


Arrakis: The Operating System is the Control Plane

Cornell CS 6410 Fall 2025
Presented by: Sheng-Yen Chou


Building an OS for the Data Center

- Server I/O performance matters
 - Key-value stores, web & file servers, lock managers, ...
- **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

Can't we just use Linux?

Linux Network Architecture

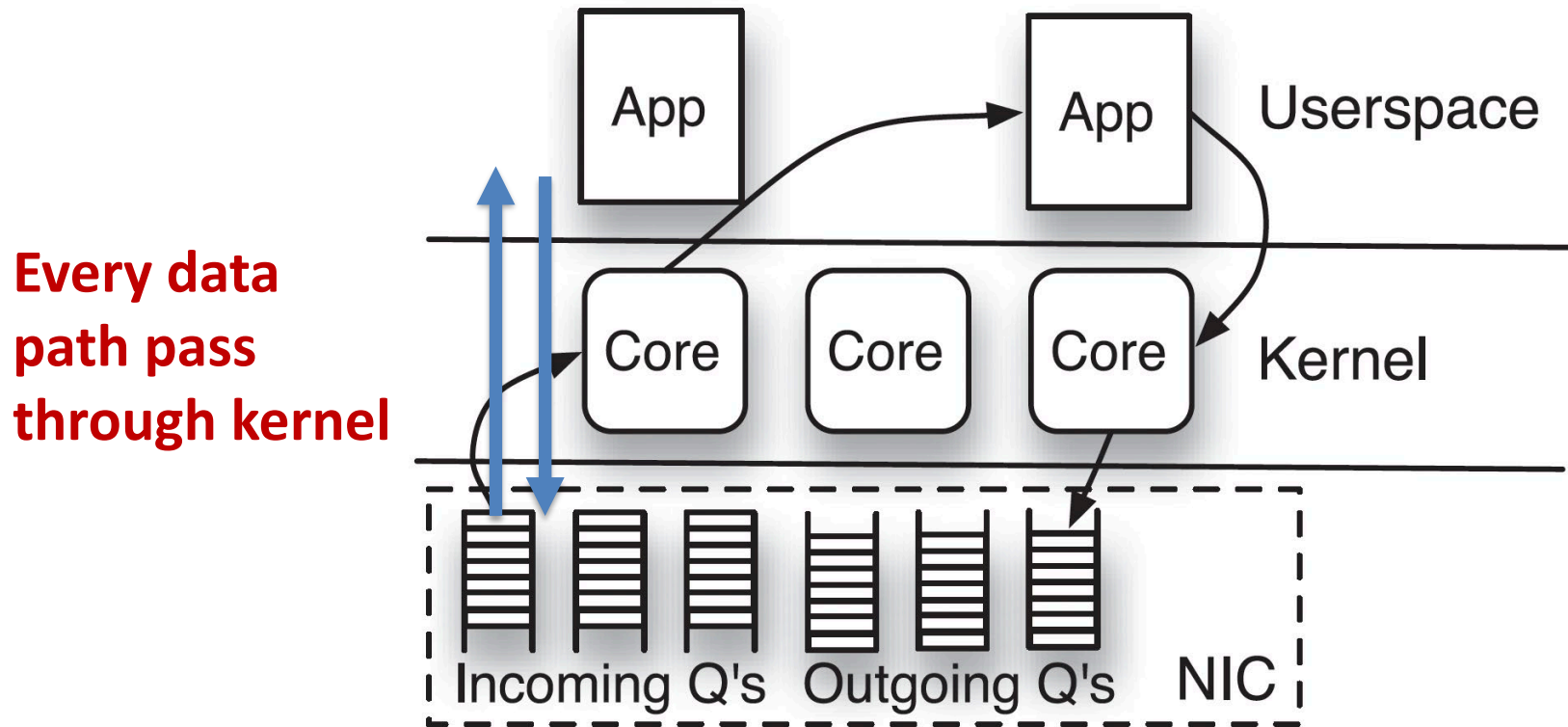


Fig. 1. Linux networking architecture and workflow.

However, I/O bandwidth Surpassed CPU bandwidth

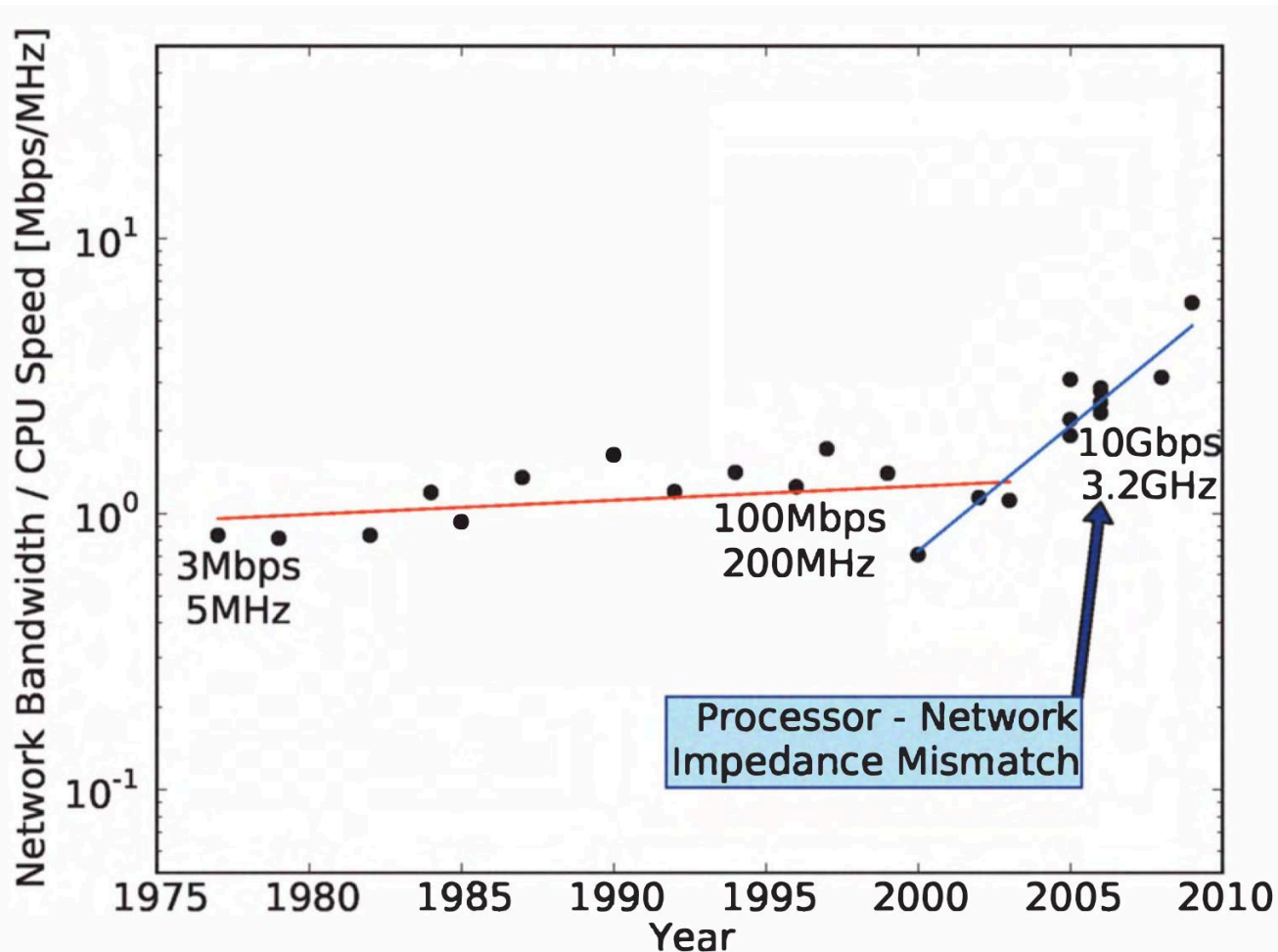
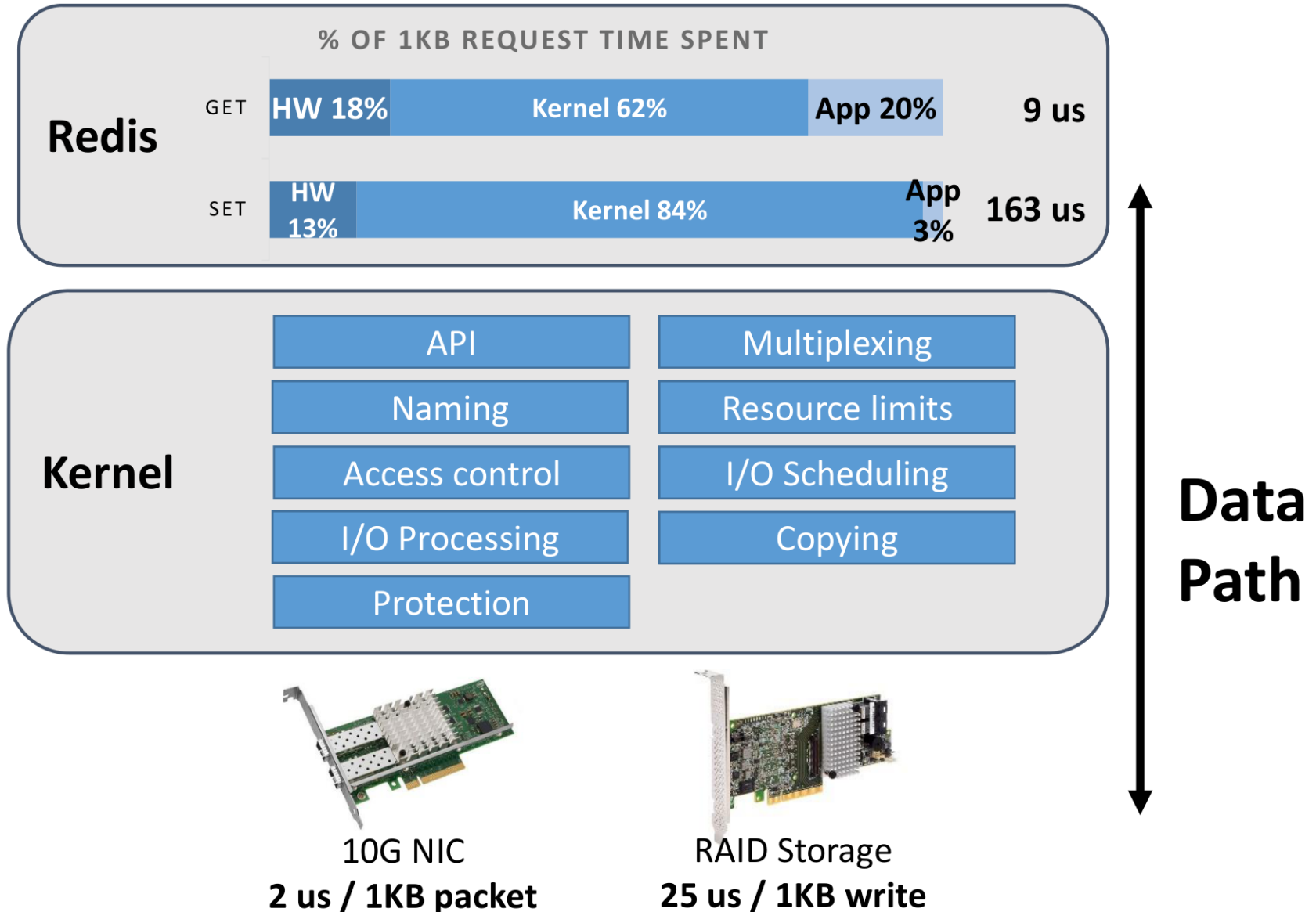


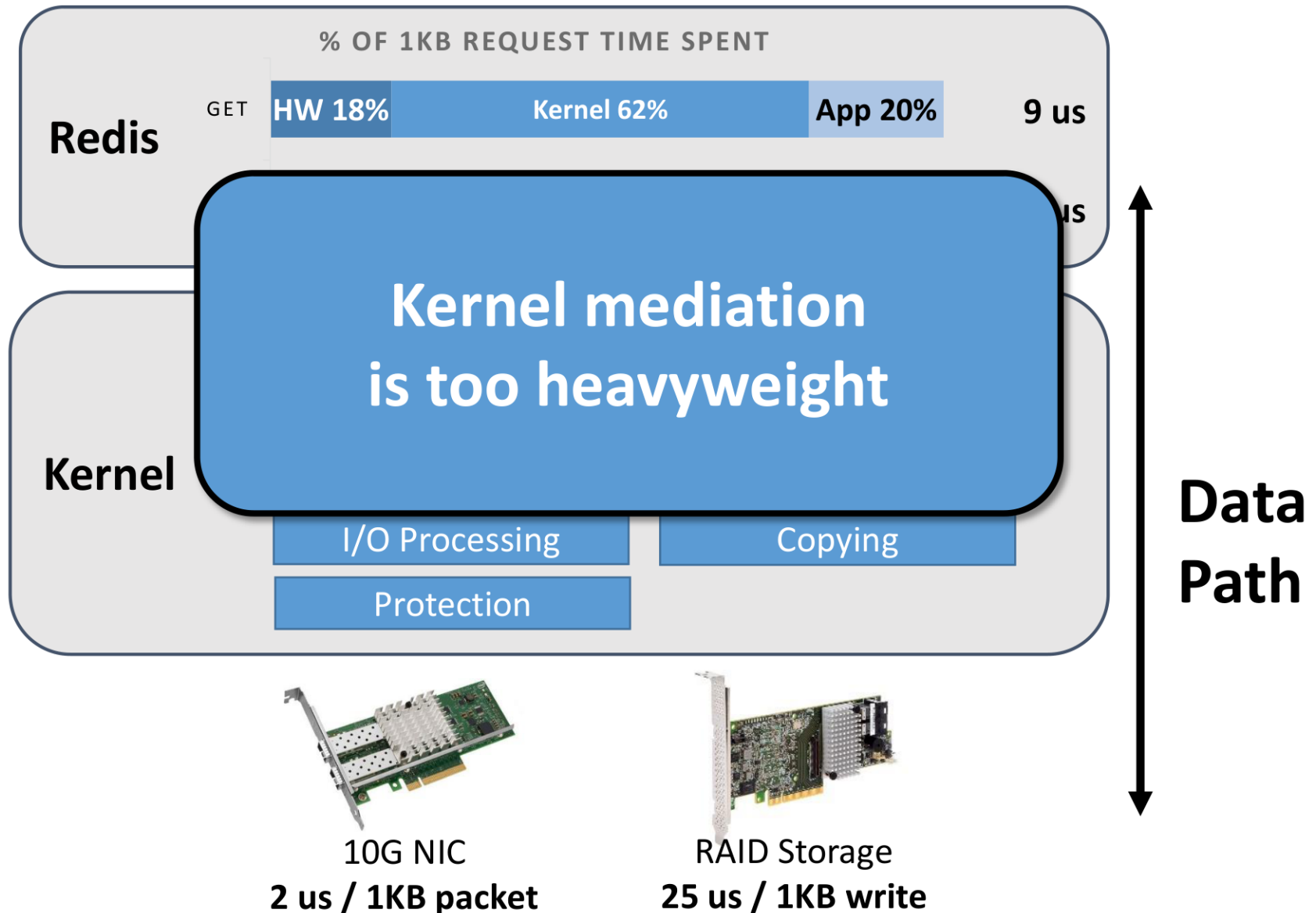
Figure 1. Network to processor speed ratio.

Photo Credit: [Empirical Characterization of Uncongested Optical Lambda Networks and 10GbE Commodity Endpoints](#)

Linux I/O Performance



Linux I/O Performance



Discussion

- **How to deal with this issue?**
- **Why Linux let every data path pass through kernel?**

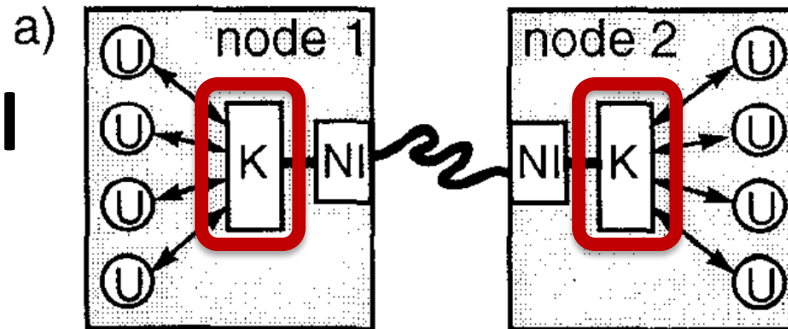
Existing Work from Cornell: SIGOP'95

U-Net: A User-Level Network Interface for Parallel and Distributed Computing

Proposes

Remove kernel from data path

Traditional Network



Kernel interleaves network data path

Legend:

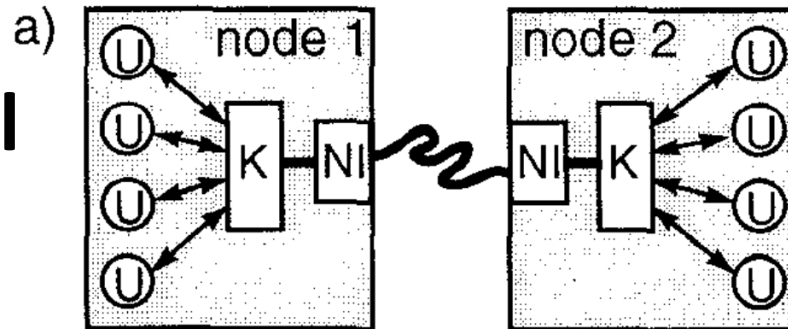
Ⓢ User application

Ⓚ Operating system kernel

Ⓝ Network interface

Ⓝ NI with message multiplex

Traditional Network



Legend:

