Datacenter Operating Systems

CSE451

Simon Peter

With thanks to Timothy Roscoe (ETH Zurich)

Autumn 2015

This Lecture

- What's a datacenter
  - Why datacenters
  - Types of datacenters
- Hyperscale datacenters
  - Major problem: Server I/O performance
- Arrakis, a datacenter OS
  - Addresses the I/O performance problem (for now)

What's a Datacenter?

- Large facility to house computer systems
  - 10,000s of machines
- Independently powered
  - Consumes as much power as a small town
- First built in the early 2000s
  - In the wake of the Internet
- Runs a large portion of the digital economy

Why Datacenters?

- **Consolidation**
  - Run many people's workloads on the same infrastructure
  - Use infrastructure more efficiently (higher utilization)
  - Leverage workload synergies (eg., caching)
- **Virtualization**
  - Build your own private infrastructure quickly and cheaply
  - Move it around anywhere, anytime
- **Automation**
  - No need for expensive, skilled IT workers
  - Expertise is provided by the datacenter vendor

Types of Datacenters

- Supercomputers
  - Compute intensive
  - Scientific computing: weather forecast, simulations, …
- **Hyperscale** (this lecture)
  - I/O intensive => Makes for cool OS problems
  - Large-scale web services: Google, Facebook, Twitter, …
- Cloud
  - Virtualization intensive
  - Everything else: “Smaller” businesses (eg., Netflix)

Hyperscale Datacenters

- *Hyperscale*: Provide services to billions of users
- Users expect response at interactive timescales
  - Within milliseconds
- Examples: Web search, Gmail, Facebook, Twitter

- Built as *multi-tier* application
  - Front end services: Load balancer, web server
  - Back end services: database, locking, replication
- Hundreds of servers contacted for 1 user request
  - Millions of requests per second per server

Hyperscale: I/O Problems

Hardware trend

- Network & stoage speeds keep on increasing
  - 10-100 Gb/s Ethernet
  - Flash storage
- CPU frequencies don’t
  - 2-4 GHz

