SOFTWARE ENGINEERING (Week-6)

USAMA MUSHARAF

LECTURER (Department of Computer Science)

FAST-NUCES PESHAWAR

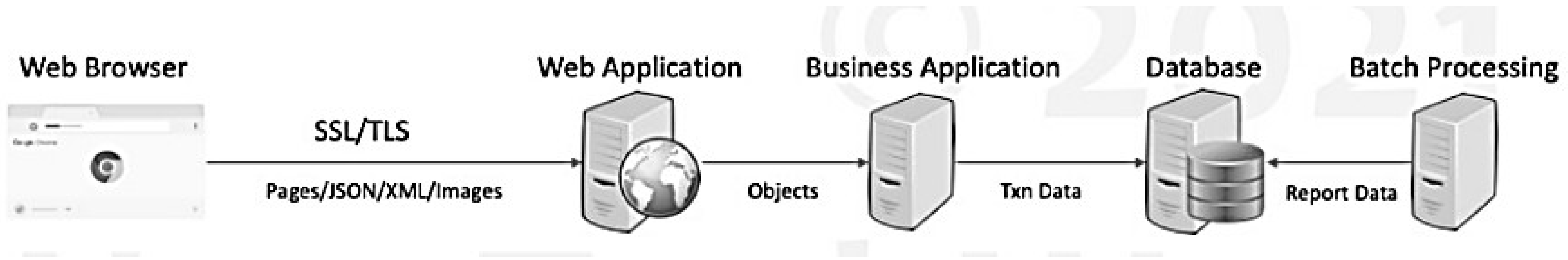
CONTENTS OF WEEK # 6

Software Quality Attributes

- Performance
- Scalability
- Reliability

Application Architecture to System Architecture

Sample System

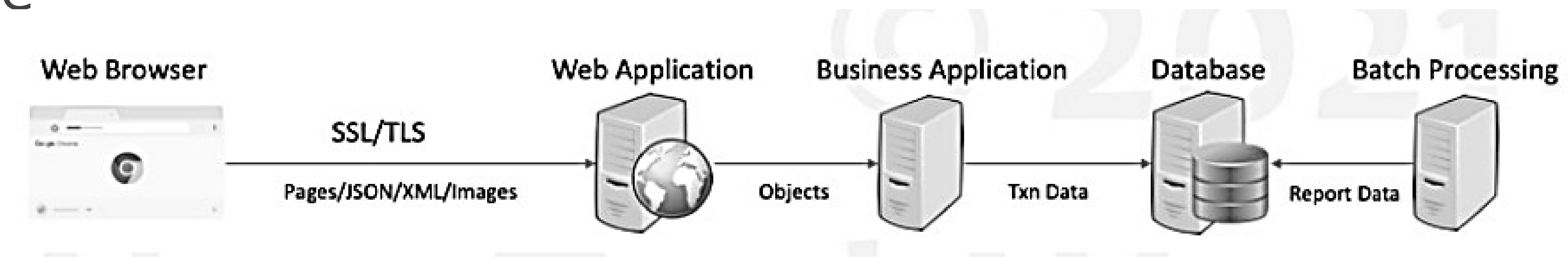


Performance

Performance

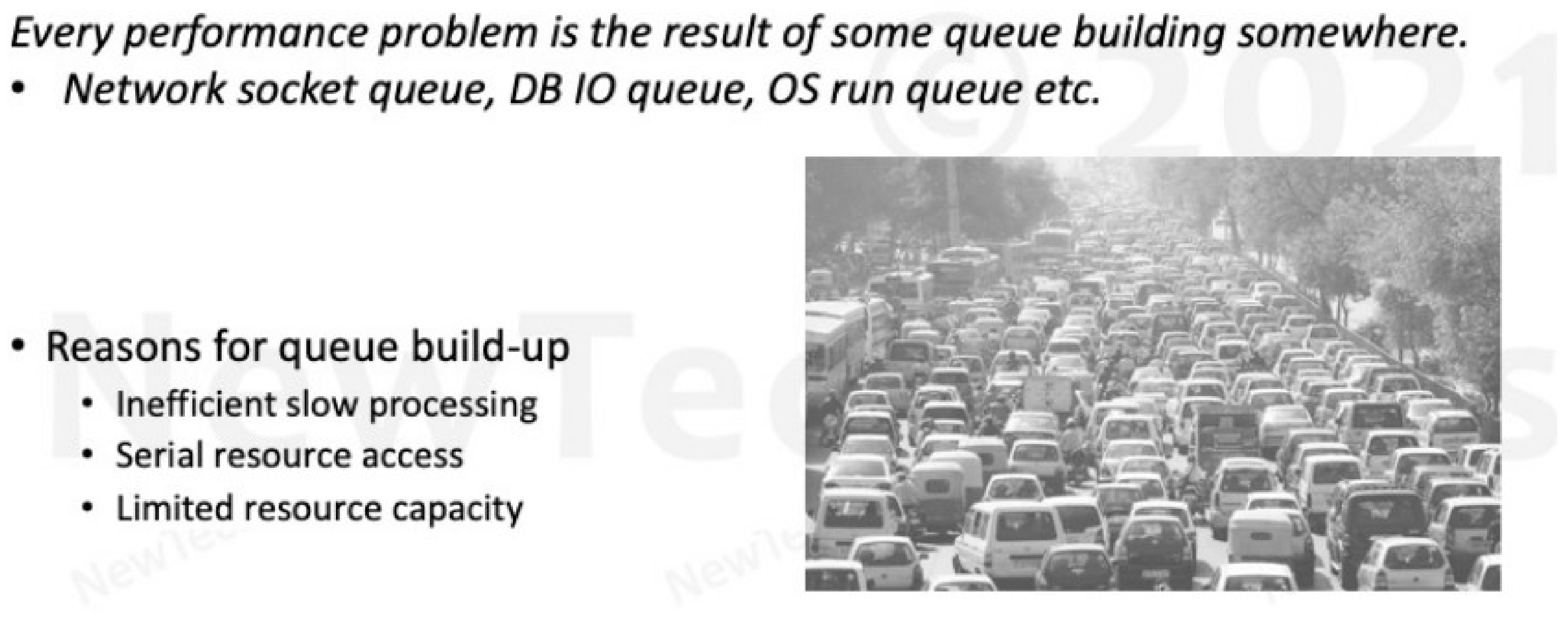
Measure of how fast or responsive a system is under

- A given workload
  - Backend data
  - Request Volume
- A given hardware
  - Kind
  - capacity



Performance Problems

How to spot a performance problem? How does it look like?



Every performance problem is the result of some queue building somewhere.

- *Network socket queue, DB IO queue, OS run queue etc.*

- Reasons for queue build-up
  - Inefficient slow processing
  - Serial resource access
  - Limited resource capacity

Performance Principles

Efficiency

- Efficient Resource Utilization
  - IO- Network, Memory, Disk
  - CPU

Efficient Logic

- Algorithms
- DB Queries

Efficient Data Storage

- Data Structures
- DB Schema

Caching

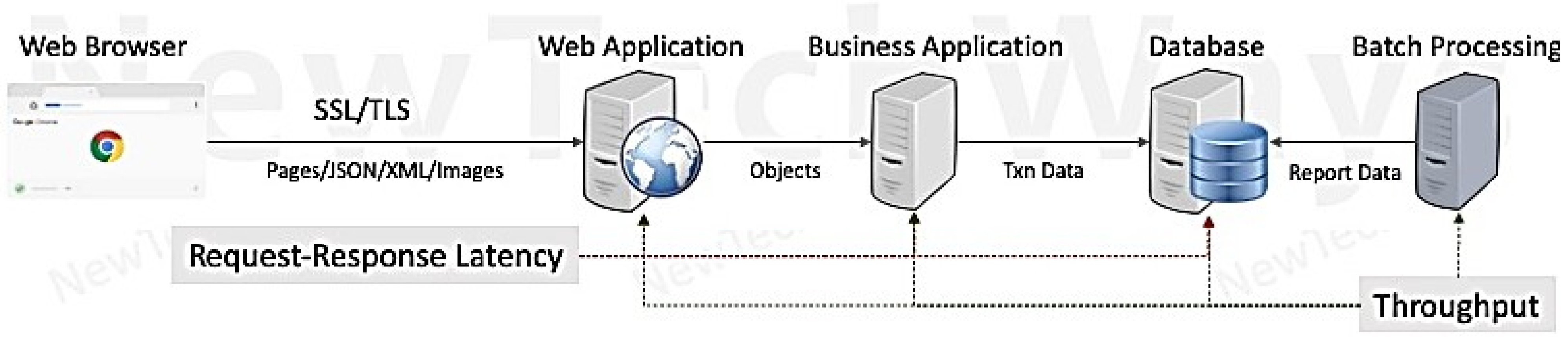
Concurrency

- **Hardware**
- **Software**
  - Queuing
  - Coherence
- **Capacity**

Performance Objectives

- Minimize Request-Response Latency
- Latency is Measured in Time Units
- Depends on
  - Wait/Idle Time
  - Processing Time

- Maximize Throughput
- Throughput is Measured as Rate of Request processing
- Depends on
  - Latency
  - Capacity



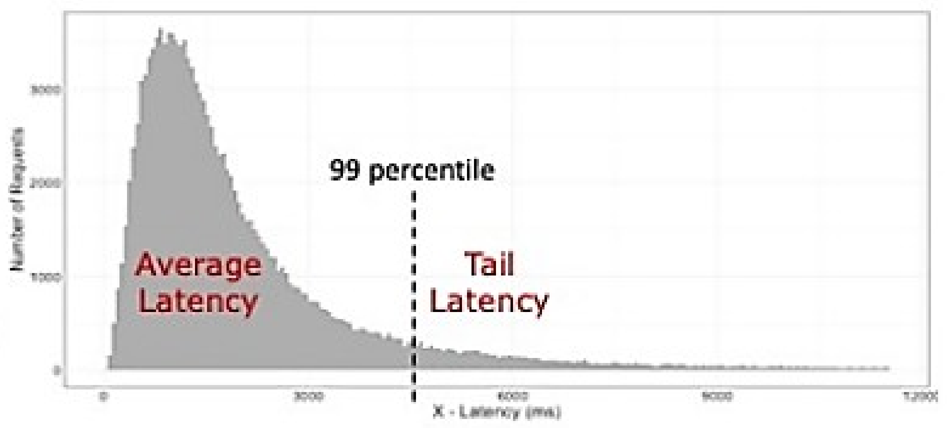
Performance Measurement Metrics

Latency
- Affects– User Experience
- Desired– As low as possible

Throughput
- Affects– Number of users that can be supported
- Desired– Greater than the request rate

Errors
- Affects– Functional Correctness
- Desired– None

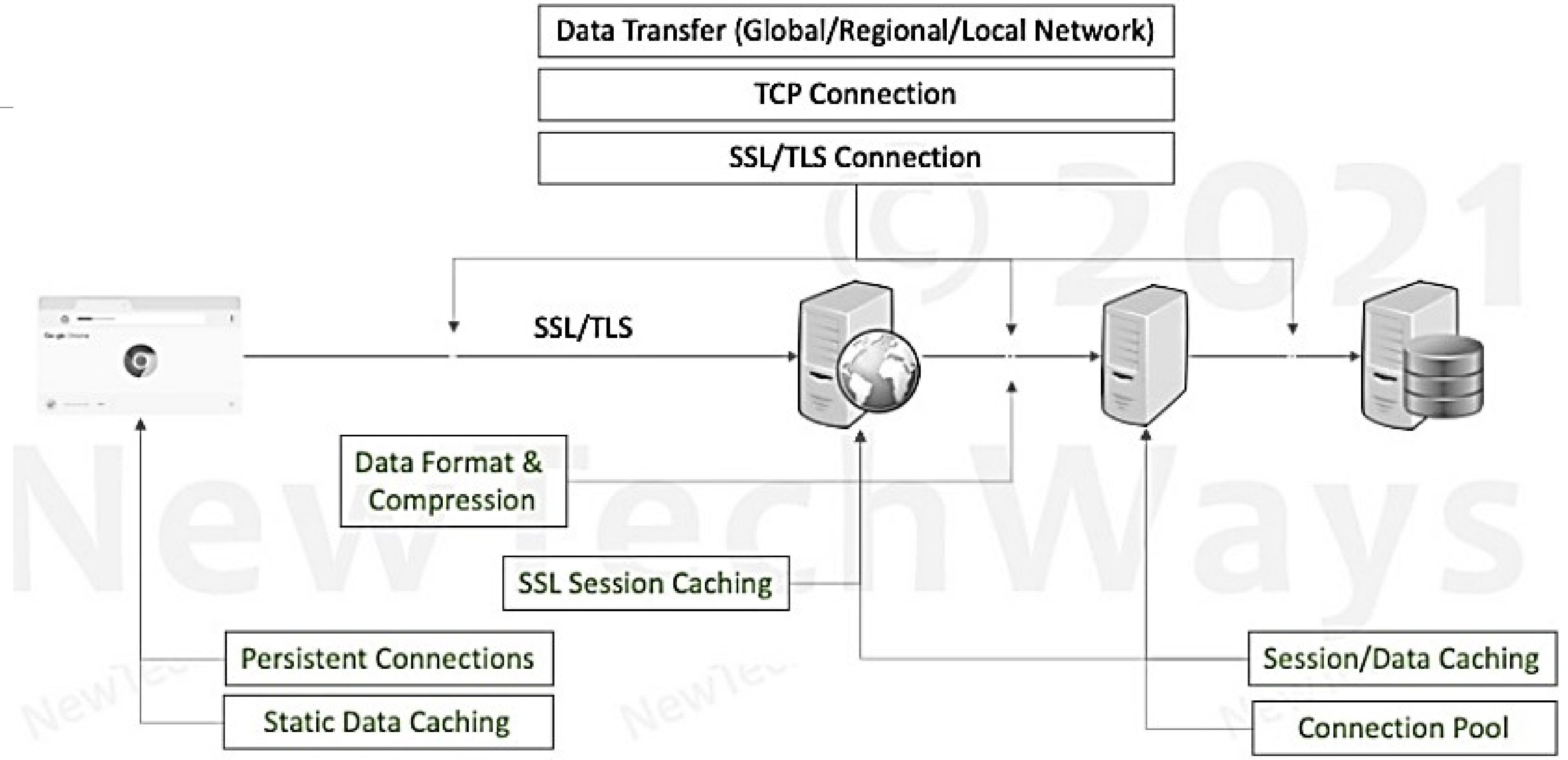
Resource Saturation
- Affects– Hardware Capacity Required
- Desired– Efficient utilization of all system resources.



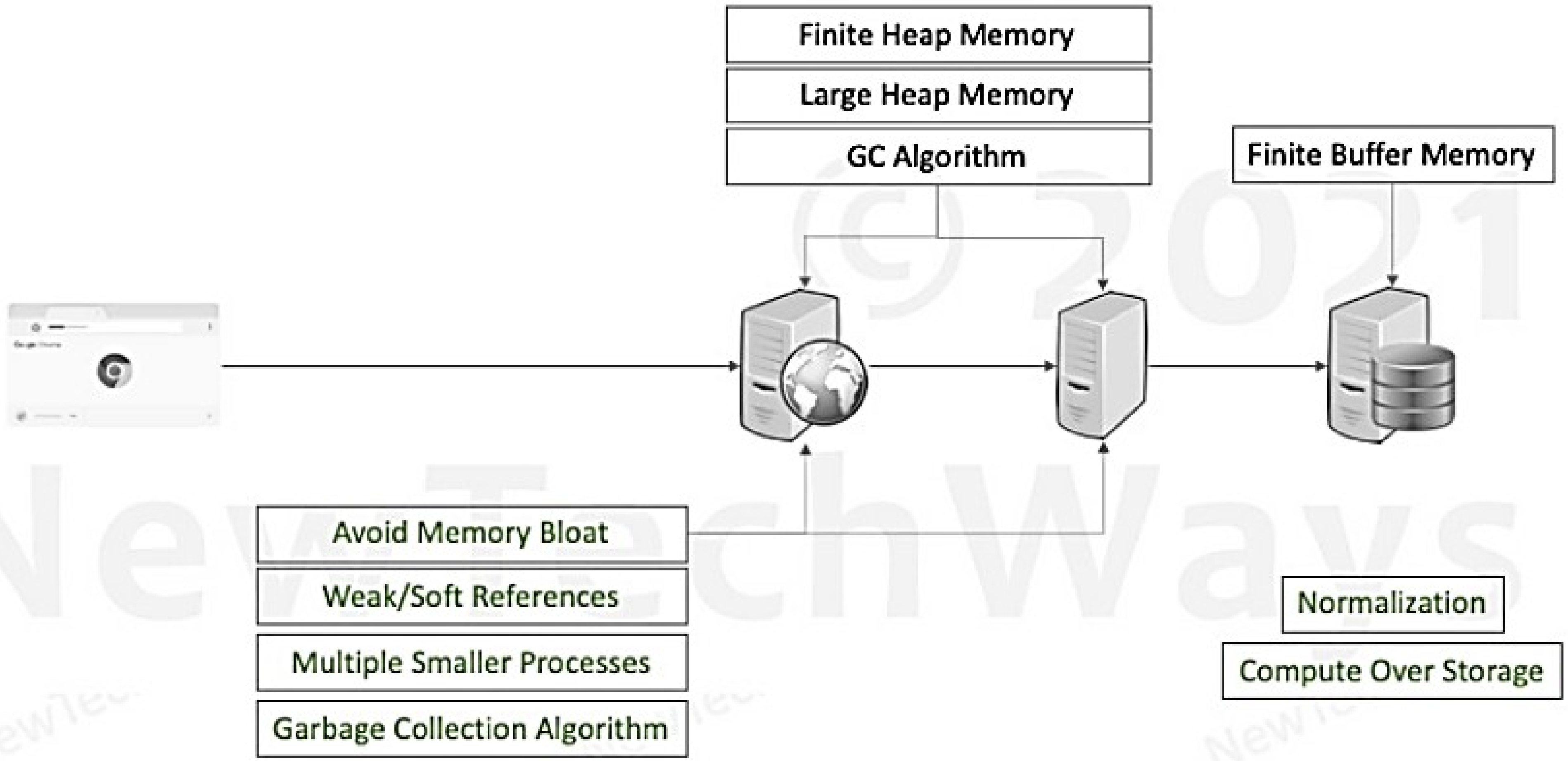
- Tail latency is an indication of queuing of requests
  - Gets worse with higher workloads
- Average latency hides the effects of tail latency
  - Also measure 99 (or 99.9) percentile latency

