Обзор библиотеки TPL Dataflow

TPL Dataflow – что это и для чего?

- Это библиотека, разрабатываемая Microsoft, распространяемая через NuGet
- TPL Dataflow помогает повысить производительность и устойчивость нагруженных параллельных приложений
- TPL Dataflow является реализацией “модели акторов”
- Данный подход, основан на построении вычислительного конвейера, состоящего из вычислительных блоков, работающих параллельно и независимо друг от друга
- Взаимодействие происходит путём передачи асинхронных сообщений между блоками

Модель акторов

Это математическая модель параллельных вычислений в которой актор является вычислительной сущностью. Актор может:

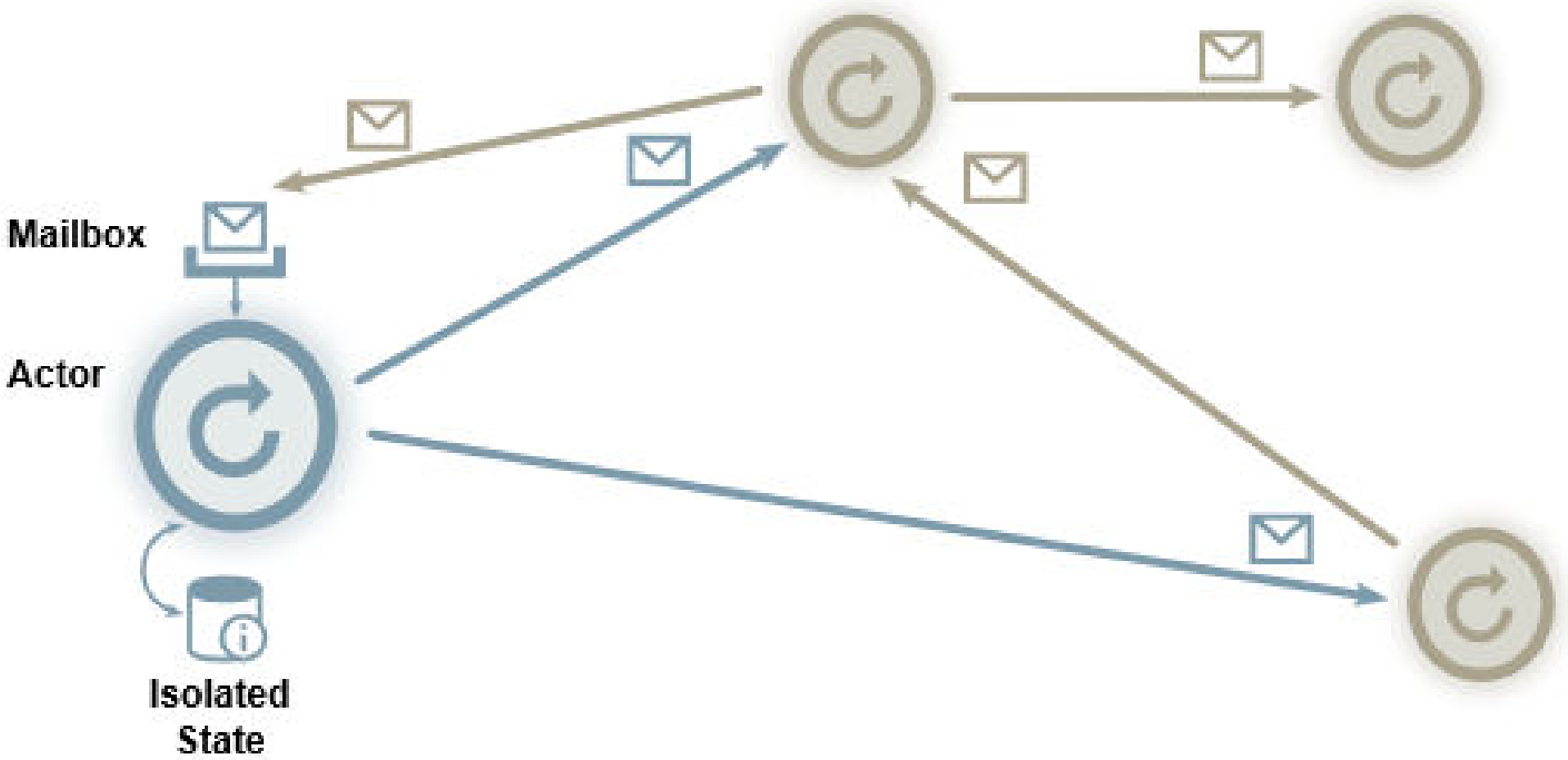
- принимать сообщения от других акторов
- отправить сообщения другим акторам
- создать новые акторы
- выполнить какую-либо присущую актору работу
- иметь состояние

