CS6465: Advanced Operating System Implementation

Lecture 13 – Performance, Containers, Virtualization

Anton Burtsev

December 2024

Data and Control Plane Separation

Linux network packet

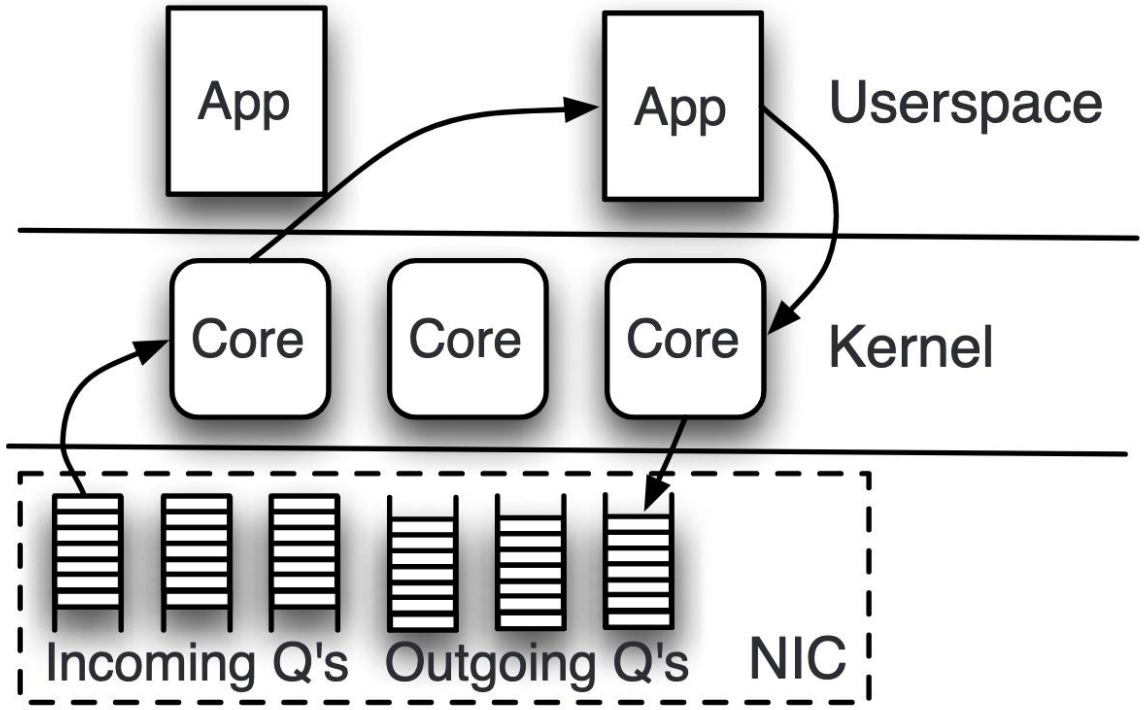


Figure 1: Linux networking architecture and workflow.

https://www.usenix.org/system/files/conference/osdi14/osdi14-paper-peter_simon.pdf

Linux network packet

		Linux			
		Receiver running		CPU idle	
Network stack	in	1.26	(37.6%)	1.24	(20.0%)
	out	1.05	(31.3%)	1.42	(22.9%)
Scheduler		0.17	(5.0%)	2.40	(38.8%)
Copy	in	0.24	(7.1%)	0.25	(4.0%)
	out	0.44	(13.2%)	0.55	(8.9%)
Kernel crossing	return	0.10	(2.9%)	0.20	(3.3%)
	syscall	0.10	(2.9%)	0.13	(2.1%)
Total		3.36	($\sigma = 0.66$)	6.19	($\sigma = 0.82$)

- 2000 cycles to send a packet
- Network stack costs: packet processing at the hardware, IP, and UDP layers
- Scheduler overhead: waking up a process (if necessary), selecting it to run, and context switching to it
- Kernel crossings: from kernel to user space and back
- Copying of packet data: from the kernel to a user buffer on receive, and back on send

https://www.usenix.org/system/files/conference/osdi14/osdi14-paper-peter_simon.pdf

Arrakis

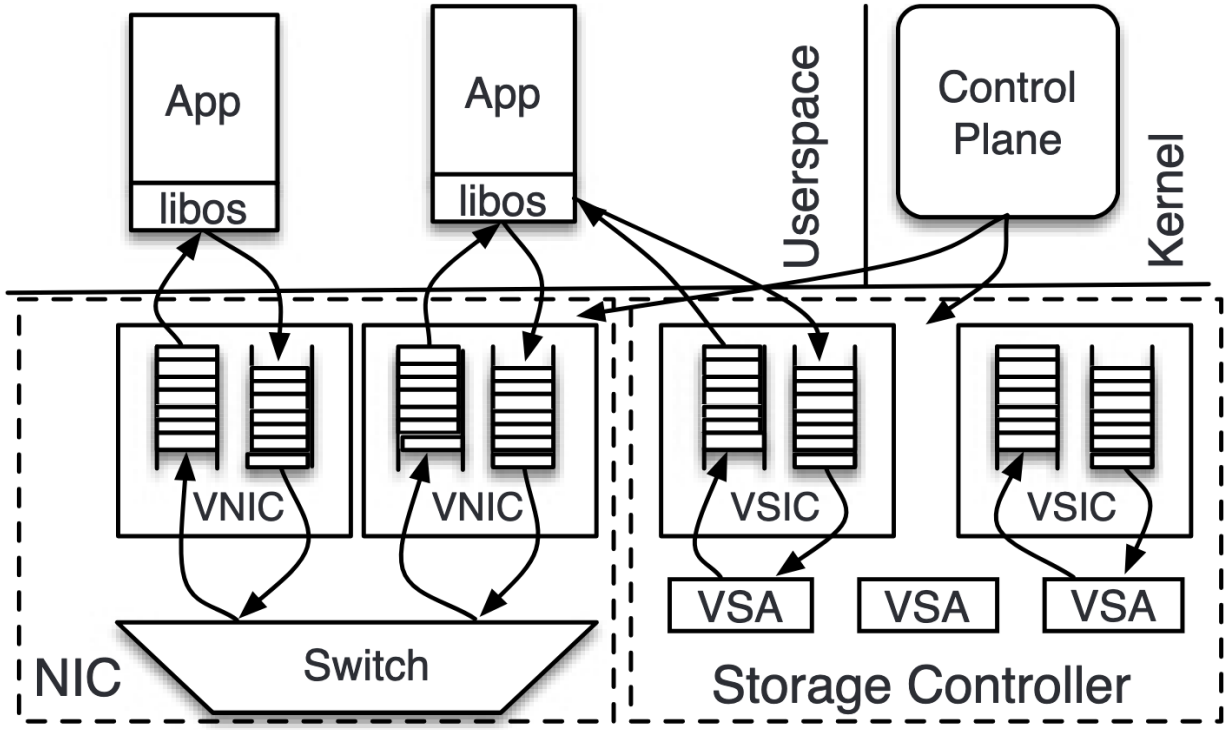


Figure 3: Arrakis architecture. The storage controller maps VSAs to physical storage.

