Goka

Painless stream processing with Go and Kafka

Franz Eichhorn Diogo Behrens

2019-01-29

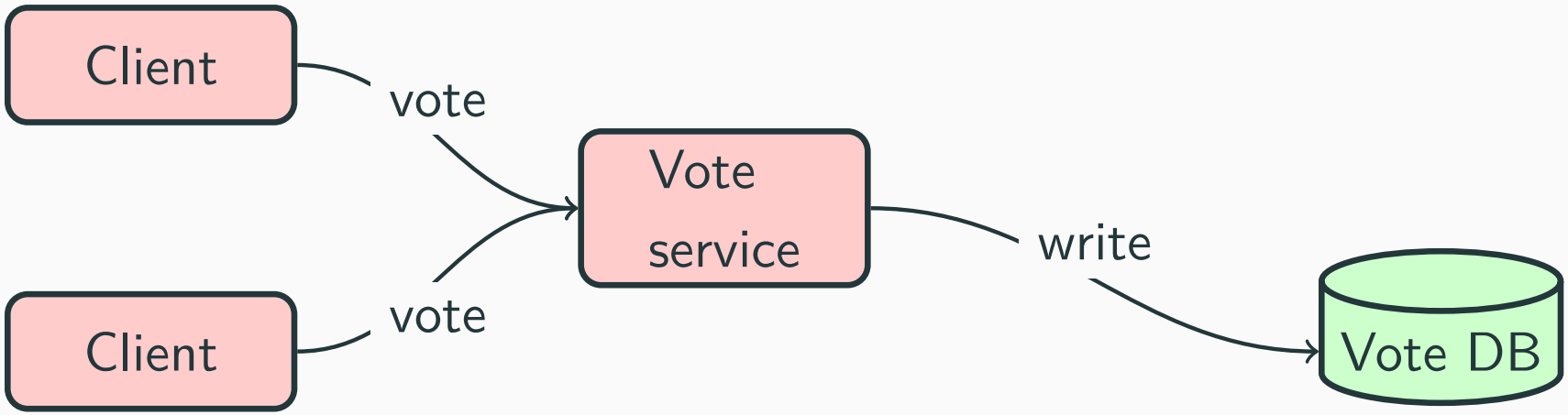
Agenda

- Streaming microservices
  → problems and approach
- Mini Kafka review
  → basic concepts and hands-on warmup
- Learning Goka the “hard way”
  → components introduction with hands-on assignments
- Closing discussion
  → best practices, monitoring, testing, links,. . .

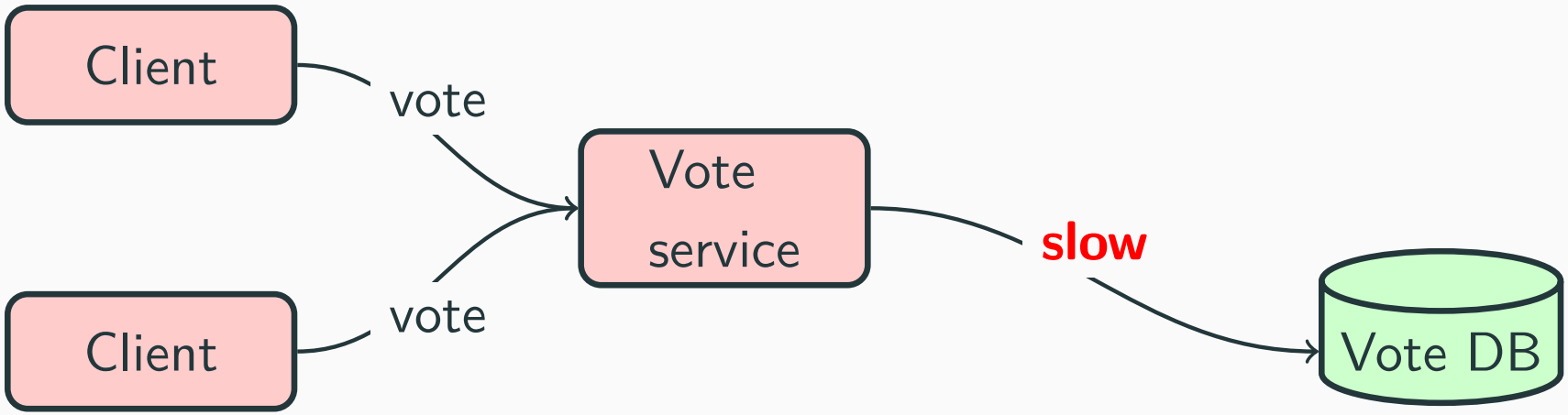
Streaming microservices

queues, DBs, logs, caches,...

Writing data: response time and complexity issues

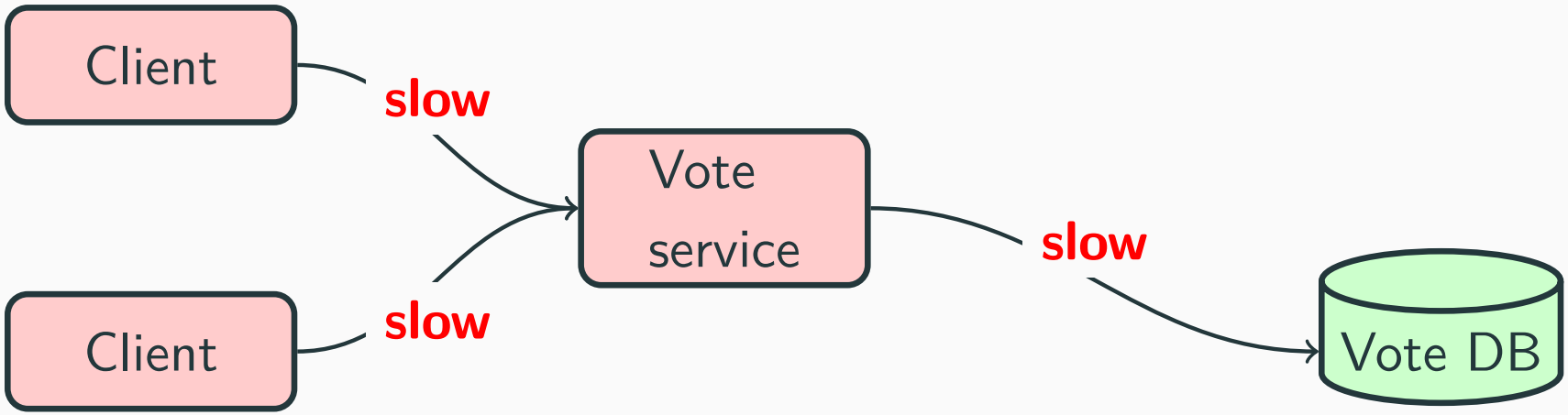


Writing data: response time and complexity issues

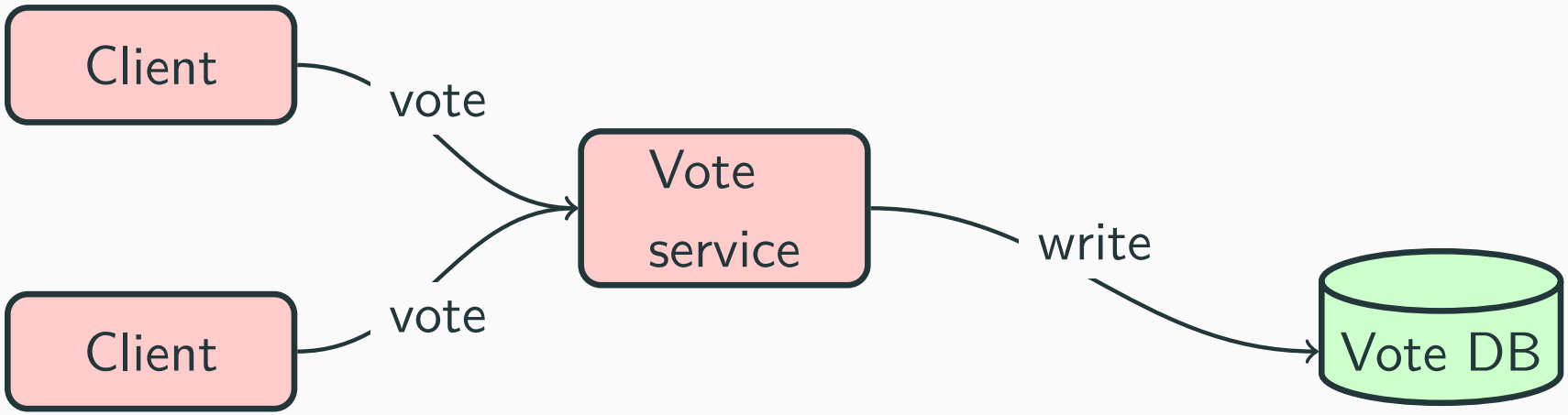


Writing data: response time and complexity issues

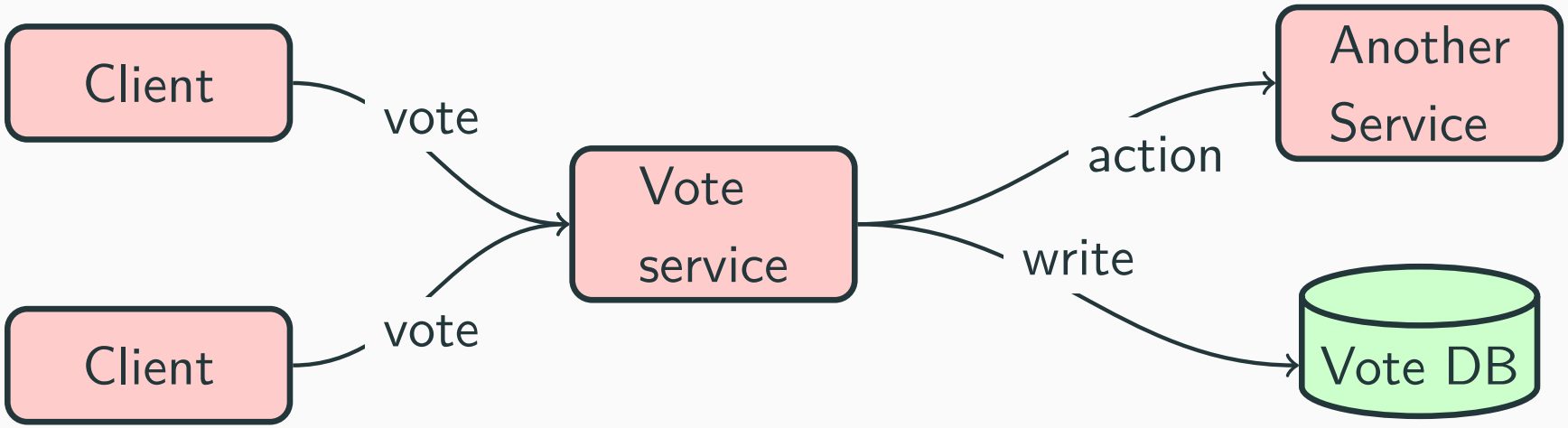
Problem 1:
if DB slow, client slow



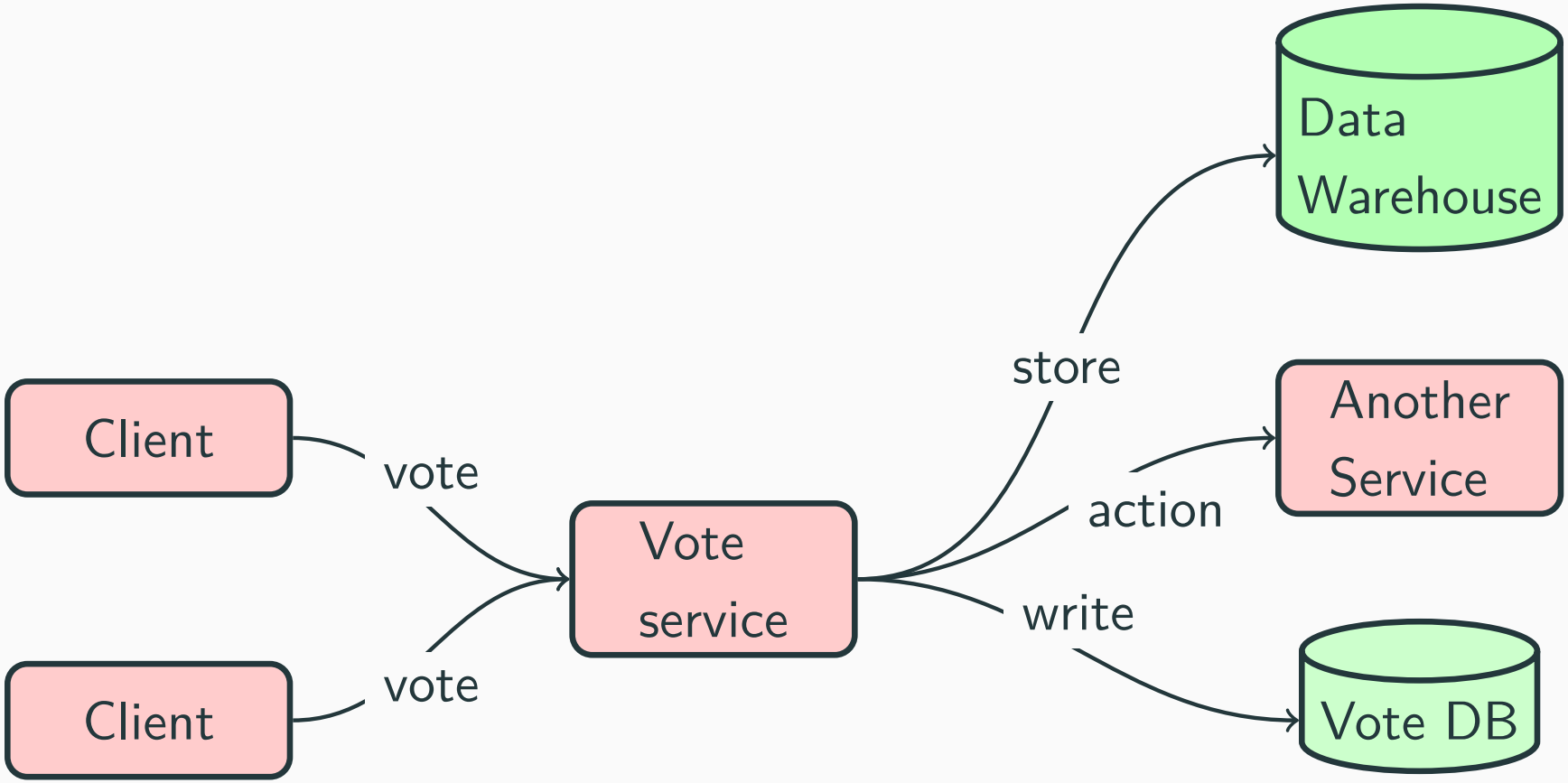
Writing data: response time and complexity issues



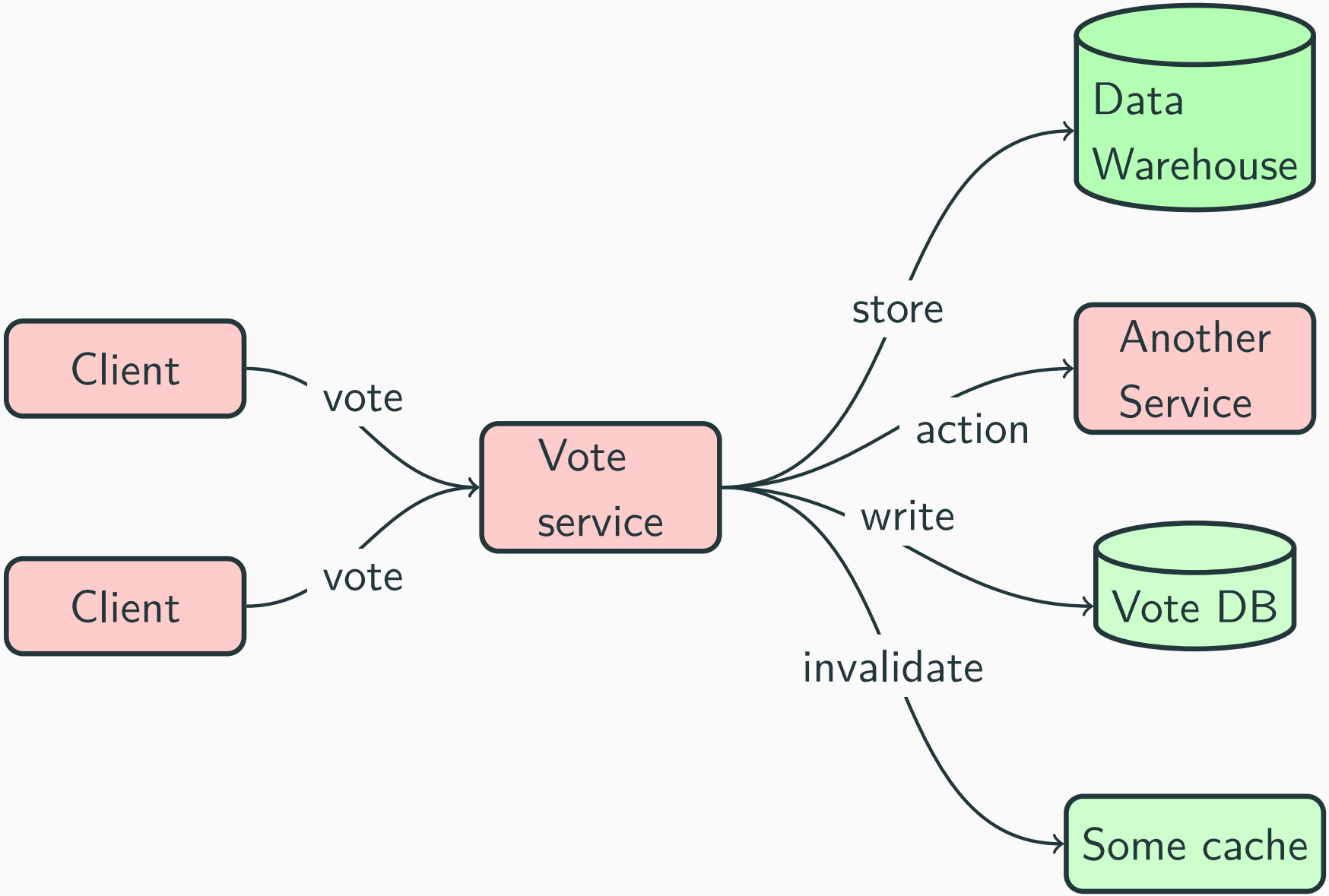
Writing data: response time and complexity issues



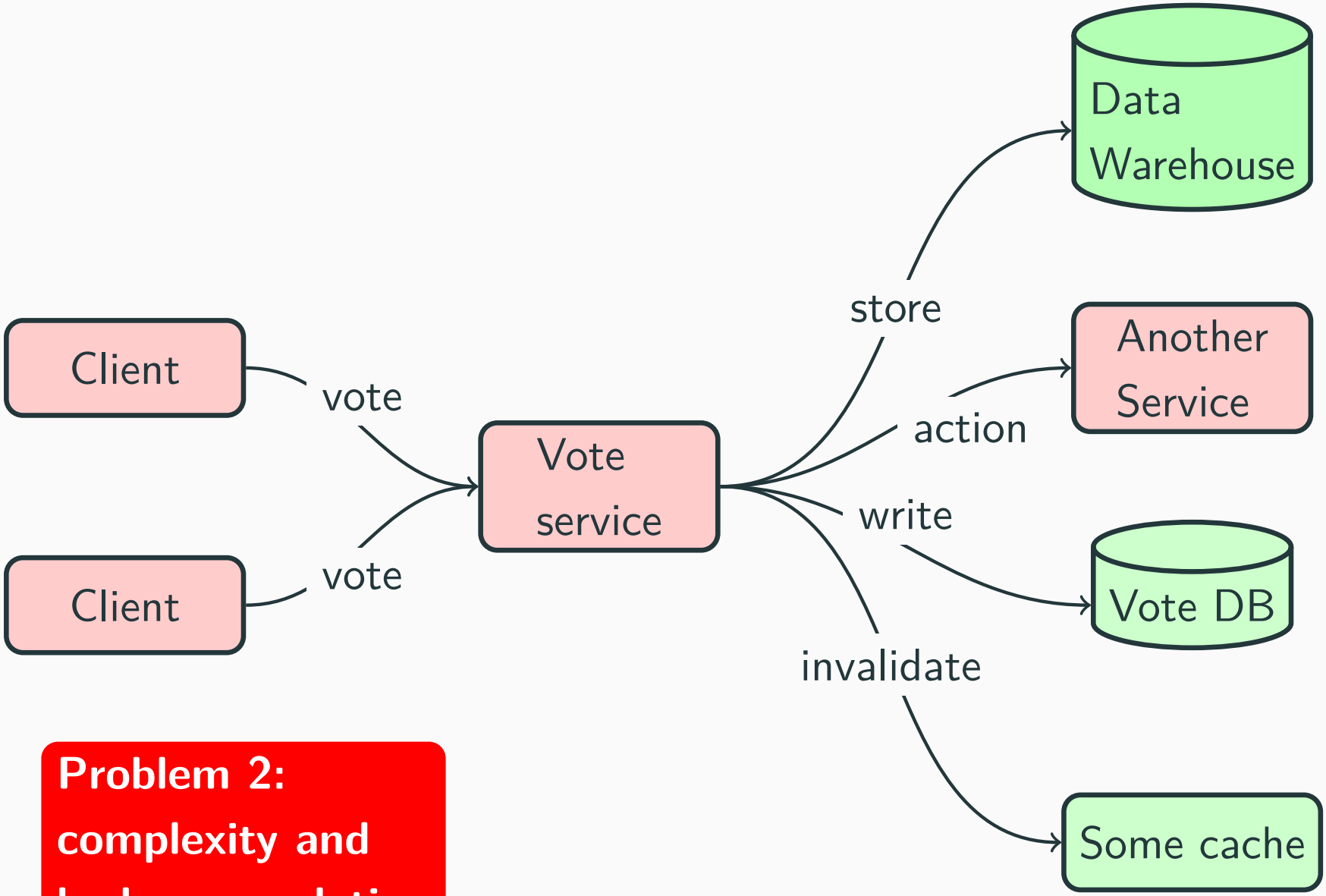
Writing data: response time and complexity issues



Writing data: response time and complexity issues

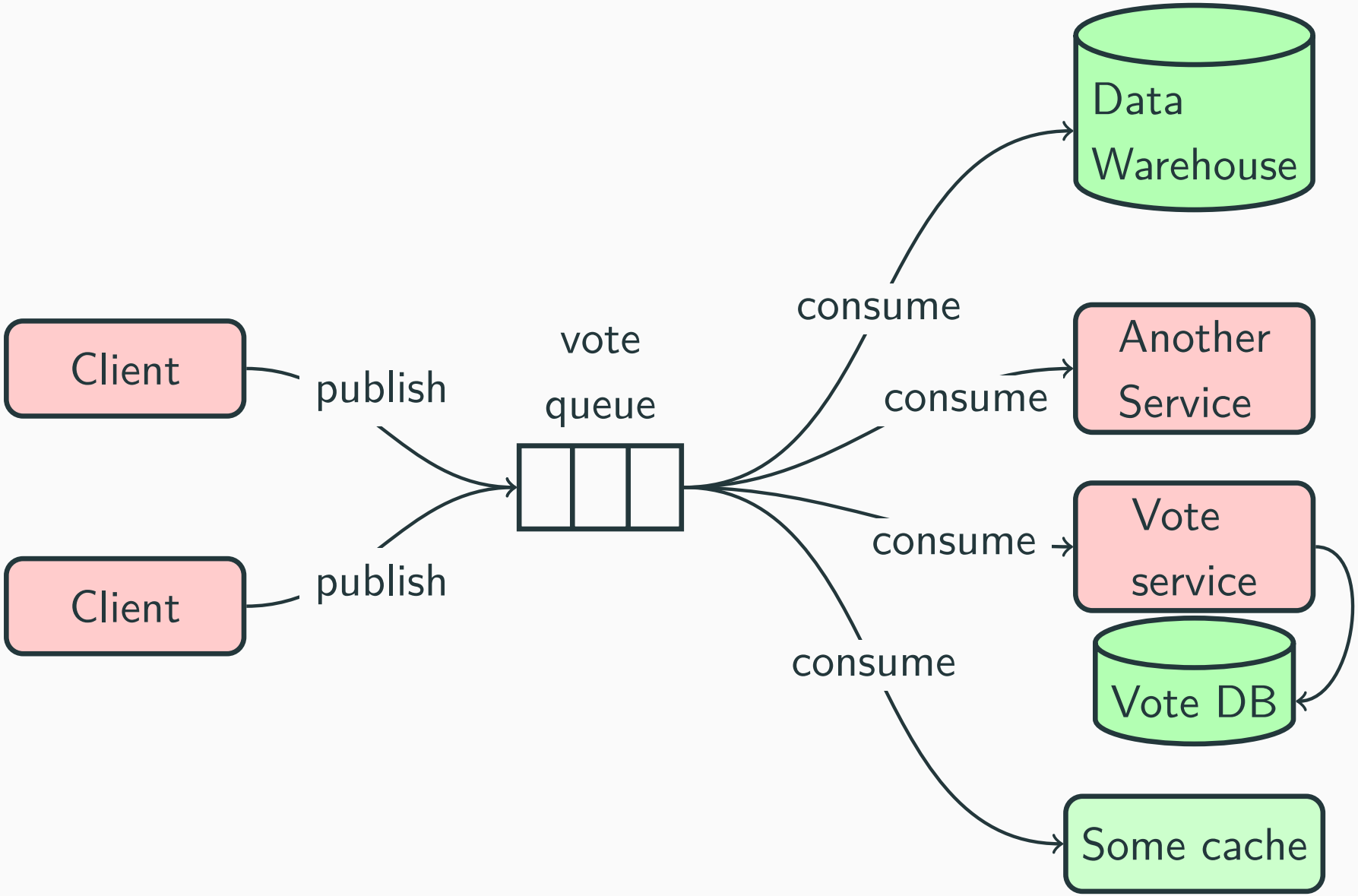


Writing data: response time and complexity issues



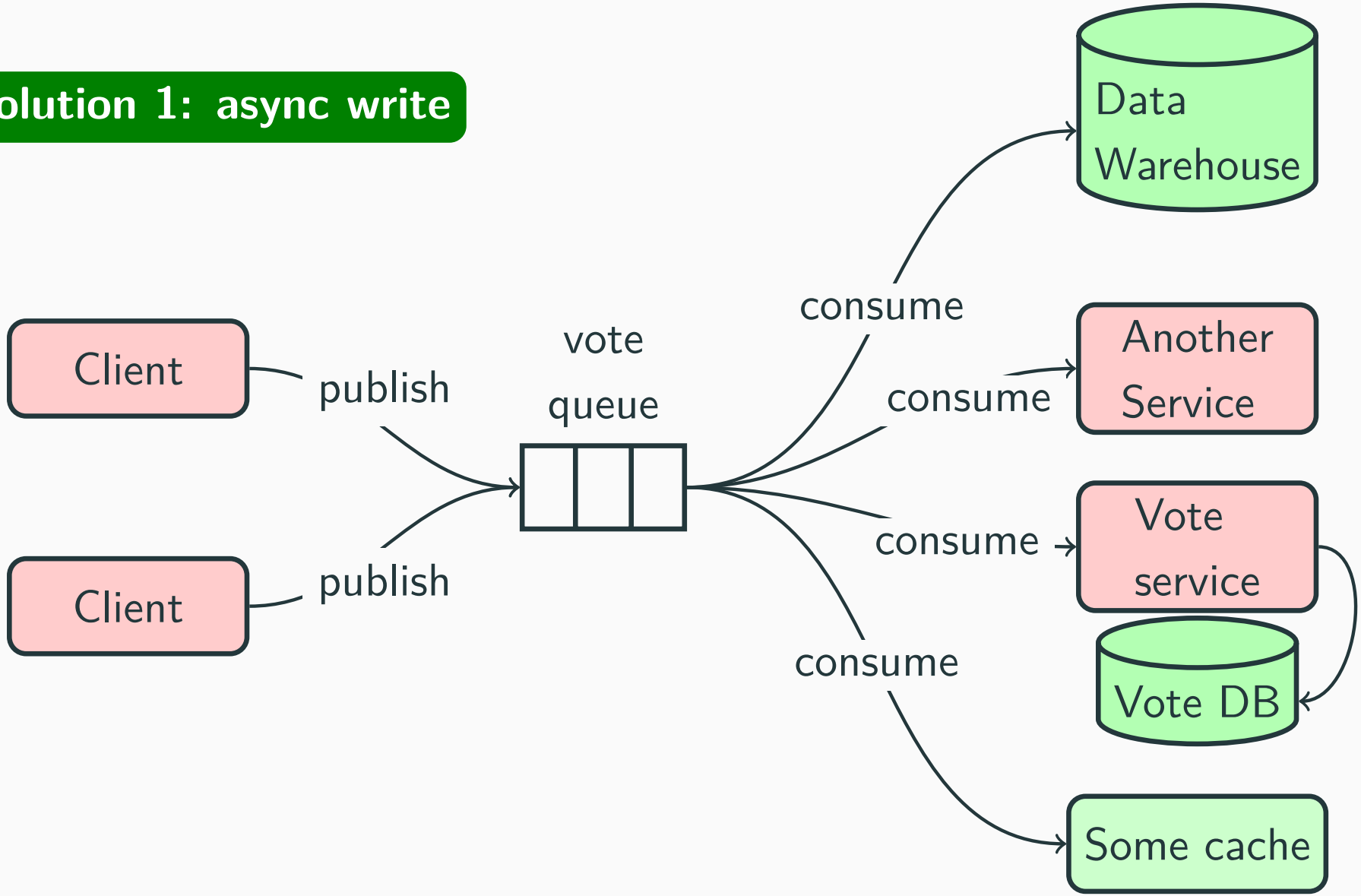
Problem 2: complexity and bad encapsulation

Writing data: decouple clients and services in space and time



Writing data: decouple clients and services in space and time

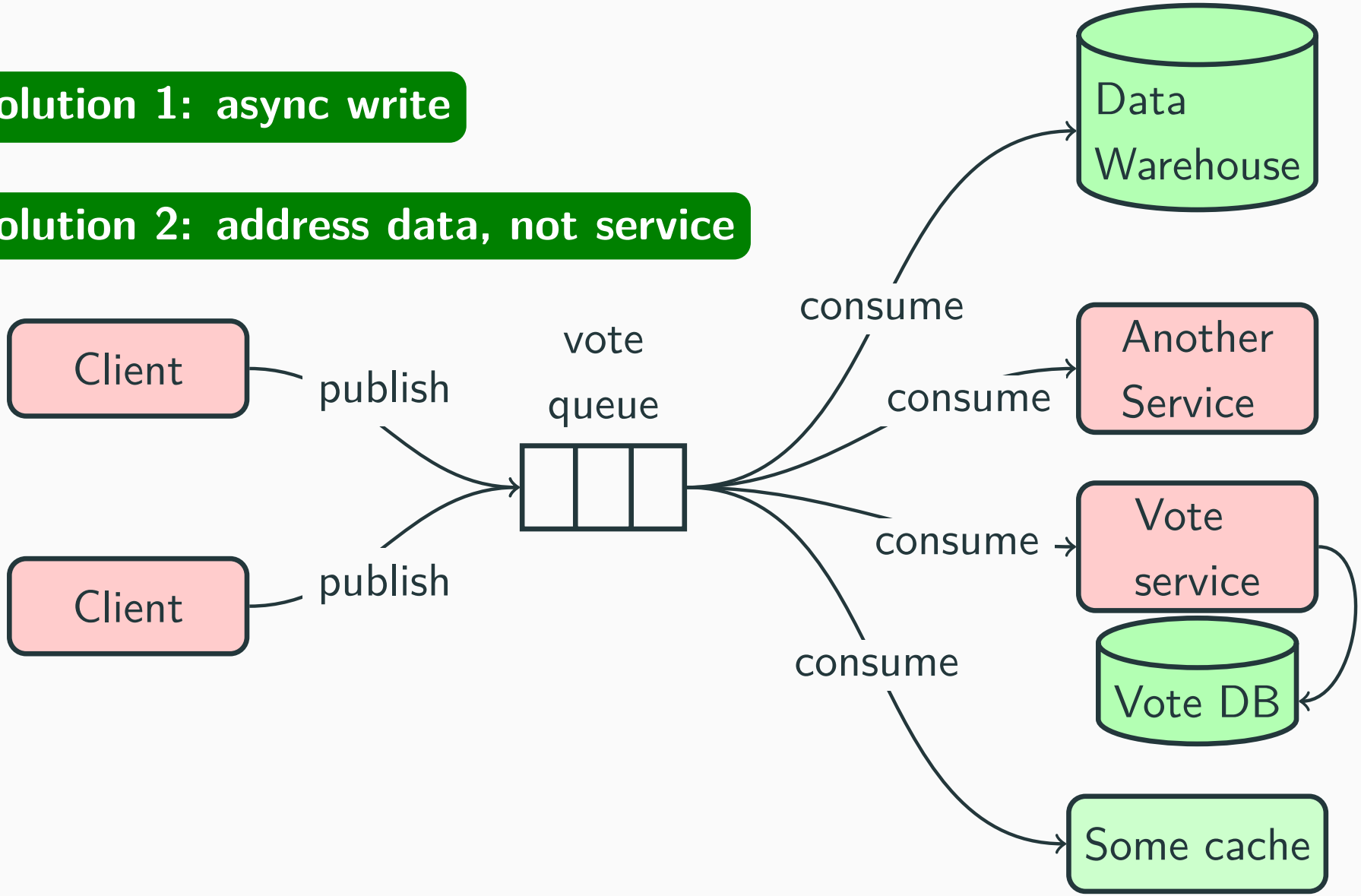
Solution 1: async write



Writing data: decouple clients and services in space and time

Solution 1: async write

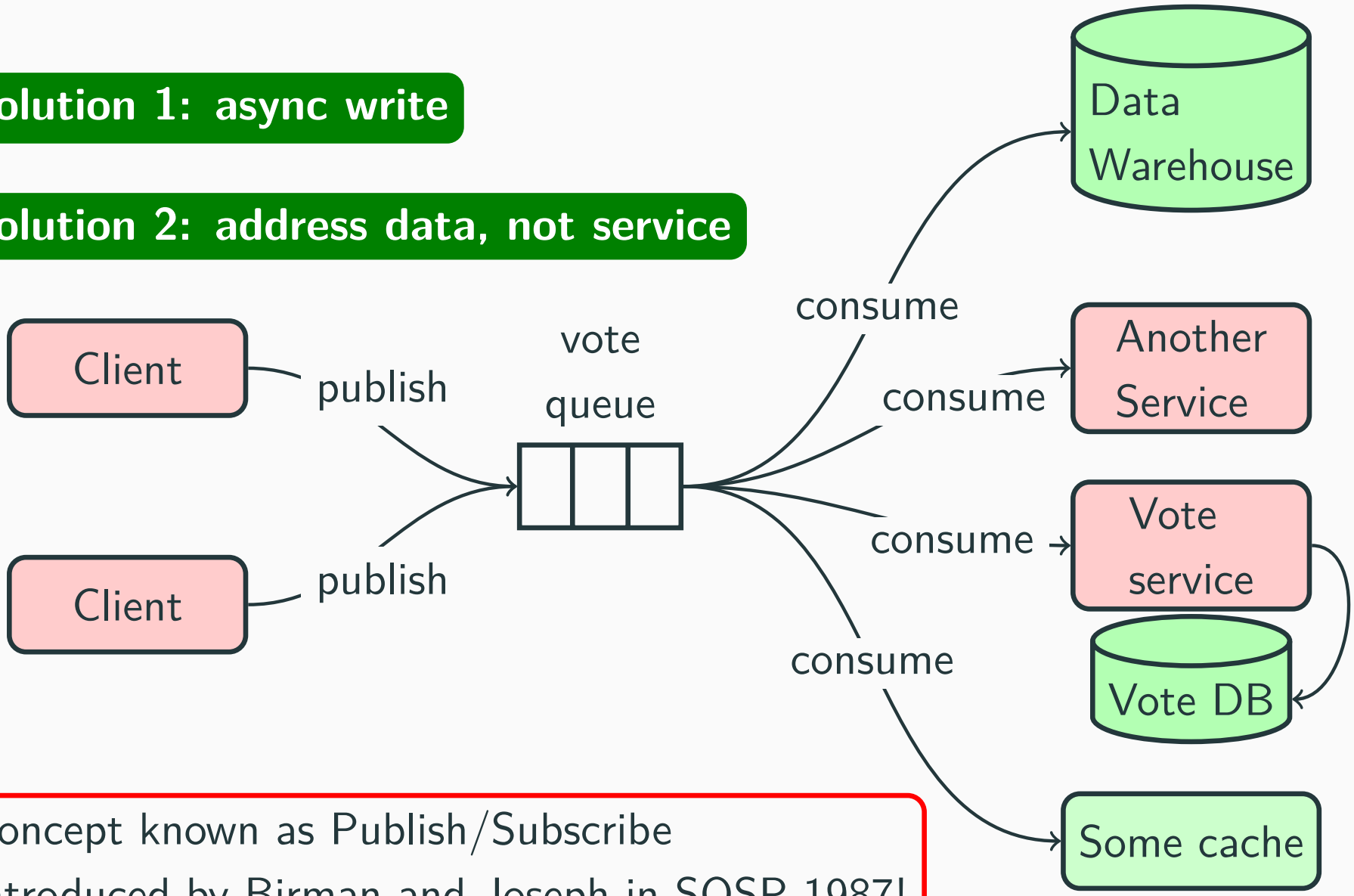
Solution 2: address data, not service



Writing data: decouple clients and services in space and time

Solution 1: async write

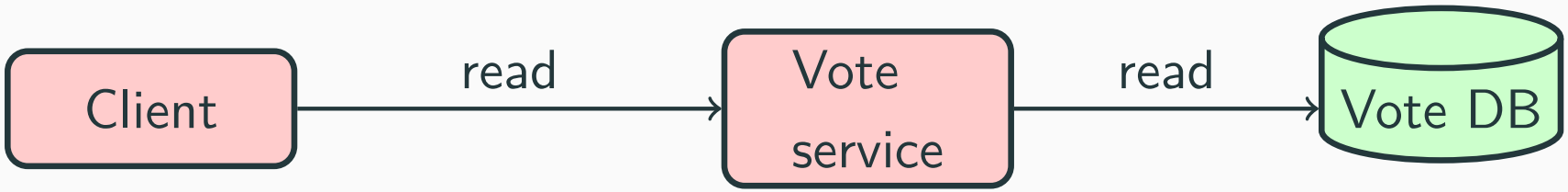
Solution 2: address data, not service



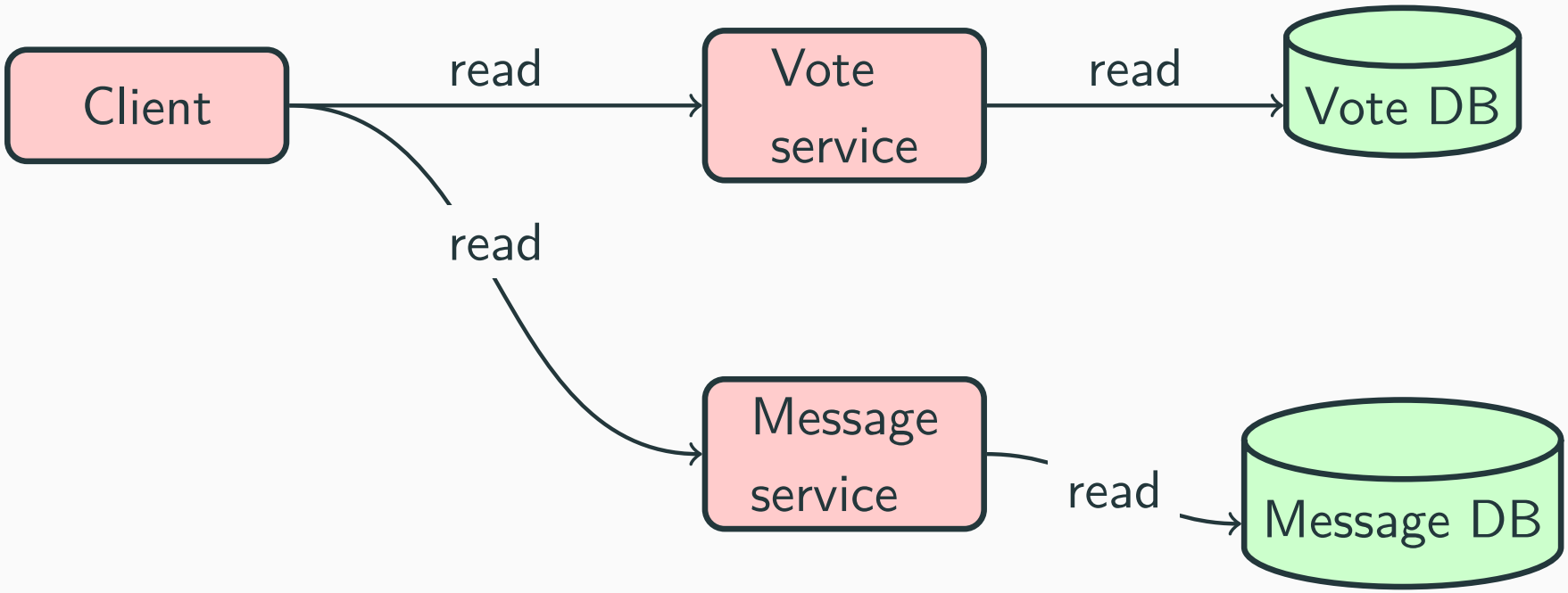
Concept known as Publish/Subscribe

Introduced by Birman and Joseph in SOSP 1987!

