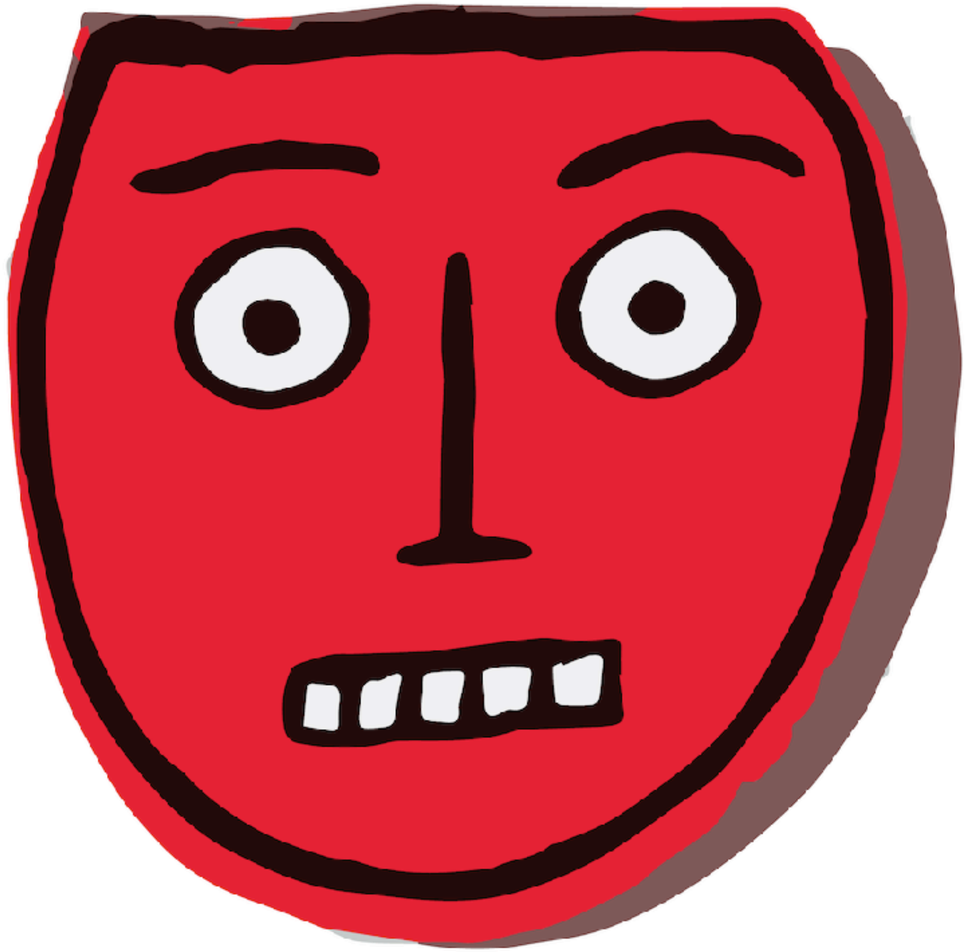
OK Log 

Distributed and coördination-free logging



@peterbourgon
github.com/**peterbourgon**
peter.bourgon.org


fastly

Contextualizing

Outline

Gremlins of distsys

Logging systems

OK Log design

Outline

☛ Gremlins of distsys

Logging systems

OK Log design

My incredible journey

- Software engineer — not computer scientist
- (Though I love a good paper)
- Distributed systems journeyman
- This shit is hard!!!

Why is it hard

- Failure modes are intractable
- Symptoms are subtle
- Diagnosis and triage requires specialized knowledge
- Computer science fundamentals are insufficient

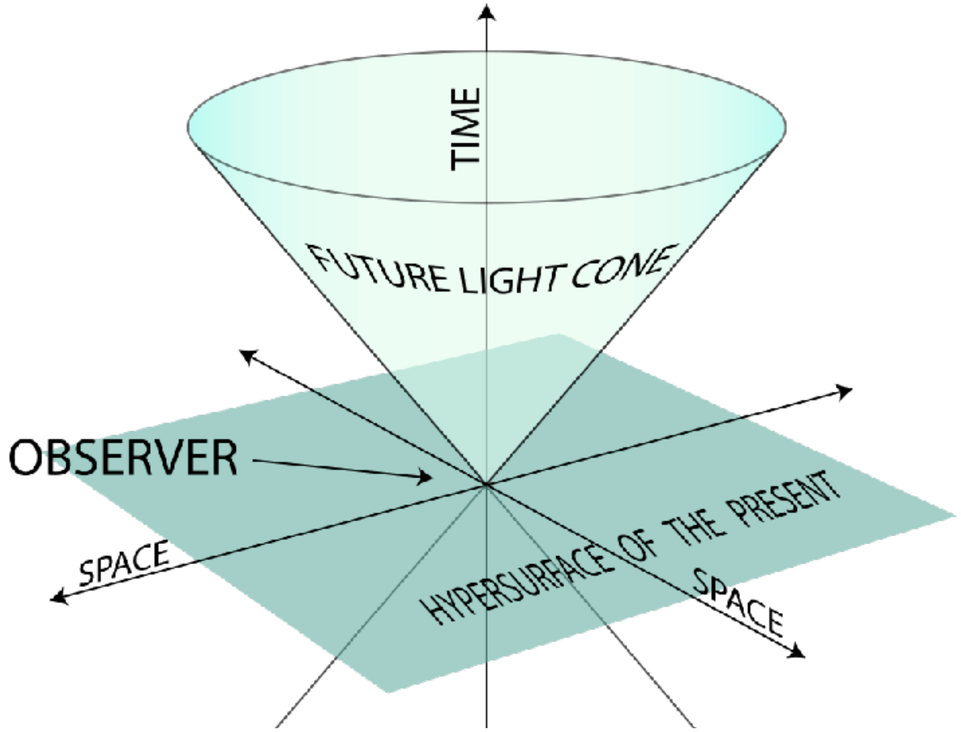
You know this

```
>>> x = 1

>>> print x
1
```

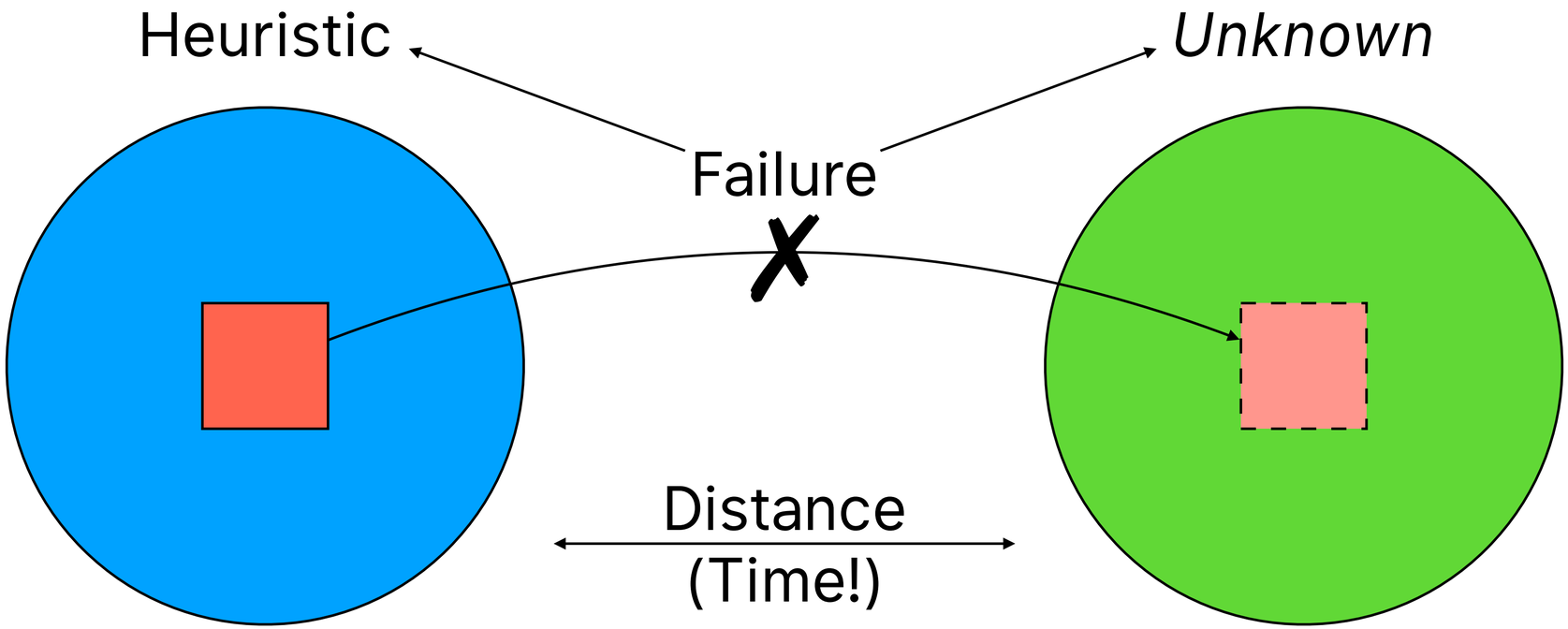
You know this

```
$ curl -XPOST http://db:9091/vars/x/1
HTTP 502 (Bad Gateway)

$ curl -XGET http://db:9091/vars/x
HTTP 500 (Internal Server Error)
```

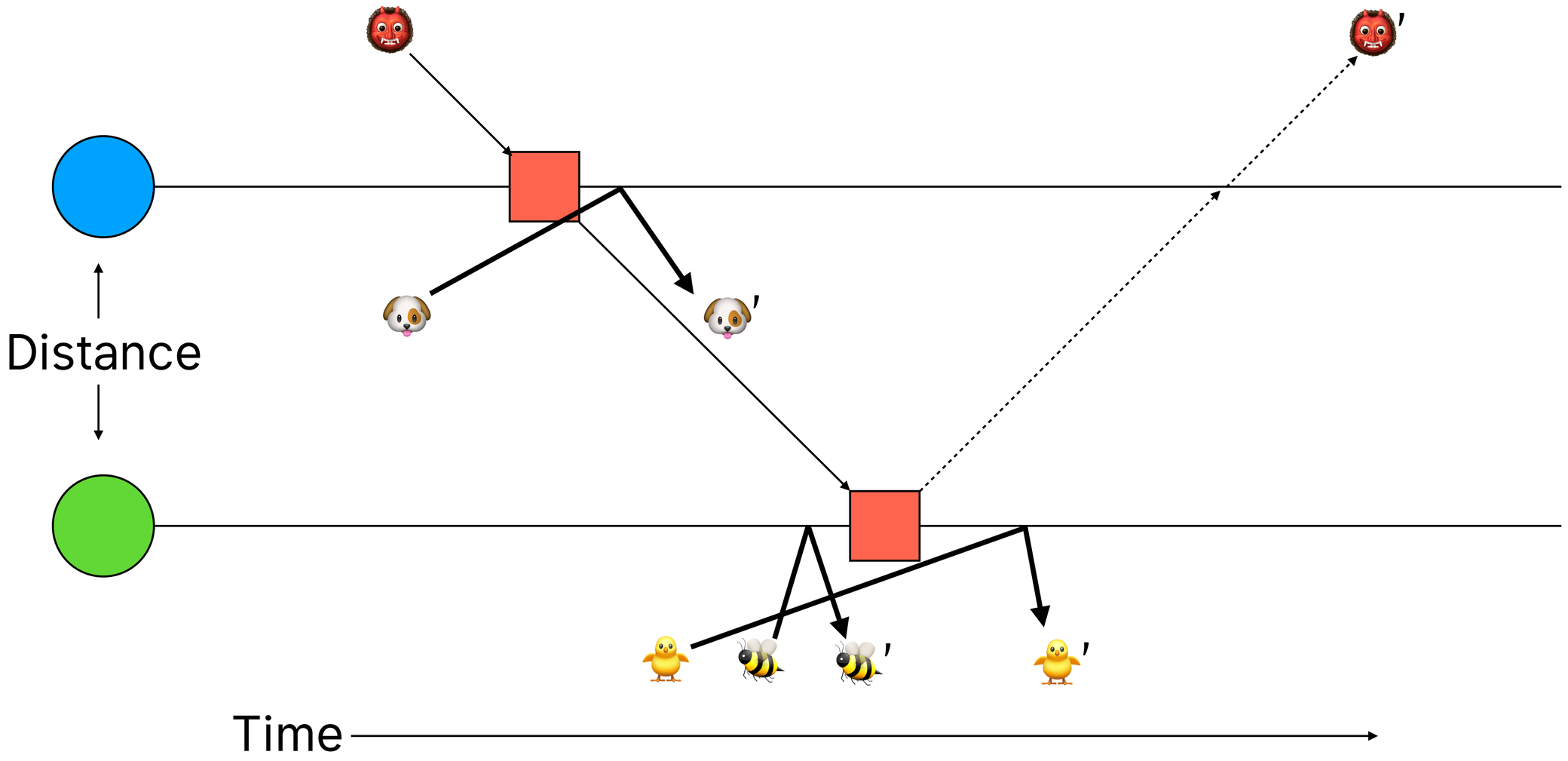
Fundamentally hard



Fundamentally hard



Fundamentally hard



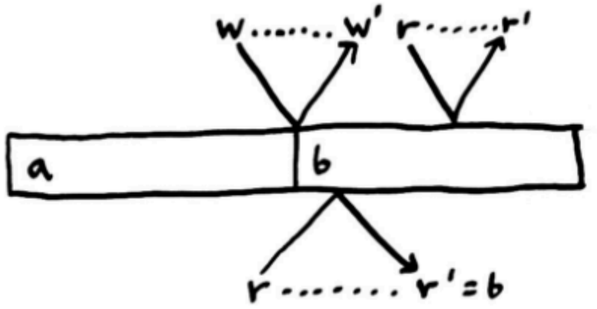
Strong consistency models

Hamburger

Secure | https://aphyr.com/posts/313-strong-consistency-models

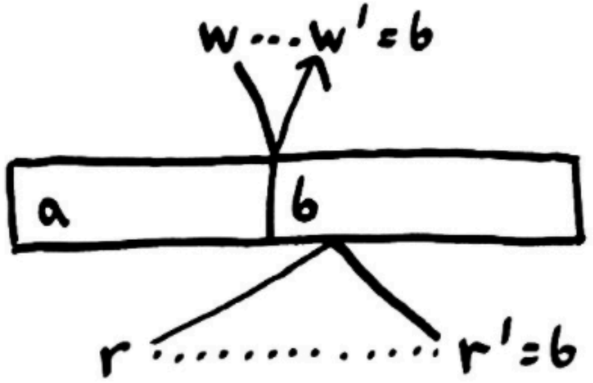
Light cones

Howerver, this is not the full story: in almost every real-world system, processes are *distant* from each other. An uncached value in memory, for instance, is likely on a DIMM *thirty centimeters away* from the CPU. It takes light over a full nanosecond to travel that distance–and real memory accesses are much slower. A value on a computer in a different datacenter could be thousands of kilometers–hundreds of milliseconds–away. We just can't send information there any faster; physics, thus far, forbids it.



This means our operations are *no longer instantaneous*. Some of them might be so fast as to be negligible, but in full generality, operations *take time*. We *invoke* a write of a variable; the write travels to memory, or another computer, or the moon; the memory changes state; a confirmation travels back; and then we *know* the operation took place.

The delay in sending messages from one place to another implies *ambiguity* in the history of operations. If messages travel faster or slower, they could take place in unexpected orders. Here, the bottom process invokes a read when the value is `a`. While the read is in flight, the top process writes `b`–and by happenstance, its write arrives *before* the read. The bottom process finally completes its read and finds `b`, not `a`.



This history violates our concurrent register consistency model. The bottom process did *not* read the current value at the time it invoked the read. We might try to use the completion time, rather than the invocation time, as the "true time" of the operation, but this fails by symmetry as well; if the read arrives *before* the write, the process would receive `a` when the current value is `b`.

@peterbourgon

It's phenomenological

- Immanuel Kant → Edmund Husserl → ⋯
- Actual thing (*Noumena*) vs. our experience of it (*Phenomena*)
- There is no "true" state — just a union of expressed intents
- Set aside the fiction of a global truth!
- We can only reason about each node's unique perception
- Successful distsys thinking starts with *phenomenological reduction*

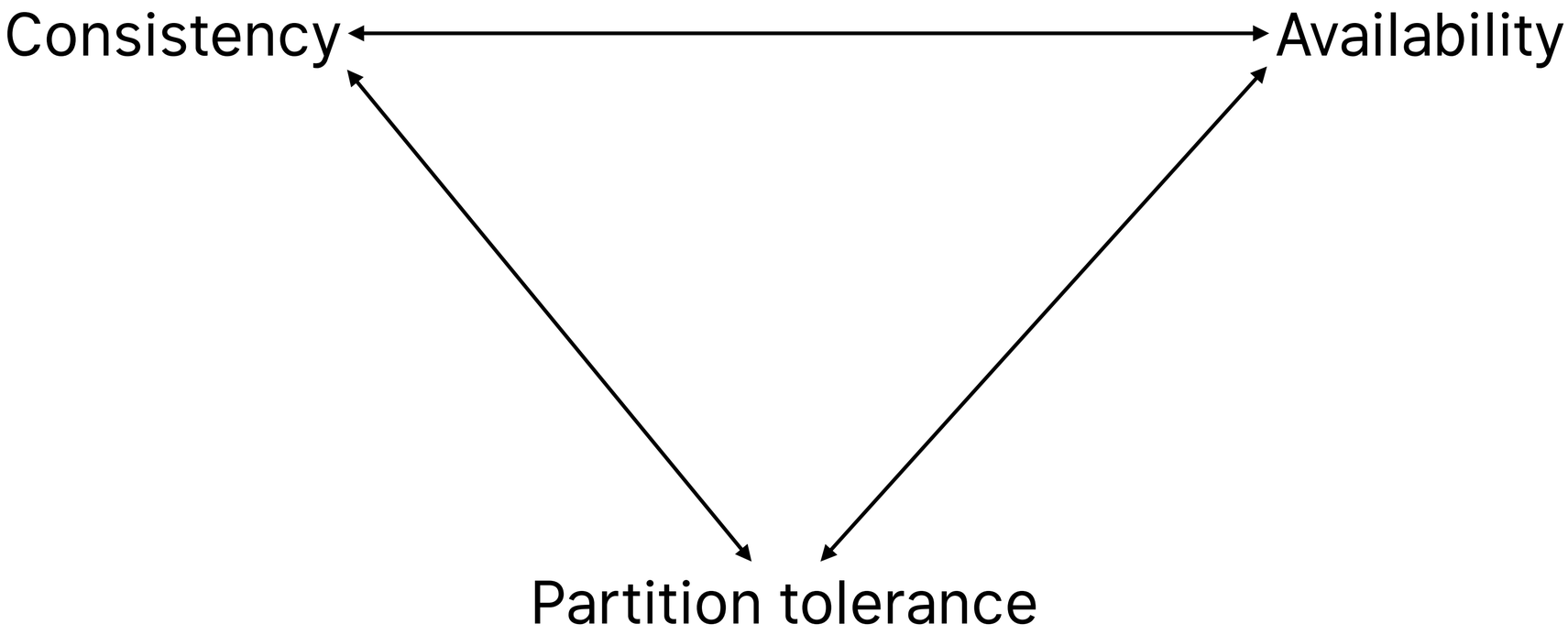
Idioms

1980s: RPC

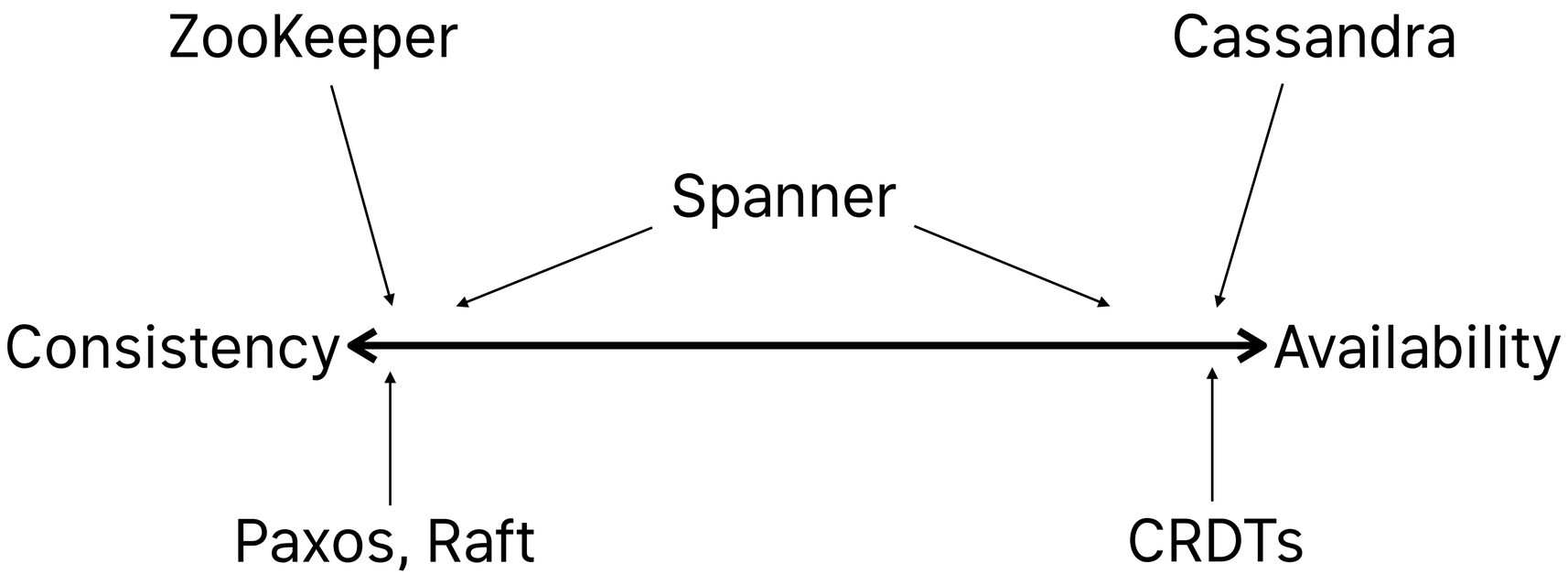
1990s: CORBA

2000s: CAP

CAP



CAP


ZooKeeper
Cassandra
Spanner
Consistency
Availability
Paxos, Raft
CRDTs

PACELC

- In the case of network **P**artition
- You have to choose between **A**vailability and **C**onsistency
- **E**lse
- You can choose between **L**atency and **C**onsistency
- → PA/EL, PA/EC, PC/EL, PC/EC

This is all very interesting

- Main point: situation is complex
- And becoming moreso

Danger: Hot Takes

Outline

☛ Gremlins of distsys

Logging systems

OK Log design

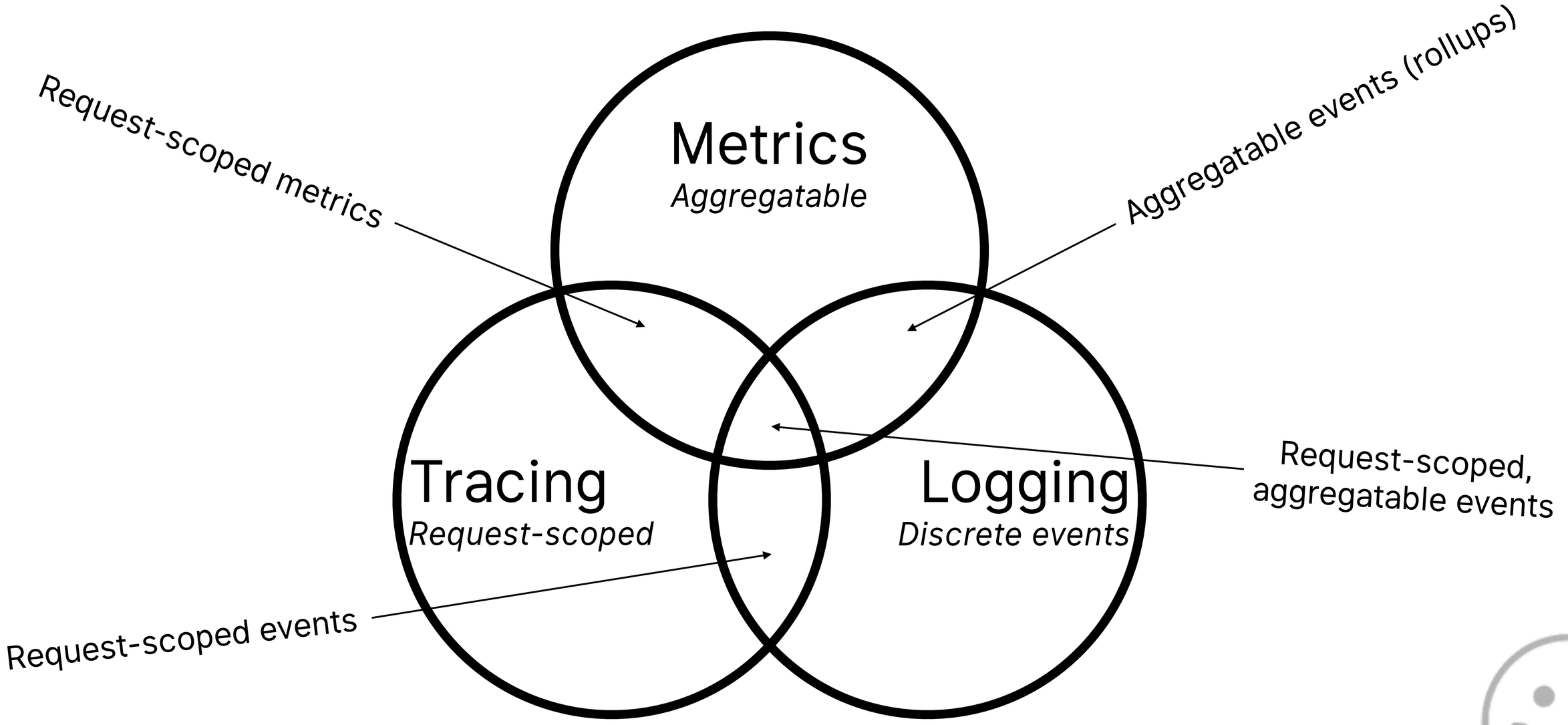
Outline

Gremlins of distsys

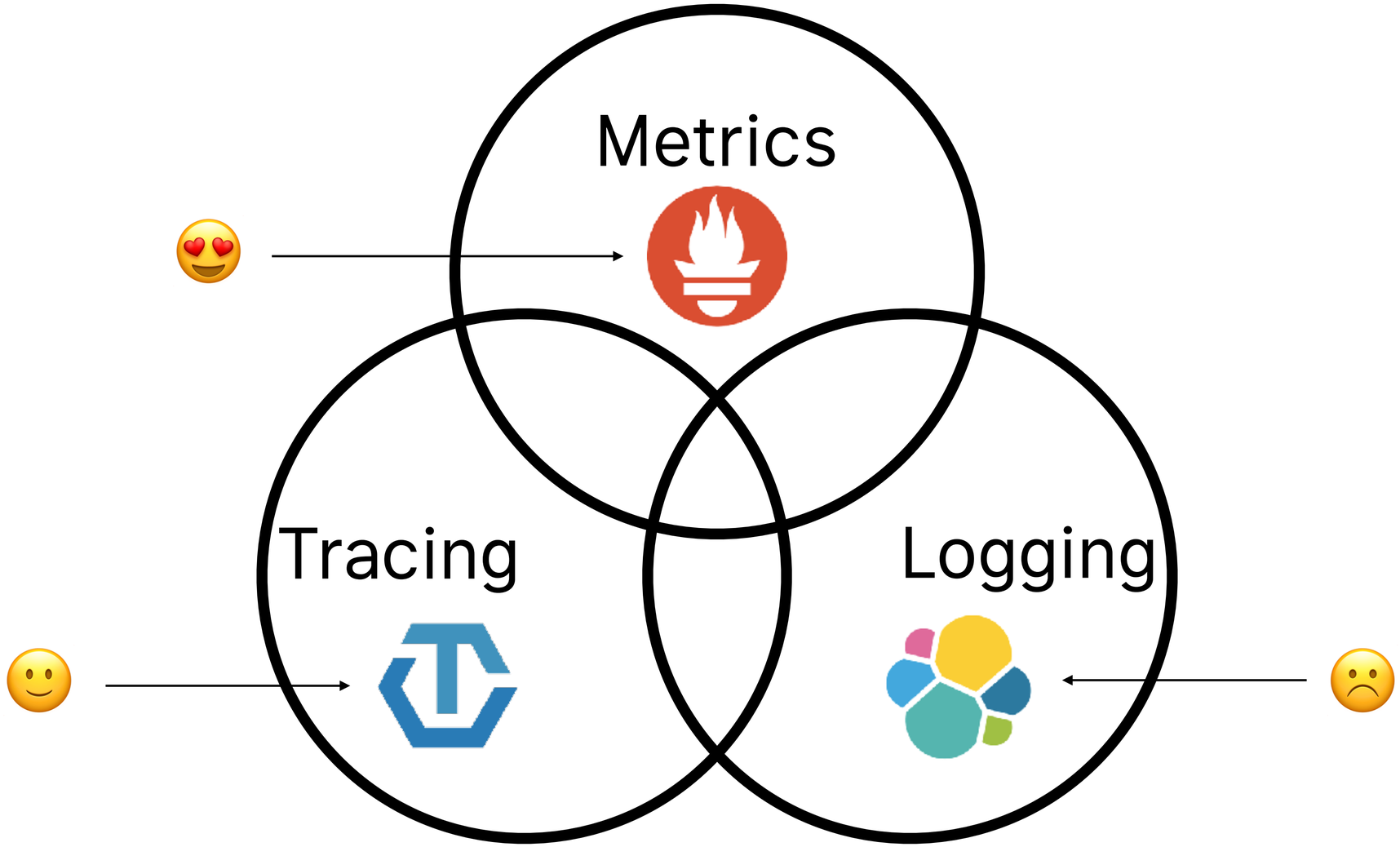
☛ Logging systems

OK Log design

Observability



Observability



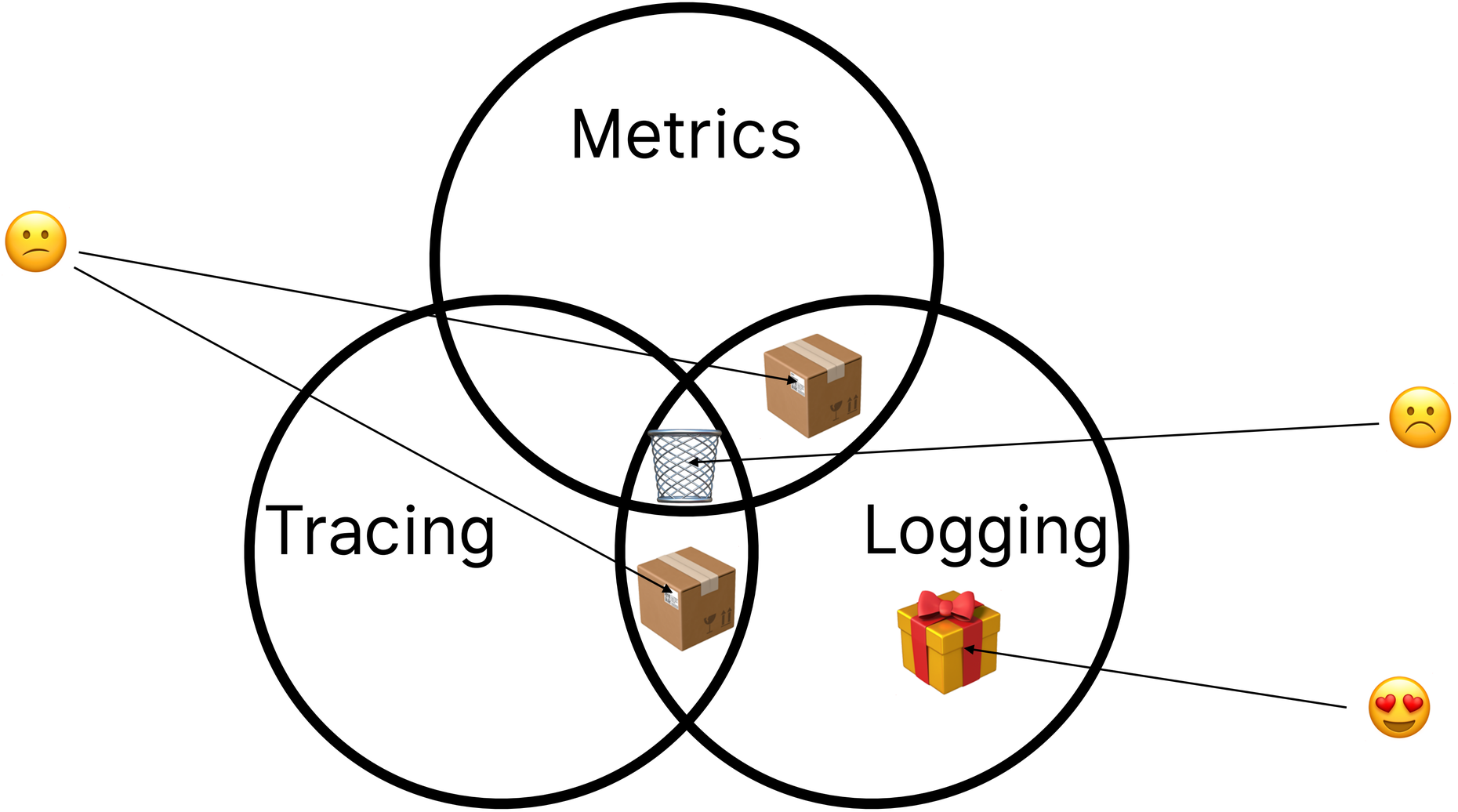
☹️?

