Как мы строили высокопроизводительную систему на Akka с нуля: джентльменский набор и грабли

Кирилл Данилов
АО «НСПК»

Докладчик

Данилов Кирилл

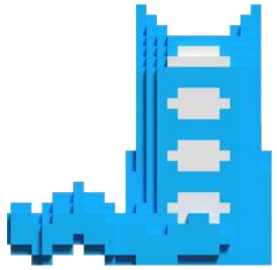
Руководитель направления

danilovkv@nspk.ru

Что такое СБП?

СБП – Система Быстрых Платежей

Межбанковский перевод



Card2Card

Перевод по номеру телефона

C2C

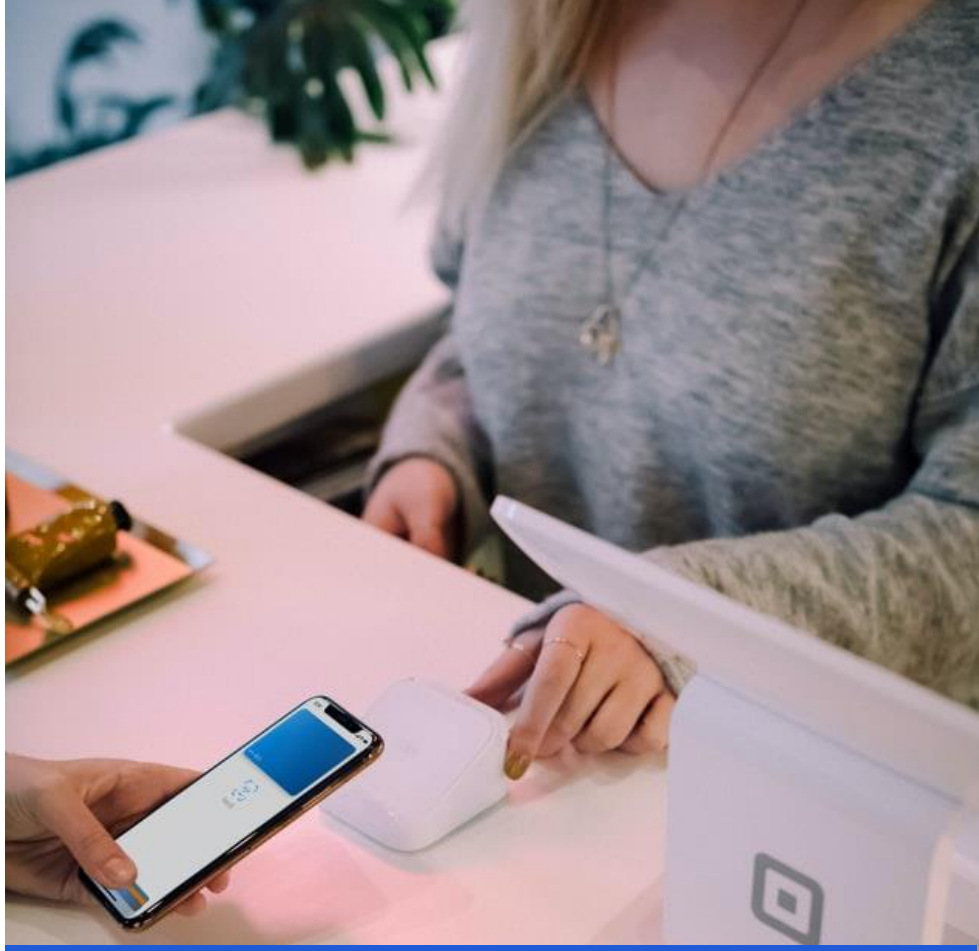
Физик физику: обычные переводы друг другу

28.01.2019 C2C Friends&Family

28.02.2019 C2C танцуют все!

Физик юрику: переводы от физлиц юрлицам, или, обычным языком, оплата в магазинах

Сентябрь 2019 – запуск C2B



C2B

B2C

Юрик физику: выплата зарплаты,
выплата за возврат товара,
страховые выплаты

Ноябрь 2020 – запуск B2C

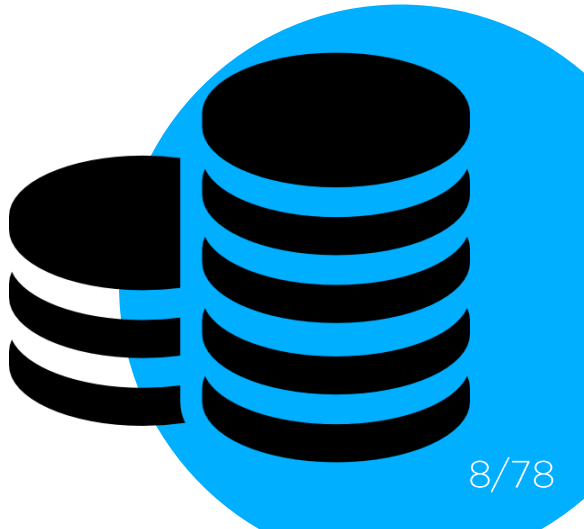
Подключение всех системообразующих банков

Май 2020
Подключение Сбера

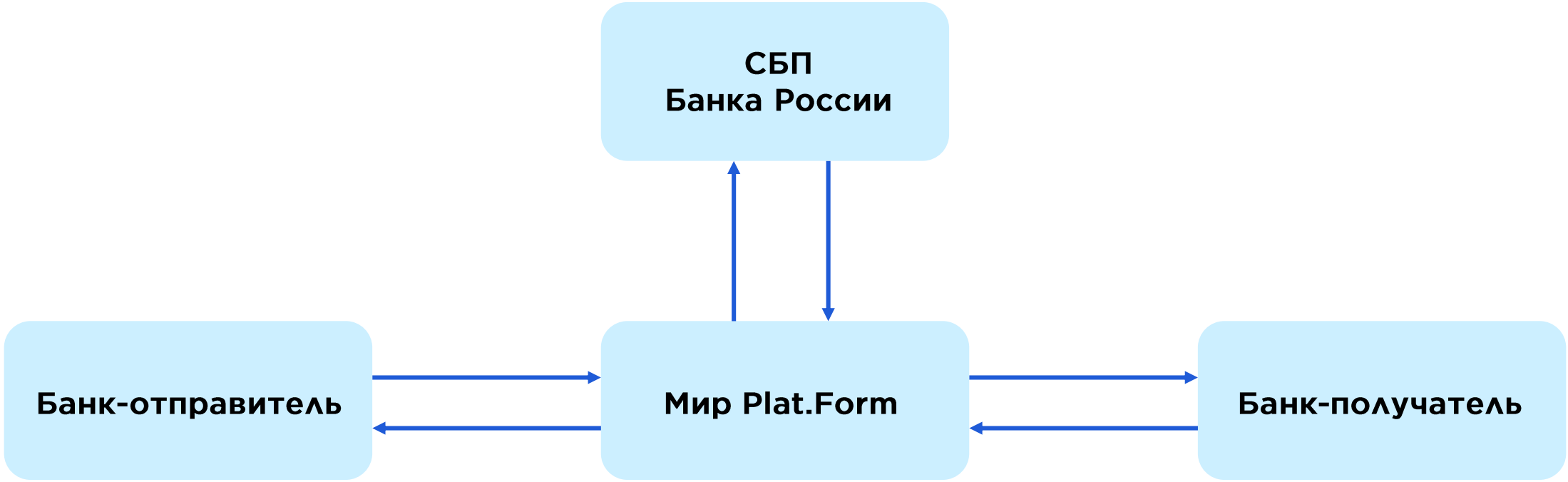
1 000 000 000 000

28.01.2021

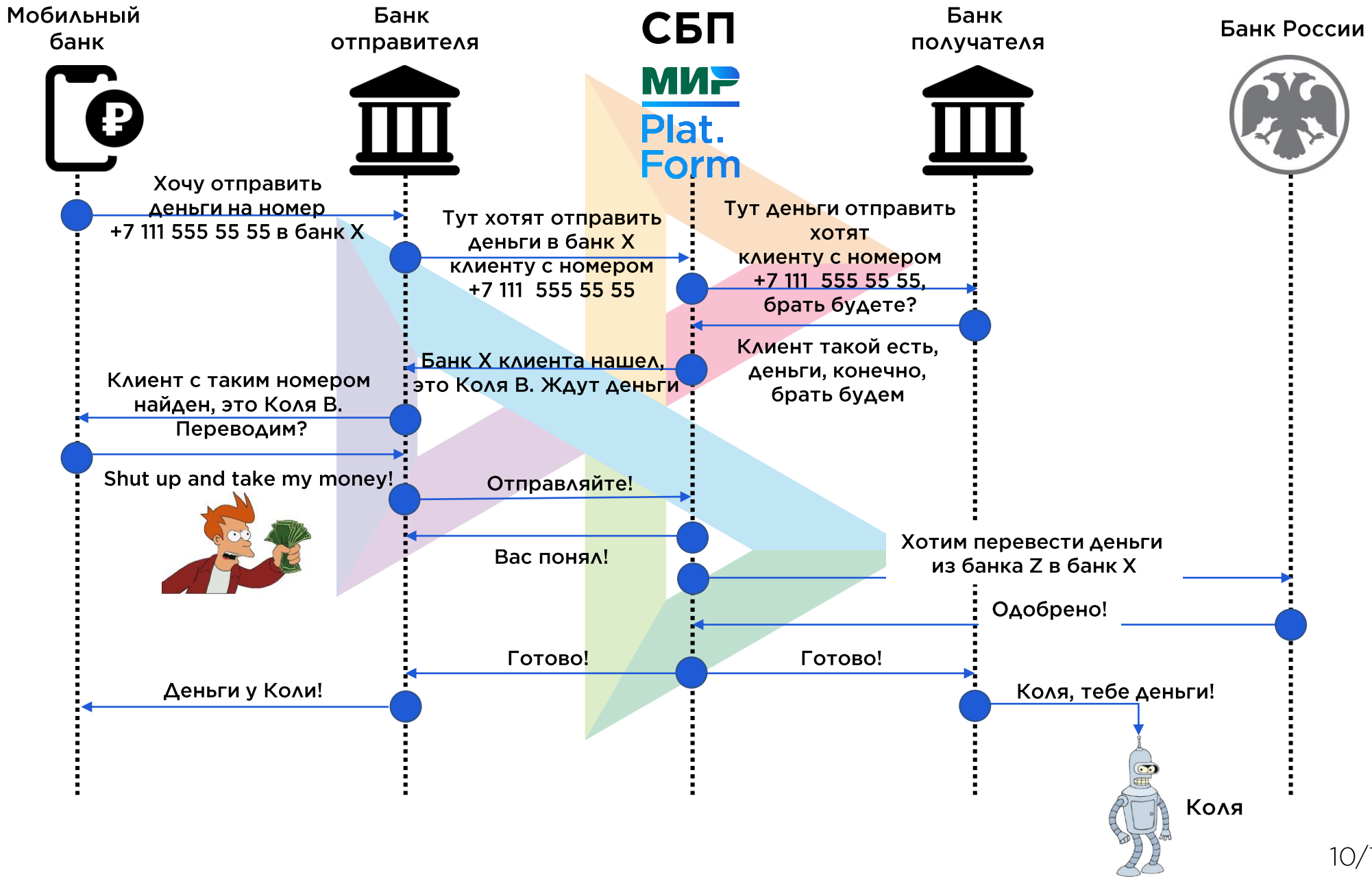
1 триллион рублей



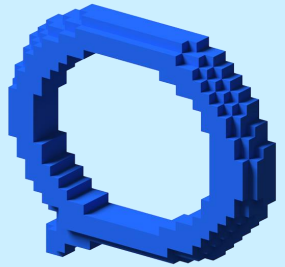
Высокоуровневая архитектура



Взаимодействие при переводе



Немного чисел про нагрузку



- На старте были требования выдерживать 1500 tps
  Сейчас в пике до 50 tps

- Одна успешная транзакция – 14-40 сообщений

Выбор платформы

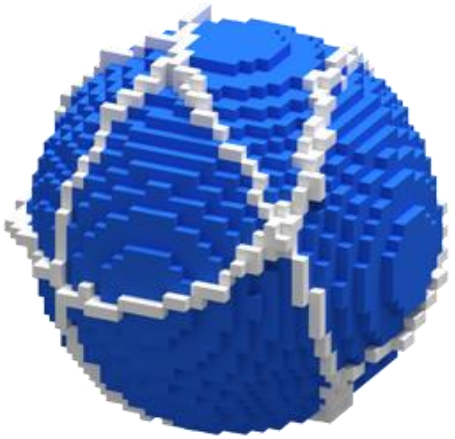
Akka or not Akka?