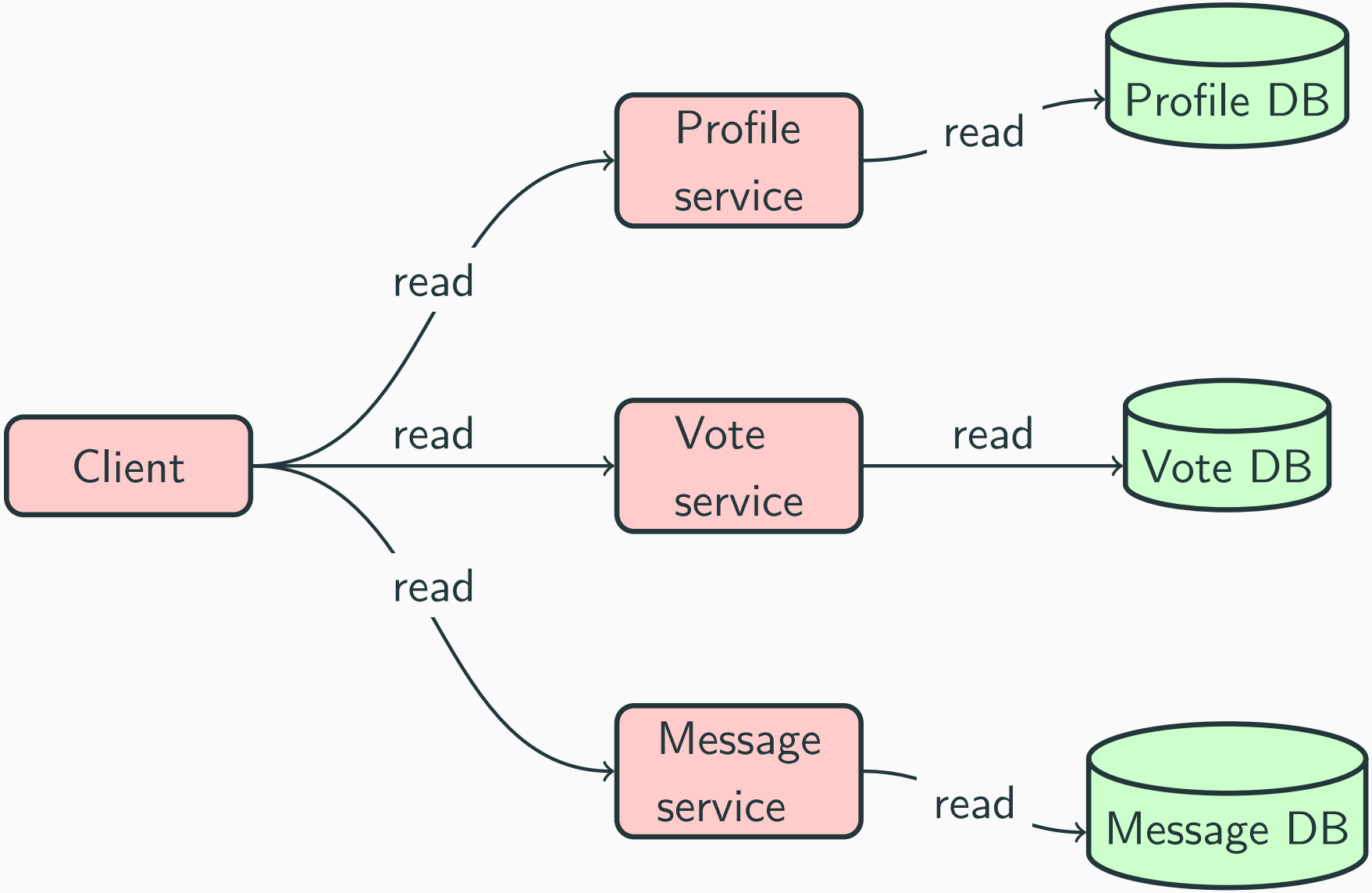
Reading data: microservice dependencies



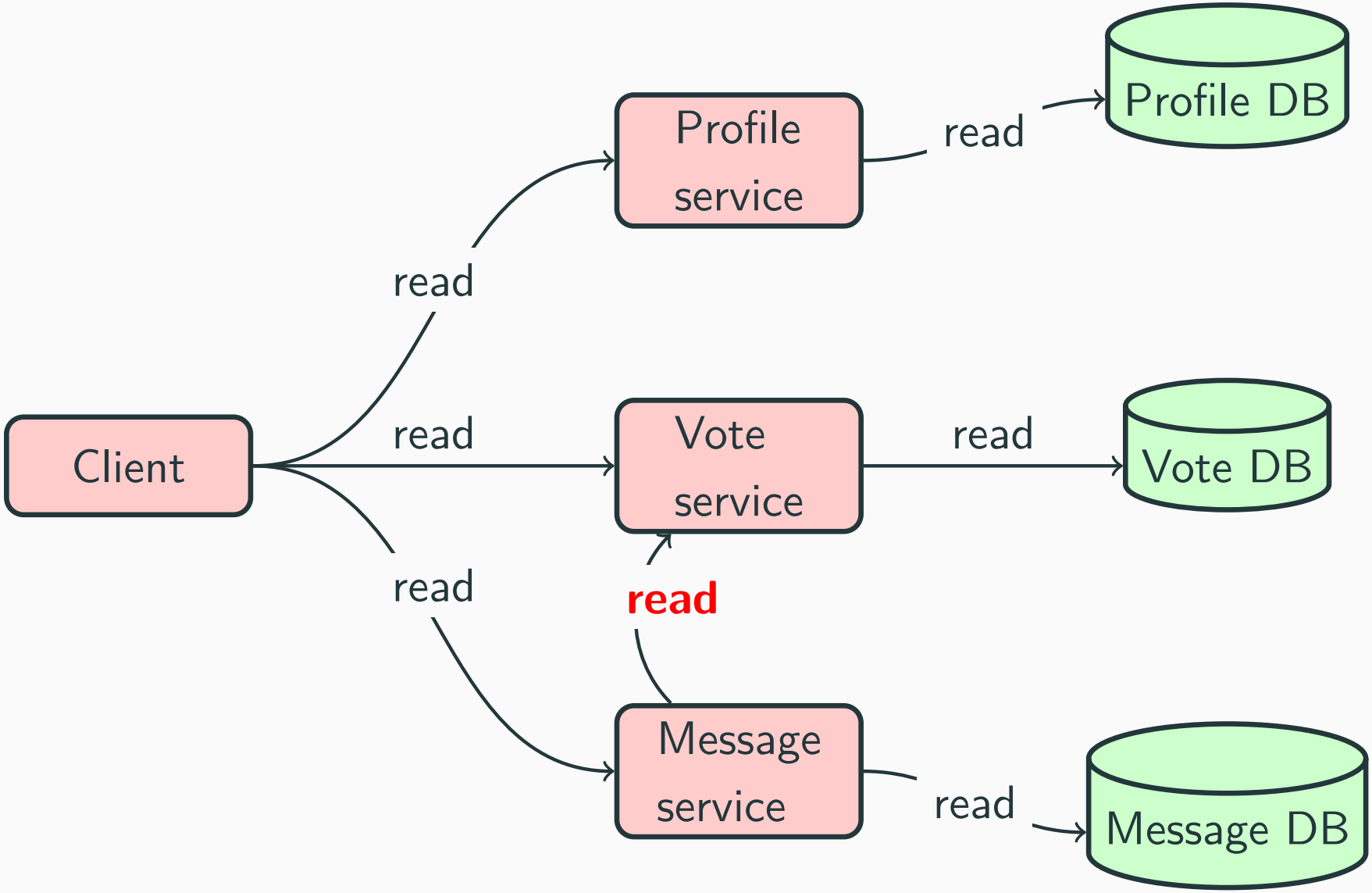
Reading data: microservice dependencies



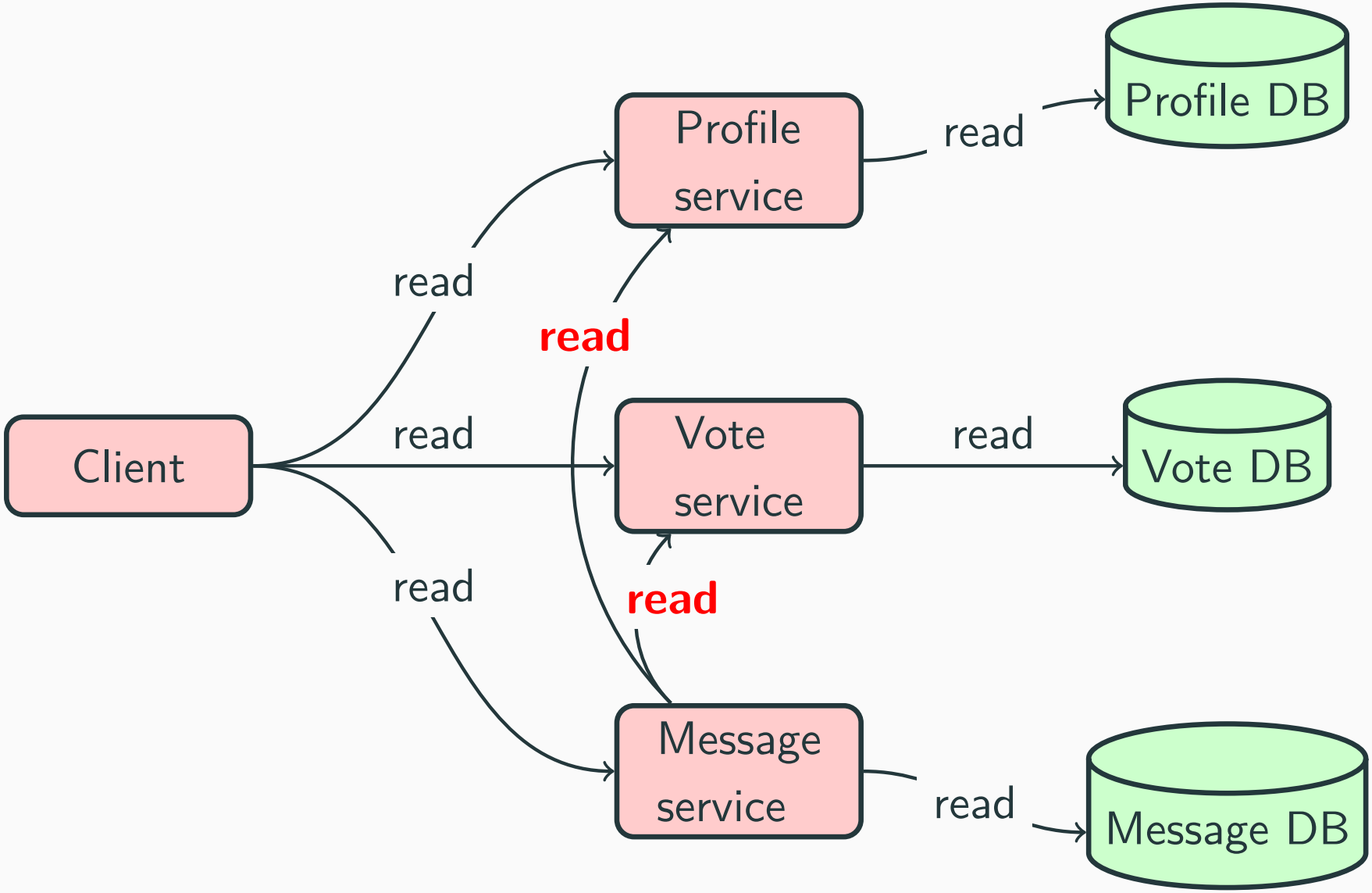
Reading data: microservice dependencies



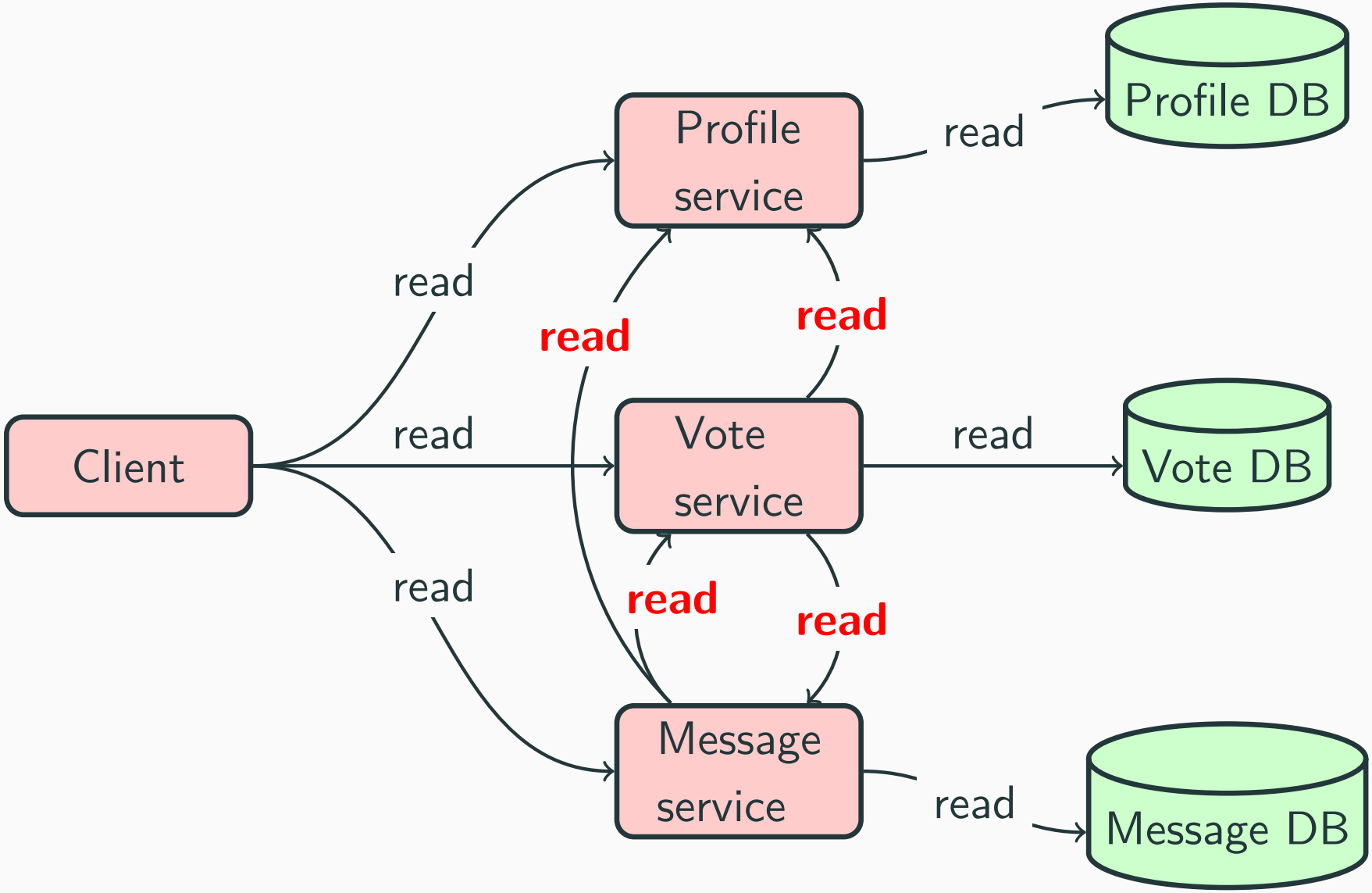
Reading data: microservice dependencies



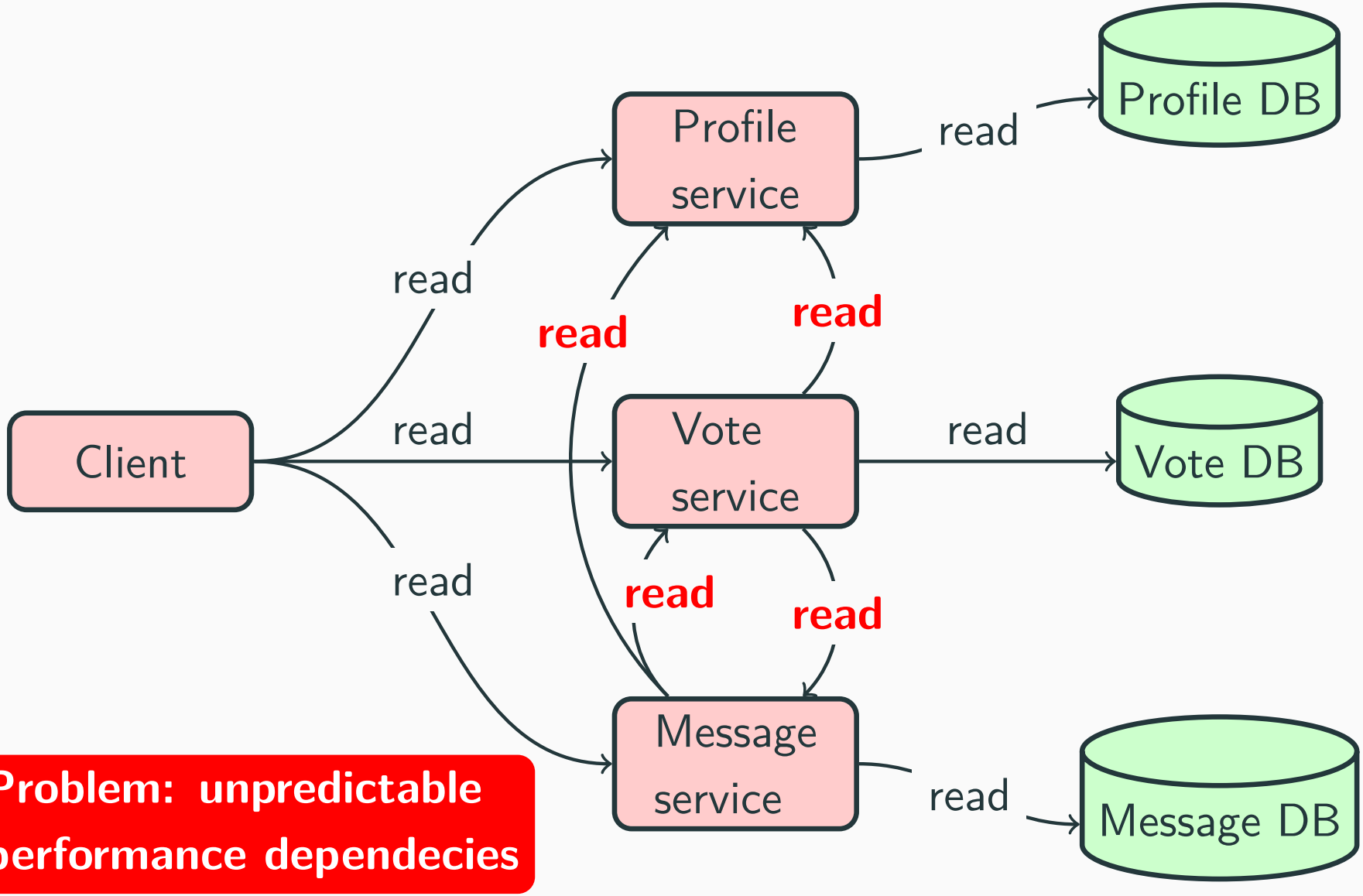
Reading data: microservice dependencies



Reading data: microservice dependencies

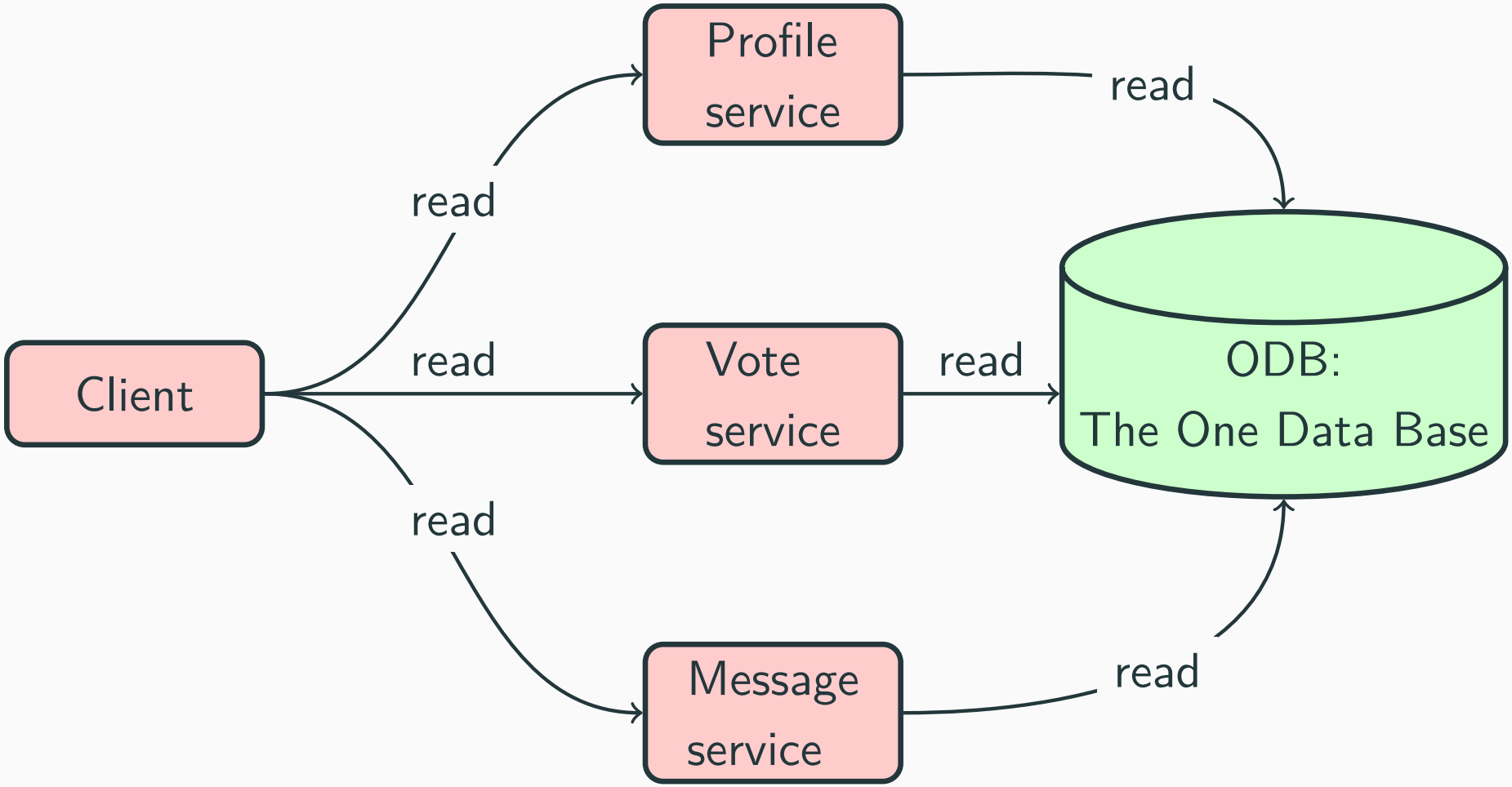


Reading data: microservice dependencies

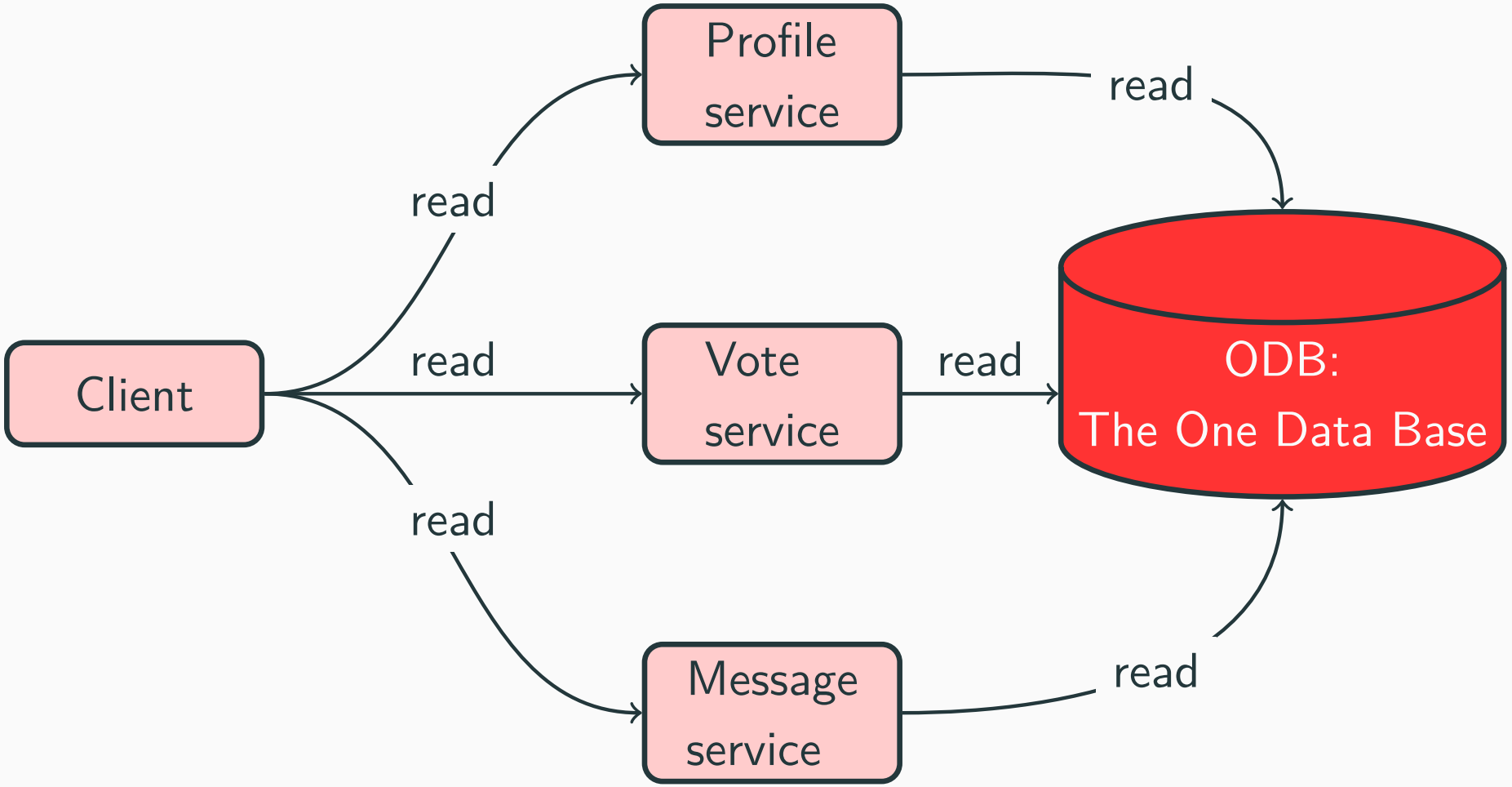


Problem: unpredictable performance dependecies

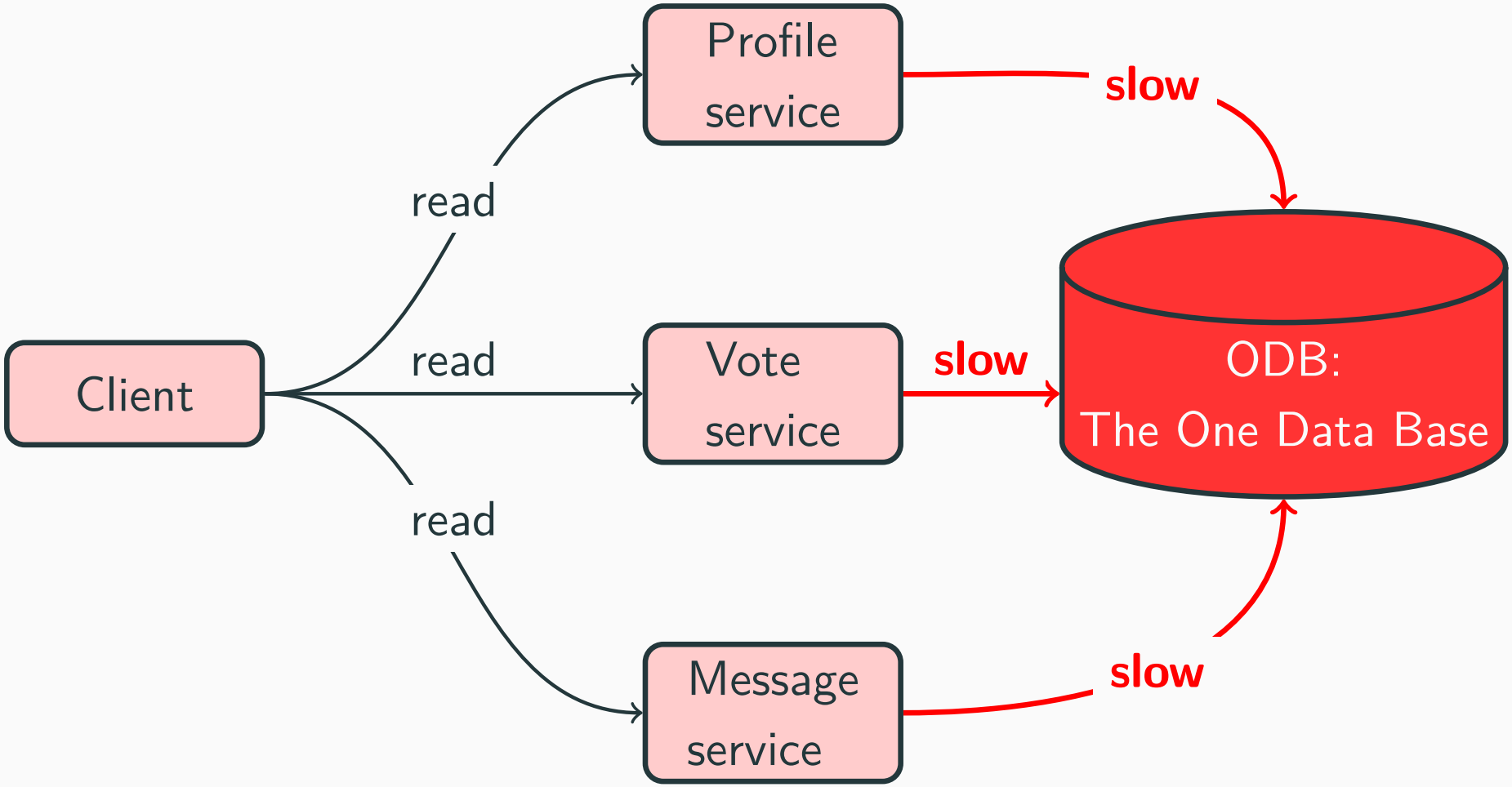
Reading data: “old school” approach



Reading data: “old school” approach

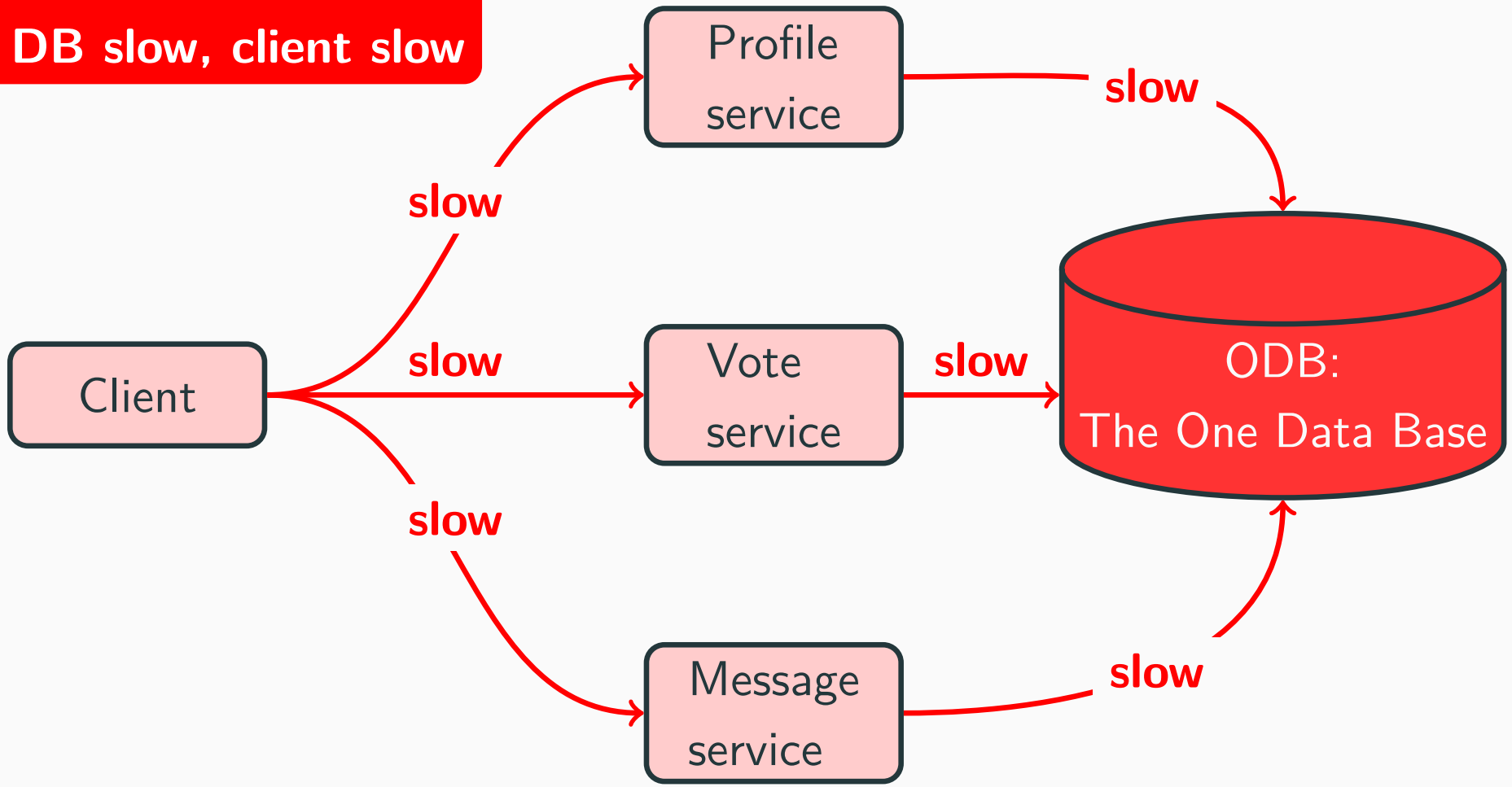


Reading data: “old school” approach



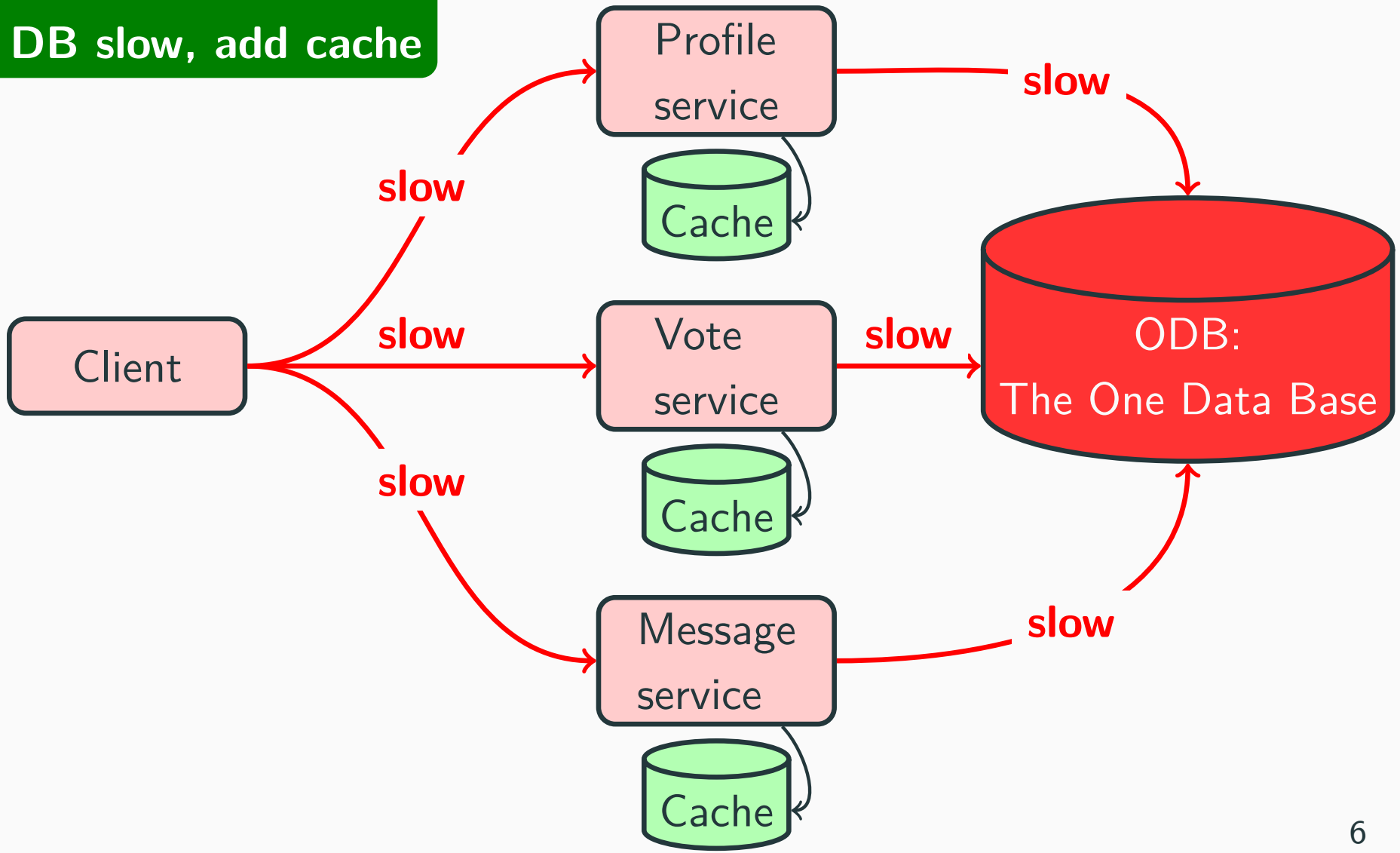
Reading data: “old school” approach

Problem:
if DB slow, client slow



