Arrakis

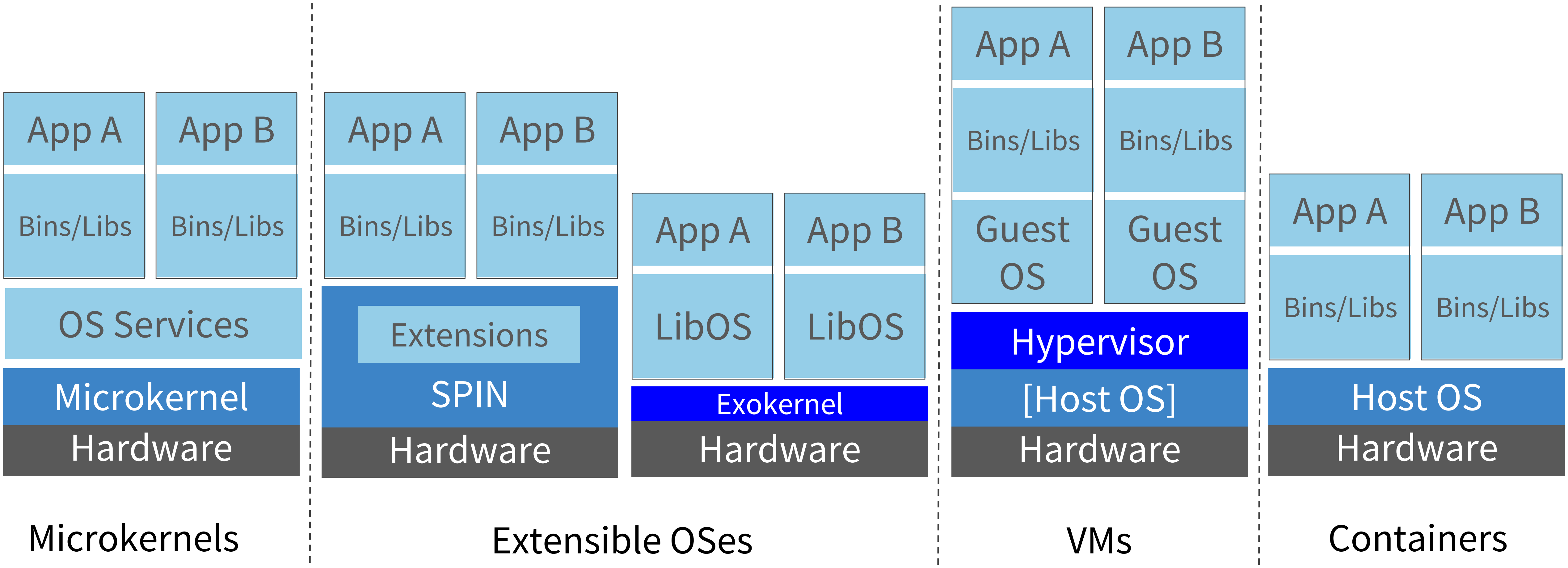
Emmett Witchel

CS380L

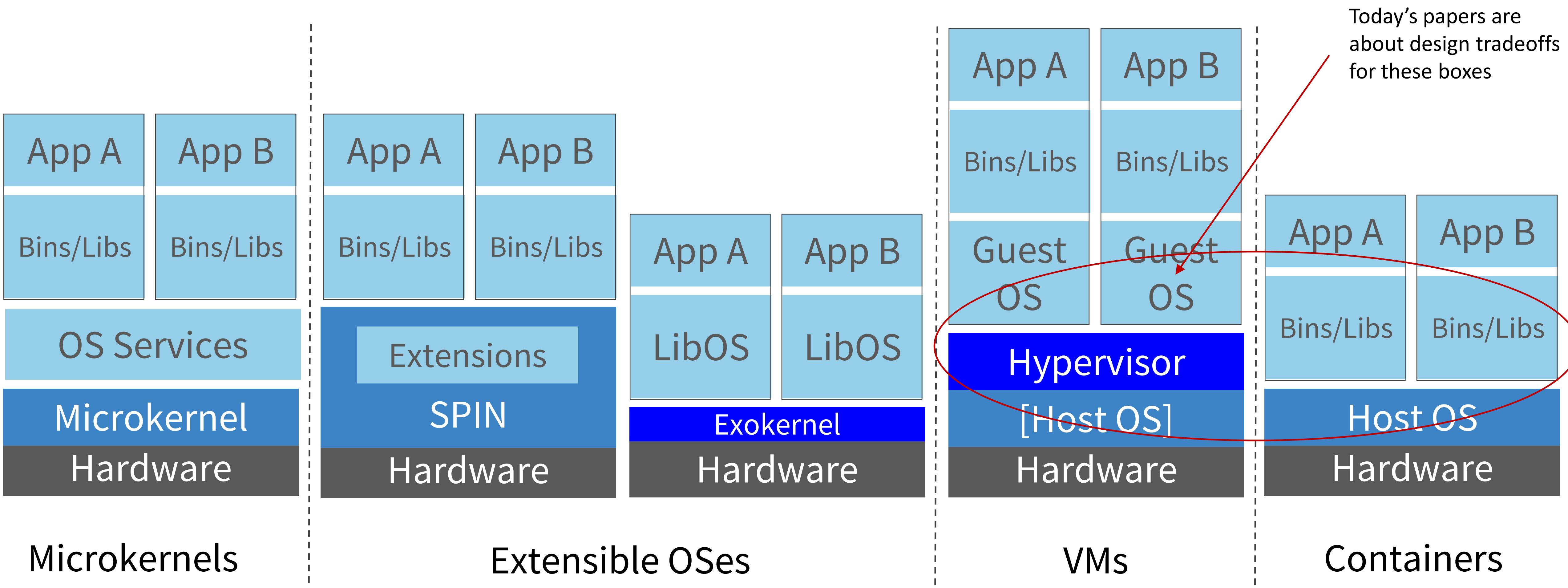
Arrakis faux quiz (pick 2, 5 min)

- Compare and contrast interposition techniques in ESX, Xen, Arrakis
- What is the difference between Arrakis/P and Arrakis/N?
- How much control does the OS have over I/O scheduling in Arrakis?
- How would you implement a MLFQ I/O scheduler in Arrakis? An priority aging I/O scheduler?
- What new abstractions does Arrakis suggest for user-space I/O?
- Compare/contrast Arrakis with Exokernel.
- How are file systems shared across processes in Arrakis?

Box drawing Potpourri: OSes, VMs, Containers



Box drawing Potpourri: OSes, VMs, Containers



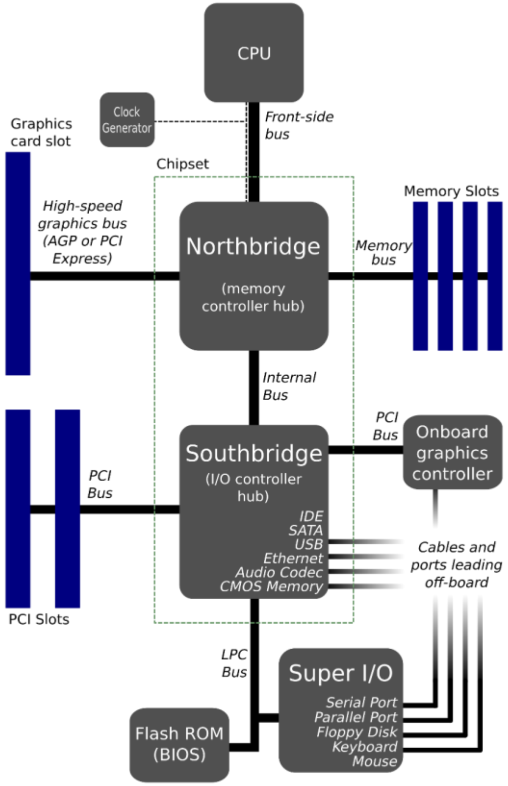
Arrakis

- Background
  - I/O Architecture
  - DMA
  - I/O Virtualization
  - SR-IOV
- Arrakis
  - Some slides adapted (with thanks!) from:

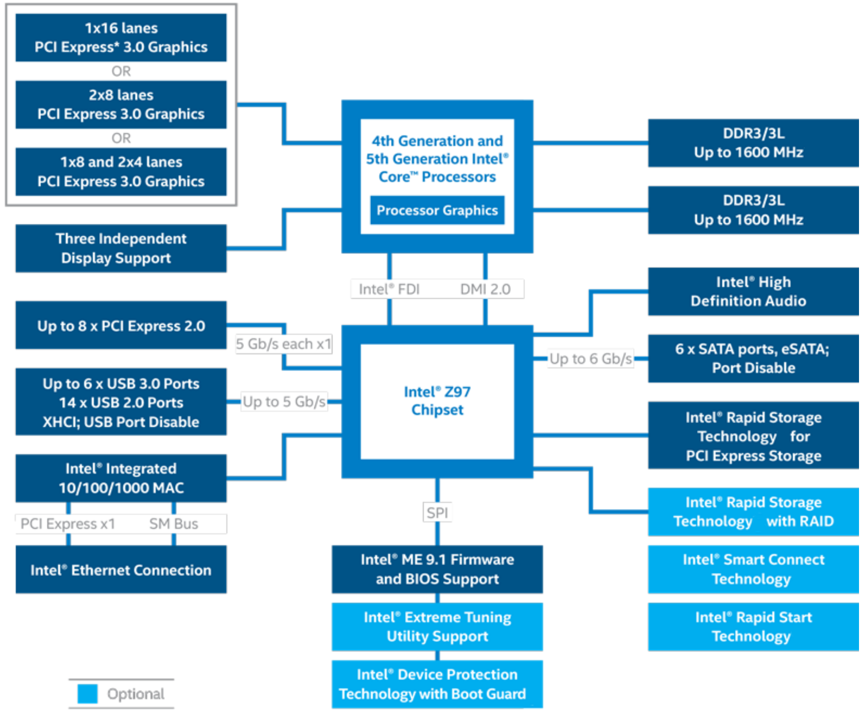
    web.eecs.umich.edu/~mosharaf/Slides/EECS582/2016.../101916-JimmyArrakis.pptx

    Some slide materials adapted from Simon's OSDI talk

I/O Architecture



Traditional
w/ Northbridge/Southbridge



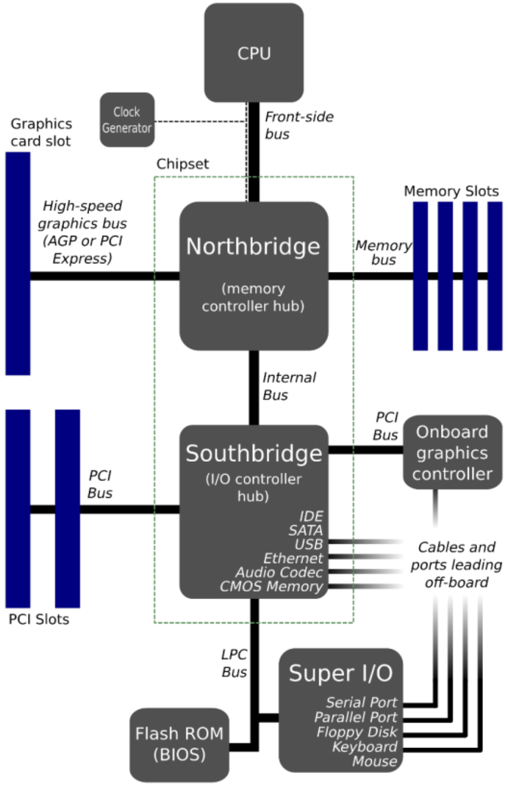
Modern:
QPI or HyperTransport

PCIe 4.0 reaches 2 GB/s per lane

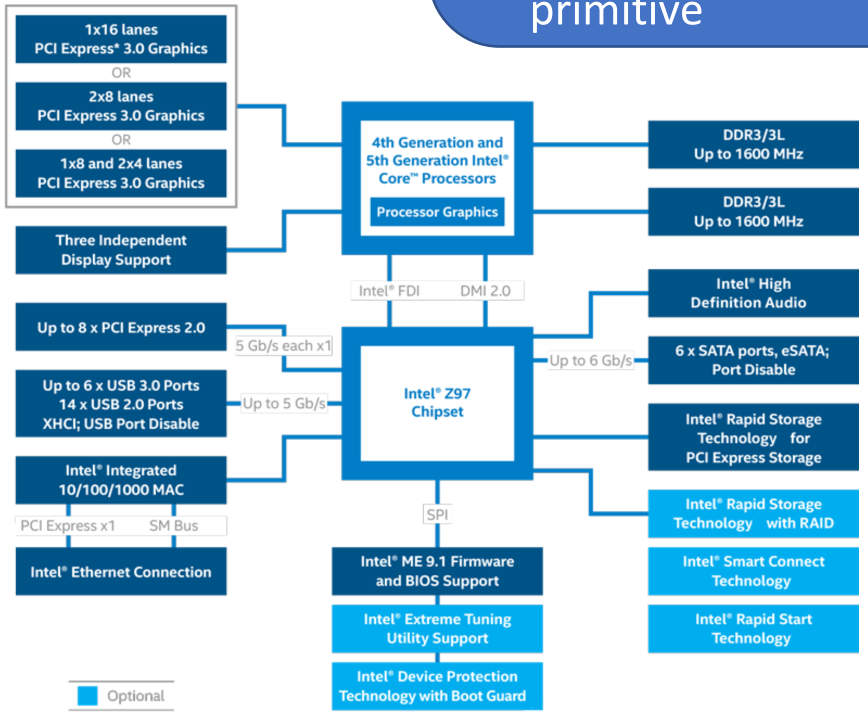
I/O Architecture

Things to observe:

