Arrakis: The Operation System is the control plane Reading Report

Simon Peter, University of Washionton .osdi
邓志会 2015210926
元东 2015210938

Building an os for data center

Server I/O performance matters

key-value stores, web & file servers, lock management

- Can we deliver performance close to hardware?
- Example system: Dell PowerEdge R520

Intel X520
10G NIC
2 us / 1KB packet

+

Intel RS3 RAID
1GB flash-backed cache
25 us / 1KB write

+

Sandy Bridge CPU
6 cores, 2.2 GHz

= **$1,200**

Packet processing overhead

Table I. 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.

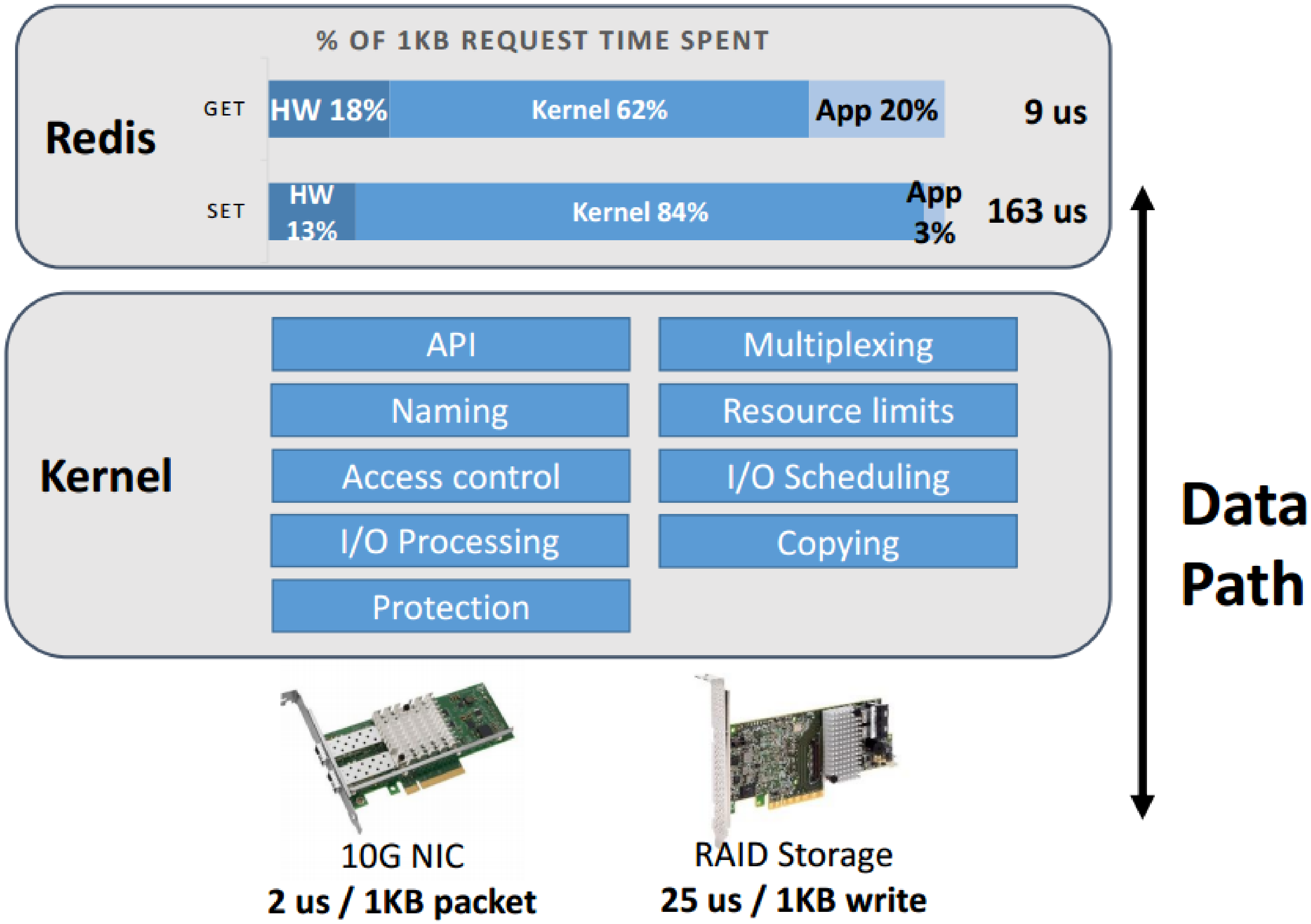
		Linux				Arrakis			
		Receiver running		CPU idle		Arrakis/P		Arrakis/N	
Network stack	in	1.26	(37.6%)	1.24	(20.0%)	0.32	(22.3%)	0.21	(55.3%)
	out	1.05	(31.3%)	1.42	(22.9%)	0.27	(18.7%)	0.17	(44.7%)
Scheduler		0.17	(5.0%)	2.40	(38.8%)	-		-	
Copy	in	0.24	(7.1%)	0.25	(4.0%)	0.27	(18.7%)	-	
	out	0.44	(13.2%)	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		6.19		1.44		0.38	
Std. dev.		0.66		0.82		<0.01		<0.01	

Redis NoSqlstore overheads

	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	

Table 2: Overheads in the Redis NoSQL store for memory reads (hits) and durable writes (legend in Table 1).

Linux I/O Performance



Kernel mediation is too heavyweight

Arrakis Goals

Skip kernel & deliver I/O directly to applications

- Reduce OS overhead

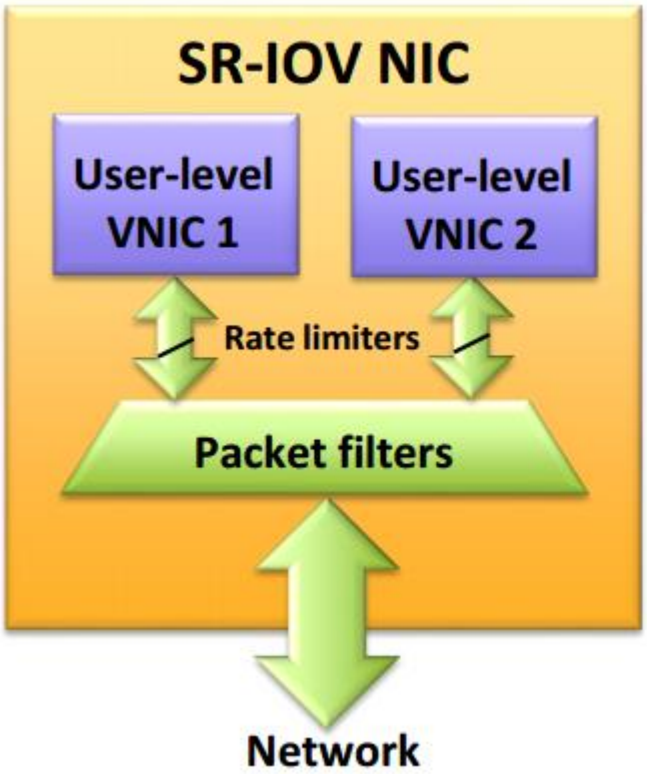
Keep classical server OS features

- Process protection
- Resource limits
- I/O protocol flexibility
- Global naming

