Design Optimization of Computing System

Introduction

Mainack Mondal
Sandip Chakraborty

CS 60203
Autumn 2024

Today's class

- **Course logistics**
- Why performance?
- What is performance? Really
- Case studies
- Terminology
- Measurement
  - Observability
    - Counters
    - Profiling
    - Tracing
  - Experimentation

Instructors

- **Mainack Mondal:** system security and privacy, human factors, networked systems, internet measurement
  - Office: CSE 316

Instructors

- **Sandip Chakraborty:** Computer systems, human computer interaction and machine learning
  - Office: CSE 311

Two TAs

Shiladitya De

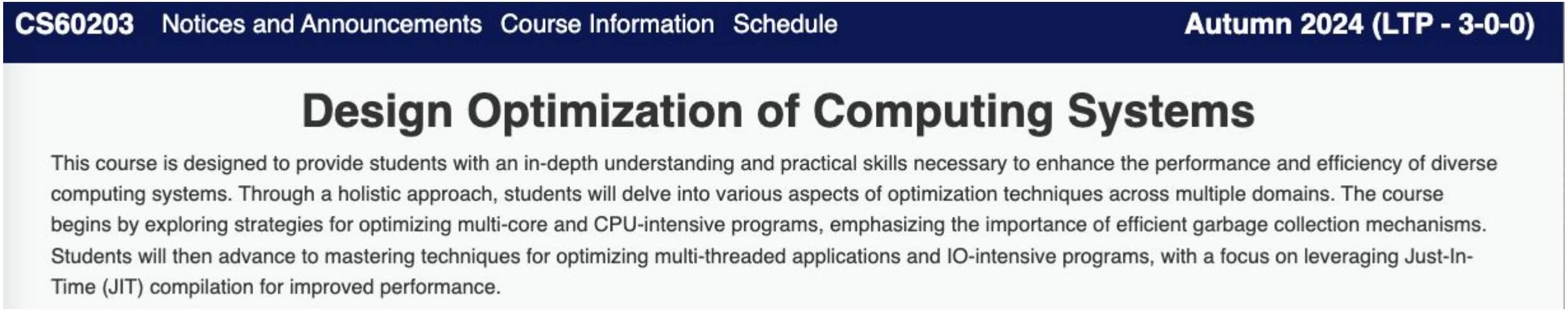
shiladitya [DOT] de [AT] kgpian [DOT] iitkgp [DOT] ac [DOT] in

Yatindra Indoria

yatindraindoria75 [AT] kgpian [DOT] iitkgp [DOT] ac [DOT] in

Website

https://kronos-192081.github.io/DOCS-2024/



Course timings

- Credit : 3 – 0 – 0

- Thursday 3:00 noon - 5:00 pm
- Friday 3:00 pm to 4:00 pm

Mode of teaching

- Offline lectures
  - Please come to the class (no recordings)

- (occasional) Pre-recorded lectures for special topics
  - We will upload the recorded lectures via MS teams

- Two exams + scribe + 3-4 assignments

MS teams

- Will be announced soon

CSE Moodle

- All submissions will be via CSE moodle unless otherwise stated
  - Search in CSE Moodle for “CS60203: Design Optimization of Computing Systems”
    - Use the key STUD-DOCS-MM

Course evaluation: Exam

- Two Exams (60%)
  - Syllabus : Everything until that point
  - Dates will be in the webpage and announced in academic calendar

Course evaluation: Assignment / Scribes

- 40% of the evaluation
  - Expected: You apply the knowledge gained from this class on a hands on practical problem
  - We will communicate about fixing specific scribes for specific lectures

Course logistics

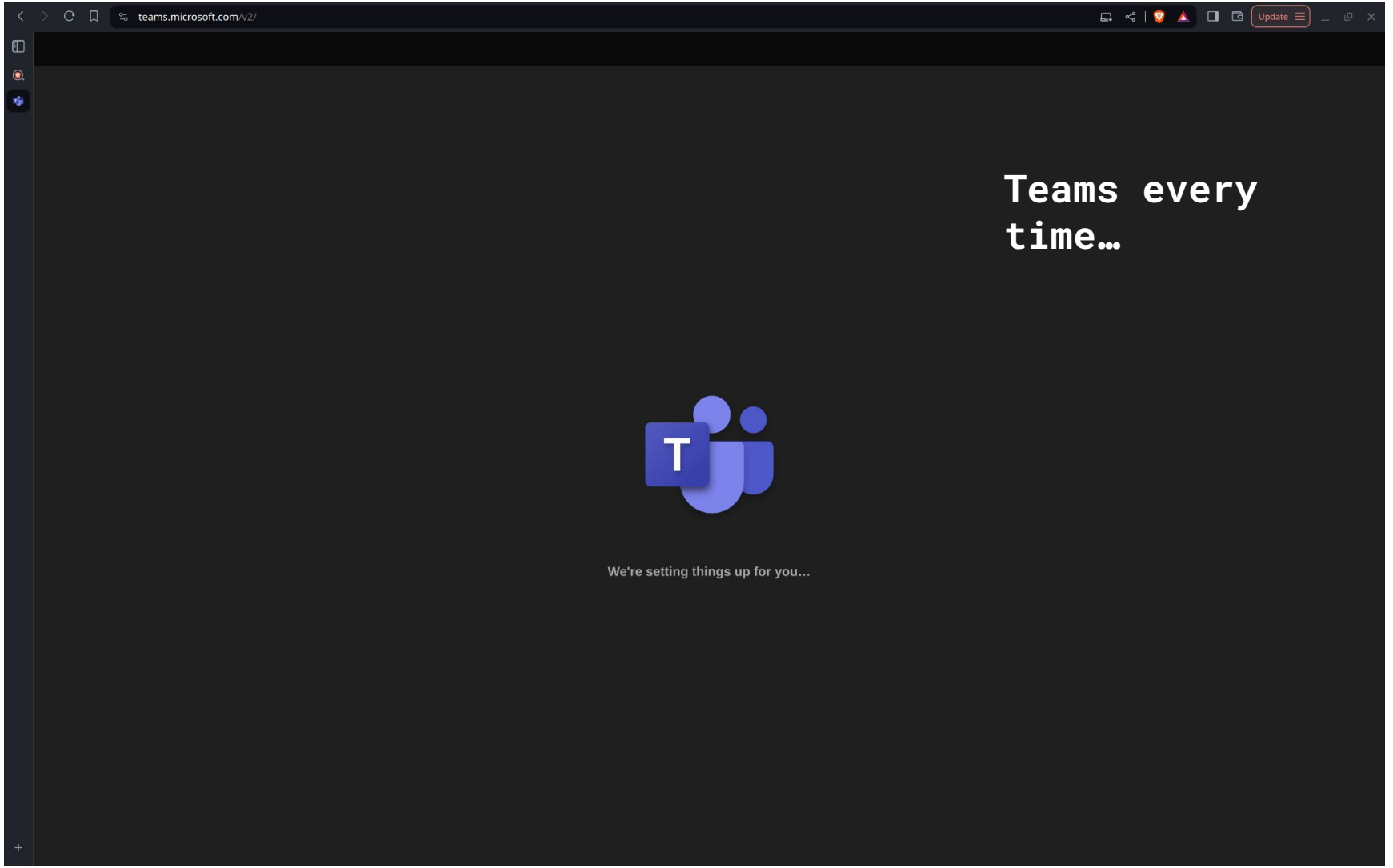
- Questions?

Today's class

- Course logistics
- **Why performance?**
- What is performance? Really
- Case studies
- Terminology
- Measurement
  - Observability
    - Counters
    - Profiling
    - Tracing
  - Experimentation

Why performance?

