METERG: Measurement-Based End-to-End Performance Estimation Technique in QoS-Capable Multiprocessors

Jae W. Lee and Krste Asanovic
{leejw, krste}@mit.edu

MIT Computer Science and Artificial Intelligence Lab
April 5, 2006

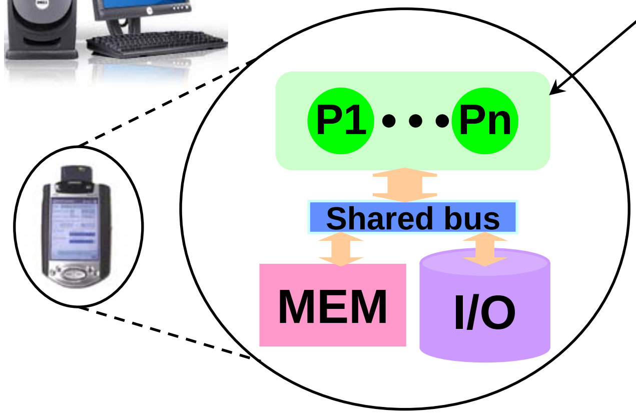
Multiprocessors Are Ubiquitous

Servers

Desktops

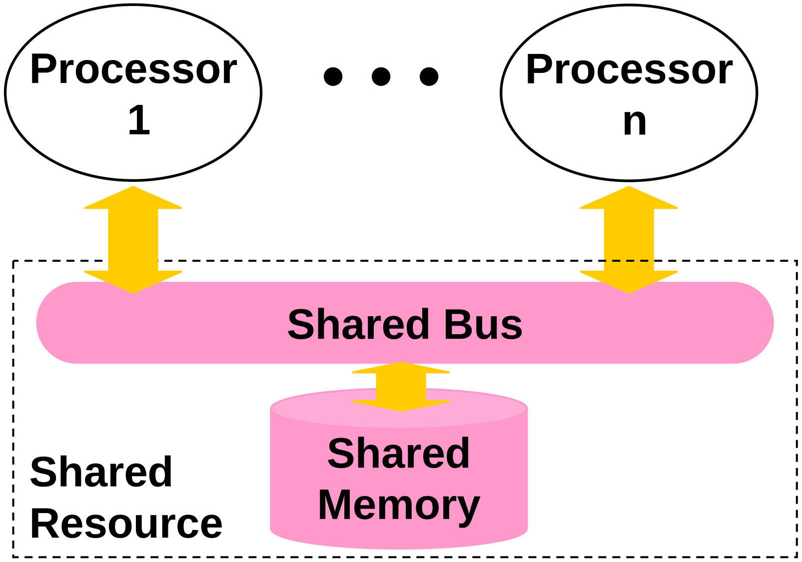
Handheld/
Embedded

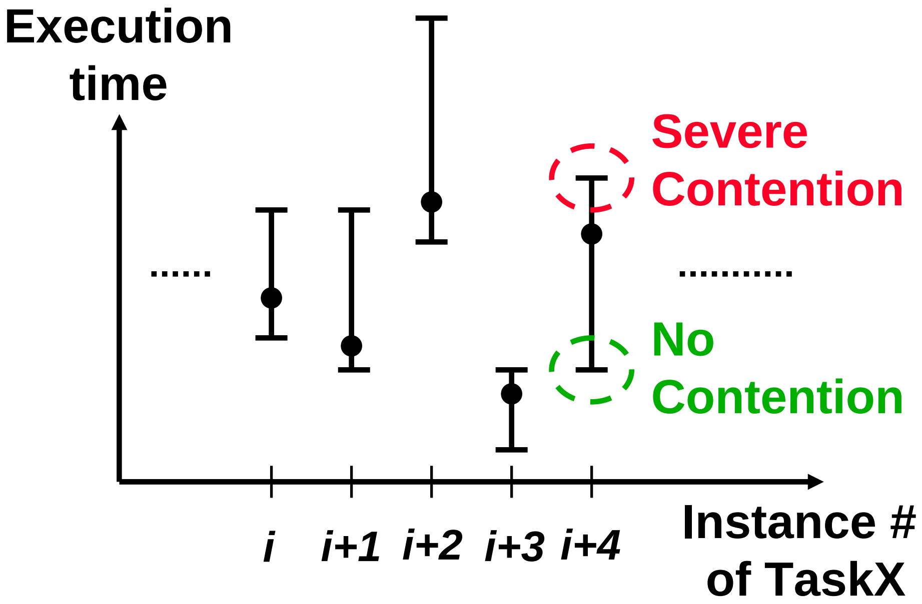
Embedded CMP
(e.g. ARM MPCore)



- Our focus: Running soft real-time apps (e.g. media codecs, web services) on general-purpose MPs
  - Assuming multiprogrammed workloads: real-time + best-effort apps
- Problem: Contention for shared resources causes wider variation of a task's execution time.