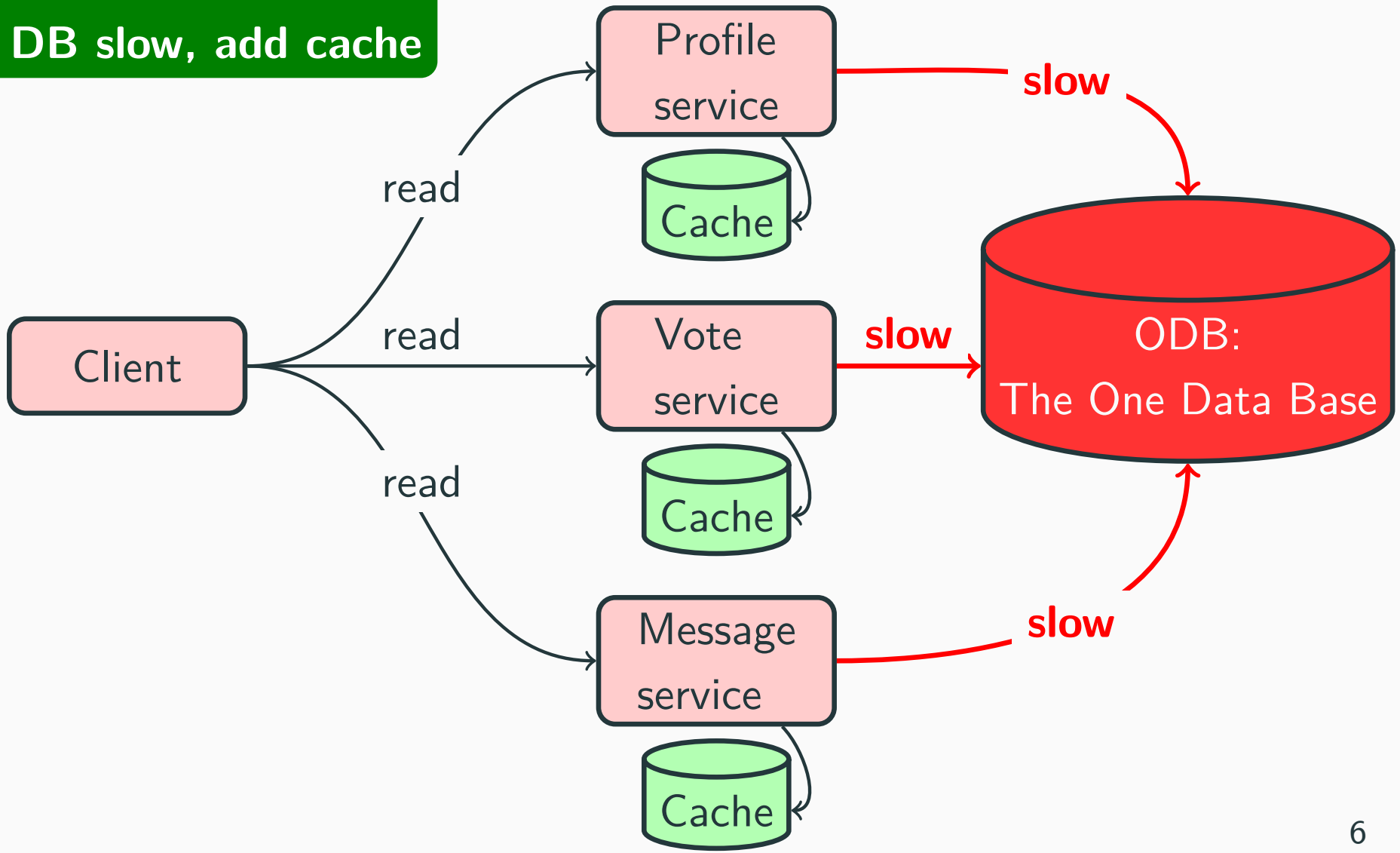
Reading data: “old school” approach

Solution:
if DB slow, add cache



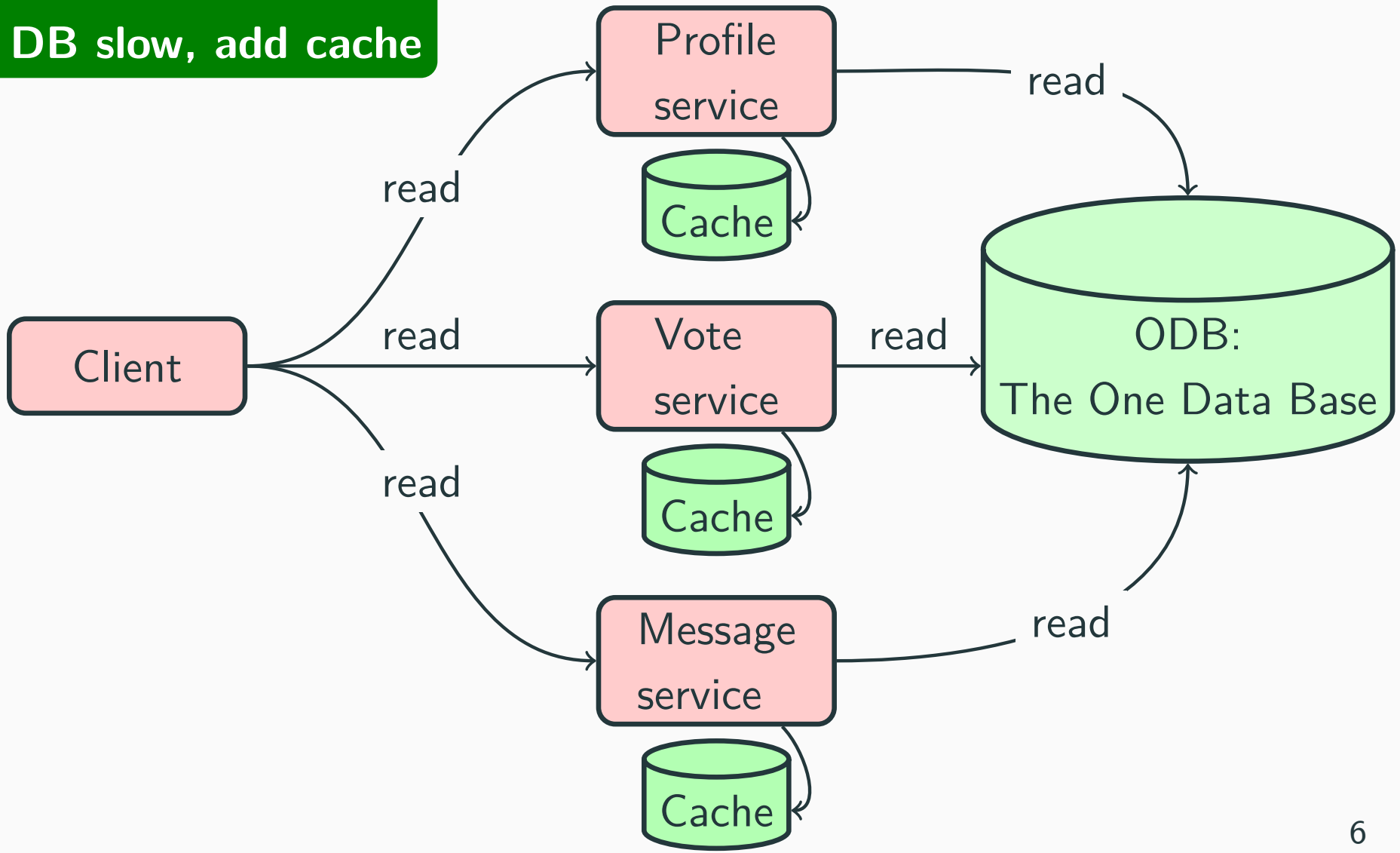
Reading data: “old school” approach

Solution:
if DB slow, add cache



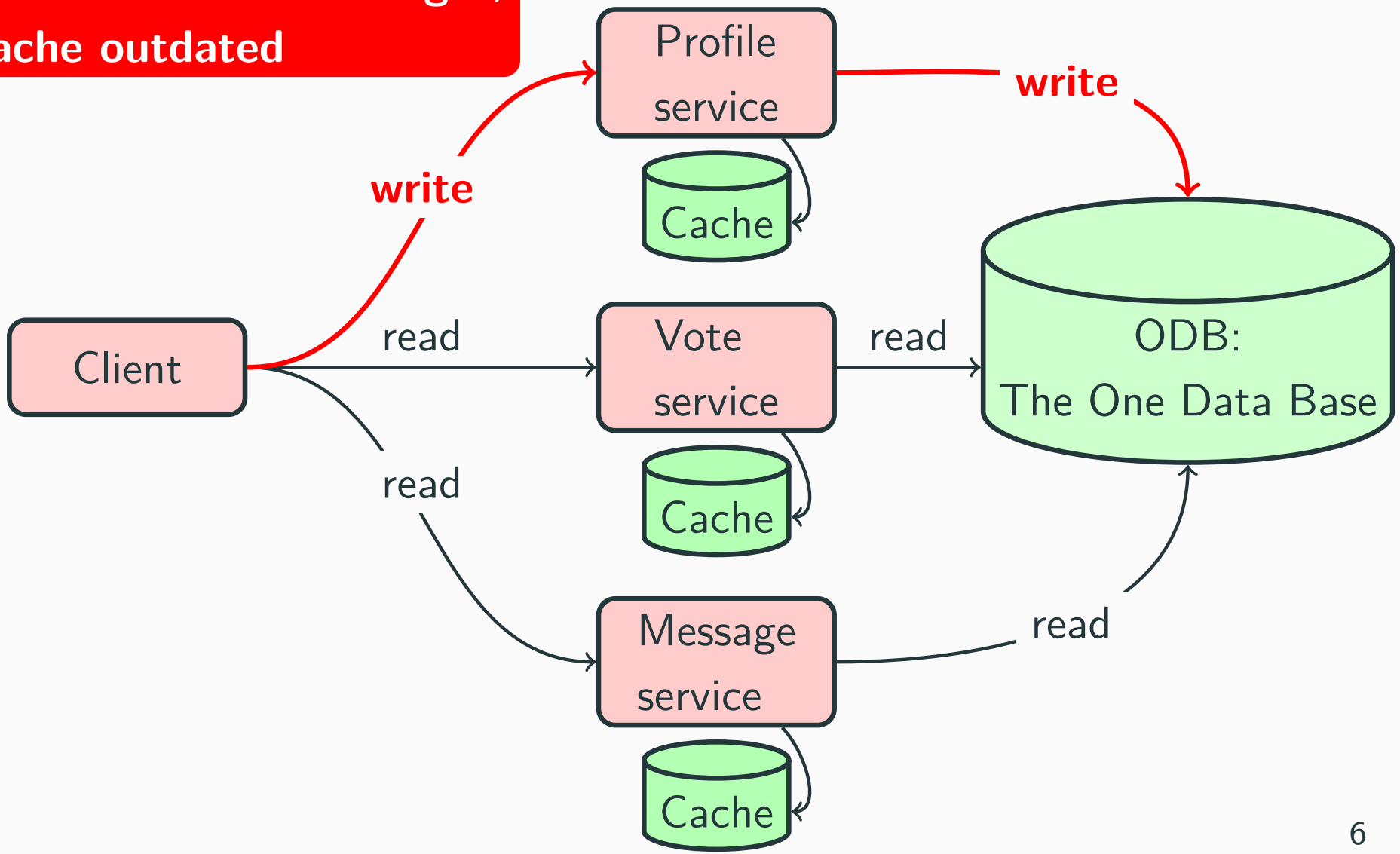
Reading data: “old school” approach

Solution:
if DB slow, add cache



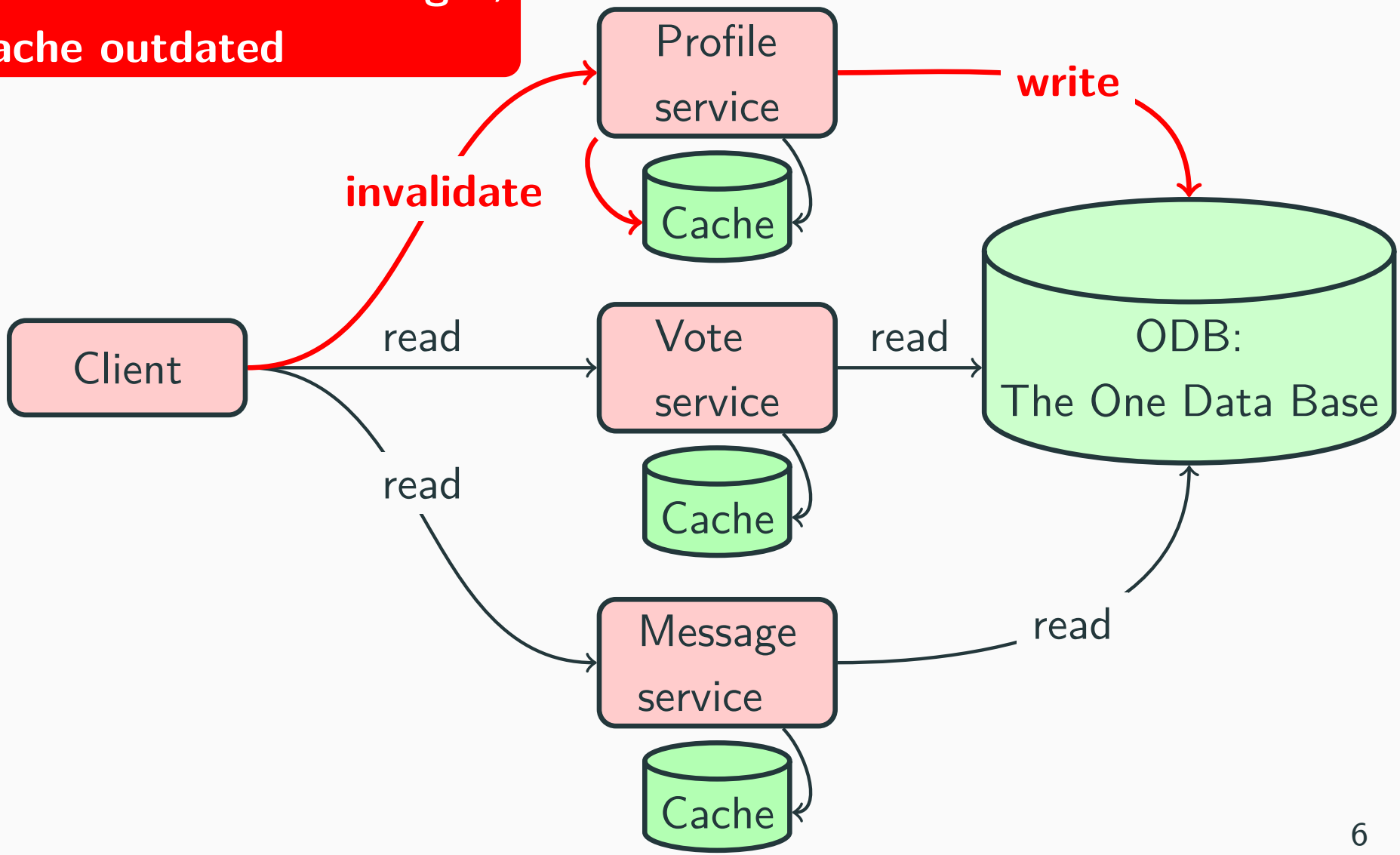
Reading data: “old school” approach

Problem: if data changes, cache outdated



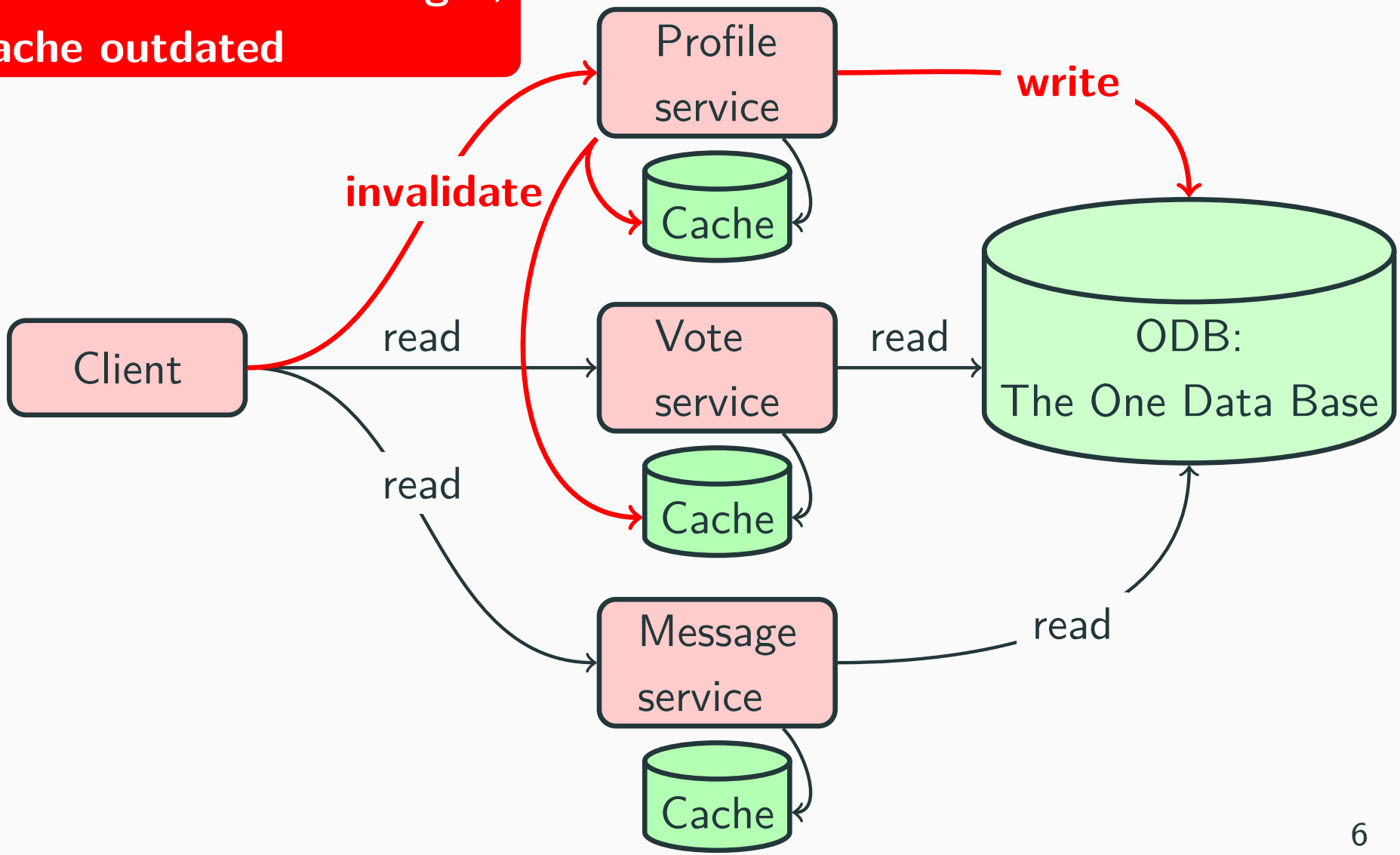
Reading data: “old school” approach

Problem: if data changes, cache outdated



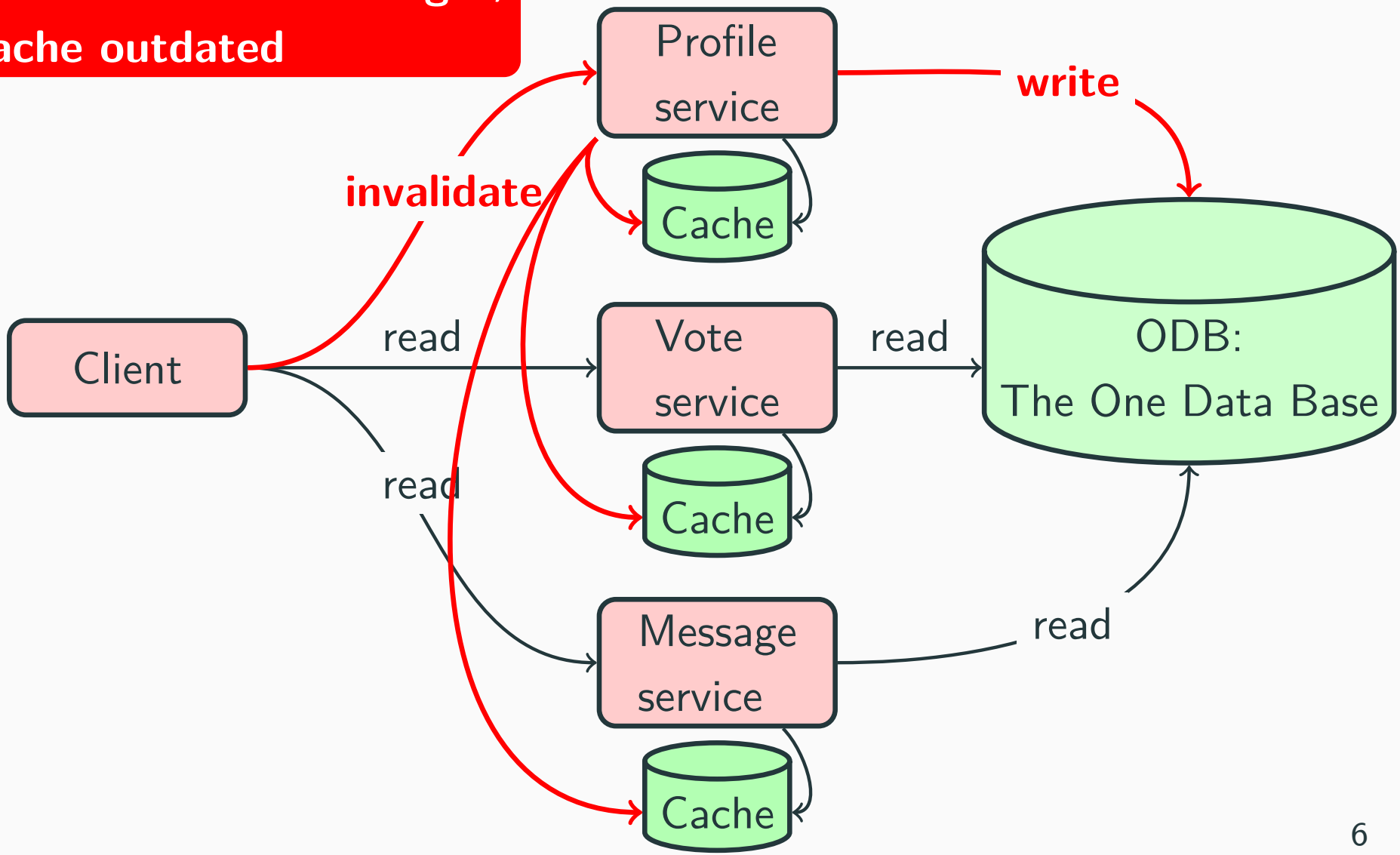
Reading data: “old school” approach

Problem: if data changes, cache outdated

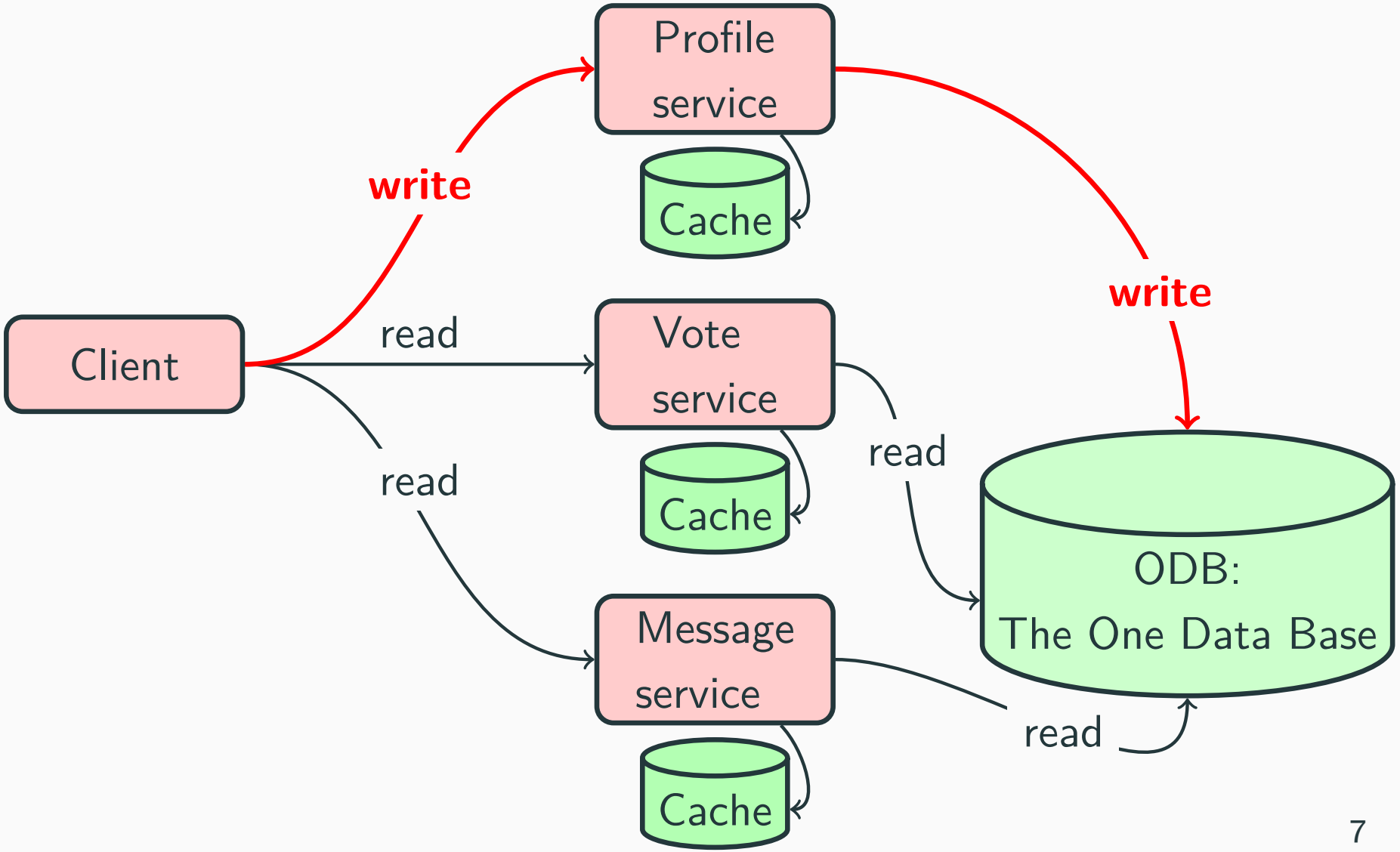


Reading data: “old school” approach

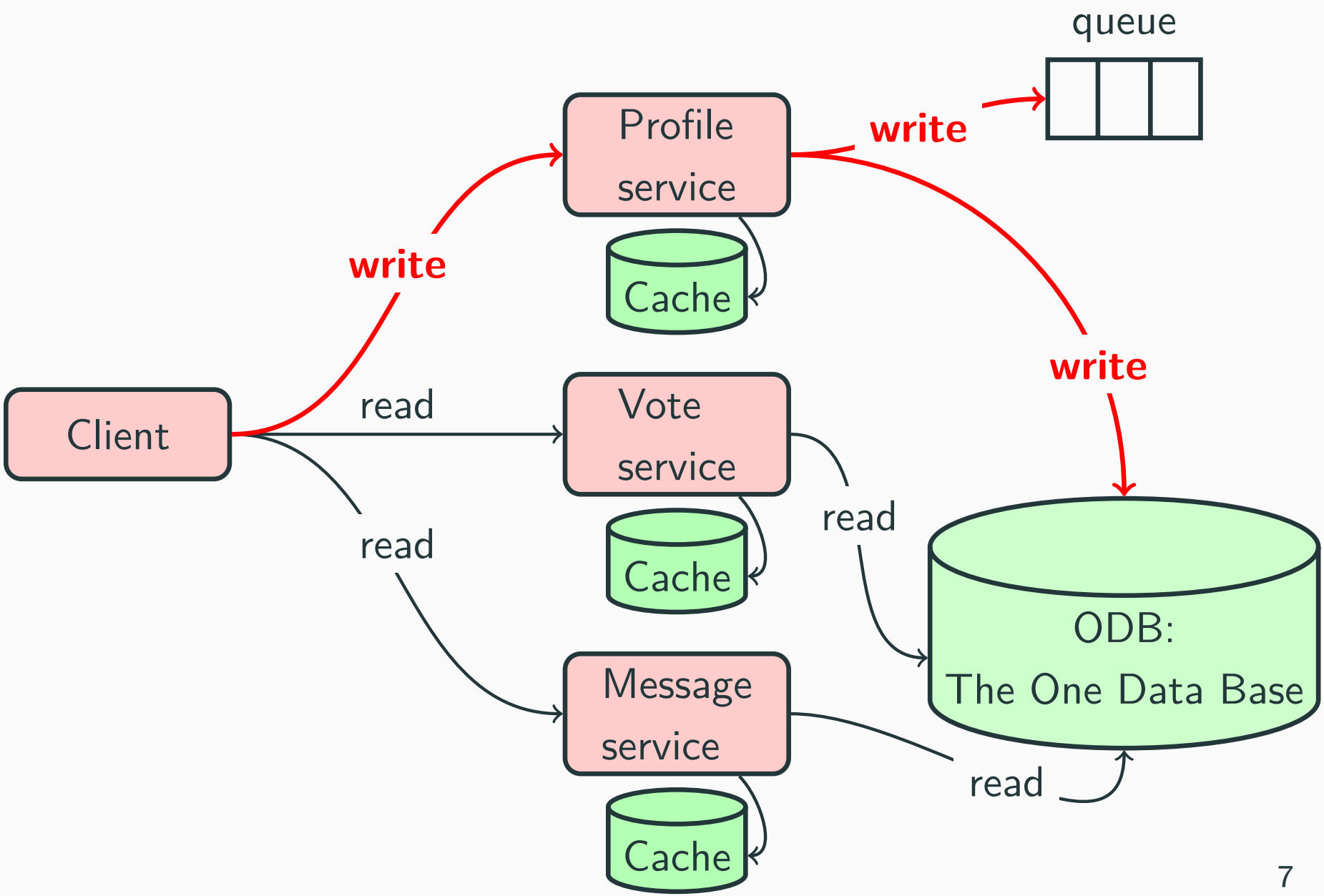
Problem: if data changes, cache outdated



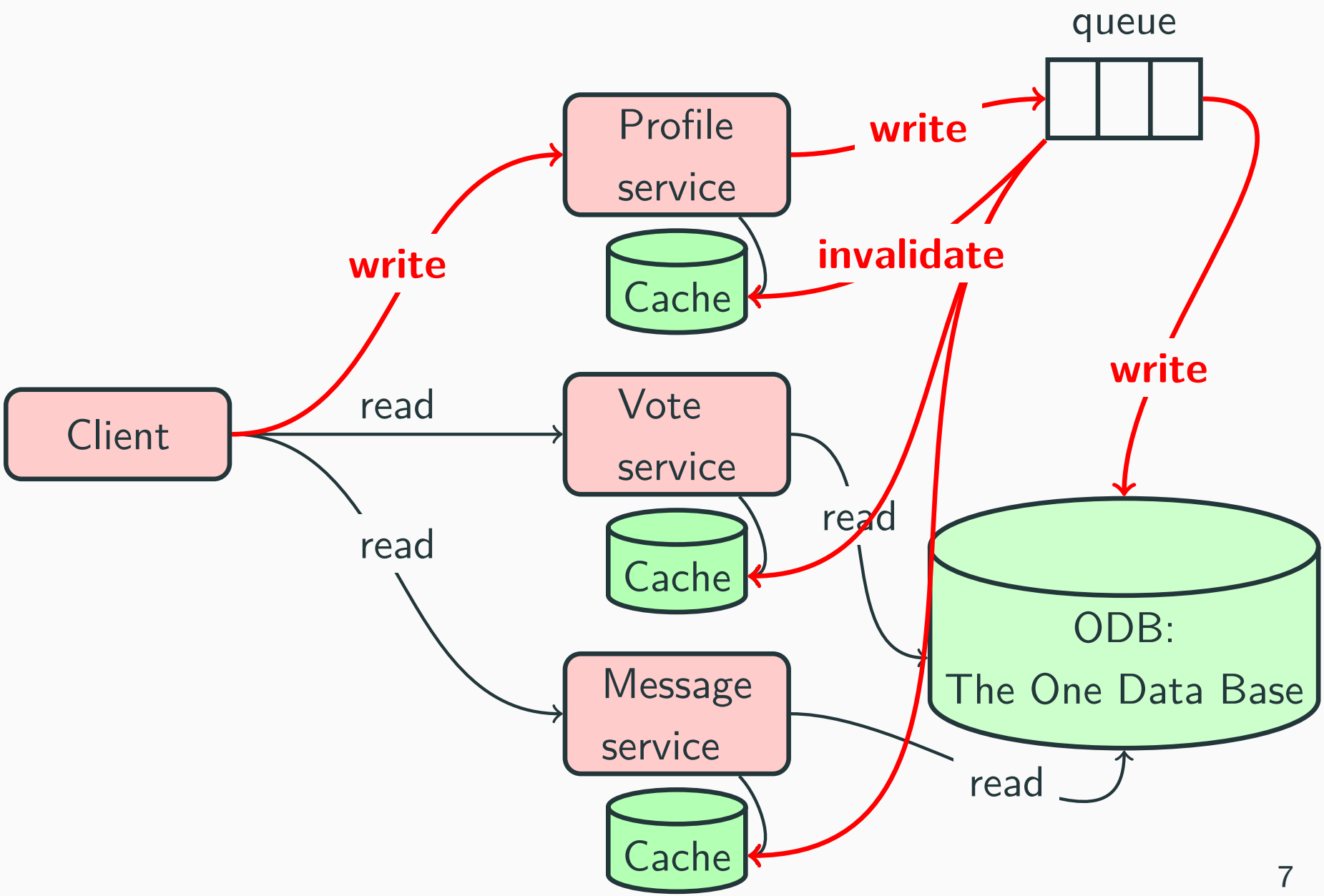
Reading data: decoupling microservices in space and time



