6.5810: Firecracker

Adam Belay <abelay@mit.edu>

Agenda for today

- Firecracker: Amazon’s serverless runtime
  - VM-based, but lighter weight than a traditional VM
  - Recall: Gvisor uses a libOS instead
- Reminders:
  - No class Wednesday
  - Work on lab 3
  - Propose a final project (alone or in a group of two)
  - Send us an email with your project idea and how you will evaluate it
  - After, we will meet and provide feedback on your proposal

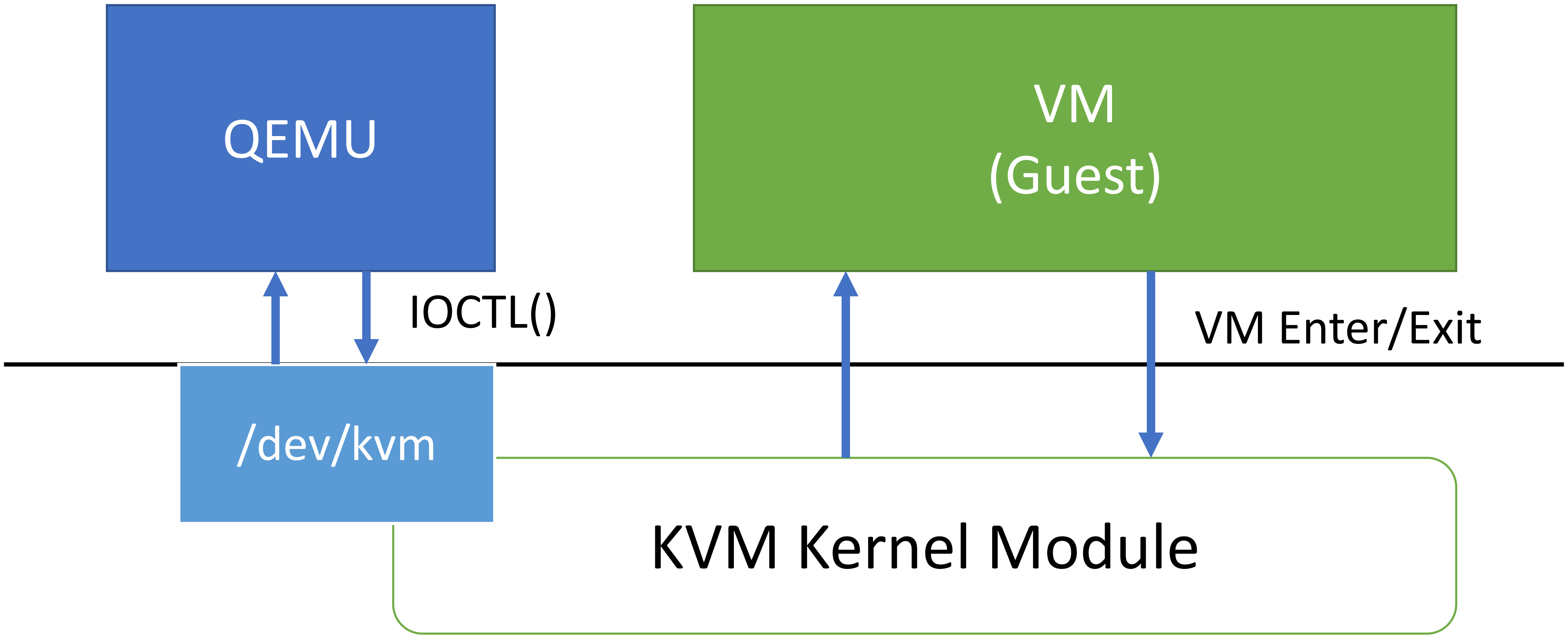
Recap: Serverless

- Goal: Eliminate packaging and management (e.g., VMs or containers)
- Instead, function as a service (FaaS):
  - Less time spent operating servers and capacity
  - Automatic scaling
  - Pay per-use of resources
  - Integration with source events / streaming data

