Firecracker

Lightweight Virtualization for Serverless Applications

Alexandru Agache, **Marc Brooker**, Andreea Florescu, Alexandra Iordache, Anthony Liguori, Rolf Neugebauer, Diana-Maria Popa, and Phil Piwonka

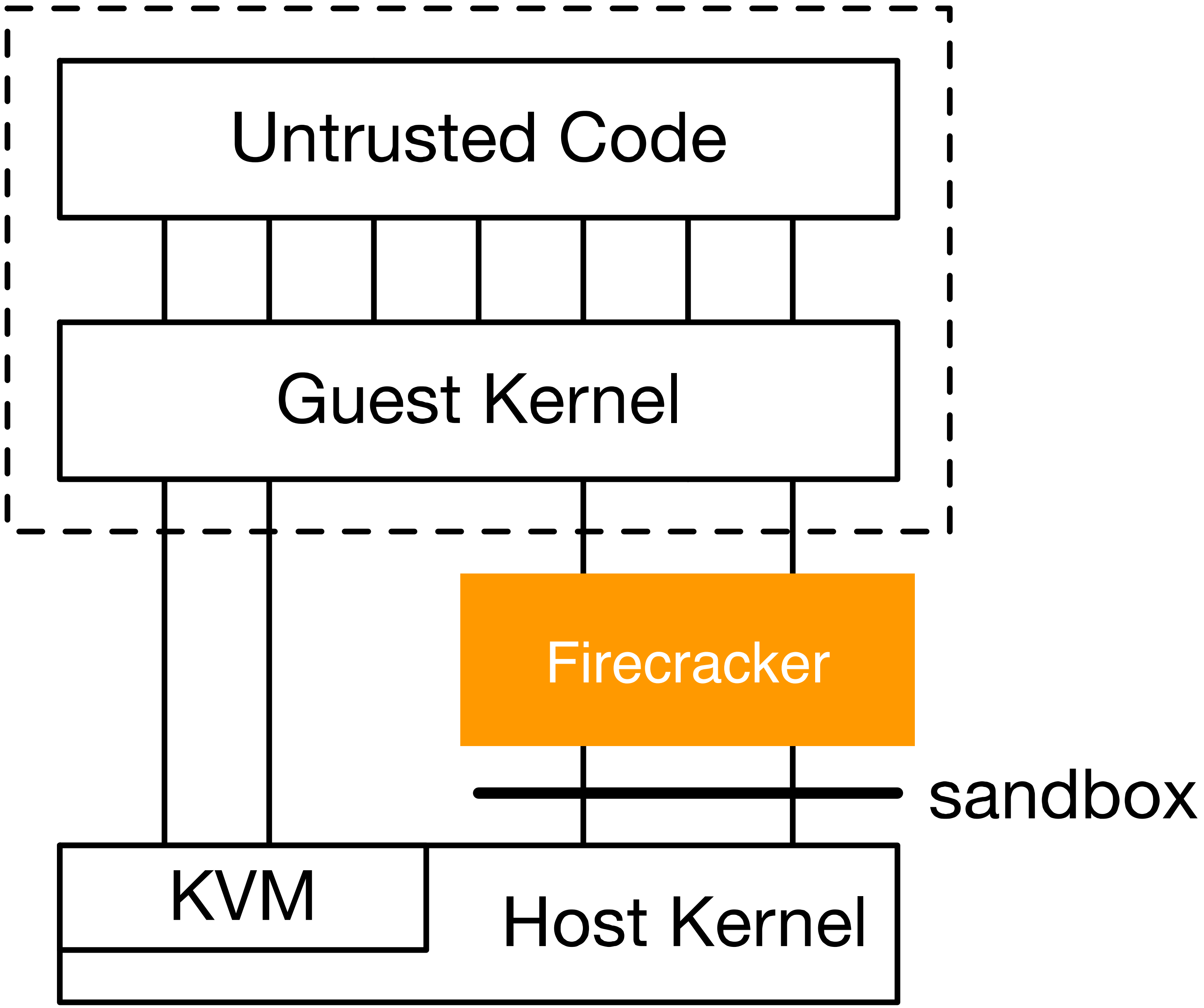
February 2020

- What is Firecracker?
- Why Firecracker?
- Performance
- Challenges for the Future

Firecracker

aws

Firecracker is an open source VMM that is purpose-built for creating and managing secure, multi-tenant container and function-based services.