Serial Request Latency

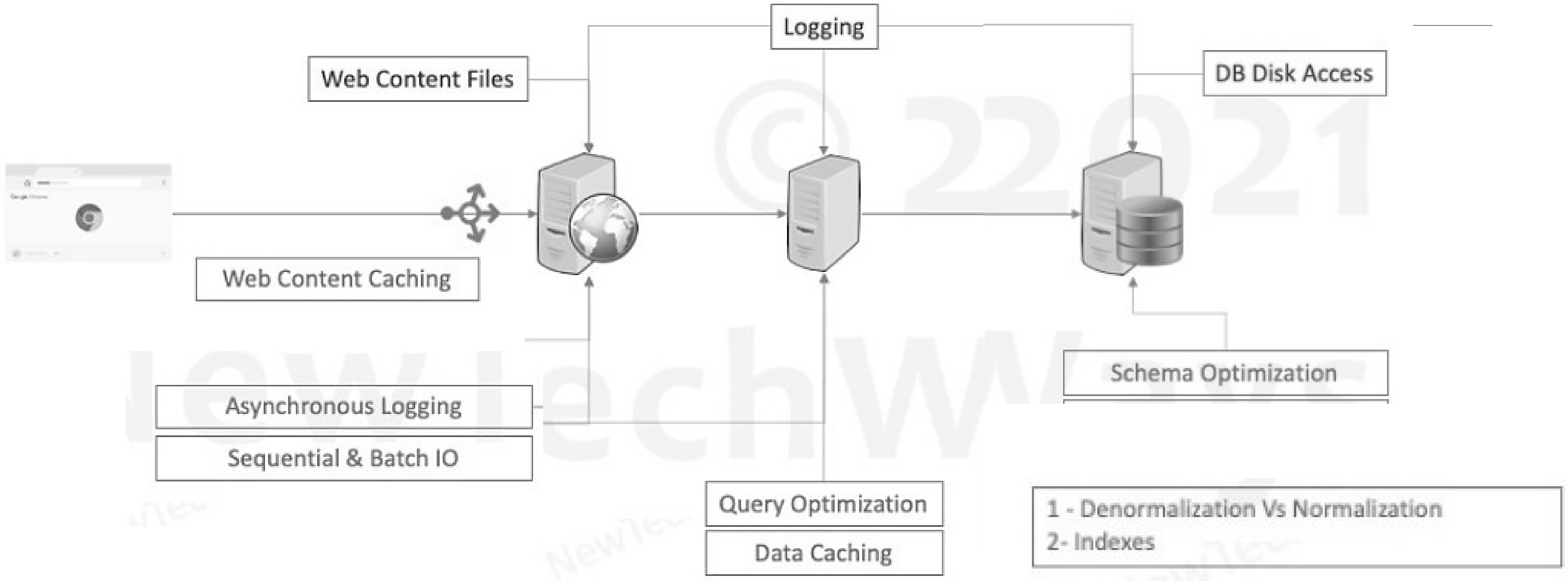
Network Latency – Approaches



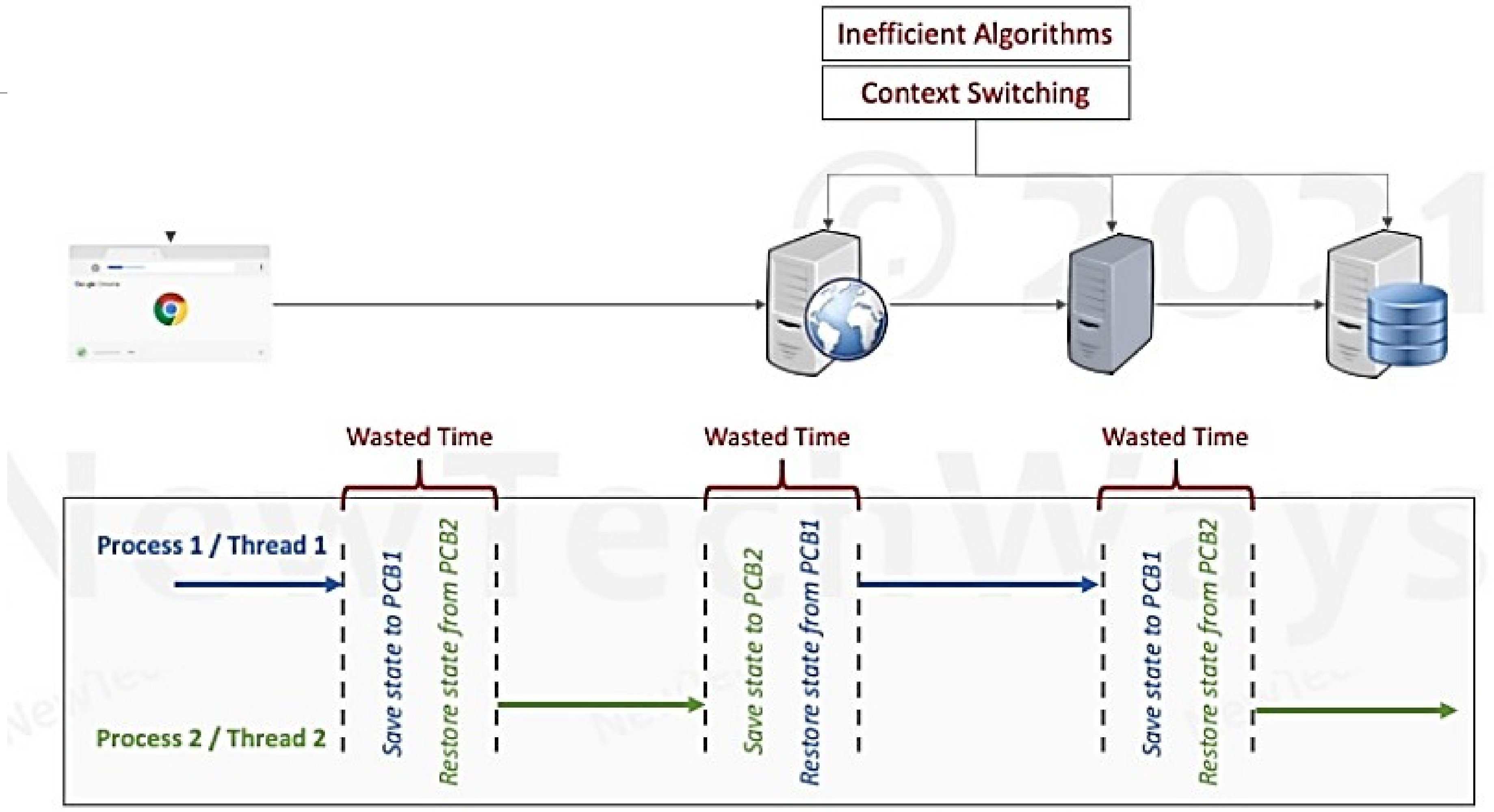
Memory Latency – Approaches



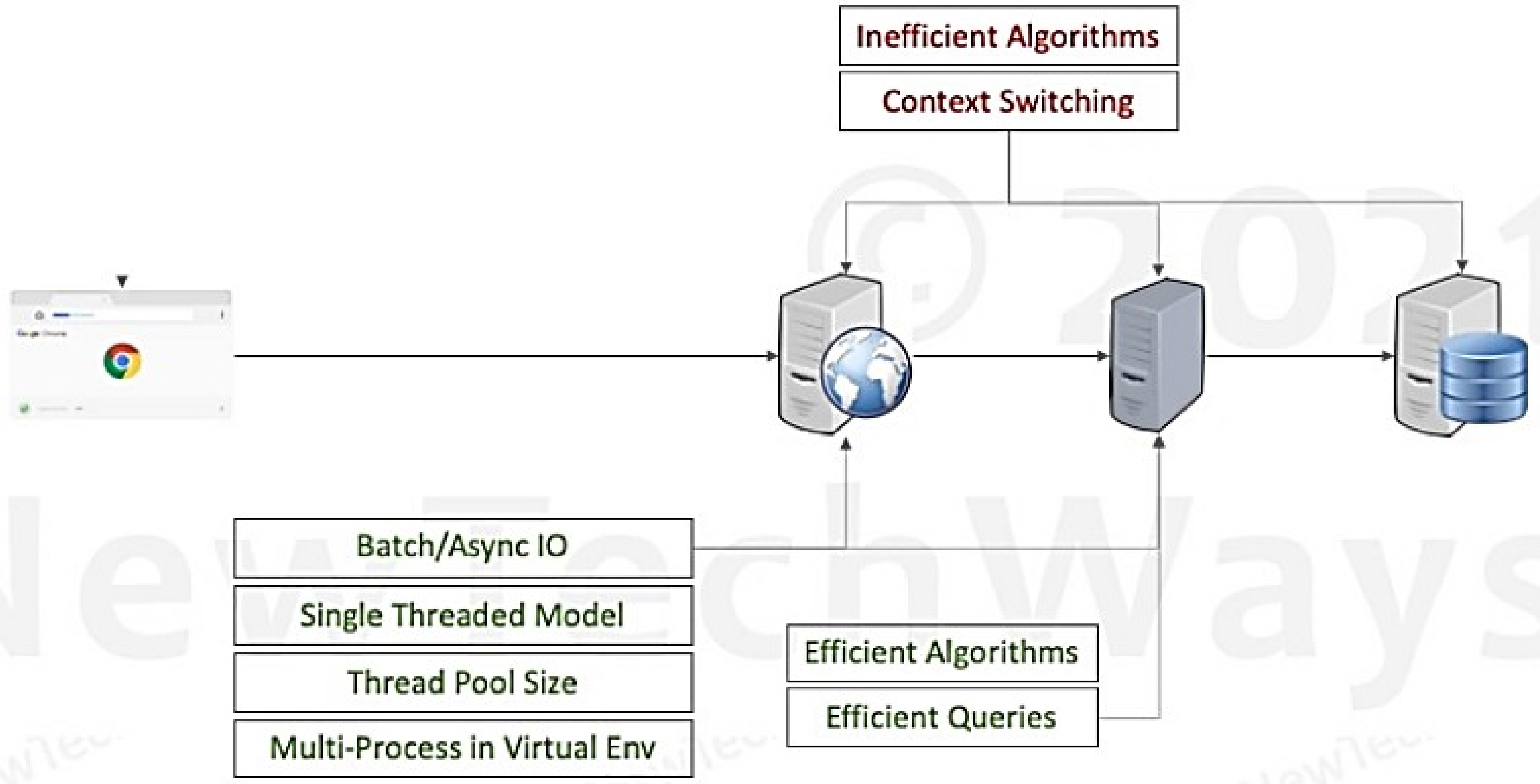
Disk Latency - Approaches



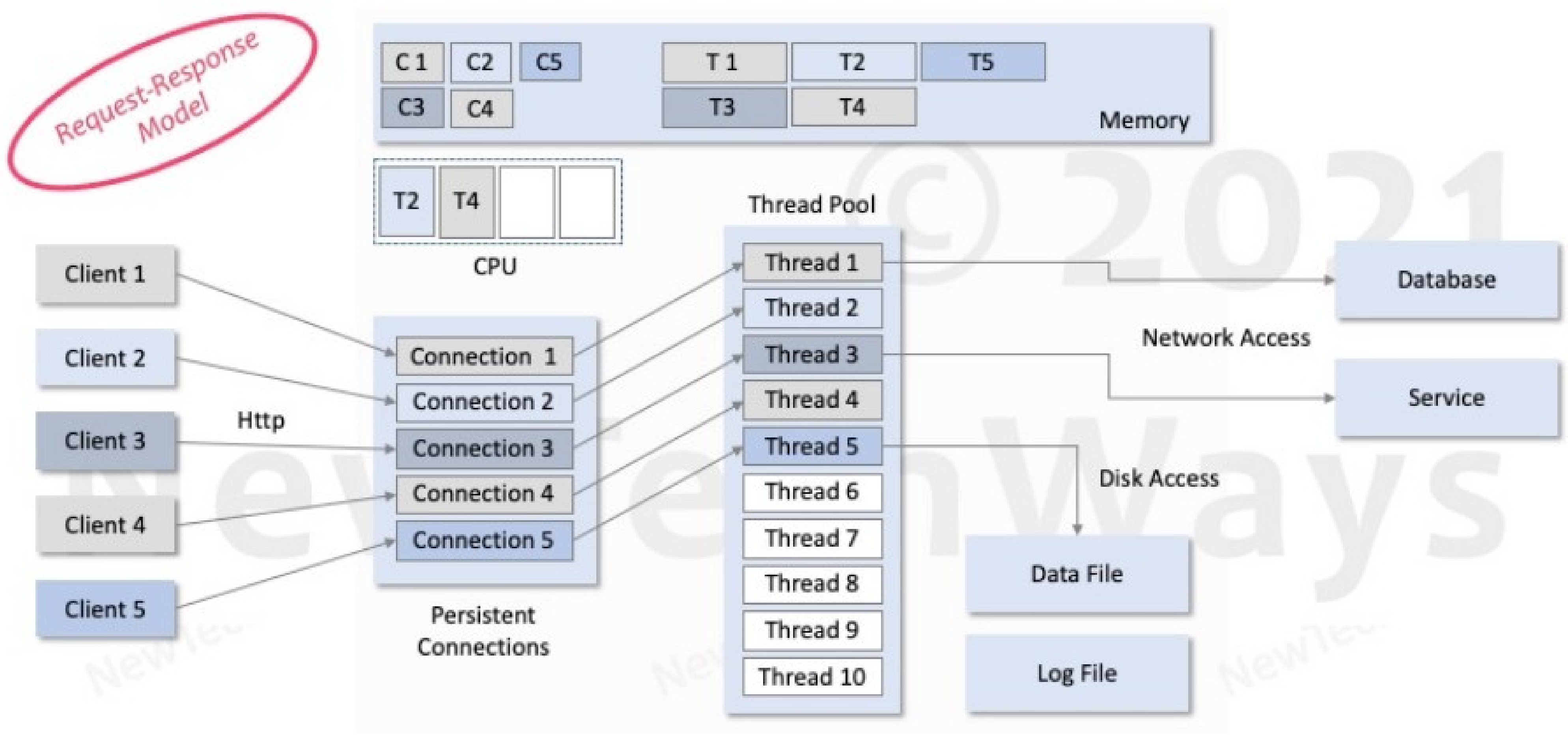
CPU Latency


Inefficient Algorithms
Context Switching
Wasted Time
Wasted Time
Wasted Time
Process 1 / Thread 1
Process 2 / Thread 2
Save state to PCB1
Restore state from PCB2
Save state to PCB2
Restore state from PCB1
Save state to PCB1
Restore state from PCB2

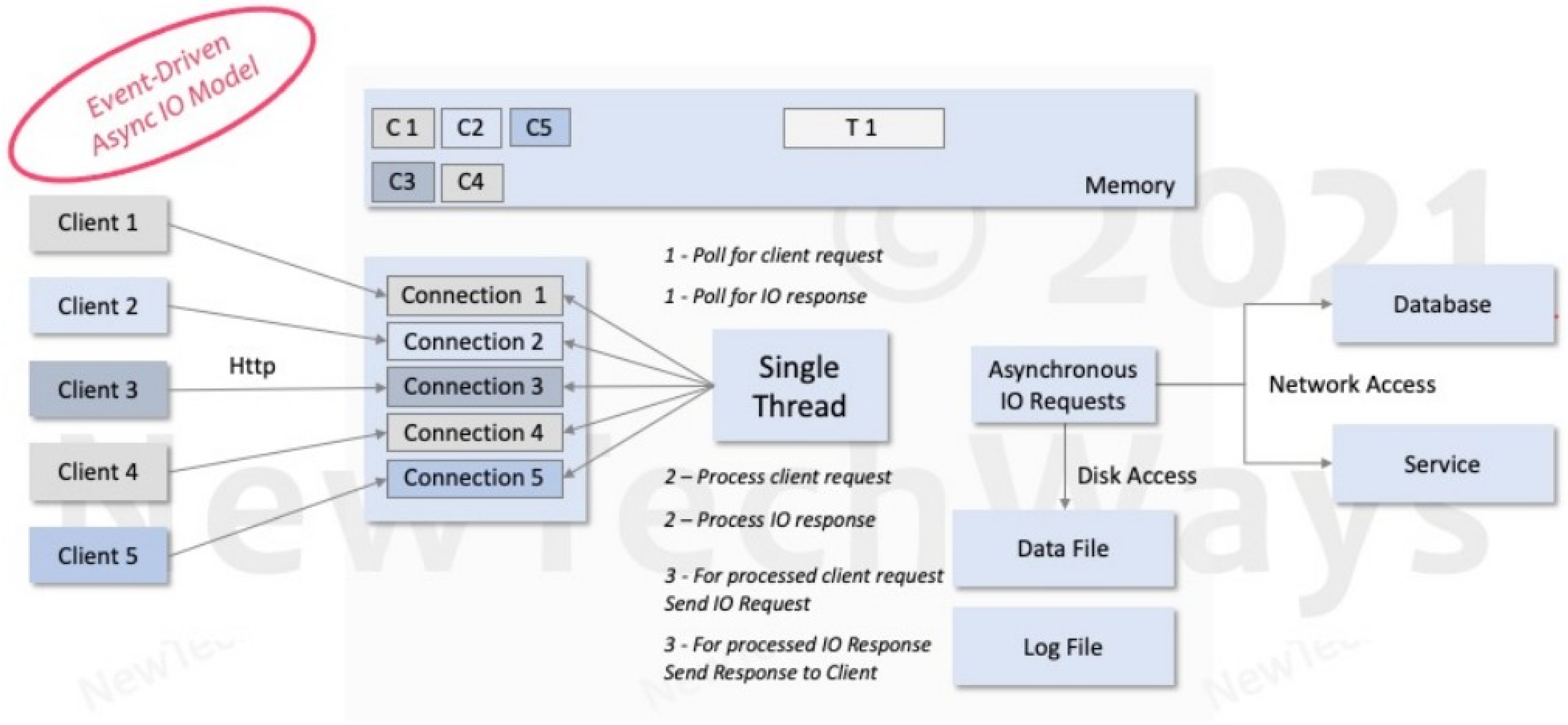
CPU Latency – Approaches



Apache Webserver Architecture



Nginx Architecture

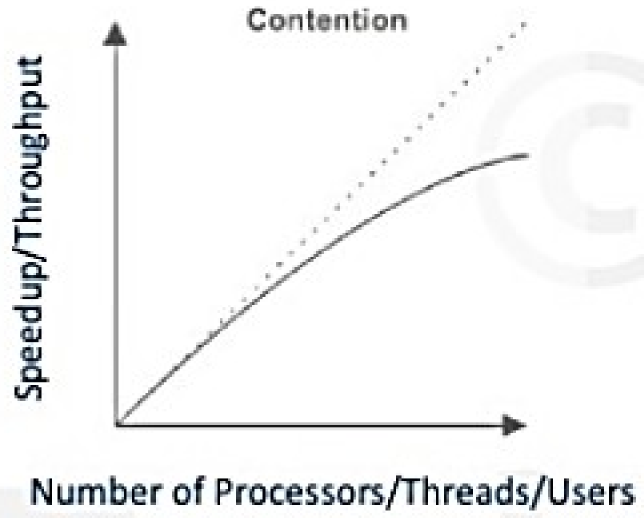


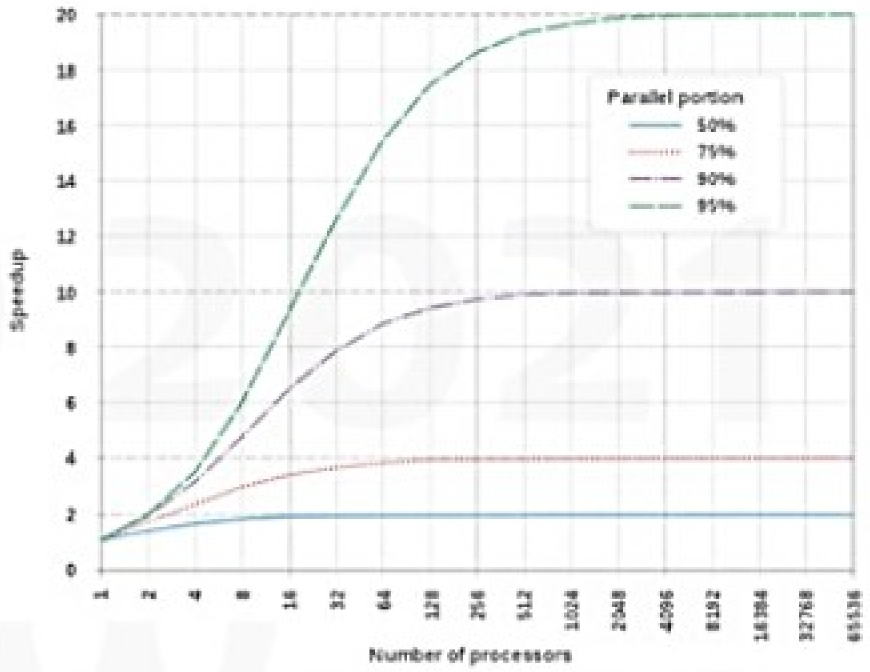
Parallel Request (Concurrency)

Concurrent Processing

- Amdhal's Law
  - $C(N) = \frac{N}{[1+\alpha(N-1)]}$

- C is capacity
- N is scaling dimension
  - like CPU or Load
- Alpha is resource contention
  - Alpha =0 , for linear performance

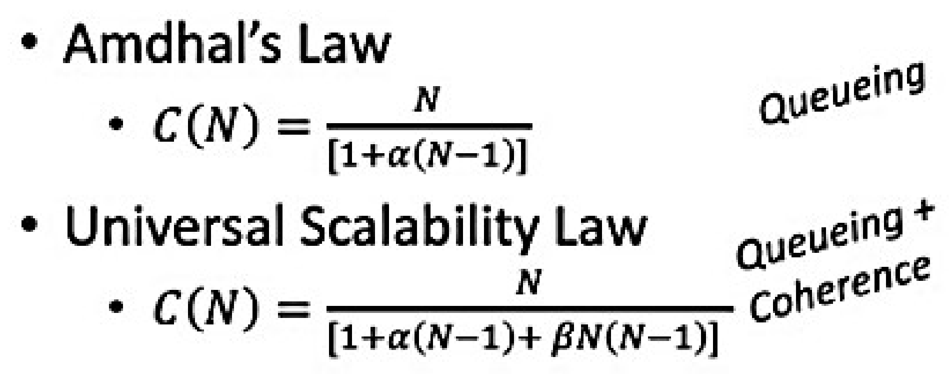




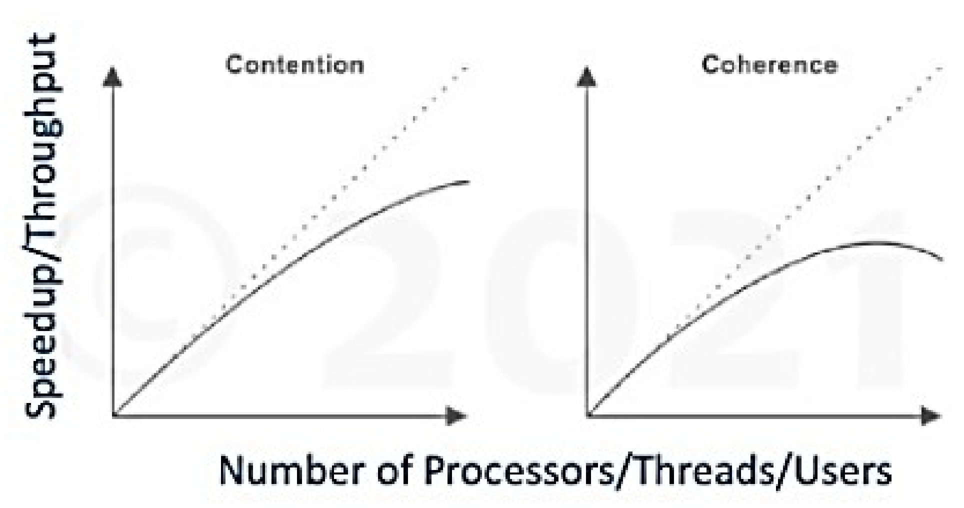
C(N) = theoretical speed

Parallel + Serial

Concurrent Processing

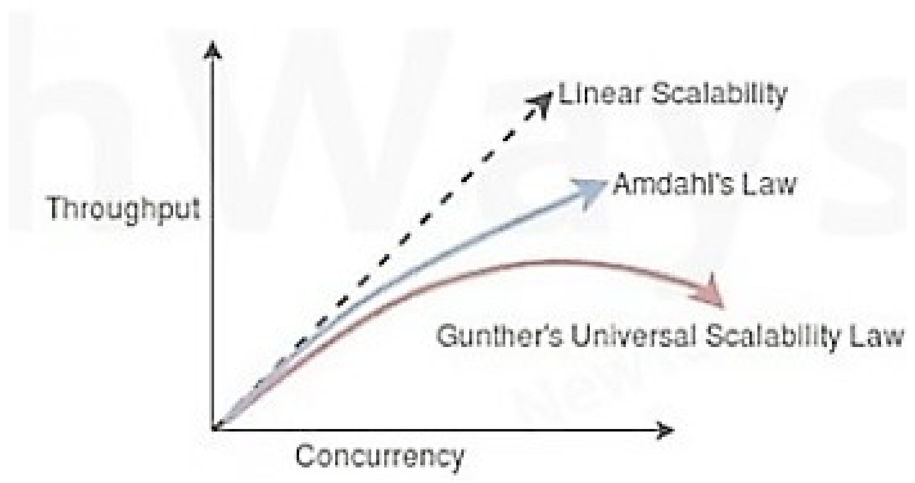


- Amdhal's Law
  - $C(N) = \frac{N}{[1+\alpha(N-1)]}$ Queueing
- Universal Scalability Law
  - $C(N) = \frac{N}{[1+\alpha(N-1)+ \beta N(N-1)]}$ Queueing + Coherence

