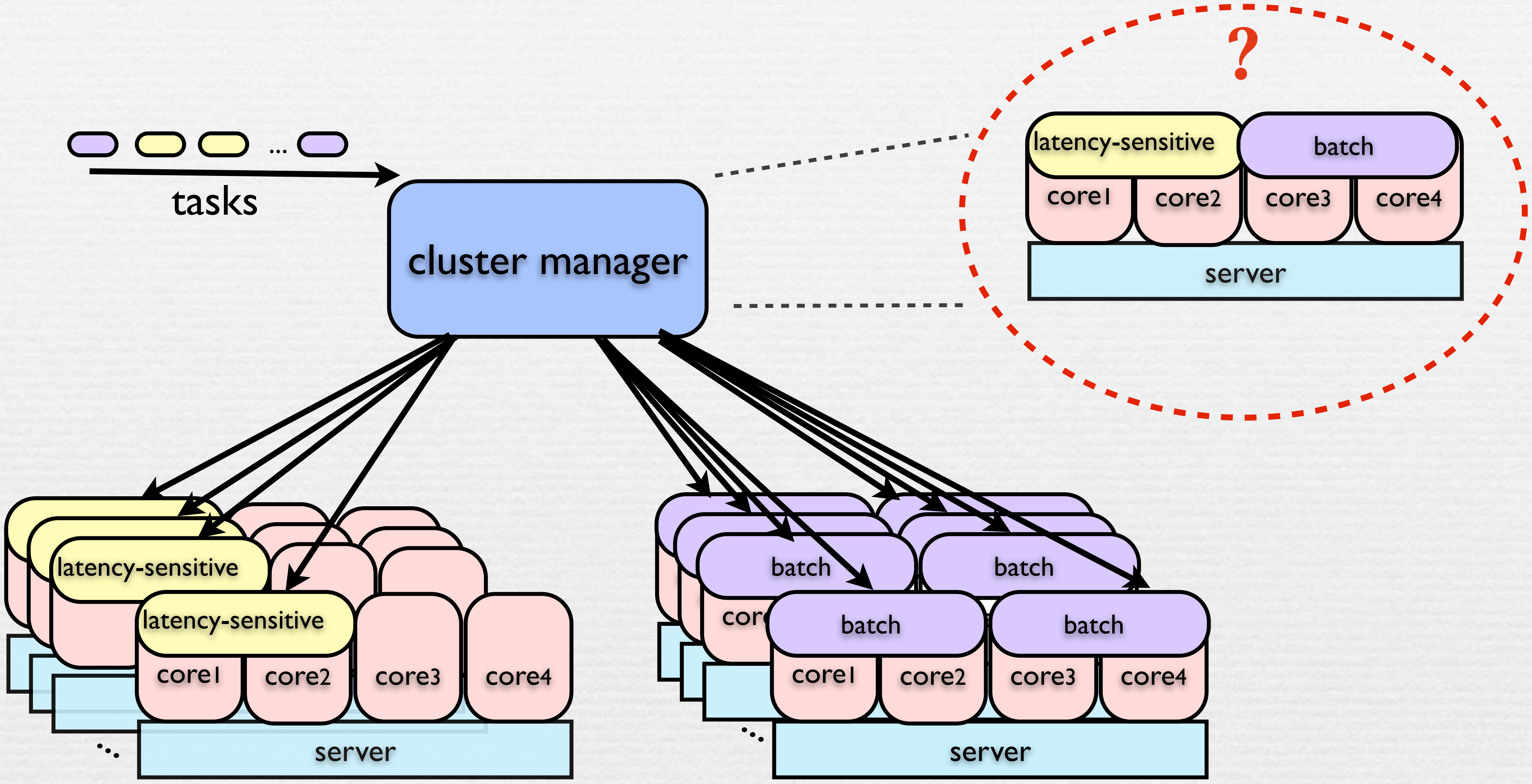
- Uncertain QoS interference leads to over-provisioning and ultimately, expensive, low utilization

Goal



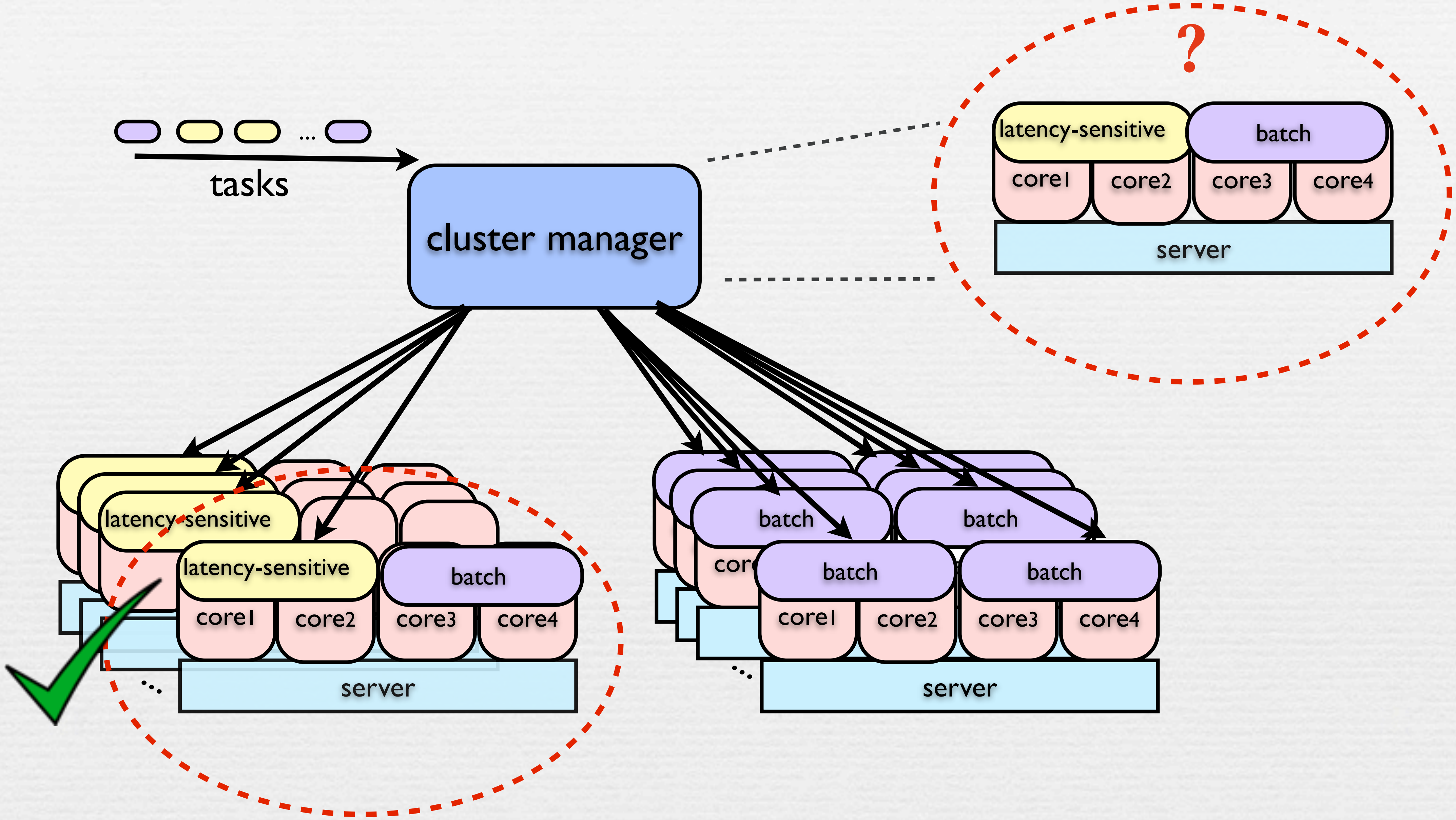
- Predict and manage interference to facilitate “safe” colocation to increase utilization without QoS degradation

Goal



- Predict and manage interference to facilitate "safe" colocation to increase utilization without QoS degradation

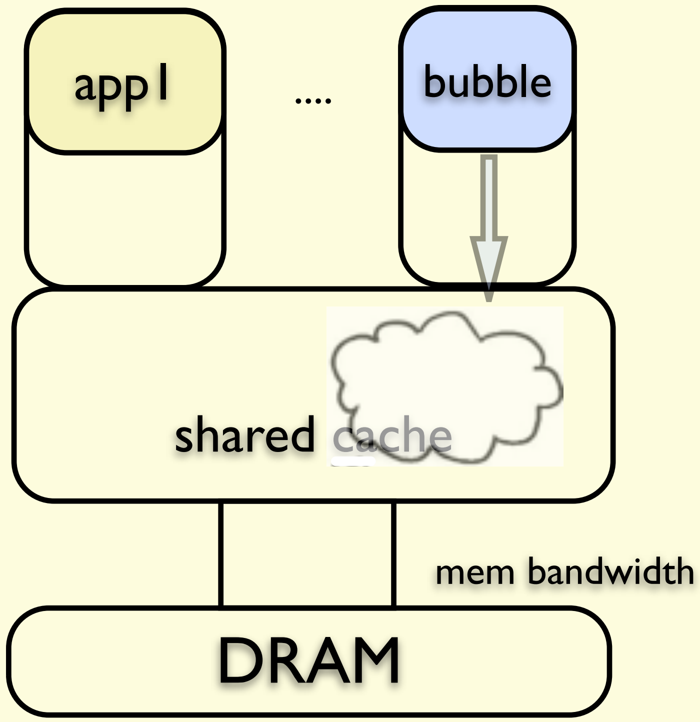
Goal

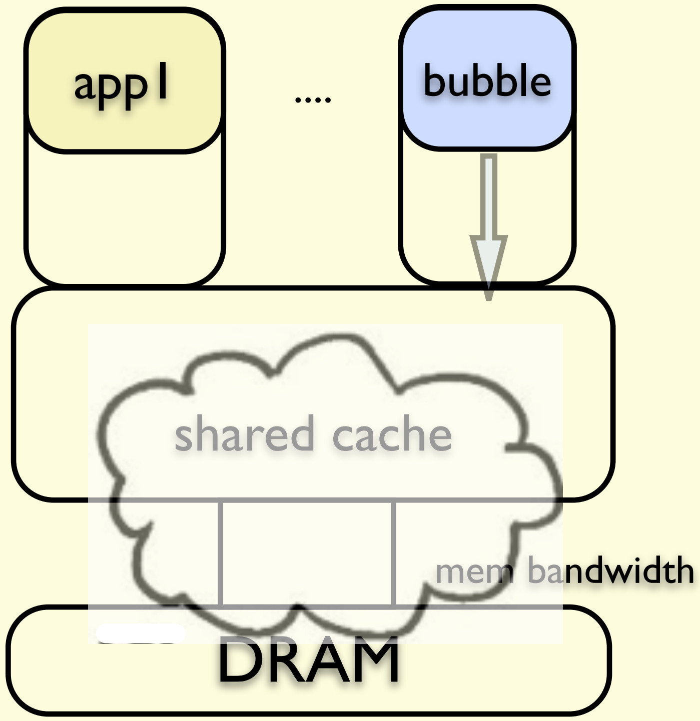


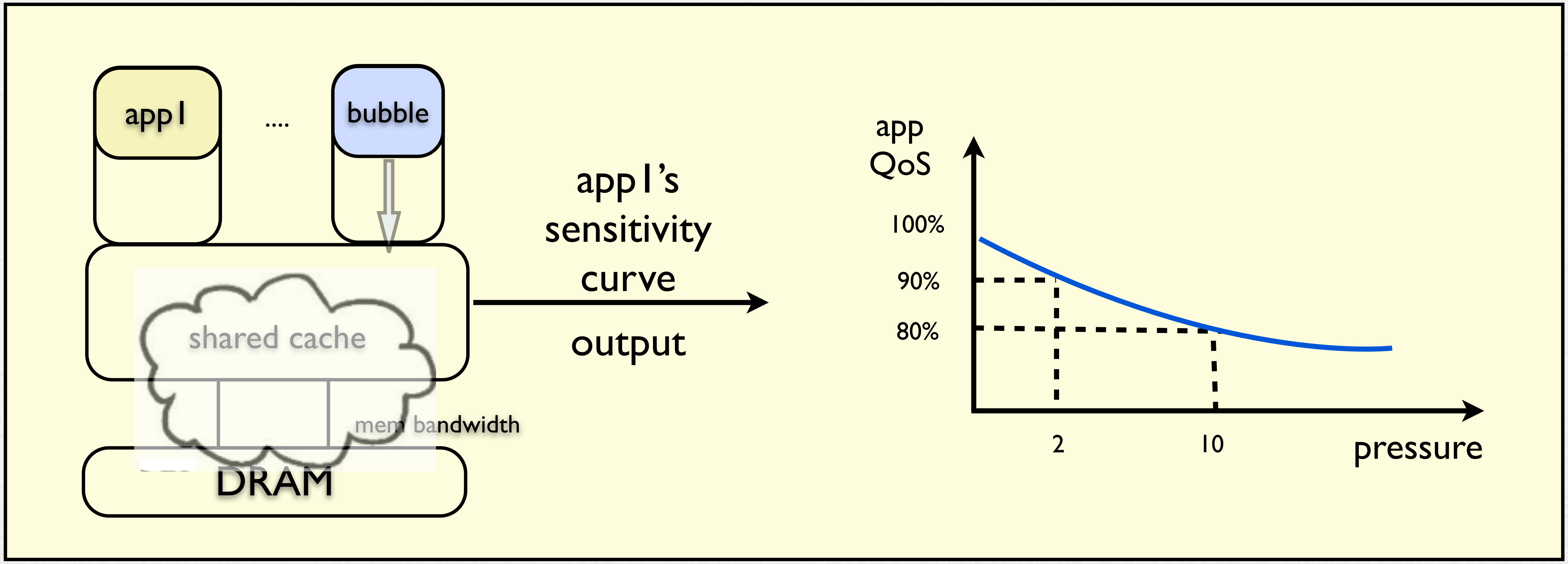
- Predict and manage interference to facilitate "safe" colocation to increase utilization without QoS degradation

Bubble-Up [Mars et al. Micro '11]

- State-of-the-art
- Static profiling to precisely predict the QoS interference and degradation for latency-sensitive applications
- 2% prediction error for large-scale applications on real hardware
- Insights:
  - Black box approach on real systems instead of detailed HW resource component modeling
  - Capture application's **sensitivity** to resource contention and **aggressiveness** separately


app1
....
bubble
shared cache
mem bandwidth
DRAM


app1
....
bubble
shared cache
mem bandwidth
DRAM



app1

....