- Lucene not the best for log data
- Operationally complex, can be unstable
- Resource heavy
- Does too much — most of the time I just wanna, like, grep my logs

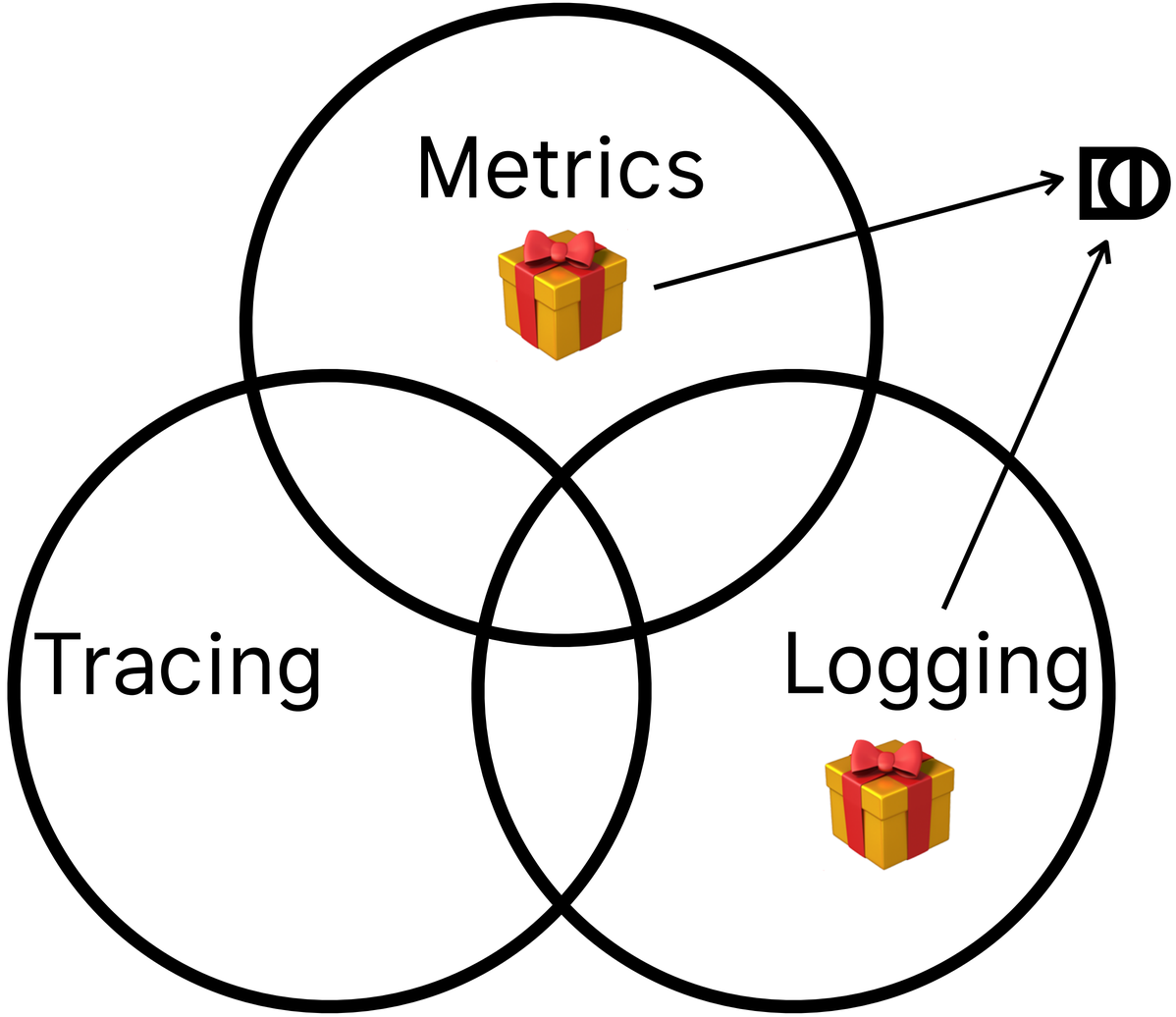
What are we doing here

- Observability = mechanisms for understanding
- Given a *what*, logging helps us answer the *why* and *how*
  - *What* = high-level, via metrics and alerts
  - *Why, how* = low-level, via error messages and contextual info

Well-defined scope



Well-defined scope



Actual requirements

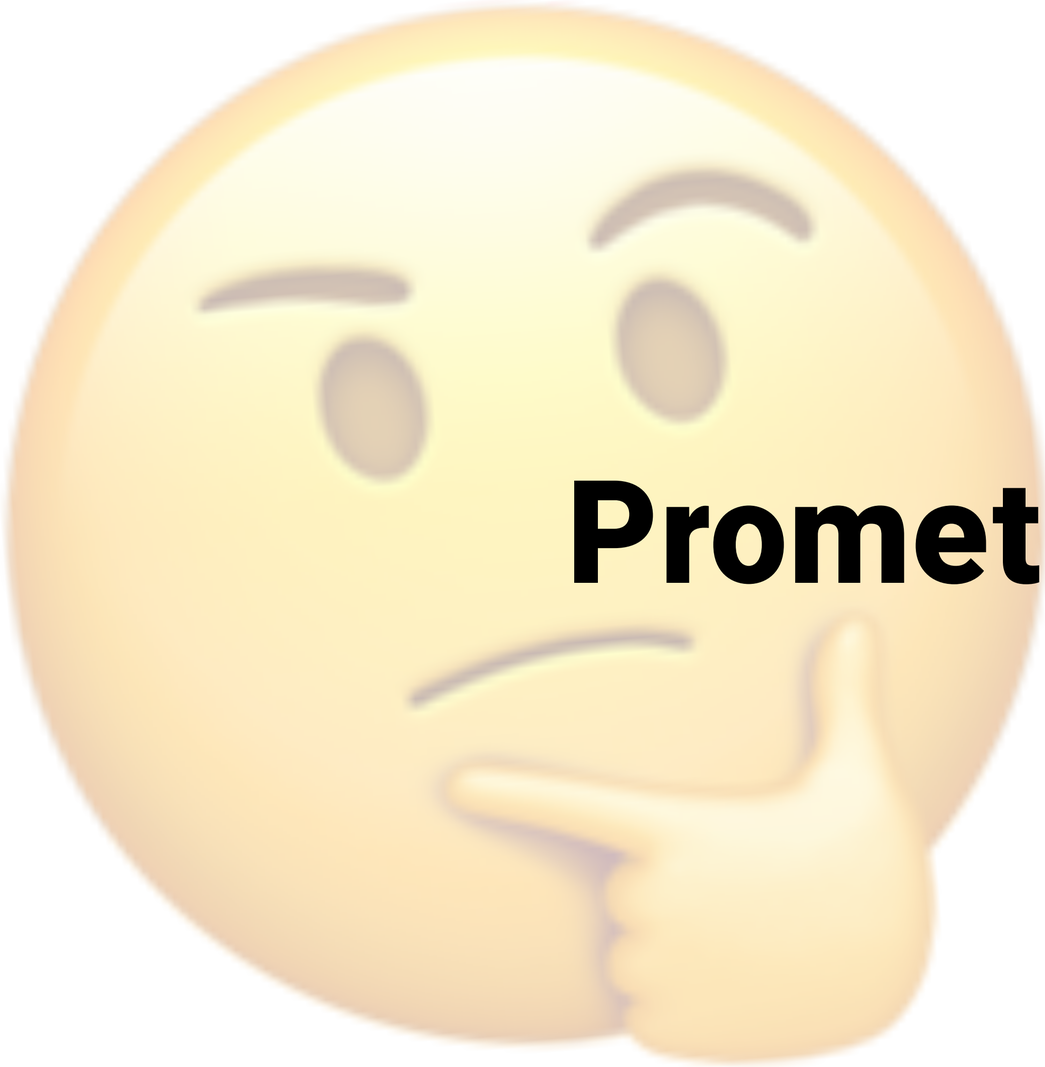
- Often: grep for some error in the recent past
- Often: grep for some customer/transaction ID
- Sometimes: grep for a class of things in the last days/weeks
- Rarely: roll-ups, statistical analysis — *this is the domain of metrics!*

Products

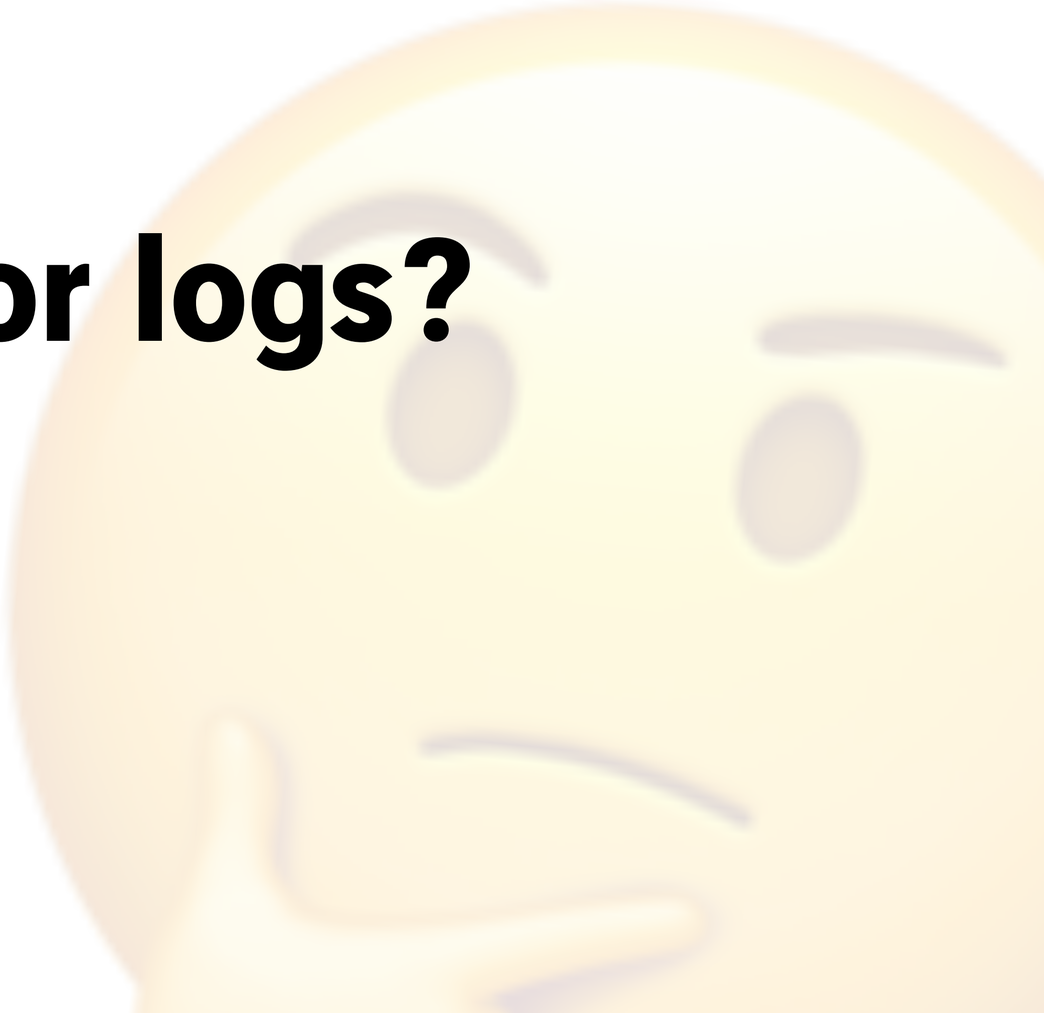
- Kafka? Whoa boy
- HBase? Hadoop lol
- Heka? Abandonware
- Ekanite? syslog seems wrong
- Fluentd/Logstash? Only part of the problem
- Splunk/Loggly/Honeycomb/...? Not on-prem, $/€/£

Why is Prometheus good?

- Run it yourself — open-source and deployable on-prem
- "Cloud-Native" — dynamic, containerized, microservicey workloads
- Easy to operate — local storage, no clustering, pull model
- Complete system — doesn't need separate web UI, TSDB, etc.
- Scales up & out — 90% of use cases satisfied without special effort



Prometheus for logs?



Outline

Gremlins of distsys

☛ Logging systems

OK Log design

Outline

Gremlins of distsys

Logging systems

☛ OK Log design

☛ **OK Log design**

High-level goals

Components

Behaviors

☛ **OK Log design**

→ *High-level goals*

Components

Behaviors

High-level goals

- Prometheus for logs
  - Run it yourself — open-source and deployable on-prem ✔
  - "Cloud-Native" — dynamic, microservicey workloads ✔
  - Easy to operate — local storage, no clustering, pull model *?*
  - Complete system — doesn't need separate web UI, TSDB, etc. ✔
  - Scales up & out — 80% of use cases satisfied w/o special effort ✔

High-level goals

- Principal use case: application logs from typical microservices
  - High-volume, low-QOS
- Secondary use case: event or audit logs
  - Low-volume, high-QOS
- Universal log consumer, including from third-party software
  - Can't mandate structure; should treat each record as opaque

Constraining the problem

- Opaque records imply a pure *transport* system
- Leave application concerns to the edges
  - e.g. Contextual annotation can happen prior to ingest
  - e.g. Queryable indices, aggregates modeled as stream consumers
- Only available metadata = timestamp at ingest

Query interface

- Is time-bounded grep sufficient as a query interface?
- Yes
- Maybe
- Let's say yes

A note on scale

- Prometheus has it easy — metrics are aggregatable
- A single server can conceivably handle 10k+ target services
- Core Prometheus opts-out of clustering/distsys — *hell yeah*
- To operate at desired scale, we (unfortunately) need more capacity
- Logging systems are distributed systems, by necessity

"Easy to operate"

- Must not require specialized knowledge or arcana
- << 1x FTE to deploy and maintain
- Embeddable in product team infrastructure — sysadmin not required
- No byzantine or pathological failure modes!!!
  - Kill processes, [delete files,] start again = must work

"Easy to operate"

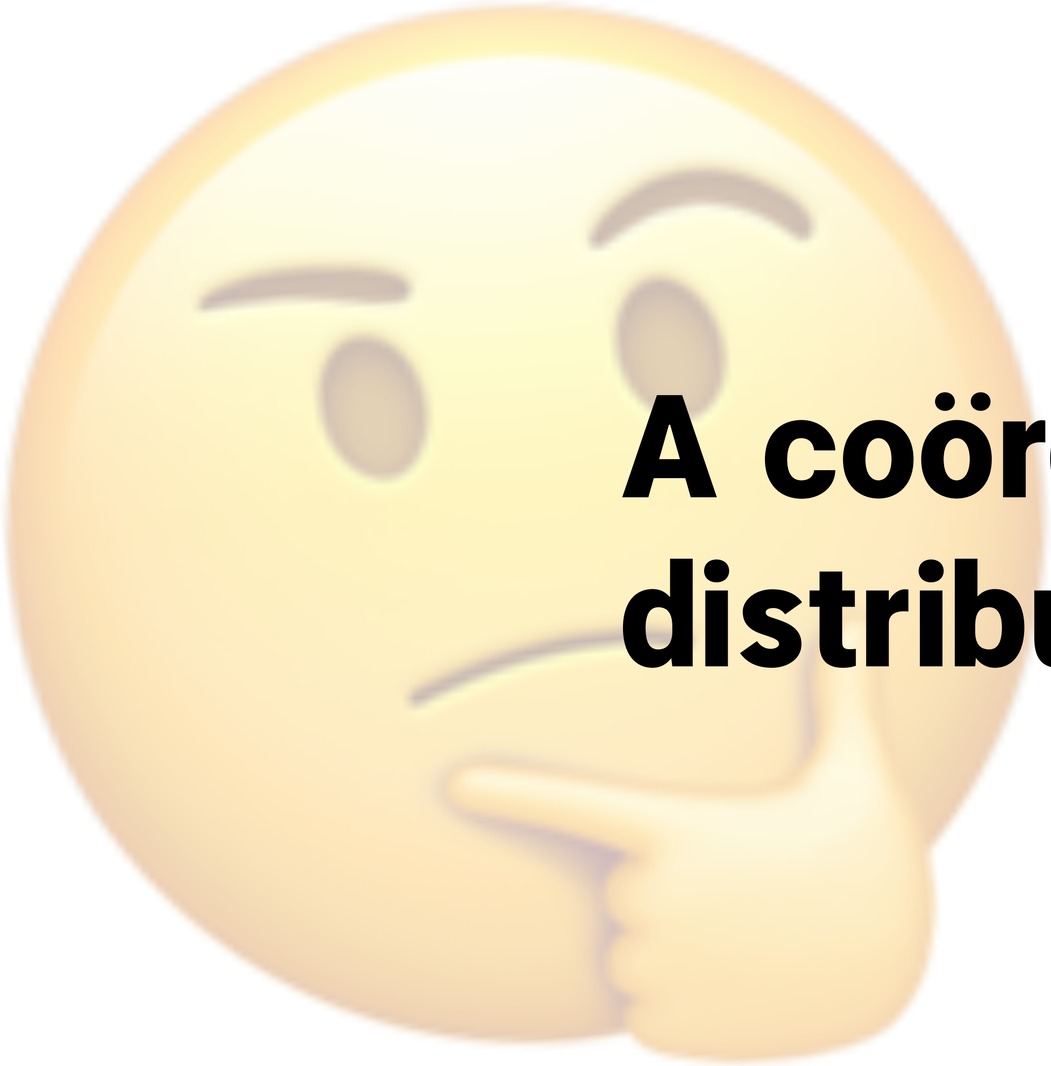
- As few knobs as possible — sensible defaults
- Automatic peer discovery and cluster management
- Adding nodes = adding capacity, ideally no rebalancing/reindexing/…
- Losing nodes = decrease capacity, ideally no other side effects
- ☞ Minimize transactions and other coördination

Coördination-free

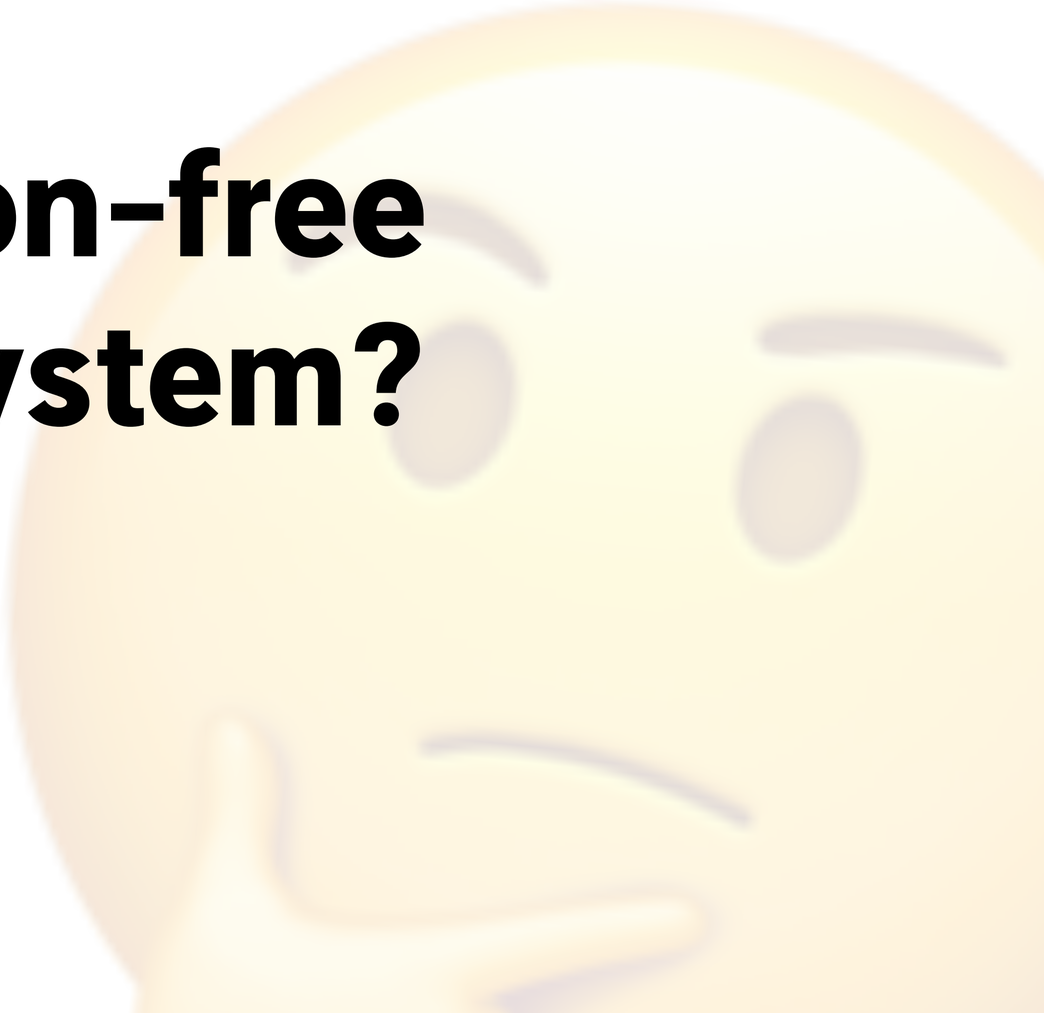
- Coördination: lifting from local truth (*Phenomena*) to global (*Noumena*)
- Expensive, error-prone, often literally impossible (CAP)
- "The best way to solve a problem is to opt-out of it" —*A smart*
- Set aside the fiction of a global truth!
- Coördination-free = phenomenological reduction

In practice

- No master election
- No knowledge of allocations
- No shard table
- No distributed index
- No vnodes, no ring



A coördination-free distributed system?



☛ **OK Log design**

→ *High-level goals*

Components

Behaviors

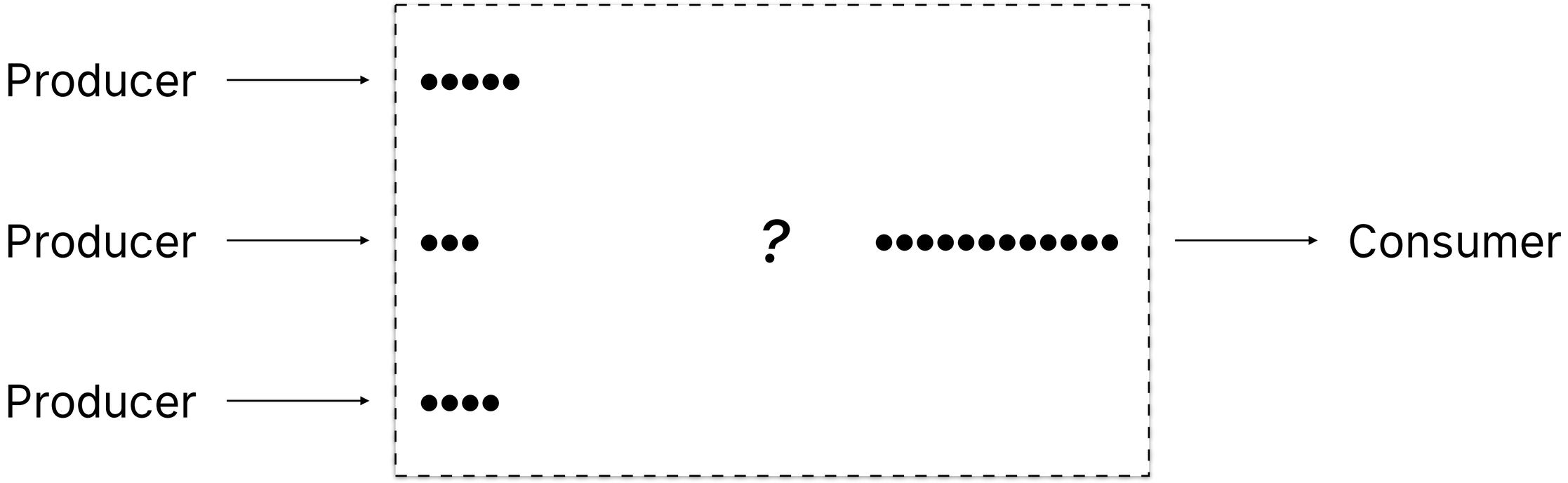
☛ **OK Log design**

High-level goals

→ *Components*

Behaviors

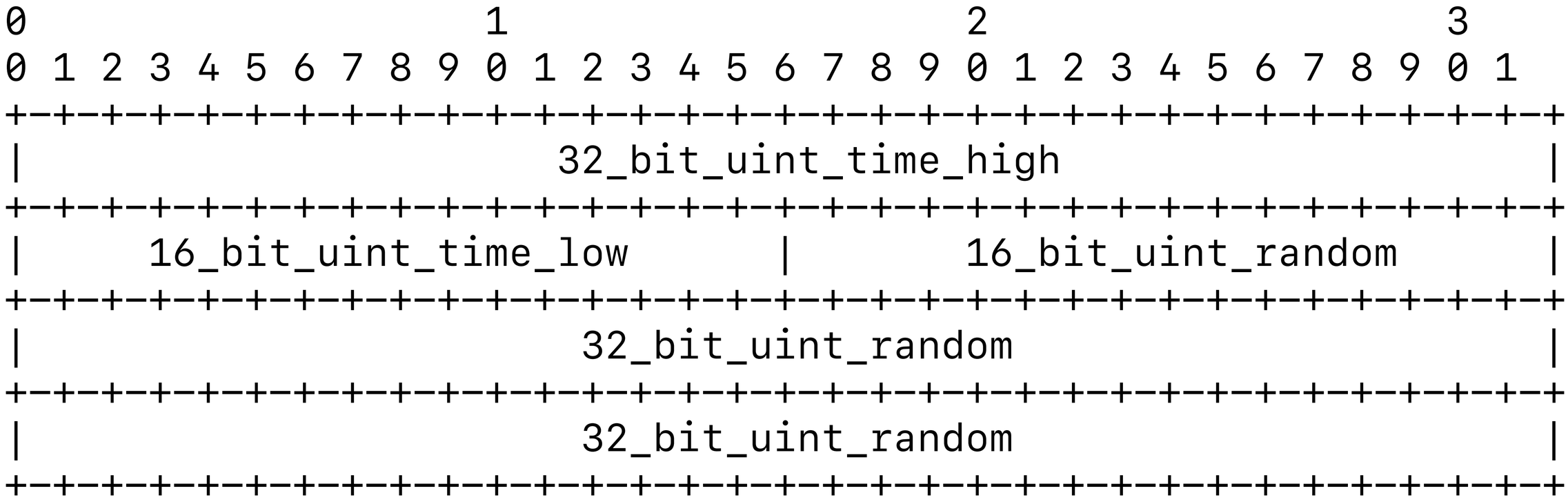
The importance of writes



Ingestion

- Different performance requirements for ingestion vs. querying
  - Ingest must take precedence
- Makes sense to isolate those workloads to different machines
  - They could be colocated for small or trial installations
- Ideally ingestion is limited by hardware

ULID


0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
32_bit_uint_time_high
16_bit_uint_time_low
16_bit_uint_random
32_bit_uint_random
32_bit_uint_random

