SLEEPING AT SCALE

LILY MARA

HUNTER LAINE

OneSignal

Refactoring to Rust

Lily Mara
Joel Holmes

MEAP

MANNING

WHAT WE'LL COVER

WHAT WE'LL COVER

- Motivation

WHAT WE'LL COVER

- Motivation
- Architecture

WHAT WE'LL COVER

- Motivation
- Architecture
- Performance

WHAT WE'LL COVER

- Motivation
- Architecture
- Performance
- Scaling

WHAT WE'LL COVER

- Motivation
- Architecture
- Performance
- Scaling
- Future work

ONCE UPON A TIMER

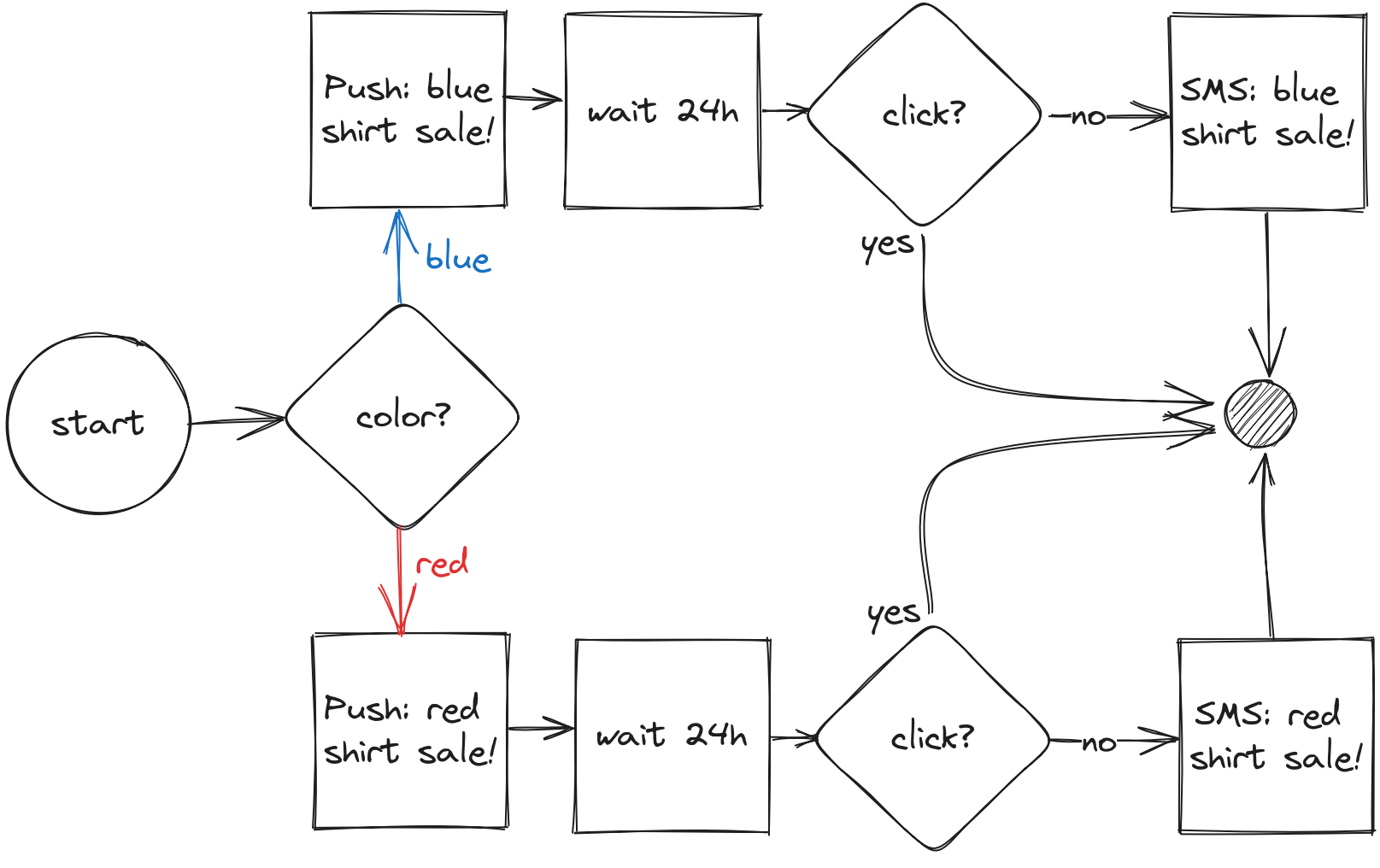
ONCE UPON A TIMER

- 2019

ONCE UPON A TIMER

- 2019
- "Journey builders" are becoming popular

JOURNEY BUILDERS



REQUIREMENTS

REQUIREMENTS

- Store millions/billions of concurrent timers

REQUIREMENTS

- Store millions/billions of concurrent timers
- Expire them performantly

REQUIREMENTS

- Store millions/billions of concurrent timers
- Expire them performantly
- Minimize data loss

REQUIREMENTS

- Store millions/billions of concurrent timers
- Expire them performantly
- Minimize data loss
- Integrate with the rest of our systems

GET IN THE HEADSPACE

GET IN THE HEADSPACE

- To build a timer

GET IN THE HEADSPACE

- To build a timer
- Think like a timer

GET IN THE HEADSPACE

- To build a timer
- Think like a timer
- Come back to the project in a year

JUMPING FORWARD

Q1 2021

BUILD OR BUY

BUILD OR BUY

- Sidekiq / RabbitMQ

BUILD OR BUY

- Sidekiq / RabbitMQ
- General queuing systems

BUILD OR BUY

- Sidekiq / RabbitMQ
- General queuing systems
- We didn't need all of their features

BUILD OR BUY

- Sidekiq / RabbitMQ
- General queuing systems
- We didn't need all of their features
- We didn't believe they'd scale to our needs

BUILD OR BUY

- Sidekiq / RabbitMQ
- General queuing systems
- We didn't need all of their features
- We didn't believe they'd scale to our needs
- Performance seemed orders of magnitude off

WE'RE BUILDING!

REQUIREMENTS

REQUIREMENTS

- Store millions/billions of concurrent timers

REQUIREMENTS

- Store millions/billions of concurrent timers
- Expire them performantly

REQUIREMENTS

- Store millions/billions of concurrent timers
- Expire them performantly
- Minimize data loss

REQUIREMENTS

- Store millions/billions of concurrent timers
- Expire them performantly
- Minimize data loss
- Integrate with the rest of our systems

EXISTING SYSTEMS

EXISTING SYSTEMS

- Rust

EXISTING SYSTEMS

- Rust
- Kafka

EXISTING SYSTEMS

- Rust
- Kafka
- Go

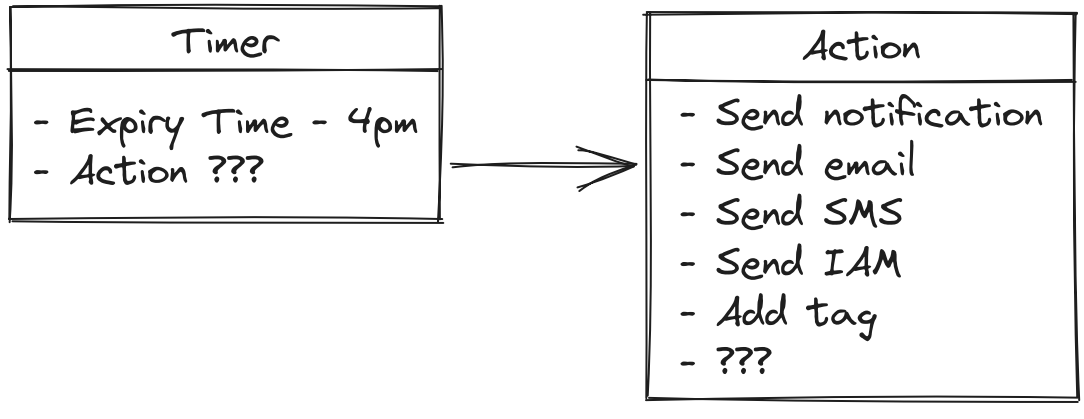
EXISTING SYSTEMS

- Rust
- Kafka
- Go
- gRPC

EXISTING SYSTEMS

- Rust
- Kafka
- Go
- gRPC
- Scylla (Cassandra)

INS & OUTS


Timer
- Expiry Time - 4pm
- Action ???
Action
- Send notification
- Send email
- Send SMS
- Send IAM
- Add tag
- ???

NEW REQUIREMENT

GENERIC

ACTIONS

ACTIONS

- What are they?

ACTIONS

- What are they?
- Should be one option

ACTIONS

- What are they?
- Should be one option
- HTTP + JSON?

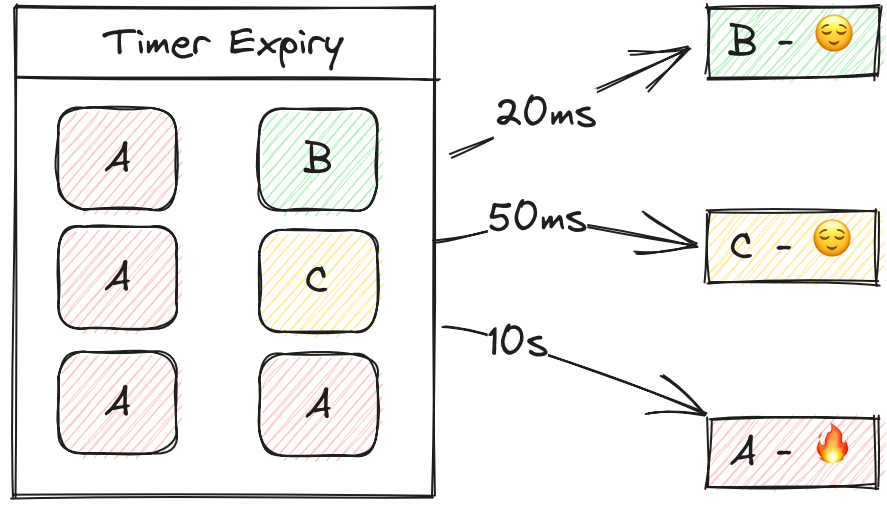
ACTIONS

- What are they?
- Should be one option
- HTTP + JSON?
- gRPC requests?

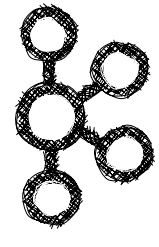
ACTIONS

- What are they?
- Should be one option
- HTTP + JSON?
- gRPC requests?
- Something async

ASYNC

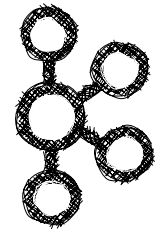


APACHE KAFKA



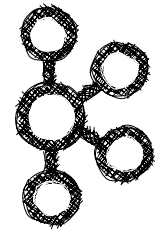
APACHE KAFKA

- Already used



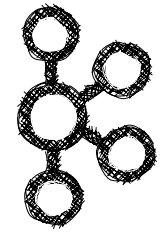
APACHE KAFKA

- Already used
- Queueing system



APACHE KAFKA

- Already used
- Queueing system
- Not reliant on end-system performance



WHAT ABOUT THE INPUTS?

WHAT ABOUT THE INPUTS?

