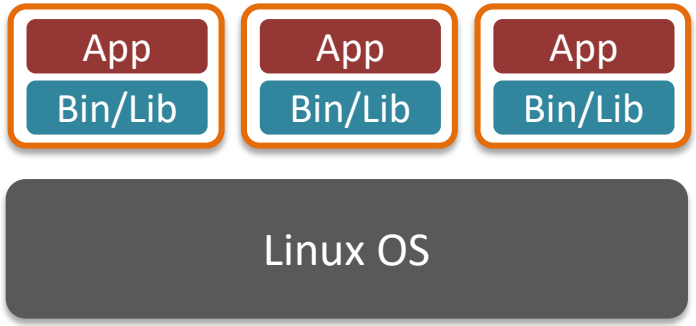
Library OS is the New Container.

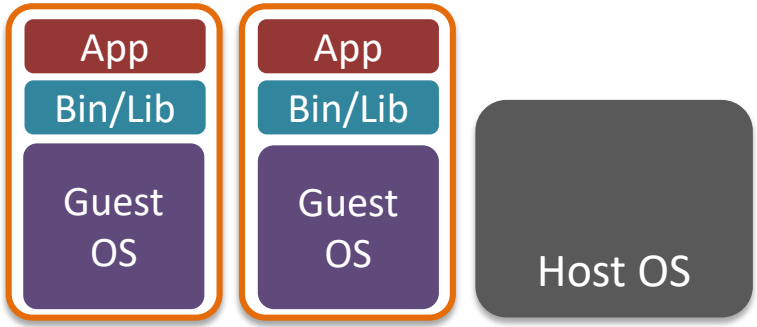
Chia-Che Tsai / RISE Lab @ UC Berkeley

Talking Points

- In a nutshell, what is LibOS?
- Why you may want to consider LibOS?
- What's our experience?
- Introducing **Graphene: an open-source Linux libOS**

Containers vs VMs





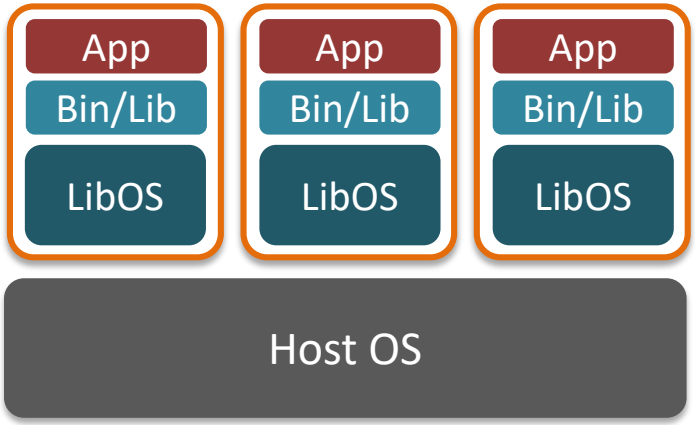
Containers

- Host-dependent
- Light resources
- Binary/library compatibility
- Userland isolation

VMs

- Host-independent
- Heavy resources
- System ABI compatibility
- Kernel isolation

LibOS: Pack Your OS with You



- A part of the OS as a library
- Per-application OS isolation
- Can be light-weight
- Can be compatible as system ABI
- Can be host-independent

Depend on how you implement the libOS

LibOS and Friends

- Drawbridge

BIZ & IT —

How an old Drawbridge helped Microsoft bring SQL Server to Linux

There are certainly risks involved, but a clever research project makes it all possible.

PETER BRIGHT - 12/16/2016, 9:00 AM

- Unikernels

A new riff on containers

TechBeacon

Containers 2.0: Why unikernels will rock the cloud

- Google gVisor

GOOGLE CLOUD PLATFORM

Open-sourcing gVisor, a sandboxed container runtime

Graphene: An Open-source Linux LibOS

- An ambitious project to build an ultimate libOS

As **host-independent** as it can be
(Maybe even more than VMs - Explain later)

As **light-weight** as it can be

As **securely isolated** as it can be

https://github.com/oscarlab/graphene

A Research Prototype Turned Open-source

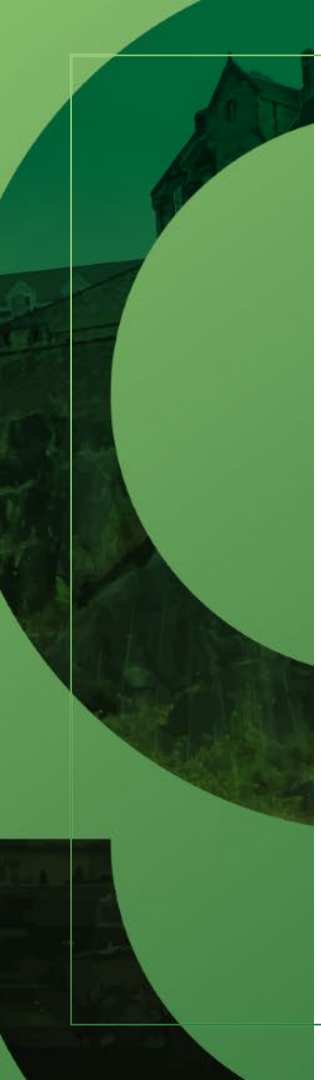
2014 — Graphene released as an artifact

2016 — First to support native Linux applications on **hardware enclaves (Intel SGX)**

Today — Working toward code stability and community building

Main contributors:

Intel Labs, Golem, Invisible Things Lab, Fortanix



Getting Compatibility For Any Host

Compatibility Goal of Graphene

- **Running a Linux application on any platform**
  - Off-the-shelf binaries
  - Without relying on virtualization

Linux Compatibility is Hard

- Imagine implementing 300+ system calls on any host
  - Flags, opcodes, corner cases (see “**man 2 open**”)
  - Namespaces and idiosyncratic features
  - IOCTL() and pseudo-filesystems
  - Architectural ABI (e.g., thread-local storage)
  - Unspecific behaviors (bug-for-bug compatibility)

