Фантастические акторы

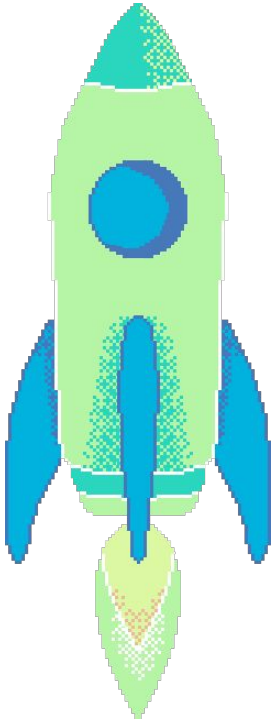
и где они обитают

Андрей Парамонов

Directed by
DODO ENGINEERING

Цель

- Рассказать
- Показать
- Обсудить



Парамонов Андрей

a.paramonov@dodopizza.com

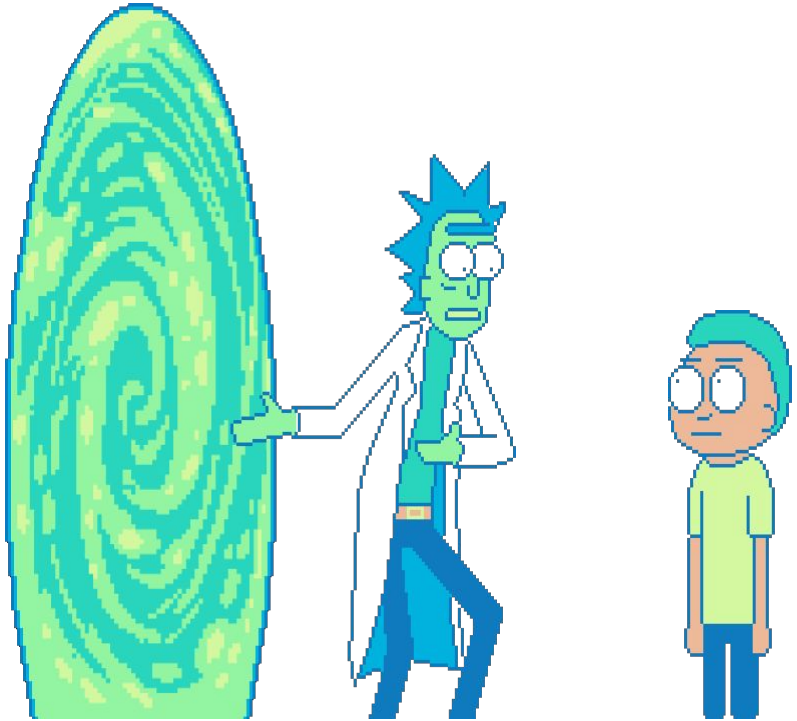
@Pr1vetAndrey

@Pr1vetAndrey

- Software engineer
- TechLead
- 7 лет – один .NET

О чём поговорим?

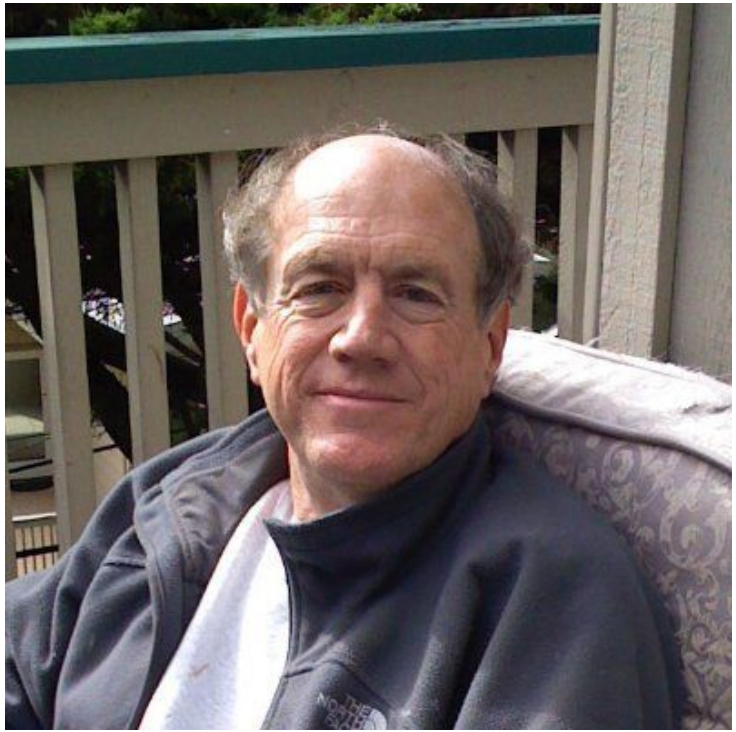
- Акторная модель
- Распределённые акторы
- Пример



Начало

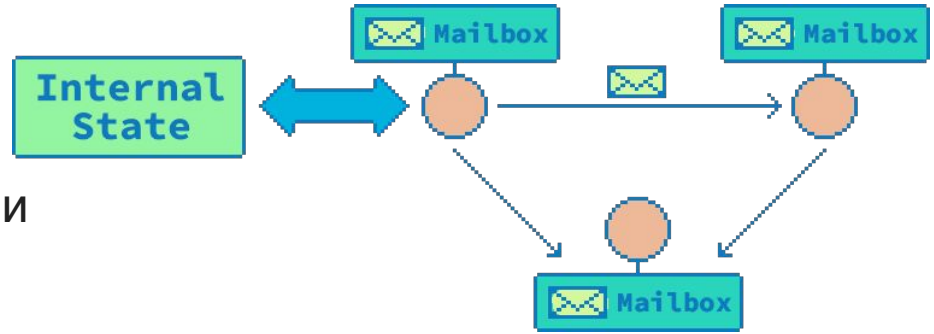
Hewitt, Carl; Bishop, Peter; Steiger, Richard (1973). "A Universal Modular Actor Formalism for Artificial Intelligence"

https://professorhewitt.blogspot.com/



Actor model — что это?

- Модель параллельных вычислений (1973 г.)
- Актор — примитив параллельно выполняемого действия
- Акторы обмениваются сообщениями

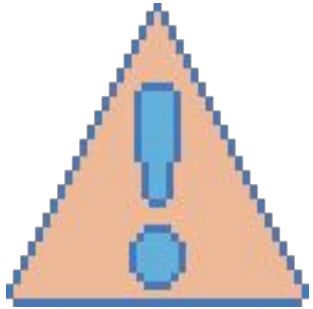


Who is mr. Актор

- примитив (всё — актор)
- работает параллельно с другими акторами
- имеет входную очередь сообщений
- получает сообщения извне или от других акторов
- обрабатывает одно сообщение в любой момент времени и полностью*
- имеет «адрес»