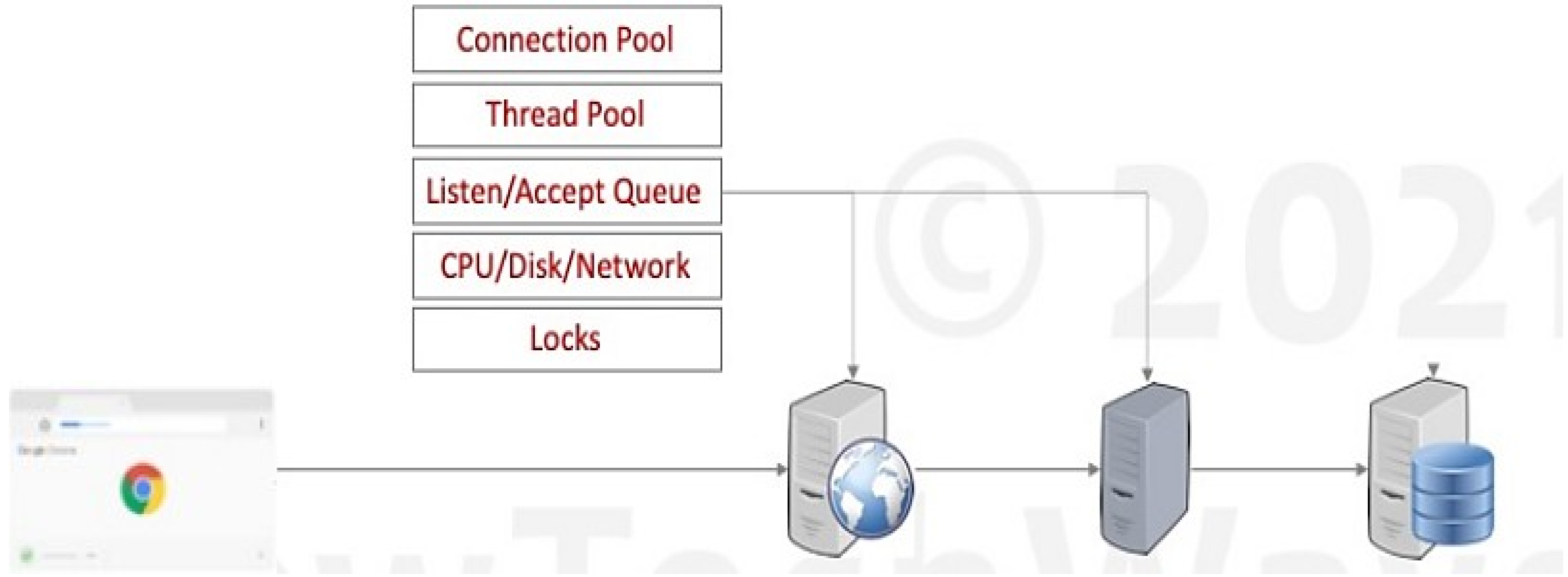


C is capacity
N is scaling dimension like CPU or load
Alpha represents resource contention
Beta represents coherence delay

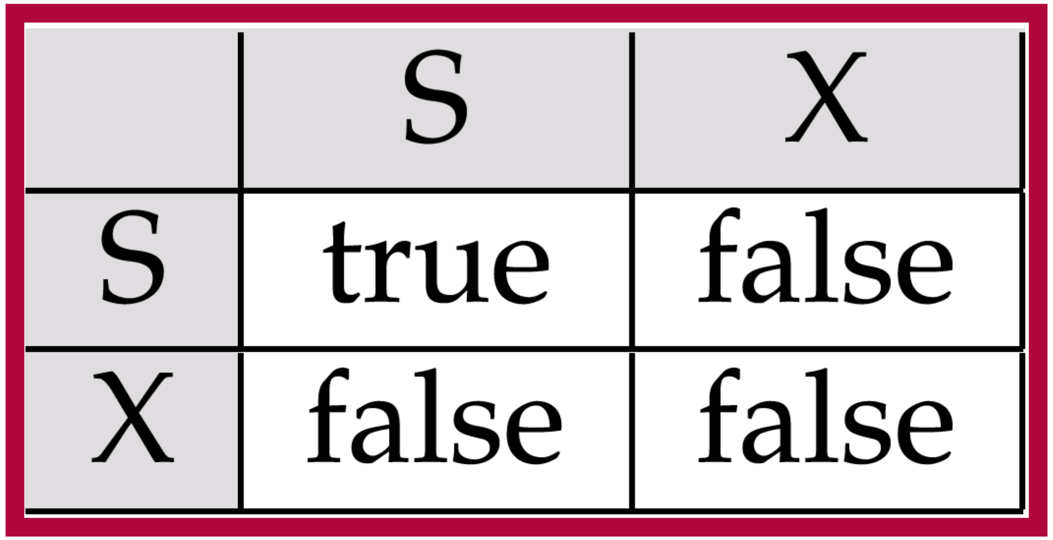
Linear performance when alpha and beta are zero



Contention


Connection Pool
Thread Pool
Listen/Accept Queue
CPU/Disk/Network
Locks
© 2021

Locking Compatibility Matrix



	S	X
S	true	false
X	false	false

Coherence

Suppose that two threads are working on SharedObj. If two threads run on different processors each thread may have its own local copy of sharedVariable.

If one thread modifies its value the change might not reflect in the original one in the main memory instantly. This depends on the write policy of cache. Now the other thread is not aware of the modified value which leads to data inconsistency.

“Volatile” keyword in java tells the compiler that the value of a variable must never be cached as its value may change outside of the scope of the program itself.

Scalability

Performance vs Scalability

FIXED LOAD

Performance

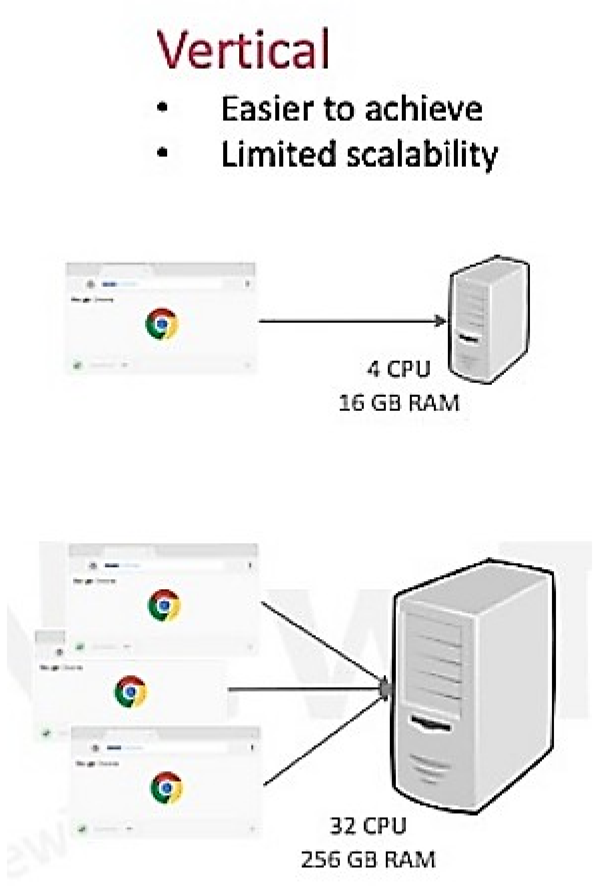
- Low Latency
- High Throughput
  - Concurrency
    - Single-Machine- Multi-Threading
    - Multi-Machine- Multi-Threading + Multiprocessing = Distributed Processing

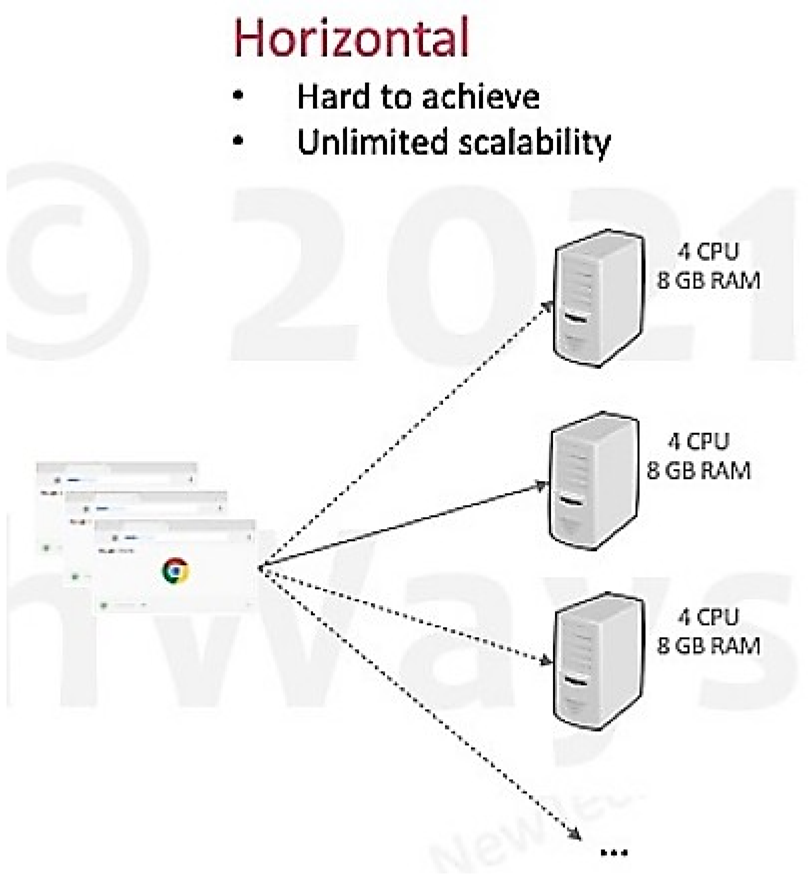
VARIABLE LOAD

Scalability

- High Throughput
  - Ability of a system to increase its throughput by adding more hardware capacity.
  - Both Ways: Up and Down

Vertical & Horizontal Scalability

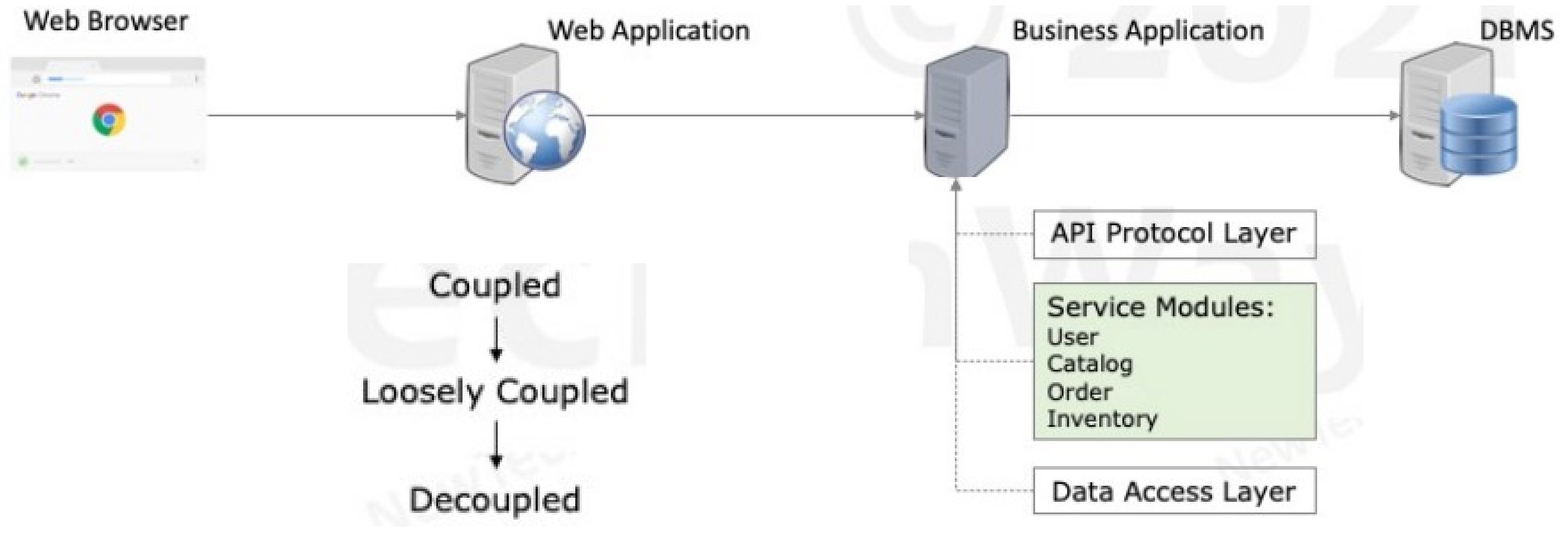




Modularity

Scalable architecture starts with Modularity

- Provides the foundation for breaking an application into more specialized functions/services.

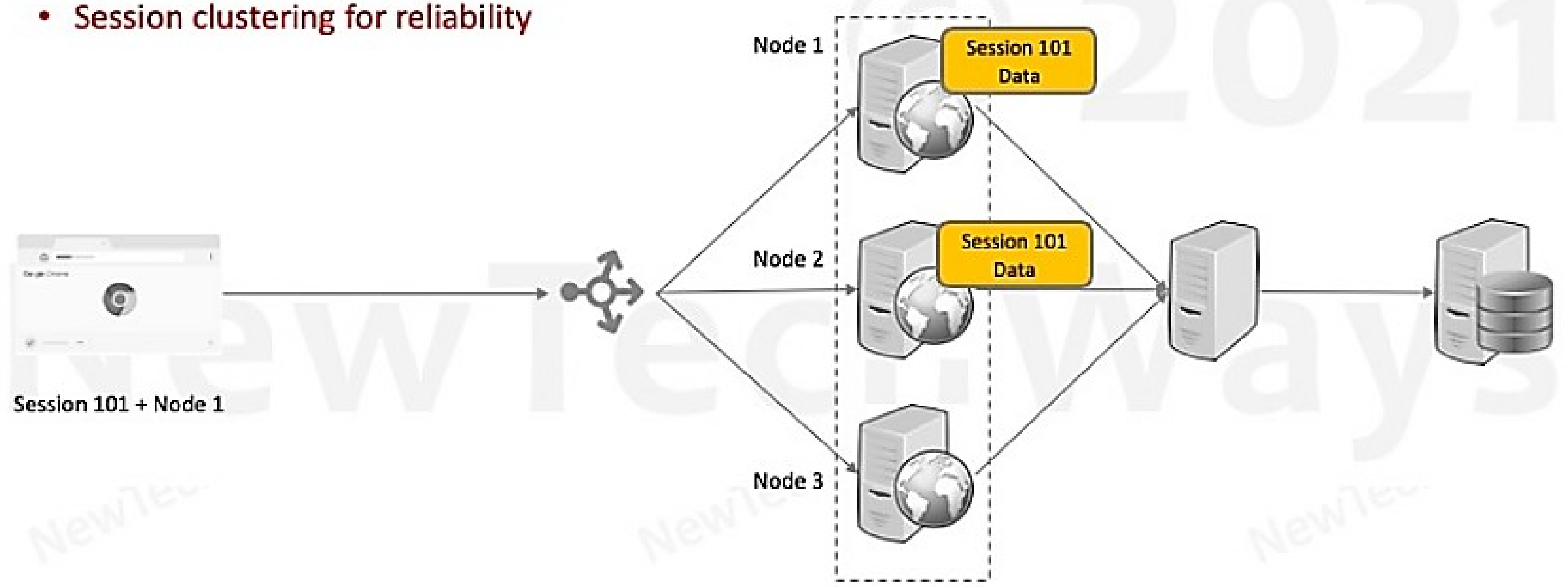


Horizontal Scaling Methods

1. **Services**
2. **Replication**
    1. Stateful
    2. Stateless
3. **Portioning**
    1. Vertical / Functionality Portioning
    2. Database portioning
4. **Asynchronous Calls**

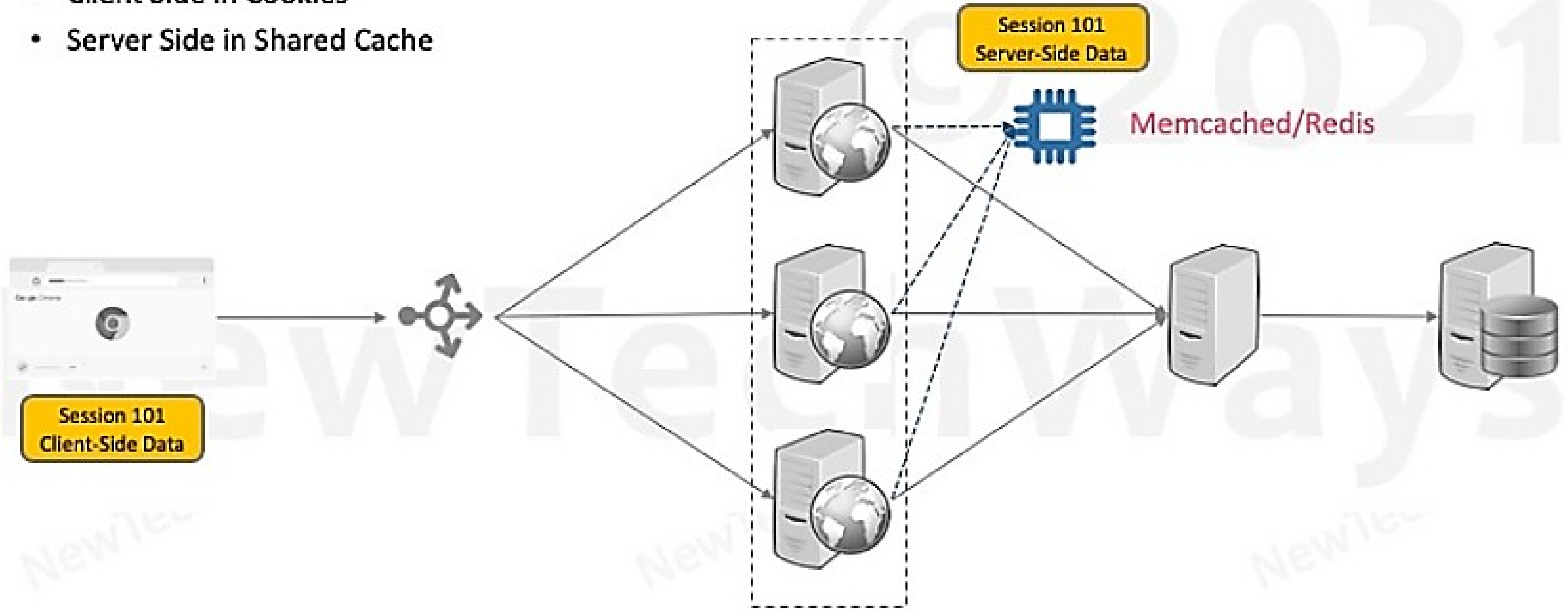
Web Stateful Replication

- When low latency is required
  - Sticky sessions/Session affinity
  - Sessions occupy memory
  - Session clustering for reliability



Web Stateless Replication

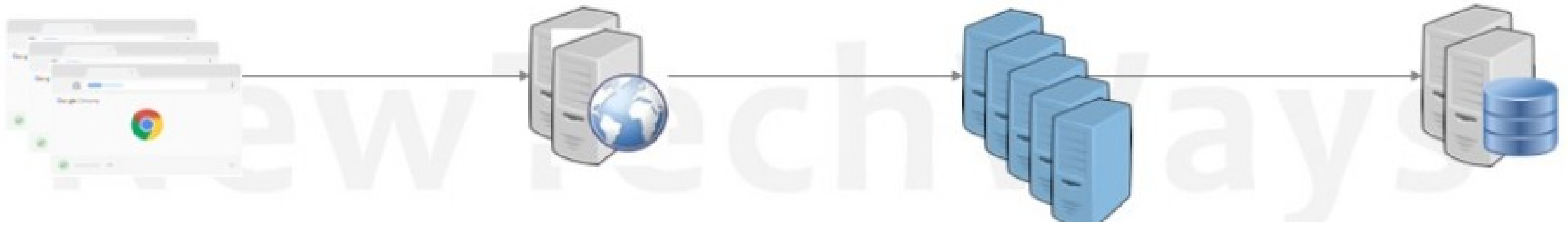
- For higher scalability at the expense of higher latency
- Session data can be stored on
  - Client Side in Cookies
  - Server Side in Shared Cache



Need For Specialized Services

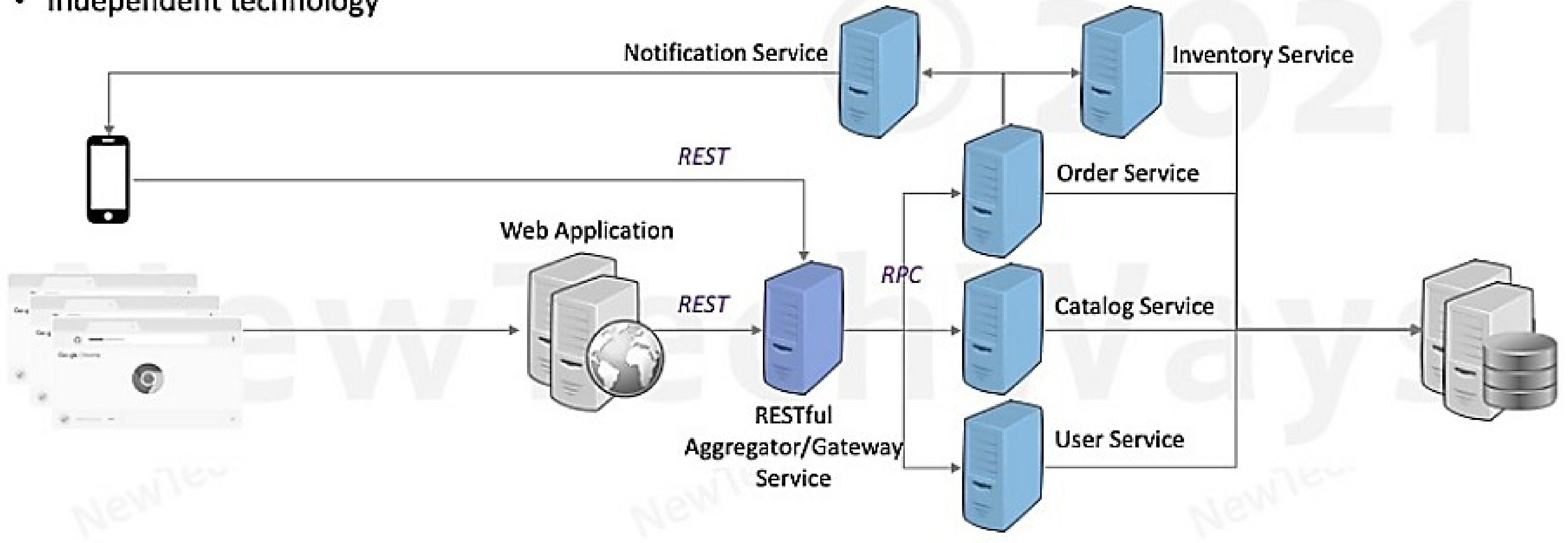
Service Modules:

- User
- Catalog
- Order
- Inventory
- Notification



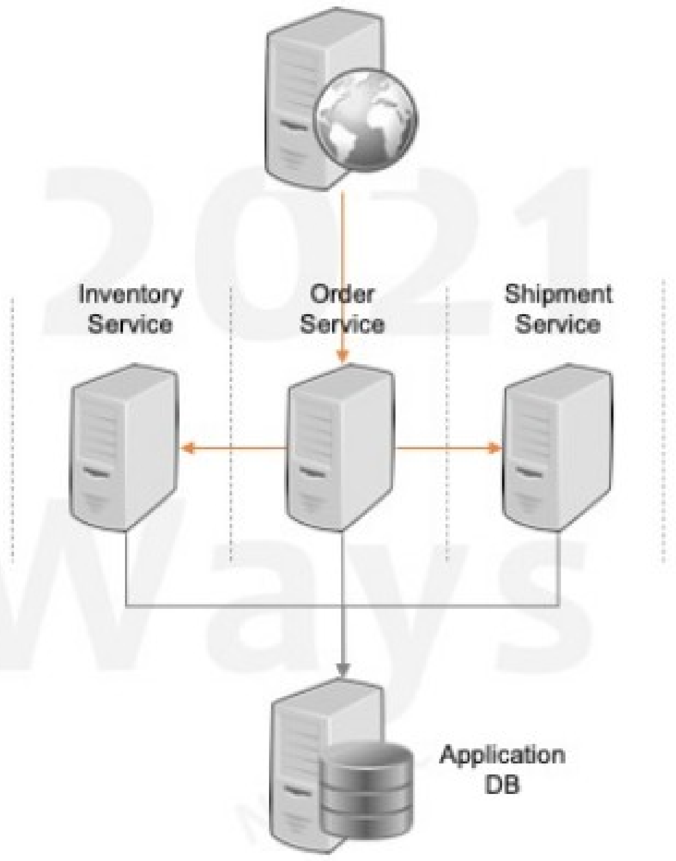
Specialized Services (SOAP/REST-Services)

- Partially independent development and deployment
- Independent scalability
- Independent technology

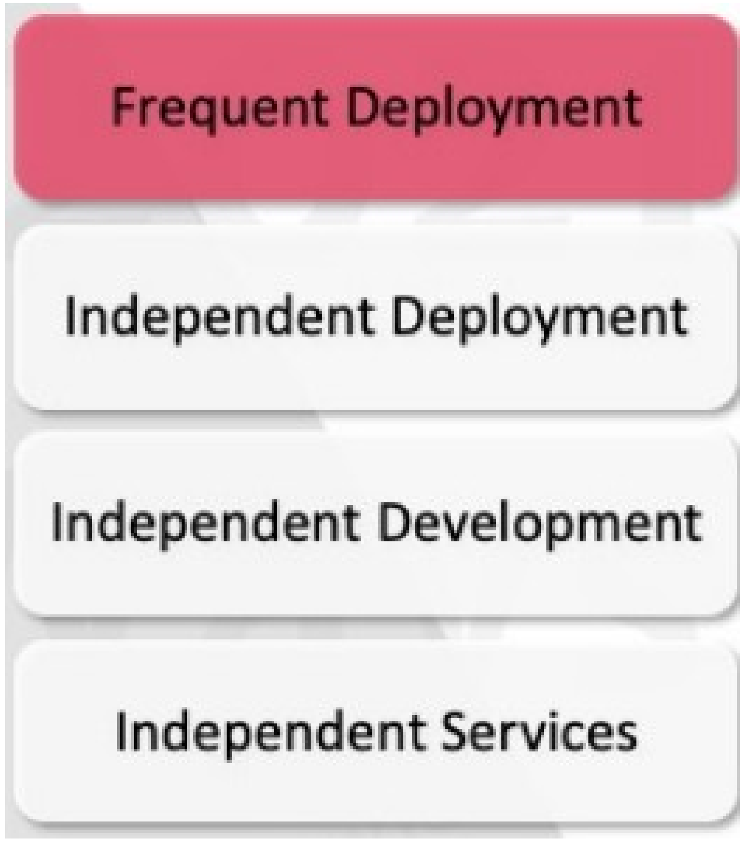


Service Oriented Architecture

- Independent
  - Each service can have its own technology stack, libraries, frameworks etc.
  - Each service can be scaled independently and differently
- Not Independent
  - Common interface schema
    - XML schema
  - Common database schema
    - RDBMS schema
- Issues
  - Service development may be independent but not deployment
  - Single database has scalability limitations



Microservices

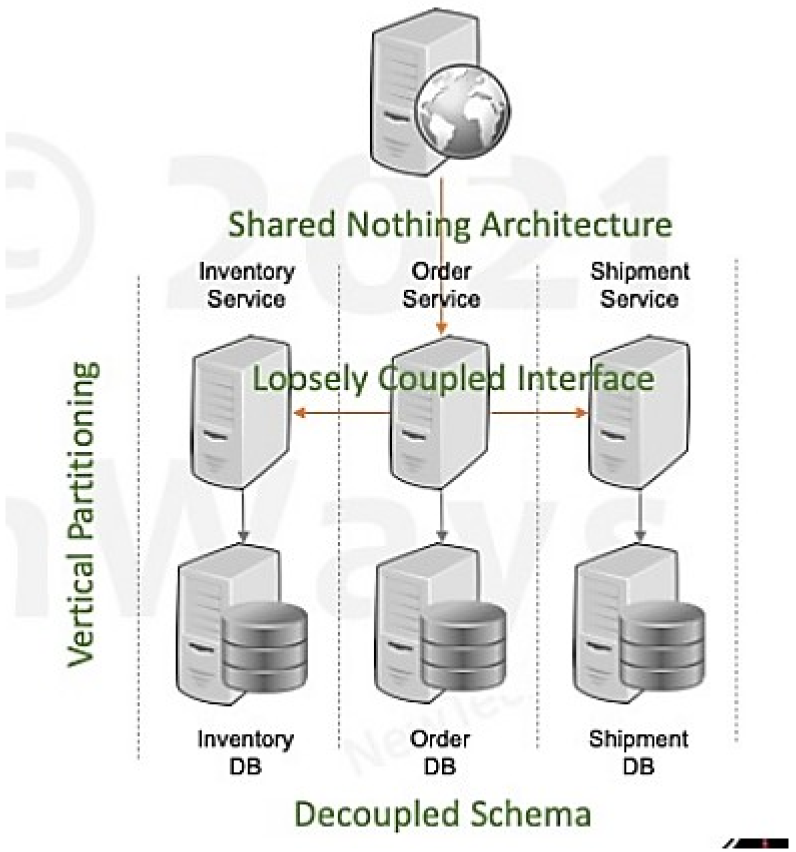


Frequent Deployment

Independent Deployment

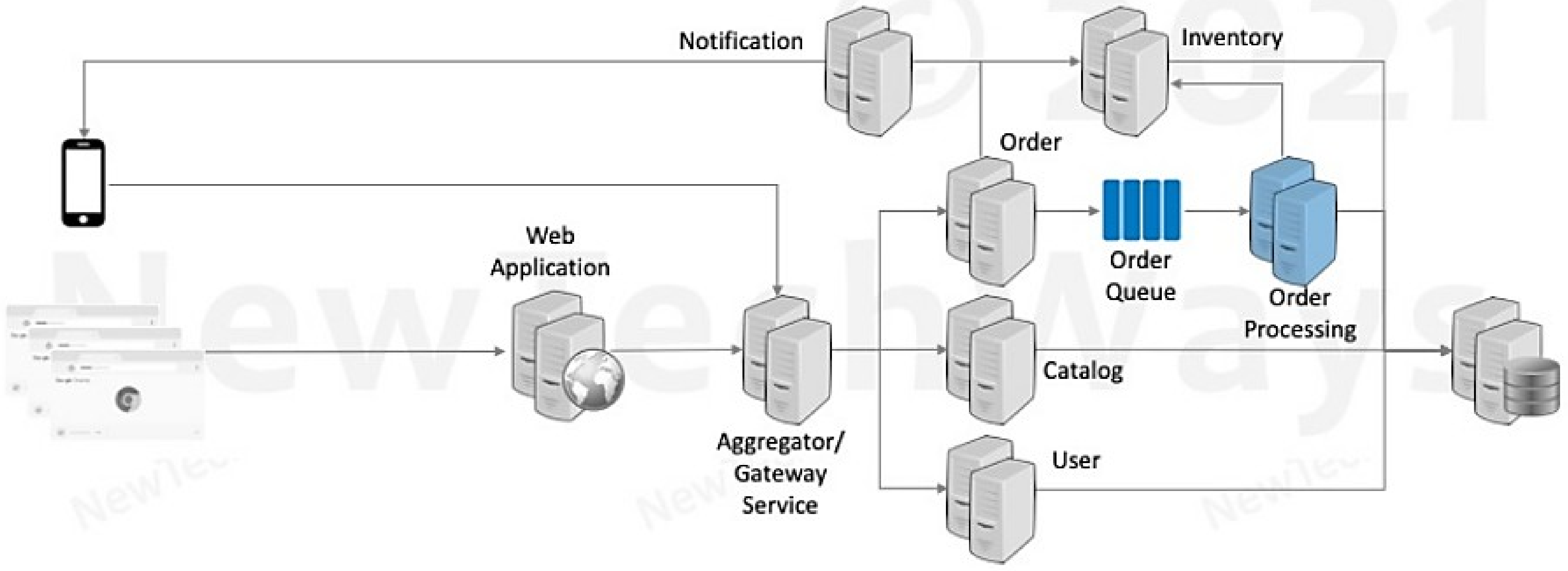
Independent Development

Independent Services



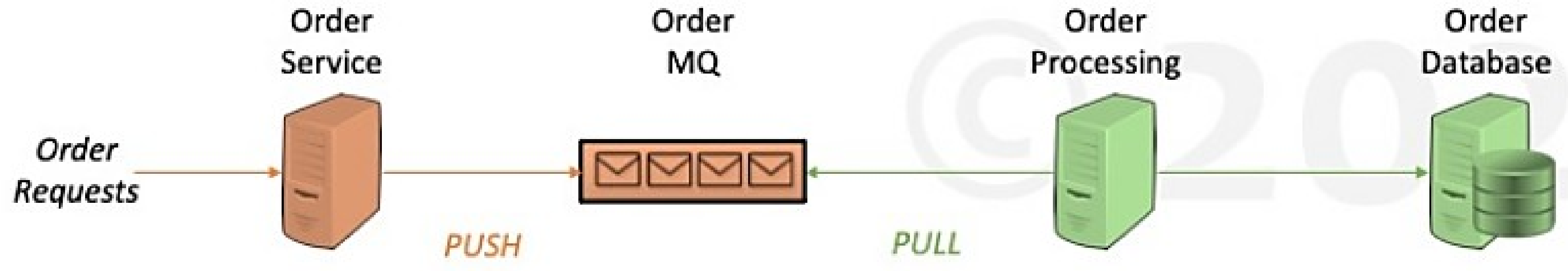
Asynchronous Services

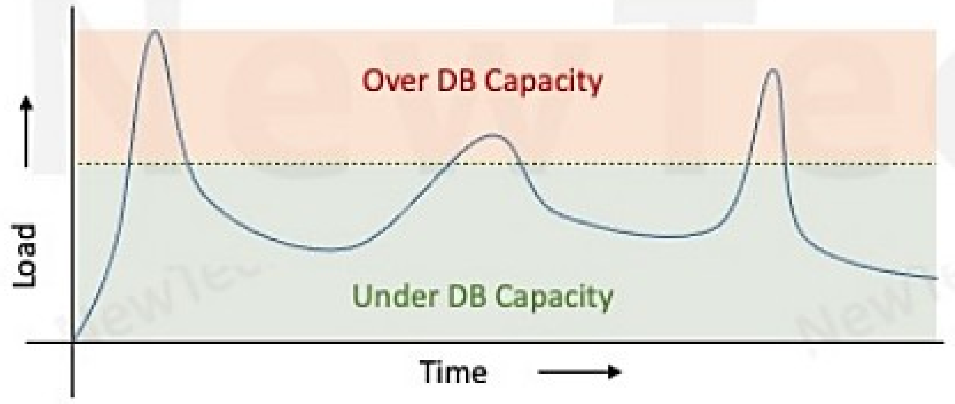
- Async services effectively reduces write load from a database

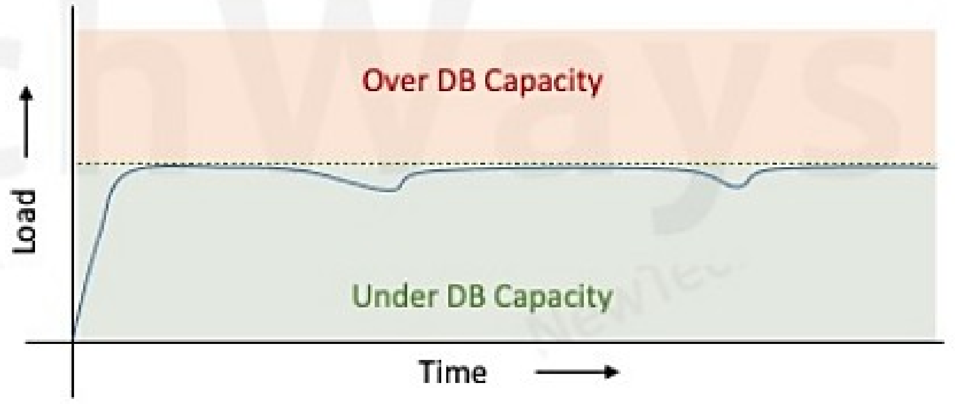


Asynchronous Processing & Scalability

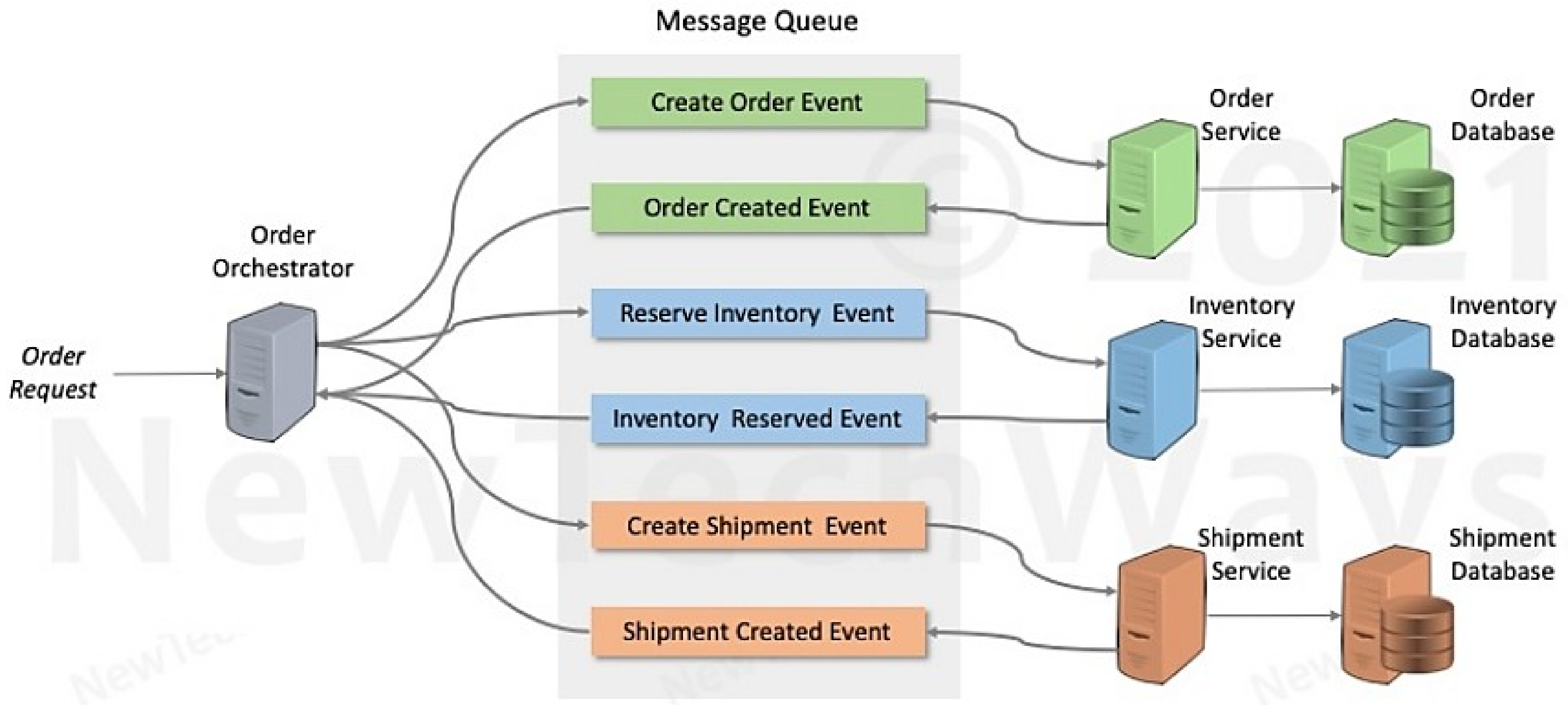
- Async services require infrastructure for average load as opposed to peak load







Micro-Services Event Driven Transactions



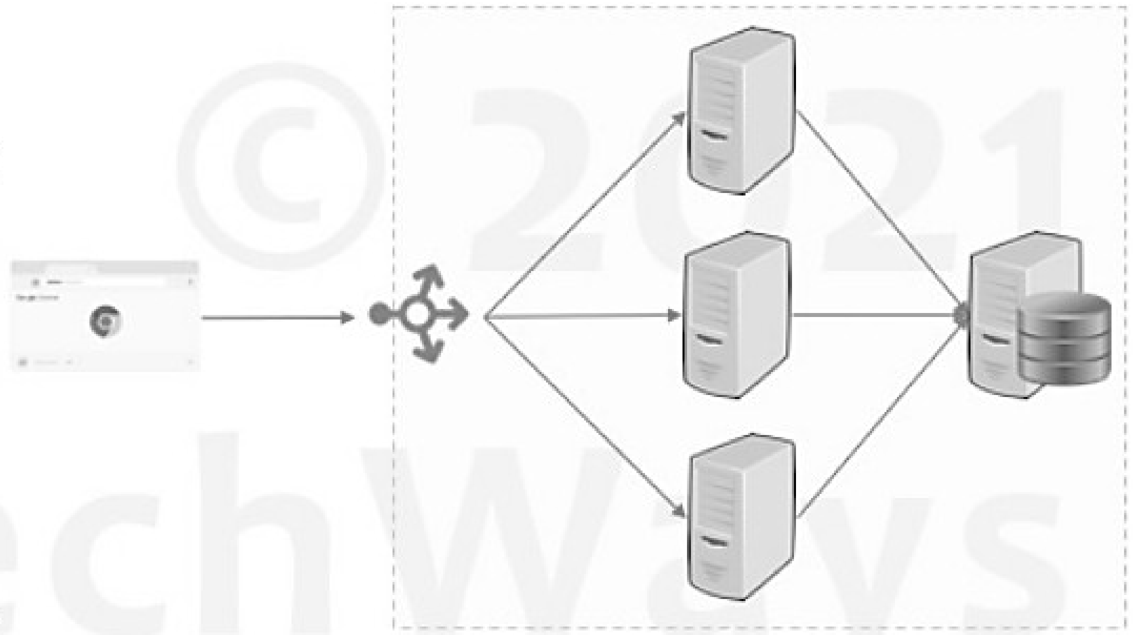
Reliability

Reliability

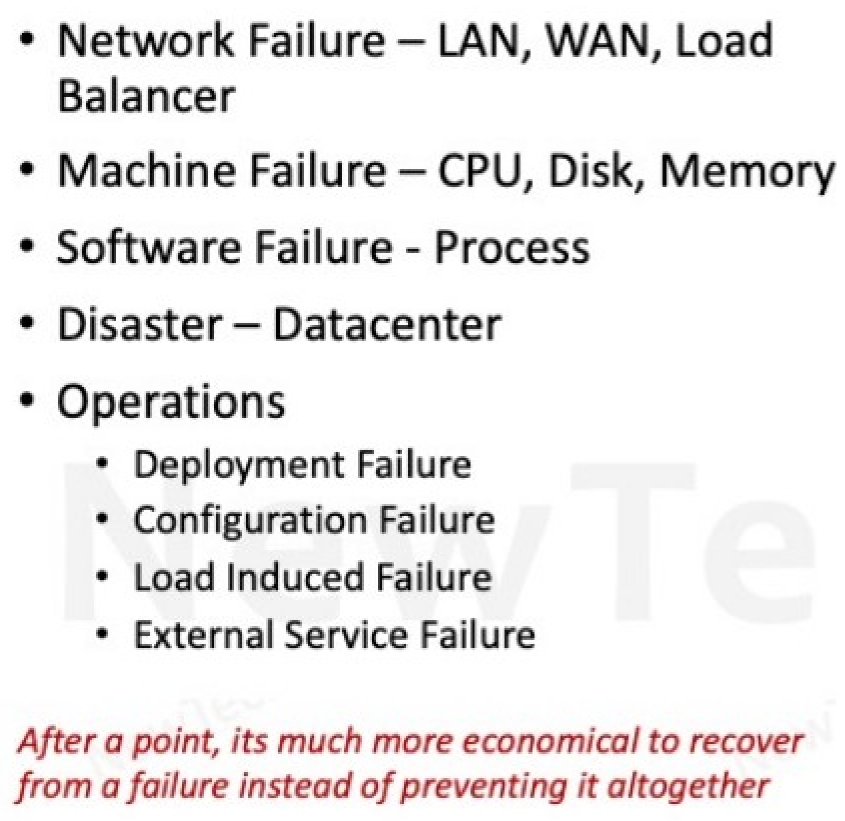
Software reliability is defined as the probability of failure-free operation of a software system for a specified time in a specified environment.

