Implementacja agregatów (DDD) w Akka

Krzysztof Muchewicz

Solutions Architect, Grupa Allegro

Anemic domain model

„In an anemic domain design, business logic is typically implemented in separate classes which transform the state of the domain objects.”

Anemic Domain Model

```
class User {
    String name;
    String address;
}

class UserController {
    void validate(User user) { … }
}

class UserValidator { … }

class UserRepository {
    void save(User user) { … }
}
```

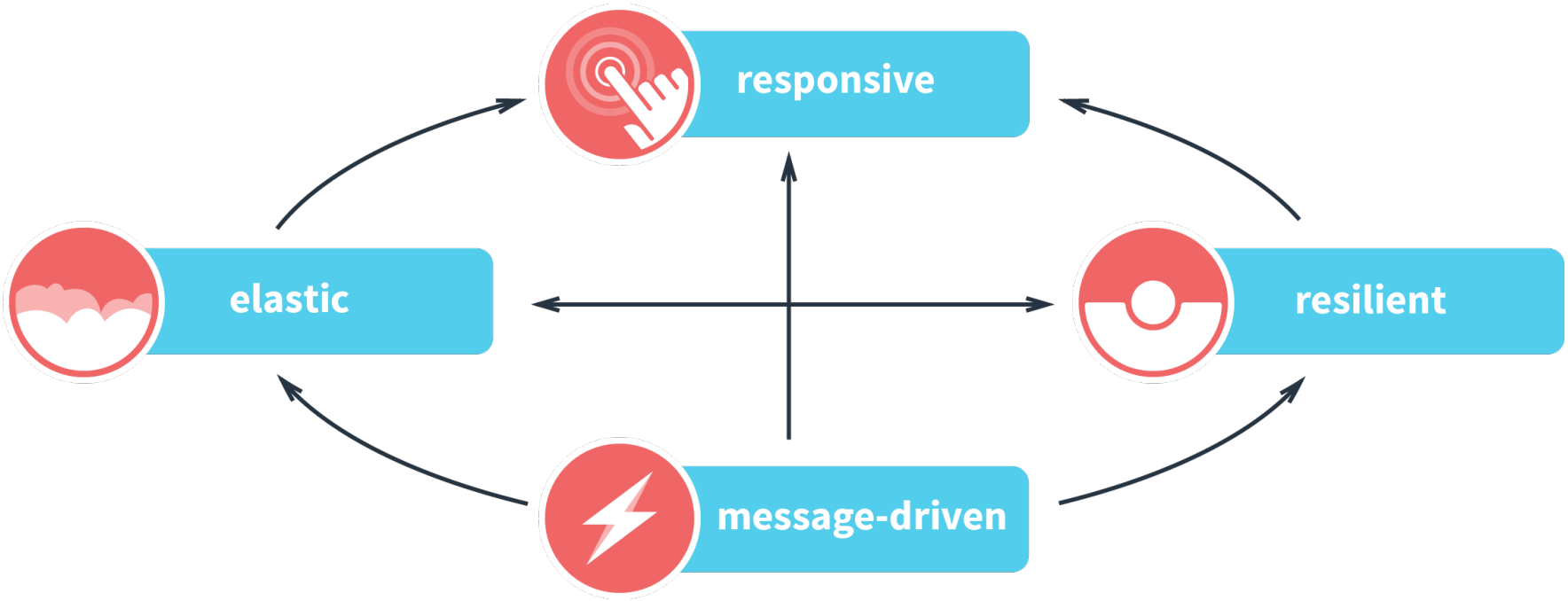
Czasem anemiczny nie daje rady

Agregaty i DDD

„Each aggregate has to be consistent across whole system.”


akka
Build powerful concurrent & distributed applications more easily.
Akka is a toolkit and runtime for building highly concurrent, distributed, and fault tolerant event-driven applications on the JVM.
Simple Concurrency & Distribution
Asynchronous and Distributed by design. High-level abstractions like Actors, Futures and STM.
Resilient by Design
Write systems that self-heal. Remote and/or local supervisor hierarchies.
High Performance
50 million msg/sec on a single machine. Small memory footprint; ~2.5 million actors per GB of heap.
Elastic & Decentralized
Adaptive load balancing, routing, partitioning and configuration-driven remoting.
Extensible
Use Akka Extensions to adapt Akka to fit your needs.