Dilemma for API Compatibility

Cannot achieve all these properties at the same time

Rich of features

Having a rich set of APIs defined for application developers

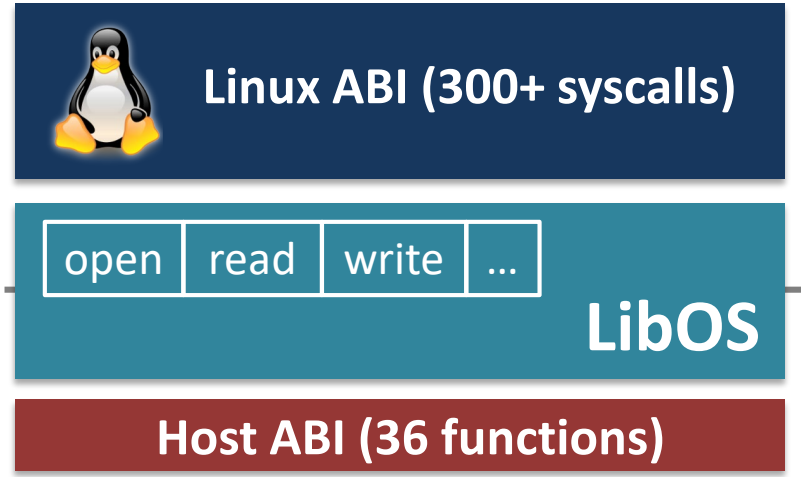
Ease of porting

Being easy to port to other platforms or maintain in new versions

Compatibility

Being able to reuse existing application binaries as they are

Solving the Dilemma



Rich features

Backward-compatible

Easy to port

Backward-compatible

Host options:

Linux Kernel Versions

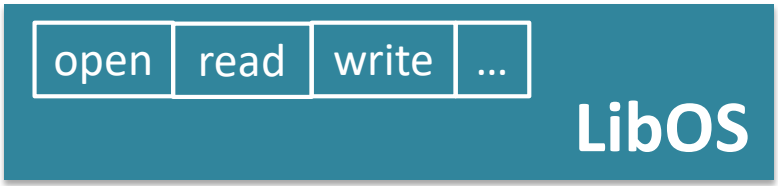
BSD

OSX

Win

Intel SGX

Components of Graphene



Host ABI (36 functions)

Platform Adaption Layers (PAL):

Linux PAL | **BSD PAL** | **OSX PAL** | **WIN PAL** | **SGX PAL**

- System calls implemented from scratch (one-time effort)
- Designed for portability
  - Short ans: **UNIX**
  - Long ans: a common subset of all host ABIs
- The only part that has to be ported for each host

How Easy is Porting Our Host ABI?

BSD PAL

(Released)

2 MS students
x term project

WIN PAL

(Experimental)

Problem:
mmap() vs MapViewofFile()

1 MS students
x 3 semesters

OSX PAL

(Experimental)

Problem:
can't set FS register!

1 MS students
x 2 semesters

SGX PAL

(Released)

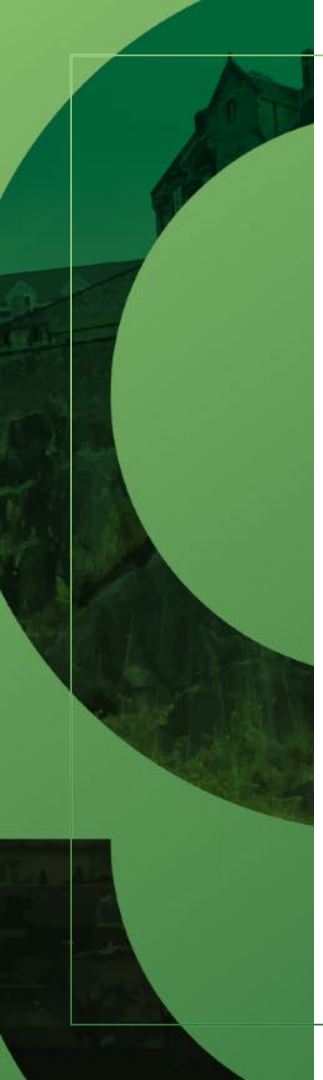
1 PhD student (Me)
x 3 months

Not all straightforward, but we learned where the pains are.

Summary

How does Graphene gain compatibility?

- A LibOS to implement Linux ABI; painful, but reusable
- Host ABI is simple and portable
- Porting a PAL = Porting all applications

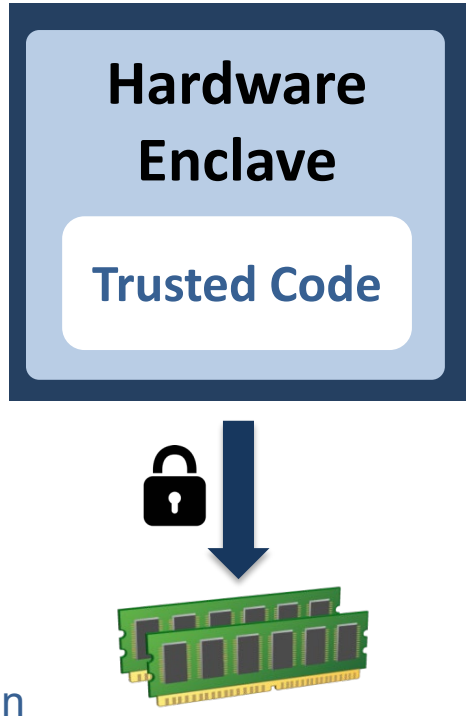


Porting to Intel SGX

(A Uniquely-Challenging Example)

What Is Intel SGX?

Software Guard Extensions



Program integrity

CPU attestation

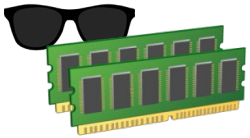
Data stay encrypted on DRAM

Available on Intel 7+ gen
E3 / i5 / i7 CPUs

What Can Intel SGX Do?

