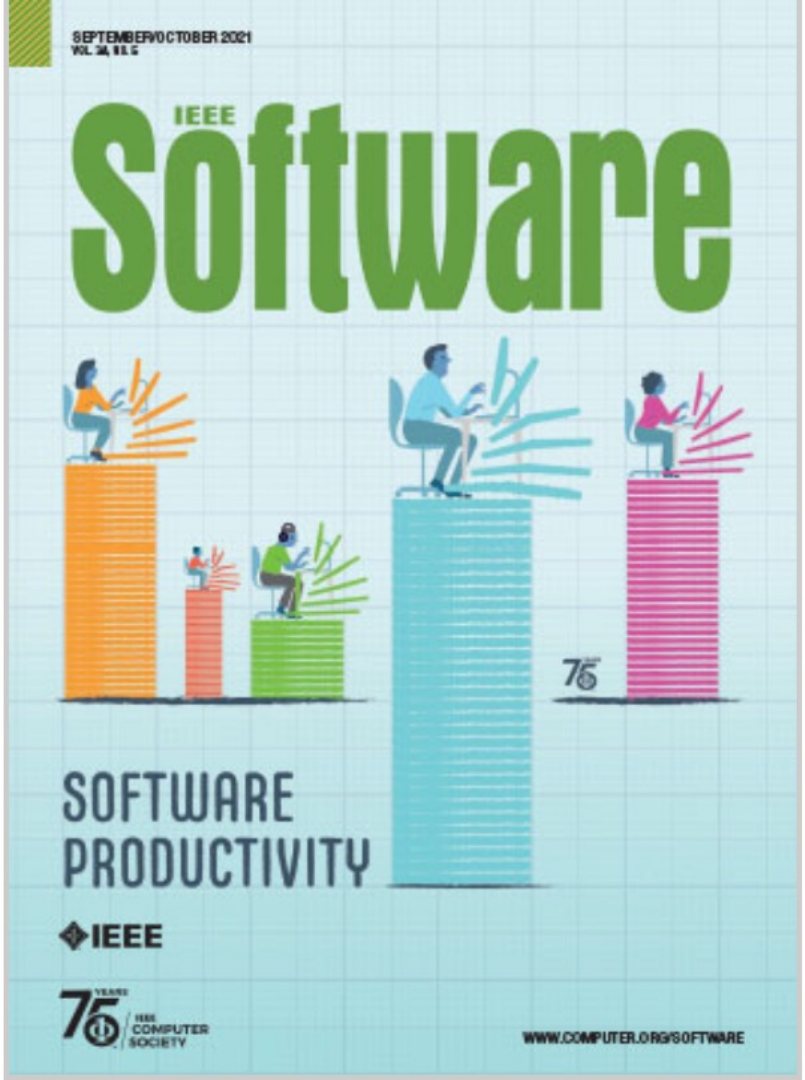
Predicting Performance of Data Centers at Scale

Dr. Lizy Kurian John
Laboratory for Computer Architecture (LCA)
University of Texas at Austin
October 2021