Teams loading times



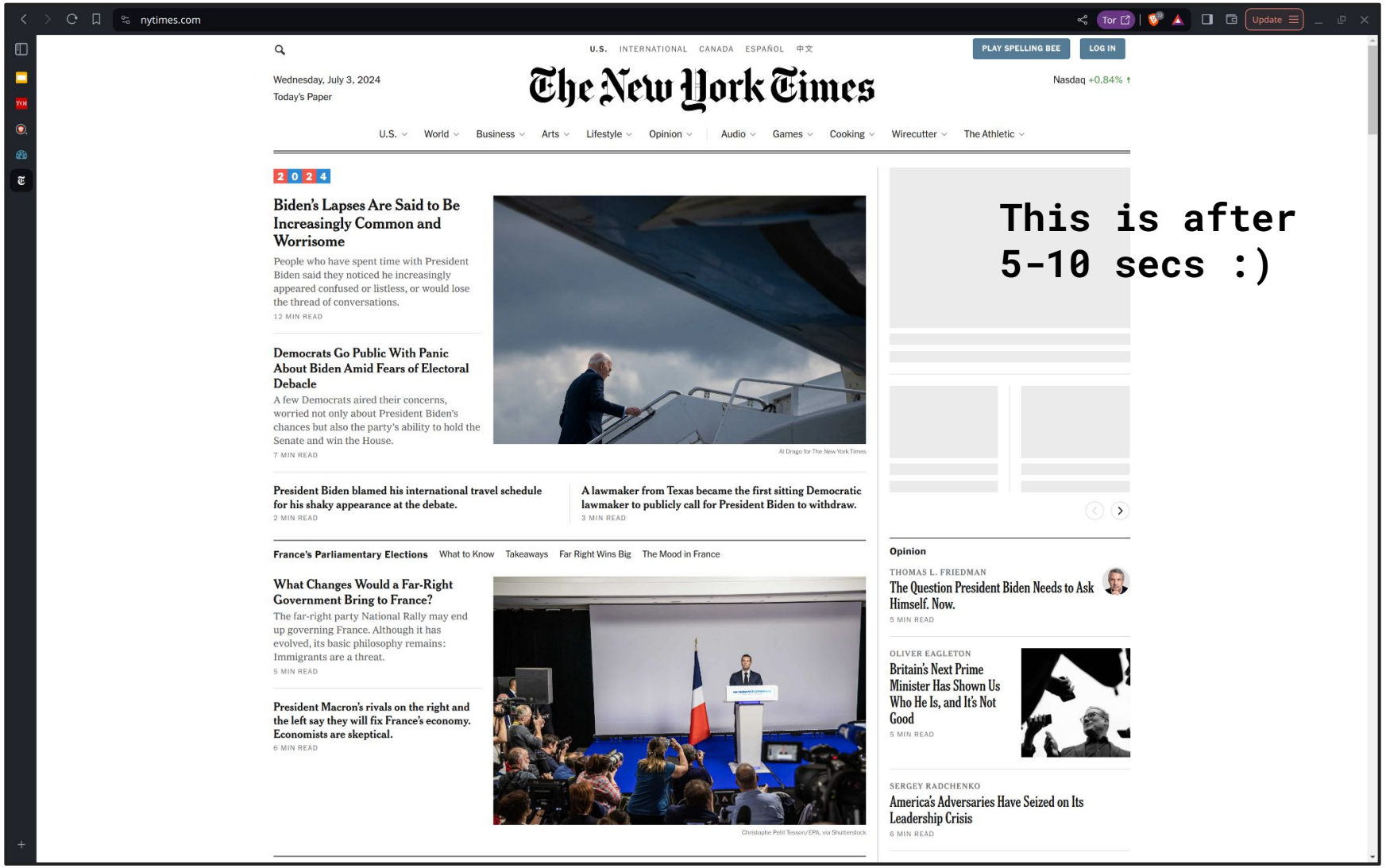
ERP

CV spaces disappear, or whole CV disappears

Subject registration => ERP down 70% of the time

Really what even works…

NYTimes loading times

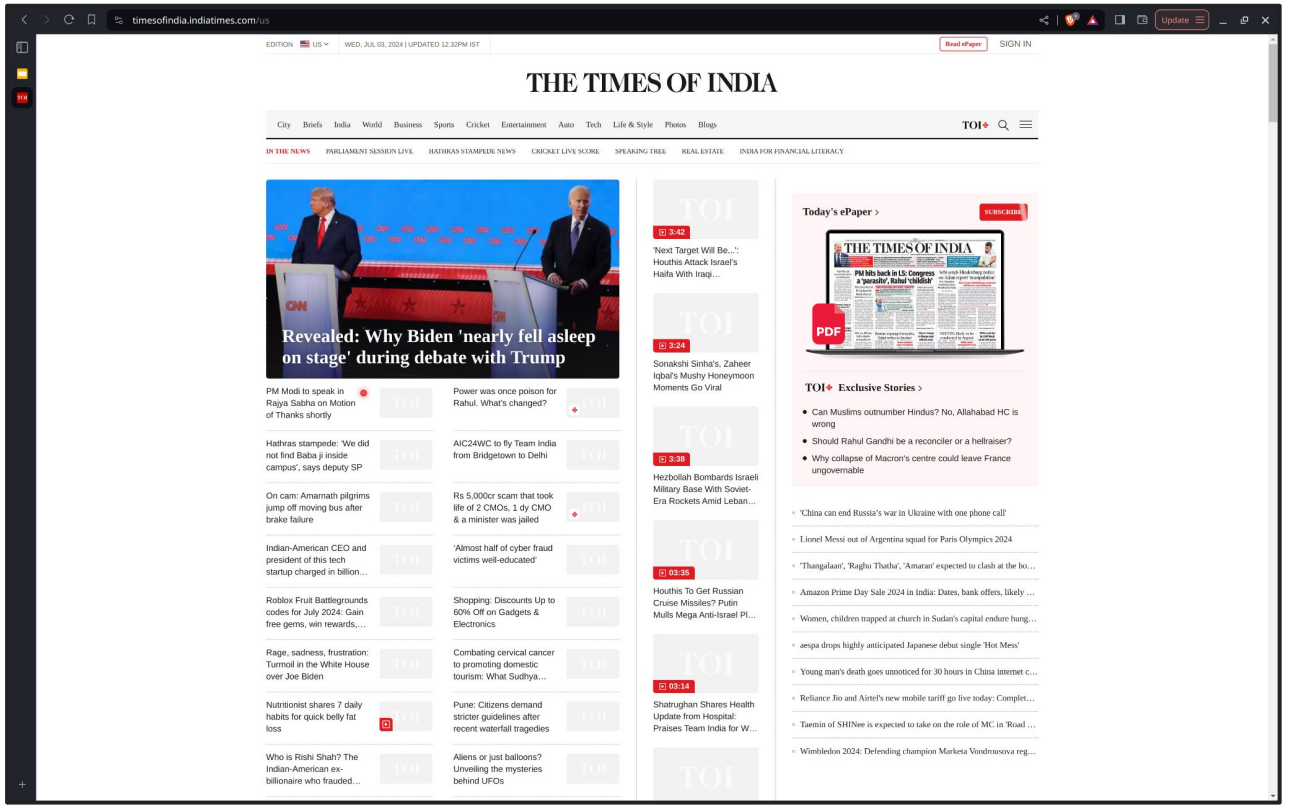


Moodle in online sems

Software often doesn't **perform** well

- **Windows search:**
  slow, and wrong
- **Microsoft Teams:**
  slow, and resource intensive
- **News websites:**
  slow, and bloated

…



Why should we care…professionally?

There are two general reasons for doing this:

- Improving price/performance: especially for large environments, where even small performance wins can add up to large savings in IT spend.
- Reducing latency outliers: for an environment of any size, occasional high latency I/O can slow application requests, causing unhappy customers.

Systems Performance: Enterprise and the Cloud, 1st Edition (2013)

Today's class

- Course logistics
- Why performance?
- **What is performance? Really**
- Case studies
- Terminology
- Measurement
  - Observability
    - Counters
    - Profiling
    - Tracing
  - Experimentation

What is performance? Really

What is performance?

System behaves in a desirable manner, across all intended circumstances, and usage patterns

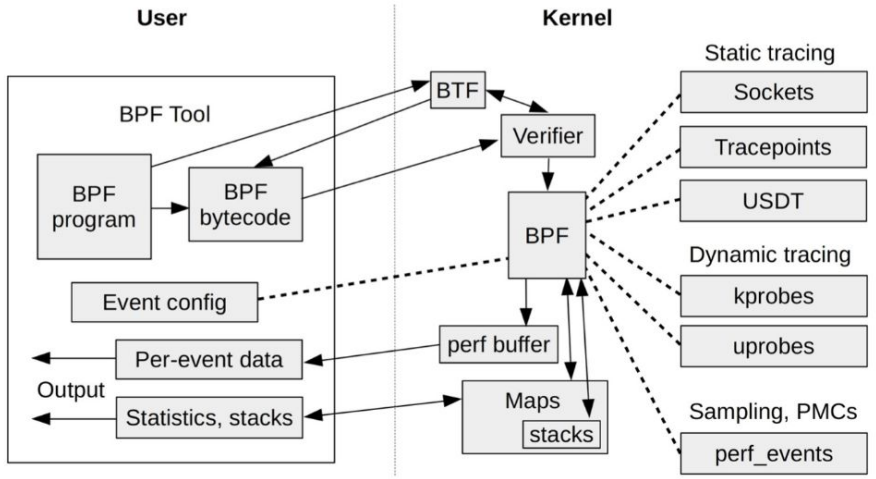
Which can encompass multiple things, like a responsive UI, or handling millions of concurrent users.

