Modern patterns of concurrent and parallel programming in .NET

BY RICCARDO TERRELL - @TRIKACE

“I’ve spent several hours on attempts to parallelize my code, but it only get slower when I do so”

- on StackOverflow

“I’ve parallelize my code, and now it runs 4x slower”

- on Twitter

“I asked my team to parallelize the code… NOT to paralyse the code”

- on Twitter


We need a way to make sure our programs continue to improve their speed and performance as hardware evolves. Have any ideas?

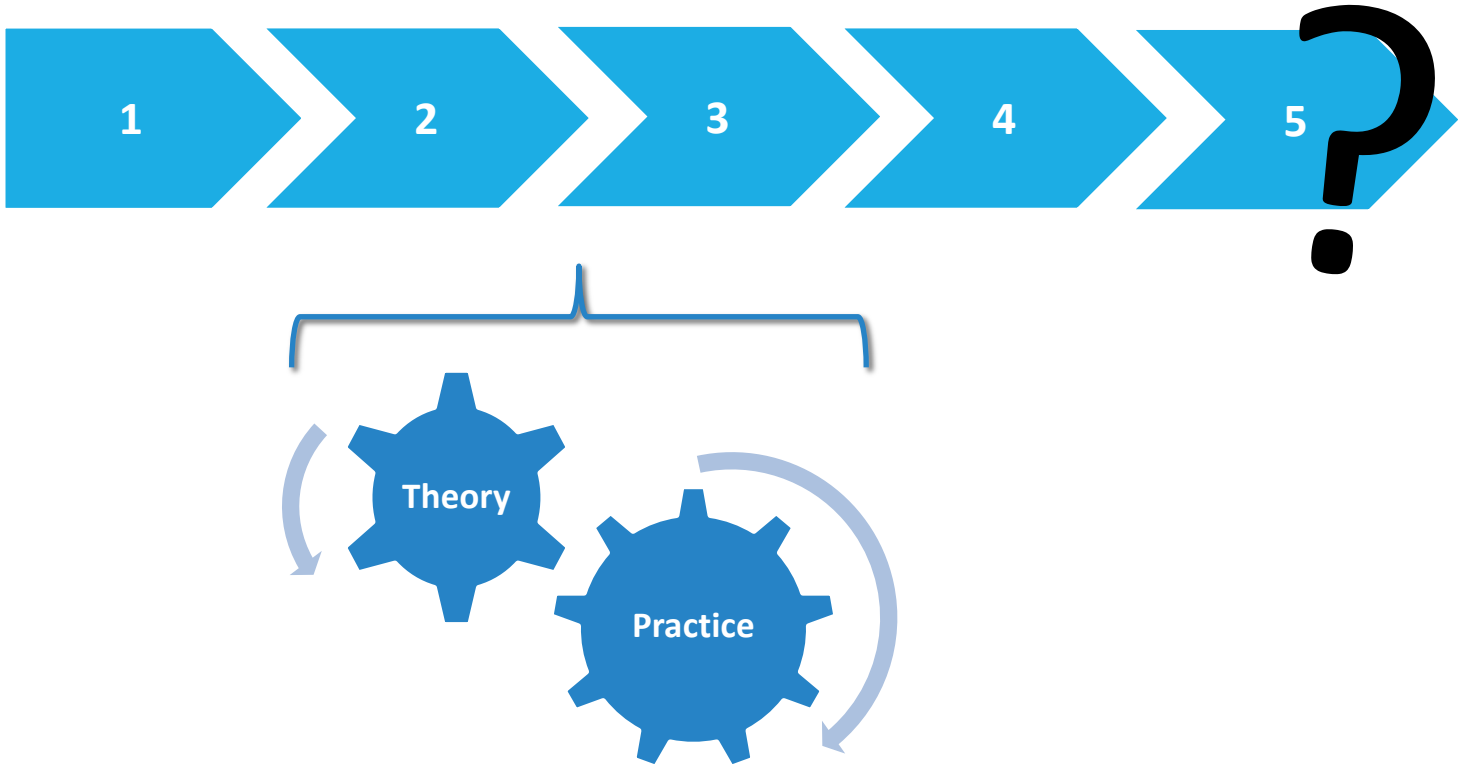
Its about **maximizing** resource use

To get the **best** performance, your application has to partition and divide processing to take full advantage of multicore processors – enabling it to do **multiple** things at the same time, i.e. **concurrently**.

Agenda

- The rise of concurrent programming and slow adoption
- Managed parallelism with Task Parallel Libraries
- Combinators to compose Tasks and Asynchronous operations deterministically
- Message passing programming model

Modules



Pre-requisites

> Windows

- > Visual Studio 2015/2017 Community or
- > Visual Studio Code
  - > C# Extensions
  - > F# Compiler + Ionide package

>Mac

- > Visual Studio for Mac + Mono or
- > Xamarin Studio 6.x + Mono or
- > Visual Studio Code + Mono + Ionide package

>Linux

- > Visual Studio Code + Mono + Ionide package

Download links:
https://github.com/rikace/ncrafts-ws

See README section pre-requisites

Disclaimer

- Let’s keep the session interactive
- Skipping slides
- After this class you will need to keep practicing
- This is just an introduction
- The code is not production-ready
  - You can use either C# or F# or both

