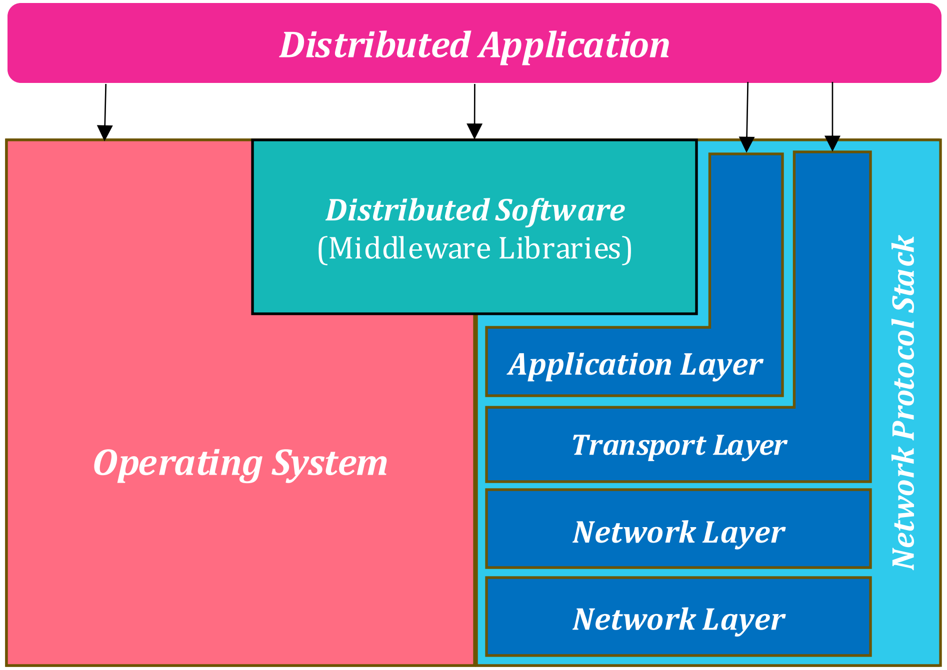
Introduction

Distributed Systems

Distributed Systems

- Components located at networked computers (with geographical separation)
- Characteristics
  - Lack of Global Clock
    - Action Coordination
  - No Shared Memory
    - Communication (message passing)
  - Geographical Separation
    - Network
  - Autonomy & Heterogeneity



Reference: *Distributed Computing: Principles, Algorithms, and Systems, Kshemkalyani & Singhal (2008)*

Distributed Systems: Examples

- Web Search
- Web-based services
  - Multimedia
  - Telemedicine
  - E-learning
- Industrial IoT
  - Transport and Logistics
  - Environmental Monitoring
- Financial Trading
- Massively Multiplayer Online Games (MMOGs)

Reference: *Distributed Systems: Concepts and Design, Coulouris, Dollimore, Kindberg, and Blair (2012)*

Distributed Systems

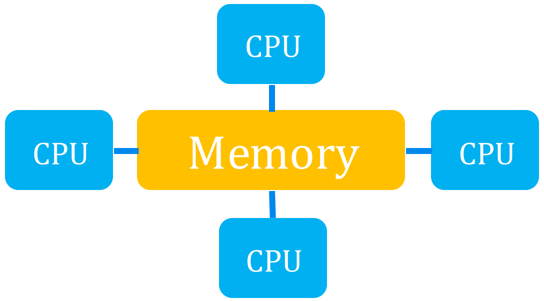
- Collaboratively achieve a common goal

- Why do we need them?
    - Inherently distributed computations
    - Resource sharing
    - Access to remote resources
    - Enhanced reliability
    - Increased performance to cost ratio
    - Scalability
    - Modularity & incremental extendibility

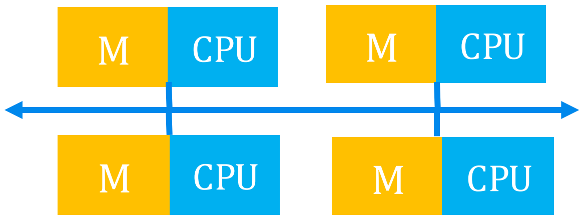
Reference: *Distributed Computing: Principles, Algorithms, and Systems, Kshemkalyani and Siinghal (2008)*

