Real Logic

ACCELERATING SOFTWARE

Event Log Architectures

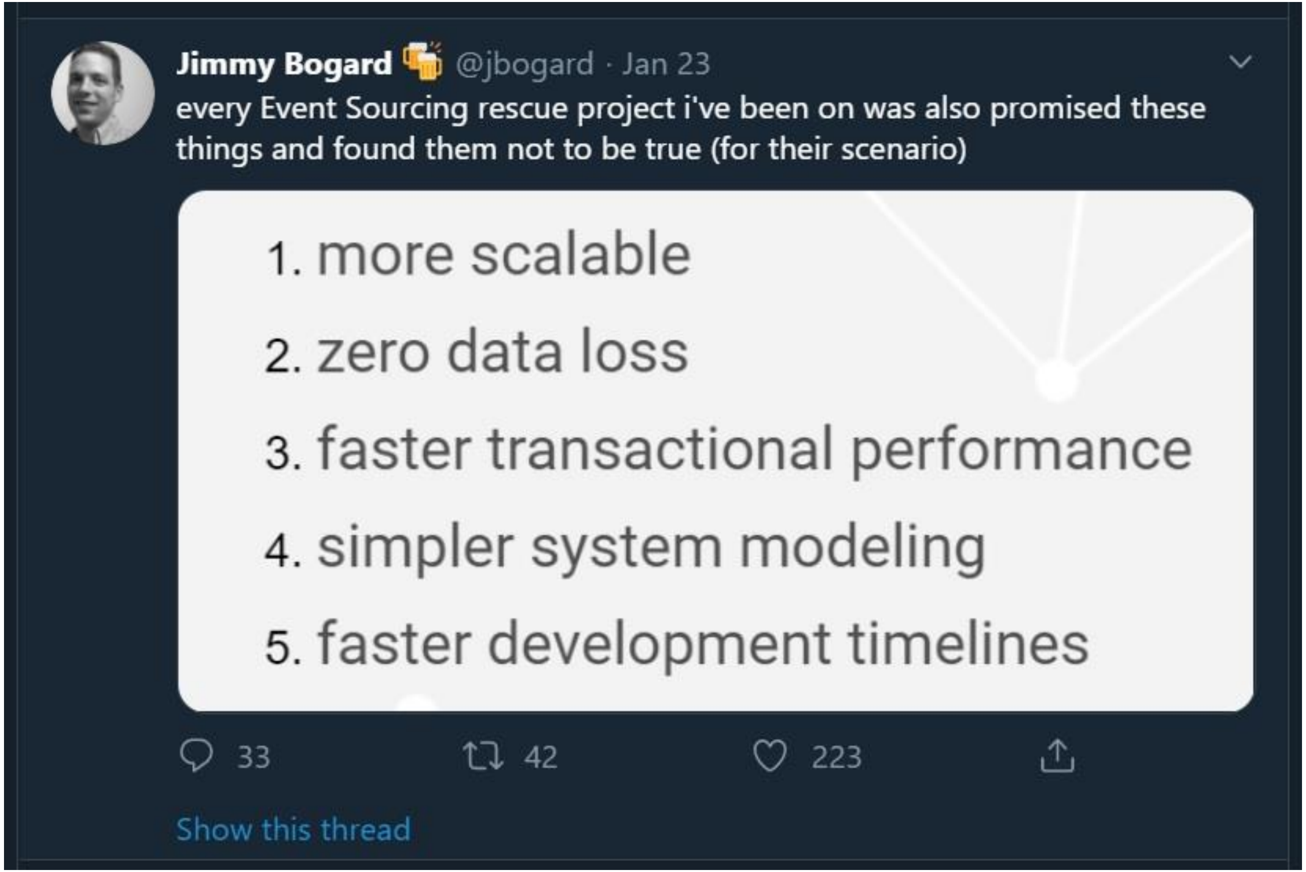
When quality matters

Martin Thompson - @mjpt777

Event Sourcing

Command Sourcing

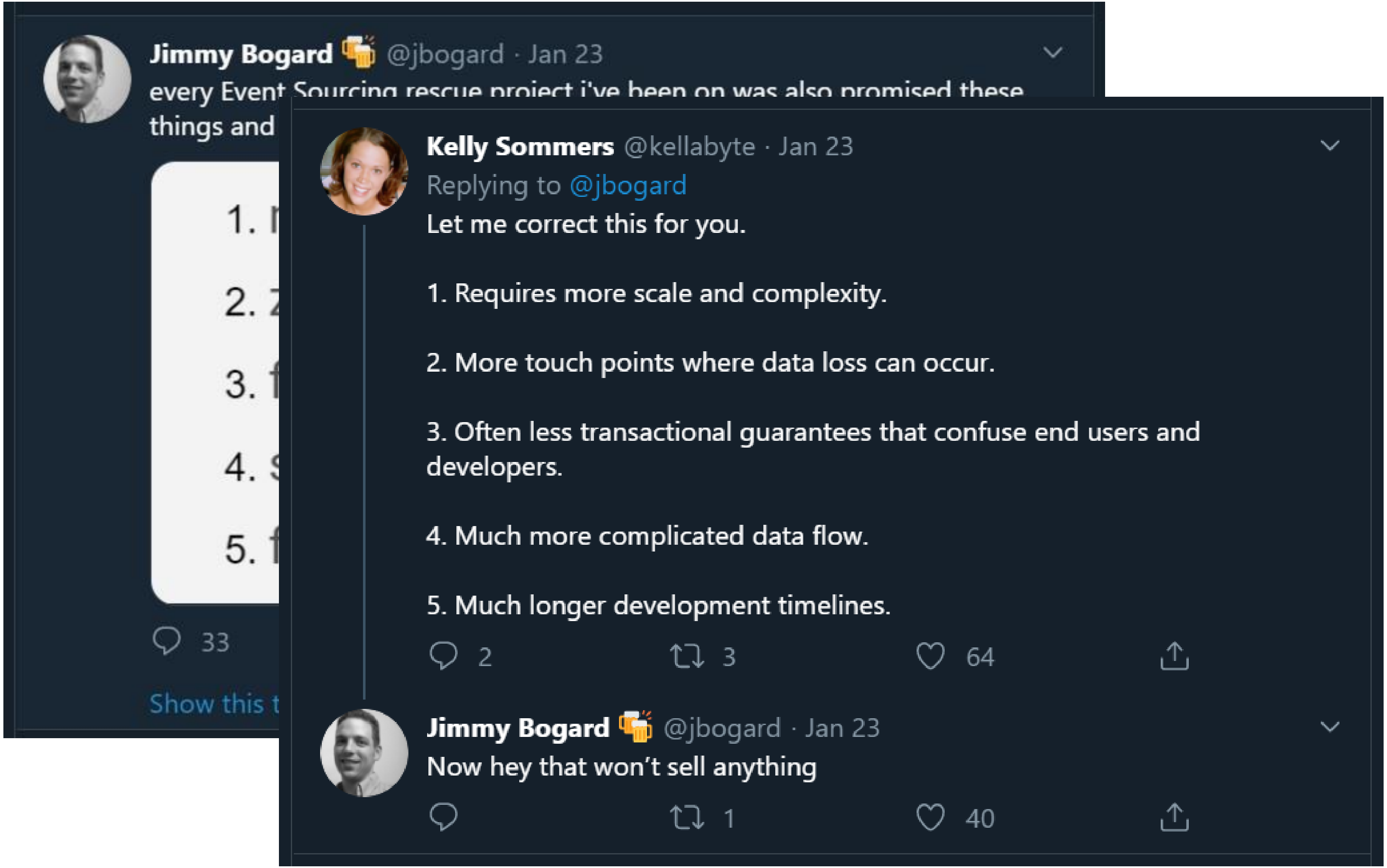
CQRS



Jimmy Bogard 🍻 @jbogard · Jan 23

every Event Sourcing rescue project i've been on was also promised these things and found them not to be true (for their scenario)

1. more scalable
2. zero data loss
3. faster transactional performance
4. simpler system modeling
5. faster development timelines



Jimmy Bogard 🍻 @jbogard · Jan 23

every Event Sourcing rescue project i've been on was also promised these things and

1. r

2. z

3. f

4. s

5. f

33

Show this t

Kelly Sommers @kellabyte · Jan 23

Replying to @jbogard

Let me correct this for you.

