.NET 4.5 Task Parallel Library Dataflow Networks

Alon Fliess

CodeValue

Agenda

- Introduction to Dataflow Networks
- “Around the Block”
- Building the Network
- Concurrency Control
- Advanced TDF Topics

Task Parallel Library

- Task & Task Scheduler are the building blocks of the .NET parallel execution engine
  - You can use task to spawn parallel execution, or
  - You can use higher level abstract such as
    - PLINQ & Data Parallelism (`Parallel.For`, …)
    - Task Parallel Library Dataflow Network ➔ TDF
- TDF
  - A new way to abstract sequences of code execution over tasks

The Actor Model

- "The *Actor model adopts the philosophy that everything is an actor. This is similar to the everything is an object philosophy used by some object-oriented programming languages, but differs in that object-oriented software is typically executed sequentially, while the Actor model is inherently concurrent.*"

Wikipedia

Introduction to Dataflow Networks

- TDF is a flavor of the actor model
  - It is the managed implementation of the VC++ Asynchronous Agents Library
- TDF is used to create and execute a Dataflow network
  - A good example of such a network is Microsoft Office Excel
  - Any pipeline is a simple Dataflow network
- TDF is a technology for message passing and parallel execution of CPU and I/O intensive applications
- TDF promises high performance, high throughput with low latency

“Around the Block”

- TDF network is built from message passing dataflow blocks
  - *IDataflowBlock*
- There are three types of blocks:
  - Source blocks
    - *ISourceBlock<out Toutput>*
    - *IReceivableSourceBlock<TOutput>*
  - Propagator Blocks
    - IPropagatorBlock<in TInput, out TOutput>
  - Target Blocks
    - ITargetBlock<in TInput>

IDataflowBlock

IPropagatorBlock<T, T>
ITargetBlock<T>
IReceivableSourceBlock<T>
ISourceBlock<T>
IDataflowBlock

BroadcastBlock<T>
Generic Sealed Class

Properties
- Completion

Methods
- BroadcastBlock (+ 1 overload)
- Complete
- LinkTo
- ToString
- TryReceive

Completion — Result Task

Complete
Fault — Shutdown Request

ISourceBlock

IReceivableSourceBlock<TOutput>
Generic Interface
- ISourceBlock<TOutput>
- IDataflowBlock

Methods
- TryReceive
- TryReceiveAll

ISourceBlock<TOutput>
- ConsumeMessage
- LinkTo
- ReleaseReservation
- ReserveMessage

Build the m

ITargetBlock

ITargetBlock<TInput>
- OfferMessage

Request a message from source

Try to provide a message to the target block

Provide transaction like behavior

IPropagatorBlock

IPropagatorBlock<TInput, TOutput>

Generic Interface
- ITargetBlock<TInput>
- ISourceBlock<TOutput>
- IDataflowBlock

ActionBlock<in TIn>

- This is a simple target block
  - It receives data and acts upon it

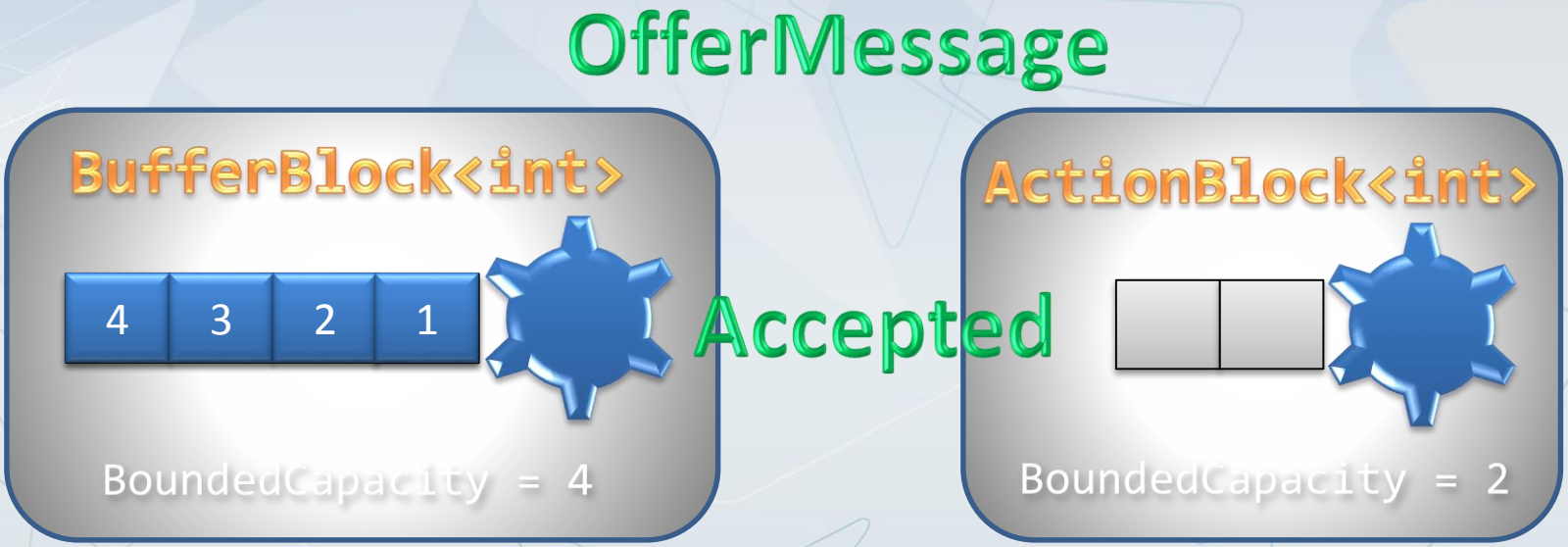
```
var ab = new ActionBlock<int>(i =>
    {
            Console.Write("{0}) Task Id:{1}\t", i, Task.CurrentId);
            Thread.Sleep(10);
    },
    new ExecutionDataflowBlockOptions
            { MaxDegreeOfParallelism = 4 });

    for (int i = 1; i <= 10; i++)
    {
            ab.Post(i);
    }
    ab.Complete();
    ab.Completion.Wait();
}
```