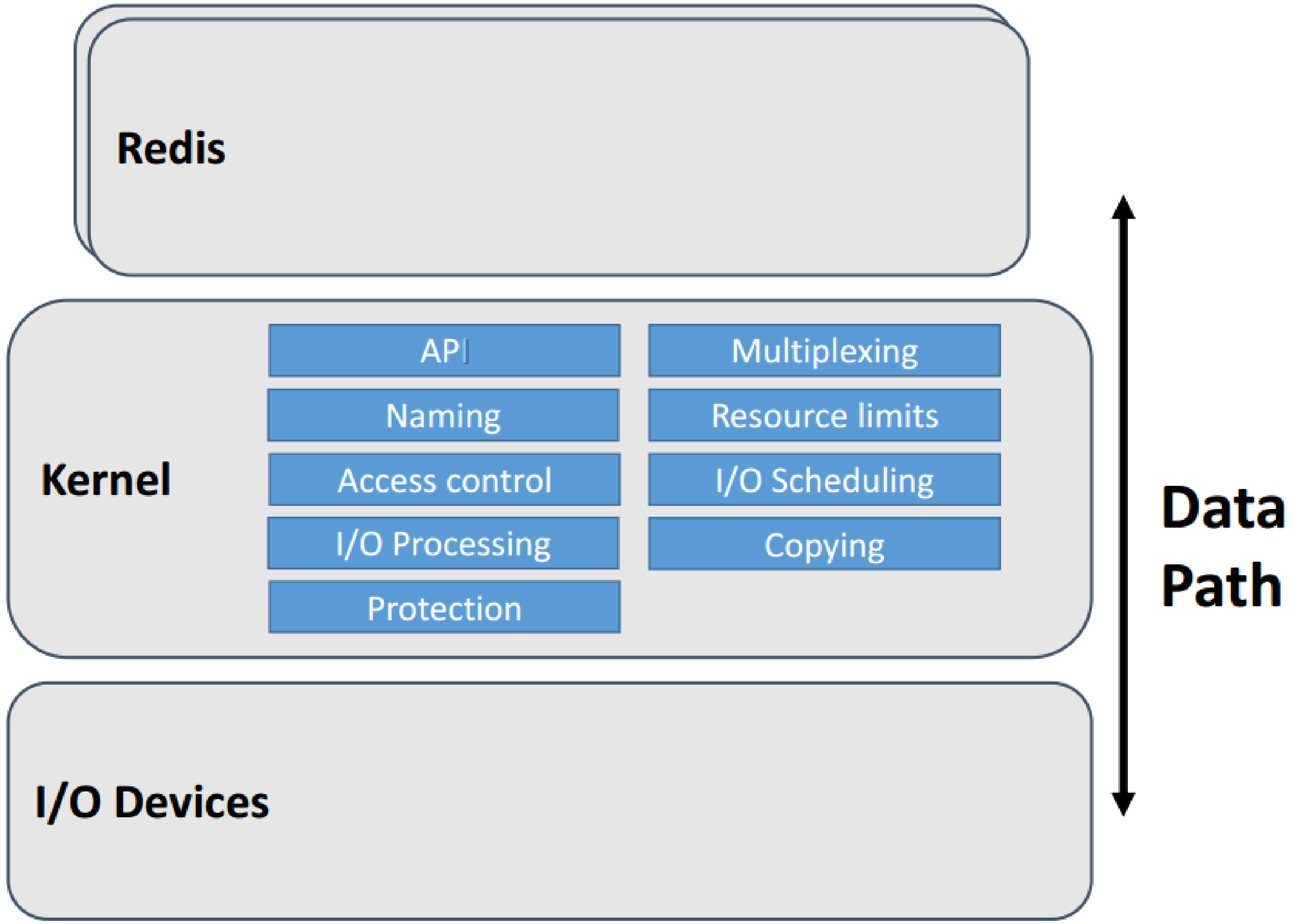
The hardware can help us…

Hardware I/O Virtualization

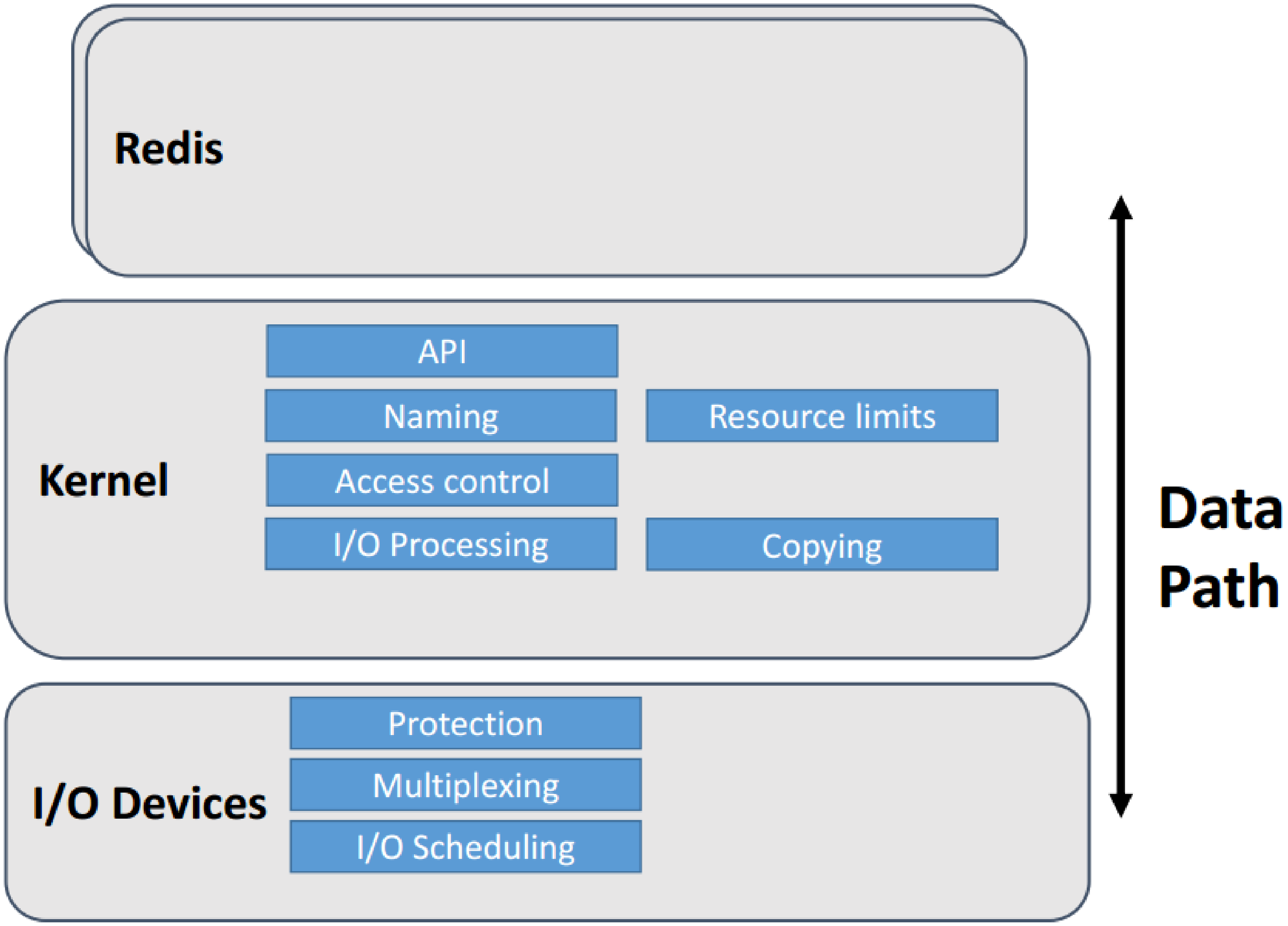
- Standard on NIC, emerging on RAID
- Multiplexing
  - SR-IOV: Virtual PCI devices
    w/ own registers, queues, INTs
- Protection
  - IOMMU:
    Devices use app virtual memory
  - Packet filters, logical disks:
    Only allow eligible I/O
- I/O Scheduling
  - NIC rate limiter, packet schedulers