- We own the latency

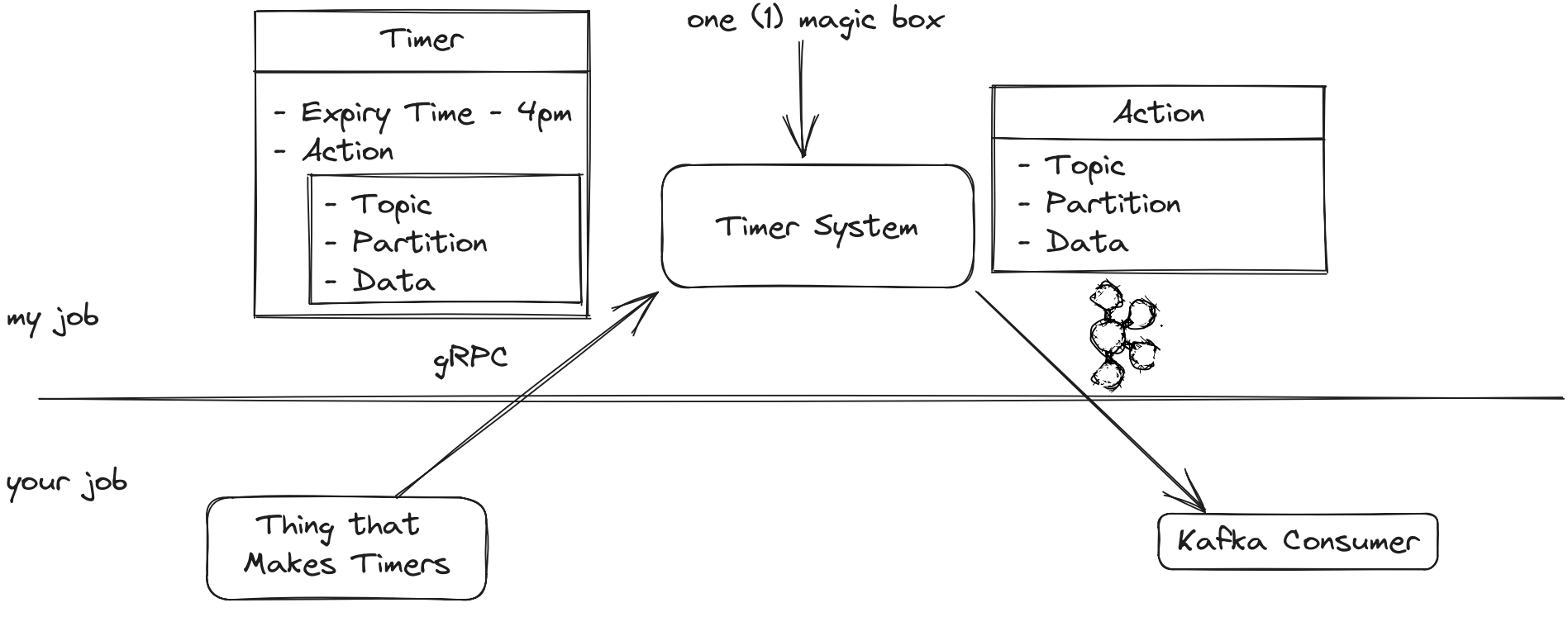
WHAT ABOUT THE INPUTS?

- We own the latency
- This can be synchronous

WHAT ABOUT THE INPUTS?

- We own the latency
- This can be synchronous
- gRPC interface

INTERFACE


Timer
- Expiry Time - 4pm
- Action
- Topic
- Partition
- Data
one (1) magic box
Timer System
Action
- Topic
- Partition
- Data
my job
gRPC
your job
Thing that Makes Timers
Kafka Consumer

INTERNALS

INTERNALS

- Store timers?

INTERNALS

- Store timers?
- Expire timers?

INTERNALS

- Store timers?
- Expire timers?
- Metrics?

TIMER EXPIRY

TIMER EXPIRY

- Write timers to Kafka

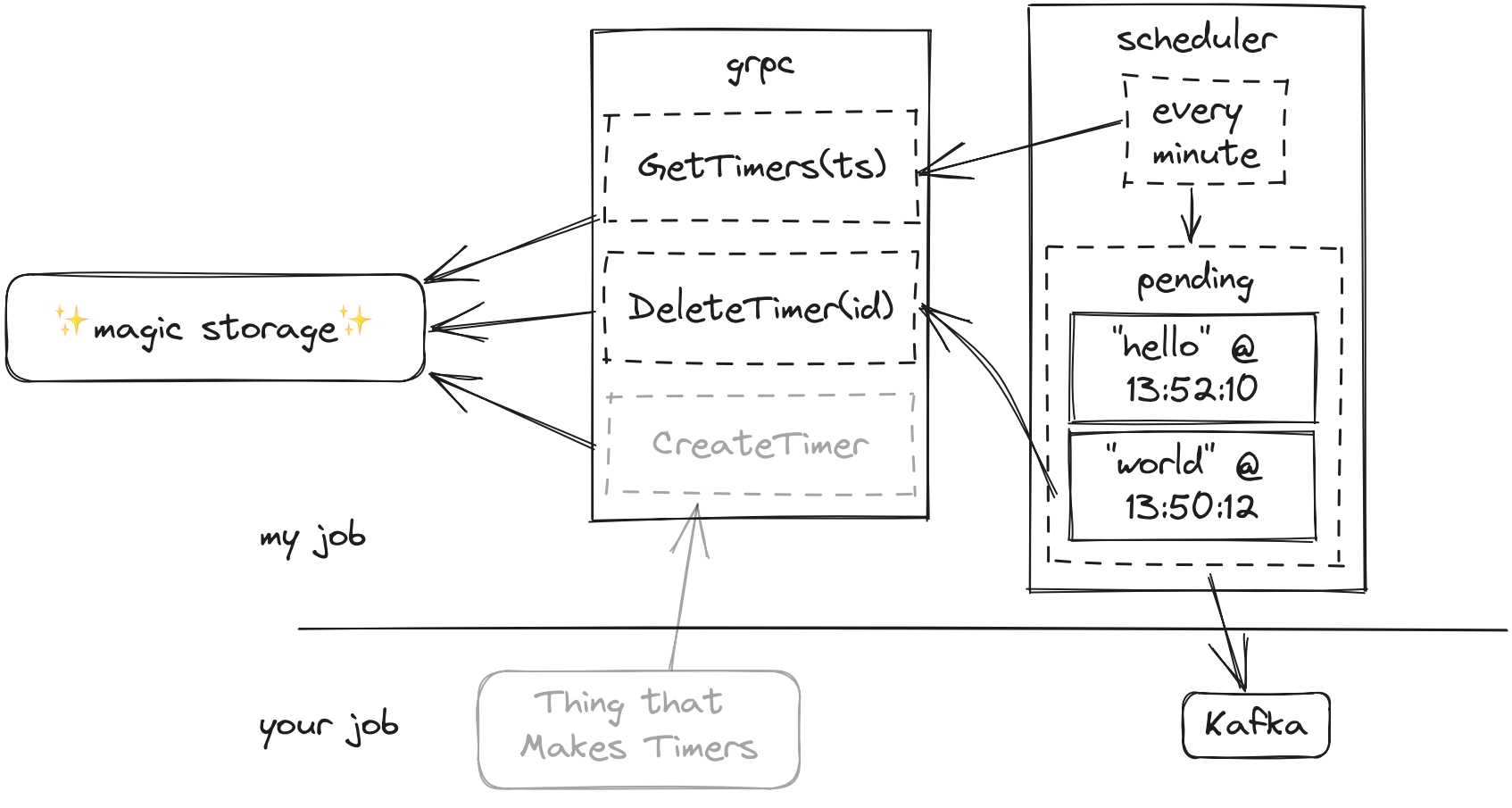
TIMER EXPIRY

- Write timers to Kafka
- Given gRPC server with storage

TIMER EXPIRY

- Write timers to Kafka
- Given gRPC server with storage
- Attempt simplicity

SIMPLE PLAN


grpc
GetTimers(ts)
DeleteTimer(id)
CreateTimer
scheduler
every minute
pending
"hello" @ 13:52:10
"world" @ 13:50:12
✨magic storage✨
my job
your job
Thing that Makes Timers
Kafka

IT CAN'T BE THAT SIMPLE

IT CAN'T BE THAT SIMPLE

- How to represent?

IT CAN'T BE THAT SIMPLE

- How to represent?
- Avoid double-enqueuing

IT WAS THAT SIMPLE

```rust
let mut pending = HashSet::new();
loop {
    let timers = grpc::get_timers().await;
    for timer in timers {
        if pending.insert(timer.id) {
            tokio::spawn(async move {
                tokio::time::
                    sleep_until(timer.expire).await;
                kafka::produce(timer).await;
                grpc::delete_timer(timer.id).await;
                // remove from pending
            });
        }
    }
    tokio::time::sleep(Duration::from_secs(60)).await;
}
```

IT WAS THAT SIMPLE

```
let mut pending = HashSet::new();
loop {
    let timers = grpc::get_timers().await;
    for timer in timers {
        if pending.insert(timer.id) {
            tokio::spawn(async move {
                tokio::time::
                    sleep_until(timer.expire).await;
                kafka::produce(timer).await;
                grpc::delete_timer(timer.id).await;
                // remove from pending
            });
        }
    }
    tokio::time::sleep(Duration::from_secs(60)).await;
}
```

IT WAS THAT SIMPLE

```
let mut pending = HashSet::new();
loop {
    let timers = grpc::get_timers().await;
    for timer in timers {
        if pending.insert(timer.id) {
            tokio::spawn(async move {
                tokio::time::
                    sleep_until(timer.expire).await;
                kafka::produce(timer).await;
                grpc::delete_timer(timer.id).await;
                // remove from pending
            });
        }
    }
    tokio::time::sleep(Duration::from_secs(60)).await;
}
```

IT WAS THAT SIMPLE

```rust
let mut pending = HashSet::new();
loop {
    let timers = grpc::get_timers().await;
    for timer in timers {
        if pending.insert(timer.id) {
            tokio::spawn(async move {
                tokio::time::
                    sleep_until(timer.expire).await;
                kafka::produce(timer).await;
                grpc::delete_timer(timer.id).await;
                // remove from pending
            });
        }
    }
    tokio::time::sleep(Duration::from_secs(60)).await;
}
```

IT WAS THAT SIMPLE

```
let mut pending = HashSet::new();
loop {
    let timers = grpc::get_timers().await;
    for timer in timers {
        if pending.insert(timer.id) {
            tokio::spawn(async move {
                tokio::time::
                    sleep_until(timer.expire).await;
                kafka::produce(timer).await;
                grpc::delete_timer(timer.id).await;
                // remove from pending
            });
        }
    }
    tokio::time::sleep(Duration::from_secs(60)).await;
}
```

IT WAS THAT SIMPLE

```rust
let mut pending = HashSet::new();
loop {
    let timers = grpc::get_timers().await;
    for timer in timers {
        if pending.insert(timer.id) {
            tokio::spawn(async move {
                tokio::time::
                    sleep_until(timer.expire).await;
                kafka::produce(timer).await;
                grpc::delete_timer(timer.id).await;
                // remove from pending
            });
        }
    }
    tokio::time::sleep(Duration::from_secs(60)).await;
}
```

IT WAS THAT SIMPLE

```rust
let mut pending = HashSet::new();
loop {
    let timers = grpc::get_timers().await;
    for timer in timers {
        if pending.insert(timer.id) {
            tokio::spawn(async move {
                tokio::time::
                    sleep_until(timer.expire).await;
                kafka::produce(timer).await;
                grpc::delete_timer(timer.id).await;
                // remove from pending
            });
        }
    }
    tokio::time::sleep(Duration::from_secs(60)).await;
}
```

IT WAS THAT SIMPLE

```
let mut pending = HashSet::new();
loop {
    let timers = grpc::get_timers().await;
    for timer in timers {
        if pending.insert(timer.id) {
            tokio::spawn(async move {
                tokio::time::
                    sleep_until(timer.expire).await;
                kafka::produce(timer).await;
                grpc::delete_timer(timer.id).await;
                // remove from pending
            });
        }
    }
    tokio::time::sleep(Duration::from_secs(60)).await;
}
```

IT WAS THAT SIMPLE

```
let mut pending = HashSet::new();
loop {
    let timers = grpc::get_timers().await;
    for timer in timers {
        if pending.insert(timer.id) {
            tokio::spawn(async move {
                tokio::time::
                    sleep_until(timer.expire).await;
                kafka::produce(timer).await;
                grpc::delete_timer(timer.id).await;
                // remove from pending
            });
        }
    }
    tokio::time::sleep(Duration::from_secs(60)).await;
}
```

NUMBERS

NUMBERS

- `tokio::spawn` + `sleep_until`

NUMBERS

- `tokio::spawn` + `sleep_until`
- 1M timers

NUMBERS

- `tokio::spawn` + `sleep_until`
- 1M timers
- 350ms

NUMBERS

- `tokio::spawn` + `sleep_until`
- 1M timers
- 350ms
- 600 MB

NUMBERS

- `tokio::spawn` + `sleep_until`
- 1M timers
- 350ms
- 600 MB
- "good enough"

METRICS

METRICS

- Pending timers

METRICS

- Pending timers
- Last expired

STORAGE

WHAT DO WE NEED?

WHAT DO WE NEED?

- High write throughput (100k/s)

WHAT DO WE NEED?

- High write throughput (100k/s)
  - Possible but difficult with Postgres

WHAT DO WE NEED?

- High write throughput (100k/s)
  - Possible but difficult with Postgres
- Simple to scale

WHAT DO WE NEED?

- High write throughput (100k/s)
  - Possible but difficult with Postgres
- Simple to scale
- Simple to maintain

WHAT DO WE NEED?

- High write throughput (100k/s)
  - Possible but difficult with Postgres
- Simple to scale
- Simple to maintain
  - Zero-downtime upgrades

WHAT DO WE NEED?

