Real Logic

ACCELERATING SOFTWARE

Cluster Consensus

When Aeron Met Raft

Martin Thompson - @mjpt777

What does “Consensus” mean?

con•sen•sus

noun \ kən-ˈsen(t)-səs \

: **general agreement : <u>unanimity</u>**

Source: http://www.merriam-webster.com/

con•sen•sus

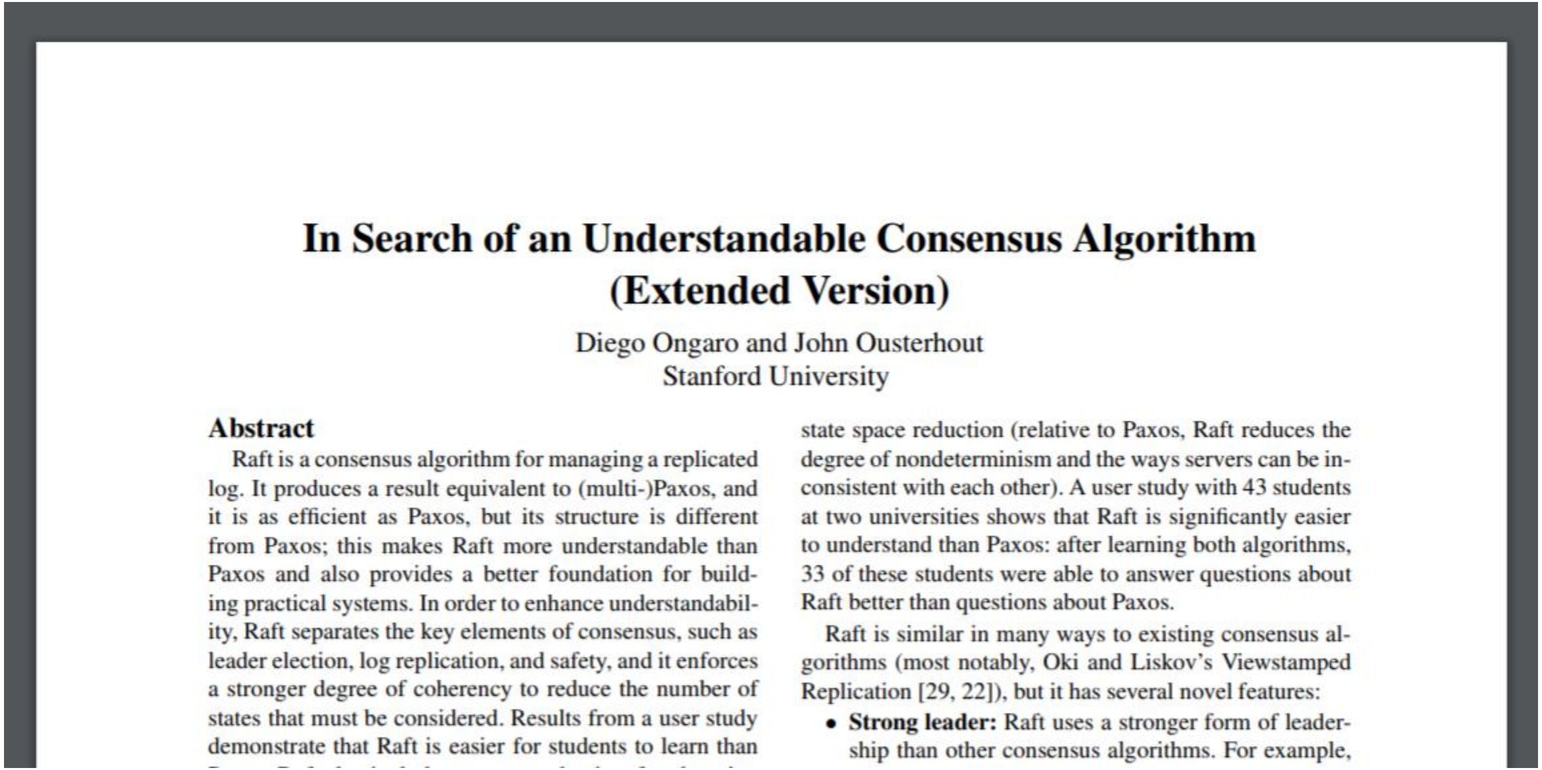
noun \ kən-ˈsen(t)-səs \

: **general agreement : <u>unanimity</u>**

: **the judgment arrived at by <u>most</u> of those concerned**

Source: http://www.merriam-webster.com/

https://raft.github.io/raft.pdf



In Search of an Understandable Consensus Algorithm (Extended Version)

Diego Ongaro and John Ousterhout
Stanford University

Abstract

Raft is a consensus algorithm for managing a replicated log. It produces a result equivalent to (multi-)Paxos, and it is as efficient as Paxos, but its structure is different from Paxos; this makes Raft more understandable than Paxos and also provides a better foundation for building practical systems. In order to enhance understandability, Raft separates the key elements of consensus, such as leader election, log replication, and safety, and it enforces a stronger degree of coherency to reduce the number of states that must be considered. Results from a user study demonstrate that Raft is easier for students to learn than

state space reduction (relative to Paxos, Raft reduces the degree of nondeterminism and the ways servers can be inconsistent with each other). A user study with 43 students at two universities shows that Raft is significantly easier to understand than Paxos: after learning both algorithms, 33 of these students were able to answer questions about Raft better than questions about Paxos.

Raft is similar in many ways to existing consensus algorithms (most notably, Oki and Liskov's Viewstamped Replication [29, 22]), but it has several novel features:

- **Strong leader:** Raft uses a stronger form of leadership than other consensus algorithms. For example, https://www.cl.cam.ac.uk/~ms705/pub/papers/2015-osr-raft.pdf

Raft Refloated: Do We Have Consensus?

Heidi Howard Malte Schwarzkopf Anil Madhavapeddy Jon Crowcroft
University of Cambridge Computer Laboratory
first.last@cl.cam.ac.uk

ABSTRACT

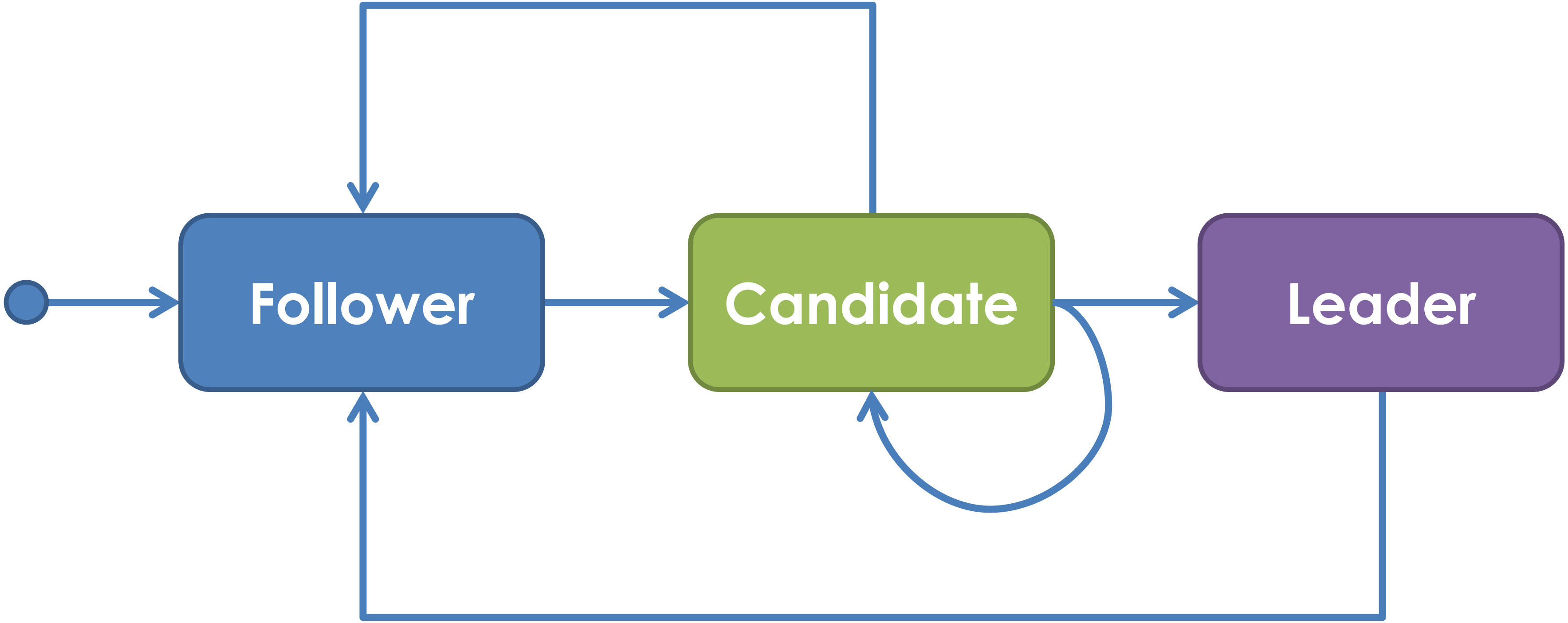
The Paxos algorithm is famously difficult to reason about and even more so to implement, despite having been synonymous with distributed consensus for over a decade. The recently proposed Raft protocol lays claim to being a new, understandable consensus algorithm, improving on Paxos without making compromises in performance or correctness.

ation ought to be far easier than with Multi-Paxos. Our study in this paper evaluates the claims about Raft made by its designers. Is it indeed easily understandable, and can the encouraging performance and correctness results presented by Ongaro and Ousterhout be independently confirmed?