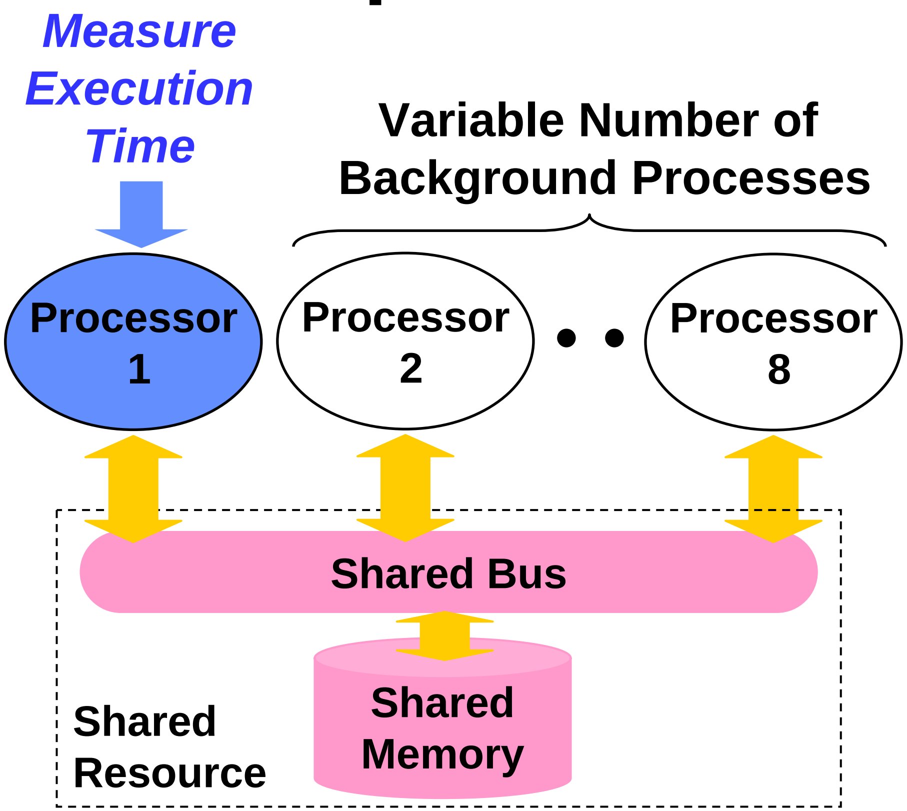
Execution Time Variation in MPs

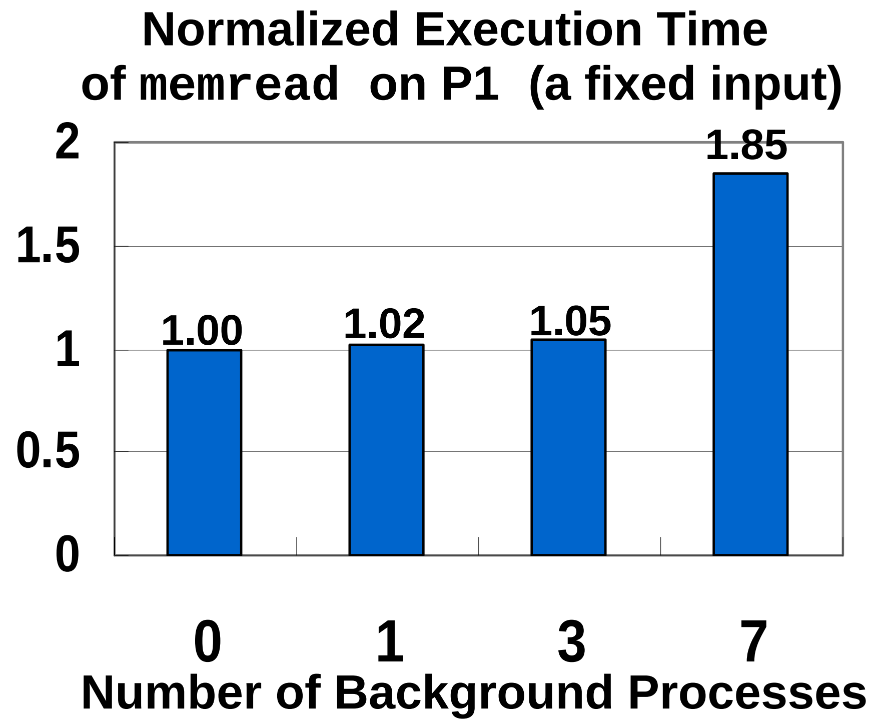




- For fixed input to fixed task, execution time varies depending on the activities of other processors because of shared memory system.

Impact of Resource Contention

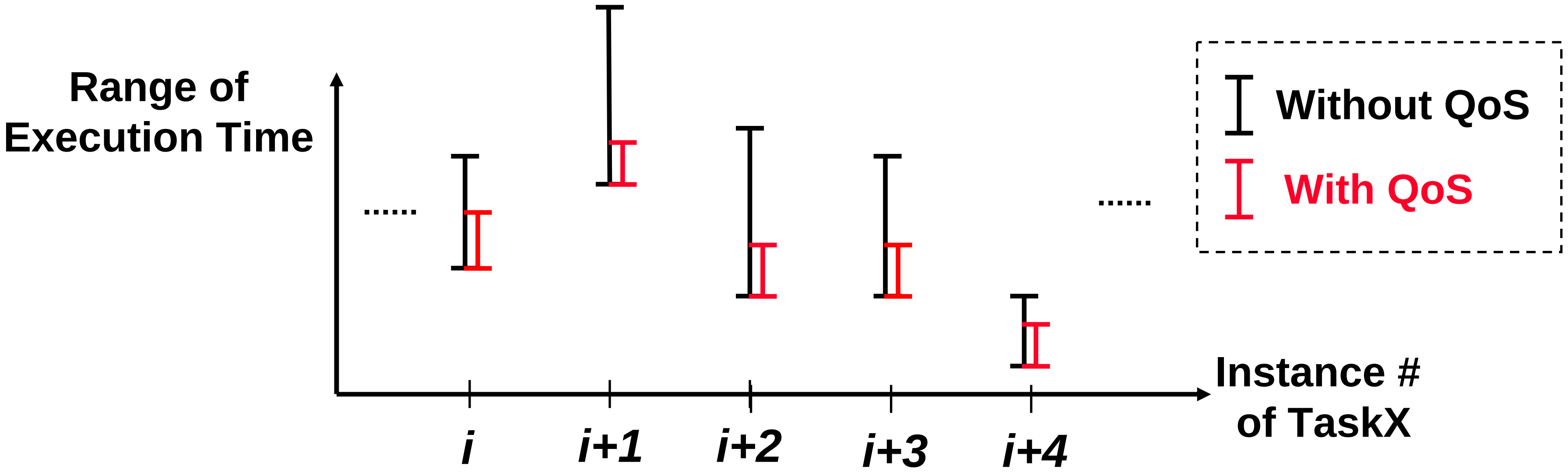




- Execution time increases by 85 % when number of background processes increases from 0 to 7.
- How can we limit the impact of resource contention?

  → QoS Support

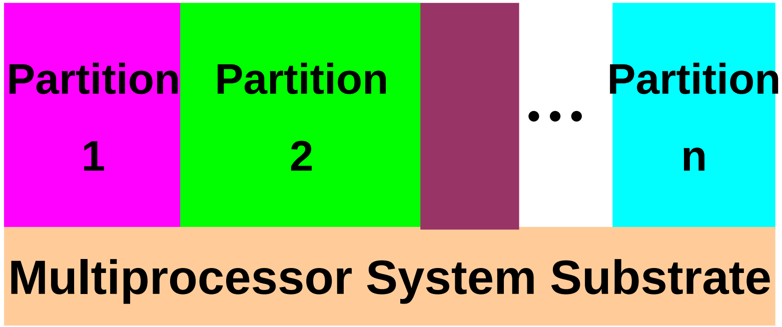
Memory QoS Reduces MP Execution Time Variation



- Memory QoS guarantees per-processor memory bandwidth and latency.
  – It guarantees minimum performance and distributes unclaimed bandwidth to sharers in order to maximize throughput.
- Challenge: How can we find the minimal QoS parameters that meet a RT task’s given performance goal?
  – Minimal reservation improves total system throughput.

Translating Performance Goal in User Metric into QoS Parameters

- User's performance goal (user metric) should be translated into QoS parameters (system metric).
  - User metric: Execution time, Transactions per Second (TPS), Frames per Second (FPS)
  - System metric: Memory bandwidth, Latency
- Example: MP server virtualization
  - Question: What guarantees can you make to users?
  - Users prefer **user metric** (e.g. TPS) to **system metric** (e.g. Memory bandwidth).