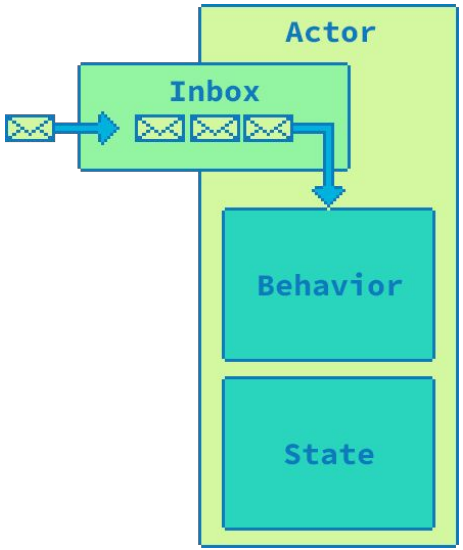
Но

- отсутствуют ограничения на порядок прибытия сообщений
- отсутствуют гарантии доставки сообщения
- отсутствует синхронная коммуникация



Обработка сообщений

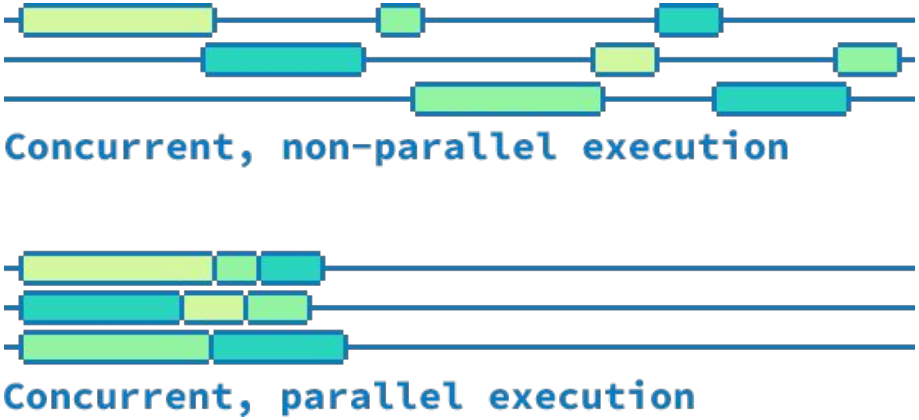
- отослать конечное число сообщений другим акторам*
- создать конечное число новых акторов
- поменять своё состояние
- поменять поведение для обработки следующих сообщений
- выполнить произвольное действие



** ответить отправителю 0 или более раз*

Выполнение

Actor model = ~~Parallelism~~ Concurrency



Concurrency is not Parallelism by Rob Pike
https://www.youtube.com/watch?v=oV9rvDllKEg

Проблемы стандартного подхода

- Не так поняли ООП
- Многопоточное программирование сложное!

Dr. Alan Kay on the Meaning of “Object-Oriented Programming”

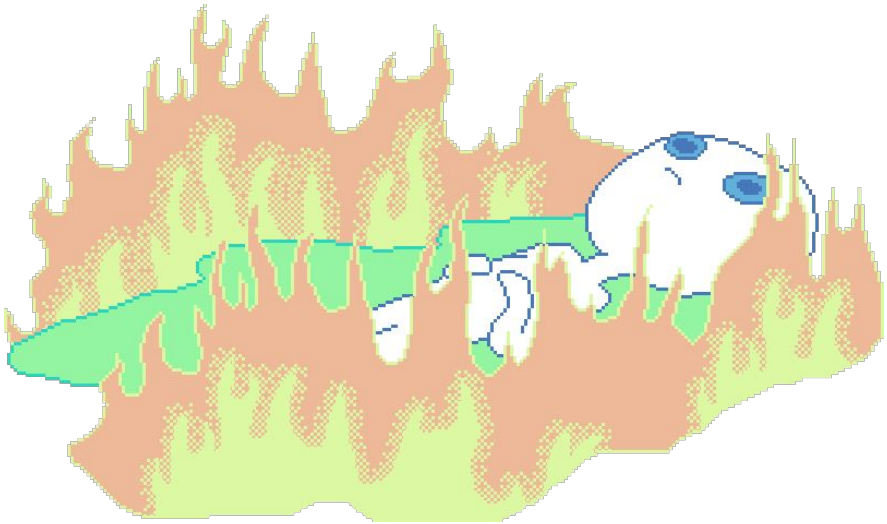
OOP to me means only messaging, local retention and protection and hiding of state-process, and extreme late-binding of all things.

ООП для меня означает только обмен сообщениями, локальное состояние, защиту и сокрытие состояния-процесса и крайне позднее связывание всех вещей.

https://userpage.fu-berlin.de/~ram/pub/pub_jf47ht81Ht/doc_kay_oop_en

Concurrency models

- Процессы
- Треды (системные и зеленые)
- Futures and tasks
- Корутины
- CSP (каналы)
- etc.



Проблемы

- Разделяемая память
- Блокировки & синхронизации
- Гонки & отладка

История вопроса

- многоядерные процессоры
- Thread в C, C++, C#, Java
- Apple way

История вопроса

- многоядерные процессоры
- Thread в C, C++, C#, Java
- Apple way

```
actor BankAccount {
    private var balance: Int
    init(initialBalance: Int) {
        balance = initialBalance
    }
}
```

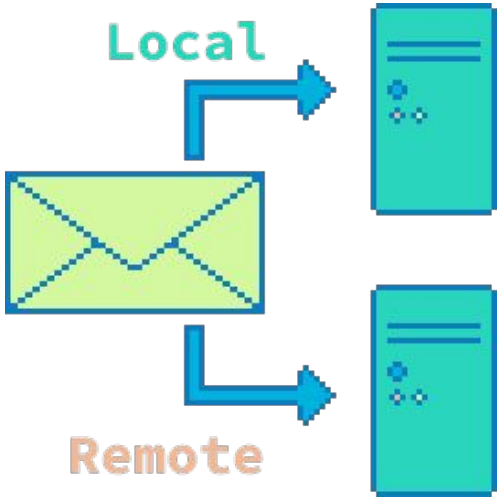
https://github.com/apple/swift-evolution/blob/main/proposals/0306-actors.md

Erlang

- Joe Armstrong и Co разработали в Ericsson в 1986 г.
- Создавался под задачи телекоммуникации (параллельные и распределённые вычисления)
- Оперирует легковесными процессами (Process)
- Процессы изолированы друг от друга, не имеют общего состояния и выполняются параллельно

Работает!

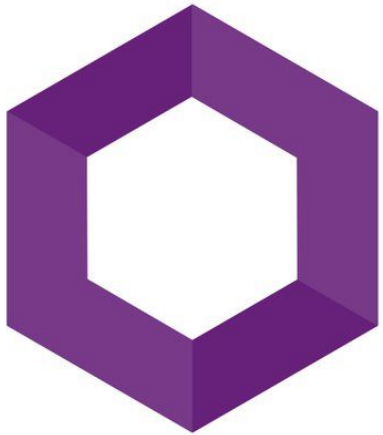
- можно создать миллионы акторов на обычной машине
- за переключение контекста процессов (акторов) отвечает виртуальная машина
- location transparency*
- линейная масштабируемость



Virtual actors

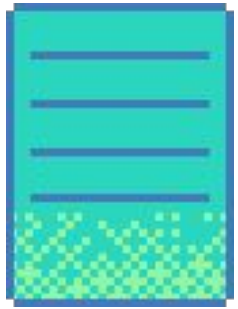
Orleans: Distributed Virtual Actors for Programmability and Scalability

https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/Orleans-MSR-TR-2014-41.pdf



Свойства виртуальных акторов

- Вечное существование
- Автоматическое создание
- Ссылочная прозрачность
- Автоматическое масштабирование
  (single activation & stateless)
- Promise based programming (async-await)



