Building microservices with Scala, functional domain models and Spring Boot

Chris Richardson

Author of POJOs in Action

Founder of the original CloudFoundry.com

@crichardson

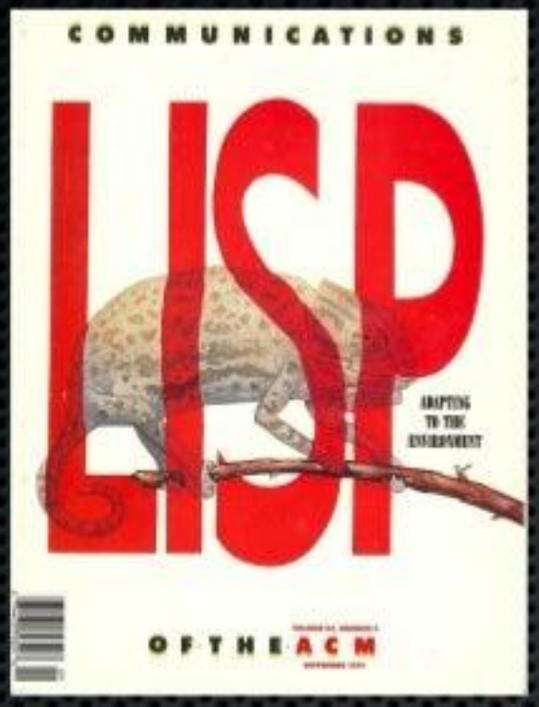
chris@chrisrichardson.net

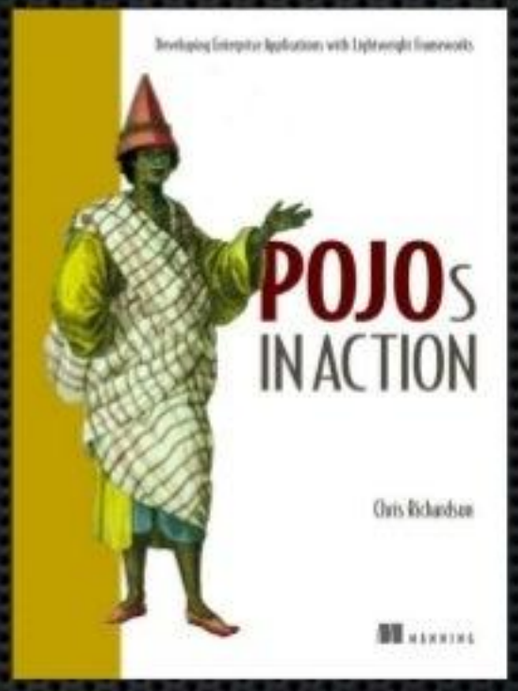
http://plainoldobjects.com

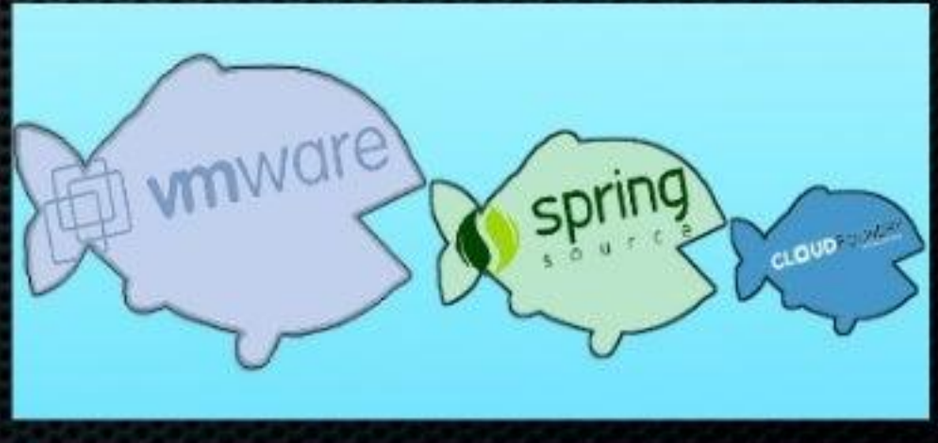
Presentation goal

Share my experiences with building an application using Scala, functional domain models, microservices, event sourcing, CQRS, and Spring Boot

