

CSE 291 Virtualization and Cloud Computing: Course Summary

Yiying Zhang

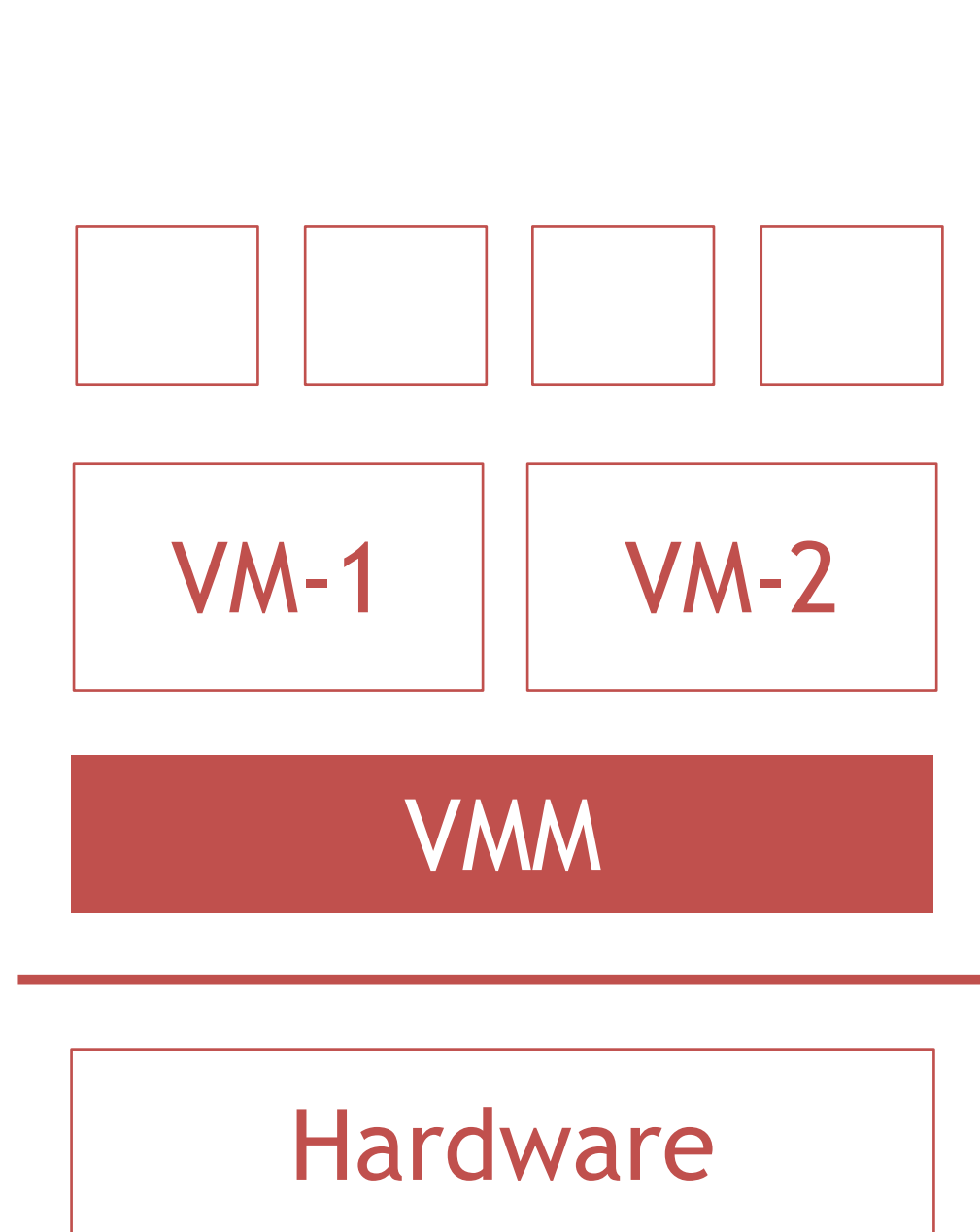
Outline

- Virtualization and cloud computing summary
- Hints for computer systems design
- Quiz 3
- Final project presentation this Wed
 - (7-8min presentation + 2-3min Q&A) * 8 groups
- Final project summary report due 6/13. No extension!
- Course evaluation!