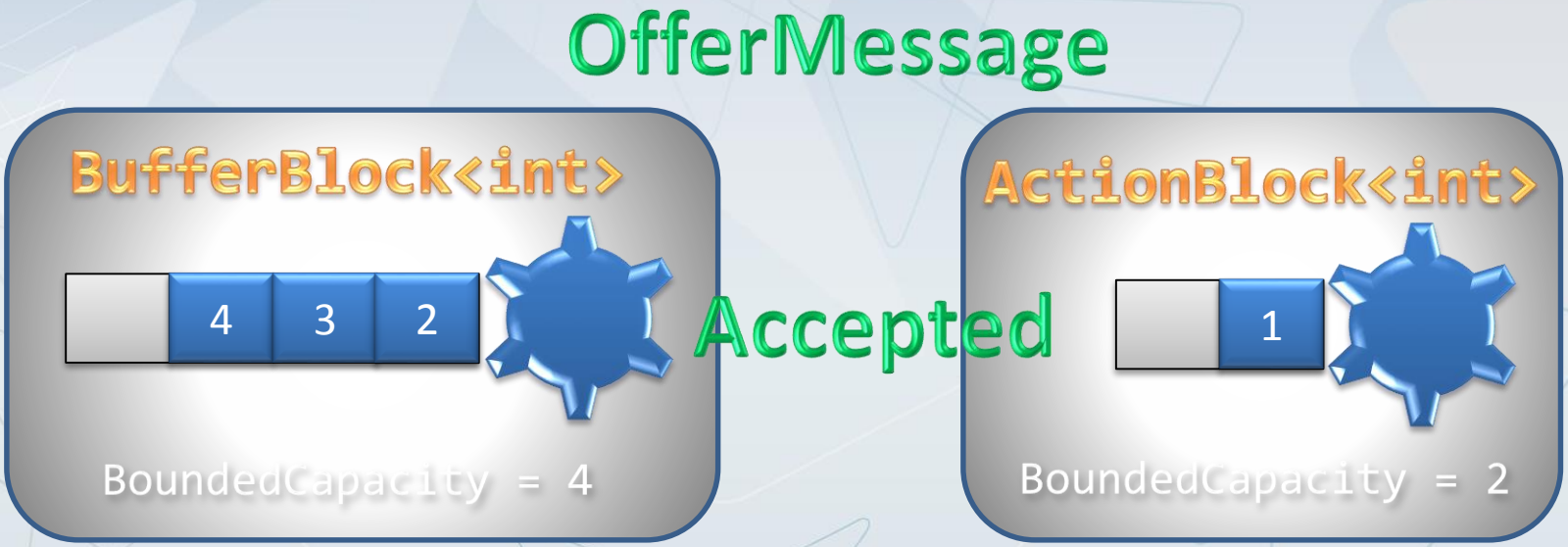
C:\Windows\system32\cmd.exe

```
3) Task Id:1    1) Task Id:2    4) Task Id:3    2) Task Id:4    6) Task Id:1
5) Task Id:2    7) Task Id:3    8) Task Id:4    9) Task Id:1    10) Task Id:3
```