→ ***Minimal QoS parameters are important to maximize system throughput.***

Finding Minimal Resource Reservation (1): Analysis

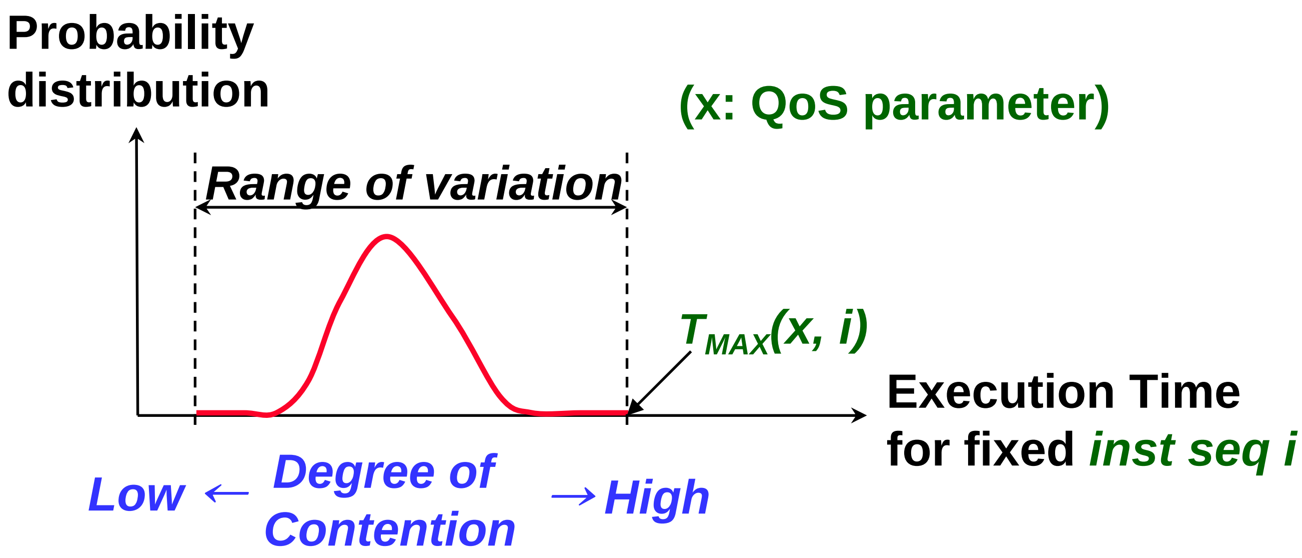
- Static timing analyses: Tools for WCET analyses
  - Consist of Hardware modeling + program analysis
  - [**+**] Can find minimal resource reservation for all possible inputs
  - [**-**] Analysis cost is prohibitive.
    - » Becoming harder for increasing hardware/software complexity in MPs
    - » Possibly overkill for soft RT apps:

      Our primary concern is performance degradation from resource contention (not from input data).

      We can tolerate a small number of deadline violations to maximize overall system throughput.

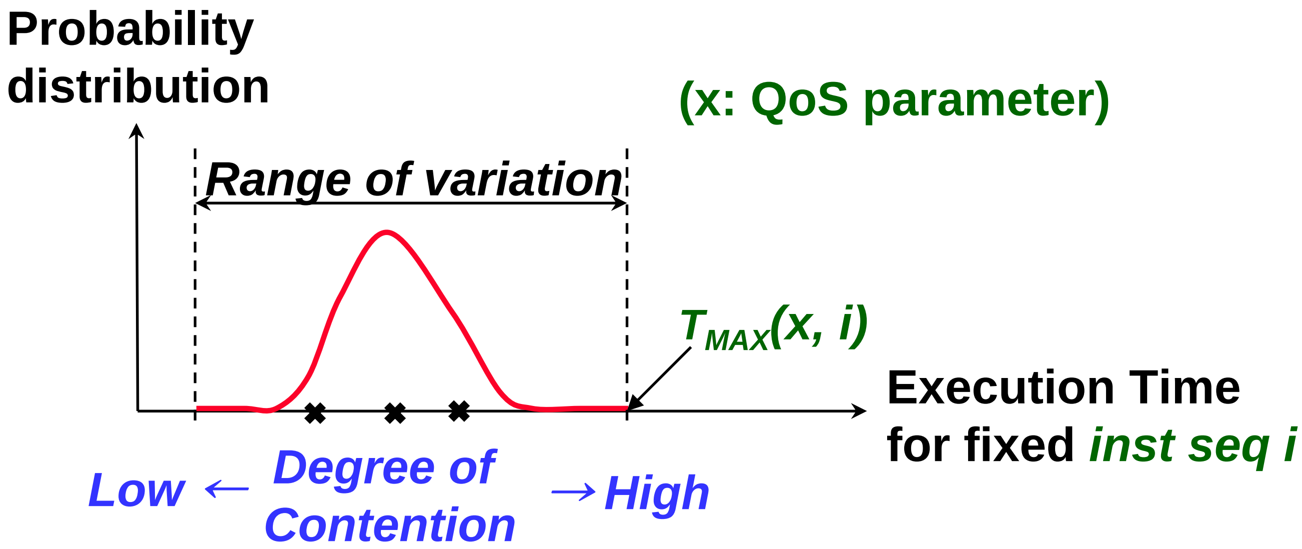
Finding Minimal Resource Reservation (2): Measurement

- Measurement-based Approach
  - Measures task's execution time for a certain input set
  - Motto: "The best model for a system is the system itself." [Colin-RTSS '03]
  - Goal: To find execution time under worst-case contention (=$T_{MAX}(x, i)$) for given QoS parameter (***x***) and instruction sequence (***i***)



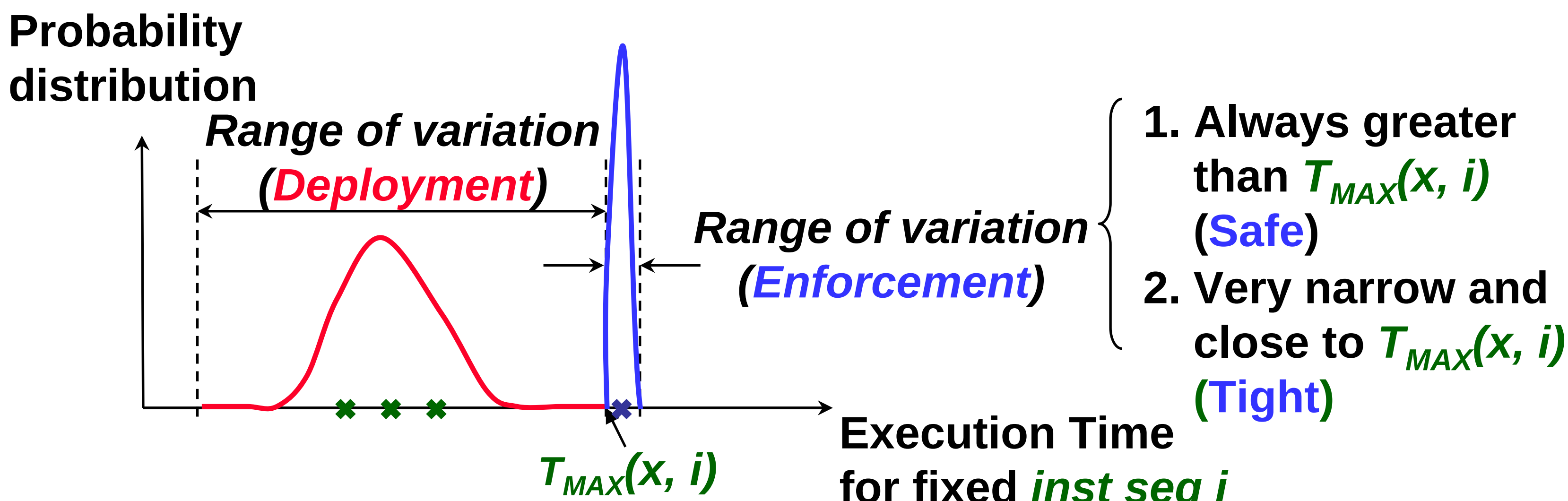
Finding Minimal Resource Reservation (2): Measurement