Parallel Systems

- Multi-processor system
  - No Common Clock
  - **Access to Shared Memory (Unified Memory Access: UMA)**
  - Interconnect: Omega Network, Butterfly Network, Clos, Shuffle-Exchange



- Array processors
  - **Common Clock**
  - No Shared Memory



- Multi-computer parallel system
  - No Common Clock
  - No Shared Memory (Non-Unified Memory Access: NUMA)

__Reference__: Distributed Computing: Principles, Algorithms, and Systems, Kshemkalyani & Singhal (2008)

Distributed Program

- **Coupling:** Interdependency and binding among modules
  - Tight or Loose
- **Parallelism:** Speedup on a specific system
  - Ratio of the time $T(1)$ with a single processor to the time $T(n)$ with n processors
- **Concurrency**
  - Ratio of the number of local operations to the total number of operations
- **Granularity**
  - Ratio of the amount of computation to the amount of communication
    - Coarse (More CPU instructions to communication), Fine (Fewer)

Reference: Distributed Computing: Principles, Algorithms, and Systems, Kshemkalyani & Singhal (2008)

Operating Systems

- Network Operating System
  - Operating System running on **loosely** coupled processors running **loosely** coupled software

- Distributed Operating System
  - Operating System running on **loosely** coupled processors running **tightly** coupled software

- Multi-Processor Operating System
  - Operating System running on **tightly** coupled processors running **tightly** coupled software

__Reference__: Distributed Computing: Principles, Algorithms, and Systems, Kshemkalyani & Singhal (2008)

Communication

- Communication
  - Shared Memory: Common Shared Address Space
    - Communication through shared data and control variables
    - Easier than message passing
  - **Message Passing**

- Emulation
  - Message passing on a shared memory system (MP → SM)
    - Address space partitioning
    - Send and Receive operations among partitions
  - Shared memory on a message-passing system (SM → MP)
    - Emulation of shared location as a separate process
    - Use of Send and Receive for Write and Read operations
    - Distributed Shared Memory

***Reference**: Distributed Computing: Principles, Algorithms, and Systems, Kshemkalyani & Singhal (2008)*

Distributed Systems: Challenges

- Heterogeneity (network, hardware, OS, programming languages, development)
- Openness (key interfaces)
- Security (confidentiality, integrity, availability)
- Scalability (response to increased load)
- Failures (detecting, tolerating, recovery)
- Concurrency (ensuring consistency in concurrency)
- Transparency (concealment of details from the user)
- Quality of Service

Reference: *Distributed Systems: Concepts and Design, Coulouris, Dollimore, Kindberg, and Blair (2012)*

The Cloud

Cloud Computing: What is it?

- **Distributed** Computing as a **utility**
  - The illusion of infinite computing resources available on demand
  - The elimination of an up-front commitment by users
  - Ability to pay for use of computing resources in a short-term basis as needed and release them as needed

- **Applications** delivered as **services** over the internet

- **NIST (National Institute of Standards and Technology)** Definition

  "a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model is composed of ***five essential characteristics****,* ***three service models****, and* ***four deployment models****."*

Reference: *https://csrc.nist.gov/publications/detail/sp/800-145/final (2011)* ***and*** *Above The Clouds: A Berkeley View of Cloud Computing (2009)*

Cloud: Characteristics

- On-demand Self-service
- Broad Network Access
- Resource Pooling
- Rapid Elasticity
- Measured Services

