ILLINOIS
CSL | Coordinated Science Lab
COLLEGE OF ENGINEERING

Subho S. Banerjee, Saurabh Jha,
Zbigniew Kalbarczyk, Ravishankar K. Iyer

BayesPerf: Minimizing Performance Monitoring Errors Using Bayesian Statistics

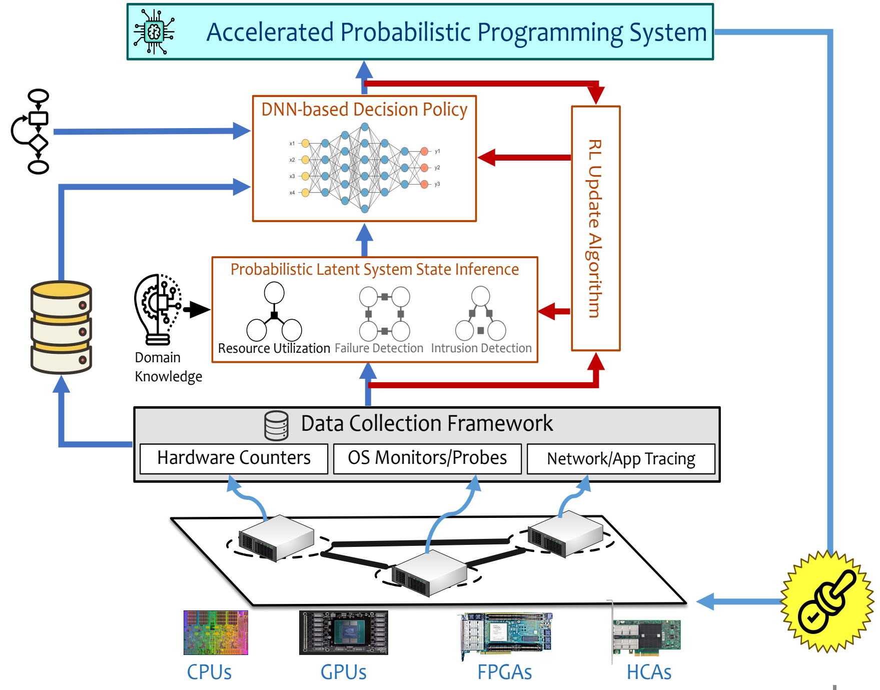
ASPLOS 2021

Motivation

- Hardware performance counters (HPCs) are low-level monitors that provide a window into the system

Motivation

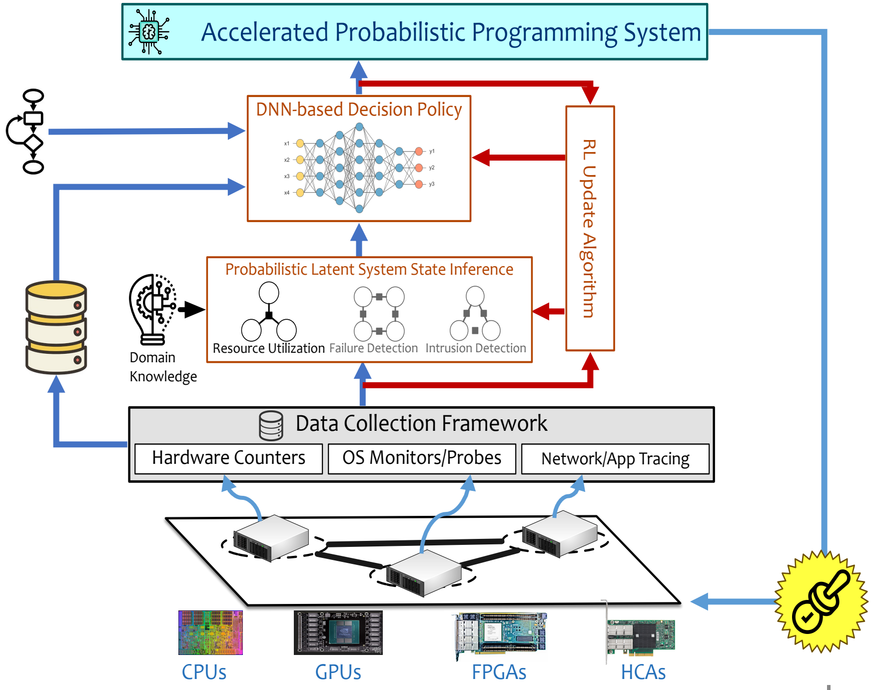
- Hardware performance counters (HPCs) are low-level monitors that provide a window into the system
  - Performance Profiling – Why is my code slow?
  - Profile-Guided Optimization – Provide sample traces to a compiler
  - **“Learned” controllers for scheduling, reliability, DVFS, security**



Motivation

- Hardware performance counters (HPCs) are low-level monitors that provide a window into the system
  - Performance Profiling – Why is my code slow?
  - Profile-Guided Optimization – Provide sample traces to a compiler
  - **"Learned" controllers for scheduling, reliability, DVFS, security**

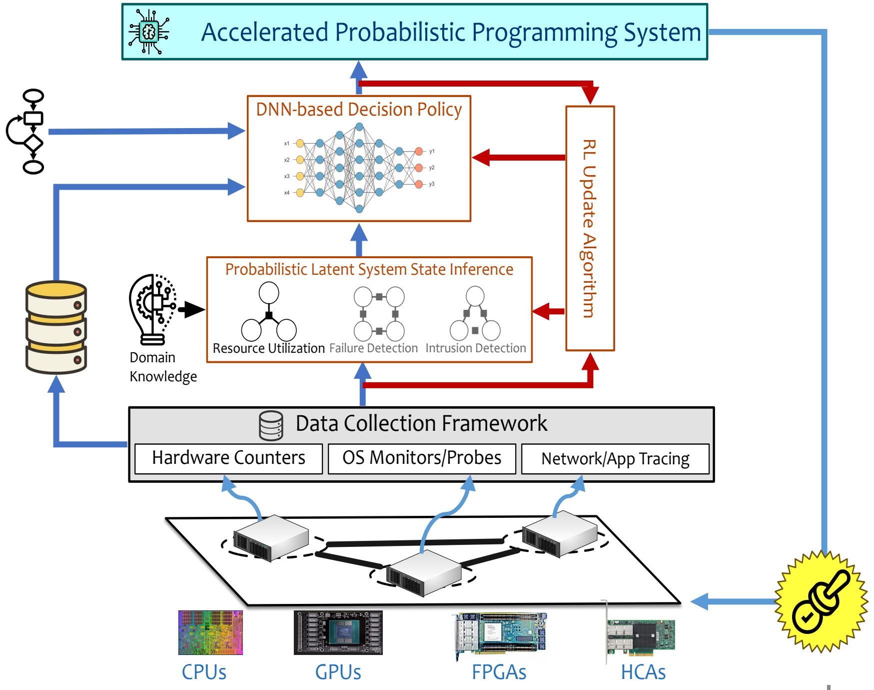
- **Issue**: HPC measurement noise does not scale with # of measurements



Motivation

- Hardware performance counters (HPCs) are low-level monitors that provide a window into the system
  - Performance Profiling – Why is my code slow?
  - Profile-Guided Optimization – Provide sample traces to a compiler
  - **"Learned" controllers for scheduling, reliability, DVFS, security**

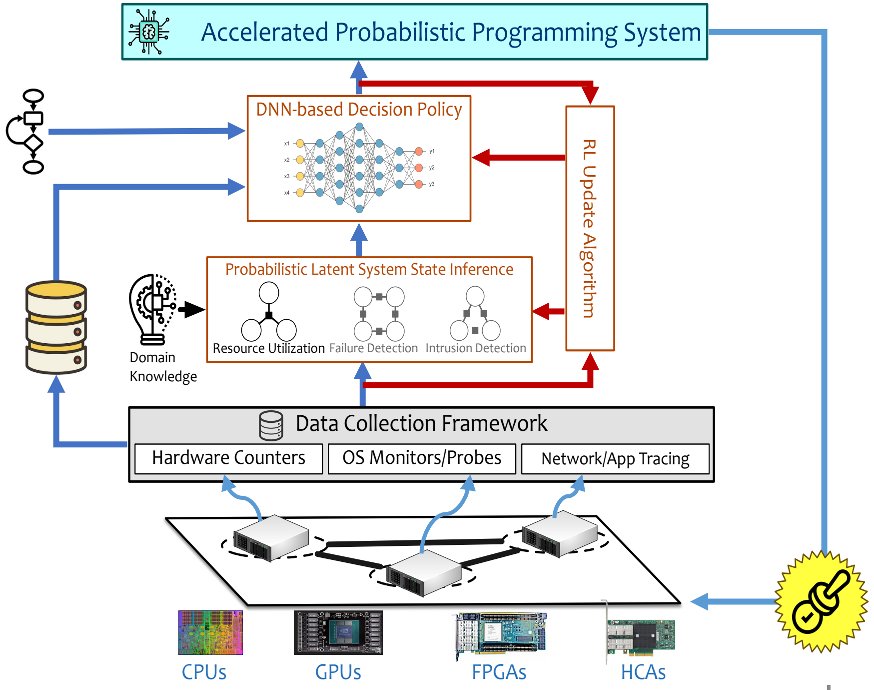
- **Issue**: HPC measurement noise does not scale with # of measurements
  - Quantifying/Correcting errors is difficult: Ground truth not known
  - No real time correction techniques
  - Directly limits the scalability of "learned" controllers



Motivation

- Hardware performance counters (HPCs) are low-level monitors that provide a window into the system
  - Performance Profiling – Why is my code slow?
  - Profile-Guided Optimization – Provide sample traces to a compiler
  - **“Learned” controllers for scheduling, reliability, DVFS, security**

- **Issue**: HPC measurement noise does not scale with # of measurements
  - Quantifying/Correcting errors is difficult: Ground truth not known
  - No real time correction techniques
  - Directly limits the scalability of “learned” controllers



BayesPerf: A system to quantify and minimize errors HPCs
- **Bayesian generative model of HPC error process**
- **System implementation for Linux on x86 and ppc64 CPUs**

Hardware Performance Counter Primer

Counters used in ***Polling Mode***

```
ReadCounter(&start);

/* Sum two arrays */
for(i = 0; i < len; i++)
        z[i] = x[i] + y[i];

ReadCounter(&end);
```

Counted Events = End - Start

- Time
- MB of memory read/written
- TLB misses
- …

Hardware Performance Counter Primer

Counters used in ***Polling Mode***

```
ReadCounter(&start);

/* Sum two arrays */
for(i = 0; i < len; i++)
        z[i] = x[i] + y[i];

ReadCounter(&end);
```

Counted Events = End - Start

- Time
- MB of memory read/written
- TLB misses
- …

	Clock: Unhalted Core	Clock: Unhalted Ref	Instructions Retired	
Fixed HPC:	F-HPC0	F-HPC1	F-HPC2	
Programmable HPC:	P-HPC3	P-HPC4	P-HPC5	P-HPC6
Control Registers:	C-HPC3	C-HPC4	C-HPC5	C-HPC6

Hardware Performance Counter Primer

Counters used in ***Polling Mode***

```
ReadCounter(&start);

/* Sum two arrays */
for(i = 0; i < len; i++)
        z[i] = x[i] + y[i];

ReadCounter(&end);
```

Counted Events = End - Start

- Time
- MB of memory read/written
- TLB misses
- …

	Clock: Unhalted Core	Clock: Unhalted Ref	Instructions Retired	
Fixed HPC:	F-HPC0	F-HPC1	F-HPC2	
Programmable HPC:	P-HPC3	P-HPC4	P-HPC5	P-HPC6
Control Registers:	C-HPC3	C-HPC4	C-HPC5	C-HPC6

Read counters on x86 processors
`rdmsr, rdpmc, rdtsc, rdtscp`

Write event configuration on x86 processors
`wrmsr`

Hardware Performance Counter Primer

Counters used in ***Polling Mode***

```
ReadCounter(&start);

/* Sum two arrays */
for(i = 0; i < len; i++)
        z[i] = x[i] + y[i];

ReadCounter(&end);
```

Counted Events = End - Start

- Time
- MB of memory read/written
- TLB misses
- …

	Clock: Unhalted Core	Clock: Unhalted Ref	Instructions Retired	
Fixed HPC:	F-HPC0	F-HPC1	F-HPC2	
Programmable HPC:	P-HPC3	P-HPC4	P-HPC5	P-HPC6
Control Registers:	C-HPC3	C-HPC4	C-HPC5	C-HPC6

Read counters on x86 processors
`rdmsr, rdpmc, rdtsc, rdtscp`

Write event configuration on x86 processors
`wrmsr`

Huge Imbalance
#Events >> #Counters

Hardware Performance Counter Primer

Counters used in ***Polling Mode***

```
ReadCounter(&start);

/* Sum two arrays */
for(i = 0; i < len; i++)
        z[i] = x[i] + y[i];

ReadCounter(&end);
```

Counted Events = End - Start

- Time
- MB of memory read/written
- TLB misses
- …

Counters used in ***Sampling Mode***

```
ReadCounter1(&start1);
ReadCounter2(&start2);

/* Sum two arrays */
for(i = 0; i < len; i++) {
        z[i] = x[i] + y[i];
        if (i%2) SwapCounters()
}

ReadCounter1(&end1);
ReadCounter2(&end2);
```

	Clock: Unhalted Core	Clock: Unhalted Ref	Instructions Retired	
Fixed HPC:	F-HPC0	F-HPC1	F-HPC2	
Programmable HPC:	P-HPC3	P-HPC4	P-HPC5	P-HPC6
Control Registers:	C-HPC3	C-HPC4	C-HPC5	C-HPC6

Read counters on x86 processors
`rdmsr, rdpmc, rdtsc, rdtscp`

Write event configuration on x86 processors
`wrmsr`

Huge Imbalance
#Events >> #Counters

Hardware Performance Counter Primer

Counters used in ***Polling Mode***

```
ReadCounter(&start);

/* Sum two arrays */
for(i = 0; i < len; i++)
        z[i] = x[i] + y[i];

ReadCounter(&end);
```

Counted Events = End - Start

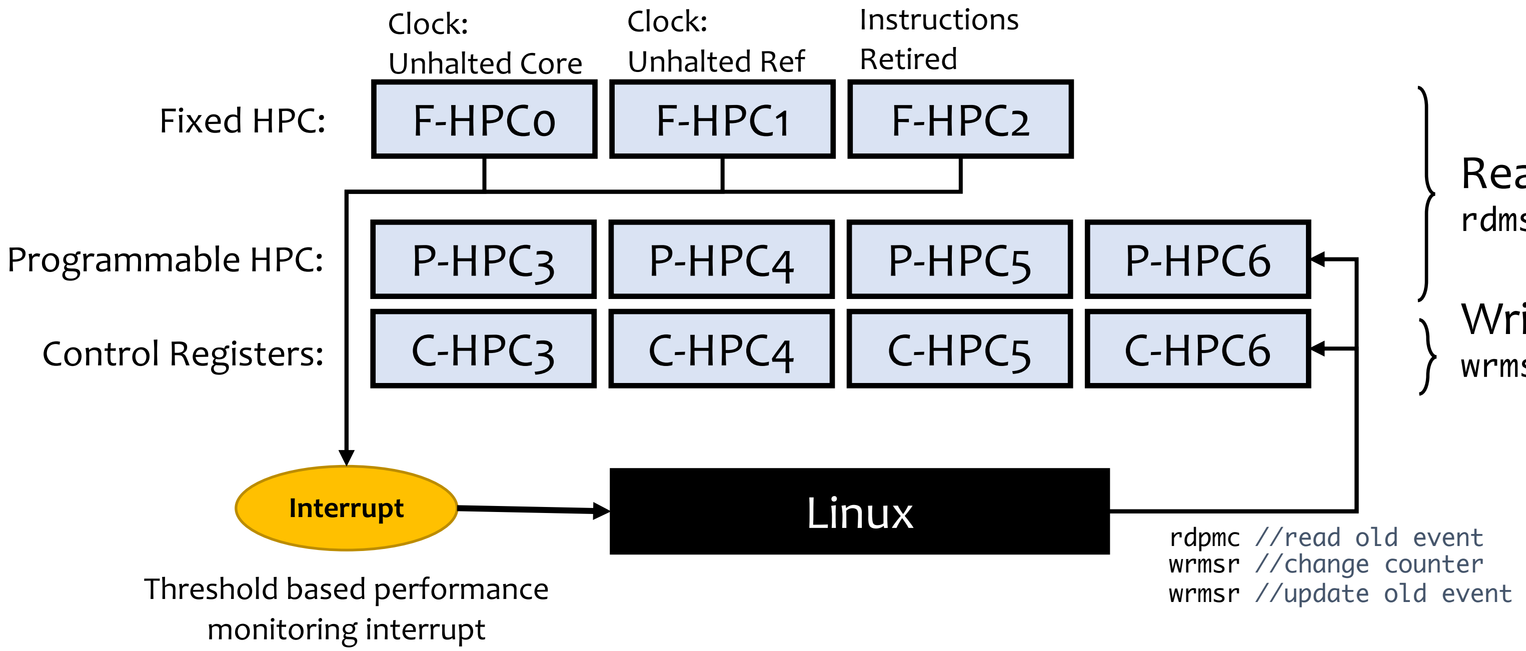
- Time
- MB of memory read/written
- TLB misses
- …

Counters used in ***Sampling Mode***

```
ReadCounter1(&start1);
ReadCounter2(&start2);

/* Sum two arrays */
for(i = 0; i < len; i++) {
        z[i] = x[i] + y[i];
        if (i%2) SwapCounters()
}

ReadCounter1(&end1);
ReadCounter2(&end2);
```