1. Requires more scale and complexity.

2. More touch points where data loss can occur.

3. Often less transactional guarantees that confuse end users and developers.

4. Much more complicated data flow.

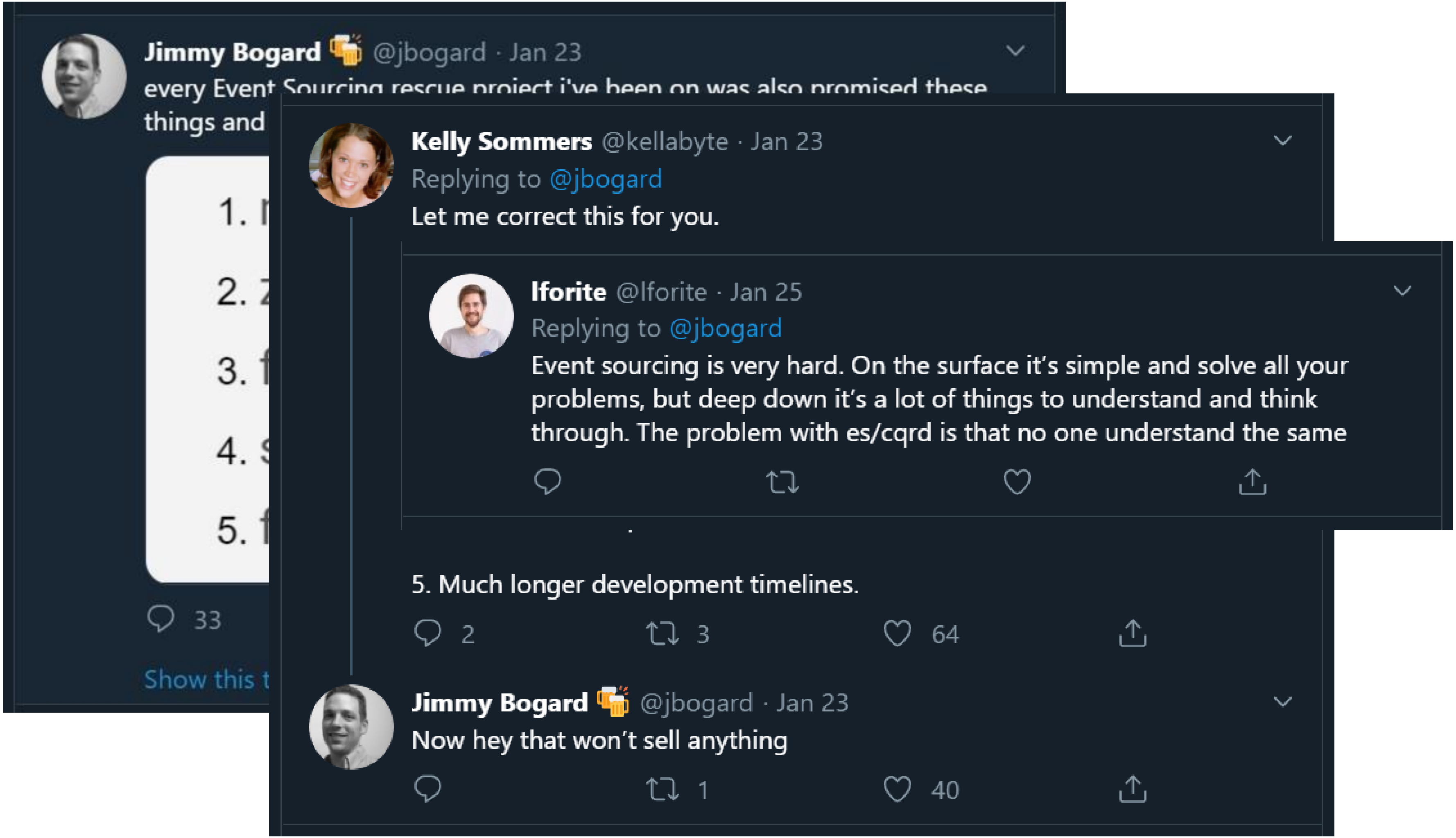
5. Much longer development timelines.

2 3 64

Jimmy Bogard 🍻 @jbogard · Jan 23

Now hey that won’t sell anything

1 40



Jimmy Bogard @jbogard · Jan 23

every Event Sourcing rescue project i've been on was also promised these things and

1. n

2. Z

3. f

4. s

5. f

33

Show this t

Kelly Sommers @kellabyte · Jan 23

Replying to @jbogard

Let me correct this for you.

lforite @lforite · Jan 25

Replying to @jbogard

Event sourcing is very hard. On the surface it’s simple and solve all your problems, but deep down it’s a lot of things to understand and think through. The problem with es/cqrd is that no one understand the same