Failures in Large-Scale Systems

- Large scale systems are generally distributed systems
  - Large number of components
  - Large number of component instances
- Failures can be
  - Partial
  - Independent
  - Single point of failures
- Increased chance of partial failures
- Partial failures can lead to complete system failures

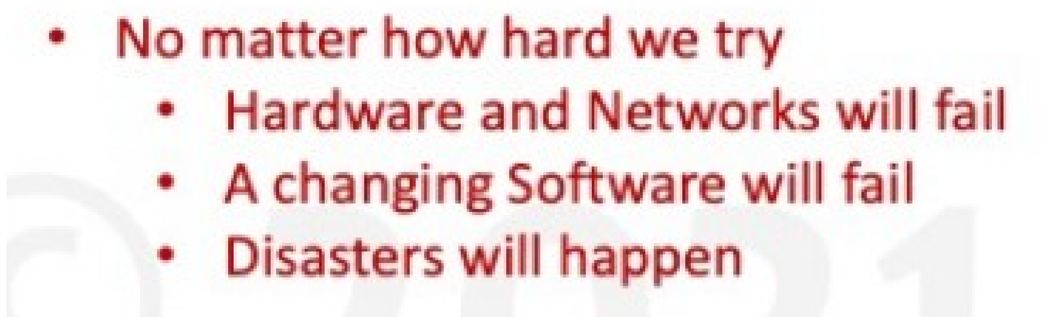


Partial Failures



- Network Failure – LAN, WAN, Load Balancer
- Machine Failure – CPU, Disk, Memory
- Software Failure - Process
- Disaster – Datacenter
- Operations
  - Deployment Failure
  - Configuration Failure
  - Load Induced Failure
  - External Service Failure

After a point, its much more economical to recover from a failure instead of preventing it altogether

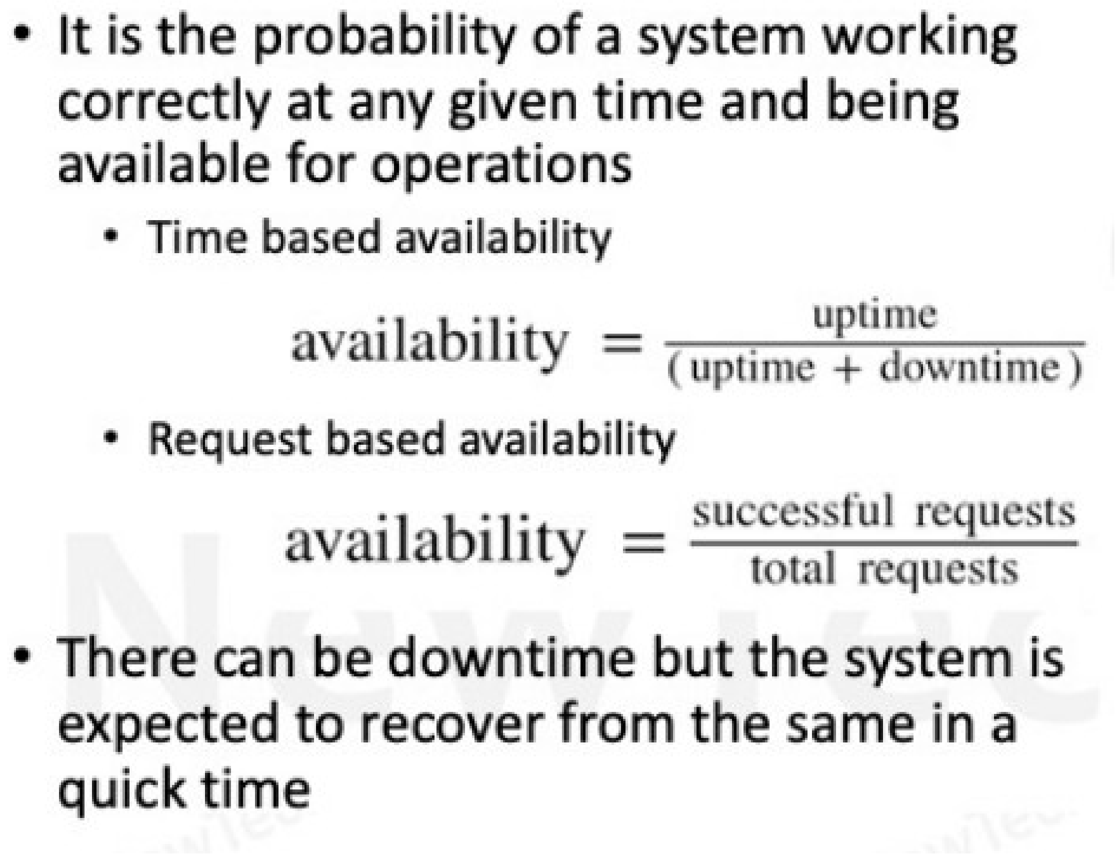


- No matter how hard we try
  - Hardware and Networks will fail
  - A changing Software will fail
  - Disasters will happen

Reliability Engineering

1. Reliability
2. Availability
3. Fault Tolerance

Availability



- It is the probability of a system working correctly at any given time and being available for operations
  - Time based availability

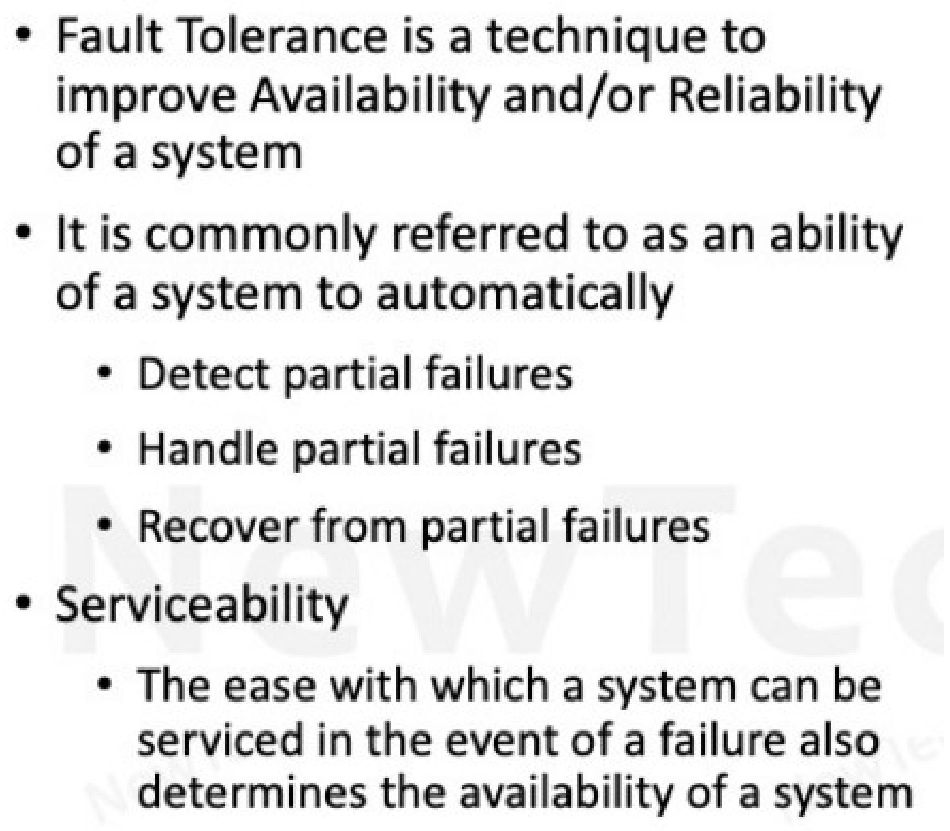
$$\text{availability} = \frac{\text{uptime}}{(\text{uptime} + \text{downtime})}$$

  - Request based availability

$$\text{availability} = \frac{\text{successful requests}}{\text{total requests}}$$

- There can be downtime but the system is expected to recover from the same in a quick time

Fault Tolerance

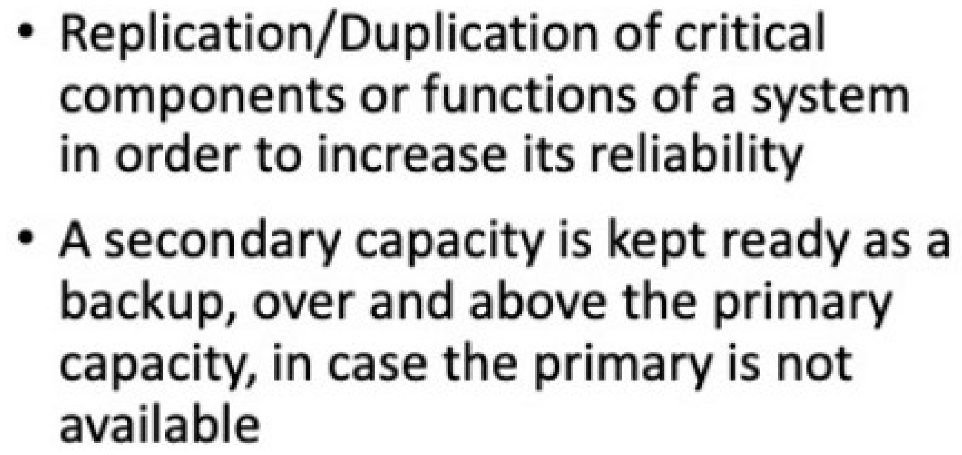


- Fault Tolerance is a technique to improve Availability and/or Reliability of a system
- It is commonly referred to as an ability of a system to automatically
  - Detect partial failures
  - Handle partial failures
  - Recover from partial failures
- Serviceability
  - The ease with which a system can be serviced in the event of a failure also determines the availability of a system

Fault Tolerant Design

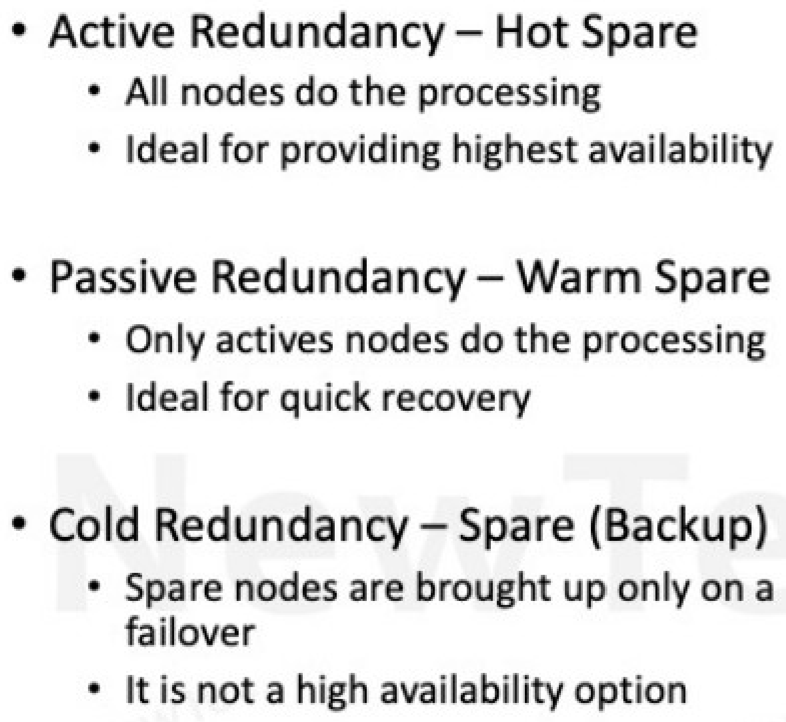
1. Redundancy
2. Fault Detection
3. Recovery

Redundancy

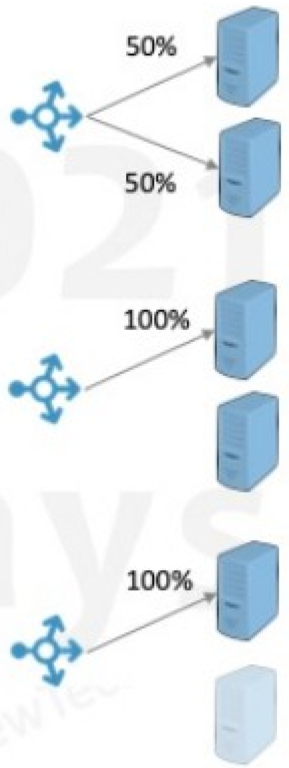


- Replication/Duplication of critical components or functions of a system in order to increase its reliability
- A secondary capacity is kept ready as a backup, over and above the primary capacity, in case the primary is not available

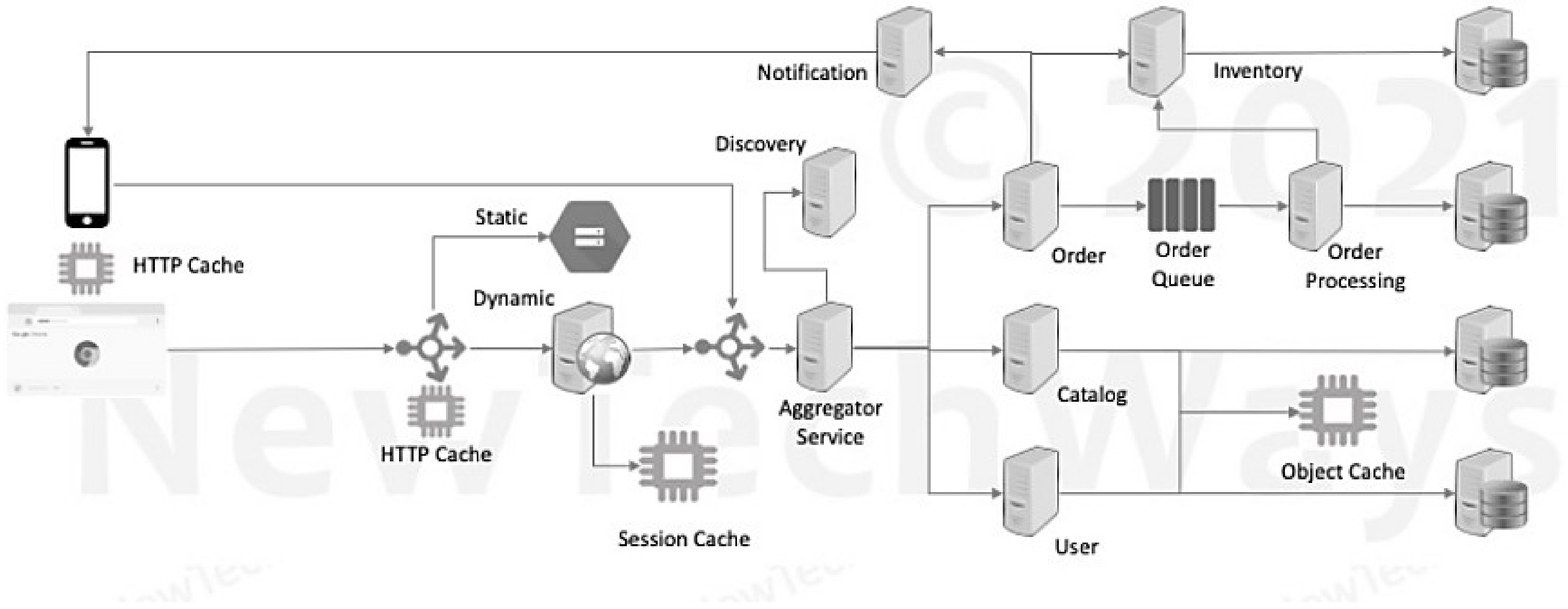
Types of Redundancy



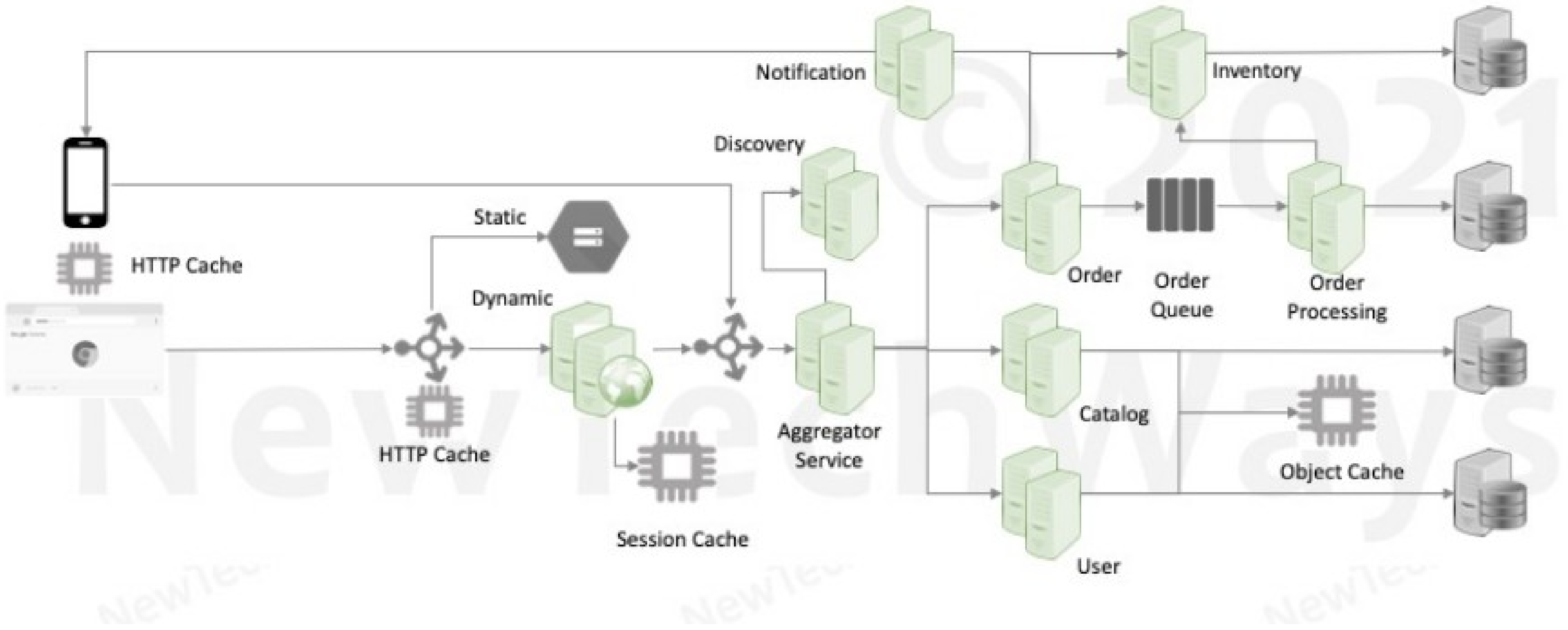
- Active Redundancy – Hot Spare
  - All nodes do the processing
  - Ideal for providing highest availability
- Passive Redundancy – Warm Spare
  - Only actives nodes do the processing
  - Ideal for quick recovery
- Cold Redundancy – Spare (Backup)
  - Spare nodes are brought up only on a failover
  - It is not a high availability option