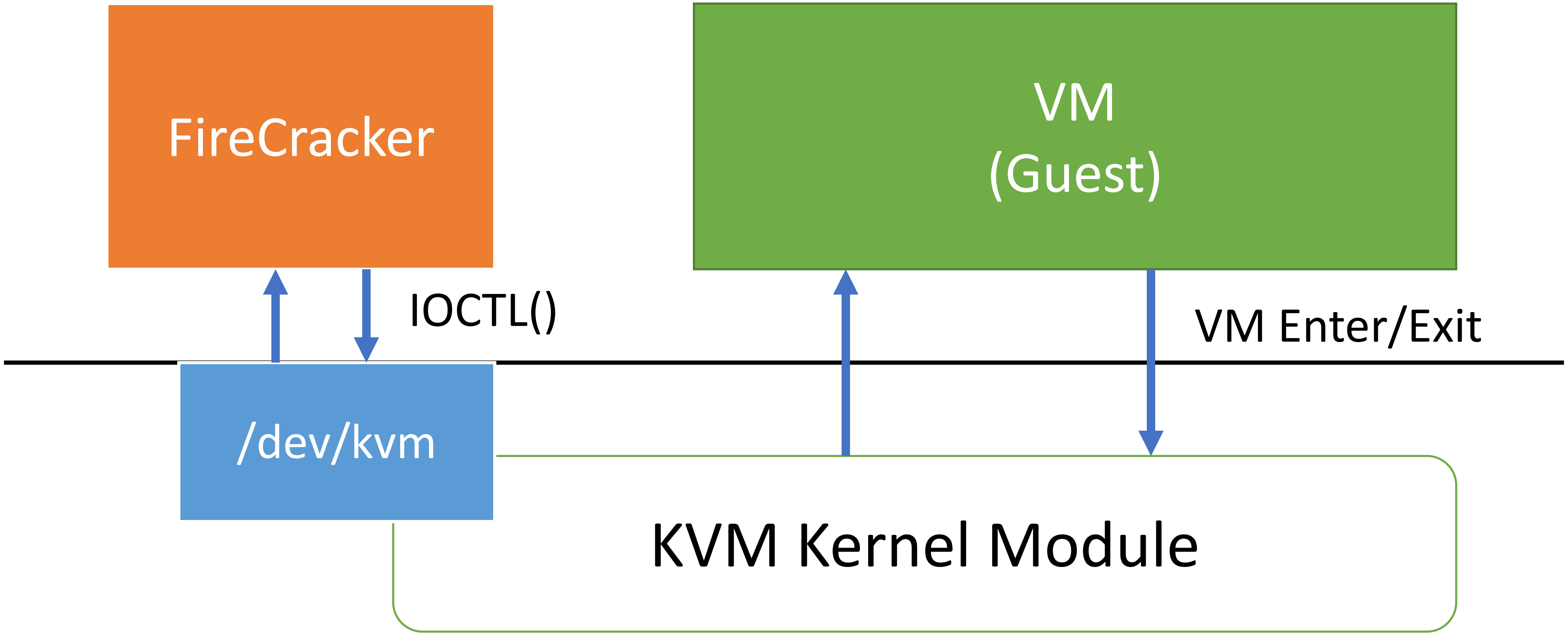
Challenge: Multitenancy

- Must isolate for security
  - One function can not violate the memory isolation of another
- Must isolate for performance
  - Must avoid the noisy neighbor problem
- Firecracker takes steps to overcome both challenges

Virtualization in Linux normally



Firecracker replaces QEMU



What does QEMU do?

- Performs VMM functions in userspace
- Provides device model (i.e., emulates virtual hardware devices)
- Manages VM creation and deletion

- Firecracker does less intentionally!
  - No BIOS, no CPU emulation, no legacy devices or PCI, no VM migration
  - E.g., it could not boot Windows
  - Why?

