Imperial College London

LSDS
Large-Scale Data & Systems Group

CAP-VMs: Capability-Based Isolation and Sharing in the Cloud

ARTIFACT EVALUATED usenix ASSOCIATION REPRODUCED

ARTIFACT EVALUATED usenix ASSOCIATION FUNCTIONAL

ARTIFACT EVALUATED usenix ASSOCIATION AVAILABLE

Vasily A. Sartakov

Imperial College London

http://lsds.doc.ic.ac.uk
<v.sartakov@imperial.ac.uk>

Joint work with Lluís Vilanova[1], David Eyers[2], Takahiro Shinagawa[3], Peter Pietzuch[1]

Imperial College London[1], University of Otago[2], The University of Tokyo[3]

UK Research and Innovation

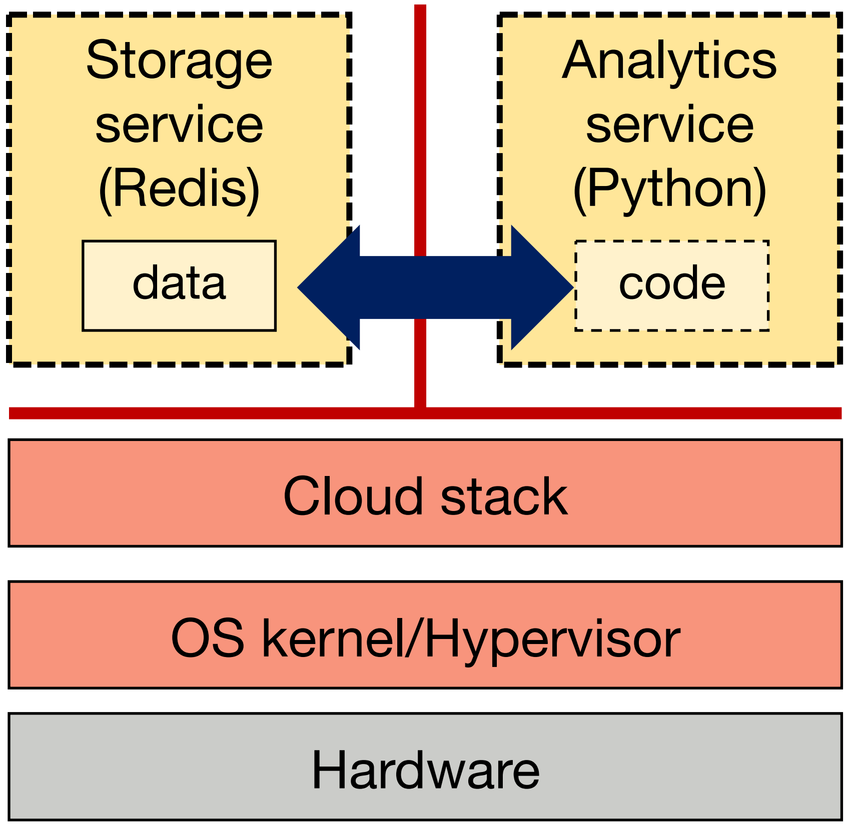
TII Technology Innovation Institute

USENIX OSDI – July 2022

Clouds: Isolation vs. Sharing

