

DRIVETRIBE ENGINEERING

A SOCIAL NETWORK ON STREAMS

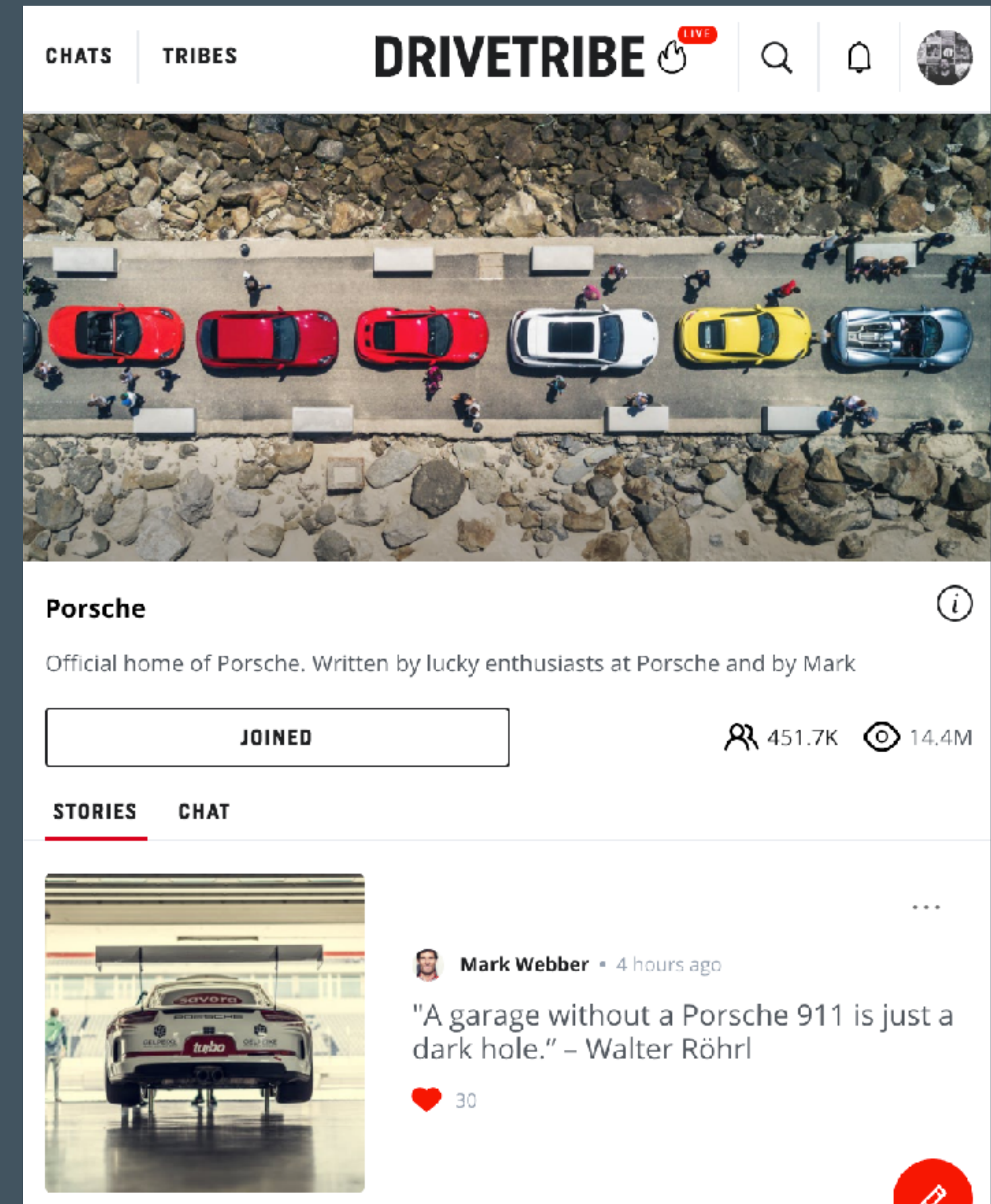
DRIVETRIBE

- ▶ **The world biggest motoring community.**
- ▶ **A social platform for petrolheads.**
- ▶ **By Clarkson, Hammond and May.**







DRIVETRIBE

- ▶ A content destination at the core.
- ▶ Users consume feeds of content: images, videos, long-form articles.
- ▶ Content is organised in homogenous categories called “tribes”.
- ▶ Different users have different interests and the tribe model allows to mix and match at will.






DRIVETRIBE ARTICLE


- ▶ Single article by James May.
- ▶ Contains a plethora of content and engagement information.
- ▶ What do we need to compute an aggregate like this?


DRIVETRIBE CHATS TRIBES    

GOODBYE GAS STATION

 James May  posted in **JAMES MAY'S CARBOLICS** 

3 months ago • 187.7K Views





COMMENTS (352) REPOST BUMPS (366) 






- I'm looking forward to the first scandal, so we can talk about 'Elongate'.

DRIVETRIBE ARTICLE


▶ `getUser(id: Id[User])`


DRIVETRIBE CHATS TRIBES    

GOODBYE GAS STATION

 James May  posted in **JAMES MAY'S CARBOLICS** 

3 months ago • 187.7K Views





COMMENTS (352) REPOST BUMPS (366) 






- I'm looking forward to the first scandal, so we can talk about 'Elongate'.

DRIVETRIBE ARTICLE


- ▶ `getUser(id: Id[User])`
- ▶ `getTribe(id: Id[Tribe])`


DRIVETRIBE CHATS TRIBES    

GOODBYE GAS STATION

 James May  posted in **JAMES MAY'S CARBOLICS** 

3 months ago • 187.7K Views





COMMENTS (352) REPOST BUMPS (366) 






- I'm looking forward to the first scandal, so we can talk about 'Elongate'.

DRIVETRIBE ARTICLE


- ▶ `getUser(id: Id[User])`
- ▶ `getTribe(id: Id[Tribe])`
- ▶ `getArticle(id: Id[Article])`


DRIVETRIBE CHATS TRIBES    

GOODBYE GAS STATION

 James May  posted in **JAMES MAY'S CARBOLICS** 

3 months ago • 187.7K Views

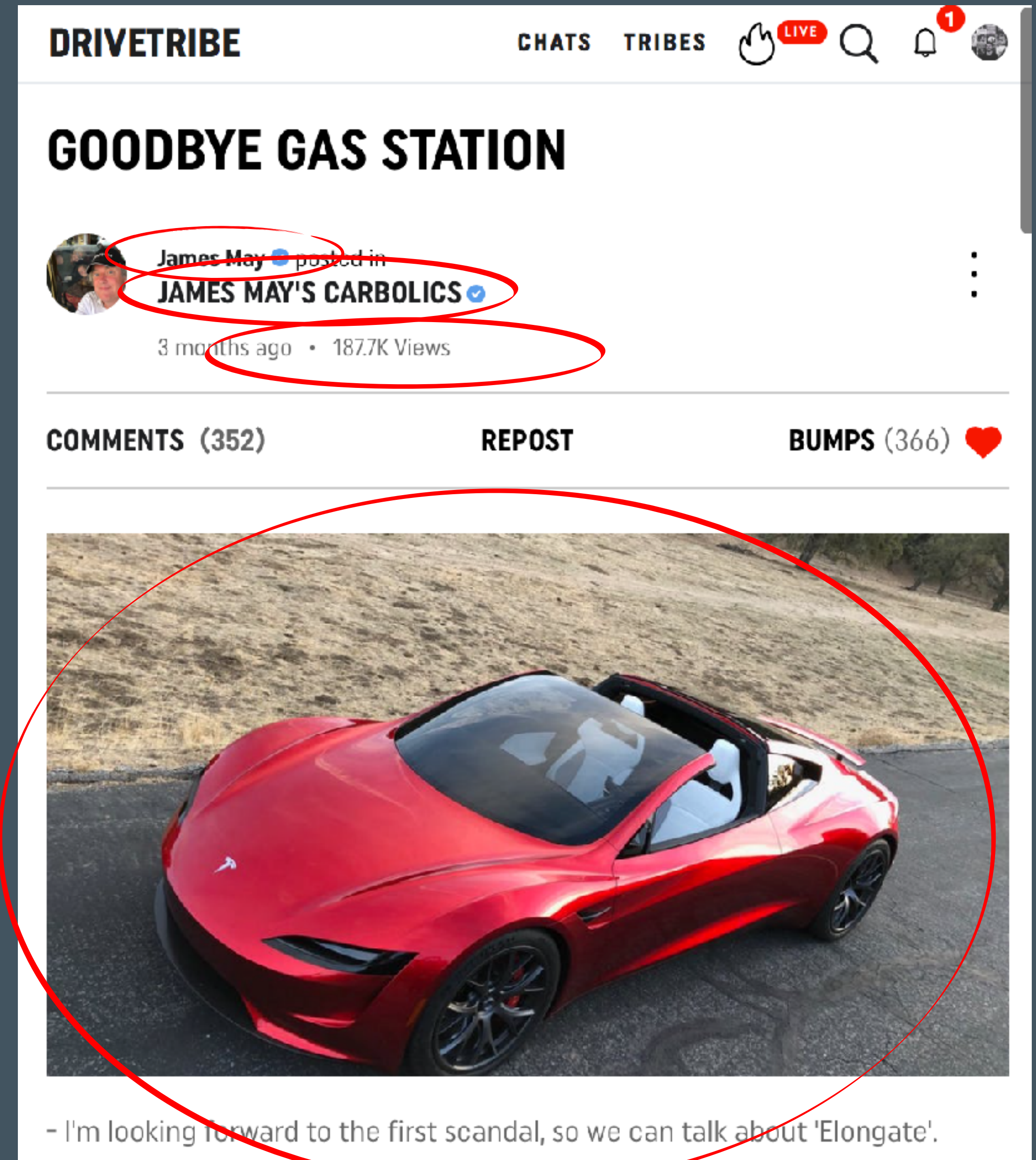
COMMENTS (352) REPOST BUMPS (366) 



- I'm looking forward to the first scandal, so we can talk about 'Elongate'.

DRIVETRIBE ARTICLE

- ▶ `getUser(id: Id[User])`
- ▶ `getTribe(id: Id[Tribe])`
- ▶ `getArticle(id: Id[Article])`
- ▶ `countViews(id: Id[Article])`




DRIVETRIBE CHATS TRIBES LIVE 1

GOODBYE GAS STATION

James May posted in **JAMES MAY'S CARBOLICS**
3 months ago • 187.7K Views

COMMENTS (352) REPOST BUMPS (366)



- I'm looking forward to the first scandal, so we can talk about 'Elongate'.

DRIVETRIBE ARTICLE

- ▶ `getUser(id: Id[User])`
- ▶ `getTribe(id: Id[Tribe])`
- ▶ `getArticle(id: Id[Article])`
- ▶ `countViews(id: Id[Article])`
- ▶ `countComments(id: Id[Article])`

DRIVETRIBE CHATS TRIBES LIVE 1

GOODBYE GAS STATION

James May posted in **JAMES MAY'S CARBOLICS**
3 months ago • 187.7K Views

COMMENTS (352) REPOST BUMPS (366)

- I'm looking forward to the first scandal, so we can talk about 'Elongate'.

DRIVETRIBE ARTICLE

- ▶ `getUser(id: Id[User])`
- ▶ `getTribe(id: Id[Tribe])`
- ▶ `getArticle(id: Id[Article])`
- ▶ `countViews(id: Id[Article])`
- ▶ `countComments(id: Id[Article])`
- ▶ `countBumps(id: Id[Article])`

DRIVETRIBE CHATS TRIBES LIVE 1

GOODBYE GAS STATION

James May posted in **JAMES MAY'S CARBOLICS**
3 months ago • 187.7K Views

COMMENTS (352) REPOST BUMPS (366)

- I'm looking forward to the first scandal, so we can talk about 'Elongate'.

DRIVETRIBE FEED OF ARTICLES

- ▶ `rankArticles(forUserId).flatMap { a => ... }`
- ▶ `getUser(id: Id[User])`
- ▶ `getTribe(id: Id[Tribe])`
- ▶ `getArticle(id: Id[Article])`
- ▶ `countViews(id: Id[Article])`
- ▶ ...

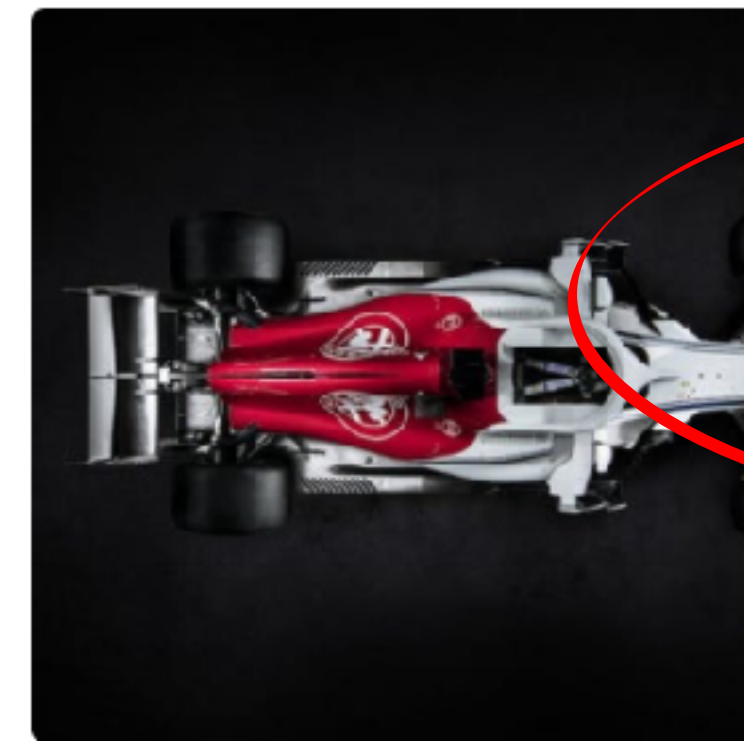


 Craig Scarborough in Everything Technical

F1: THE FERRARI SF71H IN DETAIL – PROGRESS ON ALL FRONTS

FERRARI'S NEWLY RELEASED F1 CAR SHOWS AGGRESSIVE DESIGN IN EVERY AREA IN ORDER TO CHASE MERCEDES FOR THE CROWN IN 2018

 129



 James May in James May's Carbolics

THE WEIGHT IS OVER

But it should be under, surely?