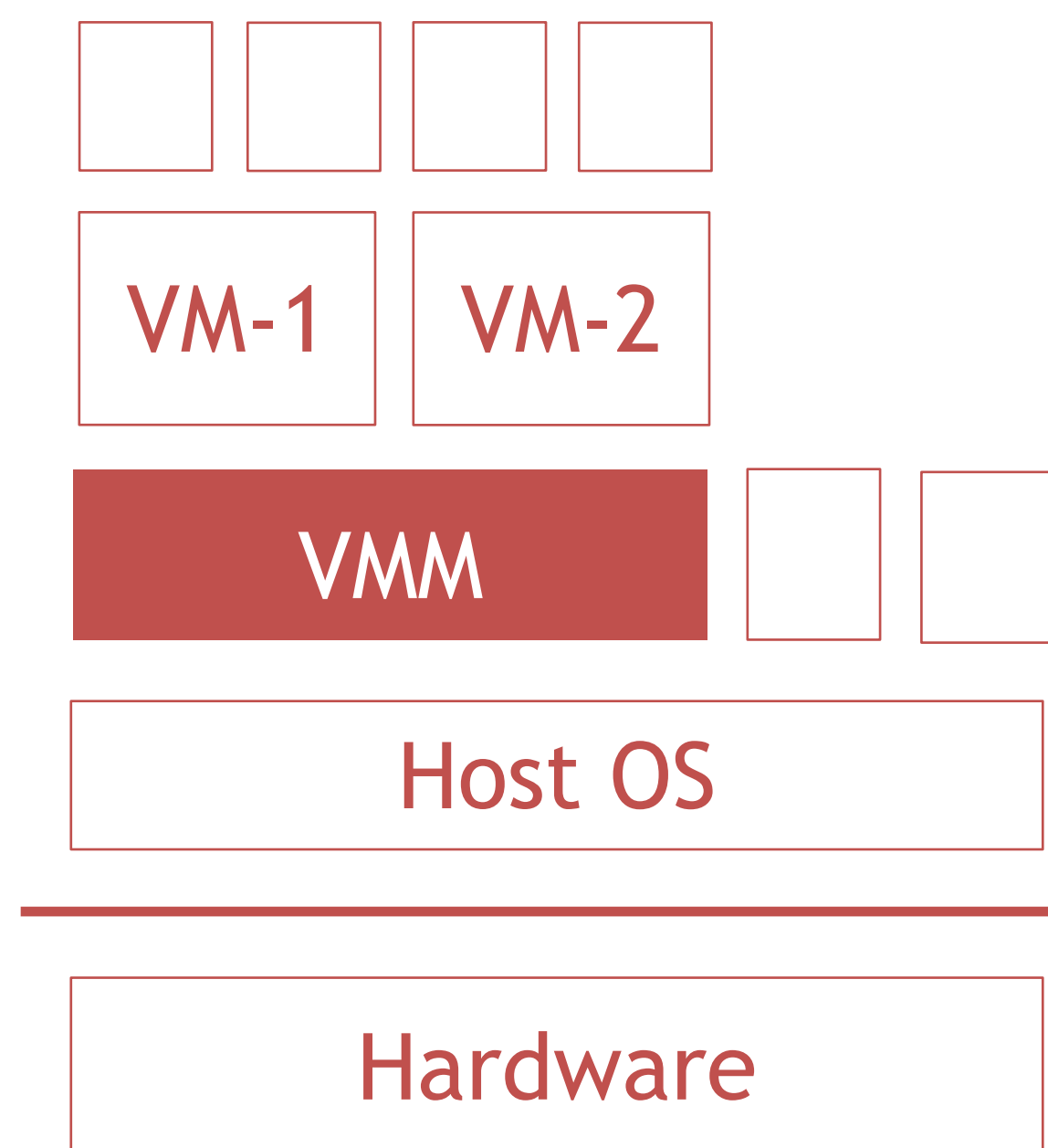
Virtualization Approaches

- Hosted interpretation
 - Interpret each instruction, super slow (e.g., Virtual PC on Mac)
- Direct execution with trap-and-emulate
 - Requires a virtualizable processor and only works for the same architecture
- Direct execution with binary translation
 - Works with non-virtualizable processor, but implementing VMM is tricky
- Direct execution with hardware-assisted virtualization
 - Needs new generation of hardware (which is the norm now), mode switching is still not optimized
- Direct execution with paravirtualization
 - Good performance and works with non-virtualizable processors, but require guest OS changes
- OS-level virtualization, library-level, language (app)-level, unikernels, etc.
 - More lightweight and faster to start, but less secure