- Measurement-based Approach (Cont)
  - [**+**] Easy
  - [**-**] Not absolutely safe
    - → Can be practically useful for soft real-time apps
    - → Can be useful if we are given “near-worst” input set
  - [**-**] Measured time varies by degree of contention in runtime
    - → We will address this problem in the rest of this talk!



Our Approach: METERG

- METERG QoS System: Improving measurement-based approach
  - Estimates $T_{MAX}(x, i)$ easily by measurement
  - Provides hardware support for safe and tight estimation for $T_{MAX}(x, i)$
  - Introduces two QoS modes
    - » Deployment mode (operation mode)
    - » Enforcement mode (measurement mode for estimation)

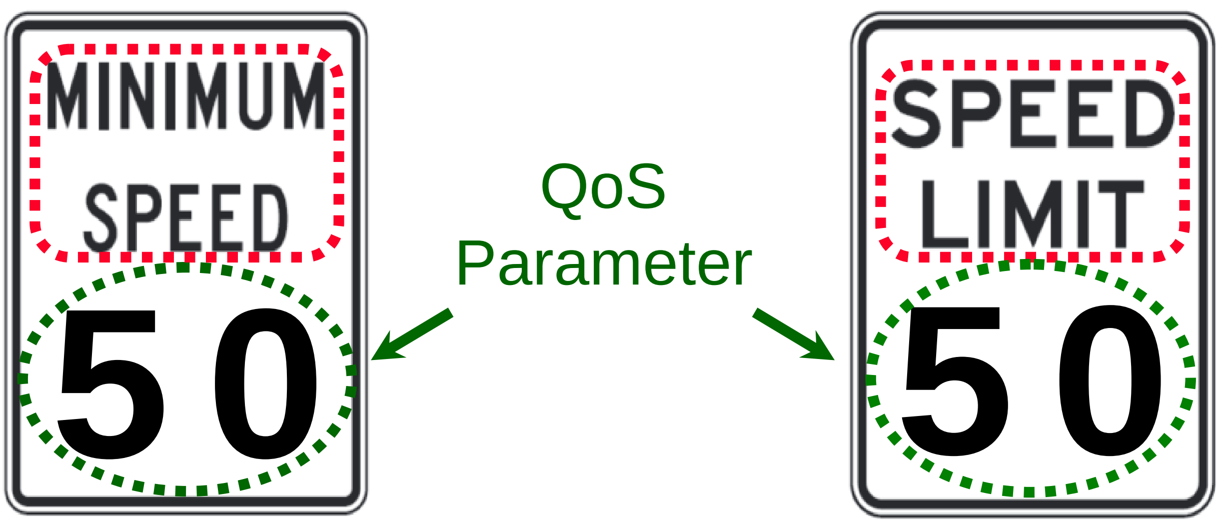


Modifying Shared Blocks to Support Two QoS Modes

- Now each QoS block supports two operation modes:

  1) Deployment mode: Conventional QoS mode (Operation)
     - QoS parameters are treated as a **lower bound** of received resources in runtime.

  2) Enforcement mode: New QoS Mode (Estimation)
     - QoS parameters are treated as an **upper bound** of received resources in runtime.

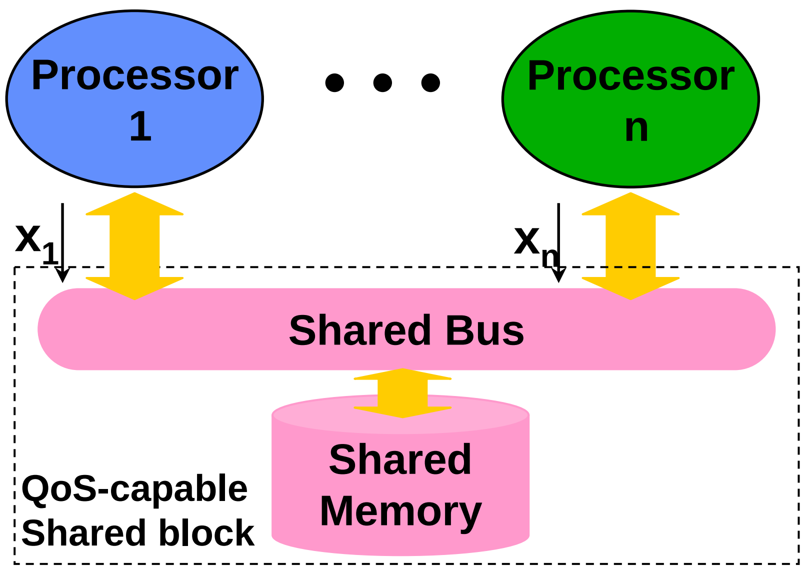


1) Deployment Mode ***2) Enforcement Mode***

Assumptions

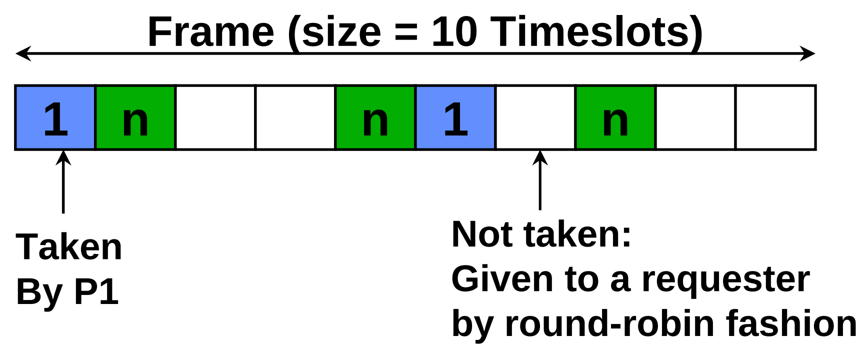
- Single-threaded apps; no shared objects.
- No contentions within a node.
  (e.g. multithreaded processors)
- No preemption of a scheduled task.
- Negligible impact of initial states on execution time.