Оперативная база?

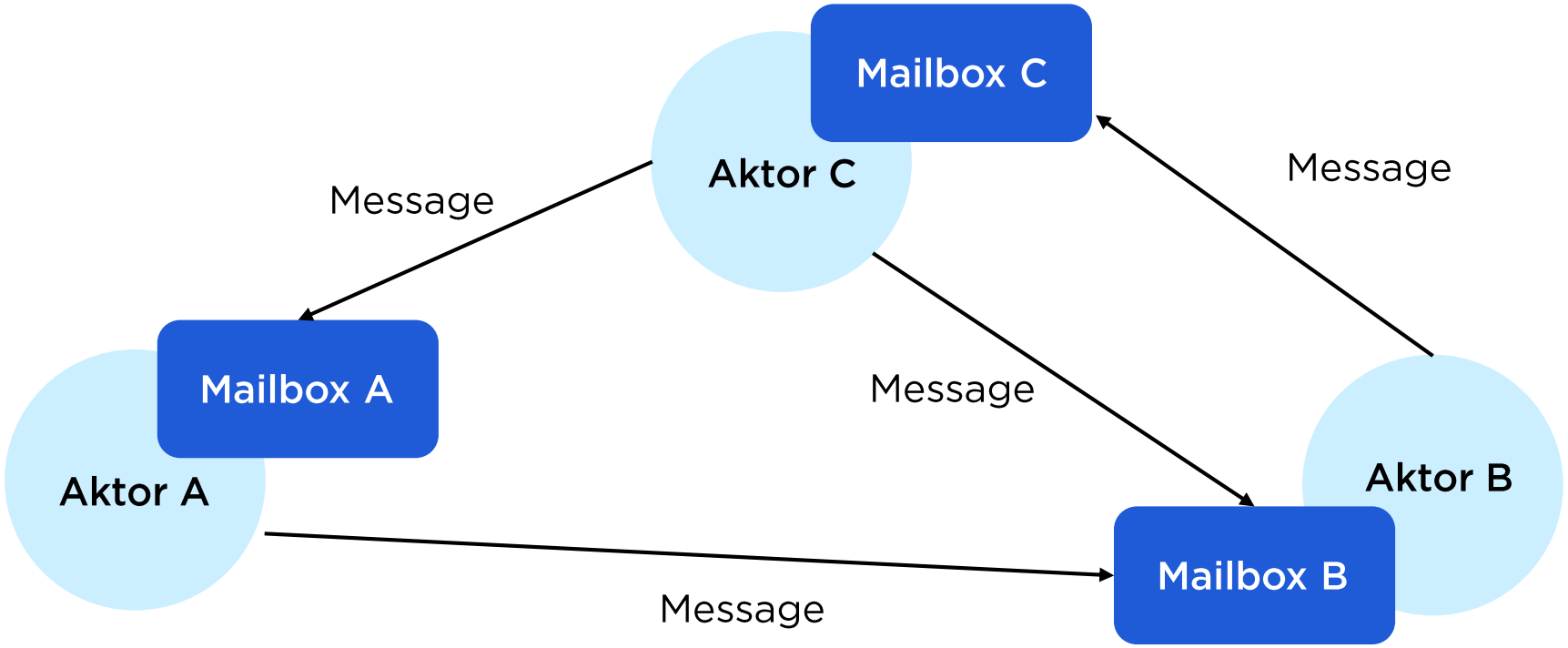
And the winner i-i-i-i-is.... Akka + IMDG Hazelcast!

Акторная модель

Акторная модель – ядро Akka.

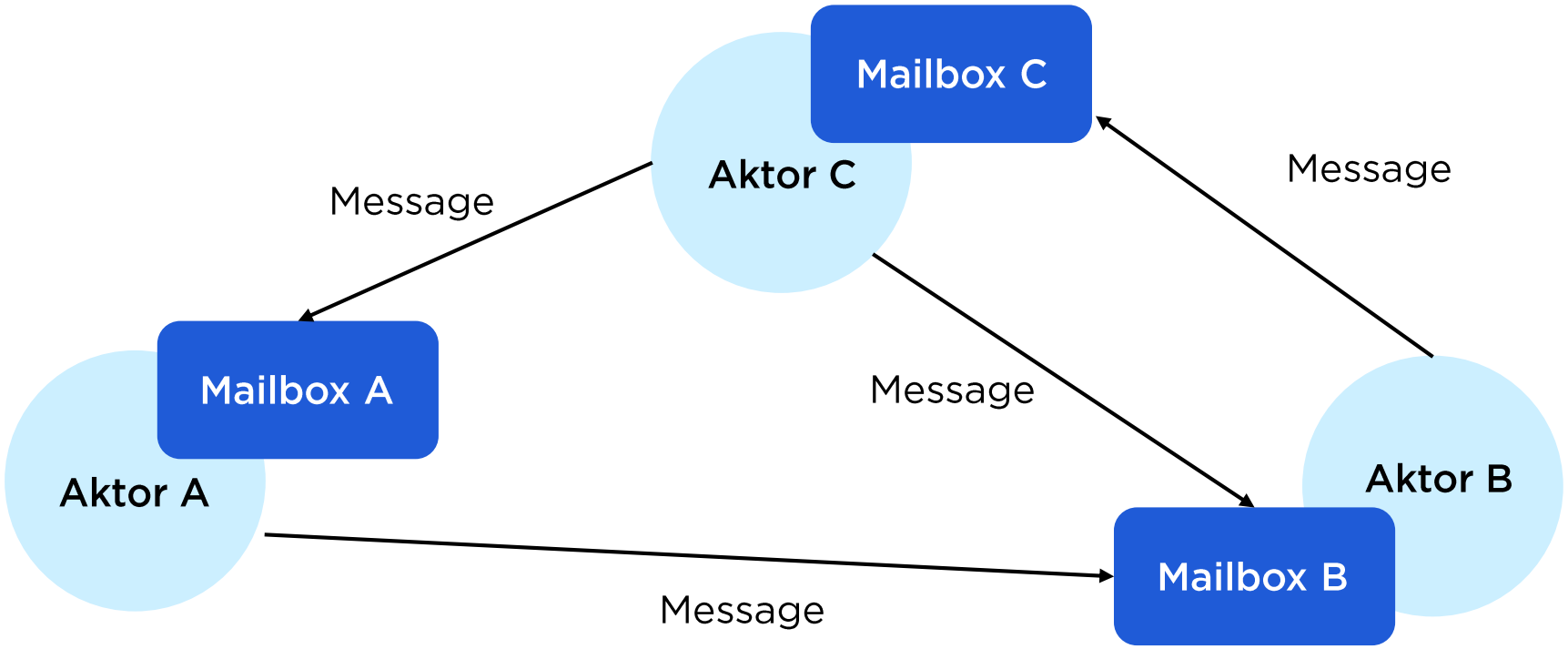


- В 1973 году заложена идея.
- В 1985 окончательно сформировалась.



