Low End-to-End Latency atop a Speculative Shared Log with Fix-Ante Ordering





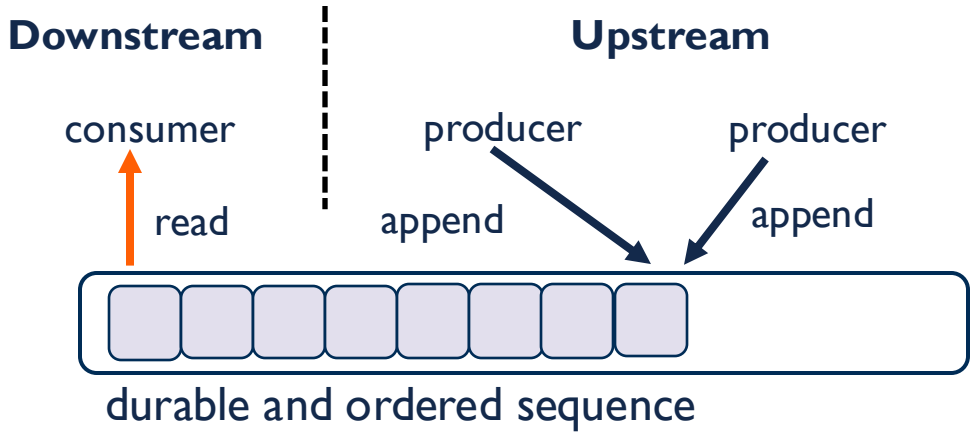

Shreesha G. Bhat
Tony Hong
Xuhao Luo
Jiyu Hu
Aishwarya Ganesan
Ram Alagappan

University of Illinois Urbana-Champaign

Current Shared Logs → High App End-to-End Latency

Current Shared Logs → High App End-to-End Latency

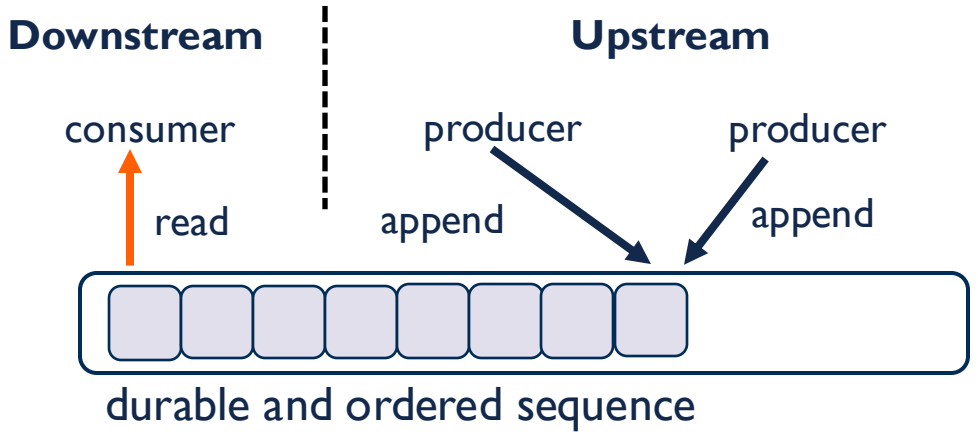
Shared logs are widely used by today's real-time, data-driven applications



Current Shared Logs → High App End-to-End Latency

Shared logs are widely used by today's real-time, data-driven applications

Well studied: Corfu, Scalog, Boki, LazyLog…

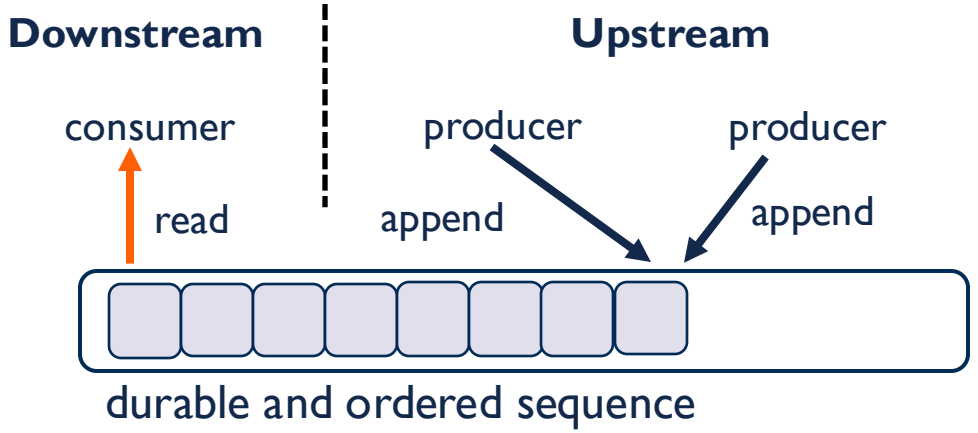


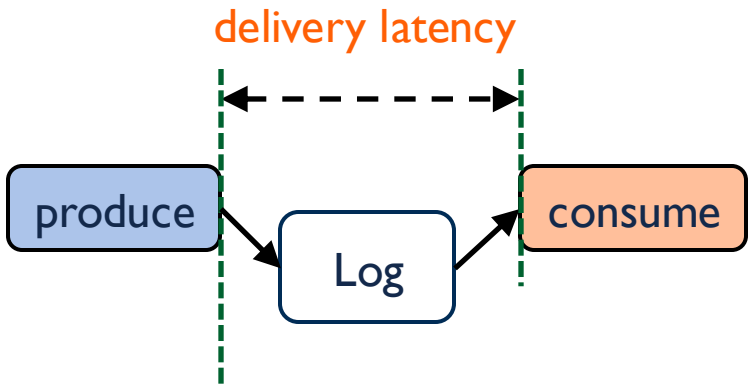
Current Shared Logs → High App End-to-End Latency

Shared logs are widely used by today's real-time, data-driven applications

Well studied: Corfu, Scalog, Boki, LazyLog…

State-of-the-art shared logs incur high delivery latencies





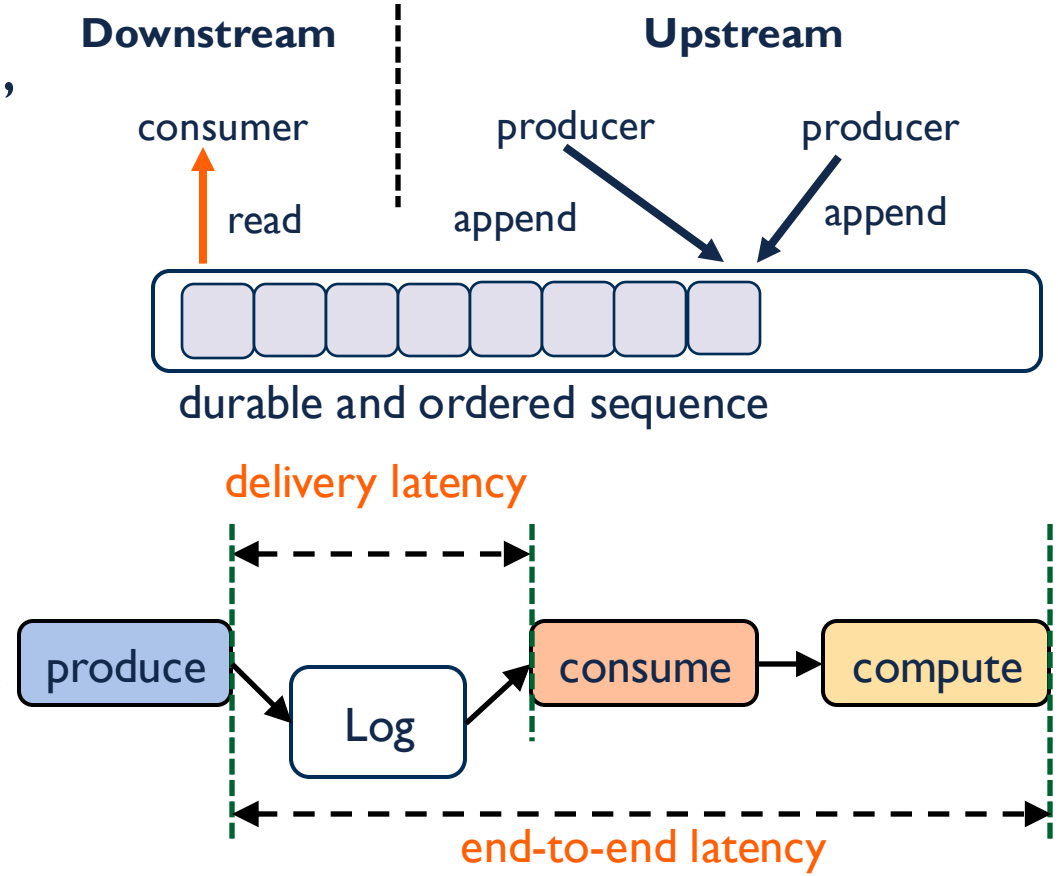
Current Shared Logs → High App End-to-End Latency

Shared logs are widely used by today's real-time, data-driven applications

Well studied: Corfu, Scalog, Boki, LazyLog…

State-of-the-art shared logs incur high delivery latencies

high delivery latency → high end-to-end latency



Current Shared Logs → High App End-to-End Latency

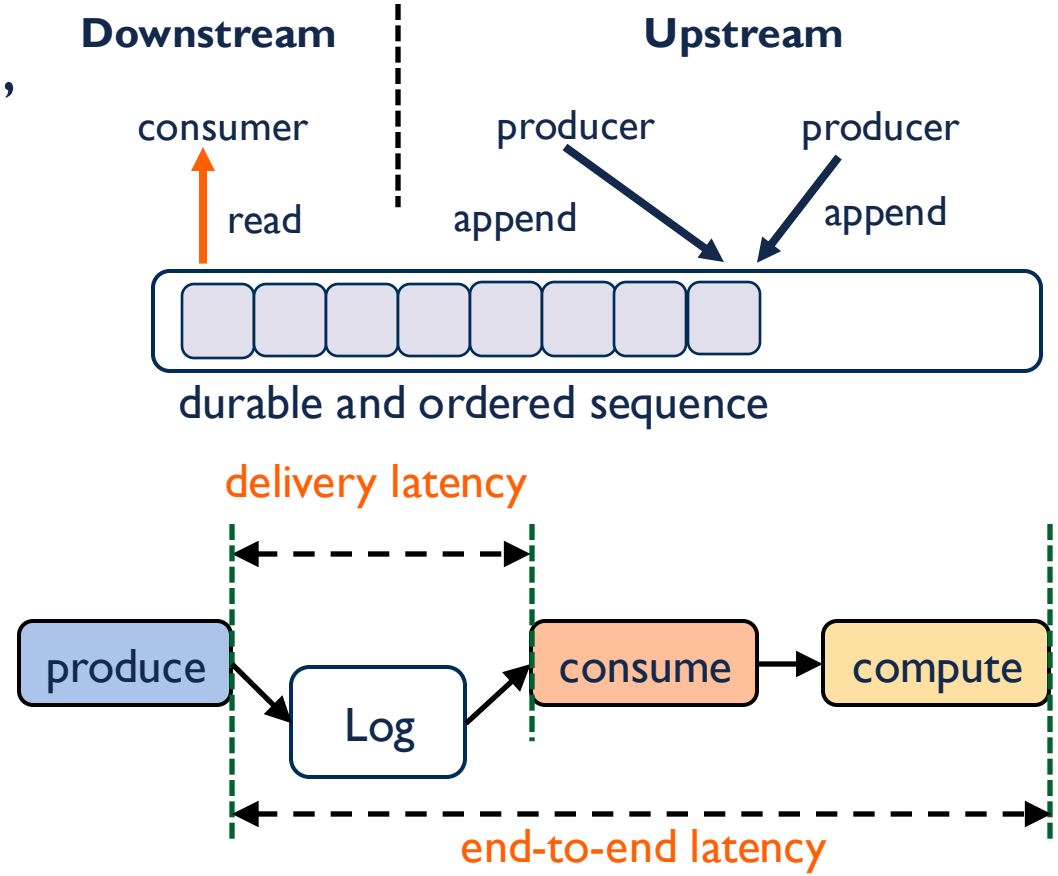
Shared logs are widely used by today's real-time, data-driven applications

Well studied: Corfu, Scalog, Boki, LazyLog…

State-of-the-art shared logs incur high delivery latencies

high delivery latency → high end-to-end latency

Cause: expensive ordering before record delivery



The SpecLog Abstraction and Fix-Ante Ordering

The SpecLog Abstraction and Fix-Ante Ordering

We note an opportunity for speculative execution

The SpecLog Abstraction and Fix-Ante Ordering

We note an opportunity for speculative execution

SpecLog Abstraction

Allow speculative consumption of records by predicting order

Overlap ordering and application compute

Confirm the order after compute

The SpecLog Abstraction and Fix-Ante Ordering

We note an opportunity for speculative execution

SpecLog Abstraction

Allow speculative consumption of records by predicting order

Overlap ordering and application compute

Confirm the order after compute

Fix-Ante Ordering

Predetermine the global order for easy prediction

Make system adhere to that order

High speculation success!

Contributions

- *SpecLog* – A new speculative shared log abstraction, uses fix-ante ordering to enable near perfect speculation

- *Belfast* – Implementation of *SpecLog* abstraction and fix-ante ordering
  - Addresses practical challenges that arise in implementing fix-ante ordering
  - Enables low e2e latency while retaining benefits and guarantees of today's shared logs

- E2E latency benefits in end applications like fraud detection, intrusion detection and high-frequency trading

Outline

- Introduction
- Motivation
- SpecLog Abstraction and Interface
- Belfast – An Implementation of SpecLog
- Evaluation

Shared Logs

Shared Logs

Expose simple interface

```
pos_t append(record_t r);
record_t read(pos_t pos);
// streaming records
stream<record_t> subscribe(pos_t start);
```

Shared Logs

Expose simple interface

```
pos_t append(record_t r);
record_t read(pos_t pos);
// streaming records
stream<record_t> subscribe(pos_t start);
```

Durable & linearizably ordered records

- Respect real-time order of appends

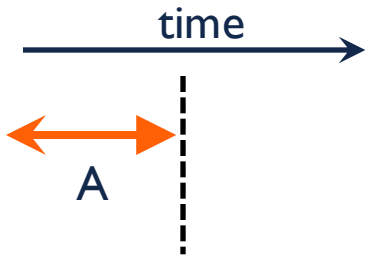
Shared Logs

Expose simple interface

```
pos_t append(record_t r);
record_t read(pos_t pos);
// streaming records
stream<record_t> subscribe(pos_t start);
```

Durable & linearizably ordered records

- Respect real-time order of appends



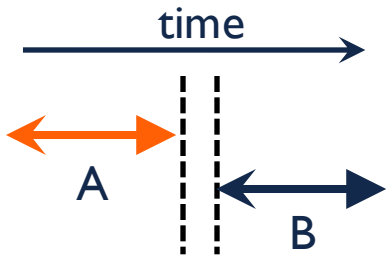
Shared Logs

Expose simple interface

```
pos_t append(record_t r);
record_t read(pos_t pos);
// streaming records
stream<record_t> subscribe(pos_t start);
```

Durable & linearizably ordered records

- Respect real-time order of appends



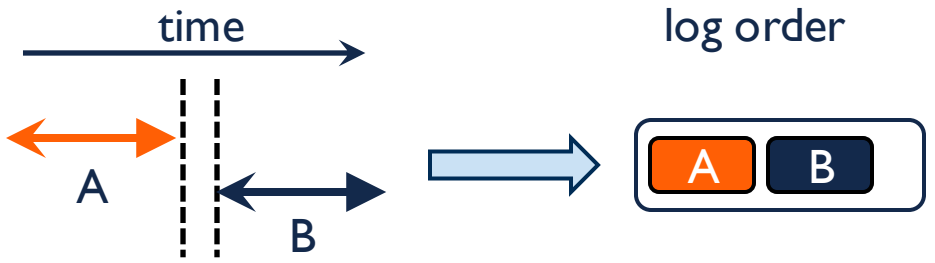
Shared Logs

Expose simple interface

```
pos_t append(record_t r);
record_t read(pos_t pos);
// streaming records
stream<record_t> subscribe(pos_t start);
```

Durable & linearizably ordered records

- Respect real-time order of appends



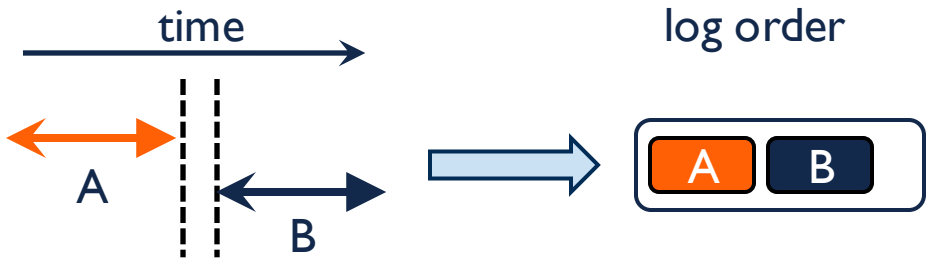
Shared Logs

Expose simple interface

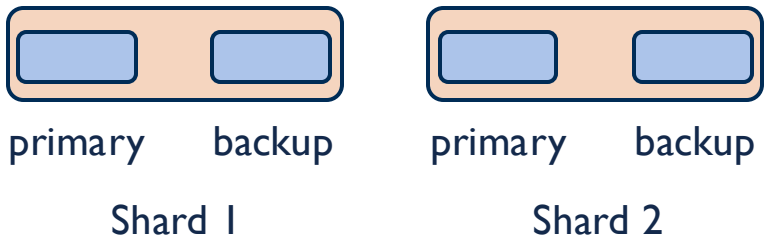
```
pos_t append(record_t r);
record_t read(pos_t pos);
// streaming records
stream<record_t> subscribe(pos_t start);
```

Durable & linearizably ordered records

- Respect real-time order of appends



Store records over multiple *storage shards*



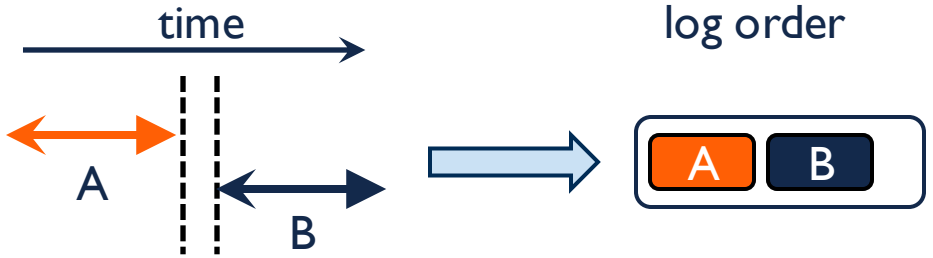
Shared Logs

Expose simple interface

```
pos_t append(record_t r);
record_t read(pos_t pos);
// streaming records
stream<record_t> subscribe(pos_t start);
```

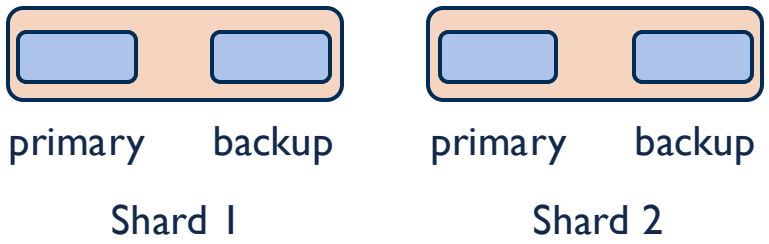
Durable & linearizably ordered records

- Respect real-time order of appends



Store records over multiple *storage shards*

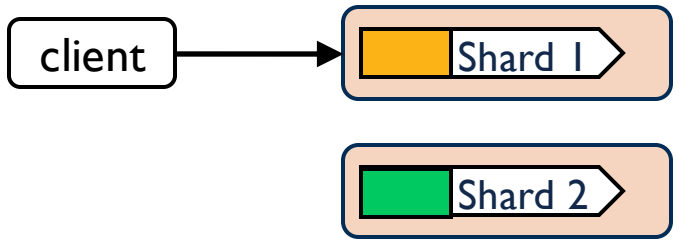
- Total order of records across shards



State-of-the-art – Durability-First Shared Logs (Scalog)

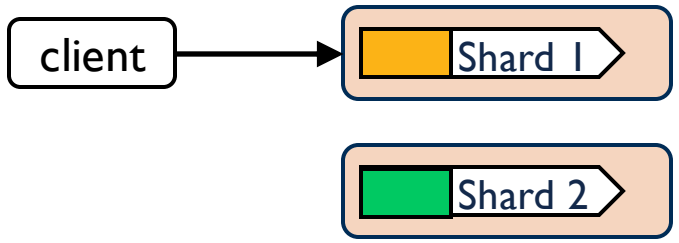
State-of-the-art – Durability-First Shared Logs (Scalog)

Durability first – appended records first made durable



State-of-the-art – Durability-First Shared Logs (Scalog)

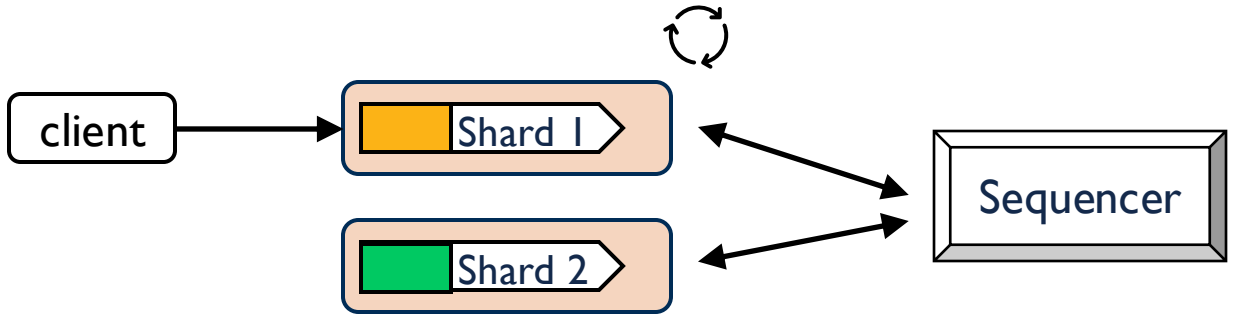
Durability first – appended records first made durable



State-of-the-art – Durability-First Shared Logs (Scalog)

Durability first – appended records first made durable

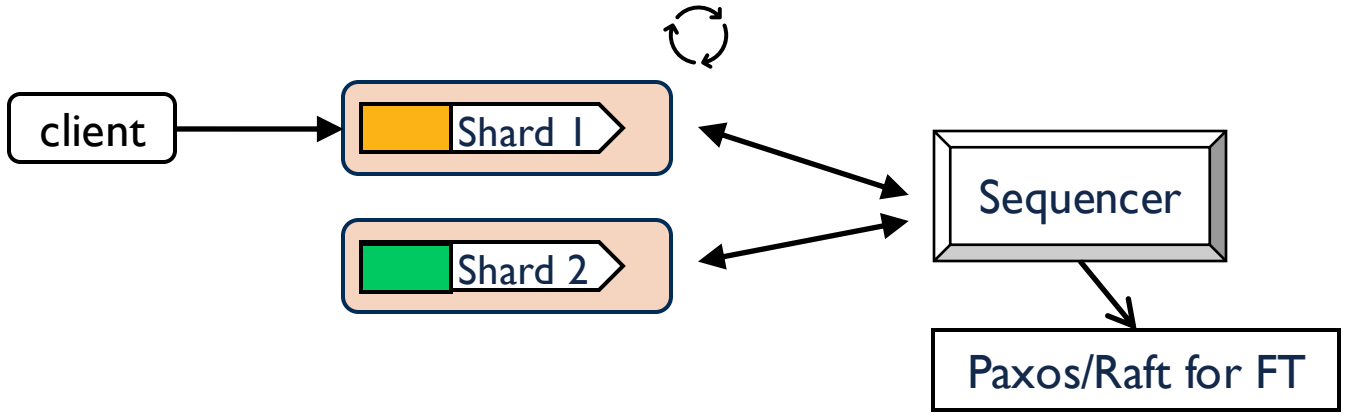
Batched ordering – amortize cost of ordering through batching



State-of-the-art – Durability-First Shared Logs (Scalog)

Durability first – appended records first made durable

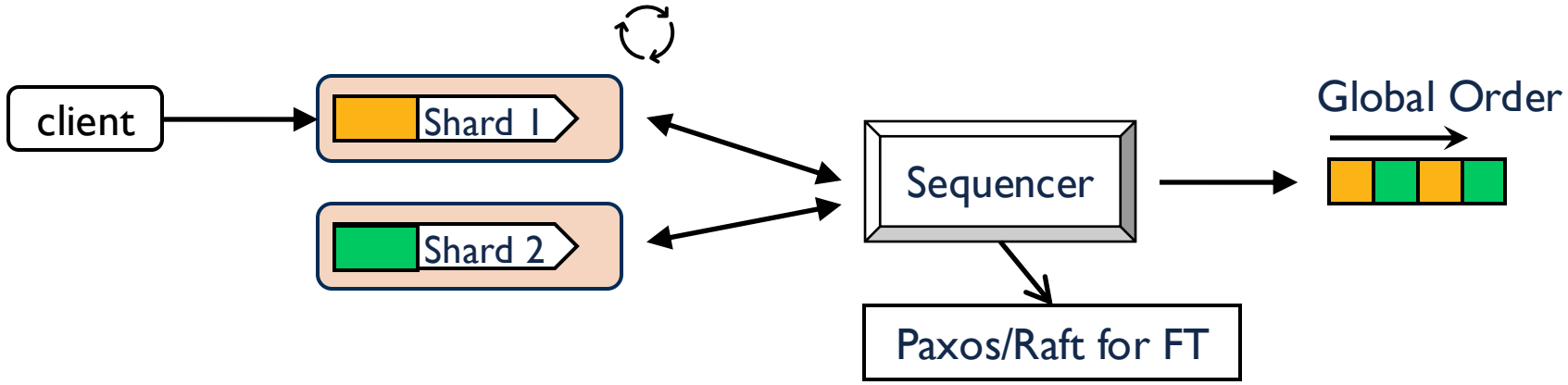
Batched ordering – amortize cost of ordering through batching



State-of-the-art – Durability-First Shared Logs (Scalog)