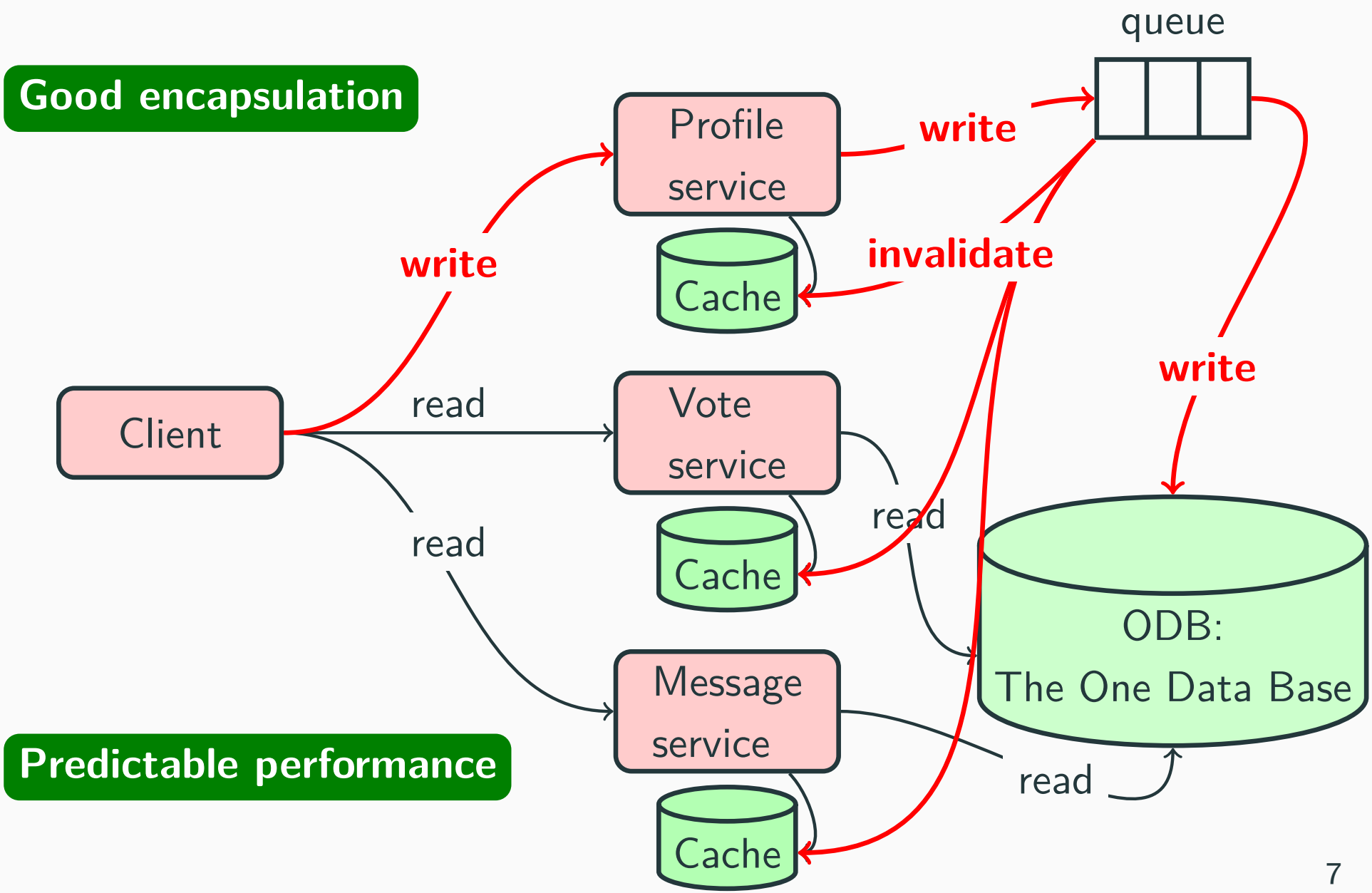
Reading data: decoupling microservices in space and time



Reading data: decoupling microservices in space and time



Reading data: decoupling microservices in space and time



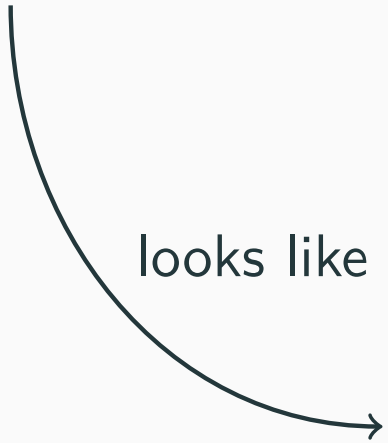
Database abstraction

key	value

SET key1 = value1;

key	**value**

SET key1 = value1;



looks like

key	**value**
key1	value1

SET key1 = value1;

looks like

key	**value**
key1	value1

but actually

The Log

SET key1 = value1;

looks like

key	**value**
key1	value1

but actually

(0, key1, value1)

SET key1 = value1;

SET key2 = value2;

looks like

key	**value**
key1	value1
key2	value2

append

(0, key1, value1)

(1, key2, value2)

SET key1 = value1;

SET key2 = value2;

SET key1 = value3;

looks like

key	**value**
key1	value3
key2	value2

append

(0, key1, value1)

(1, key2, value2)

(2, key1, value3)

SET key1 = value1;

SET key2 = value2;

SET key1 = value3;

append

master

key	value
key1	value3
key2	value2

(0, key1, value1)

(1, key2, value2)

(2, key1, value3)

SET key1 = value1;

SET key2 = value2;

SET key1 = value3;

append

(0, key1, value1)

(1, key2, value2)

(2, key1, value3)

master

key	value
key1	value3
key2	value2

backup

key	value

SET key1 = value1;

SET key2 = value2;

SET key1 = value3;

append

(0, key1, value1)

(1, key2, value2)

(2, key1, value3)

catchup

master

key	value
key1	value3
key2	value2

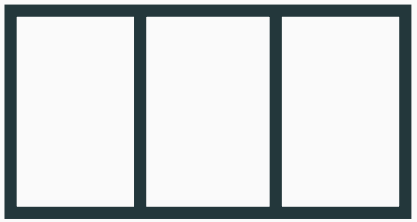
backup

key	value
key1	value3
key2	value2

=

?

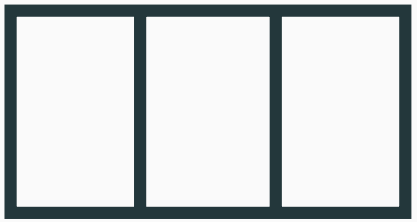
$=$



?

Log is really FIFO: order matters!

$=$

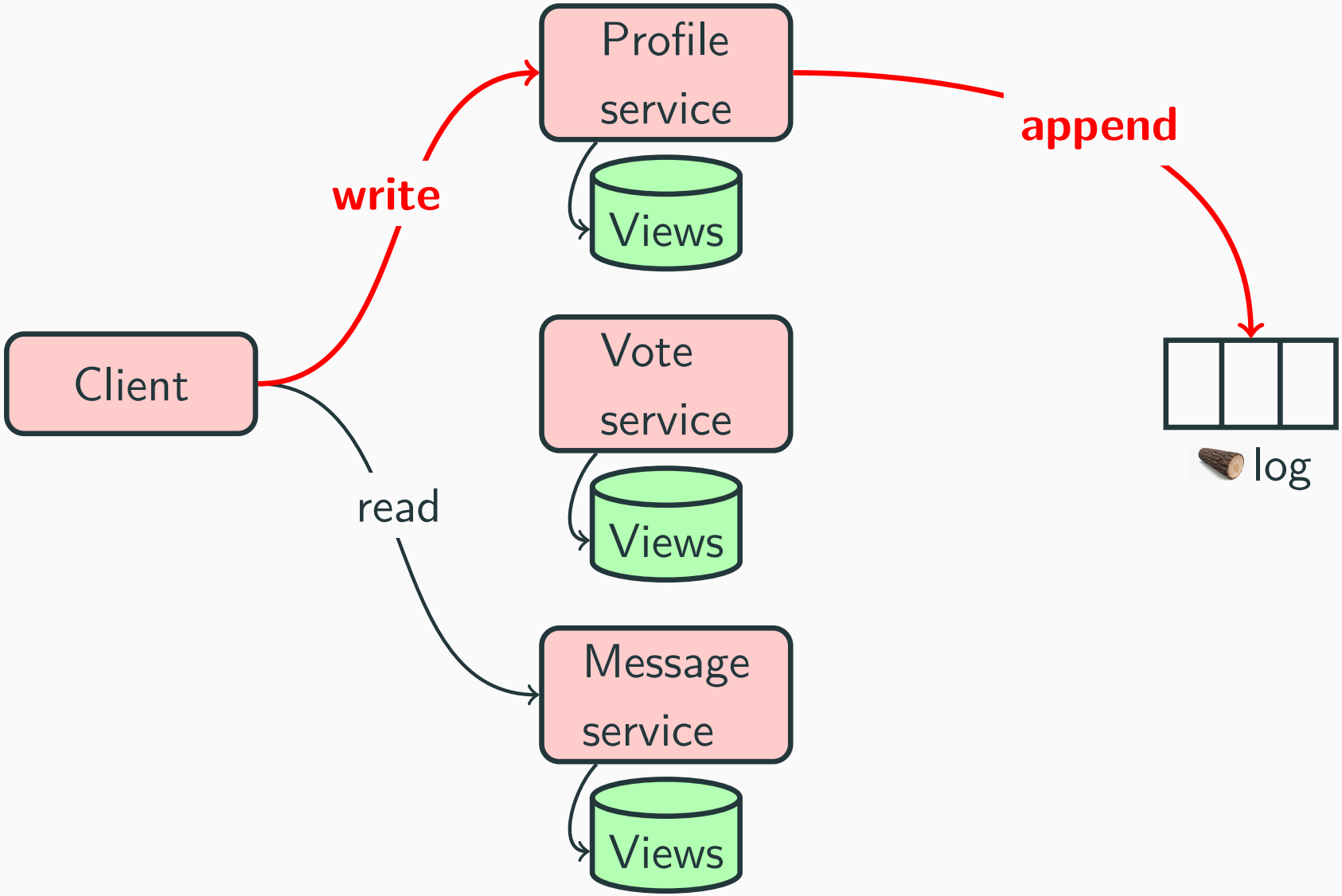


?

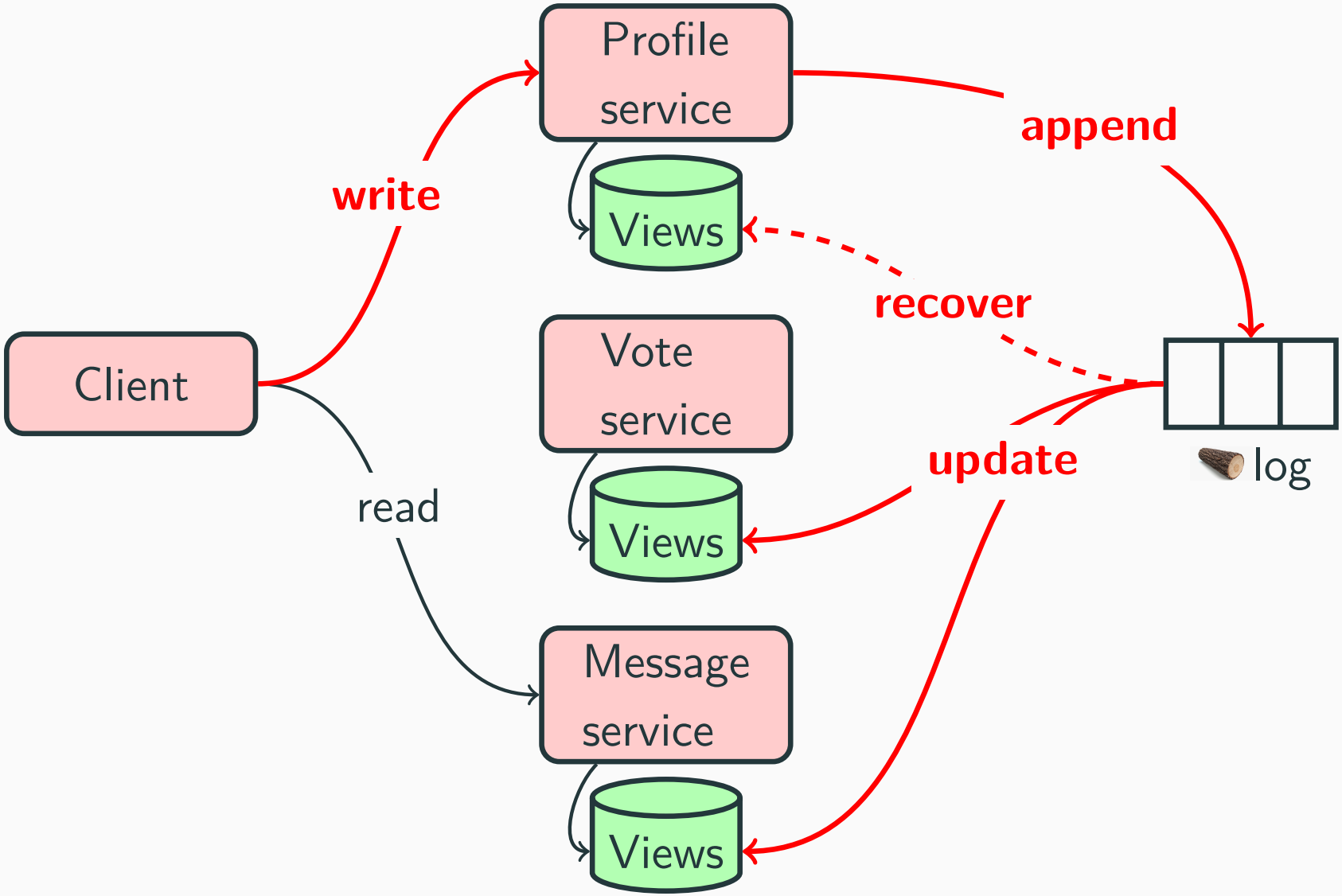
Log is really FIFO: order matters!

