

# LeapIO: Efficient and Portable Virtual NVMe Storage on ARM SoCs

Huaicheng Li, Mingzhe Hao, Stanko Novakovic,  
Vaibhav Gogte, Sriram Govindan, Dan Ports, Irene Zhang,  
Ricardo Bianchini, Haryadi S. Gunawi, Anirudh Badam



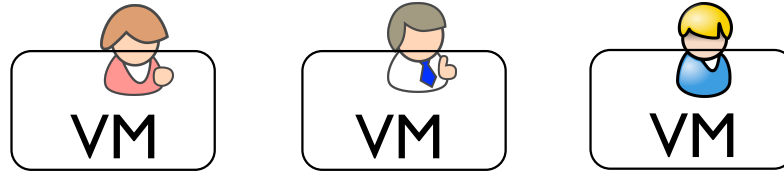
THE UNIVERSITY OF  
CHICAGO



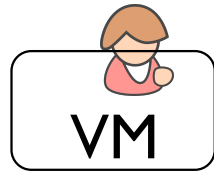
Microsoft



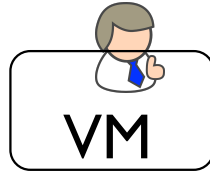
# Today's Cloud Storage Model



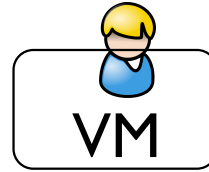
# Today's Cloud Storage Model



/dev/sda

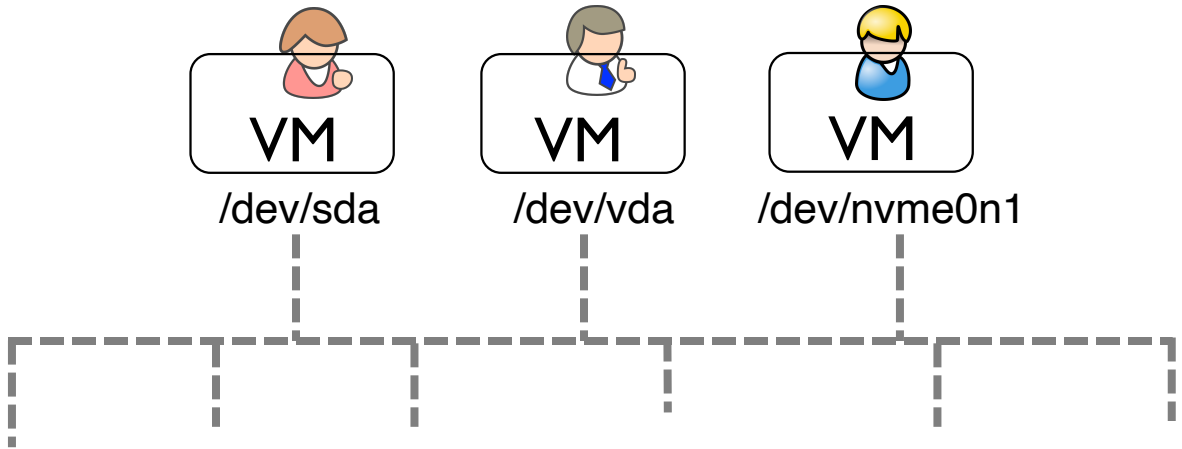


/dev/vda

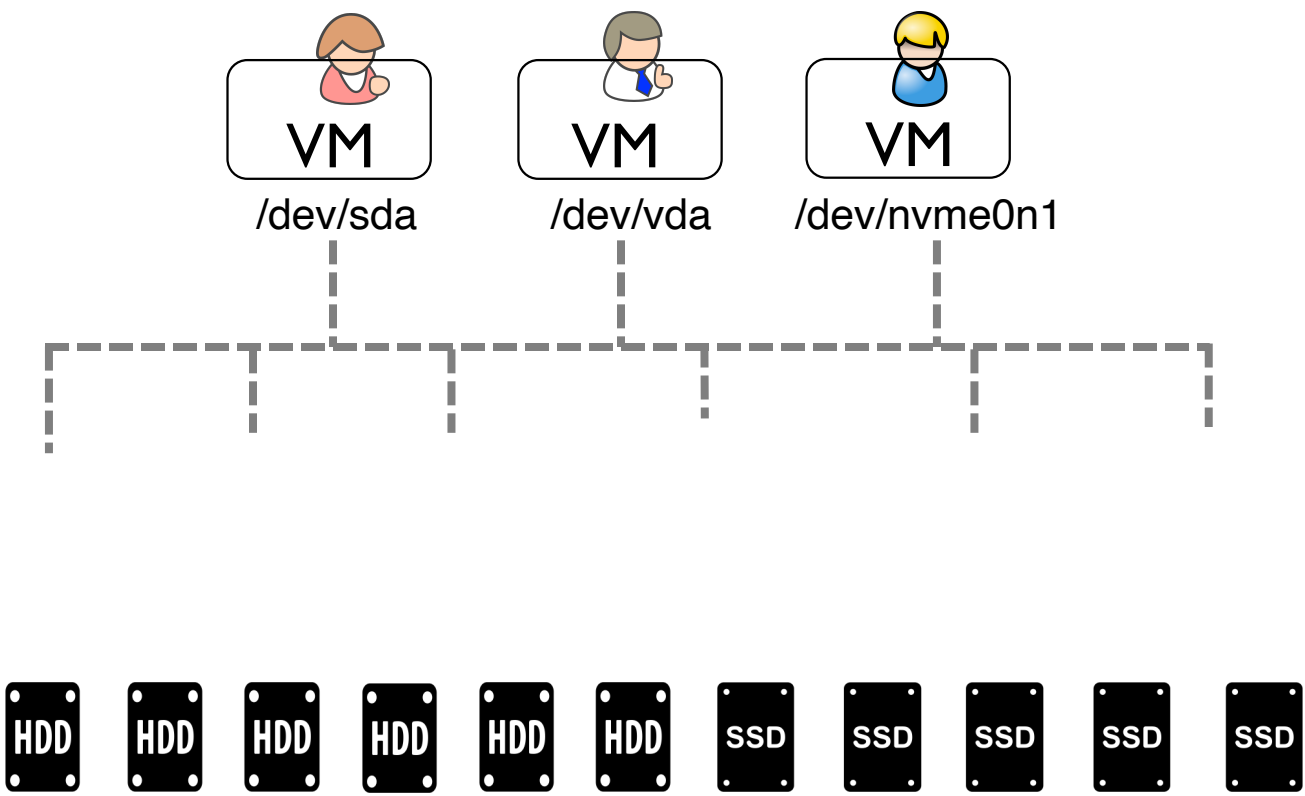


/dev/nvme0n1

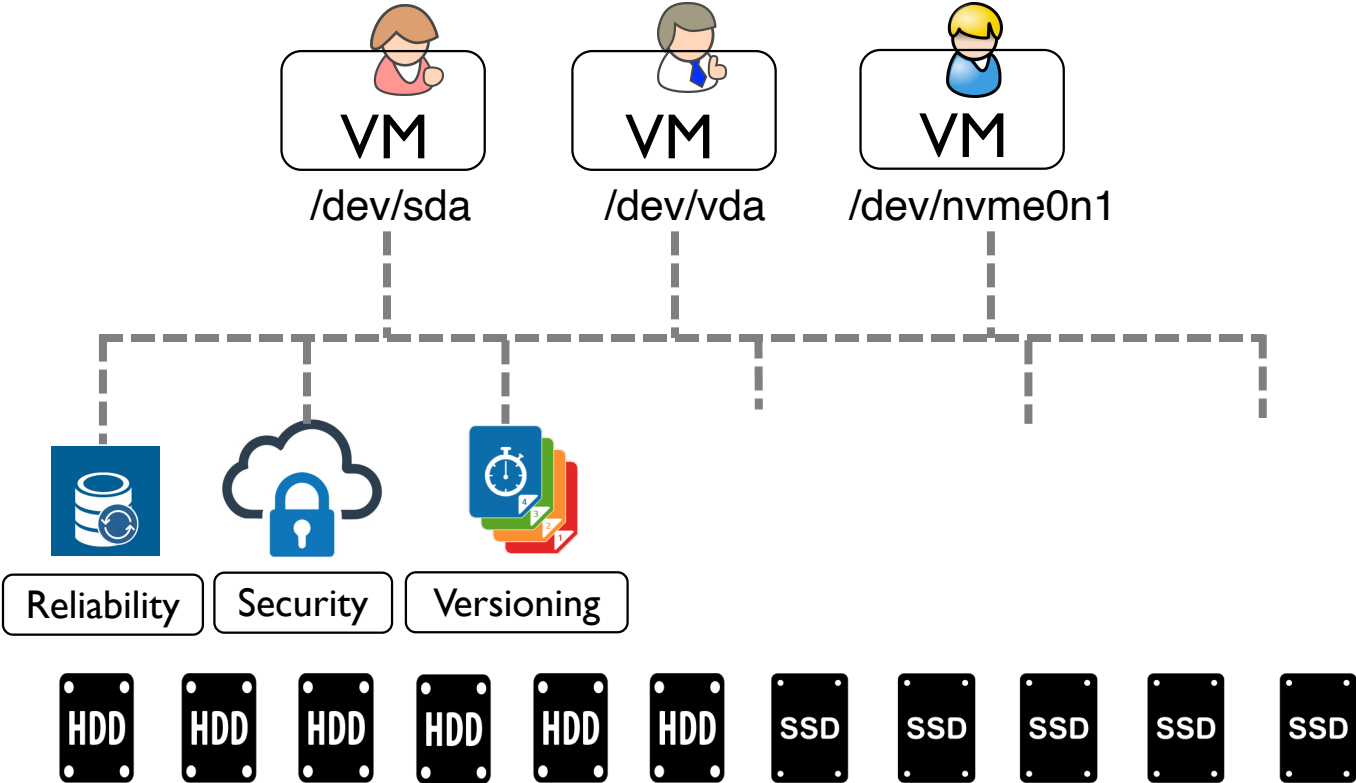
# Today's Cloud Storage Model



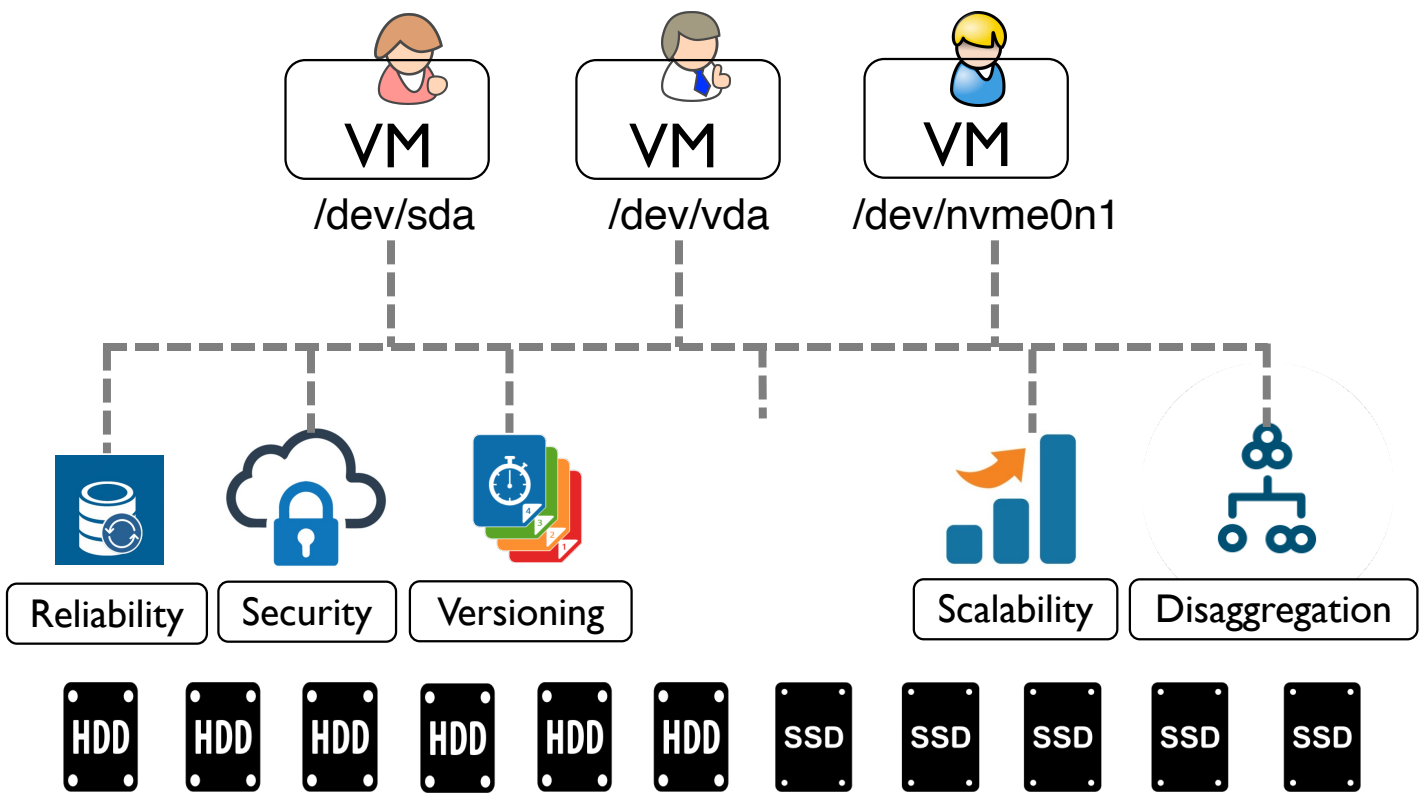
# Today's Cloud Storage Model



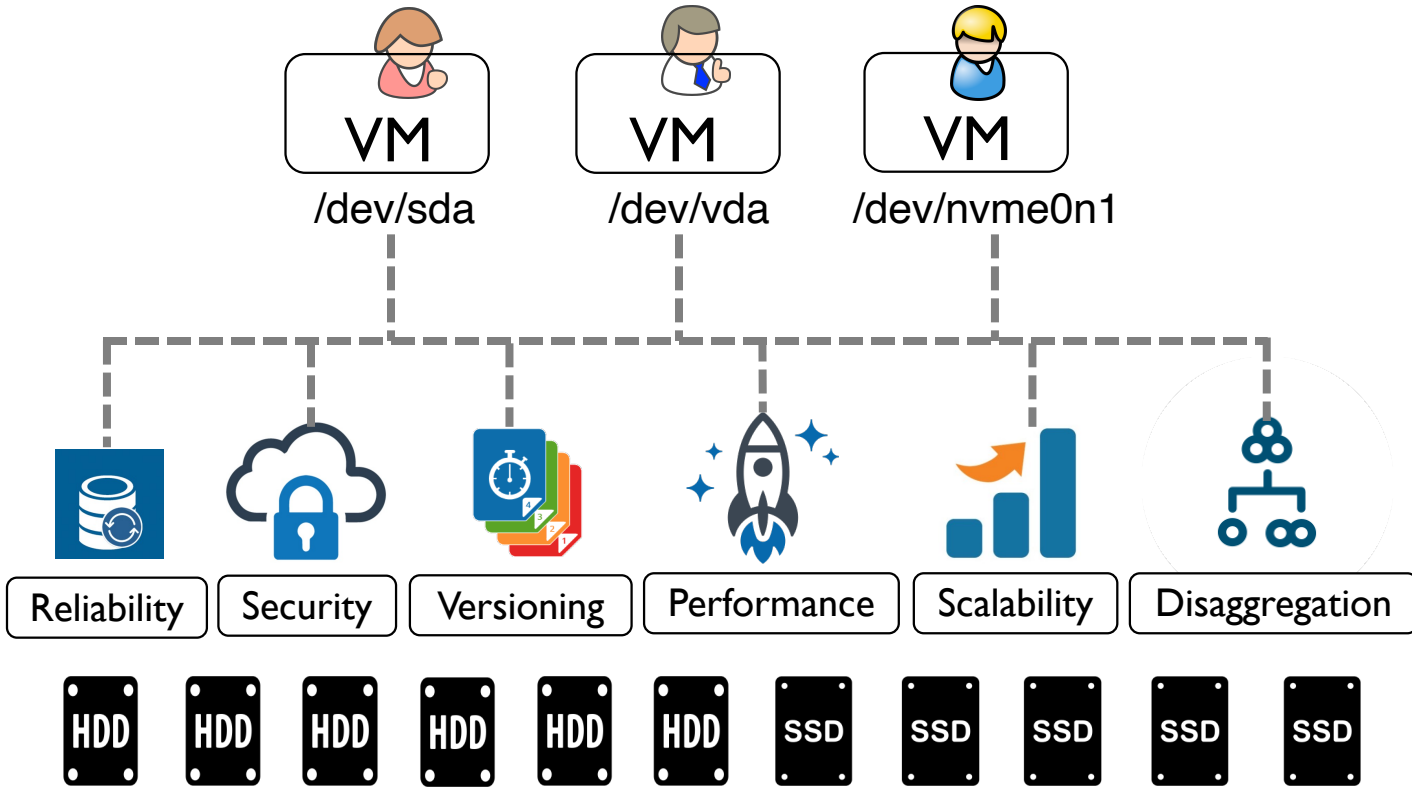
# Today's Cloud Storage Model



# Today's Cloud Storage Model



# Today's Cloud Storage Model

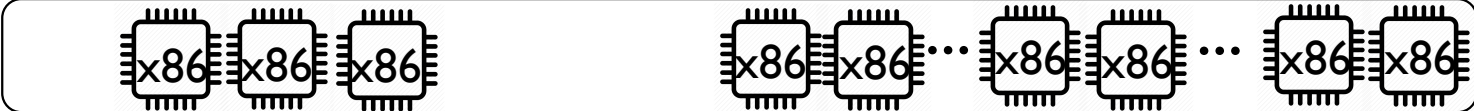




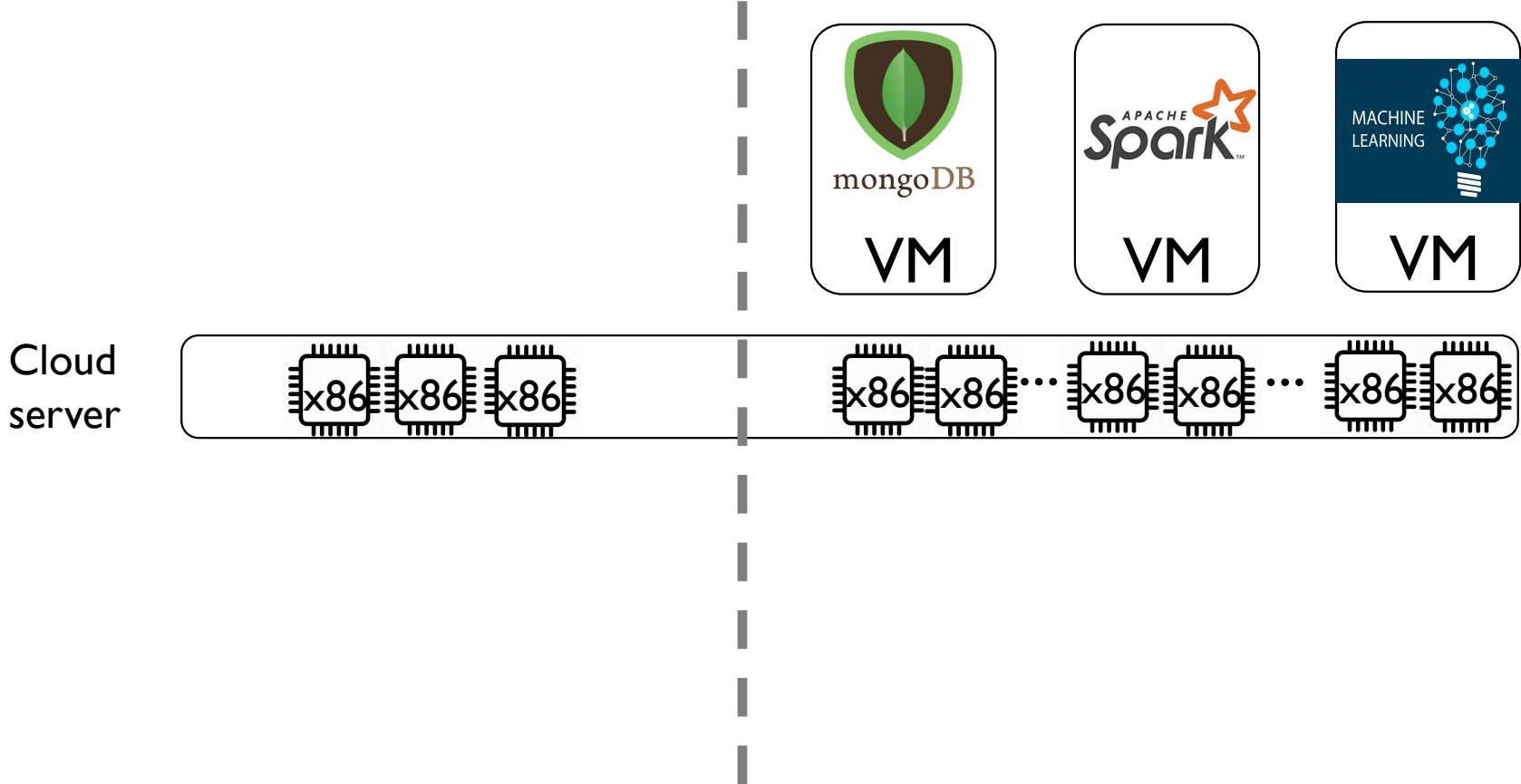
# The Heavy Cloud Storage Tax

# The Heavy Cloud Storage Tax

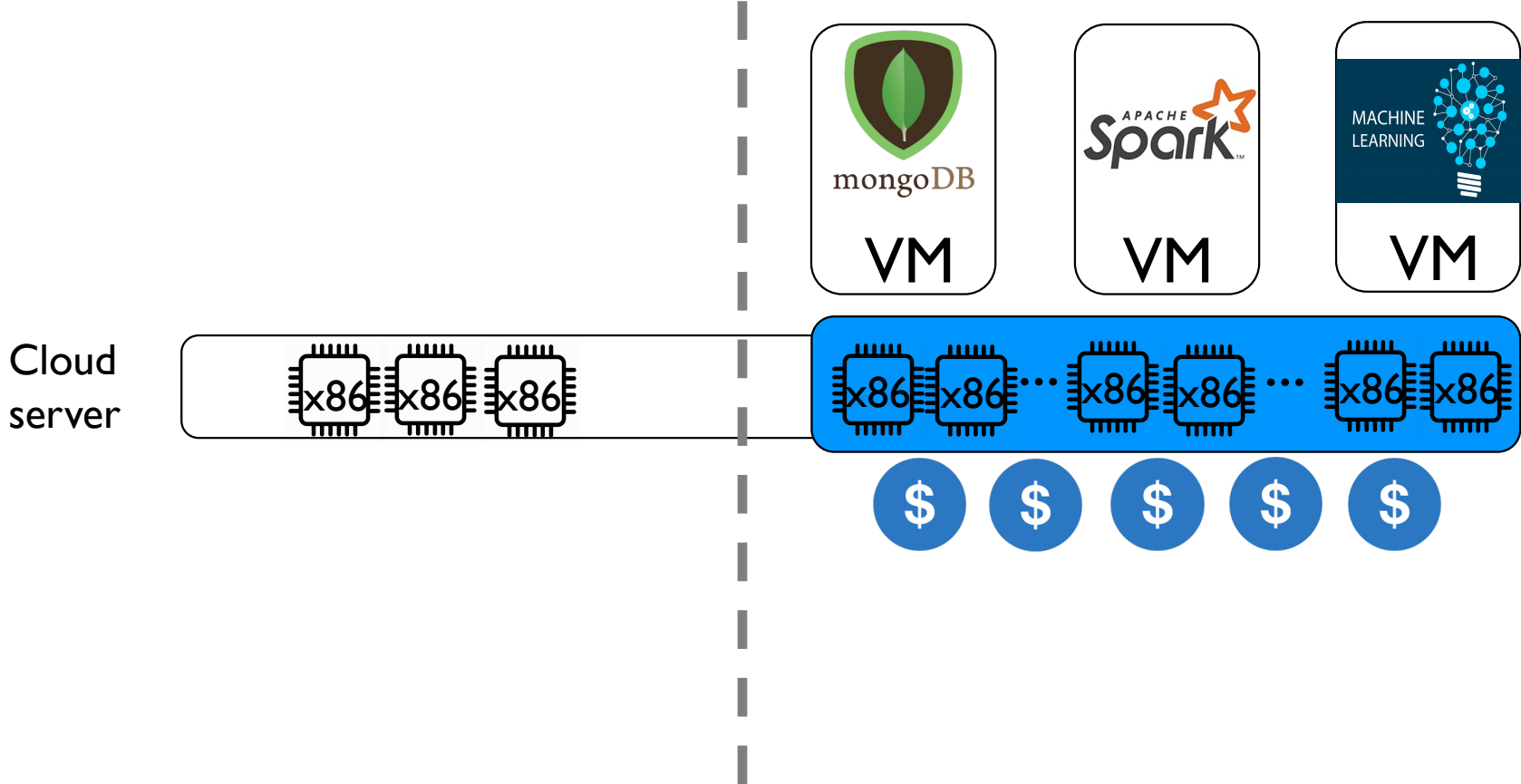
Cloud server



# The Heavy Cloud Storage Tax



# The Heavy Cloud Storage Tax

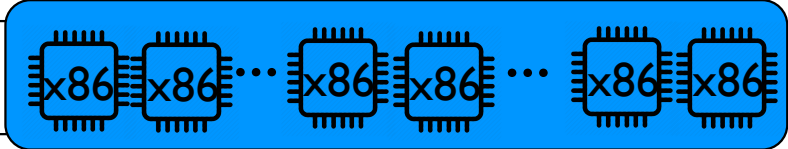
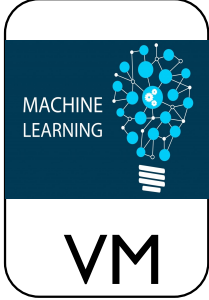
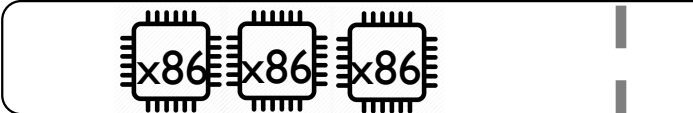