Untrusted Code
Guest Kernel
Firecracker
sandbox
KVM
Host Kernel

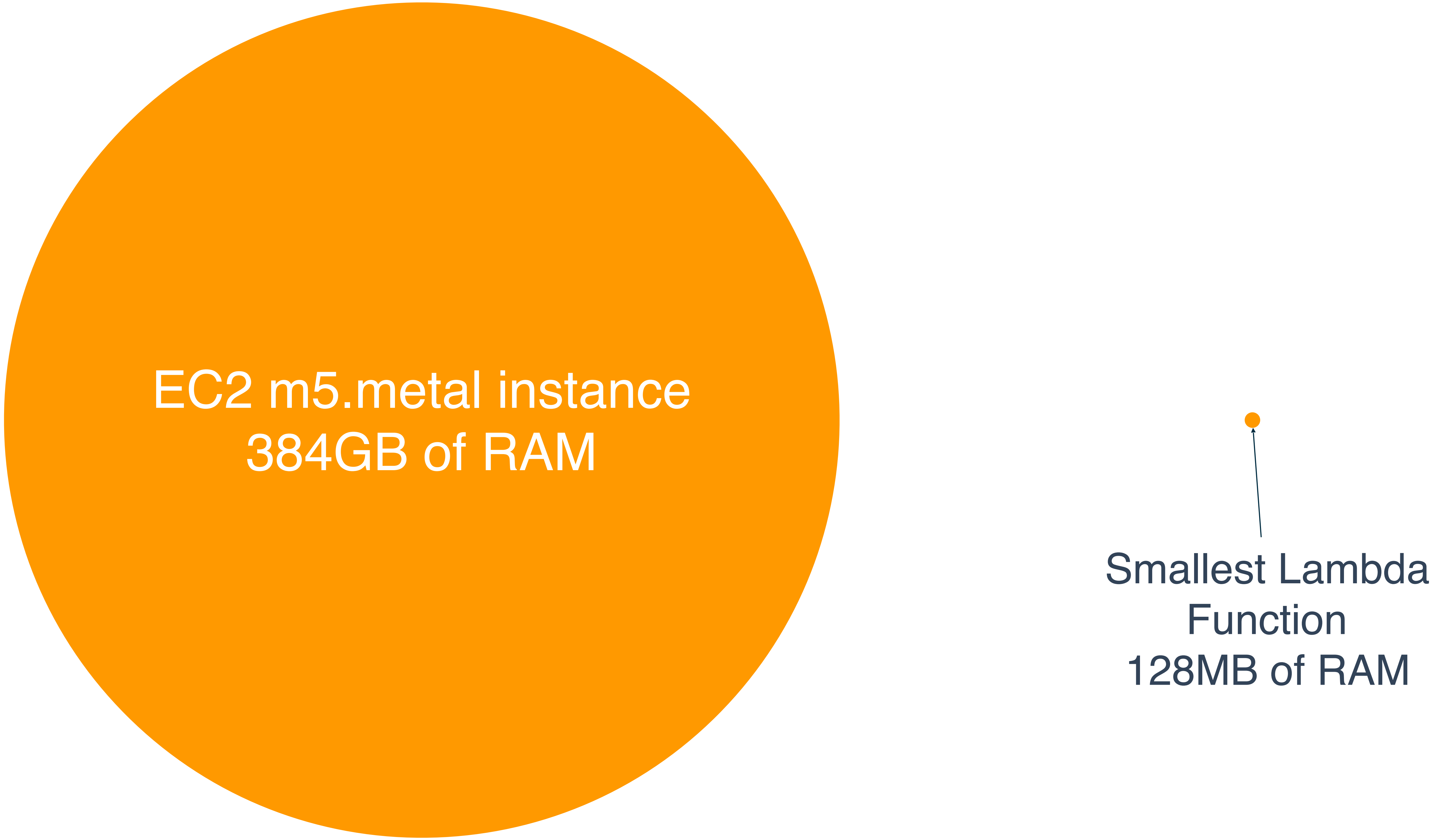
- Started with a branch of *crosvm*
  - Removed >50% of the code
- 96% fewer lines of code than QEMU
- Simplified device model
  - no BIOS, no PCI, etc

- Linux and OSv guests
- Integrated with container ecosystem
  - Kata, FireKube, containerd
- Apache 2.0 license
  - https://github.com/firecracker-microvm/firecracker

- In production in AWS Lambda
  - Millions of workloads
  - Trillions of requests/month

Why Firecracker?

