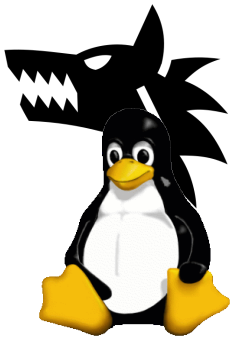
Lupine Linux



“A Linux in Unikernel Clothing”

Dan Williams (IBM)

With Hsuan-Chi (Austin) Kuo (UIUC + IBM), Ricardo Koller (IBM), Sibin Mohan (UIUC)

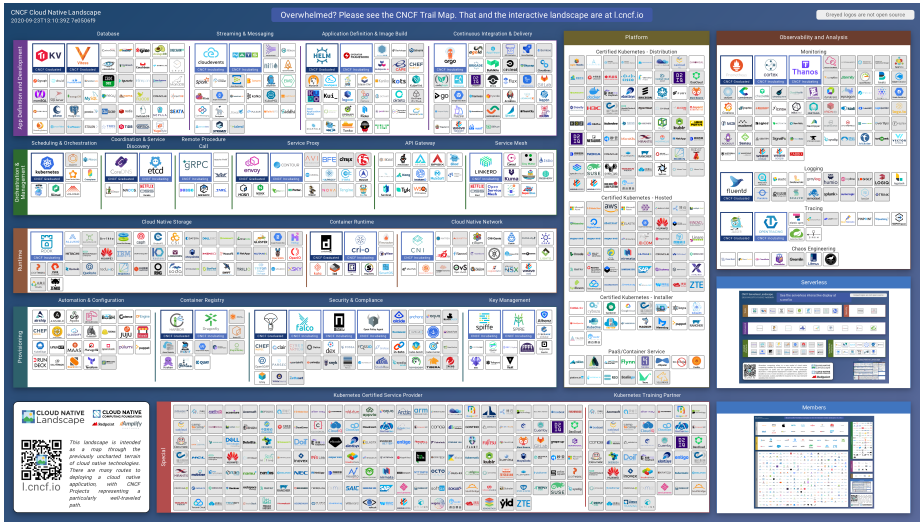
IBM

Roadmap

- Context
  - Containers and isolation
  - Unikernels
  - Nabla containers
- Lupine Linux
  - A linux in unikernel clothing
- Concluding thoughts

Containers are great!

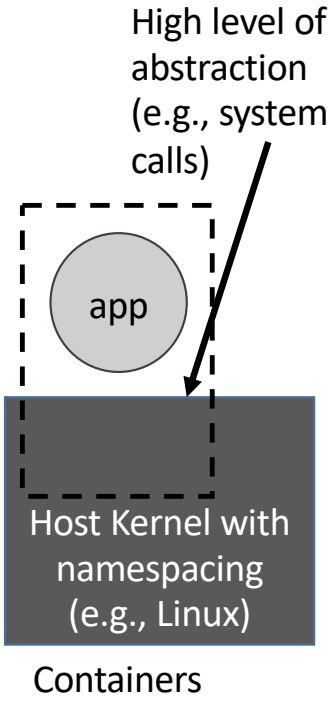
- Have changed how applications are packaged, deployed and developed
- Normal processes, but “contained”
  - Namespaces, cgroups, chroot
- Lightweight
  - Start quickly, “bare metal”
  - Easy image management (layered fs)
- Tooling/orchestration ecosystem



But…

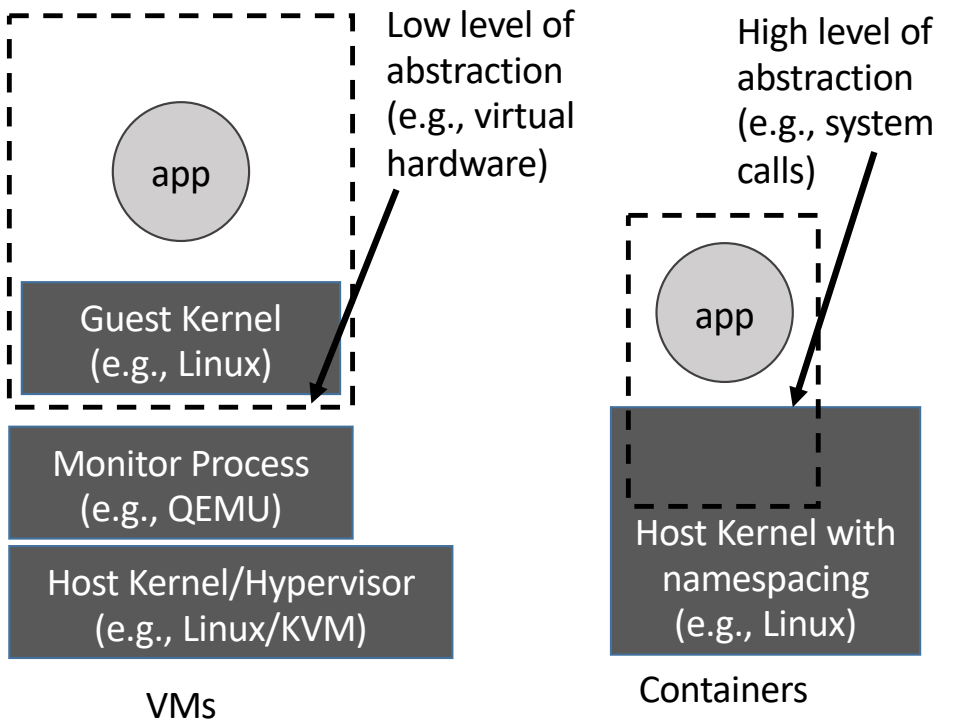
- Large attack surface to the host
- Limits adoption of container-first architecture

- Fortunately, we know how to reduce attack surface!



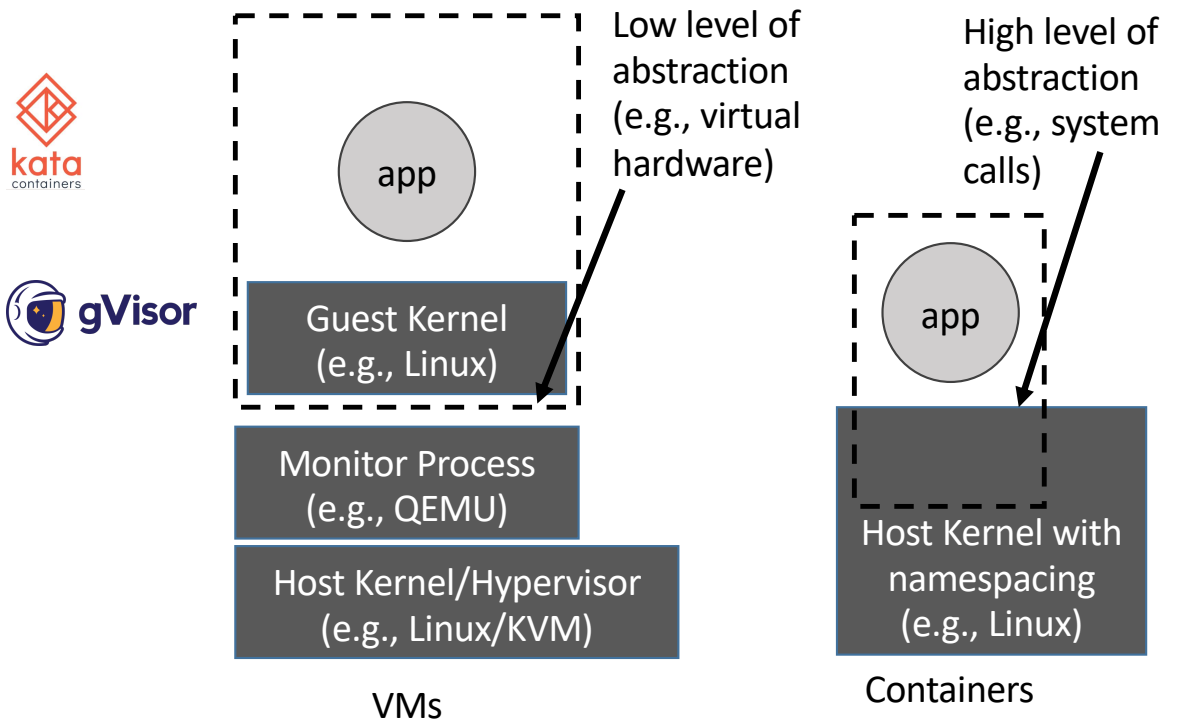
Deprivileging and unsharing kernel functionality

- Virtual machines (VMs)
  - Guest kernel
  - Thin interface



Deprivileging and unsharing kernel functionality

- Virtual machines (VMs)
  - Guest kernel
  - Thin interface
- Userspace kernel
  - Performance issues



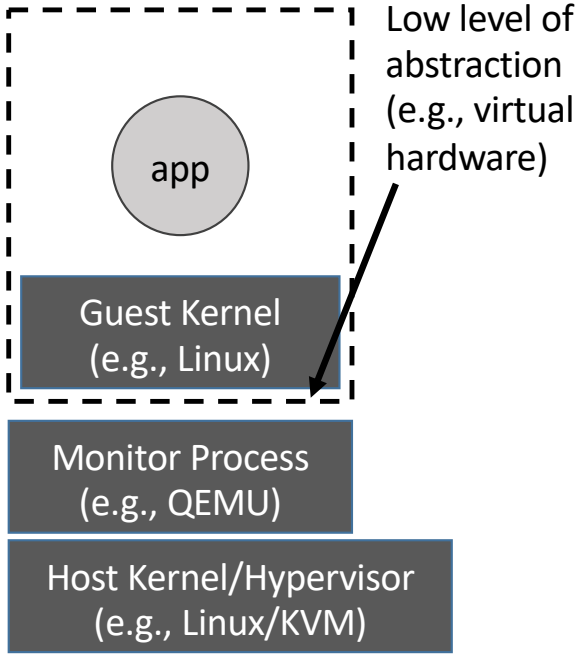
But wait? Aren't VMs slow and heavyweight?

- Boot time?
- Memory footprint?

- Especially for environments like serverless??!!