Durability first – appended records first made durable

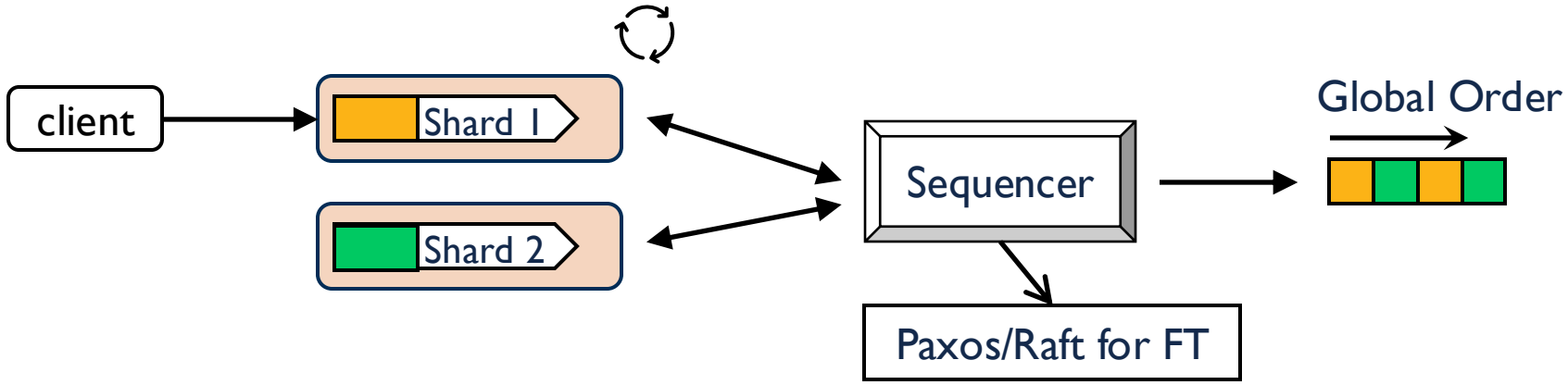
Batched ordering – amortize cost of ordering through batching



State-of-the-art – Durability-First Shared Logs (Scalog)

Durability first – appended records first made durable

Batched ordering – amortize cost of ordering through batching

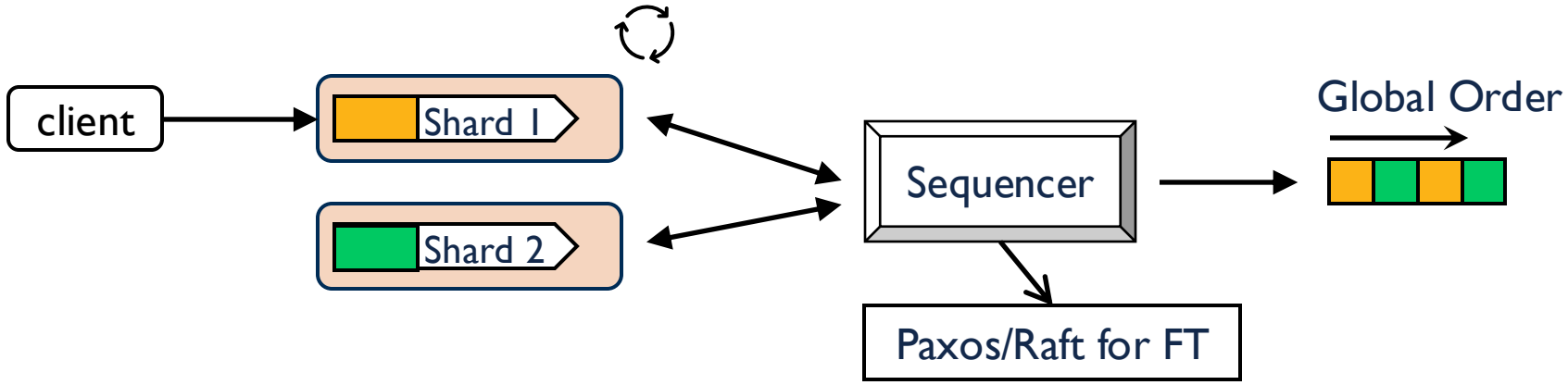


Flexible Data Placement **Seamlessly Reconfigurable** **Scalablity**

State-of-the-art – Durability-First Shared Logs (Scalog)

Durability first – appended records first made durable

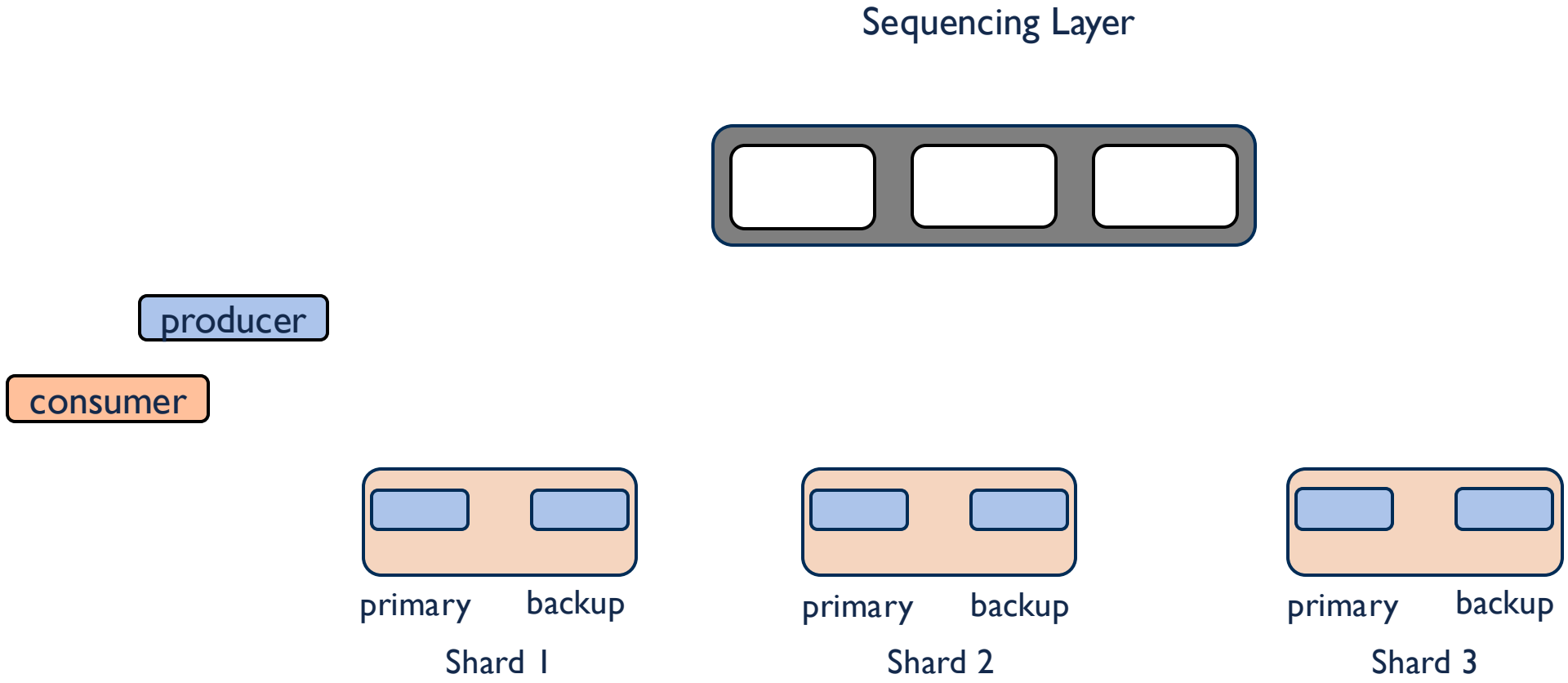
Batched ordering – amortize cost of ordering through batching



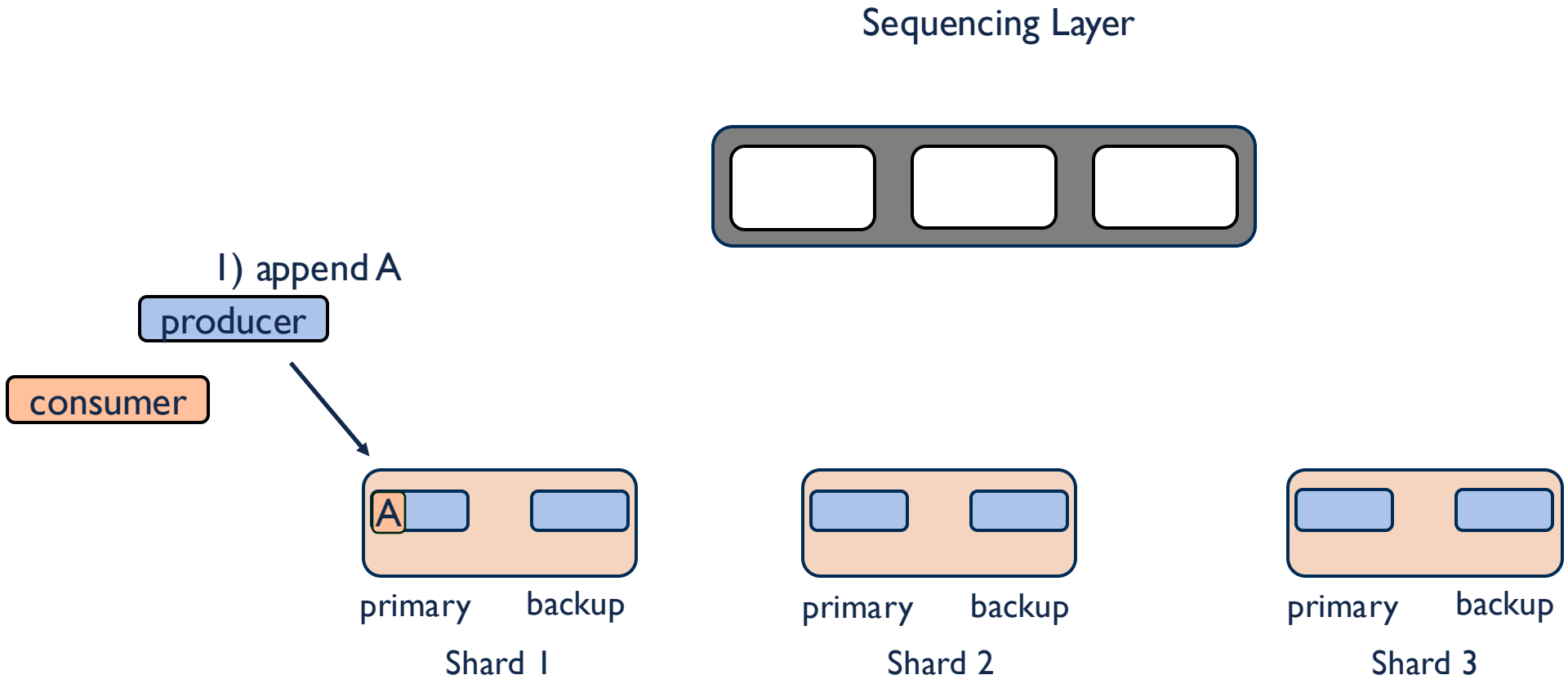
Flexible Data Placement **Seamlessly Reconfigurable** **Scalablity**

Similar designs adopted by Boki [SOSP'21], FlexLog [HPDC'23]