bubble

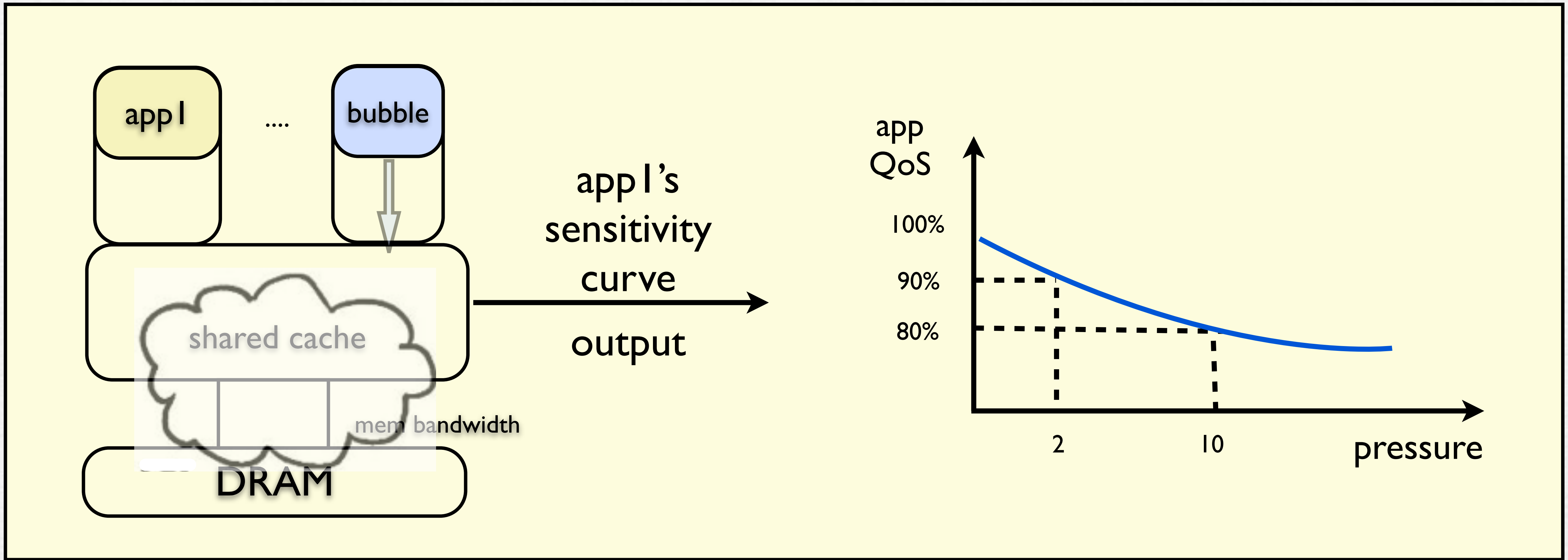
shared cache

mem bandwidth

DRAM

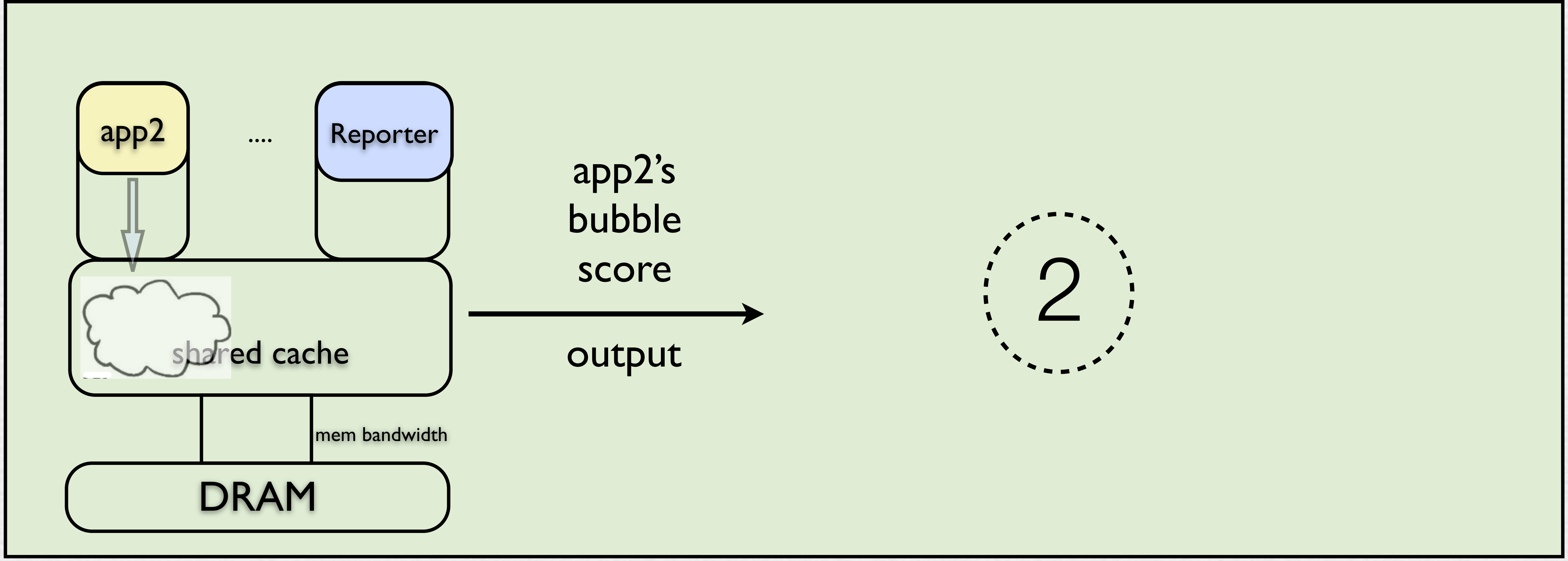
app1's sensitivity curve

output



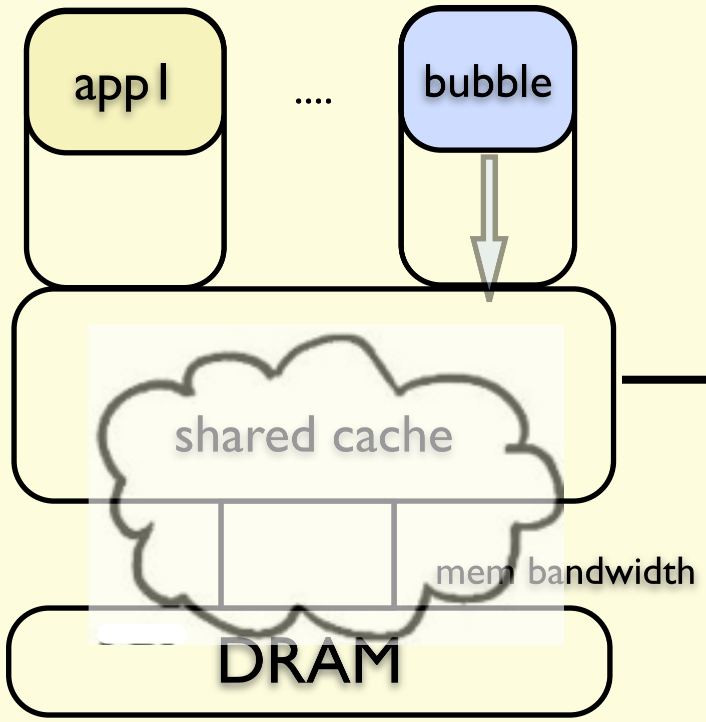
app1's sensitivity curve

output



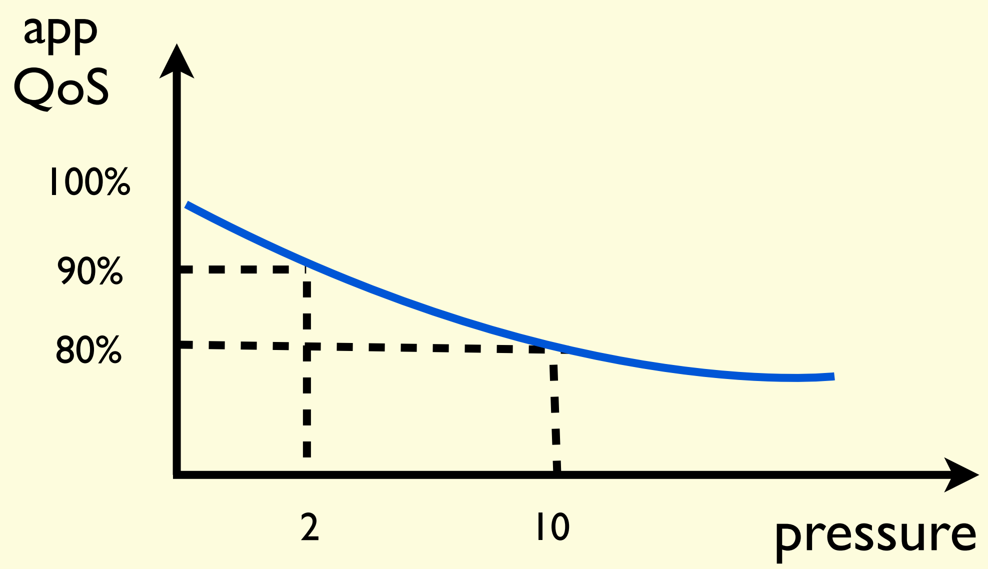
app2's bubble score

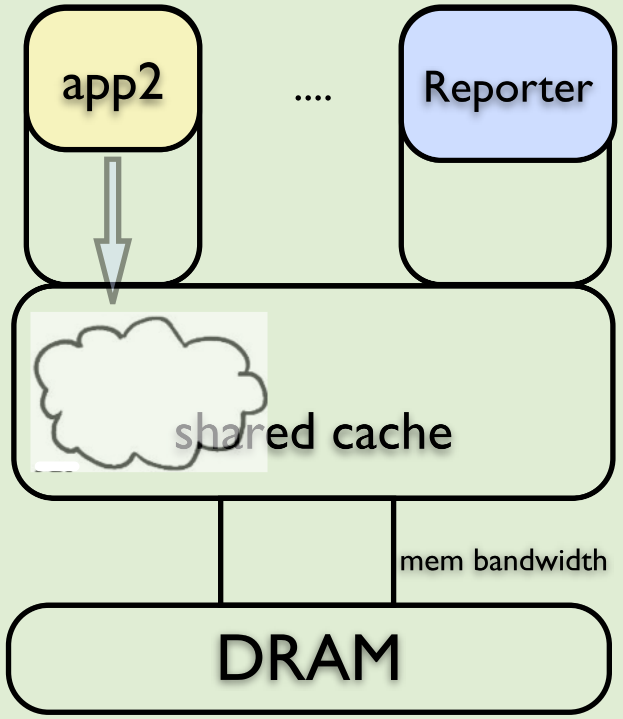
output



app1's sensitivity curve

output

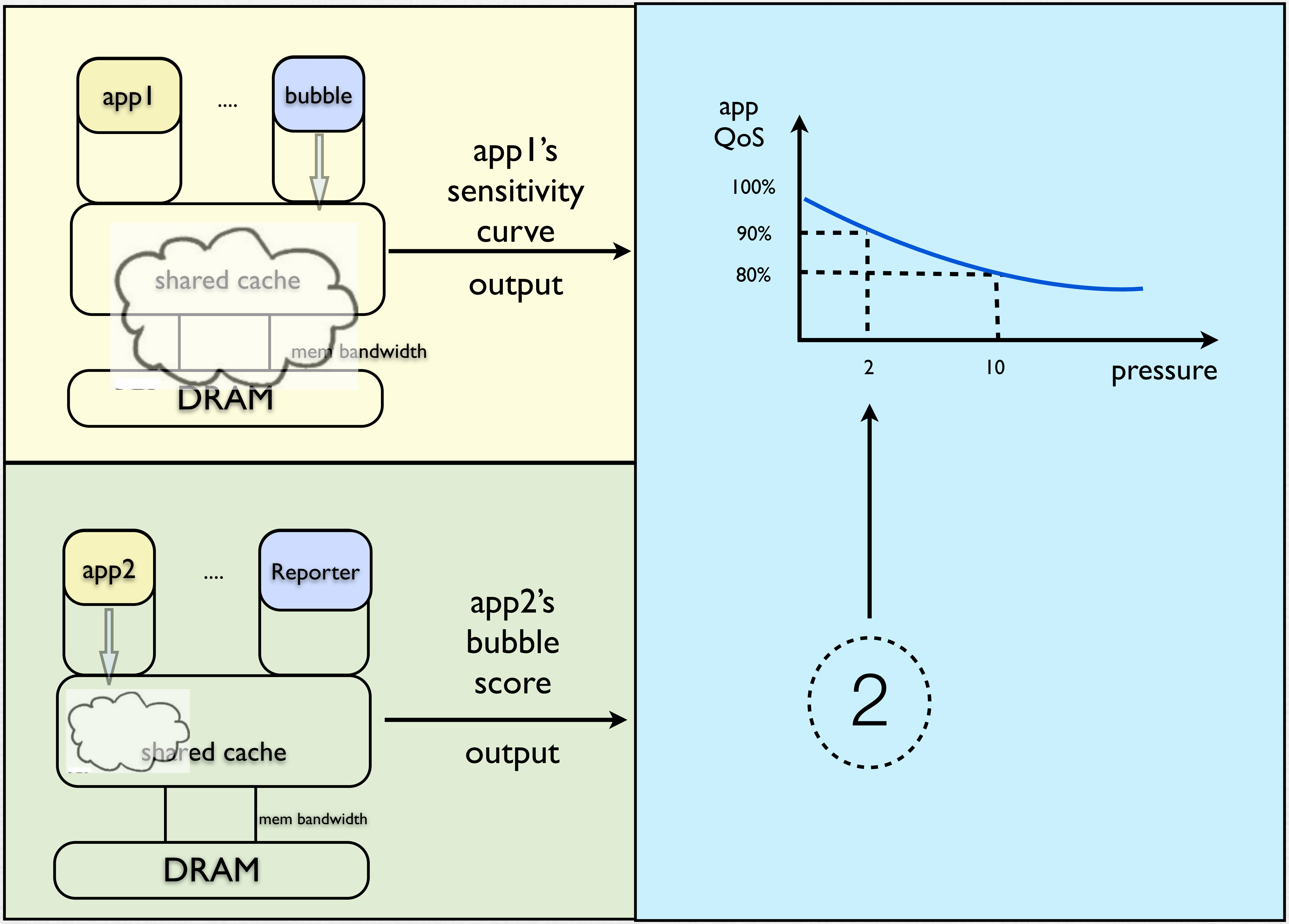




app2's bubble score

output

2



app1 .... bubble

shared cache

mem bandwidth

DRAM

app1's sensitivity curve

output

app2 .... Reporter

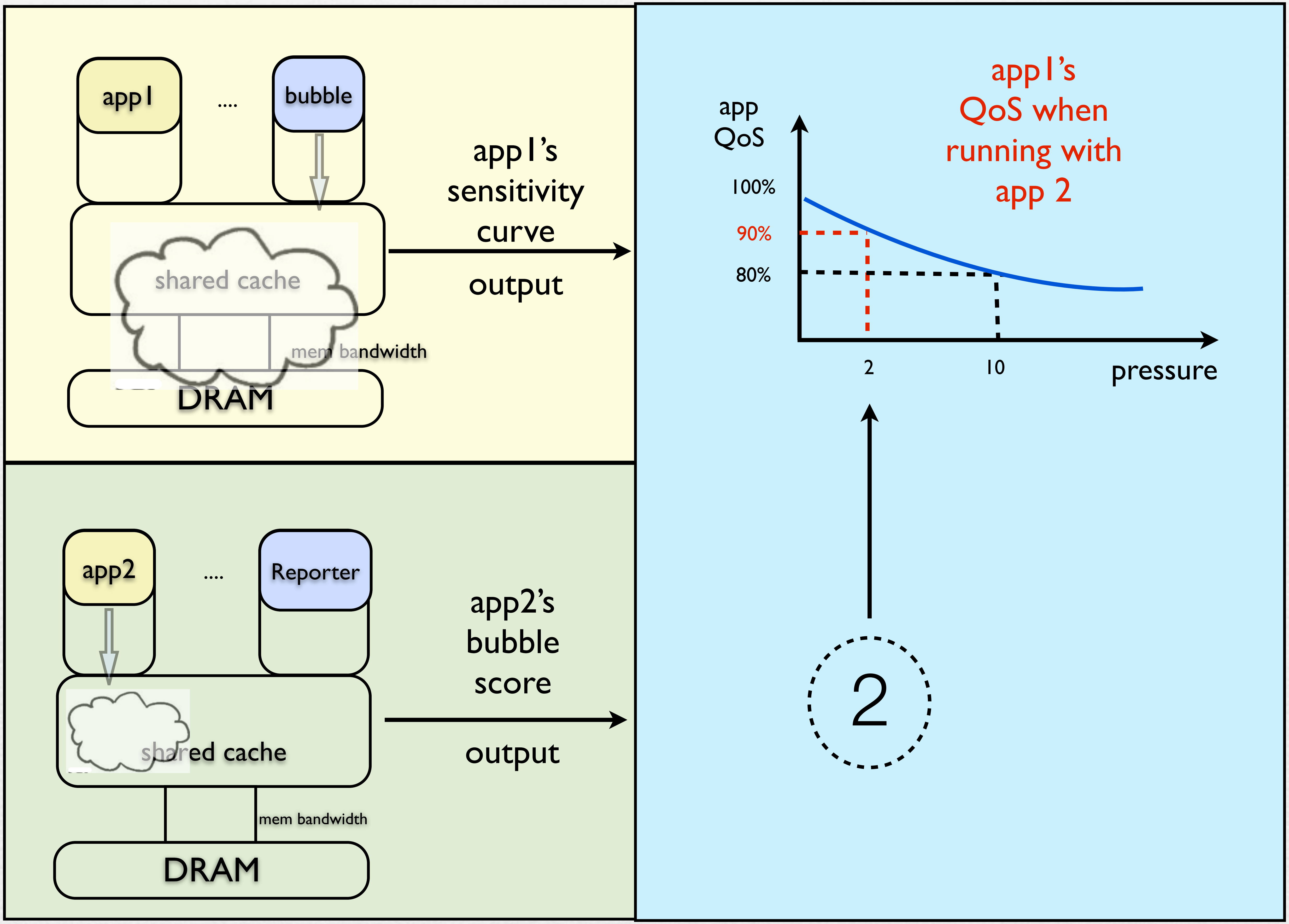
shared cache

mem bandwidth

DRAM

app2's bubble score

output

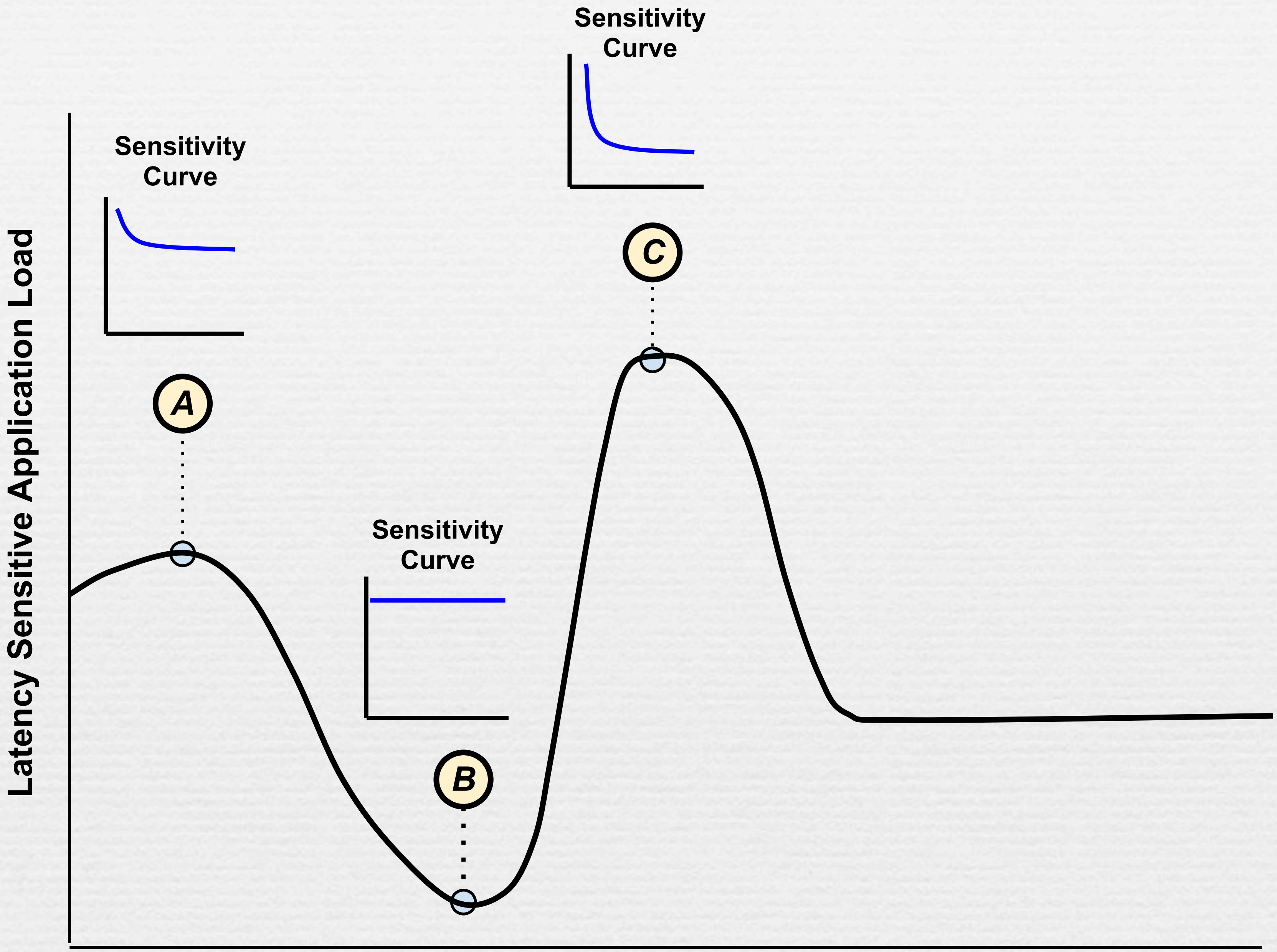

app1
....
bubble
shared cache
mem bandwidth
DRAM
app1's
sensitivity
curve
output
app2
....
Reporter
shared cache
mem bandwidth
DRAM
app2's
bubble
score
output
app
QoS
100%
90%
80%
2
10
pressure
app1's
QoS when
running with
app 2
2

Limitations of Bubble-Up

Limitations of Bubble-Up

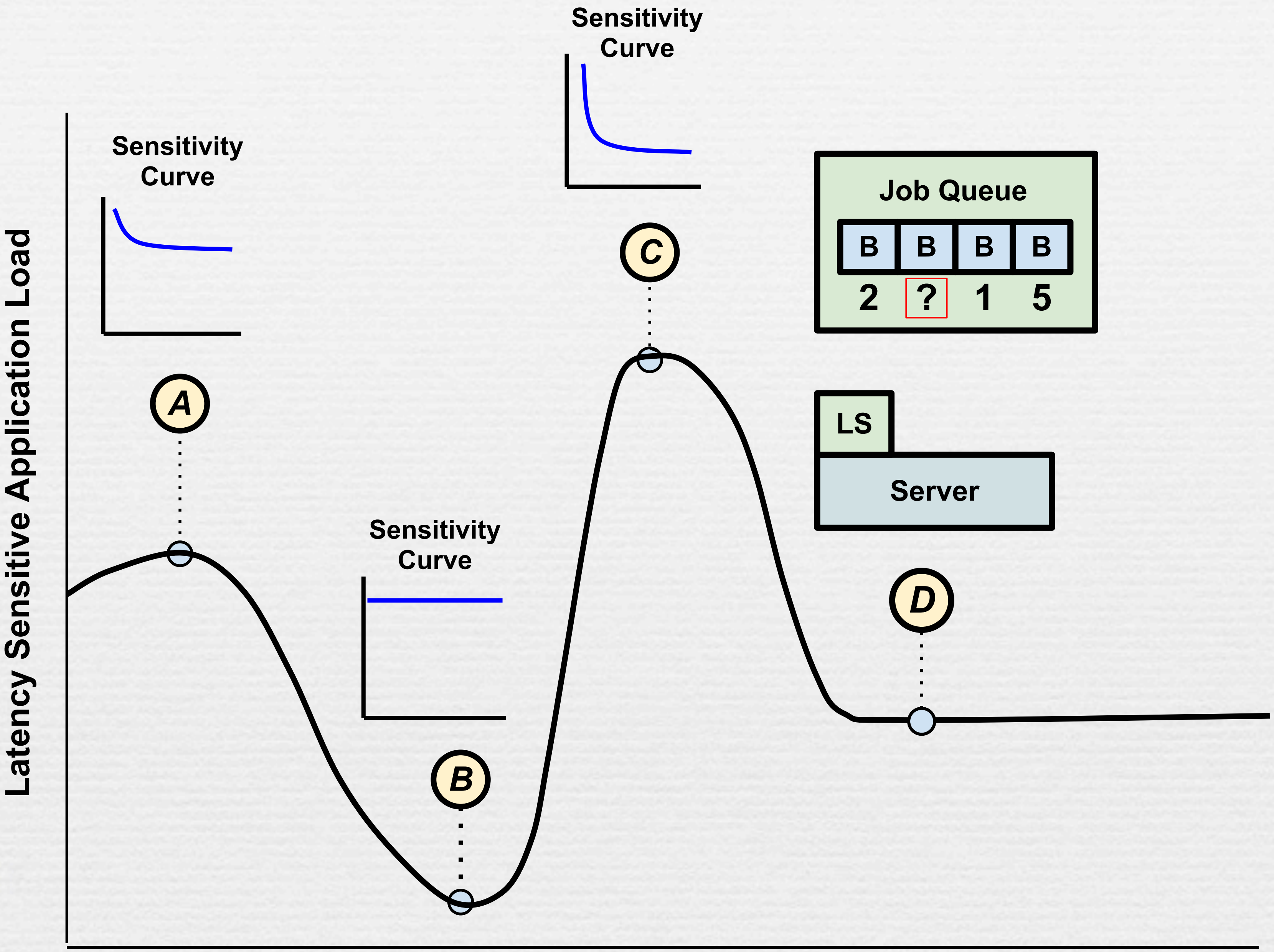
- **Limitation 1** - Inability to adapt, which significantly limits utilization opportunities

Limitations of Bubble-Up



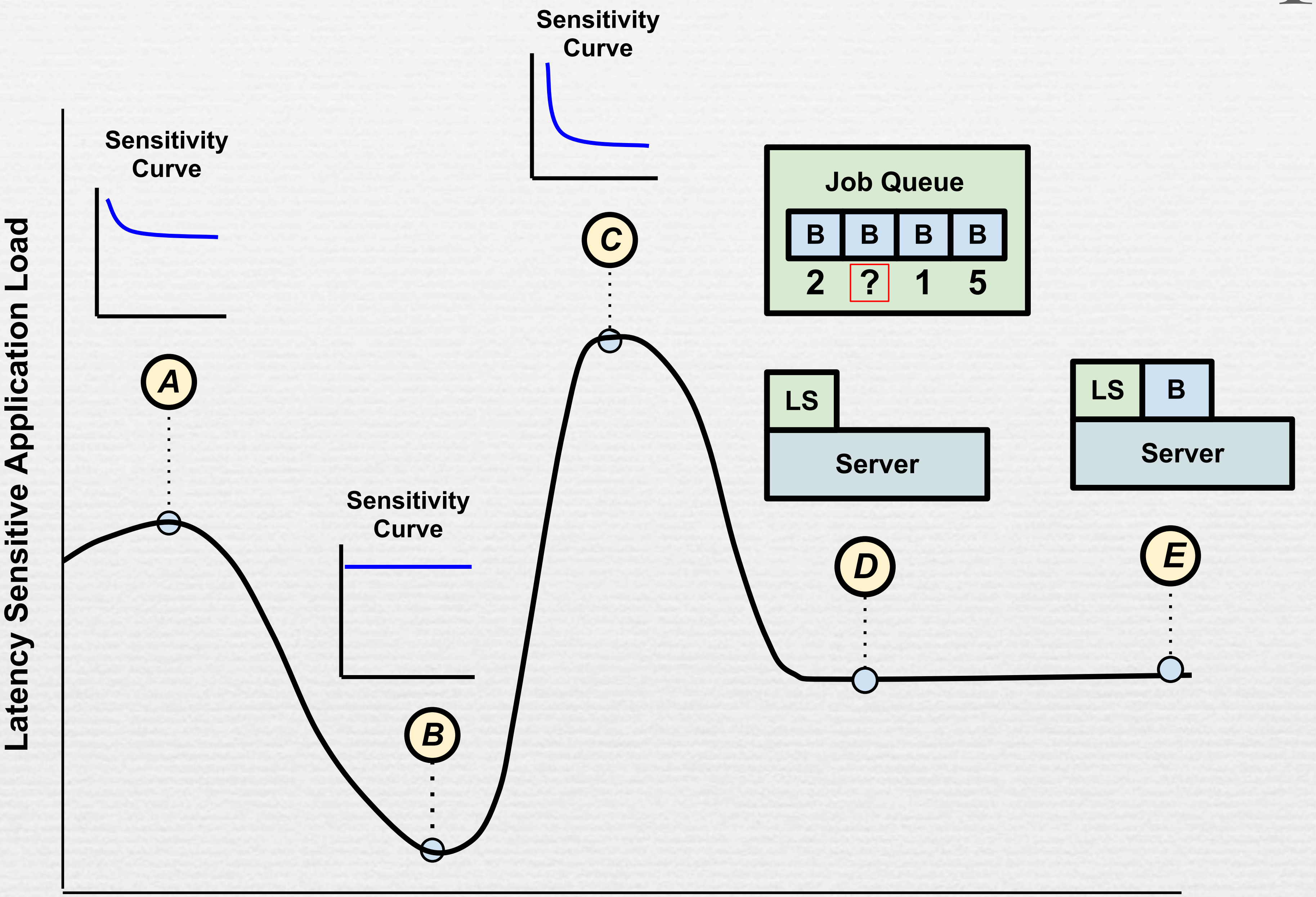
- **Limitation 1** - Inability to adapt, which significantly limits utilization opportunities

Limitations of Bubble-Up



- **Limitation 1** - Inability to adapt, which significantly limits utilization opportunities
- **Limitation 2** - A priori knowledge required

Limitations of Bubble-Up



- **Limitation 1** - Inability to adapt, which significantly limits utilization opportunities
- **Limitation 2** - A priori knowledge required
- **Limitation 3** - Limited Co-location Scalability

Bubble-Flux

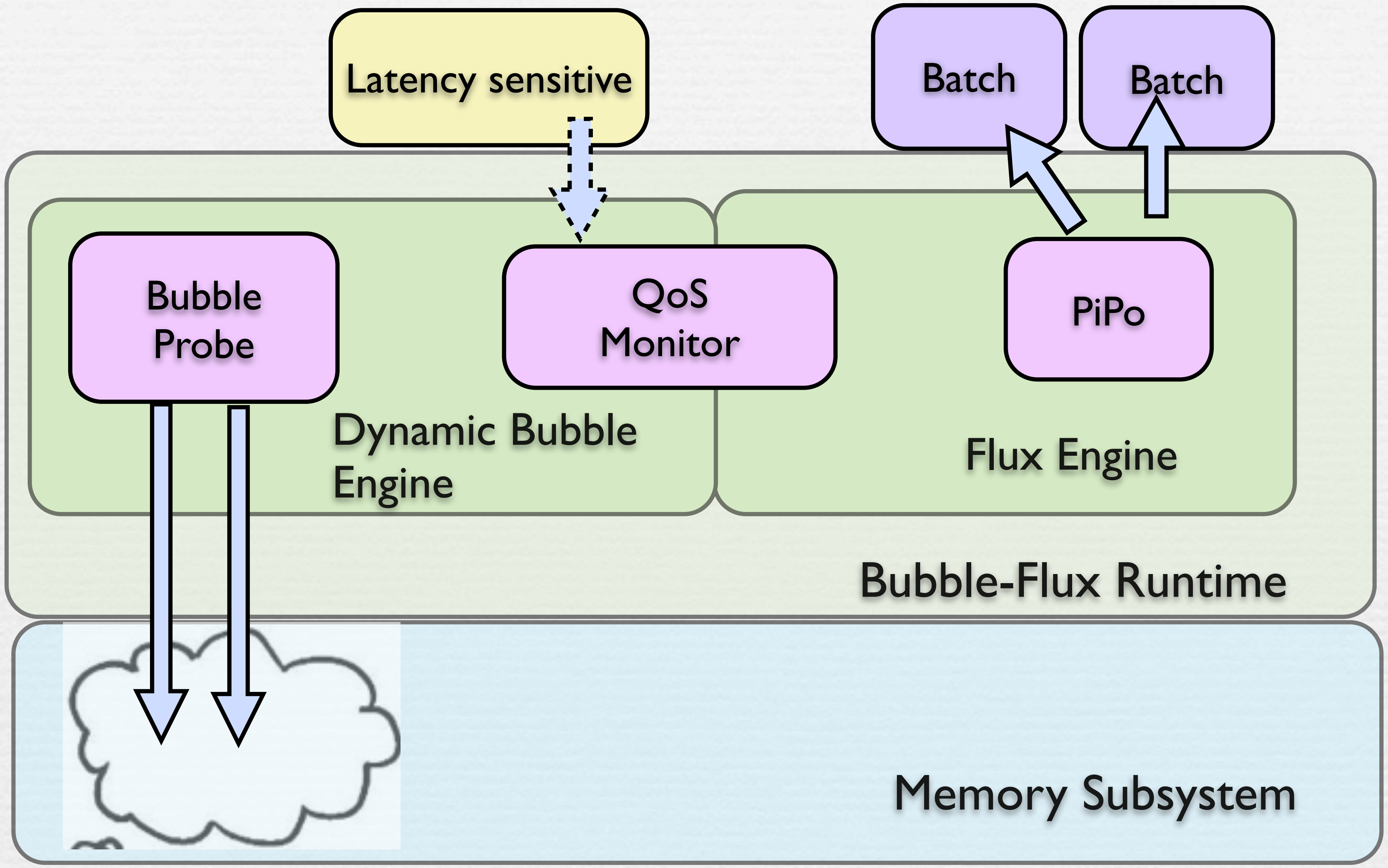
Bubble-Flux

- Instantaneous measurement of the application's sensitivity for each live server in production
    - Real time instead of static profiling
    - Adapt to load changes: reflect application's sensitivity at the current load level
    - Scale beyond pairwise
    - Better prediction-based "safe" co-location identification to maximize utilization