Productivity

Two other terms you hear a lot

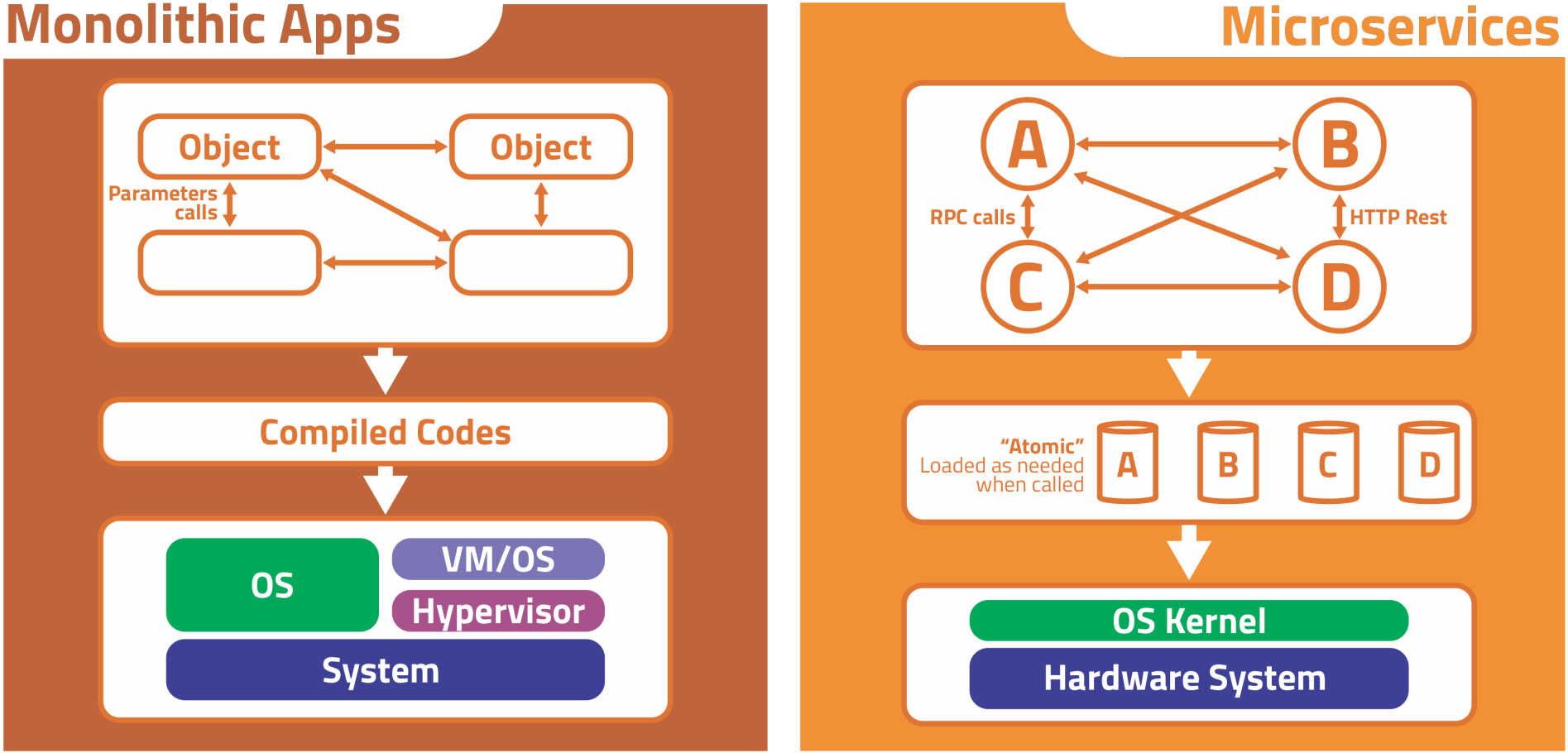
Microservices

Serverless

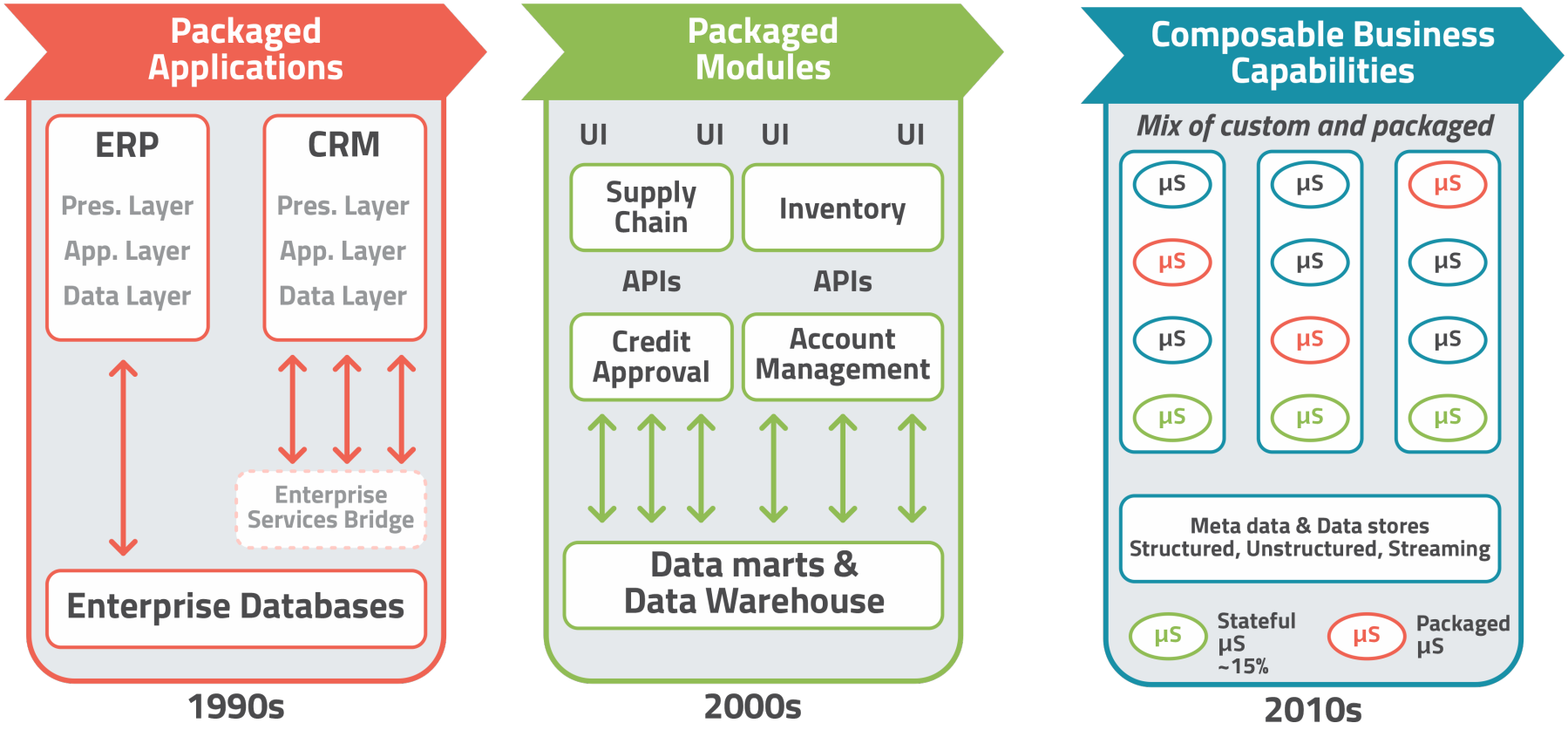
Microservices

- Microservices originated from the evolution of software architectures
- Reduce the complexity of monolithic systems and service-oriented architectures
- Small and autonomous services with a single and clearly defined purpose
- Enable vertically decomposition of applications into a subset of business-driven independent services
- Connected with lightweight communication protocols—usually HTTP application programming interfaces (APIs) or RPCs