Baseline MP System



x_i: Fraction of time slots reserved for Processor i

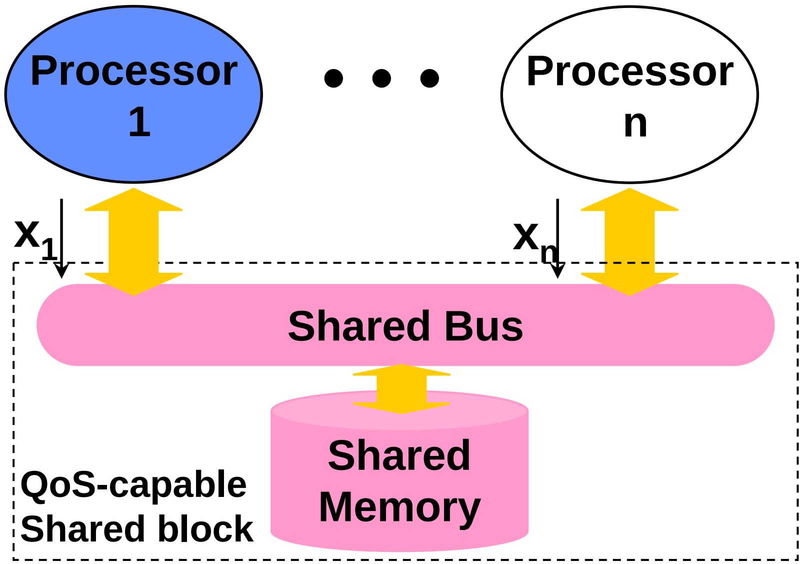
- Simple fixed frame-based scheduling for memory access
  - Example (x_1=0.2, x_n=0.3)

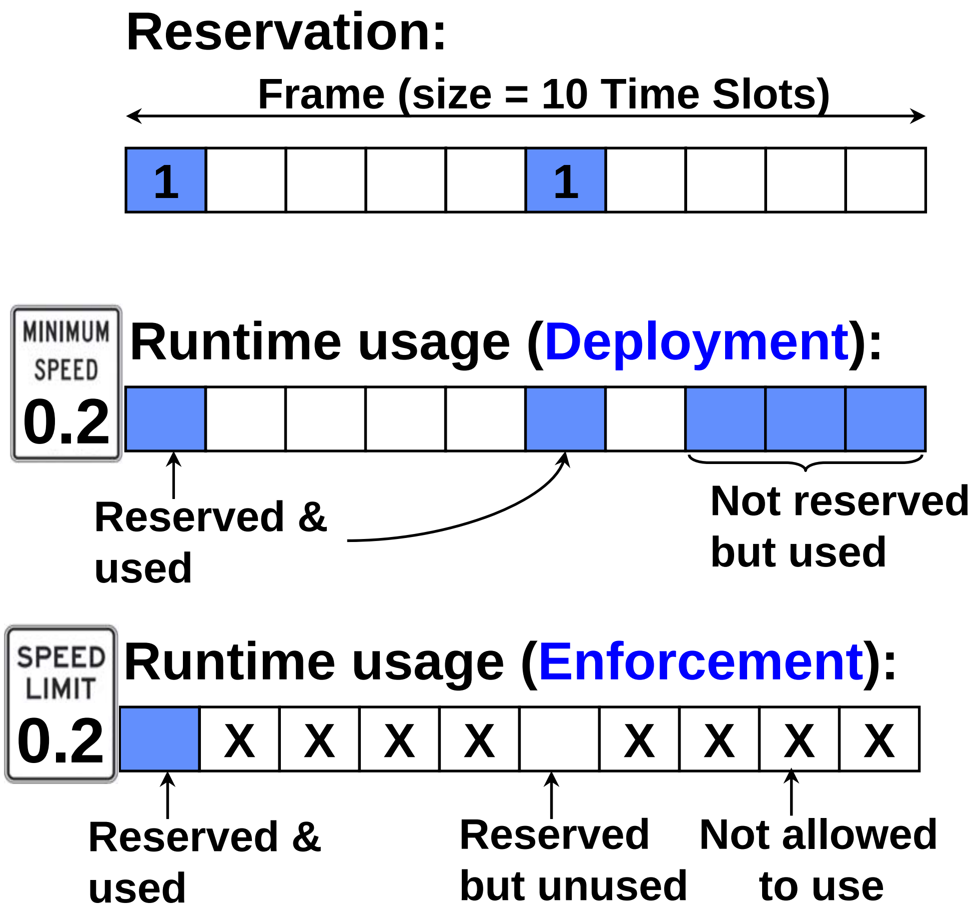


  - x_i determines
    - » Minimum bandwidth
    - » Worst-case latency

Runtime Usage Example in METERG System

- Example (x_1=0.2)





Obtain Minimal Resource Reservation Using METERG

How to find minimal resource reservation with METERG:

In measurement phase:

- Step 1: Measure performance in **enforcement** mode for a QoS parameter.
- Step 2: Iterate Step 1 to find a minimal QoS parameter (yet meeting the performance goal).
- Step 3: Store the minimal QoS parameter.

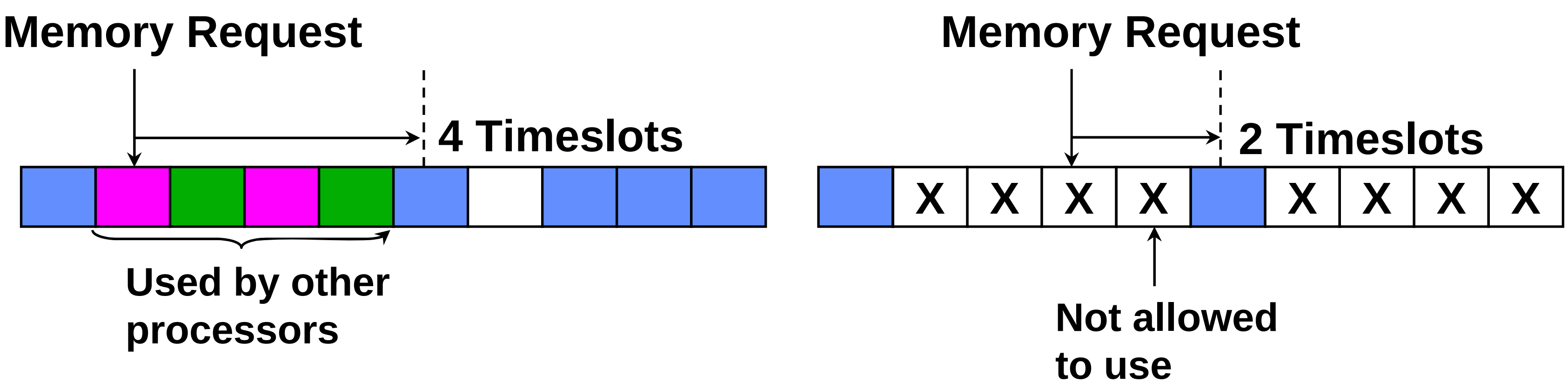
In operation phase:

- Step 4: Execute the program in **deployment** mode with the stored QoS parameter for guaranteed performance.