Amazon originally used VMs + Containers

- Originally Amazon used containers to isolate functions, and then VMs to isolate customers
- Problem: Containers sacrifice security (or compatibility)
- Problem: Hard to binpack functions onto fixed-sized VMs
- Not mentioned in paper, but significant memory overhead (density)

Firecracker’s goals

- **Isolation:** Run multiple functions on same hardware. Protect against privilege escalation, information disclosure, side channels, etc.
- **Overhead and Density:** Must handle thousands of functions on the same machine with minimal waste
- **Performance:** Functions must perform similarly to running natively. Performance must be isolated across neighbors
- **Compatibility:** Must support arbitrary Linux binaries and libraries without code changes or recompilation

More goals…

- **Fast Switching:** It must be possible to start new functions and clean up old ones quickly
- **Soft Allocation:** Must be able to overcommit resources; each function consumes only the resources it needs, not the amount it is entitled to

- Why does Amazon care about these?

Options for AWS Lambda

- Paper considers containers; virtualization; or language-based isolation
- Library OS (e.g., gVisor) also possible alternative for containers

Recap: Containers

- A composition of Linux Kernel features (not a real subsystem)
- *cgroups*: provide resource limits for memory and CPUs
- *namespaces*: provide separate UIDs, PIDs, and network interfaces
- *seccomp-bpf*: limits system calls and their arguments
- *chroot*: provides file system isolation

Problem: Isolation is a challenge: e.g., typical Ubuntu install requires 224 system calls and 52 unique ioctls

Language-based isolation

- Use a runtime system to run multiple functions in the same process
- E.g., JVM or V8 uses safe languages to provide isolation
- Each function is called an *isolate*
- See Cloudflare for a production example of this approach

Problem: Compatibility -> cannot support arbitrary binaries. Also, side-channels are a potential concern

Virtualization

- Uses Intel VT-x (or equivalent) to provide each function its own virtual hardware, page tables, and kernel
- Better security and compatibility

Many challenges: Density; VMM + Guest Kernel consume memory. Startup time; takes in the range of seconds to start VM; still large attack surface in VMM

Amazon’s plan: Improve virtualization and overcome these challenges

Debate: What approach do you think is the best? Why?

Firecracker replaces QEMU

- 50k lines of Rust code (a safe, native language)
- 96% fewer lines than QEMU (written in C)
- MicroVMs: stripped down Linux guests with minimal virtual HW
- One firecracker process handles one MicroVM

Q: What virtual devices does firecracker provide?

Firecracker’s device model

- Mainly network and block devices
- But also serial ports and PS/2 keyboard for debugging
- *Virtio* provides an interface for both network and block I/O
  - Shared memory channel between guest and firecracker
- Firecracker only exposes blocks, never the Linux filesystem… why?

Rate limiters

- Challenge: One function could monopolize I/O resources
- Solution: rate limiting
- Firecracker can be configured to enforce a max bytes/s of networking or IO/s of storage
- Firecracker uses token bucket algorithm; allowing short bursts to exceed limits
- *cgroups* still needed to enforce memory and CPU use limits

