Towards a Practical Ecosystem of Specialized OS Kernels

Conghao Liu and Kyle C. Hale

Illinois Institute of Technology

ILLINOIS INSTITUTE OF TECHNOLOGY

HExSA Lab

Overview

1. We were working on Nautilus, an Unikernel developed at NU and IIT.

2. The development and deployment of new applications on Unikernels like Nautilus is really tedious.

3. What tool can we use/create to help us?

4. We developed Diver, a prototype tool aims to make Specialized Operating Systems “easier” to use.

Outline

1. Specialized OSes and problems they face.
2. Our solution and Design goals.
3. Details of our solution.
4. Three important deployment modes.
5. Conclusion and future works.

Outline

1. Specialized OSes and problems they face.
2. Our solution and Design goals.
3. Details of our solution.
4. Three important deployment modes.
5. Conclusion and future works.

Resurgence of SOSes

1. Several reasons like hardware heterogeneity and application diversity impose new challenges to General Purpose Operating Systems.

2. Specialized Operating Systems provide one avenue for addressing these challenges.

3. Examples of SOSes: OSv, Libra, Nautilus, …

Challenges of SOSes

1. POSIX compatibility.

2. Pick the right abstractions for the target workloads.

3. Decide on the right level of protection.

All of these challenges make SOSes “hard” to use. Can we make them “easier” to use without introducing much performance overheads?

Outline

1. Specialized OSes and problems they face.
2. Our solution and Design goals.
3. Details of our solution.
4. Three important deployment modes.
5. Conclusion and future works.

Inspiration from existing tools

1. Capstan for OSv.

2. Cargo for Rust.

Capstan

```
$ cd $PROJECT_DIR
$ capstan package init --name {name} --title {title} --author {author}
$ capstan runtime init --runtime {runtime}
# edit meta/run.yaml to match your application structure
$ capstan package compose {unikernel-name}
```

Cargo

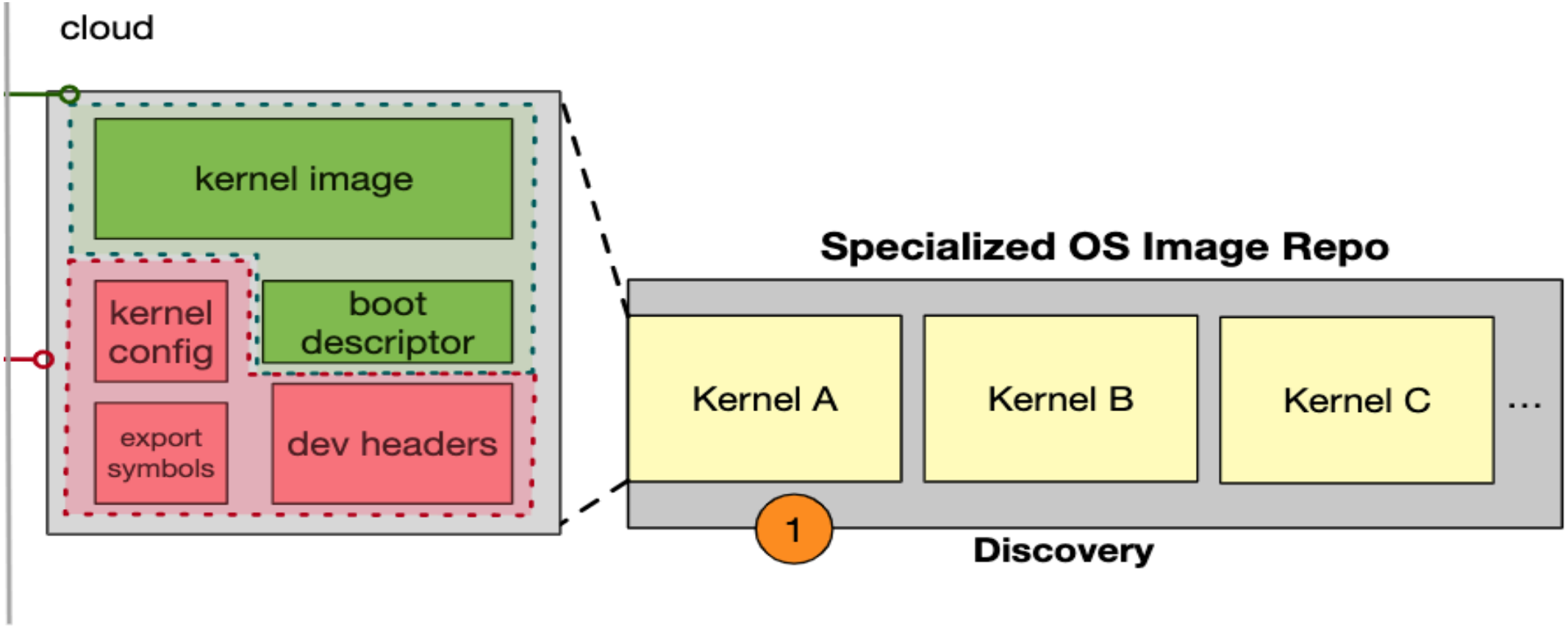
```
$ cargo new {program}
# development
$ cargo build
$ cargo test
$ cargo run
...
```