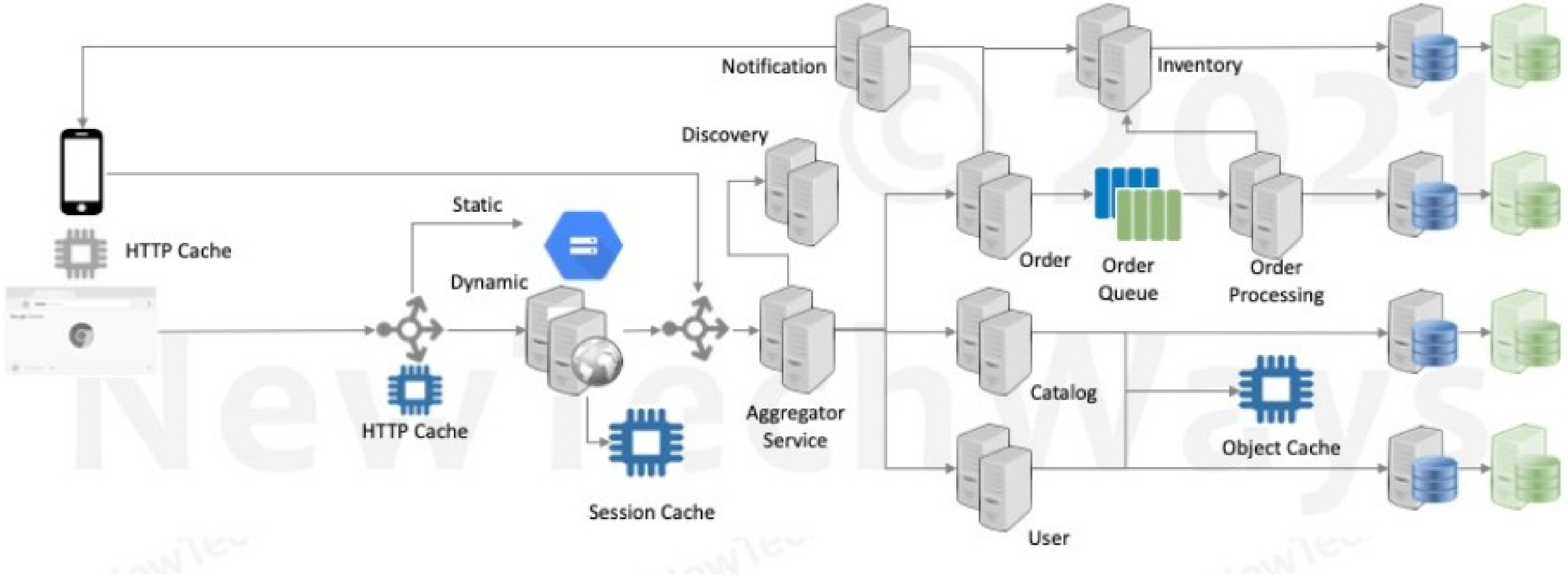
Single Points of Failure



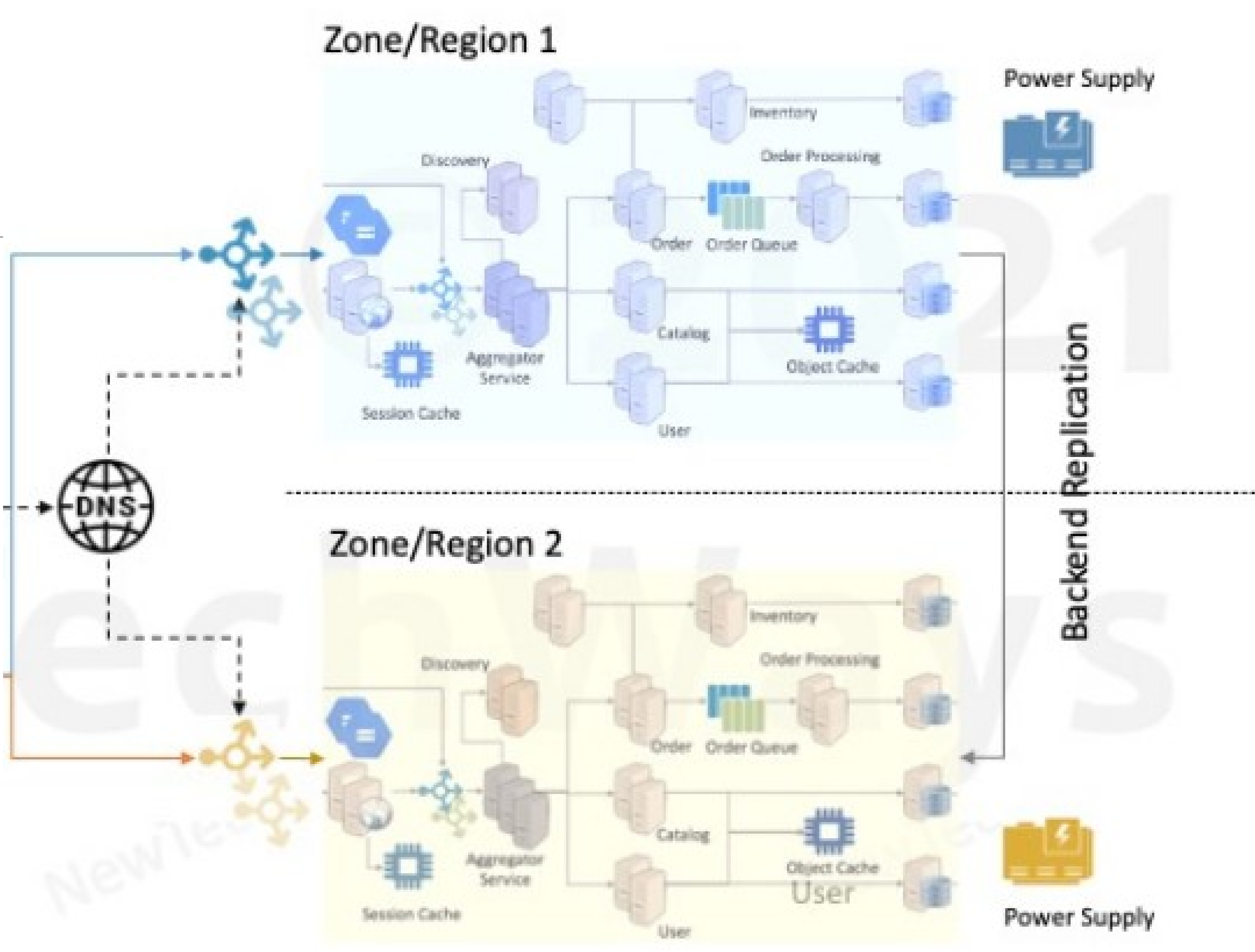
Redundancy for Stateless Components



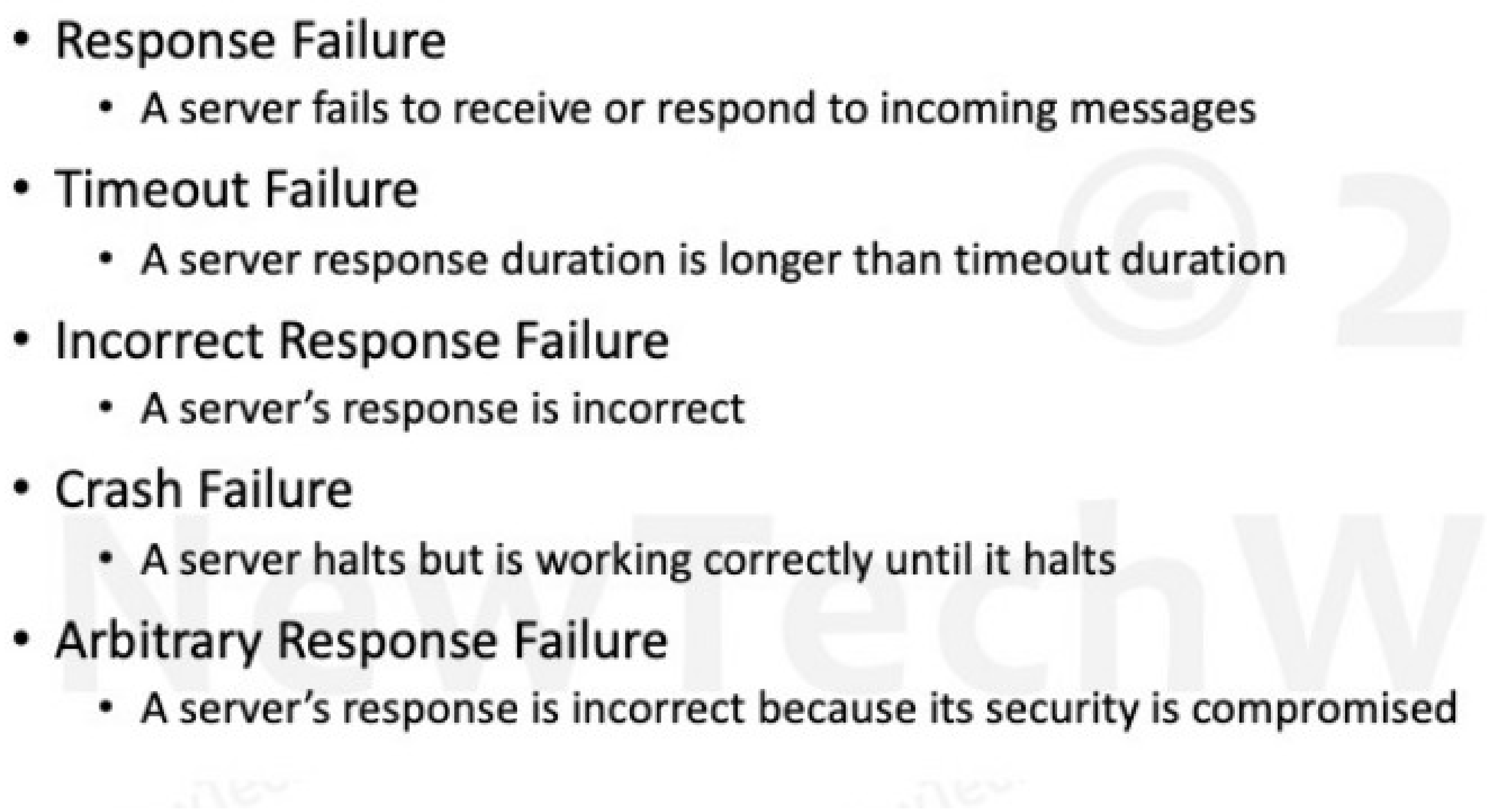
Redundancy for Stateful Components



Datacenter Redundancy



Fault Models

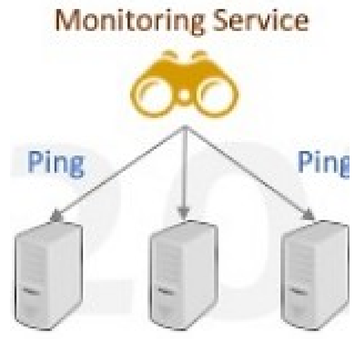


- Response Failure
  - A server fails to receive or respond to incoming messages
- Timeout Failure
  - A server response duration is longer than timeout duration
- Incorrect Response Failure
  - A server's response is incorrect
- Crash Failure
  - A server halts but is working correctly until it halts
- Arbitrary Response Failure
  - A server's response is incorrect because its security is compromised

Health Checks

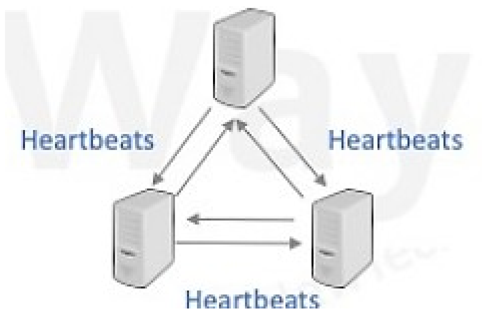
External Monitoring Service

- Ping based



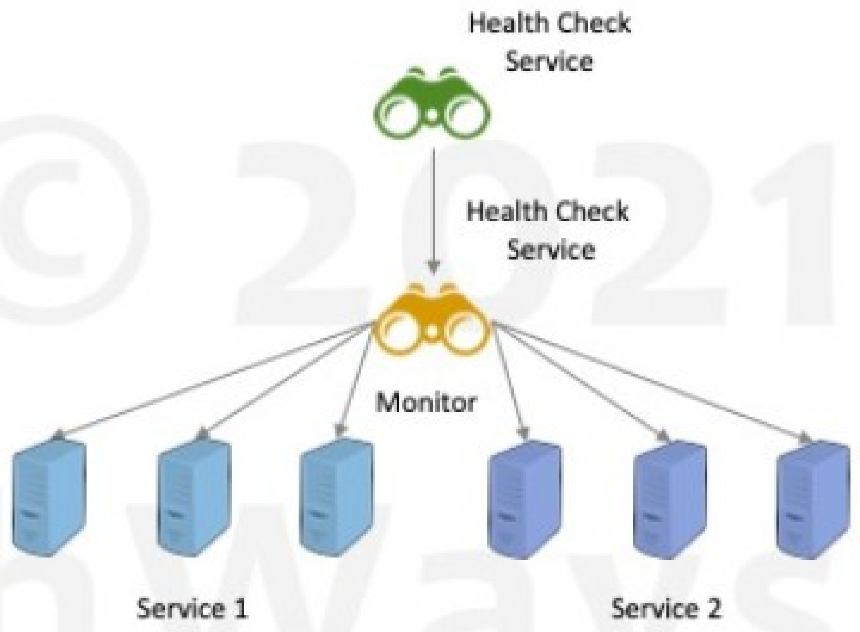
Internal Cluster Monitoring

- Heartbeat based



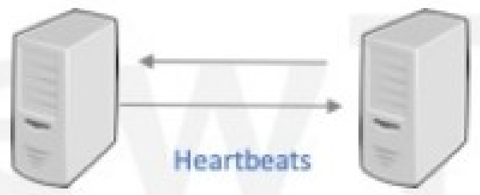
External Monitoring Service

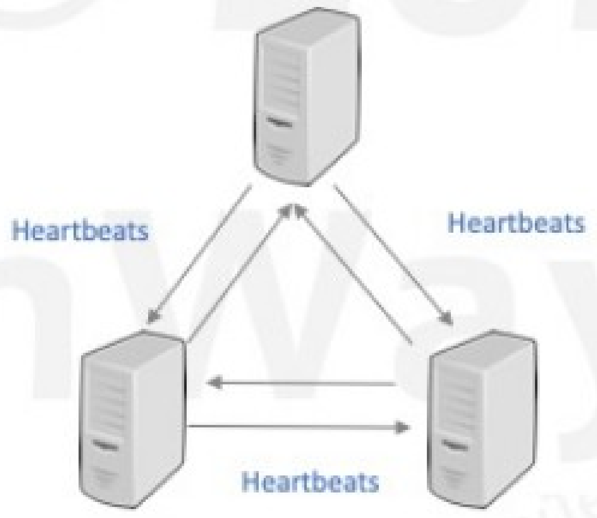
- Health check service generates
  - Alerts – for recovery
  - Events – for scaling
- Application Health Checks
  - HTTP Response
  - TCP Response
- Periodic Health Checks
  - Response Code
  - Response Time
  - Number of Retries
    - Up
    - Down



Internal Cluster Monitoring

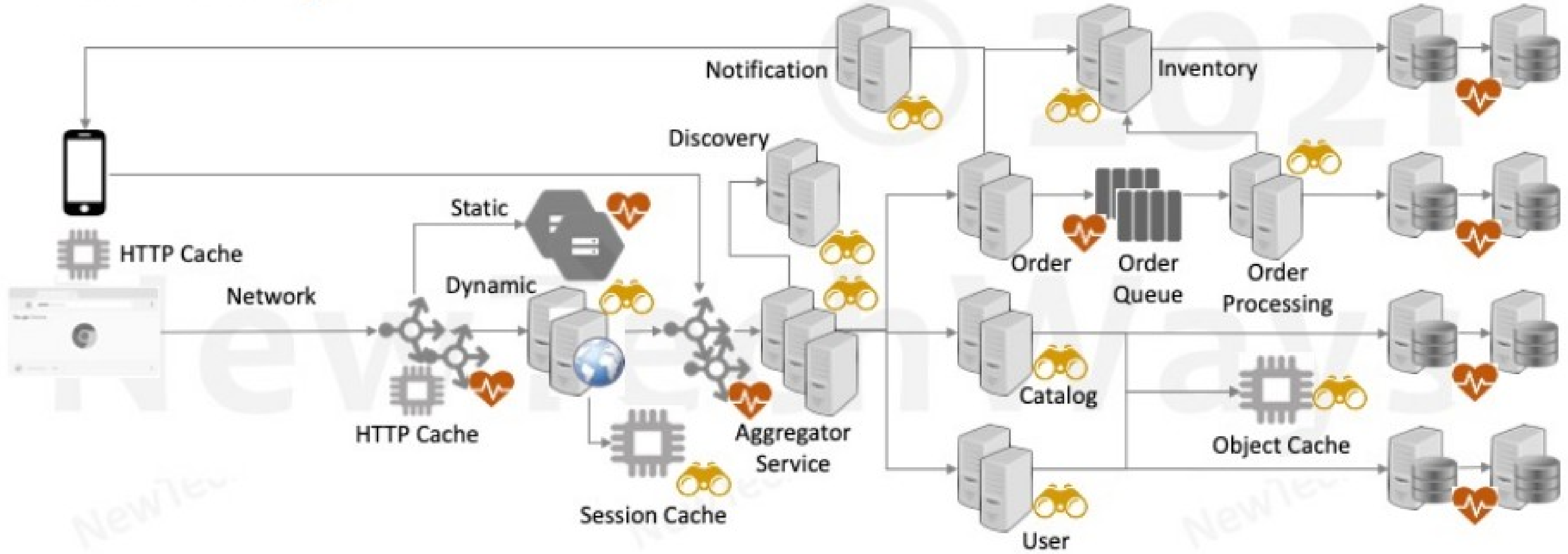
- Periodic exchange of heartbeats between redundancy cluster nodes
  - Requires protocols for communication and recovery
- Useful for stateful cluster components
  - Examples are NoSQL DB cluster and Load Balancers





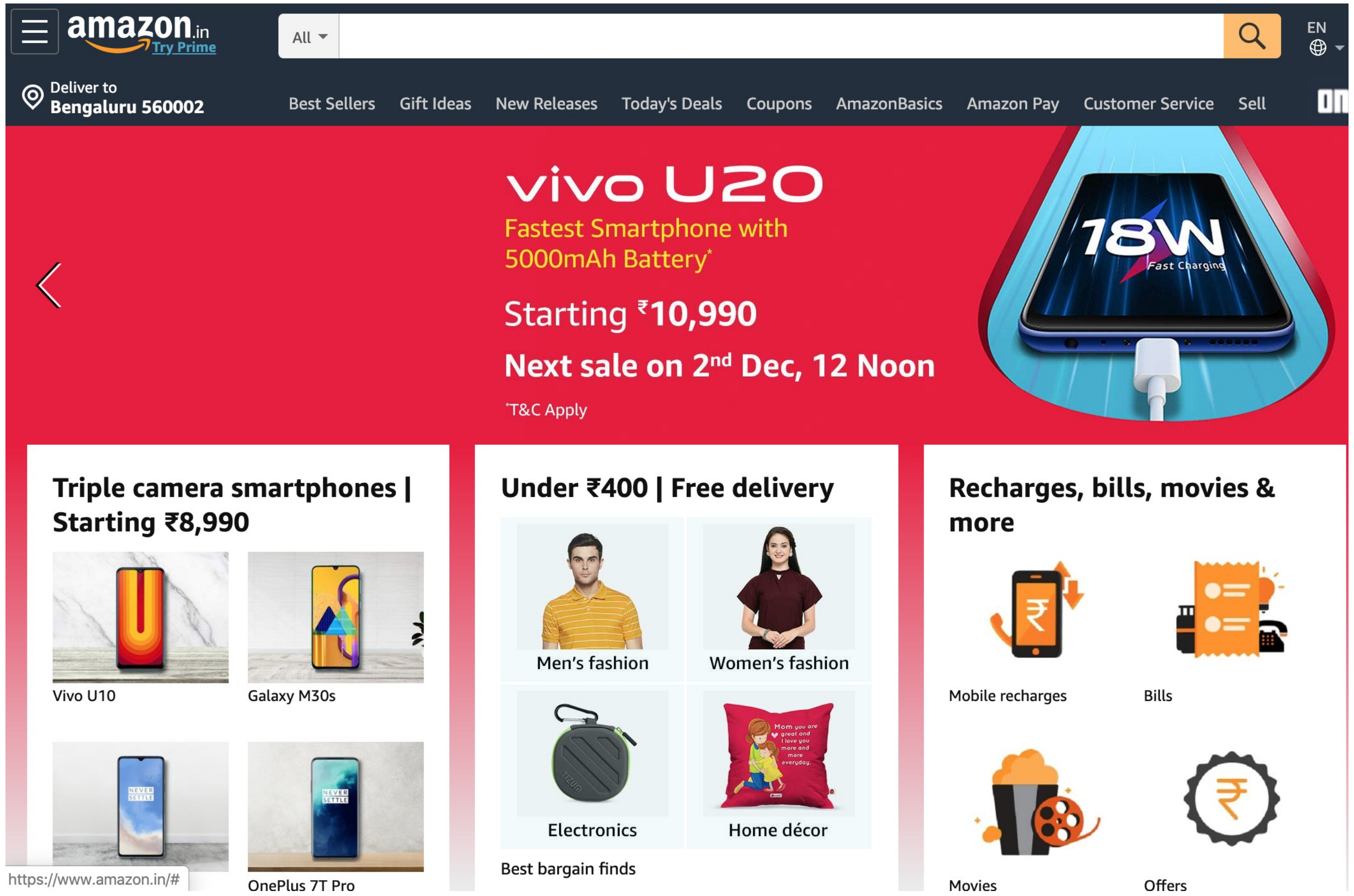
Fault Detection – Monitoring

- Health Checks
- Heart Beats

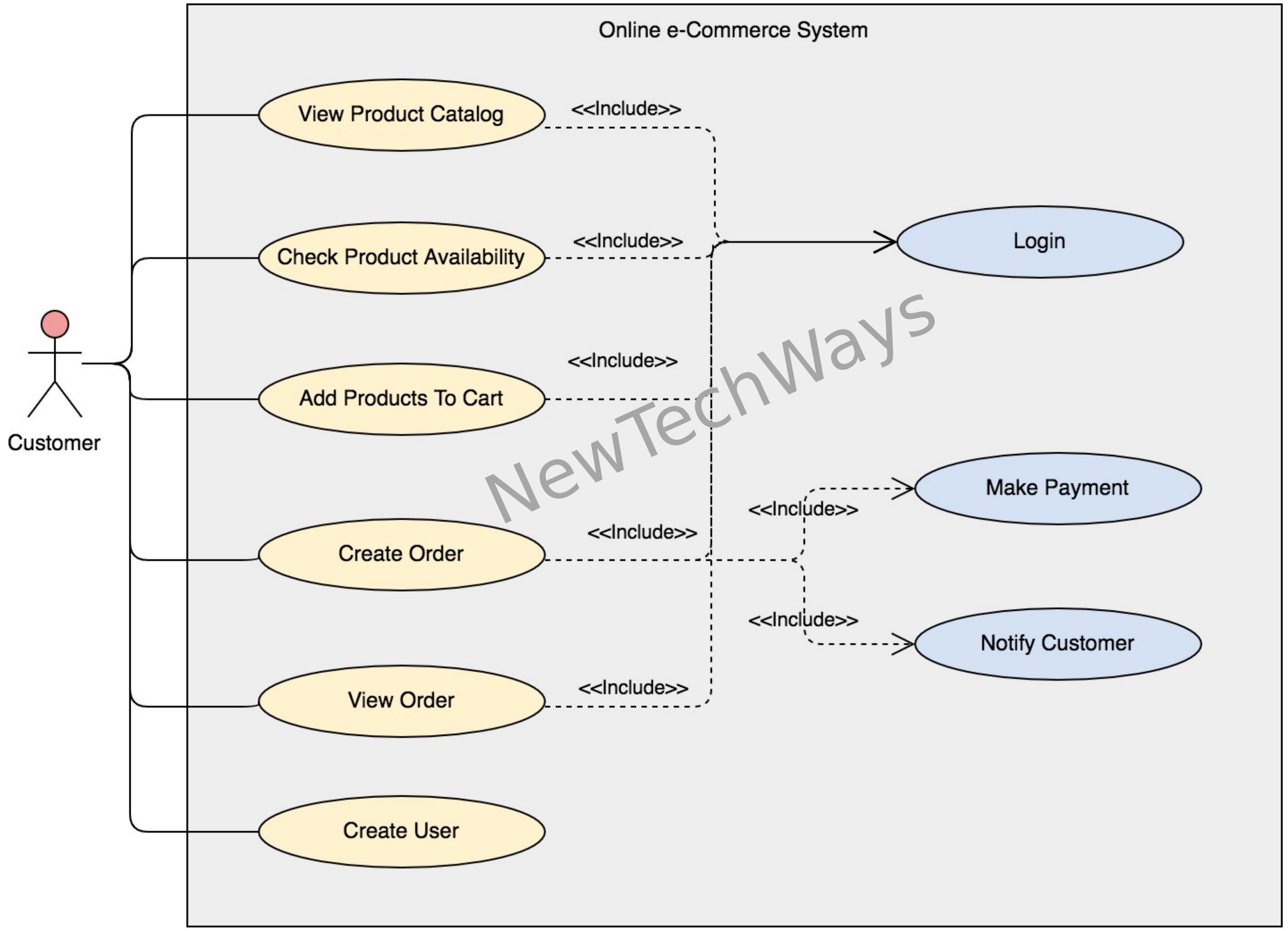


Application Architecture

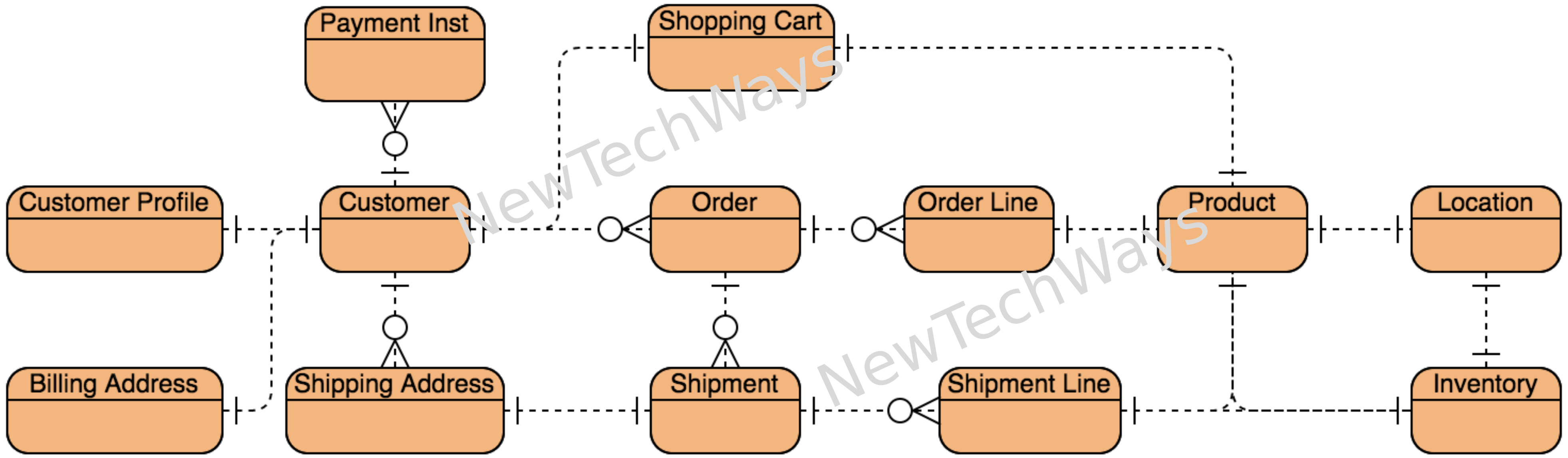
Assume ‘Architecting’ a big e-commerce system