Security

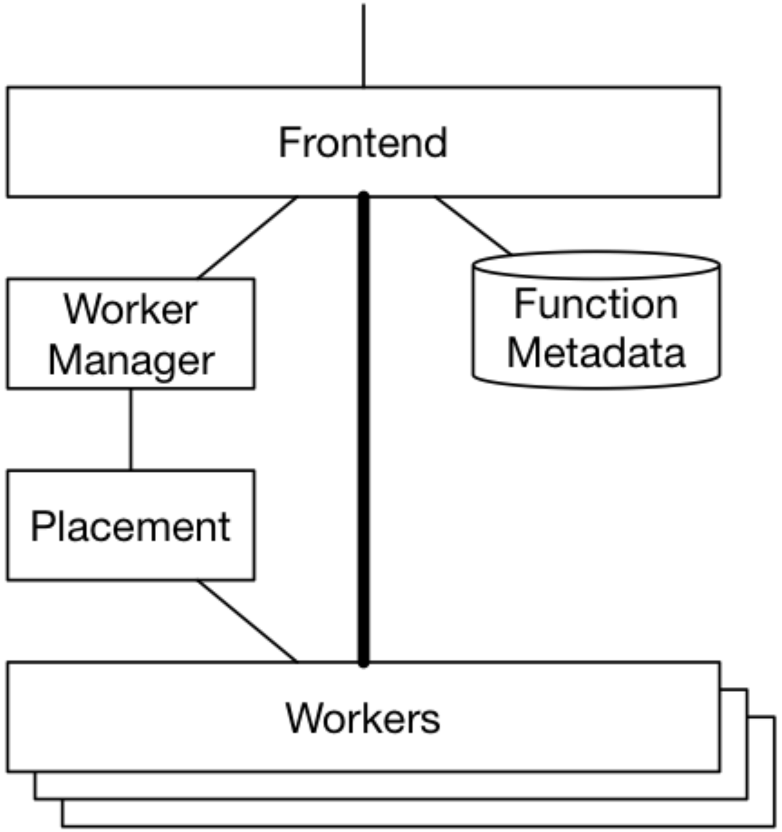
- Context: This paper was published when Spectre and Meltdown were recently discovered
- Therefore, side channels were a big concern
- Amazon’s solution:
    - Disable hyperthreads completely… why?
    - Enable Kernel mitigations: KPTI, indirect branch barriers, cache flushing, etc.
    - Downside? Higher overhead

Jailer

- Threat: What if attacker injects code into the Firecracker VMM
- Solution Jailer: Defense in depth
  - Places Firecracker in a sandbox before it boots the Guest
  - Uses seccomp-filter to restrict system calls, chroot + namespaces, and drops privileges

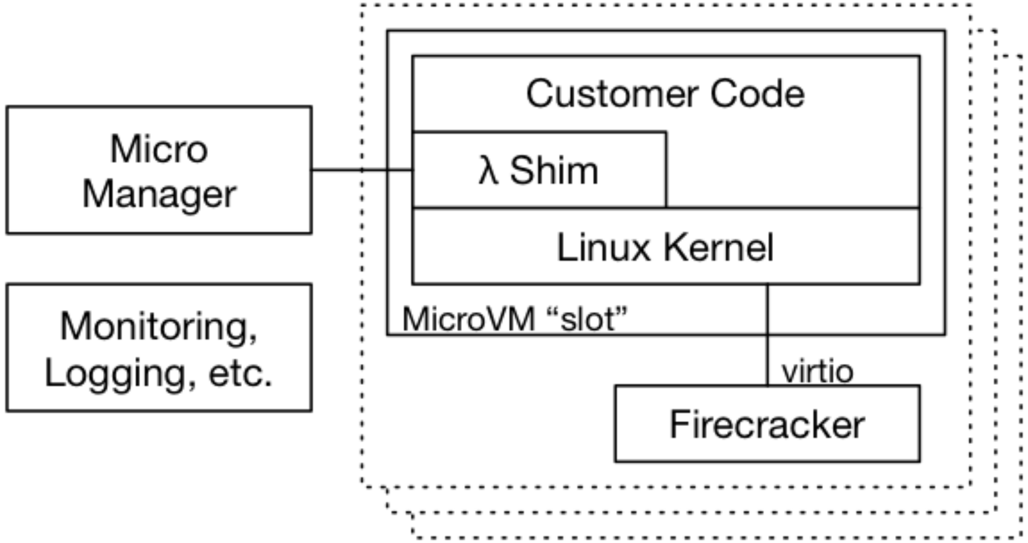
AWS Lambda service

- Built on top of functions / Firecracker
- Events sticky-routed to as few workers as possible
- Slots: Pre-loaded execution environments for functions
- MicroVM reused