- Assume the host is untrusted

Hacked OS or hypervisor

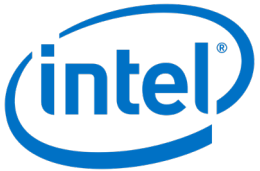


Interposed DRAM

Modified Devices

Compromised Admins

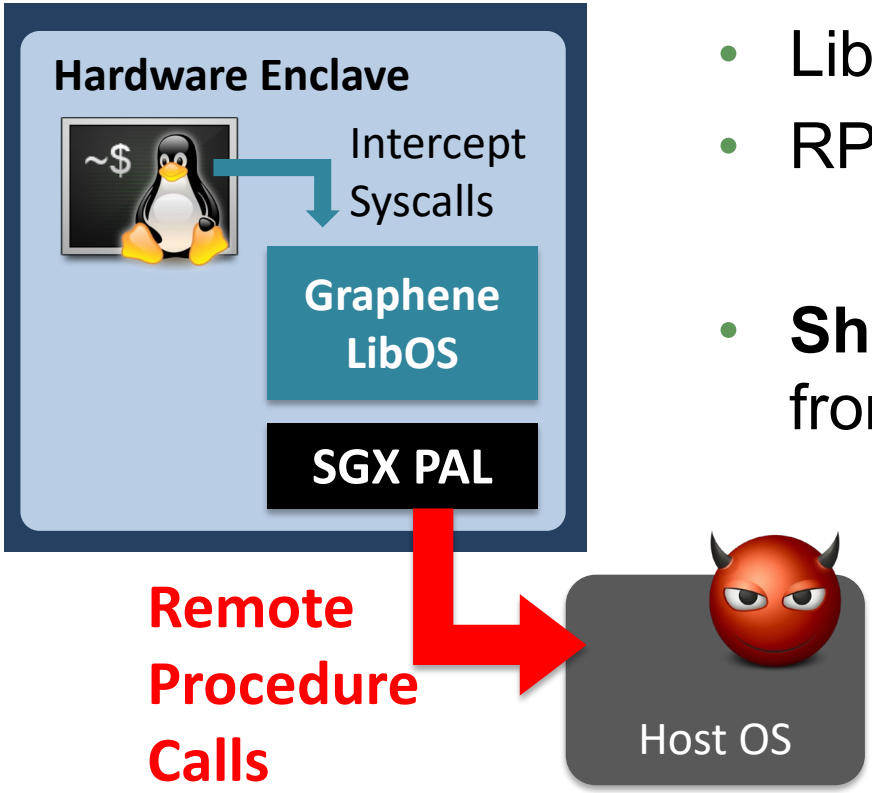
- You only have to trust **your software** and



As a Platform, SGX Has Many Restrictions

- Limited physical memory (93.5MB)
- Only ring-3 (no VT)
- **Cannot make system calls**
  (for explicit security reasons)

Serving System Calls Inside Applications

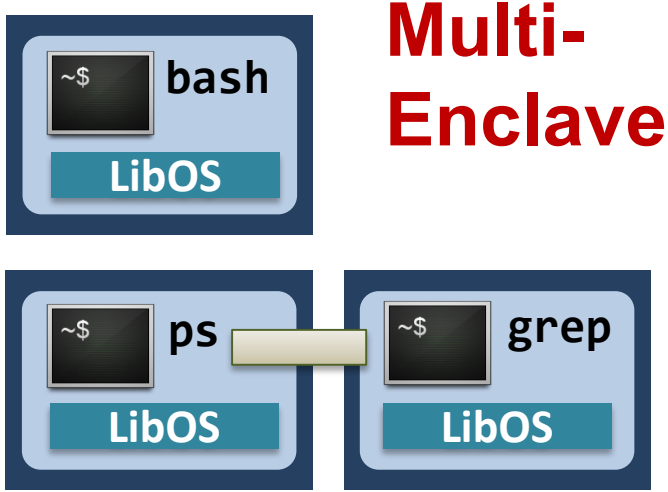


- LibOS absorbs all system calls
- RPCs for I/O & sched

- **Shielding**: verify RPC results from untrusted hosts

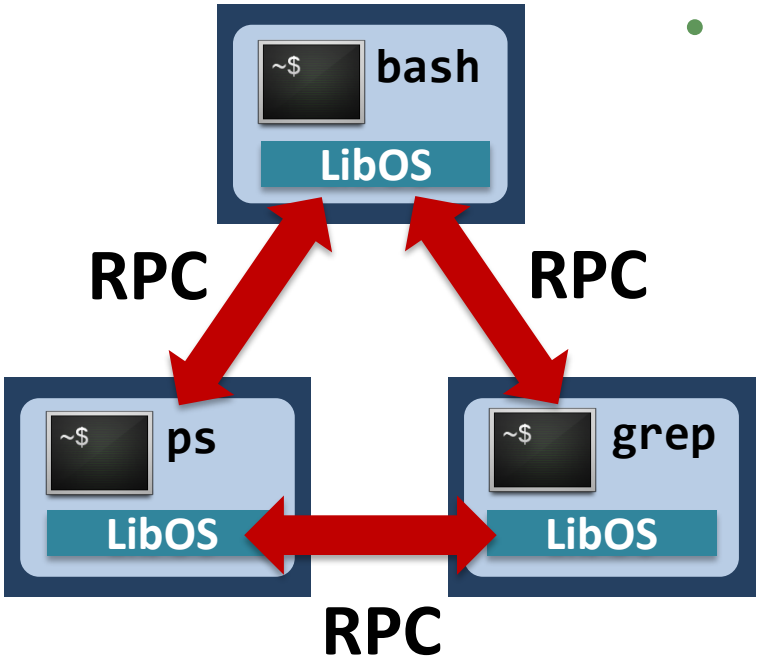
Sharing Memory is a Big Problem

Linux is multi-proc:
servers, shells, daemons



- Enclaves can't share memory
- Why not single-enclave?
  - Position-dependent binaries
  - Process means isolation
- LibOSes need to share states:
  - Fork, IPCs, namespaces