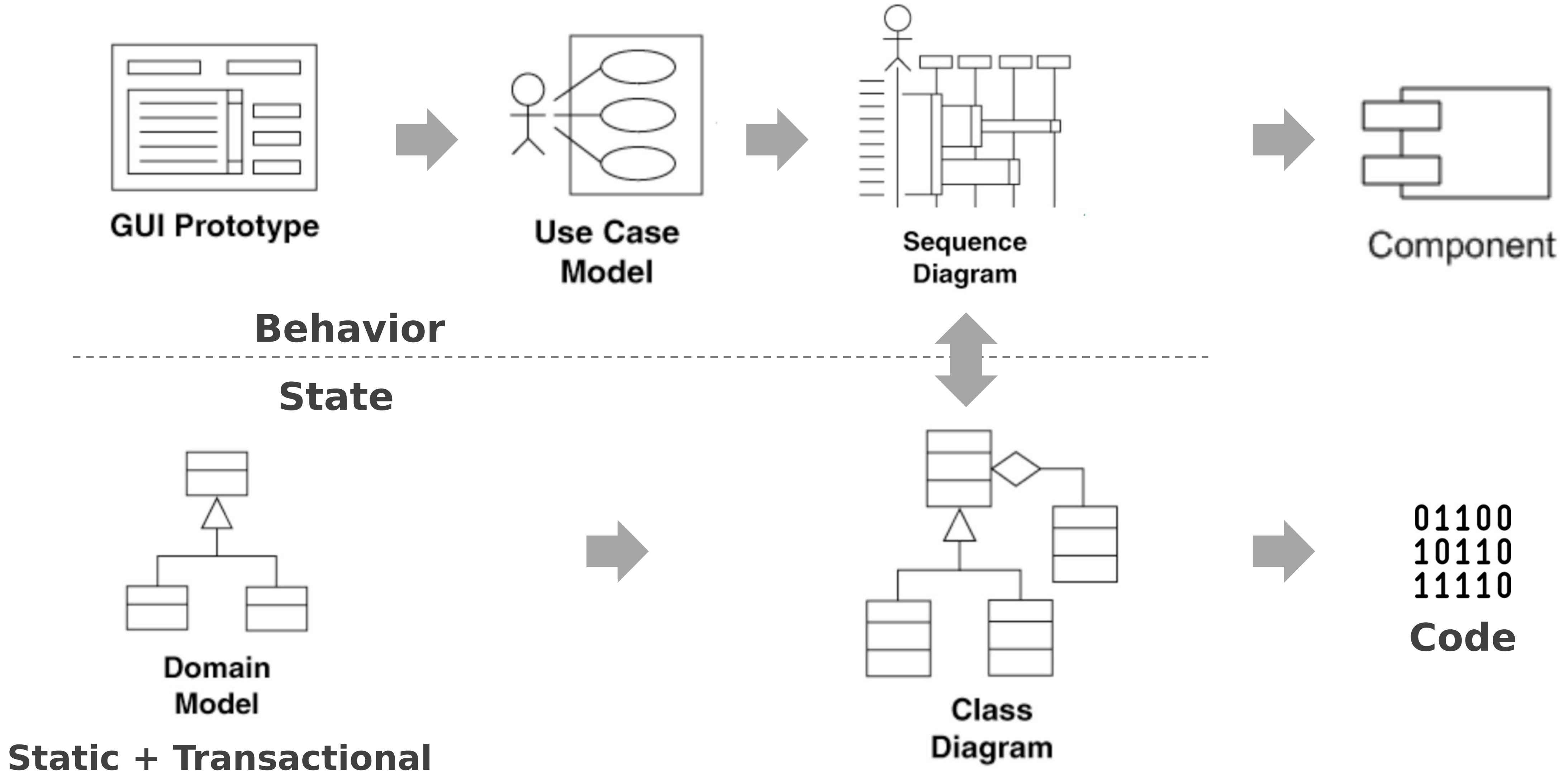
Use Case Model



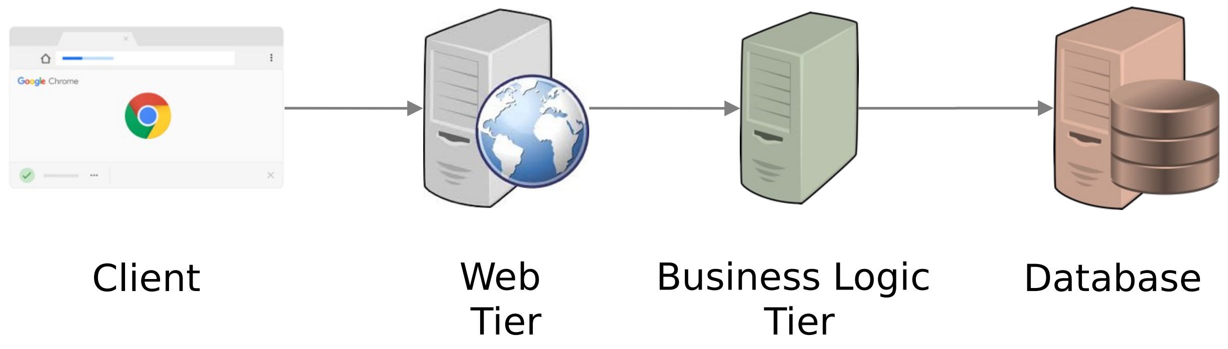
Domain Model

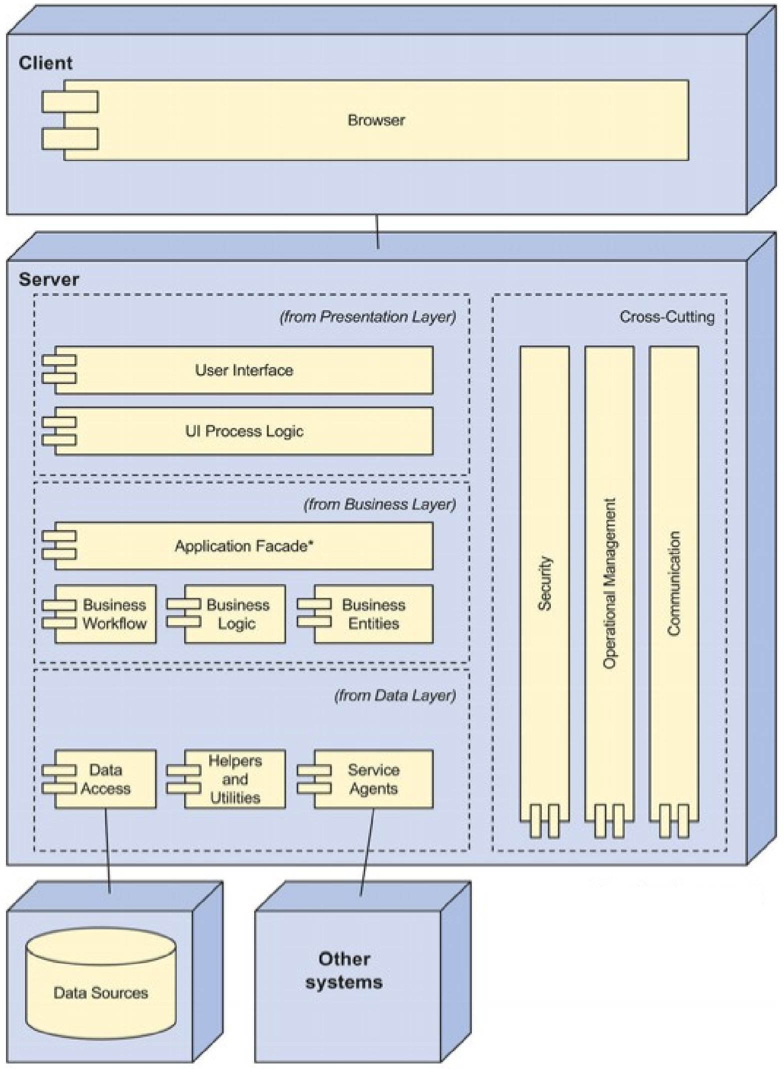


Low Level Design



Component Model





What About?

High Availability

Response Latency

Global Customers

System & Data Security

File storage for Catalog Images

Mobile Support

Unstructured Data Storage & Analytics

Cloud Deployment

Going Beyond

Application Architecture To System Architecture

Application Architecture Vs System Architecture

System level challenges
surface up in large scale systems

Scale, Reliability, Security, Deployment are
biggest concerns for a large-scale system

Latency Requirements

- 10 years ago, Amazon found that every 100ms of latency cost them 1% in sales
- Now Akamai study shows that every 100-millisecond delay in website load time can hurt sales by 6%
- Google found an extra .5 seconds in search page generation time dropped traffic by 20%

Scalability Requirements

10 million requests/day

1K to 100K simultaneous users

100 million products

Transaction data for last 5 years

Petabytes of log data

- For 100 M products with average product description/image size of 1 MB
  - Data Storage = 100M x 1MB = 100 TB
- Logs generated and archived
  - Data Storage => in Petabytes

Availability & Reliability Requirements

Unavailability results in

- Business loss
- Reputation loss

99.95% Availability

- Maximum cumulative disruption of 4 hours 22 minutes in a year

99.999999999% Durability for storage systems

- Data once stored is practically never lost

Disaster Recovery

- Operations to continue even if a region goes down due to a natural calamity

Security Requirements

Infrastructure protection

- Network access
- System access
- Service access

Data Protection

- Data sensitivity classification
- Protect data at rest
- Protect data on wire
- Data backup & replication

Identity & Access Management

- Authentication, Authorization

Designing System Architecture

Scalability Principle

Monolith is an anti-pattern for Scalability

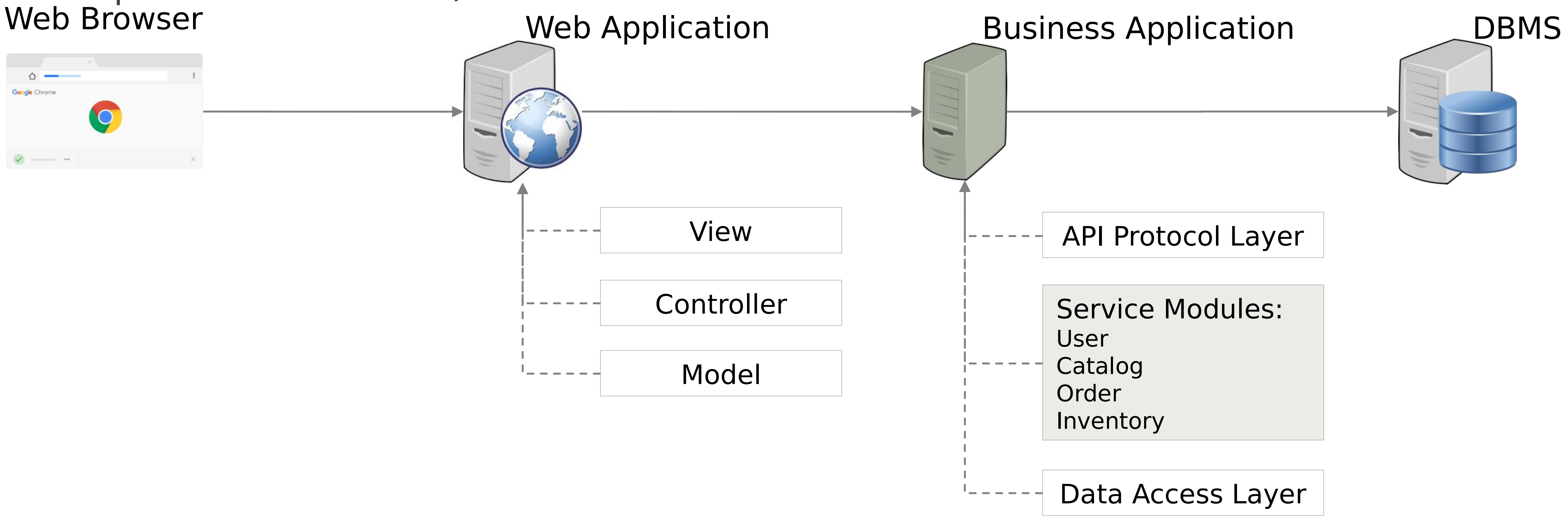
Scalability goes up with

- Decentralization
  - More specialized workers – Services
  - More workers – Instances, Processors, Threads
- Independence
  - Multiple workers are as good as a single worker if they can’t work independently
  - They must work concurrently to maximum extent

Modularity

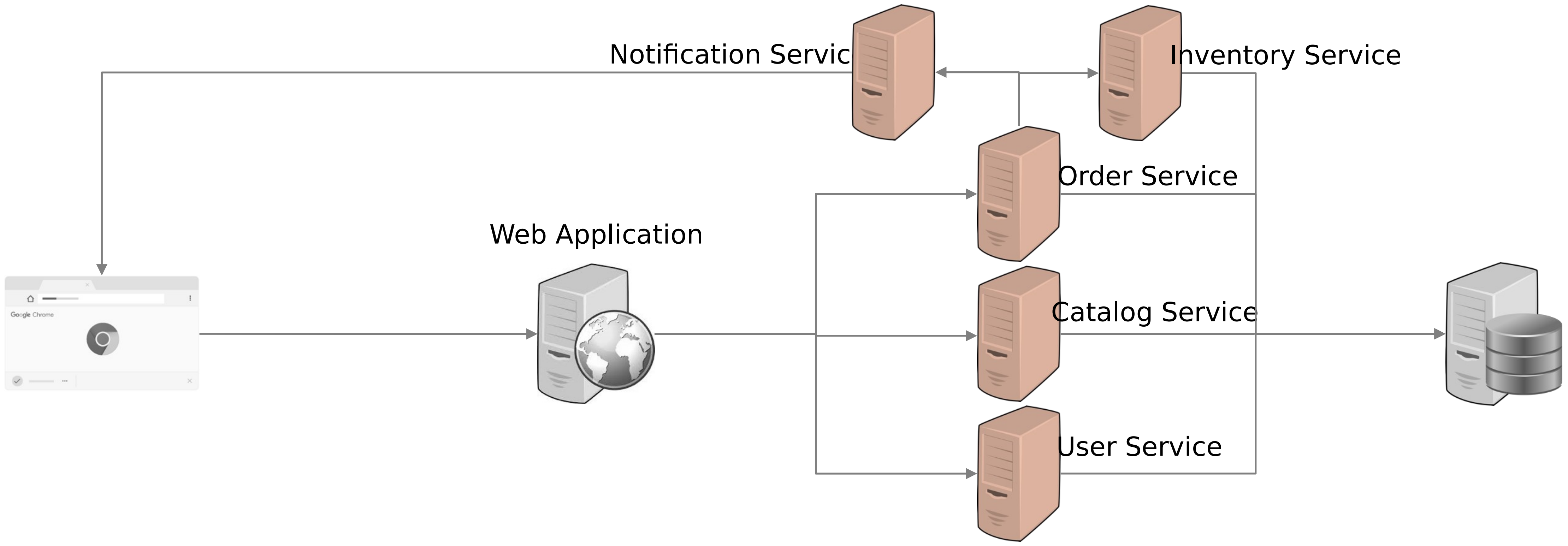
Scalable architecture starts with modularity

- Provides the foundation for breaking a system function/service into more specialized functions/services



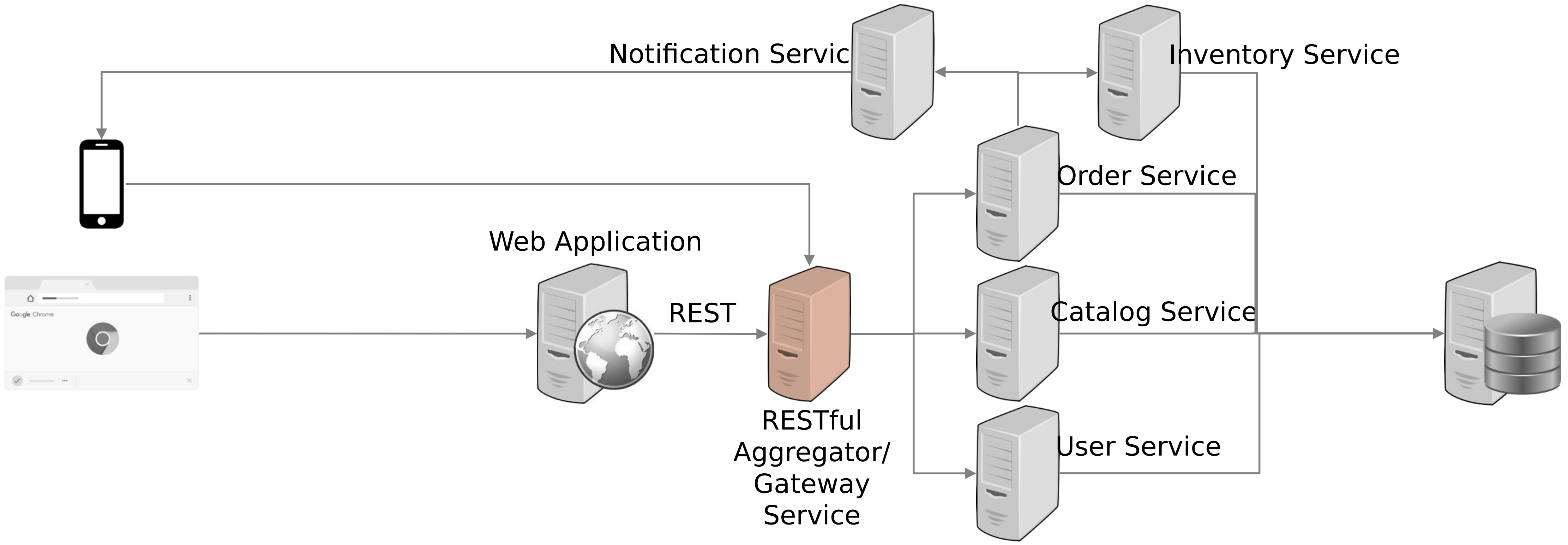
Specialized Services – WebServices

Services can be scaled differently e.g., Number of instances



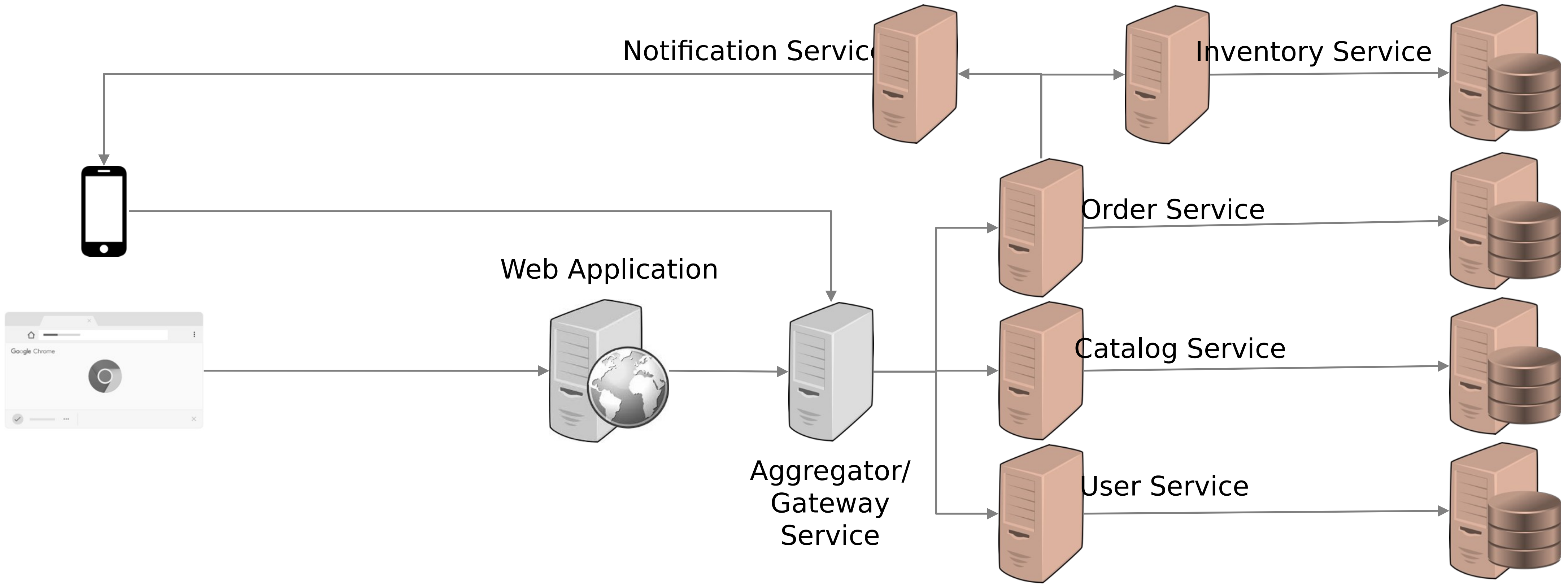
Aggregator Service & RESTful API – Mobile Support

