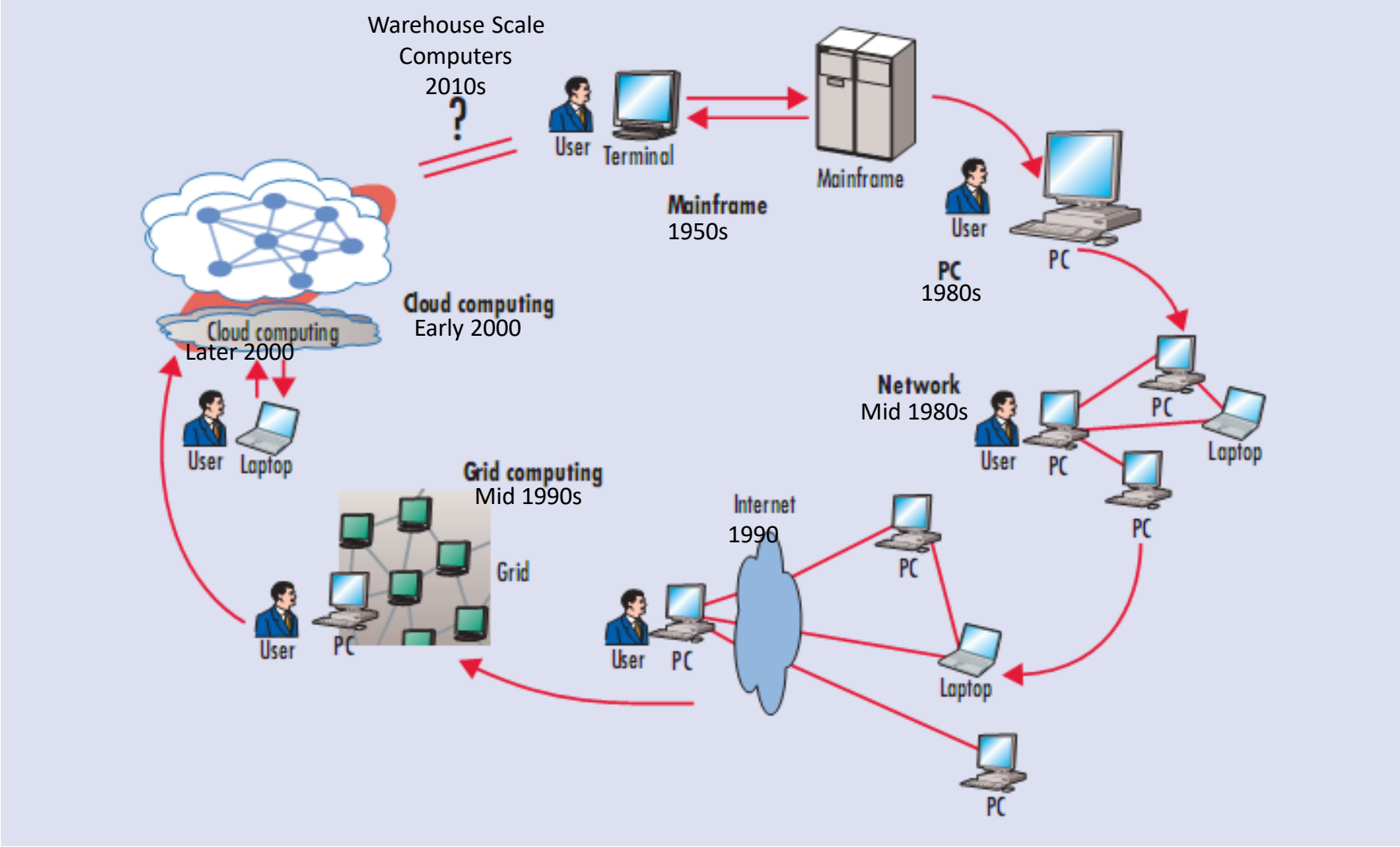
Evolution of Virtual Machines

Topic coverage

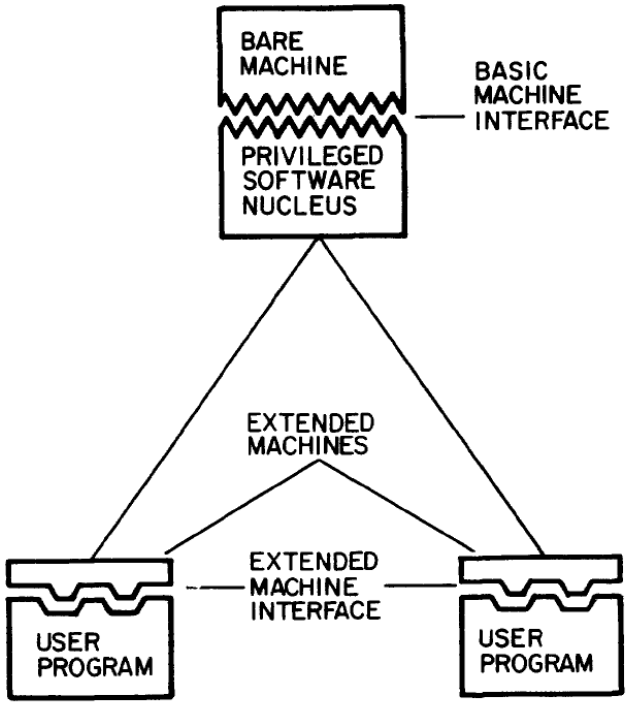
- Evolution of computing practices
- Why Virtual Machines?
- Motivation for Using Virtualization
- Data Center evolution
- Current day motivations for Virtual Machines.

