Как собрать всю базу в файл и зачем тут akka streams


zhitkon@gmail.com


@cdcvmm

Компании - разработчики

2013

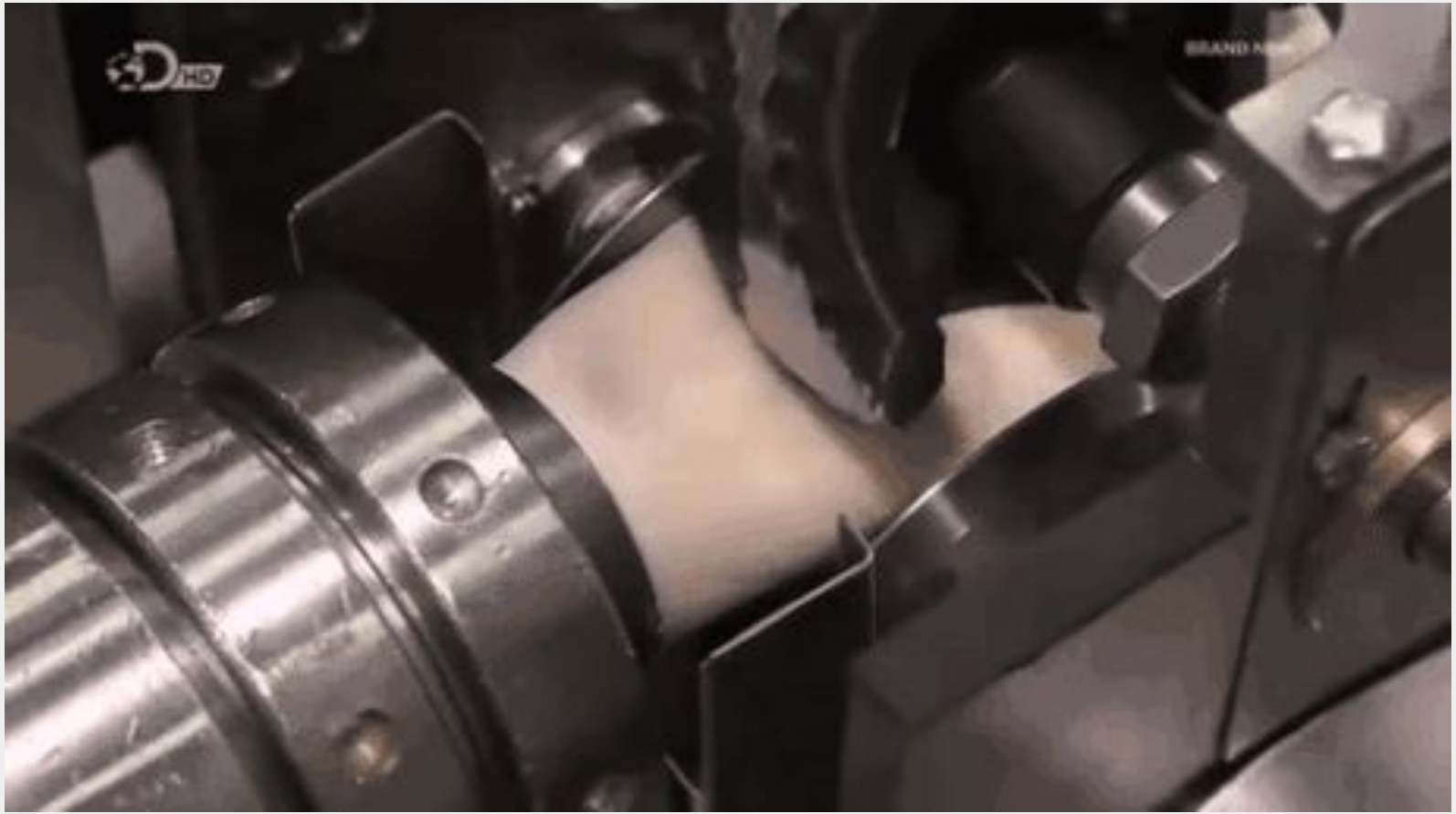
2015

2017

Традиционный подход

- Данные должны быть собраны
- Ожидаем выполнение запросов
- Анализируем постфактум

Пример преобразования потока данных

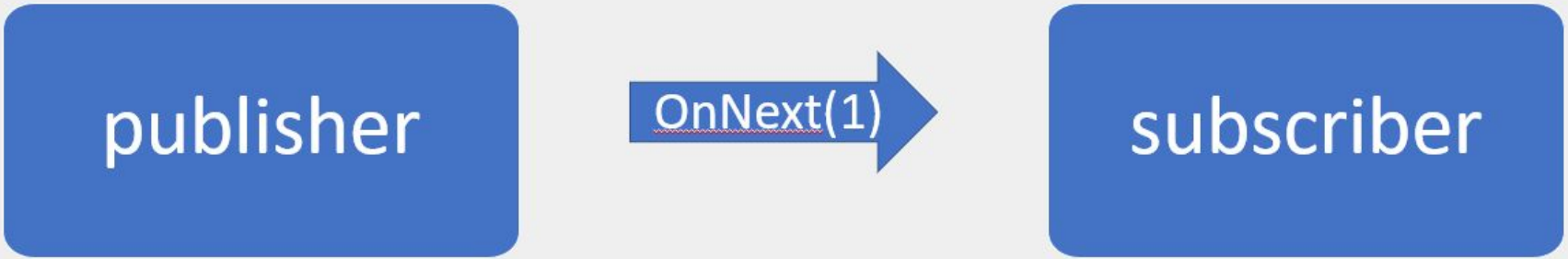


gif-изображение получено из открытых источников

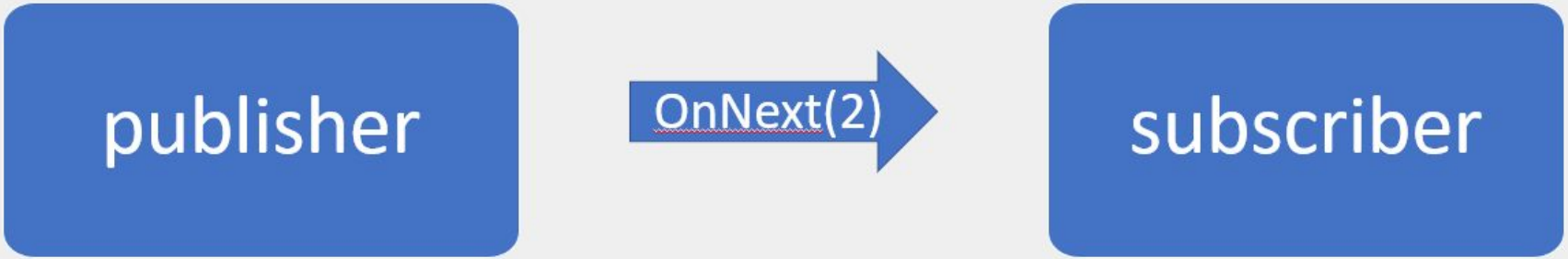
Прохождение данных в reactive streams

Back-pressure в reactive streams

Back-pressure в reactive streams



Back-pressure в reactive streams

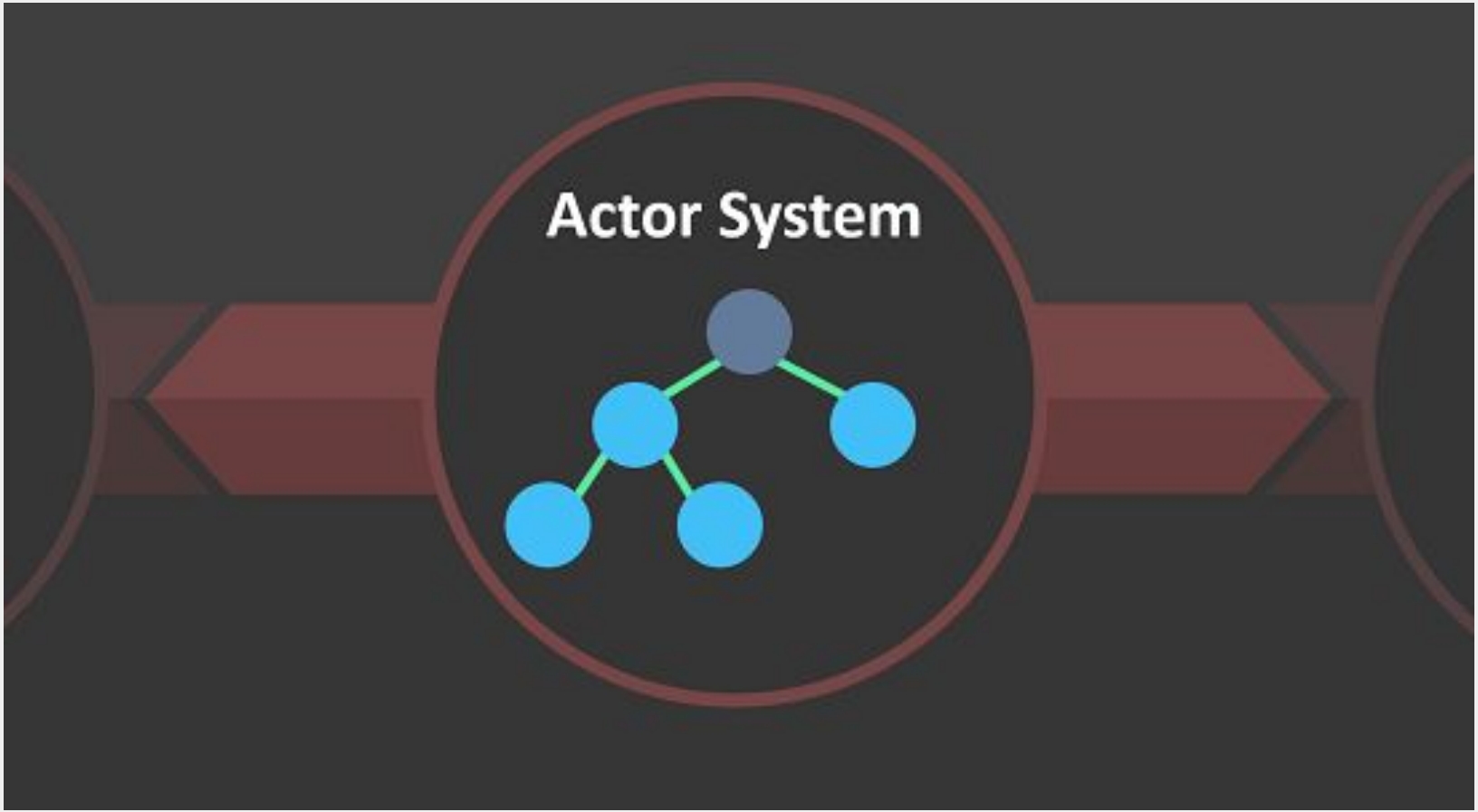


Back-pressure в reactive streams

Реактивное программирование

- Обрабатываем данные в момент их получения
- Нет ожиданий
- Работаем с данными в удобном темпе