In the endeavour to answer this question, we re-implemented Raft in a functional programming language (OCaml) and repeat the

Raft in a Nutshell

Roles



RPCs

1. **RequestVote RPC**
   Invoked by candidates to gather votes

2. **AppendEntries RPC**
   Invoked by leader to replicate and heartbeat

Safety Guarantees

- Election Safety
- Leader Append-Only
- Log Matching
- Leader Completeness
- State Machine Safety

Monotonic Functions

Version all the things!

Clustering Aeron

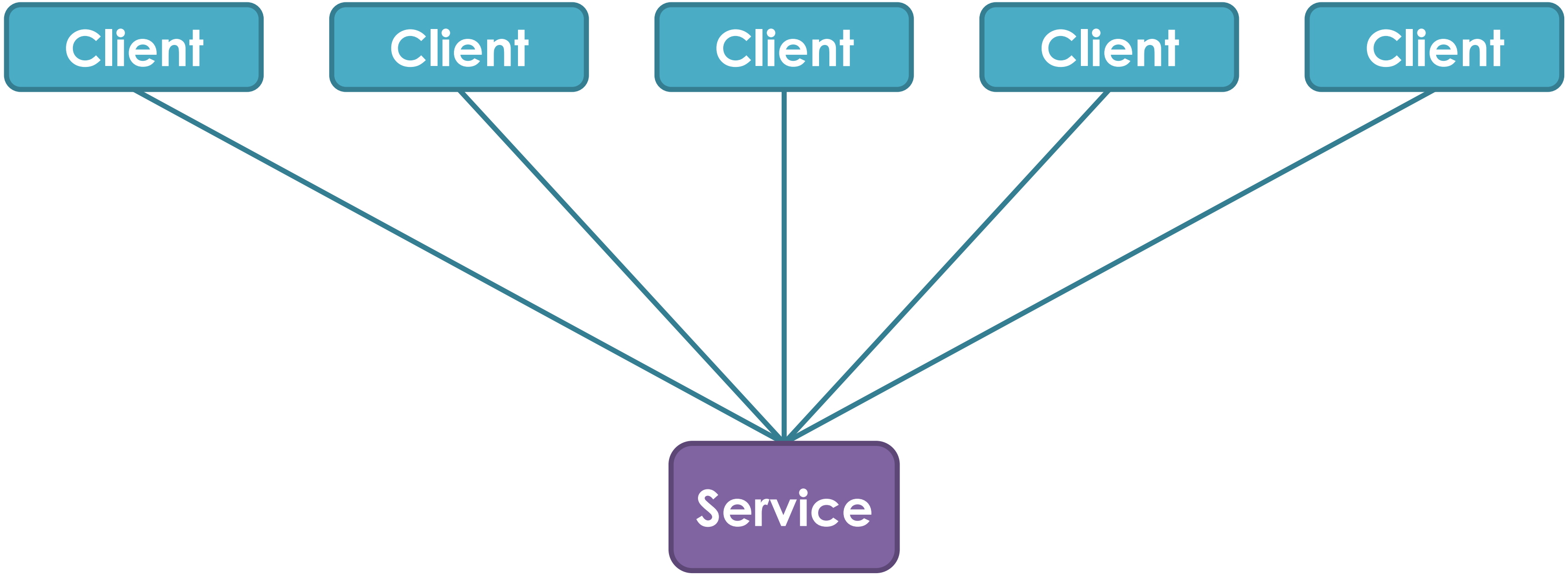
Is it Guaranteed Delivery™ ???

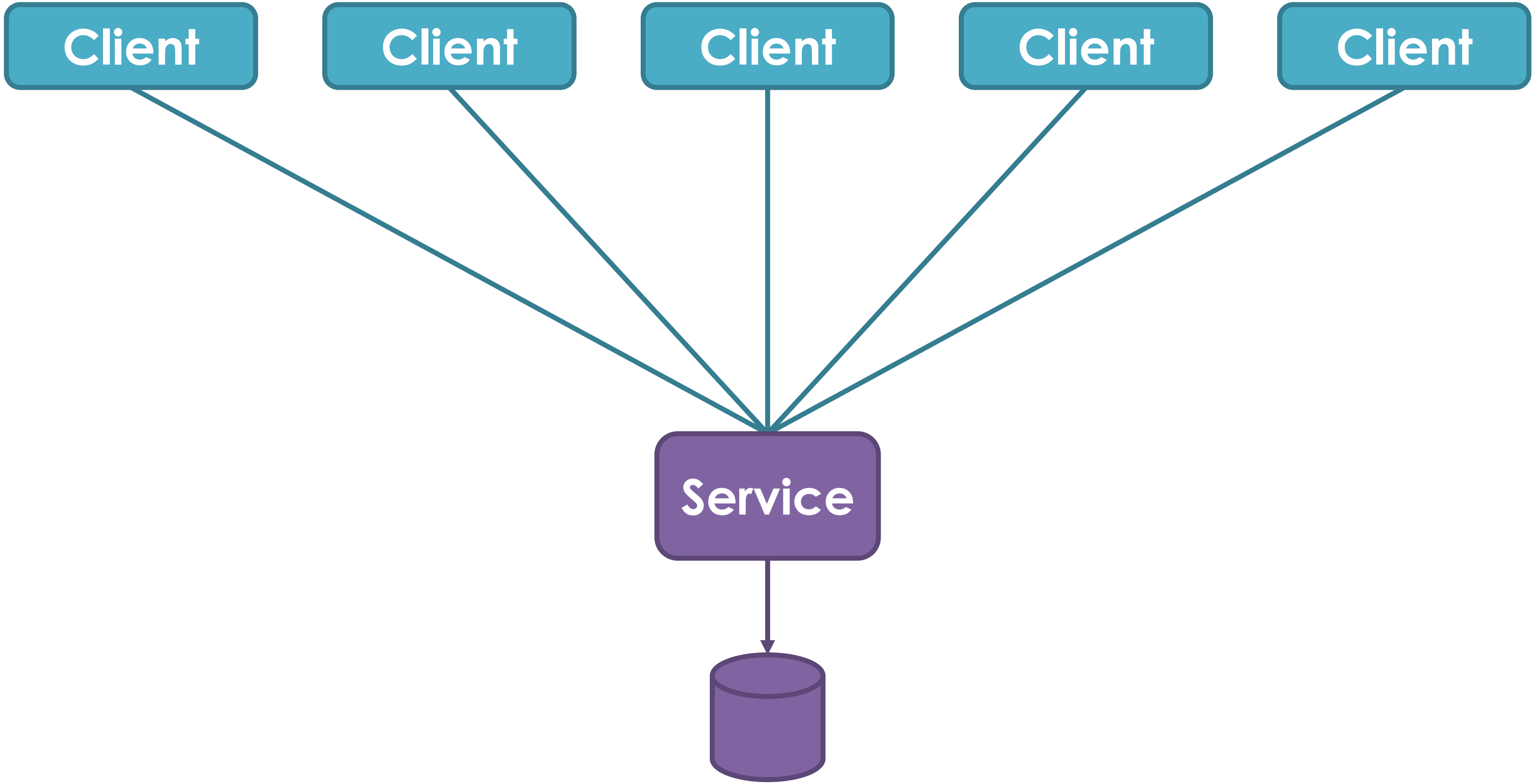
What is the "Architect" really looking for?