- High write throughput (100k/s)
  - Possible but difficult with Postgres
- Simple to scale
- Simple to maintain
  - Zero-downtime upgrades
- Simple queries with large result sets

END RESULT

- We picked Scylla

END RESULT

- We picked Scylla
- Queries inform data structure

DATA MODELING

DATA MODELING

- Don't ask "what are the data?"

DATA MODELING

- Don't ask "what are the data?"
- SSTables, from Cassandra

DATA MODELING

- Don't ask "what are the data?"
- SSTables, from Cassandra
- Ask "how will we access the data?"

DATA MODELING

- Don't ask "what are the data?"
- SSTables, from Cassandra
- Ask "how will we access the data?"
    - Fetch all timers about to expire

WHAT'S IN A TIMER?

```
1 create table timer.timers (
2     expire timestamp,
3     data blob,
4     k_topic string,
5     k_partition int,
6 );
```

WHAT'S IN A TABLE?

WHAT'S IN A TABLE?

- Note - "Scylla" may mean "Scylla & Casandra"

WHAT'S IN A TABLE?

- Note - "Scylla" may mean "Scylla & Casandra"
- Data distributed across cluster

WHAT'S IN A TABLE?

- Note - "Scylla" may mean "Scylla & Casandra"
- Data distributed across cluster
- IN GENERAL - 1 query hits 1 node

KEYS!

KEYS!

- Primary Key - two parts

KEYS!

- Primary Key - two parts
  - Partition (1+ fields) - which node?

KEYS!

- Primary Key - two parts
  - Partition (1+ fields) - which node?
  - Clustering (0+ fields) - where on the node?

KEYS!

- Primary Key - two parts
  - Partition (1+ fields) - which node?
  - Clustering (0+ fields) - where on the node?
- `SELECT ... WHERE ParKey = "..."`

KEYS!

- Primary Key - two parts
  - Partition (1+ fields) - which node?
  - Clustering (0+ fields) - where on the node?
- `SELECT ... WHERE ParKey = "..."`
- Get timers about to expire

KEYS!

- Primary Key - two parts
  - Partition (1+ fields) - which node?
  - Clustering (0+ fields) - where on the node?
- `SELECT ... WHERE ParKey = "..."`
- Get timers about to expire
- Need to pre-bucket the data

BUCKETING

BUCKETING

- `16:48:30` buckets to `16:45:00`

BUCKETING

- `16:48:30` buckets to `16:45:00`
- All timers `16:45:00` to `16:50:00` are in the same bucket

BUCKETING

- `16:48:30` buckets to `16:45:00`
- All timers `16:45:00` to `16:50:00` are in the same bucket
- Bucket alone cannot be Primary Key

BUCKETING

- `16:48:30` buckets to `16:45:00`
- All timers `16:45:00` to `16:50:00` are in the same bucket
- Bucket alone cannot be Primary Key
- UUID will be clustering key

TABLE

```
create table timer.timers (
    id uuid, -- unique per-row
    bucket timestamp, -- round `expire` to nearest 5 mins
    expire timestamp,
    data blob,
    k_topic string,
    k_partition int,

    PRIMARY KEY (bucket, id)
);
```

TABLE

```
create table timer.timers (
    id uuid, -- unique per-row
    bucket timestamp, -- round `expire` to nearest 5 mins
    expire timestamp,
    data blob,
    k_topic string,
    k_partition int,

    PRIMARY KEY (bucket, id)
);
```

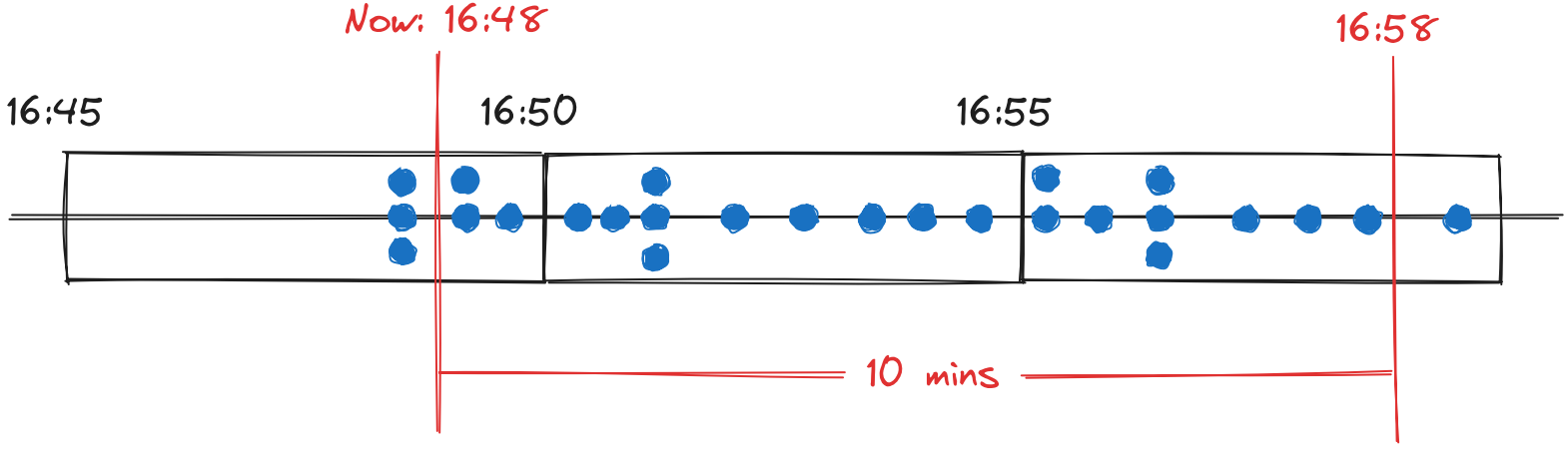
QUERY

```
1 SELECT * FROM
2 timer.timers
3 WHERE bucket = "2023-09-27 16:45:00"
```

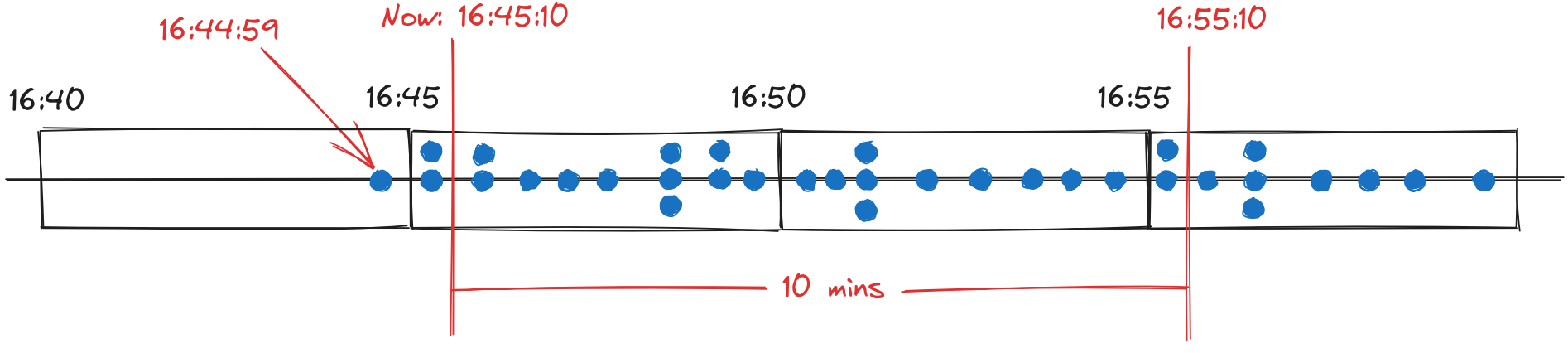
QUERY

```
1 SELECT * FROM
2 timer.timers
3 WHERE bucket = "2023-09-27 16:45:00"
```

BUT



BIGGER BUT



QUERY BAD

- Querying the "active bucket" isn't good enough
- What buckets exist?
- What buckets fall within our lookahead window?
- Query all of those buckets

ANOTHER TABLE!

```
create table timer.buckets (
    bucket timestamp,

    PRIMARY KEY (bucket)
)
```

INSERTIONS

```
1 INSERT INTO timer.timers (bucket, expire, ...)
2     VALUES (16:40, 16:42);
3
4 INSERT INTO timer.buckets (bucket) VALUES (16:40);
```

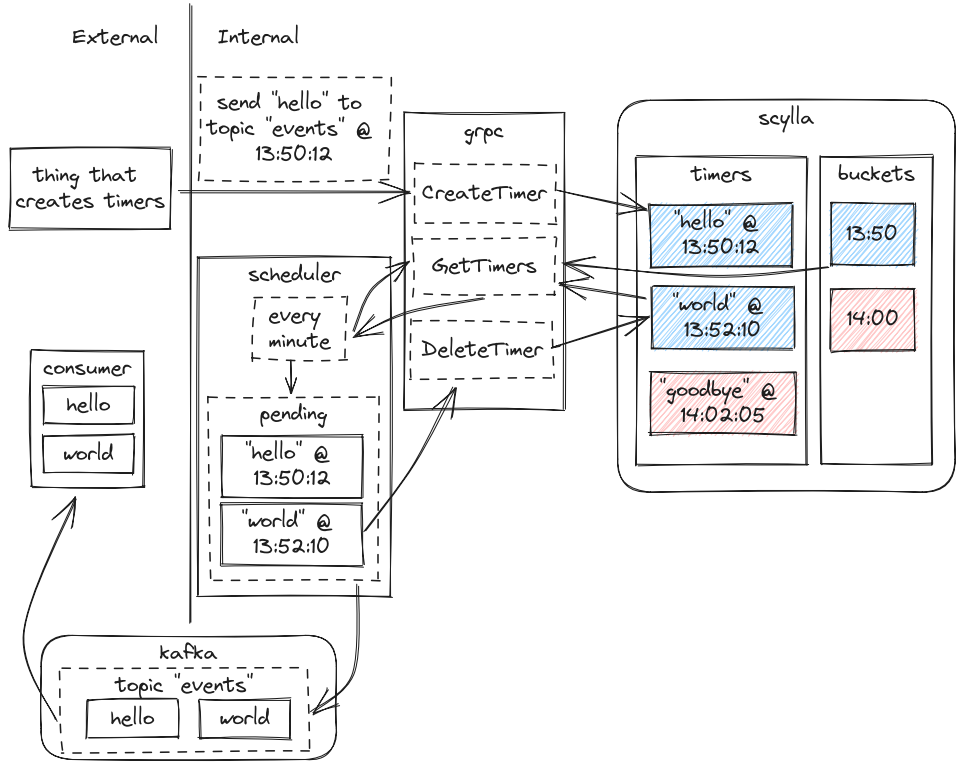
QUERY PATTERN

```
1 SELECT bucket FROM timer.buckets;
2 => 16:40, 16:45, 16:50, 16:55, 17:00...
3 SELECT * FROM timer.timers WHERE bucket=16:40;
4 SELECT * FROM timer.timers WHERE bucket=16:45;
5 SELECT * FROM timer.timers WHERE bucket=16:50;
6 SELECT * FROM timer.timers WHERE bucket=16:55;
```

QUERY PATTERN

```
1 SELECT bucket FROM timer.buckets;
2 => 16:40, 16:45, 16:50, 16:55, 17:00...
3 SELECT * FROM timer.timers WHERE bucket=16:40;
4 SELECT * FROM timer.timers WHERE bucket=16:45;
5 SELECT * FROM timer.timers WHERE bucket=16:50;
6 SELECT * FROM timer.timers WHERE bucket=16:55;
```

TOGETHER


External
Internal
send "hello" to topic "events" @ 13:50:12
thing that creates timers
grpc
CreateTimer
GetTimers
DeleteTimer
scylla
timers
buckets
"hello" @ 13:50:12
"world" @ 13:52:10
"goodbye" @ 14:02:05
13:50
14:00
scheduler
every minute
pending
"hello" @ 13:50:12
"world" @ 13:52:10
consumer
hello
world
kafka
topic "events"
hello
world

JUMPING FORWARD

Q1 2023

JUMPING FORWARD

Q1 2023

- Store billions of timers

JUMPING FORWARD

Q1 2023

- Store billions of timers
- Expire them performantly

JUMPING FORWARD

Q1 2023

- Store billions of timers
- Expire them performantly
- Minimize data loss

JUMPING FORWARD

Q1 2023

- Store billions of timers
- Expire them performantly
- Minimize data loss
- Easily integrate