ULID

Timestamp
10 chars, 48 bits
ms precision

`01AN4Z07BY79KA1307SR9X4MV3`

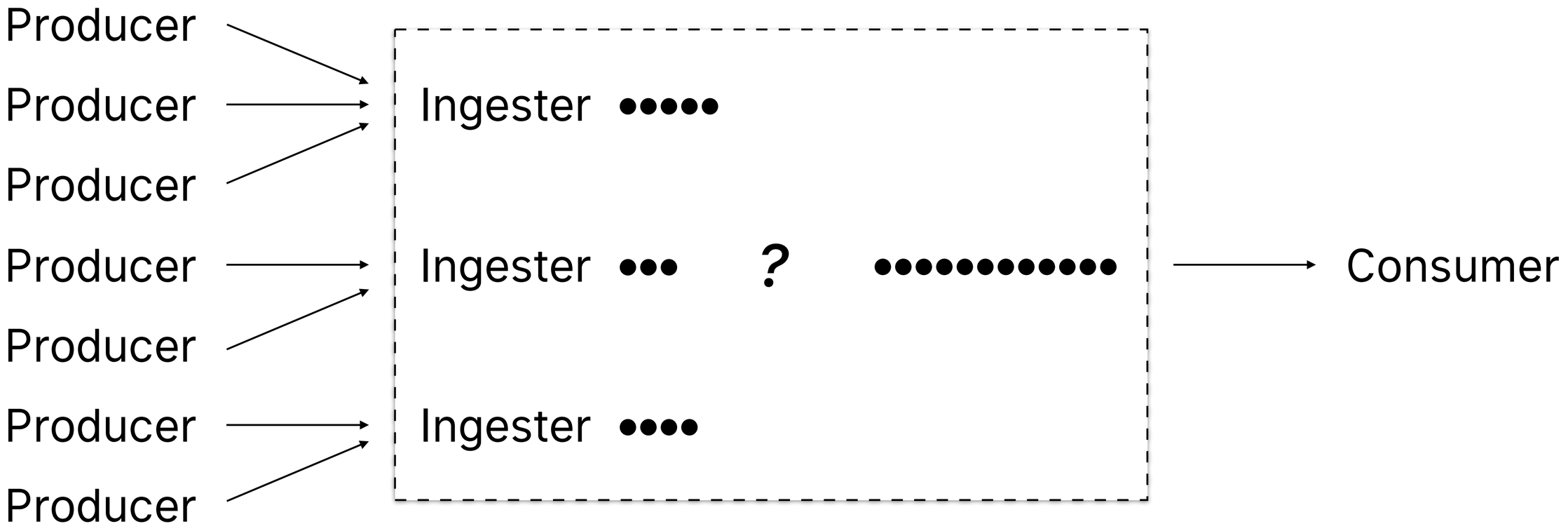
Entropy
16 chars, 80 bits
1.208 septillion per ms

Good enough

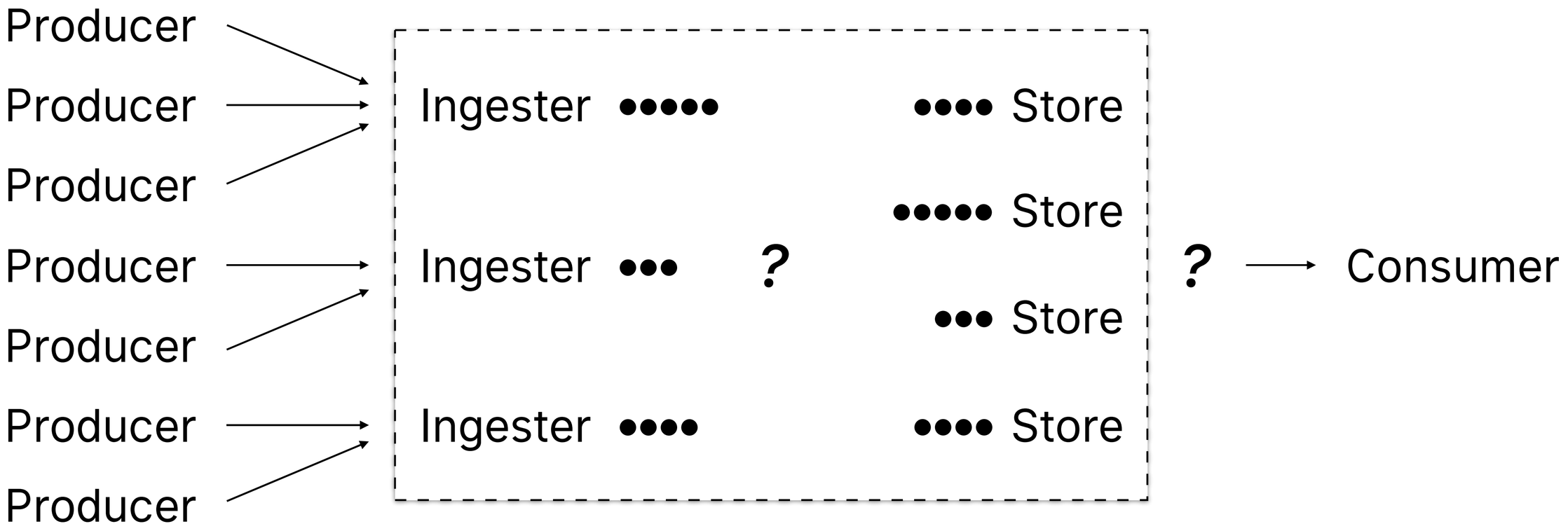
Durability modes

- *Fast* mode — typical for application logs
  - Write to file descriptor without explicit disk sync
- *Durable* mode — typical for event logs
  - Regular syncs to disk
- *Bulk* mode — typical for compliance/audit logs
  - Ingest only acks when entire segment has been replicated

Component model



Component model



☛ **OK Log design**

High-level goals

→ *Components*

Behaviors

☛ **OK Log design**

High-level goals

Components

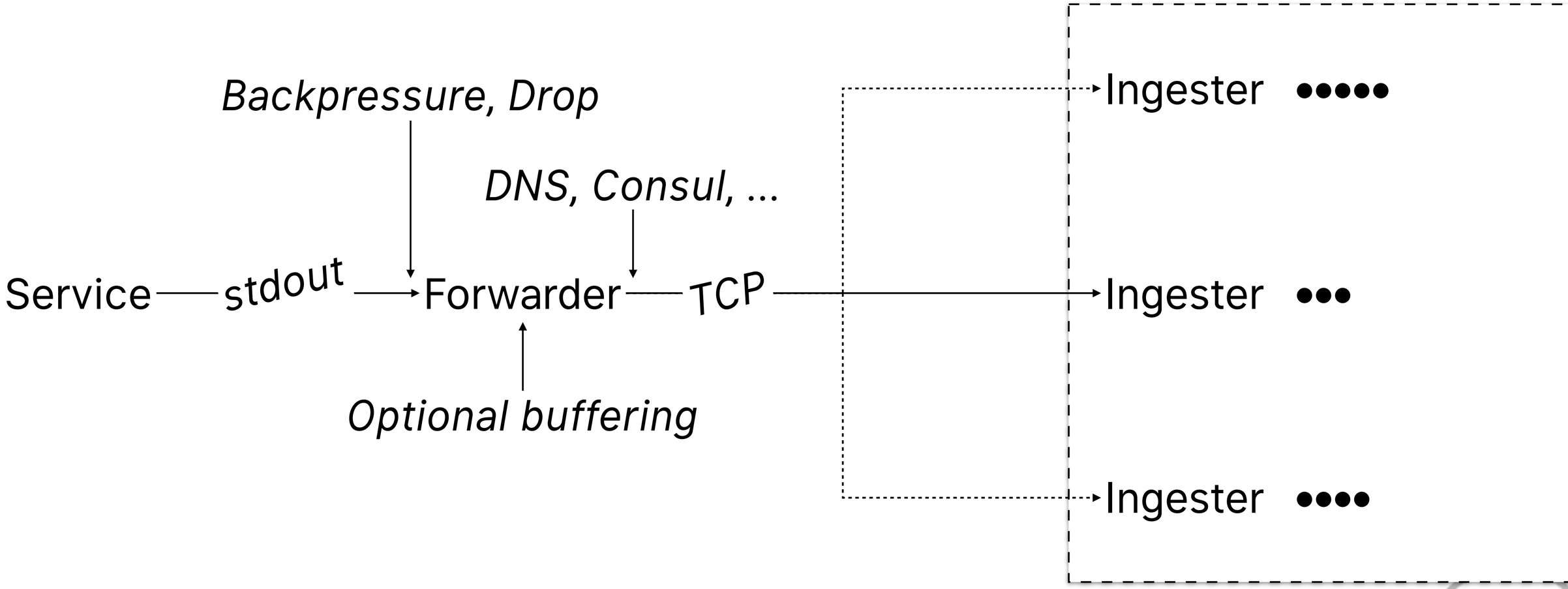
→ *Behaviors*

Ingest

Fast writes

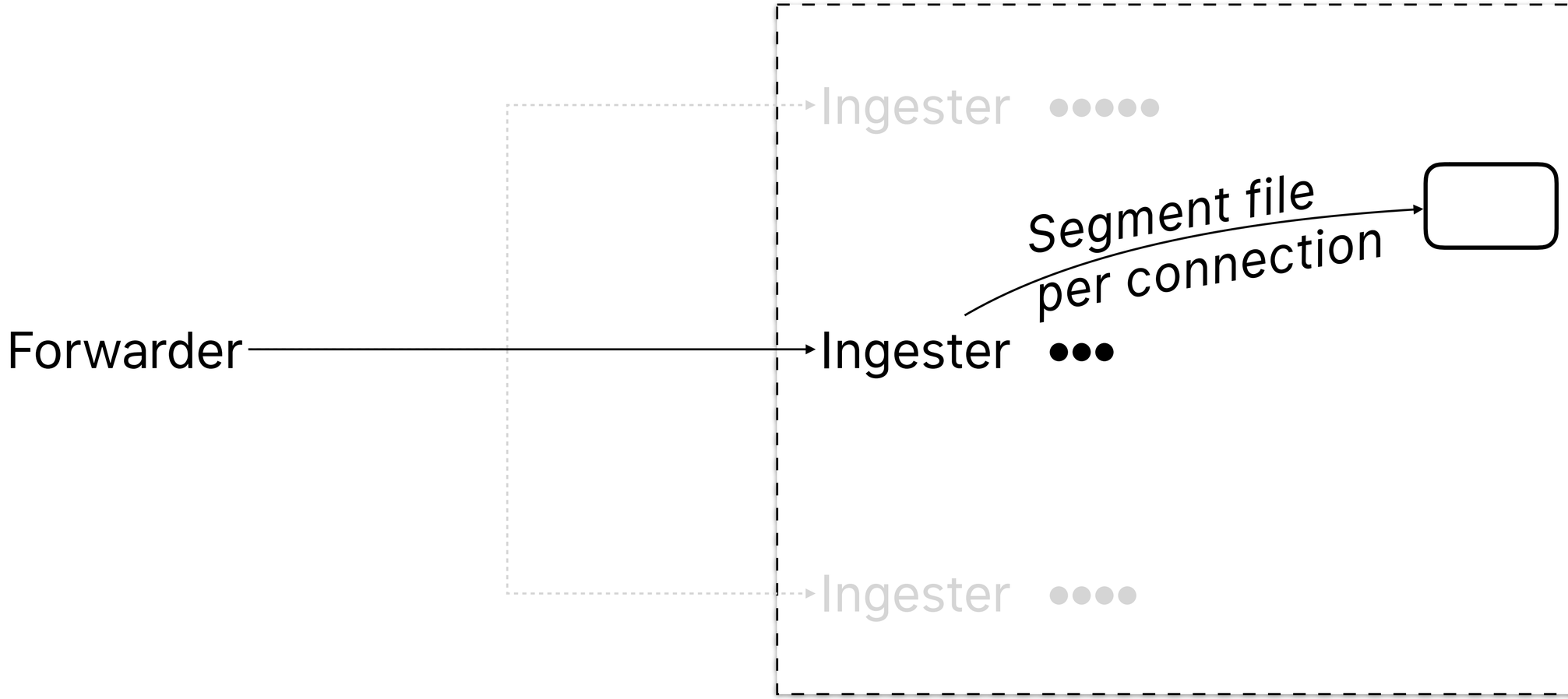
Ingest

Ingest

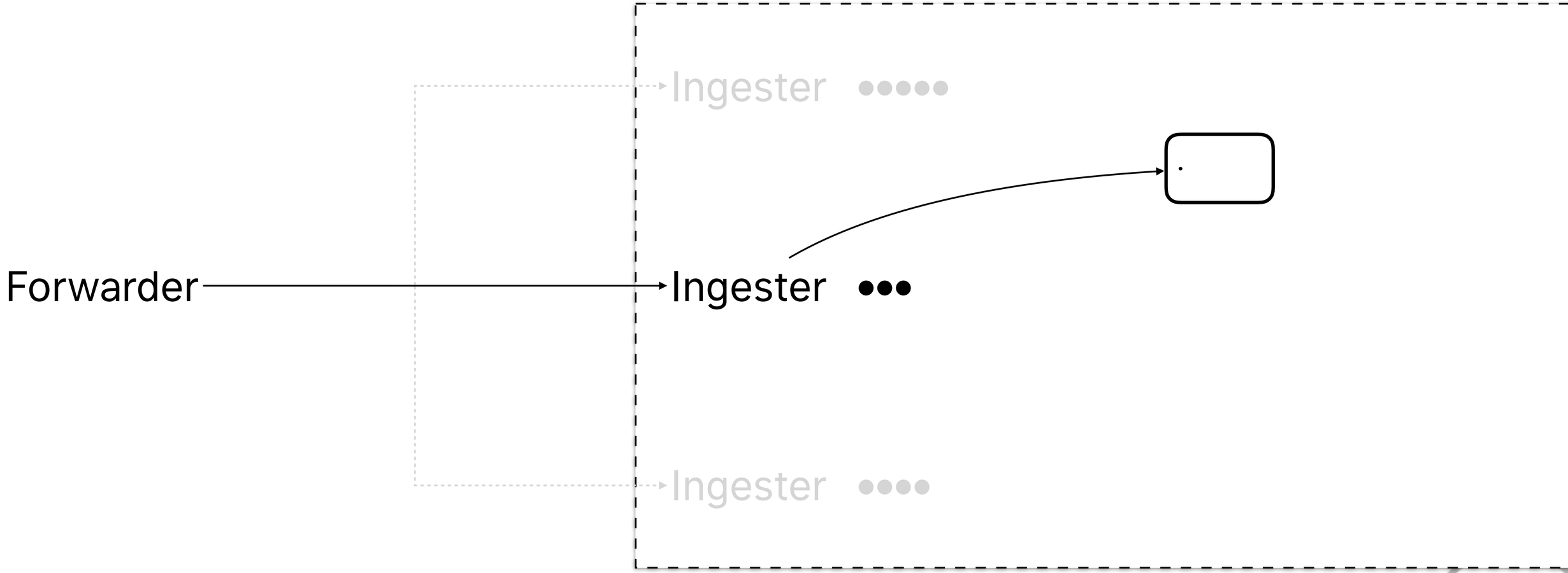

Backpressure, Drop
DNS, Consul, ...
Service
stdout
Forwarder
TCP
Optional buffering
Ingester
Ingester
Ingester