- 2000 cycles to send a packet
- Network stack costs: packet processing at the hardware, IP, and UDP layers
- Scheduler overhead: waking up a process (if necessary), selecting it to run, and context switching to it
- Kernel crossings: from kernel to user space and back
- Copying of packet data: from the kernel to a user buffer on receive, and back on send

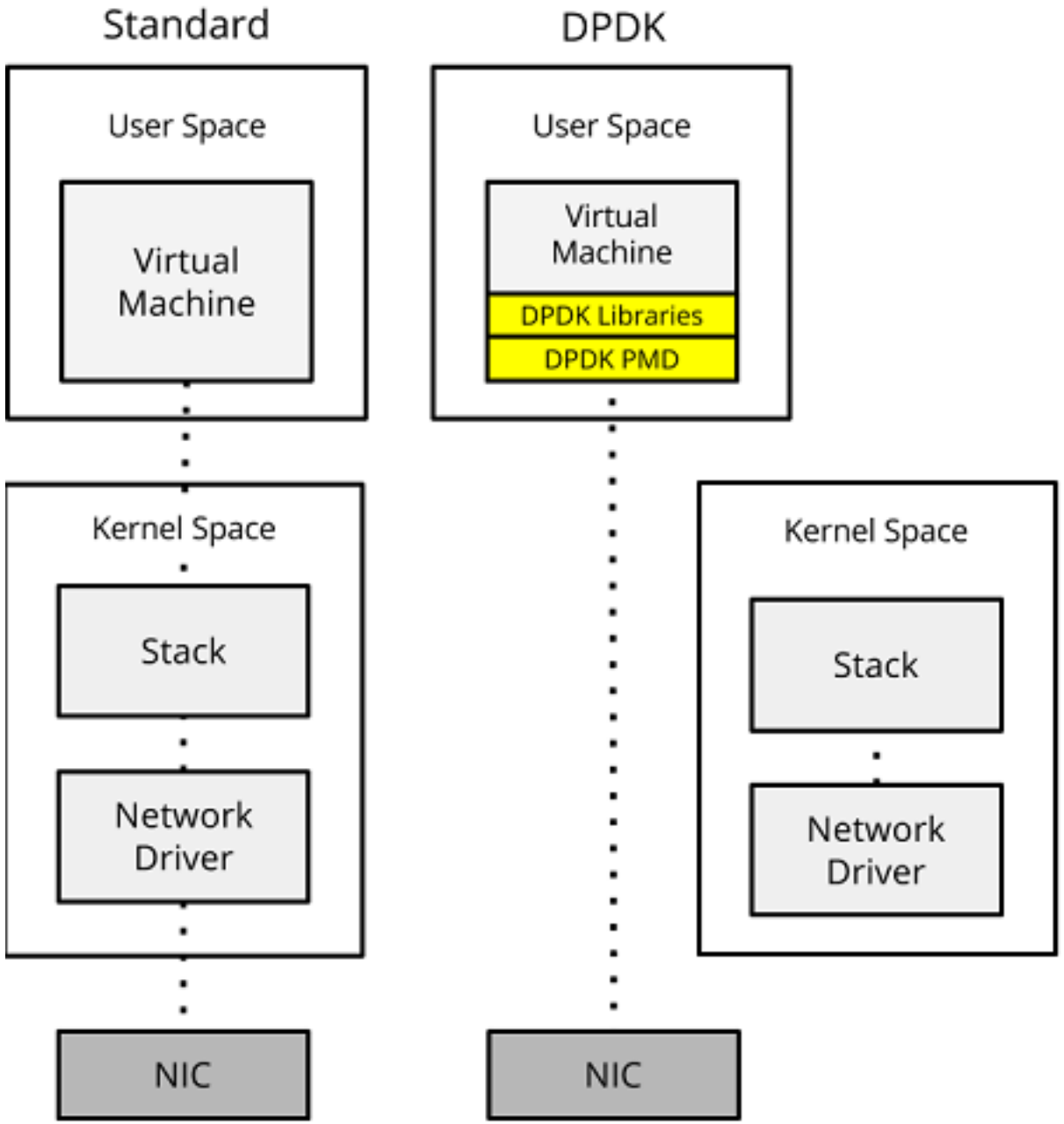
https://www.usenix.org/system/files/conference/osdi14/osdi14-paper-peter_simon.pdf

Arrakis

		Linux				Arrakis			
		Receiver running		CPU idle		Arrakis/P		Arrakis/N	
Network stack	in	1.26	(37.6%)	1.24	(20.0%)	0.32	(22.3%)	0.21	(55.3%)
	out	1.05	(31.3%)	1.42	(22.9%)	0.27	(18.7%)	0.17	(44.7%)
Scheduler		0.17	(5.0%)	2.40	(38.8%)	-		-	
Copy	in	0.24	(7.1%)	0.25	(4.0%)	0.27	(18.7%)	-	
	out	0.44	(13.2%)	0.55	(8.9%)	0.58	(40.3%)	-	
Kernel crossing	return	0.10	(2.9%)	0.20	(3.3%)	-		-	
	syscall	0.10	(2.9%)	0.13	(2.1%)	-		-	
Total		3.36	($\sigma = 0.66$)	6.19	($\sigma = 0.82$)	1.44	($\sigma < 0.01$)	0.38	($\sigma < 0.01$)

Table 1: Sources of packet processing overhead in Linux and Arrakis. All times are averages over 1,000 samples, given in μs (and standard deviation for totals). Arrakis/P uses the POSIX interface, Arrakis/N uses the native Arrakis interface.

https://www.usenix.org/system/files/conference/osdi14/osdi14-paper-peter_simon.pdf