http://www.reactivemanifesto.org/



Actor

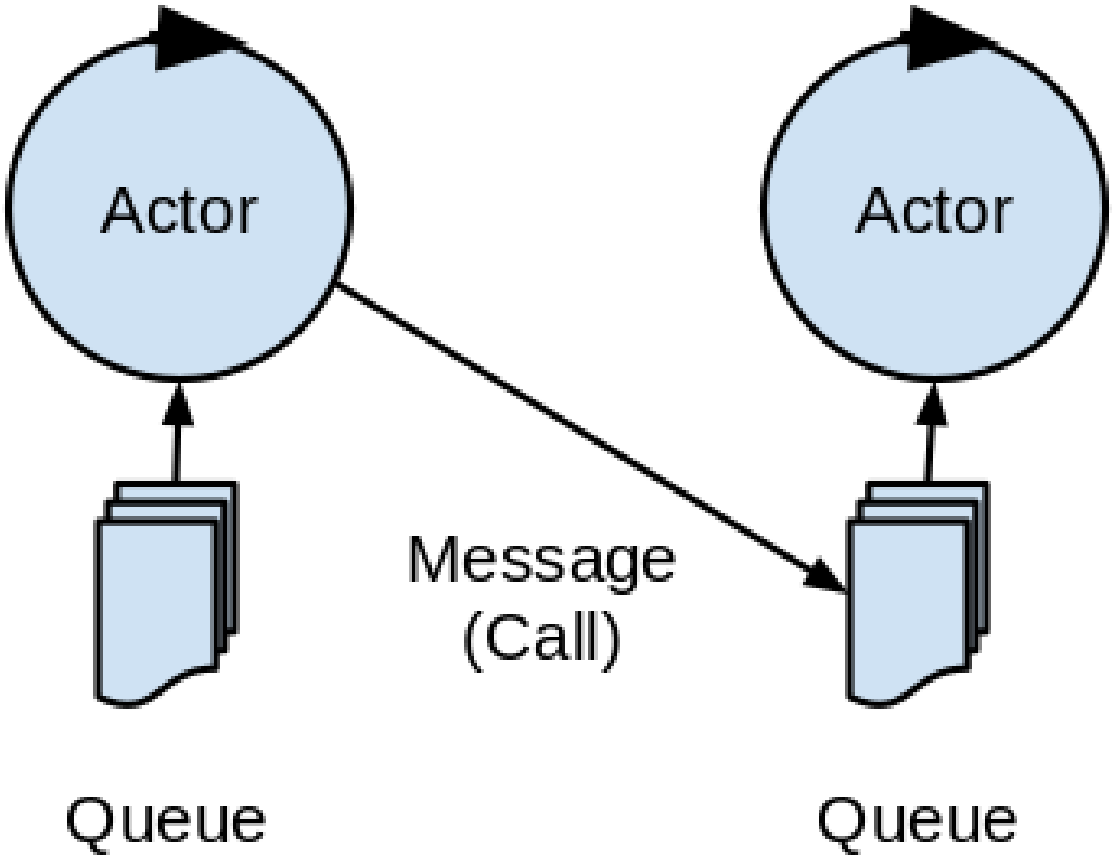
Fundamental unit of computation that embodies:

- Processing
- Storage
- Communication

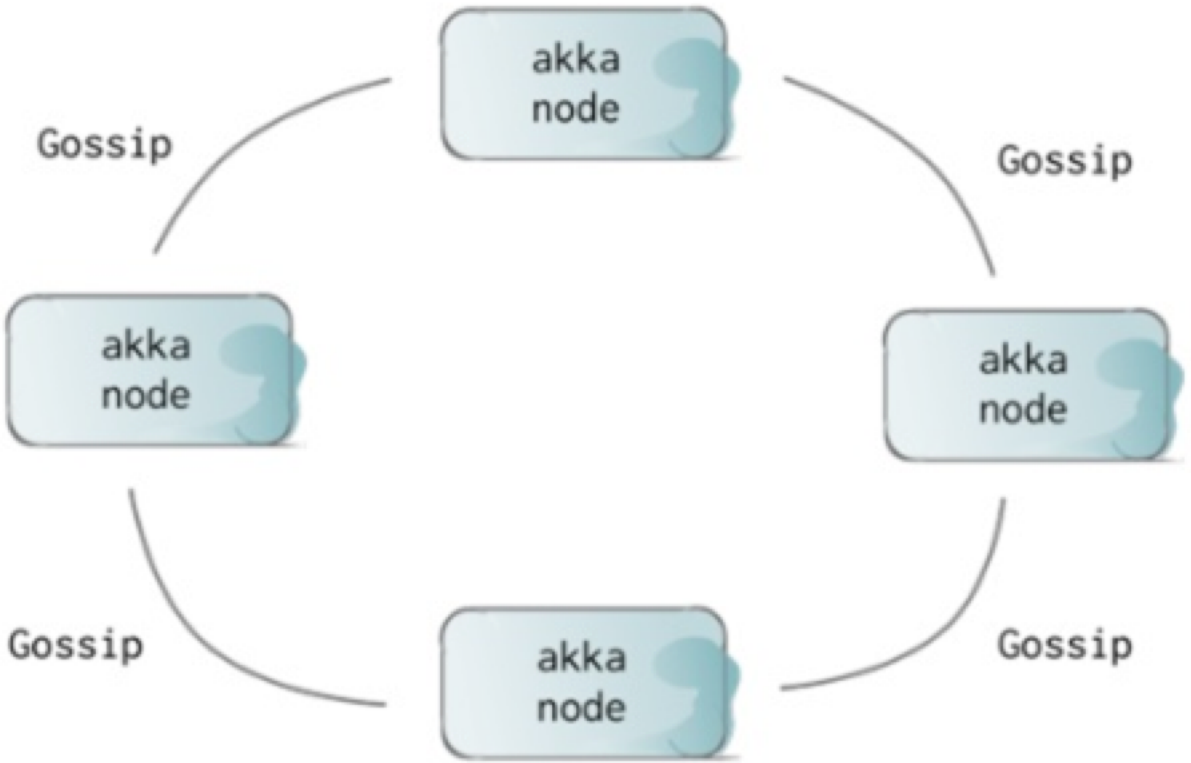
3 axioms – When actor receives message it can:

- Create new Actor
- Send message to Actor it knows
- Designate how it should handle the next message it receives

```
class MyActor extends Actor {
    override def receive: Actor.Receive = {
        case event => "do the job”
        case otherEvent => ”other job”
    }
}
```


Actor
Actor
Message
(Call)
Queue
Queue

Akka clustering



Actor to doskonała abstrakcja dla Agregatu

Pytania