5. Much longer development timelines.

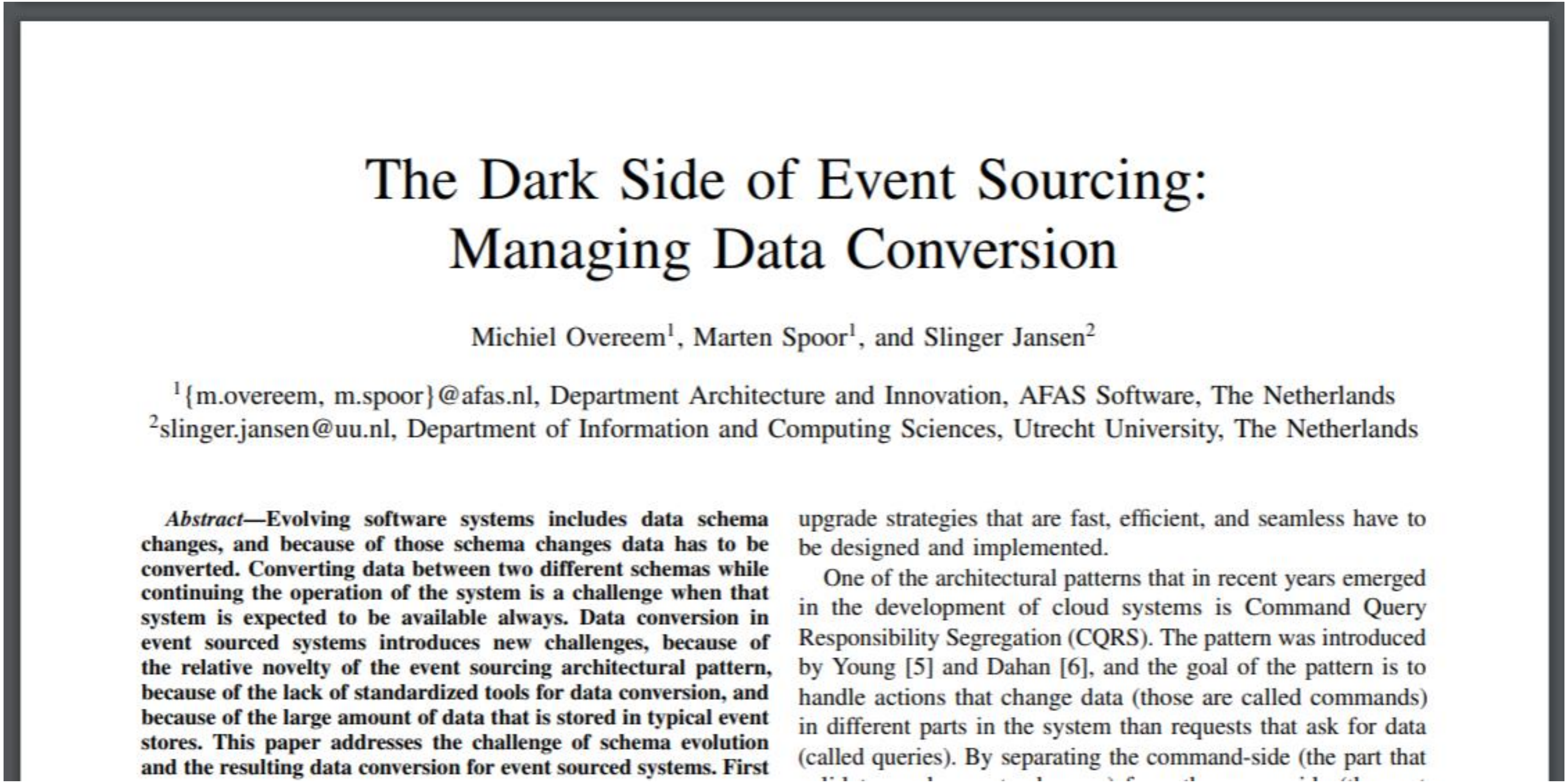
2 3 64

Jimmy Bogard @jbogard · Jan 23

Now hey that won’t sell anything

1 40

Fowler vs Event Store



The Dark Side of Event Sourcing: Managing Data Conversion

Michiel Overeem[1], Marten Spoor[1], and Slinger Jansen[2]

[1]{m.overeem, m.spoor}@afas.nl, Department Architecture and Innovation, AFAS Software, The Netherlands
[2]slinger.jansen@uu.nl, Department of Information and Computing Sciences, Utrecht University, The Netherlands

***Abstract*—Evolving software systems includes data schema changes, and because of those schema changes data has to be converted. Converting data between two different schemas while continuing the operation of the system is a challenge when that system is expected to be available always. Data conversion in event sourced systems introduces new challenges, because of the relative novelty of the event sourcing architectural pattern, because of the lack of standardized tools for data conversion, and because of the large amount of data that is stored in typical event stores. This paper addresses the challenge of schema evolution and the resulting data conversion for event sourced systems. First**

upgrade strategies that are fast, efficient, and seamless have to be designed and implemented.

One of the architectural patterns that in recent years emerged in the development of cloud systems is Command Query Responsibility Segregation (CQRS). The pattern was introduced by Young [5] and Dahan [6], and the goal of the pattern is to handle actions that change data (those are called commands) in different parts in the system than requests that ask for data (called queries). By separating the command-side (the part that

Events, Logs,
& State Machines

State Machines

Input × State → State

State Machines

Input × State → State

Input × State → Output

Replicated State Machines

Ordered Inputs

+

Deterministic Execution

=>

Same State & Outputs

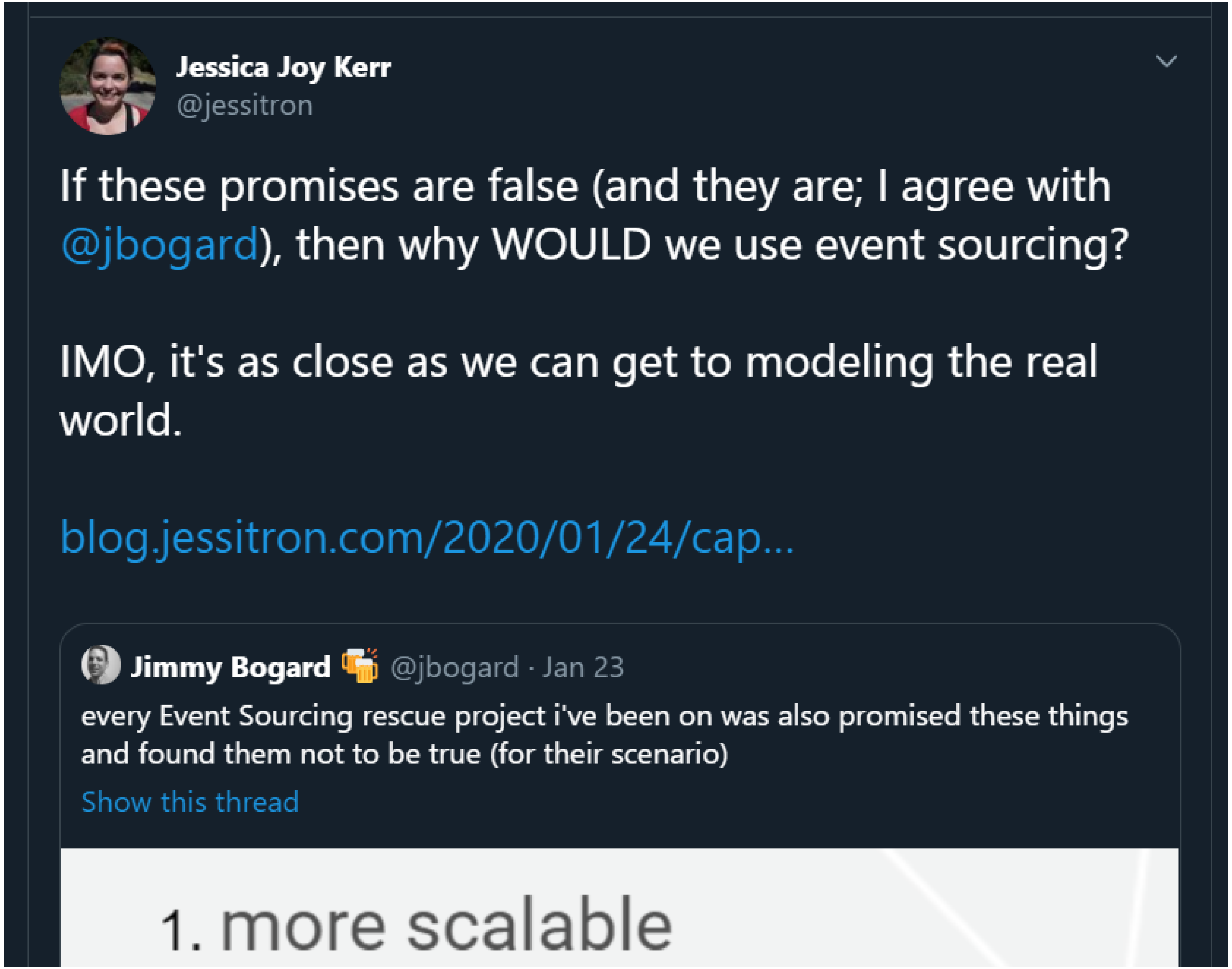
Event Log

(Ordered Inputs)

Inputs vs Changes vs Outputs

Don’t throw away data without good reason!

