# Research Challenges for a Future Serverless Cloud

Rodrigo Fonseca

Azure Systems Research

SESAME Workshop

Rome, Italy, May 8, 2023

Microsoft

**Azure Systems Research**

*Cloud systems innovation at the core of Azure*

**aka.ms/AzSR**

# Context

- Who am I?
  - Managing the Azure Systems Research group (aka.ms/azsr)
  - We do research in all aspects of cloud infrastructure
- I am not speaking for Azure Functions ☺
- I mention a lot of works here
  - Most not mine!
  - Any errors or omissions are my fault!
- Representing many, from Microsoft and external collaborators

Microsoft
Azure Systems Research
*Cloud systems innovation at the core of Azure*
aka.ms/AzSR

# “Research Challenges for a Future Serverless Cloud”

Is the future of the cloud serverless?

# What is serverless?*

- Operationally
  - “No-ops” – (almost) no configuration
  - Autoscaling down to 0
  - Pay-per-use (rather than per allocation)
  - Fine-grained billing
- Many *services* fit these
  - e.g., Serverless DBs, KVS, OpenAI, ...
- Focus: serverless *custom code*
  - Most popular: Function-as-a-Service, Containers-as-a-Service

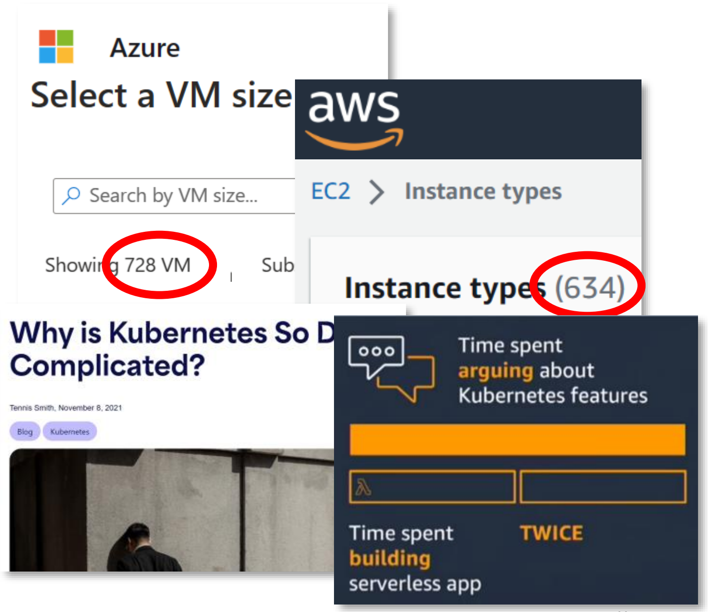
*YDMV

# What is serverless?*

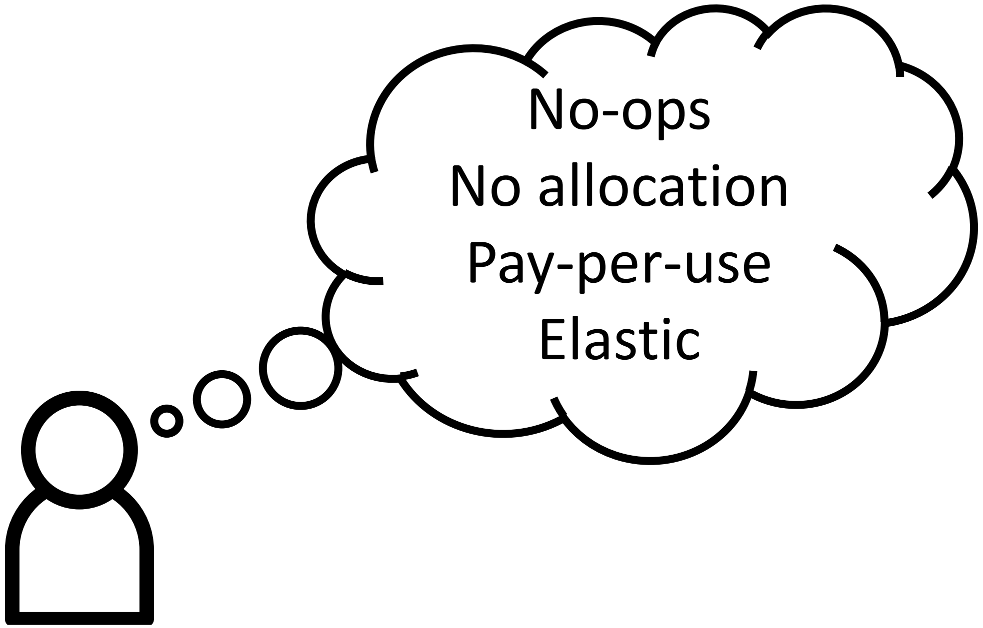
- Function-as-a-Service
  - First model of mostly general computing to have all those characteristics
  - Well-defined life-cycle: triggers, invocation
  - Platform has access to source
    - Optimization opportunity
  - Limitations in duration, memory, communication, state
    - Short, small, ephemeral, stateless
- Easier to pack, measure, autoscale, move!
  - Can improve resource utilization, sustainability

*YDMV

# Is the future of the cloud serverless?




@adrianco, AWS, “Serverless First”




All else being equal: ***rational choice for users***
+ ***competition*** among providers:
probably yes!

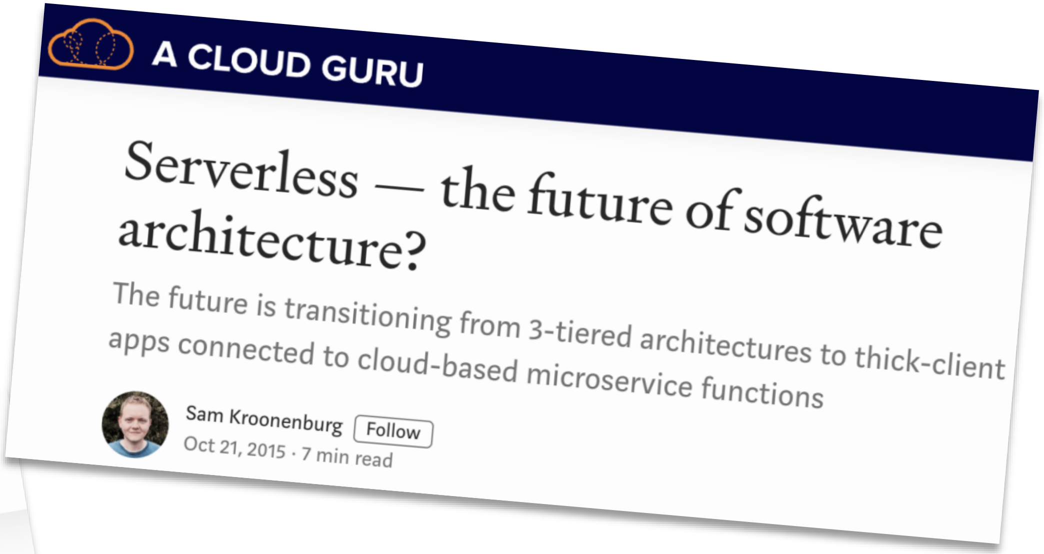
"…more than 20 percent of global enterprises will have deployed serverless computing technologies by 2020."