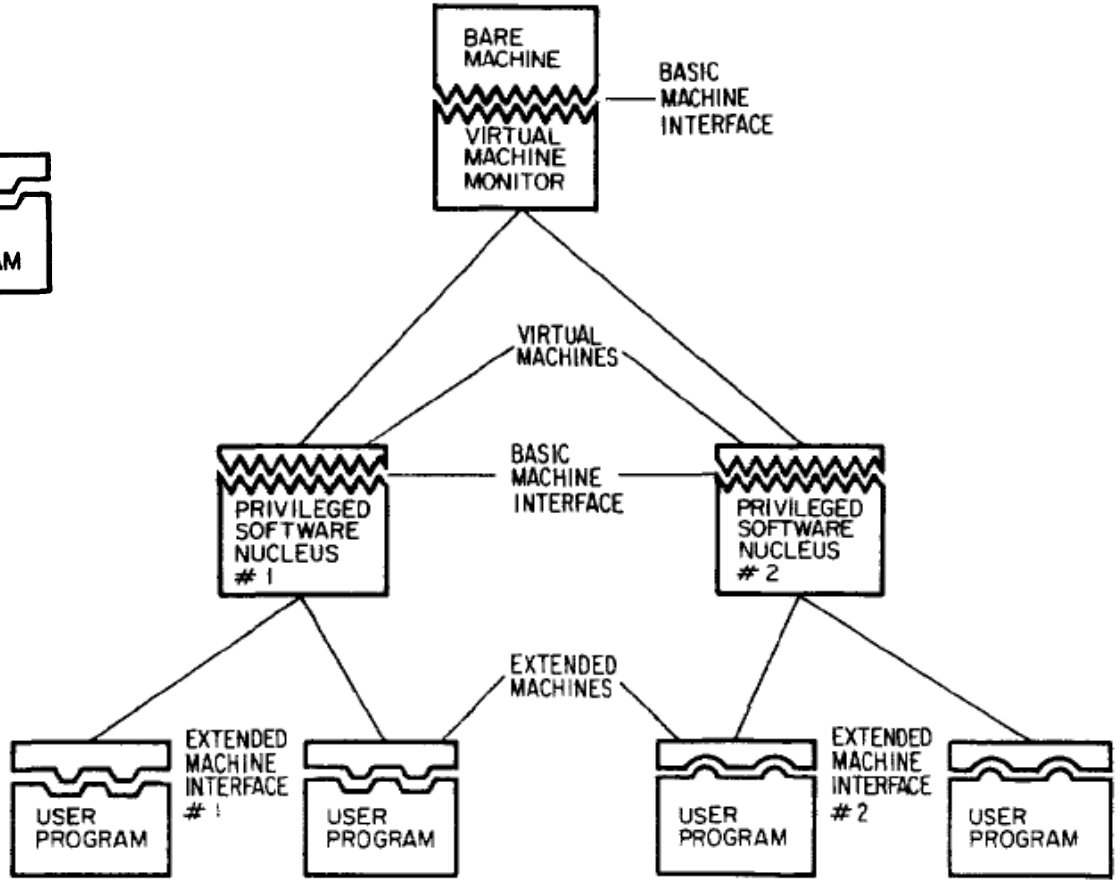
Evolution of Computing Practices



Why Virtual Machines?

- In the mainframe era (1960 – early 1980s)
  - Computer architecture change to support multi-user, multi-programming and introduction of I/O processors lead to dual-privilege processor architectures.
  - Introduced better system utilization but brought in more complexities associated with testing and development of new systems and system software.


BARE
MACHINE
BASIC
MACHINE
INTERFACE
PRIVILEGED
SOFTWARE
NUCLEUS
EXTENDED
MACHINES
EXTENDED
MACHINE
INTERFACE
USER
PROGRAM
USER
PROGRAM


BARE
MACHINE
BASIC
MACHINE
INTERFACE
VIRTUAL
MACHINE
MONITOR
VIRTUAL
MACHINES
BASIC
MACHINE
INTERFACE
PRIVILEGED
SOFTWARE
NUCLEUS
#1
PRIVILEGED
SOFTWARE
NUCLEUS
#2
EXTENDED
MACHINES
EXTENDED
MACHINE
INTERFACE
#1
EXTENDED
MACHINE
INTERFACE
#2
USER
PROGRAM
USER
PROGRAM
USER
PROGRAM
USER
PROGRAM

Idea of the Privileged Software

- The privileged software of the OS-kernel was built to manage the system resources.
- The idea of multi-programming and multi-user is the functionality provided by the Privileged software.
- System hardware has sufficient constructs to support this idea of sharing through the Privileged software.
- Most of these sharing constructs happened to evolve around the idea of time-multiplexing of a resource.
  - Computing machinery was expensive so important to keep it busy as much as possible!

Premise for System Design

- Till year 2000 most system designs adopted approaches where single OS kernel is used for managing the resources.
- The applications are built to the extended machine functionality rather than the actual machine!

Era of PCs and Network of Computers

- Dual state architectures sufficed for PCs and later for applications built on Network of computers and subsequently for Distributed systems and Grid computing model.
- Many applications got built using the client-server or distributed services model.
- In all such scenarios, independent systems with their own privileged software topped with necessary application runtime software for distributed systems was sufficient to provide the desired functionality.

Concerns of Server Sprawl

- As hardware became cheaper many applications got built with their own hardware and associated software tiers.
- This model ensured applications delivering required performance.
- Increase in throughput or availability was mostly handled using isolated duplicate or redundant servers.
- With the advent of WWW this practise exploded into server sprawl!

Symptoms of Server Sprawl

- Enterprise server utilizations below 10%!
- Huge IT Capex but the inability to host newer applications due to mismatch of software requirements!
- Applications-Platform coupling led to unwarranted dependencies between unrelated applications!
  - Side-effects: failure of one application causes outages on the other!
- Ever increasing Opex bills!
- Precursor to Green computing!

Re-emergence of Virtualization

- Improve server utilization with application isolation.
- Enables co-hosting of multiple applications on single platform, each with their independent software and runtime environment – genesis for server consolidation.
- Offers platform independence by way of Virtual Machine encapsulation
  - Enables application scalability with varying workload demands
  - Improved application availability by way of isolation and elastic scaling capability
  - Faster provisioning for newer applications