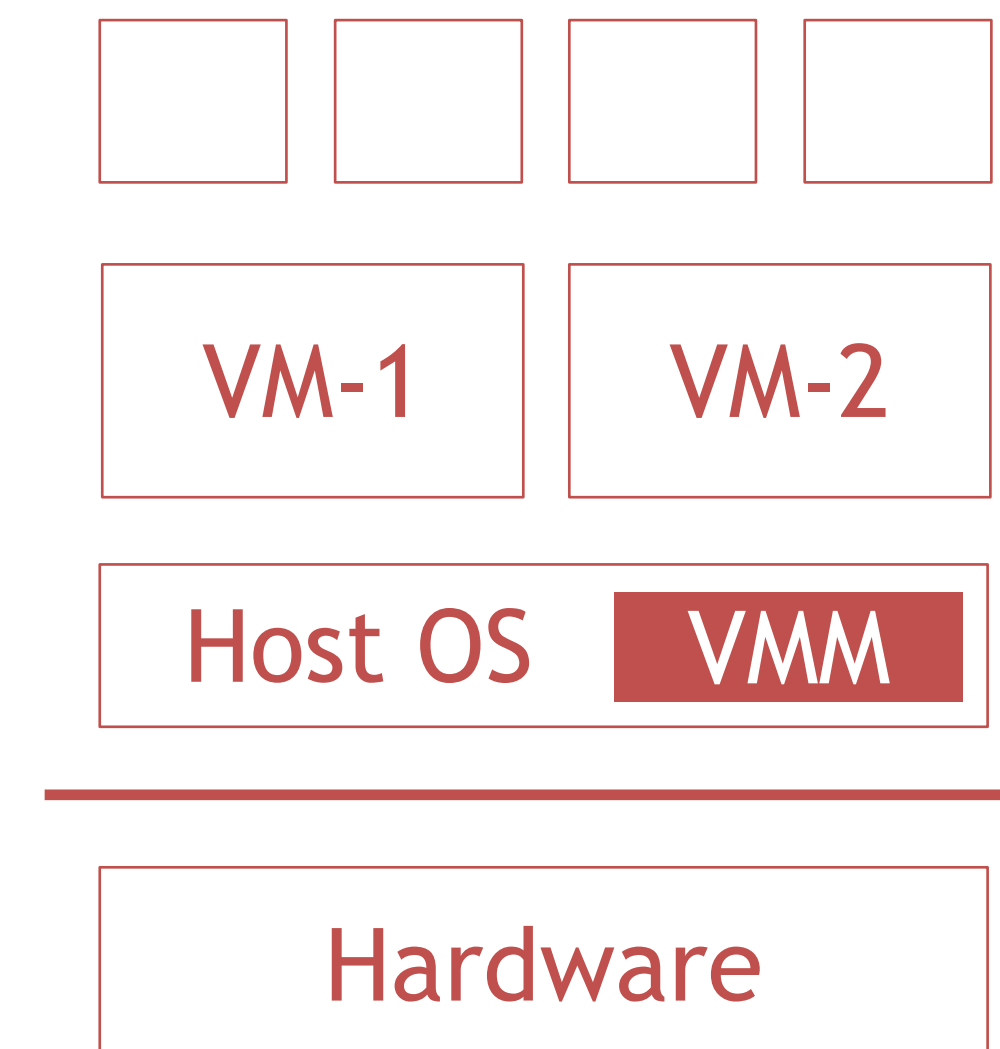
Virtual Machine Architectures



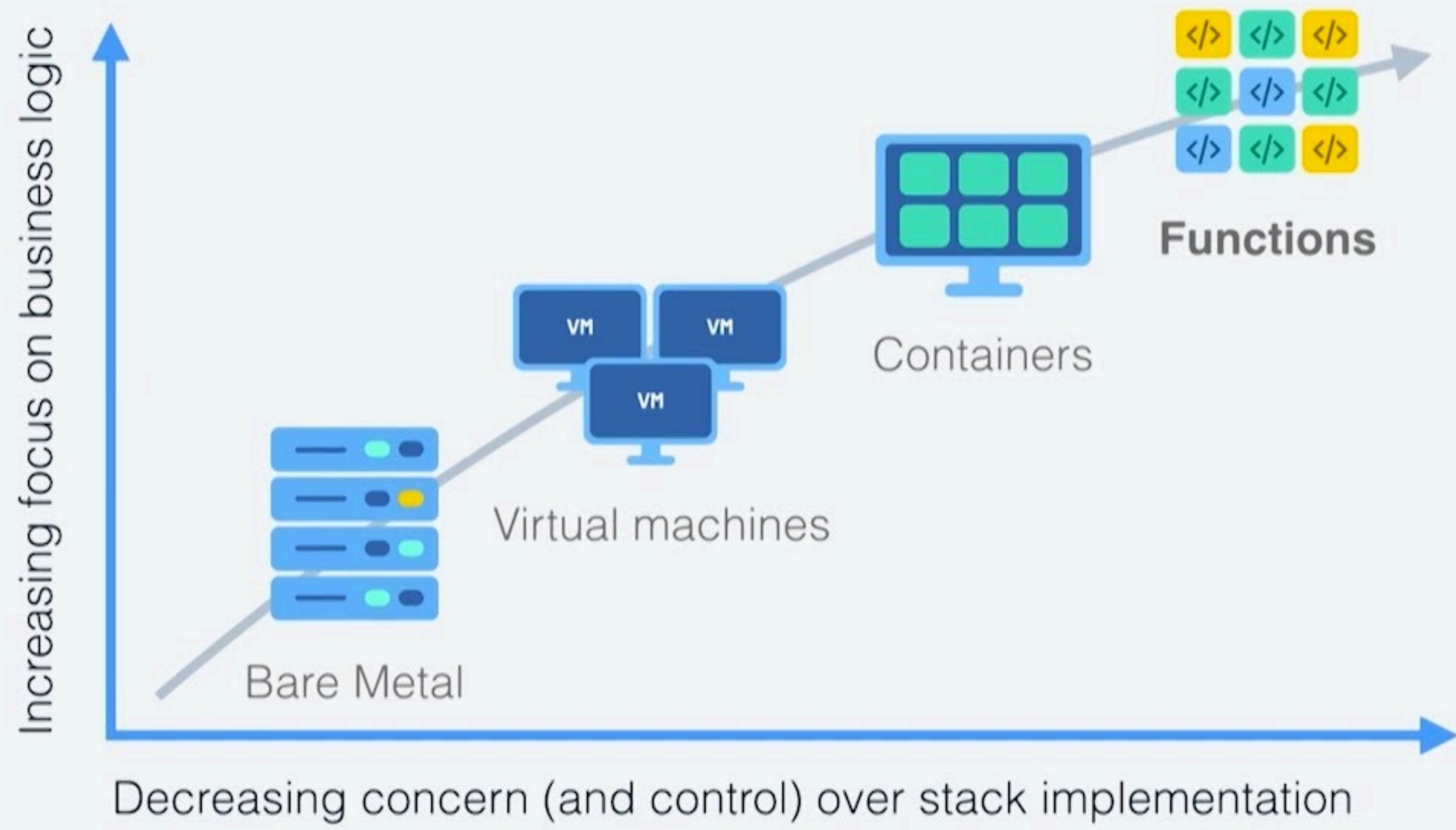
Xen / VMware ESX



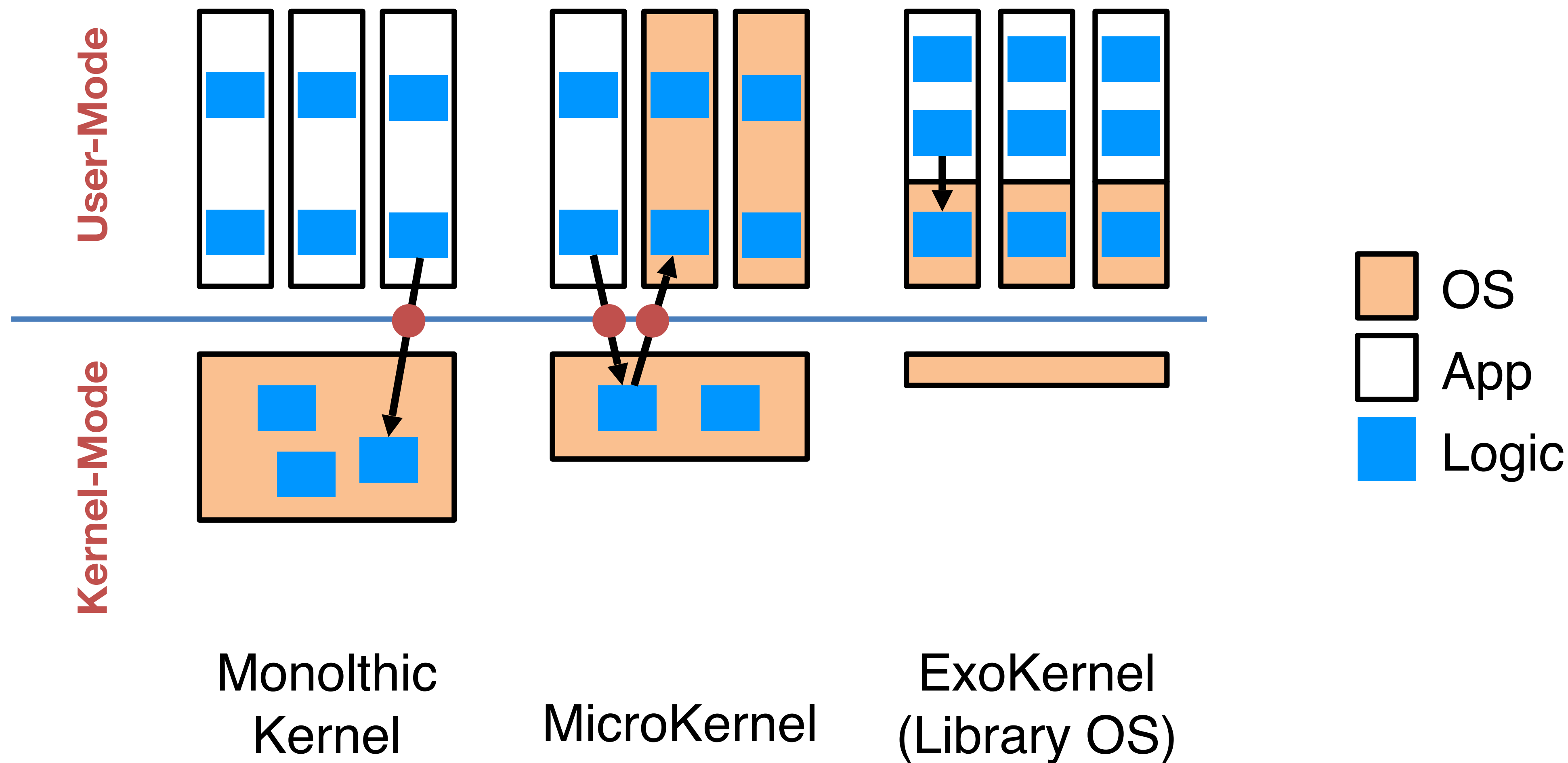
**VMware Workstation
/ VirtualBox**



Linux KVM