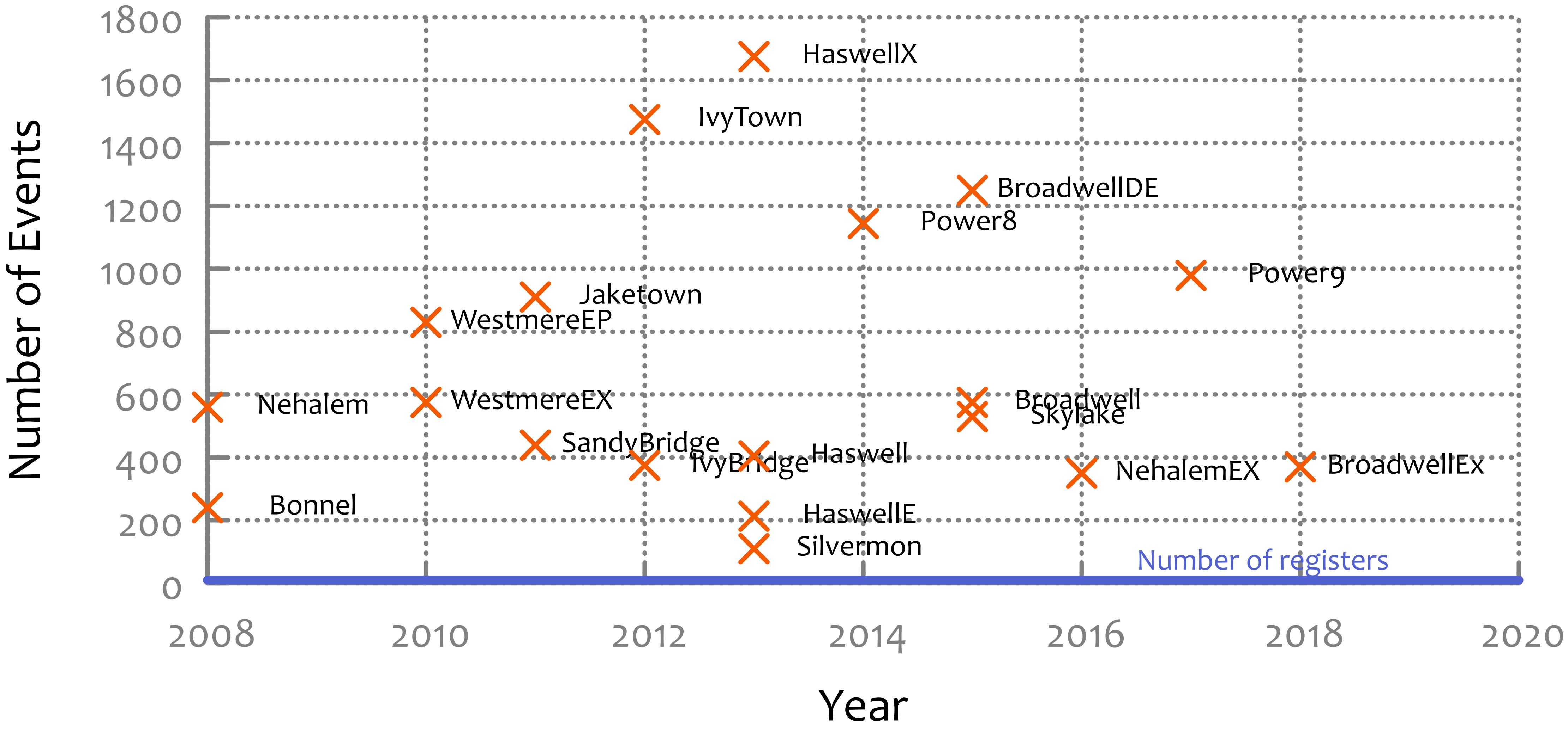


Read counters on x86 processors
`rdmsr, rdpmc, rdtsc, rdtscp`

Write event configuration on x86 processors
`wrmsr`

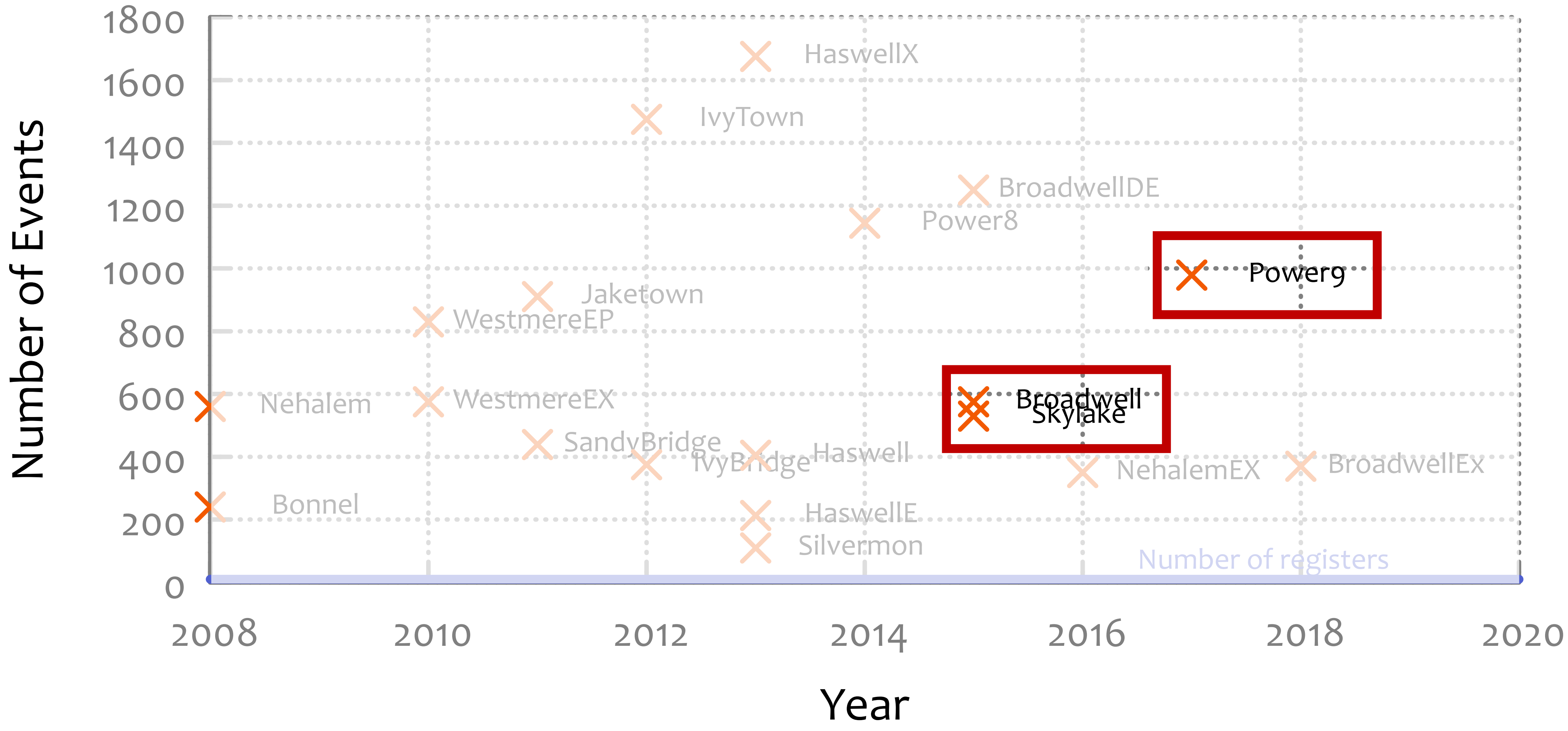
Huge Imbalance
#Events >> #Counters

Imbalance between Events & Counters



[Adapted from "So many performance events, so little time," APSys 2016]

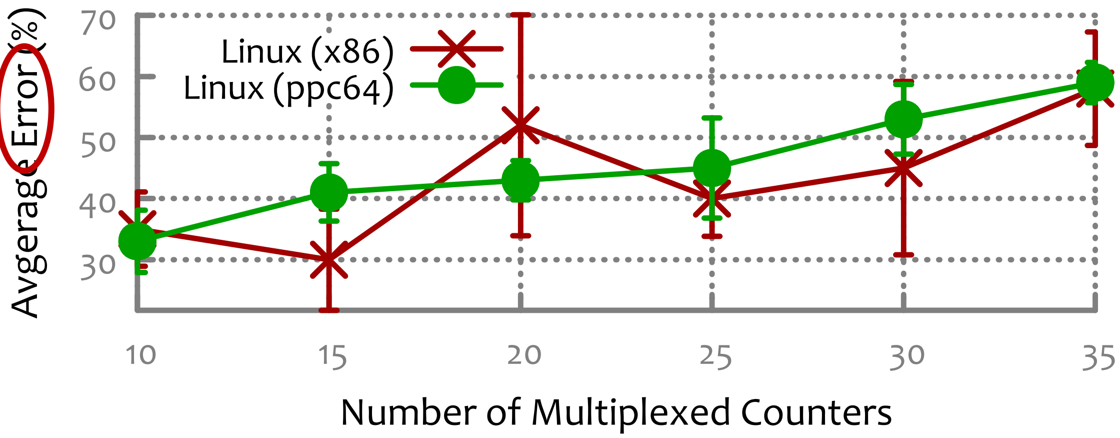
Imbalance between Events & Counters



[Adapted from "So many performance events, so little time," APSys 2016]

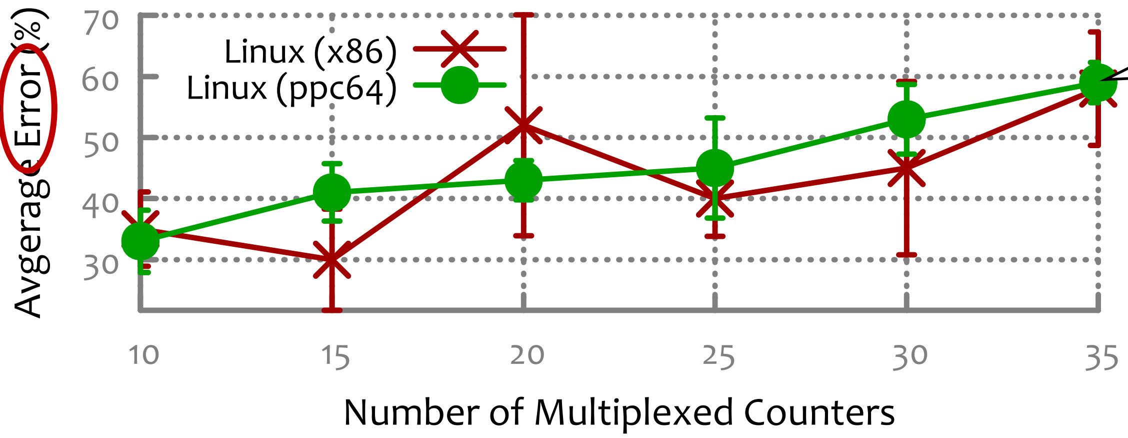
Errors in HPC Measurements

Error = |Sampling** - Polled|



Errors in HPC Measurements

Error = |Sampling** - Polled|



Multiplexing is bad:
Errors increase linearly with multiplexing

Many sources of error:

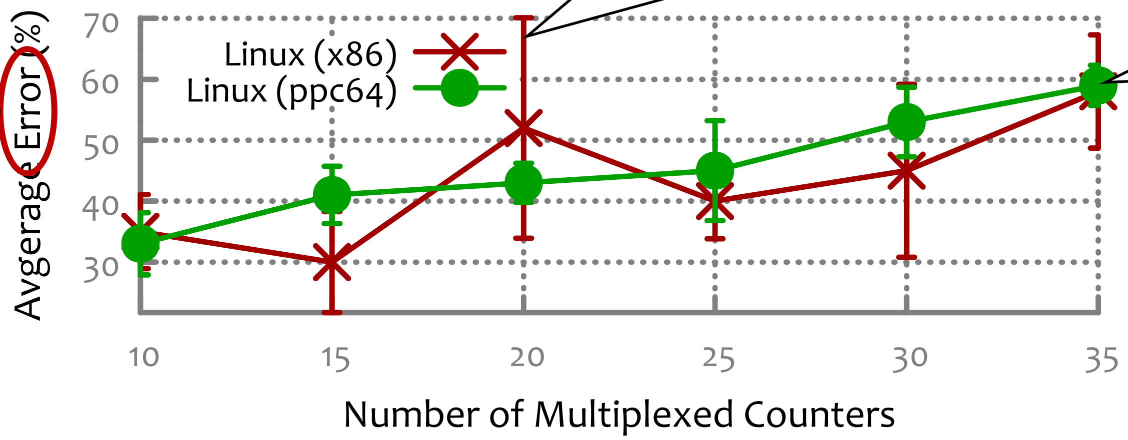
- Multiplexing HPCs
  - How to scale up counters for the time that they were not scheduled?

Errors in HPC Measurements

Large variance: Non-determinism in measurements

Multiplexing is bad: Errors increase linearly with multiplexing

Error = |Sampling** - Polled|



Many sources of error:

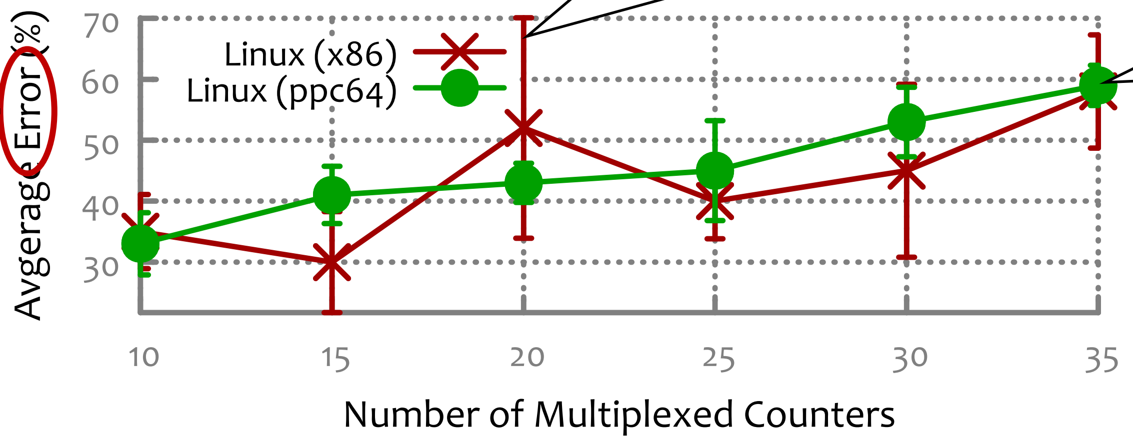
- Multiplexing HPCs
  - How to scale up counters for the time that they were not scheduled?
- Non-determinism
  - Order in which interrupts are served
  - Dropped measurements: Backpressure in ring-buffers between kernel and userspace
  - Interactions between collocated processes/threads

Errors in HPC Measureme

Large variance: Non-determinism in measurements

Multiplexing is bad: Errors increase linearly with multiplexing

Error = |Sampling** - Polled|

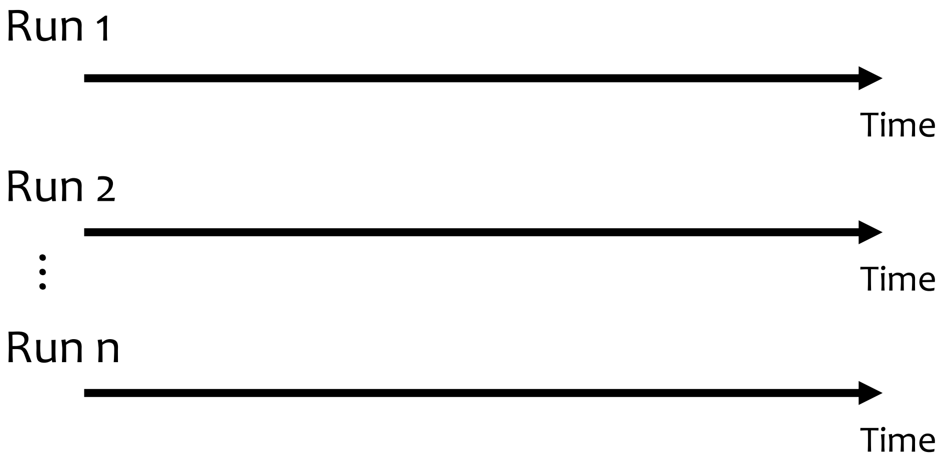


Many sources of error:

- Multiplexing HPCs
  - How to scale up counters for the time that they were not scheduled?
- Non-determinism
  - Order in which interrupts are served
  - Dropped measurements: Backpressure in ring-buffers between kernel and userspace
  - Interactions between collocated processes/threads
- (Instruction Pointer) Skid
  - Instruction level parallelism: Counters change between time interrupt enters processor pipeline and the interrupt handler is triggered
- CPU Design/Implmentation Bug

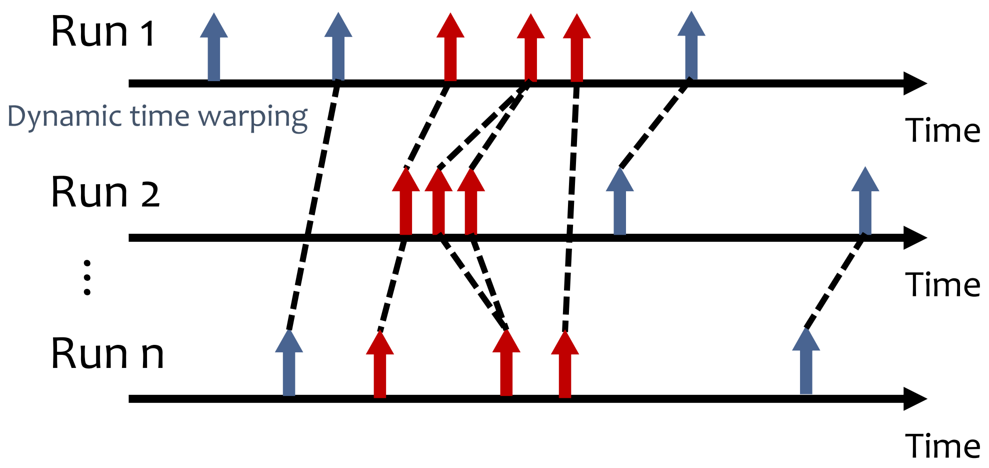
Error Correction for HPCs

Traditional Solution (Offline Variance Reduction)



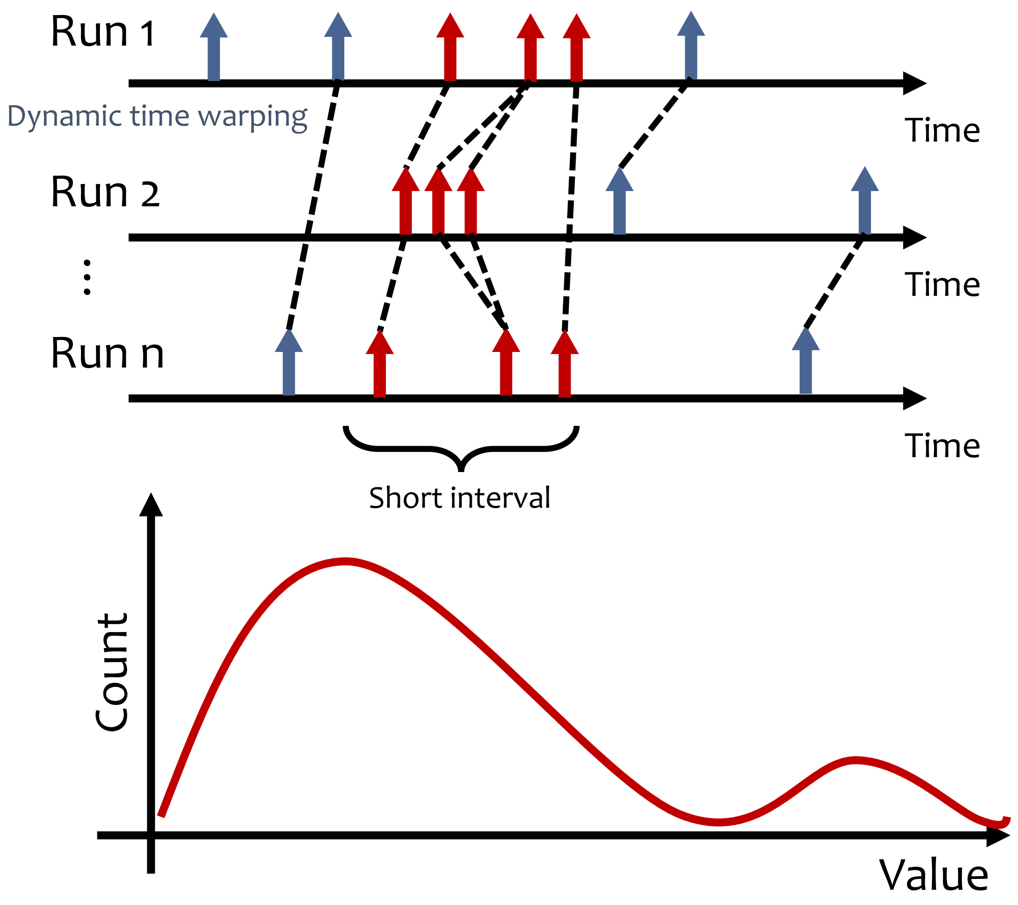
Error Correction for HPCs

Traditional Solution (Offline Variance Reduction)



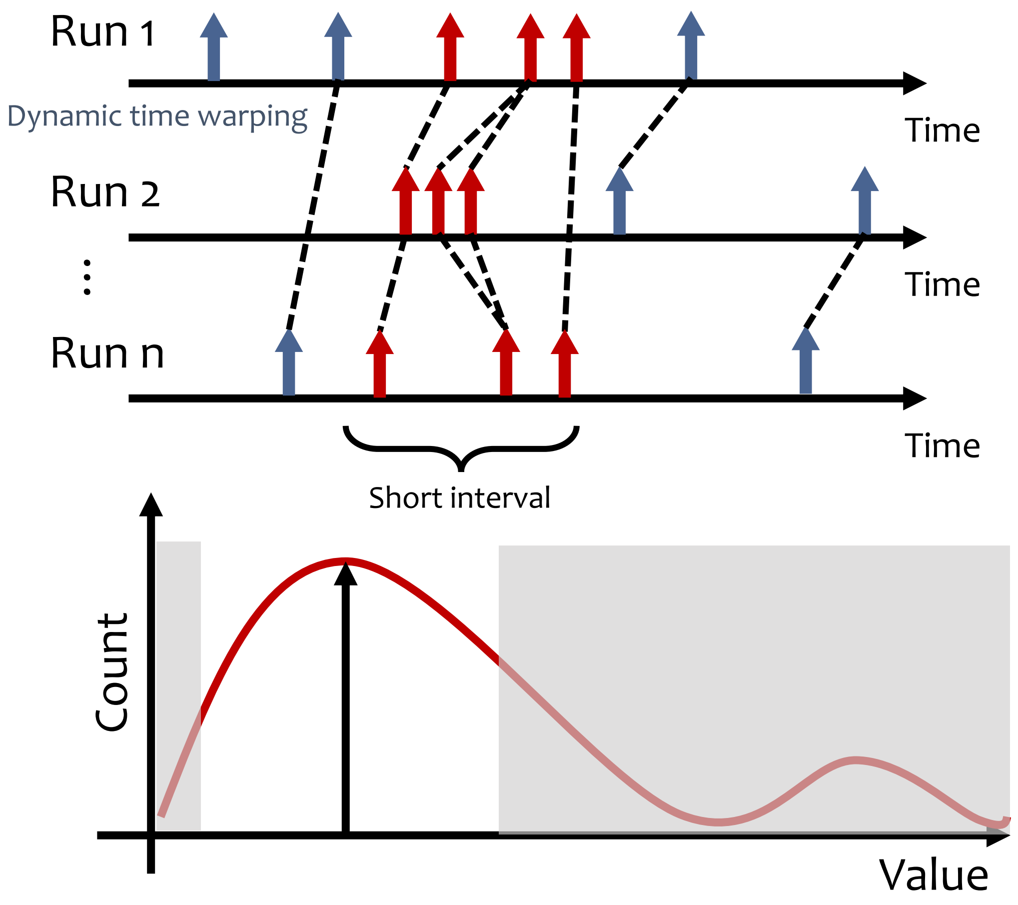
Error Correction for HPCs

Traditional Solution (Offline Variance Reduction)



