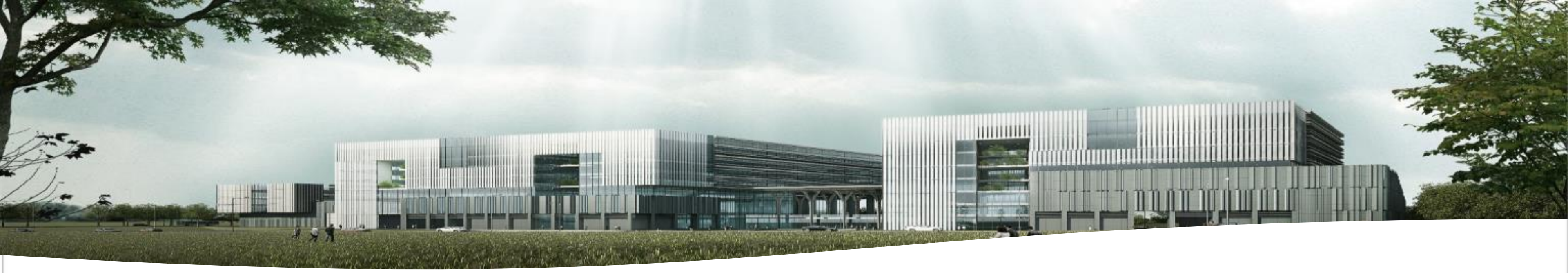


Datacenter Technology (Fall, 2018)

Software-Defined Data Center

Lingfang Zeng (曾令仿)

Wuhan National Laboratory for Optoelectronics (WNLO)

Huazhong University of Science and Technology (HUST)

https://lingfangzeng.github.io/

Outline

SDN

SOFTWARE DEFINED NETWORKS

UOS

UBIQUITOUS OPERATING SYSTEMS

SDFS

SOFTWARE-DEFINED FILE SYSTEM

Thanks

- Scott Shenker. *The Future of Networking, and the Past of Protocols*. Fall 2011.
- Hong Mei, Yao Guo. *Toward Ubiquitous Operating Systems: A Software-Defined Perspective*. IEEE Computer 51(1): 50-56 (2018)
- Jiahao Liu, Fang Wang, Lingfang Zeng, Dan Feng, Tingwei Zhu. *SDFS: A Software-Defined File System for Multi-Tenant Cloud Storage*. Software: Practice and Experience. (Accepted, 2018) (CCF B)

01

SDN

SOFTWARE DEFINED NETWORKS

Key to Internet Success: Layers

Applications

…built on…

Reliable (or unreliable) transport

…built on…

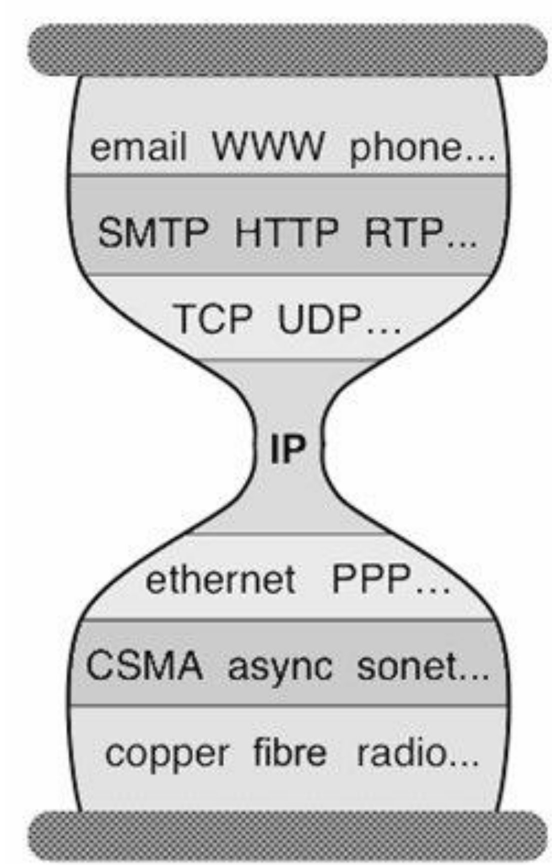
Best-effort global packet delivery

…built on…

Best-effort local packet delivery

…built on…

Physical transfer of bits



Why Is Layering So Important?

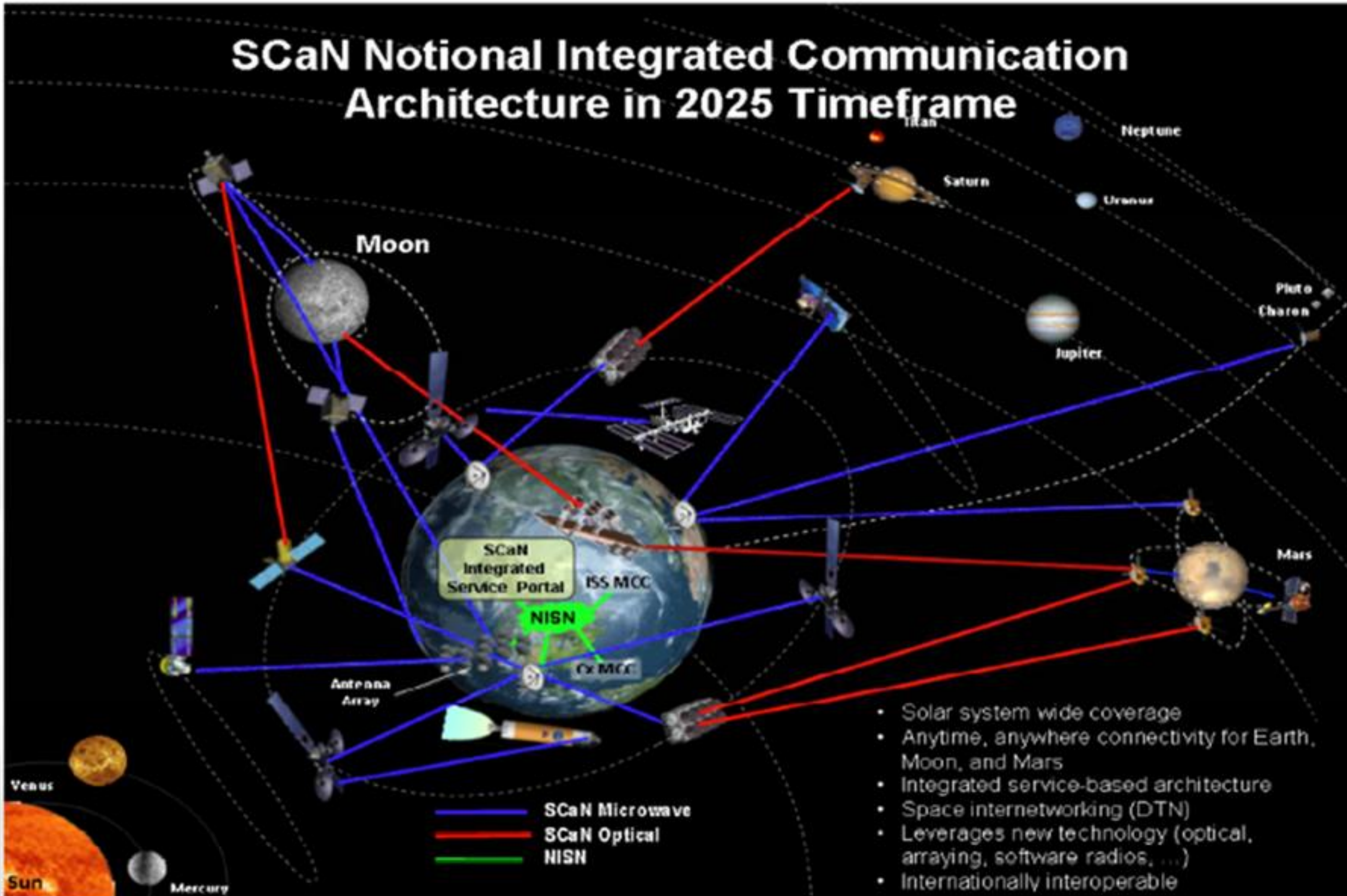
- Decomposed delivery into fundamental components
- Independent but compatible innovation at each layer
- A practical success of unprecedented proportions…
- …but an academic failure

Built an Artifact, Not a Discipline

- Other fields in "systems": OS, DB, DS, etc.
  - Teach basic principles
  - Are easily managed
  - Continue to evolve
- Networking
  - Teach big bag of protocols
  - Notoriously difficult to manage
  - Evolves very slowly

Why Does Networking Lag Behind?

- Networks used to be simple: Ethernet, IP, TCP….
- New **control** requirements led to great complexity
  - Isolation → VLANs, ACLs
  - Traffic engineering → MPLS, ECMP, Weights
  - Packet processing → Firewalls, NATs, middleboxes
  - Analysis → Deep packet inspection (DPI)
- Mechanisms designed and deployed independently
  - Complicated “control plane” design, primitive functionality
  - Stark contrast to the elegantly modular “data plane”


SCaN Notional Integrated Communication Architecture in 2025 Timeframe
Titan
Neptune
Saturn
Uranus
Moon
Pluto
Charon
Jupiter
Mars
SCaN Integrated Service Portal
ISS MCC
NISN
CxMCC
Antenna Array
Venus
Sun
Mercury
SCaN Microwave
SCaN Optical
NISN
Solar system wide coverage
Anytime, anywhere connectivity for Earth, Moon, and Mars
Integrated service-based architecture
Space internetworking (DTN)
Leverages new technology (optical, arraying, software radios, ...)
Internationally interoperable

Layers are Great Abstractions

- Layers only deal with the data plane
- We have no powerful control plane abstractions!
- How do we find those control plane abstractions?
- Two steps: define problem, and then decompose it.

The Network Control Problem

- Compute the configuration of each physical device
  - E.g., Forwarding tables, ACLs,…
- Operate without communication guarantees
- Operate within given network-level protocol

Only people who love complexity would find this a reasonable request

From Requirements to Abstractions

- Operate without communication guarantees
  - Need an abstraction for distributed state
- Compute the configuration of each physical device
  - Need an abstraction that simplifies configuration
- Operate within given network-level protocol
  - Need an abstraction for general forwarding model

Once these abstractions are in place, control mechanism has a much easier job!

(1) Distributed State Abstraction

- Shield control mechanisms from state distribution
  - While allowing access to this state
- Natural abstraction: global network view
  - Annotated network graph provided through an API
- Implemented with “Network Operating System”
  - Control mechanism is now program using API
- No longer a distributed protocol, now just a graph algorithm
  - E.g. Use Dijkstra rather than Bellman-Ford

Software Defined Network (SDN)

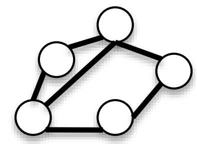
Networks of Switches and/or Routers

Traditional Control Mechanisms

e.g. routing, access control

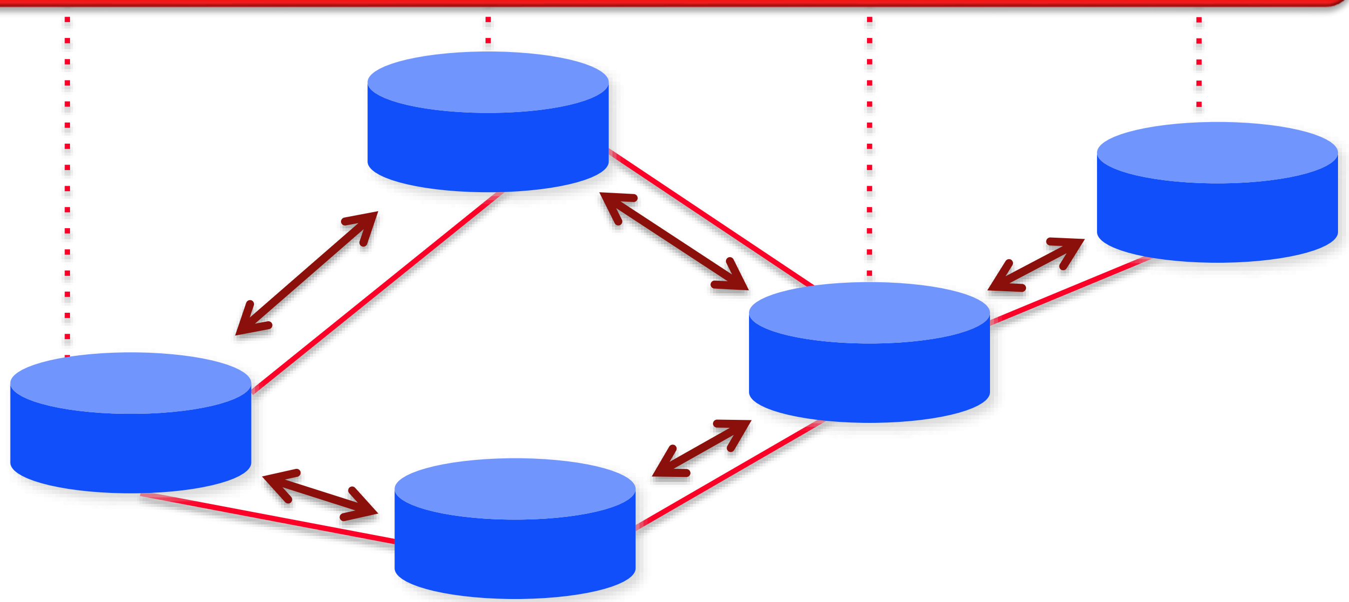
Control Program

Global Network View



Distributed algorithm running between neighbors

Network OS



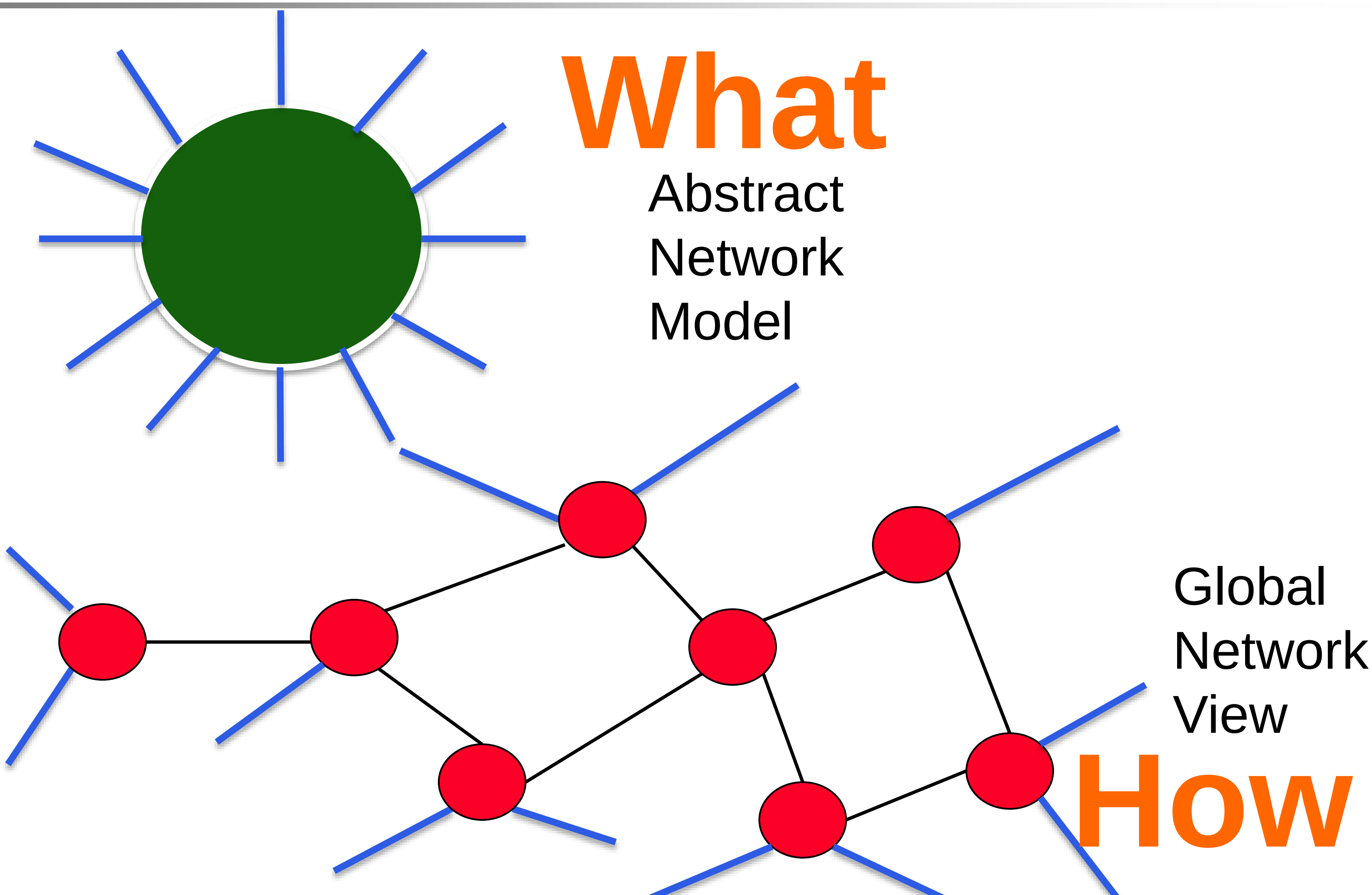
Major Change in Paradigm

- No longer designing distributed control protocols
- Design one distributed system (NOS)
- Use for all control functions
- Now just defining a centralized control function
- Configuration = Function(view)