Bandwidth Guarantees Are Not Enough: An Example

- Shared bus example again (x_1=0.2)

Deployment Mode ***Enforcement Mode***



- Bandwidth inequality is guaranteed but not latency.

  $\mathbf{\mathit{Bandwidth_{DEP}\ (x_i) \geq Bandwidth_{ENF}\ (x_i)}}$ **(O)**

  $\mathbf{\mathit{MAX\ [Latency_{DEP}\ (x_i)] \leq MIN\ [Latency_{ENF}\ (x_i)]}}$ **(X)**

  → May cause end-to-end performance inversion.

Two Enforcement Modes: Safety-Tightness Tradeoff

- Relaxed enforcement (R-ENF) mode: Bandwidth-only guarantees

  $$\textit{Bandwidth}_{DEP}(x_i) \geq \textit{Bandwidth}_{R\text{-}ENF}(x_i)$$

  – There is a (small) chance for observed execution time *in enforcement mode* to be smaller than $T_{MAX}(x, i)$. (Not Safe)

- Strict enforcement (S-ENF) mode: Bandwidth and latency guarantees

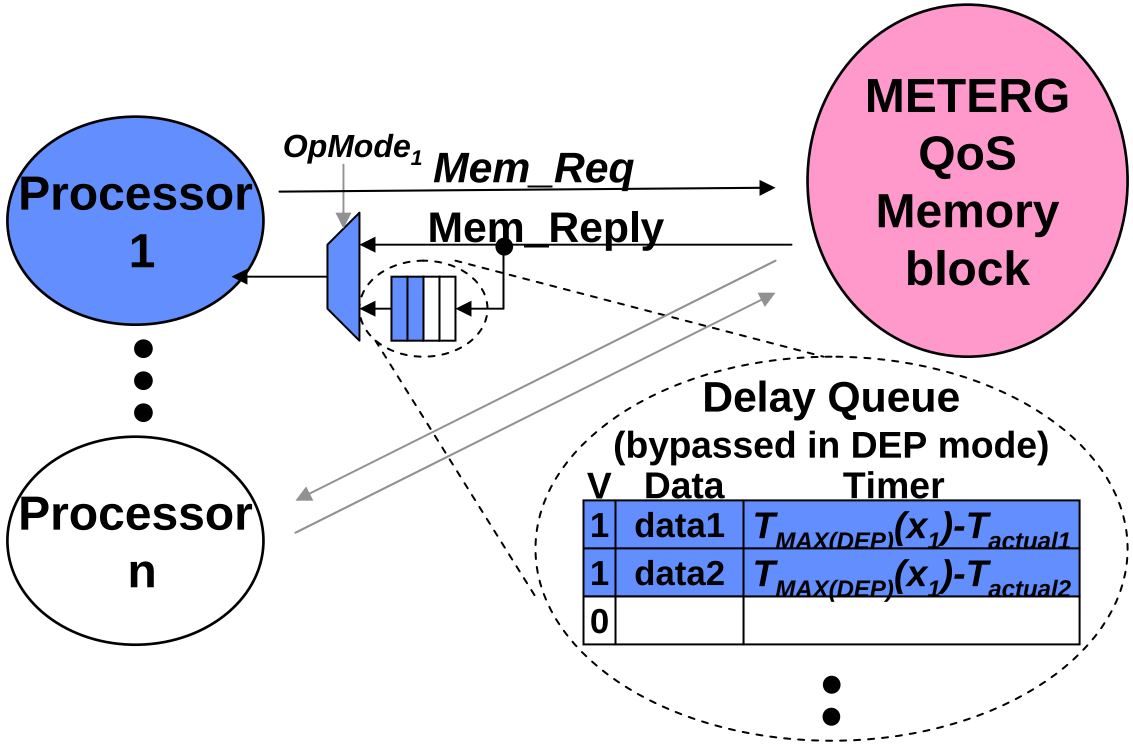
  $$\textit{Bandwidth}_{DEP}(x_i) \geq \textit{Bandwidth}_{S\text{-}ENF}(x_i) \ \&\&$$

  $$MAX[\textit{Latency}_{DEP}(x_i)] \leq MIN[\textit{Latency}_{S\text{-}ENF}(x_i)]$$

  – Observed execution time is always greater than $T_{MAX}(x, i)$ *in enforcement mode* as long as there is no timing anomaly in processor. (Safe)

  – Estimation of $T_{MAX}(x, i)$ is looser.