*Gartner, Dec 2018*

7 Reasons

Why Serverless is the Future

claudiobernasconi.ch



A CLOUD GURU

Serverless — the future of software architecture?

The future is transitioning from 3-tiered architectures to thick-client apps connected to cloud-based microservice functions

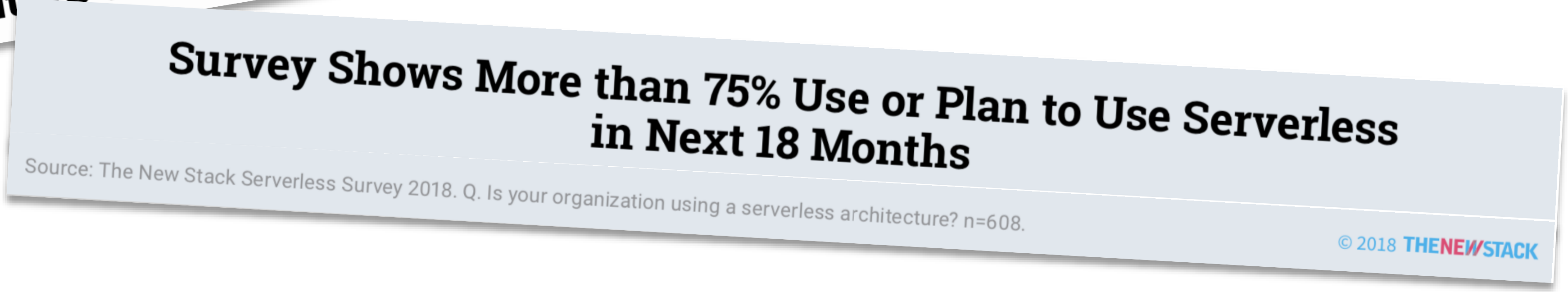
Sam Kroonenburg Follow

Oct 21, 2015 · 7 min read

psc_ TRANSITION TECHNOLOGIES

13 September 2019 ★★★★★ 5 (4)

Why serverless is the future of software and apps



Survey Shows More than 75% Use or Plan to Use Serverless in Next 18 Months

Source: The New Stack Serverless Survey 2018. Q. Is your organization using a serverless architecture? n=608.

© 2018 THENEWSTACK


December 2019

**Serverless Computing: One Step Forward, Two Steps Back**

Joseph M. Hellerstein, Jose Faleiro, Joseph E. Gonzalez, Johann Schleier-Smith, Vikram Sreekanti, Alexey Tumanov and Chenggang Wu
UC Berkeley
{hellerstein,jmfaleiro,jegonzal,jssmith,vikrams,atumanov,cgwu}@berkeley.edu

**Cloud Programming Simplified: A Berkeley View on Serverless Computing**

Eric Jonas, Anurag Khandelwal, Karl Krauth, Johann Schleier-Smith, Qifan Pu, Neeraja Yadwadkar, Ion Stoica, Vikram Sreekanti, Vaishaal Shankar, Joseph E. Gonzalez, David A. Patterson, Chia-Che Tsai, Joao Carreira, Raluca Ada Popa
UC Berkeley
serverlessview@berkeley.edu

**"… we predict that (…) serverless computing will grow to dominate the future of cloud computing."**

# Serverless today

(all else is *not* equal)

- FaaS is used mostly for simple or coarse-grained tasks
  - Stateless, embarrassingly parallel tasks, simple workflows
    - ETL, software testing, API middleware, image processing, etc.
  - Glue to other serverless backends
- Lots of problems are limiting scope
  - Poor performance (vs time to run actual code)
  - Poor handling of state
  - Composition, error handling, communication, coordination are hard
  - No accelerators
  - Very resource-inefficient and costly for serverless provider
- Orders of magnitude too slow and inefficient for many "killer" apps
  - Microservices, ML inference, …

prime video | TECH

Homepage Our Innovation

Video Streaming

# Scaling up the Prime Video audio/video monitoring service and reducing costs by 90%

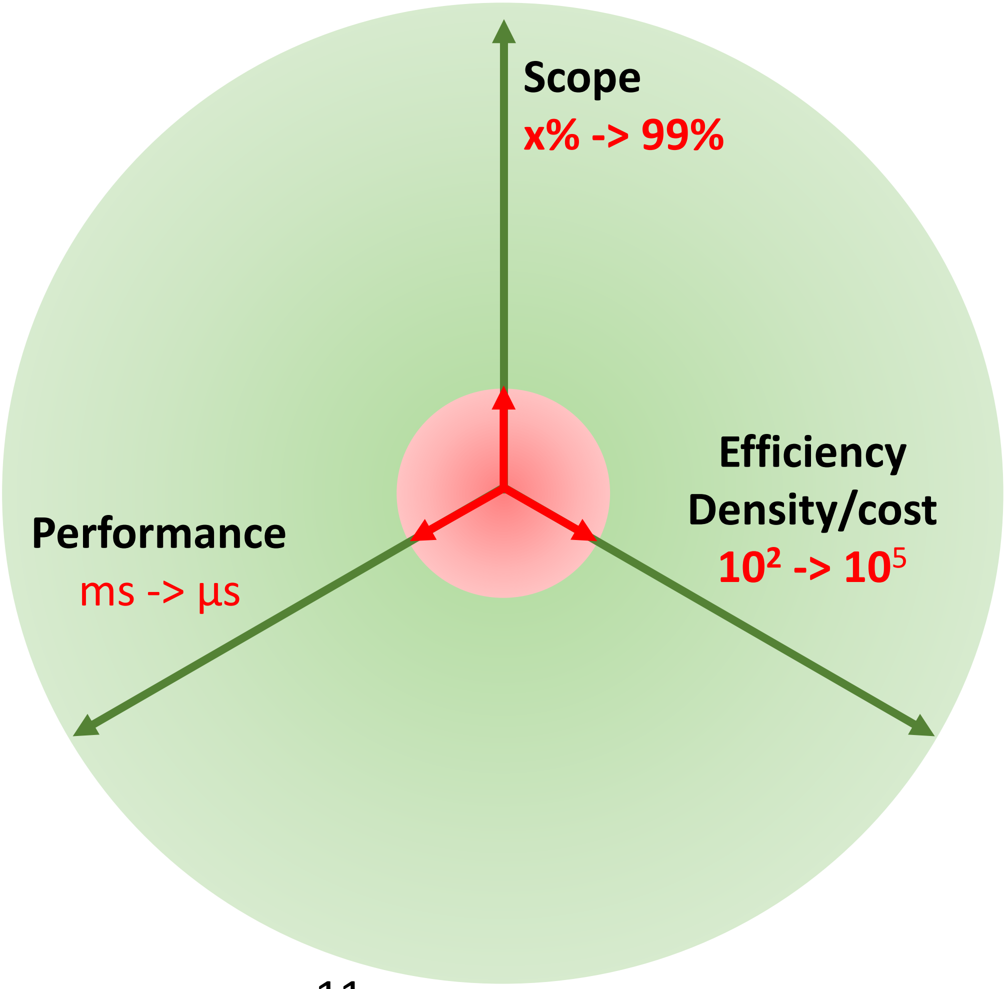
The move from a distributed microservices architecture to a monolith application helped achieve higher scale, resilience, and reduce costs.