 168



 Craig Scarborough in Everything Technical

F1: THE MERCEDES W09 IN DETAIL, MAKING THE BEST EVEN BETTER

A deep dive into what's new and what's going to change on Mercedes new F1 challenger

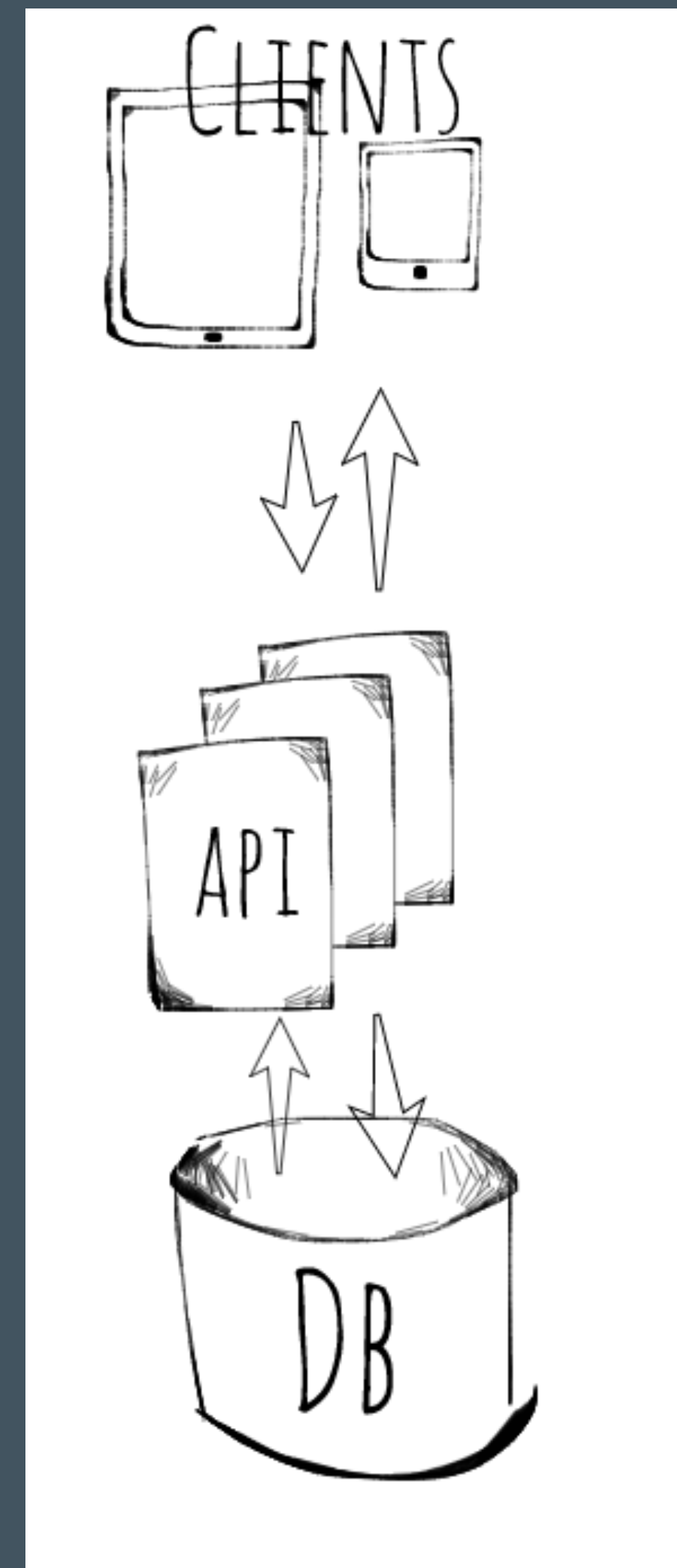
 69

QUINTESSENTIAL PREREQUISITES

- ▶ **Scalable.** Jeremy Clarkson has 7.2M Twitter followers. Cannot really hack it and worry about it later.
- ▶ **Performant.** Low latency is key and mobile networks add quite a bit of it.
- ▶ **Flexible.** Almost nobody gets it right the first time around. The ability to iterate is paramount.
- ▶ **Maintainable.** Spaghetti code works like interest on debt.

THREE TIER APPROACH

- ▶ Clients interact with a fleet of stateless servers (aka “API” servers or “Backend”) via HTTP (which is stateless).
- ▶ Global shared mutable state (aka the Database).
- ▶ Starting simple: Store data in a DB.
- ▶ Starting simple: Compute the aggregated views on the fly.



DRIVETRIBE ARTICLE


- ▶ `getUser(id: Id[User])`
- ▶ `getTribe(id: Id[Tribe])`
- ▶ `getArticle(id: Id[Article])`
- ▶ `countComments(id: Id[Article])`
- ▶ `countBumps(id: Id[Article])`
- ▶ `countViews(id: Id[Article])`

The screenshot shows the Drivetribe mobile app interface. At the top, the 'DRIVETRIBE' logo is on the left, and navigation icons for 'CHATS', 'TRIBES', 'LIVE', search, and notifications are on the right. The article title 'GOODBYE GAS STATION' is prominently displayed. Below the title, the author's profile 'JAMES MAY'S CARBOLICS' is shown with a verified badge. The article's metadata indicates it was posted '3 months ago' and has '187.7K Views'. The interaction bar at the bottom of the article header shows 'COMMENTS (352)', 'REPOST', and 'BUMPS (366)' with a heart icon. The main content area features a high-quality photograph of a red sports car, a McLaren GT, parked on a paved surface. Below the image, the beginning of the article text is visible: '- I'm looking forward to the first scandal, so we can talk about 'Elongate'.'

READ TIME AGGREGATION


- ▶ (6 queries per Item) x (Y items per page)
- ▶ Cost of ranking and personalisation.
- ▶ Quite some work at read time.
- ▶ Slow. Not really **Performant**.



 Craig Scarborough in Everything Technical

F1: THE FERRARI SF71H IN DETAIL – PROGRESS ON ALL FRONTS

FERRARI'S NEWLY RELEASED F1 CAR SHOWS AGGRESSIVE DESIGN IN EVERY AREA IN ORDER TO CHASE MERCEDES FOR THE CROWN IN 2018

 129



 James May in James May's Carbolics

THE WEIGHT IS OVER

But it should be under, surely?

 168



 Craig Scarborough in Everything Technical

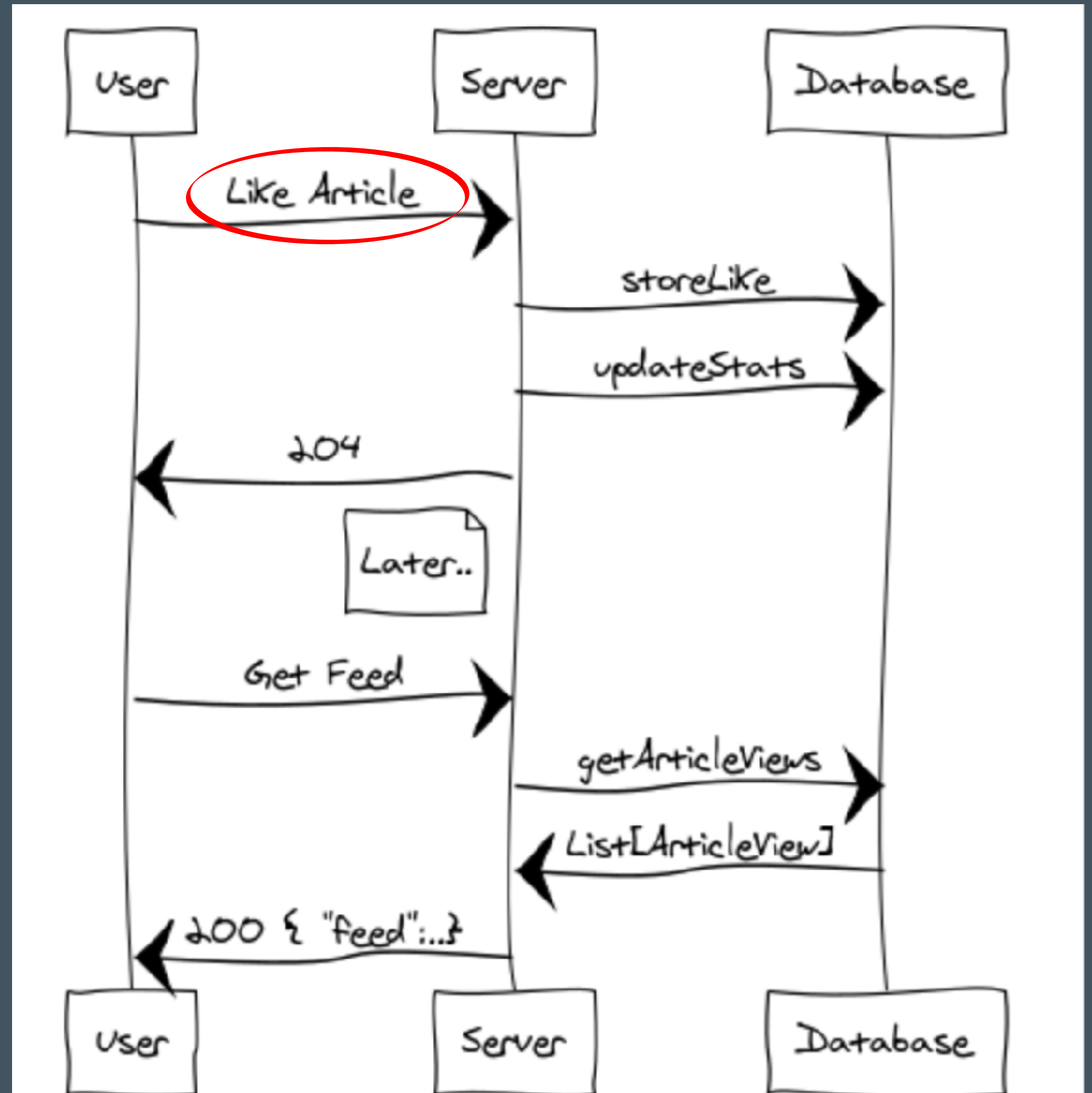
F1: THE MERCEDES W09 IN DETAIL, MAKING THE BEST EVEN BETTER

A deep dive into what's new and what's going to change on Mercedes new F1 challenger

 69

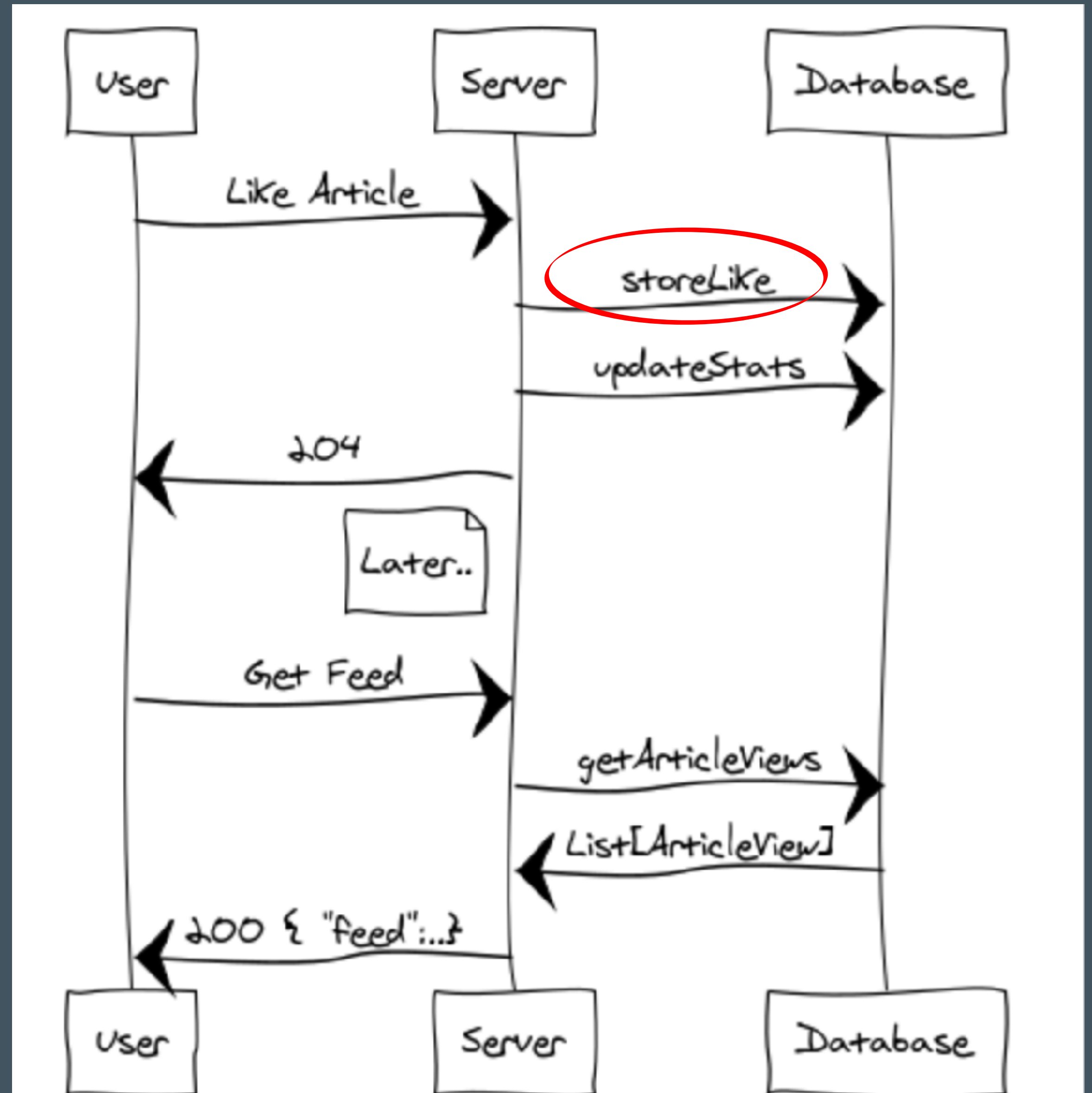
WRITE TIME AGGREGATION

- ▶ Compute the aggregation at write time.
- ▶ Then a single query can fetch all the views at once. That scales.



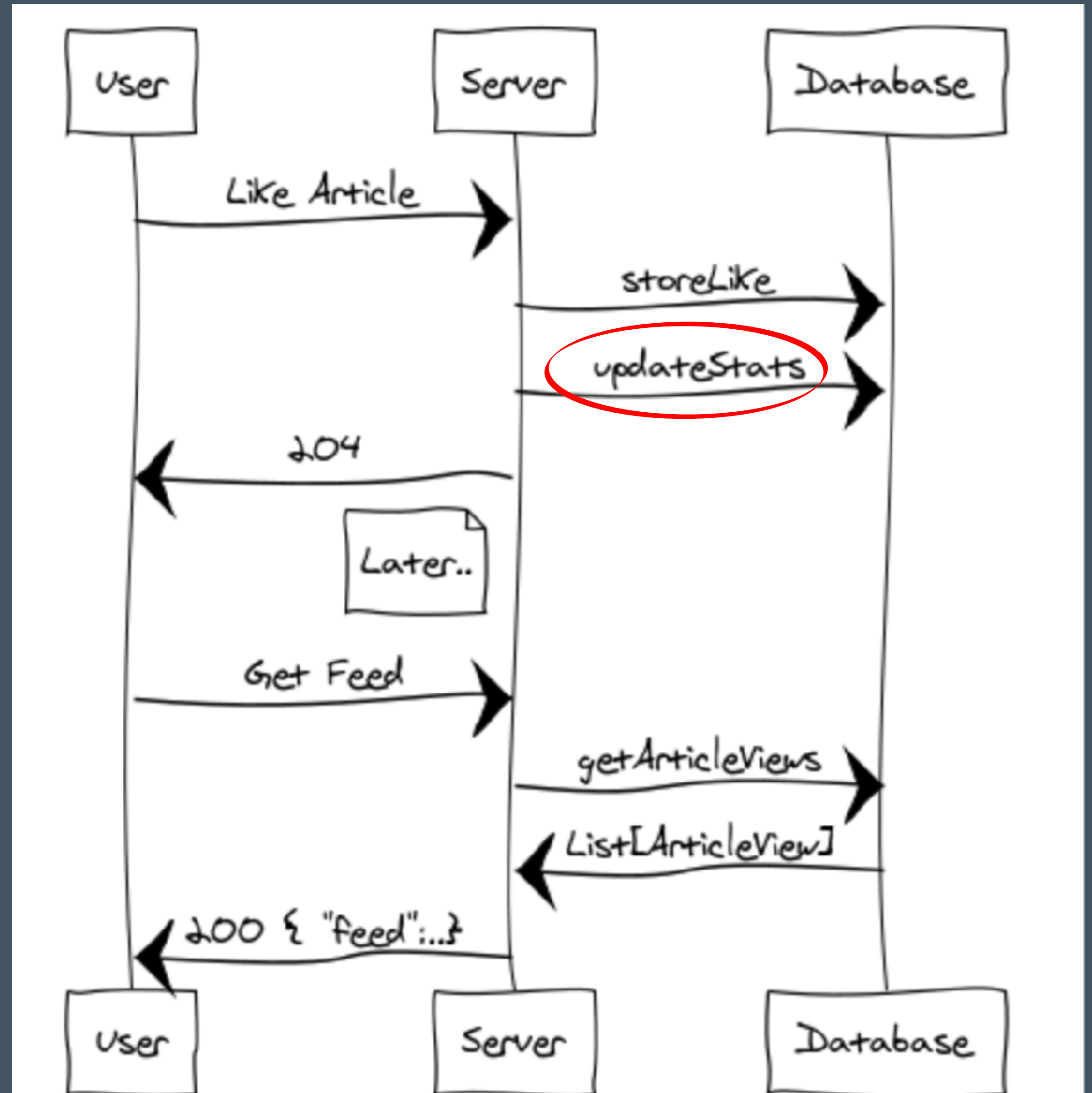
WRITE TIME AGGREGATION

- ▶ Compute the aggregation at write time.
- ▶ Then a single query can fetch all the views at once. That scales.