Акторам присущий внутренний параллелизм

Акторы “изолированы” друг от друга

Акторы взаимодействуют друг с другом путём обмена **асинхронными** сообщениями

Картинка из интернета



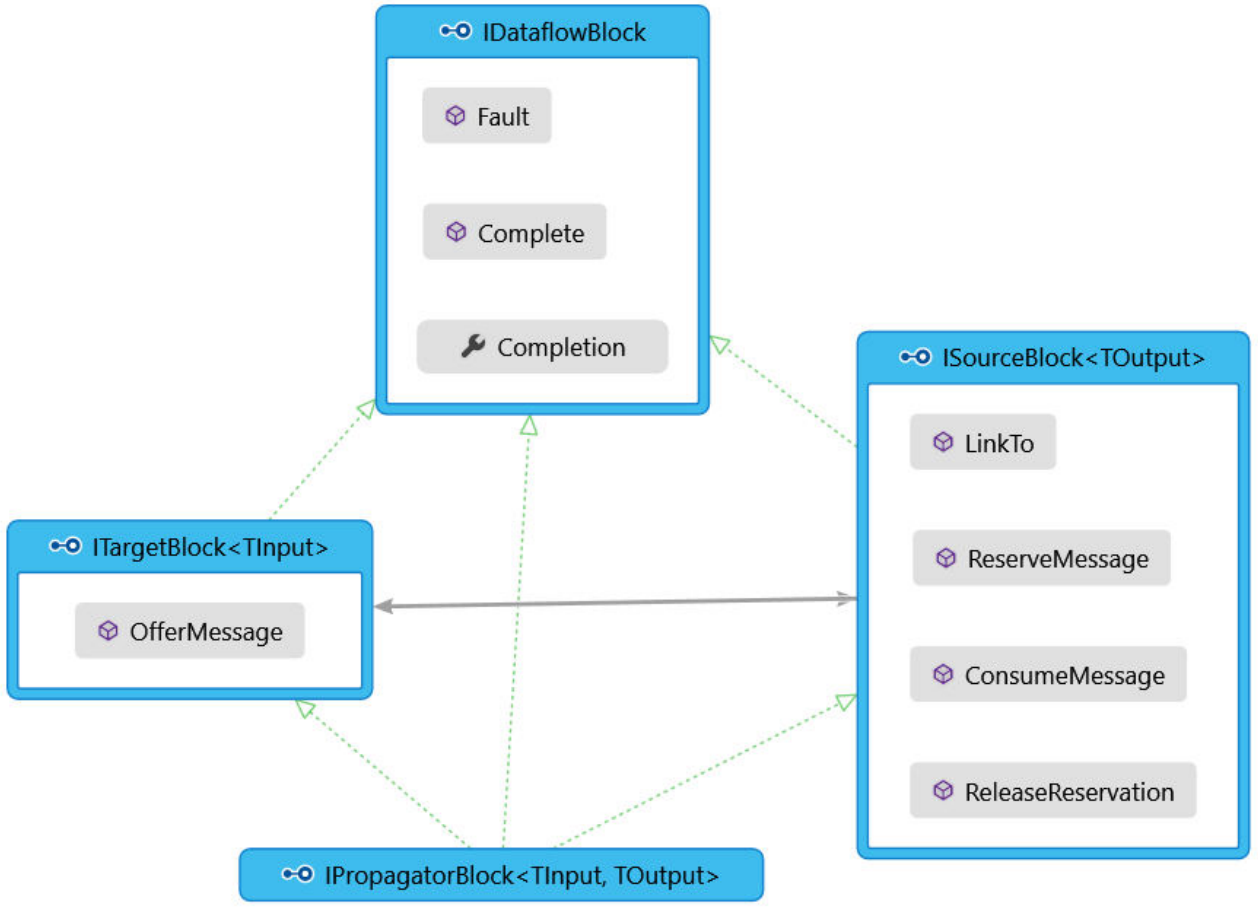
Программная модель

Базовая сущность – блок dataflow (dataflow block)

Есть три вида блоков:

- Блоки источники (source blocks) – действуют как источники данных – из них можно читать
- Блоки приёмники (target blocks) – действуют как приёмники данных – в них можно писать
- Блоки “распространители” (propagator blocks) – действуют как одновременно источники и приёмники данных – из/в них можно и читать, и писать

Диаграмма интерфейсов



Запись и чтение сообщений из блока Dataflow

Блоки предоставляют различные способы записи и чтения сообщений

Демнстрация асинхронной записи/чтения:

```
var bufferBlock = new BufferBlock<int>();

// Send messages to the block asynchronously.
for (var i = 0; i < 3; ++i)
{
  await bufferBlock.SendAsync(i);
}

// Asynchronously receive the messages back from the block.
for (var i = 0; i < 3; ++i)
{
  Console.WriteLine(await bufferBlock.ReceiveAsync());
}

/* Output:
   0
   1
   2 */
```

Связывание блоков

- Блоки можно связывать, образуя конвейеры или графы
- Один блок источник можно связать с множеством блоков приёмников
- Один блок приёмник можно связать с множеством блоков источников
- Используетмя метод ISourceBlock<TOutput>.LinkTo
- Метод LinkTo дополнительно может принимать предикат, позволяющий фильтровать сообщения

Связывание блоков - демонстрация

```
var bufferBlock = new BufferBlock<int>();
var batchBlock = new BatchBlock<int>(4);
var actionBlock = new ActionBlock<int[]>(values => Console.WriteLine(String.Join(" ", values)));

// Link bufferBlock output to batchBlock input
bufferBlock.LinkTo(batchBlock, new DataflowLinkOptions { PropagateCompletion = true });

// Link batchBlock output to actionBlock input
batchBlock.LinkTo(actionBlock, new DataflowLinkOptions { PropagateCompletion = true });

for (var i = 0; i < 16; ++i)
{
  await bufferBlock.SendAsync(i);
}

// Complete blocks and wait for completion to avoid them to be early collected by GC
bufferBlock.Complete();
await actionBlock.Completion;

/* Output:
    0 1 2 3
    4 5 6 7
    8 9 10 11
    12 13 14 15 */
```