Message Passing - Handshaking



Message Passing - Handshaking

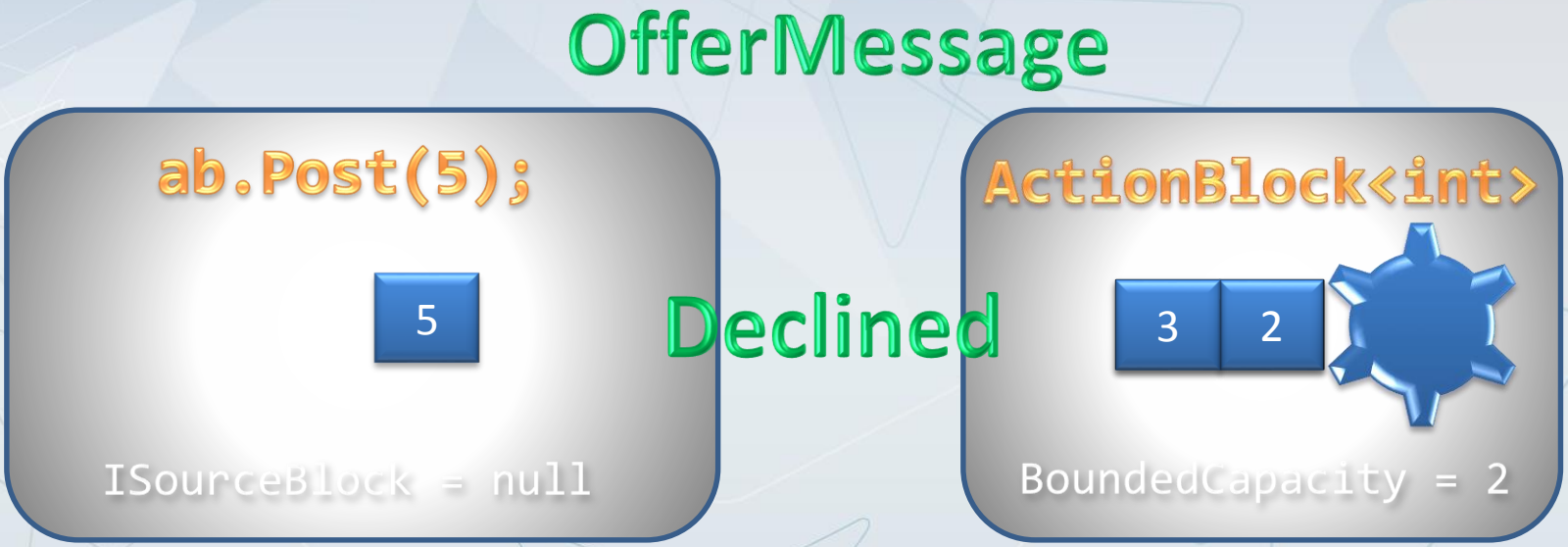


Message Passing - Handshaking

Message Passing - Handshaking

CodeValue College

Message Passing - Handshaking



Message Passing - Handshaking

OfferMessage

ab.Post(6);

6

ISourceBlock = null

ActionBlock<int>

3 2

BoundCapacity = 2

DecliningPermanently

ab.Complete();

CodeValue College

Building the Network

- Building the network is a simple sequence call to:
  - `ISourceBlock<TO>.LinkTo(ITargetBlock<TOutput>, DataflowLinkOptions)`
- Or to one of the DataflowBlock extension methods:
  - `LinkTo<TO>(ISourceBlock<TO>, ITargetBlock<TO>)`
  - `LinkTo<TO>(ISourceBlock<TO>, ITargetBlock<TO>,Predicate<TOutput>)`
  - `LinkTo<TOutput>(ISourceBlock<TOutput>, ITargetBlock<TOutput>, DataflowLinkOptions, Predicate<TOutput>)`

Dataflow Link Options

Name	Description
Append (Links Order)	Determines whether the link should be appended or prepended to the source's list of links
MaxMessages (Link Life Time)	Determines the maximum number of messages that may be consumed across the link
PropagateCompletion (Teardown behavior)	Gets or sets whether the linked target will have completion and faulting notification automatically propagated

