CSP 554
BIG DATA TECHNOLOGIES

Module 1b

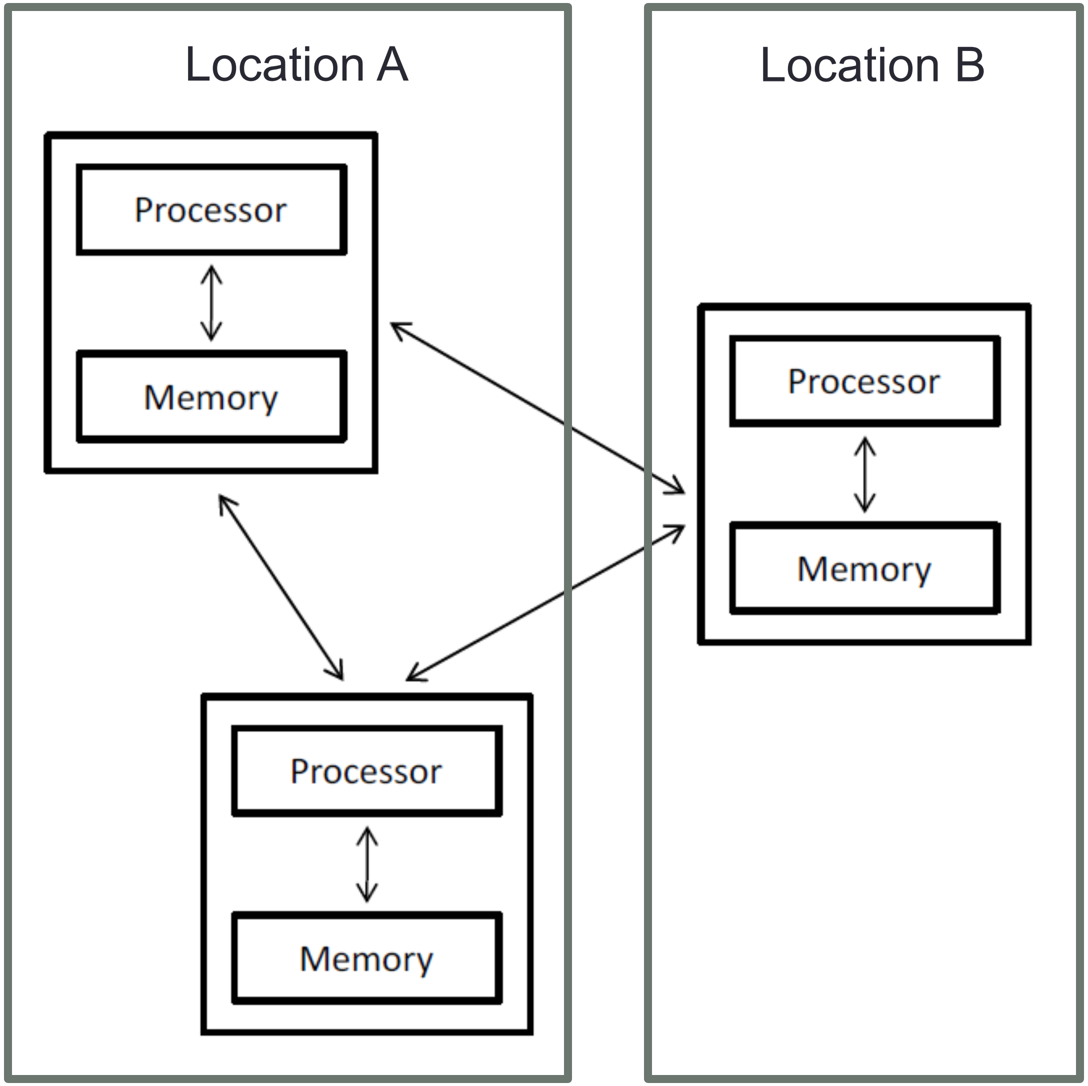
Distributed System Concepts

Topics

- Distributed System Concepts
  - Most big data technology is underpinned by distributed systems
  - We will explore the nature, advantages, disadvantages and also the architecture of these systems
- Failure Modes in Distributed Systems
  - Distributed systems while beneficial in enhancing our ability to handle big data introduce a range of unique failure modes
  - We will review the categories of failures to which distributed systems are subject (against which big data software must compensate)

Distributed System: Definition

- A collection of independent computers (nodes) that appears to its users as a single coherent system
- Computers in distributed systems don't share memory, CPUs or clock and relay information over a communication medium



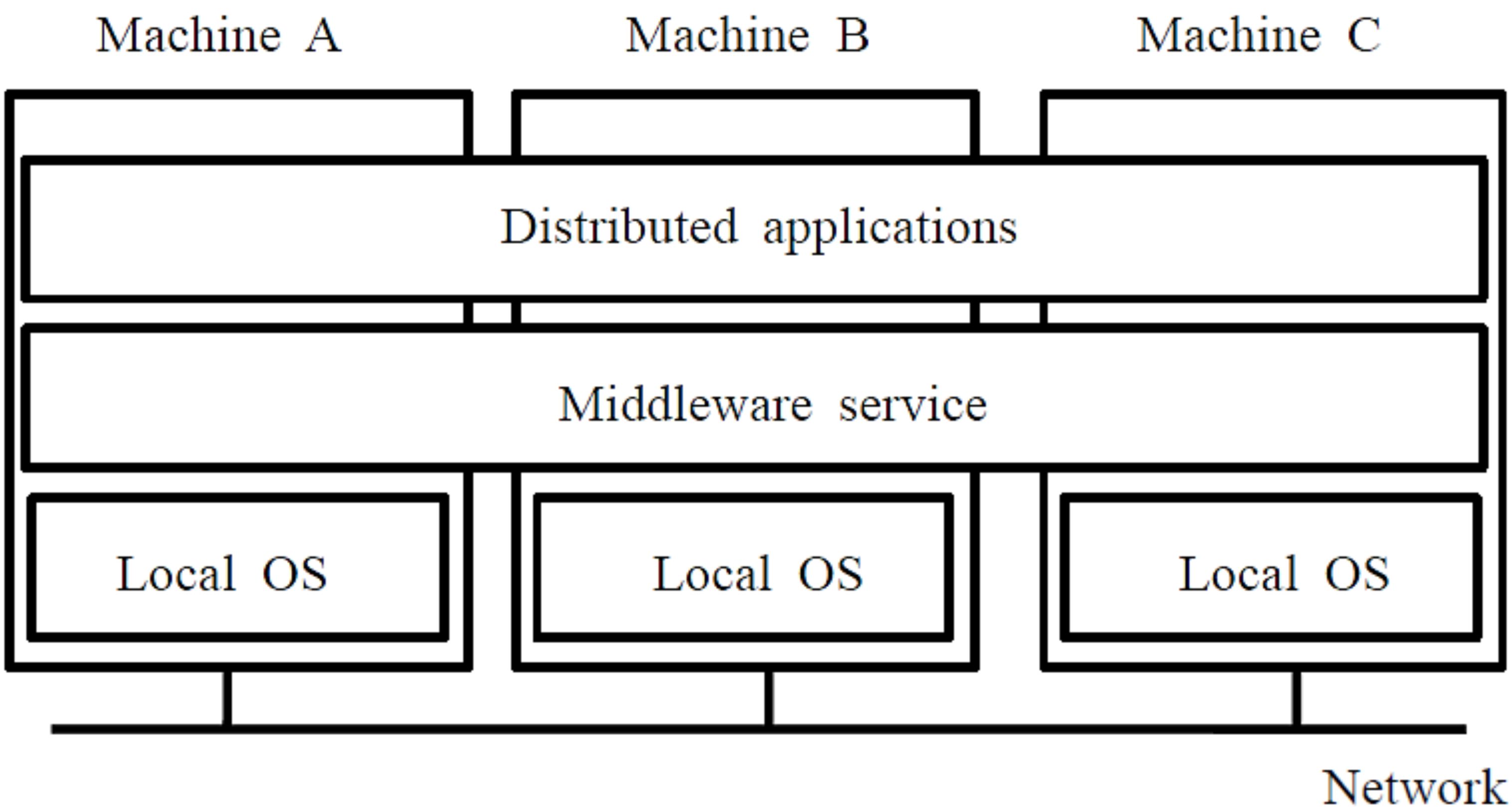
What Does Coherent System Mean?