Domain Models



Jessica Joy Kerr
@jessitron

If these promises are false (and they are; I agree with @jbogard), then why WOULD we use event sourcing?

IMO, it's as close as we can get to modeling the real world.

blog.jessitron.com/2020/01/24/cap…

Jimmy Bogard 🍻 @jbogard · Jan 23

every Event Sourcing rescue project i've been on was also promised these things and found them not to be true (for their scenario)

Show this thread

1. more scalable

Modelling free from the restrictions of data stores

Development Considerations

Development Considerations

1. **Choose only deterministic algorithms**

Development Considerations

1. Choose only deterministic algorithms
2. Validate Inputs before mutation

Development Considerations

1. Choose only deterministic algorithms
2. Validate Inputs before mutation
3. Avoid Head-of-Line blocking

Development Considerations

1. Choose only deterministic algorithms
2. Validate Inputs before mutation
3. Avoid Head-of-Line blocking
4. Version messages & extend via options

Snapshots

Snapshots

1. Separate model from implementation

Snapshots

1. Separate model from implementation
2. Choose an efficient & compact codec

Snapshots

1. Separate model from implementation
2. Choose an efficient & compact codec
3. Version everything

Snapshots

1. Separate model from implementation
2. Choose an efficient & compact codec
3. Version everything
4. Store in a form that is easy to distribute

Time & Timers

Time & Timers

1. Don’t read the system clock

Time & Timers

1. Don’t read the system clock
2. Don’t read the system clock!

Time & Timers

1. Don’t read the system clock
2. Don’t read the system clock!
3. Timestamp events / messages

Time & Timers

1. Don't read the system clock
2. Don't read the system clock!
3. Timestamp events / messages
4. Manage timers centrally and reliably

Integration

Integration

1. Adapt other systems with gateways

Integration

1. Adapt other systems with gateways
2. Treat RDBMSs like a ledger

Integration

1. Adapt other systems with gateways
2. Treat RDBMSs like a ledger
3. Negotiate shared state on start-up

Integration

1. Adapt other systems with gateways
2. Treat RDBMSs like a ledger
3. Negotiate shared state on start-up
4. Track responses with timers

Quality Attributes

Non-Functional Requirements?