Introduction - Riccardo Terrell

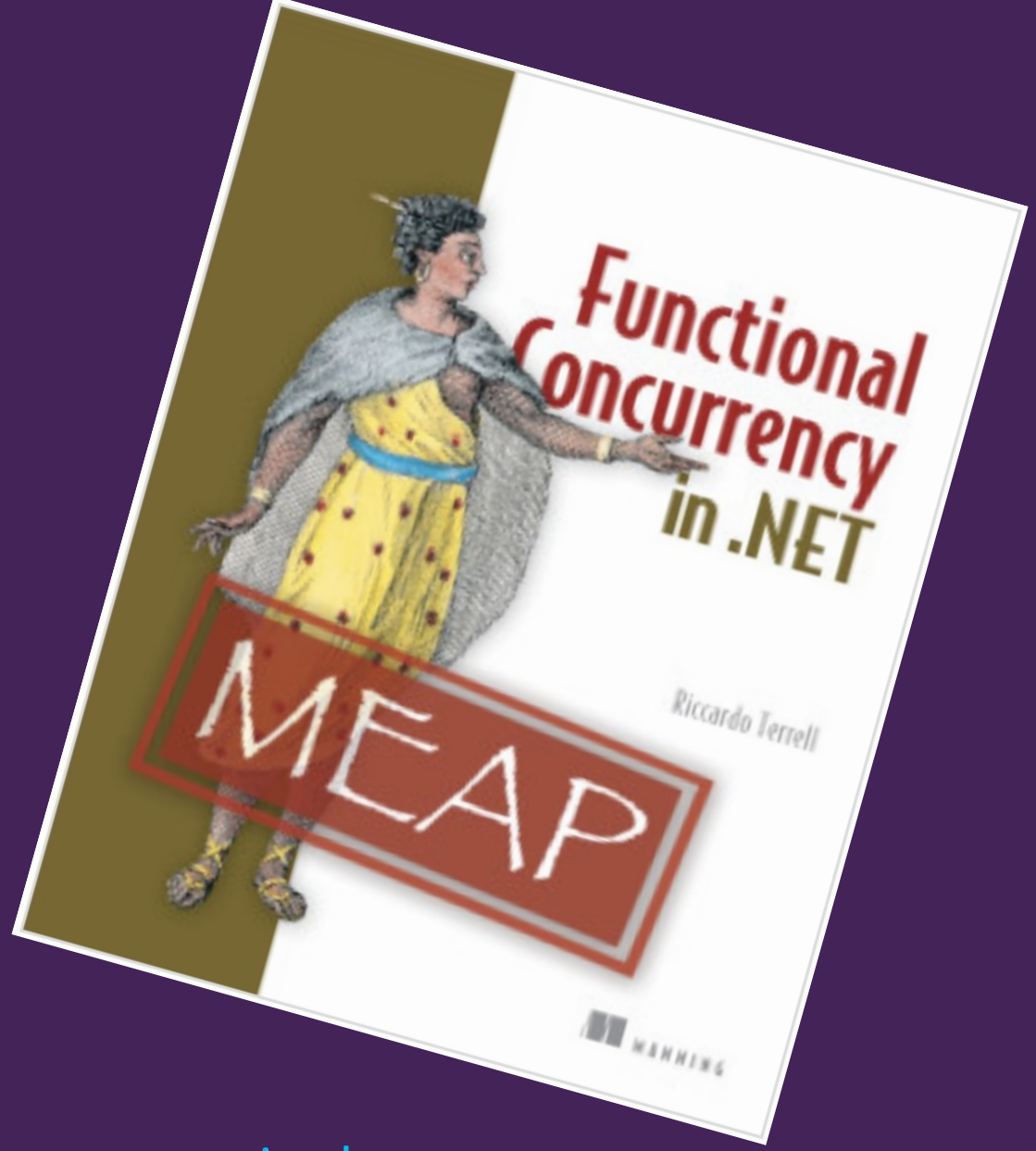
- Originally from Italy, currently - Living/working in Washington DC
- +/- 20 years in professional programming
  - C++/VB → Java → .Net C# → Scala → Haskell → C# & F# → ??
- Author of the book “Functional Concurrency in .NET” – Manning Pub.
- *Polyglot programmer - believes in the art of finding the right tool for the job*
- *Organizer of the DC F# User Group*

@trikace *www.rickyterrell.com* *tericcardo@gmail.com*

MVP
Microsoft®
Most Valuable
Professional

terrellpc

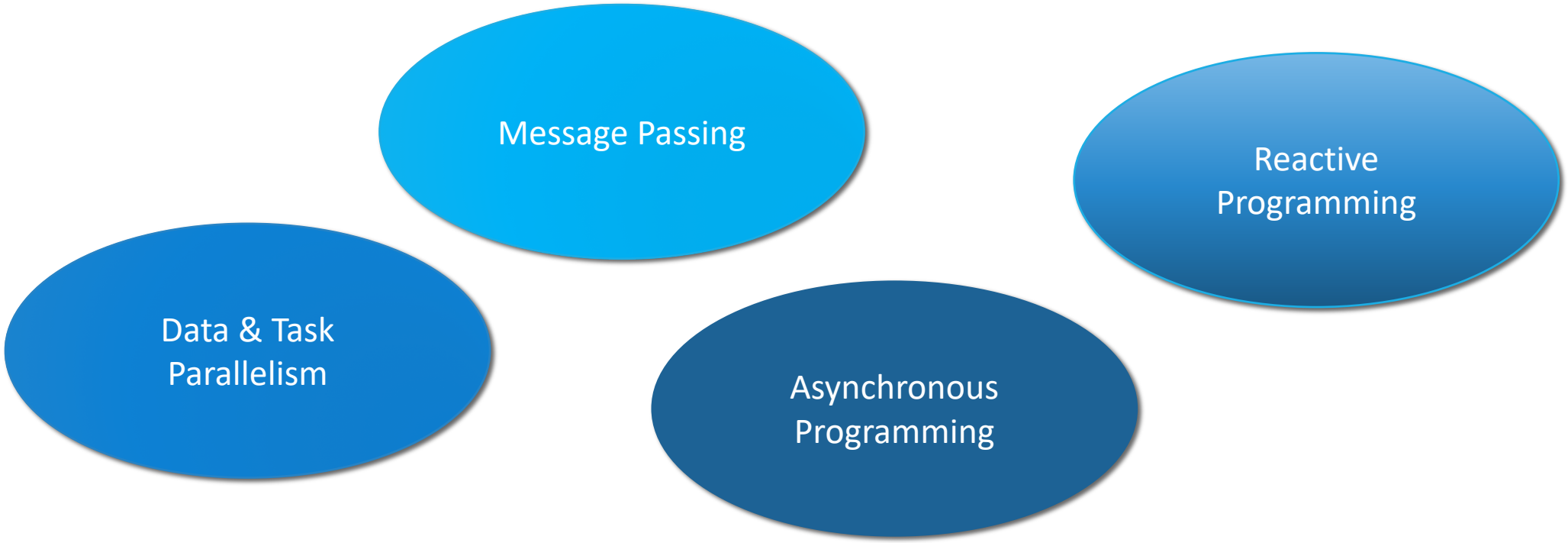
40% off Functional Concurrency in .NET
(all formats)



https://www.manning.com/books/functional-concurrency-in-dotnet


What about you?

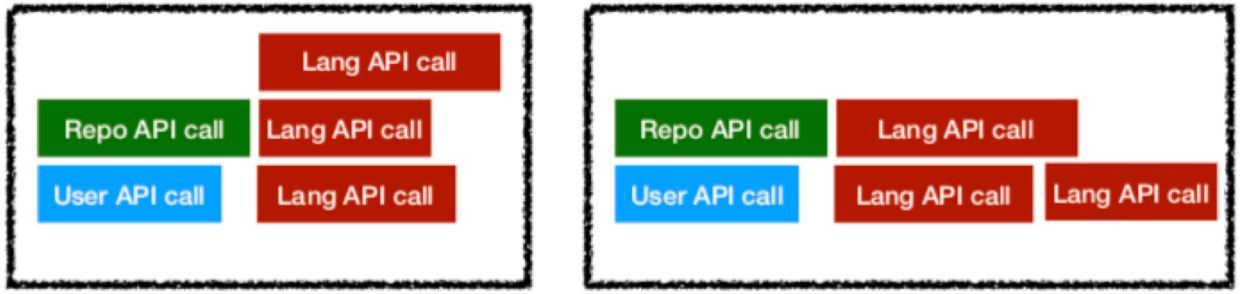
Concurrency Core Concepts




Let’s start!