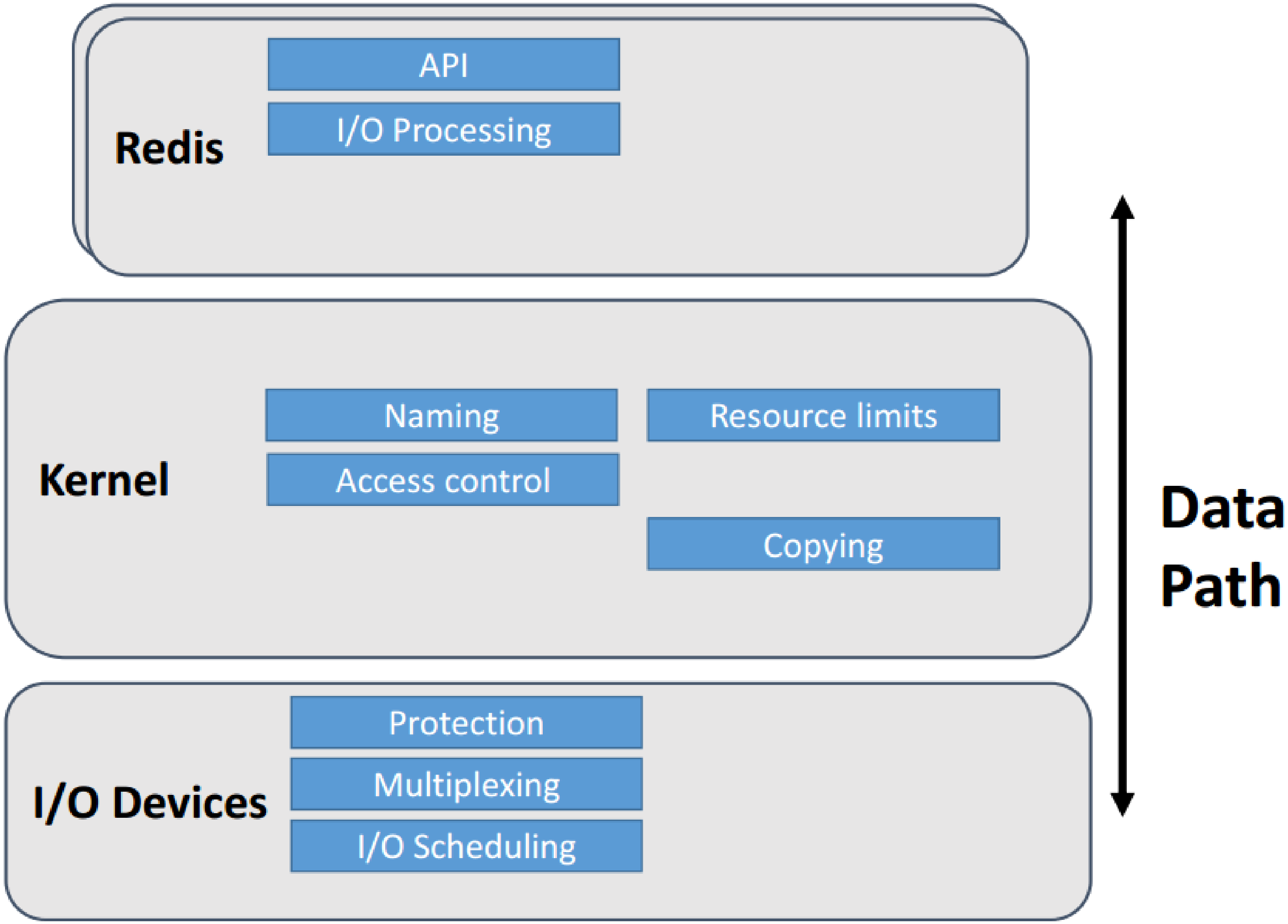
How to skip the kernel?