- *Access transparency*: enables local and remote resources to be accessed using identical operations.
- *Location transparency*: enables resources to be accessed without knowledge of their physical or network location (for example, which building or IP address).
- *Concurrency transparency*: enables several processes to operate concurrently using shared resources without interference between them.
- *Replication transparency*: enables multiple instances of resources to be used to increase reliability and performance without knowledge of the replicas by users or application programmers.

What Does Coherent System Mean?

- *Failure transparency*: enables the concealment of faults, allowing users and application programs to complete their tasks despite the failure of hardware or software components.
- *Mobility transparency*: allows the movement of resources and clients within a system without affecting the operation of users or programs.
- *Performance transparency*: allows the system to be reconfigured to improve performance as loads vary.
- *Scaling transparency*: allows the system and applications to expand in scale without change to the system structure or the application algorithms.

Distributed System Architecture



Independent computers and coherent system => middleware

Distributed System Advantages

- Distributed computing systems offer a better price/performance ratio than centralized systems
- Redundancy increases availability when parts of a system fail
- Applications that can be parallelized also offer benefits in terms of faster performance vis-à-vis centralized solutions
- Distributed systems can be extended through the addition of components, providing better scalability compared to centralized systems

Distributed System Challenges

- Building reliable systems from unreliable components
  - Nodes fail independently
  - A distributed system can “partly fail”
  - Unreliable network communication
- Administration issues
- Placing data and computation for effective resource sharing, hiding latency
  - And finding data/results again once you put it somewhere
- Managing coordination and shared state
  - What should the system components do and when should they do it?
  - Once they’ve all done it, can they all agree on what they did and when?

Distributed System Challenges

- Troubleshooting and diagnosing problems in a distributed system can also become more difficult
  - The analysis may require connecting to remote nodes or inspecting communication between nodes.
- Many types of computation are not well suited for distributed environments
  - Typically owing to the amount of network communication or synchronization that would be required between nodes
  - If bandwidth, latency, or communication requirements are too significant, then the benefits of distributed computing may be negated
    - Performance may be worse than a non-distributed environment

Why Distributed Systems

Availability	serve *every* request
Fault Tolerance	resilient to failures
Throughput	parallel computation
Architecture	decoupled, focused services
Economics	scale-out becoming manageable/ cost-effective

Reliable Distributed Systems

Fault-tolerant	nodes can fail
Available	serve *all* the requests, *all* the time
Scalable	behave correctly with changing topologies
Consistent	state is coordinated across nodes
Secure	access is authenticated
Performant	it's *fast!*

Scalability

- Scalability is sometimes defined as "the ease with which a system or component can be modified to fit the problem area."
- A scalable system has three simple characteristics:
  - The system can accommodate increased usage,
  - The system can accommodate an increased data set,
  - The system is maintainable and works with reasonable performance

Scalability Types

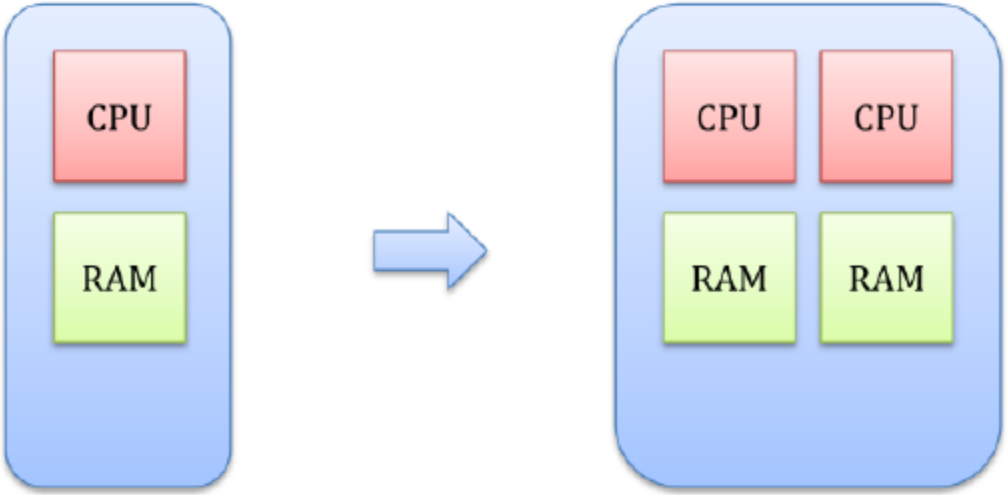
- Vertical scaling, where you scale by adding more resources, like CPU, memory, or storage to a single system.
- Horizontal scaling, where you scale by adding more machines.

One Way to Do Heavy Lifting

Scaling Up: if the train car is too heavy find a bigger crane

Scaling Up (Vertical Scaling)

- Refers to resource maximization of a single unit to expand its ability to handle increasing load.
- In hardware terms, this includes adding processing power and memory to the physical machine running the server.
- In software terms, scaling up may include optimizing algorithms and application code.