Cloud services must be **isolated** from each other and the cloud stack

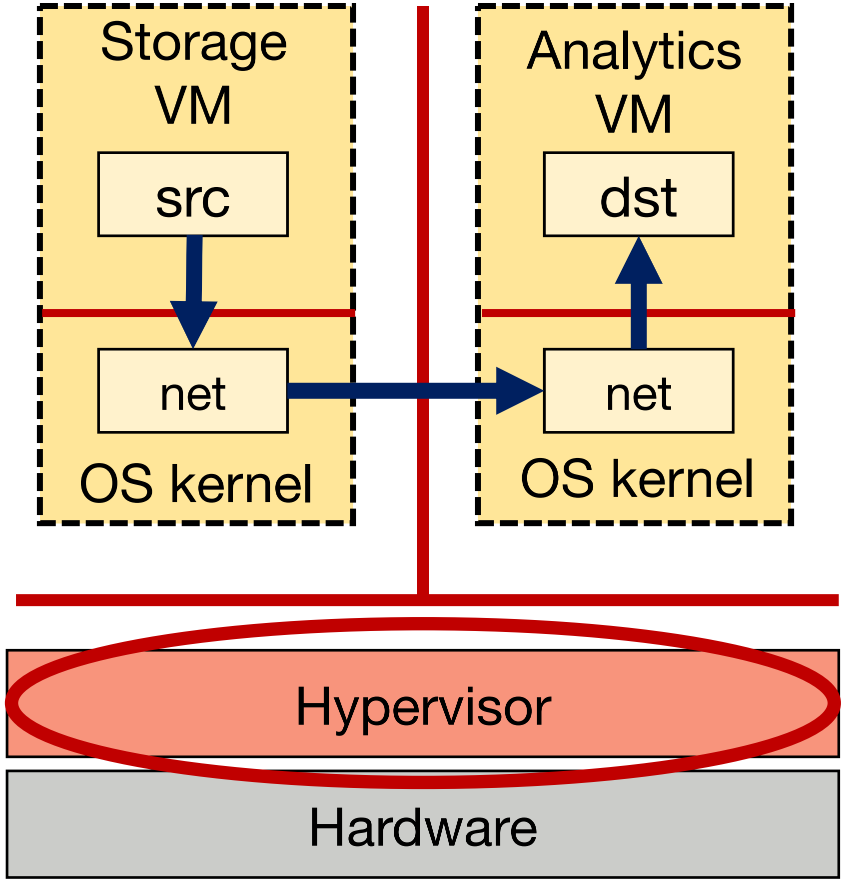
Services must **share** data efficiently by crossing isolation boundaries



VMs: Strong, Heavyweight Isolation

+ Strong isolation guarantees
+ Small(ish) trusted computing base (TCB)
  – Only consisting of hypervisor

- Network communication for sharing → TCP/IP
  – Requires data serialisation and copying
- Expensive transitions between services
  – Hypercall ≈ 50 × syscall

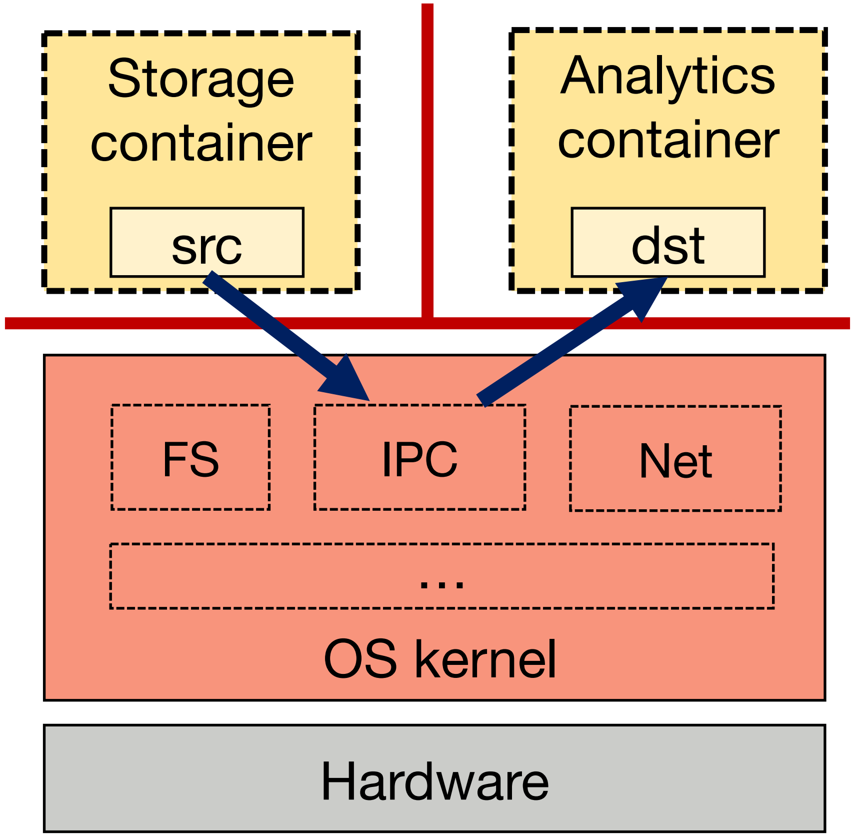


Containers: Weak, Lightweight Isolation

+ Lightweight OS namespace isolation
+ Efficient IPC mechanisms

- Large TCB due to shared OS kernel
  – Shared kernel has much unnecessary functionality