- Significant evolution of path from devices to memory
- Significant diversity in I/O architectures/chipsets
- DMA dominant data movement primitive



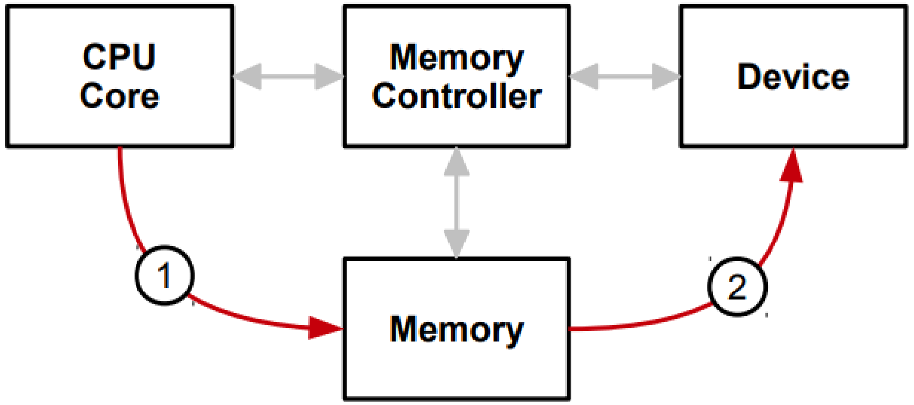
Traditional
w/ Northbridge/Southbridge

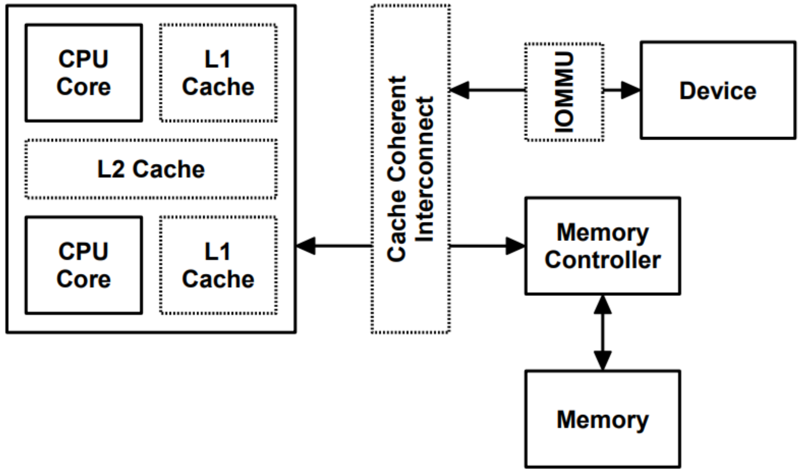


Modern:
QPI or HyperTransport

PCIe 4.0 reaches 2 GB/s per lane

DMA evolution



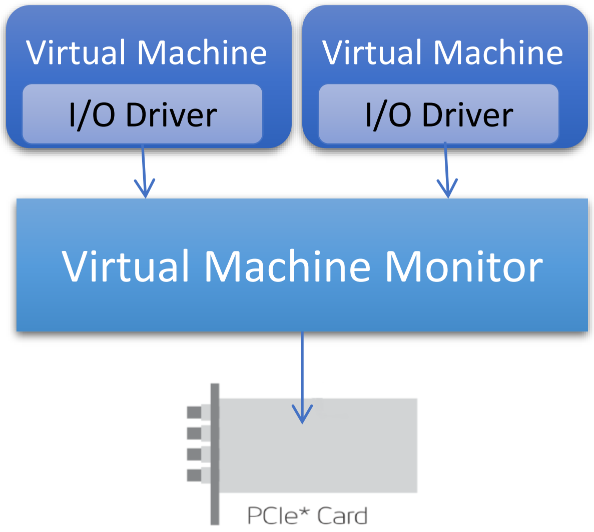


- Device can read/write memory
- CPU sets up DMA transfers
- Device uses ***physical*** address space
  - When is/isn't that OK?

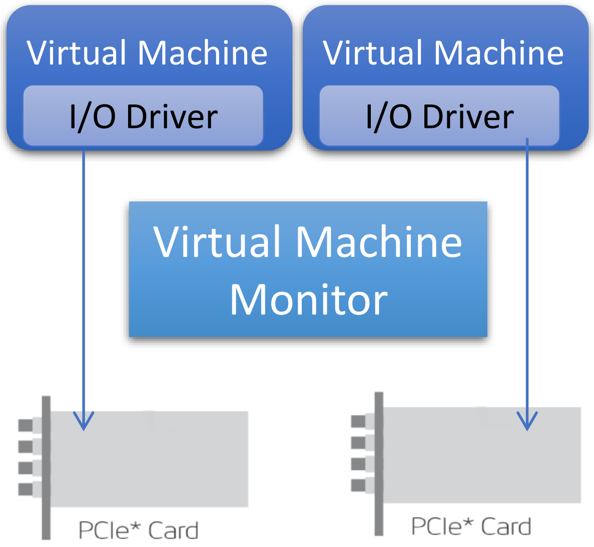
- Device uses translated ***device*** addresses
  - DMA mappings must be configured
  - What mappings to use?
  - Who configures IOMMU mappings?
  - How many device address spaces?
- Simple if OS mediates access to device
  - How to virtualize in VMM?

I/O Virtualization Techniques

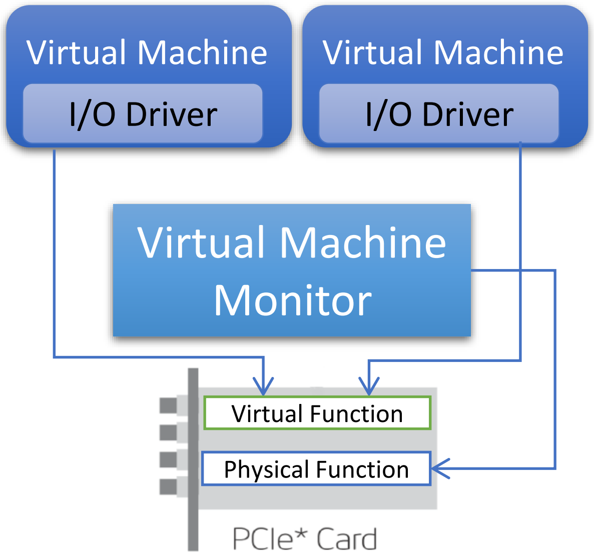
- A - Software only
- B - Directed I/O **(enhance performance)**
- C – Directed I/O and Device Sharing **(resource saving)**