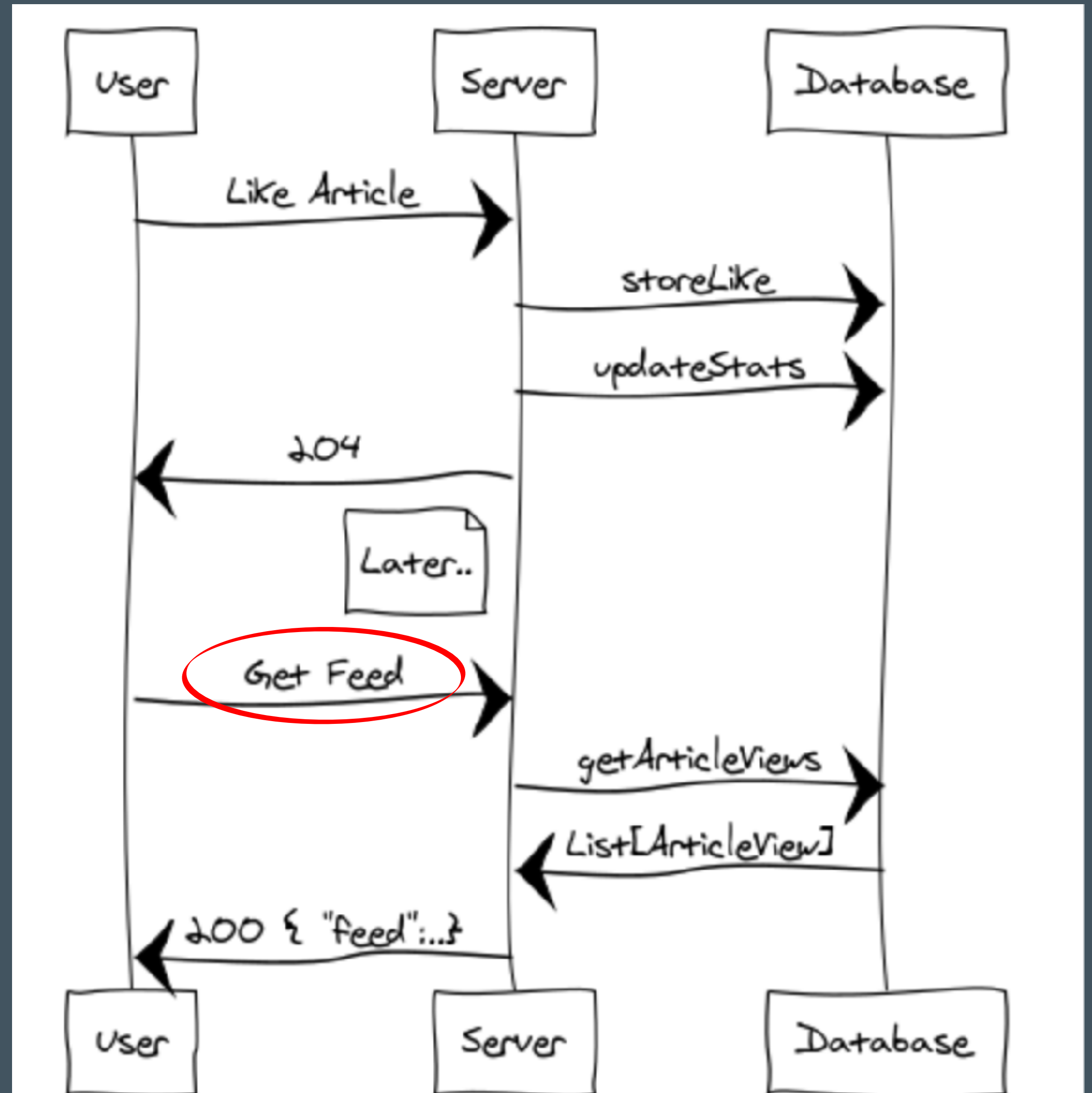
WRITE TIME AGGREGATION

- ▶ Compute the aggregation at write time.
- ▶ Then a single query can fetch all the views at once. That scales.



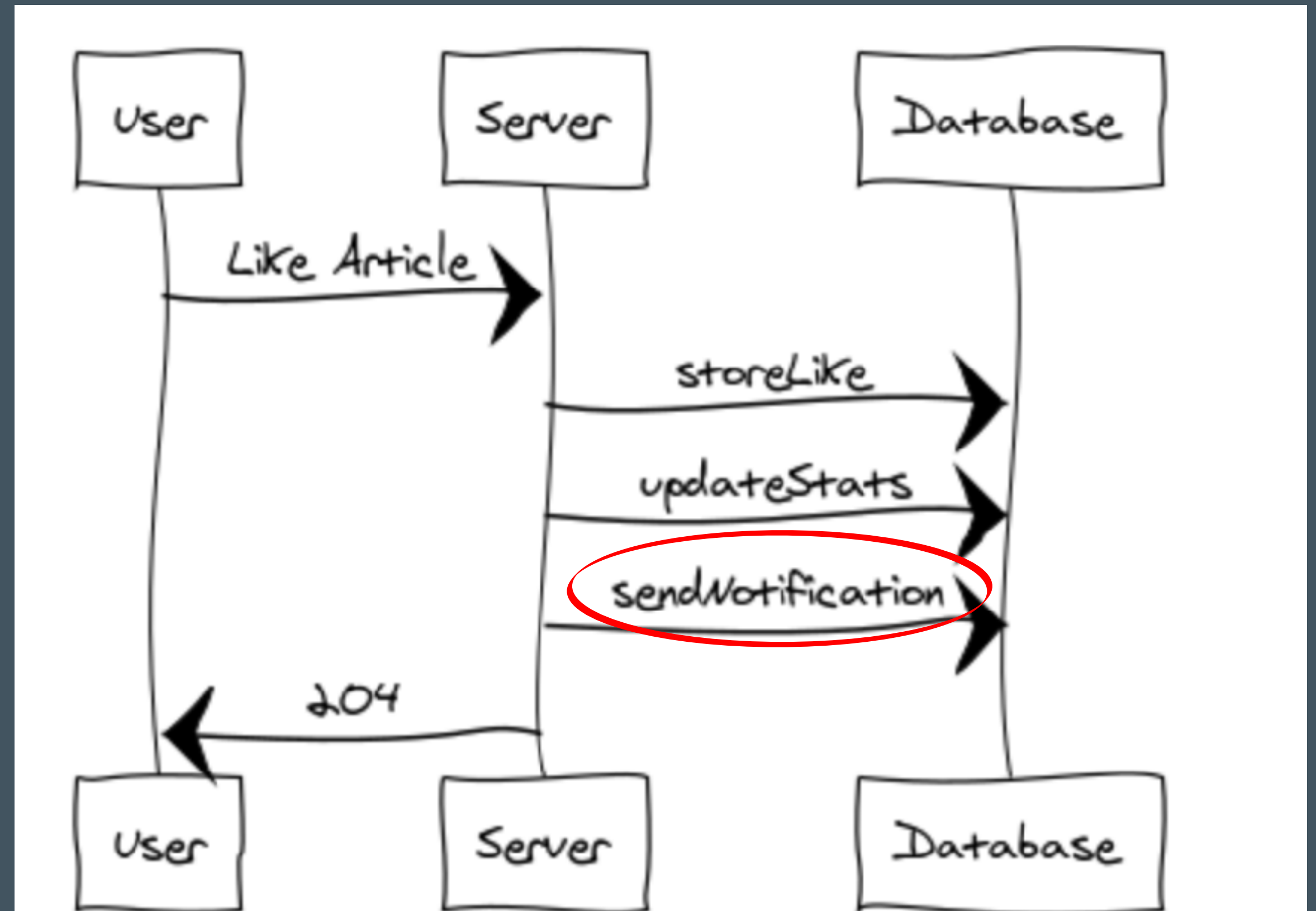
WRITE TIME AGGREGATION

- ▶ Compute the aggregation at write time.
- ▶ Then a single query can fetch all the views at once. That scales.