Cloud: Deployment Models

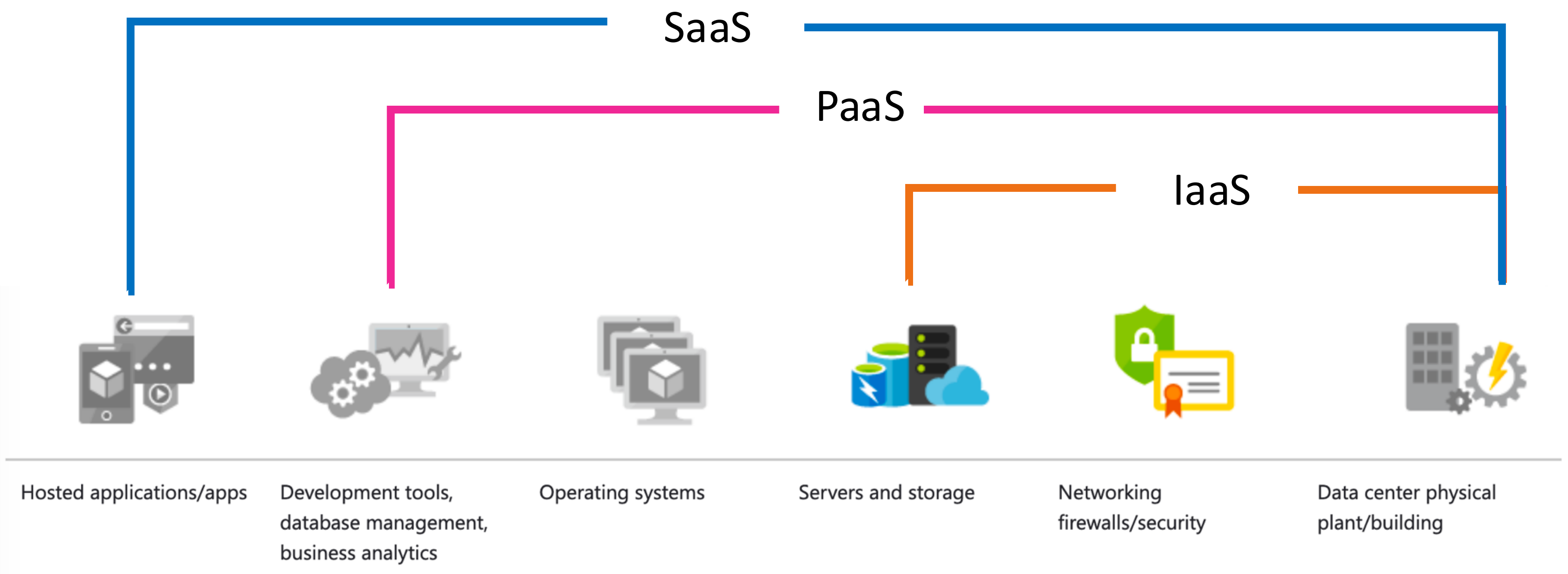
- Private Cloud
- Community Cloud
- Public Cloud
- Hybrid Cloud

- When Utility Computing preferable to Private Cloud?
    - Parallel batch processing
    - When demand is unknown in advance
    - When demand for a service varies with time

Reference: *https://csrc.nist.gov/publications/detail/sp/800-145/final (2011)* ***and*** *Above The Clouds: A Berkeley View of Cloud Computing (2009)*

Cloud: Service Models

Cloud computing
- Application + Cloud = **SaaS** (Software as a service)
- Platform + Cloud = **PaaS** (Platform as a service)
- Infrastructure + Cloud = **IaaS** (Infrastructure as a service)



Reference: *https://csrc.nist.gov/publications/detail/sp/800-145/final*
Picture [edited] from: *https://azure.microsoft.com/en-ca/overview/what-is-iaas/*

Cloud Services

Data	Data	Data	Data	Data	Data	Data
Application	Application	Application	Application	Application	Application	Application
Databases	Databases	Databases	Databases	Databases	Databases	Databases
Containers (Optional)	Containers (Optional)	Containers (Optional)	Containers (Optional)	**Containers**	Containers (Optional)	Containers (Optional)
Operating System	Operating System	Operating System	Operating System	Operating System	Operating System	Operating System
Virtualization	Virtualization	Virtualization	Virtualization	Virtualization	Virtualization	Virtualization
Physical Servers	Physical Servers	Physical Servers	Physical Servers	Physical Servers	Physical Servers	Physical Servers
Network & Storage	Network & Storage	Network & Storage	Network & Storage	Network & Storage	Network & Storage	Network & Storage
Data Center	Data Center	Data Center	Data Center	Data Center	Data Center	Data Center
On-Premises	Bare Metal	Hosting	IaaS	CaaS	PaaS	SaaS

Cloud System Stack (Software Layers)