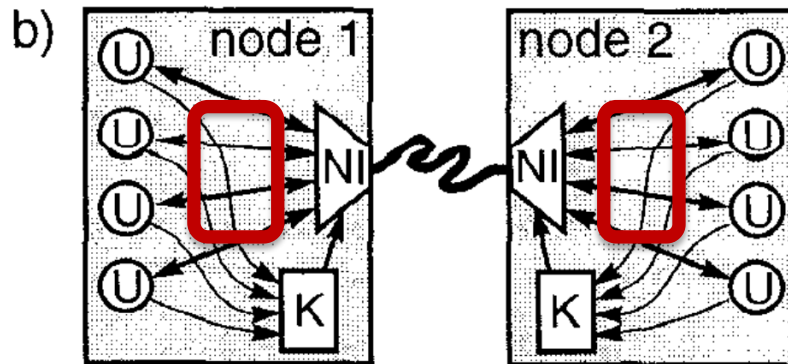
Ⓢ User application

Ⓚ Operating system kernel

ⓃⓈ Network interface

ⓃⓈ NI with message multiplex

U-Net



Kernel is removed from the data path

Arrakis:

The Operating System is the Control Plane

Simon Peter, Jialin Li, Irene Zhang,
Dan Ports, Doug Woos,
Arvind Krishnamurthy, Tom Anderson
University of Washington

Timothy Roscoe
ETH Zurich

Authors



Simon Peter
UW



Saint Peter



Dan R. K. Ports
MSR



Irene Zhang
MSR



Jialin Li
NUS



Doug Woos
Voltus



Timothy Roscoe
ETHz

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

Outline

1. Existing I/O Path without Kernel Involving
 1. SR-IOV
 2. RDMA
2. Arrakis' Main Idea
3. Arrakis' Components
 1. Control Panel in Kernel
 2. Network Interface Card Hardware Model (NIC and VNIC)
 3. Storage Hardware Model (VSA and VSIC)
4. Cross Application Files Read / Write
5. Designs of Persistent Data Structures
6. Evaluation

Outline

1. Existing I/O Path without Kernel Involving

1. SR-IOV

2. RDMA

2. Arrakis' Main Idea

3. Arrakis' Components

1. Control Panel in Kernel

2. Network Interface Card Hardware Model (NIC and VNIC)

3. Storage Hardware Model (VSA and VSIC)

4. Cross Application Files Read / Write

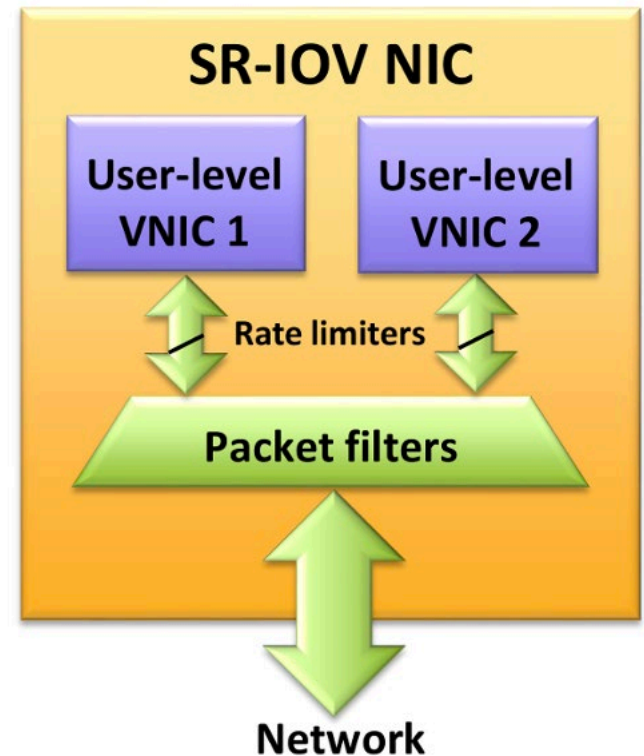
5. Designs of Persistent Data Structures

6. Evaluation

Existing Methods – SR-IOV

Single-Root I/O Virtualization (SR-IOV):

1. It's a hardware virtualization technology. It **allows a single physical PCIe device**, like a Network Interface Card (NIC), **to be split and appear as multiple independent virtual devices (VNIC)**.
2. Protection: IOMMU, Packet Filter, and Logical Disk

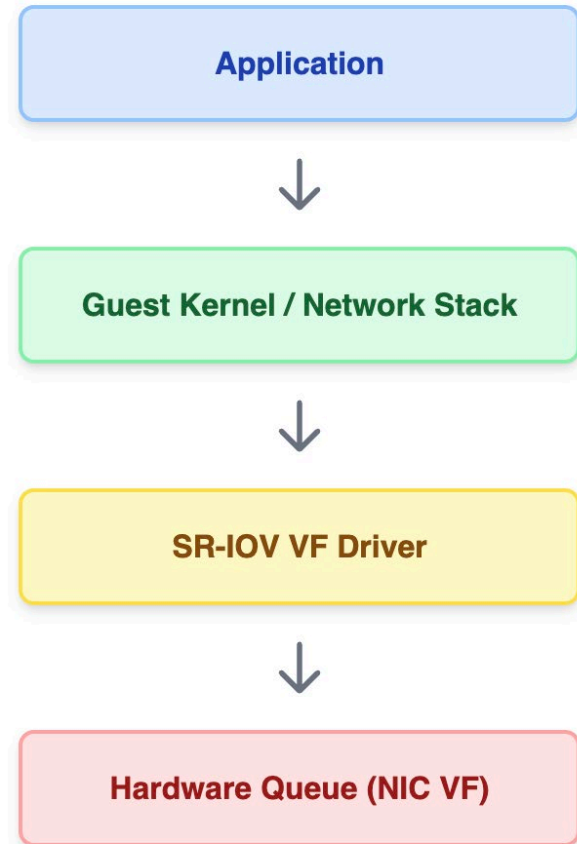


Existing Methods – SR-IOV

Drawback:

1. Although it doesn't need the kernel supervisor to manage the device, **the OS kernel still mediates access to data packet.**
2. By the year of paper published, **SR-IOV is rarely supported by the storage and I/O devices.**

SR-IOV Data Flow



Outline

1. Existing I/O Path without Kernel Involving

- 1. SR-IOV

2. RDMA

- 2. Arrakis' Main Idea

- 3. Arrakis' Components

 - 1. Control Panel in Kernel

 - 2. Network Interface Card Hardware Model (NIC and VNIC)

 - 3. Storage Hardware Model (VSA and VSIC)

- 4. Cross Application Files Read / Write

- 5. Designs of Persistent Data Structures

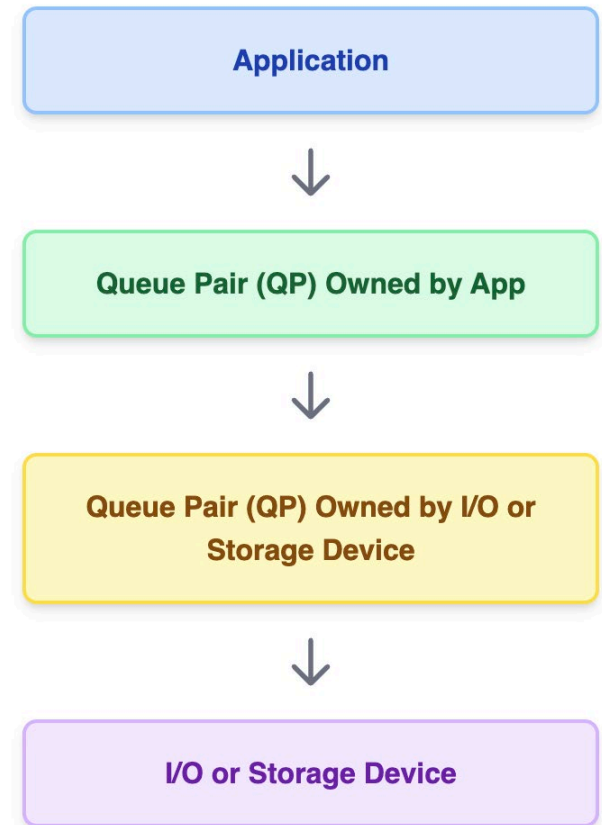
- 6. Evaluation

Existing Methods - RDMA

RDMA Flow: App to Storage Device

Remote Direct Memory Access (RDMA):

1. **Read / write a region of virtual memory on a remote machine directly from user-space**, bypassing the operating system kernel on both sides.
2. Protection: IOMMU
3. **Drawback:**
 1. It's hard to apply it to broader scenarios because client and servers don't trust each other.



Outline

1. Existing I/O Path without Kernel Involving
 1. SR-IOV
 2. RDMA
- 2. Arrakis' Main Idea**
3. Arrakis' Components
 1. Control Panel in Kernel
 2. Network Interface Card Hardware Model (NIC and VNIC)
 3. Storage Hardware Model (VSA and VSIC)
4. Cross Application Files Read / Write
5. Designs of Persistent Data Structures
6. Evaluation

Arrakis Architecture

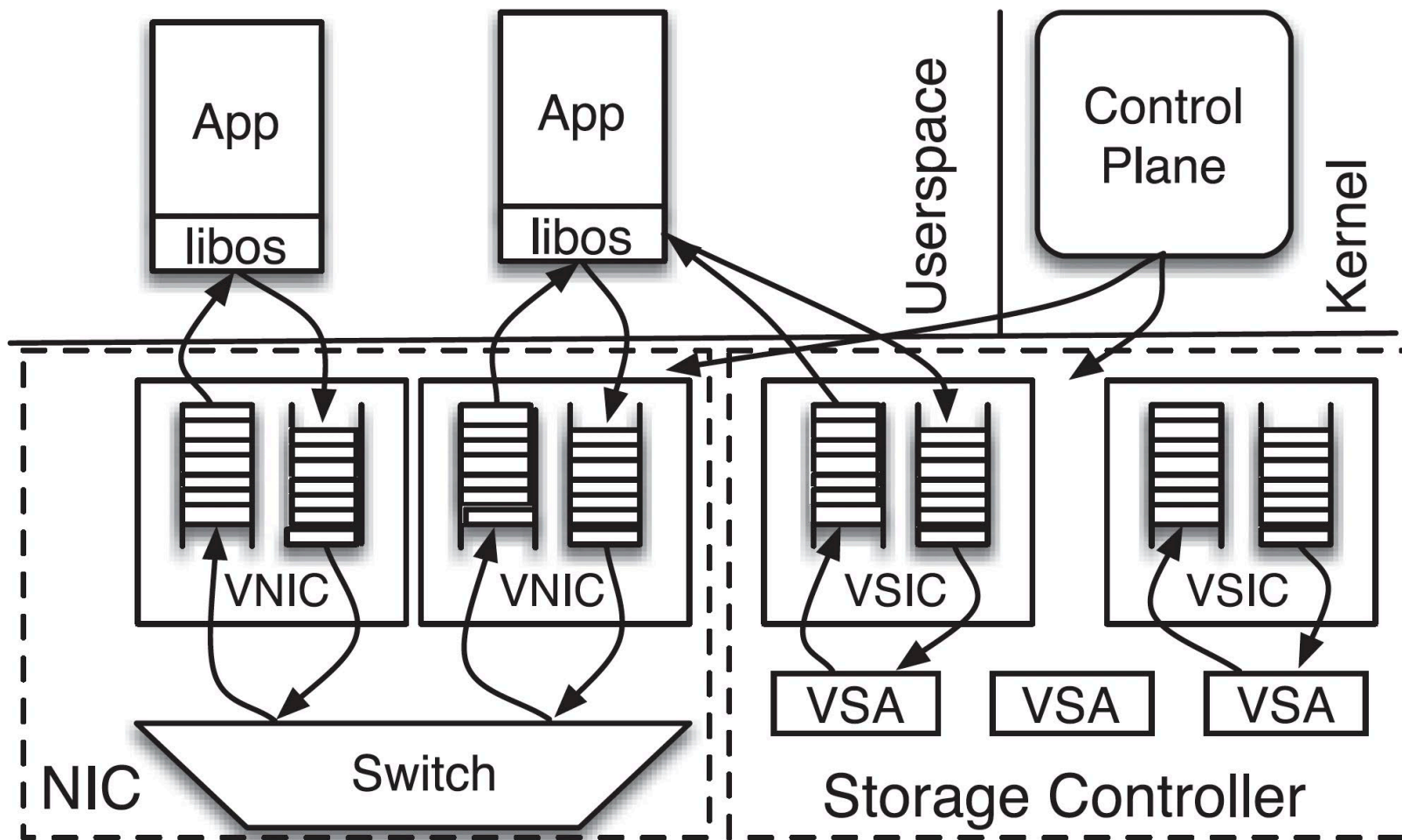
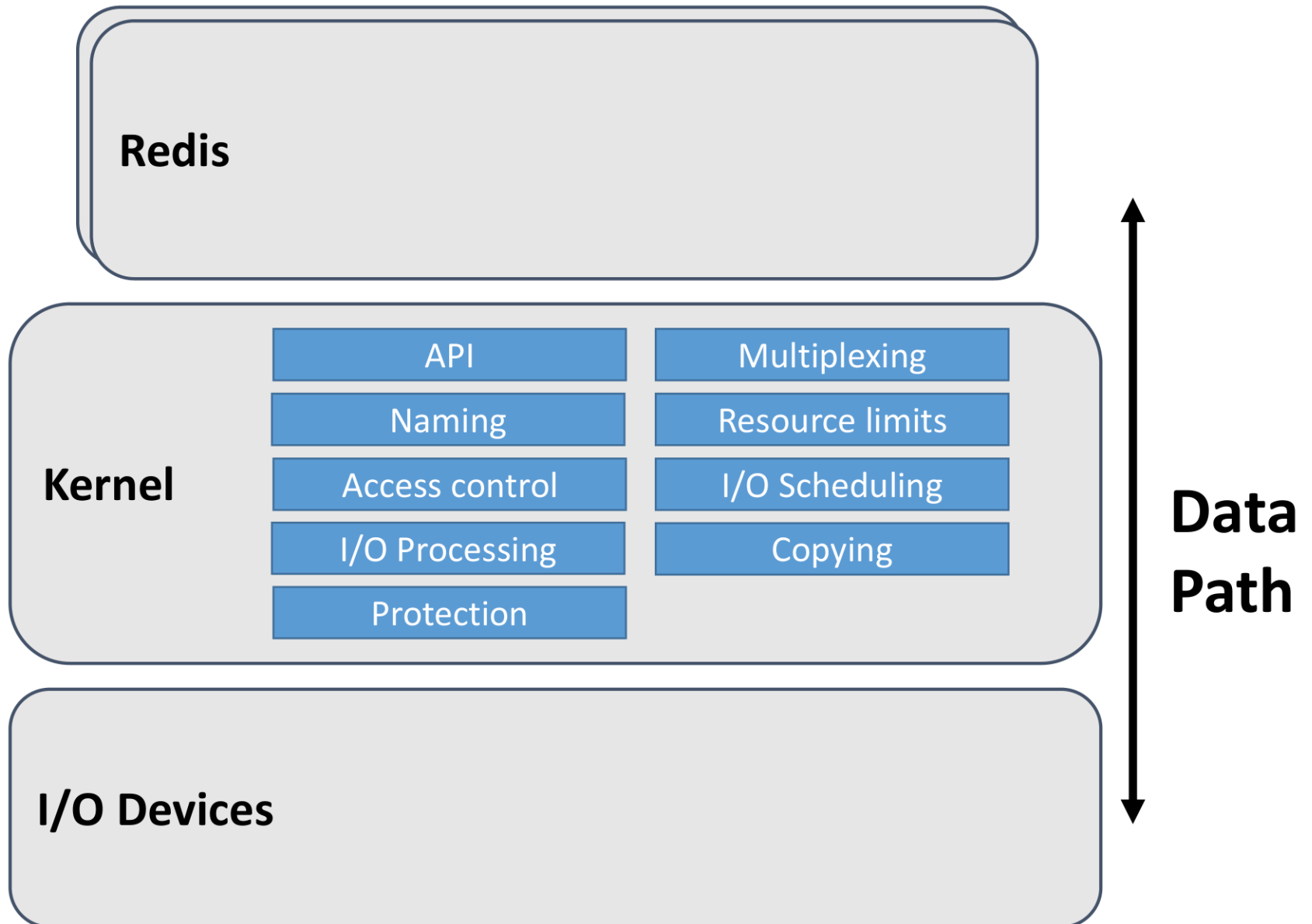
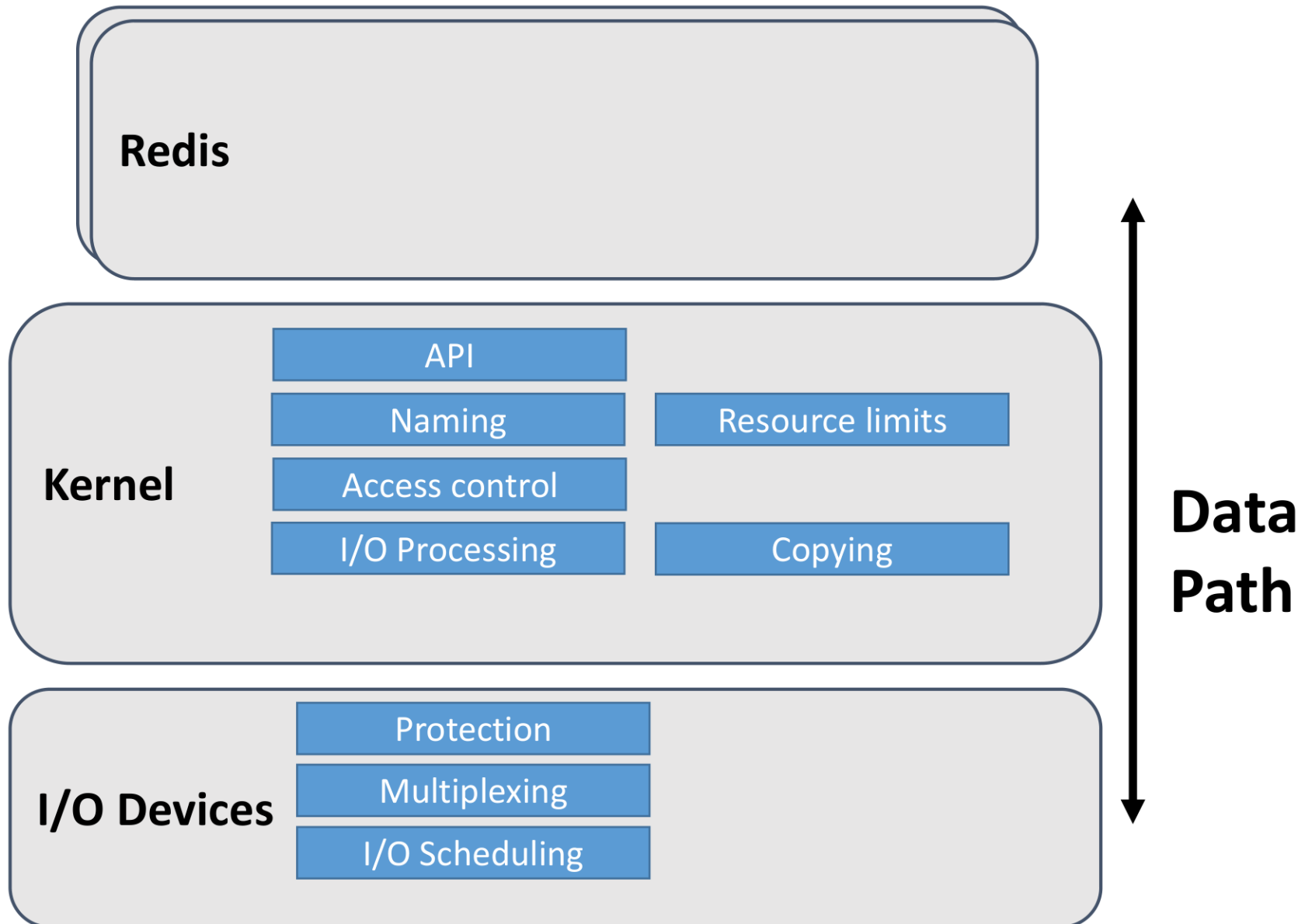


Fig. 4. Arrakis architecture. The storage controller maps VSAs to physical storage.

