Architecting for Big Data Analytics: Think Dubai rather than Venice

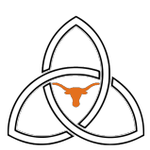
Vijay Janapa Reddi

Visiting Research Scientist @ Google
Associate Professor @ The University of Texas at Austin

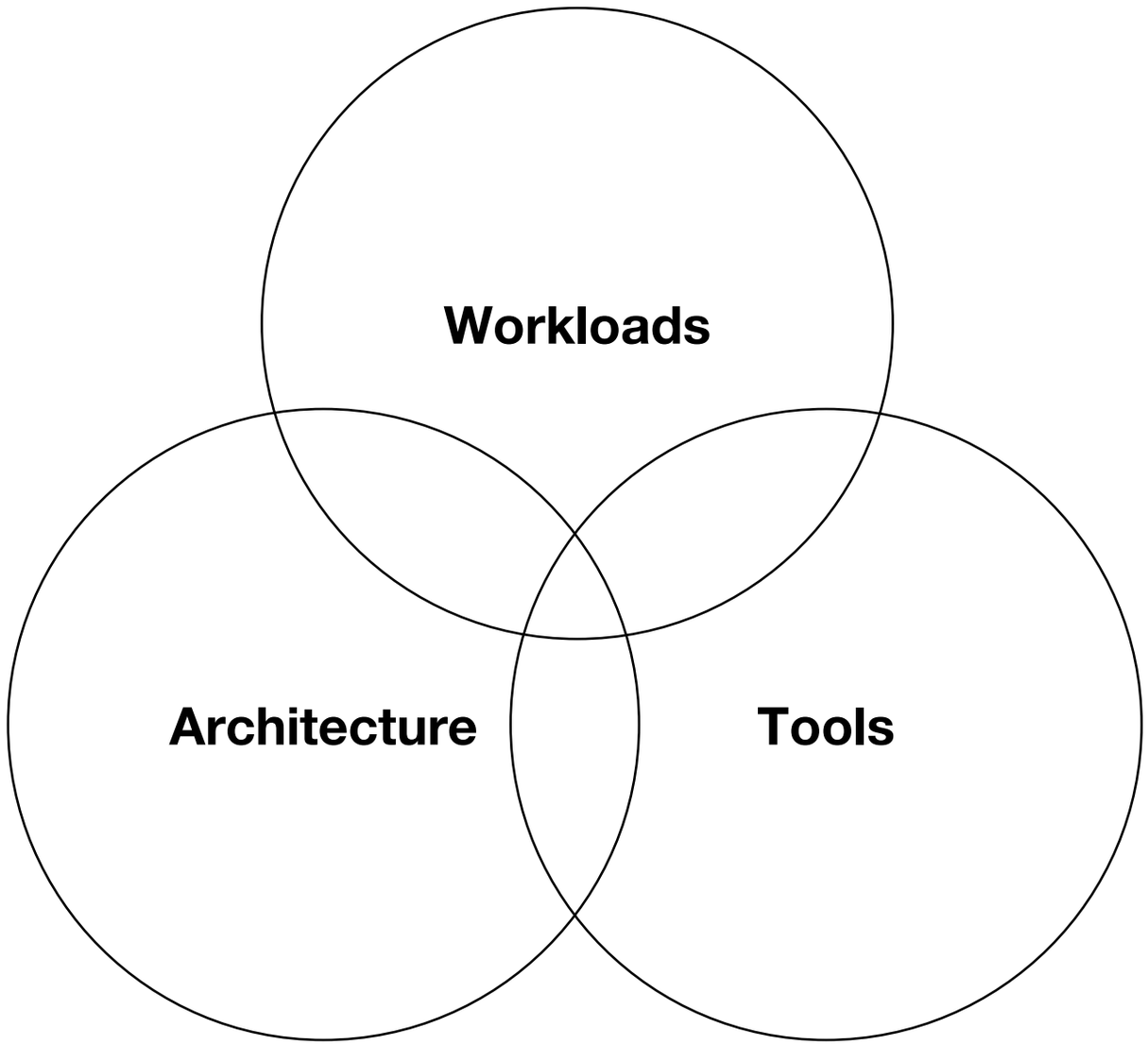
Workshop on BigData Benchmarks, Performance, Optimization and Emerging Hardware
March 24, 2018

TRINITY

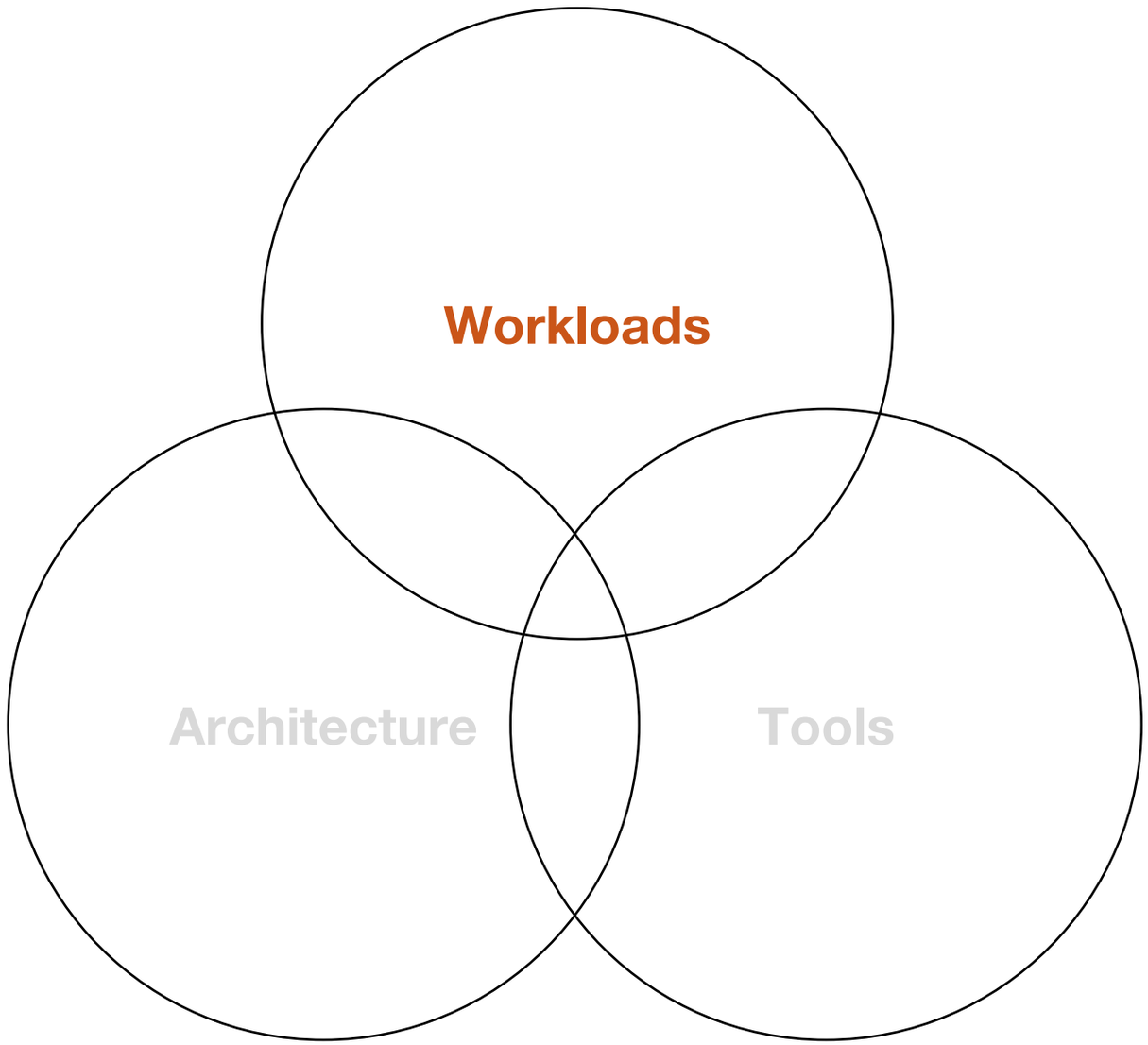
This workshop, the ninth its series, focuses on **architecture and system support for big data systems**, aiming at bringing researchers and practitioners from data management, architecture, and systems research communities together to discuss the research issues at the intersection of these areas.



Focus of My Talk



Focus of My Talk



What is Big Data Analytics?

Big data analytics is the application of **varied techniques** to very **large and diverse data sets** in order to uncover **hidden patterns** and produce **meaningful insights**.

Big data analytics deals with data sets **too large**, problems **too complex**, and patterns **too subtle** to be handled by conventional relational databases.

Big Data Benchmarks

There are ample big data-related benchmarks.

CloudSuite

What is it and why should computer architects care?

AMP Lab Benchmarks

LinkBench

YCSB

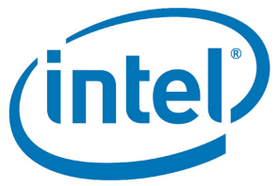
DCBench

HiBench

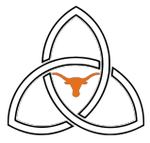
TPCx-BB BigBench

TPCx-BB Origin and Features

Proposed at SIGMOD 2013, BigBench was developed with input from many industry partners.



BigBench was standardized as TPCx-BB in 2014.



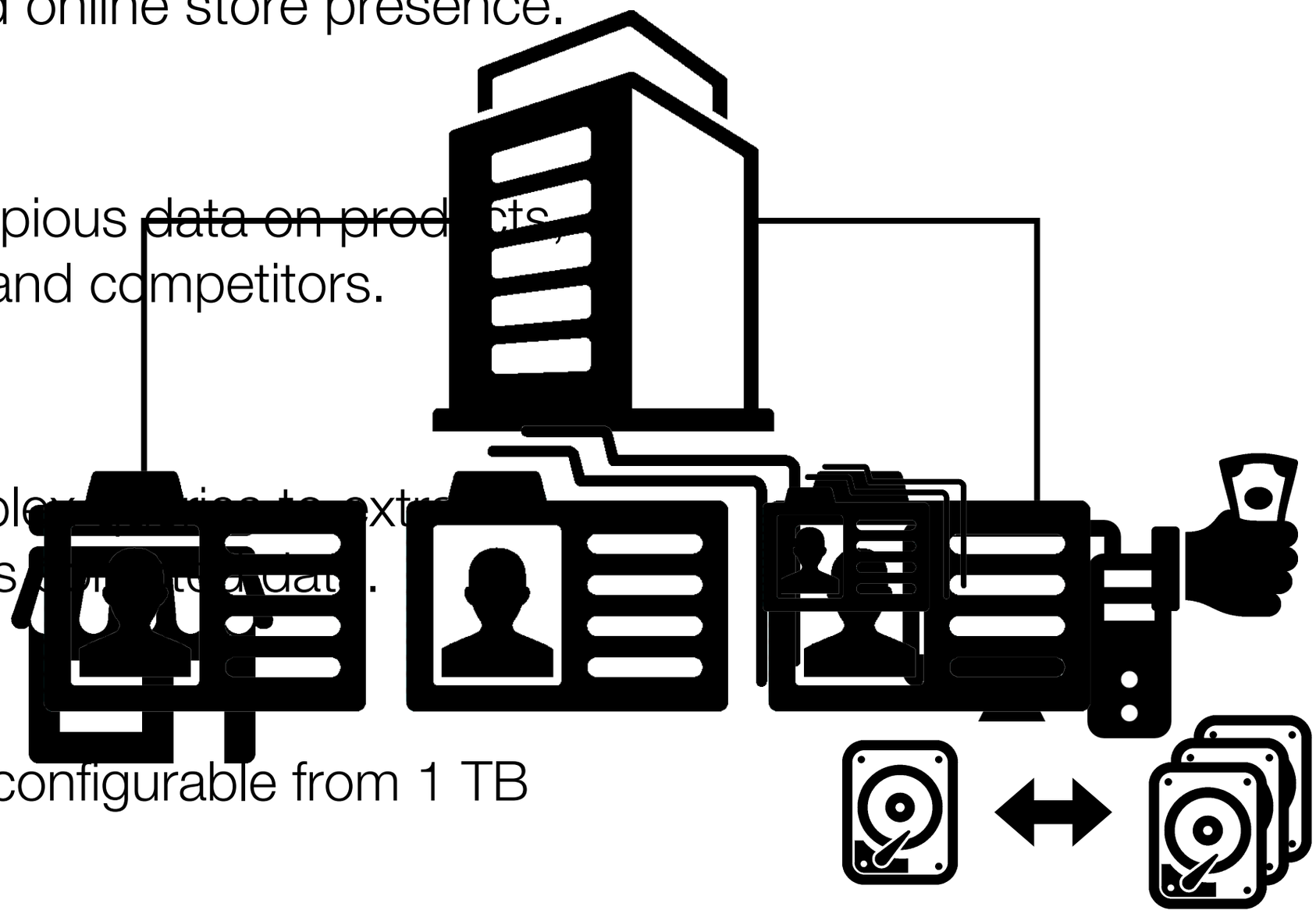
TPCx-BB (BigBench) is Uniquely Realistic

It simulates a modern retailer with a physical and online store presence.

It gathers copious data on products, customers, and competitors.

It uses complex queries to extract value from its semi-structured data.

Data size is configurable from 1 TB to 1 PB.



BigBench Queries

30 queries operate on collected data to extract useful information.

Q01: Find the top 100 products frequently purchased together.

Q12: Find customers who viewed certain categories online then made an in-store purchase in the same category.

Q27: Extract competitor product names from online reviews.

Coverage

BigBench data includes
structured (e.g. customer demographics) data,
semi-structured (e.g. web site click streams) data, and
un-structured (e.g. online product reviews) data.

BigBench operates on the data using
MapReduce,
machine learning,
user-defined functions,
query language operations,
and **natural language processing**.

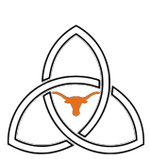
Most queries take multiple steps, and many cover multiple data types and operation types.

Why Should We Study One More Benchmark?

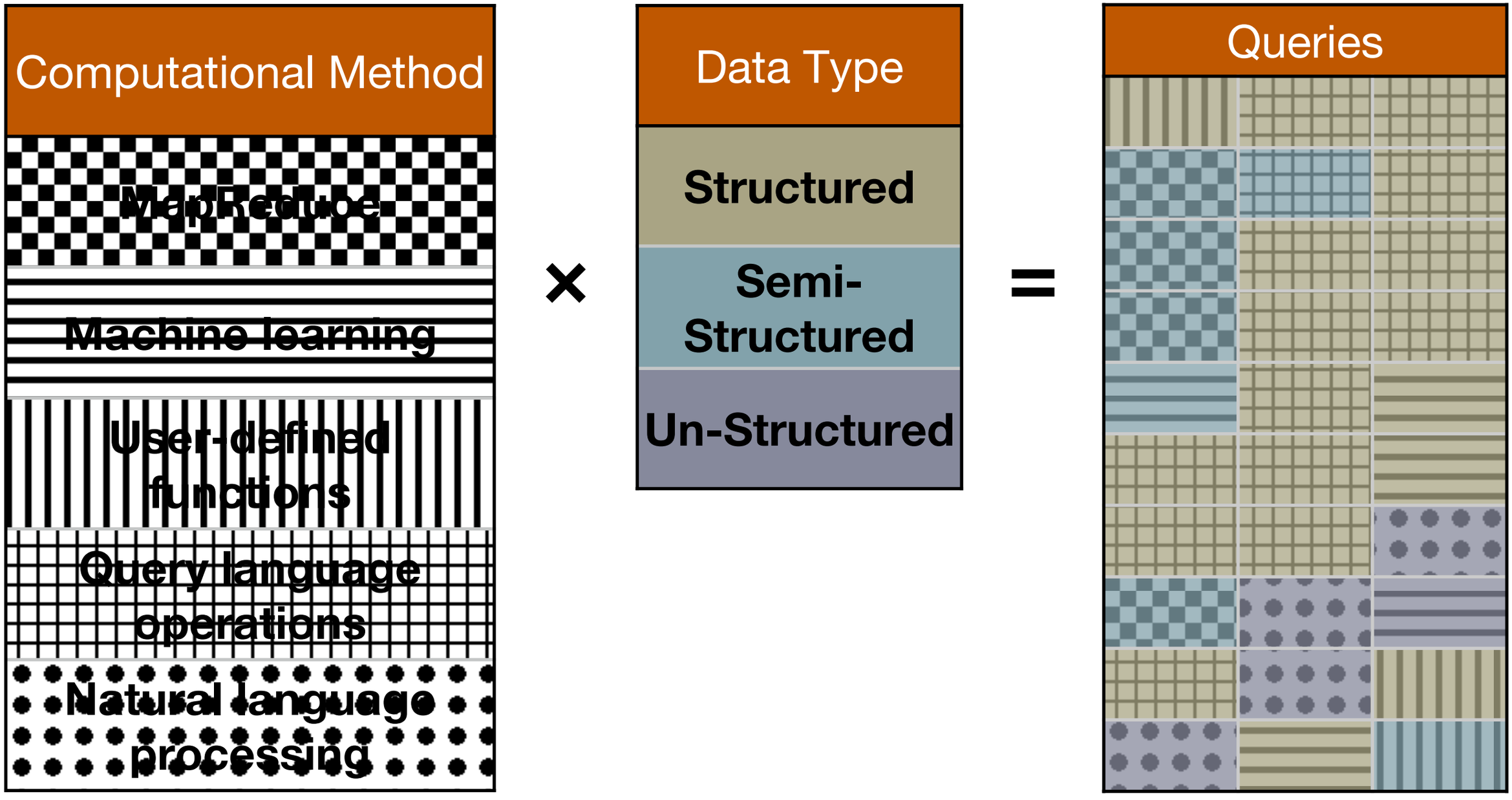
It is essential to **understand, emulate, and study industry perspectives** in order to produce believable and relevant insights.