Assumes No Shared Memory

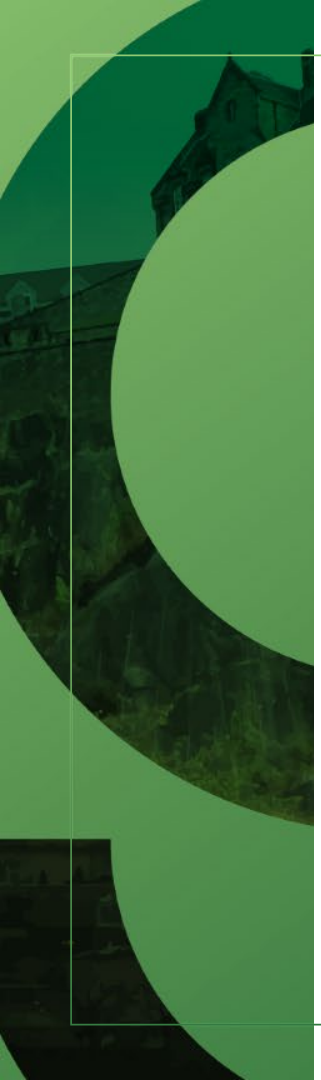


- Basically a distributed OS w/ RPCs
  - Shared namespaces
  - Fork by migration
  - IPCs: signal, msg queue, semaphore
  - No System V shared mem

Summary

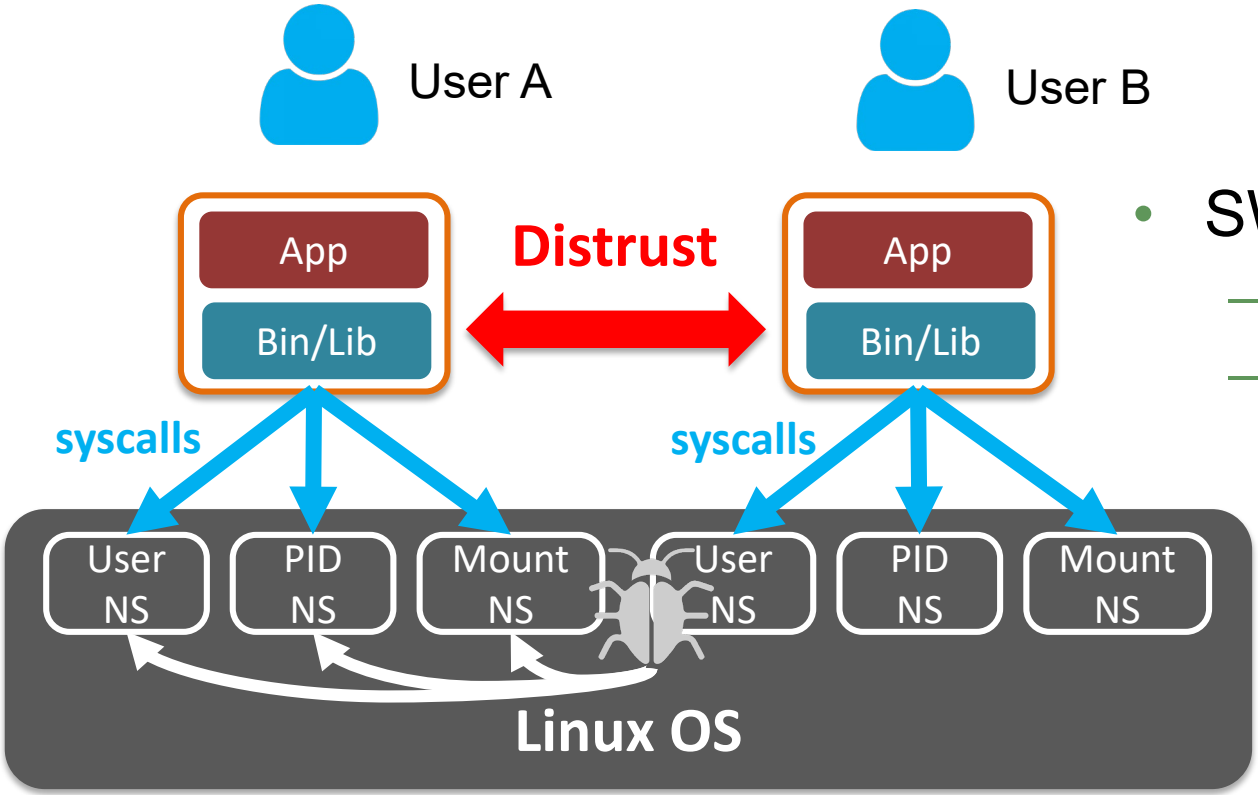
Why does Graphene work on SGX while containers/VMs don't?

- LibOS serves APIs on a flattened architecture
- For multi-proc: Graphene keeps distributed OS views without shared memory



