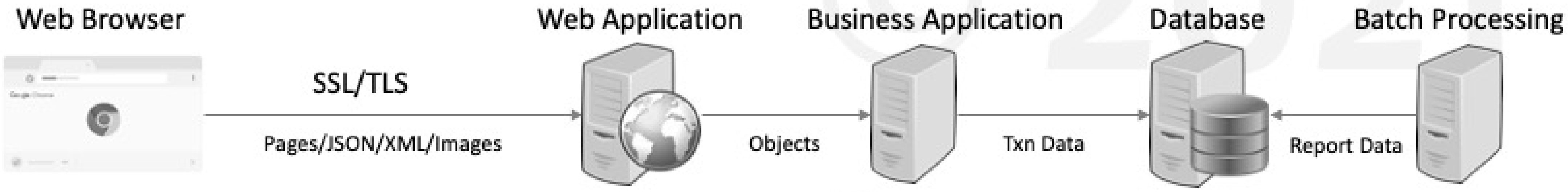
System Performance

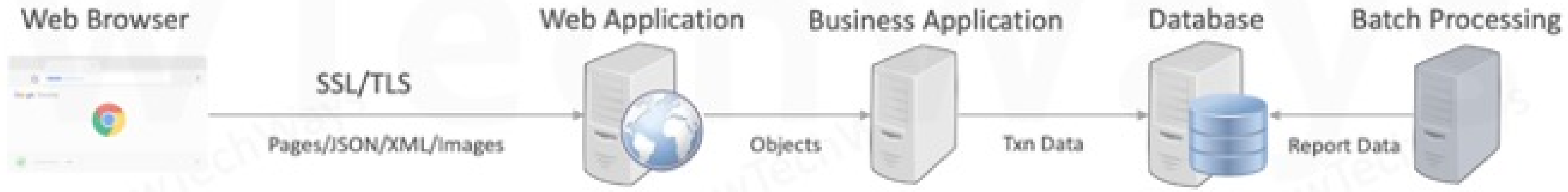
- Understanding Performance
  - Problems
  - Measurement
  - Principles
- Latency
  - CPU
  - Memory
  - Network
  - Disk
- Concurrency
  - Locking
    - Pessimistic
    - Optimistic
  - Coherence
- Caching
  - Static Data
  - Dynamic Data

Sample System



Performance

- Measure of how fast or responsive a system is under
  - A given workload
    - Backend data
    - Request volume
  - A given hardware
    - Kind
    - Capacity

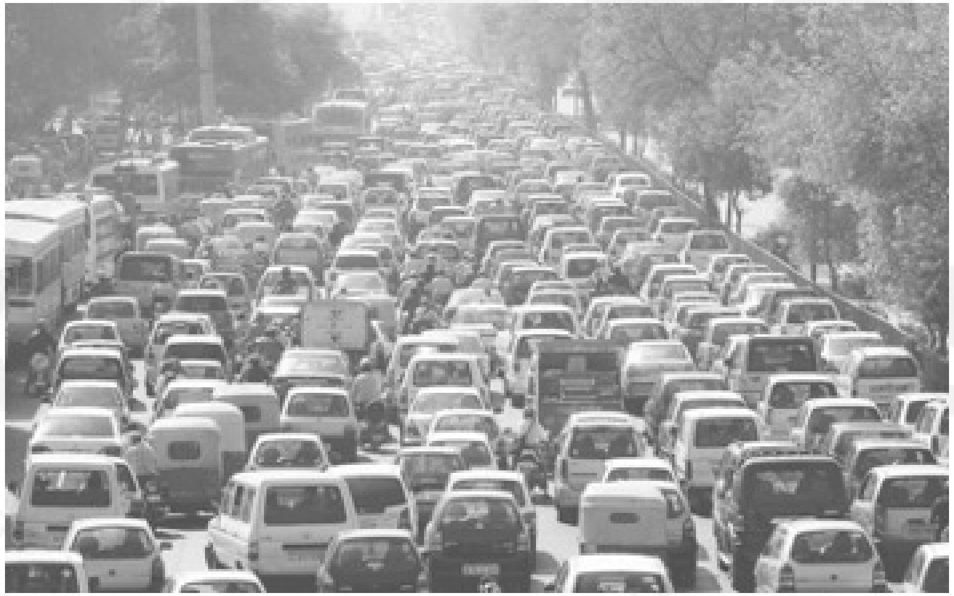


Performance Problems

How to spot a Performance Problem? How does it look like?

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

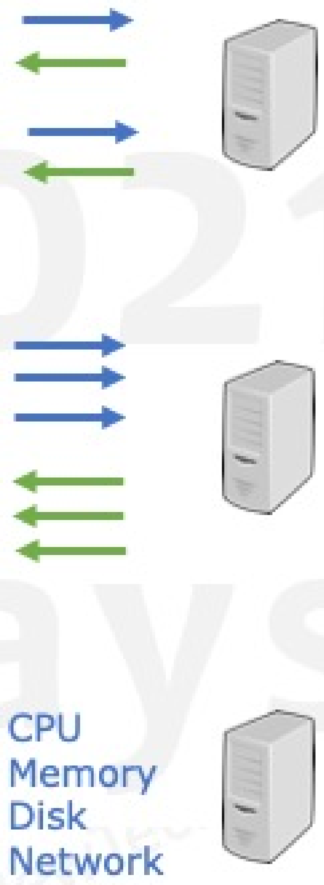
- *Network socket queue, DB IO queue, OS run queue etc.*



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

Performance Principles

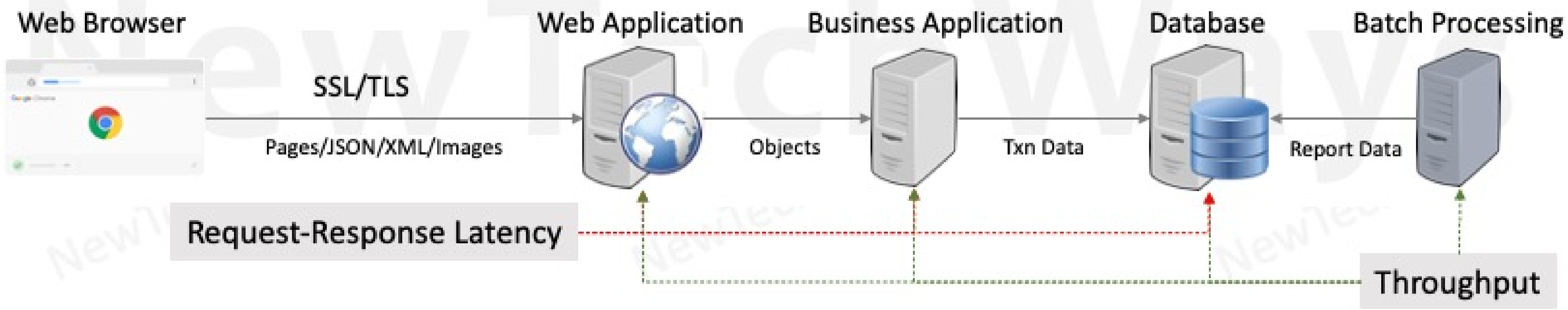
- Efficiency
  - Efficient Resource Utilization
    - IO – Memory, Network, Disk
    - CPU
  - Efficient Logic
    - Algorithms
    - DB Queries
  - Efficient Data Storage
    - Data Structures
    - DB Schema
  - Caching
- Concurrency
  - Hardware
  - Software
    - Queuing
    - Coherence
- Capacity



System Performance Objectives

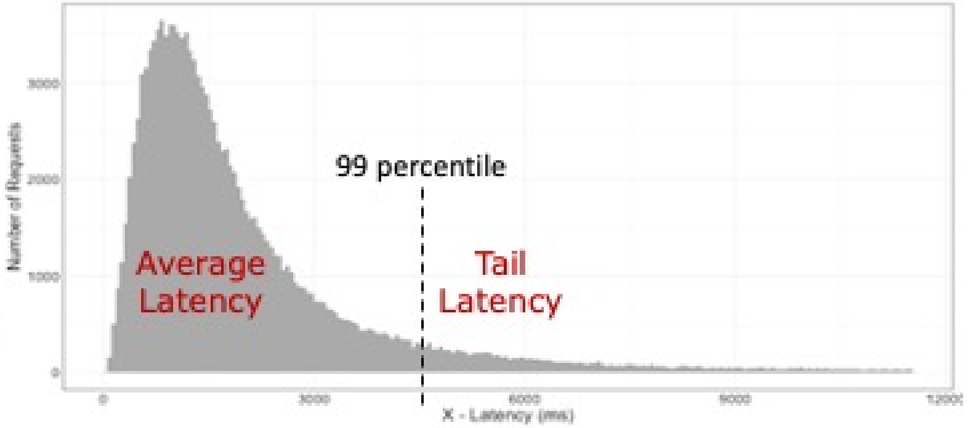
- Minimize Request-Response Latency
- Latency is Measured in Time Units
- Depends on
  - Wait/Idle Time
  - Processing Time