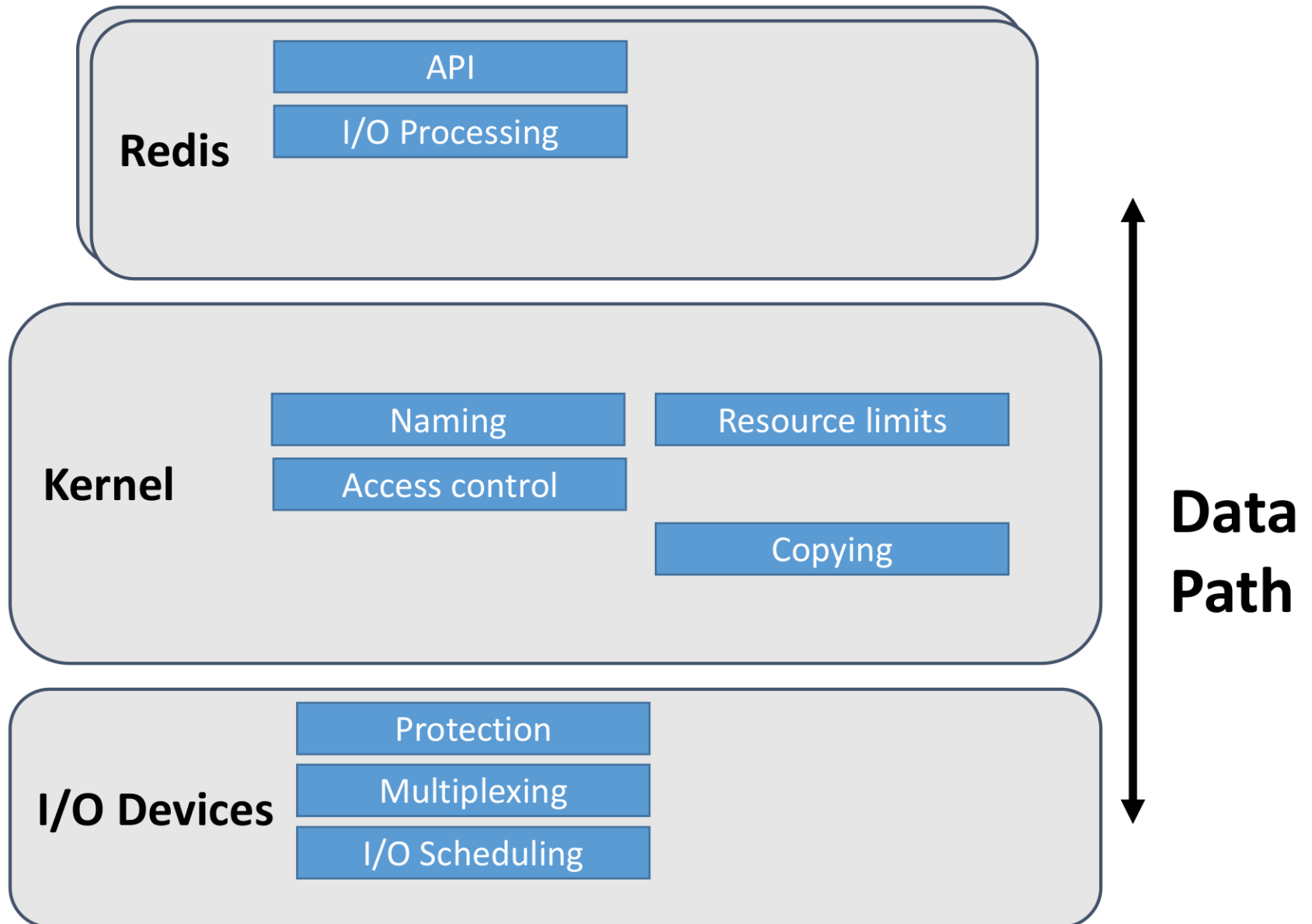
How to skip the kernel?



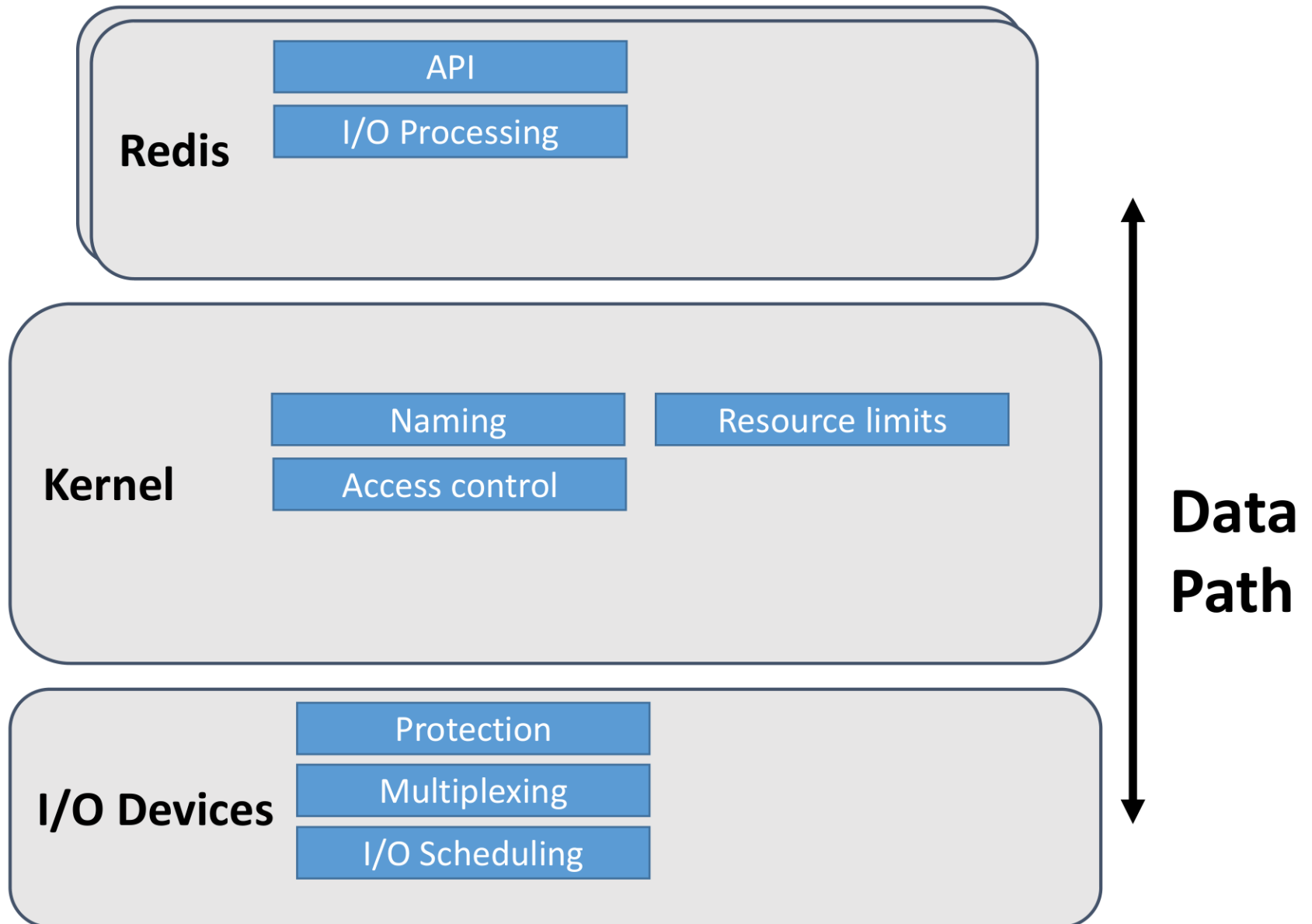
How to skip the kernel?



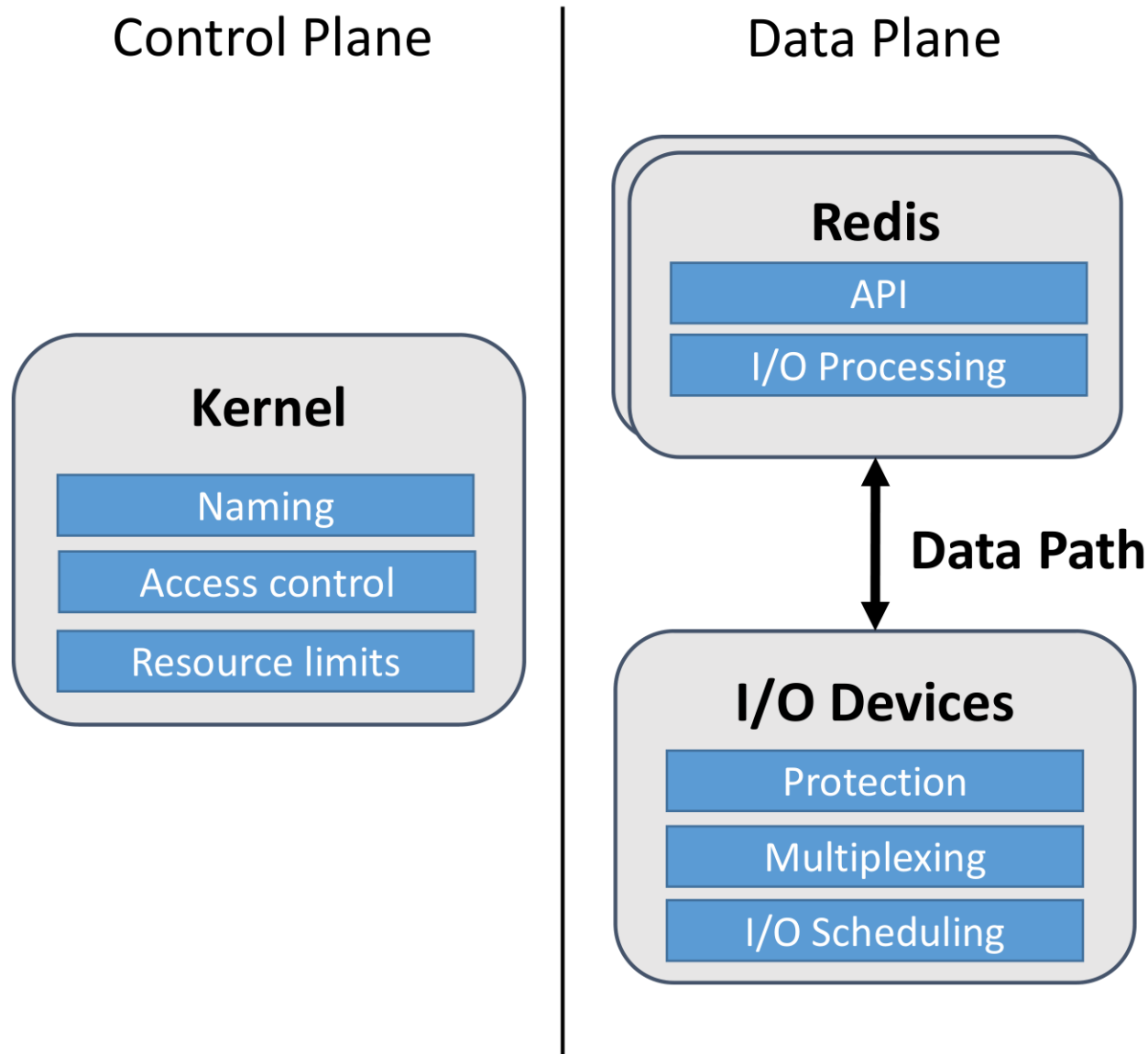
How to skip the kernel?



How to skip the kernel?



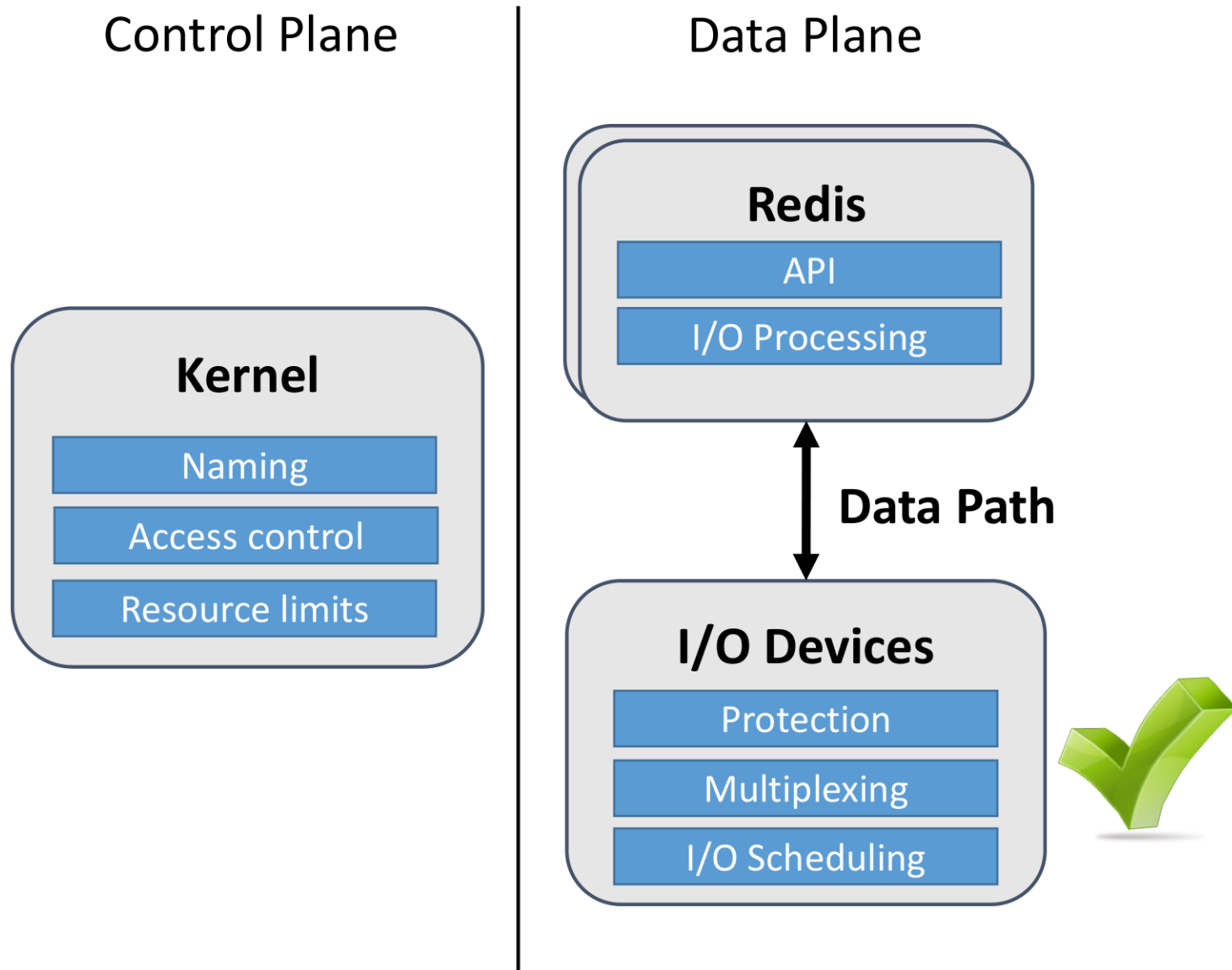
Arrakis I/O Architecture



Discussion

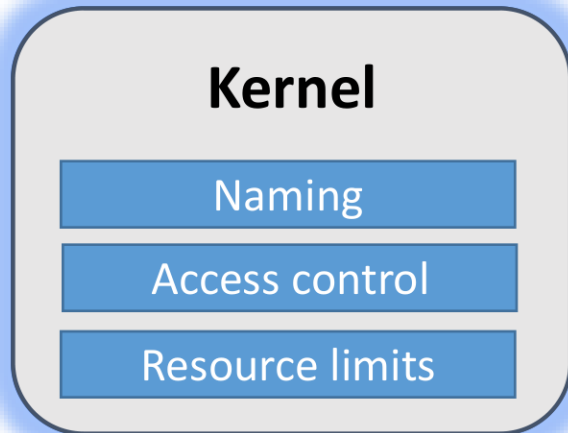
- **What's the potential challenge of such design?**

Arrakis I/O Architecture

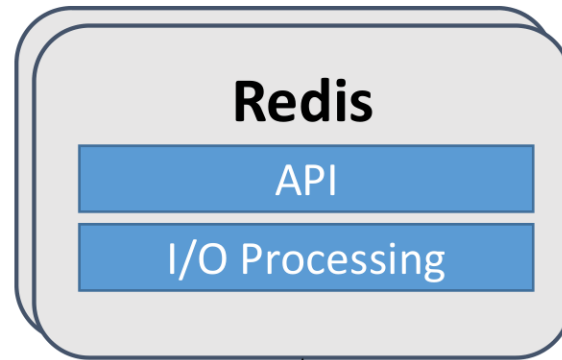


Arrakis I/O Architecture

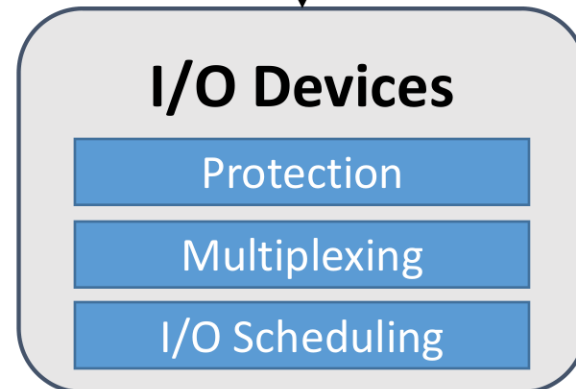
Control Plane



Data Plane



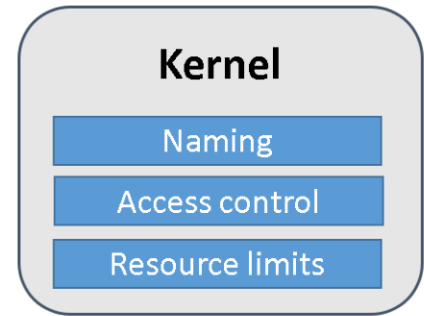
Data Path



Outline

1. Existing I/O Path without Kernel Involving
 1. SR-IOV
 2. RDMA
2. Arrakis' Main Idea
- 3. Arrakis' Components**
 - 1. Control Panel in Kernel**
 2. Network Interface Card Hardware Model (NIC and VNIC)
 3. Storage Hardware Model (VSA and VSIC)
4. Cross Application Files Read / Write
5. Designs of Persistent Data Structures
6. Evaluation

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

Component Configured by Kernel

Components

Functionality

Control Plane Operation

VNIC (Virtual Network Interface Card)

The unified abstraction for a **virtualized I/O device**.