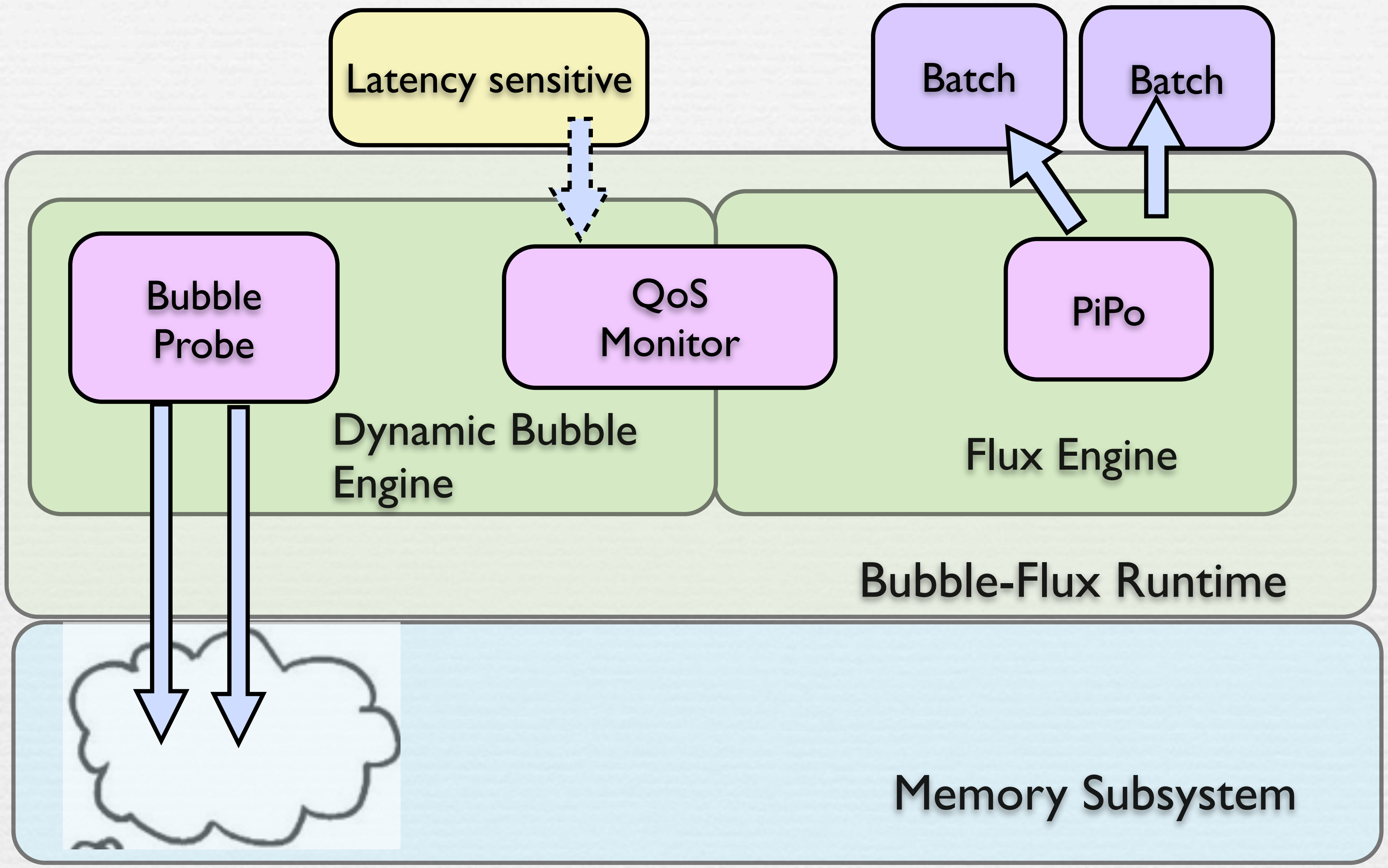
Bubble-Flux

- Instantaneous measurement of the application's sensitivity for each live server in production
  - Real time instead of static profiling
  - Adapt to load changes: reflect application's sensitivity at the current load level
  - Scale beyond pairwise
  - Better prediction-based "safe" co-location identification to maximize utilization
- Continuous online precise QoS management after the task is mapped
  - Adapt to load, phase, input changes
  - Handles unknown applications and beyond pairwise colocations

Bubble-Flux Overview

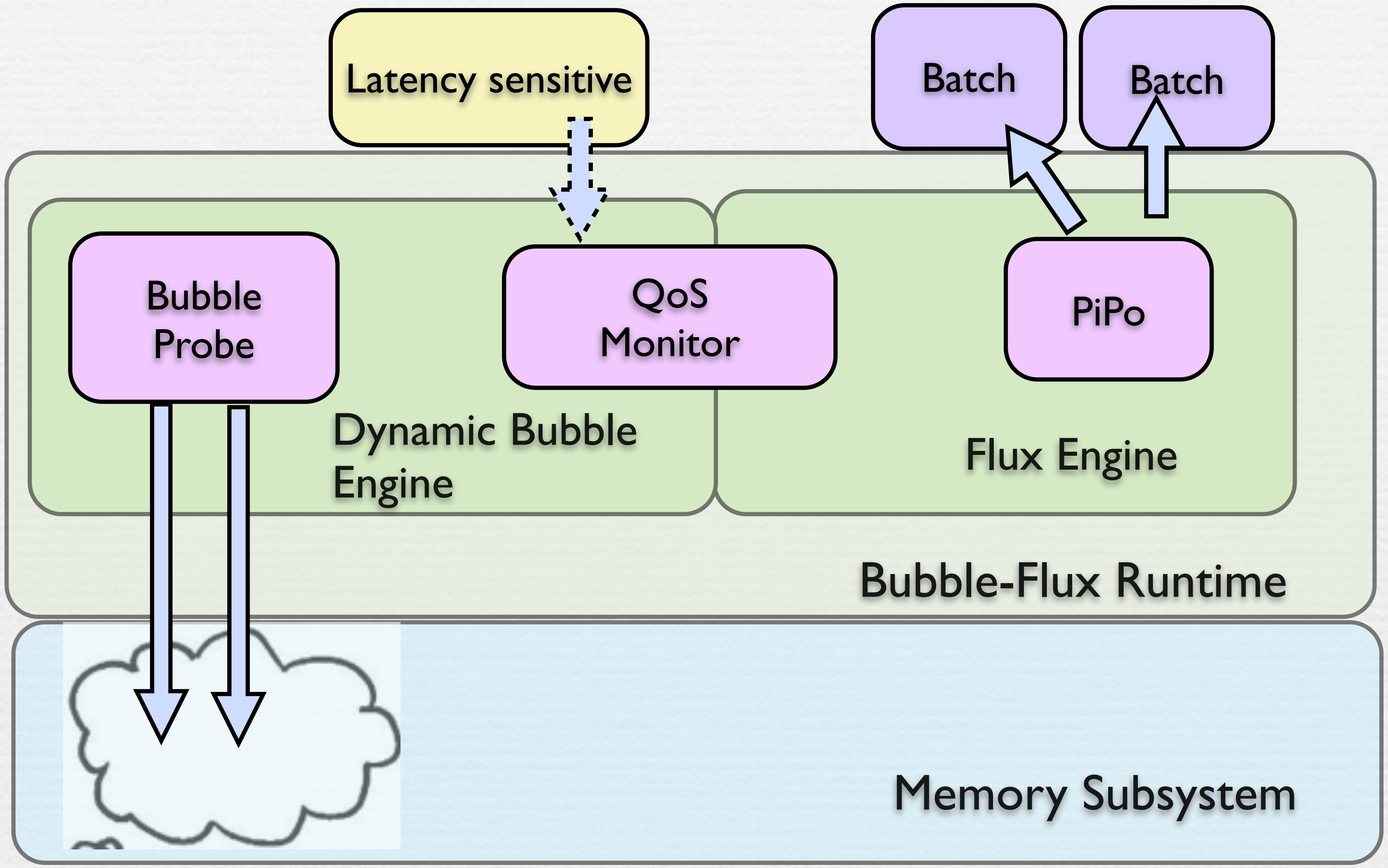


Bubble-Flux Overview



- **Dynamic Bubble -** Dynamically probe the machines to measure the latency-sensitive application's instantaneous sensitivity to the pressure on the shared hardware resources

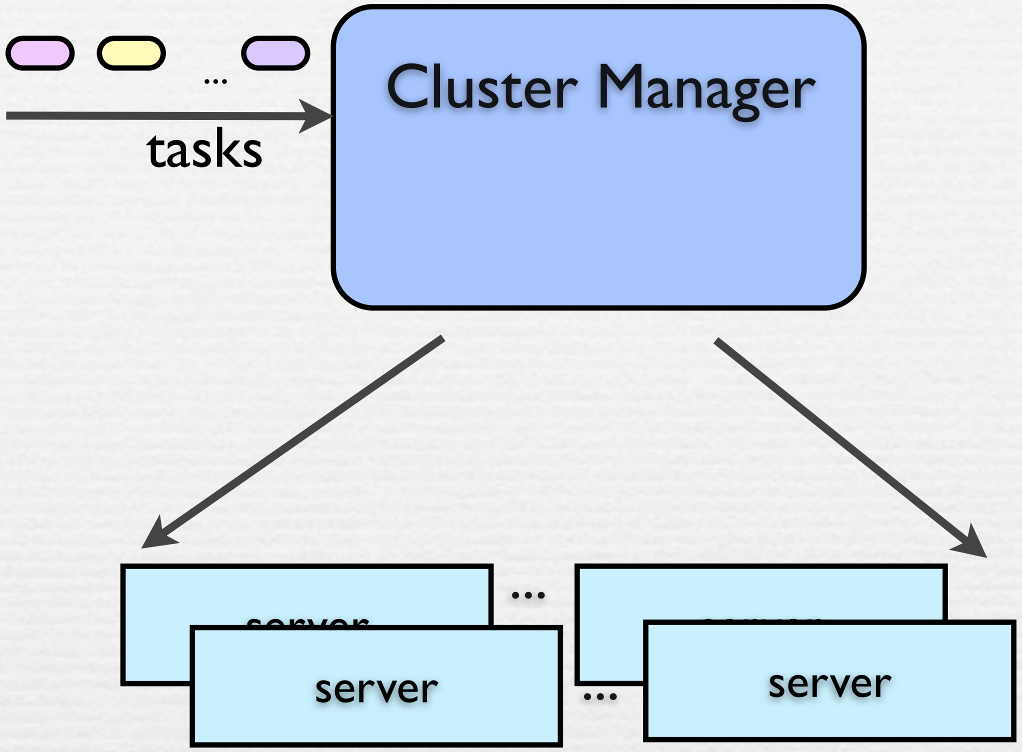
Bubble-Flux Overview



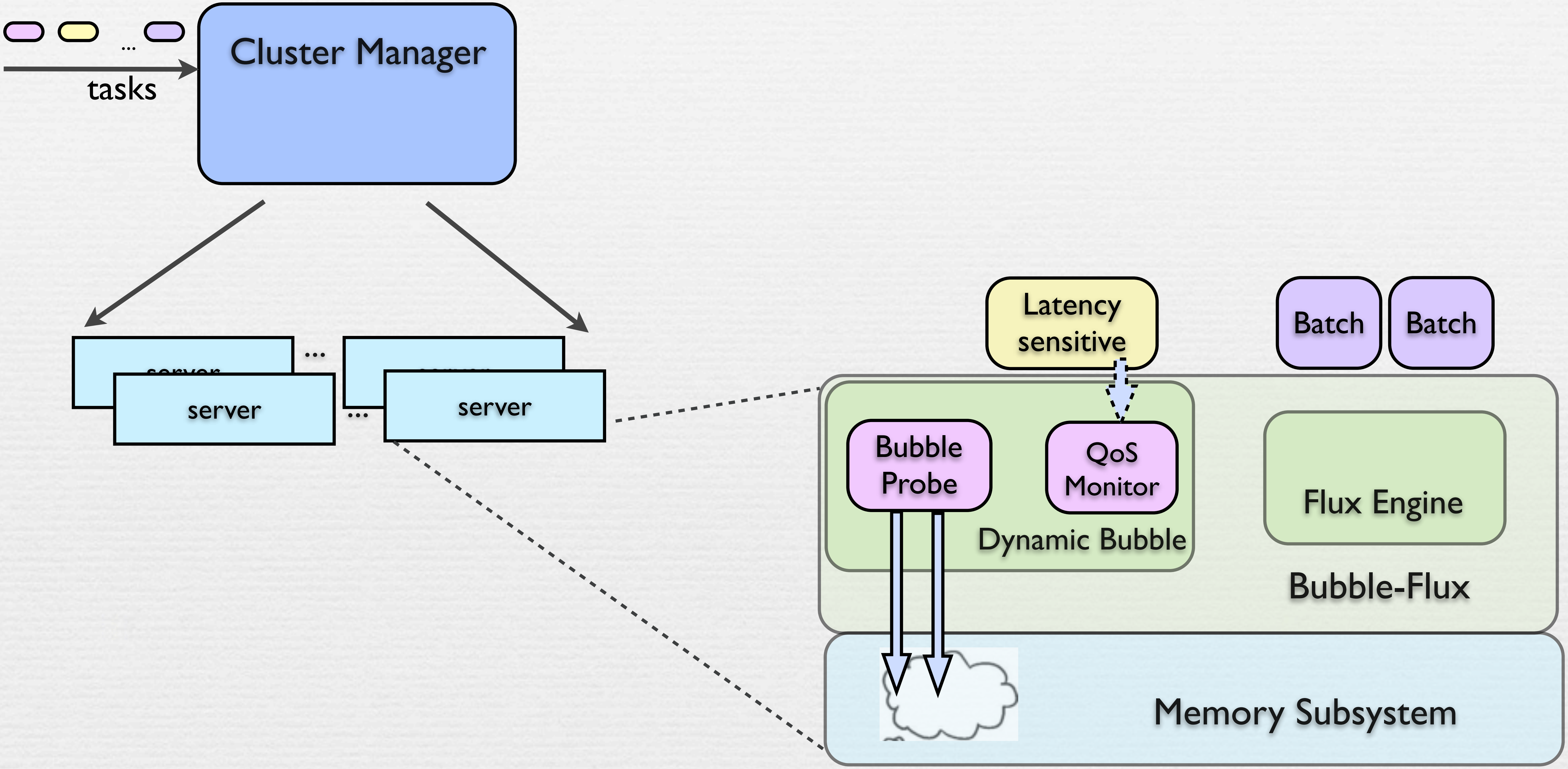
- **Dynamic Bubble -** Dynamically probe the machines to measure the latency-sensitive application's instantaneous sensitivity to the pressure on the shared hardware resources

- **Online Flux Engine -** Continuous QoS monitoring and dynamic throttling of batch applications (Phase-in/Phase-out) for QoS management

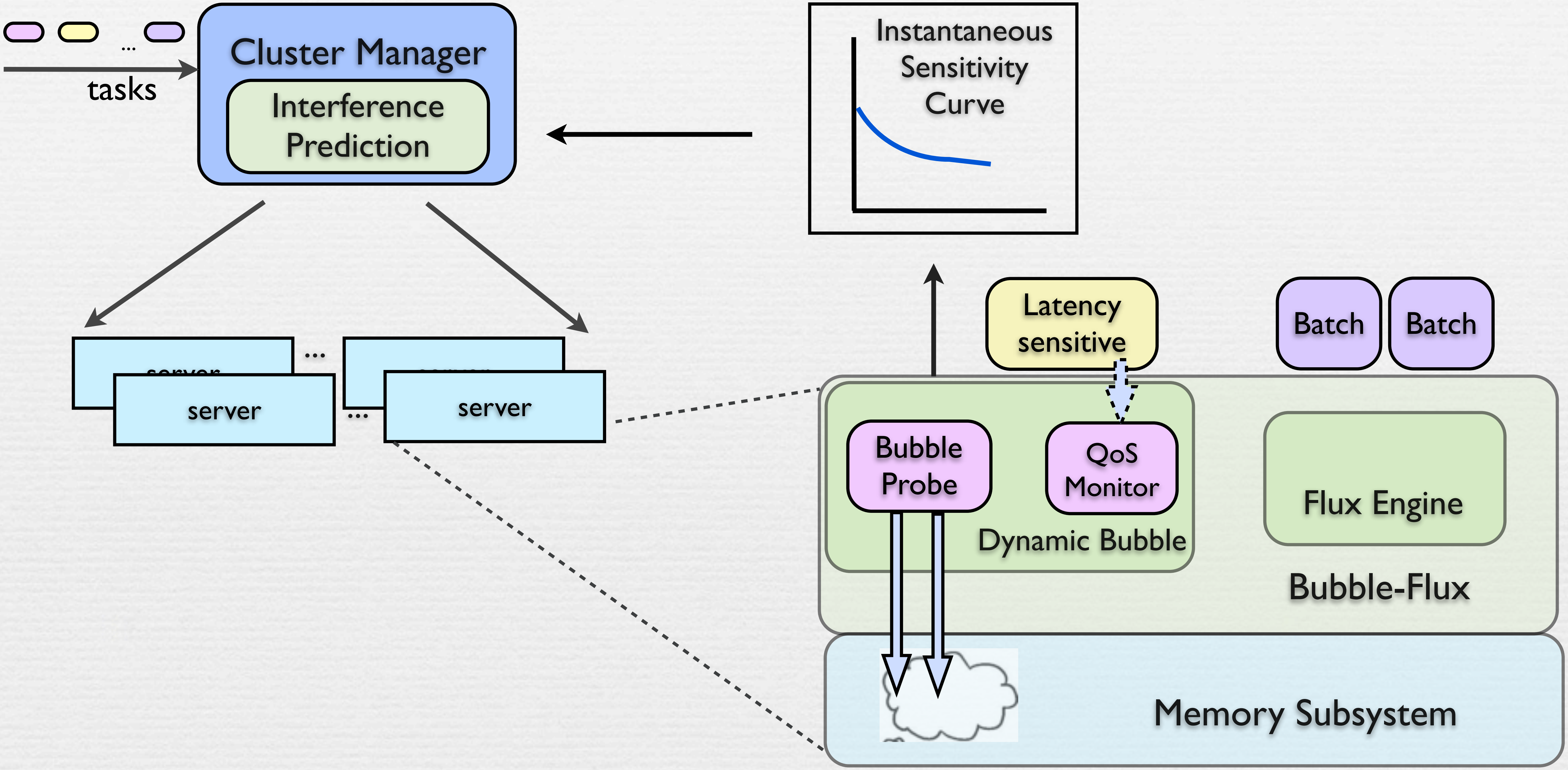
Dynamic Bubble



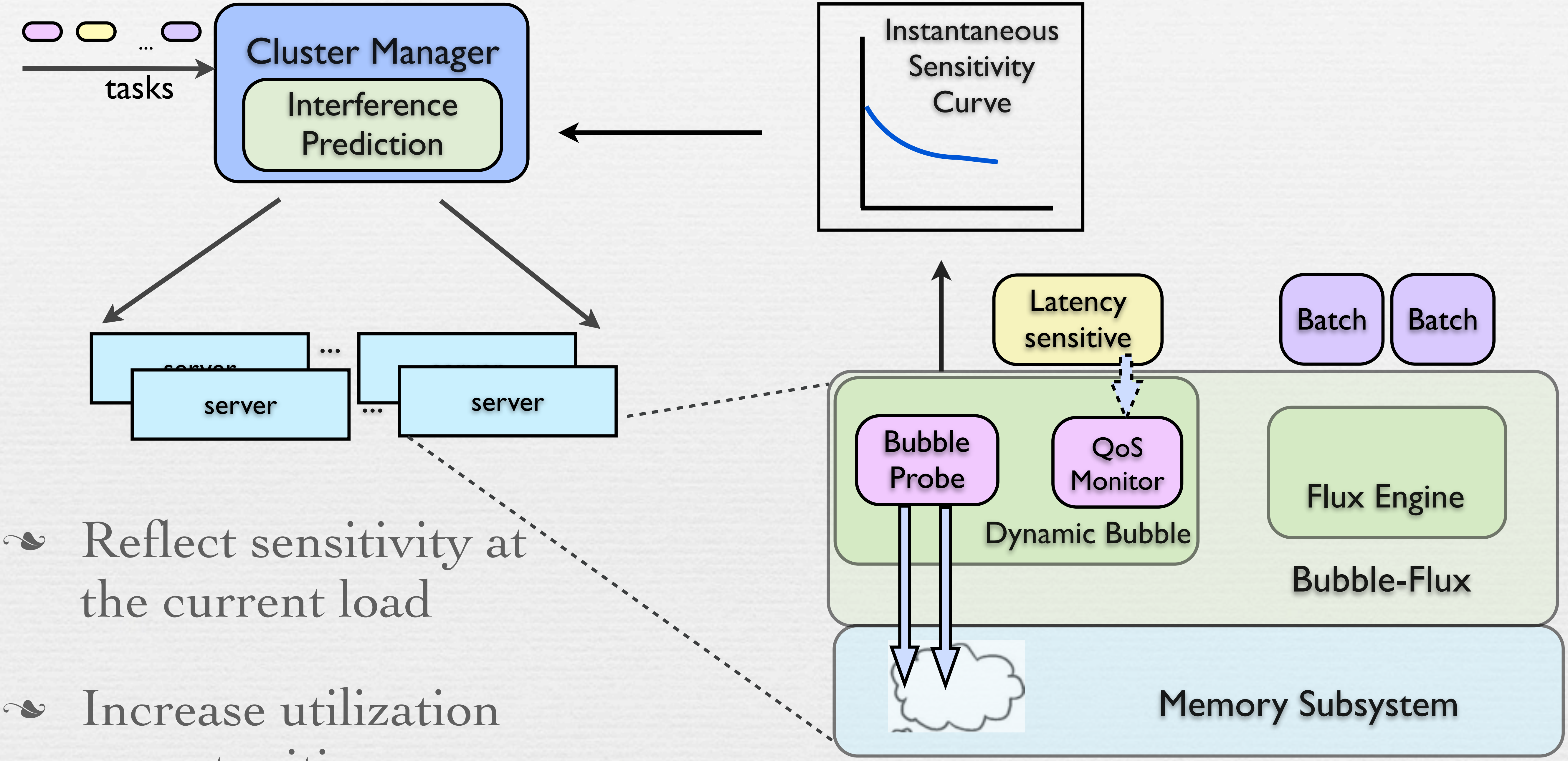
Dynamic Bubble



Dynamic Bubble



Dynamic Bubble

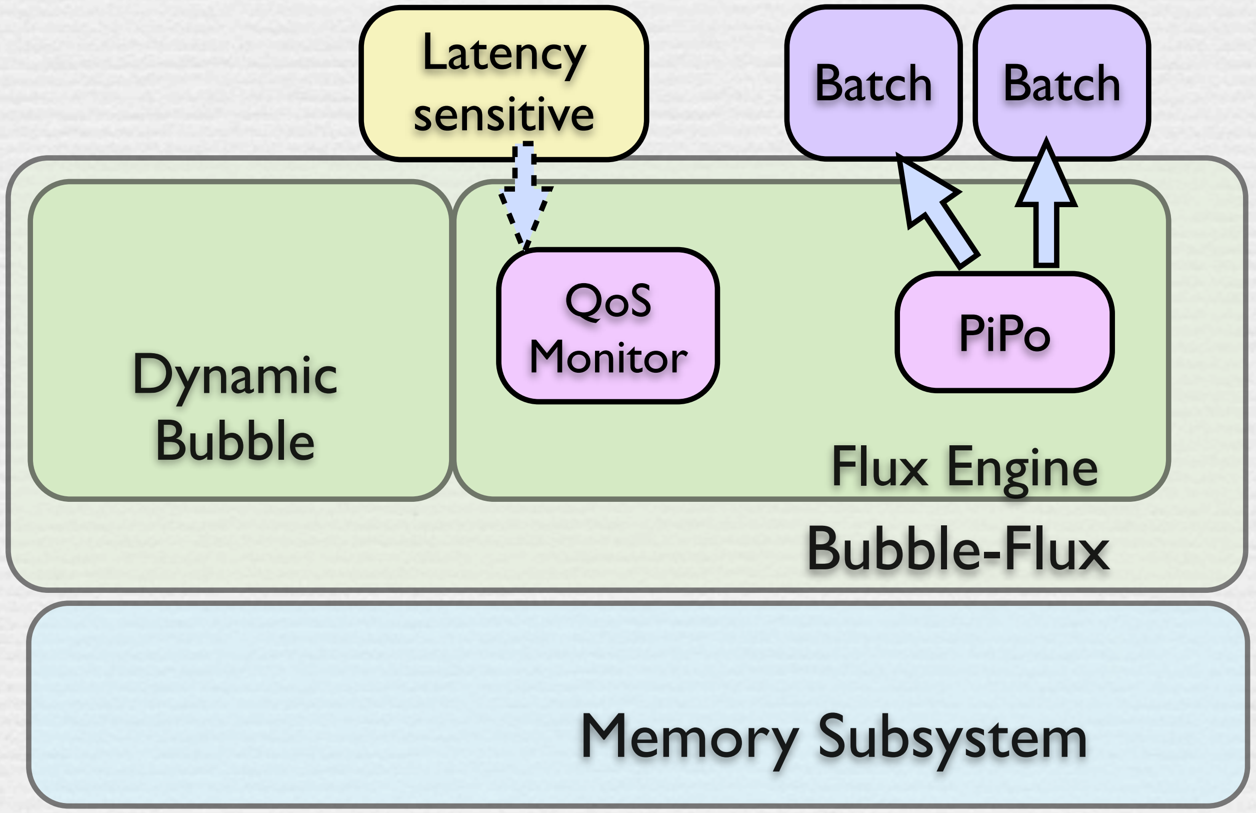


- Reflect sensitivity at the current load
- Increase utilization opportunities
- Beyond pairwise
- Low-overhead