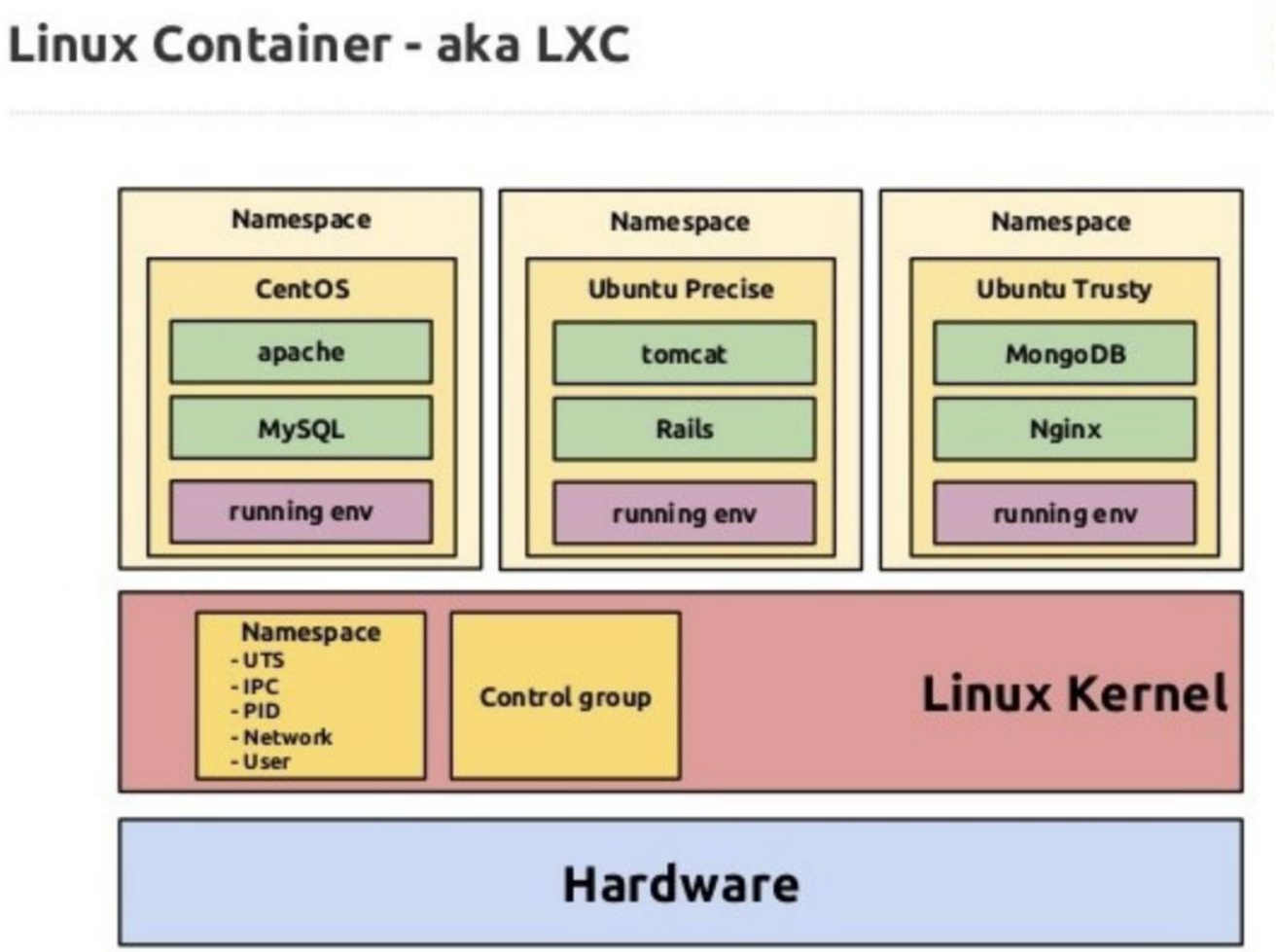


Kernel bypass in Linux: DPDK

- Data Plane Processing Kit

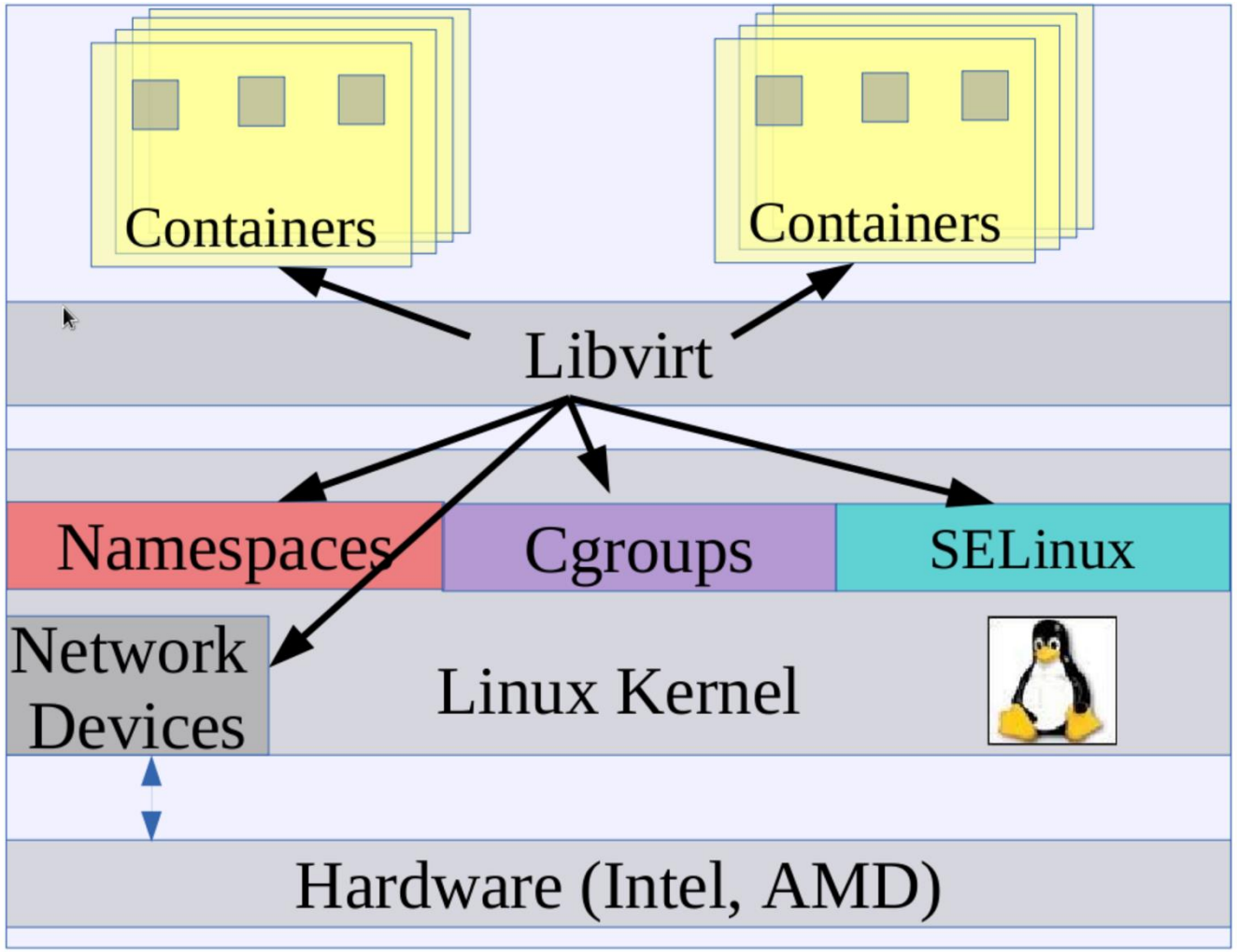
Linux Containers

OS-level virtualization



https://systron.net/blog/wp-content/uploads/2022/01/cgroups_namespaces.jpg

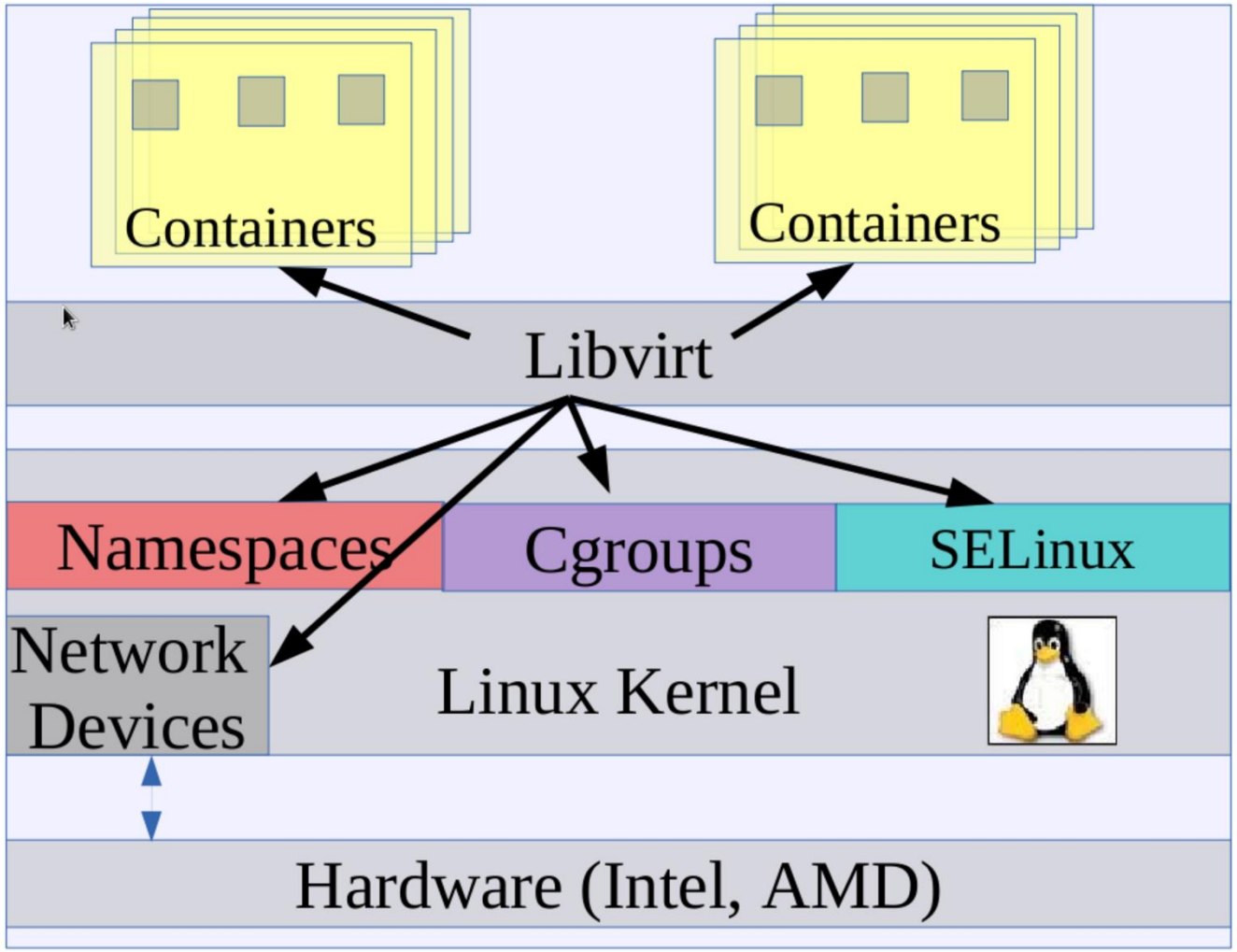
OS-level virtualization



- Namespaces:
  - Private view of resources
  - Evolution of chroot
  - Process trees
  - Domain and host names
  - User mounts
  - File names
  - Users (UID) and groups (GID)
- Control groups (cgroups)
  - Resource isolation
  - CPU time
  - Memory
  - Network I/O

https://systron.net/blog/wp-content/uploads/2022/01/cgroups_namespaces.jpg

OS-level virtualization



- SECCOMP
  - Secure computing
  - Restrict lists of system calls allowed for the app