A – Software only

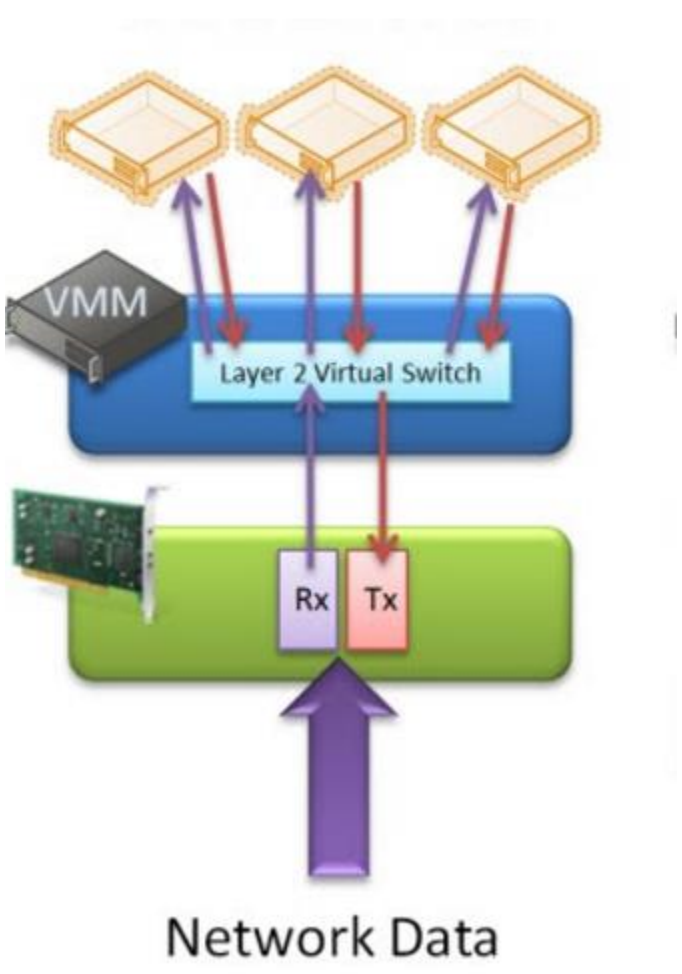


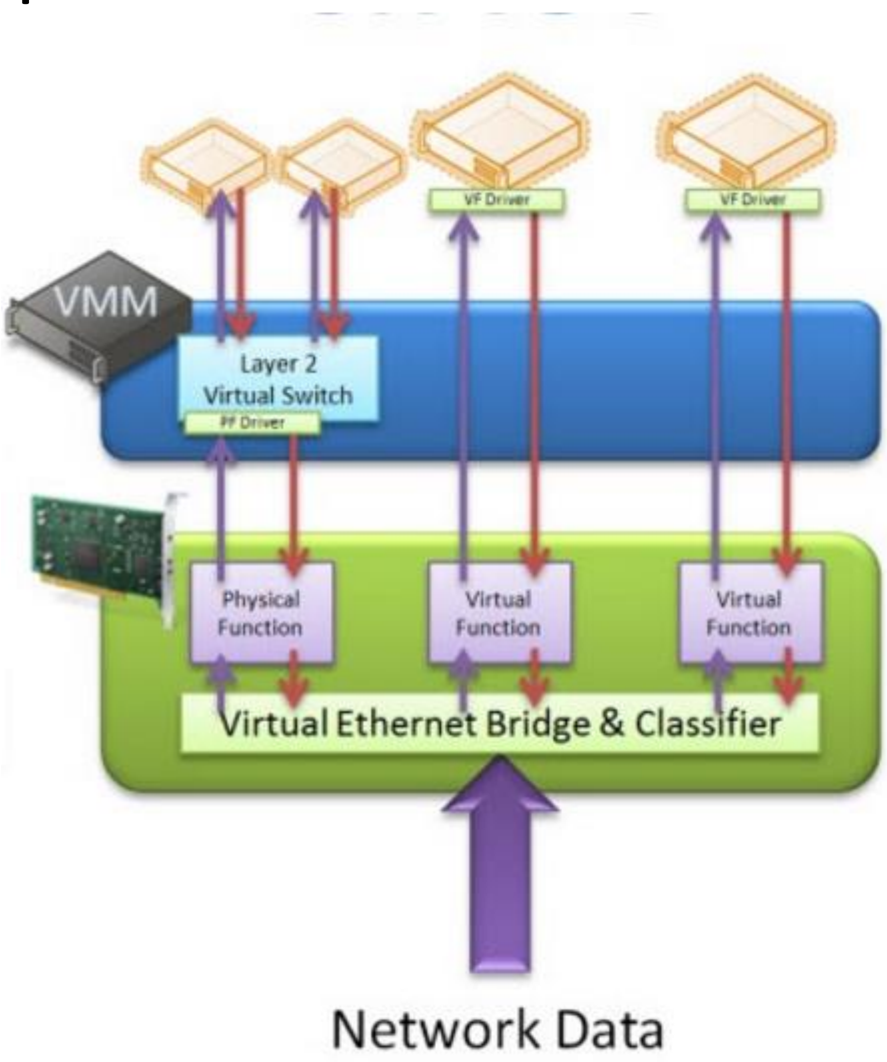
B – Directed I/O



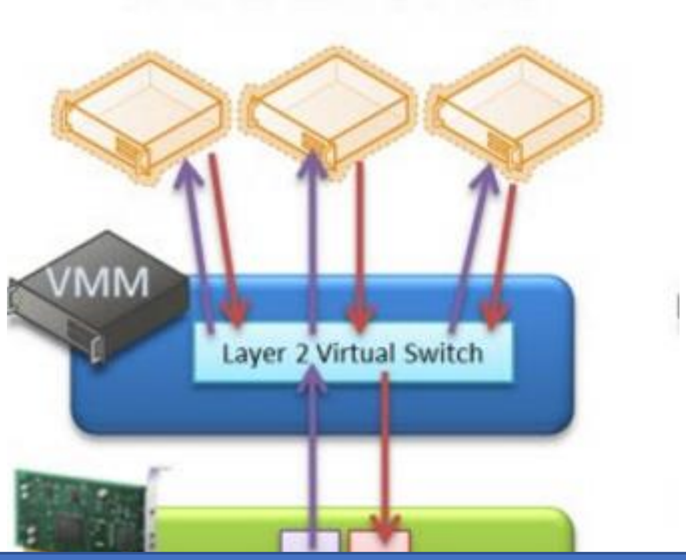
C – Directed I/O & Device Sharing

I/O Virtualization Techniques



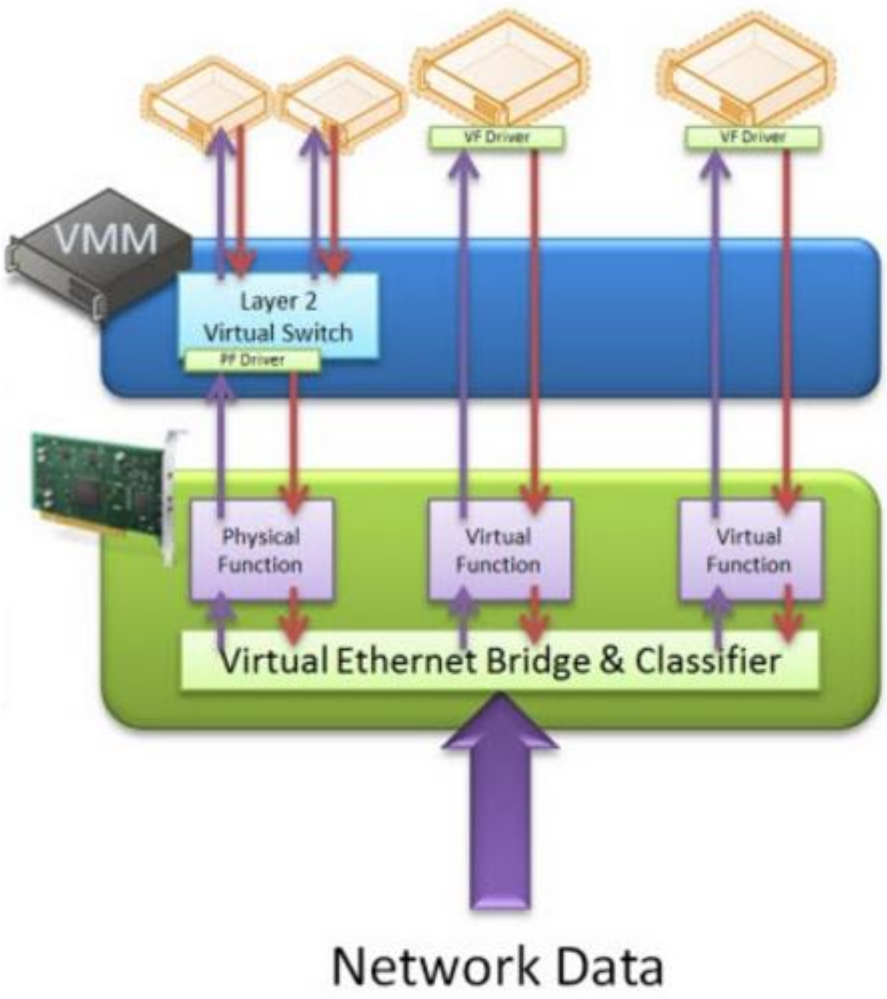


I/O Virtualization Techniques

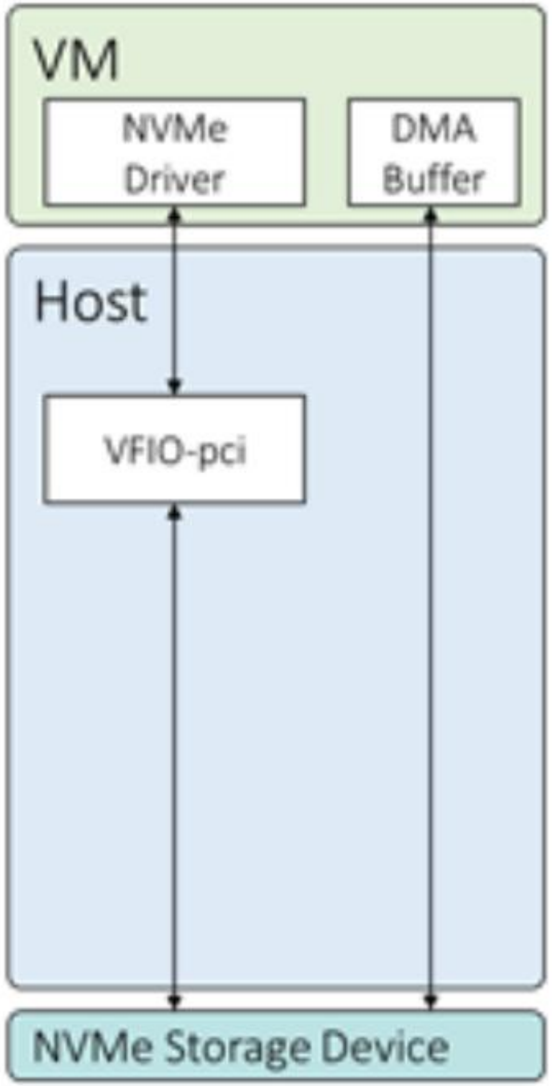


SR-IOV

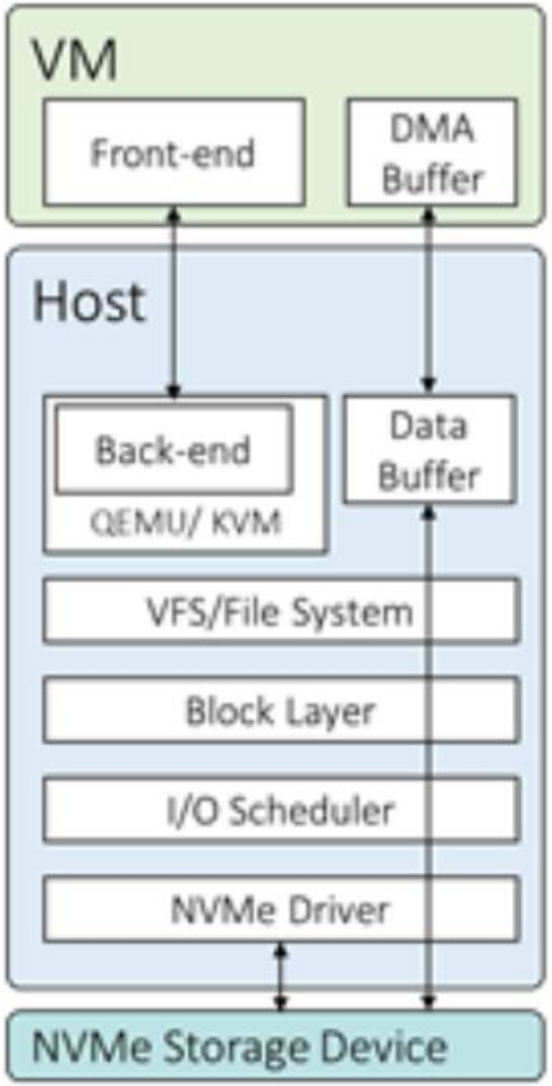
- Illusion of multiple virtual devices supported in HW
- Simplifies sharing for VMM
- Enables direct VM → device communication
- Drawbacks?
- MR-IOV?



Storage vritualization



SR-IOV



Common software approach

Modern HW is fast

Typical commodity desktop (Dell PowerEdge R520 ~$1000):

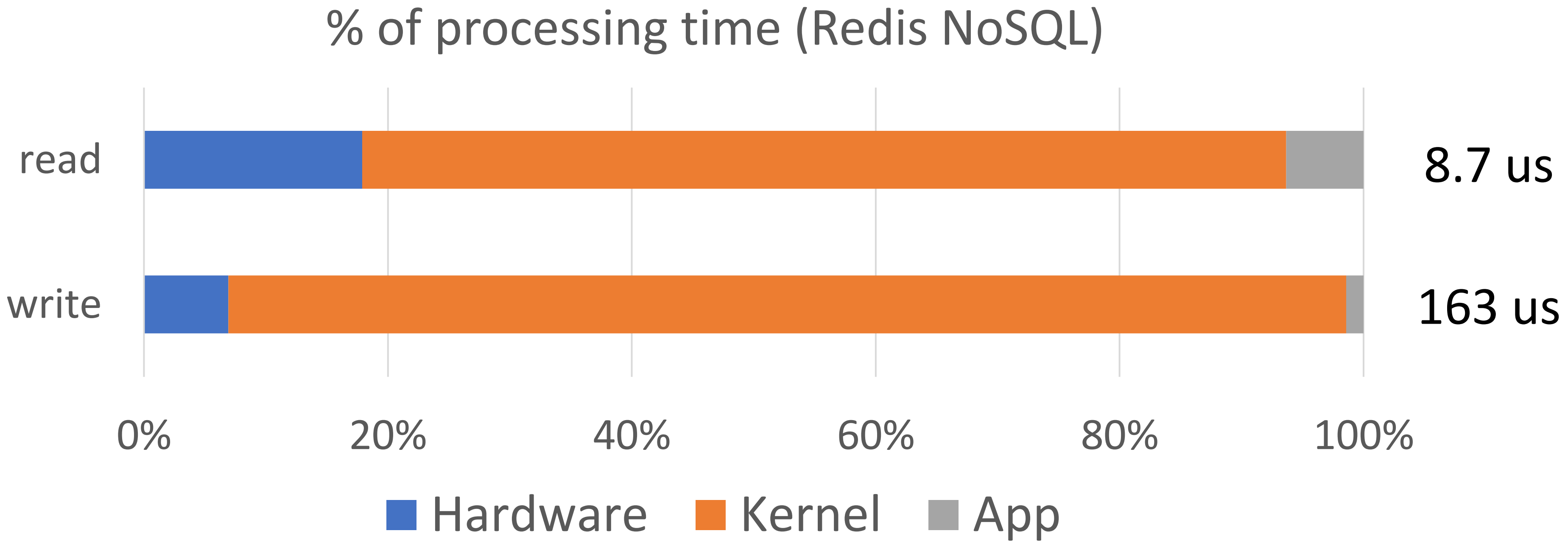
10G NIC
~2us / 1KB pkt

6-core CPU

RAID w/ 1G cache
~25 us / 1KB write

Background

- Balance between I/O and CPU speeds has shifted
- CPUs can't keep up!



Where are all the in-kernel cycles going?

	Read hit				Durable write			
	Linux		Arrakis/P		Linux		Arrakis/P	
`epoll`	2.42	(27.91%)	1.12	(27.52%)	2.64	(1.62%)	1.49	(4.73%)
`recv`	0.98	(11.30%)	0.29	(7.13%)	1.55	(0.95%)	0.66	(2.09%)
Parse input	0.85	(9.80%)	0.66	(16.22%)	2.34	(1.43%)	1.19	(3.78%)
Lookup/set key	0.10	(1.15%)	0.10	(2.46%)	1.03	(0.63%)	0.43	(1.36%)
Log marshaling	-		-		3.64	(2.23%)	2.43	(7.71%)
`write`	-		-		6.33	(3.88%)	0.10	(0.32%)
`fsync`	-		-		137.84	(84.49%)	24.26	(76.99%)
Prepare response	0.60	(6.92%)	0.64	(15.72%)	0.59	(0.36%)	0.10	(0.32%)
`send`	3.17	(36.56%)	0.71	(17.44%)	5.06	(3.10%)	0.33	(1.05%)
Other	0.55	(6.34%)	0.46	(11.30%)	2.12	(1.30%)	0.52	(1.65%)
Total	8.67	($\sigma = 2.55$)	4.07	($\sigma = 0.44$)	163.14	($\sigma = 13.68$)	31.51	($\sigma = 1.91$)
99th percentile	15.21		4.25		188.67		35.76	

Where are all the in-kernel cycles going?