There is a **lot of heterogeneity in the applications**, something that has been oversimplified in past benchmarking efforts.

Thread-limited execution is still pervasive even in scale-out big data analytics and demands better scale-up performance.

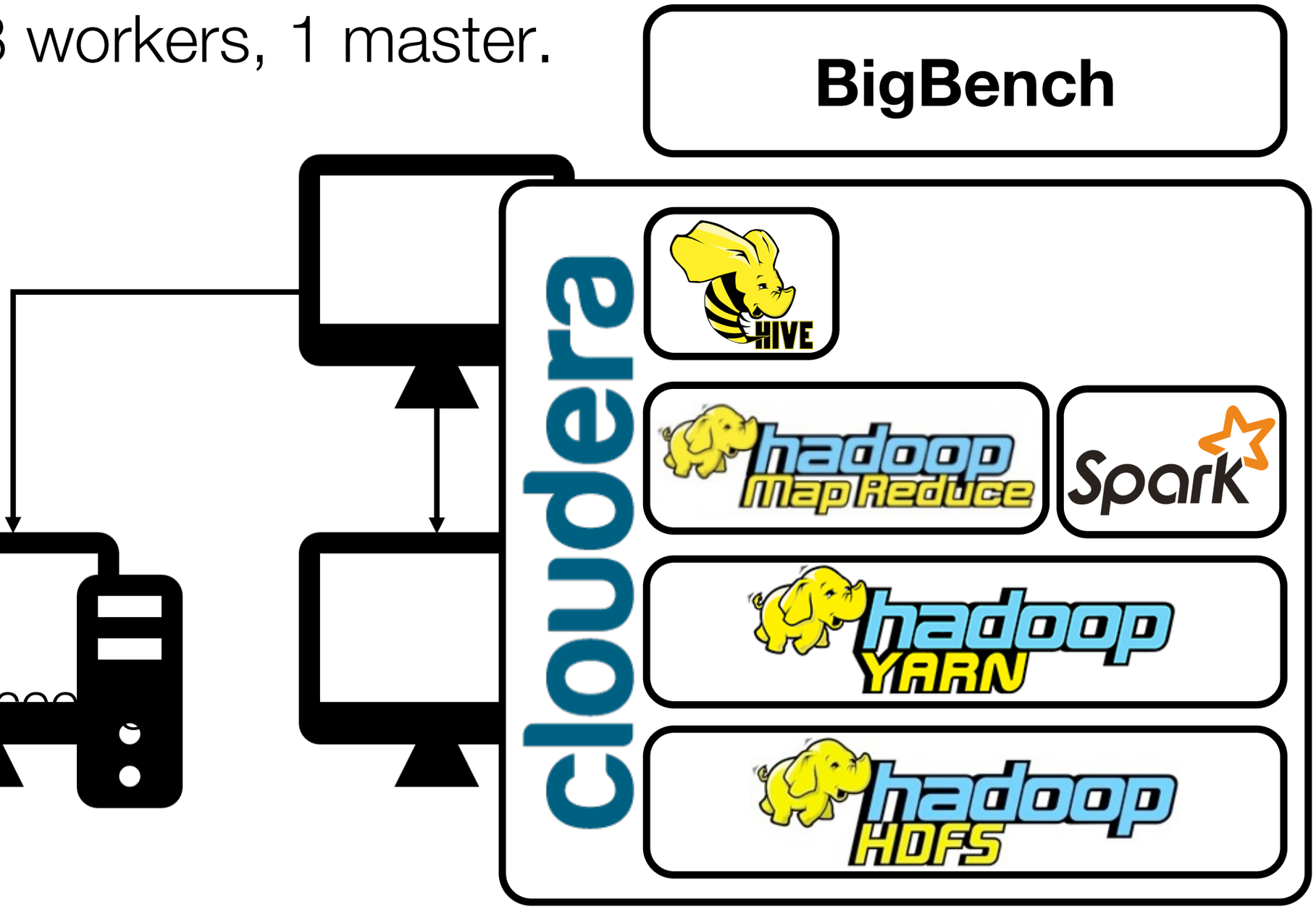


Big Data Analytics is a Rich and Varied Field



Experimental Setup

3+1 setup: 3 workers, 1 master.



Worker nodes:

Intel Xeon E5-2660

384 GB RAM

SSD Storage

10 Gbps Ethernet

BigBench Stages

Data Generation
Creates the data used by the benchmark.

Load Test
Extracts/transforms/loads the data into a usable form.

Power Test
Runs each query sequentially to minimize latency.

Throughput Test
Runs multiple queries at once to maximize throughput.

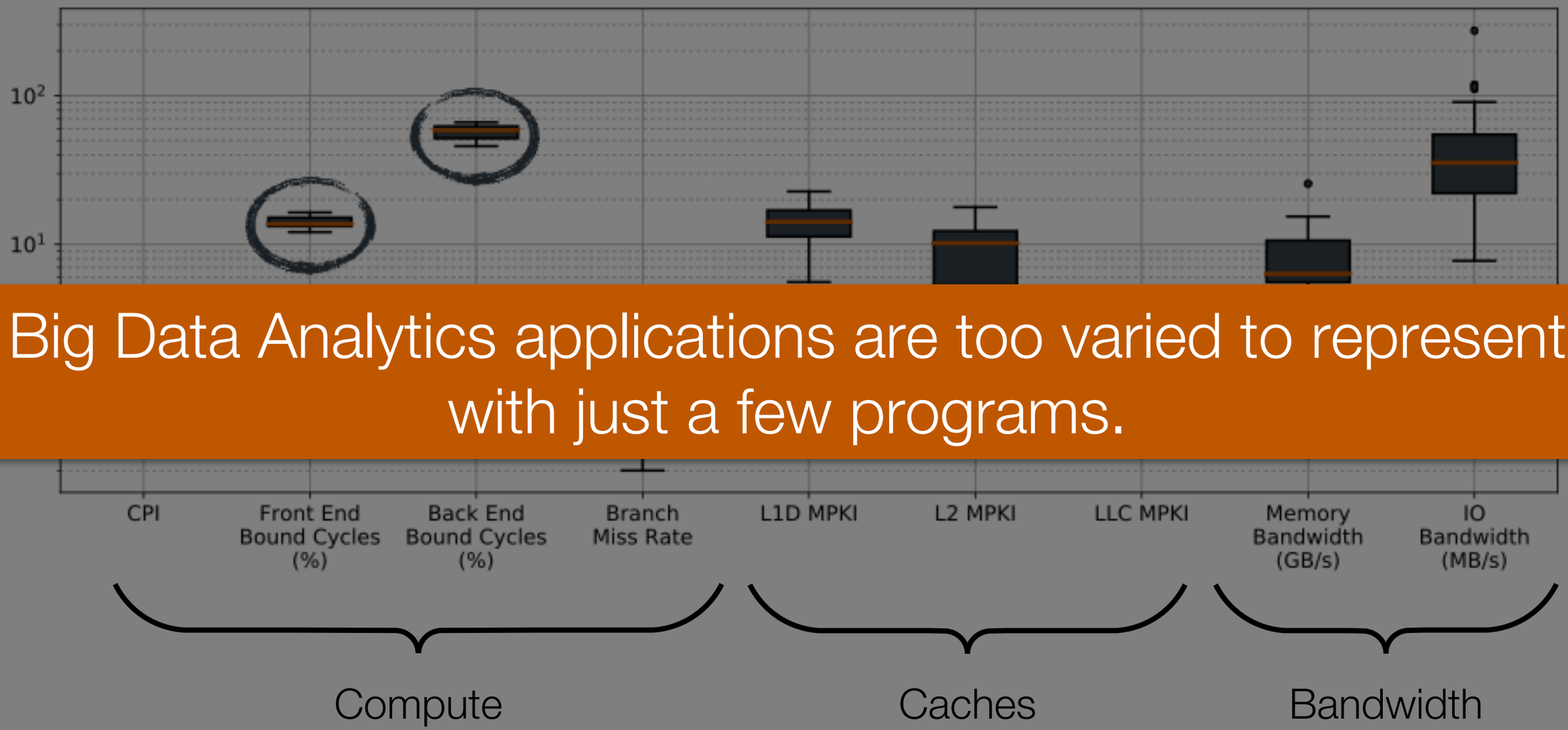
Score
Runs

ad

le

sequenti

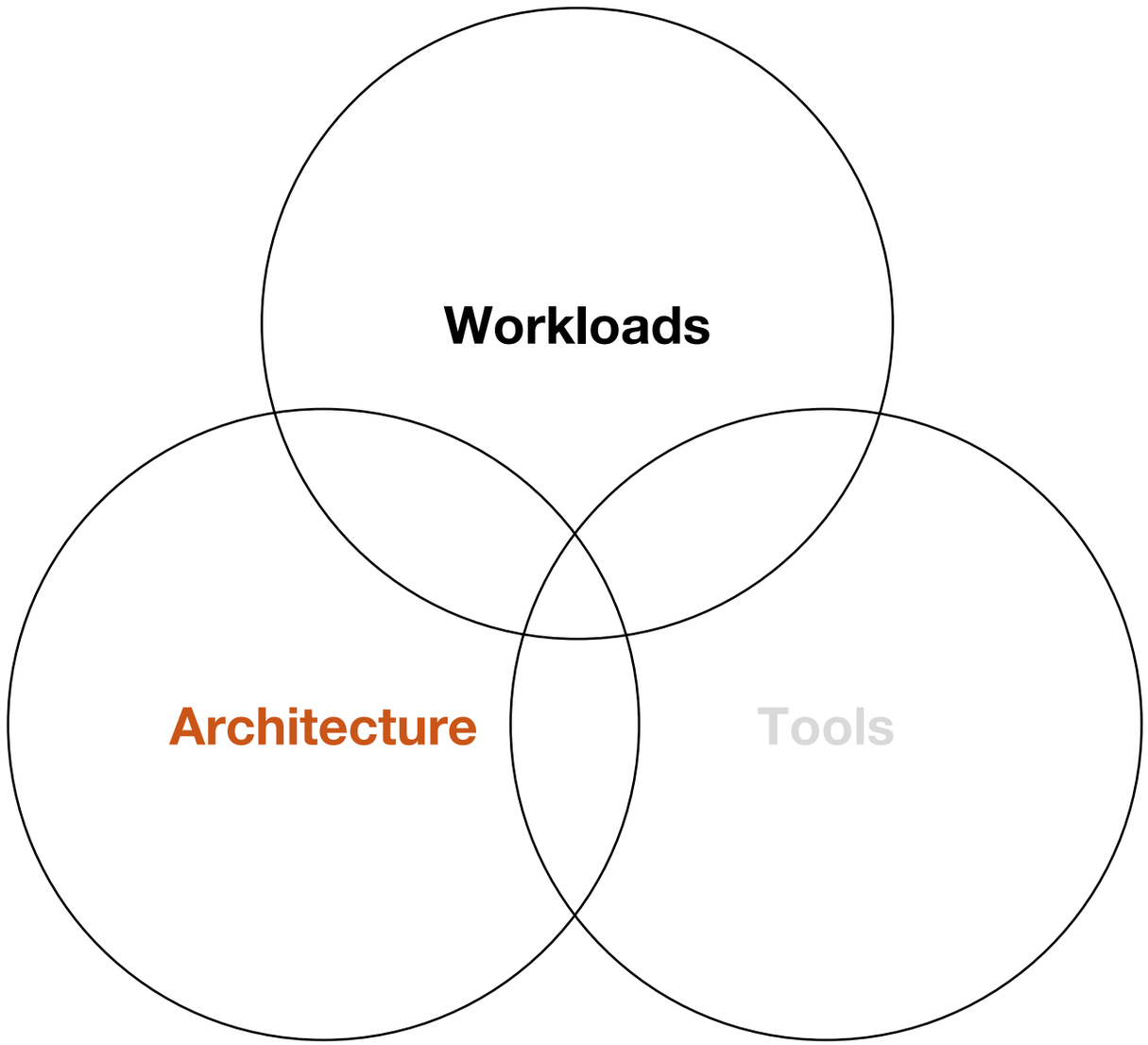
Characterization:
A Caution Against Oversimplification



Big Data Analytics applications are too varied to represent with just a few programs.

Only the front and back end-bound cycles are stable across queries.

Focus of My Talk



With the Promises of what Big Data Can Do, We Forget About What it Cannot Do

---

“ Hadoop makes reliable, scalable, distributed computing possible

We see these claims about Hadoop scaling to thousands of machines and we have a tendency to think we don’t need to worry about scale-up.

“ Apache Hadoop is a highly scalable storage platform designed to process very large data sets across hundreds to thousands of

”

In reality, scale-up performance is just as important in big data analytics as it has ever been.

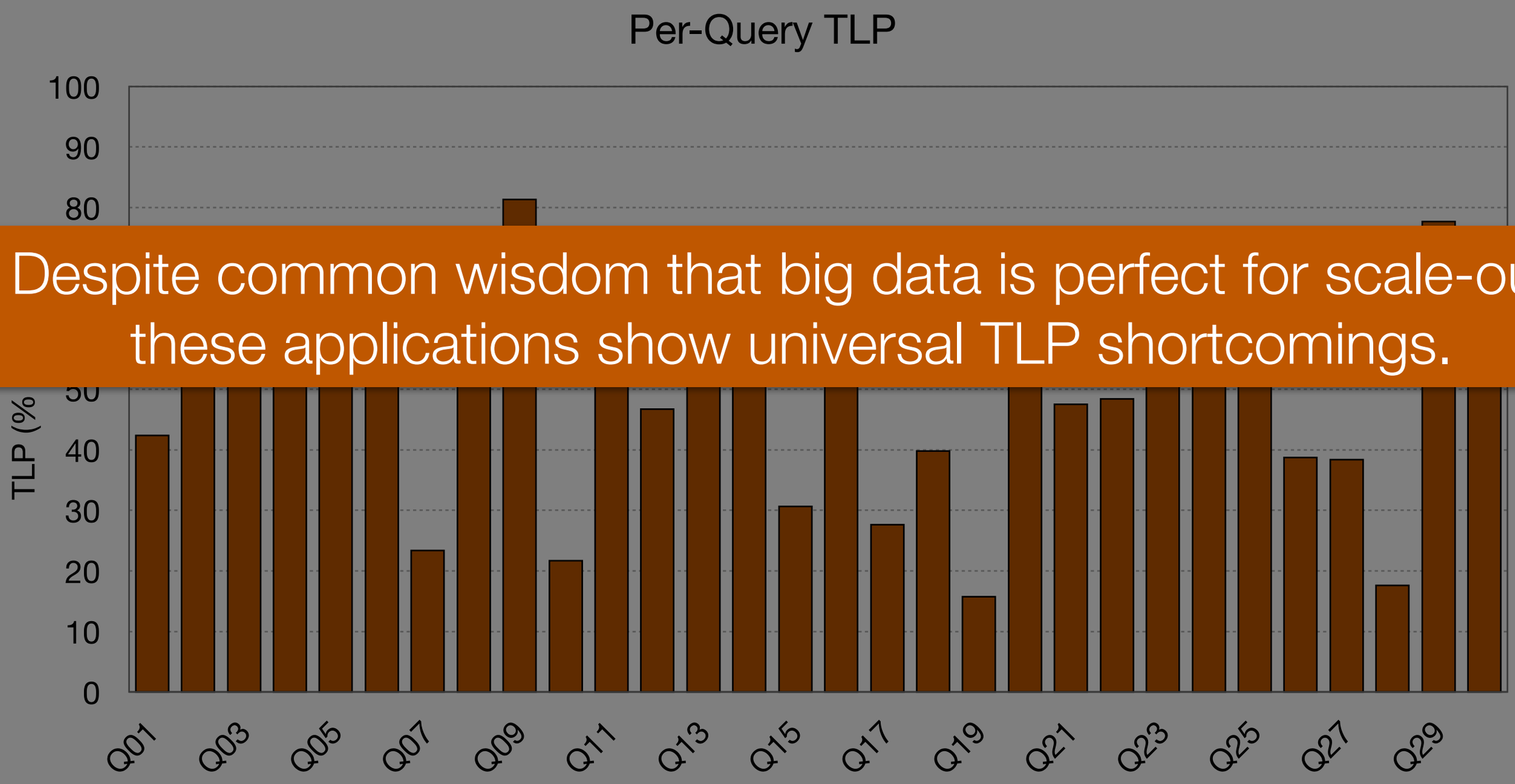
”

because Hadoop spreads it out…You’ve got all of these processors, working in parallel, harnessed together.