Предопределённые блоки

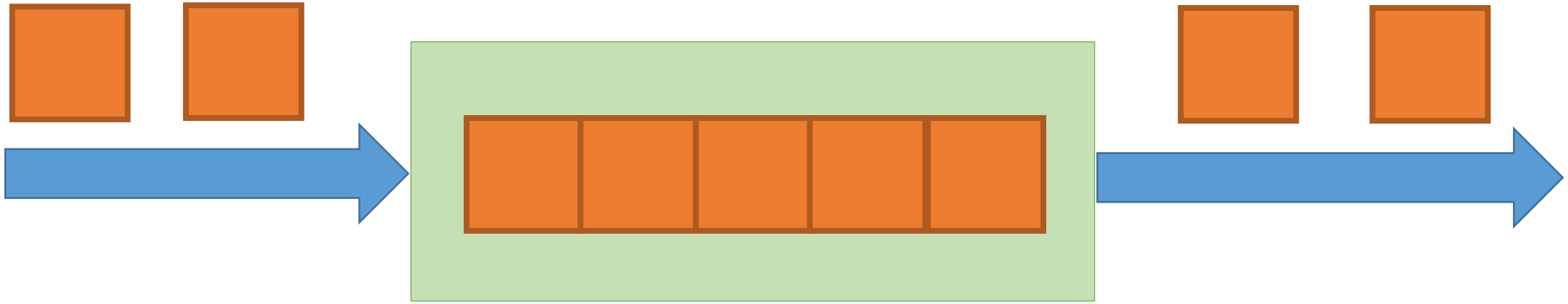
Они делятся на 3 категории:

- Буферные блоки (buffering blocks)
- Исполнительные блоки (execution blocks)
- Группирующие блоки (grouping blocks)

Буферные блоки (buffering blocks)

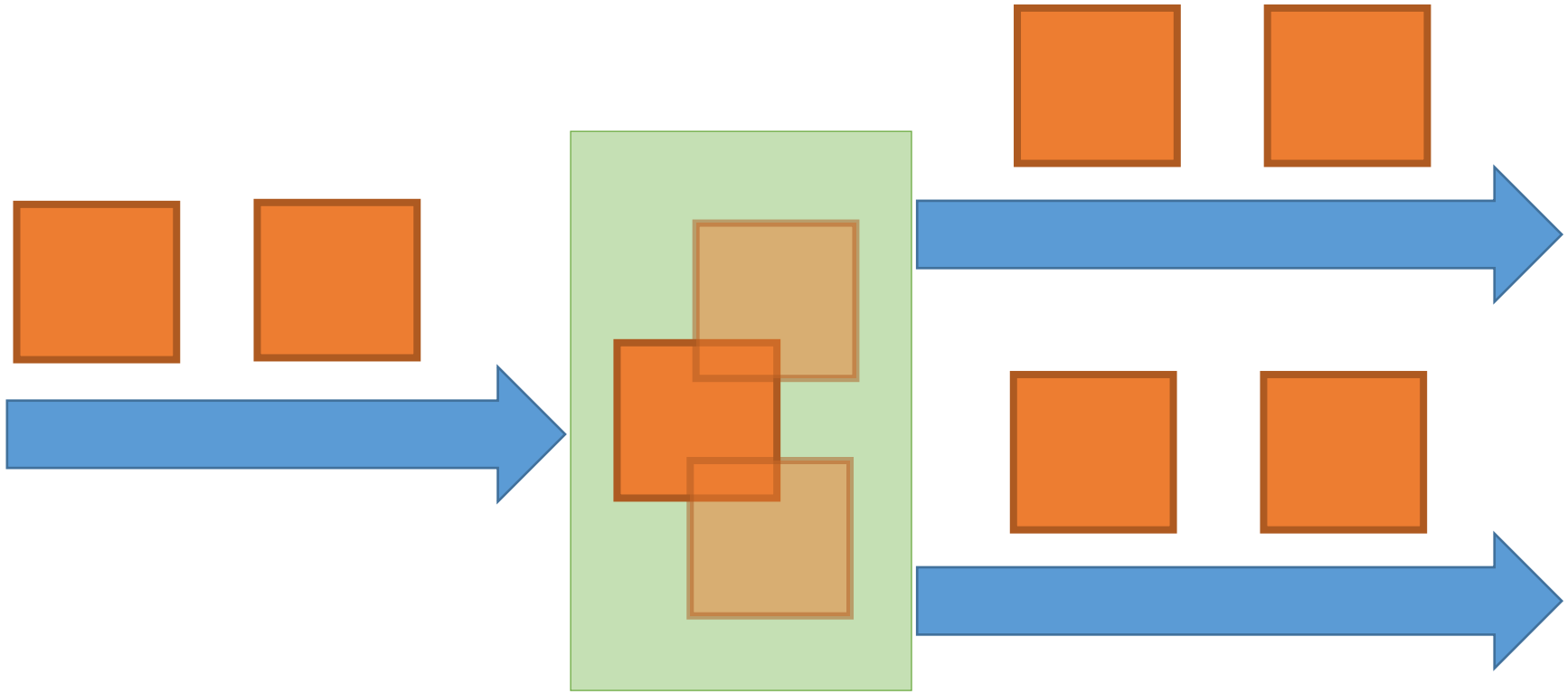
BufferBlock<T>

- Реализует очередь FIFO общего назначения
- Сообщения могут приходить из множества источников у уходить во множество приёмников при этом каждое сообщение попадает только в один приёмник
- Полезен если требуется гарантированная доставка всех сообщений в определённый приёмник



BroadcastBlock<T>

- Полезен, когда нужно передать каждое сообщение всем приёмникам
- Также полезен, когда требуется передавать множество сообщений определённому приёмнику и при этом приёмнику всегда требуется только самое последнее сообщение (т. е. нас не волнует потеря сообщений)