- SELinux
  - More restrictions, e.g., an application (which can be compromised) has acces only to its own files

https://systron.net/blog/wp-content/uploads/2022/01/cgroups_namespaces.jpg

gVizor: better container isolation

gVizor Architecture

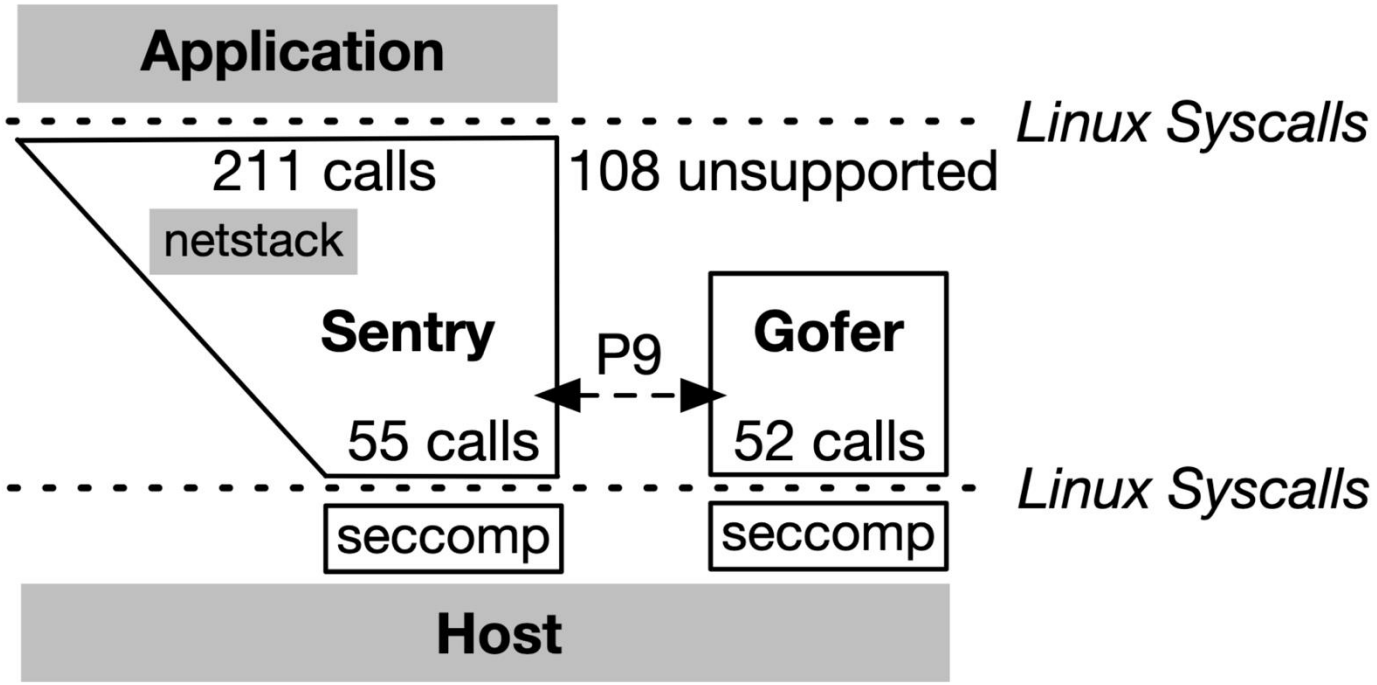


Figure 1: **gVisor Architecture.**

- Sentry
  - Kernel implemented in Go
  - 211 system calls
  - Small: 15MB footprint
- Can run in either
  - VT-x non-root ring 0, then a syscall from application is fast
  - Or as a user process on Linux (and use ptrace to route system calls)
  - Seccomp to limit system calls from Sentry to the kernel
- Gofer
  - Implements file access
  - Runs as host ring3
  - Requires a VM exit
  - GR0 (Guest Ring 0) to HR3 (Host Ring 3)