Our requirements for the ecosystem

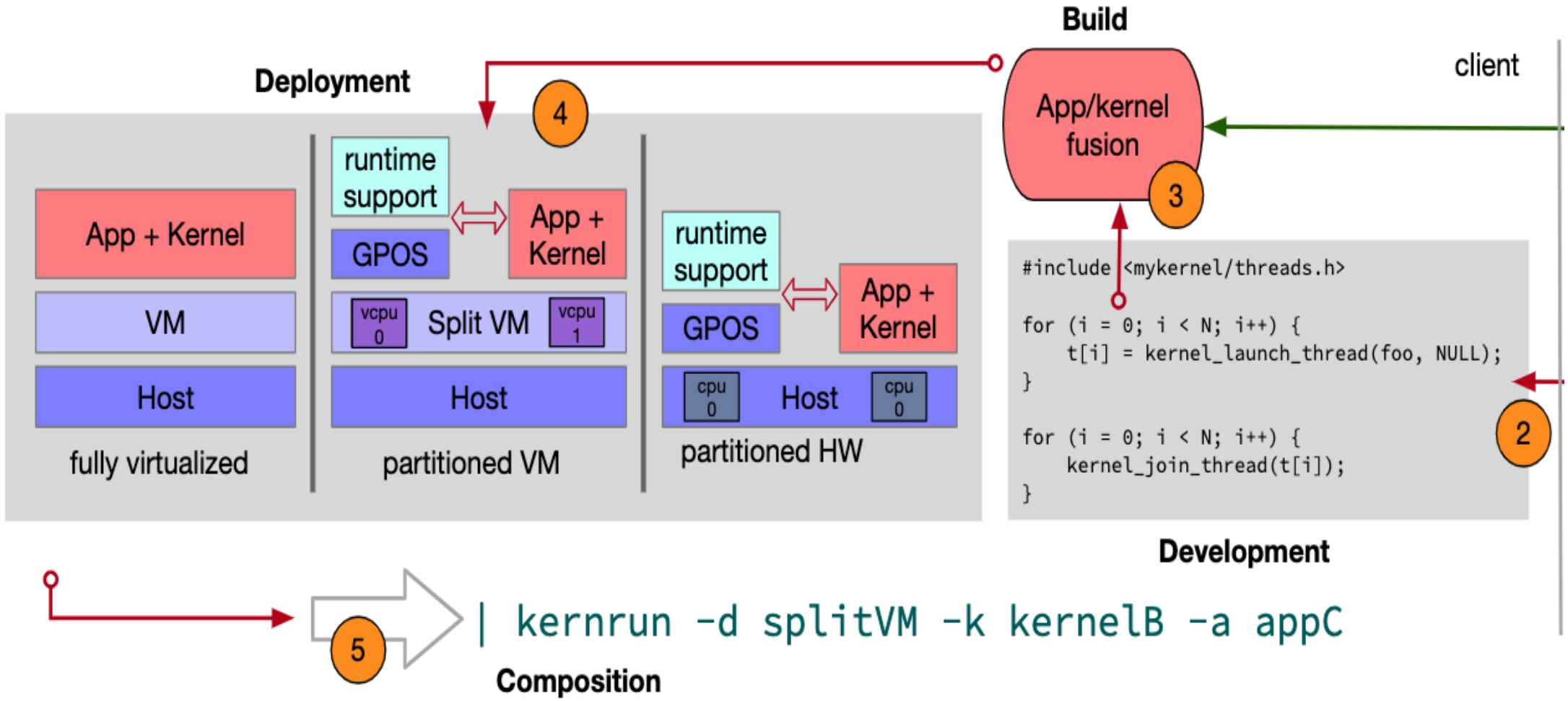
1. Discoverability: Easy to find the kernel we need. (dnf/apt)
2. Ease-of-Use: Easy to build, easy to deploy. (capstan/cargo)
3. Composability: Pipelined workflow using different kernel deployed in different ways. (cat …|grep …)
4. Customizability. Kernel modification.