- 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

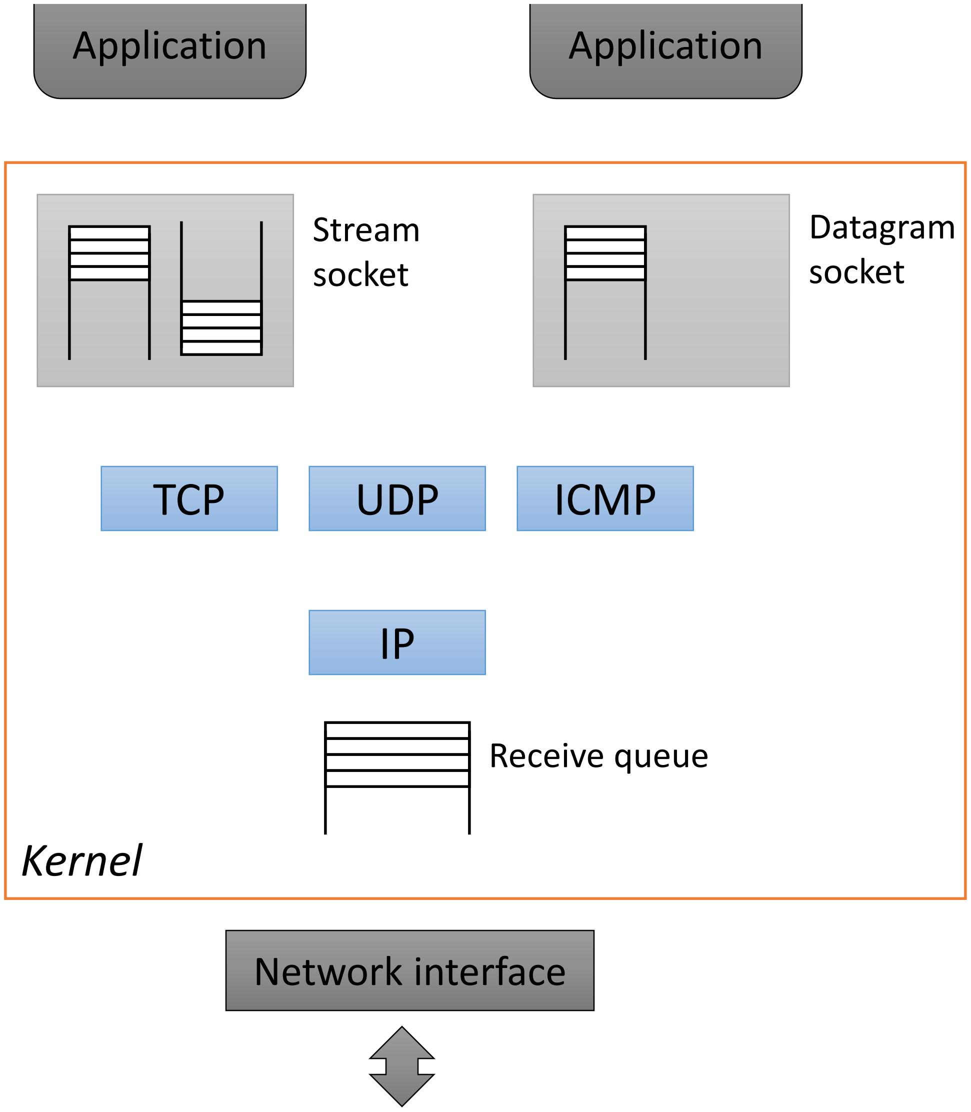
=

Hyperscale: OS I/O Problems

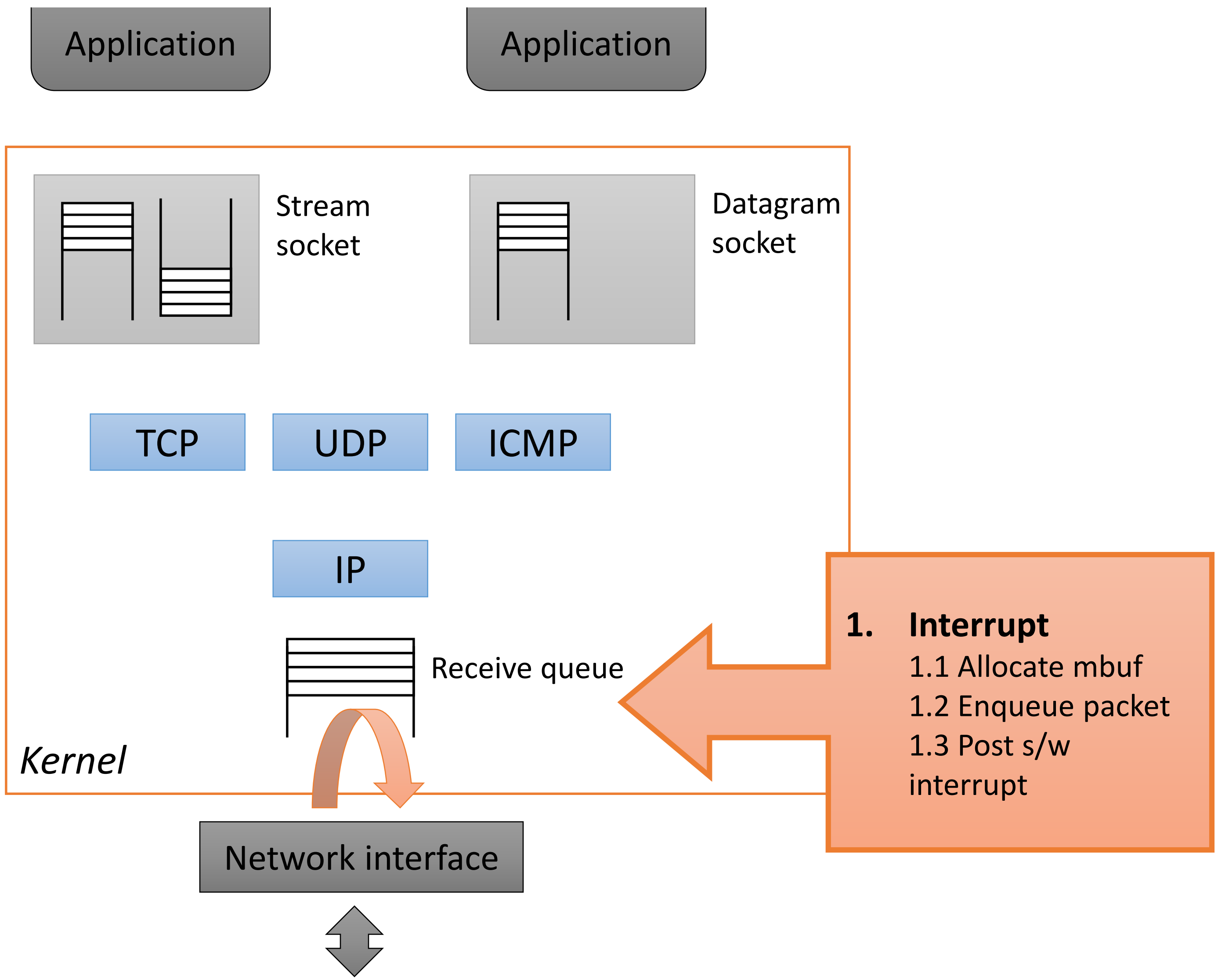
OS problem

- Traditional OS: Kernel-level I/O processing => slow
  - Shared I/O stack => Complex
  - Layered design => Lots of indirection
  - Lots of copies