About Chris







@crichardson

About Chris

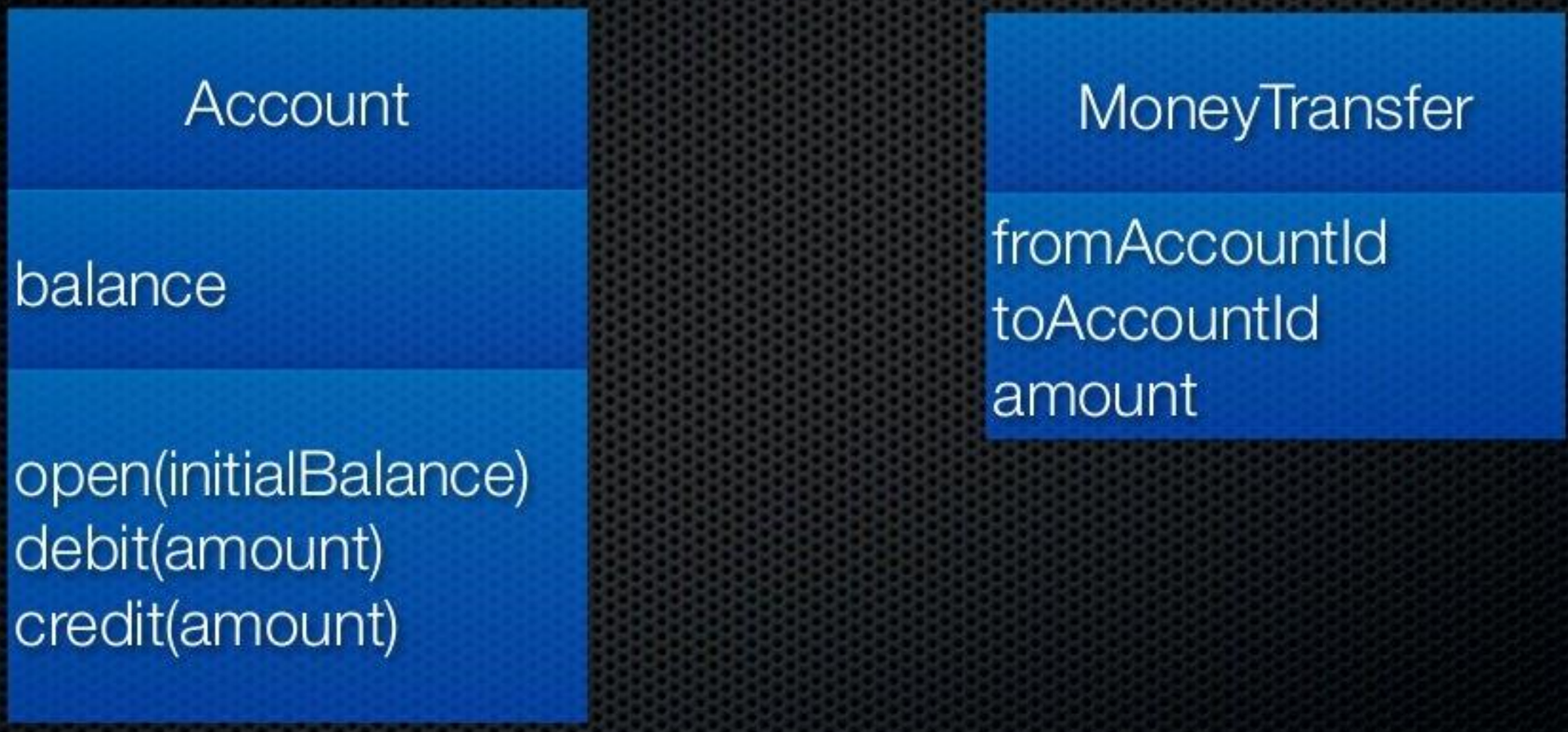
- Founder of a buzzword compliant (stealthy, social, mobile, big data, machine learning, ...) startup
- Consultant helping organizations improve how they architect and deploy applications using cloud, micro services, polyglot applications, NoSQL, ...