Create / Delete / Associate with Doorbell

Doorbell

Notification mechanism

— allows the hardware to signal events directly to user-space queues.

Create / Delete / Associate with VNIC

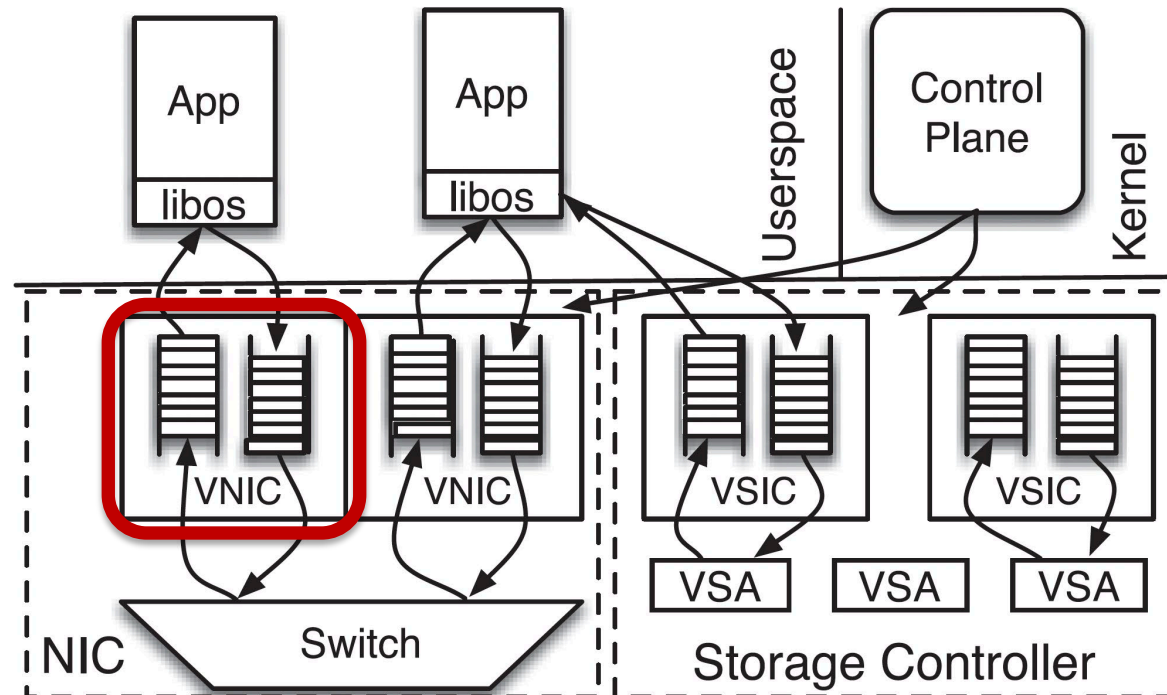


Fig. 4. Arrakis architecture. The storage controller maps VSAs to physical storage.

Component Configured by Kernel

Component

VSA (Virtual Storage Area)

Functionality

Logical storage partition
owned by a process.

Control Plane Operation

Create / Resize

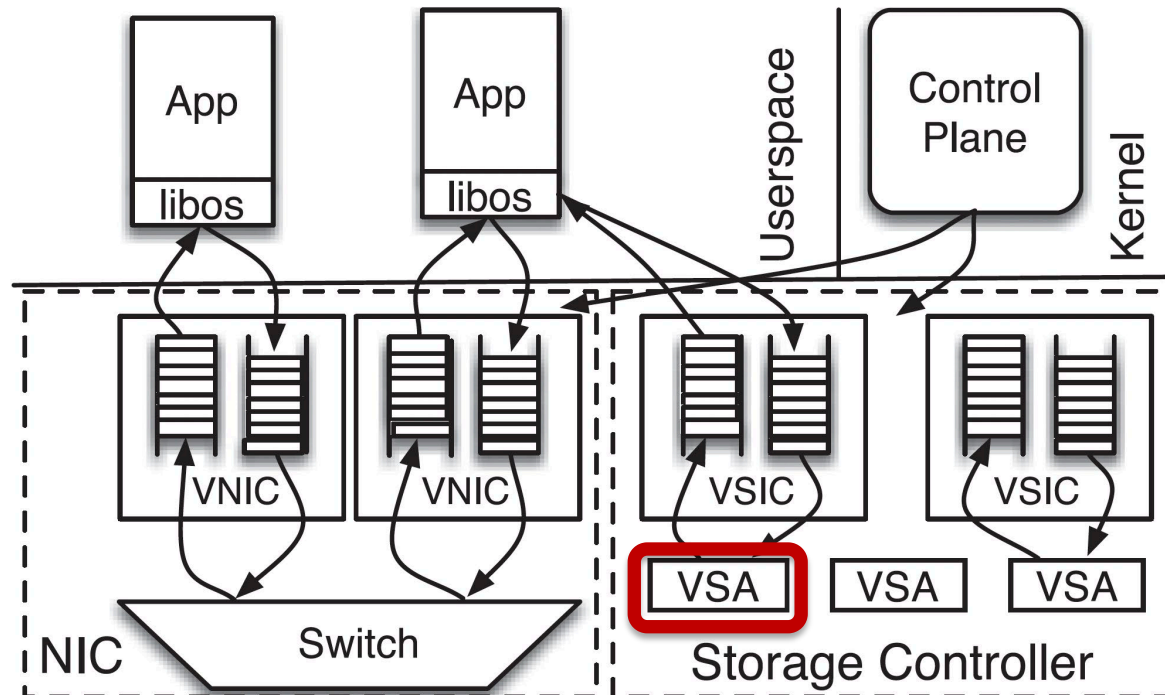


Fig. 4. Arrakis architecture. The storage controller maps VSAs to physical storage.

Component Configured by Kernel

Component	Functionality	Control Plane Operation
Filter	Predicate defining which packets (or requests) belong to which queue.	Create
Rate specifier	Bandwidth control object, specifying transmission or I/O rates.	Configure

Library Operating Systems (libos)

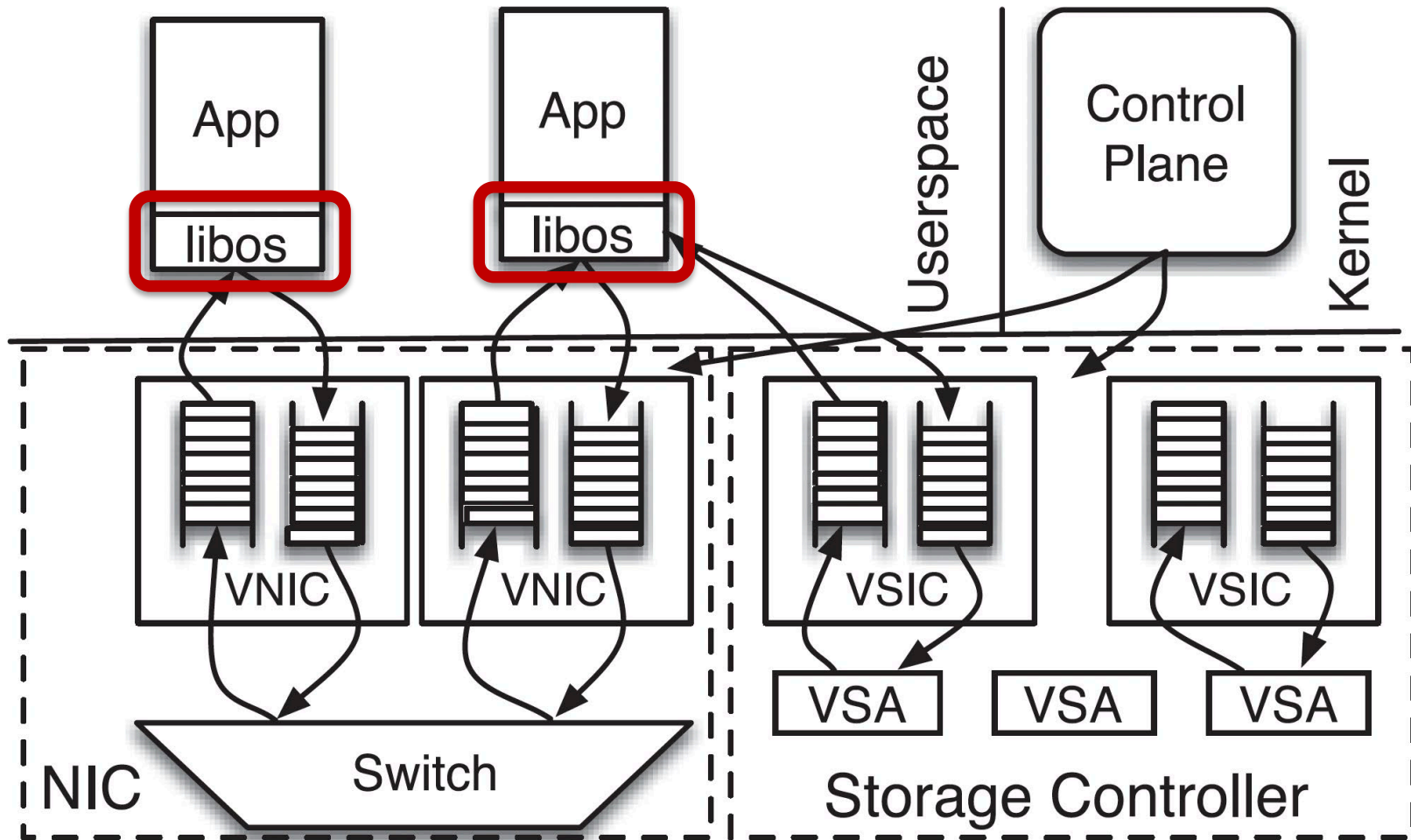
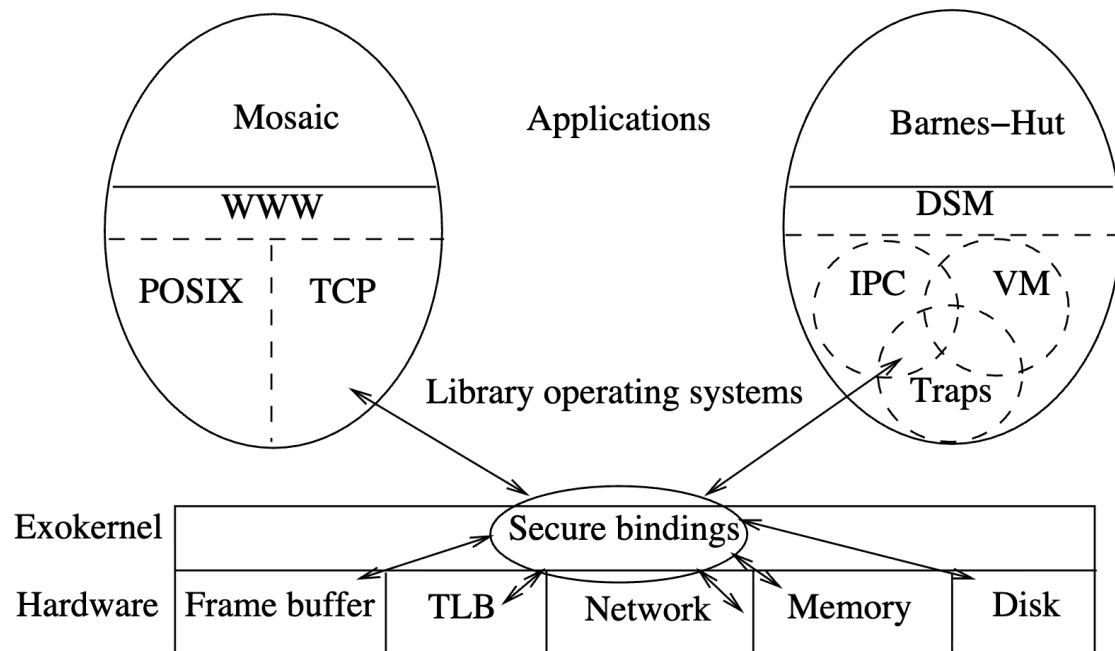


Fig. 4. Arrakis architecture. The storage controller maps VSAs to physical storage.

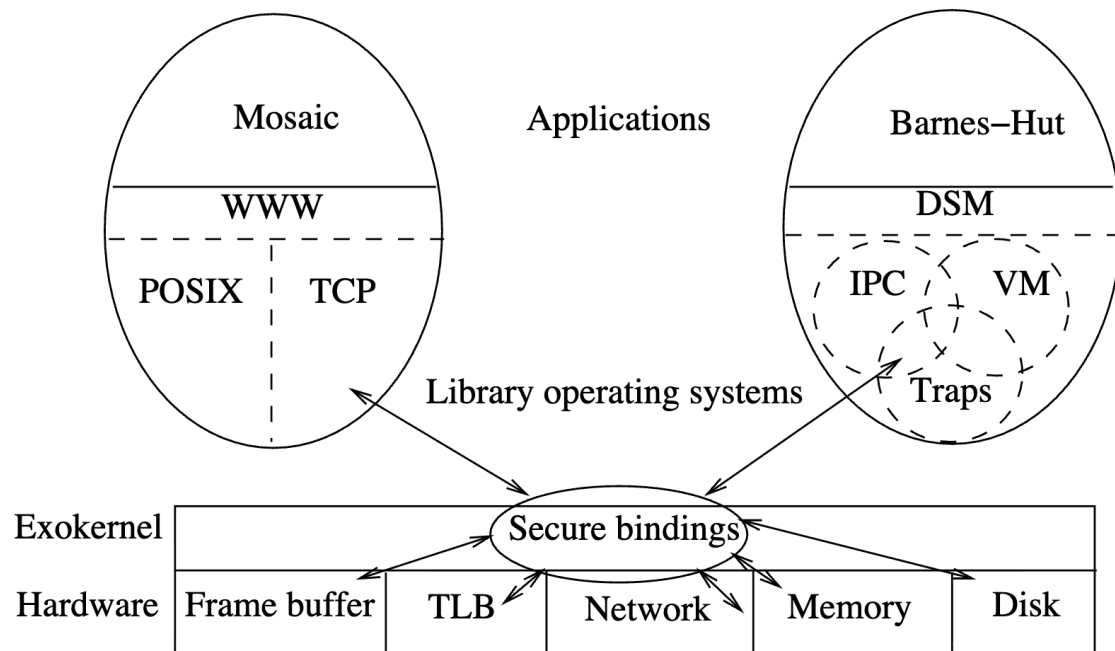
Library Operating Systems (libos)

1. Proposes by Exokernel
2. Purpose: Allowing user define **flexible hardware abstraction**, the **kernel only handle the hardware protection and multiplexing**, leaving all other stuff handled by libos in user space



Library Operating Systems (libos)

1. Advantages
 1. Flexibility
 2. Improved direct memory access (DMA) and interprocess communication by 40 times
2. Disadvantages
 1. **I/O Data path still involves kernels**



Outline

1. Existing I/O Path without Kernel Involving
 1. SR-IOV
 2. RDMA
2. Arrakis' Main Idea
- 3. Arrakis' Components**
 1. Control Panel in Kernel
 - 2. Network Interface Card Hardware Model (NIC and VNIC)**
 3. Storage Hardware Model (VSA and VSIC)
4. Cross Application Files Read / Write
5. Designs of Persistent Data Structures
6. Evaluation