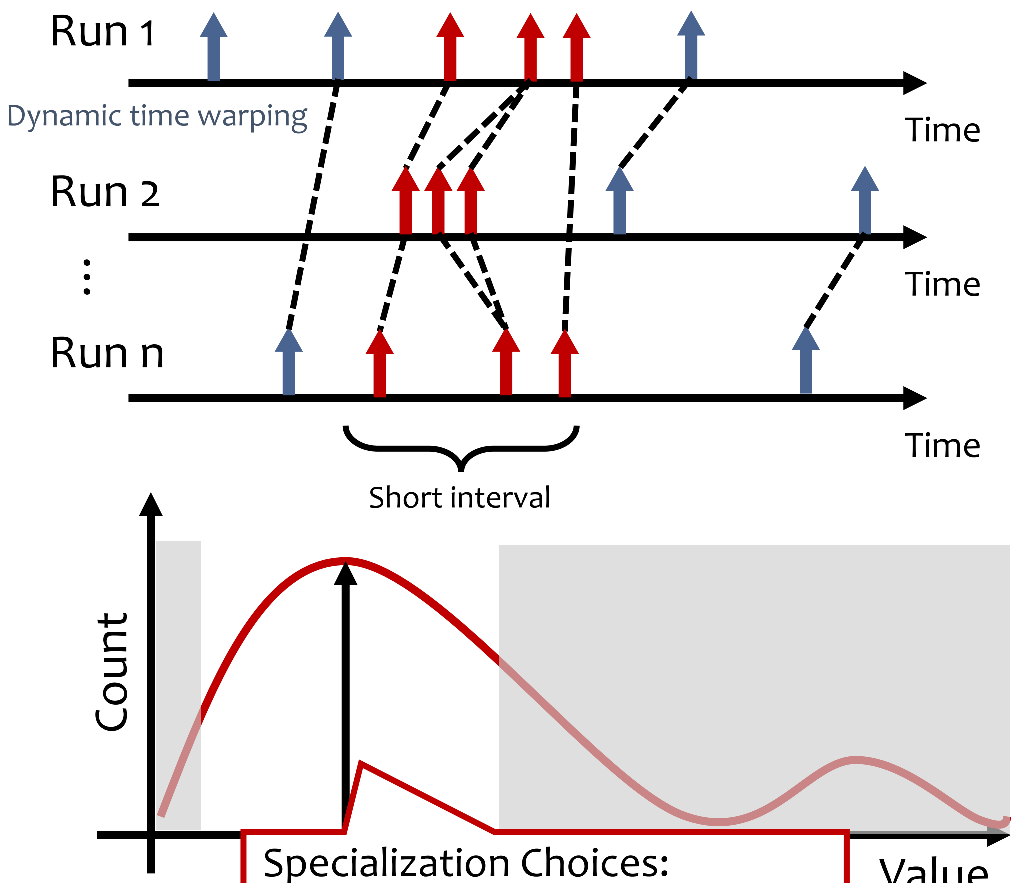
Error Correction for HPCs

Traditional Solution (Offline Variance Reduction)



Error Correction for HPCs

Traditional Solution (Offline Variance Reduction)

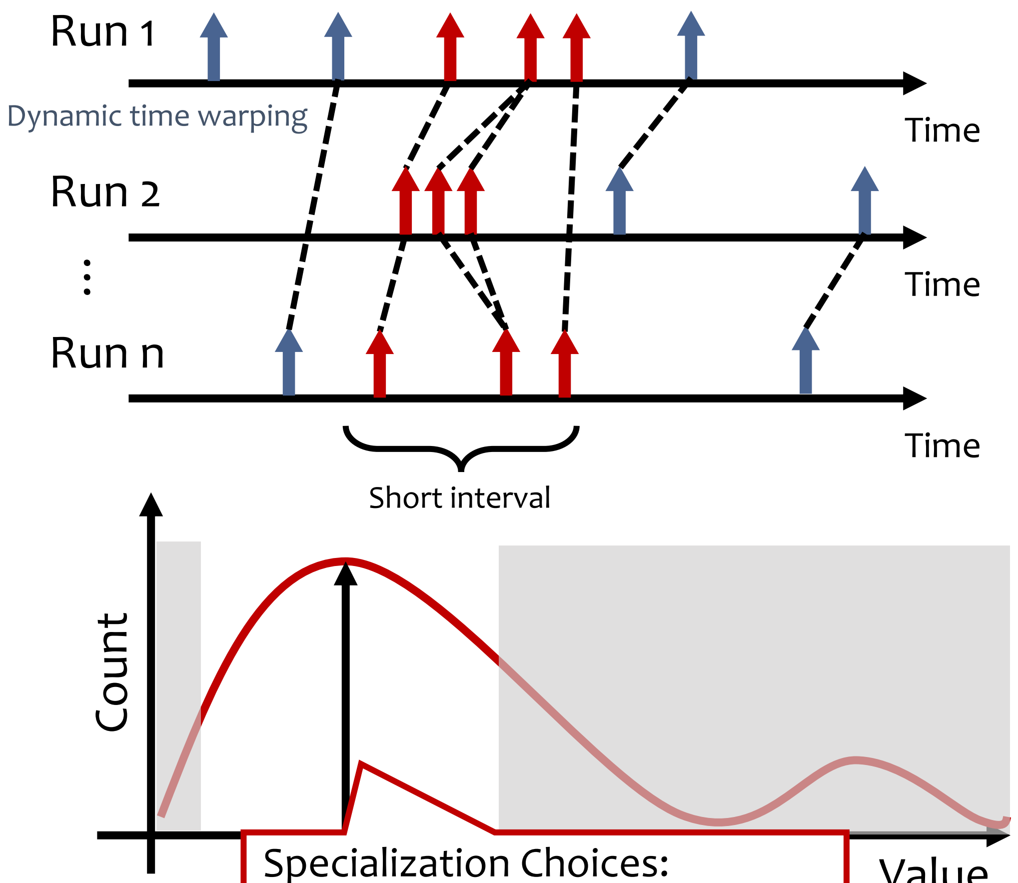


Specialization Choices:
- Time warping cost function
- Shape of curve
- Thresholds

Unusable – Not real time!

Error Correction for HPCs

Traditional Solution (Offline Variance Reduction)



Specialization Choices:
- Time warping cost function
- Shape of curve
- Thresholds

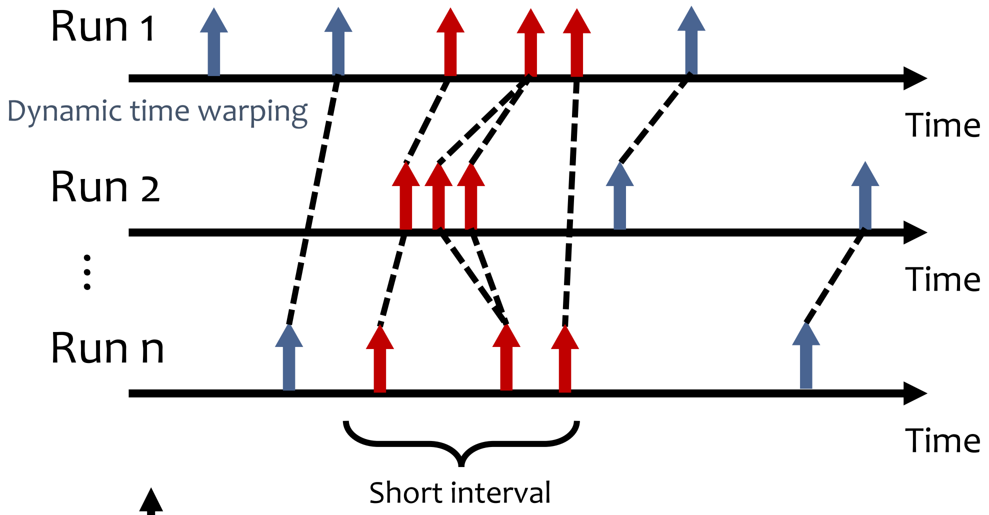
Unusable – Not real time!

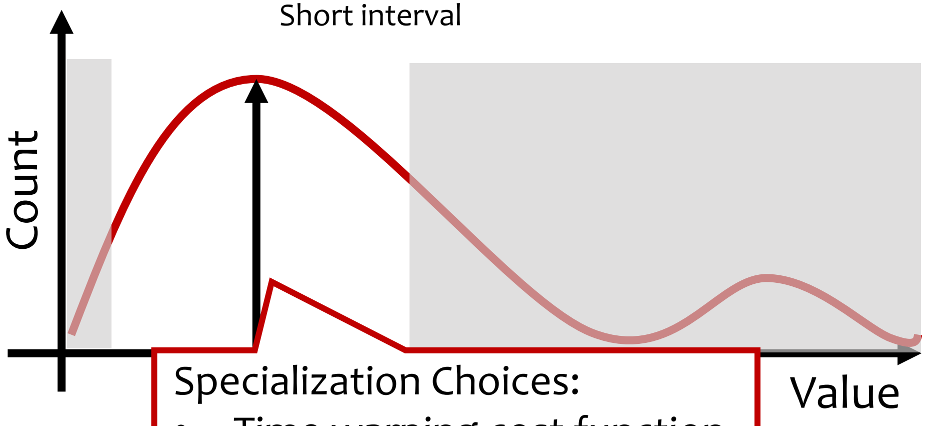
BayesPerf Correction (Key Idea)

Key Insight: Different perf counters are interrelated based on system architecture

Error Correction for HPCs

Traditional Solution (Offline Variance Reduction)





Specialization Choices:
- Time warping cost function
- Shape of curve
- Thresholds

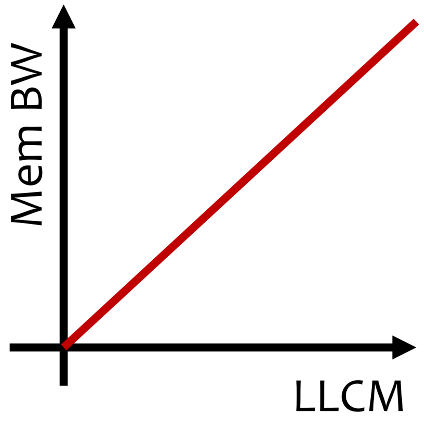
Unusable – Not real time!

BayesPerf Correction (Key Idea)

Key Insight: Different perf counters are interrelated based on system architecture

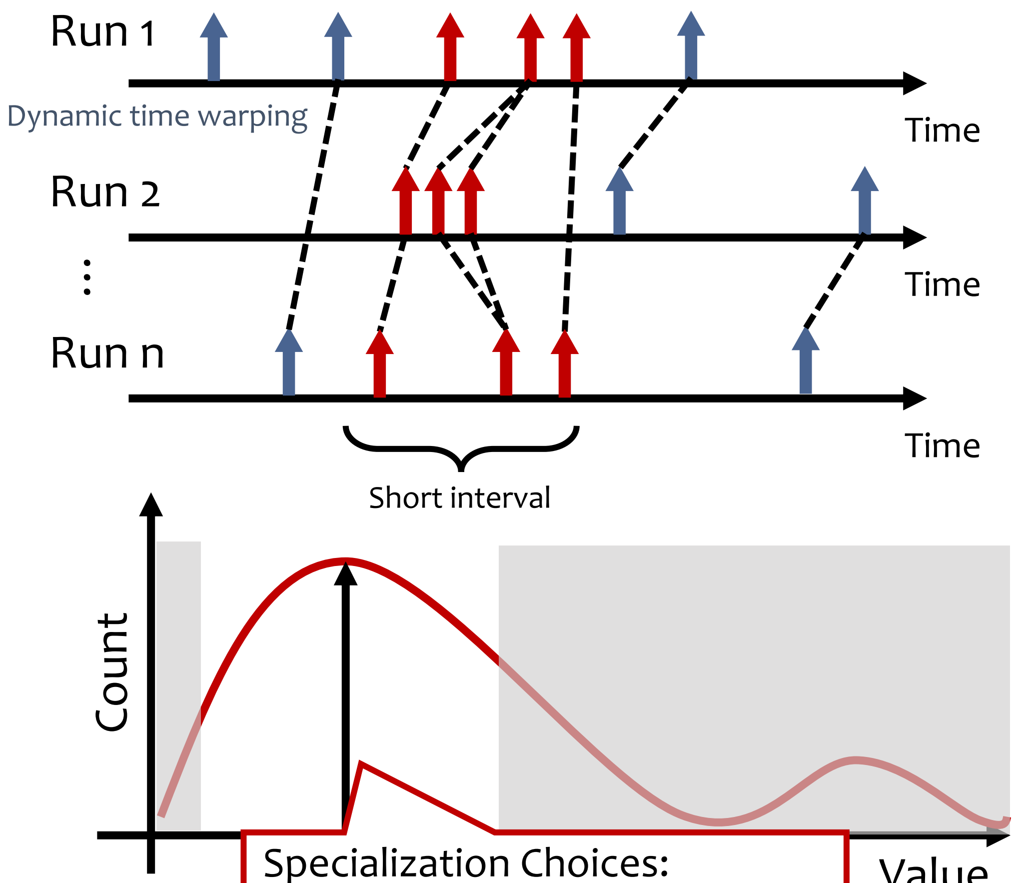
Perf Counters: $Memory\ BW, LLC\ Misses$

$$Memory\ BW = \frac{LLC\ Misses \times Cacheline\ Size}{\delta T}$$



Error Correction for HPCs

Traditional Solution (Offline Variance Reduction)



Specialization Choices:
- Time warping cost function
- Shape of curve
- Thresholds

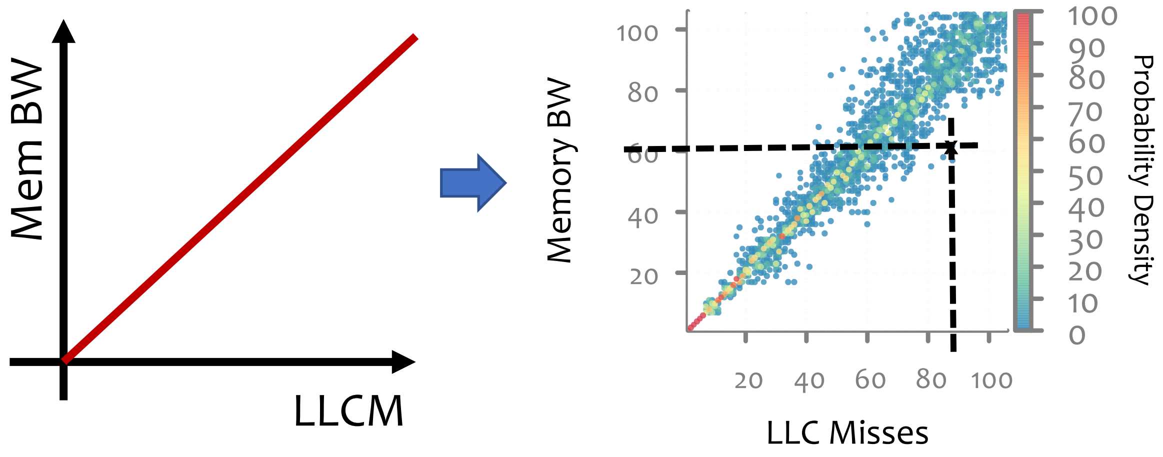
Unusable – Not real time!

BayesPerf Correction (Key Idea)

Key Insight: Different perf counters are interrelated based on system architecture

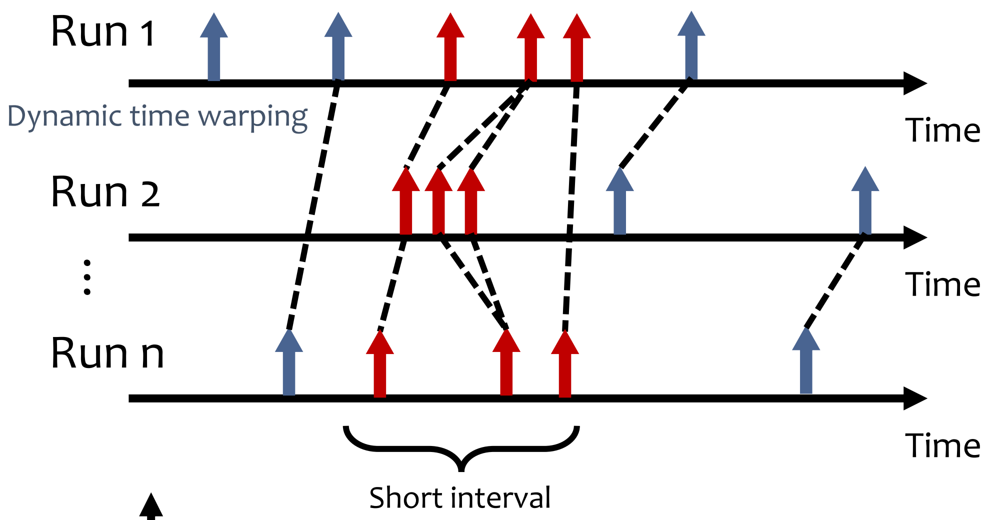
Perf Counters: $Memory\ BW, LLC\ Misses$

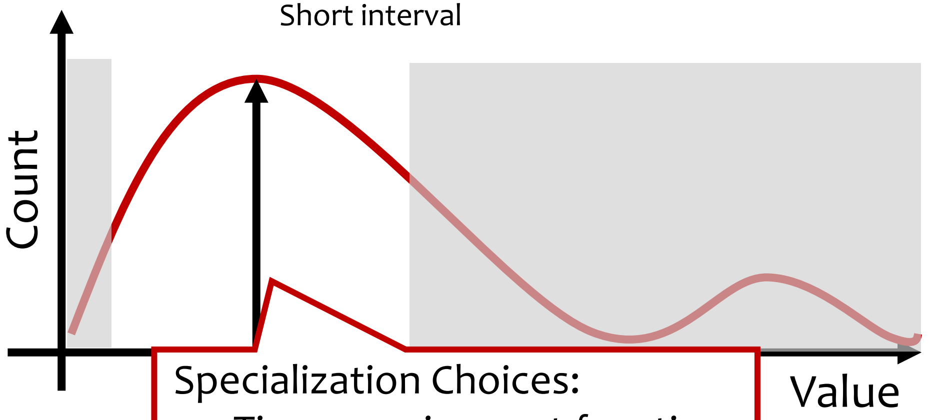
$$Memory\ BW = \frac{LLC\ Misses \times Cacheline\ Size}{\delta T}$$



Error Correction for HPCs

Traditional Solution (Offline Variance Reduction)





Specialization Choices:
- Time warping cost function
- Shape of curve
- Thresholds

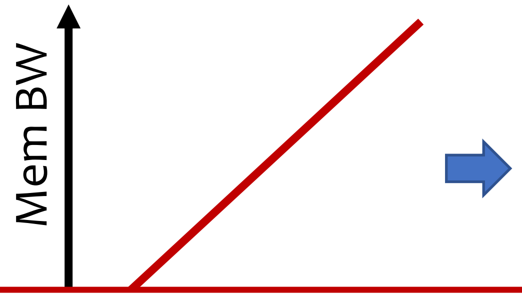
Unusable – Not real time!