→ Challenge: efficient data sharing with small TCB



VMs & Containers: The MMU Tax

Memory Management Unit (MMU) is privileged entity

– Intermediary (kernel) always involved in IPC → Shared TCB, syscalls/hypercalls

MMU shares data at page granularity

– Sharing may expose extra data

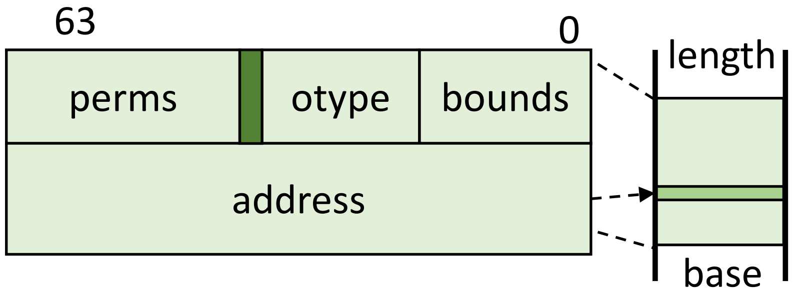
Can we use another technology for isolation and sharing?

→ CHERI: isolation at byte granularity, low dependency on the kernel

CHERI Capabilities

Fat pointers protected by hardware:

- base + length, cursor
- permission, tag
- byte-granularity*



Fine-grained isolation

Limited dependency on OS kernel

Available: Arm CHERI Morello Boards (Armv8)

Capabilities can be created only from capabilities

- Using cap-aware instructions, but not the intermediary

Challenges for Cloud Stacks with Hardware Capabilities

What would a cloud stack look like if hardware provided **efficient** mechanisms to share **arbitrary-sized** memory regions between otherwise **isolated** entities?

Challenges:

C1. Support capability-unaware software

C2. Provide small-TCB OS functionality

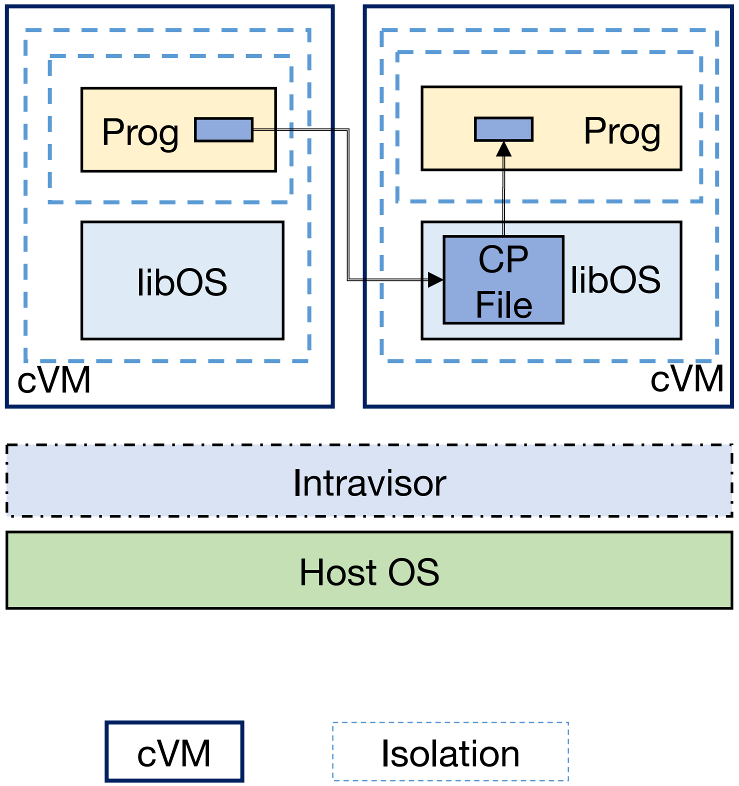
C3. Enable efficient capability-based IPC interfaces

cVM: Intra-Process VM-like Abstraction

1. Support cap-unaware software
→ Isolated execution of native applications

2. Small shared TCB
→ Private namespaces by library OSs

3. Cap-based IPC interfaces
    `CP_File`: efficient data sharing
    `CP_Call`: remote code invocation



C1. Isolation/Sharing for Legacy Cloud Apps?