Hardware Model – NIC and VNIC

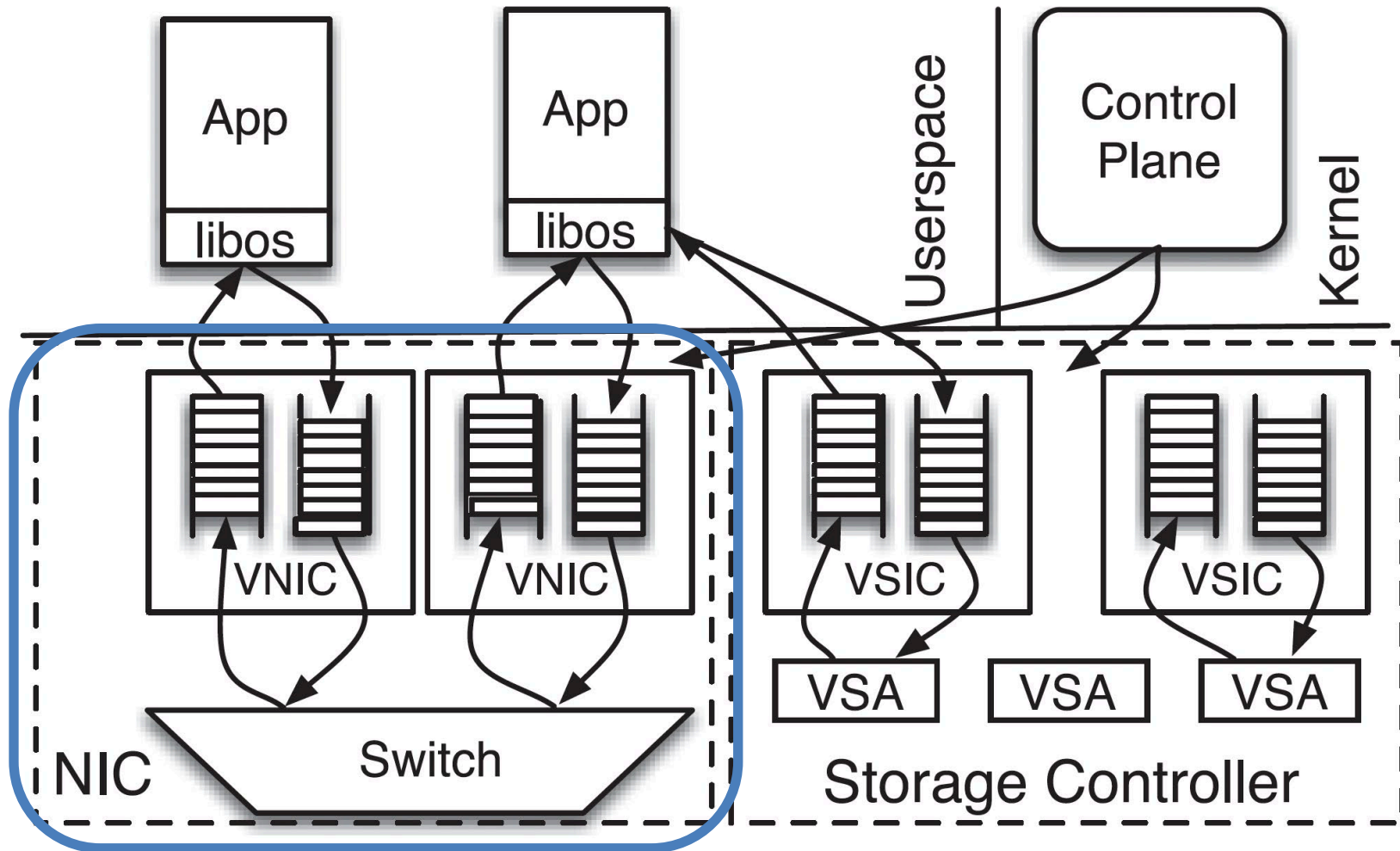


Fig. 4. Arrakis architecture. The storage controller maps VSAs to physical storage.

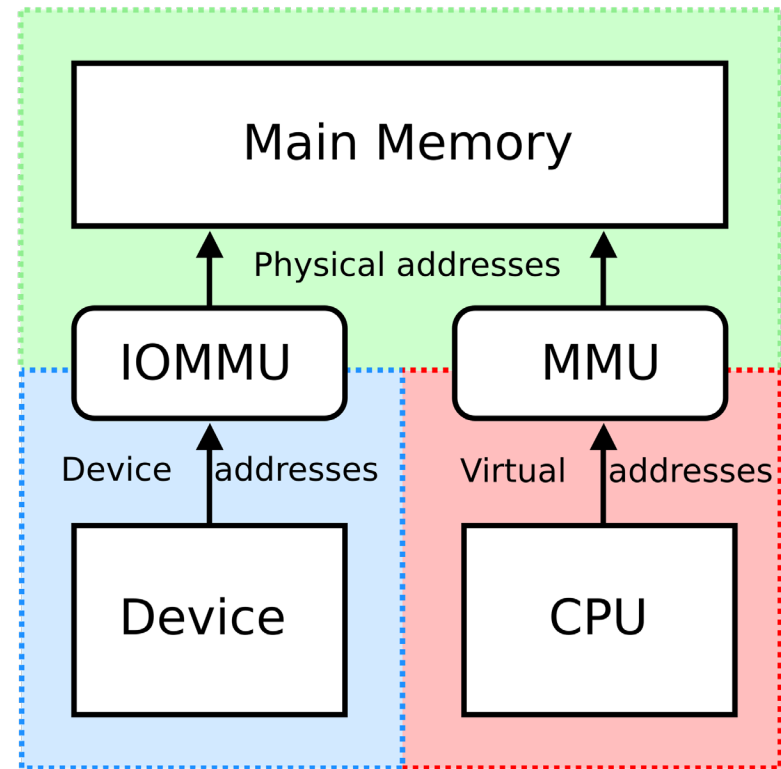
Handling Device I/O and Network Communication

Hardware Model – NIC and VNIC

Hardware Mechanism	Function	Purpose in Arrakis
Queues (Transmit/Receive)	Private queues of each virtual device for sending and receiving.	No shared kernel queues for isolation
Protection Domains	Each queue is bound via the IOMMU .	Prevents DMA into unauthorized memory regions.
Filter Tables	Packets and blocks router decide which queue goes for.	Direct demultiplexing without kernel.
Bandwidth Allocators	Hardware rate limiters	Provide per-app QoS

Input–Output Memory Management Unit (IOMMU)

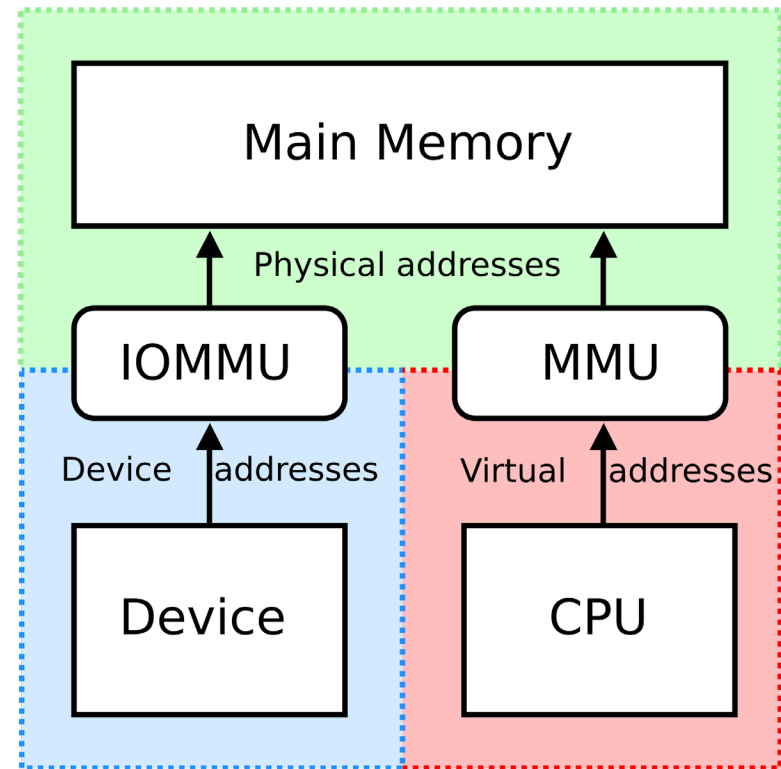
Purpose: Translate the device address to physical address for main memory



Input–Output Memory Management Unit (IOMMU)

Advantages

1. **Shorter** device address
2. Allowed **fragmented memory access**, enhancing the flexibility
3. Restrict the accessible memory region for device, **protecting from malicious device**



Outline

1. Existing I/O Path without Kernel Involving
 1. SR-IOV
 2. RDMA
2. Arrakis' Main Idea
- 3. Arrakis' Components**
 1. Control Panel in Kernel
 2. Network Interface Card Hardware Model (NIC and VNIC)
 - 3. Storage Hardware Model (VSA and VSIC)**
4. Cross Application Files Read / Write
5. Designs of Persistent Data Structures
6. Evaluation

Hardware Model – VSA and VSIC

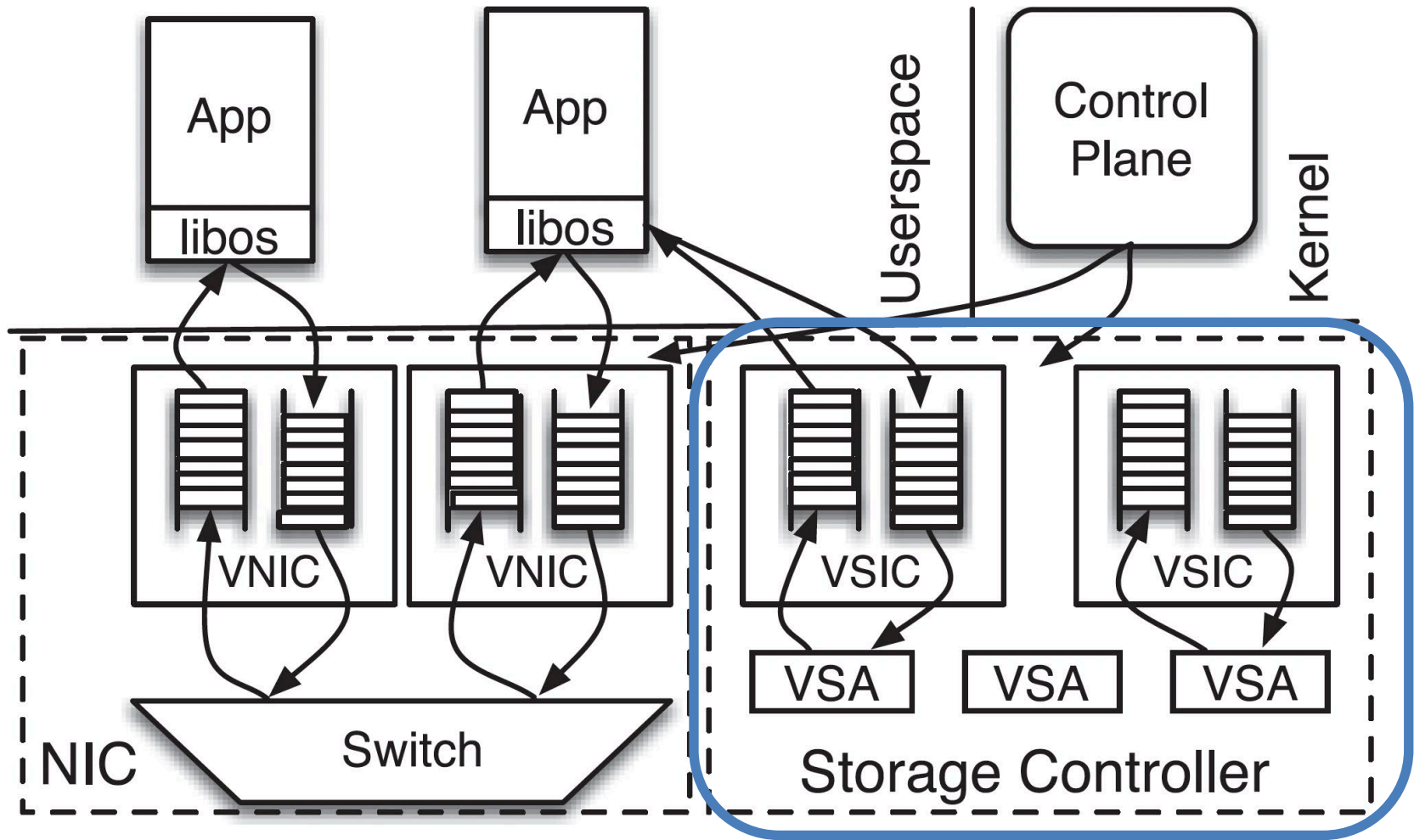


Fig. 4. Arrakis architecture. The storage controller maps VSAs to physical storage.

Handling Storage I/O

Hardware Model – VSA and VSIC

Motivation of Virtualizing Storage

By the time for the paper published, the storage devices (HDD, SSD, and RAID) don't support SR-IOV, which means they **did not support per-application virtualization**

1. **NO** Separate queues per process,
2. **NO** Per-queue rate limiters,
3. **NO** Direct mapping of user-space buffers for DMA.

Hardware Model – VSA and VSIC

(a) Application side

Each app owns:

- A **Request Descriptor Queue (RDQ)** → lists read/write requests.
- A **Response Descriptor Queue (RSQ)** → holds completions.
- Both are circular buffers (ring queues) in **shared memory**.

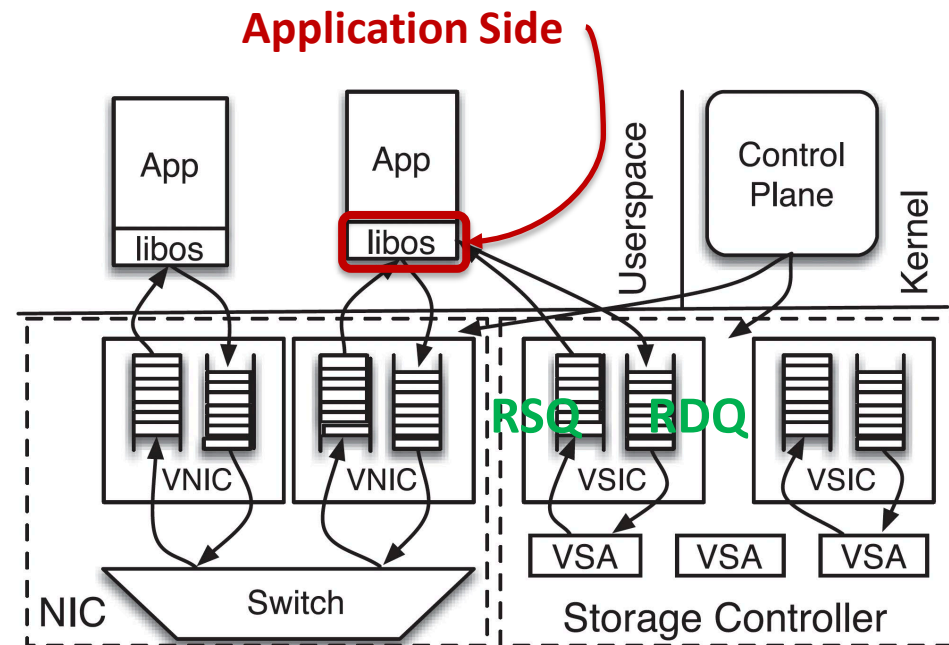


Fig. 4. Arrakis architecture. The storage controller maps VSAs to physical storage.

Hardware Model – VSA and VSIC

(b) VSIC Core (Emulation Thread)

- Reads requests from each app's **RDQ**.
- Checks access bounds to ensure requests are inside valid memory.
- Translates the virtual block address into a **physical disk address**.
- Submits the request to the physical storage driver / controller.
- Writes completion info back to the app's **RSQ** when done.

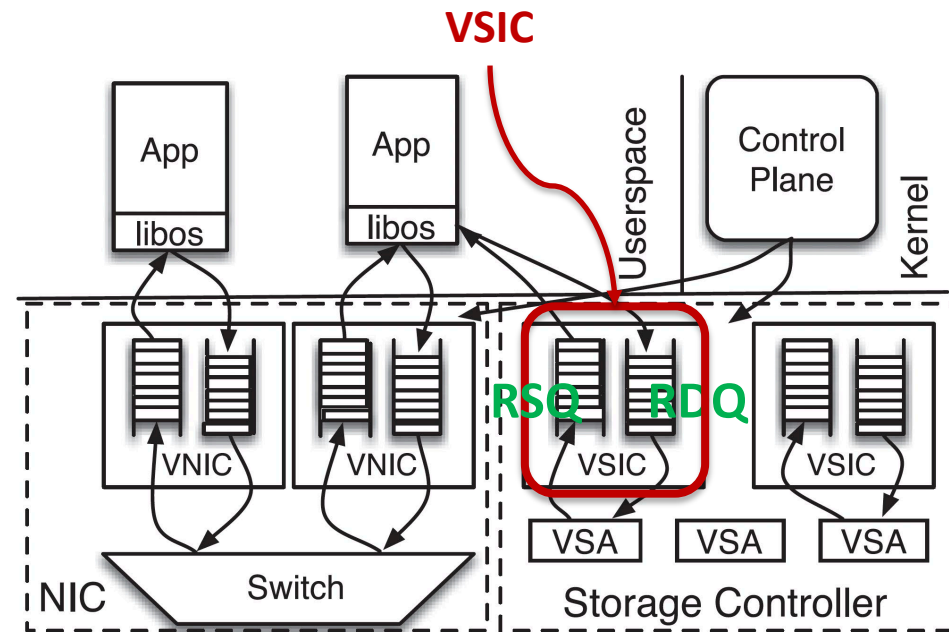


Fig. 4. Arrakis architecture. The storage controller maps VSAs to physical storage.

Hardware Model – VSA and VSIC

(b) Virtual Storage Area (VSA)

A **VSA** is defined as a contiguous range of blocks on a physical storage device. The control plane (kernel) assigns each VSA to an application.