THE CASE FOR SCALING UP

THE CASE FOR SCALING UP

- 13 billion notifications a day

THE CASE FOR SCALING UP

- 13 billion notifications a day
  - Retry on failure

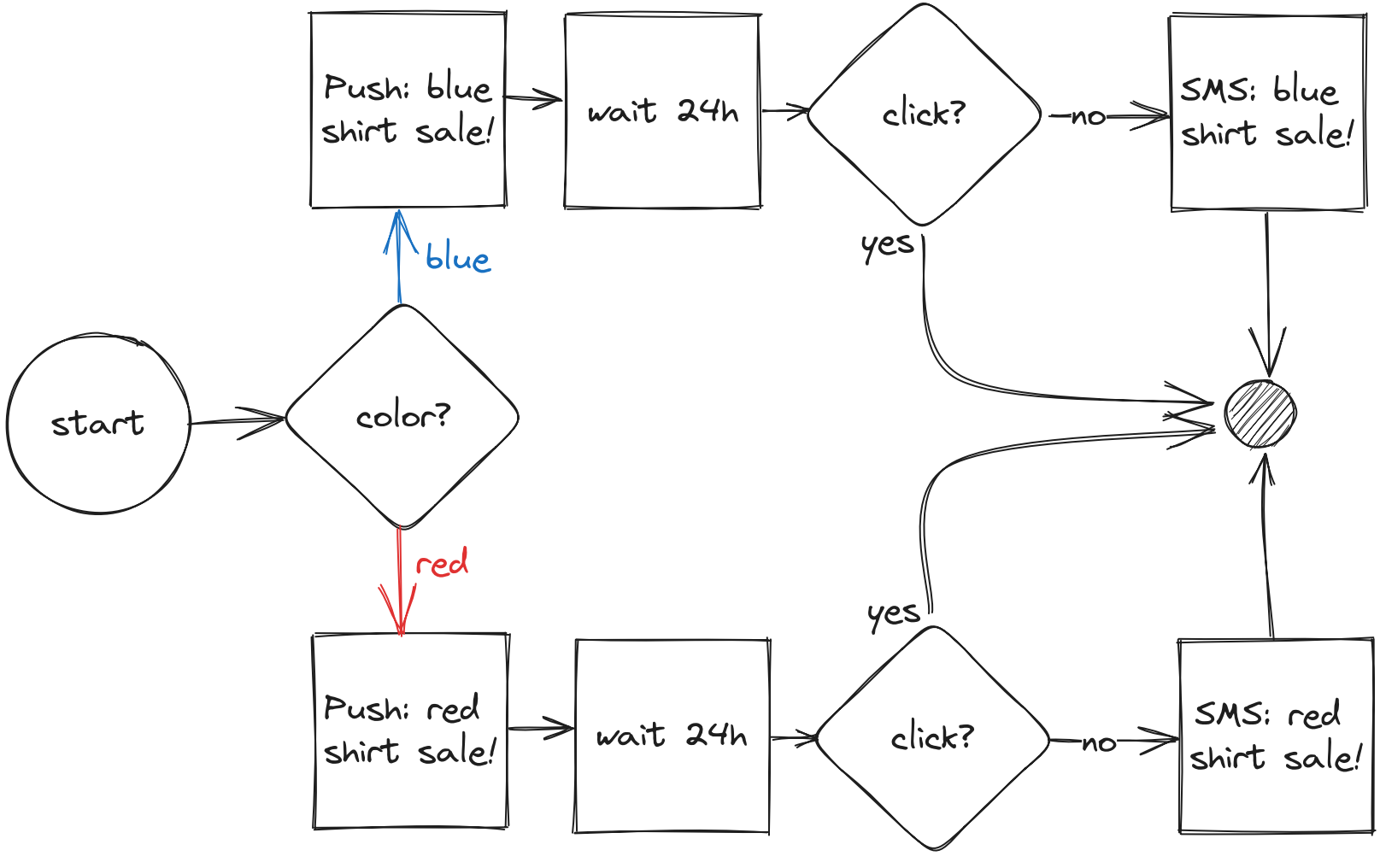
THE CASE FOR SCALING UP

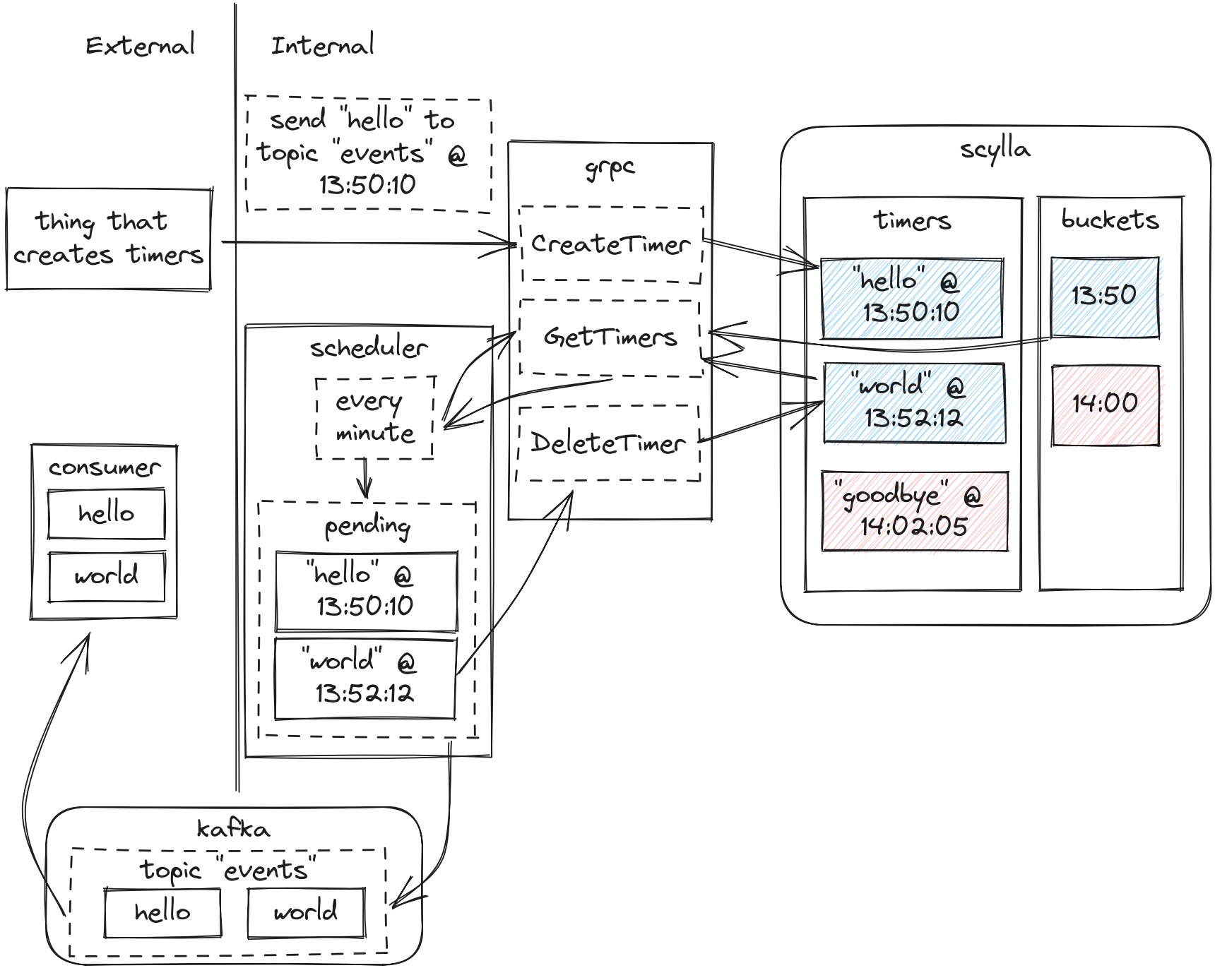
- 13 billion notifications a day
  - Retry on failure
  - Schedule future notifications