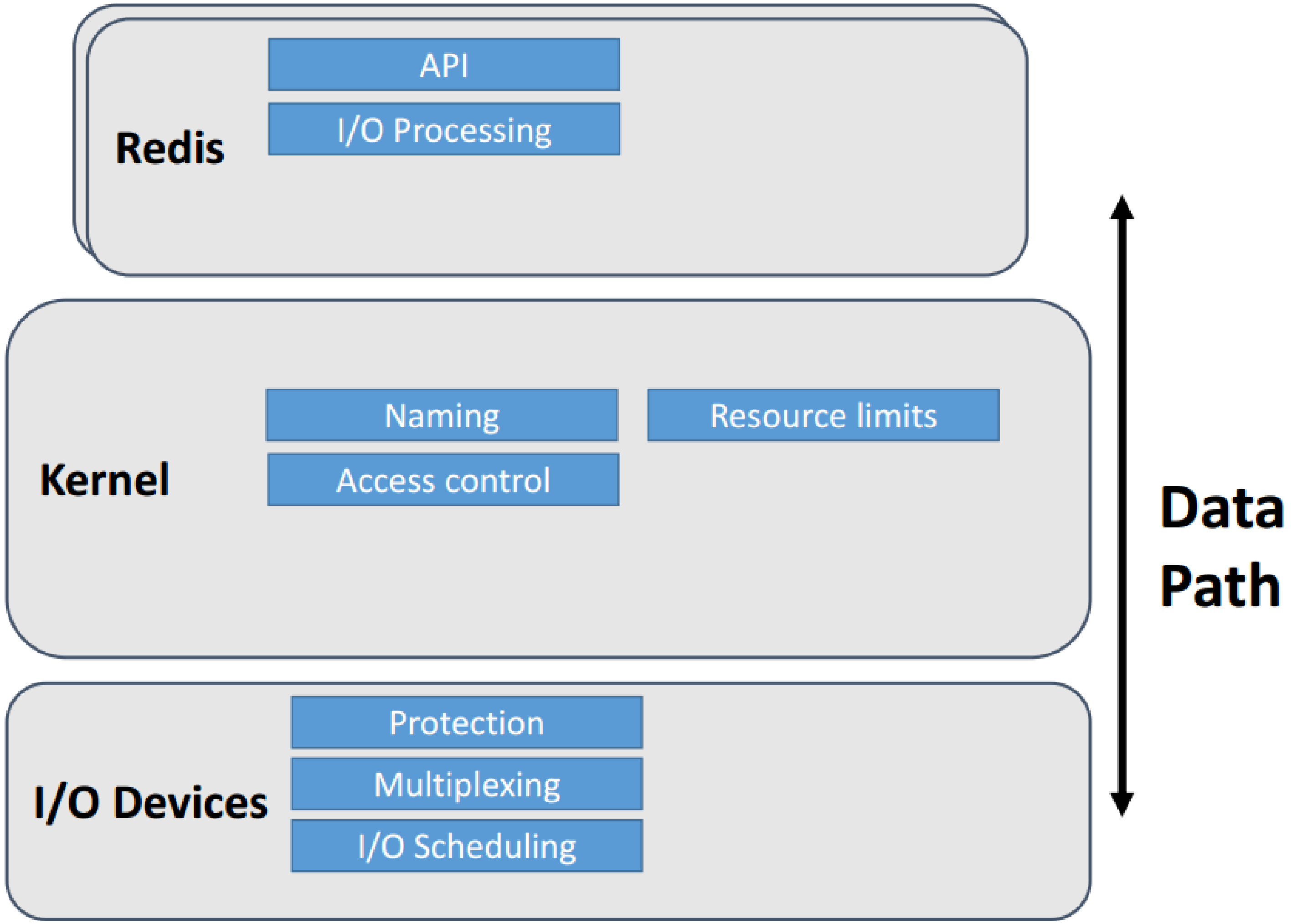
How to skip the kernel?



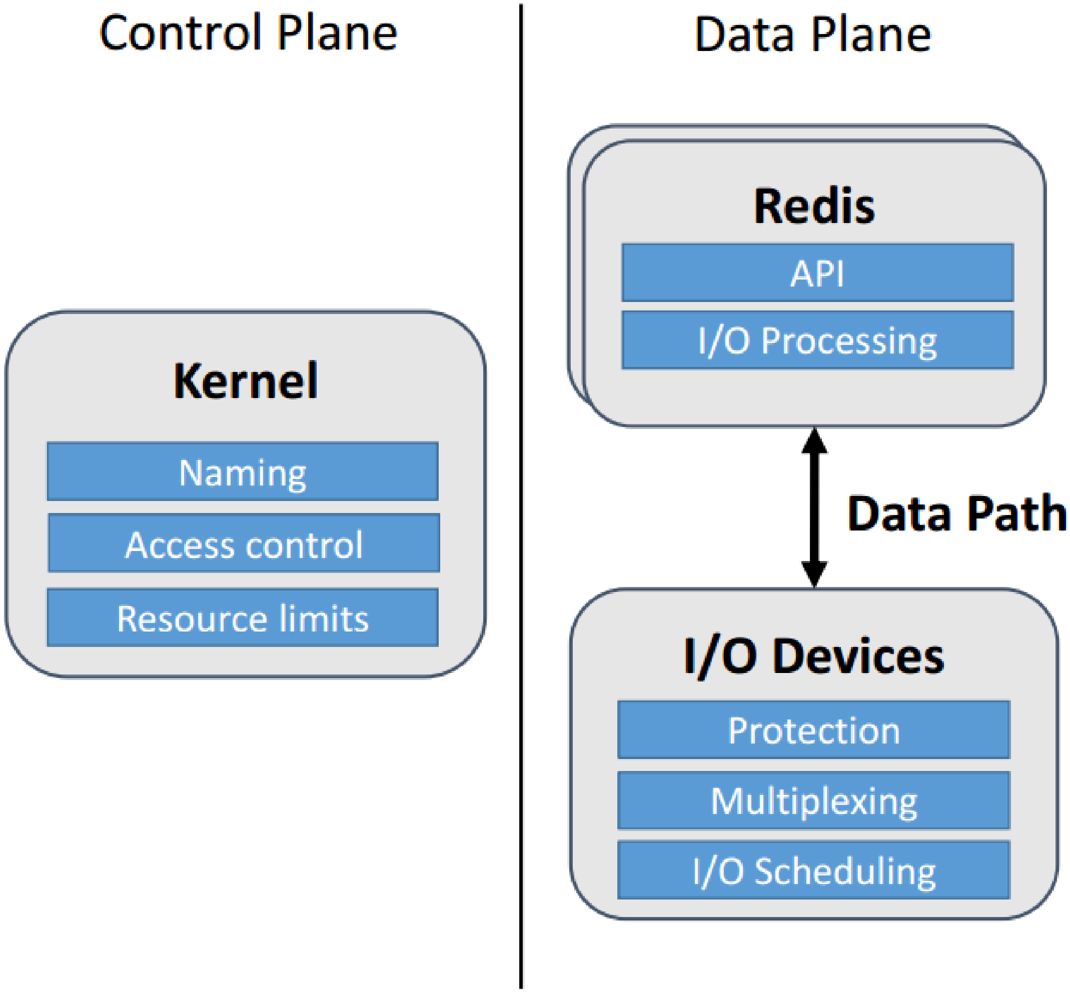
How to skip the kernel?



How to skip the kernel?