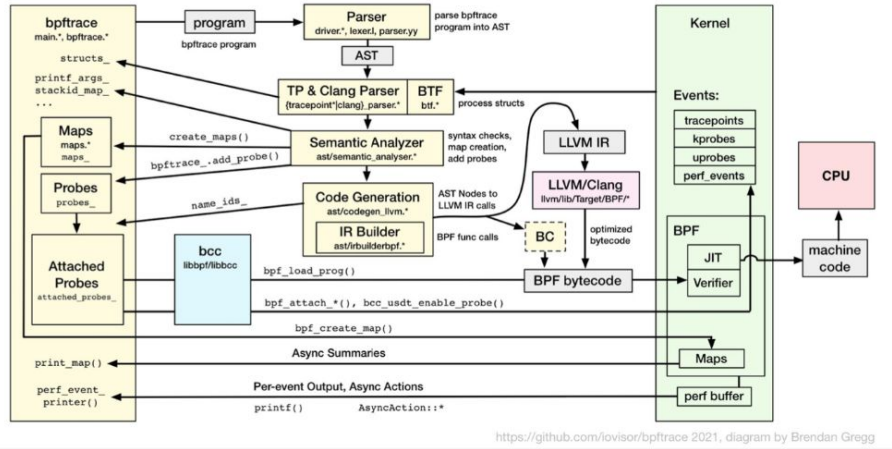
What is performance?

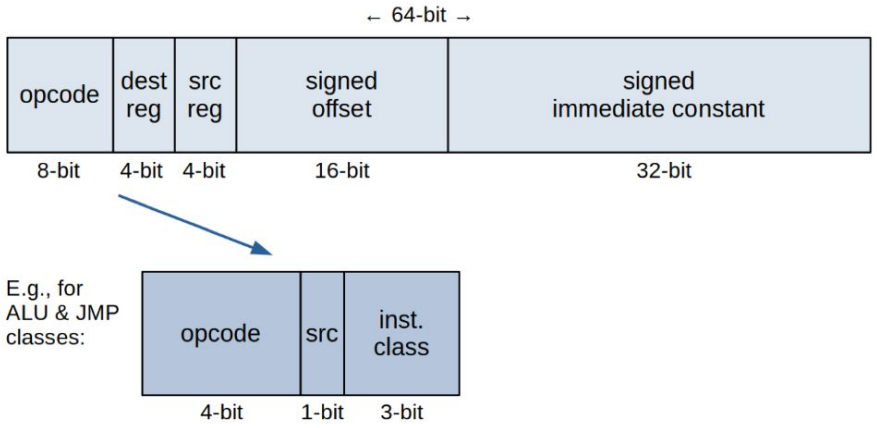
1. **Scaling** systems to handle **variable** loads
2. Reducing system **latency**
3. Increasing system **throughput**
4. Reliability, durability, and **availability**

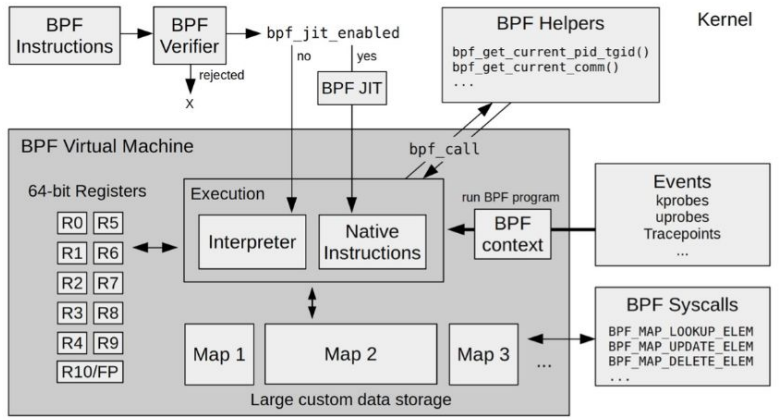
…

How to achieve performance?





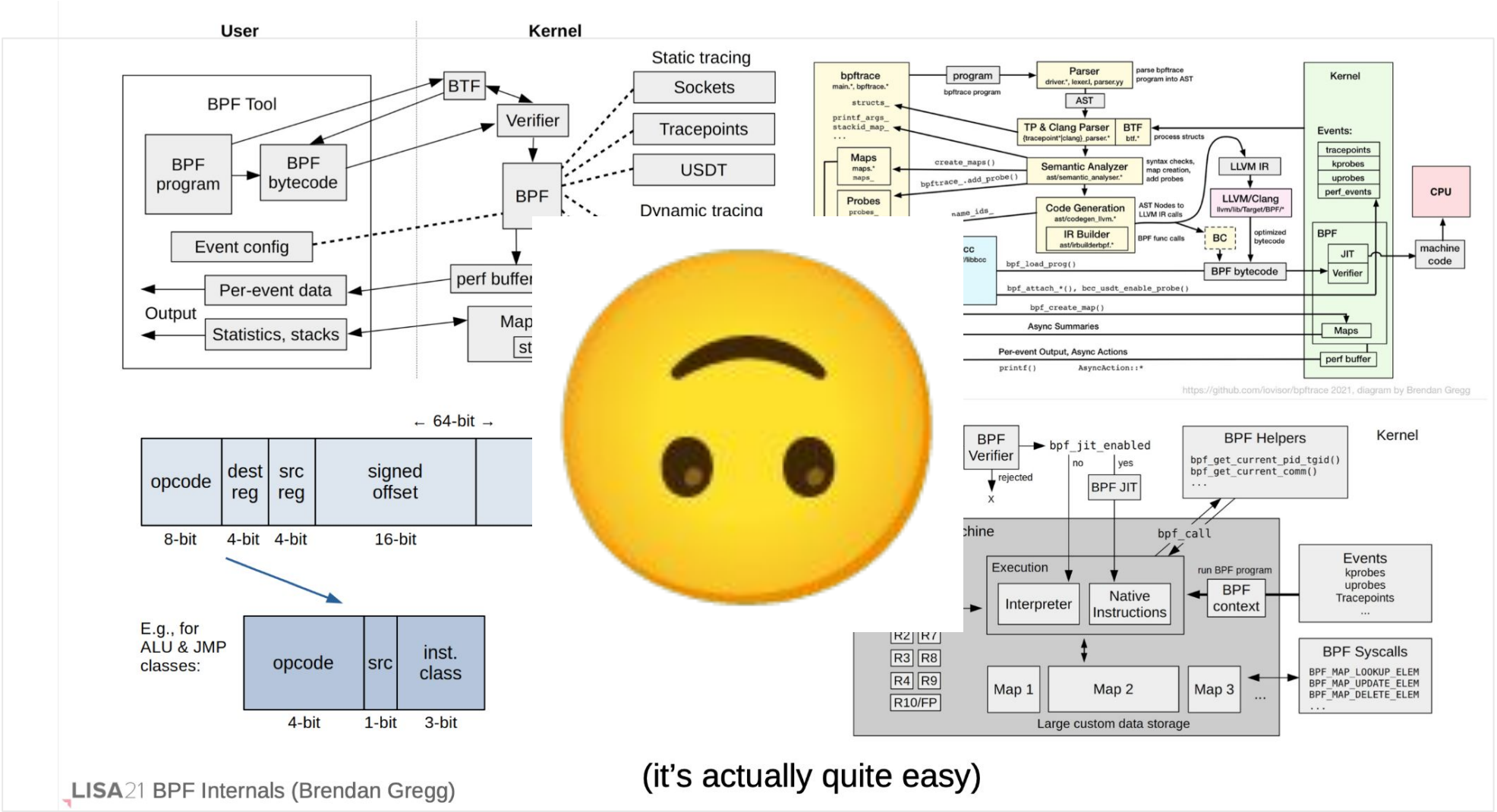




(it's actually quite easy)