Акторная модель

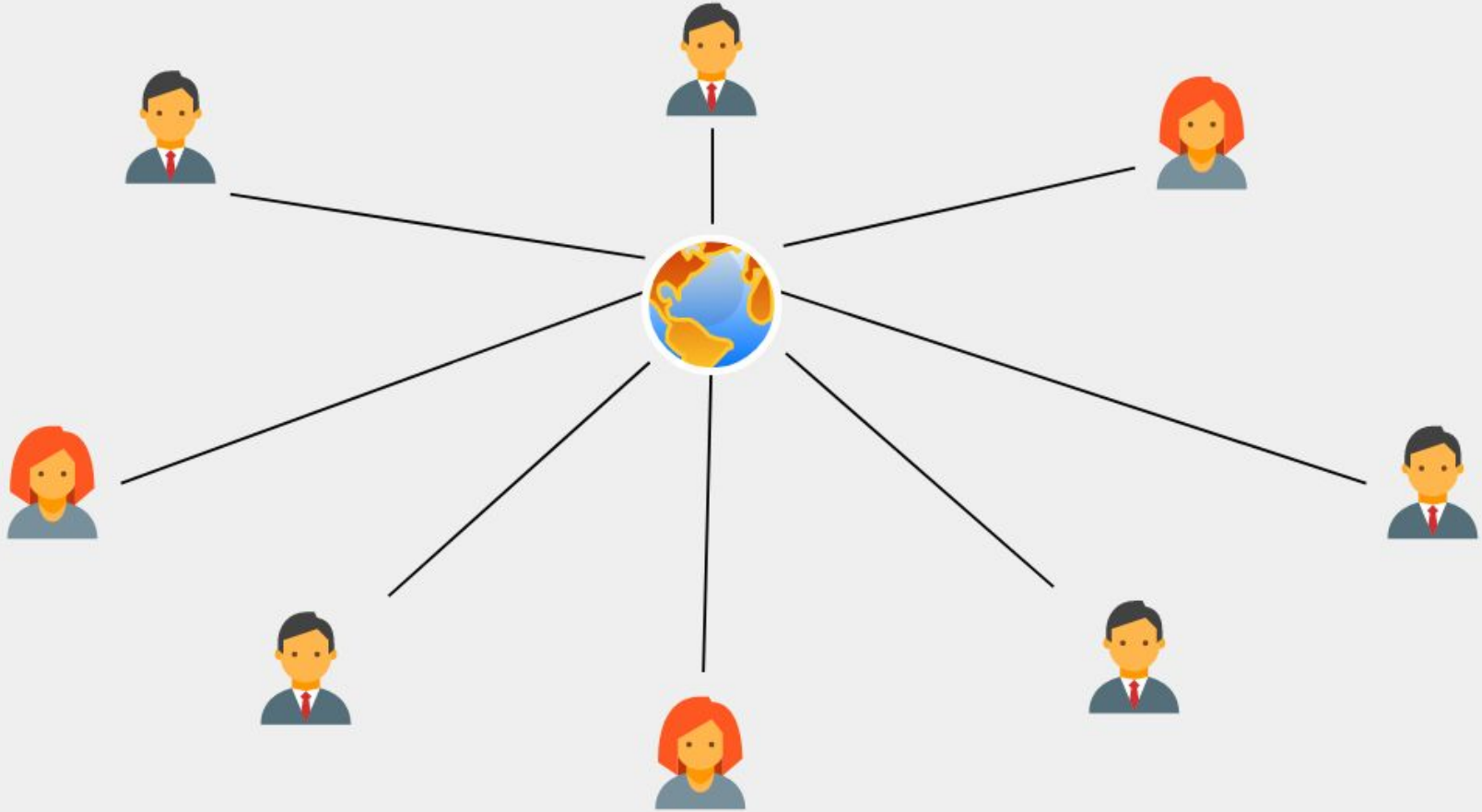


Акторная модель

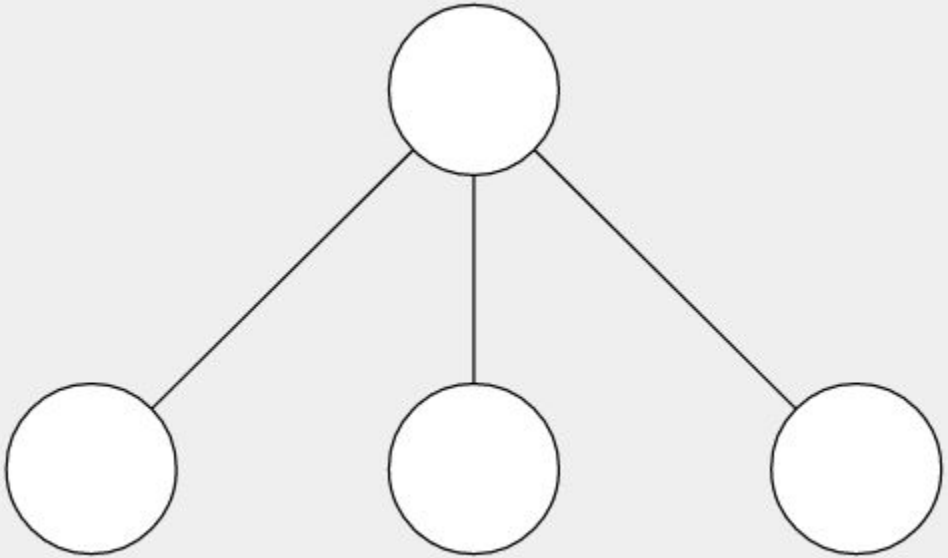
Сообщения

- Отправляет новое сообщение
- Создает дочерний актор
- Вызывает сторонние сервисы
- Изменяет внутреннее состояние