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



Performance Measurement Metrics

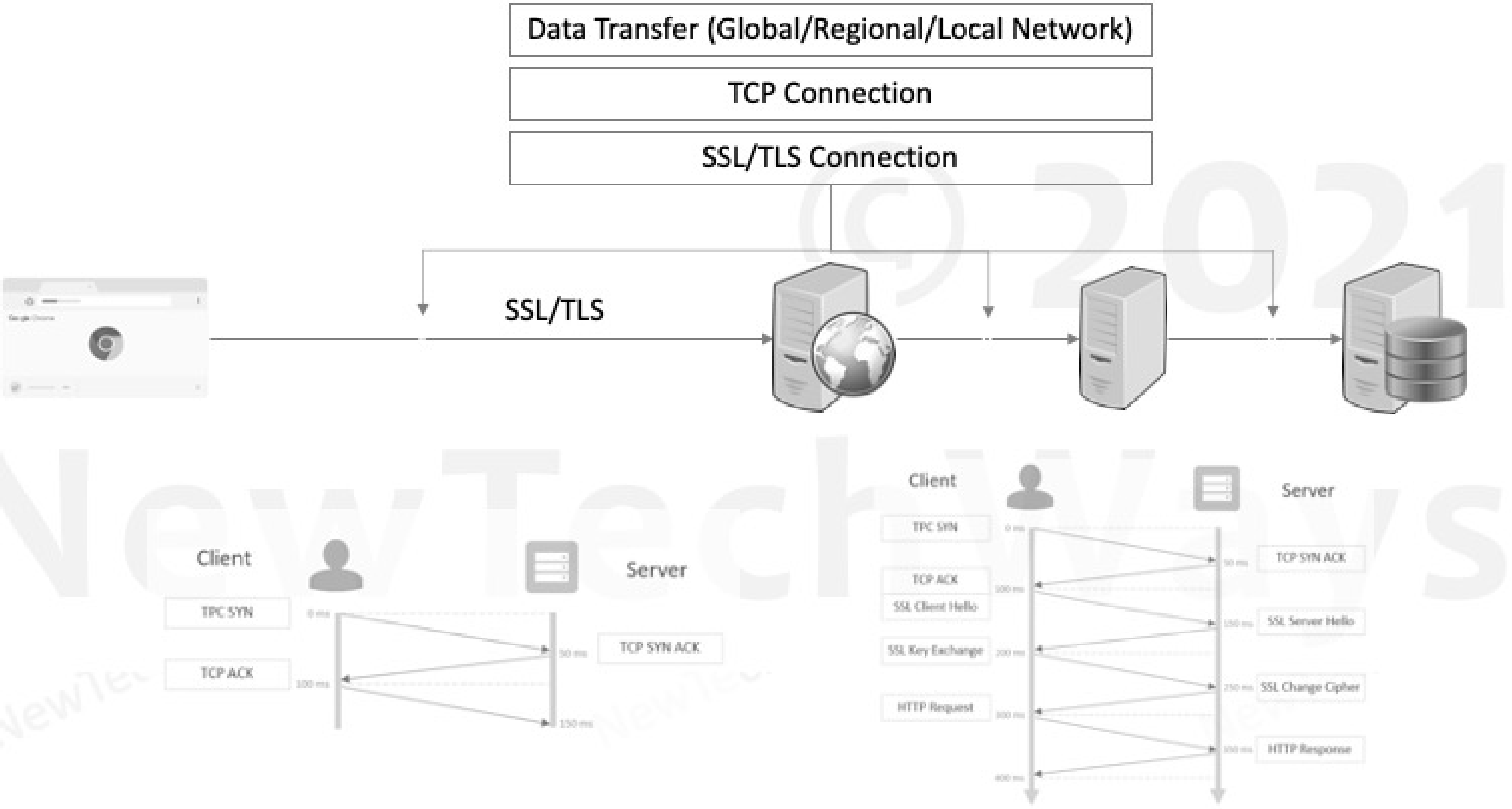
- Latency
  - Affects – User Experience
  - Desired – As low as possible
- Throughput
  - Affects – Number of users that can be supported
  - Desired – Greater than the request rate
- Errors
  - Affects – Functional Correctness
  - Desired – None
- Resource Saturation
  - Affects - Hardware capacity 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