	Read hit				Durable write			
	Linux		Arrakis/P		Linux		Arrakis/P	
epoll	2.42	(27.91%)	1.12	(27.52%)	2.64	(1.62%)	1.49	(4.73%)
recv	0.98	(11.30%)	0.29	(7.13%)	1.55	(0.95%)	0.66	(2.09%)
Parse input	0.85	(9.80%)	0.66	(16.22%)	2.34	(1.43%)	1.19	(3.78%)
Lookup/set key	0.10	(1.15%)	0.10	(2.46%)	1.03	(0.63%)	0.43	(1.36%)
Log marshaling	-		-		3.64	(2.23%)	2.43	(7.71%)
write	-		-		6.33	(3.88%)	0.10	(0.32%)
fsync	-		-		137.84	(84.49%)	24.26	(76.99%)
Prepare response	0.60	(6.92%)	0.64	(15.72%)	0.59	(0.36%)	0.10	(0.32%)
send	3.17	(36.56%)	0.71	(17.44%)	5.06	(3.10%)	0.33	(1.05%)
			0.46	(11.30%)	2.12	(1.30%)	0.52	(1.65%)
			4.07	(σ=0.44)	163.14	(σ=13.68)	31.51	(σ=1.91)
			4.25		188.67		35.76	

System Calls are slow:

- epoll : 27% time of read
- recv : 11% time of read
- send : 37% time of read
- fsync : 84% time of write

Where are all the in-kernel cycles going?

	Read hit				Durable write			
	Linux		Arrakis/P		Linux		Arrakis/P	
`epoll`	2.42	(27.91%)	1.12	(27.52%)	2.64	(1.62%)	1.49	(4.73%)
`recv`	0.98	(11.30%)	0.29	(7.13%)	1.55	(0.95%)	0.66	(2.09%)
Parse input	0.85	(9.80%)	0.66	(16.22%)	2.34	(1.43%)	1.19	(3.78%)
Lookup/set key	0.10	(1.15%)	0.10	(2.46%)	1.03	(0.63%)	0.43	(1.36%)
Log marshaling	-		-		3.64	(2.23%)	2.43	(7.71%)
`write`	-		-		6.33	(3.88%)	0.10	(0.32%)
`fsync`	-		-		137.84	(84.49%)	24.26	(76.99%)
Prepare response	0.60	(6.92%)	0.64	(15.7				
`send`	3.17	(36.56%)	0.71	(1				
			0.46	(1				
			4.07	(σ				
			4.25					

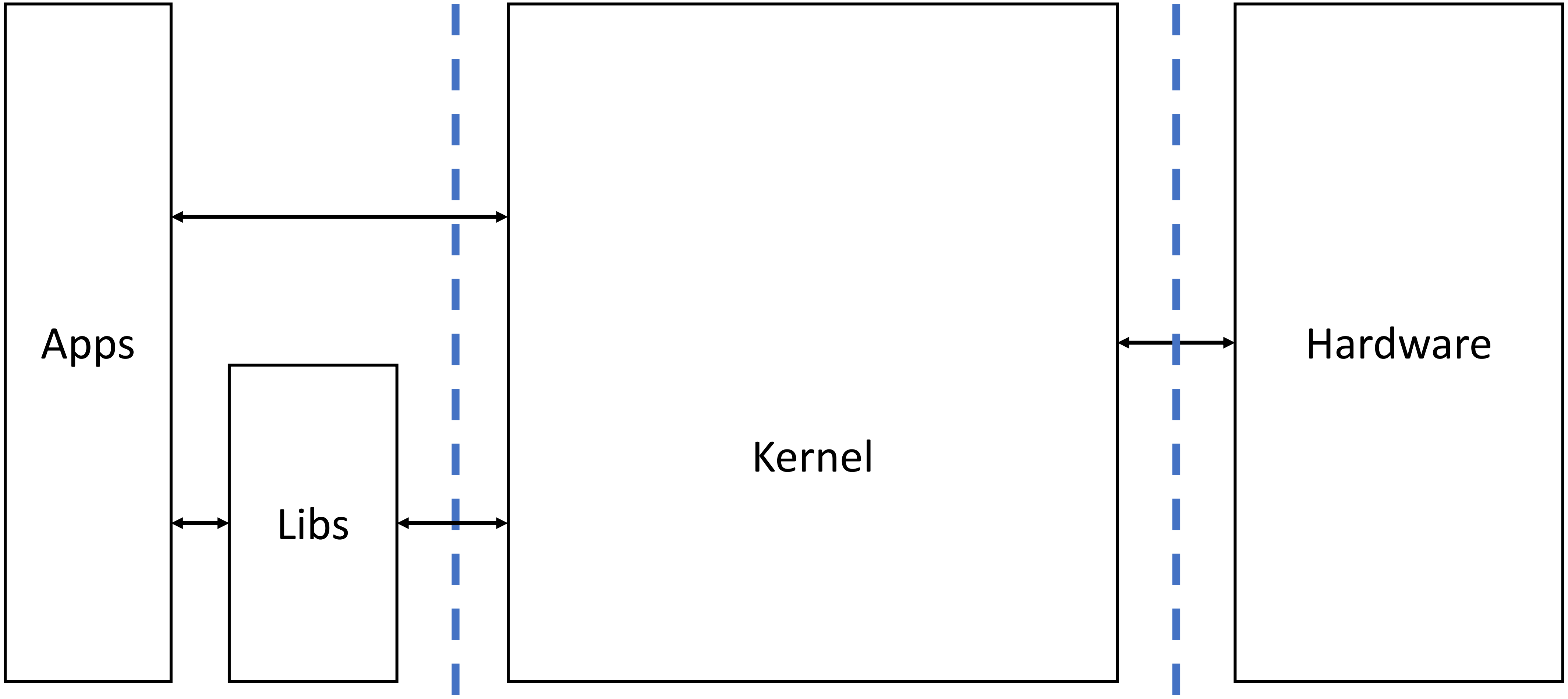
System Calls are slow:

- `epoll` : 27% time of read
- `recv` : 11% time of read
- `send` : 37% time of read
- `fsync` : 84% time of write

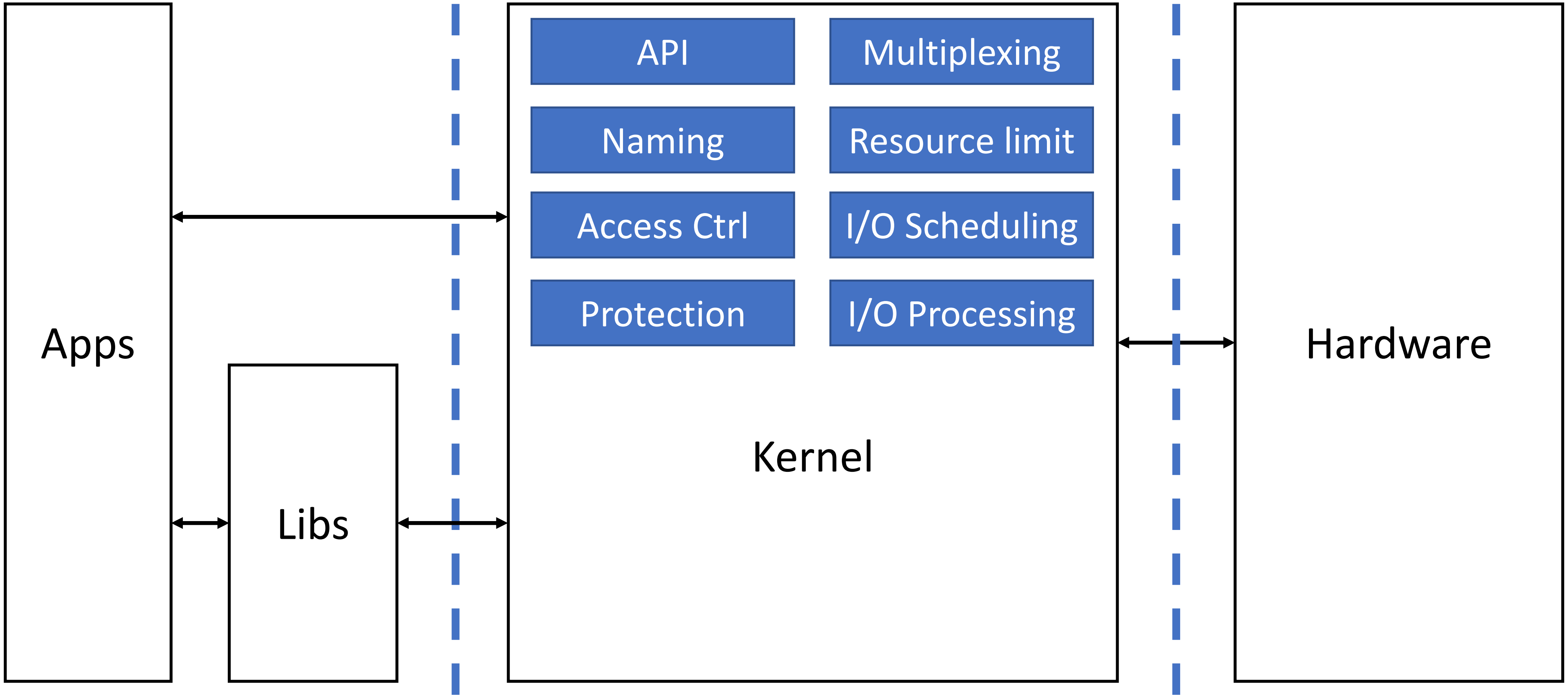
Arrakis→I/O centric design

- *Bypass* kernel
- Abstractions: user-space device access
- SR-IOV higher in stack
  - Leverage packet filter/load-balance/scheduling support