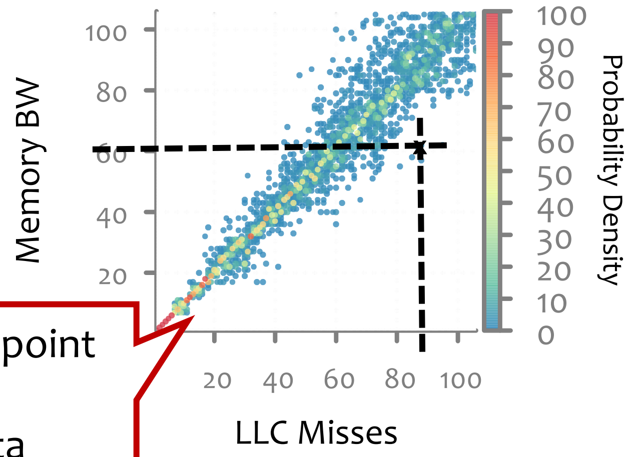
BayesPerf Correction (Key Idea)

Key Insight: Different perf counters are interrelated based on system architecture

Perf Counters: $\mathit{Memory\ BW}, \mathit{LLC\ Misses}$

$$\mathit{Memory\ BW} = \frac{\mathit{LLC\ Misses} \times \mathit{Cacheline\ Size}}{\delta T}$$





- **Quantify** uncertainty the data point
  - Uncertainty ~ Color
- **Correct:** MLE to "denoise" data
- Does not require ground truth

Stationary BayesPerf: The PGM Model

Question 1: How do we define the HPC distribution for a short interval of time?

Stationary BayesPerf: The PGM Model

Question 1: How do we define the HPC distribution for a short interval of time?

Noisy values of HPCs C'_1 … C'_n

Measured

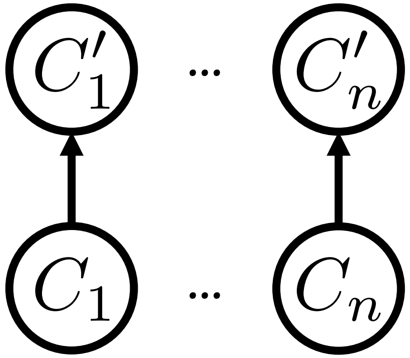
Stationary BayesPerf: The PGM Model

Question 1: How do we define the HPC distribution for a short interval of time?

$$C_i' \sim C_i + \frac{\alpha\sigma(\bar{C}_i)}{\sqrt{N}} Student(\nu = N - 1)$$

Noisy values of HPCs
Measured

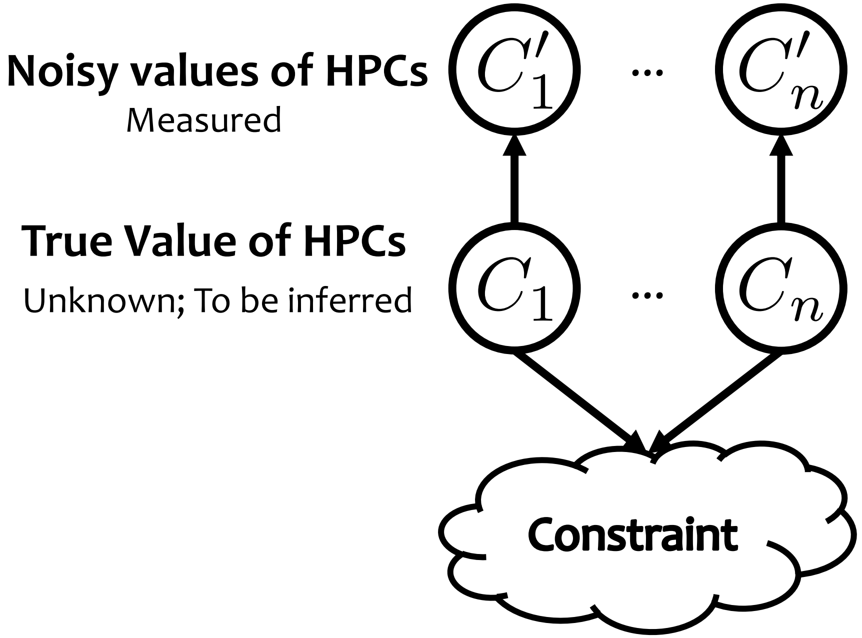
True Value of HPCs
Unknown; To be inferred



Stationary BayesPerf: The PGM Model

Question 1: How do we define the HPC distribution for a short interval of time?

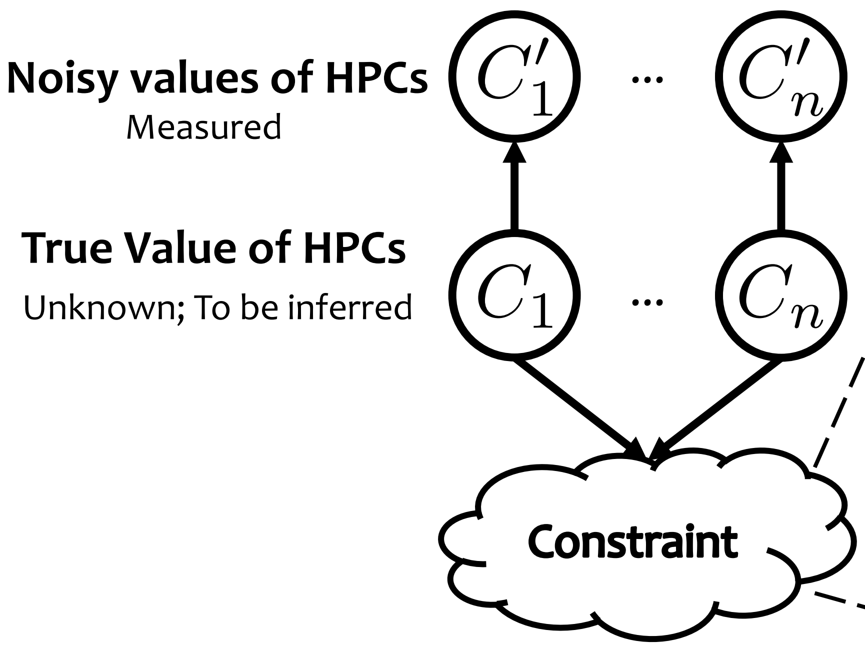
$$C_i' \sim C_i + \frac{\alpha\sigma(\bar{C}_i)}{\sqrt{N}} Student(\nu = N-1)$$

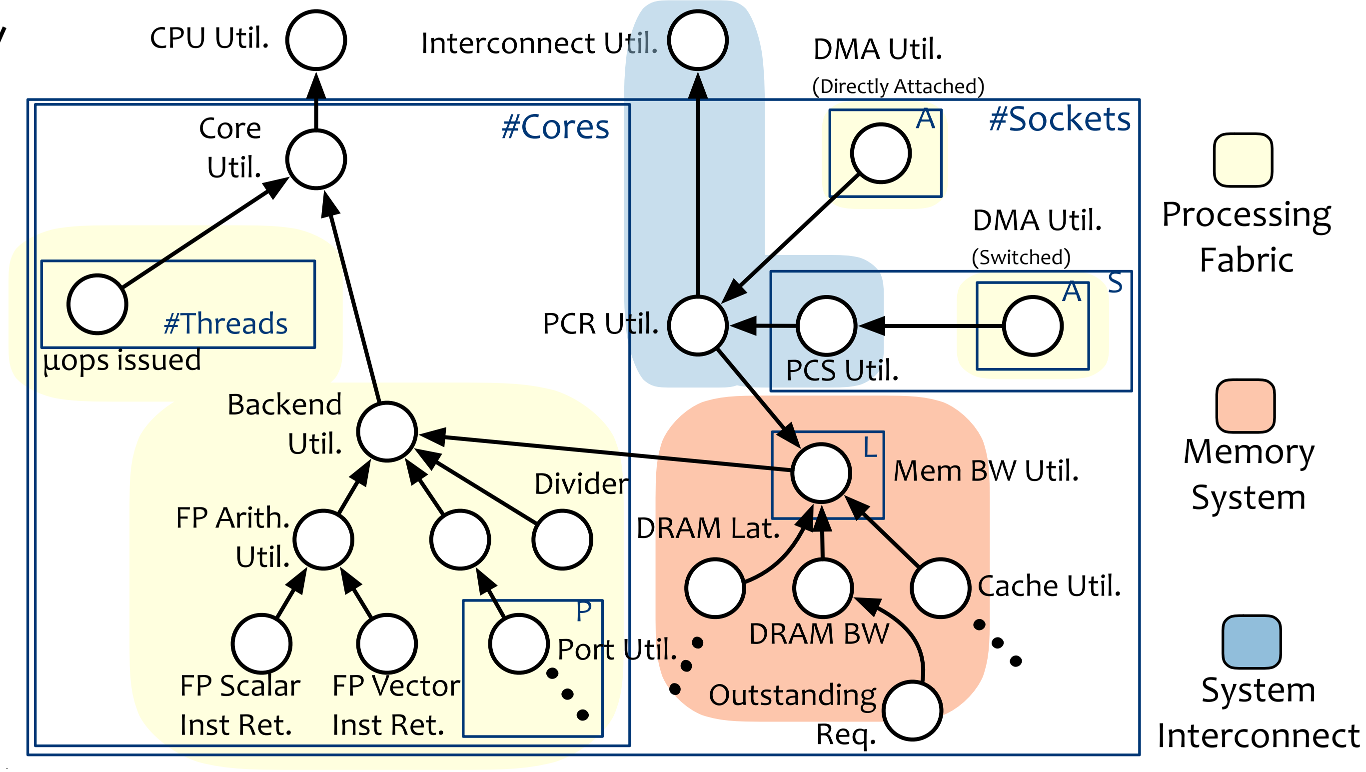


Stationary BayesPerf: The PGM Model

Question 1: How do we define the HPC distribution for a short interval of time?

$$C_i' \sim C_i + \frac{\alpha\sigma(\bar{C}_i)}{\sqrt{N}} Student(\nu = N - 1)$$





- Scalable, general & works for real processors
  - x86 (Intel), ppc64 (IBM)
  - Based on Intel's "Top-Down Microarchitectural Analysis" in VTune
- Parse BN automatically from per µ-arch listing in Linux Source Tree
  - Contributed by vendors to Linux

Dynamic BayesPerf: Scheduling Counters

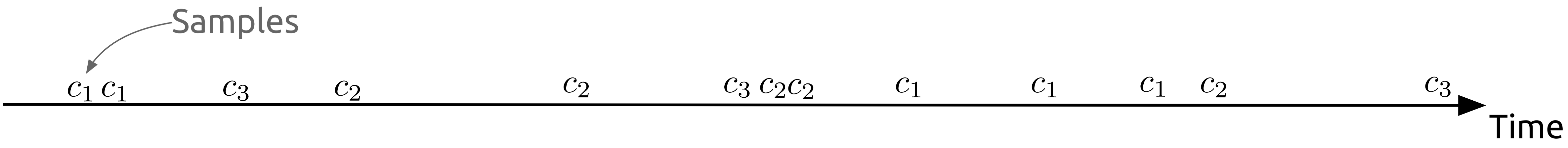
Question 2: How do we track HPC measurements over larger intervals of time?

Hypothetical example: Measure events $\{e_a, e_b, e_c, e_d, e_e, e_f\}$ on counters $\{c_1, c_2, c_3\}$

Dynamic BayesPerf: Scheduling Counters

Question 2: How do we track HPC measurements over larger intervals of time?

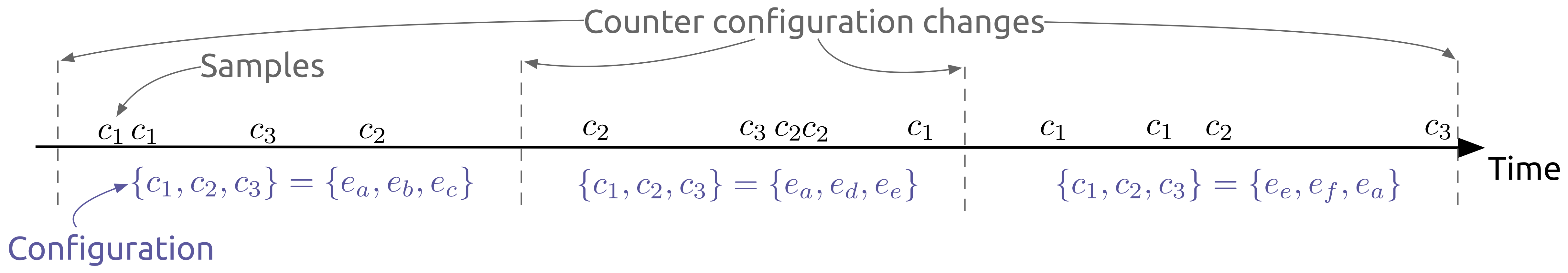
Hypothetical example: Measure events $\{e_a, e_b, e_c, e_d, e_e, e_f\}$ on counters $\{c_1, c_2, c_3\}$



Dynamic BayesPerf: Scheduling Counters

Question 2: How do we track HPC measurements over larger intervals of time?

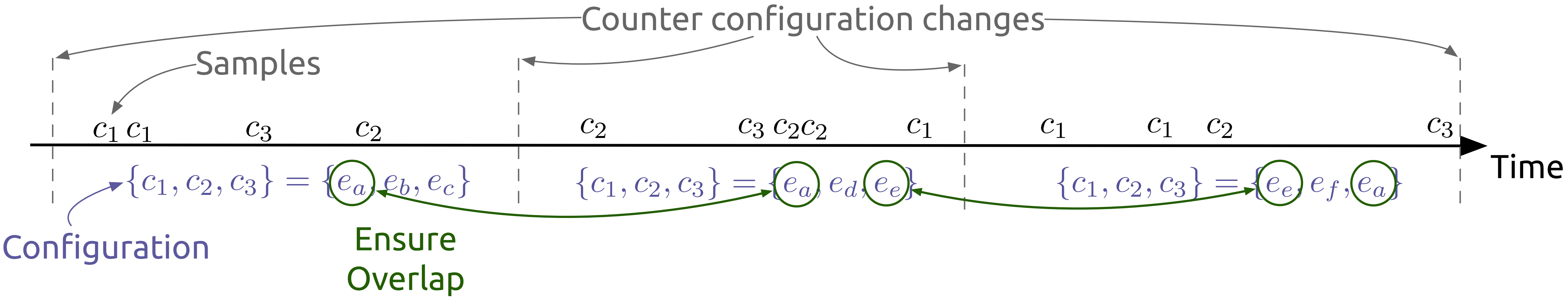
Hypothetical example: Measure events $\{e_a, e_b, e_c, e_d, e_e, e_f\}$ on counters $\{c_1, c_2, c_3\}$



Dynamic BayesPerf: Scheduling Counters

Question 2: How do we track HPC measurements over larger intervals of time?

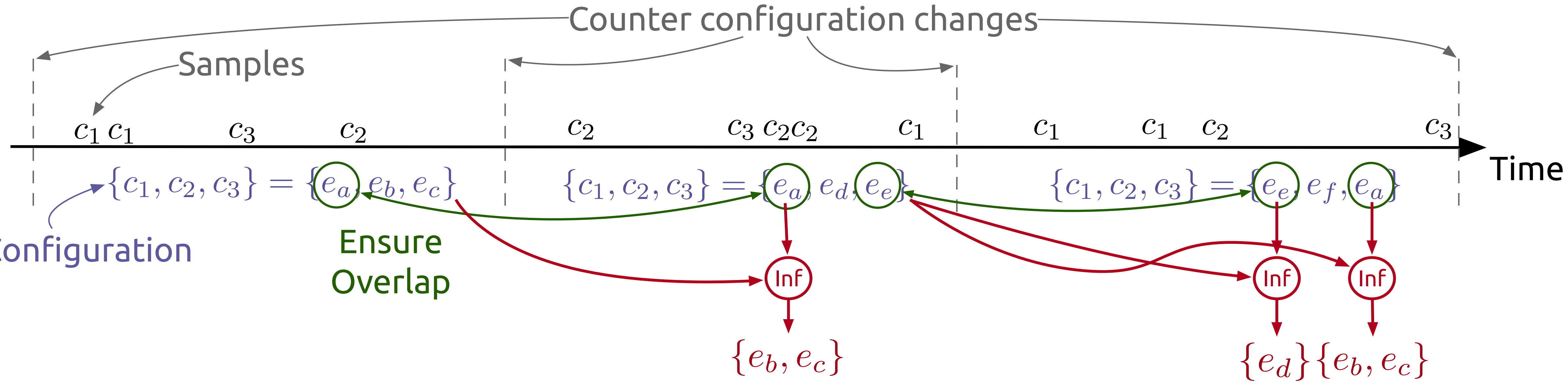
Hypothetical example: Measure events $\{e_a, e_b, e_c, e_d, e_e, e_f\}$ on counters $\{c_1, c_2, c_3\}$



Dynamic BayesPerf: Scheduling Counters

Question 2: How do we track HPC measurements over larger intervals of time?

Hypothetical example: Measure events $\{e_a, e_b, e_c, e_d, e_e, e_f\}$ on counters $\{c_1, c_2, c_3\}$

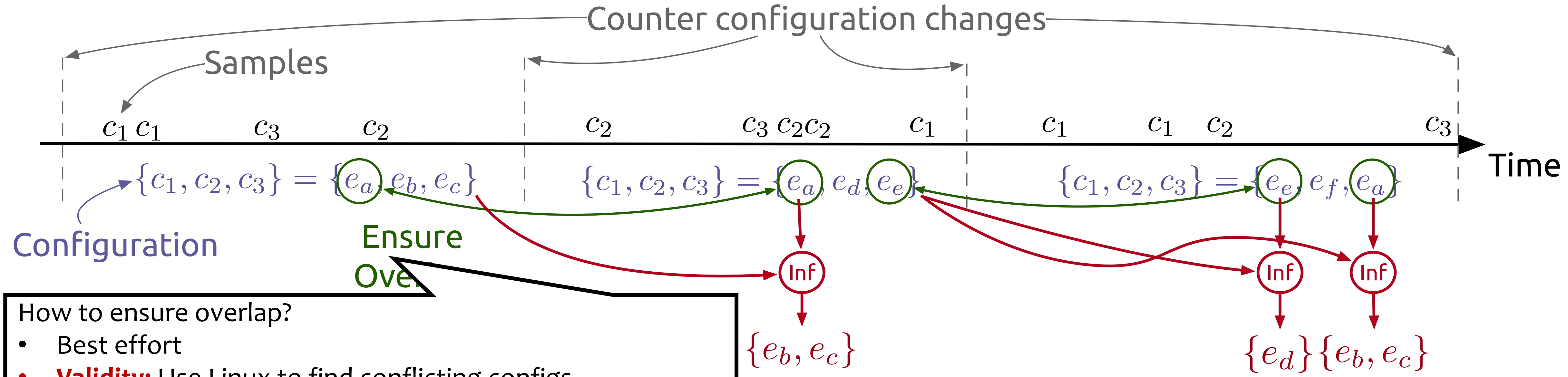


Effectively, with time BayesPerf is "inferring" all 6 counters in every interval

Dynamic BayesPerf: Scheduling Counters

Question 2: How do we track HPC measurements over larger intervals of time?

Hypothetical example: Measure events $\{e_a, e_b, e_c, e_d, e_e, e_f\}$ on counters $\{c_1, c_2, c_3\}$



How to ensure overlap?

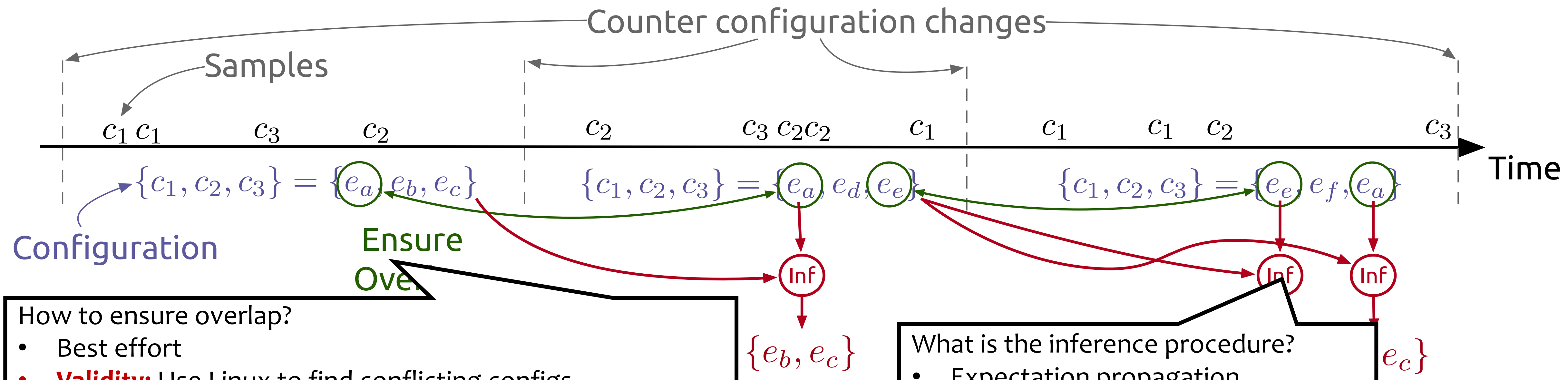
- Best effort
- **Validity:** Use Linux to find conflicting configs
- **Overlap:** Remove nodes from combined Markov Blanket of configurations across consecutive time slices

Effectively, with time BayesPerf is "inferring" all 6 counters in every interval

Dynamic BayesPerf: Scheduling Counters

Question 2: How do we track HPC measurements over larger intervals of time?