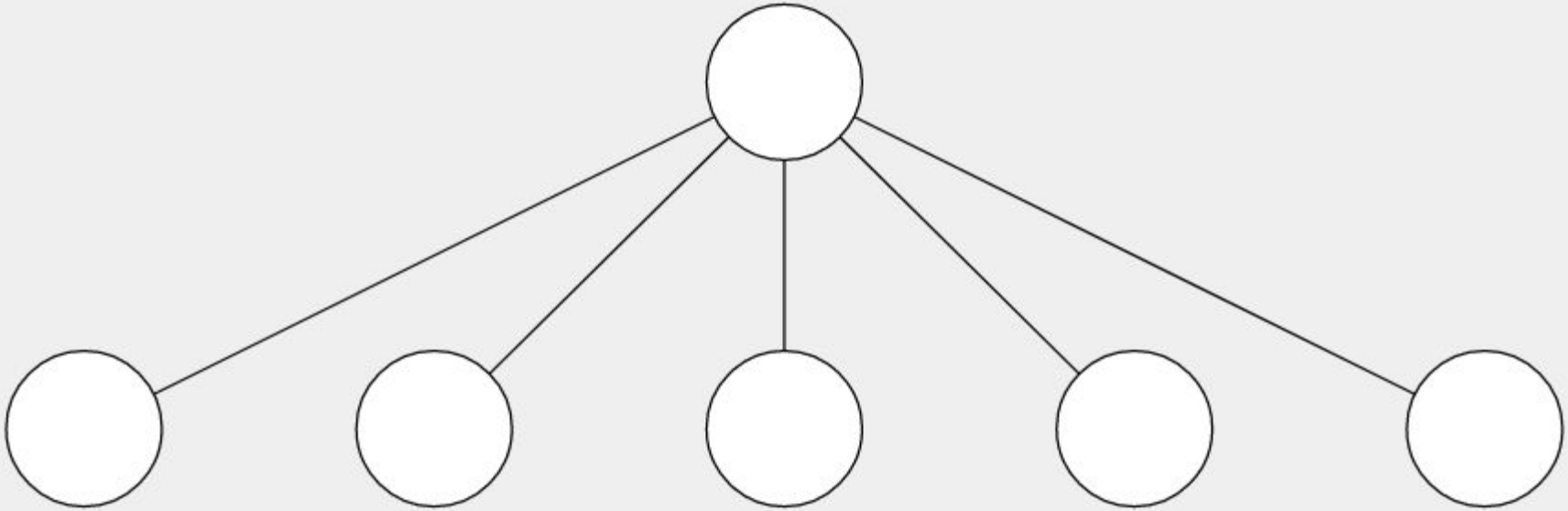
Акторная модель



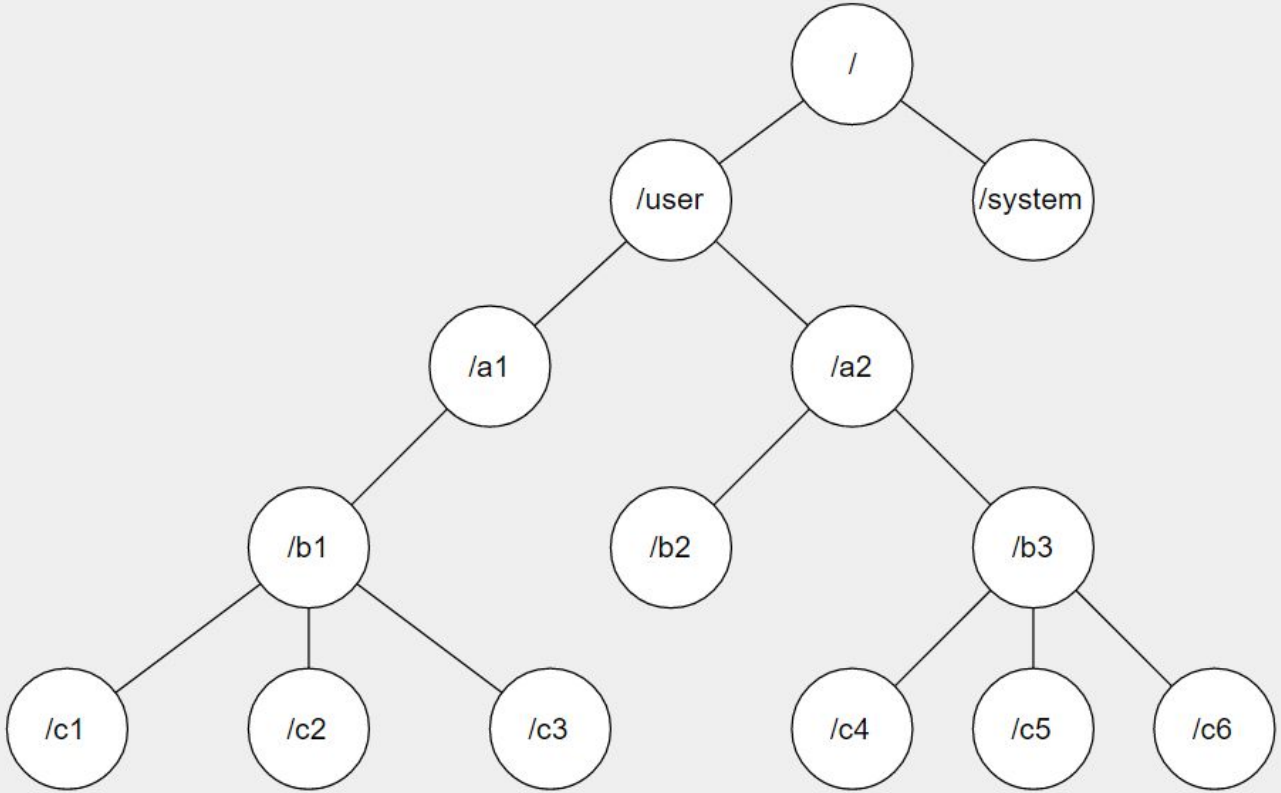
Масштабирование акторов



Масштабирование акторов



Иерархия акторов



DOTNEXT

Moscow 2017

Вагиф Абилов

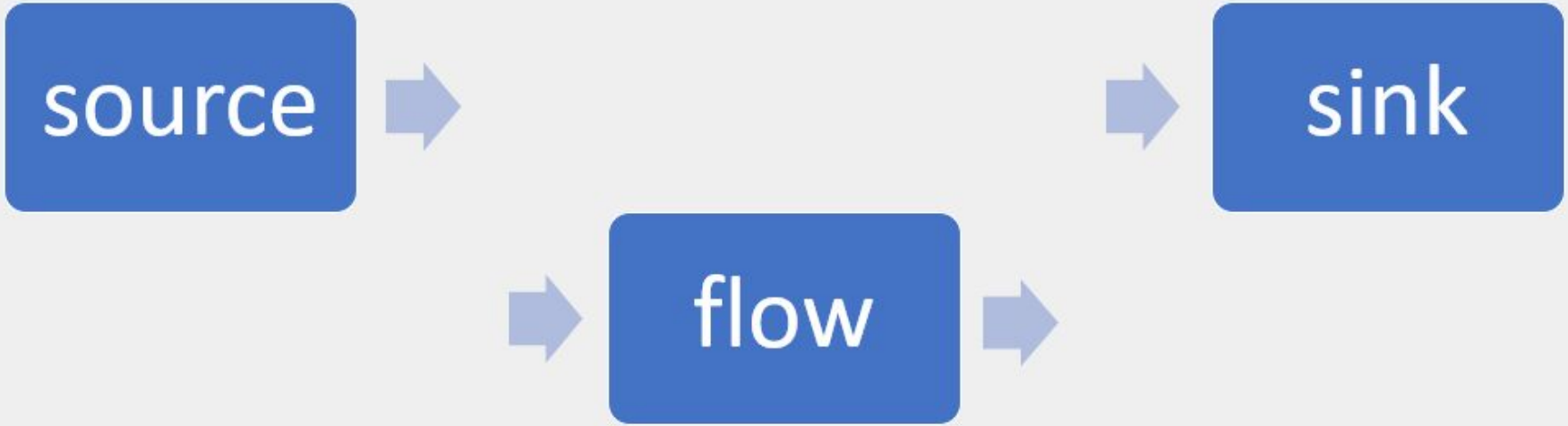
Miles

Akka Streams для простых смертных

Прохождение данных в reactive streams

Akka.net streams

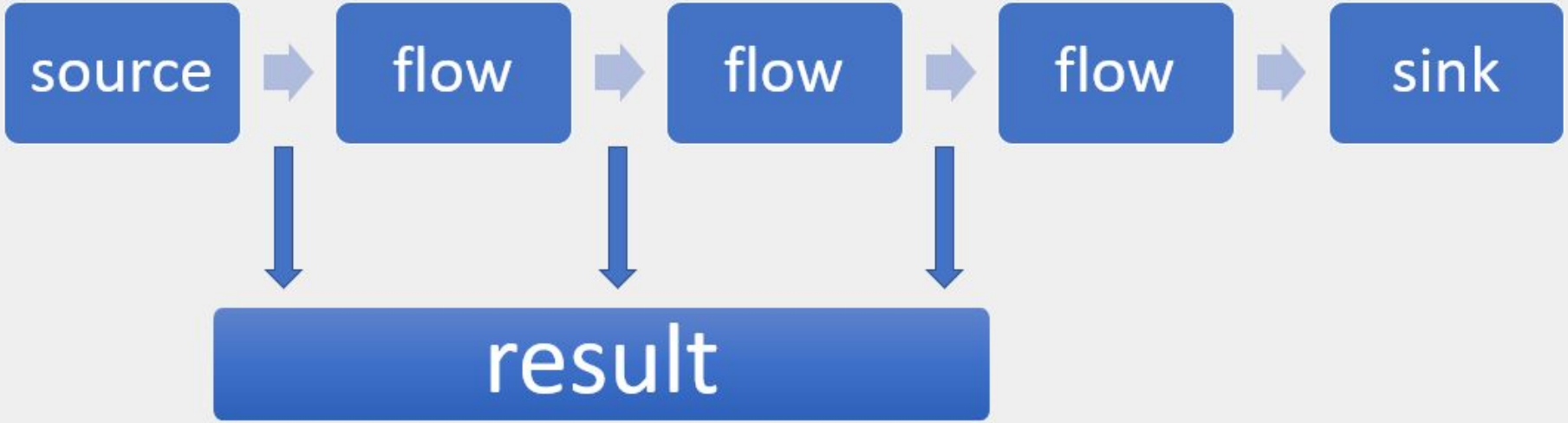
Основные блоки построения



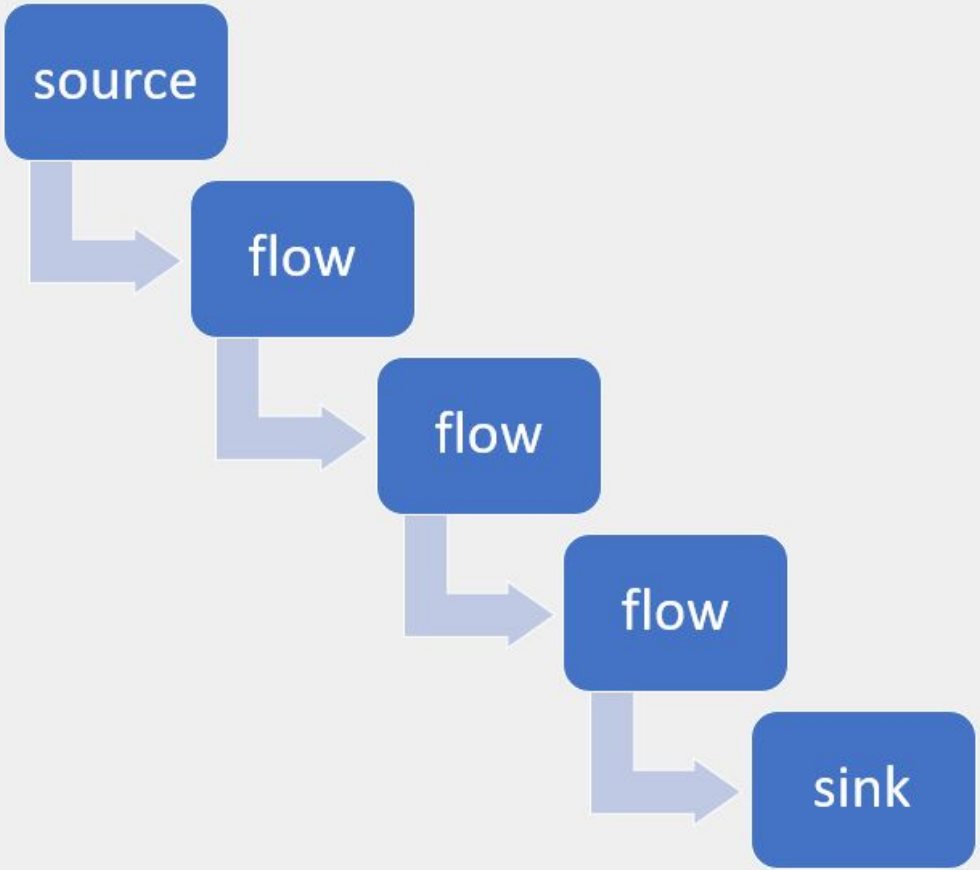
Пример построения графа

```
var source = new List<int> {1, 4, 10};
var graph =
    Source.From(source)
        .Select(x => x + 2)
        .Select(x => x % 2 == 0)
        .Select(x => $"Test {x}")
        .To(Sink.LastOrDefault<string>());
```