Serverless Computing

Lambda/Functions vs Actors

Serverless Computing

Lambda/Functions vs Actors

Проблемы:

- Оркестрация (уже есть k8s)
- Сетевые задержки
- Resource Overhead

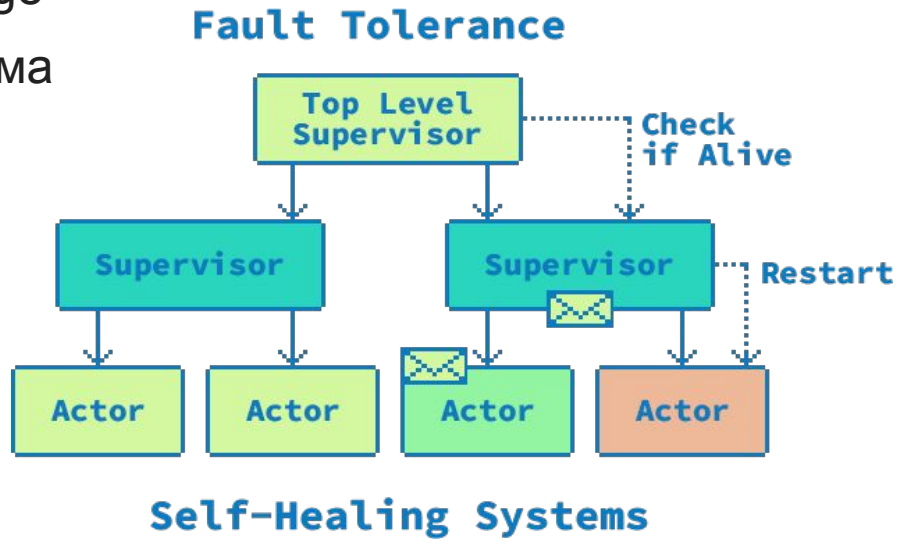
https://www.primevideotech.com/video-streaming/scaling-up-the-prime-video-audio-video-monitoring-service-and-reducing-costs-by-90

Модель отношения акторов

- Иерархическая
- Одноранговая

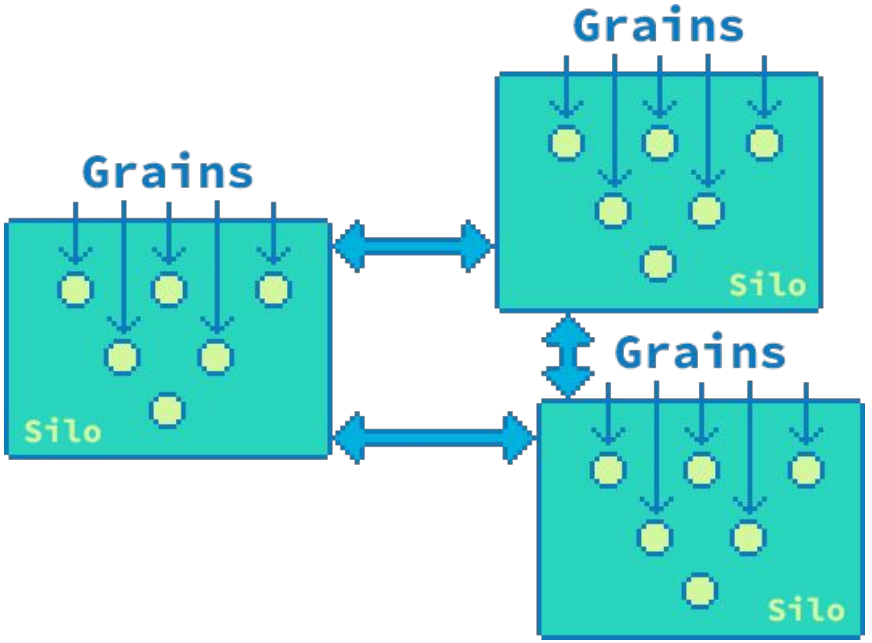
Иерархическая модель (Akka)

- «Родитель» решает, что делать в случае ошибки
- Может отправить PoisonPill message
- Самовосстанавливающаяся система

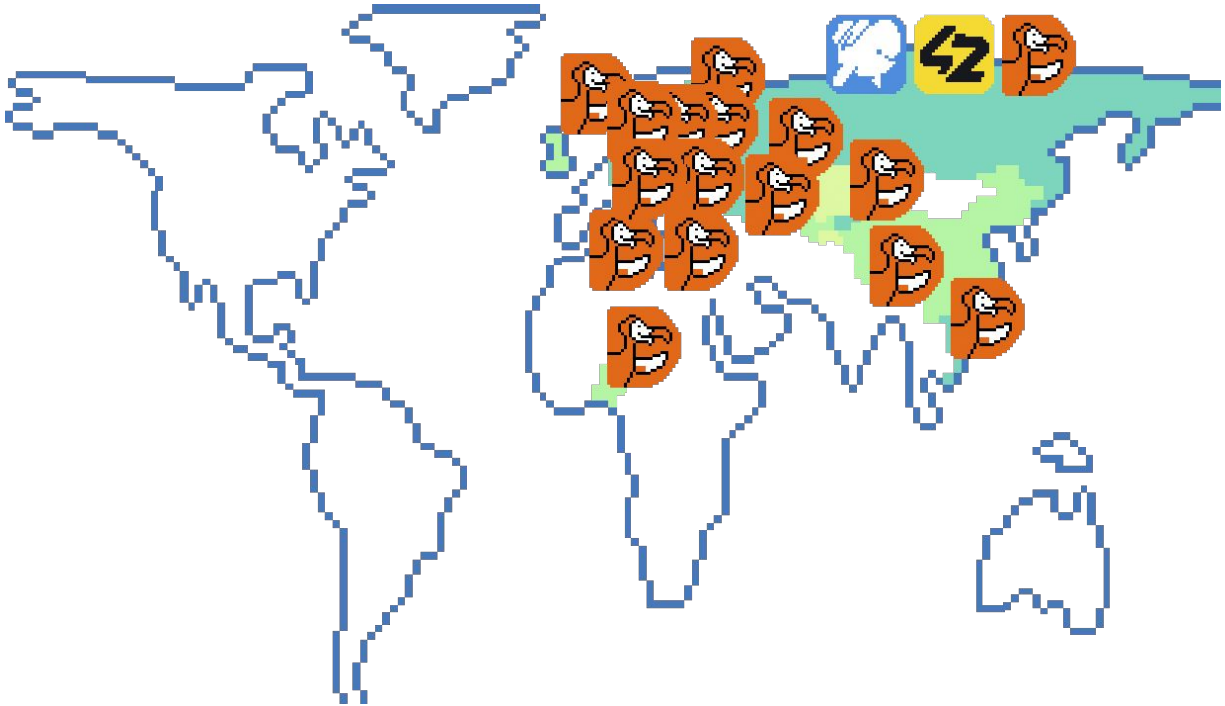


Одноранговая модель (Orleans)