- Application-Level Software
  - Software that implements a specific service
- Monitoring & Development Software
  - Software that keeps track of system health and availability by monitoring application performance, identifying system bottlenecks, and measuring cluster health
- Cluster-Level Software
  - Collection of distributed systems managing resources & providing services at the cluster level
    - OS of a data center
  - Examples: Distributed file systems, Remote Procedure Calls, Schedulers, programming models
    - MapReduce, Hadoop, Dynamo, Chubby, Dryad, BigTable, etc.
- Platform-Level Software
  - Common firmware, kernel, OS distribution, and libraries expected to be present in all servers to abstract hardware of a single machine & provide basic machine abstraction layer

Reference: *The Data Center As a Computer, Third Edition, Chapter 2 (2019)*

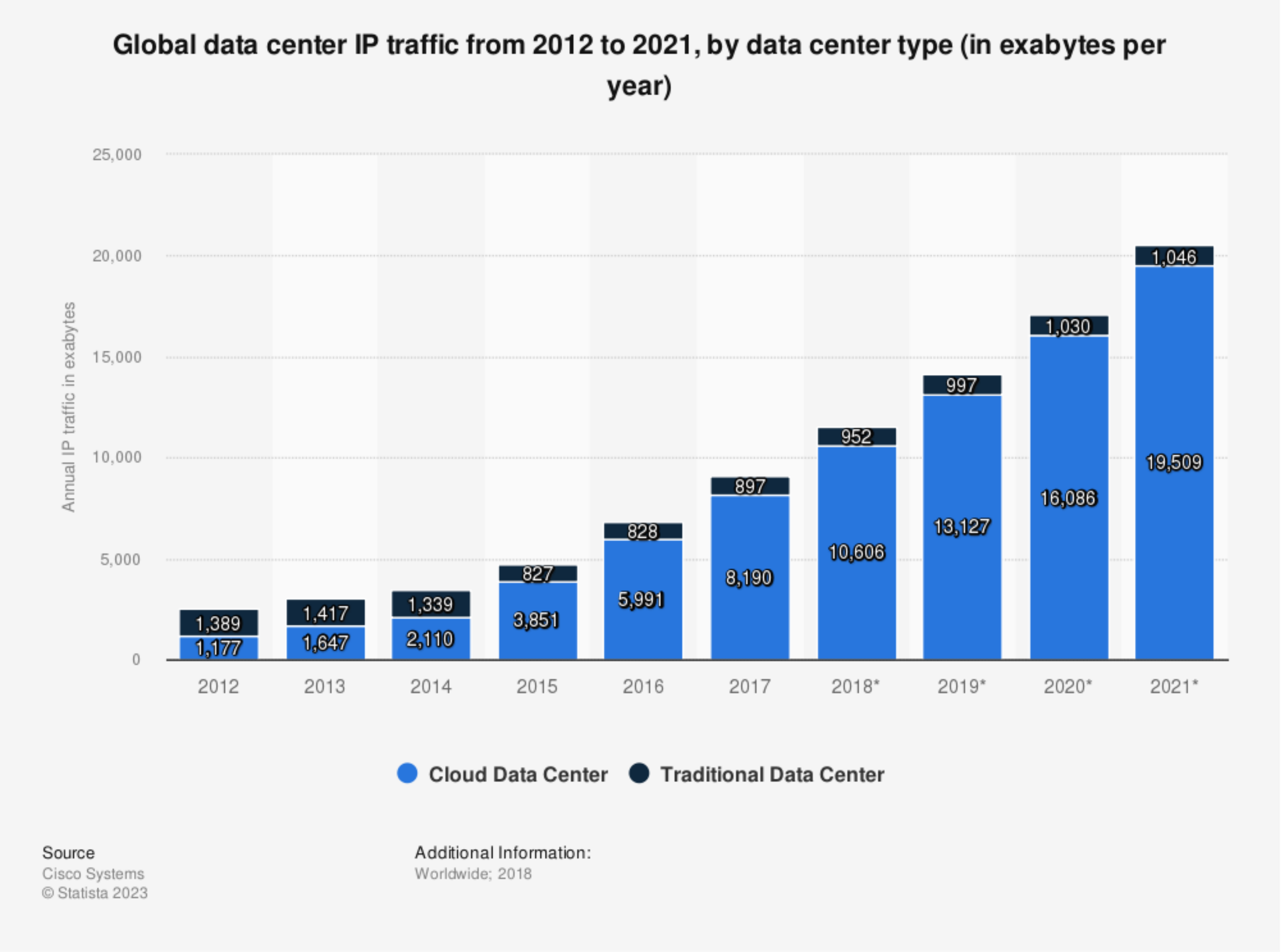
Conventional Cloud Computing

- Migration to the cloud of
  - More successful for batch style workloads (MapReduce, High Performance Computing)
  - Less successful for stateful services (Database Management Systems)