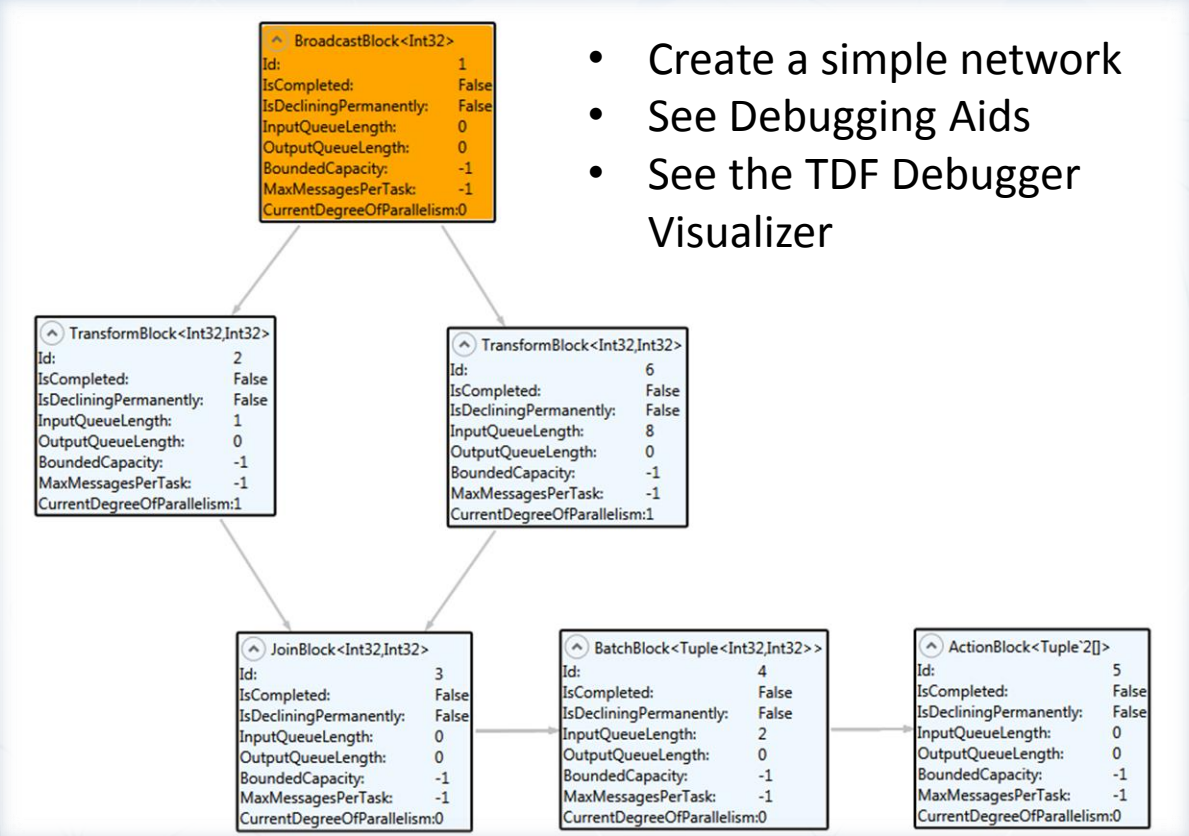
Link Predicate

- You can supply a predicate to the `LinkTo` extension method

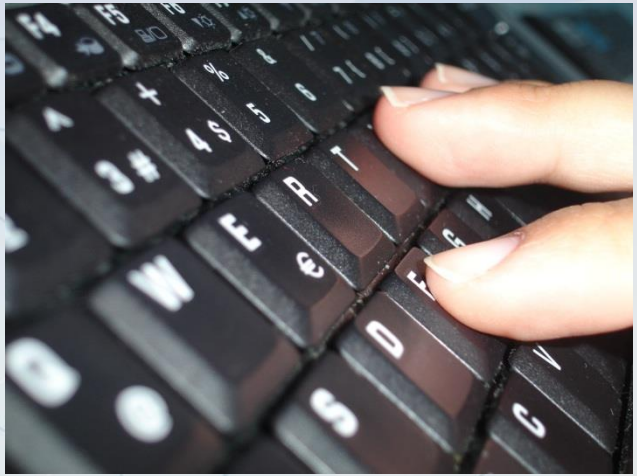
```
src.LinkTo(target, n => n % 2 == 0);
```

  - Internally `LinkTo` creates new private **`FilteredLinkPropagator`** block and inserts it as a filter just before the target block
  - The **`FilteredLinkPropagator`** block executes the predicate for all offered message
    - If the predicate returns true, it offers the message to the target block
    - Otherwise it declines the message

Simple Dataflow Network



- Create a simple network
- See Debugging Aids
- See the TDF Debugger Visualizer



Demo

Network Builder Pattern

- You can build the network in the naïve way
  - Many `LinkTo` statements
- Or you can build the network using a build pattern
  - The Dataflow network is just a graph of objects
  - You can encapsulate the network building knowledge in a builder class
  - You can provide the building rule from the application logic and information file, metadata or use a DSL tool