Operating System Architectures

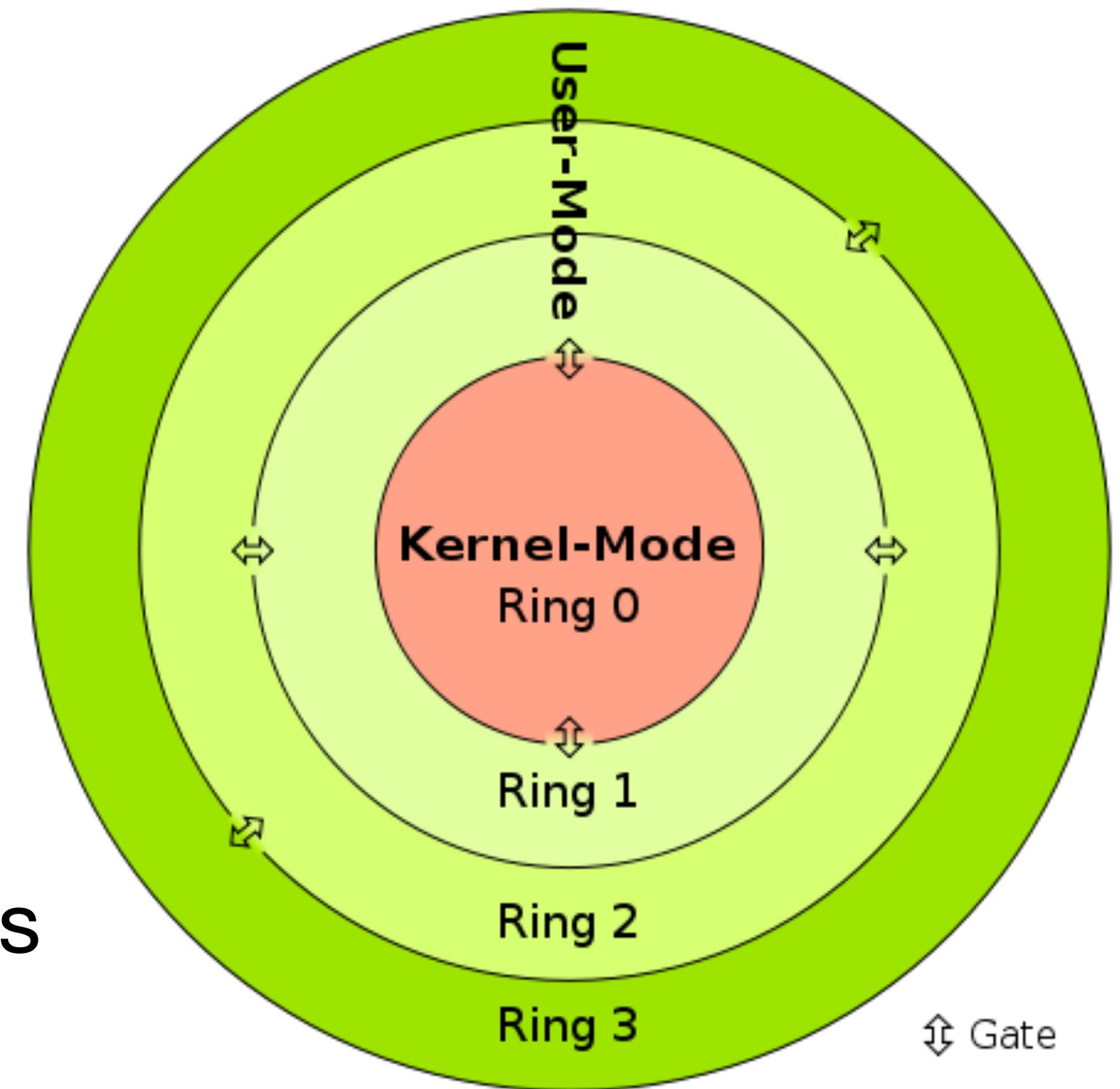


X86 Difficulties and Possible Solutions

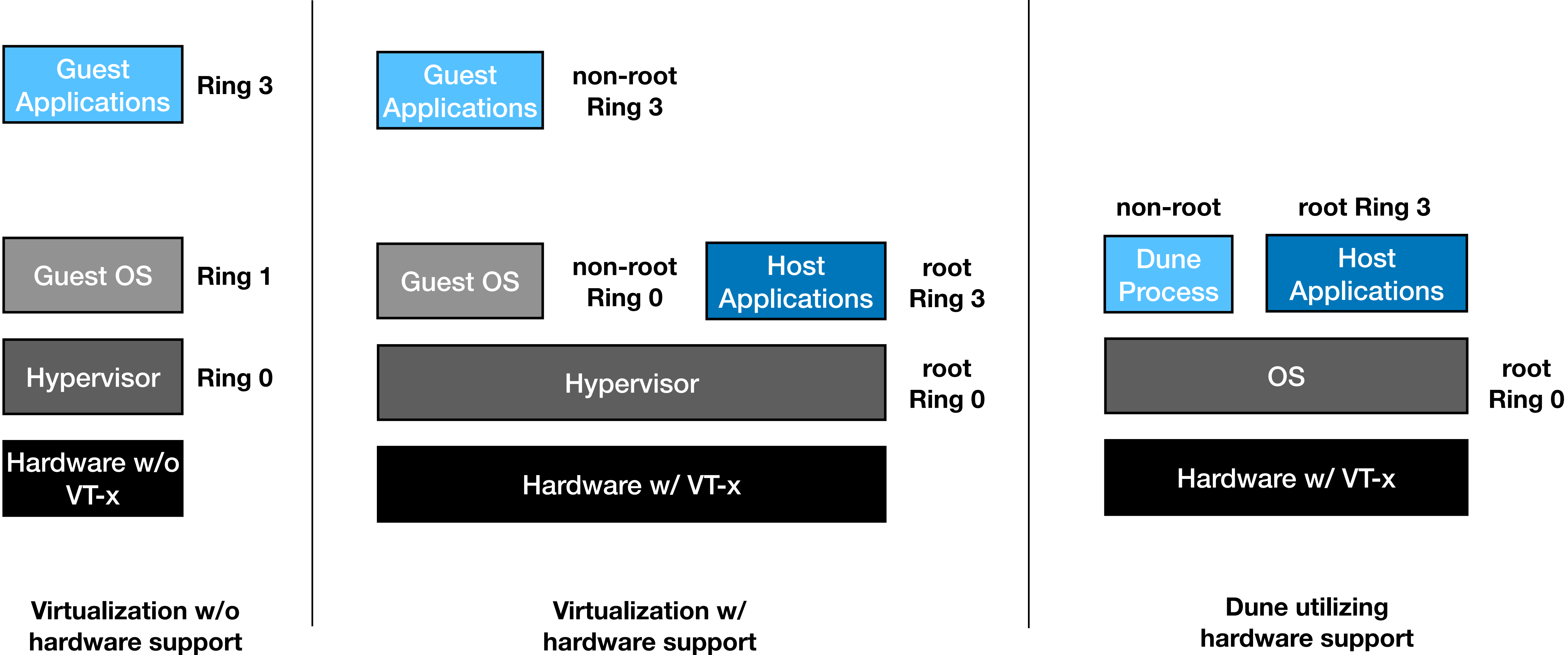
- Not all sensitive instructions are privileged with x86, i.e., non-virtualizable processor
 - These instructions do not trap and behave differently in kernel and user mode
- Hardware-managed TLB
- Emulate
 - Interpret each instruction, super slow (e.g., Virtual PC on Mac)
- Binary translation
 - Rewrite non-virtualizable instructions (e.g., VMware)
- Para-virtualization
 - Modify guest OS to avoid non-virtualizable instructions (e.g., Xen)
- Change hardware
 - Add new CPU mode, extend page table, and other hardware assistance (e.g., Intel VT-x, EPT, VT-d, AMD-V)