https://www.brendangregg.com/Slides/LISA2021_BPF_Internals.pdf

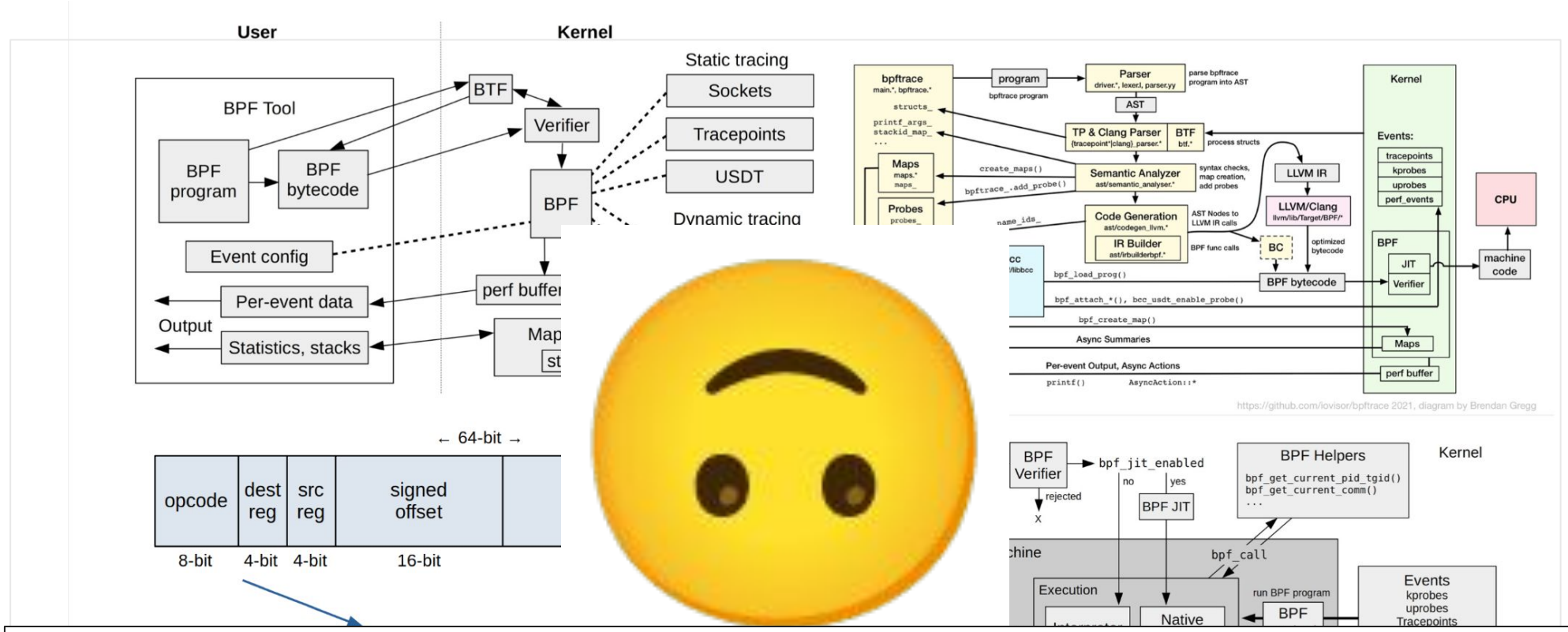
How to achieve performance?



(it's actually quite easy)

https://www.brendangregg.com/Slides/LISA2021_BPF_Internals.pdf

How to achieve performance?



* used to convey confusion, dry humor, or
to show that the user isn't being serious

https://www.brendangregg.com/Slides/LISA2021_BPF_Internals.pdf

Today's class

- Course logistics
- Why performance?
- What is performance? Really
- **Case studies**
- Terminology
- Measurement
  - Observability
    - Counters
    - Profiling
    - Tracing
  - Experimentation

Case Studies

Some cool examples!

We have seen the bad, now it’s time for the good!

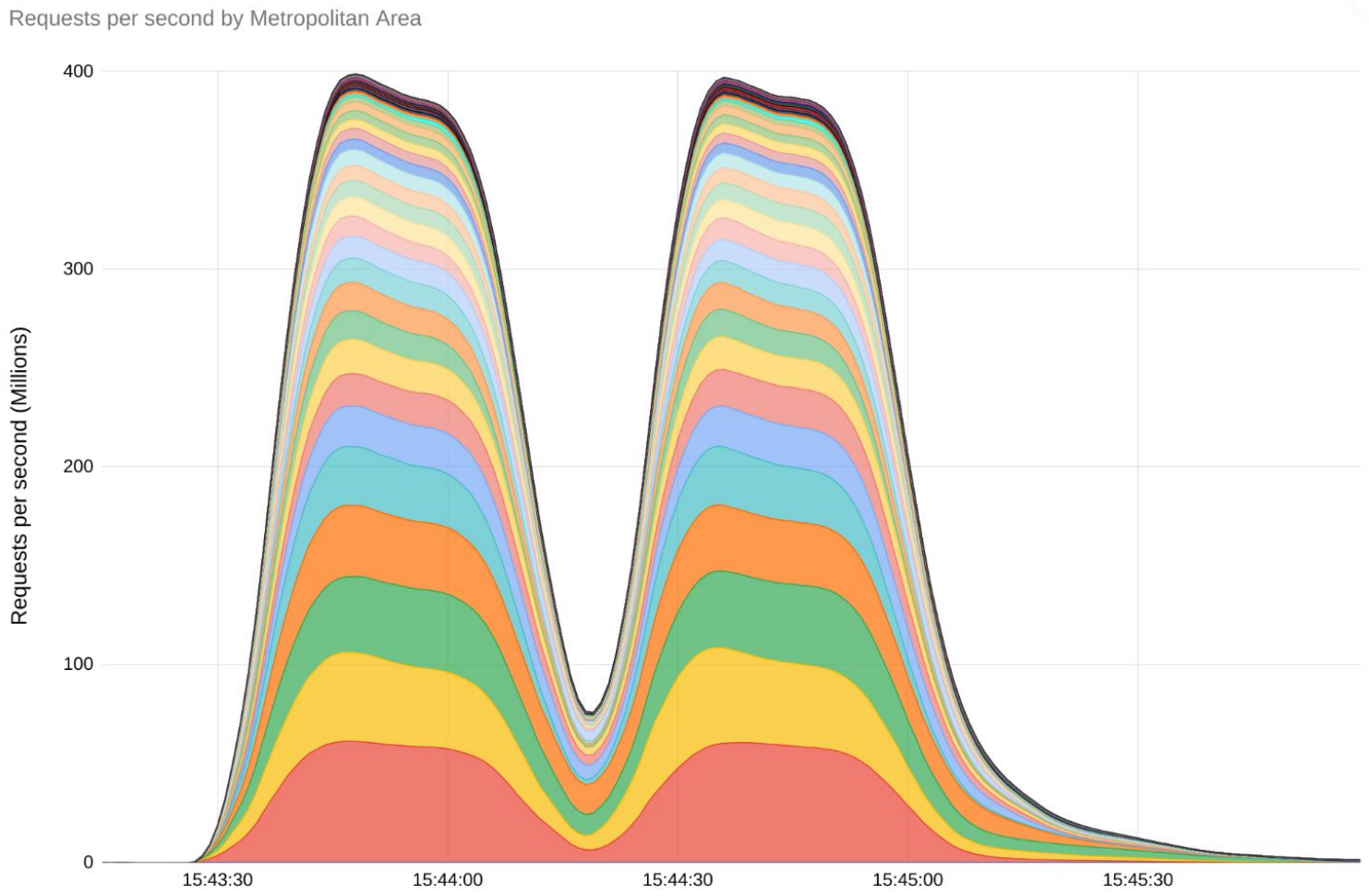
Let’s look at what we can do with the **state of the art performance**

Case studies

1. Adapting to a huge **DDoS spike!**
2. Trading at **O(100ns) latency**
3. Handling **O(100M) transactions/second** on O(100 Trillion) objects
4. **~100% uptime** by simulating every failure in production, just to stay ready!

Case Study 1: DDoS attack on Google

…This new series of **DDoS attacks reached a peak of 398 million requests per second (rps)**, …largest-recorded DDoS attack peaked at 46 million rps…



Google Cloud mitigated largest DDoS attack, peaking above 398 million rps

Case Study 1: Scaling to Variable Loads

This is the case of **history's largest DDoS attack**. The attack was using a novel "Rapid Reset" technique that leverages stream multiplexing, a feature of the widely-adopted HTTP/2 protocol.

Google's service went from receiving **<10M rps to 400M rps** in under a minute!

Google Cloud mitigated largest DDoS attack, peaking above 398 million rps

Case Study 2: Low Latency Trading

As a community, it's starting to push the limits of physics. Today it is possible to buy a custom ASIC (application- specific integrated circuit) to parse market data and send executions in 740 nanoseconds (or 0.00074 milliseconds).[4] (Human reaction time to a visual stimulus is around 190 million nanoseconds.)

This is HFT in 2013!

Barbarians at the Gateways - ACM Queue

Case Study 2: Low Latency at Scale!

"…To give you a feeling of scale, the current exchange technology is benchmarked in nightly builds to run a series of simulated market data feeds at **1 million messages per second, as a unit test…**"

This is HFT in 2013!

Barbarians at the Gateways - ACM Queue

Case Study 3: Amazon S3 Throughput

Fast forward to 2023, Amazon Simple Storage Service (Amazon S3) holds more than 280 trillion objects and averages over 100 million requests per second. To protect data integrity, Amazon S3 performs over four billion checksum computations per second. Over the years, we added many capabilities, such as a range of storage classes, to store your colder data cost effectively. Every day, you restore on average more than 1 petabyte from the S3 Glacier Flexible Retrieval and S3 Glacier Deep Archive storage classes. Since launch, you have saved $1 billion from using Amazon S3 Intelligent-Tiering. In 2015, we added the possibility of replicating your data across Regions. Every week, Amazon S3 Replication moves more than 100 petabytes of data. Amazon S3 is also at the core of hundreds of thousands of data lakes. It also has become a critical component of a growing ecosystem of serverless applications. Every day, Amazon S3 sends over 125 billion event notifications to serverless applications. Altogether, Amazon S3 is helping people around the world securely store and extract value from their data.

