What the heck is an In-Memory Data Grid?

@addisonhuddy

How are we going to answer this question?

1. Tell you about my first introduction to IMDGs
2. See some real-world use cases
3. Design an IMDG
4. Implement Use Cases

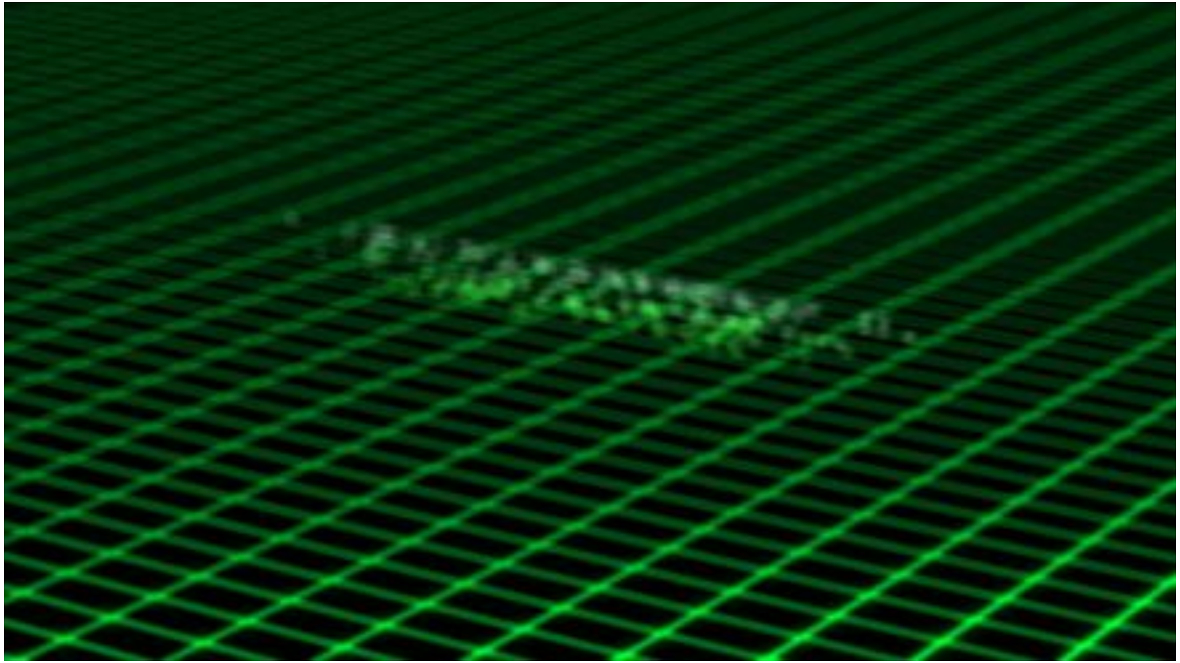
Definition

> IMDGs provide a lightweight, distributed, scale-out in-memory object store — the data grid. Multiple applications can concurrently perform transactional and/or analytical operations in the low-latency data grid, thus minimizing access to high-latency, hard-disk-drive-based or solid-state-drive-based data storage.[1]

Gartner

[1] https://www.gartner.com/reviews/market/in-memory-data-grids

My First Thought



My Second Thought

Two Examples

China Railway Corporation

5,700 train stations
4.5 million tickets per day
20 million daily users
1.4 billion page views per day
40,000 visits per second

Southwest Airlines

70+ cities
4,000 daily flights
706 aircraft
Largest airline website by visitors

When Not To Use An IMDG

- Small Amounts of Data
- Low-latency isn’t mission critical
- Not a total replacement for RDBMS

Let’s Make an IMDG

Design Goals

- Extremely Low Latency
- High Throughput
- Durability
- Large Datasets
- Consistency?

Design Goals

- Extremely Low Latency
- High Throughput
- Durability
- Large Datasets
- Consistency

- Memory First
- Horizontal Scalability / Elasticity
- Data Aware Routing
- Serialization / Deserialization

APACHE GEODE

https://github.com/apache/geode

Memory First

Latency Comparison

```
Latency Comparison Numbers
--------------------------
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
SSD Seek                               100,000   ns      100 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
```

[1] Credit Jeff Dean, Peter Norvig, and Jonas Bonér

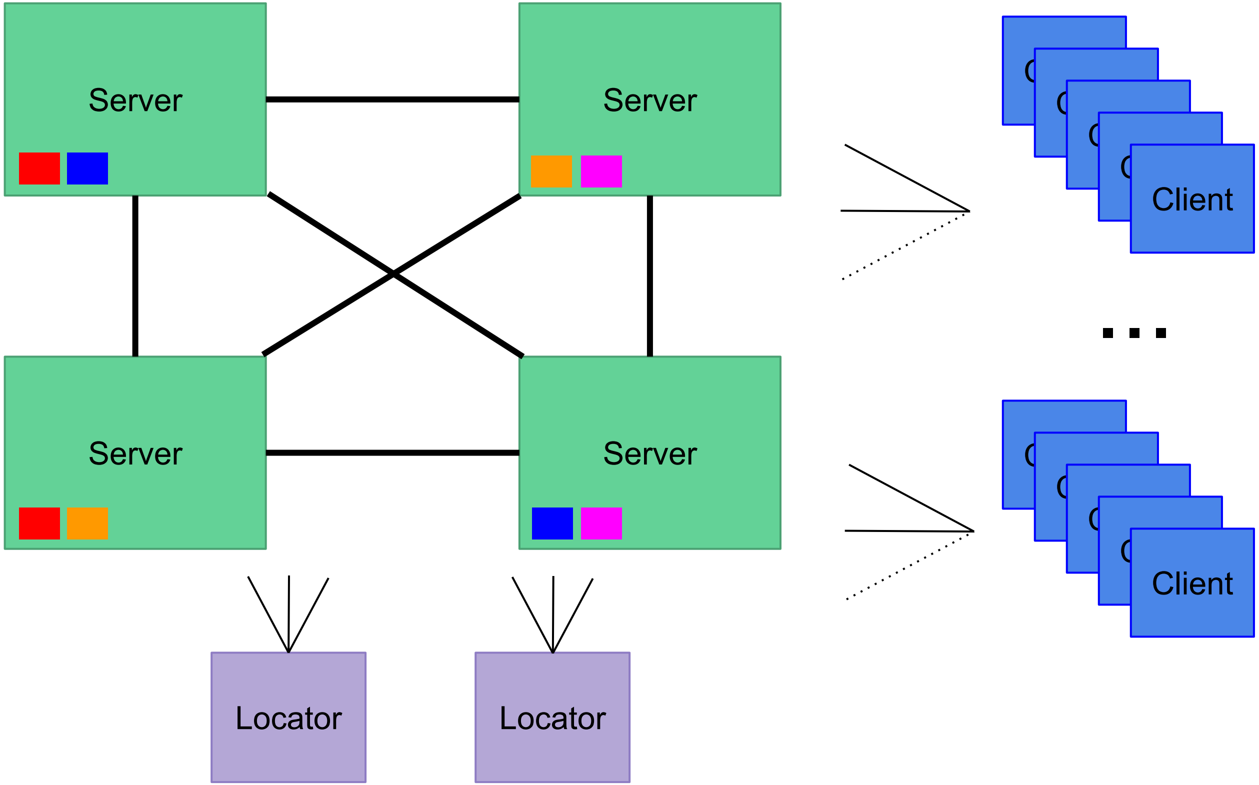
Why Memory?