Ingest

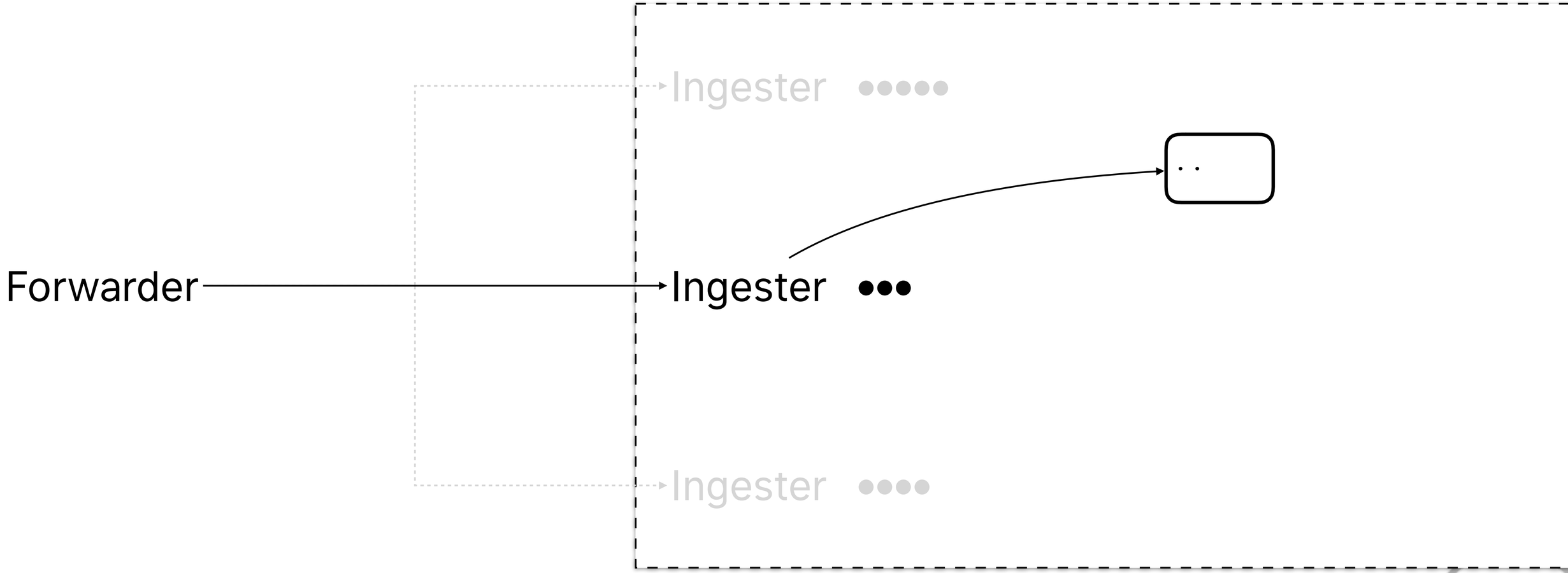
Ingest


Ingester
Segment file
per connection
Forwarder
Ingester
Ingester

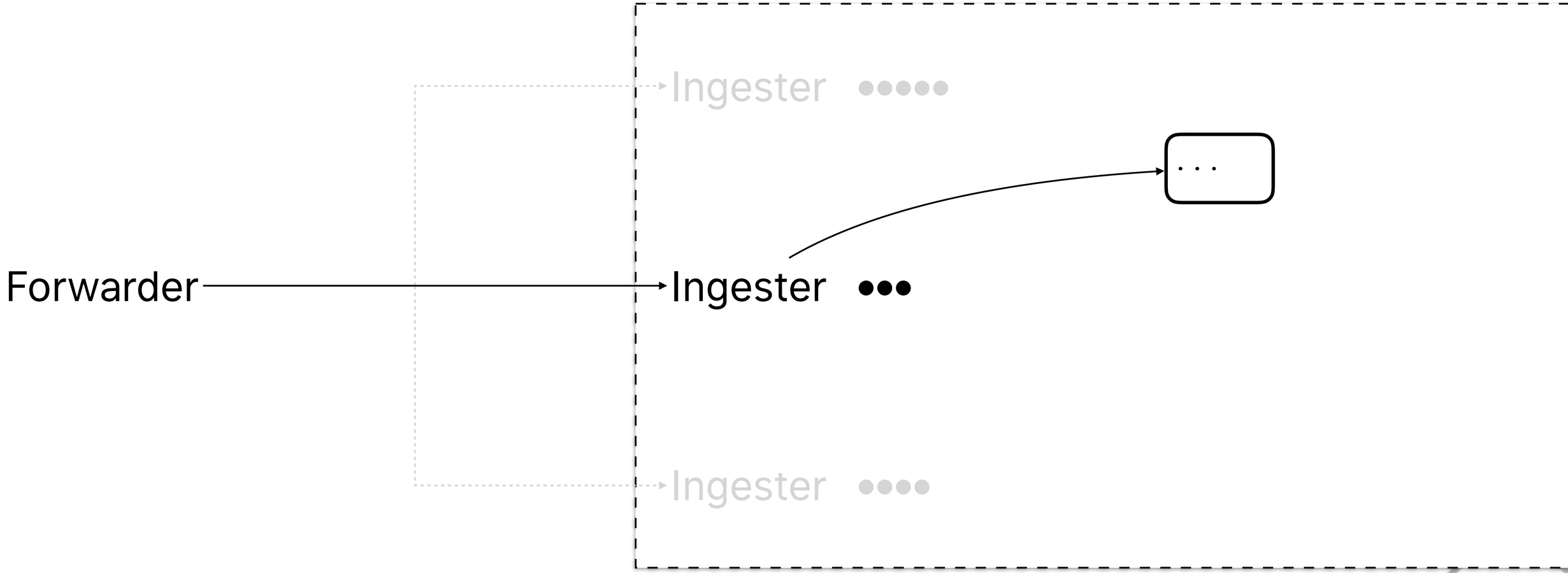
Ingest



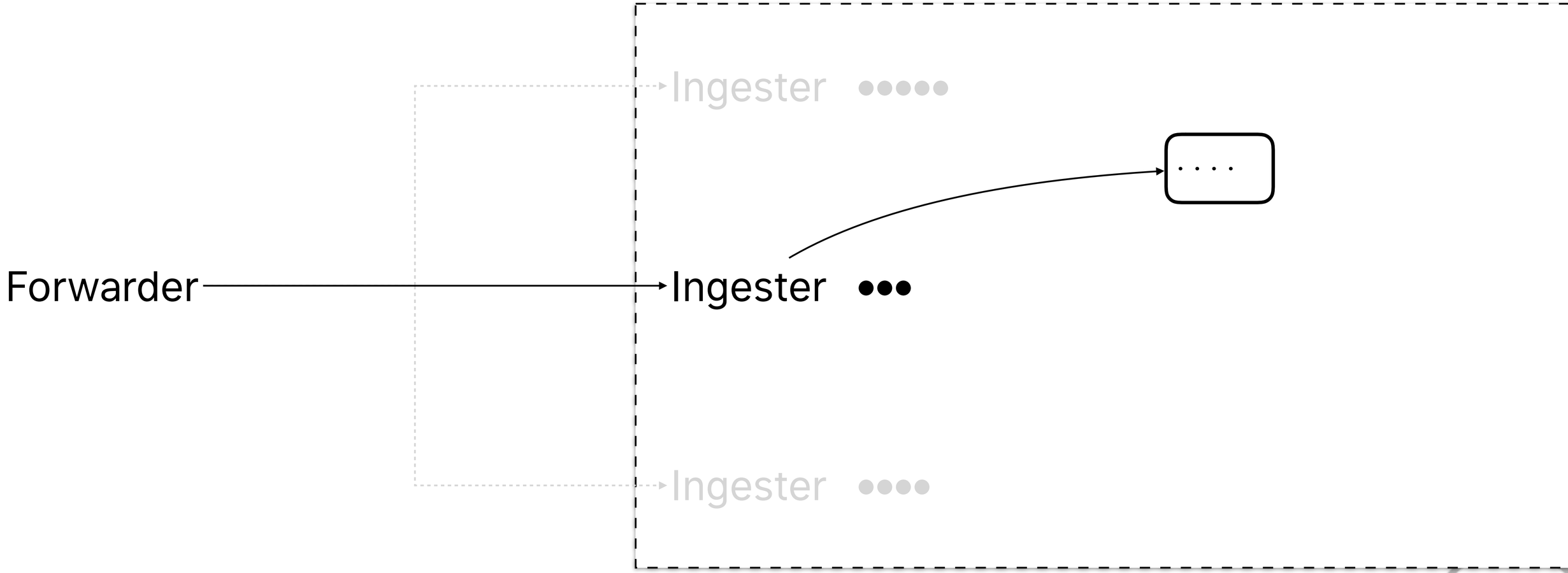
Ingest



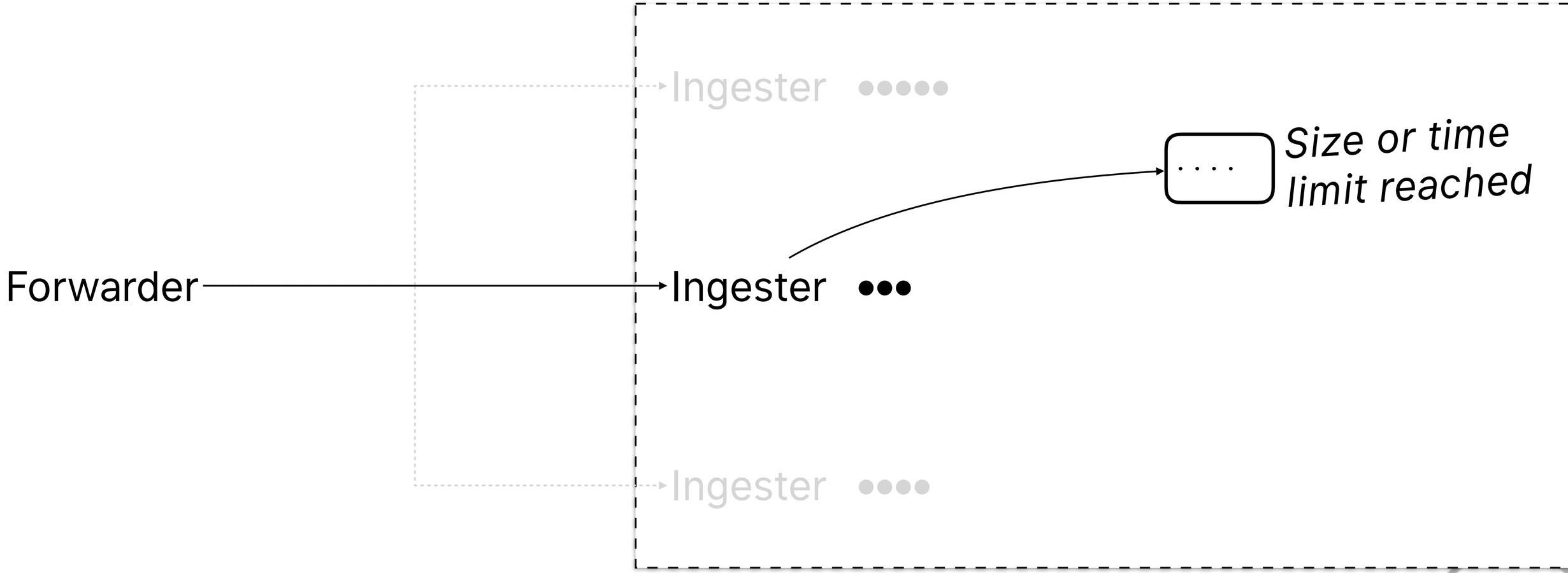
Ingest



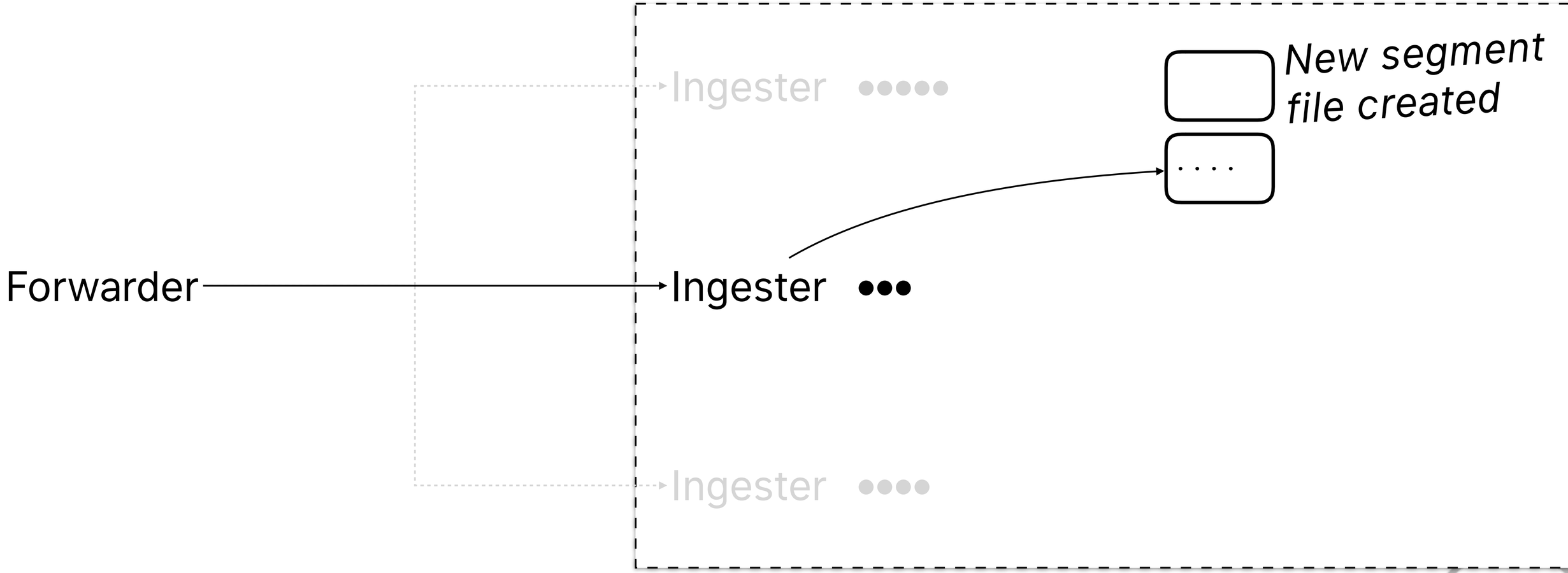
Ingest



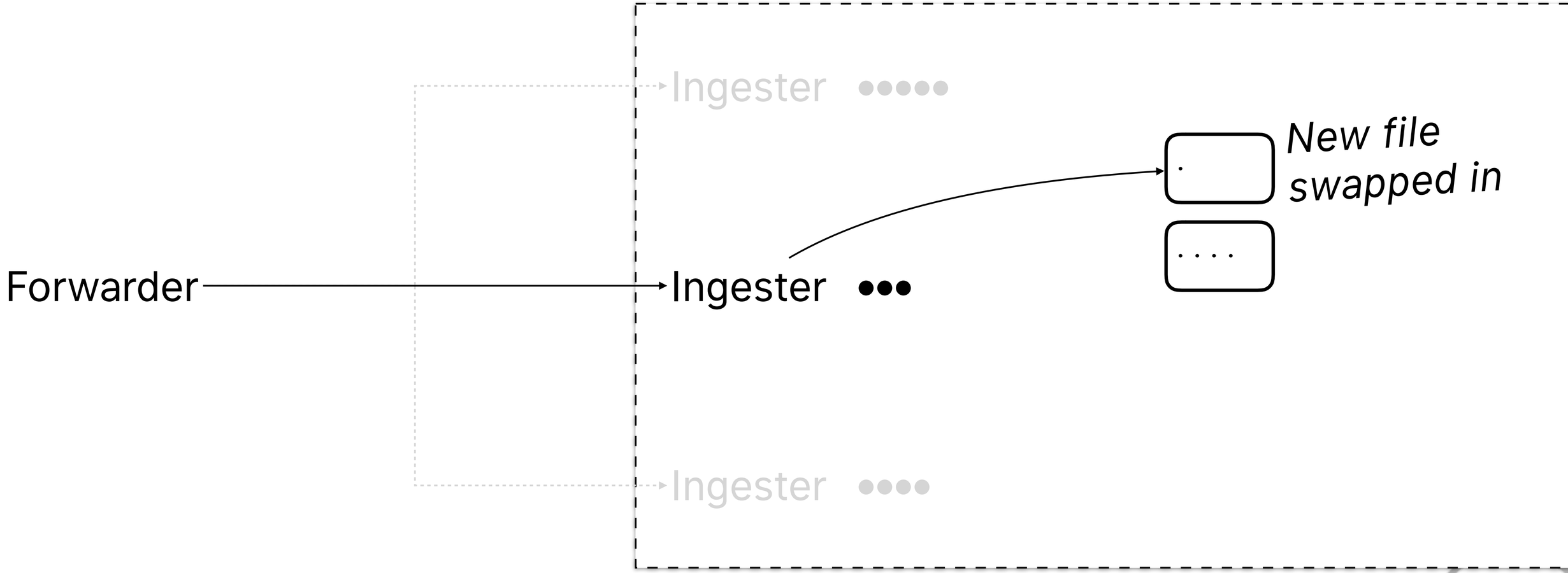
Ingest


Ingester
Size or time
limit reached
Forwarder
Ingester
Ingester

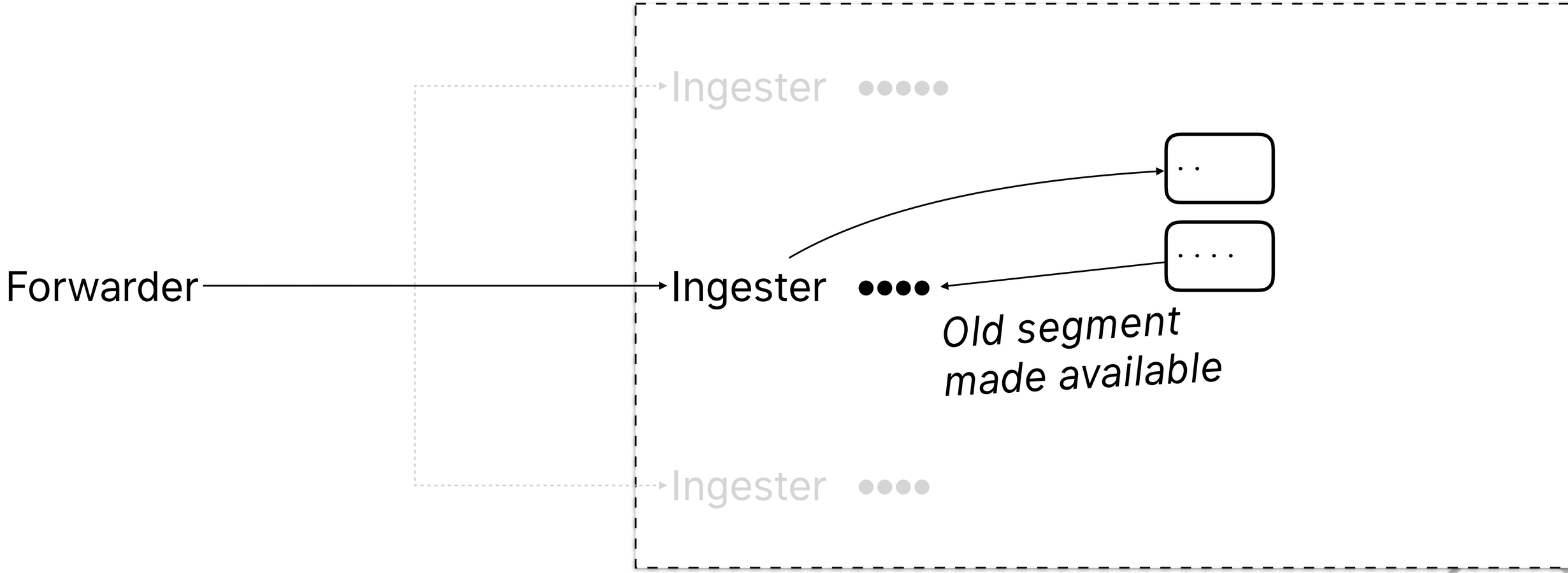
Ingest


Ingester
New segment
file created
Forwarder
Ingester
Ingester

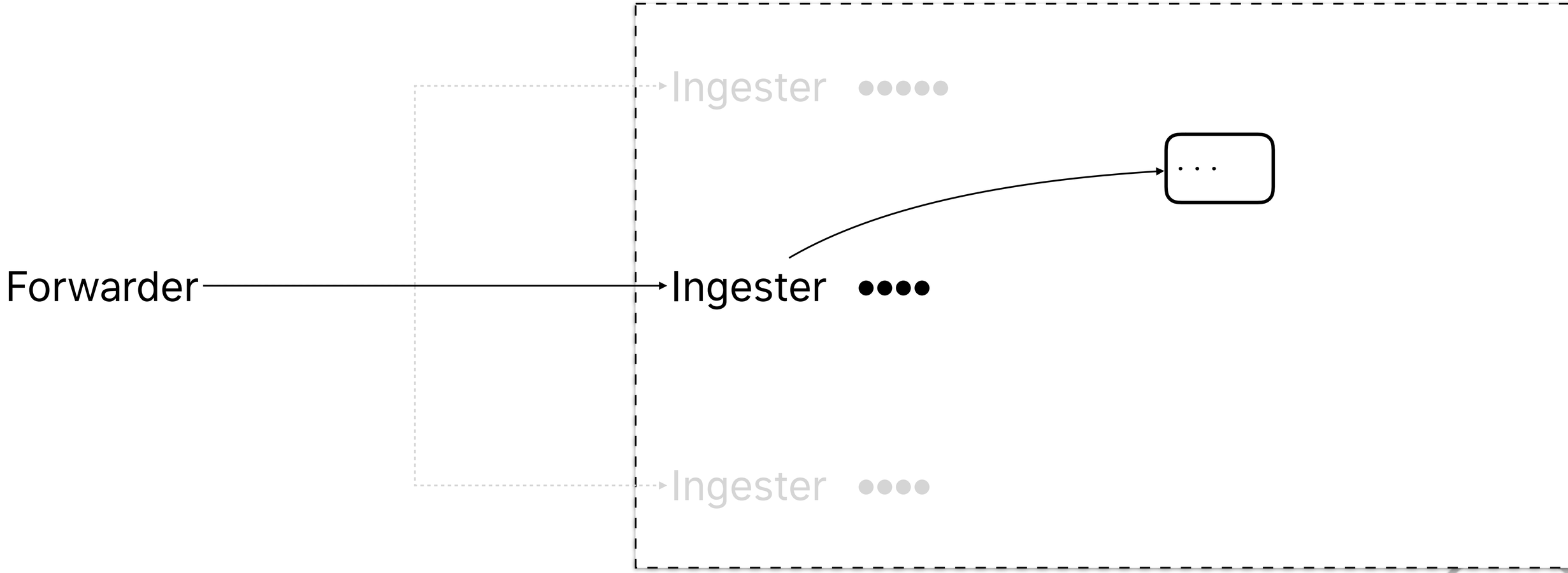
Ingest



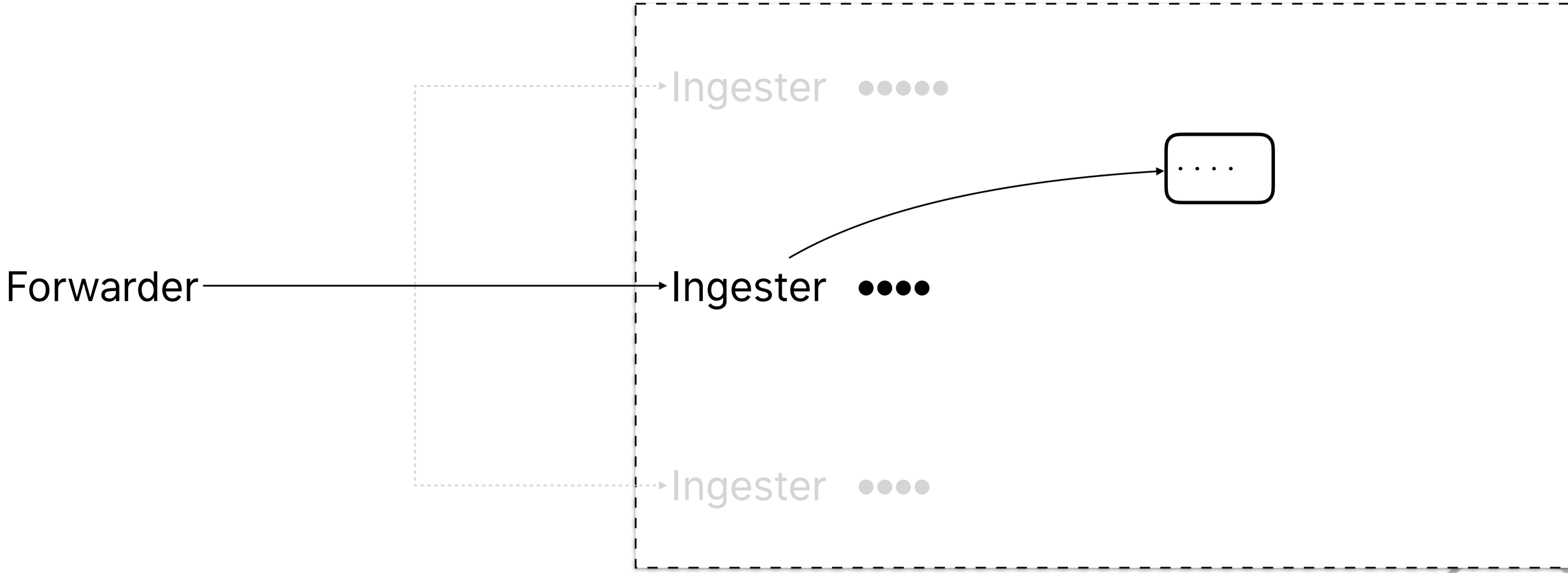
Ingest


Ingester
Forwarder
Ingester
Old segment
made available
Ingester

Ingest



Ingest



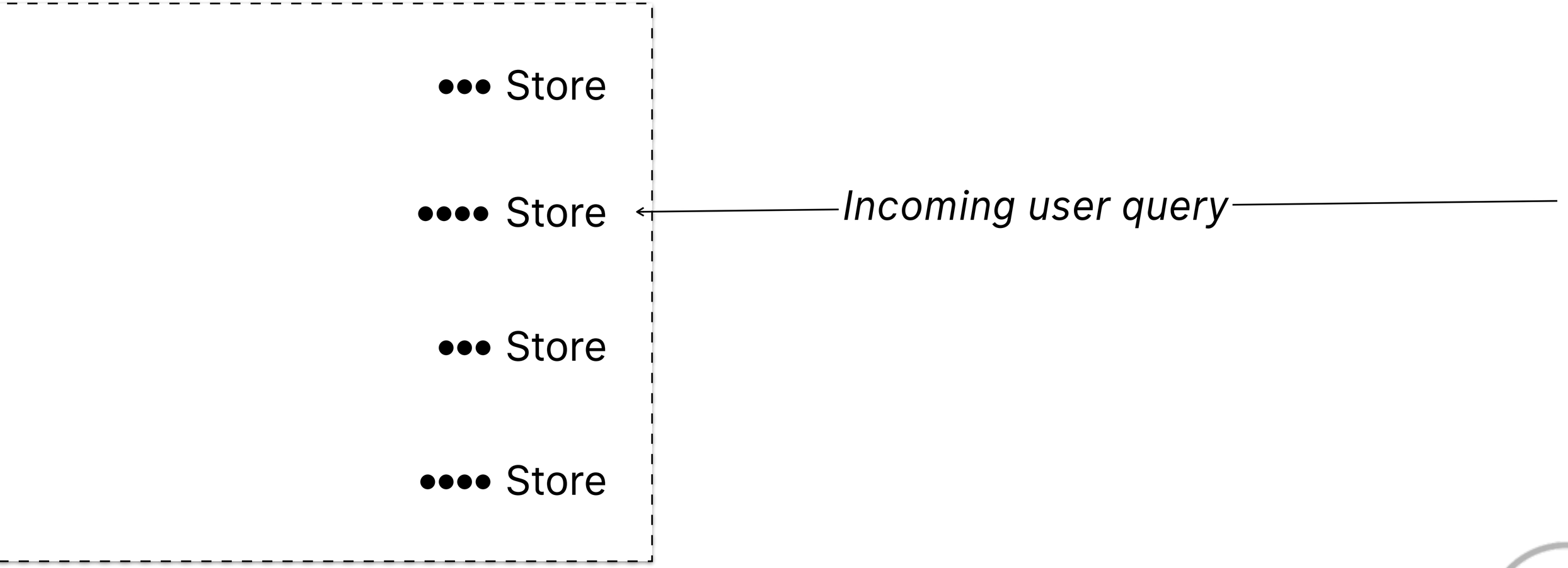
Priorities

- Writes are the hot path, they should succeed
- Everything is optimized to get writes to disk
- Segment size limits affect liveliness in high-throughput environments
- Segment time limits affect liveliness in low-throughput environments
- Reasonable defaults of 16MB and 1s respectively

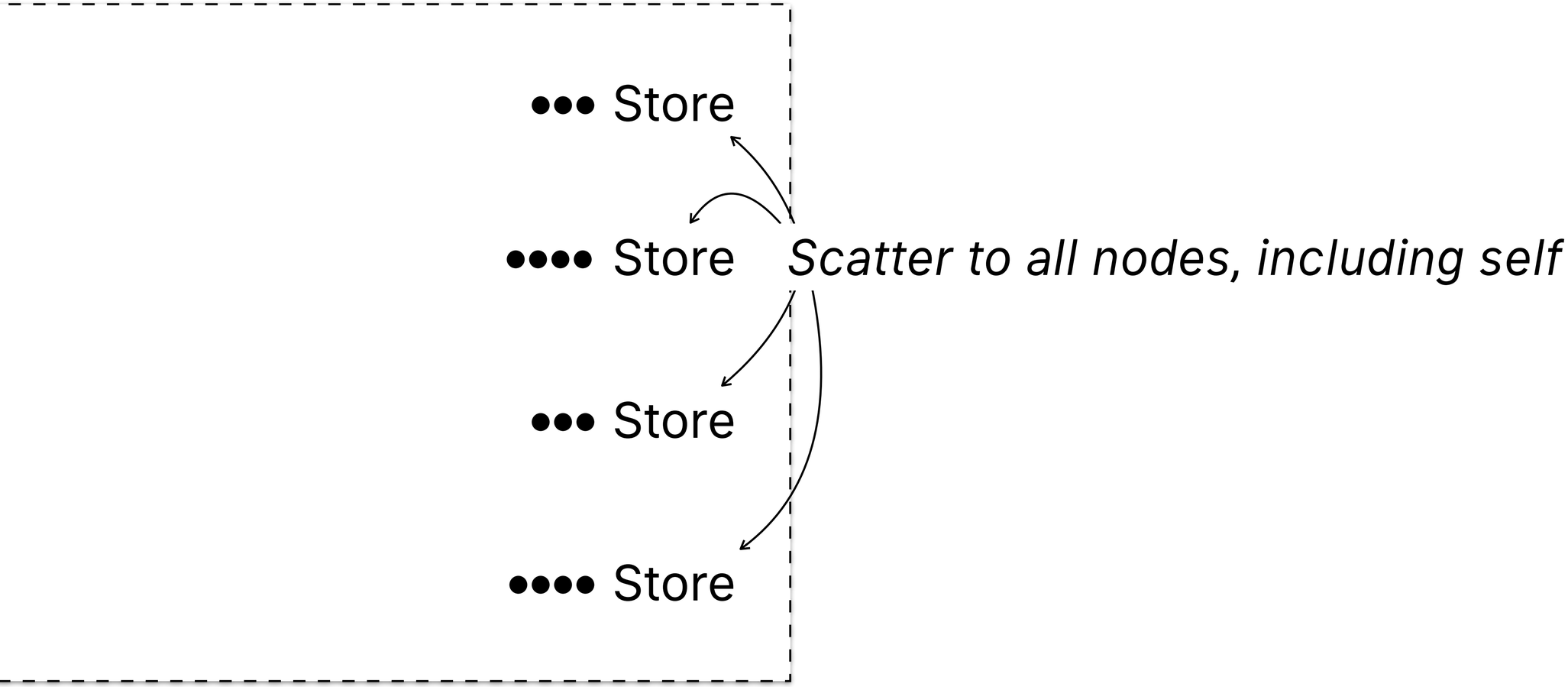
Query

Classic scatter/gather

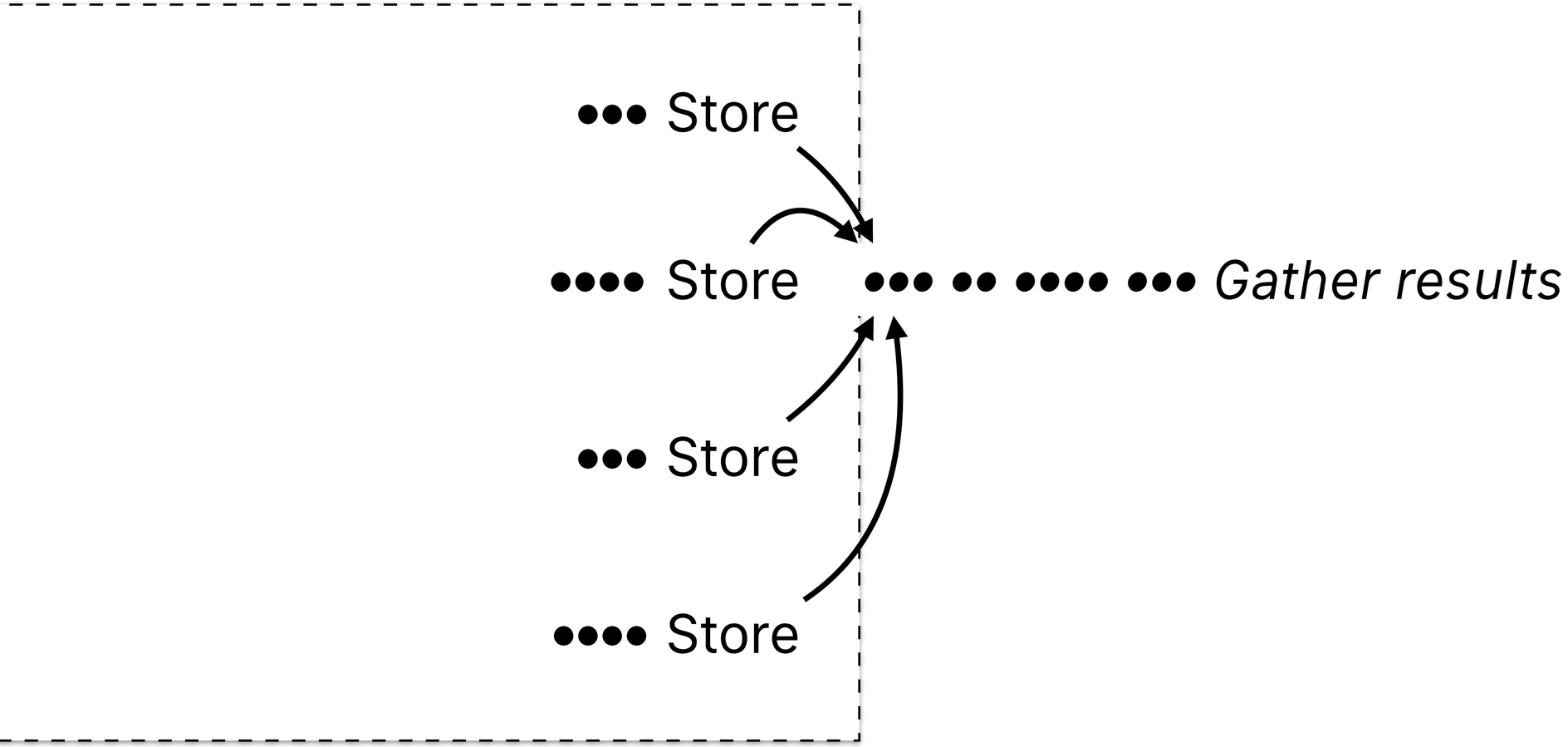
Query



Query


••• Store
•••• Store
••• Store
•••• Store
Scatter to all nodes, including self

Query



Query

••• Store

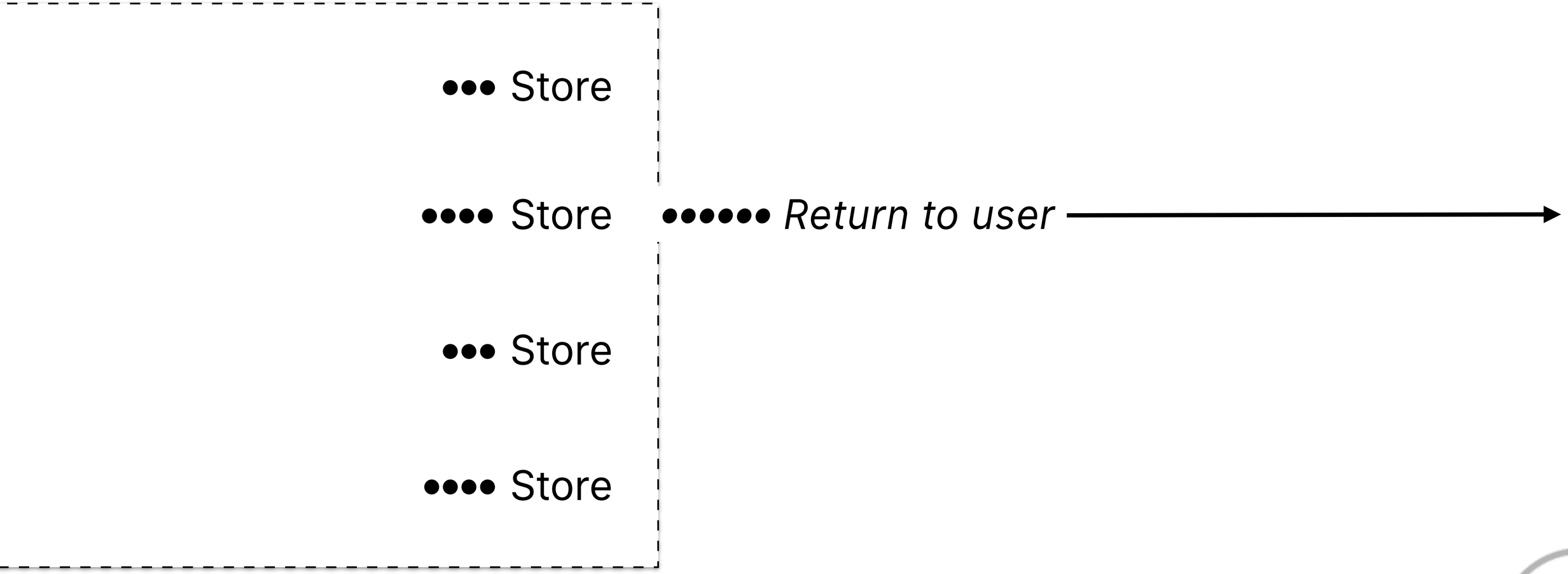
•••• Store

••••• *Merge and deduplicate*

••• Store

•••• Store

Query



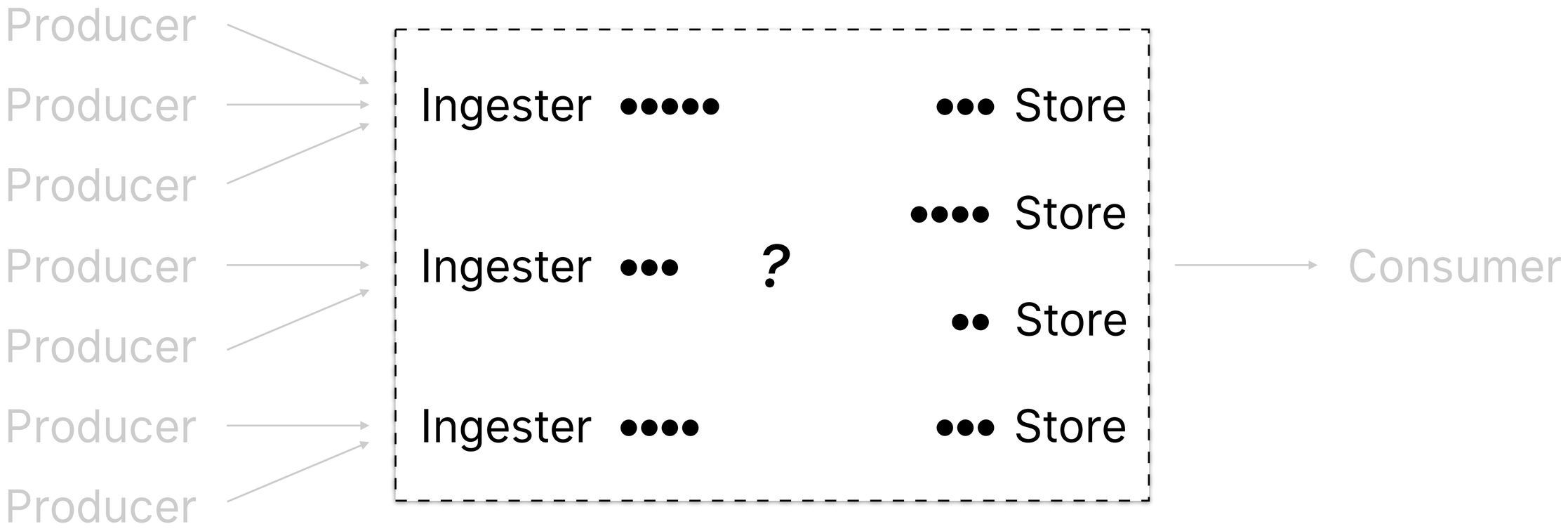
Tradeoffs

- True: queries are only as fast as your slowest machine
- Need aggressive timeouts to maintain liveliness
- Still, we can tolerate partial node failure without compromising results
- And more substantial node failure degrades gracefully
- Since we get duplicates, we can also do basic read-repair