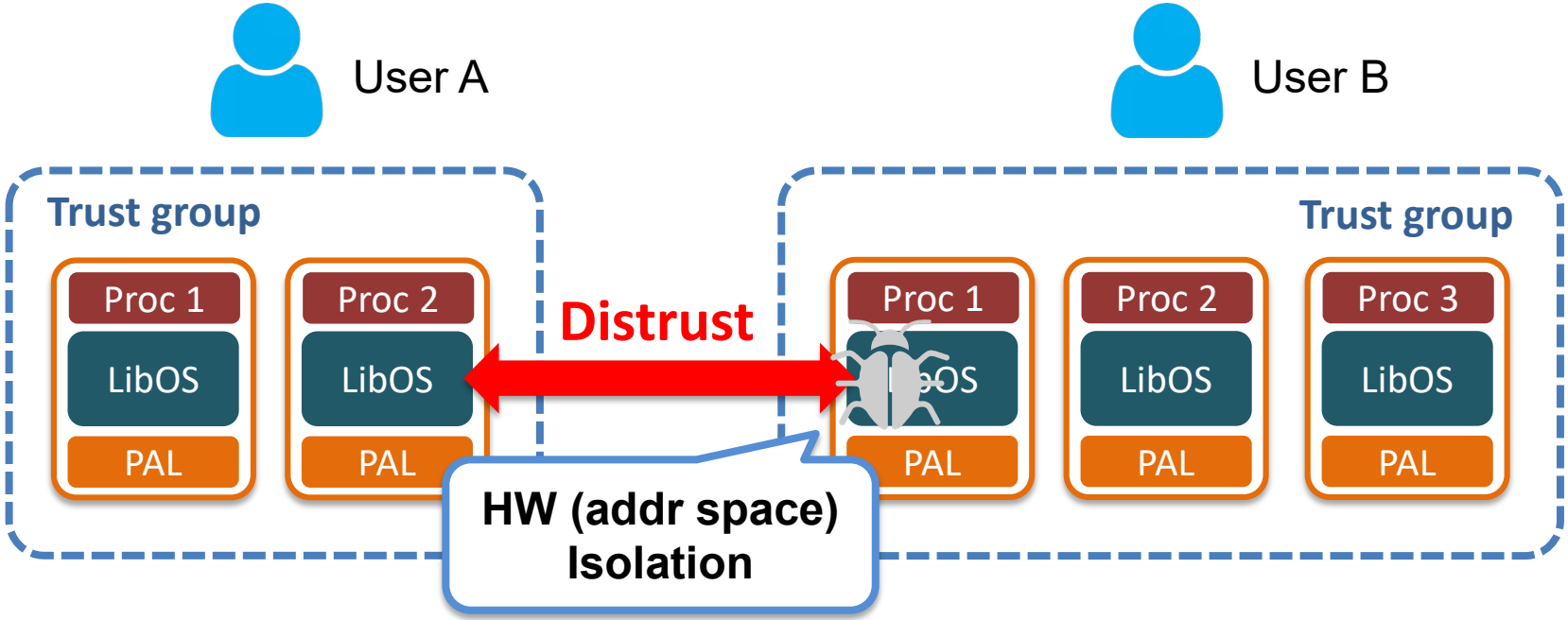
Security Isolation & Sandboxing

Mutually-Distrusting Containers



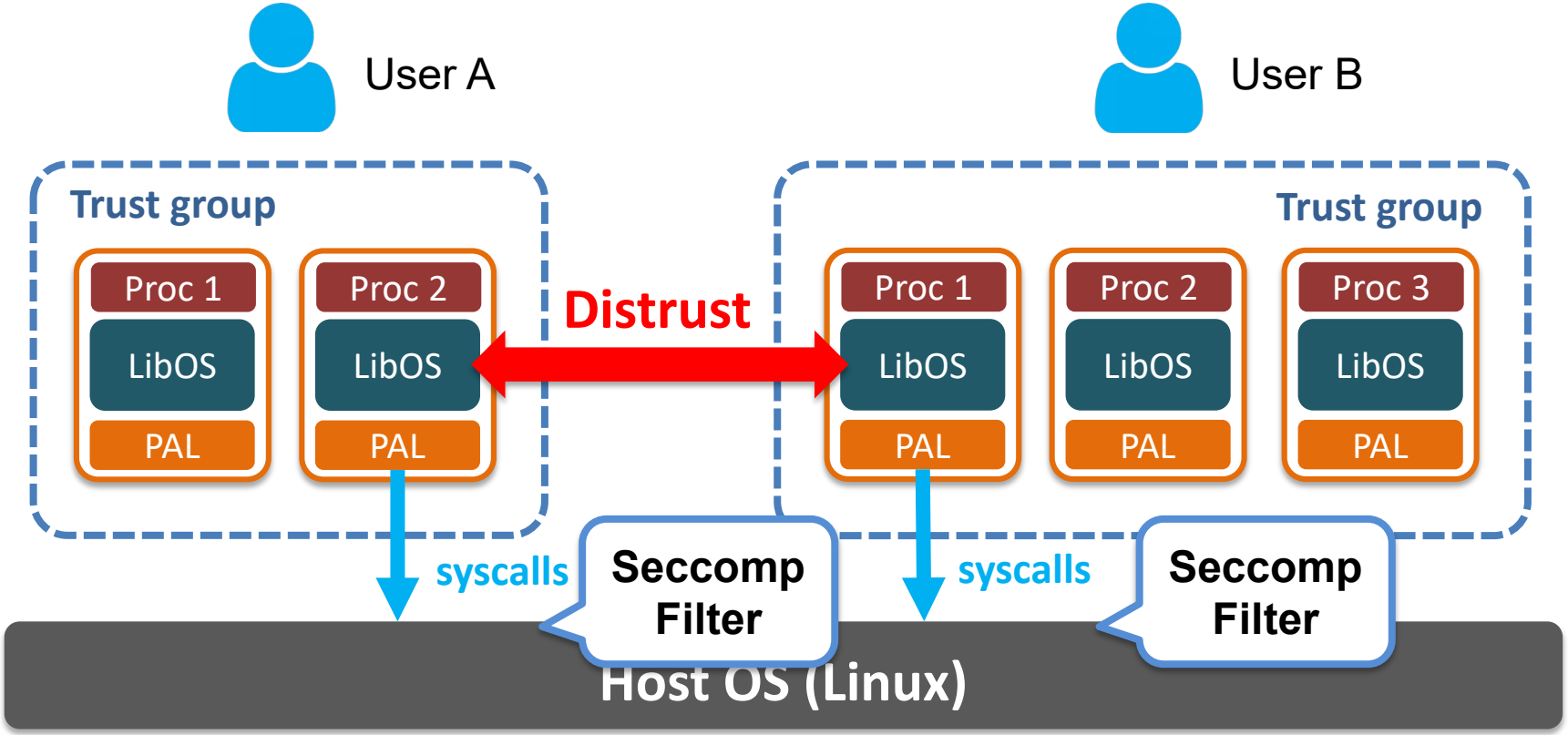
- SW technique
  - No HW isolation
  - Can’t stop kernel bugs