Read 1 MB Comparison

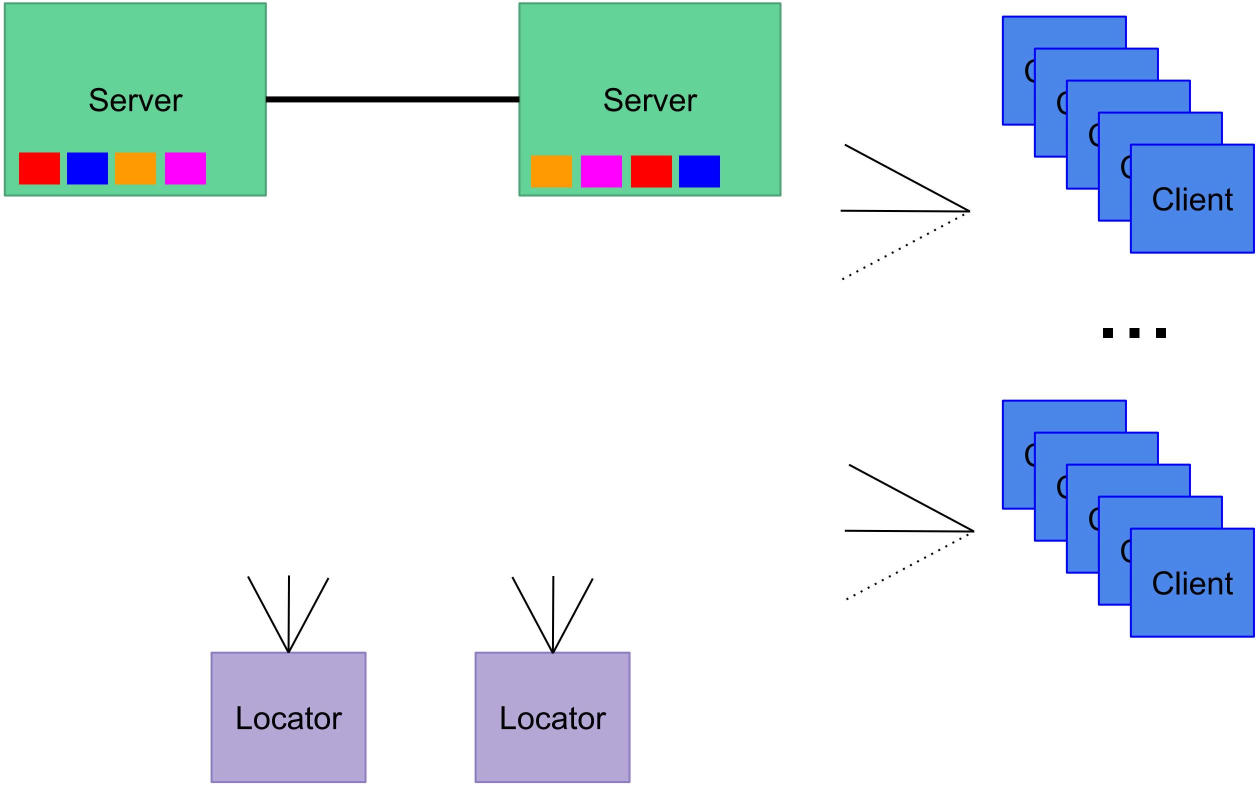
Hardware	True Time	Scaled Time
Memory	250,100 ns	2 days
SSD	1,100,000 ns	9 days
Disk	30,000,000	8 months