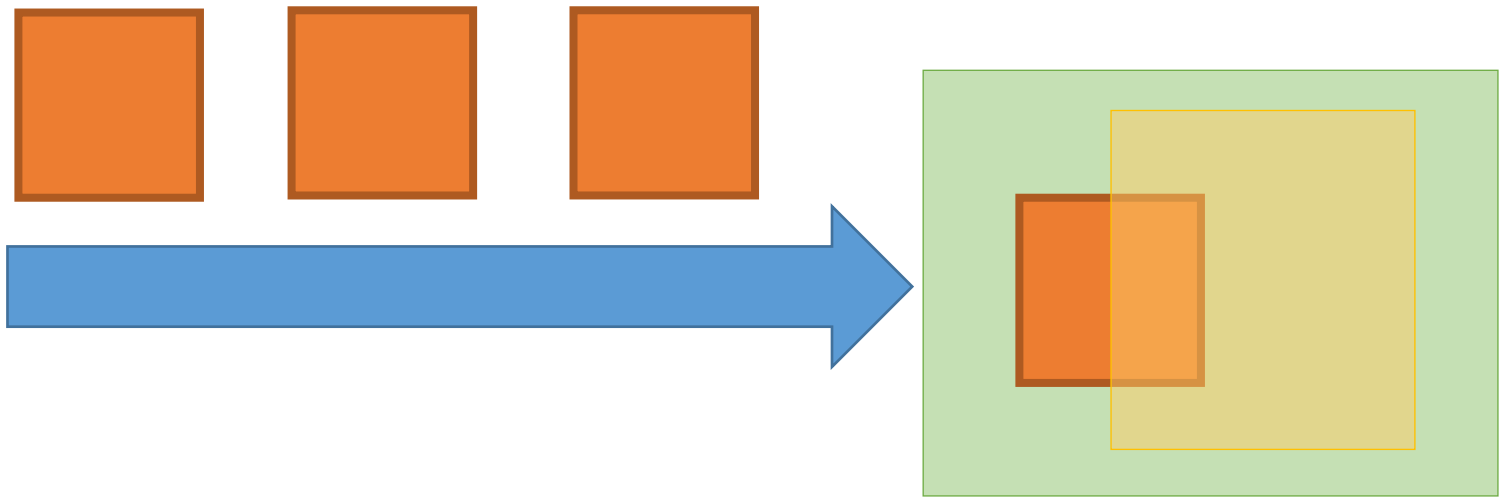
WriteOnceBlock<T>

- Является частным случаем BroadcastBlock<T>
- Отличие состоит в том что в него можно записать всего одно сообщение

Исполнительные блоки (execution blocks)

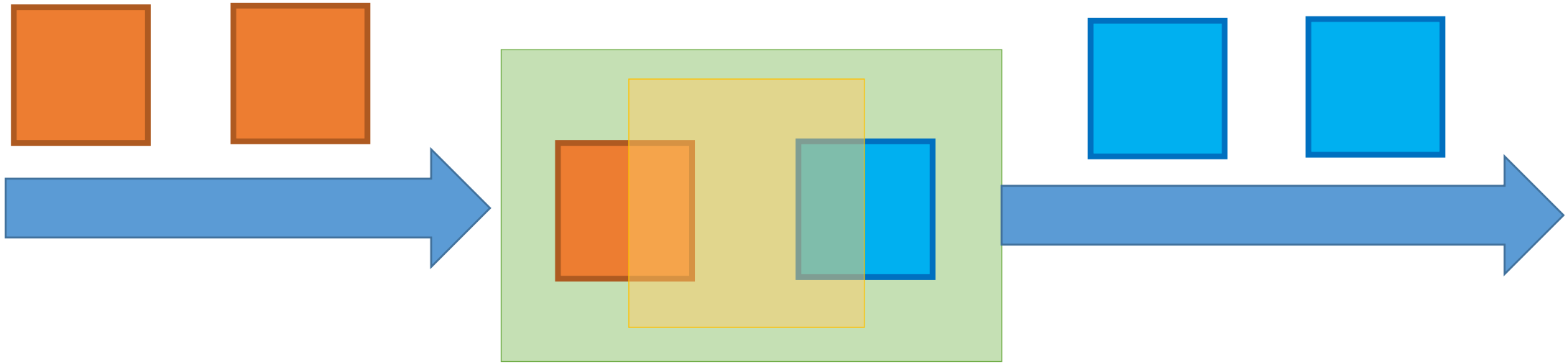
ActionBlock<TInput>

- Это блок-приёмник
- выполняет заданное действие над каждым поступающим сообщением
- Действие может быть:
  - синхроное – задаётся с помощью Action<TInput>
  - асинхроное – задаётся с помощью Func<TInput, Task>