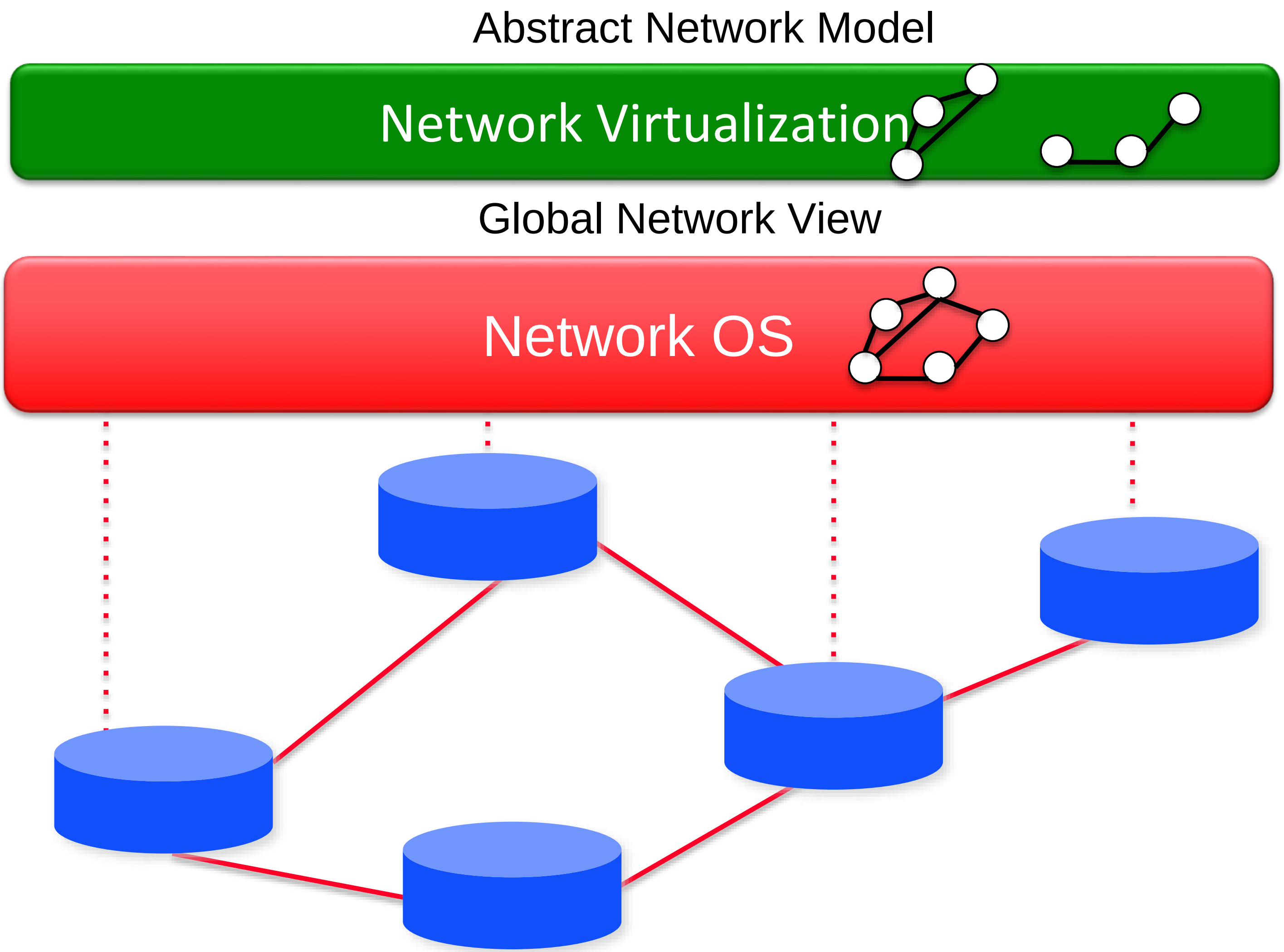
(2) Specification Abstraction

- Control program should express desired behavior
- It should not be responsible for implementing that behavior on physical network infrastructure
- Natural abstraction: simplified model of network
  - Simple model with only enough detail to specify goals
- Requires a new shared control layer
  - Map abstract configuration to physical configuration
- This is “network virtualization”

Simple Example: Access Control



Software Defined Network: Take 2



What Does This Picture Mean?

- Write a simple program to configure a simple model
  - Configuration merely a way to specify what you want
- Examples
  - ACLs: who can talk to who
  - Isolation: who can hear my broadcasts
  - Routing: only specify routing to the degree you care
    - Some flows over satellite, others over landline
  - TE: specify in terms of quality of service, not routes
- Virtualization layer "compiles" these requirements
  - Produces suitable configuration of actual network devices
- NOS then transmits these settings to physical boxes

Software Defined Network: Take 2

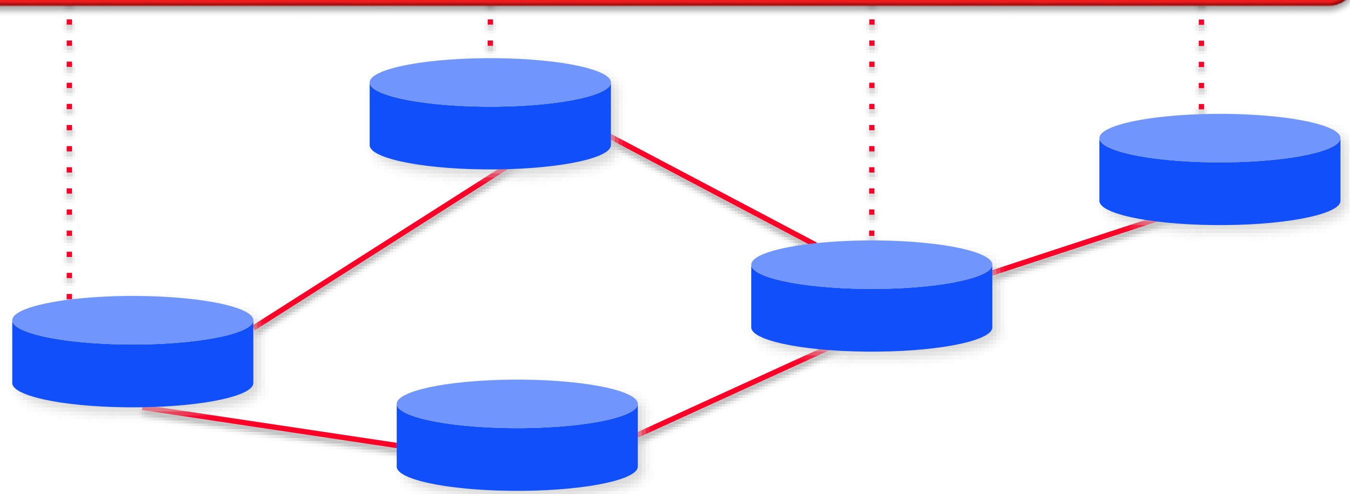
Specifies behavior — Control Program

Abstract Network Model

Compiles to topology — Network Virtualization

Global Network View

Transmits to switches — Network OS



SDN: Clean Separation of Concerns

- Control prgm: specify behavior on abstract model
  - Driven by Operator Requirements
- Net Virt'n: map abstract model to global view
  - Driven by Specification Abstraction
- NOS: map global view to physical switches
  - API: driven by Distributed State Abstraction
  - Switch/fabric interface: driven by Forwarding Abstraction

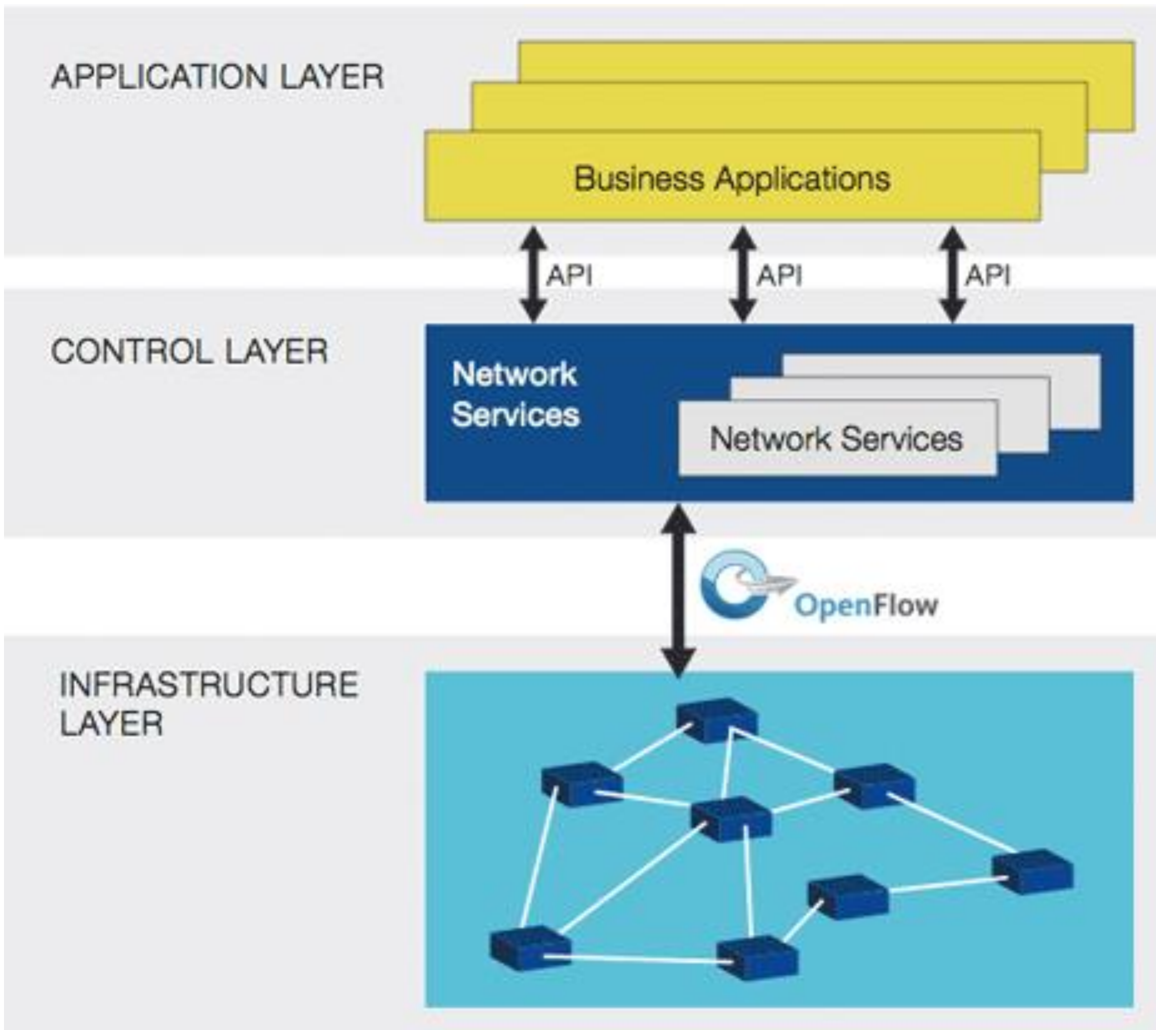
(2) Forwarding Abstraction

- Switches have two "brains"
  - Management CPU (smart but slow)
  - Forwarding ASIC (fast but dumb)
- Need a forwarding abstraction for both
  - CPU abstraction can be almost anything
- ASIC abstraction is much more subtle: OpenFlow
- OpenFlow
  - Control switch by inserting <header;action> entries
  - Essentially gives NOS remote access to forwarding table
  - Instantiated in OpenvSwitch

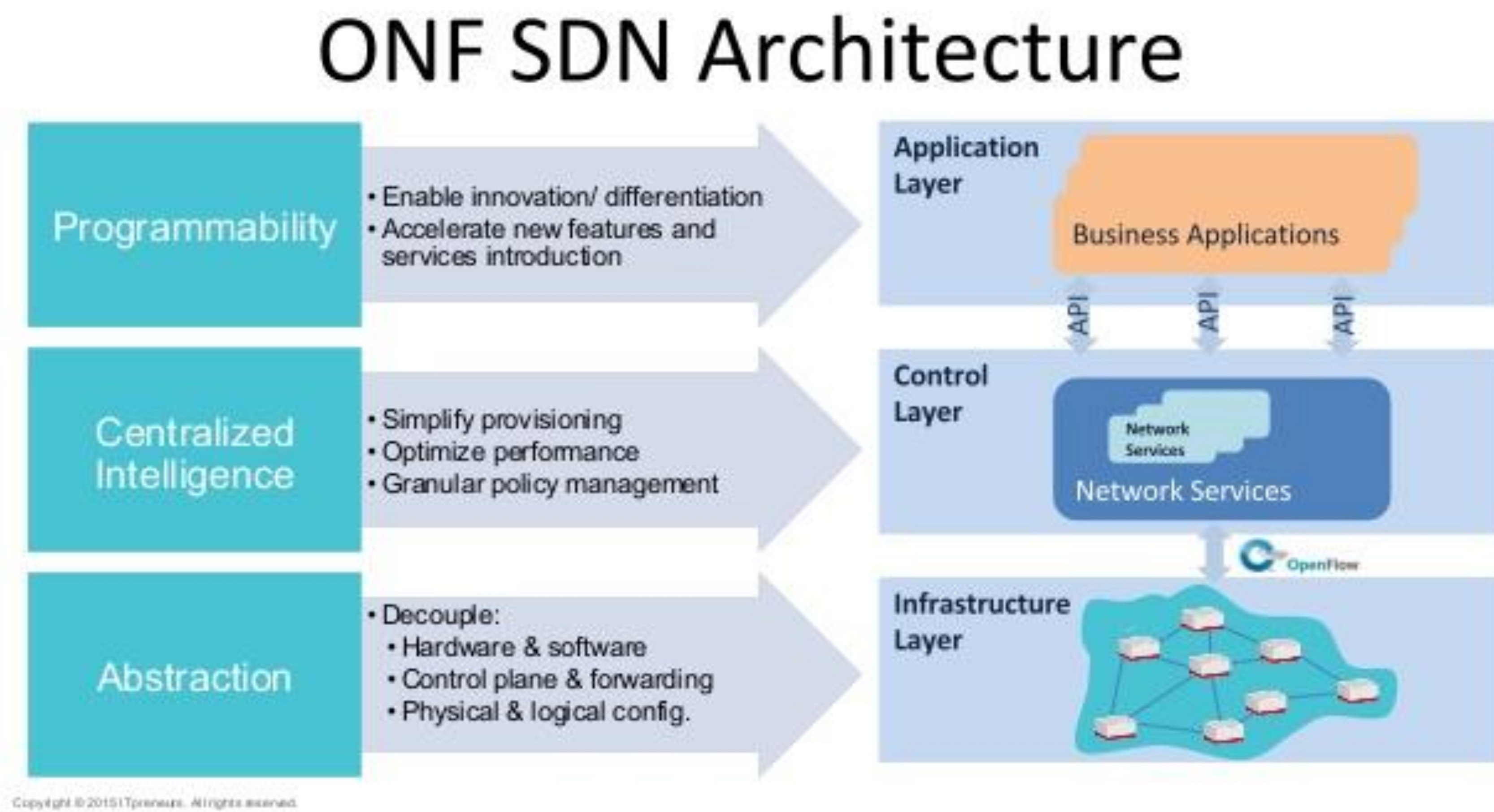
SDN: Clean Separation of Concerns

- Control prgm: specify behavior on abstract model
  - Driven by Operator Requirements
- Net Virt'n: map abstract model to global view
  - Driven by Specification Abstraction
- NOS: map global view to physical switches
  - API: driven by Distributed State Abstraction
  - Switch/fabric interface: driven by Forwarding Abstraction