Horizontal Scalability / Elasticity

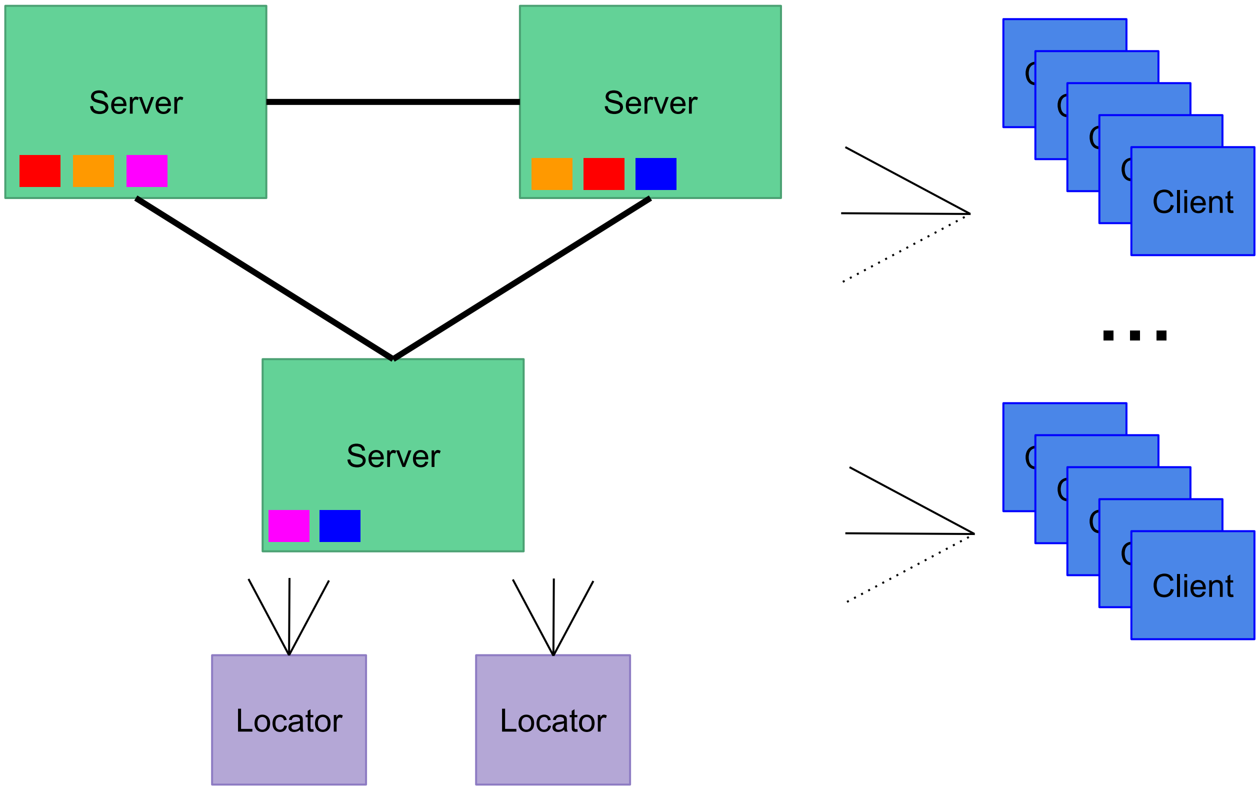
System Architecture



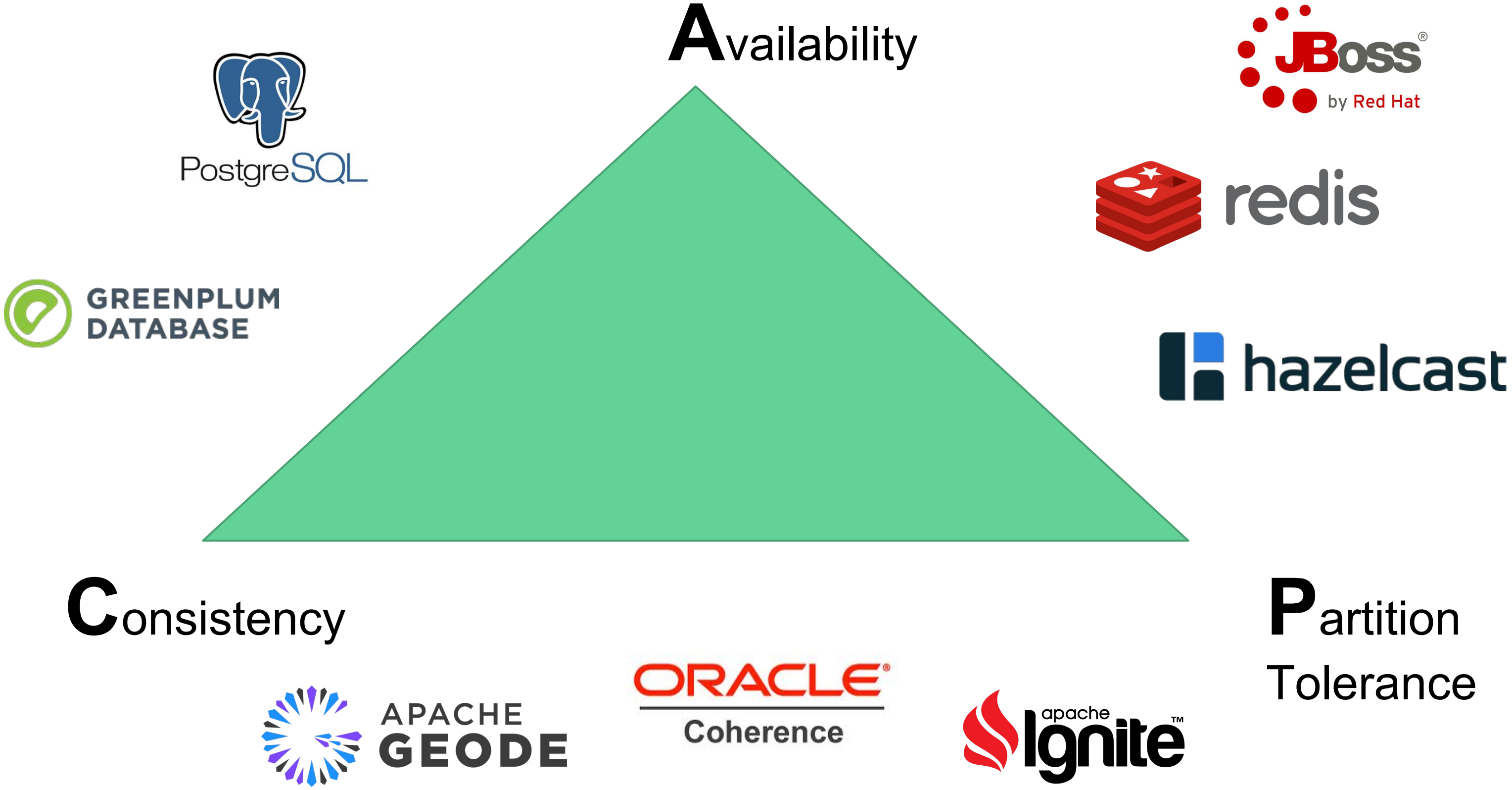
System Architecture



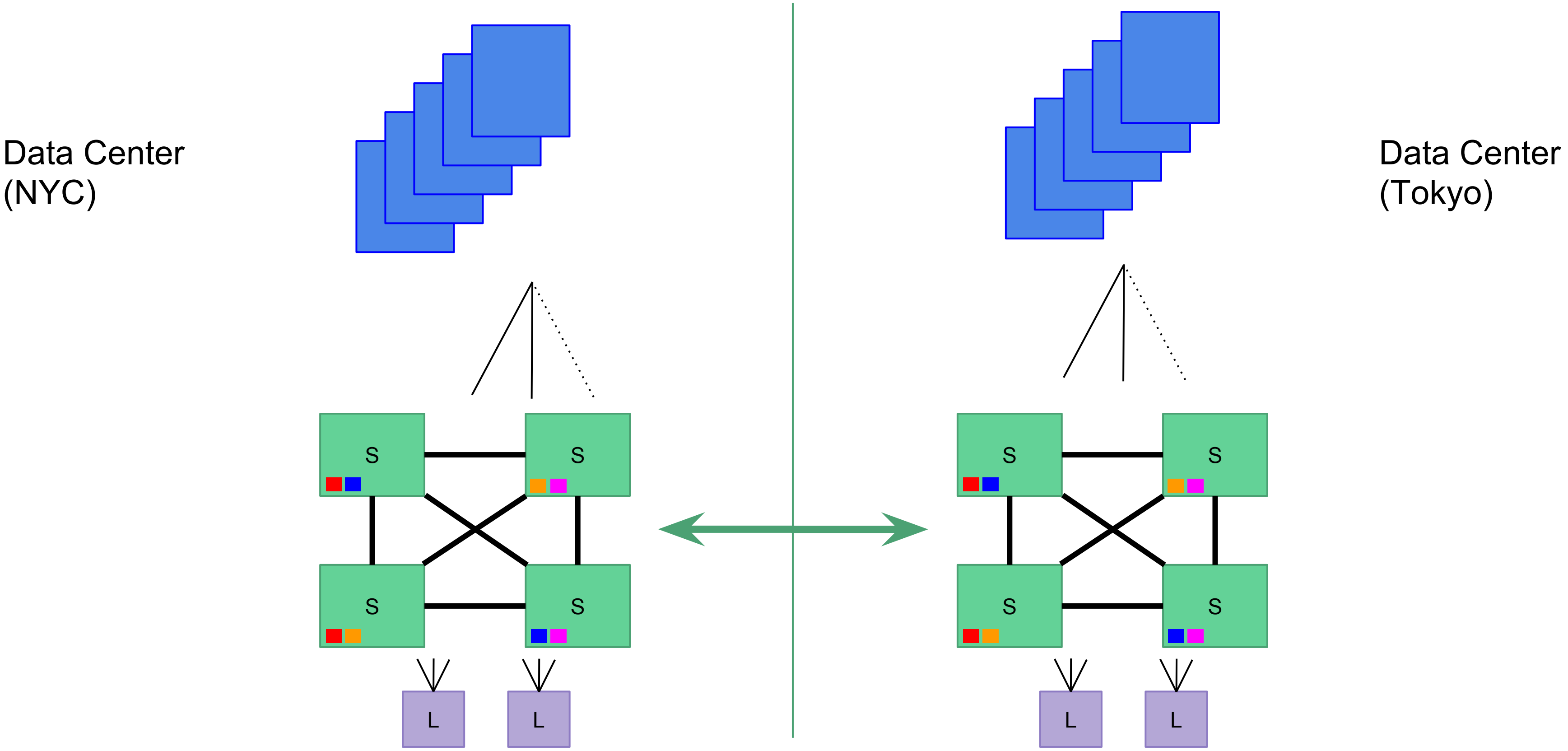
System Architecture



IMDGs & CAP Theorem



WAN Replication



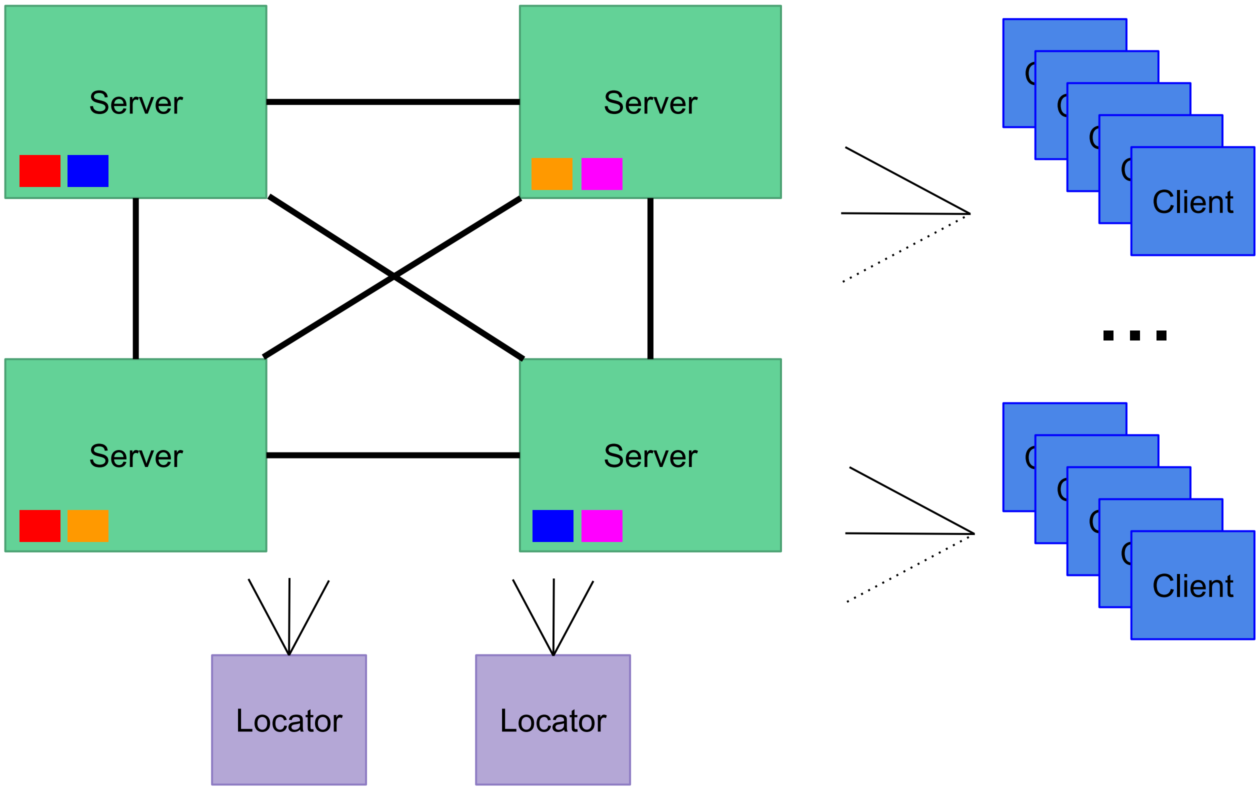
Data Aware Routing

Latency Comparison

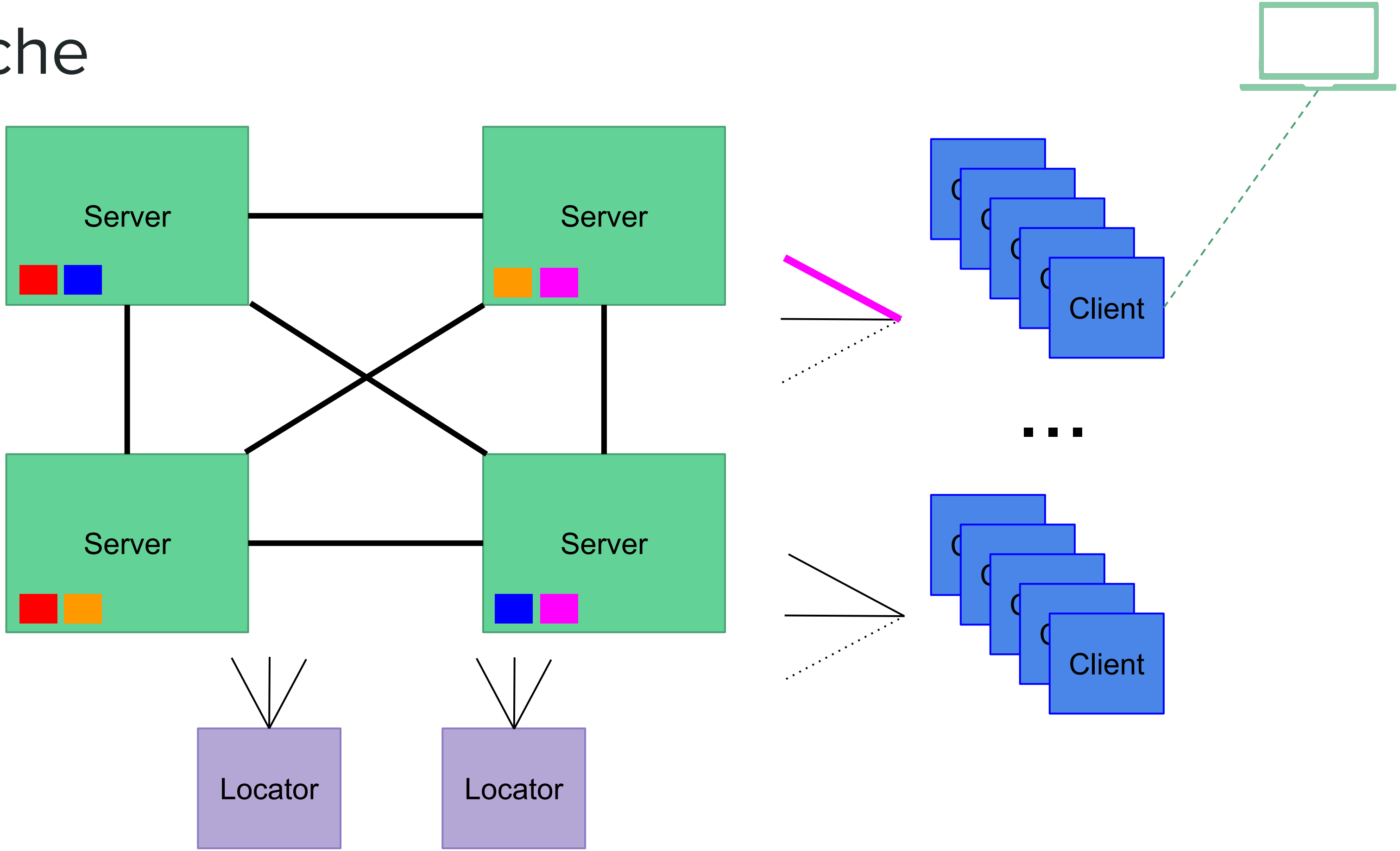
```
Latency Comparison Numbers
--------------------------
L1 cache reference                         0.5 ns
Branch mispredict                          5   ns
L2 cache reference                         7   ns                      14x L1 cache
Mutex lock/unlock                         25   ns
Main memory 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
SSD Seek                             100,000   ns      100 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
```