Материализация значений



Построение пайплайна

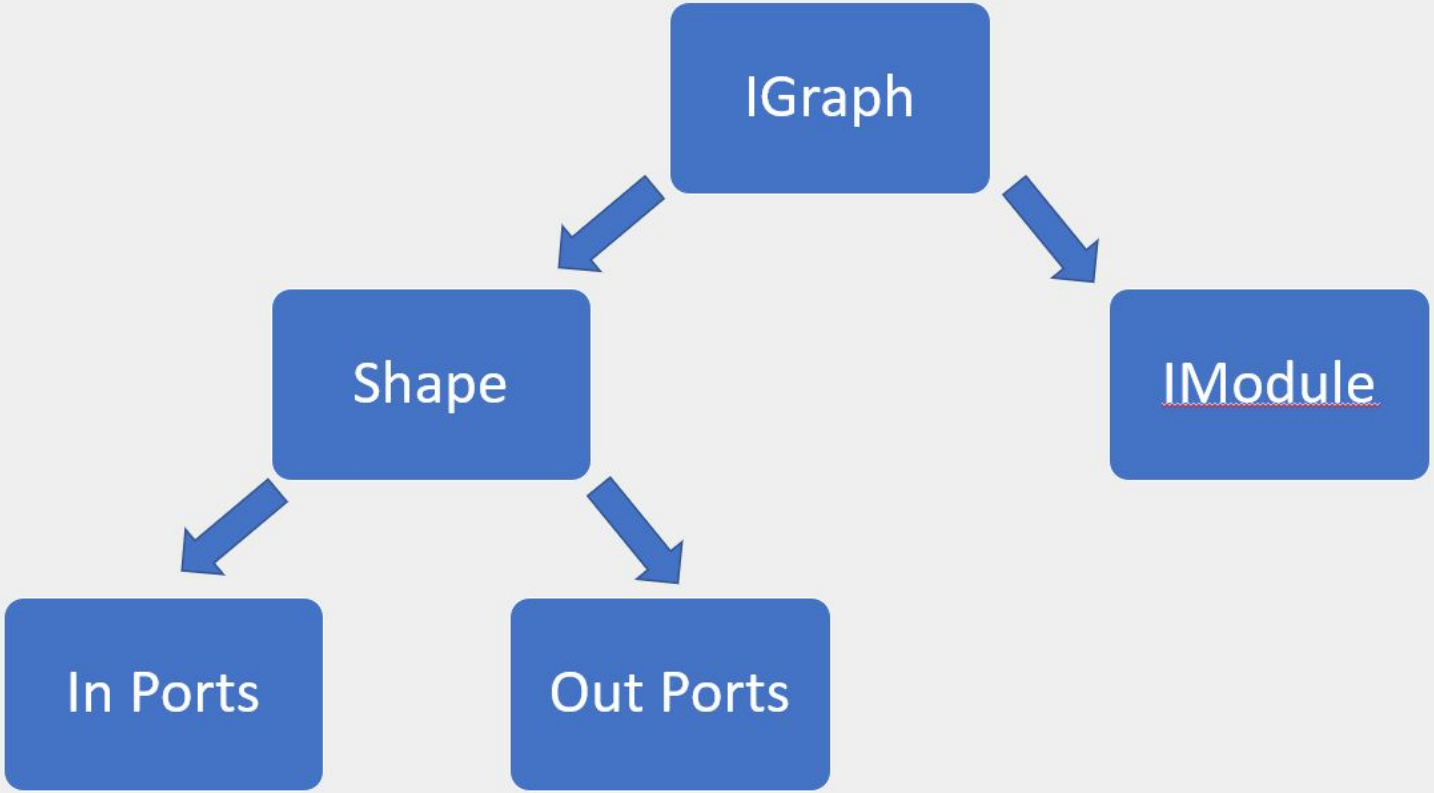



```
let seqTest n =
  Seq.init (n + 1) id
  |> Seq.map    int64
  |> Seq.map    ((+) 1L)
  |> Seq.sum
```

Построение пайплайна через GraphDSL

```
IGraph<ClosedShape, NotUsed> graph
    = GraphDsl.Create( buildBlock: builder =>
    {
        var broadcast = builder.Add(new Broadcast<int>( outputPorts: 2));
        var merge = builder.Add(new Merge<string>( inputPorts: 2));

        builder.From(Source.Empty<int>()).To(broadcast.In);

        builder.From( outlet: broadcast.Out( n: 0))
            .Via(Flow.Create<int>().Select(x => x.ToString()))
            .To( inlet: merge.In( n: 0));
        builder.From( outlet: broadcast.Out( n: 1))
            .Via(Flow.Create<int>().Select(x => x * 2))
            .Via(Flow.Create<int>().Select(x => x.ToString()))
            .To( inlet: merge.In( n: 1));

        builder.From(merge.Out).To(Sink.ForEach<string>(Console.WriteLine));

        return ClosedShape.Instance;
    });
```

Структура графа



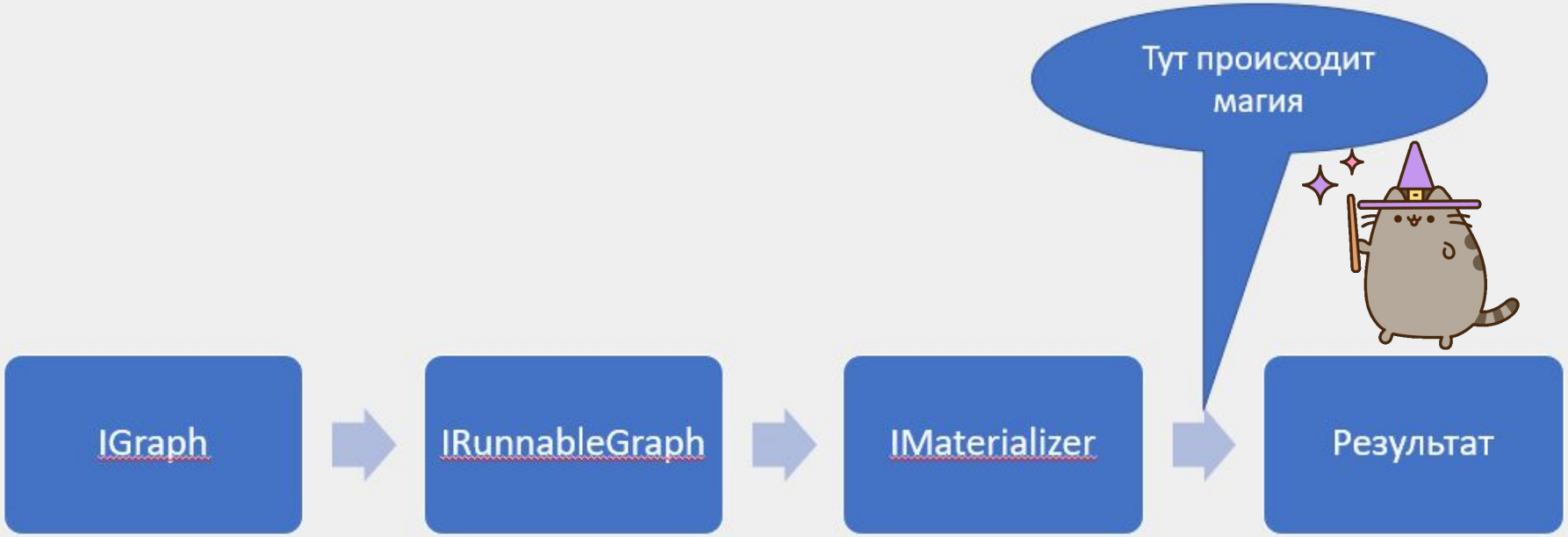
Построение пайплайна через GraphDSL

```
IGraph<ClosedShape, NotUsed> graph
    = GraphDsl.Create( buildBlock: builder =>
    {
        var broadcast = builder.Add(new Broadcast<int>( outputPorts: 2));
        var merge = builder.Add(new Merge<string>( inputPorts: 2));

        builder.From(Source.Empty<int>()).To(broadcast.In);

        builder.From( outlet: broadcast.Out( n: 0))
            .Via(Flow.Create<int>().Select(x => x.ToString()))
            .To( inlet: merge.In( n: 0));
        builder.From( outlet: broadcast.Out( n: 1))
            .Via(Flow.Create<int>().Select(x => x * 2))
            .Via(Flow.Create<int>().Select(x => x.ToString()))
            .To( inlet: merge.In( n: 1));

        builder.From(merge.Out).To(Sink.ForEach<string>(Console.WriteLine));

        return ClosedShape.Instance;
    });
```