# The Heavy Cloud Storage Tax

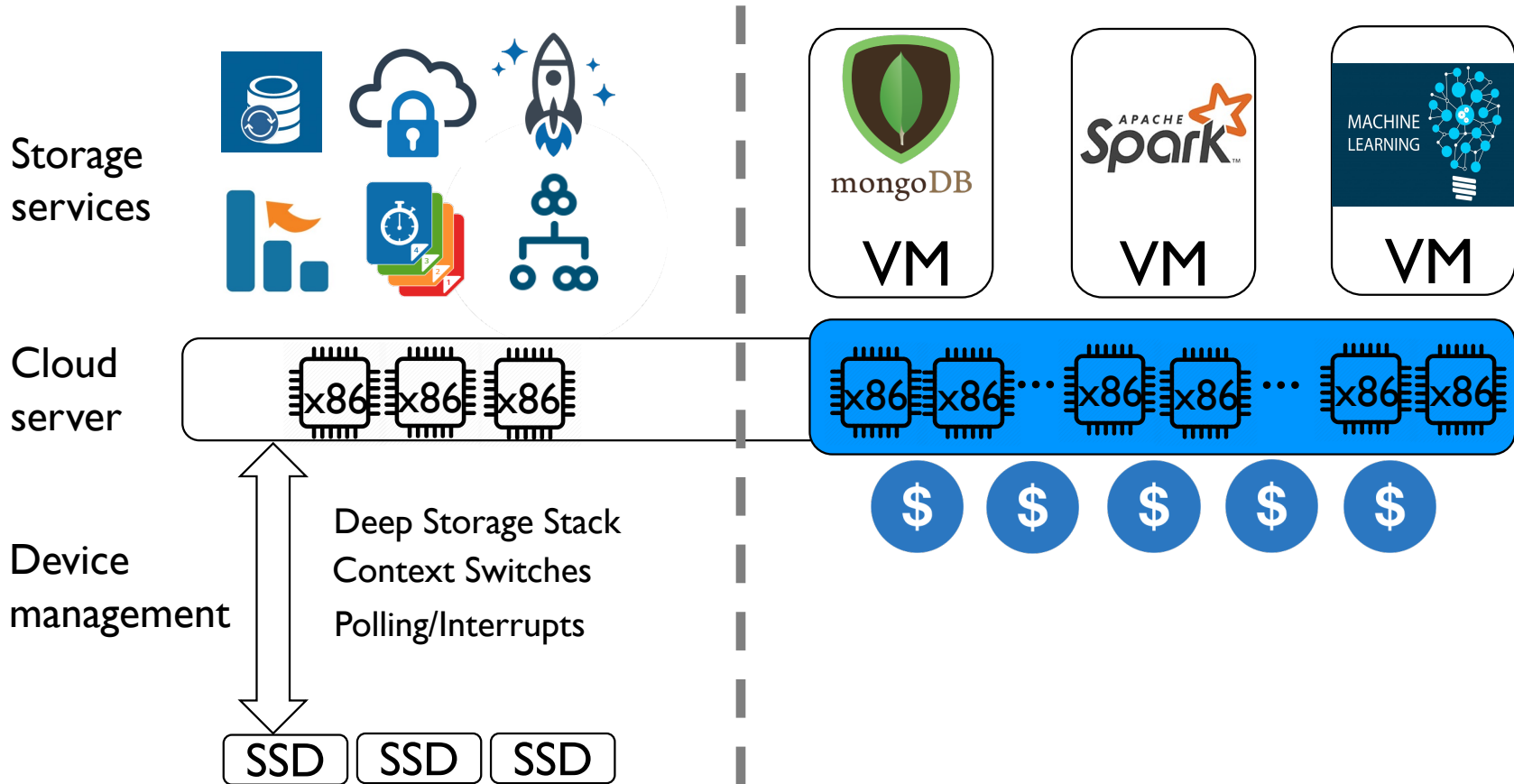
Storage services



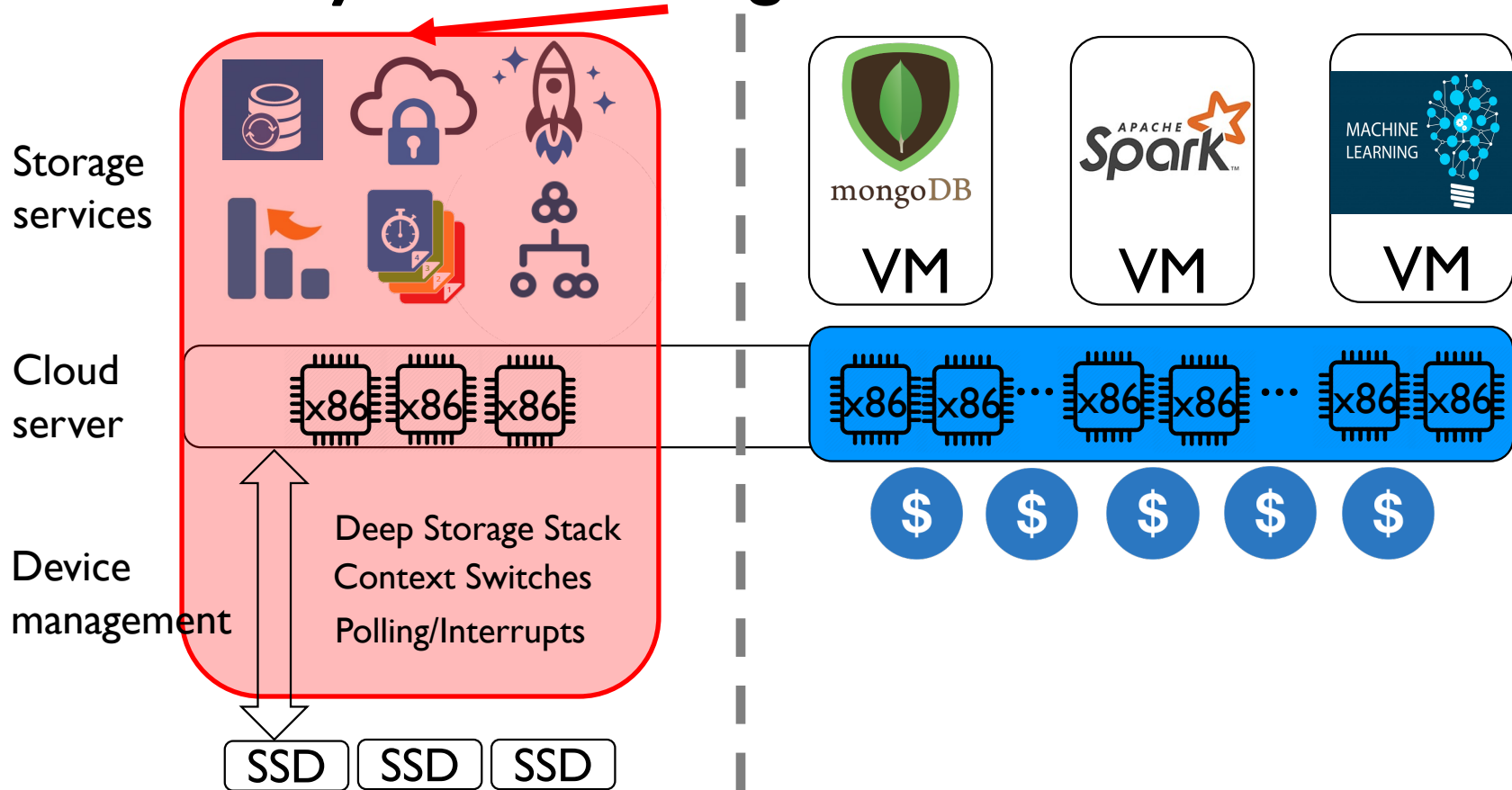
Cloud server



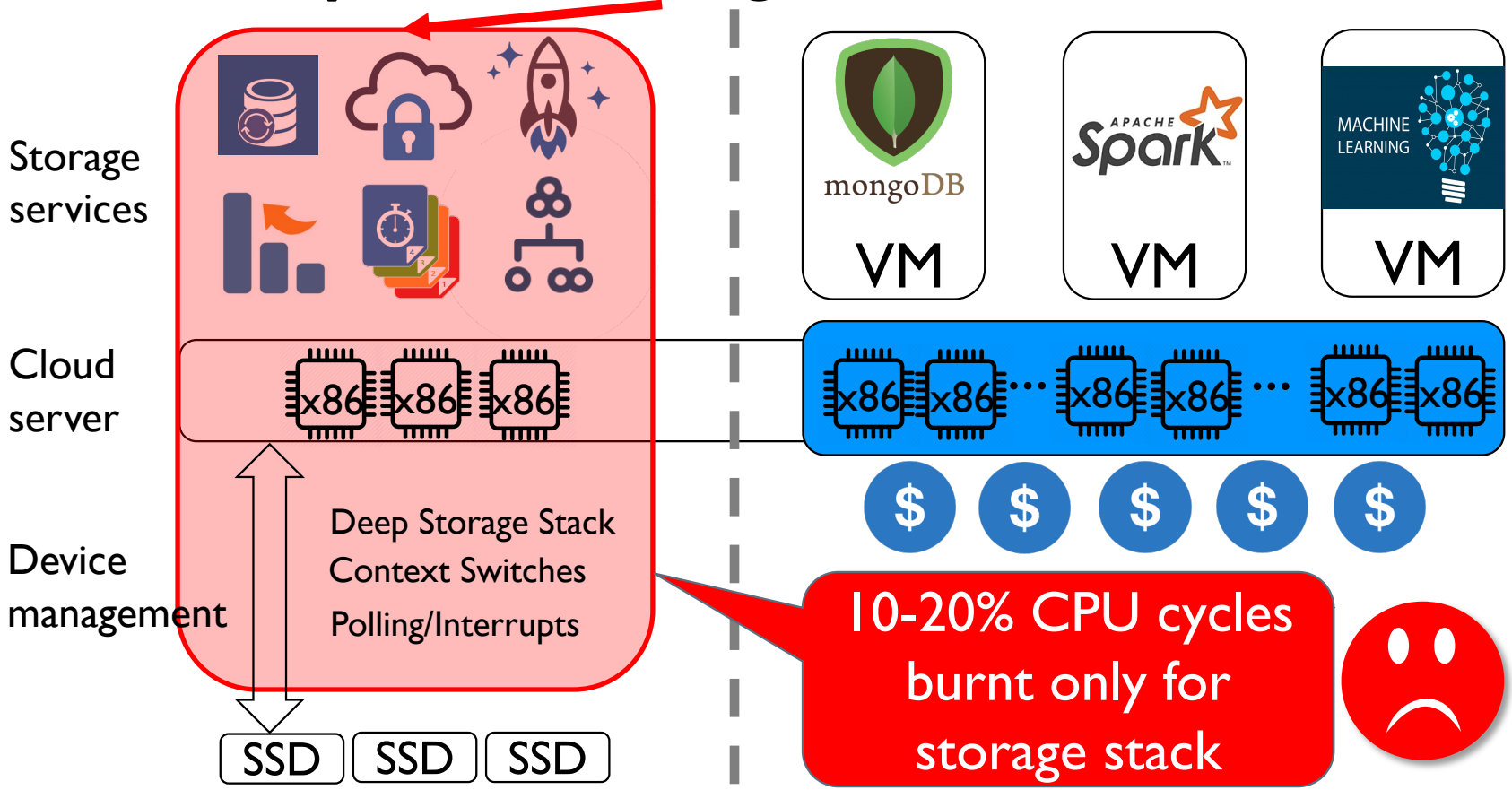
# The Heavy Cloud Storage Tax



# The Heavy Cloud Storage Tax

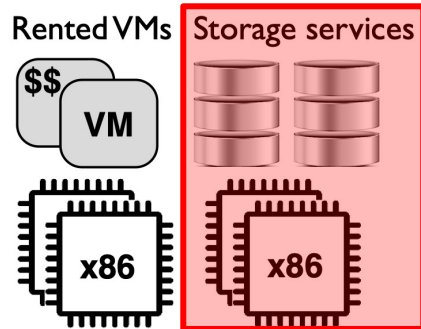


# The Heavy Cloud Storage Tax

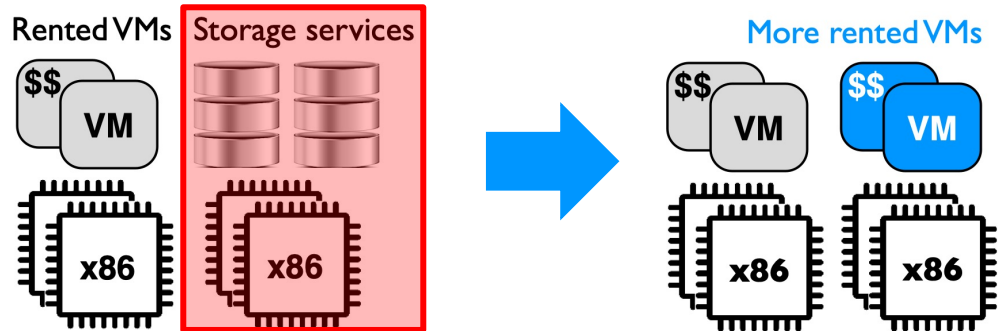




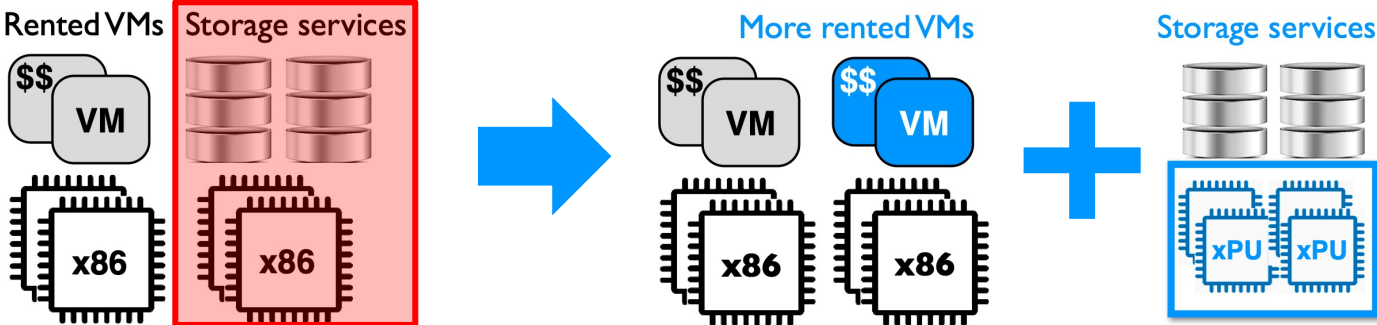
# ARM Offloading for Cost-Efficiency



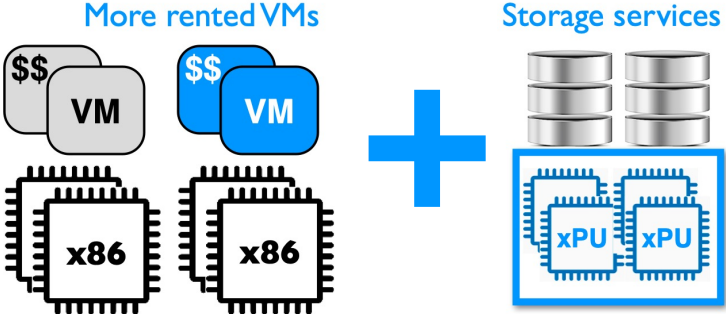
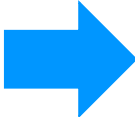
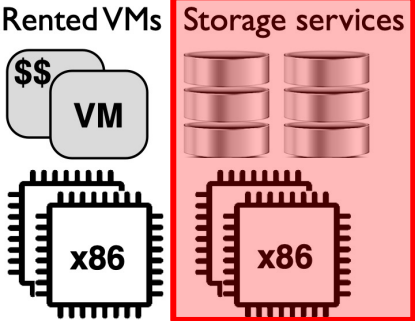
# ARM Offloading for Cost-Efficiency



# ARM Offloading for Cost-Efficiency



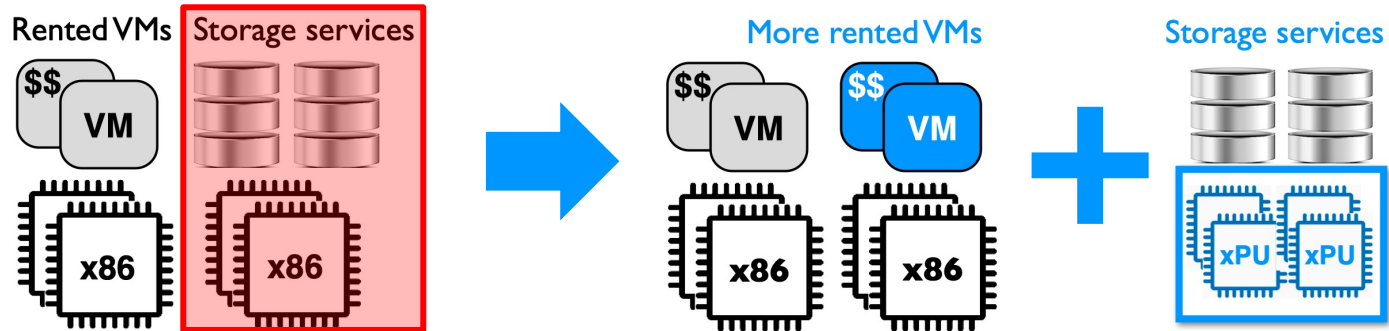
# ARM Offloading for Cost-Efficiency



# ARM Offloading for Cost-Efficiency

Marvell's ThunderX2 Solution Now Deployed for Microsoft Azure Development

Arm-based server clusters for internal workloads spur continued product innovation

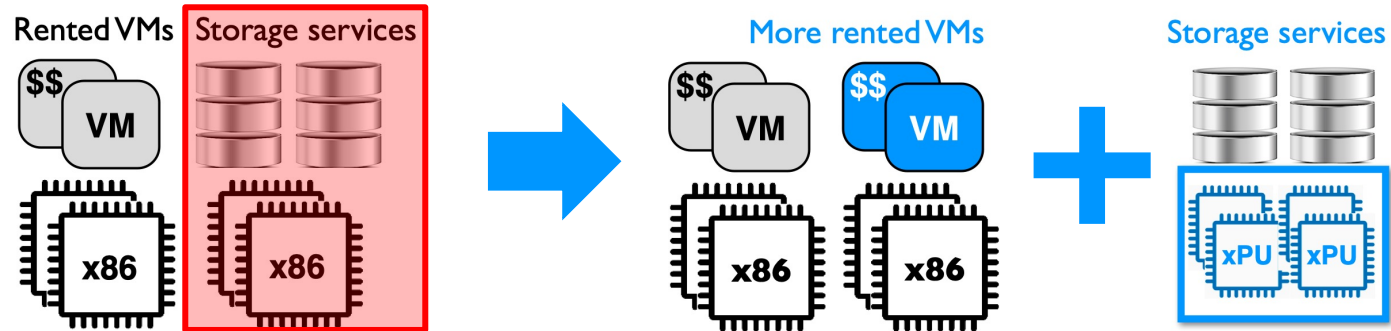


# ARM Offloading for Cost-Efficiency

**FINALLY: AWS GIVES SERVERS A REAL SHOT IN THE ARM**

Marvell's ThunderX2 Solution Now Deployed for Microsoft Azure Development