[1] Credit Jeff Dean, Peter Norvig, and Jonas Bonér

Single Hop

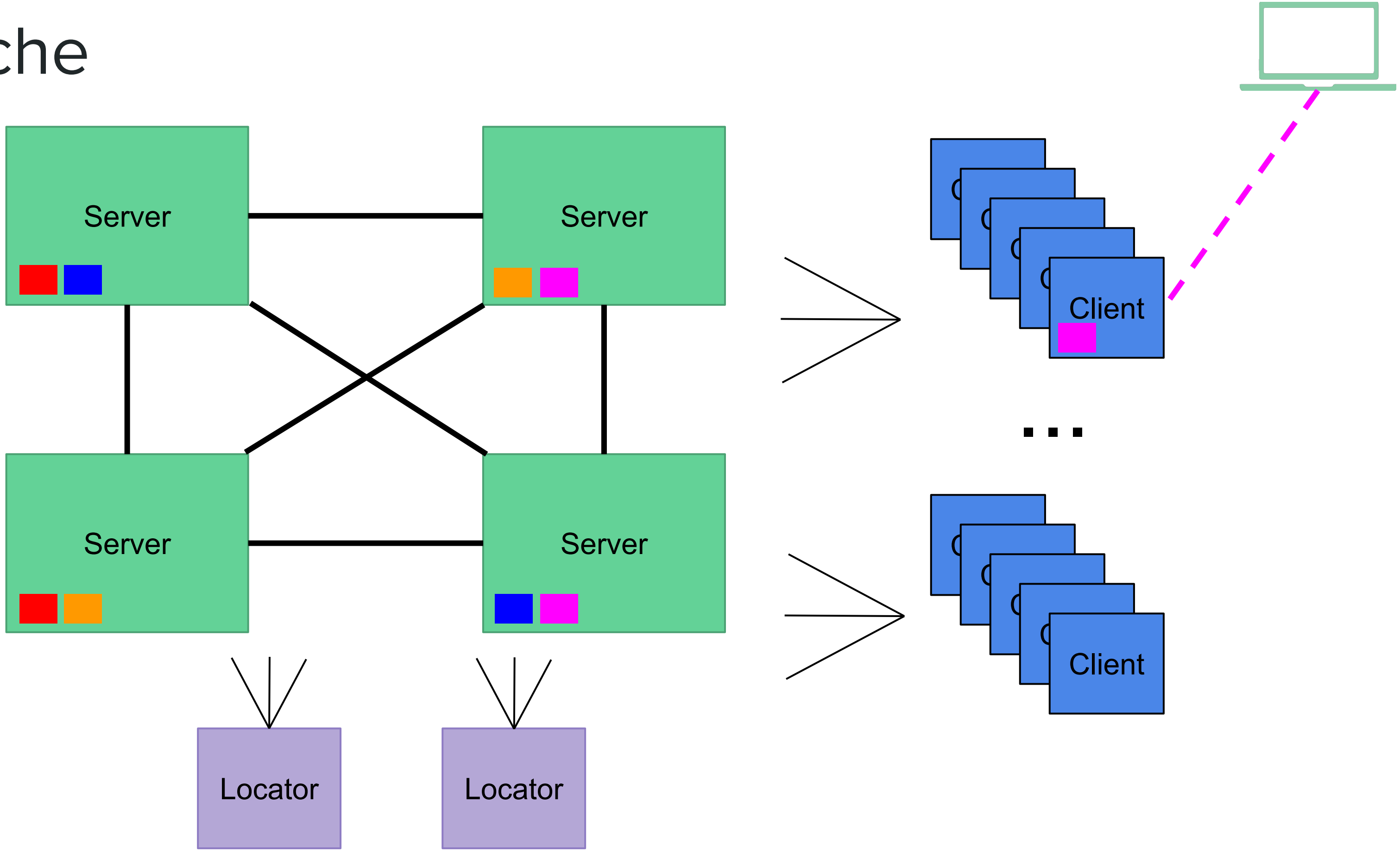

Server
Server
Server
Server
Client
Client
Locator
Locator

Local Cache



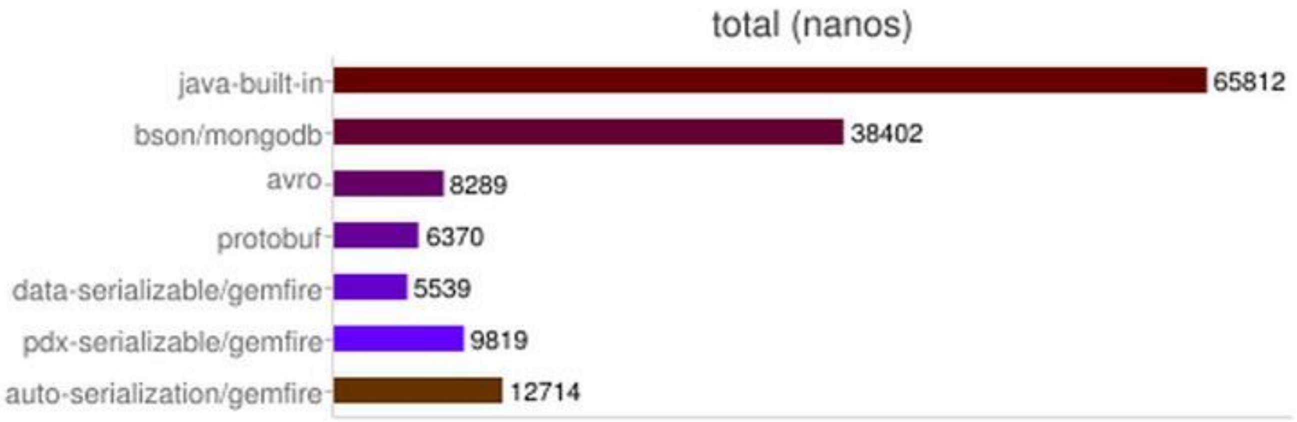
Server
Server
Server
Server
Client
Client
Locator
Locator

Local Cache



Server
Server
Server
Server
Client
...
Client
Locator
Locator

Serialization



1. Only (de)serialize when it is necessary
2. Only (de)serialize what is absolutely necessary
3. Distribute (de)serialize cost as much as possible

Basic User Operations

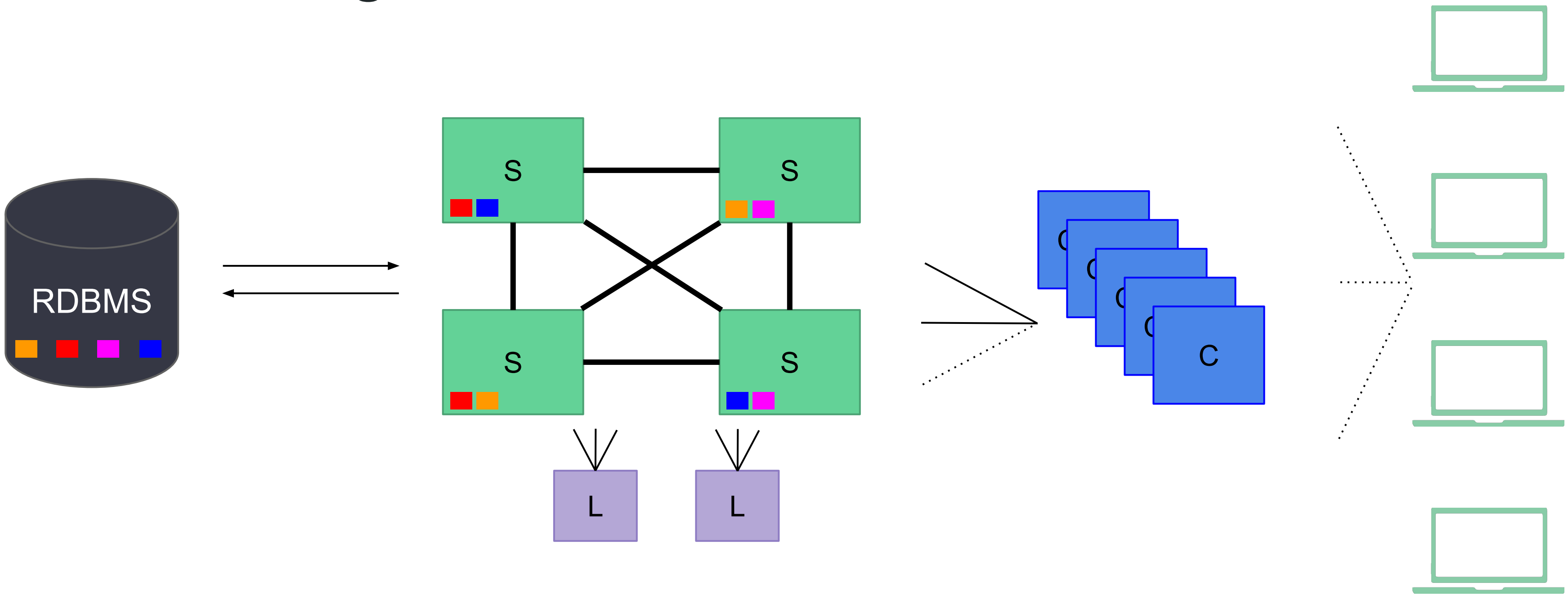
What have we created?