- Виртуальные акторы (Grain)
- Lifetime управляется runtime'ом
- AP-система

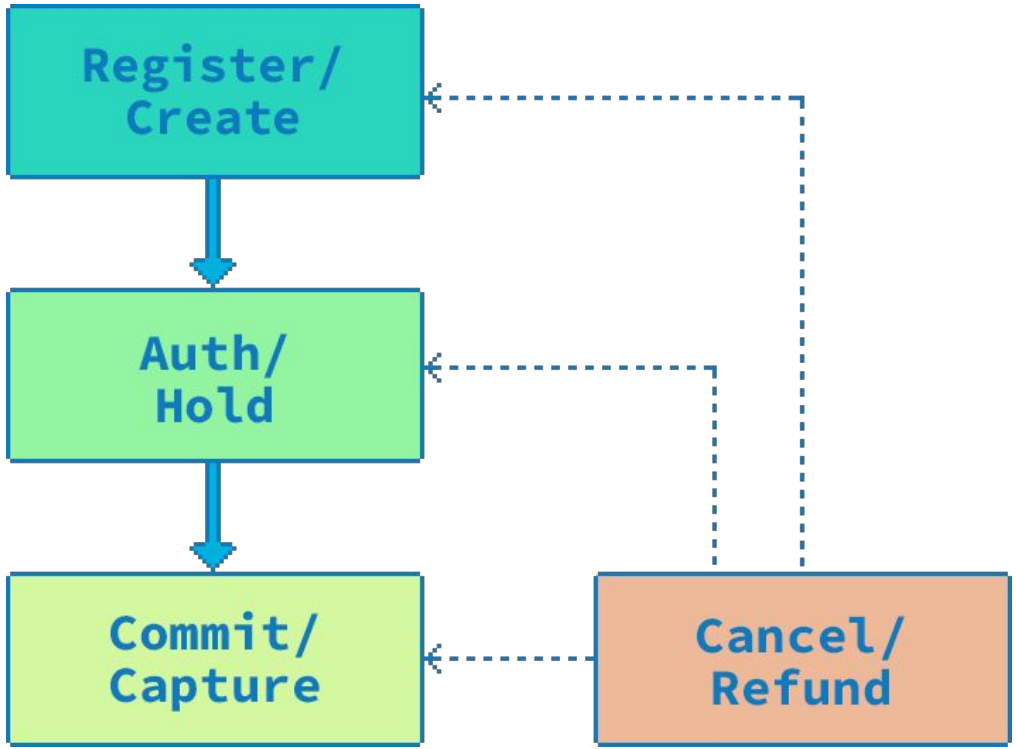


Подход к снаряду



Платеж

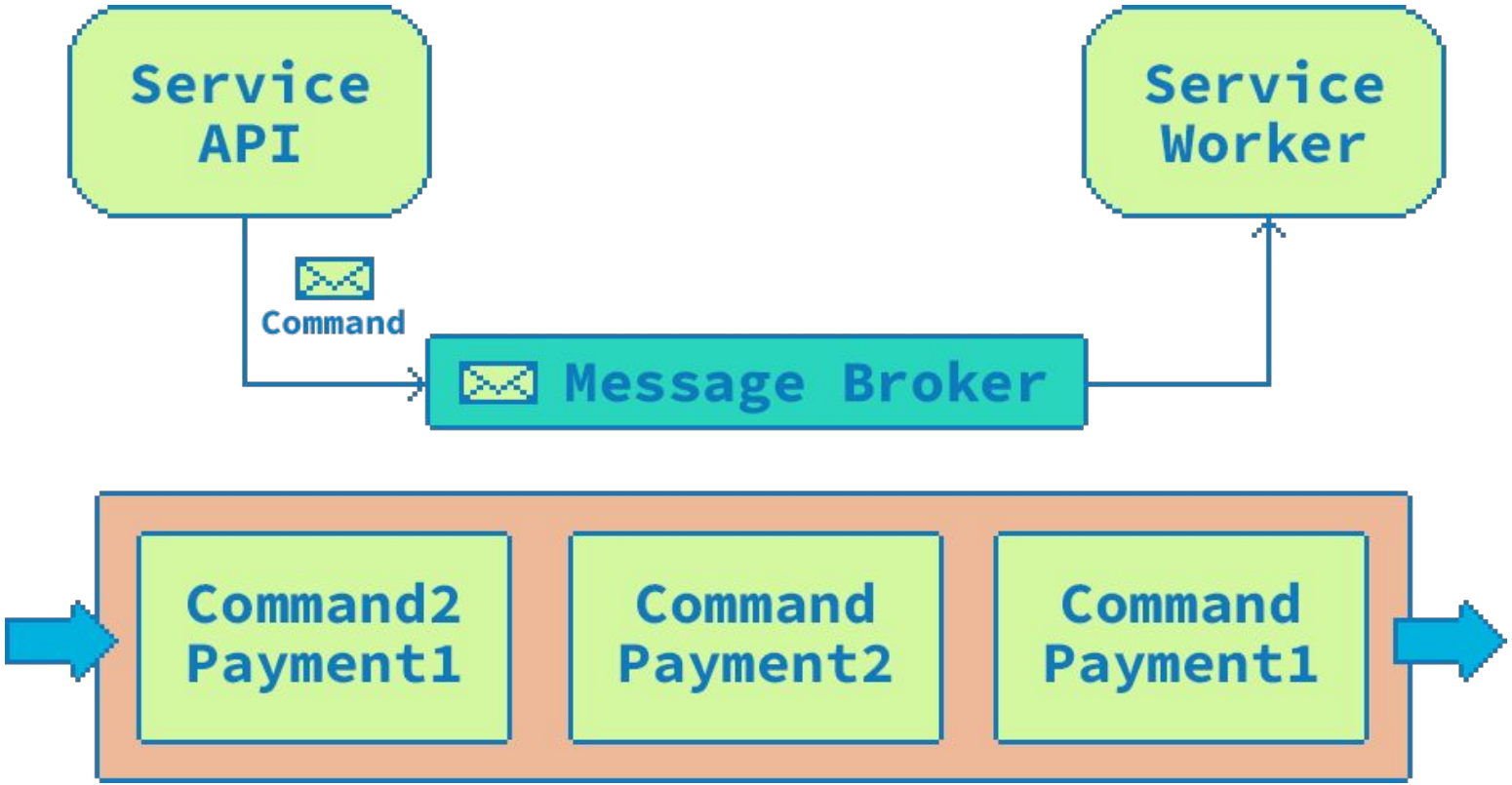
- Зарегистрировать
- Авторизовать
- Подтвердить
- Отменить*



Действия

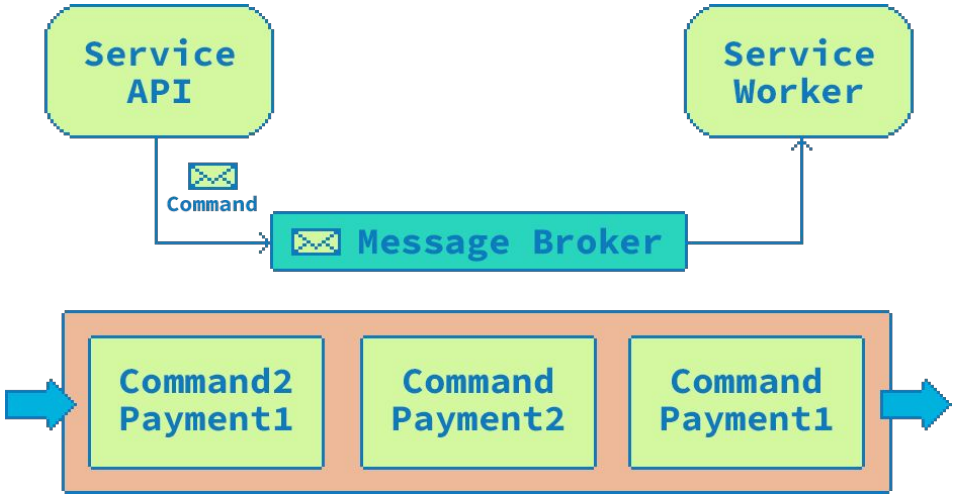
- Команда
- Команды имеют side-эффекты
- Side-эффекты могут быть неидемпотентными
- Результат команды может быть синхронным или асинхронным
- Должны выполняться последовательно