Replication

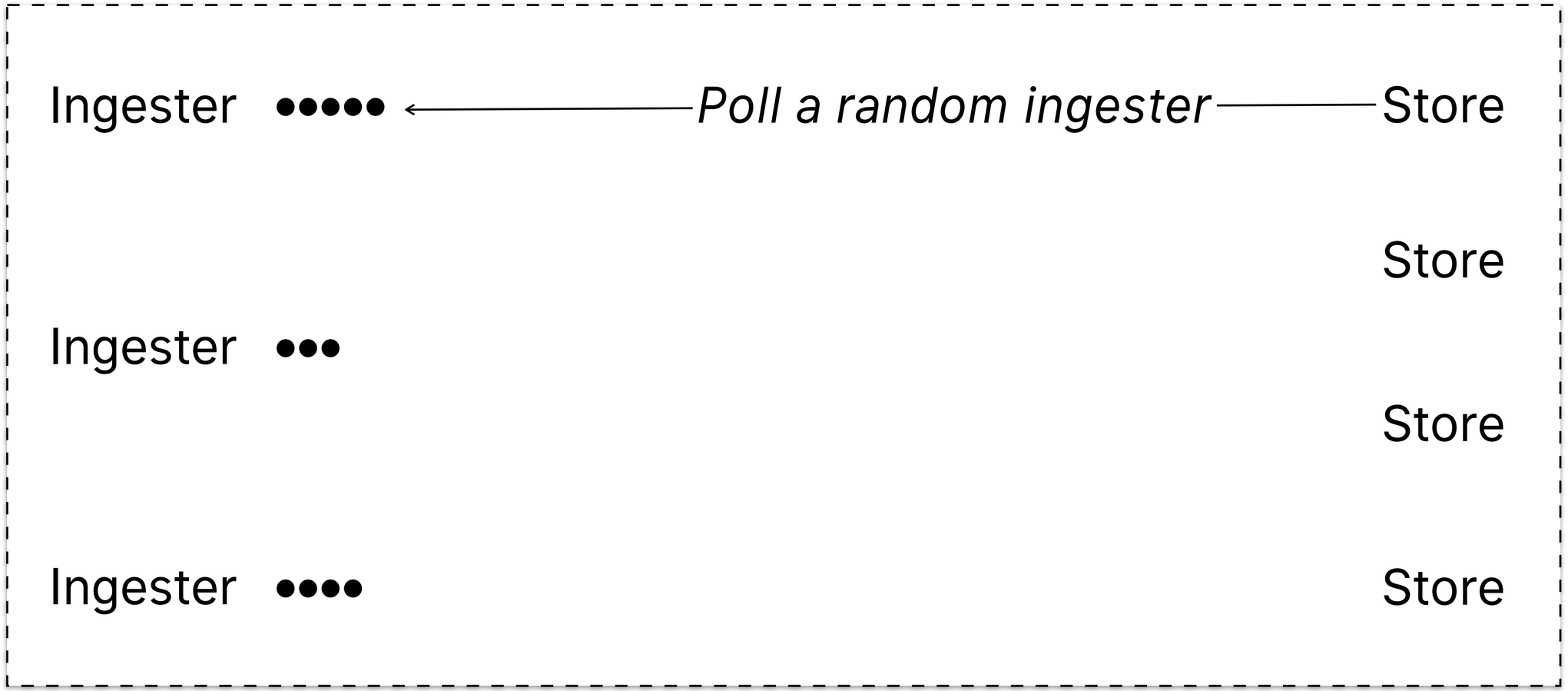
Inspired by Prometheus

Replication

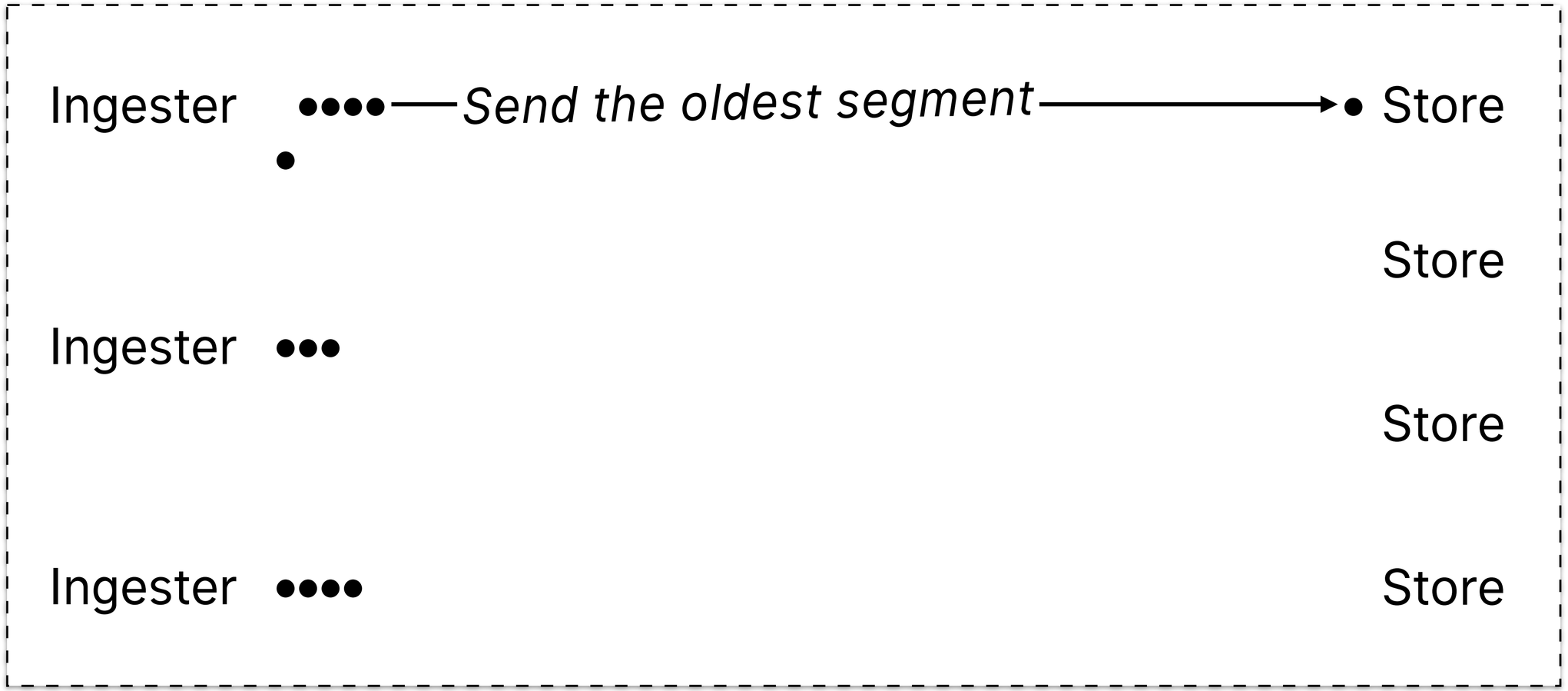


Replication

Replication

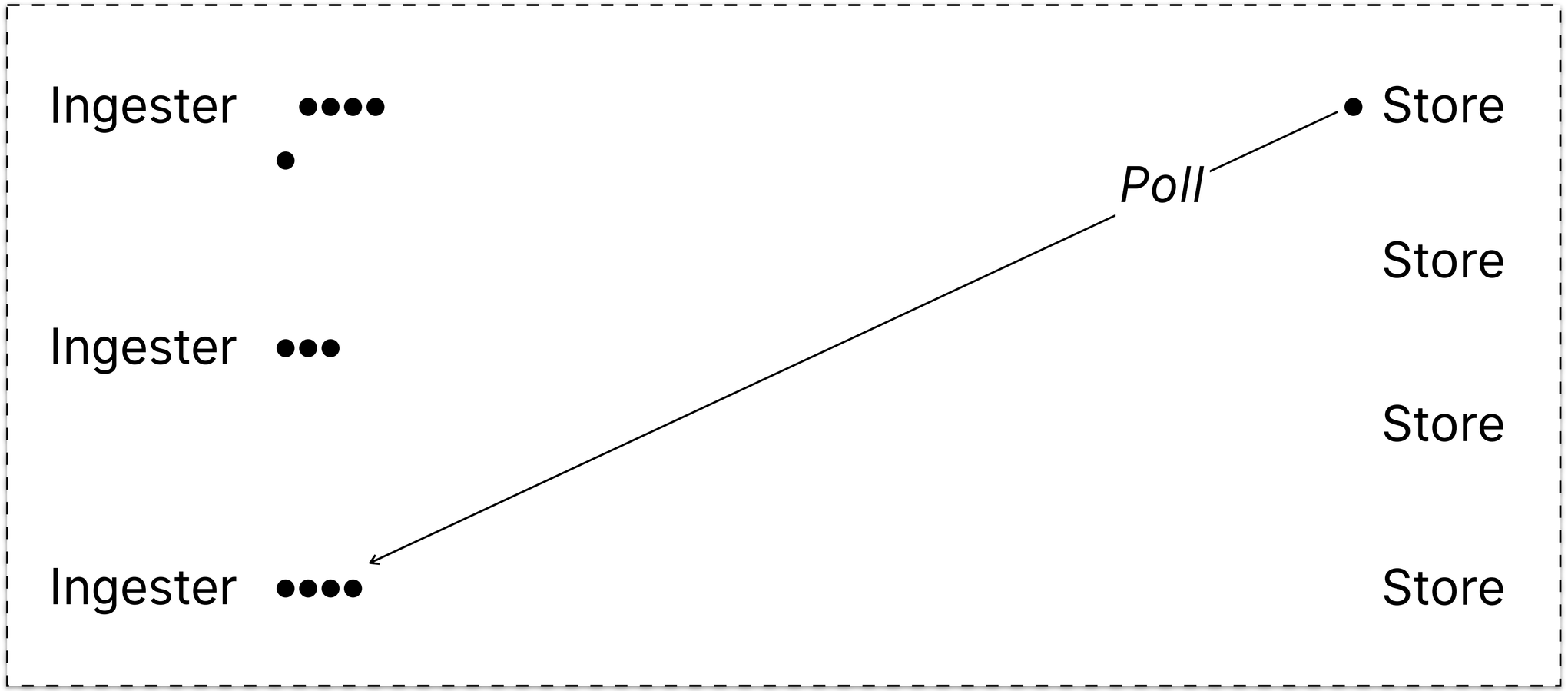


Replication

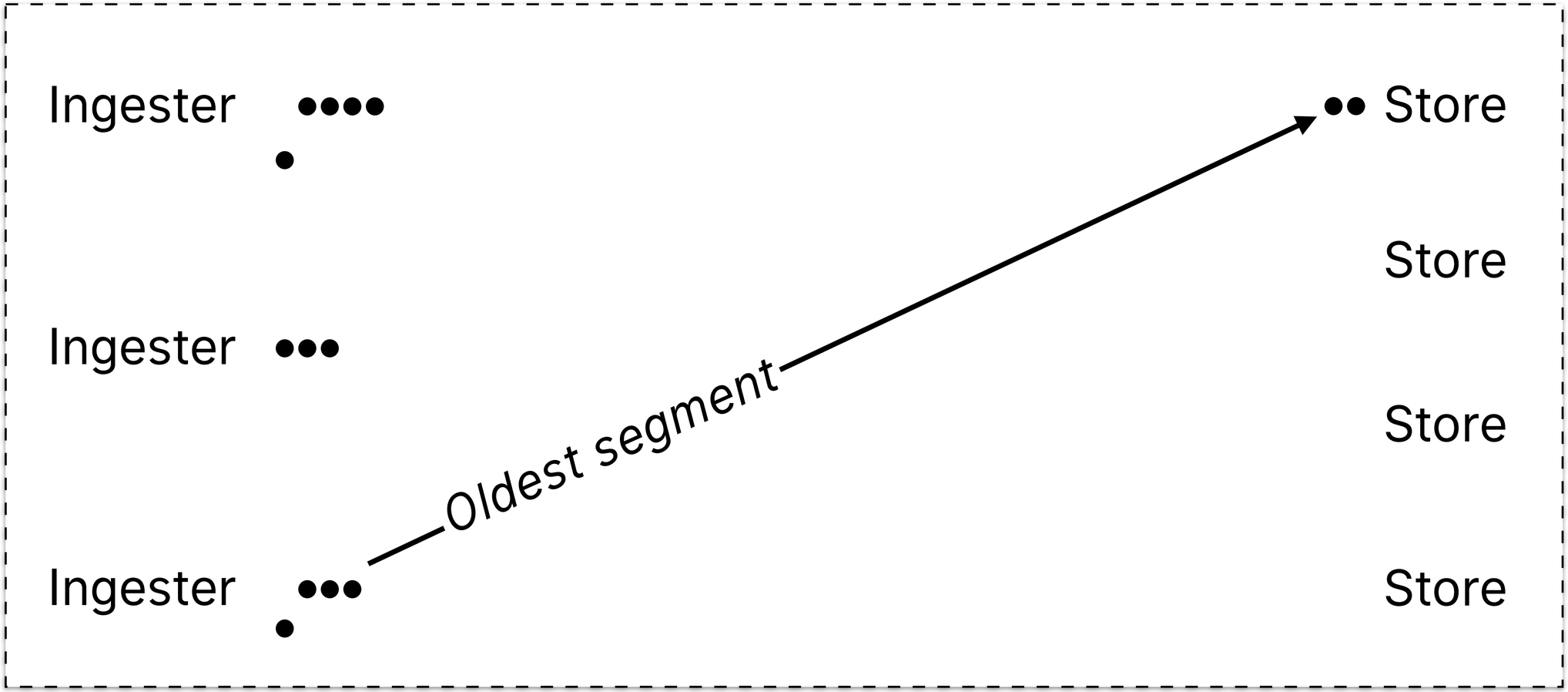


Replication

Replication

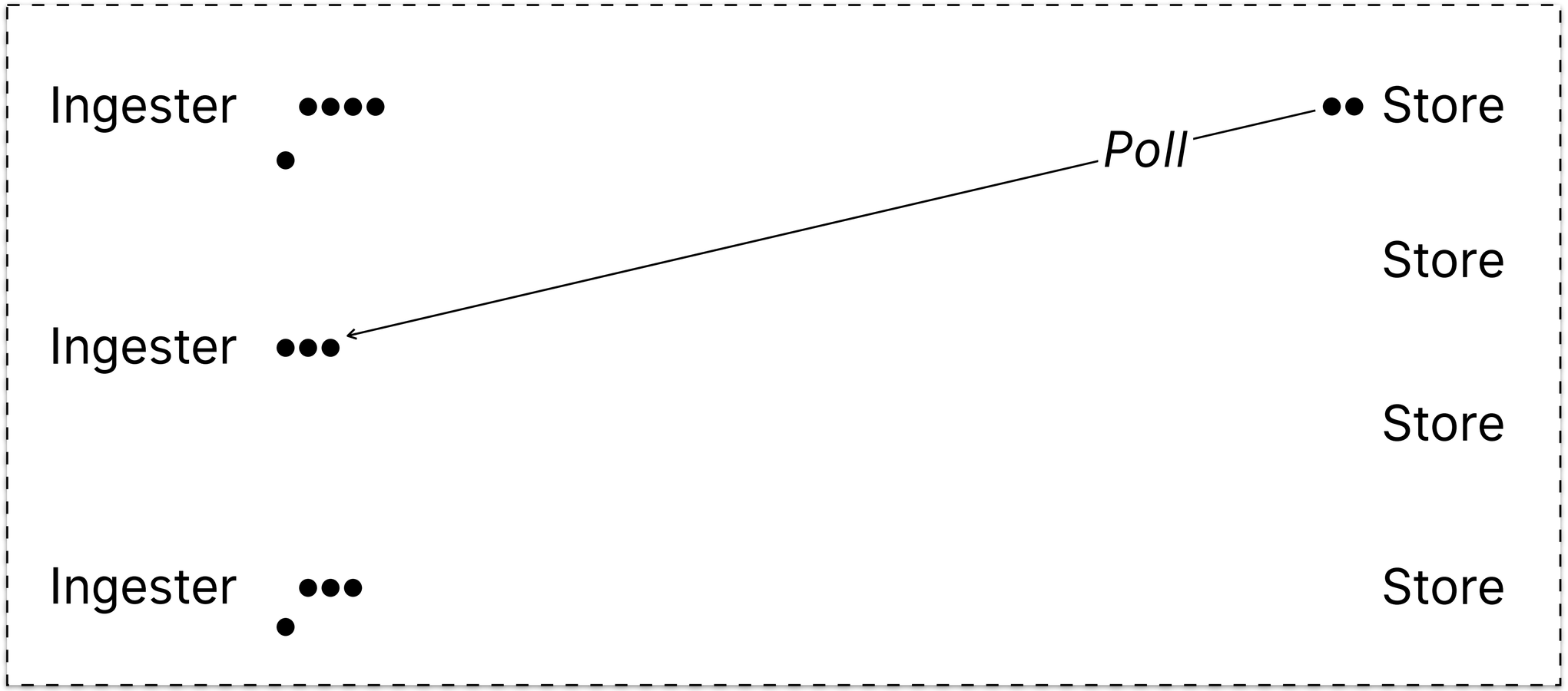


Replication

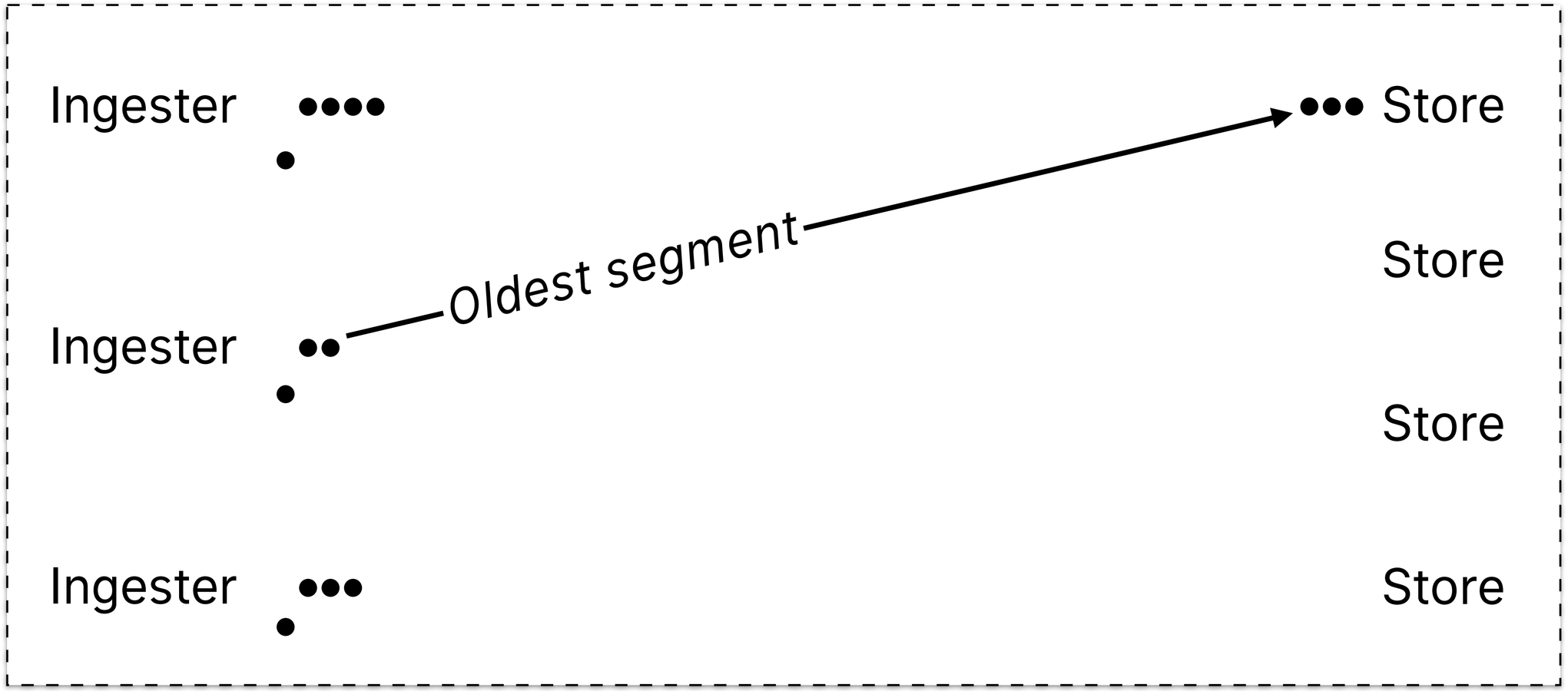


Replication

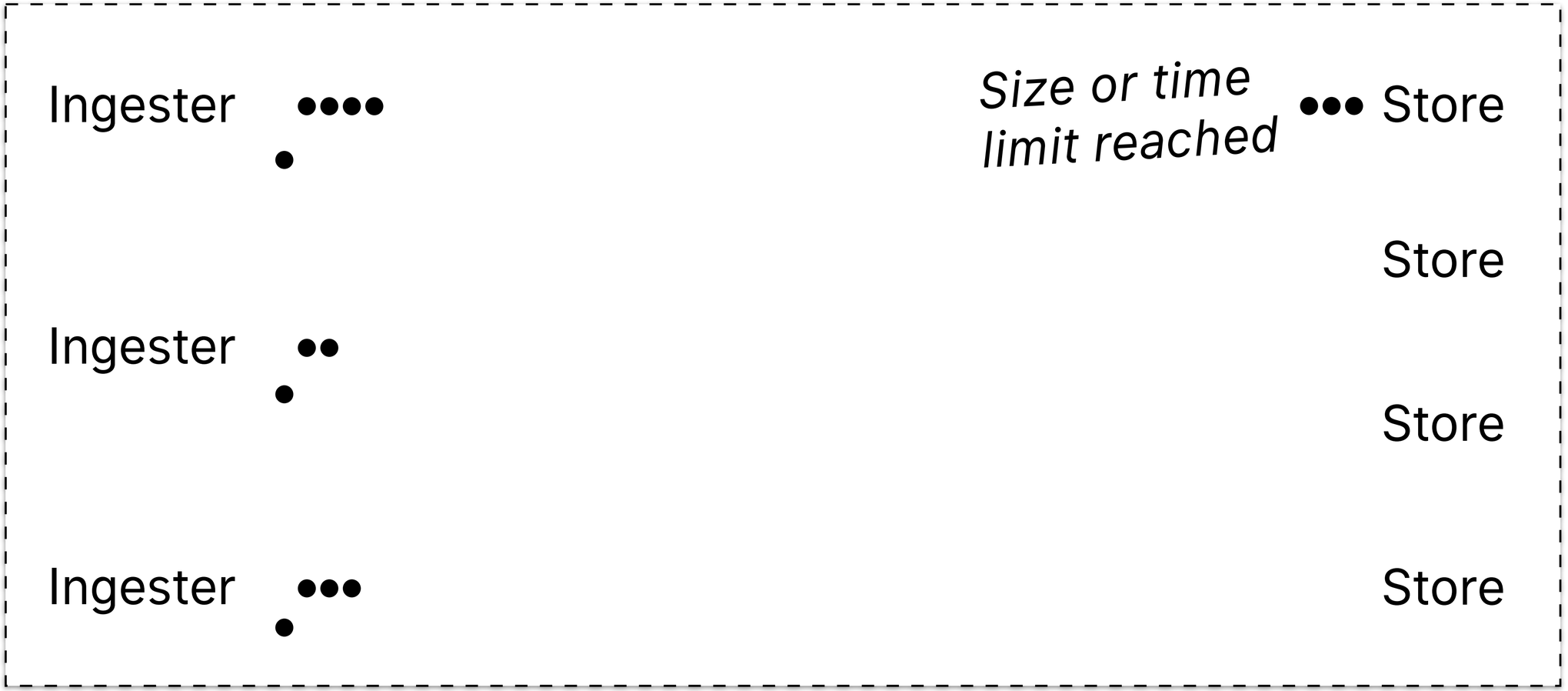
Replication



Replication



Replication



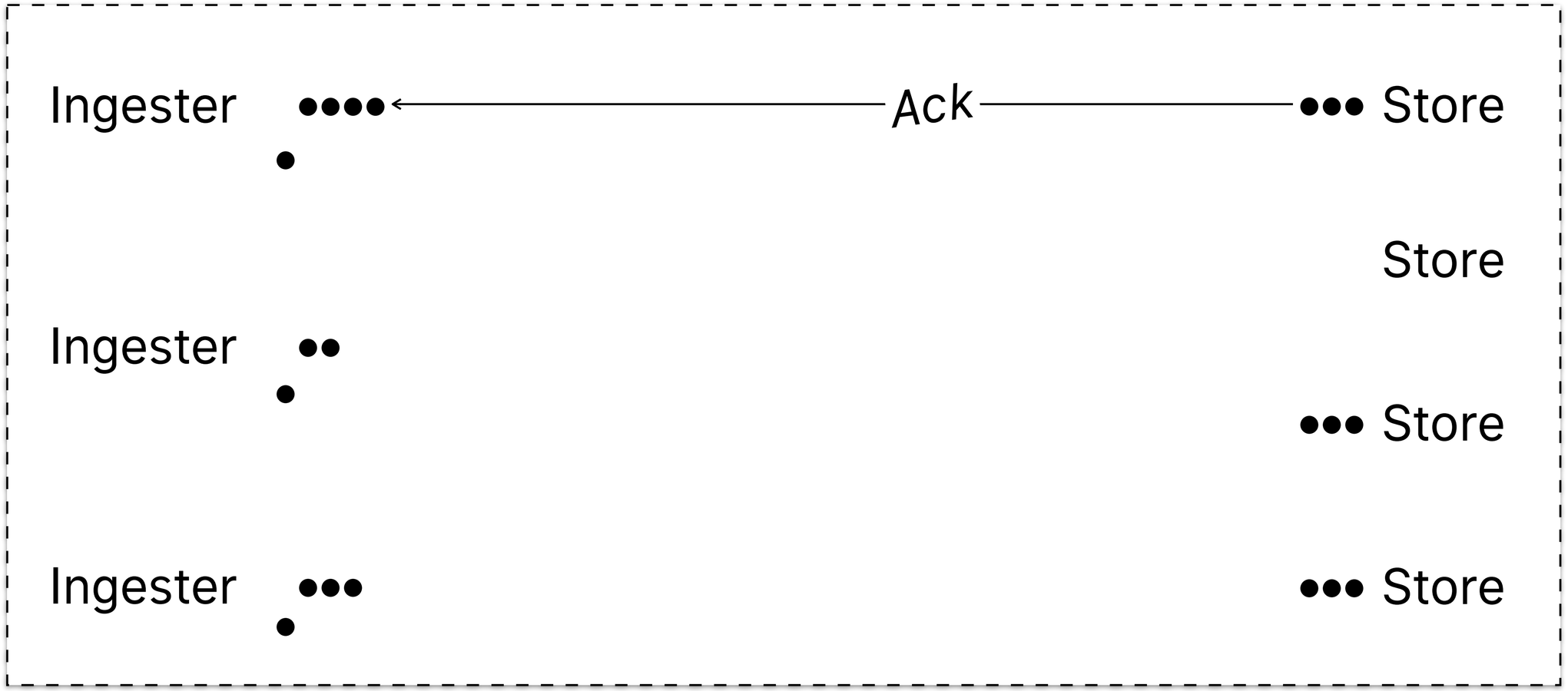
Replication

Replication

Replication

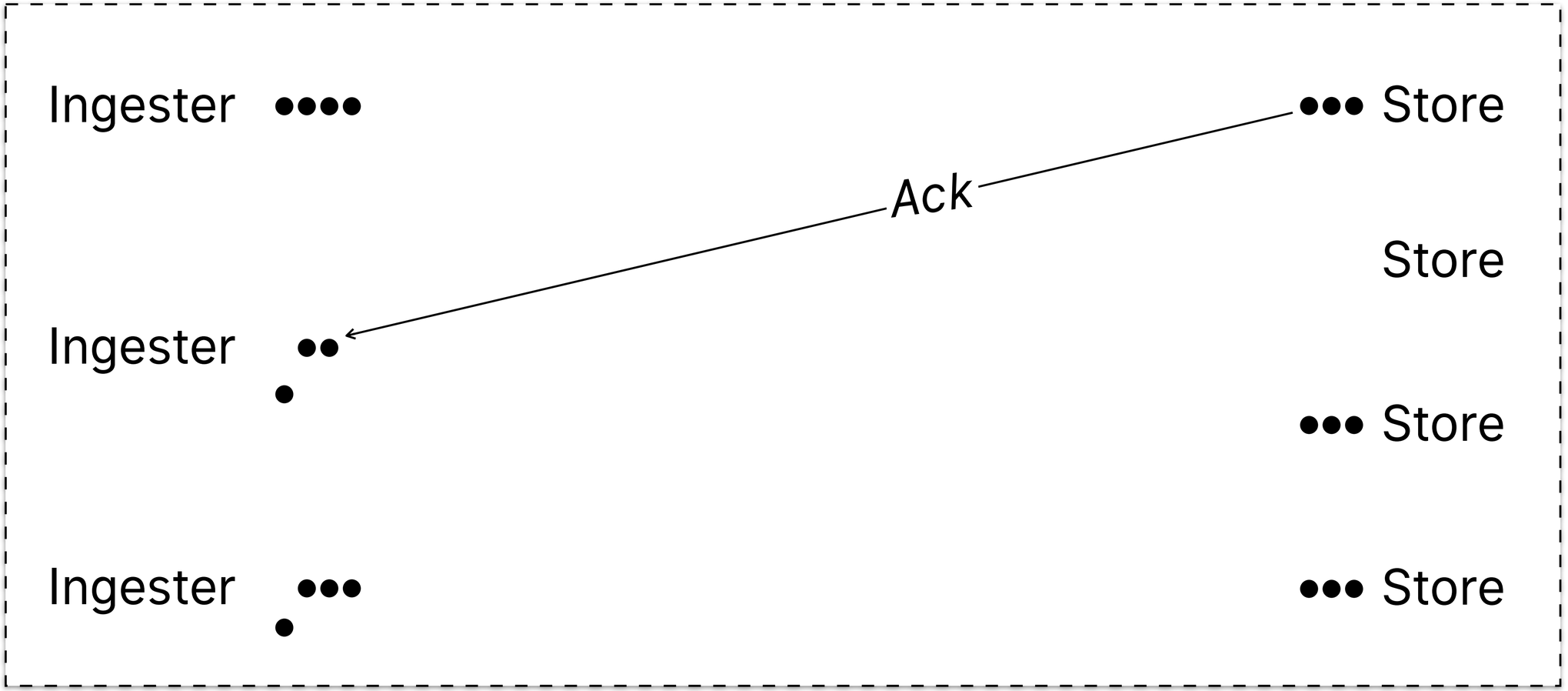
Replication

Replication



Replication

Replication



Replication

Ingester ••••

Ingester ••

Ingester •••
•

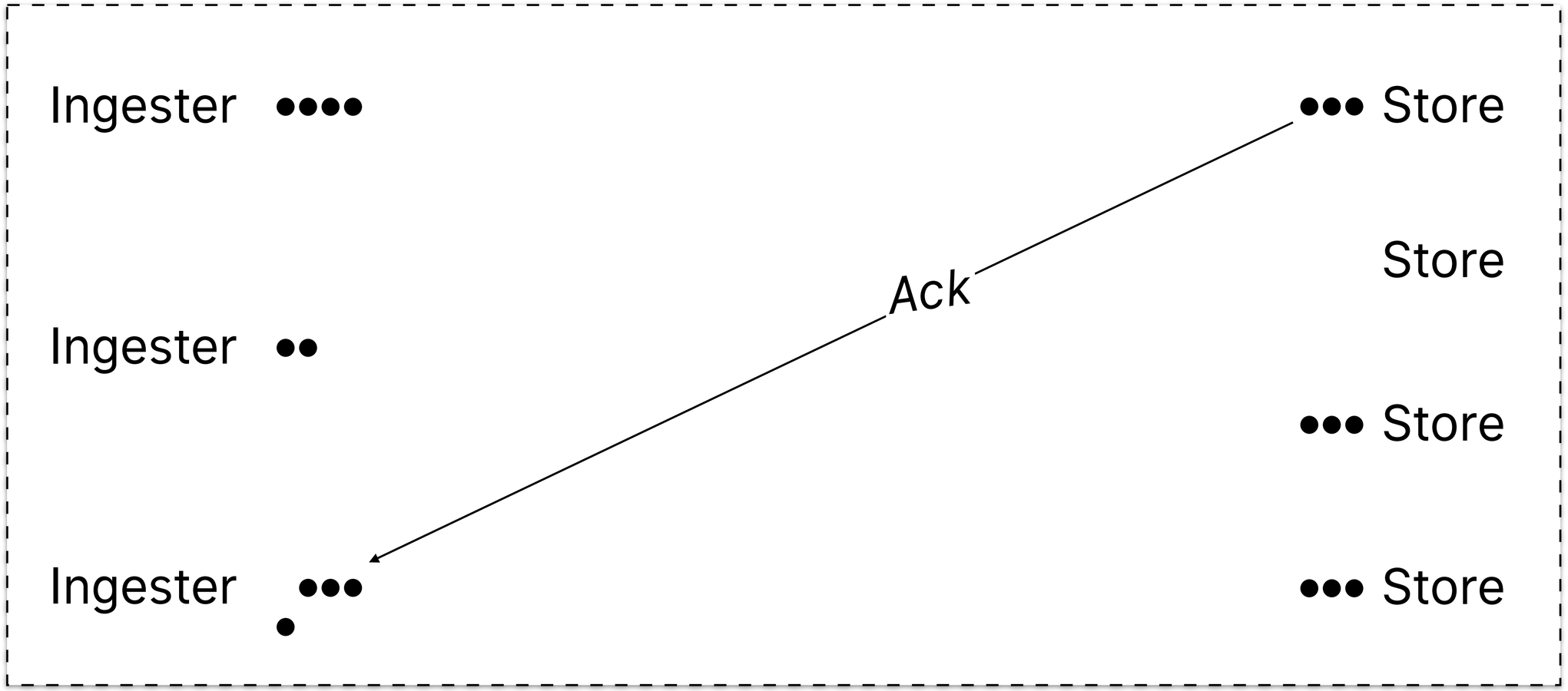
••• Store

Store

••• Store

••• Store

Replication



Replication

Ingester ••••

Ingester ••

Ingester •••

••• Store

Store

••• Store

••• Store

Transactional coördination

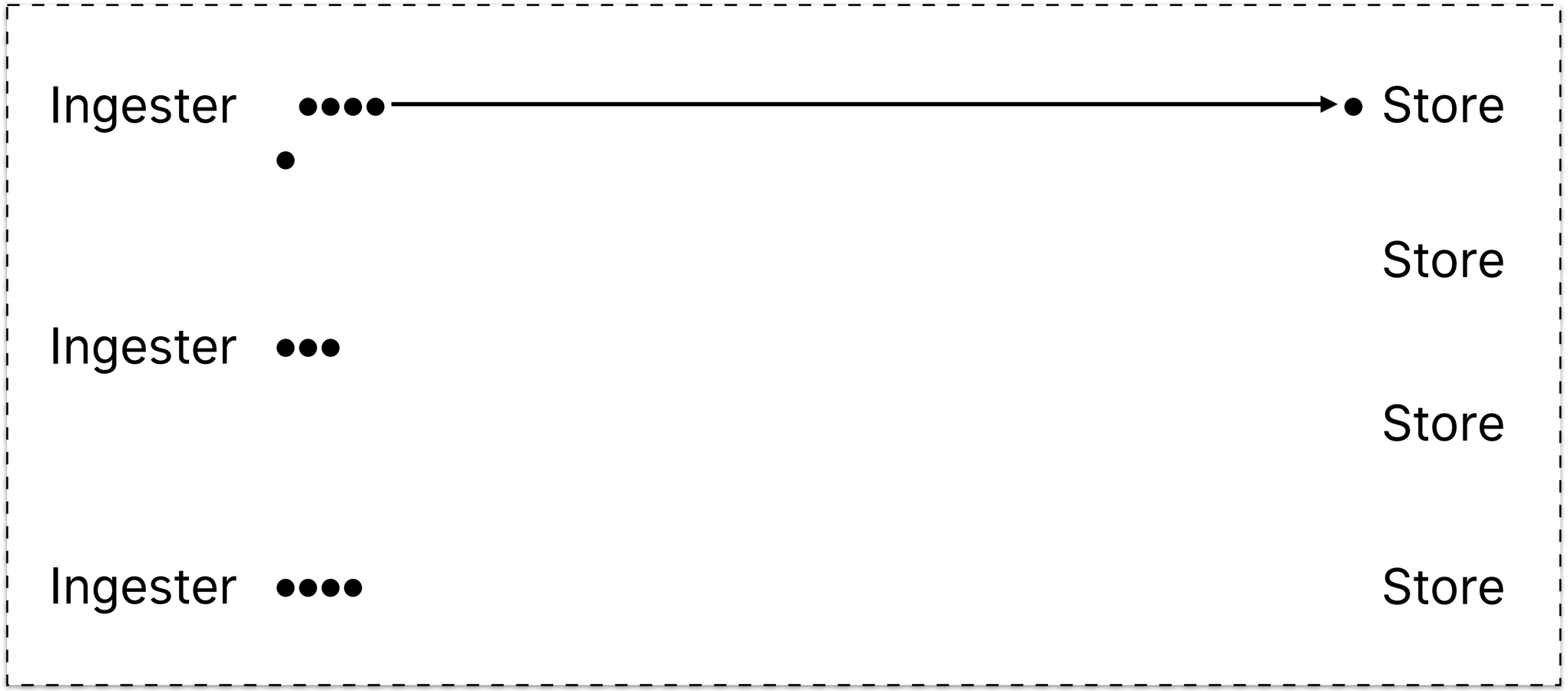
- Okay, it's not totally coördination-free 😐
- We have coördination within each gather/replicate transaction
- But those transactions are short-lived and tolerate failure well