Memory System with Strict Enforcement Mode: An Implementation

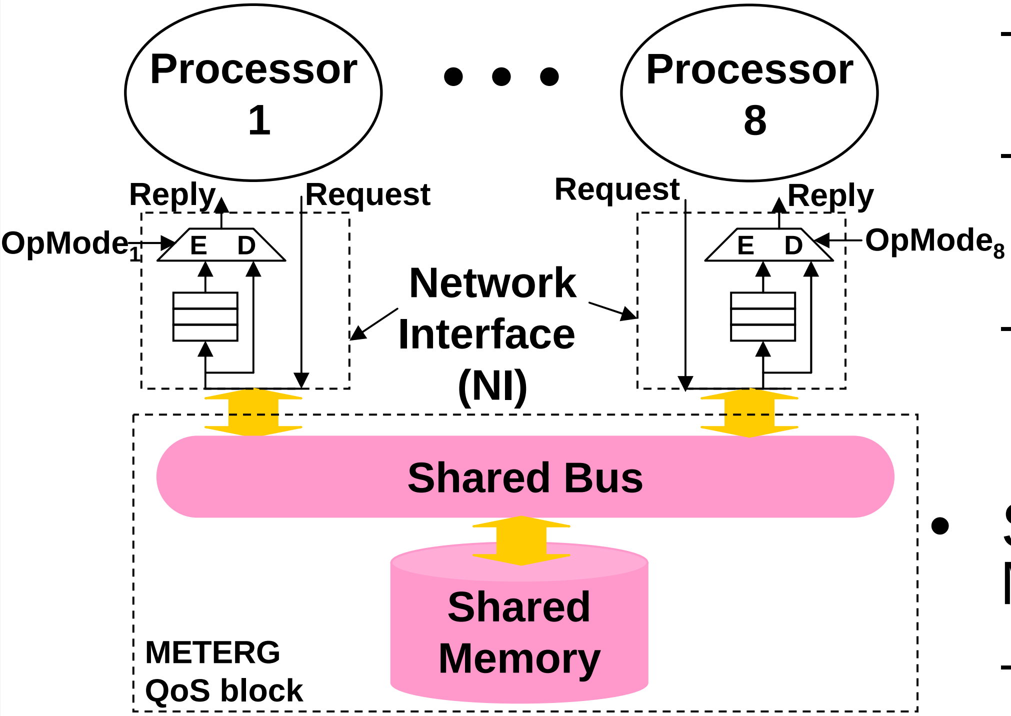


$T_{MAX(DEP)}(x_1)$:
Worst-Case latency in Deployment Mode for given param x_1

$T_{actual1}$:
Actual time taken to service request 1

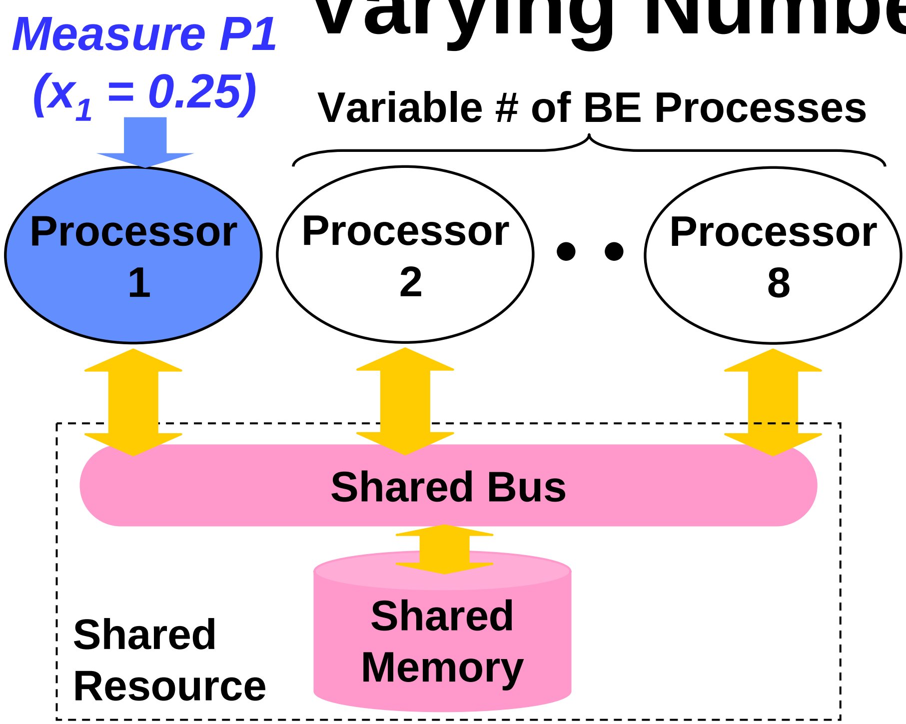
- Delay queue
  - **In Deployment mode: Bypassed**
  - **In Enforcement mode: Deferring delivery to processor for “lucky” messages**

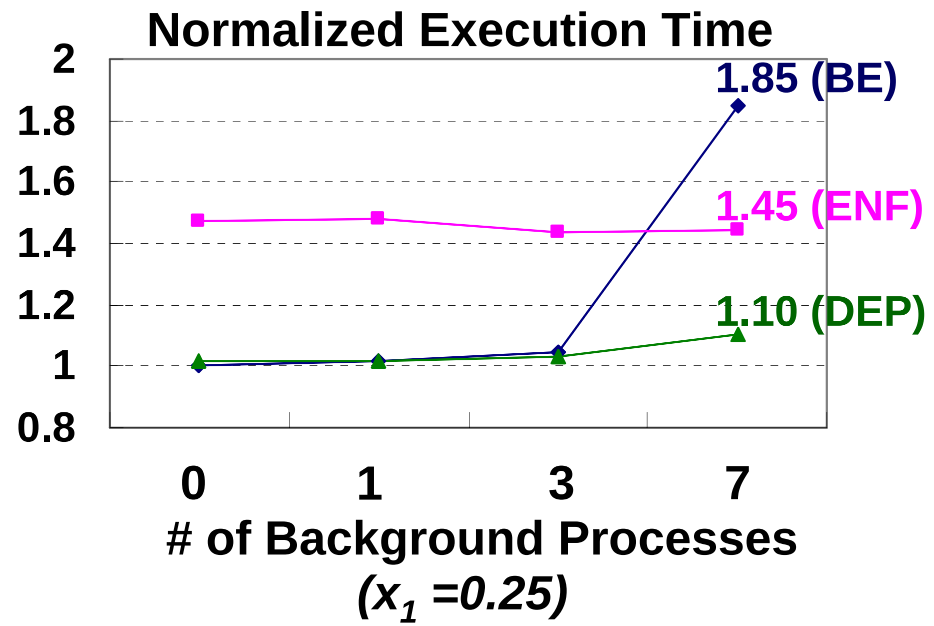
Evaluation: Simulation Setup



- Simulation Setup
  - 8-way SMP running Linux
  - In-order core running with 5x faster clock than the system bus
  - Detailed memory contention model with a simple fixed-frame TDM bus
  - 32 KB unified blocking L1 cache / No L2 cache
- Synthetic benchmark: `Memread`
  - Infinite loop accessing a large chunk of memory sequentially
  - Performance bottlenecked by memory system performance

Evaluation: Varying Number of Processes





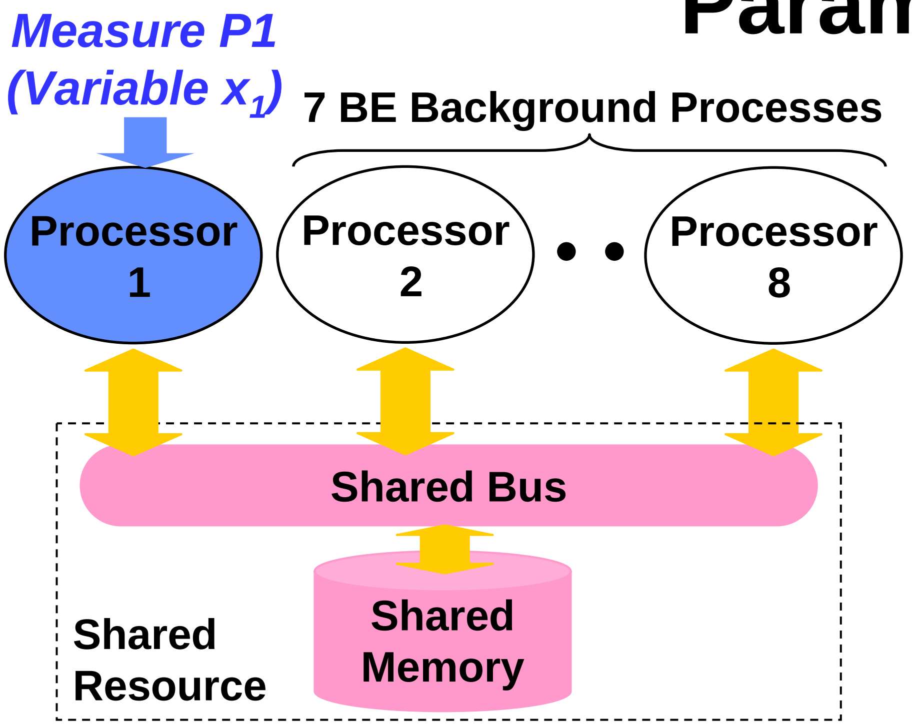
- Execution time degradation as number of background processes increases from 0 to 7