Hypothetical example: Measure events $\{e_a, e_b, e_c, e_d, e_e, e_f\}$ on counters $\{c_1, c_2, c_3\}$



How to ensure overlap?

- Best effort
- **Validity:** Use Linux to find conflicting configs
- **Overlap:** Remove nodes from combined Markov Blanket of configurations across consecutive time slices

What is the inference procedure?

- Expectation propagation
- Parallelize inference across configuration changes

Effectively, with time BayesPerf is "inferring" all 6 counters in every interval

Training the BayesPerf Model

- Does this model require training? – Yes
  - The measurement noise model: Parameter α_i

Training the BayesPerf Model

- Does this model require training? – Yes
  - The measurement noise model: Parameter α_i

- In this paper: Make BayesPerf model application agnostic
  - Assume a normal prior
  - Compute MLE of α_i from the same data as C_i
  - “Works well enough”

Training the BayesPerf Model

- Does this model require training? – Yes
  - The measurement noise model: Parameter α_i

- In this paper: Make BayesPerf model application agnostic
  - Assume a normal prior
  - Compute MLE of α_i from the same data as C_i
  - "Works well enough"

- In general: BayesPerf can be trained with representative workload set
  - Train BayesPerf model using backpropagation – [ICML 2020]
  - What if error model is not Student-t? – DNNs

Inductive-bias-driven Reinforcement Learning for Efficient Schedules in Heterogeneous Clusters

Subho S. Banerjee [1] Saurabh Jha [1] Zbigniew T. Kalbarczyk [1] Ravishankar K. Iyer [1]

Abstract

The problem of scheduling of workloads onto heterogeneous processors (e.g., CPUs, GPUs, FPGAs) is of fundamental importance in modern data centers. Current system schedulers rely on application/system-specific heuristics that have to be built on a case-by-case basis. Recent work has demonstrated ML techniques for automating the heuristic search by using black-box approaches which require significant training data and time, which make them challenging to use in practice. This paper presents Symphony, a scheduling framework that addresses the challenge in two ways: (i) a domain-driven Bayesian reinforcement learning (RL) model for scheduling, which inherently models the resource dependencies identified from the system architecture; and (ii) a sampling-based technique to compute the gradients of a Bayesian model without performing full probabilistic inference. Together, these techniques reduce both the amount of training data and the time required to produce scheduling policies that significantly outperform black-box approaches by up to 2.2×.

1. Introduction

The problem of scheduling of workloads on heterogeneous processing fabrics (i.e., accelerated datacenters including GPUs, FPGAs, and ASICs, e.g., Asanović (2014); Shao & Brooks (2015)), is at its core an intractable NP-hard problem (Mastrolilli & Svensson, 2008; 2009). System schedulers generally rely on application- and system-specific heuristics with extensive domain-expert-driven tuning of scheduling policies (e.g., Isard et al. (2009); Giceva et al. (2014); Lyerly et al. (2018); Mars et al. (2011); Mars & Tang (2013); Ousterhout et al. (2013); Xu et al. (2018); Yang et al. (2013); Zhang et al. (2014); Zhuravlev et al. (2010); Zaharia et al. (2010)). Such heuristics are difficult to generate, as variations across applications and system configurations mean that significant amounts of time and money must be spent in painstaking heuristic searches. Recent work has demonstrated machine learning (ML) techniques (Delimitrou & Kozyrakis, 2013; 2014; Mao et al., 2016; 2018) for automating heuristic searches by using black-box approaches which require significant training data and time, making them challenging to use in practice.

This paper presents Symphony, a scheduling framework that addresses the challenge in two ways: (i) we use a domain-guided Bayesian-model-based partially observable Markov decision process (POMDP) (Astrom, 1965; Kaelbling et al., 1998) to decrease the amount of training data (i.e., sampled trajectories); and (ii) a sampling-based technique that allows one to compute the gradients of a Bayesian model without performing full probabilistic inference. We thus, significantly reduce the costs of (i) running a large heterogeneous computing system that uses an efficient scheduling policy; and (ii) training the policy itself.

Reducing Training Data. State-of-the-art methods for choosing an optimal action in POMDPs rely on training of neural networks (NNs) (Mnih et al., 2016; Dhariwal et al., 2017). As these approaches are model-free, training of the NN requires large quantities of data and time to compute meaningful policies. In contrast, we provide an inductive bias for the reinforcement learning (RL) agent by encoding domain knowledge as a Bayesian model that can infer the latent state from observations, while at the same time leveraging the scalability of deep learning methods through end-to-end gradient descent. In the case of scheduling, our inductive bias is a set of statistical relationships between measurements from microarchitectural monitors (Dreyer & Alpert, 1997). To the best of our knowledge, this is the first paper to exploit those relationships and measurements to infer resource utilization in the system (i.e., latent state) to build RL-based scheduling polices.

Reducing Training Time. The addition of the inductive bias, while making the training process less data-hungry (i.e., requiring fewer workload executions to train the model), comes at the cost of additional training time: the cost of performing full-Bayesian inference at every training step (Dagum & Luby, 1993; Russell et al., 1995; Binder

[1] University of Illinois at Urbana-Champaign, USA. Correspondence to: Subho S. Banerjee <ssbaner2@illinois.edu>.

Proceedings of the 37th International Conference on Machine Learning, Online, PMLR 119, 2020. Copyright 2020 by the author(s).