Основные сущности акторной модели

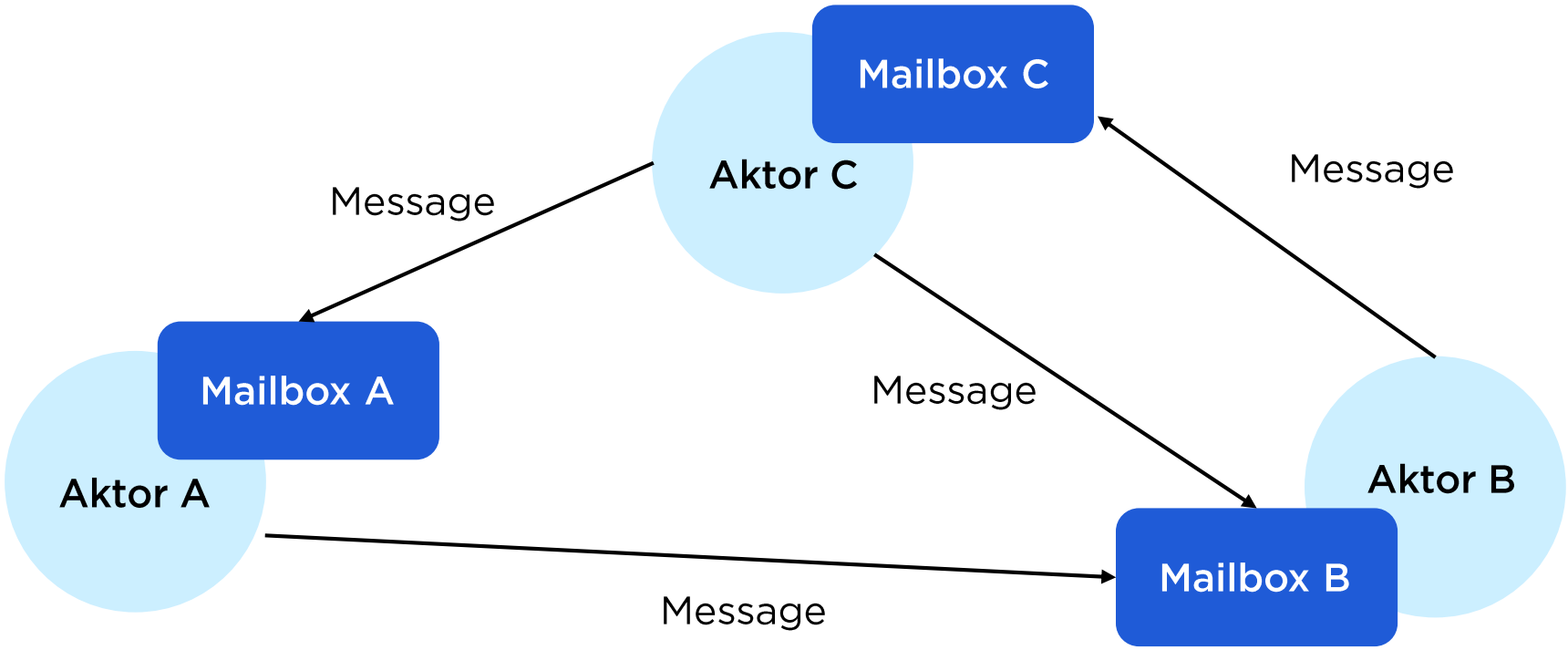
Актор – обработчик данных



Основные сущности акторной модели

Актор – обработчик данных

Сообщение – единица данных для обработки

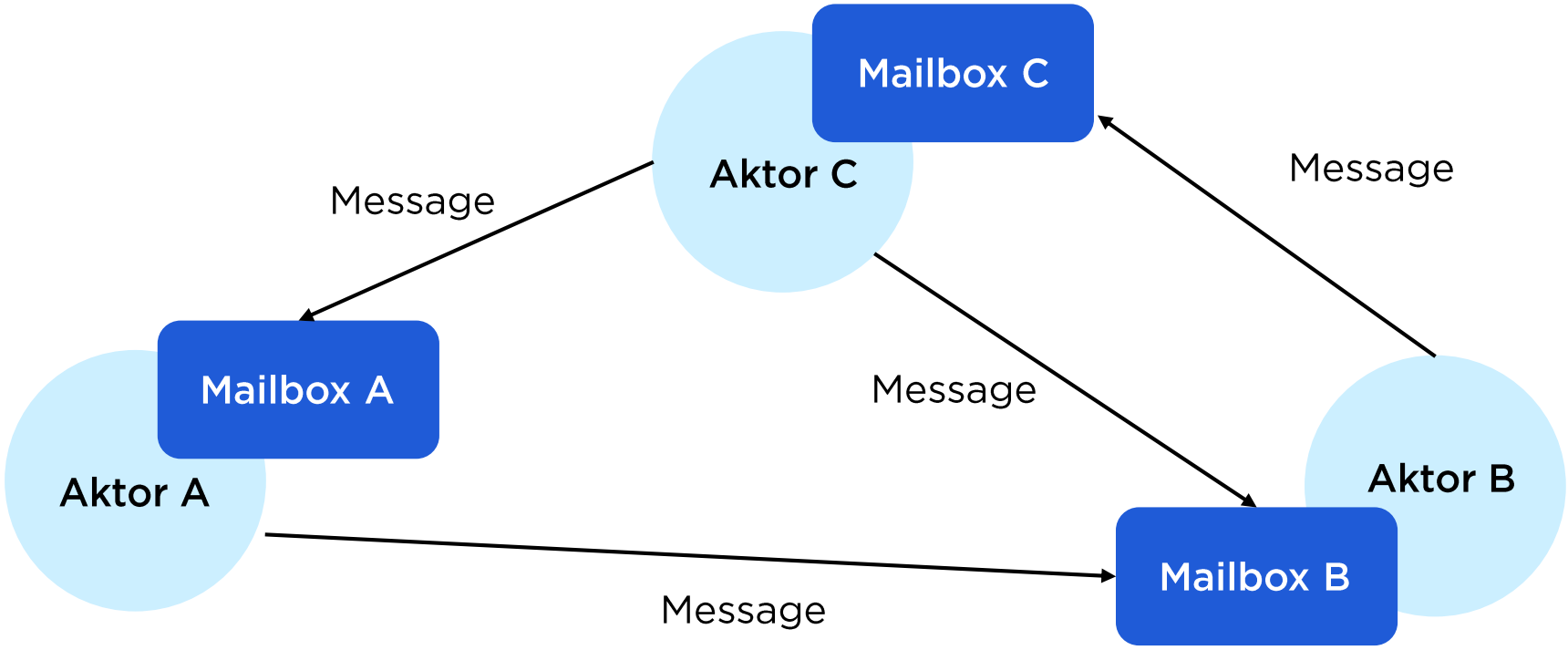


Основные сущности акторной модели

Актор – обработчик данных

Сообщение – единица данных для обработки

Mailbox – хранилище входящих сообщений



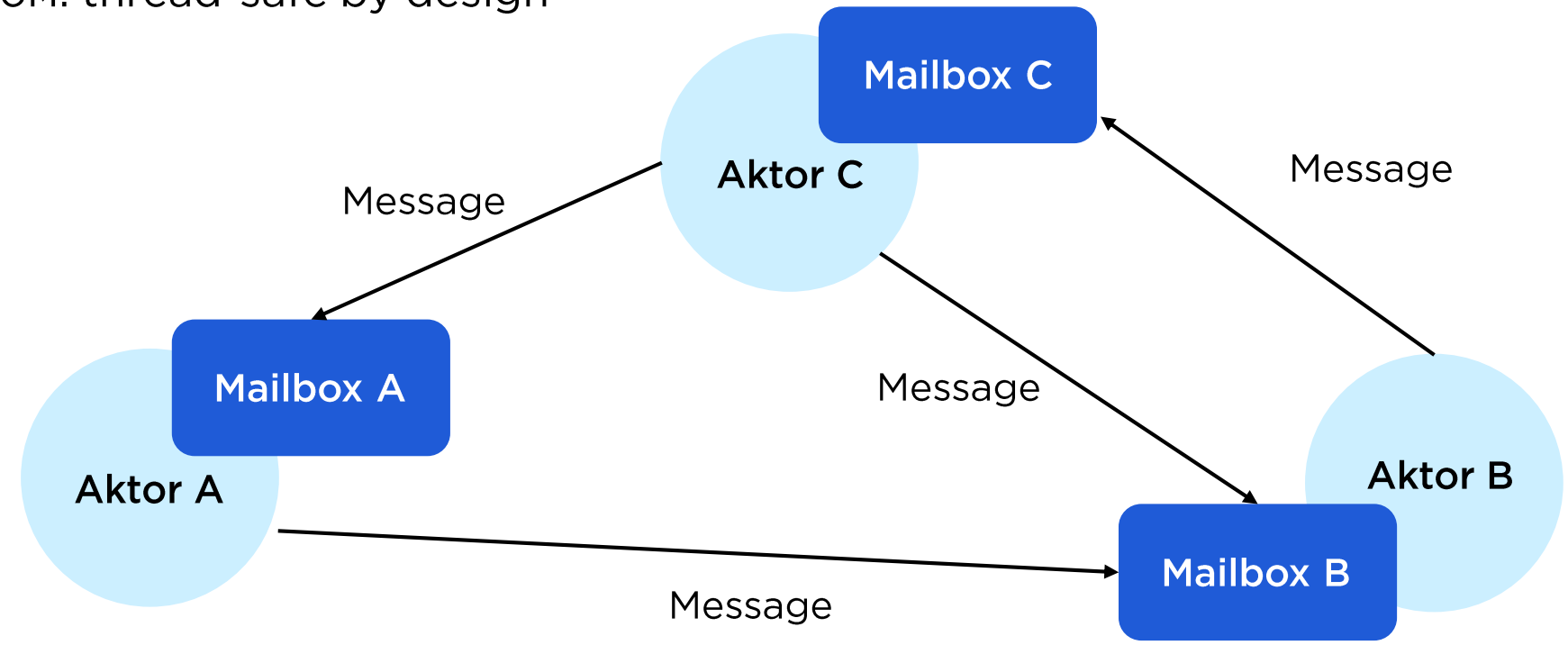
Основные сущности акторной модели

Актор – обработчик данных

Сообщение – единица данных для обработки

Mailbox – хранилище входящих сообщений

> Один обработчик обрабатывает одно сообщение в один момент времени одним потоком: thread-safe by design



Как появилась Akka

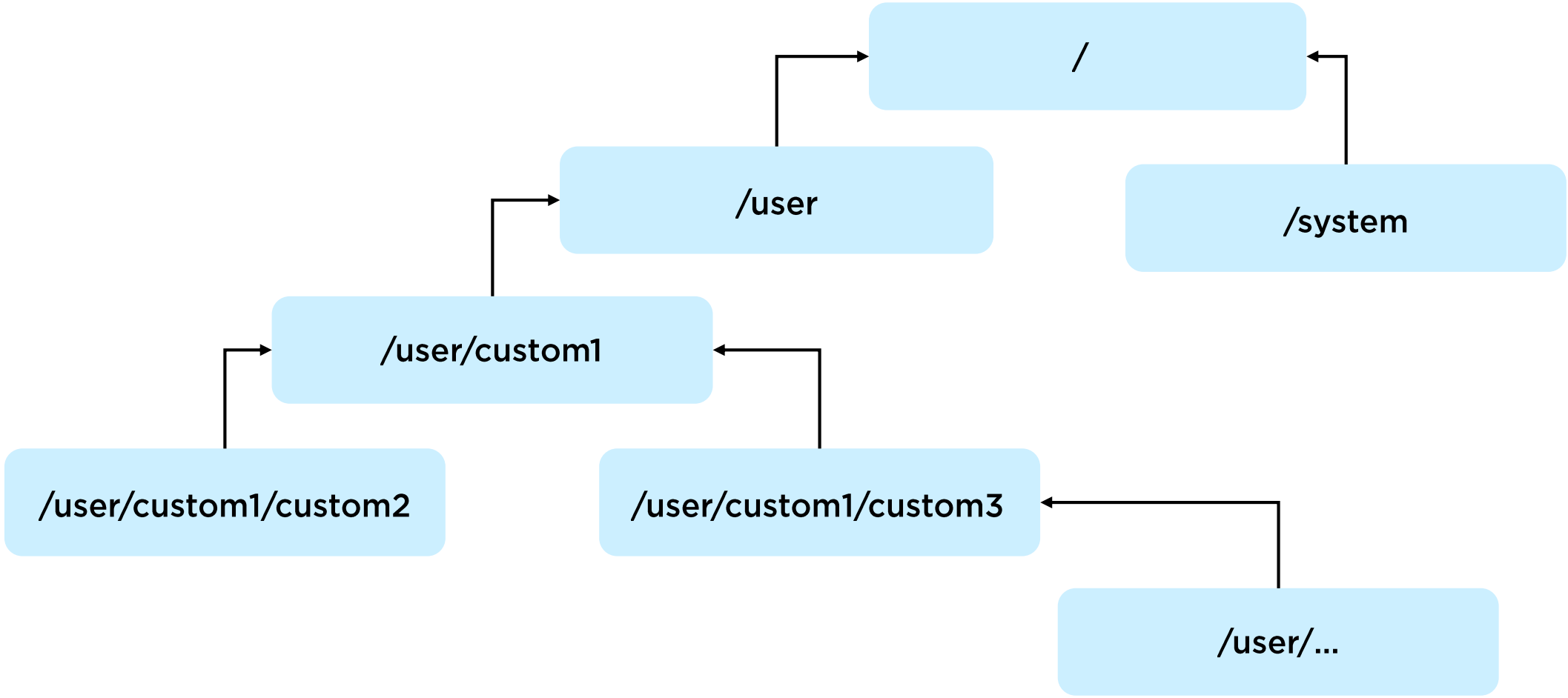
Jonas Bonér + Erlang + акторная модель = Akka

Проекту более десяти лет: версия
0.5 в 2009 -> 2.6.14 в 2021

Lightbend

Акторы

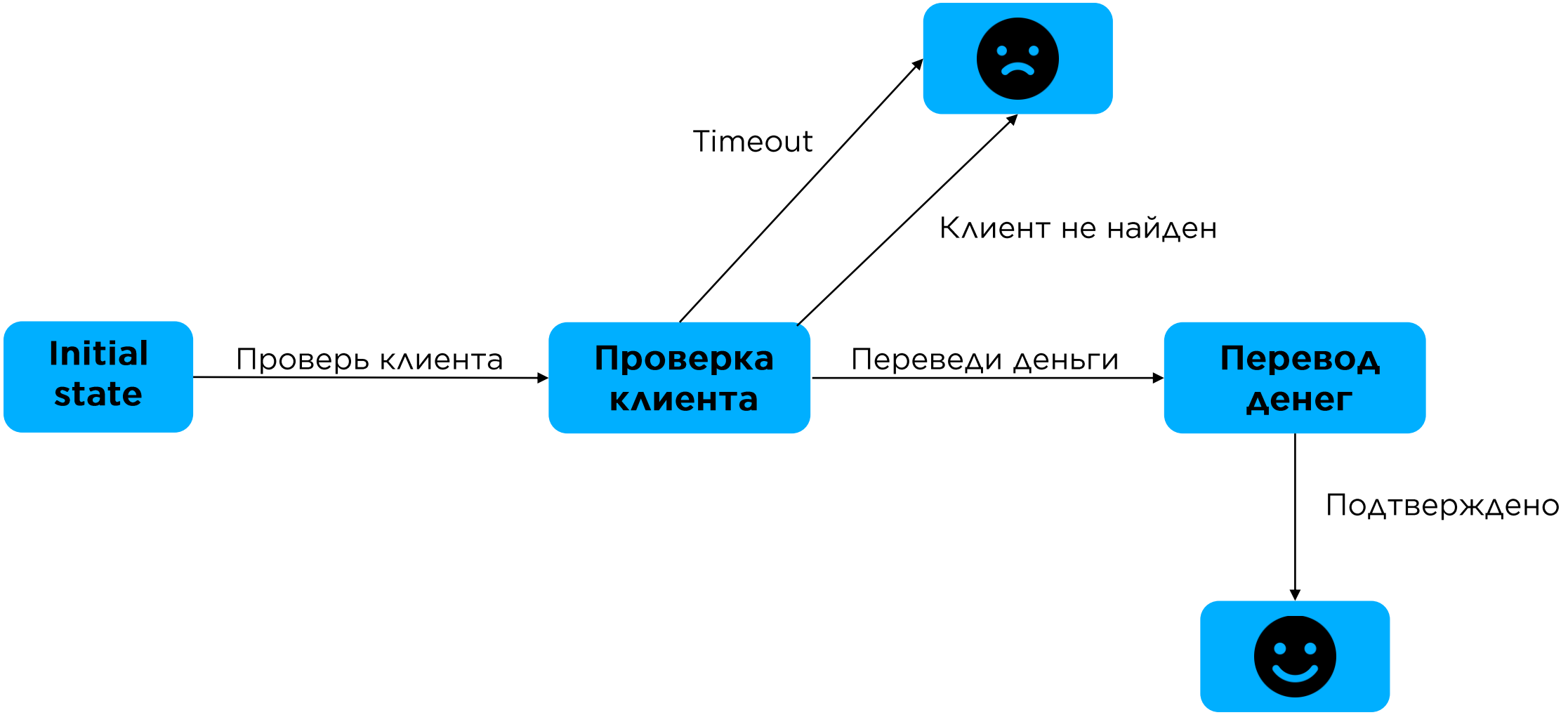
Все акторы находятся в иерархии



Что представляет собой актор

```
public class TransferMoneyActor extends AbstractActor {

    @Override
    public Receive createReceive() {
        return receiveBuilder()
            .match(CheckClient.class, this::checkClient)
            .matchAny(this::unknownMessage)
            .build();
    }

    private void checkClient(CheckClient message) {
        // some logic for checking client
    }

    private void unknownMessage(Object msg) {
        log.error("Received unknown message: {}", msg);
    }
}
```