Agenda

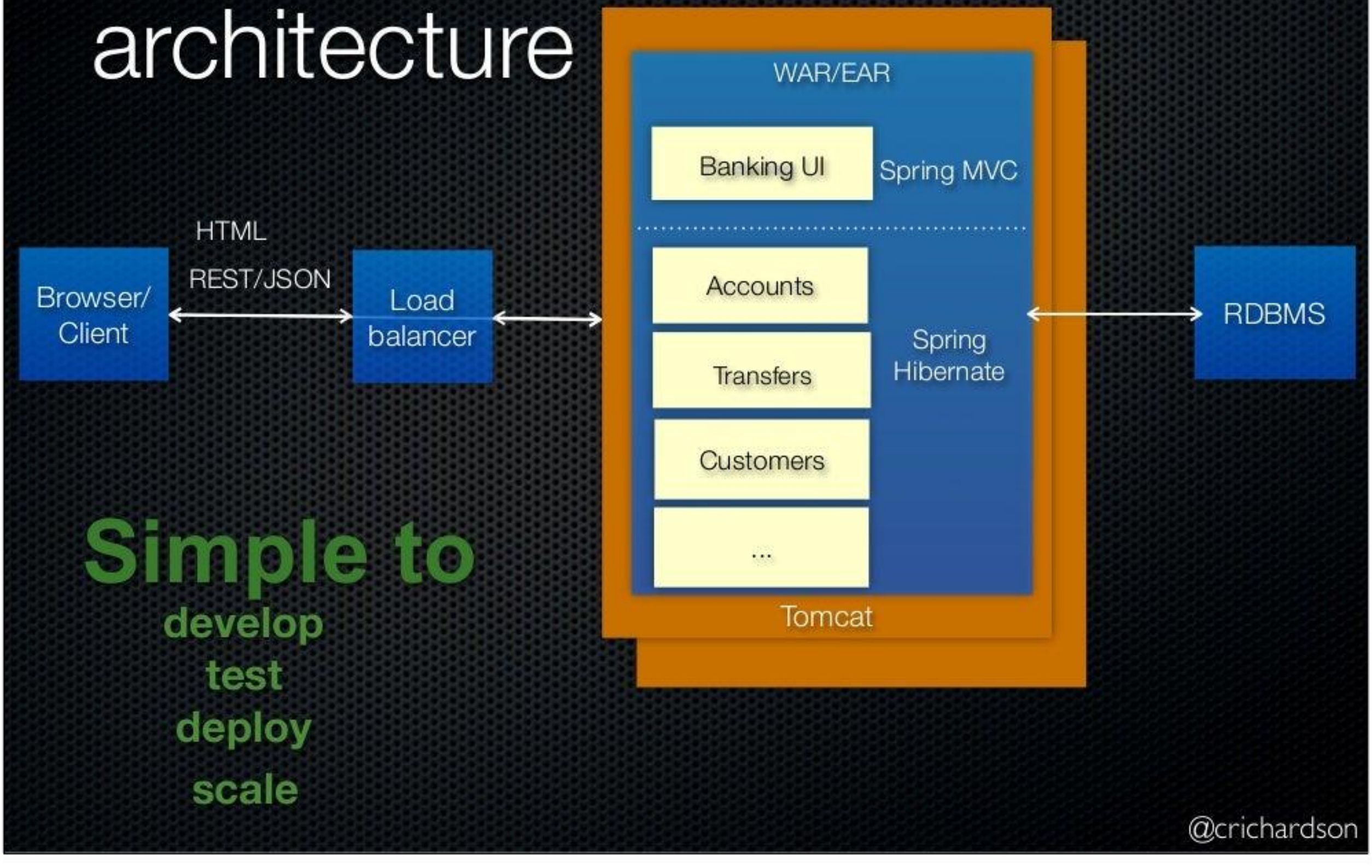
- Why build event-driven microservices?
- Overview of event sourcing
- Designing microservices with event sourcing
- Implementing queries in an event sourced application
- Building and deploying microservices

Let’s imagine that you are building a banking app...