The BayesPerf System

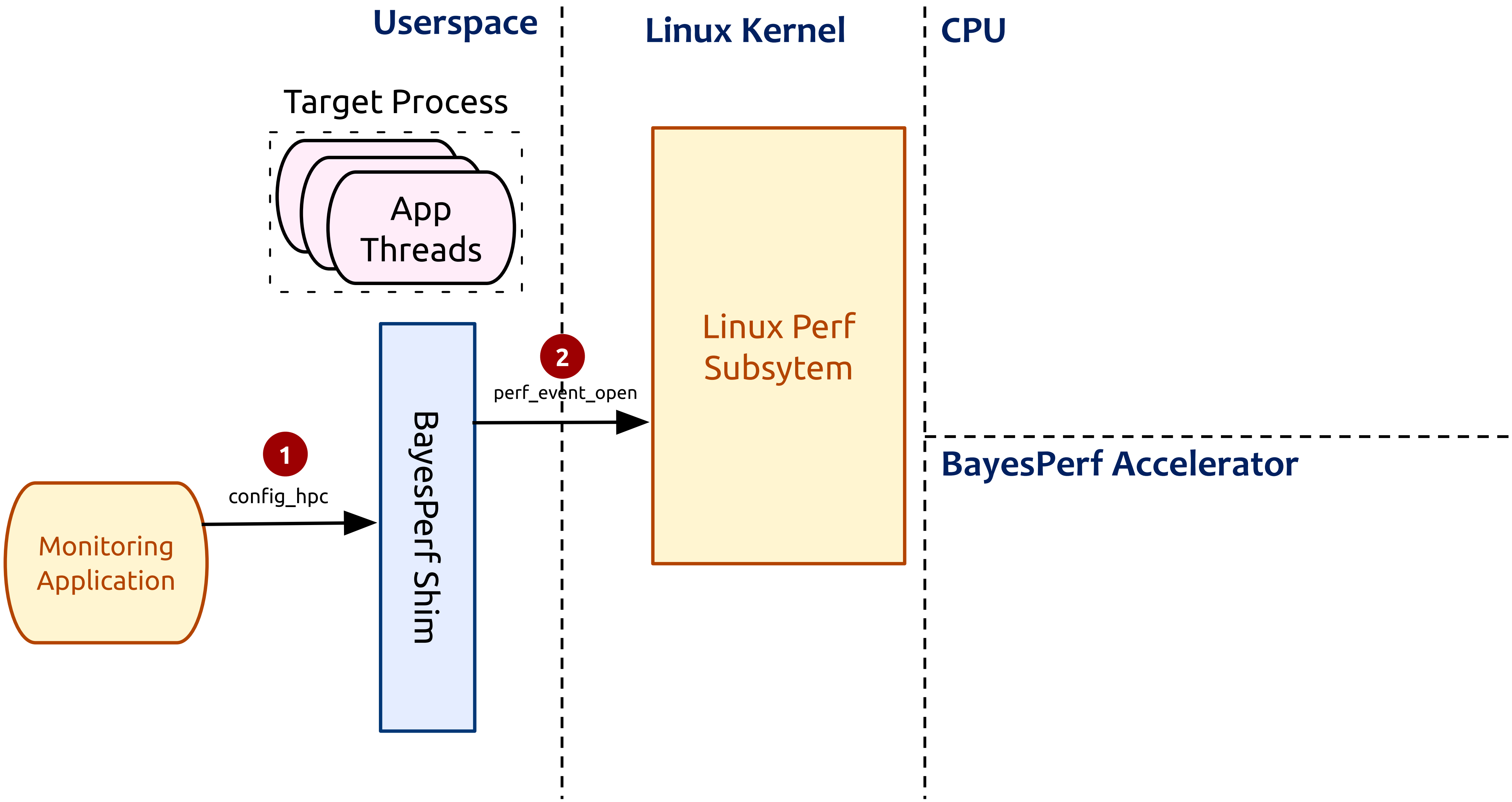
Userspace

Linux Kernel

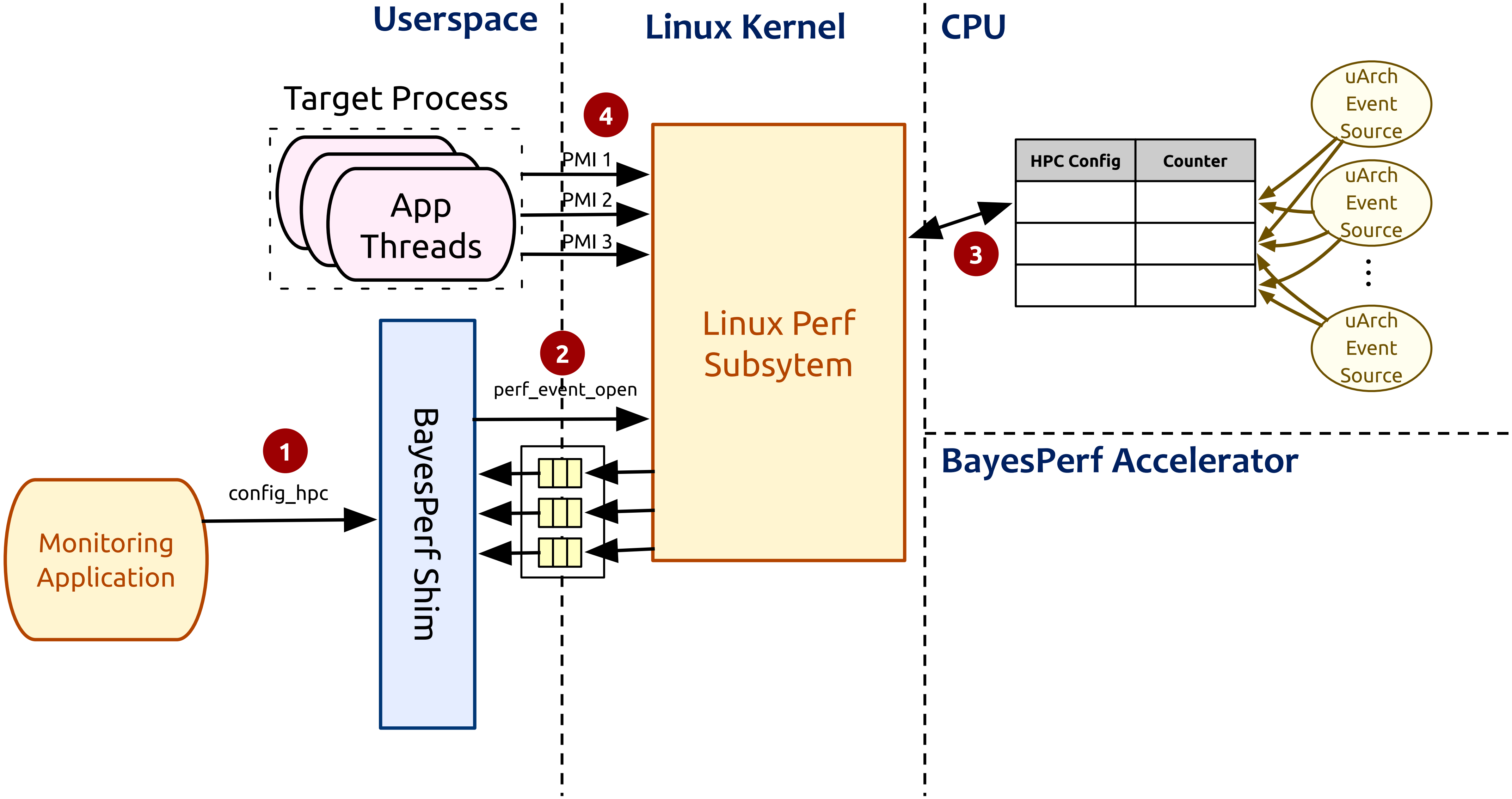
CPU

BayesPerf Accelerator

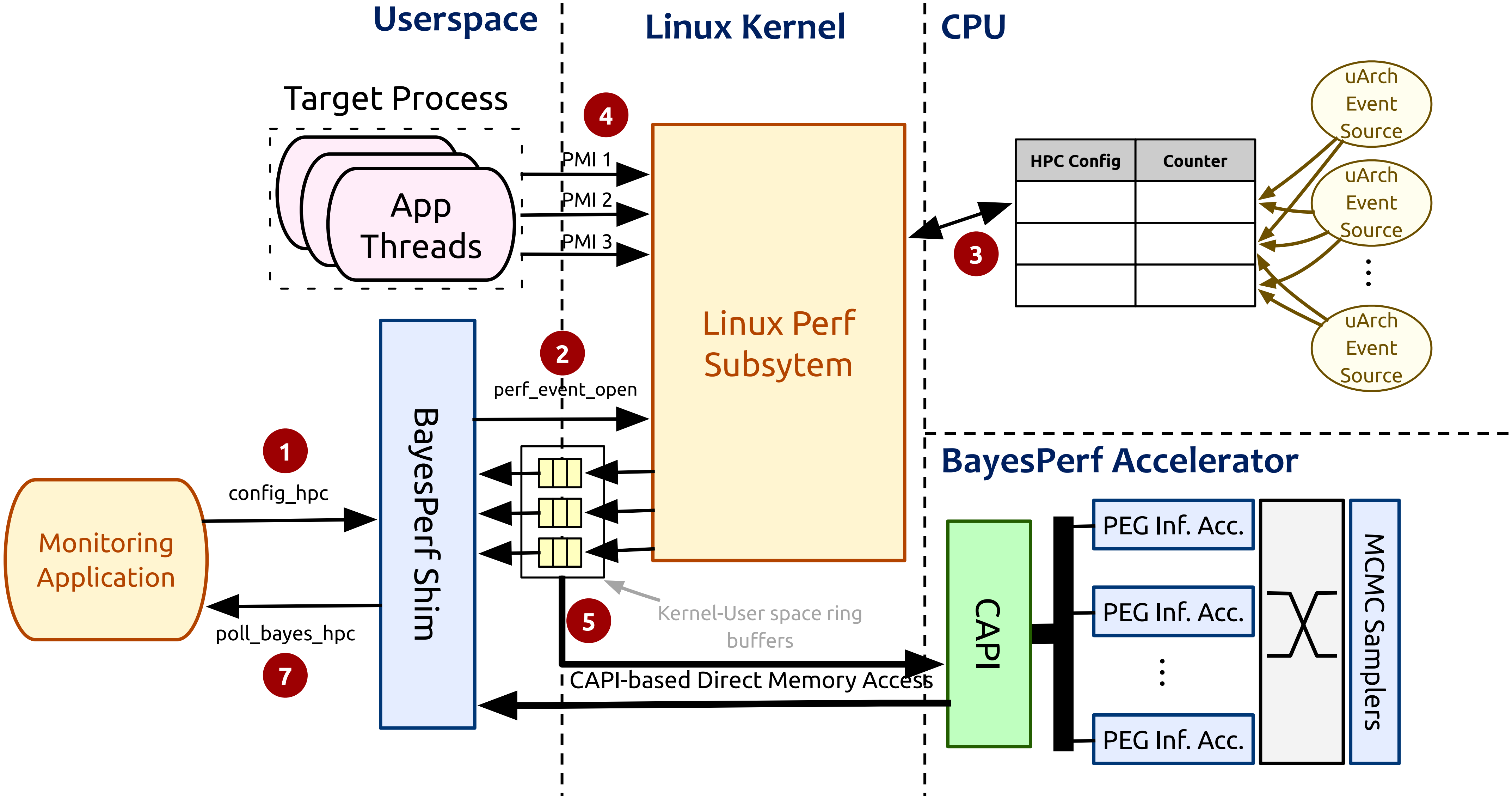
The BayesPerf System



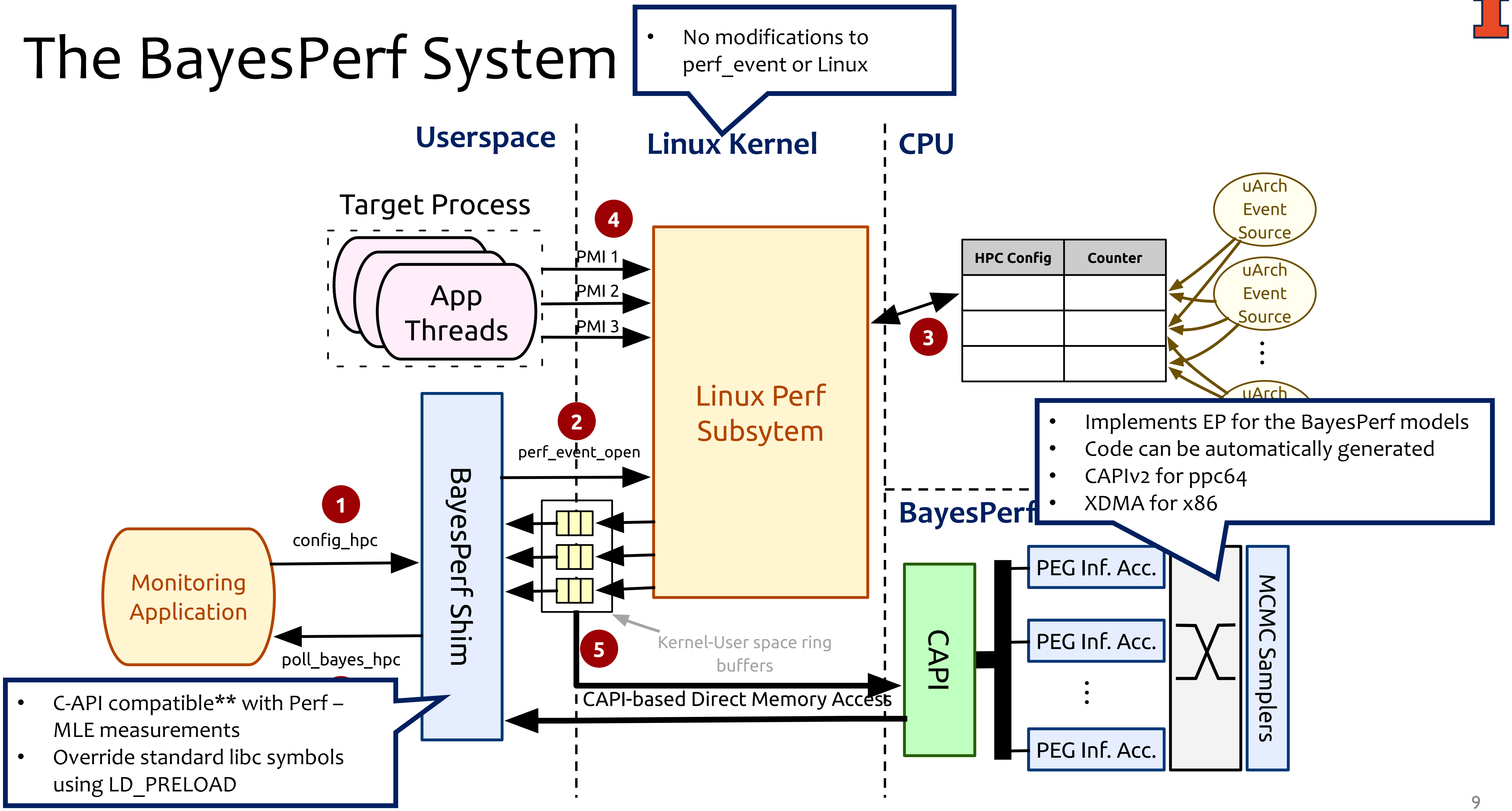
The BayesPerf System



The BayesPerf System



The BayesPerf System



The BayesPerf Accelerator

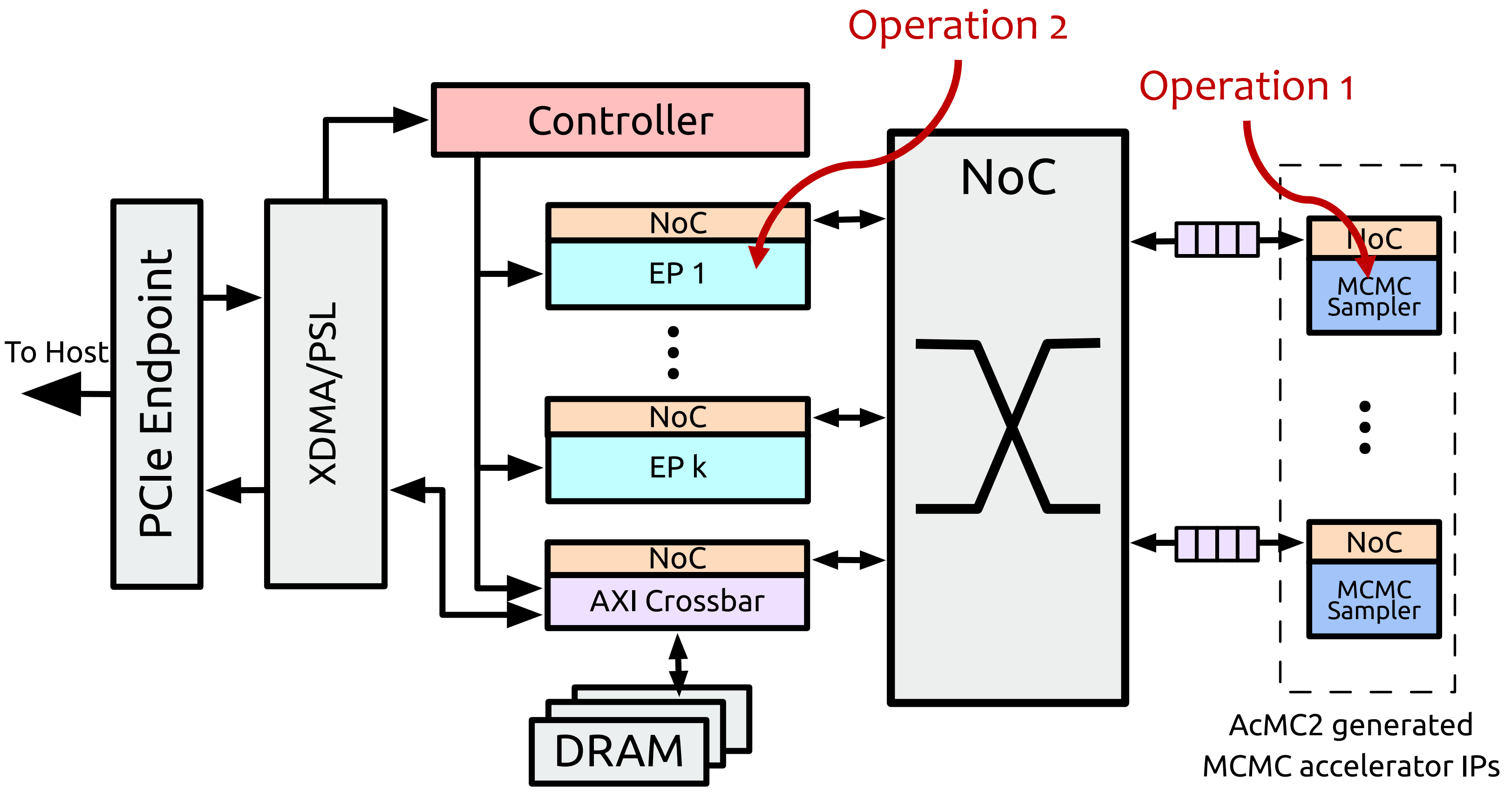
Accelerates a Bayesian inference algorithm called Expectation Propagation

- Core operation 1: MCMC Sampling **(85+% of runtime)**
  - Core operation 2: Vector Dot Product + Update
    - Keep data in flight between Op1 and Op2

The BayesPerf Accelerator

Accelerates a Bayesian inference algorithm called Expectation Propagation

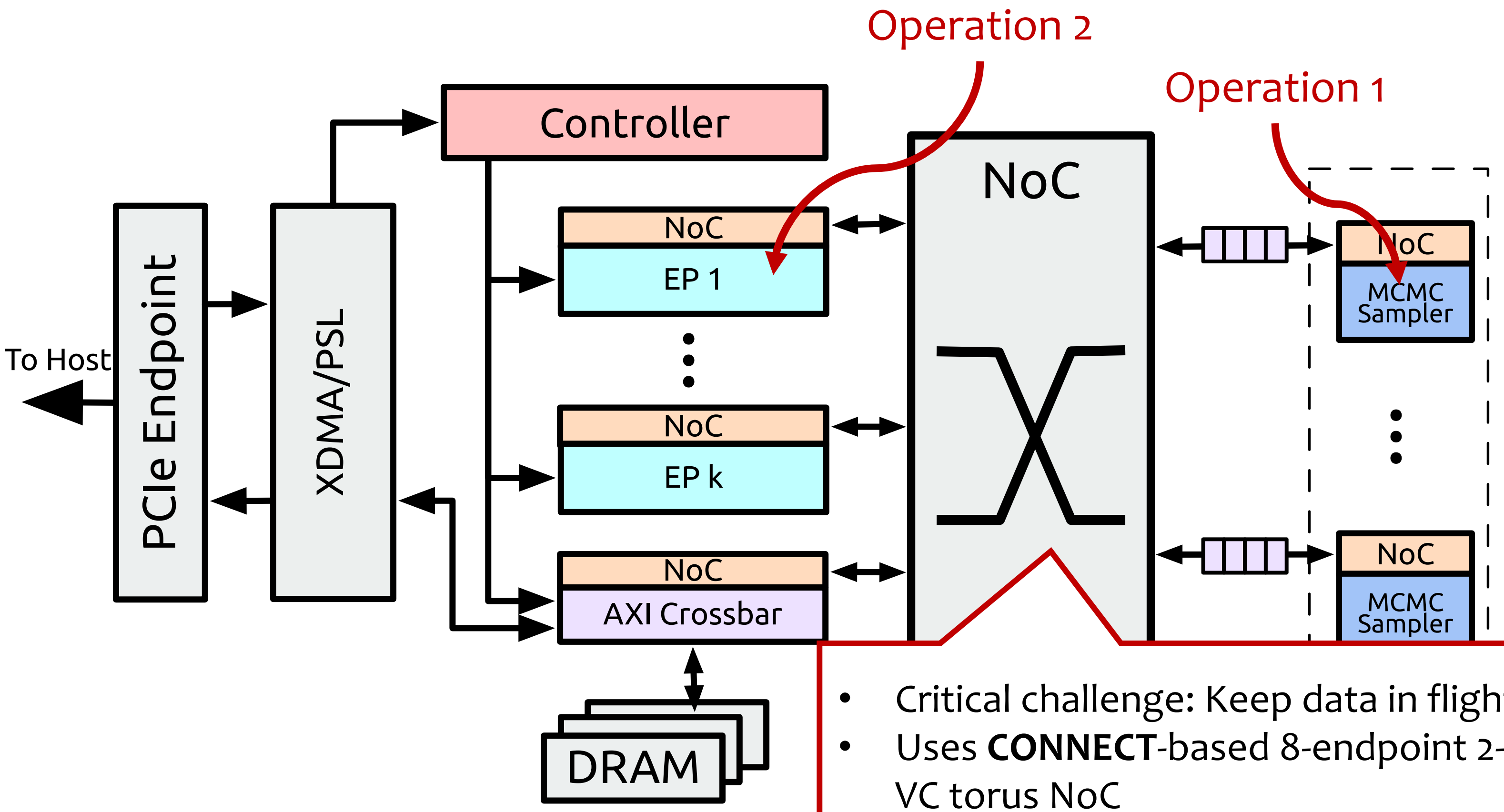
- Core operation 1: MCMC Sampling **(85+% of runtime)**
  - Core operation 2: Vector Dot Product + Update
    - Keep data in flight between Op1 and Op2



The BayesPerf Accelerator

Accelerates a Bayesian inference algorithm called Expectation Propagation

- Core operation 1: MCMC Sampling **(85+% of runtime)**
  - Core operation 2: Vector Dot Product + Update
    - Keep data in flight between Op1 and Op2

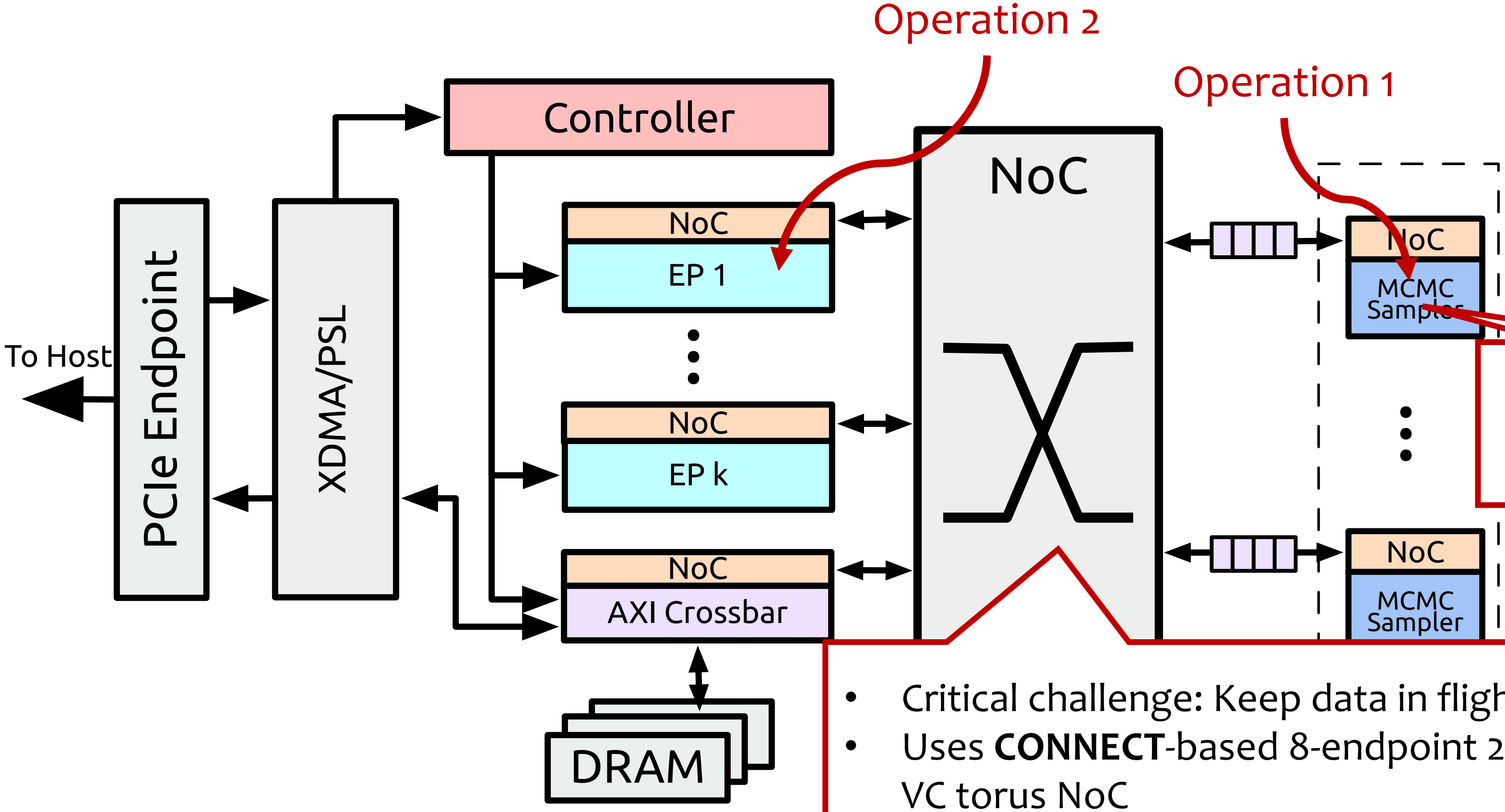


- Critical challenge: Keep data in flight
- Uses **CONNECT**-based 8-endpoint 2-VC torus NoC
- 228GB/s peak bisection bandwidth

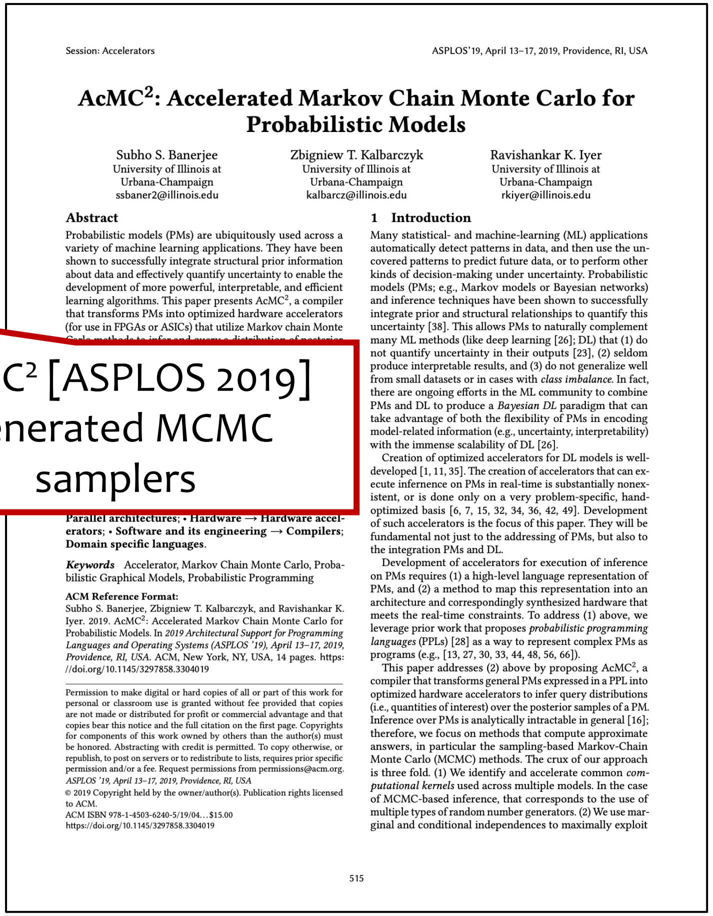
The BayesPerf Accelerator

Accelerates a Bayesian inference algorithm called Expectation Propagation

- Core operation 1: MCMC Sampling **(85+% of runtime)**
  - Core operation 2: Vector Dot Product + Update
    - Keep data in flight between Op1 and Op2



- Critical challenge: Keep data in flight
- Uses **CONNECT**-based 8-endpoint 2-VC torus NoC
- 228GB/s peak bisection bandwidth



AcMC^2: Accelerated Markov Chain Monte Carlo for Probabilistic Models

Subho S. Banerjee
University of Illinois at Urbana-Champaign
ssbaner2@illinois.edu

Zbigniew T. Kalbarczyk
University of Illinois at Urbana-Champaign
kalbarcz@illinois.edu

Ravishankar K. Iyer
University of Illinois at Urbana-Champaign
rkiyer@illinois.edu

Abstract

Probabilistic models (PMs) are ubiquitously used across a variety of machine learning applications. They have been shown to successfully integrate structural prior information about data and effectively quantify uncertainty to enable the development of more powerful, interpretable, and efficient learning algorithms. This paper presents AcMC^2, a compiler that transforms PMs into optimized hardware accelerators (for use in FPGAs or ASICs) that utilize Markov chain Monte

Parallel architectures; • **Hardware** → **Hardware accelerators**; • **Software and its engineering** → **Compilers**; **Domain specific languages**.

Keywords Accelerator, Markov Chain Monte Carlo, Probabilistic Graphical Models, Probabilistic Programming

ACM Reference Format:
Subho S. Banerjee, Zbigniew T. Kalbarczyk, and Ravishankar K. Iyer. 2019. AcMC^2: Accelerated Markov Chain Monte Carlo for Probabilistic Models. In *2019 Architectural Support for Programming Languages and Operating Systems (ASPLOS '19), April 13–17, 2019, Providence, RI, USA*. ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3297858.3304019

ACM ISBN 978-1-4503-6240-5/19/04…$15.00
https://doi.org/10.1145/3297858.3304019

1 Introduction

Many statistical- and machine-learning (ML) applications automatically detect patterns in data, and then use the uncovered patterns to predict future data, or to perform other kinds of decision-making under uncertainty. Probabilistic models (PMs; e.g., Markov models or Bayesian networks) and inference techniques have been shown to successfully integrate prior and structural relationships to quantify this uncertainty [38]. This allows PMs to naturally complement many ML methods (like deep learning [26]; DL) that (1) do not quantify uncertainty in their outputs [23], (2) seldom produce interpretable results, and (3) do not generalize well from small datasets or in cases with *class imbalance*. In fact, there are ongoing efforts in the ML community to combine PMs and DL to produce a *Bayesian DL* paradigm that can take advantage of both the flexibility of PMs in encoding model-related information (e.g., uncertainty, interpretability) with the immense scalability of DL [26].

Creation of optimized accelerators for DL models is well-developed [1, 11, 35]. The creation of accelerators that can execute infernence on PMs in real-time is substantially nonexistent, or is done only on a very problem-specific, hand-optimized basis [6, 7, 15, 32, 34, 36, 42, 49]. Development of such accelerators is the focus of this paper. They will be fundamental not just to the addressing of PMs, but also to the integration PMs and DL.

Development of accelerators for execution of inference on PMs requires (1) a high-level language representation of PMs, and (2) a method to map this representation into an architecture and correspondingly synthesized hardware that meets the real-time constraints. To address (1) above, we leverage prior work that proposes *probabilistic programming languages* (PPLs) [28] as a way to represent complex PMs as programs (e.g., [13, 27, 30, 33, 44, 48, 56, 66]).