Marcin Kolny
Mar 22, 2023

- Initially built with Lambda and Step Functions
  - “(…) good choice for building the service quickly.”
- Too many state transitions on StepFunctions (slow, $$)
- Every frame -> S3 -> Lambda ($$)
- Moved to Elastic Container Service
  - Frame data does not leave container
  - Had to replicate containers, implement load balancer manually

# How do we get there?

- Radically increase
  - Scope: what is serverless good for?
    - From x% -> 99% of applications
  - Performance: closer to hardware limits
    - From ms -> µs
  - Efficiency: make it cost effective
    - Time: minimize overheads (non-billable time!)
    - Space: from $10^2$ to $10^5$ per node



# Increasing scope

- "Serverless should be the default choice

  Only go away for niche use cases."

  Sebastian Burckhardt (paraphrased)

# Increasing scope

- Programming model
  - Lots of great research here
    - Many “X as serverless” papers
  - Stateful computation
    - Azure Durable Functions, Step Functions
  - Correct
    - Beldi [OSDI’20]
  - Transformation
    - Crucial [ACM ToSEM v31i3], Wukong [SoCC’20]
  - …

# Increasing scope

- Improving performance
- Reducing overheads
- Reducing complexity
- and lots of other things must be right
  - Security, debugging, observability, pricing, …

# Increasing scope

## With Great Freedom Comes Great Opportunity: Rethinking Resource Allocation for Serverless Functions

Muhammad Bilal*
IST(ULisboa)/INESC-ID and UCLouvain

Marco Canini
KAUST

Rodrigo Fonseca
Azure Systems Research

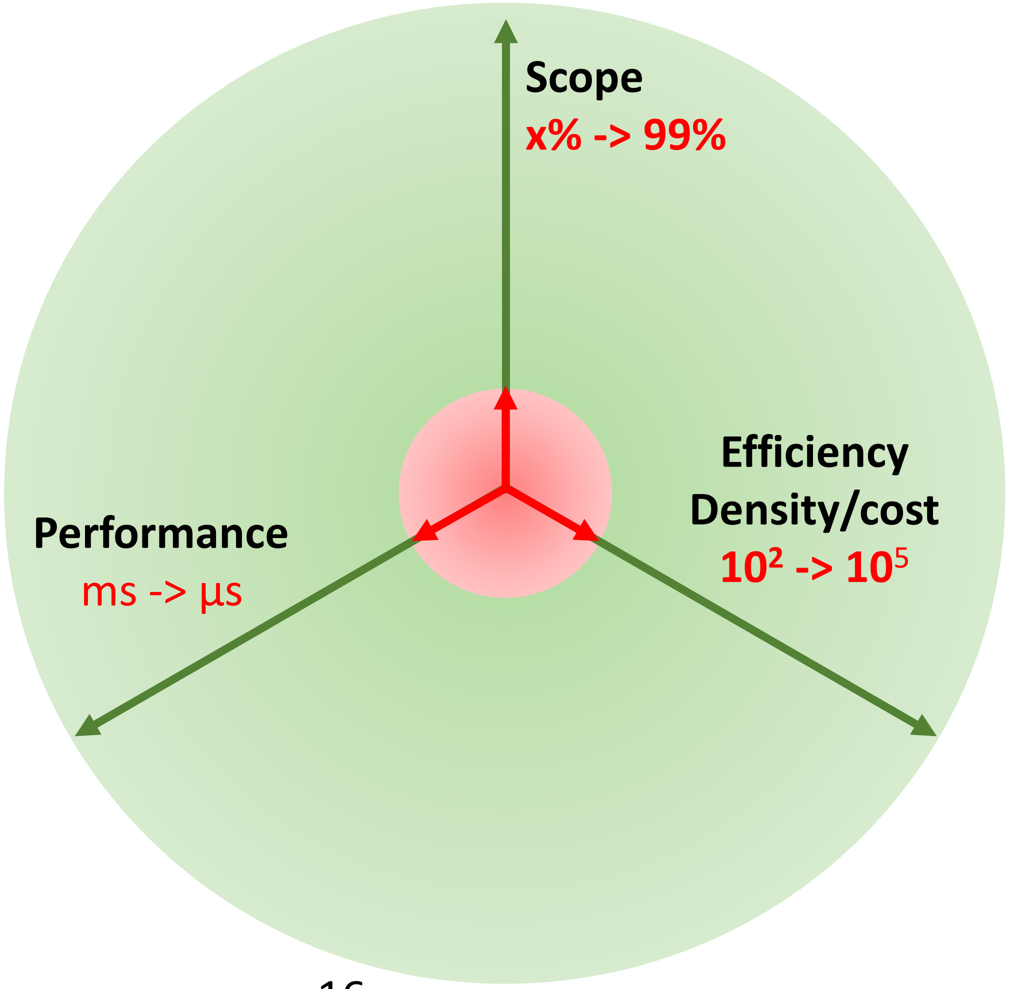
Rodrigo Rodrigues
IST(ULisboa)/INESC-ID

*EuroSys'23, Wednesday 14:50*

- Changes the interface to *tangibles*:
  - Provider chooses resources (CPU, memory, arch)
  - Exposes Price, Performance choices
    - Points in the Pareto front or
    - Best point given a user preference for $ or perf
    - Best performance given a budget
  - Could also include carbon