CHERI:

Native ABI: cap-unaware code

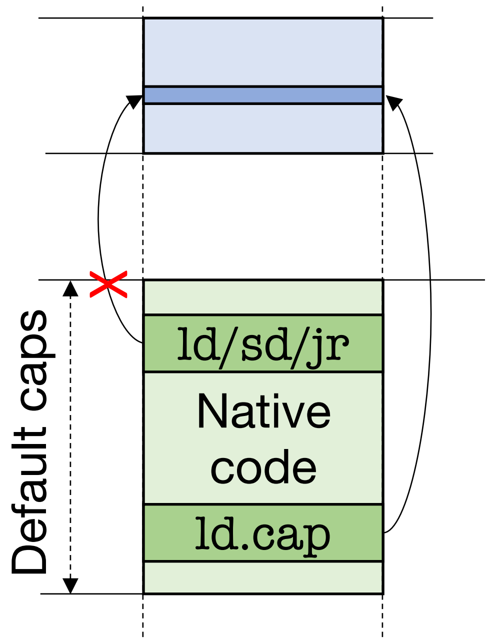
Pure-capability ABI: requires porting

Hybrid-capability ABI: native + cap-aware code

Fine-grained compartmentalisation:

- Cap-unaware instructions constrained by **default** caps
- Hybrid code can use capability-aware instructions

→ Can be used for isolation and IPC primitives



Support for Native Software

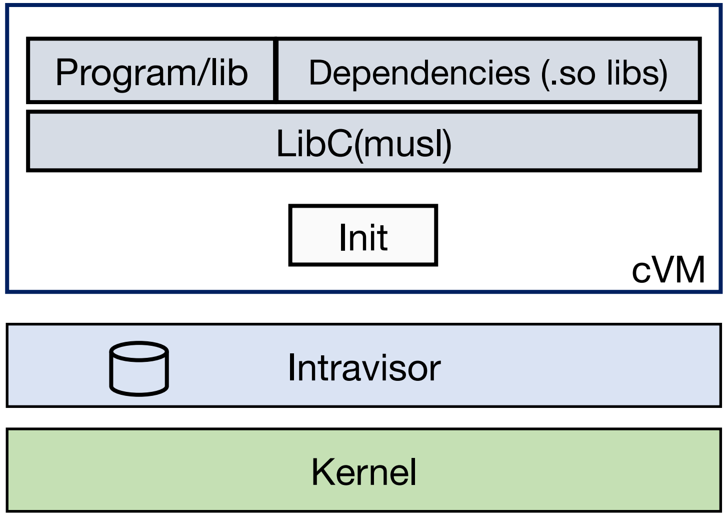
Goals:

- POSIX environment
- Cloud deployment model (e.g. Docker or VMs)

→ Service for cVM shipped as disk image

- Native cap-unaware PIE binaries
- Compatibility: C standard library (musl libc)

→ Intravisor allocates cVM, loads Init and disk



C2. Small-TCB OS Functionality

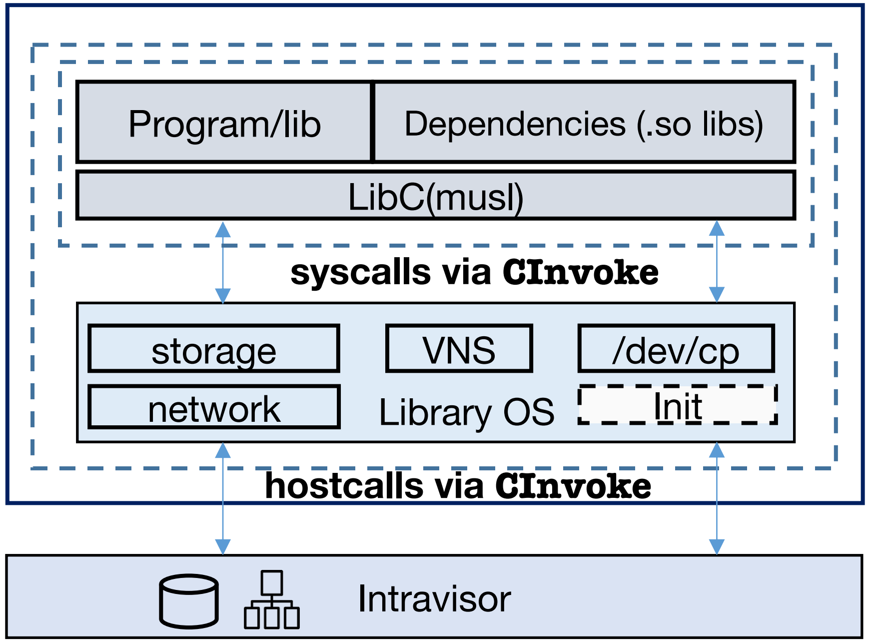
Goals:

- Necessary OS components
- Small attack surface

→ Private LibraryOSs provide OS functionality

→ Intravisor provides time/network/disk I/O

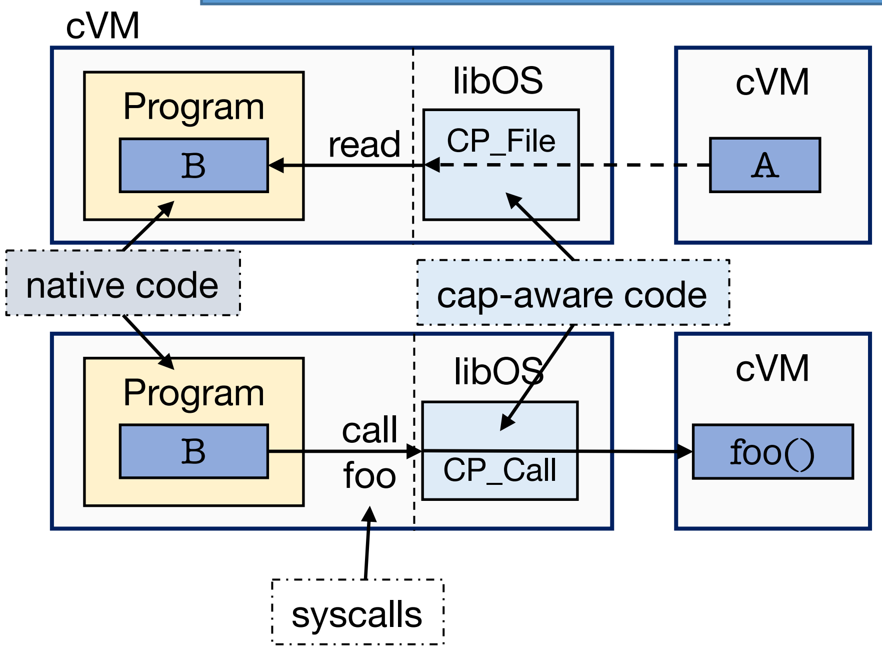
→ Nested isolation layers



C3. IPC Interfaces Using Capabilities

Data sharing primitives efficient if:

- Non-shared and without intermediary on critical path
- Well-known API (POSIX)
- Usable by cap-unaware code



`CP_File` – read/write remote memory at byte granularity using caps

`CP_Call` – call function in cVM

`CP_Stream` – stream-oriented IPC interface

CAP-VM Prototype

Platforms:

- CHERI RISC-V64, QEMU, AWS F1 (agfi-026d853003d6c433a)
- CheriBSD (host), LKL v4.17 with musl v1.2.1 (cVMs)
- SiFive HiFive Unmatched (No CHERI, but multi-core)

