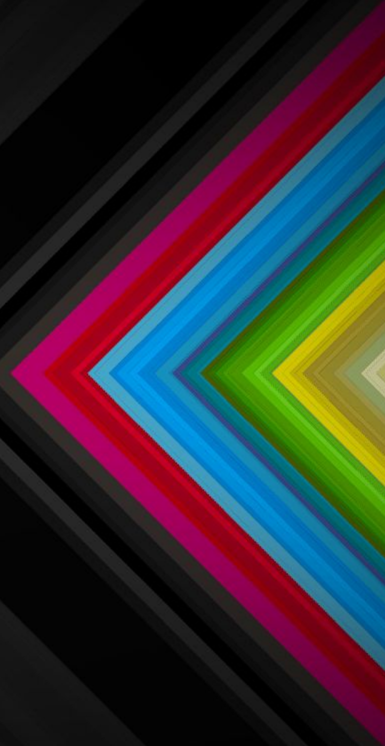
AKKA

Mastering event sourcing

A game changing design pattern for distributed systems



Overview

AKKA

01 **Distributed systems problems**
Difficulties building them today

02 **Distributed systems solutions**
Tackling distributed systems complexity

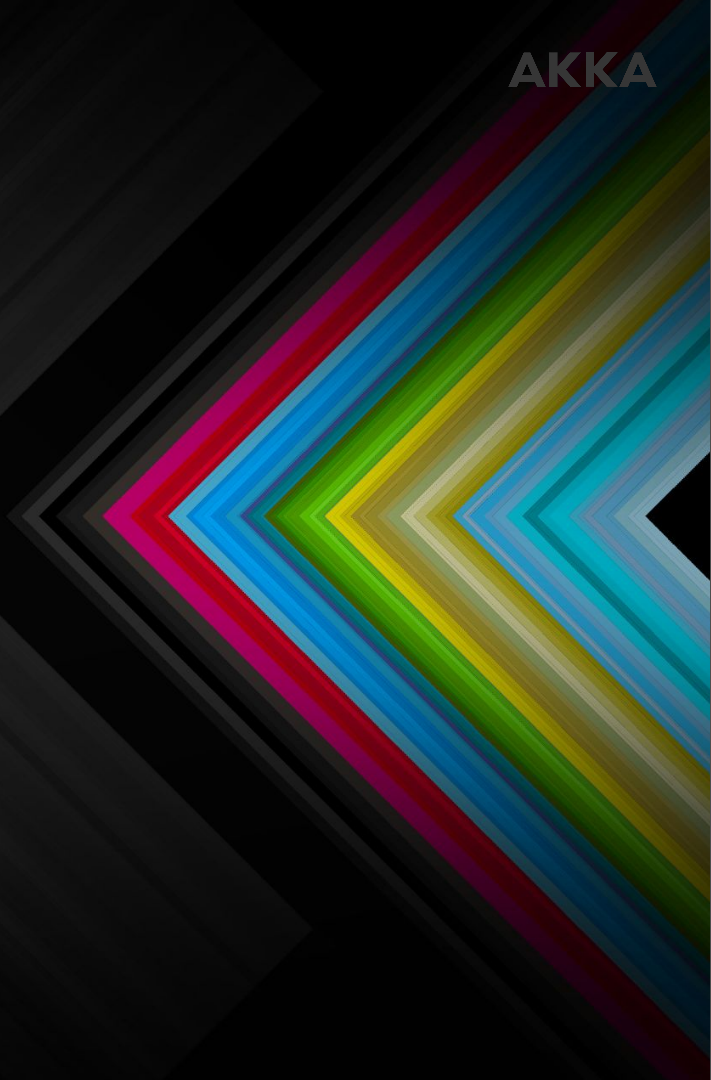
03 **Event sourcing**
Formalizing solution patterns into rules

04 **Demo**
A Real Event-Sourced Application

05 **Q&A**

Today's problems

Building distributed apps is hard



AKKA

Multiple sources of truth

AKKA

- Which truth is the real truth?
- Data from the edge
- Data from mobile
- Data from the cloud
- Conflicting data from multiple sources
- Derived, n-level data

Lack of **audit trail**

AKKA

- *How* did we get here?
- *Why* is the state the way it is?
- *When* did it become that way?
- Can we regenerate state based on new logic?
- Run what-if scenarios?