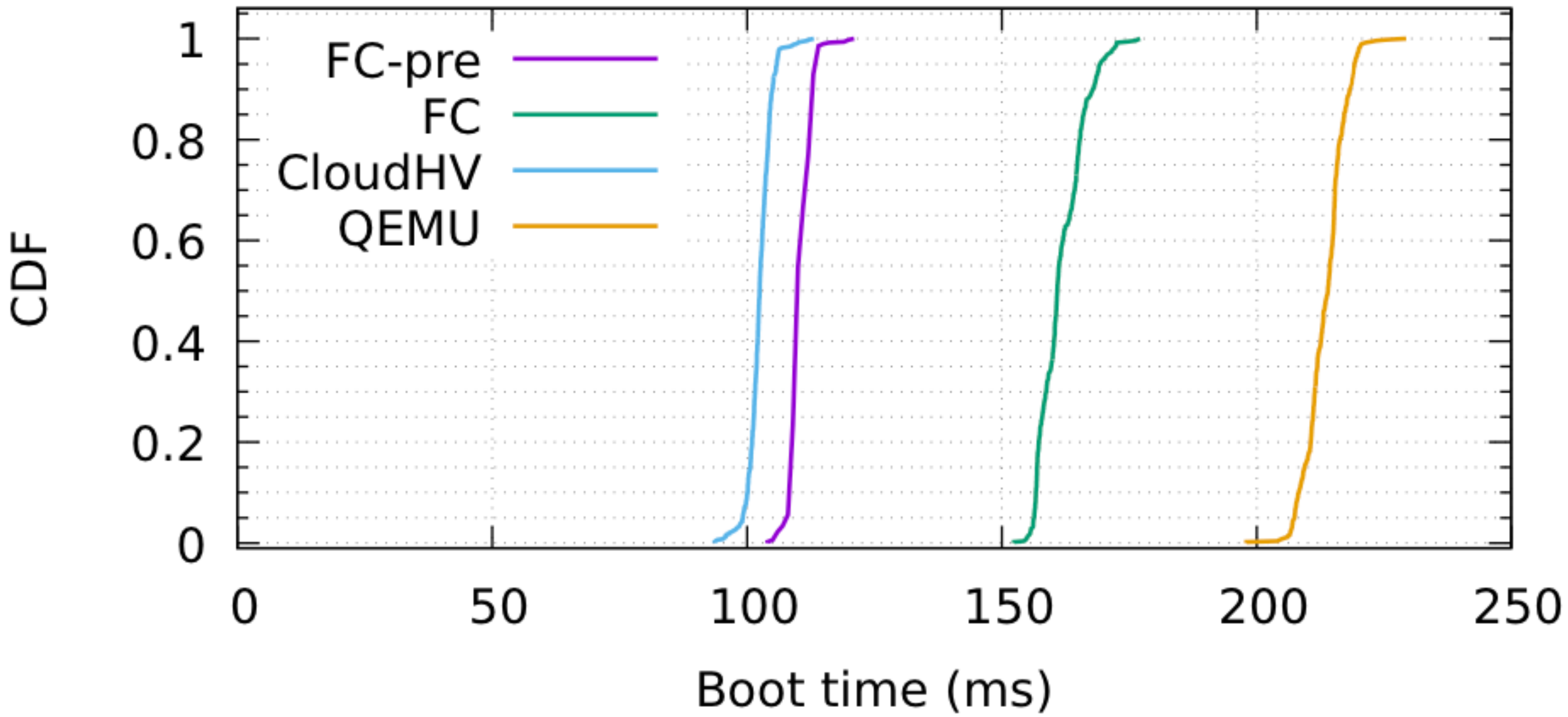


Lambda workers

- Each slot consists of a MicroVM and its Firecracker VMM instance
- Lambda shim handles control messages and launches functions
- Micro manager maintains a pool of pre-booted MicroVMs