”

Cloudera Co-Founder Mike Olson

Amdahl's Law is Alive and Kicking



Despite common wisdom that big data is perfect for scale-out, these applications show universal TLP shortcomings.

How Do We Design For This?

Big data analytics already scales out. How can we get the performance to scale up?

Contrasting Scale-Out and Scale-Up

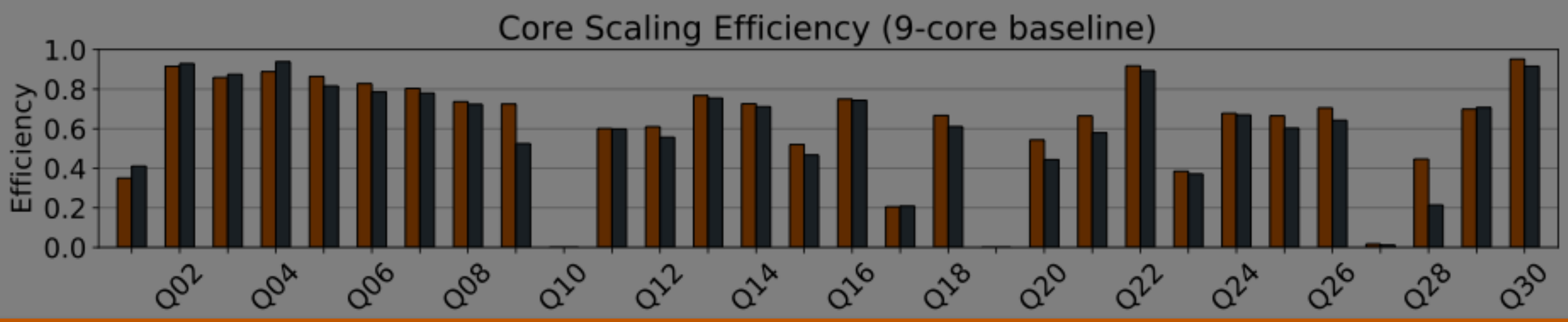
We measure the change in runtime as we scale resources:

- Number of cores
- Operating frequency

We report the ***efficiency*** of resource scaling.

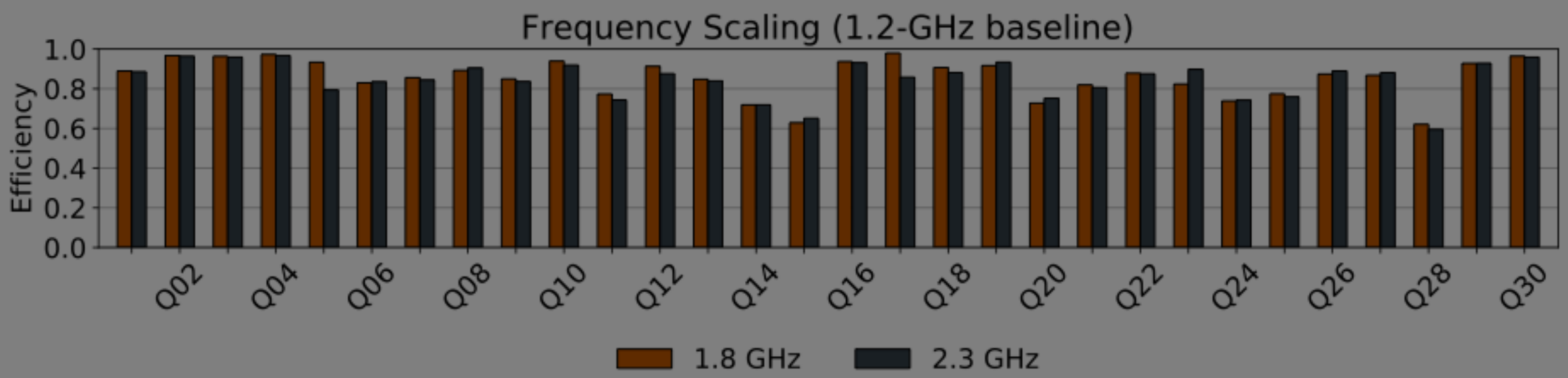
$$Efficiency = \frac{\text{Change in run time}}{\text{Expected change}}$$

Scaling Efficiency



Frequency scaling is more efficient than core scaling.

all.



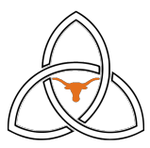
What About Turbo Boost?

Turbo Boost uses slack in thermal and current margins to increase the operating frequency of CPUs.

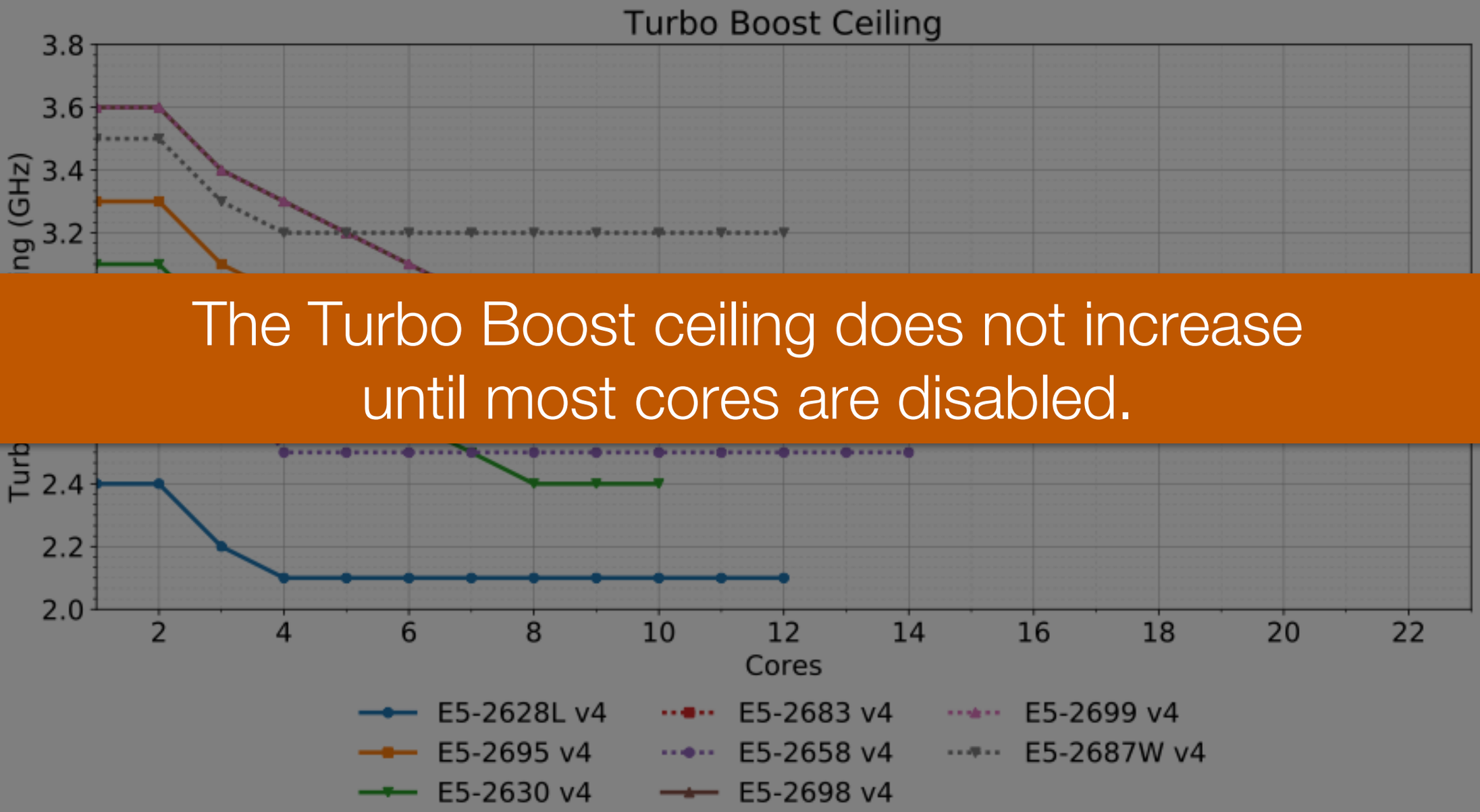
The Turbo Boost ceiling is a function of the number of active cores. The fewer the cores, the higher the ceiling.

With so many halted cycles, Turbo Boost should be perfect for BigBench, right?

Not quite…

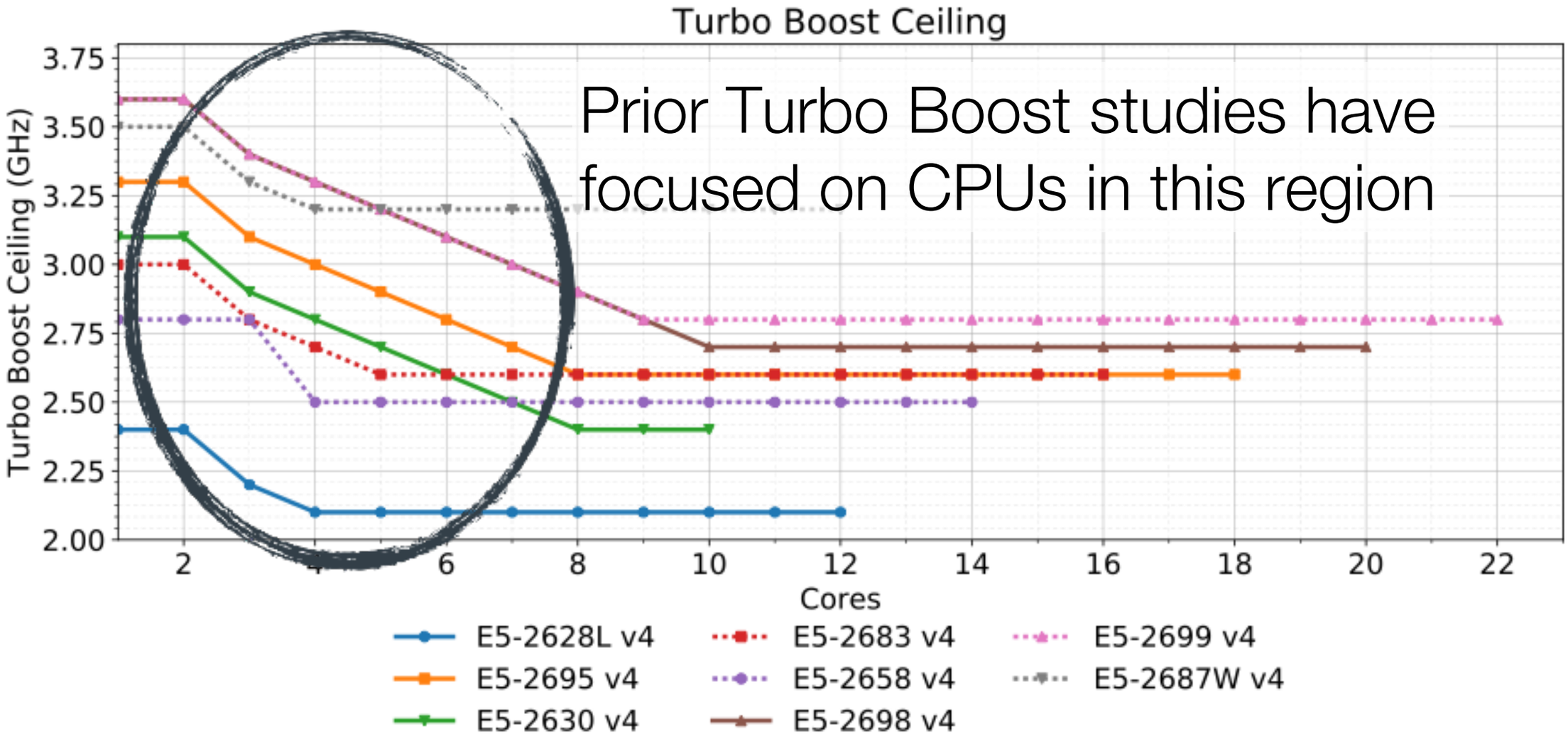


A Deeper Problem



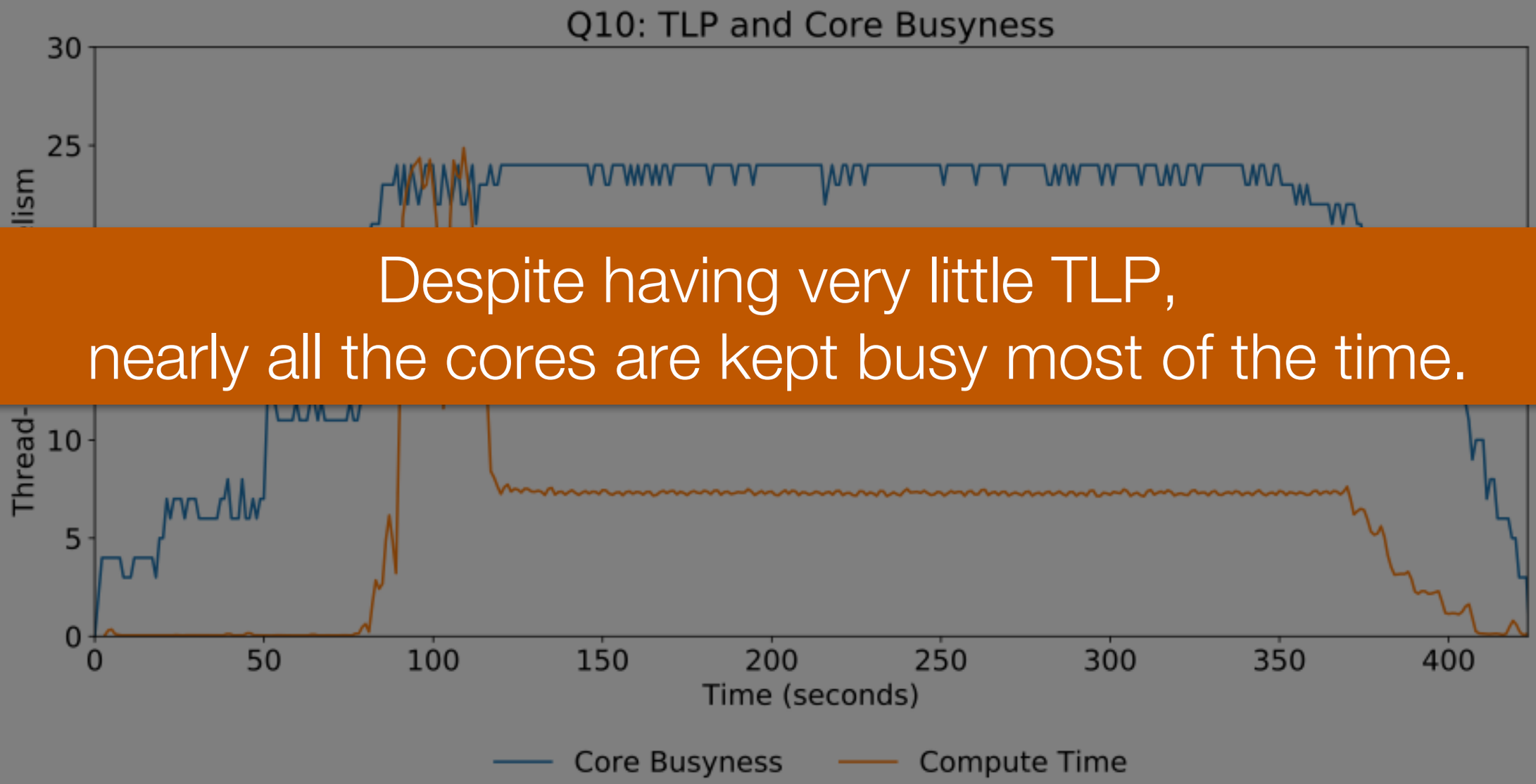
The Turbo Boost ceiling does not increase until most cores are disabled.