Log is permanent – Queue is transient

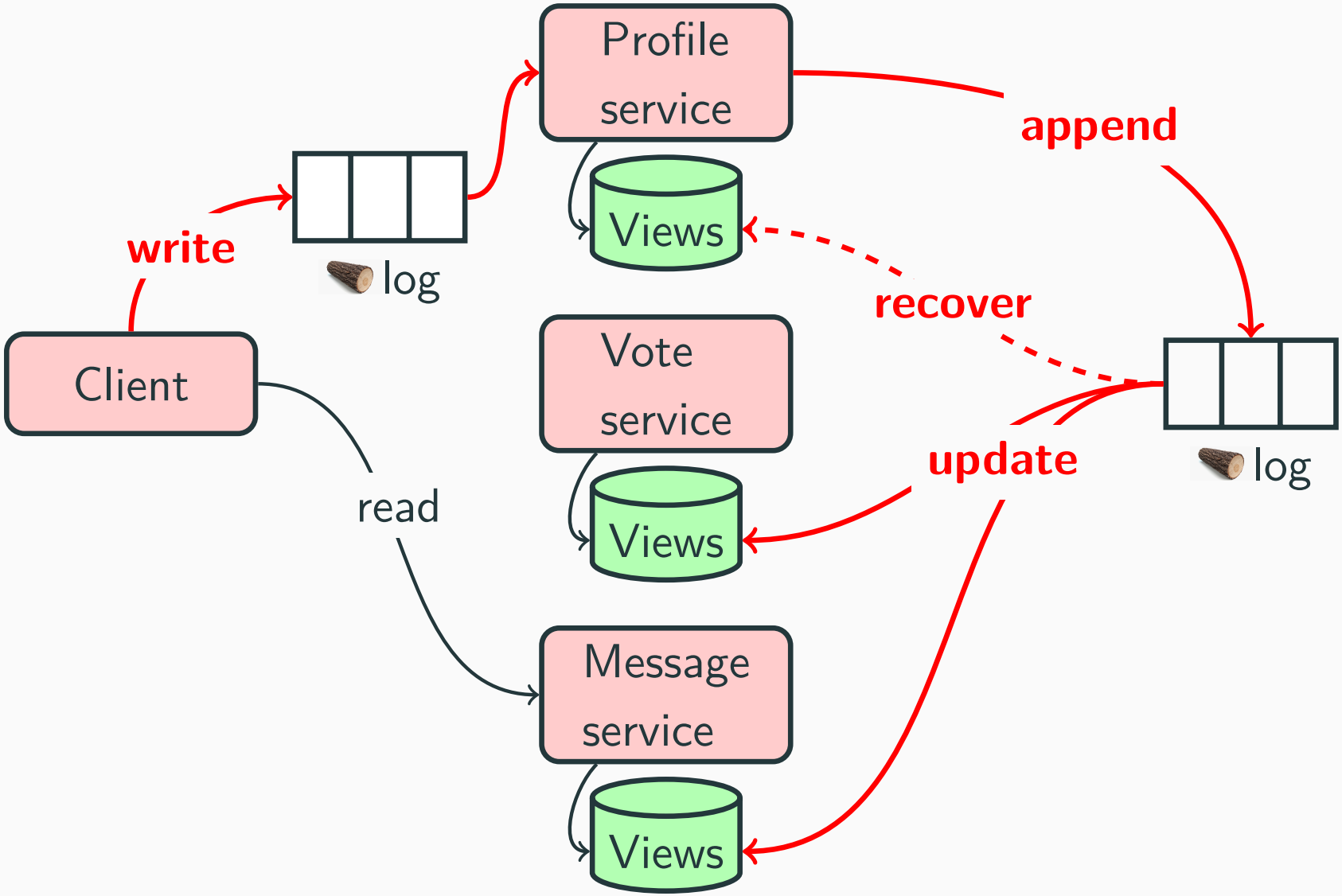
Stateful Stream Processing



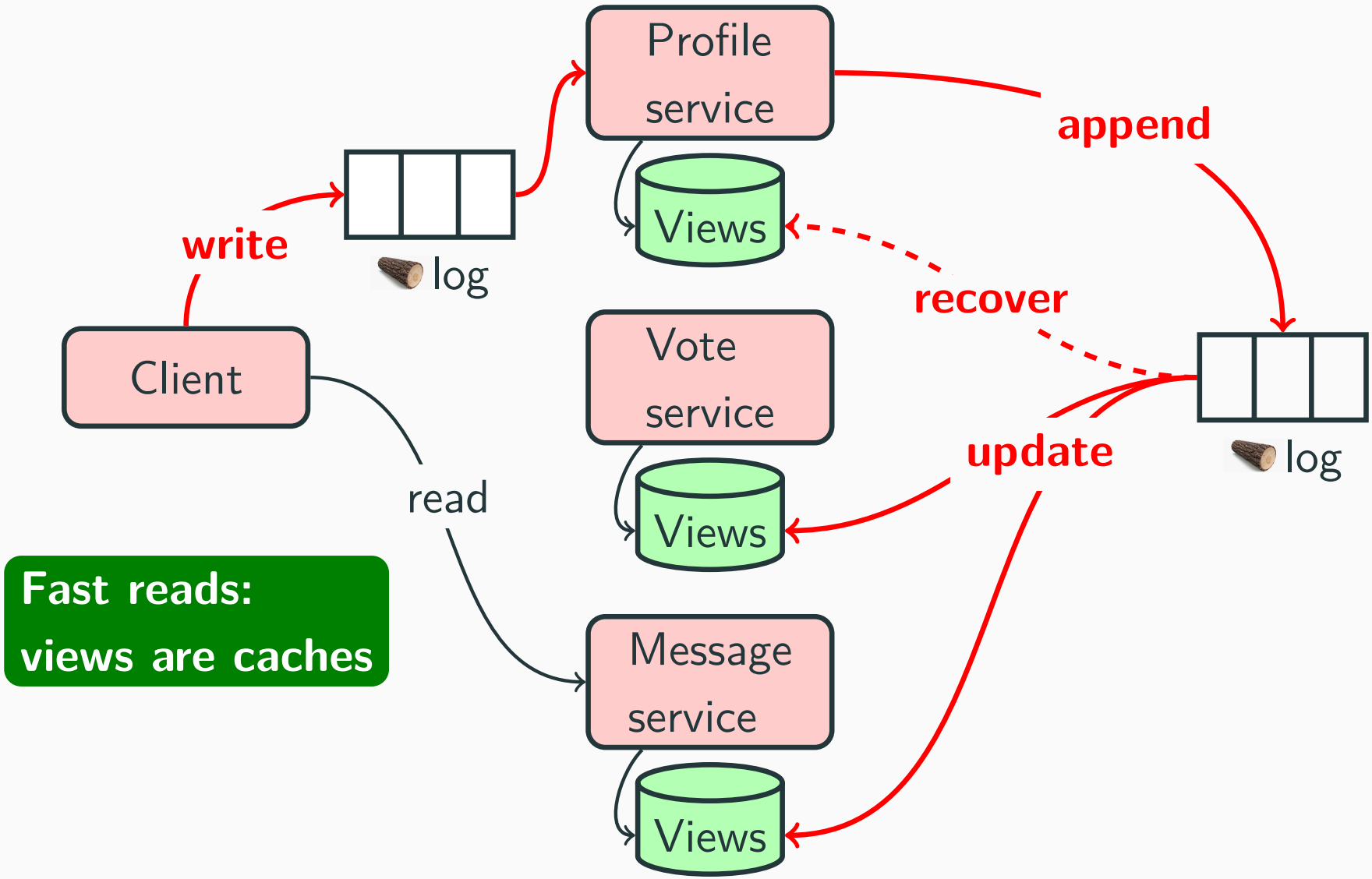
Stateful Stream Processing



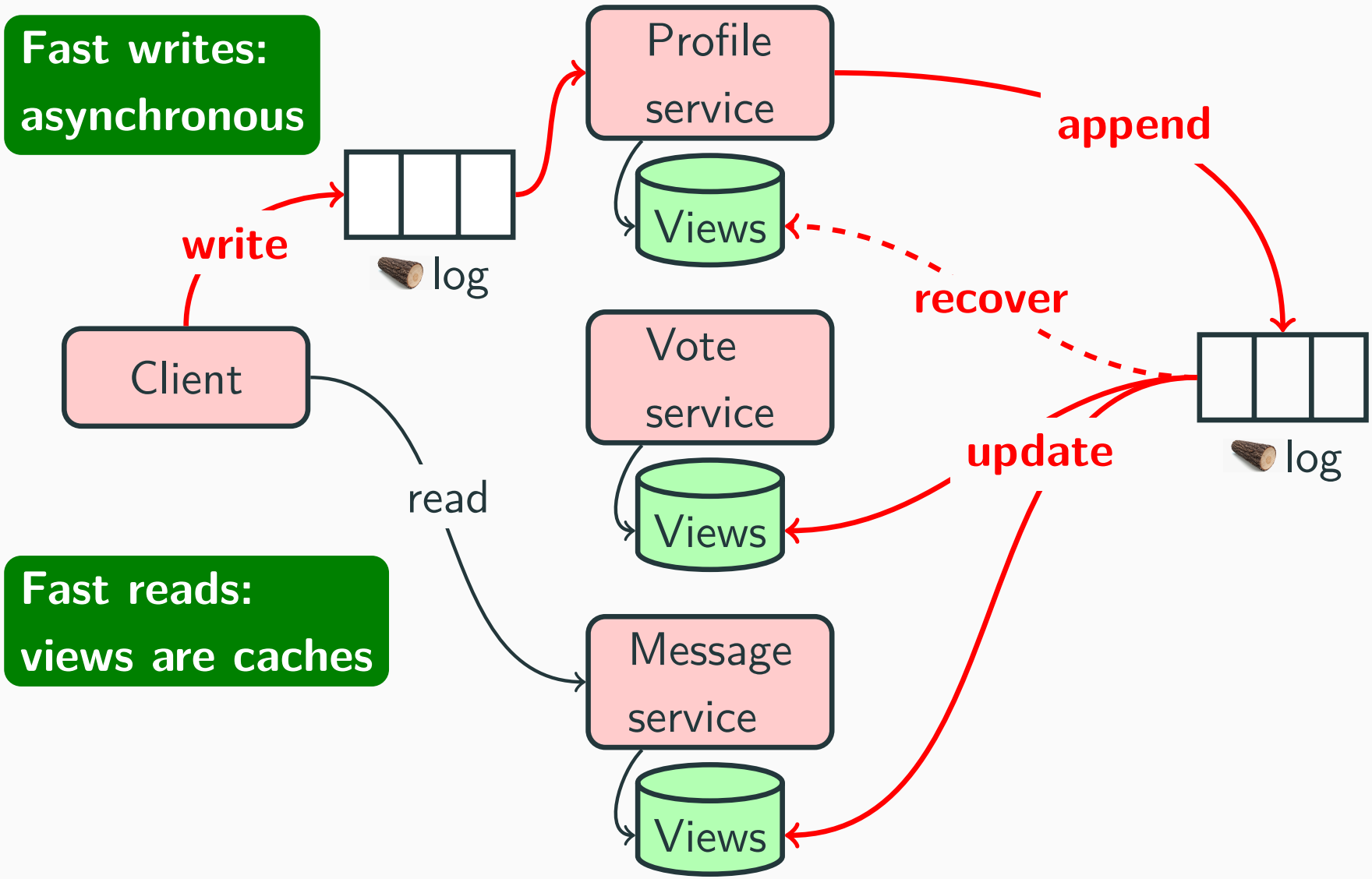
Stateful Stream Processing



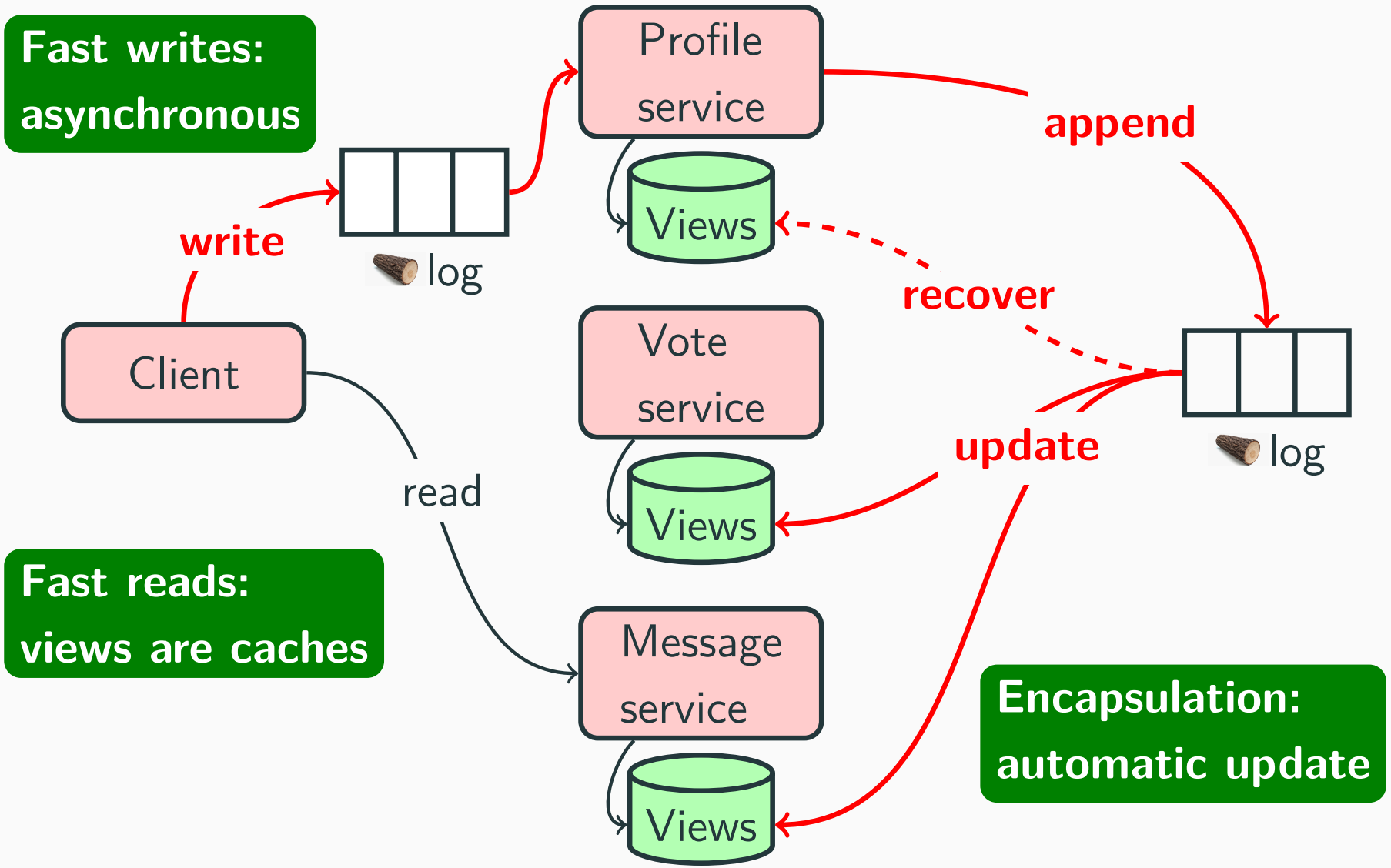
Stateful Stream Processing



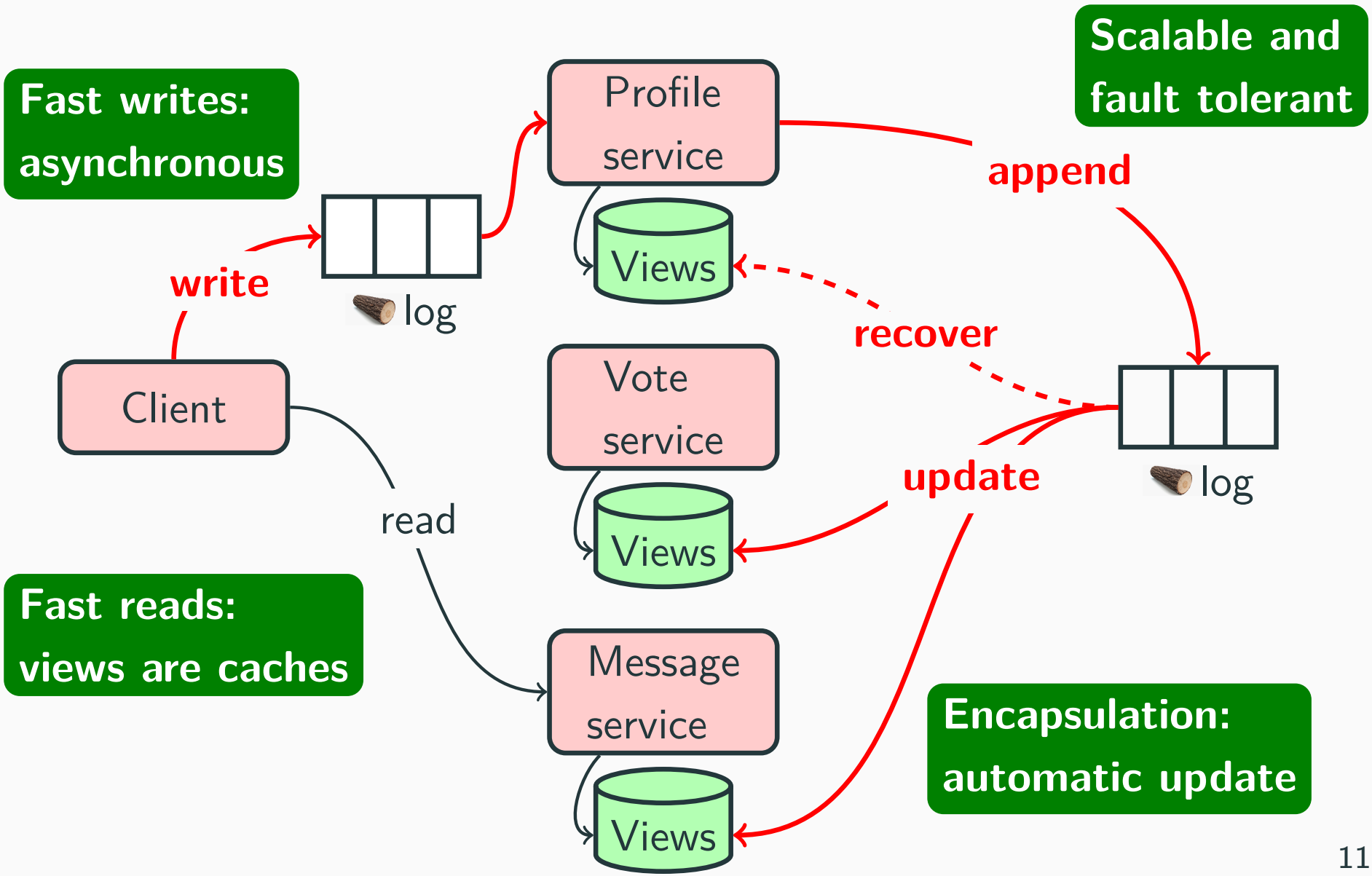
Stateful Stream Processing



Stateful Stream Processing

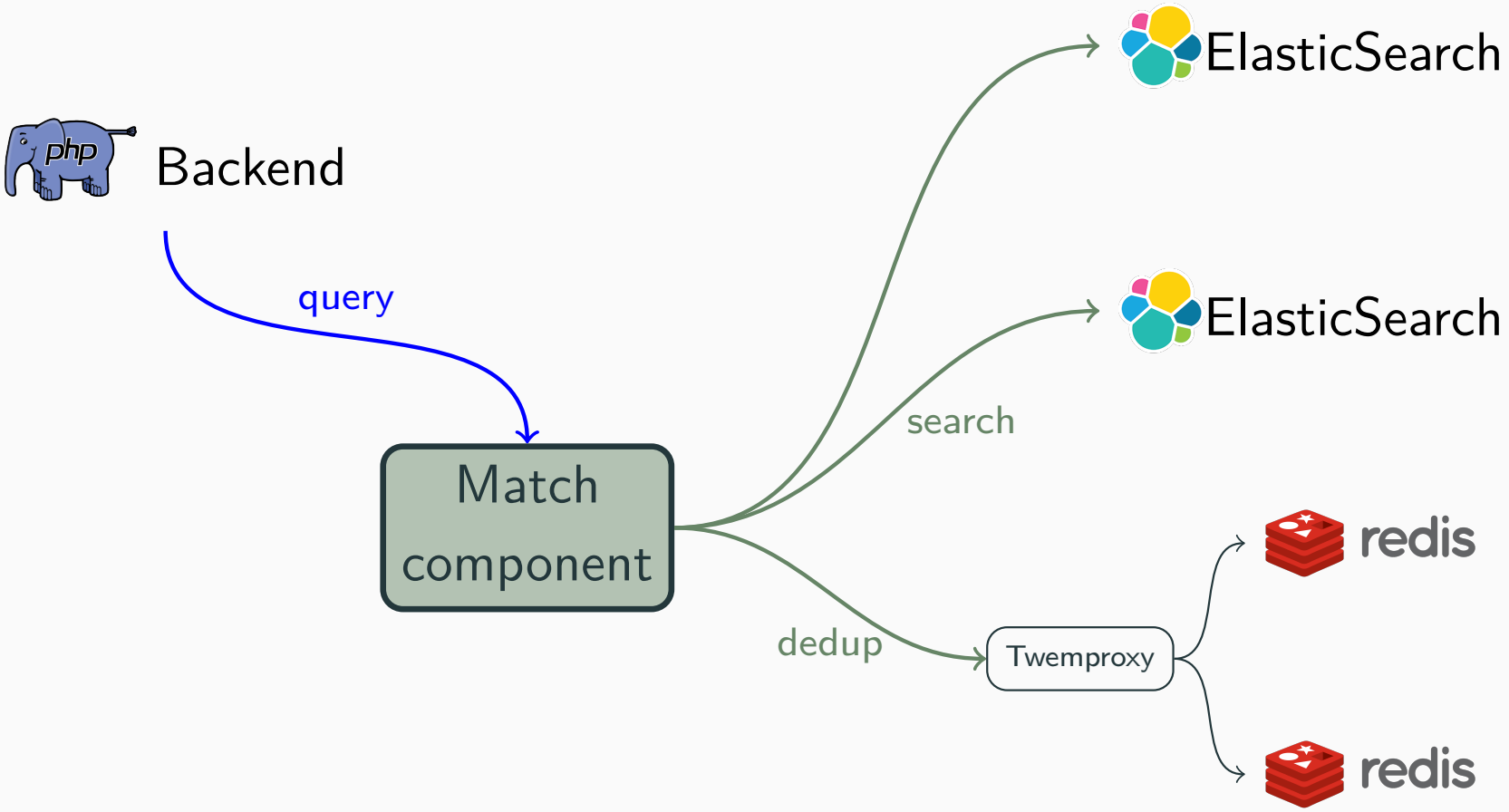


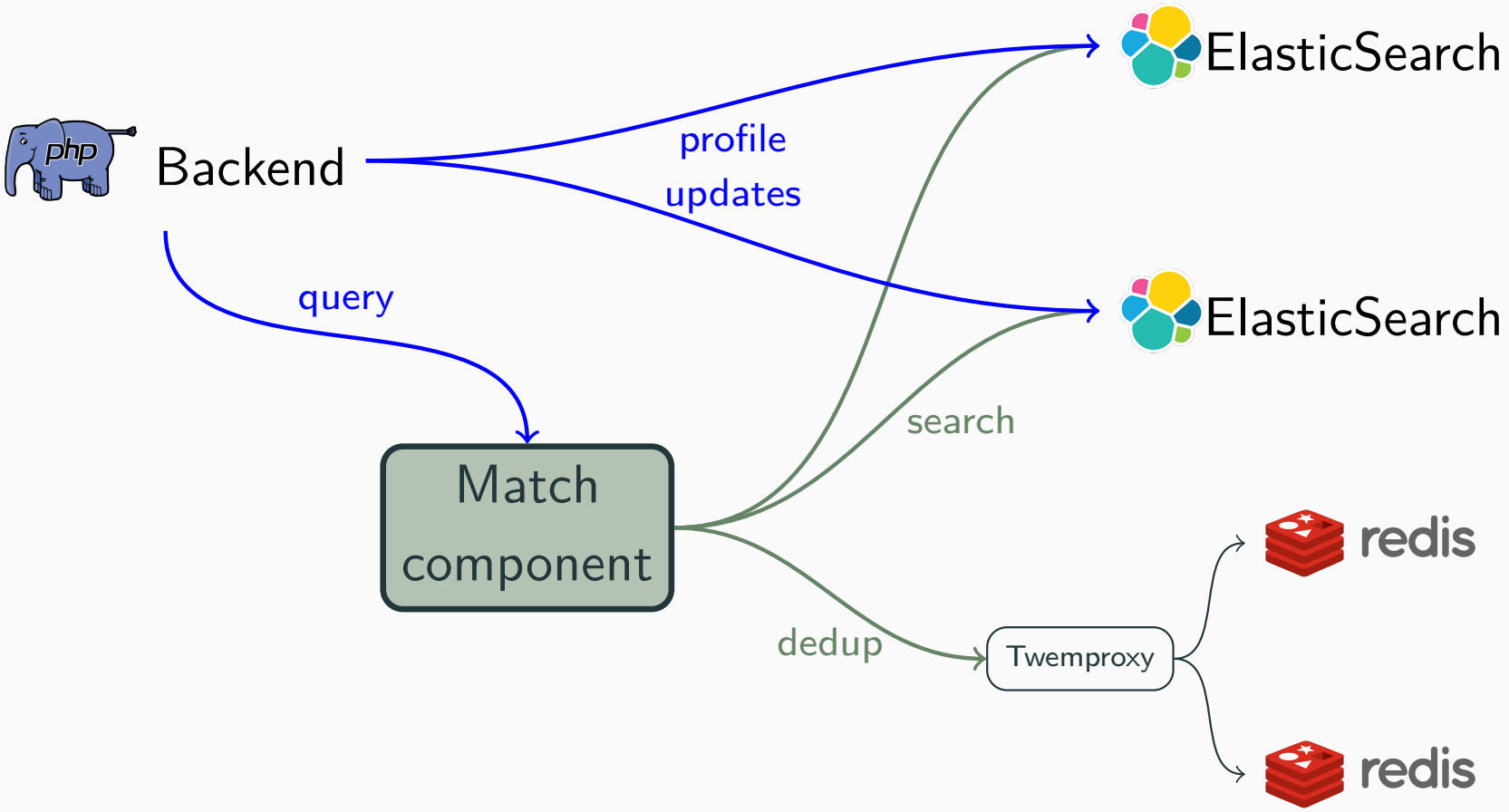
Stateful Stream Processing

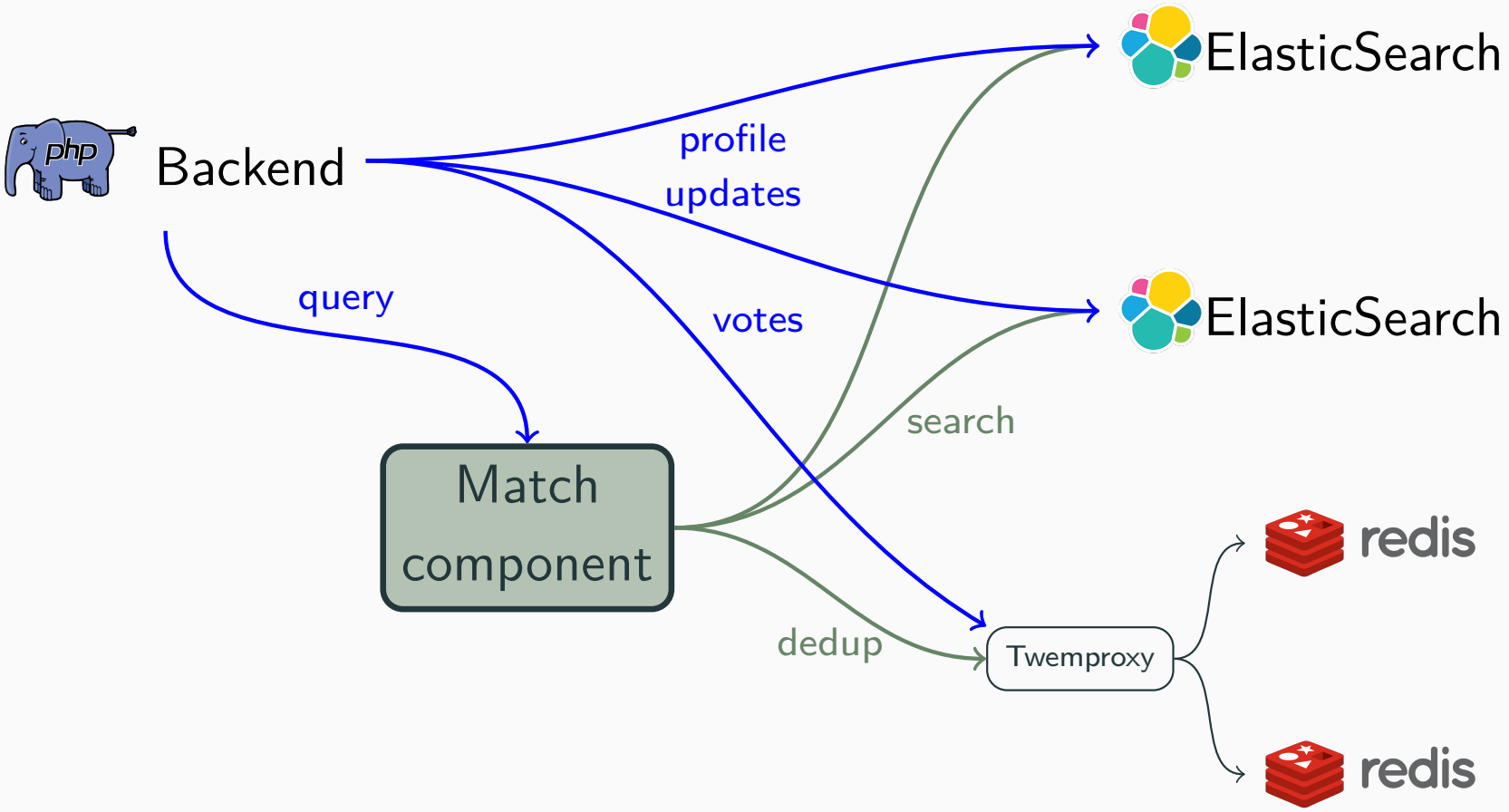


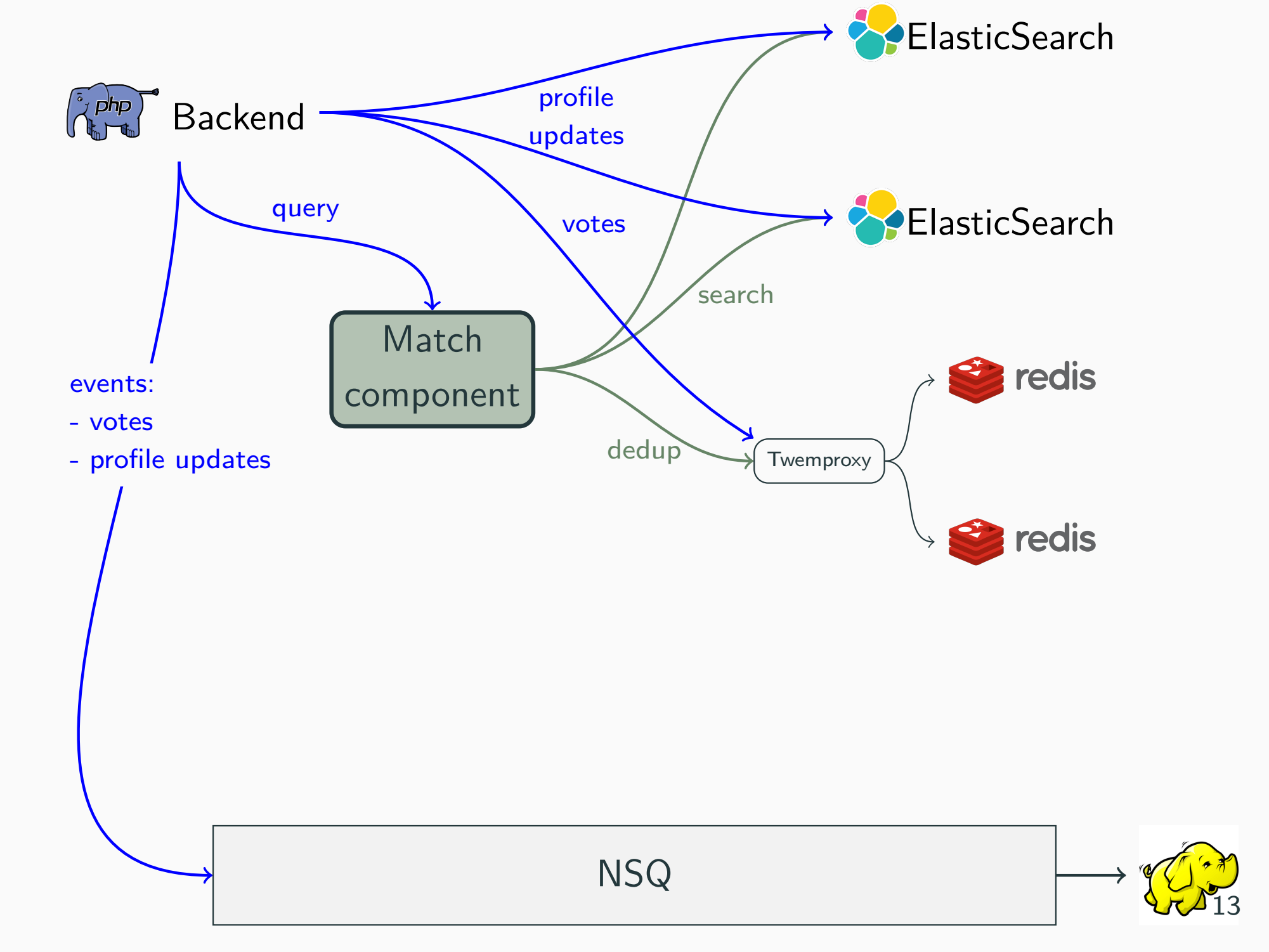
And now for something completely different...

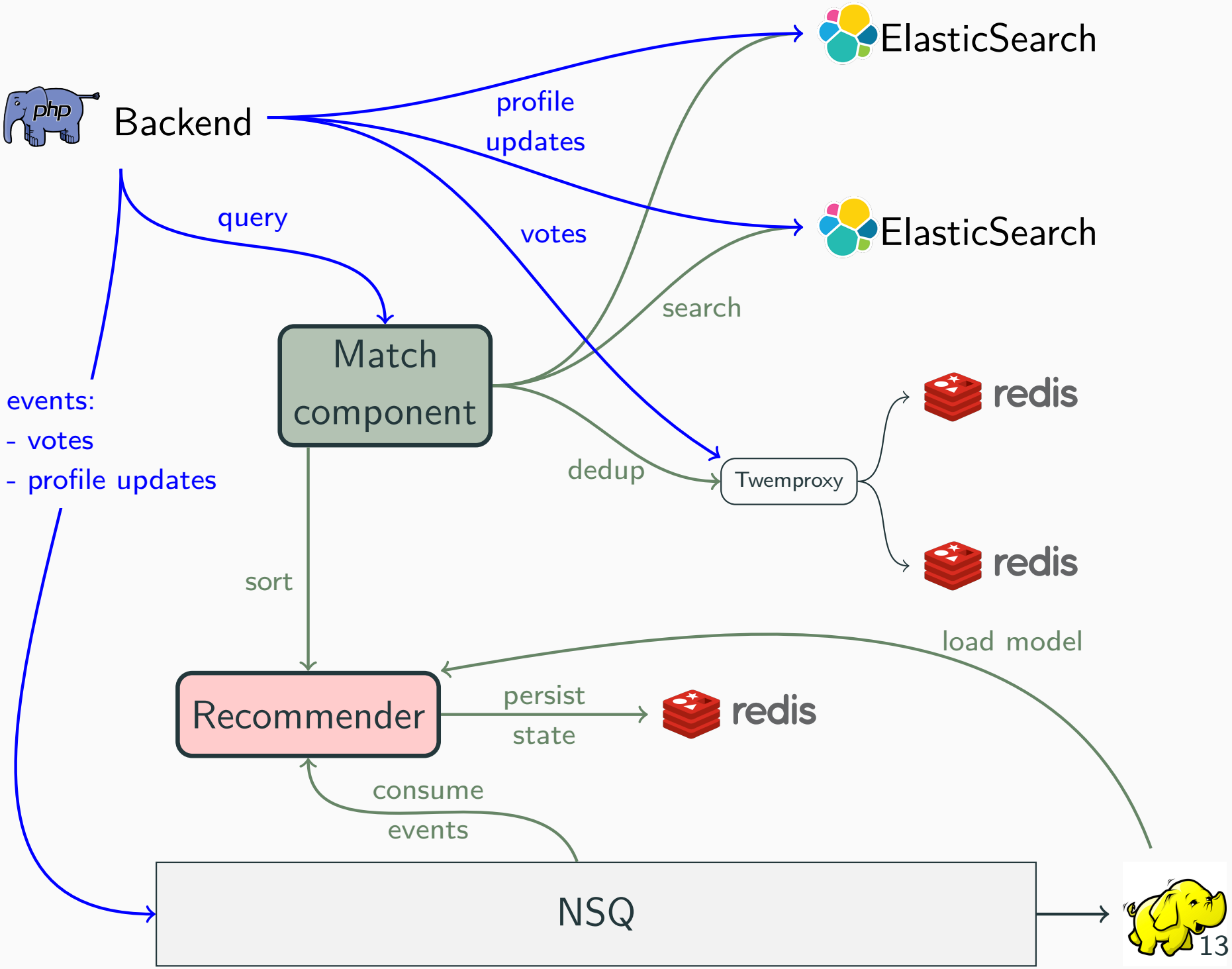
A Real Example!

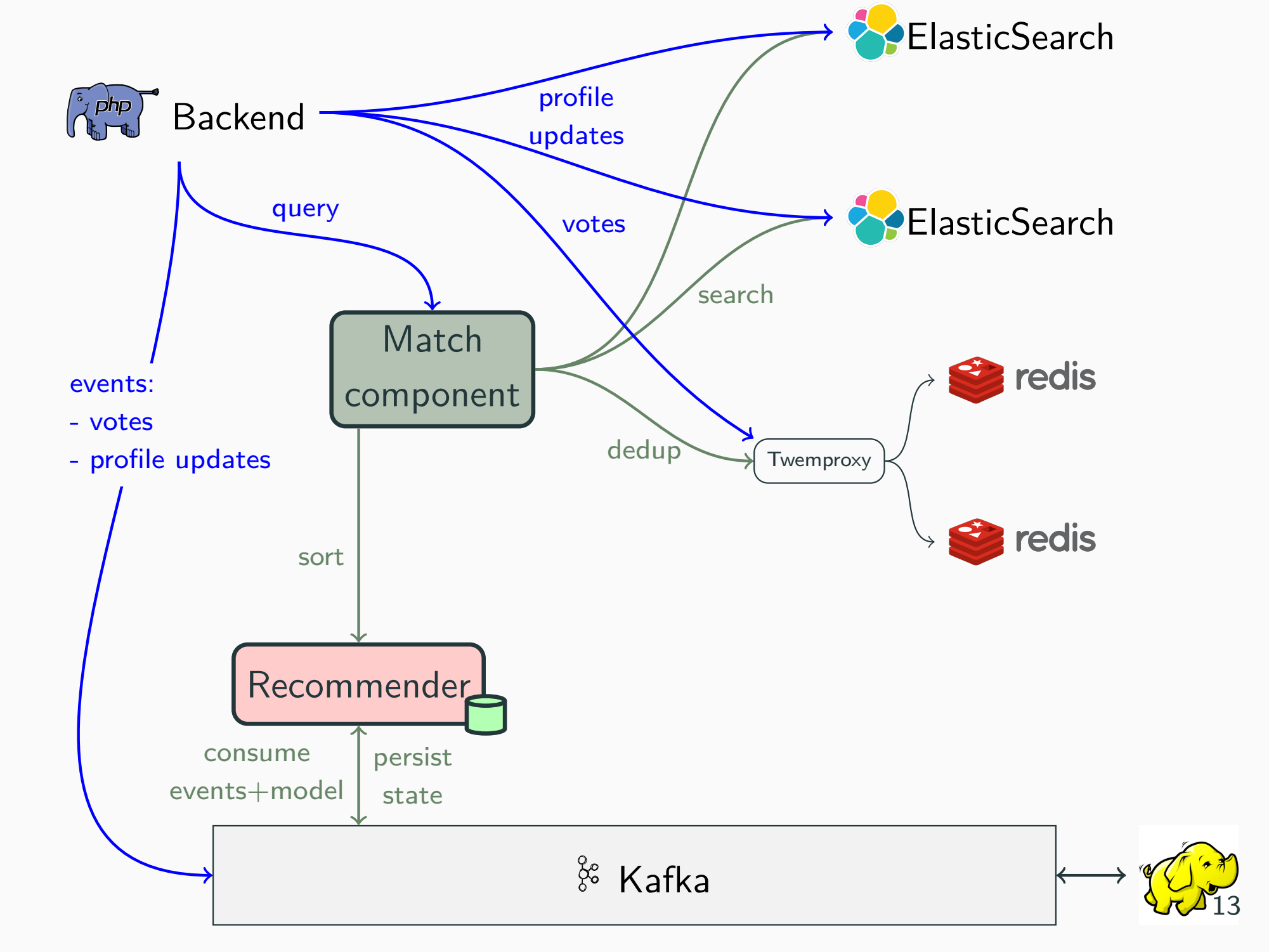

Backend
query
Match component
ElasticSearch
ElasticSearch
search
dedup
Twemproxy
redis
redis

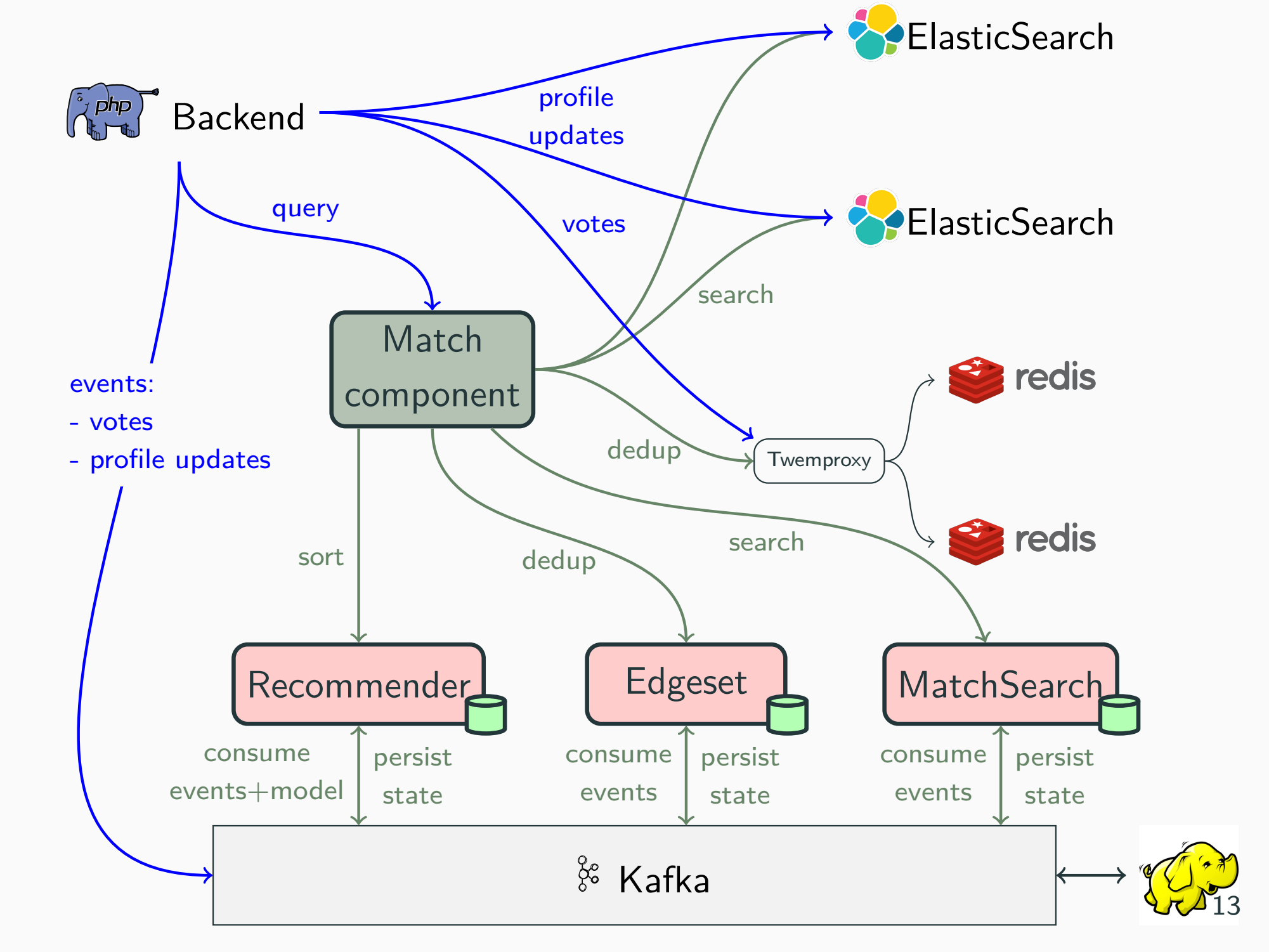

ElasticSearch
Backend
profile
updates
ElasticSearch
query
search
Match
component
redis
dedup
Twemproxy
redis

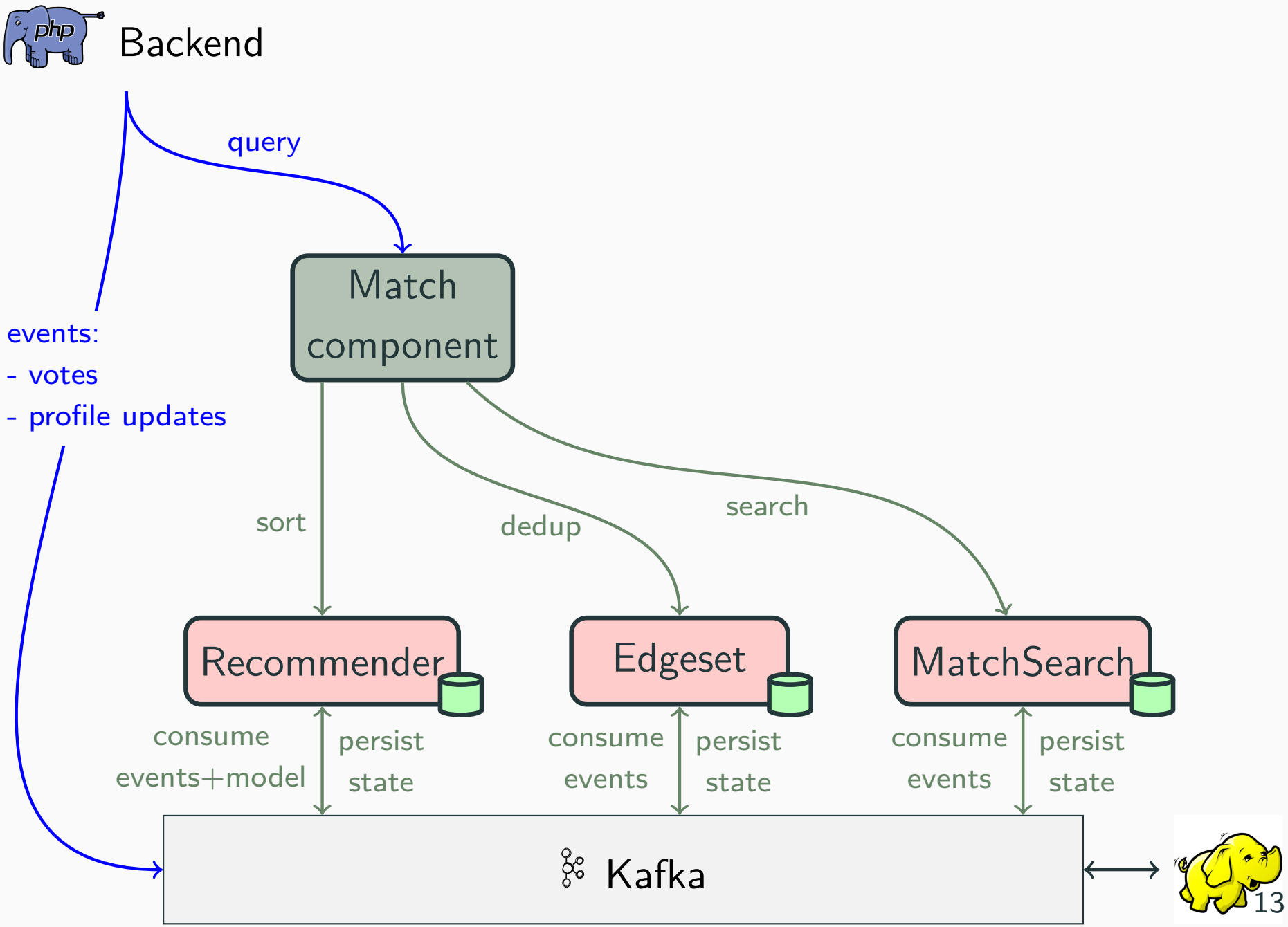

Backend
profile
updates
query
votes
search
ElasticSearch
ElasticSearch
Match
component
dedup
Twemproxy
redis
redis


ElasticSearch
Backend
profile
updates
ElasticSearch
query
votes
search
Match
component
events:
- votes
- profile updates
redis
dedup
Twemproxy
redis
NSQ


ElasticSearch
Backend
profile
updates
ElasticSearch
query
votes
search
Match
component
events:
- votes
- profile updates
redis
dedup
Twemproxy
redis
sort
load model
Recommender
persist
state
redis
consume
events
NSQ


ElasticSearch
Backend
profile
updates
ElasticSearch
query
votes
search
Match
component
events:
- votes
- profile updates
redis
dedup
Twemproxy
redis
sort
Recommender
consume
events+model
persist
state
Kafka

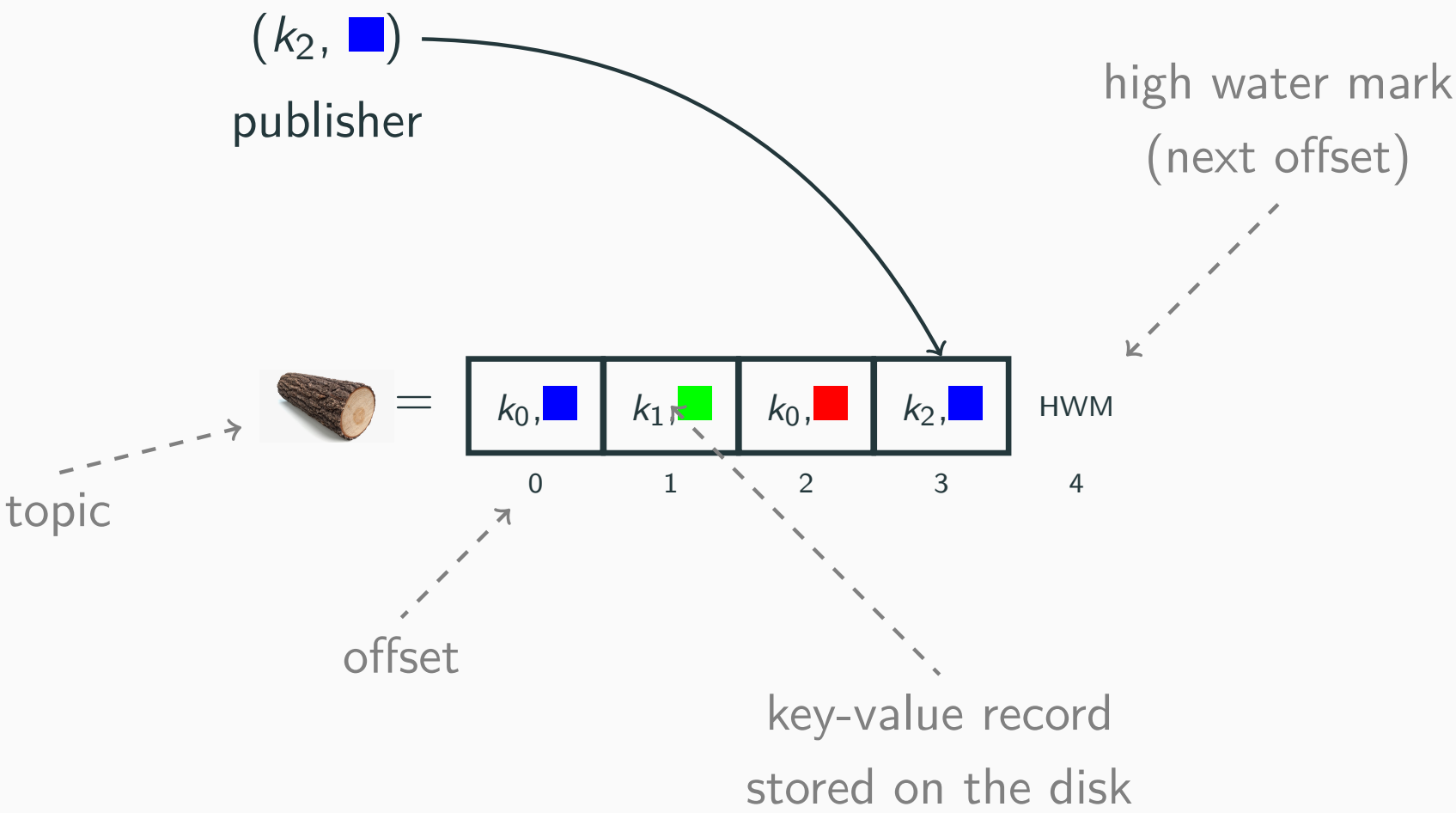

ElasticSearch
Backend
profile
updates
ElasticSearch
query
votes
search
Match
component
events:
- votes
- profile updates
redis
dedup
Twemproxy
redis
search
sort
dedup
Recommender
Edgeset
MatchSearch
consume
events+model
persist
state
consume
events
persist
state
consume
events
persist
state
Kafka


Backend
query
events:
- votes
- profile updates
Match component
sort
dedup
search
Recommender
Edgeset
MatchSearch
consume events+model
persist state
consume events
persist state
consume events
persist state
Kafka