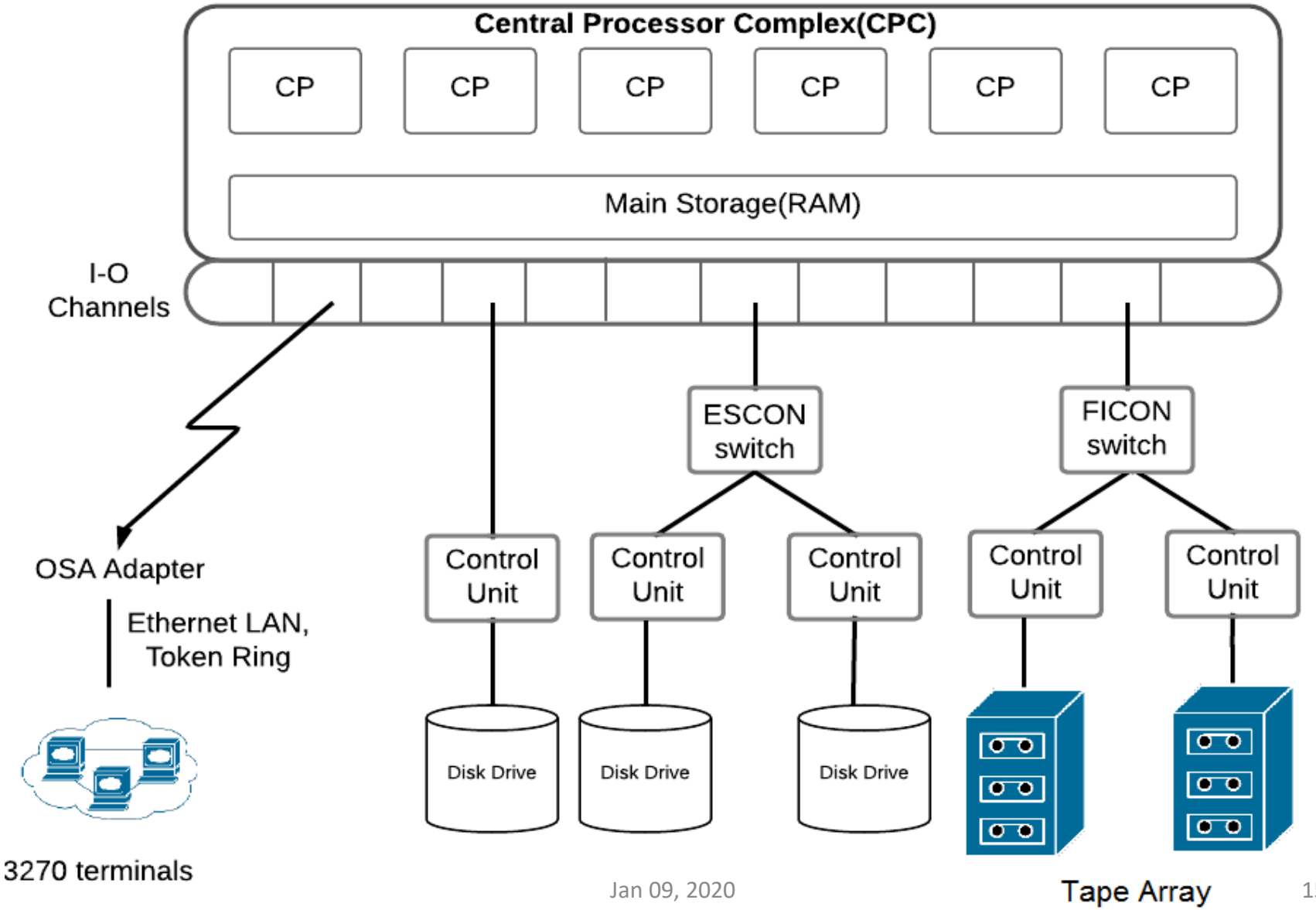
Cloud Setups

- Re-emergence of virtualization, service oriented architectures, the world wide web (Internet?) and utility computing paradigm enabled the realization of Cloud setups and cloud computing.
- Grid computing ushered in loose coupling of many, heterogeneous distributed systems through middleware (resource aggregation) for large-scale scientific computations with access from anywhere.
- Enterprises realized the benefit of segregation using virtualization with middleware for seamless use of computing infrastructure.