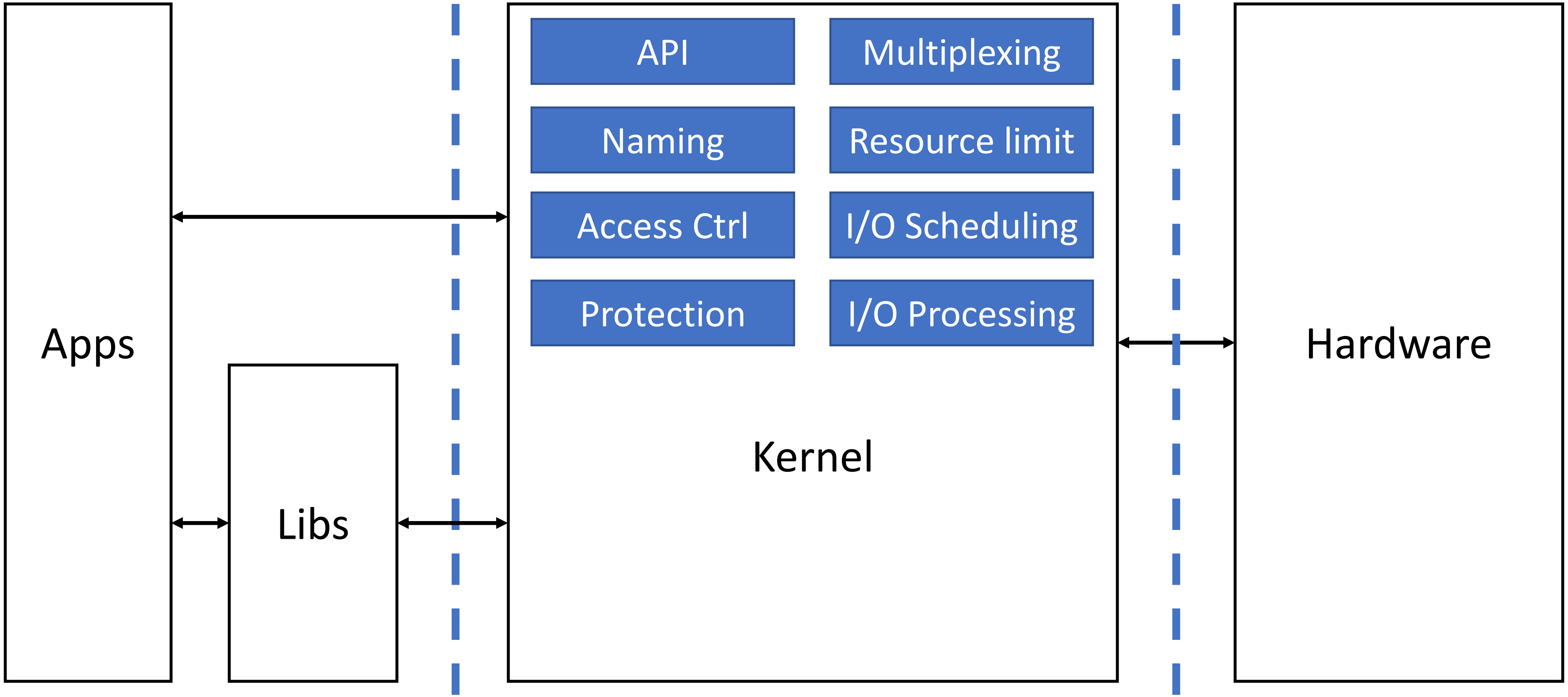
Traditional OS



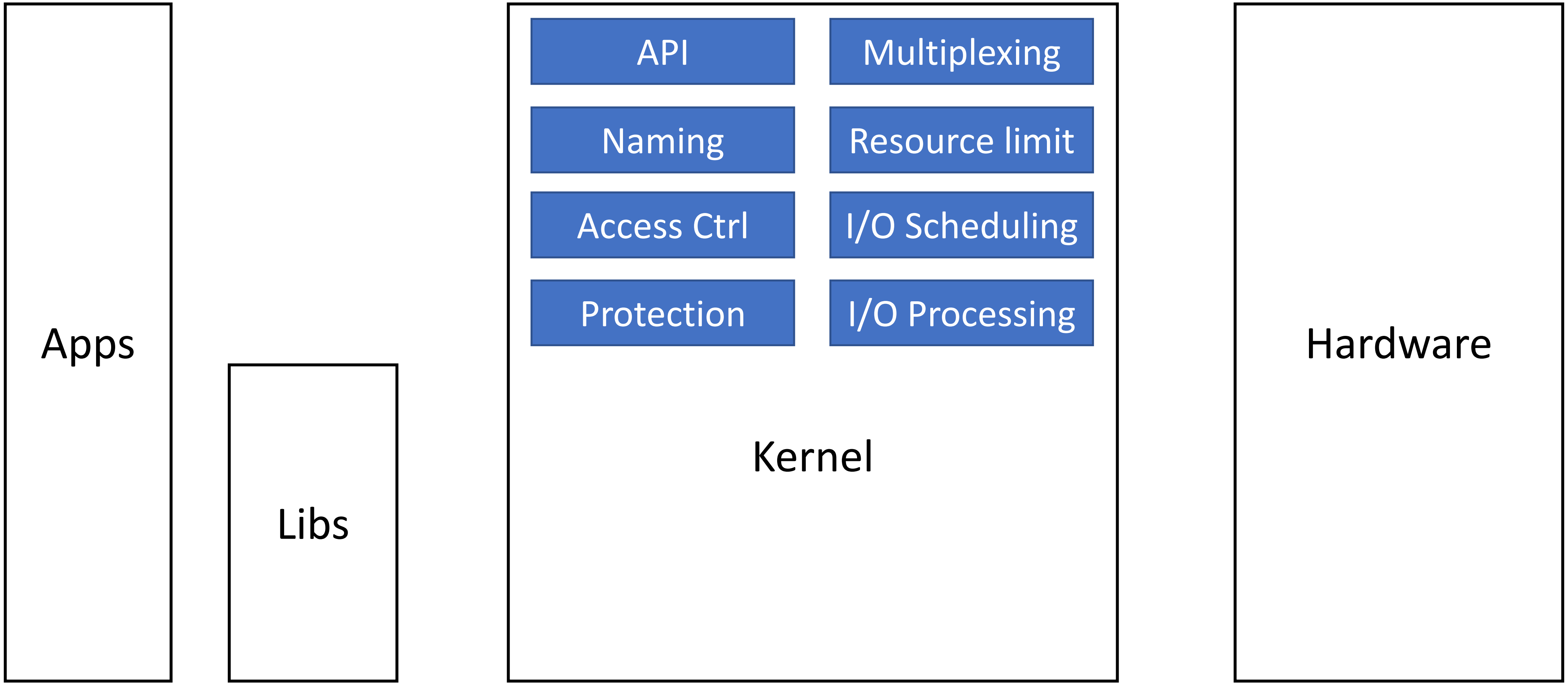
Traditional OS



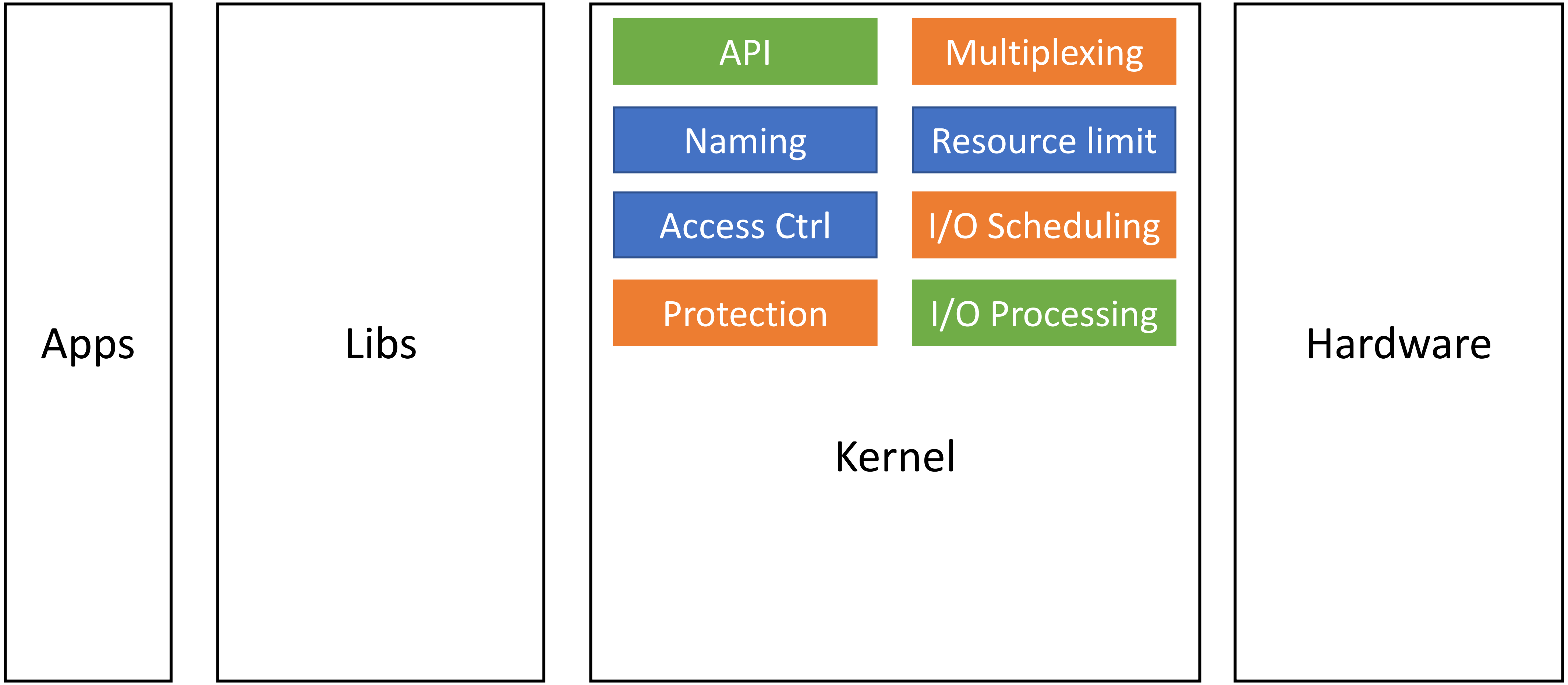
Kernel bypass



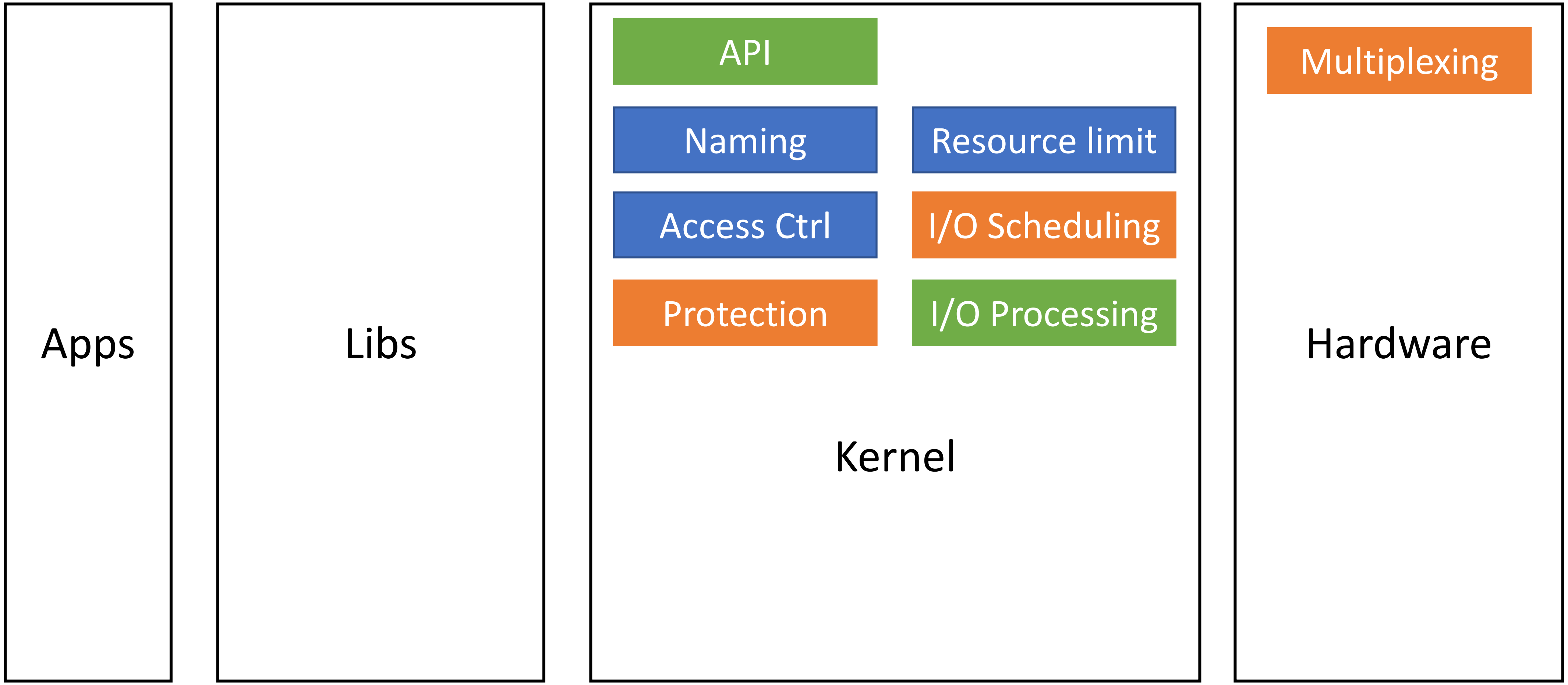
Kernel bypass



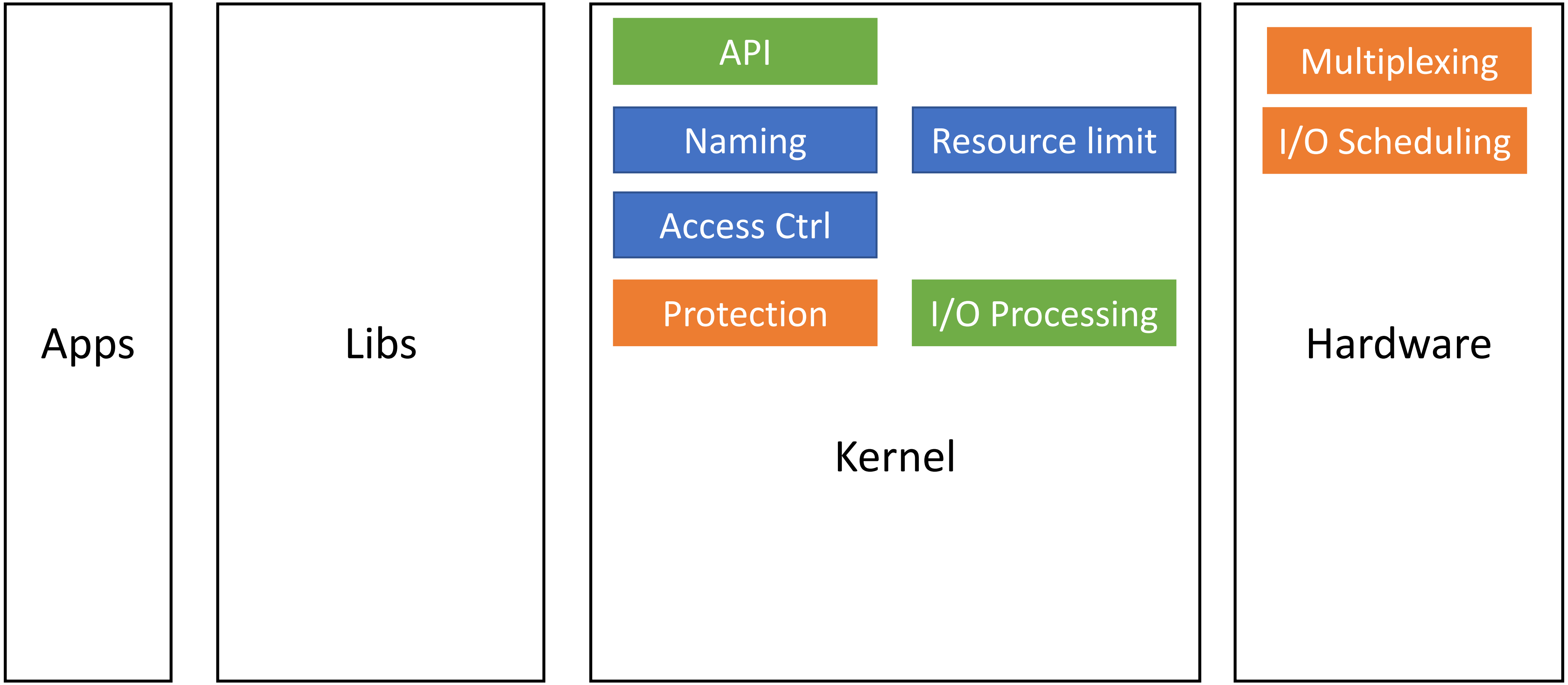
Kernel bypass



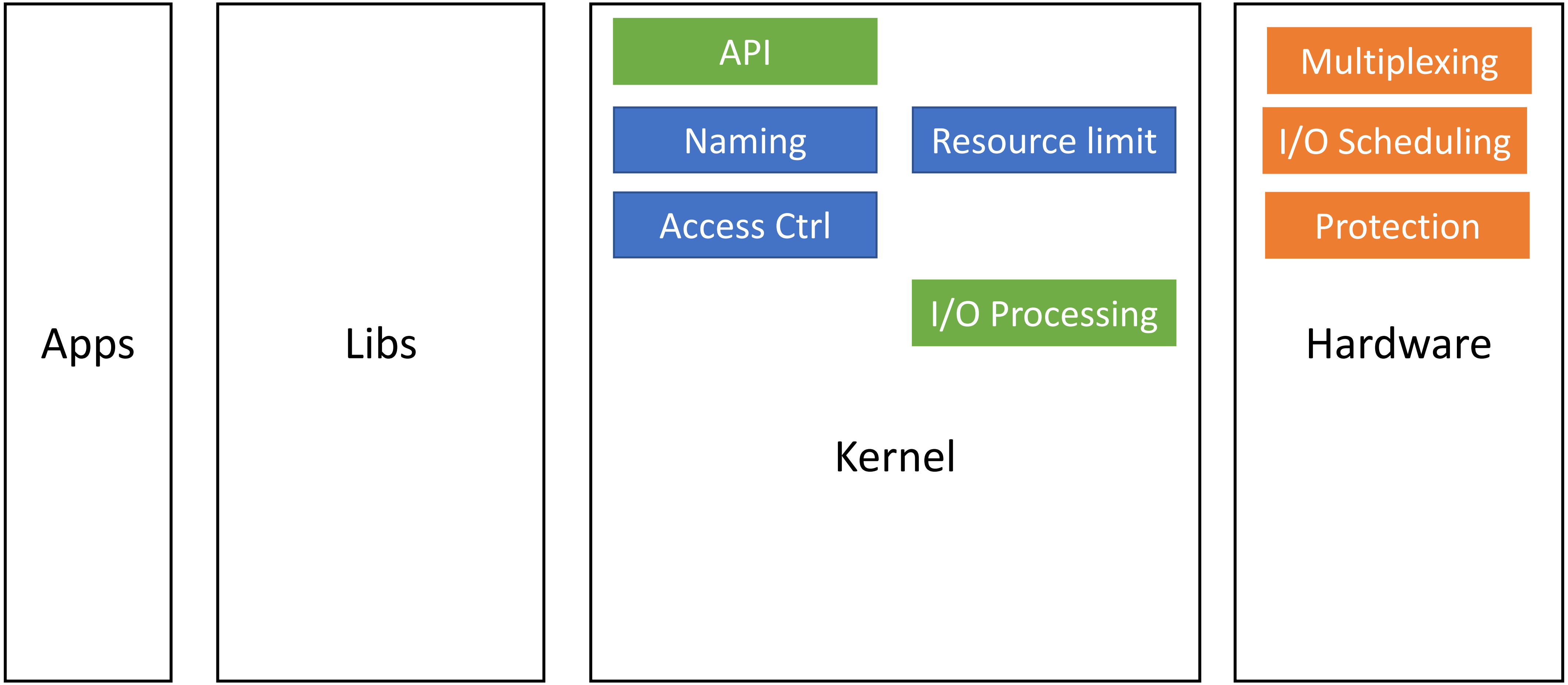
Kernel bypass



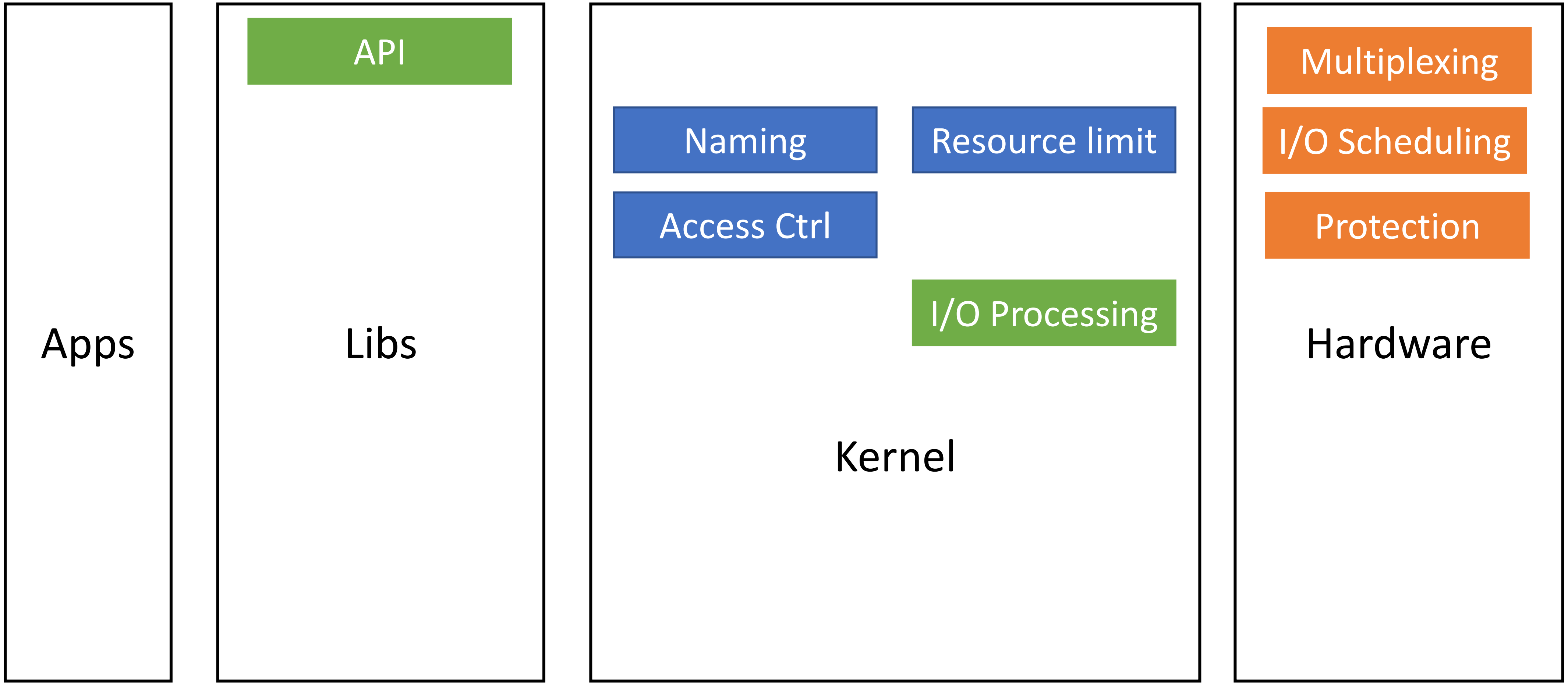
Kernel bypass



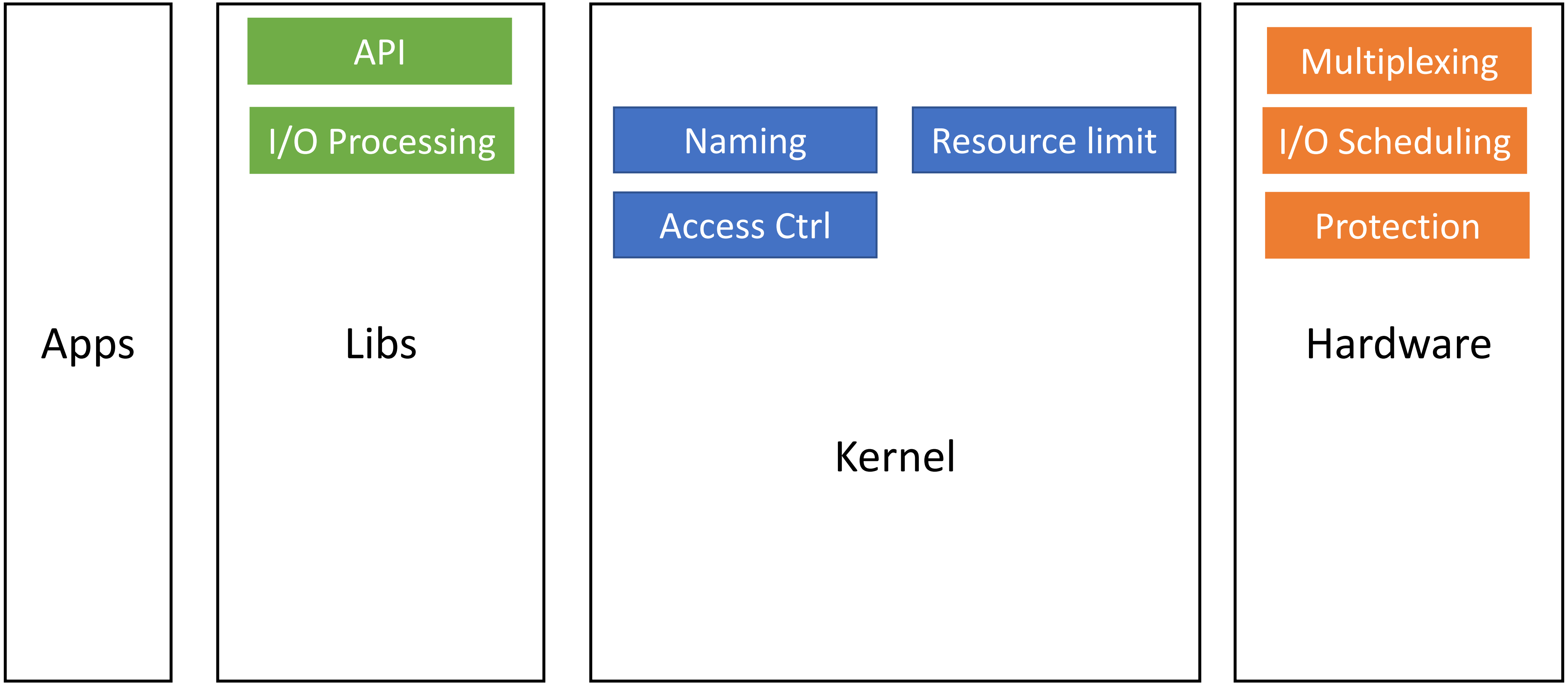
Kernel bypass



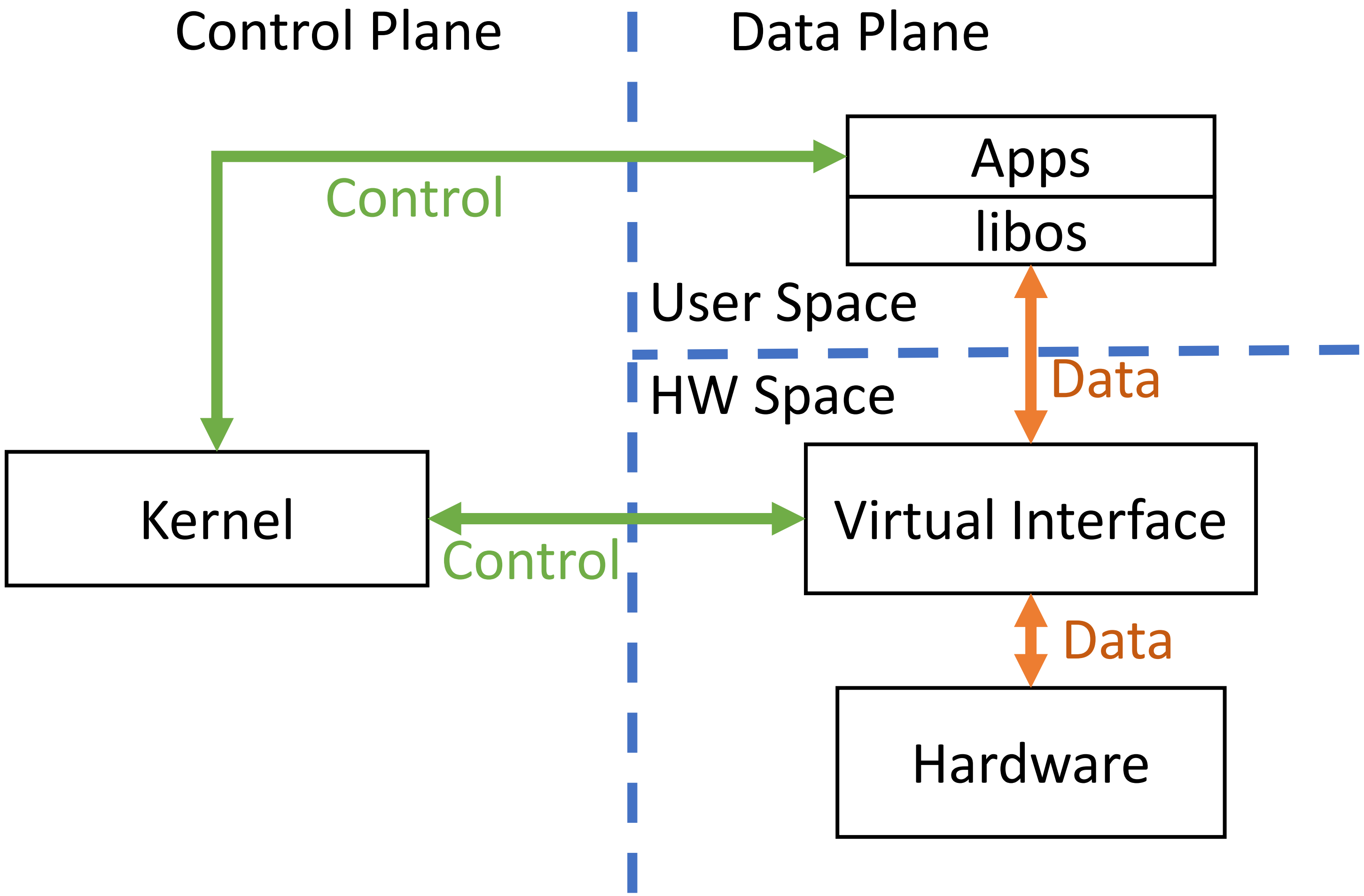
Kernel bypass



Kernel bypass



Kernel bypass. The OS is the control plane.



Hardware Model

- NICs (Multiplexing, Protection, Scheduling)

- Storage
  - VSIC (Virtual Storage Interface Controller)
    - each w/ queues etc.
  - VSA (Virtual Storage Areas)
    - mapped to physical devices
    - associated with VSICs