A Deeper Problem



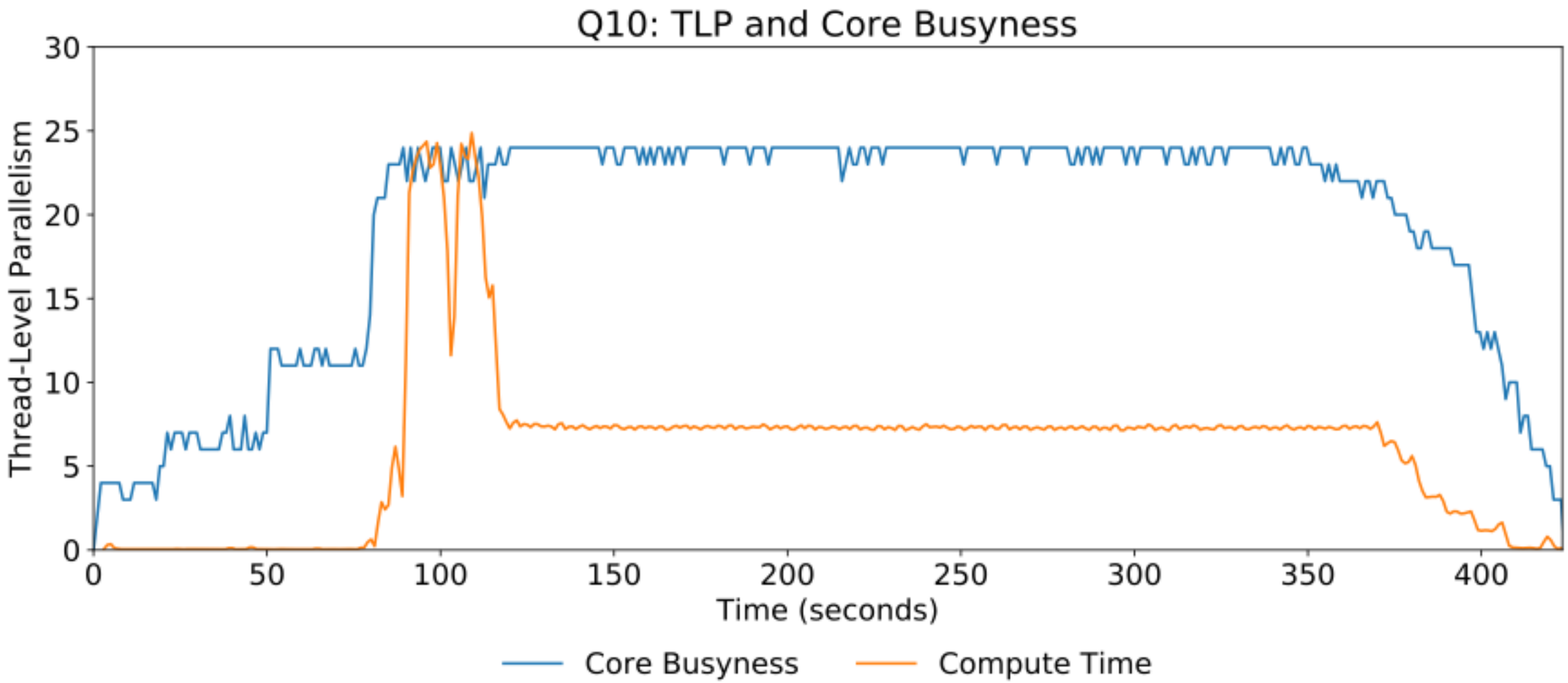
The Turbo Boost ceiling does not increase until most cores have been disabled.

A Deeper Problem…Compounded By Software



Despite having very little TLP, nearly all the cores are kept busy most of the time.

A Deeper Problem…Compounded By Software



Despite having very little TLP, nearly all the cores are kept busy most of the time.

Double Whammy

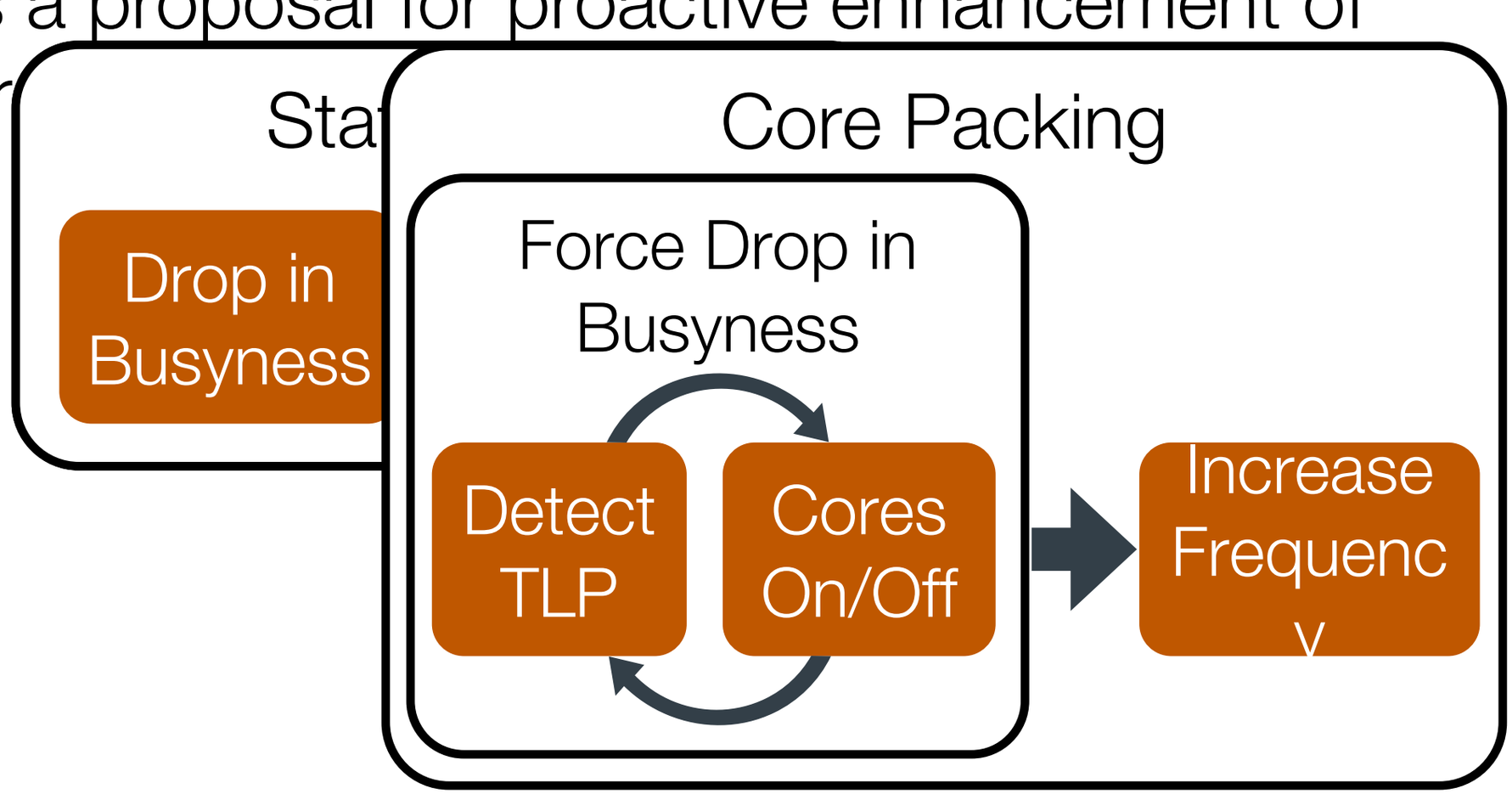
Hardware: Slack in thermal and current margins does not translate to a higher Turbo Boost ceiling until most cores

Hardware and software cooperatively prevent Turbo Boost from ever exceeding its baseline ceiling.

Software: The abundant software threads are being scheduled onto virtually all available cores, whether they are actually needed or not.

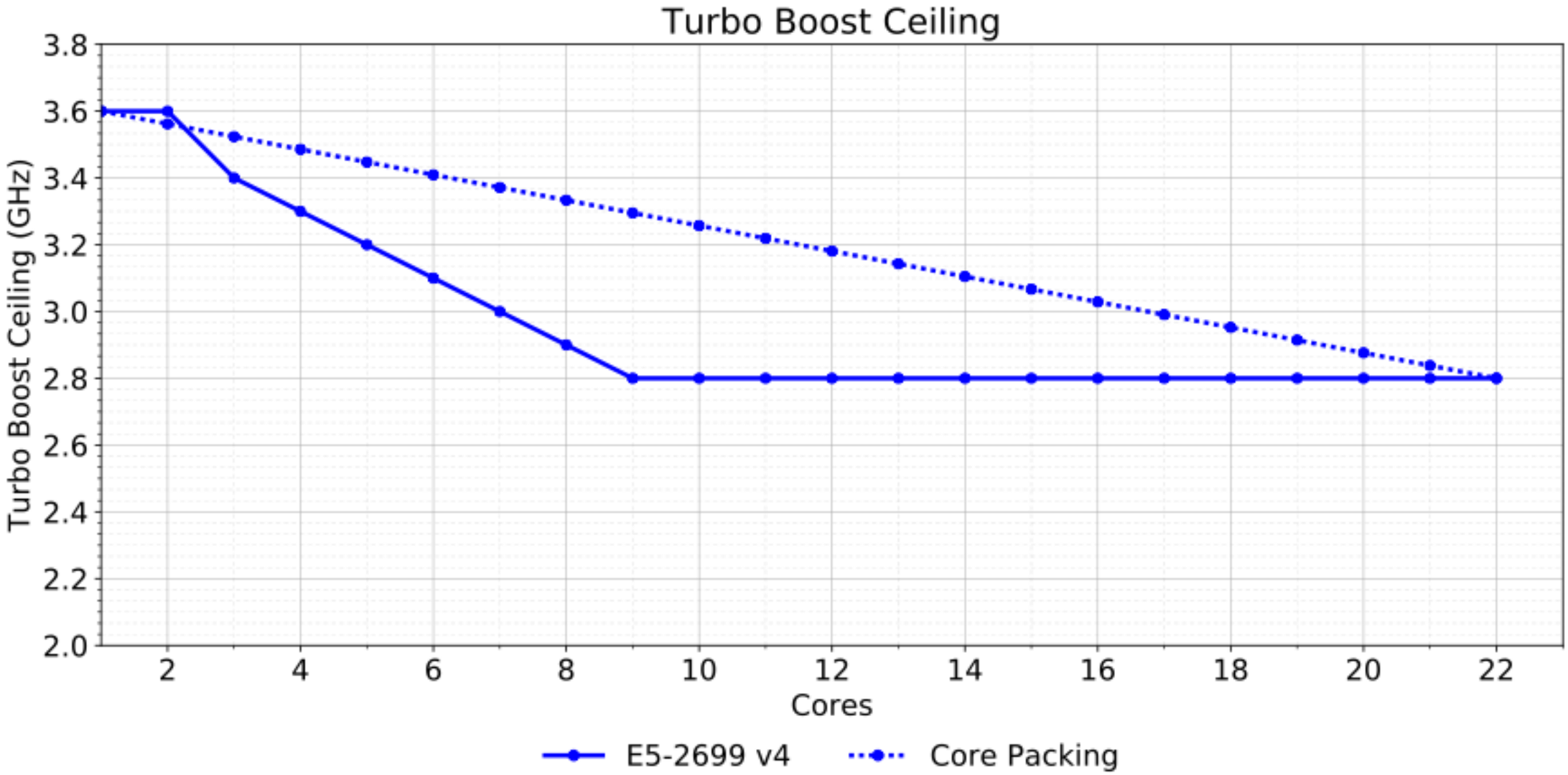
Core Packing

Core packing is a proposal for proactive enhancement of Turbo Boost thr



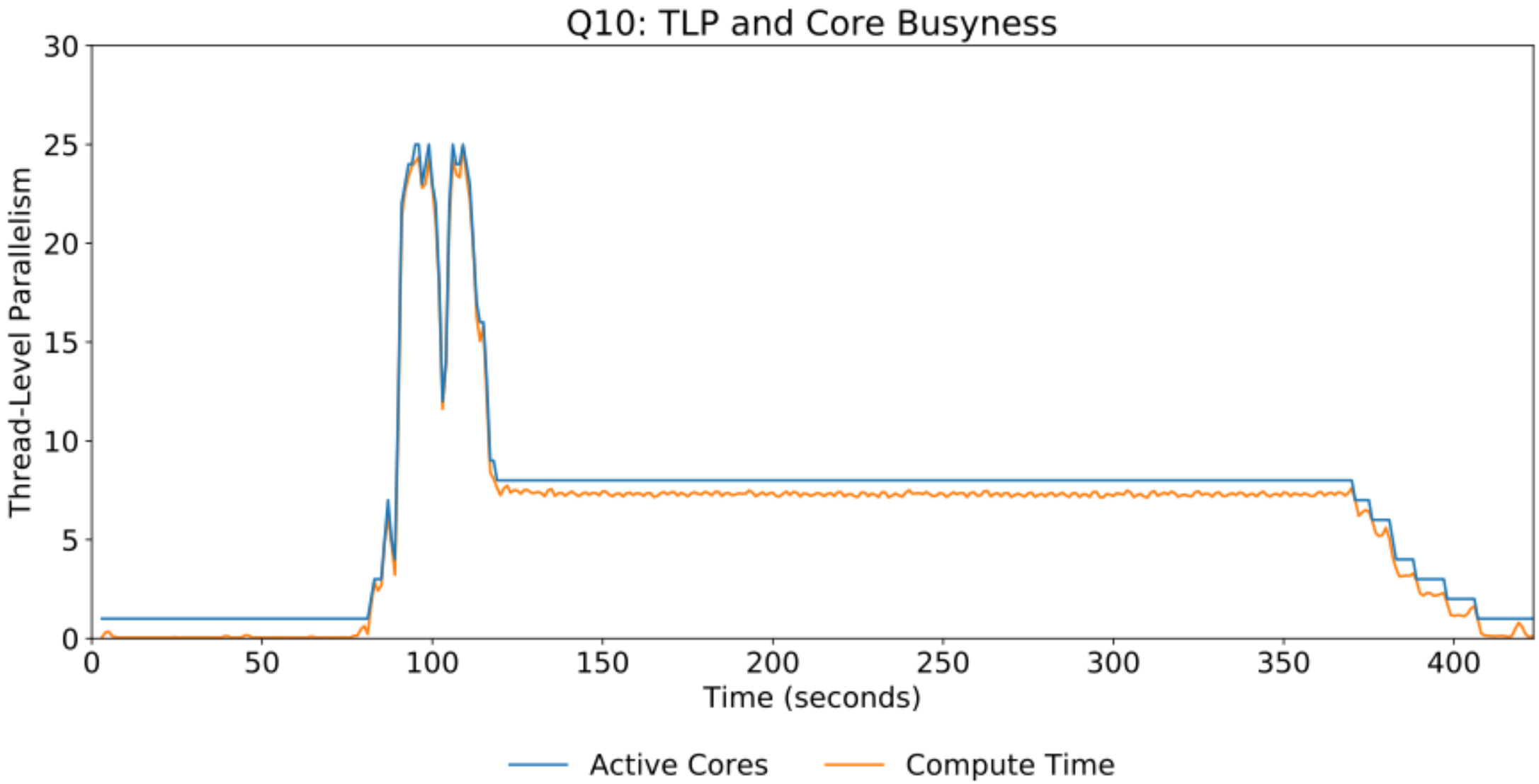
Core Packing—Hardware Proposal

Allow for linear increase in Turbo Boost ceiling with the number of deactivated cores.