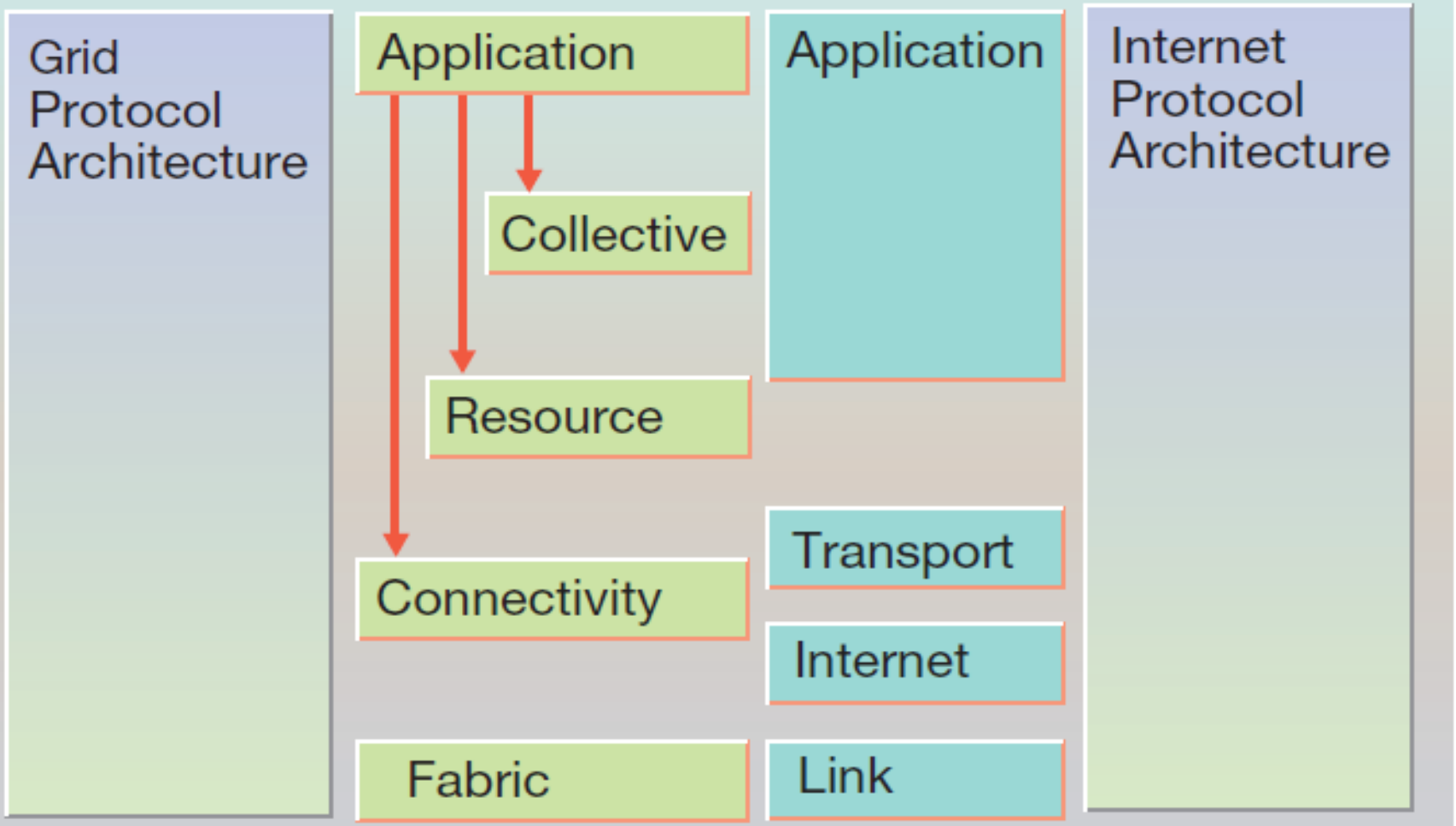
Data Center Evolution

- Mainframes:
  - Expensive hardware; virtualization enabled multi-user and multi-tenanted OS for system development
- Distributed Systems and Grid Computing: (Scientific workloads)
  - Multiple independent systems for increased throughput or availability of applications.
  - Limiting processor speeds forced users to use many cores for improved application performance using parallel programming.
- Cloud Data centres: (Enterprise workloads)
  - Server consolidation through virtualization yields improved server utilization and reduced power and real-estate footprints.
- Hyperconverged data centres:
  - Heterogeneous platforms with server virtualization results in resource fragmentation
  - Software defined data centers with commensurate hardware interconnects enable flexible hardware compositions for better utilization at data centers!

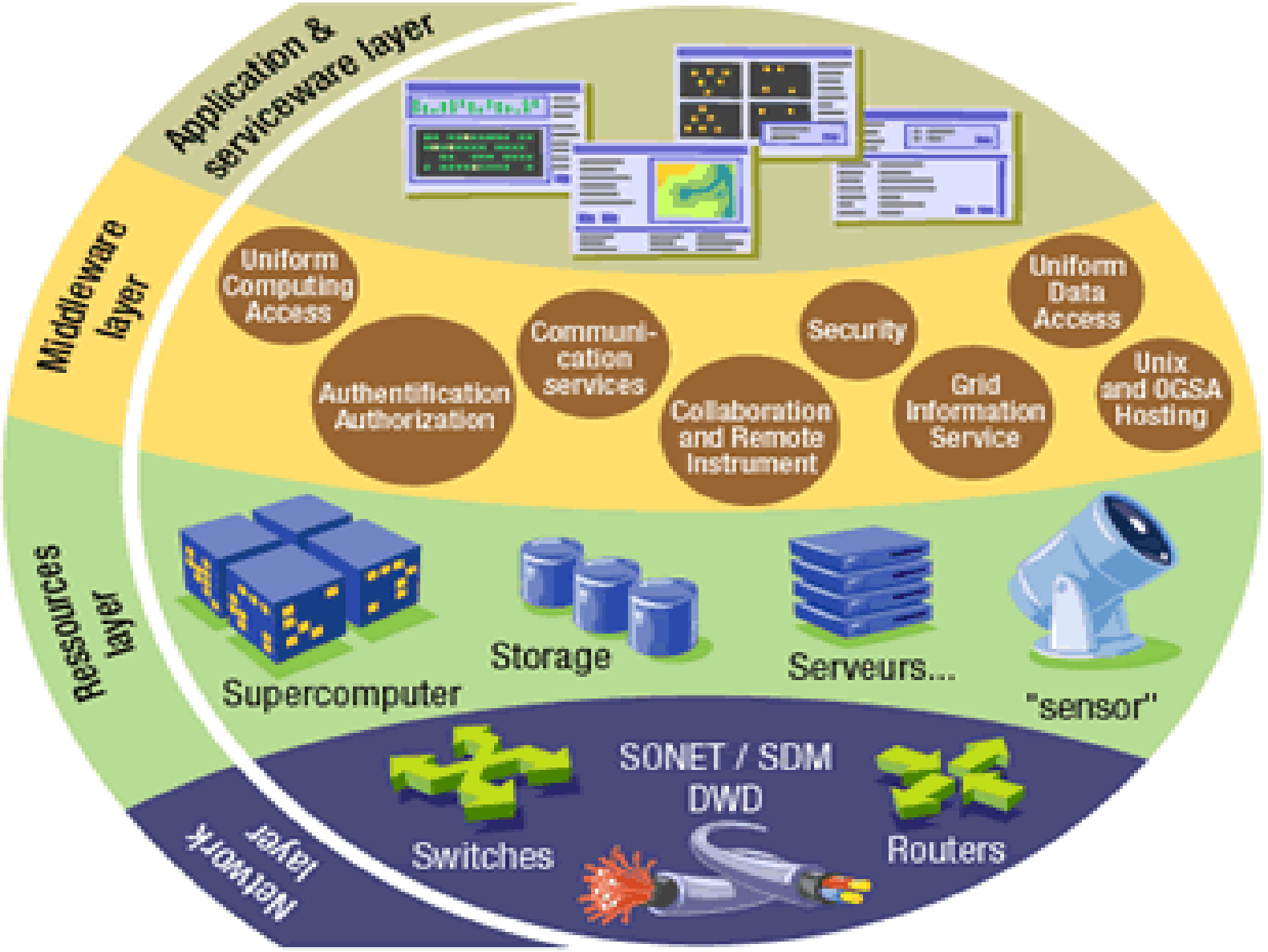
Mainframes


Central Processor Complex(CPC)
CP
CP
CP
CP
CP
CP
Main Storage(RAM)
I-O Channels
OSA Adapter
Ethernet LAN, Token Ring
3270 terminals
ESCON switch
FICON switch
Control Unit
Control Unit
Control Unit
Control Unit
Control Unit
Disk Drive
Disk Drive
Disk Drive
Tape Array