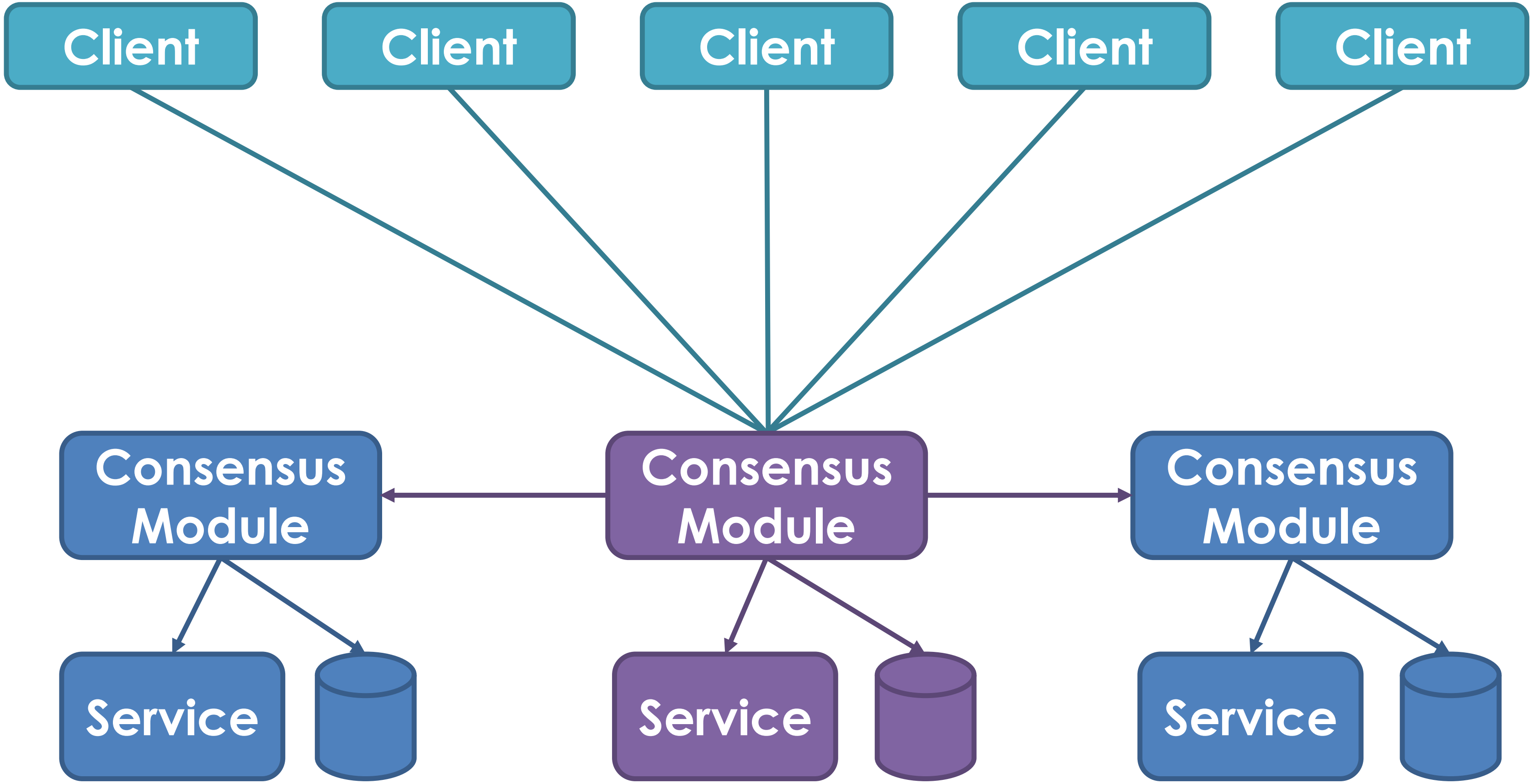
Replicated State Machines

=>

Redundant Deterministic Services


Client
Client
Client
Client
Client
Service


Client
Client
Client
Client
Client
Service


Client
Client
Client
Client
Client
Consensus Module
Consensus Module
Consensus Module
Service
Service
Service

NIO Pain

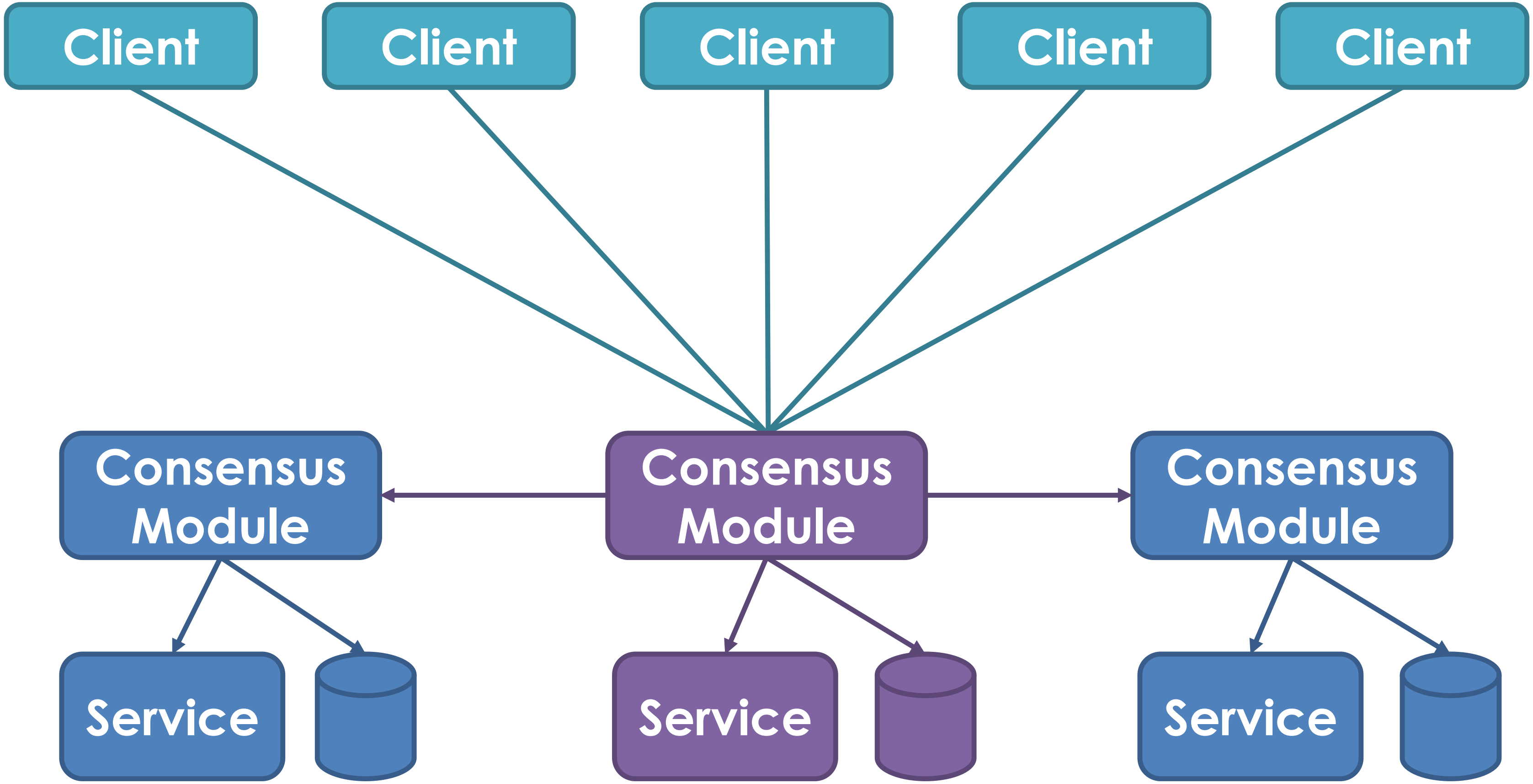
```java
FileChannel channel = null;
try
{
    channel = FileChannel.open(directory.toPath());
}
catch (final IOException ignore)
{
}

if (null != channel)
{
    channel.force(true);
}
```

Directory Sync

```
Files.force(directory.toPath(), true);
```

Performance

Let’s consider the application of an RPC design approach


Client
Client
Client
Client
Client
Consensus Module
Consensus Module
Consensus Module
Service
Service
Service

Should we consider concurrency and parallelism with Replicated State Machines?

“Concurrency is about dealing with lots of things at once. Parallelism is about doing lots of things at once.”

– Rob Pike

1. ***Parallel*** *is the opposite of* ***Serial***
2. ***Concurrent*** *is the opposite of* ***Sequential***
3. ***Vector*** *is the opposite of* ***Scalar***