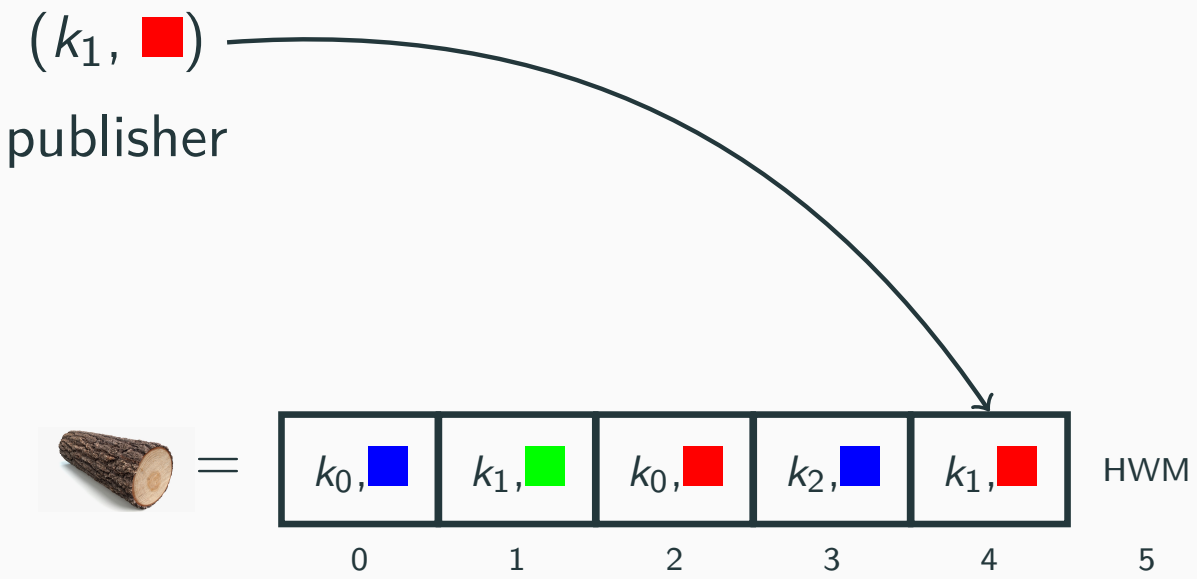
Kafka

topics, partitions, streams and tables

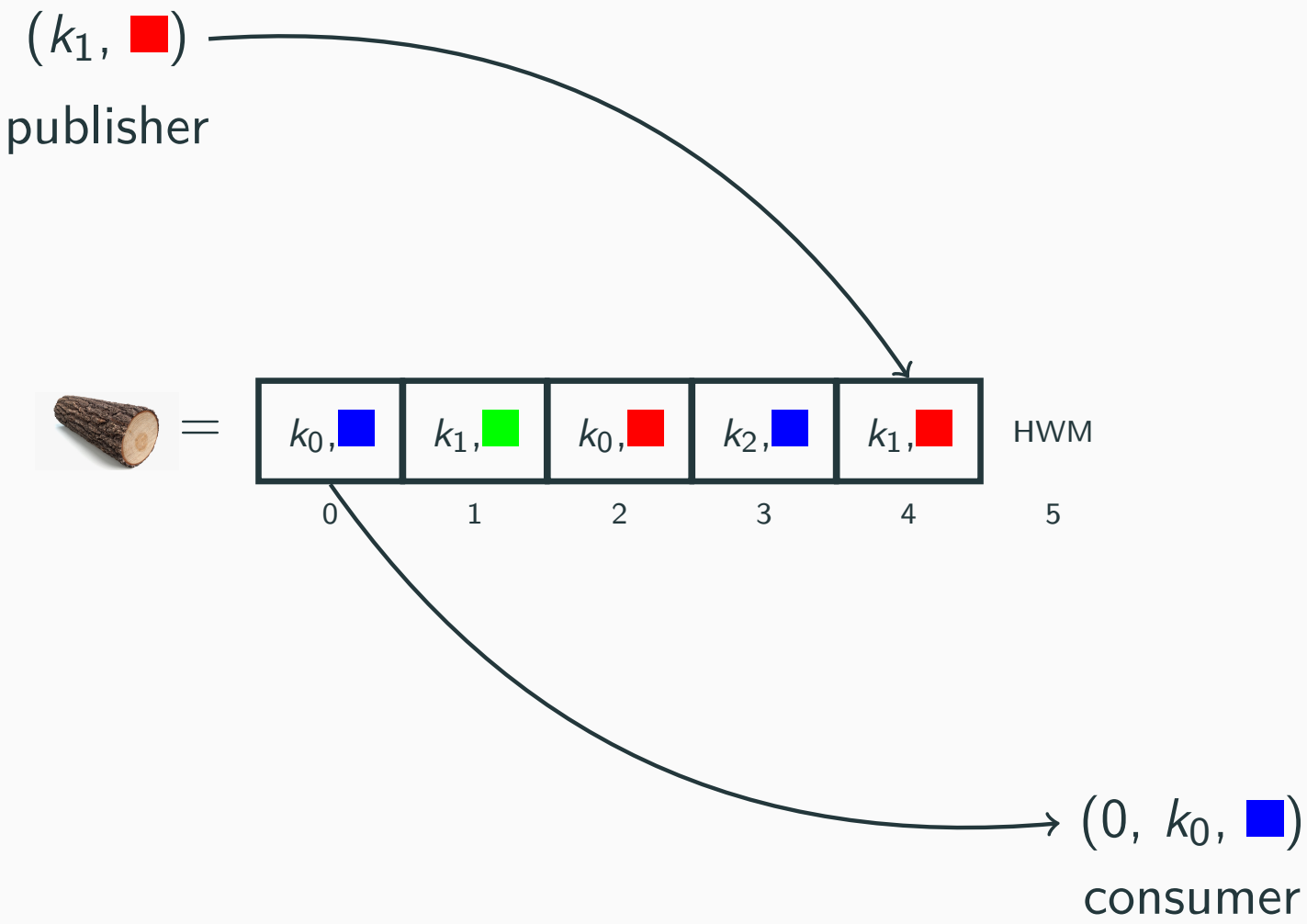
Kafka: Basics



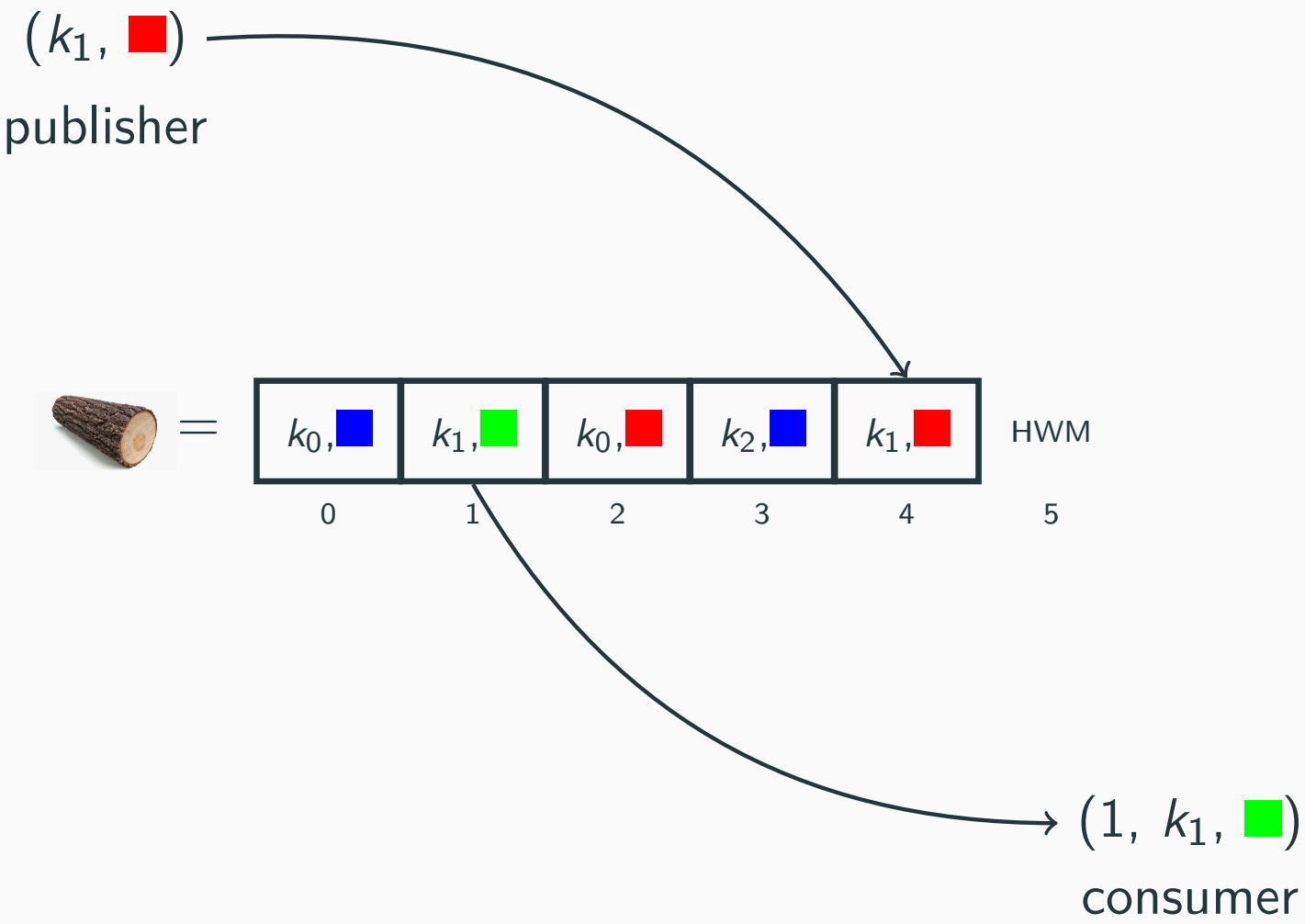
Kafka: Basics



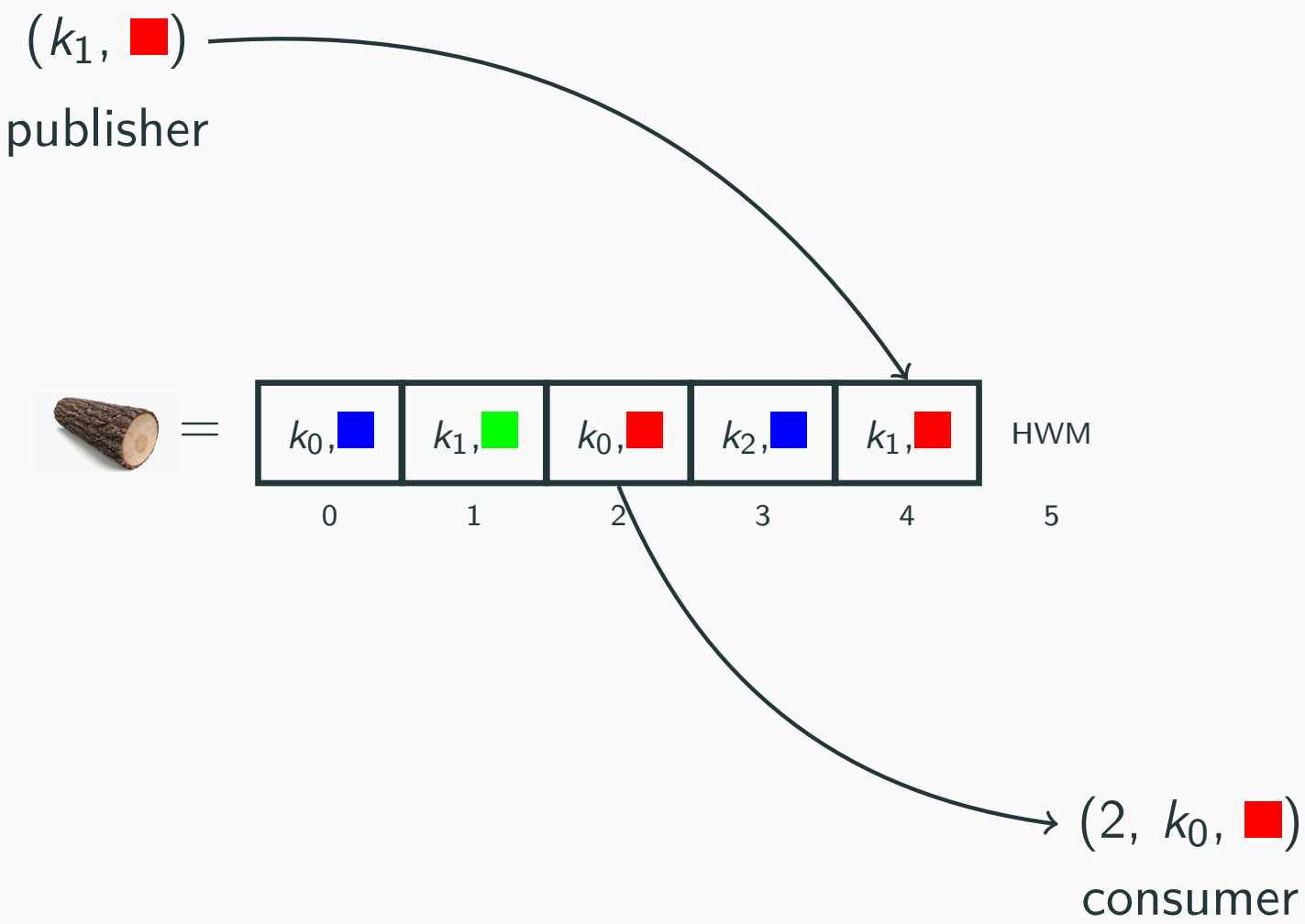
Kafka: Basics



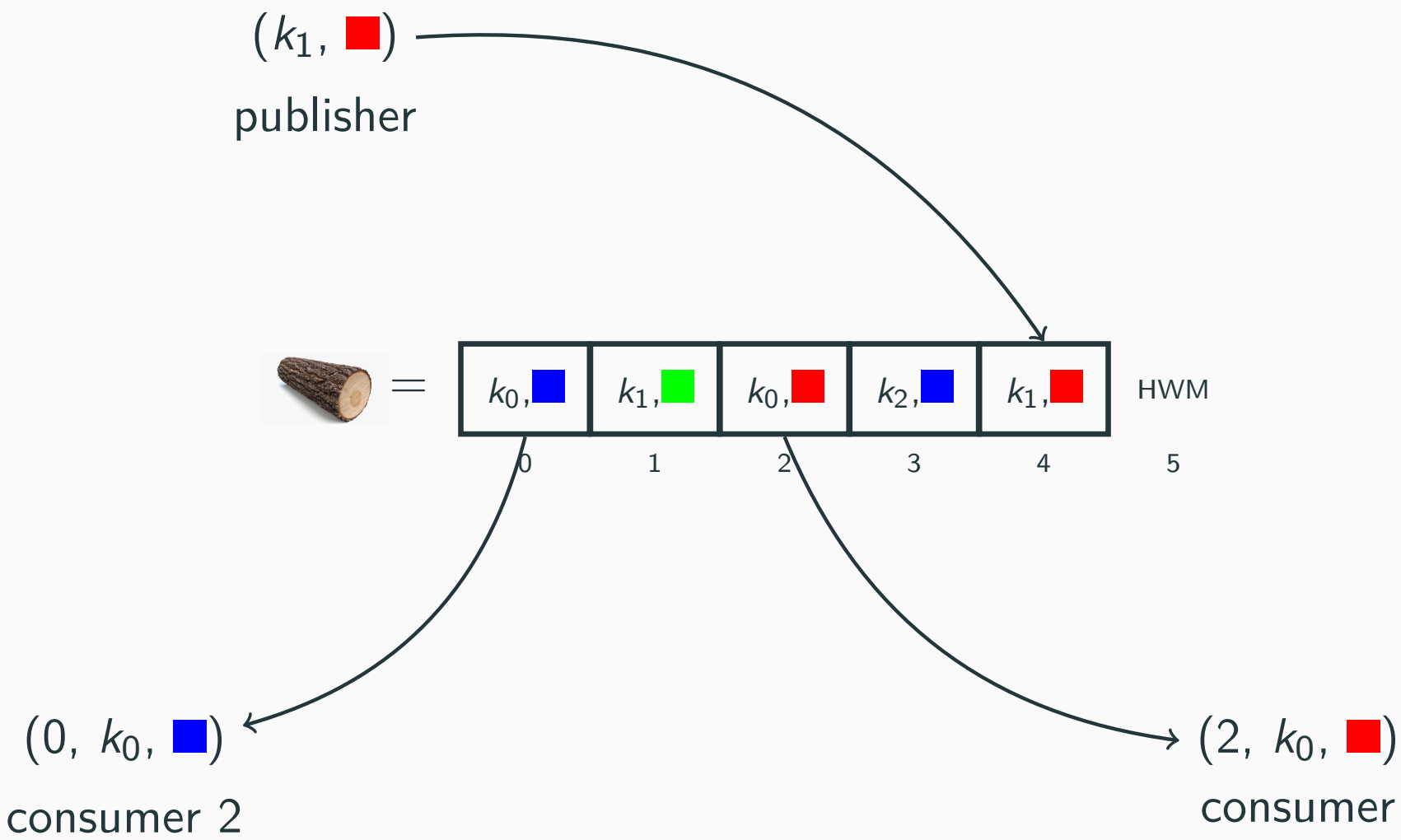
Kafka: Basics



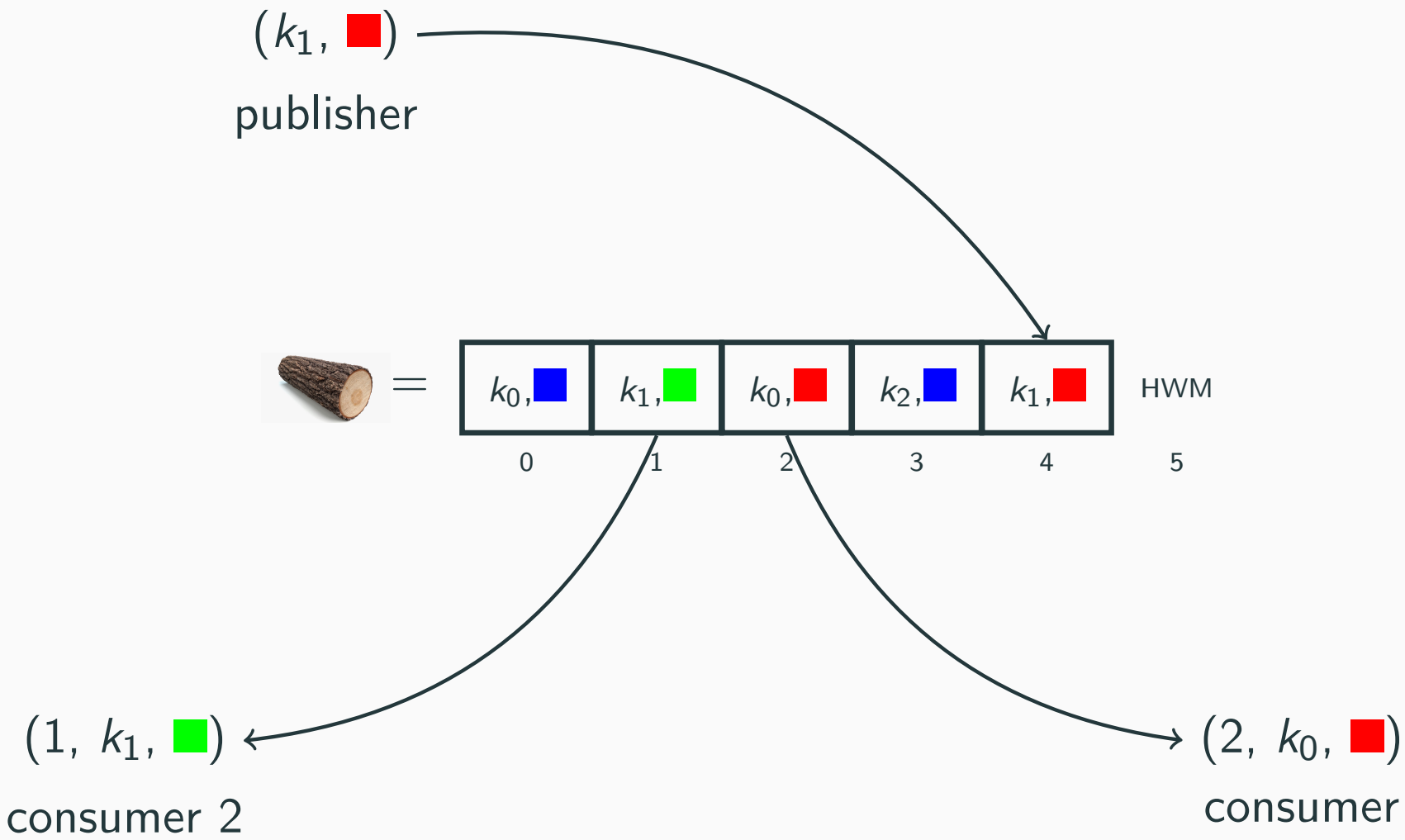
Kafka: Basics



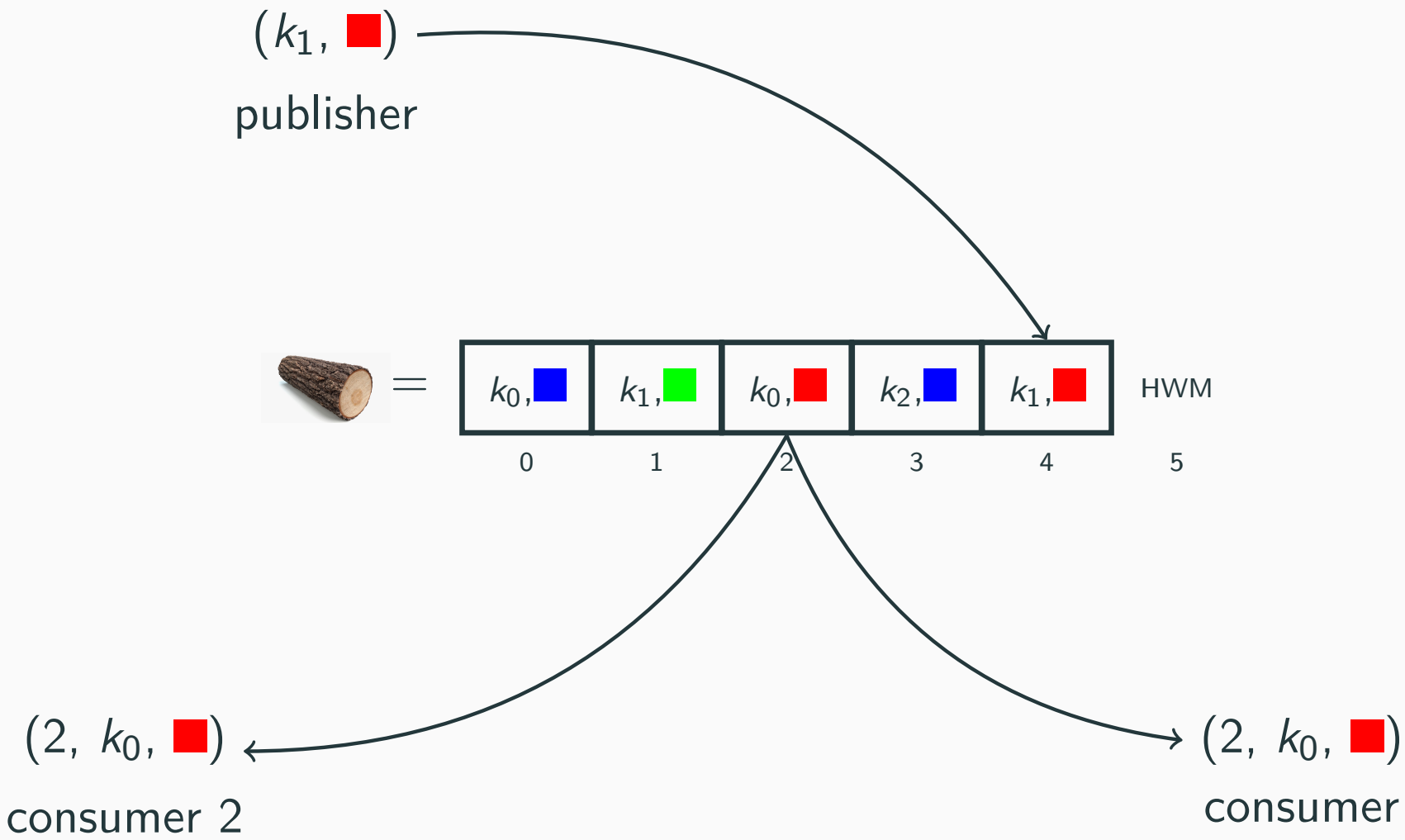
Kafka: Basics



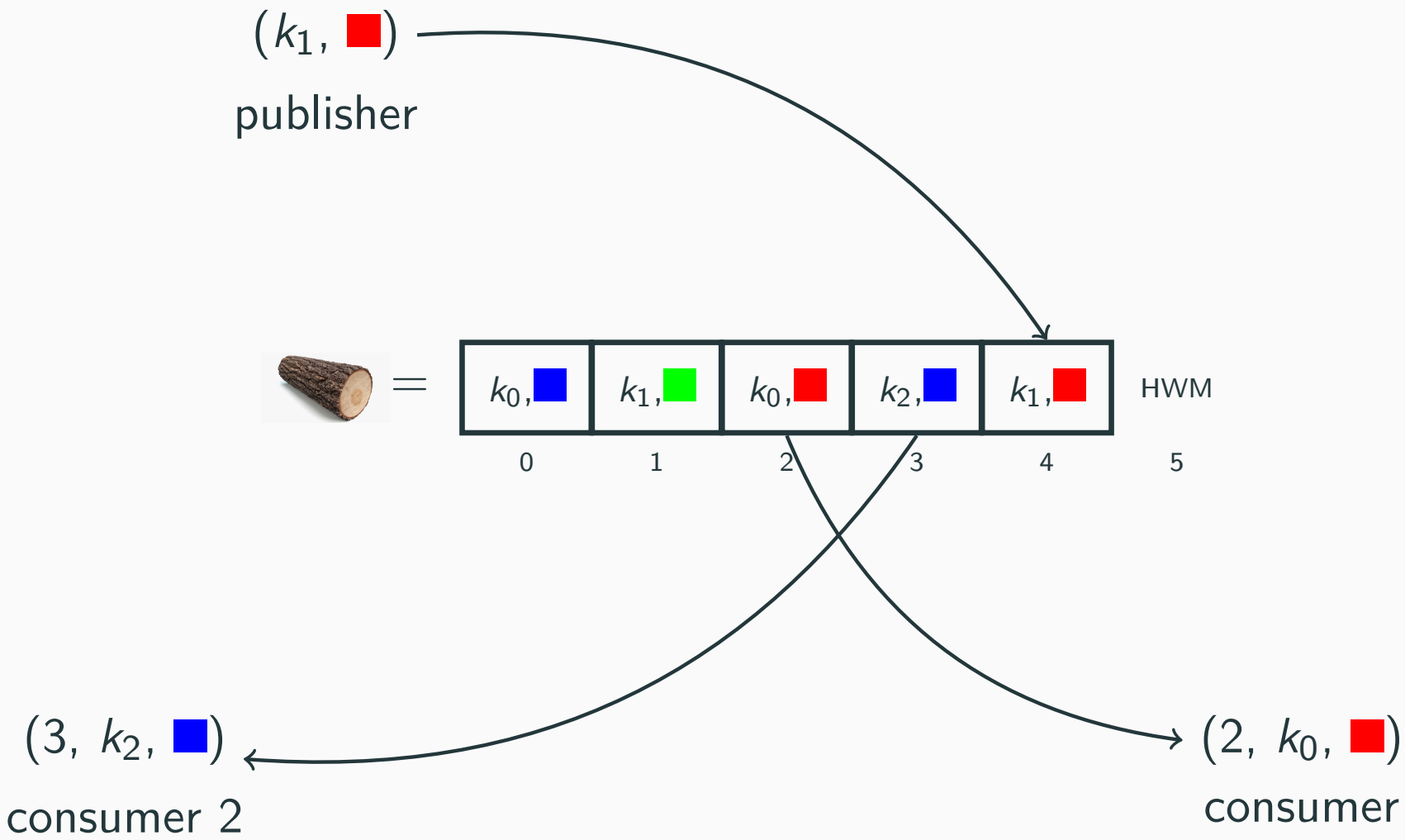
Kafka: Basics



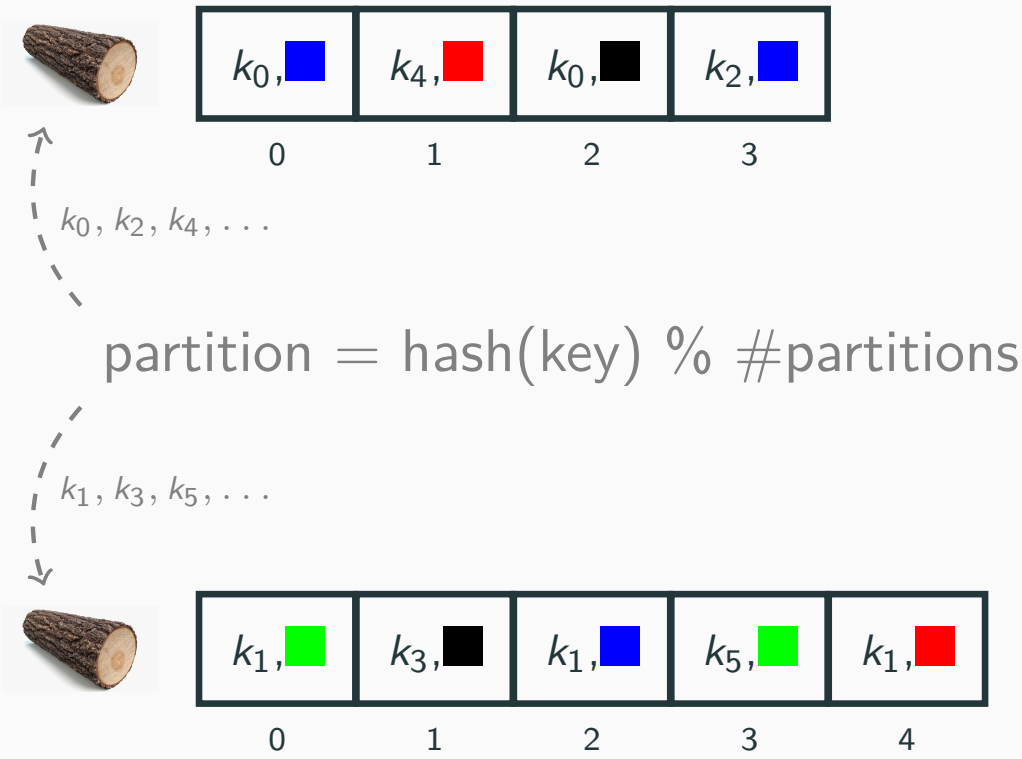
Kafka: Basics



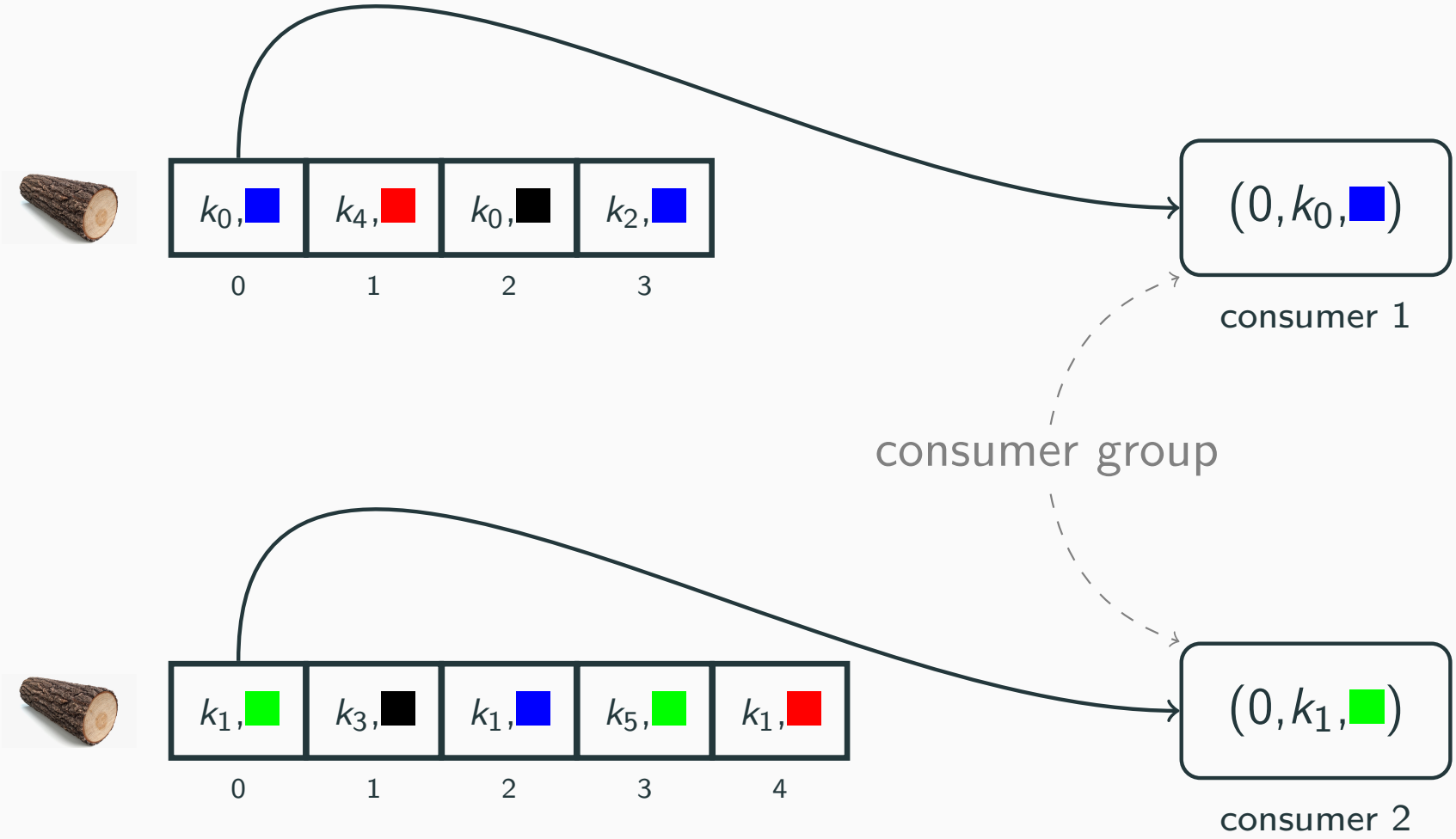
Kafka: Basics



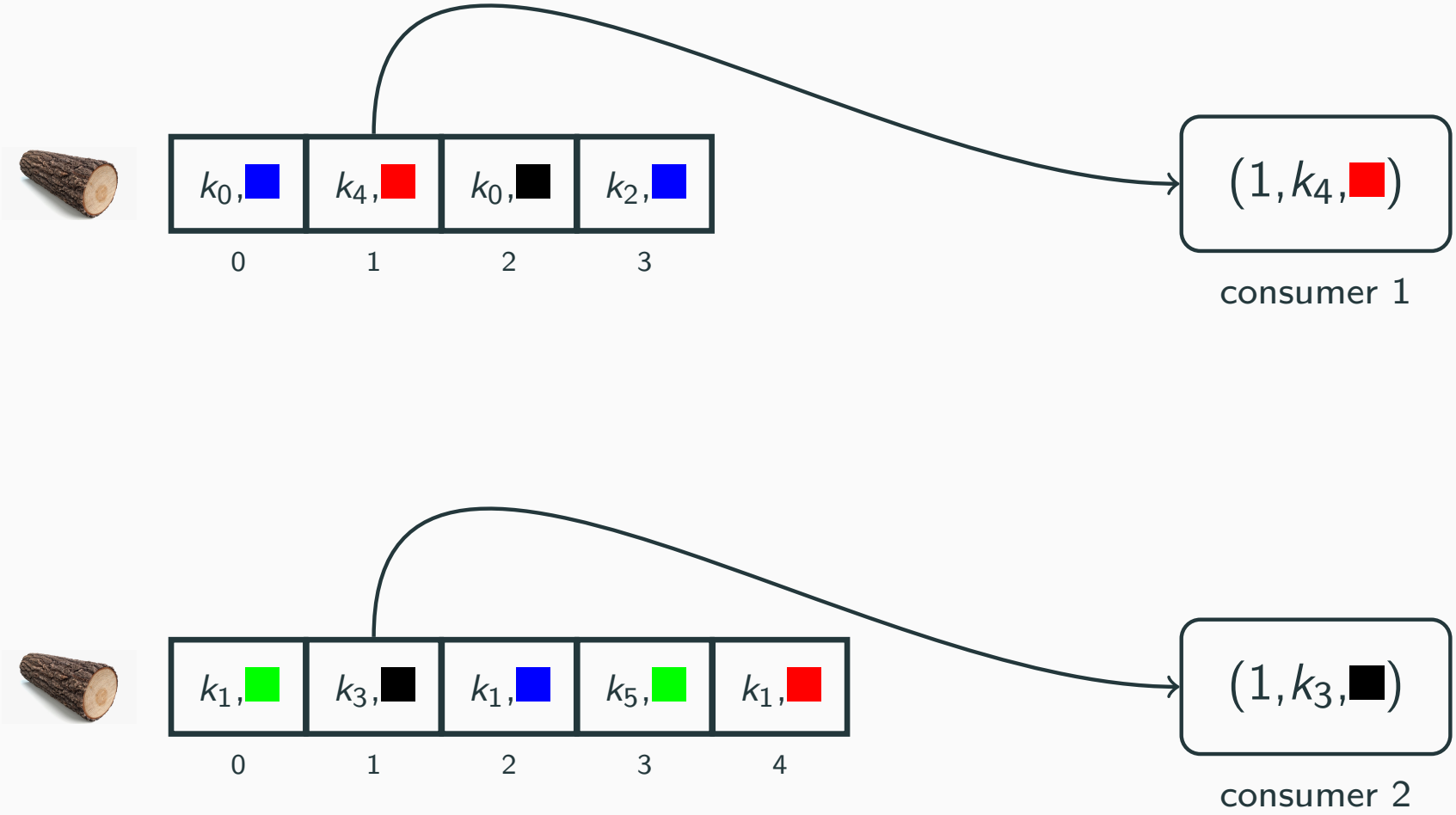
Kafka: Scalability and Fault Tolerance



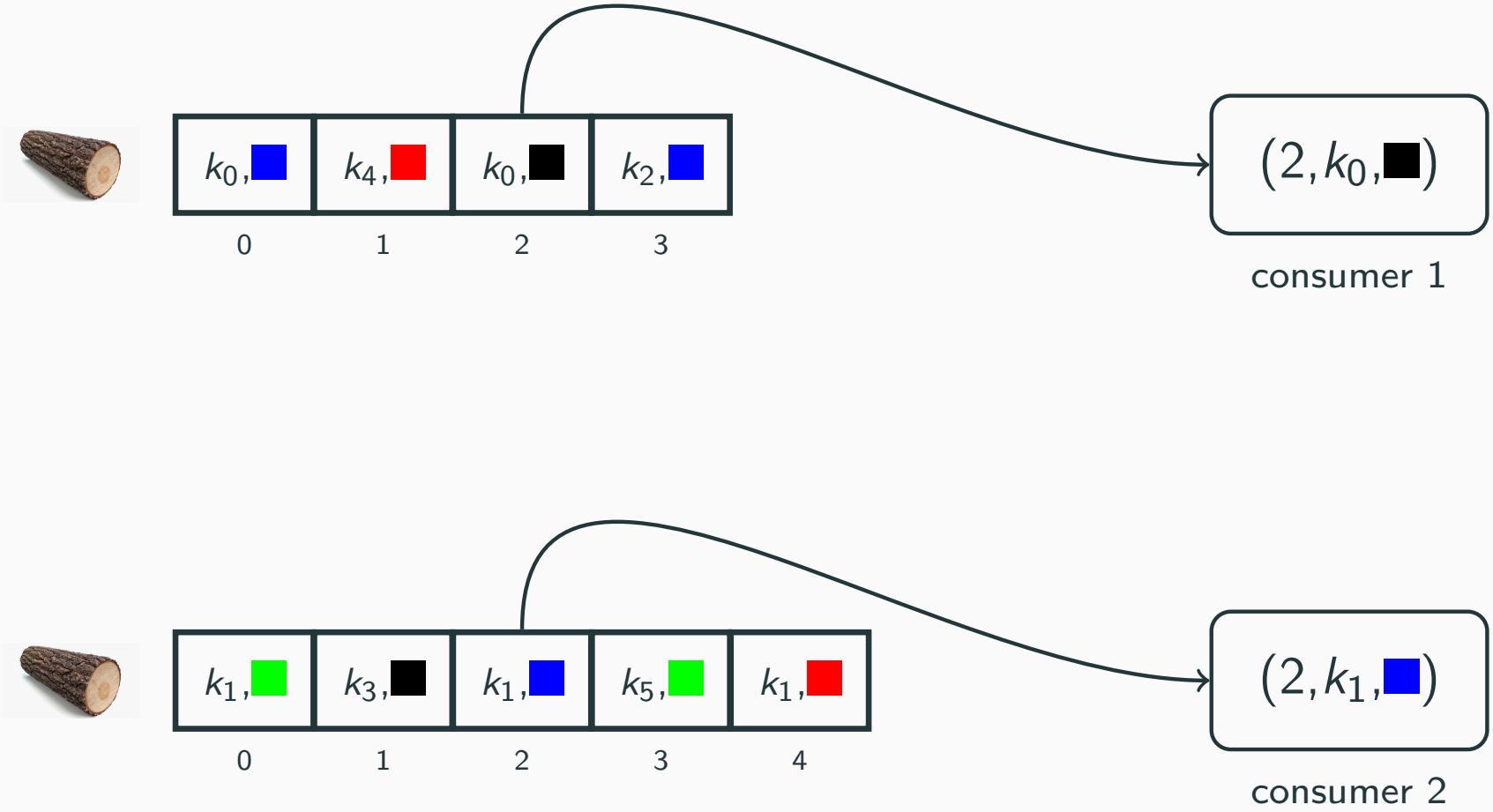
Kafka: Scalability and Fault Tolerance



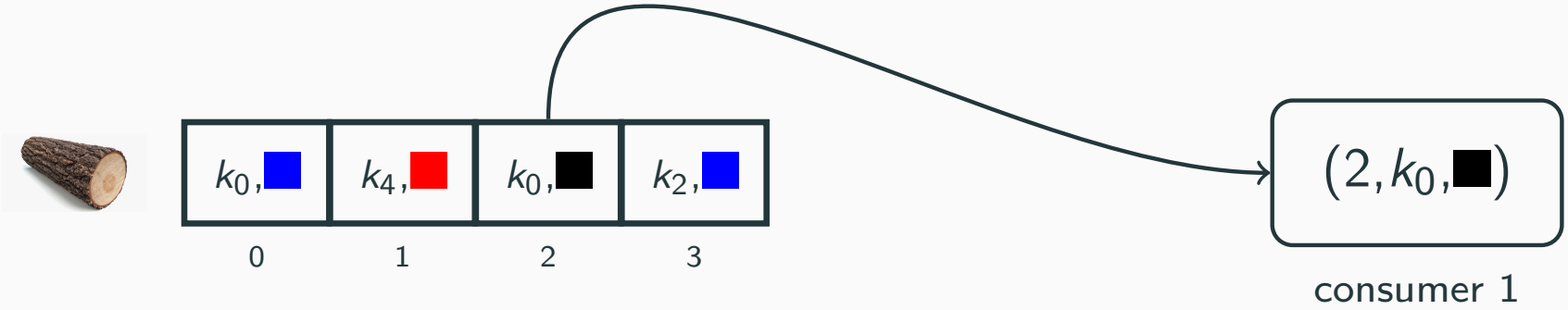
Kafka: Scalability and Fault Tolerance

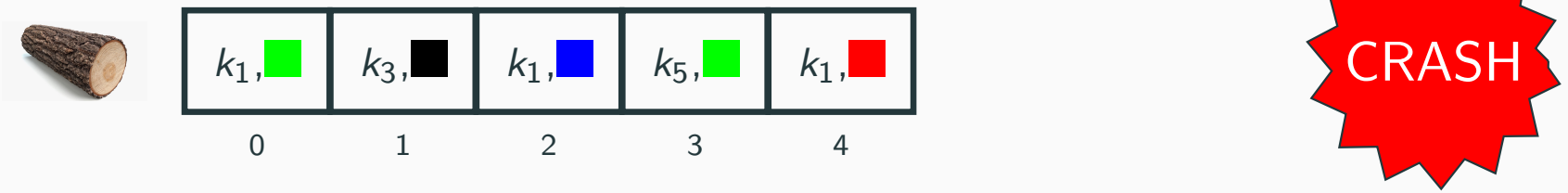


Kafka: Scalability and Fault Tolerance

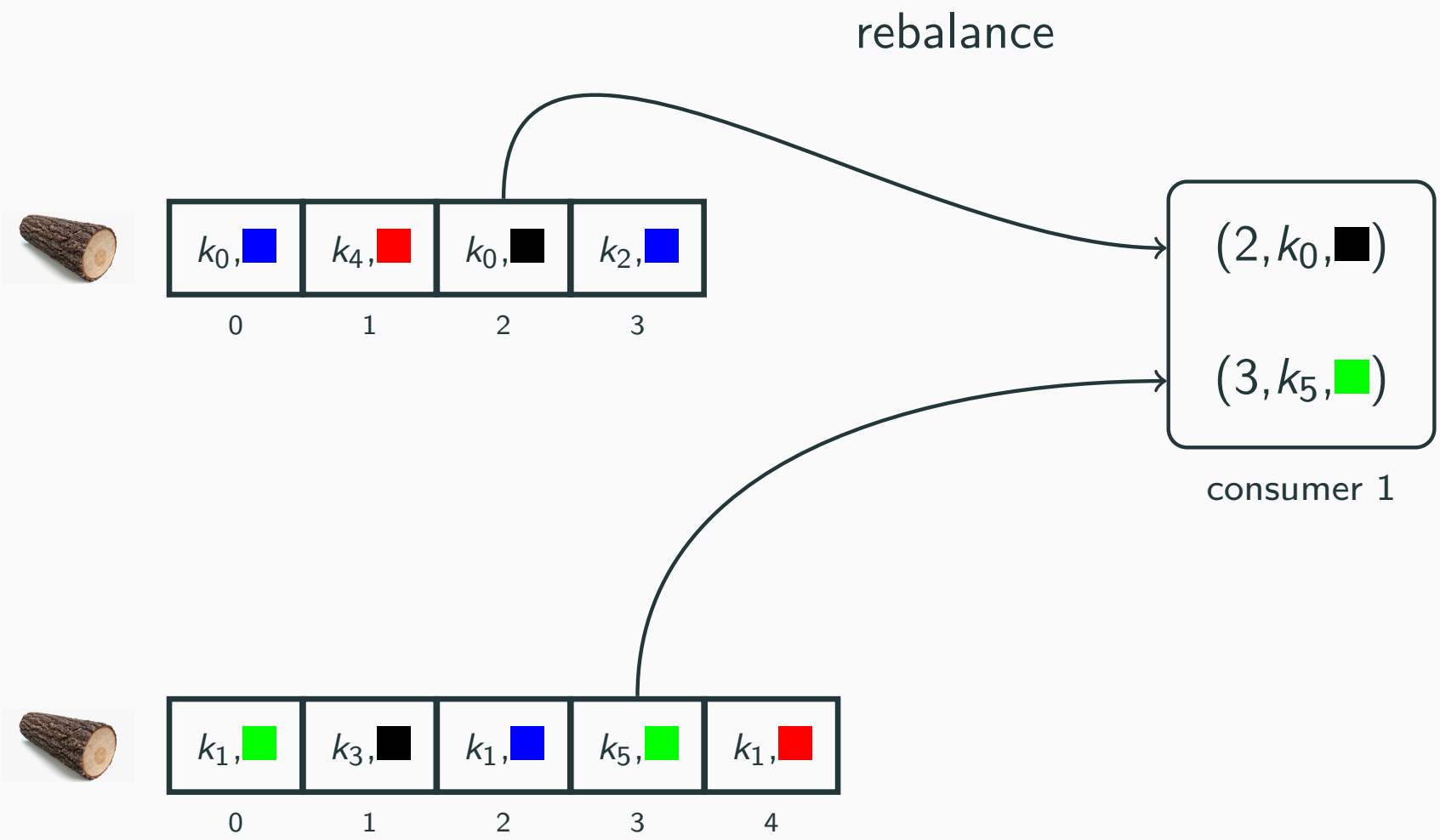


Kafka: Scalability and Fault Tolerance

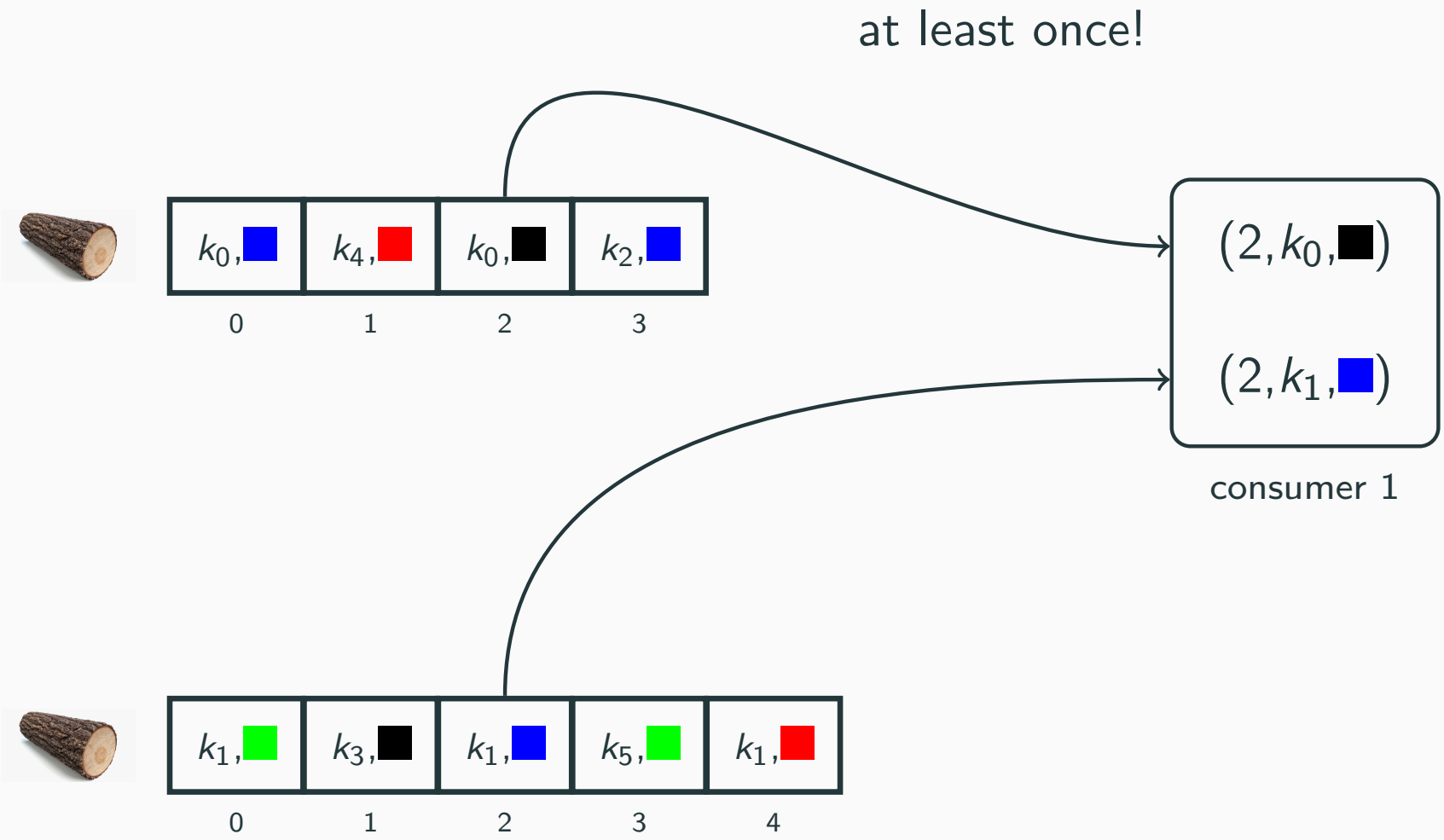




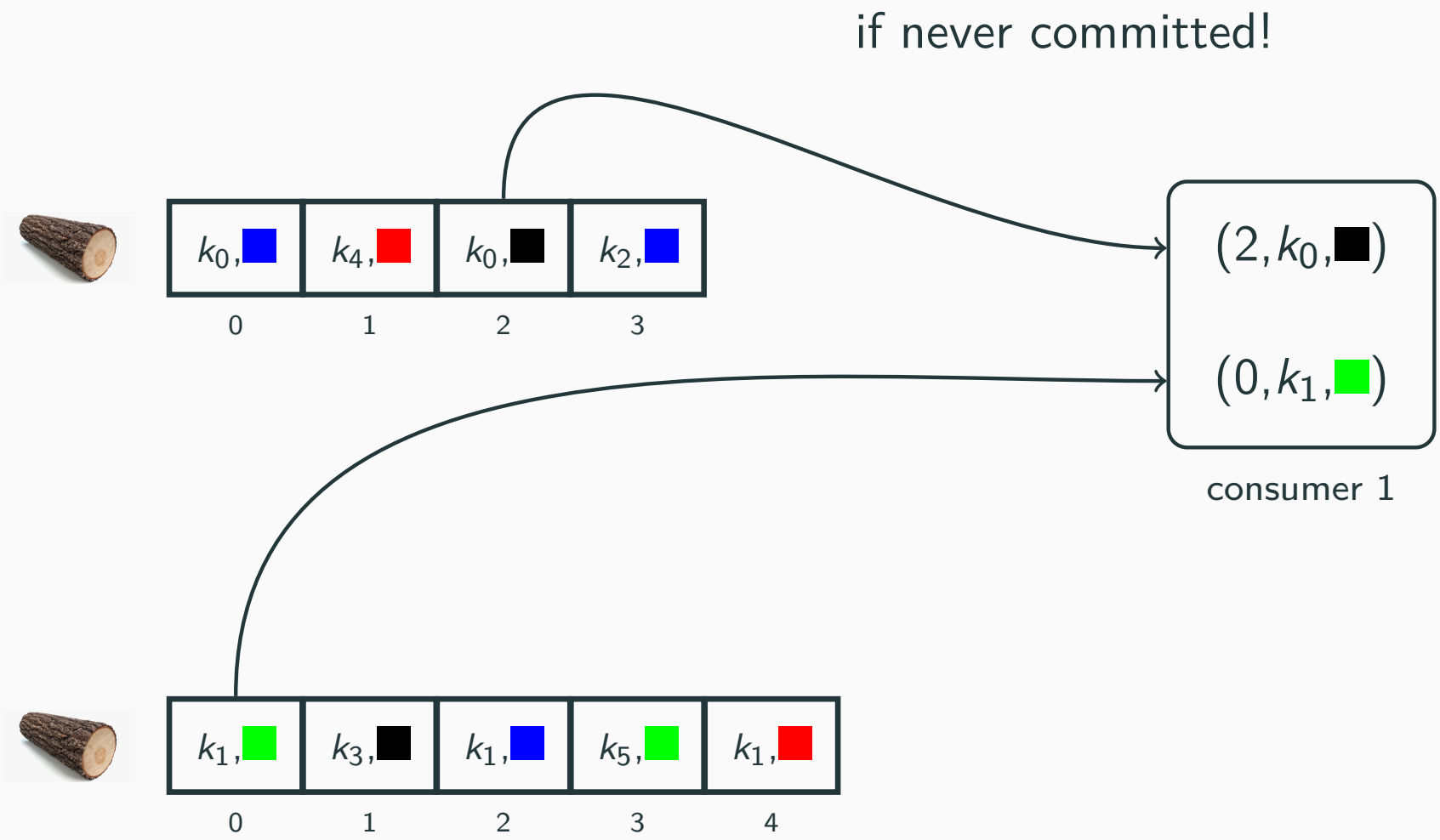
Kafka: Scalability and Fault Tolerance



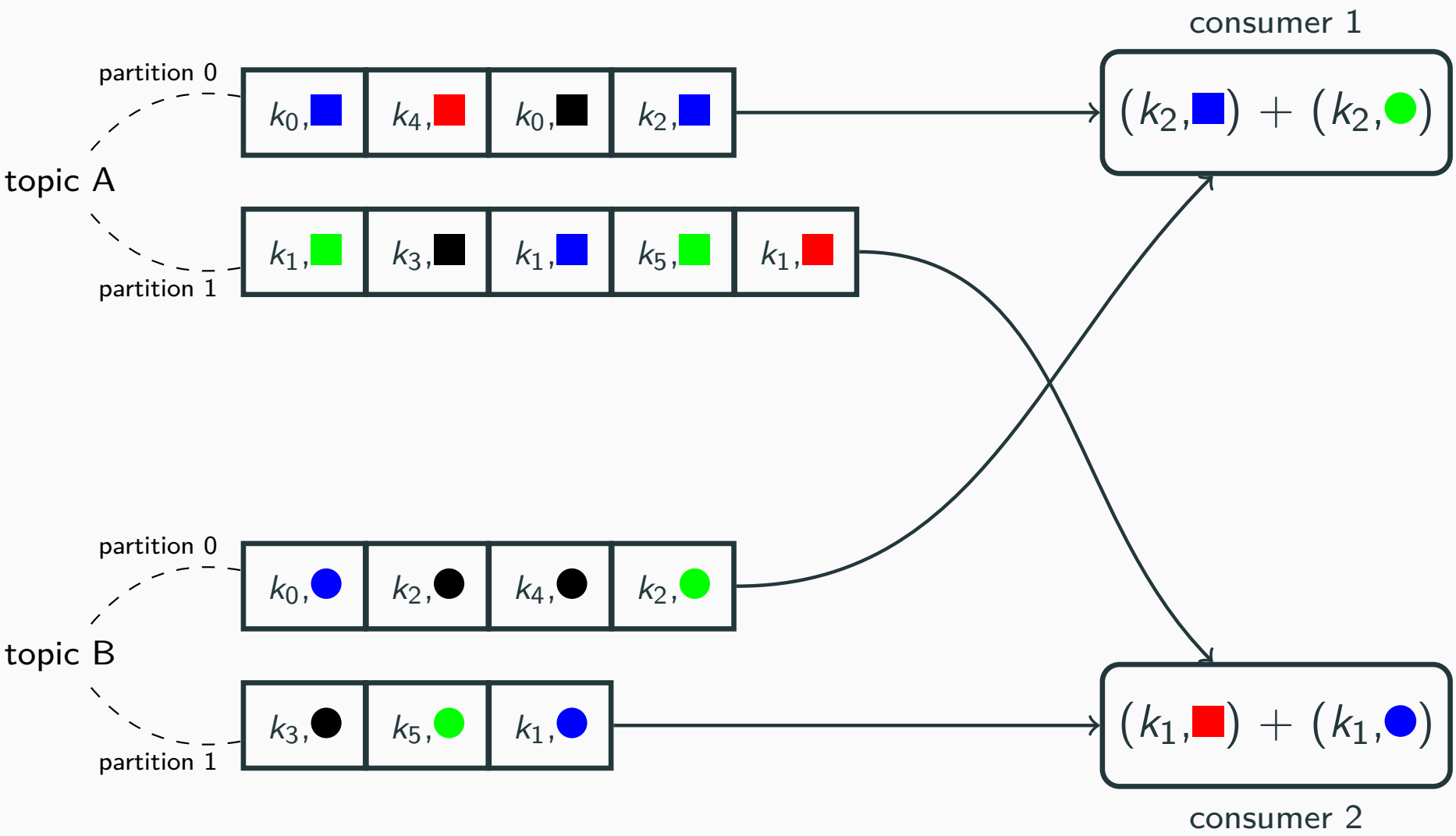
Kafka: Scalability and Fault Tolerance



Kafka: Scalability and Fault Tolerance

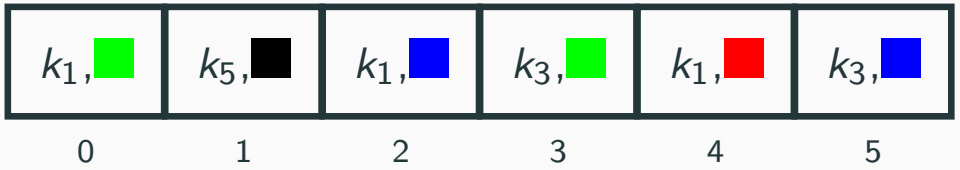


Kafka: Copartitioning



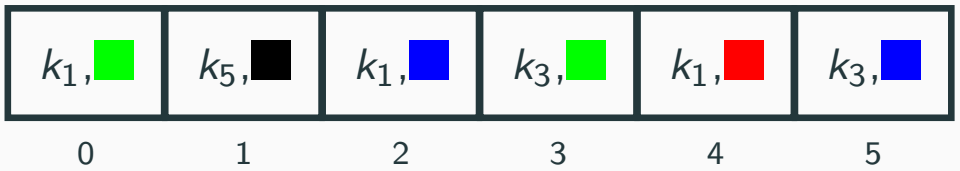
Kafka: Cleanup Policies