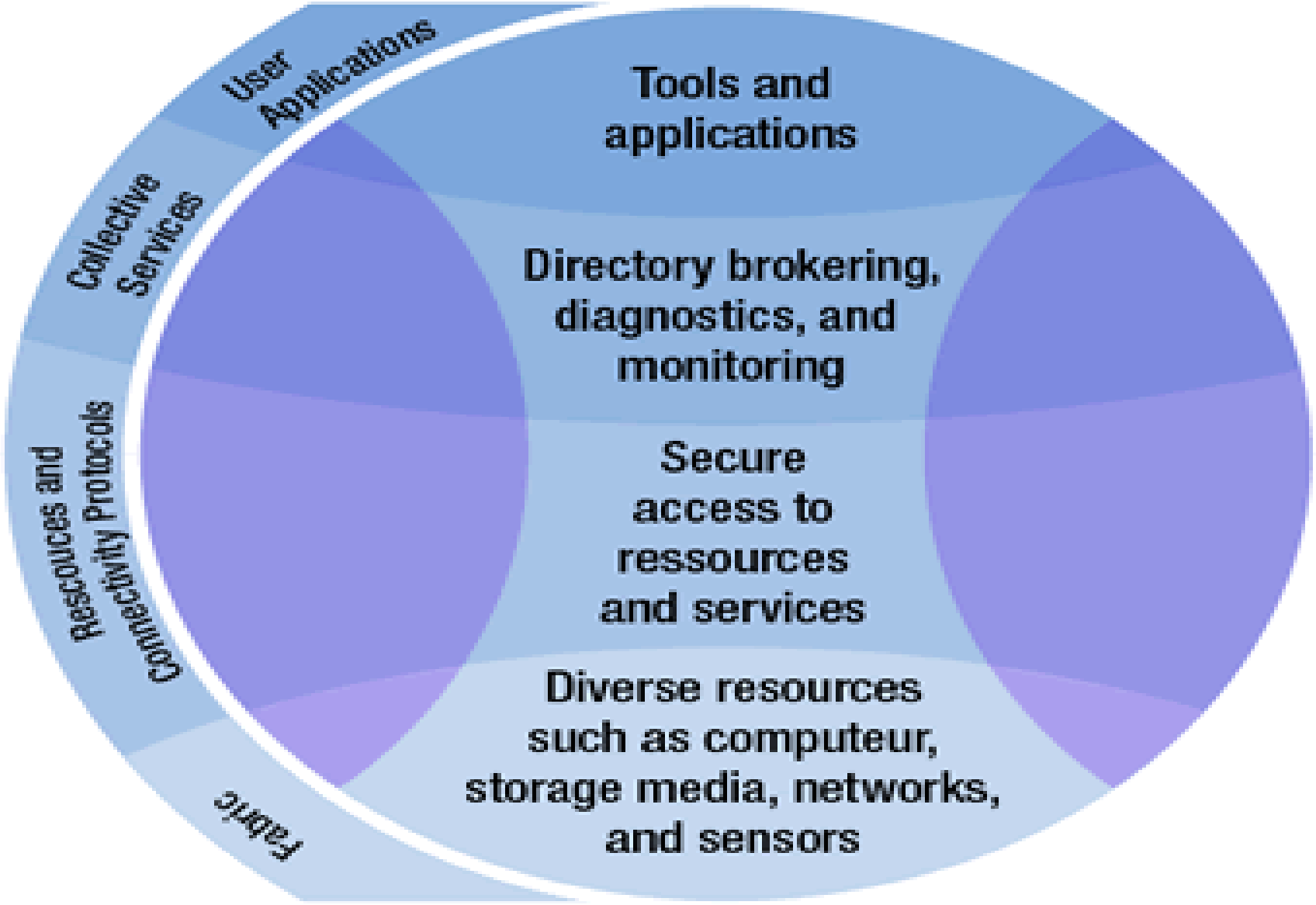
Grid Computing Architecture



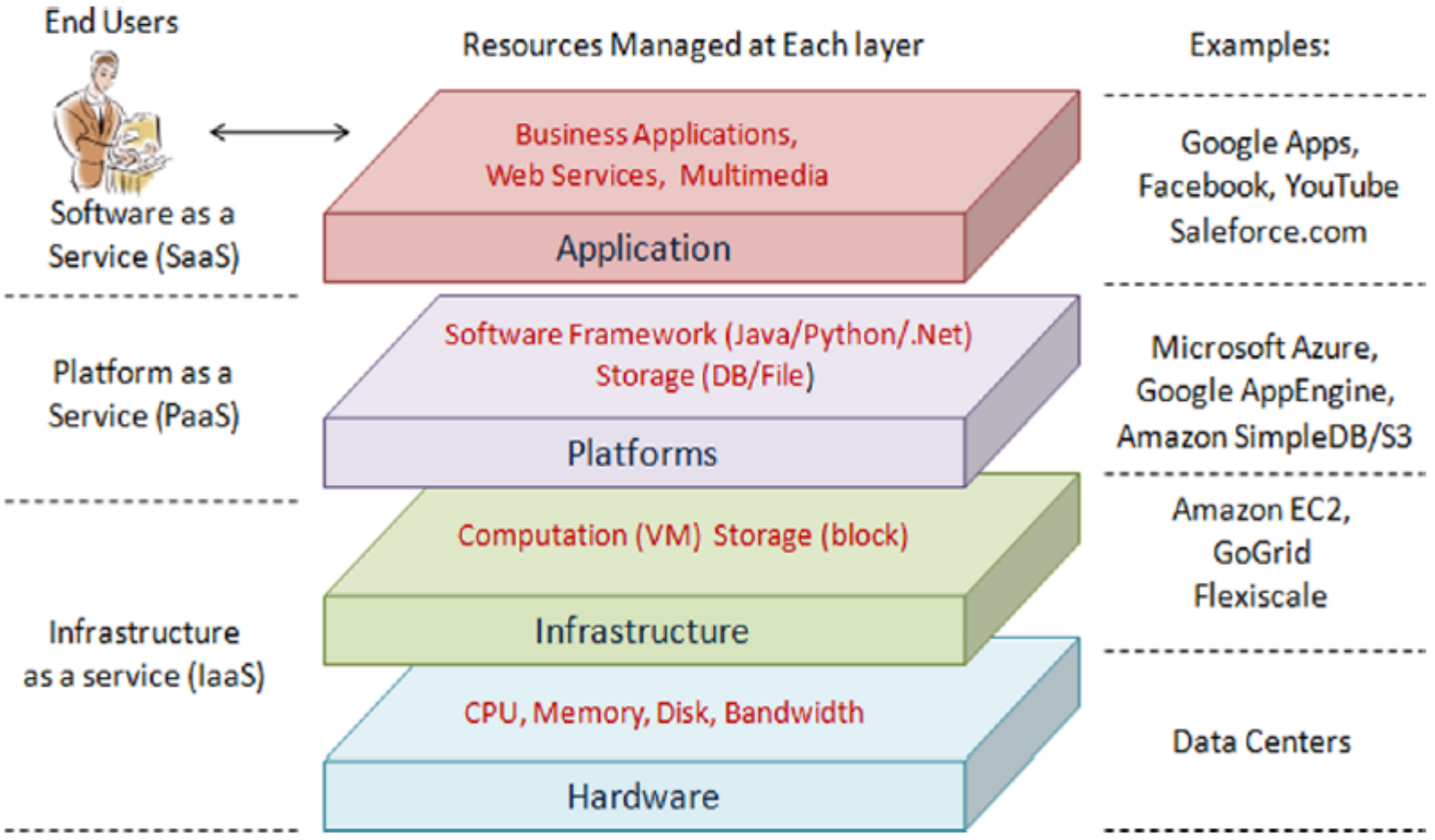
Grid computing Logical Layers