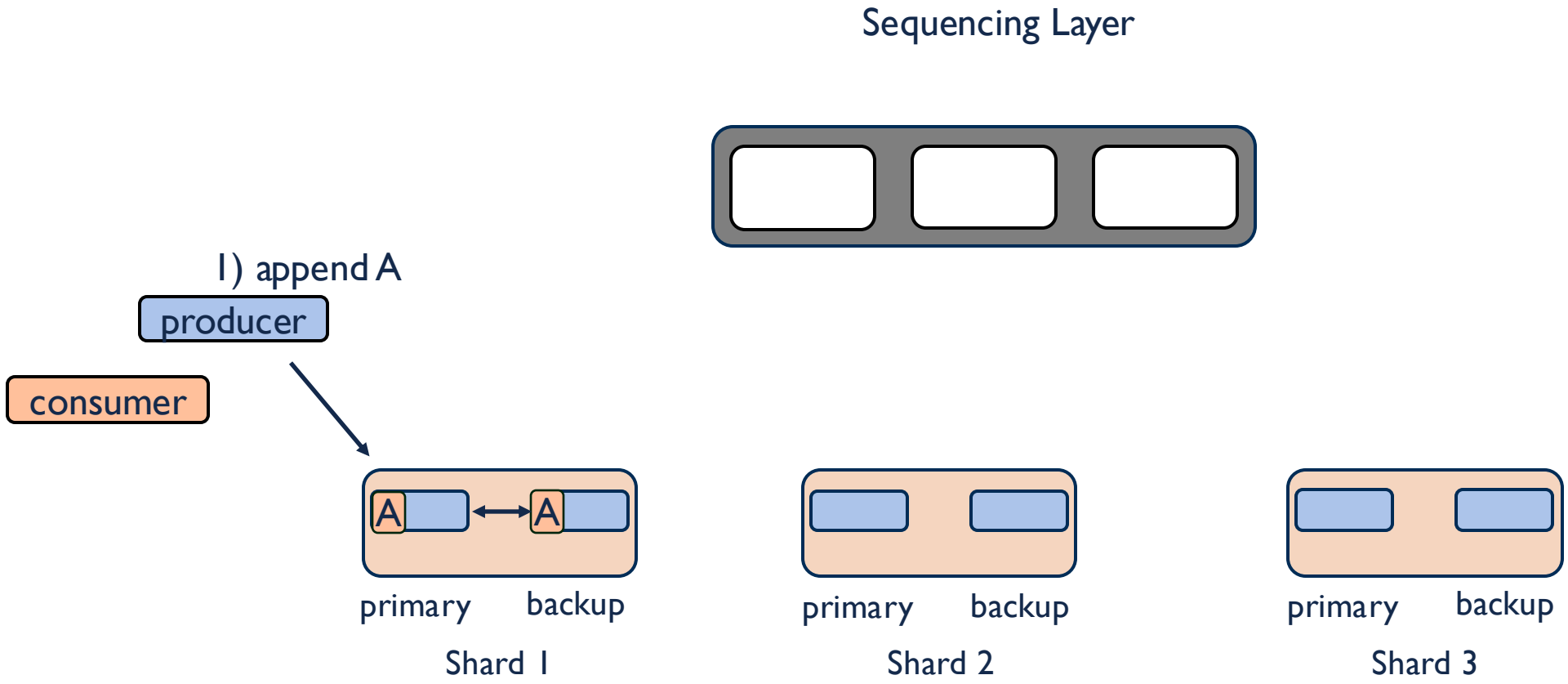
Problem: High Delivery Latency



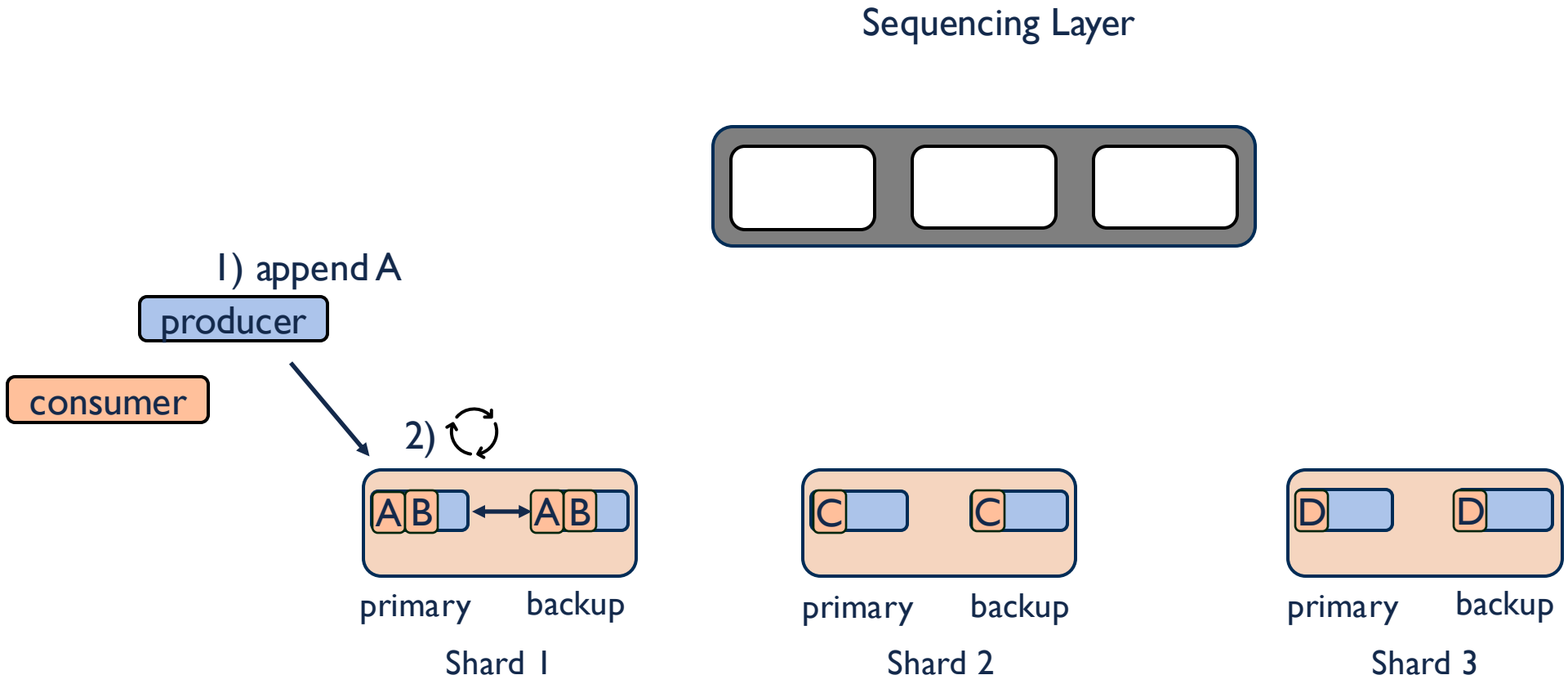
Problem: High Delivery Latency



Problem: High Delivery Latency

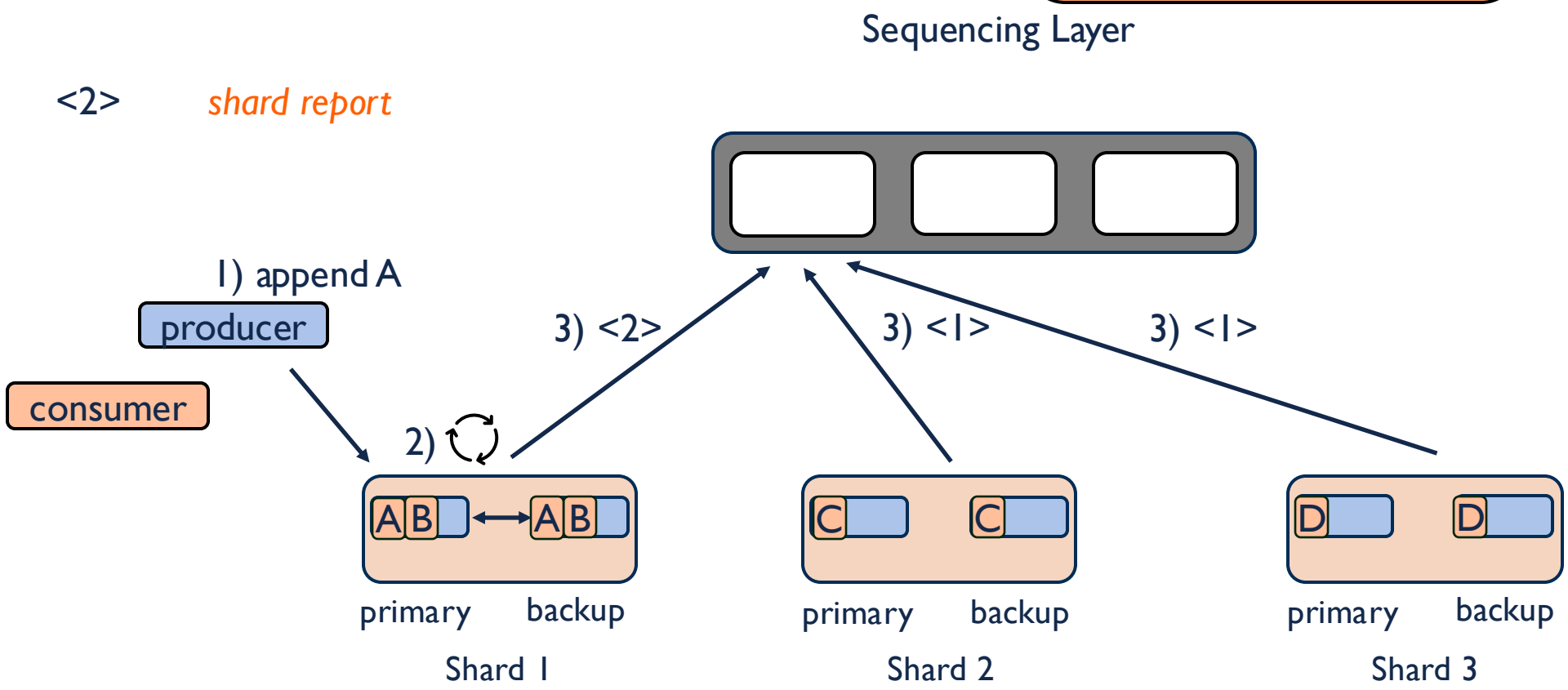


Problem: High Delivery Latency



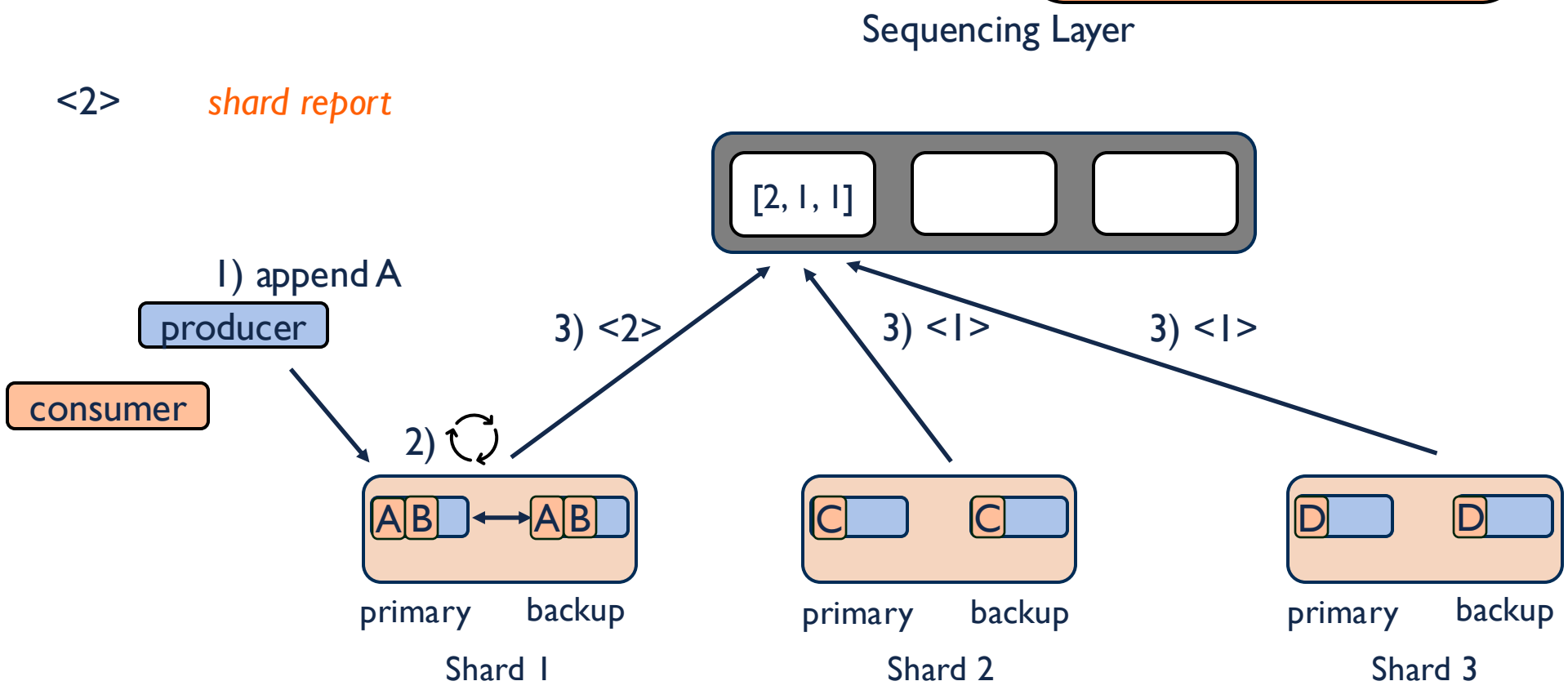
Problem: High Delivery Latency

Note:
Shard reports <L> imply a **local log length of L** at the shard



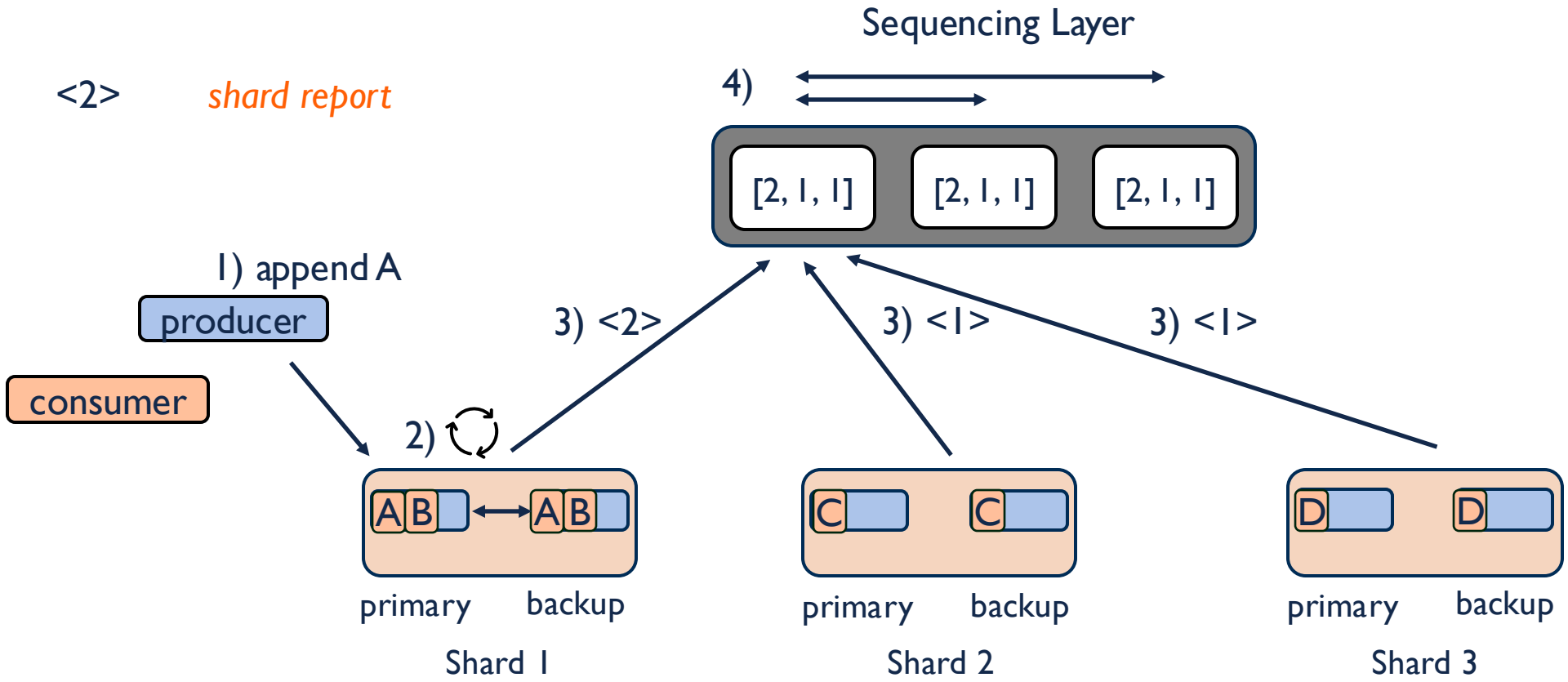
Problem: High Delivery Latency

Note:
Shard reports <L> imply a **local log length of L** at the shard



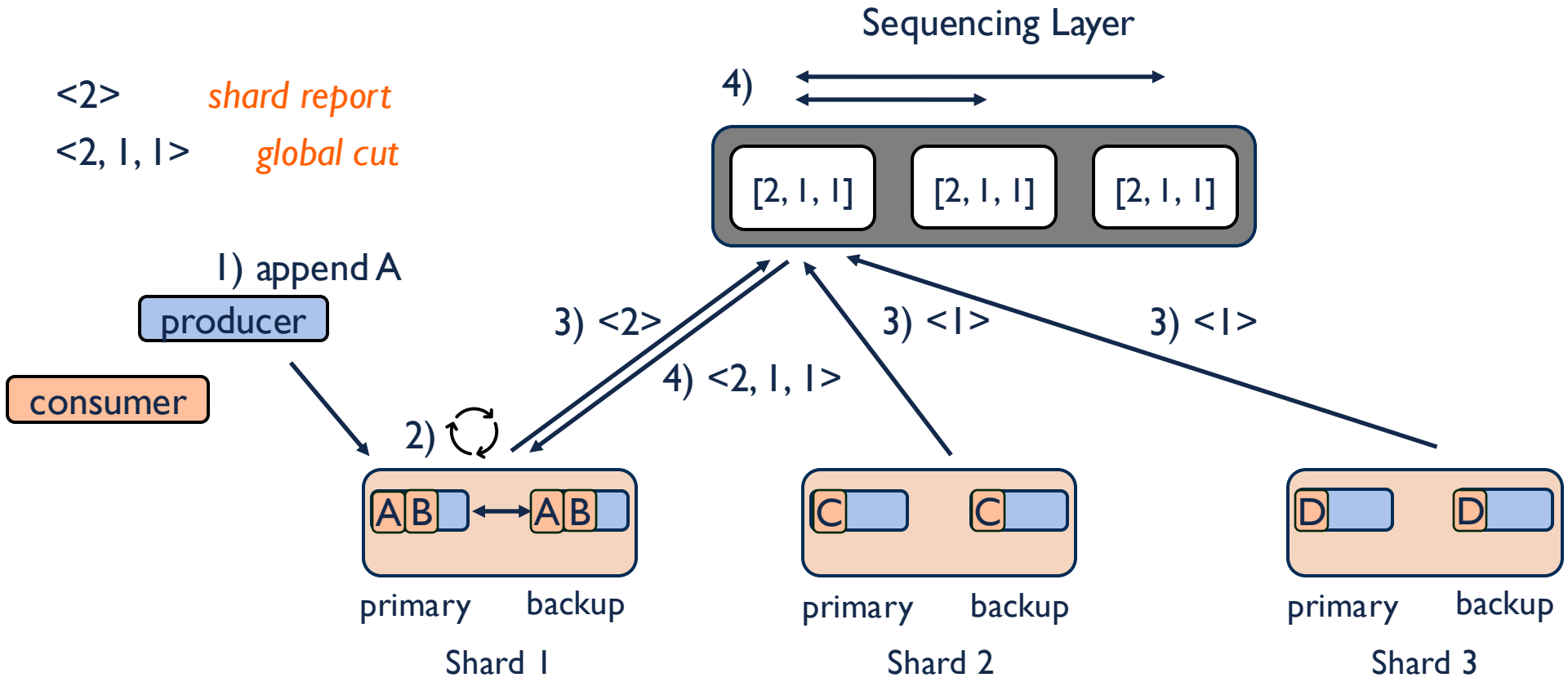
Problem: High Delivery Latency

Note:
Shard reports <L> imply a **local log length of L** at the shard



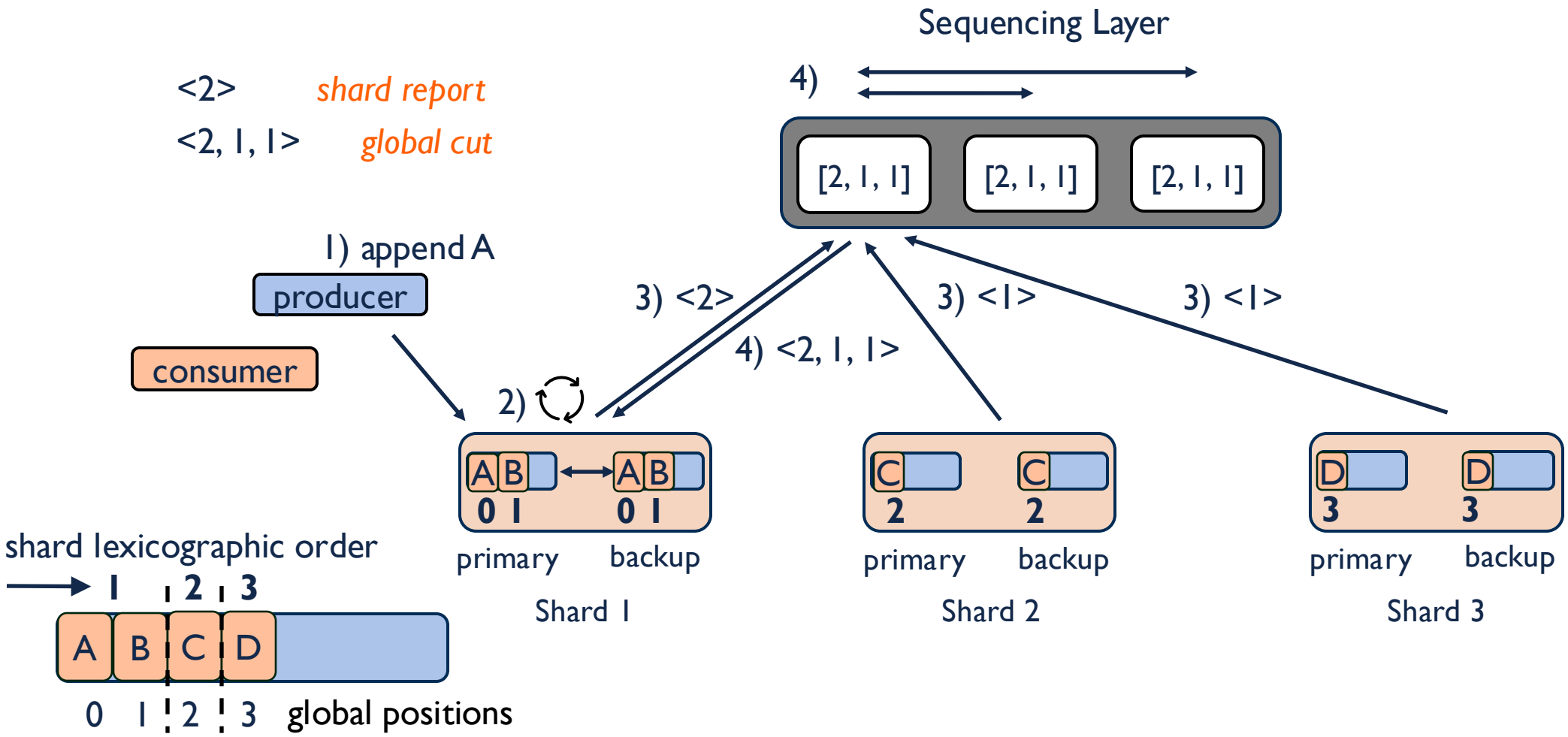
Problem: High Delivery Latency

Note:
Shard reports <L> imply a **local log length of L** at the shard



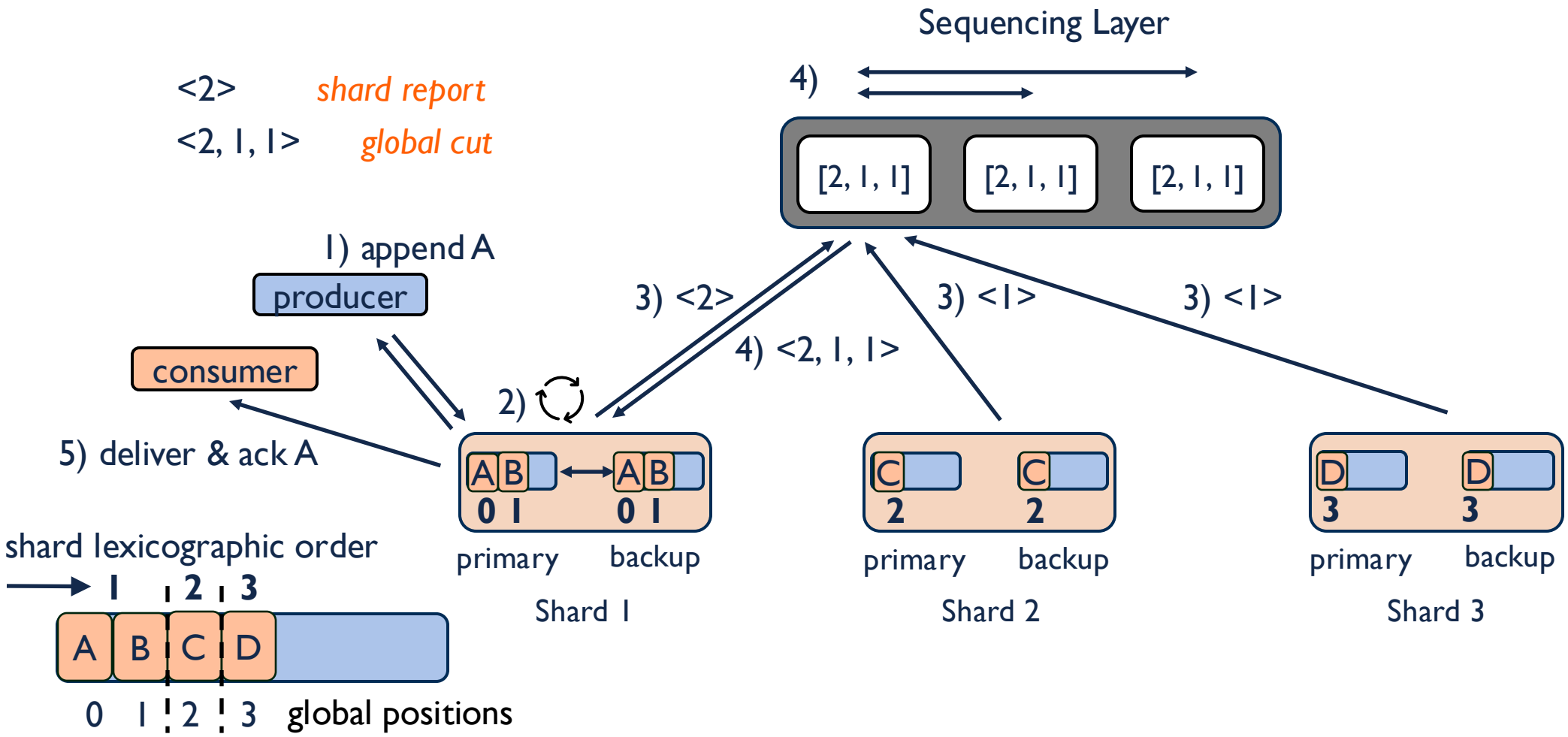
Problem: High Delivery Latency

Note:
Shard reports <L> imply a **local log length of L** at the shard



Problem: High Delivery Latency

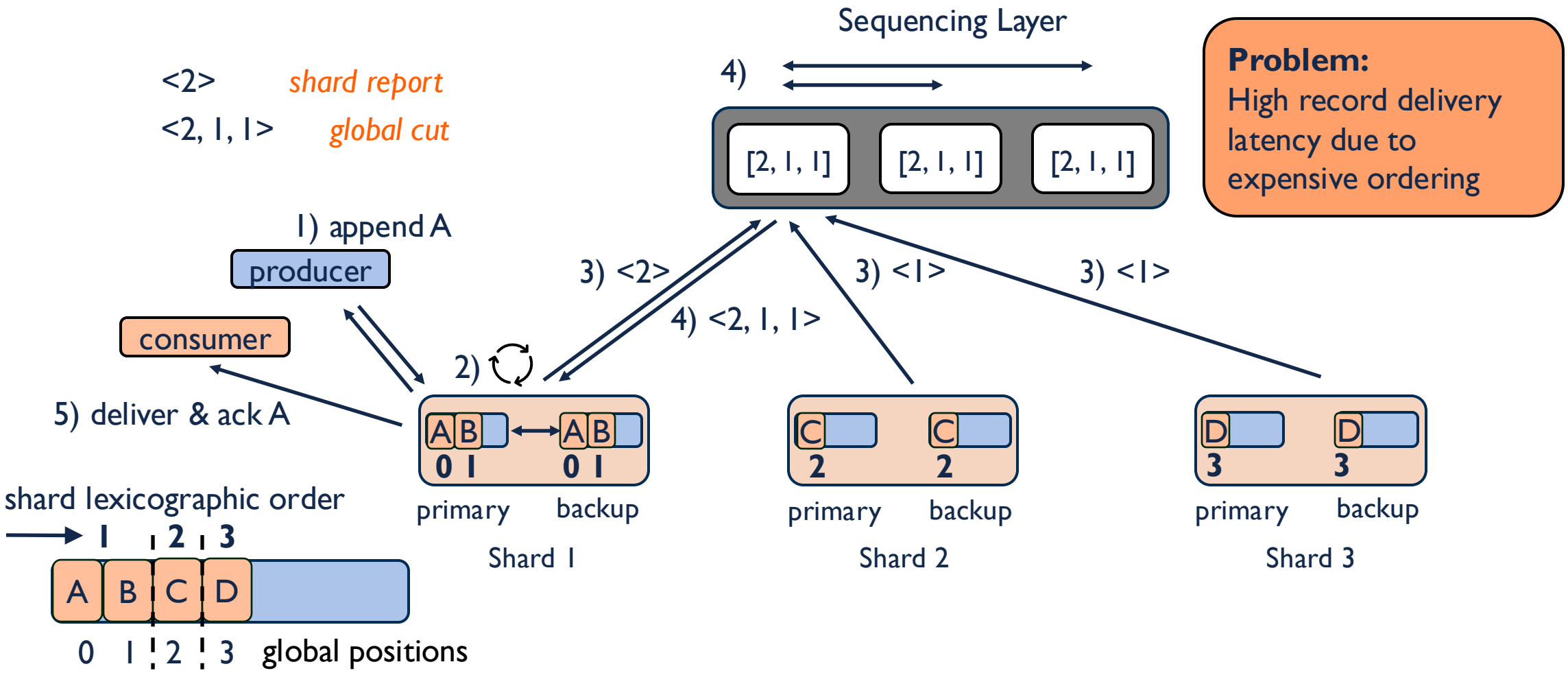
Note:
Shard reports <L> imply a **local log length of L** at the shard



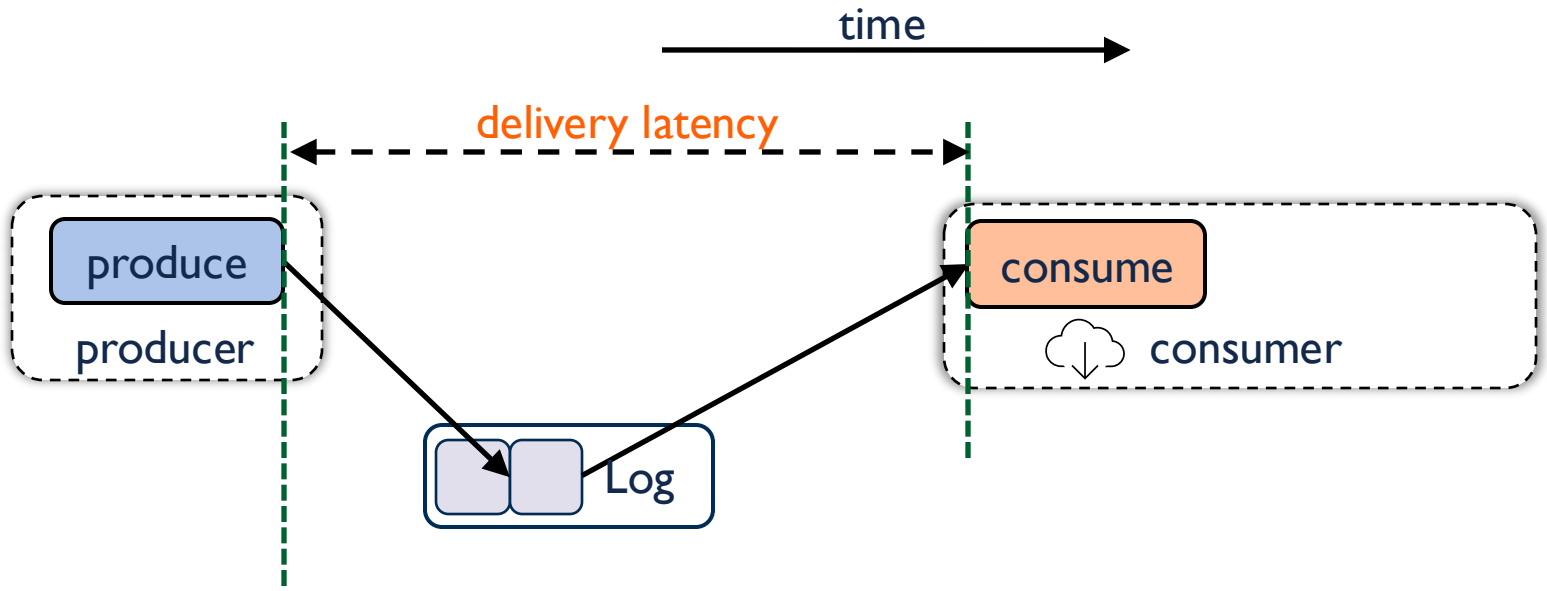
Problem: High Delivery Latency

Note:
Shard reports <L> imply a **local log length of L** at the shard

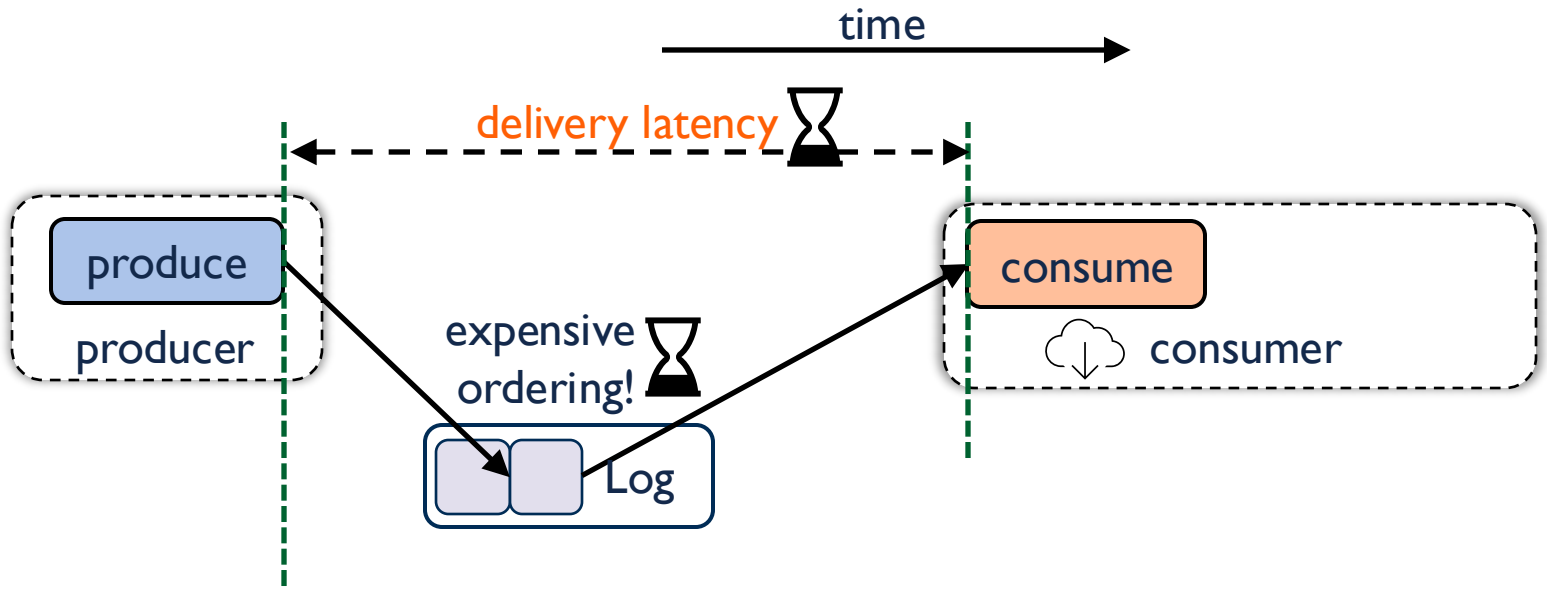
Problem:
High record delivery latency due to expensive ordering



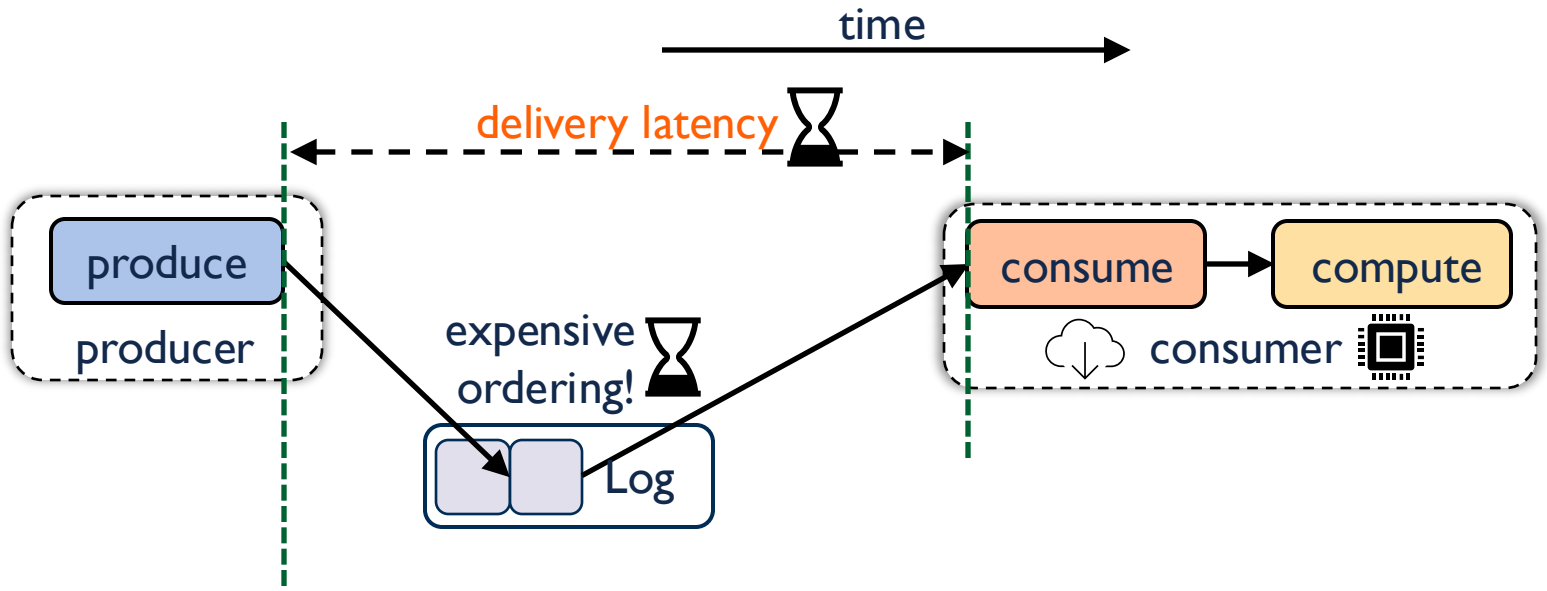
High Delivery Latency → High App E2E Latency