Challenges and Design

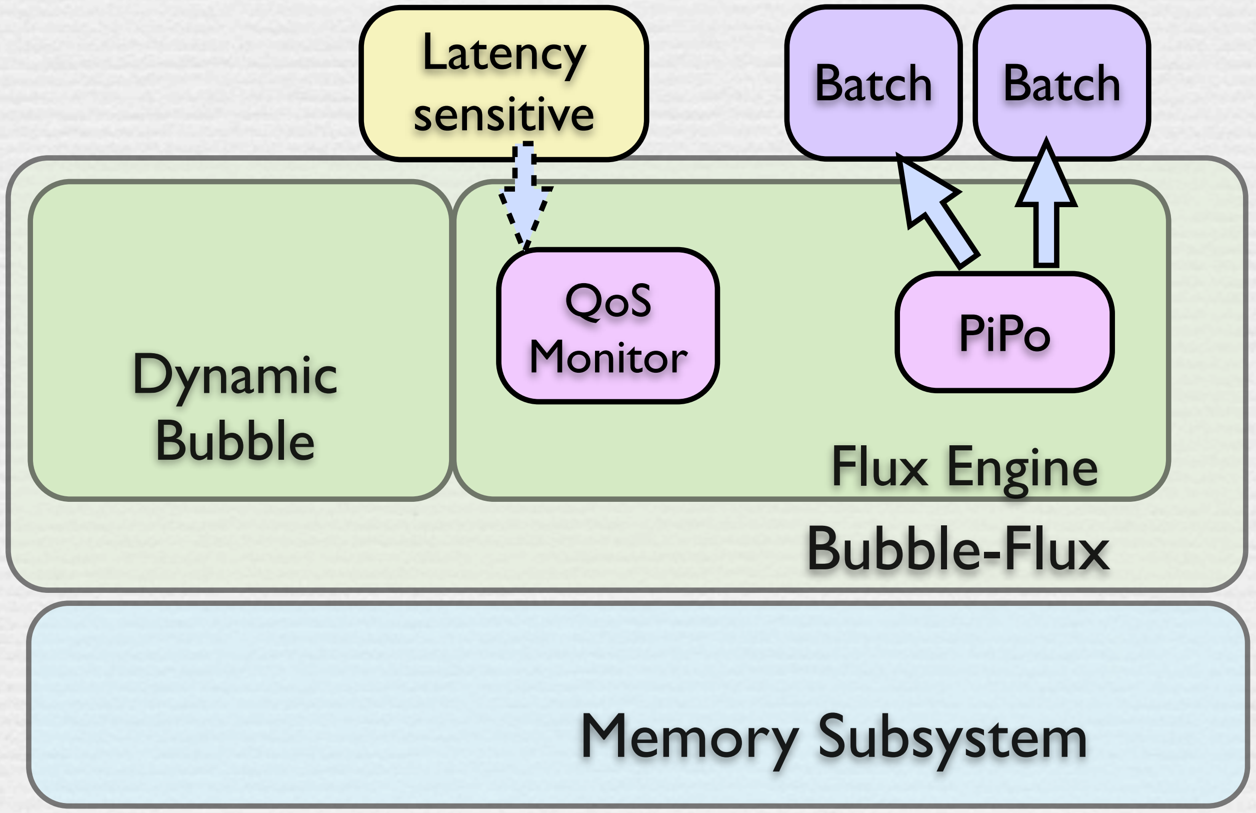
- Challenges
  - To generate a complete sensitivity curve with minimum runtime overhead and interference
- Design: rely on the Flux engine to control the interference caused by the dynamic bubble
  - Phase-in and Phase-out (PiPo)
  - Measure the QoS delta when bubble is phased in and phased out with *controllable* interference (e.g., 2%)
  - Generate sensitivity curve without violating QoS target

Online Flux Engine



Online Flux Engine

- Continuous QoS monitoring after tasks are mapped
- PiPo (Phase-in/Phase-out): Dynamic throttling of batch applications for QoS management of latency-sensitive application



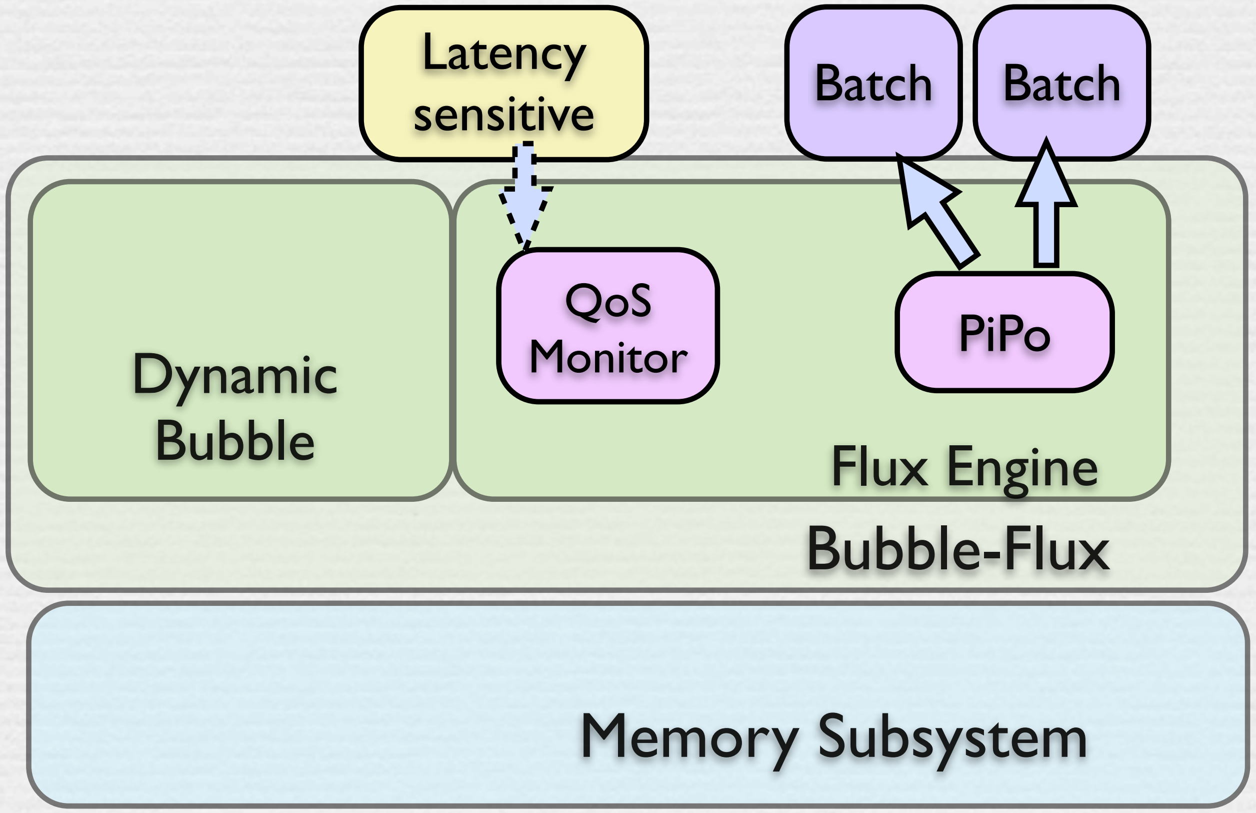
Online Flux Engine

- Continuous QoS monitoring after tasks are mapped

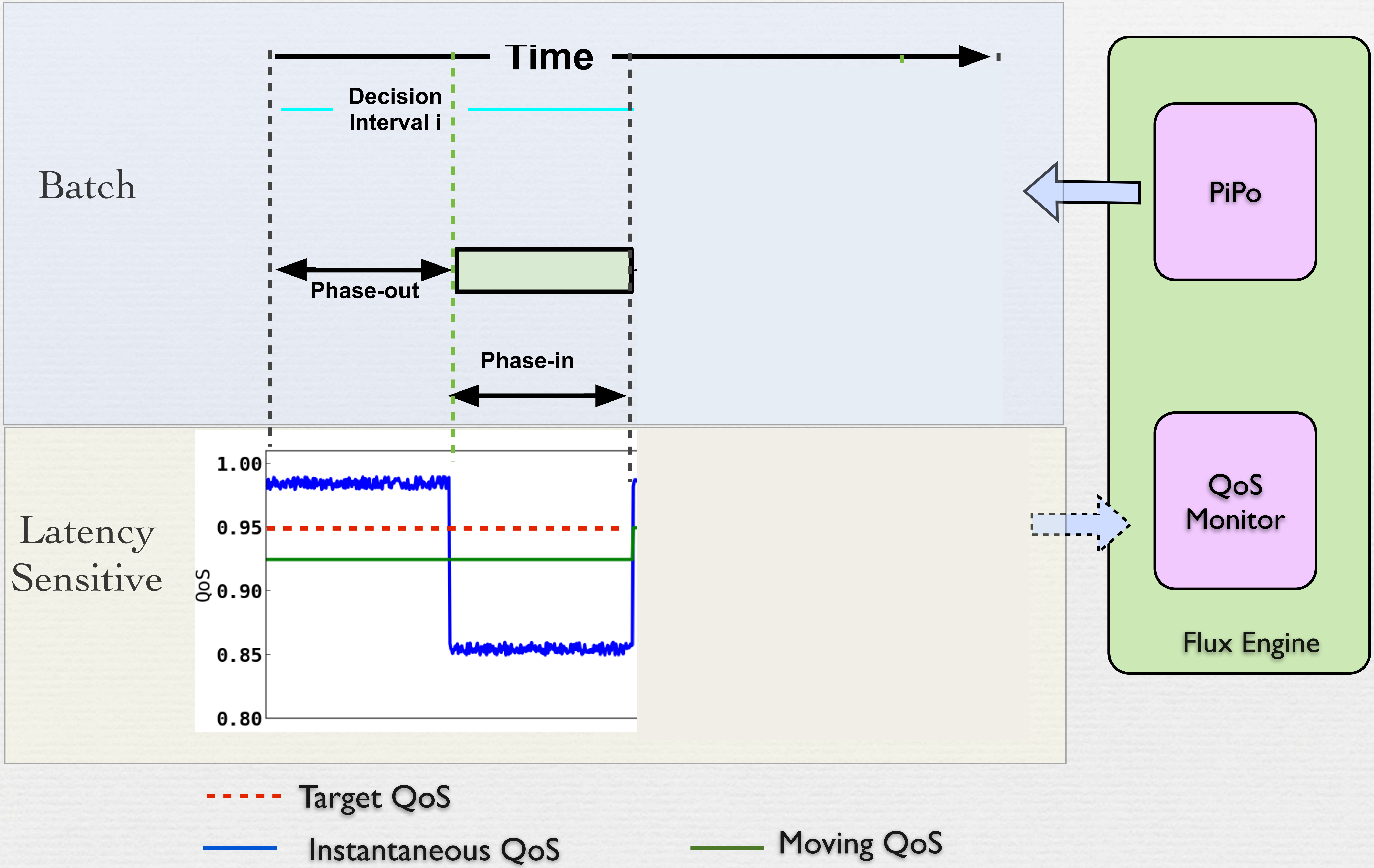
- PiPo (Phase-in/Phase-out): Dynamic throttling of batch applications for QoS management of latency-sensitive application

- Respond to execution phase changes, input changes, and load variations

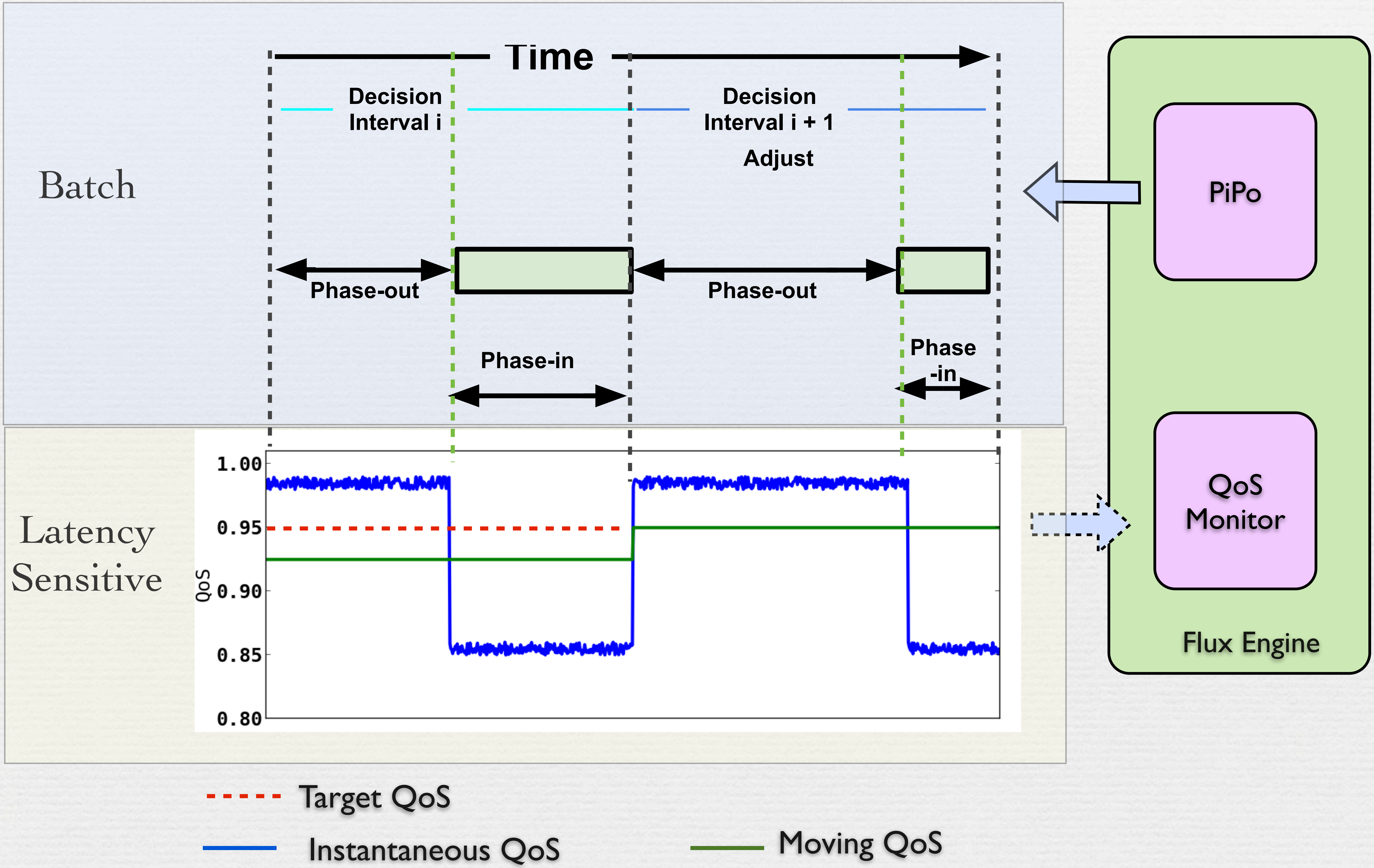
- Scale up beyond pair-wise, work with unknown applications



Online Flux Engine

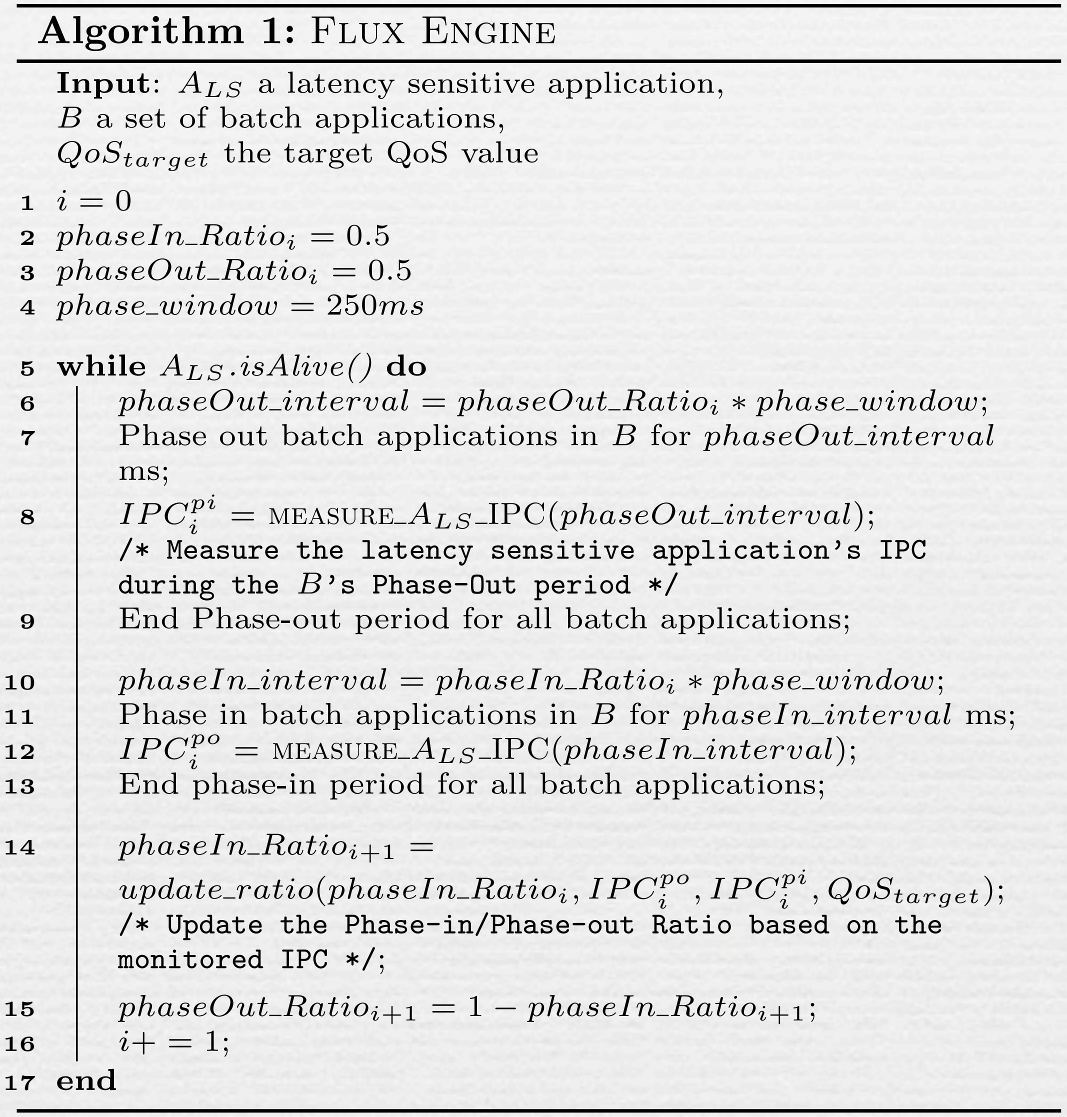


Online Flux Engine



Online Flux Engine

- Monitor: hardware performance counters (IPC)
- Phase-in/Phase-out: SIGSTOP and SIGCONT
- Phase-in/phase-out ratio in the next iteration:



Algorithm 1: Flux Engine

```
Input: $A_{LS}$ a latency sensitive application,
$B$ a set of batch applications,
$QoS_{target}$ the target QoS value
$i = 0$
$phaseIn\_Ratio_i = 0.5$
$phaseOut\_Ratio_i = 0.5$
$phase\_window = 250ms$
while $A_{LS}.isAlive()$ do
    $phaseOut\_interval = phaseOut\_Ratio_i * phase\_window$;
    Phase out batch applications in $B$ for $phaseOut\_interval$ ms;
    $IPC_i^{pi}$ = MEASURE_$A_{LS}$_IPC($phaseOut\_interval$);
    /* Measure the latency sensitive application's IPC during the $B$'s Phase-Out period */
    End Phase-out period for all batch applications;
    $phaseIn\_interval = phaseIn\_Ratio_i * phase\_window$;
    Phase in batch applications in $B$ for $phaseIn\_interval$ ms;
    $IPC_i^{po}$ = MEASURE_$A_{LS}$_IPC($phaseIn\_interval$);
    End phase-in period for all batch applications;
    $phaseIn\_Ratio_{i+1} = update\_ratio(phaseIn\_Ratio_i, IPC_i^{po}, IPC_i^{pi}, QoS_{target})$;
    /* Update the Phase-in/Phase-out Ratio based on the monitored IPC */;
    $phaseOut\_Ratio_{i+1} = 1 - phaseIn\_Ratio_{i+1}$;
    $i += 1$;
end
```

$$phaseInRatio_{i+1}^{pi} = phaseInRatio_i^{pi} + \frac{QoS_{target} - QoS_i}{QoS_{target}}$$

Evaluation Objectives

- How Bubble-Flux addresses 3 limitations of Bubble-Up:
  - L1: Unknown applications
  - L2: Adapt to load/input/phase changes
  - L3: Scale beyond pair-wise
- Applying Bubble-Flux in datacenter scenarios

Evaluation Setup

- Benchmark Suites
  - Cloud suite (Web-search, Data-serving, Data-analytics, Media-streaming, etc.) [Ferdman ’12]
  - SPEC CPU 2006
- Machine
  - 2.2 Ghz dual-socket Intel Xeon E5-2660 (Sandy bridge)
  - 8 cores + 32GB of DRAM per socket
  - 32KB L1 i-cache, 32KB L1 d-cache, 256 KB L2 cache, 20MB L3 cache