Немного асинхронного

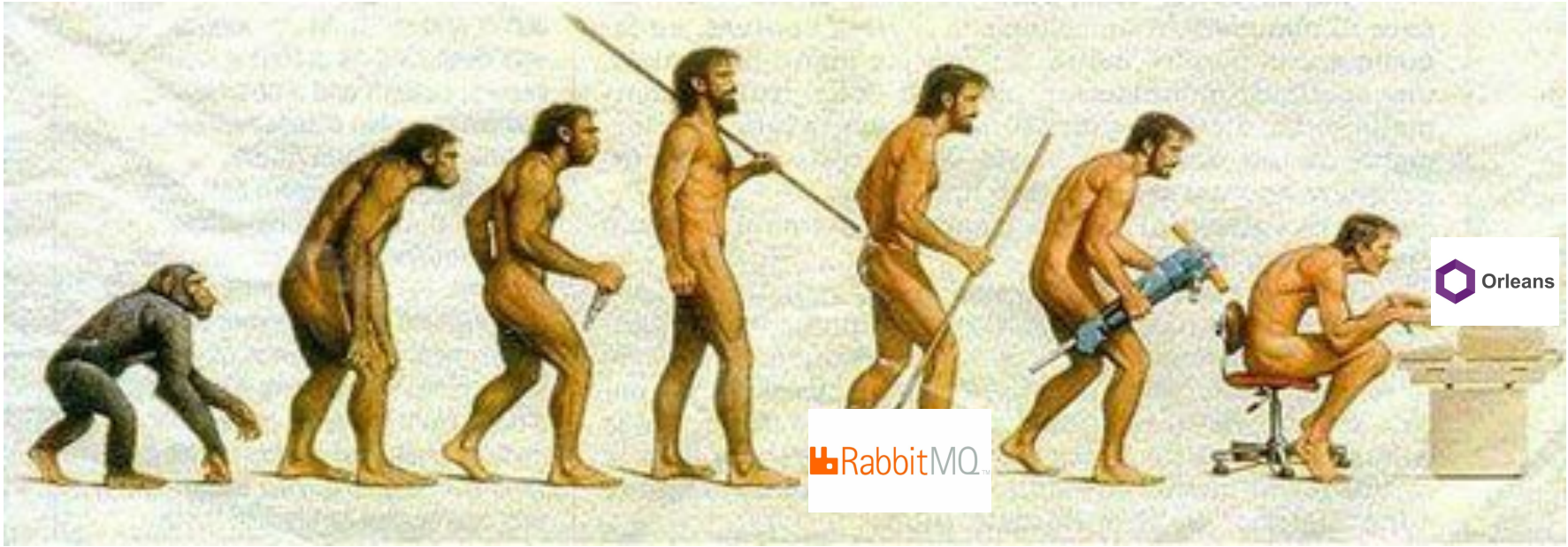


Проблемы

- Не всегда есть синхронный результат (в большинстве случаев)
- Нотификации
- Порядок
- Pessimistic lock
- Head of line blocking



Эволюция



Orleans 3.x → 7

- Dramatic performance improvements
- Package hell

Orleans 3.x → 7

- Dramatic performance improvements
- Package hell
- Grain identities
- Serialization

Orleans 3.x → 7

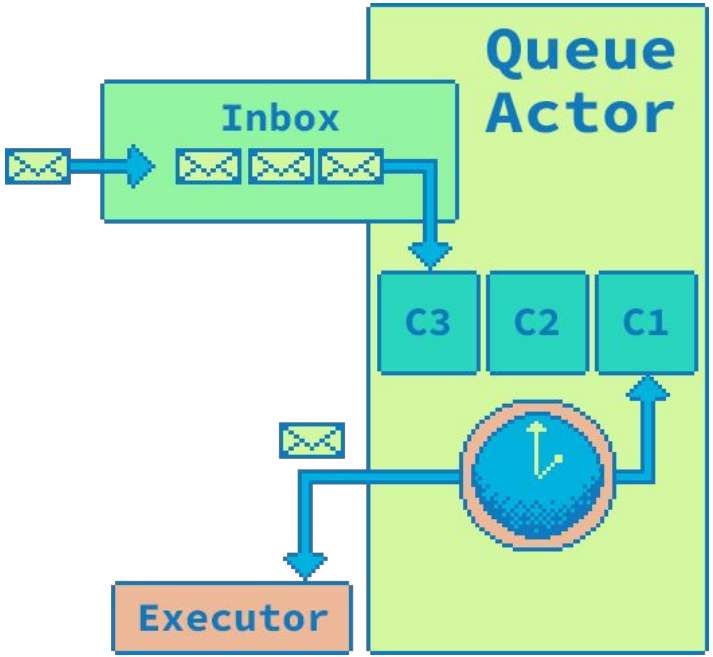
- Dramatic performance improvements
- Package hell
- Grain identities
- Serialization
- OpenTelemetry
- etc.

Orleans 3.x → 7

- Dramatic performance improvements
- Package hell
- Grain identities
- Serialization
- OpenTelemetry
- etc.
- No backward compatibility

Решение

- Очередь — актор
- Актор имеет таймер
- По таймеру проверяет состояние (очереди | исполнения)
- Рантайм пересоздаёт акторы (reminders)



Контракт


```
public interface ICommandsQueue : IGrainWithStringKey, IRemindable, IHaveTimer
{
    public Task Enqueue(ICommand command);

    public Task SetCompleted(string commandId);

    public Task<CommandView> GetCurrentCommand();

    Task Notify(IQueueNotification notification);
}
```

Асинхронное api


```
public interface ICommandsQueue : IGrainWithStringKey, IRemindable, IHaveTimer
{
    public Task Enqueue(ICommand command);

    public Task SetCompleted(string commandId);

    public Task<CommandView> GetCurrentCommand();

    Task Notify(IQueueNotification notification);
}
```

Идентификатор


```
public interface ICommandsQueue : IGrainWithStringKey, IRemindable, IHaveTimer
{
    public Task Enqueue(ICommand command);

    public Task SetCompleted(string commandId);

    public Task<CommandView> GetCurrentCommand();

    Task Notify(IQueueNotification notification);
}
```

Отправка сообщения


```
public static ICommandsQueue GetCommandsQueue(
    this IClusterClient factory,
    string queueId)
     => factory.GetGrain<ICommandsQueue>(queueId);

public async Task Enqueue(string queueId, ICommand command)
    => await _clusterClient.GetCommandsQueue(queueId)
        .Enqueue(command);
```