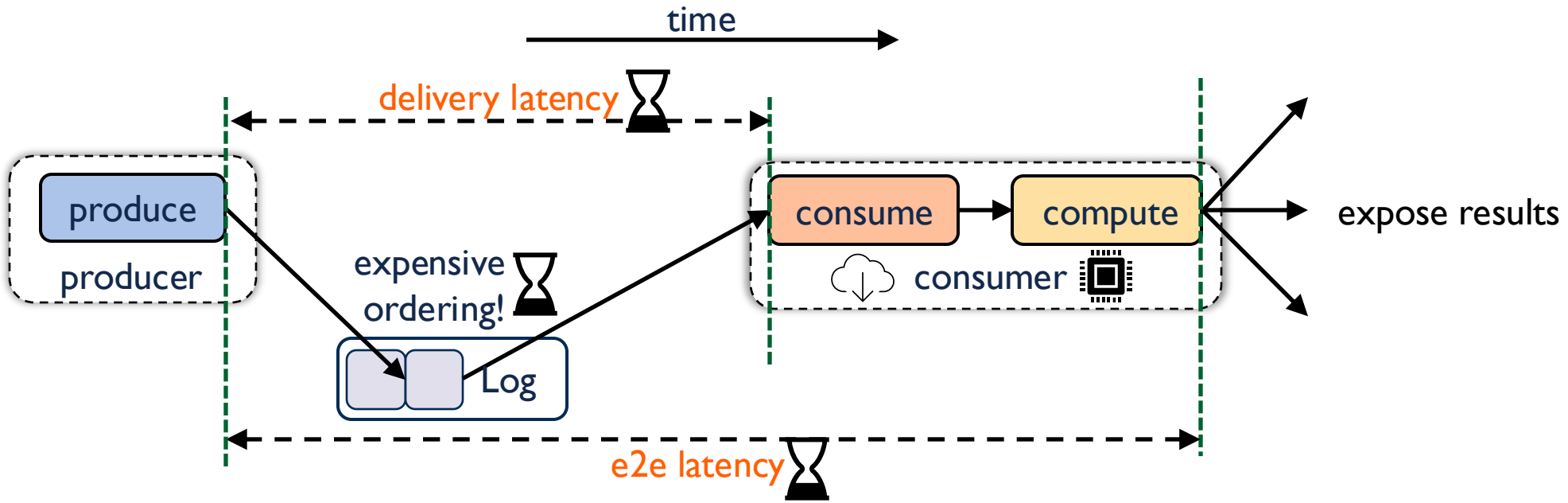
High Delivery Latency → High App E2E Latency



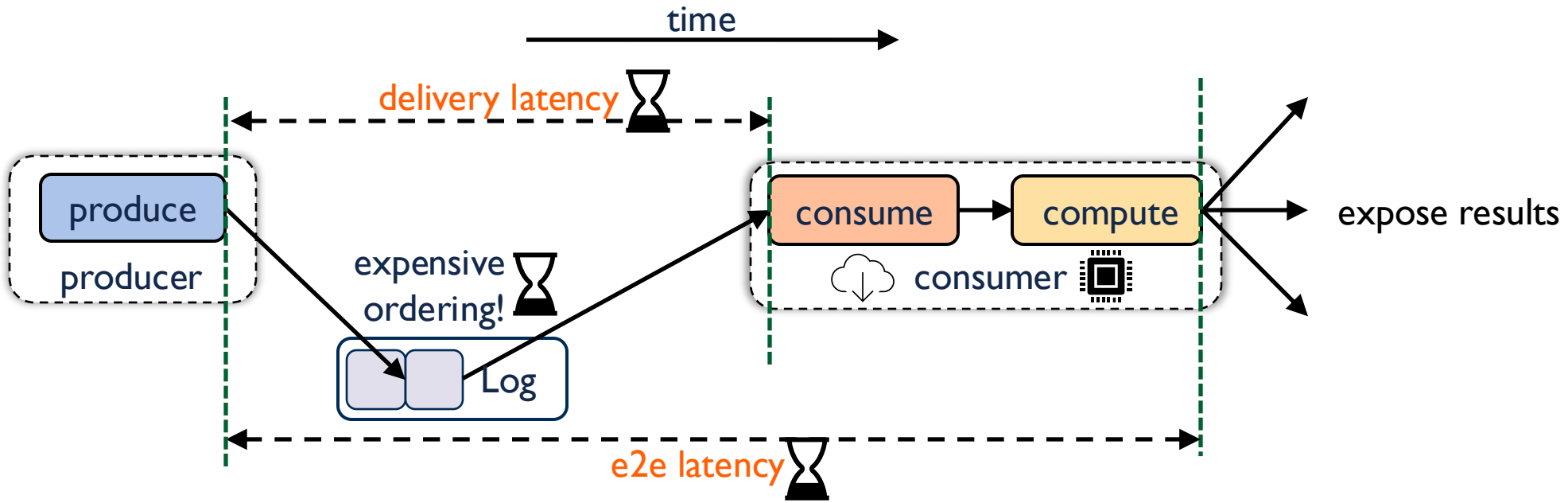
High Delivery Latency → High App E2E Latency



High Delivery Latency → High App E2E Latency



High Delivery Latency → High App E2E Latency



Fraud detection: consume txn → match w/ history (compute) → flag txn

Latency Demands of Real-Time Applications

Latency Demands of Real-Time Applications

Many applications care about e2e latency

- Real-time analytics, high-frequency trading, fraud and intrusion detection systems

These apps demand a low delivery latency from the shared log

Latency Demands of Real-Time Applications

Many applications care about e2e latency

- Real-time analytics, high-frequency trading, fraud and intrusion detection systems

These apps demand a low delivery latency from the shared log

2023 streaming report by RedPanda

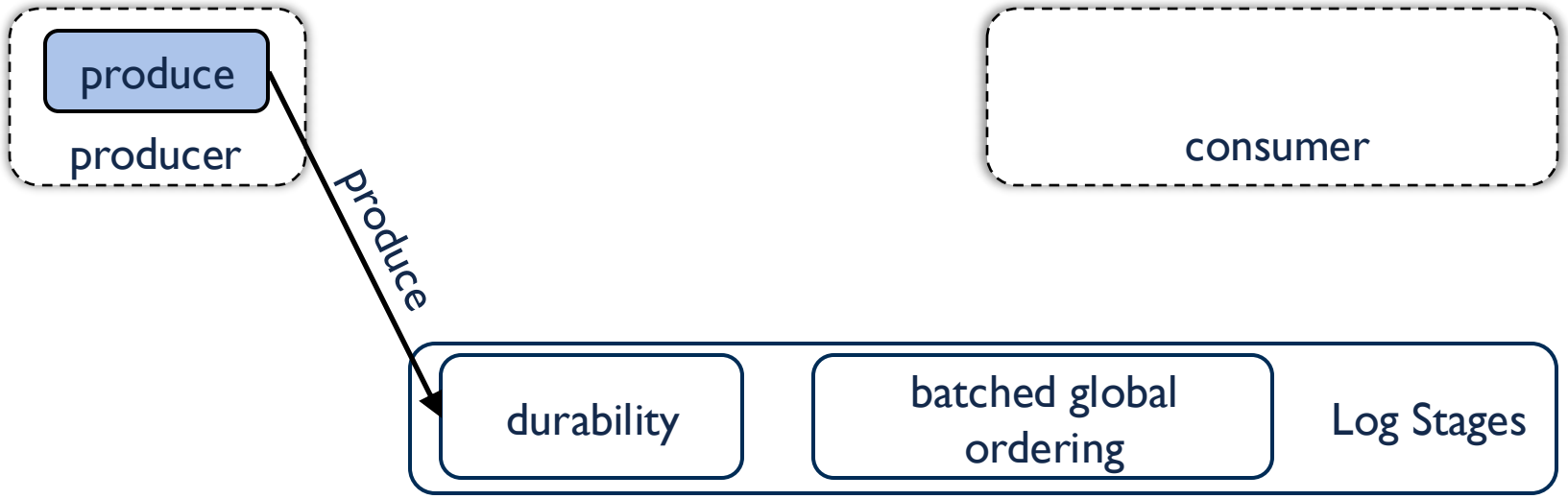
~35% practitioners primarily care about delivery latency of their streaming system

Outline

- Introduction
- Motivation
- SpecLog Abstraction and Interface
- Belfast – An Implementation of SpecLog
- Evaluation

An Opportunity for Speculation in Shared Logs

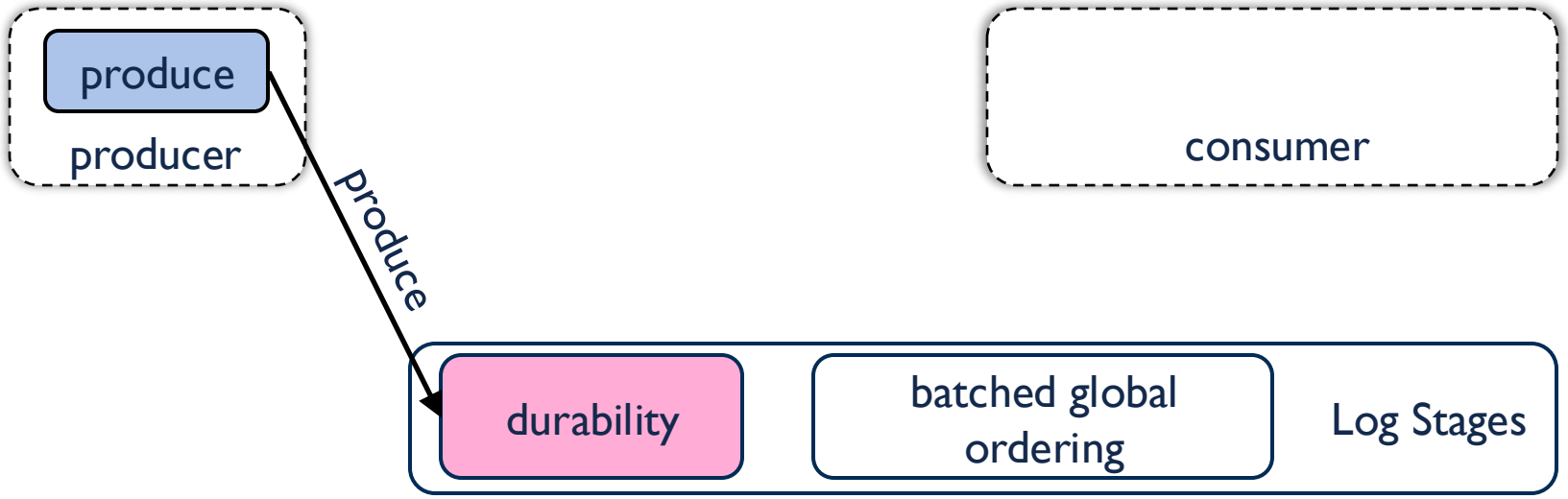
SpecLog identifies opportunity for speculative execution



Effectively overlaps global ordering with application compute → low end-to-end latency

An Opportunity for Speculation in Shared Logs

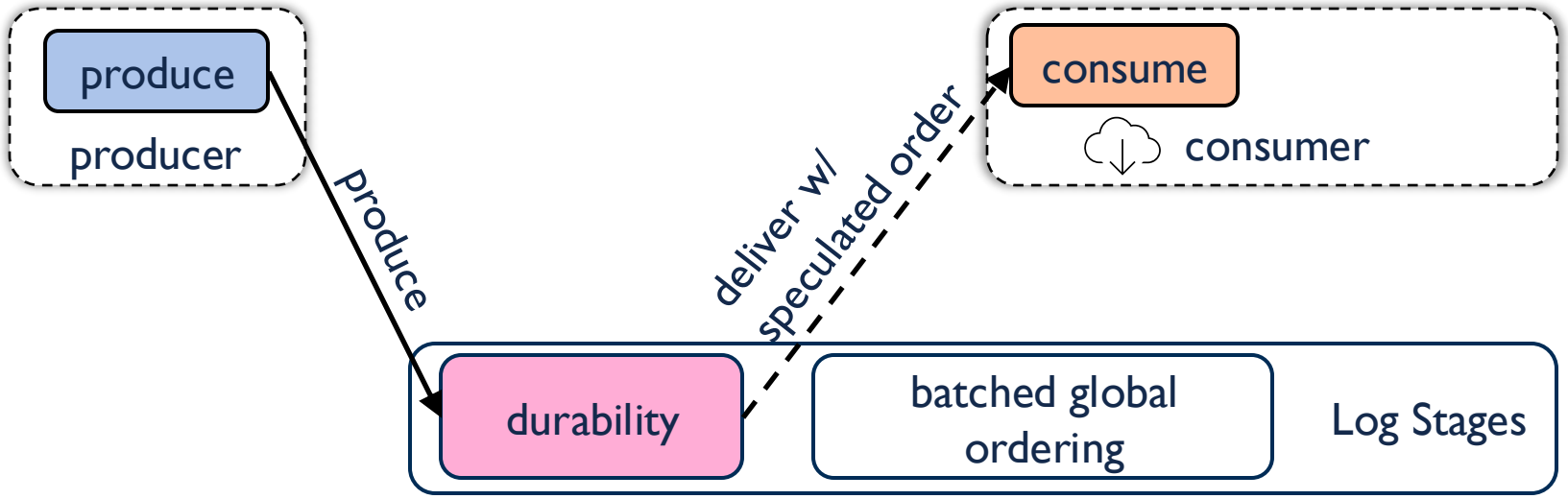
SpecLog identifies opportunity for speculative execution



Effectively overlaps global ordering with application compute → low end-to-end latency

An Opportunity for Speculation in Shared Logs

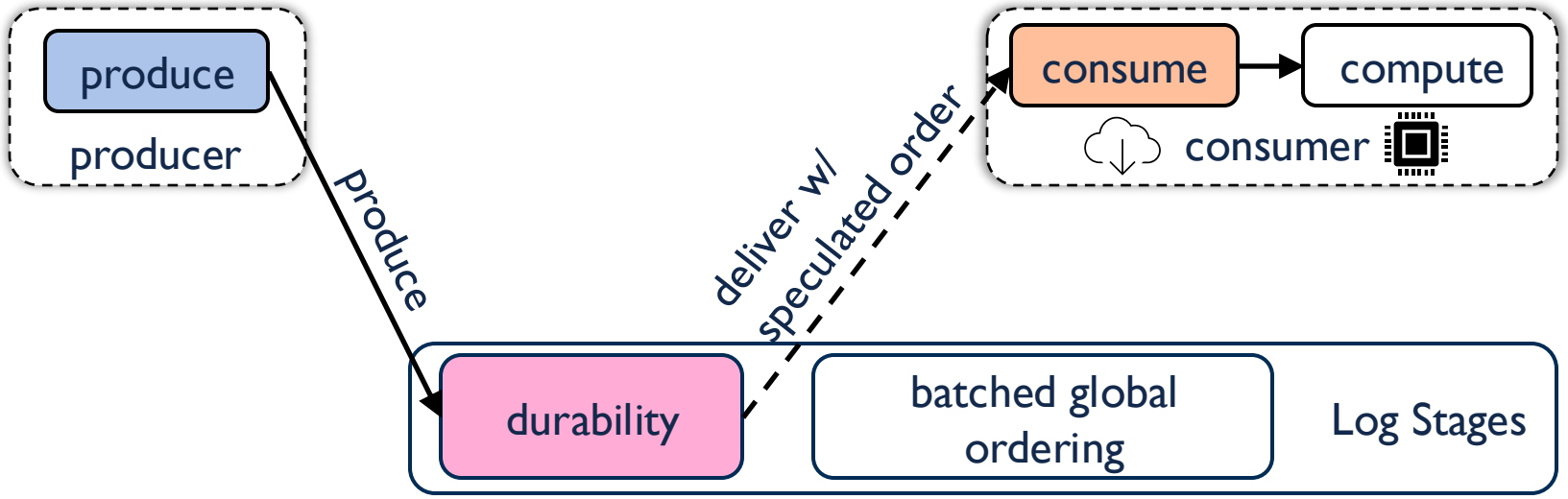
SpecLog identifies opportunity for speculative execution



Effectively overlaps global ordering with application compute → low end-to-end latency

An Opportunity for Speculation in Shared Logs

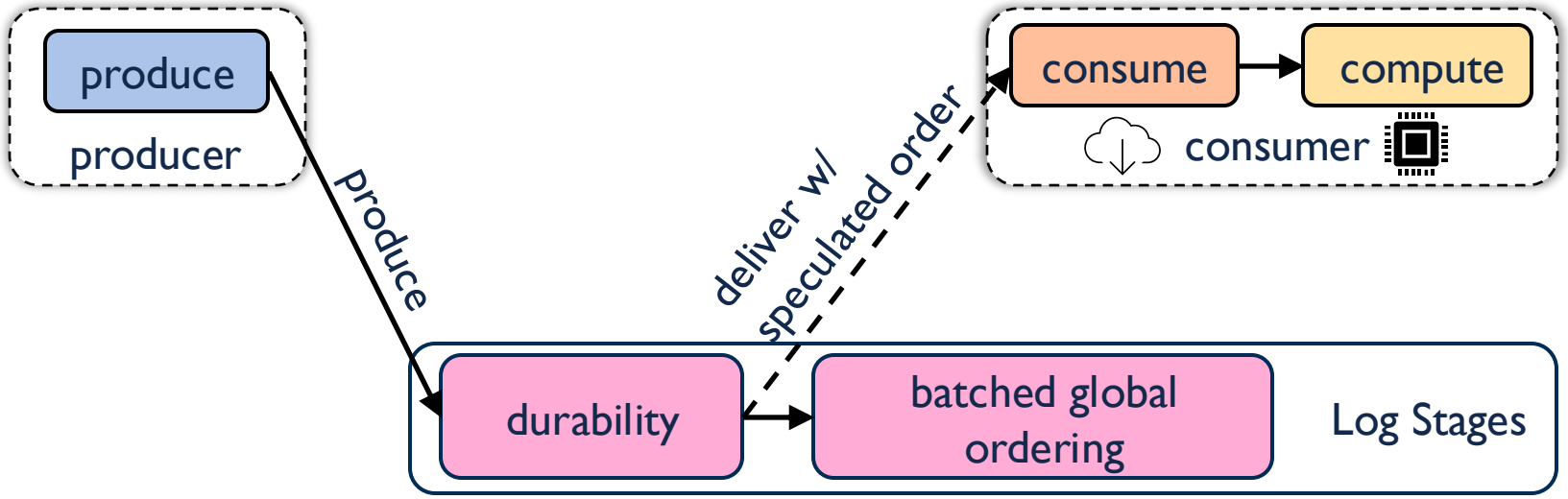
SpecLog identifies opportunity for speculative execution



Effectively overlaps global ordering with application compute → low end-to-end latency

An Opportunity for Speculation in Shared Logs

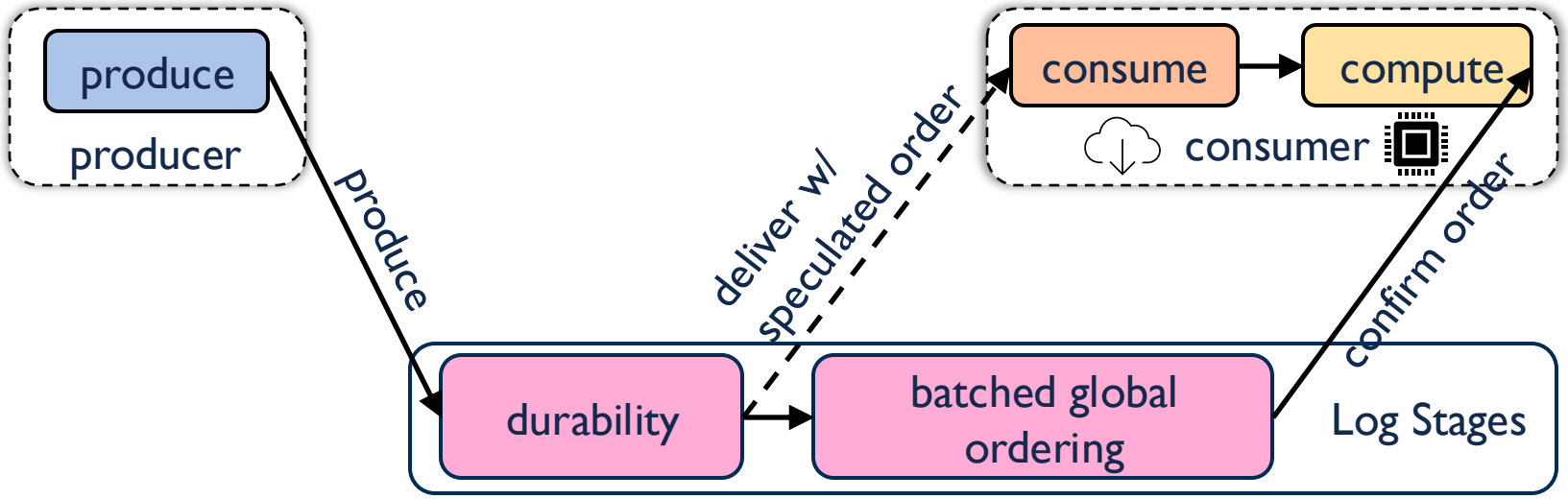
SpecLog identifies opportunity for speculative execution



Effectively overlaps global ordering with application compute → low end-to-end latency

An Opportunity for Speculation in Shared Logs

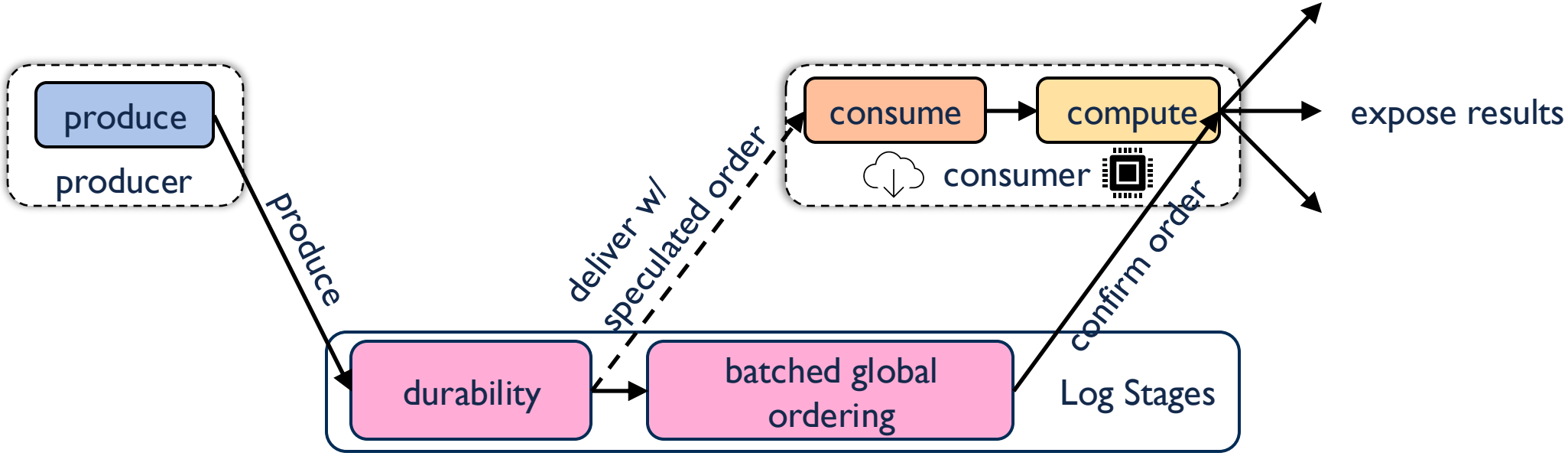
SpecLog identifies opportunity for speculative execution



Effectively overlaps global ordering with application compute → low end-to-end latency

An Opportunity for Speculation in Shared Logs

SpecLog identifies opportunity for speculative execution



Effectively overlaps global ordering with application compute → low end-to-end latency

SpecLog – Abstraction and Interface

Abstraction

Interface

SpecLog – Abstraction and Interface

Abstraction

SpecLog appends return after global ordering

Interface

```
// append to shard; return position of record in log
pos_t append(record_t r, sid_t shard);
```

SpecLog – Abstraction and Interface

Abstraction

SpecLog appends return after global ordering

SpecLog allows speculative consumption of records

Interface

```
// append to shard; return position of record in log
pos_t append(record_t r, sid_t shard);

// speculatively consume records after start
stream<record_t> subscribe(pos_t start);
```

SpecLog – Abstraction and Interface

Abstraction

SpecLog appends return after global ordering

SpecLog allows speculative consumption of records

SpecLog confirms or fails speculation

Interface

```
// append to shard; return position of record in log
pos_t append(record_t r, sid_t shard);

// speculatively consume records after start
stream<record_t> subscribe(pos_t start);

// callbacks for confirmations and mis-spec
void confirm_spec(pos_t upto);
void fail_spec(pos_t after);
```

New interface callbacks

SpecLog – Abstraction and Interface

Impact to Apps

Interface

```
// append to shard; return position of record in log
pos_t append(record_t r, sid_t shard);

// speculatively consume records after start
stream<record_t> subscribe(pos_t start);

// callbacks for confirmations and mis-spec
void confirm_spec(pos_t upto);
void fail_spec(pos_t after);
```

New interface callbacks

SpecLog – Abstraction and Interface

Impact to Apps

Apps listen to confirmations and mis-speculations

Interface

```
// append to shard; return position of record in log
pos_t append(record_t r, sid_t shard);

// speculatively consume records after start
stream<record_t> subscribe(pos_t start);

// callbacks for confirmations and mis-spec
void confirm_spec(pos_t upto);
void fail_spec(pos_t after);
```

New interface callbacks

SpecLog – Abstraction and Interface

Impact to Apps

Apps listen to confirmations and mis-speculations

Upon `confirm_spec(k)` → expose compute results till `k`

Interface

```
// append to shard; return position of record in log
pos_t append(record_t r, sid_t shard);

// speculatively consume records after start
stream<record_t> subscribe(pos_t start);

// callbacks for confirmations and mis-spec
void confirm_spec(pos_t upto);
void fail_spec(pos_t after);
```