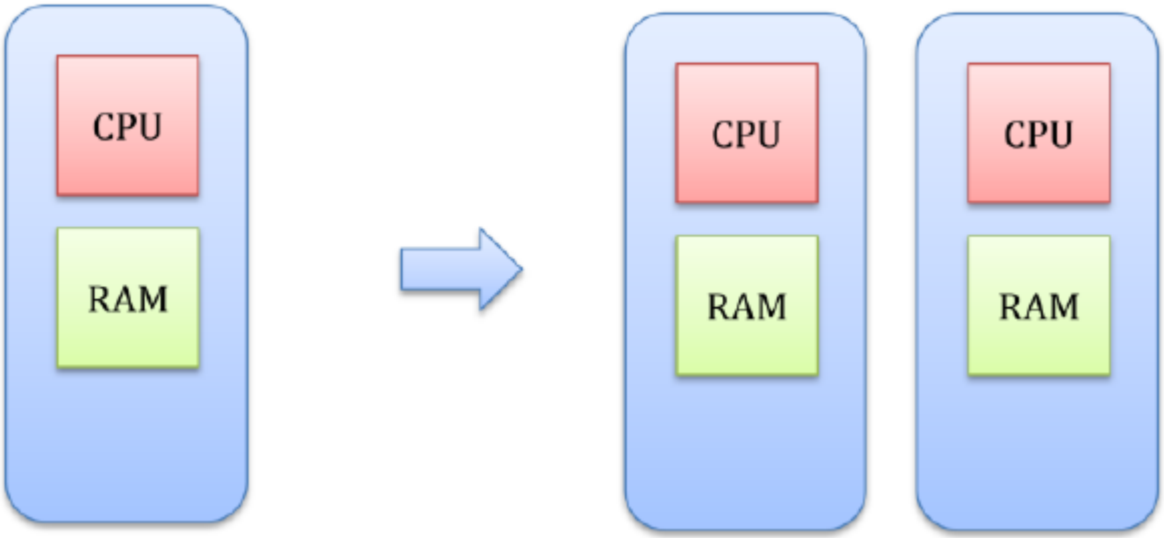


Another Way to Do Heavy Lifting

Scaling Out: if the train car is too heavy find lots more people

Scaling Out (Horizontal Scaling)

- Refers to resource increment by the addition of units to the system.
- This means adding more units of smaller capacity instead of adding a single unit of larger capacity.
- The requests for resources are then spread across multiple units thus reducing the excess load on a single machine.



Horizontal Scalability Advantages

- Real driving forces behind the trend toward scale-out are semiconductor physics and economics
  - Over a quarter of a century ago, Herb Grosch stated :The computing power of a CPU is proportional to the square of its price.
  - By paying twice as much, you could get four times the performance.
  - This observation fit the mainframe technology of its time quite well, and led most organizations to buy the largest single machine they could afford.
- With microprocessor technology, Grosch's law no longer holds.
  - For a few hundred dollars you can get a CPU chip that can execute more instructions per second than one of the largest 1980s mainframes.
  - But if you are willing to pay twice as much, you get the same CPU, running at a somewhat higher clock speed (physics limitations)
  - So the most cost-effective solution is to harness a large number of (relatively) cheap CPUs together in a system.

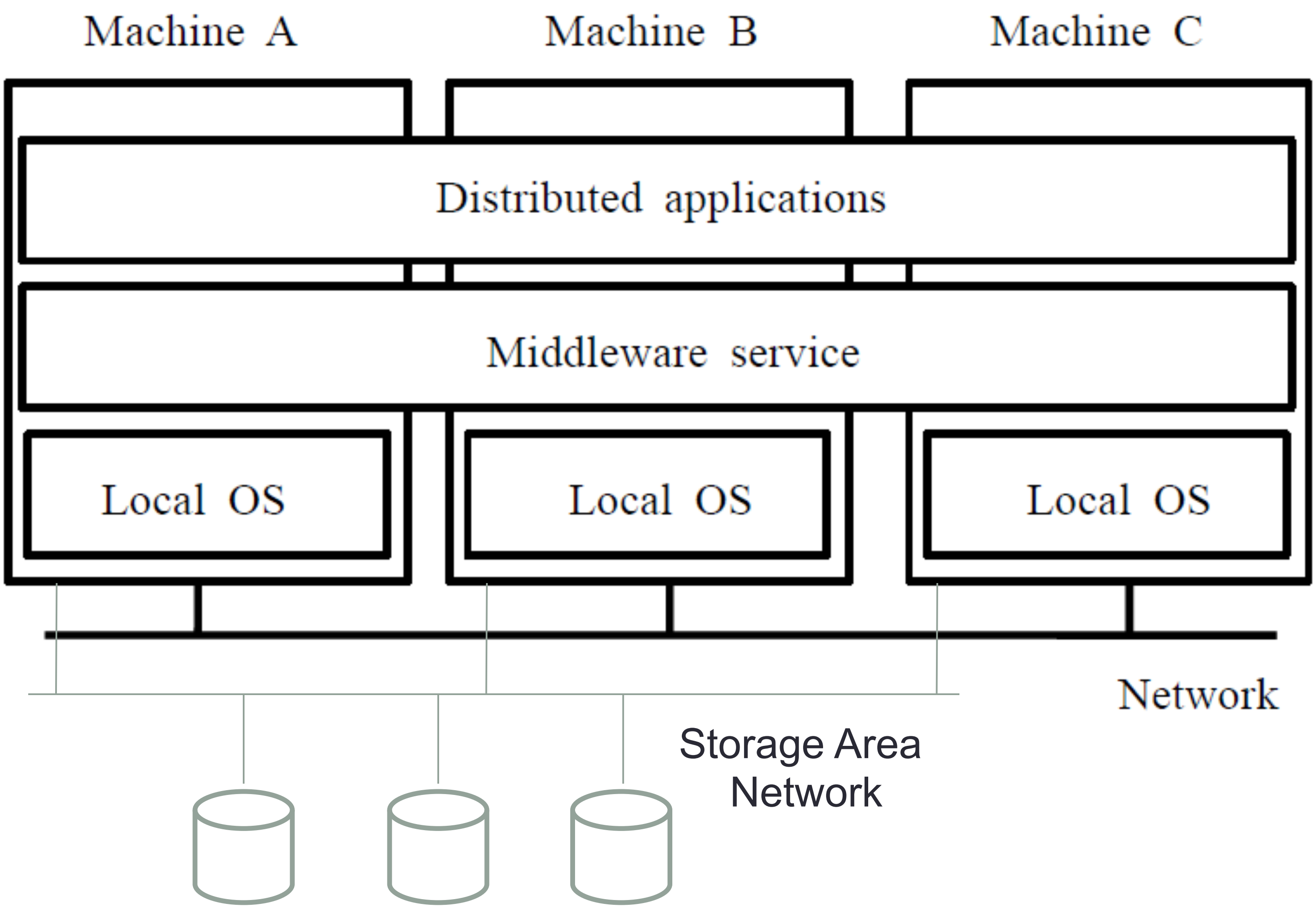
Horizontal Scalability Challenges

- Parallelize the workload across the various machines
- Consistent, shared view of the entire data set
- Handle distributed concurrency and consistency
- Minimize the amount of coordination overhead