- Key/Value Object Store
- Share-nothing architecture
- Memory Oriented
- Strongly Consistent

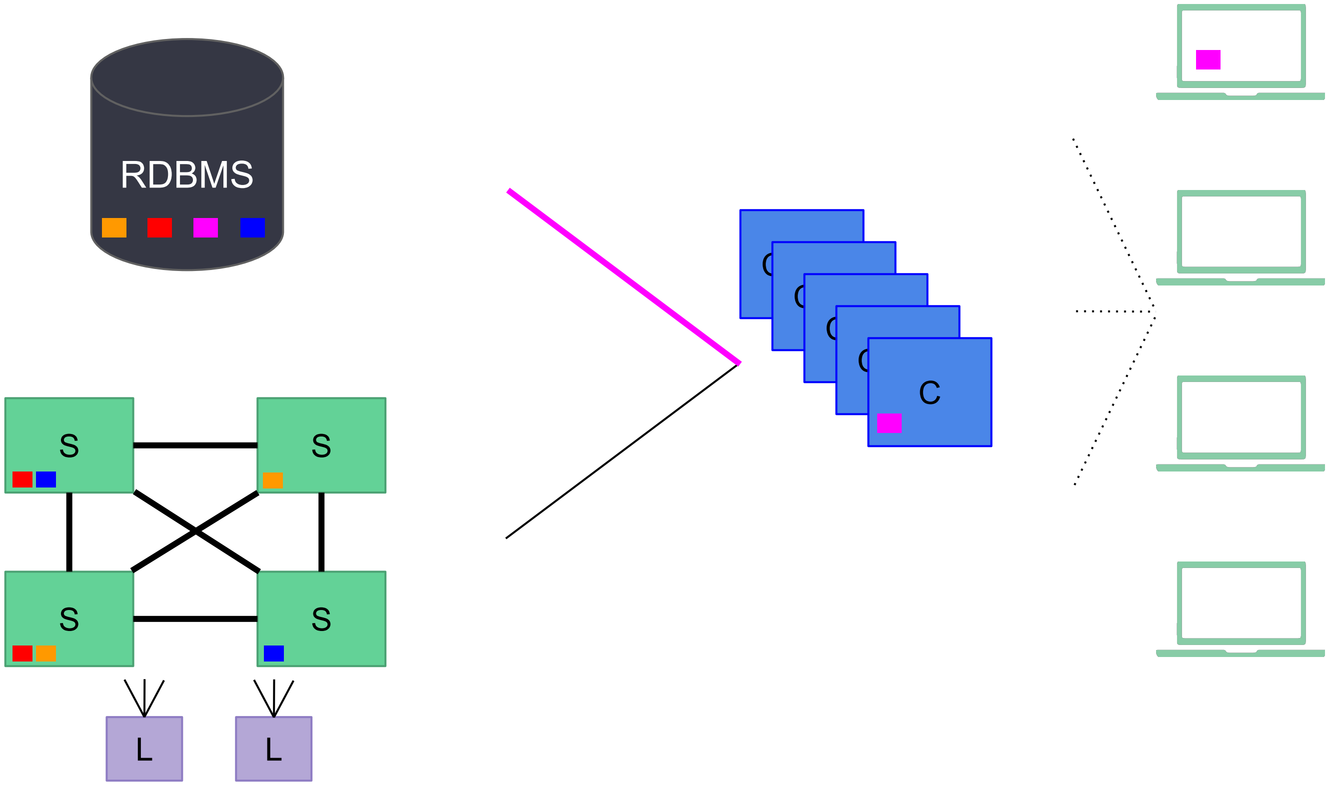
- Put/Get
- Queries
- Server-side functions
- Registered Interests
- Continuous Queries
- Event Queues