- Developers had to deal with
  - Redundancy for availability
  - Geographic distribution of redundant copies
  - Load balancing and request routing
  - Autoscaling in response to changes in load to scale up or down the system.
  - Monitoring
  - Logging for debugging or performance tuning
  - System upgrades, including security patching
  - Migration to new instances as they became available

Reference: *Cloud Programming Simplified: A Berkeley View on Serverless Computing (2019)*

Data Center Traffic Growth



Reference: *Statistica 2022 (original linked document: Cisco Global Cloud Index: 2018)*

Data Center Networks

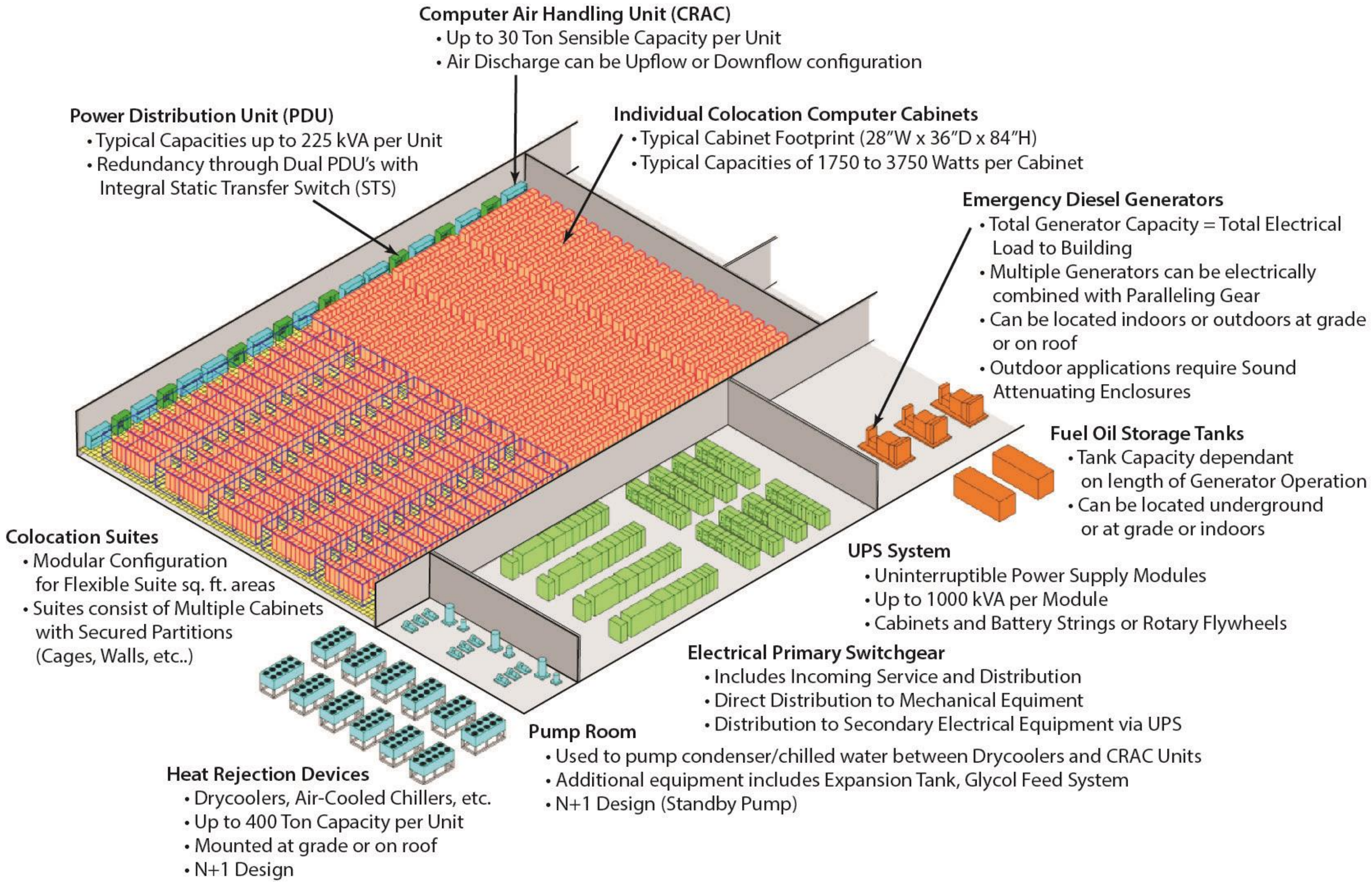
- Tens to hundreds of thousands of hosts, often closely coupled, in close proximity
  - E-business (Amazon)
  - Content-servers (YouTube, Akamai, Apple, Microsoft)
  - Search engines, data mining (Google)