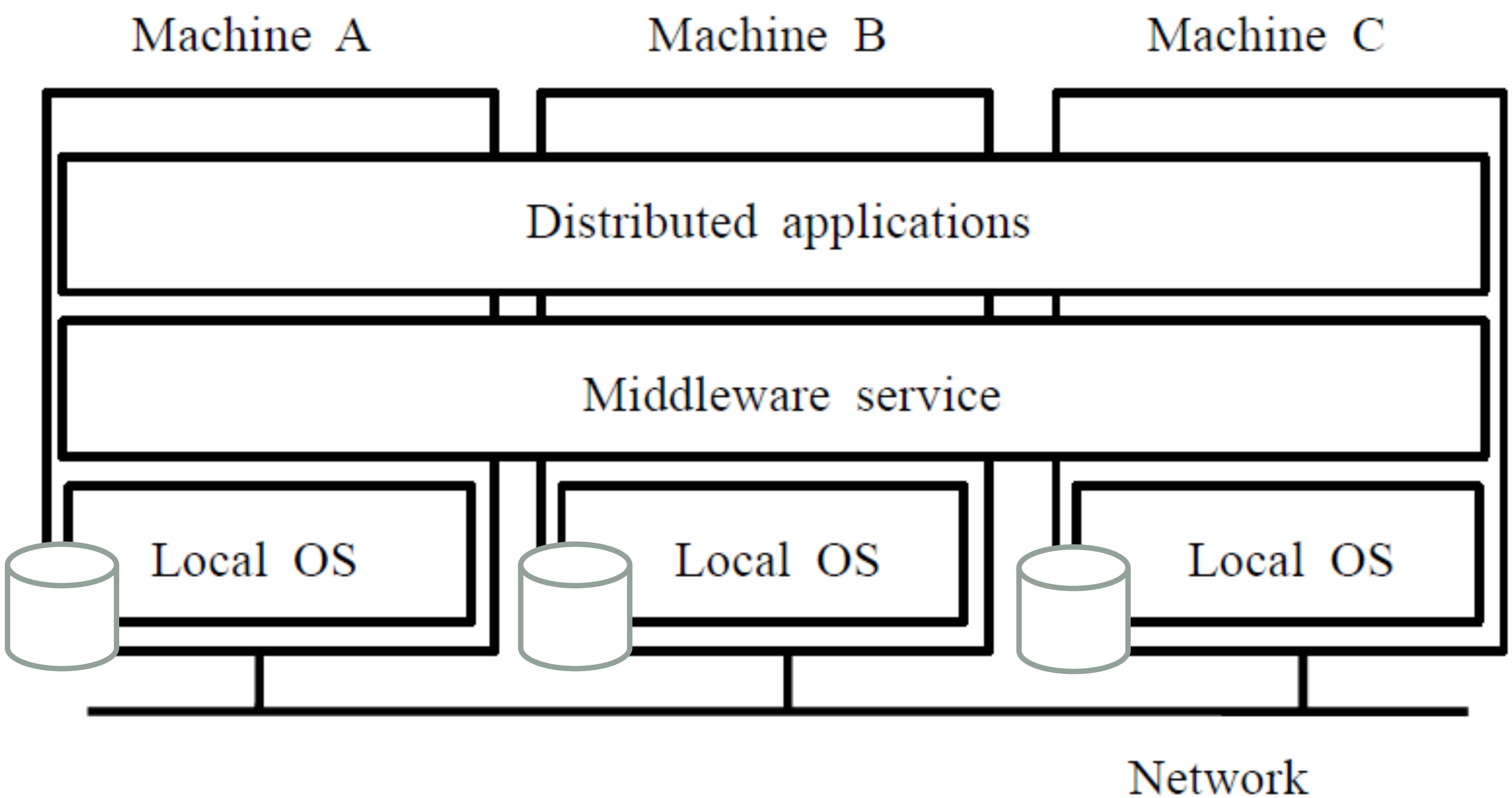
Scale-Out Today

- A well architected collection of microprocessors yields an absolute performance that no mainframe can achieve at any price
  - With current technology it is possible to build a system from 10,000 modern CPU chips, each of which runs at 50 MIPS, for a total performance of 500,000 MIPS.
  - No existing machine even comes close to this, and both theoretical and engineering considerations make it unlikely that any machine ever will