Protection Rings

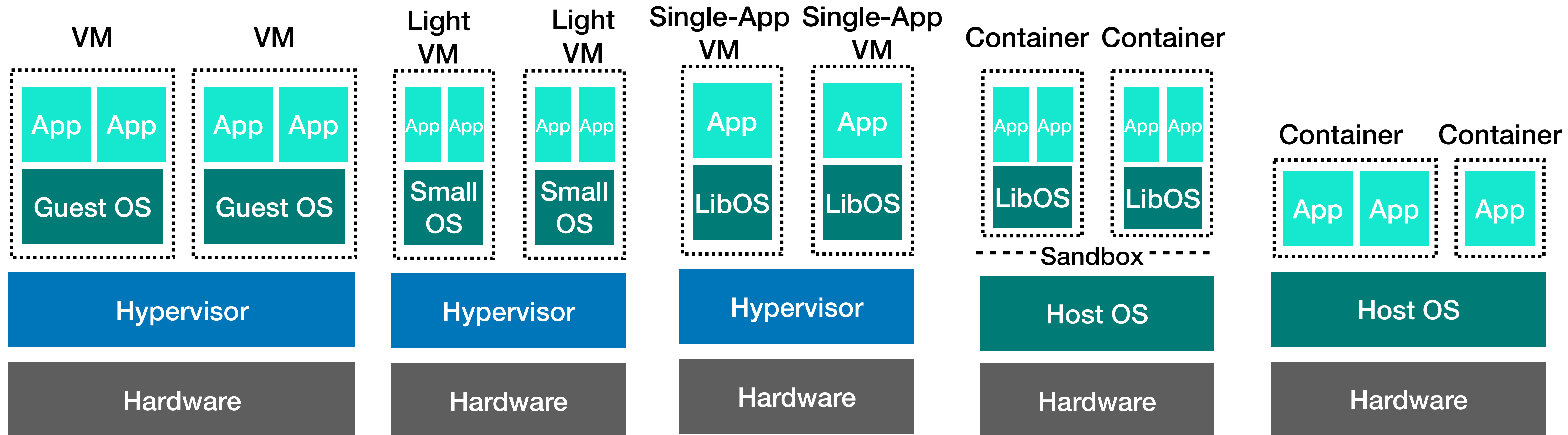
- More privileged rings can access memory of less privileged ones
- Calling across rings can only happen with hardware enforcement
- Only Ring 0 can execute privileged instructions
- Rings 1, 2, and 3 trap when executing privileged instructions
- Usually, the OS executes in Ring 0 and applications execute in Ring 3



Hardware Virtualization Support



VM <—> Containers



Virtual Machines

- + Strong security
- + Fully compatible
- Heavy weight

Lightweight VM (LightVM, Firecracker)

- + Strong security
- + Mostly compatible
- Medium weight

Unikernels

- + Strong security
- Not compatible
- Light weight

Secure Container (gVisor, Unikernels as processes)

- + Strong security
- + Mostly compatible
- Light weight

Containers

- Weak security
- + Compatible
- Lightest weight

Major Clouds and Datacenters

AWS

- Biggest market share, longest history
- Highest compute (and other service) options
 - >= 136 instance types in 26 families
- Storage
 - Simple Storage Service (S3)
 - Elastic File Service (EFS) / Elastic Block Service (EBS)
- Many other services
 - Lambda (serverless)
 - ECS/EKS (managed containers)
 - ...