Monolithic Apps versus Microservices



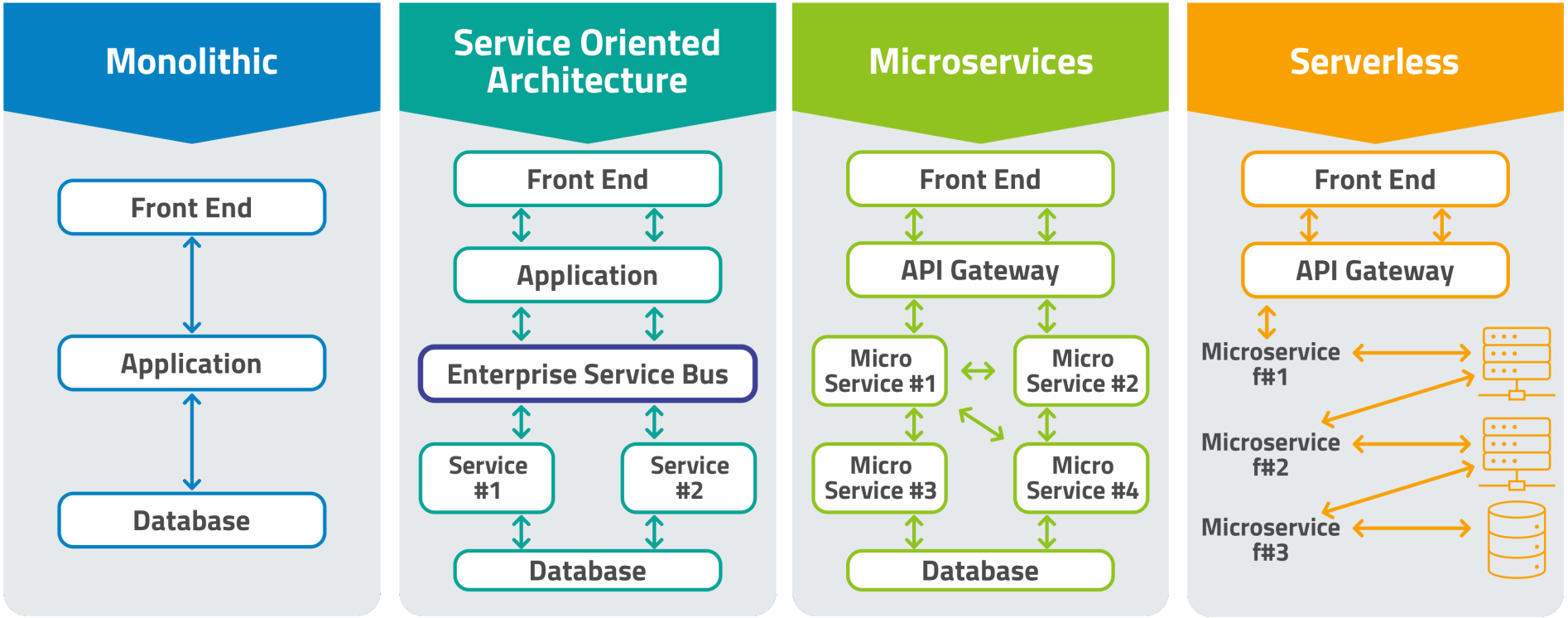
Application Evolution



Serverless

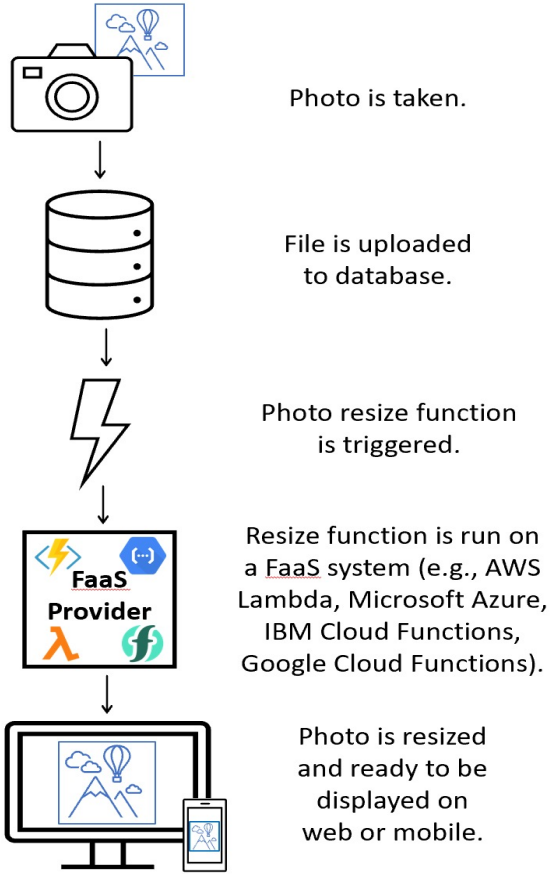
- A platform to efficiently develop and deploy applications to the market without having to manage any underlying infrastructure
- Developers worry about operating their systems instead of operating their servers
- Consider applications as workflows, distributed logic, and externally managed data stores
- Cloud provider dynamically allocates the servers
- Code is executed in event-triggered containers
- Different technologies, back end as a service (BaaS) and functions as a service (FaaS)
- Google Firebase, a fully managed database that can be used directly from an application, is an example of BaaS
- FaaS Examples - JSON, MAX, array copy

Evolution of Software Architectures



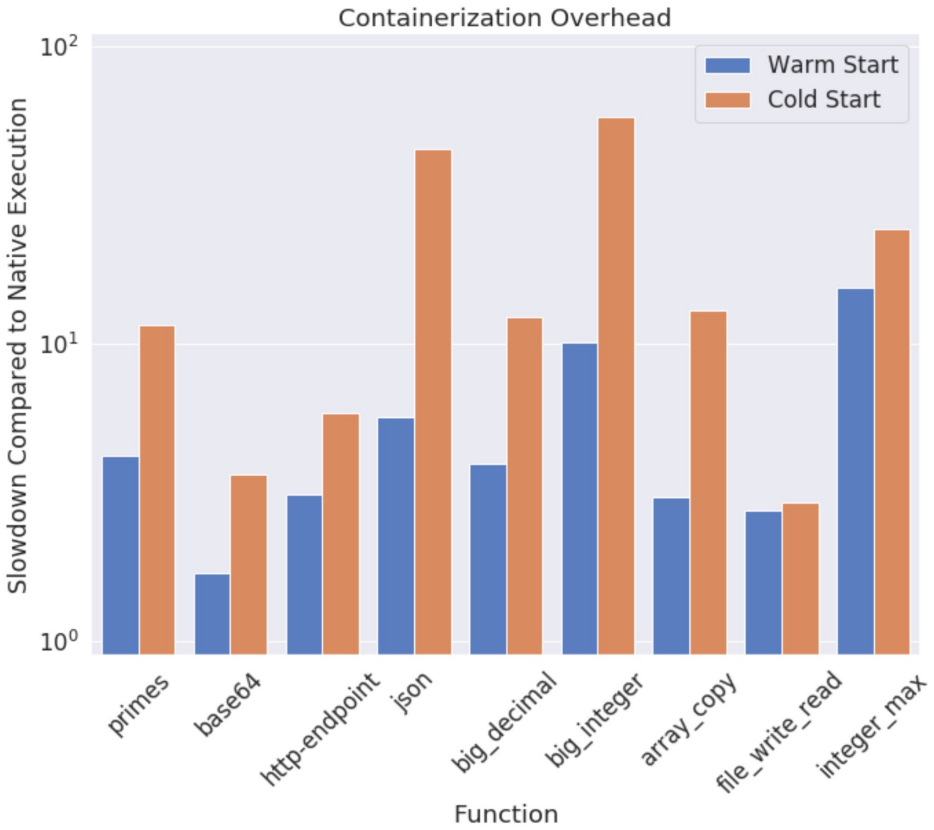
Modern Web Apps

- Social Media
- Online Messaging
- Web Search
- Video Streaming
- Billions of Users
- Require 100s of 1000s of servers
- Amazon, Netflix, LinkedIn, Facebook, etc. have adopted microservice architectures
- Serverless enables fine-grained, short-lived cloud execution
- Improved Web Service Development
- Improved Scalability