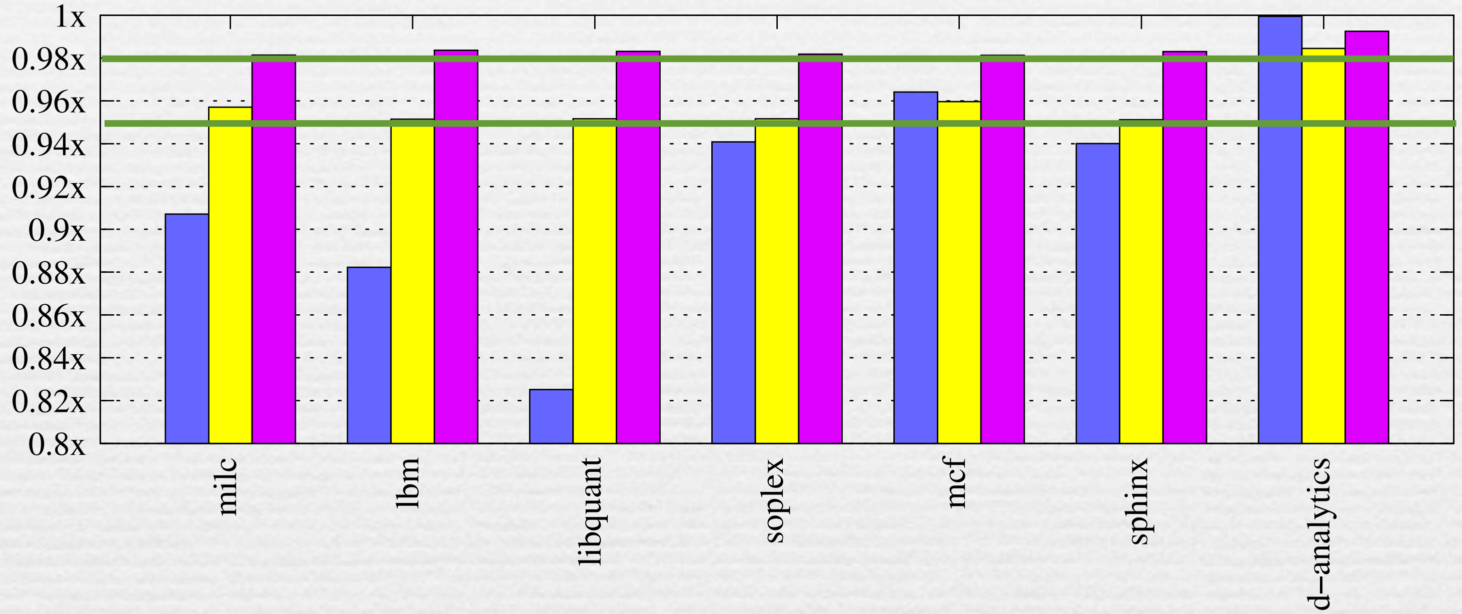
Flux: Effectiveness w/o a priori knowledge (L1)

🟦 Baseline: original

🟨 Flux targeting 95% QoS

🟪 Flux targeting 98% QoS

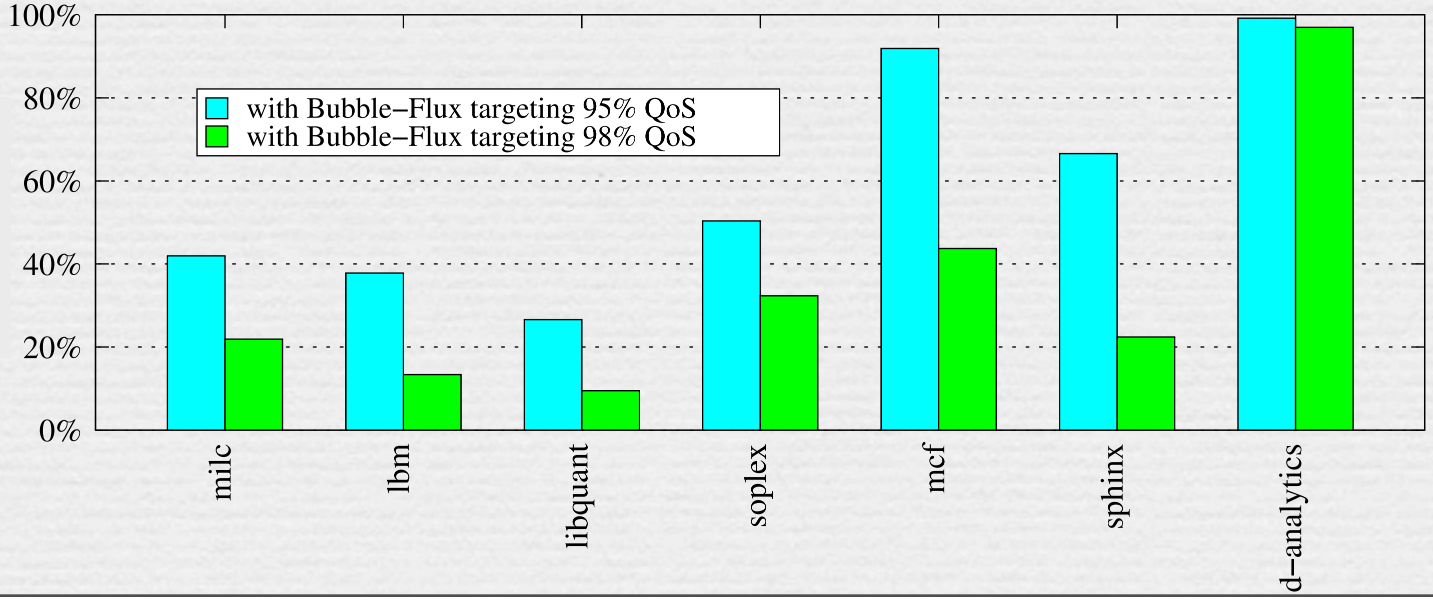
QoS of Web-search



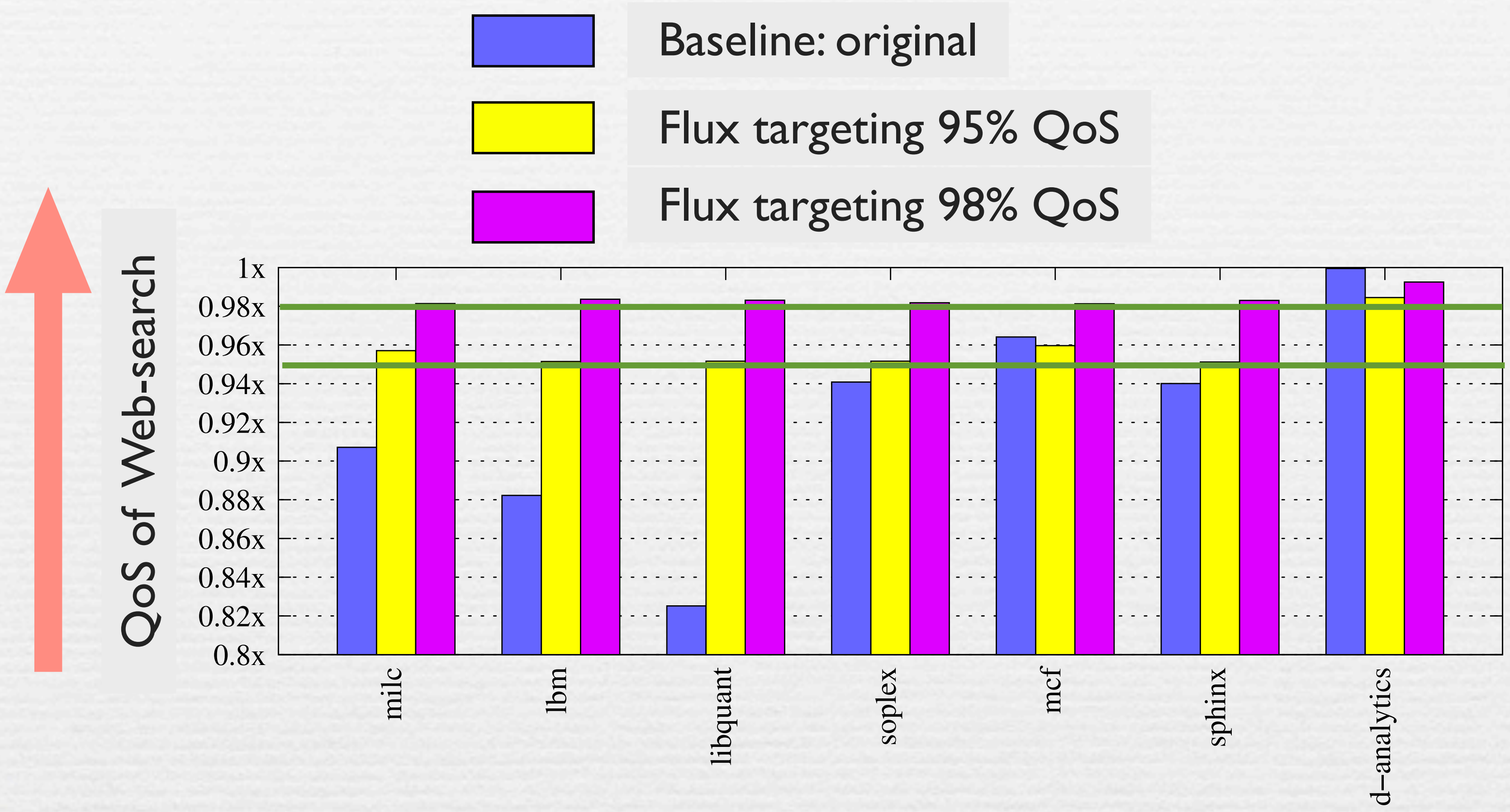
🟦 Flux: targeting 95% QoS

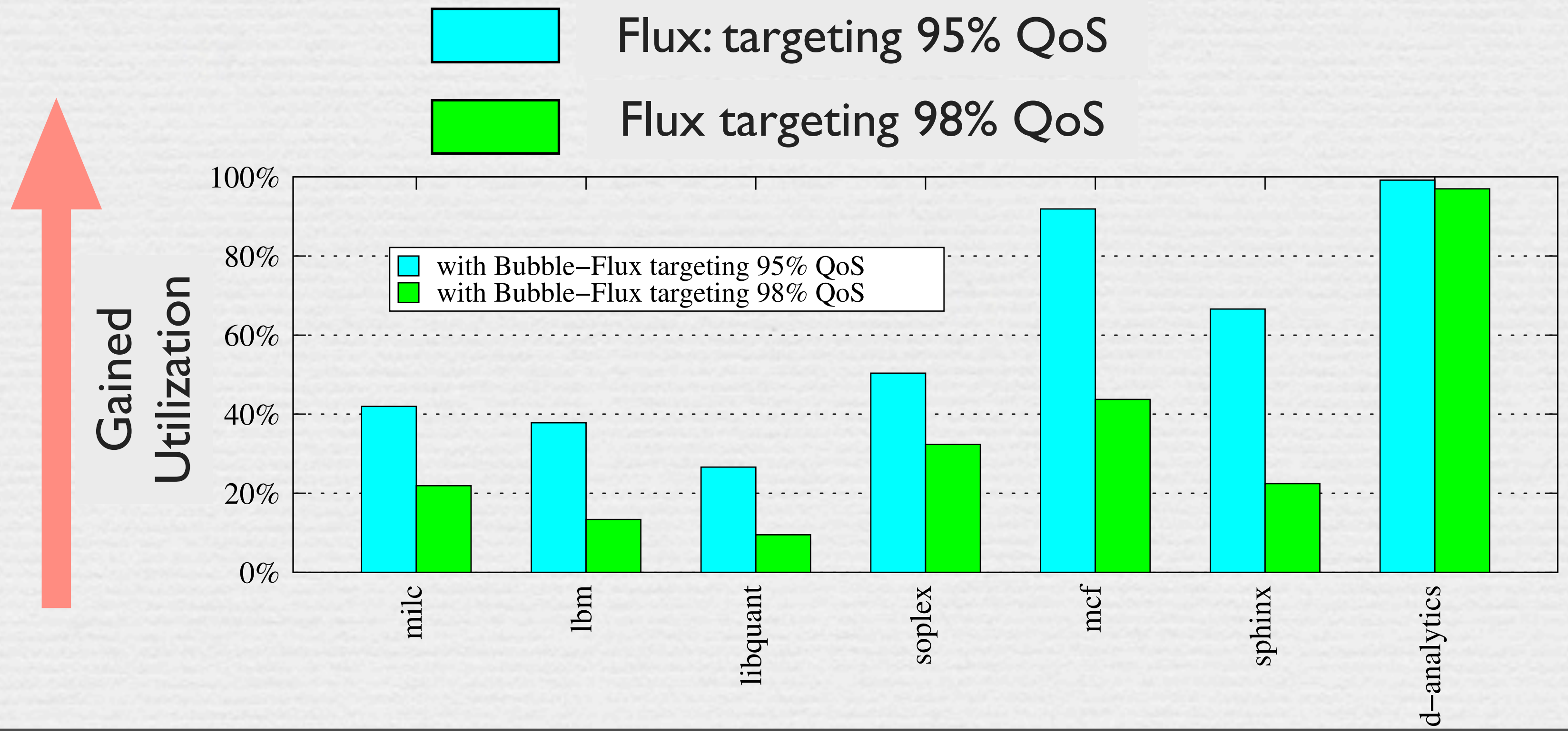
🟩 Flux targeting 98% QoS

Gained Utilization



Flux: Effectiveness w/o a priori knowledge (L1)

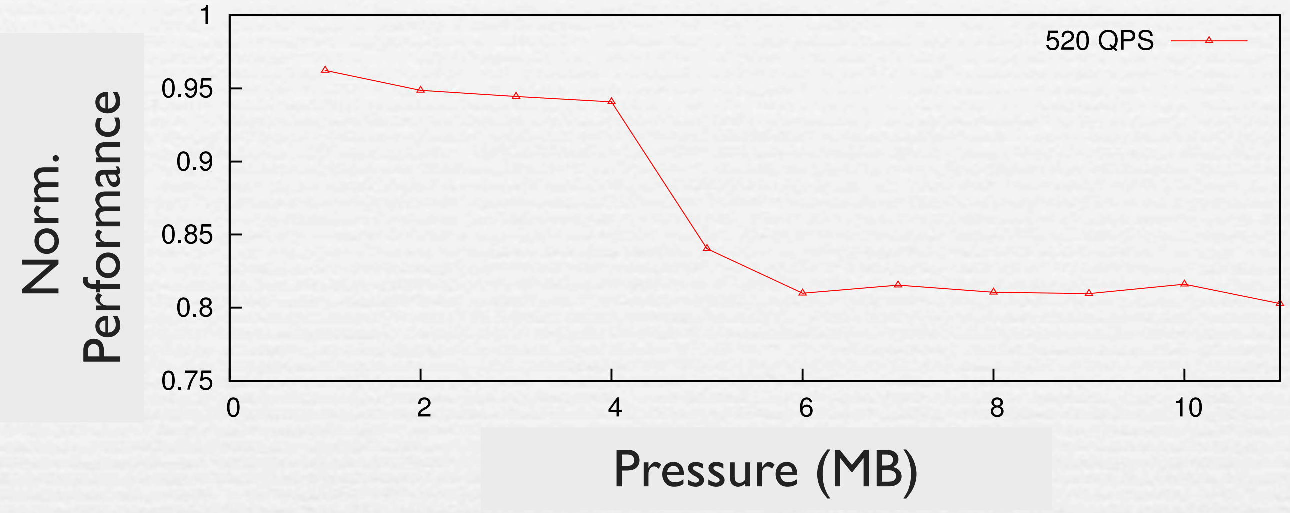


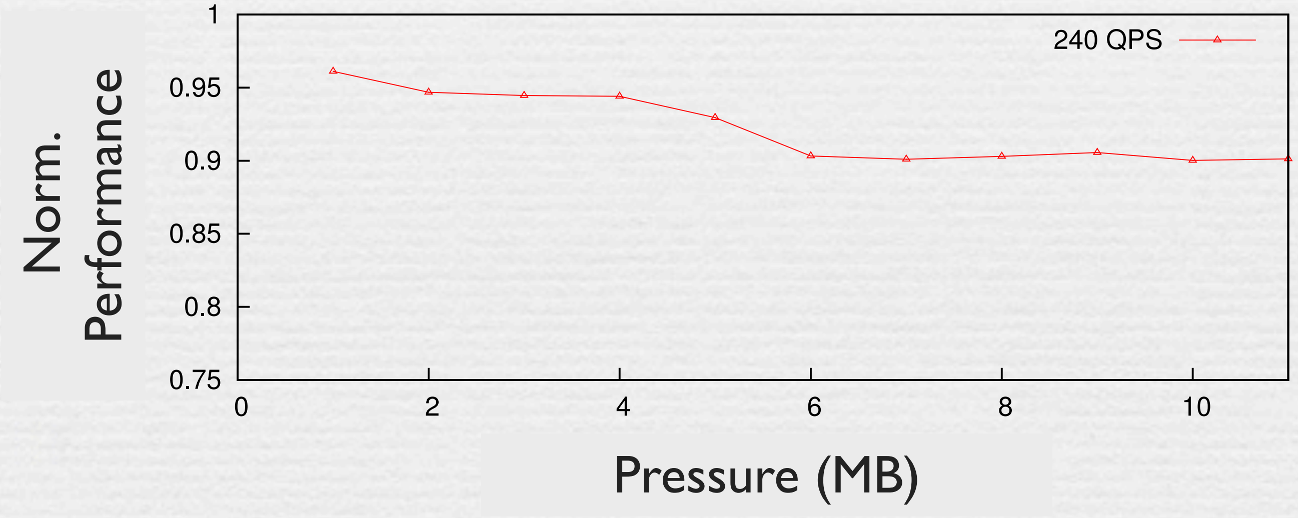


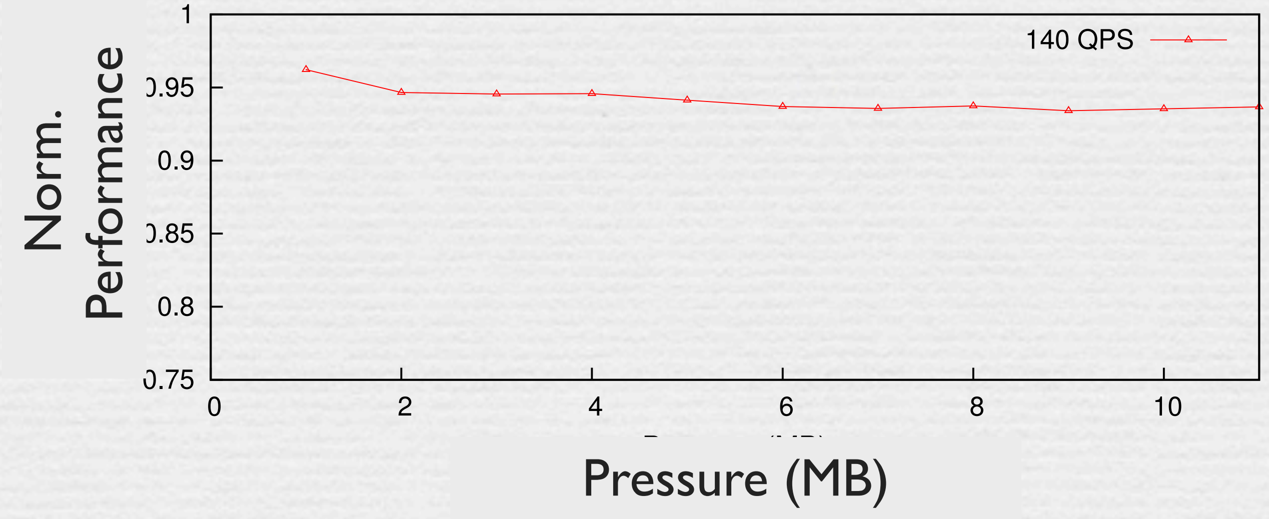
- Without a priori knowledge, the Flux Engine achieves accurate QoS control while gaining utilization

Adapt to load changes (L2)

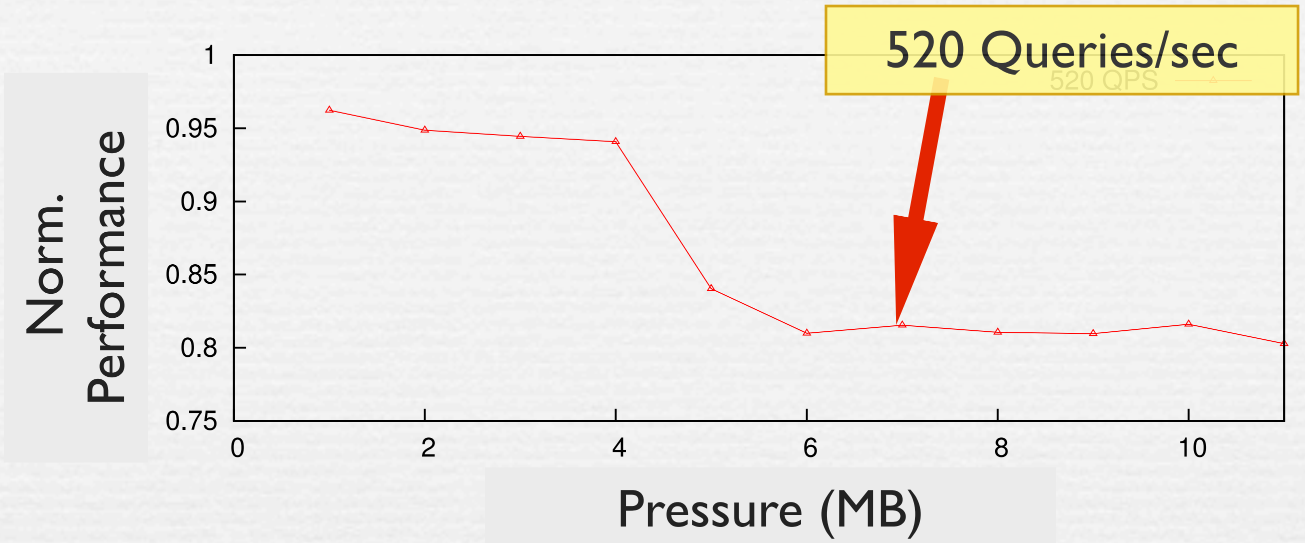
- Generating instantaneous sensitivity curves using Dynamic Bubble