New interface callbacks

SpecLog – Abstraction and Interface

Impact to Apps

Apps listen to confirmations and mis-speculations

Upon `confirm_spec(k)` → expose compute results till `k`

Upon `fail_spec(k)` → rollback state until `k` and recompute

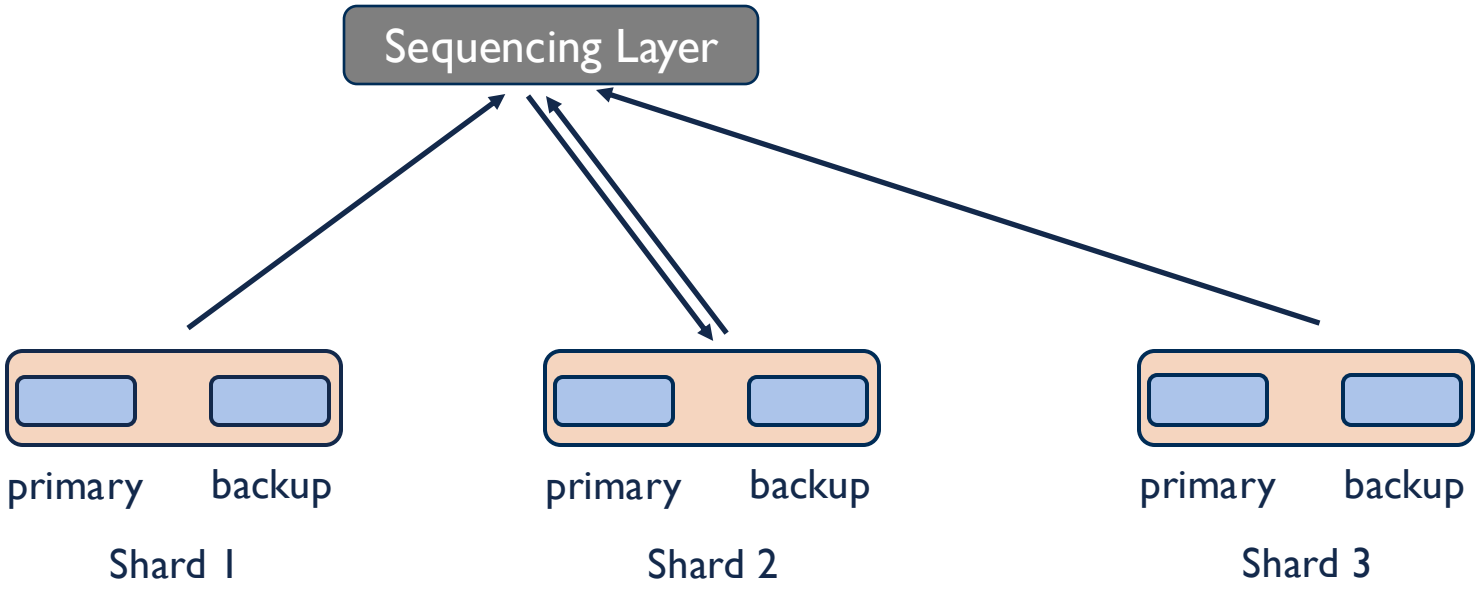
Interface

```
// append to shard; return position of record in log
pos_t append(record_t r, sid_t shard);

// speculatively consume records after start
stream<record_t> subscribe(pos_t start);

// callbacks for confirmations and mis-spec
void confirm_spec(pos_t upto);
void fail_spec(pos_t after);
```

New interface callbacks

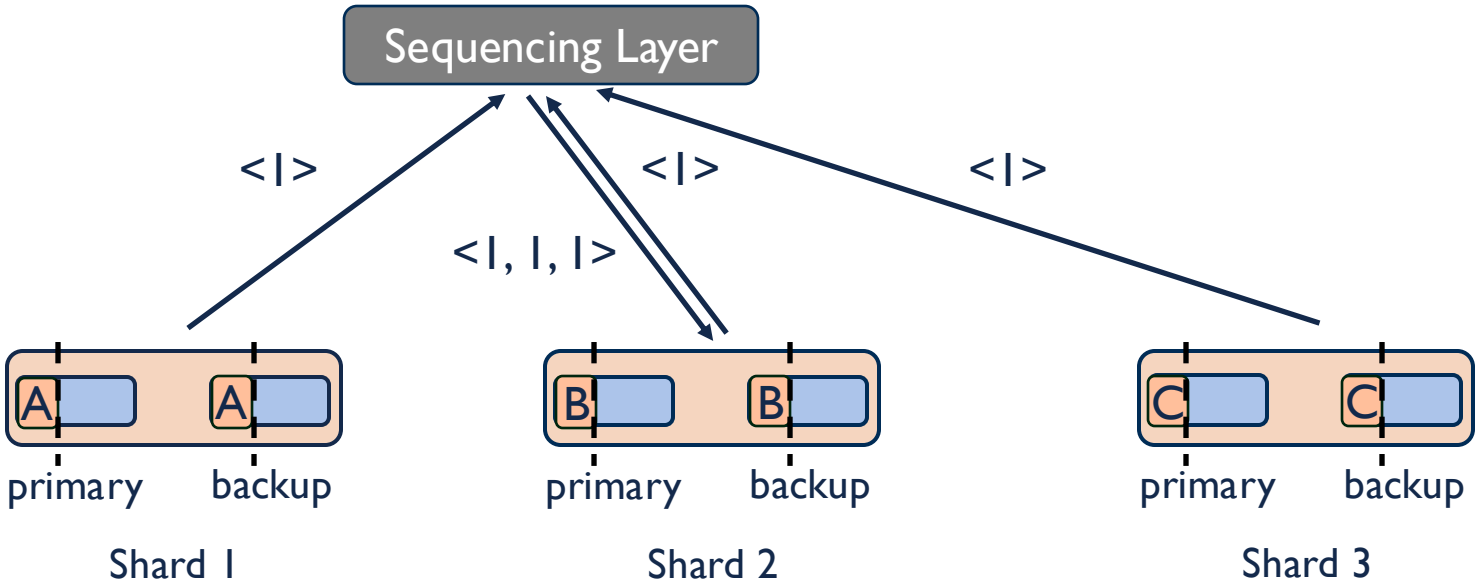
Challenge – Prediction is Hard

Hard to predict position in global order



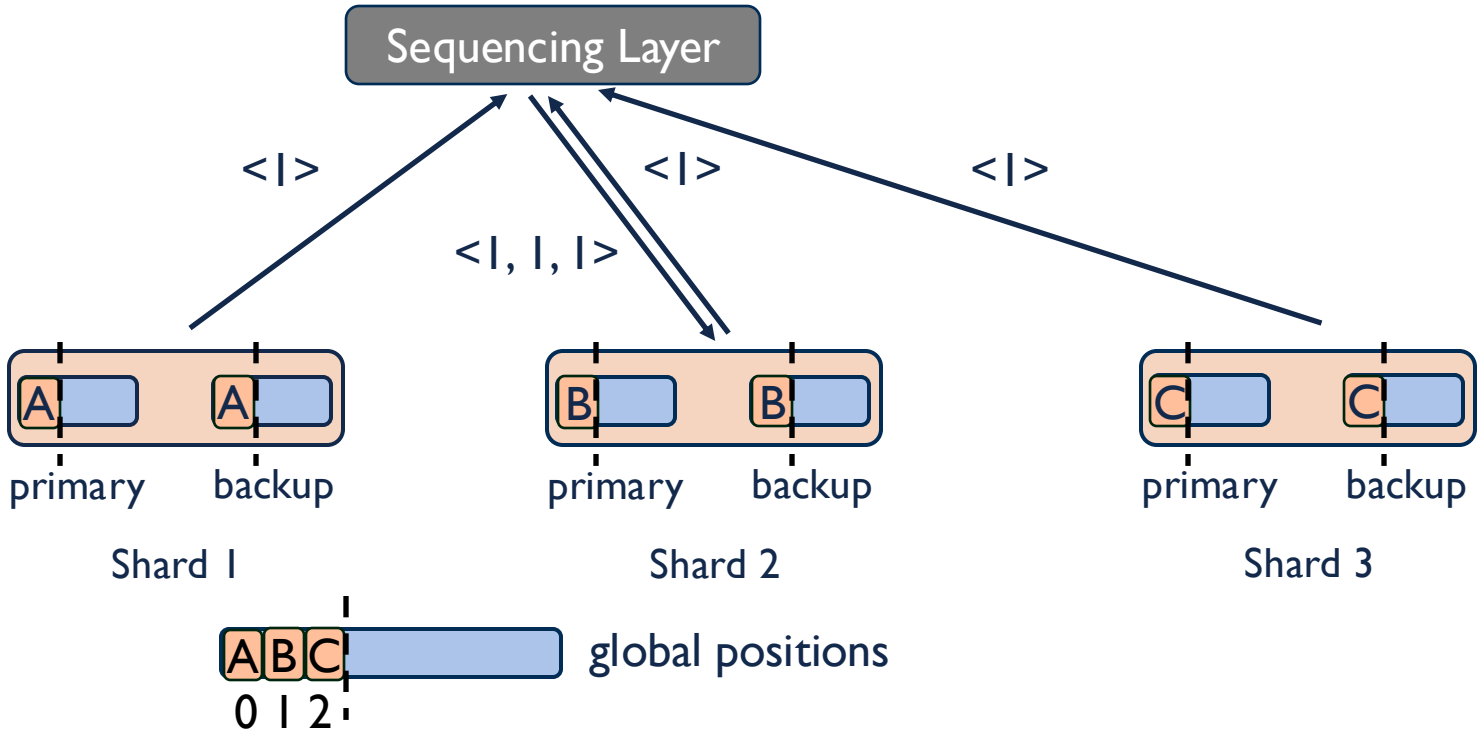
Challenge – Prediction is Hard

Hard to predict position in global order



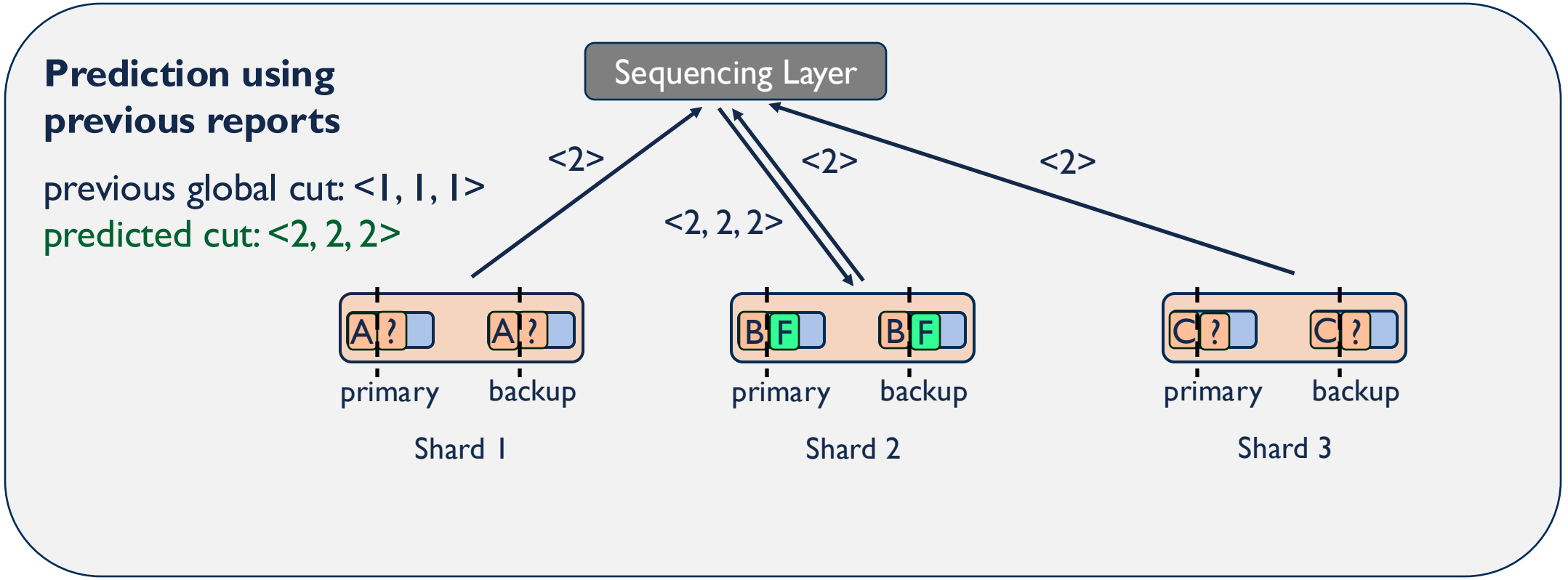
Challenge – Prediction is Hard

Hard to predict position in global order



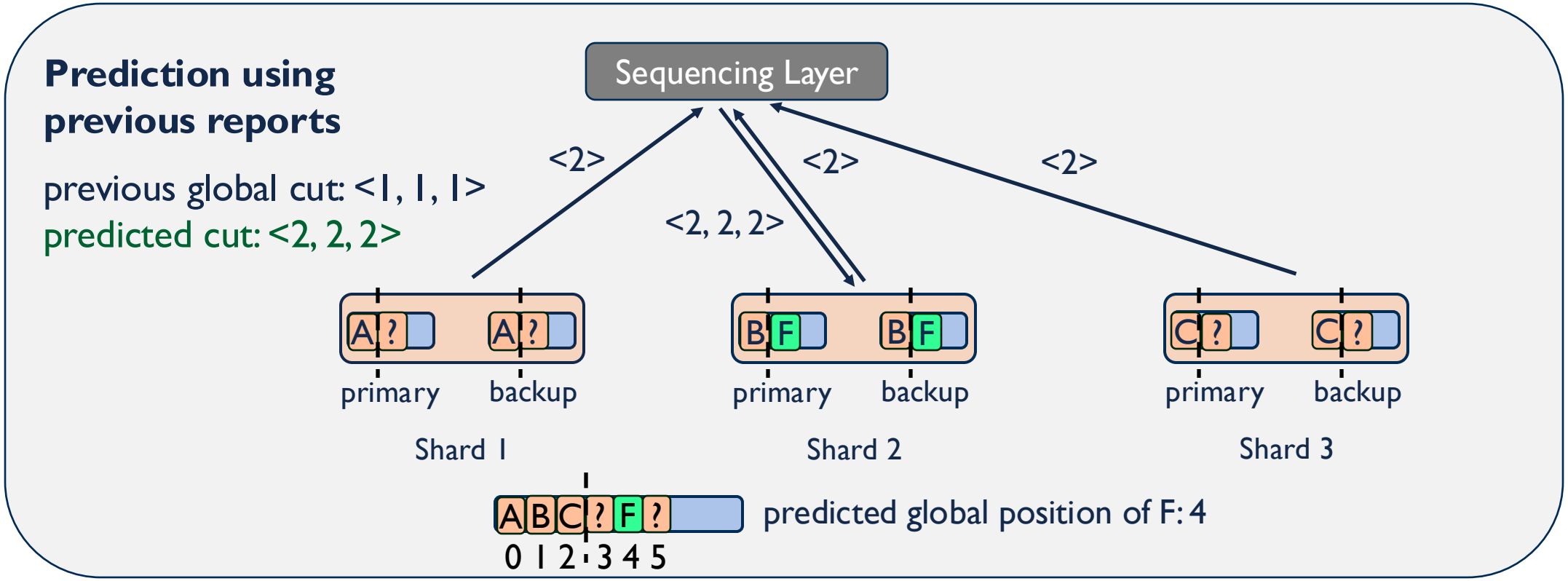
Challenge – Prediction is Hard

Hard to predict position in global order



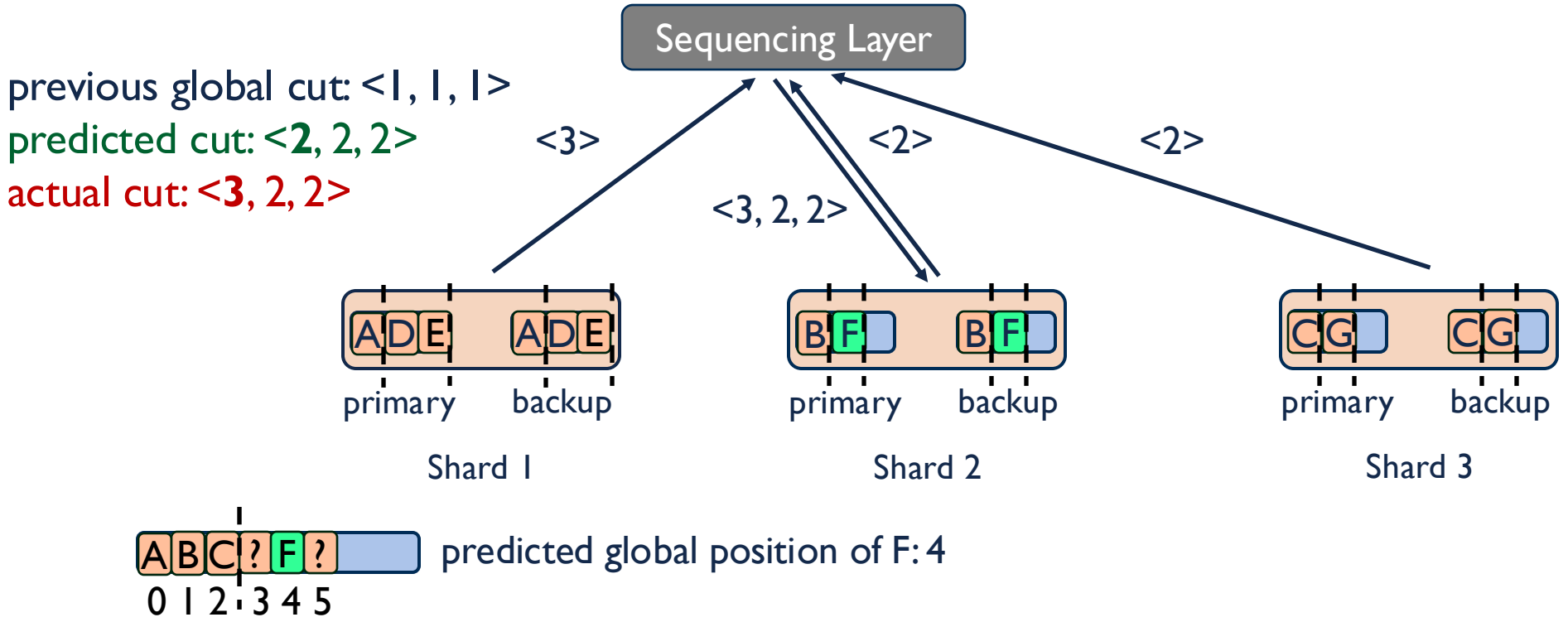
Challenge – Prediction is Hard

Hard to predict position in global order



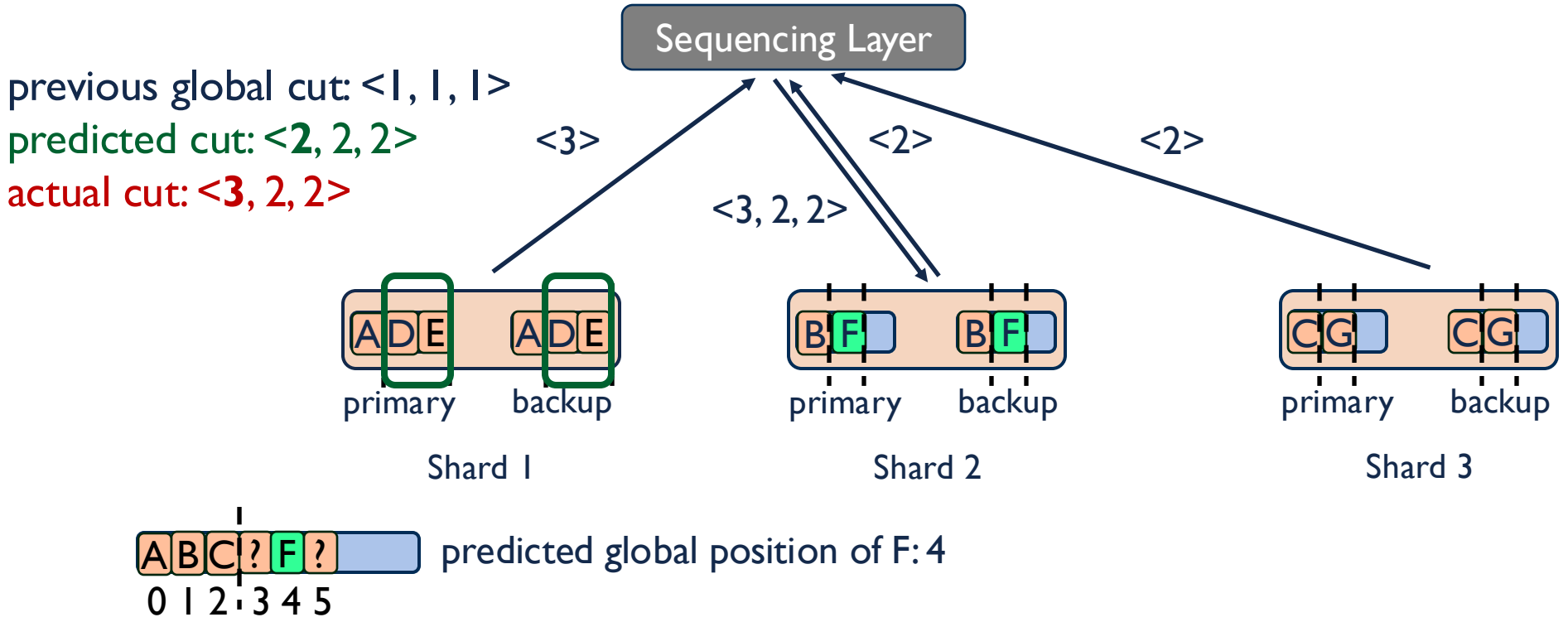
Challenge – Prediction is Hard

Hard to predict position in global order



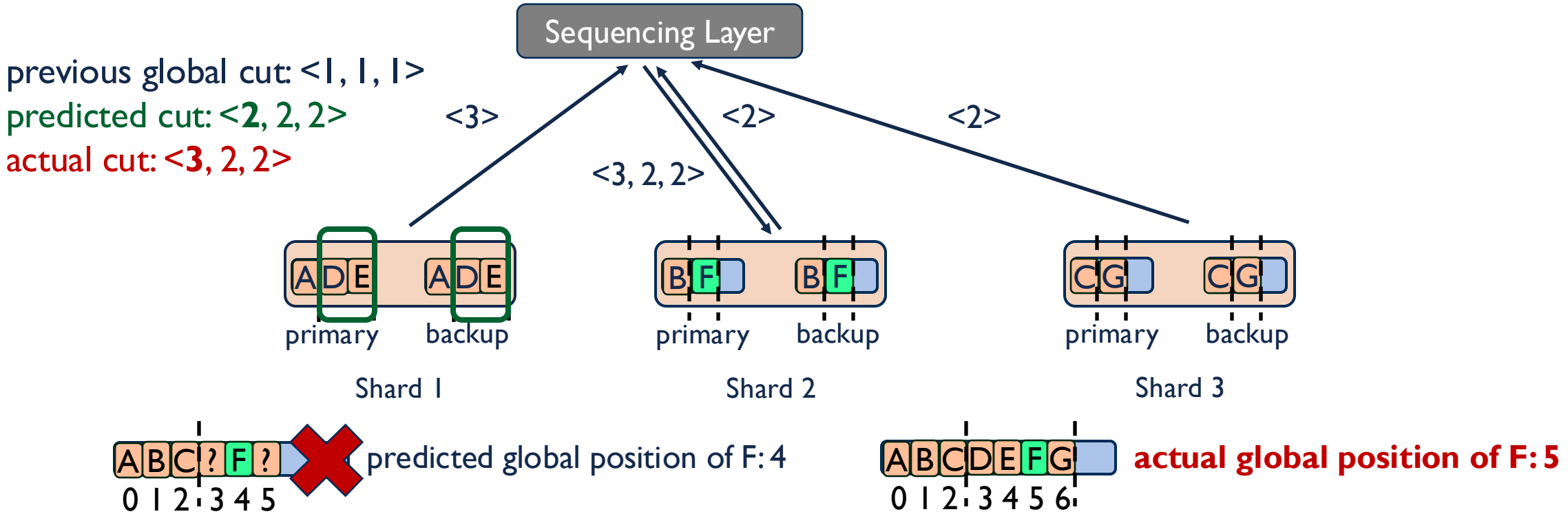
Challenge – Prediction is Hard

Hard to predict position in global order



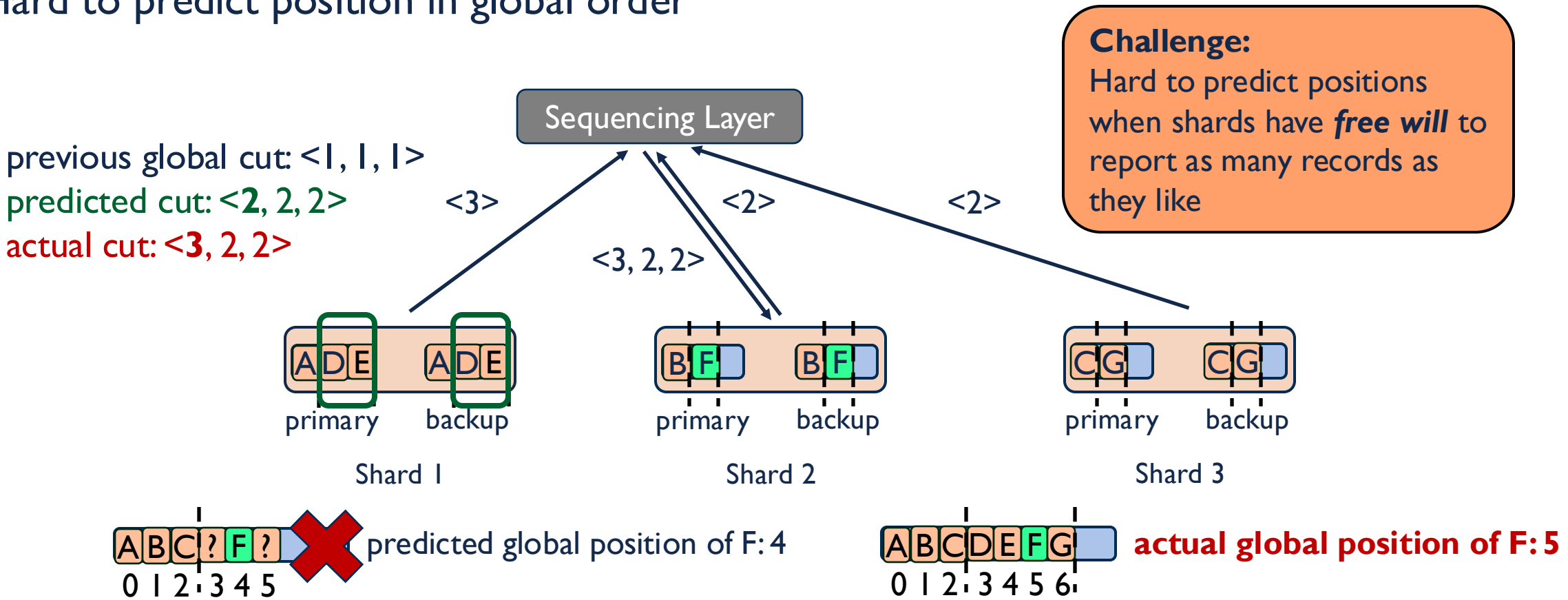
Challenge – Prediction is Hard

Hard to predict position in global order



Challenge – Prediction is Hard

Hard to predict position in global order



Challenge:
Hard to predict positions when shards have ***free will*** to report as many records as they like