Firecracker


EC2 m5.metal instance
384GB of RAM
Smallest Lambda
Function
128MB of RAM

Isolation:

It must be safe for multiple functions to run on the same hardware.

Overhead & Density:

Thousands of functions on a single machine.

Performance:

Functions must perform similarly to running natively.

Compatibility:

Arbitrary Linux binaries and libraries. No code changes or recompilation.

Soft Allocation:

It must be possible to over commit CPU, memory and other resources.

Firecracker ticks all these boxes

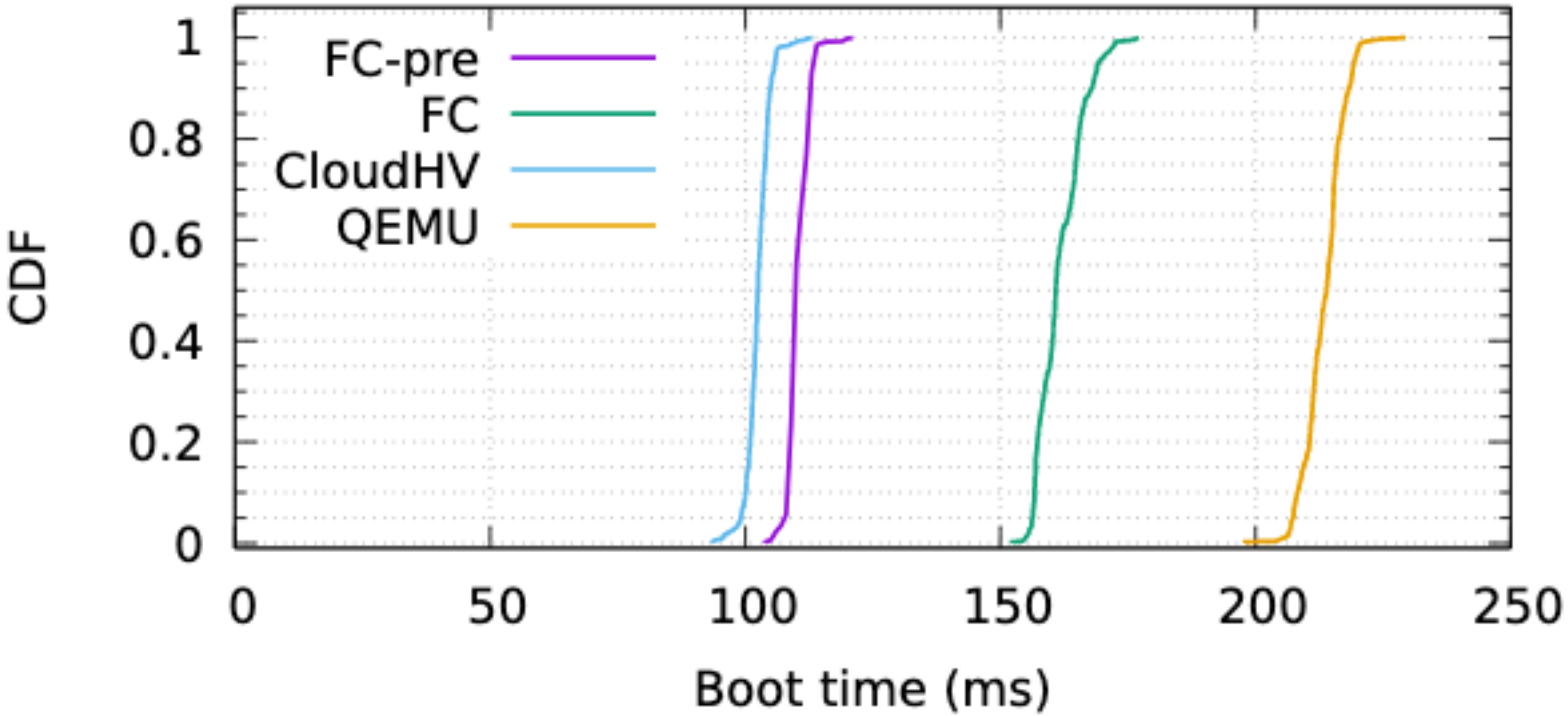
- QEMU/KVM: density and overhead challenges
- Linux *containers*: isolation and compatibility challenges
- LibOS approaches: compatibility challenges
- Language VM isolation: compatibility and isolation challenges