nonfuncational

adjective \ nän-ˈfəŋ(k)-shnəl \

: having no function : serving or performing no useful purpose

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

nonfuncational

adjective \ nän-ˈfəŋ(k)-shnəl \

: having no function : serving or performing no useful purpose

: not performing or able to perform a regular function

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

Words Matter

Fault Tolerance

Fault Tolerance

Primary / Secondary

vs

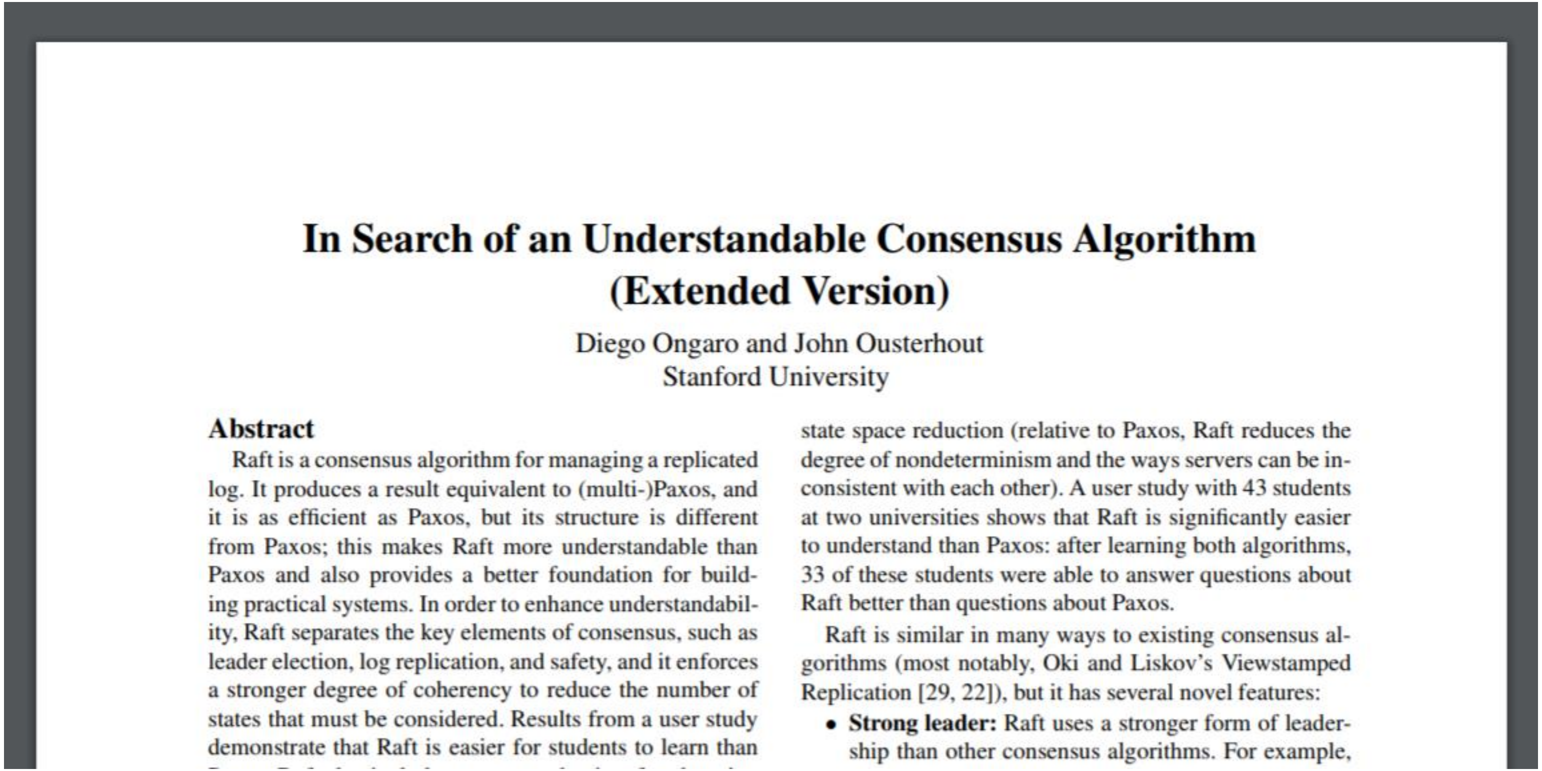
Consensus

Leslie Lamport - Paxos

Barbara Liskov - Viewstamp Replication

Ken Birman - Virtual Synchrony

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

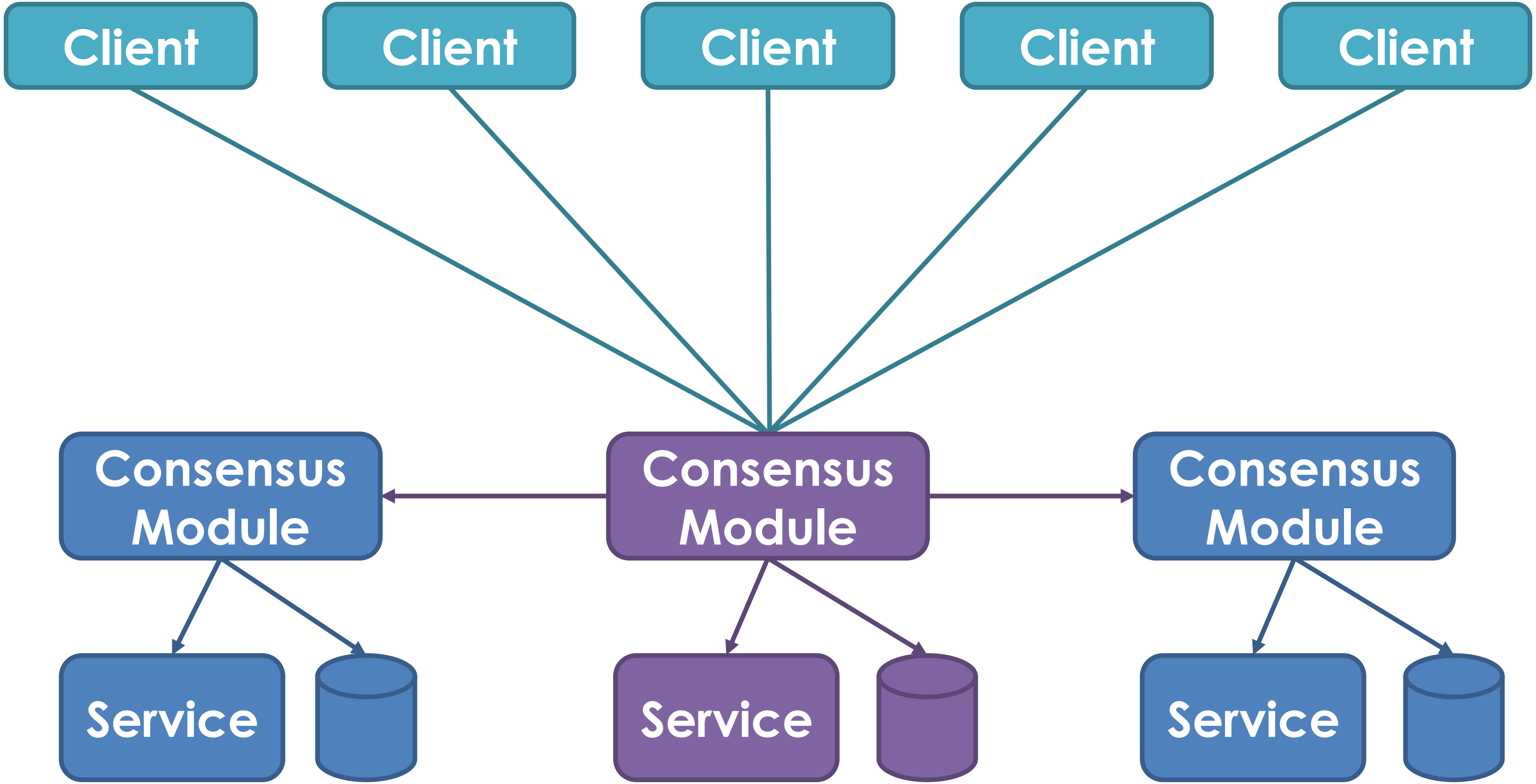
state space reduction (relative to Paxos, Raft reduces the degree of nondeterminism and the ways servers can be inconsistent with each other). A user study with 43 students at two universities shows that Raft is significantly easier to understand than Paxos: after learning both algorithms, 33 of these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

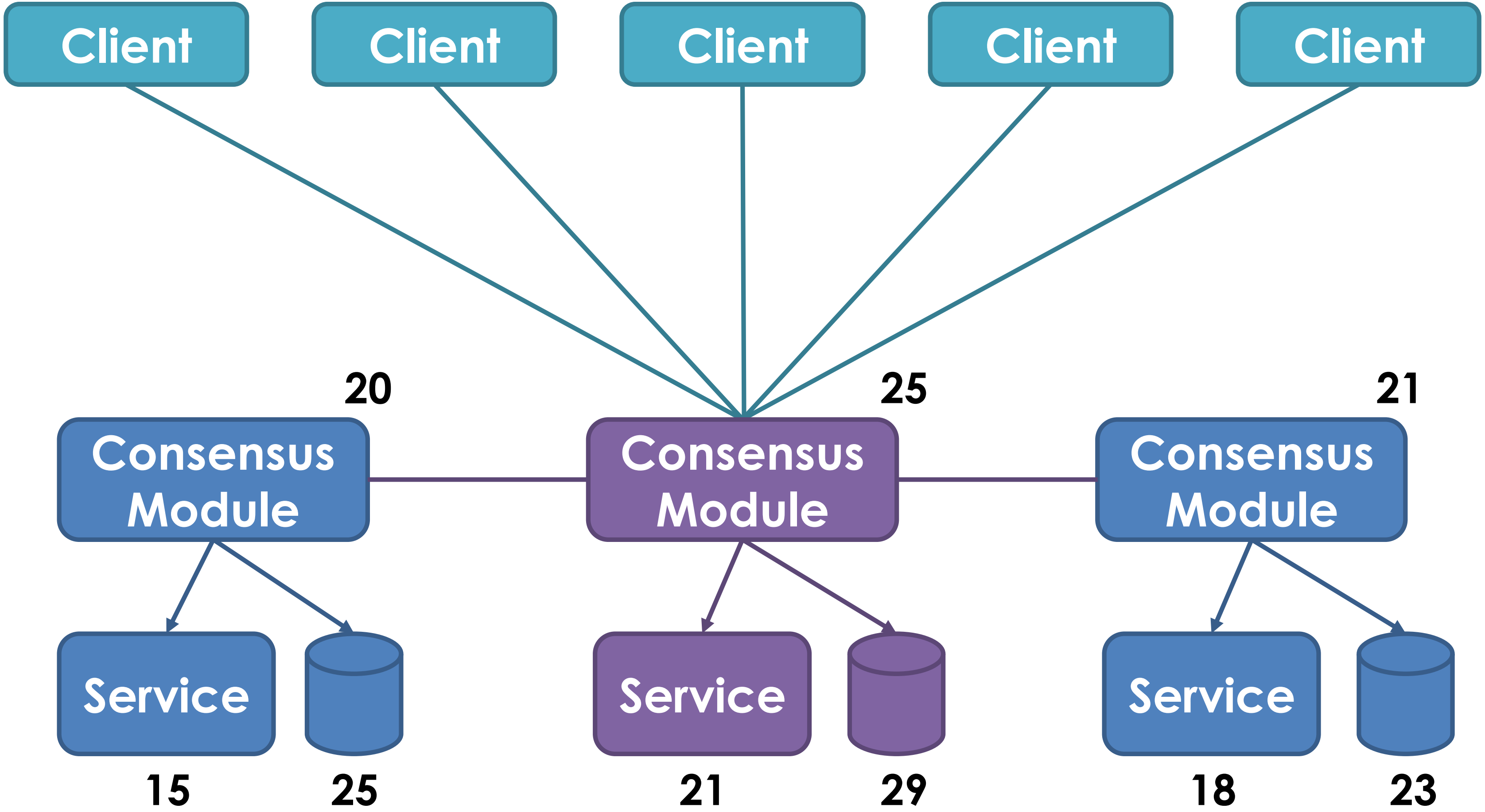
Raft is similar in many ways to existing consensus algorithms (most notably, Oki and Liskov's Viewstamped Replication [29, 22]), but it has several novel features:

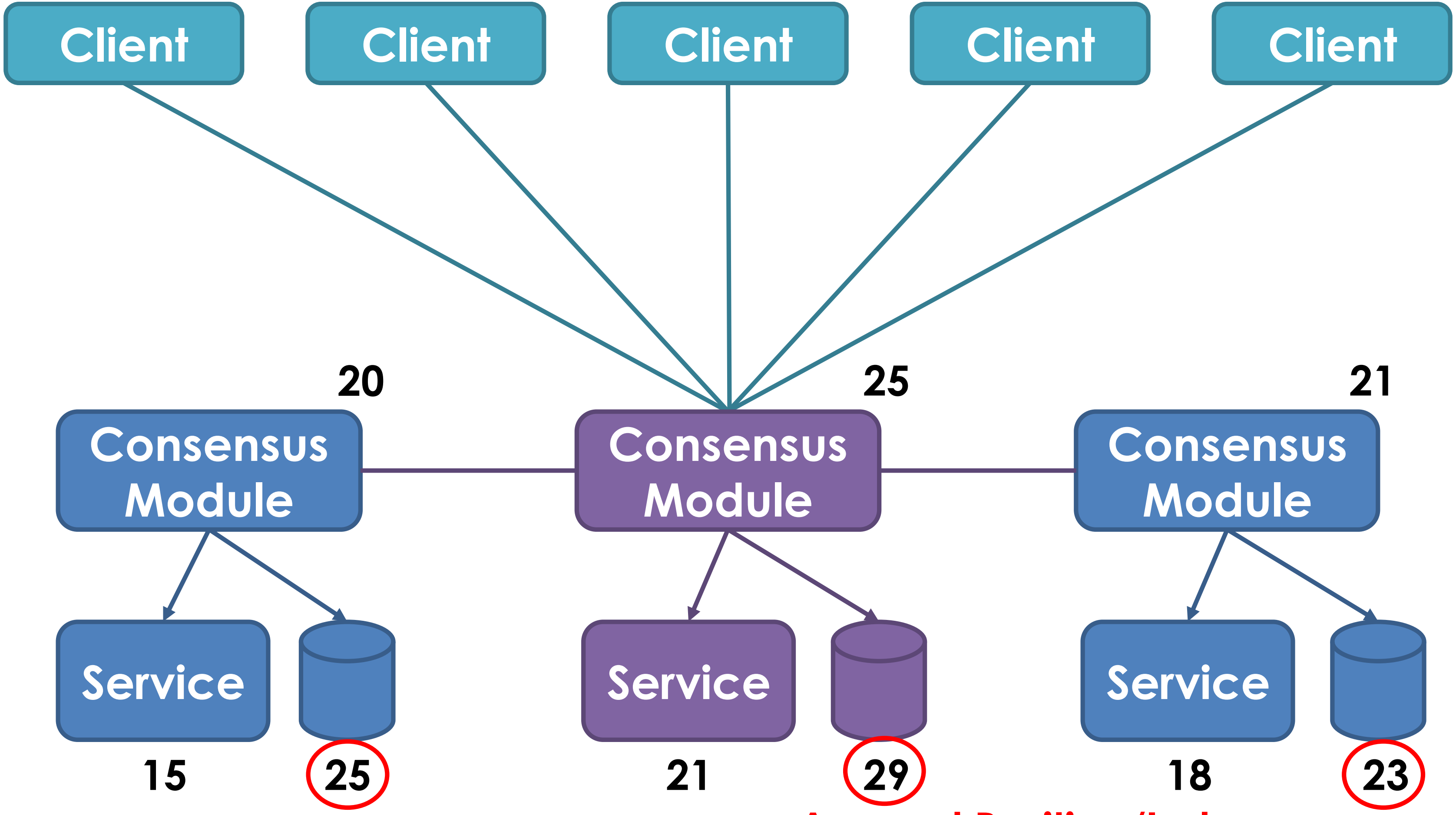
- **Strong leader:** Raft uses a stronger form of leadership than other consensus algorithms. For example,

Raft Safety Guarantees

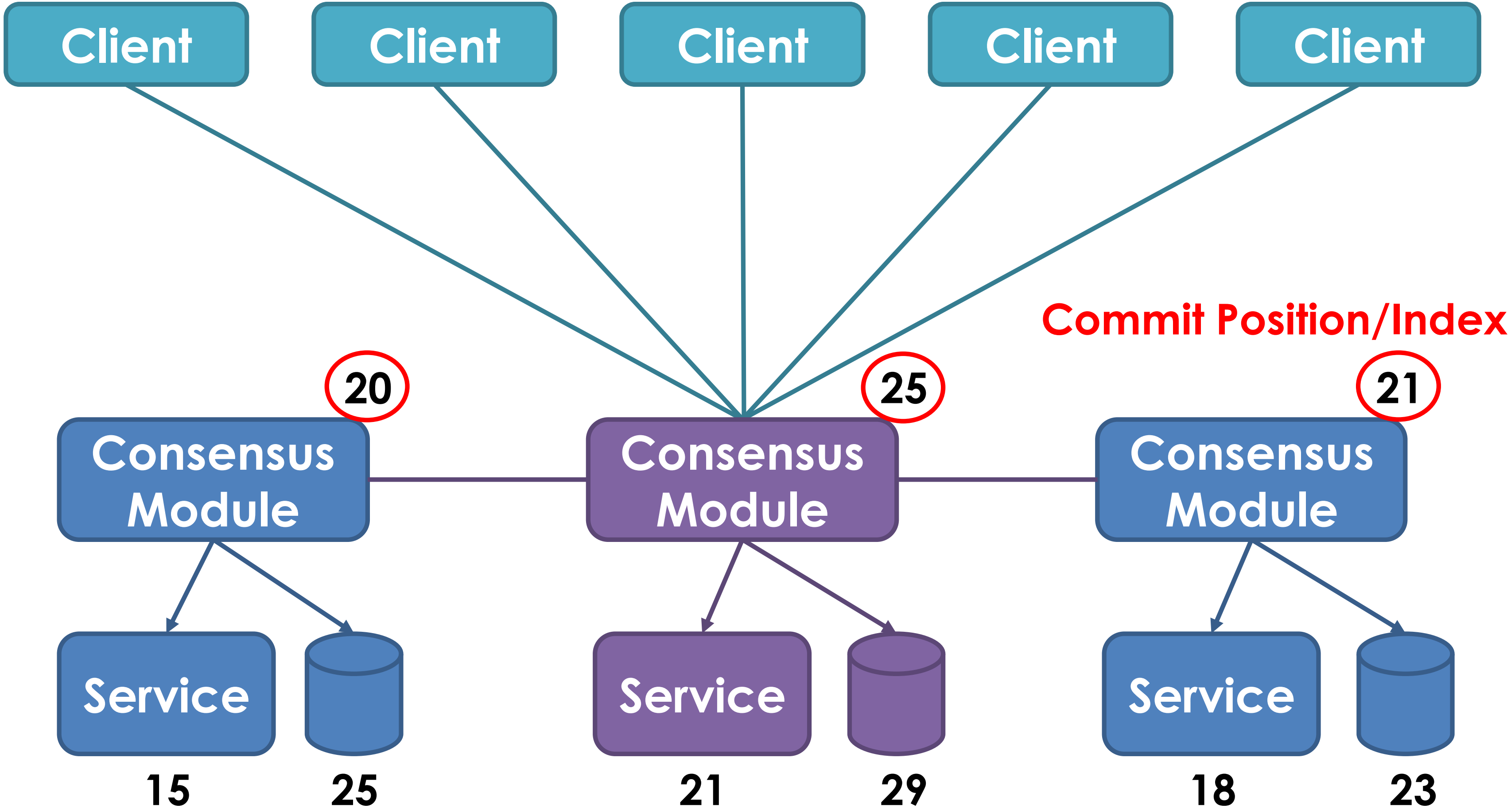
- Election Safety
- Leader Append-Only
- Log Matching
- Leader Completeness
- State Machine Safety