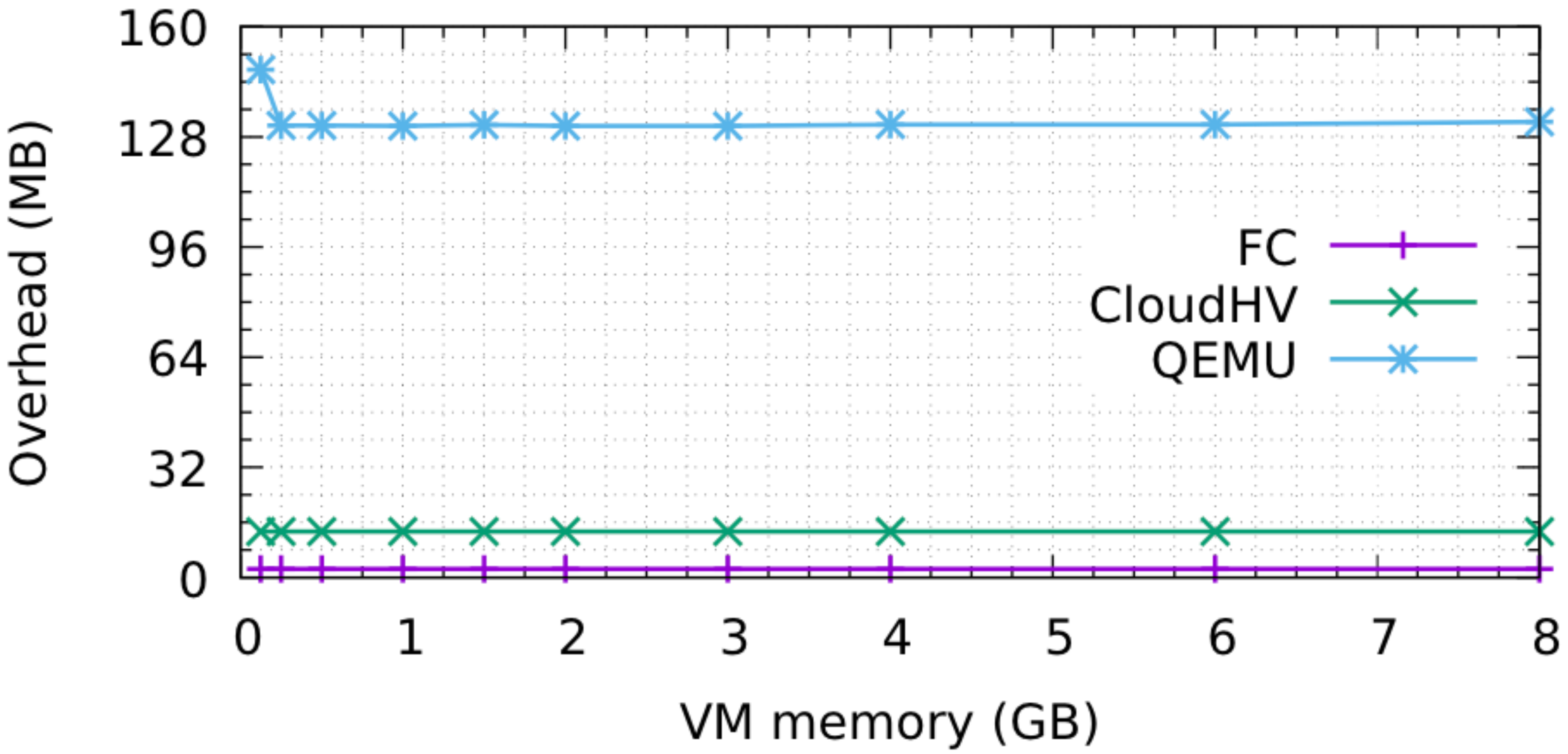
Boot time


CDF
1
0.8
0.6
0.4
0.2
0
FC-pre
FC
CloudHV
QEMU
0
50
100
150
200
250
Boot time (ms)