Concurrency programming

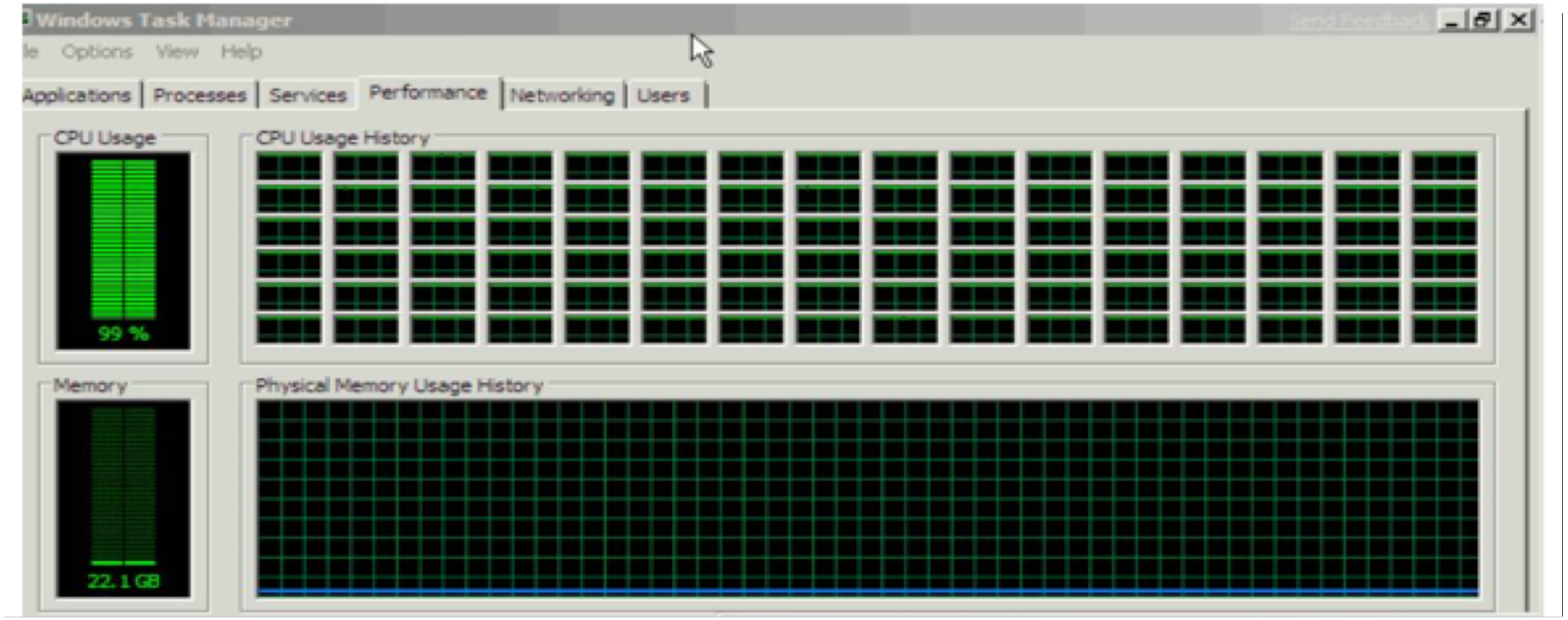
Concurrency provides a way to **structure a solution** to solve a problem that may (but not necessarily) be parallelizable.



Concurrency is the **composition** of independently executing computations.

Moore’s law - The Concurrency challenge

Modern computers come with multiple processors, each equipped with multiple cores, but the single processor is slower than it used to be!



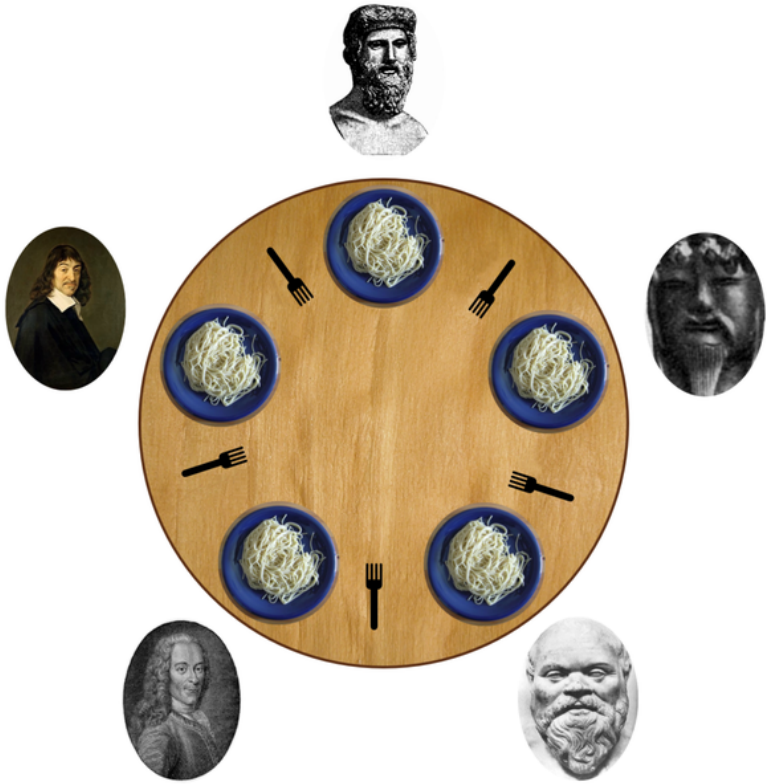

MULTITHREADING
THEORY


PRACTICE

The issue is Shared Memory

- Shared Memory Concurrency
- Data Race / Race Condition
- Works in sequential single threaded environment
- Very difficult to parallelize
- Difficult to maintain and test
- Locking, blocking, call-back hell

Fully Synchronized Objects



Immutability

Like string in C# **everything** is immutable

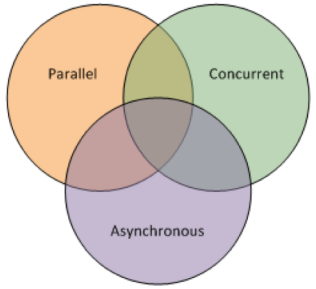
Help in concurrent/parallel code

Immutability is **default** in F#

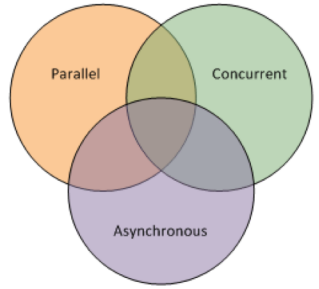
You can choose to go mutable (**use carefully!**)

Copy-on-Write

Immutability




```
public class ImmutablePerson
{
    public ImmutablePerson(string firstName, string lastName, int age) {
        FirstName = firstName;
        LastName = lastName;
        Age = age;
    }
    public string LastName { get; }
    public string FirstName { get; }
    public int Age { get; }

    public ImmutablePerson UpdateAge(int age)
    {
            return new ImmutablePerson(this.FirstName, this.LastName, age);
    }
}
```