Performance

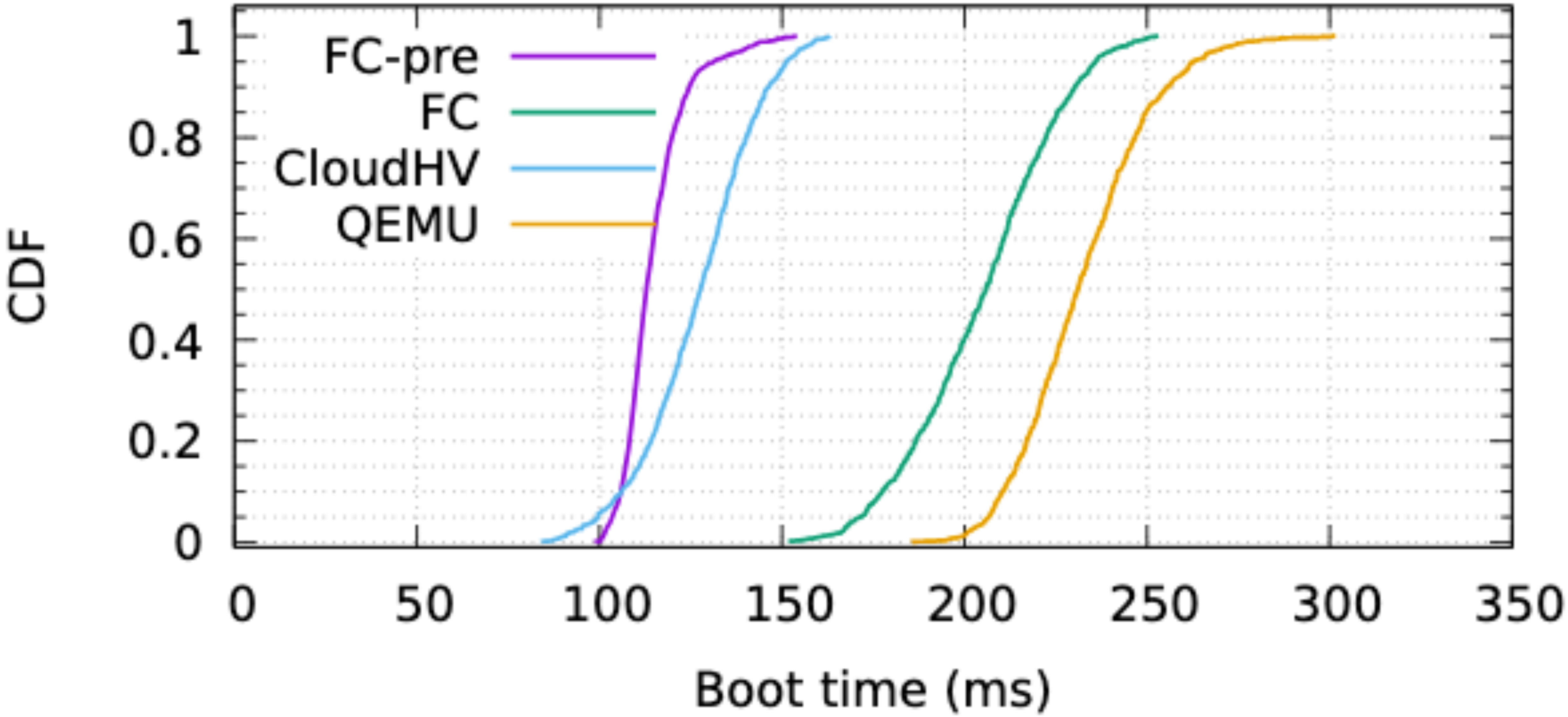
Firecracker

aws