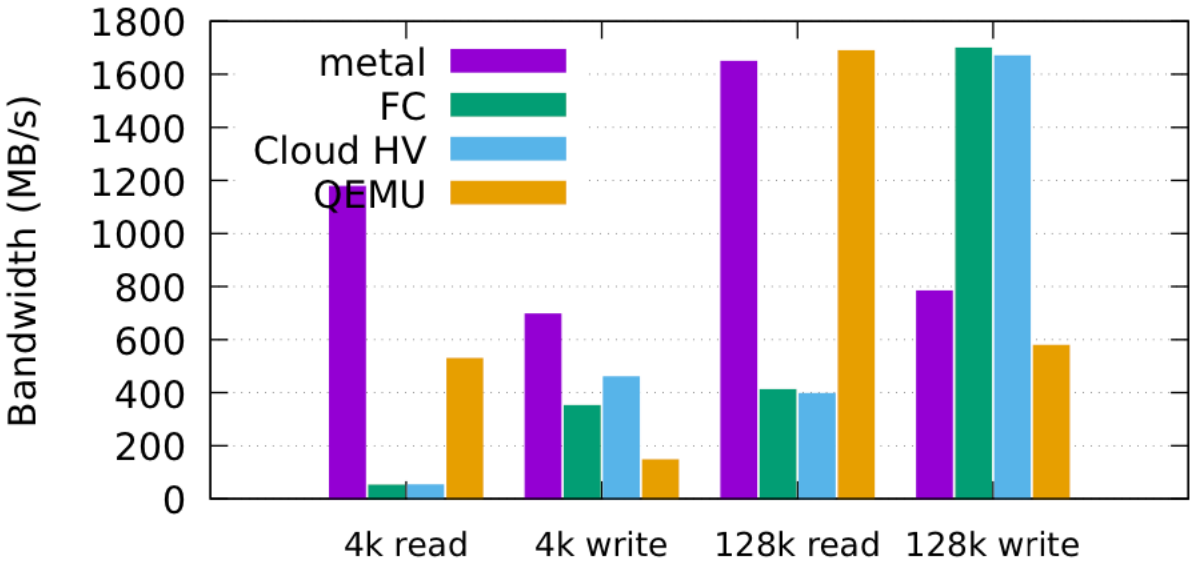
Modifying the guest improves boot time!

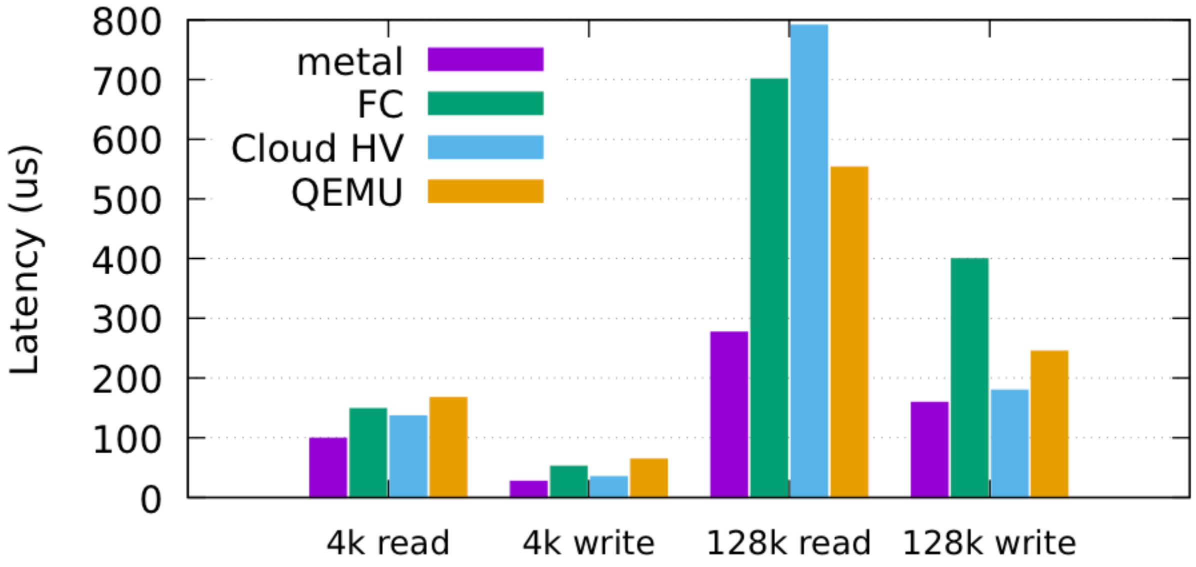
- Supporting legacy devices adds 900ms to startup
- Normally kernel image is compressed, firecracker loads uncompressed
- All kernel modules disabled; no extra hardware support included
- Disabling logging to serial console saves 70ms
- No BIOS saves boot time too