Celebrate Amazon S3's 17th birthday at AWS Pi Day 2023 | AWS News Blog

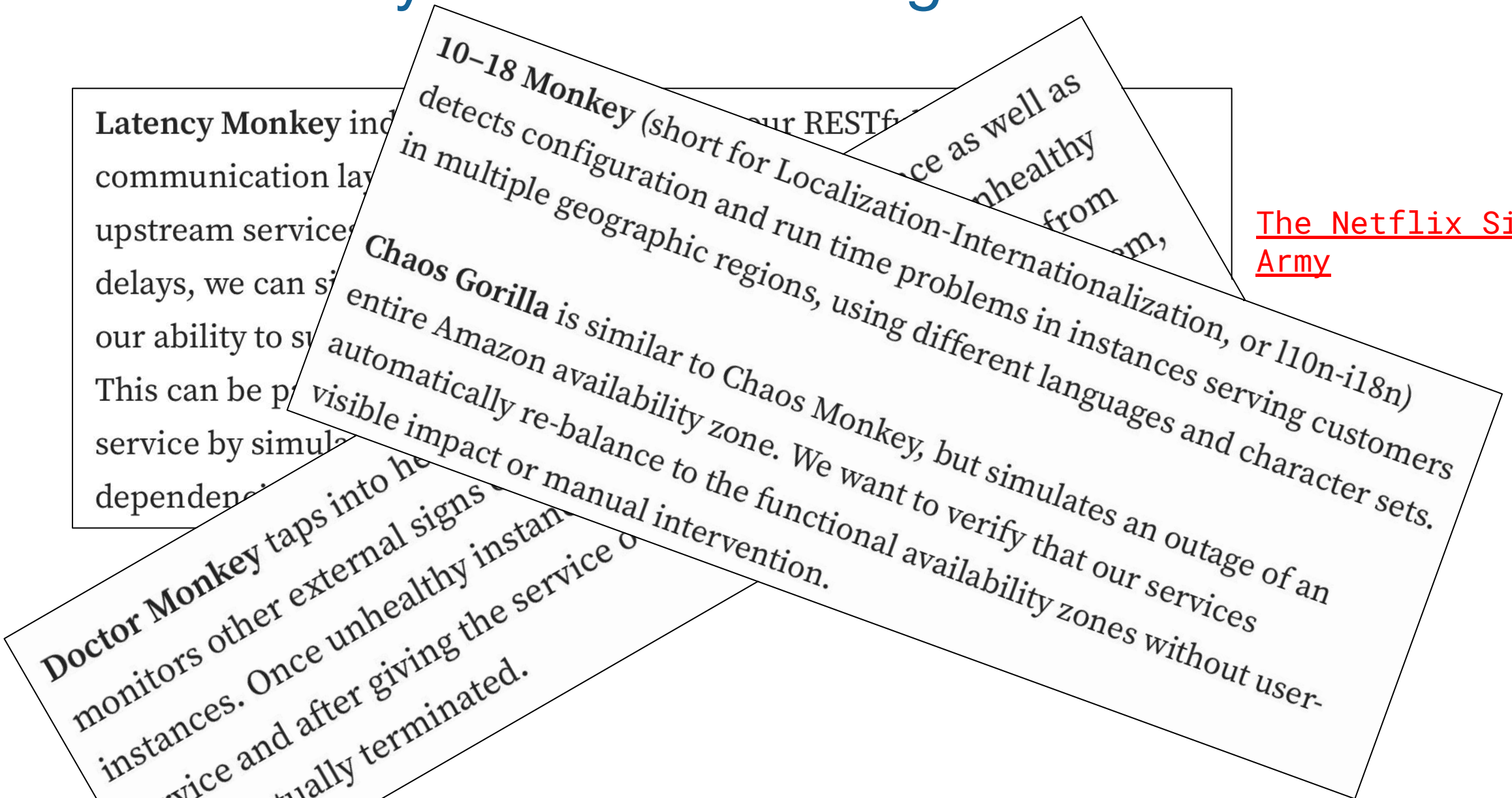
Case Study 3: Durability & Throughput at Scale

On this day 17 years ago, we launched a very simple object storage service. It allowed developers to create, list, and delete private storage spaces (known as buckets), upload and download files, and manage their access permissions. The service was available only through a REST and SOAP API. It was designed to provide highly durable data storage with 99.999999999 percent data durability (that's 11 nines!).

Amazon S3's designed for durability is a function of storage device failure rates and the rate at which S3 can detect failure, and then re-replicate data on those devices. S3 has end-to-end integrity checking on every object upload and verifies that all data is correctly and redundantly stored across multiple storage devices before it considers your upload to be successful. Once your data is stored in S3, S3 continuously monitors data durability over time with periodic integrity checks of all data at rest. S3 also actively monitors the redundancy of your data to help verify that your objects are able to tolerate the concurrent failure of multiple storage devices.

Celebrate Amazon S3's 17th birthday at AWS Pi Day 2023 | AWS News Blog

Case Study 4: Netflix never goes down!



The Netflix Simian Army

Today's class

- Course logistics
- Why performance?
- What is performance? Really
- Case studies
- **Terminology**
- Measurement
  - Observability
    - Counters
    - Profiling
    - Tracing
  - Experimentation

Terminology

Terminology

- **Basic metrics**
- Utilization
  - Time based
  - Capacity based
- Saturation
- Queuing System
- Instrumentation
  - Static Instrumentation
  - Dynamic Instrumentation

Basic Metrics

we will use these metrics to motivate other important concepts

- **Throughput:** Operations or data volume per unit time
- **Scalability:** A profile/spec of how the system behaves under varying load
- **Latency:** Operation time, as an average or percentile

Terminology

- Basic metrics
- **Utilization**
  - Time based
  - Capacity based
- Saturation
- Queuing System
- Instrumentation
  - Static Instrumentation
  - Dynamic Instrumentation

Utilization

Utilization is defined for resources, like memory, CPU or IO devices, and depicts **how busy a resource is**. It can be defined in 2 ways

1. **Time based**

2. **Capacity based**

Time-based Utilization

Time based

> Utilization **over a time period T** is defined as **B/T**, where B is the time period for which the resource was busy

This is readily available (using tools like `vmstat`, and `iostat`), and can be calculated using sampling.

Time-based Utilization is Misleading!

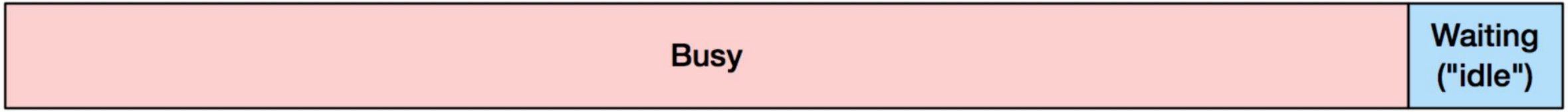
Time based utilization can be misleading.

For example, a **disk that is 100% busy may also be able to accept and process more work**, for example, by buffering writes in the on-disk cache to be completed later.

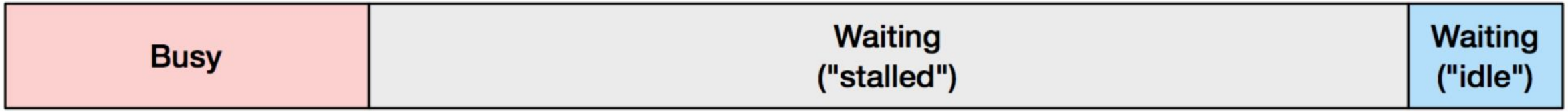
Storage arrays frequently run at 100%utilization because some disk is busy 100% of the time

CPU Utilization Example

What you may think 90% CPU utilization means:



What it might really mean:



CPU Utilization is Wrong

(https://www.brendangregg.com/blog/2017-05-09/cpu-utilization-is-wrong.html)

Capacity-based Utilization

Capacity based

A system or component (such as a disk drive) is **able to deliver a certain amount of throughput**.

At any level of performance, the system or component is working at some proportion of its capacity. That **proportion is called the utilization**.

Why even bother with Utilization?

Utilization is Virtually Useless as a Metric!

Adrian Cockcroft

Netflix Inc.

Utilization is Virtually Useless as a Metric!

Why do we care then?