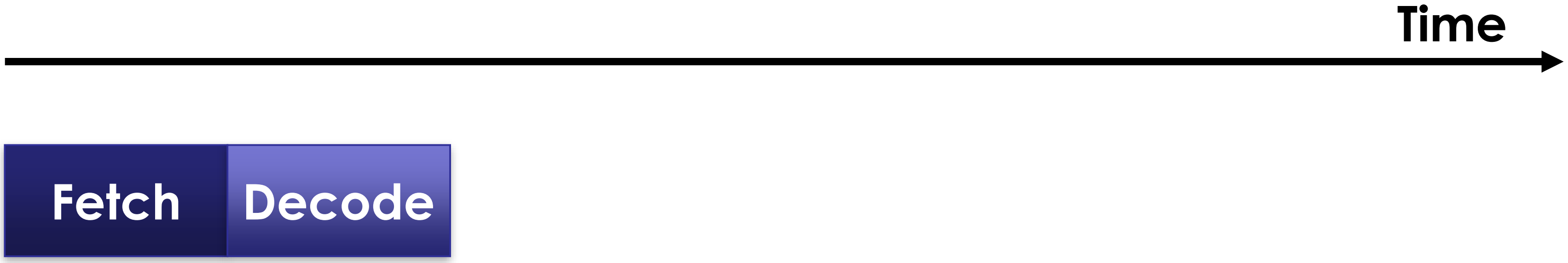
– John Gustafson

Instruction Pipelining

Time

Fetch

Instruction Pipelining

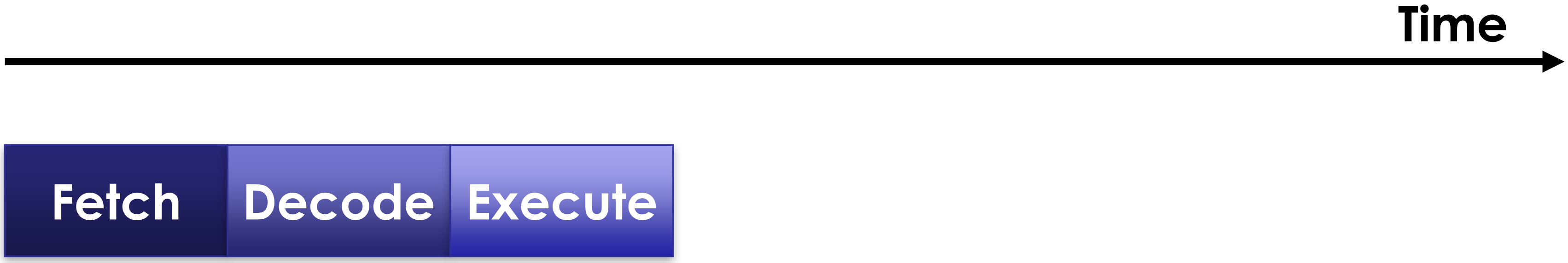


Time

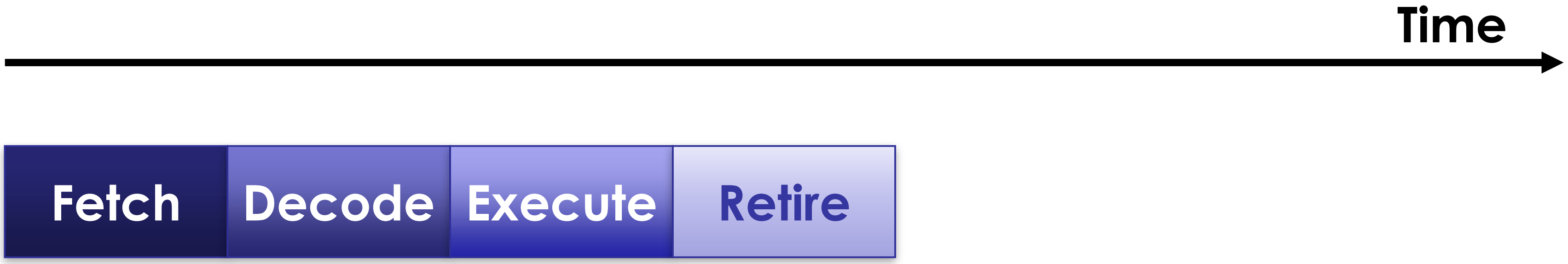
Fetch

Decode

Instruction Pipelining



Instruction Pipelining



Time

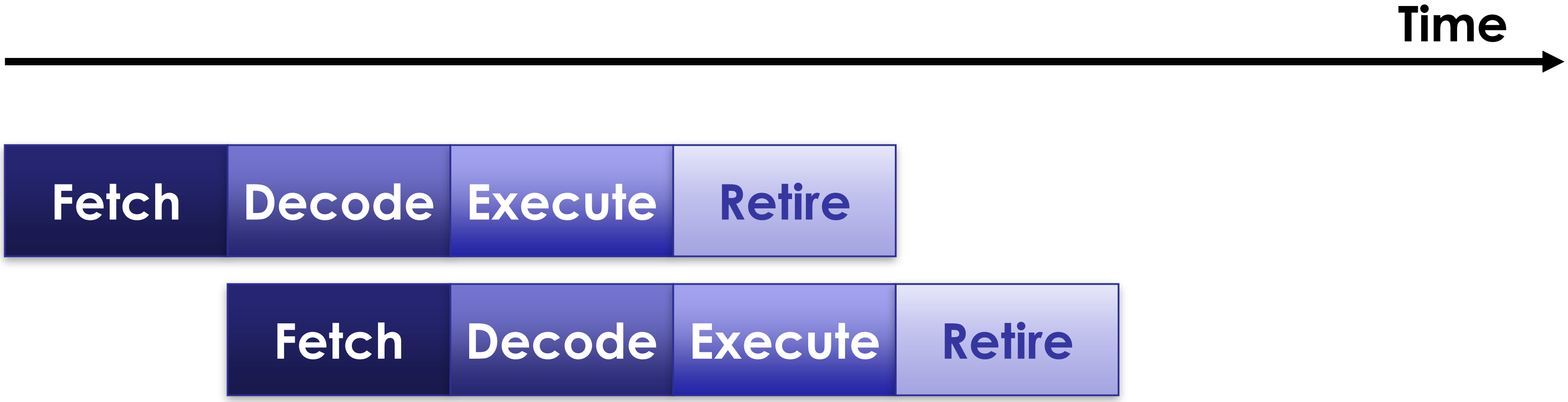
Fetch

Decode

Execute

Retire

Instruction Pipelining

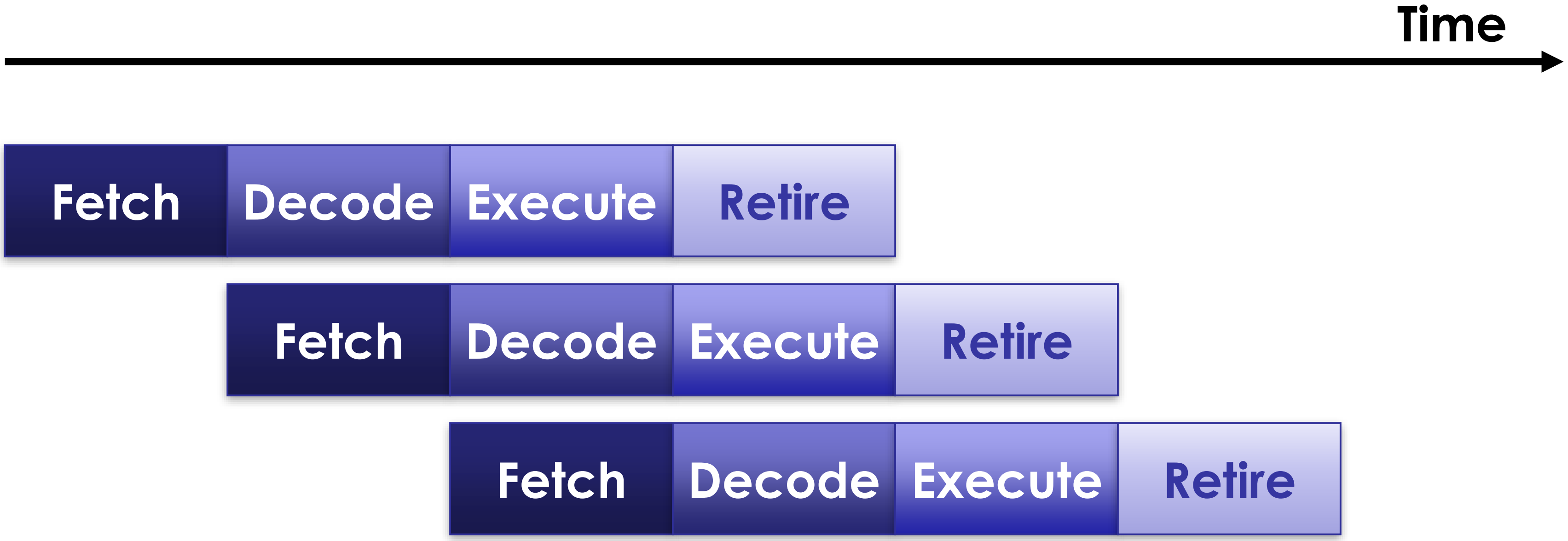


Time

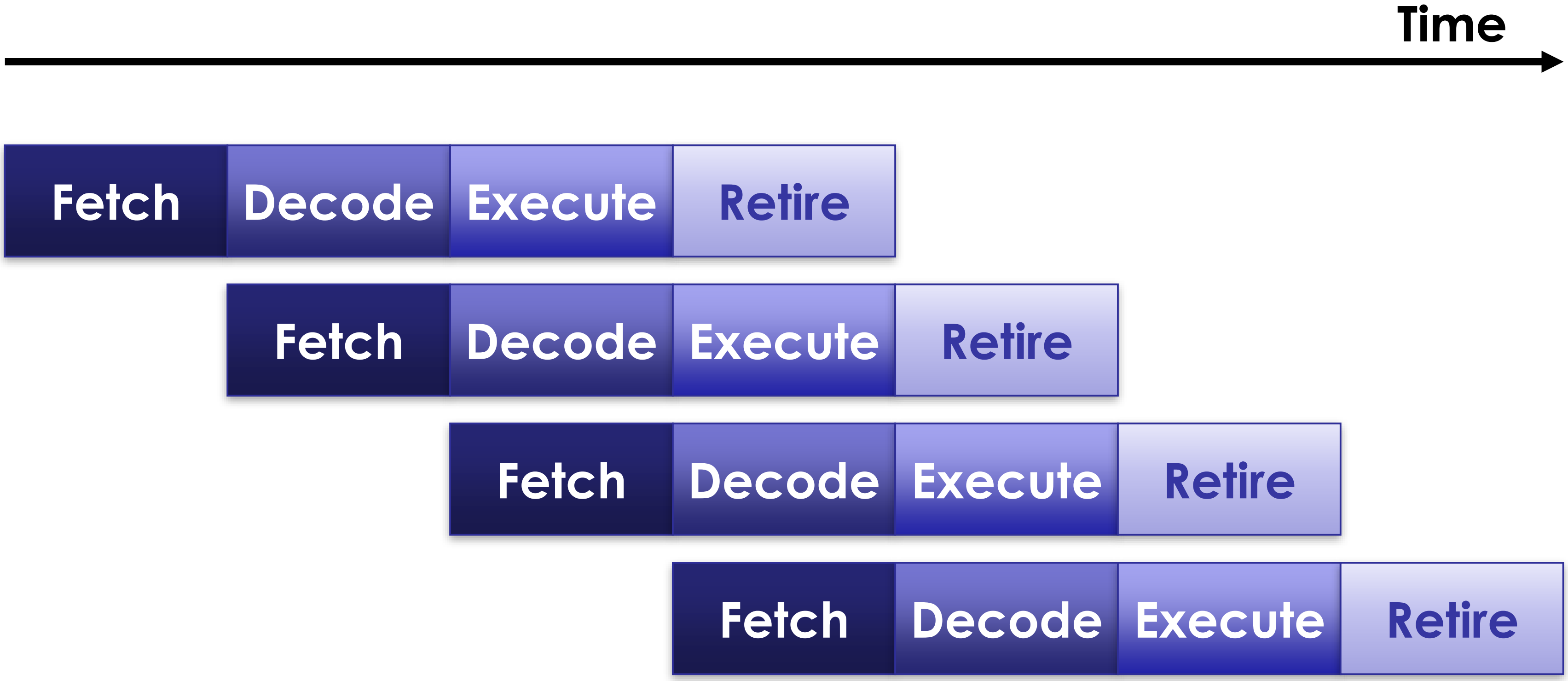
Fetch Decode Execute Retire

Fetch Decode Execute Retire

Instruction Pipelining



Instruction Pipelining

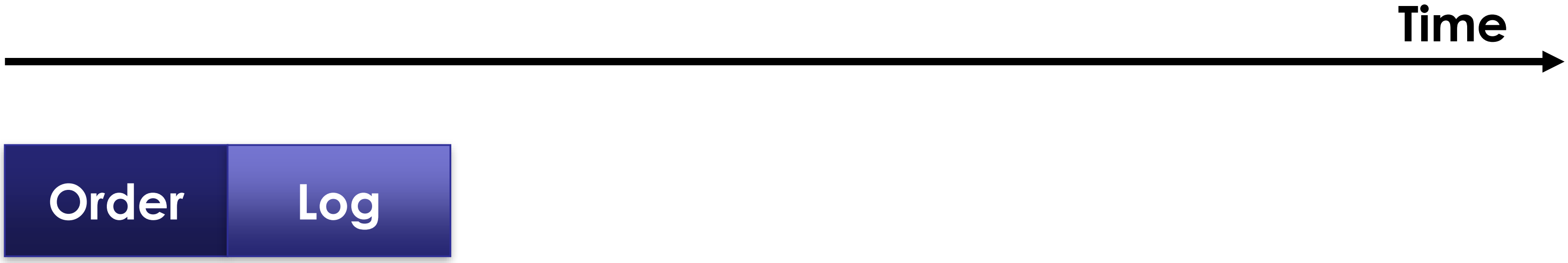


Consensus Pipeline

Time

Order

Consensus Pipeline