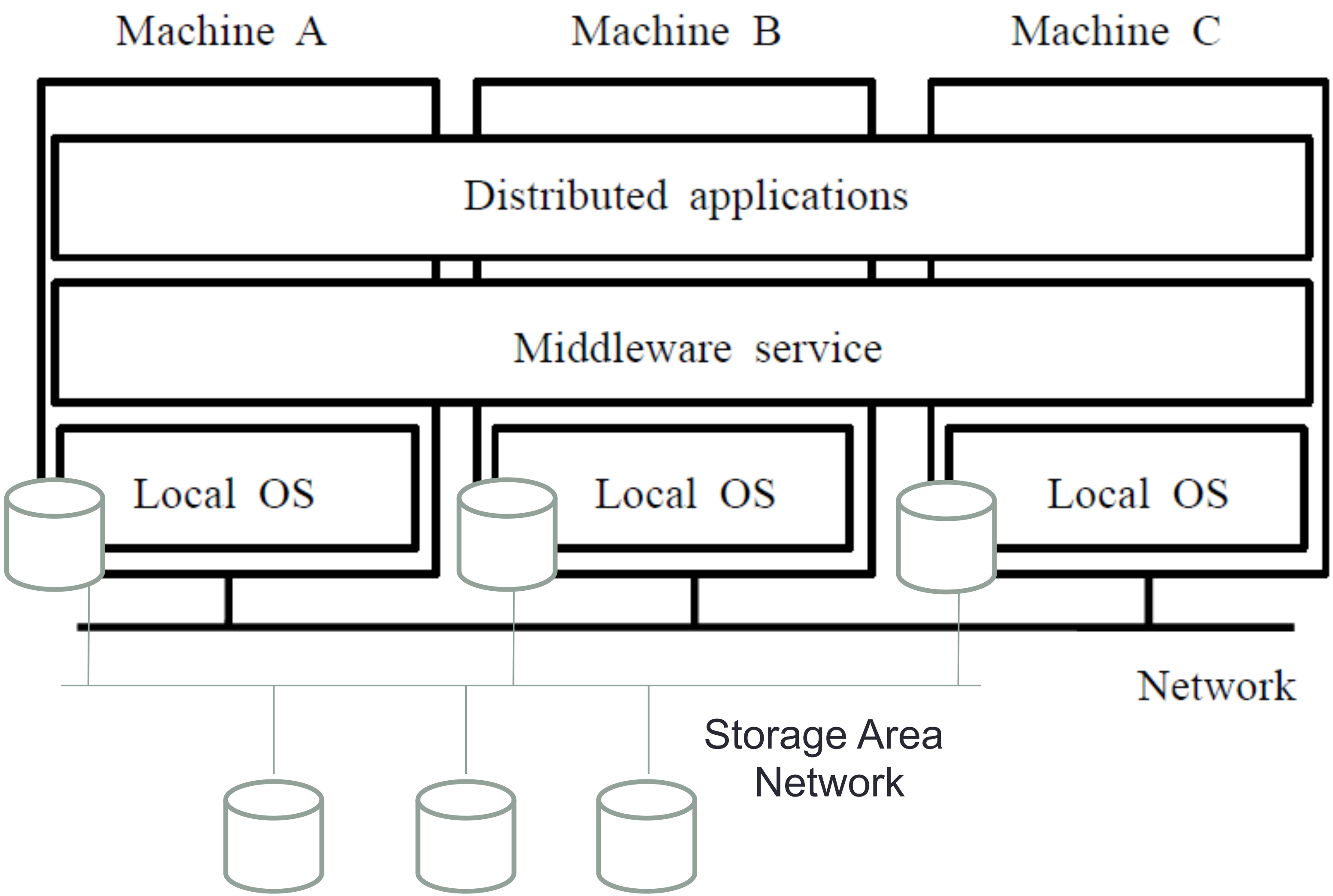
Shared Everything Storage



Shared Nothing Storage



Mixed Mode Storage

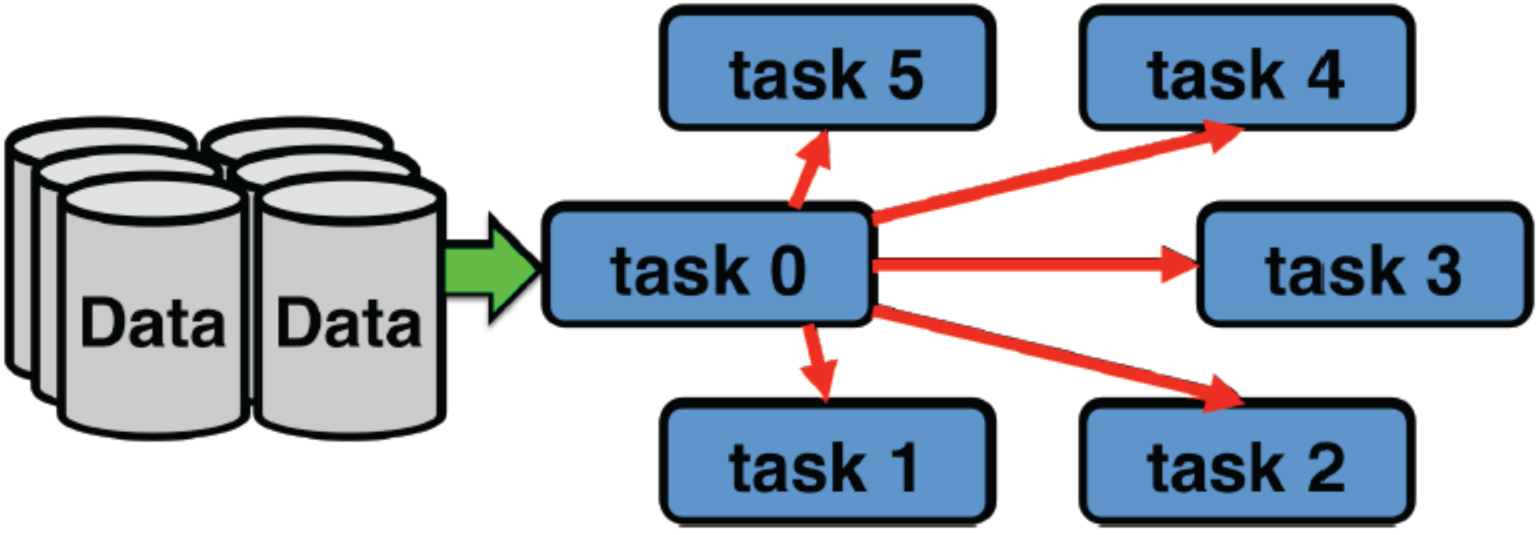


Cluster: Definition

- Cluster is a special case of distributed system
- Defining feature of a cluster is tight coupling between computers in the system
- Typically located next to each other and connected via high-speed network
- The fastest supercomputers in the world are clusters
- Composed of many 1000's of microprocessors or GPUs located in a single data center
- The primary approach to handling big data