Arm-based server clusters for internal workloads spur continued product innovation



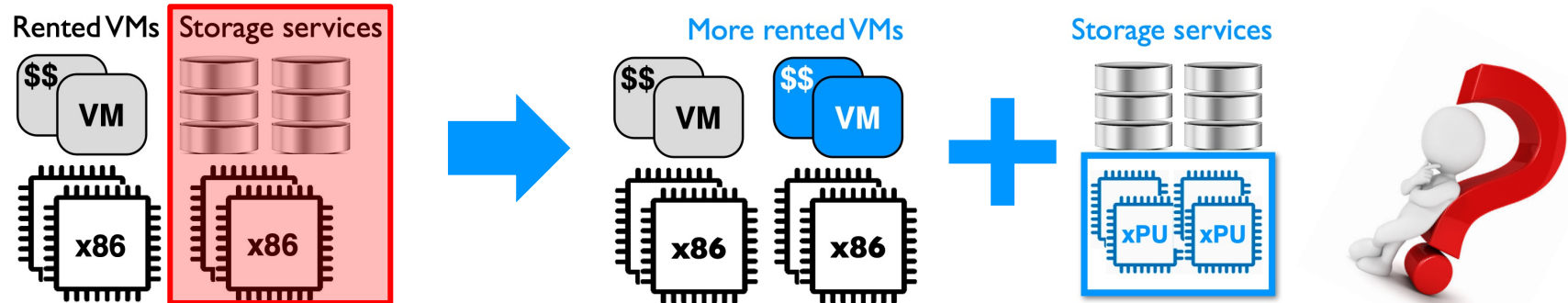
# ARM Offloading for Cost-Efficiency

**FINALLY: AWS GIVES SERVERS A REAL SHOT IN THE ARM**

Marvell's ThunderX2 Solution Now Deployed for Microsoft Azure Development

Arm-based server clusters for internal workloads spur continued product innovation

— ARM SoC TCO: \$100/year  
+ x86 for VMs: \$2000/year



# ARM Offloading for Cost-Efficiency

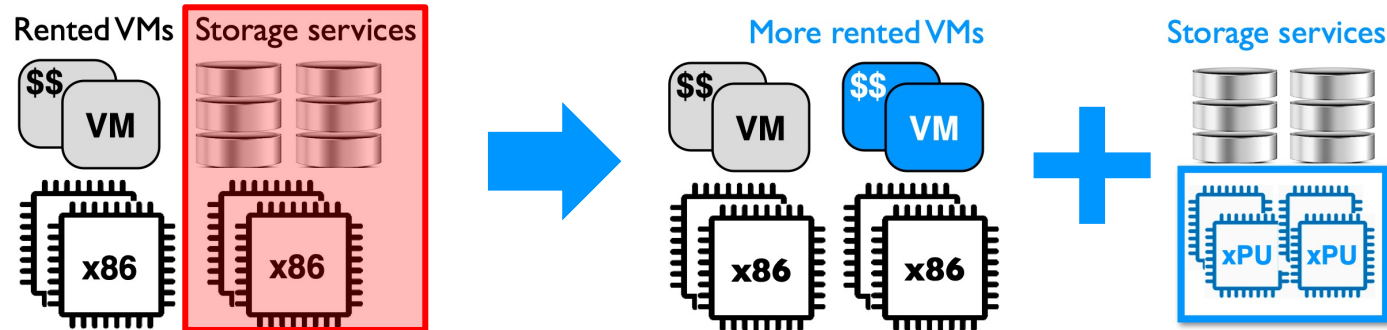
**FINALLY: AWS GIVES SERVERS A REAL SHOT IN THE ARM**

Marvell's ThunderX2 Solution Now Deployed for Microsoft Azure Development

Arm-based server clusters for internal workloads spur continued product innovation

— ARM SoC TCO: \$100/year  
+ x86 for VMs: \$2000/year

~20x revenue gains





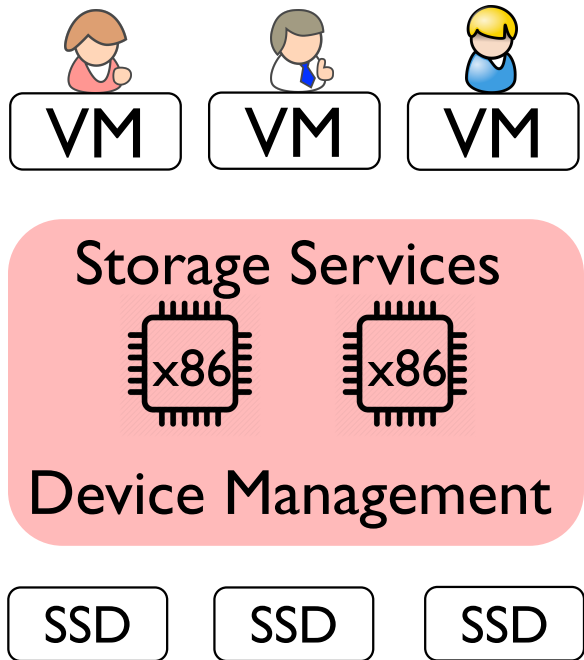
# LeapIO Overview

- Next generation of Cloud Storage Stack **offload-ready** to **ARM** System-on-Chip (SoC)



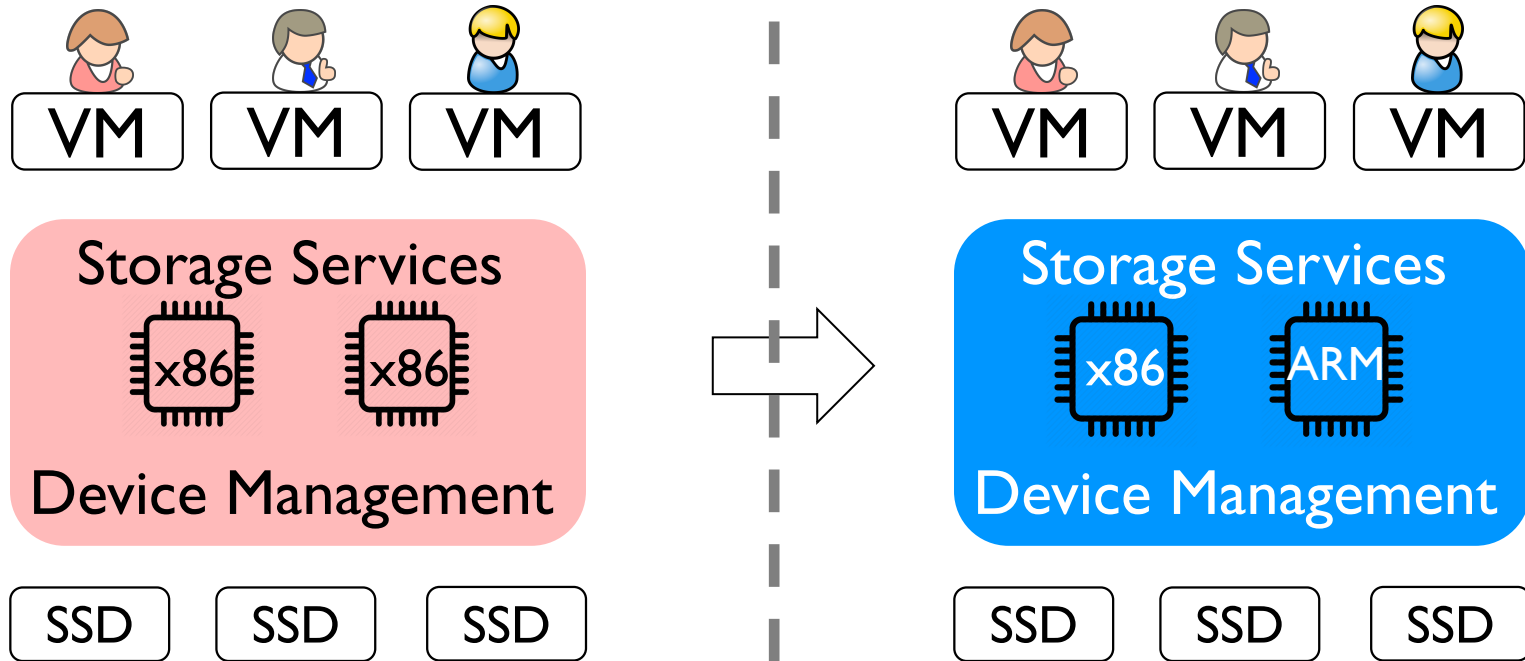
# LeapIO Overview

- Next generation of Cloud Storage Stack **offload-ready** to **ARM** System-on-Chip (SoC)



# LeapIO Overview

- Next generation of Cloud Storage Stack **offload-ready** to **ARM** System-on-Chip (SoC)