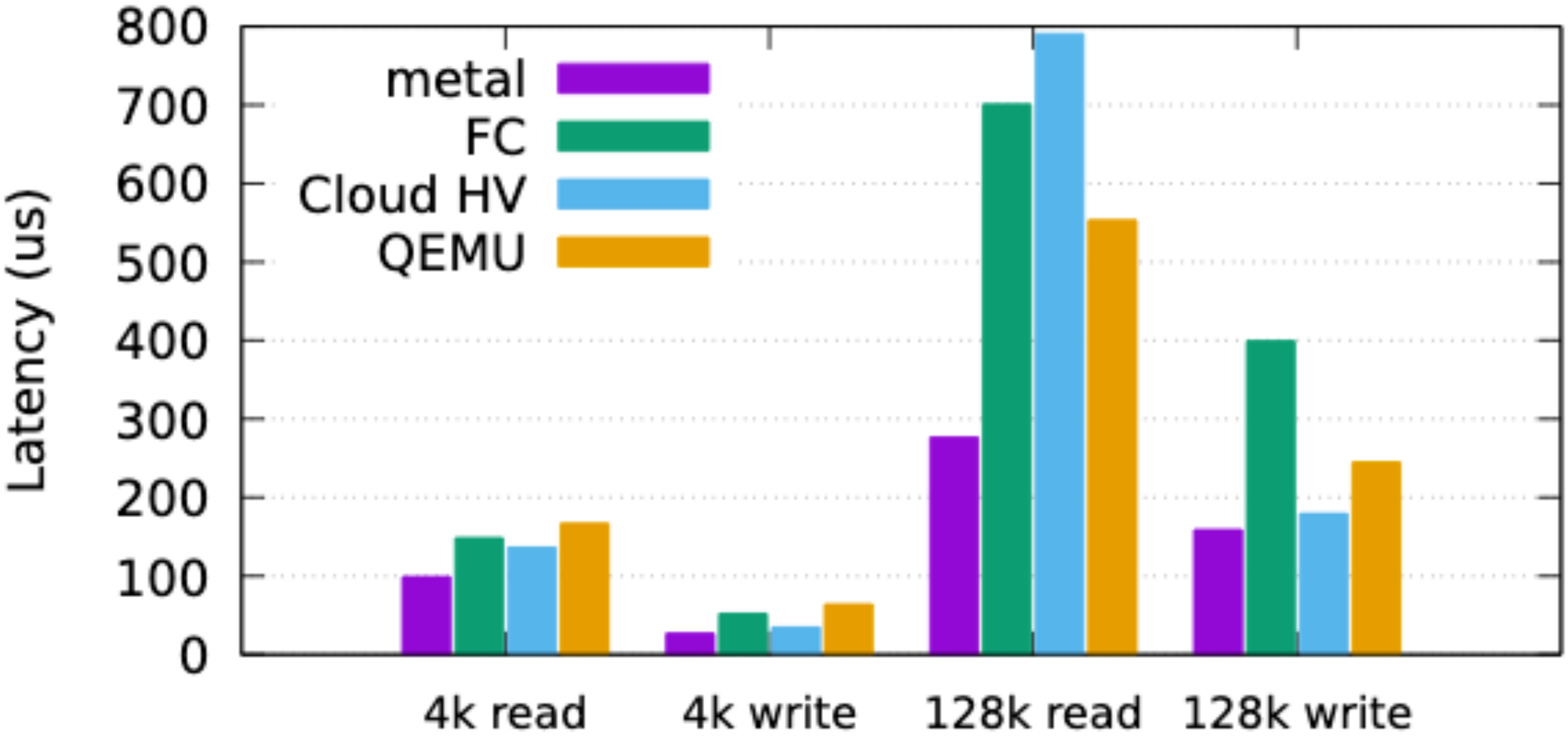
MicroVM start latency (serial)