Domain model



Traditional application architecture



Simple to

develop

test

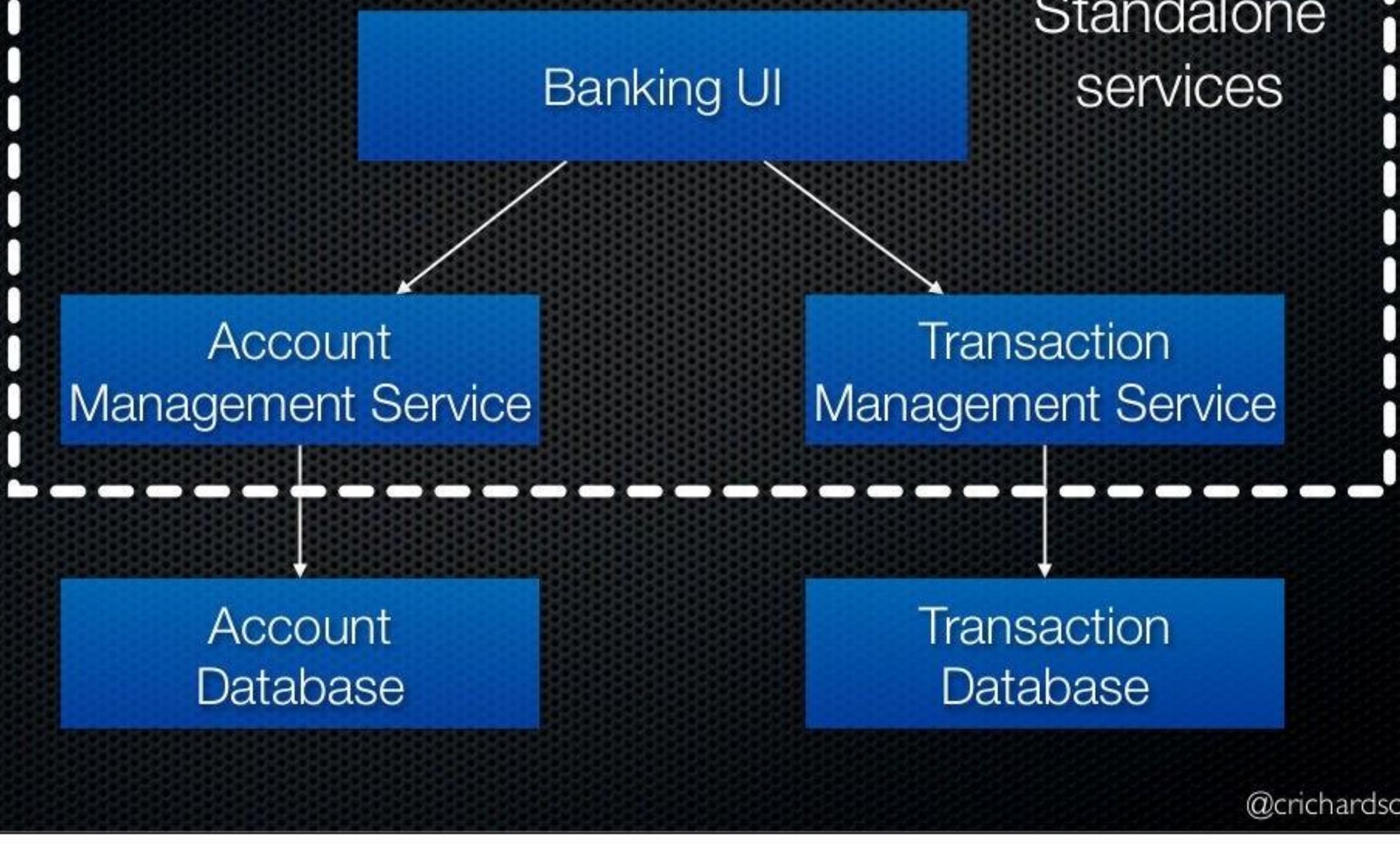
deploy

scale

Problem #1: monolithic architecture

- Intimidates developers
- Obstacle to frequent deployments
- Overloads your IDE and container
- Obstacle to scaling development
- Requires long-term commitment to a technology stack