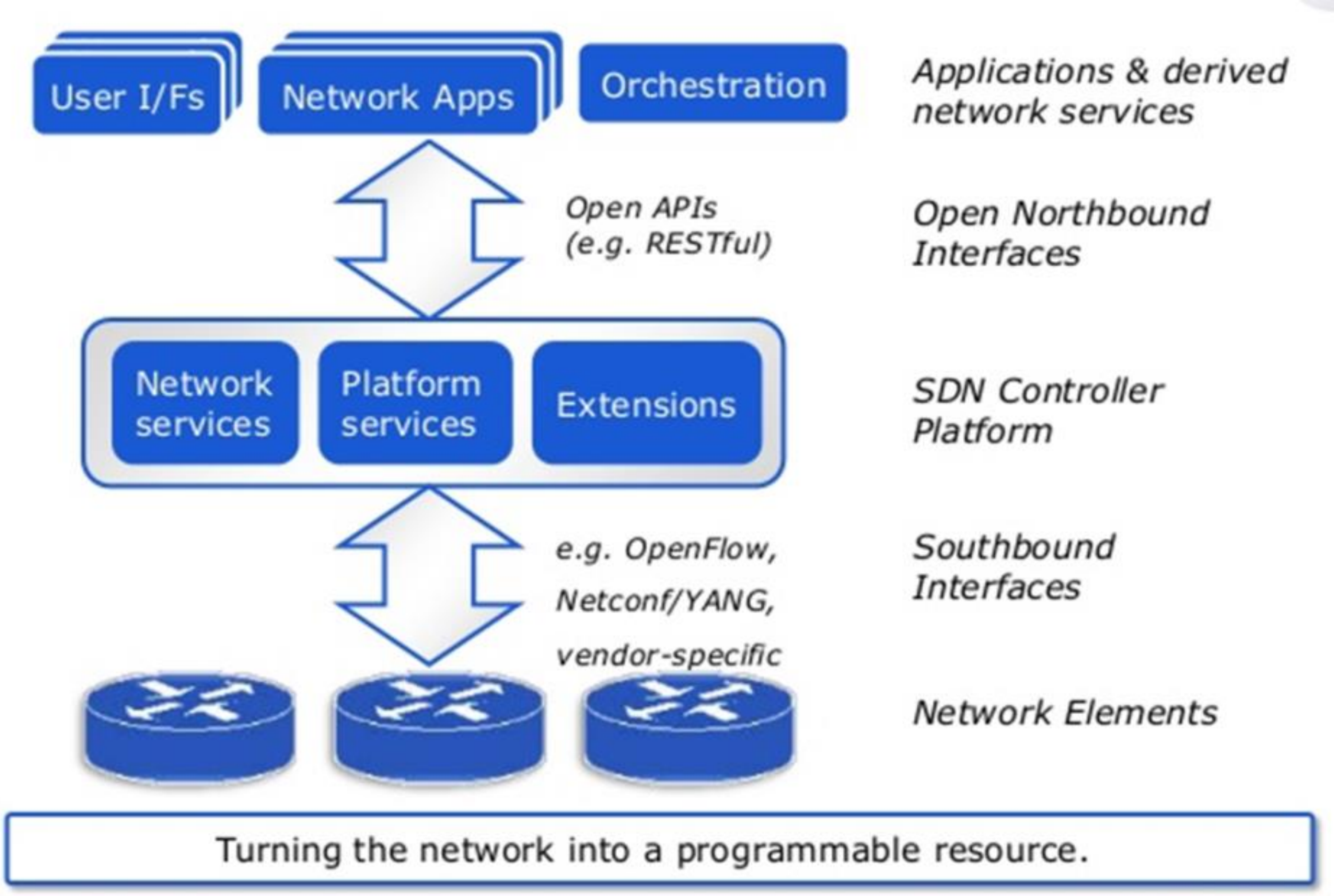
SDN - Open Networking Foundation



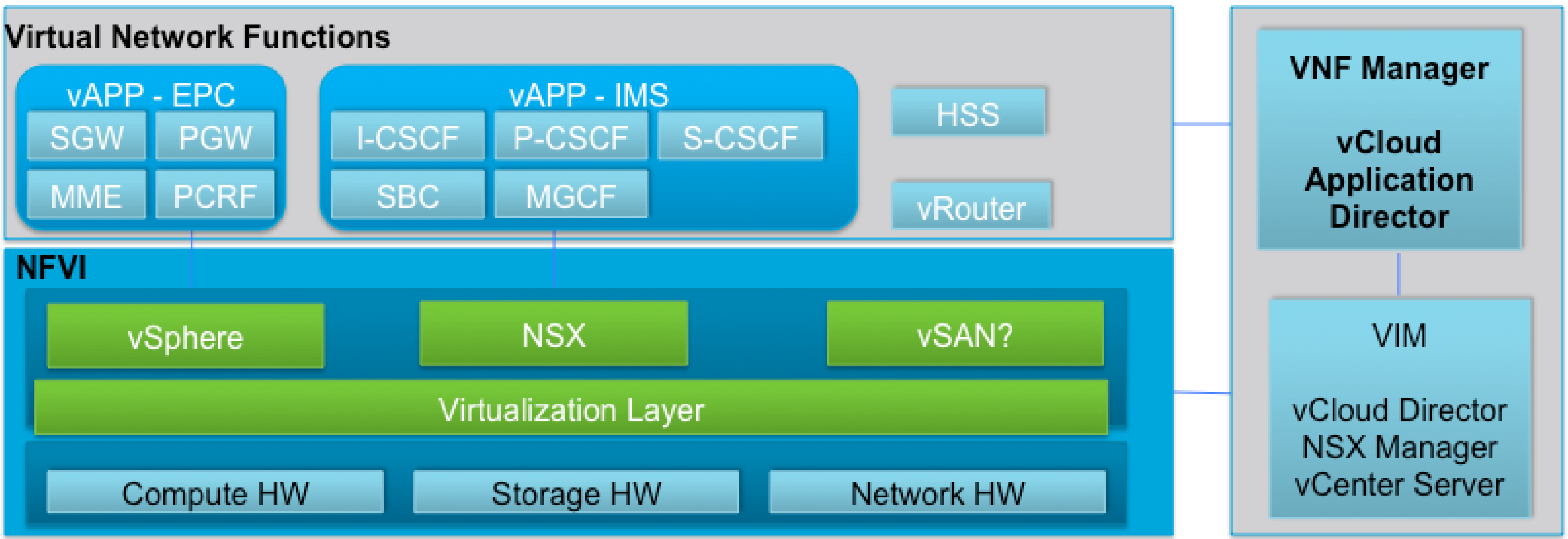
SDN - Open Networking Foundation



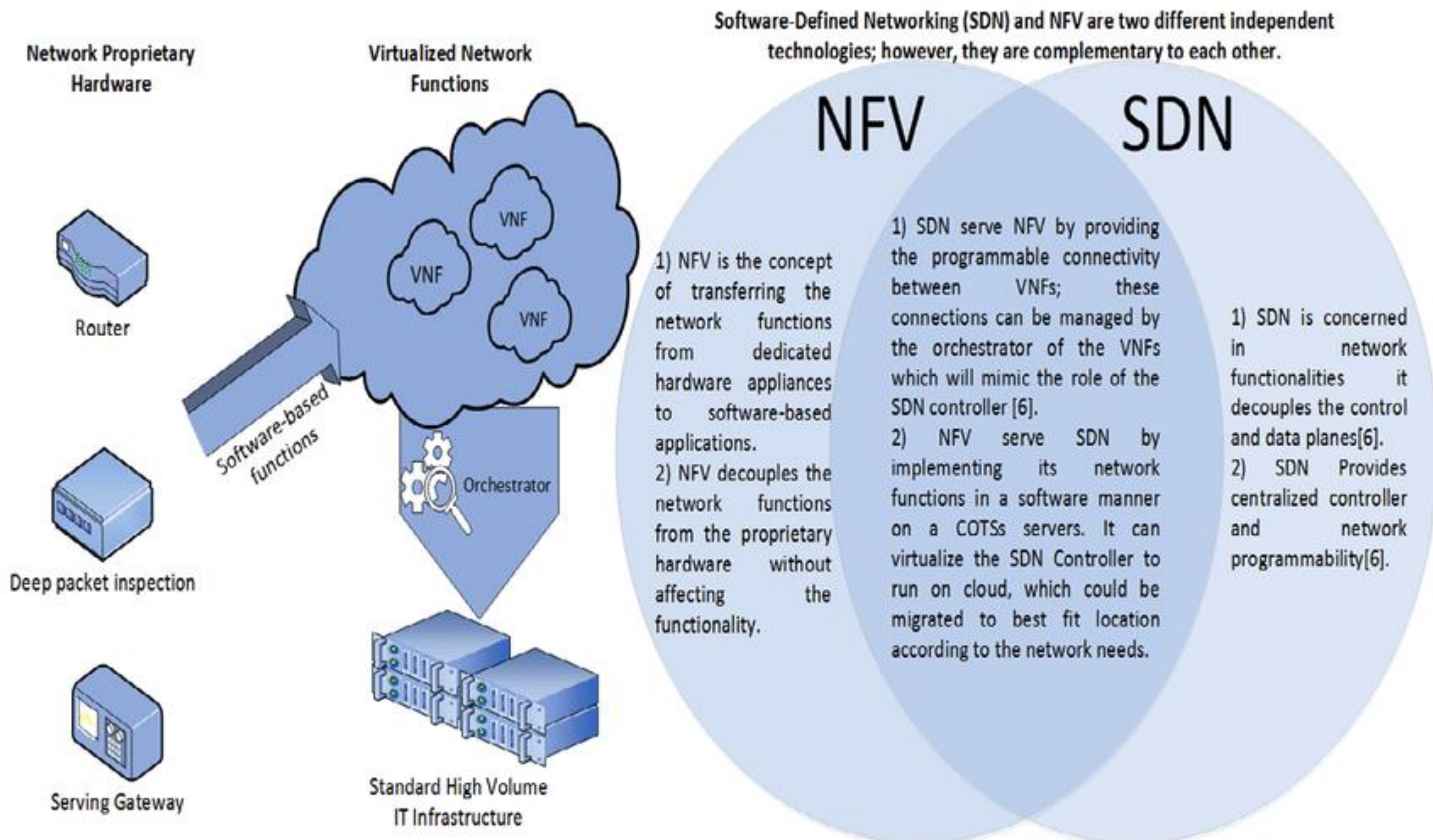
SDN - Open Networking Foundation



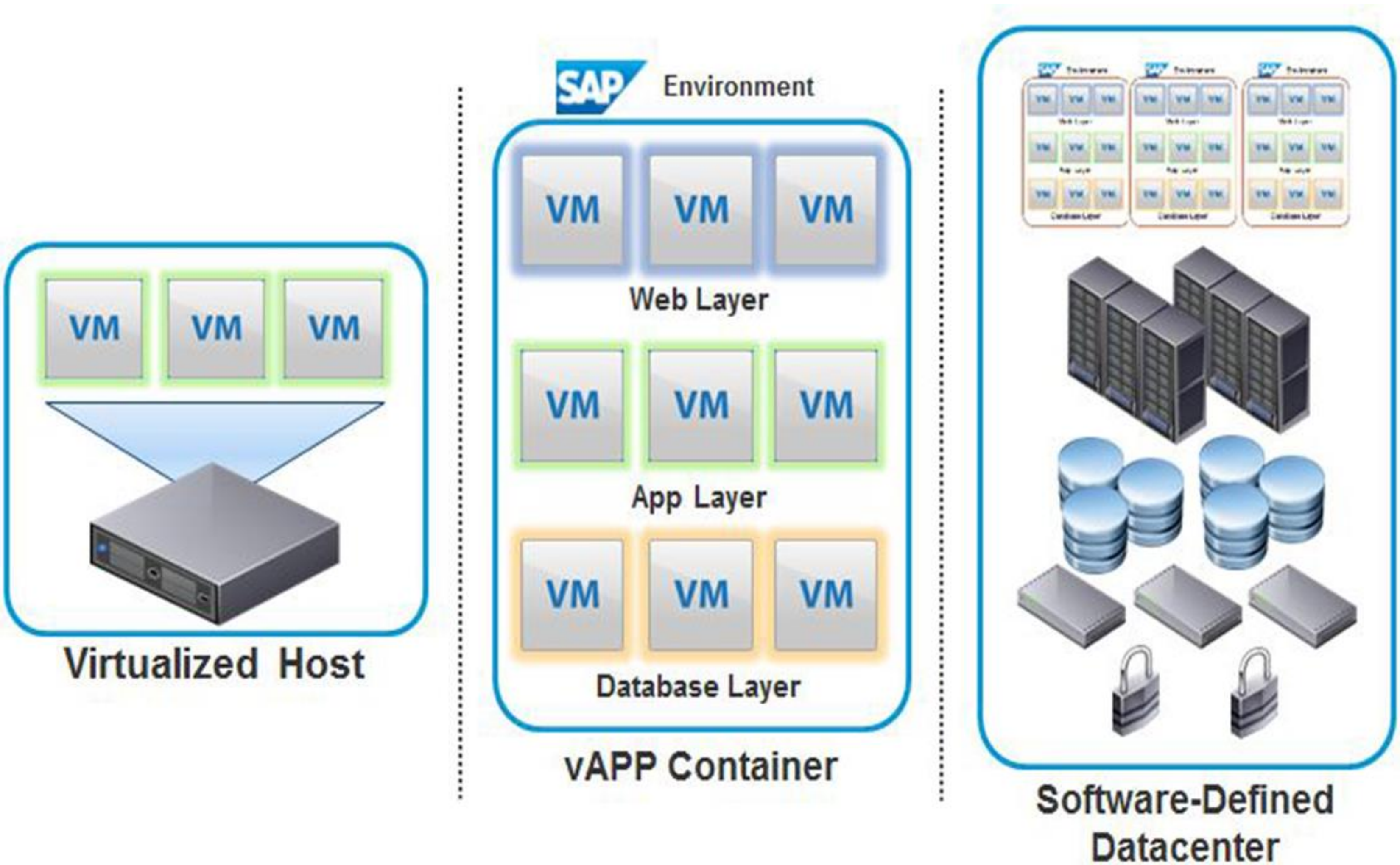
SDN vs. Virtual Network Functions



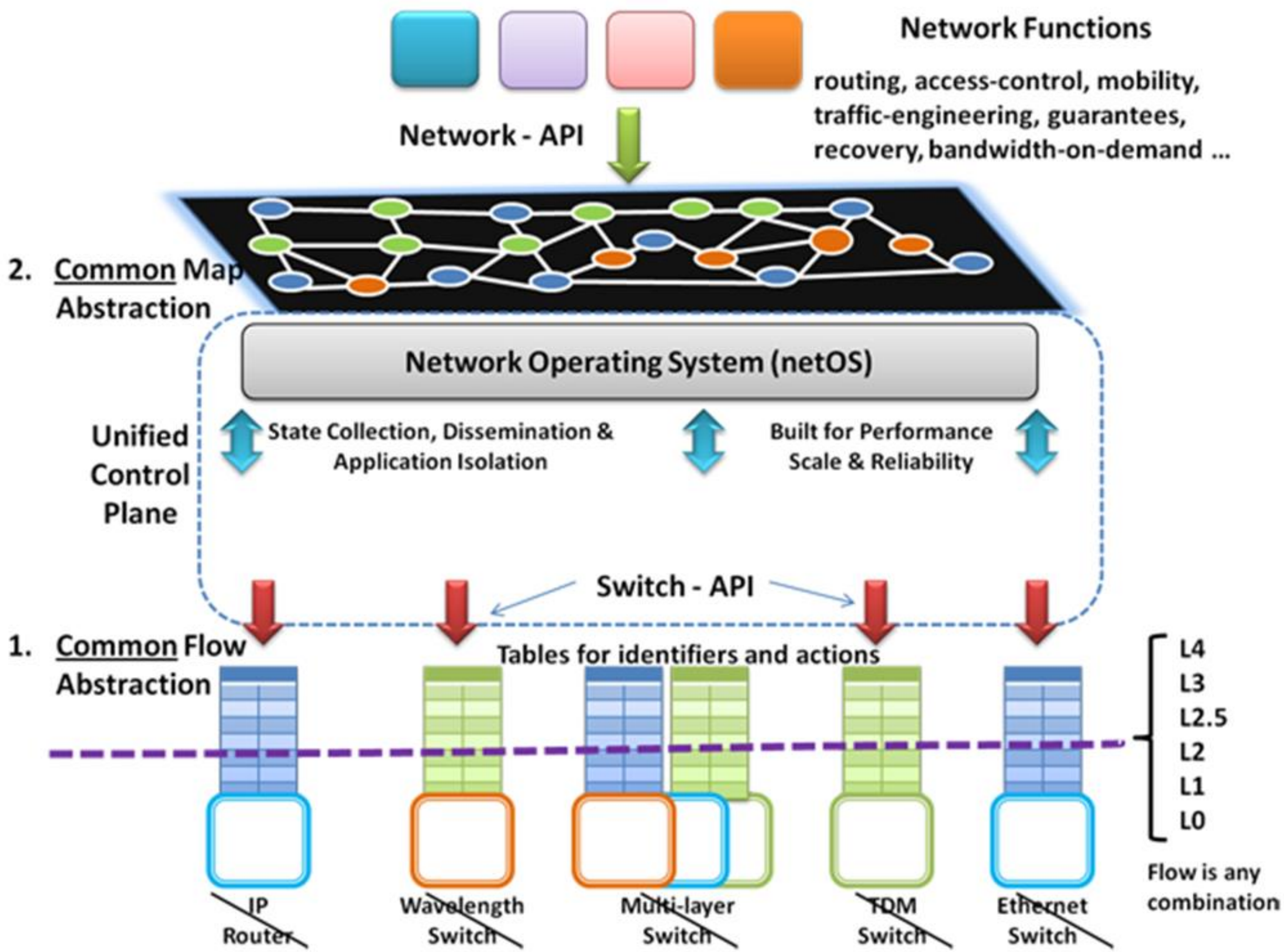
SDN vs. Virtual Network Functions



Software-Defined-Anything (SDx)