Запросить ref


```
public static ICommandsQueue GetCommandsQueue(
    this IClusterClient factory,
    string queueId)
     => factory.GetGrain<ICommandsQueue>(queueId);

public async Task Enqueue(string queueId, ICommand command)
    => await _clusterClient.GetCommandsQueue(queueId)
        .Enqueue(command);
```

Отправить собщение


```
public static ICommandsQueue GetCommandsQueue(
    this IClusterClient factory,
    string queueId)
     => factory.GetGrain<ICommandsQueue>(queueId);


public async Task Enqueue(string queueId, ICommand command)
    => await _clusterClient.GetCommandsQueue(queueId)
        .Enqueue(command);
```

Напоминания


```
public interface ICommandsQueue : IGrainWithStringKey, IRemindable, IHaveTimer
{
    public Task Enqueue(ICommand command);

    public Task SetCompleted(string commandId);

    public Task<CommandView> GetCurrentCommand();

    Task Notify(IQueueNotification notification);
}
```

Надо регистрировать


```
public class CommandsQueueGrain : Grain, ICommandsQueue
{
    public override async Task OnActivateAsync(CancellationToken ct)
      => await this.RegisterOrUpdateReminder(
          reminderName: ConstantName,
          dueTime: TimeSpan.FromMinutes(1),
          period: TimeSpan.FromMinutes(1)
      );

   public Task ReceiveReminder(string reminderName, TickStatus status)
      => Task.CompletedTask;
}
```

При активации


```
public class CommandsQueueGrain : Grain, ICommandsQueue
{
    public override async Task OnActivateAsync(CancellationToken ct)
        => await this.RegisterOrUpdateReminder(
            reminderName: ConstantName,
            dueTime: TimeSpan.FromMinutes(1),
            period: TimeSpan.FromMinutes(1)
        );

    public Task ReceiveReminder(string reminderName, TickStatus status)
        => Task.CompletedTask;
}
```

Create or update


```
public class CommandsQueueGrain : Grain, ICommandsQueue
{
    public override async Task OnActivateAsync(CancellationToken ct)
        => await this.RegisterOrUpdateReminder(
            reminderName: ConstantName,
            dueTime: TimeSpan.FromMinutes(1),
            period: TimeSpan.FromMinutes(1)
        );

    public Task ReceiveReminder(string reminderName, TickStatus status)
        => Task.CompletedTask;
}
```

Надо получить


```
public class CommandsQueueGrain : Grain, ICommandsQueue
{
    public override async Task OnActivateAsync(CancellationToken ct)
        => await this.RegisterOrUpdateReminder(
            reminderName: ConstantName,
            dueTime: TimeSpan.FromMinutes(1),
            period: TimeSpan.FromMinutes(1)
        );

    public Task ReceiveReminder(string reminderName, TickStatus status)
        => Task.CompletedTask;
}
```

Timer


```
public interface ICommandsQueue : IGrainWithStringKey, IRemindable, IHaveTimer
{
    public Task Enqueue(ICommand command);

    public Task SetCompleted(string commandId);

    public Task<CommandView> GetCurrentCommand();

    Task Notify(IQueueNotification notification);
}
```

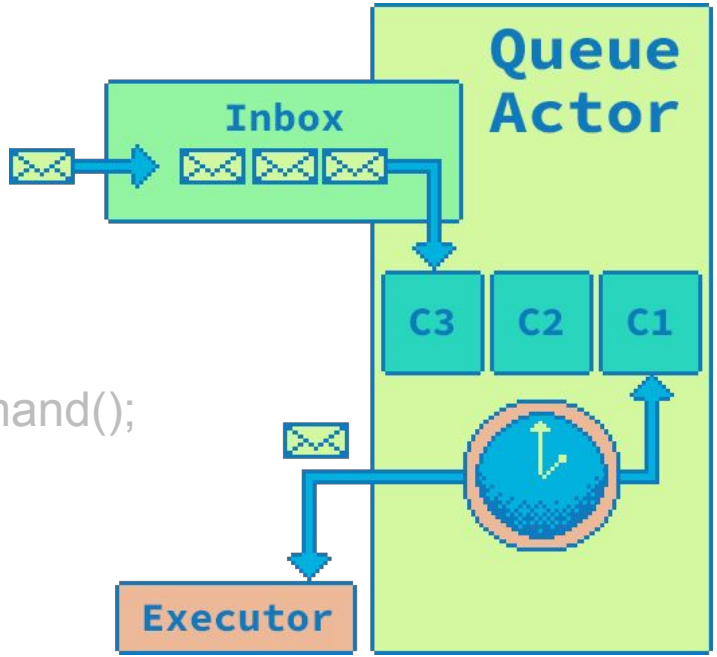
Когда запускать?


```
public interface ICommandsQueue : IGrainWithStringKey, IRemindable, IHaveTimer
{
    public Task Enqueue(ICommand command);

    public Task SetCompleted(string commandId);

    public Task<CommandView> GetCurrentCommand();

    Task Notify(IQueueNotification notification);
}
```



Зарегистрировать


```
public override async Task OnActivateAsync(CancellationToken ct)
{
    _timer = RegisterTimer(
        asyncCallback: GrainTimerUtils.OnTimerTickCallback<ICommandsQueue>,
        state: this,
        dueTime: TimeSpan.FromMilliseconds(FirstTimerTickDelayMs),
        period: TimeSpan.FromMilliseconds(TimerTickPeriodMs)
    );
}
```

Ticks


```
public override async Task OnActivateAsync(CancellationToken ct)
{
    _timer = RegisterTimer(
        asyncCallback: GrainTimerUtils.OnTimerTickCallback<ICommandsQueue>,
        state: this,
        dueTime: TimeSpan.FromMilliseconds(FirstTimerTickDelayMs),
        period: TimeSpan.FromMilliseconds(TimerTickPeriodMs)
    );
}
```