https://www.usenix.org/system/files/hotcloud19-paper-young.pdf

gVizor performance

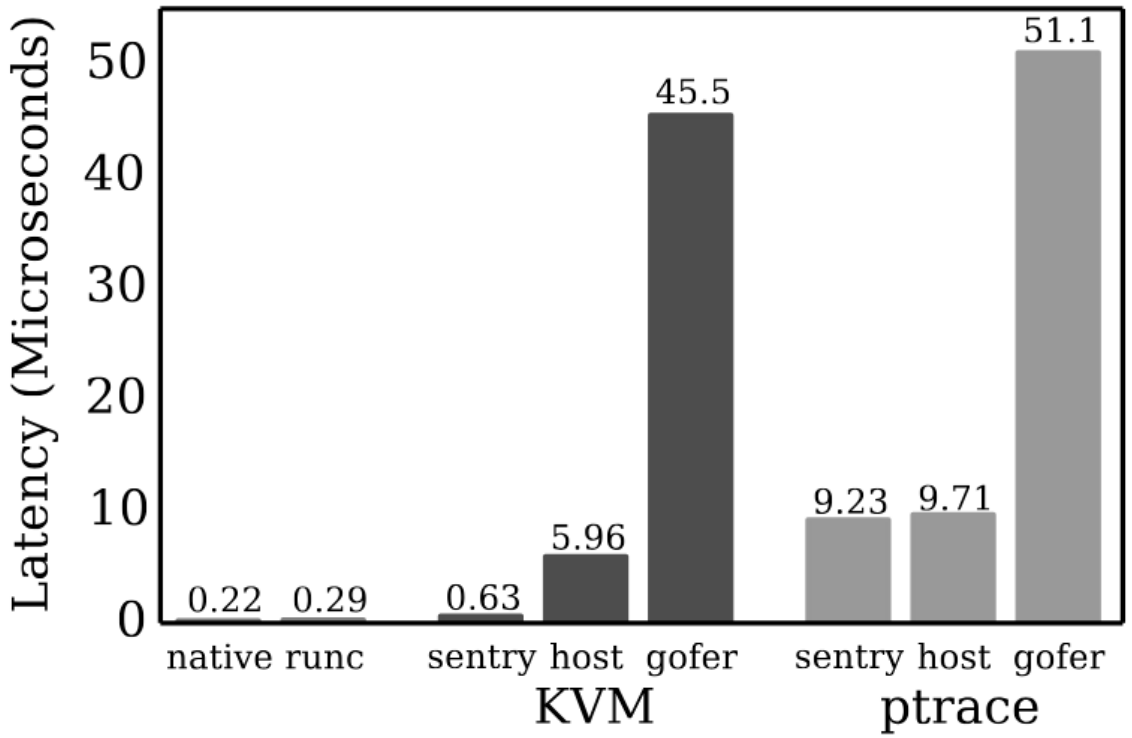


Figure 3: **System Call Overhead.** *The bars show the average latency for* `gettimeofday` *across 100M executions.*

gVizor performance

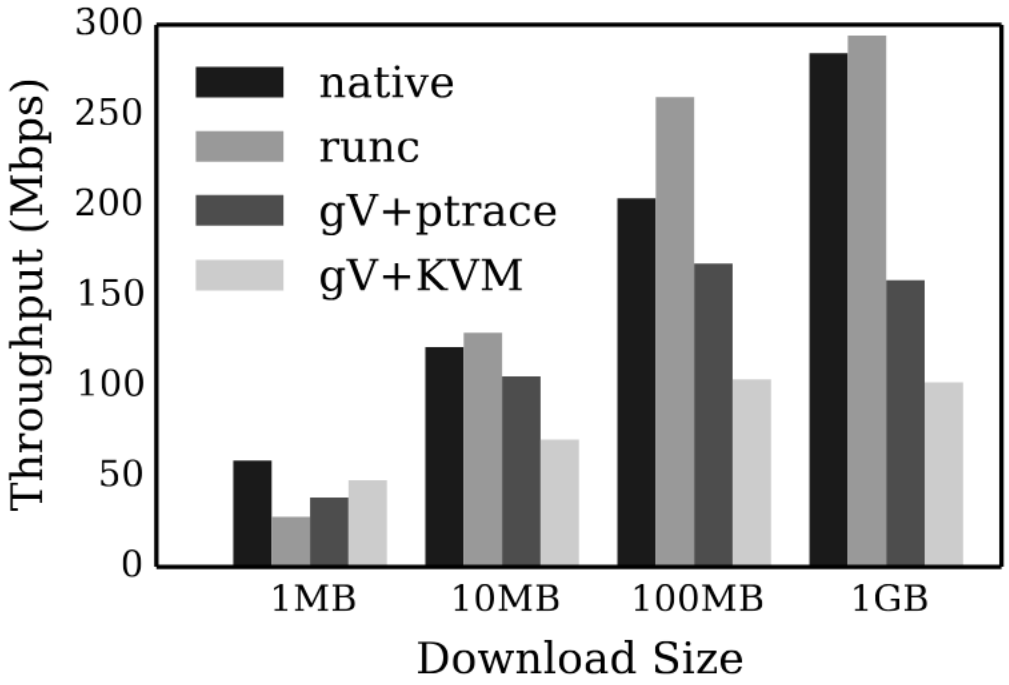


Figure 5: **Network Throughput.** *Throughput is measured by downloading files of varying sizes (x-axis) with* `wget`.