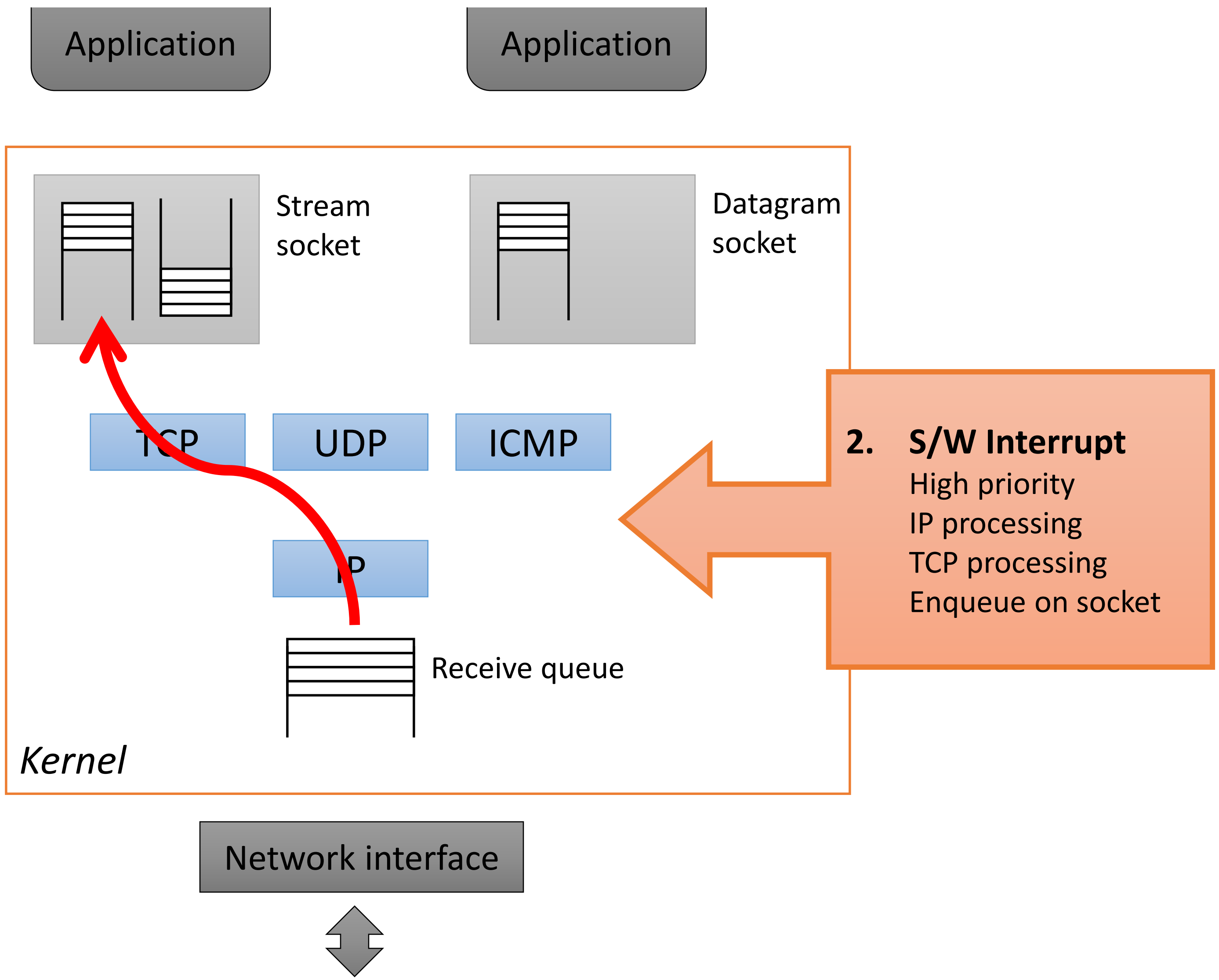
Receiving a packet in BSD



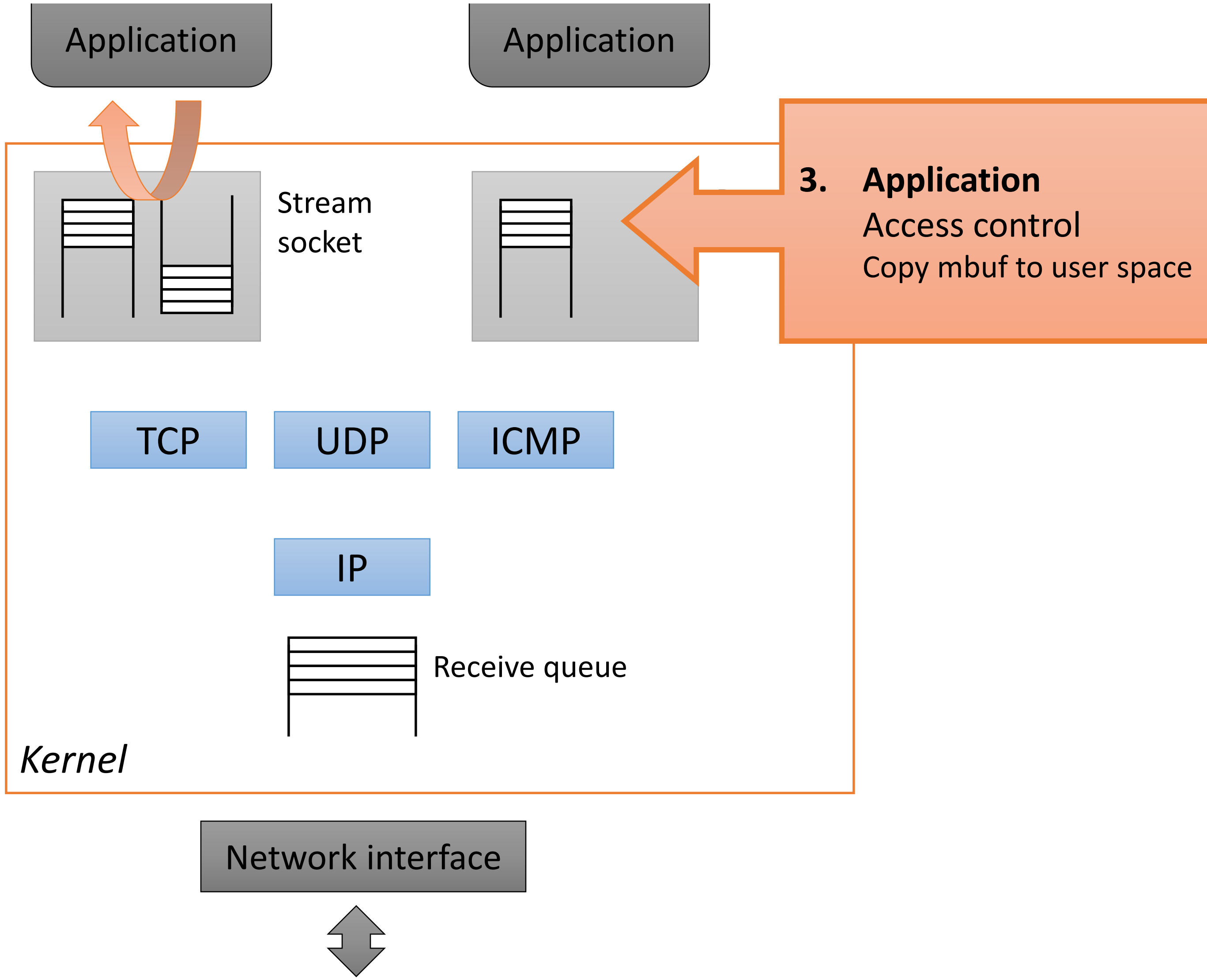
Receiving a packet in BSD



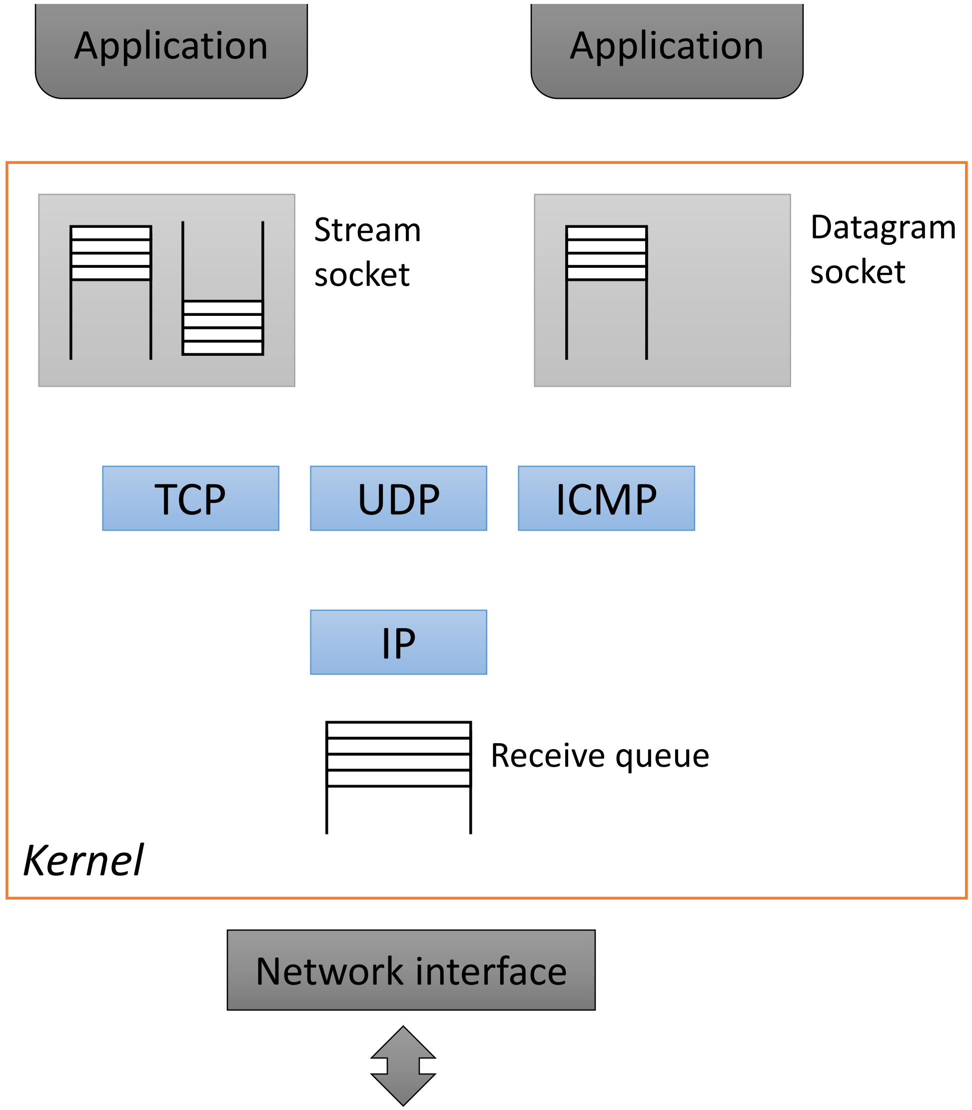
Receiving a packet in BSD