Callback

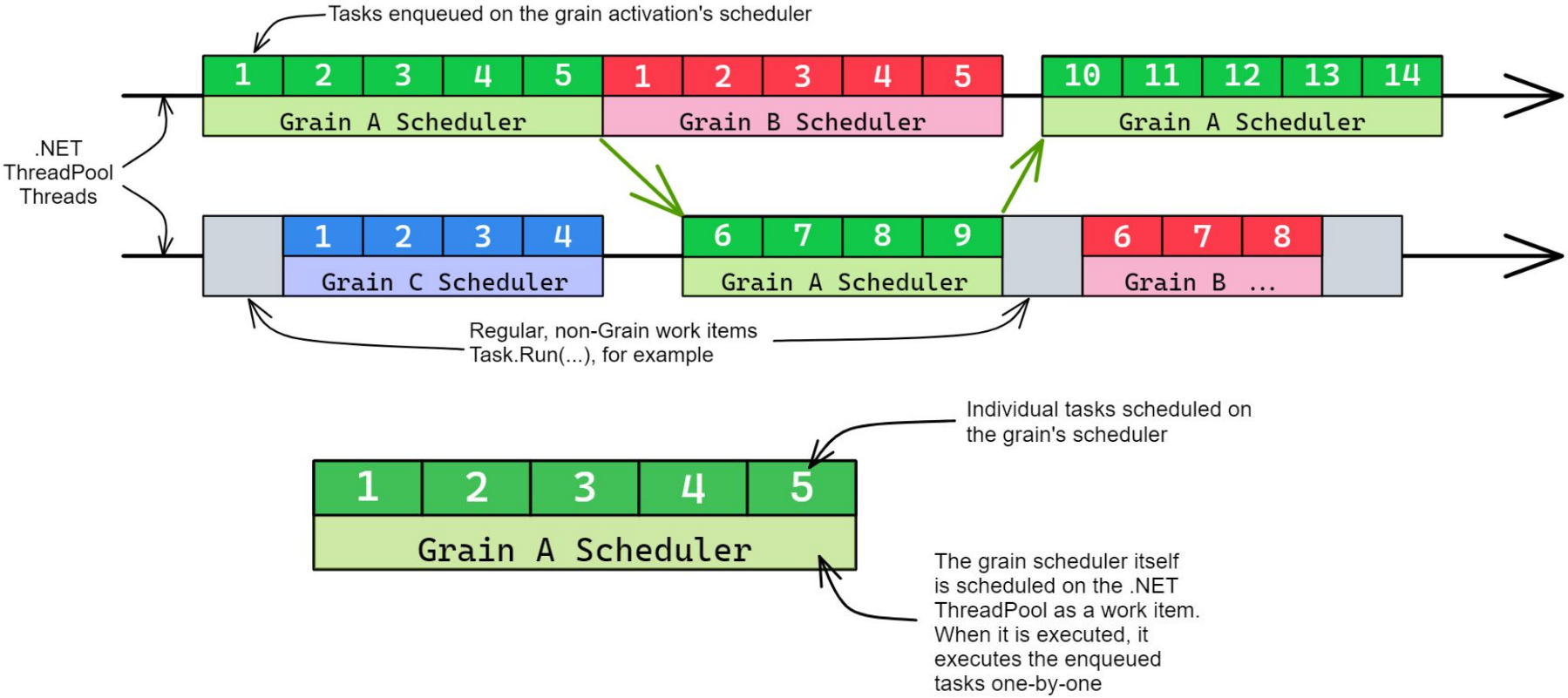

```
public override async Task OnActivateAsync(CancellationToken ct)
{
    _timer = RegisterTimer(
        asyncCallback: GrainTimerUtils.OnTimerTickCallback<ICommandsQueue>,
        state: this,
        dueTime: TimeSpan.FromMilliseconds(FirstTimerTickDelayMs),
        period: TimeSpan.FromMilliseconds(TimerTickPeriodMs)
    );
}
```

Terms


```
public async Task Enqueue(ICommand command)
{
    await _commandsRepository.Add(_queueId, command);       // term
    _commandsQueue.Enqueue(command);
}

public async Task OnTimerTick()
{
    await RefreshCommandExecutionState();                   // term
    await CheckIdle();                                      // term
}
```

ActivationTaskScheduler & WorkItemGroup



No "Atomicity"


```
public async Task Enqueue(ICommand command)
{
    await _commandsRepository.Add(_queueId, command);      // 1
    _commandsQueue.Enqueue(command);                       // ?
}

public async Task OnTimerTick()
{
    await RefreshCommandExecutionState();                  // 2
    await CheckIdle();                                     // ?
}
```

Hack


```
public static Task OnTimerTickCallback<TGrain>(object grain)
    where TGrain : IGrain, IHaveTimer
    => ((TGrain) grain).AsReference<TGrain>().OnTimerTick();
```

Message 1

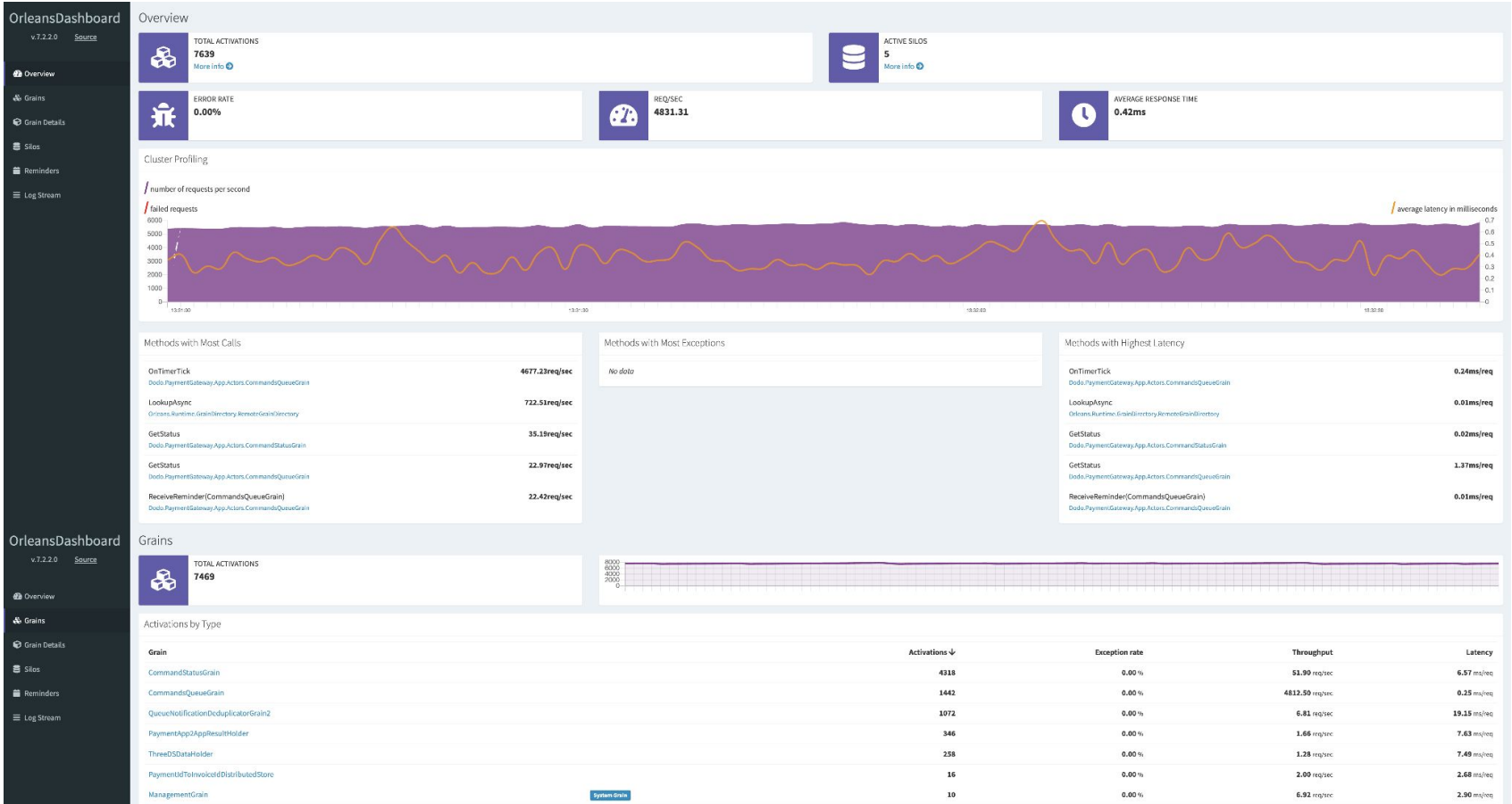

```
public static Task OnTimerTickCallback<TGrain>(object grain)
    where TGrain : IGrain, IHaveTimer
    => ((TGrain) grain).AsReference<TGrain>().OnTimerTick();


public async Task Enqueue(ICommand command)
{
    await _commandsRepository.Add(_queueId, command);       // 1
    _commandsQueue.Enqueue(command);                        // 1
}
```