Because it's **easily available**, and **using it with other indicators** is helpful for a great analysis of systems performance

Thinking Methodically About Performance

Terminology

- Basic metrics
- Utilization
  - Time based
  - Capacity based
- **Saturation**
- Queuing System
- Instrumentation
  - Static Instrumentation
  - Dynamic Instrumentation

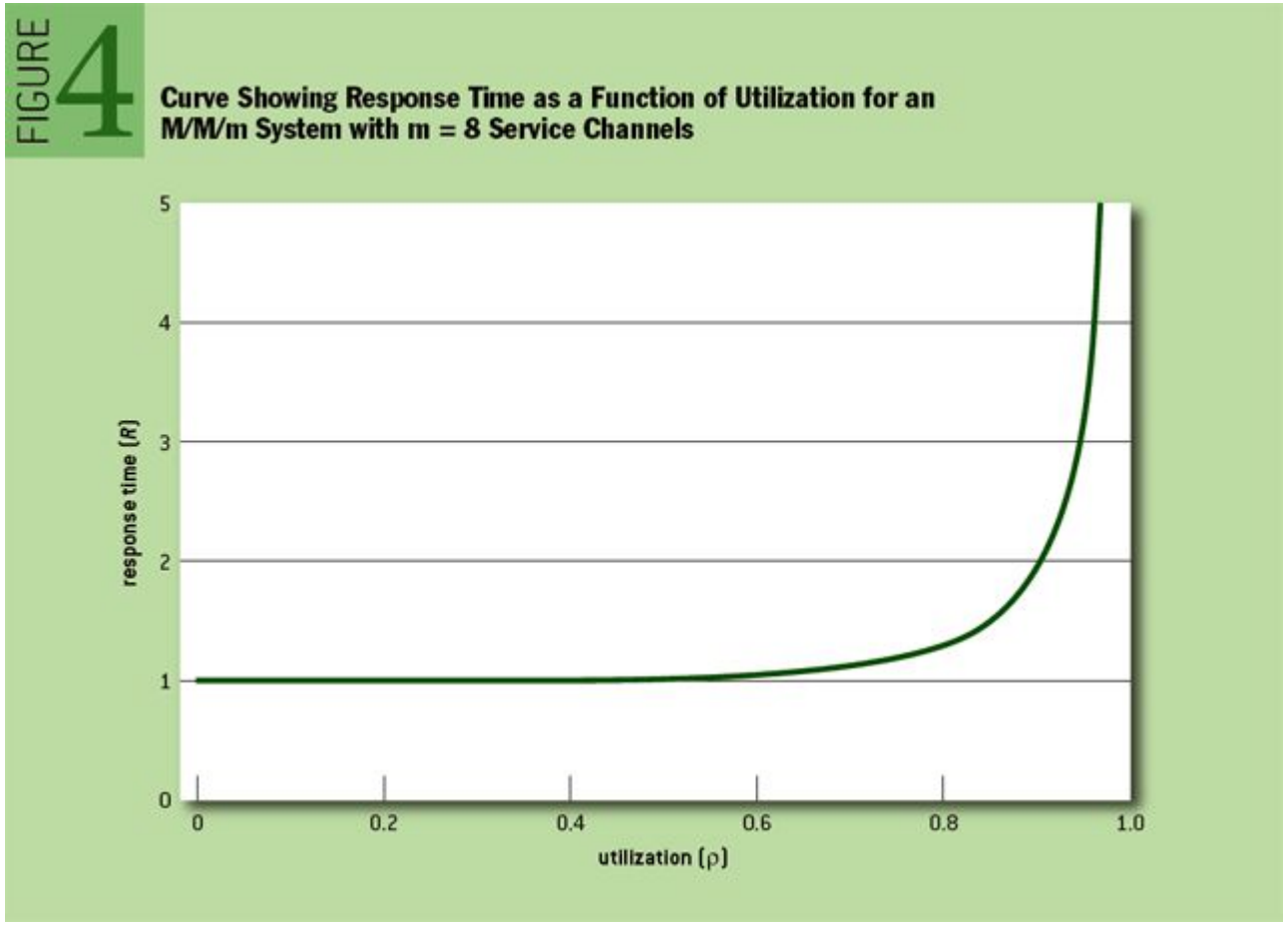
Saturation

"…The **degree to which more work** is requested of a resource **than it can process** is saturation. **Saturation begins to occur at 100% utilization (capacity-based)**, as extra work cannot be processed and begins to queue…"

Systems Performance 2nd Ed., Brendan Gregg

Effect of Saturation: Bottlenecks!

Saturation is always problematic, and often a strong signal for possible bottleneck. It is **valuable to understand the behaviour of a system under saturation.**



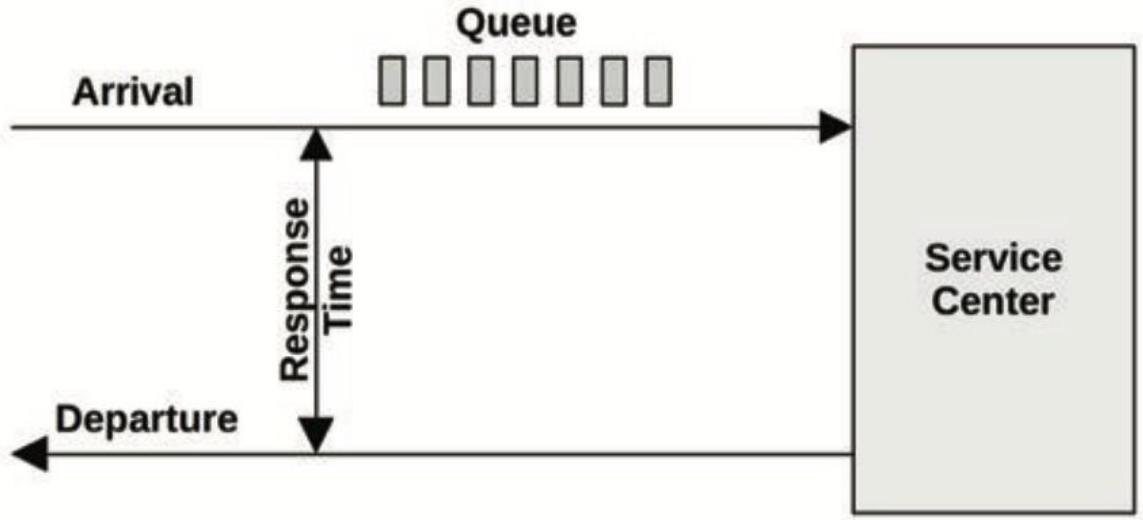
[Thinking Clearly about Performance - ACM Queue]

Terminology

- Basic metrics
- Utilization
  - Time based
  - Capacity based
- Saturation
- **Queuing System**
- Instrumentation
  - Static Instrumentation
  - Dynamic Instrumentation

Queuing Systems

…resources can be **modeled as a queueing system** so that their performance under different situations can be predicted based on the model. Disks are commonly modeled as a queueing system



Systems Performance, Brendan Gregg, 2nd ed.

Terminology

- Basic metrics
- Utilization
  - Time based
  - Capacity based
- Saturation
- Queuing System
- **Instrumentation**
  - Static Instrumentation
  - Dynamic Instrumentation

Instrumentation

Instrumentation is the **methodology of adding code** to a process to **gain valuable information**, like counters in the kernel, or logging start and end times of an operation.

This can be done statically (compile-time), or dynamically (run-time)

Static Instrumentation

…**Static instrumentation describes hard-coded software** instrumentation points added to the source code. There are **hundreds of these points in the Linux kernel** that instrument disk I/O, scheduler events, system calls, and more.

Systems Performance 2nd Ed., Brendan Gregg

Static Instrumentation

…The Linux technology for **kernel static instrumentation is called *tracepoints***. There is also a static instrumentation **technology for user-space** software called ***user statically defined tracing*** **(USDT).**

Systems Performance 2nd Ed., Brendan Gregg

Dynamic Instrumentation

Dynamic instrumentation **creates instrumentation points after the software is running**, by modifying in-memory instructions to insert instrumentation routines. This is **similar to how debuggers can insert a breakpoint** on any function in running software.

Systems Performance 2nd Ed., Brendan Gregg

Importance of Dynamic Instrumentation

… analyzing kernel internals can be like venturing into a **dark room, with candles (system counters) placed where the kernel engineers thought** they were needed. **Dynamic instrumentation is like having a flashlight** that you can point anywhere.

Systems Performance 2nd Ed., Brendan Gregg

DynamoRIO

Home

DynamoRIO is a runtime code manipulation system that supports code transformations on any part of a program, while it executes. DynamoRIO exports an interface for building dynamic tools for a wide variety of uses: program analysis and understanding, profiling, instrumentation, optimization, translation, etc. Unlike many dynamic tool systems, DynamoRIO is not limited to insertion of callouts/trampolines and allows arbitrary modifications to application instructions via a powerful IA-32/AMD64/ARM/AArch64 instruction manipulation library. DynamoRIO provides efficient, transparent, and comprehensive manipulation of unmodified applications running on stock operating systems (Windows, Linux, or Android, with experimental Mac support) and commodity IA-32, AMD64, ARM, and AArch64 hardware.