Построение пайплайна через GraphDSL

```
var source = Source.Empty<int>();
var sink = Sink.Ignore<string>();
IGraph<ClosedShape, (NotUsed, Task)> graph
    = GraphDsl.Create(
        source, sink,
        combineMaterializers: (s, d) => (s, d),
        buildBlock: (builder, src, dst) =>
    {
        var broadcast = builder.Add(new Broadcast<int>( outputPorts: 2));
        var merge = builder.Add(new Merge<string>( inputPorts: 2));

        builder.From(src).To(broadcast.In);

        builder.From( outlet: broadcast.Out( n: 0)).Via(Flow.Create<int>().Select(x => x.ToString()))
            .To( inlet: merge.In( n: 0));
        builder.From( outlet: broadcast.Out( n: 1)).Via(Flow.Create<int>().Select(x => x * 2))
            .Via(Flow.Create<int>().Select(x => x.ToString()))
            .To( inlet: merge.In( n: 1));

        builder.From(merge.Out).To(dst);

        return ClosedShape.Instance;
    });
```

Работа графа



Виды этапов

Источники

- FromEnumerator
- Single
- Repeat
- Cycle
- Tick
- FromTask
- Empty
- Maybe
- Failed
- Lazily
- ActorPublisher
- ActorRefUnfoldResource
- Queue

Flow

- Select
  SelectAsync
- SelectAsyncUnordered
- GroupedWithin
- Balance
- Merge
- Log
- …. ….
- ….
- ….
- ….
- ….
- их очень много

Стоки

- First
- FirstOrDefault
- Last
- LasrOrDefault
- Ignore
- Cancelled
- Seq
- Foreach
- ForeachParallel
- OnComplete
- AggregateActorRef
- ActorRefWithAck
- ActorSubscriber

Посмотрим, что внутри

Фото взято из открытых источников

TBD. TBD everywhere

```
/// <summary>
/// TBD
/// </summary>
/// <param name="isFuzzingMode">TBD</param>
/// <returns>TBD</returns>
```

4 usages Sean Gilliam +2

```
public ActorMaterializerSettings WithFuzzingMode(bool isFuzzingMode)
{
    return new ActorMaterializerSettings(InitialInputBufferSize, MaxI
}
```

Хорошее, но приватное описание

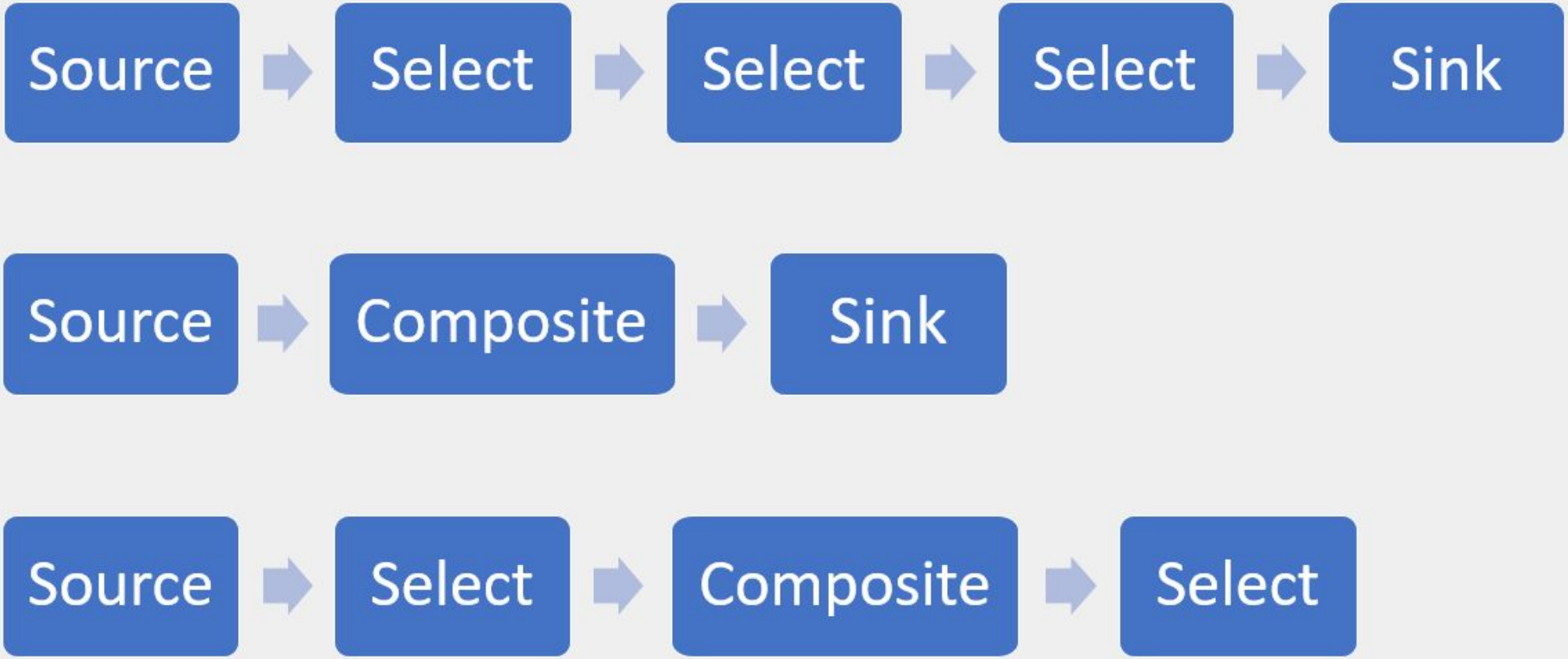
```
/// <summary>
/// Limits the number of events processed by the interpreter before scheduling
/// a self-message for fairness with other actors. The basic assumption here is
/// to give each input buffer slot a chance to run through the whole pipeline
/// and back (for the elements).
///
/// Considered use case:
///  - assume a composite Sink of one expand and one fold
///  - assume an infinitely fast source of data
///  - assume maxInputBufferSize == 1
///  - if the event limit is greater than maxInputBufferSize * (ins + outs) than there will always be expand activity
///  because no data can enter "fast enough" from the outside
/// </summary>
private readonly int _shellEventLimit;
```

Доступные настройки материализации

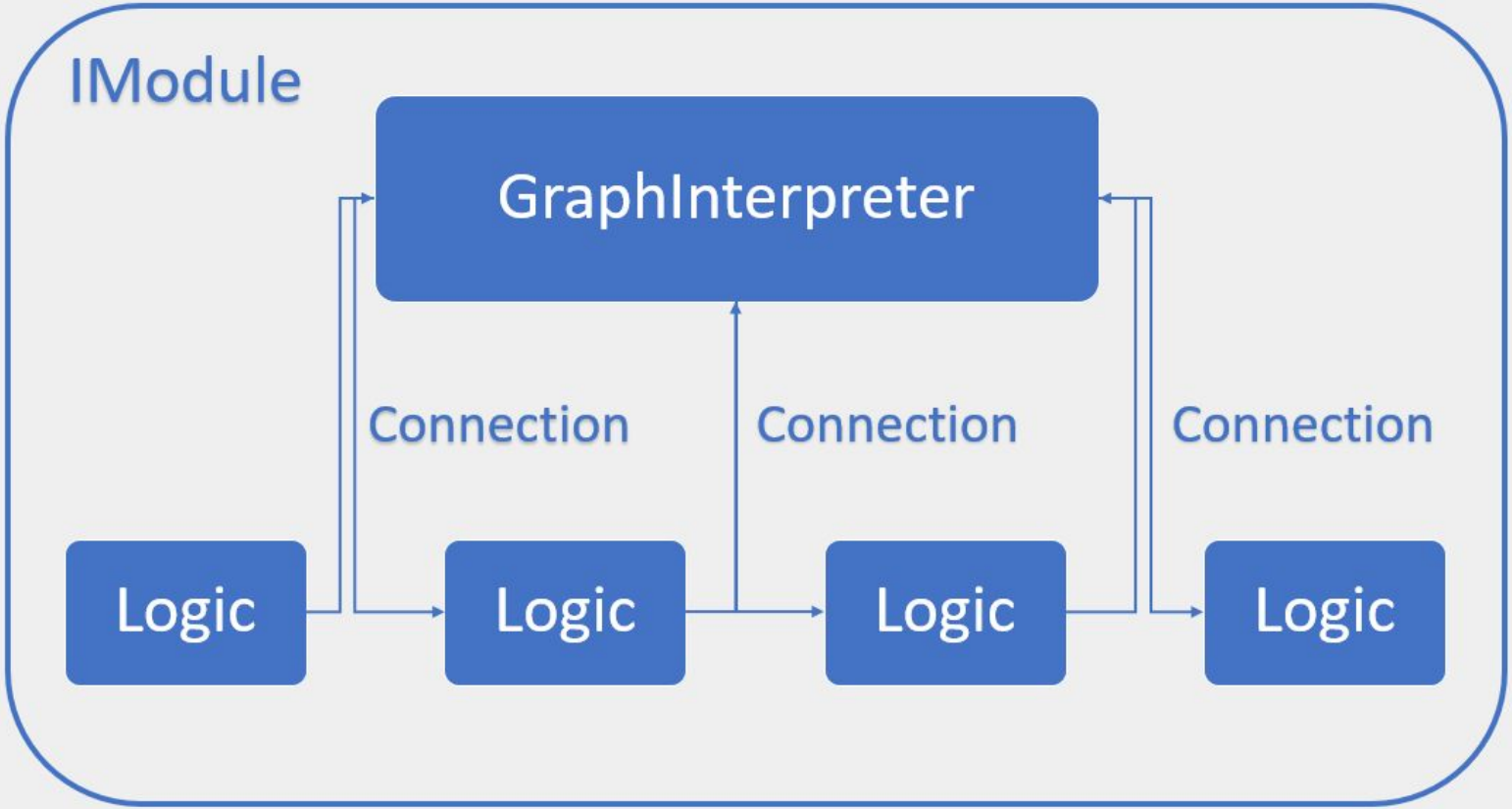
```
private const int DefaultlMaxFixedbufferSize = 1000;
/// <summary>TBD</summary>
public readonly int InitialInputBufferSize;
/// <summary>TBD</summary>
public readonly int MaxInputBufferSize;
/// <summary>TBD</summary>
public readonly string Dispatcher;
/// <summary>TBD</summary>
public readonly Akka.Streams.Supervision.Decider SupervisionDecider;
/// <summary>TBD</summary>
public readonly StreamSubscriptionTimeoutSettings SubscriptionTimeoutSettings;
/// <summary>TBD</summary>
public readonly bool IsDebugLogging;
/// <summary>TBD</summary>
public readonly int OutputBurstLimit;
/// <summary>TBD</summary>
public readonly bool IsFuzzingMode;
/// <summary>TBD</summary>
public readonly bool IsAutoFusing;
/// <summary>TBD</summary>
public readonly int MaxFixedBufferSize;
/// <summary>TBD</summary>
public readonly int SyncProcessingLimit;
```