- Execute physical Read/Write via DMA
- Submit the completion

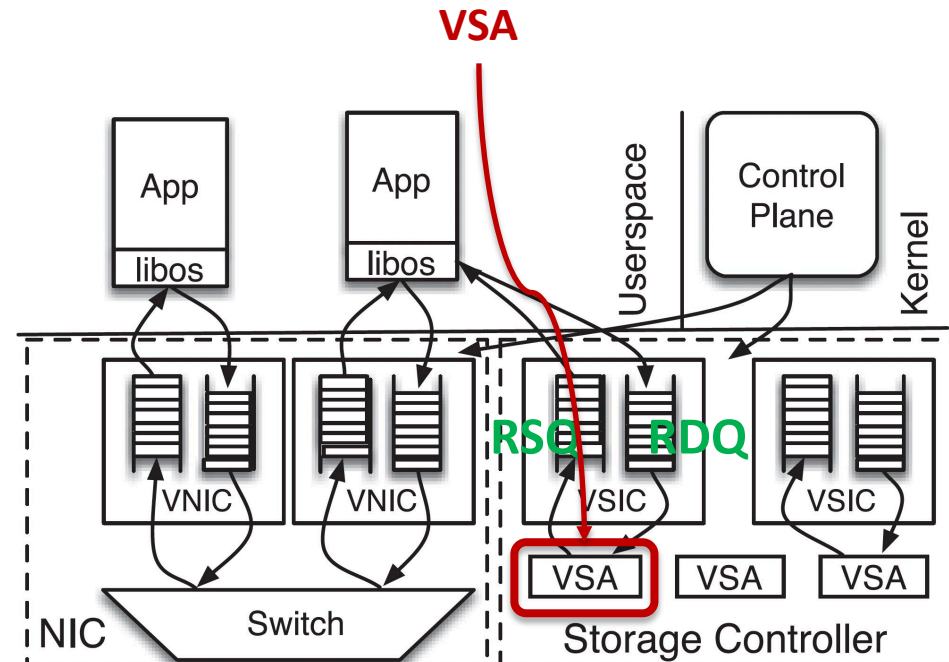


Fig. 4. Arrakis architecture. The storage controller maps VSAs to physical storage.

Hardware Model – Compare Arrakis & SR-IOV

Aspect	SR-IOV	Arrakis Hardware Model
Abstraction Unit	Virtual Function (VF) per application	Virtual I/O Controller (VIC) per application
Hardware Mechanism	PCIe SR-IOV (PF/VF)	Generalized queue-level virtualization
Protection	Provided by IOMMU	IOMMU, configured by Arrakis control plane
Data Path	Still passes through OS network stack	Direct user-level queue access
Control Plane Role	Hypervisor configures VFs	OS kernel configures filters, queues, and rate limits
Scale	Tens to hundreds of VFs	Thousands of queues (fine-grained)

Discussion

- **What's the drawback of such design?**
- **What if an App want to open a file belonging to another App?**

Outline

1. Existing I/O Path without Kernel Involving
 1. SR-IOV
 2. RDMA
2. Arrakis' Main Idea
3. Arrakis' Components
 1. Control Panel in Kernel
 2. Network Interface Card Hardware Model (NIC and VNIC)
 3. Storage Hardware Model (VSA and VSIC)
- 4. Cross Application Files Read / Write**
5. Designs of Persistent Data Structures
6. Evaluation

What if I want to access the files belonging to another App?

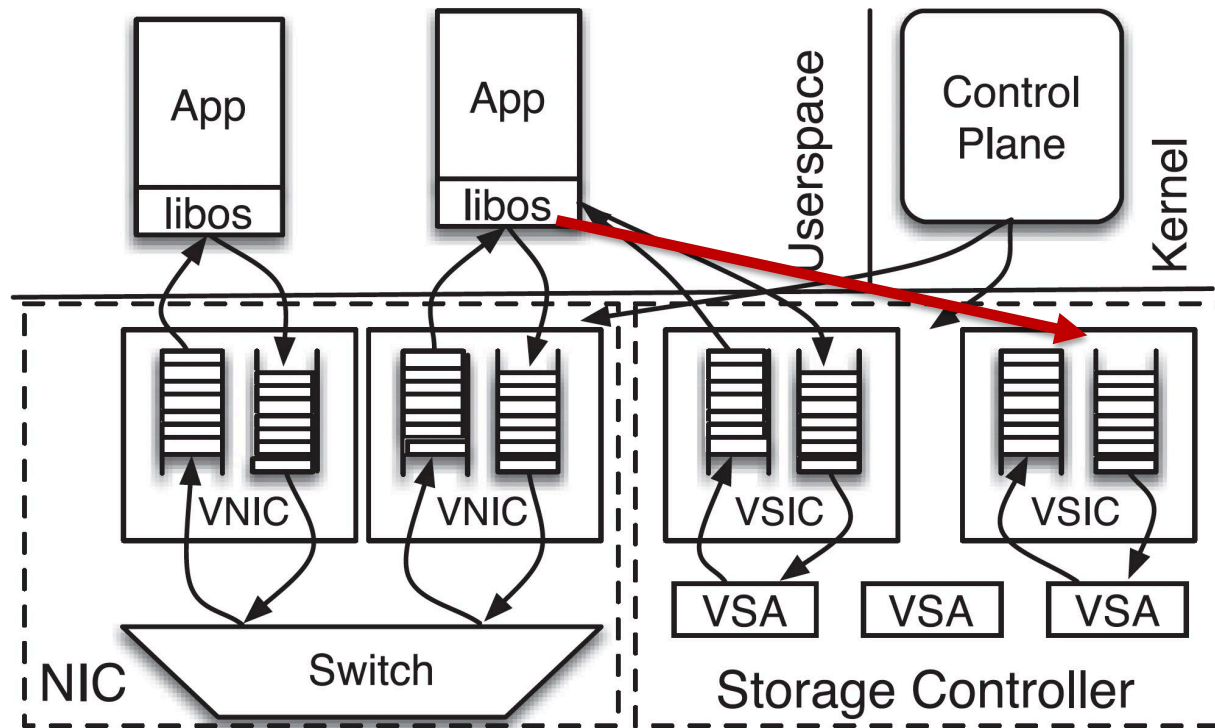


Fig. 4. Arrakis architecture. The storage controller maps VSAs to physical storage.

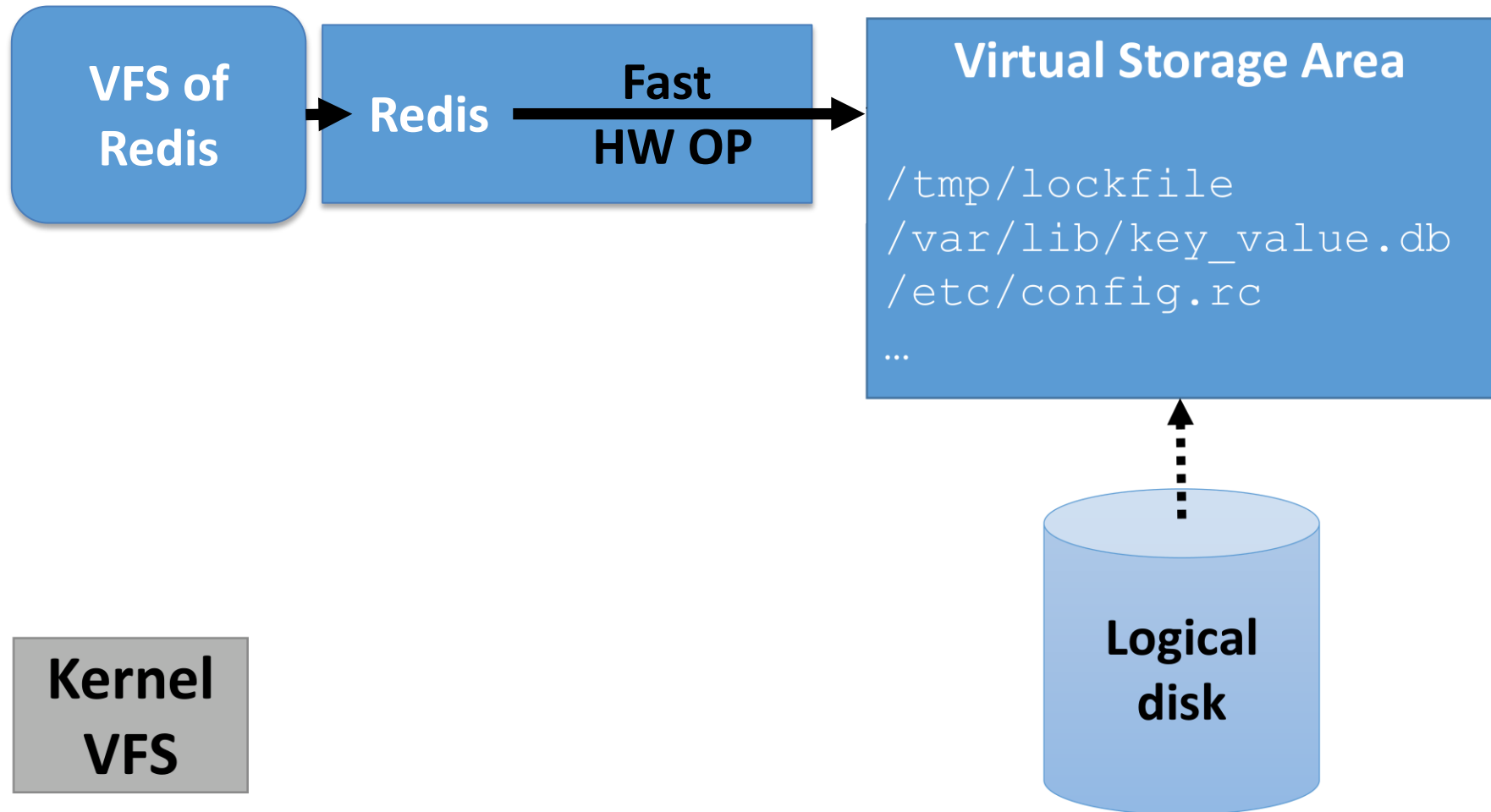
3 Ways to Access Files Belonging to Another App

- 1. Default: User-Level File Server RPC:** Every App runs a virtual file server (VFS) in user space. Kernel VFS redirects the file request to corresponding File server.
- 2. Shared VSA Mapping:** Multiple Apps share the same VSA.
- 3. Export as a Kernel File:** The App can export its own file to kernel VFS.

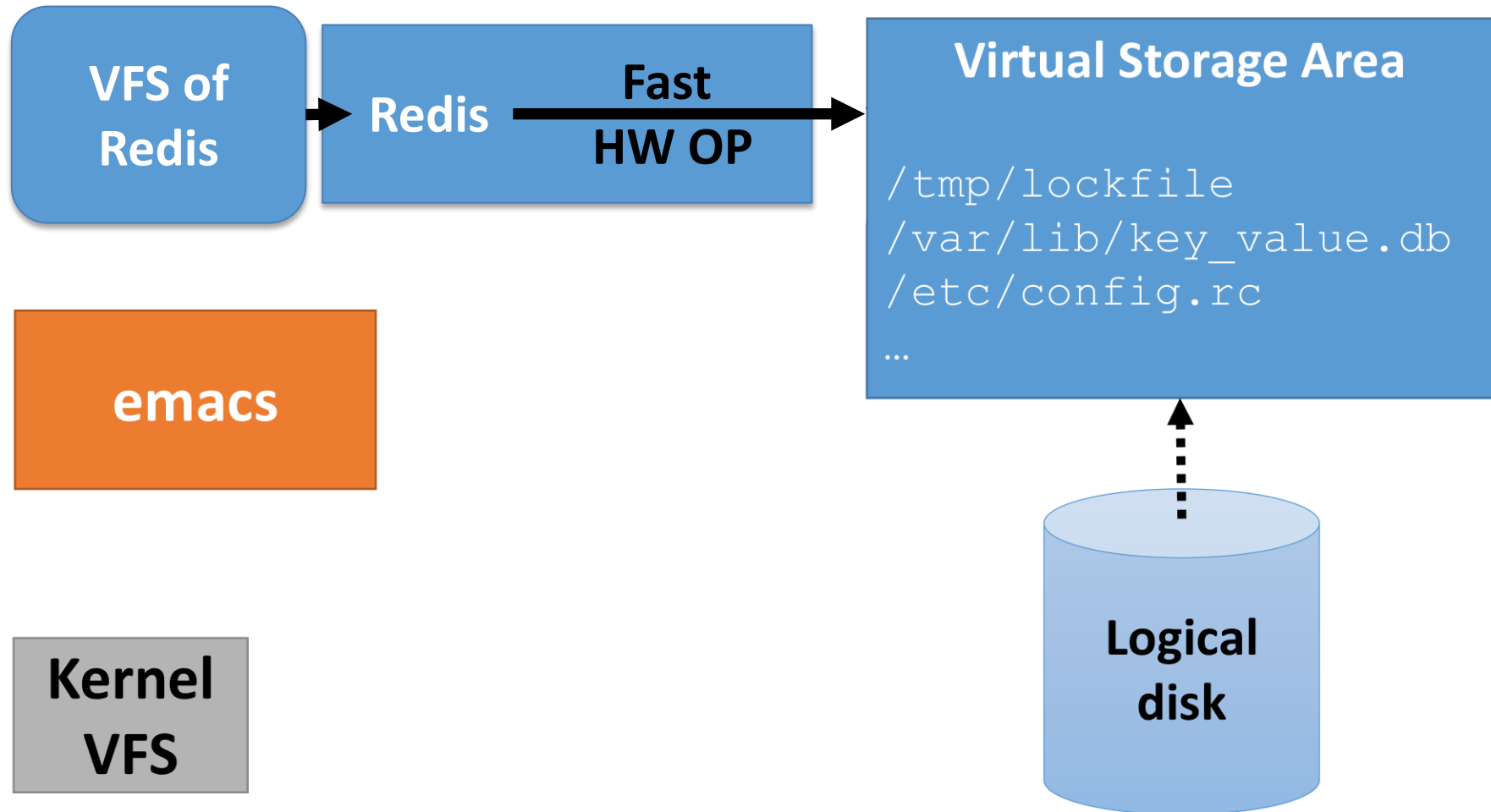
3 Ways to Access Files Belonging to Another App

1. **Default: User-Level File Server RPC:** Every App runs a virtual file server (VFS) in user space. Kernel VFS redirects the file request to corresponding File server.
2. **Shared VSA Mapping:** Multiple Apps share the same VSA.
3. **Export as a Kernel File:** The App can export its own file to kernel VFS.