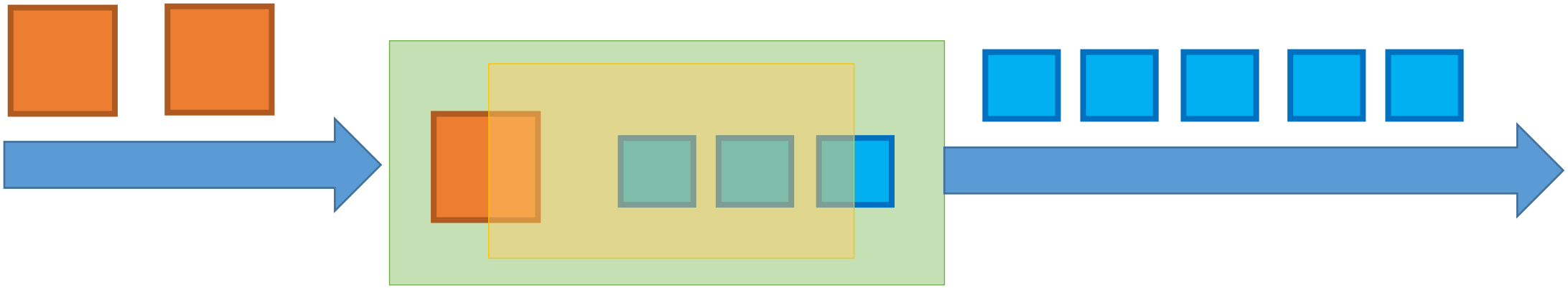
TransformBlock<TInput, TOutput>

- Трансформирует каждое входящее сообщение в исходящее сообщение выполняя заданное действие
- Действие может быть:
  - синхроное – задаётся с помощью Func<TInput, TOutput>
  - асинхроное – задаётся с помощью Func<TInput, Task<TOutput>>



TransformManyBlock<TInput, TOutput>

- Похож на TransformBlock
- Трансформирует каждое входящее сообщение в последовательность исходящих сообщений выполняя заданное действие
- Действие может быть:
  - синхроное – задаётся с помощью Func<TInput, IEnumerable<TOutput>>
  - асинхроное – задаётся с помощью Func<TInput, Task<IEnumerable<TOutput>>>



Уровень параллелизма

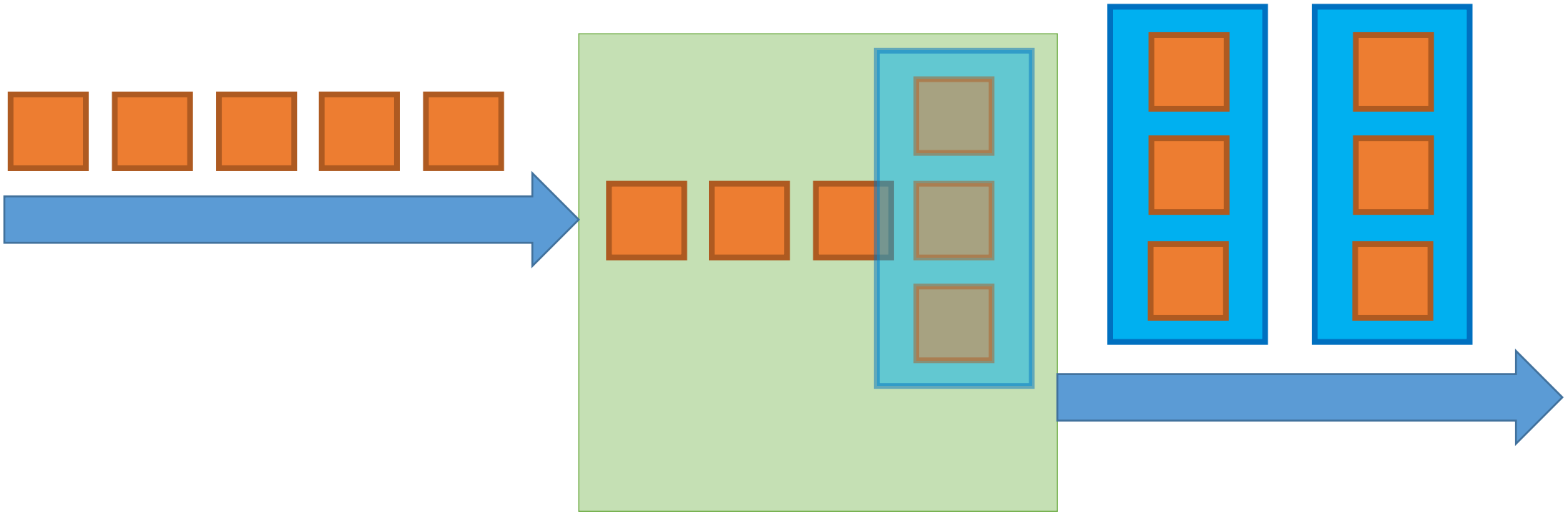
По умолчанию все исполнительные блоки обрабатывают только одно сообщение одновременно

Поведение по-умолчанию можно изменить, задавая уровень параллелизма – это позволяет обрабатывать одновременно множество сообщением

Группирующие блоки (grouping blocks)