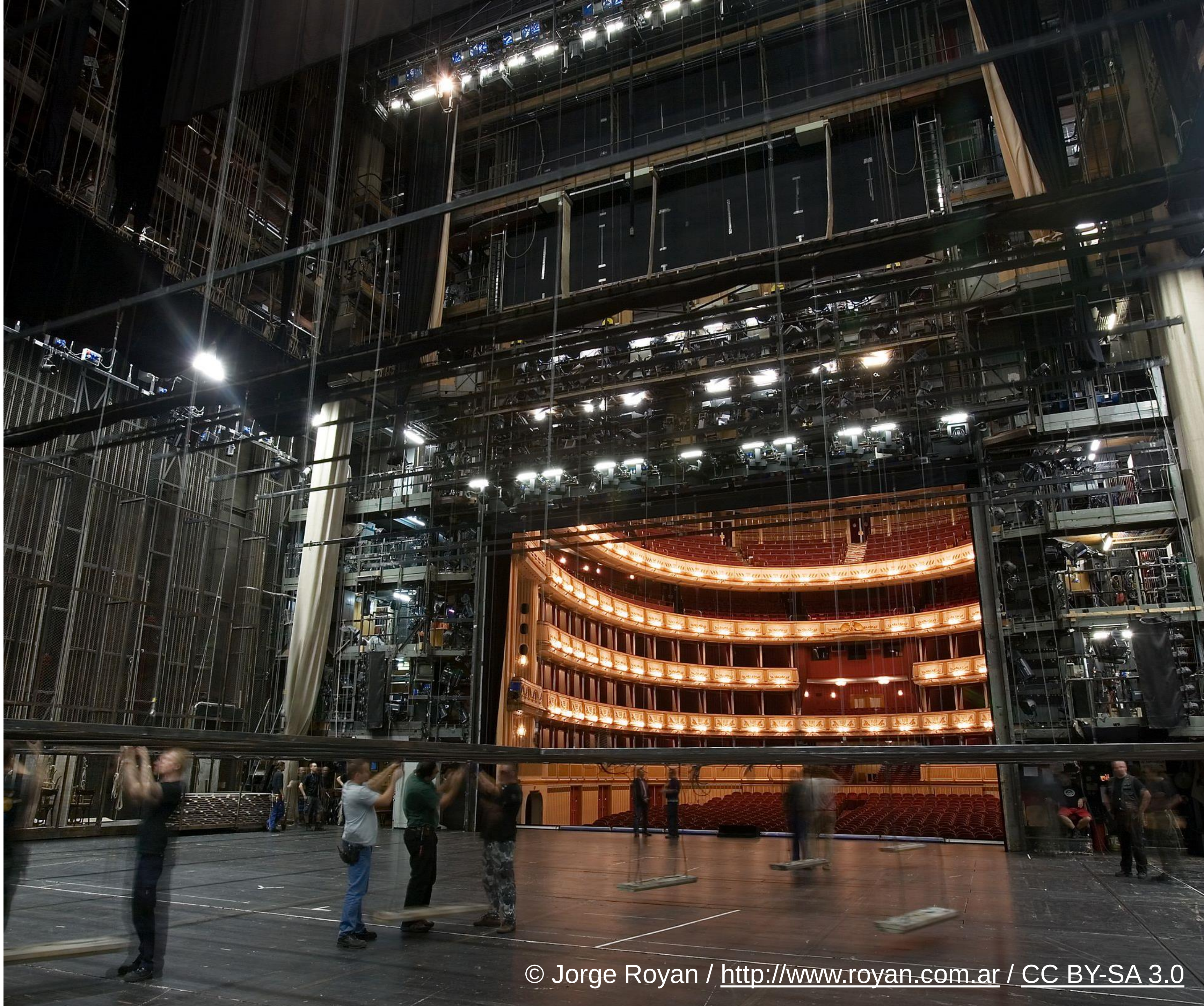
# How do we get there?

- Radically increase
  - Scope: what is serverless good for?
    - From x% -> 99% of applications
  - Performance: closer to hardware limits
    - From ms -> µs
  - Efficiency: make it cost effective
    - Time: minimize overheads (non-billable time!)
    - Space: from $10^2$ to $10^5$ per node



# Provider challenges

# Performance and efficiency



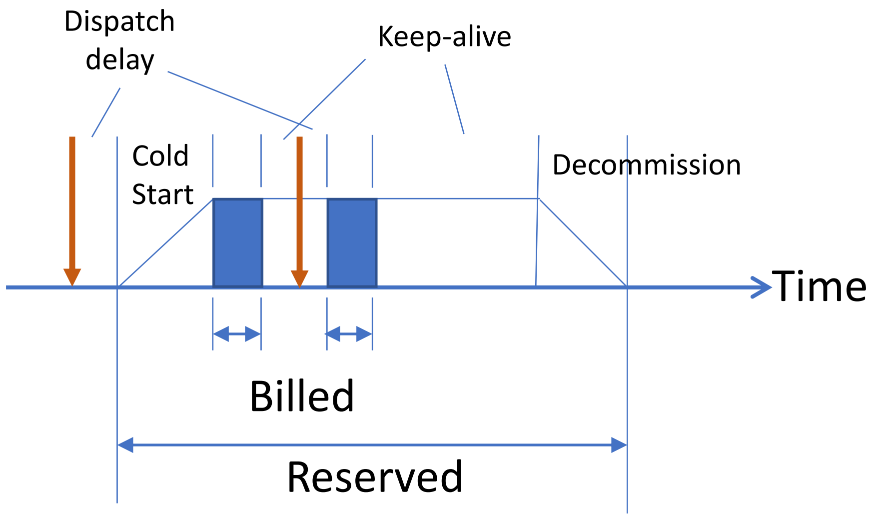
# Performance and efficiency

Legend: ✓ - Feature helps achieve the goal
C - Feature conflicts with the goal

*Features*

| *Goals* | Snapshots | Minimalist Environment | Efficient Control Plane | Instruction Prefetching | Local Scheduling | Direct HW Access | Caching | Memory sharing | **Hypervisor Isolation** |
|---|---|---|---|---|---|---|---|---|---|
| Fast Cold Starts | | | | | | | | | |
| Fast Warm Starts | | | | | | | | | |
| High Density | | | | | | | | | |
| Efficient Data Sharing | | | | | | | | | |
| Locality | | | | | | | | | |

# Fast cold starts




- A lot of research!
  - 34 out of 164 papers in [1]
- Goal: from many seconds to sub-ms

[1] Jinfeng Wen et al. “Rise of the Planet of Serverless Computing: A Systematic Review”. ACM TOSEM, Jan 2023

# Fast cold starts

- Snapshots
  - Catalyst [ASPLOS’19], REAP[ASPLOS’20], FaaSnap [EuroSys’22], Faasm [ATC’20], Virtines [EuroSys’22],…
- Sharing compiler (JIT) state
  - Hot starts [HotOS’21]
- Minimalist environment
  - Firecracker [NSDI’20], Virtines, Faasm,…
- Reducing cold start numbers
  - Serverless in the Wild [ATC’19], FaasCache [ASPLOS’21]