Failure during gather

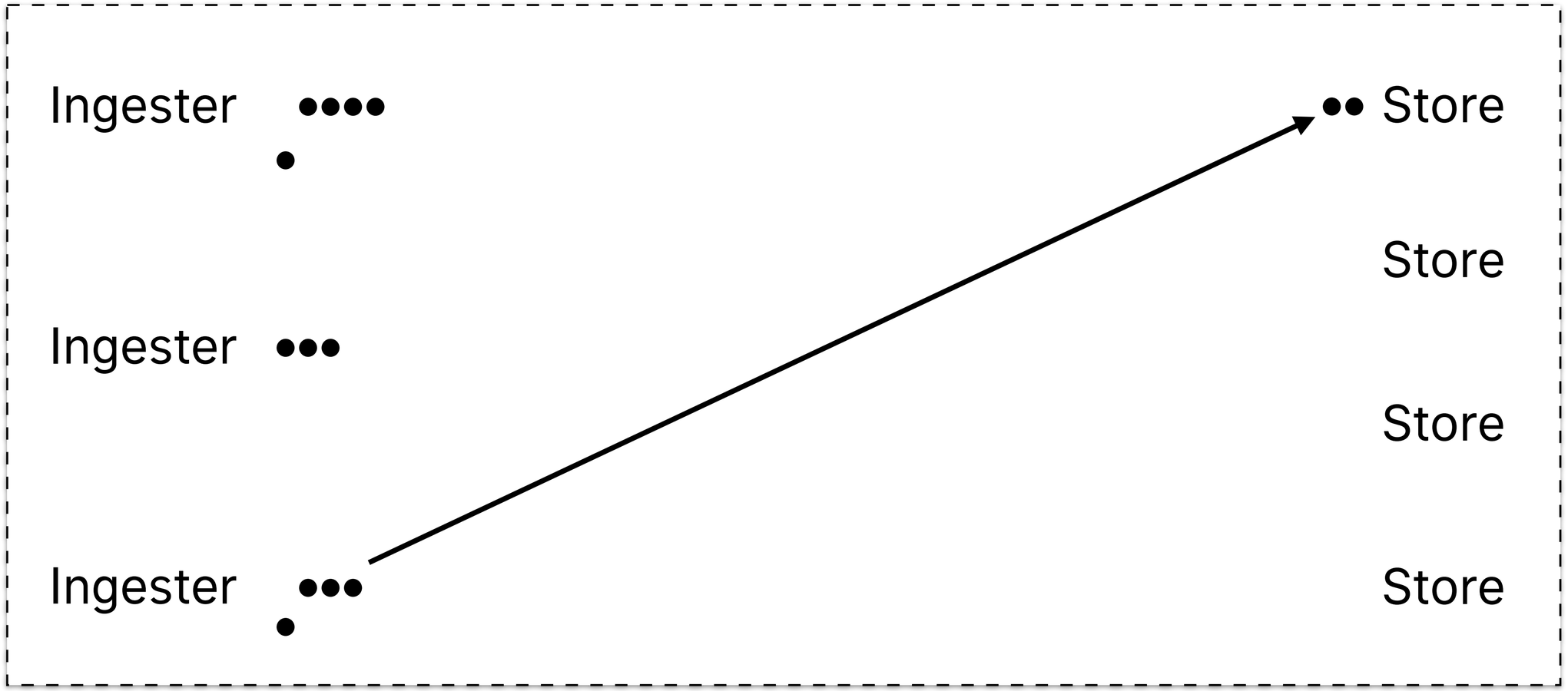
Timeouts to the rescue

Failure during gather

Failure during gather



Failure during gather



Failure during gather

Failure during gather

Failure during gather

Failure during gather

Failure during gather

Ingester •••••

•• Store

Store

Ingester •••

Store

Ingester ••••

Store

Failure during replicate

We've got dupes!

Failure during replicate

Failure during replicate

Failure during replicate

Failure during replicate

Failure during replicate

Ingester •••••
Timeout

••• Store

Store

Ingester •••

••• Store

Ingester ••••
Timeout

Store

Failure during replicate

Ingester •••••

Ingester •••

Ingester ••••

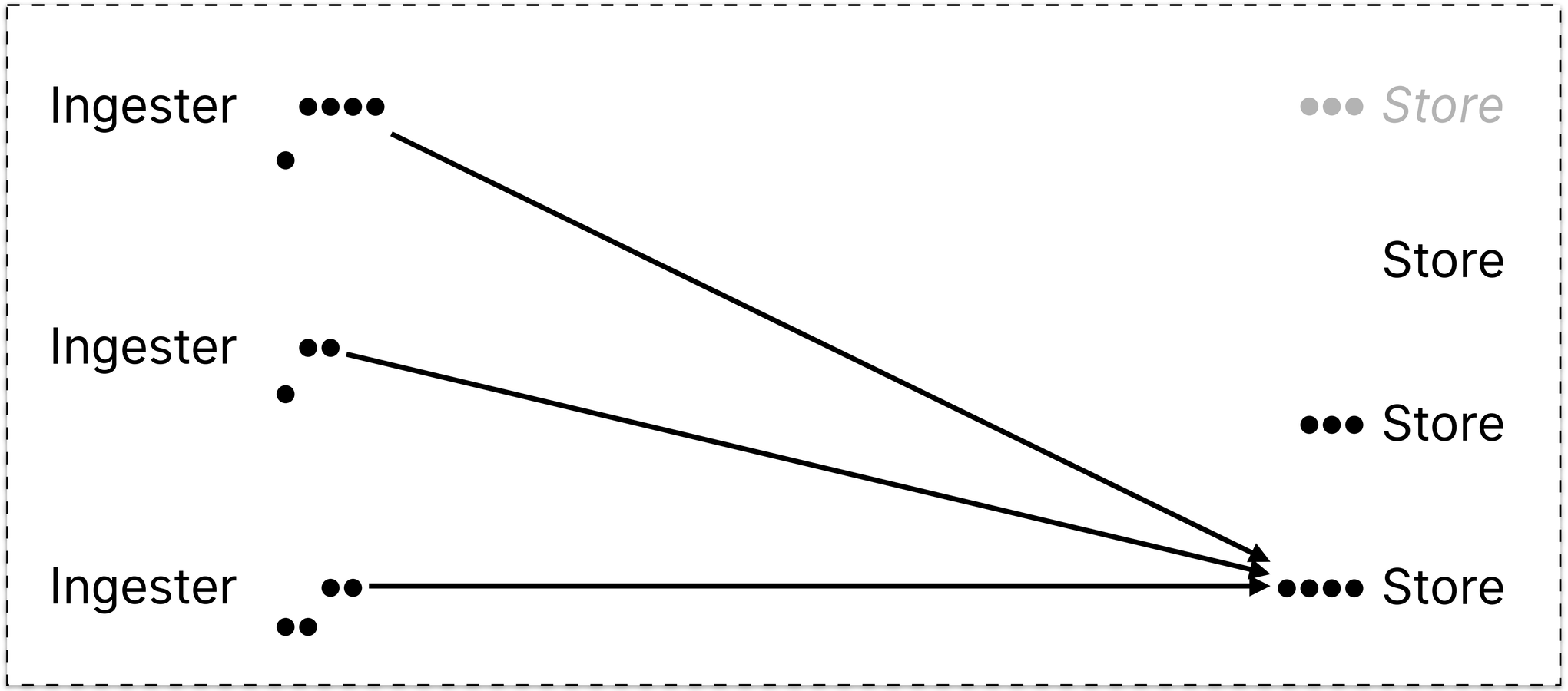
••• Store

Store

••• Store

Store

Failure during replicate

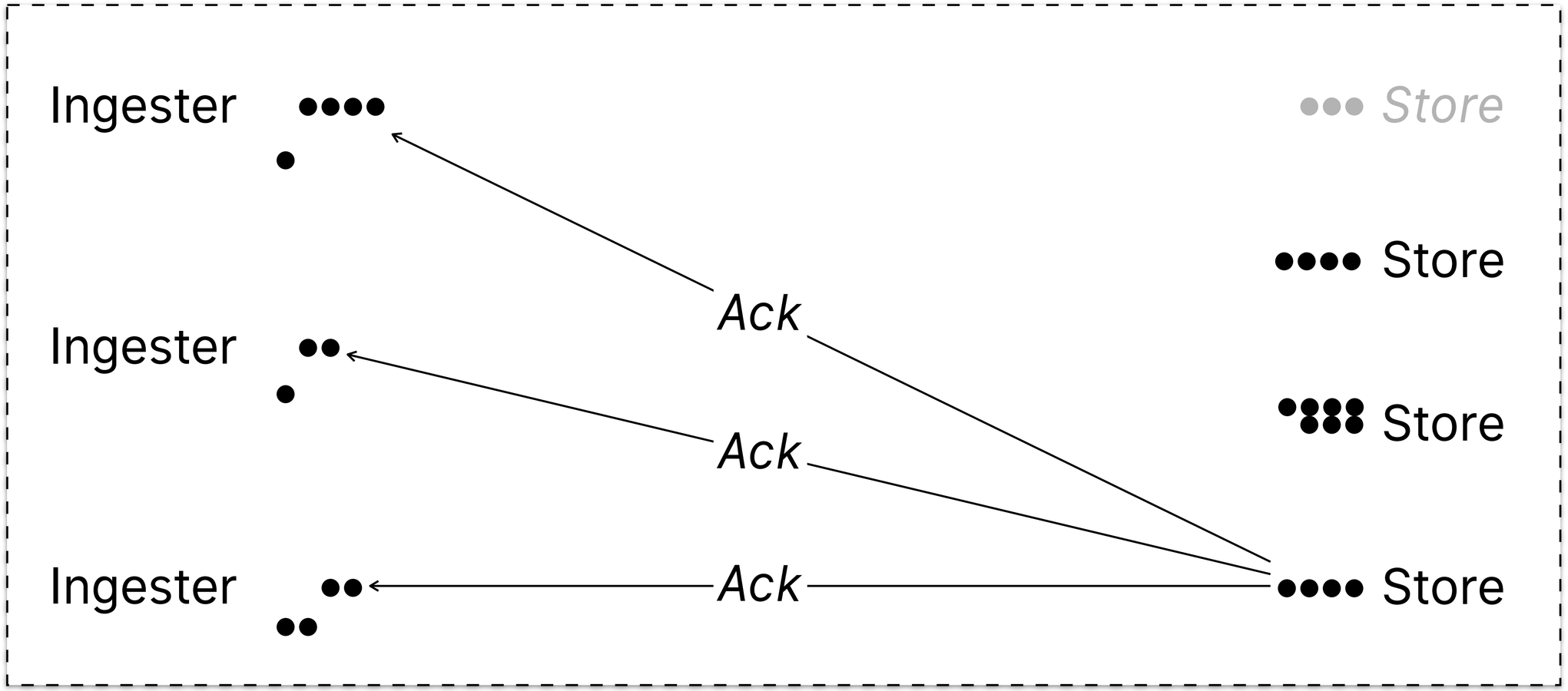


Failure during replicate

Failure during replicate

Failure during replicate

Failure during replicate



Failure during replicate

Ingester ●●●●

Ingester ●●

Ingester ●●

●●● *Store*

●●●● Store

●●●● Store

●●●● Store

Failure during replicate

Ingester ••••

••• Store

•••• Store

Ingester ••

Dupes!