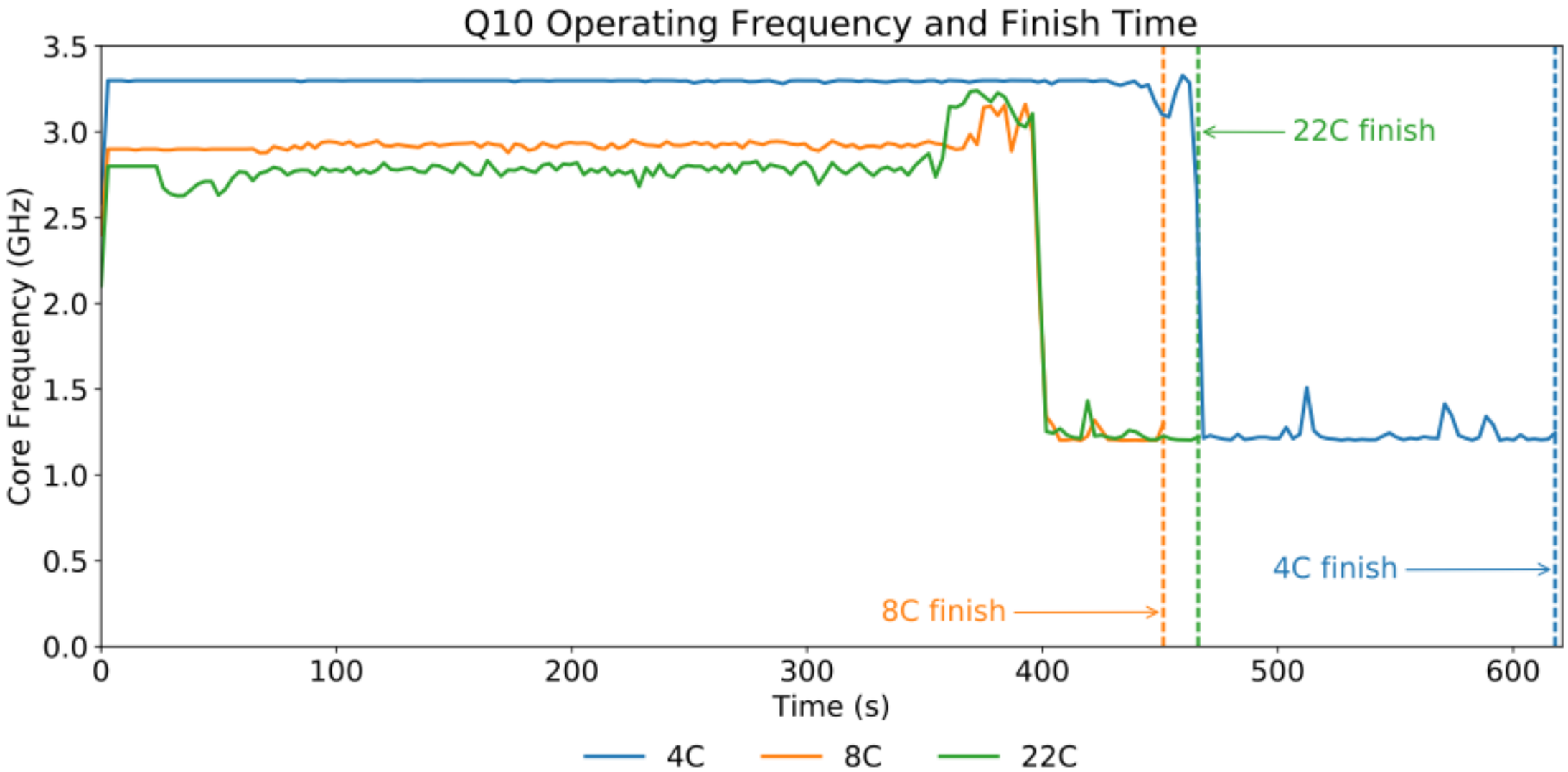
Core Packing

Proactively restrict the number of active cores to just meet the workload.

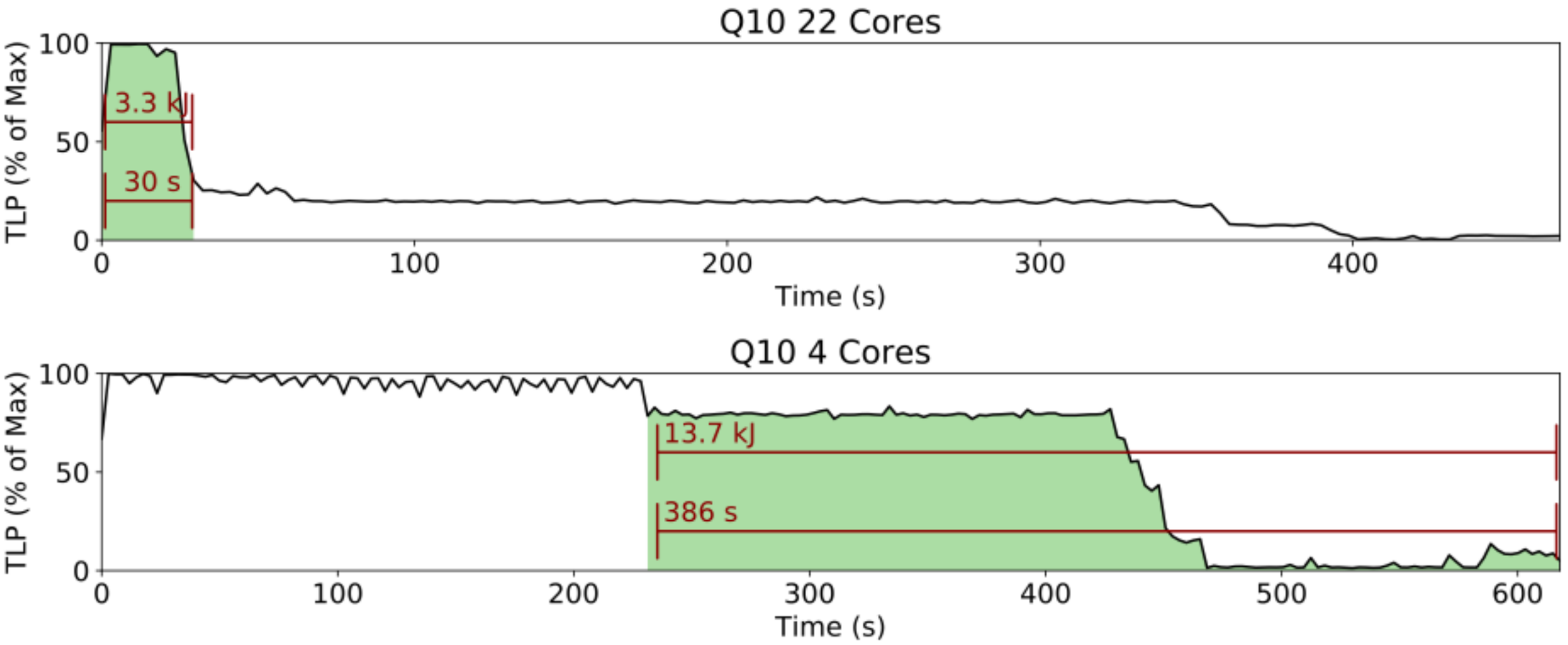


Core Packing — Simple Approximation

Q10 has such consistently low TLP, we can use it to approximate core packing behavior.

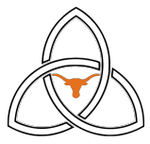


Core Packing — Better Approximation



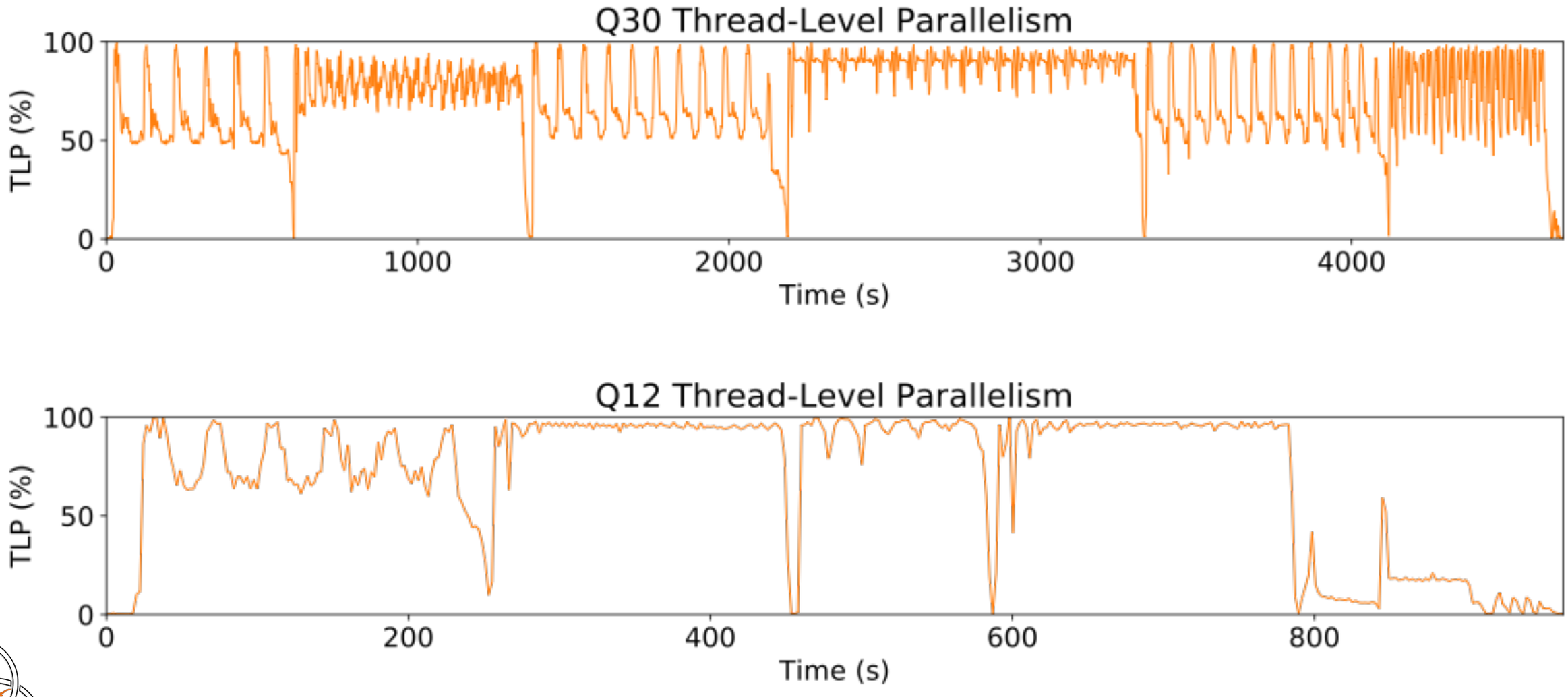
Query 10 could finish in 416 seconds: a 13.5% speedup.

Furthermore, energy consumption decreases by 30%.



Core Packing — Complexity

Most queries are far too complex to enable such simple core packing approximation.



Analytical Model—Turbo Boost Frequency

Assuming a linear increase in Turbo Boost ceiling for each deactivated core, at what frequency can each query run?

$$freq_{TB} = freq_0 + (1 - TLP) \cdot (freq_{Max} - freq_0)$$

$freq_0$ Base (current) frequency

$(1 - TLP)$ Available slack in cores

$(freq_{Max} - freq_0)$ Turbo Boost ceiling room

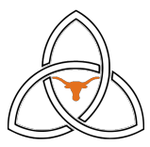
Analytical Model—Packed Cores

Given a potentially higher operating frequency, how much speedup can we expect?

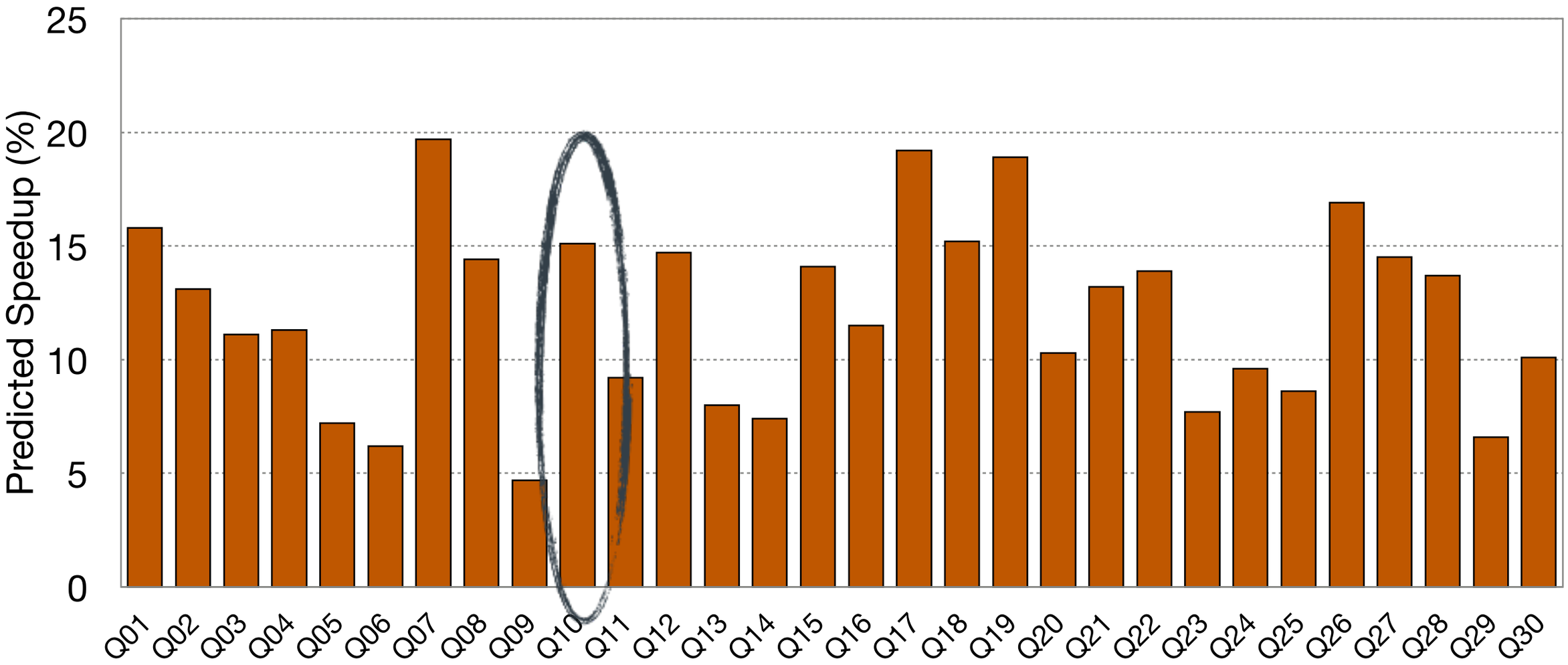
$$speedup = \frac{freq_{TB} - freq_0}{freq_0} \cdot efficiency_{freq}$$

$\frac{freq_{TB} - freq_0}{freq_0}$ Fractional increase in frequency

$efficiency_{freq}$ Efficiency of frequency scaling



Analytical Model—Evaluation



Notably, Q10's modeled speedup of 15.1% is remarkably close to our predicted speedup of 13.5%.

Takeaways

TPCx-BB (**BigBench**) is a new benchmark that strives to capture the realism and diversity of industry workloads.

Thread-level parallelism is not abundant in big data workloads as common wisdom would have us believe.

Need to **explore new, more proactive solutions** like Core Packing by digging deep into the individual queries.

Scale-out

Scale-up

Amdahl's Law in Big Data Analytics: Alive and Kicking in TPCx-BB (BigBench)

Daniel Richins[†] Tahrina Ahmed[*] Russell Clapp[*] Vijay Janapa Reddi[†‡]

[†]*The University of Texas at Austin* [*]*Stanford University* [*]*Intel* [‡]*Google*

ABSTRACT

Big data, specifically data analytics, is responsible for driving many of consumers' most common online activities, including shopping, web searches, and interactions on social media. In this paper, we present the first (micro)architectural investigation of a new industry-standard, open source benchmark suite directed at big data analytics applications—TPCx-BB (BigBench). Where previous work has usually studied benchmarks which oversimplify big data analytics, our study of BigBench reveals that there is immense diversity among applications, owing to their varied data types, computational paradigms, and analyses. In our analysis, we also make an important discovery generally restricting processor performance in big data. Contrary to conventional wisdom that big data applications lend themselves naturally to parallelism, we discover that they lack sufficient thread-level parallelism (TLP) to fully utilize all cores. In other words, they are constrained by Amdahl's law. While TLP may be limited by various factors, ultimately we find that single-thread performance is as relevant in scale-out workloads as it is in more classical applications. To this end we present core packing: a software and hardware solution that could provide as much as 20% execution speedup for some big data analytics applications.

1. INTRODUCTION

Big data analytics is the application of advanced analytic techniques to large, diverse structured and un-structured data. It empowers users with a granular perspective of complex business operations and customer habits that rarely find their way into traditional data warehouses or standardized reports. Using techniques such as predictive analytics, data mining, statistics, machine learning, and natural language processing, big data analytics enables its users to understand the current state of the business and track complex and continuously evolving behavior such as end-user customer traits.