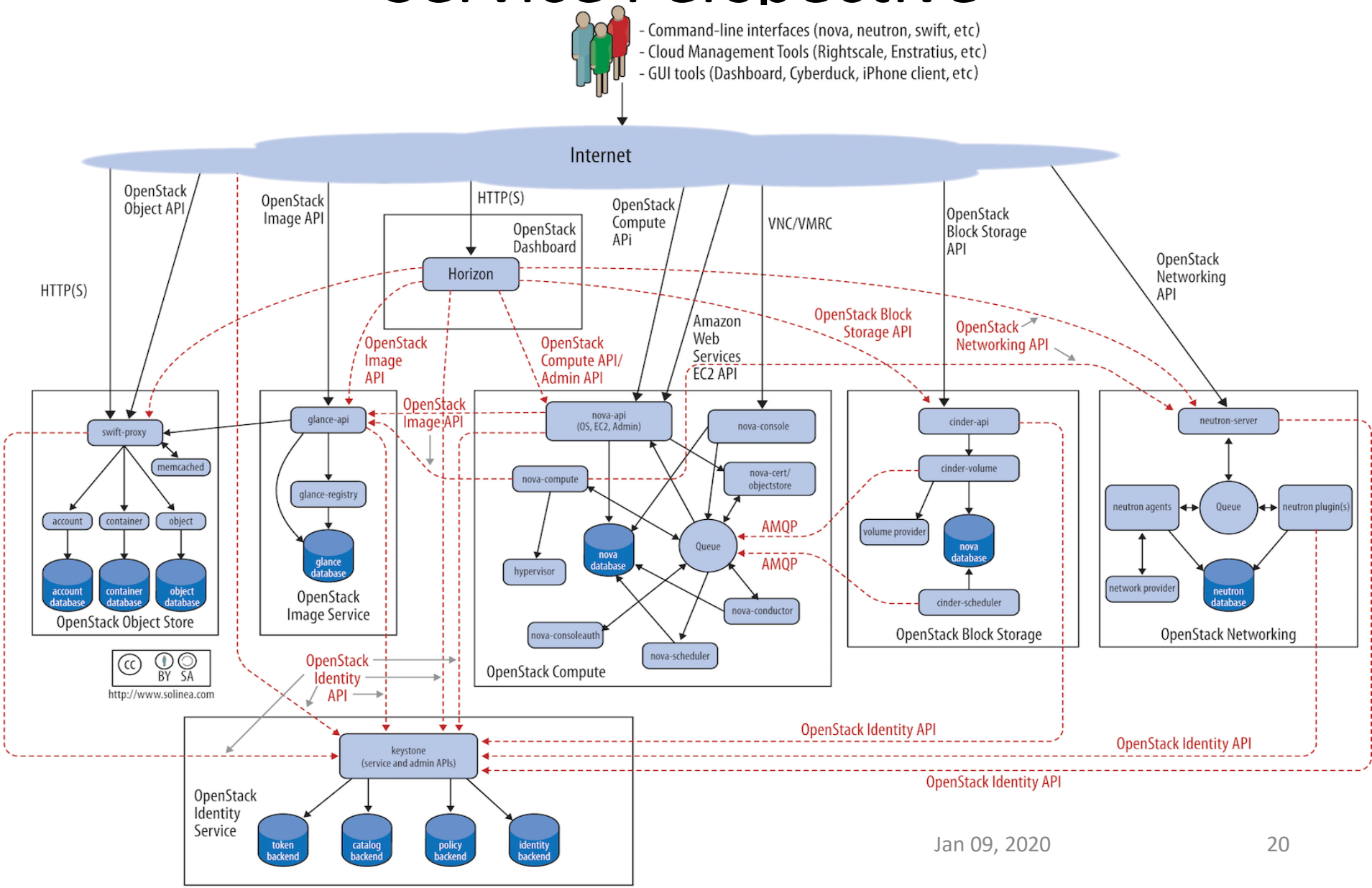
Grid Computing Service Layers



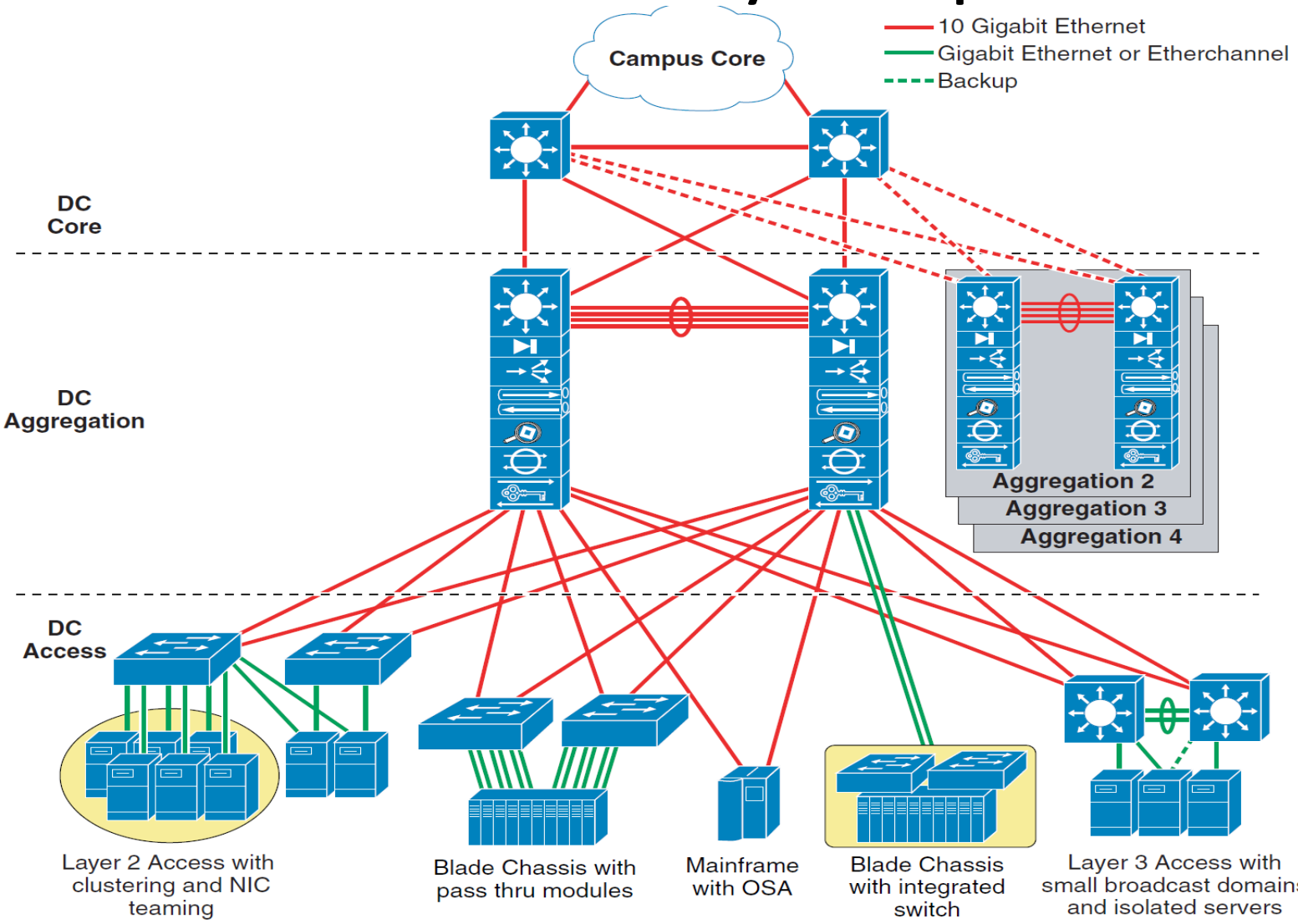
Cloud Computing Architecture
Logical and Service Layers