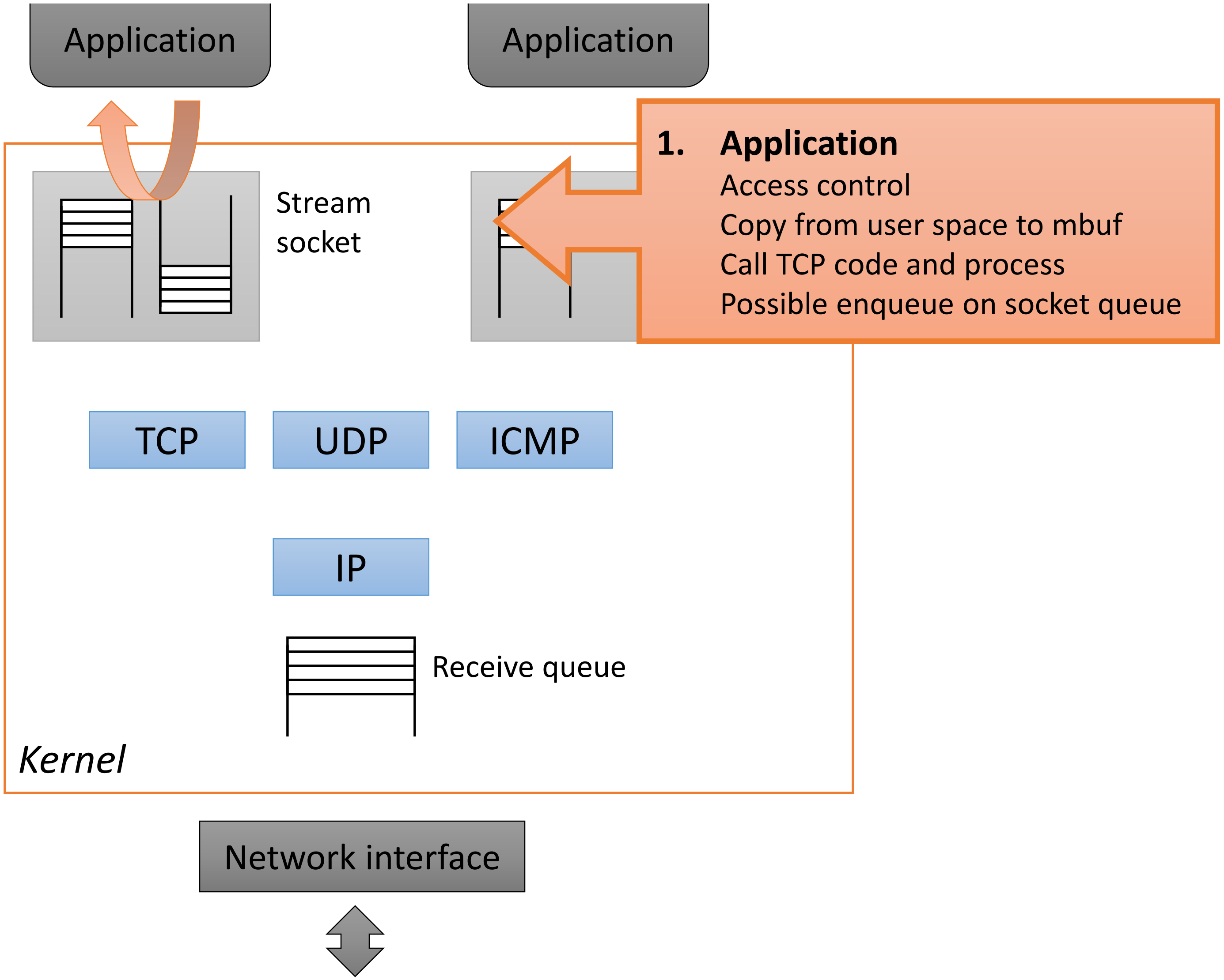
Receiving a packet in BSD



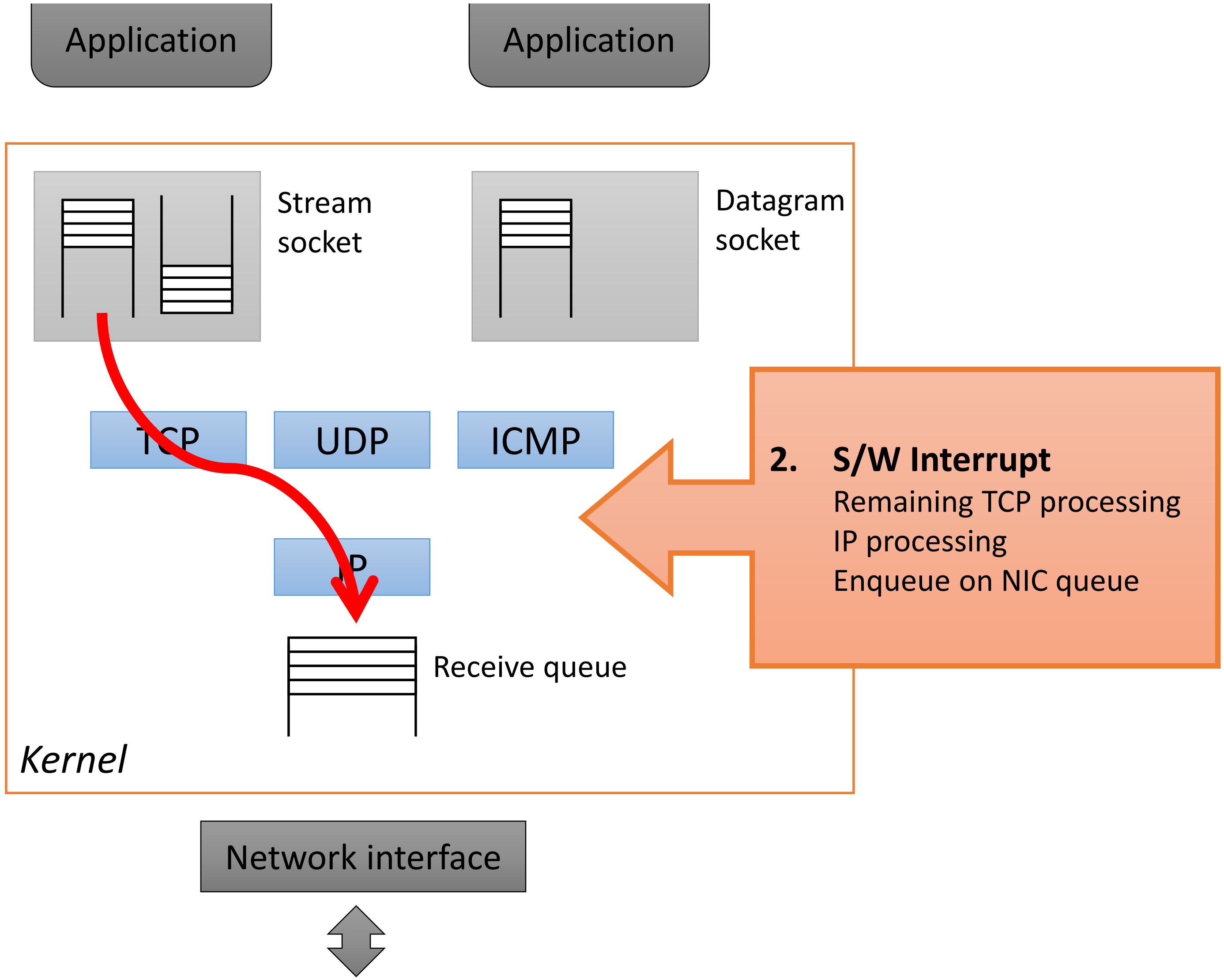
Sending a packet in BSD



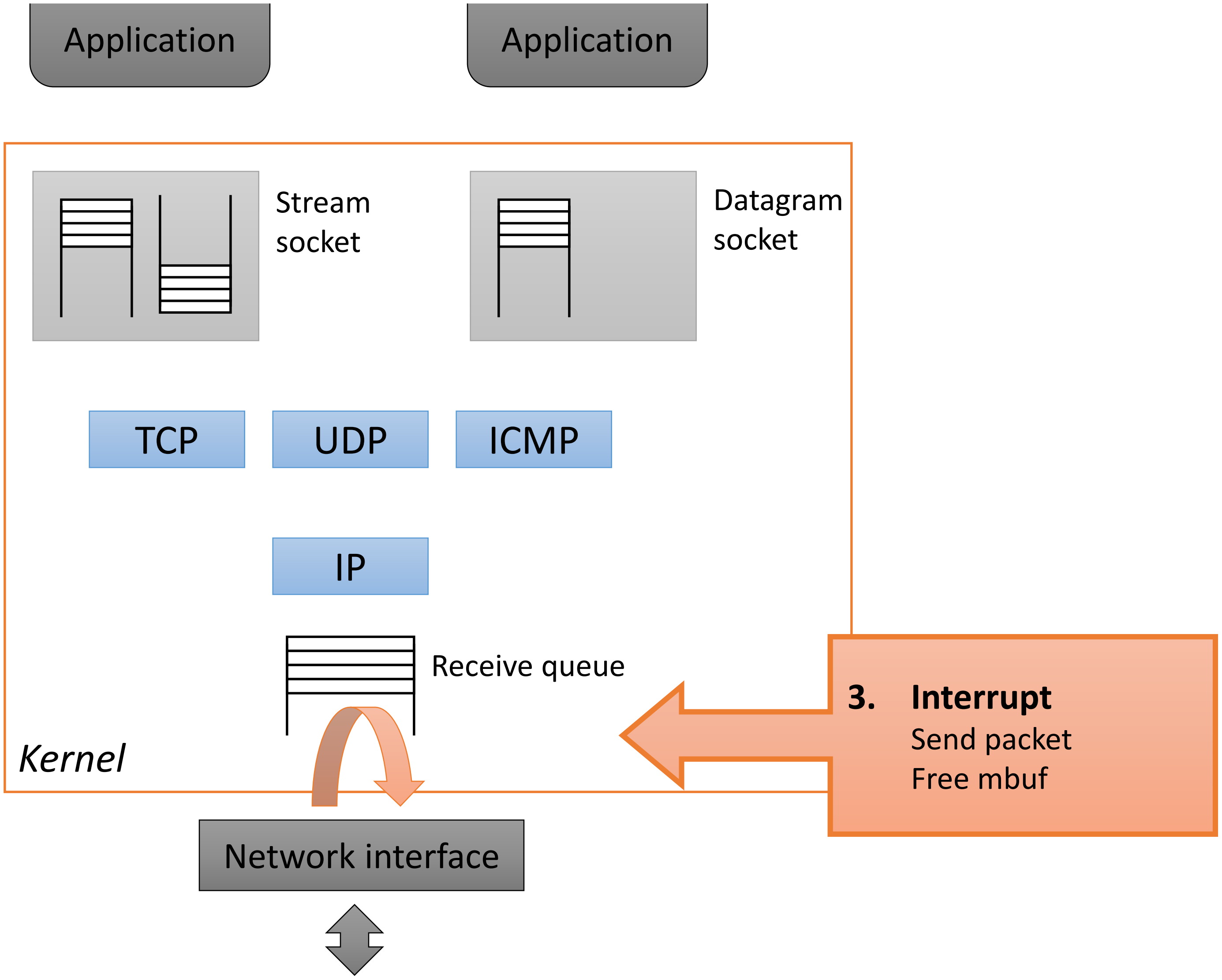
Sending a packet in BSD



Sending a packet in BSD