Google Douglas County, Georgia data center

Microsoft, Chicago data center

Facebook, Mexico data center

Data Center



***Reference**: The Datacenter as a Computer (Second Edition)*

Data Center Costs

Amortized Cost	Component	Sub-Components
~45%	Servers	CPU, memory, disk
~25%	Power infrastructure	UPS, cooling, power distribution
~15%	Power draw	Electrical utility costs
~15%	Network	Switches, links, transit

Reference: *The Cost of a Cloud: Research Problems in Data Center Networks (SIGCOMM 2009)*

Cloud Resources

- Cloud Resources
  - Compute (CPU and Memory)
  - Store (Disk)
  - Network
- Cloud Applications
  - Content (Store & Network)
  - Events (Network)
  - Computing (Compute)
  - Data Processing (Compute & Store)

Resource Allocation Challenges

- Multiple applications serving massive numbers of clients
- Managing and balancing load
- Avoiding processing
- Networking
- Data bottlenecks

Reference: *Computer Networking : A Top-Down Approach. James F. Kurose, Keith W. Ross, 7th Edition, Pearson, 2017*

Cloud Resources

- Instance Types
  - General Purpose
    Web servers, code repositories
  - Compute Optimized (vCPUs)
    Media Transcoding, High Performance Computing (HPC), Scientific Modeling, Machine Learning (ML)
  - Memory Optimized
    Big Data Processing
  - Accelerated Computing (includes GPUs)
    Graphic processing, floating point computations, data pattern matching
  - Storage Optimized (IOPS optimized)
    High sequential read and write access to very large data sets
  - HPC Optimized
    Deep learning workloads

Reference: *Amazon Instance offering at https://aws.amazon.com/ec2/instance-types/ (Accessed January 2024)*

Elasticity

- Application Needs
  - Model of Computation
  - Model of Storage
  - Model of Communication

$$\underbrace{UserHours_{Cloud} \times (revenue - Cost_{cloud})}_{\text{Expected Profit}} \geq \underbrace{UserHours_{Cloud} \times \left(revenue - \frac{Cost_{datacenter}}{Utilization}\right)}_{\text{Factoring the cost of utilization}}$$