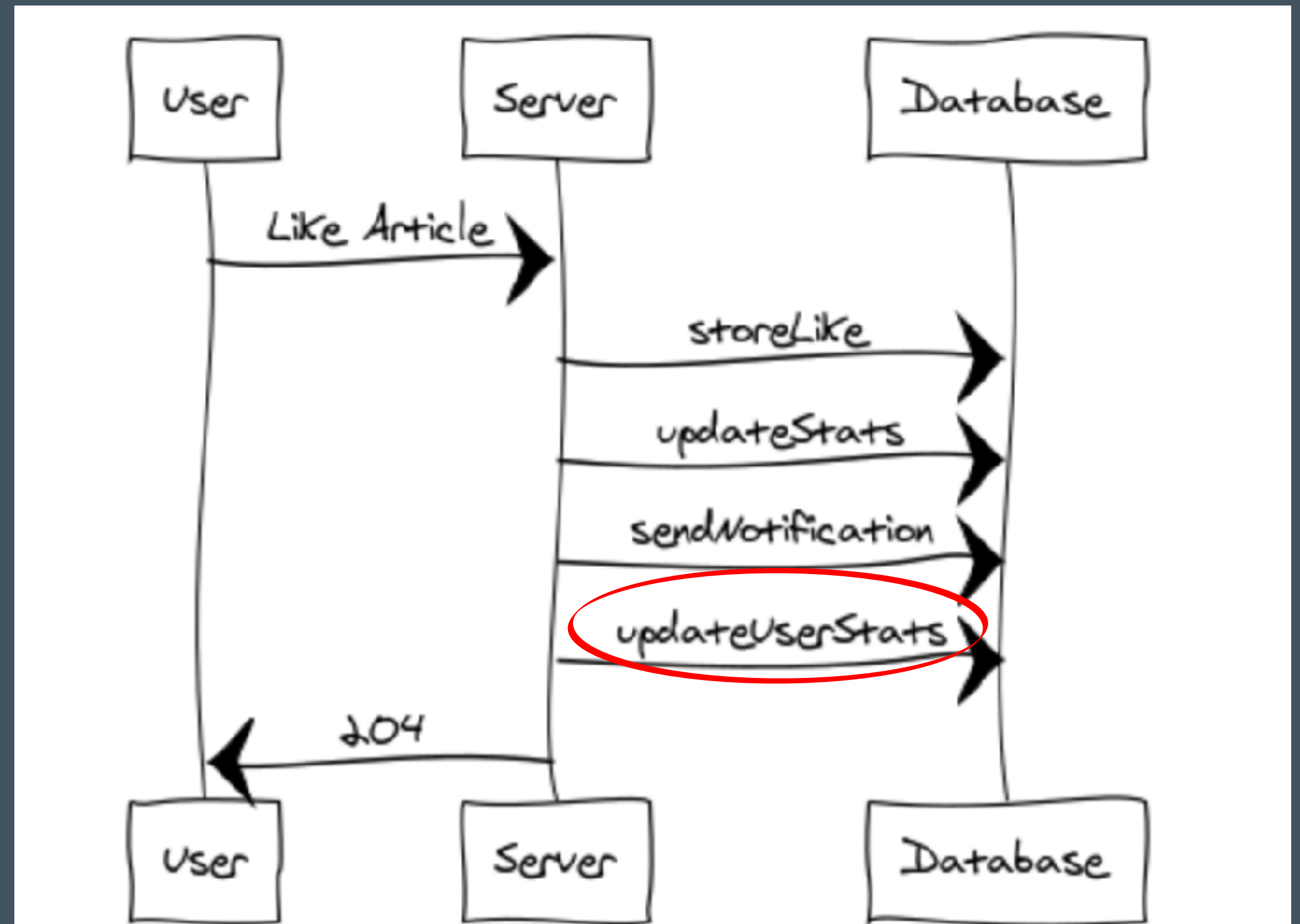
WRITE TIME AGGREGATION - EVOLUTION

- ▶ sendNotification



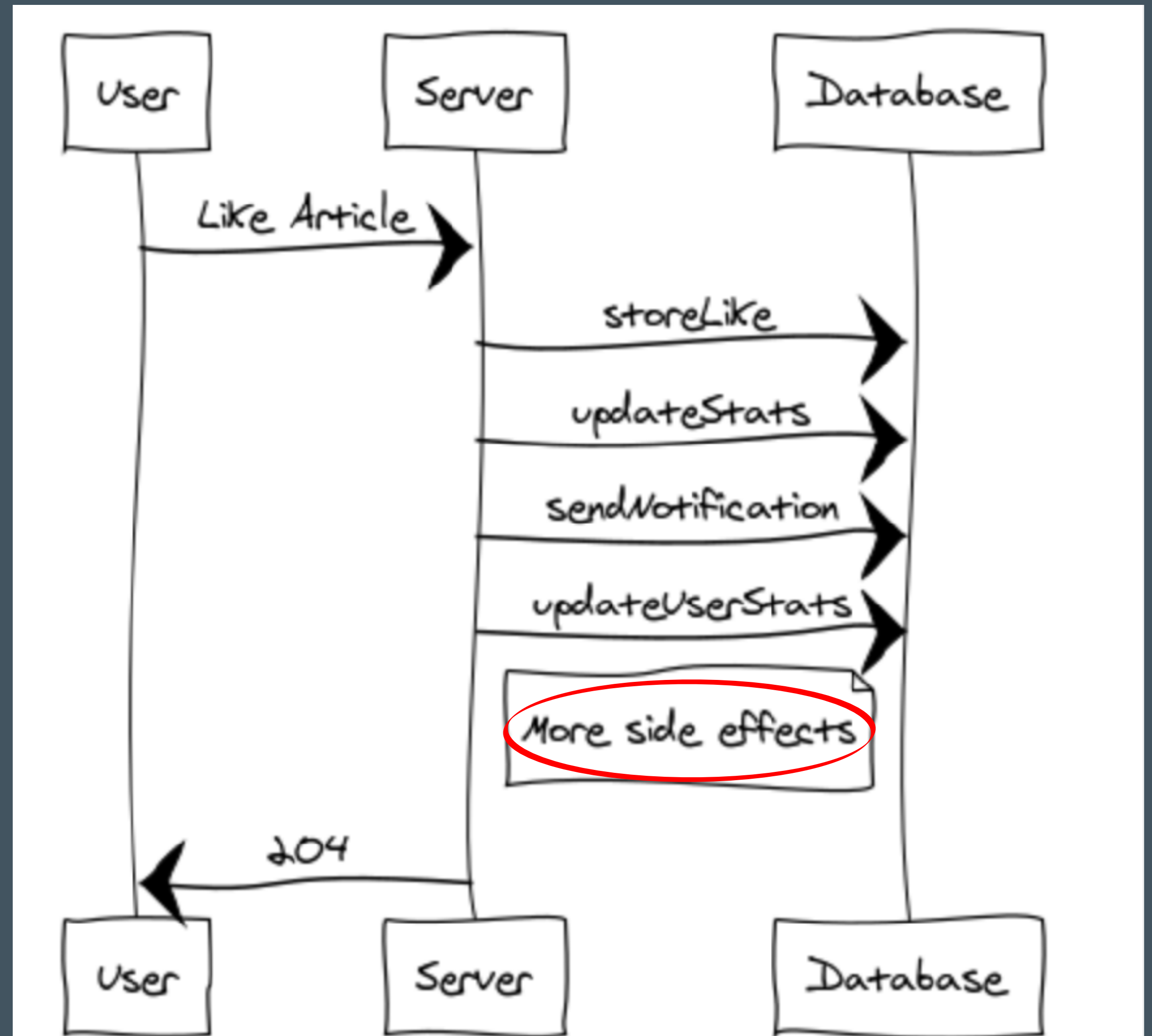
WRITE TIME AGGREGATION - EVOLUTION

- ▶ sendNotification
- ▶ updateUserStats



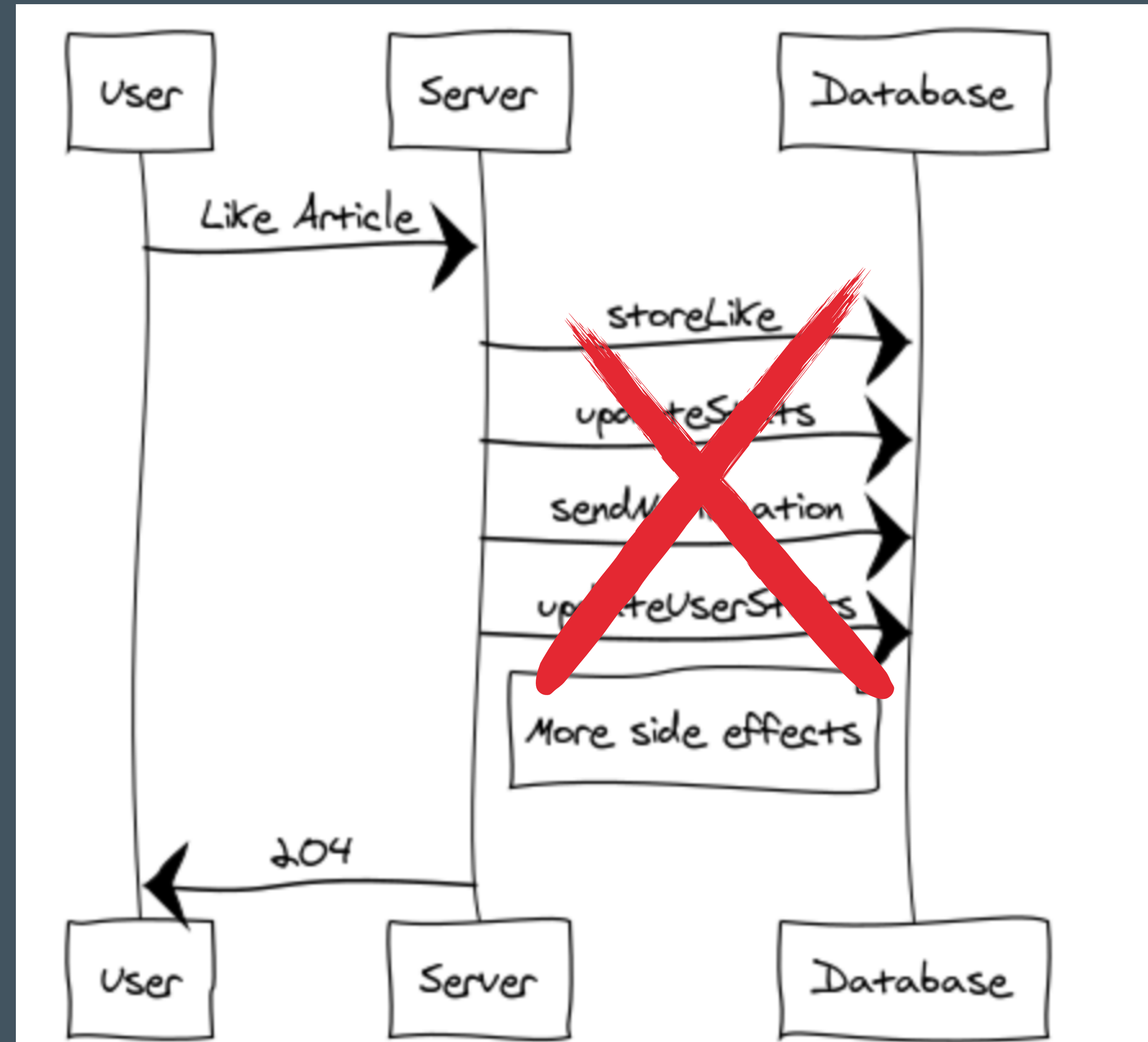
WRITE TIME AGGREGATION - EVOLUTION

- ▶ sendNotification
- ▶ updateUserStats.
- ▶ What if we have a cache?
- ▶ Or a different database for search?



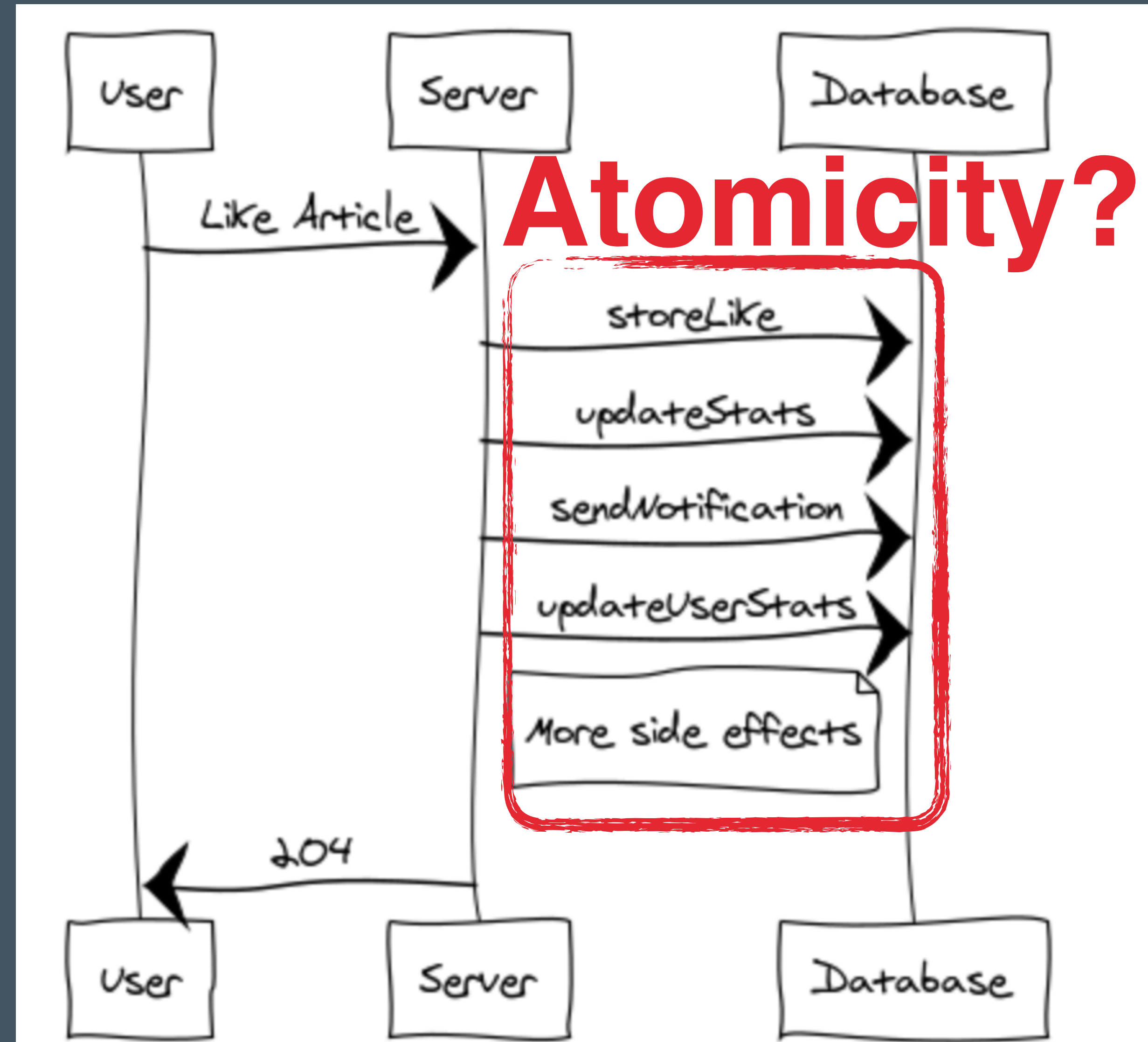
WRITE TIME AGGREGATION

- ▶ A simple user action is triggering a potentially endless sequence of side effects.
- ▶ Most of which need network IO.
- ▶ Many of which can **fail**.



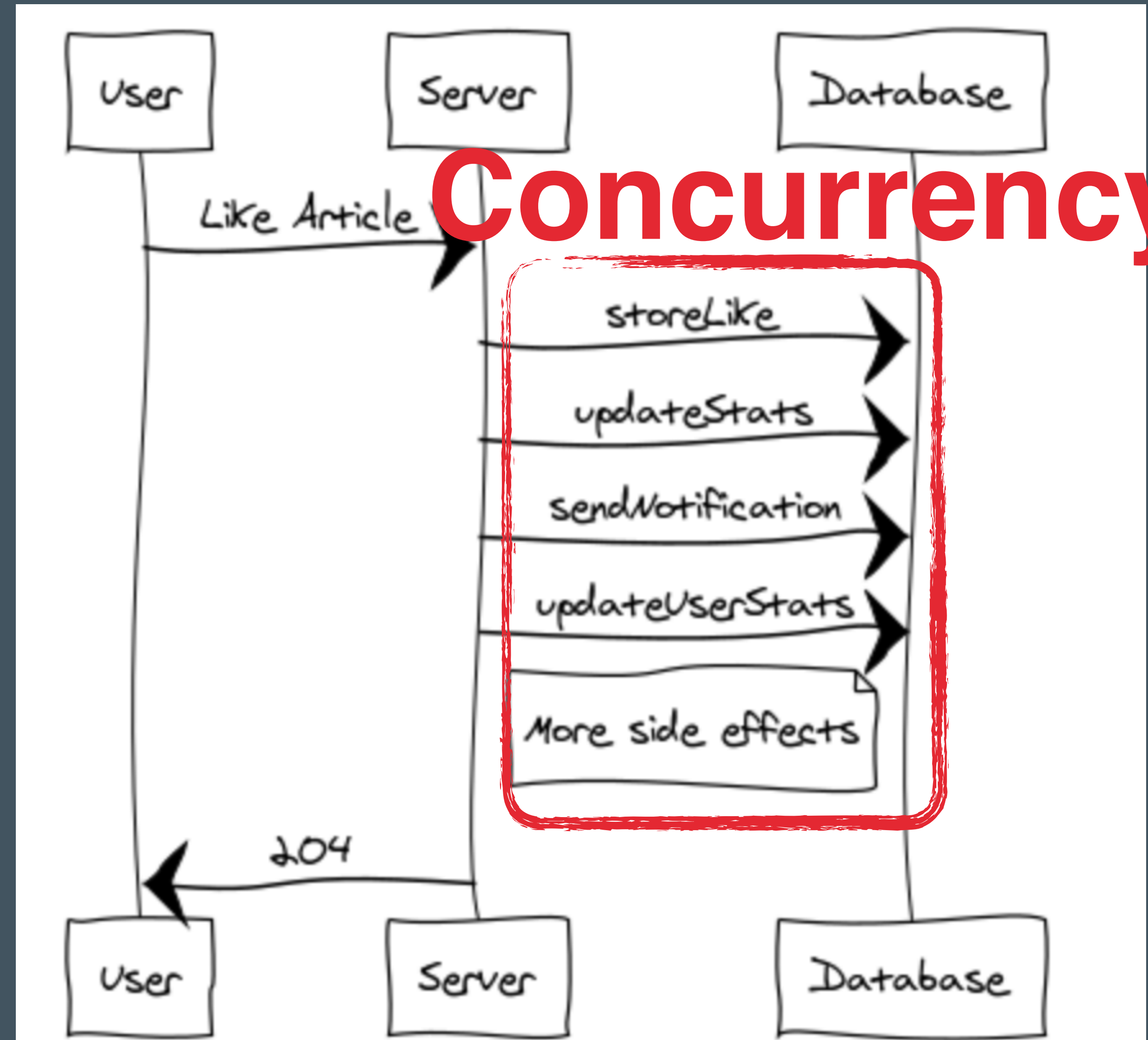
ATOMICITY

- ▶ What happens if one of them fails?
What happens if the server fails in the middle?
- ▶ We may have transaction support in the DB, but what about external systems?
- ▶ Inconsistent.



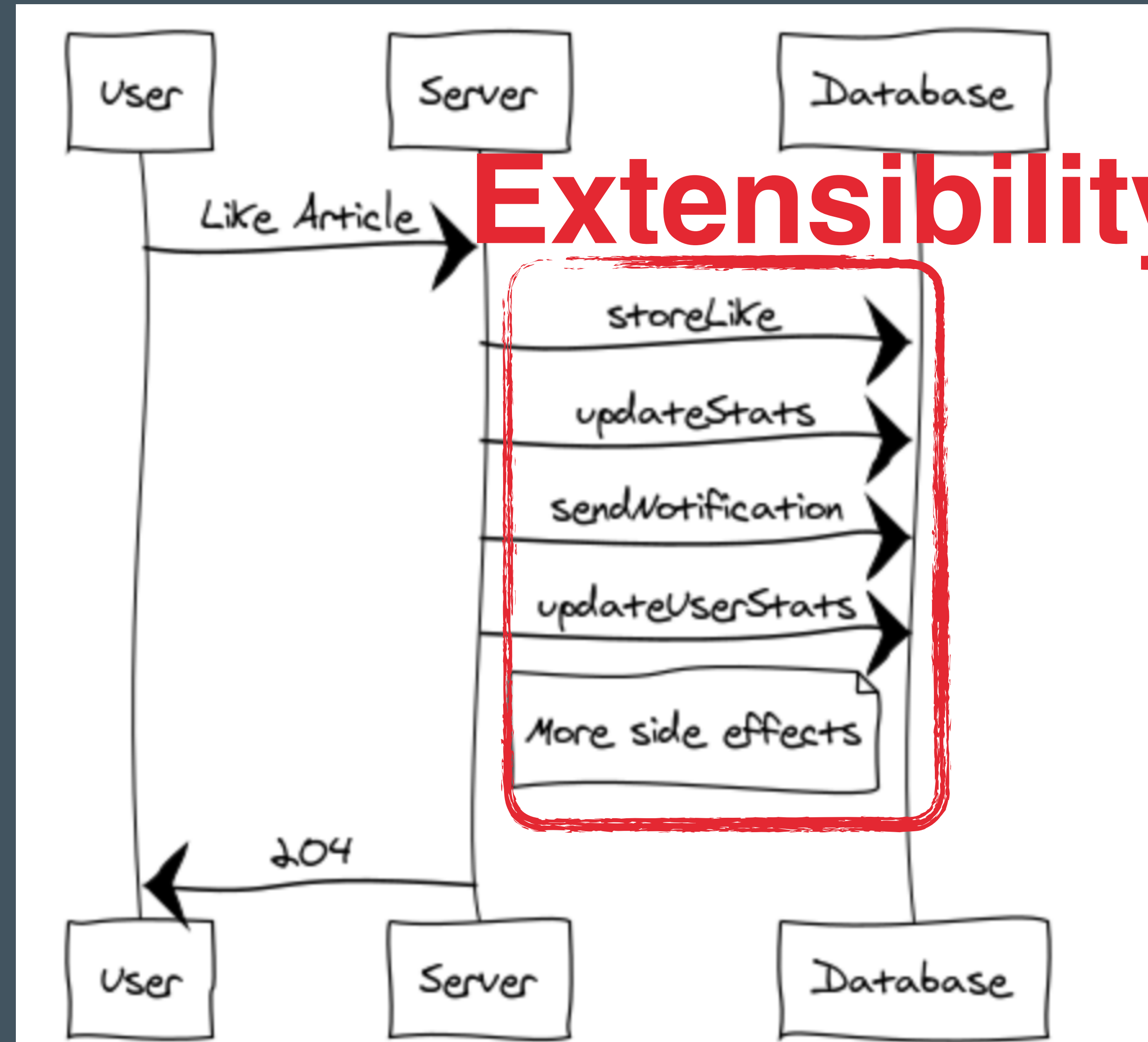
CONCURRENCY

- ▶ Concurrent mutations on a global shared state entail race conditions.
- ▶ State mutations are destructive and can not be (easily) undone.
- ▶ A bug can corrupt the data permanently.