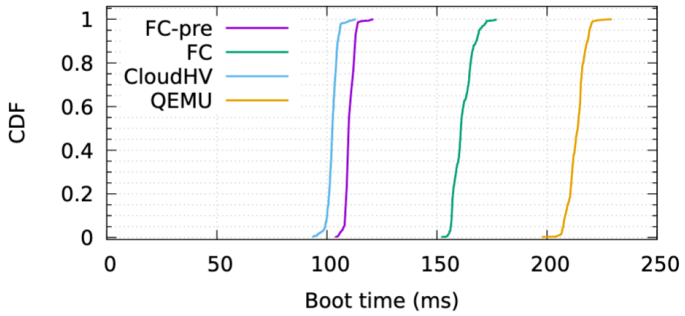
VMs are becoming lightweight

- Thin monitors
  - e.g., AWS Firecracker
  - Reduce complexity for performance (e.g., no PCI)

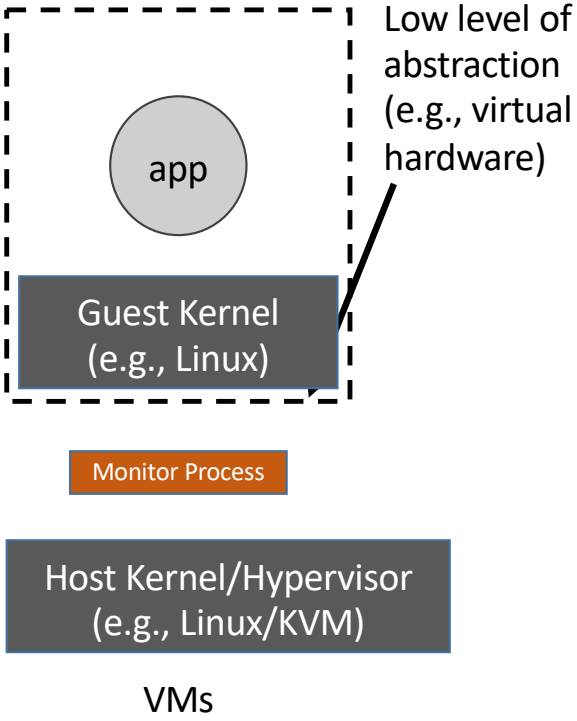


VMs are becoming lightweight

- Thin monitors
  - e.g., AWS Firecracker
  - Reduce complexity for performance (e.g., no PCI)

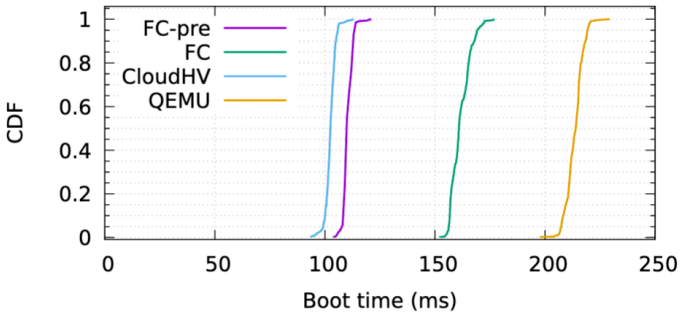


Firecracker boot times as reported in Agache et al., NSDI 2020

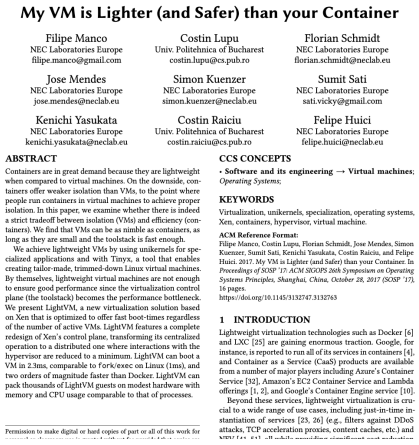


VMs are becoming lightweight

- Thin monitors
  - e.g., AWS Firecracker
  - Reduce complexity for performance (e.g., no PCI)



Firecracker boot times as reported in Agache et al., NSDI 2020



My VM is Lighter (and Safer) than your Container

Filipe Manco, NEC Laboratories Europe, filipe.manco@gmail.com
Costin Lupu, Univ. Politehnica of Bucharest, costin.lupu@cs.pub.ro
Florian Schmidt, NEC Laboratories Europe, florian.schmidt@neclab.eu
Jose Mendes, NEC Laboratories Europe, jose.mendes@neclab.eu
Simon Kuenzer, NEC Laboratories Europe, simon.kuenzer@neclab.eu
Sumit Sati, NEC Laboratories Europe, sati.vicky@gmail.com
Kenichi Yasukata, NEC Laboratories Europe, kenichi.yasukata@neclab.eu
Costin Raiciu, Univ. Politehnica of Bucharest, costin.raiciu@cs.pub.ro
Felipe Huici, NEC Laboratories Europe, felipe.huici@neclab.eu

ABSTRACT

Containers are in great demand because they are lightweight when compared to virtual machines. On the downside, containers offer weaker isolation than VMs, to the point where people run containers in virtual machines to achieve proper isolation. In this paper, we examine whether there is indeed a strict tradeoff between isolation (VMs) and efficiency (containers). We find that VMs can be as nimble as containers, as long as they are small and the toolstack is fast enough.

We achieve lightweight VMs by using unikernels for specialized applications and with Tinyx, a tool that enables creating tailor-made, trimmed-down Linux virtual machines. By themselves, lightweight virtual machines are not enough to ensure good performance since the virtualization control plane (the toolstack) becomes the performance bottleneck. We present LightVM, a new virtualization solution based on Xen that is optimized to offer fast boot-times regardless of the number of active VMs. LightVM features a complete redesign of Xen's control plane, transforming its centralized operation to a distributed one where interactions with the hypervisor are reduced to a minimum. LightVM can boot a VM in 2.3ms, comparable to `fork/exec` on Linux (1ms), and two orders of magnitude faster than Docker. LightVM can pack thousands of LightVM guests on modest hardware with memory and CPU usage comparable to that of processes.

CCS CONCEPTS

• **Software and its engineering → Virtual machines**; *Operating Systems*;

KEYWORDS

Virtualization, unikernels, specialization, operating systems, Xen, containers, hypervisor, virtual machine.

ACM Reference Format:

Filipe Manco, Costin Lupu, Florian Schmidt, Jose Mendes, Simon Kuenzer, Sumit Sati, Kenichi Yasukata, Costin Raiciu, and Felipe Huici. 2017. My VM is Lighter (and Safer) than your Container. In *Proceedings of SOSP '17: ACM SIGOPS 26th Symposium on Operating Systems Principles, Shanghai, China, October 28, 2017 (SOSP '17),* 16 pages.

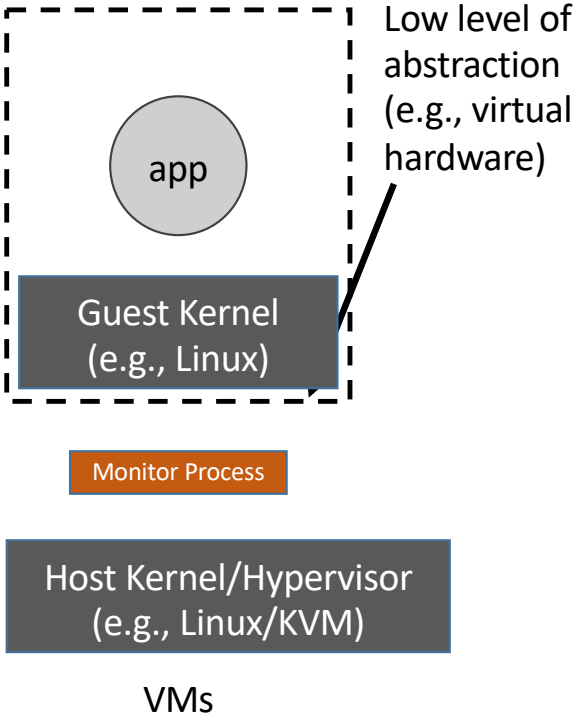
https://doi.org/10.1145/3132747.3132763

1 INTRODUCTION

Lightweight virtualization technologies such as Docker [6] and LXC [25] are gaining enormous traction. Google, for instance, is reported to run all of its services in containers [4], and Container as a Service (CaaS) products are available from a number of major players including Azure's Container Service [32], Amazon's EC2 Container Service and Lambda offerings [1, 2], and Google's Container Engine service [10].

Beyond these services, lightweight virtualization is crucial to a wide range of use cases, including just-in-time instantiation of services [23, 26] (e.g., filters against DDoS attacks, TCP acceleration proxies, content caches, etc.) and

Manco et al., SOSP 2017



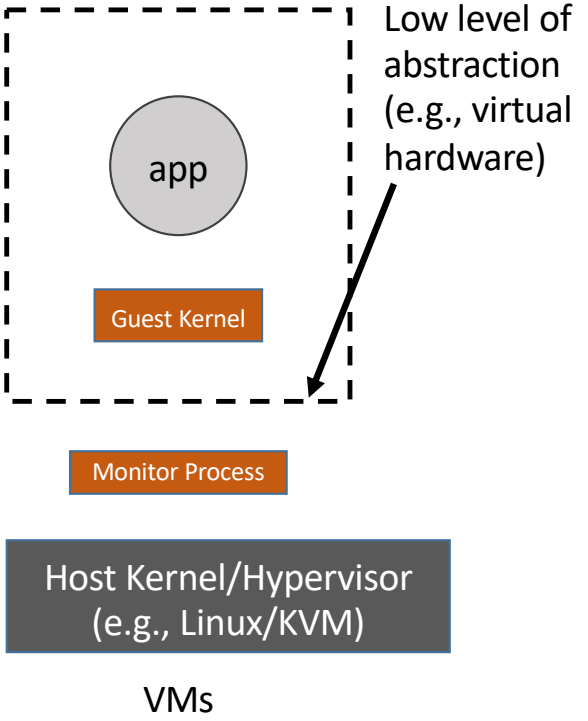
VMs are becoming lightweight

- Thin monitors
  - e.g., AWS Firecracker
  - Reduce complexity for performance (e.g., no PCI)

- Thin guests?
  - Userspace: (e.g., Ubuntu --> Alpine Linux)
  - Kernel configuration (e.g., TinyX, **Lupine**)
  - Unikernels

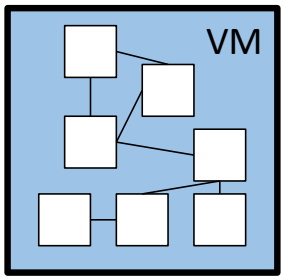
Eurosys 2020

KubeCon 2020



Unikernels are thin guests to the extreme

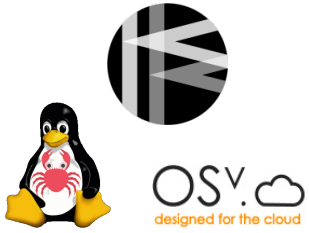
- An application linked with **library OS** components
- Run on **virtual hardware** (like) abstraction
- Single CPU



- Language-specific
  - MirageOS (OCaml)
  - IncludeOS (C++)