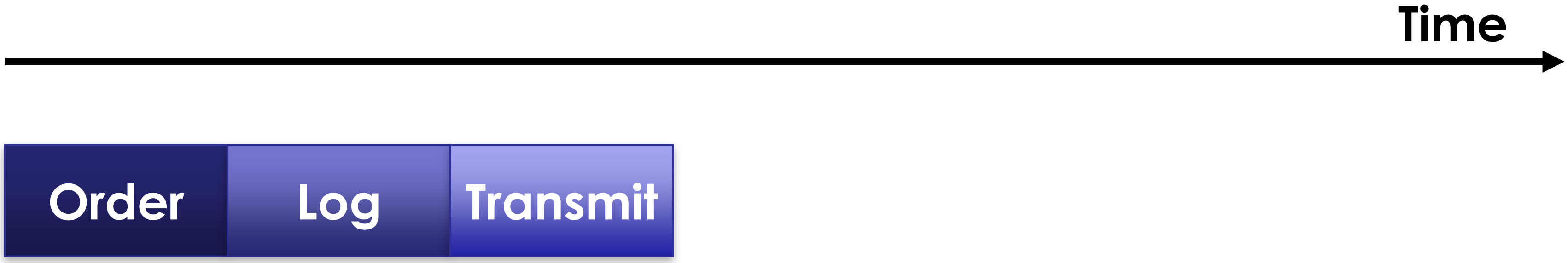


Time

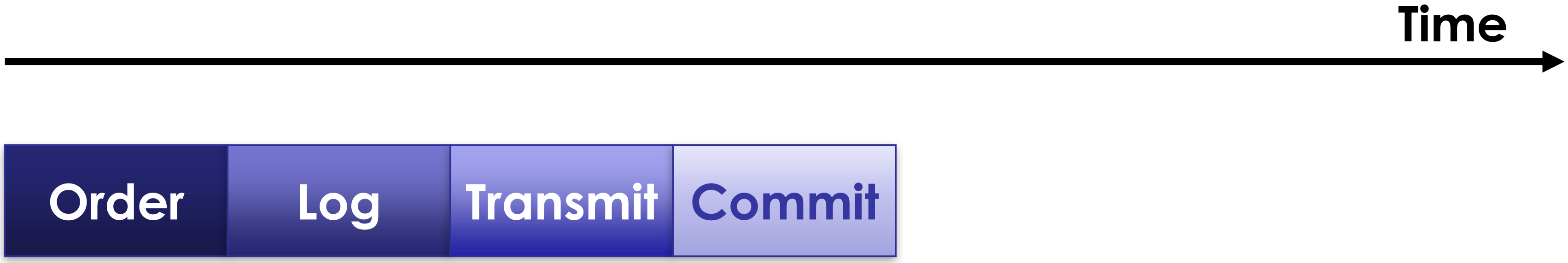
Order

Log

Consensus Pipeline



Consensus Pipeline



Time

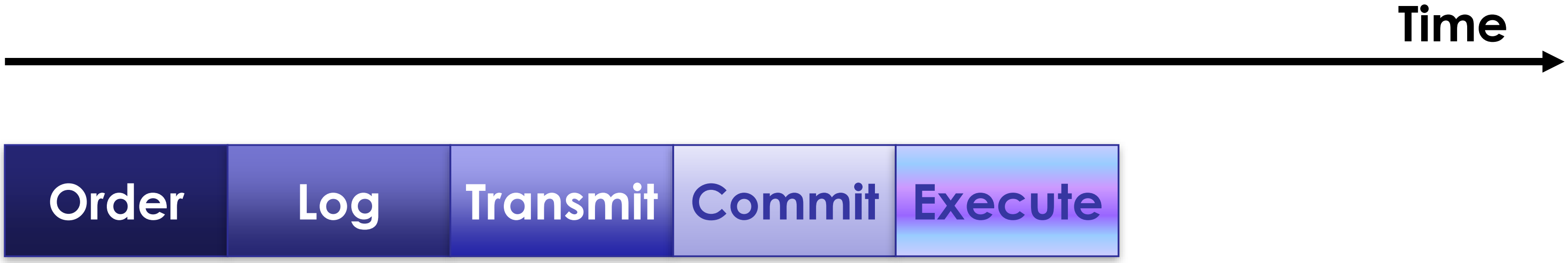
Order

Log

Transmit

Commit

Consensus Pipeline



Time

Order

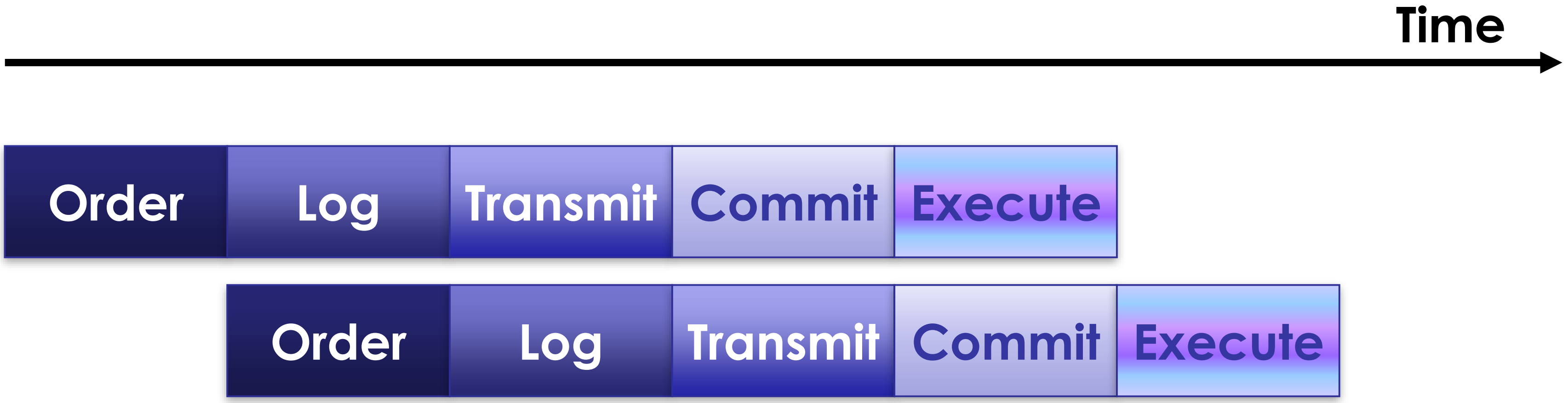
Log

Transmit

Commit

Execute

Consensus Pipeline

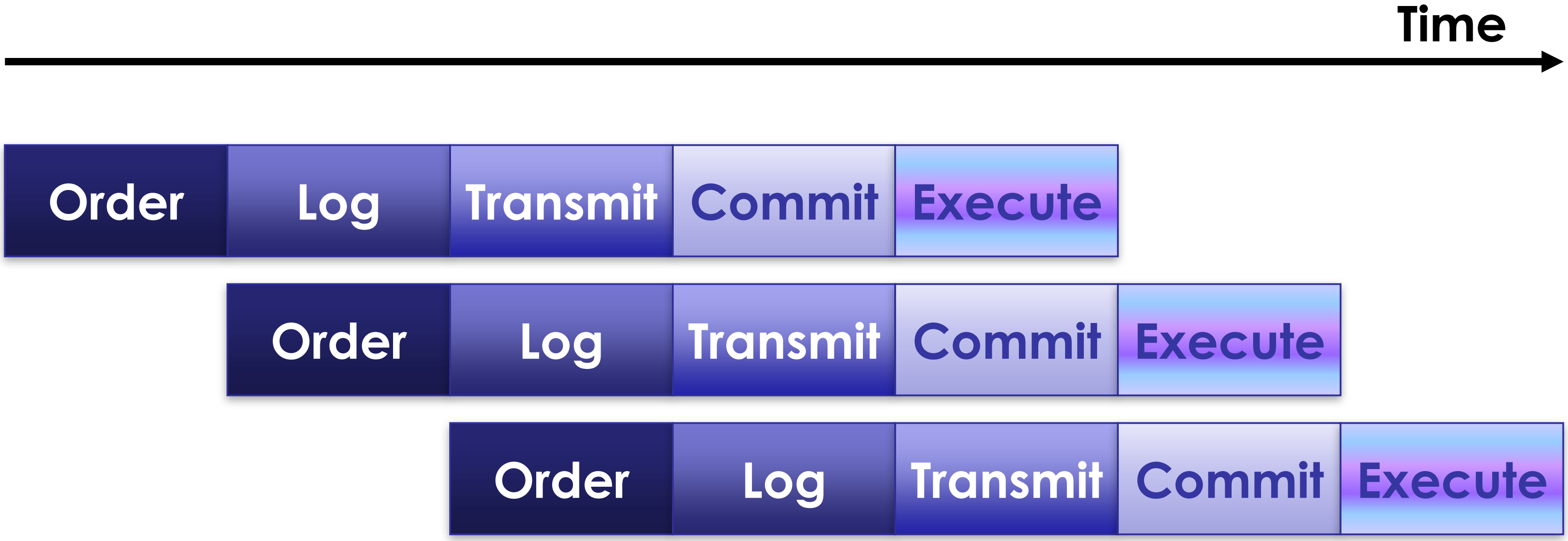


Time

Order | Log | Transmit | Commit | Execute

Order | Log | Transmit | Commit | Execute

Consensus Pipeline

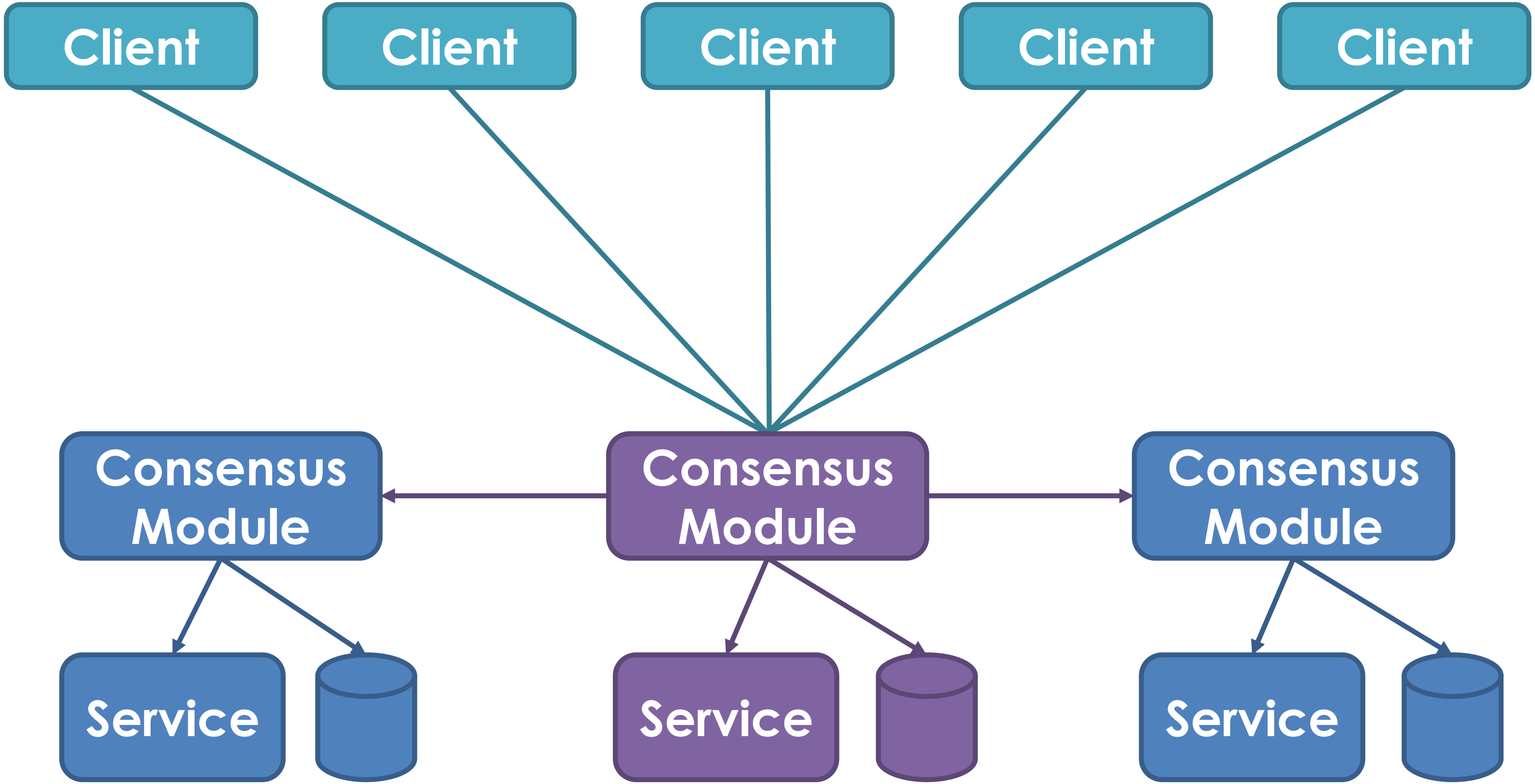


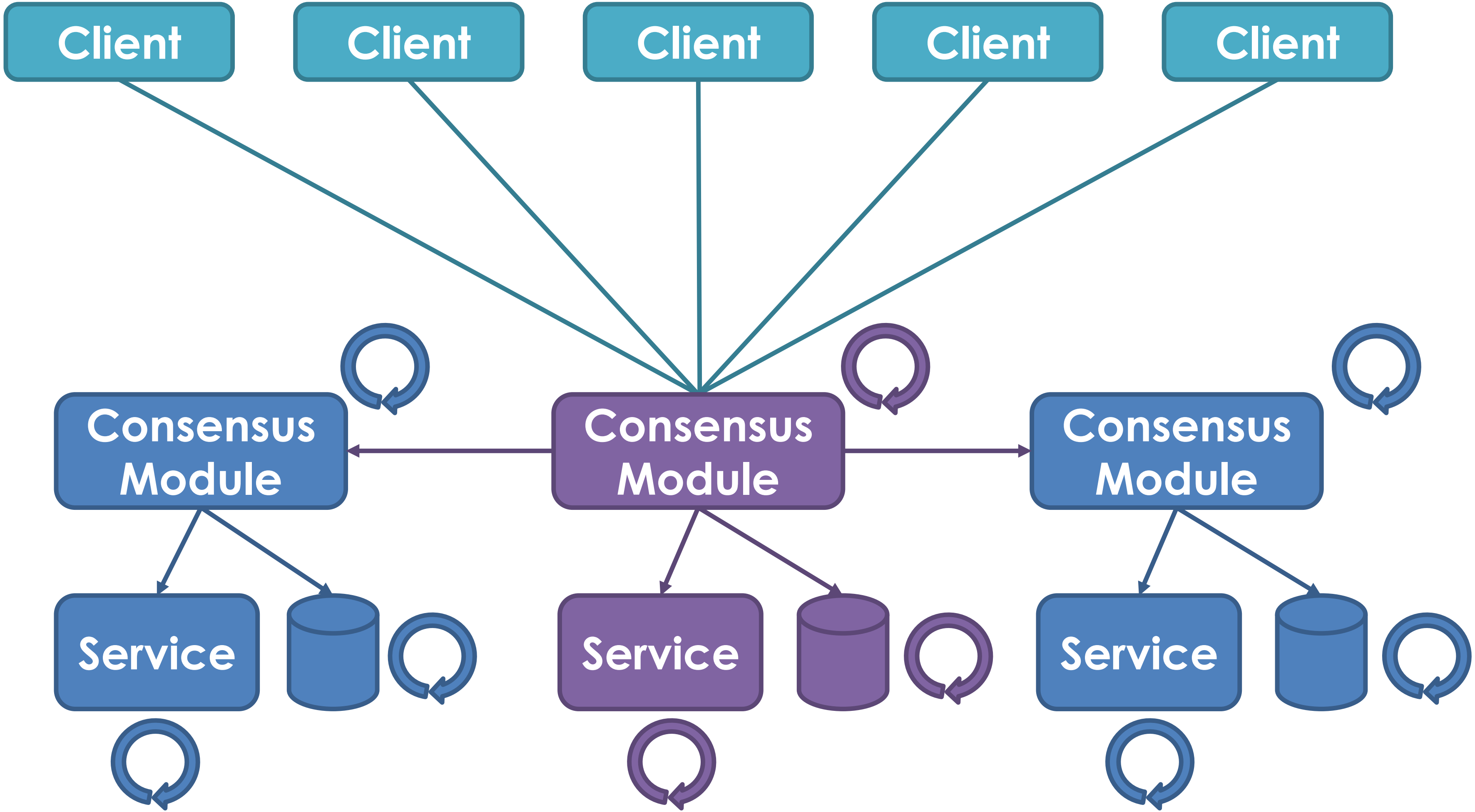
Time

Order Log Transmit Commit Execute

Order Log Transmit Commit Execute

Order Log Transmit Commit Execute


Client
Client
Client
Client
Client
Consensus Module
Consensus Module
Consensus Module
Service
Service
Service


Client
Client
Client
Client
Client
Consensus Module
Consensus Module
Consensus Module
Service
Service
Service

NIO Pain

ByteBuffer byte[] copies

```
ByteBuffer byteBuffer = ByteBuffer.allocate(64 * 1024);

byteBuffer.putInt(index, value);
```

ByteBuffer byte[] copies

```
ByteBuffer byteBuffer = ByteBuffer.allocate(64 * 1024);

byteBuffer.putBytes(index, bytes);
```

ByteBuffer byte[] copies

```
ByteBuffer byteBuf                        ocate(64 * 1024);

byteBuffer                         es);
```

https://bugs.openjdk.java.net/browse/JDK-5029431

2004-04-08 20:46

How can Aeron help?