Idea – Fix-Ante Ordering

Idea – Fix-Ante Ordering

predetermined cuts – fix beforehand global cuts system is expected to produce

Idea – Fix-Ante Ordering

predetermined cuts – fix beforehand global cuts system is expected to produce

quota – fixed number of records in each shard report

Idea – Fix-Ante Ordering

predetermined cuts – fix beforehand global cuts system is expected to produce

quota – fixed number of records in each shard report

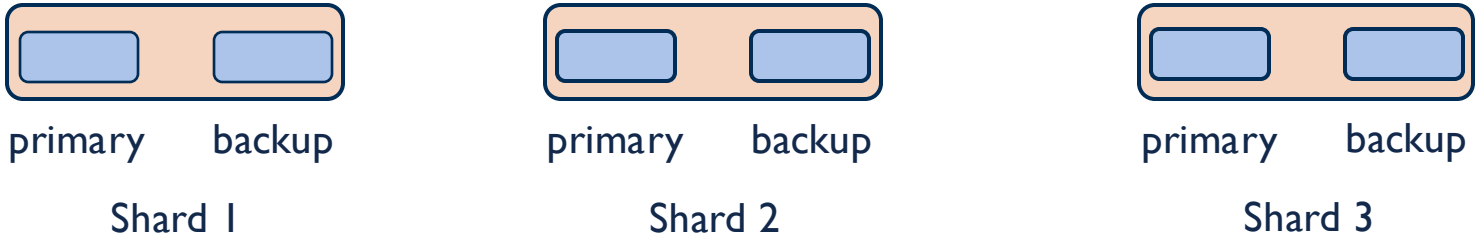
Sequencing layer waits for predetermined cut before sending global cut

Idea – Fix-Ante Ordering

predetermined cuts – fix beforehand global cuts system is expected to produce

quota – fixed number of records in each shard report

Sequencing layer waits for predetermined cut before sending global cut

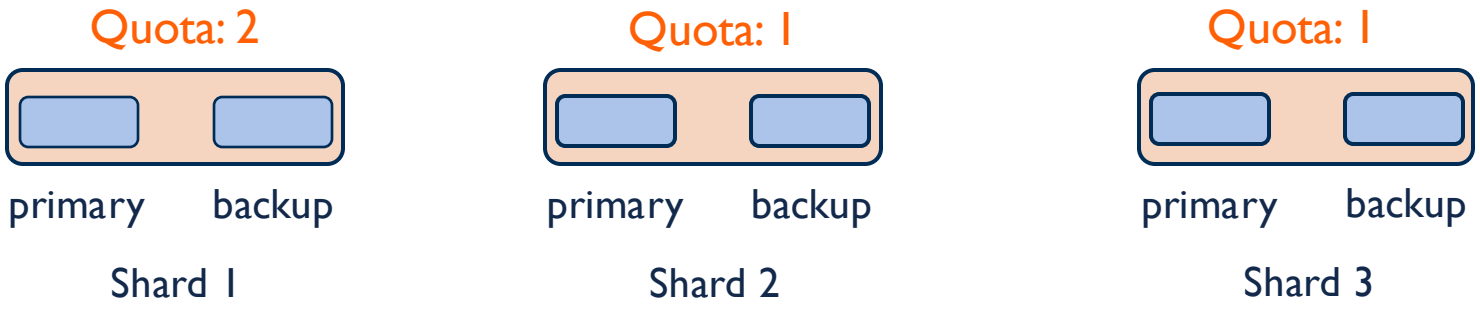


Idea – Fix-Ante Ordering

predetermined cuts – fix beforehand global cuts system is expected to produce

quota – fixed number of records in each shard report

Sequencing layer waits for predetermined cut before sending global cut

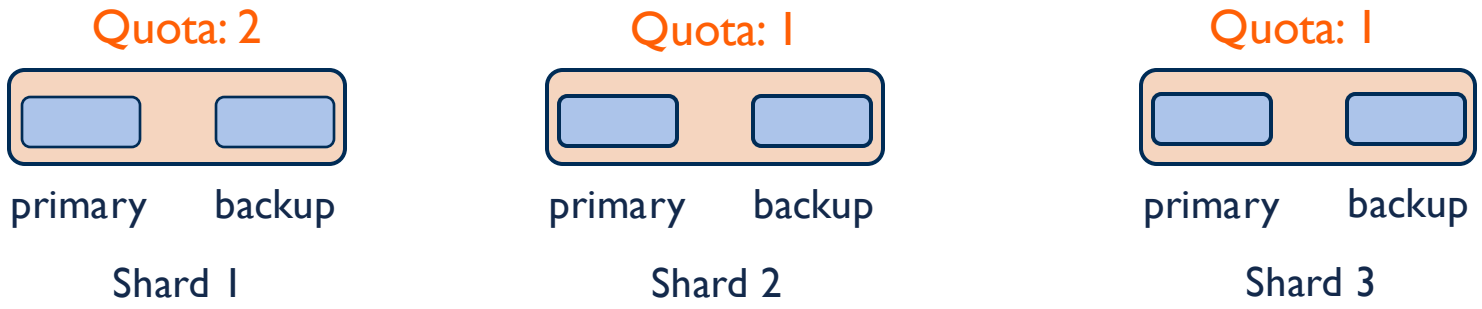


Idea – Fix-Ante Ordering

predetermined cuts – fix beforehand global cuts system is expected to produce

quota – fixed number of records in each shard report

Sequencing layer waits for predetermined cut before sending global cut

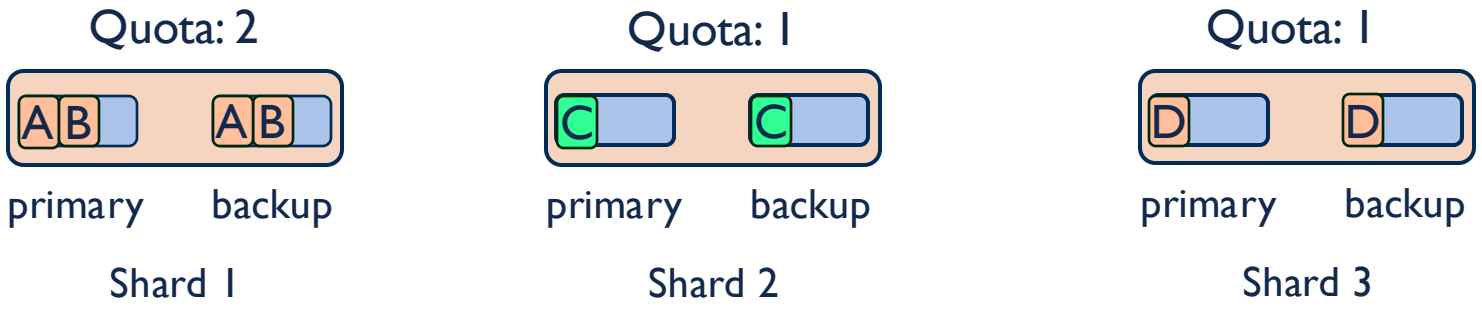


Predetermined cuts - <2, 1, 1> <4, 2, 2> <6, 3, 3> … <2n, n, n>

Idea – Fix-Ante Ordering

Each shard knows the sequence of global cuts the system will produce

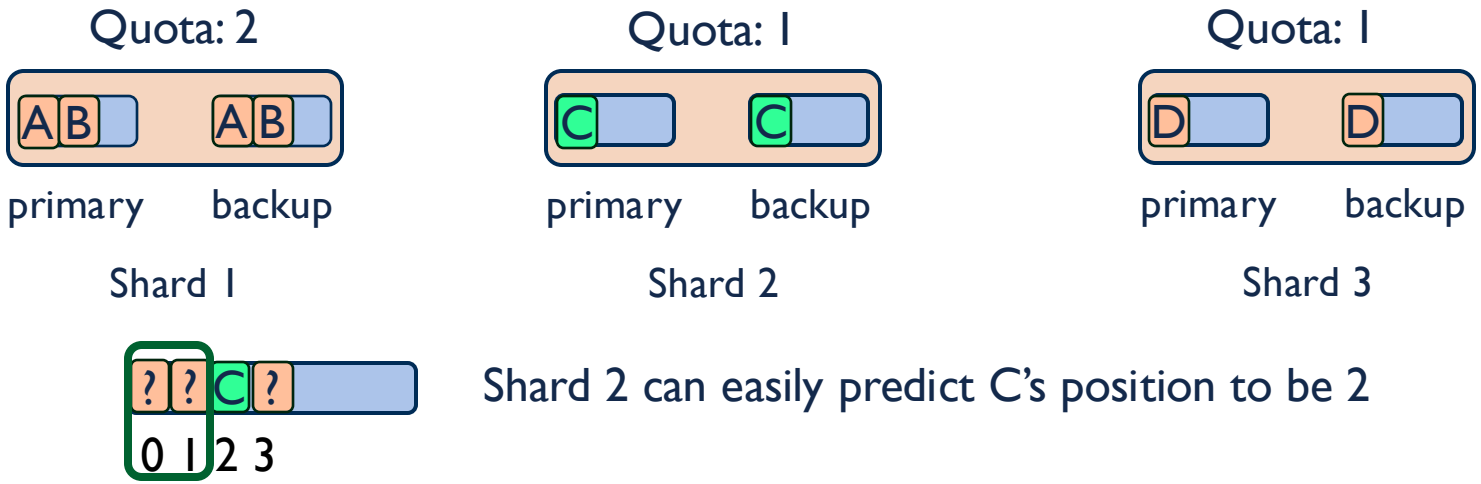
Any shard can predict positions of its records easily!



Idea – Fix-Ante Ordering

Each shard knows the sequence of global cuts the system will produce

Any shard can predict positions of its records easily!

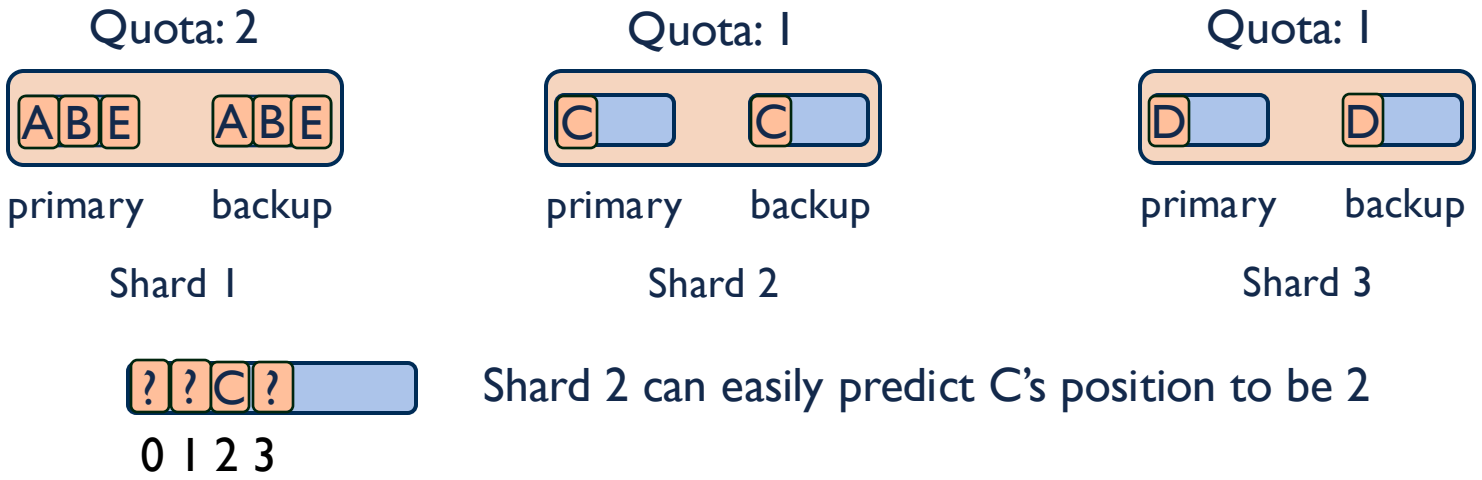


Idea – Fix-Ante Ordering

Each shard knows the sequence of global cuts the system will produce

Any shard can predict positions of its records easily!

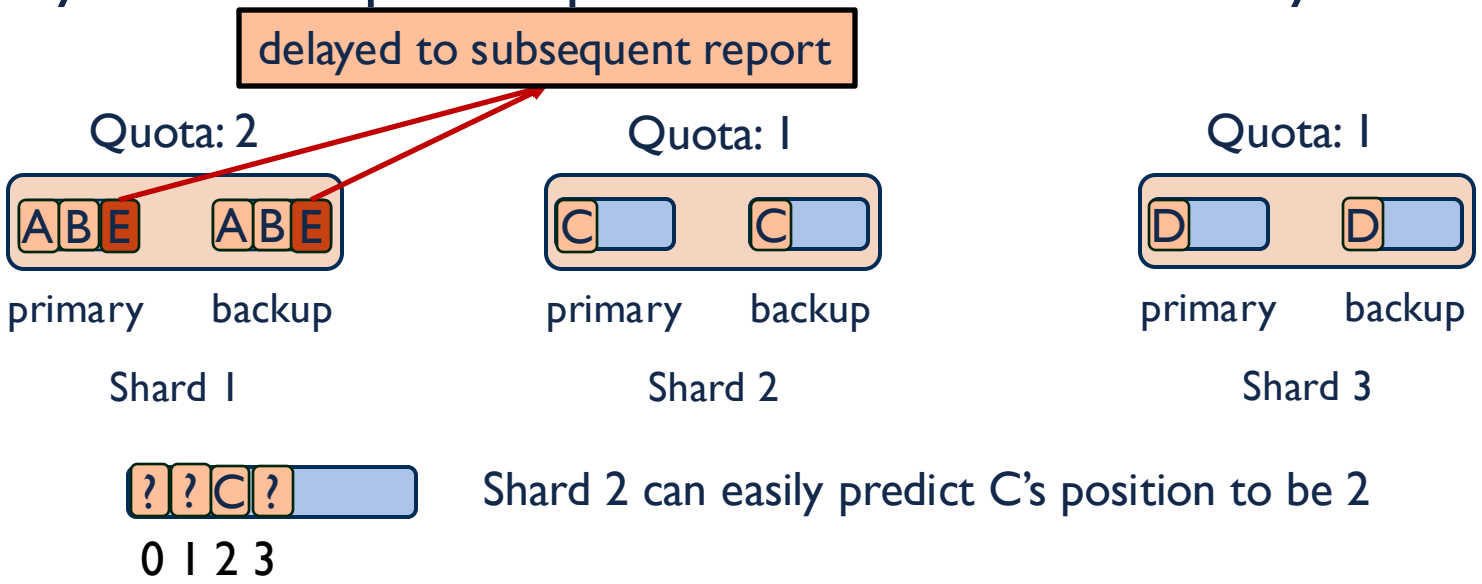
What if a shard has more records to report?



Idea – Fix-Ante Ordering

Each shard knows the sequence of global cuts the system will produce

Any shard can predict positions of its records easily!

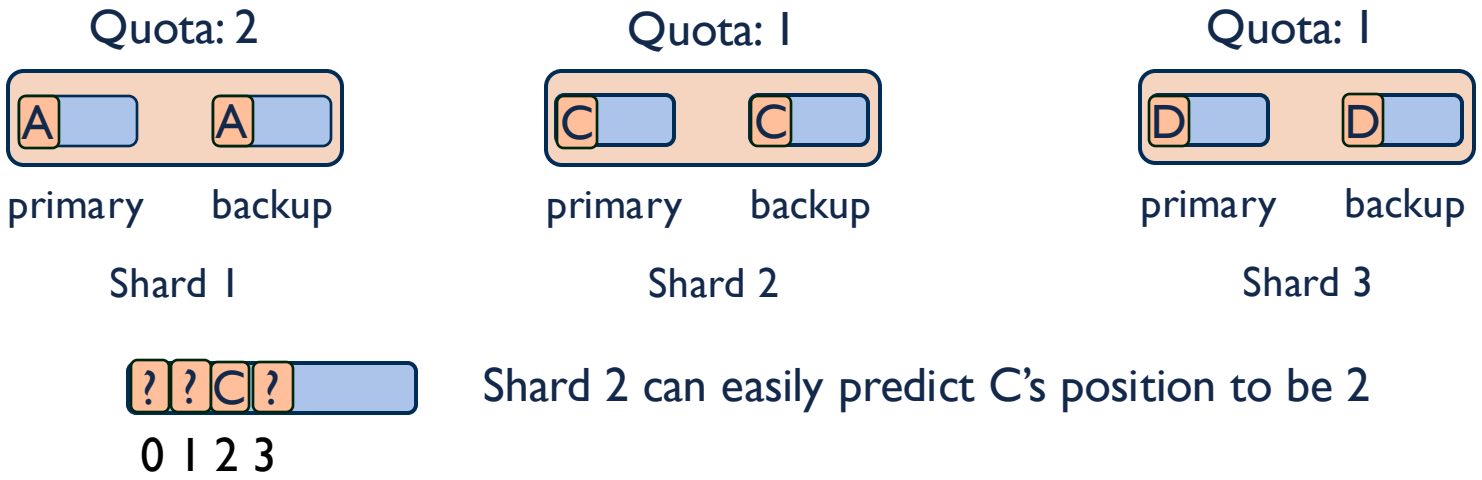


What if a shard has more records to report?
Solution:
Delay to subsequent reports

Idea – Fix-Ante Ordering

Each shard knows the sequence of global cuts the system will produce

Any shard can predict positions of its records easily!



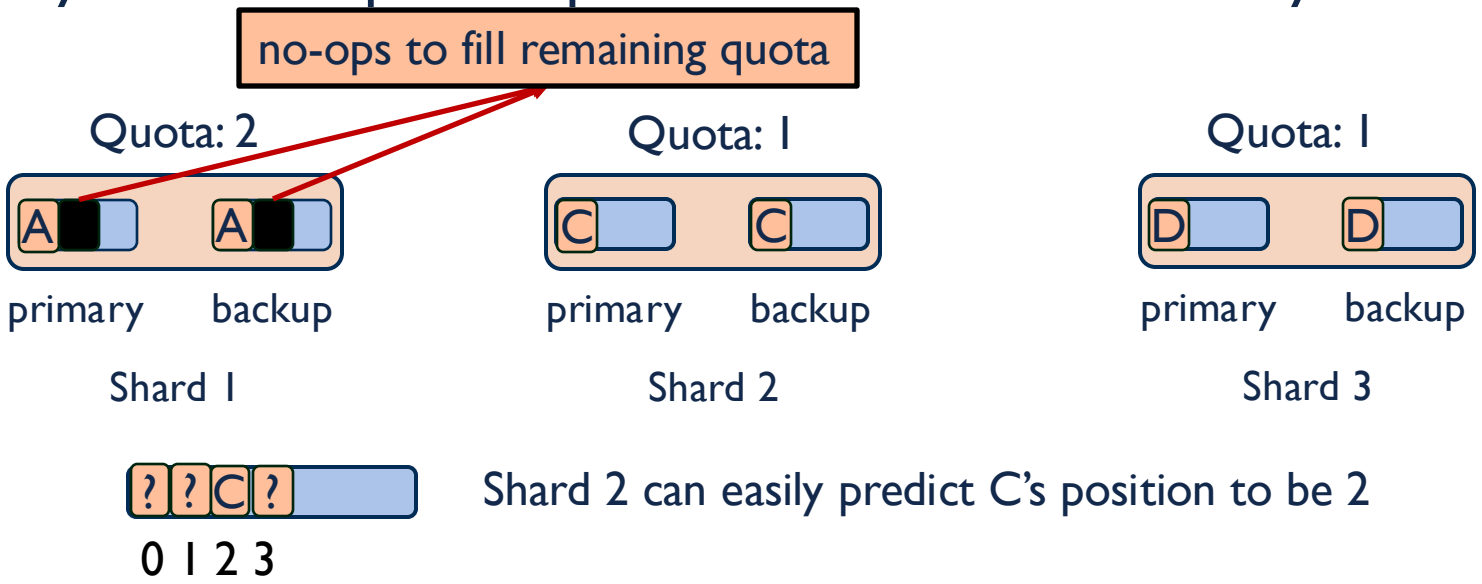
What if a shard has more records to report?
Solution:
Delay to subsequent reports

What if a shard does not have enough records?

Idea – Fix-Ante Ordering

Each shard knows the sequence of global cuts the system will produce

Any shard can predict positions of its records easily!



What if a shard has more records to report?
Solution:
Delay to subsequent reports

What if a shard does not have enough records?
Solution:
Pad quota with no-ops

If Predetermined, Why Speculate!?

If Predetermined, Why Speculate!?

If shards meet quota, predetermined order same as actual global order

If Predetermined, Why Speculate!?

If shards meet quota, predetermined order same as actual global order

In this case → speculation succeeds

If Predetermined, Why Speculate!?

If shards meet quota, predetermined order same as actual global order

In this case → speculation succeeds

If not → speculation fails

If Predetermined, Why Speculate!?

If shards meet quota, predetermined order same as actual global order

In this case → speculation succeeds

If not → speculation fails

Cases:

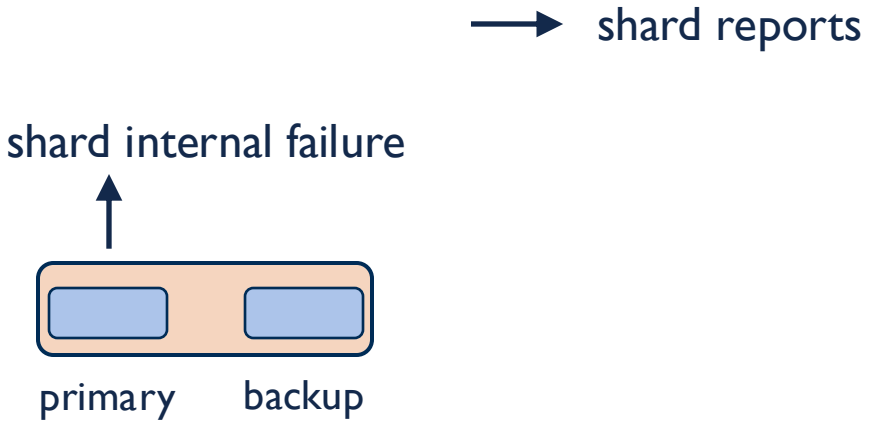
If Predetermined, Why Speculate!?

If shards meet quota, predetermined order same as actual global order

In this case → speculation succeeds

If not → speculation fails

Cases:



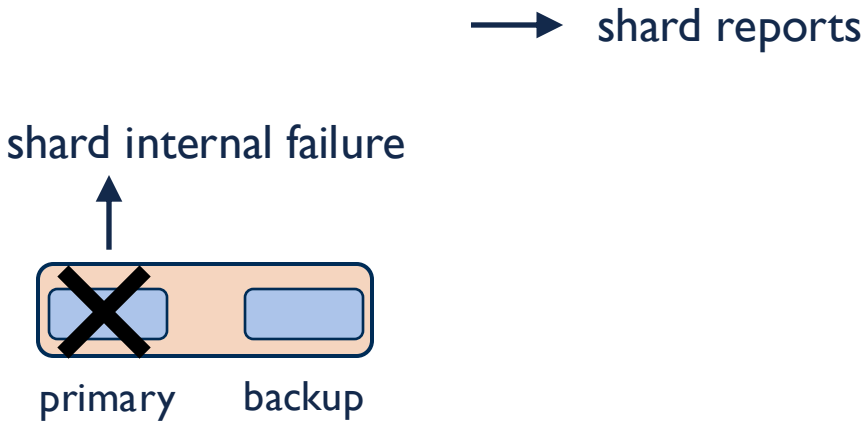
If Predetermined, Why Speculate!?

If shards meet quota, predetermined order same as actual global order

In this case → speculation succeeds

If not → speculation fails

Cases:



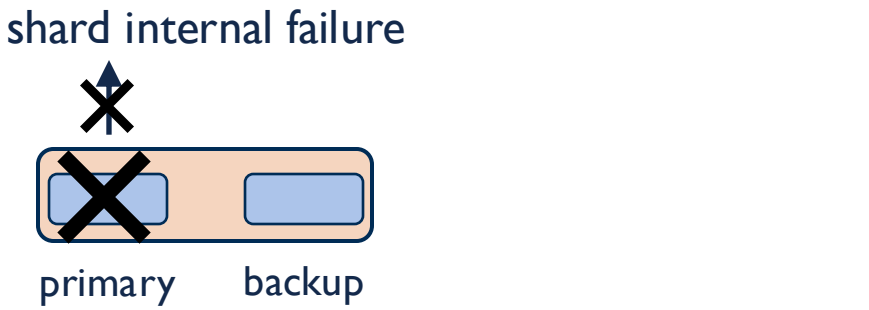
If Predetermined, Why Speculate!?

If shards meet quota, predetermined order same as actual global order

In this case → speculation succeeds

If not → speculation fails

Cases:



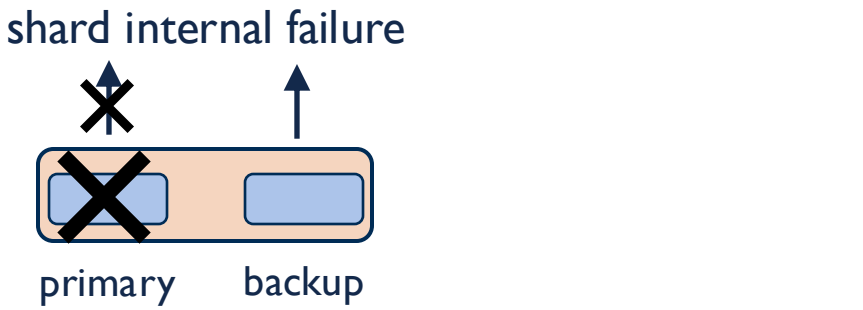
If Predetermined, Why Speculate!?

If shards meet quota, predetermined order same as actual global order

In this case → speculation succeeds

If not → speculation fails

Cases:



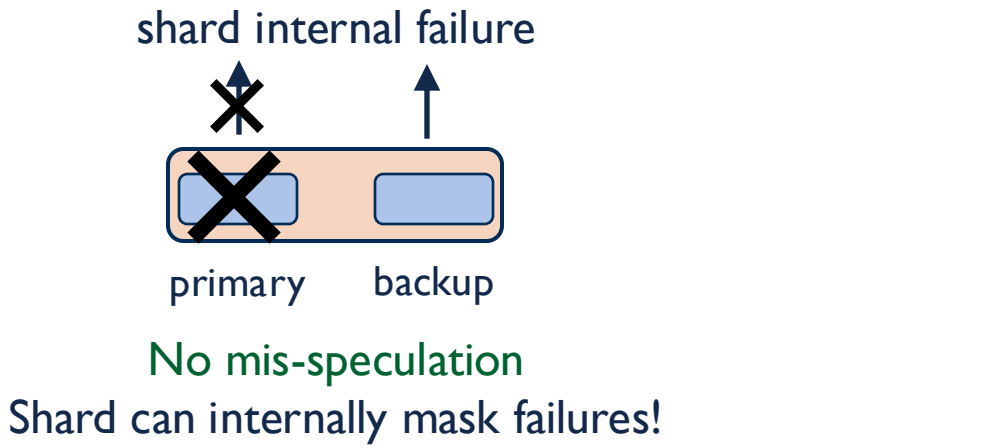
If Predetermined, Why Speculate!?