- Legacy-oriented
  - Rumprun (NetBSD-based)
  - Hermitux
  - OSv

Claim binary compatibility with Linux



IBM

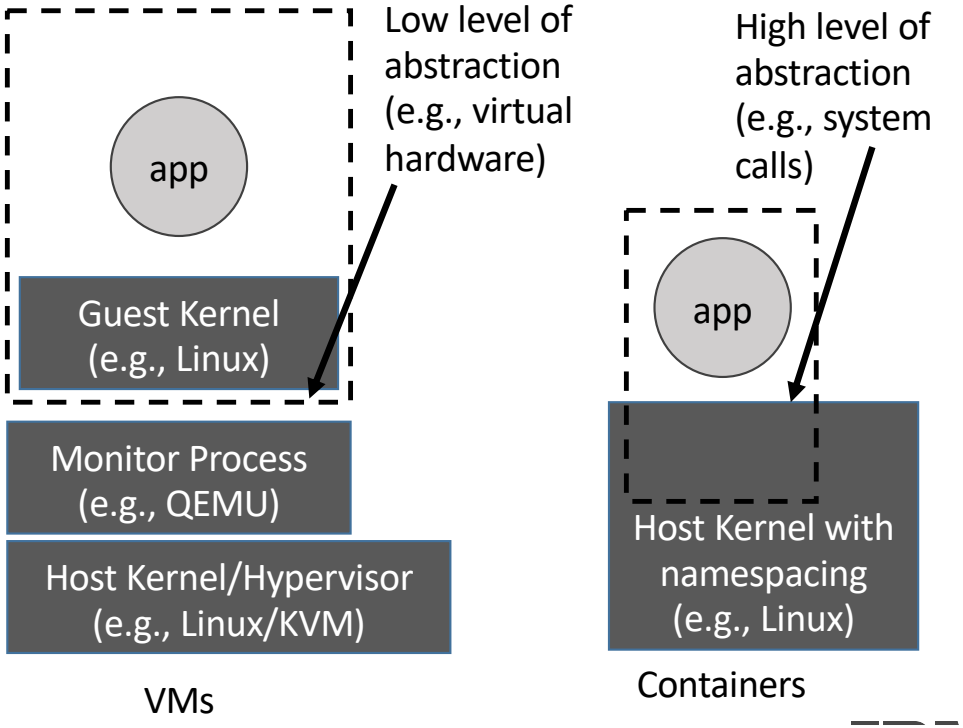
Deprivileging and unsharing kernel functionality

- Virtual machines (VMs)
  - Guest kernel
  - Thin interface
- Userspace kernel
  - E.g., UML
  - Performance issues
- Library OS / unikernel
  - Only-what-you need
  - Lightweight

kata containers

gVisor

∇



IBM

What we learned from Nabla containers

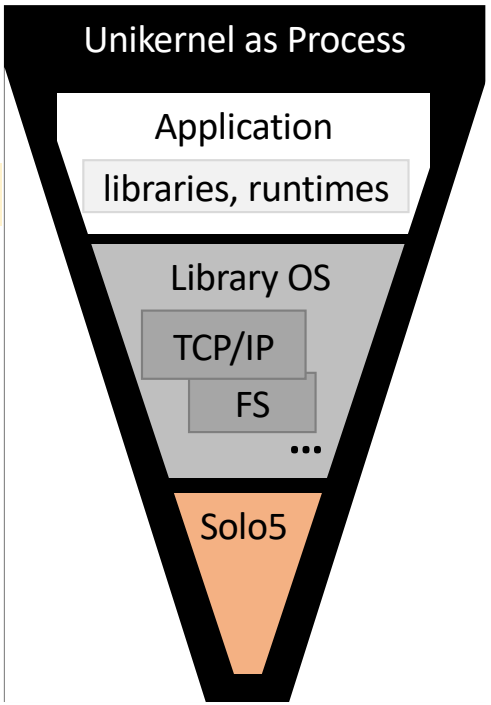
- Nabla containers are unikernels as processes
  - Can achieve or exceed lightweight characteristics of containers
  - Interfaces are what matter, not virtualization HW

HotCloud '16, HotOS '17

HotCloud '18, SOCC '18

- But we lose a lot: Generality
  - Lupine Linux: applying unikernel techniques to Linux VMs

Eurosys '20



What we learned from Nabla containers

- Nabla containers are unikernels as processes
  - Can achieve or exceed lightweight characteristics of containers
  - Interfaces are what matter, not virtualization HW

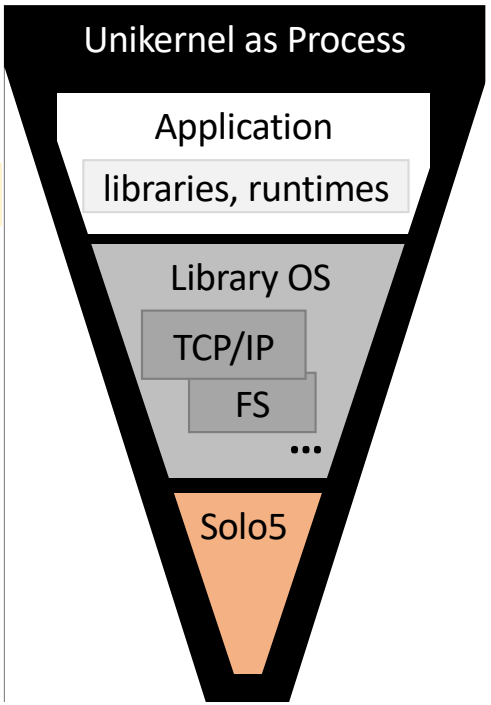
HotCloud '16, HotOS '17

HotCloud '18, SOCC '18

- But we lose a lot: Generality
  - Lupine Linux: applying unikernel techniques to Linux VMs

this talk →

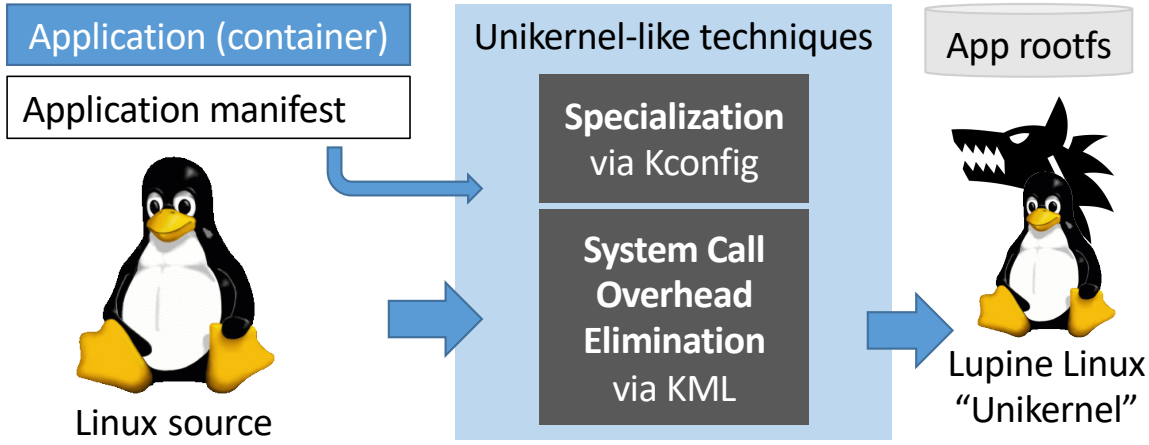
Eurosys '20



IBM

Lupine Linux Overview and Roadmap

- Introduction
- Lupine Linux
  - Specialization
  - System Call Overhead Elimination
  - Putting it together
- Evaluation
- Discussion
- Related Work



Unikernels are great

- Small kernel size
- Fast boot time
- Performance
- Security

Unikernels are great… but

- Small kernel size
- Fast boot time
- Performance
- Security

- Lack full Linux support
- Hermitux: supports only 97 system calls
- OSv:
  - application needs to be compiled with –PIE, can’t use TLS
  - Static-linked applications are not supported
  - Fork() , execve() are not supported
  - Special files are not supported such as /proc
  - Signal mechanism is not complete
- Rumprun: only 37 curated applications
- Community is too small to keep it rolling

IBM

Lupine Linux
“Unikernel”

Can Linux

> be as small as

> boot as fast as

> outperform

unikernels?

IBM

Lupine Linux
“Unikernel”

Can Linux

> be as small as

> boot as fast as

> outperform

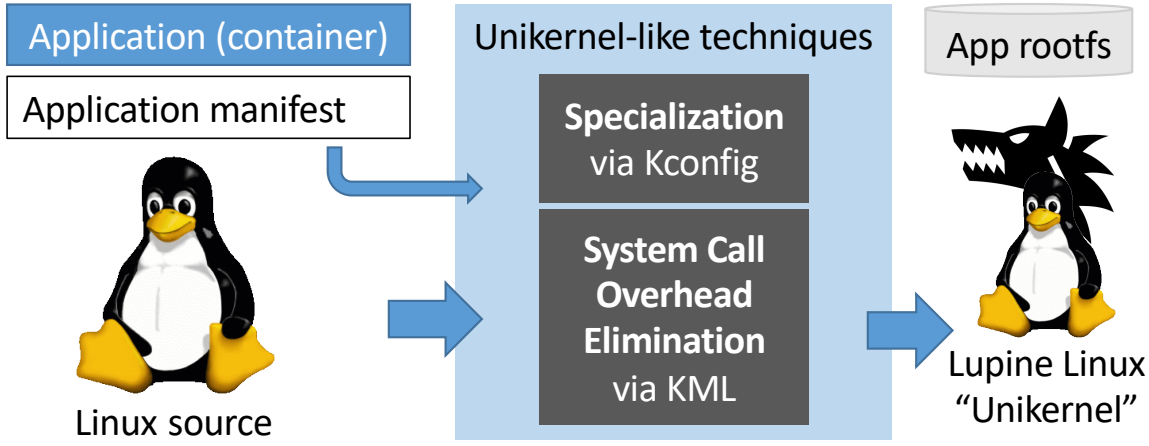
unikernels?