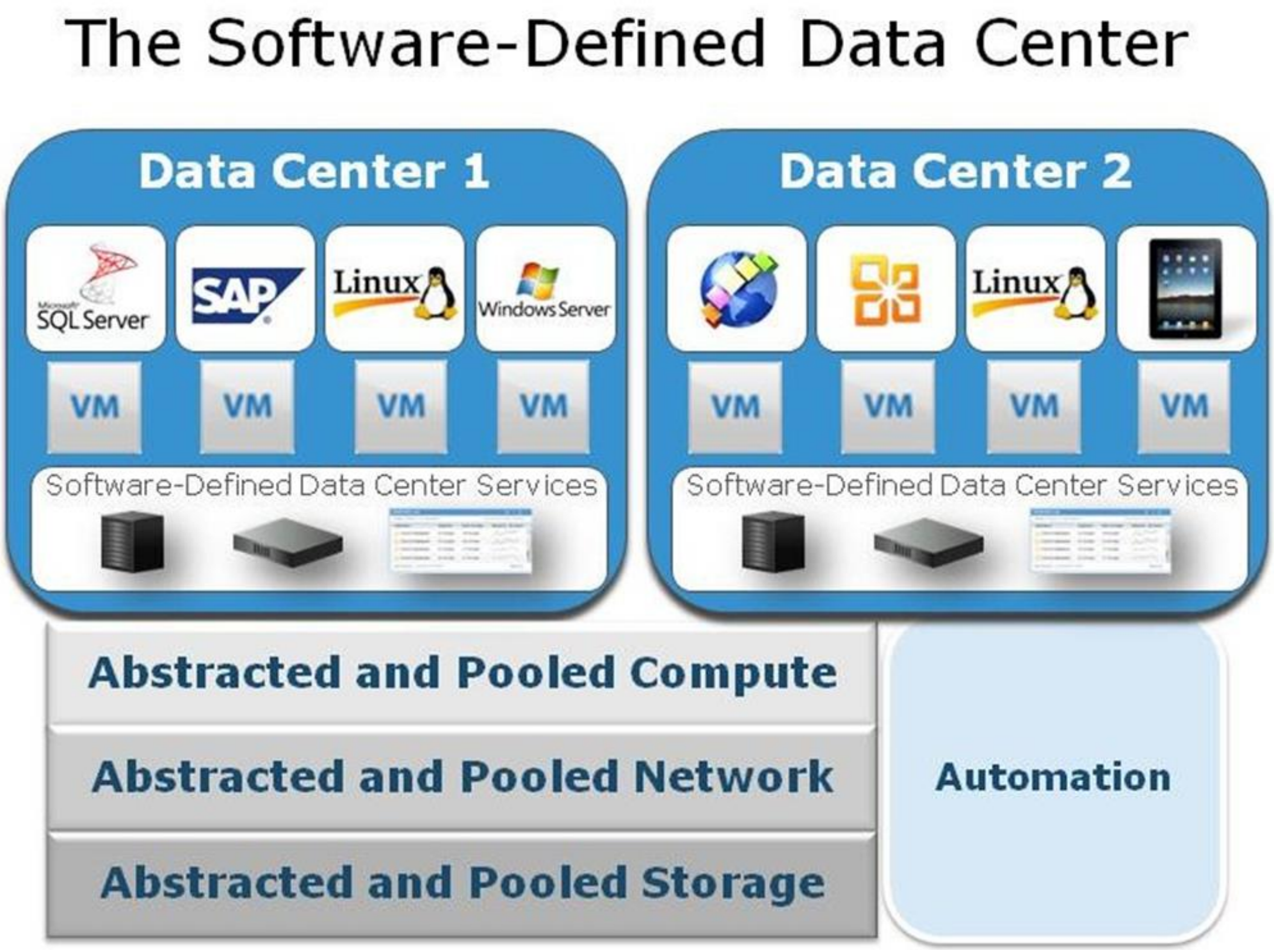


Software-Defined-Anything (SDx)

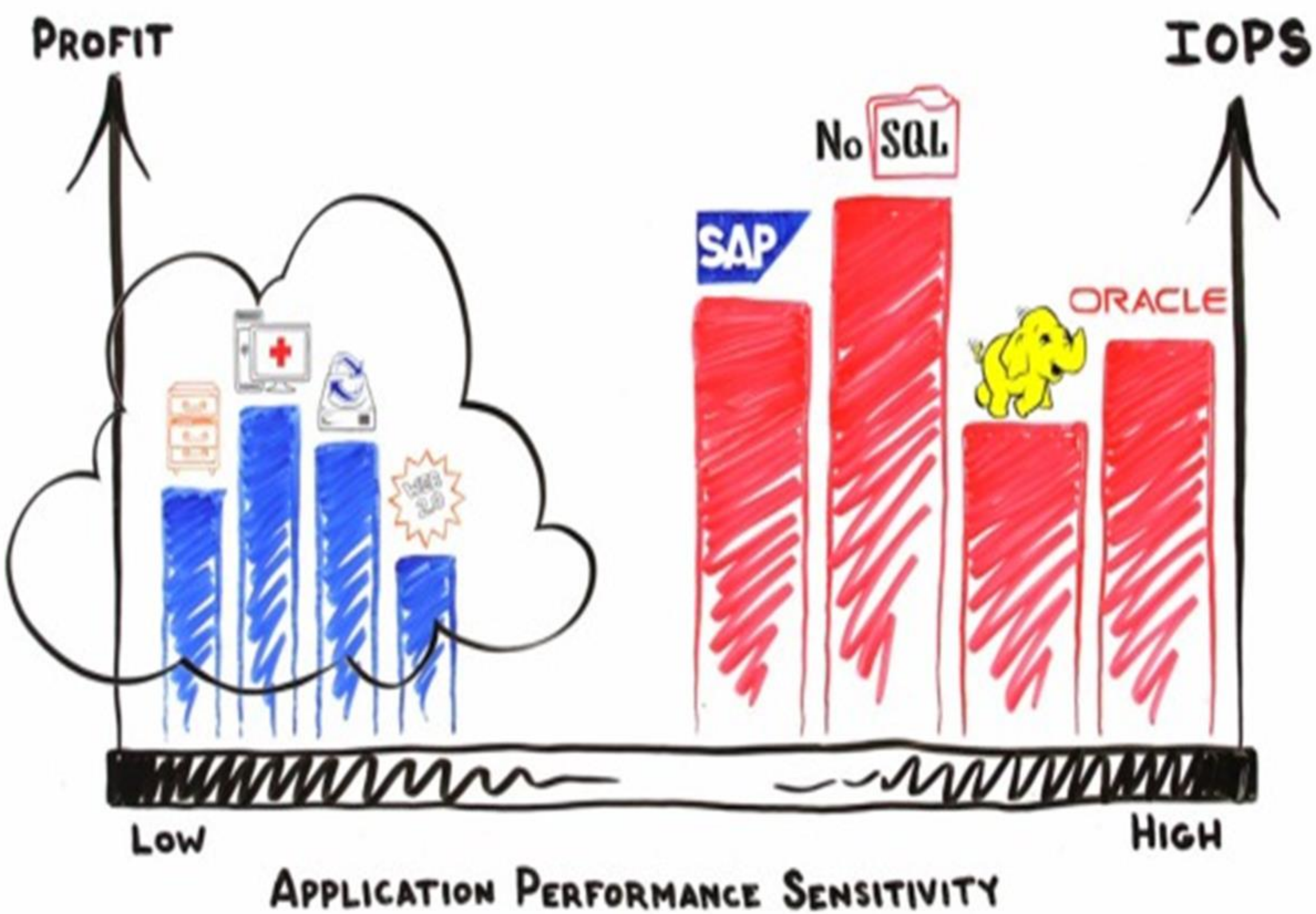


Software-Defined-Anything (SDx)

Software-Defined-Anything (SDx)



Software-Defined-Anything (SDx)



02

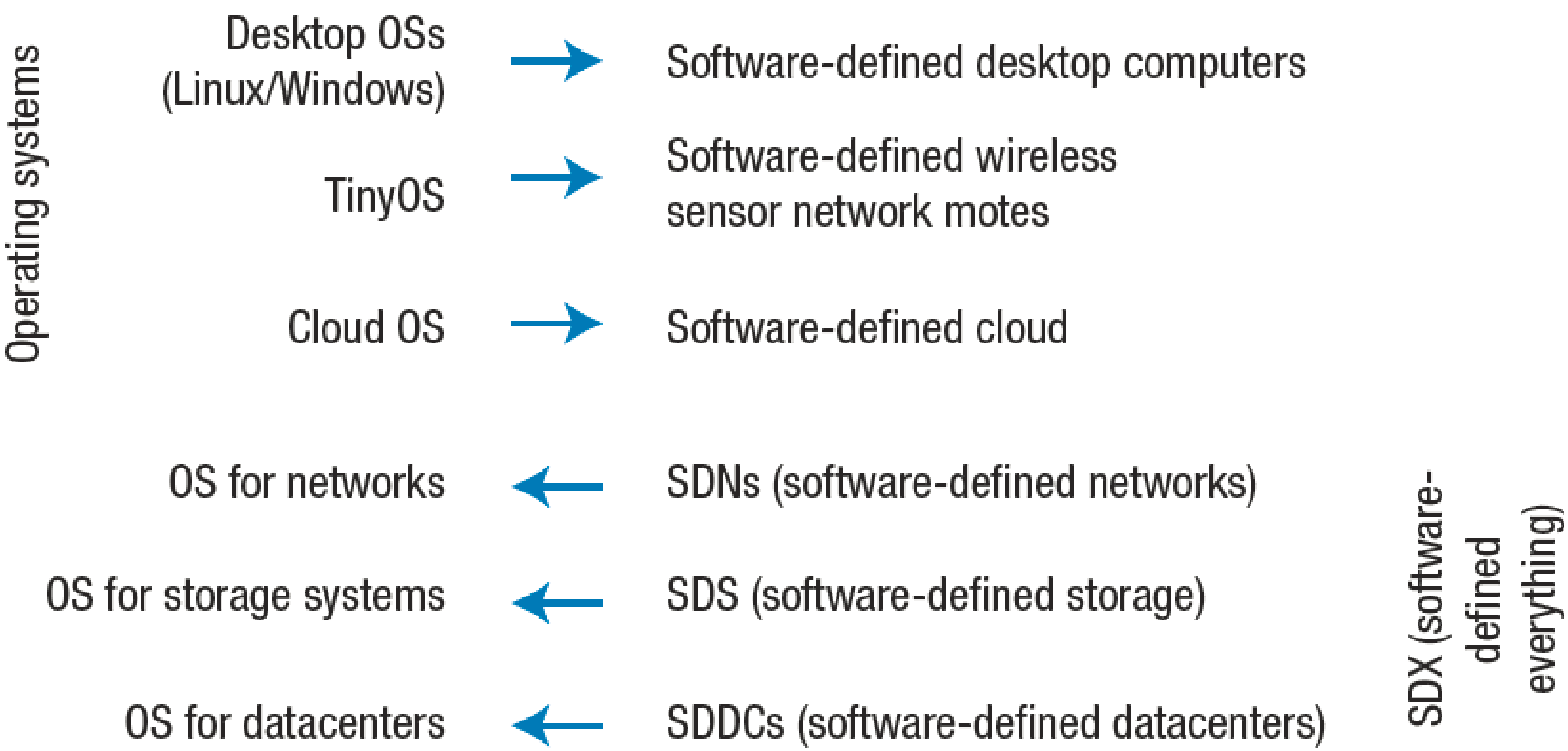
UOS

UBIQUITOUS OPERATING SYSTEMS

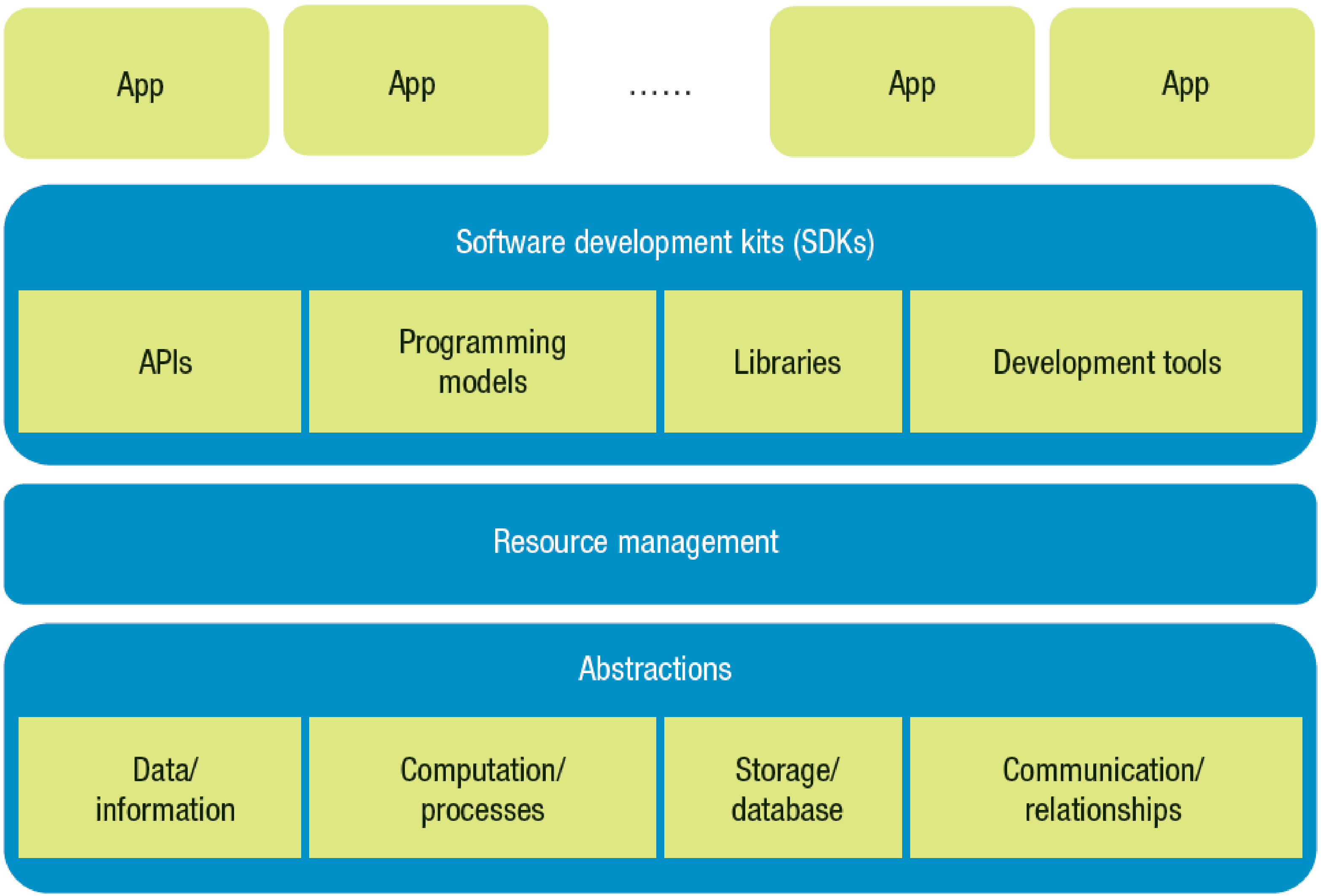
Case Study 1 - Toward Ubiquitous Operating Systems

Timeframe	Representative OS(s)	Computer system	Main characteristics
1956	GM-NAA I/O	IBM 704	The first practical OS Simple batch processing I/O management
1960s	IBM OS/360 series	IBM 360 series—mainframes	Time-sharing Multibatch processing Memory management Virtual machines (VM/370)
1970s	Unix	Minicomputers/workstations	First modern OS Developed with machine-independent languages (C) Provides standard interfaces Integrated development environment
1980s	Mac OS, Windows, Linux	Personal computers (PCs)	Provides modern GUI Improves usability for personal users
2000s	Apple iOS, Google Android, Windows Phone	Smartphones	Customization of traditional OSs Improves usability for mobile devices New app delivery model (App Store, Google Play)