REST for external interface interoperability & Mobile Support



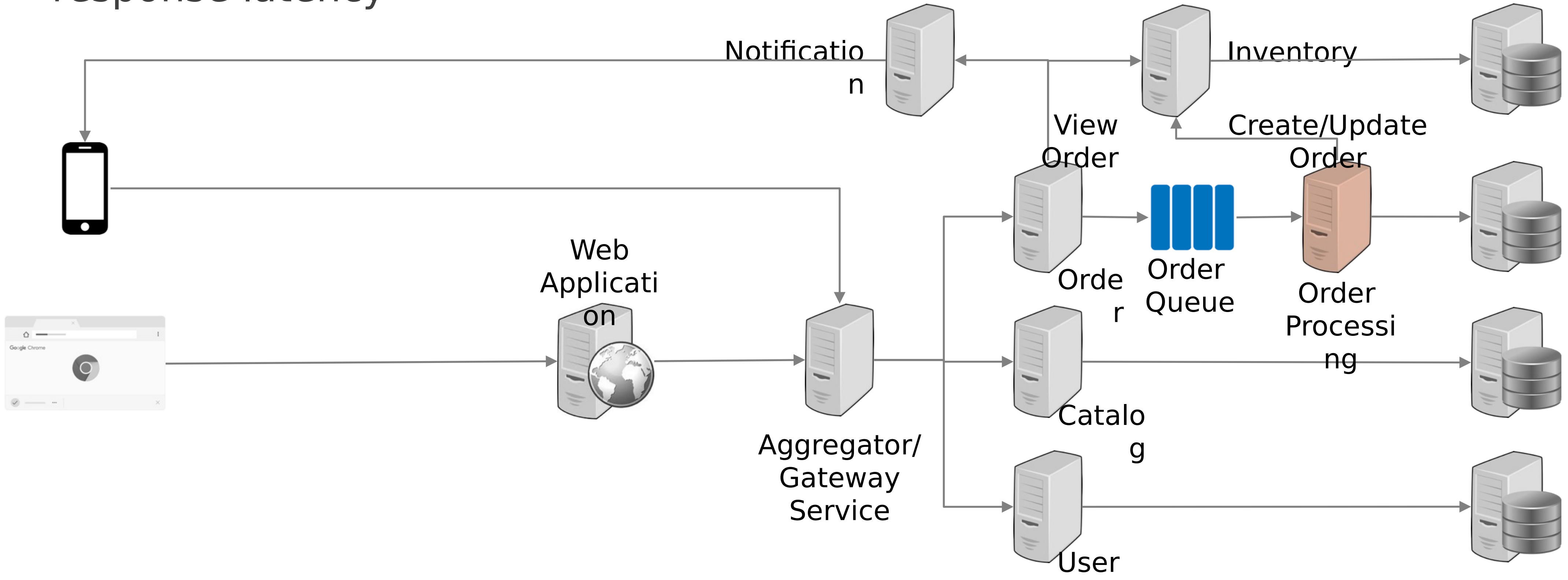
Independent Services – MicroServices

Micro-Services can be scaled differently and deployed independently



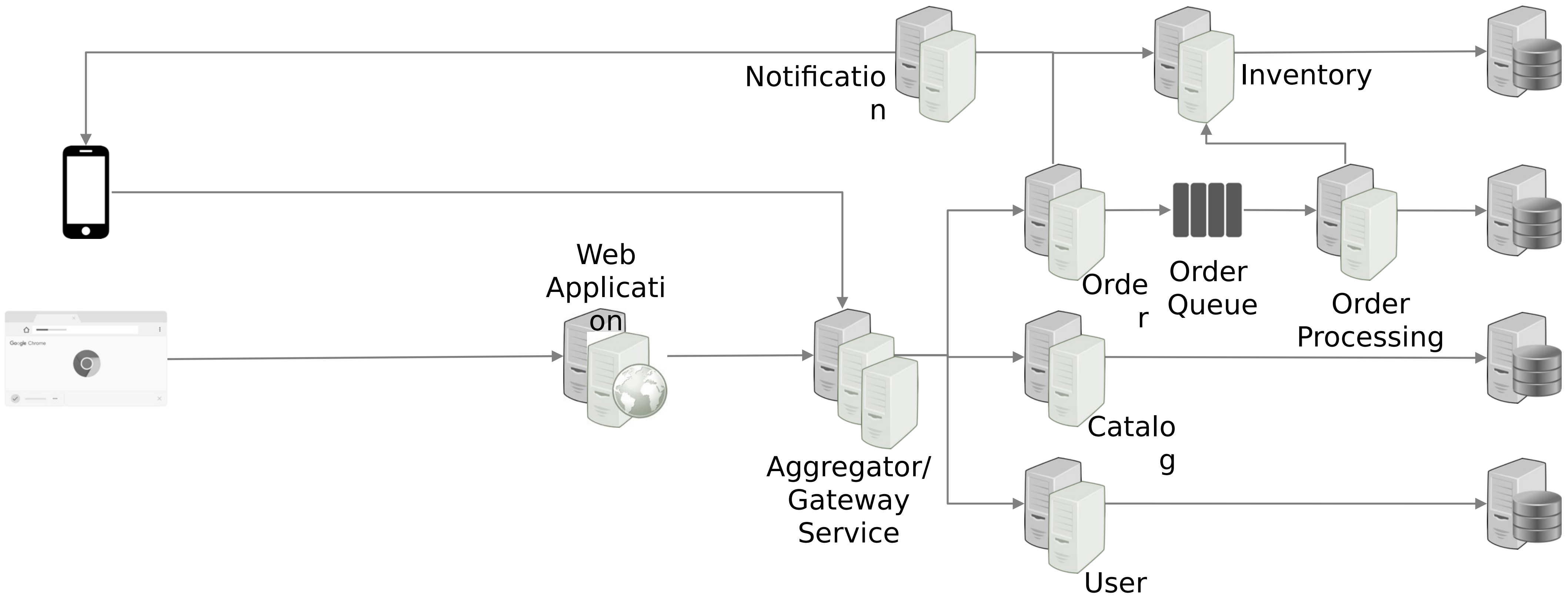
Asynchronous Services

Updates can be done asynchronously to buffer peak loads and to reduce response latency



Stateless Replication

Replication provides more computation power



Reliability Principle

Reliability
- Normal functioning even in the presence of faults

Availability
- Always available even in the presence of faults

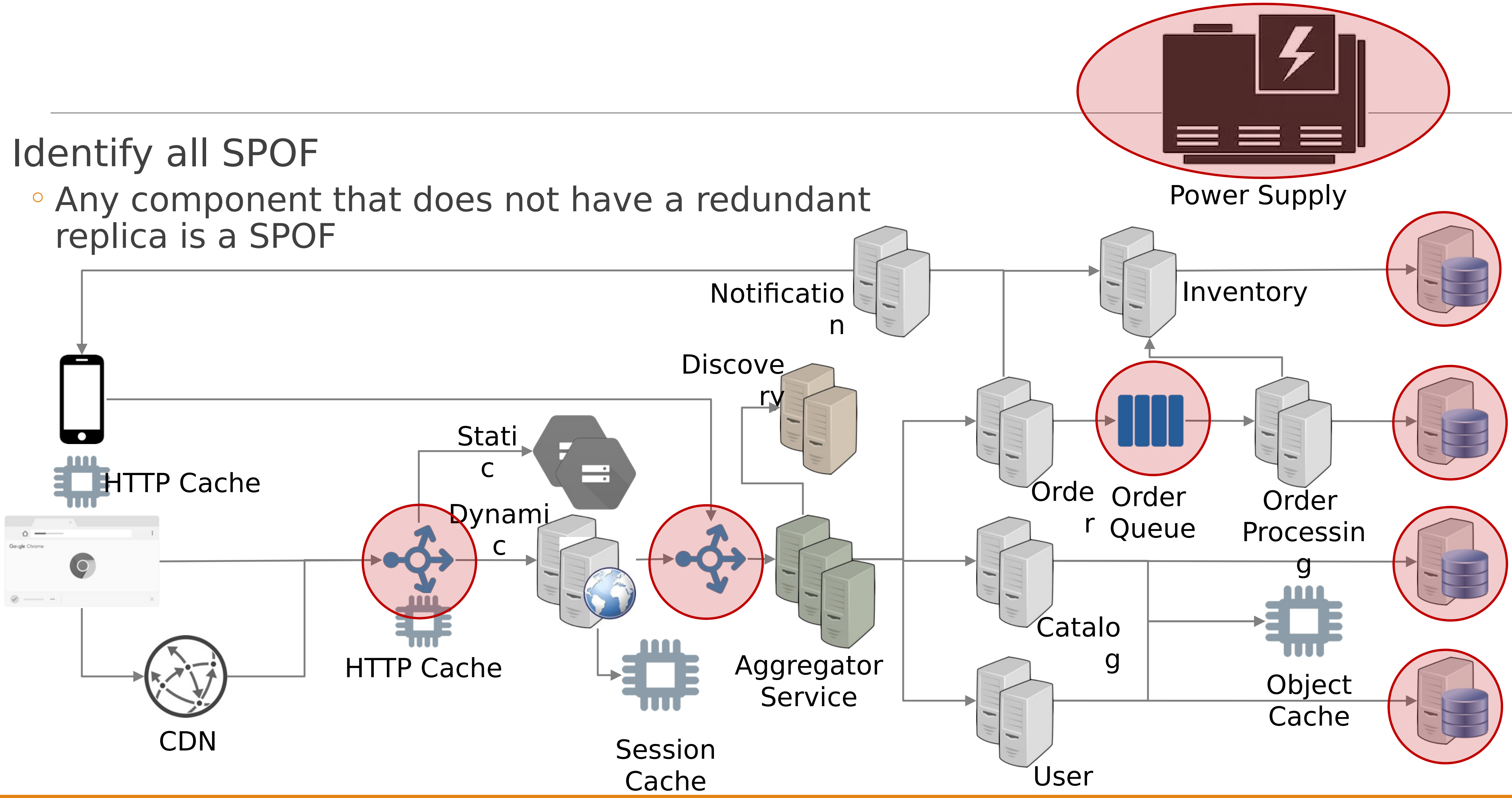
Reliability and Availability are achieved mainly through Fault Tolerance

Fault Tolerance requires
- Provisioning Redundancy
- Fault Detection Mechanisms
  - Health-checks, heart-beats

Single Point Of Failures

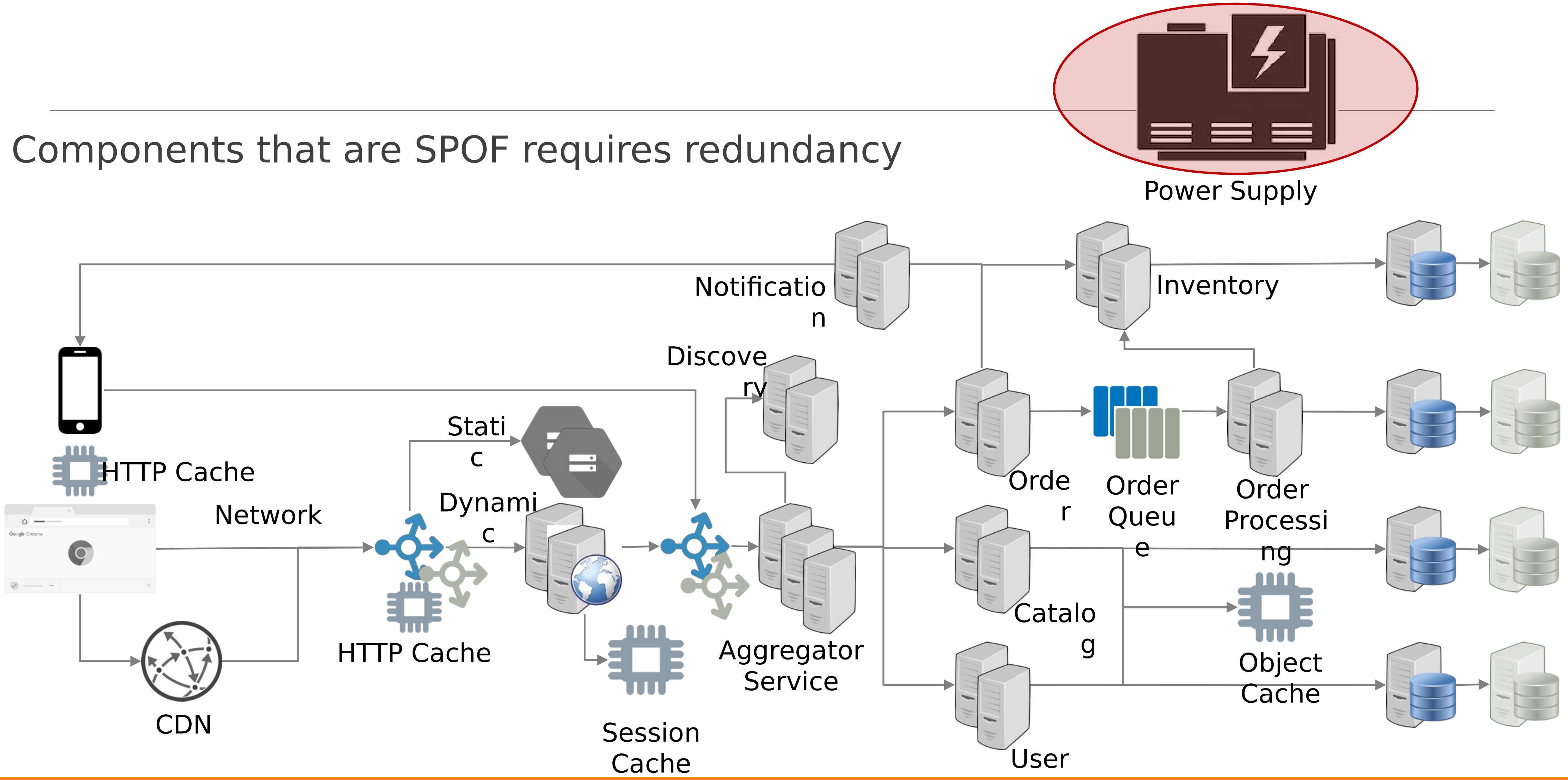
Identify all SPOF

- Any component that does not have a redundant replica is a SPOF



Redundancy

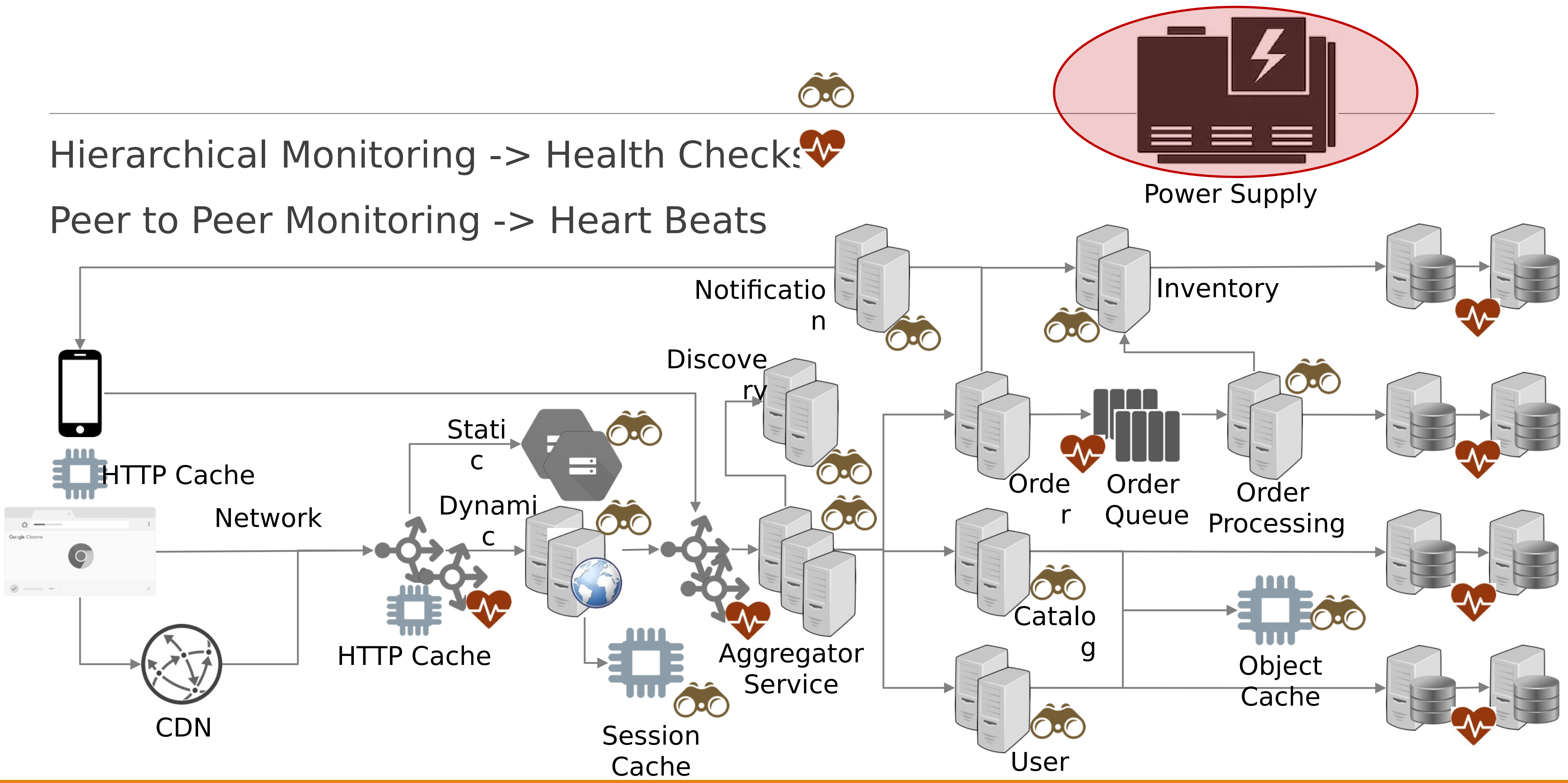
Components that are SPOF requires redundancy



Monitoring For Fault Detection

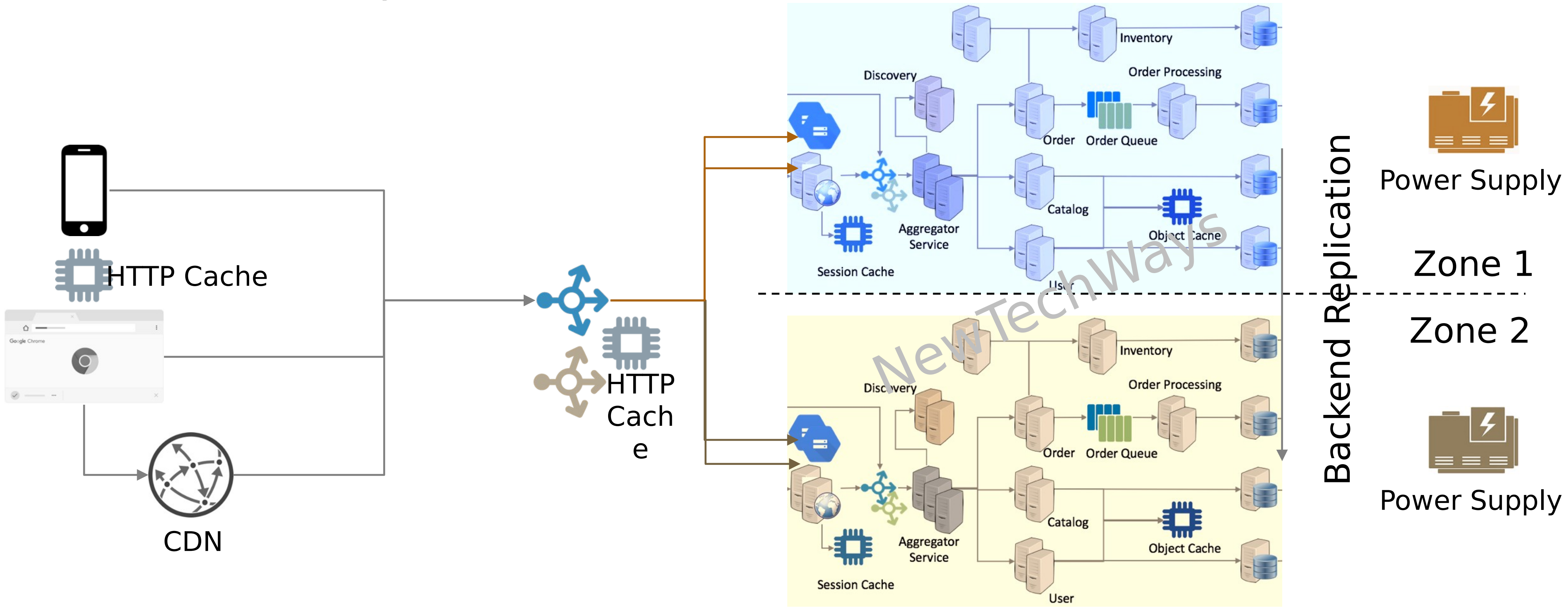
Hierarchical Monitoring -> Health Checks

Peer to Peer Monitoring -> Heart Beats



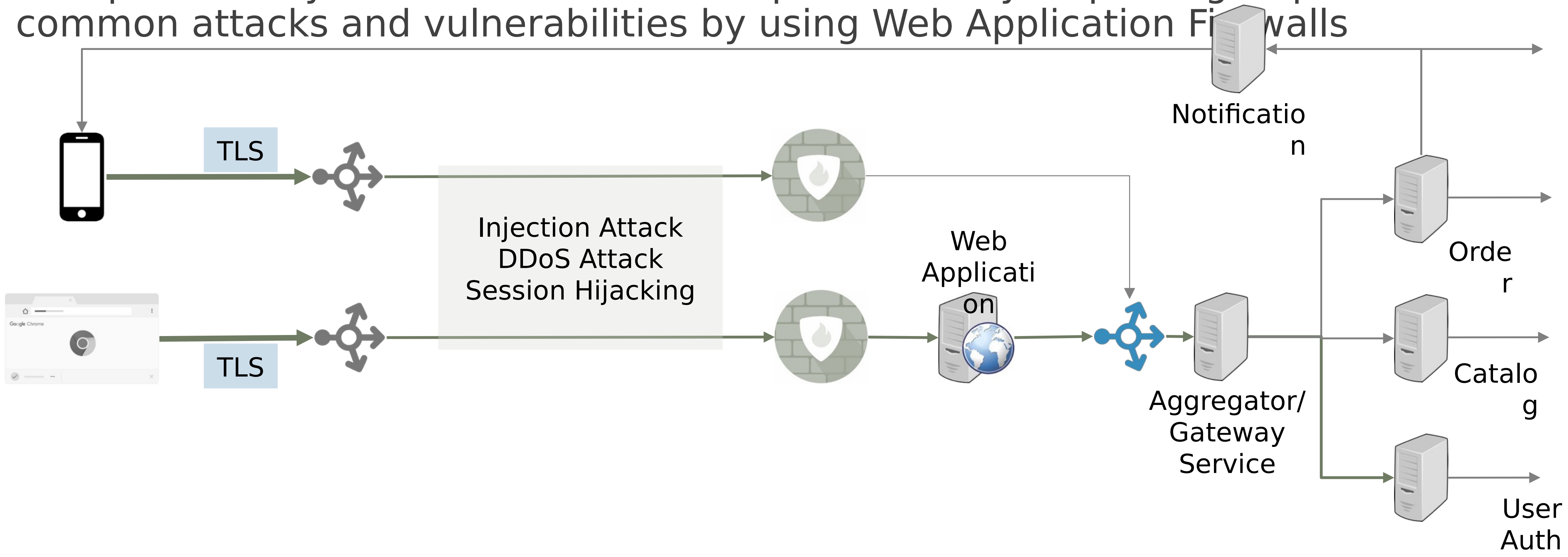
Zonal Redundancy

Zonal redundancy for fault isolation



Security-Web Application Firewalls

Multiple security related attacks can be prevented by inspecting requests for common attacks and vulnerabilities by using Web Application Firewalls



HAVE A GOOD DAY!