- runc is Docker

gVizor performance

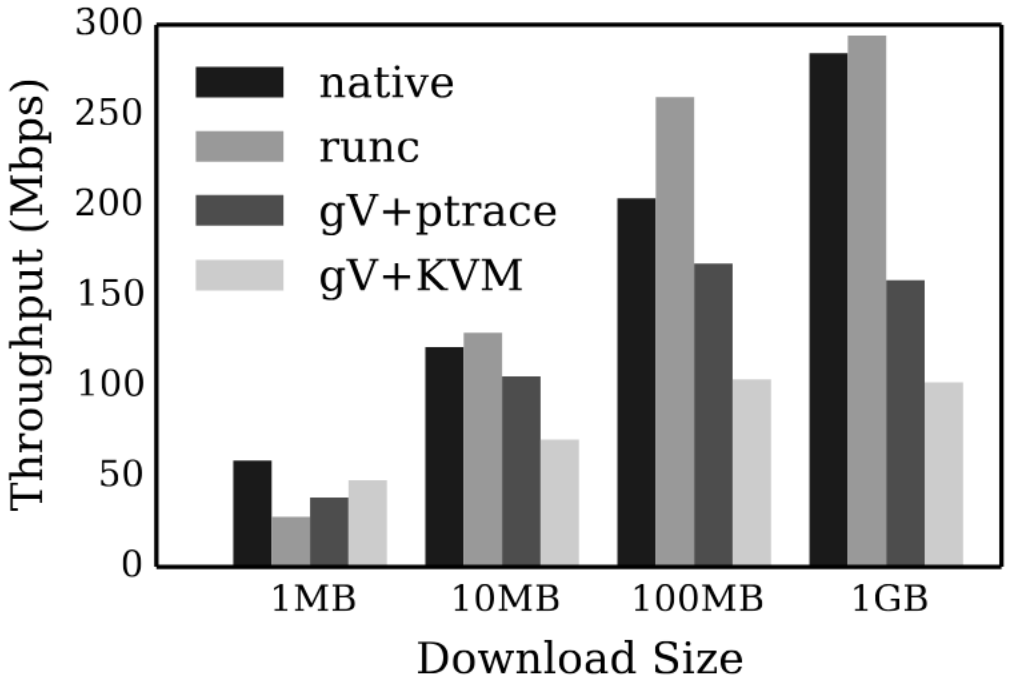


Figure 5: **Network Throughput.** *Throughput is measured by downloading files of varying sizes (x-axis) with* `wget`.

- runc is Docker

gVizor performance

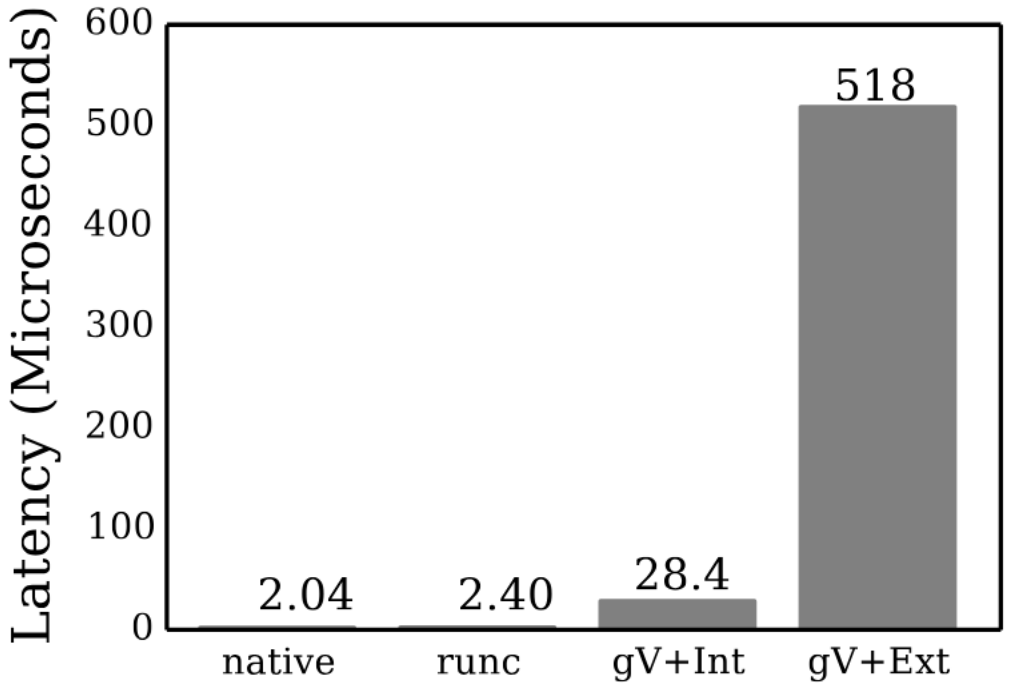


Figure 6: **Open and Close Latency.** *Results are an average over 100K consecutive accesses to the same file on native, runc, internal tmpfs (gV+Int), and external tmpfs (gV+Ext).*

- runc is Docker

Lightweight VMs

Mirage OS: Ocaml Language Runtime

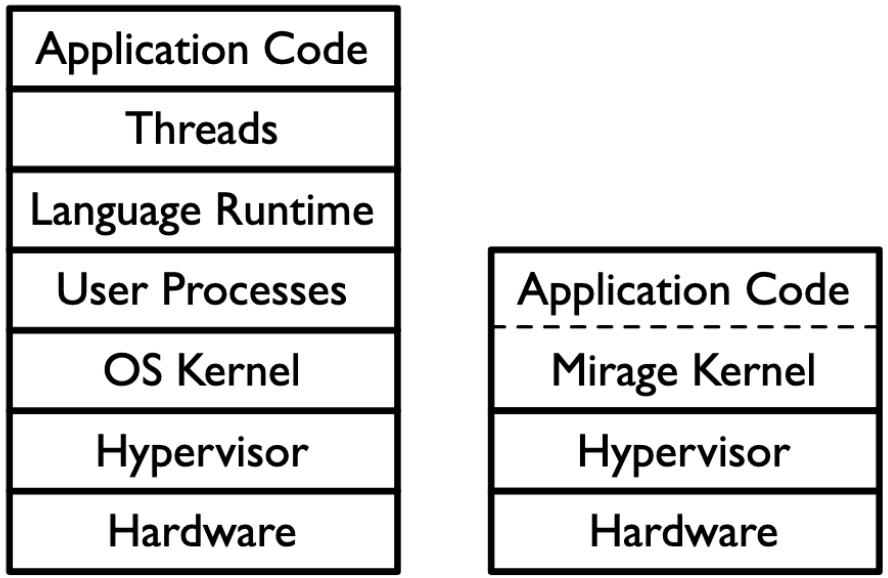


Figure 1: A conventional software stack (*left*) and the statically-linked Mirage approach (*right*).