# Fast cold starts – scale out

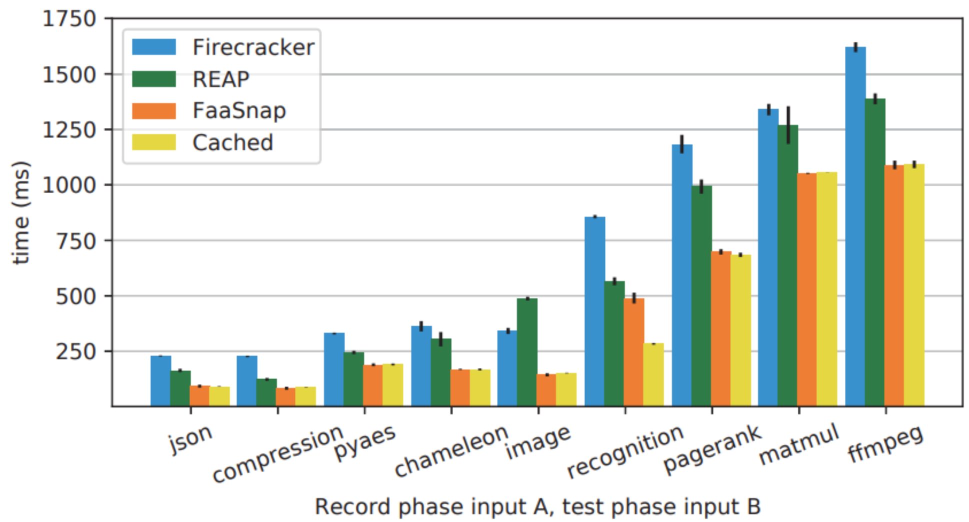
- Increase scope to very elastic applications
  - E.g., wide DAGs
- Efficient control plane is critical and under-studied
  - Networking: Particle [SoCC’20], Mohan et al. [HotCloud’19]
  - *Do we always need full-fledged networking?*
- Next session:
  - Work in Progress: The Neglected Cost of Serverless Cluster Management. Lazar Cvetković (ETH Zürich); me; Ana Klimovic (ETH Zürich)
  - Cluster schedulers not designed to schedule very ephemeral sandboxes
  - *What is special about serverless for cluster schedulers?*

# Cold starts & hypervisors

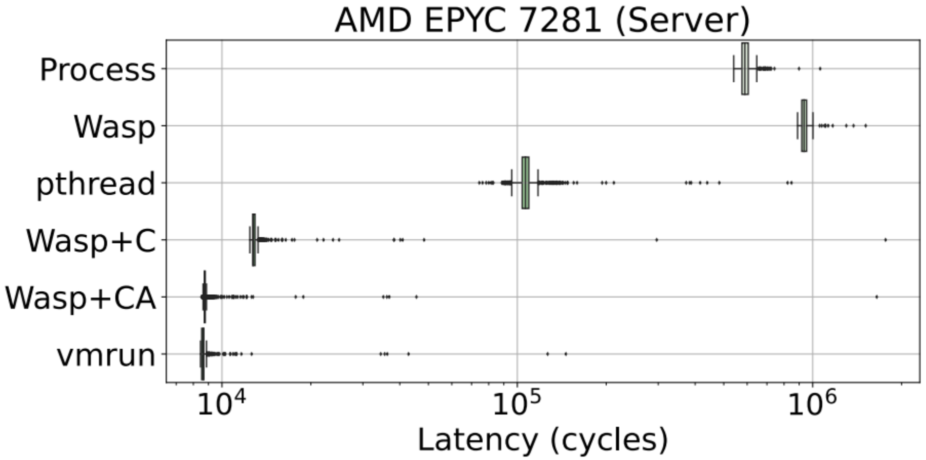
- Tradeoff between isolation cold starts?
  - Faasm [ATC'20]
  - Firecracker [NSDI'20], REAP [ASPLOS'20], FaaSnap [EuroSys'22]
  - Virtines [EuroSys'22]

| | | Containers | VMs | Unikernel | SFI | Faaslet |
|---|---|---|---|---|---|---|
| Func. | Memory safety | ✓ | ✓ | ✓ | ✓ | ✓ |
| | Resource isolation | ✓ | ✓ | ✓ | ✗ | ✓ |
| | Efficient state sharing | ✗ | ✗ | ✗ | ✗ | ✓ |
| | Shared filesystem | ✓ | ✗ | ✗ | ✓ | ✓ |
| Non-func. | Initialisation time | 100 ms | 100 ms | 10 ms | 10 µs | 1 ms |
| | Memory footprint | MBs | MBs | KBs | Bytes | KBs |
| | Multi-language | ✓ | ✓ | ✓ | ✗ | ✓ |

**Table 1: Isolation approaches for serverless** (Initialisation times include ahead-of-time snapshot restore where applicable [16,25,61].)




FaaSnap [EuroSys'22]




~5µs ~50µs ~500µs

Virtines [EuroSys'22]

# Performance and Efficiency

Legend: ✓ - Feature helps achieve the goal
C - Feature conflicts with the goal

*Features*

*Goals*

| | Snapshots | Minimalist Environment | Efficient Control Plane | Instruction Prefetching | Local Scheduling | Direct HW Access | Caching | Memory sharing | **Hypervisor Isolation** |
|---|---|---|---|---|---|---|---|---|---|
| Fast Cold Starts | ✓ | ✓ | ✓ | | | | | | C* |
| Fast Warm Starts | | | | | | | | | |
| High Density | | | | | | | | | |
| Efficient Data Sharing | | | | | | | | | |
| Locality | | | | | | | | | |

# Fast warm starts

- Two components:
  - Invocation / Return – “killer microseconds”
  - Computation – ideally native speeds (but WASM is not bad!)
- Gap to RPC systems: ~2-3 Orders of magnitude, ms -> µs

Cloudburst [VLDB’20]