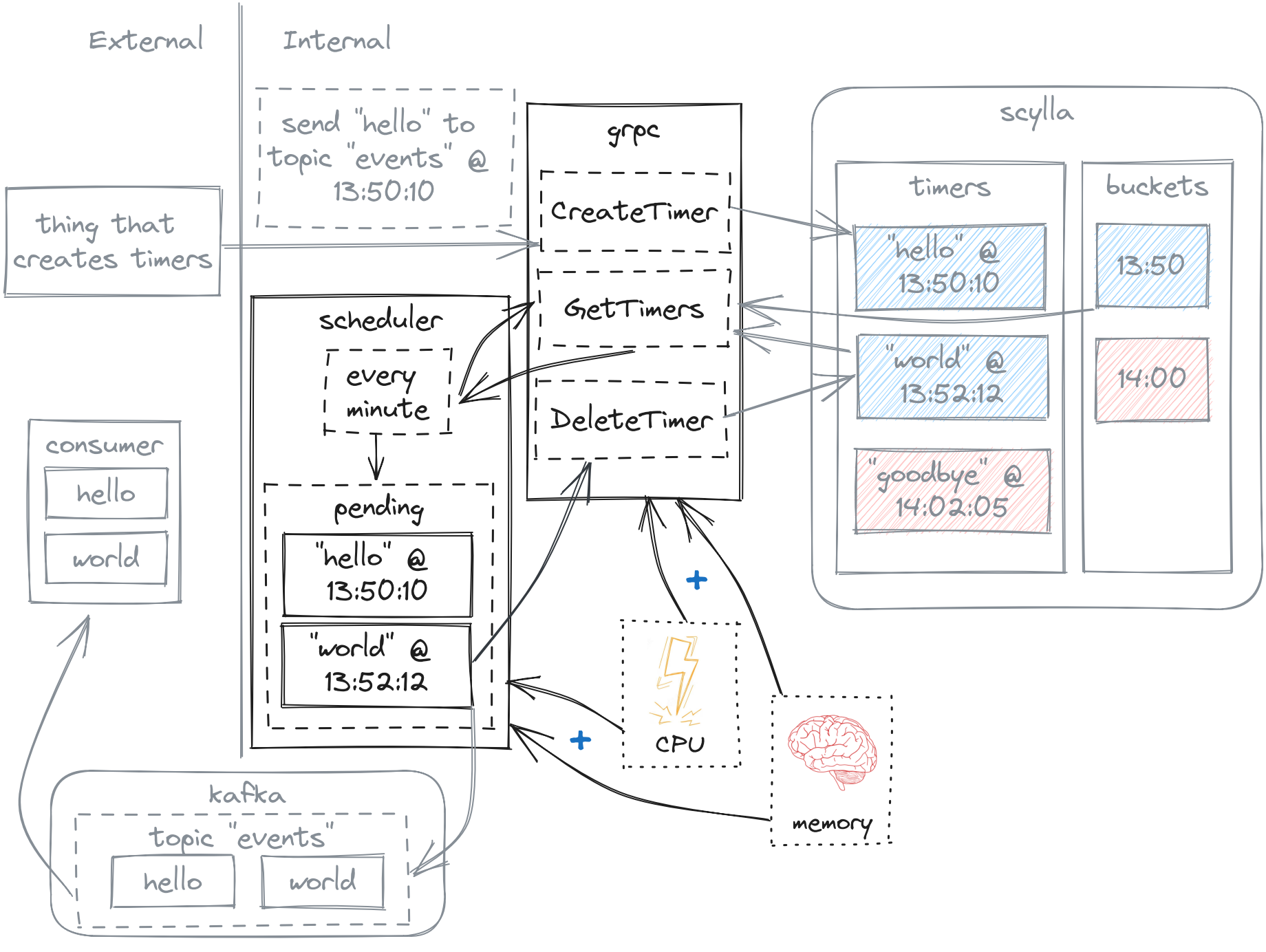
THE CASE FOR SCALING UP

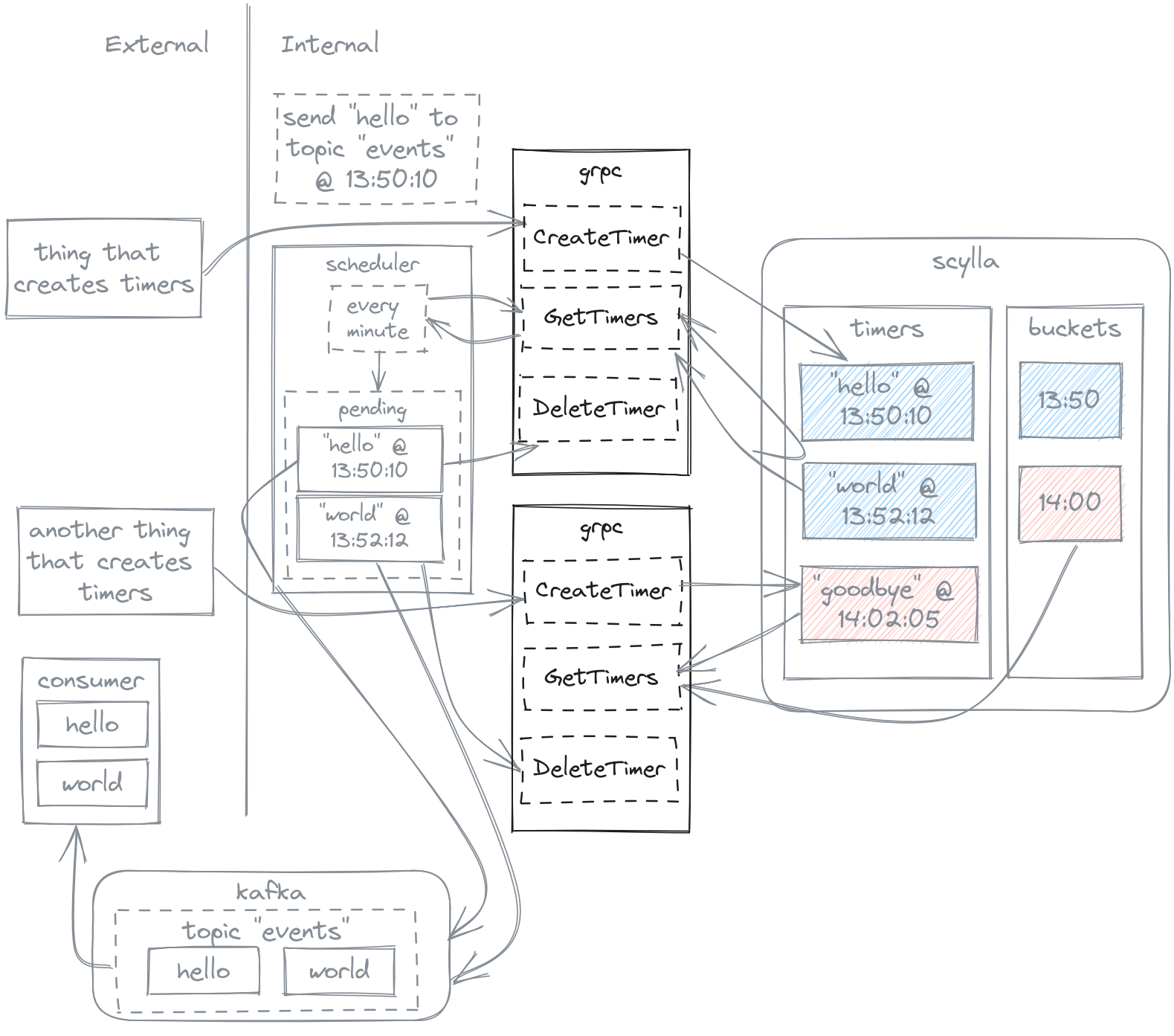
- 13 billion notifications a day
  - Retry on failure
  - Schedule future notifications
- Handle more timers

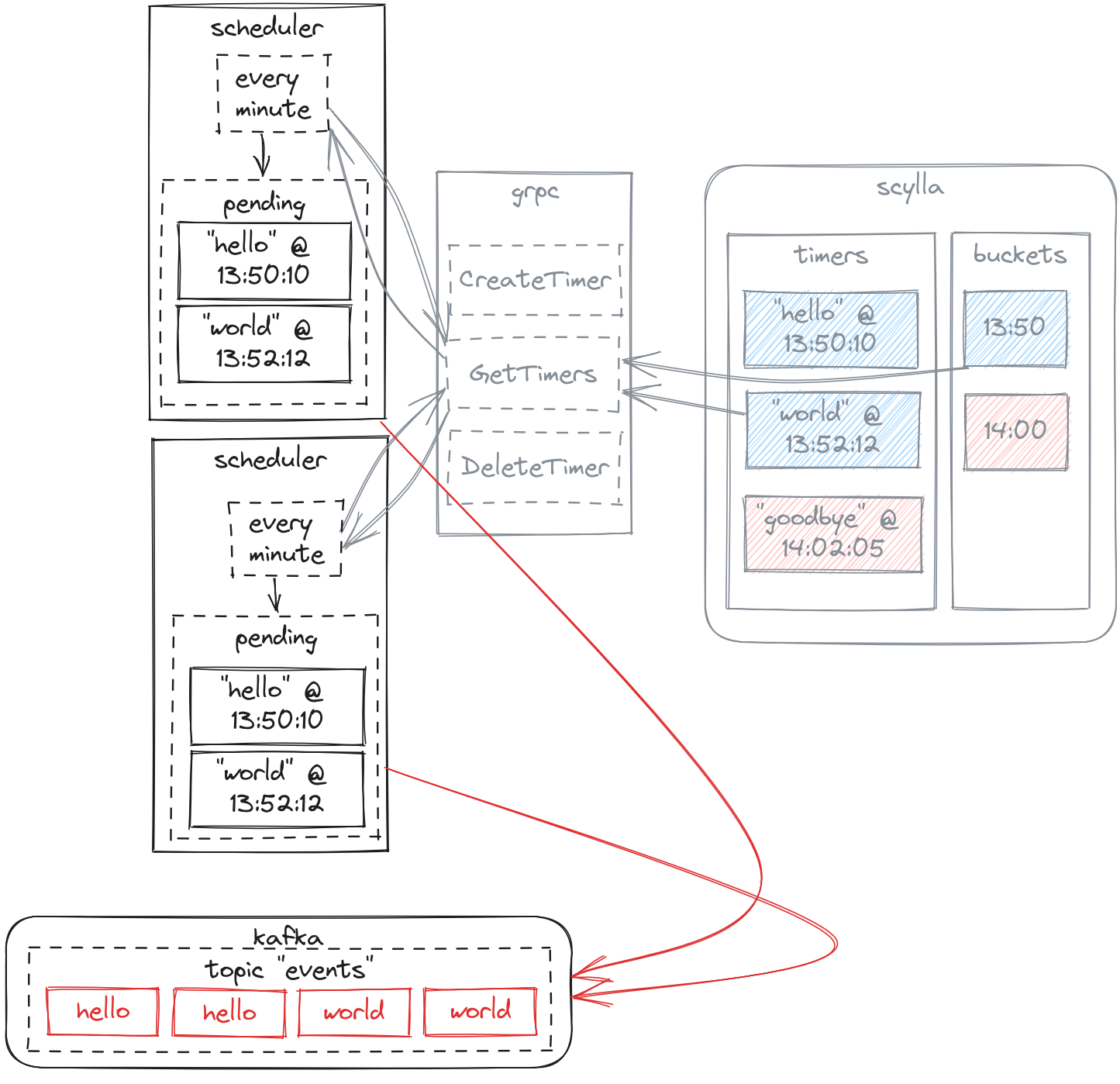
JOURNEY BUILDERS



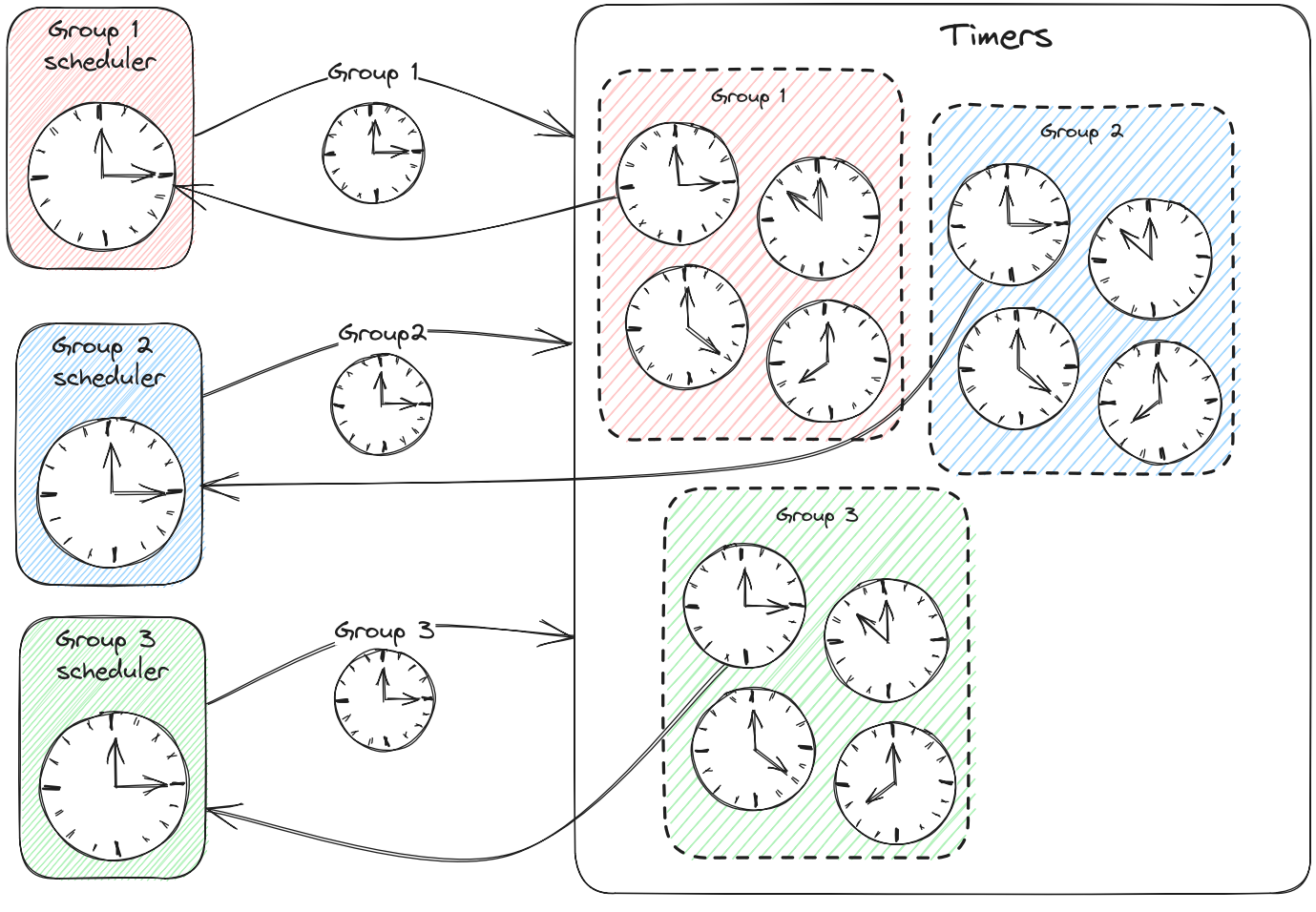

External
Internal
send "hello" to topic "events" @ 13:50:10
thing that creates timers
grpc
CreateTimer
GetTimers
DeleteTimer
scylla
timers
"hello" @ 13:50:10
"world" @ 13:52:12
"goodbye" @ 14:02:05
buckets
13:50
14:00
scheduler
every minute
pending
"hello" @ 13:50:10
"world" @ 13:52:12
consumer
hello
world
kafka
topic "events"
hello
world


External
Internal
send "hello" to topic "events" @ 13:50:10
thing that creates timers
grpc
CreateTimer
GetTimers
DeleteTimer
scylla
timers
buckets
"hello" @ 13:50:10
13:50
"world" @ 13:52:12
14:00
"goodbye" @ 14:02:05
scheduler
every minute
pending
"hello" @ 13:50:10
"world" @ 13:52:12
consumer
hello
world
CPU
memory
kafka
topic "events"
hello
world