- Spoiler alert: Yes!
  - 4MB image size
  - 23 ms boot time
  - Up to 33% higher throughput

IBM

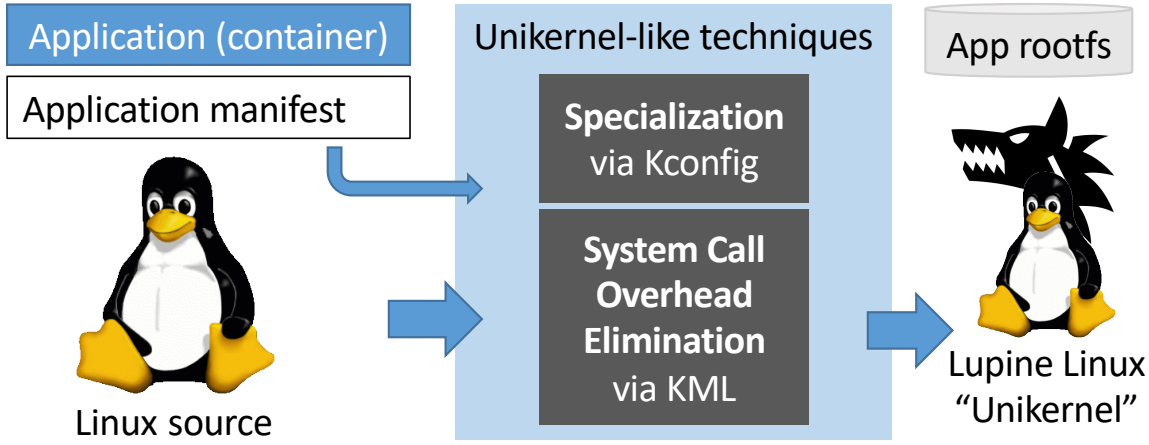
Lupine Linux Overview and Roadmap

- Introduction
- Lupine Linux
  - Specialization
  - System Call Overhead Elimination
  - Putting it together
- Evaluation
- Discussion
- Related Work



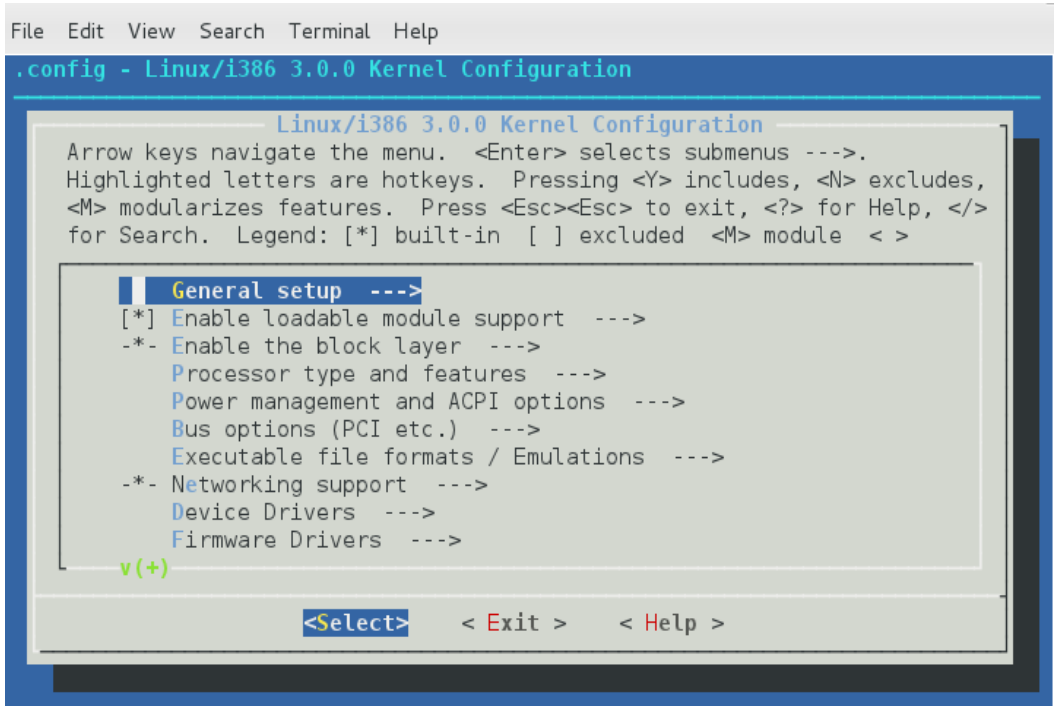
Lupine Linux Overview and Roadmap

- Introduction
- Lupine Linux
  - **Specialization**
  - System Call Overhead Elimination
  - Putting it together
- Evaluation
- Discussion
- Related Work



Unikernel technique #1: Specialization

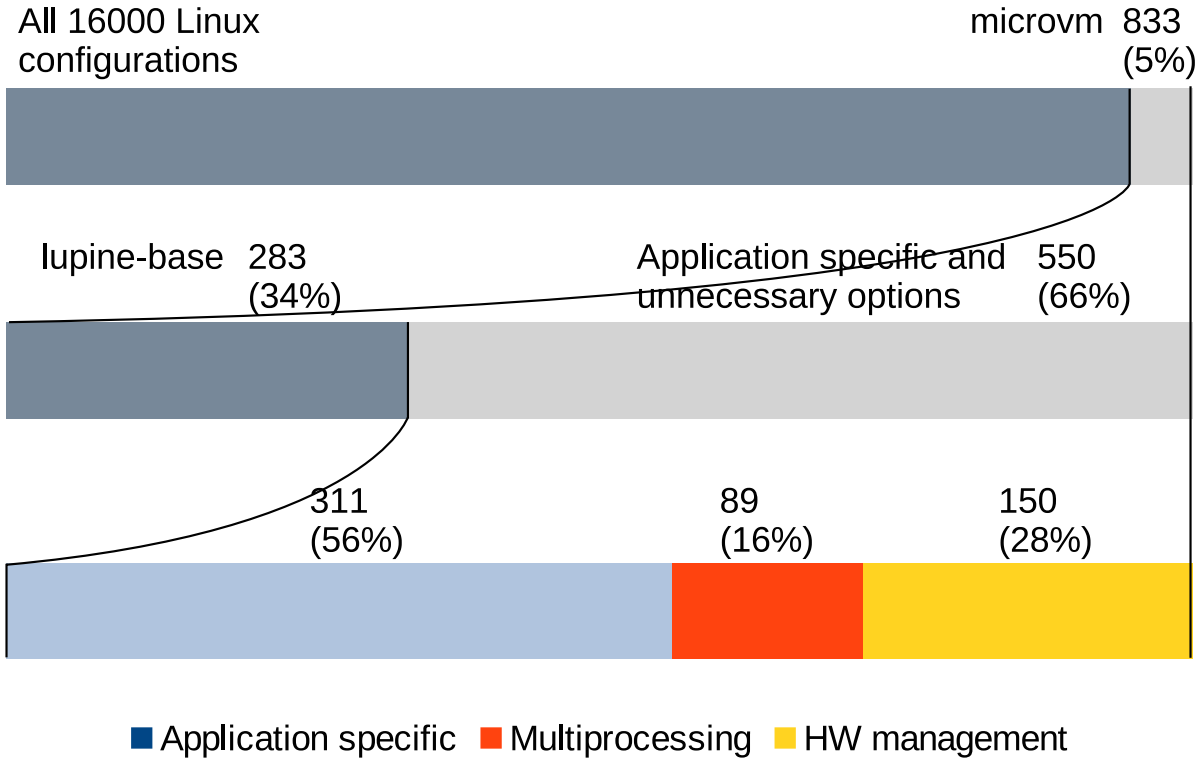
- Unikernels include only what is needed
- Linux is very configurable
  - Kconfig
  - 16,000 options
    - Drivers
    - Filesystems
    - Processor features
    - …



IBM

Specializing Linux through configuration

- Start with Firecracker microvm configuration
- Assuming unikernel-like workload, can remove even more!
  - Application-specific options
  - Multiprocessing
  - HW management



Application-specific options

- Example: system calls

Option	Enabled System Call(s)
ADVISE_SYSCALLS	madvise, fadvise64
AIO	io_setup, io_destroy, io_submit, io_cancel, io_getevents
BPF_SYSCALL	bpf
EPOLL	epoll_ctl, epoll_create, epoll_wait, epoll_pwait
EVENTFD	eventfd, eventfd2
FANOTIFY	fanotify_init, fanotify_mark
FHANDLE	open_by_handle_at, name_to_handle_at
FILE_LOCKING	flock
FUTEX	futex, set_robust_list, get_robust_list
INOTIFY_USER	inotify_init, inotify_add_watch, inotify_rm_watch
SIGNALFD	signalfd, signalfd4
TIMERFD	timerfd_create, timerfd_gettime, timerfd_settime

- Kernel services
  - e.g., /proc, sysctl
- Kernel library
  - Crypto routines
  - Compression routines
- Debugging/information

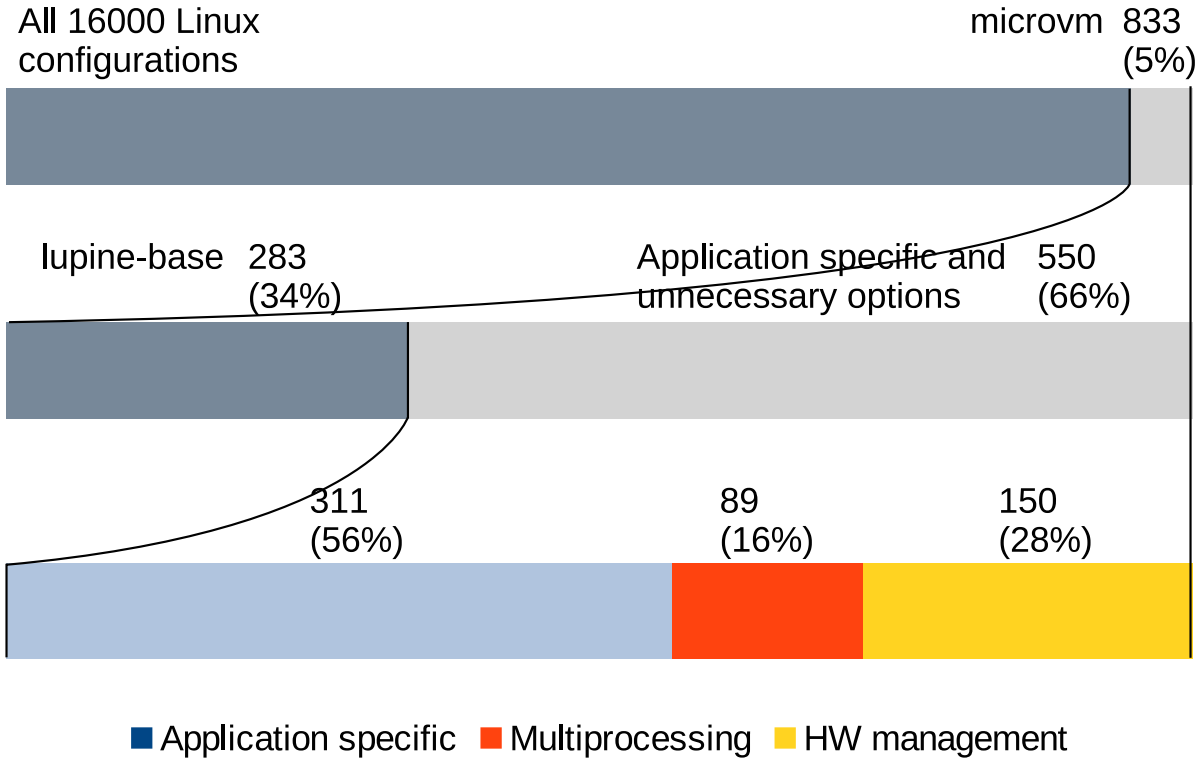
Other assumptions from unikernels

- Unikernels are not intended for multiple processes
  - Related to isolating, accounting for processes
    - Cgroups, namespaces, SElinux, seccomp, KPTI
  - SMP, NUMA
  - Module support
- Unikernels are not intended for general hardware
  - Intended to run as VMs in the cloud
  - microVM removes many drivers and arch-specific configs
  - Lupine removes more, including power mgmt