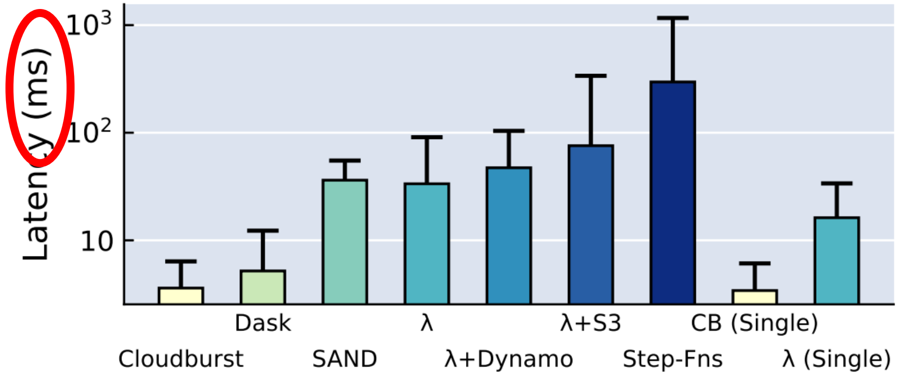
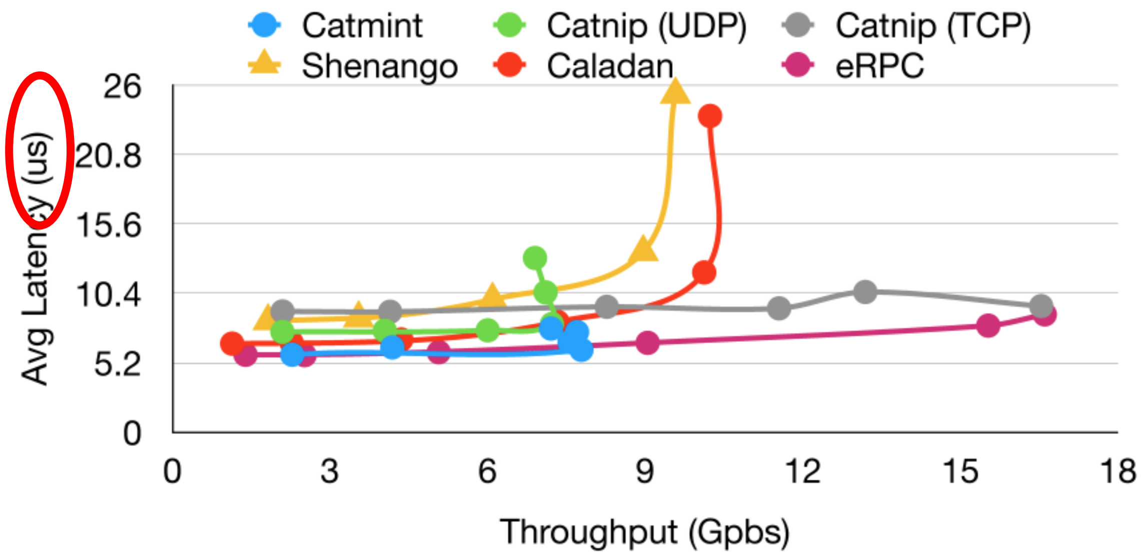


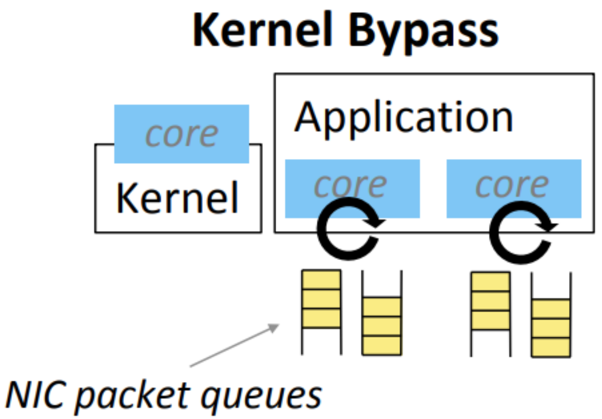

**Figure 1: Median (bar) and 99th percentile (whisker) latency for `square(increment(x: int))`. Cloudburst matches the best distributed Python systems and outperforms other FaaS systems by over an order of magnitude (§6.1).**



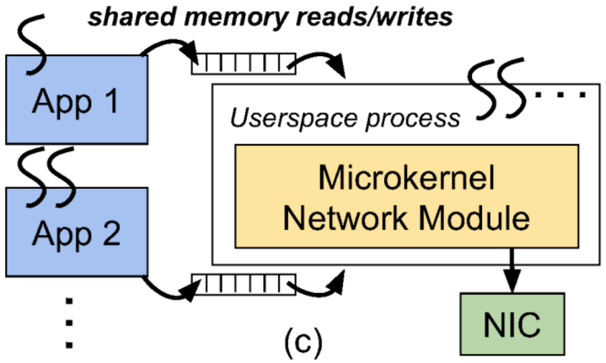

Demikernel [SOSP’21]

# Fast warm starts

- *Is this gap fundamental?*
- Direct access to hardware
  - E.g., DPDK + LibOS
- Tight control of threading, core scheduling
- Conflicts with fast cold start, density
  - Some designs: dedicated cores (polling)
  - Fixed-size buffers (partition memory)
- Tradeoff
  - One copy vs **single** core polling
    - E.g. Shenango [NSDI'19], SNAP [SOSP'19]
- Vs Hypervisor
  - *Can we achieve the same performance under virtualization?*




Example from Shenango (not the Shenango design)




Google's SNAP design

# Fast warm starts

- Instruction pre-fetching
  - Jukebox [ISCA’22]: combat thrashing of instruction cache with lukewarm functions
- Sharing compiler (JIT) state
  - Hot starts [HotOS’21]
- Local scheduling
  - e.g., Nigthcore [ASPLOS’21]: bypass cluster scheduler if next function can be run locally

# Performance and efficiency

Legend: ✓ - Feature helps achieve the goal
C - Feature conflicts with the goal

*Features*

| *Goals* | Snapshots | Minimalist Environment | Efficient Control Plane | Instruction Prefetching | Local Scheduling | Direct HW Access | Caching | Memory sharing | **Hypervisor Isolation** |
|---|---|---|---|---|---|---|---|---|---|
| Fast Cold Starts | ✓ | ✓ | ✓ | | | **C** | | | C* |
| Fast Warm Starts | | | | ✓ | ✓ | ✓ | | | C |
| High Density | | | | | | **C** | | | |
| Efficient Data Sharing | | | | | | | | | |
| Locality | | | | | | | | | |

# Increasing density

- Crucial for cost reduction
- With elasticity, can greatly improve sustainability
  - Both scope 2 (electricity), and scope 3 (embedded carbon)

# Increasing density

| | Docker | Faaslets | Proto-Faaslets | vs. Docker |
|---|---|---|---|---|
| Initialisation | 2.8 s | 5.2 ms | 0.5 ms | **5.6K×** |
| CPU cycles | 251M | 1.4K | 650 | **385K×** |
| PSS memory | 1.3 MB | 200 KB | 90 KB | **15×** |
| RSS memory | 5.0 MB | 200 KB | 90 KB | **57×** |
| Capacity | ~8 K | ~70 K | >100 K | **12×** |