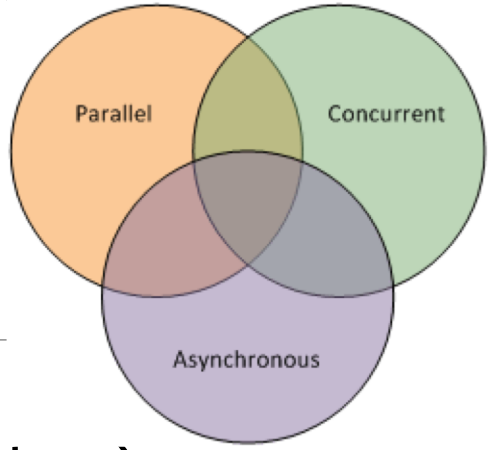


Immutability

```
public class ImmutablePerson
{
    public ImmutablePerson(string firstName, string lastName, int age) {
        FirstName = firstName;
        LastName = lastName;
        Age = age;
    }
    public string LastName { get; }
    public string FirstName { get; }
    public int Age { get; }
   public ImmutablePerson UpdateAge(int age)
   {
           return new ImmutablePerson(this.FirstName, this.LastName, age);
   }
}
```

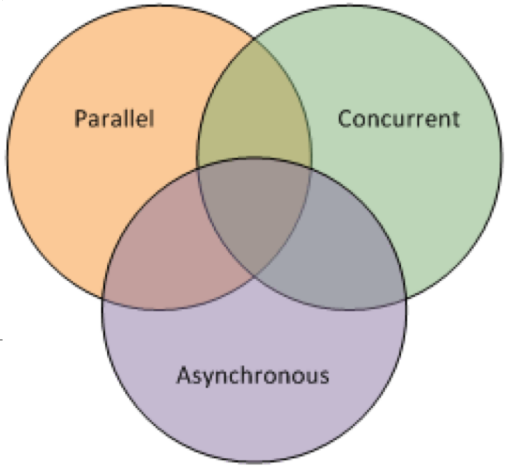
```
type Person = {FirstName:string; LastName:string; Age:int}
```

Compare And Swap - CAS




```
void DoWithCAS<T>(ref T location, Func<T, T> generator)
                                              where T : class
      {
         T temp, replace;
         do
         {
             temp = location;
             replace = generator(temp);
         } while (Interlocked.CompareExchange(ref location,
                                        replace, temp) != temp);
      }
```

Compare And Swap - CAS




```
public class LockFreeStack<T> {
    private class Node {
        public T Data;
        public Node Next;
    }
    private Node head;

    public void Push(T element) {
        Node node = new Node {Data = element};
        DoWithCAS(ref head, h => {
            node.Next = h;
            return node;
        });
    }
```

The Solution is Immutability and Isolation

IMMUTABILITY +
ISOLATION =

BEST CONCURRENT PROGRAMMING MODEL

Message Passing Concurrency (Actors)

Avoid Side Effects

Referential transparency

An expression is said to be referentially transparent if it can be **replaced** with its corresponding value **without changing the program's behavior**.

As a result, evaluating a referentially transparent function gives the same value for same arguments.

In general it means that the function **don’t have side effect** which may influence its result.

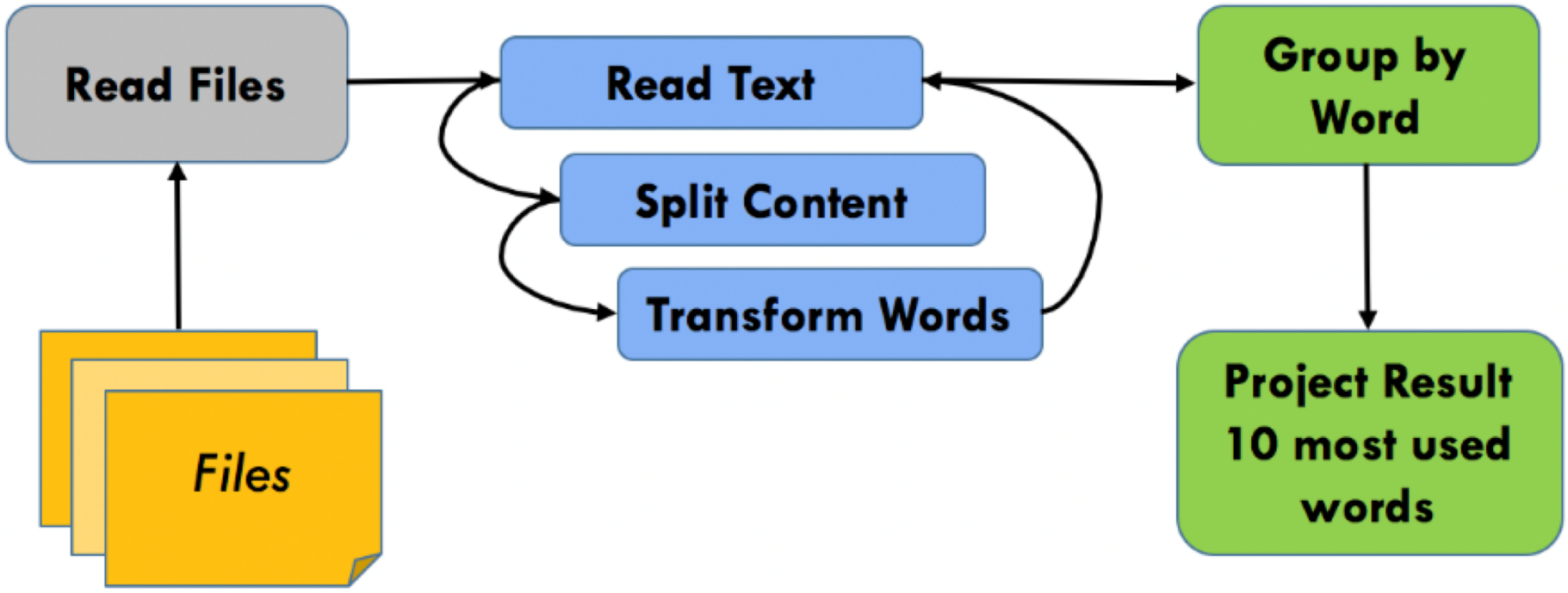
Referential transparency

Which of the following methods are an example of Referential Transparency?