How to get an app-specific kernel config

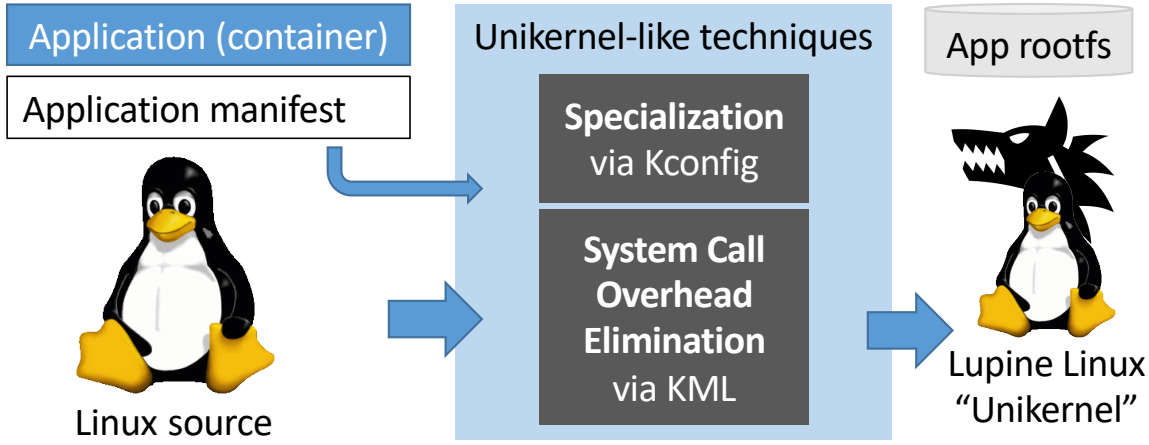
- Start with lupine-base
- Manual trial and error
  - Guided by application output
  - E.g., *the futex facility returned an unexpected error code*

    => CONFIG_FUTEX
- In general, this is a hard problem



Lupine Linux Overview and Roadmap

- Introduction
- Lupine Linux
  - Specialization
  - **System Call Overhead Elimination**
  - Putting it together
- Evaluation
- Discussion
- Related Work



Unikernel technique #2: System call overhead elimination

- Kernel Mode Linux (KML)
  - Non-upstream patch (latest Linux 4.0)
  - Execute unmodified apps in kernel mode
  - User program can directly access the kernel

- Replace “syscall” instruction with “call” in libc e.g., musl

Location exposed via vsyscall

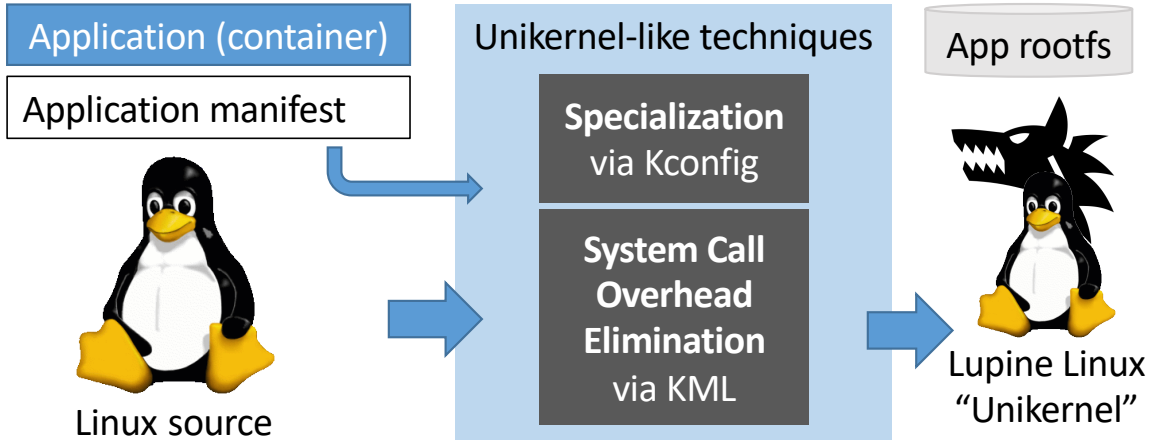
```
- __asm__ __volatile__ ("syscall" : "=a"(ret) :
+ __asm__ __volatile__ ("call *%1" : "=a"(ret) : "r"(__kml),
                        "a"(n), "D"(a1), "S"(a2),
                        "d"(a3), "r"(r10), "r"(r8),
                        "r"(r9) : "rcx", "r11", "memory");
```

- Requires relink for static binaries
  - Less invasive than build modifications for unikernels

IBM

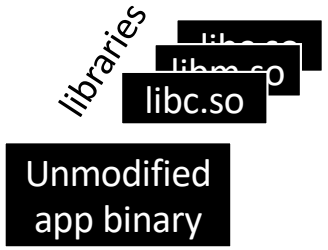
Lupine Linux Overview and Roadmap

- Introduction
- Lupine Linux
  - Specialization
  - System Call Overhead Elimination
  - **Putting it together**
- Evaluation
- Discussion
- Related Work