Mutually-Distrusting LibOS Instances



- IF syscalls are served only inside libOS, no attack can occur

Protecting Host OS From LibOS



Default Seccomp Filter: Graphene vs Docker

- What's used most of the time

Graphene:

https://github.com/oscarlab/graphene/blob/master/Pal/src/security/Linux/filter.c

```
SYSCALL(__NR_accept4,  ALLOW),
SYSCALL(__NR_clone,    JUMP(&labels, clone)),
SYSCALL(__NR_close,    ALLOW),
SYSCALL(__NR_dup2,     ALLOW),
SYSCALL(__NR_exit,     ALLOW),
...
```

48 syscalls allowed

Further checks syscall flags

Docker:

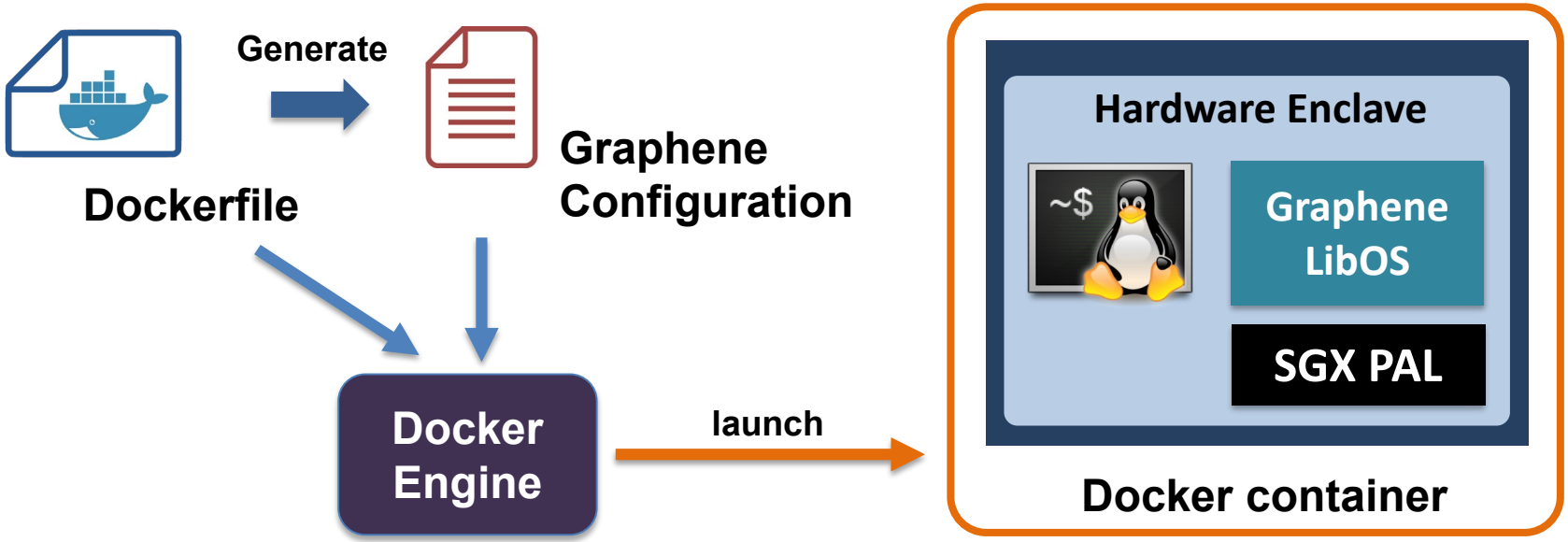
https://github.com/moby/moby/blob/master/profiles/seccomp/default.json

```
“names": [
            “accept",
            "accept4",
            "access",
            ...
        ],
"action": "SCMP_ACT_ALLOW",
```

307 syscalls allowed

Not enough? Try Graphene-SGX Containers

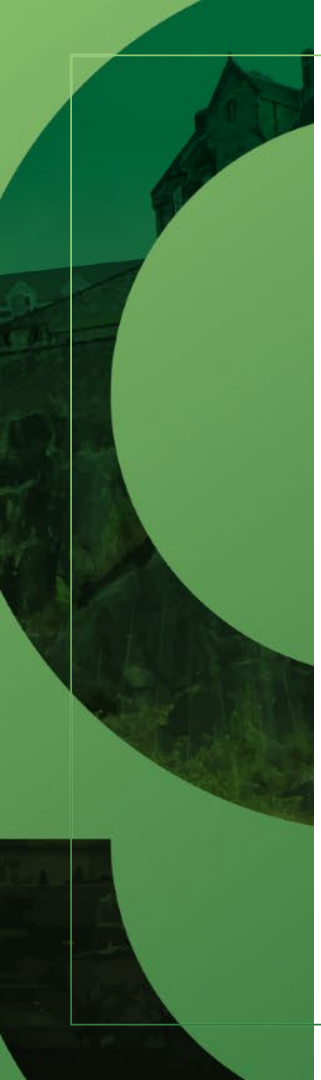
- Graphene-SGX as a backend for Docker



Summary

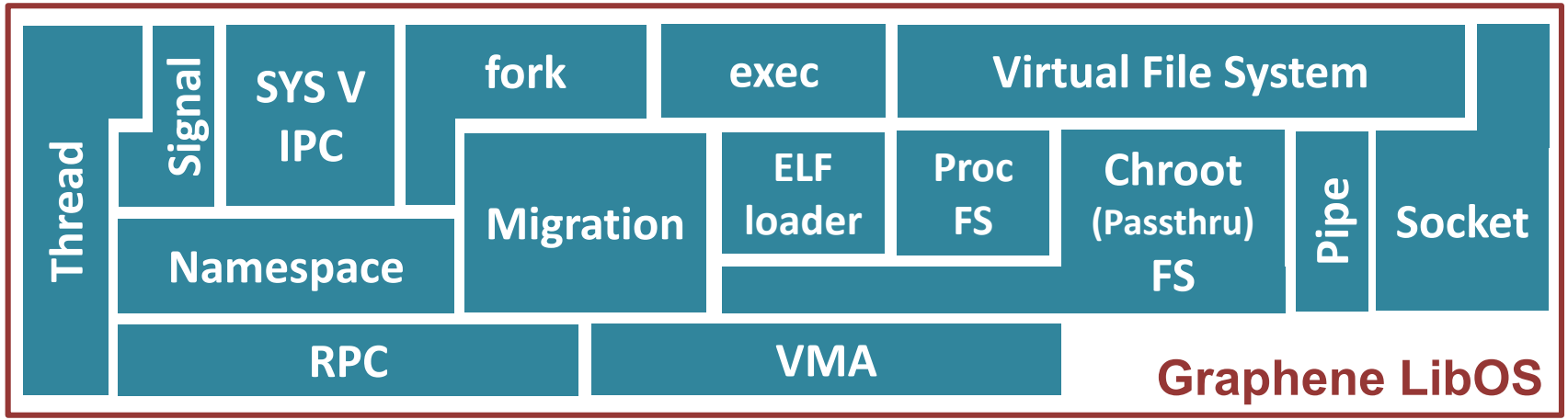
Why is Graphene better at sandboxing than containers?