**Table 3: Comparison of Faaslets vs. container cold starts (no-op function)**

- Minimalist environment
  - Faasm [ATC'20]
    - Wasm
    - 12x more instances than Docker (no-op function)
  - Firecracker
    - Smaller VMM, simplified Linux
  - Unikernels
    - e.g., SEUSS [EuroSys'20], page sharing and COW
  - Even simpler
    - Virtines [EuroSys'22]
- Recall conflict with direct HW access (not fundamental)

| Isolation Method | Creation Rate (per second) | Cache Density |
|---|---|---|
| Firecracker microVM | 1.3 | 450 |
| Docker w/ overlay2 fs | 5.3 | 3000 |
| Linux process | 45 | 4200 |
| SEUSS UC | 128.6 | 54000 |

SEUSS [EuroSys'20]

# Performance and efficiency

Legend: ✓ - Feature helps achieve the goal
C - Feature conflicts with the goal

*Features*

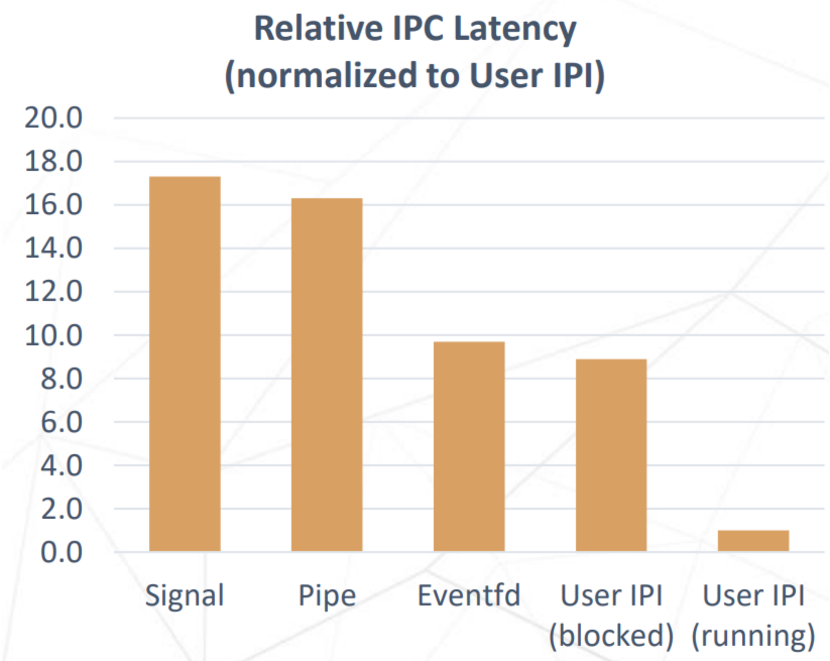
| *Goals* | Snapshots | Minimalist Environment | Efficient Control Plane | Instruction Prefetching | Local Scheduling | Direct HW Access | Caching | Memory sharing | **Hypervisor Isolation** |
|---|---|---|---|---|---|---|---|---|---|
| Fast Cold Starts | ✓ | ✓ | ✓ | | | C | | | C* |
| Fast Warm Starts | | | | ✓ | ✓ | ✓ | | | C |
| High Density | | ✓ | | | | C | | | C* |
| Efficient Data Sharing | | | | | | | | | |
| Locality | | | | | | | | | |

# Efficient data sharing

- Controlled shared memory
  - Faasm allows for shared memory among functions (Wasm)
  - Distributed KVS across functions
- Distributed caching among instances
  - OFC [EuroSys'21], Faa$T [SoCC'21] (many reads still cross the network)
- Efficient storage
  - Pocket [OSDI'18], Locust [ATC'21]
- *Can we use fast remote memory (e.g., CXL)?*

# Efficient data sharing

- vs Virtualization
  - Initially at odds, not fundamental
  - Need us-scale signaling to share among VMs, SENDUIPI promising [1]




[1] https://lpc.events/event/11/contributions/985/attachments/756/1417/User_Interrupts_LPC_2021.pdf

# Performance and efficiency

Legend: √ - Feature helps achieve the goal
C - Feature conflicts with the goal

*Features*

| *Goals* | Snapshots | Minimalist Environment | Efficient Control Plane | Instruction Prefetching | Local Scheduling | Direct HW Access | Caching | Memory sharing | **Hypervisor Isolation** |
|---|---|---|---|---|---|---|---|---|---|
| Fast Cold Starts | √ | √ | √ | | | C | | | C* |
| Fast Warm Starts | | | | √ | √ | √ | | | C |
| High Density | | √ | | | | C | | | C* |
| Efficient Data Sharing | | | | | | | √ | √ | C |
| Locality | | | | | | | | | |

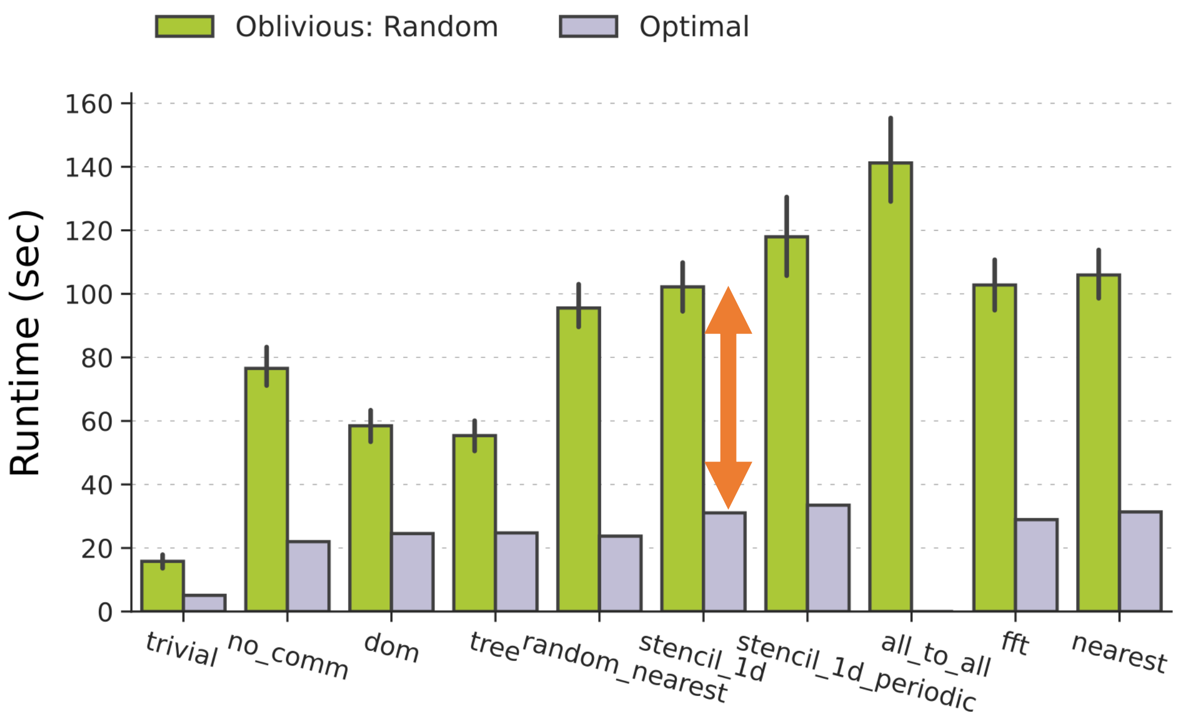
# Locality

- Plain serverless does not have a notion of locality
  - Despite reusing containers
  - Palette [EuroSys'23] allows apps to express locality through hints
  - Run where data is
- Programming model
  - Pherormone [NSDI'23], Cloudburst [VLDB'20], Ray [OSDI'18]
- Function shipping
  - Shredder [SoCC'19]