delete after retention time



Stream

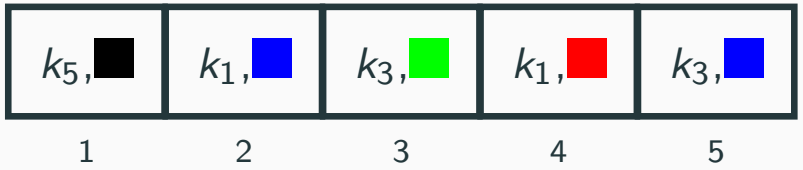
compact, leaving latest value for each key



Table

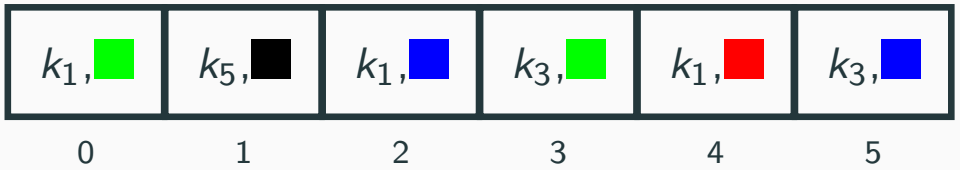
Kafka: Cleanup Policies

delete after retention time



Stream

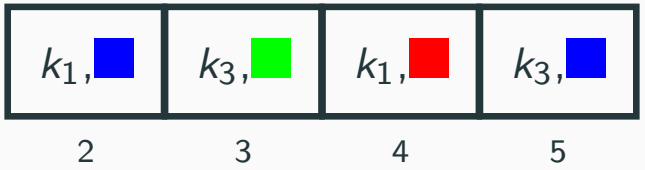
compact, leaving latest value for each key



Table

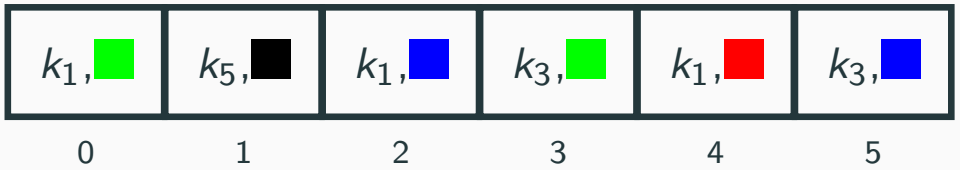
Kafka: Cleanup Policies

delete after retention time



Stream

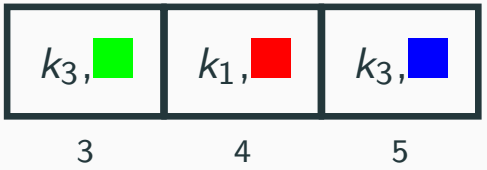
compact, leaving latest value for each key



Table

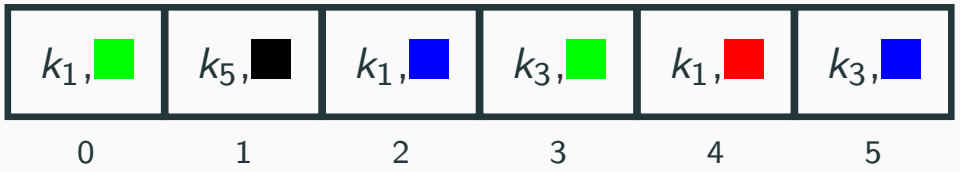
Kafka: Cleanup Policies

delete after retention time



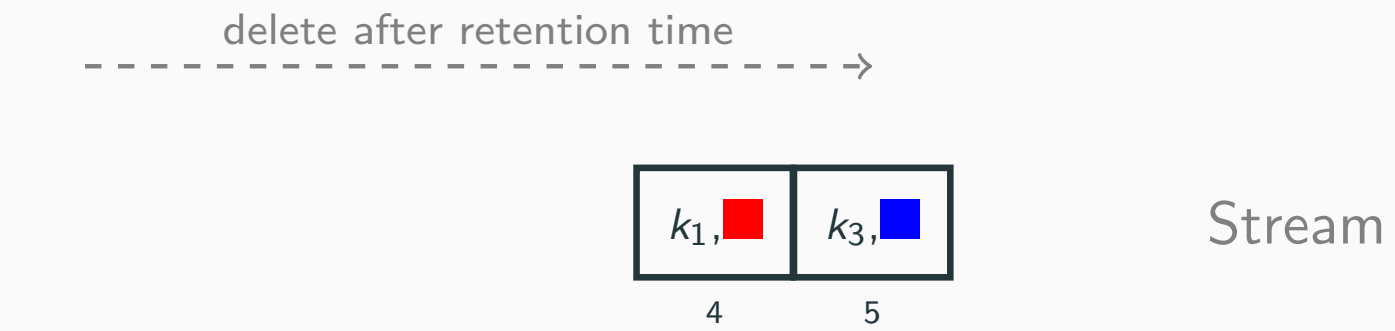
Stream

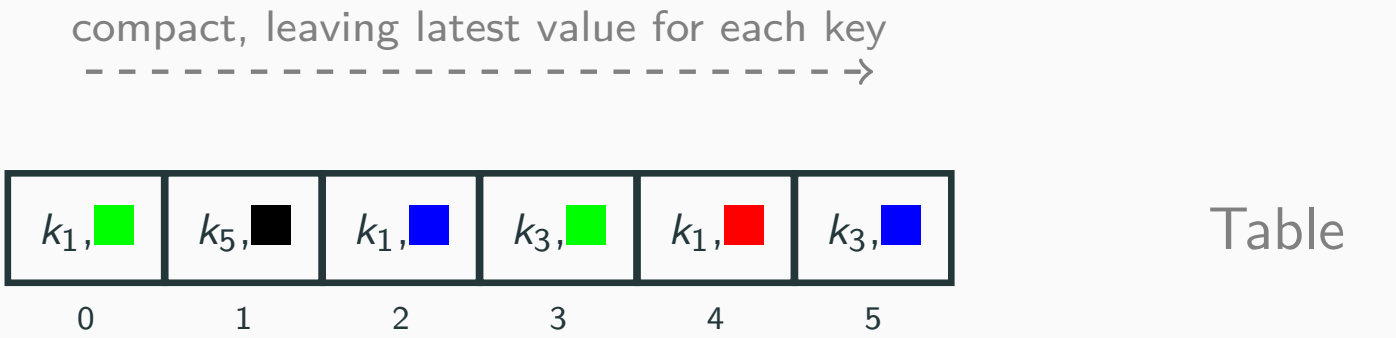
compact, leaving latest value for each key



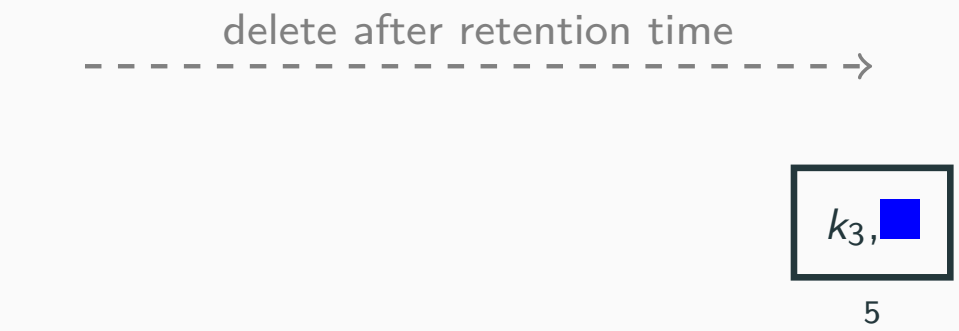
Table

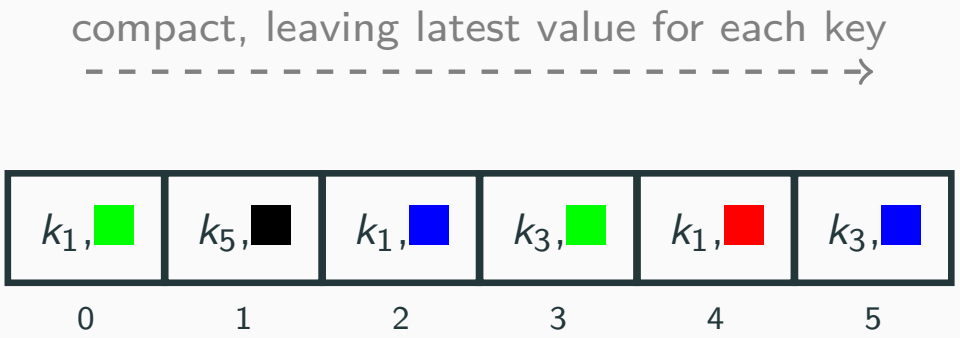
Kafka: Cleanup Policies



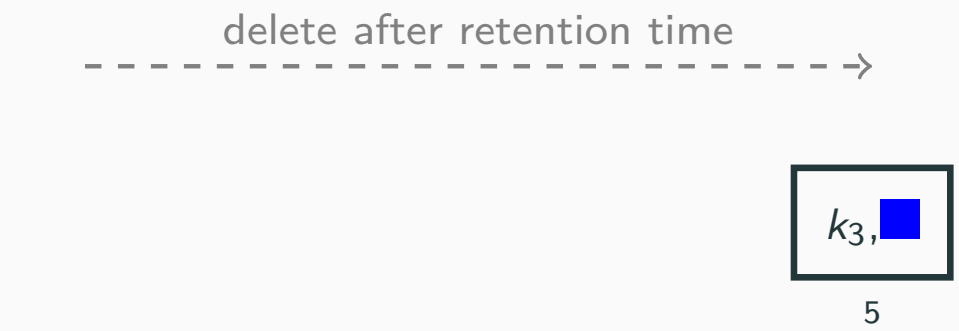


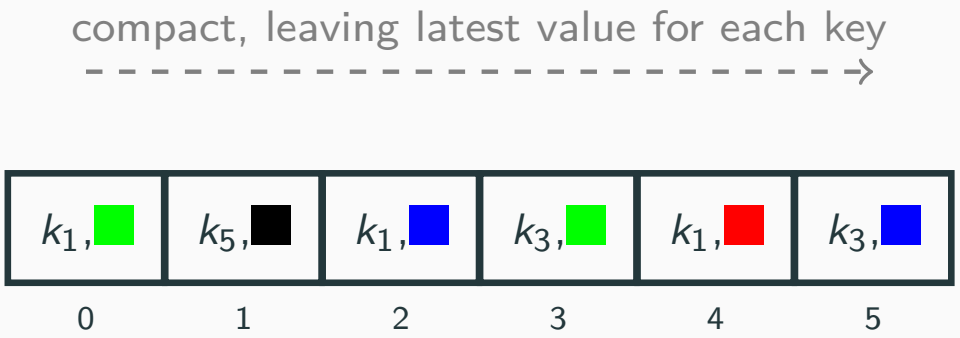
Kafka: Cleanup Policies



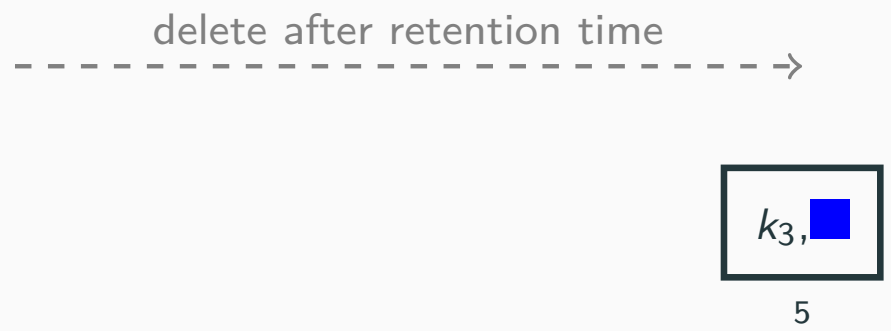


Kafka: Cleanup Policies

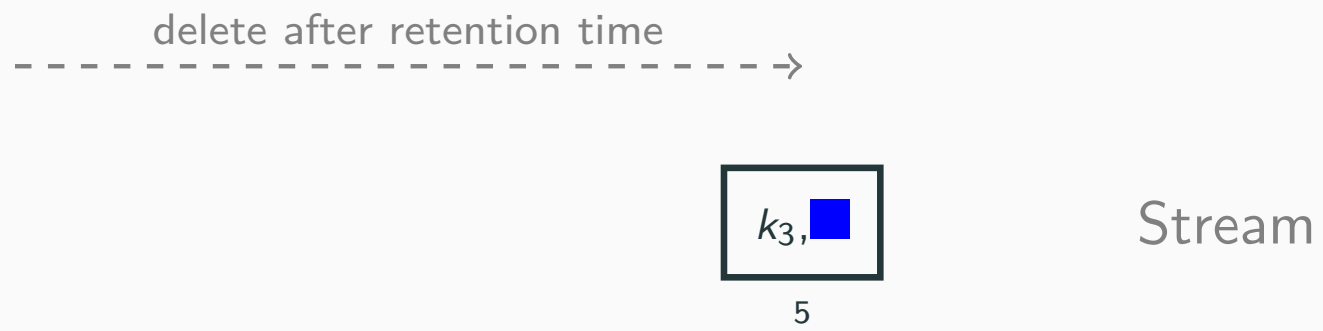


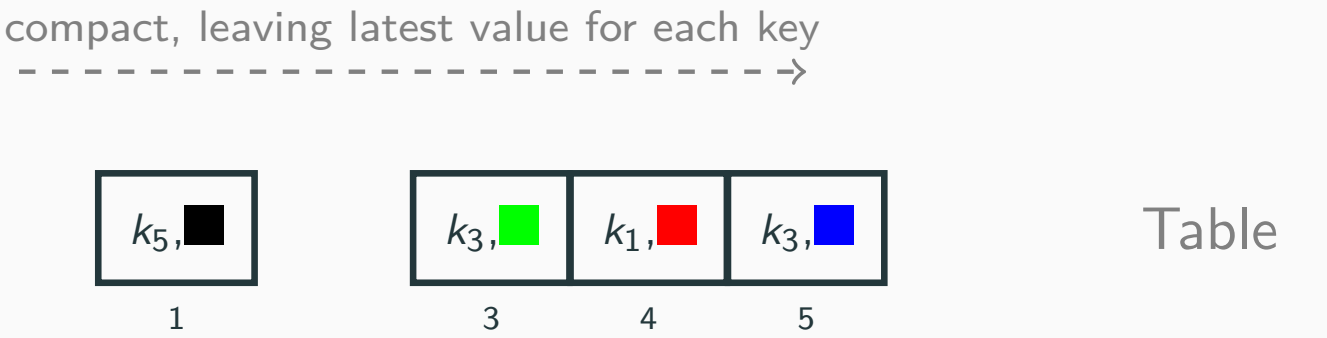


Kafka: Cleanup Policies

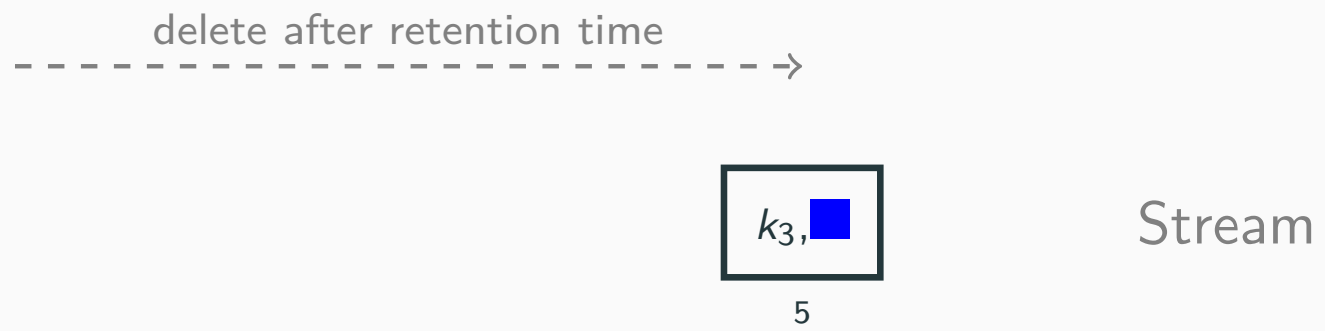


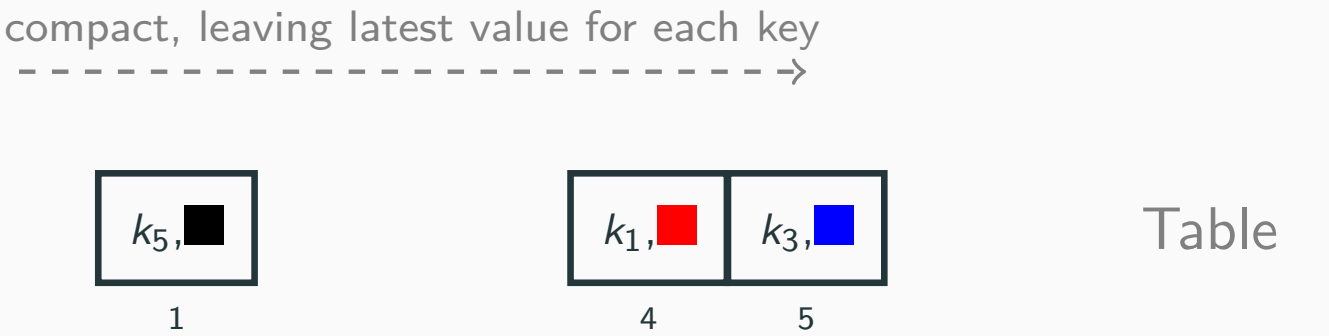
Kafka: Cleanup Policies





Kafka: Cleanup Policies





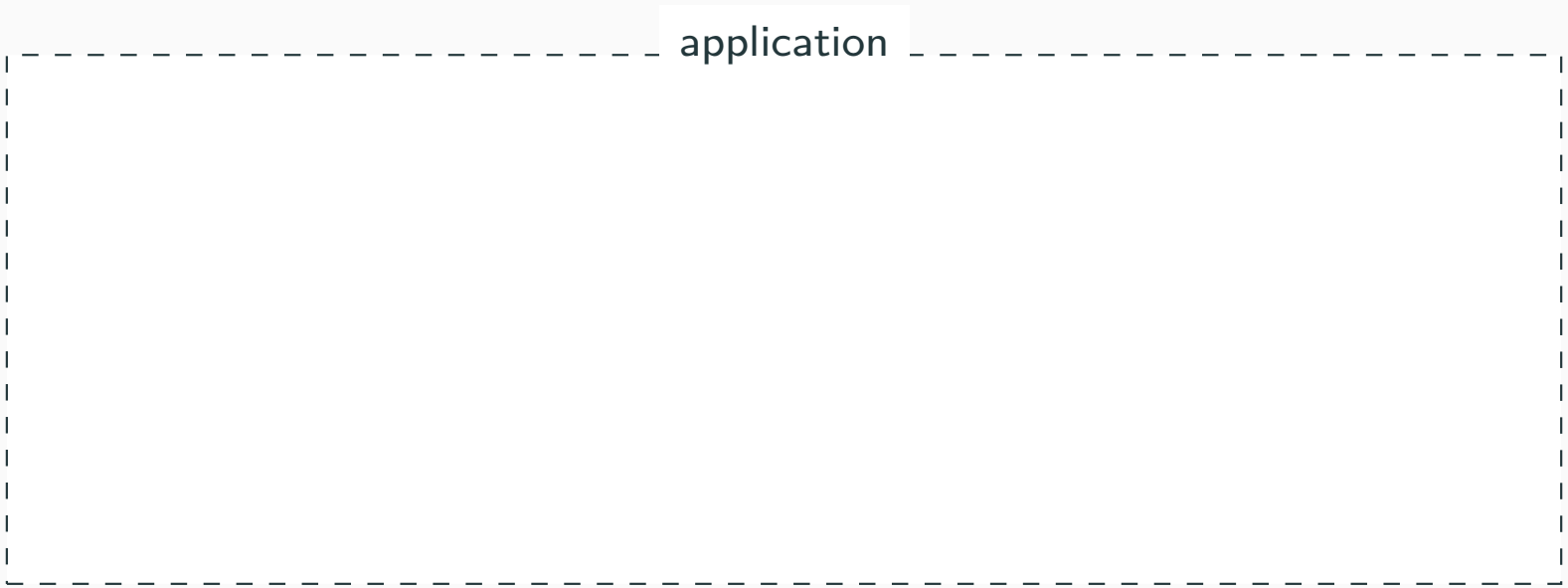
And now for something completely different...

Hands-on session!