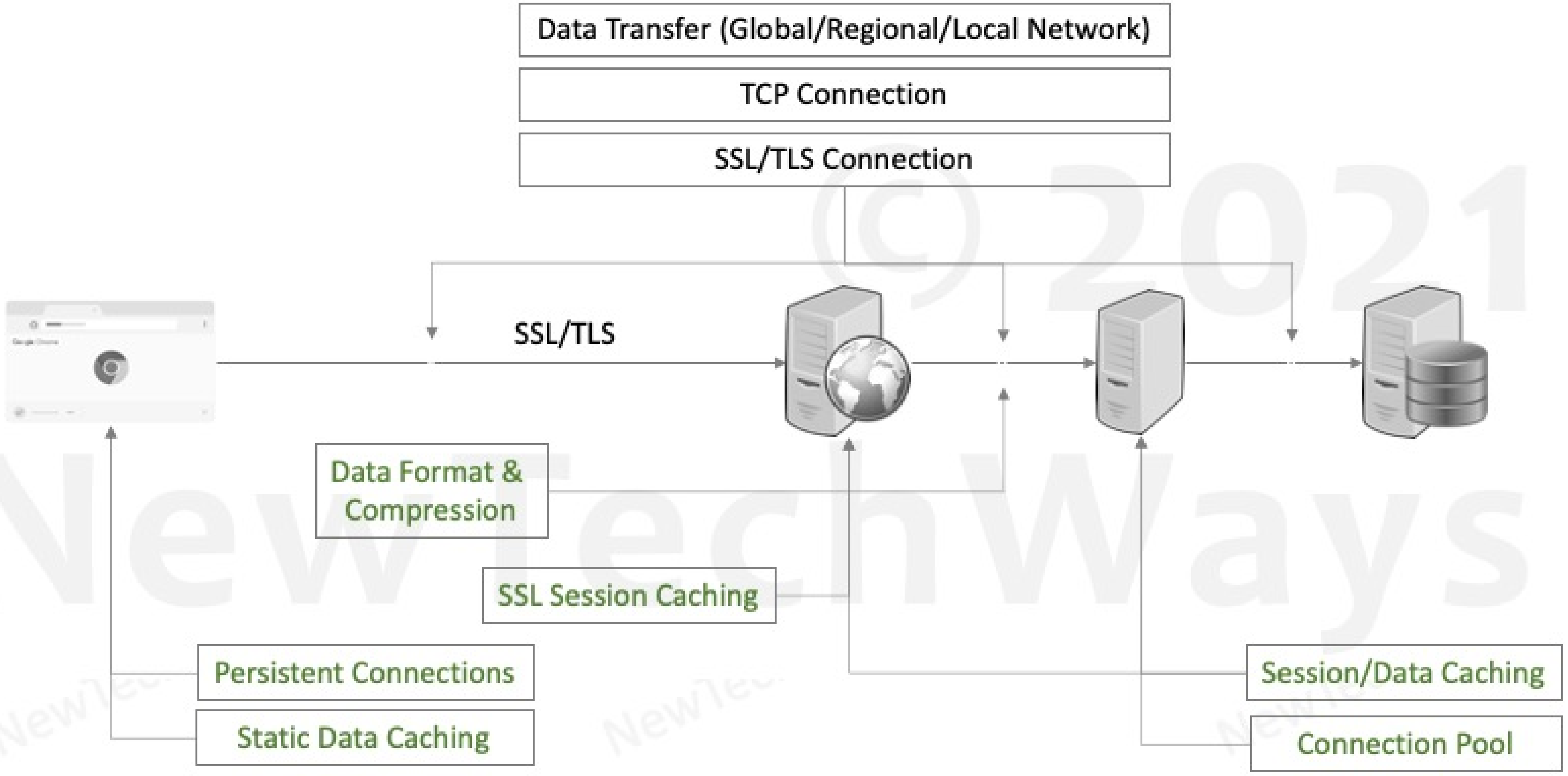
Serial Request Latency

Network Latency



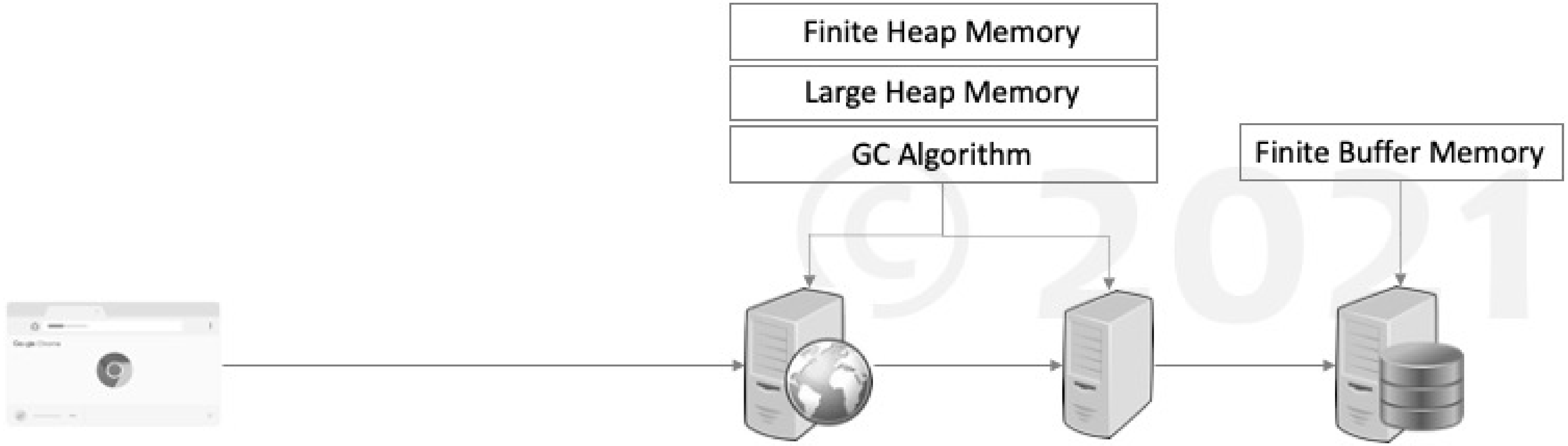
NewTechWays

Network Latency – Approaches

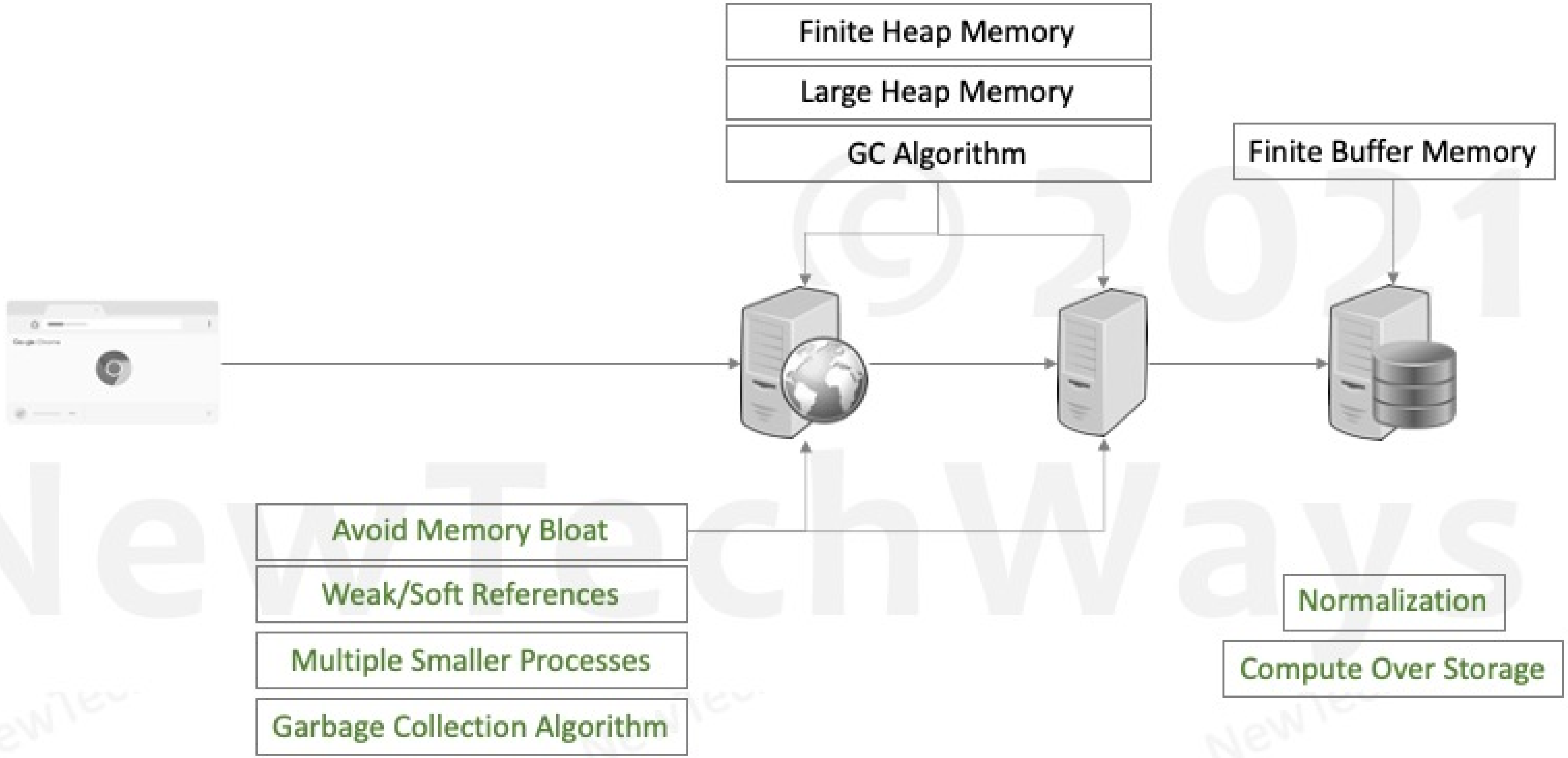


NewTechWays

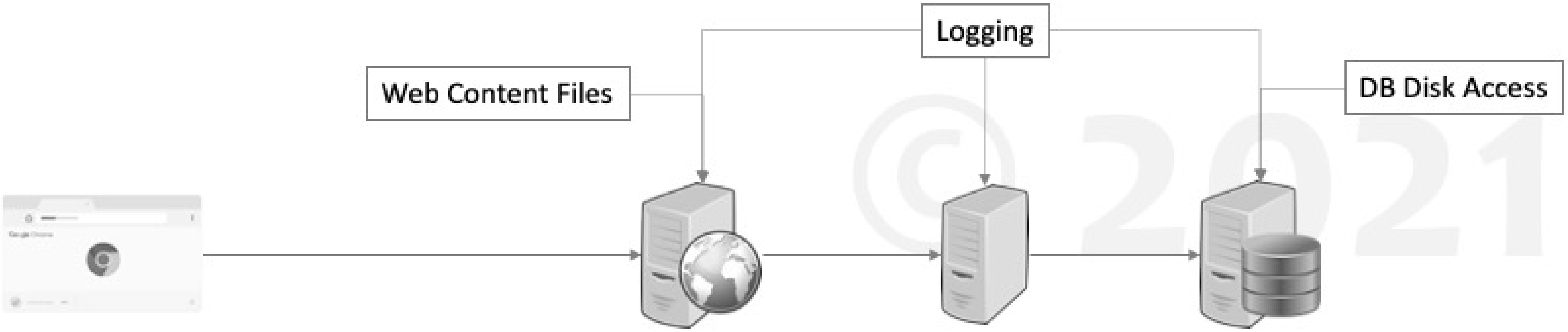
Memory Latency



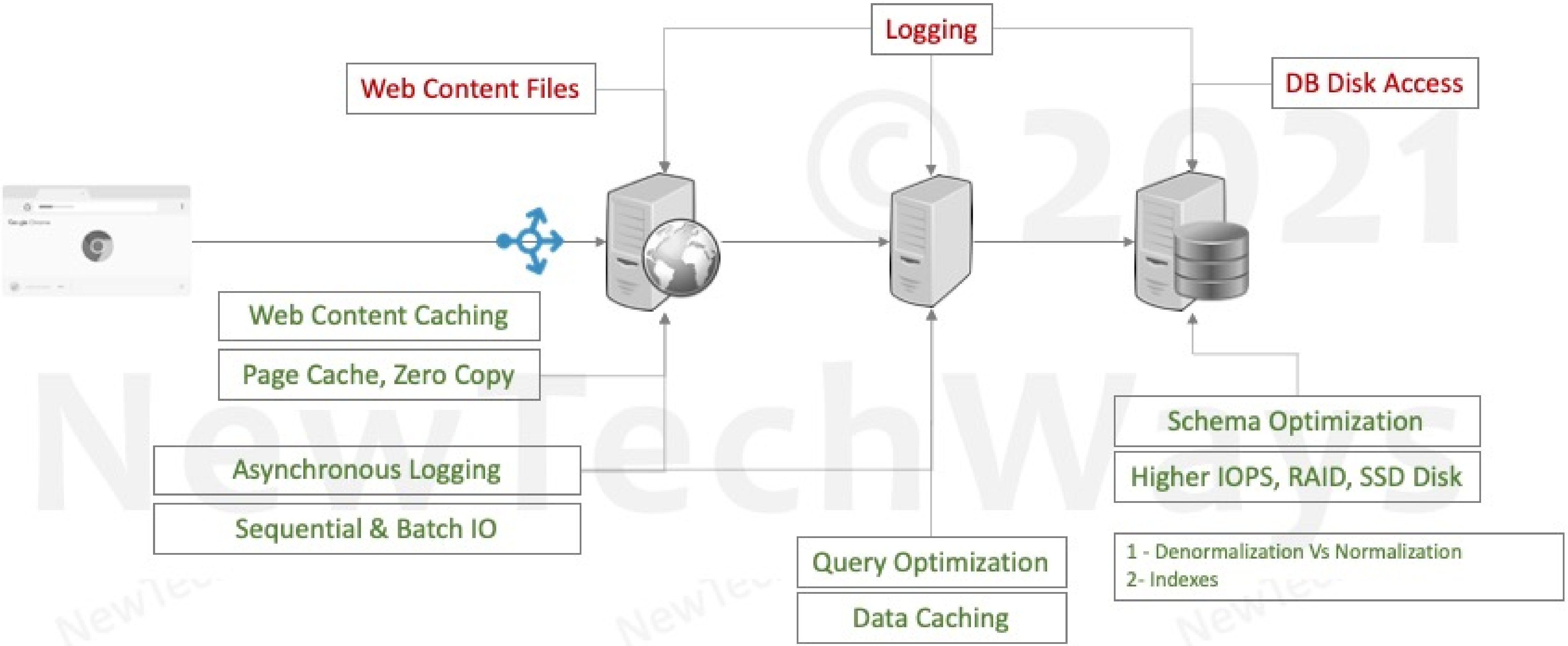
Memory Latency – Approaches



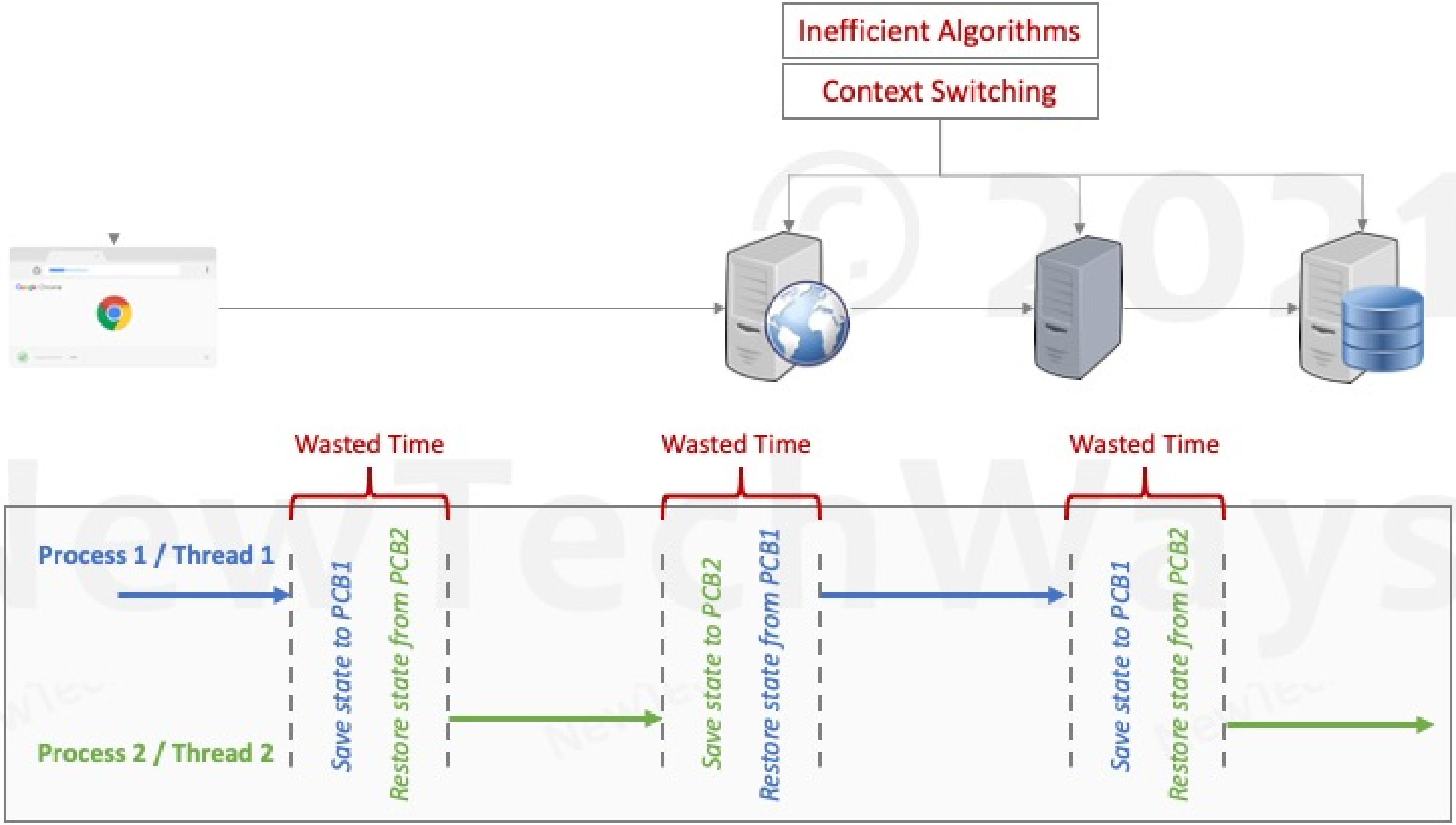
Disk Latency



Disk Latency – Approaches

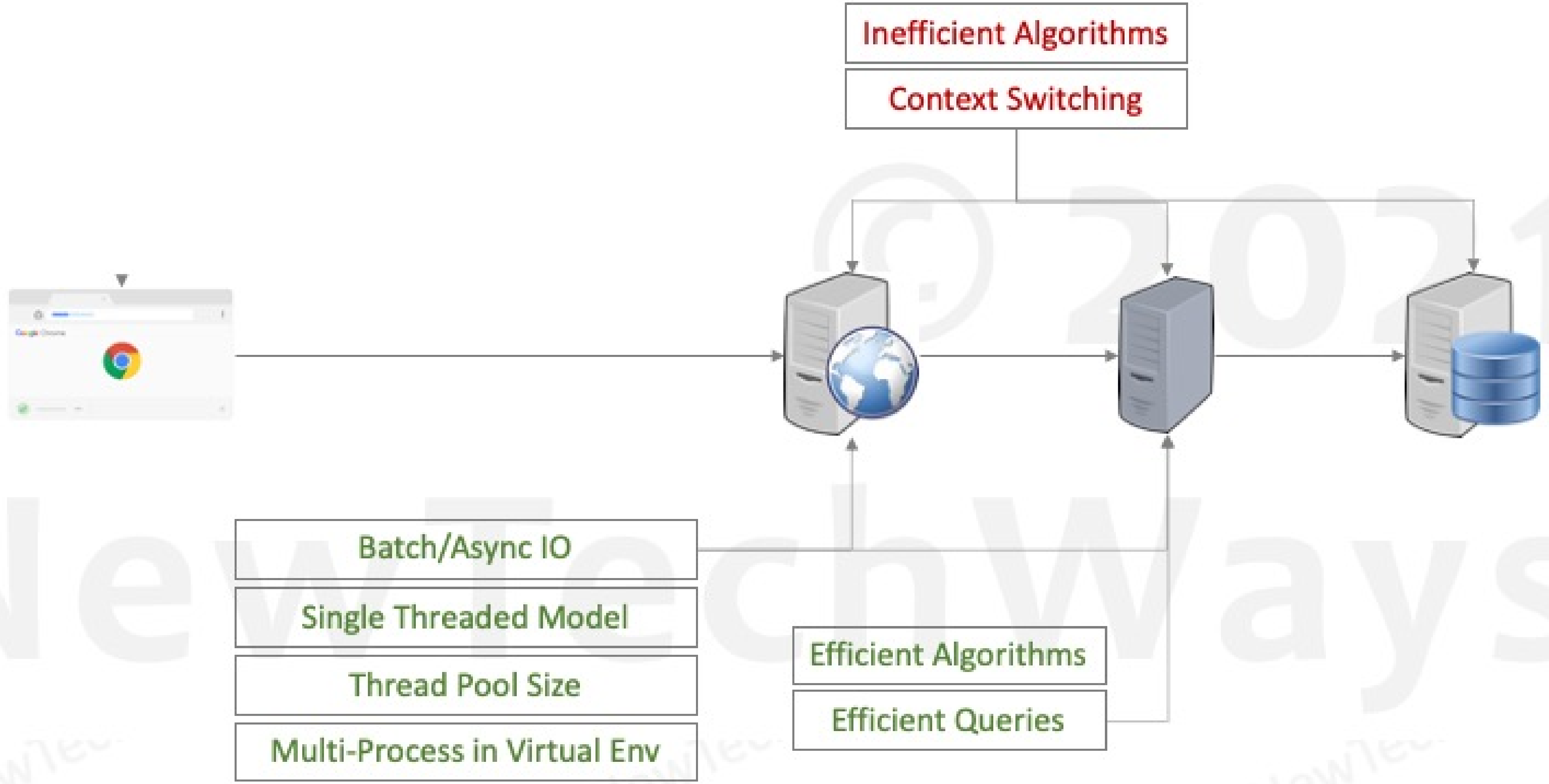


CPU Latency



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



NewTechWays

Latency Costs

```
Latency Comparison Numbers (~2012)
----------------------------------
L1 cache reference                           0.5 ns