C#

```
DateTime Add(DateTime date, TimeSpen duration) =>
                        date.Add(duration);

DateTime AddDay(DateTime date) =>
                        Add(date, TimeSpan.FromDay(1));

DateTime Tomorrow() => AddDay (DateTime.Now)

DateTime AddTrigger(DateTime date, Func<int> f) =>
                        Add(date, TimeSpan.FromDay(f()));
```

Parallel words counter with side effects



Parallel words counter with side effects

```
public static Dictionary<string, int> WordsCounter(string source)
{
    var wordsCount =
            (from filePath in
                Directory.GetFiles(source, "*.txt").AsParallel()
                from line in File.ReadLines(filePath)
                from word in line.Split(' ')
                select word.ToUpper())
        .GroupBy(w => w)
        .OrderByDescending(v => v.Count()).Take(10); //#C
    return wordsCount.ToDictionary(k => k.Key, v => v.Count());
}
```

Why removing without side effects

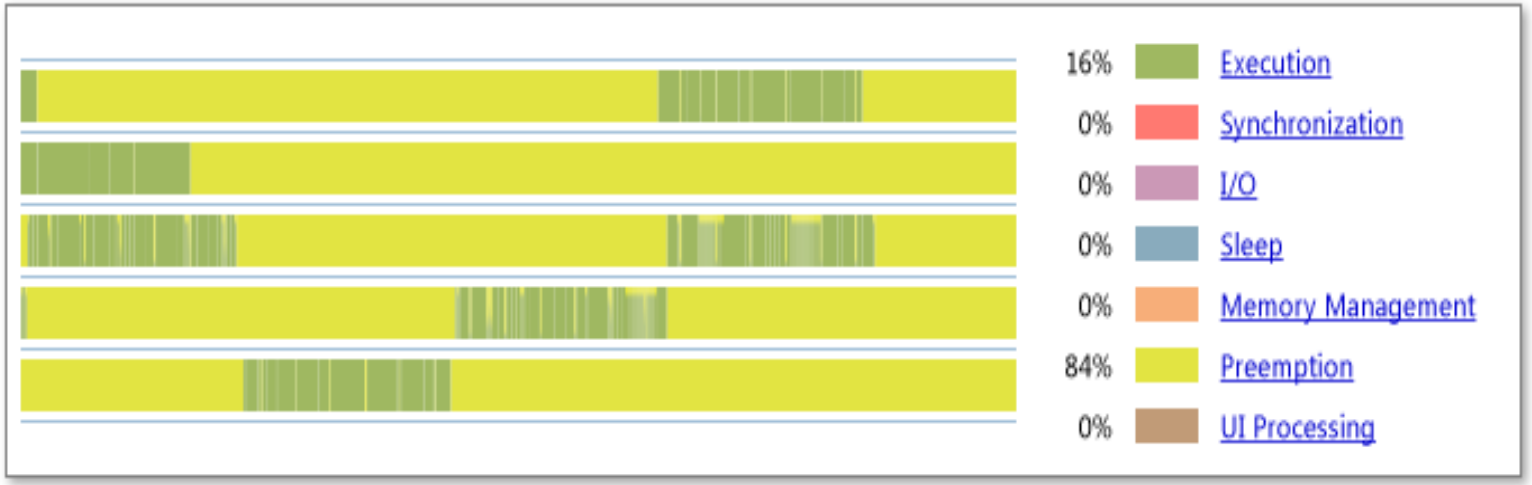
- Easy to reason about the correctness of your program
- Easy to compose functions for creating new behavior
- Easy to be isolated and, therefore, easy to test and less bug prone
- Easy to be executed in parallel; because pure functions don’t have external dependencies, their order of execution (evaluation) doesn’t matter

How to tame parallelism

Quick Sort

```
static void QuickSort_Parallel<T>(T[] items, int left, int right)
{
    if (right - left < 2)
    {
        if (left+1 == right &&
            items[left].CompareTo(items[right]) > 0)
            Swap(ref items[left], ref items[right]);
        return;
    }

    int pivot = Partition(items, left, right);

    Task leftTask = Task.Run(() => QuickSort_Parallel(items, left, pivot));
    Task rightTask = Task.Run(() => QuickSort_Parallel(items, pivot + 1, right));

    Task.WaitAll(leftTask, rightTask);
}
```