Машина состояний



Актор и машина состояний

```
public class TransferMoneyActor extends AbstractActor {
    //initial logic with createReceive method

    private void checkClient(CheckClient checkClientMessage) {
        //do some logic for checking client
        getContext().become(transferMoneyBehaviour);
    }

    private final Receive transferMoneyBehaviour = receiveBuilder()
        .match(TransferMoney.class, this::transfer)
        .matchAny(this::unknownMessage)
        .build();

    private void transfer(TransferMoney message) {
        //do some logic for transferring money
         self().tell(PoisonPill.getInstance(), ActorRef.noSender());
    }
}
```

Создание акторов

```
ActorRef transferMoneyActorRef =
        context().actorOf(Props.create(TransferMoneyActor.class));
```

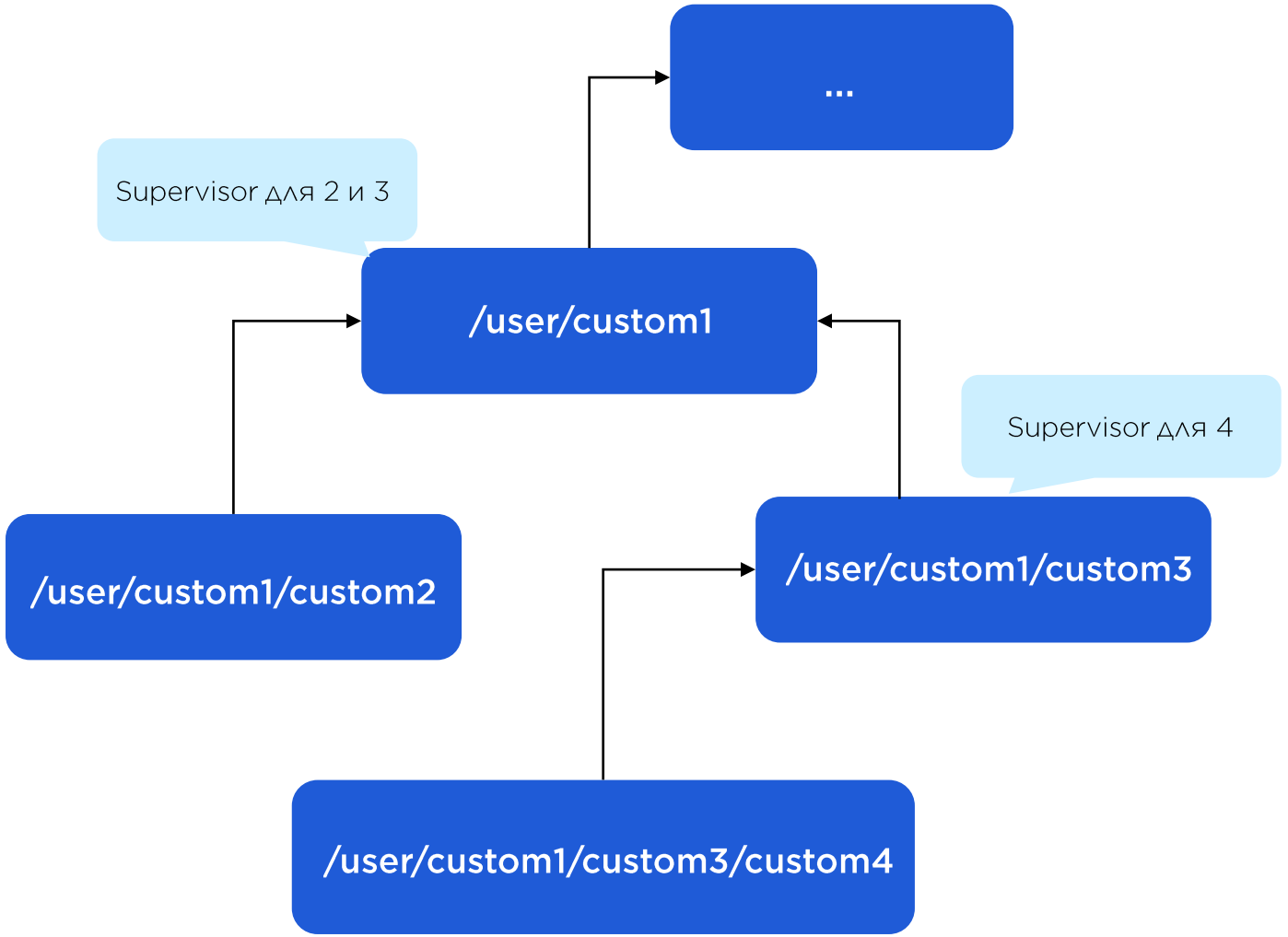
ActorRef – это ссылка на актор, но не сам актор!

Общение акторов

```
ActorRef transferMoneyActorRef =
        context().actorOf(Props.create(TransferMoneyActor.class));

TransferMoney transferMoneyMessage =
                new TransferMoney("+71115555555", "someBank", 1000);

transferMoneyActorRef.tell(transferMoneyMessage, self());
```

Супервизоры

У каждого актора есть супервизор – другой актор, который смотрит за его состоянием



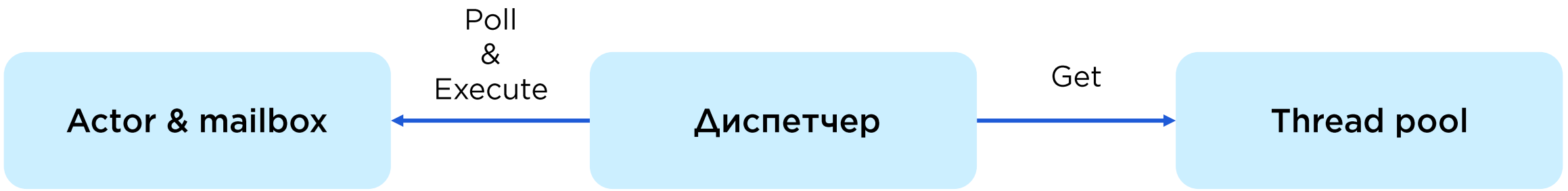
Супервизоры

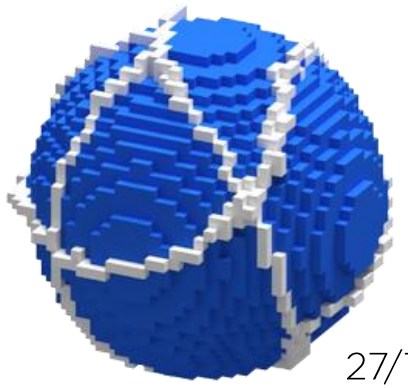
Стратегии при падении:

- поднять упавший актор – с обязательным ограничением по попыткам!
- понять и простить
- погасить все акторы этого типа

Диспетчеры

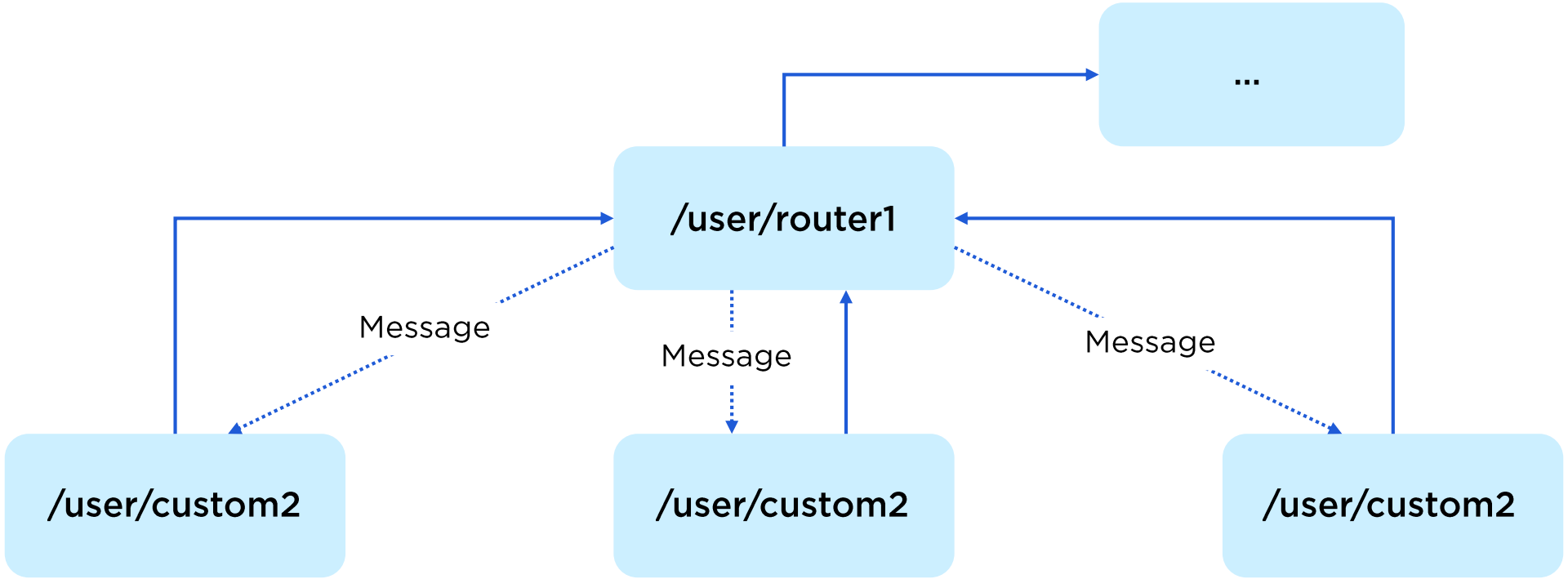
Диспетчер – менеджер обработчиков





Роутеры акторов

Роутеры дают возможность создать и управлять распределением сообщений по пулу однотипных акторов



Конфигурация: HOCON или программно

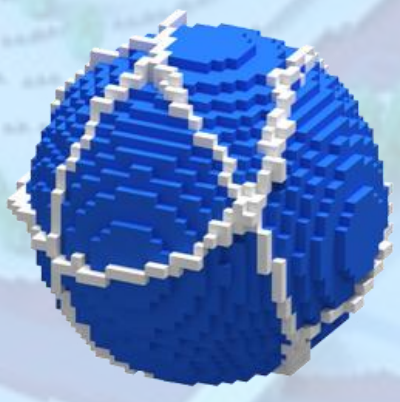
```
akka {
  actor {
    default-dispatcher {
      type = Dispatcher
      executor = "thread-pool-executor"
      thread-pool-executor {
        fixed-pool-size = ${processingThreadPoolSize}
      }
      throughput = 1
    }
  }
}
```

Stateful by design

Где хранить состояние?

- БД
- IMDG
- Пересылать каждый раз
  в самих сообщениях
- В самом обработчике транзакции

Stateful by design



Где хранить состояние, которое не является привычными для нас данными?

Таймаут, через который необходимо отклонить транзакцию, если не получили от банка следующего по логике сообщения.