**Palette Load Balancing: Locality Hints for Serverless Functions**

Mania Abdi* Northeastern University
Samuel Ginzburg* Princeton University
Charles Lin* Anyscale Inc.
José M Faleiro* Unaffiliated
Íñigo Goiri Azure Systems Research
Gohar Chaudhry Azure Systems Research
Ricardo Bianchini Azure Systems Research
Daniel S. Berger Azure Systems Research
Rodrigo Fonseca Azure Systems Research

*EuroSys'23, Wednesday 14:50*




Slaugther et al., "Task Bench: A Parameterized Benchmark for Evaluating Parallel Runtime Performance.", SC'20

# Performance and efficiency

Legend: ✓ - Feature helps achieve the goal
C - Feature conflicts with the goal

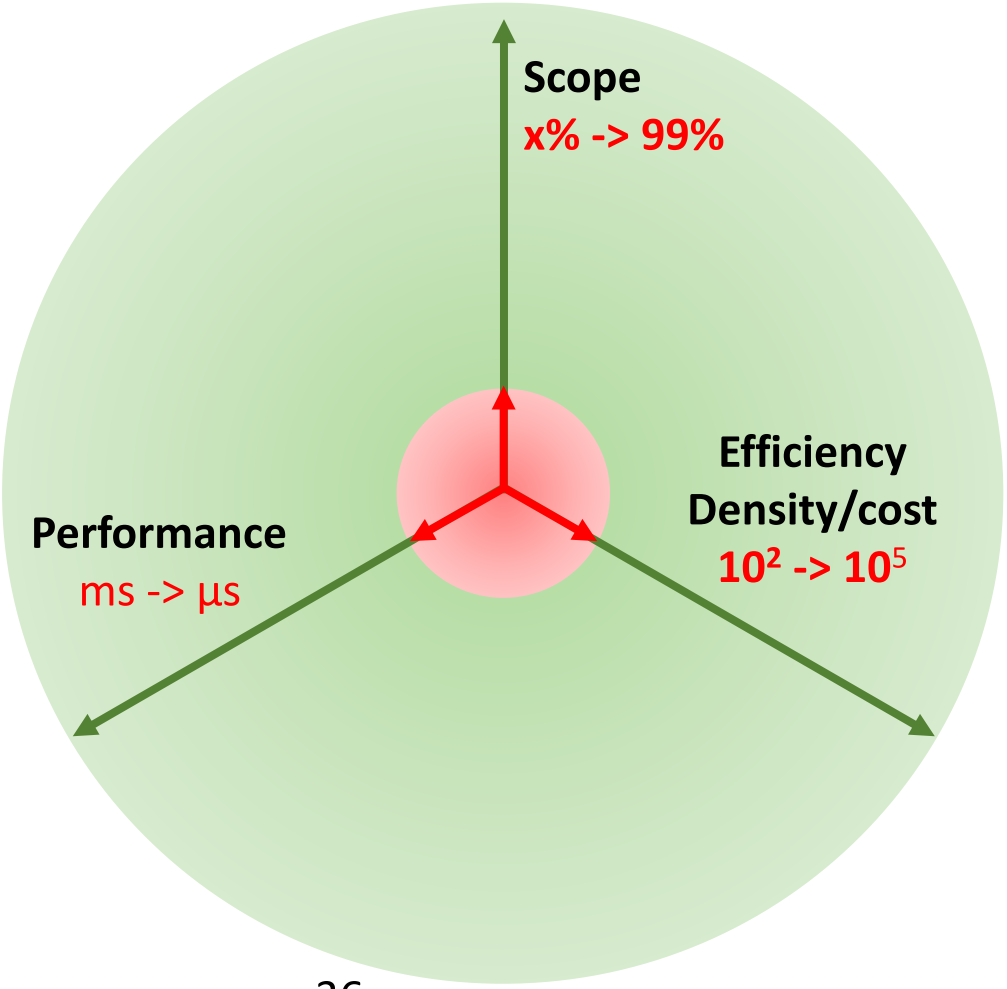
*Features*

*Goals*

| | Snapshots | Minimalist Environment | Efficient Control Plane | Instruction Prefetching | Local Scheduling | Direct HW Access | Caching | Memory sharing | **Hypervisor Isolation** |
|---|---|---|---|---|---|---|---|---|---|
| Fast Cold Starts | ✓ | ✓ | ✓ | | | C | | | C* |
| Fast Warm Starts | | | | ✓ | ✓ | ✓ | | | C |
| High Density | | ✓ | | | | C | | | C* |
| Efficient Data Sharing | | | | | | | ✓ | ✓ | C |
| Locality | | | | | ✓ | | ✓ | ✓ | |

# How do we get there?

- Radically increase
  - Scope: what is serverless good for?
    - From x% -> 99% of applications
  - Performance: closer to hardware limits
    - From ms -> µs
  - Efficiency: make it cost effective
    - Time: minimize overheads (non-billable time!)
    - Space: from $10^2$ to $10^5$ per node



# Conclusion

- Serverless will be a large part of the future of the cloud!
- Exciting set of challenges
  - Lots of work going on
- **Density & multi-tenancy** make it more interesting!
- "Plenty of room at the bottom“
- Do not be restricted by current offerings
  - Assume they can change from the inside ;)

# Collaborators

- Microsoft
  - Íñigo Goiri, Enrique Saurez, Esha Choukse, Ricardo Bianchini, Sameh Elnikety
  - Azure Functions Team
- External
  - Ana Klimovic, Lazar Cvetković (ETH)
  - Adam Belay, Gohar Chaudhry, Josh Fried (MIT)
  - Benjamin Carver, Yue Cheng (GMU)
  - Marco Canini (KAUST), Rodrigo Rodrigues (IST), Muhammad Bilal
  - Mania Abdi (NEU/Google)
  - Sam Ginzburg (Princeton), Charles Lin (Anyscale), Jose Faleiro

# Thank you & Questions

Microsoft

**Azure Systems Research**

*Cloud systems innovation at the core of Azure*

**aka.ms/AzSR**

- Contact us for collaborations, visits, internships & full-time positions!