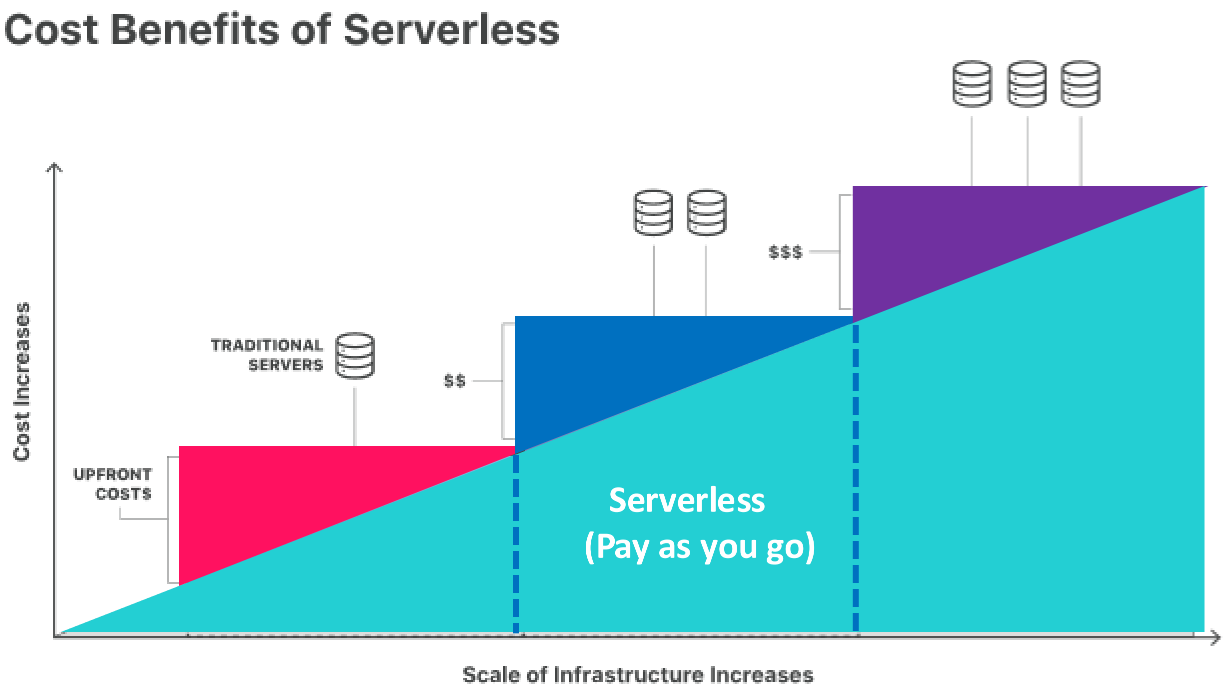
Reference: *Above The Clouds: A Berkeley View of Cloud Computing (2009)*

Cloud Resources: Serverless

- Developers to purchase infrastructure services on a flexible pay-as-you-go basis
  - Lower costs: Pay for value
  - Resource Elasticity

- Servers still used!
  - Usage managed by cloud provider

- Serverless means that developers can do their work without having to worry about server and resource provisioning



Reference (including [edited] figure): https://www.cloudflare.com/learning/serverless/what-is-serverless/