5. Performance: little performance overhead.

Design on the Server-side



Design on the client-side



Outline

1. Specialized OSes and problems they face.
2. Our solution and Design goals.
3. Details of our solution.
4. Three important deployment modes.
5. Conclusion and future works.

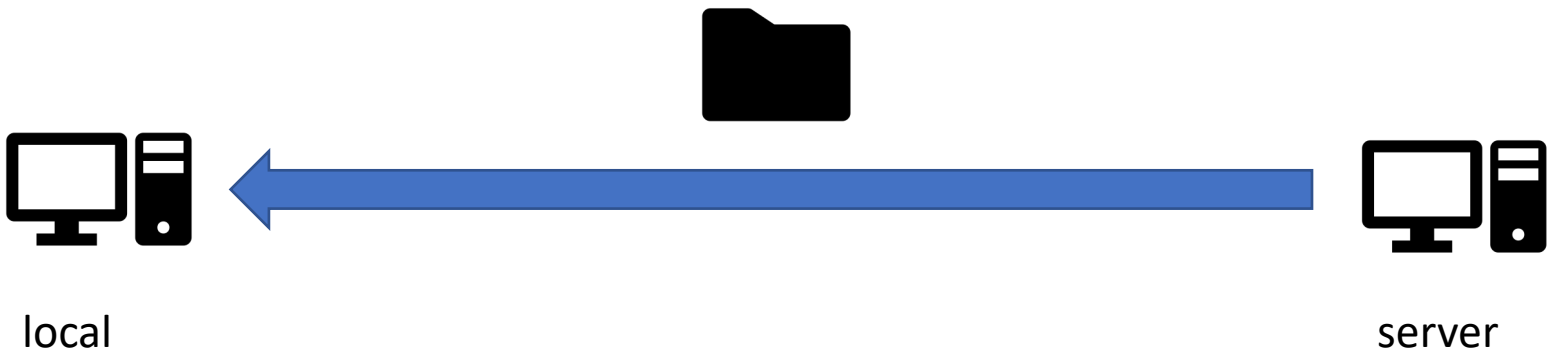
Diver

1. It can search/download kernels by name/tags.
2. It can publish new kernel images.
3. It helps build and deploy your code.

Discoverbility

$> diver init helloworld nautilus

…



Ease-of-use

#coding…

$> diver build [helloworld]