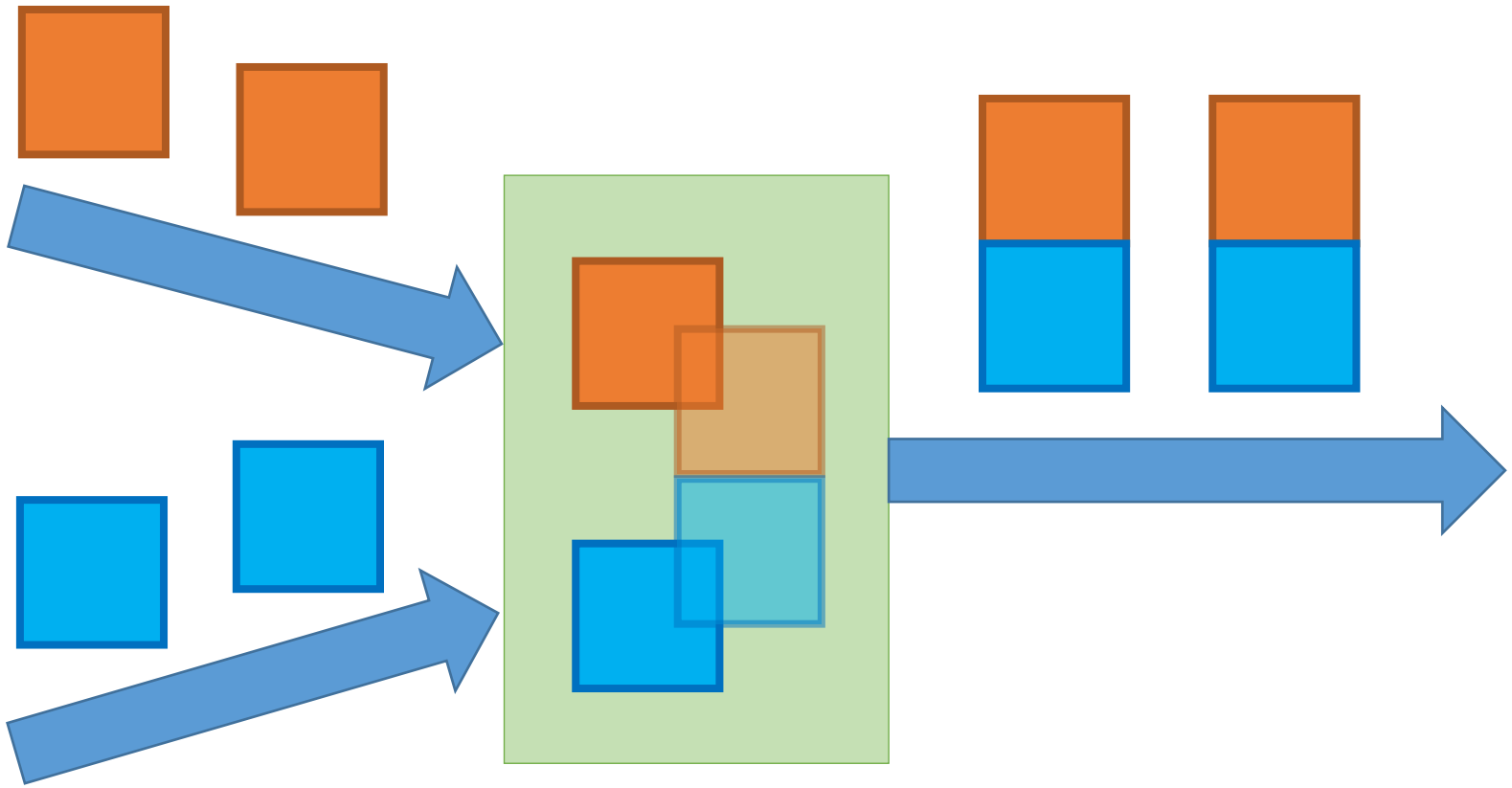
BatchBlock<T>

Комбинирует входящие сообщения в пакеты заданного размера и отдаёт их в качестве исходящих сообщений



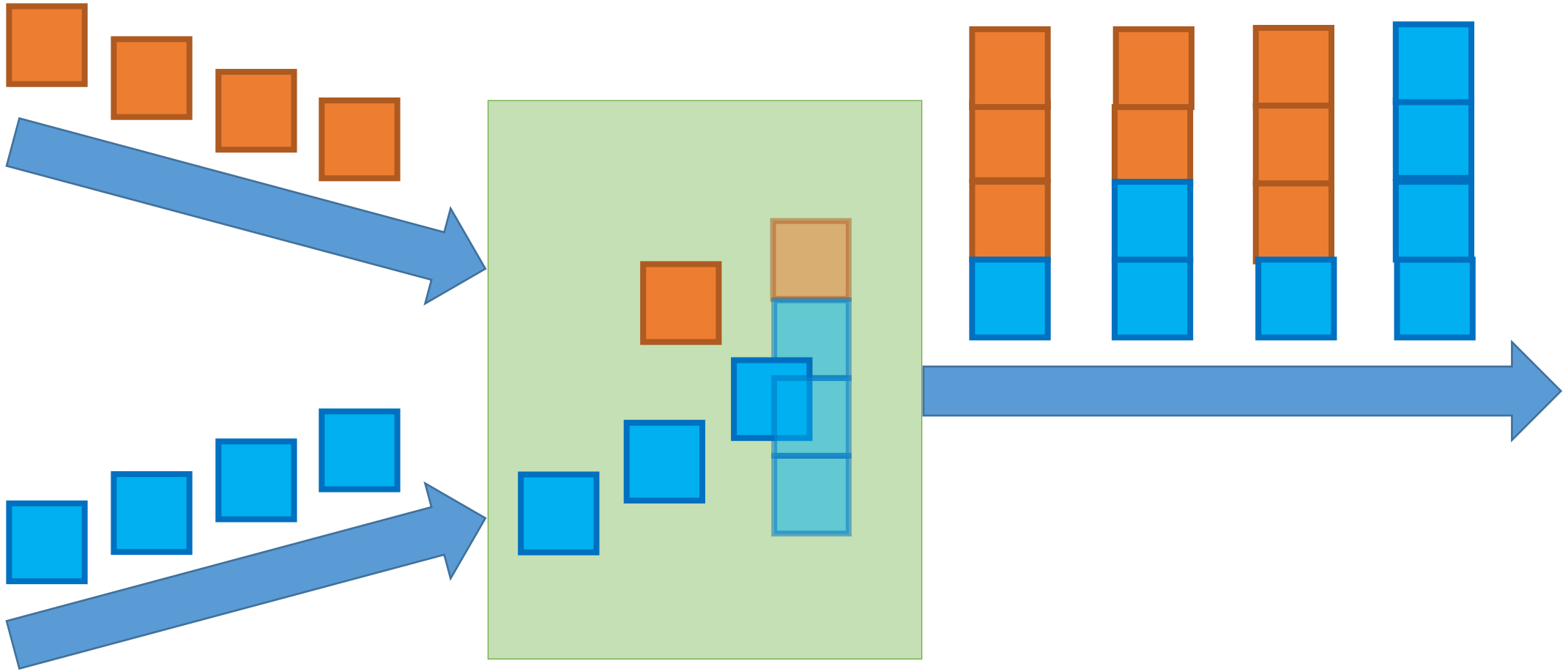
JoinBlock<T1, T2>, JoinBlock<T1, T2, T3>