Existing DynamoRIO-based tools

DynamoRIO is the basis for some well-known external tools:

- The Arm Instruction Emulator (ArmIE)
- WinAFL, the Windows fuzzing tool, as an instrumentation and code coverage engine
- The fine-grained profiler for ARM DrCCTProf
- The portable and efficient framework for fine-grained value profilers VClinic
- The sampling-based sanitizer framework GWPSan

DynamoRIO

Instrumentation Example

…For this experiment developer needs to **profile all memory access** made by his program and all calls to do_page_fault kernel function. To **instrument memory accesses** he uses well-known tool – Dyninst, to **instrument do_page_fault** Systemtap is used.

Dynamic Binary Instrumentation Framework for CE Devices

Today's class

- Course logistics
- Why performance?
- What is performance? Really
- Case studies
- Terminology
- **Measurement**
  - Observability
    - Counters
    - Profiling
    - Tracing
  - Experimentation

Measurement

Observability

Observability refers to understanding a system **through observation**, and classifies the tools that accomplish this. This includes tools that use **counters, profiling, and tracing**.

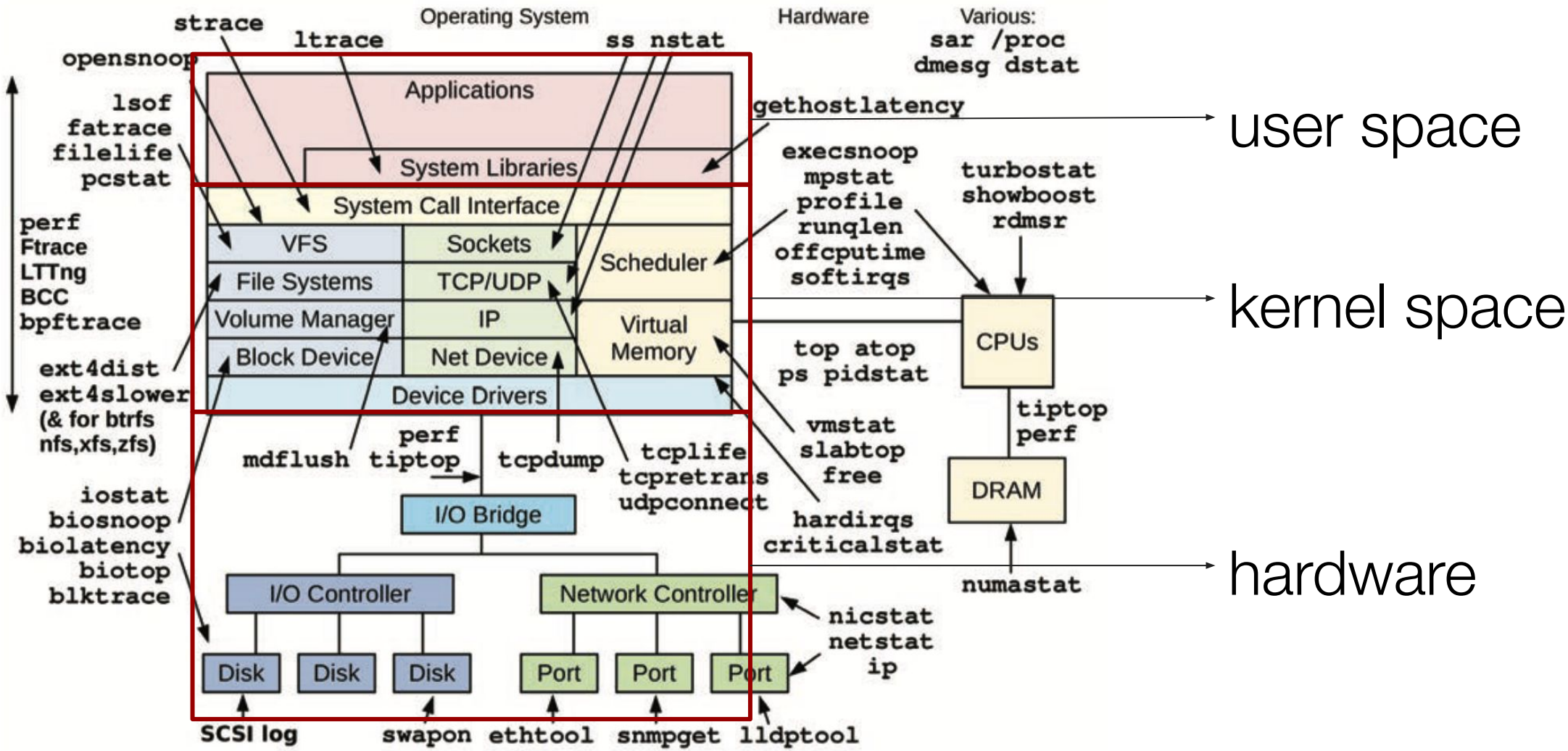
It **does not include benchmark tools**, which modify the state of the system by performing a workload *experiment*.

Observability

1. **What (how deep) are you measuring?**

2. How are you measuring?
   - Counters
   - Profiling
   - Tracing

How deep are you measuring?

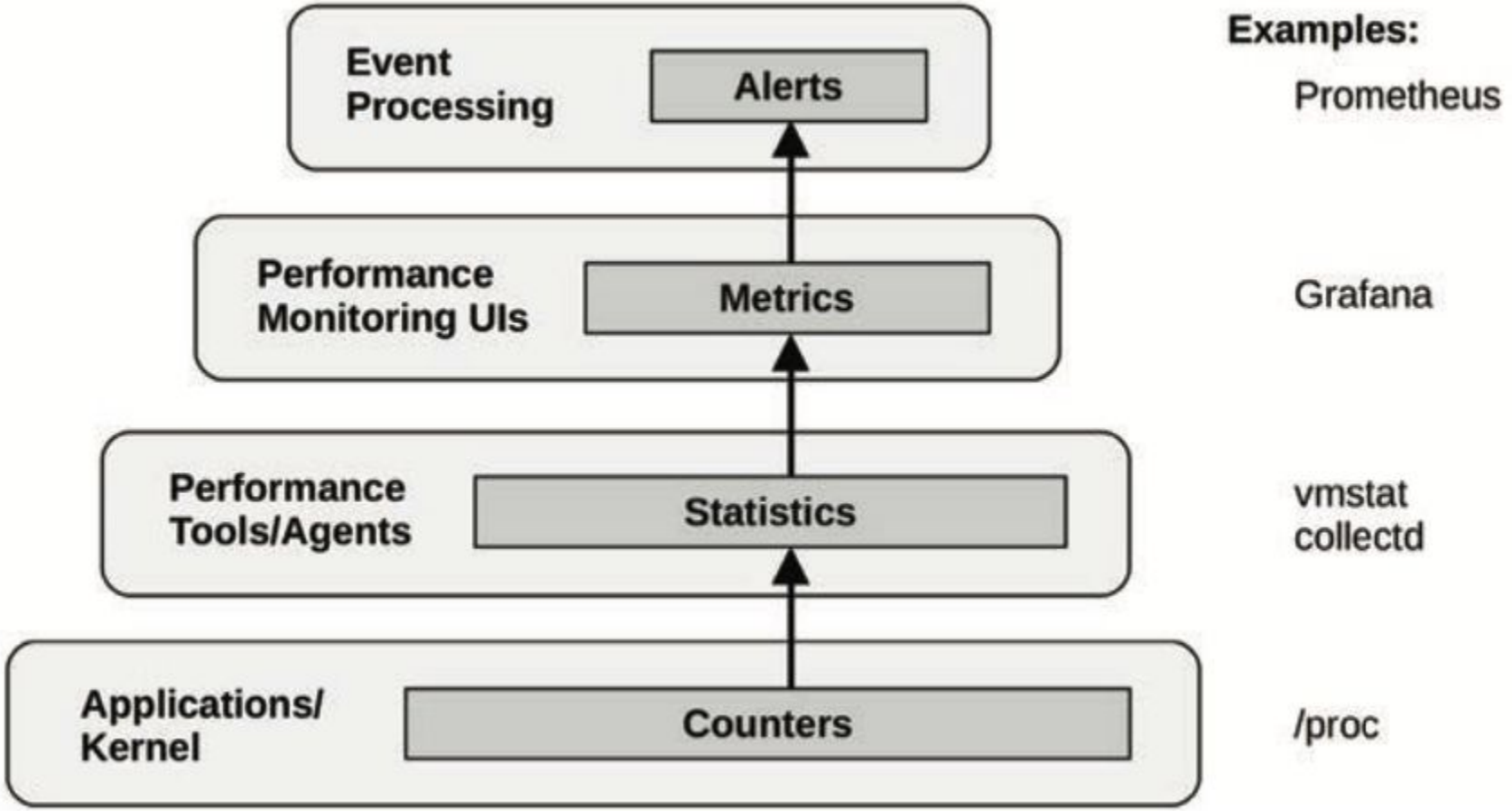


Observability

1. What (how deep) are you measuring?

2. How are you measuring?
   - **Counters**
   - Profiling
   - Tracing

Counters (remember Static Instrumentation?)