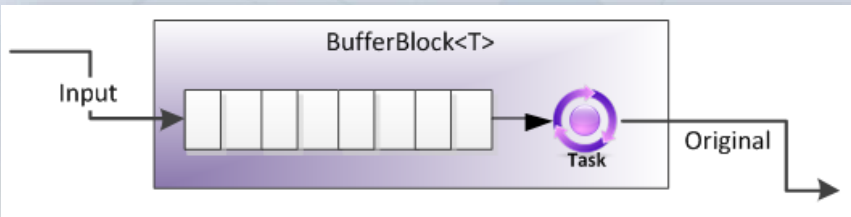
Data Block Options & Categories

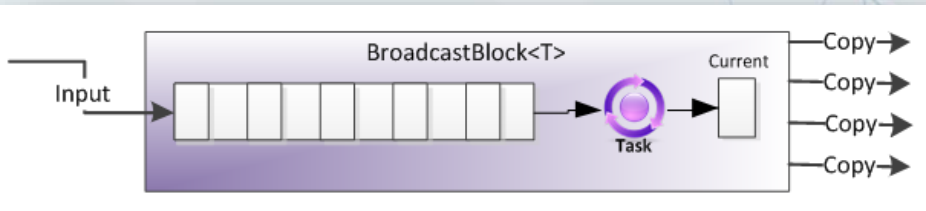
- There are 9 public built-in data blocks in 3 categories:
  - **Buffering**: use to provide a buffer, collecting messages before dispatching them
    - Configured With `DataflowBlockOptions`
      - Capacity limit, Cancelation, Max messages per task, Task Scheduler
  - **Executing**: use to act upon a message
    - Configured With `ExecutionDataflowBlockOptions`
      - `DataflowBlockOptions` + Thread (Task) control
        - » Max degree of parallelism & Single producer constrained
  - **Grouping**: use to batch or join (or both) group of messages
    - Configured With `GroupingDataflowBlockOptions`
      - `DataflowBlockOptions` + Greedy consume messages, Max number of groups

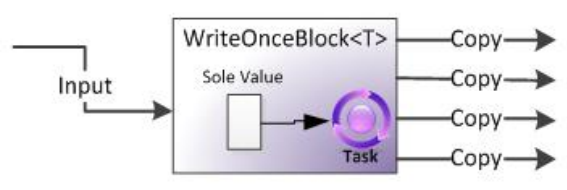
Predefine Blocks

Block	Type	Category	Description
ActionBlock	Target	Execution	Invokes an action for every message
BatchBlock	Propagator	Grouping	Batches input messages into an array
BatchedJoinBlock	Propagator	Grouping	Joins inputs to a batch of tuples
BroadcastBlock	Propagator	Buffering	Stores and distributes a copy of the last message
BufferBlock	Propagator	Buffering	Simple Buffer
JoinBlock	Propagator	Grouping	Form a tuple from a number of inputs
TransformBlock	Propagator	Execution	Transforms an input message to a different output
`TransformManyBlock`	Propagator	Execution	Transform an input message to number of outputs
WriteOnceBlock	Propagator	Buffering	Receiving and storing at most one element