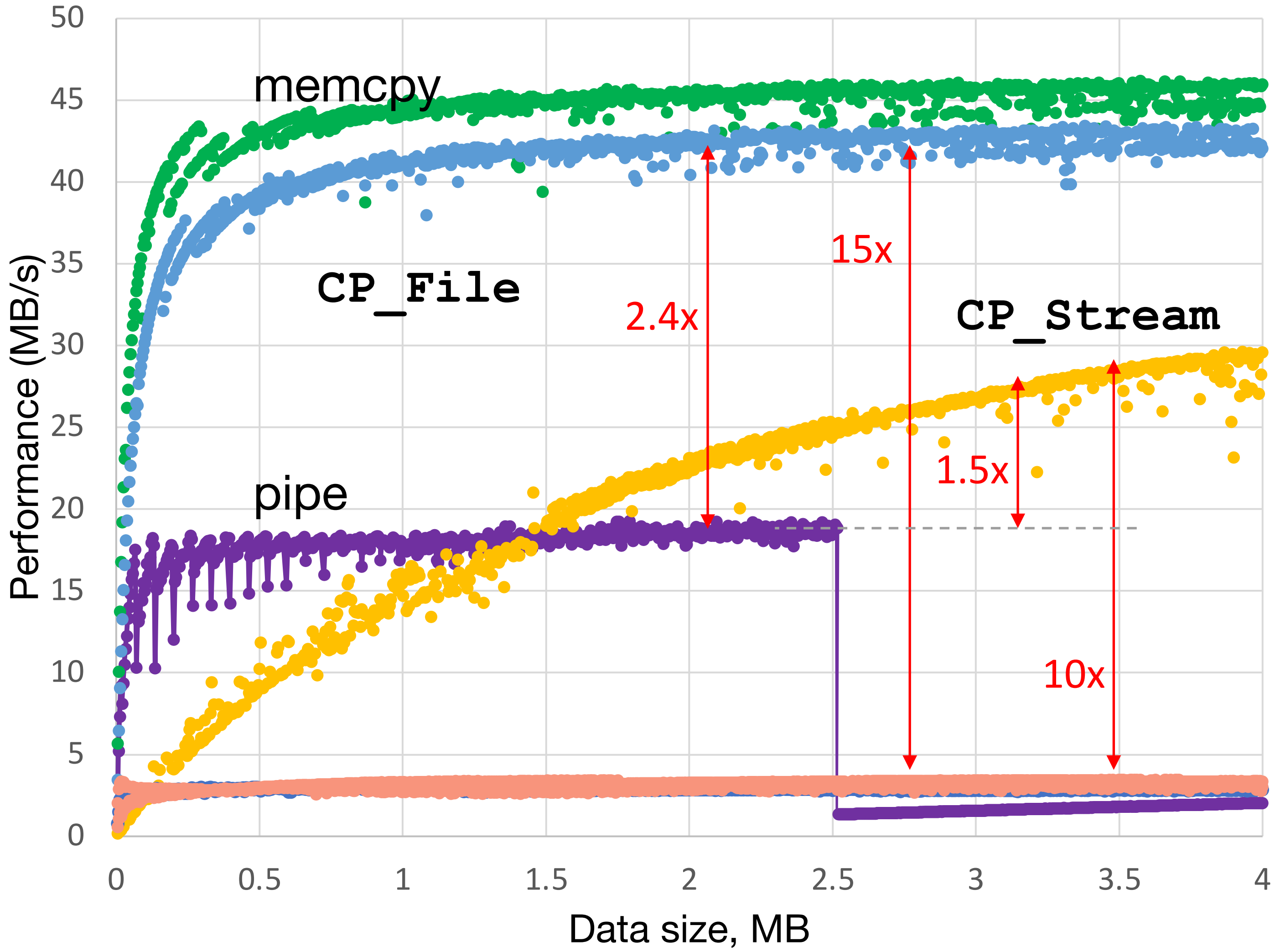
Application and services (in the paper):

- Redis, data-processing utilities, Python3 with modules, SQLite benchmarks
- Multi-tier microservice (NGINX with API gate, Redis (SiFive only))

Evaluation question: Performance of cVM IPC primitives?

- Basic: memcpy, mmap+memcpy
- cVMs: **`CP_File`**, **`CP_Stream`**
- FreeBSD: pipe, Unix, TCP sockets

Comparing with IPC Mechanisms



`CP_FILE` vs. memcpy:
- 6% slower

`CP_Stream` faster (1.2 MB+)
- Privileged execution

Unix, TCP, mmap+memcpy:
- Less than 2.4-3.6 MB/s

Processes: 1.6 MB/s max

Conclusions

Small-TCB isolation with efficient sharing in clouds hard:

- Containers → large shared TCB with relatively fast IPC mechanisms
- VMs → small TCB with slow IPC mechanisms

CAP-VMs provide VM-like abstraction using hardware capabilities:

- Secure isolation at byte granularity using memory capabilities
- Controlled shared TCB by private library OS
- Efficient data sharing using capability-based IPC primitives

Source code: http://github.com/lsds/intravisor

Thank You — Any Questions?

Vasily A. Sartakov
v.sartakov@imperial.ac.uk

LSDS
Large-Scale Data & Systems Group