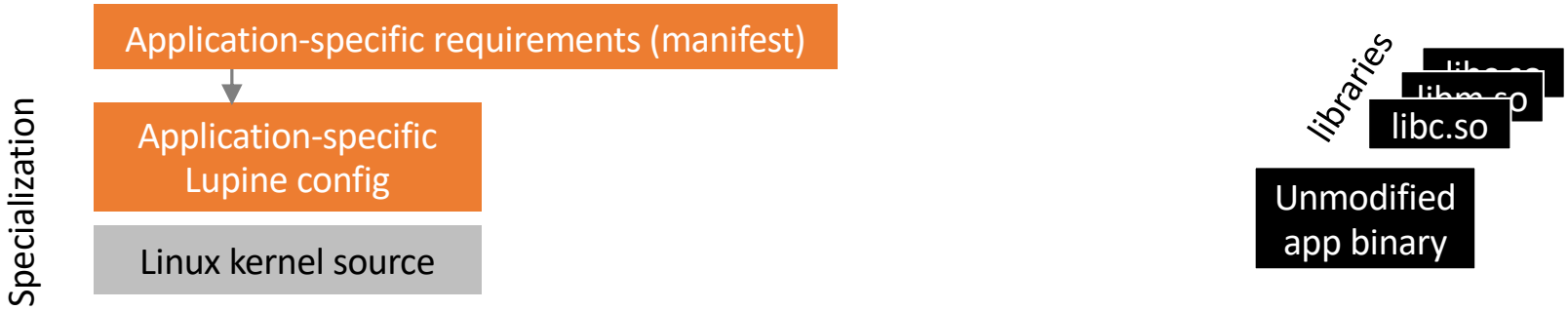
IBM

Putting it all together



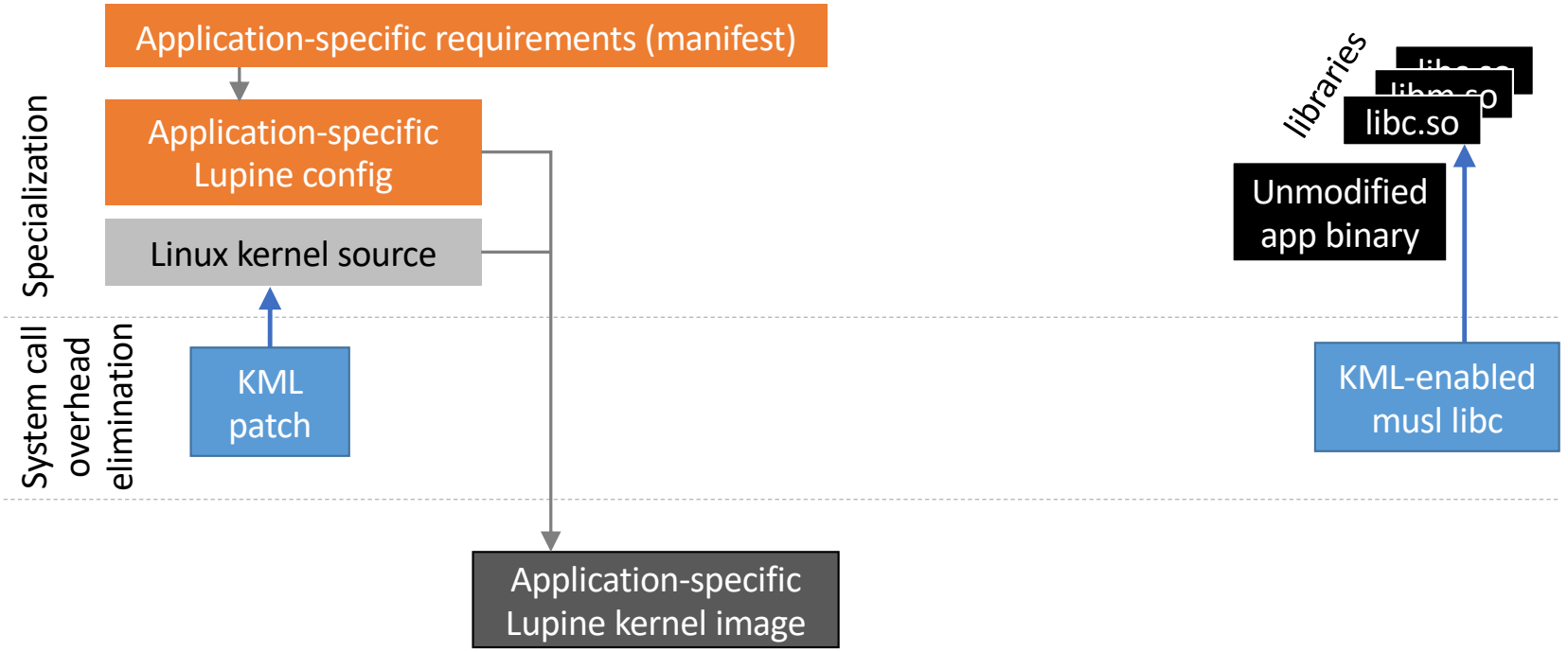
Linux kernel source

Putting it all together



Putting it all together

Putting it all together

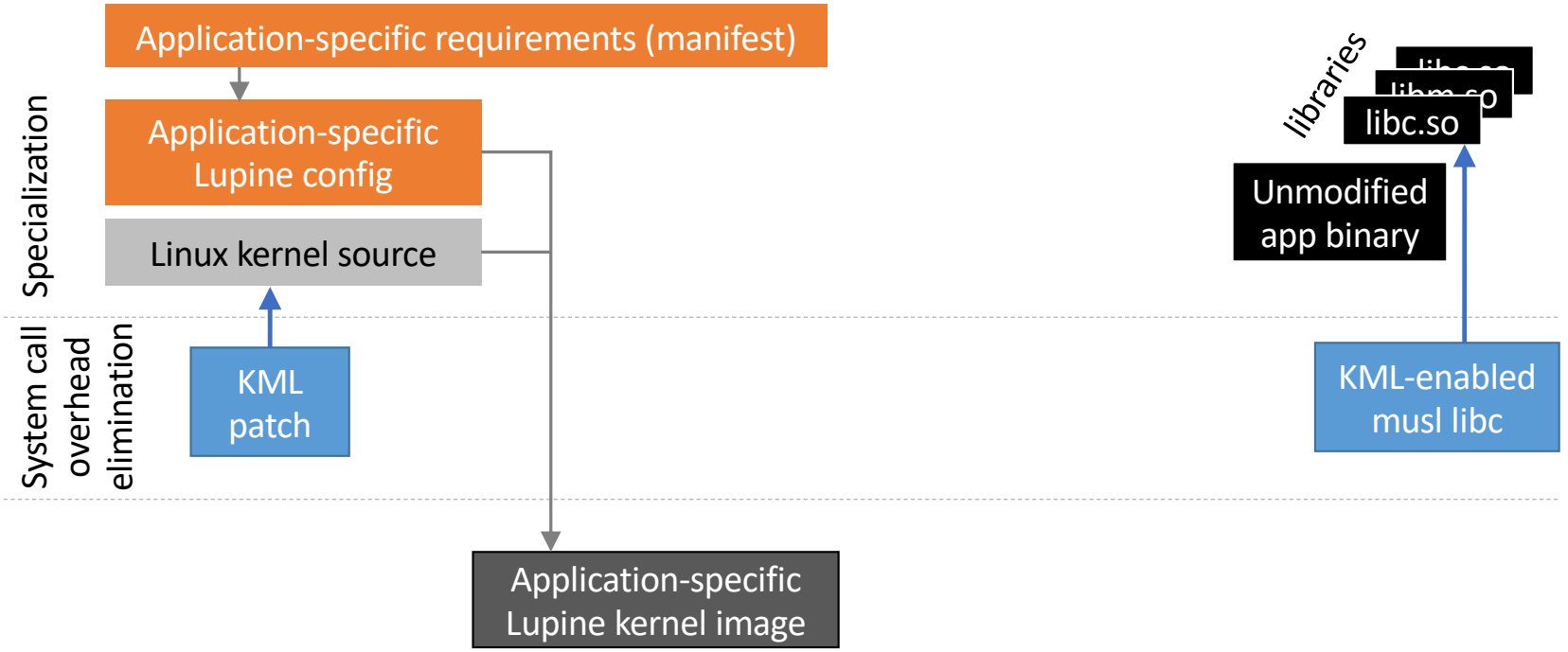


Remaining issues

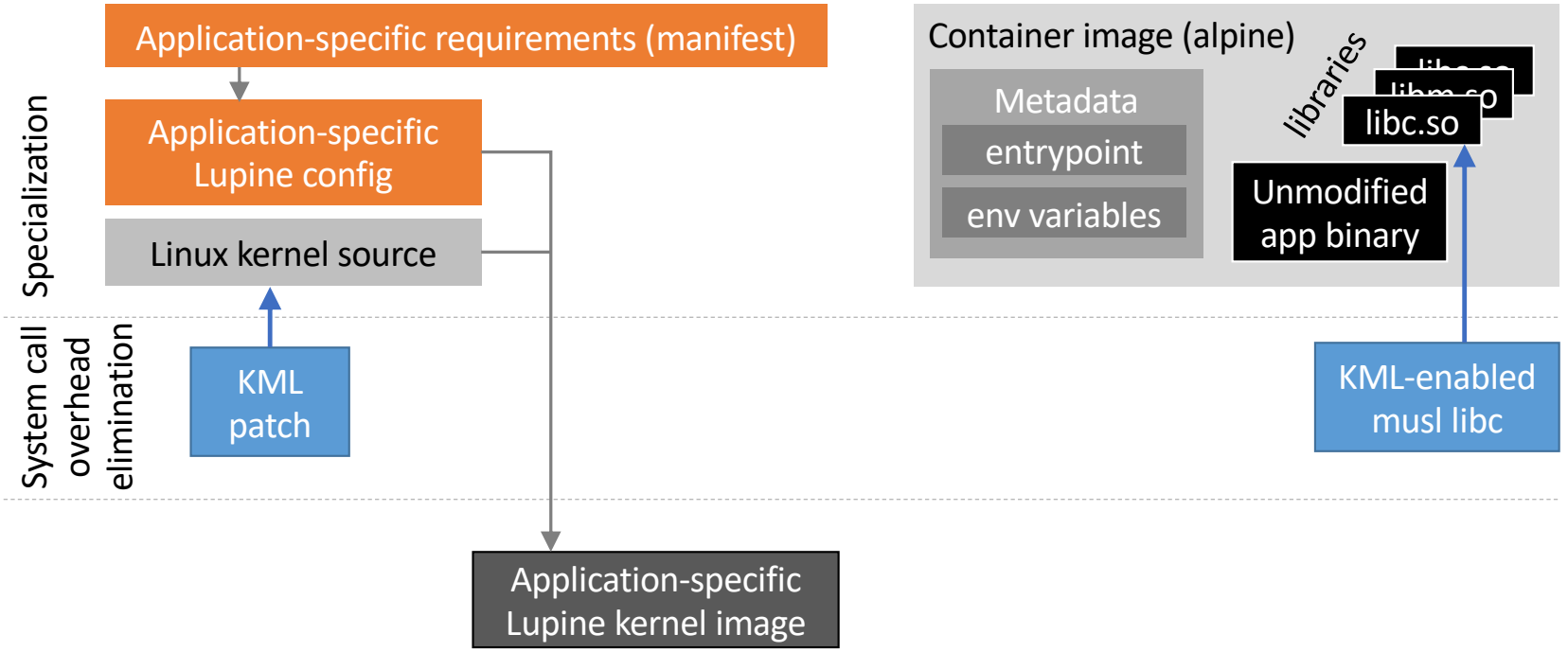
- How to build a root filesystem for Linux
  - Container images are root filesystems already
  - Contains both application and necessary libraries

- How to start the (single) application
  - Linux kernel parameter “init” specifies first program, usually “/sbin/init”
  - Boot the kernel with “init=/app”
  - Caveats:
    - May need some simple setup (e.g., network)
    - Application-specific!

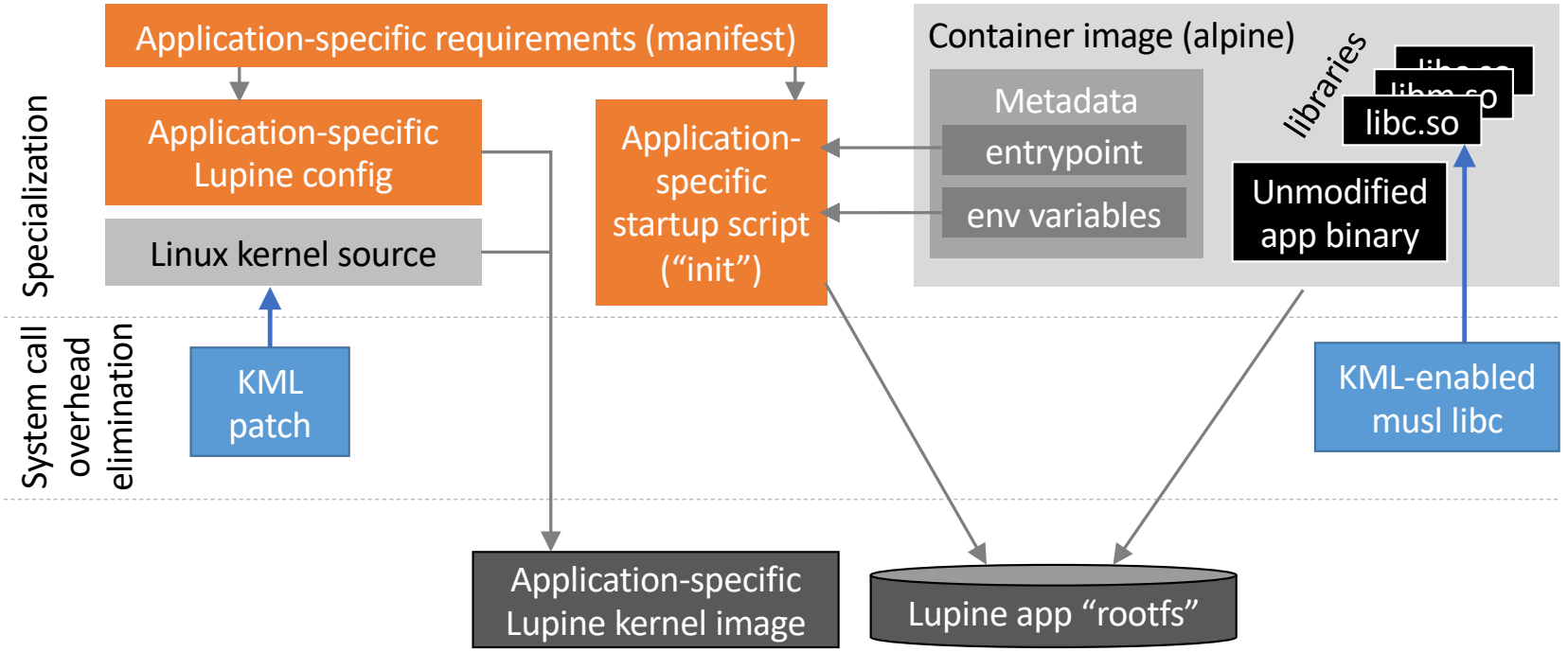
Putting it all together



Putting it all together

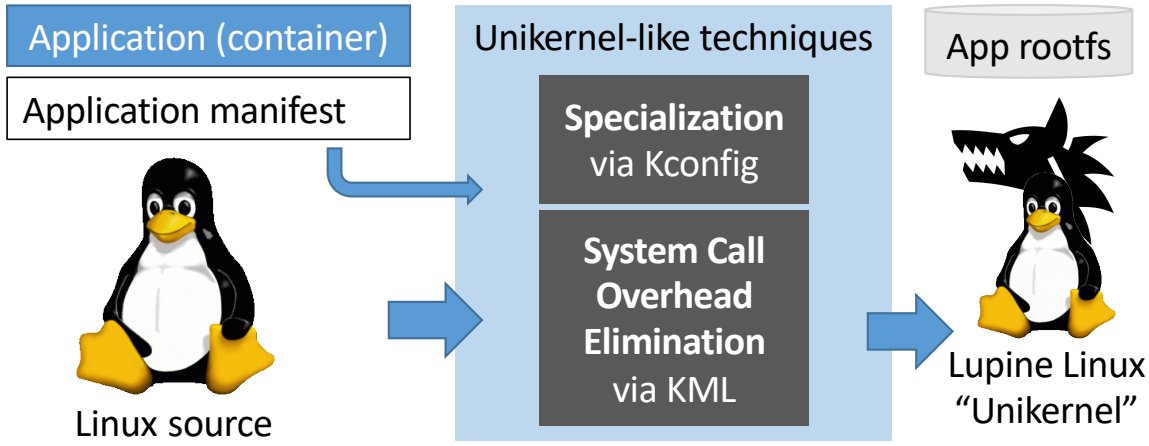


Putting it all together



Lupine Linux Overview and Roadmap

- Introduction
- Lupine Linux
  - Specialization
  - System Call Overhead Elimination
  - Putting it together
- Evaluation
- Discussion
- Related Work