- Mirage is a small kernel that uses language-based isolation of OCaml

https://www.usenix.org/legacy/event/hotcloud10/tech/full_papers/Madhavapeddy.pdf

Drawbridge: Library OS for Windows

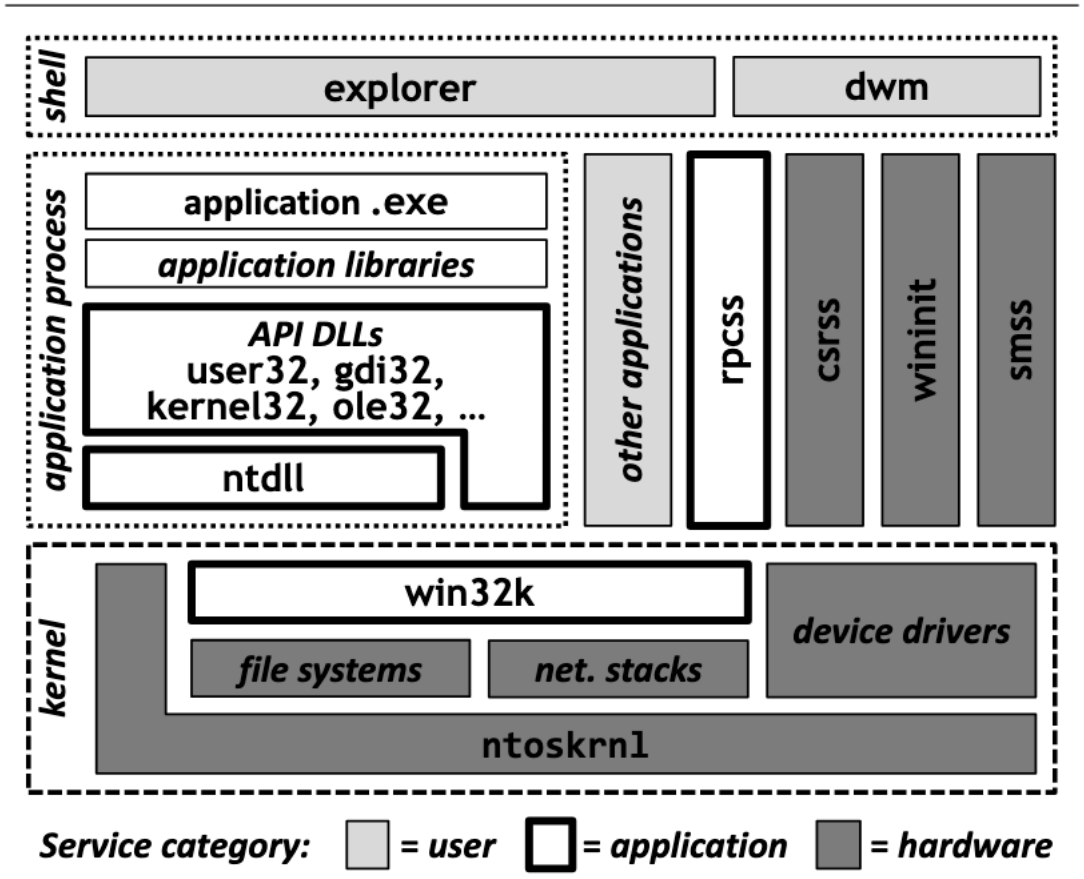


Figure 1. Windows 7 OS Architecture.

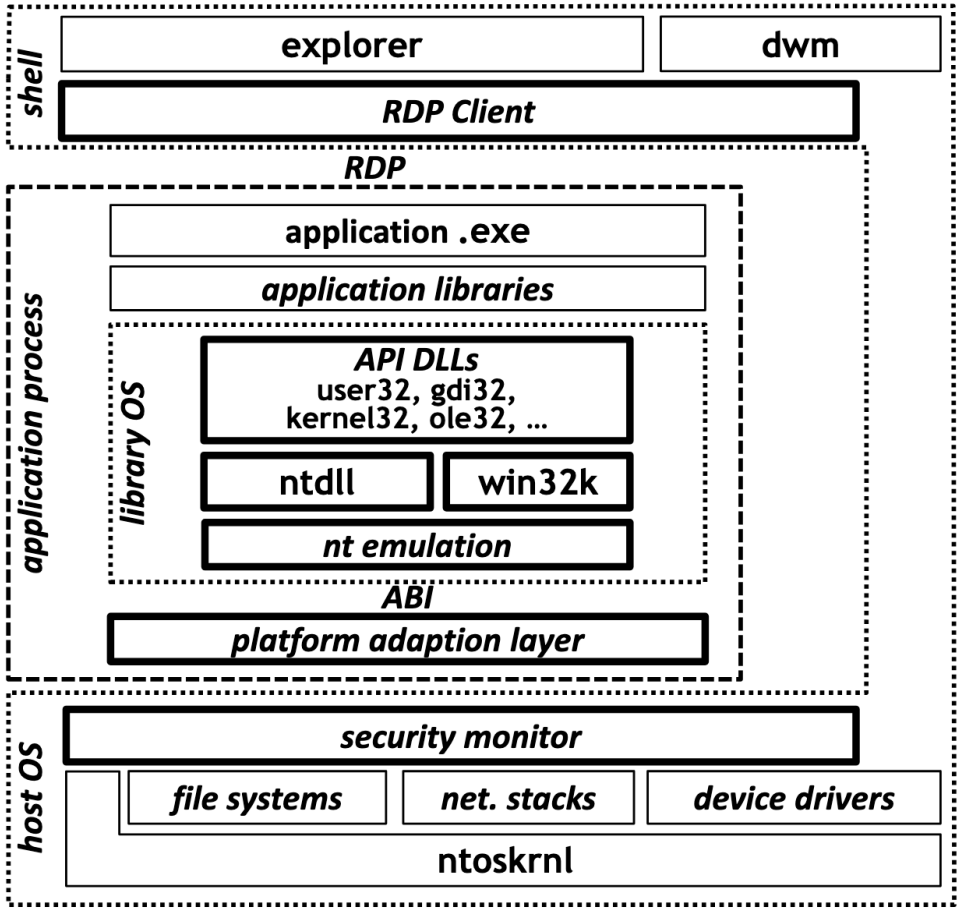
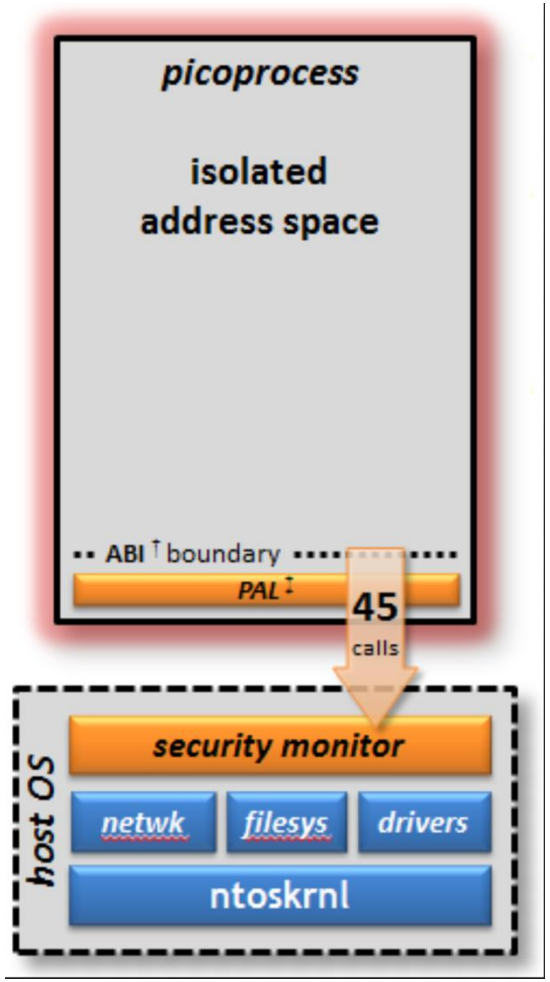