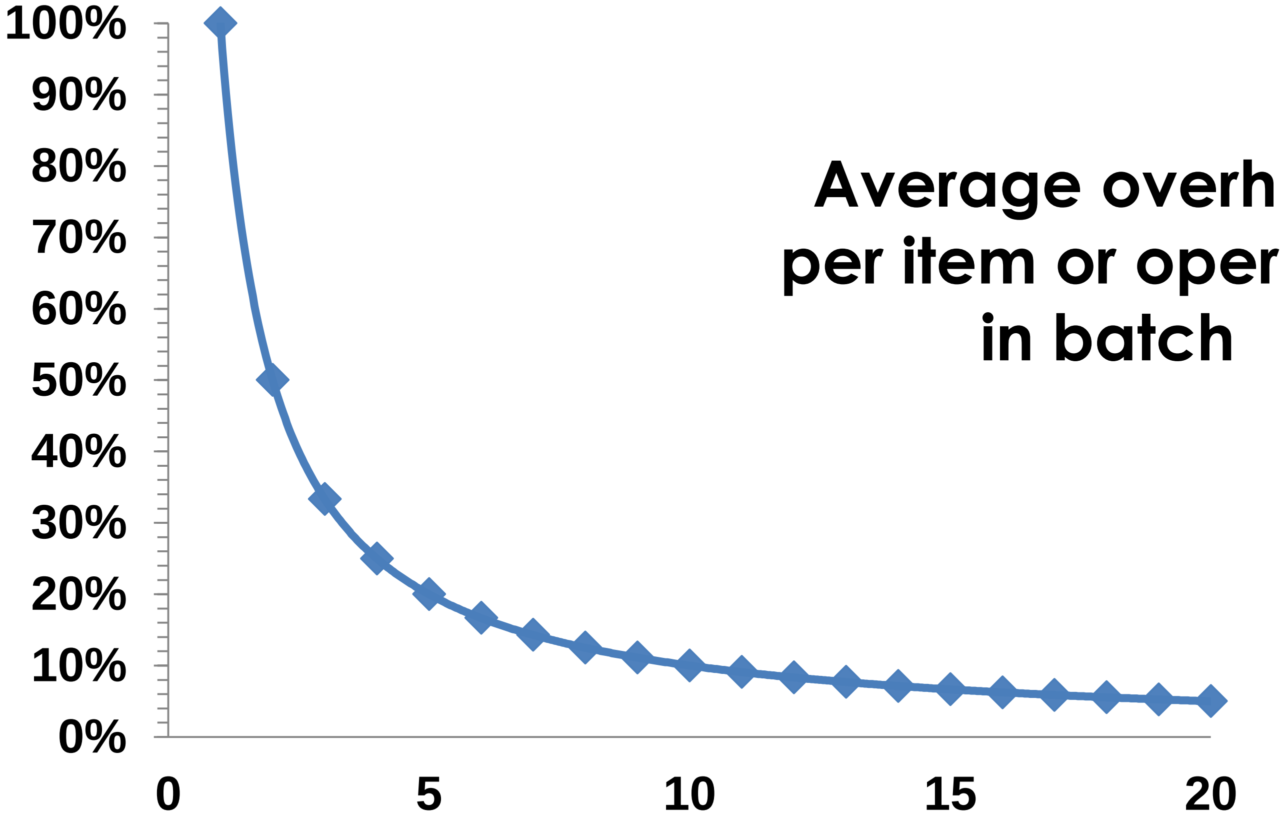
Message Index => Byte Index

Multicast, MDC, and Spy based Messaging

Counters and Bounded Consumption

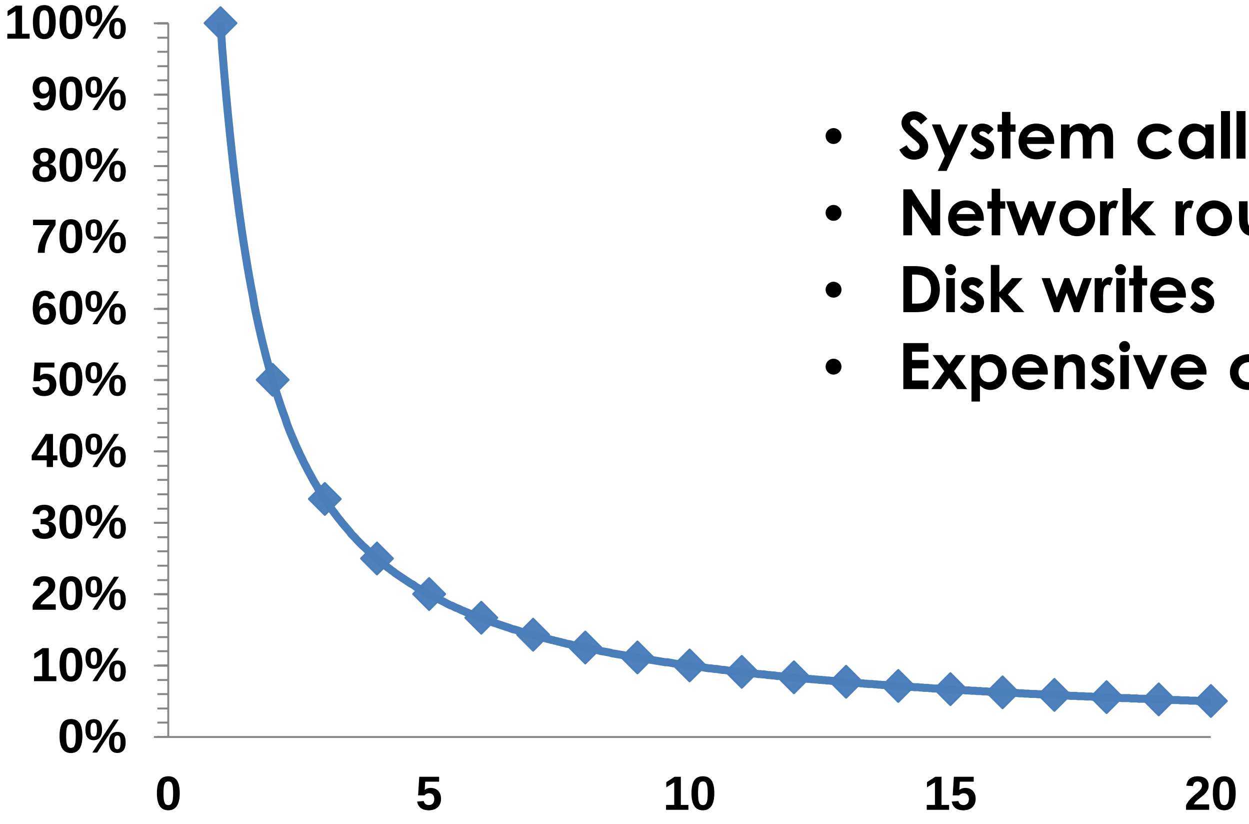
Binary Protocols &
Zero intermediate copies