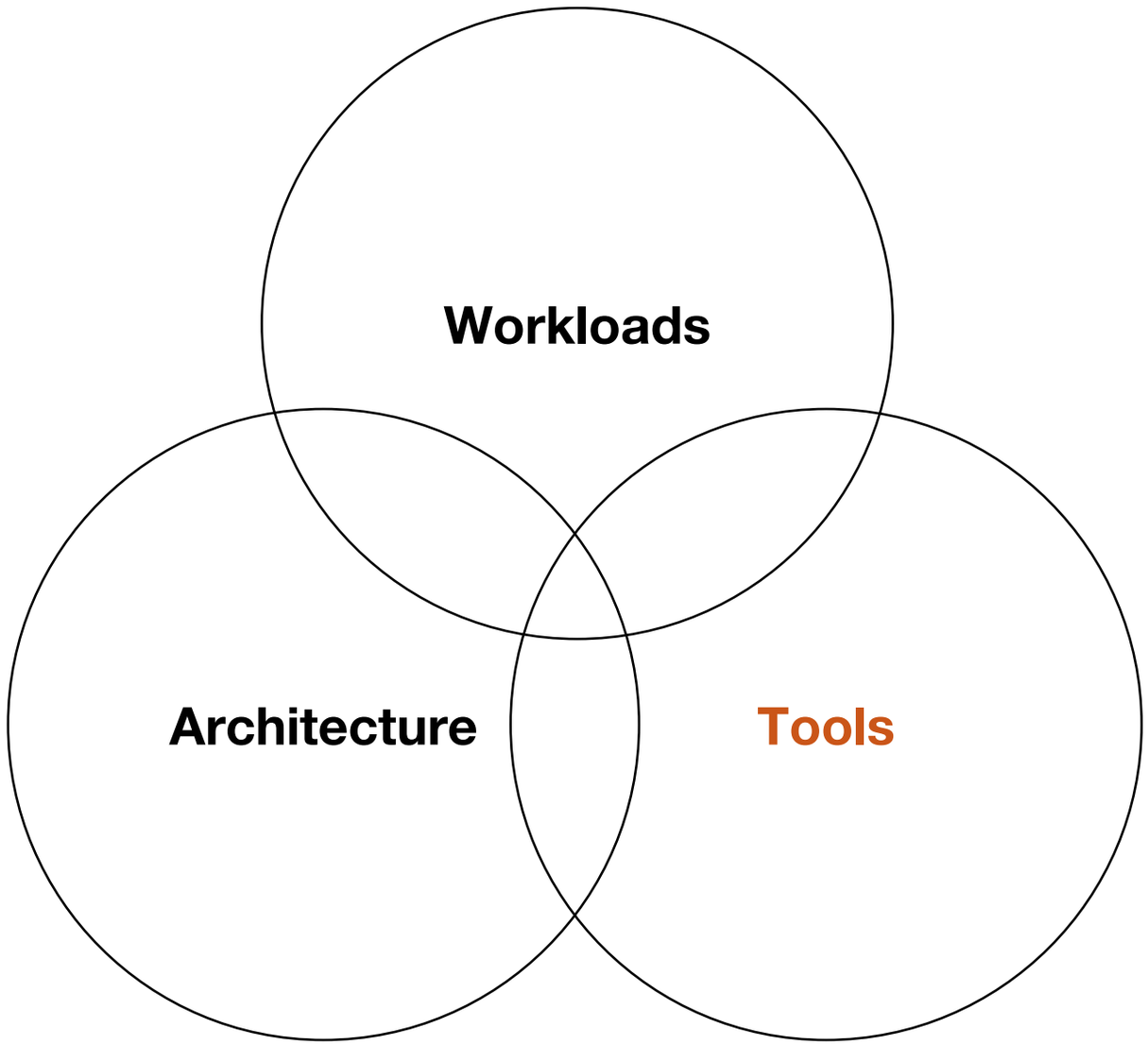
From an industry perspective, *big data analytics has been oversimplified.* Much previous research has been conducted into big data [1,2,3,4,5,6,7], but this has often taken a broad approach, covering not only data analytics but also media streaming, social networking, real-time services, etc. Often, data analytics is reduced to simple, sample applications intended as demonstrations rather than benchmarks. While not wholly without value, these micro-benchmarks are ultimately not representative of industry. To merit serious study, an analytics benchmark should be characterized by (1) **realism**, the use of applications that are representative of real-world applications, including complexity and size; (2) **comprehensiveness**, or a thorough exercise of functionalities; and (3) **usability**, which ensures reproducibility of studies.

In this work, we study TPCx-BB, which distinguishes itself as a *comprehensive* data analytics benchmark, *representative* of industry workloads, designed to be eminently *usable*. TPCx-BB (hereafter "BigBench") is a recent industry-standard workload developed through collaboration from multiple industry partners. BigBench is especially attractive because it simulates a real-world scenario: a modern retailer with both an online and physical store presence which collects a wealth of data about its customers, competitors, stores, and online reputation. It seeks, through 30 data analytics "queries," to use this data for economic gain. Each query operates on a subset of the data in a unique way: `Q01`, for example, identifies the top products sold together in given stores (structured data), while `Q28` classifies product review sentiment (positive or negative) based on the textual customer reviews (un-structured data) that are logged into the database.

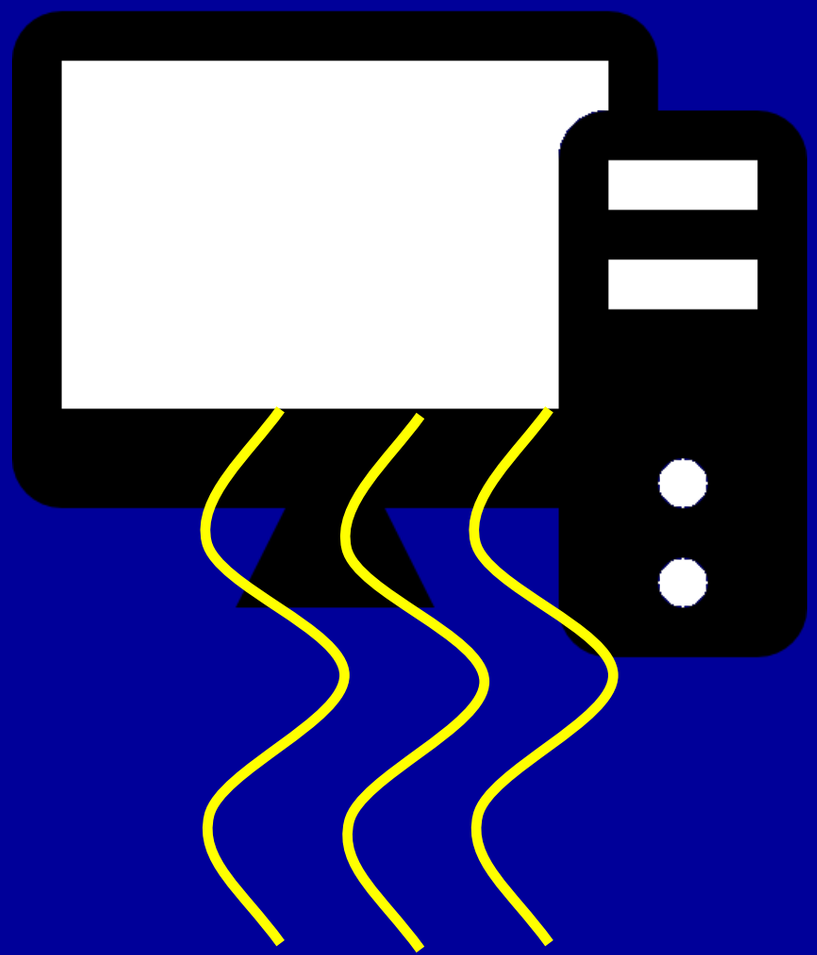
We perform the first comprehensive characterization of BigBench. We analyze BigBench from two key perspectives. First, we undertake a (micro)architectural study of its execution on an enterprise-level cluster. Compared to prior work, which often tend to use only a handful of (micro) benchmarks, the 30 component queries show *diverse behavior* that is masked when they are considered in aggregate. This diversity arises from exercising different Hadoop and Spark capabilities such as MapReduce, machine learning, natural language processing, pure query language queries, and others in various combinations on structured, semi-structured, and un-structured data. Each BigBench query is a complete application, executing multiple operations on various sources of data to produce unique and insightful takeaways. As a result, we see that some queries show great computational diversity (`Q02`, `Q09`, and `Q28`), while others show great memory (`Q06` and `Q19`) or I/O diversity (`Q05` and `Q16`). It is impossible to capture these extremes with only a handful of benchmarks.

Second, we analyze the parallelism characteristics of BigBench to reveal a startling dearth of thread-level parallelism (TLP), in stark contrast to widely held assumptions regarding big data's scale-out potential. The lack of TLP arises from various sources, but its effect is always the same: cores are being left unused by big data. This suggests that relying solely on scale-out resources is insufficient; systems must also be designed to actively monitor TLP and take proactive measures to boost single-thread performance as necessary.

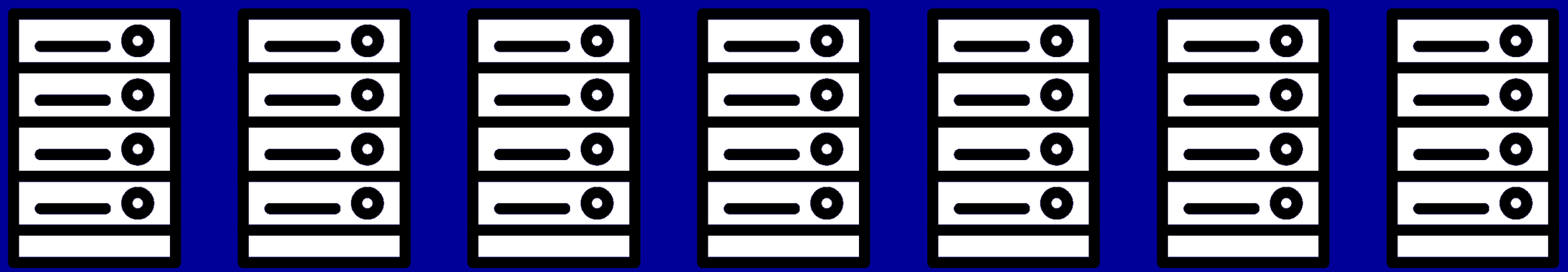
Focus of My Talk



Workload Evolution



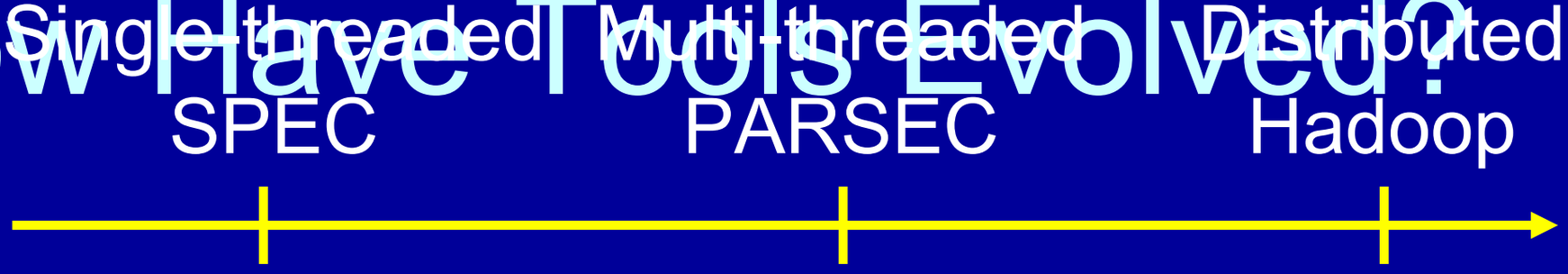
Workload Evolution



How Have Tools Evolved?

Workloads

Tools



Perf counters
gprof
Valgrind
Pin
gem5
???

Studying Scale-Out Workloads

Three fundamental requirements

Transparency

Instrumentation

Full-System

Transparency

The property that program analysis has ZERO effect on program environment and execution

Instrumentation

The ability for a user to write arbitrary tools for program analysis using a dedicated API

Full-System

The ability to study the entire system stack from BIOS to user space

Intel SAE

SAE is an x86 emulator
for dynamically instrumenting
scale-out workloads
across the entire software stack

What is Instrumentation?

A technique that inserts extra code into a program to collect runtime information

Source Instrumentation

```
int bar(int foo) {
  __bar_entry();
  int res = foo + 2;
  __bar_exit();
  return res;
}

int main() {
  __main_entry();
  int foo = 0;
  int t = bar(foo);
  ...
  __main_exit();
  return 0;
}
```

Binary Instrumentation

```
11001001000011000100011 0
call __bar_entry
00100010110101001011100 0
11101000010101010110011 0
10010101101001010010100 1
00000101000010110001110 1
10100011000001011011000 0
01011111010101001111011 1
01000111011010100110000 0
01001011110001010110101 0
00100101110100010001010 1
11111111001001010110011 0
call __bar_exit
00100111010001011010001 1
01010001010100101001001 1
```

Obviates need for recompiling or relinking

Why use Binary Instrumentation?

Enables instrumentation of existing binaries

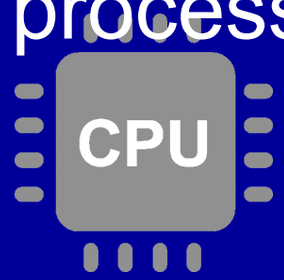
Dynamic Binary Instrumentation

Instrumentation intercepts application execution

Discover code at runtime

Handle dynamically generated code

Attach to running process



How is Binary Instrumentation Used?

Trace Generation

Architectural Structure Modeling

Fault Tolerance Studies

Emulating New Instructions

Call Graph Generation

Memory Leak Detection

Thread Profiling

Race Detection

How is Binary Instrumentation Used?

scholar.google.com

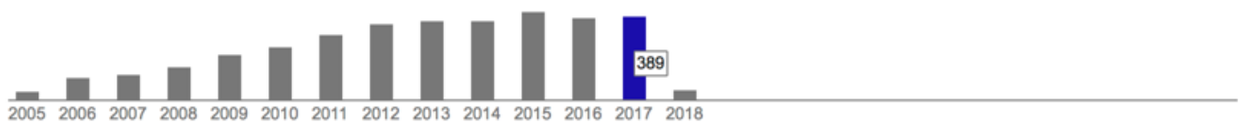
AUTONAVx Cou... Info | edX Luca http://media...bc6b69a4421 Favorites

Vijay Janapa Reddi - Google Scholar Citations

Google Scholar SIGN IN

Pin: building customized program analysis tools with dynamic instrumentation [PDF] from uci.edu

Authors	Chi-Keung Luk, Robert Cohn, Robert Muth, Harish Patil, Artur Klauser, Geoff Lowney, Steven Wallace, Vijay Janapa Reddi, Kim Hazelwood
Publication date	2005/6/12
Conference	Acm sigplan notices
Volume	40
Issue	6
Pages	190-200
Publisher	ACM
Description	Abstract Robust and powerful software instrumentation tools are essential for program analysis tasks such as profiling, performance evaluation, and bug detection. To meet this need, we have developed a new instrumentation system called Pin. Our goals are to provide easy-to-use, portable, transparent, and efficient instrumentation. Instrumentation tools (called Pintools) are written in C/C++ using Pin's rich API. Pin follows the model of ATOM, allowing the tool writer to analyze an application at the instruction level without the need for detailed knowledge of the underlying instruction set. The API is designed to be architecture independent whenever possible, making Pintools source compatible across different architectures. However, a Pintool can access architecture-specific details when necessary. Instrumentation with Pin is mostly transparent as the application and Pintool observe the …
Total citations	Cited by 3490



Scholar articles Pin: building customized program analysis tools with dynamic instrumentation
CK Luk, R Cohn, R Muth, H Patil, A Klauser, G Lowney… - Acm sigplan notices, 2005
Cited by 3490 Related articles All 50 versions

TITLE

Pin: building
CK Luk, R Coh
Acm sigplan nc

GPUWattch:
J Leng, T Hethe
ACM SIGARCH

A dynamic c
Q Wu, M Marto
Proceedings of

Web search
VJ Reddi, B Le
International Sy

PLR: A softw
A Shye, J Blom
IEEE Transacti

PIN: a binary
VJ Reddi, A Se
Proceedings of

Voltage eme
VJ Reddi, MS
High Performar

High-performance and energy-efficient mobile web browsing on big/little systems 100 2013
Y Zhu, VJ Reddi
High Performance Computer Architecture (HPCA2013), 2013 IEEE 19th …

Using process-level redundancy to exploit multiple cores for transient fault tolerance 100 2007
A Shye, T Moseley, VJ Reddi, J Blomstedt, DA Connors
Dependable Systems and Networks, 2007. DSN'07. 37th Annual IEEE/IFIP …

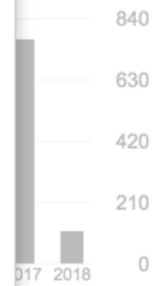
91 2007

LE

VIEW ALL

Since 2013

3658
20
30



VIEW ALL

Comp…

Harvar…

University of Rochester

Kim Hazelwood
Facebook

Jingwen Leng
Assistant Professor, Shanghai Ji…

Alex Shye

Open "https://scholar.google.com/scholar?oi=bibs&hl=en&cites=15305535231300712107&as_sdt=5&as_ylo=2017&as_yhi=2017" in a new tab