OpenStack Cloud Architecture Service Perspective



Data Centre Architecture
Fabric Connectivity Perspective



Source: https://www.cisco.com/c/en/us/td/docs/solutions/Enterprise/Data_Center/DC_Infra2_5/DCI_SRND_2_5a_book.pdf

Hyper-Converged Data Centres

- Integrated systems:
  - Initial Cloud setups started with virtualized servers and storage integrated using cloud stack.
  - Simple conglomerations of existing hardware and software with SAN based storage access
  - Vendor lock-in with server and storage OEMs
- Converged Infrastructure:
  - Server and storage components converged to a single appliance (VM)
  - Unified and simplified management and faster deployment
  - Resource ratios (cpu:storage:network) fixed and hence inflexible and have performance utilization conflicts.
  - Legacy applications still need to be re-provisioned or migrated to cloud infrastructure
- Hyper-converged Infrastructure:
  - Consolidation of required functionality into a single infrastructure stack implemented on simple, efficient and elastic resource pool
  - Software Defined Data Centers (SDDCs) enable the idea of convergence of hardware along-with functionalities like backup, replication, deduplication, elasticity, network gateways, high speed storage access through SSD cache and drive arrays, etc.

Convergence Characteristics

	Technical Attributes			Organizational Benefits	
	Data Efficiency	Single Shared Resource Pool	Global Management	TCO Improvements	Simplification
Convergence 1.0	☐	☐ Resource pooling limited to server layer	☐	☐	⊡ Some time to deployment and administrative gains
Convergence 2.0	☐	⊡ Limited to primary server and storage resources; other resources not included	☐	⊡ TCO gains primarily due to reduction of legacy gear; does not address backup, replication, and DR	⊡ Fewer products to manage
Convergence 3.0	■ Data architecture begins with one-time deduplication, compression, and optimization of data	■ All data center resources are brought into the resource stack	■ Complete management of all infrastructure resources and virtual machines; single point of administration	■ Major TCO gains through reduction of hardware resources, streamlined operations, and automation	■ Reduces hardware littered across data centers, eases management, VM-centricity

☐ ***Not supported*** ⊡ ***Partially Supported*** ■ ***Fully Supported***

Current-day Motivations for using Virtualization

1. Multi-Hypervisor Virtual Machines: Enabling an Ecosystem of Hypervisor-level Services

Abstract: Public cloud software marketplaces already offer users a wealth of choice in operating systems, database management systems, financial software, and virtual networking, all deployable and configurable at the click of a button. Unfortunately, this level of customization has not extended to emerging hypervisor-level services, partly because traditional virtual machines (VMs) are fully controlled by only one hypervisor at a time. Currently, a VM in a cloud platform cannot concurrently use hypervisor-level services from multiple third-parties in a compartmentalized manner. We propose the notion of a *multi-hypervisor VM*, which is an unmodified guest that can simultaneously use services from multiple coresident, but isolated, hypervisors. We present a new virtualization architecture, called *Span virtualization*, that leverages nesting to allow multiple hypervisors to concurrently control a guest's memory, virtual CPU, and I/O resources. Our prototype of Span virtualization on the KVM/QEMU platform enables a guest to use services such as introspection, network monitoring, guest mirroring, and hypervisor refresh, with performance comparable to traditional nested VMs.

https://www.usenix.org/conference/atc17/technical-sessions/presentation/gopalan

Current-day Motivations for using Virtualization

2. Preemptive, Low Latency Datacenter Scheduling via Lightweight Virtualization

Abstract: Data centers are evolving to host heterogeneous workloads on shared clusters to reduce the operational cost and achieve higher resource utilization. However, it is challenging to schedule heterogeneous workloads with diverse resource requirements and QoS constraints. On the one hand, latency-critical jobs need to be scheduled as soon as they are submitted to avoid any queuing delays. On the other hand, best-effort long jobs should be allowed to occupy the cluster when there are idle resources to improve cluster utilization. The challenge lies in how to minimize the queuing delays of short jobs while maximizing cluster utilization. Existing solutions either forcibly kill long jobs to guarantee low latency for short jobs or disable preemption to optimize utilization. Hybrid approaches with resource reservations have been proposed but need to be tuned for specific workloads.

In this paper, we propose and develop BIG-C, a container-based resource management framework for Big Data cluster computing. The key design is to leverage lightweight virtualization, a.k.a, containers to make tasks preemptable in cluster scheduling. We devise two types of preemption strategies: *immediate* and *graceful* preemptions and show their effectiveness and tradeoffs with loosely-coupled MapReduce workloads as well as iterative, in-memory Spark workloads. Based on the mechanisms for task preemption, we further develop a preemptive fair share cluster scheduler. We have implemented BIG-C in YARN. Our evaluation with synthetic and production workloads shows that low-latency and high utilization can be both attained when scheduling heterogeneous workloads on a contended cluster.

https://www.usenix.org/conference/atc17/technical-sessions/presentation/chen-wei

Current-day Motivations for using Virtualization

3. Lightweight Virtualization (LV) for IoT Edge Computing: Edge computing deals with building viable software constructs to handle IoT data near source. This paper discusses and compares the applicability of two LV technologies, containers and unikernels, as platforms for enabling the scalability, security, and manageability required by such pervasive applications.