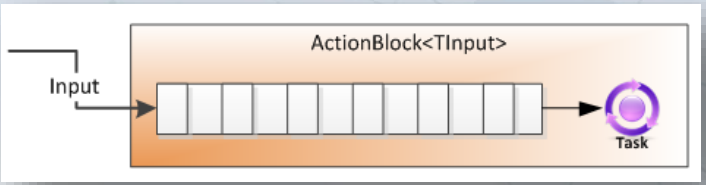
Blocks Internals - Pure Buffering

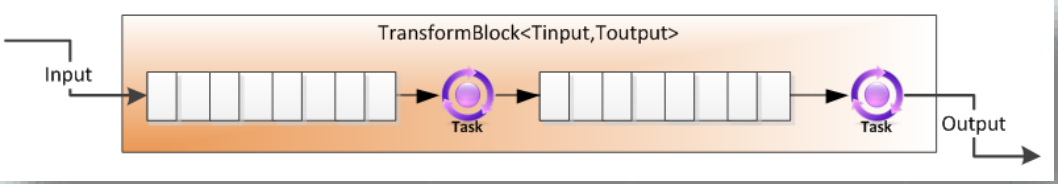


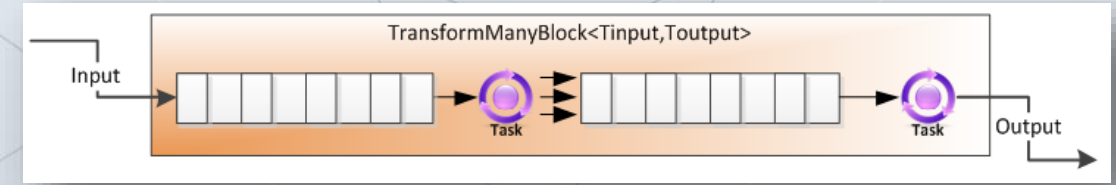




Block Internals – Execution

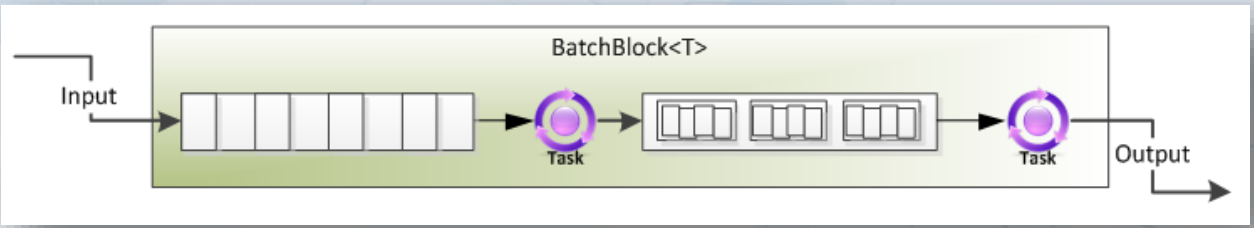


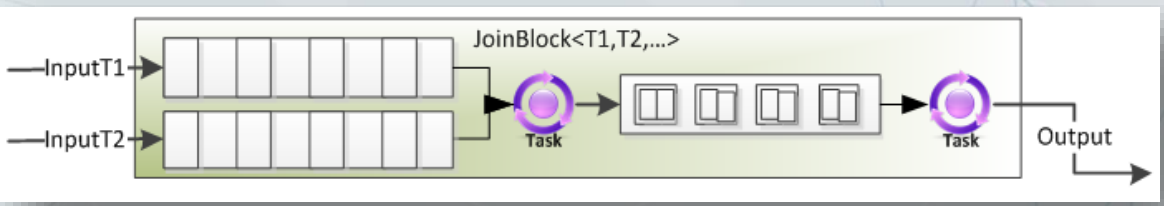


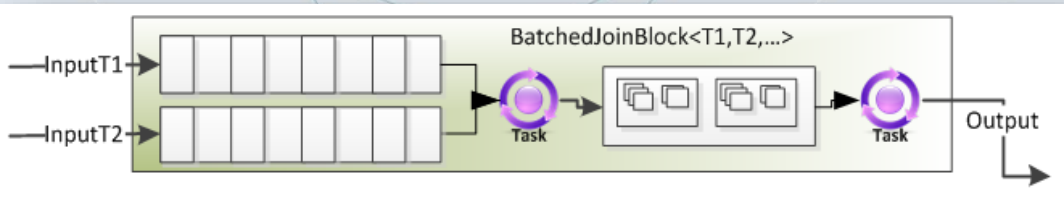


CodeValue College

Block Internals - Grouping







CodeValue College

The DataflowBlock Class

DataflowBlock

Static Class

Methods

- AsObservable<TOutput>
- AsObserver<TInput>
- Choose<T1, T2> (+ 3 overloads)
- Encapsulate<TInput, TOutput>
- LinkTo<TOutput> (+ 2 overloads)
- NullTarget<TInput>
- OutputAvailableAsync<TOutput> (+ 1 overload)
- Post<TInput>
- Receive<TOutput> (+ 3 overloads)
- ReceiveAsync<TOutput> (+ 3 overloads)
- SendAsync<TInput> (+ 1 overload)
- TryReceive<TOutput>

The DataflowBlock Class

- DataflowBlock:
  - OutputAvailableAsync: informs of whether and when more output is available
  - LinkTo: Links the ISourceBlock to the specified ITargetBlock
  - Post: Posts an item to the ITargetBlock
  - Receive: Synchronously receives an item from ISourceBlock
  - TryReceive: Attempts to synchronously receive an item from the ISourceBlock
  - SendAsync: Asynchronously offers a message to ITargetBlock
  - ReceiveAsync: Asynchronously receives a value from ISourceBlock

Sending & Receiving Messages

- **Synchronous Blocking:**

```
block.SendAsync(m).Wait()
m = block.Receive();
```

- **Synchronous non-blocking:**

```
while (!block.post(m)) ;
while (!block.TryReceive(out m)) ;
```