ITALIAN PASTA

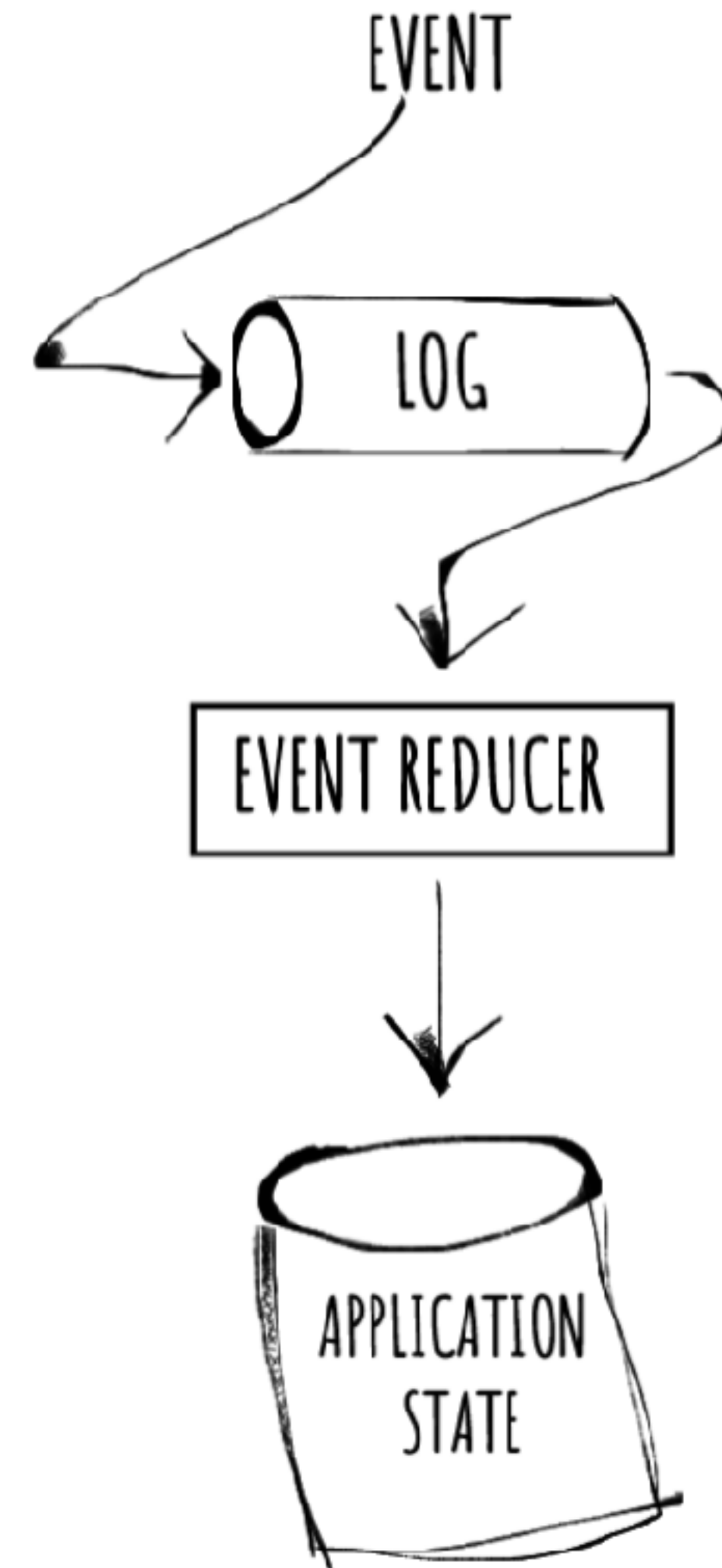
- ▶ Model evolution becomes difficult. Reads and writes are tightly coupled.
- ▶ Migrations are scary.
- ▶ This is neither **Extensible** nor **Maintainable**.



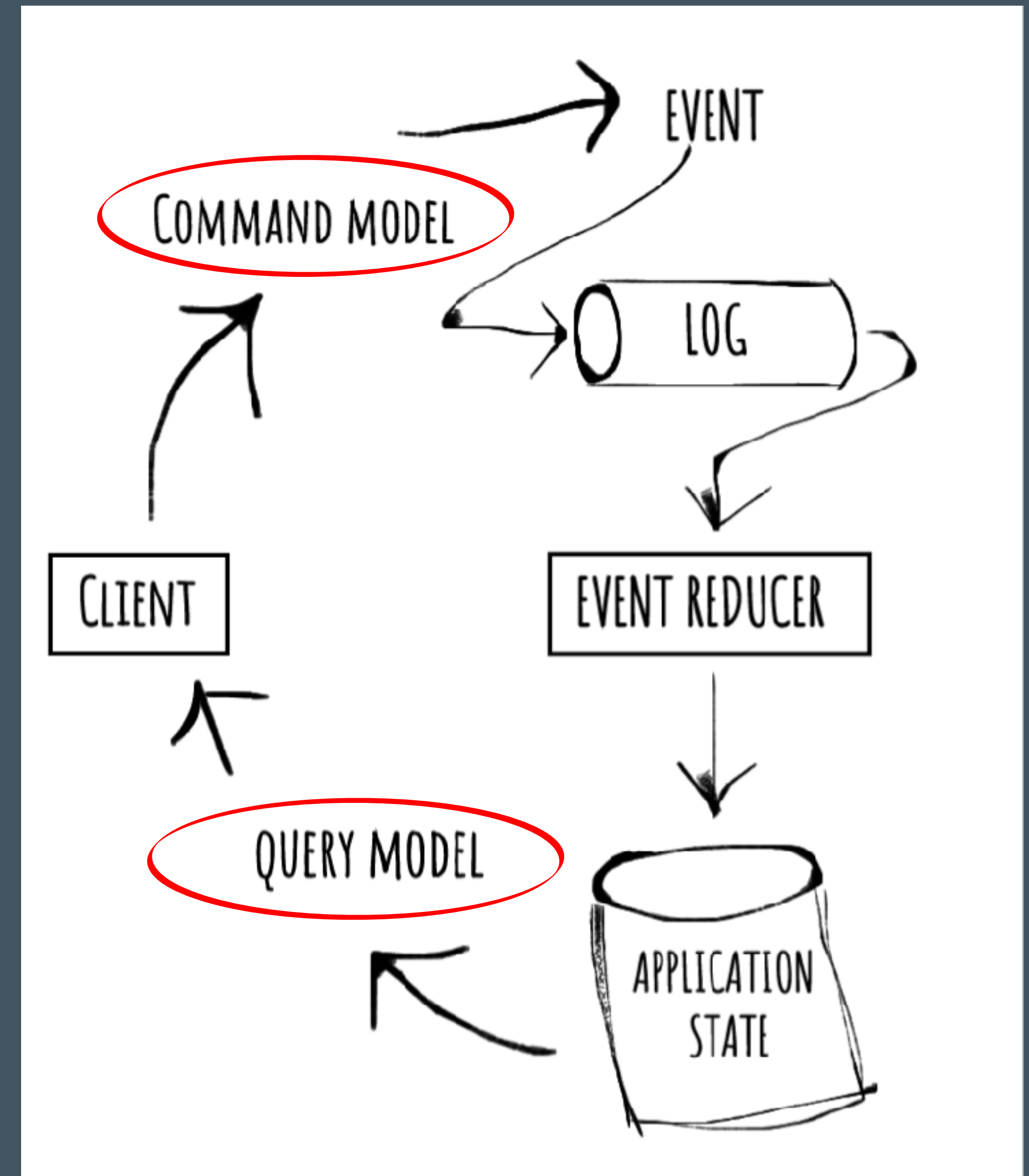
DIFFERENT APPROACH

- ▶ Let's take a step back and try to decouple things.
- ▶ Clients send events to the API: “John liked Jeremy’s post”, “Maria updated her profile”
- ▶ Events are **immutable**. They capture a user action at some point in time.
- ▶ Every application state instance can be modelled as a projection of those events.

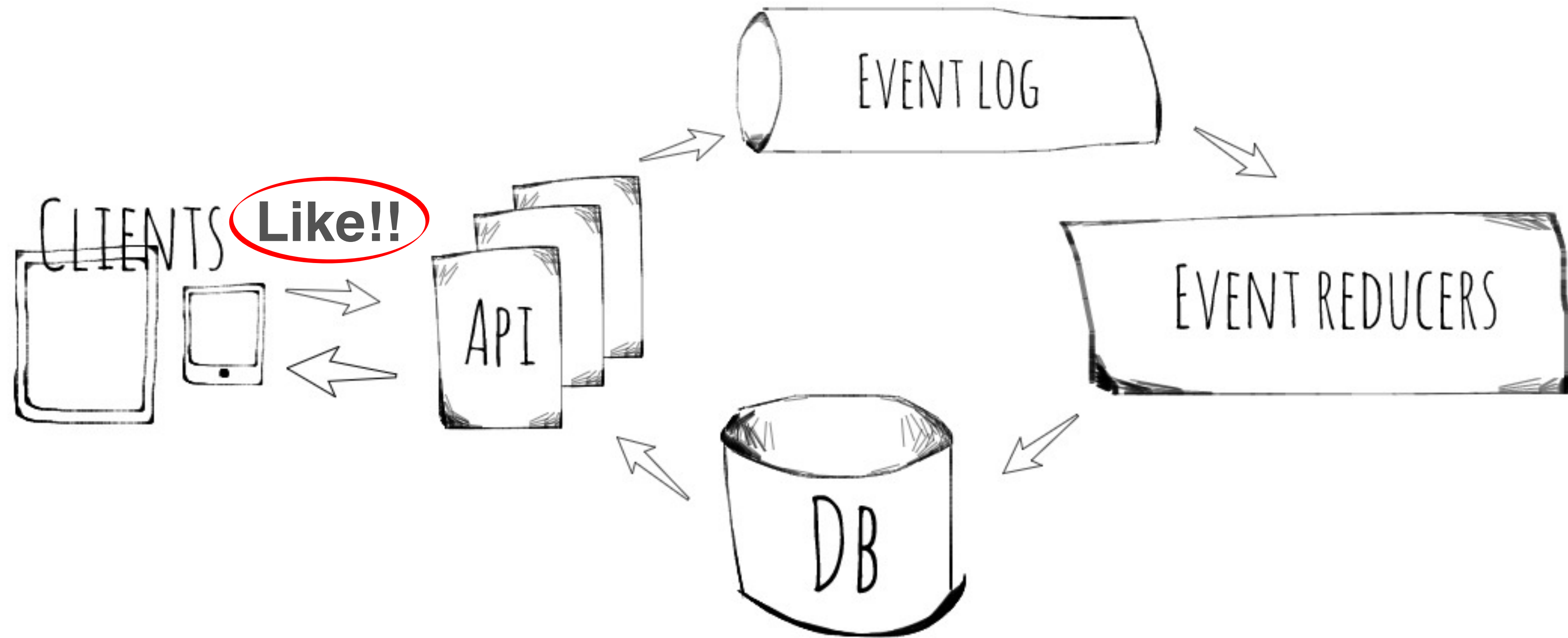
- ▶ Persisting those yields an append-only log of events.
- ▶ An event reducer can then produce application state instances.
- ▶ Even retroactively. The log is **immutable**.
- ▶ This is **event sourcing**.



- ▶ The write-time model (command model) and the read time model (query model) can be separated.
- ▶ Decoupling the two models opens the door to more efficient, custom implementations.
- ▶ This is known as Command Query Responsibility Segregation aka **CQRS**.