Message 2

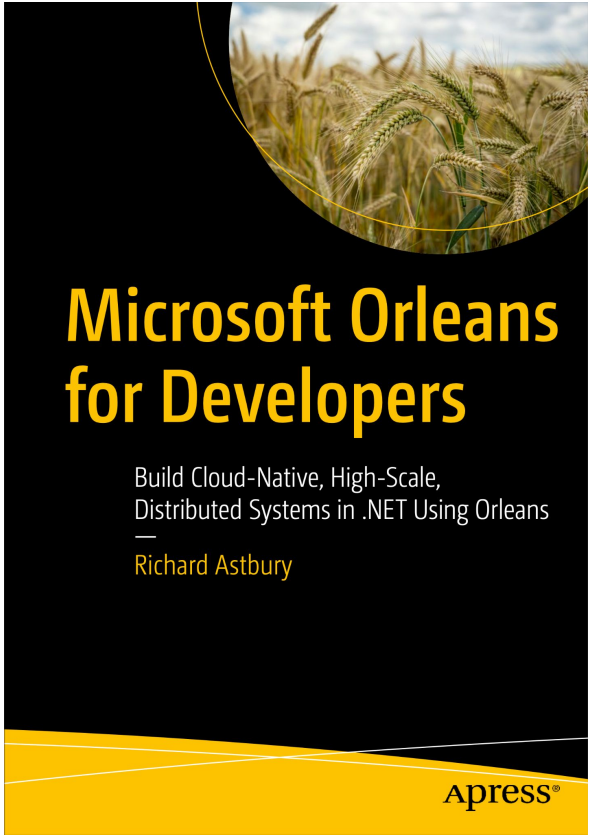

```
public async Task Enqueue(ICommand command)
{
    await _commandsRepository.Add(_queueId, command);       // 1
    _commandsQueue.Enqueue(command);                        // 1
}
public async Task OnTimerTick()
{
    await RefreshCommandExecutionState();                   // 2
    await CheckIdle();                                      // 2
}
```

Dashboard


OrleansDashboard
v.7.2.2.0 Source
Overview
Grains
Grain Details
Silos
Reminders
Log Stream
Overview
TOTAL ACTIVATIONS
7639
More info
ACTIVE SILOS
5
More info
ERROR RATE
0.00%
REQ/SEC
4831.31
AVERAGE RESPONSE TIME
0.42ms
Cluster Profiling
number of requests per second
failed requests
average latency in milliseconds
Methods with Most Calls
OnTimerTick 4677.23req/sec
LookupAsync 722.51req/sec
GetStatus 35.19req/sec
GetStatus 22.97req/sec
ReceiveReminder(CommandsQueueGrain) 22.42req/sec
Methods with Most Exceptions
No data
Methods with Highest Latency
OnTimerTick 0.24ms/req
LookupAsync 0.01ms/req
GetStatus 0.02ms/req
GetStatus 1.37ms/req
ReceiveReminder(CommandsQueueGrain) 0.01ms/req
Grains
TOTAL ACTIVATIONS
7469
Activations by Type
Grain
Activations
Exception rate
Throughput
Latency
CommandStatusGrain 4318 0.00 % 51.90 req/sec 6.57 ms/req
CommandsQueueGrain 1442 0.00 % 4812.50 req/sec 0.25 ms/req
QueueNotificationDeduplicatorGrain2 1072 0.00 % 6.81 req/sec 19.15 ms/req
PaymentApp2AppResultHolder 346 0.00 % 1.66 req/sec 7.63 ms/req
ThreeDSDataHolder 258 0.00 % 1.28 req/sec 7.49 ms/req
PaymentIdToInvoiceIdDistributedStore 16 0.00 % 2.00 req/sec 2.68 ms/req
ManagementGrain System Grain 10 0.00 % 6.92 req/sec 2.90 ms/req

One more thing

- Reentrancy and read only
- Immutable
- Event sourcing
- Grains persistence
- Grains placement
- Grains service
- Stateless workers
- Streams and observers
- Transactions
- Interface versioning
- etc.



Anti Use-Cases

- Non-concurrent system and|or no mutable state
- Performance critical app
- "Actor-based" design

О чем не забыть

- Observability (логи, метрики, трейсы)
- Overload/Deploy behavior
  - ресурсы ограничены
  - mailbox не бесконечный
  - восстановление состояния
  - балансировка нагрузки aka стратегия создания акторов
  - дедлоки (local & distributed)

О чем я не рассказал

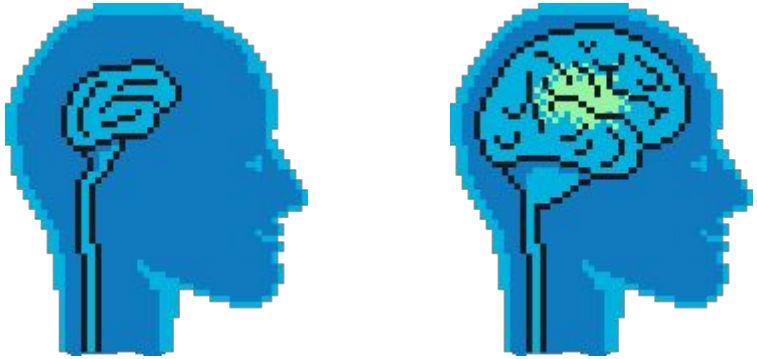
- Address != identity
- Distributed algos *under the hood*
- Use-Cases

Выводы

- Простая абстракция

Выводы

- Простая абстракция
- Абстракция для упрощения кода



Выводы

- Простая абстракция
- Абстракция для упрощения кода
- Абстракция для упрощения масштабирования

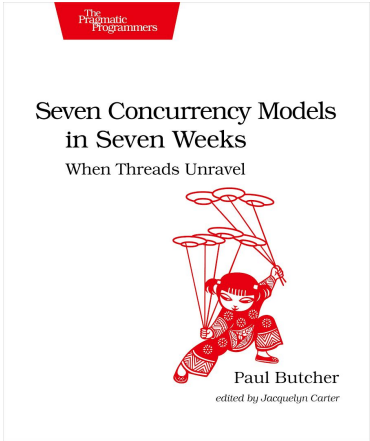
Выводы

- Простая абстракция
- Абстракция для упрощения кода
- Абстракция для упрощения масштабирования
- Актуальность …




Что посмотреть

- Hewitt, Meijer and Szyperski: The Actor Model (everything you wanted to know...)
  https://www.youtube.com/watch?v=7erJ1DV_Tlo
- Book "Seven Concurrency Models in Seven Weeks"
- Cloud Design Patterns
  https://learn.microsoft.com/en-us/azure/architecture/patterns



На что посмотреть

- Akka (Java & .NET)
- Orleans (.NET)
- Proto.Actor (.NET & Go)
- Dapr (много)

Давайте обсудим!