Arrakis I/O Architecture



Arrakis I/O Architecture

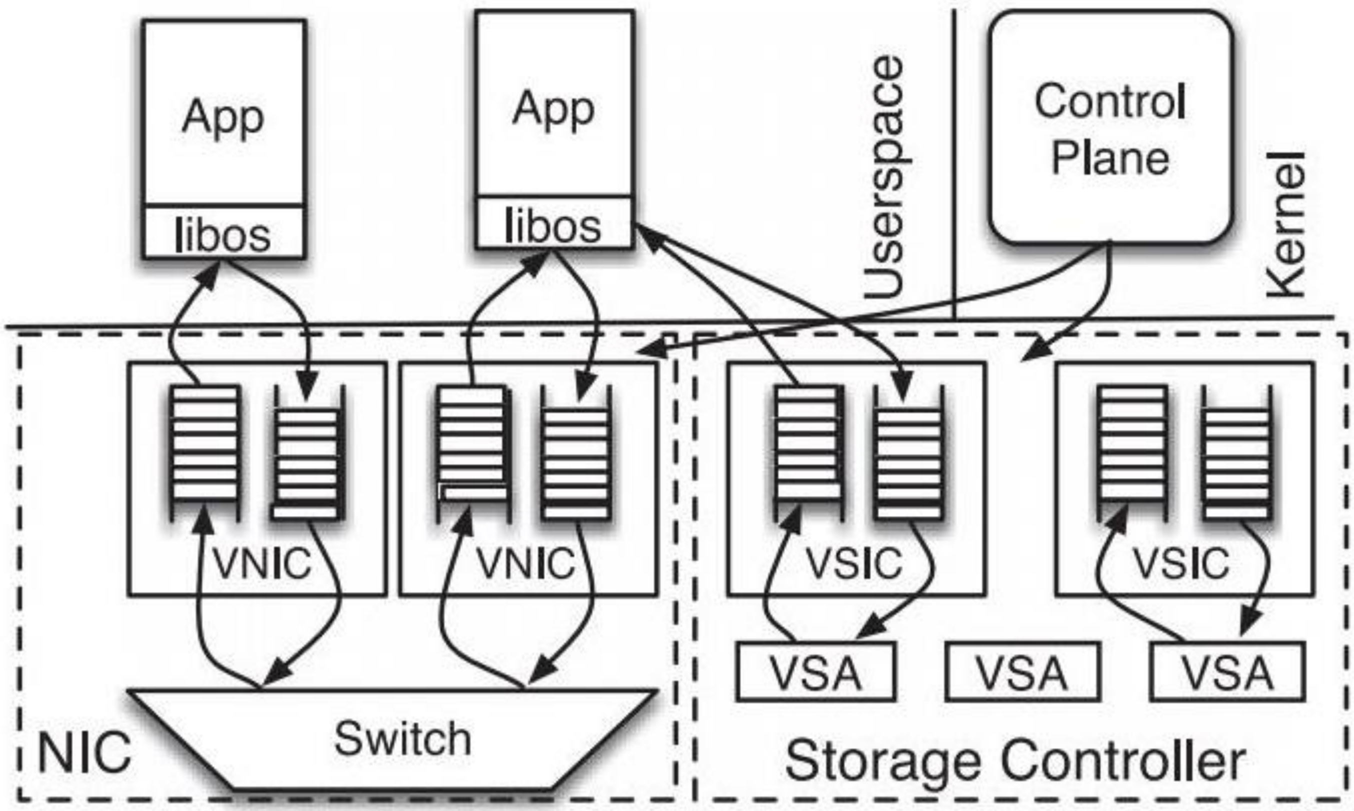
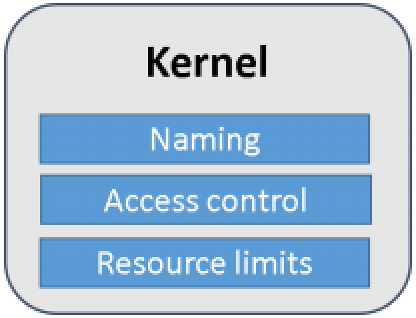


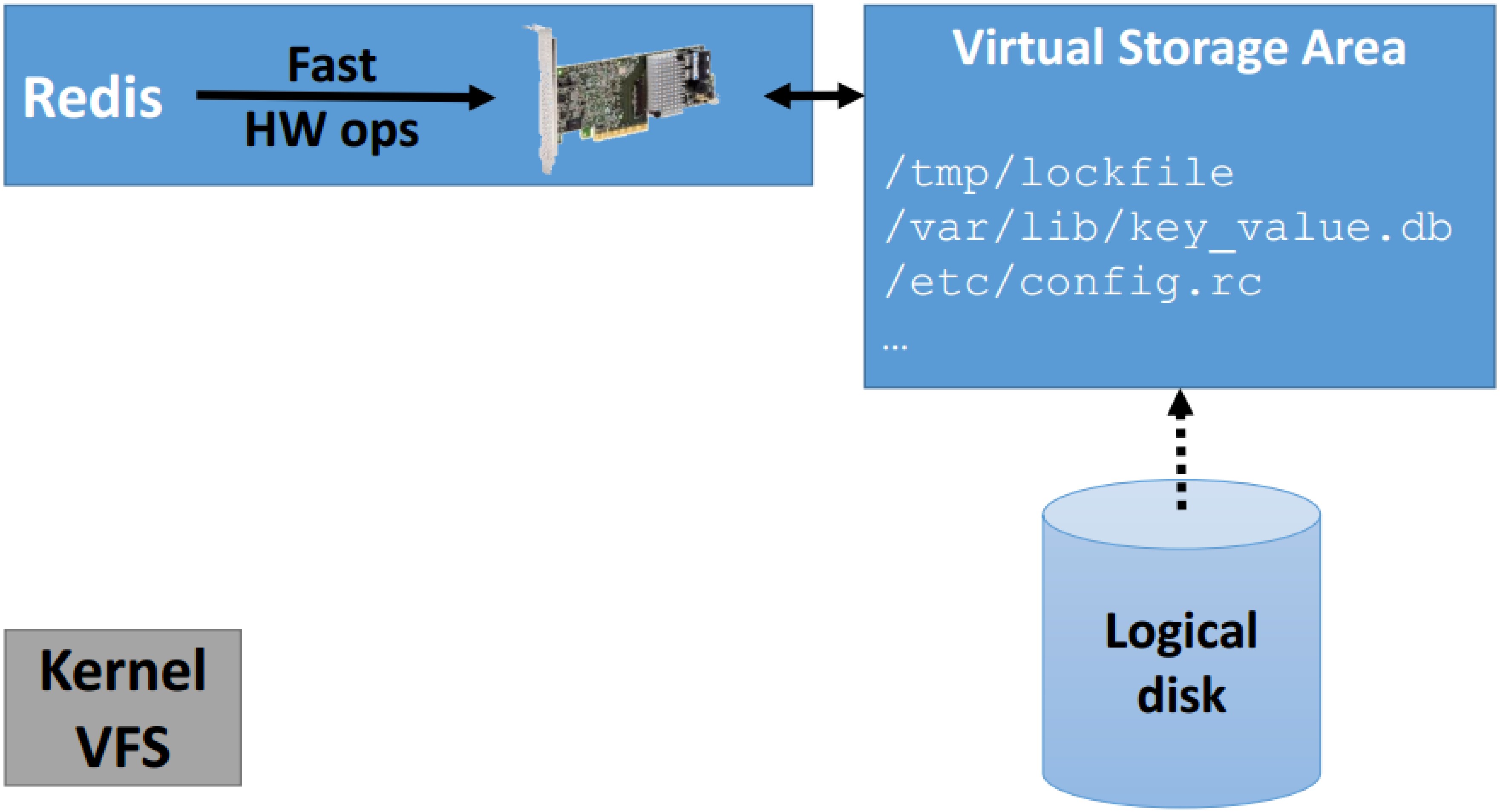
Fig. 4. Arrakis architecture. The storage controller maps VSAs to physical storage.