В Akka из коробки!

```
Cancellable timeoutCancellable = actorSystem.scheduler()
    .scheduleOnce(Duration.ofSeconds(3), self(), "timeout",
                  getContext().dispatcher(), null);


timeoutCancellable.cancel();
```

Тестирование акторов

Akka TestKit – непривычный, но самый удобный инструмент для тестирования работы акторной системы

```
new TestKit(system) {
    {
        TransferMoney transferMoneyMessage =
                new TransferMoney("+71115555555", "someBank", 1000);

        transferMoneyActorRef.tell(transferMoneyMessage);

        TransferMoneyReply reply = new TransferMoneyReply(TransferStatus.OK);
        expectMsg(Duration.of(3, TimeUnit.SECONDS), reply);
    }
}
```

Тестирование акторов

- Что видим в тесте

```
java.lang.AssertionError: assertion failed: timeout (3 seconds) during expectMsgClass waiting
   for class ru.nspk.sbp.transactions.TransferMoneyReply
        at scala.Predef$.assert(Predef.scala:219)
…
        at akka.testkit.TestKit.expectMsgClass(TestKit.scala:896)
        at ru.nspk.sbp.transactions.TransferMoneyTest.shouldOk(TransferMoneyActorTest.java:51)
```

- Что произошло на самом деле

```
[ERROR] [04/09/2021 16:39:36.141] [default-akka.actor.default-dispatcher-5]
[akka://default/system/testProbe-1/$a] Ups
java.lang.NullPointerException: Ups
        at ru.nspk.sbp.transactions.TransferMoneyActor.transfer(TransferMoneyActor.java:121)
        at akka.japi.pf.UnitCaseStatement.apply(CaseStatements.scala:26)
```

Тестирование акторов

- Все асинхроное

- Основной помощник в нахождении ошибки – логи.
Иногда помогает дебаг

Основа кластера

Akka Cluster

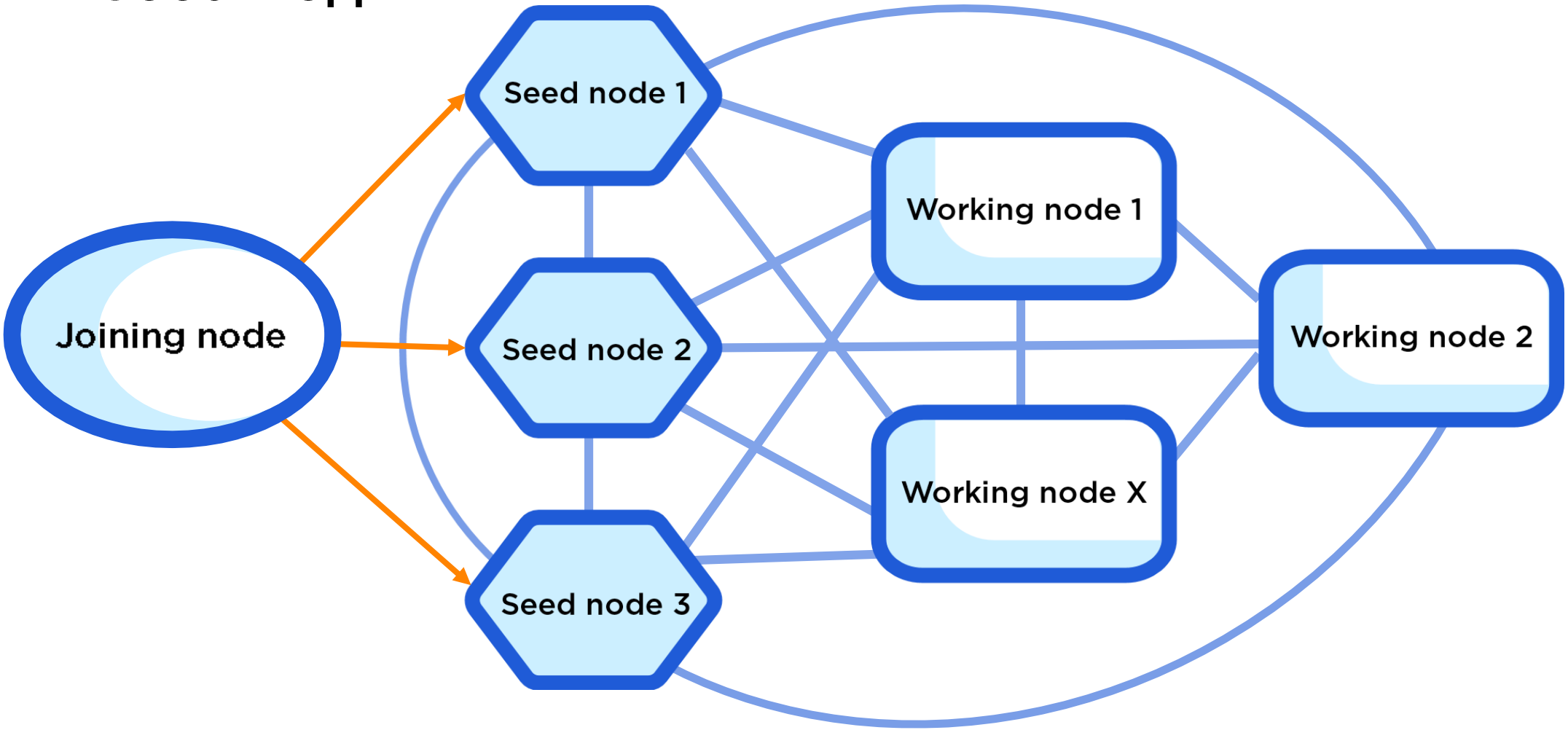
Удаленный вызов акторов

Akka Remoting

Artery Remoting

Вход в кластер

Seed-ноды



Конфигурация кластера

```
cluster {
  roles = ["TransferMoneyRole"]
  downing-provider-class = com.ajjpj.simpleakkadowning.SimpleAkkaDowningProvider
  use-dispatcher = "akka.cluster.default-cluster-dispatcher"
  default-cluster-dispatcher {
    type = Dispatcher
    executor = "thread-pool-executor"
    thread-pool-executor {
      fixed-pool-size = 30
    }
    throughput = 1
  }
  failure-detector {
    acceptable-heartbeat-pause = 10s
  }
}
```

Вход в кластер

```
ActorSystem actorSystem = ActorSystem.create();
List<Address> seedNodes = new ArrayList<Address>(){{
    add(new Address("akka.tcp", actorSystem.name(), "172.16.0.1", 1234));
}};

Cluster cluster = Cluster.get(actorSystem);
cluster.joinSeedNodes(seedNodes);
//You're in the cluster now!

//Get proxy for actor in cluster to call it
ClusterSingletonProxySettings proxySettings =
ClusterSingletonProxySettings.create(actorSystem).withRole("TransferMoneyRole");

ActorRef myActorProxy = system.actorOf(
     ClusterSingletonProxy.props("/user/transferMoneyActor", proxySettings),
                                 "transferMoneyActorProxy");
```

Ролевая модель в кластере

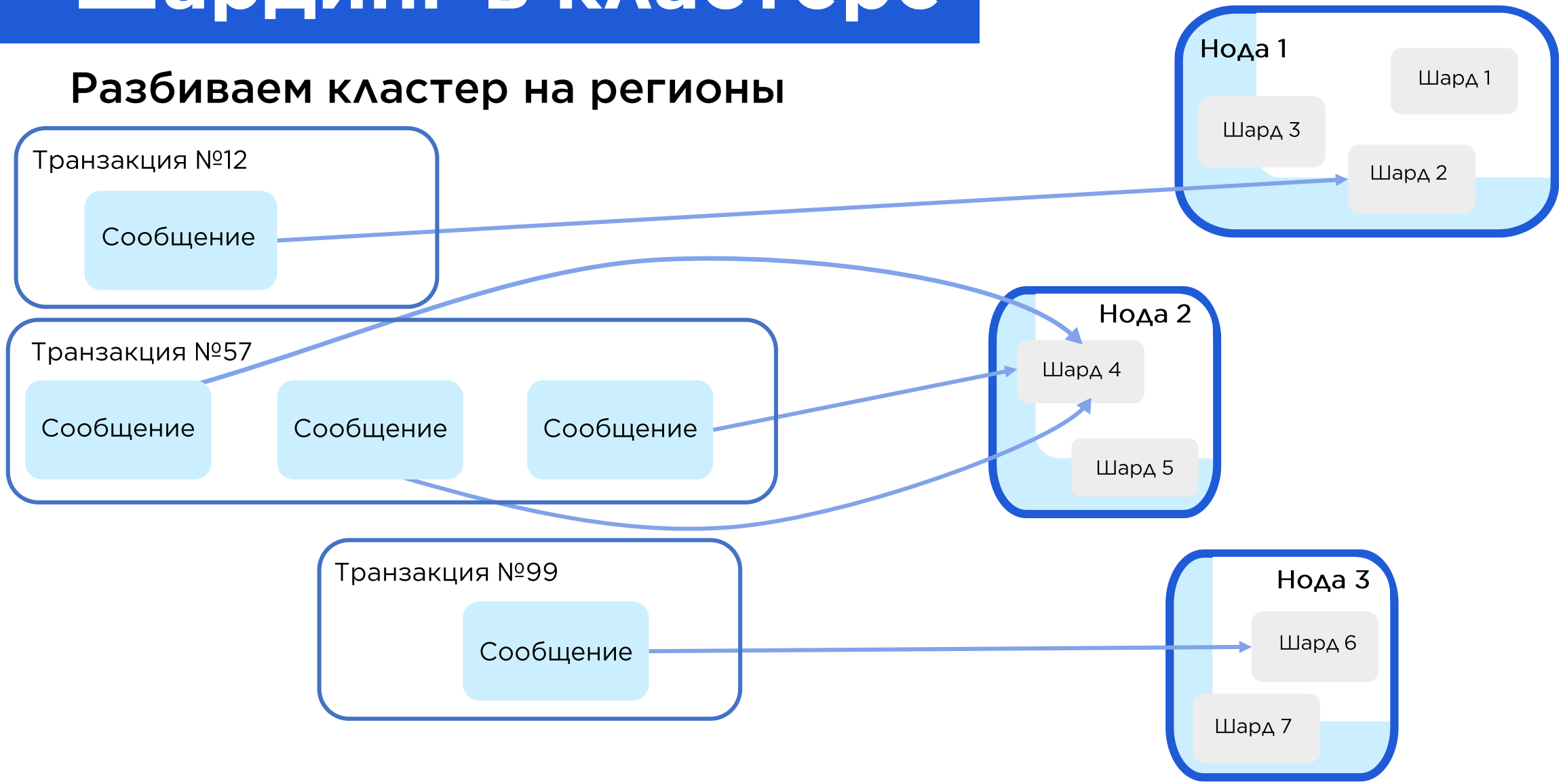
В кластере функциональность декомпозирована по разным нодам?

Просто добавь ролей!

```
ClusterSingletonProxySettings proxySettings =
     ClusterSingletonProxySettings.create(actorSystem)
     .withRole("TransferMoneyRole");

cluster {
  roles = ["TransferMoneyRole"]
  …
}
```

Шардинг в кластере

Разбиваем кластер на регионы



Шардинг в кластере

Кластер шардинг – это как кластеризованный HashMap

```
public class TransferMoneyToShardingExtractor implements MessageExtractor {
    public static final int SHARDS = 257;

    @Override
    public String entityId(final Object message) {
        if (message instanceof TransferMoney) {
            return (((TransferMoney) message).getTransactionId();
        }
        return null;
    }

    @Override
    public Object entityMessage(final Object message) { return message; }

    @Override
    public String shardId(final Object message) {
        String entityId = entityId(message);
        if (entityId != null) {
            return String.valueOf(entityId.hashCode() % SHARDS);
        } else {
            return null;
        }
    }
}
```

Настройка кластера с шардированием

Сначала обычная настройка акторной системы и самого кластера

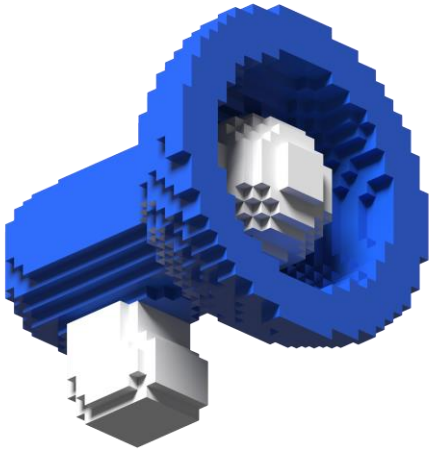
```
ClusterShardingSettings clusterShardingSettings = ClusterShardingSettings
        .create(actorSystem).withRole("TransferMoneyRole");


ActorRef transferMoneyRootRef = sharding.start("transferMoneySharding",
           Props.create(TransferMoneyActor.class),
           clusterShardingSettings,
           new TransferMoneyToShardingExtractor());


TransferMoney transferMoneyMessage =
              new TransferMoney("+71115555555", "someBank", 1000);
transferMoneyRootRef.tell(transferMoneyMessage, self());
```

Шардинг в кластере

Количество шардов
должно быть с запасом!