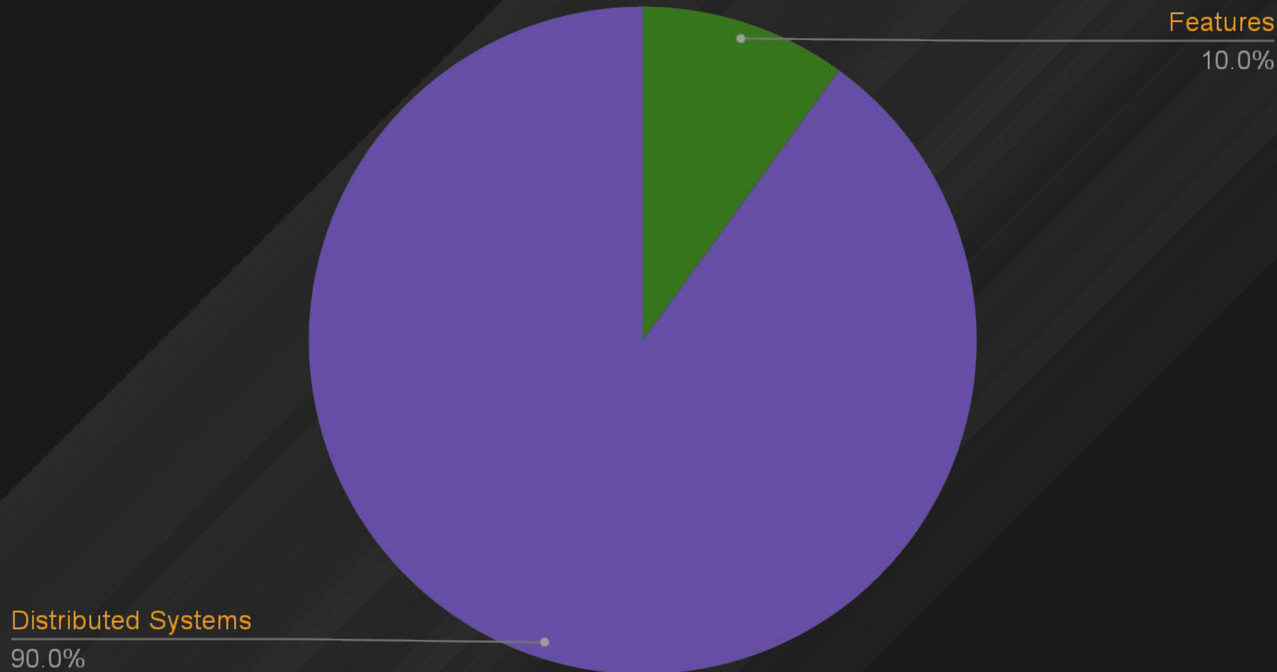
Distributed, concurrent writes

AKKA

- Last write wins
  - Writes based on outdated information
  - Often *not* the right conflict resolver
- Etags / optimistic concurrency
- Eventually we take on conflict complexity
  - Use CRDTs
  - Roll our own RAFT 😱
  - "Retry until it works" / "Hope-based consistency"
- Distributed transactions