Use Cases

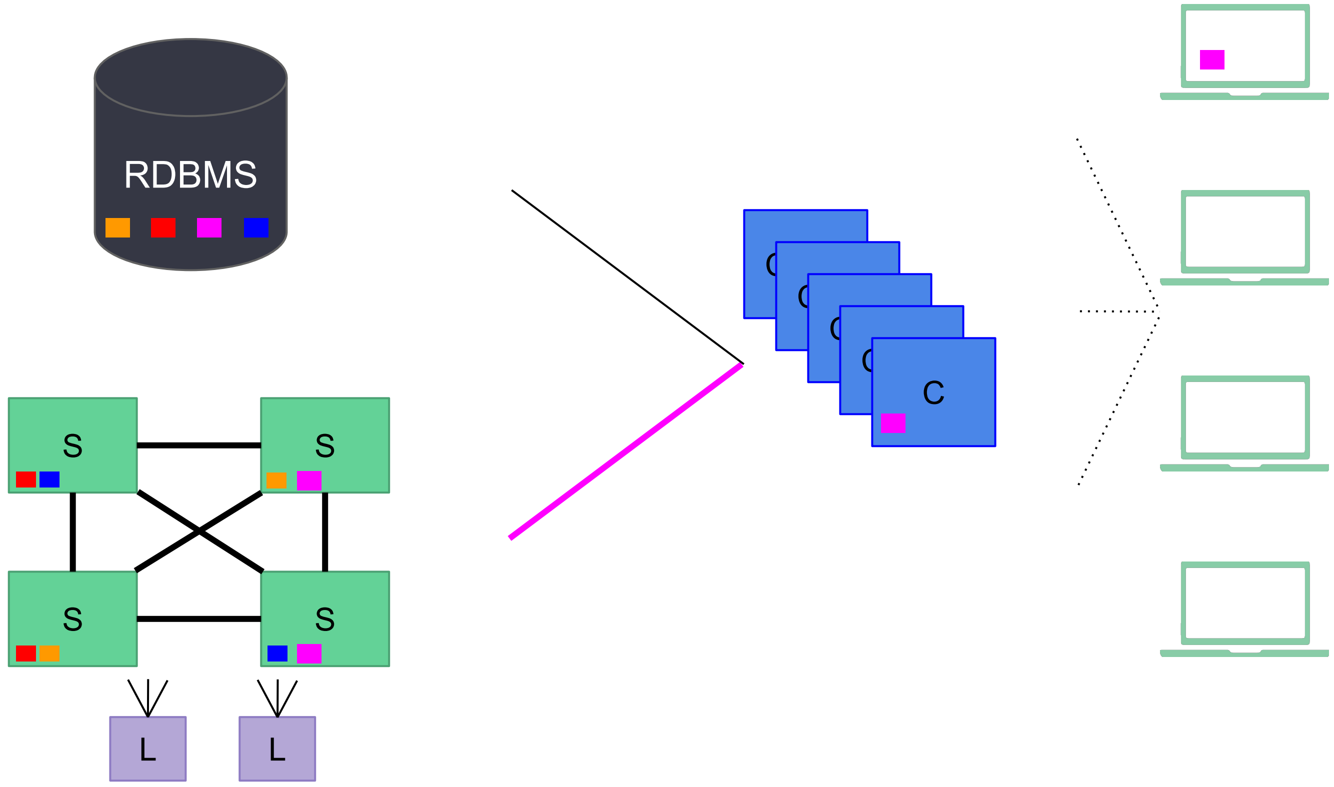
In-line Caching



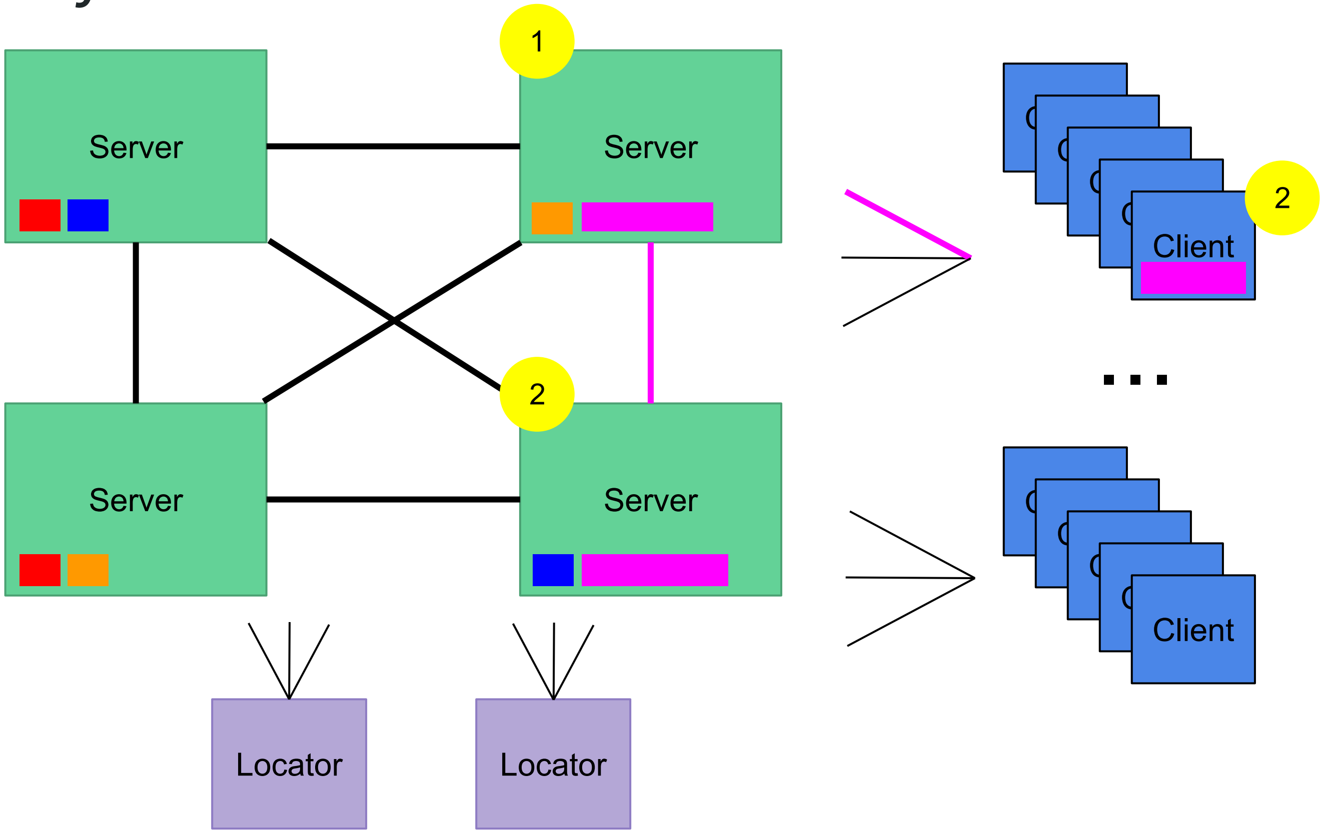
Look-Aside Caching



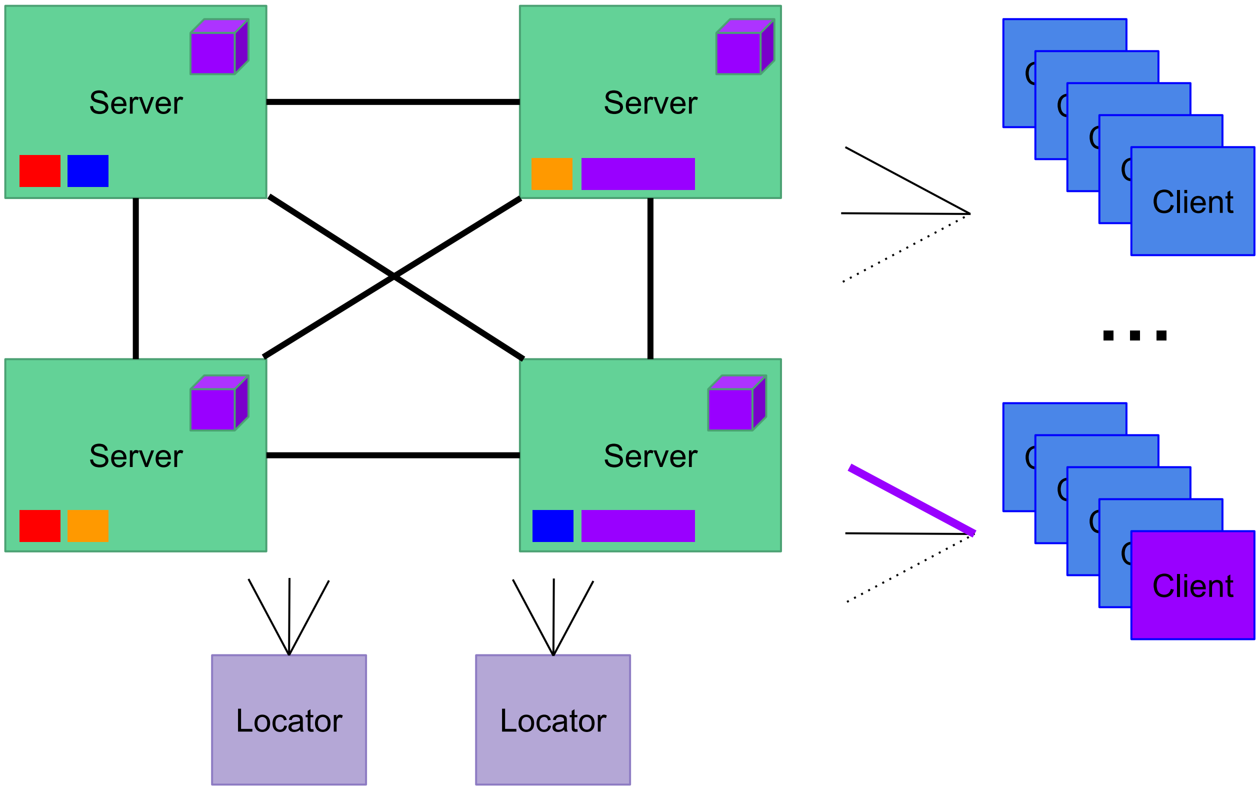
Look-Aside Caching



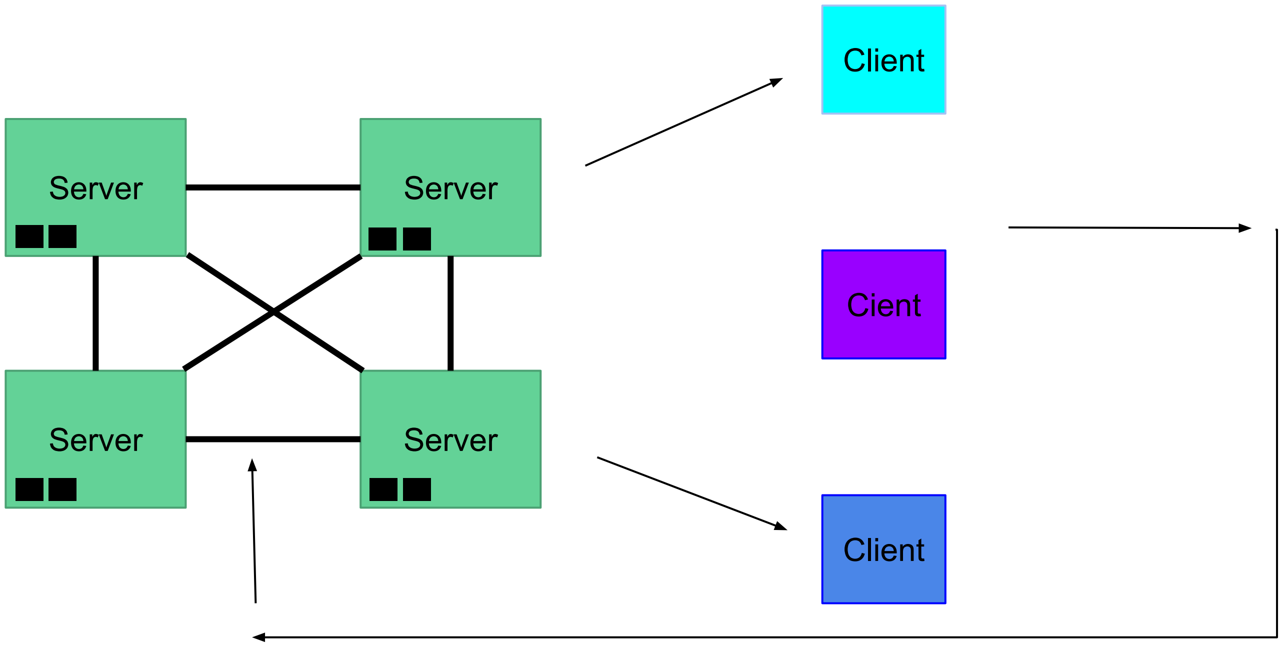