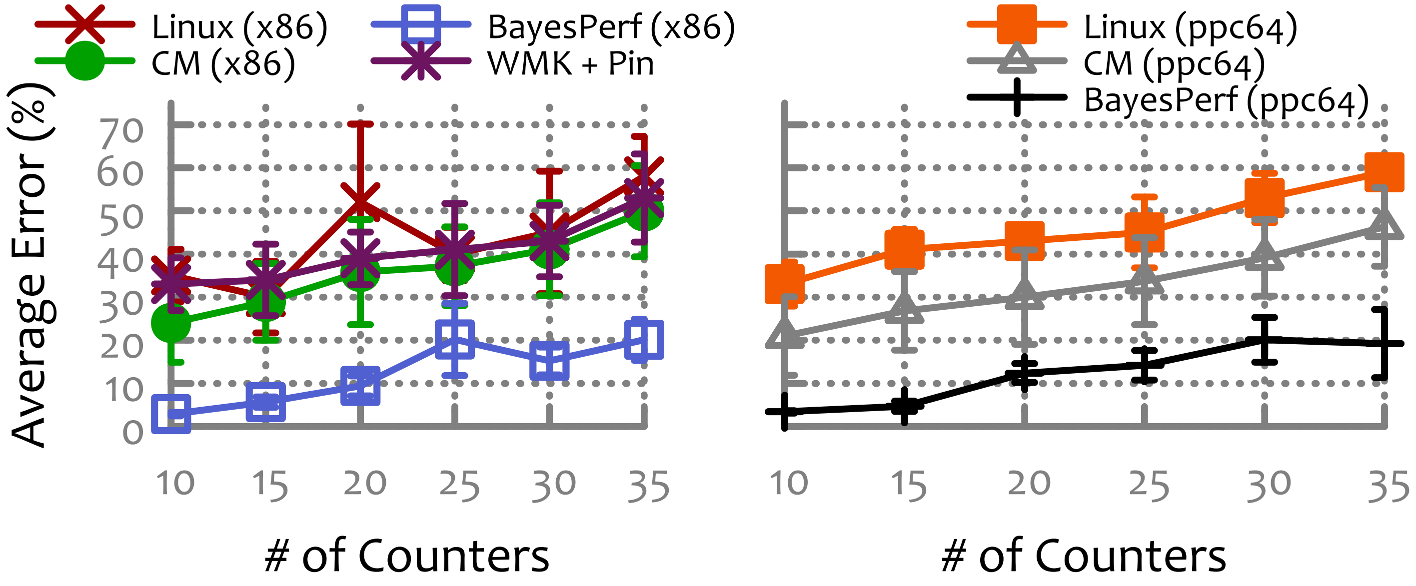
This paper addresses (2) above by proposing AcMC^2, a compiler that transforms general PMs expressed in a PPL into optimized hardware accelerators to infer query distributions (i.e., quantities of interest) over the posterior samples of a PM. Inference over PMs is analytically intractable in general [16]; therefore, we focus on methods that compute approximate answers, in particular the sampling-based Markov-Chain Monte Carlo (MCMC) methods. The crux of our approach is three fold. (1) We identify and accelerate common *computational kernels* used across multiple models. In the case of MCMC-based inference, that corresponds to the use of multiple types of random number generators. (2) We use marginal and conditional independences to maximally exploit

515

Evaluation: Error Correction Performance of BayesPerf

On average BayesPerf reduces error by as much 43.6% less error when scaling to 35 counters

[KMeans app from the HiBench benchmark suite]

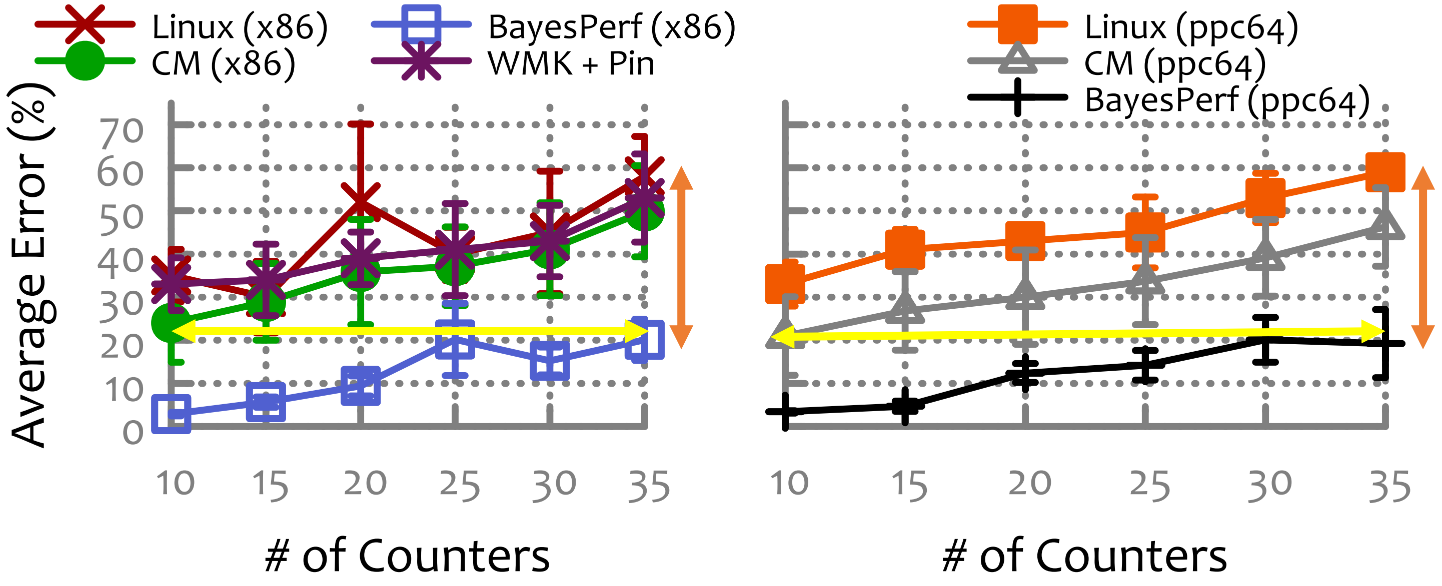


Baselines for comparison
- Linux (*) – Vanilla perf_event
- CM (*) – Counter Miner [MICRO 2018]
  - Gumbel Extreme Value Detector + Logistic Regression
- WMK+Pin – [IISWC 2008]
  - Rule-based correction

Evaluation: Error Correction Performance of BayesPerf

On average BayesPerf reduces error by as much 43.6% less error when scaling to 35 counters

[KMeans app from the HiBench benchmark suite]

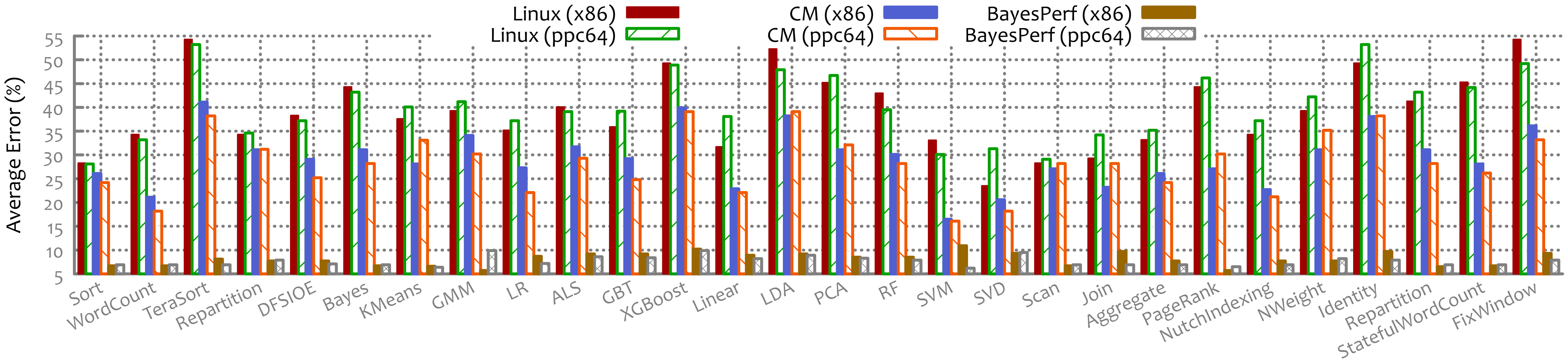


Baselines for comparison

- Linux (*) – Vanilla perf_event
- CM (*) – Counter Miner [MICRO 2018]
  - Gumbel Extreme Value Detector + Logistic Regression
- WMK+Pin – [IISWC 2008]
  - Rule-based correction

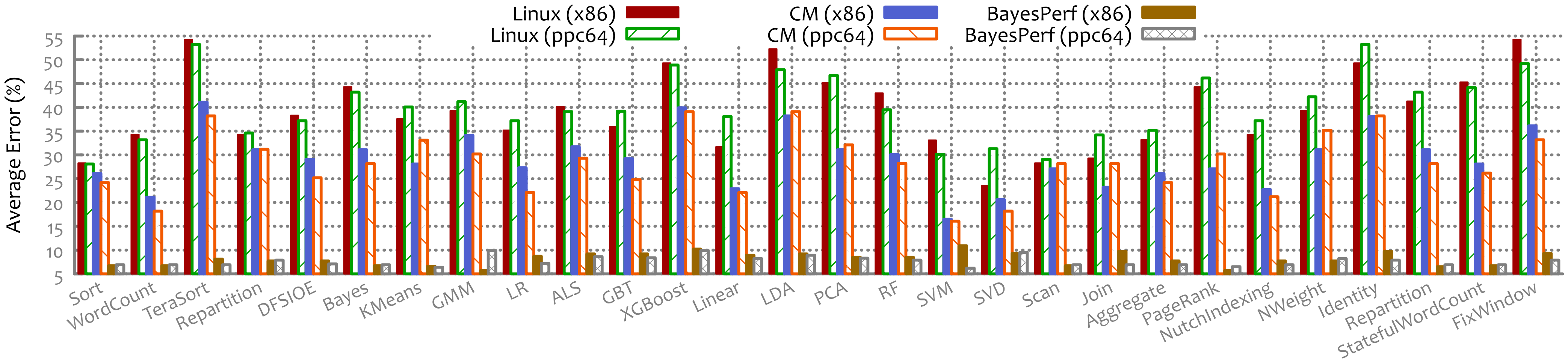
Evaluation: Error Correction Performance of BayesPerf

BayesPerf running the entire HiBench suite for 25 HPCs



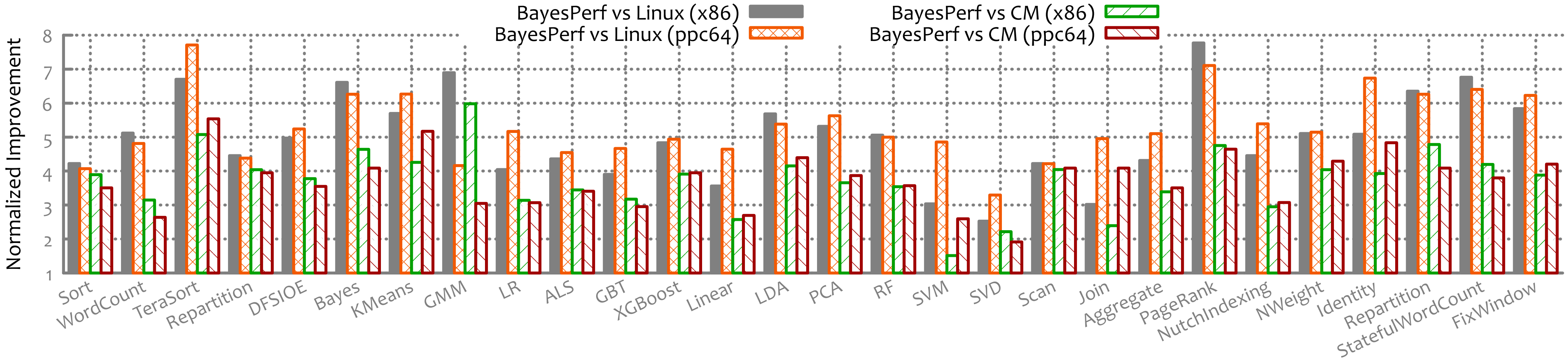
Evaluation: Error Correction Performance of BayesPerf

BayesPerf running the entire HiBench suite for 25 HPCs

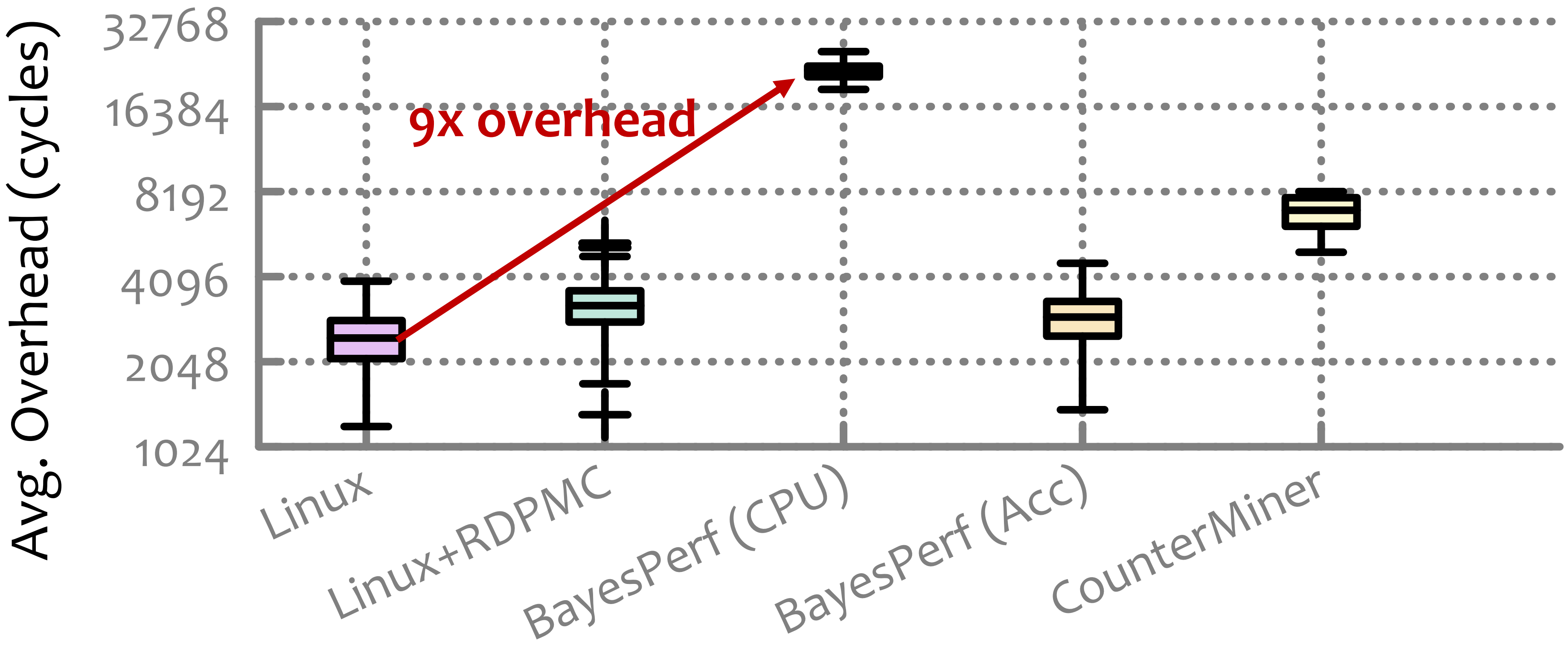


Average Improvement: **BP vs Linux = 4.9x, 5.3x** **BP vs CM = 3.6x, 3.7x**

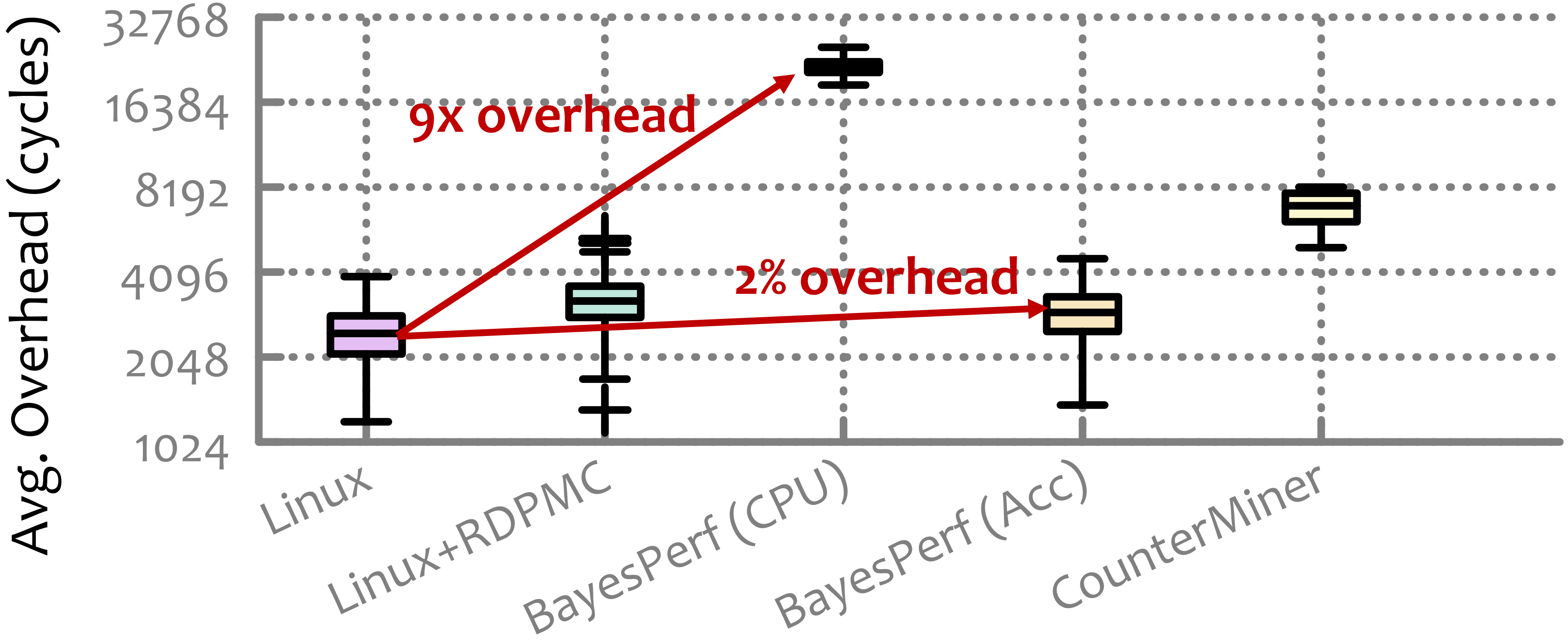
Best Improvement: **BP vs Linux = 7.8x, 7.6x** **BP vs CM = 6x, 5.4x**