Источник: фильм «Скрипач на крыше» (1971)

Принцип объединения этапов



Как происходит материализация



Почему SelectAsync - Unordered?

```
var task = _stage._mapFunc(Grab(_stage.In));
var holder = new Holder<TOut>(NotYetThere, _taskCallback);
_buffer.Enqueue(holder);

// We dispatch the task if it's ready to optimize away
// scheduling it to an execution context
if (task.IsCompleted)
{
    holder.SetElement(Result.FromTask(task));
    HolderCompleted(holder);
}
else
    task.ContinueWith(t => holder.Invoke(Result.FromTask(t)),
        TaskContinuationOptions.ExecuteSynchronously);
```

Почему SelectAsync - Unordered?

```
var task = _stage._mapFunc(Grab(_stage.In));
_inFlight++;

if (task.IsCompleted)
    TaskCompleted(Result.FromTask(task));
else
    task.ContinueWith(
        t => _taskCallback(Result.FromTask(t)),
        TaskContinuationOptions.ExecuteSynchronously);
```

Логи в akka.net

- Log() Flow для логирование проходящих
- Логи состояния работающего актора
- Логи материализации

Параметр для логирования

```
namespace Akka.Streams.Implementation
{
  /// <summary>
  /// INTERNAL API
  /// </summary>
  [InternalApi]
  public abstract class MaterializerSession
  {
    /// <summary>
    /// TBD
    /// </summary>
    public static readonly bool IsDebug;
```

Пример выводимых логов

```
beginning materialization of
  CompositeModule [1563714%08x]
  Name:
  Modules:
    enumerableSource
    53046438 copy of GraphStage(Select) [22022881%08x]
    30850230 copy of GraphStage(Log) [47344677%08x]
    764241 copy of GraphStage(Log) [14292056%08x]
    66661912 copy of GraphStage(Akka.Streams.Implementation.Fusing.Buffer`1[System.Int32]) [9340859%08x]
    40098280 copy of GraphStage(Log) [62666951%08x]
    56542610 copy of GraphStage(Select) [15246445%08x]
    31131810 copy of GraphStage(Akka.Streams.Dsl.RestartWithBackoffFlow`3[System.Int32,System.Int32,Akka.NotUsed]) [63741013%08x]
    3726363 copy of GraphStage(Select) [9359138%08x]
    15932216 copy of GraphStage(Akka.Streams.Implementation.Fusing.SelectAsync`2[System.Int32,System.Int32]) [63708843%08x]
    26563115 copy of GraphStage(IgnoreSink) [7525443%08x]
  Downstreams:
    Select.out -> RestartWithBackoffFlow.in
    Select.out -> Log`1.in
    Log`1.out -> Select.in
    Select.out -> SelectAsync.in
    Log`1.out -> Log`1.in
    RestartWithBackoffFlow.out -> Select.in
    StatefulSelectMany.out -> Select.in
    SelectAsync.out -> Ignore.in
```

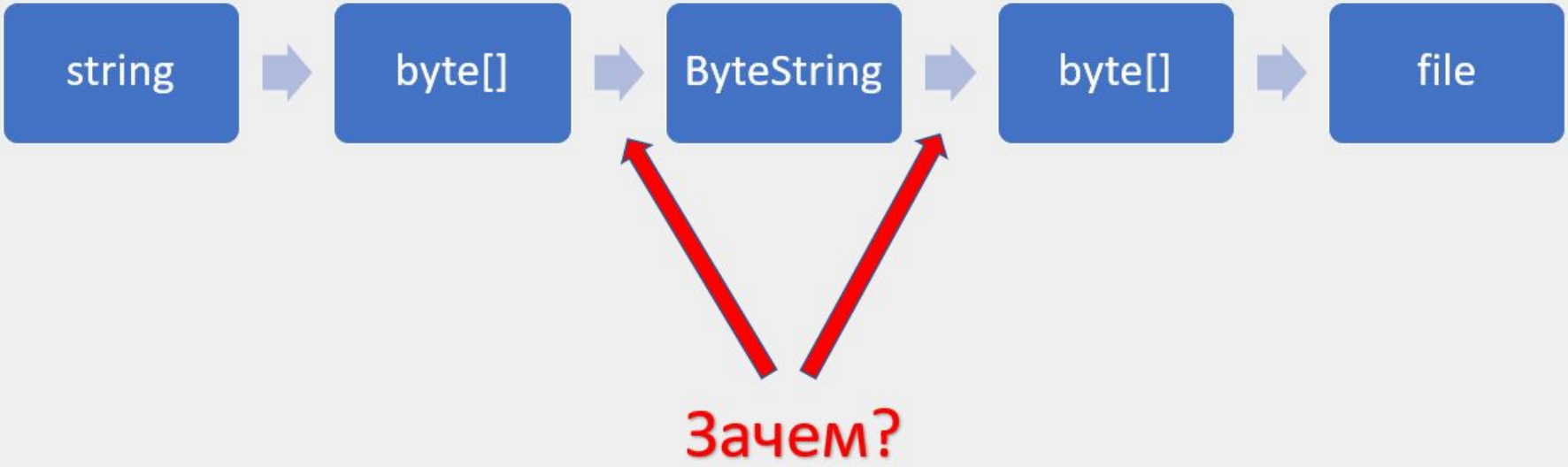
Фото взято из открытых источников

Минусы FileIO

- Возврат IOResult
- Не дает настройку возможности работы с файловой системой
- Работает только с ByteString