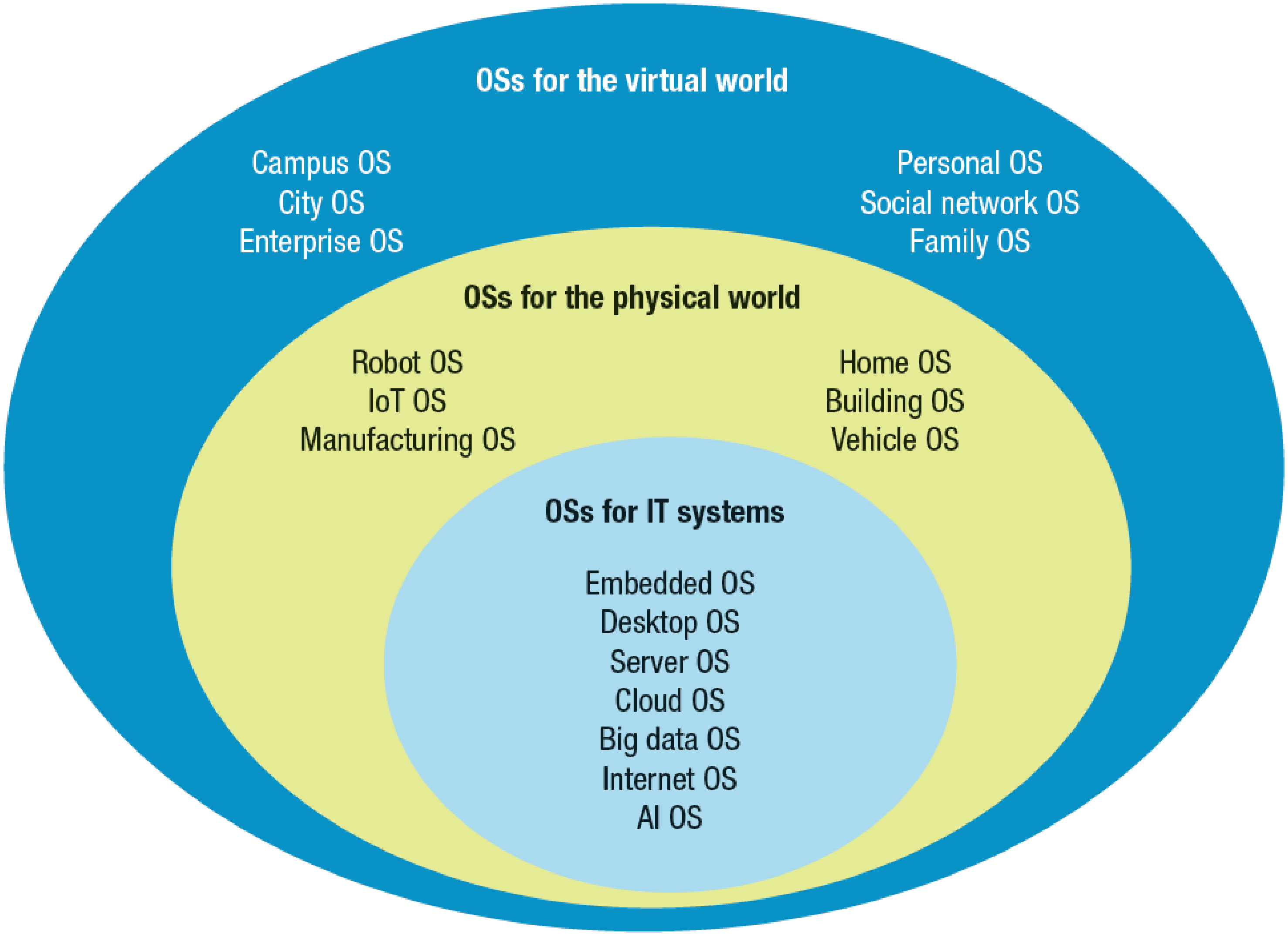
Toward Ubiquitous Operating Systems



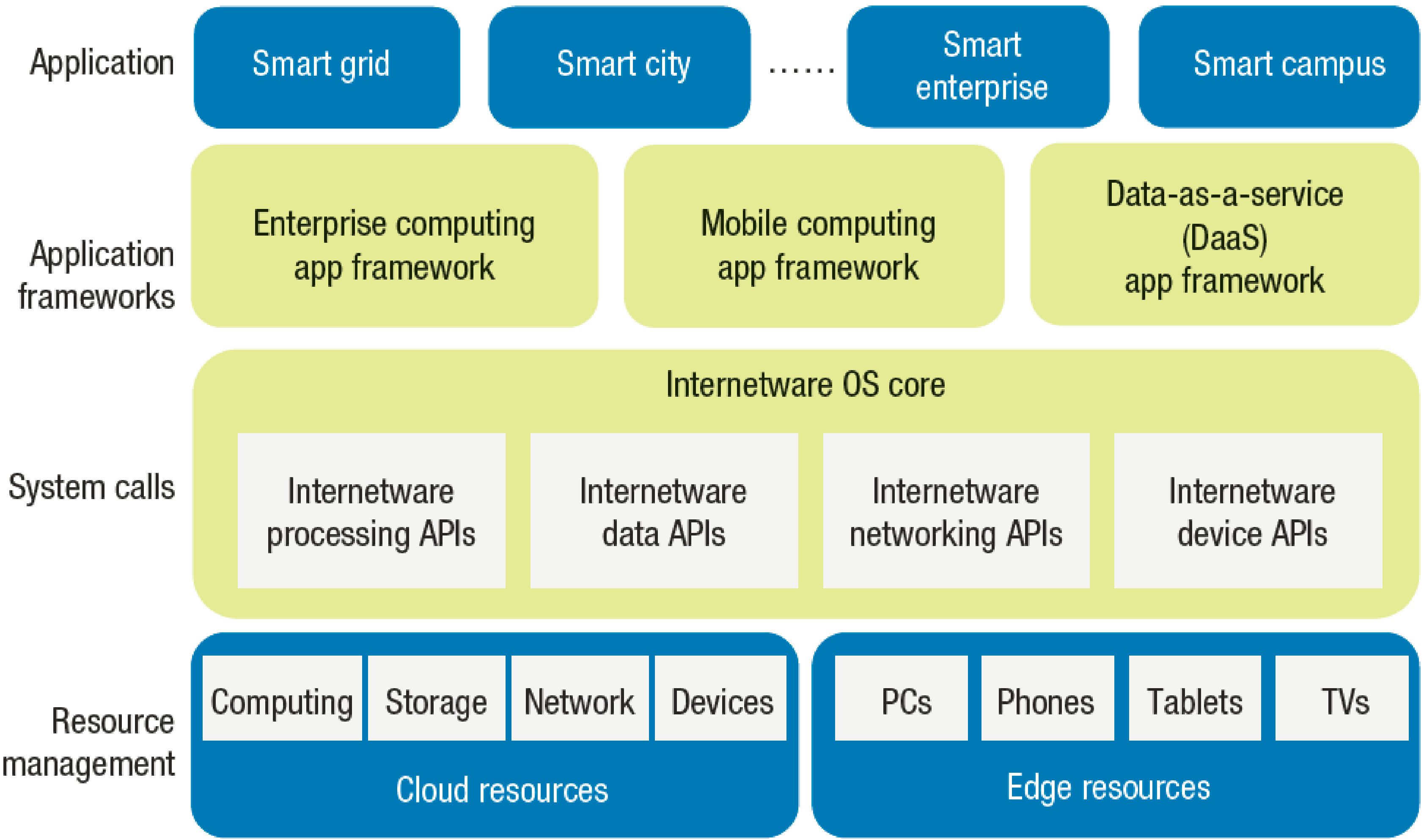
Toward Ubiquitous Operating Systems



Toward Ubiquitous Operating Systems



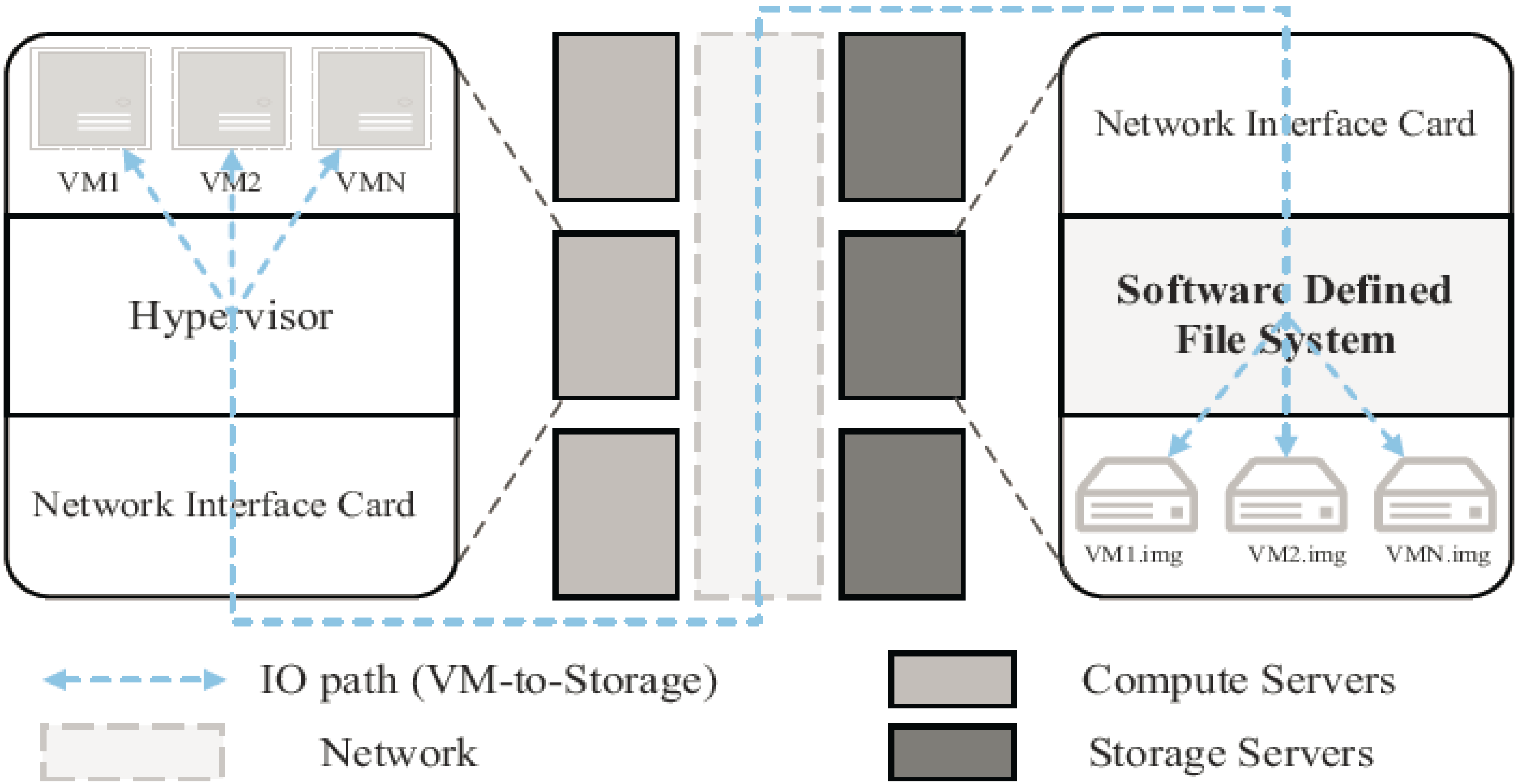
Toward Ubiquitous Operating Systems



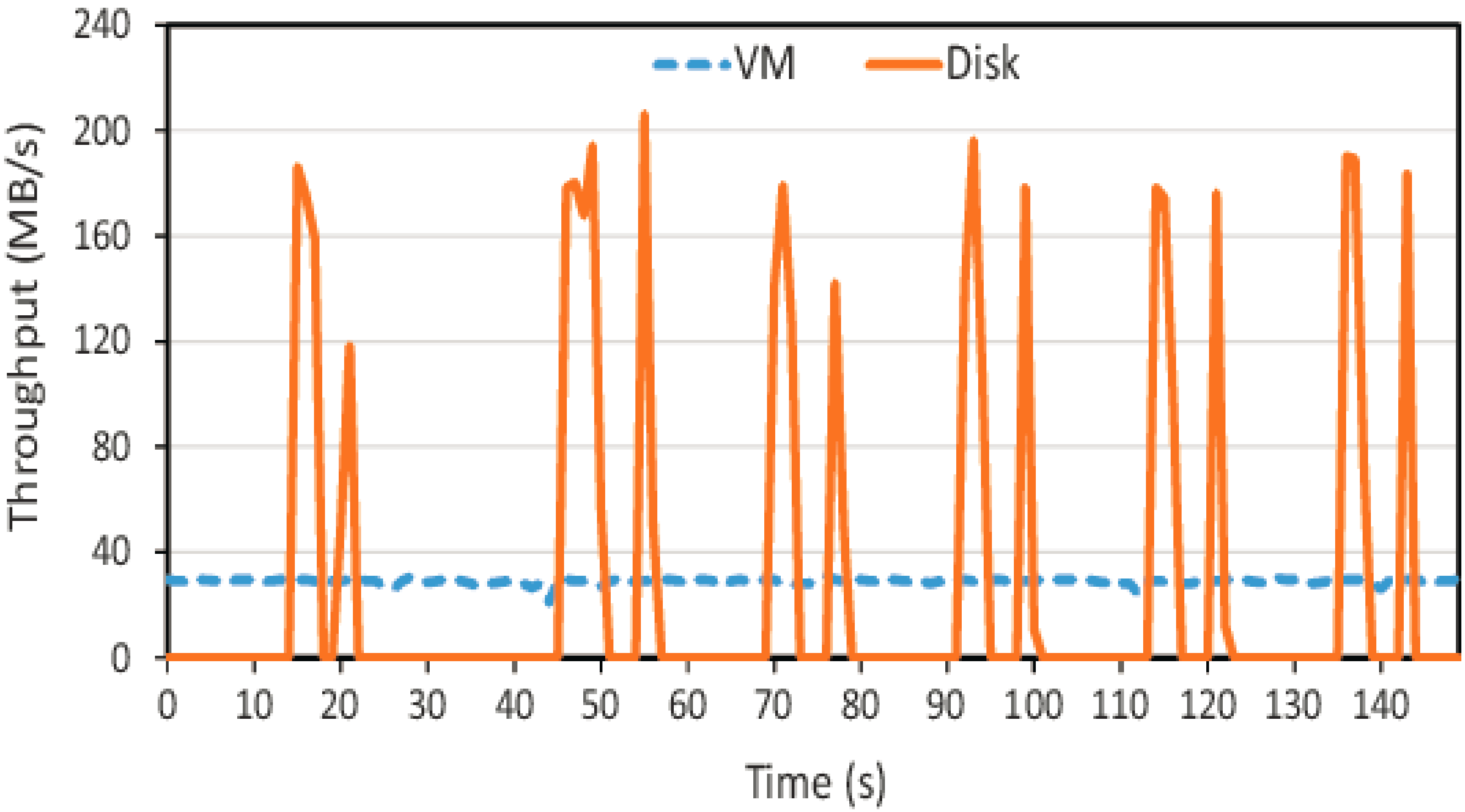
03 SDFS

SOFTWARE-DEFINED FILE SYSTEM

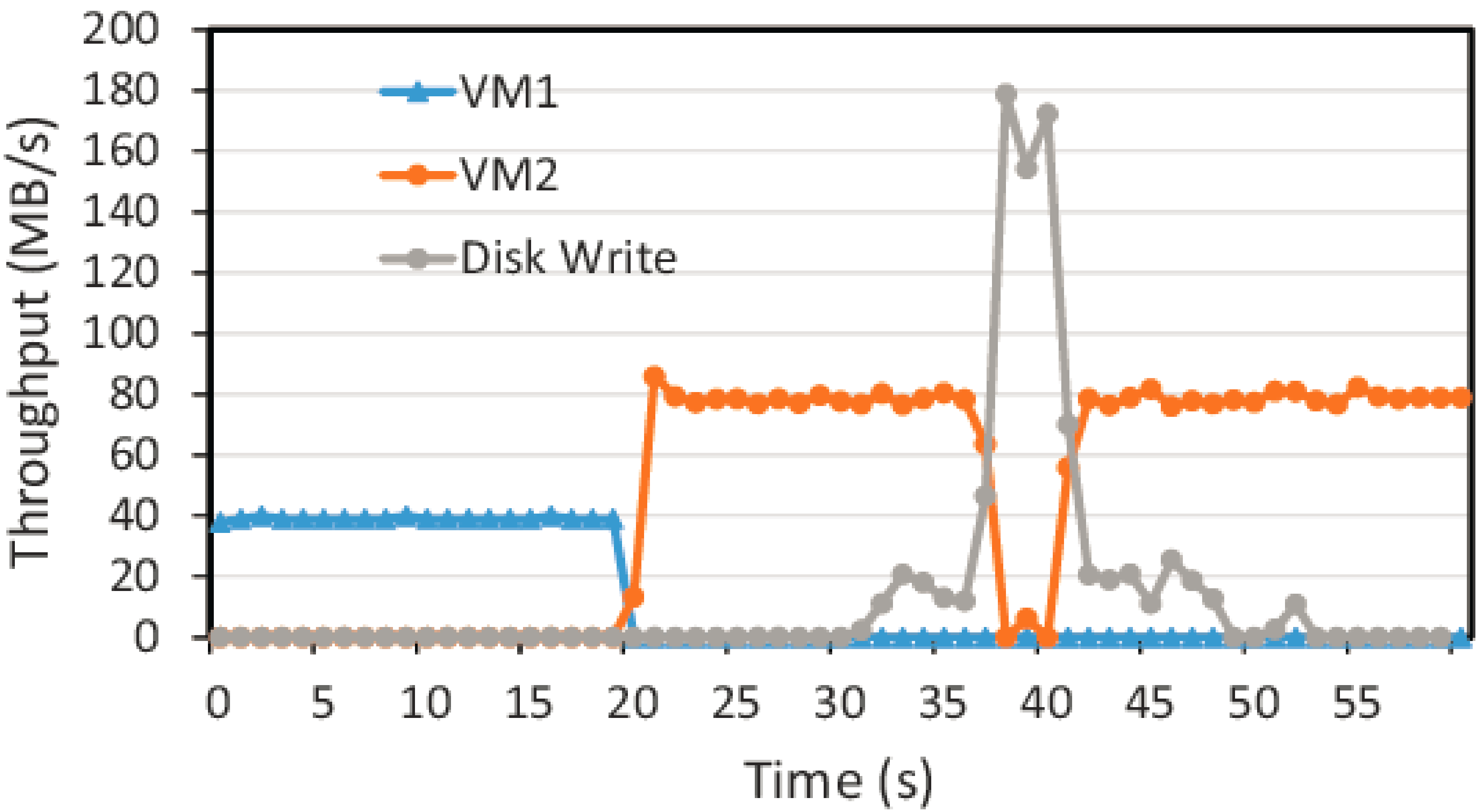
Case Study 2 - Software-Defined File System