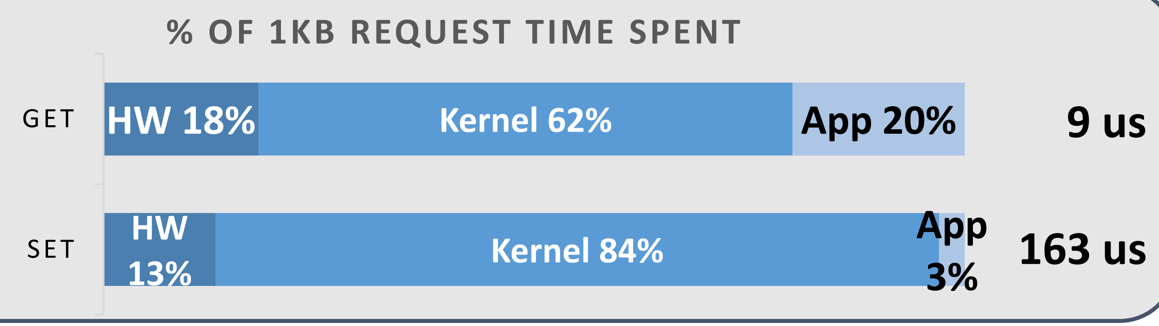


Sending a packet in BSD



Linux I/O Performance

Redis



Kernel

- API
- Multiplexing
- Naming
- Resource limits
- Access control
- I/O Scheduling
- I/O Processing
- Copying
- Protection