IBM Watson™

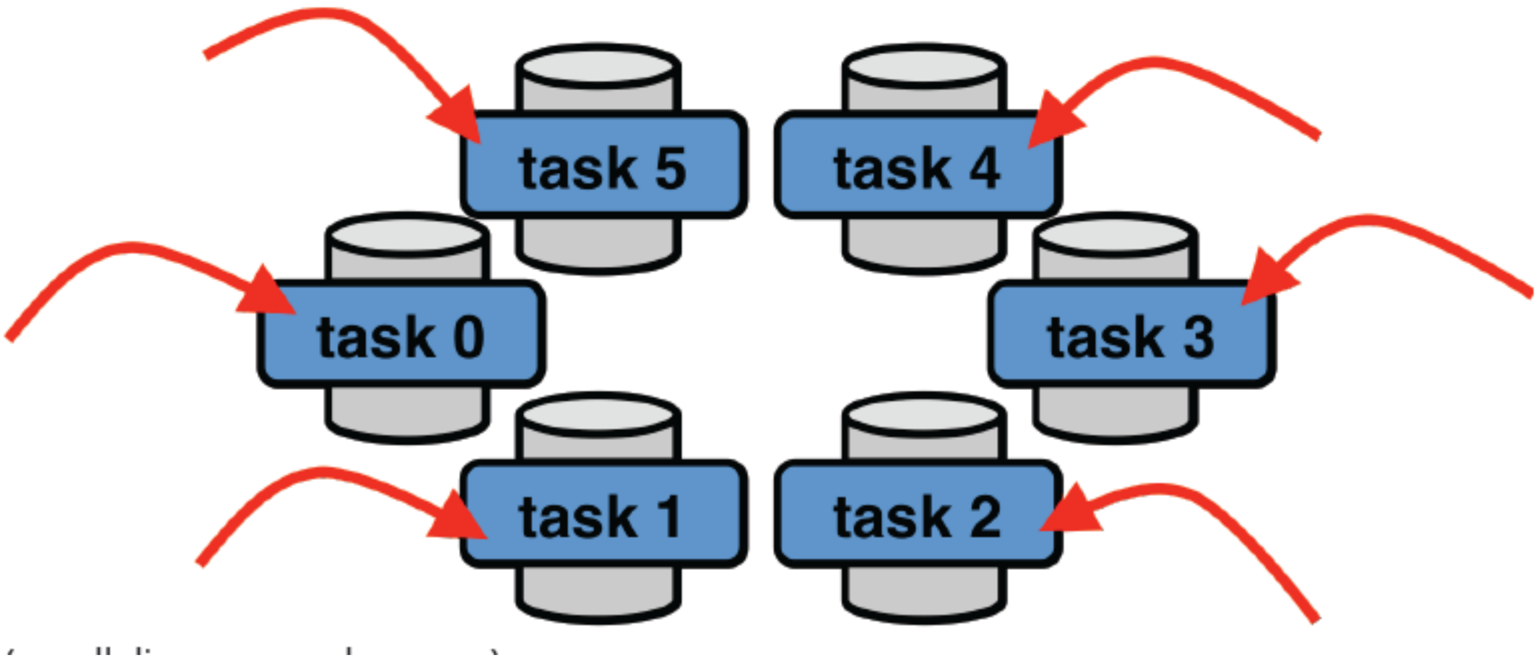
Traditional Distributed Computing Pattern



CPU bound, not I/O or data bound

Strategy: Move data to processing

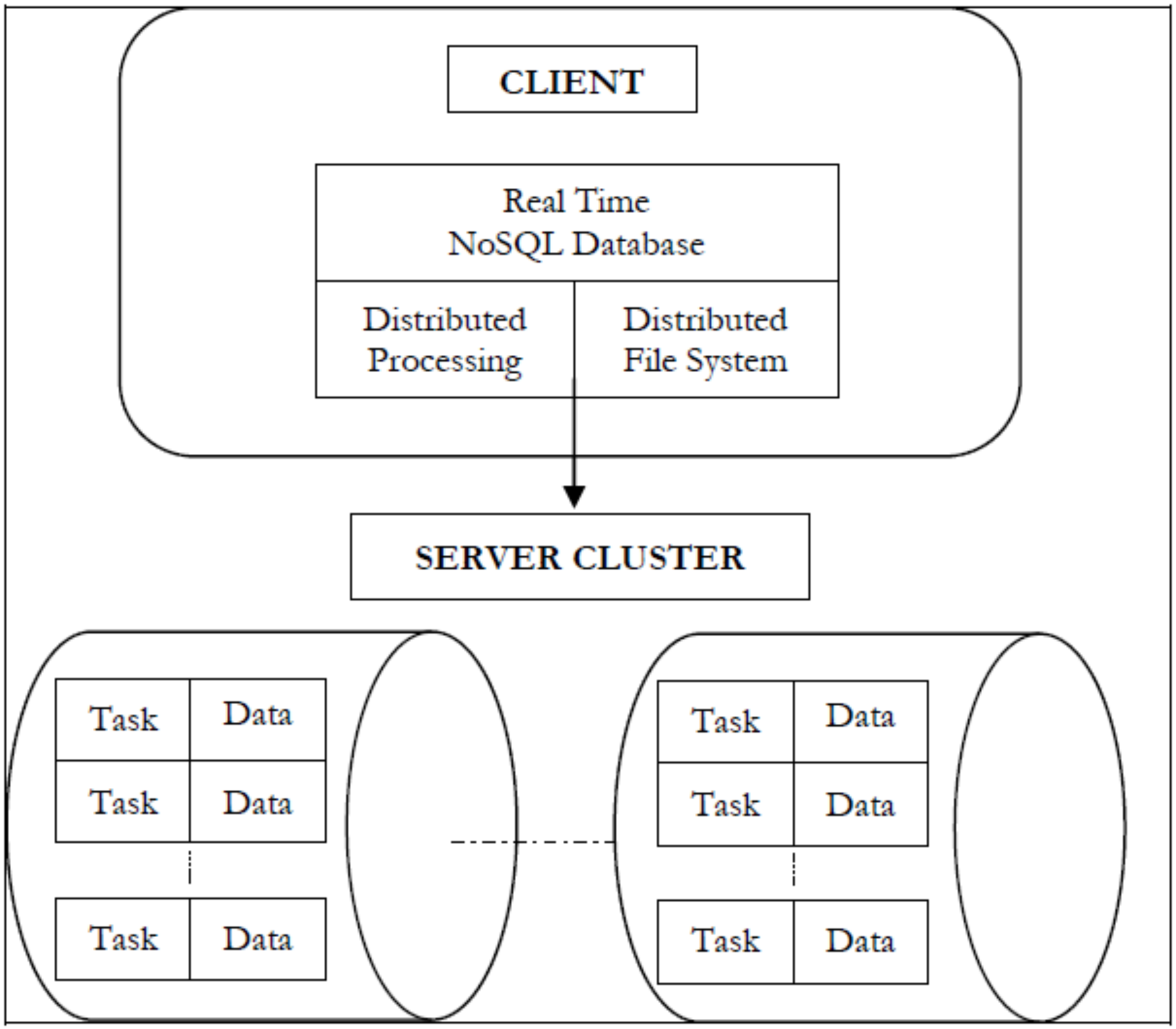
Big Data Distributed Computing Pattern



I/O or data bound, not CPU bound

Strategy: Move processing to data

Big Data Clusters



Distributed System: Definition

A distributed system is a system in which I can’t get my work done because a computer that I’ve never heard of has failed.