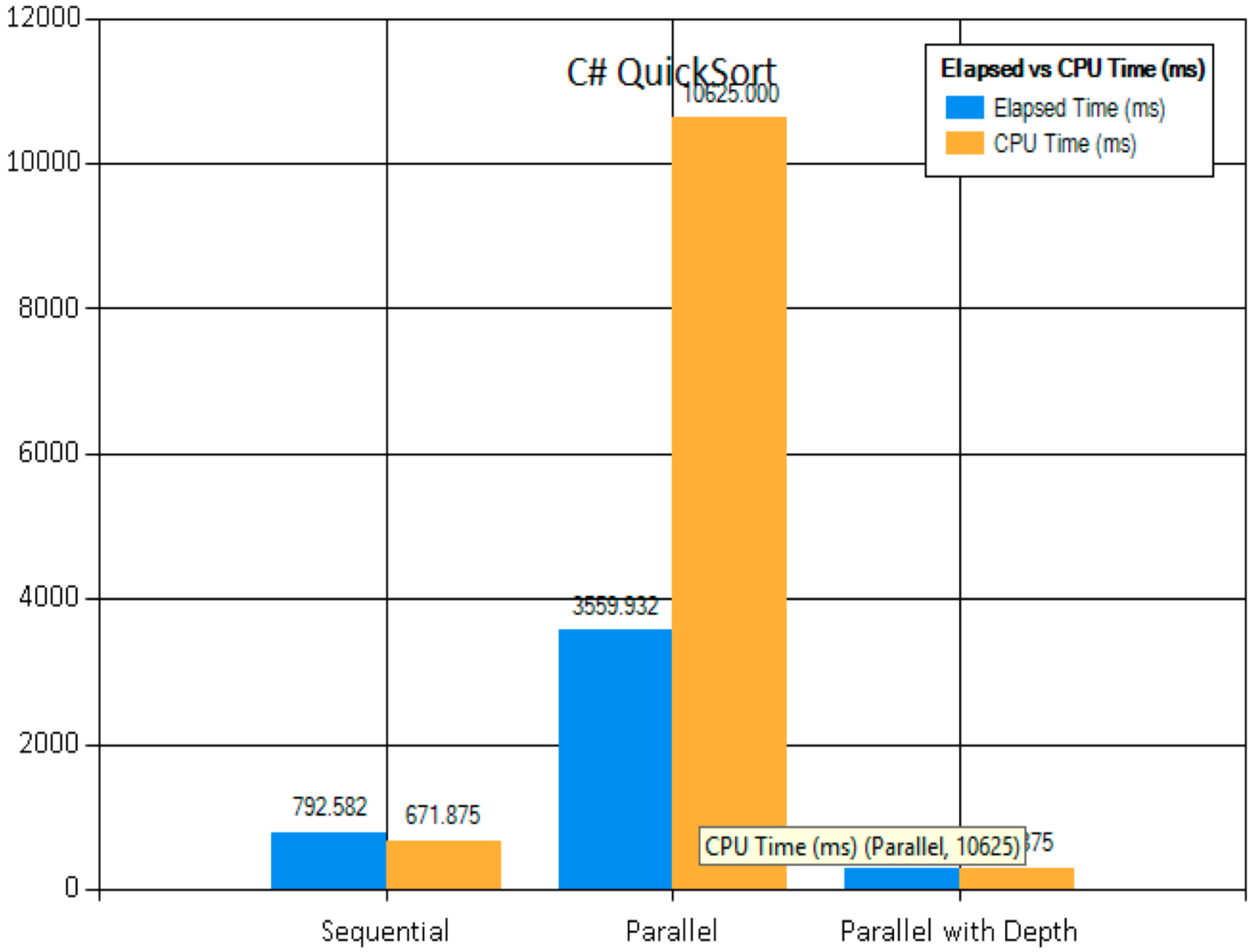
Quick Sort – Oversubscription



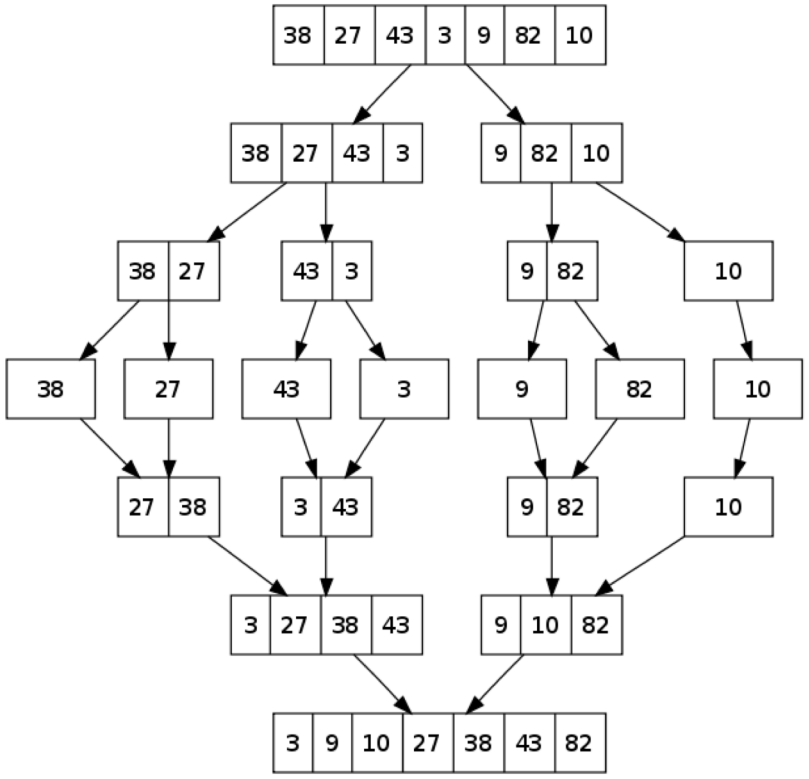
Quick Sort – with depth

```
void QuickSort_Parallel_Threshold<T>(T[] items, int left, int right, int depth)
{
    if (left >= right)  return;
    SwapElements(items, left, (left + right) / 2);
    var pivot = Partition(items);
    if (depth >= 0)     {
        QuickSort_Parallel_Threshold(items, left, pivot - 1, depth);
        QuickSort_Parallel_Threshold(items, pivot + 1, right, depth);
    }
    else
    {
        --depth;
        Parallel.Invoke(
                () => QuickSort_Parallel_Threshold(items, left, pivot - 1, depth),
                () => QuickSort_Parallel_Threshold(items, pivot + 1, right, depth)
            );
    }
```

Quick Sort benchmark



Divide and Conquer

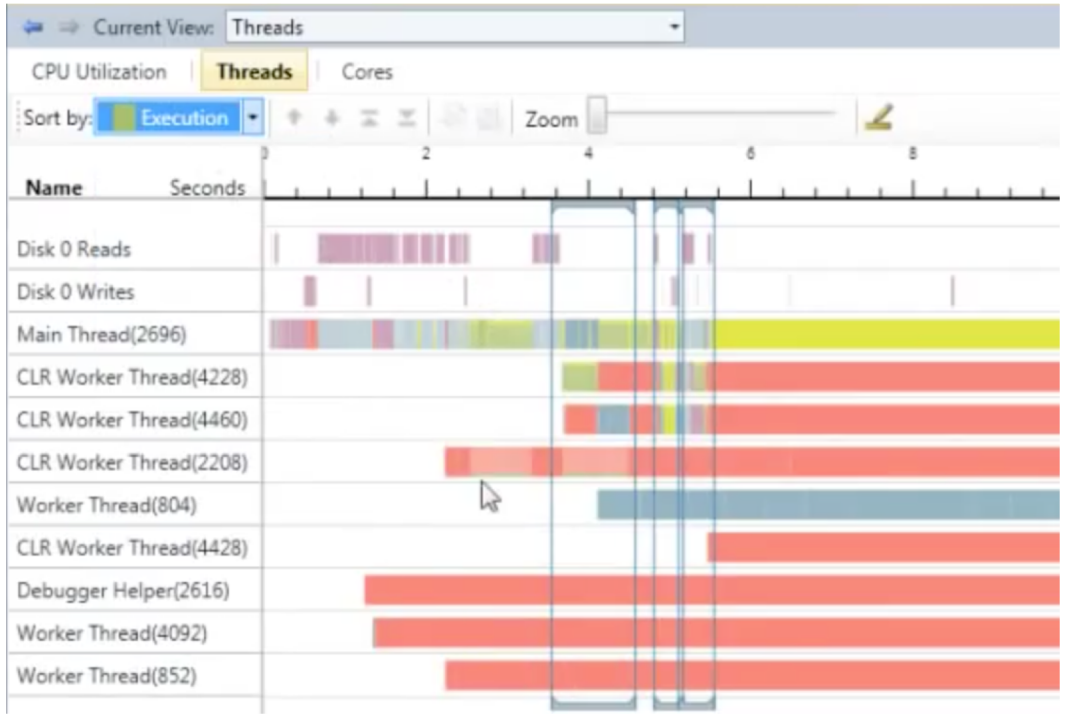



```
void QuickSort_Parallel<T>(T[] items, int left, int right)
{
    int pivot = left;


    SwapElements(items, left, pivot);

    Parallel.Invoke(
            () => QuickSort_Parallel(items, left, pivot - 1),
            () => QuickSort_Parallel(items, pivot + 1, right)
        );
}
```