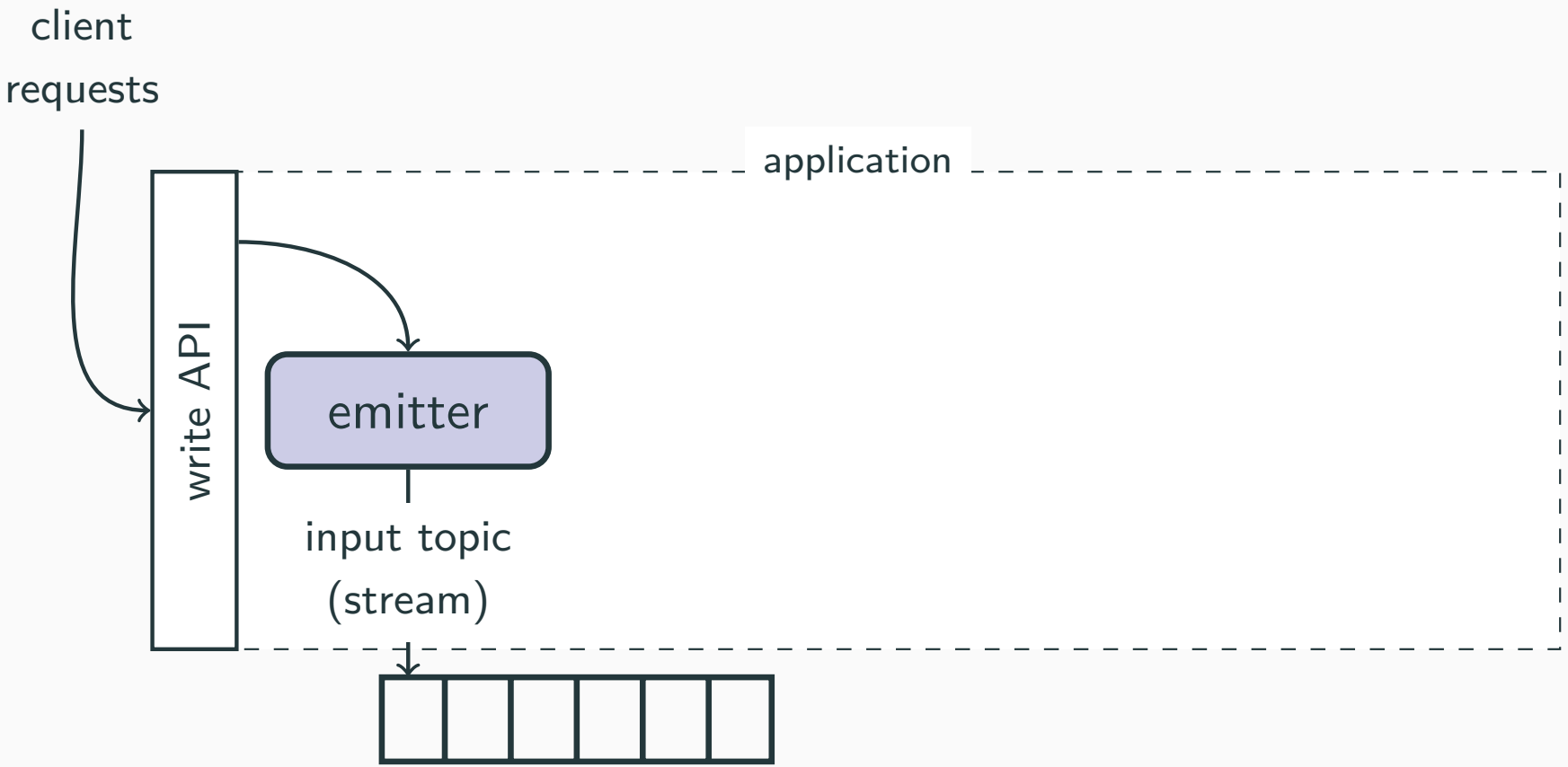
Learning Goka the “hard way”

components, code examples, and tasks

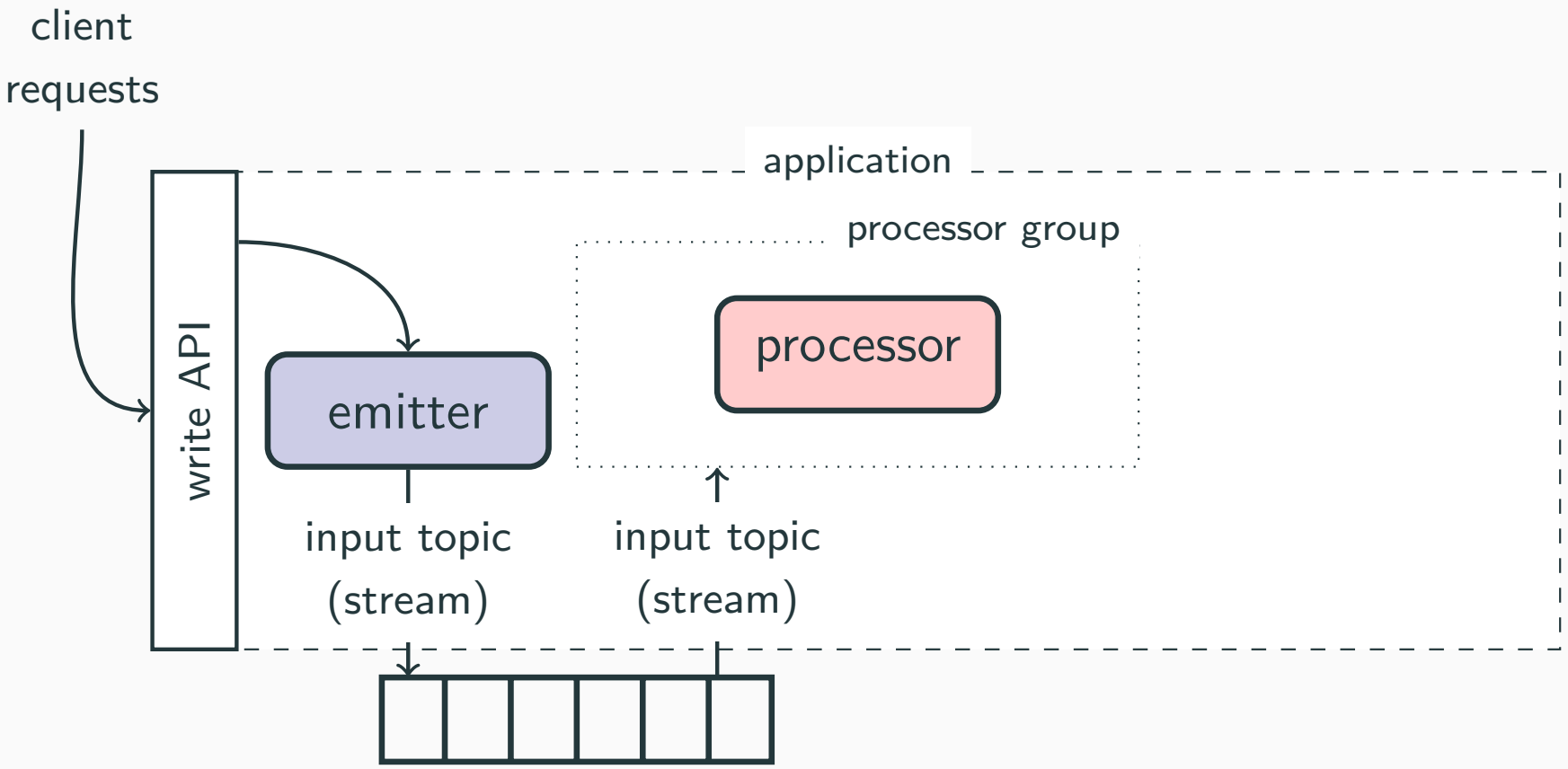
Goka Components



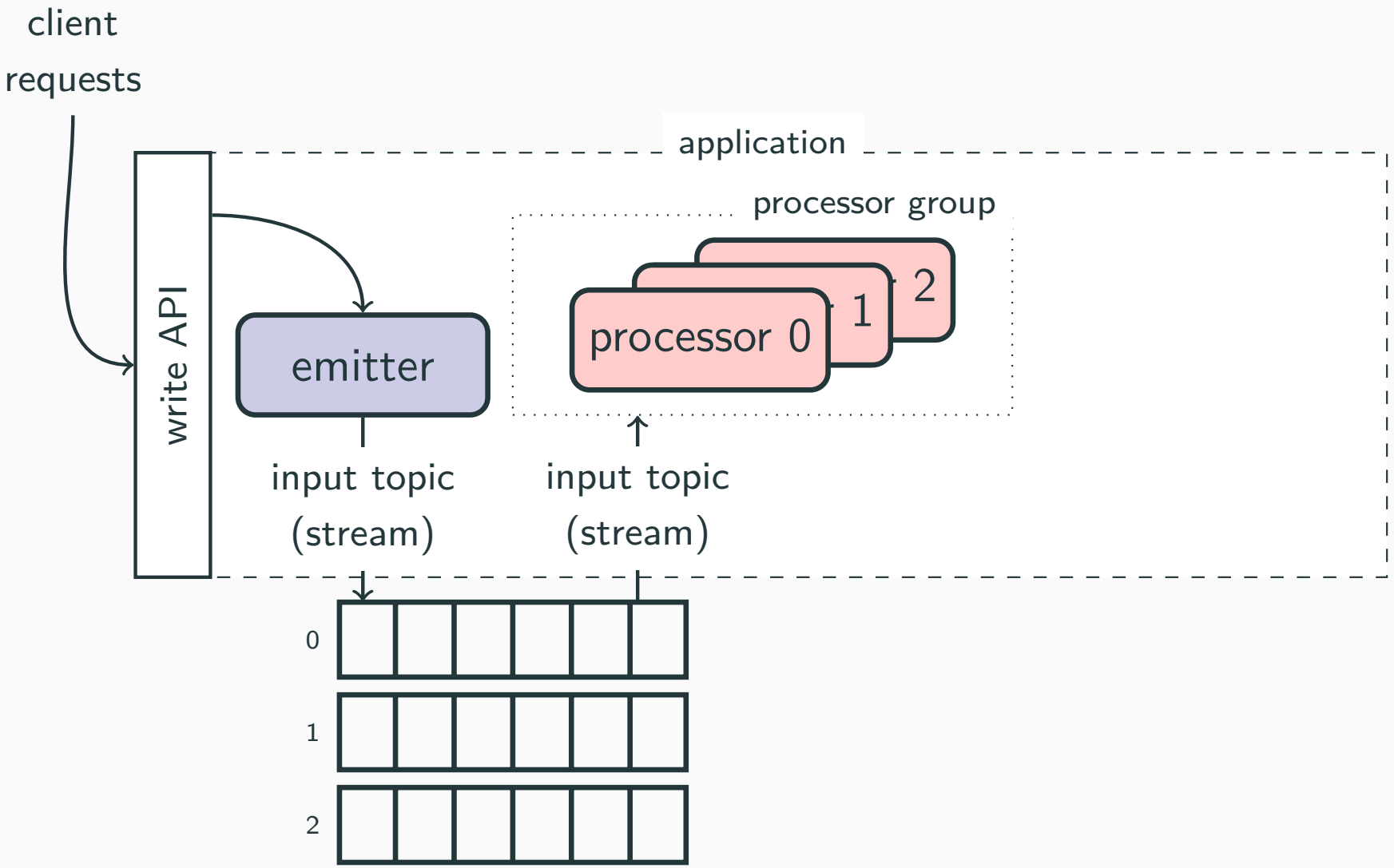
Goka Components



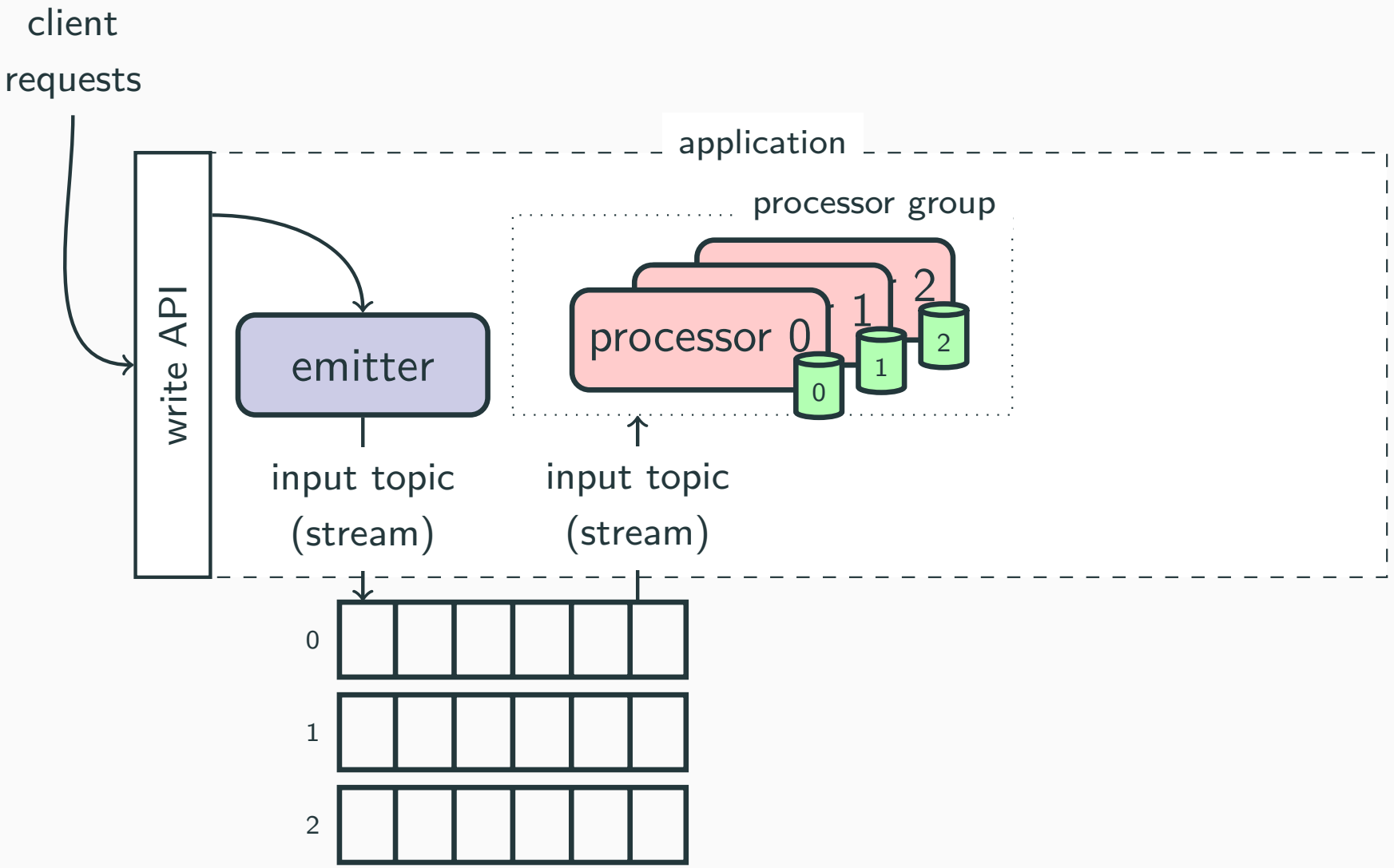
Goka Components



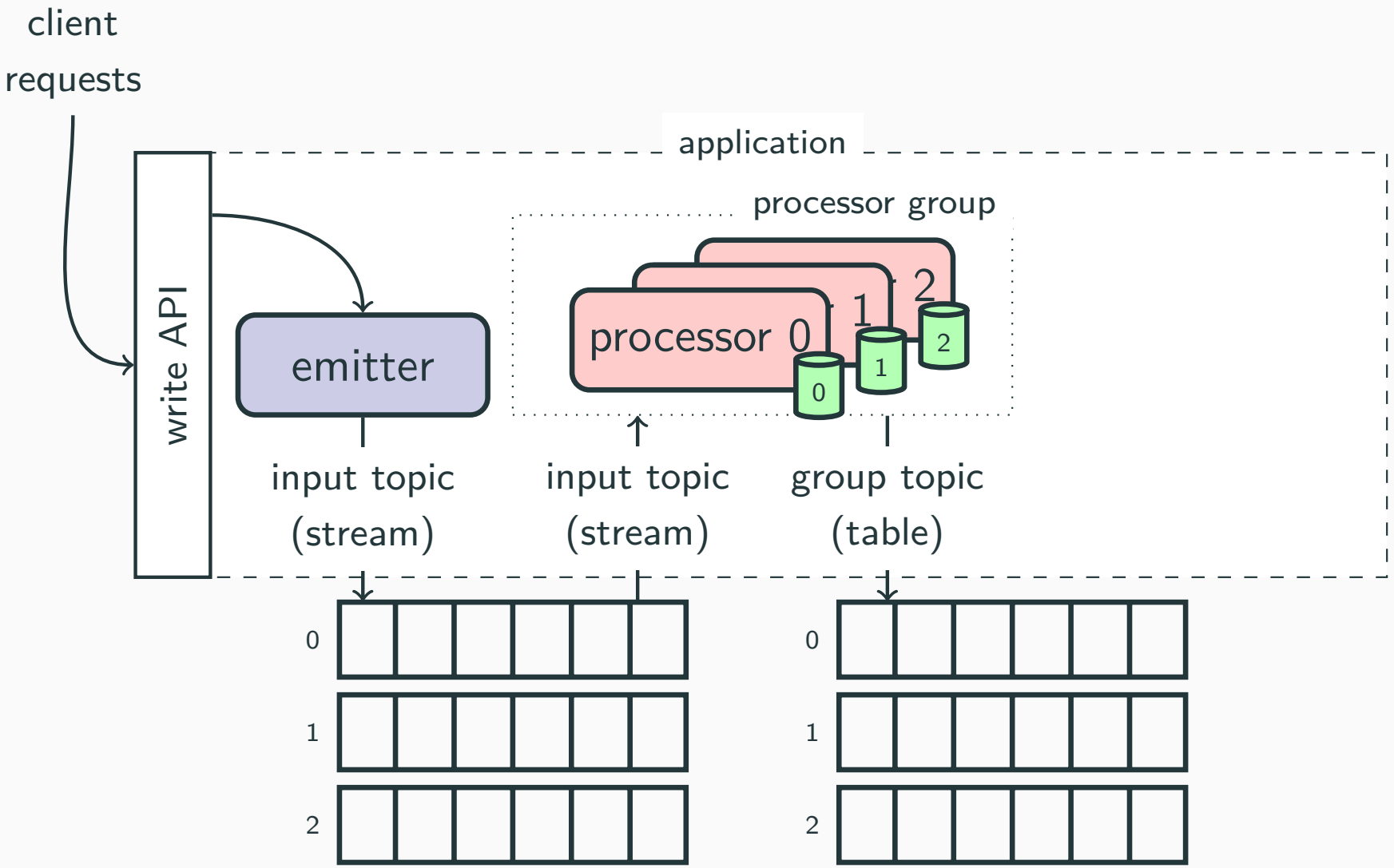
Goka Components



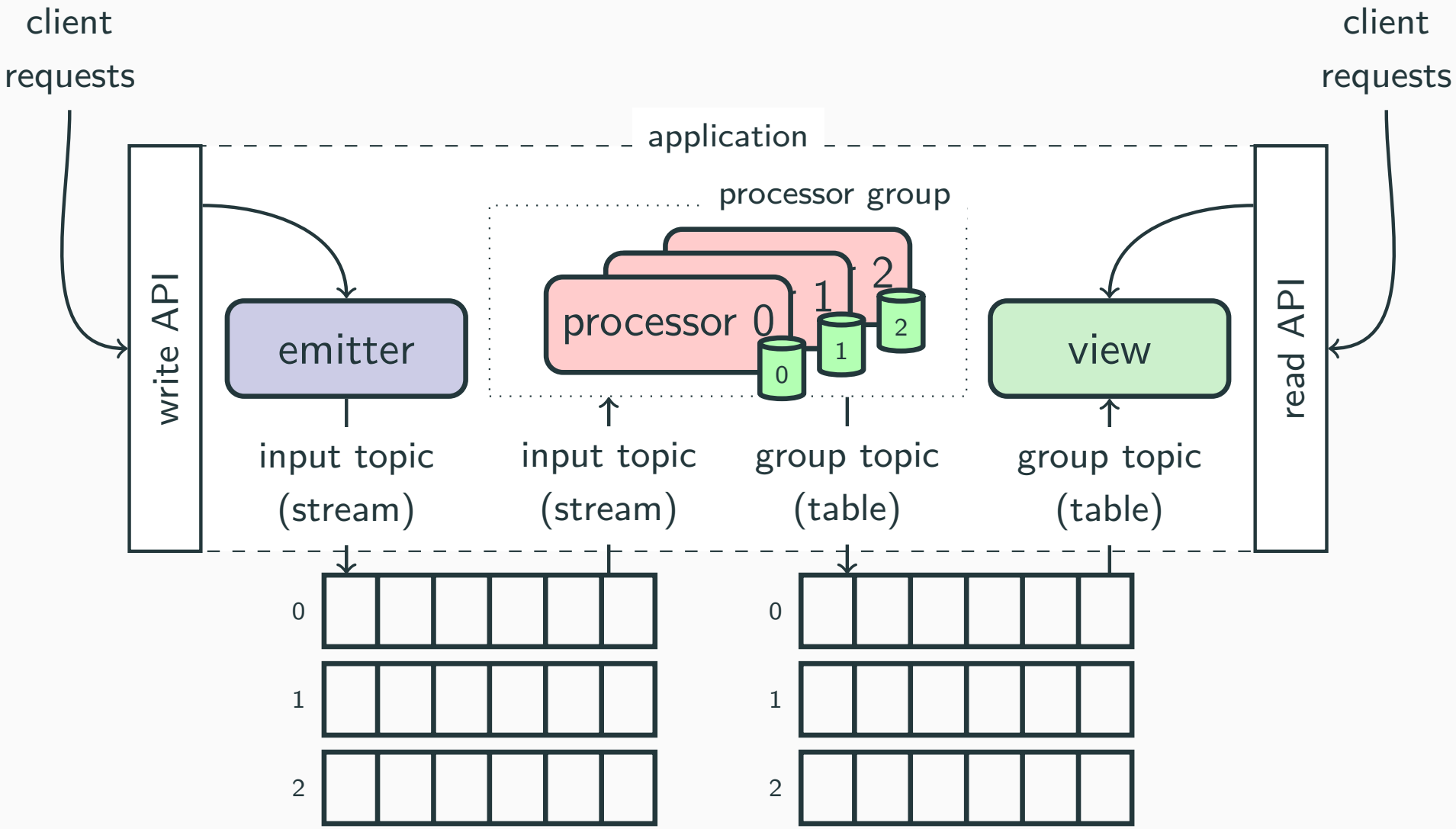
Goka Components



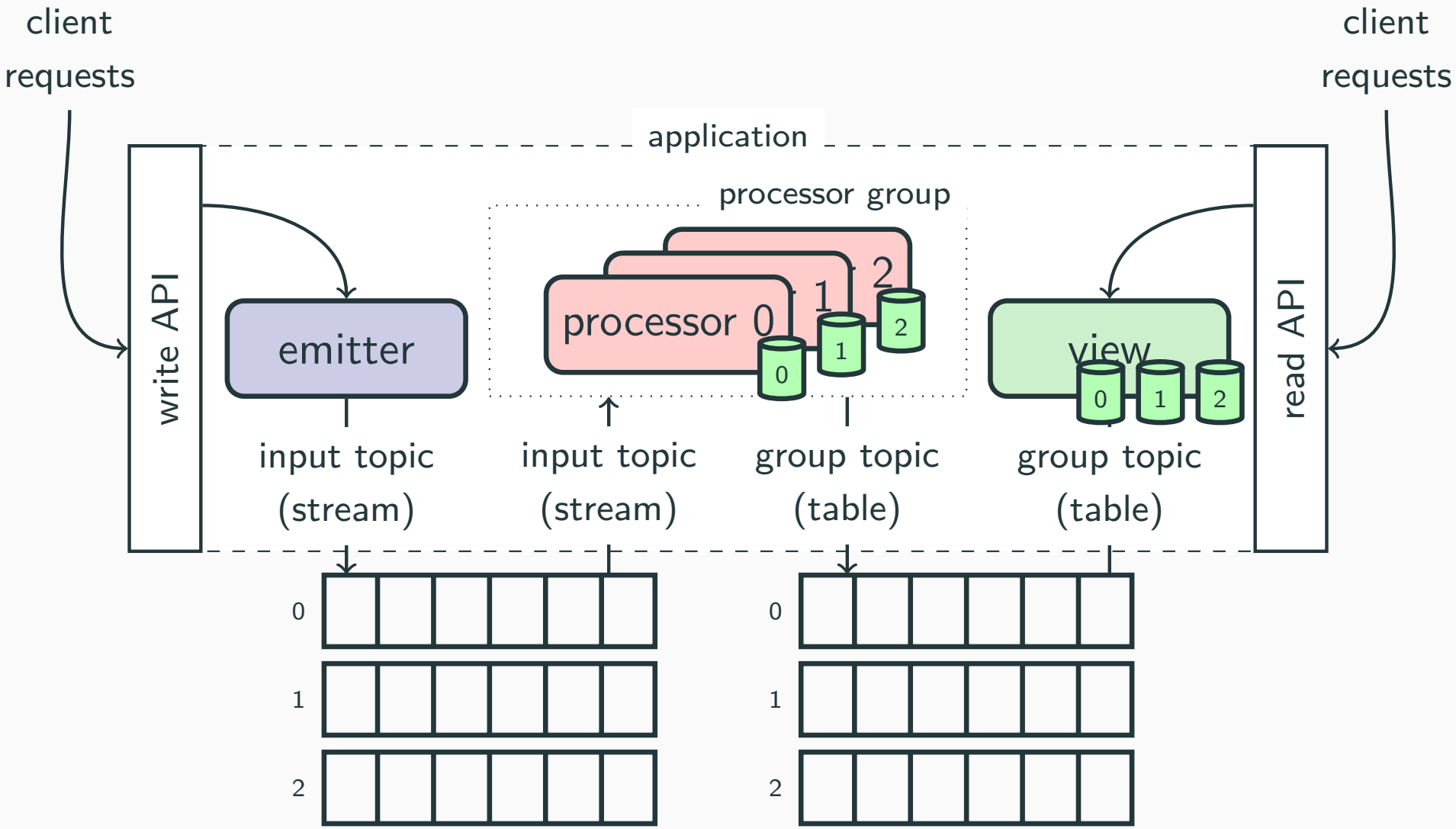
Goka Components



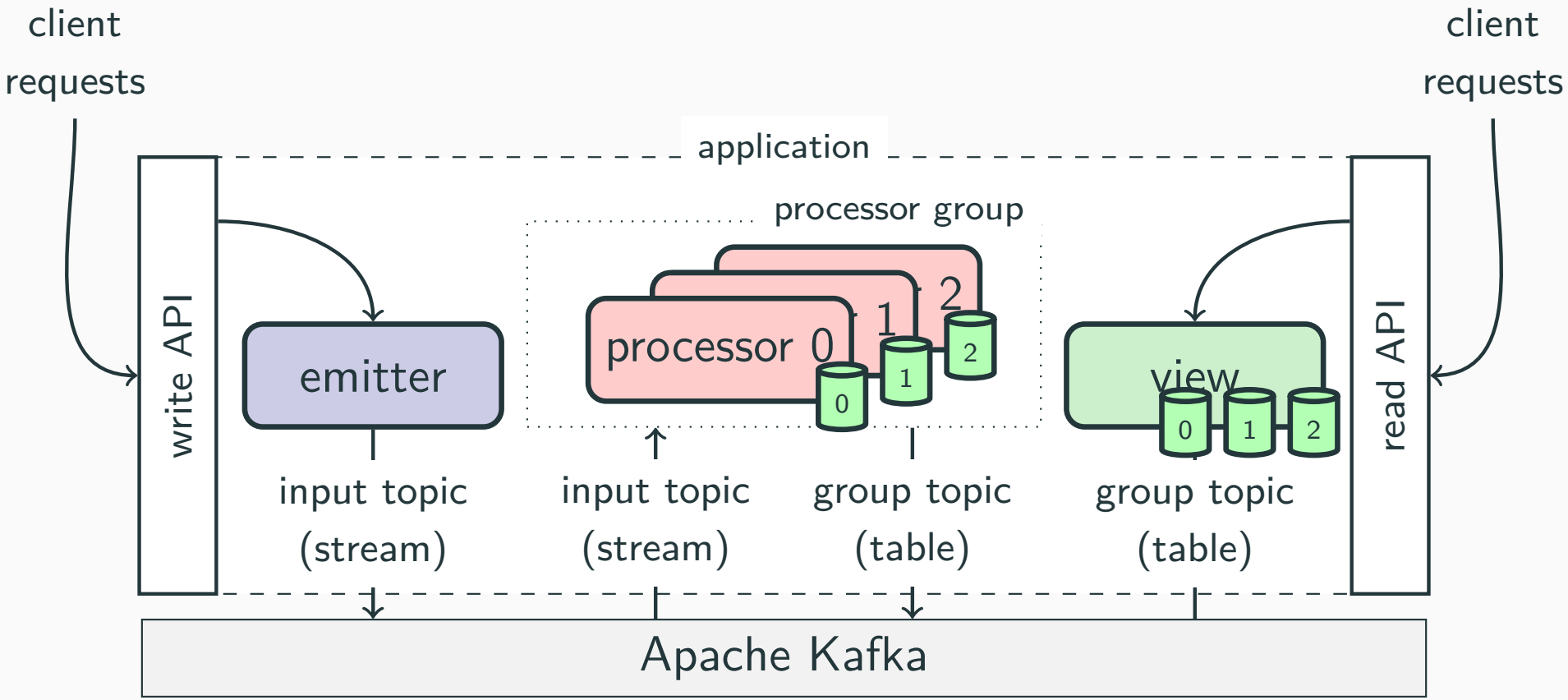
Goka Components



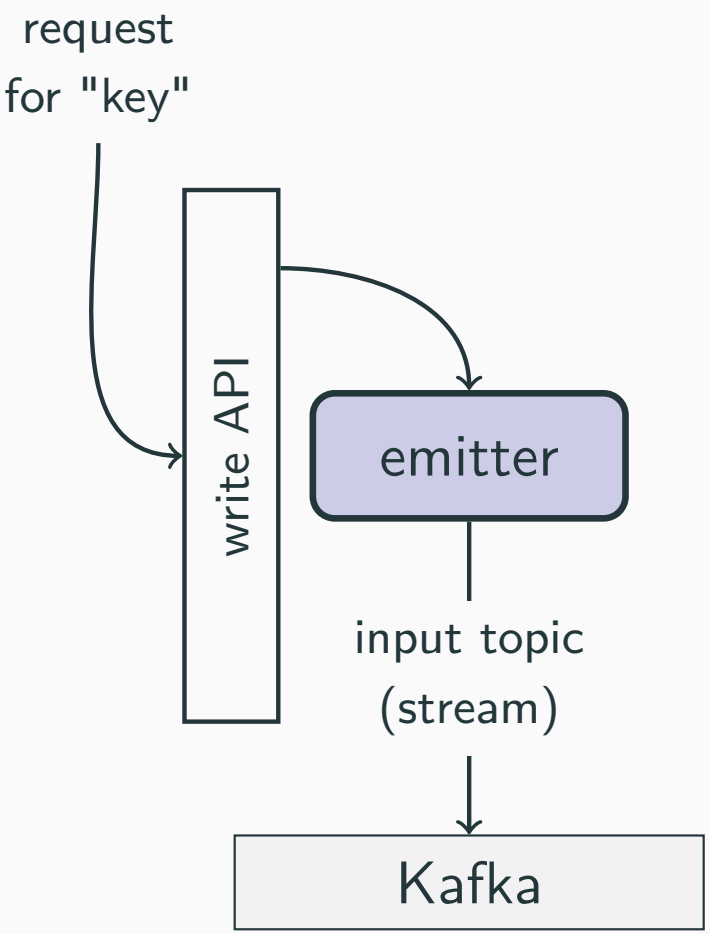
Goka Components



Goka Components

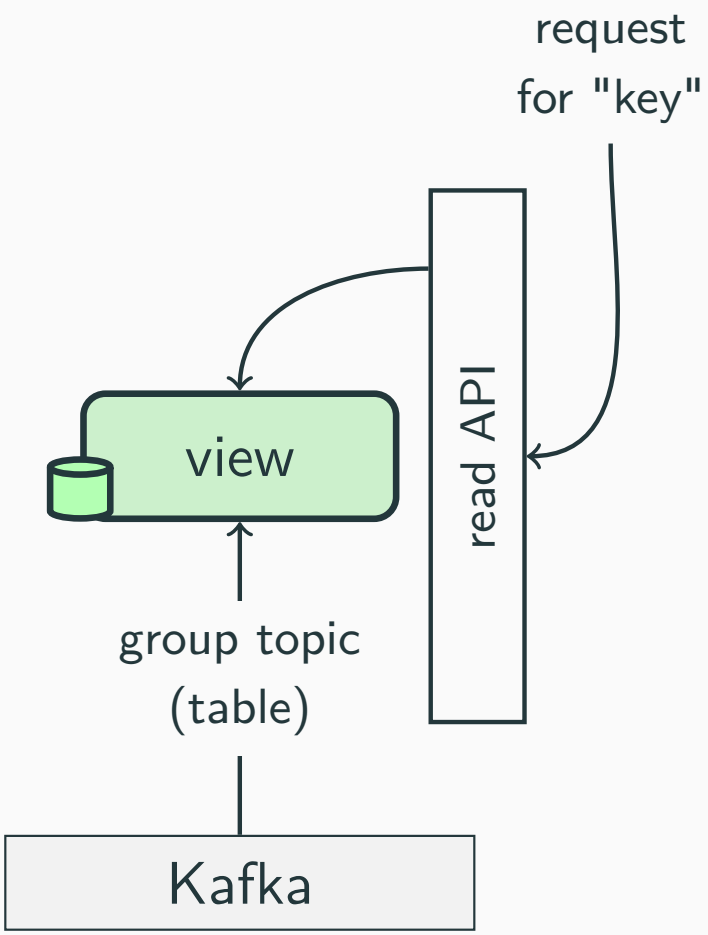


Emitter




```
// create new emitter
e, _ := goka.NewEmitter(
    brokers, // kafka:9092
    "input-stream", // target topic
    streamCodec, // message encoder
)

func handleRequest(req Request) {
  // emit asynchronously
  e.Emit(req.Key, req.Content)
}
```

View



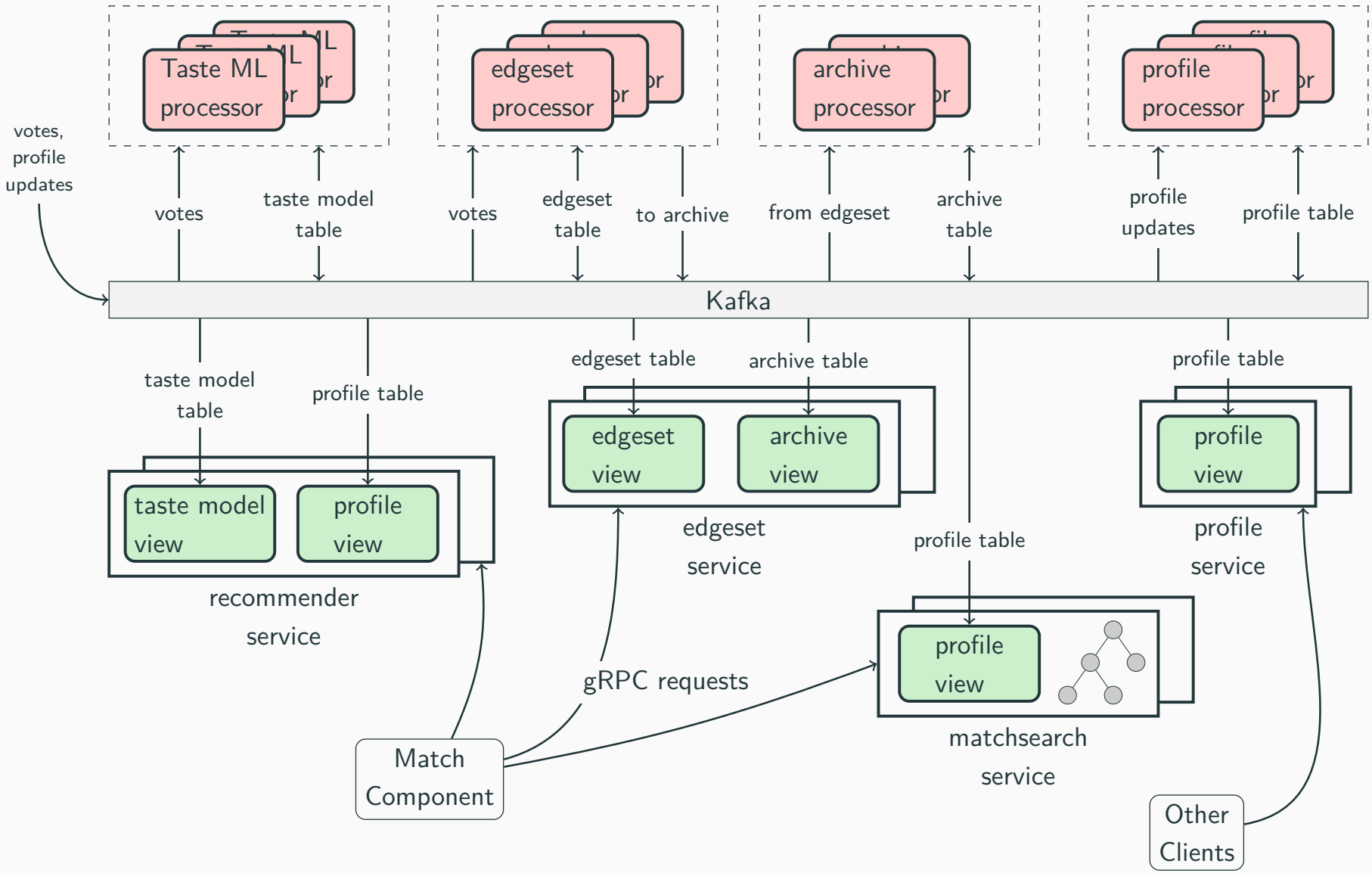

```
// create view
v, _ := goka.NewView(
        brokers,
        "mygroup-table",
        tableCodec,
)
// start view
go v.Run(context.Background())


func handleRequest(req Request) {
  // access view as a kv table
  data, _ := v.Get(req.Key)
  if data == nil {
    // not found
  }
  // safe to cast
  value := data.(*tableEntry)
}
```

Processor

Live coding **and** live drawing. . .

And now for something somewhat different. . .

A Detailed Real Example!


Taste ML processor
edgeset processor
archive processor
profile processor
votes, profile updates
votes
taste model table
votes
edgeset table
to archive
from edgeset
archive table
profile updates
profile table
Kafka
taste model table
profile table
edgeset table
archive table
profile table
taste model view
profile view
recommender service
edgeset view
archive view
edgeset service
profile view
profile service
profile table
gRPC requests
profile view
matchsearch service
Match Component
Other Clients

Closing Discussion

you choose: tips, monitoring, testing, ...

Take aways...

- Streaming microservices are cool!
- Kafka + Golang make a great combo!
- If you need state, consider Goka – it's awesome!

Where to go from here?

- **Try some examples**
  → `github.com/lovoo/goka/tree/master/examples`
  → `github.com/lovoo/cofire`

- **Read the 2-page wiki**
  → `github.com/lovoo/goka/wiki/Introduction`
  → `github.com/lovoo/goka/wiki/Tips`

- **Questions, issues, PRs**
  → `github.com/lovoo/goka/issues`

More about stream processing...

- **Readings**
  → I Heart Logs – Jay Kreps
  → Making sense of stream processing – Martin Kleppmann

- **Video**
  → Turning the database inside out with Samza – Kleppmann
  `https://bit.ly/2b7zjsx`

- **Moath's presentation tomorrow:**
  Using Apache Kafka with Golang

Last but not least...

Feedback? Want to talk? Get in touch!

Diogo – `diogo.behrens@volkswagen.de`

Franz – `franz.eichhorn@lovoo.com`