Arrakis Control Plane

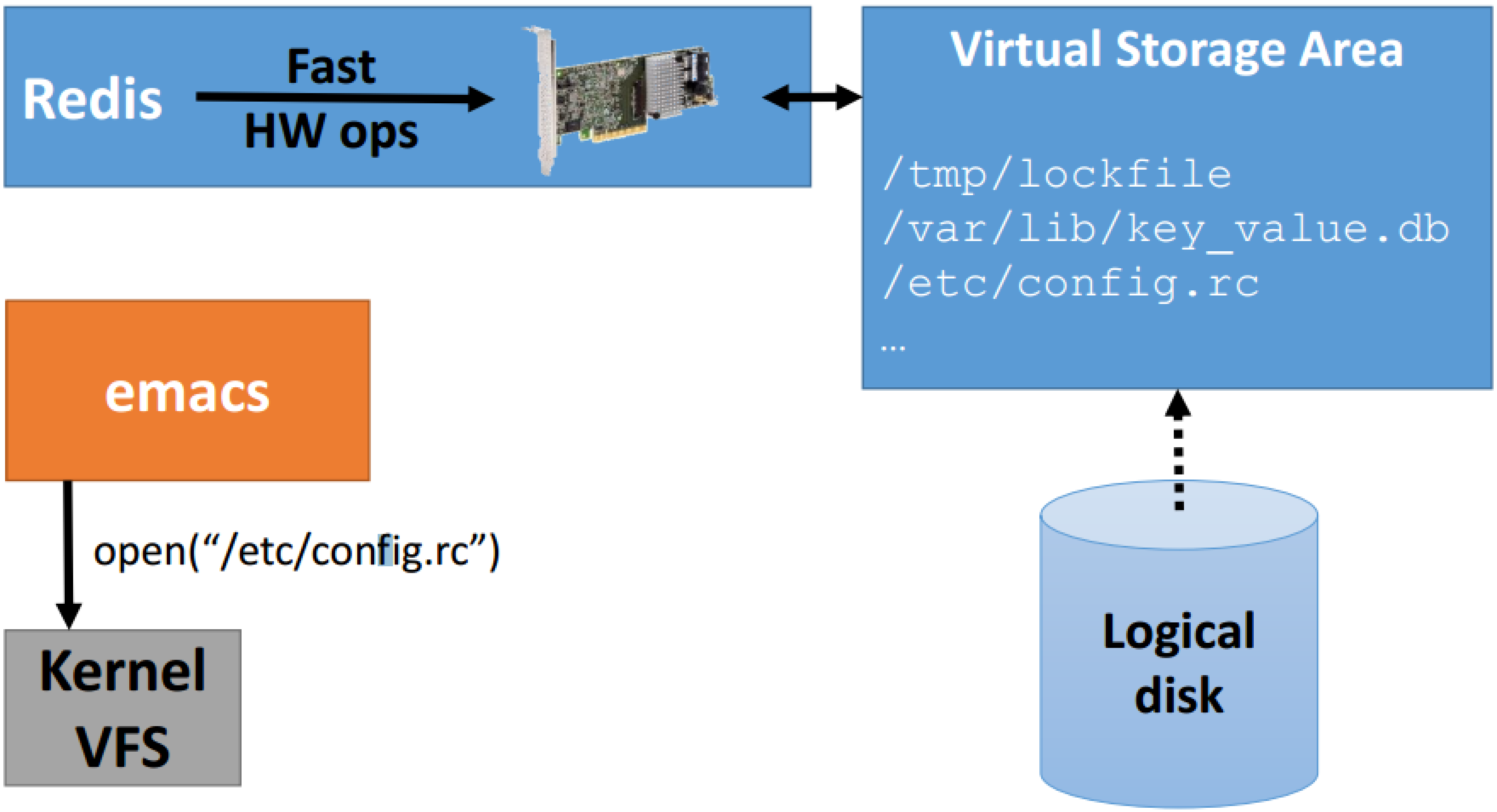


- Access control
  - Do once when configuring data plane
  - Enforced via NIC filters, logical disks
- Resource limits
  - Program hardware I/O schedulers
- Global naming
  - Virtual file system still in kernel
  - Storage implementation in applications