Запись в файл через FileIO

Запись в файл через FileIO



Диспетчеры в akka.net

ThreadPoolDispatcher

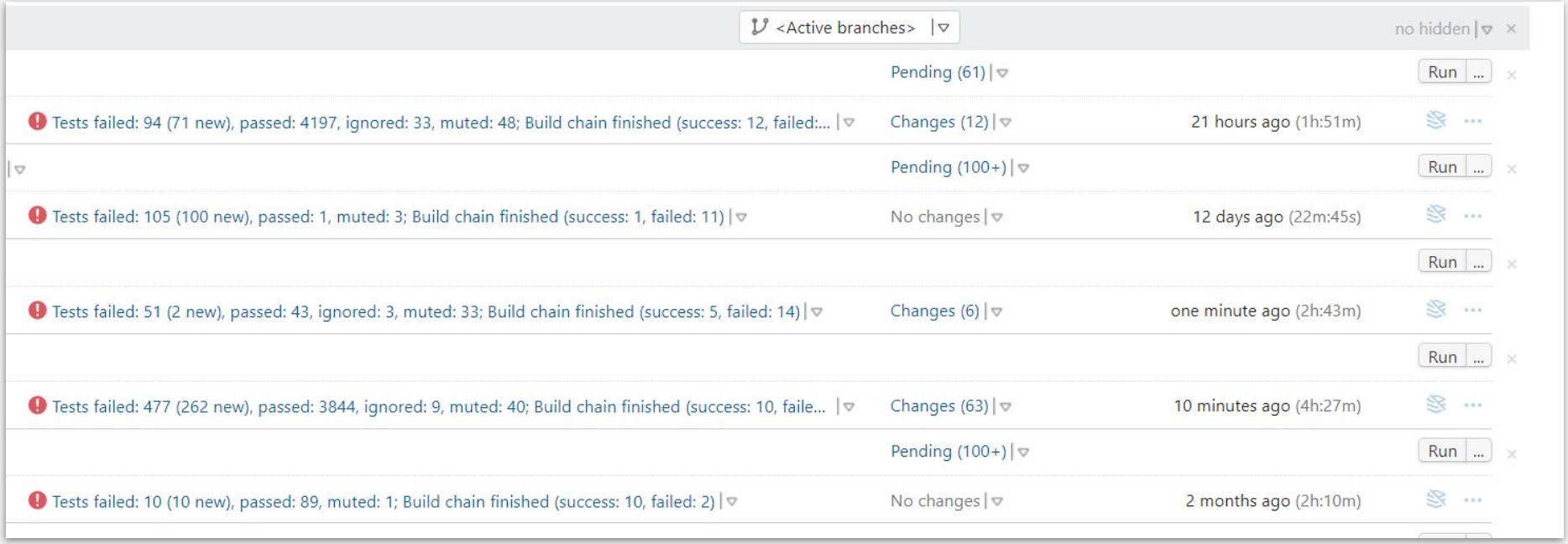
TaskDispatcher

PinnedDispatcher

ForkJoinDispatcher

SynchronizedDispatcher

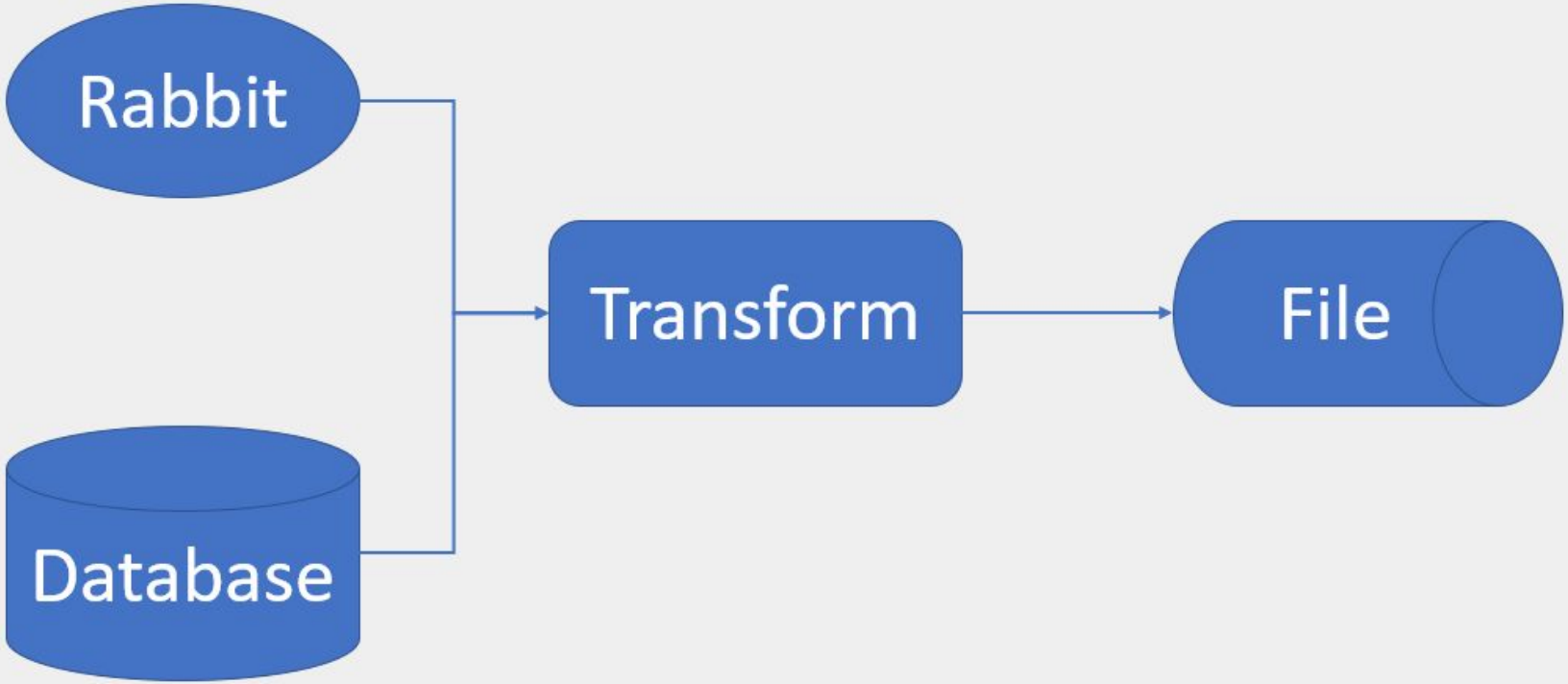
Тестирование - это хорошо.
Тестирование - это надежно