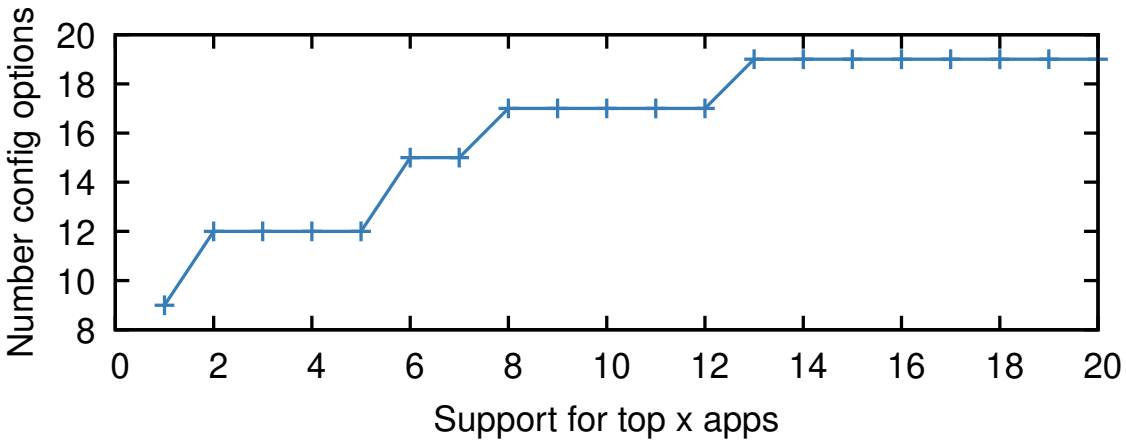


Evaluation setup

- Machine setup
  - CPU: Intel(R) Xeon(R) CPU E3-1270 v6 @ 3.80GHz
  - Mem: 16 GB
- VM setup
  - Hypervisor : firecracker
  - 1 VCPU, 512 MB Mem
  - Guest: Linux 4.0 with and without KML patches

Configuration Diversity

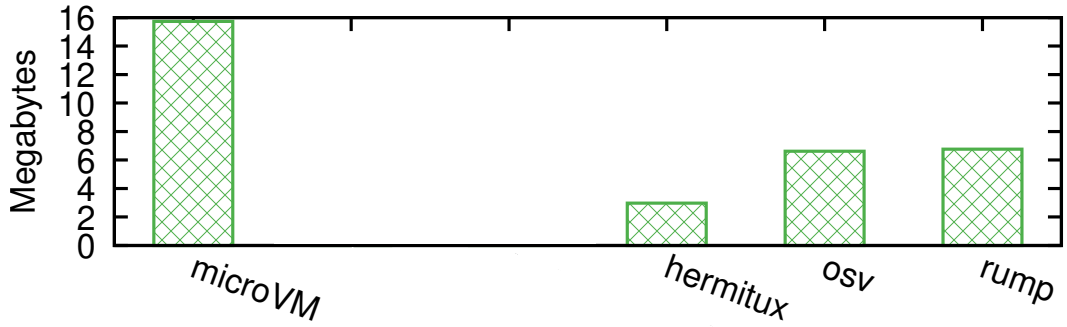
- Manually determined app-specific configurations
- 20 top apps on Docker hub (83% of all downloads)
- Only 19 configuration options required to run all 20 applications: ***lupine-general***



Name	Downloads	Description	# Options atop *lupine-base*
nginx	1.7	Web server	13
postgres	1.6	Database	10
httpd	1.4	Web server	13
node	1.2	Language runtime	5
redis	1.2	Key-value store	10
mongo	1.2	NOSQL database	11
mysql	1.2	Database	9
traefik	1.1	Edge router	8
memcached	0.9	Key-value store	10
hello-world	0.9	C program "hello"	0
mariadb	0.8	Database	13
golang	0.6	Language runtime	0
python	0.5	Language runtime	0
openjdk	0.5	Language runtime	0
rabbitmq	0.5	Message broker	12
php	0.4	Language runtime	0
wordpress	0.4	PHP/mysql blog tool	9
haproxy	0.4	Load balancer	8
influxdb	0.3	Time series database	11
elasticsearch	0.3	Search engine	12

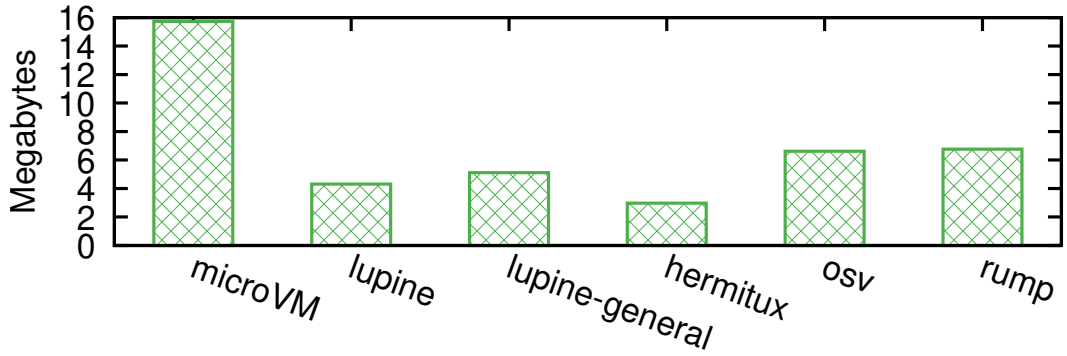
Table 3. Top twenty most popular applications on Docker Hub (by billions of downloads) and the number of additional configuration options each requires beyond the *lupine-base* kernel configuration. [9]

Kernel image size



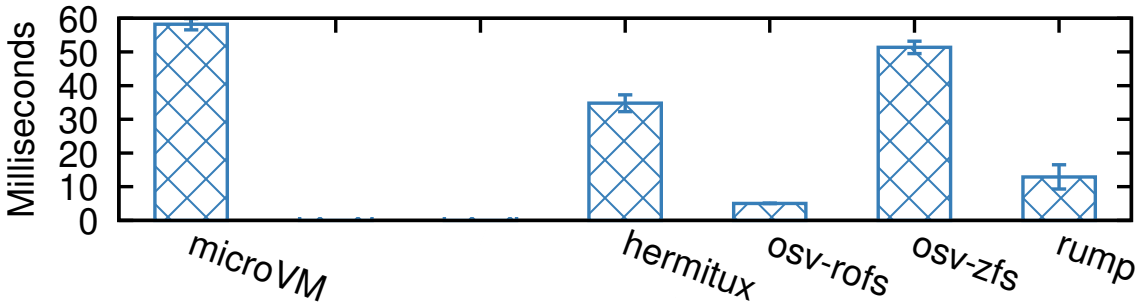
Kernel image size

- Configuration is effective
- 4 MB
- 27% (hello) - 33% of microvm
- Even *lupine-general* produces smaller images than Rump, OSv



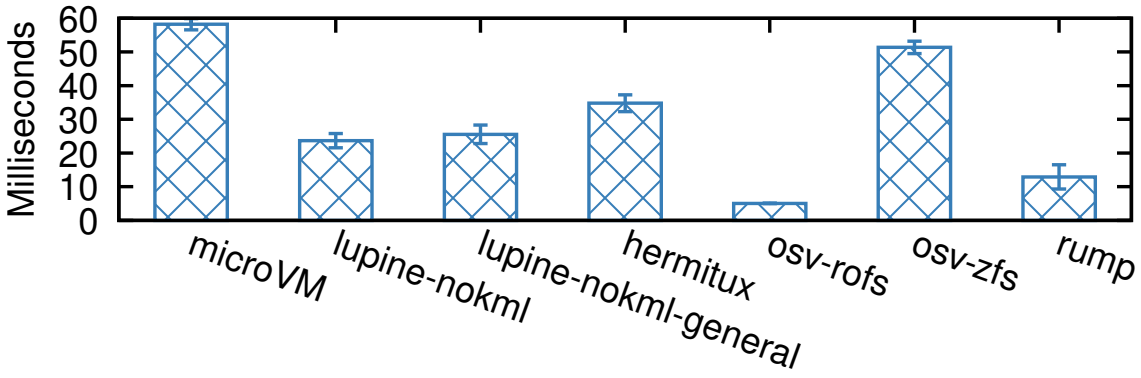
Boot time

- Measured via I/O port write from guest
- OSv boot heavily depends on FS choice



Boot time

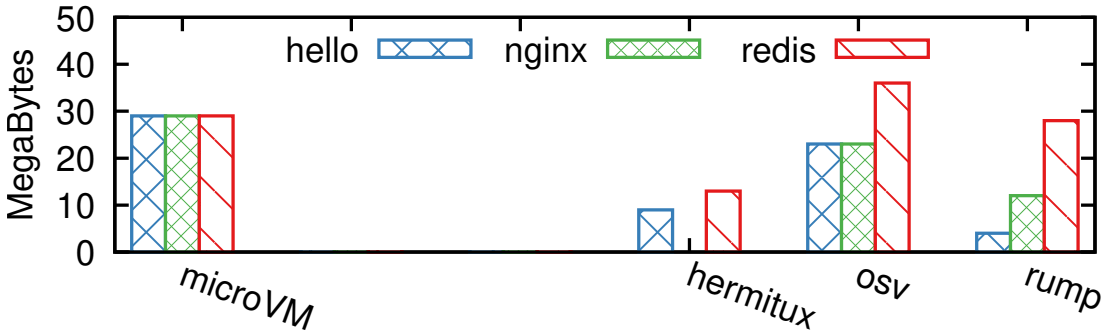
- Measured via I/O port write from guest
- OSv boot heavily depends on FS choice
- Lupine boot time without KML*
- Even *lupine-general* boots faster than Hermitux, OSv