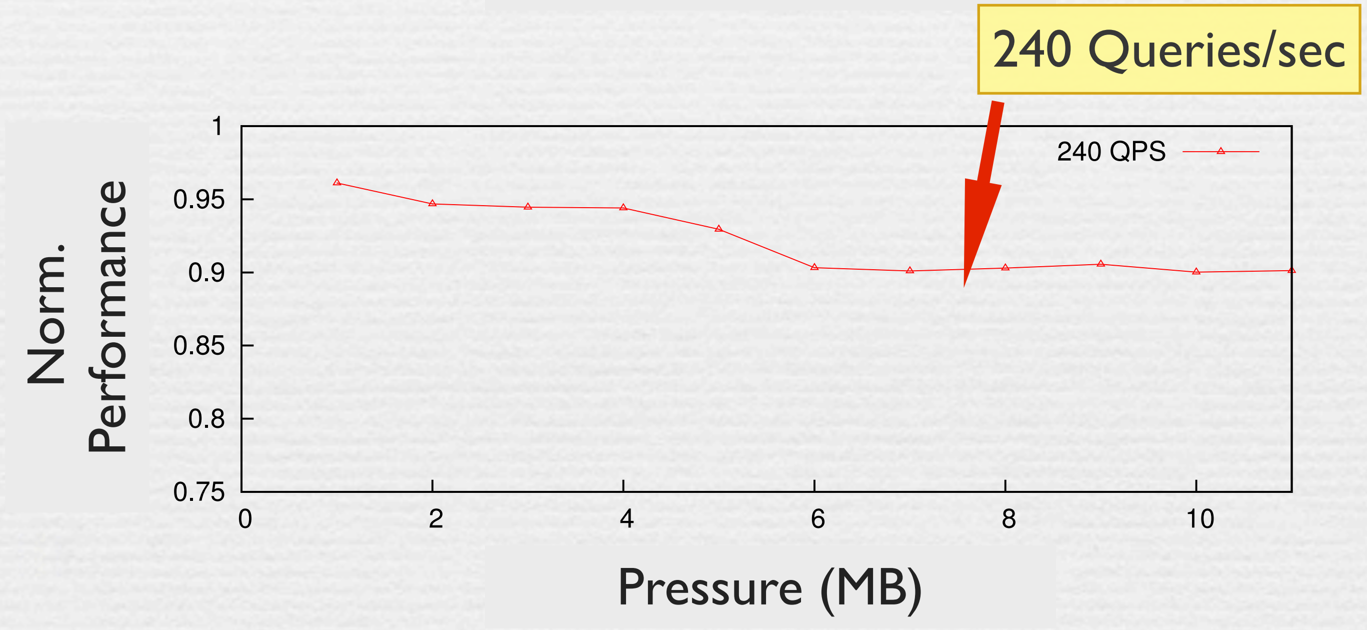


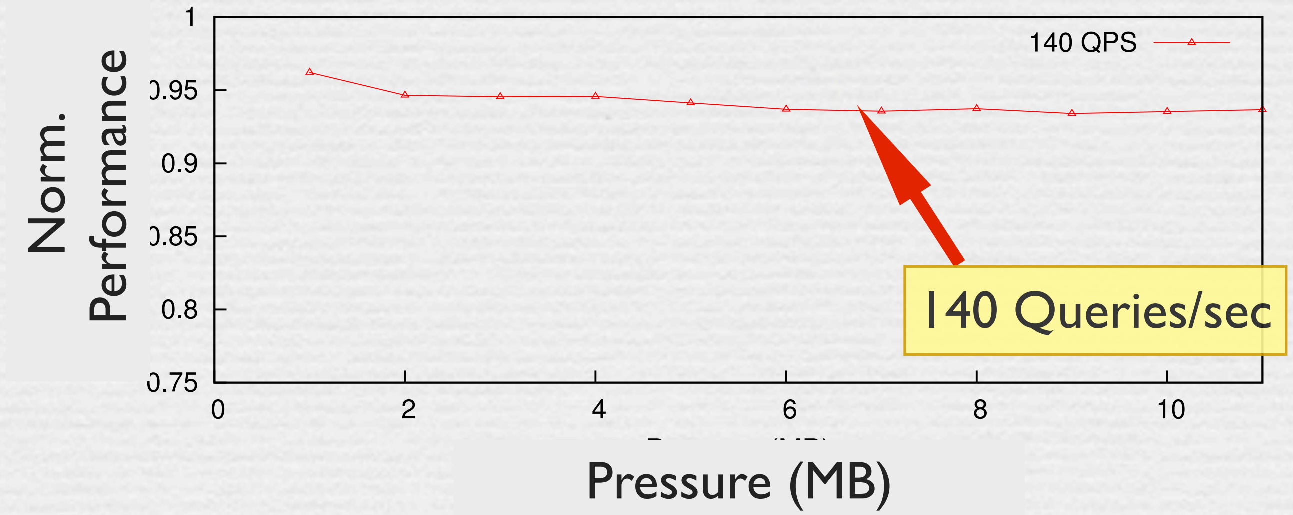




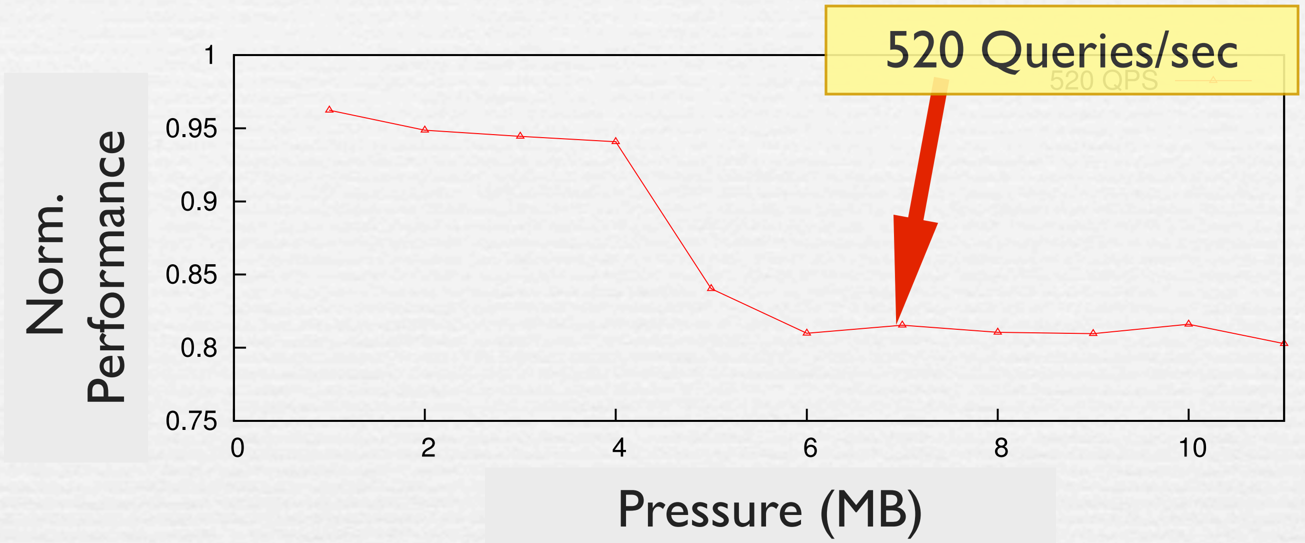
Adapt to load changes (L2)
- Generating instantaneous sensitivity curves using Dynamic Bubble

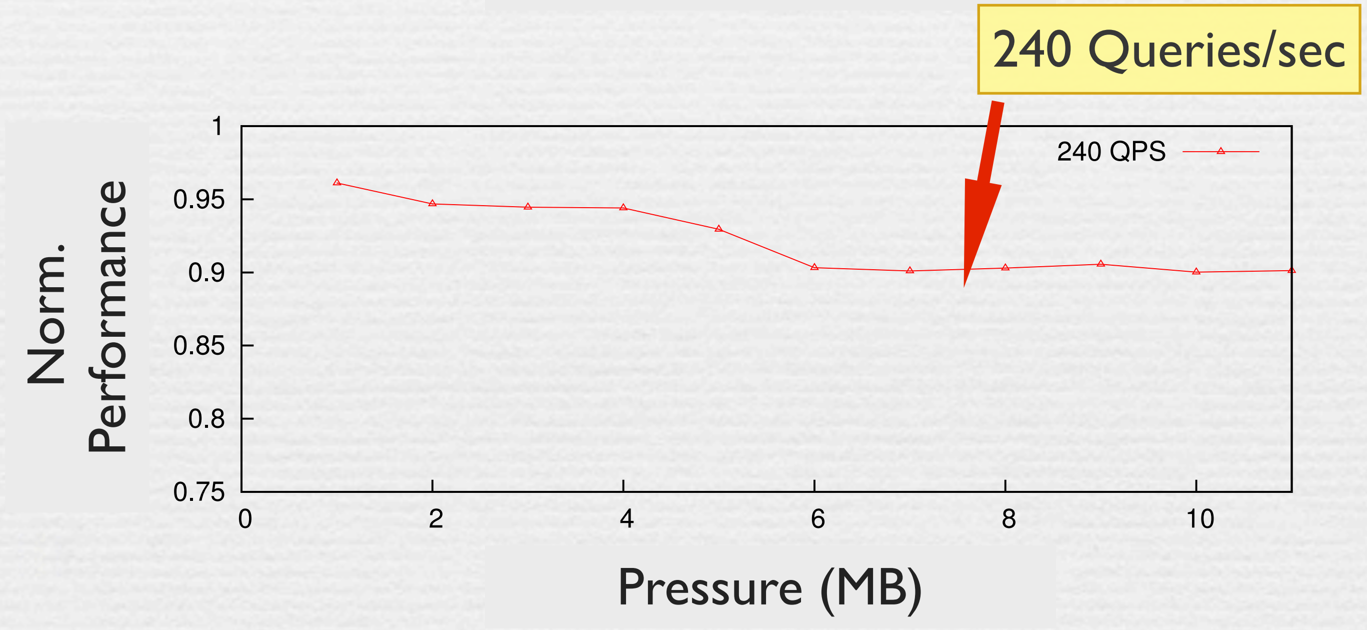


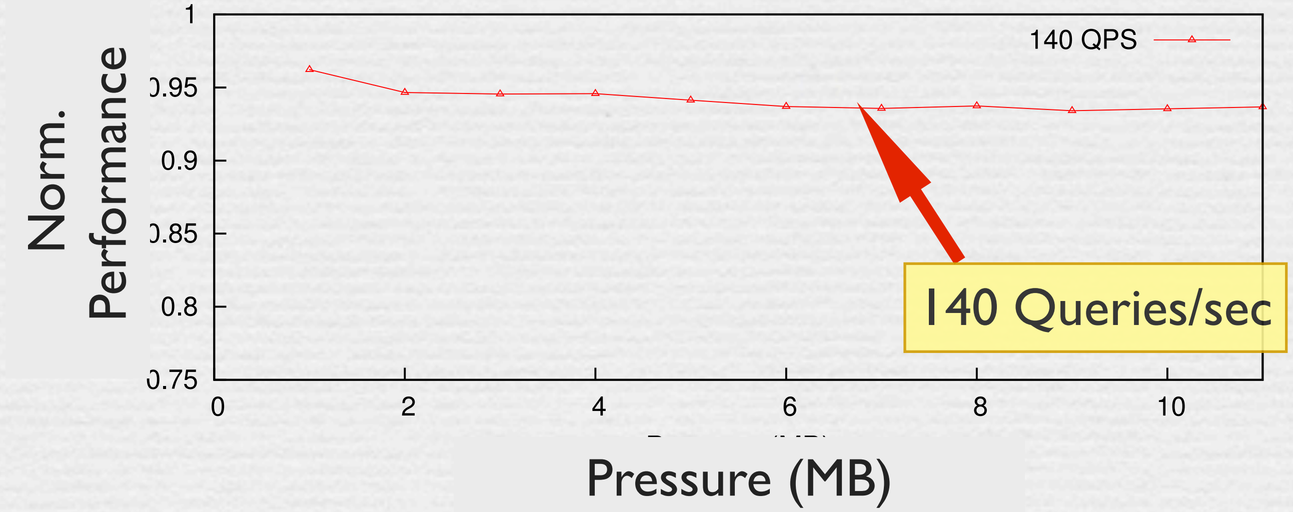




Adapt to load changes (L2)
- Generating instantaneous sensitivity curves using Dynamic Bubble

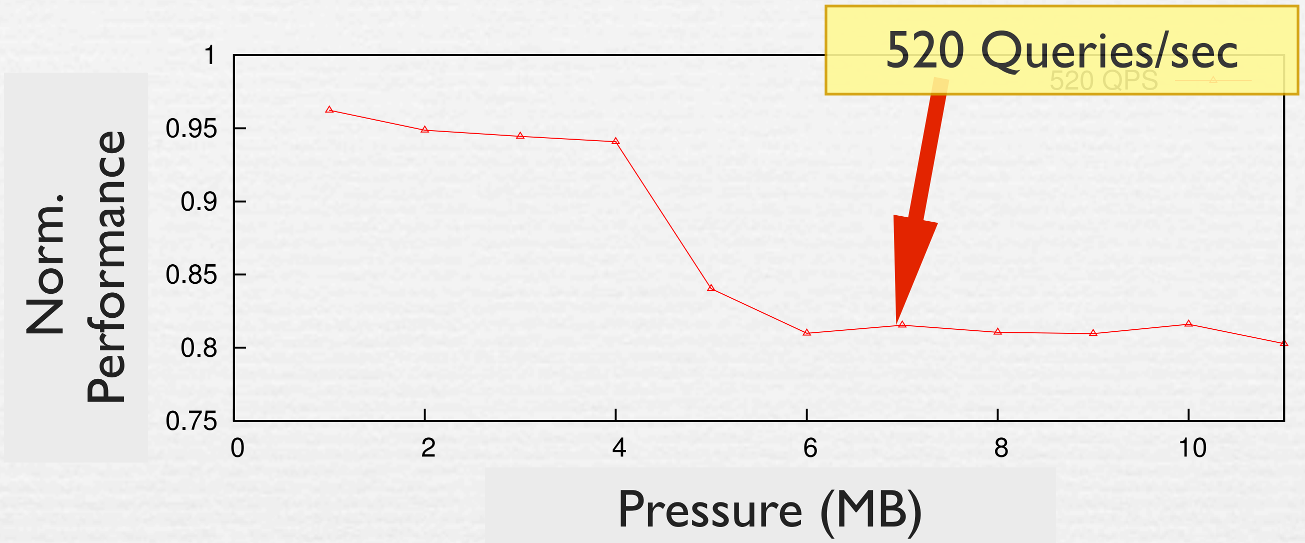

520 Queries/sec


240 Queries/sec

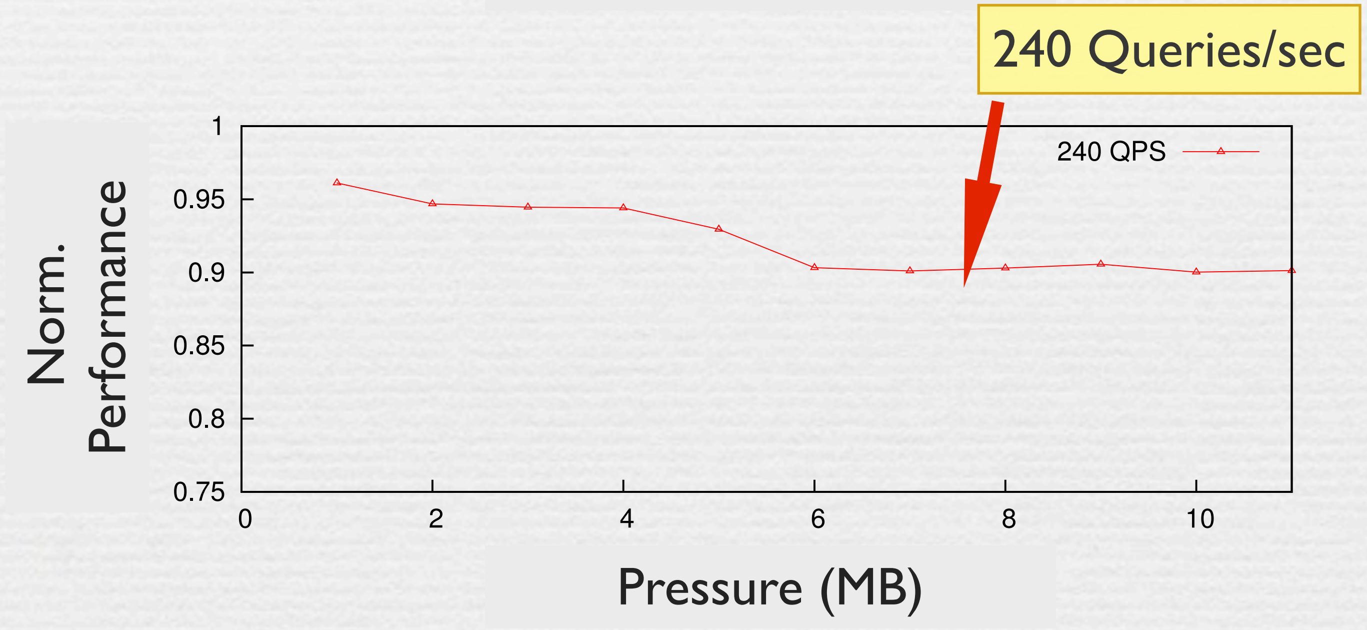

140 Queries/sec

- Dynamic bubble captures instantaneous sensitivity curves

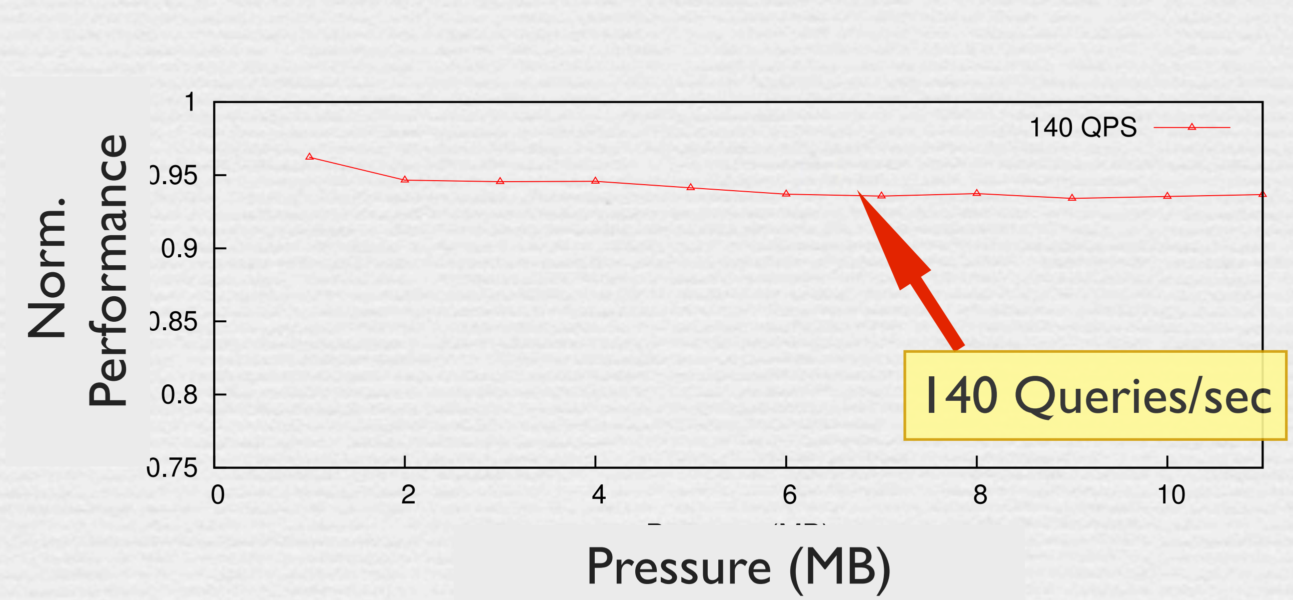
Adapt to load changes (L2)
- Generating instantaneous sensitivity curves using Dynamic Bubble



520 Queries/sec



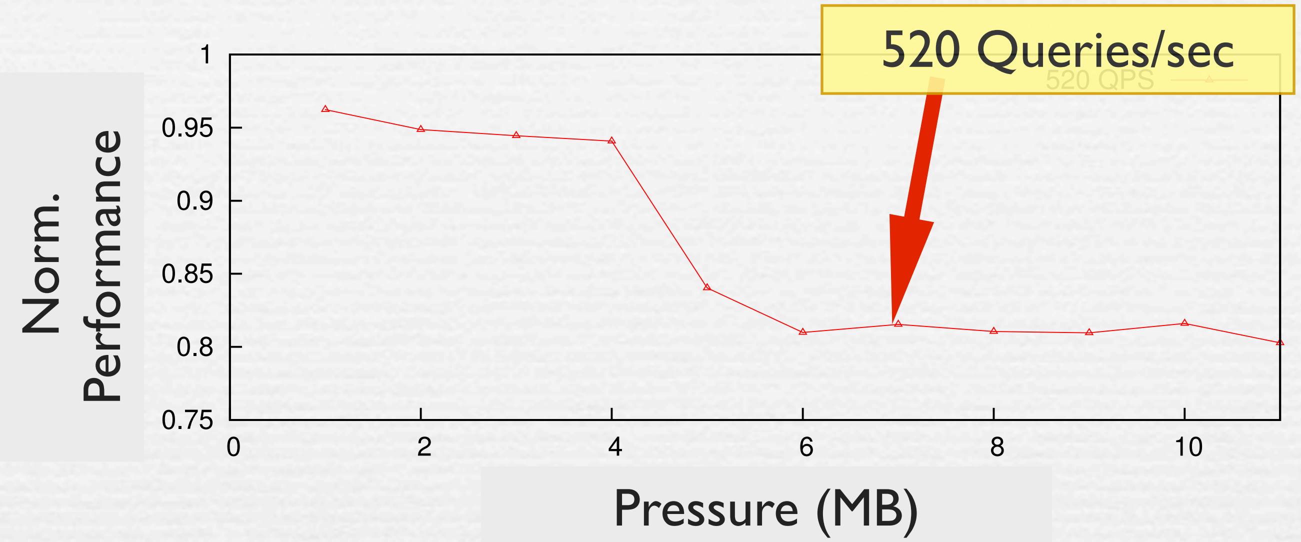
240 Queries/sec



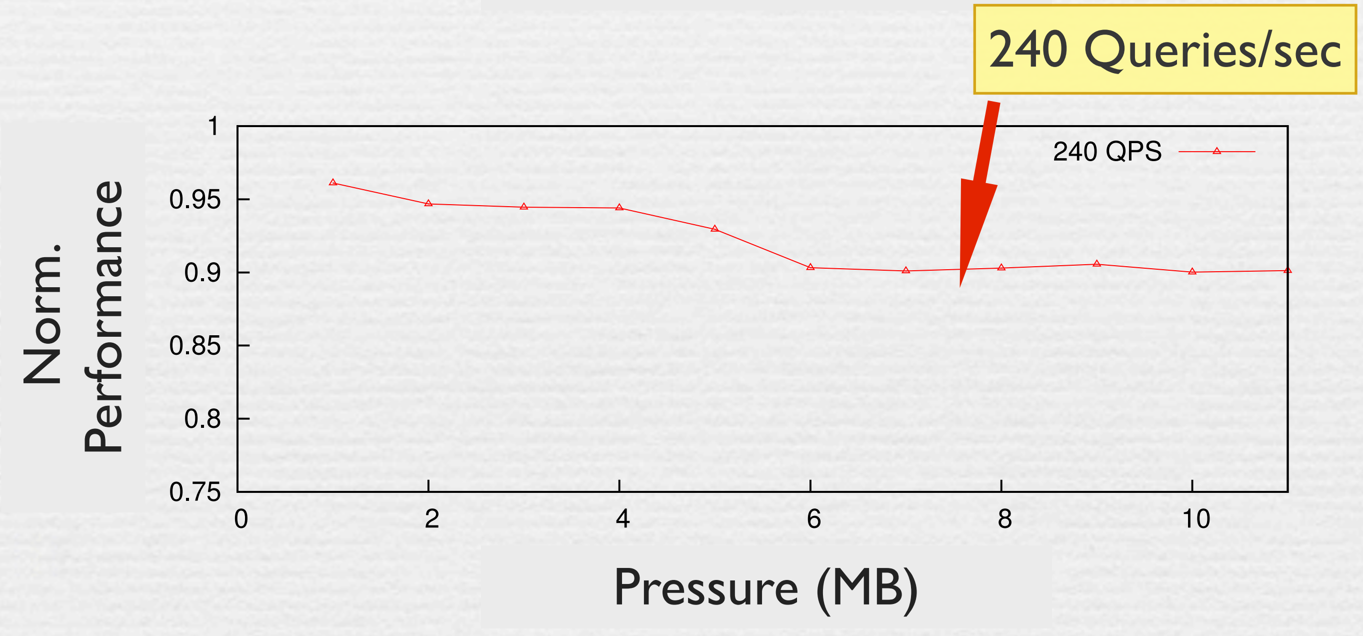
140 Queries/sec

- Dynamic bubble captures instantaneous sensitivity curves
- QoS is less sensitive to pressure on the shared resources when the load is low