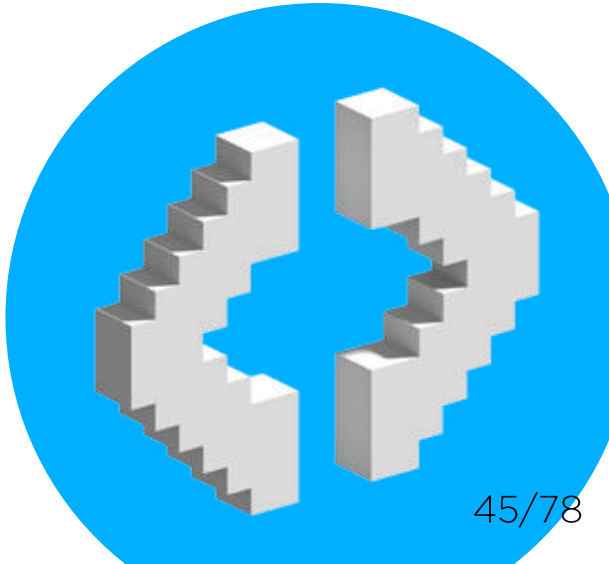
Что получаем от кластера?

Ролевую модель нод
и Service discovery



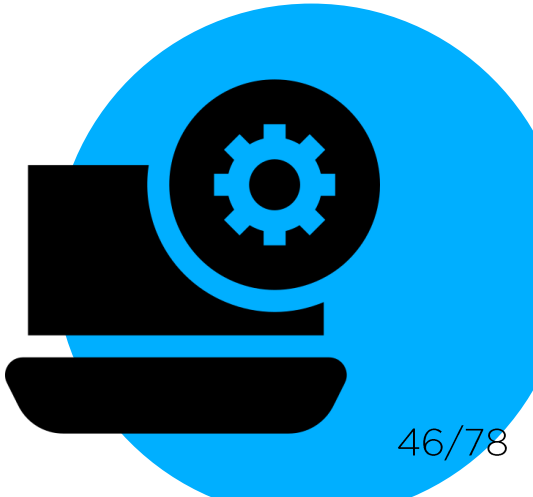
Что получаем от кластера?

Горизонтальная масштабируемость
и распределение нагрузки по нодам



Что получаем от кластера?

Определение недоступных нод



Что получаем от кластера?

Перераспределение шардов/регионов на другие ноды в случае, если одна из них упала

Что получаем от кластера?

Автоматическое пересоздание акторов упавшей ноды при использовании шардинга

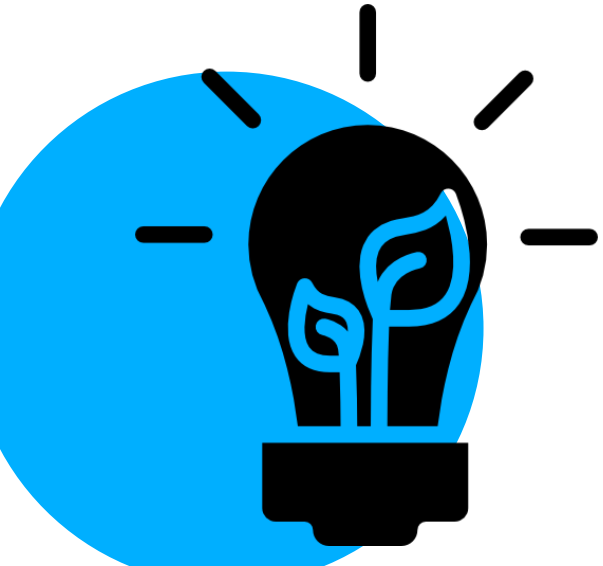
akka.cluster.sharding.remember-entities = on

Akka + Spring Boot

- Удобные актуаторы
- Spring Boot Admin
- Spring Cloud Config
- И, собственно, сам DI

Решение готово!

Profit!!!



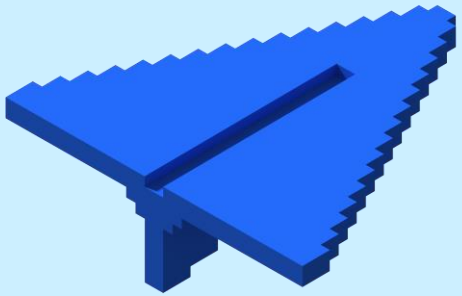
- Акторная модель в реализации Akka
- Akka TestKit
- Akka Cluster Sharding
- Spring Boot с экосистемой

Прошло два с лишним года

Lessons learned

Тот самый stateful

Stateful – внимательно следим
за хранимым контекстом!



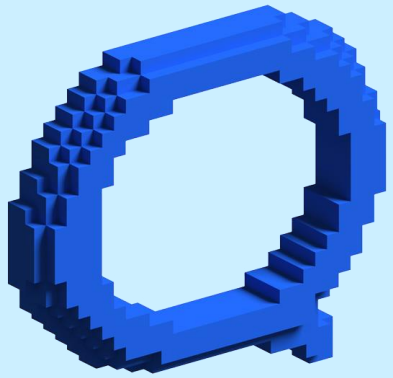
Не очень хороший актор

```
public class TransferMoneyActor extends AbstractActor {
     private ClientFull clientReceiverFull;

    private void checkClient(CheckClient checkClientMessage) {
        clientReceiverFull = getAllClientInfo(chechClientMessage.getClientId());
        someChecks(clientReceiverFull);
        getContext().become(transferMoneyBehaviour);
    }

    private final Receive transferMoneyBehaviour = receiveBuilder()
        .match(TransferMoney.class, this::transfer)
        .matchAny(this::unknownMessage)
        .build();

    private void transfer(TransferMoney message) {
         transfer(clientReceiverFull.getClientId(), message.getClientSenderId(), message.getSum());
         self().tell(PoisonPill.getInstance(), ActorRef.noSender());
    }
}
```

Вполне себе актор

```
public class TransferMoneyActor extends AbstractActor {
    private TransferContext context = new TransferContext();

    private void checkClient(CheckClient checkClientMessage) {
        ClientFull clientReceiverFull = getAllClientInfo(chechClientMessage.getClientId());
        someChecks(clientReceiverFull);
        context.setClientReceiverId(clientReceiverFull.getClientId());
        getContext().become(transferMoneyBehaviour);
    }

    private final Receive transferMoneyBehaviour = receiveBuilder()
        .match(TransferMoney.class, this::transfer)
        .matchAny(this::unknownMessage)
        .build();

    private void transfer(TransferMoney message) {
         transfer(context.getClientReceiverId(), message.getClientSenderId(), message.getSum());
         self().tell(PoisonPill.getInstance(), ActorRef.noSender());
    }
}
```

Может в IMDG?



IMDG тоже не резиновый!

Как уменьшили стейт на порядок?

- 99.99% времени жизни транзакции приходится на ожидание запроса, которому нужно минимум данных? Значит в это время храним только этот минимум данных!
- А точно ли нужен stateful сервис, там где он не нужен?
- Не надо хранить данные, которое более не будут использоваться

Вывод: хранить надо только то, что нужно, и столько, сколько нужно!

Метрики Akka

Не слишком большой размер очереди в mailbox’ах?

¯_(ツ)_/¯

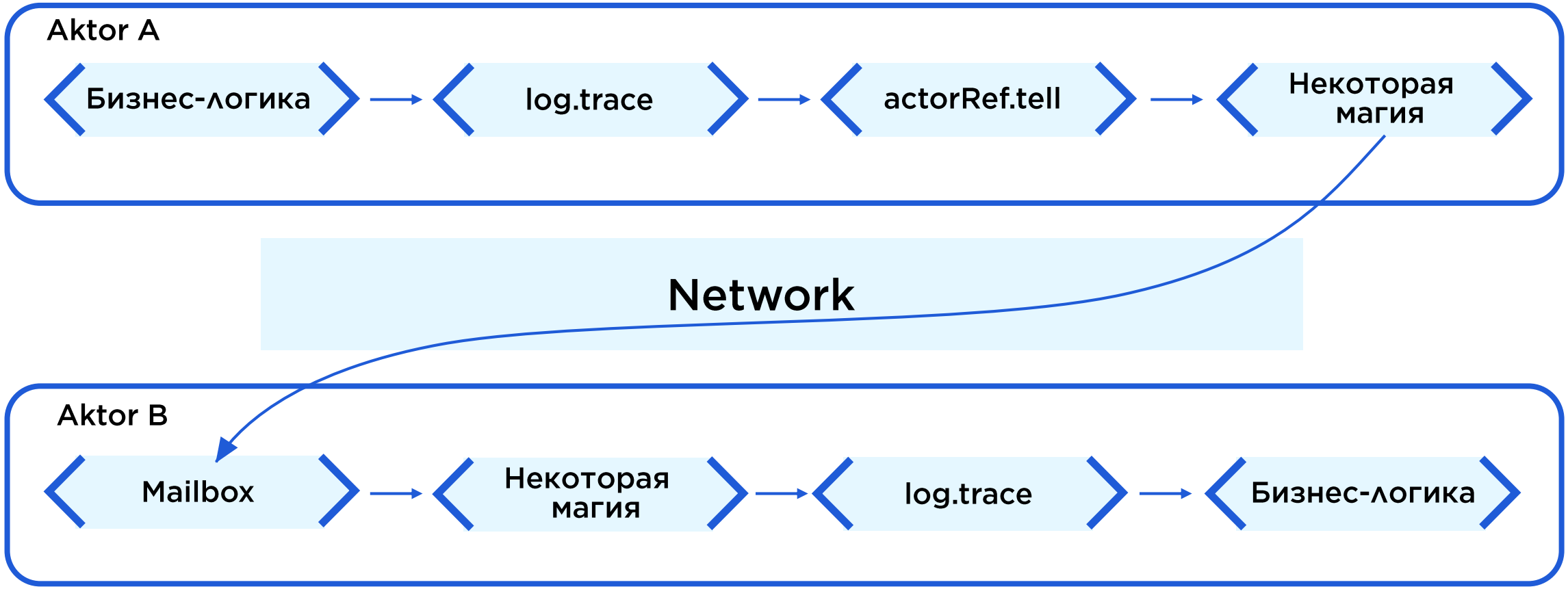
Метрики Akka

Как быстро обрабатываются сообщения акторами?

¯_(ツ)_/¯

Метрики Akka

А где именно сейчас сообщение?