Measure twice and cut once

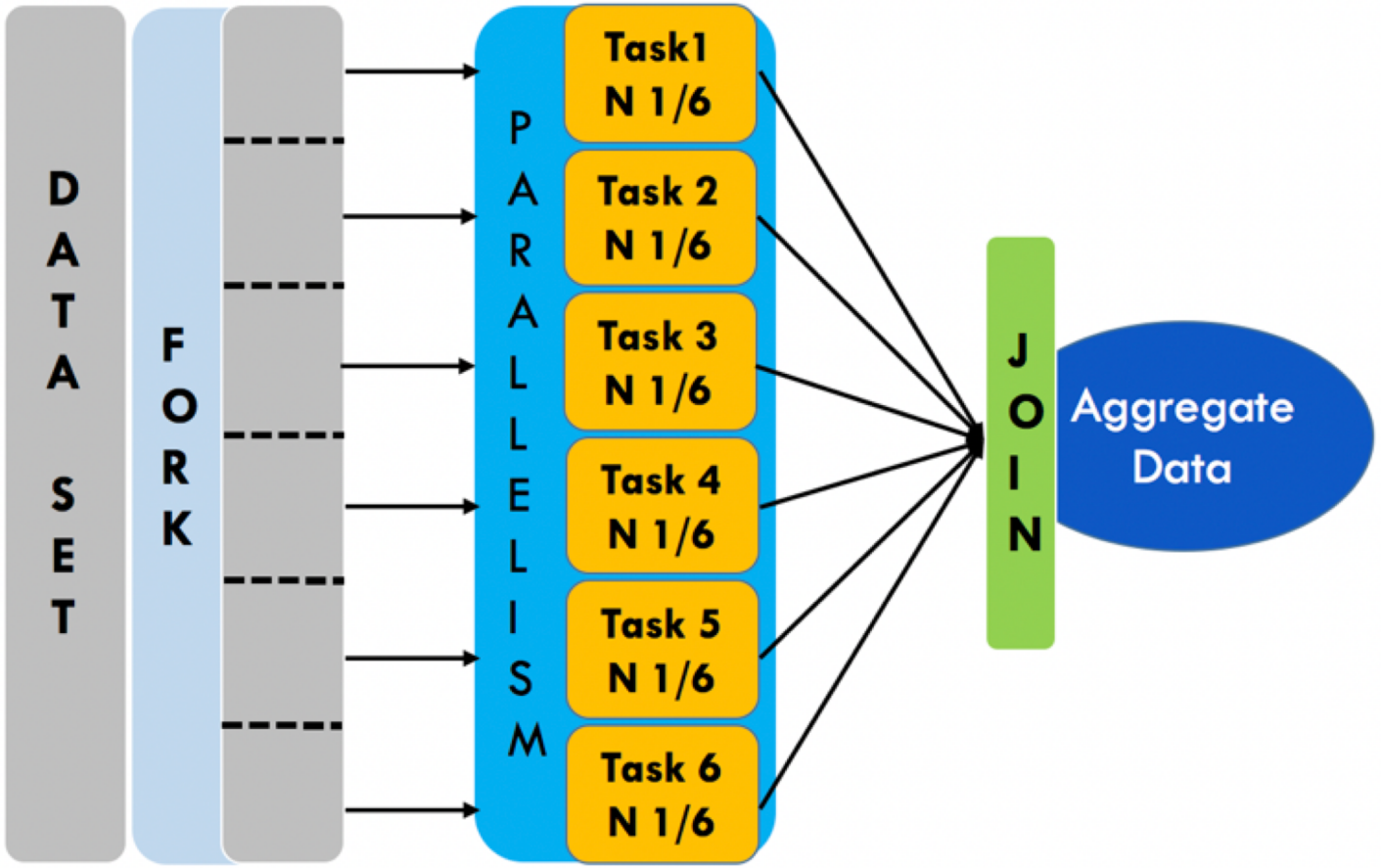
Assume nothing – Profile everything

- Be scientific
  - Do test multiple implementations
  - Don’t set out to confirm your bias
  - Instrument and profile your code



Task parallel programming in action

The Fork/Join pattern



Sequential Fuzzy Match

```
List<string> matches = new List<string>();
foreach (var word in WordsToSearch)
{
    var localMathes = JaroWinklerModule.bestMatch(Words, word);
    matches.AddRange(localMathes.Select(m => m.Word));
}
```

Fuzzy match 7 words against 13.4 Mb of text

Time execution in 4 Logical cores – 6 Gb Ram : **23,167** ms

Multi Thread Fuzzy Match

```
List<string> matches = new List<string>();
var threads = new Thread[Environment.ProcessorCount];

for (int i = 0; i < threads.Length; i++) {
      var index = i;
      threads[index] = new Thread(() =>   {

      var take = WordsToSearch.Count / (Math.Min(WordsToSearch.Count, threads.Length));
      var start = index == threads.Length - 1 ? WordsToSearch.Count - take : index * take;


      foreach (var word in WordsToSearch.Skip(start).Take(take)) {
              var localMathes = FuzzyMatch.JaroWinklerModule.bestMatch(Words, word);
              lock (matches)
                      matches.AddRange(localMathes.Select(m => m.Word));
                    }
                });
            }

 for (int i = 0; i < threads.Length; i++)
    threads[i].Start();
for (int i = 0; i < threads.Length; i++)
   threads[i].Join();
```

Time execution : **7,731** ms

Parallel Loop Fuzzy Match

```
 List<string> matches = new List<string>();
object sync = new object();

Parallel.ForEach(WordsToSearch,
                 // thread local initializer
                 () => { return new List<string>(); },
                 (word, loopState, localMatches) =>  {
                      var localMathes = FuzzyMatch.JaroWinklerModule.bestMatch(Words, word);
                        localMatches.AddRange(localMathes.Select(m => m.Word));// same code
                        return localMatches;
                 },
    (finalResult) =>
    {
        // thread local aggregator
        lock (sync) matches.AddRange(finalResult);
    }
);
```

Time execution : **7,429** ms

Parallel Loop … no for I/O

```
Task ForEachAsync<T>(this IEnumerable<T> source, int dop,
                                                Func<T, Task> body)
{
    return Task.WhenAll(
        from partition in Partitioner.Create(source).GetPartitions(dop)
        select Task.Run(async delegate {
            using (partition)
                while (partition.MoveNext())
                    await body(partition.Current);
        }));
}
```

PLINQ Fuzzy Match

```
ParallelQuery<string> matches =
        (from word in WordsToSearch.AsParallel()
         from match in JaroWinklerModule.bestMatch(Words, word)
         select match.Word);
```

Time execution : **6,347** ms

PLINQ options

```
var parallelQuery = from t in source.AsParallel()
    select t;

var cts = new CancellationTokenSource();
cts.CancelAfter(TimeSpan.FromSeconds(3));

    parallelQuery
        .WithDegreeOfParallelism(Environment.ProcessorCount)
        .WithExecutionMode(ParallelExecutionMode.ForceParallelism)
        .WithMergeOptions(ParallelMergeOptions.Default)
        .WithCancellation(cts.Token)
        .ForAll(Console.WriteLine);
```