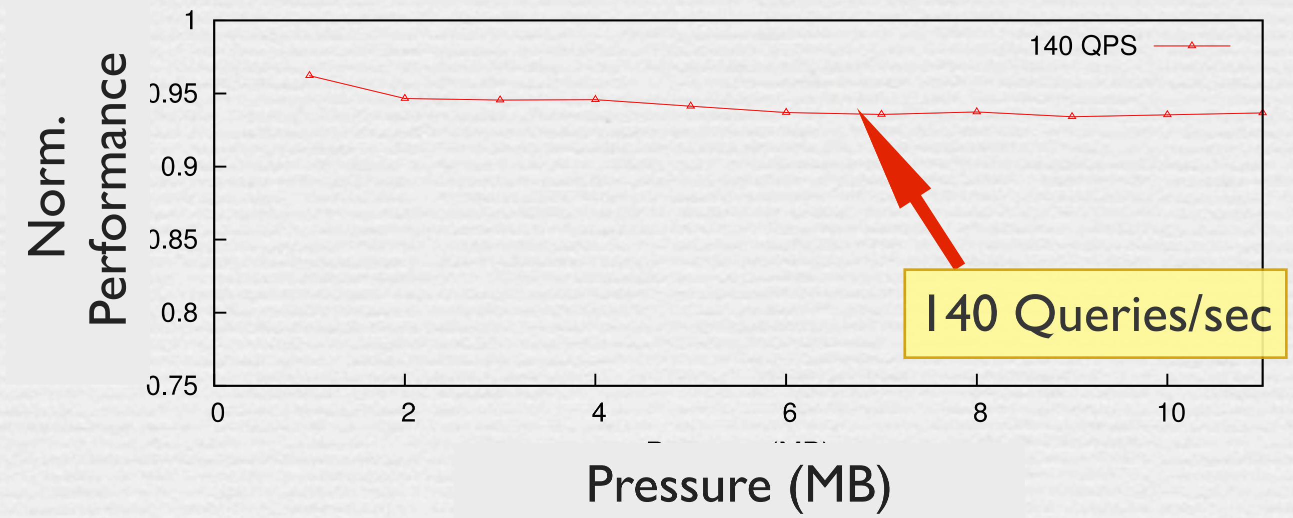
Adapt to load changes (L2)
- Generating instantaneous sensitivity curves using Dynamic Bubble



520 Queries/sec



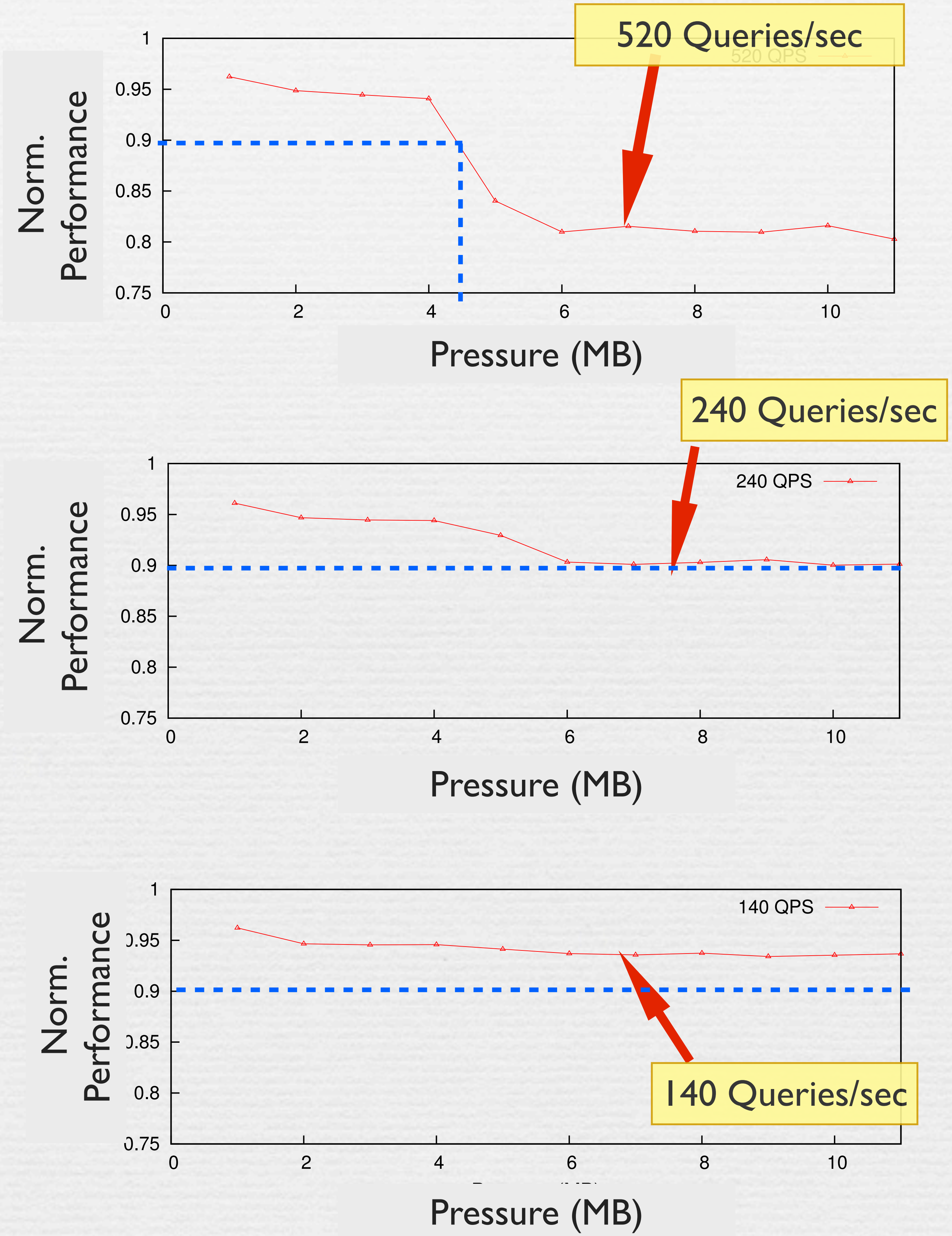
240 Queries/sec



140 Queries/sec

- Dynamic bubble captures instantaneous sensitivity curves
- QoS is less sensitive to pressure on the shared resources when the load is low
- More scheduling opportunities for low-load

Adapt to load changes (L2)
- Generating instantaneous sensitivity curves using Dynamic Bubble

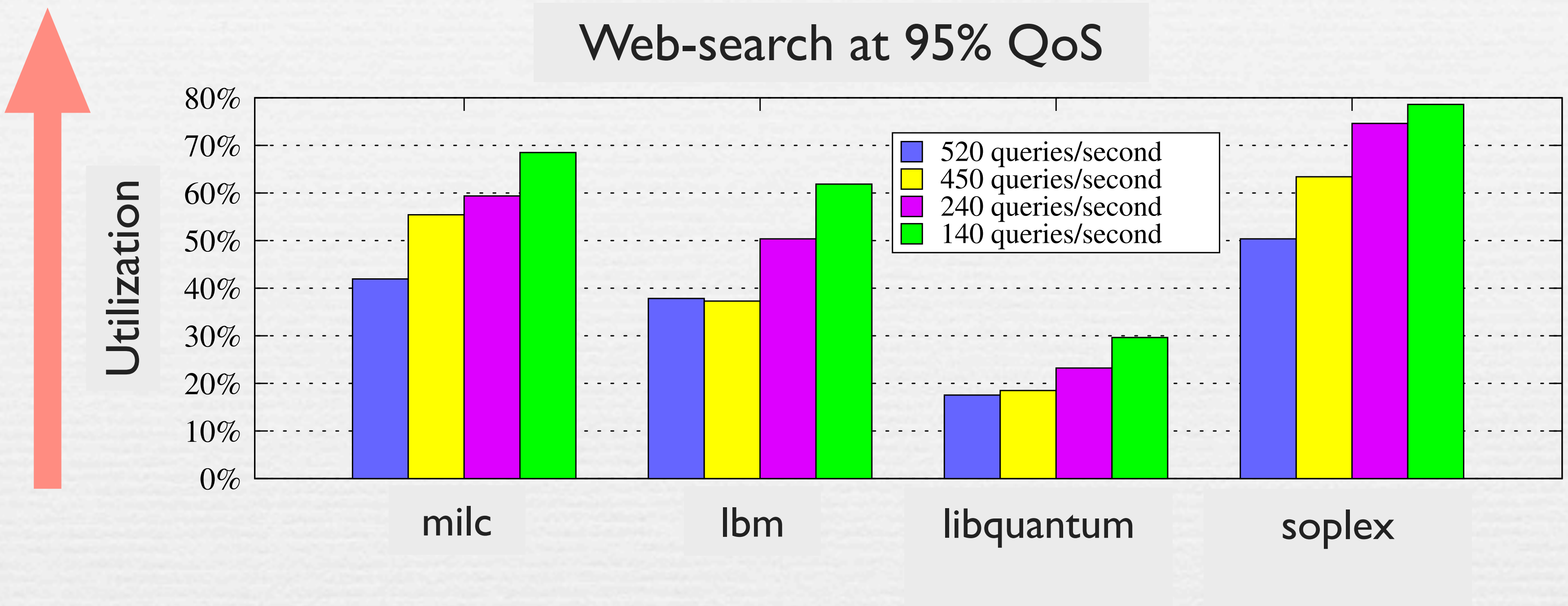

520 Queries/sec

240 Queries/sec

140 Queries/sec

- Dynamic bubble captures instantaneous sensitivity curves
- QoS is less sensitive to pressure on the shared resources when the load is low
- More scheduling opportunities for low-load

Flux: Adapt to Load Fluctuation (L2)

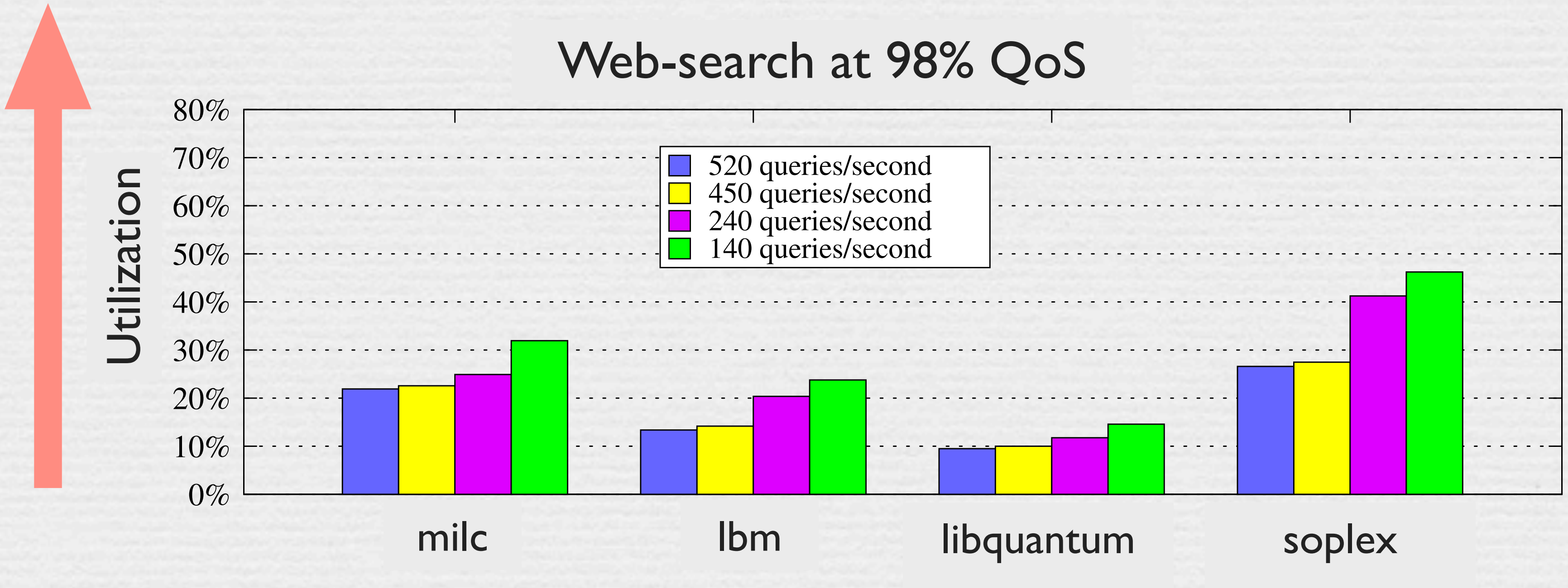


🟦 520 QPS

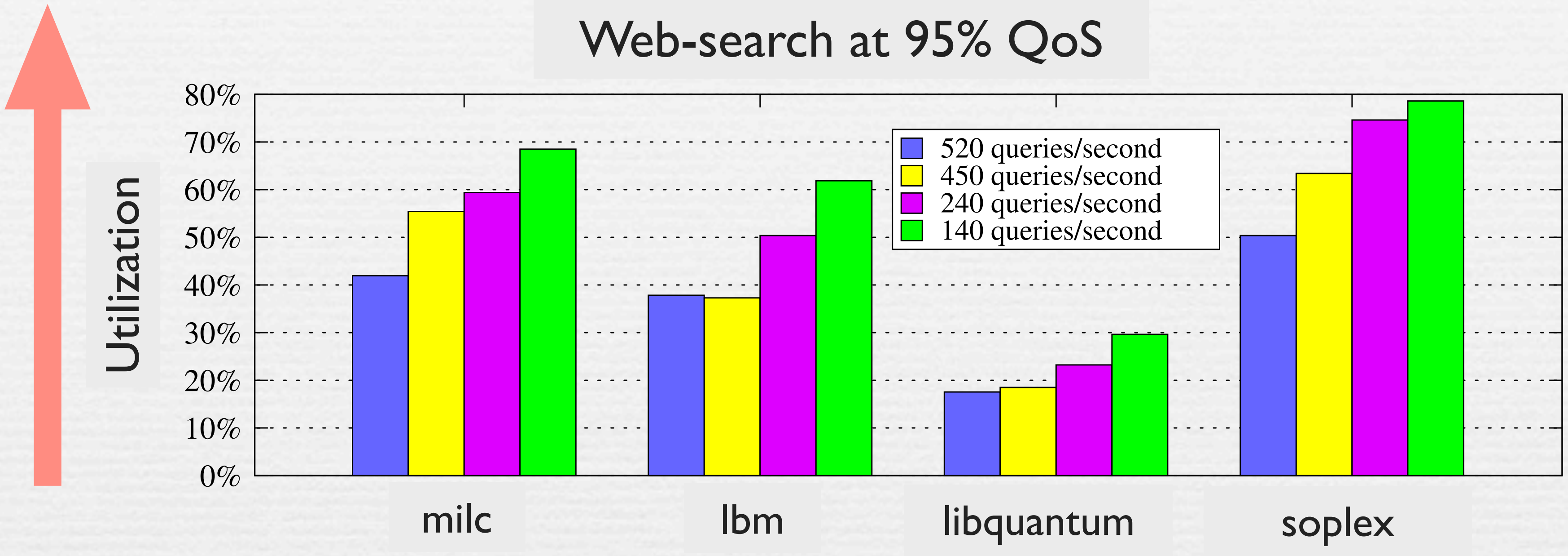
🟨 450 QPS

🟪 240 QPS

🟩 140 QPS



Flux: Adapt to Load Fluctuation (L2)



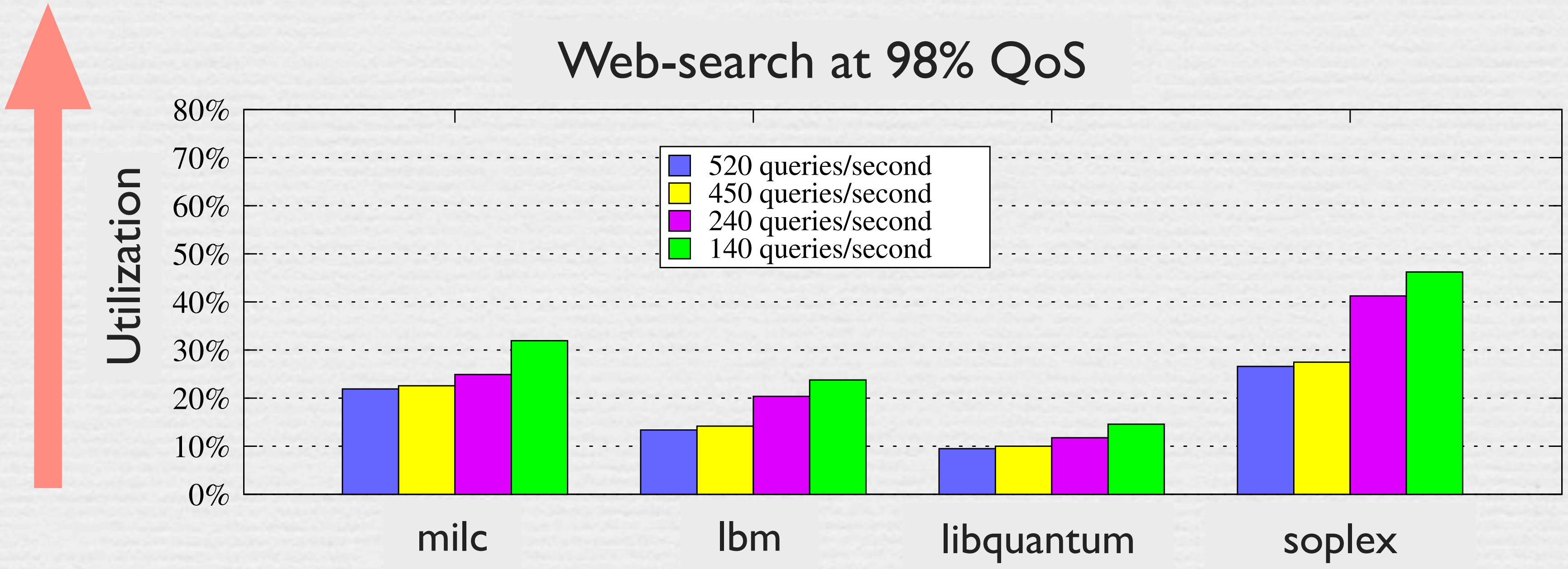
Web-search at 95% QoS

🟦 520 QPS

🟨 450 QPS

🟪 240 QPS

🟩 140 QPS



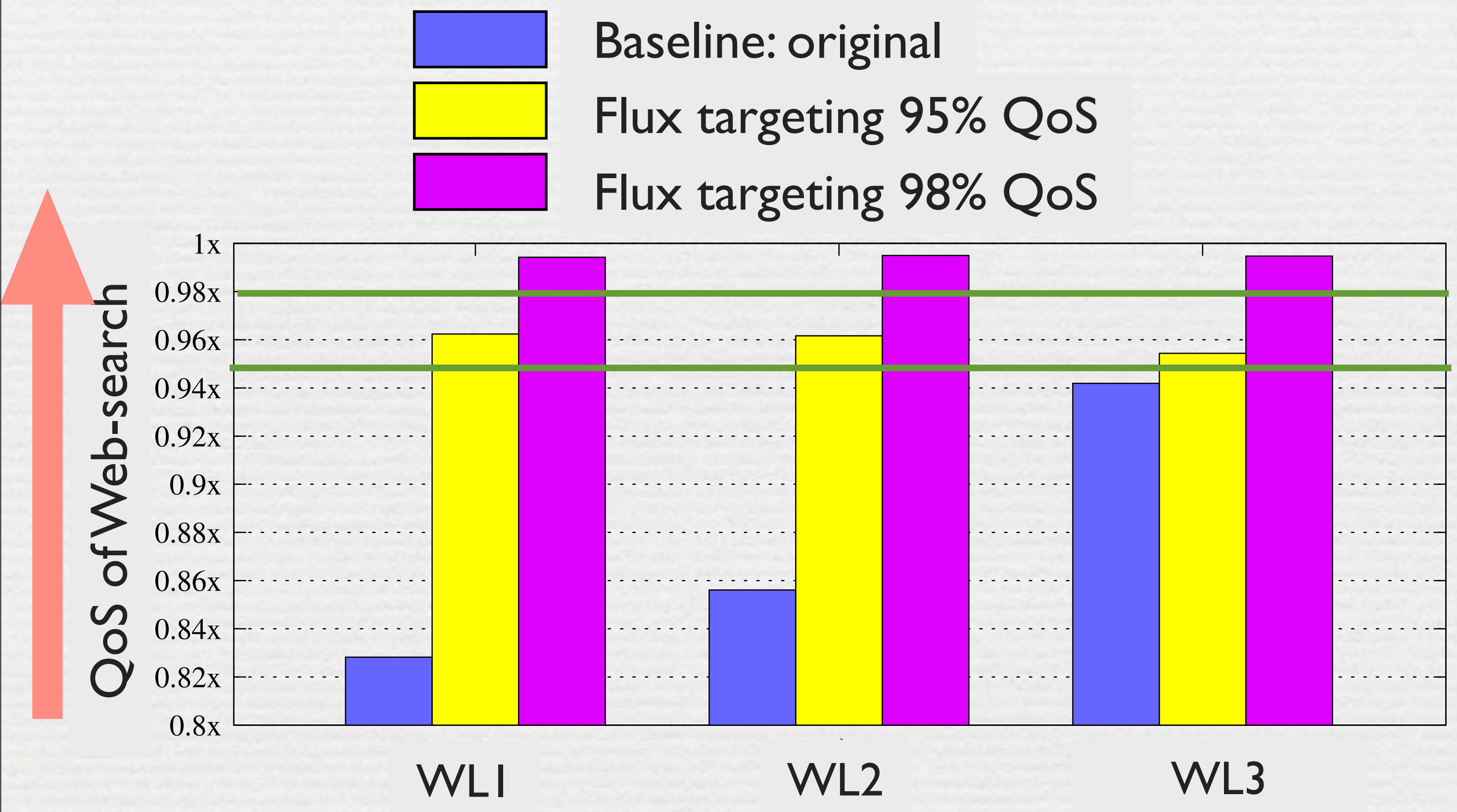
Web-search at 98% QoS

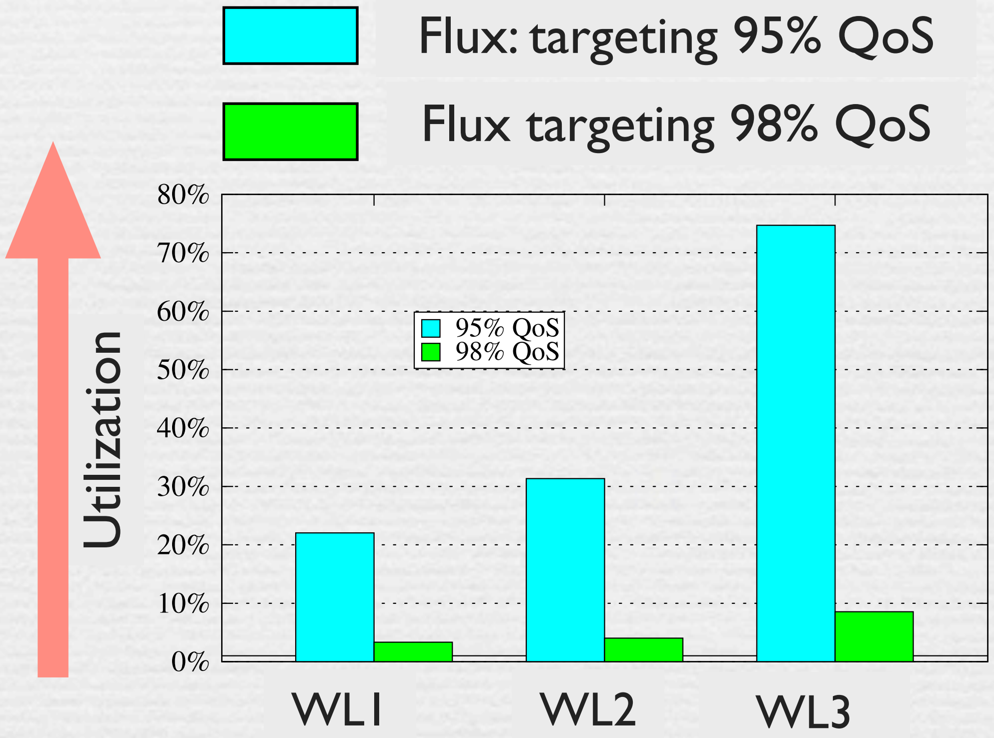
- The Flux Engine achieves higher utilization during low load period

Scale Beyond Pairwise (L3)