Data Path

10G NIC
2 us / 1KB packet

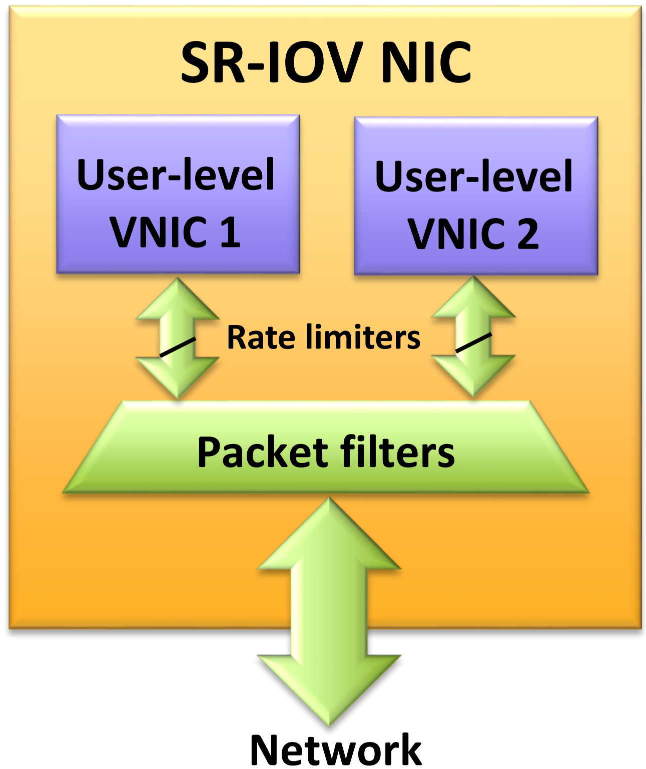
RAID Storage
25 us / 1KB write

Arrakis Datacenter OS

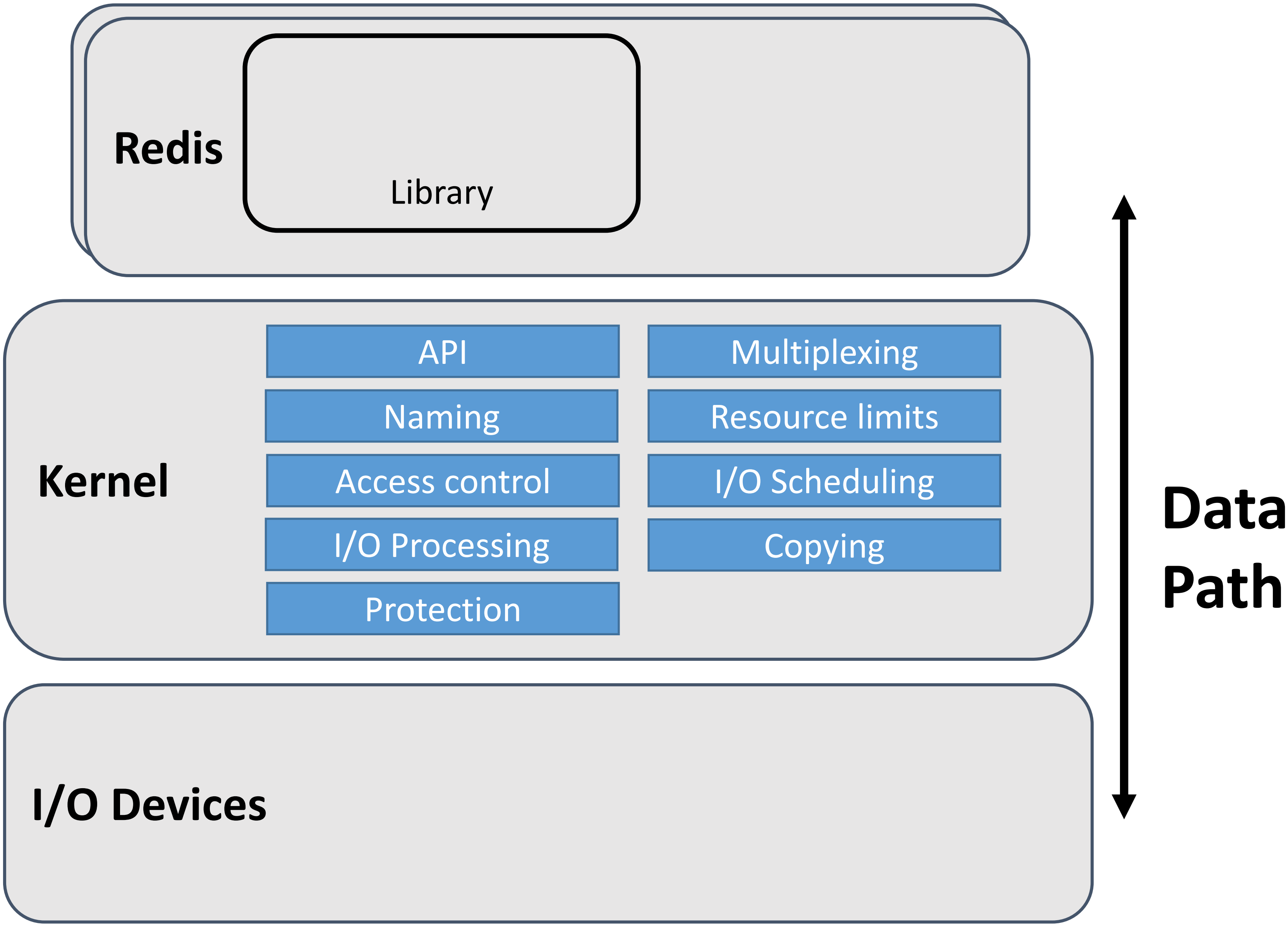
- **Can we deliver performance closer to hardware?**
- Goal: Skip kernel & deliver I/O directly to applications
  - Reduce OS overhead
- Keep classical server OS features
  - Process protection
  - Resource limits
  - I/O protocol flexibility
  - Global naming
- The hardware can help us…

Hardware I/O Virtualization

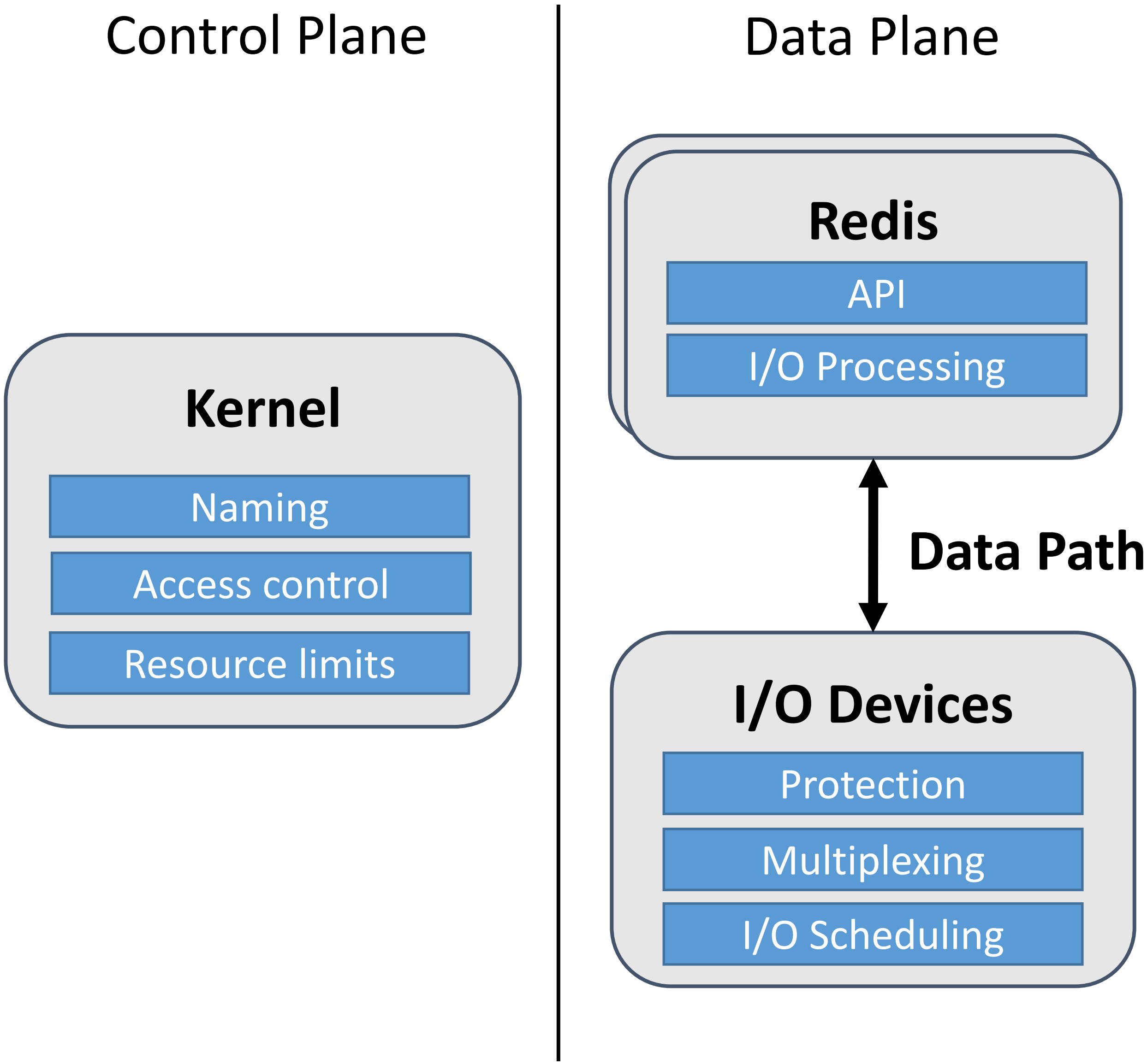
- Standard on NIC, emerging on RAID
- Multiplexing
  - **SR-IOV**: Virtual PCI devices w/ own registers, queues, INTs
- Protection
  - **IOMMU**: Devices use app virtual memory
  - **Packet filters**, **logical disks**: Only allow eligible I/O
- I/O Scheduling
  - NIC **rate limiter**, **packet schedulers**