*KML incompatibility with CONFIG_PARAVIRT

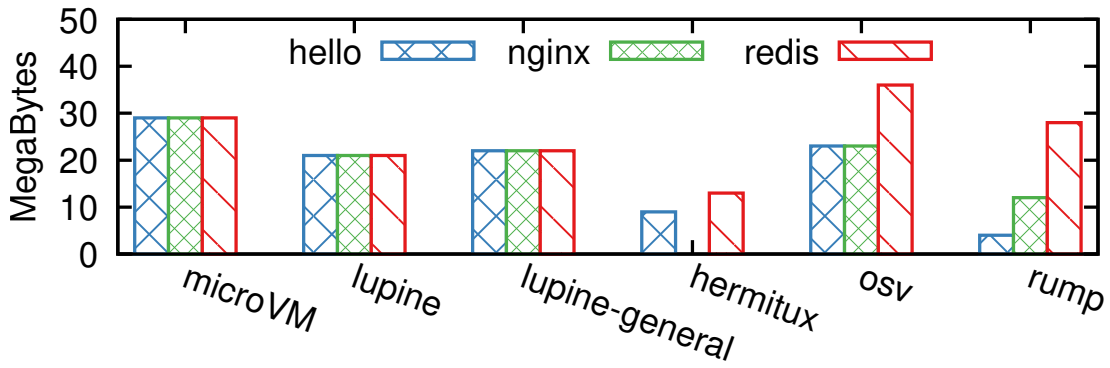
Memory Footprint

- Repeatedly tested app with decreasing memory allotment
- Choice of apps limited by unikernels



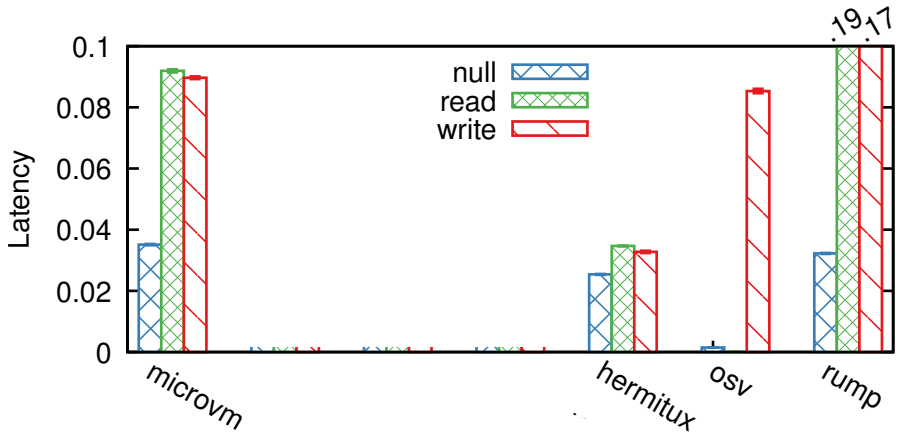
Memory Footprint

- Repeatedly tested app with decreasing memory allotment
- Choice of apps limited by unikernels
- No variation in lupine: lazy loading makes binary size irrelevant



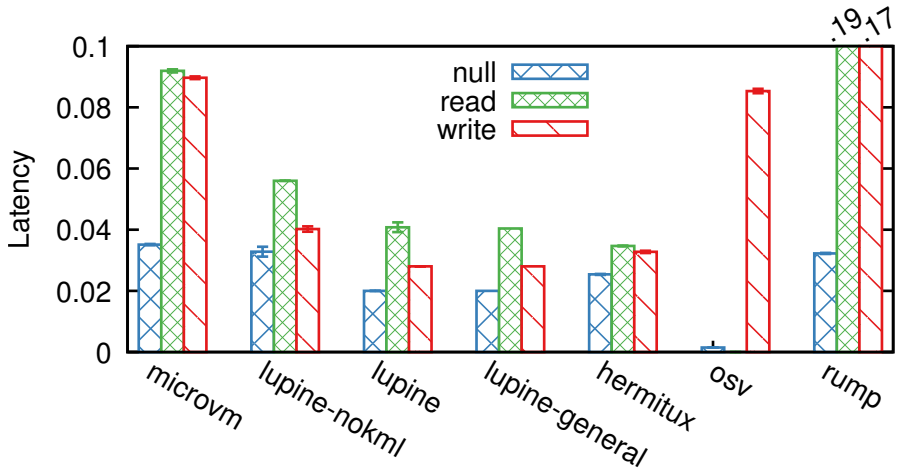
System call latency microbenchmark

- Lmbench



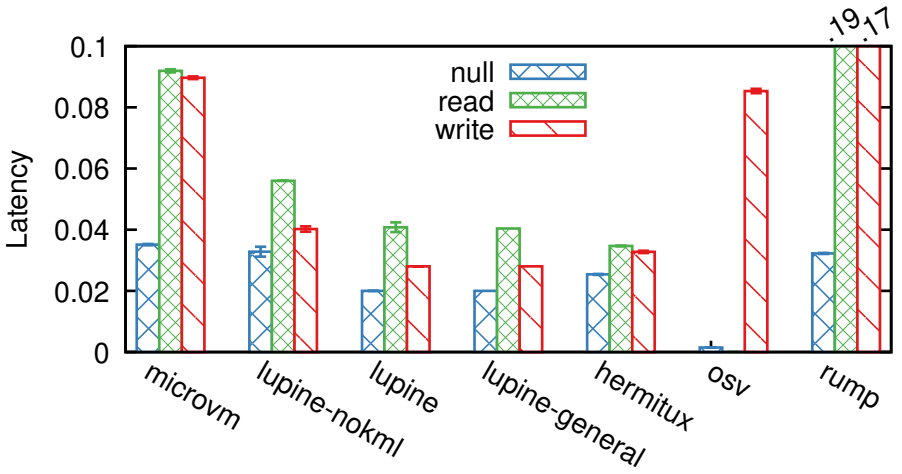
System call latency microbenchmark

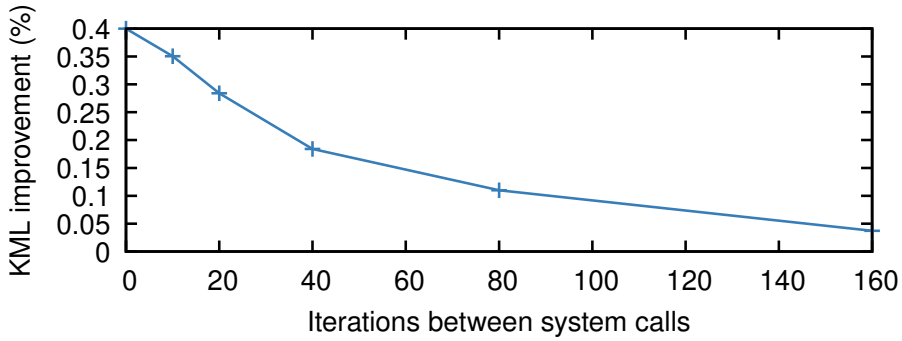
- Lmbench
- 56% improvement over microvm from specialization



System call latency microbenchmark

- Lmbench
- 56% improvement over microvm from specialization
- Additional 40% from KML
- KML benefit vanishes quickly in more realistic workloads





Application performance

- Throughput normalized to microVM
- Application choice limited by unikernels
- Lupine outperforms microVM by up to 33%
- Linux implementation is highly optimized

Name	redis-get	redis-set	nginx-conn	nginx-sess
microVM	1.00	1.00	1.00	1.00
lupine-general	1.19	1.20	1.29	1.15
lupine	**1.21**	**1.22**	**1.33**	**1.14**
lupine-tiny	1.15	1.16	1.23	1.11
lupine-nokml	1.20	1.21	1.29	1.16
lupine-nokml-tiny	1.13	1.13	1.21	1.12
hermitux	.66	.67		
osv			.87	.53
rump	.99	.99	1.25	.53

Table 4. Application performance normalized to microVM.

Takeaways

- **Specialization is important**:
  - 73% smaller image size, 59% faster boot time, 28% lower memory footprint and 33% higher throughput than the state-of-the-art VM
- **Specialization per application may not be**:
  - 19 options (lupine-general) cover at least 83% of downloaded apps with at most 4% reduction in performance
- **System call overhead elimination may not be:**
  - only 4% improvement for macrobenchmark, unlike 40% for microbenchmarks

- **Lupine avoids common pitfalls:** has support for unmodified Linux applications, optimized implementation

Lupine is still Linux

- **Graceful degradation** of unikernel properties

- Fork crashes unikernels, not Lupine
- Virtually no overhead to support multiple address spaces
  - Especially for control processes
- At worst 8% overhead to support multiple processors

IBM

Unachieved unikernel benefits

- Language-based unikernel benefits
  - Powerful static analysis / whole-system optimization

- Some unikernels (e.g., Solo5-based) have been proven to run on a thinner unikernel monitor interface
  - Potentially better security, debugging opportunities, unikernel as process, etc.
  - Linux does not (yet)

Related work

- Unikernel-like work that leverages Linux
  - LightVM (TinyX): VMs can be as light as containers
  - X-Containers: Xen paravirt for Linux to be a libOS
  - UKL: modify Linux build to include kernel call to application main

- Linux configuration studies
  - Alharthi et al.: 89% of 1530 studied vulnerabilities nullified via config specialization
  - Kurmus et al.: 50-85% of attack surface reduction via configuration

Getting Lupine benefits into community

- Most benefits are achieved through specialized config
  - But *lupine-general.config* can run top 20 Docker containers

- Challenges/risks
  - How do we know lupine-general is general enough?
    - Research needed: discovery vs. failover vs. ?
  - Tension with container ecosystem (kata agent → more general kernel config)
    - Research needed: kernel configuration-aware design?

IBM

Lupine Conclusion

- Unikernels and library OS seem attractive
- But trying to achieve generality/POSIX in unikernels is not worth it

- Linux can already behave like a unikernel!
  - Specialization via configuration
  - Can maintain Linux community and engineering effort in past three decades

- Can we apply these techniques to virtualization-enabled containers?

Thank you!

- https://github.com/hckuo/Lupine-Linux
- https://nabla-containers.github.io/

- djwillia@us.ibm.com