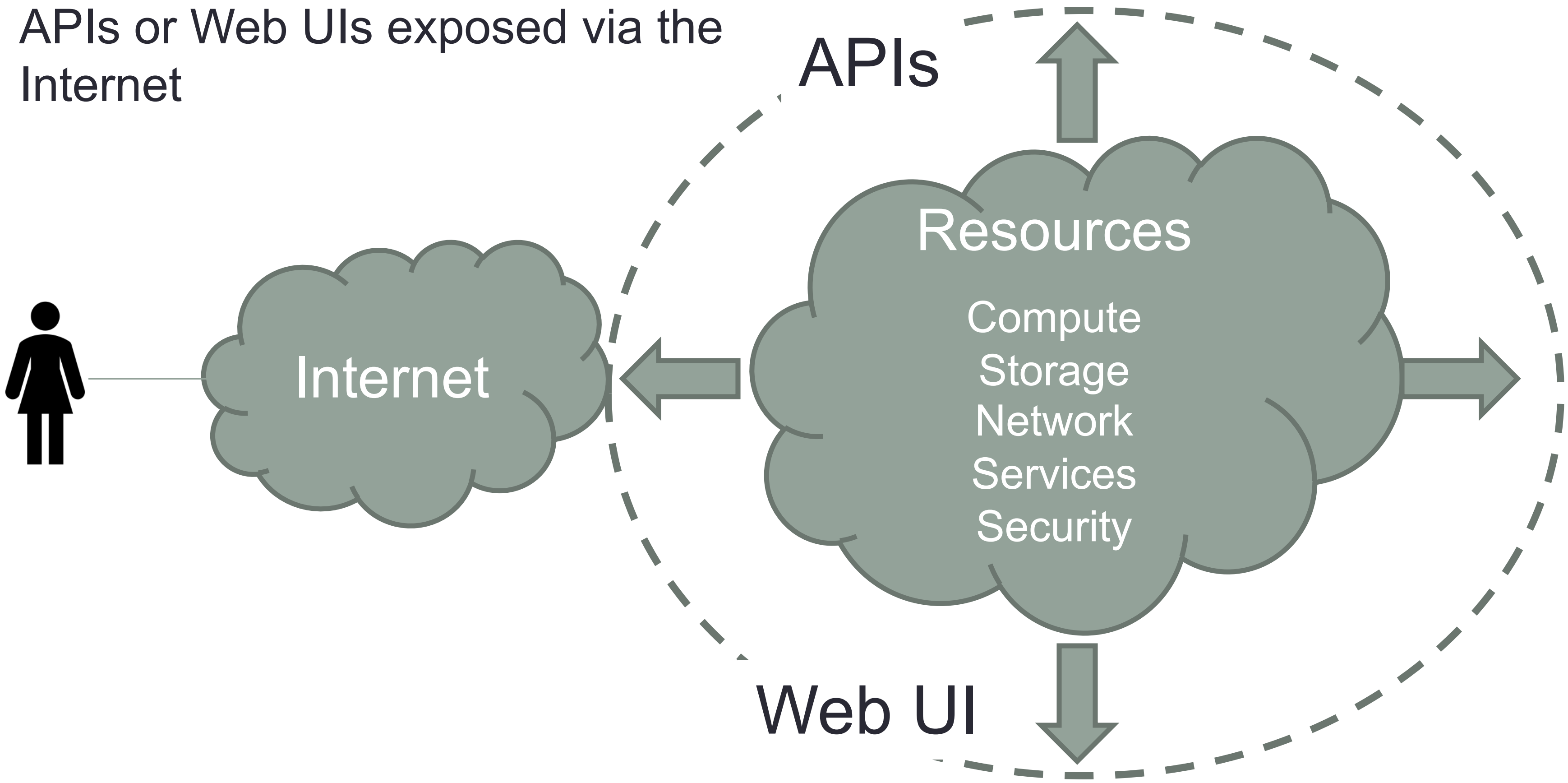
-- Butler Lampson

Cloud Computing

- Distributed systems as a utility service
- Enables convenient, on-demand Internet access to a shared pool of configurable computing resources
  - (e.g., networks, servers, storage, applications, and services)
- Resources can be rapidly provisioned and released with minimal management effort or service provider interaction

Cloud Computing

Resources are accessible through APIs or Web UIs exposed via the Internet



Cloud Computing: Essential Characteristics

Speed and Agility

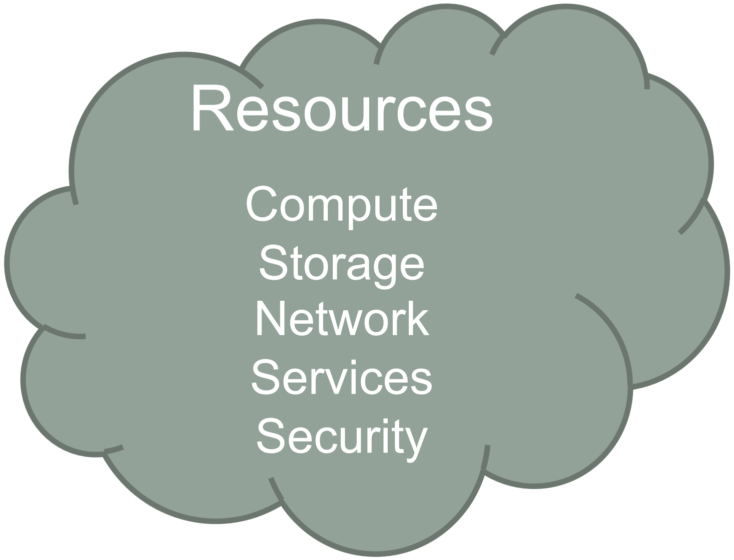
Elastic Capacity

Multiple Tenancy

Pay-as-you-go

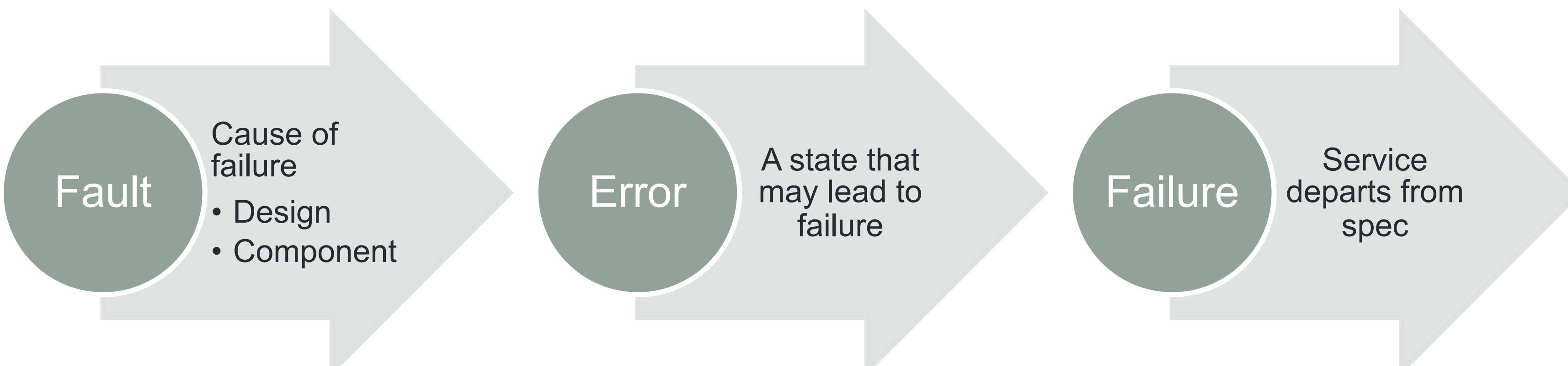
Innovative Services

Big Data Support



Definitions

- A system failure is a deviation from the service that the system is supposed to provide (visible to users)
- An error is an incorrect value of the system state which can provoke a failure
- Finally a fault is a defect in the design or in the operation of the system that may cause an error.
  - Faults can be transient (one time), intermittent (occasional) or persistent (remain until repaired)



Reliability Concerns

- Availability
  - Can I use it now?
- Reliability
  - Will it be up as long as I need it?
- Safety
  - If it fails, what are the consequences?
- Maintainability
  - How easy is it to fix if it breaks?