- System calls inside libOS are naturally isolated
- Small default system call footprint (48 calls)
- **Graphene-SGX containers:**
  Mutual protection between OS and applications



Functionality & Performance

Current LibOS Implementation



145 / 318 system calls
Implemented (core features)

34 KLOC
Source code

909 KB
Library size

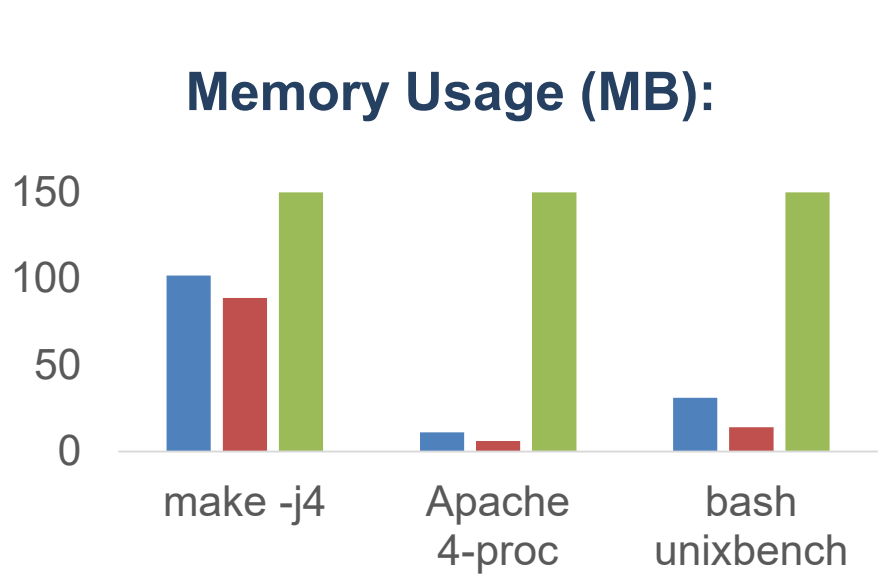
Tested Applications

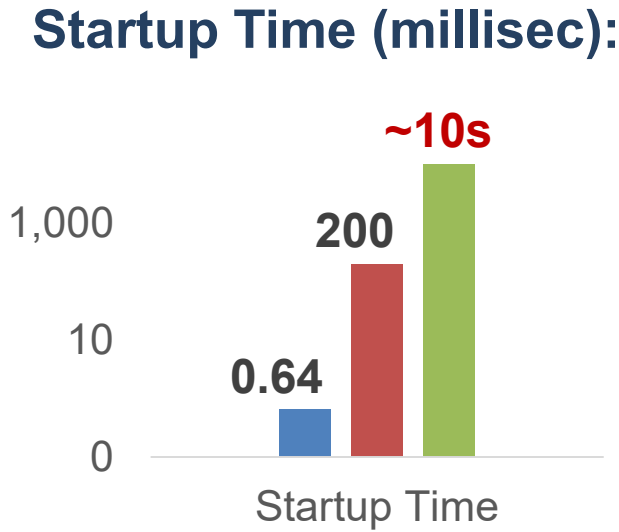
… and more.

See examples on:  **https://github.com/oscarlab/graphene**

Memory Usage & Startup Time

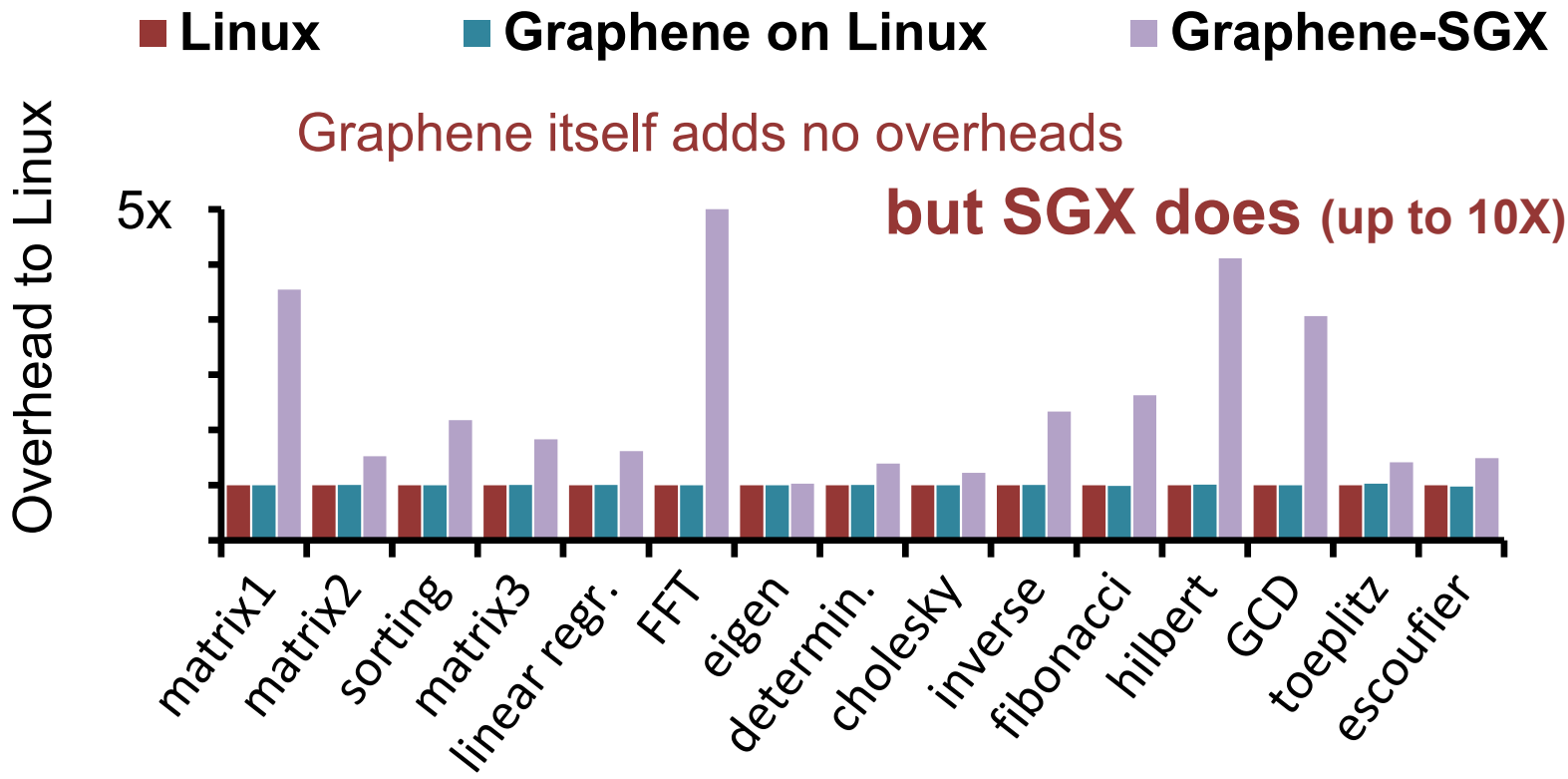
Graphene is as lightweight as containers, with extremely short startup time.