…

$> diver dive -d splitVM

Nautilus-shell>

Customizability

$> diver init hw nautilus

$> cd hw

$> ls –a

. .. .nautilus Makefile main.c

Deployment Modes

1. Fully virtualized Environment
2. Partitioned VMs
3. Partitioned hardware

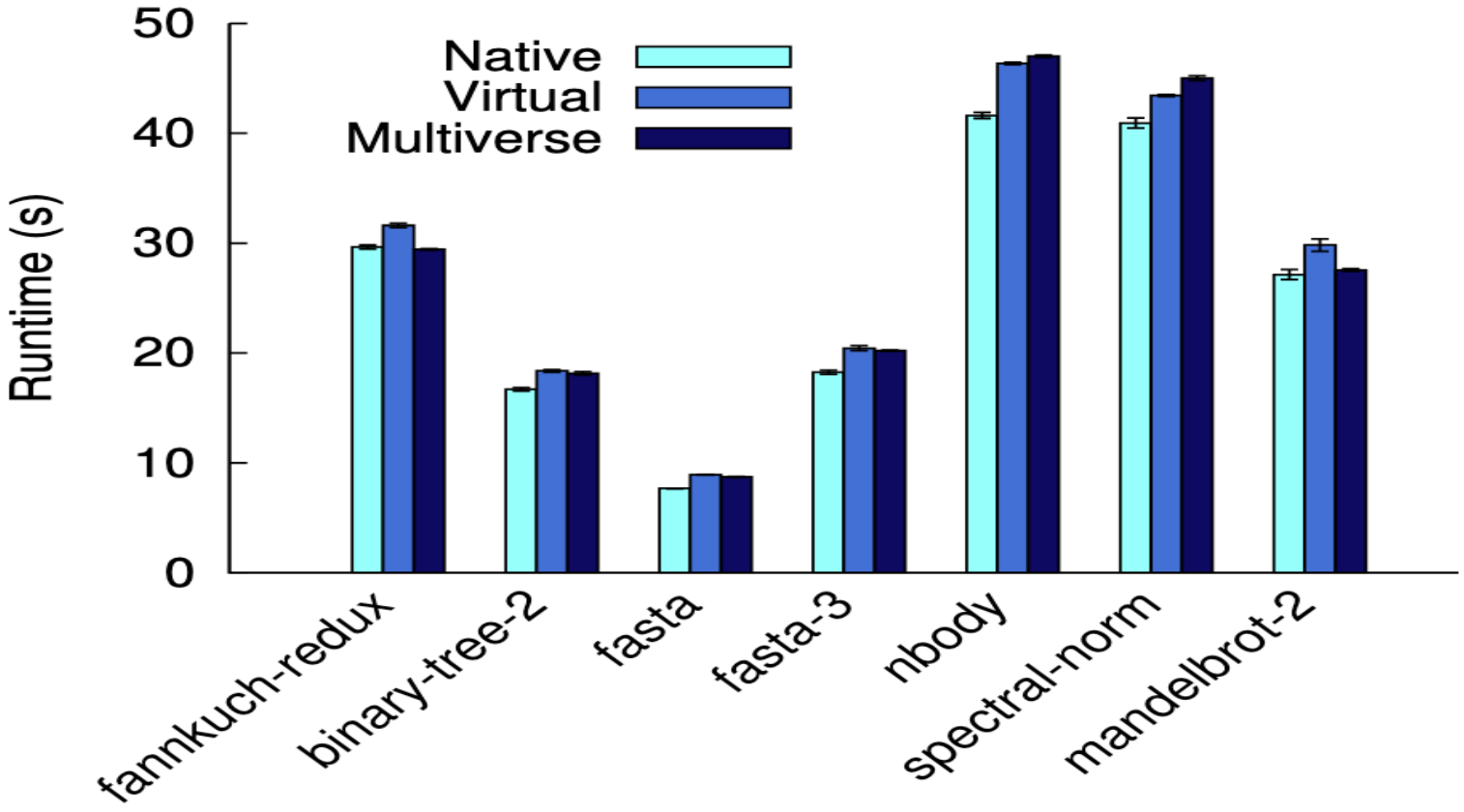
Partitioned VMs

1. Libra first explored this approach for running JVM in virtualized execution environment.

2. Co-existence of GPOS and SOS in a space-partitioned VM.
   - Multiverse and HRT.

3. Syscall-delegation makes SOS more versatile.

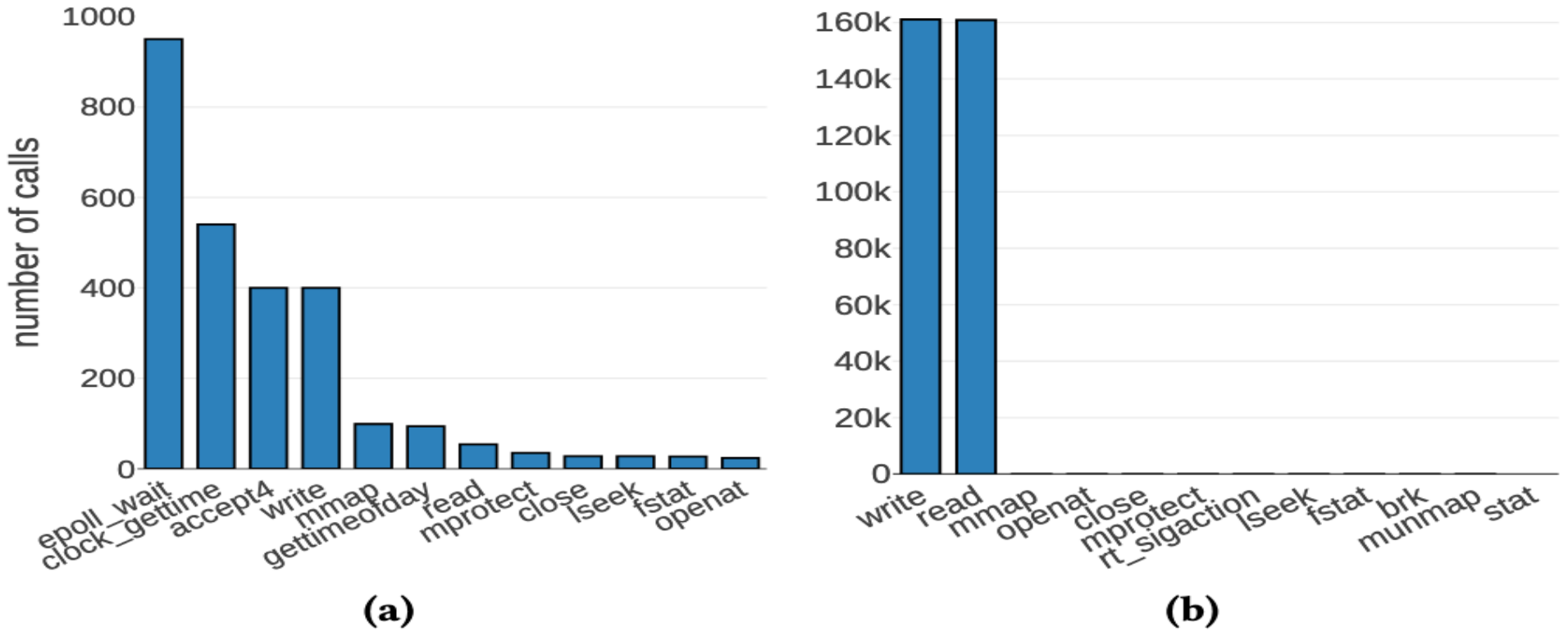
Overheads of partitioned VM are low!



Language shootout benchmark performance with Racket runtime running native, in a virtual machine, and a VM split between two OSes (using Multiverse).

System call breakdown.

Only a small set of system calls matters most!



Histograms representing syscall invocation trace for memcached and bzip2.

Why is this mode useful?

It enables incremental porting of legacy code!

Partitioned Hardware

1. Lange et al. explored this mode using the Pisces Co-kernel architecture and the XEMEM system for efficiently sharing memory between kernels.

2. Physical hardware resources partitioned between a GPOS and a specialized kernel.

3. The GPOS must support offlining cores.

4. The specialized OS must support bootup in a special software environment

Outline

1. Specialized OSes and problems they face.
2. Our solution and Design goals.
3. Details of our solution.
4. Three important deployment modes.
5. Conclusion and future works.

Conclusion

1. It’s the time to begin building ecosystems for SOSes to encourage experimentation and design iteration.
2. We described several requirements for such ecosystem that should meet.
3. We presented a prototype of such tool called Diver.

Future works

1. Integrate Partitioned Hardware deployment mode into Diver.
2. Add support for more SOSes.
3. Explore the standard interface/features SOSes should meet to fit in with Diver.
4. Explore interface/techniques to enable pipelined workflow using different kernels in different deployment mode.

Thanks!

Conghao Liu: cliu115@hawk.iit.edu

Kyle Hale: khale@cs.iit.edu

ILLINOIS INSTITUTE OF TECHNOLOGY

HExSA Lab