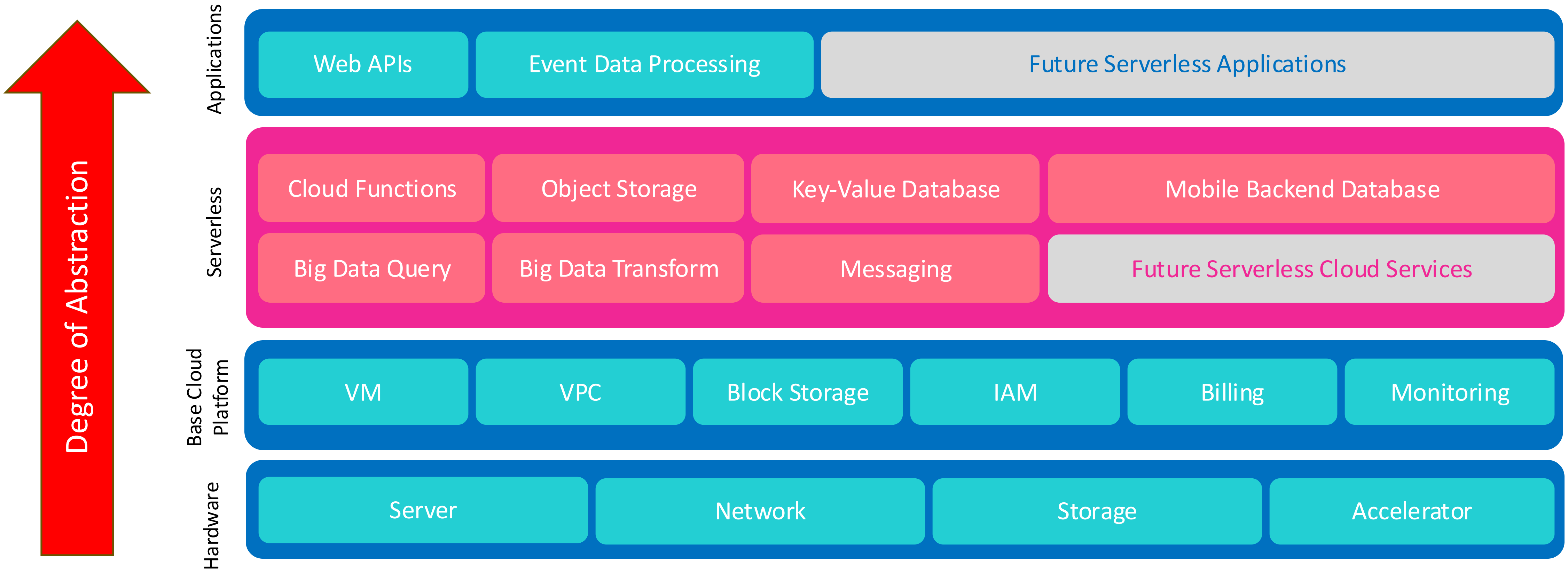
Serverless

- Serverless: Function as a Service + Backend as a Service (FaaS + BaaS)
- Performed by Cloud System
  - Instance selection
  - Scaling
  - Deployment
  - Fault tolerance
  - Monitoring, logging
  - Security patches

- Three fundamental differences between Serverless and conventional Cloud Computing:
  - Decoupling of computation and storage (Scale separately, priced independently)
  - Executing code without managing resource allocation
  - Paying for resources used instead of resources allocated

Reference: *Cloud Programming Simplified: A Berkeley View on Serverless Computing (2019)*

Serverless Cloud



Reference (and figure [edited]): *Cloud Programming Simplified: A Berkeley View on Serverless Computing (2019)*

Serverless: Use Cases

Application	*Description*	*Challenges*	*Workarounds*
Real-Time Video	*On-the-fly video encoding*	***Object store too slow*** *to support fine grained communication;* *Functions too coarse grained for tasks*	*Function-to-function communication to avoid object store; a function executes more than one task*
MapReduce	*Big data processing*	*Shuffle does not scale* ***due to object stores latency*** *and IOPS limits*	*small storage with low-latency, high IOPS to speed-up shuffle*
Linear Algebra	*Large scale linear algebra*	*Need large problem size to overcome* ***storage (S3) latency****, hard to implement efficient broadcast*	*Storage with low-latency high-throughput to handle smaller problem size*
ML Pipelines	*ML Training at scale*	***Lack of fast storage*** *to implement parameter server; hard to implement efficient broadcast, aggregation*	*Storage with low-latency, high IOPS to implement parameter server*
Databases	*OLTP*	*Lack of shared memory,* ***object store has high latency****, lack of support for inbound connectivity*	*Shared file system can work if write needs are low*

Reference: *Cloud Programming Simplified: A Berkeley View on Serverless Computing (2019)*

Serverless: Limitations

- Inadequate storage for fine-grained operations
- Lack of fine-grained coordination
- Poor performance for standard communication patterns
- Obstacles to predictable performance

Reference: *Cloud Programming Simplified: A Berkeley View on Serverless Computing (2019)*

Serverless: Challenges

- Abstraction (resource requirements, data dependencies)
- System (storage, coordination, start time)
- Networking (size, placement, and abstraction to reduce overhead on communication patterns)
- Architecture (heterogeneity)
- Security (randomization, obliviousness, granularity of security context)

Reference: *Cloud Programming Simplified: A Berkeley View on Serverless Computing (2019) and What Serverless Computing Is and Should Become: The Next Phase of Cloud Computing (2021)*

Cloud: Predictions for the Future!

- Simplified Cloud Programming Abstractions
  - General-purpose Serverless Abstractions: Support all use cases
    - Enhance Performance (Reach Serverful Performance)
      Support state management and optimizations, user-suggested or automatically inferred
  - Application-specific Serverless Abstractions

- Enhanced Cloud Functions
  - Programmer hints for optimization
  - Automated optimization

- Serverless Cost Comparable to (and expectedly less than) Serverful
- Machine Learning used by Cloud Providers for Serverless Implementations
- Computing Hardware Heterogeneity for Serverless

Reference: *What Serverless Computing Is and Should Become: The Next Phase of Cloud Computing (2021)*

Acknowledgements

https://canvas.sfu.ca/courses/88212/pages/references