Метрики Akka. micrometer-akka

- 2 актора на каждое сообщение
- 2 актора на каждую транзакцию
- 50 тпс
- 6 метрик на каждый актор

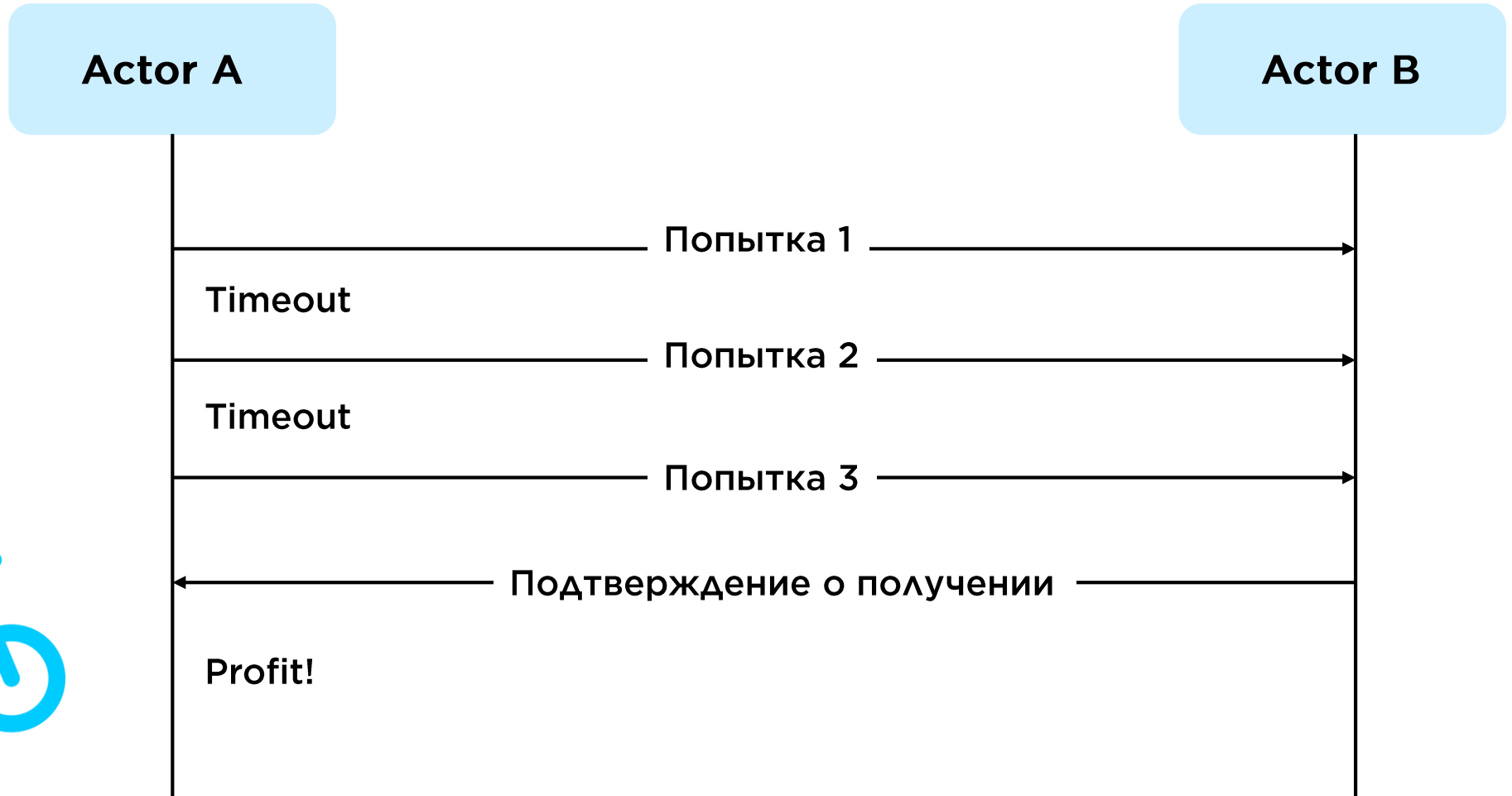
(2 * 14 + 2) * 50 * 6 = 9000 метрик в секунду. Или 32.4 млн в час

micrometer-akka

Ты,
хотящий метрики

Гарантированная доставка

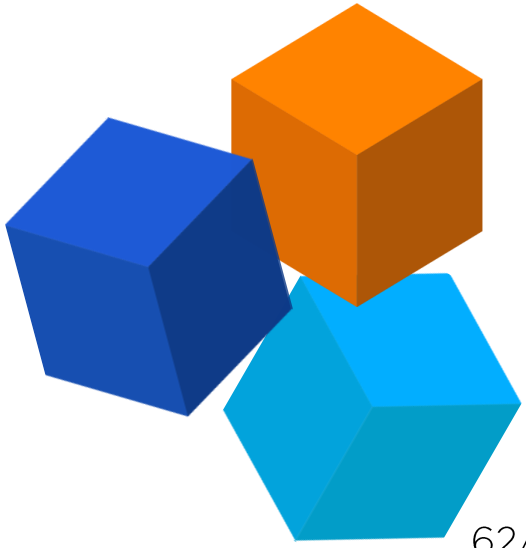
Собственный велосипед



Гарантированная доставка

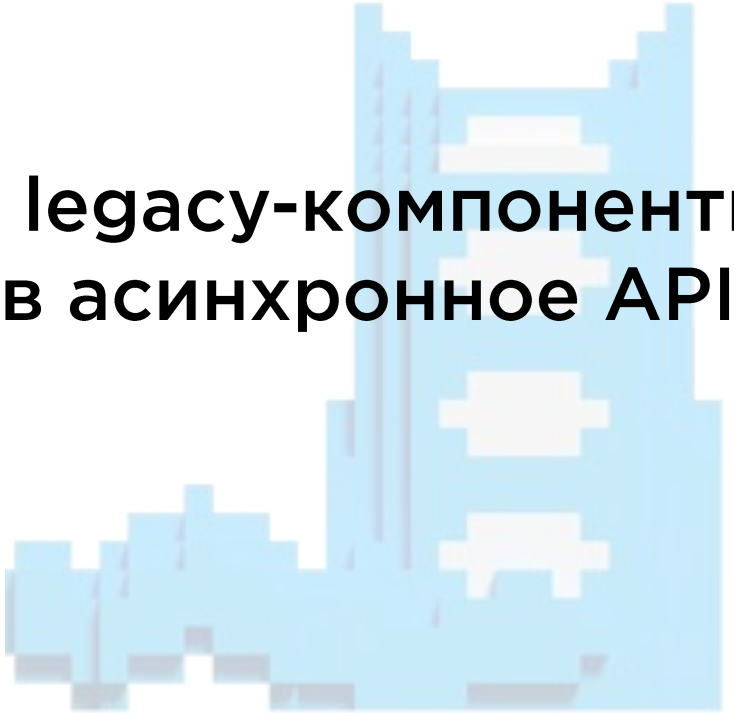
Теперь поставляется из коробки:

akka.persistence.AbstractPersistentActorWithAtLeastOnceDelivery



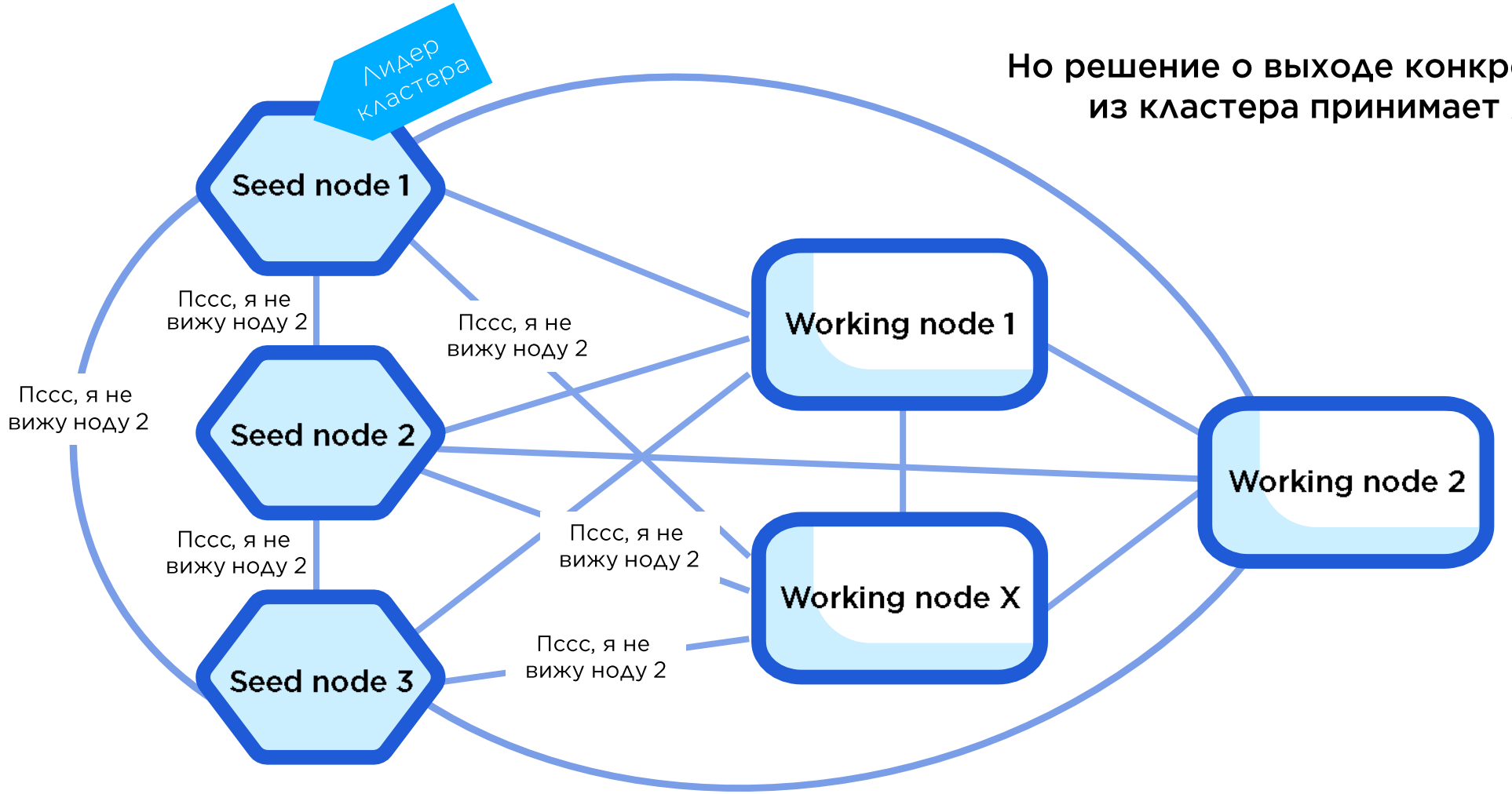
Интеграция с legacy-компонентами

Некоторые legacy-компоненты не умеют
в асинхронное API



Политика выхода нод из кластера

Протокол Gossip



Но решение о выходе конкретной ноды из кластера принимает лидер-нода

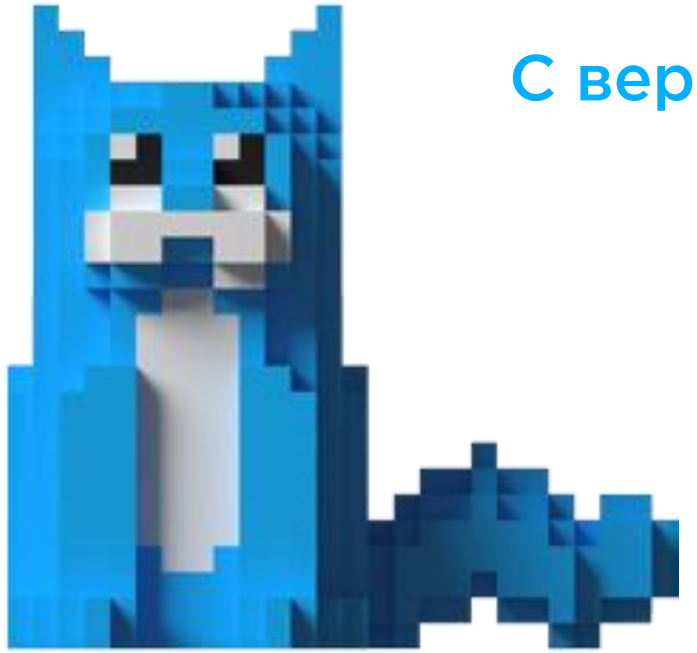
Политика выхода нод из кластера

Но если проблема связности, то как самой ноде узнать, что ей пора остановиться?

Кворум!