EVENT SOURCING APPROACH



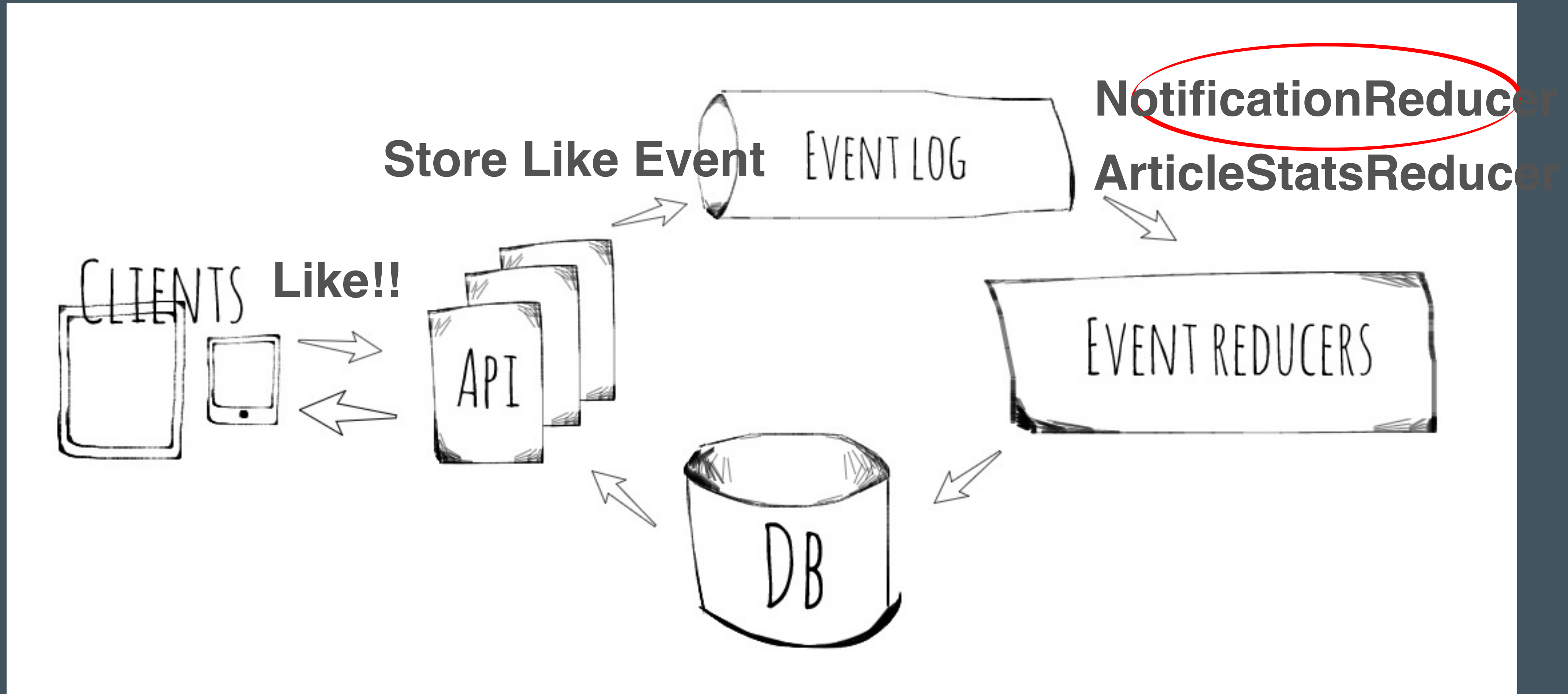
EVENT SOURCING APPROACH



EVENT SOURCING APPROACH



EVENT SOURCING APPROACH



EVENT SOURCING APPROACH



EVENT SOURCING APPROACH



EVENT SOURCING APPROACH



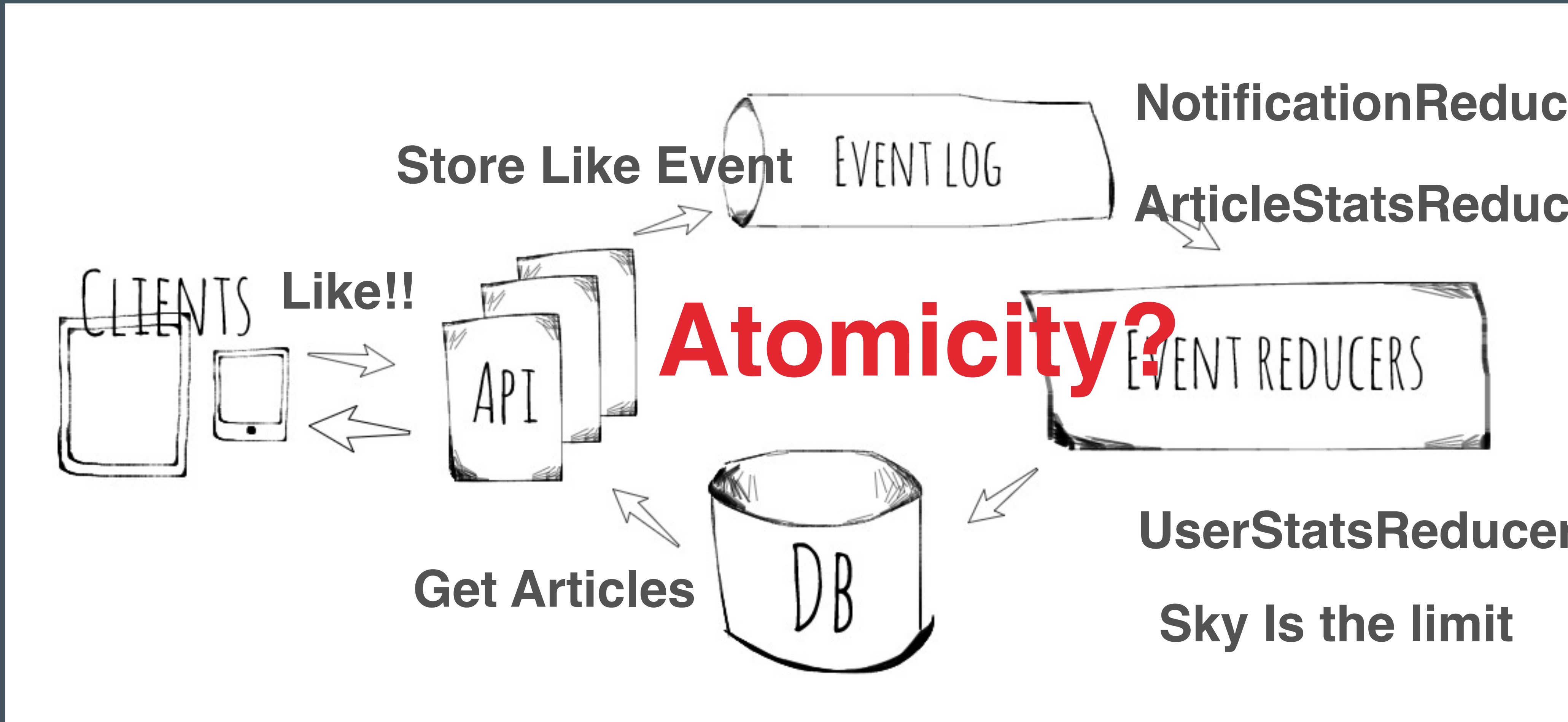
EVENT SOURCING APPROACH



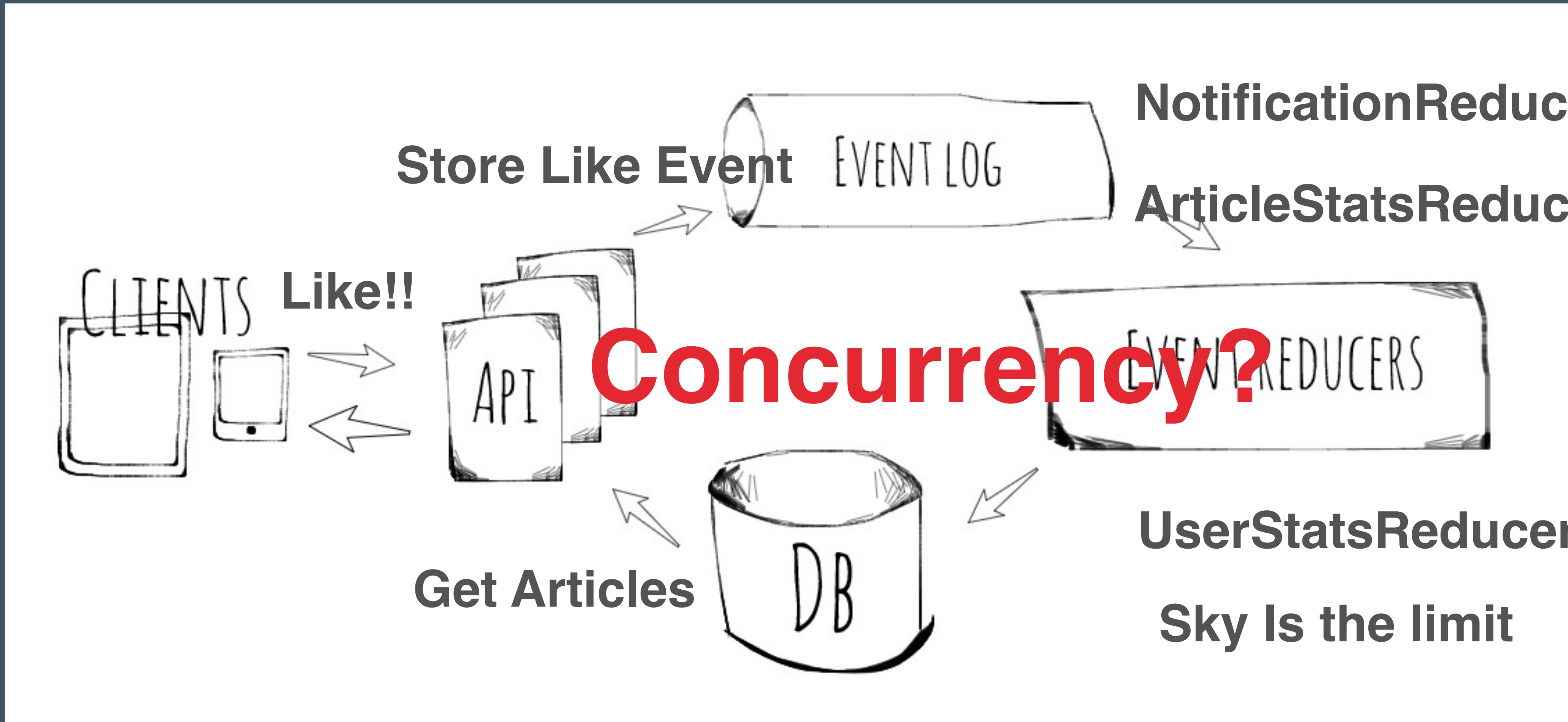
EVENT SOURCING APPROACH



EVENT SOURCING APPROACH



EVENT SOURCING APPROACH



IMPLEMENTATION?

WE NEED A LOG

APACHE KAFKA

- ▶ **Distributed, fault-tolerant, durable and fast append-only log**
- ▶ **Can scale the thousands of nodes, producers and consumers**
- ▶ **Each business event type can be stored in its own topic.**



**WE NEED A STREAM
PROCESSOR**

APACHE FLINK

- ▶ **Scalable, performant, mature.**
- ▶ **Elegant high level APIs in Scala.**
- ▶ **Powerful low level APIs for advanced tuning.**
- ▶ **Multiple battle-tested integrations.**
- ▶ **Very nice and active community.**



WE NEED DATASTORE

ELASTICSEARCH

- ▶ **Horizontally scalable document store.**
- ▶ **Rich and expressive query language.**
- ▶ **Dispensable. Can be replaced.**



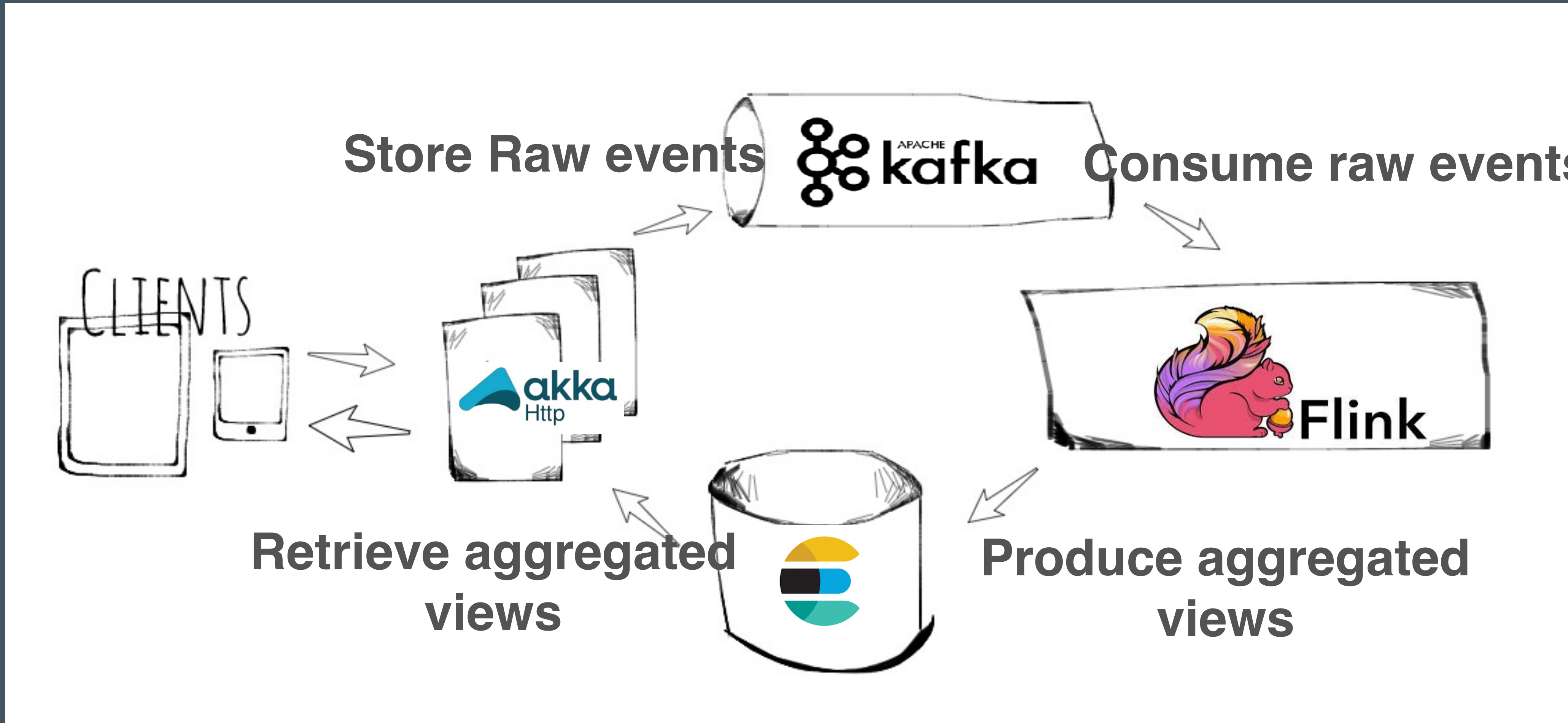
WE NEED AN API

AKKA HTTP

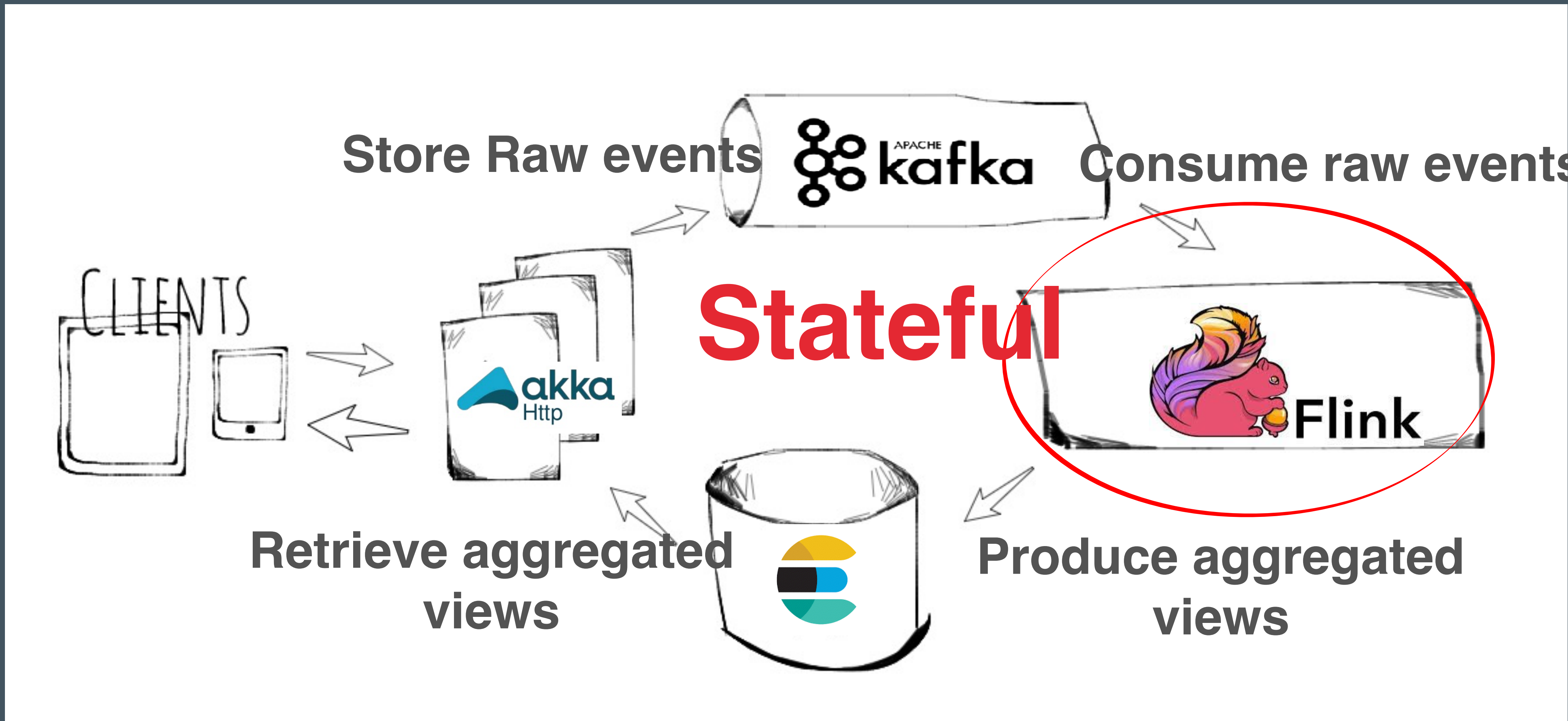
- ▶ **Asynchronous web application framework.**
- ▶ **Written in Scala.**
- ▶ **Very expressive routing DSL.**
- ▶ **Any modern web application framework would do.**