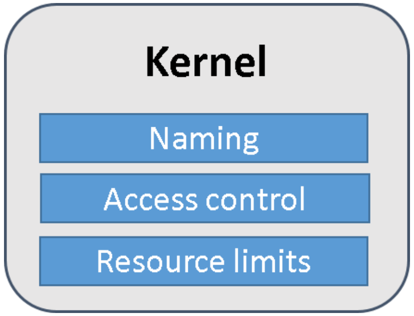
How to skip the kernel?



Arrakis I/O Architecture



Arrakis Control Plane

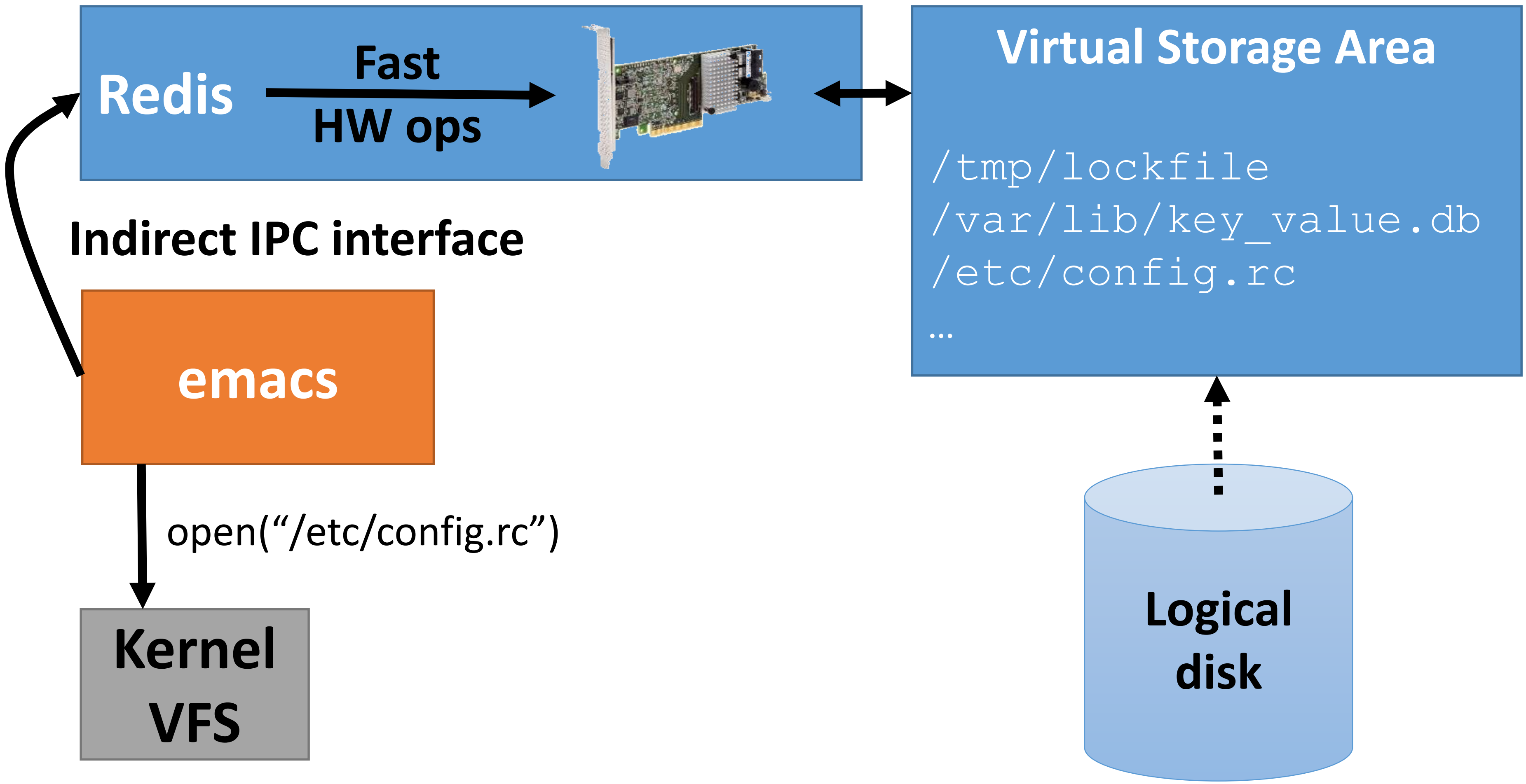


- Access control
  - Do once when configuring data plane
  - Enforced via NIC filters, logical disks

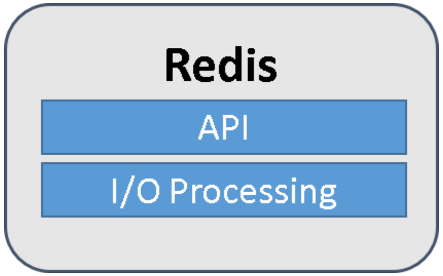
- Resource limits
  - Program hardware I/O schedulers

- Global naming
  - Virtual file system still in kernel
  - Storage implementation in applications