•••• Store

Ingester ••

•••• Store

Dupes don't matter

Store

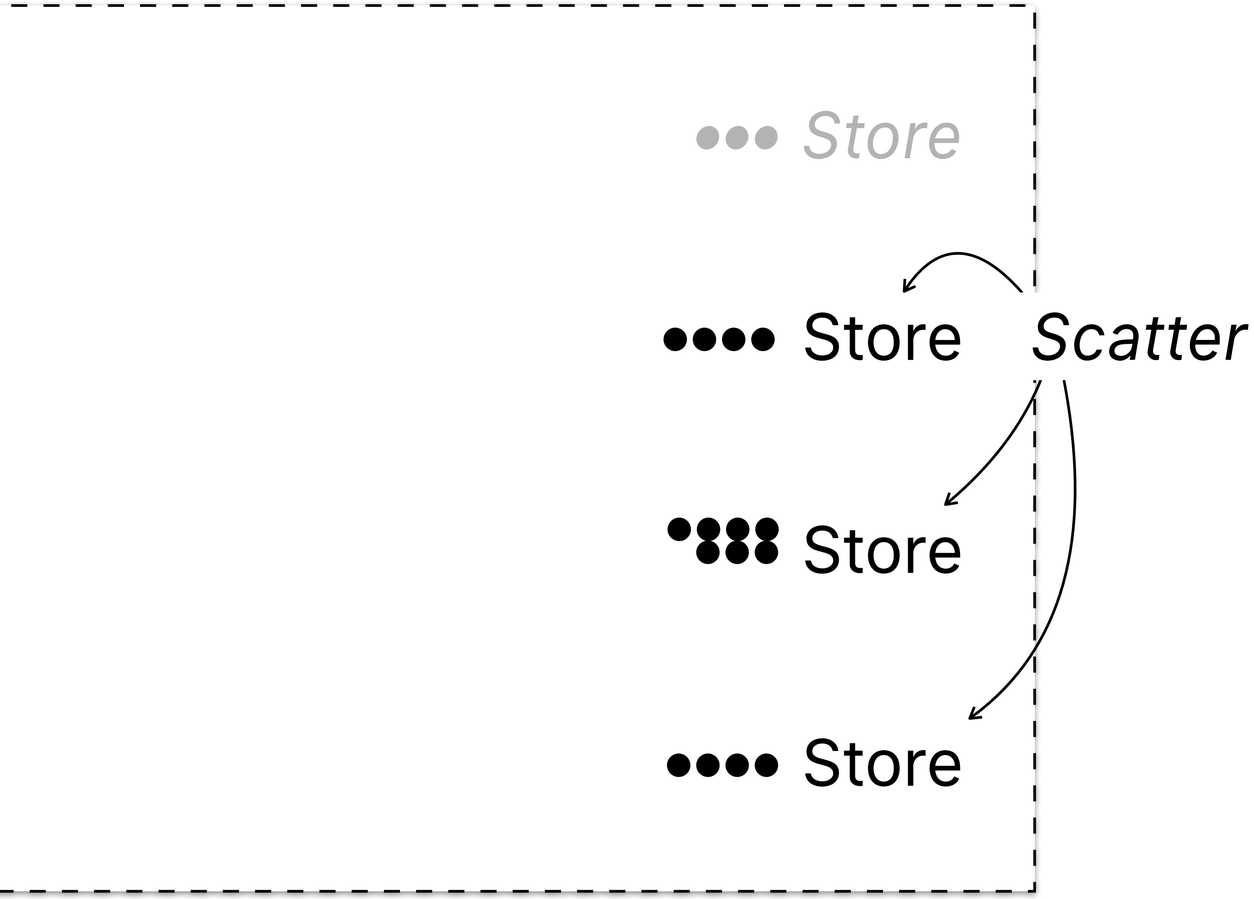
Store

Store

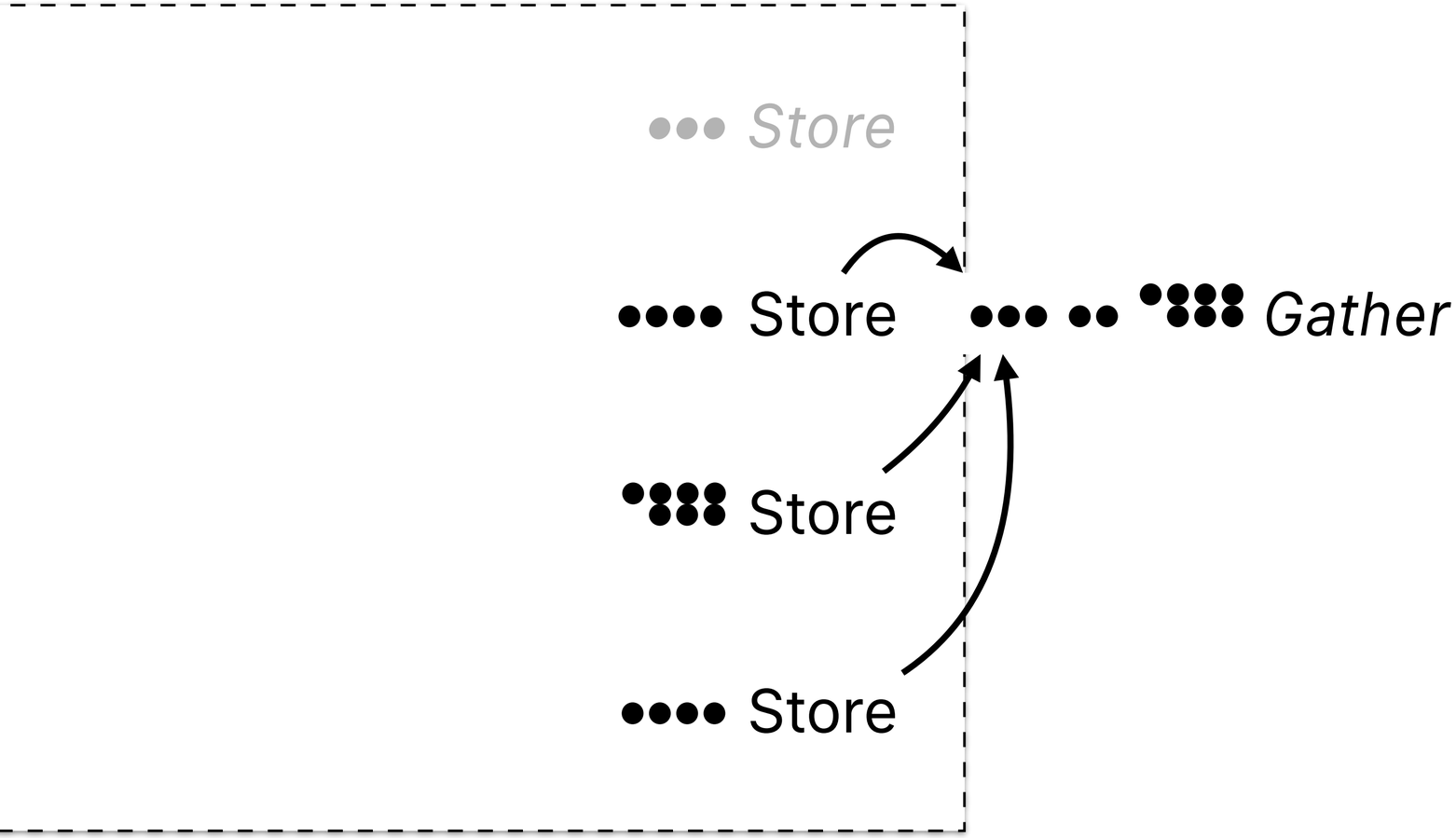
Store

Dupes don't matter

Dupes don't matter



Dupes don't matter



Dupes don't matter

Store

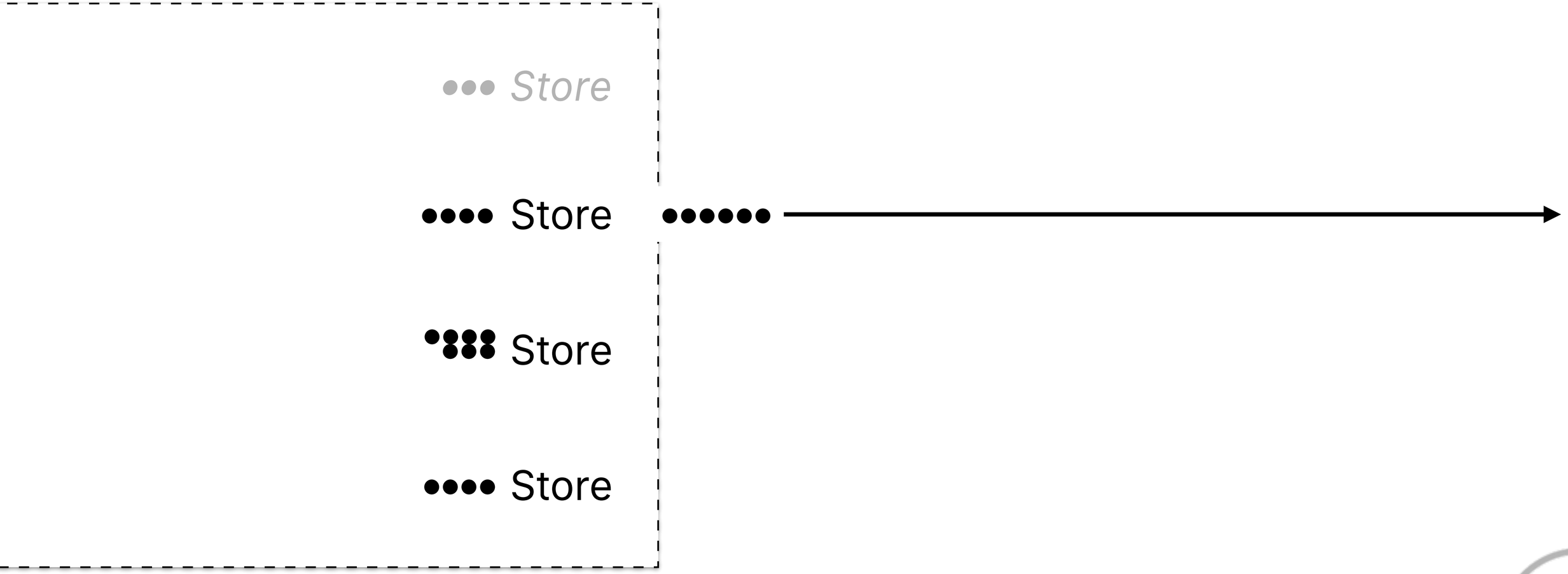
Store

Store

Store

Merge and dedupe

Dupes don't matter



Dupes get compacted

Ingester ••••

••• *Store*

•••• Store

Ingester ••

Store

Ingester ••

•••• Store

Dupes get compacted

Dupes get compacted

Dupes get compacted

Ingester ••••

Ingester ••

Ingester ••

••• *Store*

•••• Store

•••• Store

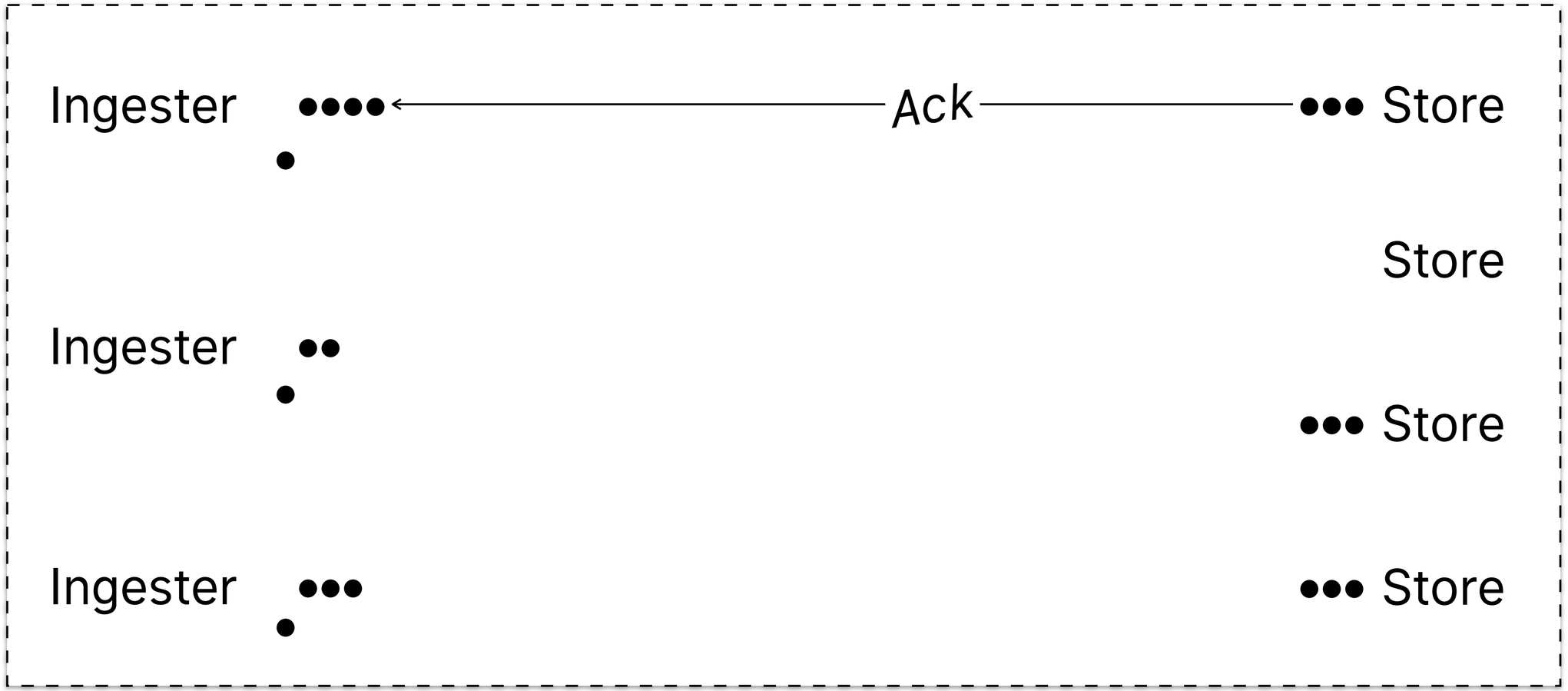
•••• Store

Failure during acks

Just a degenerate case

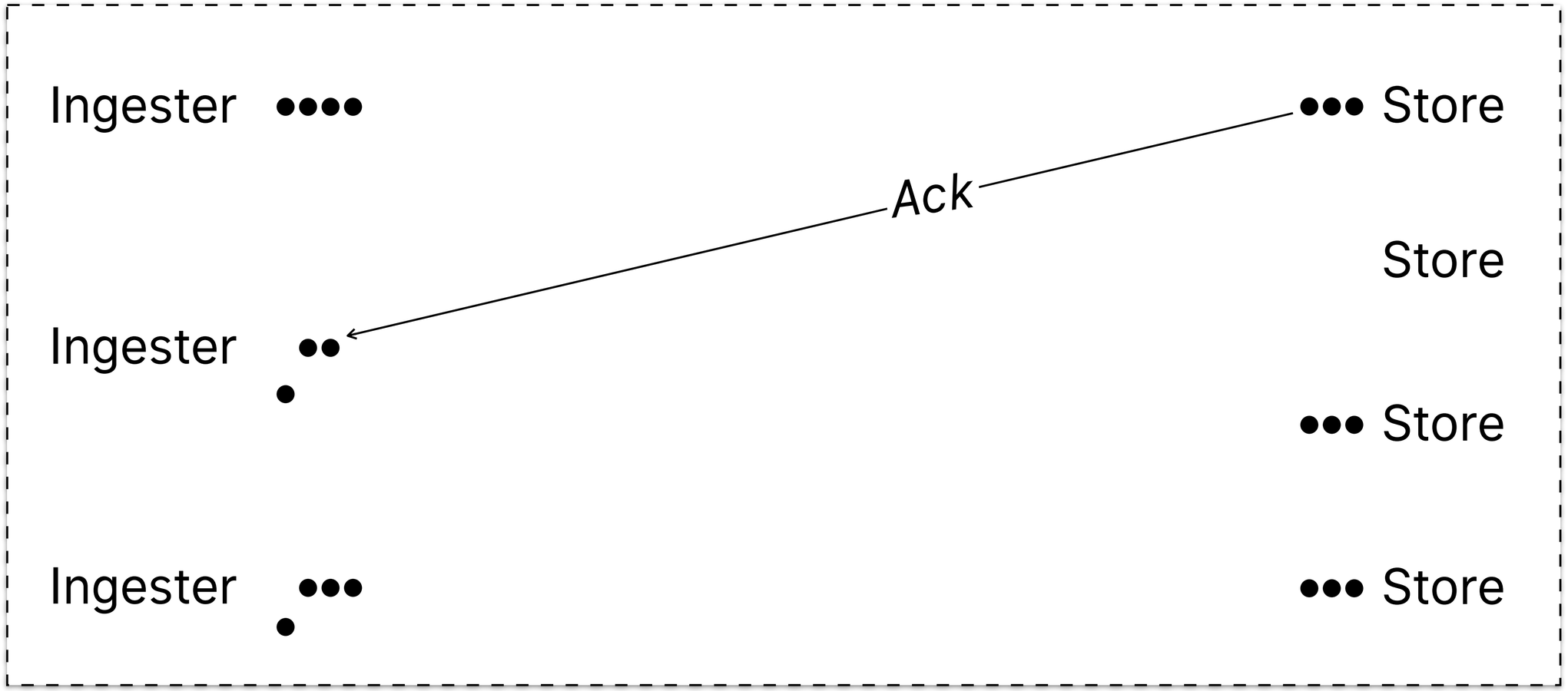
Failure during acks

Failure during acks



Failure during acks

Failure during acks



Failure during acks

Failure during acks

Failure during acks

Ingester ••••

••• Store

Store

Ingester ••

••• Store

Ingester ••••

Timeout

••• Store

Failure during acks

Ingester ••••

••• Store

Store

Ingester ••

••• Store

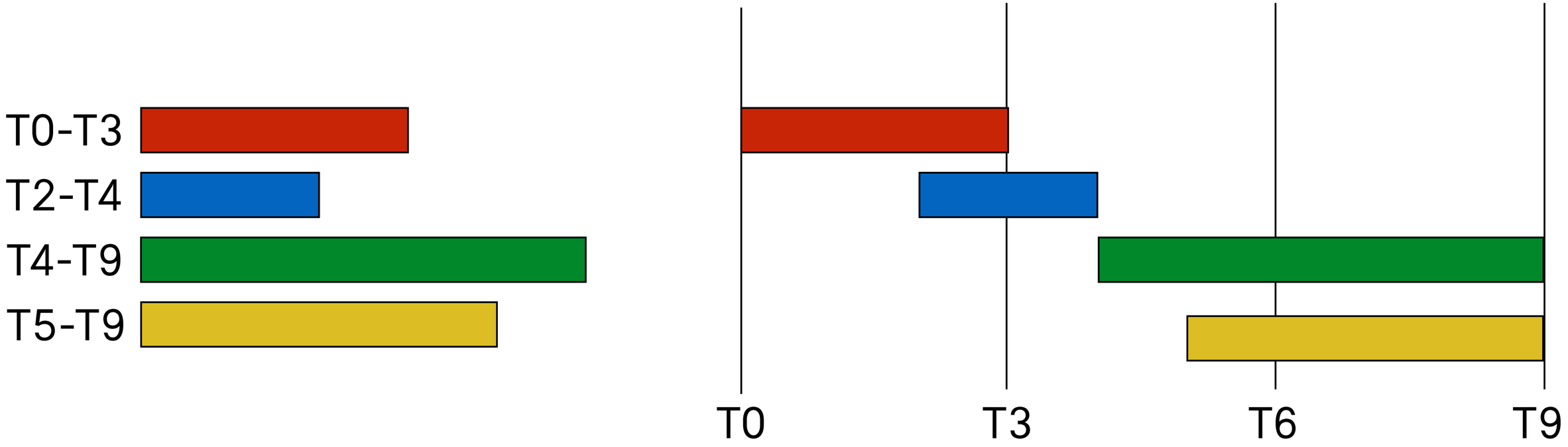
Ingester ••••

••• Store

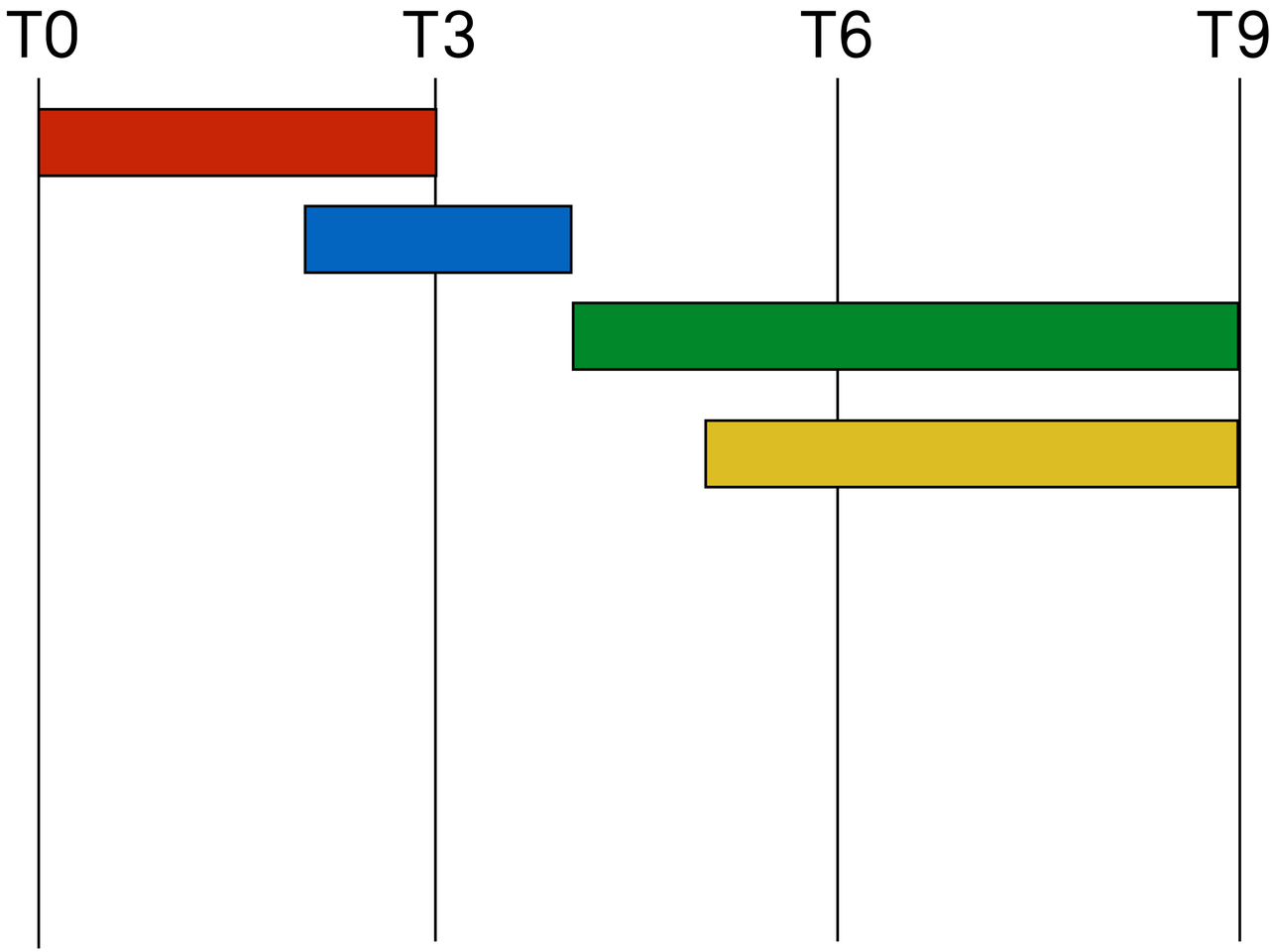
Compaction

So stupid, so good

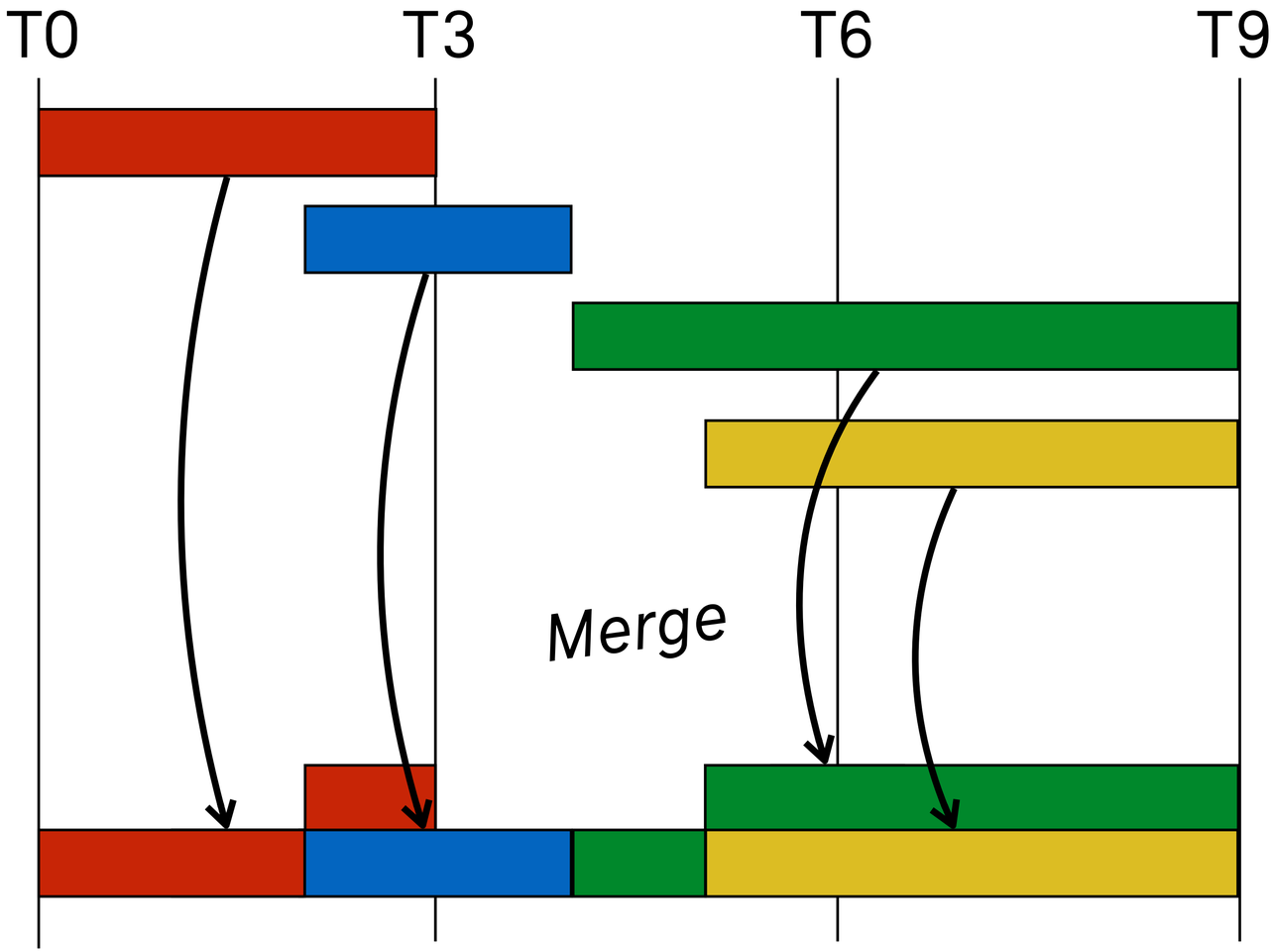
Segment files



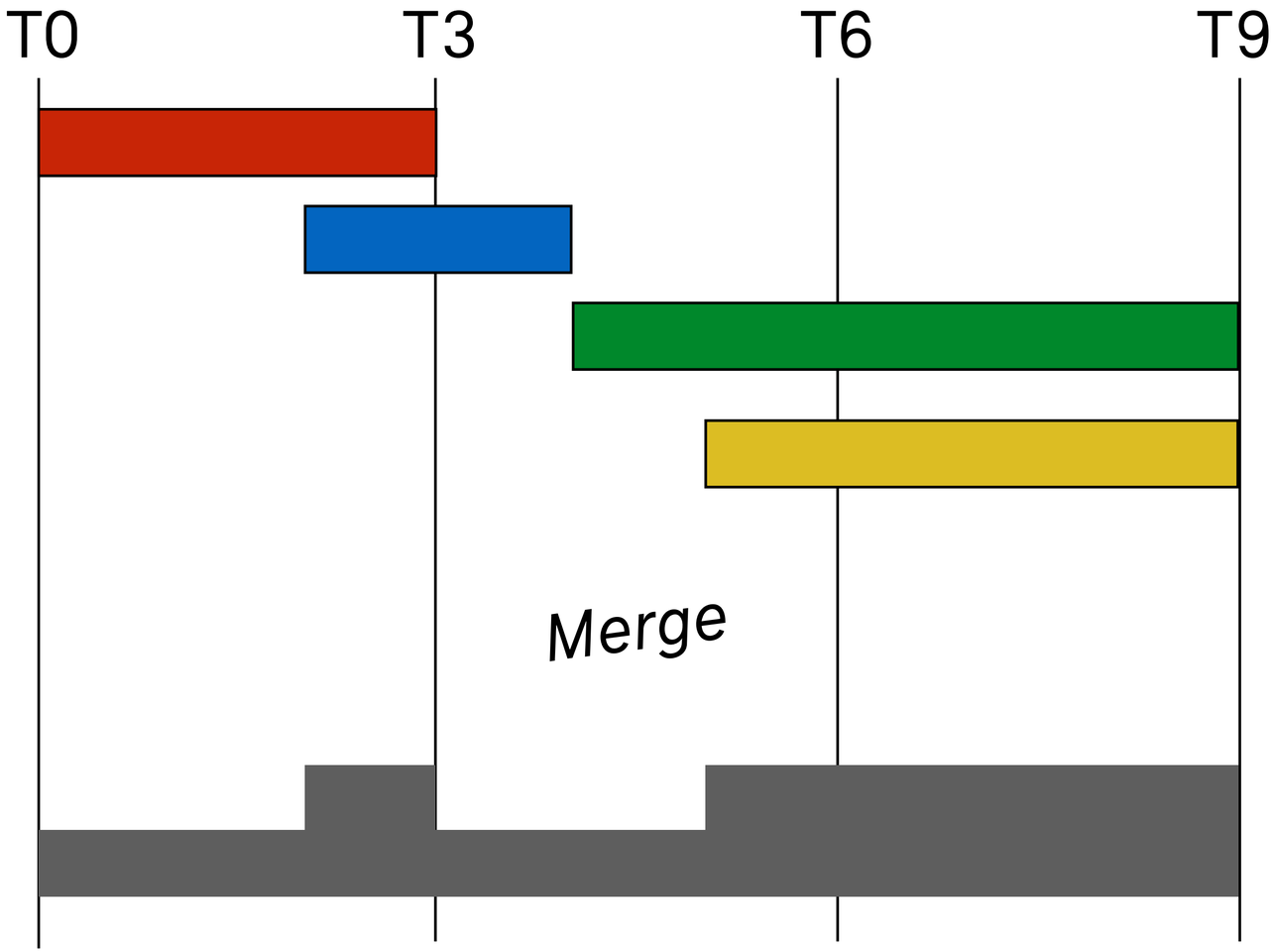
Segment files



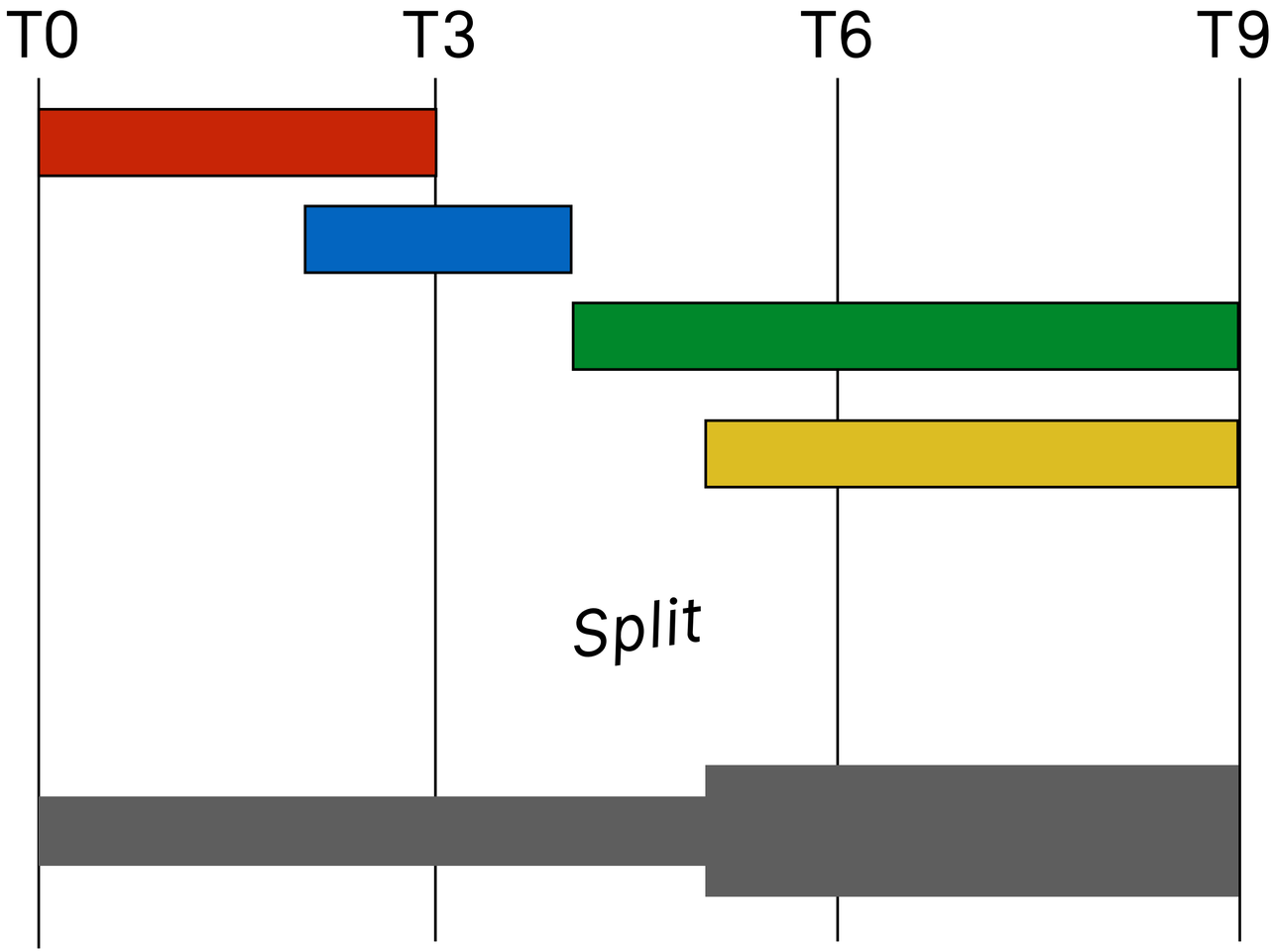
Segment files



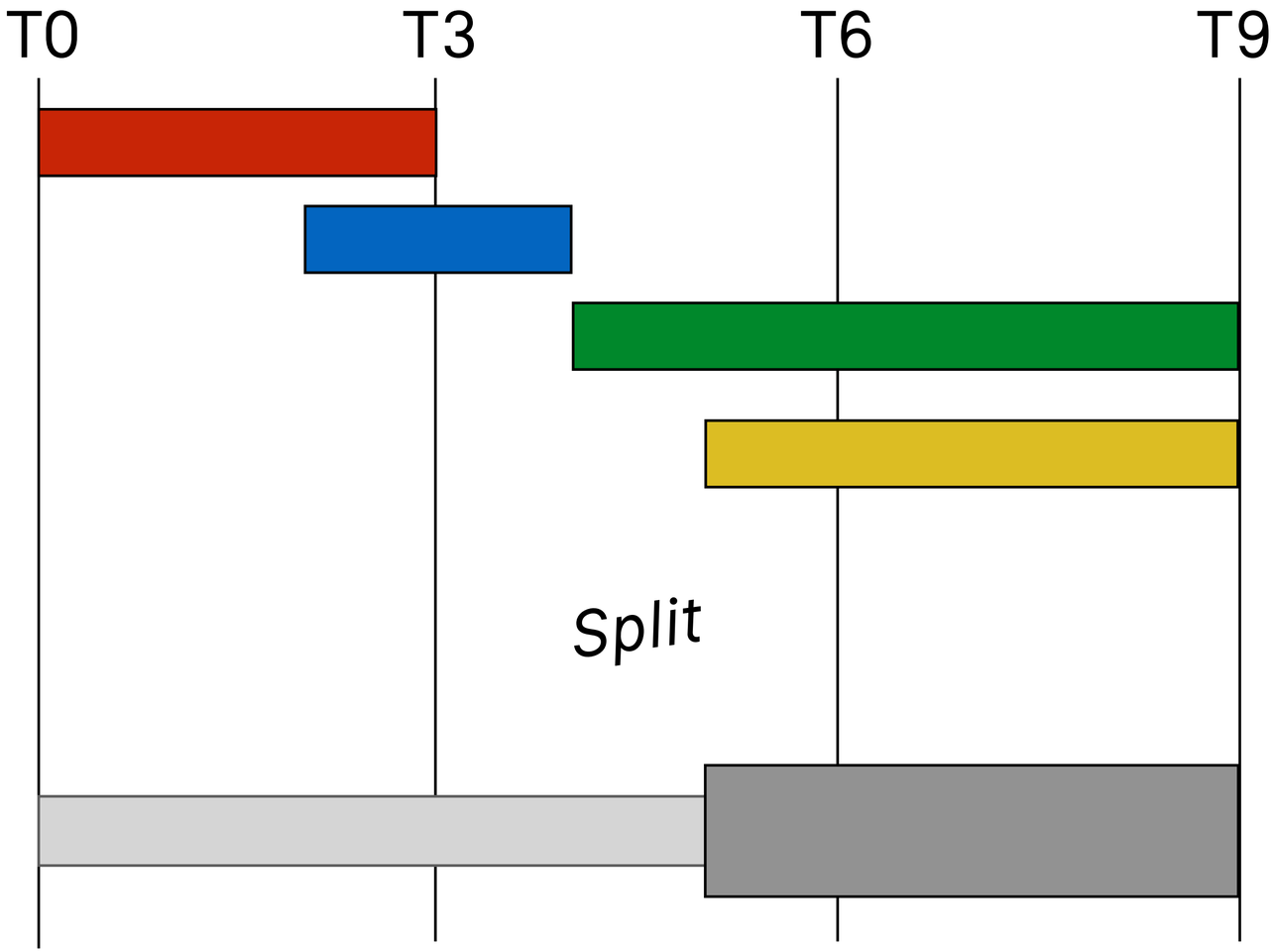
Segment files



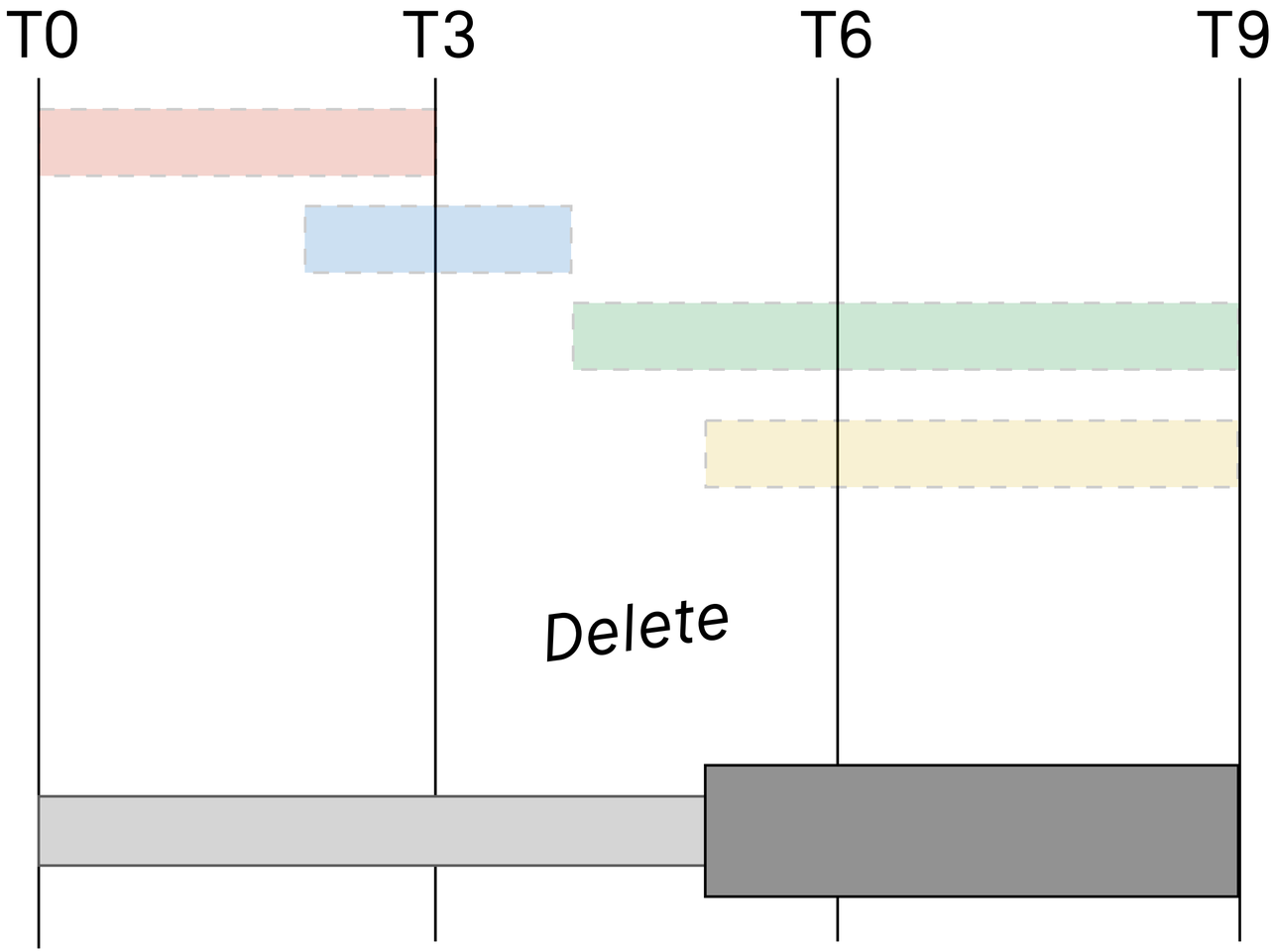
Segment files



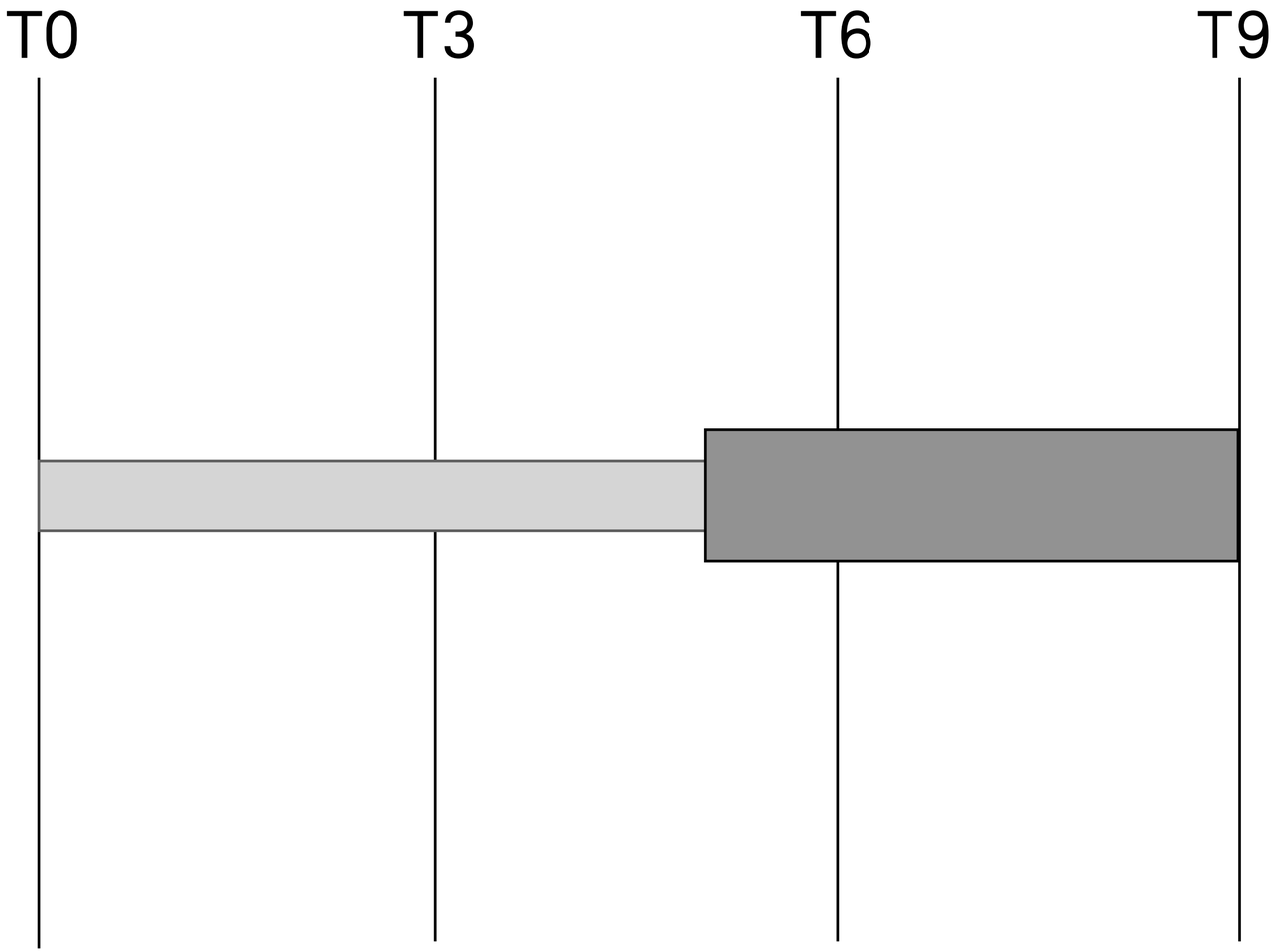
Segment files



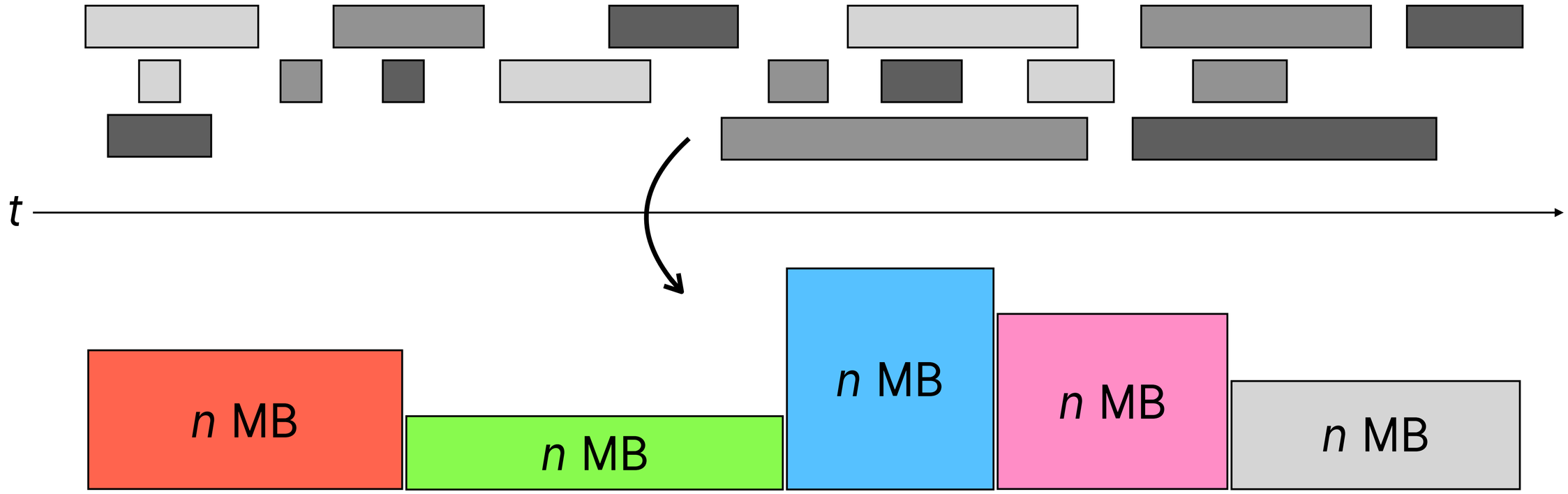
Segment files



Segment files



Goal of compaction



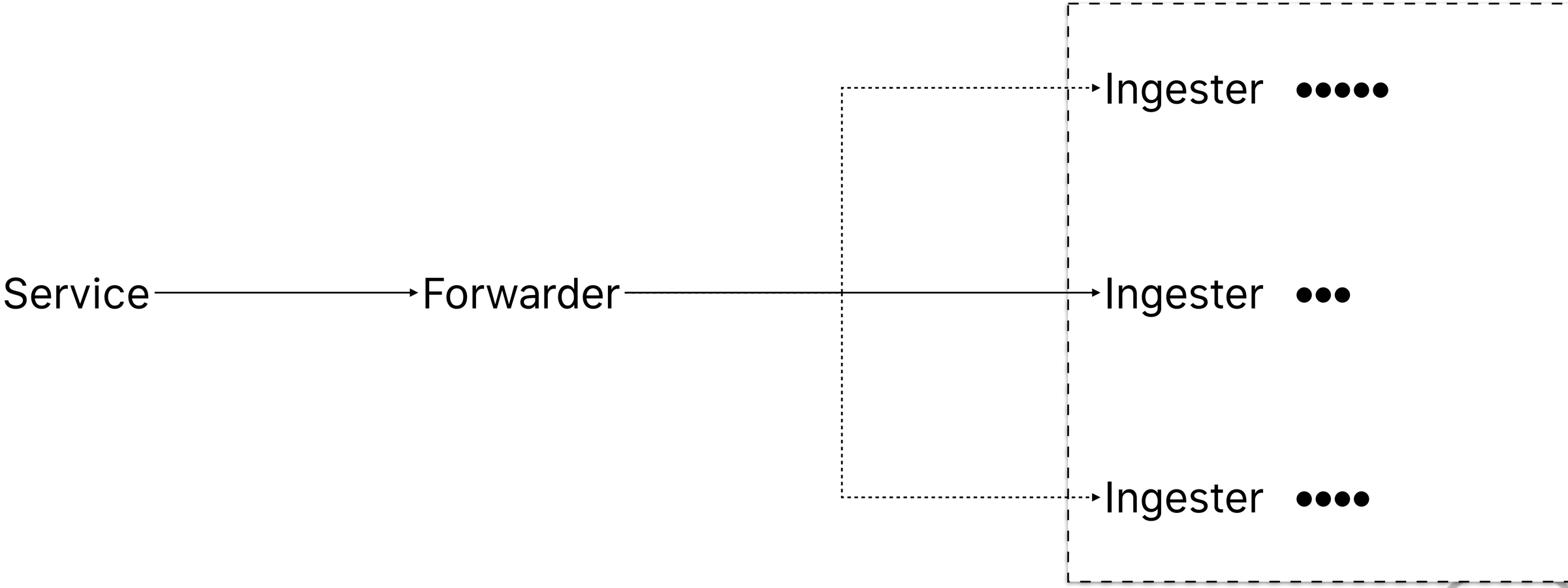
Adding capacity

Nothing fancy

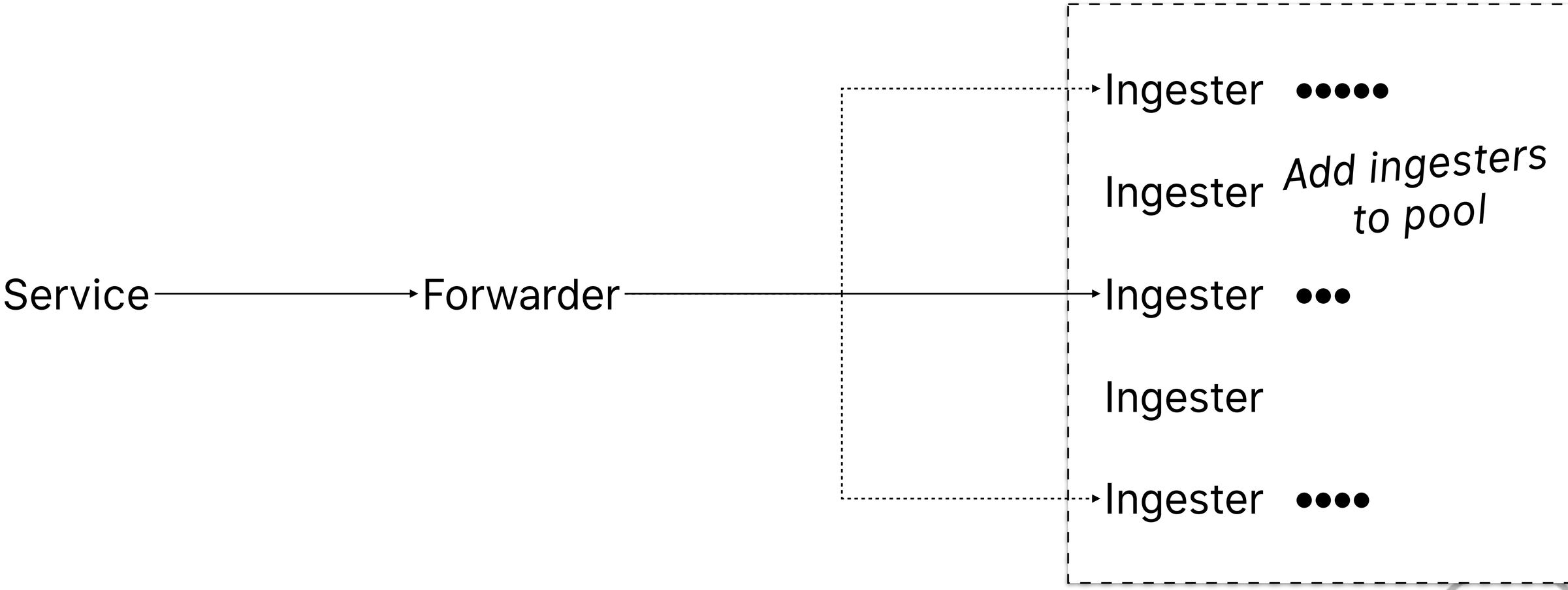
Adding ingest capacity

Literally just add ingesters

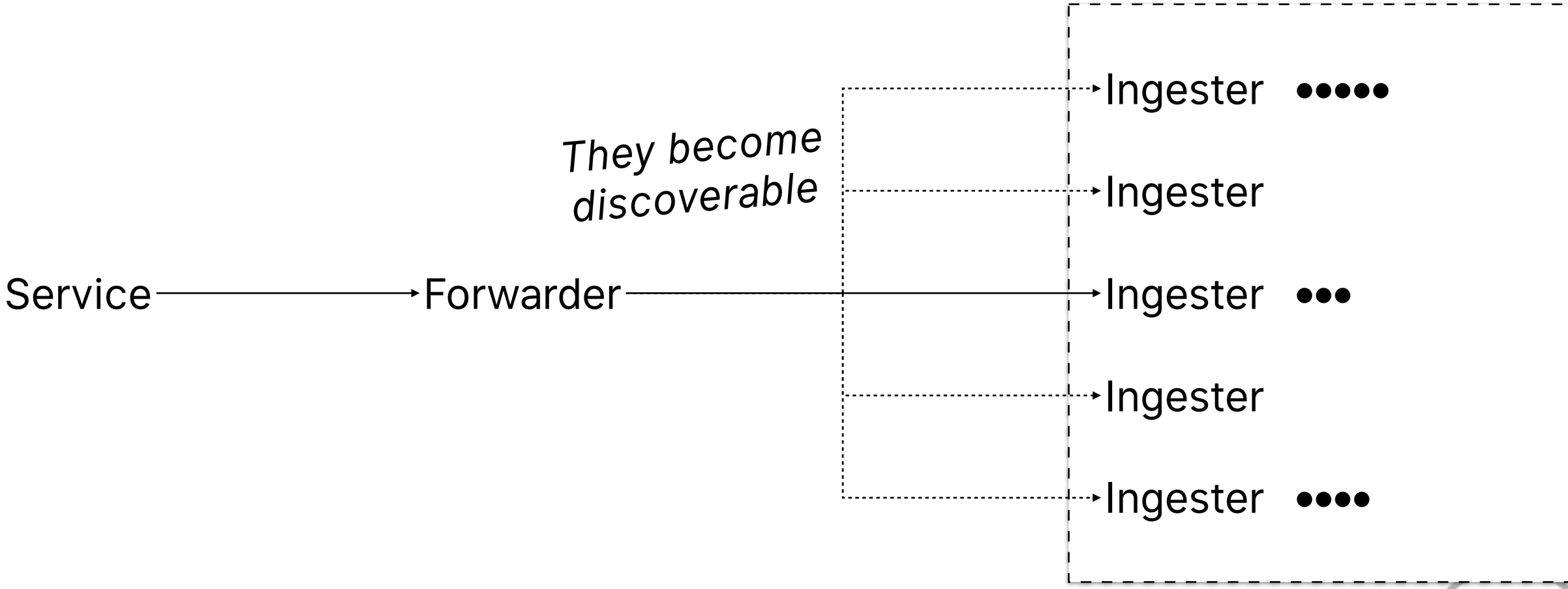
Adding ingest capacity



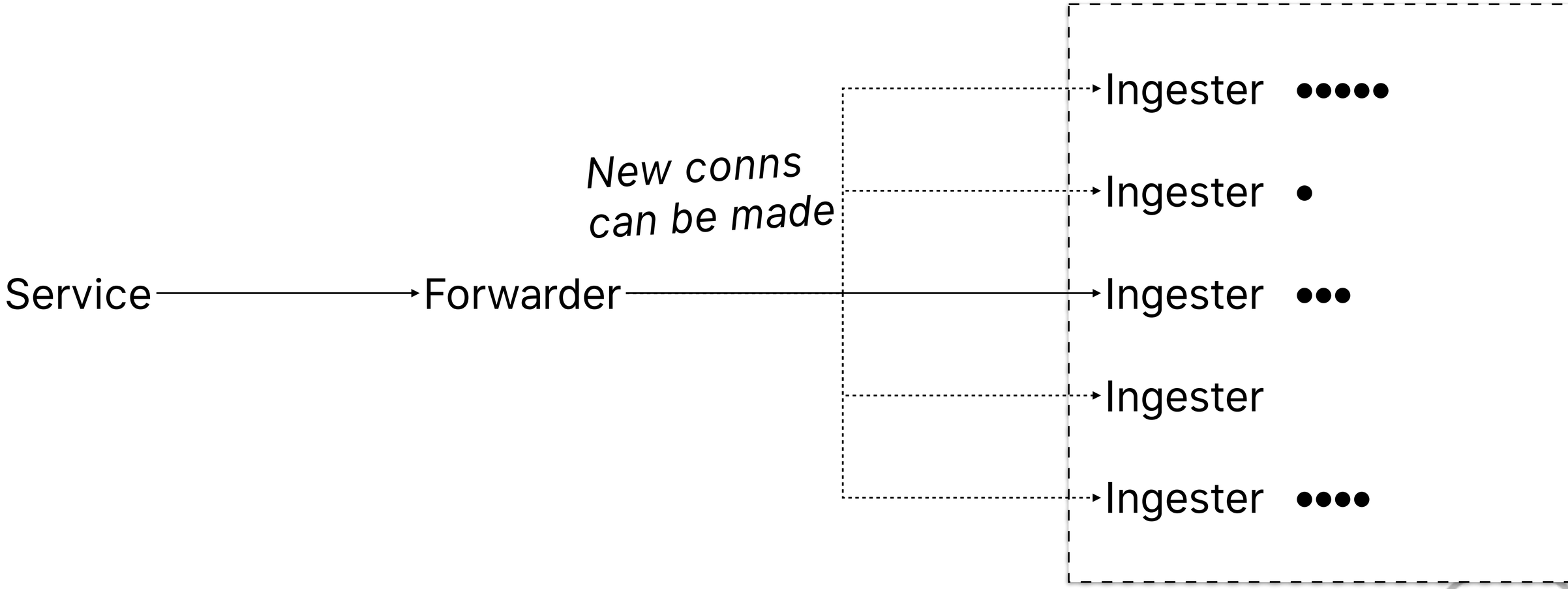
Adding ingest capacity



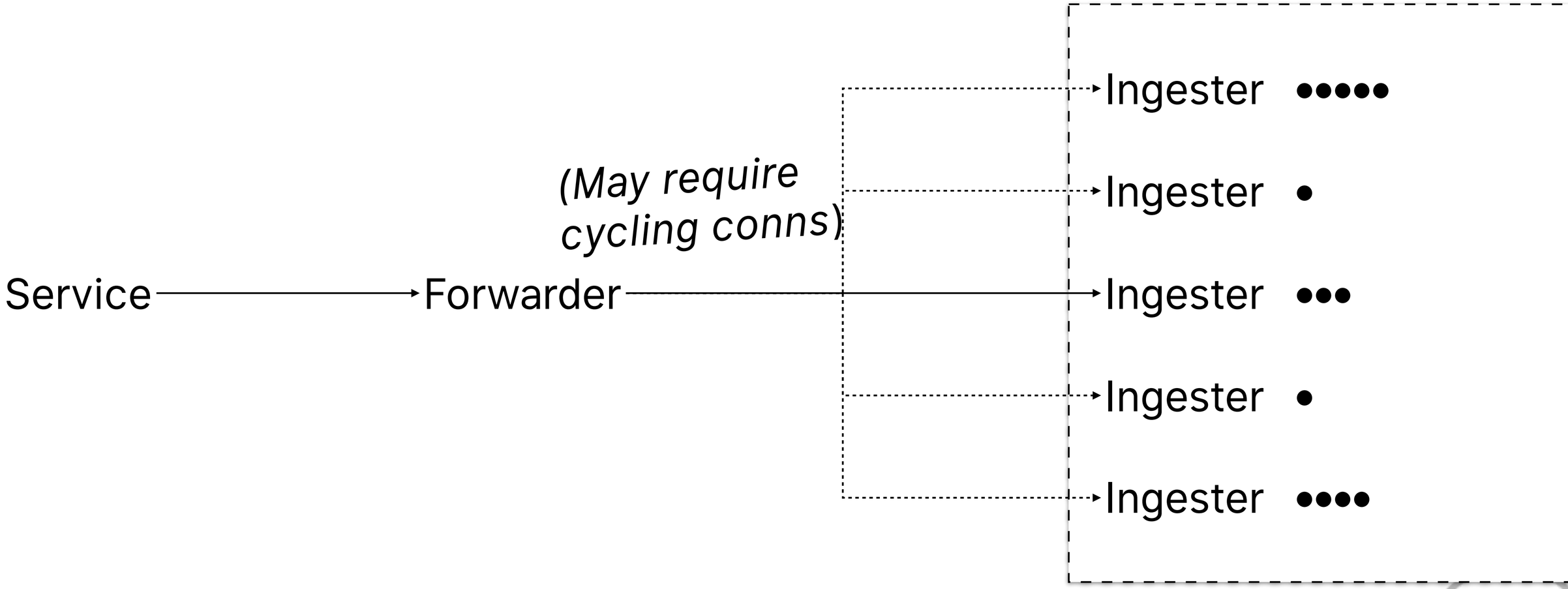
Adding ingest capacity



Adding ingest capacity



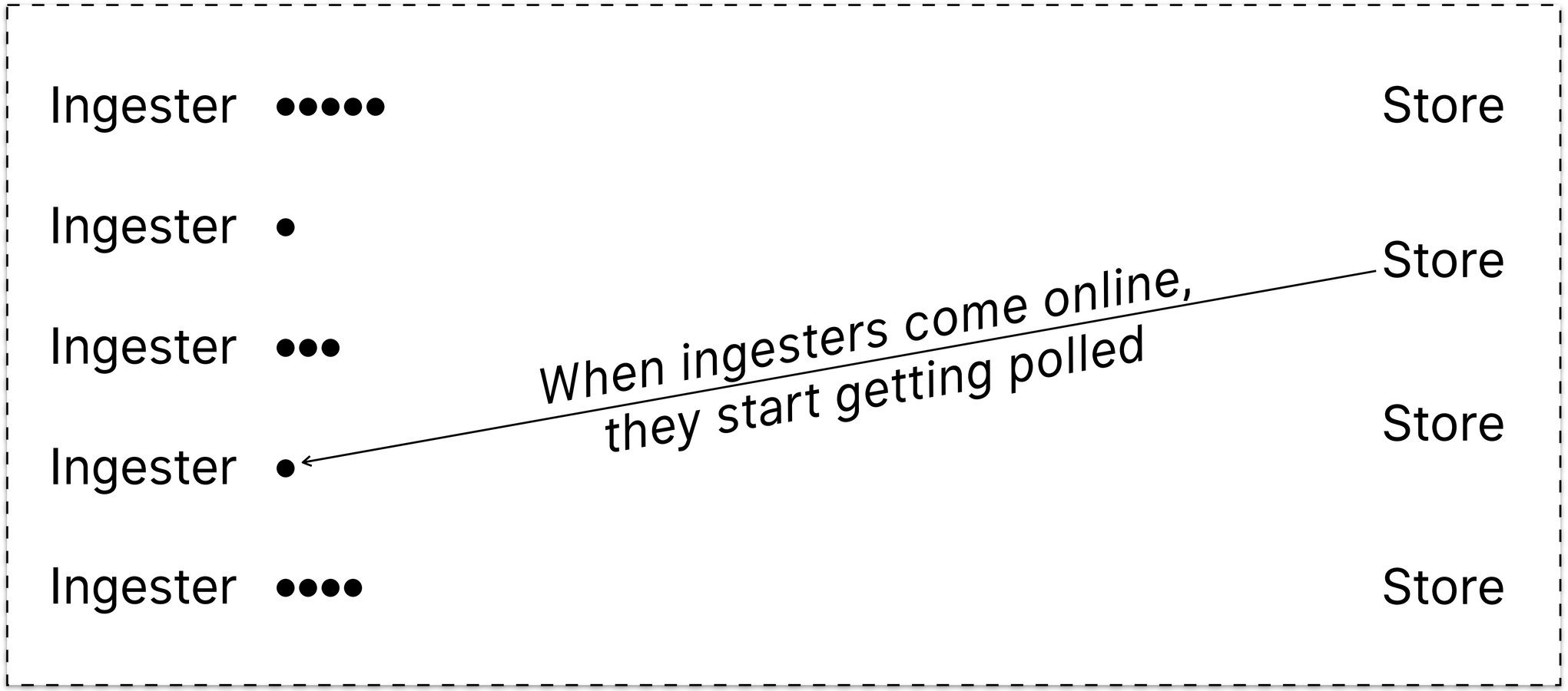
Adding ingest capacity



Adding ingest capacity

Adding ingest capacity

Adding ingest capacity



Adding query capacity

Add and wait

Adding query capacity

Ingester •••••

Ingester •

Ingester •••

Ingester •

Ingester ••••