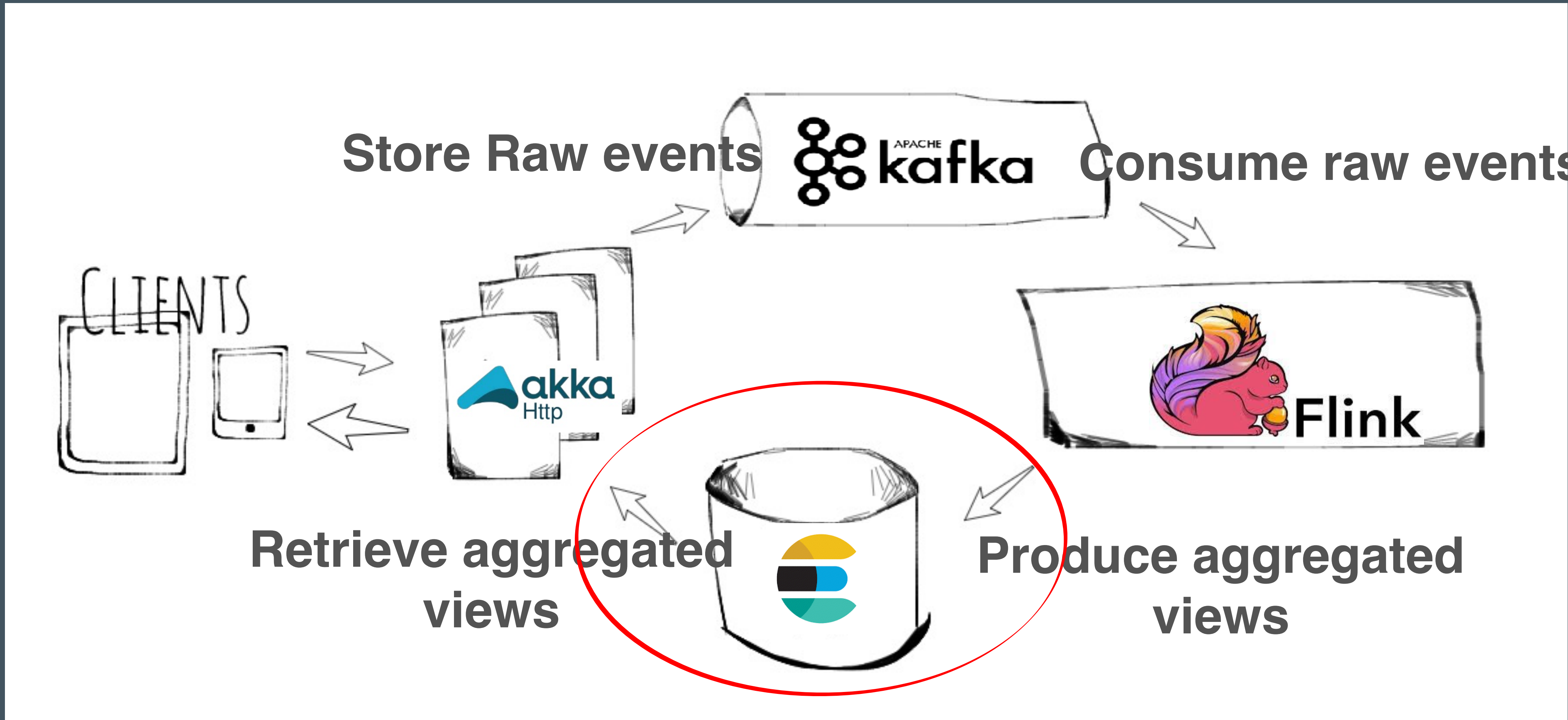
EVENT SOURCING IN PRACTICE



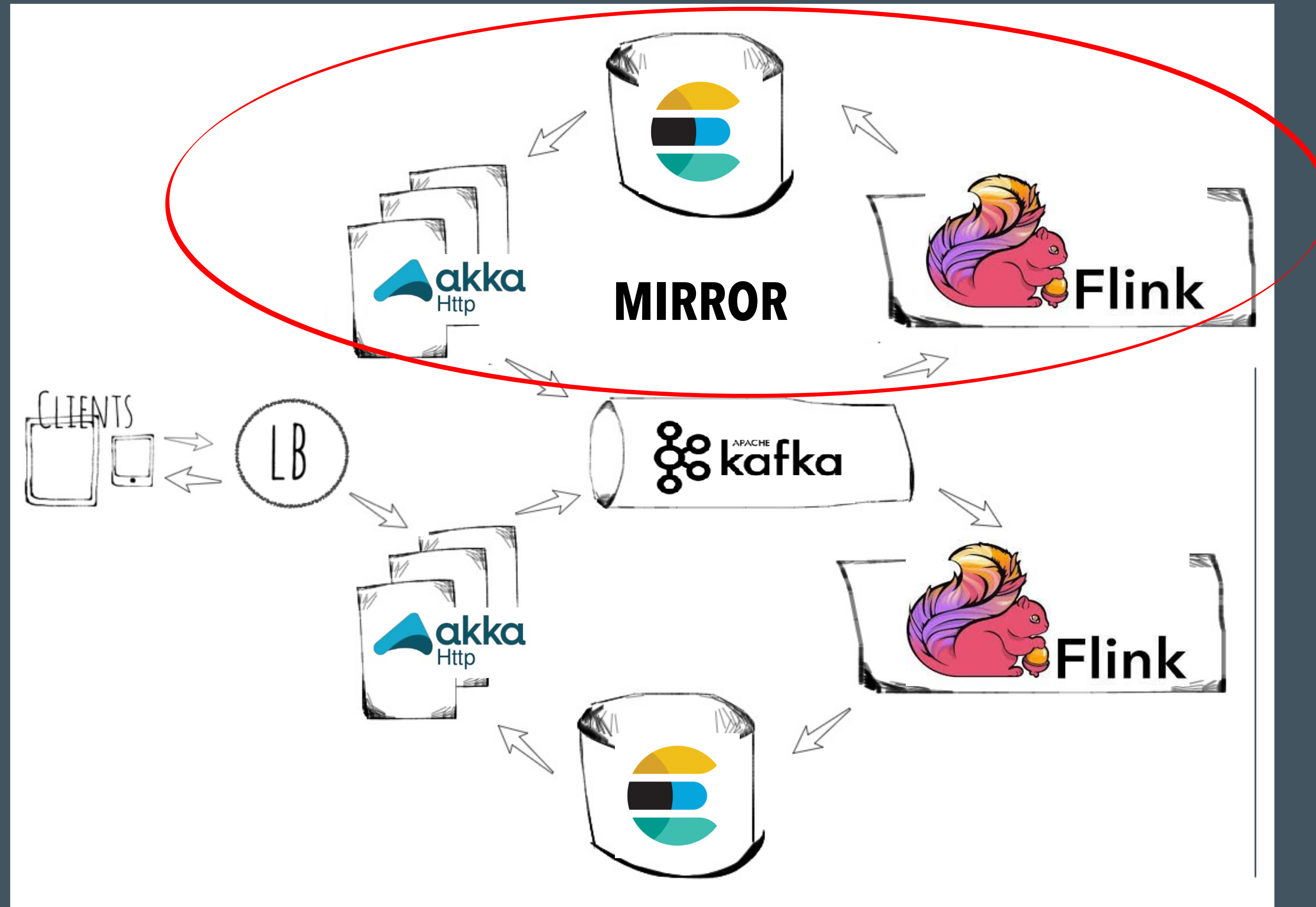
EVENT SOURCING IN PRACTICE



EVENT SOURCING IN PRACTICE



BLUE/GREEN APPROACH

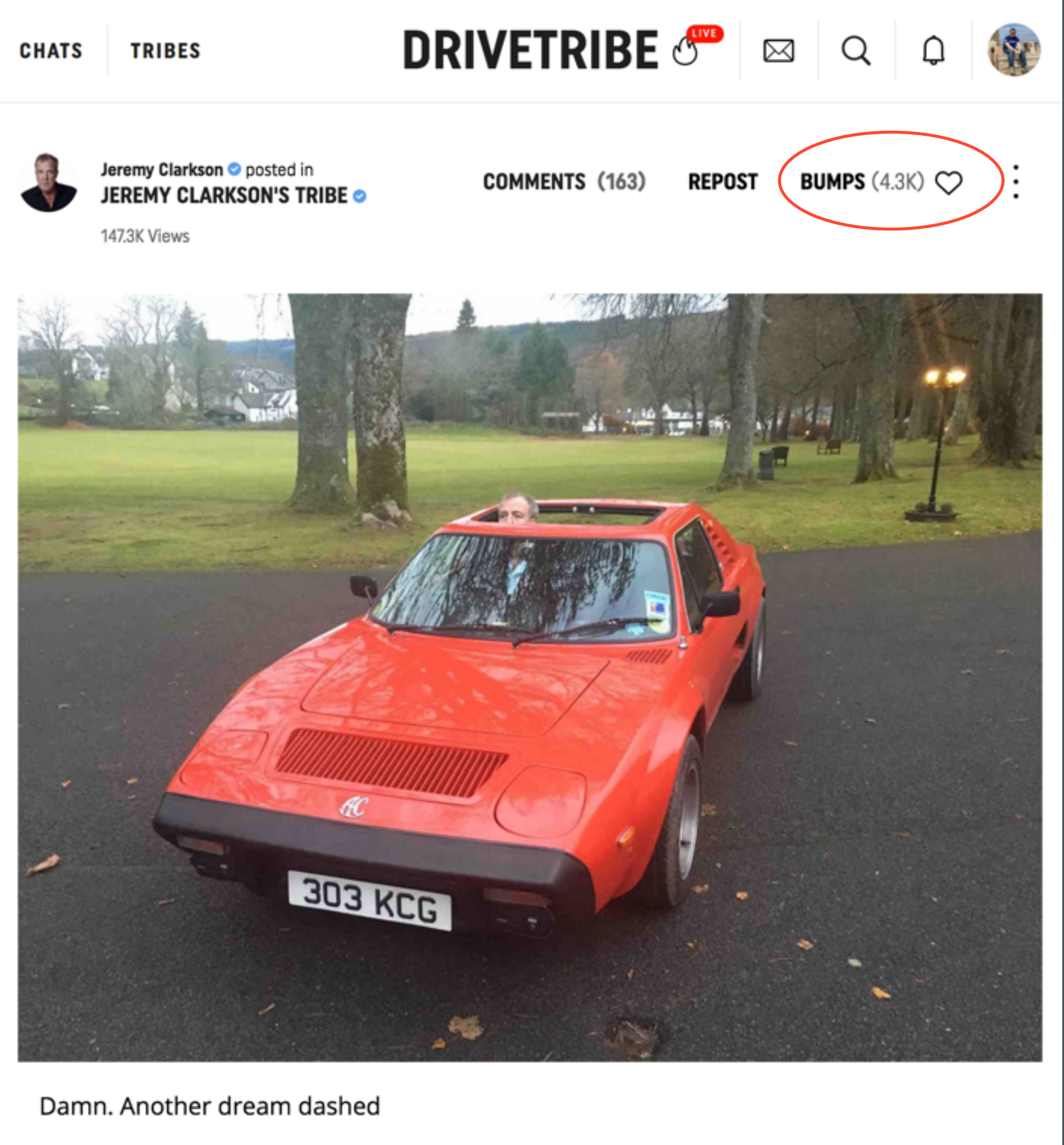


A REAL WORLD EXAMPLE

COUNTING BUMPS

- ▶ Thousands of people like the fact that Jeremy Clarkson is a really tall guy
- ▶ Users can “bump” a post if they like it

```
case class BumpEvent(  
  id: Id[Bump],  
  postId: Id[Post],  
  userId: Id[User],  
  bumpedOn: DateTime  
)
```



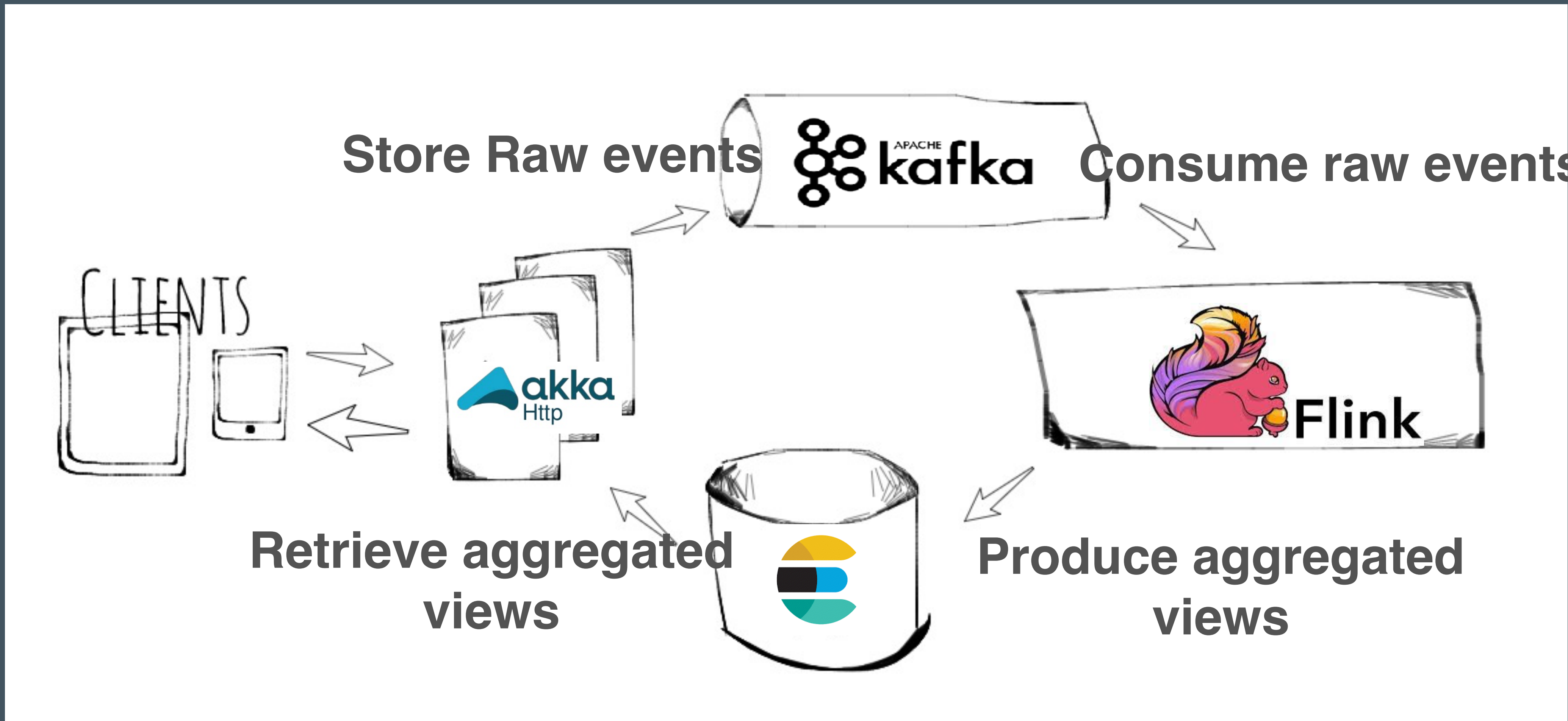
CHATS TRIBES DRIVETRIBE ^{LIVE} [mail] [search] [notifications] [profile]