Workloads

WL1	lbm, lbm, libquantum, libquantum
WL2	lbm, libquantum,soplex, milc
WL3	mcf, mcf, sphinx, soplex

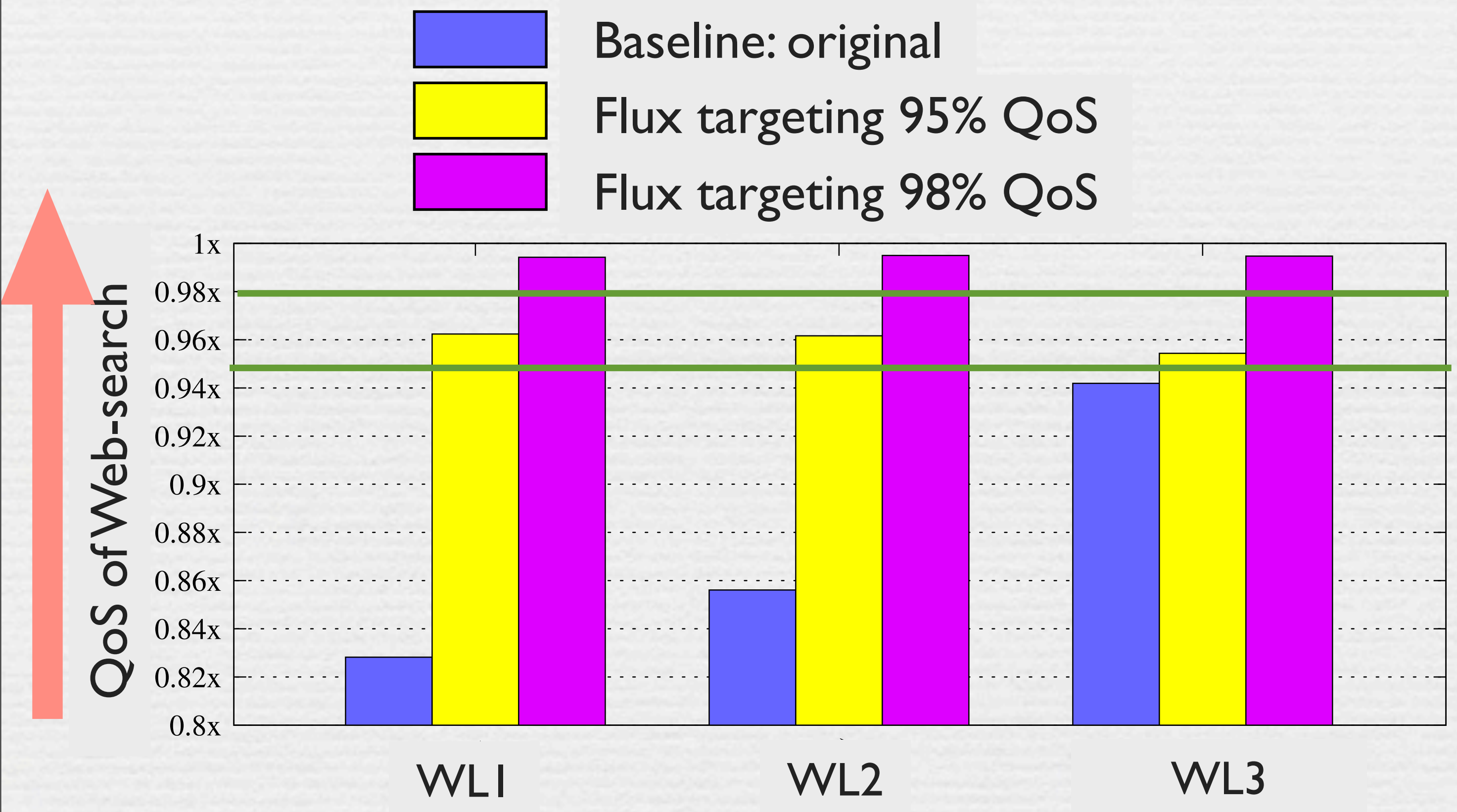


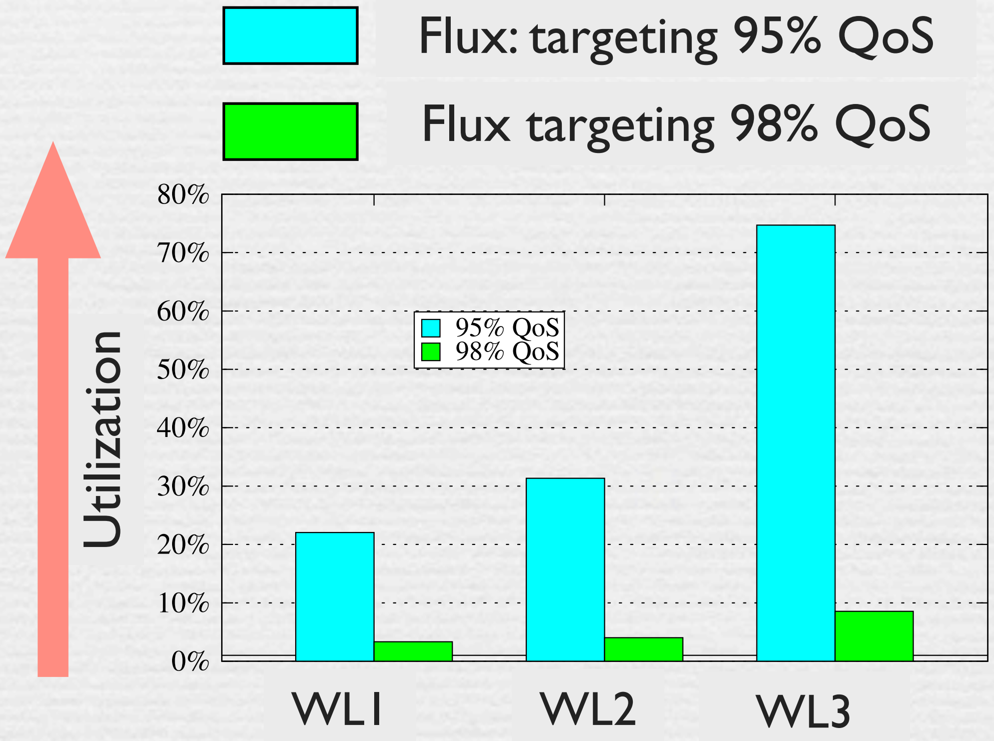


Scale Beyond Pairwise (L3)

Workloads

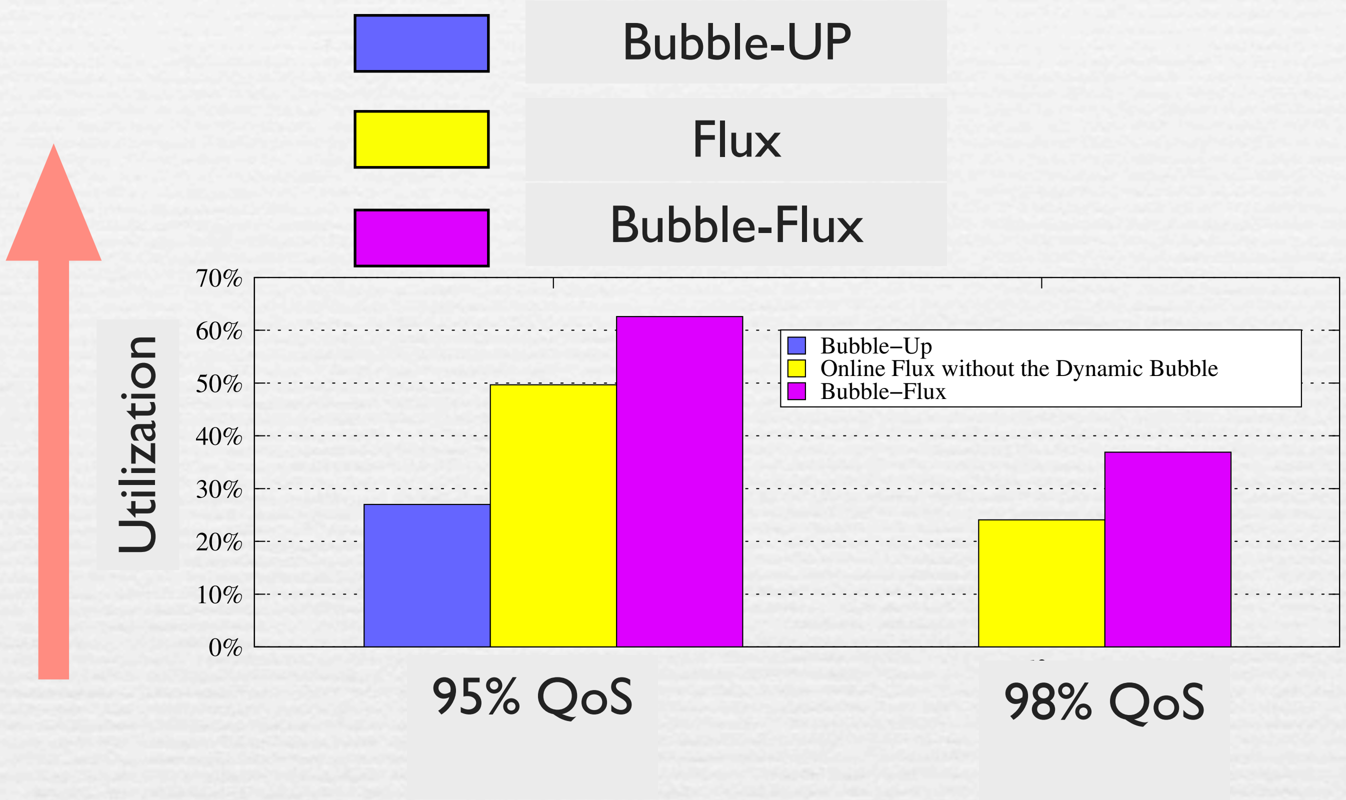
WL1	lbm, lbm, libquantum, libquantum
WL2	lbm, libquantum,soplex, milc
WL3	mcf, mcf, sphinx, soplex





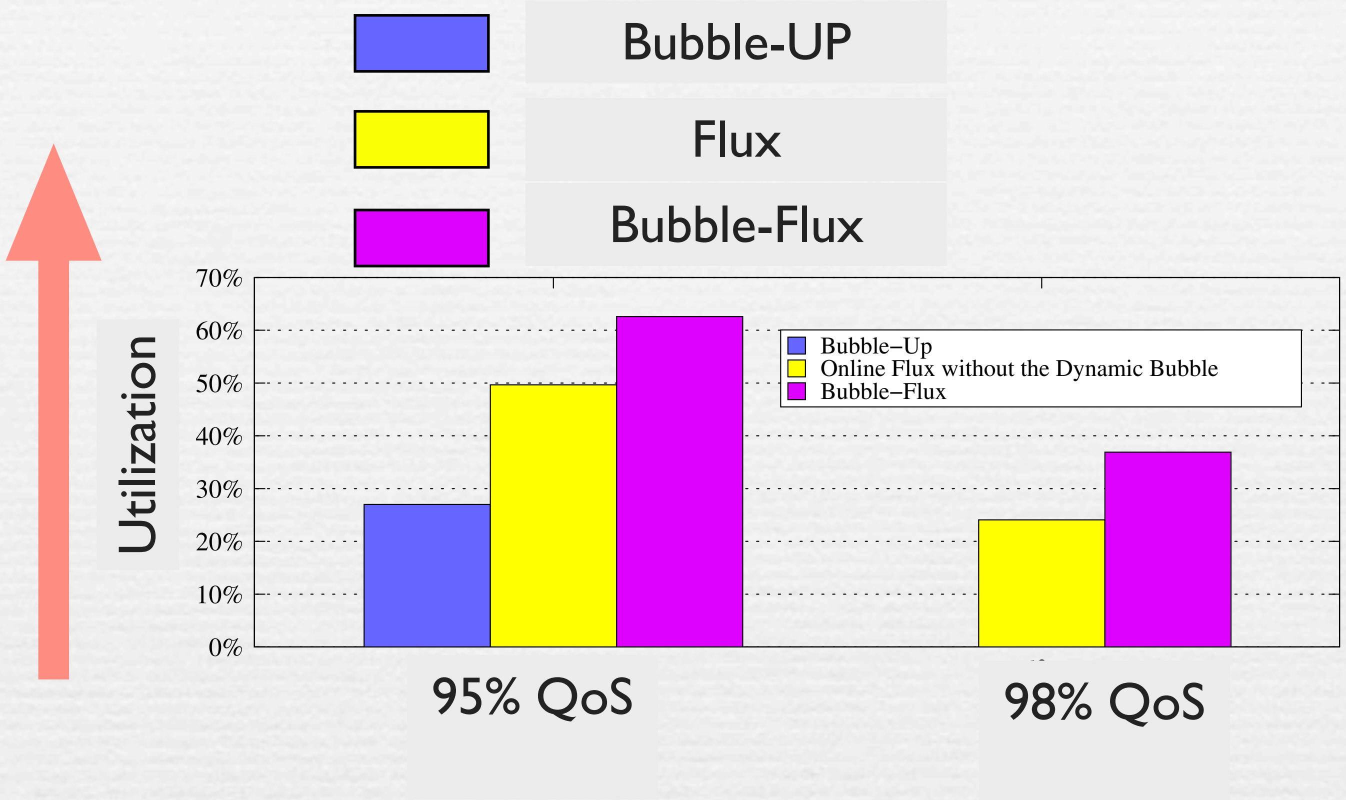
- The Flux Engine can manages more than 2 various co-runners

Put all together: Apply Bubble-Flux in a WSC

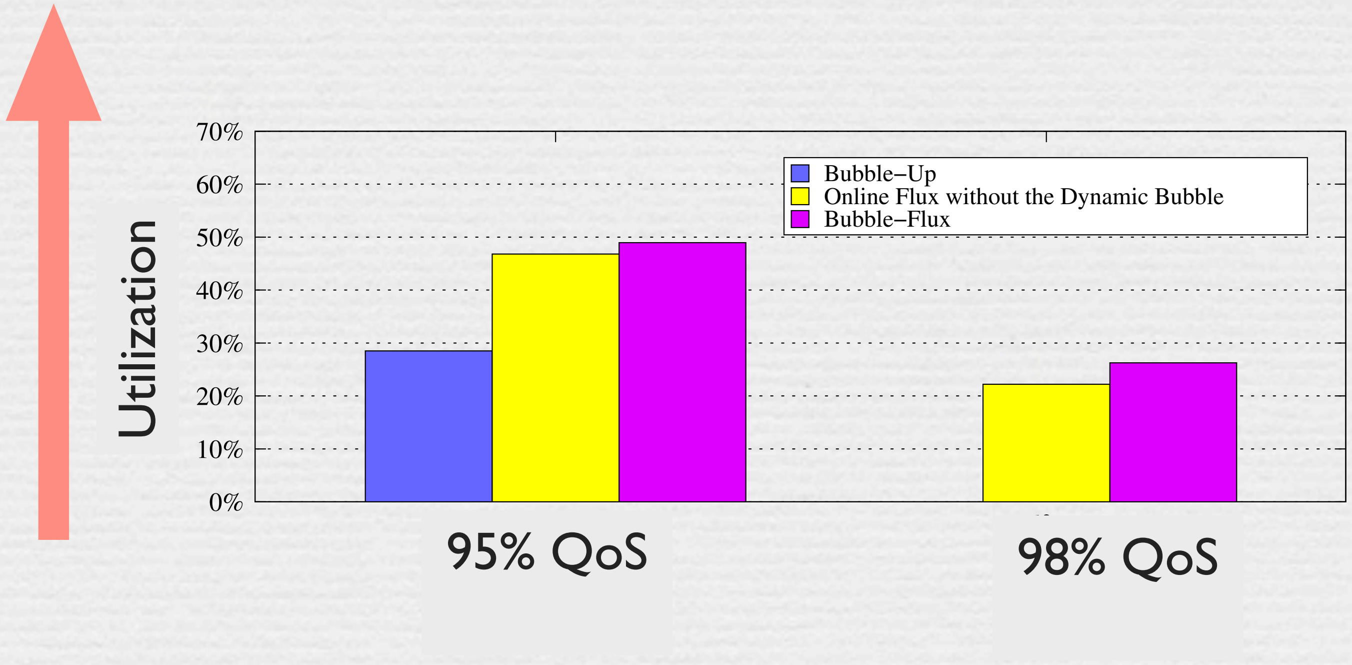


- **Scenario 1**
- 1000 machines (500 **Web-search**,500 **Data-serving**).
- Before mapping: LS on 4 cores, 4 cores idle.
- To map: batch workloads, composed of 1000 mixed applications of 7 types

Put all together: Apply Bubble-Flux in a WSC

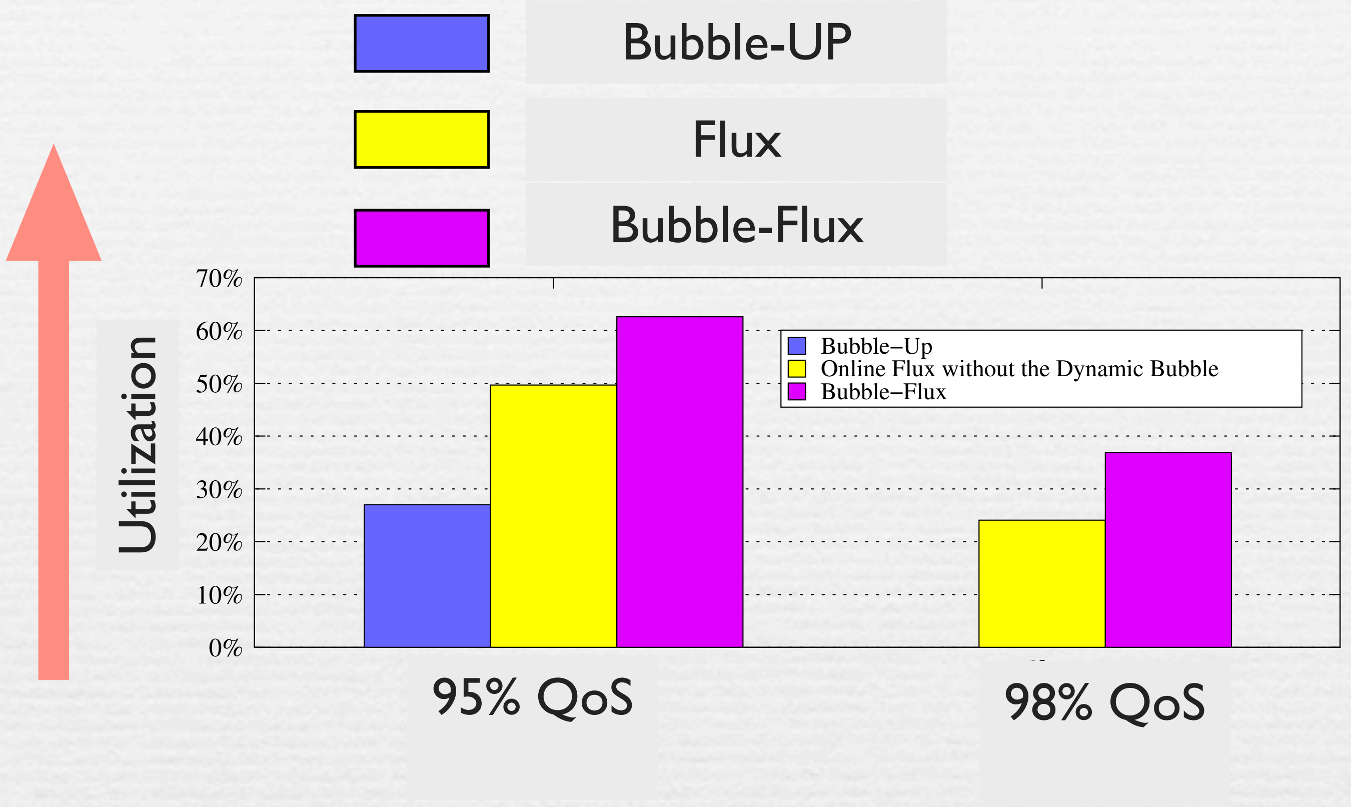


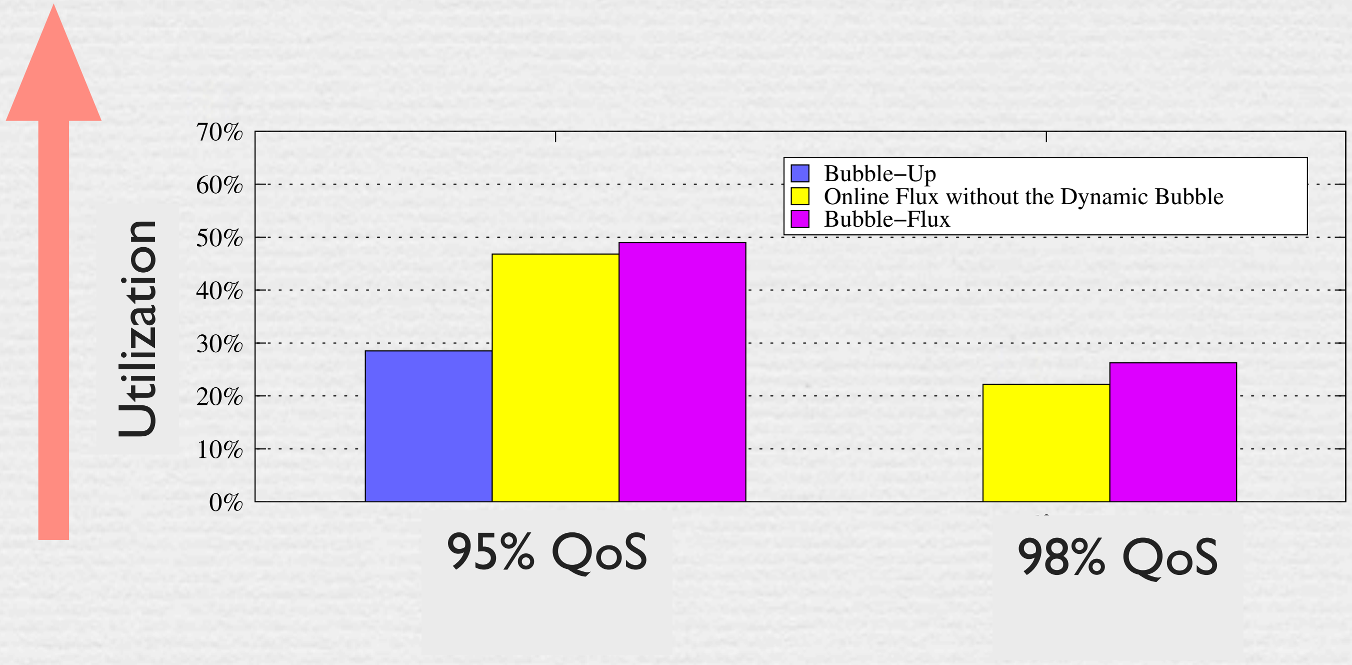
- **Scenario 1**
- 1000 machines (500 **Web-search**,500 **Data-serving**).
- Before mapping: LS on 4 cores, 4 cores idle.
- To map: batch workloads, composed of 1000 mixed applications of 7 types



- **Scenario 2**
- 1500 machines (500 **Web-search**, 500 **Data-serving**, 500 **Media-streaming**).
- Before mapping: LS on 4 cores, 4 cores idle.
- To map: batch workloads, composed of 1500 mixed applications of 7 types

Put all together: Apply Bubble-Flux in a WSC





- Bubble-Flux up to 2.2x better than Bubble-Up (62% vs. 27% utilization).
- Significant utilization when Bubble-Up fails to utilize any idle cores (24% vs. 0% utilization)
- Importance of combining prediction-based cluster-level mapping and precise server-level QoS management

Conclusion

- Bubble-Flux
  - Dynamic Bubble + Flux Engine
  - Ensure QoS while maximizing utilization
- Address three critical limitations of Bubble-Up
- Importance of combining prediction-based cluster-level mapping and server-level QoS management

when
how
who
why
which
where
what
how
when
which
who
what
why
when
which
how
what
who
how
when
how
who
which
when
how
which
what
why
how
how
which
when
which
how
which
who
why
when
what