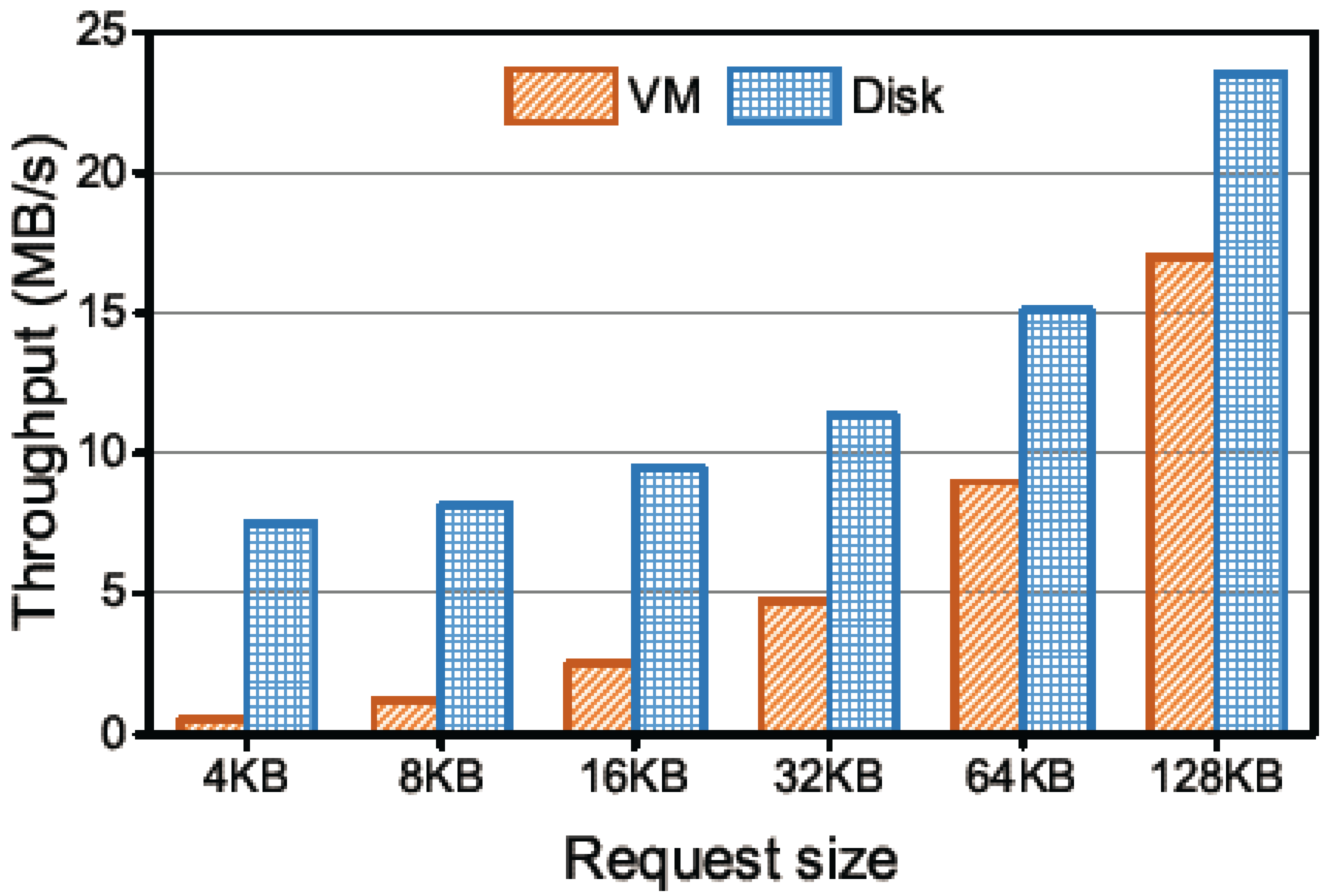
Motivation


VM
Disk
Throughput (MB/s)
Time (s)
0
40
80
120
160
200
240
0
10
20
30
40
50
60
70
80
90
100
110
120
130
140

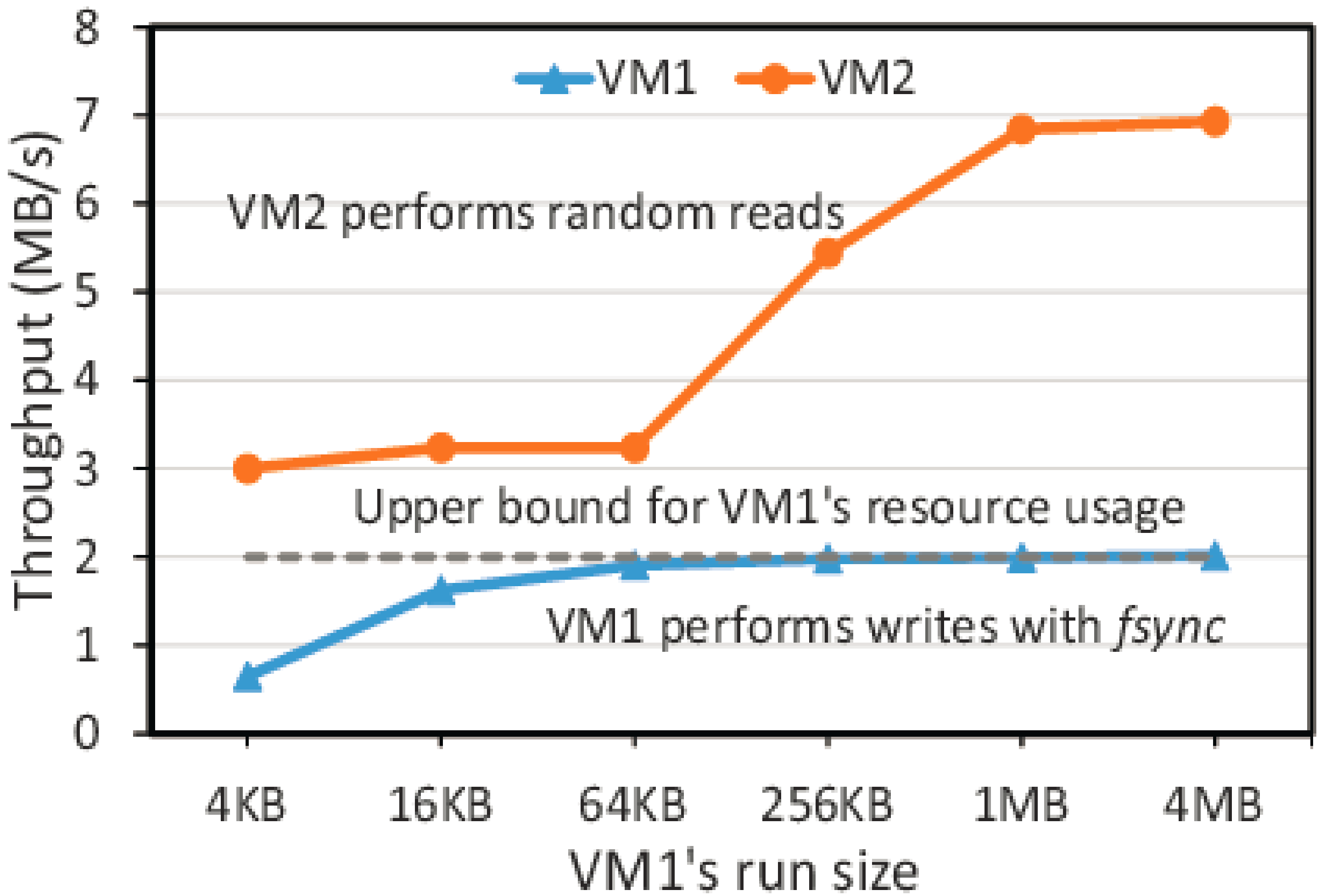
Motivation


VM1
VM2
Disk Write
Throughput (MB/s)
Time (s)
0
20
40
60
80
100
120
140
160
180
200
0
5
10
15
20
25
30
35
40
45
50
55

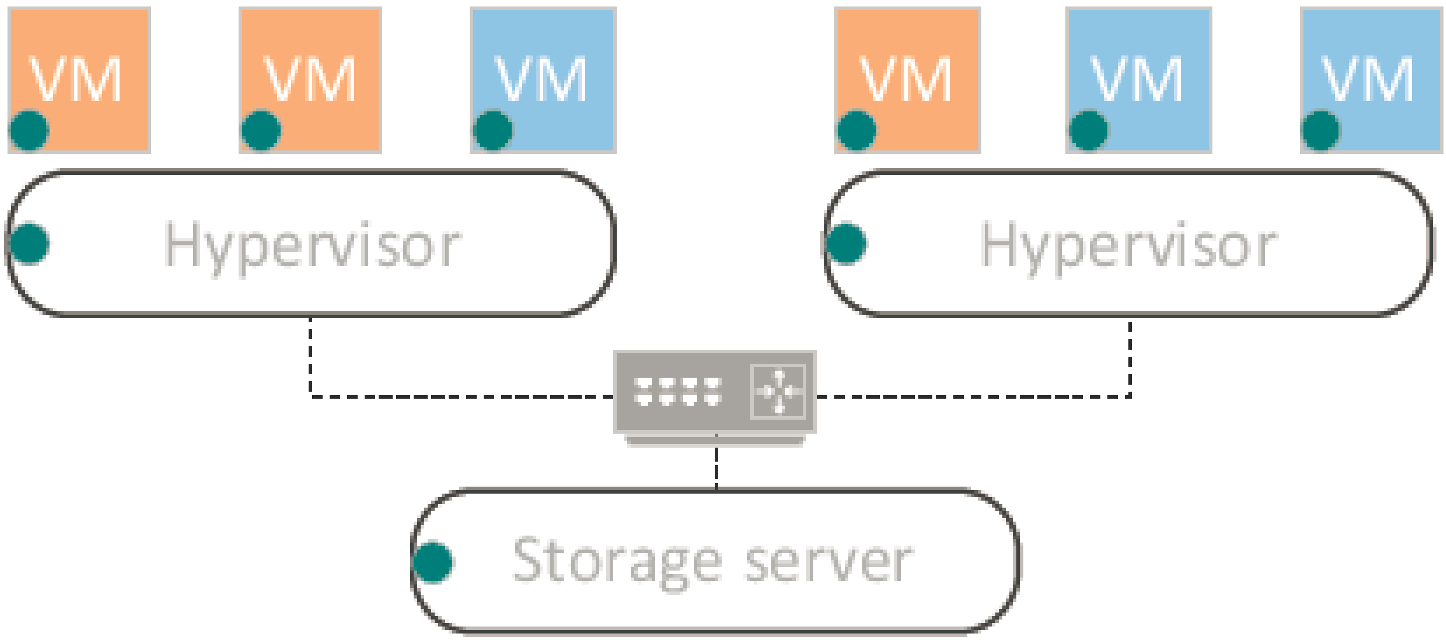
Motivation


VM
Disk
Throughput (MB/s)
25
20
15
10
5
0
4KB
8KB
16KB
32KB
64KB
128KB
Request size

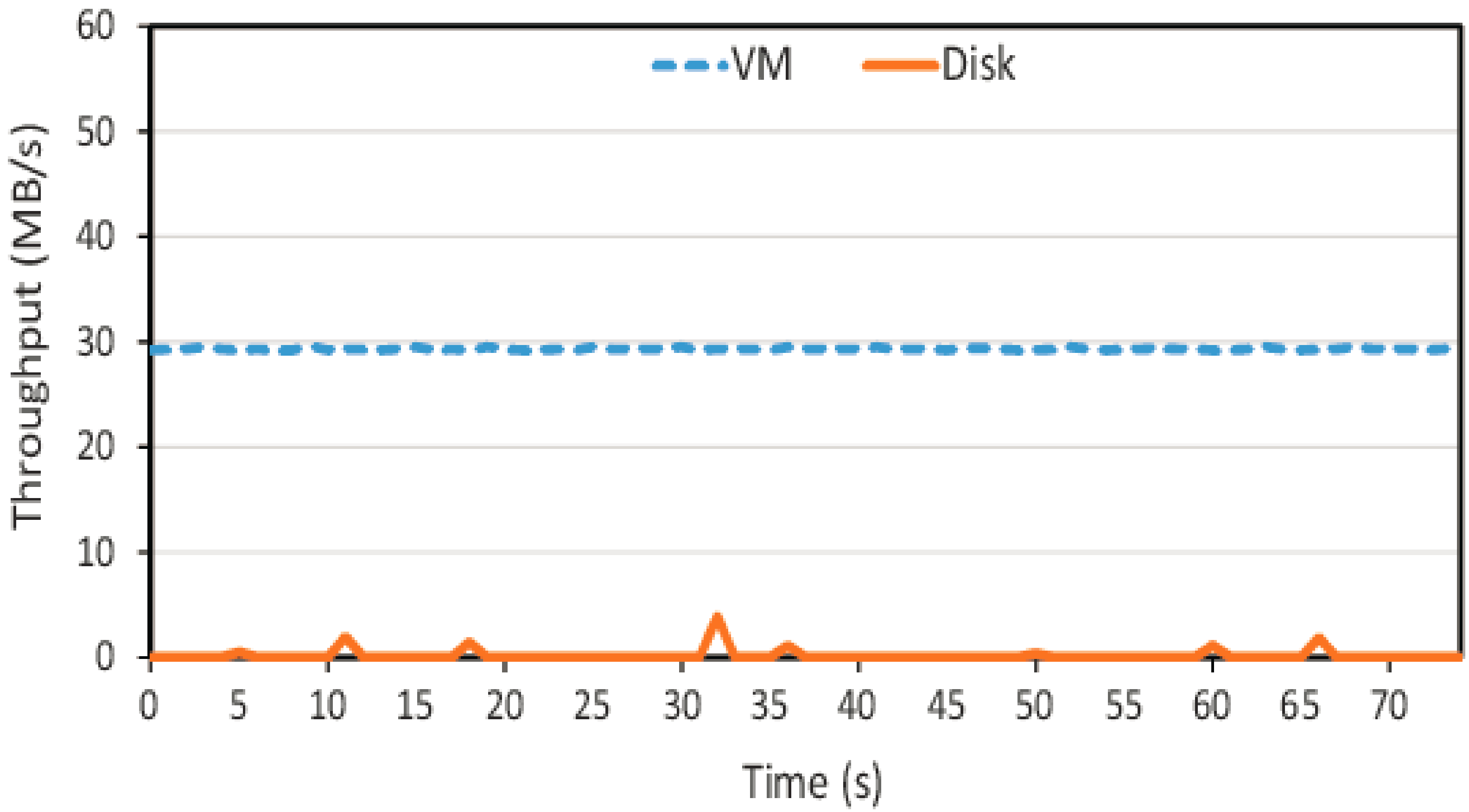
Motivation


VM1
VM2
Throughput (MB/s)
8
7
6
5
4
3
2
1
0
VM2 performs random reads
Upper bound for VM1's resource usage
VM1 performs writes with *fsync*
4KB
16KB
64KB
256KB
1MB
4MB
VM1's run size