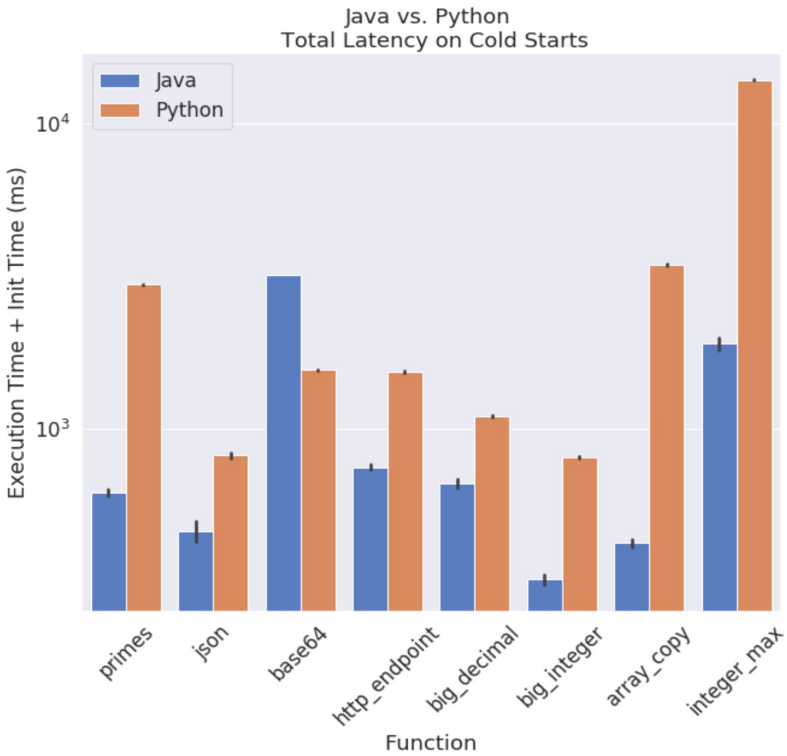


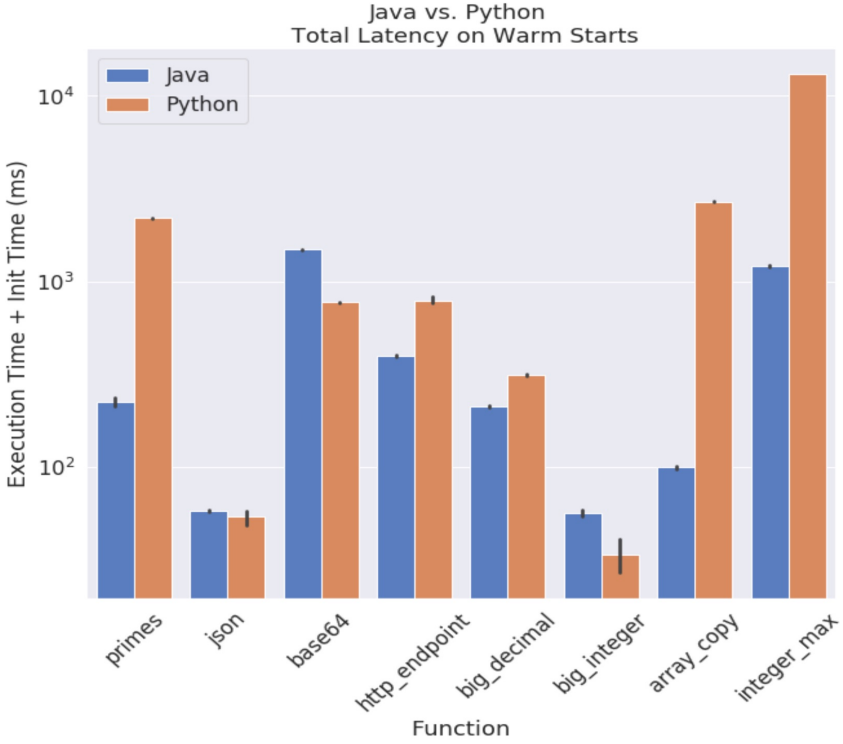
Overheads in FaaS

- Containerization Overheads
- Cold Start Overheads

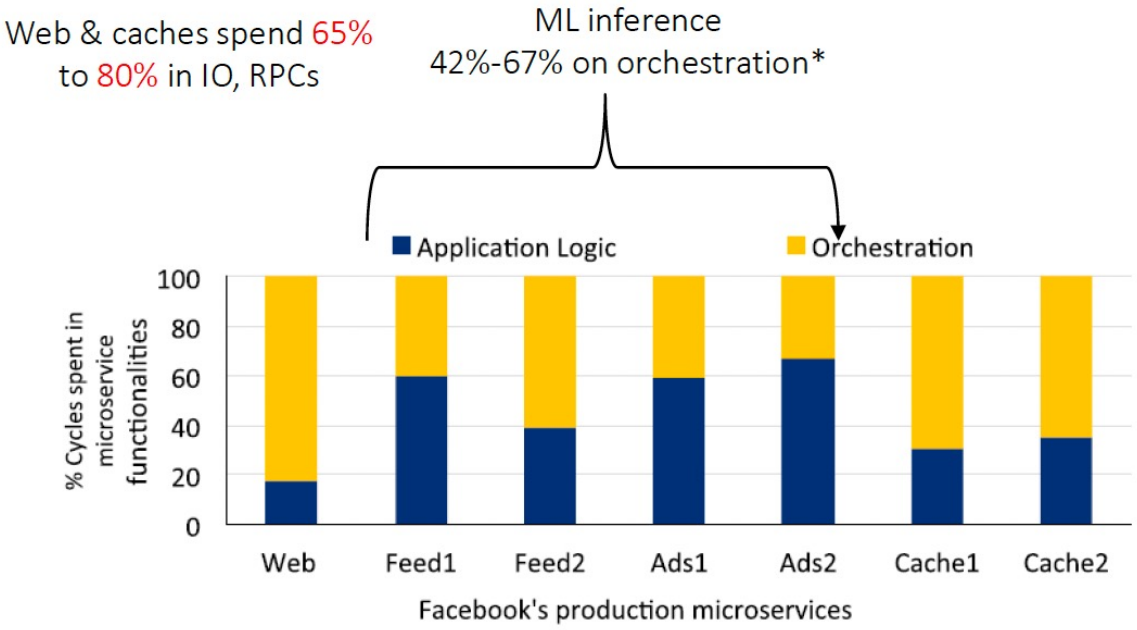


Language Dependencies



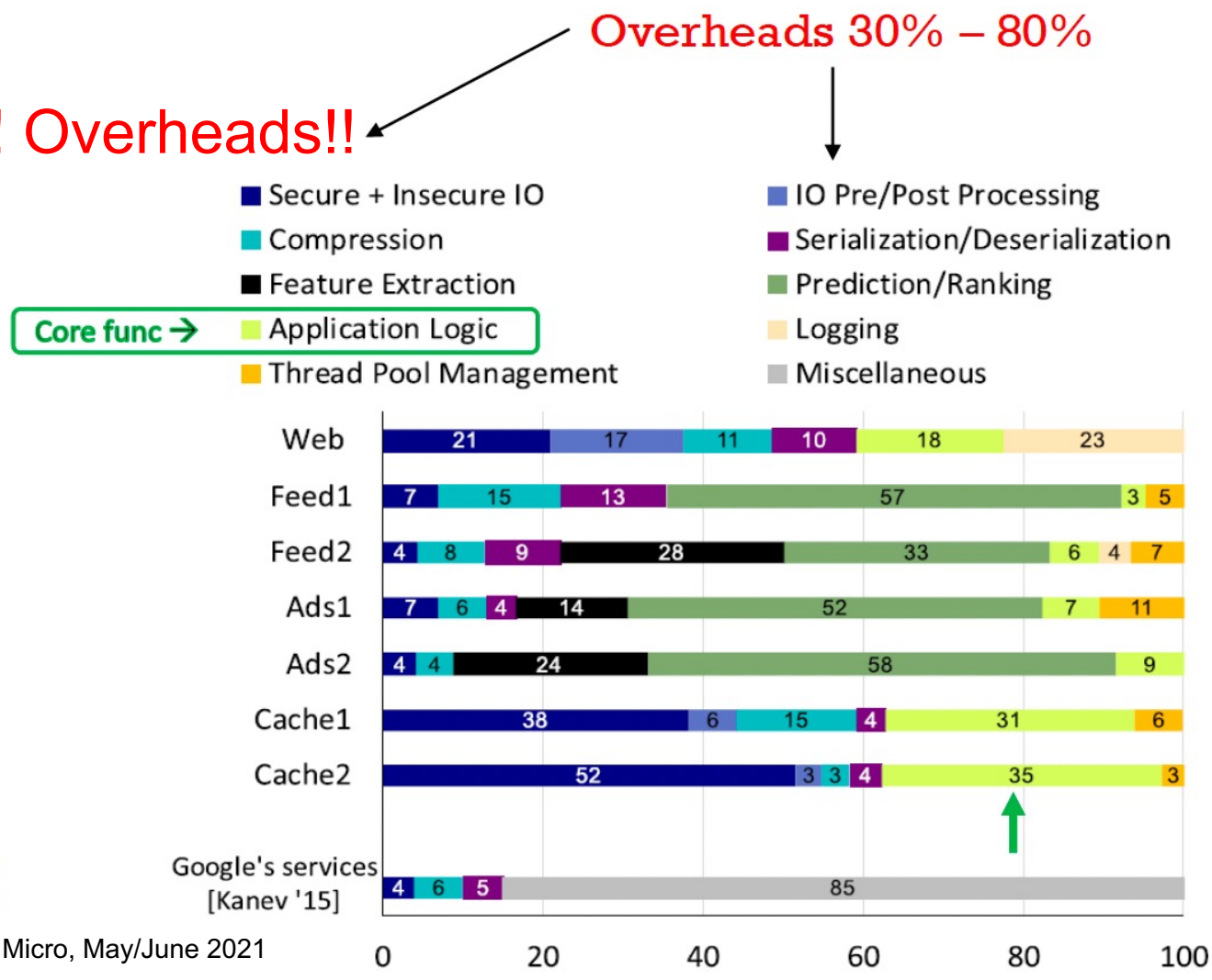


Performance Inefficiencies that Creep In Microservices



Sriraman and Dhanotia, IEEE Micro, May/June 2021

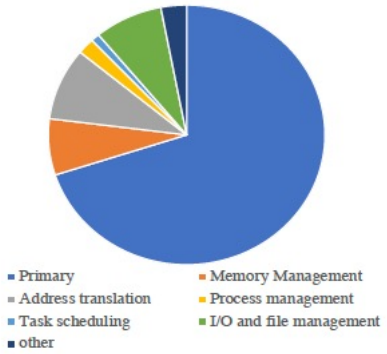
Overheads! Overheads!!



Sriraman and Dhanotia, IEEE Micro, May/June 2021

Every convenience has a price attached to it

- Loops
- Structured programming
- y(i) = y(i) + a* x (i)



Naïve Dumbo Code
If we had no loops
Eg: Unroll fully (1000 times)

y(1) = y(1) + a* x(1)
y(2) = y(2) + a* x(2)
y(3) = y(3) + a* x(3)
y(4) = y(4) + a* x(4)
y(5) = y(5) + a* x(5)
.
.

y(1000) = y(1000) + a*x(1000)

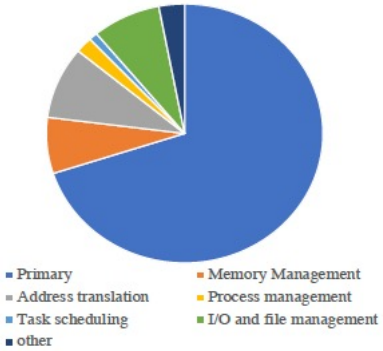
```
$32: lw   $24,8004($sp);load current i to R24     ;(OMA)
                        ;Overhead memory access
     mul  $25,$24,4     ;i times data size to R25 ;(ADDR)
                        ;Address arithmetic
     addu $8,$sp,8008   ;Array start address      ;(ADDR)
                        ;Address arithmetic
     addu $9,$25,$8     ;Base address in R9       ;(ADDR)
                        ;Address arithmetic
     lw   $10,-4008($9);operand x(i) to R10       ;(ACC)
                        ;Data access
     mul  $11,$15,$10   ;a*x(i) - R15 contains a  ;(EXE)
                        ;Execute
     lw   $12,-8008($9);operand y to R12          ;(ACC)
                        ;Data access
     addu $13,$12,$11   ;y(i) + a*x(i)            ;(EXE)
                        ;Execute
     sw   $13,-8008($9);save new y                ;(ACC)
                        ;Data access
     lw   $14,8004($sp);load current index        ;(OMA)
                        ;Overhead memory Access
     addu $24,$14,1     ;increment index          ;(ADDR)
                        ;Address arithmetic
     sw   $24,8004($sp);store new index back      ;(OMA)
                        ;Overhead memory Access
     blt  $24,1000,$32  ;inner loop ends          ;(BR)
                        ;Branch
```