Motivation & Goals

LeapIO Architecture

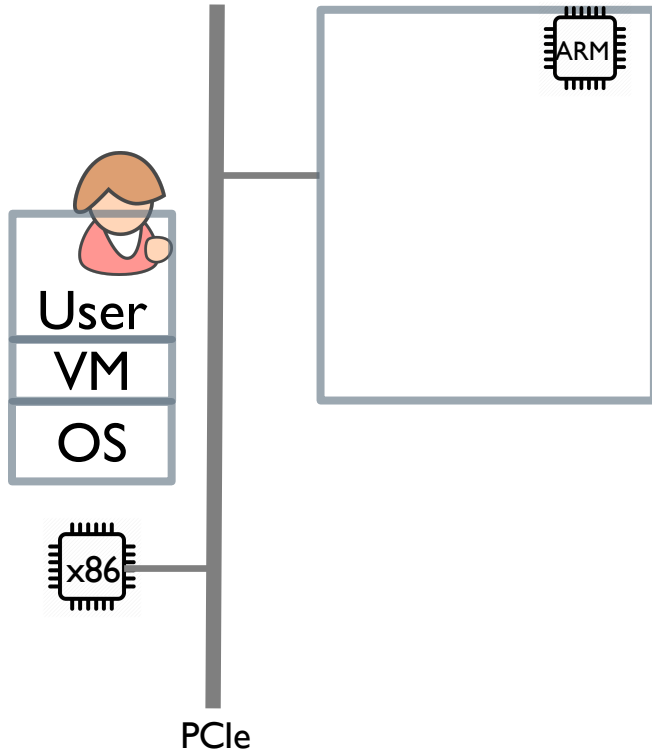
LeapIO Designs

- Portability
- Efficiency

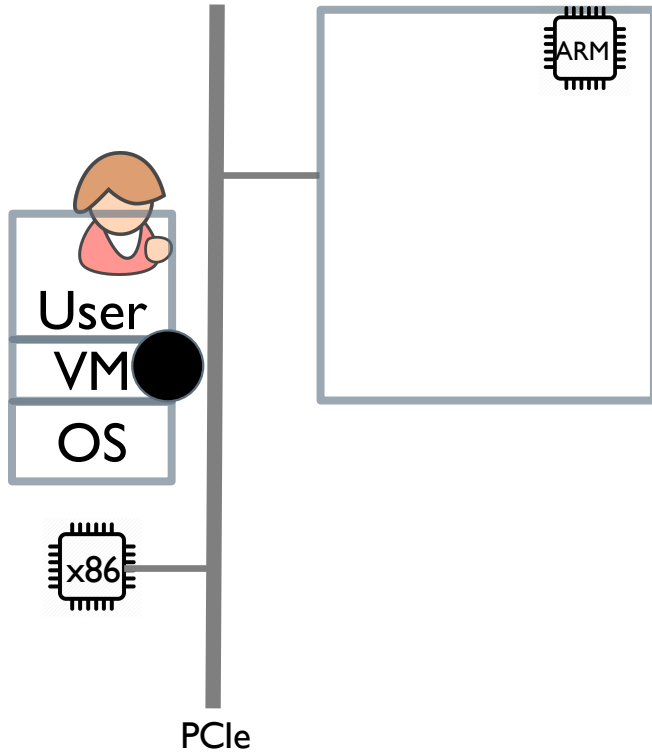
Evaluation

Conclusion

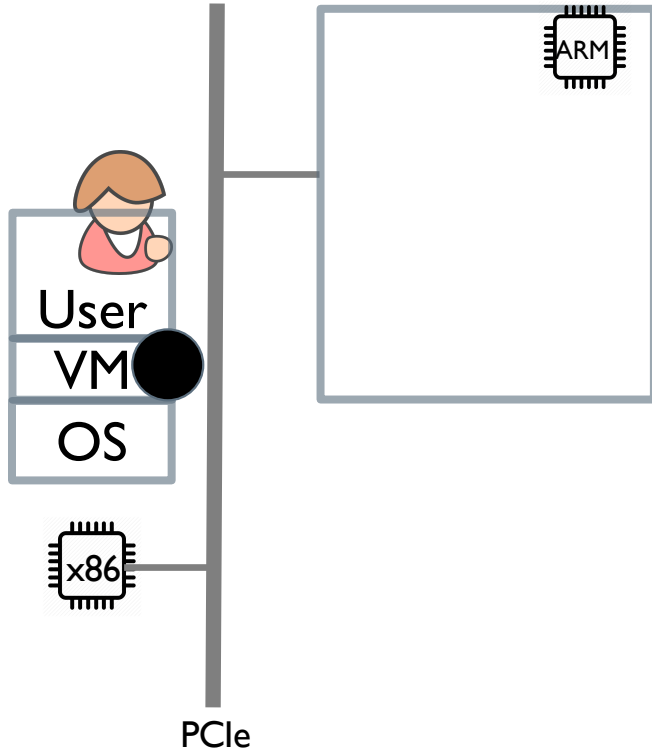
# LeapIO Architecture



# LeapIO Architecture

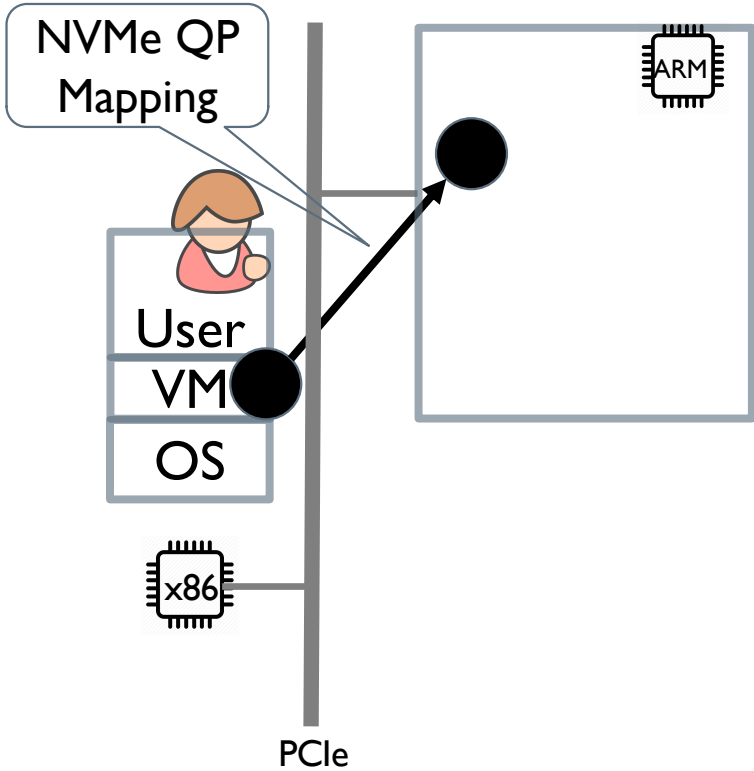


# LeapIO Architecture



NVMe Queue Pair (QP): ● =  

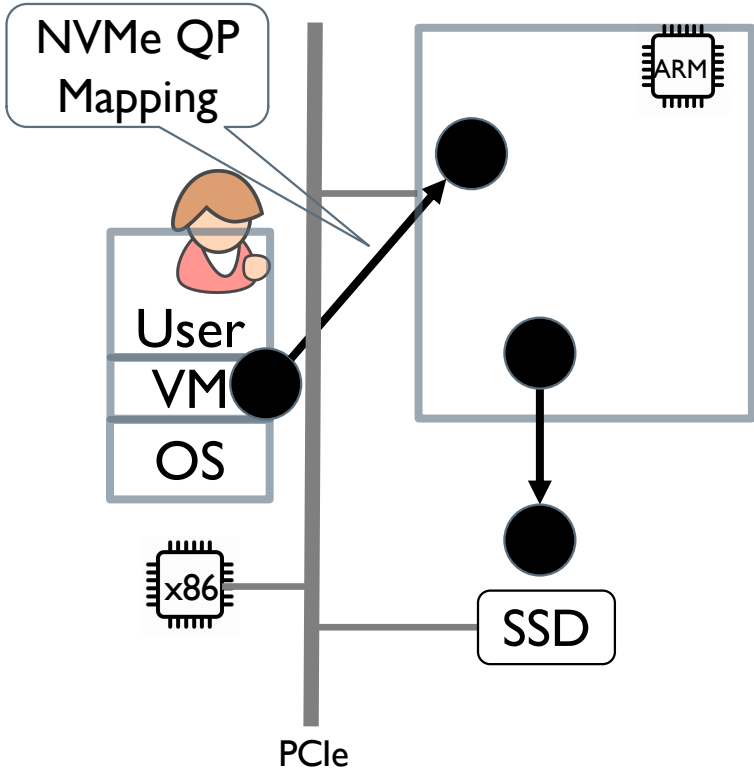
# LeapIO Architecture




NVMe Queue Pair (QP): ● = SQ CQ

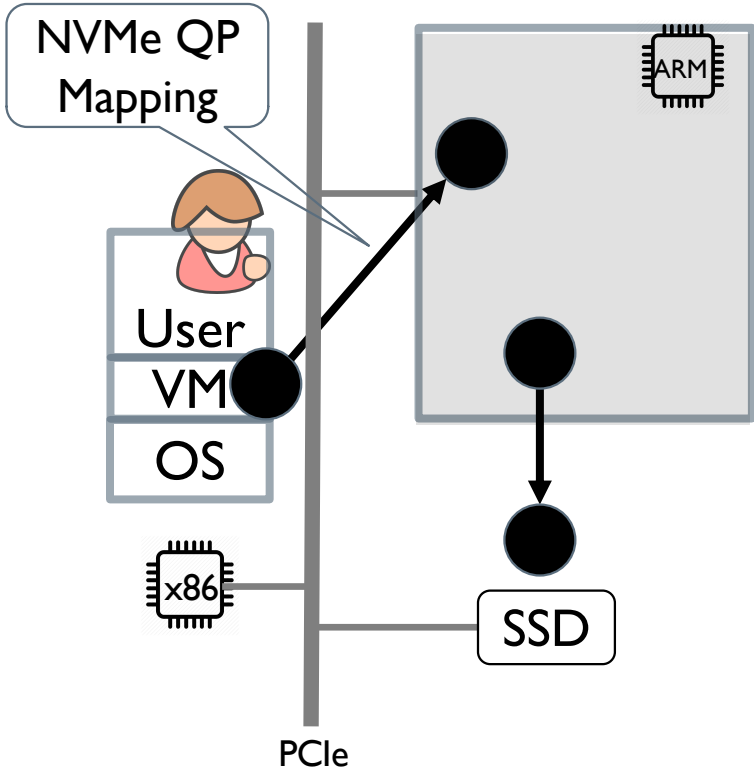


# LeapIO Architecture



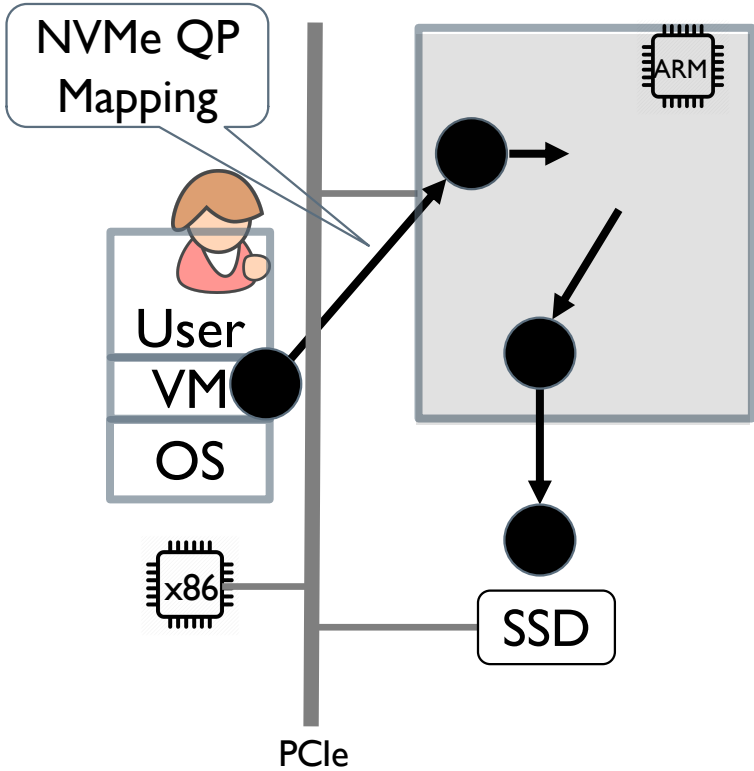
NVMe Queue Pair (QP): ● =  

# LeapIO Architecture



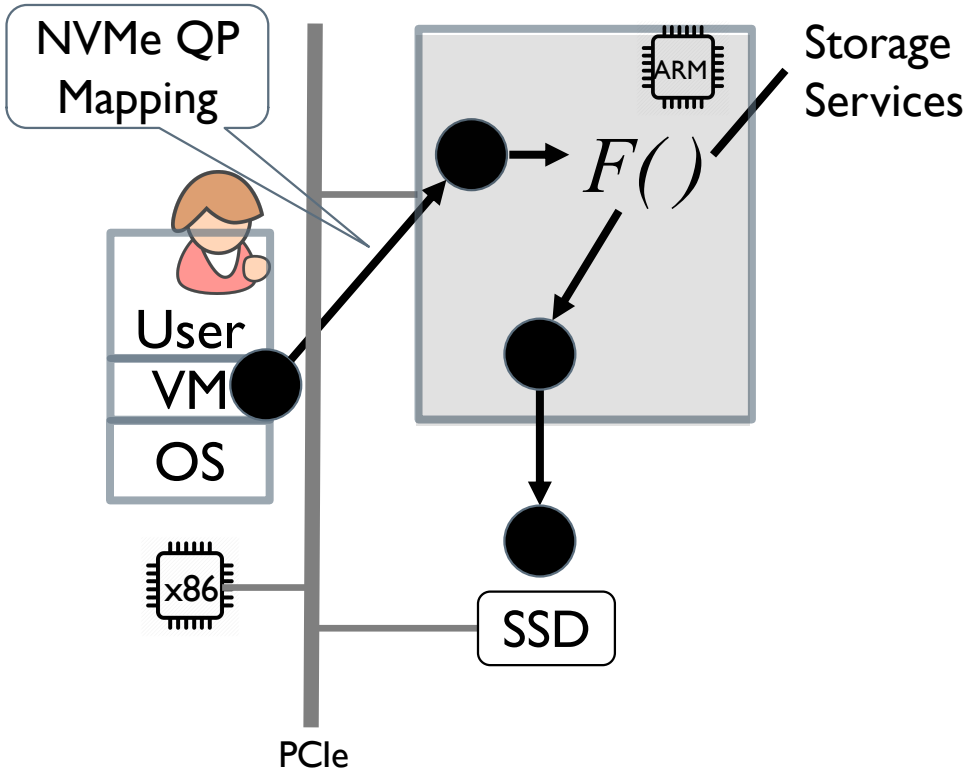
NVMe Queue Pair (QP): ● = SQ CQ

# LeapIO Architecture



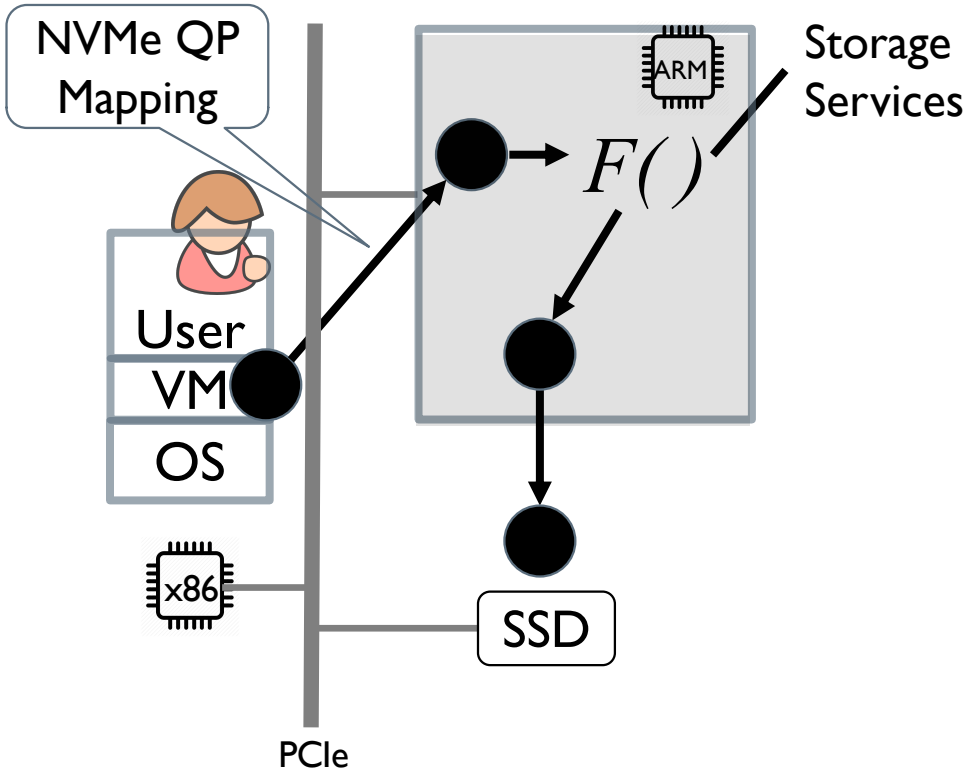
NVMe Queue Pair (QP): ● = SQ CQ

# LeapIO Architecture



NVMe Queue Pair (QP): ● = SQ CQ

# LeapIO Architecture

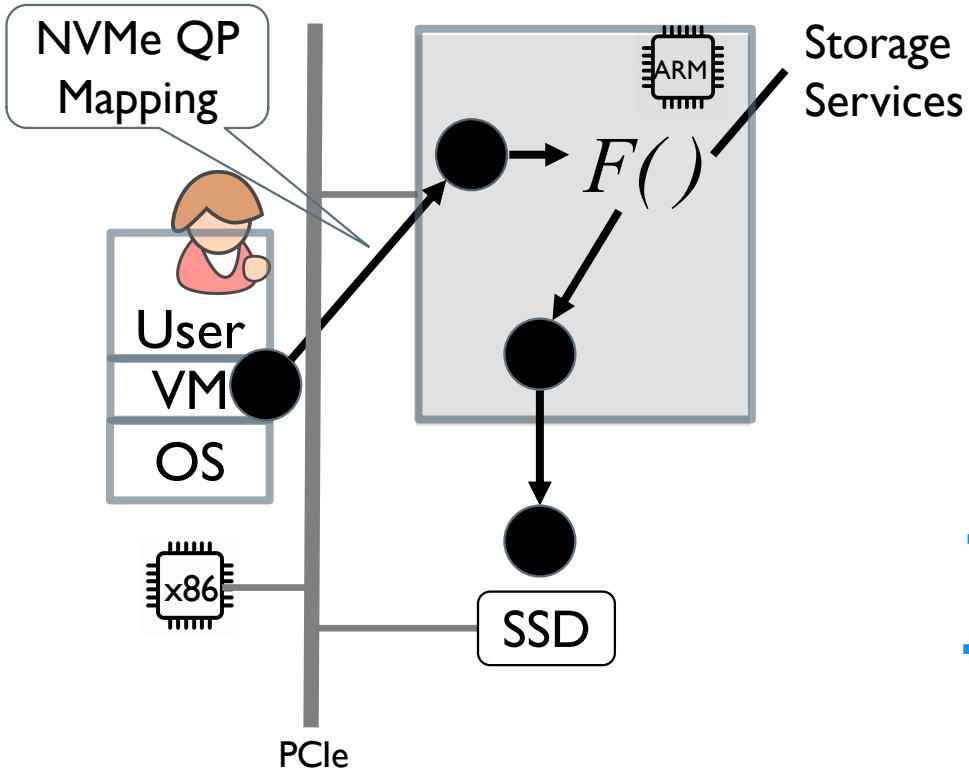


Design benefits:

- + Service extensibility

NVMe Queue Pair (QP): ● =  

# LeapIO Architecture

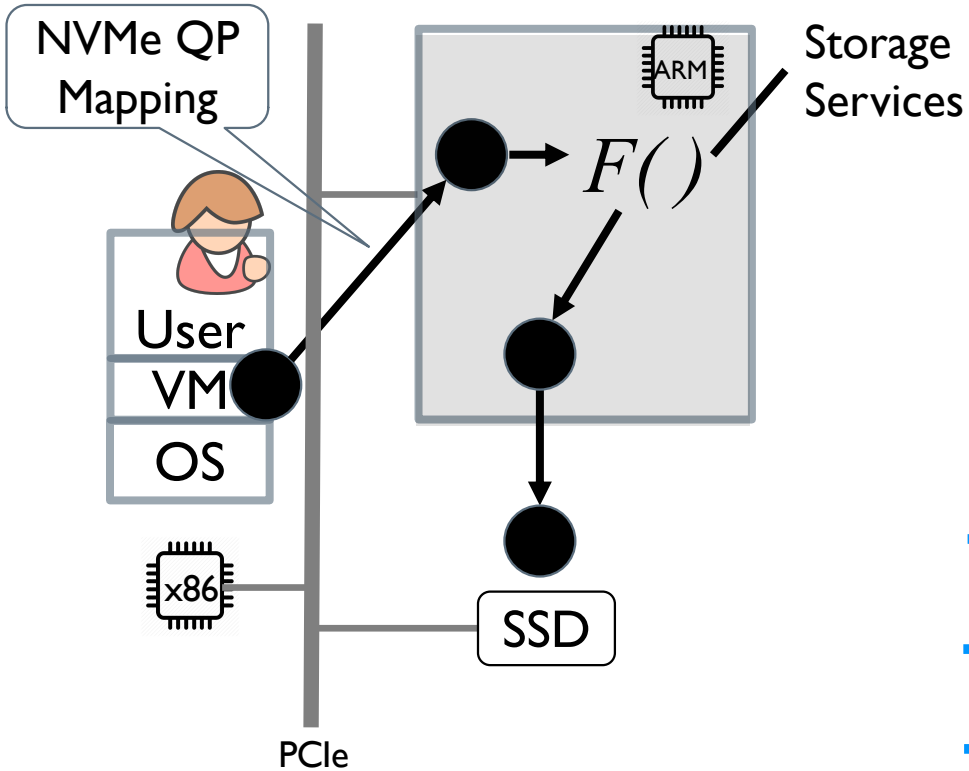


Design benefits:

- + Service extensibility
- + Polling for efficiency

NVMe Queue Pair (QP): ● =  

# LeapIO Architecture

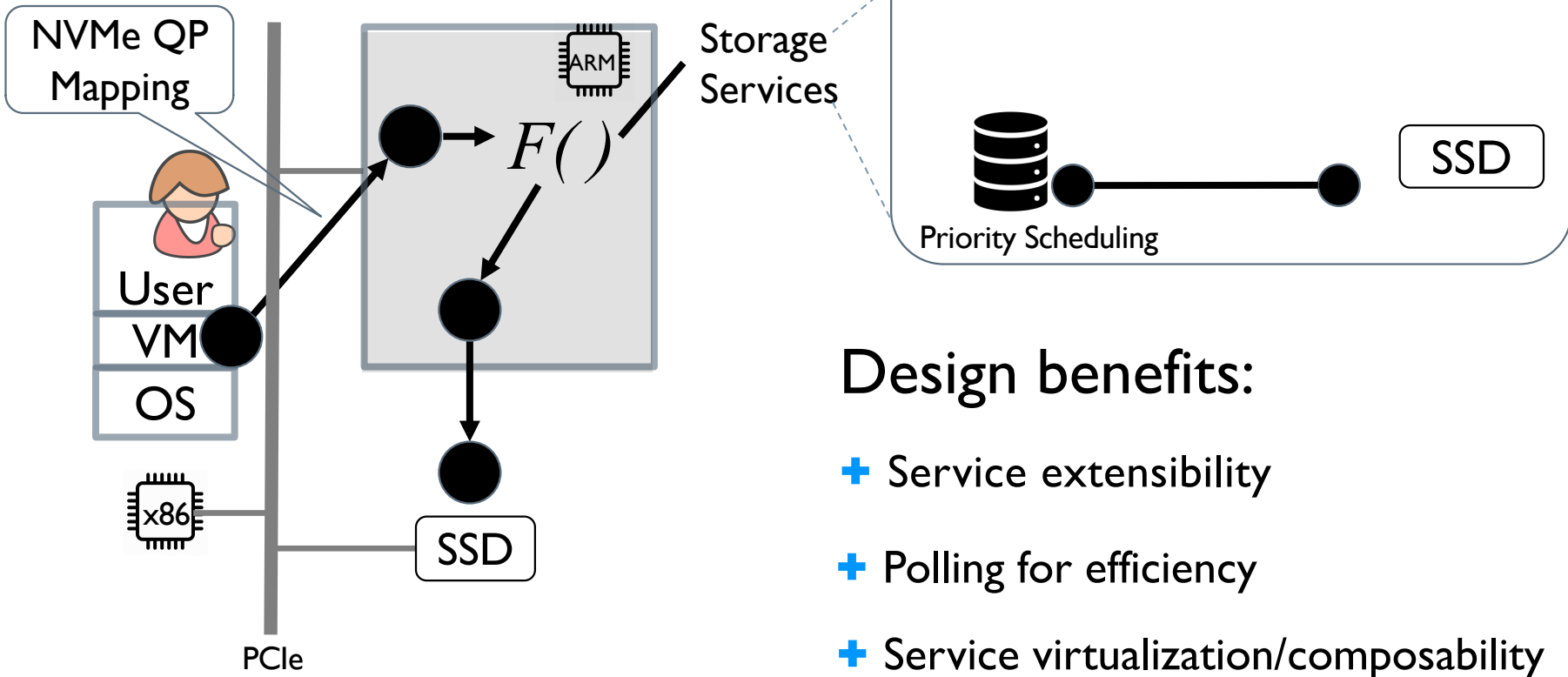


## Design benefits:

- + Service extensibility
- + Polling for efficiency
- + Service virtualization/composability

NVMe Queue Pair (QP): ● =  

# LeapIO Architecture

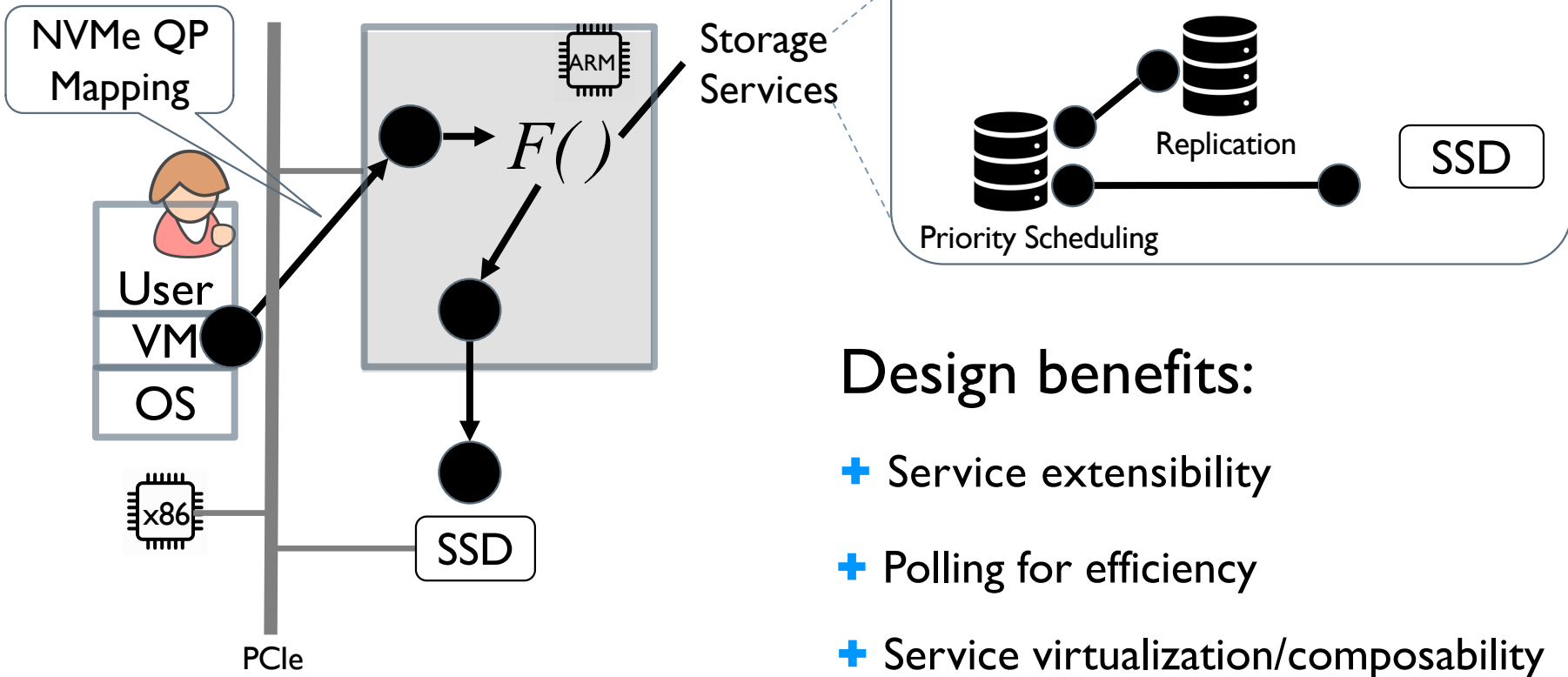


## Design benefits:

- + Service extensibility
- + Polling for efficiency
- + Service virtualization/composability



# LeapIO Architecture

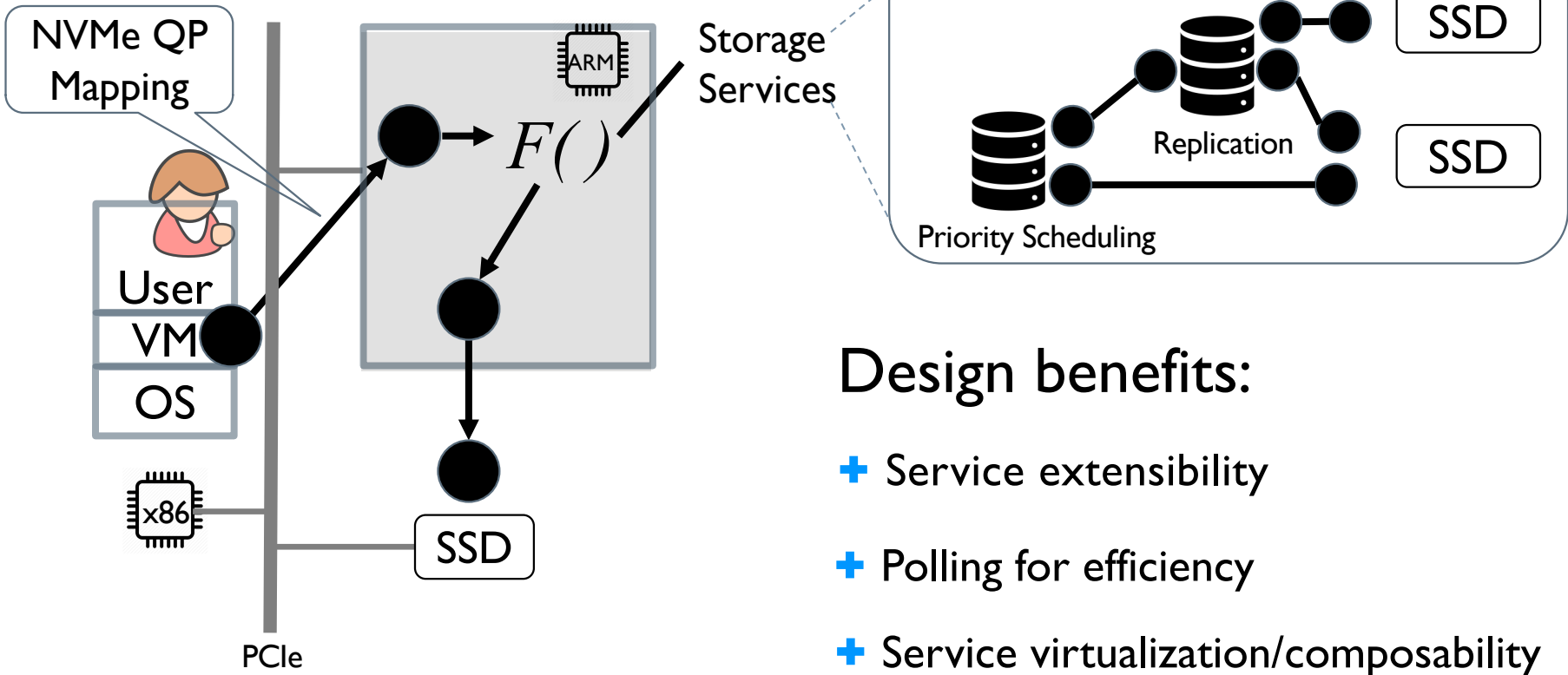


## Design benefits:

- + Service extensibility
- + Polling for efficiency
- + Service virtualization/composability

NVMe Queue Pair (QP): ● = SQ CQ

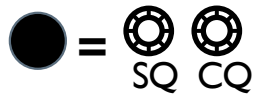
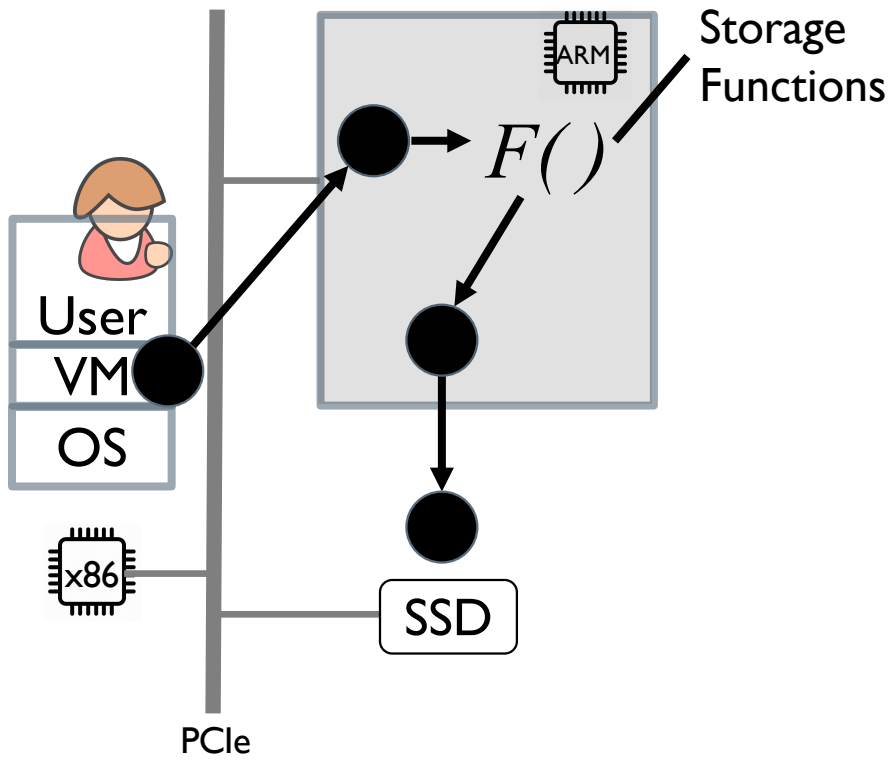
# LeapIO Architecture



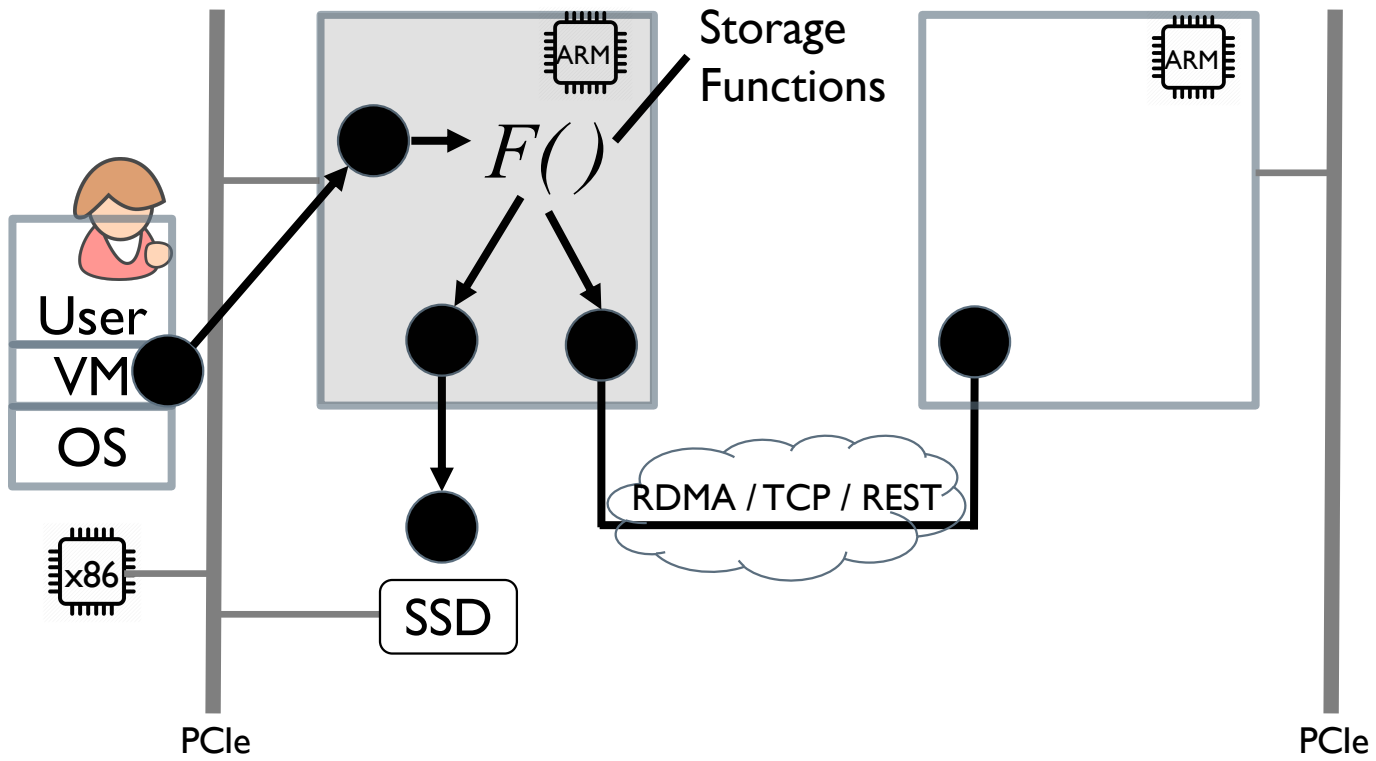
## Design benefits:

- + Service extensibility
- + Polling for efficiency
- + Service virtualization/composability