Jeremy Clarkson ^{verified} posted in JEREMY CLARKSON'S TRIBE ^{verified}
147.3K Views

COMMENTS (163) REPOST **BUMPS (4.3K)** [heart] [more]

Damn. Another dream dashed

EVENT SOURCING IN PRACTICE



COUNTING BUMPS

```
case class State(id: Id[Post], bumpCounter: ???)

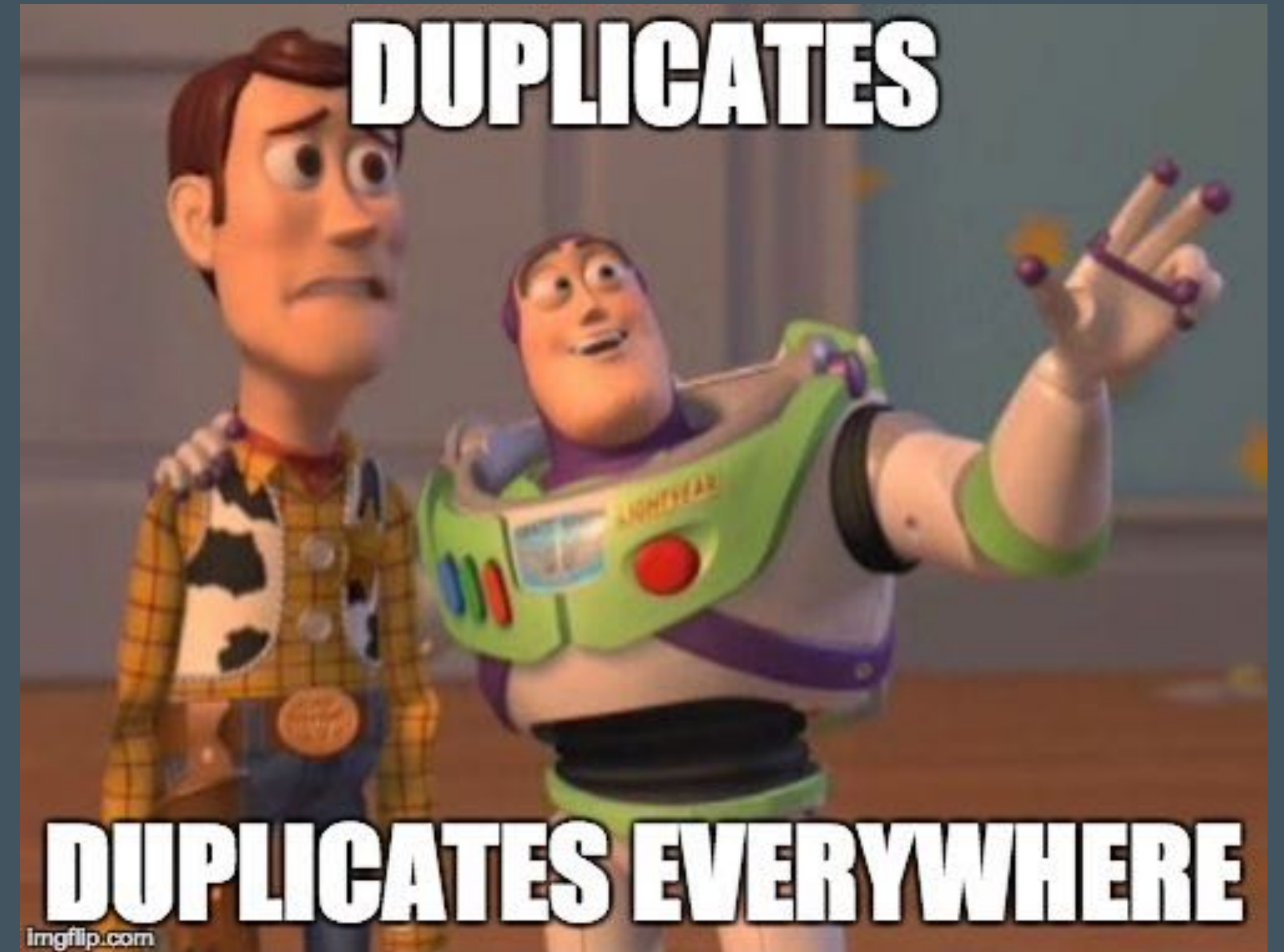
object PostStatsReducer {
  def apply(bumpEvents: DataStream[BumpEvent]): DataStream[State] = ???
}
```

COUNTING BUMPS - FIRST ATTEMPT

```
case class State(id: Id[Post], bumpCounter: Long)
object State {
  def apply (bumpEvent: BumpEvent): State =
    State(bumpEvent.postId, 1L)
}

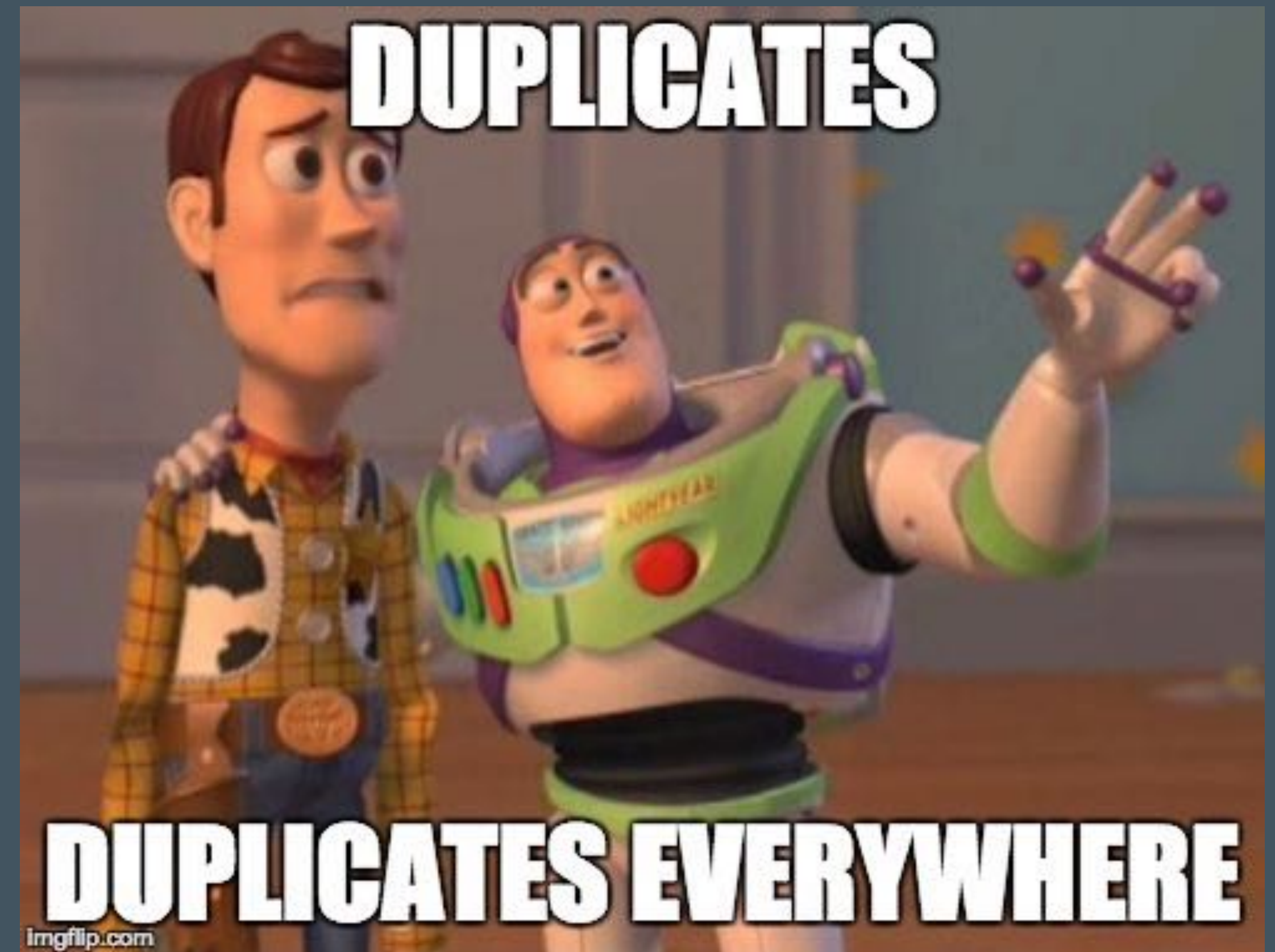
object PostStatsReducer {
  def apply(bumpEvents: DataStream[BumpEvent]): DataStream[State] =
    bumpEvents
      .map(State(_))
      .keyBy(_.id)
      ...
      State(state1.id, state1.bumpCounter + state2.bumpCounter)
      ...
      .combine
}
```

COUNTING BUMPS - FIRST ATTEMPT



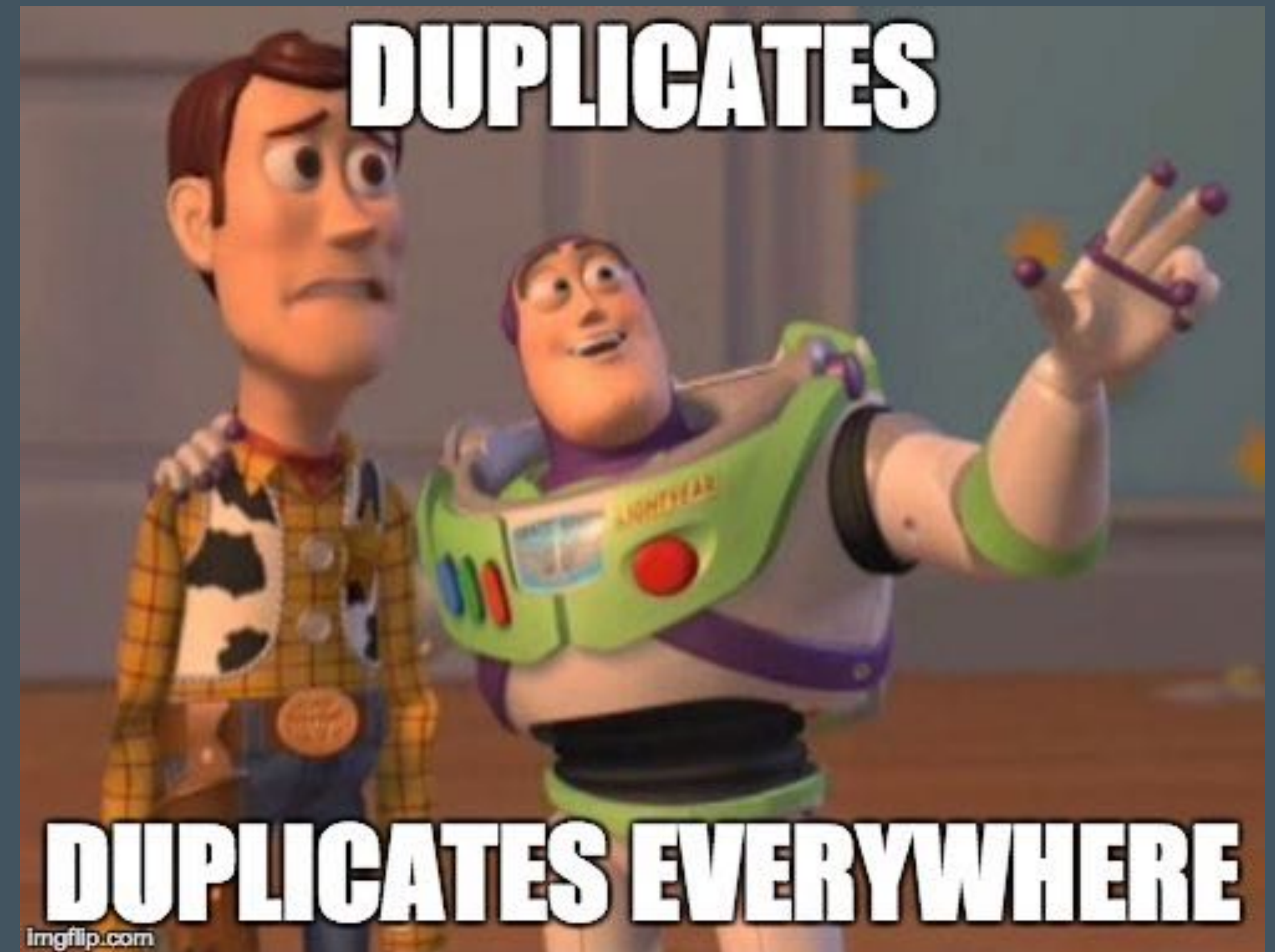
COUNTING BUMPS - FIRST ATTEMPT

- ▶ Use Flink with at least once semantics



COUNTING BUMPS - FIRST ATTEMPT

- ▶ Use Flink with at least once semantics
- ▶ Our system is eventually consistent



WHAT DO WE KNOW ABOUT OUT COUNTER?

- ▶ we know we're doing some kind of combine operation over States

WHAT DO WE KNOW ABOUT OUT COUNTER?

- ▶ we know we're doing some kind of combine operation over States
- ▶ we want our counter to be idempotent: $a | + | a === a$

WHAT DO WE KNOW ABOUT OUT COUNTER?

- ▶ we know we're doing some kind of combine operation over States
- ▶ we want our counter to be idempotent: $a \text{ |++} a === a$
- ▶ we also want our counter to be associative:
 $a \text{ |++} (b \text{ |++} c) === (a \text{ |++} b) \text{ |++} c$

BAND ALGEBRA

```
trait Band[T] {  
  def combine(t1: T, t2: T): T  
}
```

- ▶ Closed
- ▶ Idempotent
- ▶ Associative

COUNTING BUMPS - SECOND ATTEMPT

```
case class State[T: Band](id: Id[Post], bumpCounter: T)
object State {
  def apply (bumpEvent: BumpEvent): State = ???

  implicit val stateBand: Band[State] = ???
}

object PostStatsReducer {
  def apply(bumpEvents: DataStream[BumpEvent])(implicit band: Band[State]): DataStream[State] =
    bumpEvents
      .map(State(_))
      .keyBy(_.id)
      ...
      band.combine(state1, state2)
      ...
      .combine
}
```

COUNTING BUMPS - SECOND ATTEMPT

```
implicit def setBand[U] = new Band[Set[U]] {  
  def combine(s1: Set[U], s2: Set[U]): Set[U] =  
    s1 ++ s2  
}  
  
case class State(id: Id[Post], bumps: Set[Bump])  
object State {  
  ...  
  
  implicit val stateBand = new Band[State] {  
    def combine(s1: State, s2: State): State =  
      State(s1.id, s1.bumps |+| s2.bumps)  
  }  
}
```

- ▶ if all components of a case class have a band then so does the case class
- ▶ would normally bring in Set implementation from a library
- ▶ normally that library would have a law testing module

COUNTING BUMPS - PUTTING IT TOGETHER

```
case class State(id: Id[Post], bumps: Set[BumpEvent])
object State {
  def apply (bumpEvent: BumpEvent): State =
    State(bumpEvent.postId, Set(bumpEvent))

  implicit val stateBand = new Band[State] {
    def combine(s1: State, s2: State): State =
      State(s1.id, s1.bumps |+| s2.bumps)
  }
}

object PostStatsReducer {
  def apply(bumpEvents: DataStream[BumpEvent])(implicit band: Band[State]): DataStream[State] =
    bumpEvents
      .map(State(_))
      .keyBy(_.id)
      ...
      band.combine(state1, state2)
      ...
      .combine
}
```

OTHER ALGEBRAS WE USE

- ▶ Adding events: Semigroup/Monoid
- ▶ Duplicate events: Band
- ▶ Out of order and duplicate events: Semilattice

FIN