Performance is lost at each step of the convenience

Java Virtual Machine

Scripting Languages

Interpretation/JITs

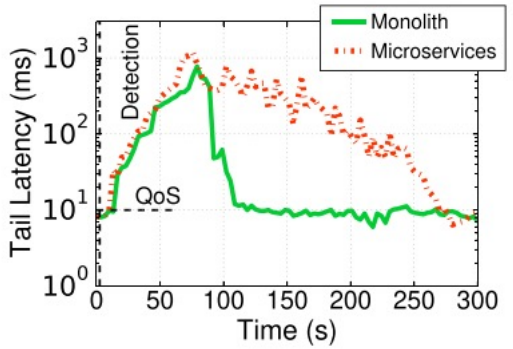


Taurus load results	Docker only "Native"	With Virtualization
Single Server (3 services)		
50 rps	34 msec	27
150 rps	104,176 (3,000x slower)	82,026 (3,000x slower)
Multiple (3) servers		
50 rps	25 ms	23
250 rps	49,667 (2,000x slower)	
300 rps		42,684 (1,800x slower)

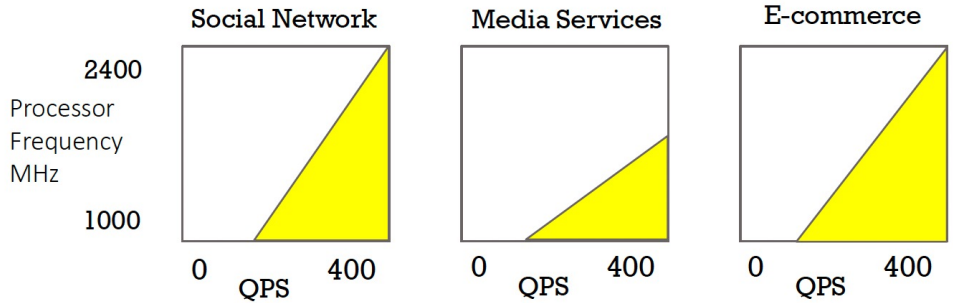
Master's Thesis by Vilhem Fridriksson, Oct 2018

Performance is lost at each step of the convenience

- Microservices take longer than monoliths to recover from QoS Violations



- More QoS violations if processor frequency reduced

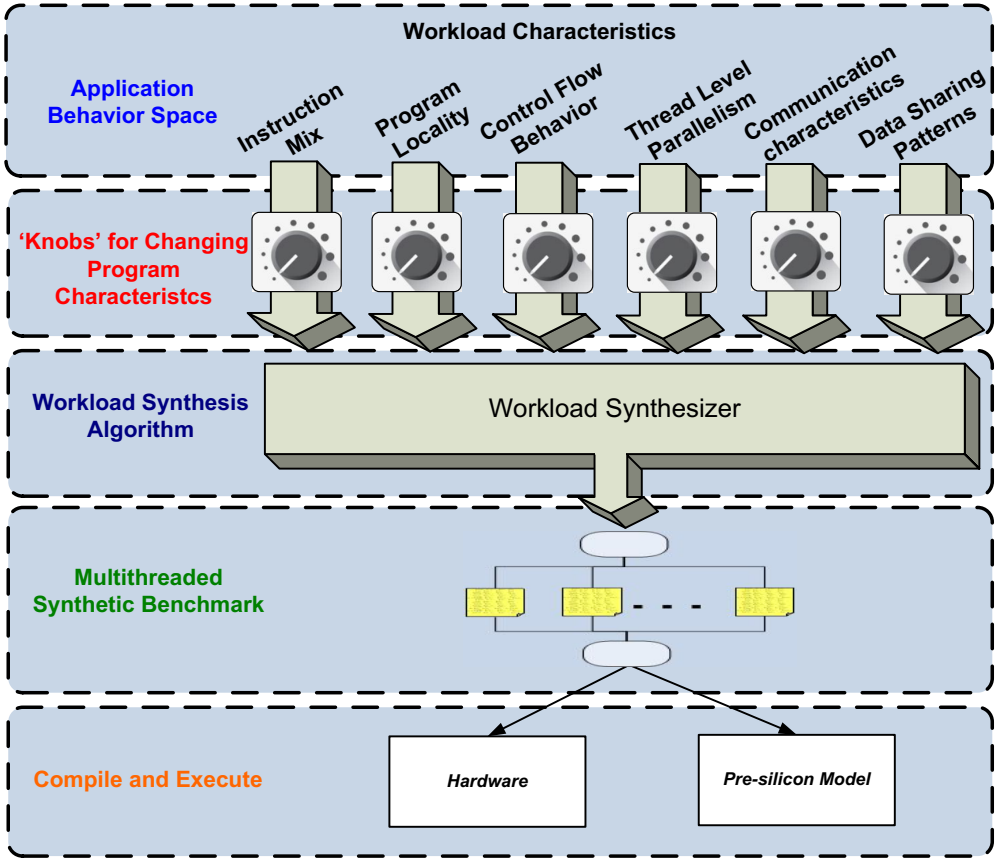


Source: Unveiling the hardware and software implications of microservices in cloud and edge systems, Cornell univ, IEEE Micro April 2020

Data Center Benchmarking

Original Workloads

Performance/Power Clones

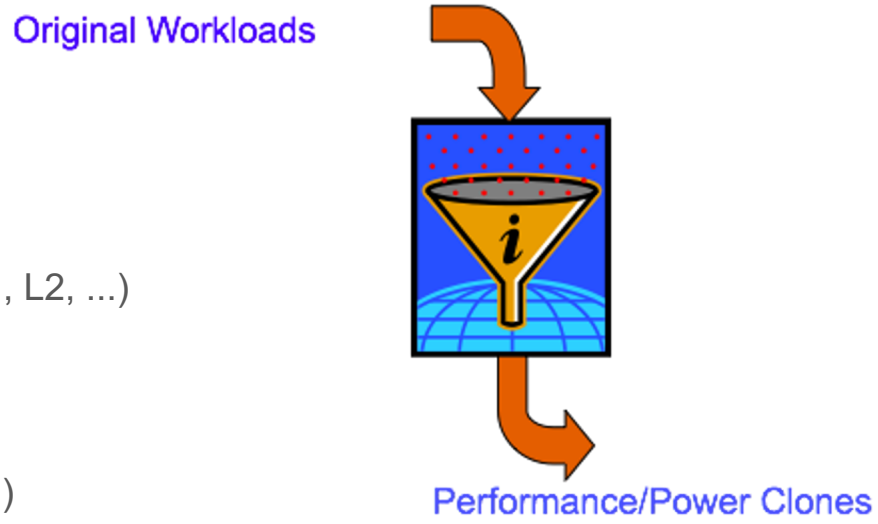


Portable Adaptable Data Center Benchmarks

- Motivation:
  - Existing benchmarks are not tunable, failing to reflect the varying workloads at Production Data Centers
  - Proprietary pieces in most production workloads
- Can you create a tunable benchmark using open-source resources?
- A tunable open source benchmark, covering a wide workload space?