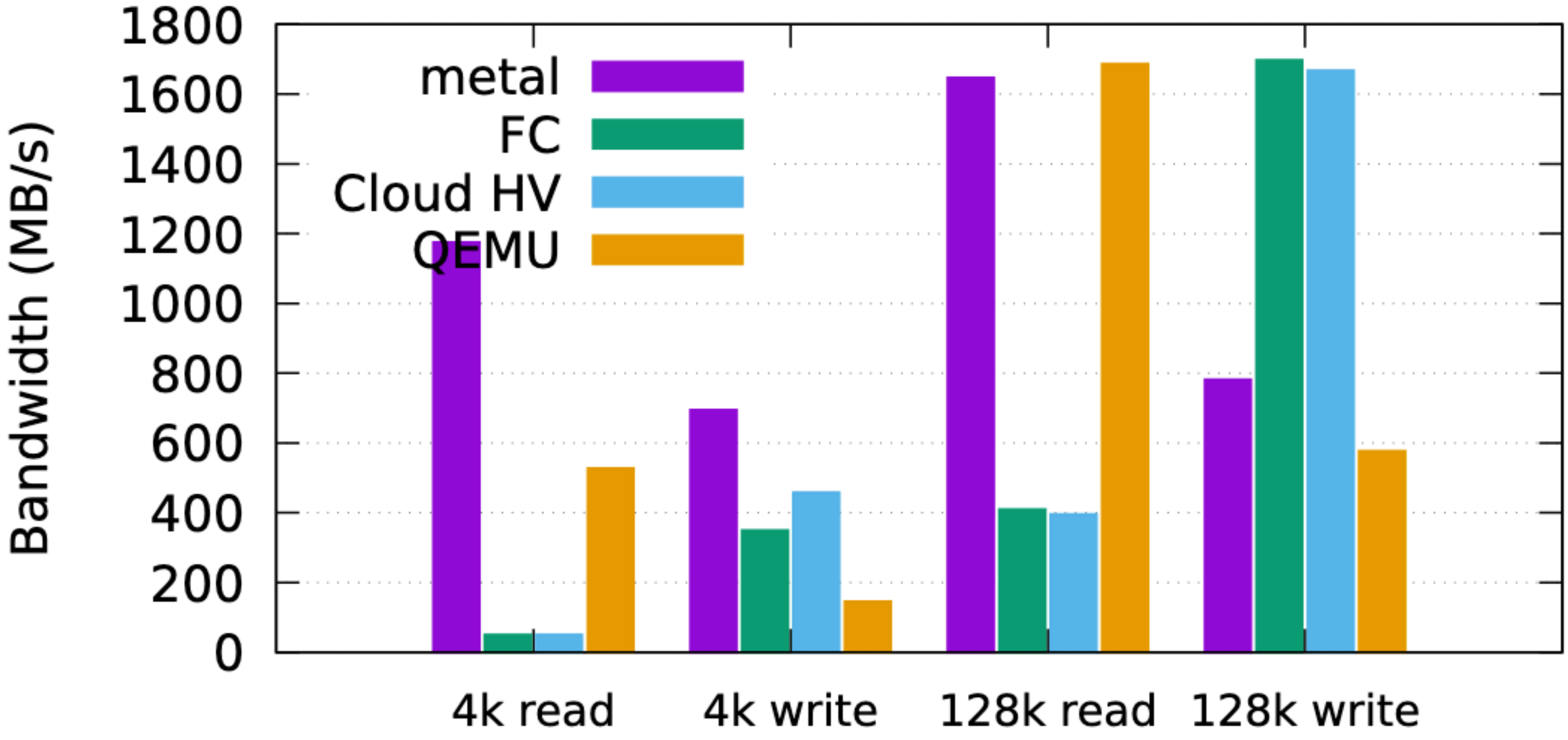
MicroVM start latency (50 parallel)



QD1 IO Latency vs Bare Metal



QD32 IO Throughput vs Bare Metal



Operational Lessons

Lesson #1: Compatibility is Hard

Just disabling Hyperthreading revealed two bugs in Apache Commons HTTP Client, and one in our own code.

Re-implementing OS components would have been worse.

Performance compatibility too!

Lesson #2:
Immutable, Time-Limited Machines

Common systems-administration tools like *rpm* and *dpkg* are non-deterministic.

Limiting max fleet life helps operational hygiene.

Lesson #3:
The Job is Never Done

Changing customer needs means that there are always improvements to be made.

The Future

Opportunities

IO performance and scalability (offload)

Scheduling, especially for tail latency

Formal correctness proofs

Features like snapshots, ballooning, etc.

rust-vmm, and the container community.

Q&A

Marc Brooker
mbrooker@amazon.com
@marcjbrooker

Firecracker

aws