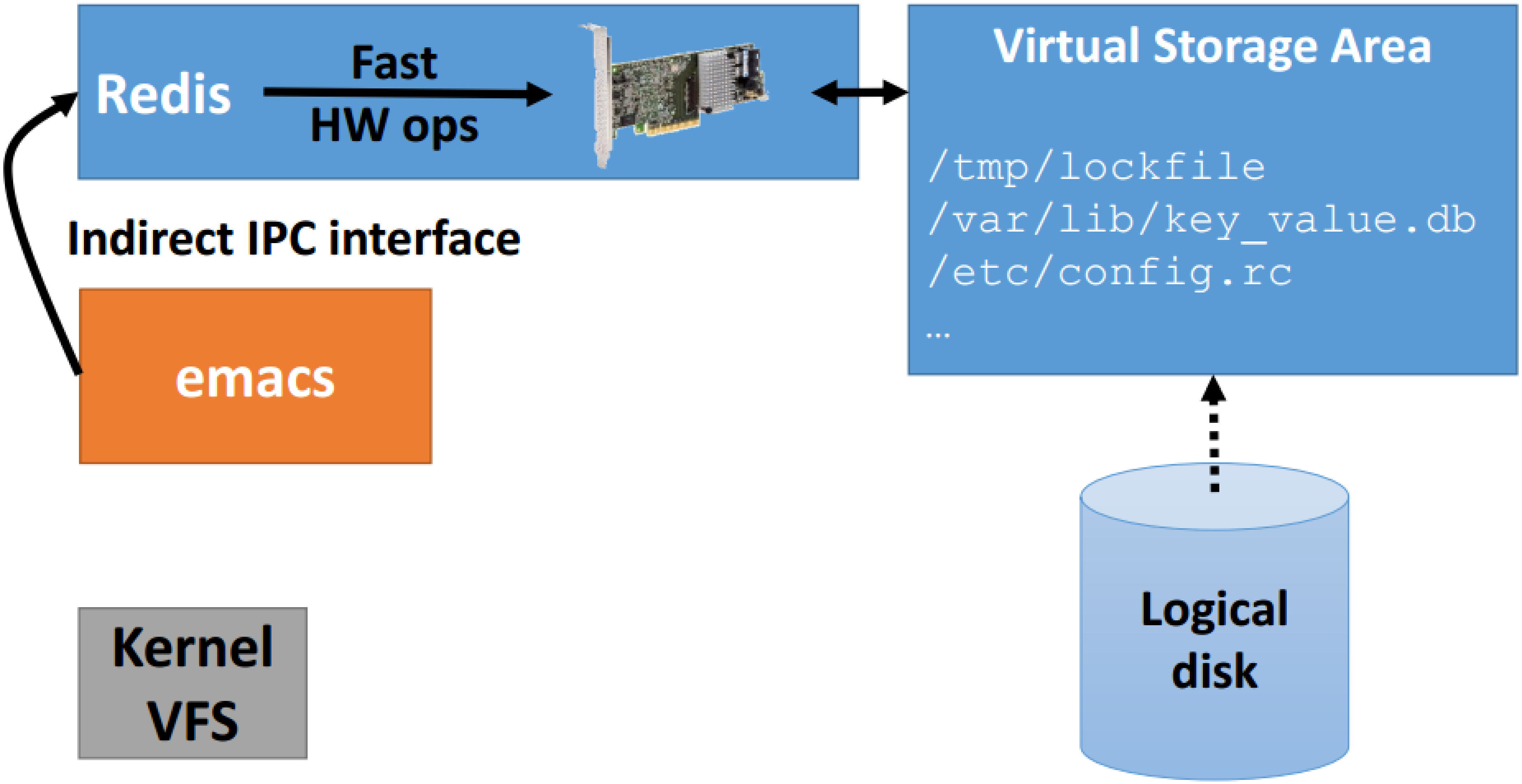
Global Naming



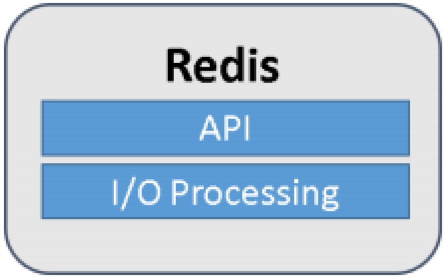
Global Naming



Global Naming



Storage Data Plane: Persistent Data Structures



- Examples: **log, queue**
- Operations immediately persistent on disk

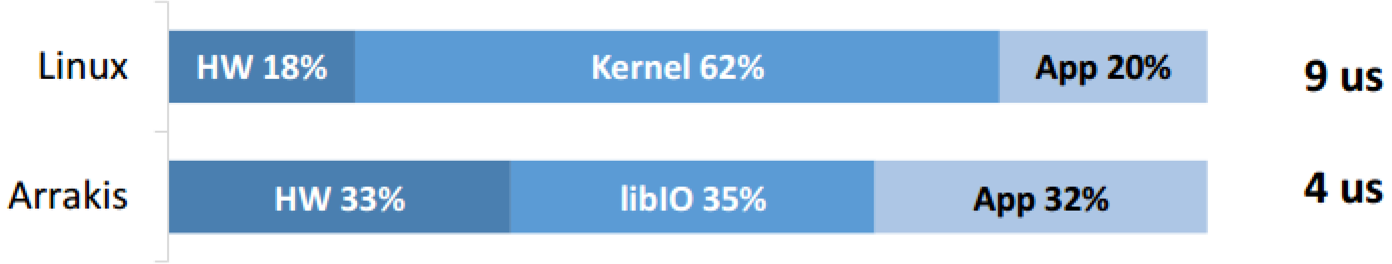
Benefits:

- In-memory = on-disk layout
  - Eliminates marshaling
- Metadata in data structure
  - Early allocation
  - Spatial locality
- Data structure specific caching/prefetching

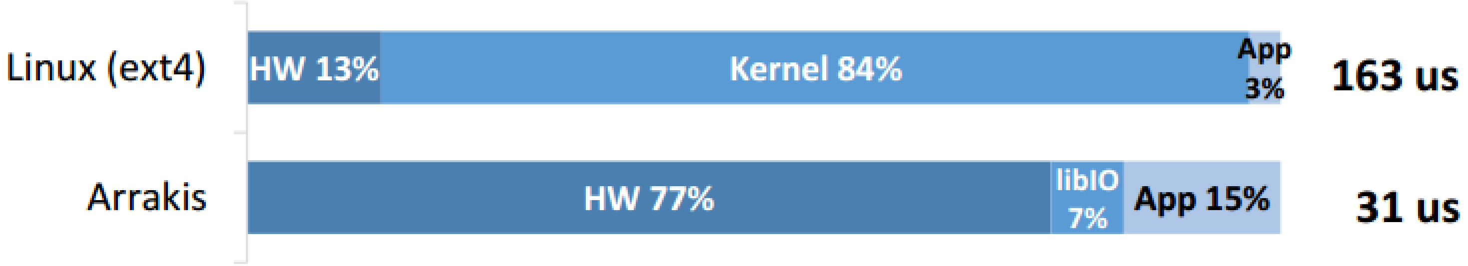
- Modified Redis to use **persistent log**: **109 LOC** changed