Global Naming



Storage Data Plane: Persistent Data Structures



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

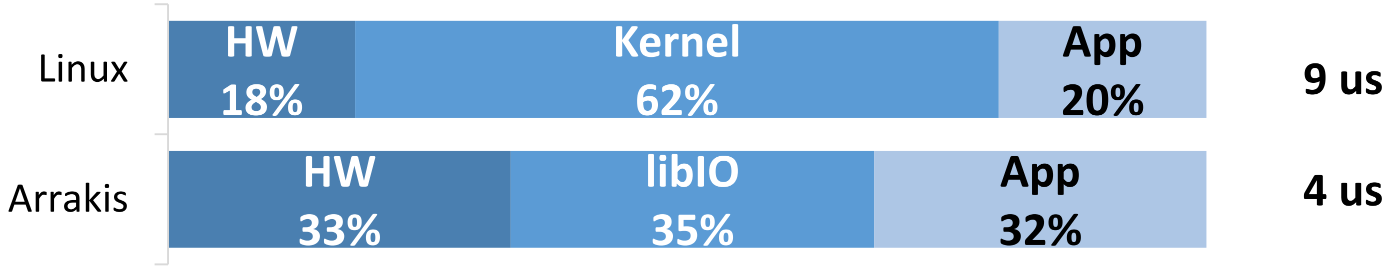
Benefits:

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

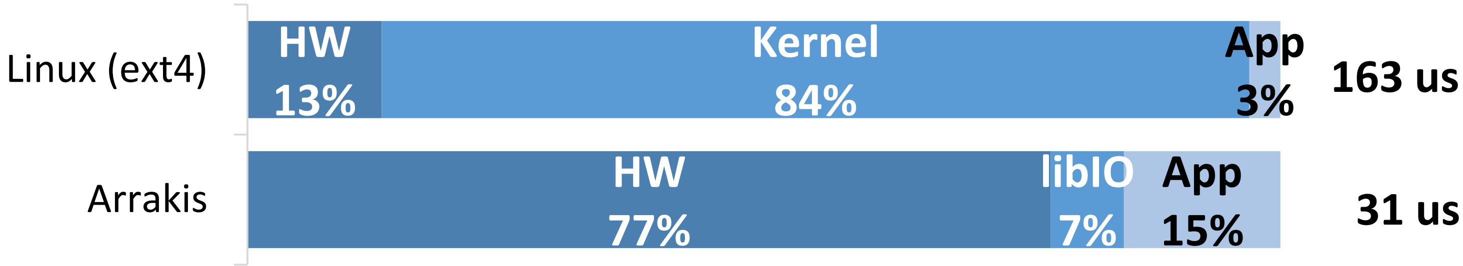
- Modified Redis to use **persistent log**: **109 LOC** changed

Redis Latency

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



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

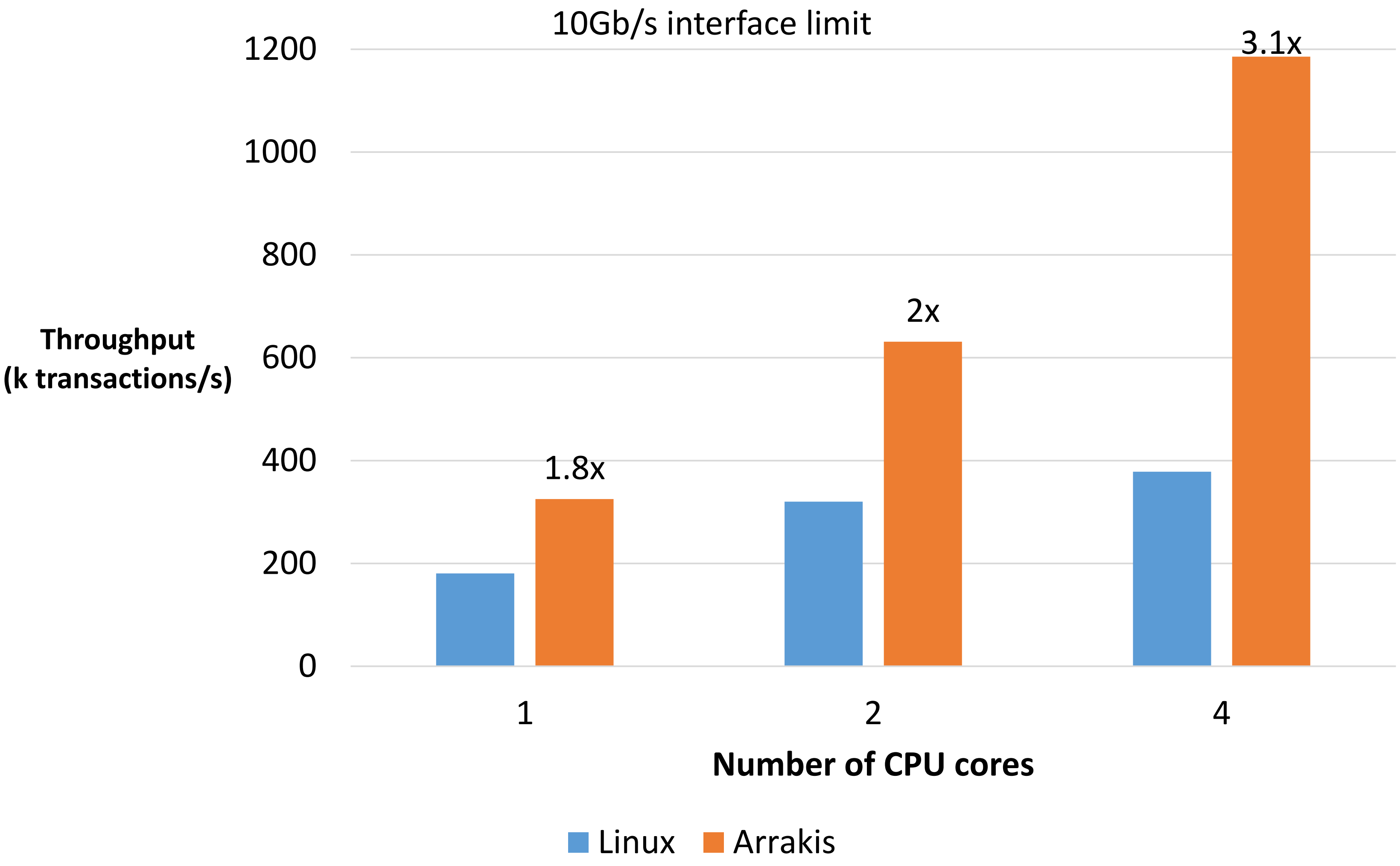


Redis Throughput

- 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

memcached Scalability



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 libraries

- Application-level I/O stacks deliver great performance
  - Redis: up to 9x throughput, 81% speedup
  - Memcached scales linearly to 3x throughput

Interested?

- **I am recruiting PhD students**
- I work at **UT Austin**

- Apply to UT Austin's PhD program:

http://services.cs.utexas.edu/recruit/grad/frontmatter/announcement.html