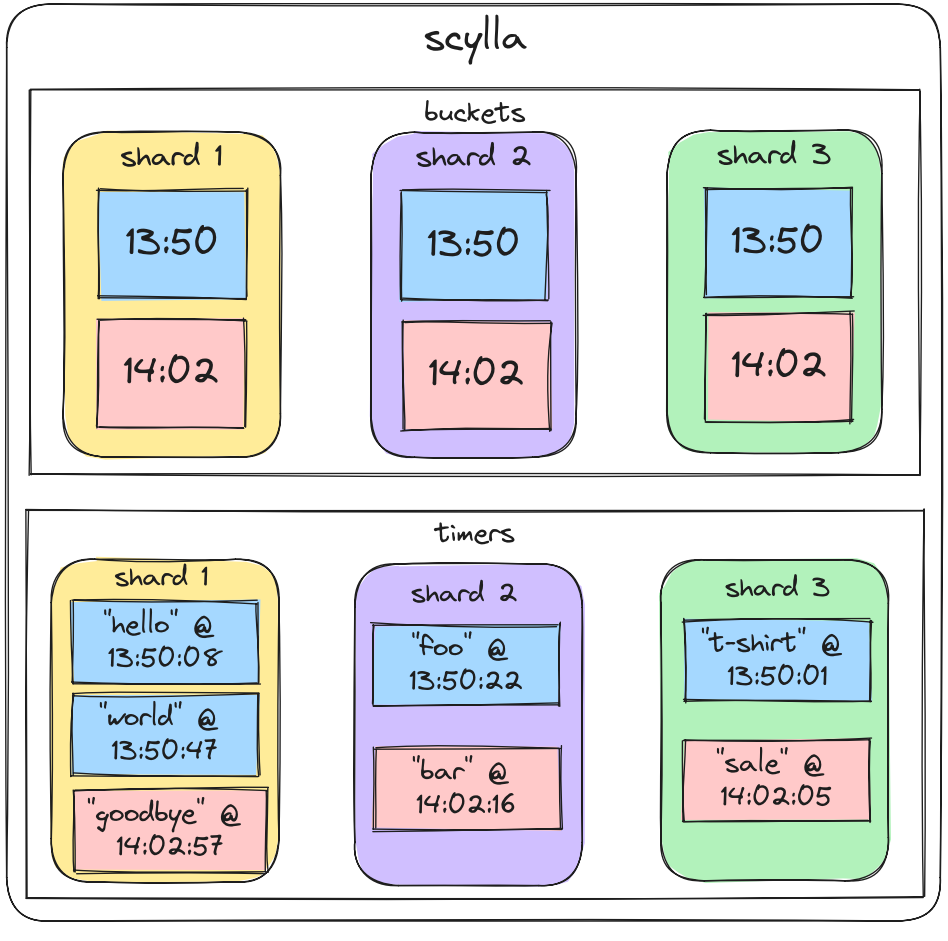

External
Internal
send "hello" to topic "events" @ 13:50:10
thing that creates timers
another thing that creates timers
scheduler
every minute
pending
"hello" @ 13:50:10
"world" @ 13:52:12
grpc
CreateTimer
GetTimers
DeleteTimer
grpc
CreateTimer
GetTimers
DeleteTimer
scylla
timers
"hello" @ 13:50:10
"world" @ 13:52:12
"goodbye" @ 14:02:05
buckets
13:50
14:00
consumer
hello
world
kafka
topic "events"
hello
world


scheduler
every minute
pending
"hello" @ 13:50:10
"world" @ 13:52:12
scheduler
every minute
pending
"hello" @ 13:50:10
"world" @ 13:52:12
grpc
CreateTimer
GetTimers
DeleteTimer
scylla
timers
"hello" @ 13:50:10
"world" @ 13:52:12
"goodbye" @ 14:02:05
buckets
13:50
14:00
kafka
topic "events"
hello
hello
world
world

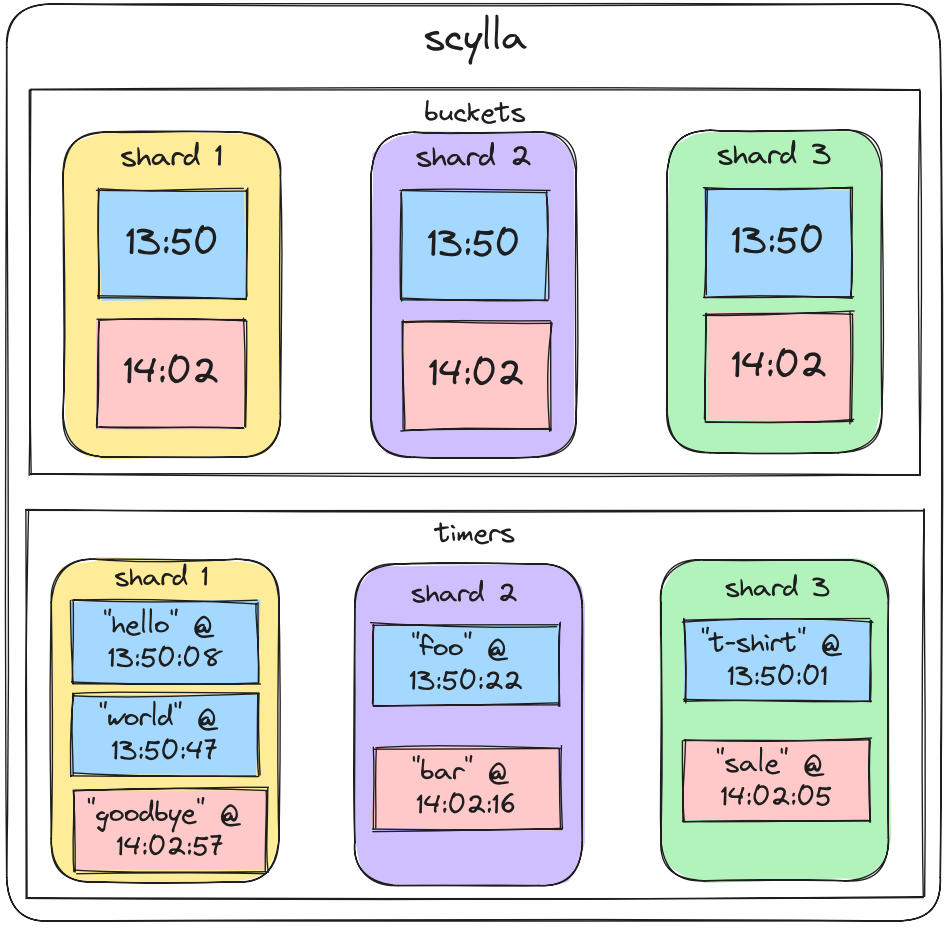
HOW TO SCALE THE SCHEDULER?



SCYLLA SCHEMA CHANGES

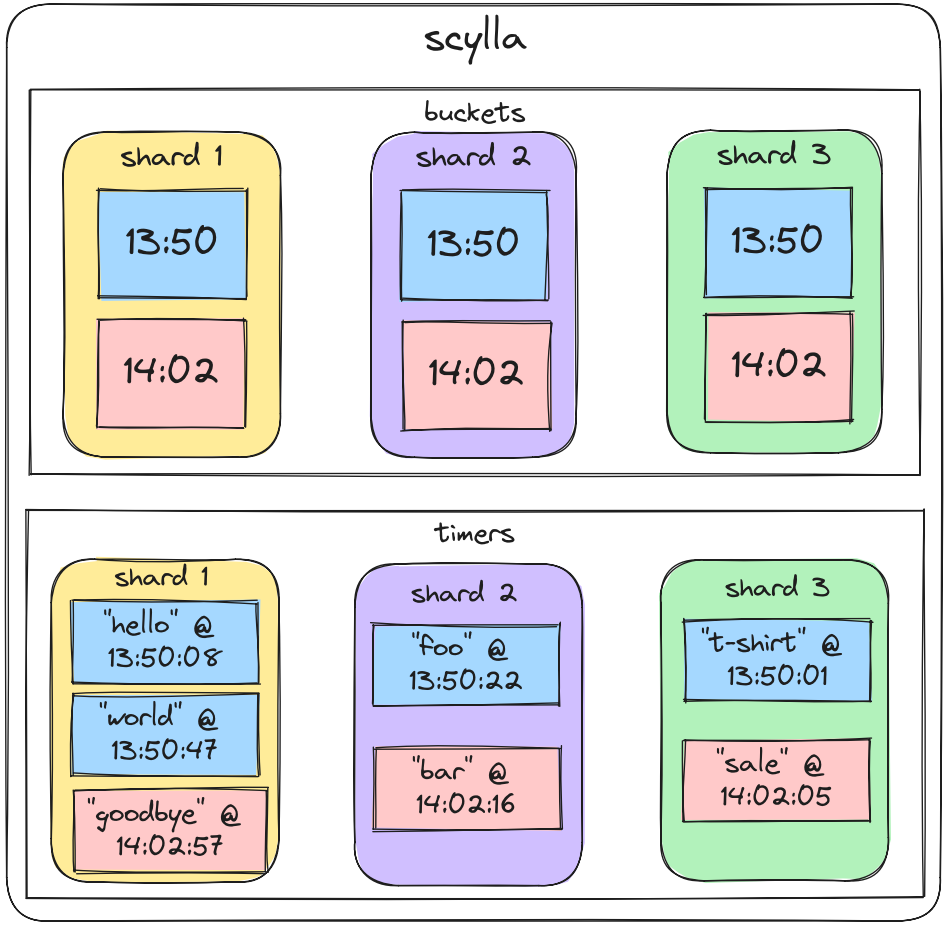


SCYLLA SCHEMA CHANGES



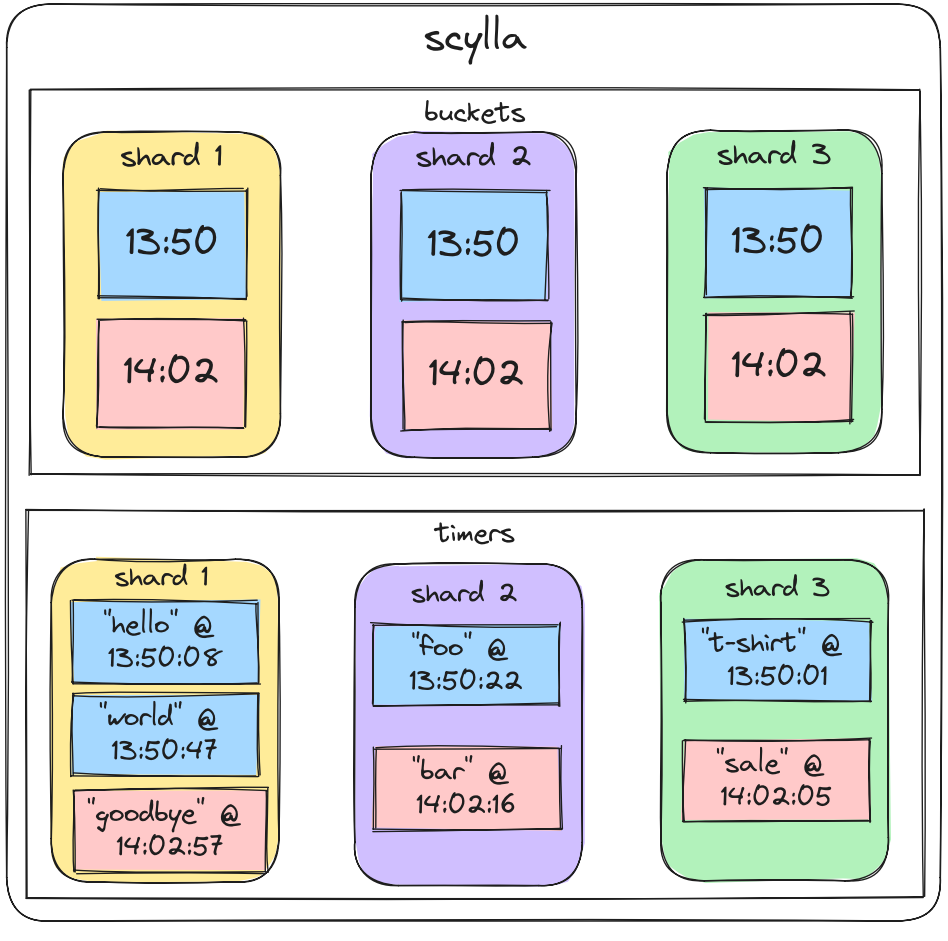
- Group by both time AND shard

SCYLLA SCHEMA CHANGES



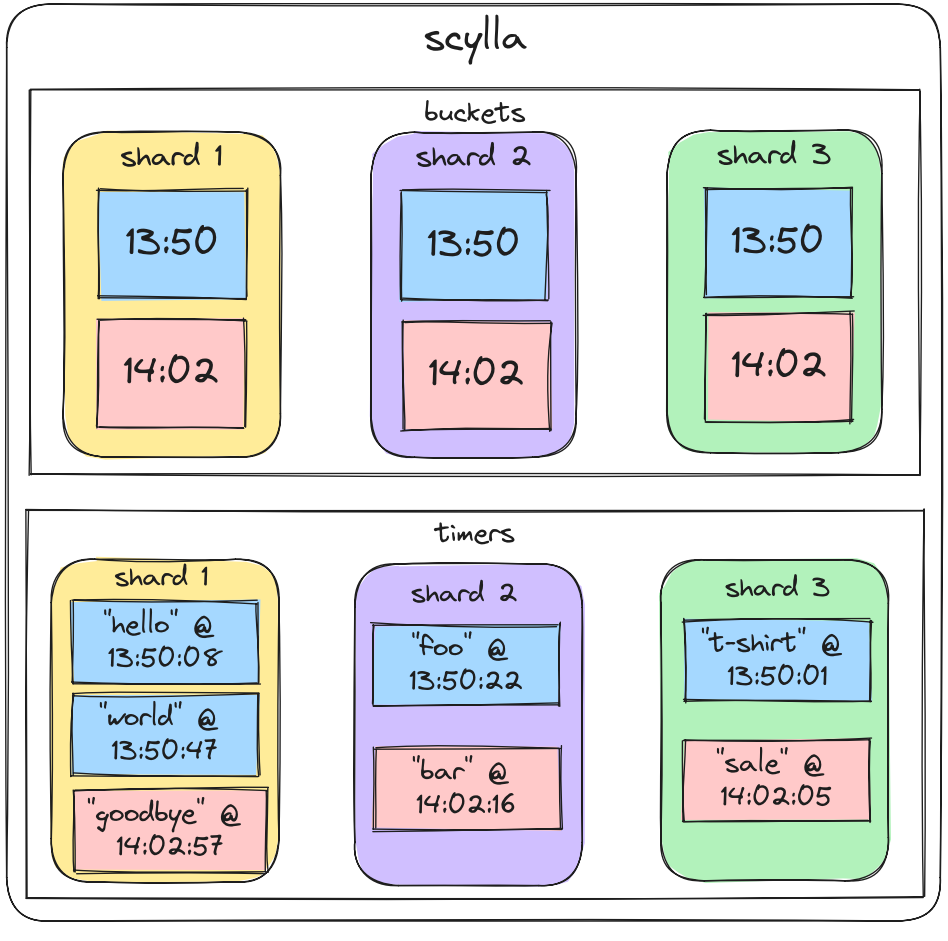
- Group by both time AND shard
- Shrink buckets