Branch mispredict                            5   ns
L2 cache reference                           7   ns                      14x L1 cache
Mutex lock/unlock                           25   ns
Main memory reference                      100   ns                      20x L2 cache, 200x L1 cache
Compress 1K bytes with Zippy             3,000   ns        3 us
Send 1K bytes over 1 Gbps network       10,000   ns       10 us
Read 4K randomly from SSD*             150,000   ns      150 us          ~1GB/sec SSD
Read 1 MB sequentially from memory     250,000   ns      250 us
Round trip within same datacenter      500,000   ns      500 us
Read 1 MB sequentially from SSD*     1,000,000   ns    1,000 us    1 ms  ~1GB/sec SSD, 4X memory
Disk seek                           10,000,000   ns   10,000 us   10 ms  20x datacenter roundtrip
Read 1 MB sequentially from disk    20,000,000   ns   20,000 us   20 ms  80x memory, 20X SSD
Send packet CA->Netherlands->CA    150,000,000   ns  150,000 us  150 ms
```

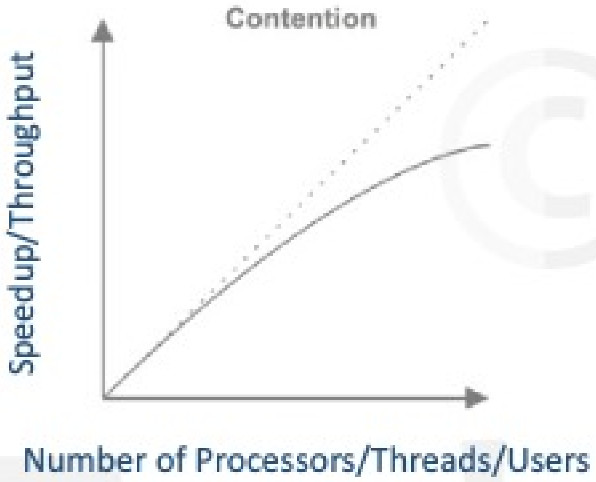
https://gist.github.com/jboner/2841832

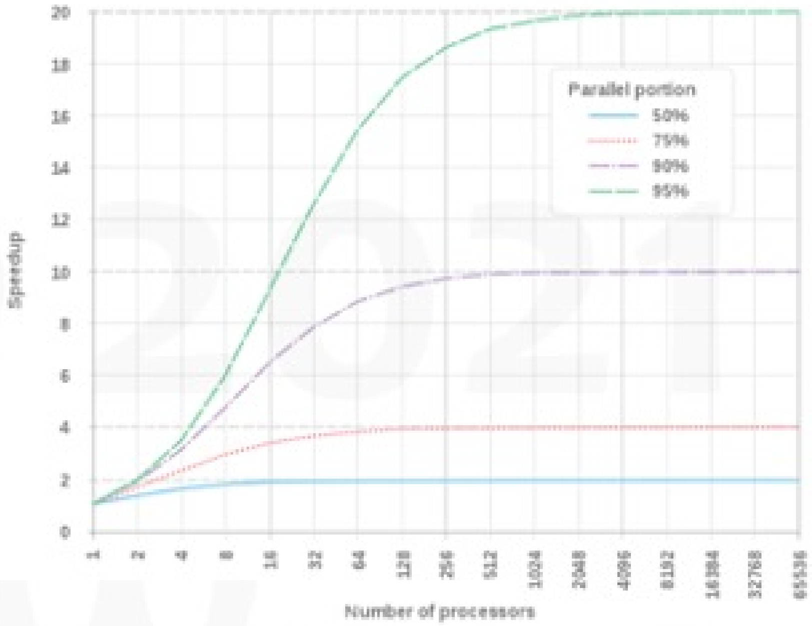
Parallel Request Concurrency

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





NewTechWays

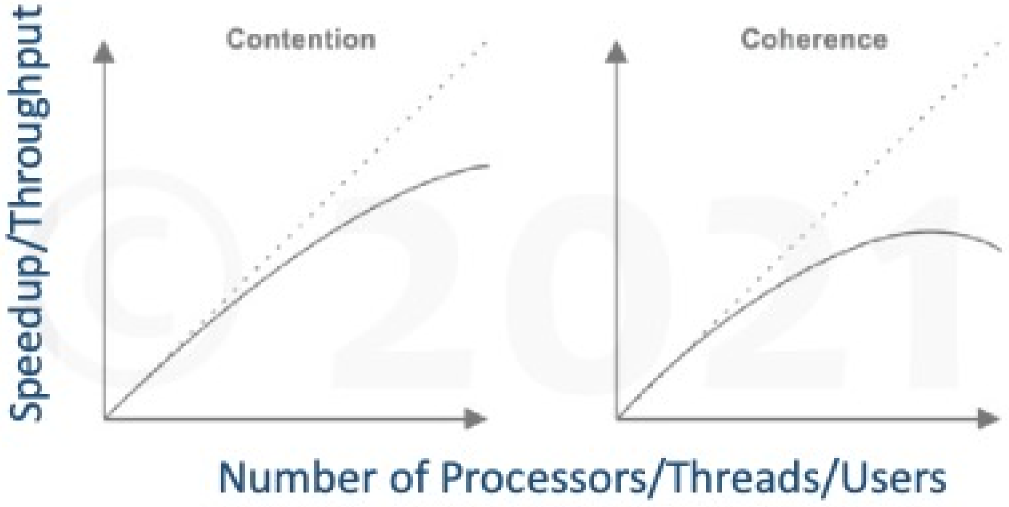
Concurrent Processing

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

Queueing

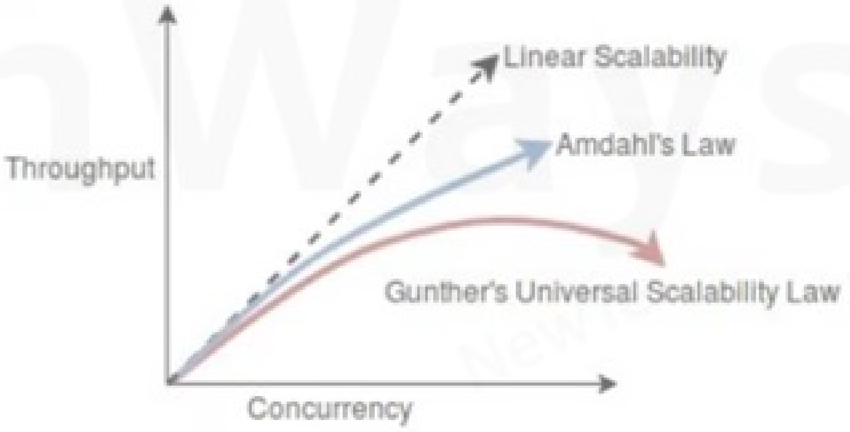
- Universal Scalability Law
  - $C(N) = \frac{N}{[1+\alpha(N-1)+\beta N(N-1)]}$