Собирает сообщения из двух (или трёх) независимых источников в пары (или тройки) и отдаёт их в качестве исходящих сообщений



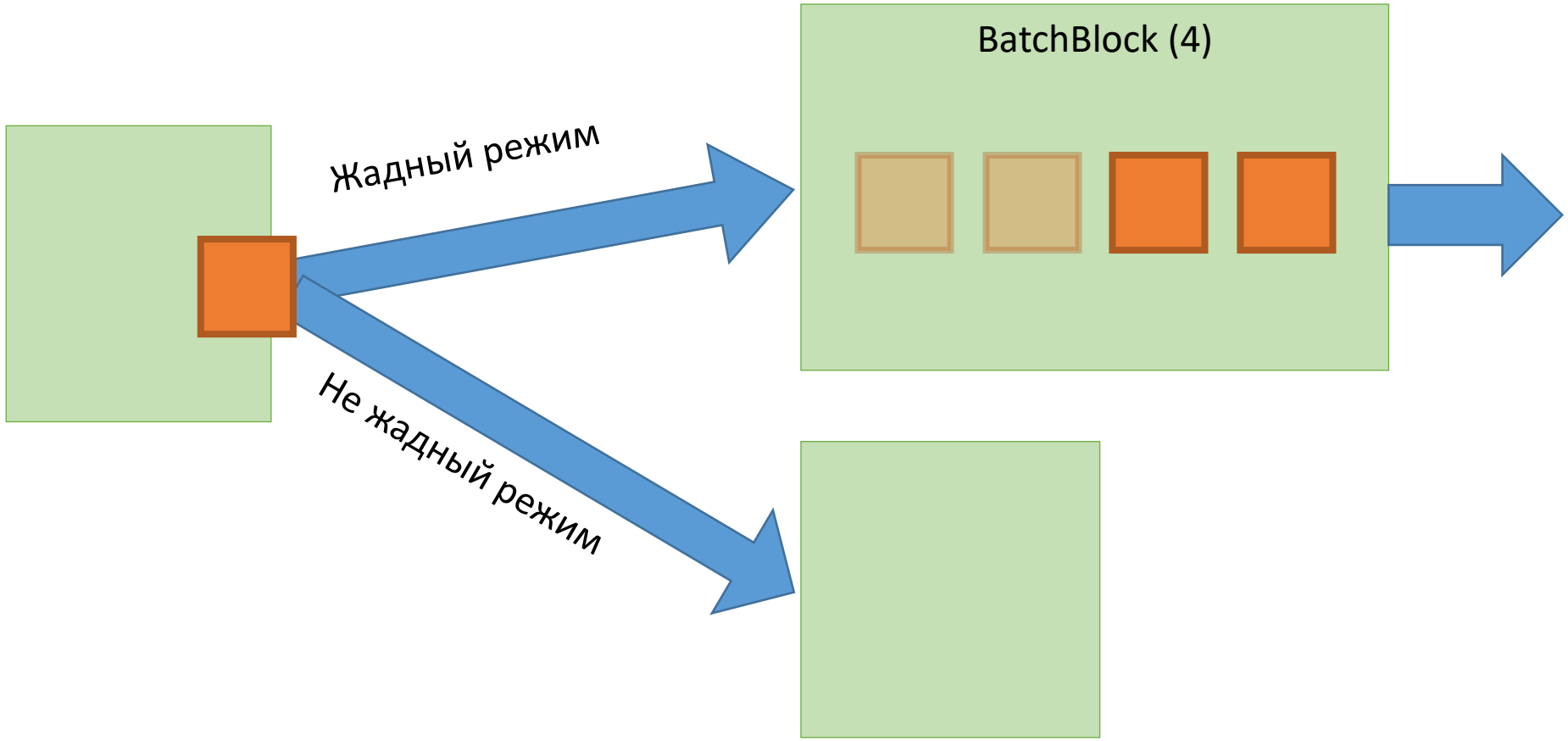
BatchedJoinBlock<T1, T2>, BatchedJoinBlock<T1, T2, T3>

Является комбинацией BatchBlock и JoinBlock



Режим “жадности” (greedy mode)

Группирующие блоки могут работать в жадном и не-жадном режимах



Режим “жадности” (greedy mode)

- В жадном режиме (по-умолчанию) они принимают все сообшения
- В не-жадном режиме они “откладывают” сообщения “на-потом” пока на входе не будет достаточное количество сообшений чтобы сформировать выходное сообщение
- Это даёт возможность отдать эти сообщения другим блокам (если таковые имеются)

Примечание. Подобного поведения так-же можно добиться устанавливая свойство BoundedCapacity – максимальный размер входящего буфера сообщений – оно доступно для всех блоков (не только группирующих)