Solution #1: use a microservice architecture



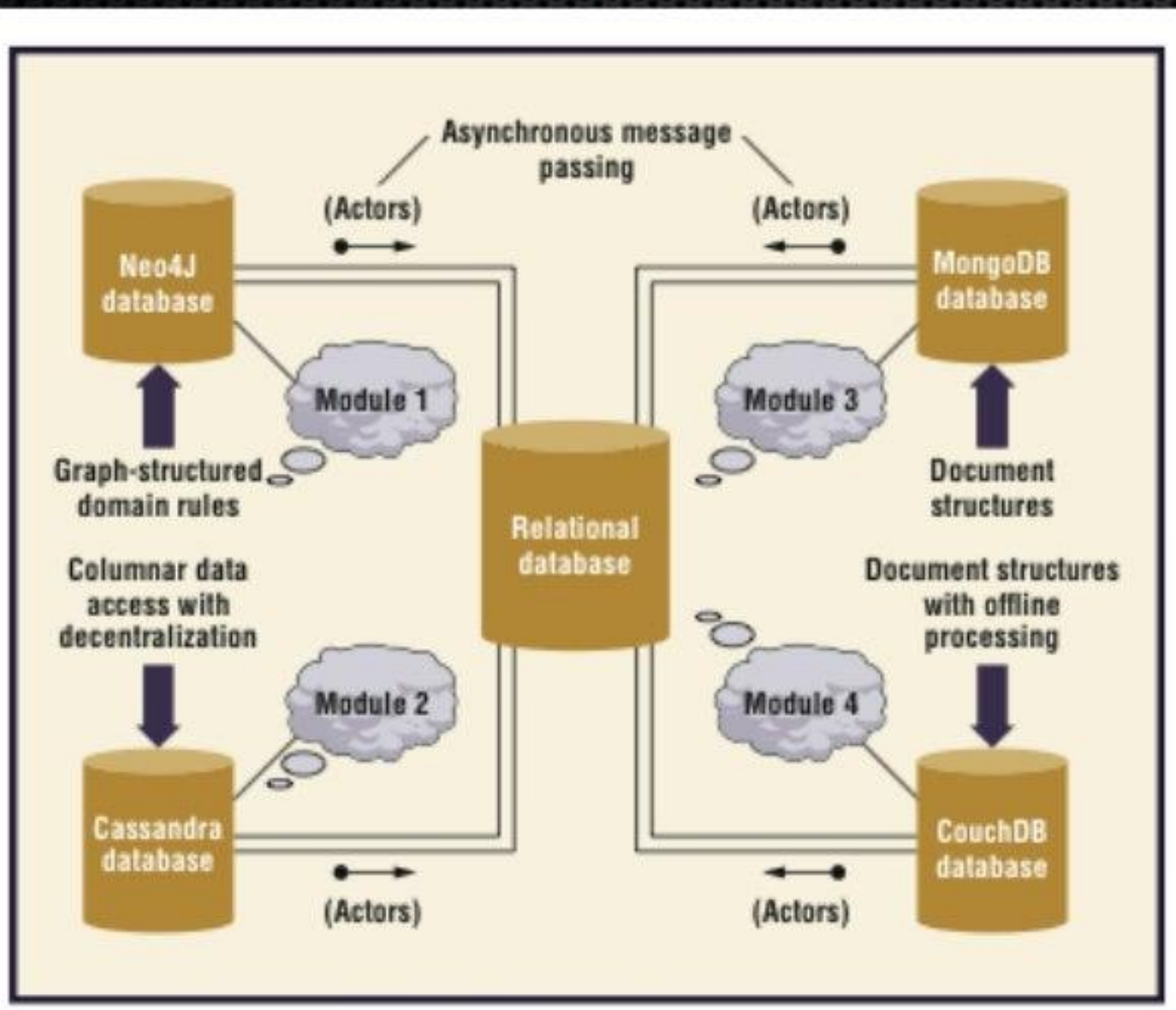
Problem #2: relational databases

- Scalability
- Distribution
- Schema updates
- O/R impedance mismatch
- Handling semi-structured data

Solution #2: use NoSQL databases

- Avoids the limitations of RDBMS
- For example,
  - text search ⇒ Solr/Cloud Search
  - social (graph) data ⇒ Neo4J
  - highly distributed/available database ⇒ Cassandra
  - ...

Different modules use different databases



IEEE Software Sept/October 2010 - Debasish Ghosh / Twitter @debasishg

@crichardson

But now we have problems with data consistency!

Problem #3: Microservices = distributed data management

- Each microservice has it's own database
  - Some data is replicated and must be kept in sync
  - Business transactions must update data owned by multiple services
- Tricky to implement reliably without 2PC

Problem #4: NoSQL = ACID-free, denormalized databases

- Limited transactions, i.e. no ACID transactions
  - Tricky to implement business transactions that update multiple rows, e.g. http://bit.ly/mongo2pc
- Limited querying capabilities
  - Requires denormalized/materialized views that must be synchronized
  - Multiple datastores (e.g. DynamoDB + Cloud Search) that need to be kept in sync

Solution to #3/#4: Event-based architecture to the rescue

- Microservices publish events when state changes
- Microservices subscribe to events
  - Synchronize replicated data
  - Maintains eventual consistency across multiple aggregates (in multiple datastores)

Eventually consistent money transfer

transferMoney()

MoneyTransferService

AccountService

Subscribes to:

AccountDebitedEvent

AccountCreditedEvent

publishes:

MoneyTransferCreatedEvent

DebitRecordedEvent

Subscribes to:

MoneyTransferCreatedEvent

DebitRecordedEvent

Publishes:

AccountDebitedEvent

AccountCreditedEvent

Message Bus

@crichardson

To maintain consistency a service must

atomically publish an event **whenever** a domain object changes

@crichardson

How to generate events reliably?

- Database triggers, Hibernate event listener, ...
  - Reliable BUT
  - Not with NoSQL
    - Disconnected from the business level event
    - Limited applicability
- Ad hoc event publishing code mixed into business logic
  - Messy code, poor separation of concerns
  - Unreliable, e.g. too easy to forget to publish an event