Queueing + Coherence



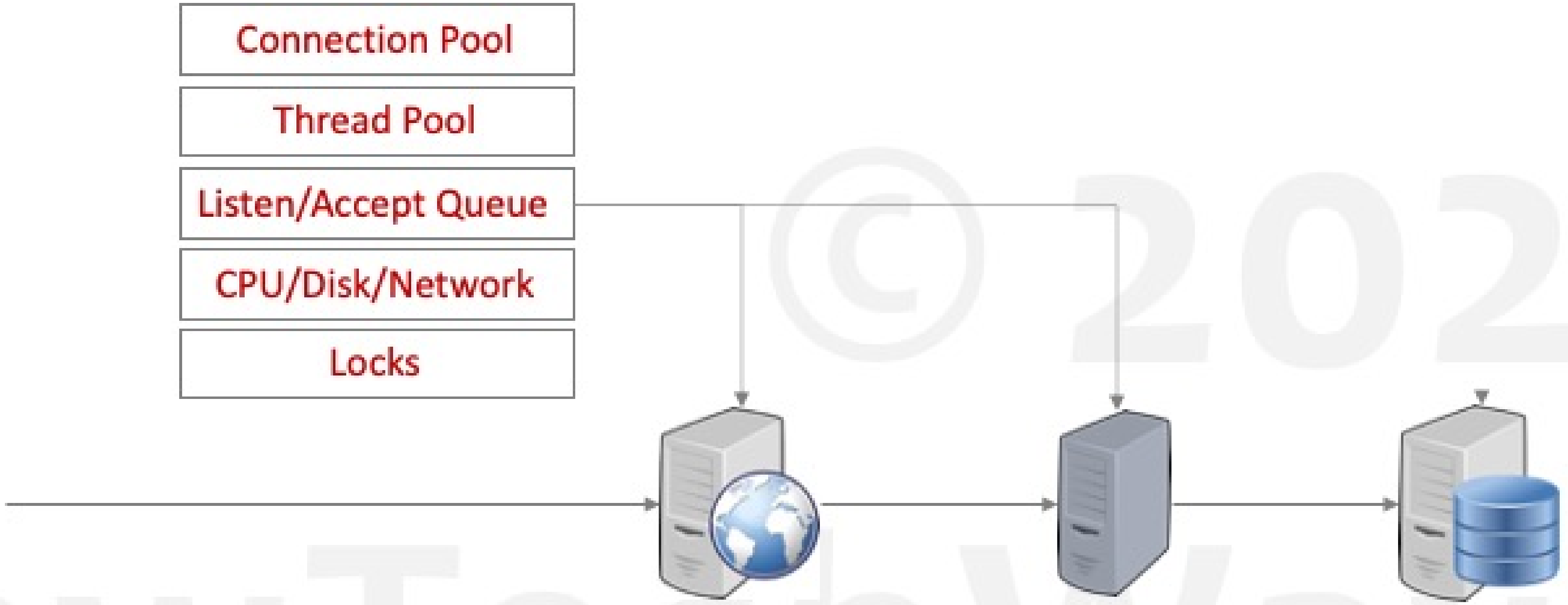
- C is capacity
- N is scaling dimension like CPU or Load
- Alpha – represents resource contention
- Beta – represents coherency delay

Linear performance when alpha and beta are zero

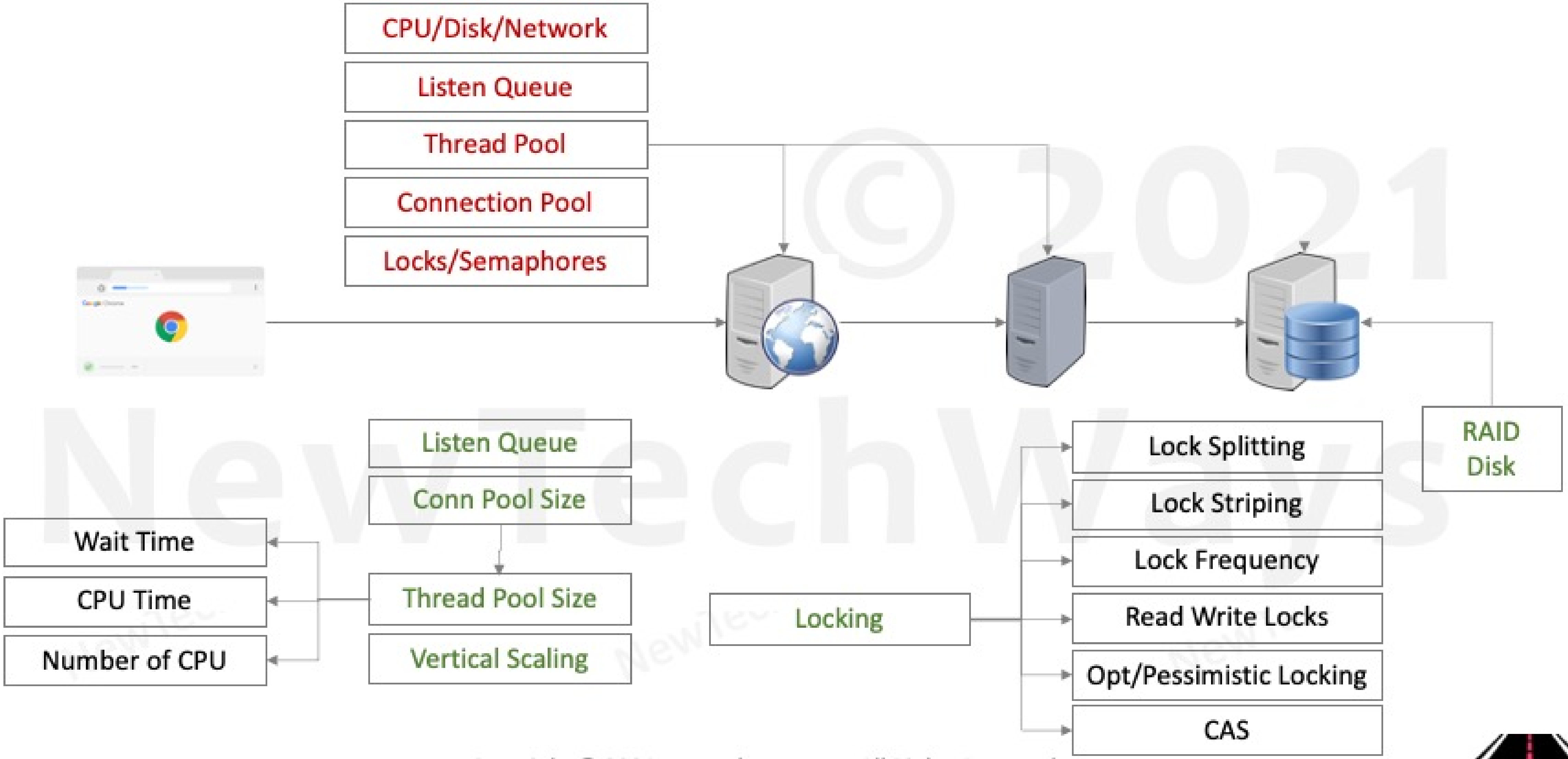


NewTechWays

Contention


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

Contention – Approaches



Minimize Lock Contention

- Reduce the duration for which a lock is held
  - Move out the code, out of synchronization block, that doesn't require a lock (especially an IO)
  - Lock Splitting – Split locks into lower granularity locks that are experiencing moderate contention
  - Lock Striping – Split locks for each partition of data like in Concurrent HashMap
- Replace exclusive locks with coordination mechanisms
  - Use ReadWriteLock/Stamped Locks
  - Use Atomic Variables (protected by CAS)

Pessimistic Locking

- Threads must wait to acquire a lock
- Used when contention is high
- May result in deadlocks
  - One of the participating thread is backed up by receiving an exception

Fetch & Lock Records	`// Begin transaction`
	`connection.setAutoCommit(false);`
	
	`// Get available inventory and lock the records`
	`connection.statement.executeQuery(`
	`" SELECT * from Inventory WHERE ProductID='XYZ' FOR UPDATE ");`

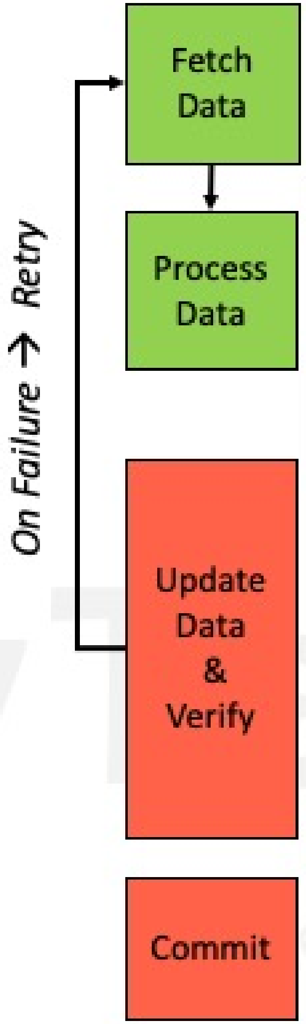
Process Records	`// Do Cart Processing`
	`//   + Get availability of other items`
	`//   + Determine when they can be delivered`
	`//   + Get the pricing and discounts of each item`

Update Records	`// Add item to cart and update inventory reservation`
	`connection.statement.executeUpdate(`
	`" UPDATE Inventory SET Quantity=(500 - 1) WHERE ProductId='XYZ' ");`

Commit	`// Commit transaction` `connection.commit();`

Optimistic Locking

- Threads do not wait for a lock
- Threads backup when they discover contention
- Use when contention is between low to moderate
- May result in starvation
  - Switch to pessimistic locking




```
// Get available inventory
connection.statement.executeQuery(
" SELECT * from Inventory WHERE ProductID='XYZ' ");


// Do Cart Processing
//    + Get availability of other items
//    + Determine when they can be delivered
//    + Get the pricing and discounts of each item


// Begin transaction
connection.setAutoCommit(false);

// Add item to cart and update inventory reservation
boolean success = connection.statement.executeUpdate(
" UPDATE Inventory SET Quantity=(Quantity - 1) WHERE ProductId='XYZ'
  WHERE (Quantity - 1) > 0 ");

if (!success) {
    // If update failed due to qty mismatch,
    // then retry by fetching the qty again
} else {
    // Commit transaction
    connection.commit();
}
```