If shards meet quota, predetermined order same as actual global order

In this case → speculation succeeds

If not → speculation fails

Cases:



No mis-speculation

Shard can internally mask failures!

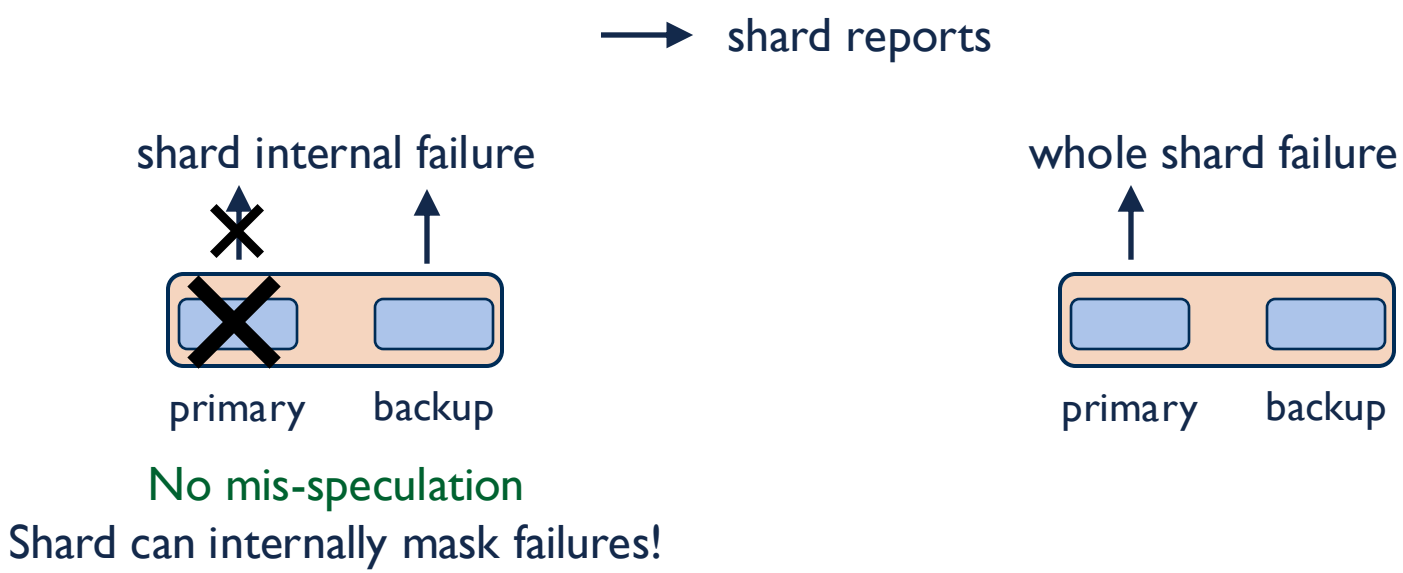
If Predetermined, Why Speculate!?

If shards meet quota, predetermined order same as actual global order

In this case → speculation succeeds

If not → speculation fails

Cases:



No mis-speculation

Shard can internally mask failures!

If Predetermined, Why Speculate!?

If shards meet quota, predetermined order same as actual global order

In this case → speculation succeeds

If not → speculation fails

Cases:

No mis-speculation

Shard can internally mask failures!

If Predetermined, Why Speculate!?

If shards meet quota, predetermined order same as actual global order

In this case → speculation succeeds

If not → speculation fails

Cases:

No mis-speculation

Shard can internally mask failures!

If Predetermined, Why Speculate!?

If shards meet quota, predetermined order same as actual global order

In this case → speculation succeeds

If not → speculation fails

Cases:

If Predetermined, Why Speculate!?

If shards meet quota, predetermined order same as actual global order

In this case → speculation succeeds

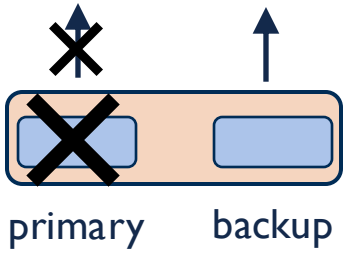
If not → speculation fails

Cases:

→ shard reports

Rare!

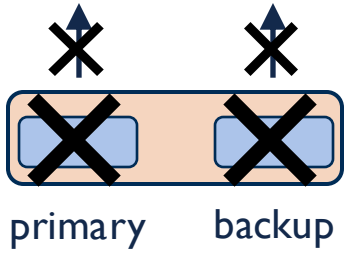
shard internal failure



No mis-speculation

Shard can internally mask failures!