Pub / Sub System



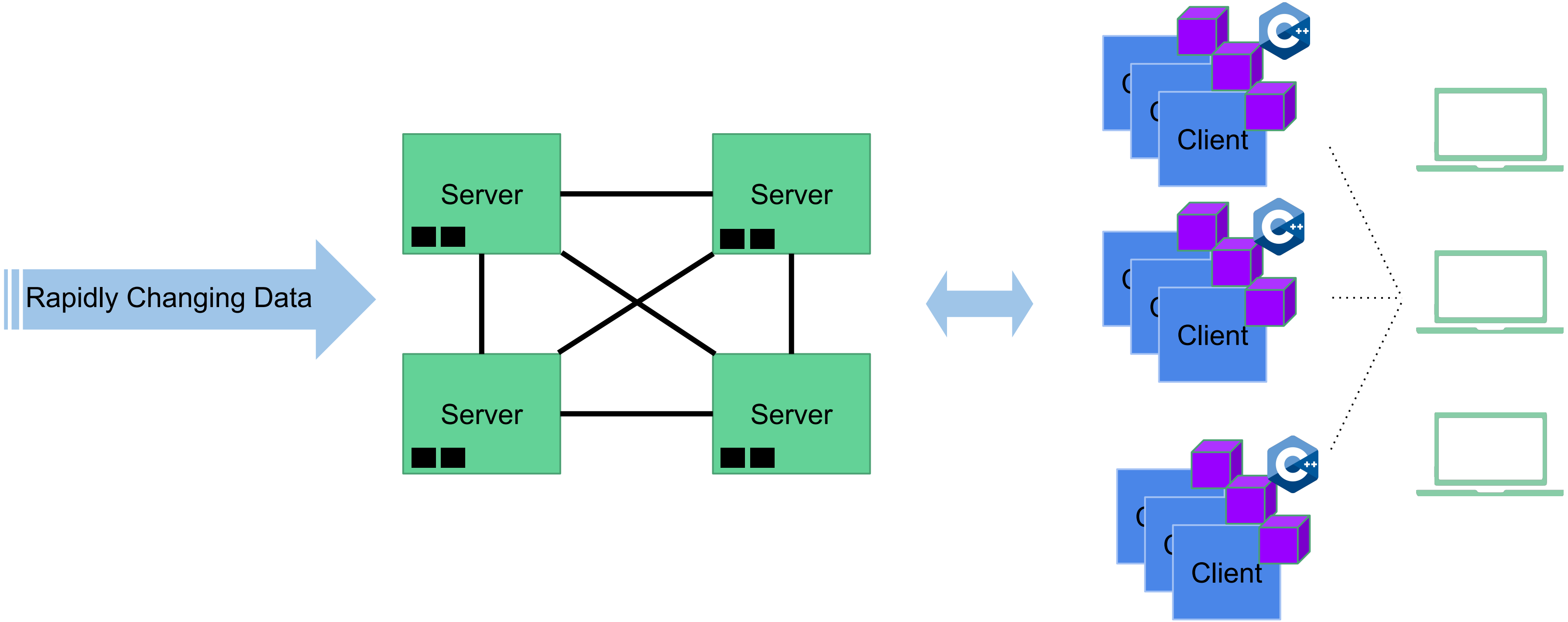
Real-Time Analytics with Functions



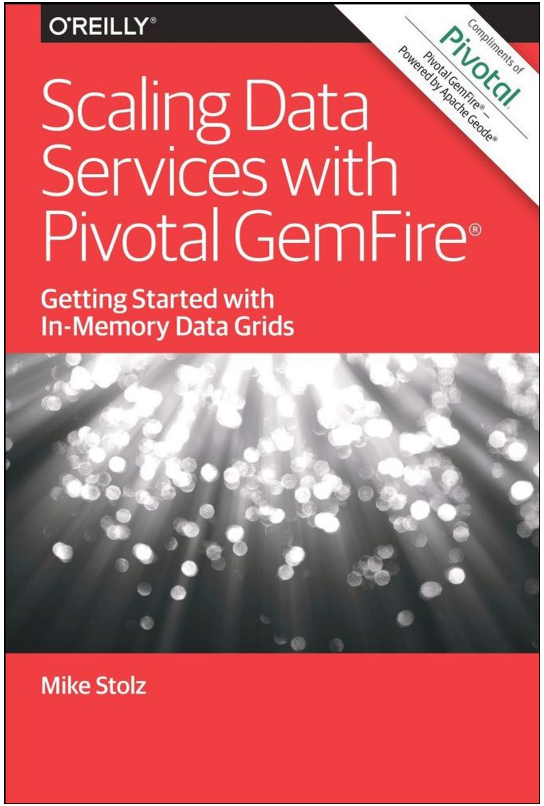
Distributed Computation



Real-Time Analytics



O’Reilly Book



Questions

@addisonhuddy