Compare & Swap

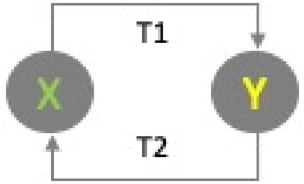
- CAS is an optimistic locking mechanism
- All modern hardware (CPU) support it
- Java implements support of CAS thorough Atomic (java.util.concurrent.atomic.*) classes

```java
AtomicInteger ai = new AtomicInteger(10);
ai.compareAndSet(10,20)

// Returns true if the value was 10
// and sets 20 as the new value

// Returns false if the value was not 10 as a result
// of a race condition with some other thread
```

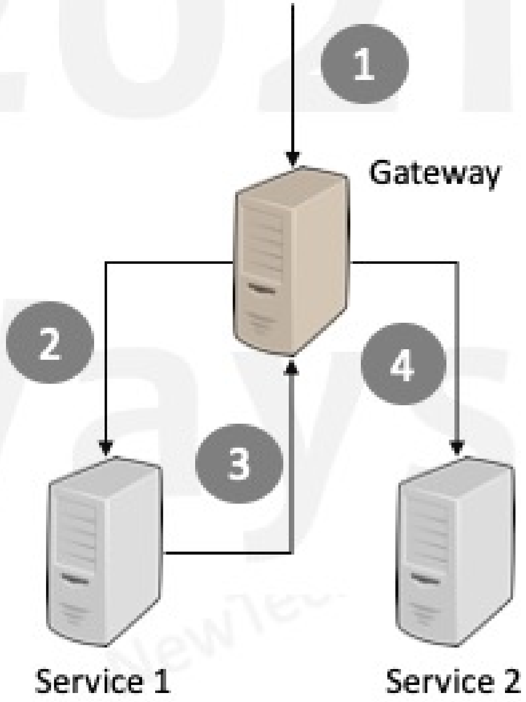
```
cqlsh:oms> select * from inventory where productId='Test-Product-7';

 productid      | quantity
----------------+----------
 Test-Product-7 |      100

(1 rows)
cqlsh:oms> update inventory set quantity=200 where productId='Test-Product-7' if quantity=0;

 [applied] | quantity
-----------+----------
     False |      100

cqlsh:oms> update inventory set quantity=200 where productId='Test-Product-7' if quantity=100;

 [applied]
-----------
      True
```

Deadlocks

- Lock Ordering Related
  - Result of threads trying to acquire multiple locks
    - Simultaneous money transfer from X and Y accounts by thread T1 and T2
    - T1: from X to Y
    - T2: from Y to X
  - Acquire locks in a fixed global order
    - Acquire locks only in the sort order of account numbers: X and then Y



- Request Load Related
  - Threads waiting for connections to multiple databases
    - May run out of enough connections resulting in deadlocks
  - Threads waiting for other threads to be spawned and perform some work
    - May run out of enough threads resulting in deadlocks



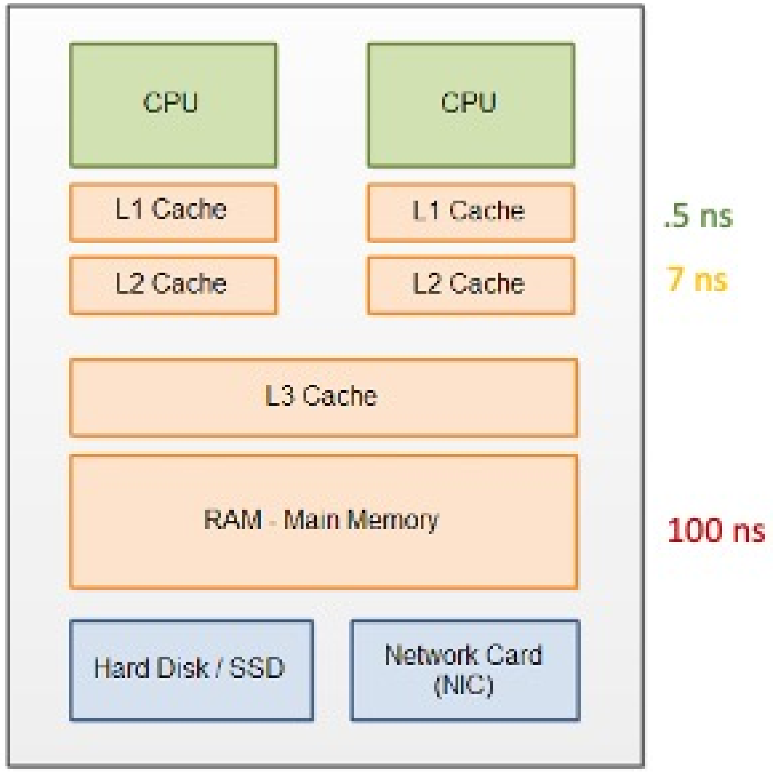
NewTechWays

Coherence Delays

- Visibility (Volatile)
  - Java guarantees that a volatile object is always read from main memory and written back to main memory when updated in a processor
- Locking (Synchronized)
  - All variables accessed inside a sync block are read from the main memory at the start of the sync block
  - All variables modified in a sync block are flushed to the main memory when the associated thread exists the sync block

  Synchronized ensures locking & visibility.
  Volatile only ensures visibility

- These guarantees are provided using memory barriers which may result in invalidating or flushing of caches