Spending on the wrong things

AKKA



Normalized, relational data

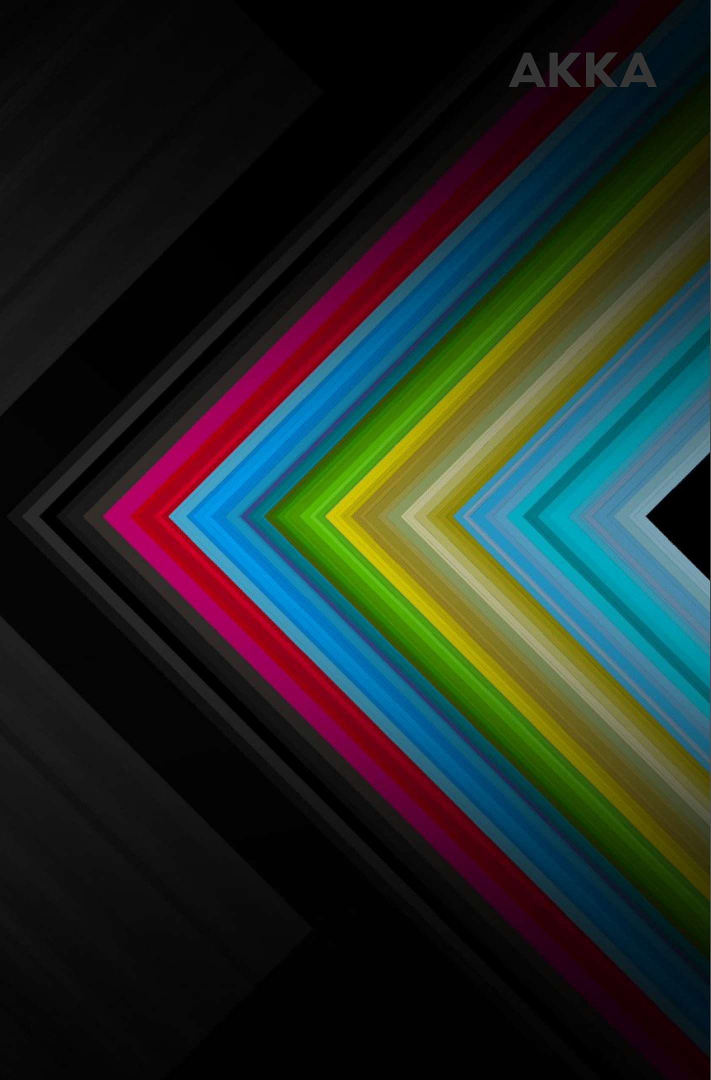
AKKA

On-demand queries often too slow and don't scale

- Maybe...move data closer to edge to make query run faster
- But...now we have more consistency problems
- Maybe...add a cache layer to improve queries
- But...now we have more complexity

Today’s solutions

Patterns for building distributed apps



AKKA

One source of truth

AKKA

- Everything is an **immutable event**
- Record of what *did* happen, not what *failed to happen*
- Eventually consistent
- Safely distributed
- Developers should not code their own conflict resolvers

Embrace **eventual consistency**

- Identify activities that need strict consistency

    → **More often than not, eventual consistency is enough**

- What data can be stale, and how stale?
- When do you need to read your own writes?
- Be explicit about consistency ←→ perf/complexity tradeoffs

There is no such thing as a **transaction**

AKKA

- Distributed transactions provide false sense of security
- What do you do when rollback/compensation fails?
- Even distributed transactions require conflict resolution

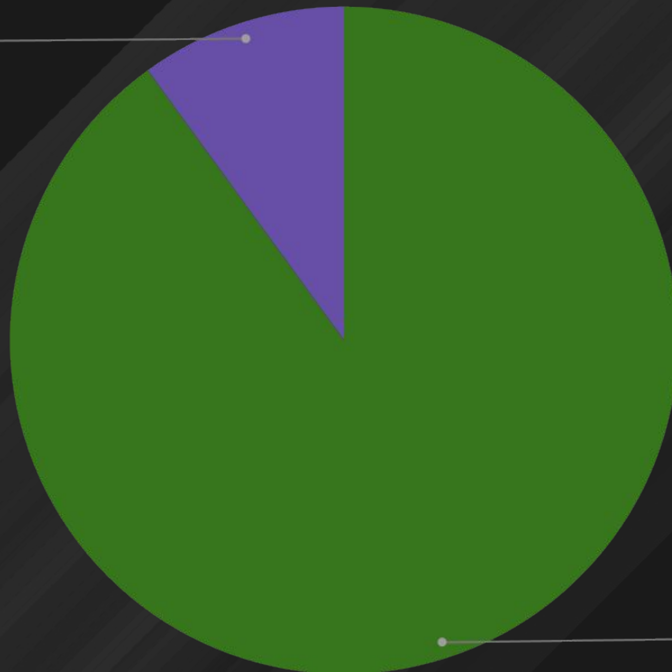
Generate query results **before** users need them

AKKA

- Materialized views
  - ***De***normalized data
  - Shaped for consumer needs, not database needs
- **O(1)** query cost whenever possible
- Makes it easy to replicate views
- Views can be used as explicit consistency boundaries