Figure 2. Drawbridge Architecture.

https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/asplos2011-drawbridge.pdf

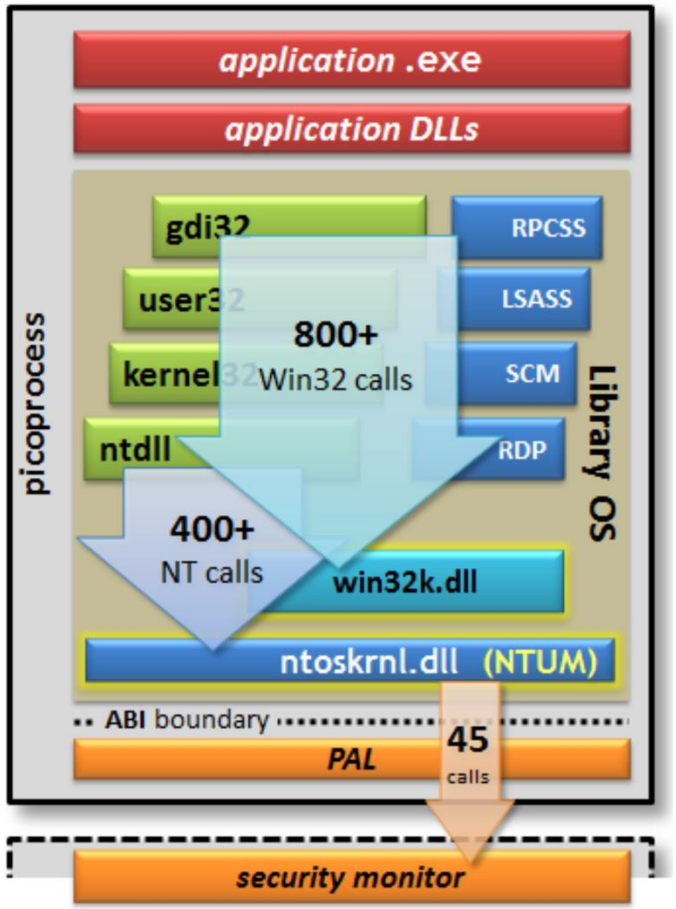
Drawbridge



- Drawbridge picoprocess

https://www.microsoft.com/en-us/research/project/drawbridge/

Drawbridge



A Windows Library OS in context

- Backward compatibility

https://www.microsoft.com/en-us/research/project/drawbridge/

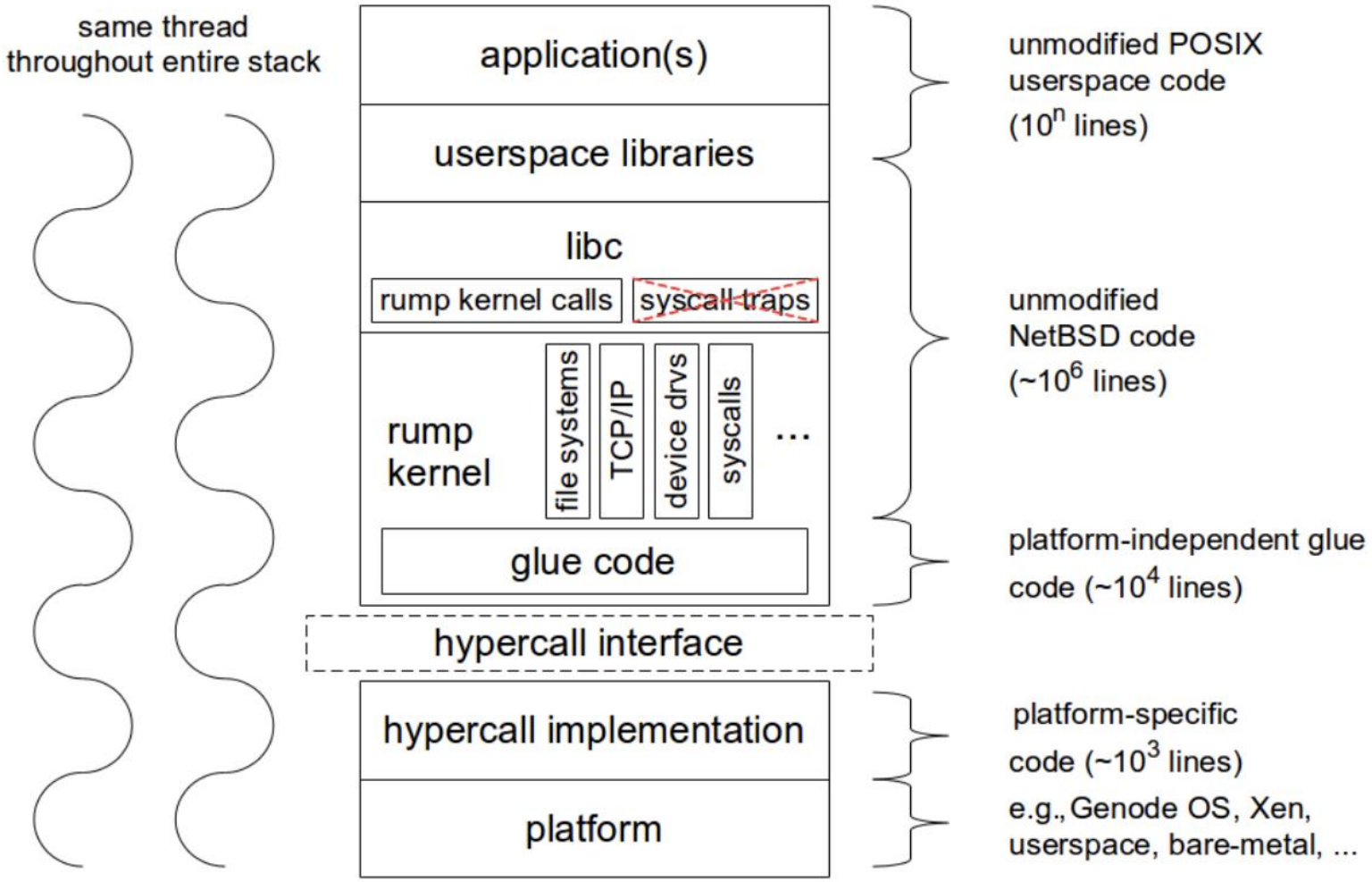


Figure 1: Rump kernels provide file system, network, and other driver support and run on bare metal systems or hypervisors by making use of a hypercall interface. In the depicted case, rump kernels provide the support necessary for running applications without requiring a full OS.

https://www.usenix.org/system/files/login/articles/login_1410_03_kantee.pdf

Rump kernel

- NetBSD that can run on top of POSIX, Xen, and even bare-metal

Thank you!