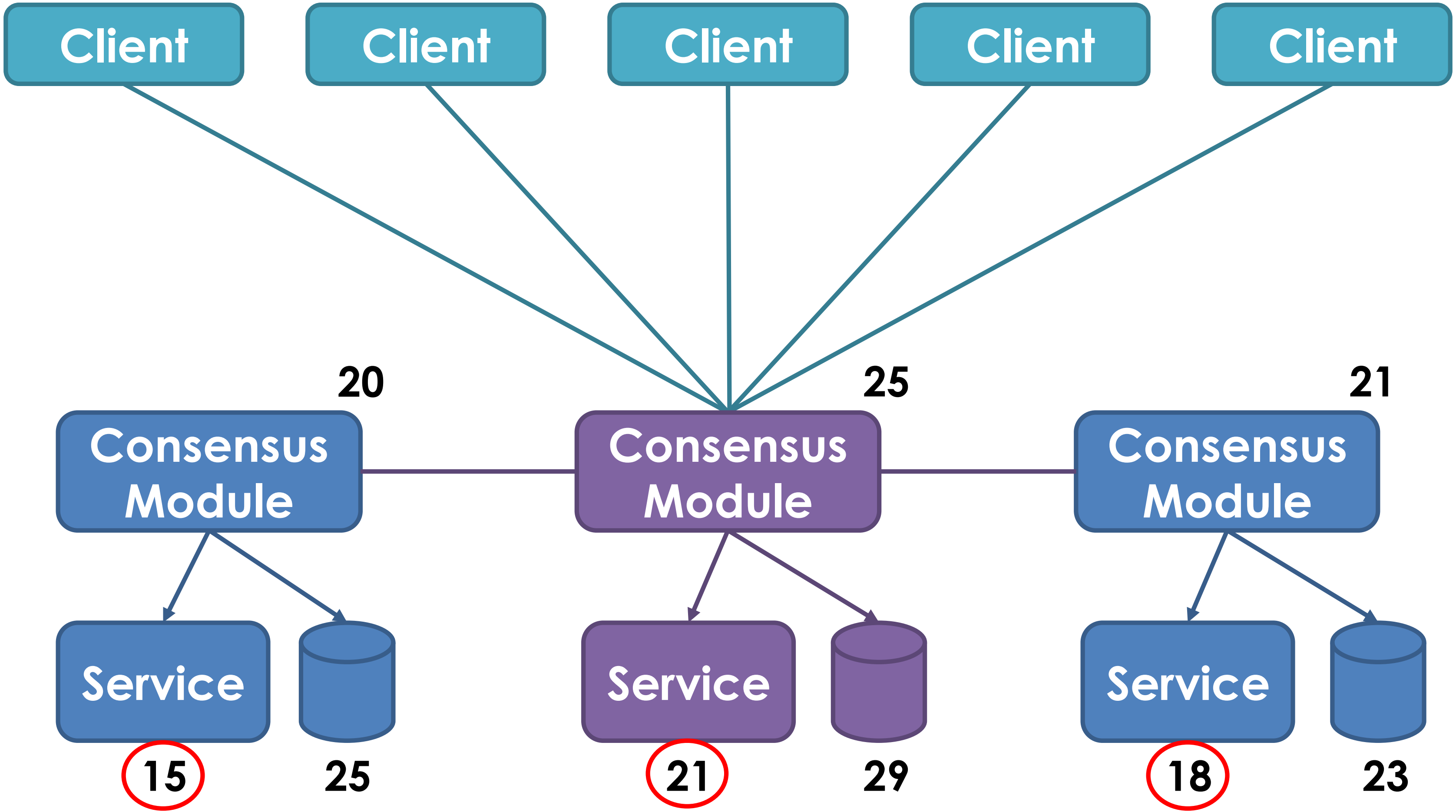

Client
Client
Client
Client
Client
Consensus Module
Consensus Module
Consensus Module
Service
Service
Service


Client
Client
Client
Client
Client
20
25
21
Consensus Module
Consensus Module
Consensus Module
Service
Service
Service
15
25
21
29
18
23