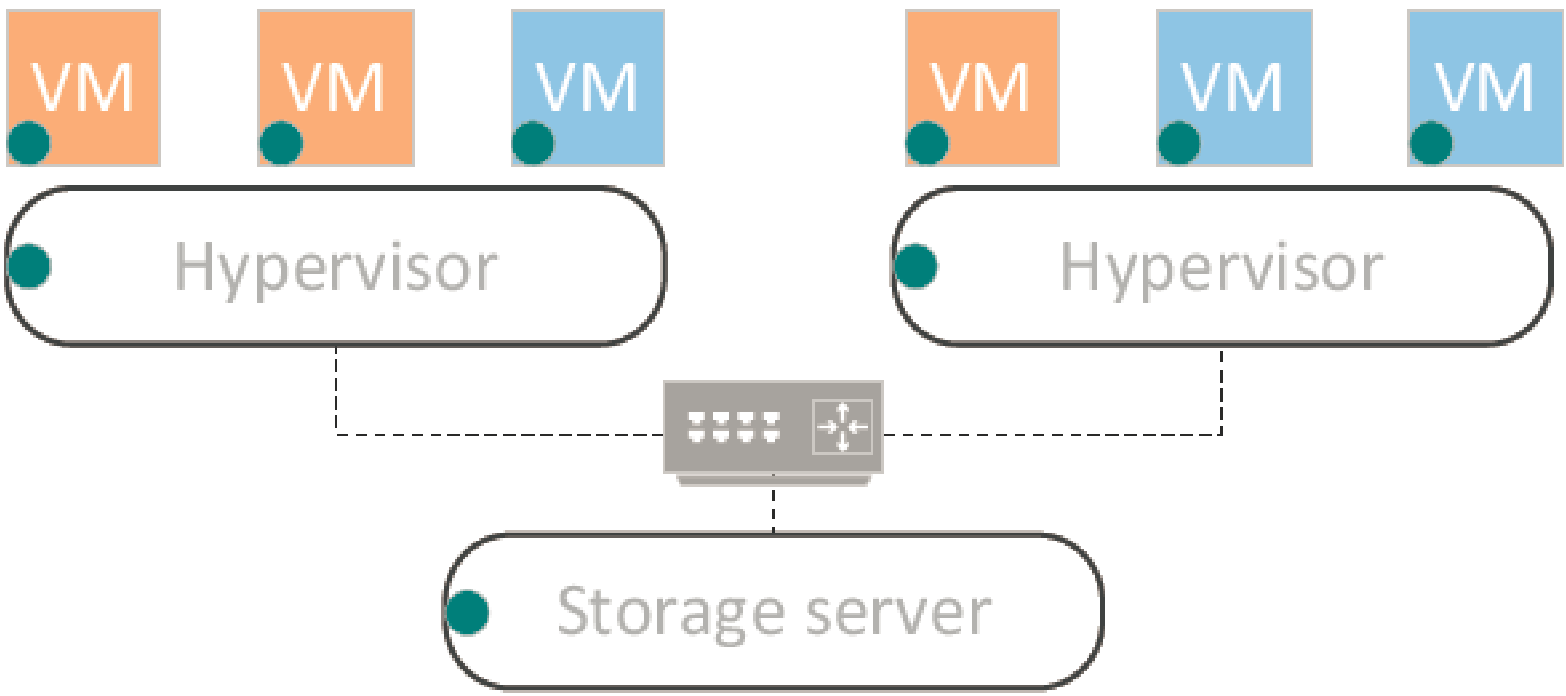
Motivation



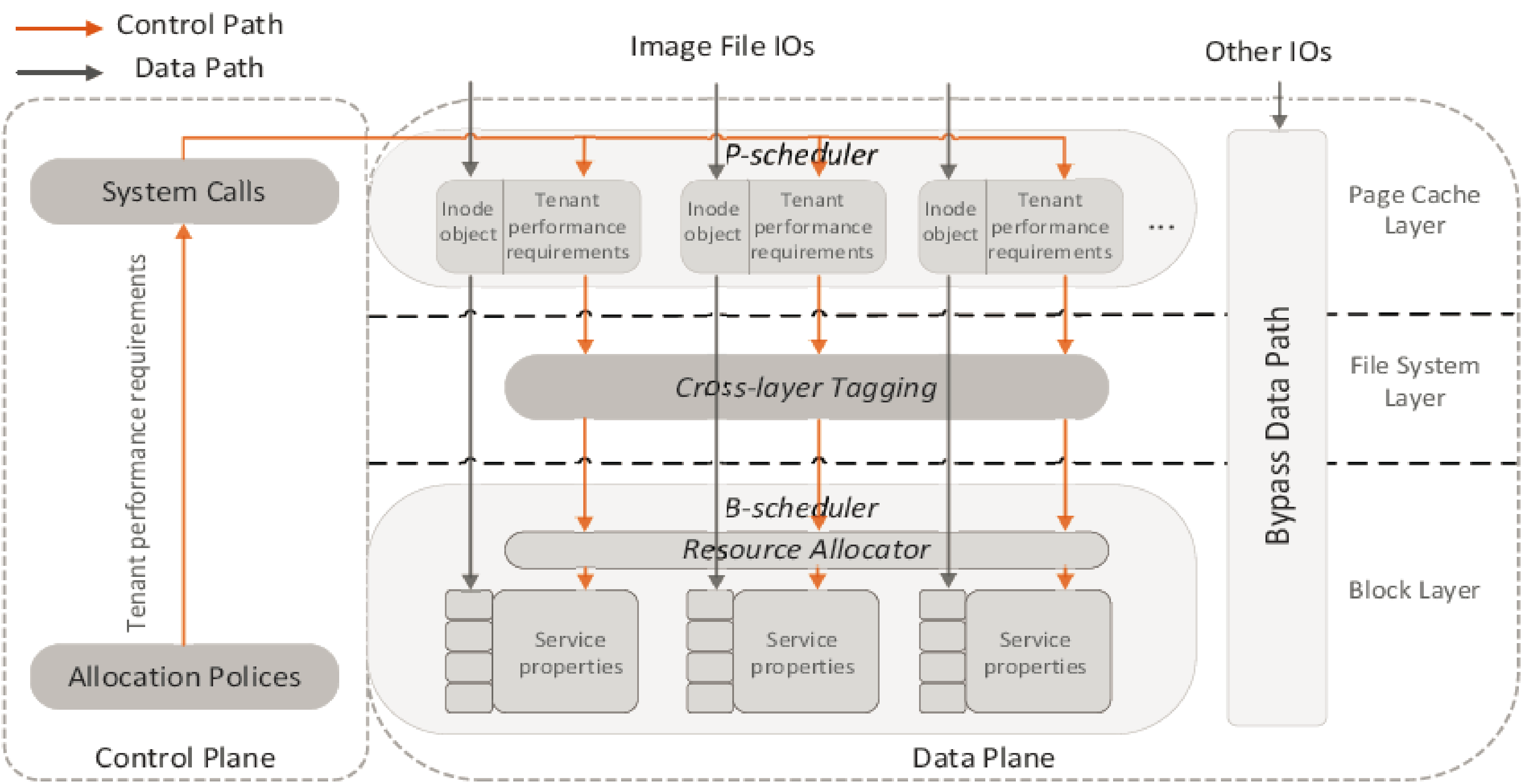
Motivation


VM
Disk
Throughput (MB/s)
Time (s)
0
10
20
30
40
50
60
5
15
25
35
45
55
65
70

Motivation



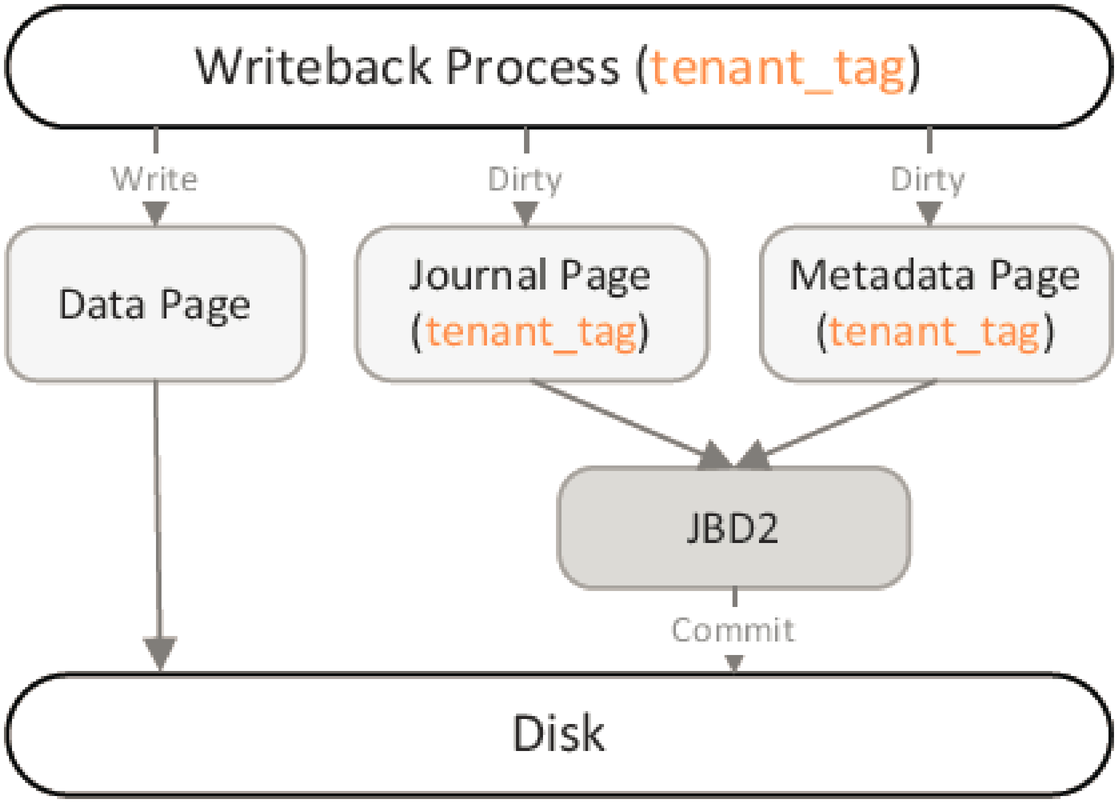
SDFS: Design


Control Path
Data Path
Image File IOs
Other IOs
System Calls
Tenant performance requirements
Allocation Polices
Control Plane
P-scheduler
Inode object
Tenant performance requirements
...
Cross-layer Tagging
B-scheduler
Resource Allocator
Service properties
Bypass Data Path
Page Cache Layer
File System Layer
Block Layer
Data Plane

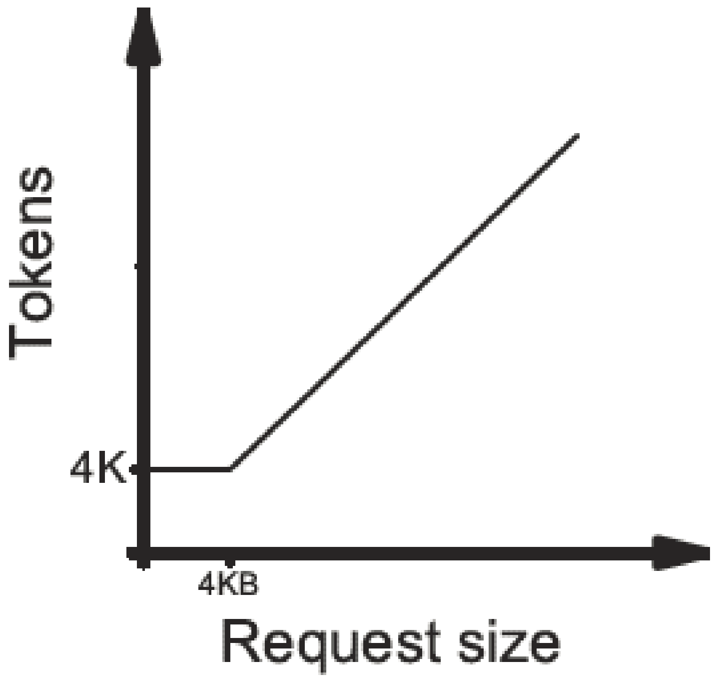
SDFS: Control Plane

Function	**Description**
set_tenant_tag_fixed	sets a fixed throughput guarantee (f) for a VM. $f \in (0, 200)$
set_tenant_tag_min	sets a minimum throughput guarantee (m) for a VM. $m \in (0, 200)$
set_tenant_tag_max	sets an upper limit on resource usage (l) for a VM. $l \in (0, 200)$
set_tenant_tag_infi	does not set resource usage limit for a VM.
get_tenant_tag	returns tenant performance requirements