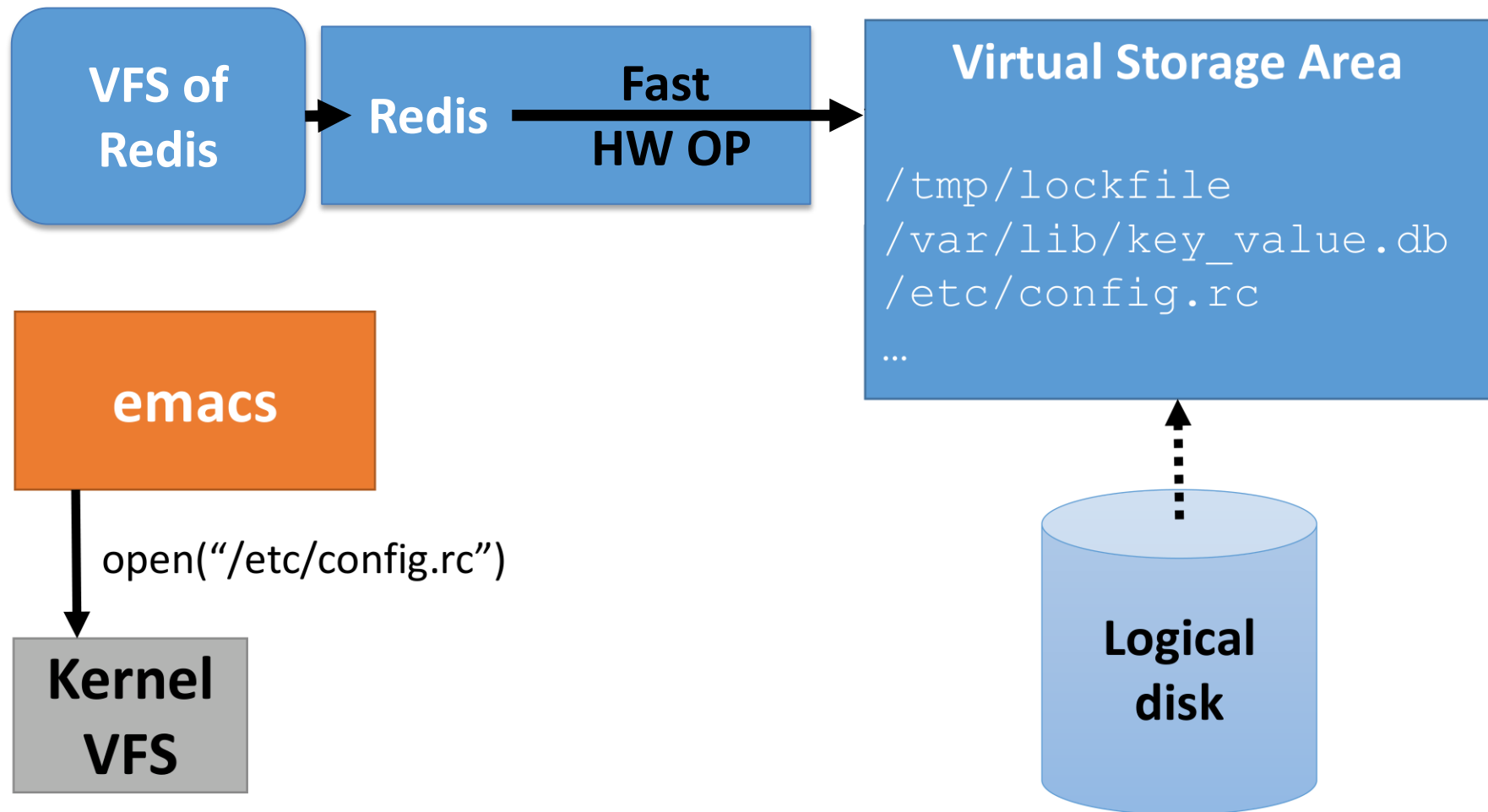
User-Level File Server RPC



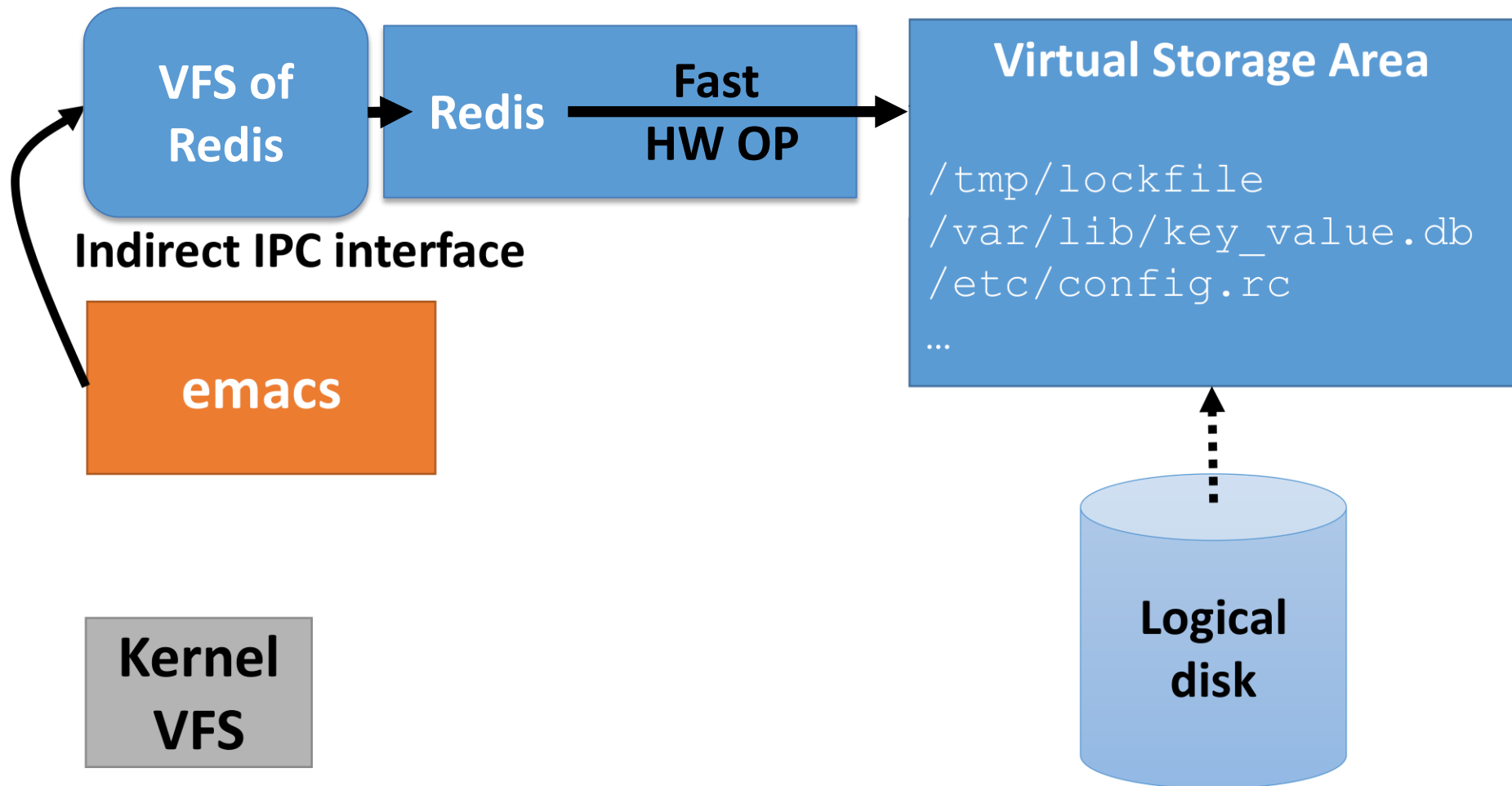
User-Level File Server RPC



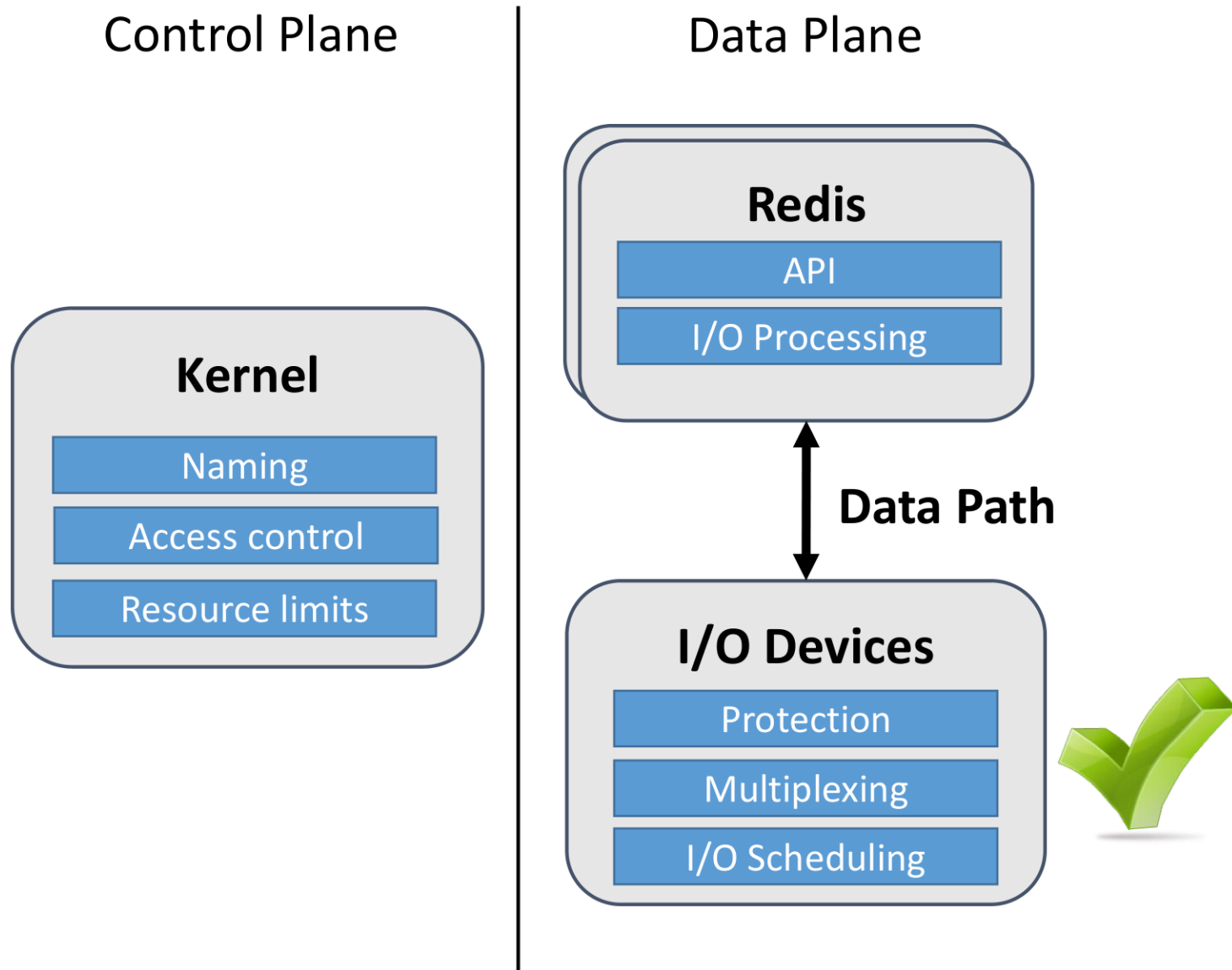
User-Level File Server RPC



User-Level File Server RPC



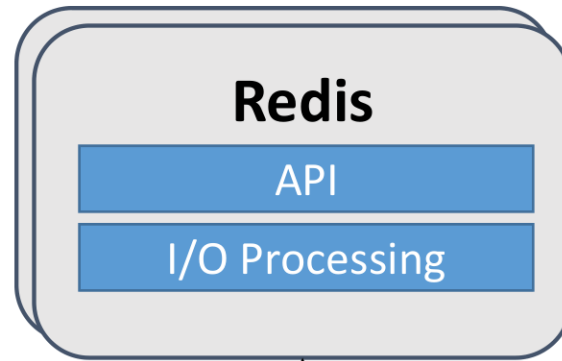
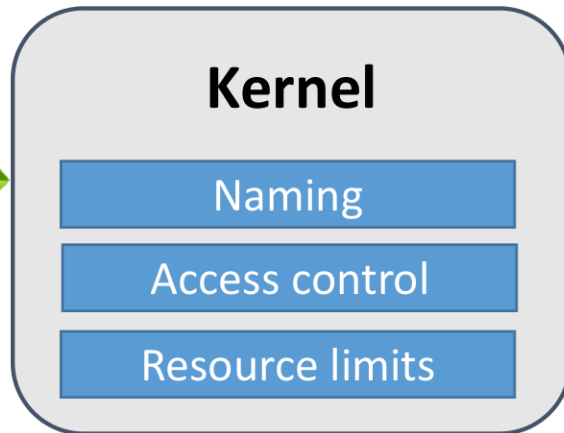
Arrakis I/O Architecture



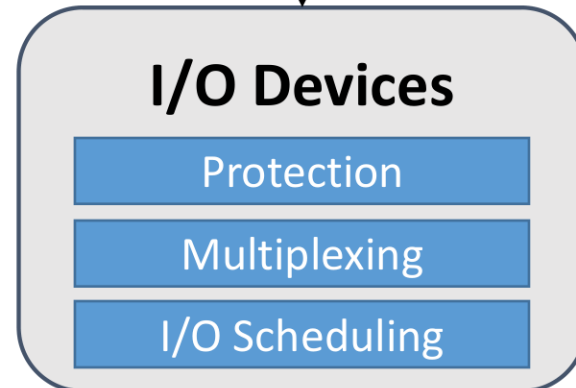
Arrakis I/O Architecture

Control Plane

Data Plane



Data Path



Arrakis I/O Architecture

Control Plane

Data Plane

Kernel

Naming

Access control

Resource limits

Redis

API

I/O Processing

Data Path

I/O Devices

Protection

Multiplexing

I/O Scheduling



Outline

1. Existing I/O Path without Kernel Involving
 1. SR-IOV
 2. RDMA
2. Arrakis' Main Idea
3. Arrakis' Components
 1. Control Panel in Kernel
 2. Network Interface Card Hardware Model (NIC and VNIC)
 3. Storage Hardware Model (VSA and VSIC)
4. Cross Application Files Read / Write
- 5. Designs of Persistent Data Structures**
6. Evaluation

Designs of Persistent Data Structures

Developed a library Caladan, Design goals

1. Operations are **immediately persistent**
2. The **structure is robust versus crash failures**
3. Operations have **minimal latency**

Designs of Persistent Data Structures

Use Log data structure to store all data with “inode”. Each Log entry contains

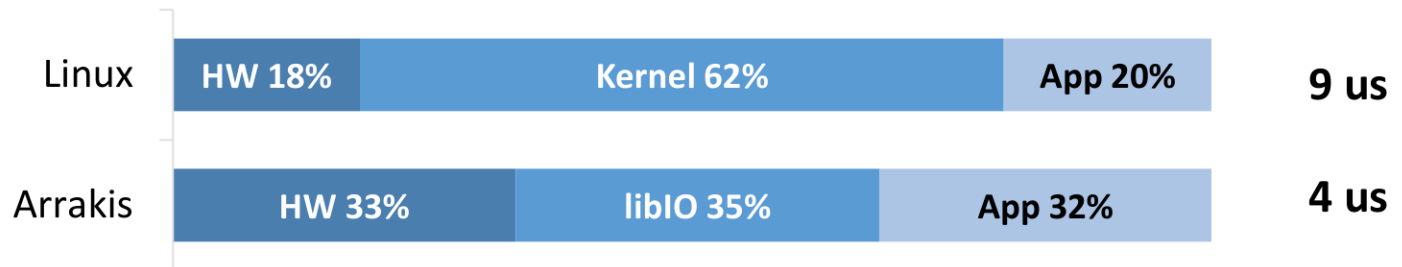
- 1. Next entry pointer**
- 2. Entry start pointer**
- 3. Entry end pointer**

Outline

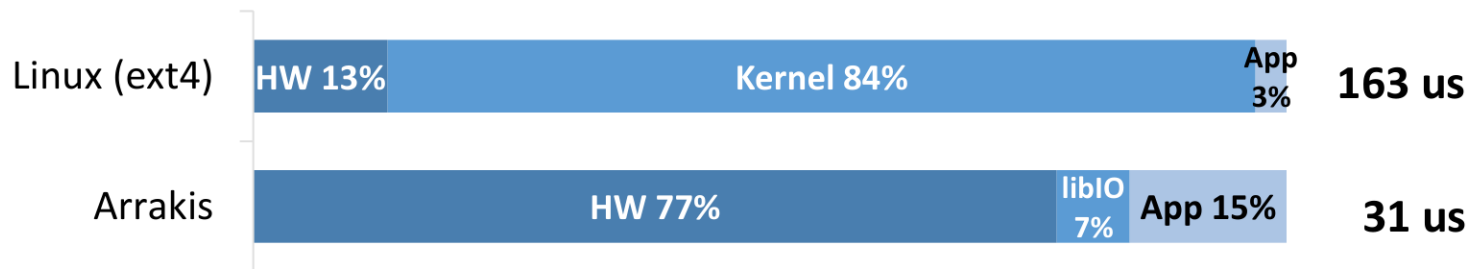
1. Existing I/O Path without Kernel Involving
 1. SR-IOV
 2. RDMA
2. Arrakis' Main Idea
3. Arrakis' Components
 1. Control Panel in Kernel
 2. Network Interface Card Hardware Model (NIC and VNIC)
 3. Storage Hardware Model (VSA and VSIC)
4. Cross Application Files Read / Write
5. Designs of Persistent Data Structures
- 6. Evaluation**

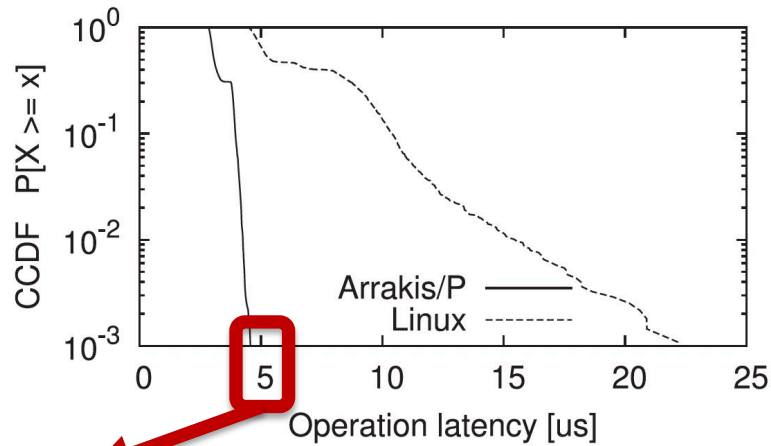
Redis Latency

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

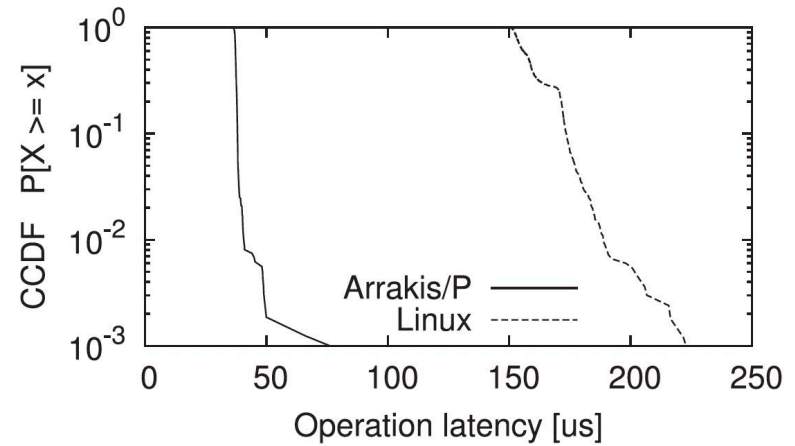


- Reduced (persistent) SET latency by **81%**





(a) Read hit



(b) Durable write

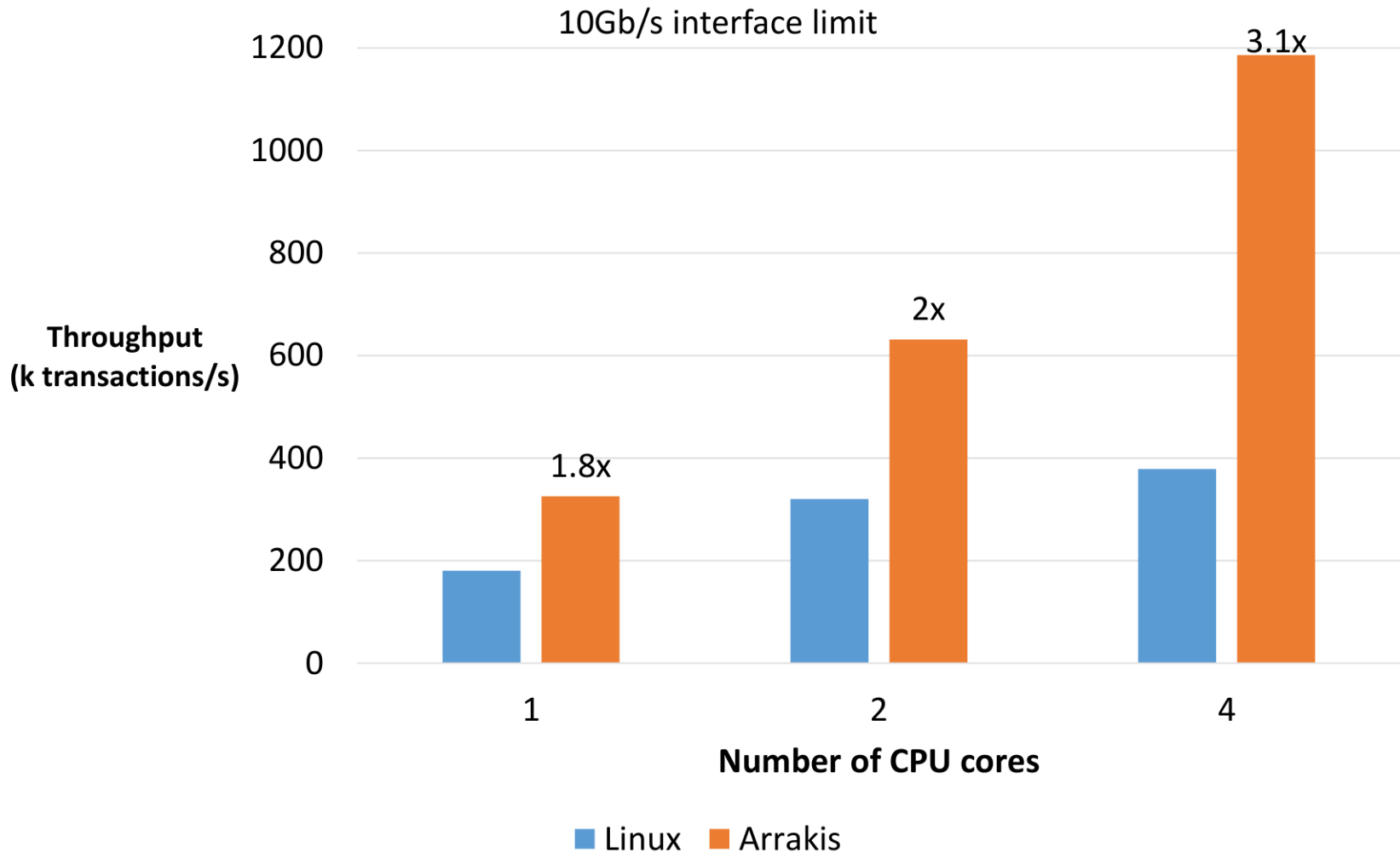
Fig. 3. Latency distribution of Redis read and write operations.