SAE Overview

Operates in Simics virtual environment

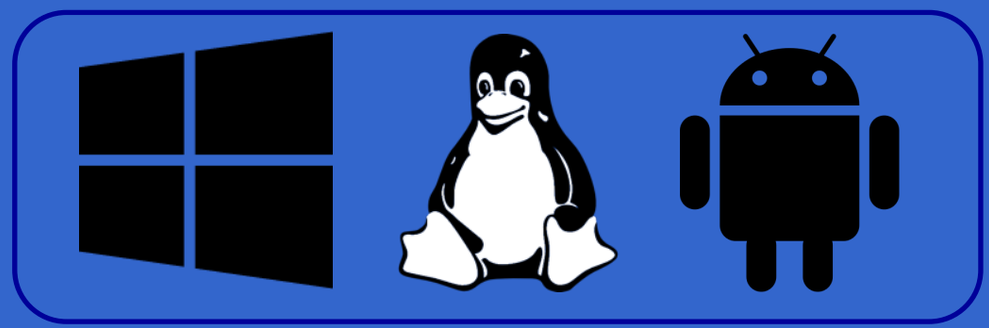
Tools operate in context of CPU, not OS

Tools exist outside of guest environment

Acts as a software CPU

SAE Overview

Operates in Simics virtual environment

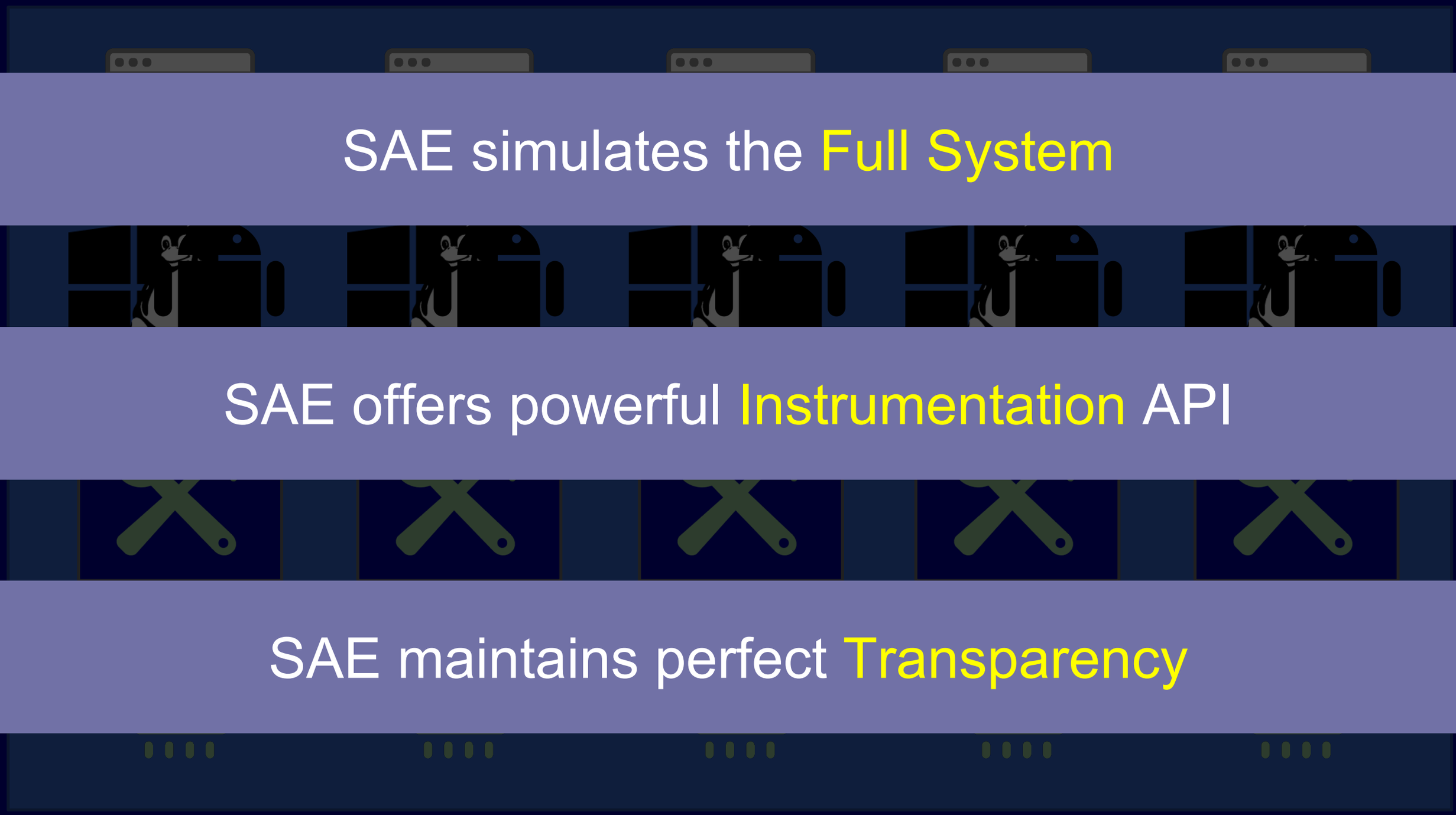


Tools operate in context of CPU, not OS

Tools exist outside of guest environment

Acts as a software CPU

SAE Overview



SAE simulates the Full System

SAE offers powerful Instrumentation API

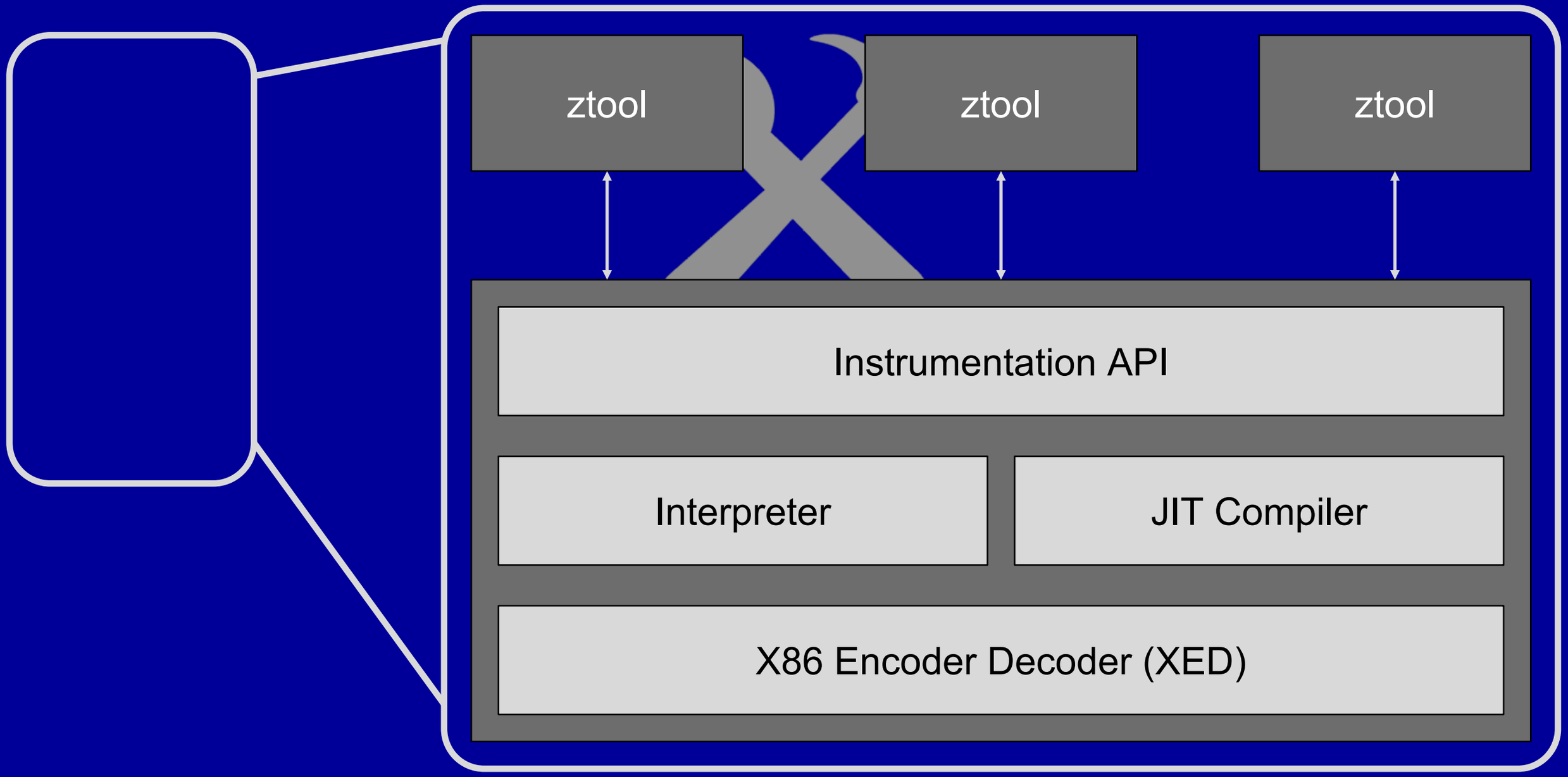
SAE maintains perfect Transparency

Terminal Shell Edit View Window Help 100% Mar 16 4:17 PM Daniel Richins

danielrichins — daniel@locutus: ~/BigData/Simics — ssh -p 2002 -Y daniel@locutus.ece.utexas.edu — 76×20

```
daniel@locutus:SAE$
```

SAE Components



Developing a ztool

```
#include "ztool-api.h"

static ztool_handle_t zhandle;
extern "C" void ztool_init(ztool_init_handle_t handle) {
  zhandle = ztool_init_get_tool_handle(handle);
}
```

Developing a ztool

```
#include "ztool-api.h"

static ztool_handle_t zhandle;
extern "C" void ztool_init(ztool_init_handle_t handle) {
  zhandle = ztool_init_get_tool_handle(handle);
  {
    ztool_instruction_exe_desc_t desc;
    desc.fn = count_inst;
    desc.when = ZTOOL_INSTRUCTION_WHEN_BEFORE;
    desc.data = NULL;
    desc.order = ZTOOL_CB_ORDER_DEFAULT;
    desc.config_key = ztool_init_get_default_config_key(handle);
    ztool_instruction_exe_register_cb(&desc);
  }
}
```

Developing a ztool

```
#include "ztool-api.h"

long unsigned icount = 0;
void count_inst(ztool_state_handle_t shandle, void *data) {
  icount++;
}

static ztool_handle_t zhandle;
extern "C" void ztool_init(ztool_init_handle_t handle) {
  zhandle = ztool_init_get_tool_handle(handle);
  {
    ztool_instruction_exe_desc_t desc;
    desc.when = ZTOOL_INSTRUCTION_WHEN_BEFORE;
    desc.fn = count_inst;
    desc.data = NULL;
    desc.order = ZTOOL_CB_ORDER_DEFAULT;
    desc.config_key = ztool_init_get_default_config_key(handle);
  }
}
```

Developing a ztool

```
#include "ztool-api.h"

long unsigned icount = 0;
void fini(ztool_fini_handle_t fhandle, void *data) {
  std::cout << "Inst count: " << icount << std::endl;
}

static ztool_handle_t zhandle;
extern "C" void ztool_init(ztool_init_handle_t handle) {
  ...
  {
    ztool_fini_desc_t desc;
    desc.fn = fini;
    desc.data = NULL;
    desc.order = ZTOOl_CB_DEFAULT_ORDER;
    desc.config_key = ztool_init_get_default_config_key(handle);
    ztool_fini_register_cb(&desc);
  }
}
```

Instrumentation Engine
CPU-Level Events

Configuration events

Per event data		
Tool loaded	Instruction/basic-block discovery (caching)	Dynamic reconfiguration

Run-time events

Architectural state (registers, memory, active core) + per event data

Instruction execution	Memory access	State change	Architectural-events (e.g. ring transition, cr3 modification, machine mode modification)
All instructions	By instruction	Interrupt (asynchronous)	
By instruction format	By interrupt/exception/ page-walk	Exception (synchronous)	
By opcode + address			

Finalization events

Per event data		
Tool unloaded	Simulation termination	Instruction deletion (un-caching)

Supporting Multi-Core

```
#include "ztool-api.h"

long unsigned *mc_icount = NULL;
void fini(ztool_fini_handle_t fhandle, void *data) {...}
void count_inst(ztool_state_handle_t shandle, void *data) {...}

extern "C" void ztool_init(ztool_init_handle_t handle) {
  int num_cores = ztool_init_get_core_count(handle);
  mc_icount = new long unsigned[num_cores];
  { ... }
  { ... }
}
```

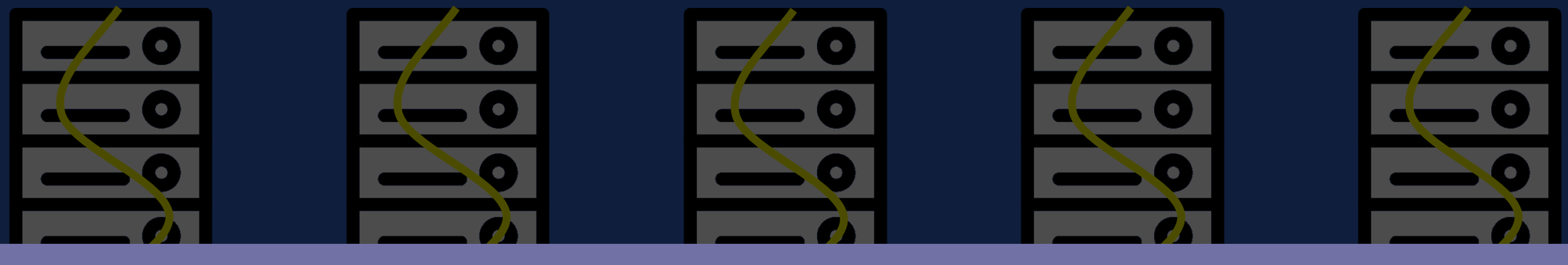
Supporting Multi-Core

```
#include "ztool-api.h"

long unsigned *mc_icount = NULL;
void fini(ztool_fini_handle_t fhandle, void *data) {...}
void count_inst(ztool_state_handle_t shandle, void *data) {
  unsigned core = ztool_state_get_core_num(shandle);
  mc_icount[core]++;
}

extern "C" void ztool_init(ztool_init_handle_t handle) {
  int num_cores = ztool_init_get_core_count(handle);
  mc_icount = new long unsigned[num_cores];
  { ... }
  { ... }
}
```

Supporting Multi-System



ztool_init() is called by each system's thread

Supporting Multi-System

```
...
static ztool_system_data_key_handle_t ms_key;
struct system_data_t {
    ztool_handle_t zhandle;
    unsigned core_count;
    long unsigned *mc_icount;
};
...
extern "C" void ztool_init(ztool_init_handle_t handle) {
  ztool_handle_t zhandle = ztool_init_get_tool_handle(handle);
  system_data_t *sdata = new system_data_t();
  sdata->zhandle = zhandle;
  sdata->core_count = ztool_init_get_core_count(handle);
  sdata->mc_icount = new long unsigned[sdata->core_count];

  ztool_system_alloc_data_key(zhandle, &ms_key);
  ztool_system_set_data(zhandle, ms_key, sdata);
  {...}
  {...}
}
```

Supporting Multi-System

```
...
struct system_data_t {
    ztool_handle_t zhandle;
    unsigned core_count;
    long unsigned *mc_icount;
};

void count_inst(ztool_state_handle_t shandle, void *data) {
  system_data_t *sdata = reinterpret_cast<system_data_t*>(
    ztool_state_get_system_data(shandle, ms_key));
  unsigned core_num = ztool_state_get_core_num(shandle);
  sdata->mc_icount[core_num]++;
}
...
```

Supporting Multi-System

```
#include <pthread.h>
static pthread_mutex_t print_mutex =
    PTHREAD_MUTEX_INITIALIZER;
...
void fini(ztool_fini_handle_t fhandle, void *data) {
  system_data_t *sdata = reinterpret_cast<system_data_t*>(
  for (int i = 0, ie = sdata->core_count; i < ie; ++i)
    cout << "Core " << i << " instruction count: "
         << sdata->mc_icount[i] << endl;
  pthread_mutex_unlock(&print_mutex);
}
...
```

Like any multi-threaded program,
ensuring thread safety is paramount

```
[service_node_cmp0.sn spec-viol] DHCP: INIT-REBOOT request with wrong 'requested ip' option; expected 10.10.0.107, got 10.0.2.15
System viper0
Core 0 instruction count: 140641350109337
Core 1 instruction count: 140645315602855
[viper0.mb.cpu0.core[0][0] info] ztool was unloaded
System viper1
Core 0 instruction count: 140625251400256
Core 1 instruction count: 140629002511079
[viper1.mb.cpu0.core[0][0] info] ztool was unloaded
System viper2
Core 0 instruction count: 140631007558963
Core 1 instruction count: 140651733768558
[viper2.mb.cpu0.core[0][0] info] ztool was unloaded
System viper3
Core 0 instruction count: 140648782566388
Core 1 instruction count: 140642041284016
[viper3.mb.cpu0.core[0][0] info] ztool was unloaded
System viper4
Core 0 instruction count: 140637433921720
Core 1 instruction count: 140633699848333
[viper4.mb.cpu0.core[0][0] info] ztool was unloaded
System viper5
Core 0 instruction count: 140629793324321
Core 1 instruction count: 140632912909424
[viper5.mb.cpu0.core[0][0] info] ztool was unloaded
System viper6
Core 0 instruction count: 140641428409636
Core 1 instruction count: 140635740830883
[viper6.mb.cpu0.core[0][0] info] ztool was unloaded
System viper7
Core 0 instruction count: 140630570713093
Core 1 instruction count: 140632395766590
[viper7.mb.cpu0.core[0][0] info] ztool was unloaded
System viper8
Core 0 instruction count: 140642074149488
Core 1 instruction count: 140636308673006
[viper8.mb.cpu0.core[0][0] info] ztool was unloaded
System viper9
Core 0 instruction count: 140620546722519
Core 1 instruction count: 140620411071336
[viper9.mb.cpu0.core[0][0] info] ztool was unloaded
^Cscript interrupted by the user
Interrupting script.
simics>
```