Other Reliability Notes

- A fault need not result in an error, nor an error in a failure.
- A *non-faulty* component will produce an output that is in accordance with this specification.
- The response from a faulty component need not be as specified, i.e., it can be anything.
- The response from a given component for a given input will be considered to be correct
  - If the output value is correct
  - The output is produced on time, i.e., produced within a specified time limit.

An alpha particle corrupting a memory location is a fault. If that memory location contains data, that corrupted data is an error. If a program crashes because of using that data, it is a failure.

Possible Failures in Distributed Systems

- Data corruption
- Hanging processes
- Misleading return values
- Misbehaving computers
- Network outages
- Hardware failures
- Insufficient resources

Failure Modes

Type of Failure	Description
Crash failure	A server halts, but is working until it halts
Omission failure Receive omission……….. Send omission…………....	A server fails to respond to incoming requests A server fails to receive incoming messages A server fails to send outgoing messages
Timing failure	A server’s response lies outside the specified time interval
Response failure Value failure……………... State transition failure…..	The servers response is incorrect The value of the response is wrong The server deviates from correct flow of control
Arbitrary failure	A server may produce arbitrary responses at arbitrary times

Omission Failure

- A component that does not respond to an input from another component, and thereby fails by not producing the expected output is exhibiting an *omission fault* and the corresponding failure an *omission failure*.

A communication link which occasionally loses messages is an example of a component suffering from an omission fault.

Value Failure

- A fault that causes a component to respond within the correct time interval but with an incorrect value is termed a value fault (with the corresponding failure called a *value failure*).

A communication link which delivers corrupted messages on time suffers from a value fault

A memory, disk or software error could also lead to a value failure

Timing Failure

- A timing fault causes the component to respond with the correct value but outside the specified interval (either too soon, or too late). The corresponding failure is a timing failure.

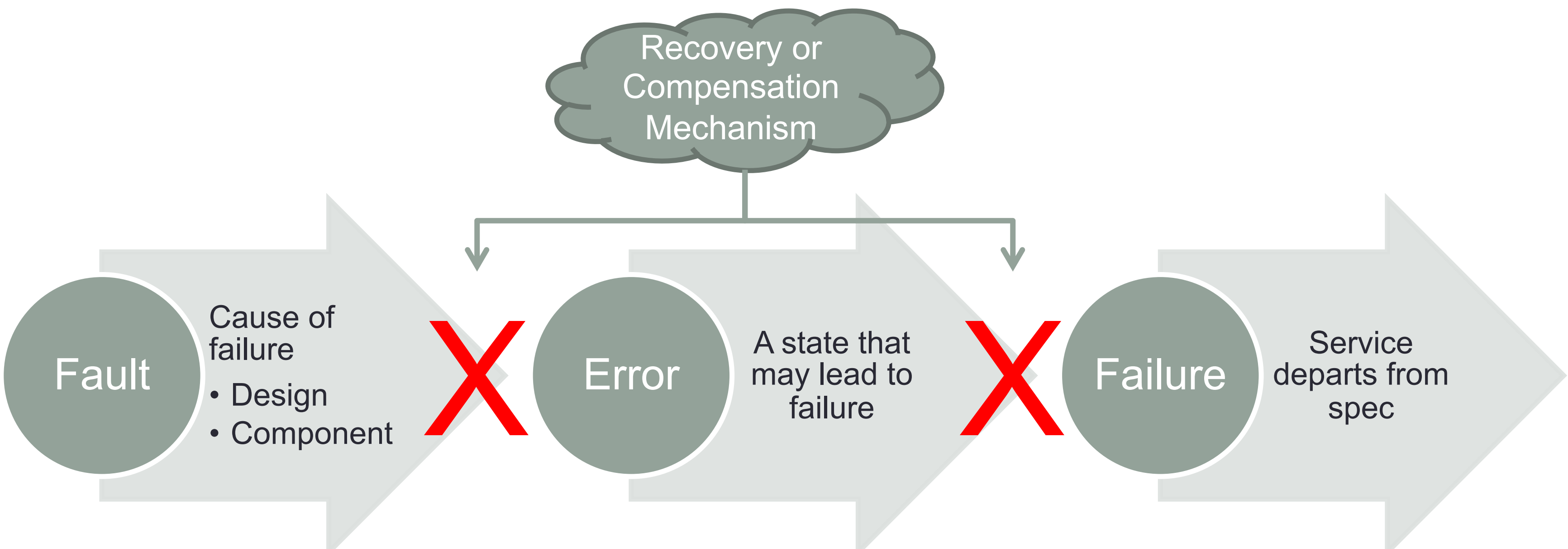
An overloaded processor which produces correct values but with an excessive delay suffers from a timing failure. Timing failures can only occur in systems which impose timing constraints on computations.

Arbitrary Failure

- The previous failure classes have specified how a component can be considered to fail in either the value or time domain.
- It is possible for a component to fail in both the domains in a manner which is not covered by one of the previous classes.
- A failed component which produces such an output will be said to be exhibiting an *arbitrary failure* (AKA, *Byzantine failure*).
- Sometimes called malicious failure because it includes
  - Two-faced behavior; the affected system can send a message “fact X is true” to one user and “fact X is false” to another user
  - Spoofing behavior; Forging of messages of other systems

Fault Tolerance

- The property that enables a system to continue operating properly (from the perspective of the user) in the event of one or more faults or errors
- Big data technology must address fault tolerance in one way or another (through software and/or hardware)



Distributed Systems Summary

- Users of big data have to accept that data needs to be distributed
  - Storing and processing datasets across a cluster of machines is unavoidable
- Once a cluster become the storage and analysis foundation then software has to account for failure
  - Because failure is inevitable when you're talking about running hundreds or thousands of machines in a cluster.
- Finally, analysis logic needs to go to the data and process it on the distributed machines
  - Rather than moving the vast quantities of data across the network.