Amazon

- Storage
 - Dynamo, S3, EFS, EBS
- Database/NoSQL
 - DynamoDB, Redshift, ElastiCache
- Network
 - Customized NICs, virtualization support
- Hardware
 - ASIC (Nitro), x86, ARM
- Resource management
 - Fargate, Kubernetes
- Execution environment
 - Disaggregated storage, many virtualization options
- Dataflow/analytics
 - EMR, Athena
- Application
 - Neptune (graph), SageMaker (ML), Kafka (streaming)

Availability

ASIC

Azure

- Good integration with Microsoft products
 - Customers that are already using Microsoft products (e.g., having existing licenses)
- Many instance types and service types
- Moved from Windows to Linux

Microsoft

- Storage

- Azure storage (erasure coding), Project Silica

- Database

- SQL Server

- Network

- RDMA, FaRM

- Hardware

- x86, FPGA (Catapult)

- Resource management

- Some research in using ML

- Execution environment

- Disaggregated storage

- Dataflow

- Dyrad, DyradLINQ

- Application

- Project Adam (ML)

Enterprise

FPGA

Google Cloud Platform (GCP)

- Latest among the three to come in play and smallest market share, but with good growth
- Cheapest among the three
- Fewest instance types, allows customized CPU/memory sizes
 - bill based on total CPU and memory usages, not on total instance time
- Native kubernetes support
- Good support for cross geo-regions

Google

- Storage
 - GFS, next-gen GFS, Intel Optane
- Database
 - BigTable, Spanner (Geo)
- Network
 - FatTree
- Hardware
 - Commodity + TPU
- Resource management
 - Borg, Kubernetes, Chubby
- Execution environment
 - Non-disaggregated, containerized
- Dataflow
 - MapReduce
- Application
 - TensorFlow (ML), Pregel (graph)

Reliability

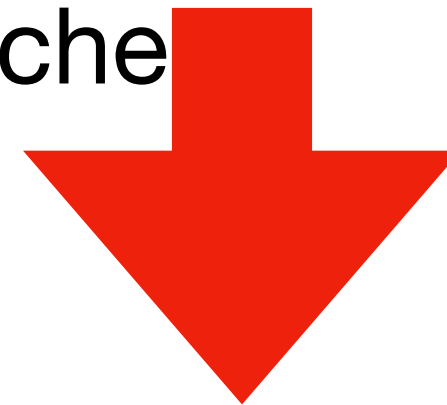
Commodity+TPU

Meta (Facebook)

- Storage
 - TAO, Haystack (photo)
- Resource management
 - Resource control

• Database/**Move Fast and Break Things**ht

- Presto, MySQL, Cassandra, Memcache
- Disaggregated (pods)
- Network
 - Dataflow



• Taiji (traffic)
• Hardware**Move Fast with Stable Infra**

- Application

• Commodity, some ASIC**Open Source** SVE (video), PyTorch (ML)

Common (and Important) Themes

- Scalability
 - Dist sys, local sys, networking
- Reliability
 - Failure, bugs, testing
- Security
 - Across customers, internal, regulations
- Resource Efficiency
- Manageability
- Open source

Outline

- Virtualization summary
- Hints for computer systems design
- Quiz 3

Systems Design

- The external interface (that is, the requirement) is less precisely defined, more complex, and more subject to change.
- The system has much more internal structure, and hence many internal interfaces.
- The measure of success is much less clear.

Why?	Functionality Does it work?	Speed Is it fast enough?	Fault-tolerance Does it keep working?
Where?			
Completeness	Separate normal and worst case	Shed load End-to-end Safety first	End-to-end
Interface	Do one thing well: Don't generalize Get it right Don't hide power Use procedure arguments Leave it to the client Keep basic interfaces stable Keep a place to stand	Make it fast Split resources Static analysis Dynamic translation	End-to-end Log updates Make actions atomic
Implementation	Plan to throw one away Keep secrets Use a good idea again Divide and conquer	Cache answers Use hints Use brute force Compute in background Batch processing	Make actions atomic Use hints

Figure 1: Summary of the slogans

Keep It Simple

- *KISS: Keep It Simple, Stupid.* (Anonymous)
- *If in doubt, leave it out.* (Anonymous)
- *Exterminate features.* (C. Thacker)
- On the other hand,
- *Everything should be made as simple as possible, but no simpler.* (A. Einstein)

Making Implementation WORK!

- *Perfection must be reached by degrees; she requires the slow hand of time.* (Voltaire)
- *Plan to throw one away; you will anyhow*
- Use a good idea (and implementation) again instead of generalizing

Continuity

- Tension between desire to improve a design and need for stability
- Backward and forward compatibility
- Never break expectation of user space
- Keep a place to stand incase of interface change

Handling Faults and Corner Cases

- Not just avg, look at the tails
- End-to-end error recovery
- Error handling in distributed systems
- Consistency, consensus, atomicity, replication, checkpointing, etc. => take diet sys course

Performance

- Use locality: caching, prefetching, etc.
 - Use parallelism and asynchronous execution
 - Use hints (application semantics)
 - Co-design different layers
 - Many other techniques
-
- Performance is not the only thing: fairness, perf/\$, SLA/SLO, ease of management, etc.

Final Thoughts?