Downing provider

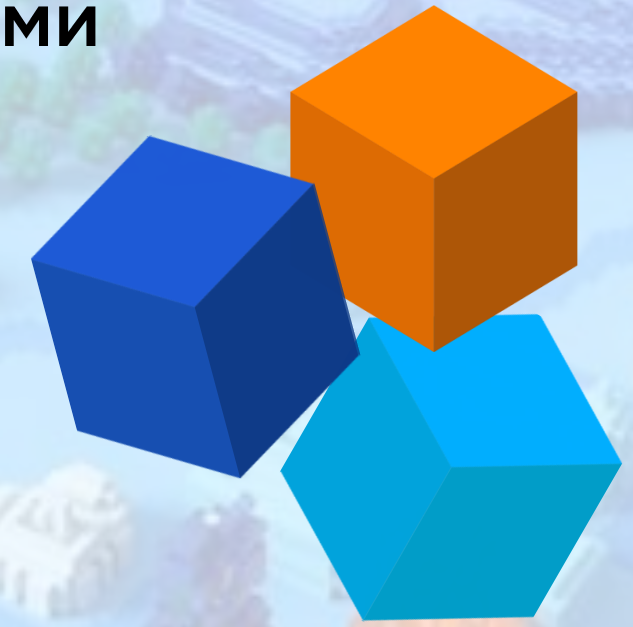
Раньше были бесплатные только опенсорсные от энтузиастов



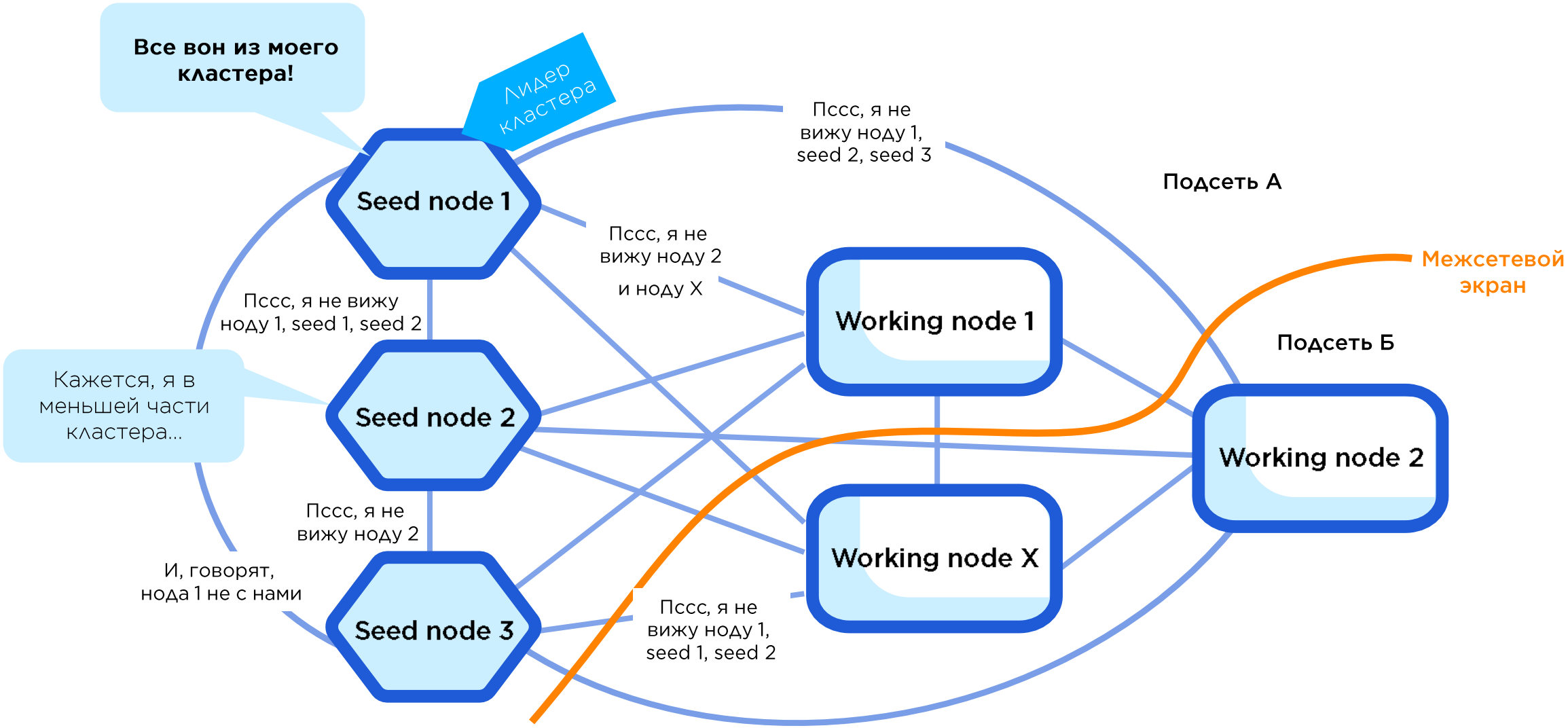
С версии 2.6 поставляется из коробки – SplitBrainResolverProvider

Мультицодовая конфигурация

Кластер не должен быть разделен какими-либо инфраструктурными компонентами

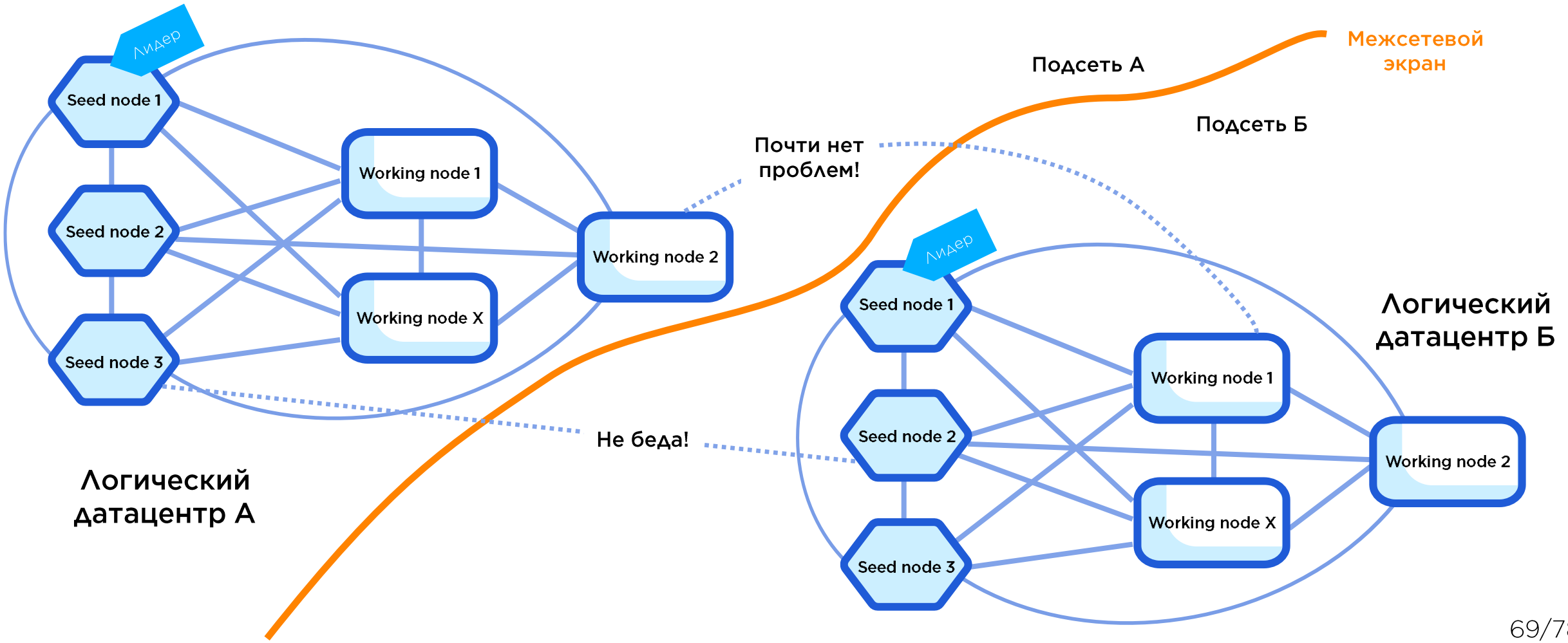


Мультицодовая конфигурация



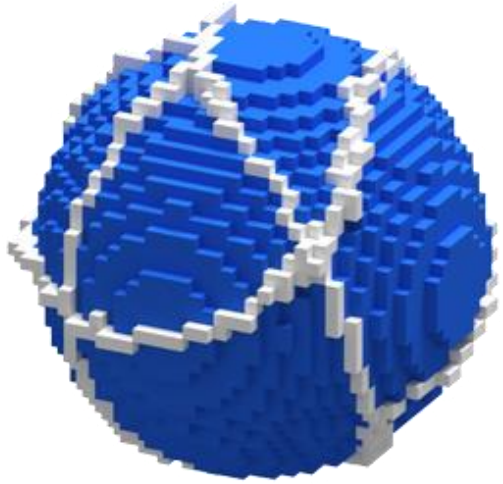
Мультицодовая конфигурация

Акка позволяет задать логический датацентр
для каждого участника кластера



Мультицодовая конфигурация

```
ActorRef transferMoneyRootRef = sharding.start(
            "transferMoneySharding",
            "TransferMoneyRole",
            "datacenterA",
            new TransferMoneyToShardingExtractor());
```



Но корректнее –
отдельный кластер в каждой подсети

Akka и Java: разные примитивы

В Java, как языке ООП, нет базовых сущностей акторной модели

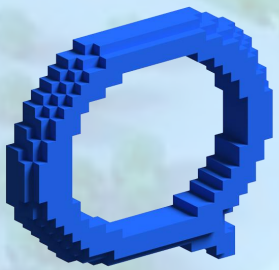
```
@Override
public Receive createReceive() {
    return receiveBuilder()
        .match(TransferMoney.class, this::transfer)
        .matchAny(this::unknownMessage)
        .build();
}

private void transfer(TransferMoney message) {
    //do some logic for transferring money
    TransferMoneyReply reply = new TransferMoneyReply(TransferStatus.OK);
    sender().tell(reply, self());
}
```

Akka и Java: отсутствие типизации

До версии 2.5 не было удобных типизированных акторов. ActorRef в качестве типа любого актора

```
ActorRef checkClientActorRef = context().actorOf(Props.create(CheckClientActor.class));
ActorRef transferActorRef = context().actorOf(Props.create(TransferMoneyActor.class));

checkClientActorRef.tell(checkClient, self());
transferActorRef.tell(transferMoney, self());


transferActorRef.tell(checkClient, self()); //абсолютно нет проблем при компиляции



akka.actor.typed.ActorRef<TransferProtocol> transferTypedActorRef =
      Adapter.spawn(actorSystem, new TransferMoneyTypedActor(), "transferActor");
transferTypedActorRef.tell(transferMoneyProtocolImpl);
```

Нужно ли понимать multi-threading?



Да, и лучше бы понимать его еще глубже

Комьюнити и обучение

Akka – не самый распространенный инструмент

StackOverflow уже не всегда спасет ОРД

Комьюнити и обучение

Виктор Тесленко

Нужен продвинутый курс?

Не забудьте сами найти тренера
и составить список интересующих тем!

Ну или обратитесь к нам, дадим наводочку ☺

KISS!

Принцип актуален, как никогда.

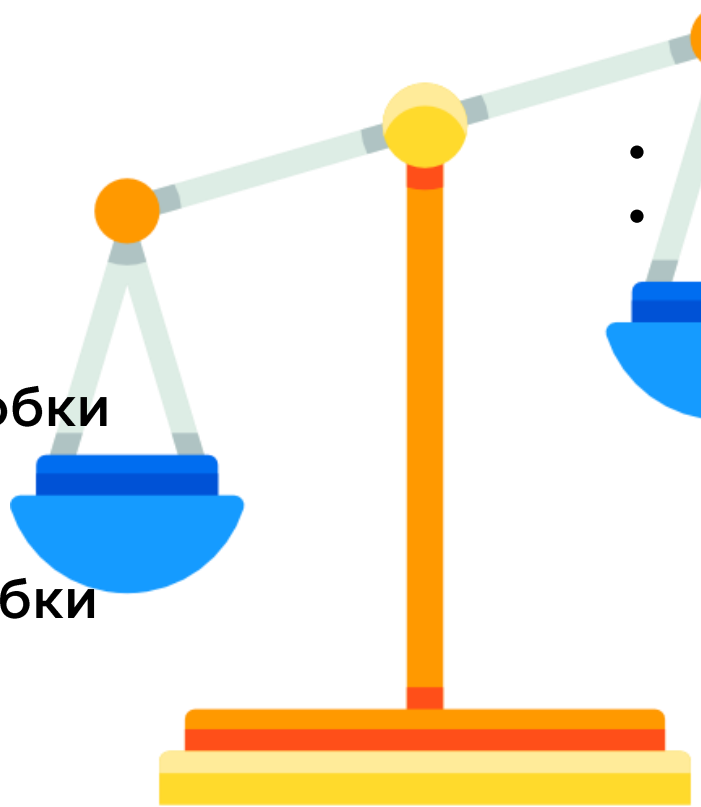
K	KEEP
I	IT
S	SIMPLE
S	STUPID

Выводы

Akka – мощнейшая экосистема, позволяющая решать сложные задачи.

Но надо понимать цену ее использования.

- Мощный кластер из коробки
- Решение проблем с multi-threading
- Удобный stateful из коробки



- Меняется подход к разработке
- Высокий порог вхождения

Спасибо за внимание!

Вопросы?