•••• Store

••••••• Store

••••• Store

•••••• Store

Adding query capacity

Ingester •••••

Ingester •

Ingester •••

Ingester •

Ingester ••••

New stores start empty

•••• Store

Store

••••••• Store

Store

••••• Store

Store

•••••• Store

Adding query capacity

Ingester •••••

Ingester •

Ingester •••

Ingester •

Ingester ••••

And start getting equal shares of segments

•••• Store

• Store

••••••• Store

• Store

••••• Store

• Store

•••••• Store

Adding query capacity

Ingester •••••

Ingester •

Ingester •••

Ingester •

Ingester ••••

(So query load is unbalanced)

••••••• Store

•• Store

••••••••• Store

•• Store

••••••• Store

•• Store

••••••• Store

Adding query capacity

Ingester •••••

Ingester •

Ingester •••

Ingester •

Ingester ••••

Over time,
shares balance...

●●●●●●● Store

●●● Store

●●●●●●●● Store

●●● Store

●●●●●●●● Store

●●● Store

●●●●●●●● Store

Adding query capacity

Ingester •••••

Ingester •

Ingester •••

Ingester •

Ingester ••••

Over time,
shares balance...

●●●●●●●● Store

●●●● Store

●●●●●●●● Store

●●●● Store

●●●●●●●● Store

●●●● Store

●●●●●●●● Store

Adding query capacity

Ingester •••••

Ingester •

Ingester •••

Ingester •

Ingester ••••

Until we reach the retention window...

••••••• Store

••••• Store

••••••• Store

••••• Store

••••••• Store

••••• Store

••••••• Store

Adding query capacity

Ingester •••••

Ingester •

Ingester •••

Ingester •

Ingester ••••

Until we reach the retention window...

●●●●●●●● Store

●●●●●● Store

●●●●●●●● Store

●●●●●● Store

●●●●●●●● Store

●●●●●● Store

●●●●●●●● Store

Adding query capacity

Ingester •••••

Ingester •

Ingester •••

Ingester •

Ingester ••••

At which point...

●●●●●●● Store

●●●●●●● Store

●●●●●●● Store

●●●●●●● Store

●●●●●●● Store

●●●●●●● Store

●●●●●●● Store

Adding query capacity

Ingester •••••

Ingester •

Ingester •••

Ingester •

Ingester ••••

We're co-equal again

Store

Store

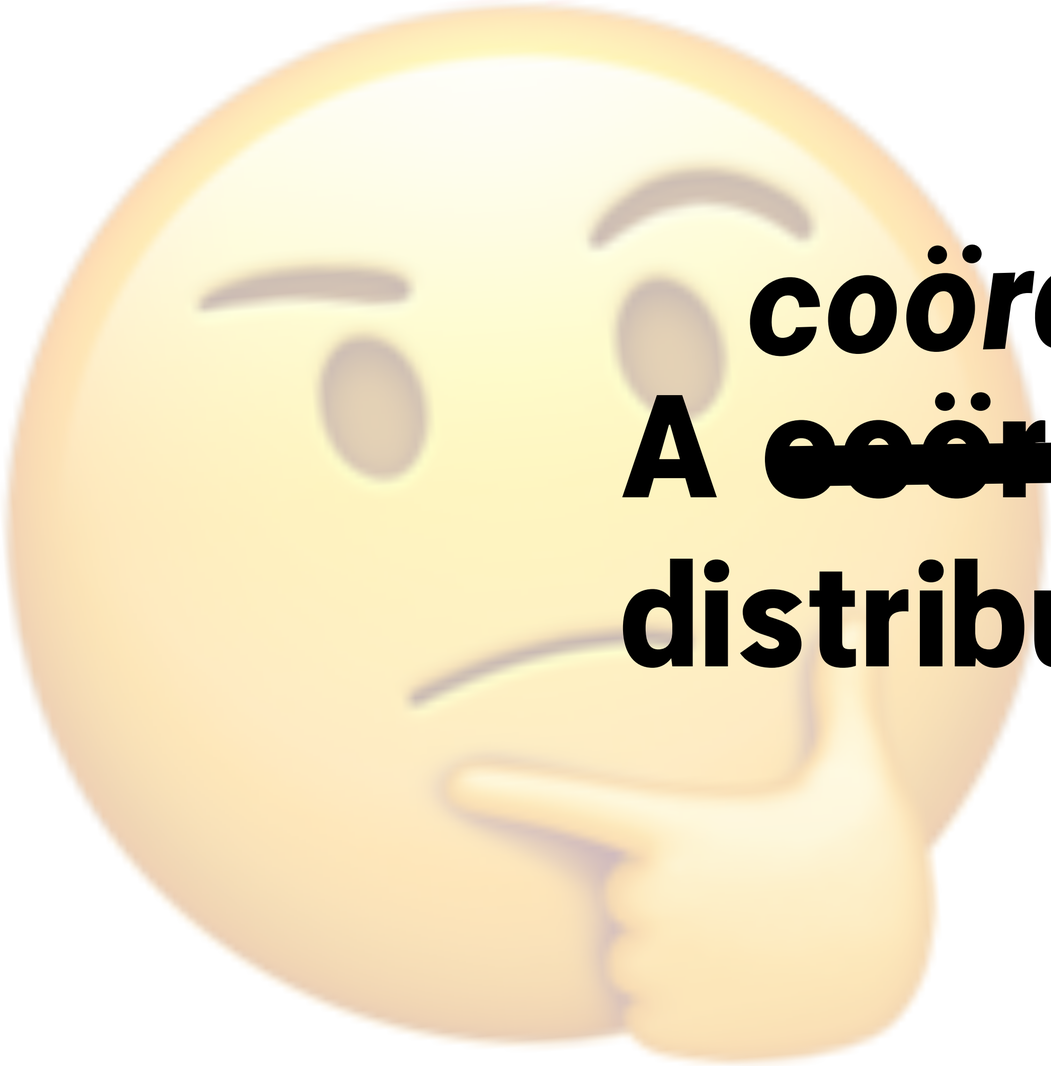
Store

Store

Store

Store

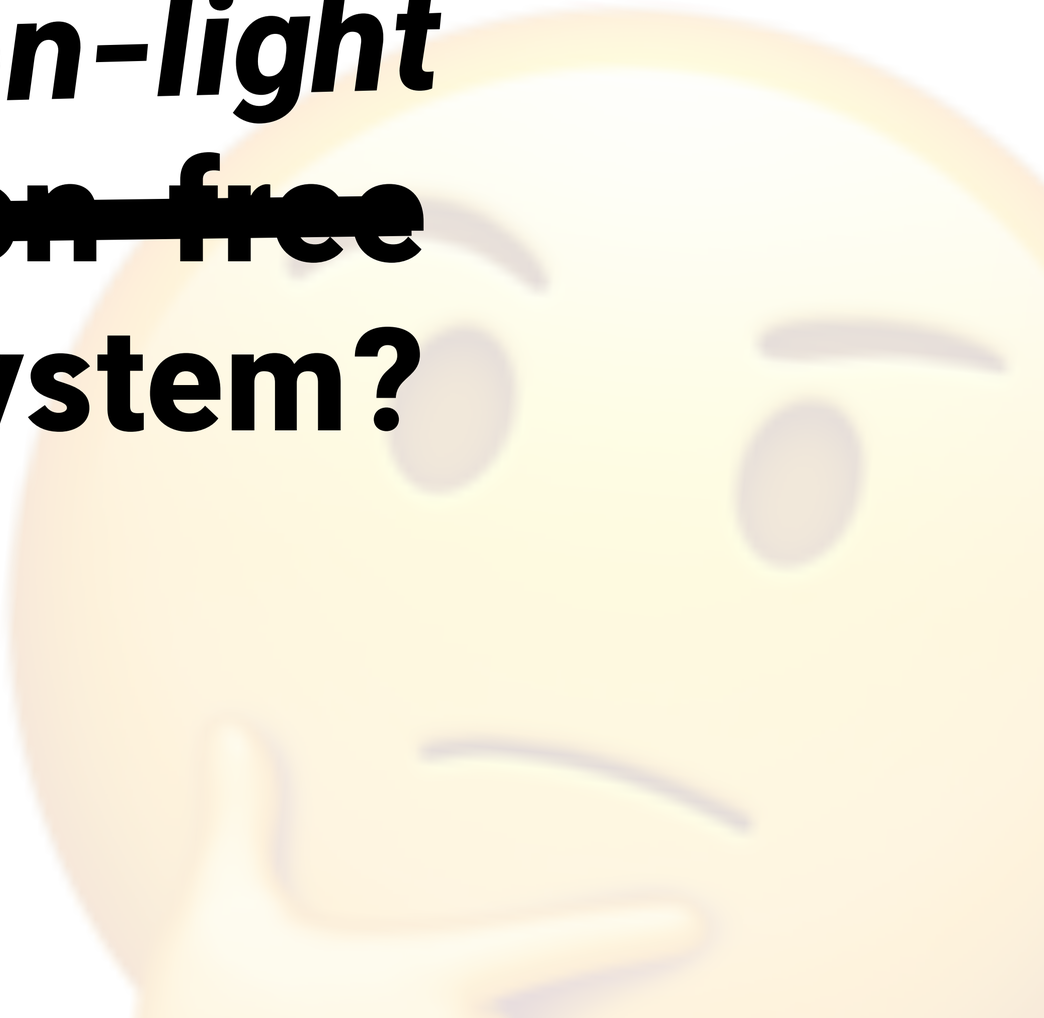
Store



coördination-light

A ~~*coördination-free*~~

distributed system?



Conclusions

Solve smaller problems

It's OK to write code

Truth is unknowable

I crave social validation
Follow me on Twitter
@peterbourgon

OK Log 

Distributed and coördination-*light* logging