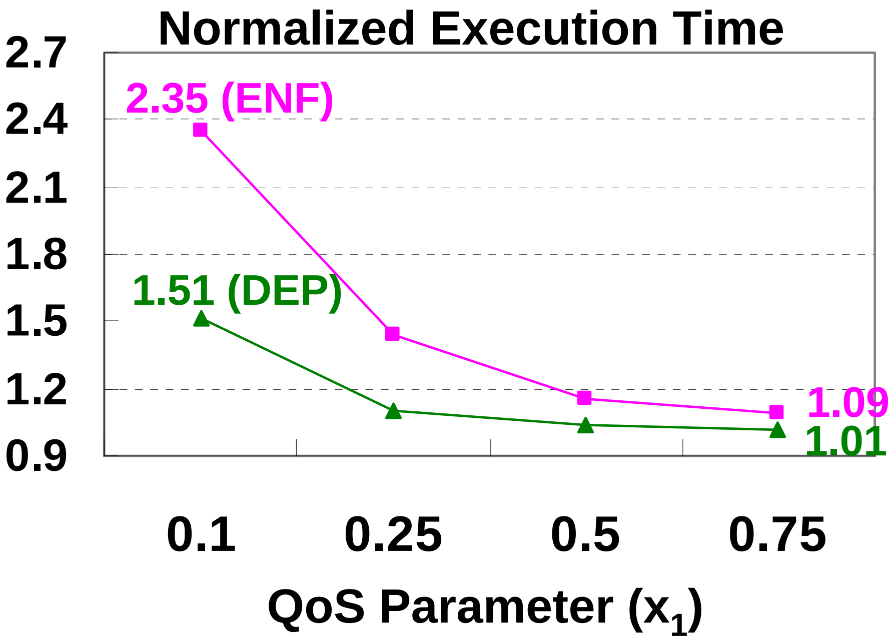
  without QoS (Best-Effort): 85 %

  with QoS (Deployment): 10 %

- Estimated execution time upper bound for given QoS parameter (x_1=0.25): 45 %

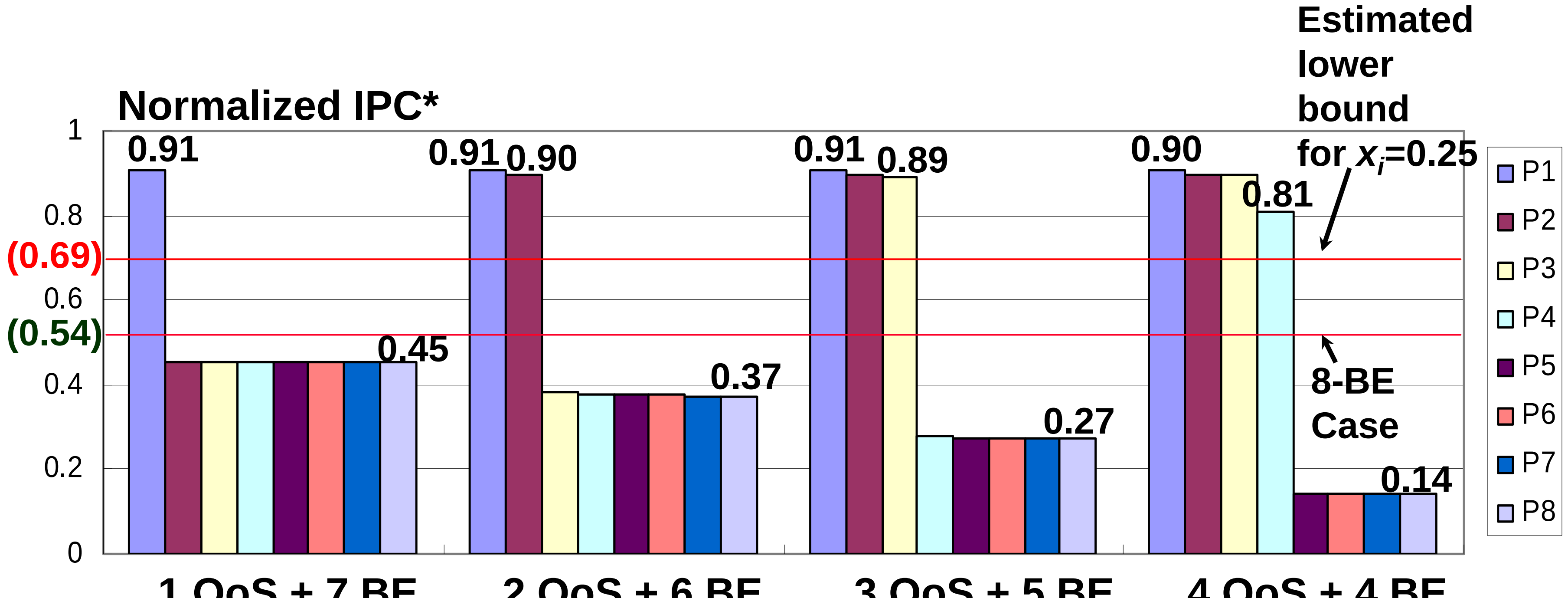
Evaluation: Varying QoS Parameters





- Performance estimation becomes tighter as QoS parameter (x_1) increases.
  - Because the worst-case latency in accessing memory decreases accordingly

Evaluation: Varying Number of QoS Processes



QoS Parameters: (0.25, 0, 0, 0, 0, 0, 0, 0) | (0.25, 0.25, 0, 0, 0, 0, 0, 0) | (0.25, 0.25, 0.25, 0, 0, 0, 0, 0) | (0.25, 0.25, 0.25, 0.20, 0, 0, 0, 0)

- Performance impact on a QoS process by other QoS processes is negligible (<2%) as long as system is not oversubscribed.

(* Higher Number = Better Performance)

Conclusion

- In general-purpose MPs, shared resource contention causes large variation of execution time of a task.

  (~2x increase observed in 8-processor case)
- METERG QoS System provides an easy way (by measurement) to estimate execution time under worst-case resource contention for a given QoS parameter.
  - Introducing a new QoS mode (Enforcement Mode) for performance estimation
- Using METERG QoS System, minimal QoS parameter can be easily found for given instruction sequence and performance goal.