SCYLLA SCHEMA CHANGES



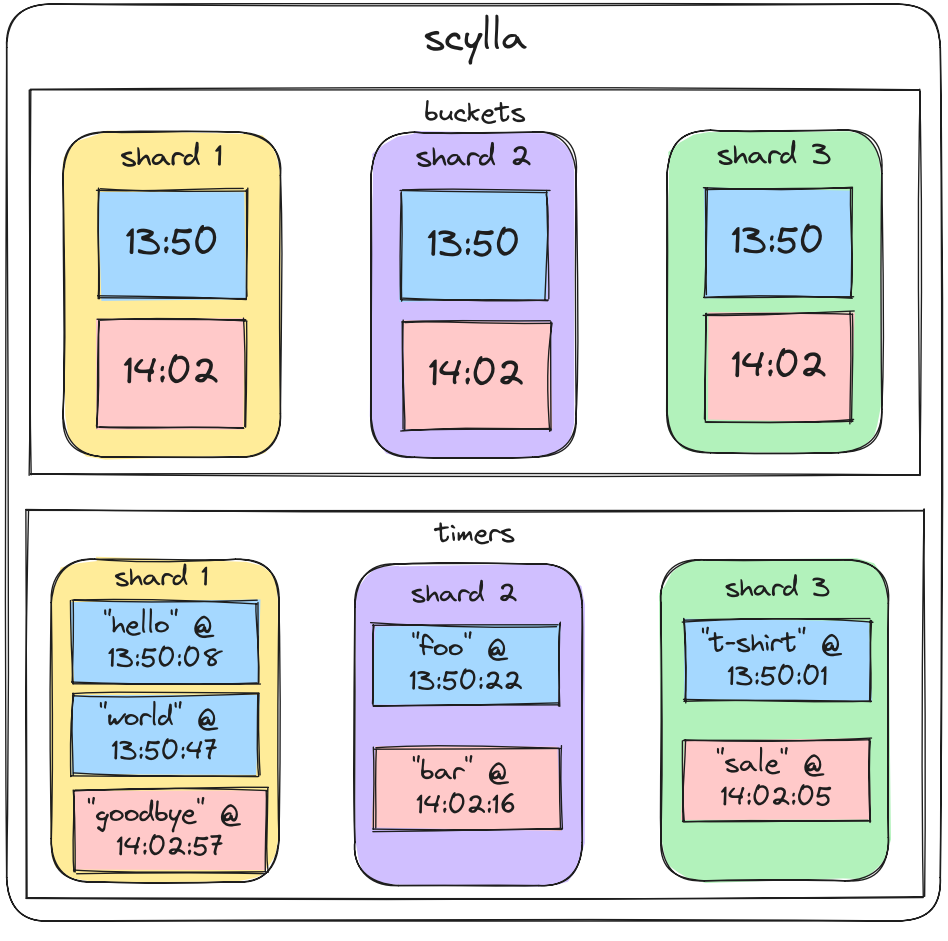
- Group by both time AND shard
- Shrink buckets
- Shard by ID

SCYLLA SCHEMA CHANGES



- Group by both time AND shard
- Shrink buckets
- Shard by ID
  - Each scheduler responsible for a range of shards

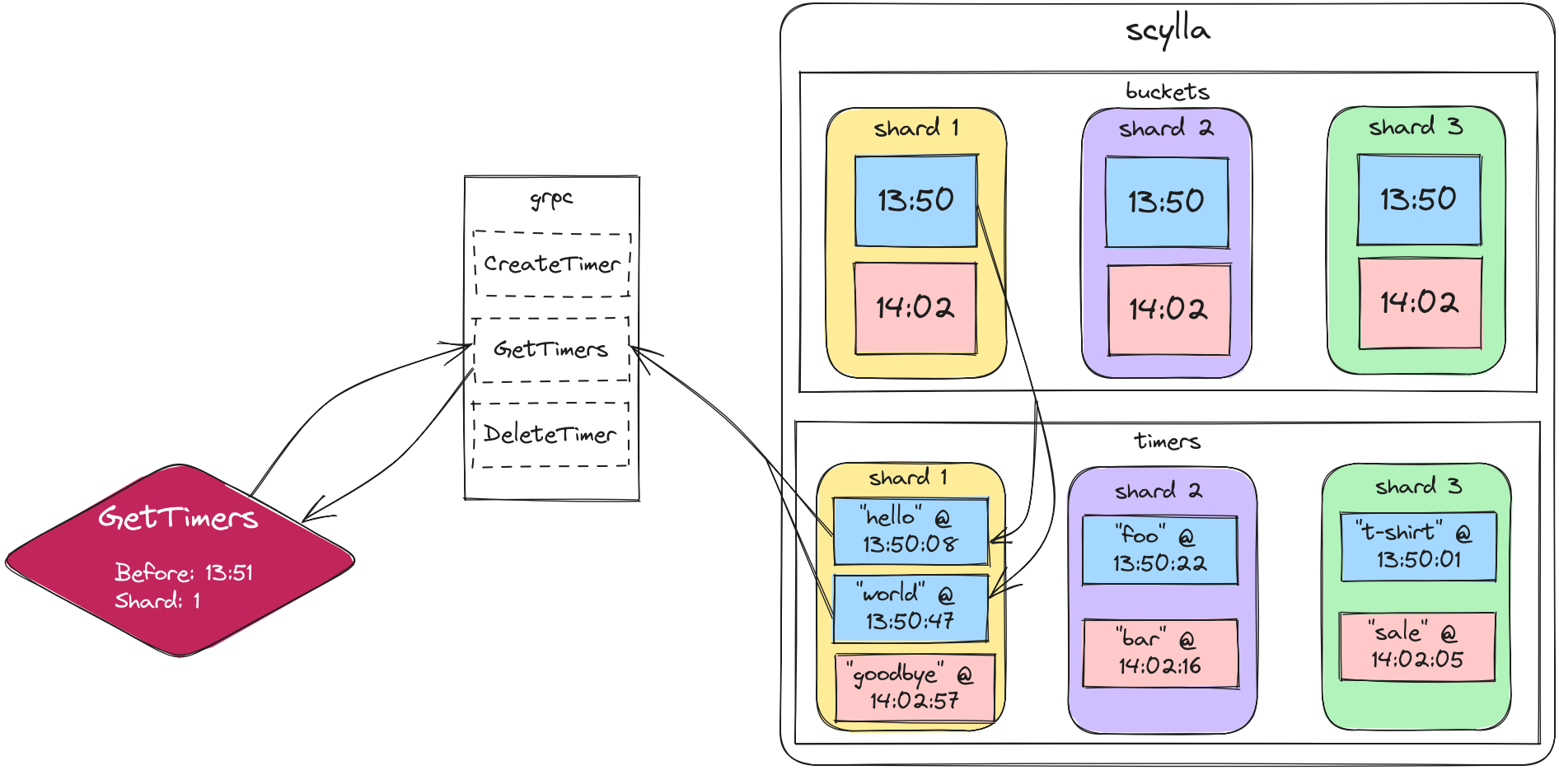
SCYLLA SCHEMA CHANGES

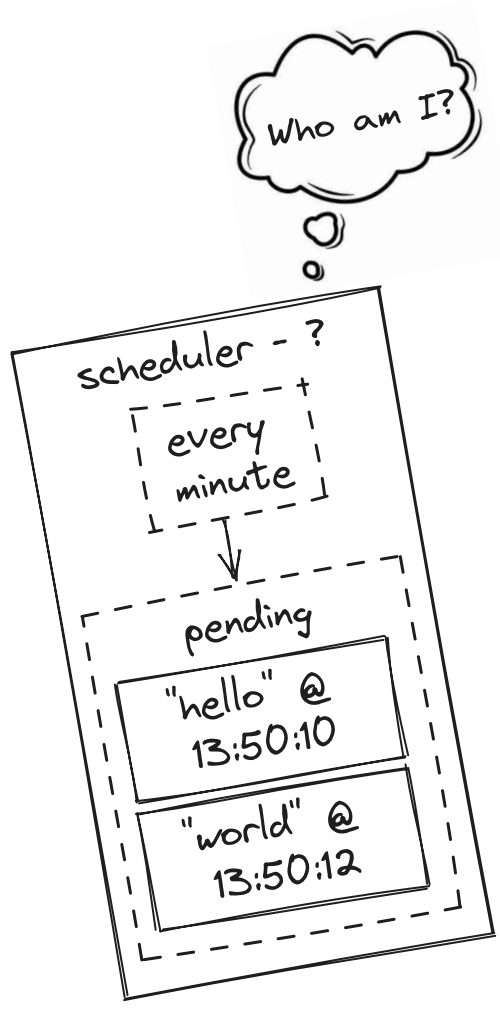


- Group by both time AND shard
- Shrink buckets
- Shard by ID
  - Each scheduler responsible for a range of shards
- More efficient bookkeeping and querying