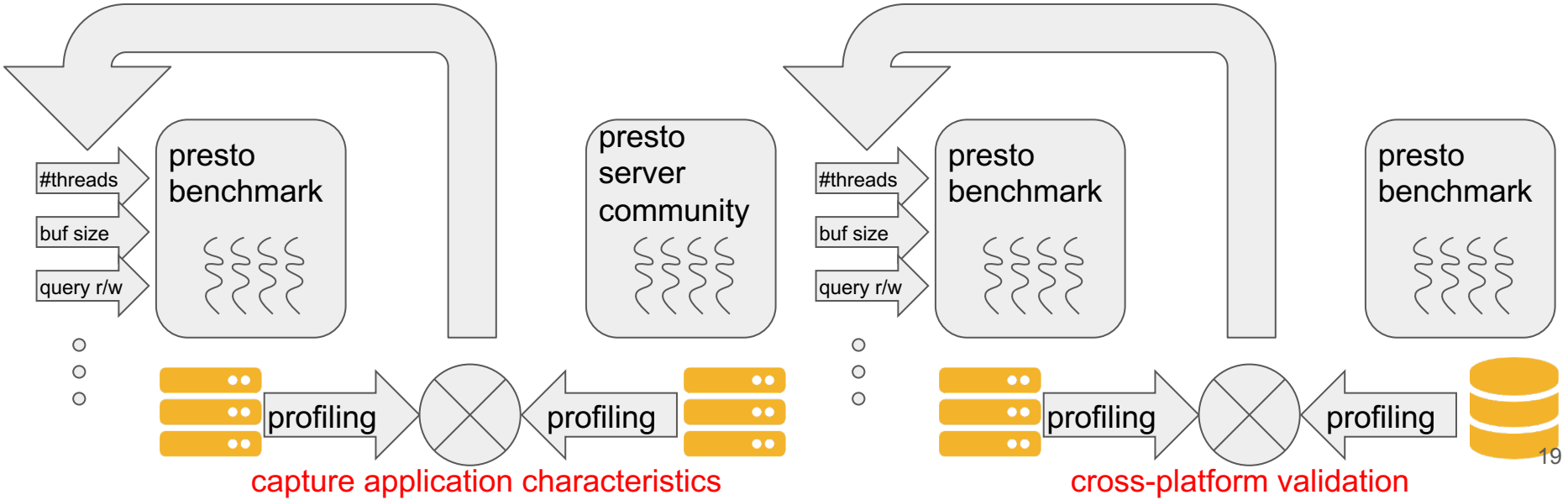
Data Center Proxy Workload Generation

- Clones for Data Center Production Workloads
- Use Open Source Set Ups
- Replicate Stats:
  - CPU Utilization
  - DRAM Bandwidth
  - Network bandwidth
- Microarchitectural Events:
  - Cache Misses (Instrn and Data) (L1, L2, ...)
  - TLB Misses (Level 1 and Level 2)
  - Branch Mispredictions
  - Retirement Rate
  - Pipeline Stalls (Front End, Backend)



Portable Benchmarks for Data Center Benchmarking

- Develop a portable benchmark
- Add performance knobs to make the benchmark configurable



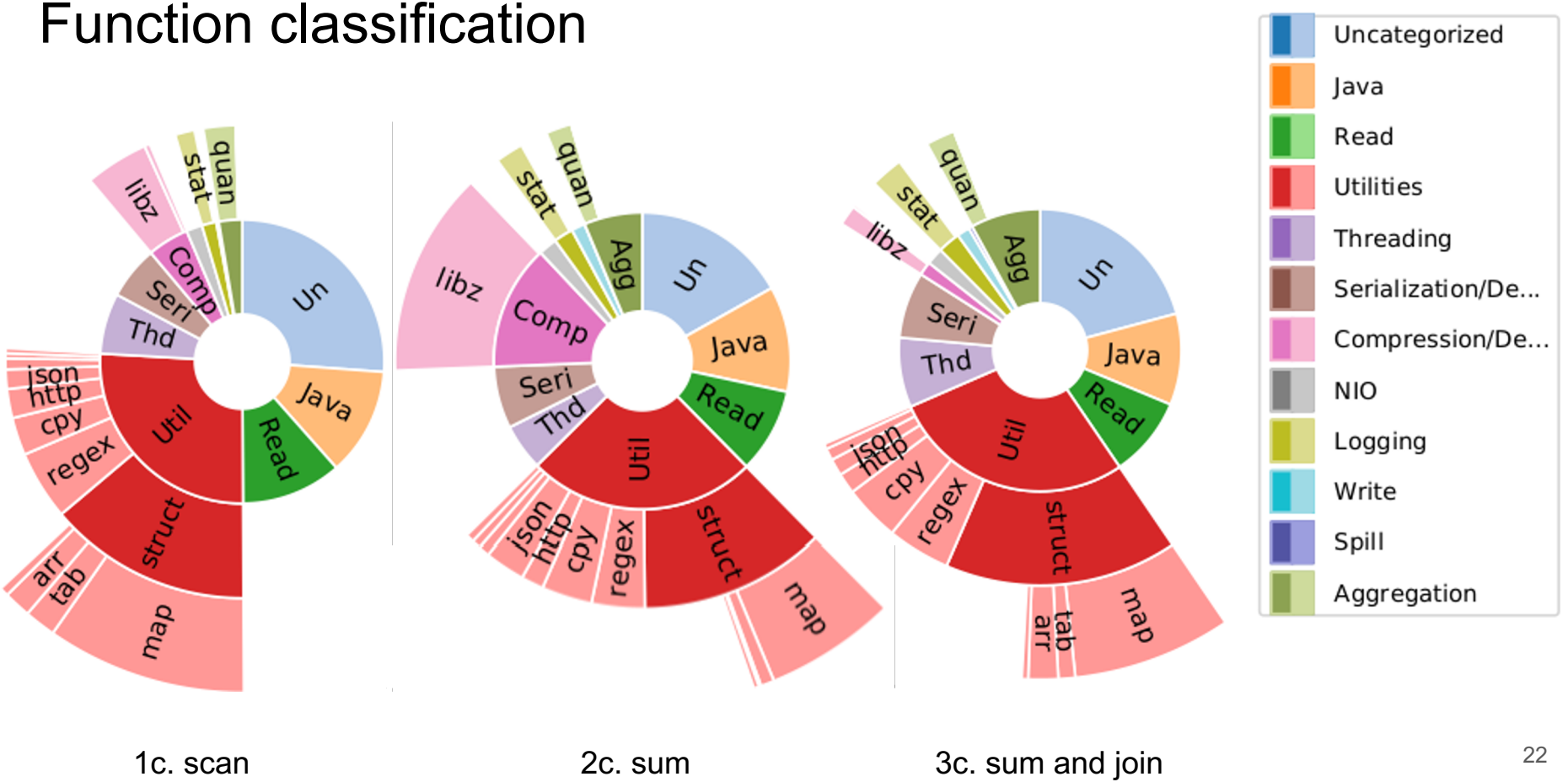
Tunable Benchmark Development Methodology