Batching – Amortising Costs



Average overhead per item or operation in batch

Batching – Amortising Costs



- System calls
- Network round trips
- Disk writes
- Expensive calculations

Interesting Features

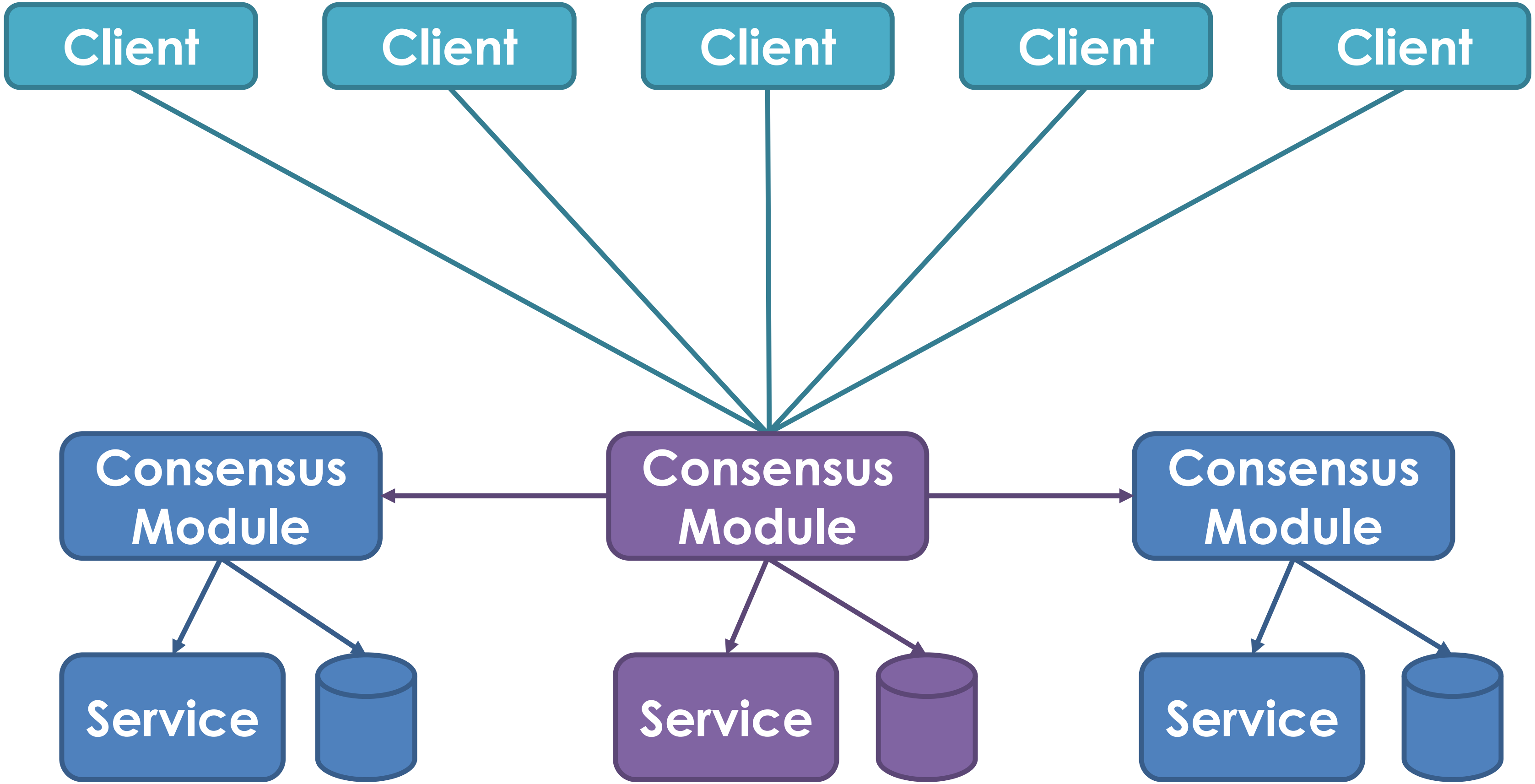
Agents and Threads

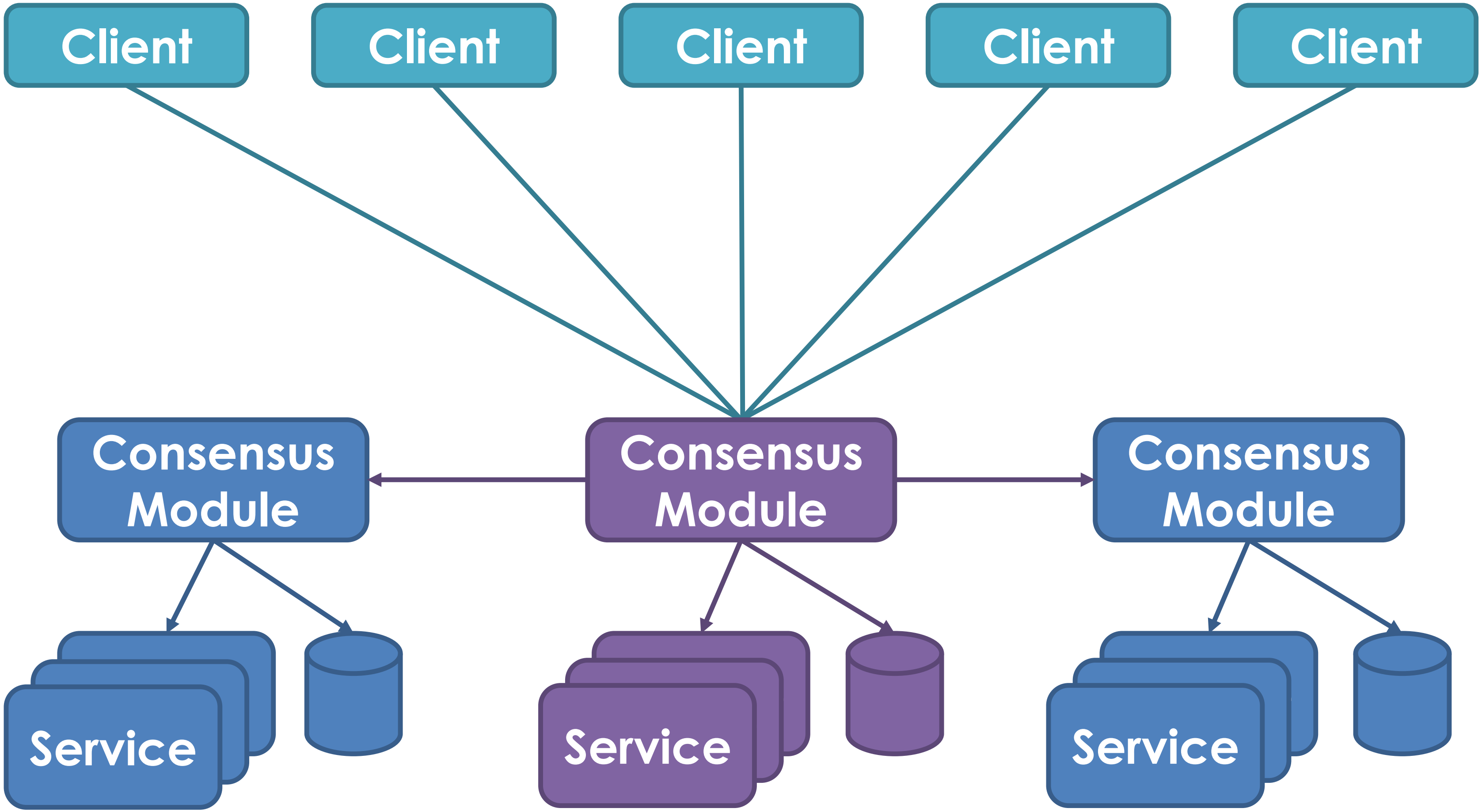
Timers

Back Pressure and Stashed Work

Replay and Snapshots

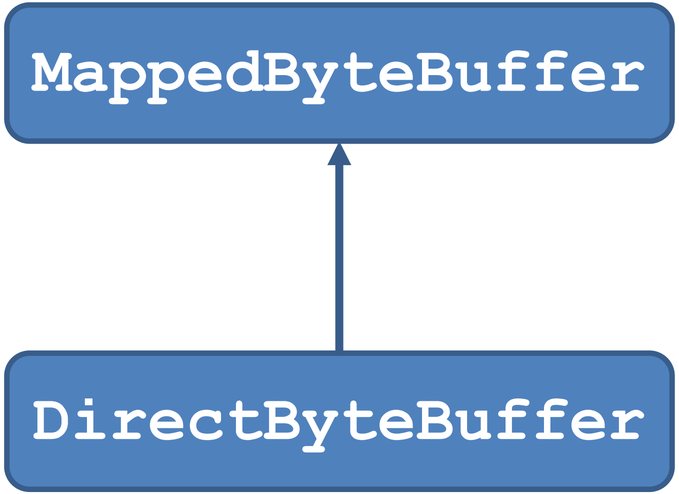
Multiple Services on the same stream


Client
Client
Client
Client
Client
Consensus Module
Consensus Module
Consensus Module
Service
Service
Service


Client
Client
Client
Client
Client
Consensus Module
Consensus Module
Consensus Module
Service
Service
Service

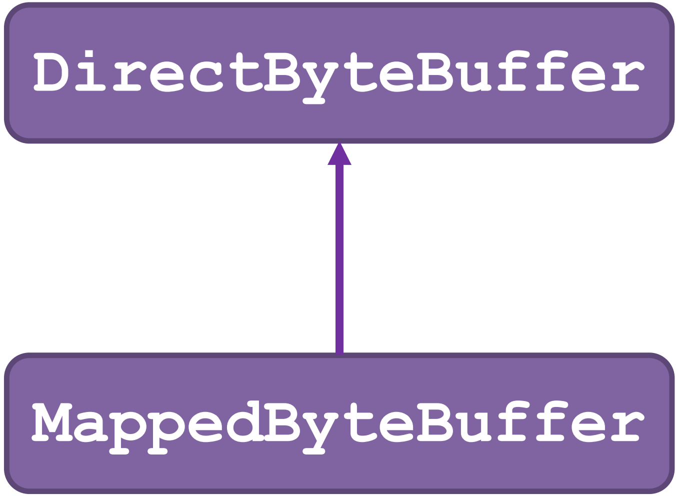
In Closing

NIO Pain



MappedByteBuffer

DirectByteBuffer



DirectByteBuffer

MappedByteBuffer

Do epic shit,
or die trying.

Questions?

https://github.com/real-logic/aeron

Twitter: @mjpt777

“A distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable.”

- Leslie Lamport