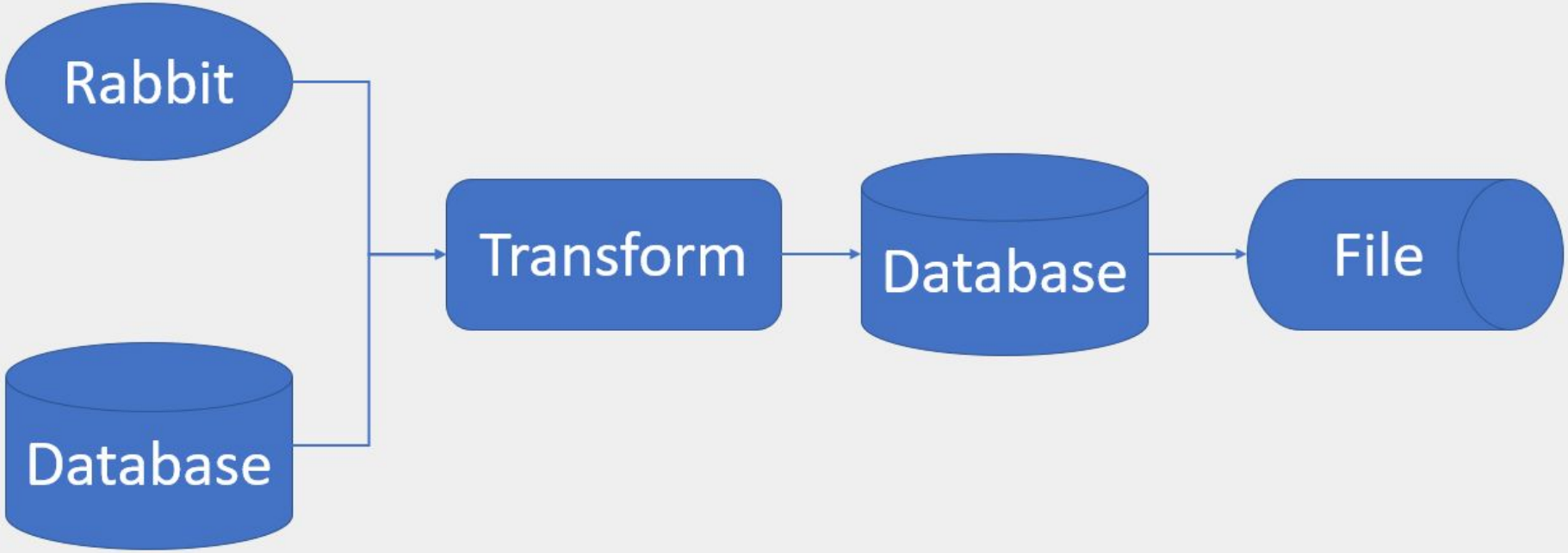
Три подхода к тестированию

- Мокирование
- Использование Akka.TestKit
- Использование Akka.Streams.Testkit

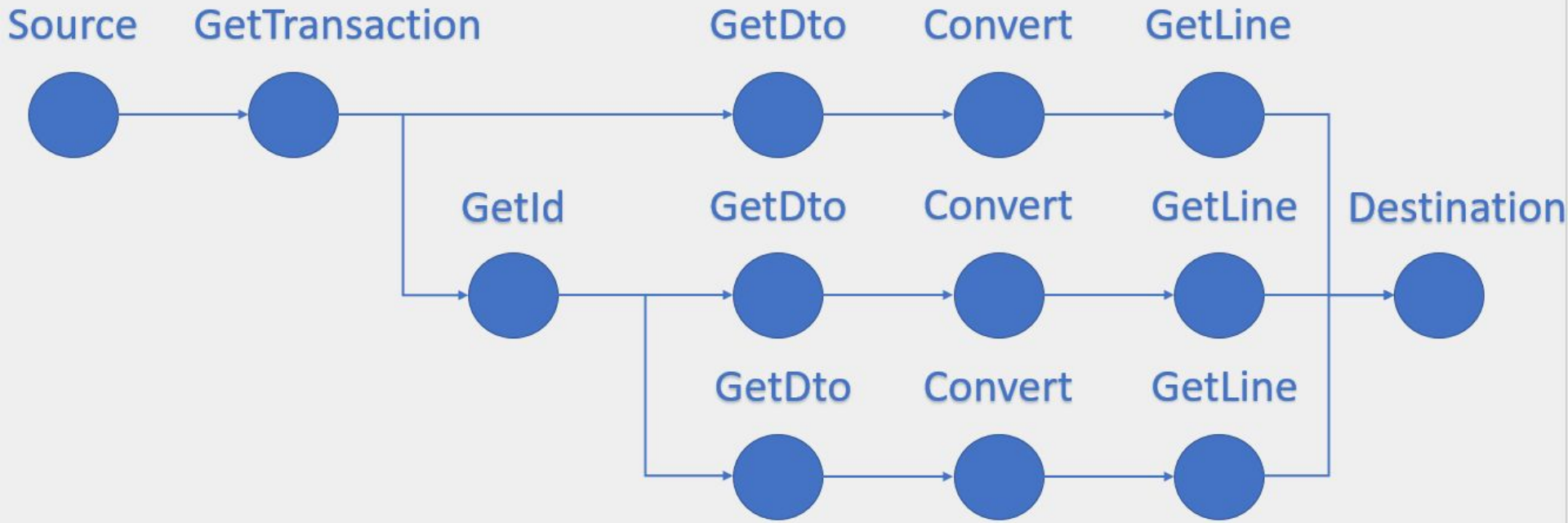
Реализуемая задача. Схема данных



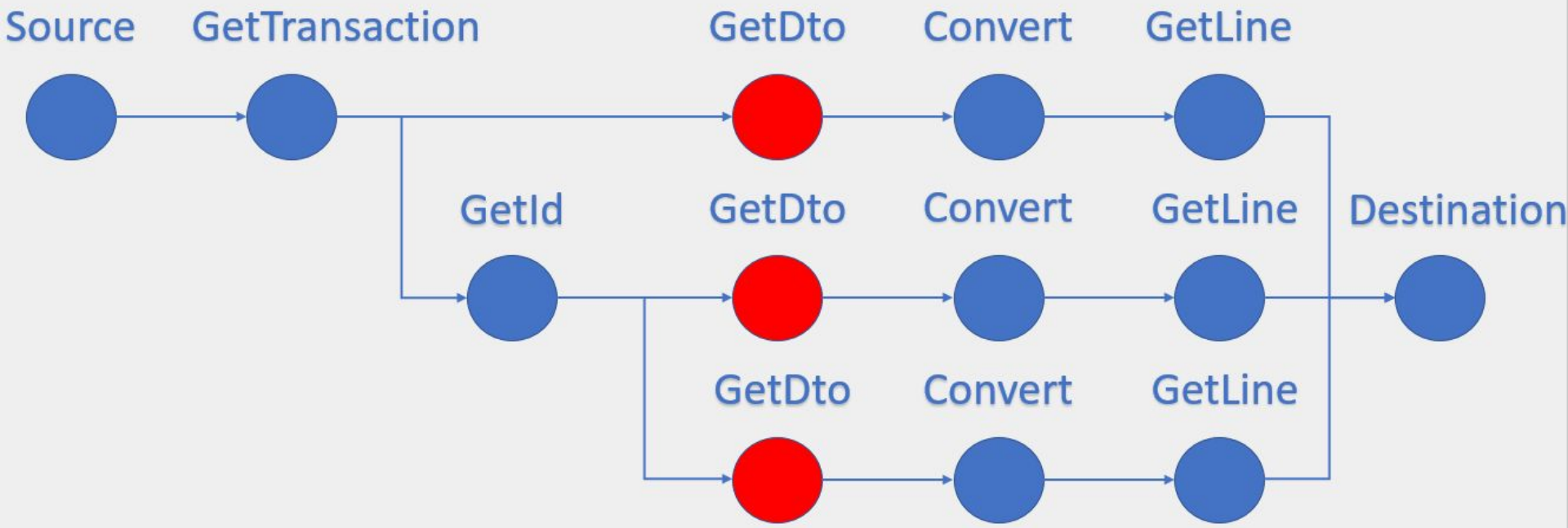
Реализуемая задача. Схема данных



Граф прохождения данных



Проблемные места



Построение через balance и merge

```
var balance = builder.Add(new Balance<int>(maxDegreeOfParallelism));
var merge = builder.Add(new Merge<int>(maxDegreeOfParallelism));

for (var i = 0; i < maxDegreeOfParallelism; i++)
{
    var flow = GetTransactionFlow();
    builder.From( outlet: balance.Out(i))
        .Via(flow)
        .To( inlet: merge.In(i));
}

return new FlowShape<int, int>(balance.In, merge.Out);
```

```
1 usage
public Logic(FileSink sink) : base(sink.Shape)
{
    _writer = new StreamWriter(File.OpenWrite(sink.FilePath)) {AutoFlush = sink.AutoFlush};
    _sink = sink;
}

public override void PreStart() => Pull(_sink.In);

public override void PostStop()
{
    _writer?.Dispose();
    base.PostStop();
}

public override void OnPush()
{
    var element = Grab(_sink.In);
    _writer.WriteLine(element);
    Pull(_sink.In);
}

public override void OnUpstreamFinish()
{
    _promise.TrySetResult(true);
    CompleteStage();
}
```

Rx.NET:

- Позволяет реализовать реактивную модель обработки
- Синхронны
- Требует написания большого количества кода
- Нет большого количества встроенных интеграций и промежуточных блоков

Orleans Streams:

- Реализует api reactive streams
- Похожи на Rx.NET
- Часть Orleans
- Позволяют динамически менять поток
- Асинхронны

TPL.DataFlow:

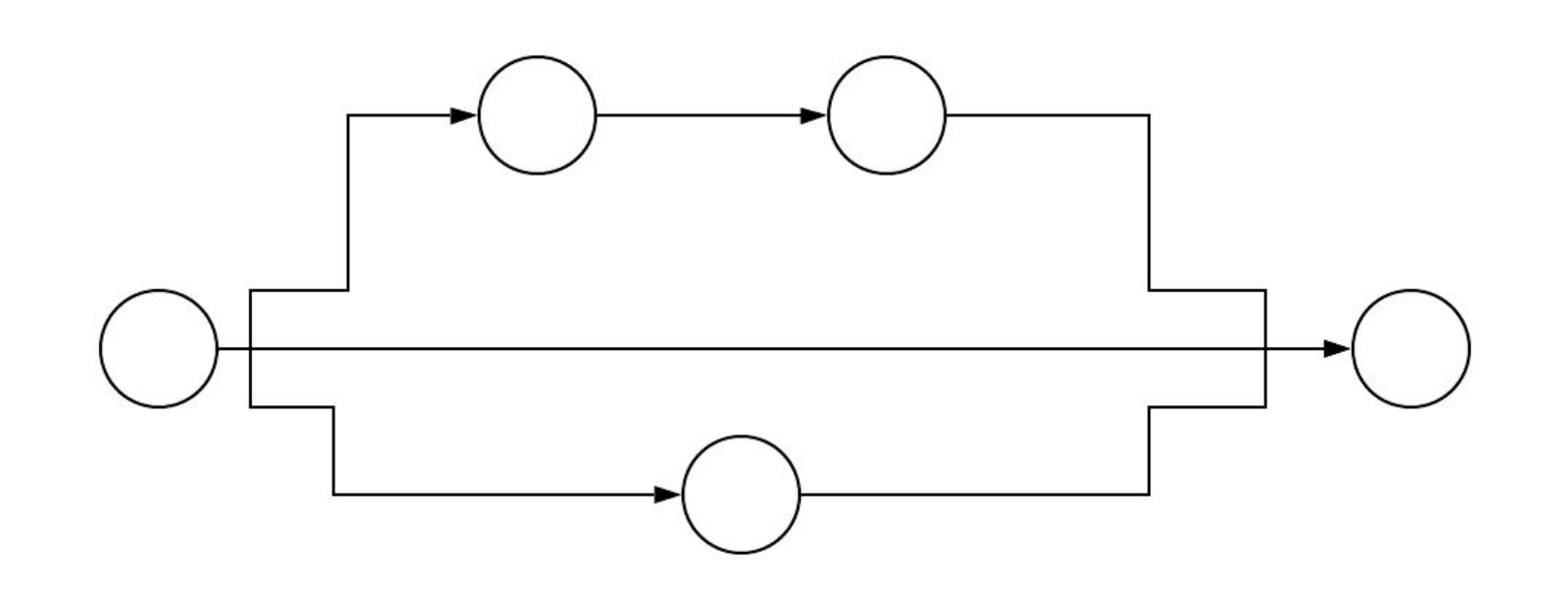
- Позволяют строить конвейеры для обработки
- Асинхронны
- Предлагают меньше вариантов управления потоком

Akka.net streams:

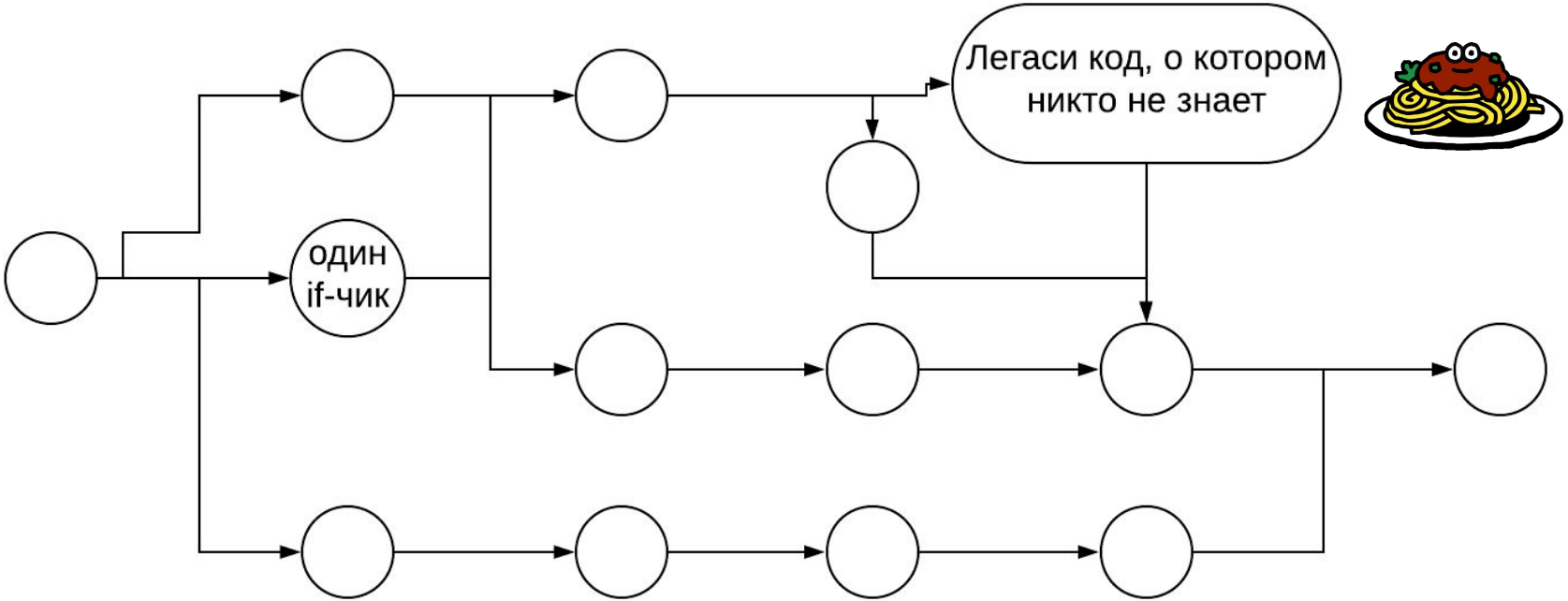
- Много готовых блоков;
- Достаточно стабильно работает;
- Код более функциональный;
- Легко вносить изменения;
- Очень высокая скорость разработки;
- Визуализация прохождения данных.

- Плохое описание публичных методов;
- Мало описания в интернете;
- Неоптимальный код;
- Есть порог входа.

Пример модели прохождения данных



Пример модели прохождения данных



Полезные ссылки

1) Документация akka.net:

   - https://getakka.net/articles/streams/introduction.html

2) Расшифровка доклада:

   - https://habr.com/ru/company/jugru/blog/418639/

3) Исходники akka.net streams:

   - https://github.com/akkadotnet/akka.net/tree/dev/src/core/Akka.Streams

zhitkon@gmail.com

@cdcvmm