NVMe Queue Pair (QP): ● = SQ CQ



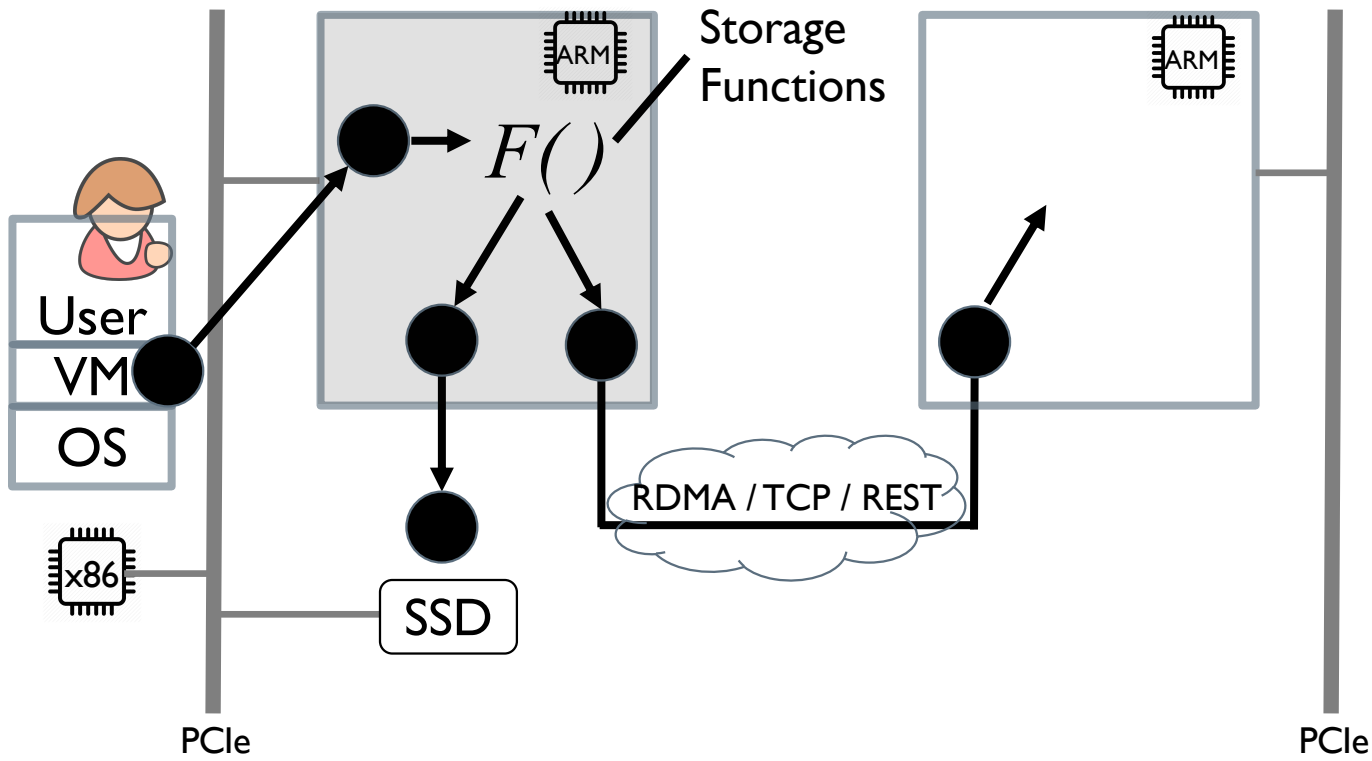
Client



● = SQ CQ

Client

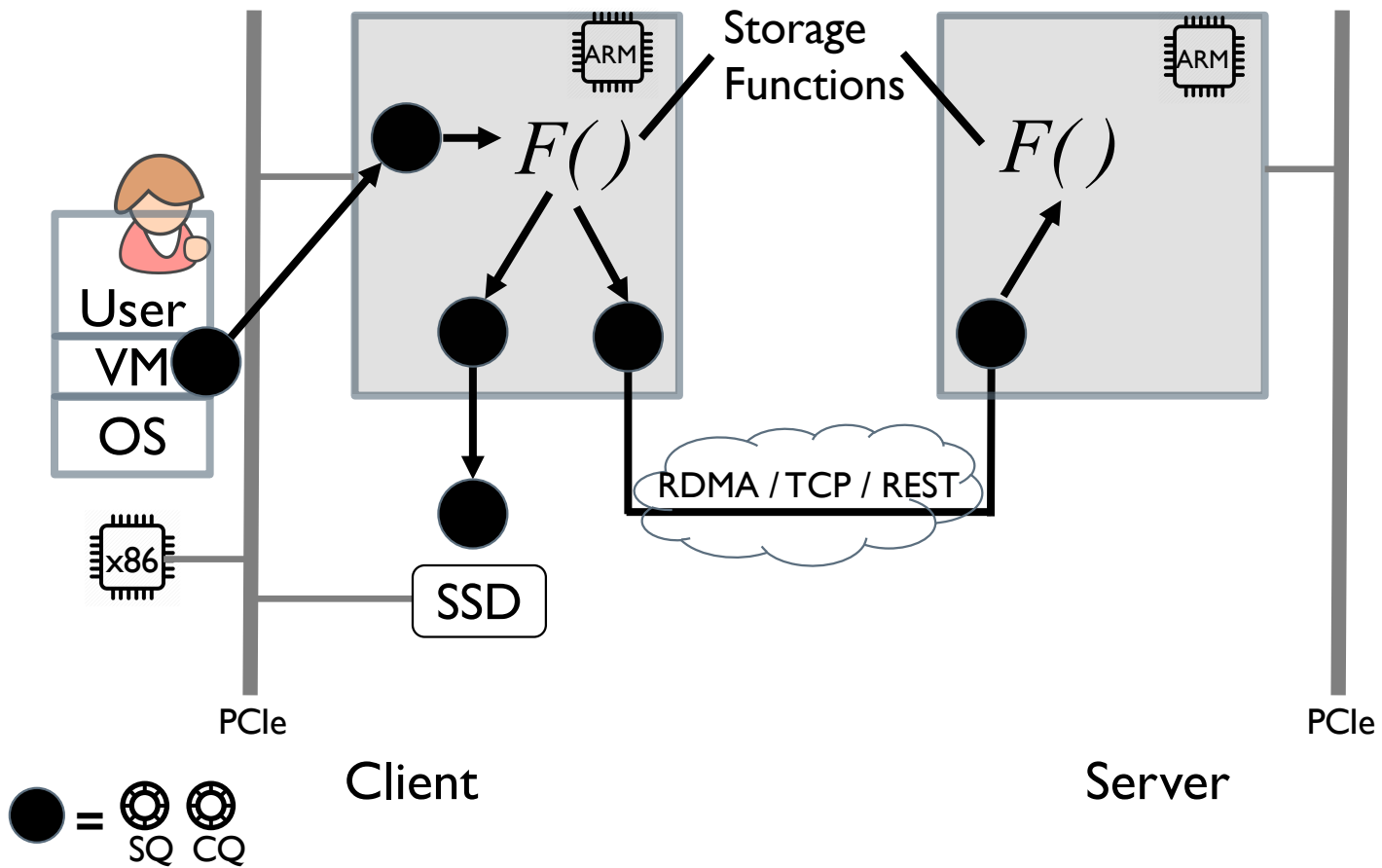
Server

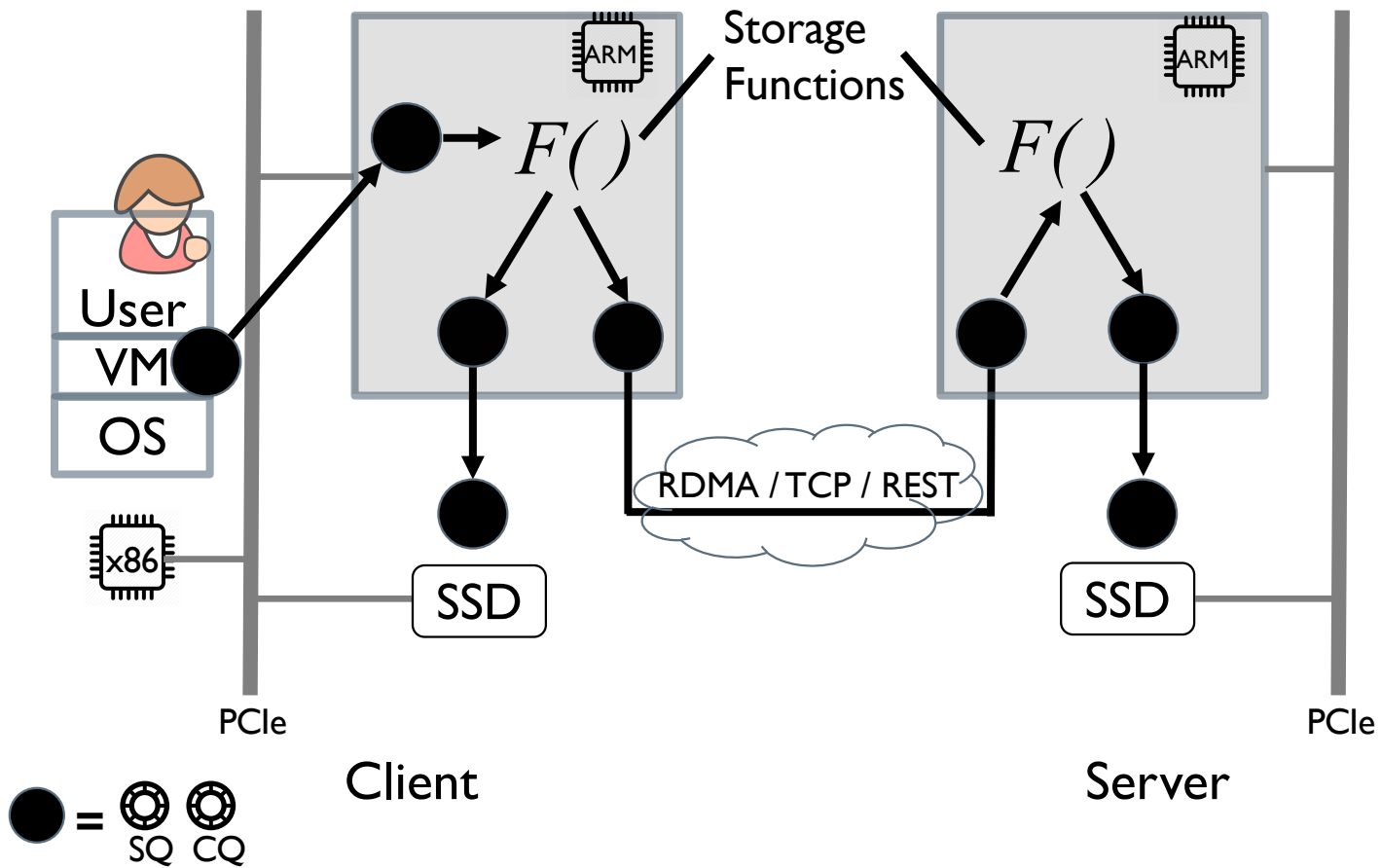


● = SQ CQ

Client

Server





# LeapIO Challenges

How to achieve portability?

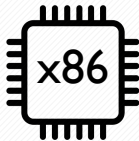
How to achieve bare-metal performance?



# The Need for Portability

# The Need for Portability

## I<sup>st</sup> Generation



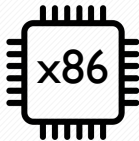
TCP

SSD

# The Need for Portability

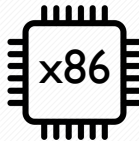
1<sup>st</sup> Generation

2<sup>nd</sup> Generation



TCP

SSD

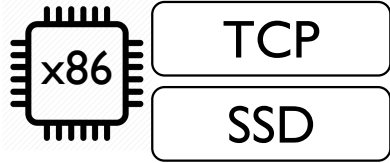


RDMA

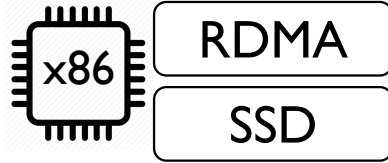
SSD

# The Need for Portability

1<sup>st</sup> Generation



2<sup>nd</sup> Generation

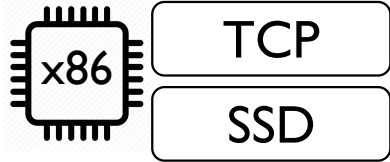


3<sup>rd</sup> Generation

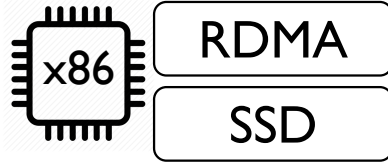


# The Need for Portability

1<sup>st</sup> Generation



2<sup>nd</sup> Generation



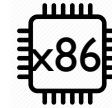
3<sup>rd</sup> Generation



Try to avoid fragmenting the server fleet into silos defined by their hardware capabilities and specific software optimizations

# LeapIO Portability

User VM

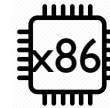


# LeapIO Portability

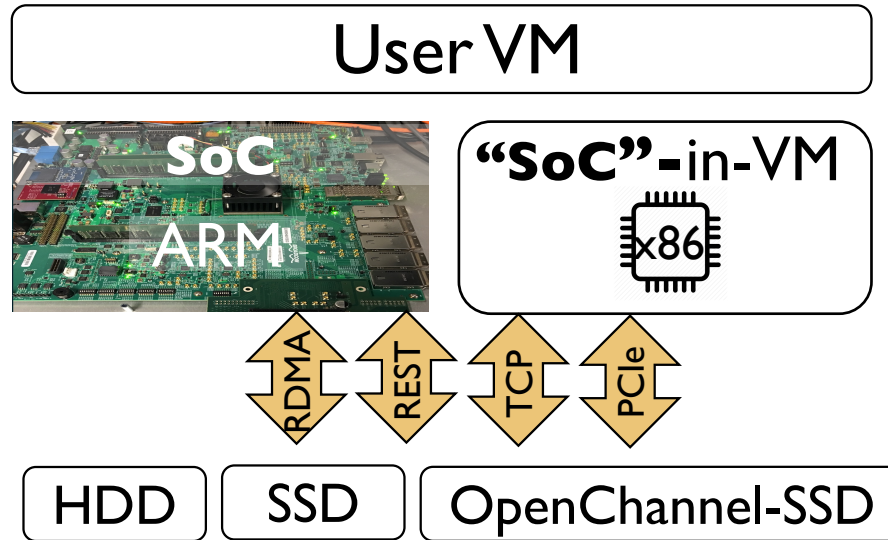
User VM



“SoC”-in-VM



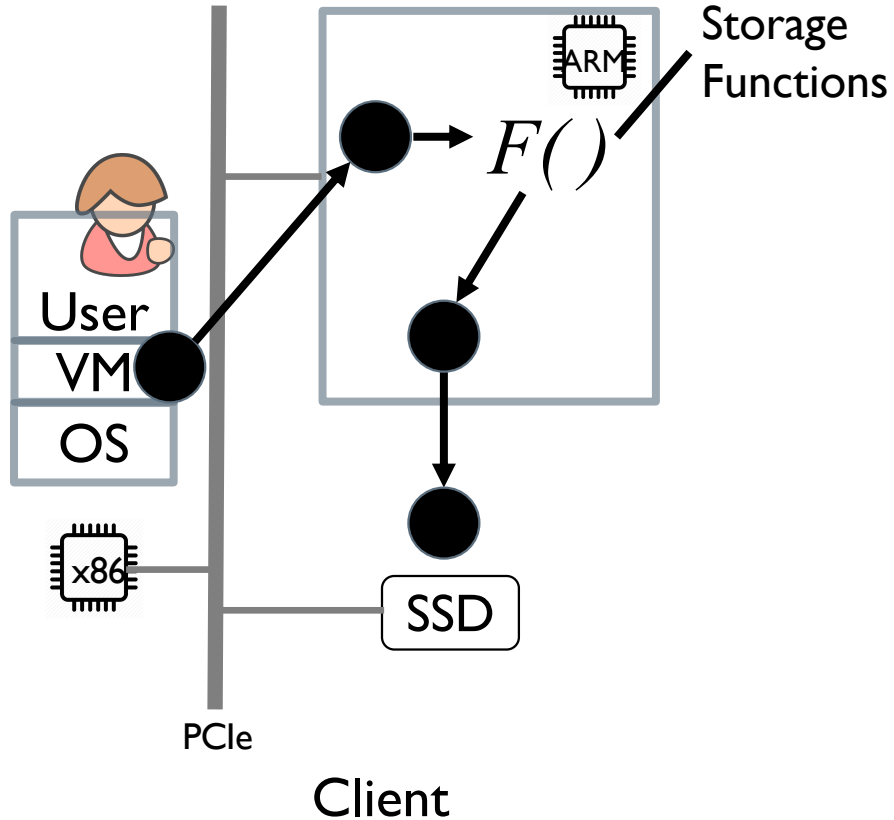
# LeapIO Portability



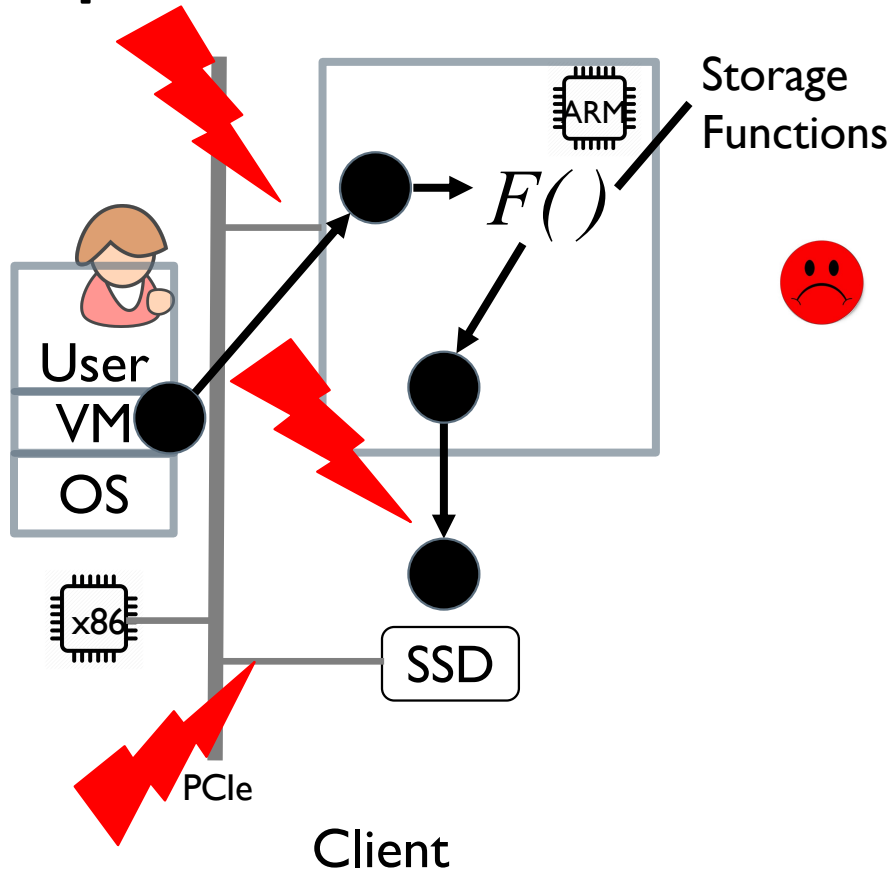
Treat Hardware Acceleration as  
Opportunity instead of Necessity