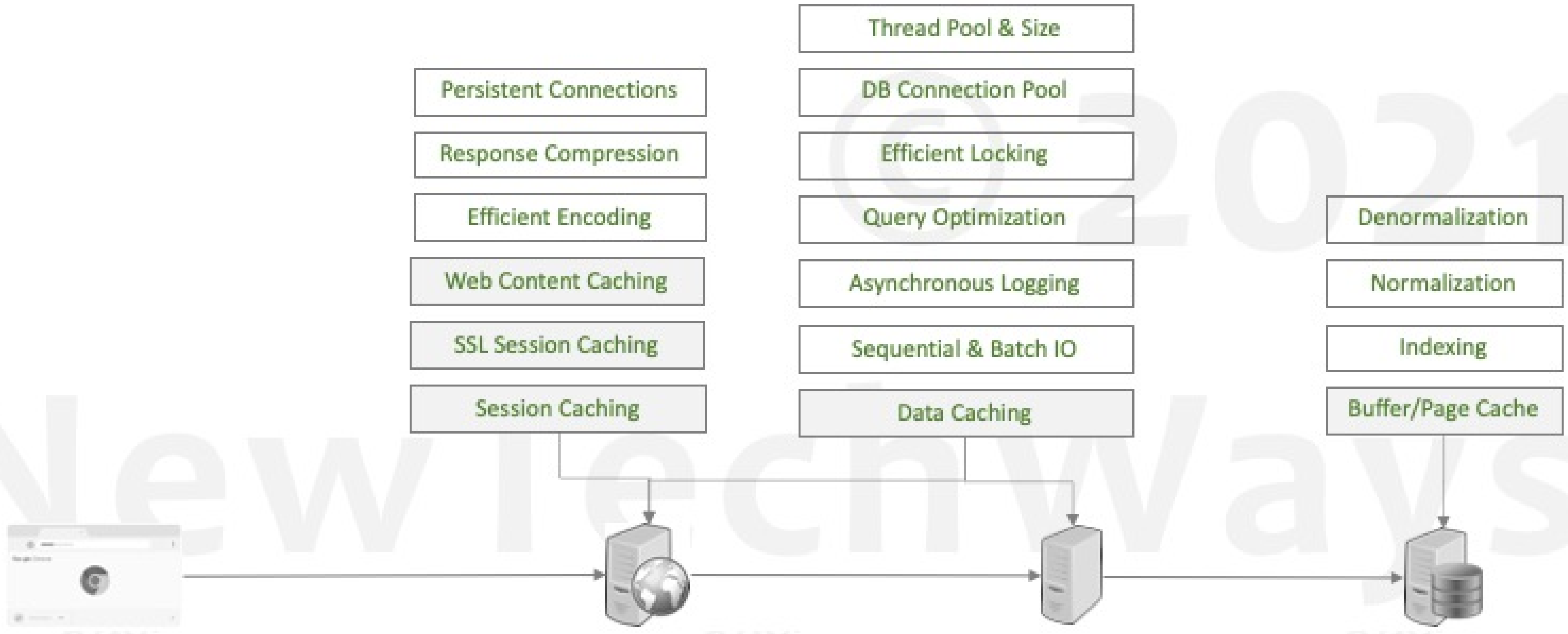
L1 Cache: Faster

L2 Cache: Slower, Bigger, Cheaper

NewTechWays

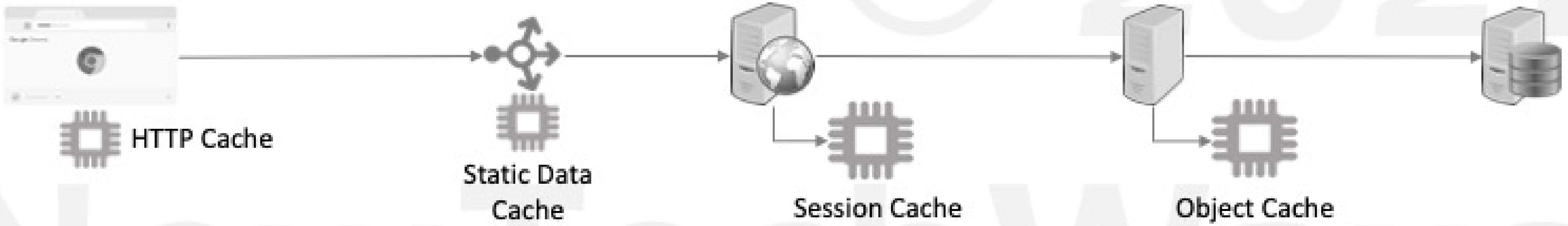
Caching

System Architecture for Performance



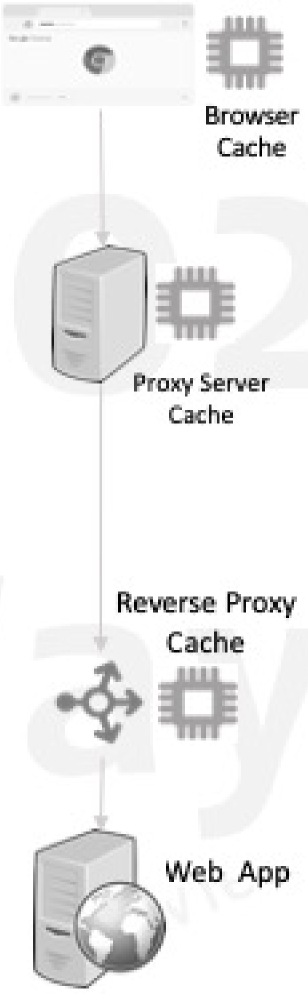
NewTechWays

Caching



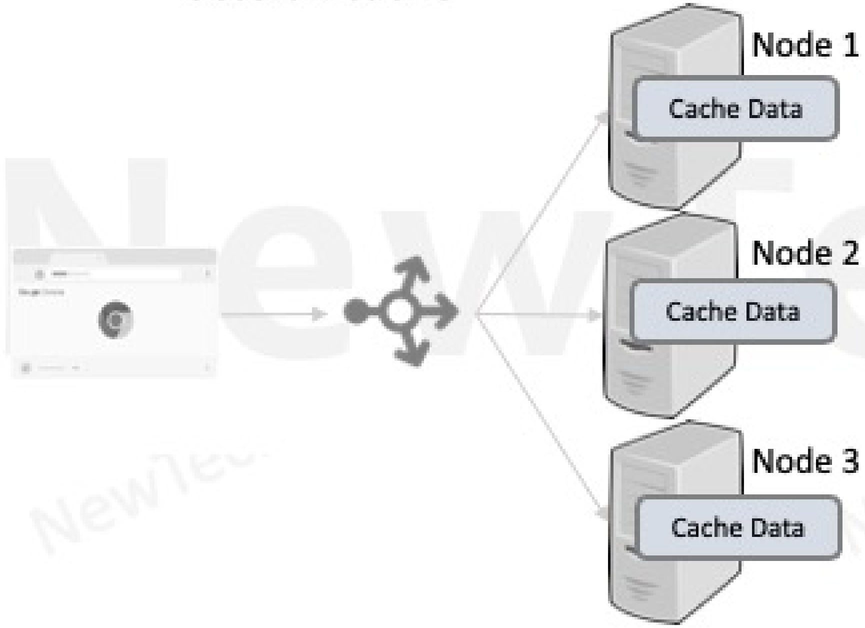
HTTP Caching for Static Data

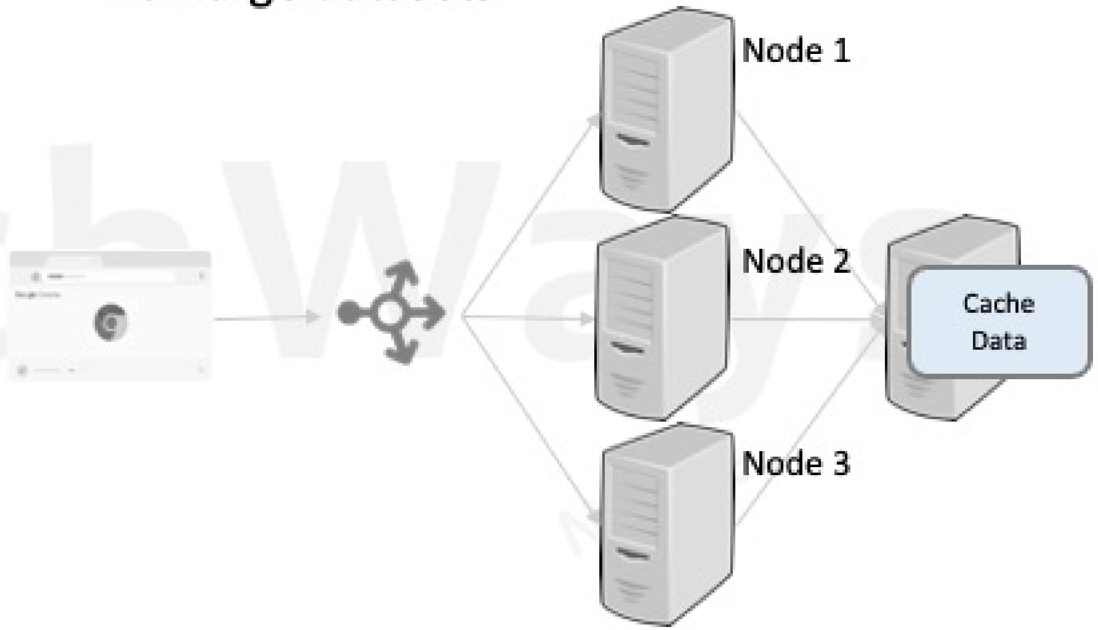
- GET method responses are idempotent and hence good candidates for caching
- Headers
  - Cache-control : If a resource can be cached
    - No-cache : Do not use cache without validating with origin server.
    - Must-revalidate: Like no-cache but need to validate only after its max-age (even if client is ready to accept stale data)
    - No-store: Do not cache at all
    - Public : Any shared cache can cache
    - Private : Only a client cache can cache
    - Max-age : Maximum age of a resource in cache, relative to resource request time
  - ETAG : A hash code for indicating version of a resource
    - Invalidates previous version cache



Caching Dynamic Data

- Exclusive Cache
  - Has low latency
  - Without routing can lead to duplication
    - Useful for smaller datasets
  - With routing can lead to uneven load balancing
    - Session cache
- Shared Cache
  - Higher latency due to an extra hop
  - Can scale out to a distributed cache
    - Memcache
    - Redis
  - For large datasets





Caching Challenges

- Limited cache space results in early evictions
  - Prefer caching for frequently accessed objects
    - Cache fast-moving consumer goods vs slow moving goods
  - Average size of cached objects should be as small as possible
    - Large sized objects results in cache getting full too soon causing evictions

$$\text{Cache Hit Ratio} = \frac{\text{\# of cache hits}}{\text{\# of cache hits + \# of cache miss}}$$

- Cache Invalidation & Cache Inconsistency
  - Requires Update/Deletion of cached value upon update
    - Not an option when a cache is outside of a system
    - No cache inconsistency
  - TTL value can be used to remove aged data
    - High TTL results in more cache hits
      - Inconsistency interval increases
    - Low TTL decreases inconsistency interval
      - Cache hits go down

Summary

- Performance Problems are a result of request/job queue building up in a system
- Performance Measurement – Latency, Throughput, and Resource Saturation
  - Watch out tail latency for hidden problems or future problems
- Improving Latency
  - Reduce request response time of serial requests by improving resource utilization
    - CPU, Network, Memory, Disk
  - Caching – Minimize fetching frequently read rarely mutated data from disk or network
- Improving Throughput
  - Improve concurrency of concurrent requests/jobs
  - Minimize request/job serialization
    - Reduce lock contention by reducing lock granularity, lock striping, lock splitting and CAS
    - Prefer Optimistic locking over Pessimistic locking when lock contention is low
    - Eliminate deadlocks when using pessimistic locking

Thanks!

NewTechWays

https://www.newtechways.com