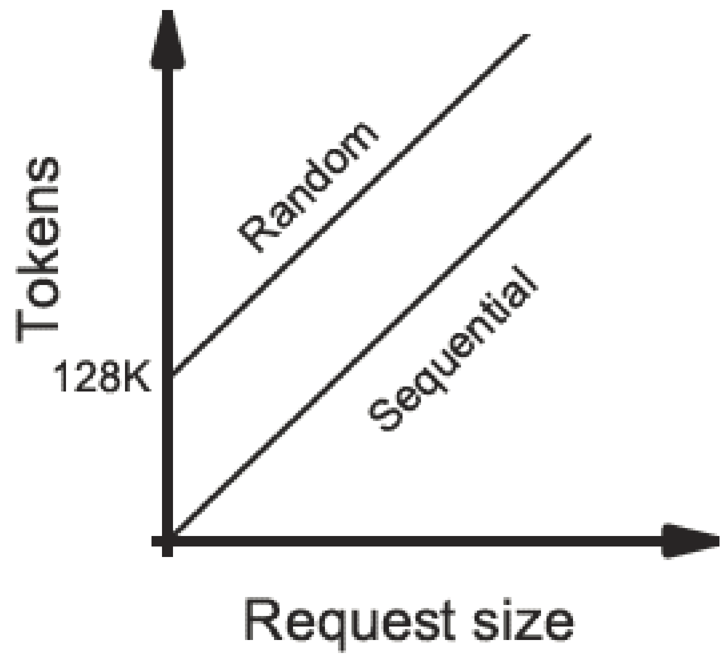
SDFS: Control Plane



SDFS: Data Plane

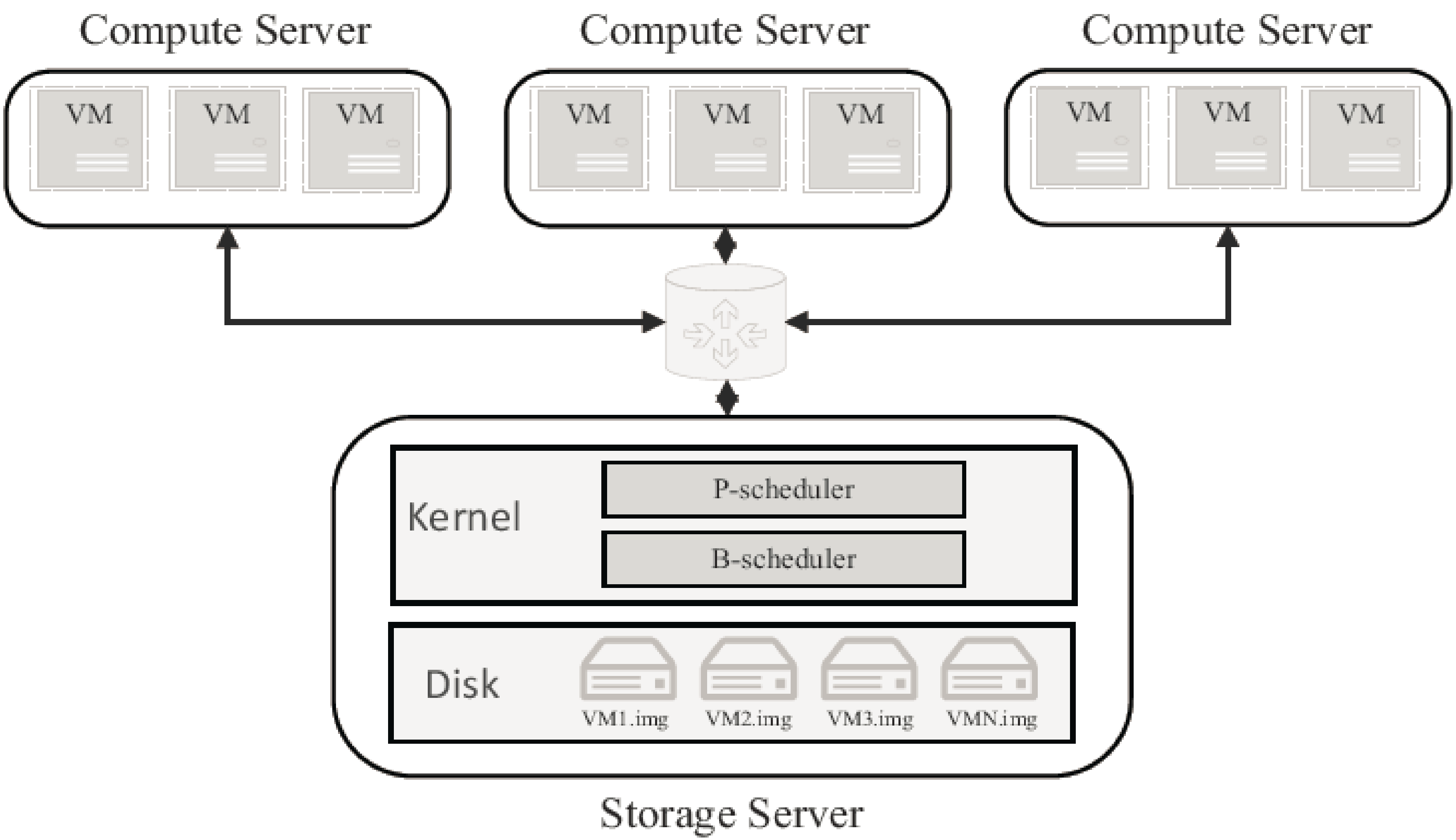


(a) P-scheduler



(b) B-scheduler

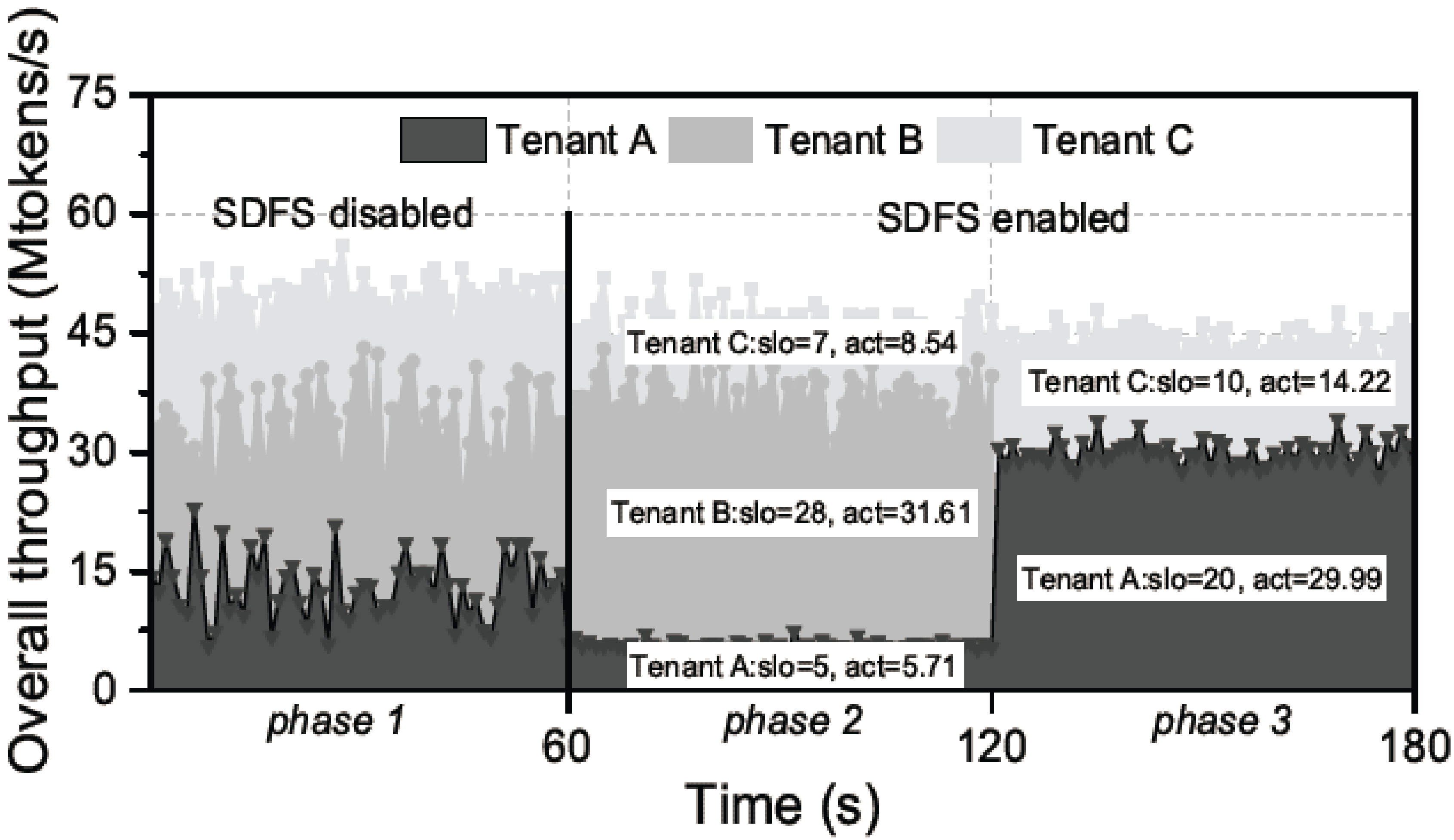
Testbed Deployment



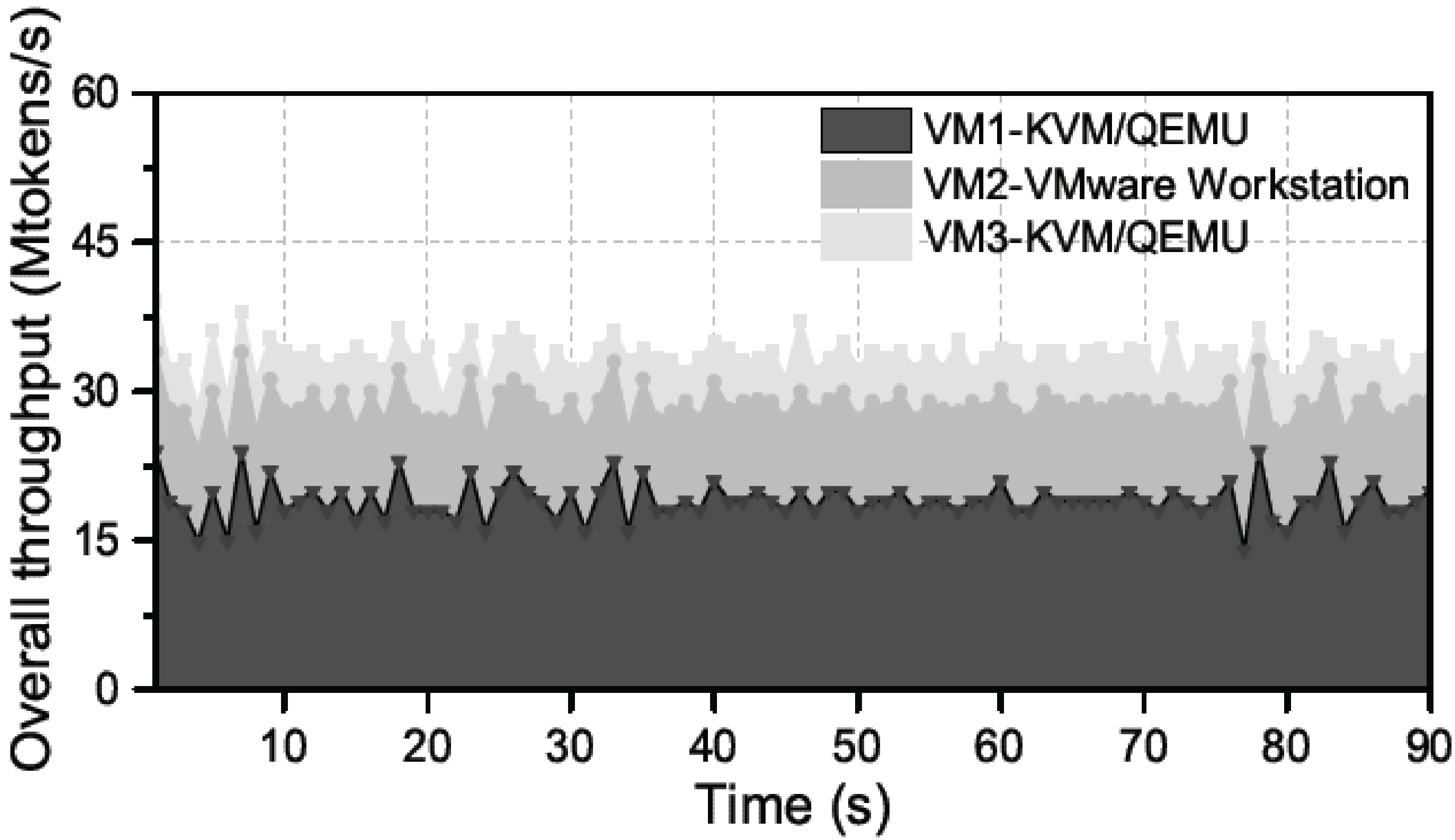
Testbed Deployment

Tenant	Workload	Read (%)	IO Sizes	Seq/rand
A (VM1~VM5)	Index	75%	64KB	rand
B (VM6~VM10)	Data	61%	8KB	rand
C (VM11~VM15)	Message	56%	64KB	rand

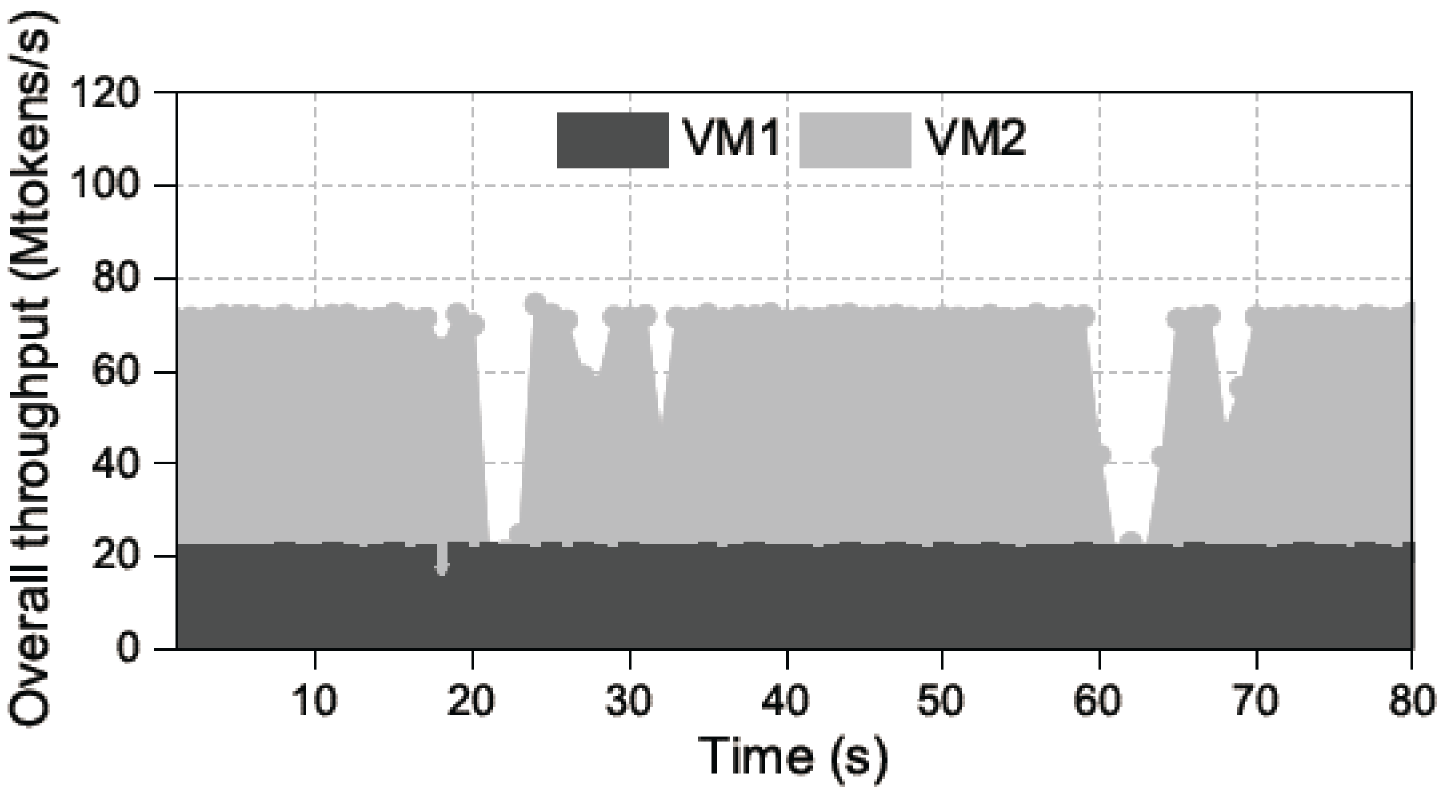
SDFS: Evaluation (one-hypervisor)



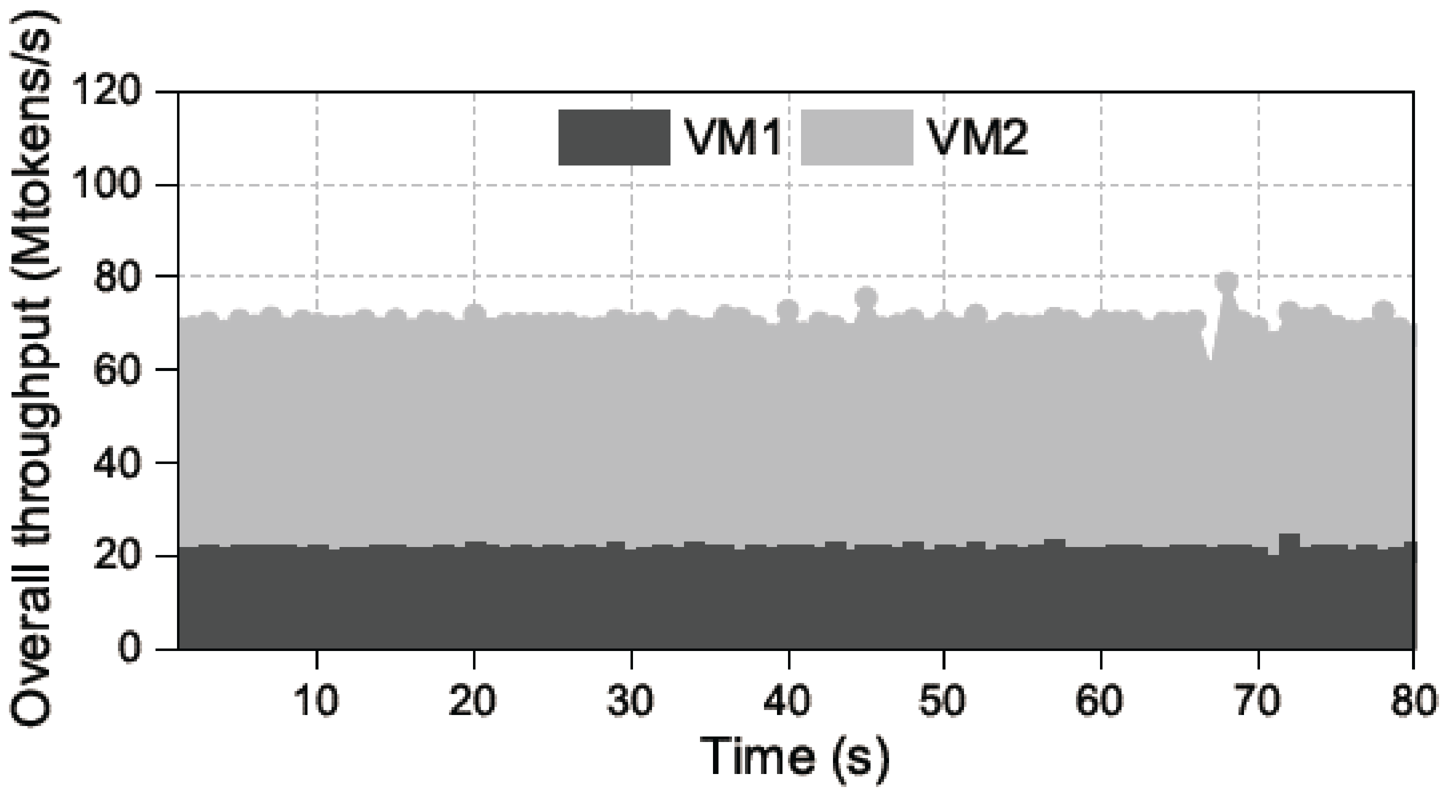
Policy Enforcement with Multi-hypervisor



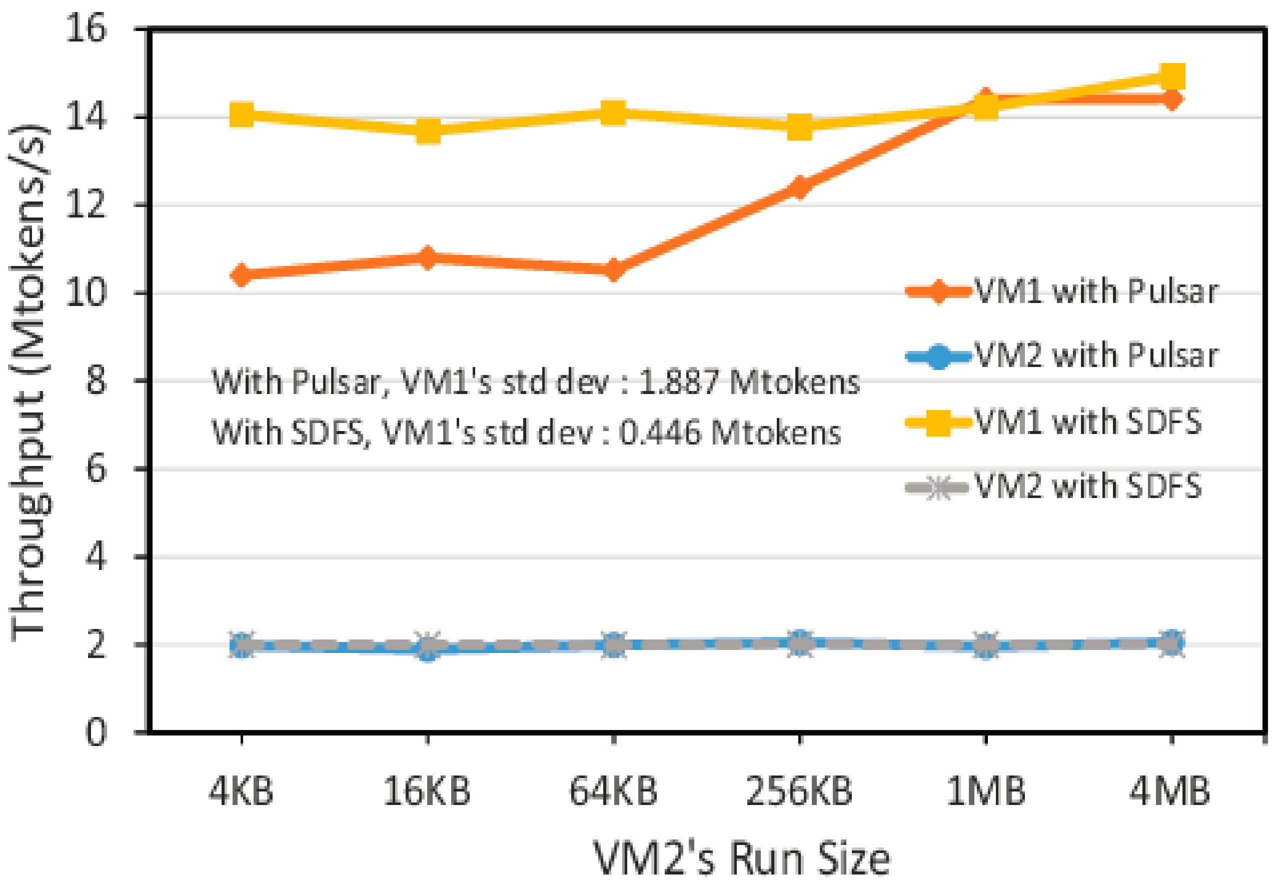
Policy Enforcement with Pulsar



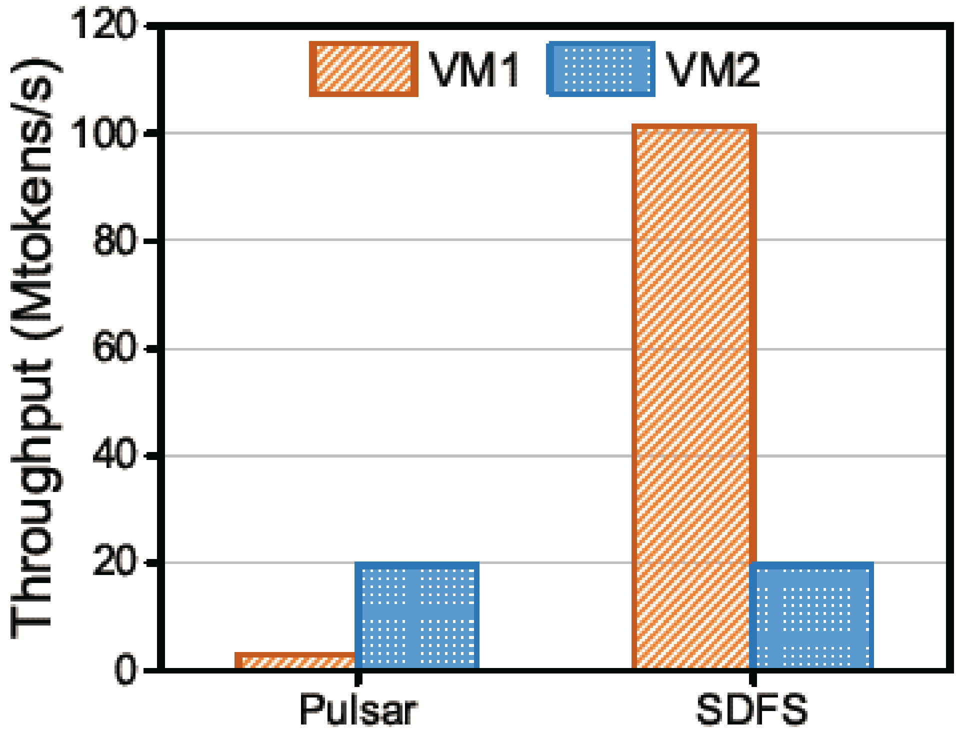
Policy Enforcement with SDFS



SDFS vs. Pulsar



SDFS vs. Pulsar



Thanks!

lfzeng@hust.edu.cn