# LeapIO Efficient Data Path

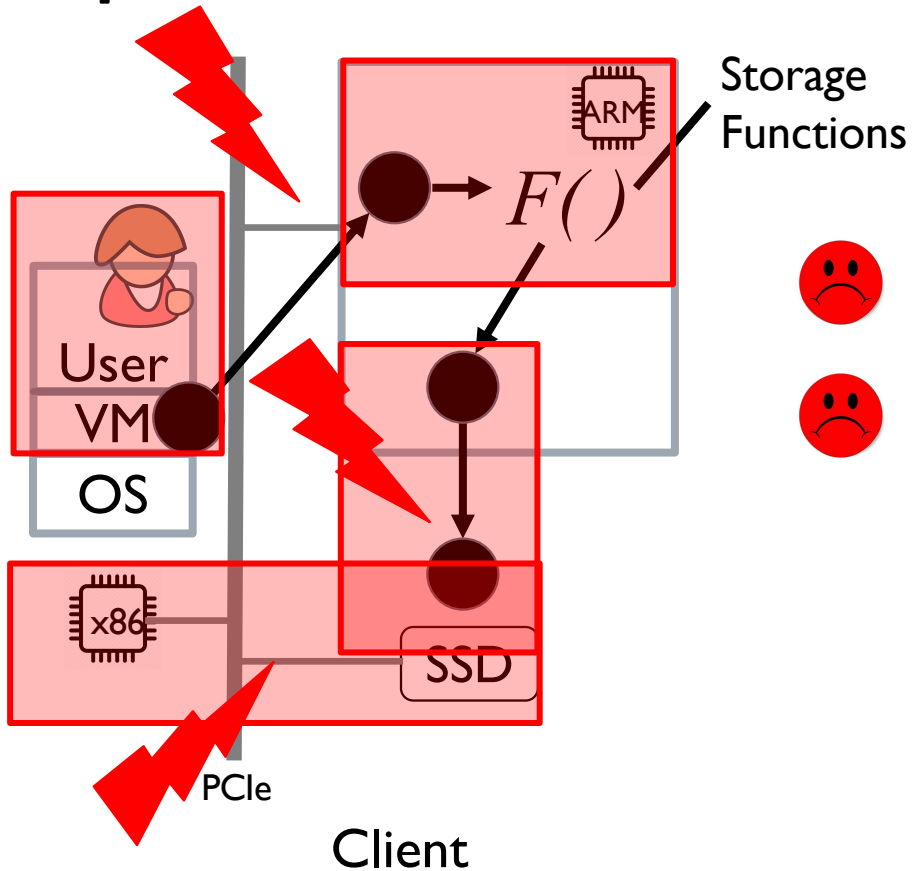


# LeapIO Efficient Data Path



No direct ARM-x86 communication

# LeapIO Efficient Data Path

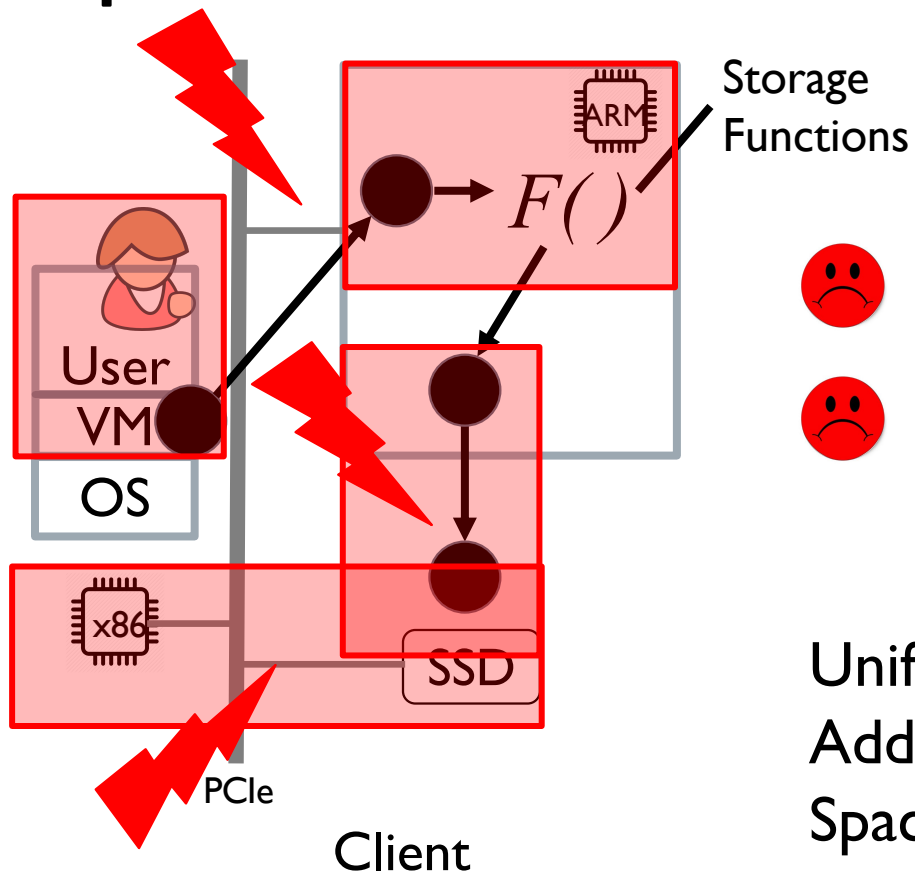


No direct ARM-x86 communication



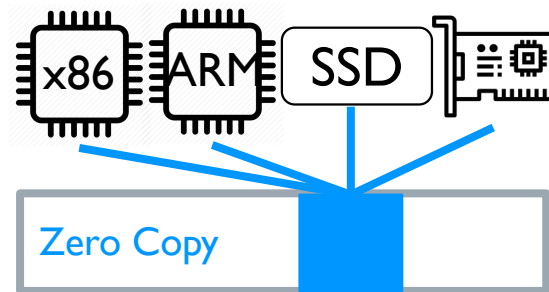
Data copies across address spaces

# LeapIO Efficient Data Path



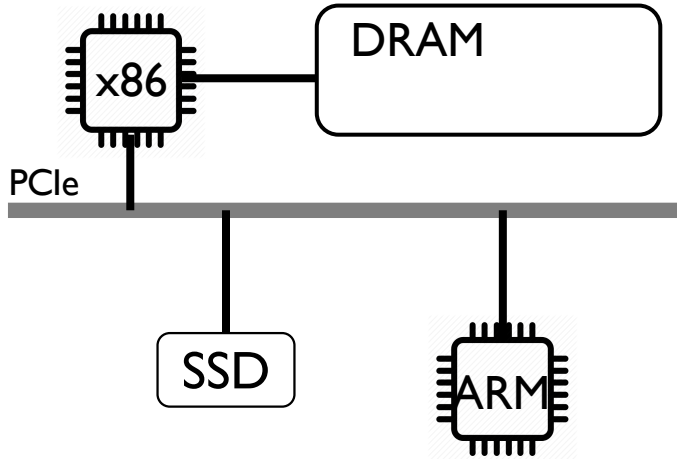
- ☹ No direct ARM-x86 communication
- ☹ Data copies across address spaces

Unified  
Address  
Space



# Existing ARM SoC Design Inefficiencies

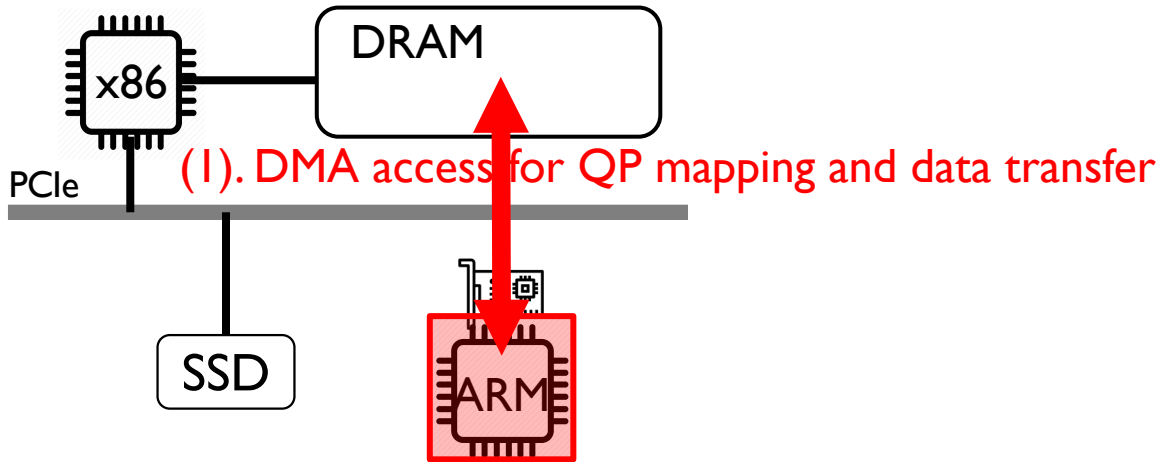
Implications for SoC Vendors



# Existing ARM SoC Design Inefficiencies

## Implications for SoC Vendors

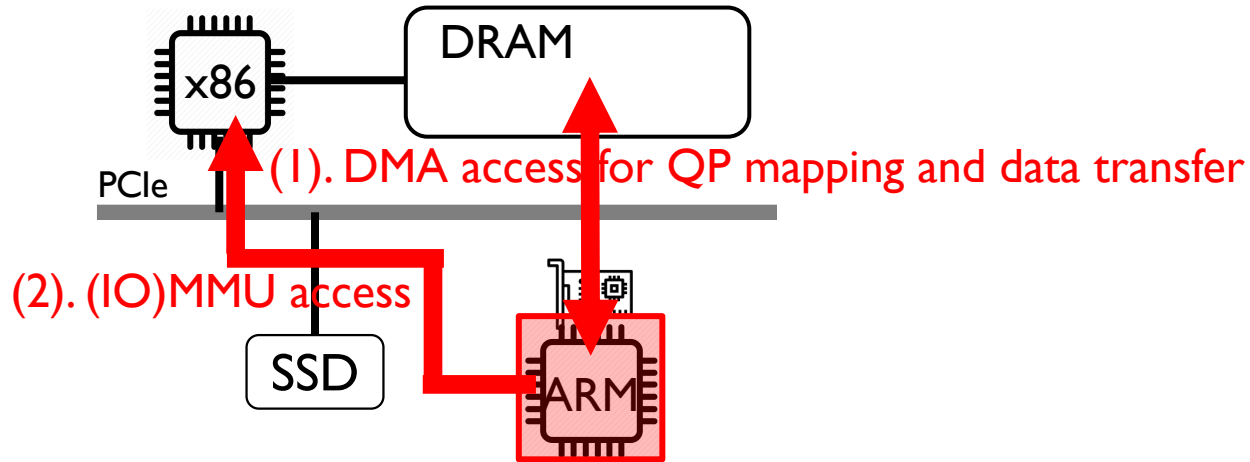
- Add Native DMA interface for ARM-x86 communication



# Existing ARM SoC Design Inefficiencies

## Implications for SoC Vendors

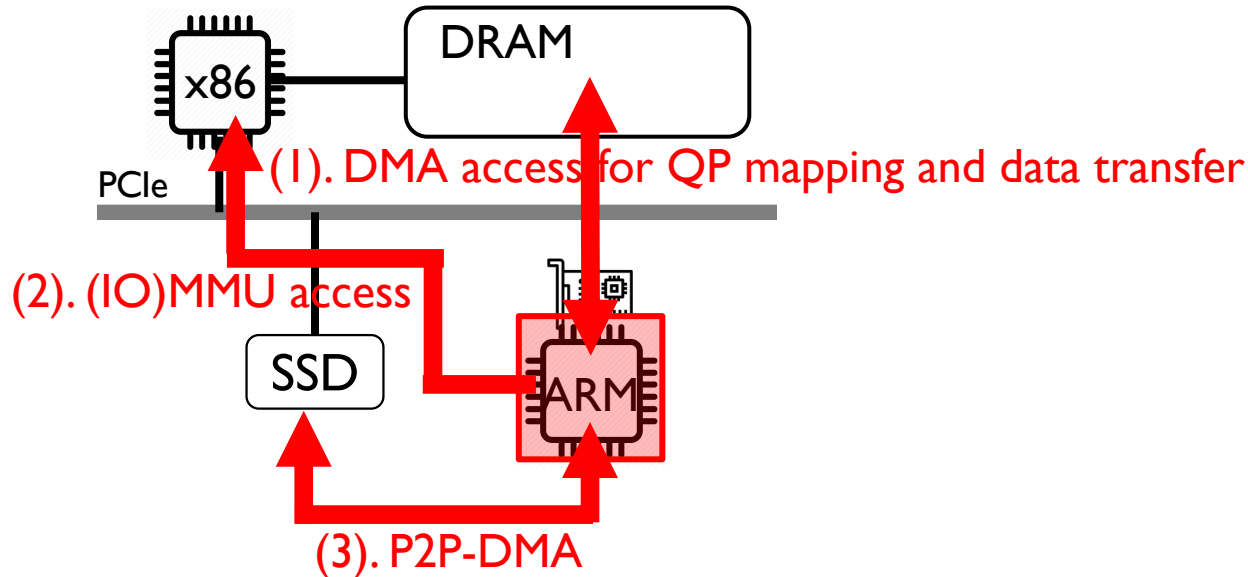
- Add Native DMA interface for ARM-x86 communication
- Enable (IO)MMU Access from ARM for Address Translation



# Existing ARM SoC Design Inefficiencies

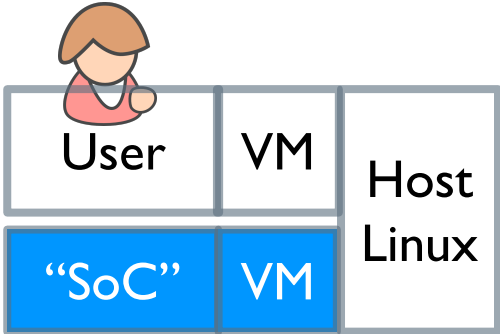
## Implications for SoC Vendors

- Add Native DMA interface for ARM-x86 communication
- Enable (IO)MMU Access from ARM for Address Translation
- Expose SoC DRAM to x86 to enable P2P-DMA

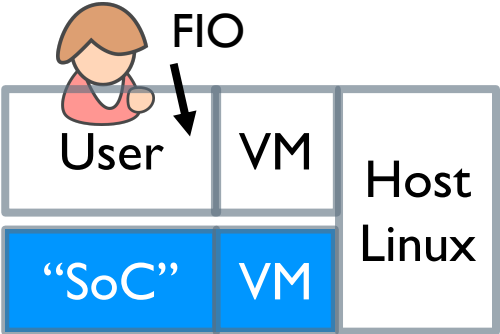




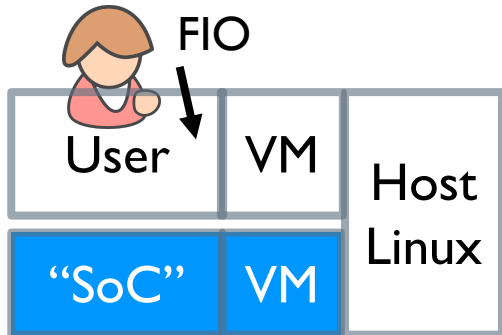
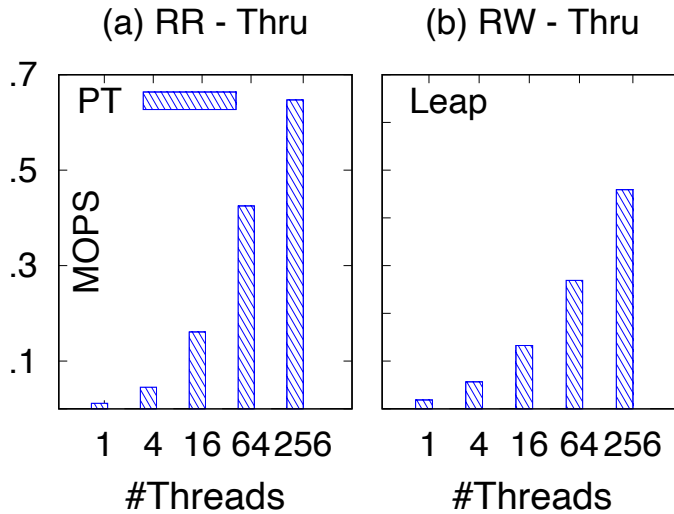
# Software Stack Overhead (“SoC”-VM)



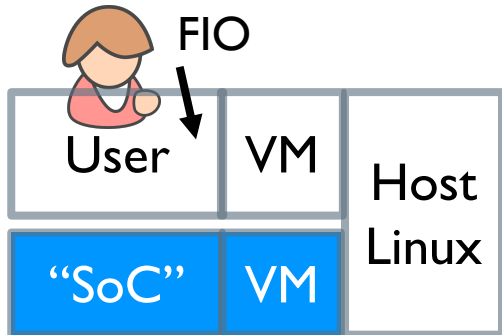
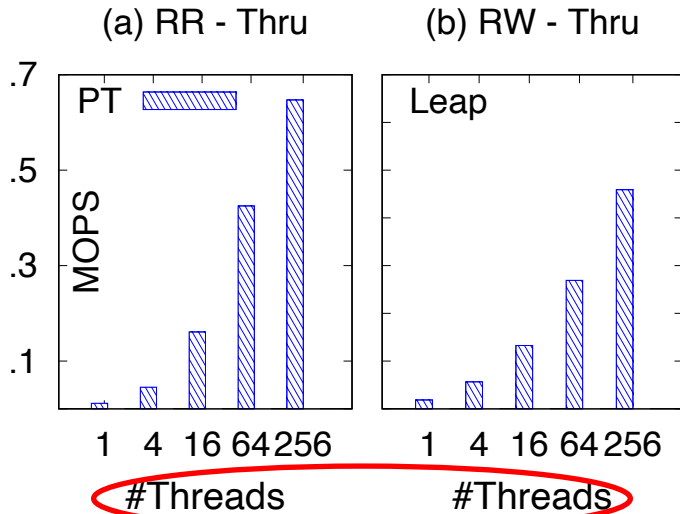
# Software Stack Overhead (“SoC”-VM)



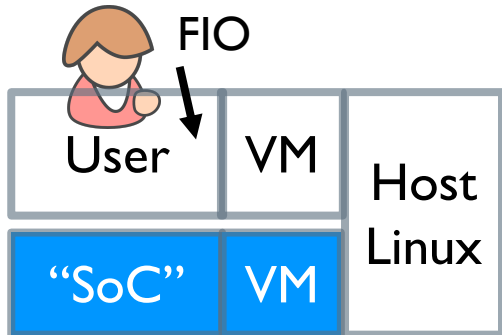
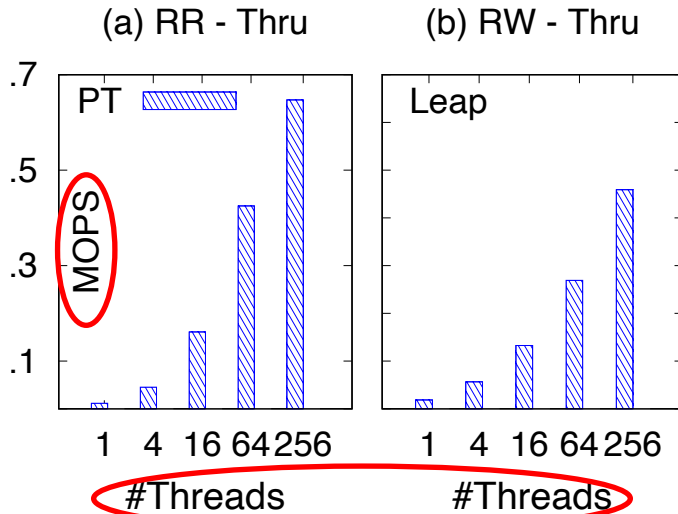
# Software Stack Overhead (“SoC”-VM)



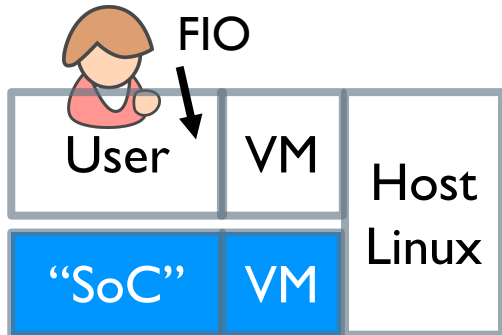
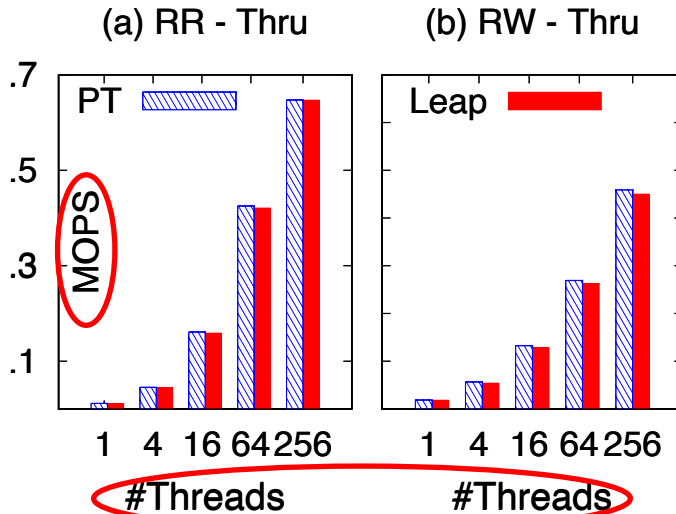
# Software Stack Overhead (“SoC”-VM)