- Profiling: Profile query workloads
- Synthesis: Tune the datasets, query mix, and insert kernels to match the benchmark with production features on key PMU metrics, Topdown, function and operator profiles
- Cross-platform validation: Find two clusters/workloads having different behaviors, then find two sets of parameters of the benchmark that make them match the two instances respectively

Profiling and Topdown (text_tiny.1c query 32 threads 5 minutes)

app & sys stats		topdown		perf	
QPS	74.08	Retiring	26.17%	IPC	0.55
cpu_util	90.00%	Frontend.Lat	38.95%	L1I_MPKI	35.08
usr/sys%	80.75%/7.92%	Frontend.BW	11.40%	iTLB_MPKI	1.71
net_in	7028 KB/s	BadSpec	9.50%	Br_MPKI	9.15
net_out	4338 KB/s	Backend.Core	4.05%	L1D_MPKI	30.62
ctx_switch	81937 /s	Backend.L1	62.59%	dTLB_MPKI	3.16
pg_fault	220.5 /s	Backend.DRAM	0.00%	MemRdBW	4.64 GB/s
pg_swap	1900.01 /s	Backend.Store	1.89%	MemWrBW	2.52 GB/s

Function classification



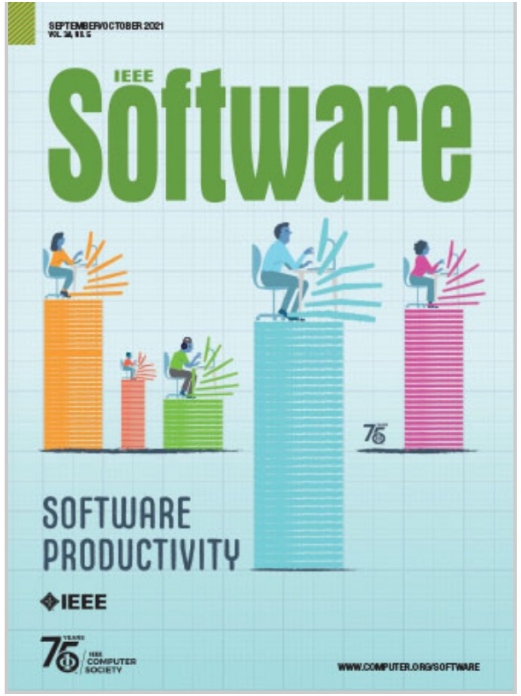
1c. scan

2c. sum

3c. sum and join

Research Opportunities

- Accelerating Orchestration Overheads
- Accelerometer (Sriraman and Dhanotia IEEE Micro May/June 2021)
- Benchmark Suite for Microservices (IISWC 2018)
- DeathStarBench (Gan and Delimitrou IEEE Micro May/June 2020)
- Domain Specific Architectures for Network Acceleration
- FPGAs for RPC Acceleration



IEEE Software

The Monolith Strikes Back: Why Istio Migrated From Microservices to a Monolithic Architecture

Sept.-Oct. 2021, pp. 17-22, vol. 38
DOI Bookmark: 10.1109/MS.2021.3080335

Authors

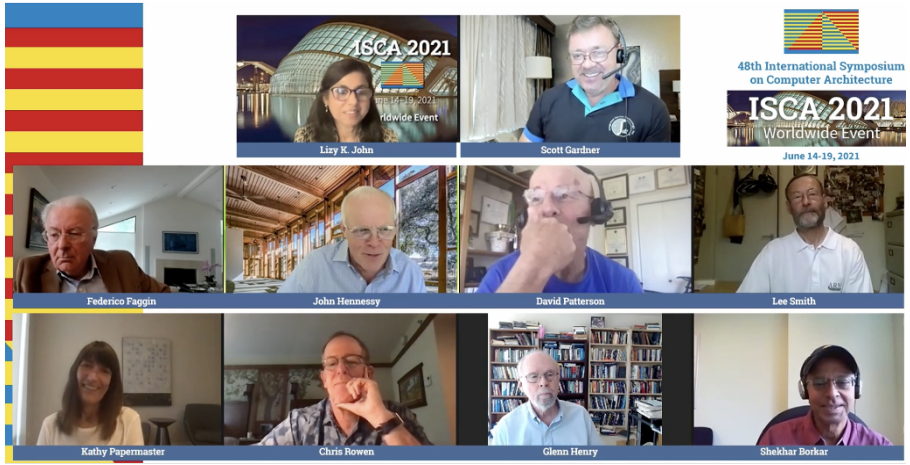
Nabor C. Mendonça, University of Fortaleza
Craig Box, Google
Costin Manolache, Google
Louis Ryan, Google

Modularity affects the entire software lifecycle;

it is not only a development and maintenance concern but also a deployment-time decision.

The operator experience matters, just like user experience and developer experience

- Editors Pautasso and Zimmerman



Workloads

Life, Death, and Games

Machine Learning

But all the stitching together code need Acceleration

Chris Rowen: *A modern application developer does not think very much about instruction-set architecture. They don't even think about the OS interface or the compiler very much. They're very often working at the level of ….Python or JavaScript... They're invoking libraries, they're invoking cloud services, and how they stitch those together is centrally important to what is the distribution of work that needs to be done.*

Thank You! Questions?

Laboratory for Computer Architecture (LCA)
The University of Texas at Austin
lca.ece.utexas.edu