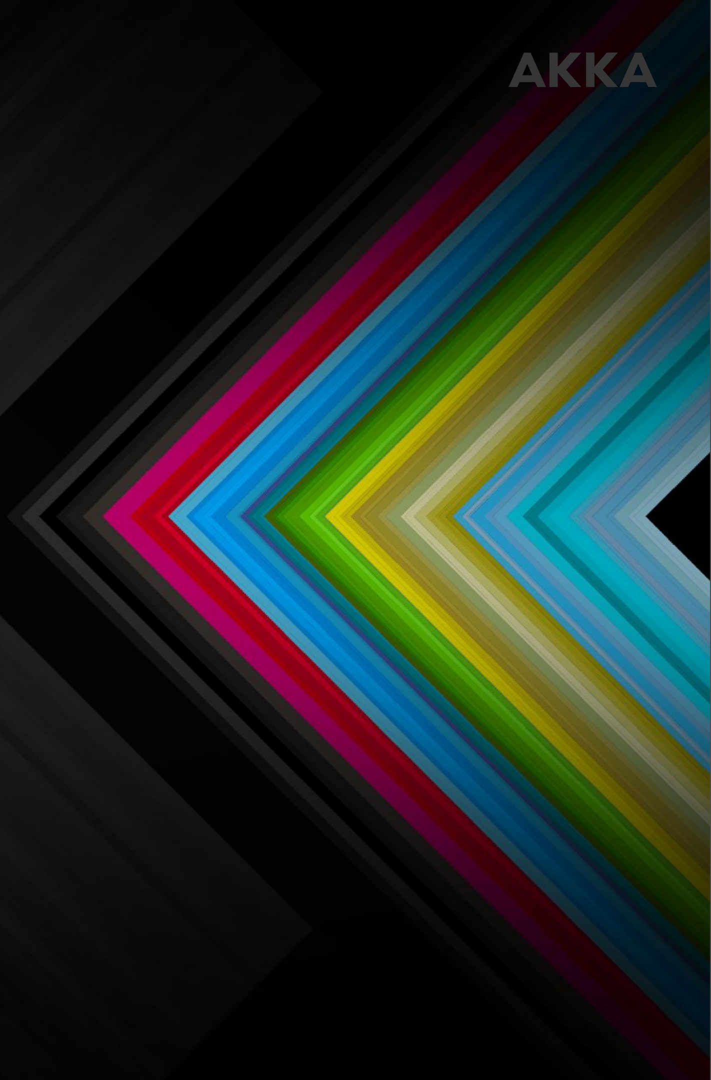
Spending on the right things

AKKA



Event Sourcing

Formalizing rules and patterns



Everything you need to know

$$f(state, event) = state'$$

$$f(state, command) = \{ event_1, event_2, \dots \}$$

Write less code, get distributed system as a bonus

The building blocks

- Events / Event log
- Command
- Entity (aggregate)
- View
- Workflow (process manager)
- Producers and Consumers (gateway)

AKKA

Commands

AKKA

- A request to produce an effect
  - Persist an event
  - Philosophy debate: should commands be used to query data?
- Ephemeral
  - *Commands do not exist*
  - Never included in replay

AKKA

Events

- Represent something that occurred in the past
- Immutable
- “Reality” is event sourced
  - Input gathered from many senses, reality (a.k.a. “a view”) produced in near-real time

Entities

$f(state, event) = state'$

$f(state, command) = \{ event_1, event_2, \ldots \}$

- Handle commands
  - Validate command against current state
  - Generate effects in response, or reject command
- Produce events
- Apply events to state

AKKA

Views

- For FP fans: *left fold over an event stream*
- Apply events to (often denormalized) data
- Consumer-friendly data, optimized for O(1) query
- Views are easily evolved:
  - Change logic in code
  - Replay event stream, regenerate view
- Different scale, resilience, replication needs than entity state
  - Views should ***not*** be used to make entity decisions

Workflows

- Manage “long-running” processes
- Define steps as code
- Examples…
  - Shopping carts
  - Ticket holding (movies, concert, airline)
  - Fulfillment
  - … many more

Mutable state vs events

AKKA

Bank Account 867001	Bank Account 867001
Balance $4200.12	**Deposit** $200.12
	Withdrawal $765.21
	Transfer from 9021123 $1321.41

AKKA

The event sourcing RULES

- Never modify an event
- Never read the “wall clock” for state
- Never* use random numbers to produce state
- Do not model “failed to create” as events
- Never* produce side effects when processing events

AKKA

How to build event sourced apps with Akka

Simple developer experience

- Model event sourced domain
  - Less code
  - Easier to maintain
  - Smaller cognitive overhead
- Let experts deal with deploy, distribute, etc
  - Trust, but verify

AKKA

Demo

Thank you

Get Started
for Free

https://akka.io
https://docs.akka.io
https://github.com/akka-samples/akka-chess