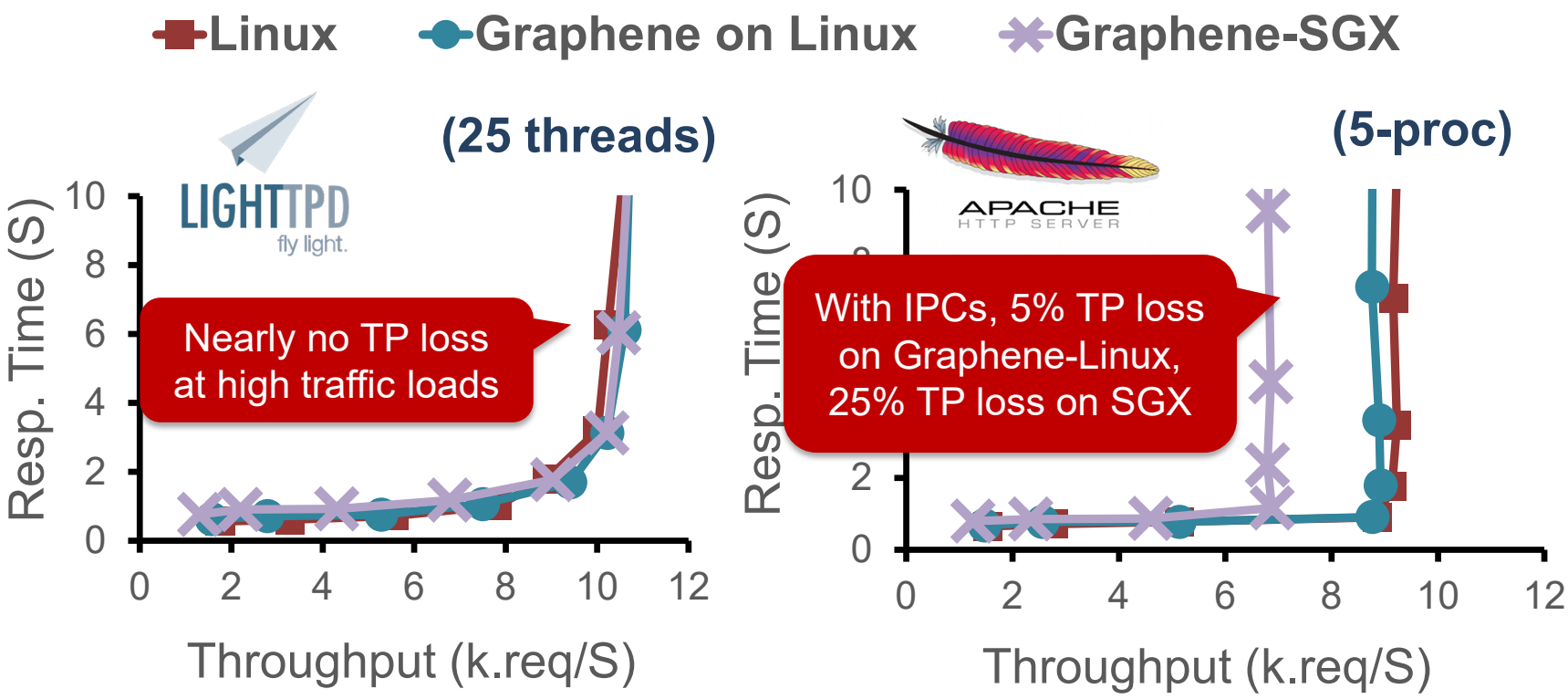


Graphene on Linux LXC KVM

R Benchmarks



Web Servers (Threads vs Processes)



Conclusions

- **LibOS**: compatibility & sandboxing w/o VMs, but light as containers.
- **Graphene LibOS:**
  - Aiming for full Linux compatibility (progress: 45%)
  - What's the craziest place you wanted to run Linux programs?
    It's possible!

https://github.com/oscarlab/graphene

Send your questions & feedbacks to:
support@graphene-project.io

THE LINUX FOUNDATION

OPEN SOURCE SUMMIT

EUROPE

THE LINUX FOUNDATION