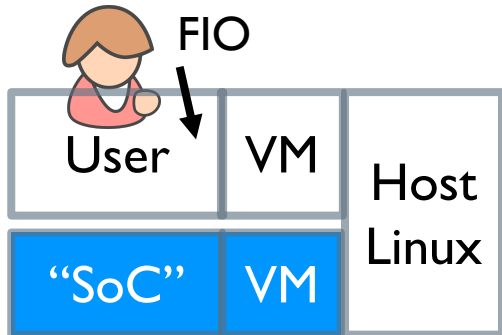
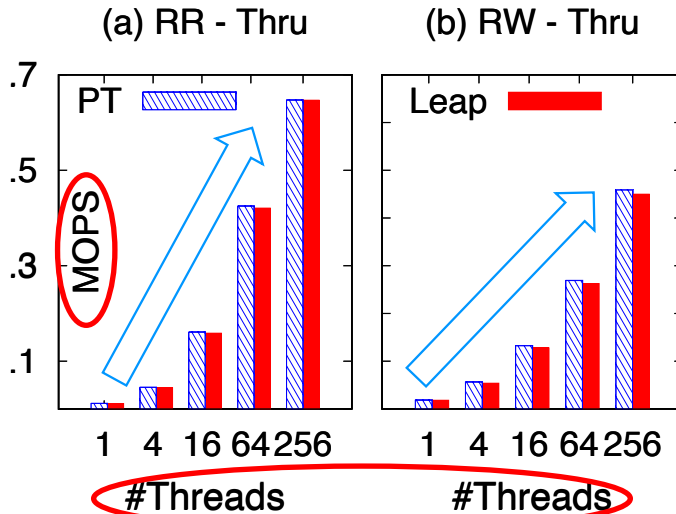
# Software Stack Overhead (“SoC”-VM)



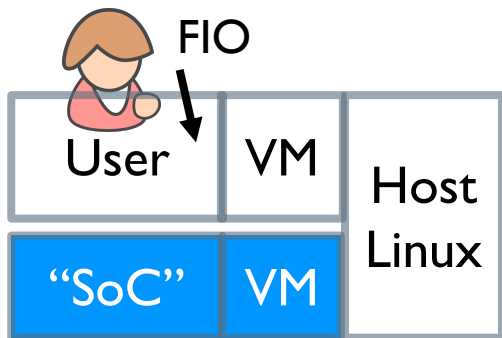
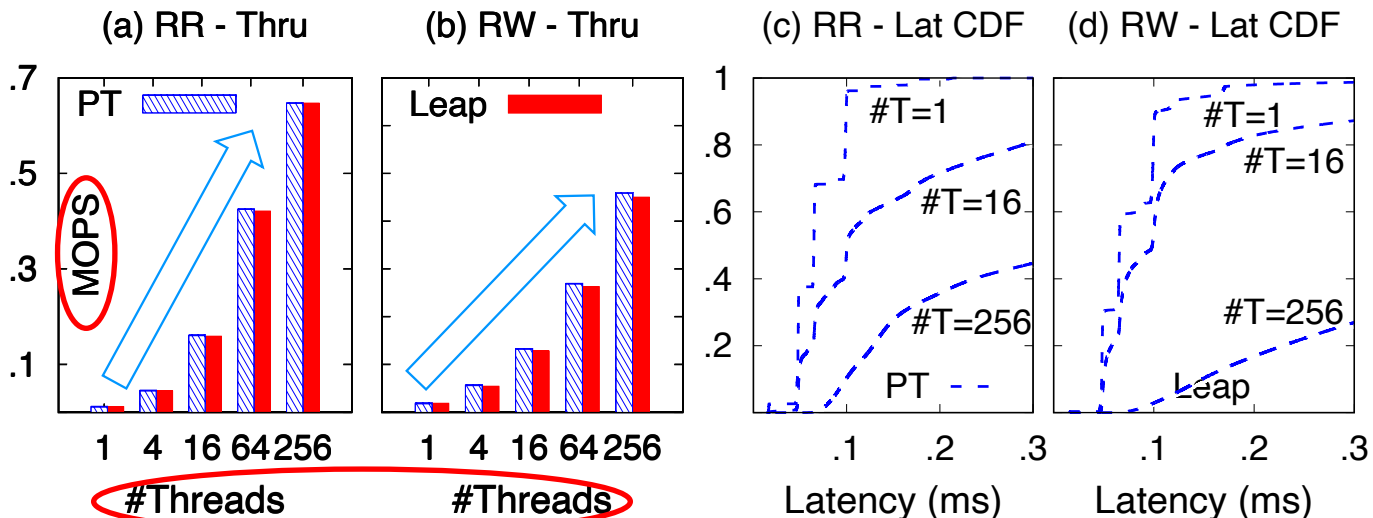
# Software Stack Overhead (“SoC”-VM)



# Software Stack Overhead (“SoC”-VM)



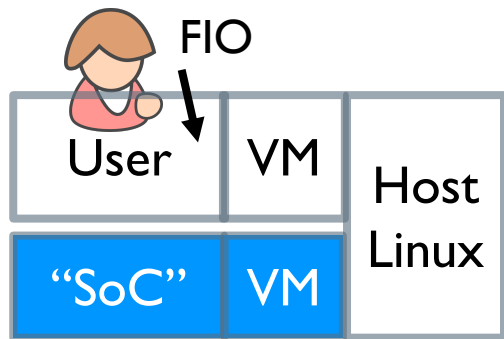
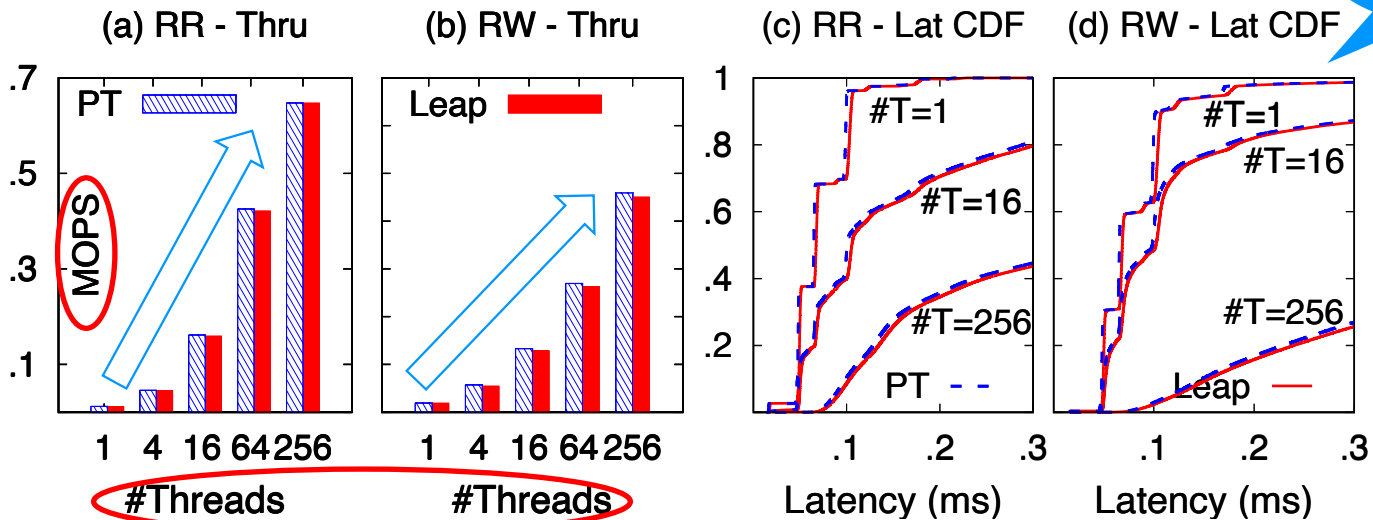
# Software Stack Overhead (“SoC”-VM)





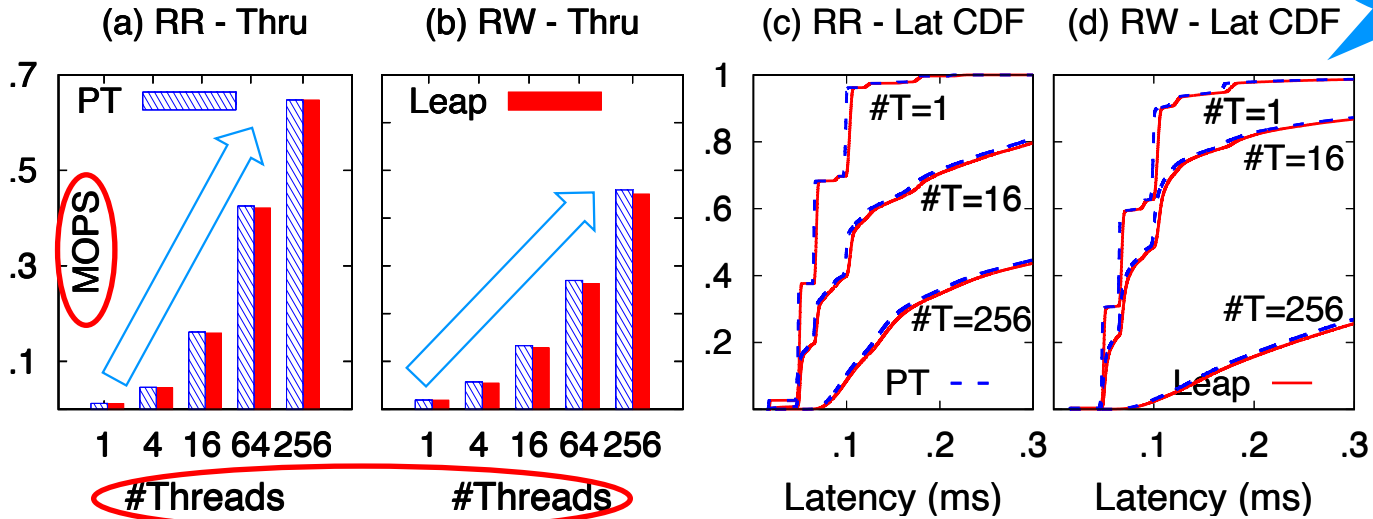
# Software Stack Overhead (“SoC”-VM)

No Tails

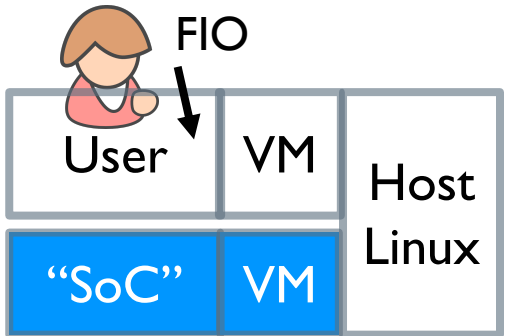


# Software Stack Overhead (“SoC”-VM)

No Tails



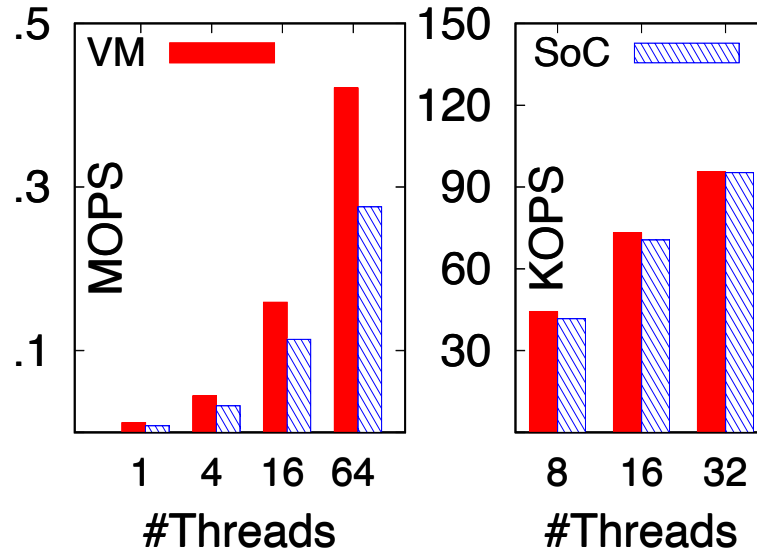
LeapIO Software Overhead: 2-5%



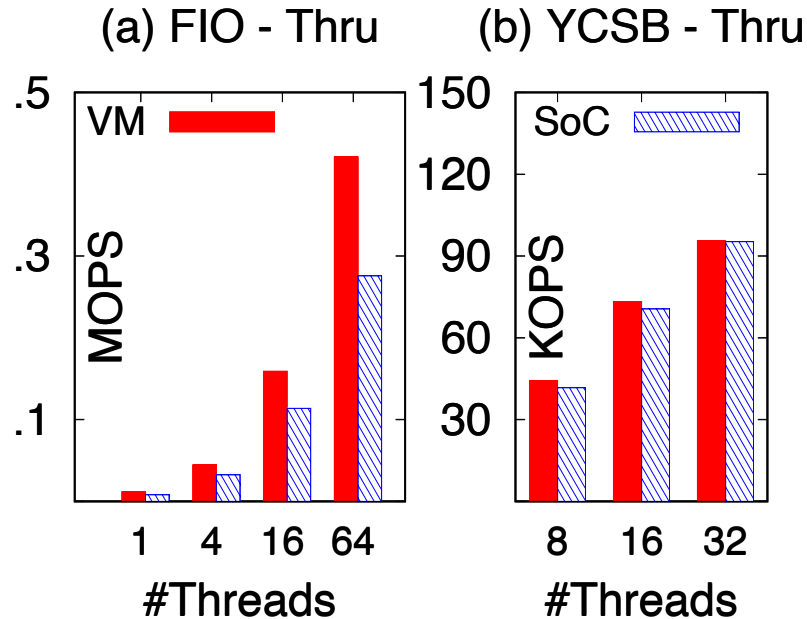
# LeapIO Performance on ARM SoC

(a) FIO - Thru

(b) YCSB - Thru



# LeapIO Performance on ARM SoC



30% Overhead due to ARM SoC Inefficiencies;  
Will improve with future ARM SoC Designs

# More in the Paper!

- ❑ LeapIO IO path and address translation
- ❑ VM-“SoC” enhancement
- ❑ Composable services
- ❑ Threat Model
- ❑ NVMe over RDMA/TCP/REST
- ❑ More evaluation results
  - Comparison with state of the art virtualization solutions
  - Service composability

# More in the Paper!

- ❑ LeapIO IO path and address translation
- ❑ VM-“SoC” enhancement
- ❑ Composable services
- ❑ Threat Model
- ❑ NVMe over RDMA/TCP/REST
- ❑ More evaluation results
  - Comparison with state of the art virtualization solutions
  - Service composability

## LeapIO: Efficient and Portable Virtual NVMe Storage on RM SoCs

Huaicheng Li, Mingzhe Hao  
University of Chicago

Sriram Govindan  
Microsoft

Stanko Novakovic  
Microsoft Research

Dan R. K. Ports, Irene Zhang,  
Ricardo Bianchini  
Microsoft Research

nirudh Badam  
Microsoft Research

Vaibhav Gogte  
University of Michigan

Haryadi S. Gunawi  
University of Chicago

### abstract

*Today's cloud storage stack is extremely resource hungry, burning 10-20% of datacenter x86 cores, a major "storage tax" that cloud providers must pay. Yet, the complex cloud storage stack is not completely offload-ready to today's IO accelerators. We present LeapIO, a new cloud storage stack that leverages RM-based co-processors to offload complex storage services. LeapIO addresses many deployment challenges, such as hardware fungibility, software portability, virtualizability, composability, and efficiency. It uses a set of OS/software techniques and new hardware properties that provide a uniform address space across the x86 and RM cores and expose virtual NVMe storage to unmodified guest VMs, at a performance that is competitive with bare-metal servers.*

**CCS Co-cepts.** • Computer systems organization → Cloud computing; Client-server architectures; System on a chip; Real-time system architecture.

**Keywords.** Data Center Storage; RM SoC; NVMe; SSD; Virtualization; Performance; Hardware Fungibility

### CM Reference Format:

Huaicheng Li, Mingzhe Hao, Stanko Novakovic, Vaibhav Gogte, Sriram Govindan, Dan R. K. Ports, Irene Zhang, Ricardo Bianchini, Haryadi S. Gunawi, and nirudh Badam. 2020. LeapIO: Efficient and Portable Virtual NVMe Storage on RM SoCs. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems (SPLoS '20)*, March 16–20, 2020, Lausanne, Switzerland. CM, New York, NY, US, 15 pages. <https://doi.org/10.1145/3373376.3378531>

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored.

Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from [permissions@acm.org](https://permissions@acm.org).

SPLoS '20, March 16–20, 2020, Lausanne, Switzerland  
© 2020 Copyright held by the owner/author(s). Publication rights licensed to CM.

CM ISBN 978-1-4503-7102-5/20/03...\$15.00  
<https://doi.org/10.1145/3373376.3378531>

### 1 Introduction

Cloud storage has improved drastically in size and speed in the last decade, with a market size expected to grow to \$88 billion by 2022 [11]. With this growth, making cloud storage efficient is paramount. On the technical side, cloud storage is facing two trends, the growing complexity of cloud drives and the rise of IO accelerators, both unfortunately have not blended to the fullest extent.

First, to satisfy customer needs, today's cloud providers must implement a wide variety of storage (drive-level) functions as listed in Table 1. Providers must support both local and remote isolated virtual drives with IOPS guarantees. Users also demand drive-level atomicity/versioning, and not to mention other performance, reliability, and space-related features (checksums, deduplication, elastic volumes, encryption, prioritization, polling for ultra-low latencies, striping, replication, etc.) that all must be composable. Last but not least, future cloud drives must support fancier interfaces [19, 24, 27, 63, 70].

As a result of these requirements, the cloud storage stack is extremely resource hungry. Our experiments suggest that the cloud provider may pay a heavy tax for storage: 10–20% of x86 cores may have to be reserved for running storage functions. Ideally, host CPU cores are better spent for providing more compute power to customer VMs.

The second trend is the increasing prevalence of IO acceleration technologies such as SmartSSDs [7, 16], SmartNICs [4, 8] and custom IO accelerators with attached computation that can offload some functionality from the host CPU and reduce the heavy tax burden. However, IO accelerators do not provide an end-to-end solution for offloading real-deployment storage stacks. Today, the storage functions in Table 1 cannot be fully accelerated in hardware for three reasons: (1) the functionalities are too complex for low-cost hardware acceleration, (2) acceleration hardware is typically designed for common-case operations but not end-to-end scenarios, or (3) the underlying accelerated functions are not composable.

# LeapIO Summary

- **End-to-end Offload-Ready** Cloud Storage Stack
  - Portability
  - Extensibility
  - Efficiency

# LeapIO Summary

- ❑ End-to-end Offload-Ready Cloud Storage Stack
  - Portability
  - Extensibility
  - Efficiency
- ❑ 20x revenue gains
- ❑ 2-5% software overhead (30% on SoC)



# LeapIO Summary

- ❑ End-to-end Offload-Ready Cloud Storage Stack
  - Portability
  - Extensibility
  - Efficiency
- ❑ 20x revenue gains
- ❑ 2-5% software overhead (30% on SoC)
- ❑ Implications for SoC vendors to bridge the performance gap between ARM and x86

# LeapIO Summary

- ❑ End-to-end Offload-Ready Cloud Storage Stack
  - Portability
  - Extensibility
  - Efficiency
- ❑ 20x revenue gains
- ❑ 2-5% software overhead (30% on SoC)
- ❑ Implications for SoC vendors to bridge the performance gap between ARM and x86

Thank you!  
Questions?