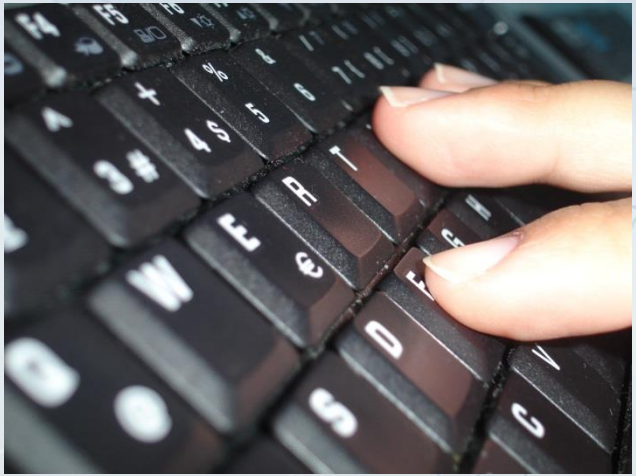
- **Asynchronous:**

```
await block.SendAsync(m);
m = await block.ReceiveAsync();
```

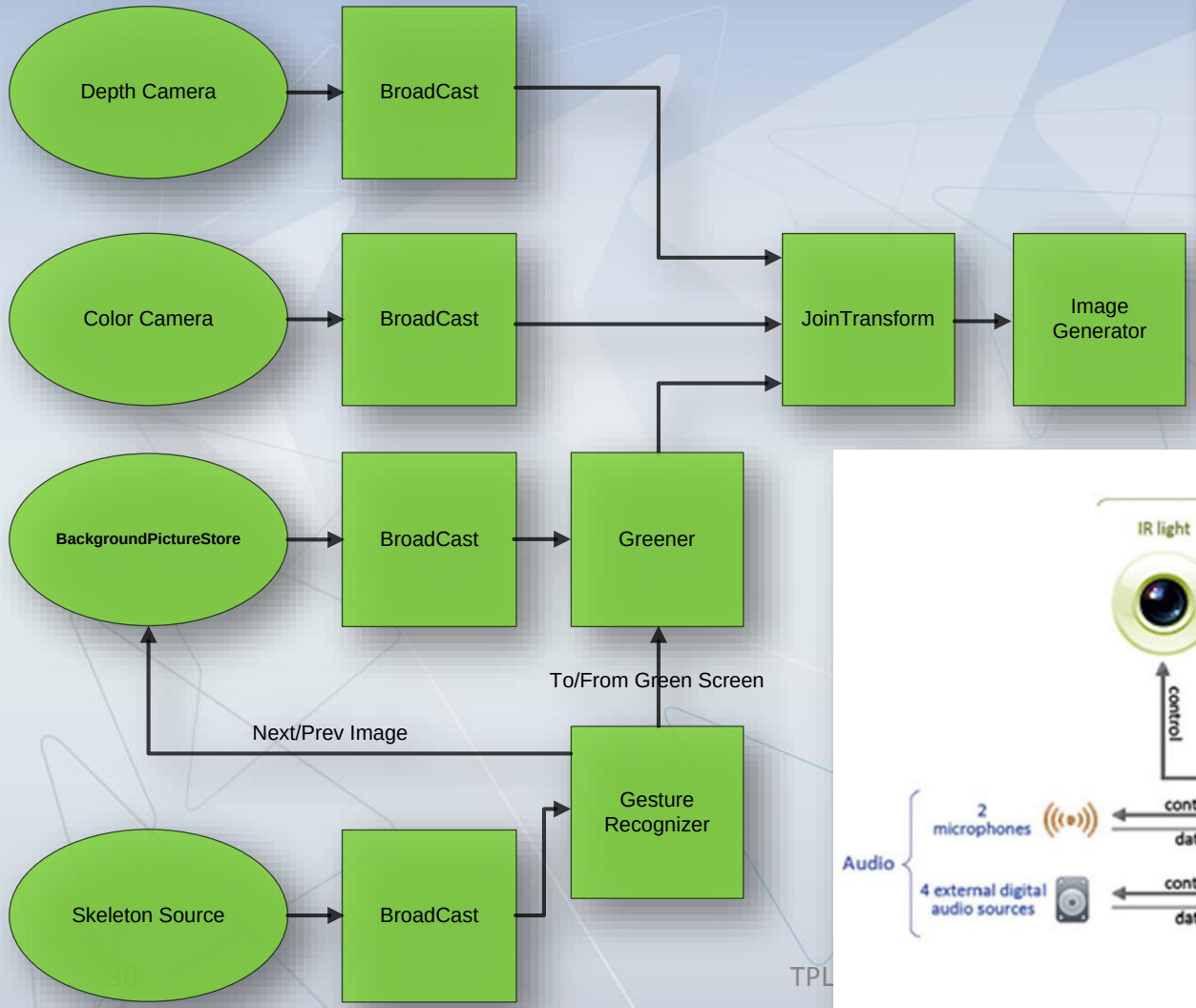
Concurrency Control

- Provide non-default scheduling schema such as priority
  - Specifying the Task Scheduler
- Concurrent execute multiple messages in execution blocks
  - Specifying the Degree of Parallelism
- Provide execution fairness among blocks
  - Specifying the Number of Messages per Task
- Cancel queued tasks
  - Provide TPL Cancelation token
- Remove unnecessary synchronization in execution block
  - Specify the a single producer drive the execution block
- Prevent deadlocks
  - Specify non-greedy behavior in grouping blocks

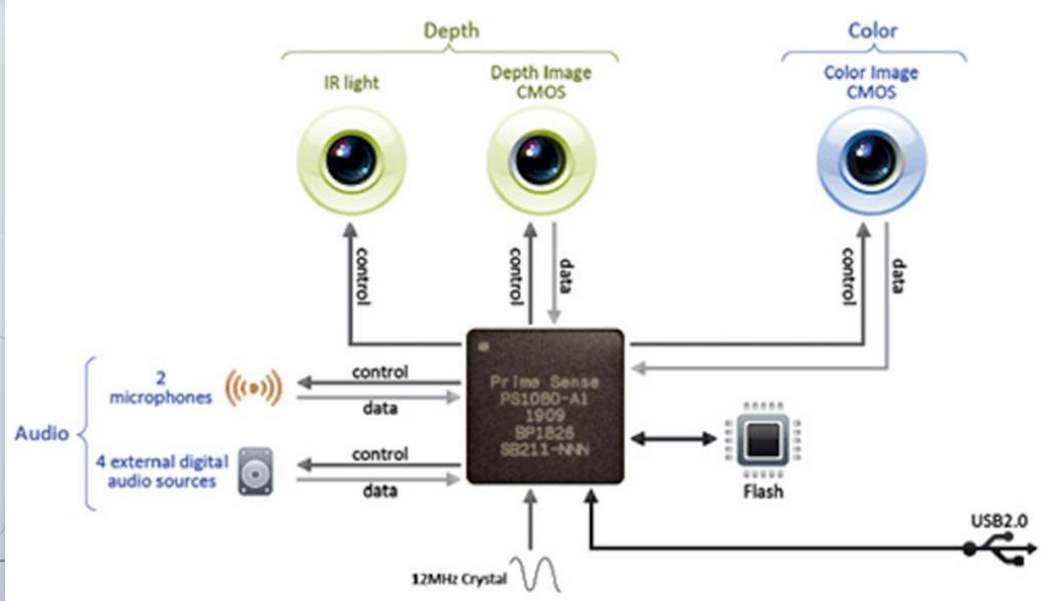
High Performance Low Latency Network



Demo


Depth Camera
BroadCast
Color Camera
BroadCast
JoinTransform
Image Generator
BackgroundPictureStore
BroadCast
Greener
To/From Green Screen
Next/Prev Image
Gesture Recognizer
Skeleton Source
BroadCast


31
KINECT


Depth
Color
IR light
Depth Image CMOS
Color Image CMOS
control
control
data
control
data
Audio
2 microphones
control
data
4 external digital audio sources
control
data
Prime Sense
PS1080-A1
1909
BP1826
SB211-NNN
Flash
USB2.0
12MHz Crystal

Handling Exceptions

- Block can become Faulted
  - Explicitly by calling to Fault
  - By calling a code that throws unhandled exception
  - By receiving a Fault from source block through Fault Propagation
  - By incorrect interface implementation
- Faulted block stop handling messages, clear its queues, fault its completion task and permanently declined messages
- Faulted Source block stop offering messages, propagates the faulted state (if it configured to do so) and unlink itself

Implementing Custom Blocks

- Don't do it if you don't have to
  - Prefer encapsulation with `DataflowBlock.Encapsulate`
  - Or embedding: build your block by implementing TDF interfaces and forwarding the messages to existing blocks
- Build the custom block the hard way by implementing TDF interfaces
- It is not easy:
  - Buffering, Messages Ownership, Synchronization, Block Completion, Non-Greediness, Reserve & Release

Summary

- We want our free lunch back
  - .NET TPL provide the needed parallel abstraction
  - TDF is a new way to architect, design & implement parallel execution following the Actor Model
- Dataflow network is built from dataflow blocks
  - You create the blocks and link them to form high performance low latency high throughput concurrent execution network
- You can customize existing blocks or even build your own

Q & A

Thank You

CodeValue College