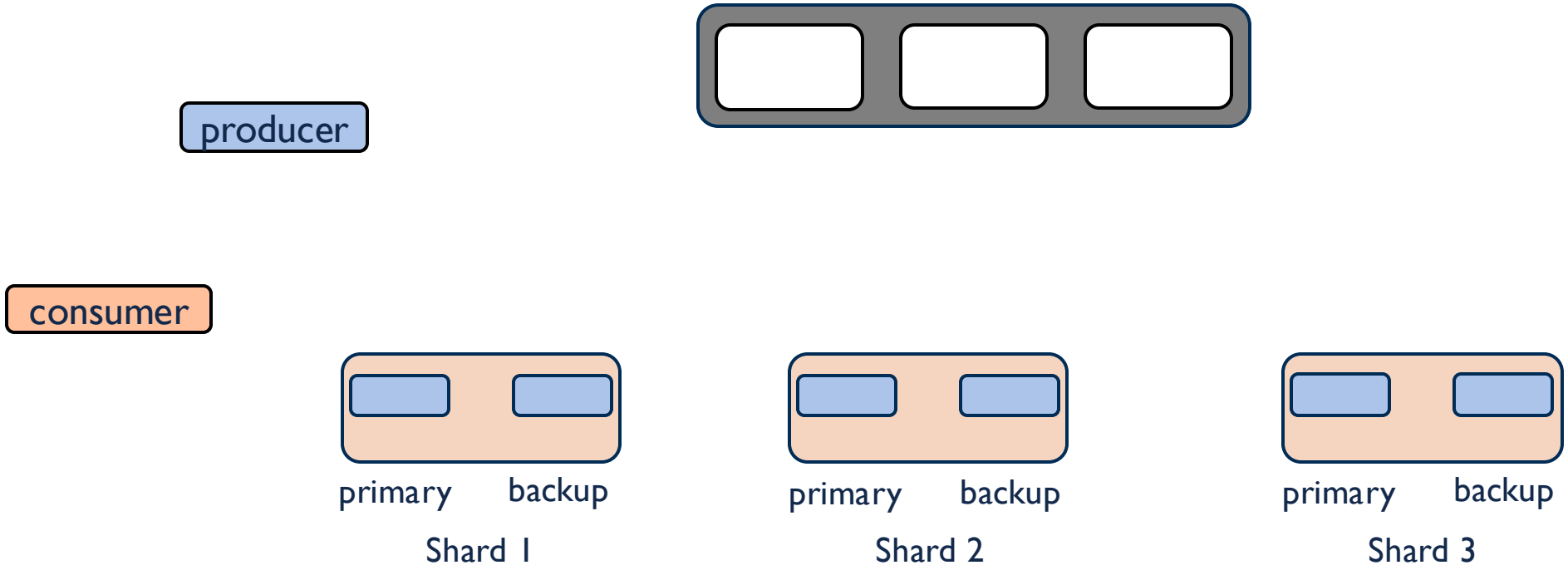
whole shard failure



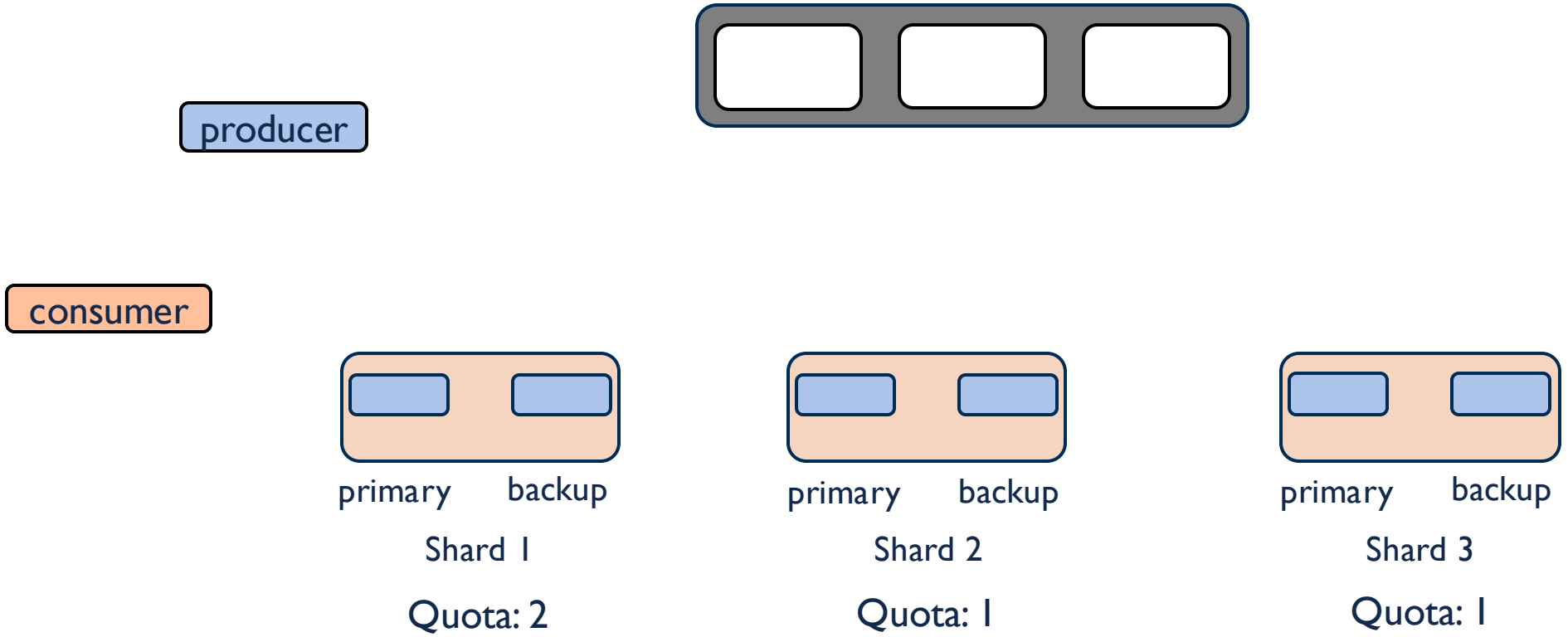
Mis-speculations occur

Shard cannot meet quota

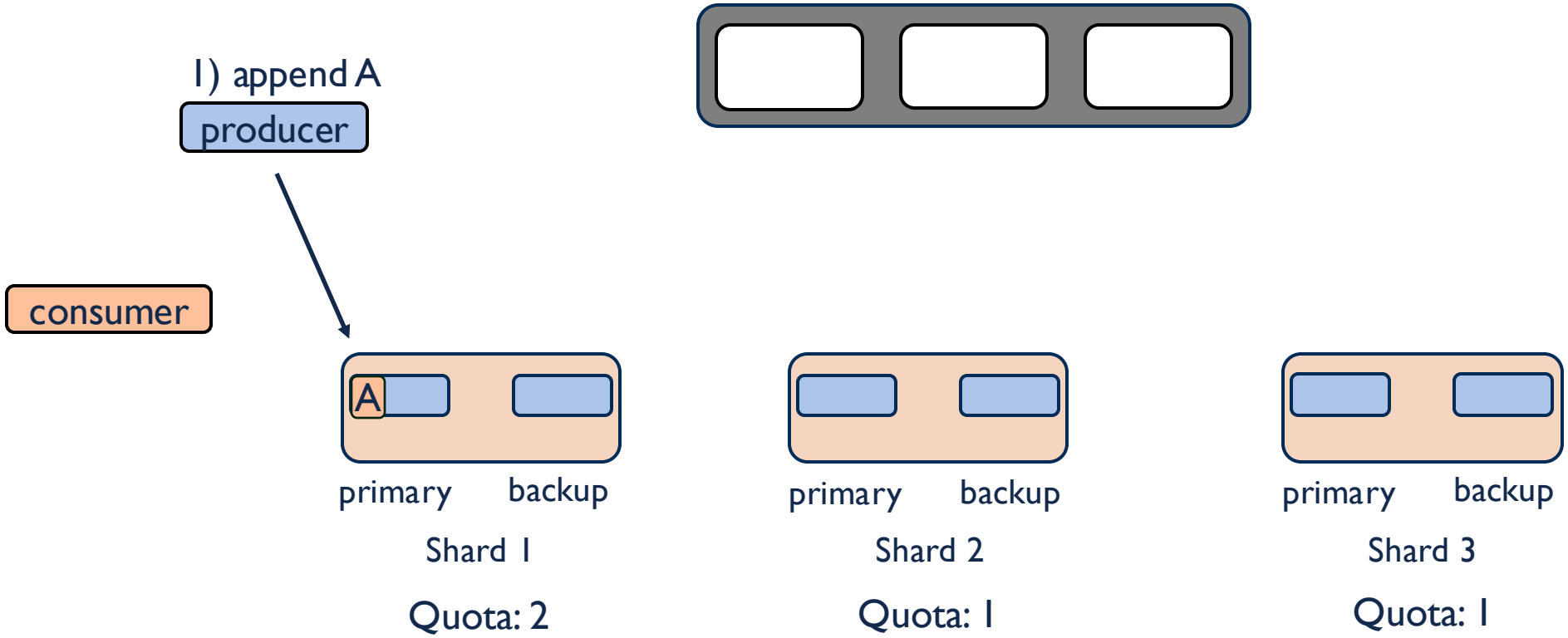
Real-Time App atop SpecLog



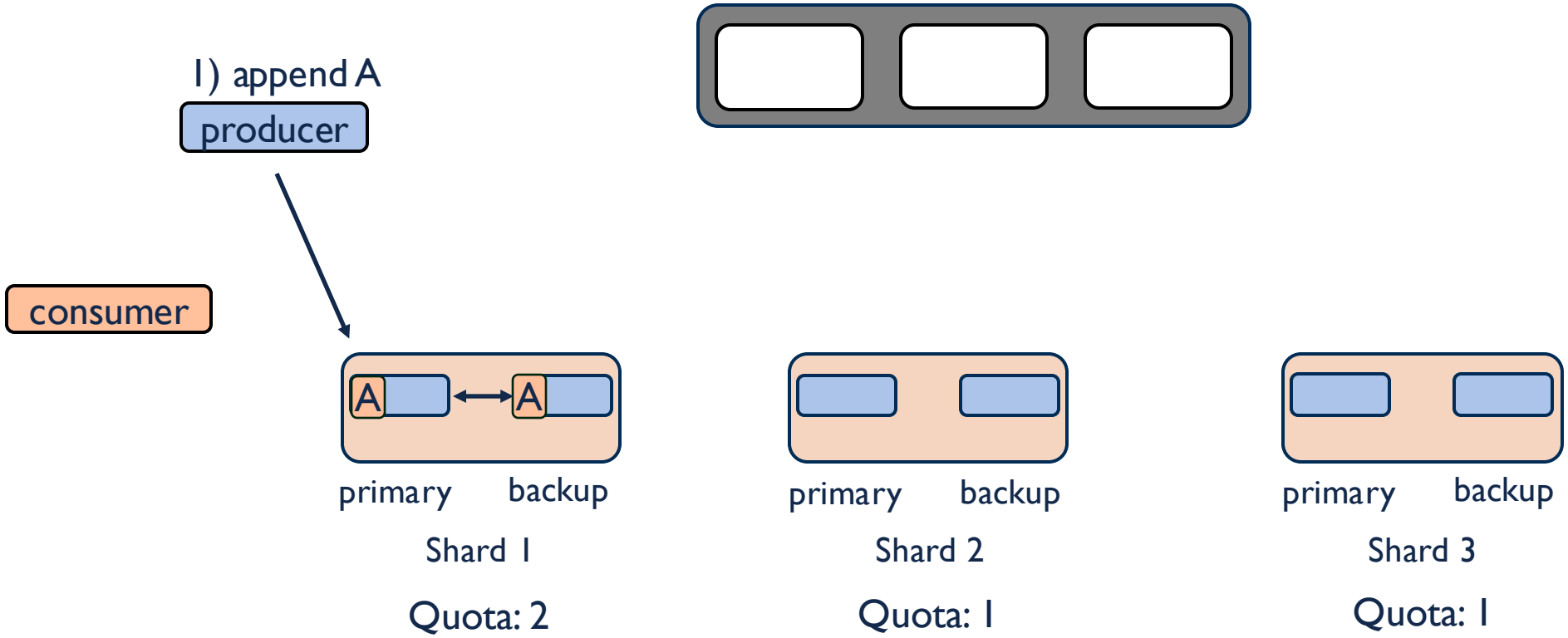
Real-Time App atop SpecLog



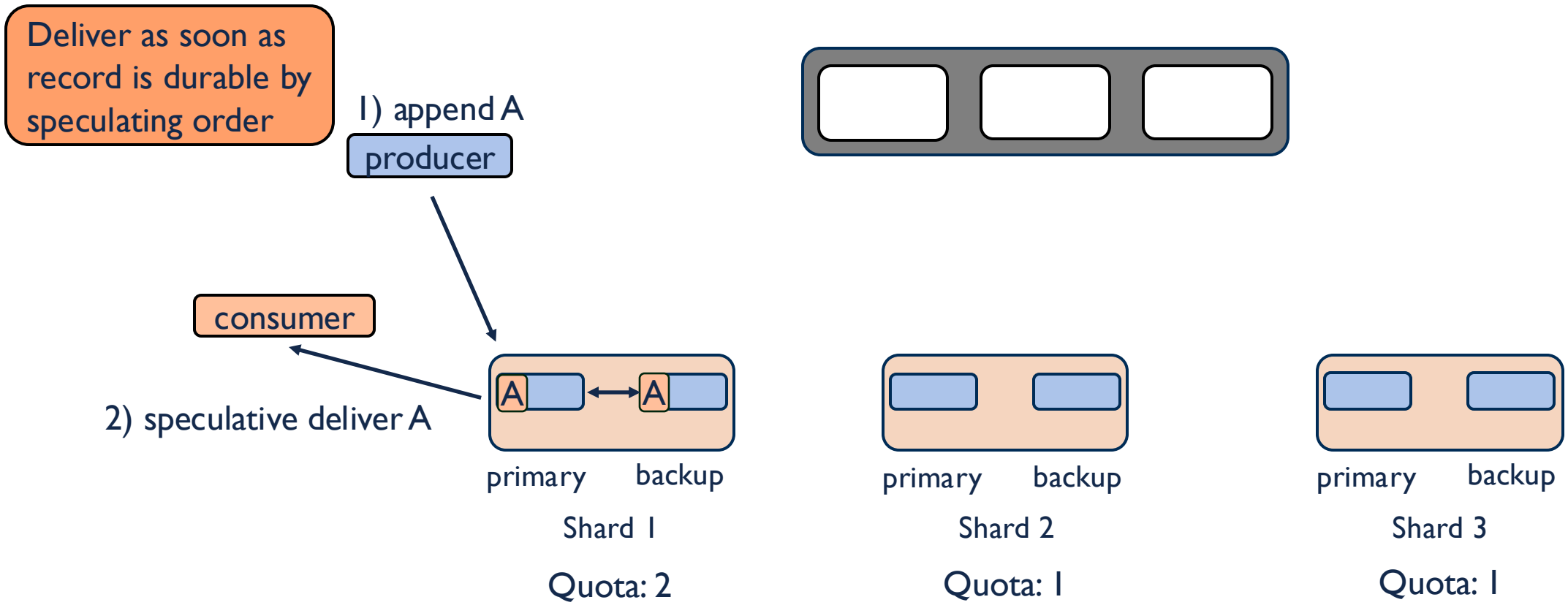
Real-Time App atop SpecLog



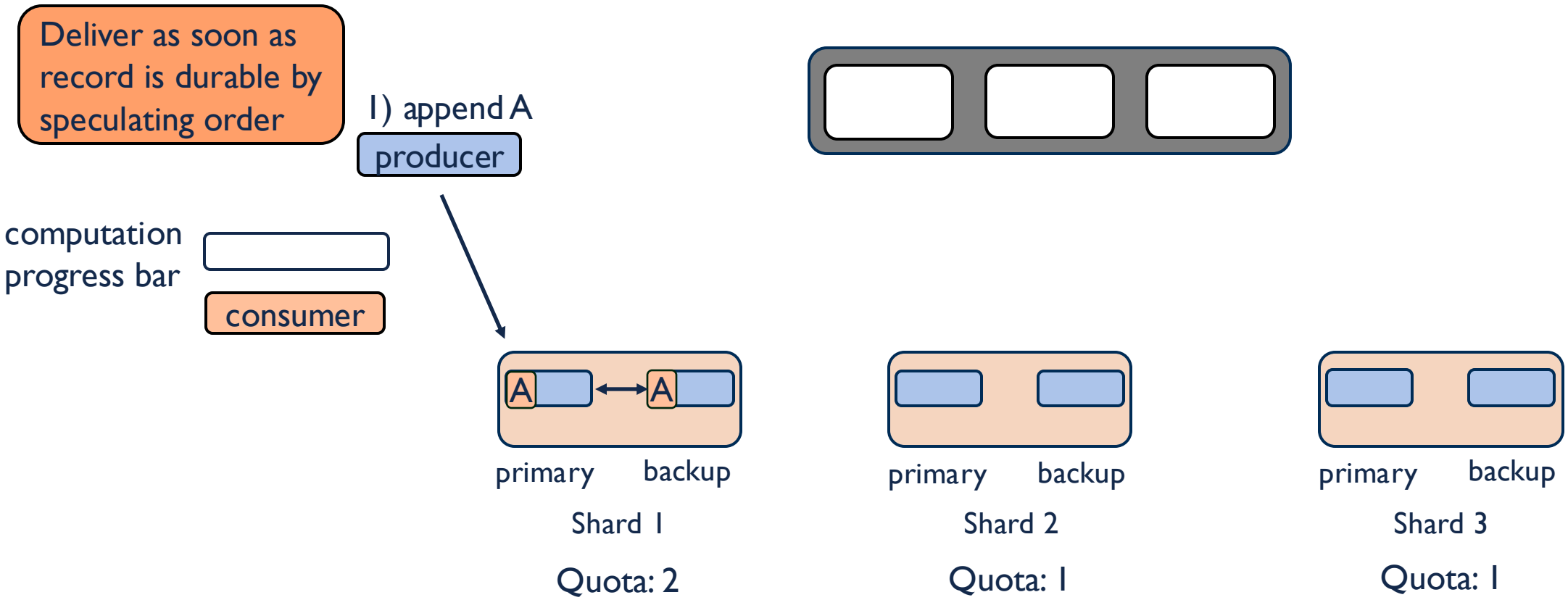
Real-Time App atop SpecLog



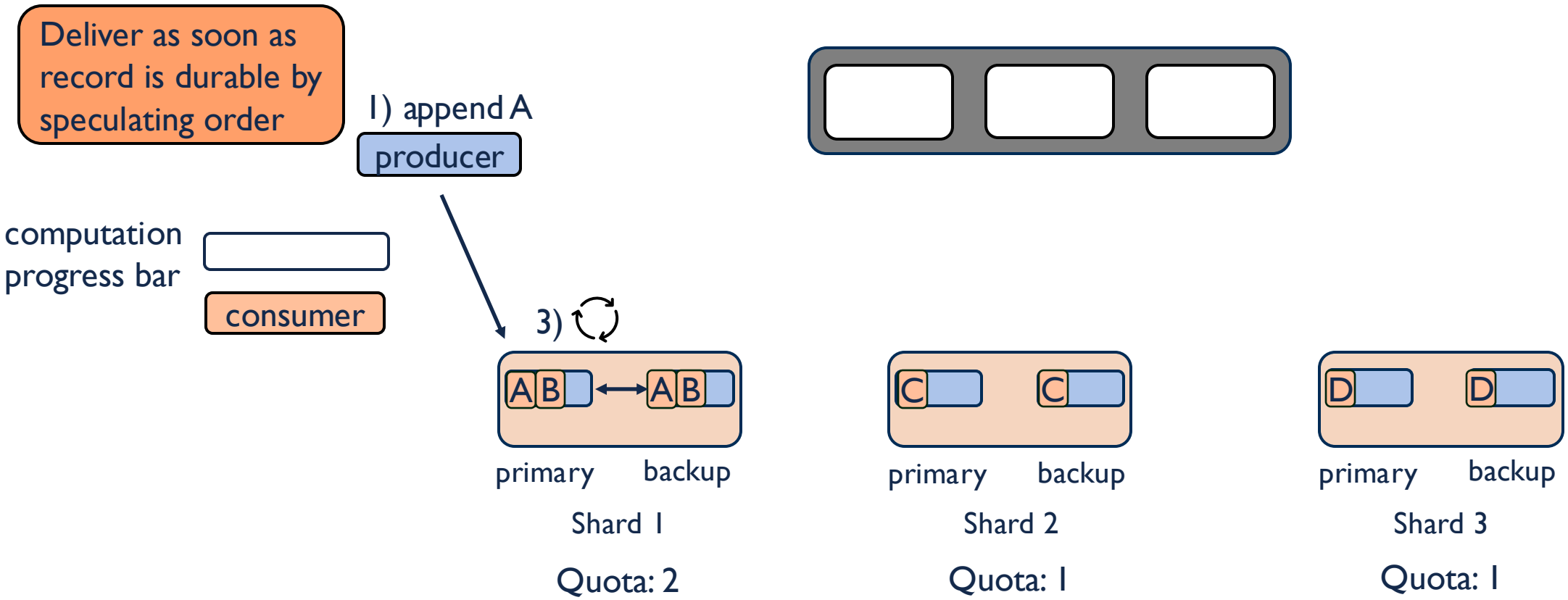
Real-Time App atop SpecLog



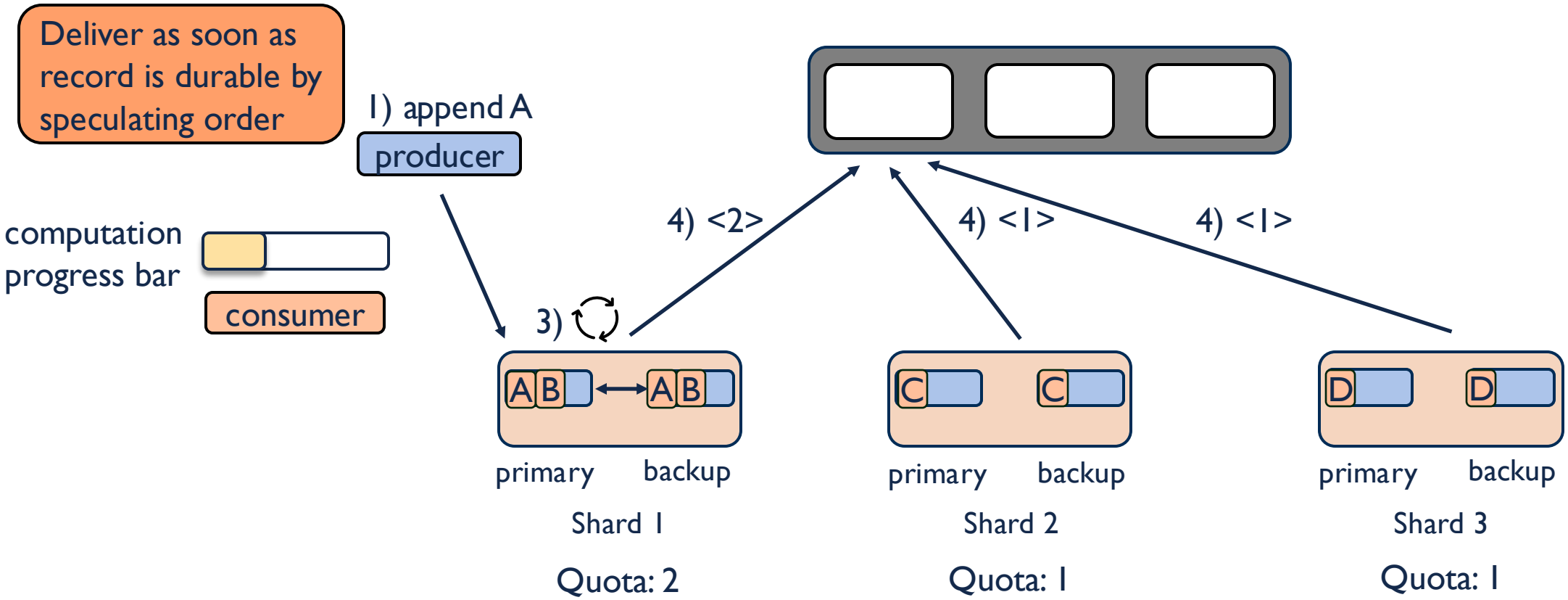
Real-Time App atop SpecLog



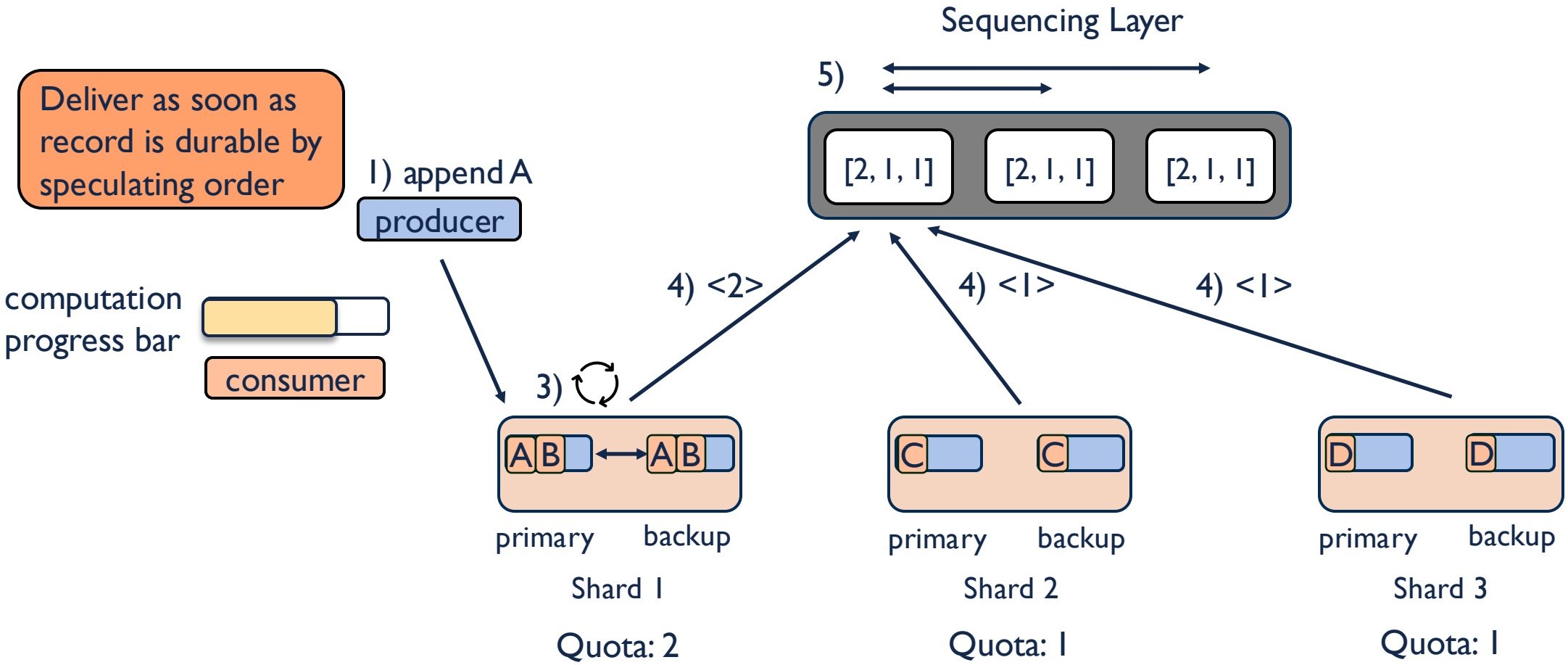
Real-Time App atop SpecLog



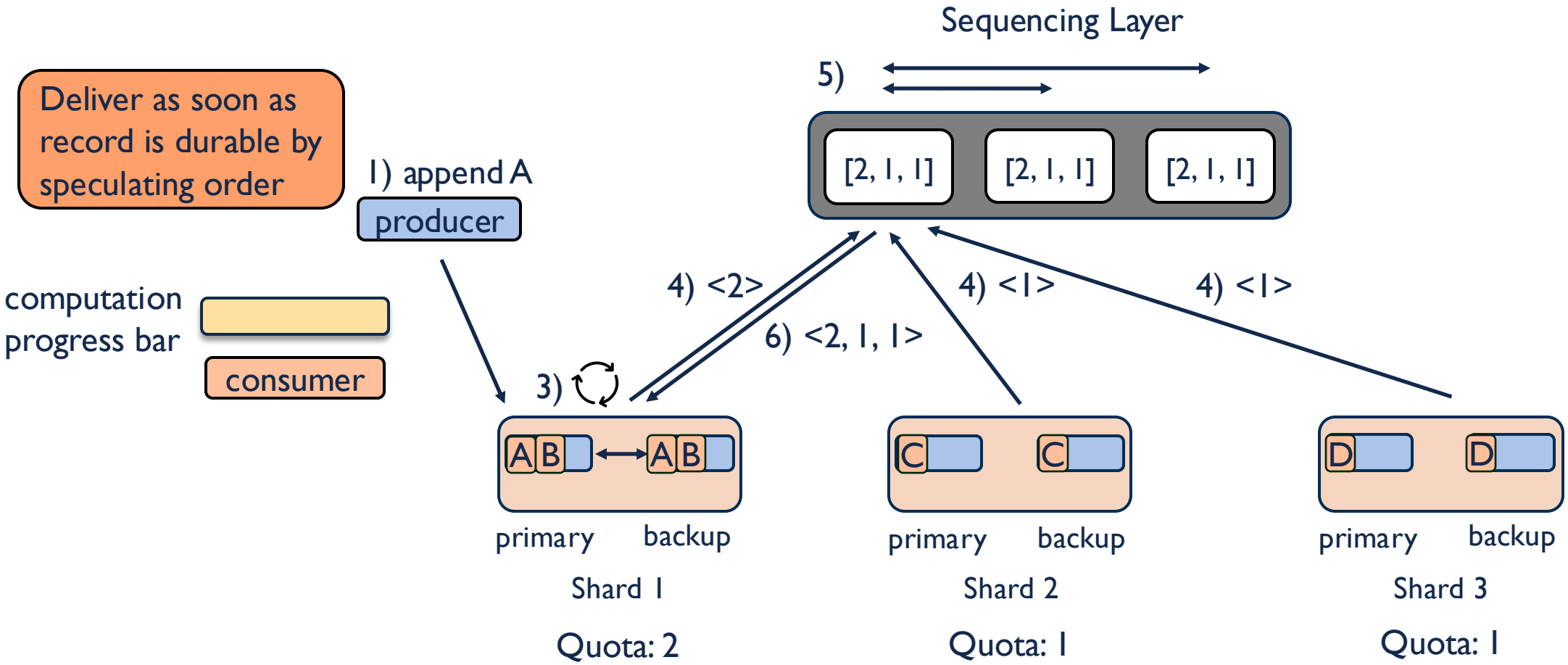
Real-Time App atop SpecLog



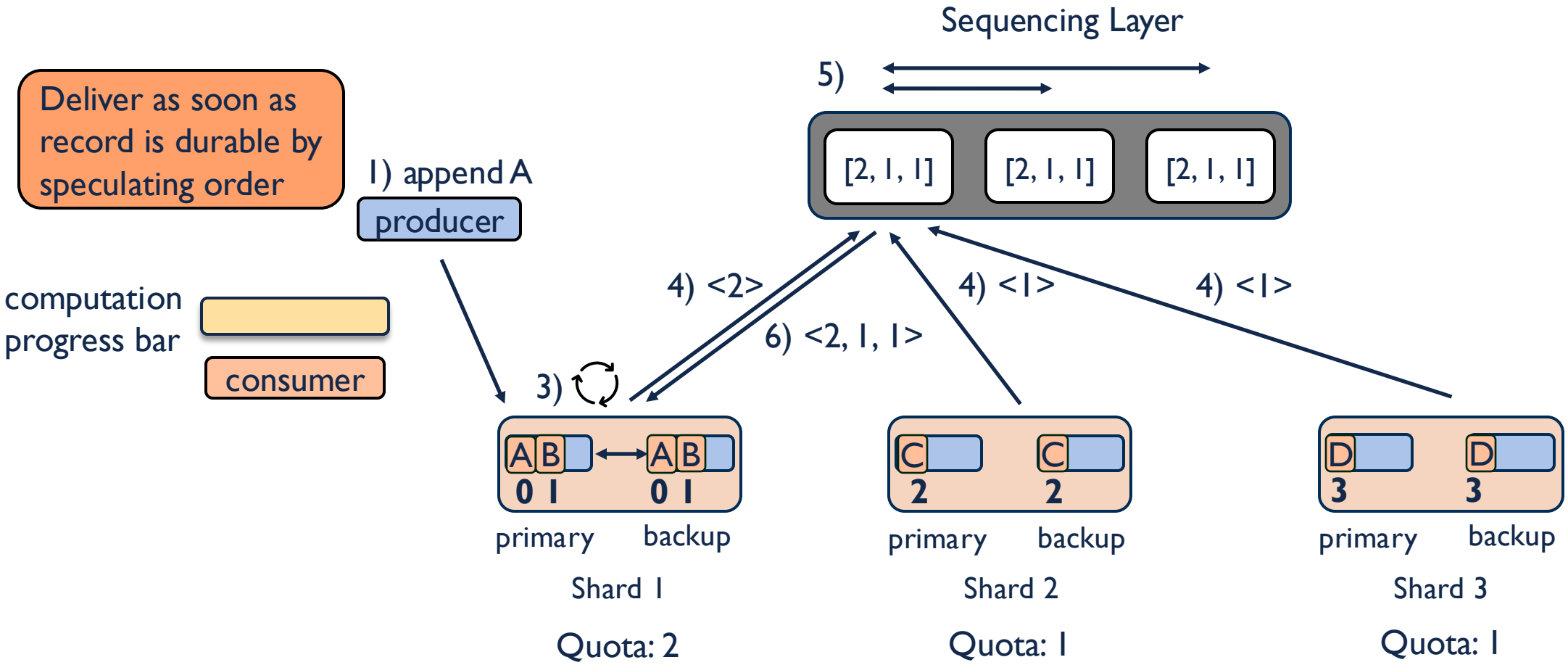
Real-Time App atop SpecLog



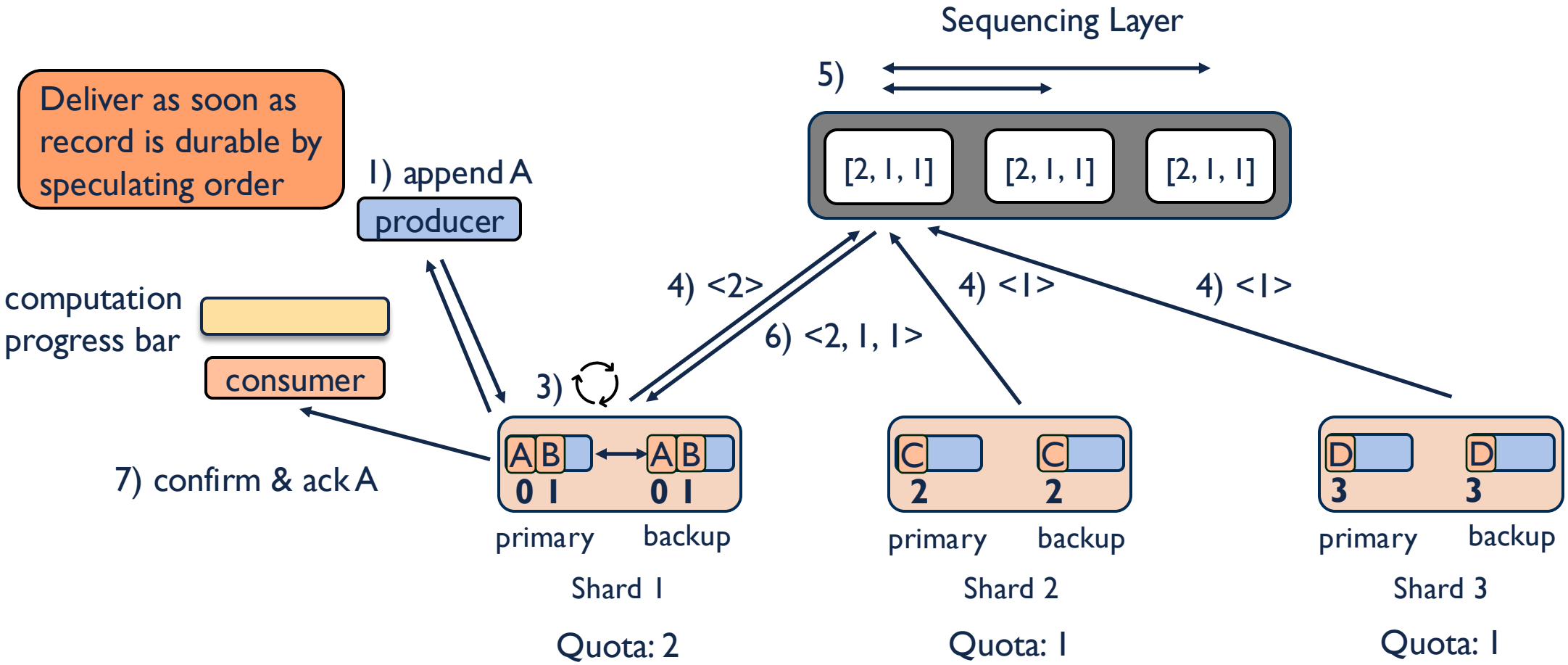
Real-Time App atop SpecLog



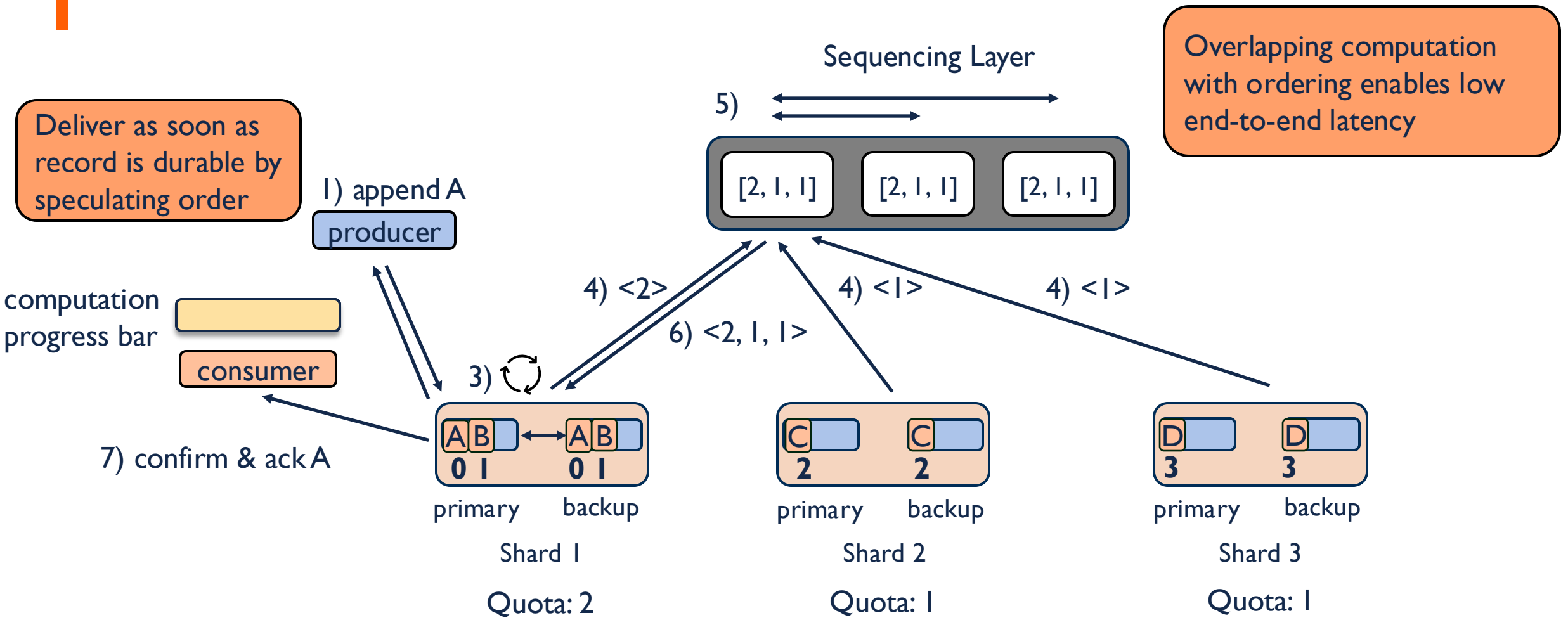
Real-Time App atop SpecLog



Real-Time App atop SpecLog



Real-Time App atop SpecLog



Outline

- Introduction
- Motivation
- SpecLog Abstraction and Interface
- Belfast Design and Implementation
- Evaluation

Implementing the SpecLog Abstraction

Fix-ante ordering and SpecLog provide a general framework for speculation

Implementing the SpecLog Abstraction

Fix-ante ordering and SpecLog provide a general framework for speculation

Belfast is an implementation, solves practical challenges

Implementing the SpecLog Abstraction

Fix-ante ordering and SpecLog provide a general framework for speculation

Belfast is an implementation, solves practical challenges

- Right-sizing quotas – rate-based quotas

Implementing the SpecLog Abstraction

Fix-ante ordering and SpecLog provide a general framework for speculation

Belfast is an implementation, solves practical challenges

- Right-sizing quotas – rate-based quotas
- Quickly absorbing bursts – lag-fix mechanism

Implementing the SpecLog Abstraction

Fix-ante ordering and SpecLog provide a general framework for speculation

Belfast is an implementation, solves practical challenges

- Right-sizing quotas – rate-based quotas
- Quickly absorbing bursts – lag-fix mechanism
- Dealing with long-term rate changes at shards

Implementing the SpecLog Abstraction

Fix-ante ordering and SpecLog provide a general framework for speculation

Belfast is an implementation, solves practical challenges

- Right-sizing quotas – rate-based quotas
- Quickly absorbing bursts – lag-fix mechanism
- Dealing with long-term rate changes at shards
- Retaining seamless reconfigurability

} speculation lease window

Implementing the SpecLog Abstraction

Fix-ante ordering and SpecLog provide a general framework for speculation

Belfast is an implementation, solves practical challenges

- Right-sizing quotas – rate-based quotas
- Quickly absorbing bursts – lag-fix mechanism
- Dealing with long-term rate changes at shards
- Retaining seamless reconfigurability

} speculation lease window

- Mis-speculations and failure handling – view-change protocol

Evaluation

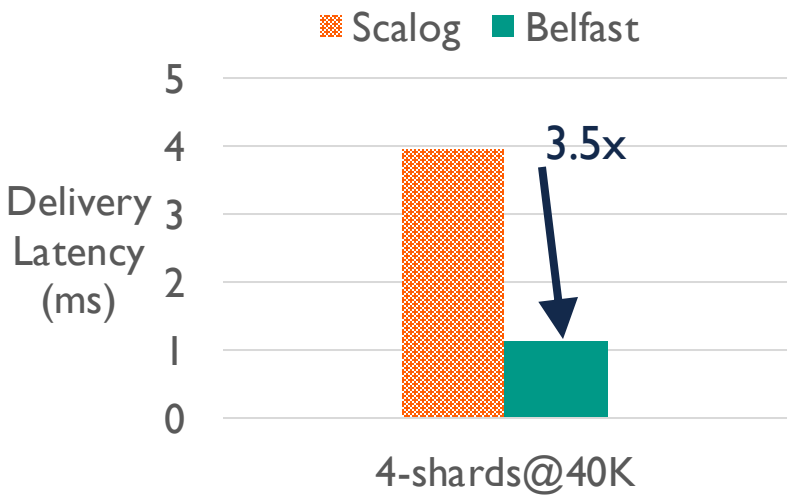
- What are the end-to-end latency benefits of Belfast?
- Does Belfast benefit end applications?

End-to-End Latency Benefits

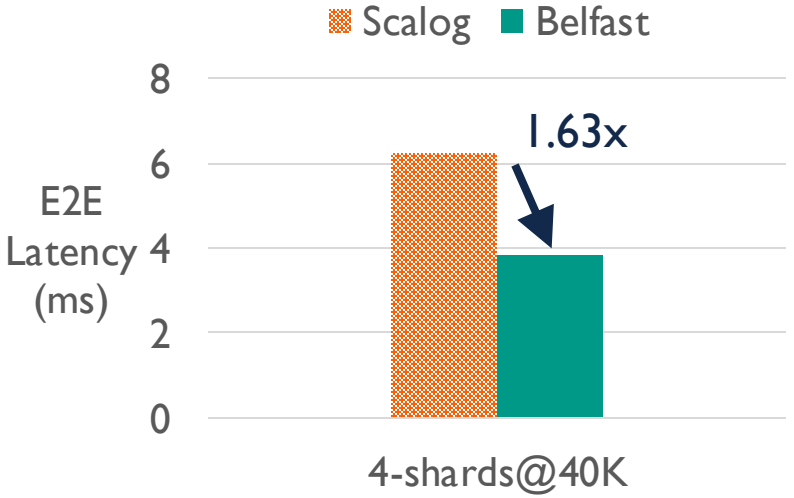
Workload: 4KB records, downstream computation of 1.5ms per batch of consumed records

End-to-End Latency Benefits

Workload: 4KB records, downstream computation of 1.5ms per batch of consumed records



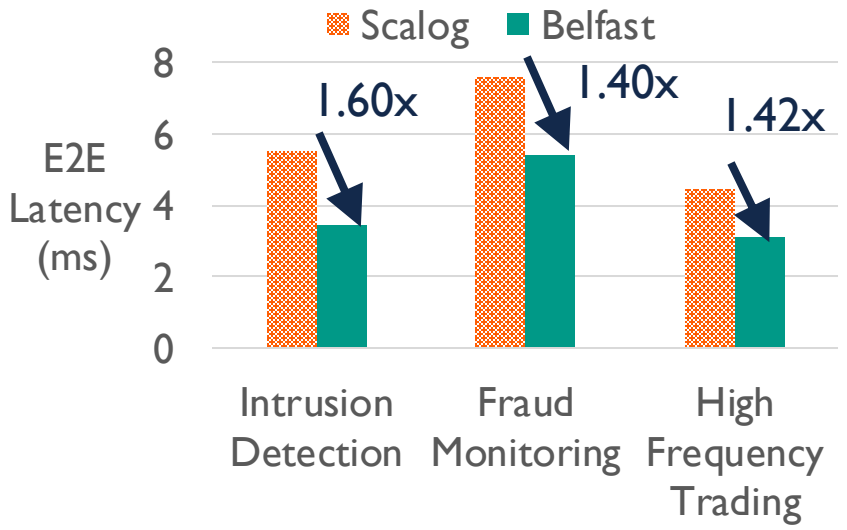
Delivers records 3.5x earlier



Enables 1.6x lower end-to-end latencies

Benefits in Applications

Build 3 applications – intrusion detection, fraud monitoring and high-frequency trading



Benefits in e2e latency for real-world applications

More details and experiments in the paper

Many more experiments in the paper

- App evaluation under failure scenarios and mis-speculations
- Evaluation under bursts, rate changes
- End-to-end latency at scale

The paper covers many more discussions about Belfast

Summary

- Today's shared logs suffer high delivery latencies due to expensive ordering
- SpecLog, a new abstraction, allows speculative delivery by predicting global order
- SpecLog uses Fix-Ante ordering, enables high speculation success
- Belfast, an implementation of SpecLog, enables low e2e latencies for end applications

Available on GitHub:
https://github.com/dassl-uiuc/speclog-artifact