   R. Morabito, V. Cozzolino, A. Y. Ding, N. Beijar and J. Ott, "Consolidate IoT Edge Computing with Lightweight Virtualization," in IEEE Network, vol. 32, no. 1, pp. 102-111, Jan.-Feb. 2018.

   doi: 10.1109/MNET.2018.1700175

Current-day Motivations for using Virtualization

4. Understanding Security Implications of Using Containers in the Cloud

Abstract: Container technology is being adopted as a mainstream platform for IT solutions because of high degree of agility, reusability and portability it offers. However, there are challenges to be addressed for successful adoption. First, it is difficult to establish the full pedigree of images downloaded from public registries. Some might have vulnerabilities introduced unintentionally through rounds of updates by different users. Second, non-conformance to the immutable software deployment policies, such as those promoted by the DevOps principles, introduces vulnerabilities and the loss of control over deployed software. In this study, we investigate containers deployed in a production cloud to derive a set of recommended approaches to address these challenges. Our analysis reveals evidences that (i), images of unresolved pedigree have introduced vulnerabilities to containers belonging to third parties; (ii), updates to live public containers are common, defying the tenet that deployed software is immutable; and (iii), scanning containers or images alone is insufficient to eradicate vulnerabilities from public containers. We advocate for better systems support for tracking image provenance and resolving disruptive changes to containers, and propose practices that container users should adopt to limit the vulnerability of their containers.

https://www.usenix.org/conference/atc17/technical-sessions/presentation/tak

Current-day Motivations for using Virtualization

5. Unikernels: Unikernels are single-purpose appliances that are compile-time specialised into standalone kernels, and sealed against modification when deployed to a cloud platform. In return they offer significant reduction in image sizes, improved efficiency and security, and should reduce operational costs.

Anil Madhavapeddy, Richard Mortier, Charalampos Rotsos, David Scott, Balraj Singh, Thomas Gazagnaire, Steven Smith, Steven Hand, and Jon Crowcroft. 2013. Unikernels: library operating systems for the cloud. SIGARCH Comput. Archit. News 41, 1 (March 2013), 461-472. DOI: https://doi.org/10.1145/2490301.2451167

Current-day Motivations for using Virtualization

6. KylinX: A Dynamic Library Operating System for Simplified and Efficient Cloud Virtualization

Abstract:Unikernel specializes a minimalistic LibOS and a target application into a standalone single-purpose virtual machine (VM) running on a hypervisor, which is referred to as (virtual) appliance. Compared to traditional VMs, Unikernel appliances have smaller memory footprint and lower overhead while guaranteeing the same level of isolation. On the downside, Unikernel strips off the process abstraction from its monolithic appliance and thus sacrifices flexibility, efficiency, and applicability.

This paper examines whether there is a balance embracing the best of both Unikernel appliances (strong isolation) and processes (high flexibility/efficiency). We present KylinX, a dynamic library operating system for simplified and efficient cloud virtualization by providing the pVM (process-like VM) abstraction. A pVM takes the hypervisor as an OS and the Unikernel appliance as a process allowing both page-level and library-level dynamic mapping. At the page level, KylinX supports pVM fork plus a set of API for inter-pVM communication (IpC). At the library level, KylinX supports shared libraries to be linked to a Unikernel appliance at runtime. KylinX enforces mapping restrictions against potential threats. KylinX can fork a pVM in about 1.3 ms and link a library to a running pVM in a few ms, both comparable to process fork on Linux (about 1 ms). Latencies of KylinX IpCs are also comparable to that of UNIX IPCs.

https://www.usenix.org/conference/atc18/presentation/zhang-yiming

Current-day Motivations for using Virtualization

7. Cntr: Lightweight OS Containers

Abstract:Container-based virtualization has become the de-facto standard for deploying applications in data centers. However, deployed containers frequently include a wide-range of tools (e.g., debuggers) that are not required for applications in the common use-case, but they are included for rare occasions such as in-production debugging. As a consequence, containers are significantly larger than necessary for the common case, thus increasing the build and deployment time.

Cntr provides the performance benefits of lightweight containers and the functionality of large containers by splitting the traditional container image into two parts: the “fat” image — containing the tools, and the “slim” image — containing the main application. At run-time, Cntr allows the user to efficiently deploy the “slim” image and then expand it with additional tools, when and if necessary, by dynamically attaching the “fat” image.

To achieve this, Cntr transparently combines the two container images using a new nested namespace, without any modification to the application, the container manager, or the operating system. We have implemented Cntr in Rust, using FUSE, and incorporated a range of optimizations. Cntr supports the full Linux filesystem API, and it is compatible with all container implementations (i.e., Docker, rkt, LXC, systemd-nspawn). Through extensive evaluation, we show that Cntr incurs reasonable performance overhead while reducing, on average, by 66.6% the image size of the Top-50 images available on Docker Hub.

https://www.usenix.org/conference/atc18/presentation/thalheim

Current-day Motivations for using Virtualization

8. A Retargetable System-Level DBT Hypervisor

Abstract: System-level Dynamic Binary Translation (DBT) provides the capability to boot an Operating System (OS) and execute programs compiled for an Instruction Set Architecture (ISA) different to that of the host machine. Due to their performance-critical nature, system-level DBT frameworks are typically hand-coded and heavily optimized, both for their guest and host architectures. While this results in good performance of the DBT system, engineering costs for supporting a new, or extending an existing architecture are high. In this paper we develop a novel, retargetable DBT hypervisor, which includes guest specific modules generated from high-level guest machine specifications. Our system simplifies retargeting of the DBT, but it also delivers performance levels in excess of existing manually created DBT solutions. We achieve this by combining offline and online optimizations, and exploiting the freedom of a Just-in-time (JIT) compiler operating in a bare-metal environment provided by a Virtual Machine. We evaluate our DBT using both targeted micro-benchmarks as well as standard application benchmarks, and we demonstrate its ability to outperform the de-facto standard Qemu DBT system. Our system delivers an average speedup of 2.21x over Qemu across SPEC CPU2006 integer benchmarks running in a full-system Linux OS environment, compiled for the 64-bit ARMv8-A ISA, and hosted on an x86-64 platform. For floating-point applications the speedup is even higher, reaching 6.49x on average. We demonstrate that our system-level DBT system significantly reduces the effort required to support a new ISA, while delivering outstanding performance.

https://www.usenix.org/conference/atc19/presentation/spink

Summary

- Evolution of computing practices
  - Usage Perspective
  - System Architecture perspective
- Why Virtual Machines?
- Motivation for Using Virtualization
- Data Center evolution
  - Why it is relevant to understand system virtualization
- Current day motivations for Virtual Machines

Questions?

Thankyou!