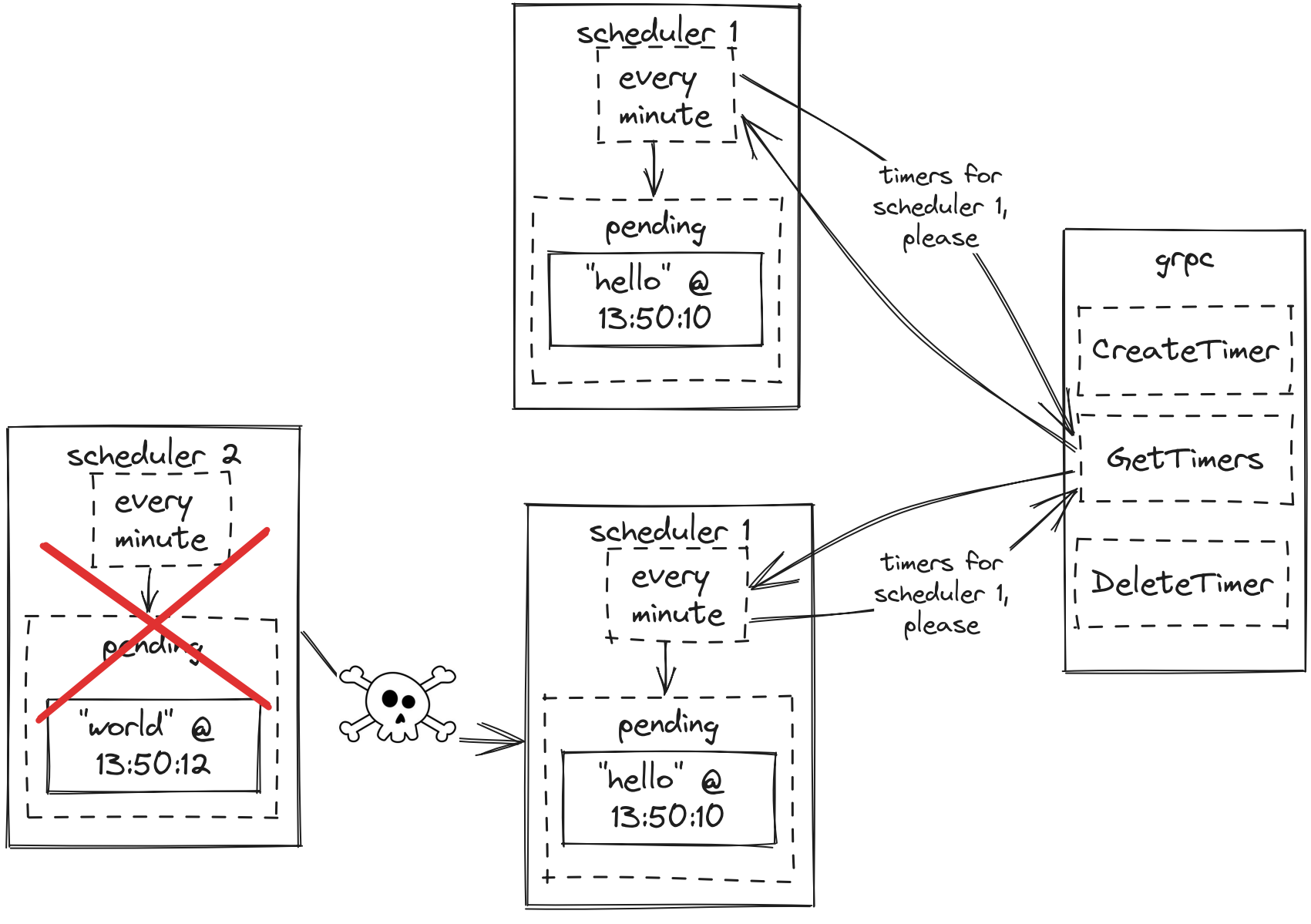
GET TIMERS REQUEST

SUCCESS! (ALMOST)




Who am I?
scheduler - ?
every minute
pending
"hello" @ 13:50:10
"world" @ 13:50:12

IDENTITY THEFT



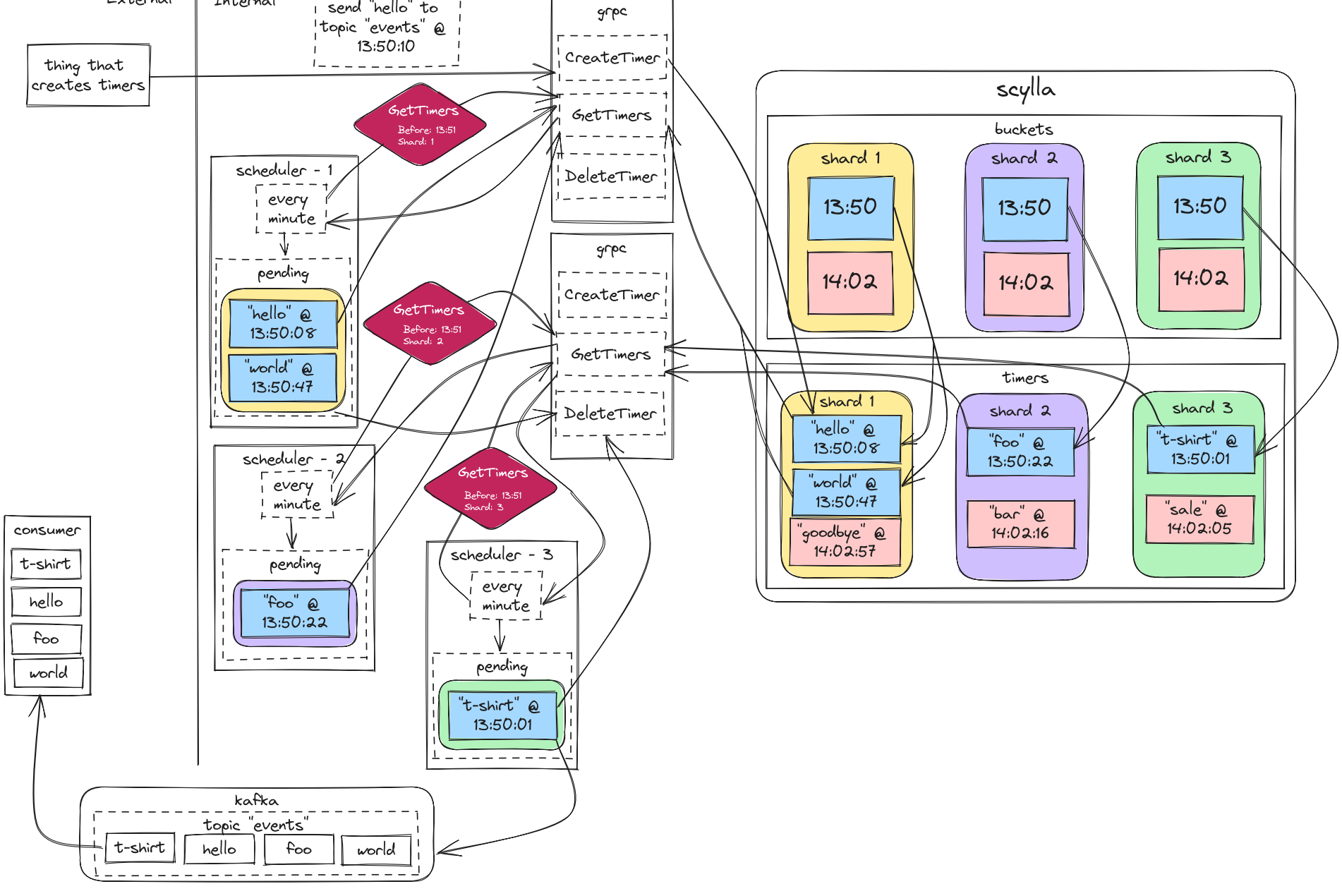
STATEFUL SET

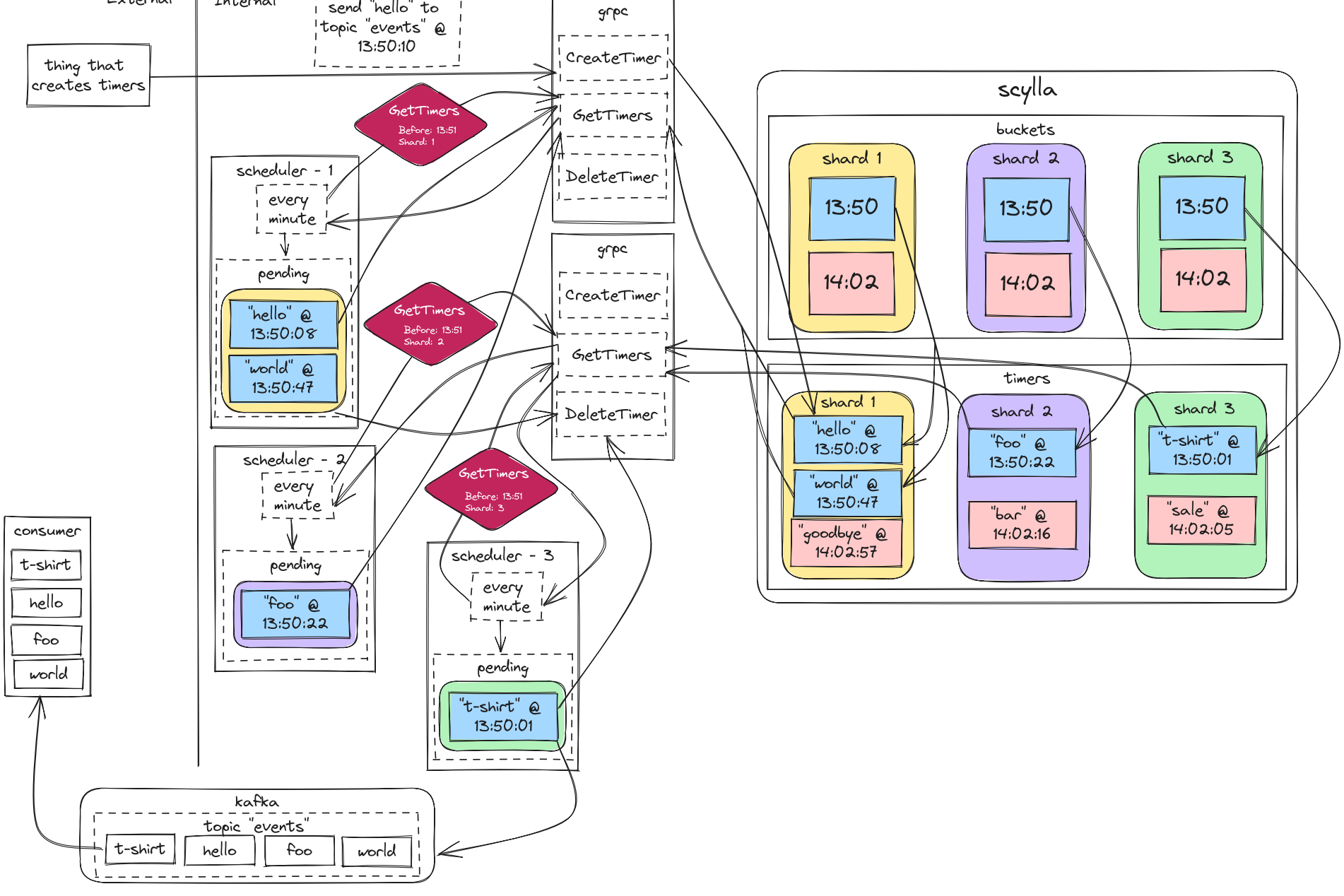
STATEFUL SET

- Stable, unique identifier

STATEFUL SET

- Stable, unique identifier
  - scheduler-0
  - scheduler-1
  - scheduler-2
  - scheduler-3...


External
Internal
send "hello" to topic "events" @ 13:50:10
thing that creates timers
grpc
CreateTimer
GetTimers
DeleteTimer
GetTimers
Before: 13:51
Shard: 1
scheduler - 1
every minute
pending
"hello" @ 13:50:08
"world" @ 13:50:47
grpc
CreateTimer
GetTimers
DeleteTimer
GetTimers
Before: 13:51
Shard: 2
scheduler - 2
every minute
pending
"foo" @ 13:50:22
GetTimers
Before: 13:51
Shard: 3
scheduler - 3
every minute
pending
"t-shirt" @ 13:50:01
scylla
buckets
shard 1
13:50
14:02
shard 2
13:50
14:02
shard 3
13:50
14:02
timers
shard 1
"hello" @ 13:50:08
"world" @ 13:50:47
"goodbye" @ 14:02:57
shard 2
"foo" @ 13:50:22
"bar" @ 14:02:16
shard 3
"t-shirt" @ 13:50:01
"sale" @ 14:02:05
consumer
t-shirt
hello
foo
world
kafka
topic "events"
t-shirt
hello
foo
world



We've achieved full scalability!

PERFORMANCE CHARACTERISTICS

PERFORMANCE CHARACTERISTICS

- Simple scaling

PERFORMANCE CHARACTERISTICS

- Simple scaling
- Less susceptible to serious outage

PERFORMANCE CHARACTERISTICS

- Simple scaling
- Less susceptible to serious outage
- 10k timers per second per scheduler node

PERFORMANCE CHARACTERISTICS

- Simple scaling
- Less susceptible to serious outage
- 10k timers per second per scheduler node
- 17k requests per second per grpc node

CALLOUTS

CALLOUTS

- At-least-once guarantee

CALLOUTS

- At-least-once guarantee
- Fire close-to scheduled time

CALLOUTS

- At-least-once guarantee
- Fire close-to scheduled time
- Cannot be cancelled after retrieved

FUTURE POTENTIAL

FUTURE POTENTIAL

- Open source

FUTURE POTENTIAL

- Open source
- Integrate more broadly

FUTURE POTENTIAL

- Open source
- Integrate more broadly
- Add features

FUTURE POTENTIAL

- Open source
- Integrate more broadly
- Add features
    - Cancelling timers possible always

WHAT WE HAVE

WHAT WE HAVE

- Store billions of timers

WHAT WE HAVE

- Store billions of timers
- Expires performantly

WHAT WE HAVE

- Store billions of timers
- Expires performantly
- Minimize data loss

WHAT WE HAVE

- Store billions of timers
- Expires performantly
- Minimize data loss
- Easy integration

WHAT WE HAVE

- Store billions of timers
- Expires performantly
- Minimize data loss
- Easy integration
- Simple scaling