Complementary Cumulative Distribution Function (CCDF)
means $P(X \geq \mu)$ with given threshold μ

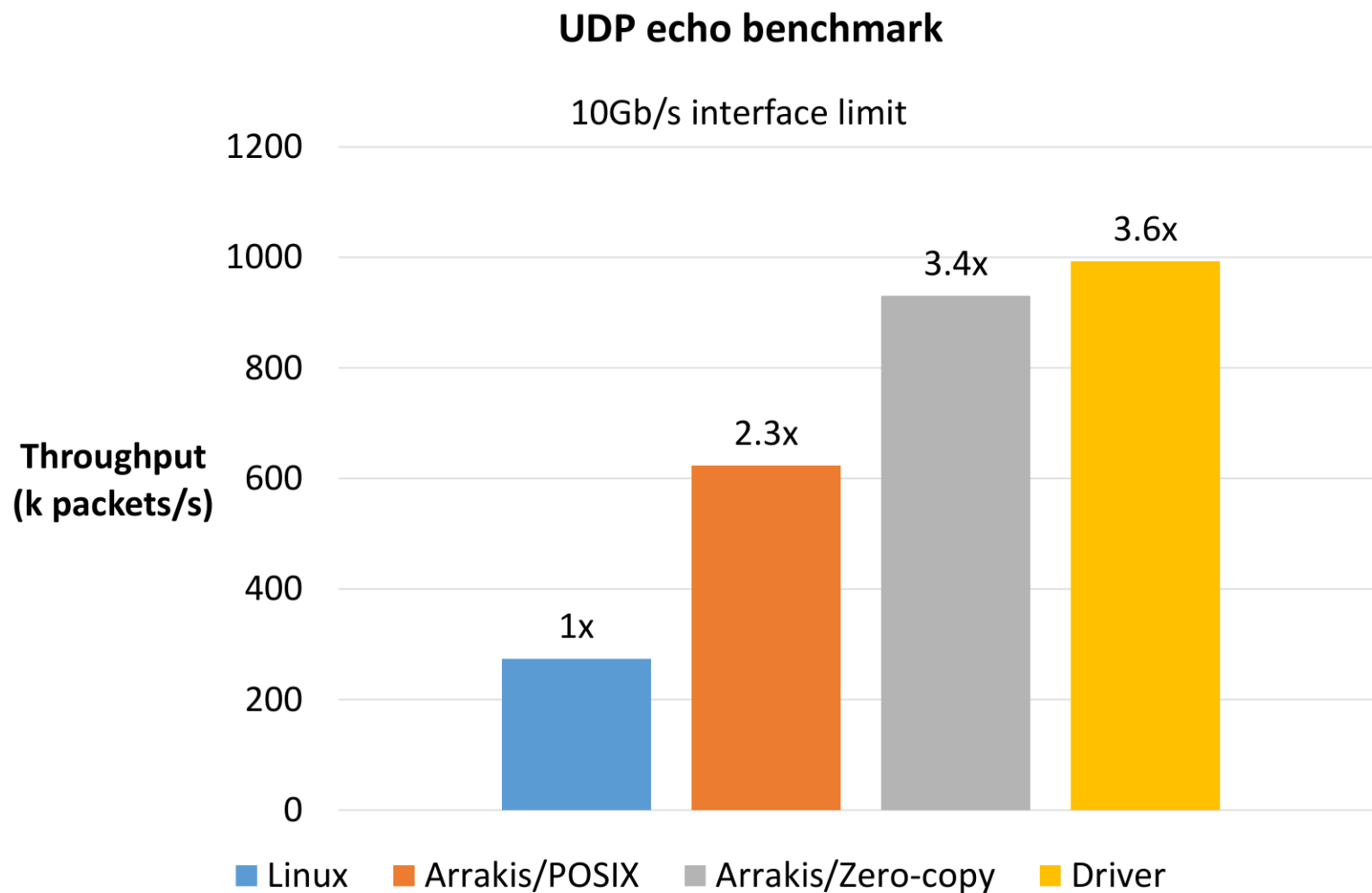
Redis Throughput

- 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



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

Source code: [**http://arrakis.cs.washington.edu**](http://arrakis.cs.washington.edu)

Related Works

1. SIGCOMM'24 - [Understanding the Host Network](#)
2. USENIX'24 - [High-throughput and Flexible Host Networking for Accelerated Computing](#)
3. SIGOPS'95 - [U-Net: A User-Level Network Interface for Parallel and Distributed Computing](#)
4. HPCA'97 - [ATM and Fast Ethernet Network Interfaces for User-level Communication](#)
5. SIGOPS'95 - [Exokernel: An Operating System Architecture for Application-Level Resource Management](#)
6. Matt Welsh's MS Thesis - [A System Supporting High-Performance Communication and I/O in Java](#)

OpenFlow: Enabling Innovation in Campus Networks

Nick McKeown, Guru Parulkar
Stanford University

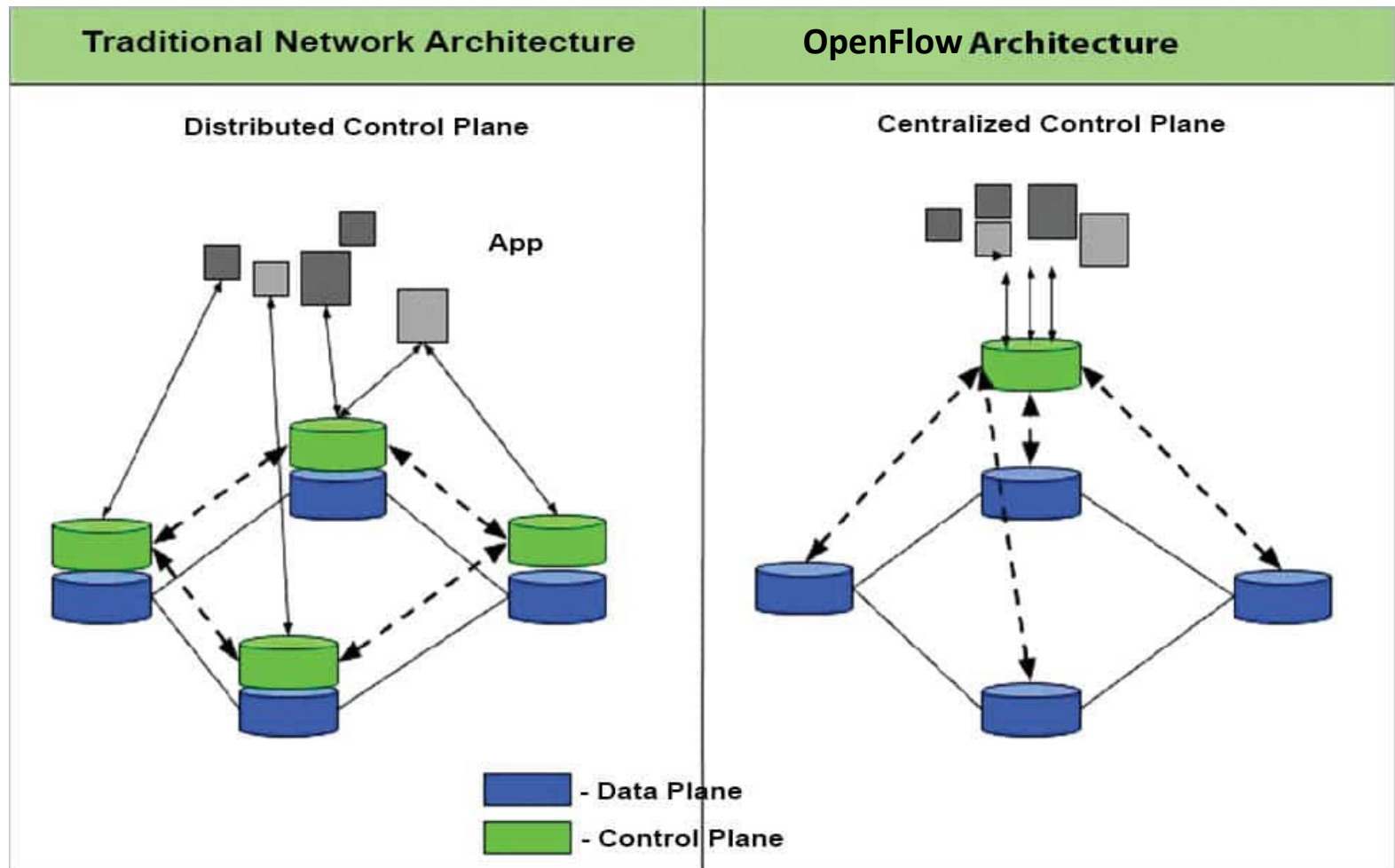
Larry Peterson, Jennifer Rexford
Princeton University

Tom Anderson
University of Washington

Hari Balakrishnan
MIT

Scott Shenker
University of California, Berkeley

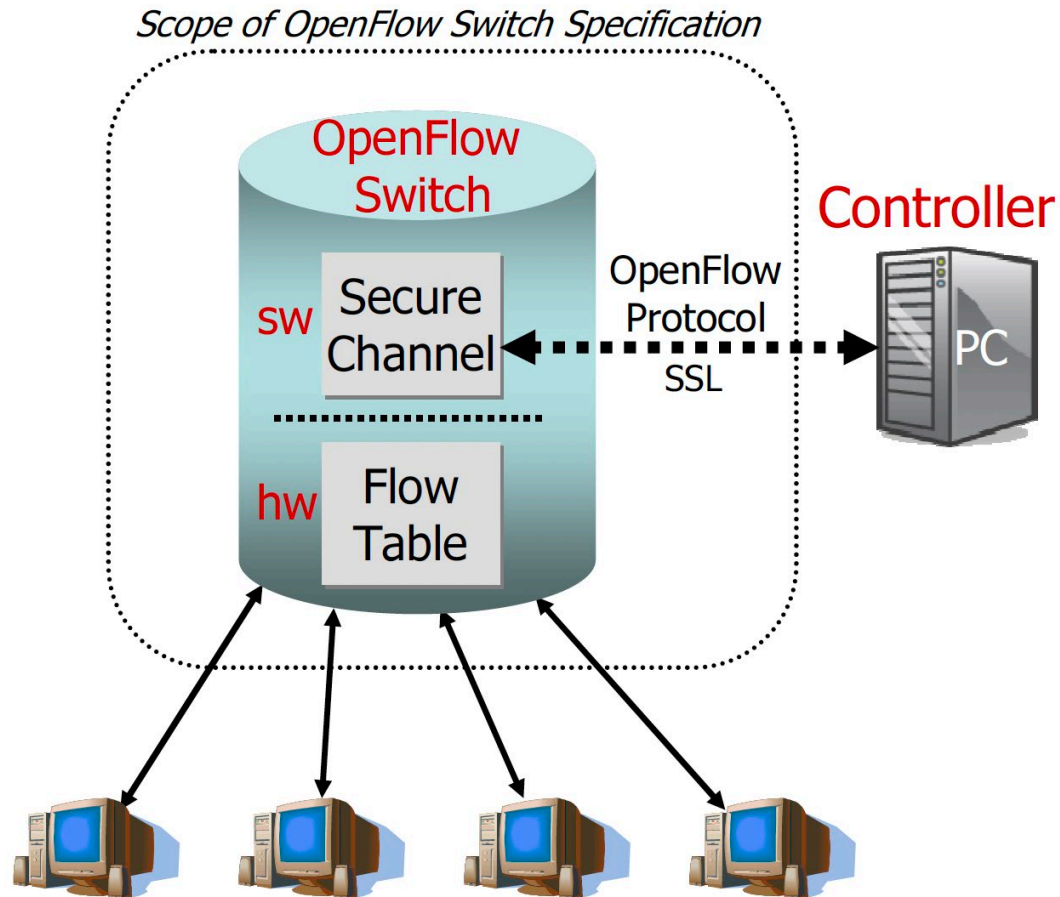
Jonathan Turner
Washington University in St. Louis



Hardware Defined
Network (HDN)

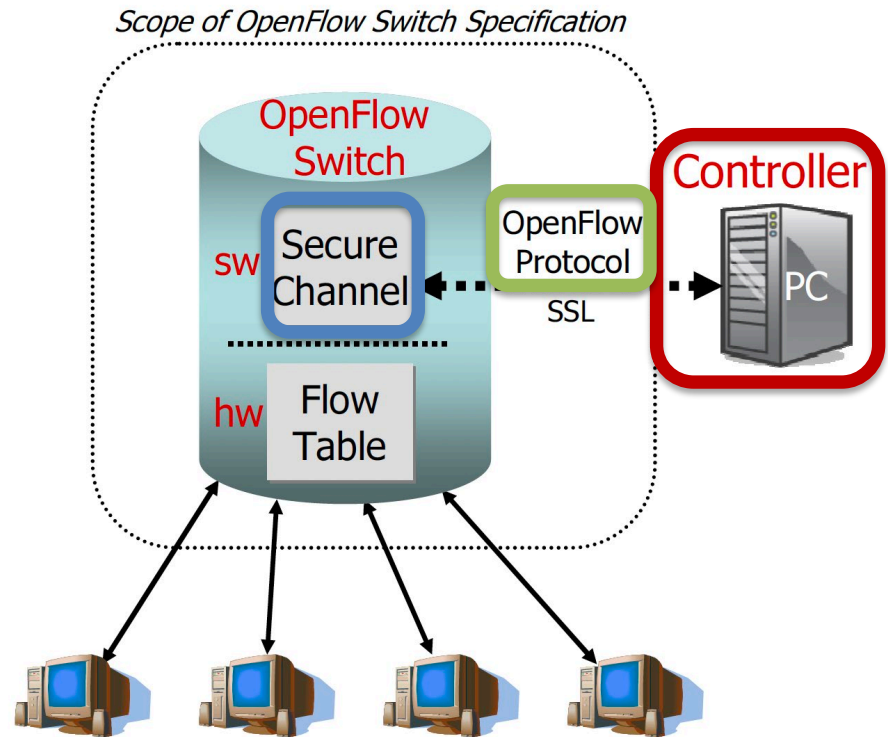
Software Defined
Network (SDN)

OpenFlow Architecture



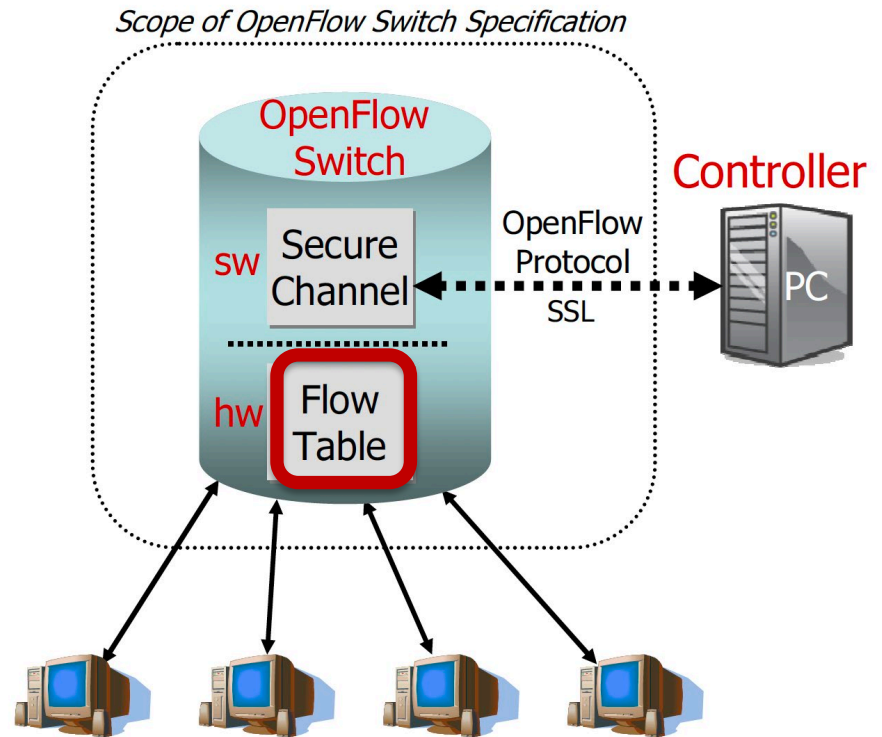
OpenFlow Architecture

- **Controller:** Configure the rules of flow table.
- **Secure Channel:** encrypted connection between switch and controller.
- **OpenFlow Protocol:** API for installing/removing rules of flow table.



OpenFlow Architecture

- **Flow Table:** Define the handling procedure for received packets, consisting of 3 components
 - Match
 - Action
 - Stats
- OpenFlow provide a general abstraction for flexible flow table



Summary

- Controller computes forwarding rules **dynamically**.
- Supports **new routing protocols without changing hardware**
- Enables **fast prototyping** with only programming