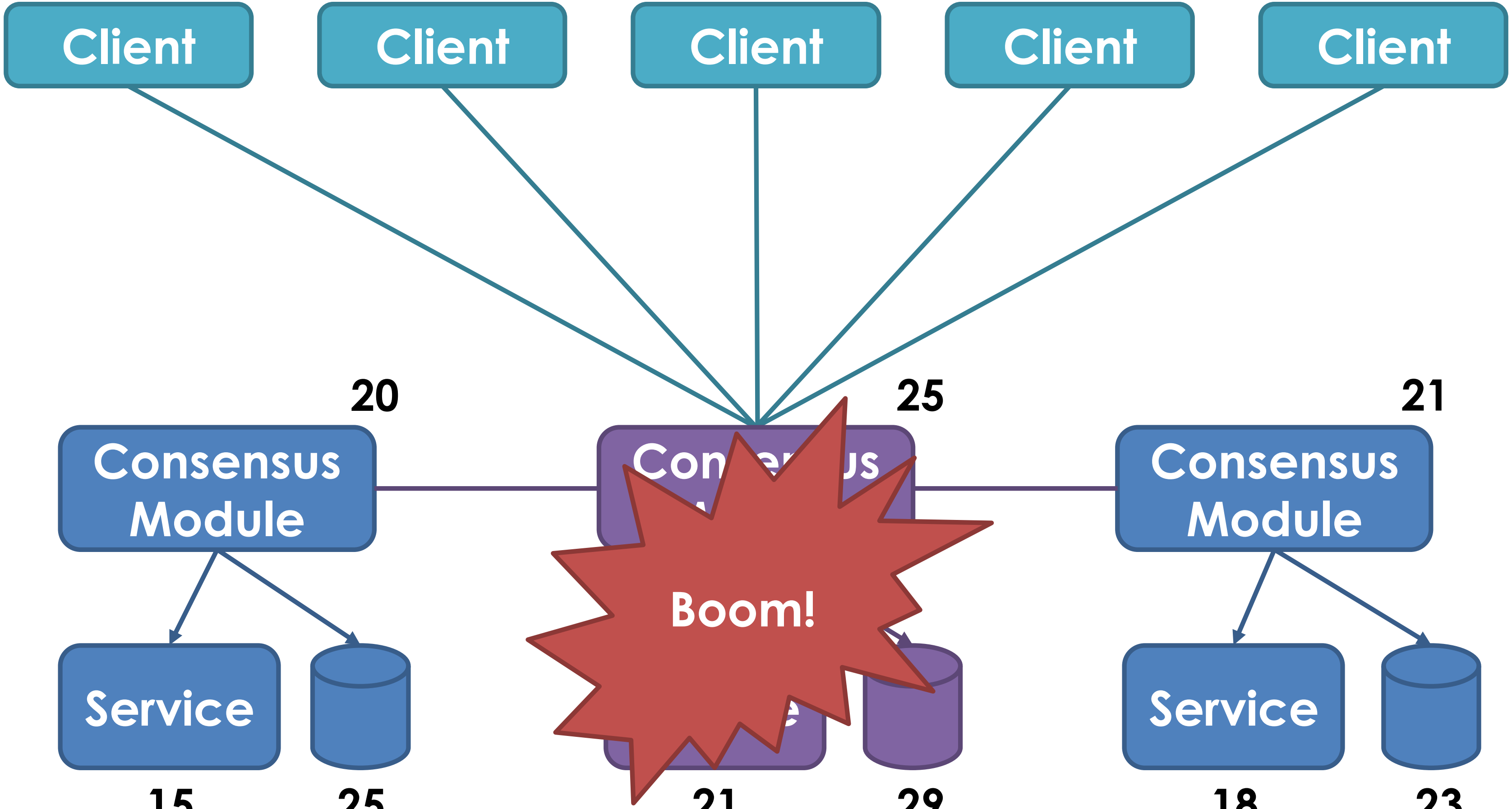


Append Position/Index


Client
Client
Client
Client
Client
Commit Position/Index
20
25
21
Consensus Module
Consensus Module
Consensus Module
Service
Service
Service
15
25
21
29
18
23



Service Position/Index



Client

Client

Client

Client

Client

20

Consensus Module

25

Con e us

21

Consensus Module

Boom!

Service

15

25

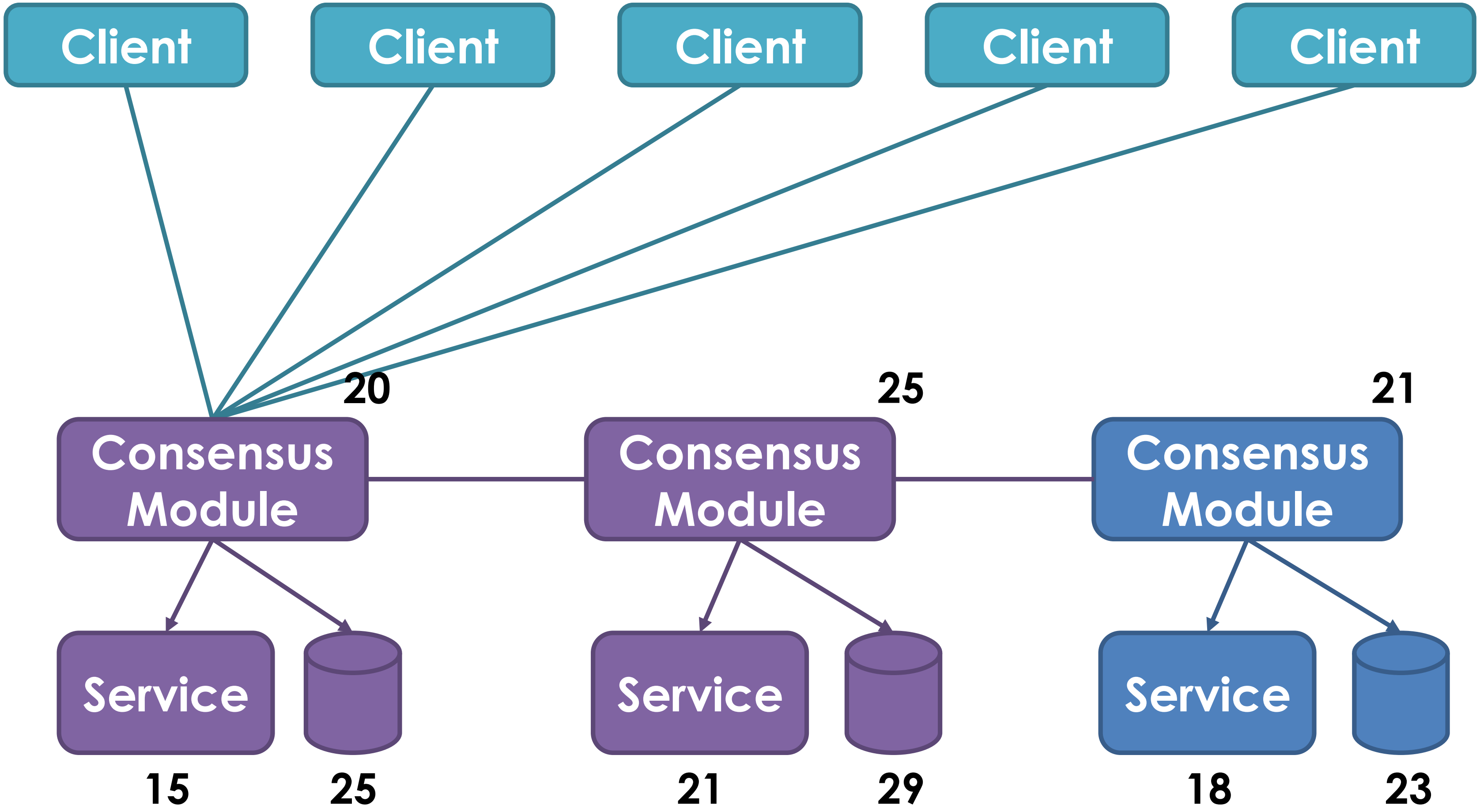
21

29

Service

18

23


Client
Client
Client
Client
Client
20
25
21
Consensus Module
Consensus Module
Consensus Module
Service
Service
Service
15
25
21
29
18
23

Performance

Performance

1. Don’t pollute the business logic thread

Performance

1. Don’t pollute the business logic thread
2. Choose appropriate data structures

Performance

1. Don’t pollute the business logic thread
2. Choose appropriate data structures
3. Batch to amortise expensive costs

Performance

1. Don’t pollute the business logic thread
2. Choose appropriate data structures
3. Batch to amortise expensive costs
4. Continuously Perf Test and Profile

Scalability

Scalability

1. **Build multiple models as Services**

Scalability

1. **Build multiple models as Services**
2. **Shard model into Services**

Scalability

1. Build multiple models as Services
2. Shard model into Services
3. Message between Services via the log

Scalability

1. Build multiple models as Services
2. Shard model into Services
3. Message between Services via the log
4. Build replicas for queries, with caution

Wrapping up…

Model Fidelity

10+ years ago…

@mjpt777

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

“The future is already here – it's just not evenly distributed”

- William Gibson