VMM process size



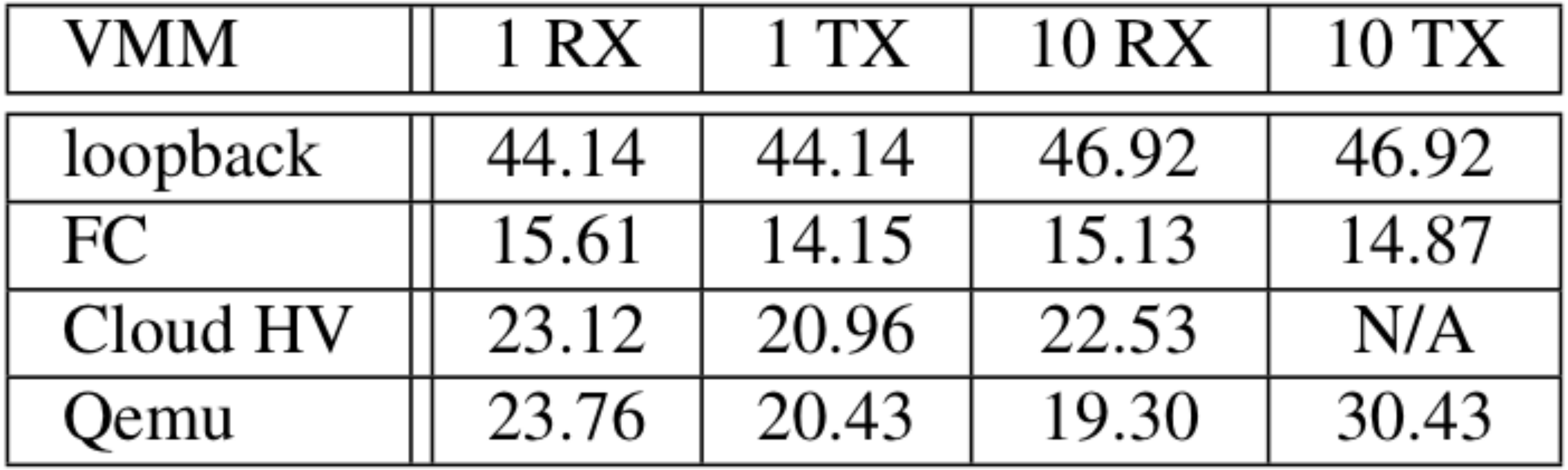
IO Performance





Hardware supports 340,000 IOPS; 1 GB/s at 4k; serial access to disk harms FC performance

Network performance



VMM	1 RX	1 TX	10 RX	10 TX
loopback	44.14	44.14	46.92	46.92
FC	15.61	14.15	15.13	14.87
Cloud HV	23.12	20.96	22.53	N/A
Qemu	23.76	20.43	19.30	30.43

Streaming throughput Gb/s for different numbers of flows and directions

Does Firecracker achieve its goals?

- **Isolation**: Yes (for security)! Virtualization reduces attack surface, jailer, and writing VMM in rust. Performance isolation is less clear.
- **Density**: Yes! Reduced memory overhead down to 3%.
- **Performance**: No! High I/O overhead observed.
- **Compatibility**: Yes! Good enough to run all customer workloads.
- **Fast switching**: Yes! 150ms start time for slots.
- **Soft allocation**: Yes! Memory and CPU oversubscribed by 10x.

Conclusion

- Firecracker is opensource; check it out!
- VM-based approach is traditionally heavy weight
  - Resolvable through new VMM (Firecracker) and stripped-down guests
- Performance is an open problem for cloud isolation mechanisms
- But Firecracker does deliver density, isolation, and fast switching
  - Research question: Can we improve these even more?
  - Do we have to sacrifice compatibility to do so?