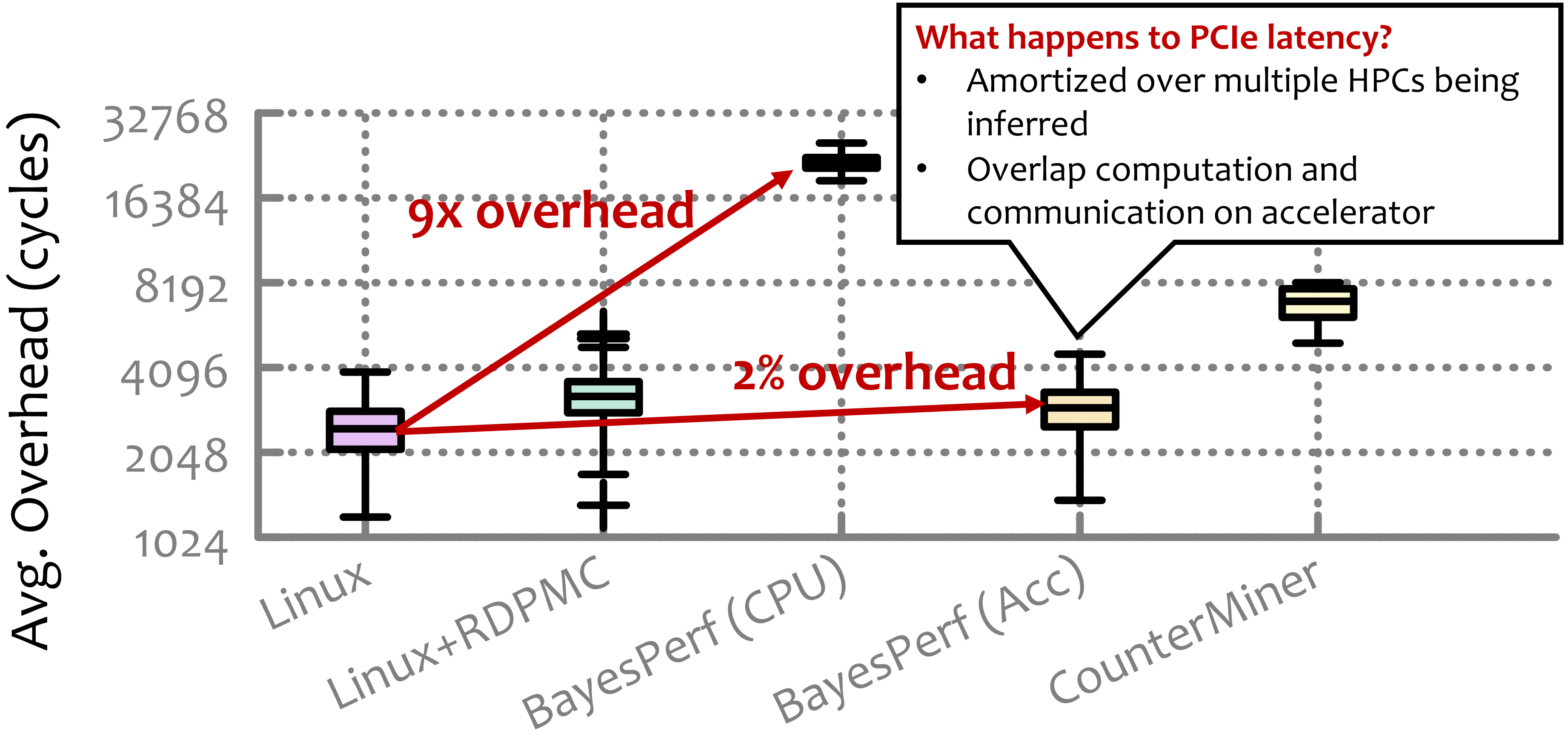
Evaluation: Overheads



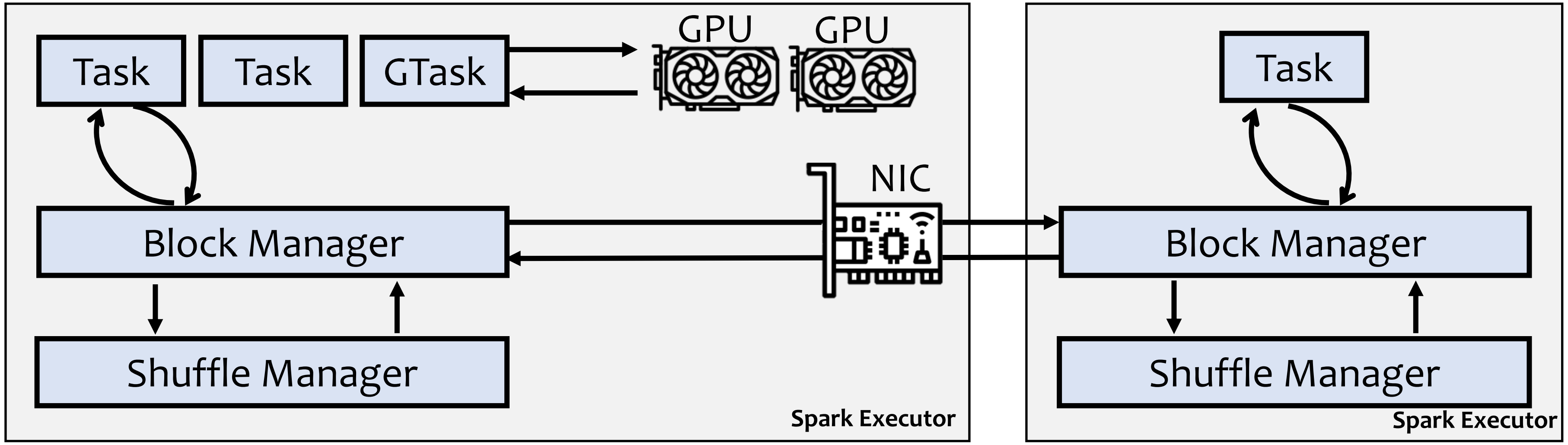
Evaluation: Overheads



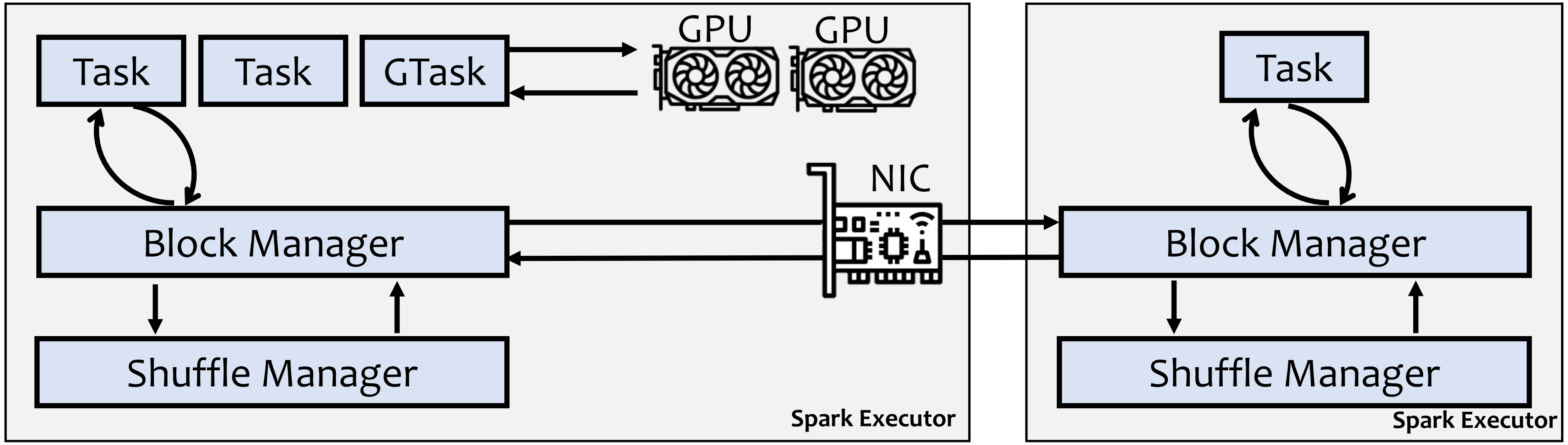
Evaluation: Overheads

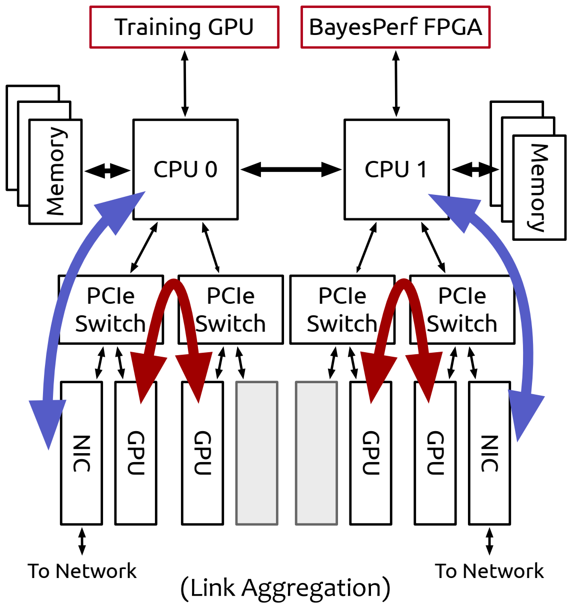


Learned Controller Case Study: Scheduling PCIe Transfers

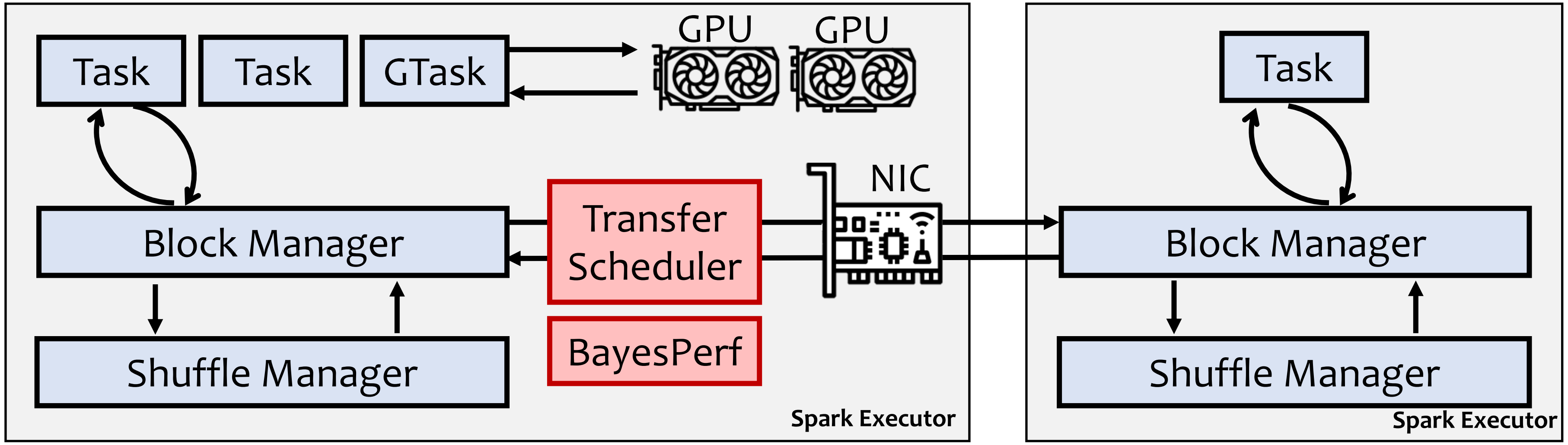


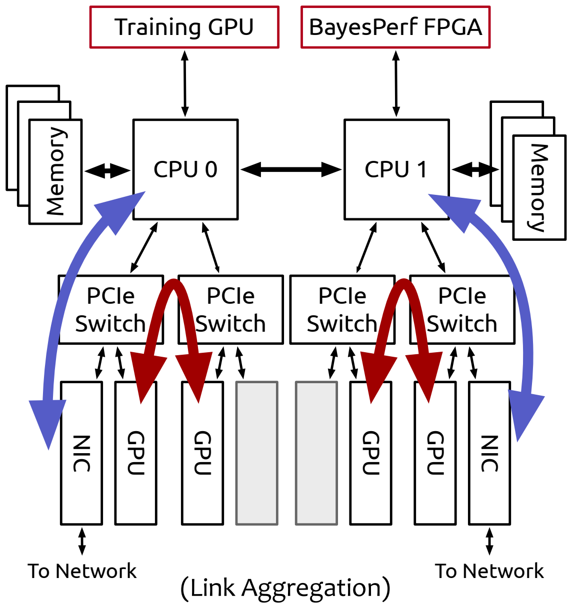
Learned Controller Case Study: Scheduling PCIe Transfers



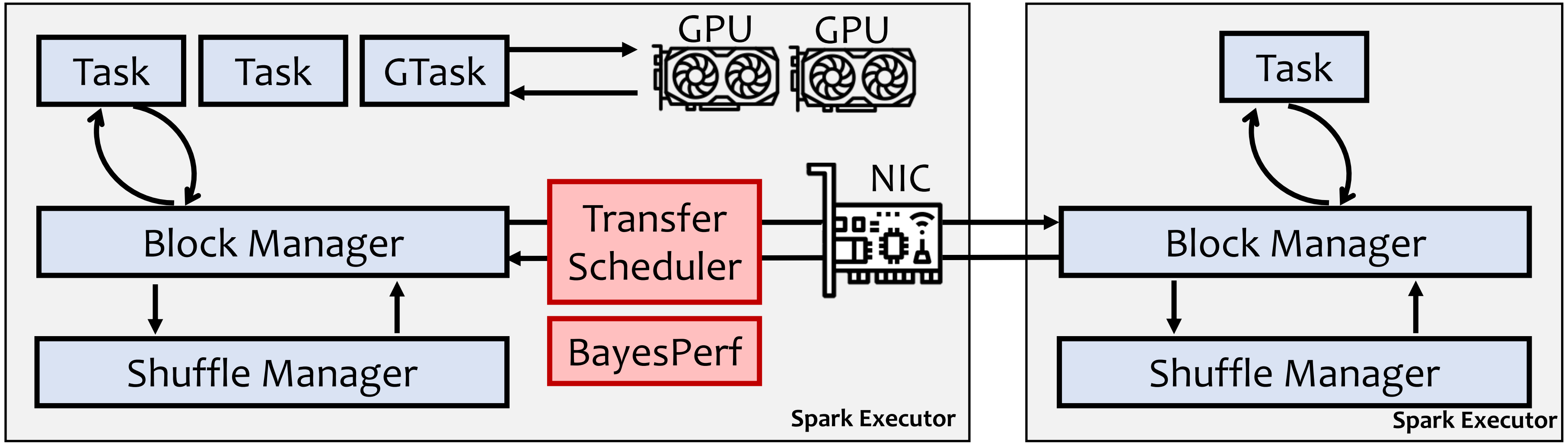


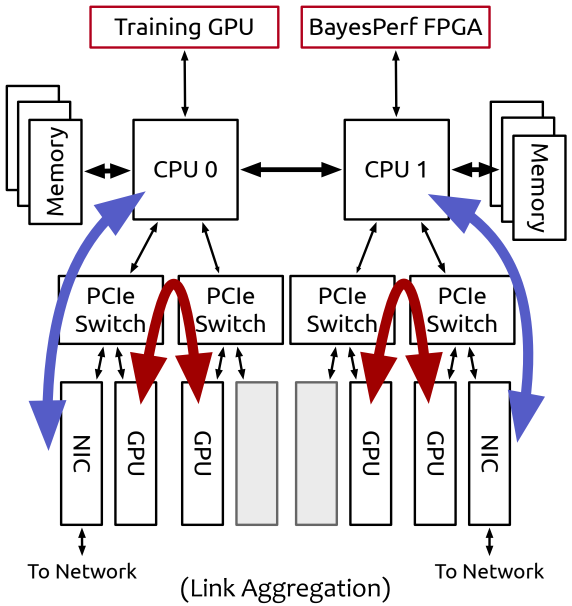
Learned Controller Case Study: Scheduling PCIe Transfers





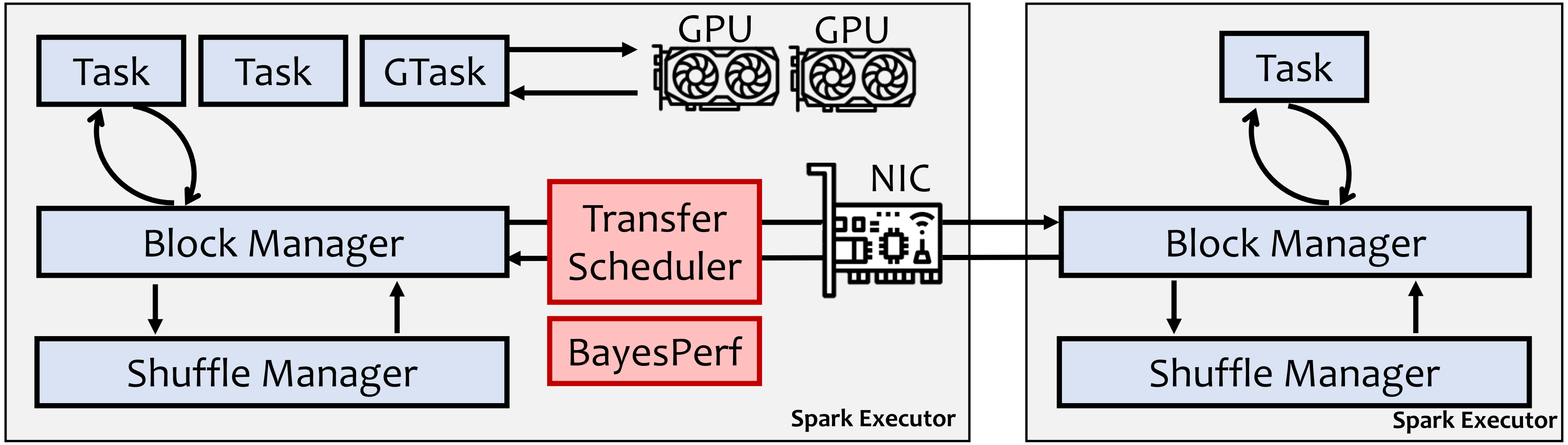
Learned Controller Case Study: Scheduling PCIe Transfers

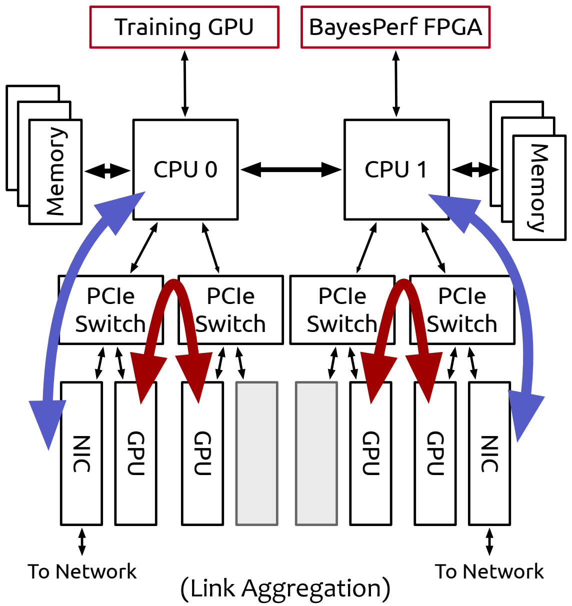




- The Transfer Scheduler is trained using: RL [Symphony - ICML2020], CF [Paragon - ASPLOS 2013]

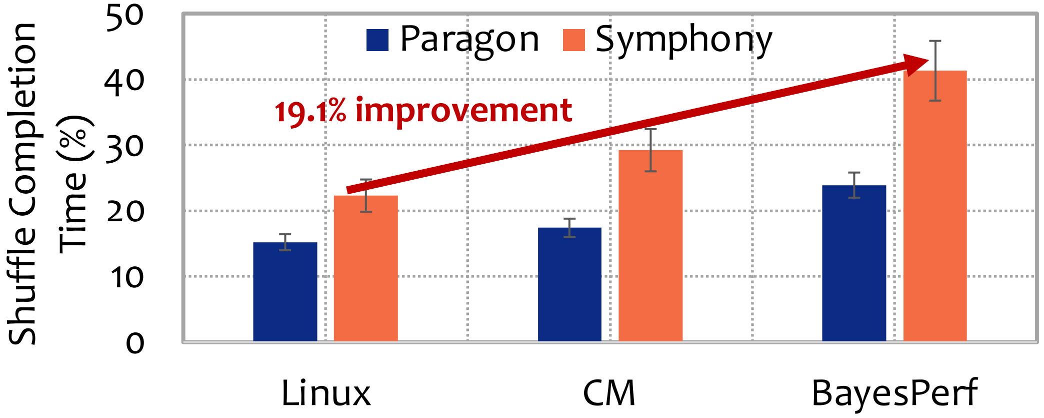
Learned Controller Case Study: Scheduling PCIe Transfers



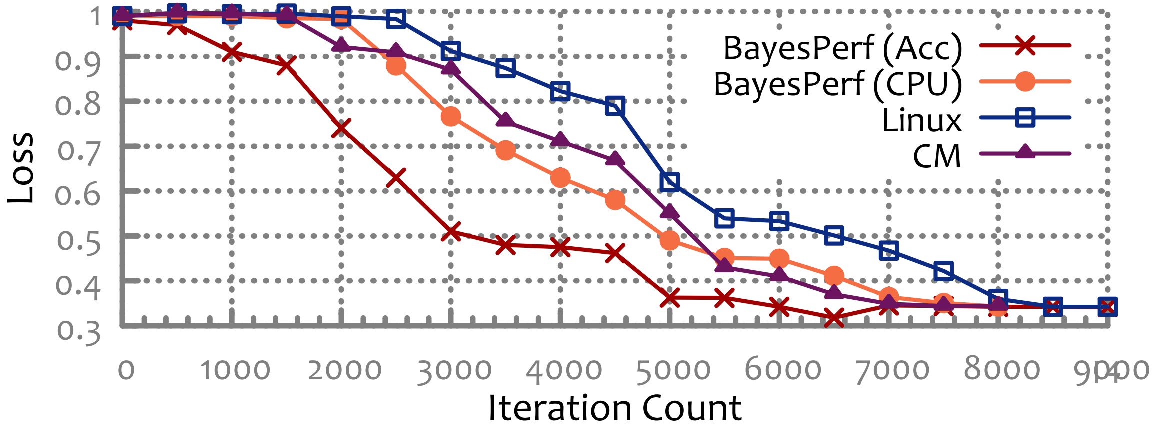


- The Transfer Scheduler is trained using: RL [Symphony - ICML2020], CF [Paragon - ASPLOS 2013]

Upto 19% improvement in overall shuffle completion time



37% reduction in time to convergence for the RL model



Conclusion

BayesPerf: A system for **real-time** **quantification** and **minimization** of HPC measurement errors

Can reduce errors by **as much as 8x** with **<2% latency overhead**

- Net effect of BayesPerf
  - Increases the number of HPC registers
  - Decreases the sampling frequency

Conclusion

BayesPerf: A system for **real-time** **quantification** and **minimization** of HPC measurement errors

Can reduce errors by **as much as 8x** with **<2% latency overhead**

- Net effect of BayesPerf
  - Increases the number of HPC registers
  - Decreases the sampling frequency
- BayesPerf will benefit the portability/scalability of ML for systems
  - More measurements at less error $\Longrightarrow$ More controllers deployed
  - Composability with other ML models

Conclusion

BayesPerf: A system for **real-time** **quantification** and **minimization** of HPC measurement errors

Can reduce errors by **as much as 8x** with **<2% latency overhead**

- Net effect of BayesPerf
  - Increases the number of HPC registers
  - Decreases the sampling frequency
- BayesPerf will benefit the portability/scalability of ML for systems
  - More measurements at less error $\Longrightarrow$ More controllers deployed
  - Composability with other ML models
- We think this idea can be used quantifying and correcting errors other ML applications