    - VSA & VSIC : many-to-many mapping

Control Plane Interface

- VIC (Virtual Interface Card)
  - Apps can create/delete VICs, associate them to doorbells
- doorbells (like interrupt?)
  - associated with events on VICs
- filter creation
  - e.g. create_filter(rx,*,tcp.port == 80)

Control Plane Features

- Access control
  - enforced by filters
  - infrequently invoked (during set-up etc.)
  - Can export an entire VSA
- Resource limiting
  - send commands to hardware I/O schedulers
- Naming
  - VFS in kernel
  - actual storage implemented in apps
  - "By default, the Arrakis application library managing the VSA exports a file server interface; other applications can use normal POSIX API calls via user-level RPC to the embedded library file server. This library can also run as a standalone process to provide access when the original application is not active" What does this sound like?

Network Data Interface

- Apps send/receive directly through sets of queues
- filters applied for multiplexing
- doorbell used for asynchronous notification (e.g. packet arrival)
- both native (w/ zero-copy) and POSIX are implemented

Storage Data Interface

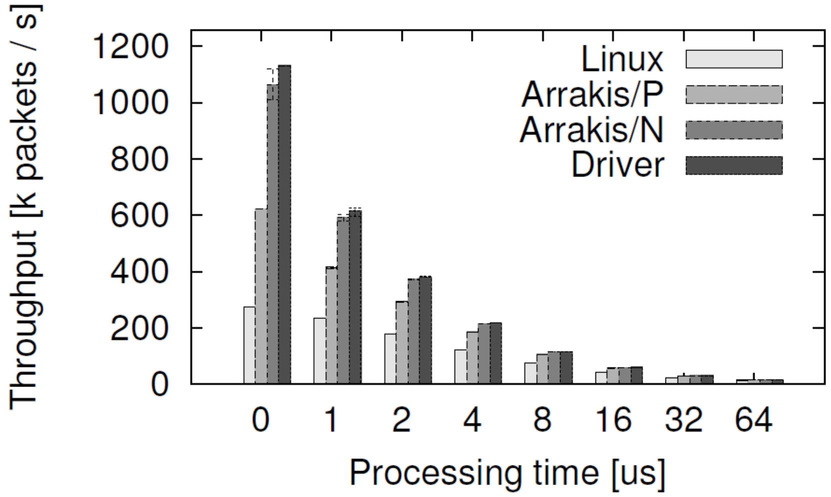
- VSA supports `read`, `write`, `flush`

- persistent data structure (log, queue)
  - modified Redis by 109 LOC
  - operations immediately persistent on disk
  - eliminate marshaling (layout in memory = in disk)
  - data structure specific caching & early allocation

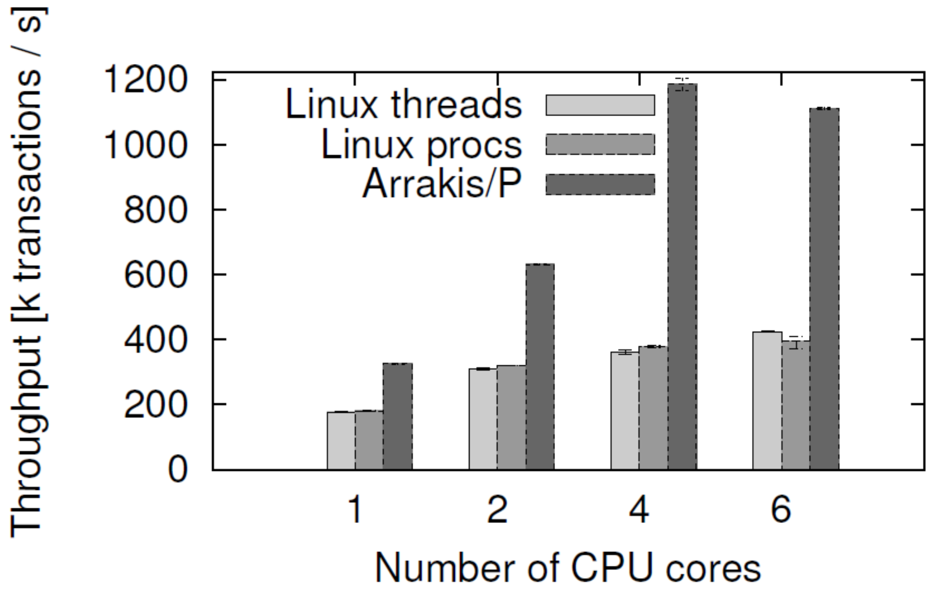
Evaluation

1. UDP echo server
2. Memcached key-value store
3. Redis NoSQL store
4. HTTP load balancer (haproxy)
5. IP-layer middle box
6. Performance isolation (rate limiting)