Systems Performance 2nd Ed., Brendan Gregg

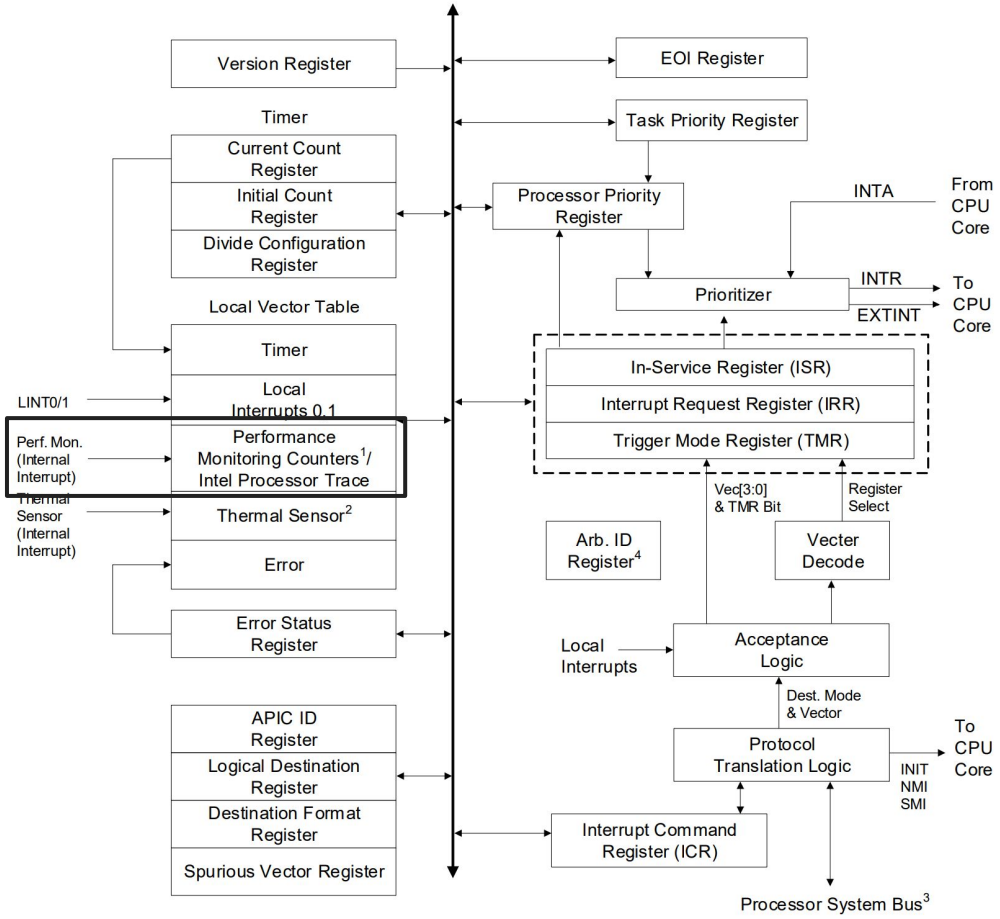
Counters in Intel Processors!

2.8.6 Reading Performance-Monitoring and Time-Stamp Counters

The RDPMC (read performance-monitoring counter) and RDTSC (read time-stamp counter) instructions allow application programs to read the processor's performance-monitoring and time-stamp counters, respectively. Processors based on Intel NetBurst® microarchitecture have eighteen 40-bit performance-monitoring counters; P6 family processors have two 40-bit counters. Intel Atom® processors and most of the processors based on the Intel Core microarchitecture support two types of performance monitoring counters: programmable performance counters similar to those available in the P6 family, and three fixed-function performance monitoring counters. Details

https://cdrdv2.intel.com/v1/dl/getContent/671447

Counters in Intel Processors!



"…APIC **registers are mapped into the 4-KByte APIC register space**. Registers are 32 bits, 64 bits, or 256 bits in width; all are aligned on 128-bit boundaries. All 32-bit registers should be accessed using 128-bit aligned 32-bit loads or stores…"

https://cdrdv2.intel.com/v1/dl/getContent/671447

Getting the real CPU utilization

Time based CPU utilization is not the answer, then what is?

By using **Performance Monitoring Counters (PMCs)**: hardware counters that can be read using **Linux perf**, and other tools.

Getting the real CPU utilization: IPC

```
# perf stat -a -- sleep 10

 Performance counter stats for 'system wide':

    641398.723351      task-clock (msec)         #   64.116 CPUs utilized           (100.00%)
         379,651      context-switches          #    0.592 K/sec                   (100.00%)
          51,546      cpu-migrations            #    0.080 K/sec                   (100.00%)
      13,423,039      page-faults               #    0.021 M/sec
 1,433,972,173,374    cycles                    #    2.236 GHz                     (75.02%)
   <not supported>    stalled-cycles-frontend
   <not supported>    stalled-cycles-backend
 1,118,336,816,068    instructions              #    0.78  insns per cycle         (75.01%)
   249,644,142,804    branches                  #  389.218 M/sec                   (75.01%)
     7,791,449,769    branch-misses             #    3.12% of all branches         (75.01%)

      10.003794539 seconds time elapsed
```

System-wide counts

- **vmstat(8)**: Virtual and physical memory statistics, system-wide
- **mpstat(1)**: Per-CPU usage
- **iostat(1)**: Per-disk I/O usage, reported from the block device interface
- **nstat(8)**: TCP/IP stack statistics
- **sar(1)**: Various statistics; can also archive them for historical reporting

Observability

1. What (how deep) are you measuring?

2. How are you measuring?
   - Counters
   - **Profiling**
   - Tracing

Profiling

In systems performance, the term ***profiling*** **usually refers to the use of tools that perform sampling: taking a subset (a sample) of measurements** to paint a coarse picture of the target.

CPUs are a common profiling target. The commonly used method to profile CPUs involves taking timed-interval samples of the on-CPU code paths.

Systems Performance 2nd Ed., Brendan Gregg

Profiling

…samples are **usually collected at a fixed rate**, such as 100 Hz (cycles per second) across all CPUs, and for a short duration such as one minute. Profiling tools, or *profilers*, often use 99 Hz instead of 100 Hz to avoid sampling in lockstep with target activity

Systems Performance 2nd Ed., Brendan Gregg

CPU Profiling, `perf top`

```
Samples: 8K of event 'cycles', 2000 Hz, Event count (approx.): 4579432780 lost: 0/0 drop: 0/0
Overhead  Shared Object       Symbol
   2.20%  [kernel]            [k] do_syscall_64
   2.17%  [kernel]            [k] module_get_kallsym
   1.49%  [kernel]            [k] copy_user_enhanced_fast_string
   1.37%  libpthread-2.29.so  [.] pthread_mutex_lock 1.31% [unknown] [.] 0000000000000000 1.07% [kernel] [k]
psi_task_change 1.04% [kernel] [k] switch_mm_irqs_off 0.94% [kernel] [k] fget
   0.74%  [kernel]            [k] entry_SYSCALL_64
   0.69%  [kernel]            [k] syscall_return_via_sysret
   0.69%  libxul.so           [.] 0x000000000113f9b0
   0.67%  [kernel]            [k] kallsyms_expand_symbol.constprop.0
   0.65%  firefox             [.] moz_xmalloc
   0.65%  libpthread-2.29.so  [.] __pthread_mutex_unlock_usercnt
…
```

https://docs.redhat.com/en/documentation/red_hat_enterprise_linux/8/html/monitoring_and_managing_system_status_and_performance/profiling-cpu-usage-in-real-time-with-top_monitoring-and-managing-system-status-and-performance#profiling-cpu-usage-in-real-time-with-top_monitoring-and-managing-system-status-and-performance

System-wide Profilers

- **perf(1):** The standard Linux profiler, which includes profiling subcommands.
- **profile(8):** A BPF-based CPU profiler from the BCC repository (covered in Chapter 15, BPF) that frequency counts stack traces in kernel context.
- **Intel VTune Amplifier XE:** Linux and Windows profiling, with a graphical interface including source browsing.

Today's class: We covered!

- **Course logistics**
- **Why performance?**
- **What is performance? Really**
- **Case studies**
- **Terminology**
- **Measurement**
  - **Observability**
    - **Counters**
    - **Profiling**
    - Tracing
  - Experimentation