Evaluation: Redis Latency

- Reduced (in-memory) GET latency by **65%**



- Reduced (persistent) SET latency by **81%**

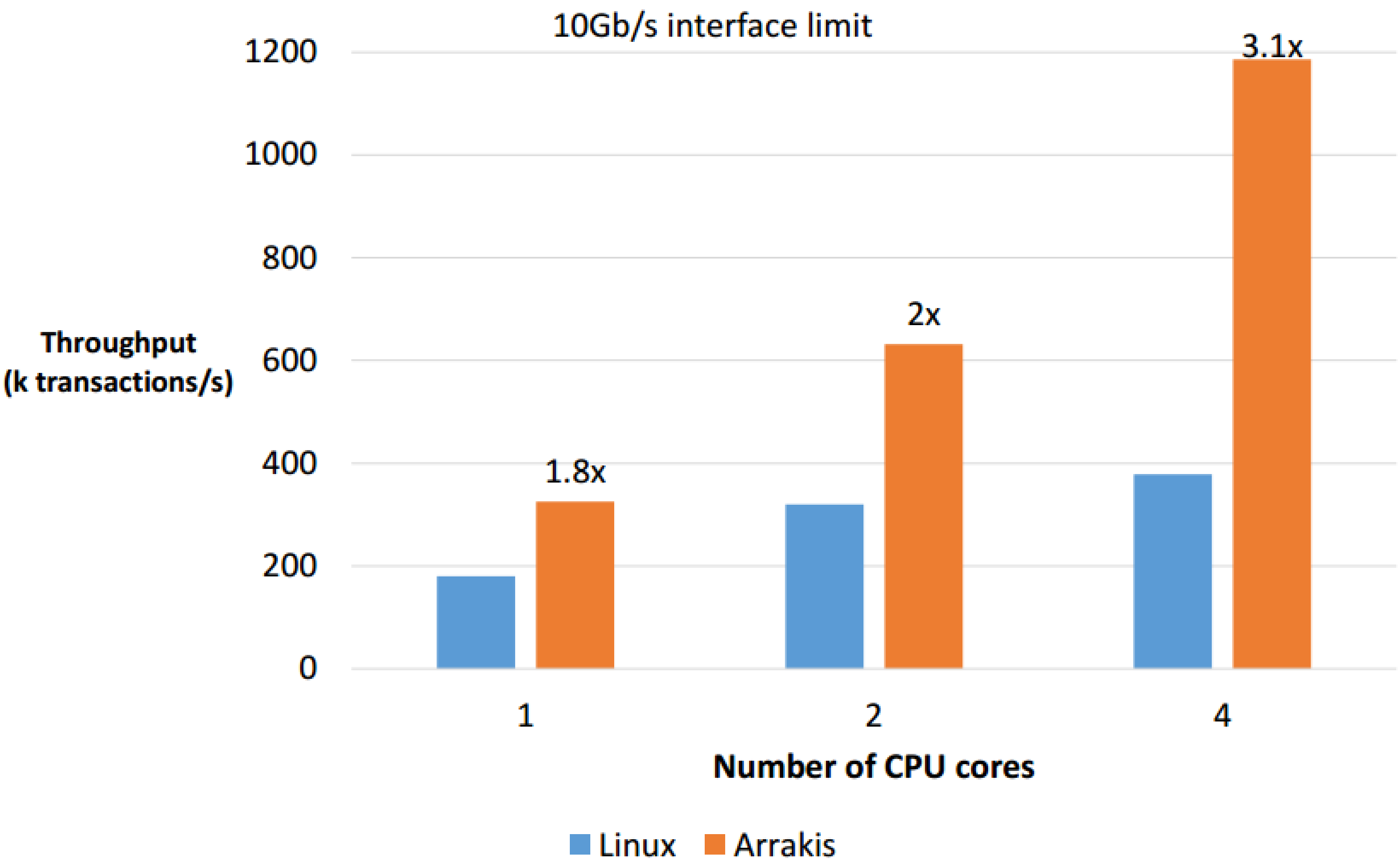


Evaluation:
Redis Throughput

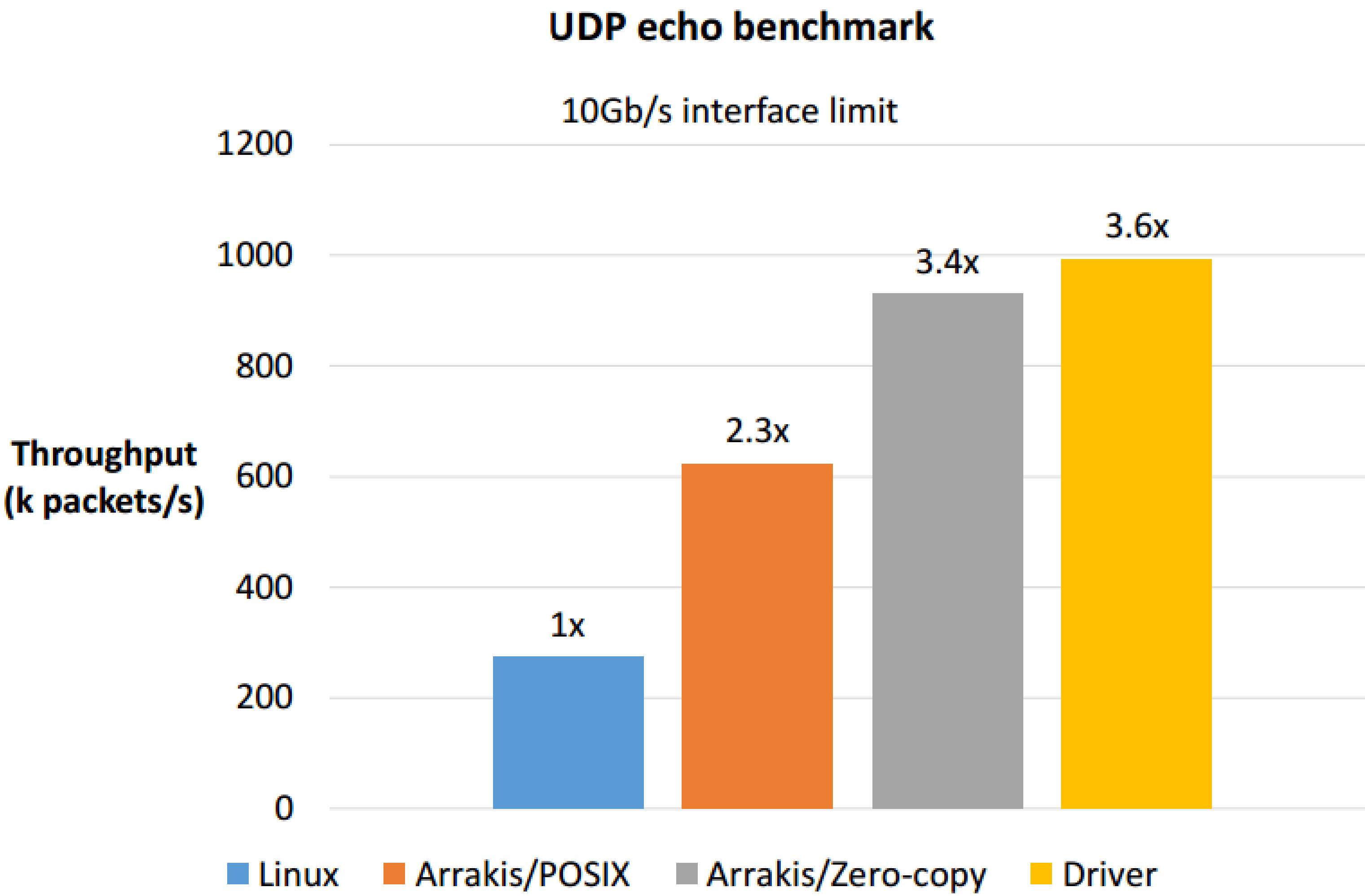
- Improved GET throughput by **1.75x**
  - Linux: **143k** transactions/s
  - Arrakis: **250k** transactions/s

- Improved SET throughput by **9x**
  - Linux: **7k** transactions/s
  - Arrakis: **63k** transactions/s

Evaluation: memcached Scalability



Evaluation:
Single-core Performance



Summary

- OS is becoming an I/O bottleneck
  - Globally shared I/O stacks are slow on data path
- **Arrakis:** Split OS into control/data plane
  - Direct application I/O on data path
  - Specialized I/O libaries
- Application-level I/O stacks deliver great performance
  - **Redis:** up to **9x** throughput, **81%** speedup
  - Memcached **scales linearly** to **3x** throughput