Создание собственных блоков

- Можно самому реализовать интерфейсы ISourceBlock<TOutput> и/или ITargetBlock<TInput>
- Можно использовать метод Encapsulate<TInput, TOutput> который инкапсулирует готовый конвеер блоков в новый блок

Демо

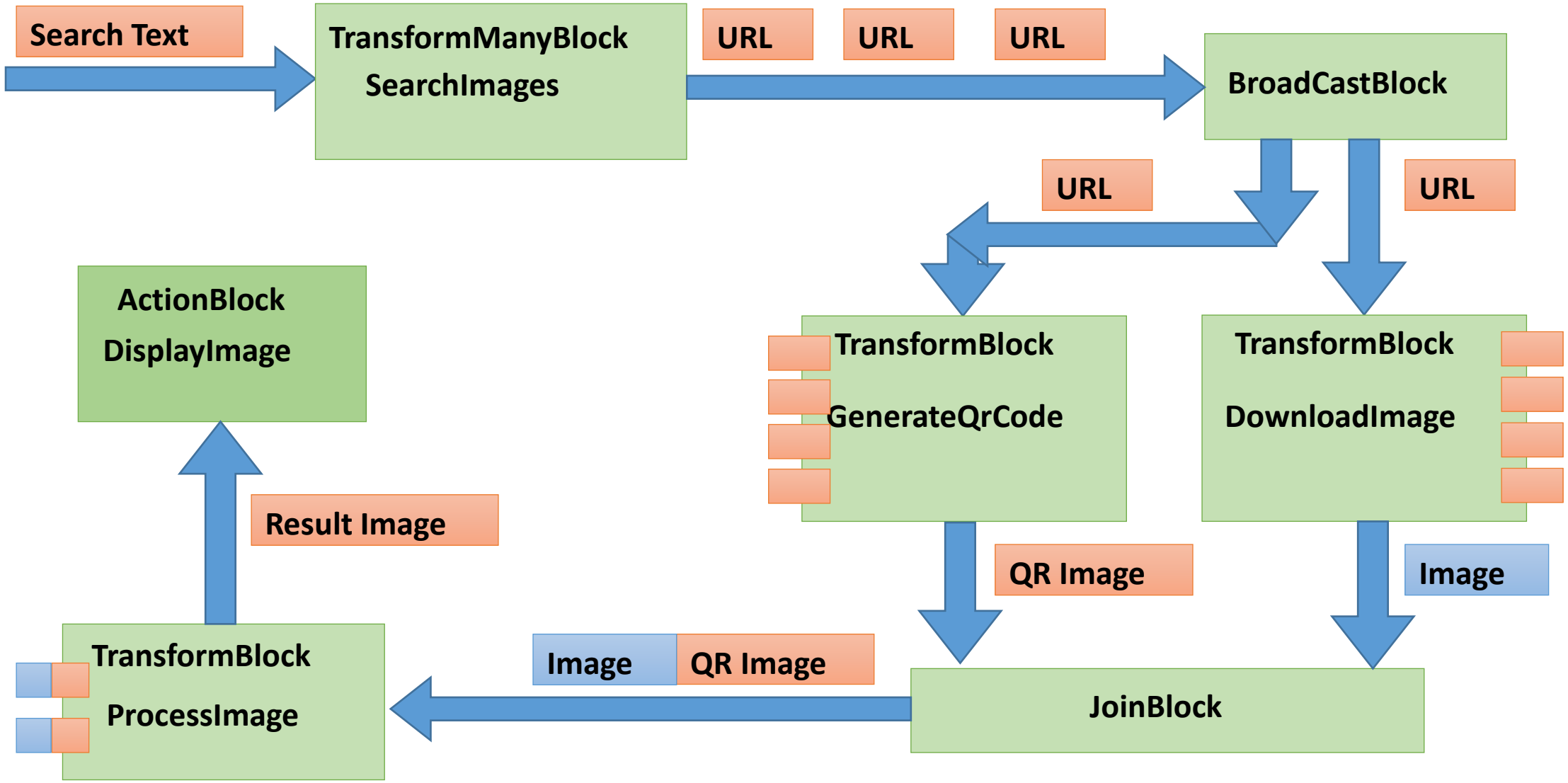
Задача

- По заданной поисковой строке текста найти изображения с помощью поискового сервиса (в данном демо используется Bing)
- Каждое найденное изображение обработать следующим образом:
  - По URL-адресу изображения сформировать QR-код
  - Масштабировать изображение, сверху наложить QR-код и логотип
- Вывести список результирующих изображений

Простое последовательное решение

```
var images = await Logic.SearchImages(searchString);
if (images.Count != 0)
{
  foreach (var imageUrl in images)
  {
    var sourceImage = await Logic.DownloadImage(imageUrl);
    if (sourceImage != null)
    {
      var qrImage = await Logic.GenerateQrCode(imageUrl, new Size(150, 150));
      var resultImage = await Logic.ProcessImage(sourceImage, qrImage, new Size(512, 384));
      AddImage(resultImage);
    }
  }
}
```

Dataflow-решение



Ссылки

- Dataflow (Task Parallel Library) on MSDN https://msdn.microsoft.com/en-us/library/hh228603.aspx
- TPL Dataflow Tour on Channel 9 http://channel9.msdn.com/posts/TPL-Dataflow-Tour
- Stephen Toub: Inside TPL Dataflow on Channel 9 http://channel9.msdn.com/Shows/Going+Deep/Stephen-Toub-Inside-TPL-Dataflow
- Actor Model in Wikipedia https://en.wikipedia.org/wiki/Actor_model