Other SAE Power Features

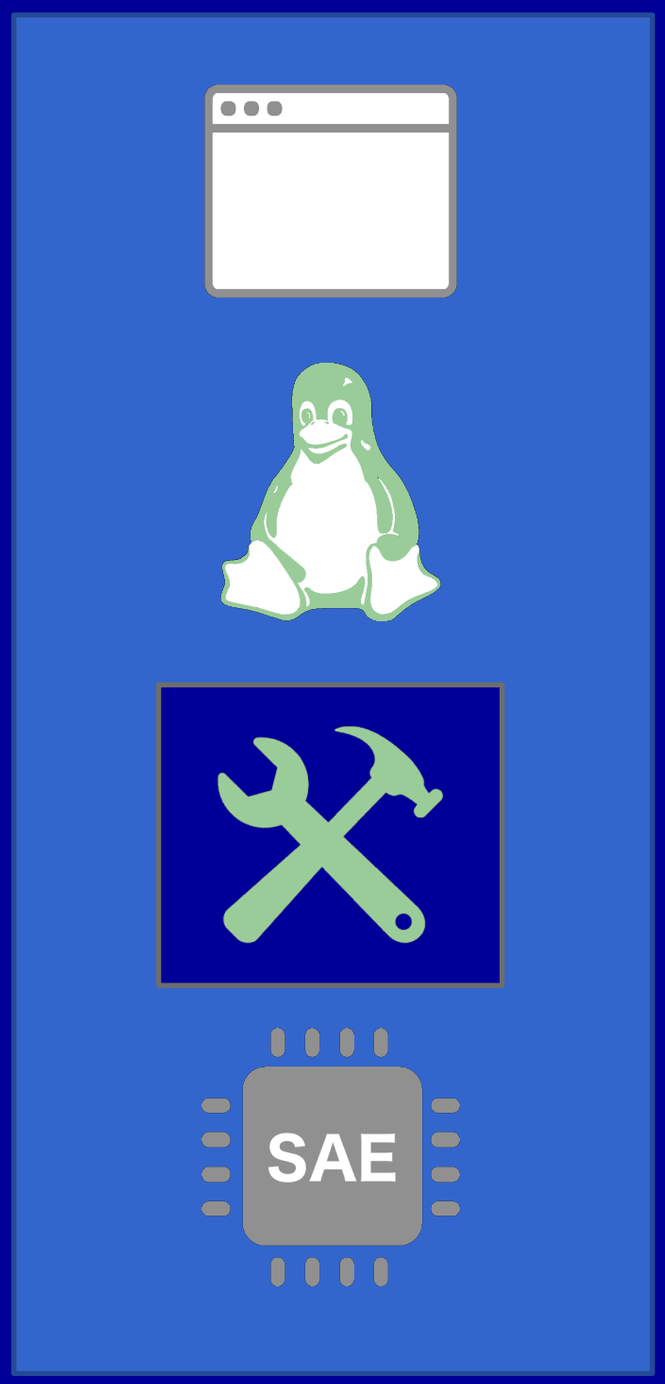
OS Awareness

Dynamic Reconfiguration

CPU Modification & ISA Extensions

Networking

OS Awareness

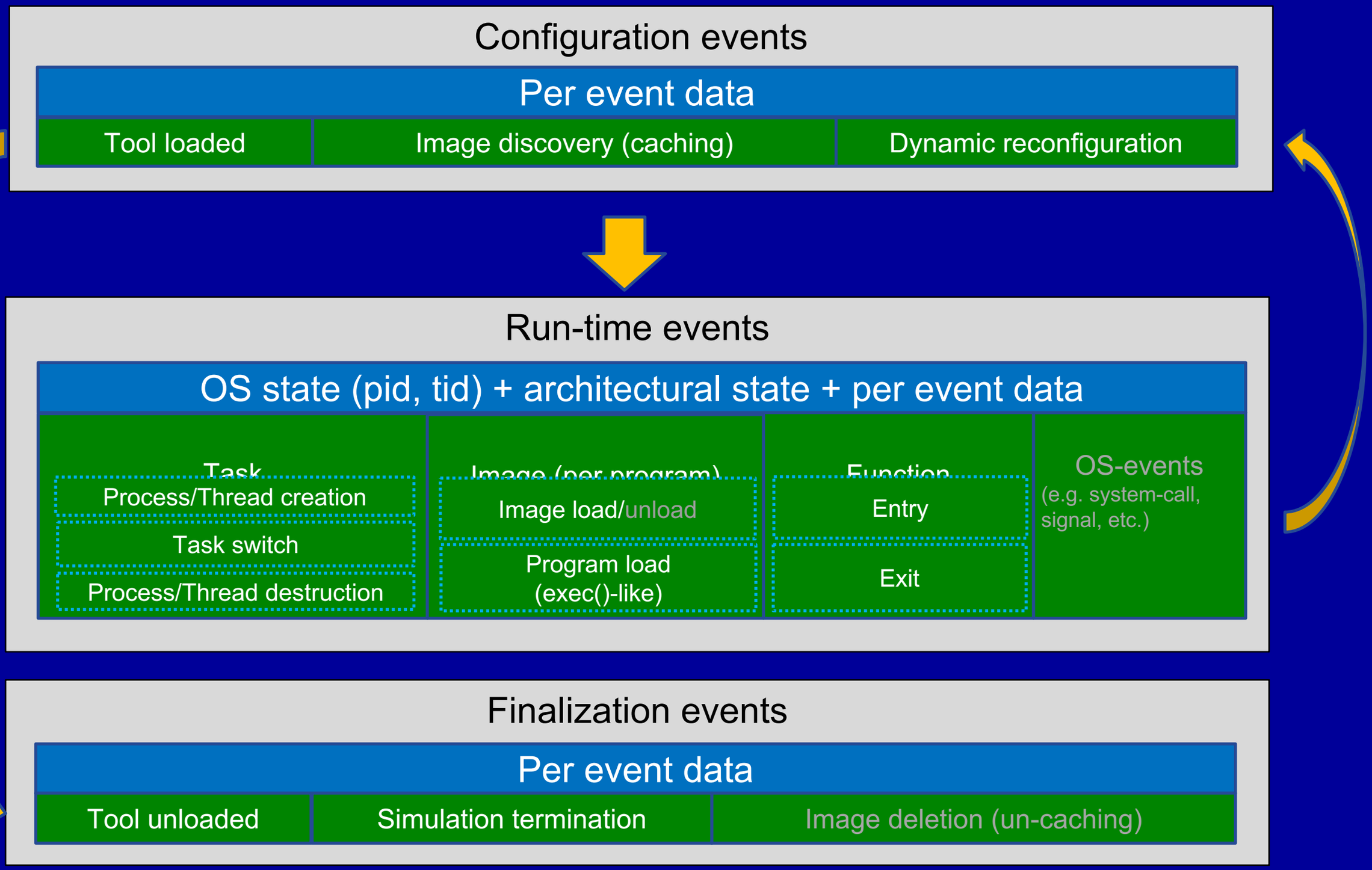


As a convenience to users, SAE provides OS-aware APIs

Currently only available for Linux

Implemented as ztools, so OS experts can extend OS awareness to additional platforms

Instrumentation Engine
OS-Level Events



OS Awareness

```
#include "ztool-api.h”
...
extern "C" void ztool_init(ztool_init_handle_t handle) {
  {
    ztool_os_staged_event_desc_t desc;
    desc.os_event = ZTOOL_OS_STAGED_EVENT_TASK_SWITCH;
    desc.when = ZTOOL_OS_WHEN_AFTER;
    desc.fn = task_switch;
    desc.data = NULL;
    desc.order = ZTOOL_CB_ORDER_DEFAULT;
    desc.config_key = ztool_init_get_default_config_key(handle);
    desc.zhandle = ztool_init_get_tool_handle(handle);
    ztool_os_staged_event_register_cb(&desc);
  }
}
```

OS Awareness

```
#include "ztool-api.h”

static void task_switch(ztool_os_staged_event_handle_t evhandle,
    void* data) {
  ztool_os_task_handle_t task =
      ztool_os_get_current_task(evhandle.oshandle);
  int pid = ztool_os_task_get_pid(current_task);
  int tid = ztool_os_task_get_tid(current_task);
  static char name[16];
  ztool_os_task_get_name(current_task, name, sizeof(name));
  cout << setw(16) << left << name << " pid=" <<
      setw(5) << pid << " tid=" << setw(5) << tid << endl;
}

extern "C" void ztool_init(ztool_init_handle_t handle) {
  {...}
}
```

OS Awareness

```
calculix          pid=1322  tid=1322
swapper           pid=0     tid=0
kblockd/1         pid=29    tid=29
calculix          pid=1322  tid=1322
swapper           pid=0     tid=0
kblockd/1         pid=29    tid=29
calculix          pid=1322  tid=1322
swapper           pid=0     tid=0
kblockd/1         pid=29    tid=29
calculix          pid=1322  tid=1322
swapper           pid=0     tid=0
kblockd/1         pid=29    tid=29
calculix          pid=1322  tid=1322
swapper           pid=0     tid=0
```

Meeting the Requirements

Transparency

Instrumentation

Performance Counters

Pin, Valgrind, DynamoRIO

SAE

QEMU

PinOS

gem5 FS

Full-System

Simulation and Analysis Engine for Scale-Out Workloads

Nadav Chachmon[†] Daniel Richins[‡] Robert Cohn[†]
Magnus Christensson[†] Wenzhi Cui[‡] Vijay Janapa Reddi[‡]

[†]Intel Corporation [‡]The University of Texas at Austin

ABSTRACT

We introduce a system-level Simulation and Analysis Engine (SAE) framework based on dynamic binary instrumentation for fine-grained and customizable instruction-level introspection of everything that executes on the processor. SAE can instrument the BIOS, kernel, drivers, and user processes. It can also instrument multiple systems simultaneously using a single instrumentation interface, which is essential for studying scale-out applications. SAE is an x86 instruction set simulator designed specifically to enable rapid prototyping, evaluation, and validation of architectural extensions and program analysis tools using its flexible APIs. It is fast enough to execute full platform workloads—a modern operating system can boot in a few minutes—thus enabling research, evaluation, and validation of complex functionalities related to multicore configurations, virtualization, security, and more. To reach high speeds, SAE couples tightly with a virtual platform and employs both a just-in-time (JIT) compiler that helps simulate simple instructions efficiently and a fast interpreter for simulating new or complex instructions. We describe SAE's architecture and instrumentation engine design and show the framework's usefulness for single- and multi-system architectural and program analysis studies.

CCS Concepts

•**Computing methodologies** → **Simulation environments**; **Simulation tools**; *Interactive simulation;*

Keywords

Analysis, instrumentation, transparency, full-system, scale-out, big data, JIT, multicore, multisystem

SAE website: https://software.intel.com/en-us/intel-sae-sdk

This research was funded by the U.S. Government. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing official policies, either expressed or implied, of the U.S. Government.

ICS '16, May 29-June 02, 2016, Istanbul, Turkey

 ISBN 978-1-4503-4361-9/16/05…$15.00

DOI: http://dx.doi.org/10.1145/2925426.2926293

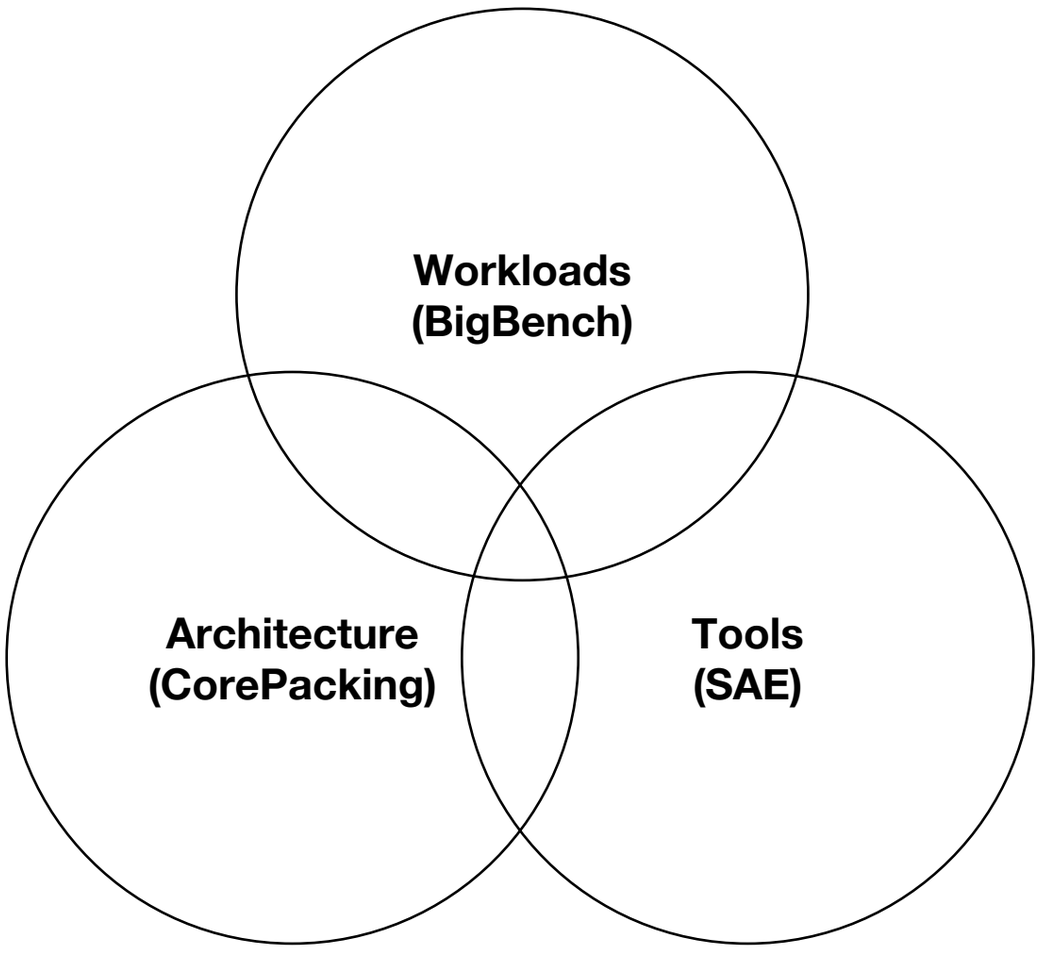
1. INTRODUCTION

The landscape of computing continues to evolve rapidly as do the workloads. Computing workloads have evolved from being simply single- or multithreaded to running across distributed systems, mostly driven by large-scale programming frameworks, such as Spark and Hadoop, that support scale-out applications and analytics. With the emergence and proliferation of these workloads, there is a need for new and robust instrumentation tools that can facilitate deep program introspection without compromising transparency. Characterizing and analyzing scale-out workloads has been challenging, if not impossible, due to the lack of industry-strength tools that can enable fine-grained, transparent program introspection both within and across nodes.

We identify three fundamental requirements for tools to study scale-out workloads. (1) *Instrumentation*: Researchers must be able to build a centralized and comprehensive view of distributed execution. This requires that they be able to write arbitrary tools and control them from a centralized interface. (2) *Full-system*: Datacenter workloads rely on kernel services and inter-process interactions. A tool to study these workloads must capture everything that executes on a processor, both in kernel- and user-space and across processes and nodes. (3) *Transparency*: In a scale-out workload, the instrumentation tool must be transparent not only to the process under study, but also to its interactions with the rest of the system and the rest of the network. A single instrumented system running slower than the rest of the nodes, for example, could compromise transparency because the node interactions would be altered by the changed speed.

We introduce the Intel Simulation and Analysis Engine (SAE)—a system-level dynamic binary instrumentation engine that meets all three requirements. (1) SAE supports fine-grained distributed workload instrumentation with a simple API to create arbitrary analysis tools. It instruments all systems from a single interface, enabling a coherent view of distributed execution. (2) At its heart, SAE is a processor model that simulates instruction execution in the context of a full or distributed system. Hence, it is not limited to user-space exploration, instead capturing literally all activity on a CPU, including even kernel, driver, and BIOS operations. (3) SAE is unique in being built atop a mature virtual system platform: Wind River Simics. Consequently, SAE resides entirely in the host machine's space: it uses none of the virtual machines' memory space, nor does it change the progress of time within a virtual machine. And because all the systems are controlled from within SAE, there is no relative slowdown between any two machines.

In Summary



Growing need for academia and industry to collaborate with one another

Big data requires a **holistic** view.

Acknowledgments

Daniel Richins
Wenzhi Cui
Nadav Chachmon
Magnus Christensson
Tahrina Ahmed
Russell Capp
Robert Cohn
Bhaskar Gowda
Intel SAE team
Intel BigBench team

Thank You!