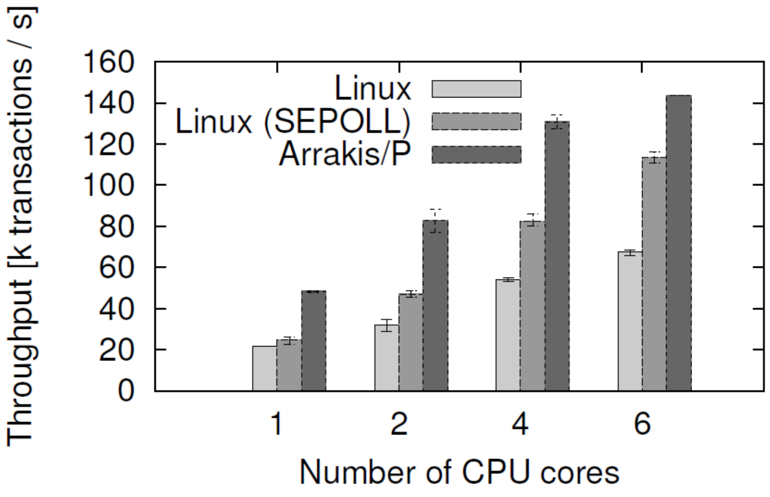
Performance



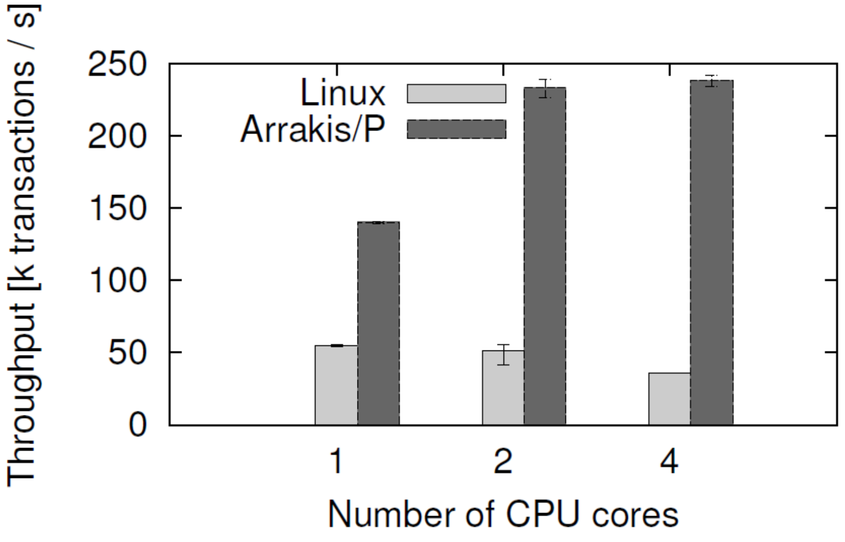
UDP Echo



memcached

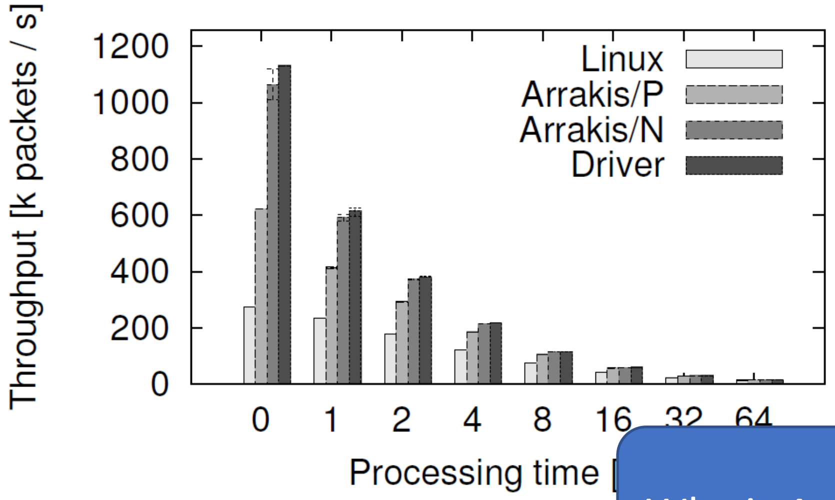


Load balancer

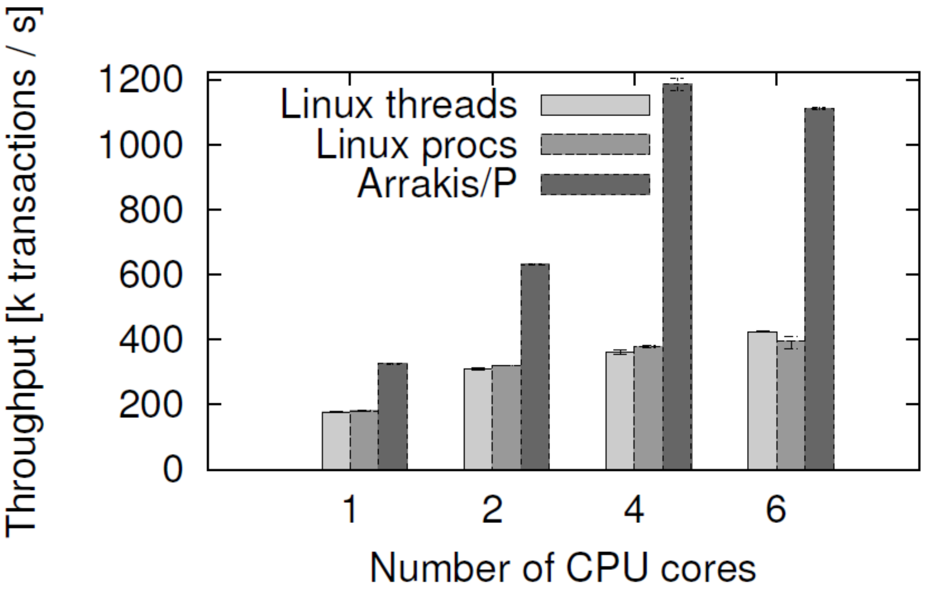


IP middlebox

Performance

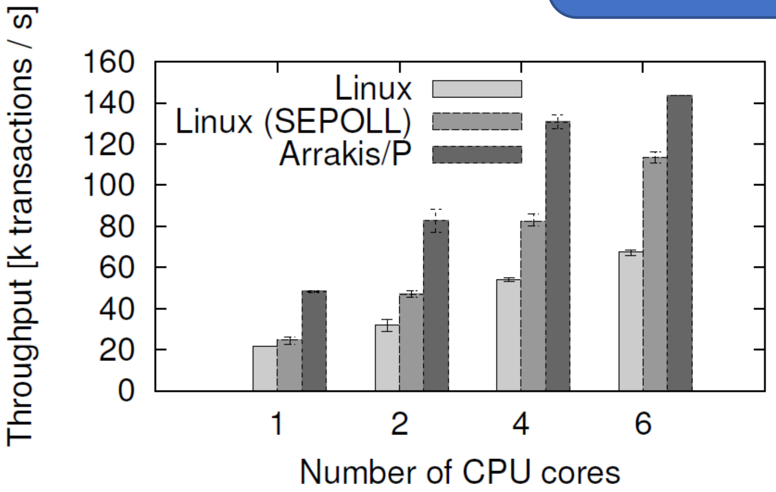


UDP Echo

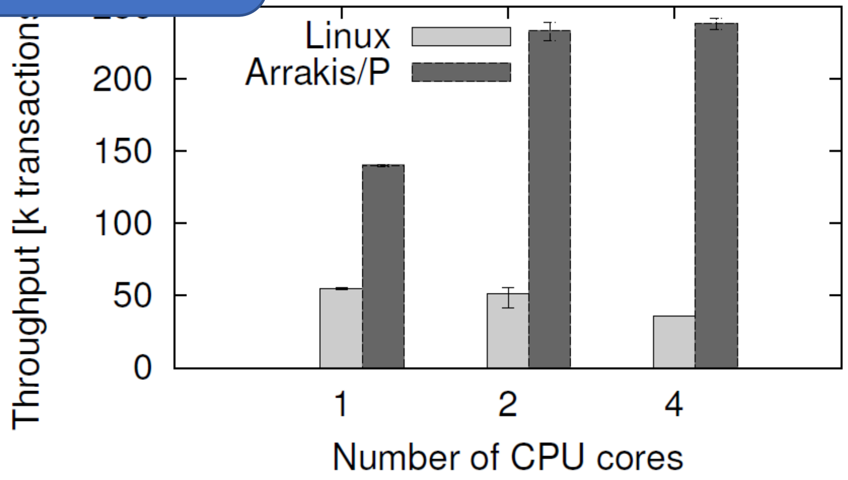


memcached

Why is Arrakis/N faster than Arrakis/P?

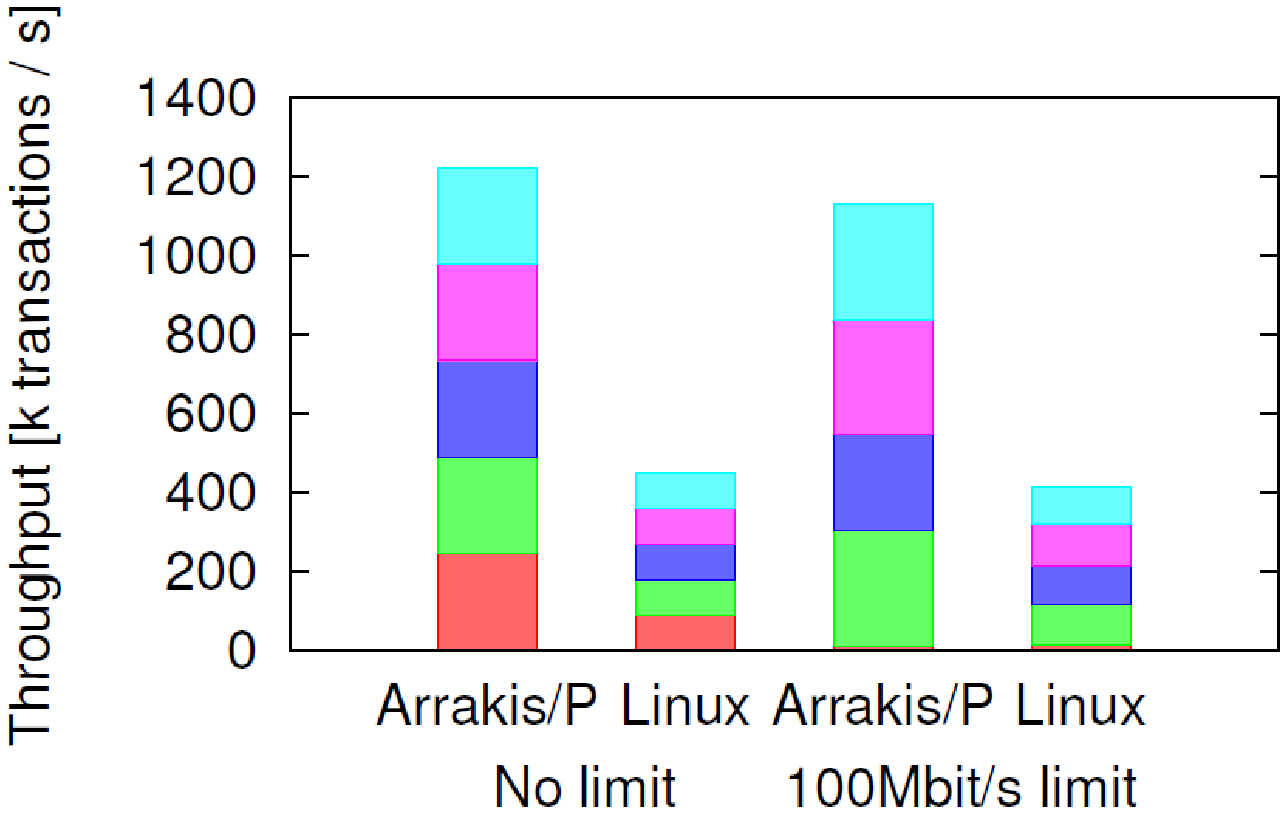


Load balancer

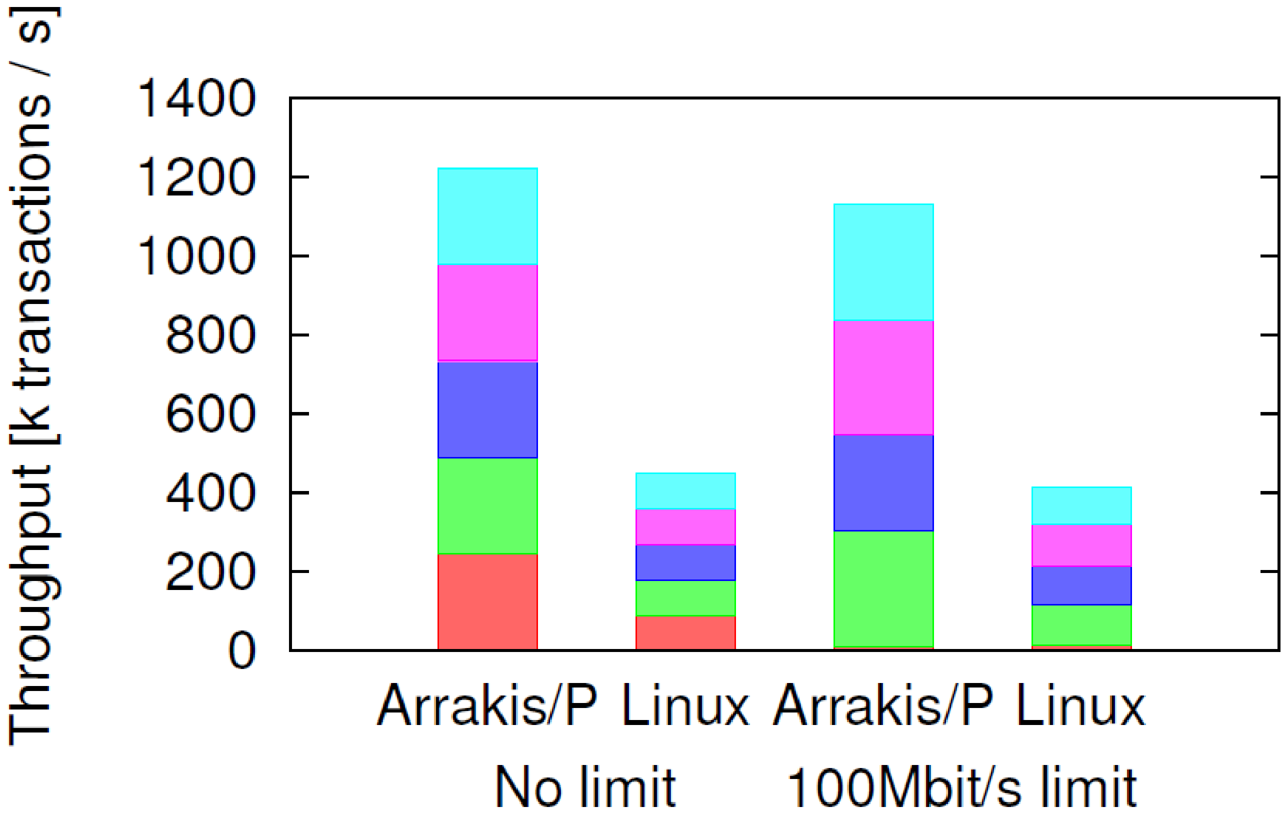


IP middlebox

Case 6: Performance Isolation



Case 6: Performance Isolation



What mechanism(s) enable(s) Arrakis to achieve proportionality?

Discussion

- Pros:
  - much better **raw** performance (for I/O intensive Data Center apps)
    - Redis: up to 9x throughput and 81% speedup
    - Memcached: scales to 3x throughput
- Cons:
  - some features require hardware functionality that is not yet available
    - will other device classes follow suit?
  - requires modification of applications
  - not clear about storage abstractions
  - not easy to track behaviors inside the hardware
- Is Arrakis trading “OS features” for raw performance?

IX, Arrakis, Exokernel, Multikernel

- Arrakis is like Exokernel built on Barrelfish (multikernel)

	IX	Arrakis
Reduce SysCall overhead	Adaptive batching Run to completion	No SysCall in data-plane
Hardware virtualization	No IOMMU No SR-IOV	Expect more than what we have
Enforcement of network I/O policy	Under software control	Rely on hardware