Partitioning

Partitioning in PLINQ

Range Partitioning

Chunk Partitioning

Striped Partitioning

Hash Partitioning

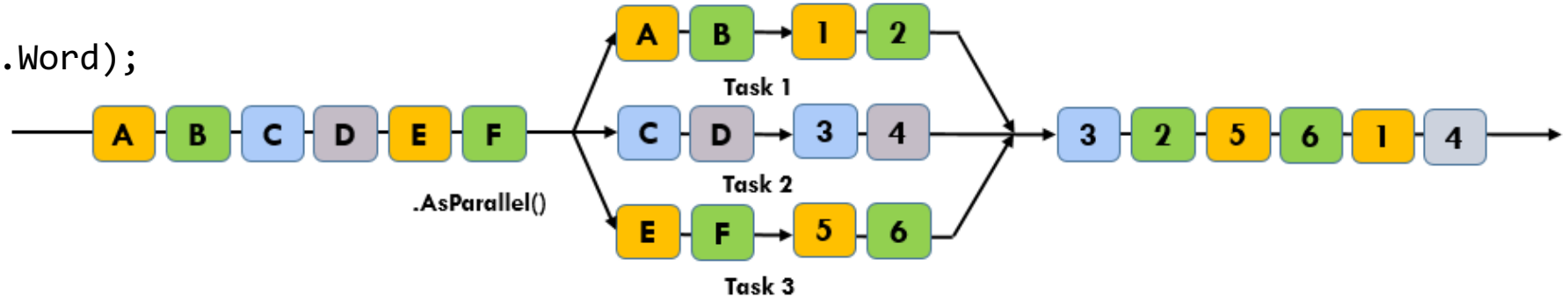
PLINQ Fuzzy Match

```
Create a load-balancing partitioner, specify false for static partitioning.

Partitioner<string> partitioner = Partitioner.Create(WordsToSearch, false);

ParallelQuery<string> matches =

        (from word in partitioner.AsParallel()

         from match in JaroWinklerModule.bestMatch(Words, word)

         select match.Word);
```

Time execution : **4,217** ms

PLINQ Fuzzy Match

```
Create a load-balancing partitioner, specify false for static partitioning.

Partitioner<string> partitioner = Partitioner.Create(WordsToSearch, false);

ParallelQuery<string> matches =

        (from word in partitioner.AsParallel()

         from match in JaroWinklerModule.bestMatch(Words, word)

         select match.Word);
```



Time execution : **4,217** ms

Lab – Faster Fuzzy Match

Composing Tasks

Task continuation

Think Continuations, not Callbacks

Task continuation

```
void RunContinuation<T>(Func<T> input, Action<T> rest)
        => ThreadPool.QueueUserWorkItem(new WaitCallback(o => rest(input())));


Action<string> printMsg = msg =>
    Console.WriteLine($"ThreadID = {Thread.CurrentThread.ManagedThreadId}, Url = {url}, {msg}");

RunContinuation(() => {
    printMsg("Creating webclient...");
    return new System.Net.WebClient();
}, (webclient) => RunContinuation(() => {
        printMsg("Downloading url...");
        return webclient.DownloadString(url);
    }, (html) => RunContinuation(() => {
            printMsg("Extracting urls...");
            return Regex.Matches(html, @"http://\S+");
        }, (matches) =>
                printMsg("Found " + matches.Count.ToString() + " links")
            )));
```

Task continuation

```
ThreadID = 19, Url = http://www.google.com/, Creating webclient...
ThreadID = 21, Url = http://microsoft.com/, Creating webclient...
ThreadID = 16, Url = http://www.wordpress.com/, Creating webclient...
ThreadID = 16, Url = http://www.wordpress.com/, Downloading url...
ThreadID = 19, Url = http://www.google.com/, Downloading url...
ThreadID = 21, Url = http://microsoft.com/, Downloading url...
ThreadID = 16, Url = http://www.wordpress.com/, Extracting urls...
ThreadID = 16, Url = http://www.wordpress.com/, Found 15 links
ThreadID = 16, Url = http://www.peta.org, Creating webclient...
ThreadID = 16, Url = http://www.peta.org, Downloading url...
ThreadID = 19, Url = http://www.google.com/, Extracting urls...
ThreadID = 19, Url = http://www.google.com/, Found 14 links
ThreadID = 20, Url = http://www.peta.org, Extracting urls...
ThreadID = 20, Url = http://www.peta.org, Found 60 links
ThreadID = 21, Url = http://microsoft.com/, Extracting urls...
ThreadID = 21, Url = http://microsoft.com/, Found 1 links
```

Task continuation

```
Task.Run(() =>
{
    printMsg("Creating webclient...");
    return new System.Net.WebClient();
}).ContinueWith(taskWebClient =>
{
    var webclient = taskWebClient.Result;
    printMsg("Downloading url...");
    return webclient.DownloadString(url);
}).ContinueWith(htmlTask =>
{
    var html = htmlTask.Result;
    printMsg("Extracting urls...");
    return Regex.Matches(html, @"http://\S+");
}).ContinueWith(rgxTask =>
{
    var matches = rgxTask.Result;
    printMsg("Found " + matches.Count.ToString() + " links");
});
```

Continuation Passing Style

```
void GetMaxCPS(int x, int y, Action<int> f)
{
    if (x > y)
        f(x);
    else
        f(y);
}
```

Task continuation

```
var matches = from webclient in Task.Run(() => new System.Net.WebClient())
              from html in Task.Run(() => webclient.DownloadString(url))
              from rgx in Task.Run(() => Regex.Matches(html, @"http://\S+"))
              select rgx;
```

Mathematical patterns for better composition

Task is a Monad

How can we compose Tasks?

C#

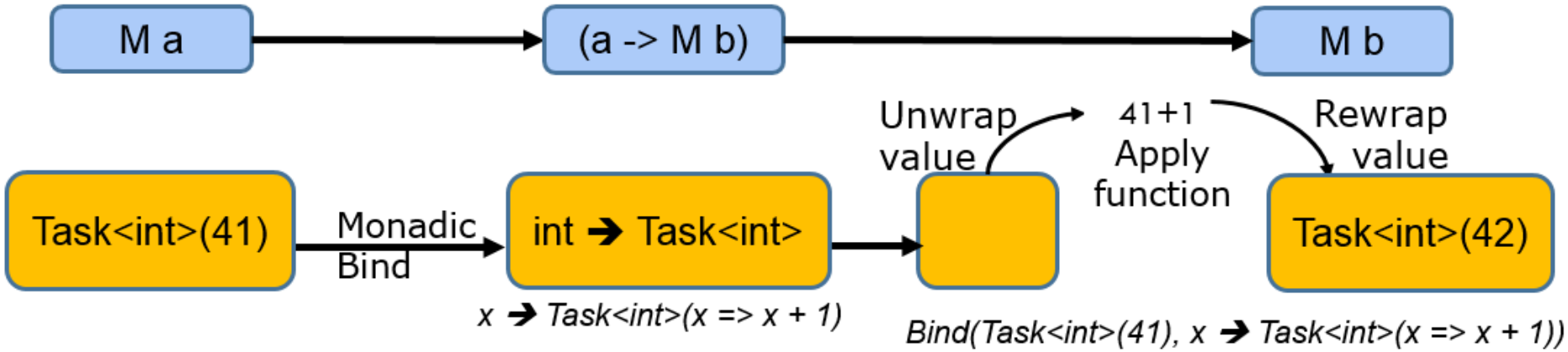
```
Task<int> RunOne() => // ...
Task<decimal> RunTwo(int input) => // ...

var result = RunTwo(RunOne()); // Error!!
```

C#

```
Task<string> RunOne(int input) =>  // ...
Task<bool> RunTwo(string input) => // ...

var result = RunOne(42).Compose(RunTwo); // Error!!
```

Composing Tasks

```
Task<R> Bind<T, R>(this Task<T> m, Func<T, Task<R>> k) …

Task<T> Return(T value) …
```

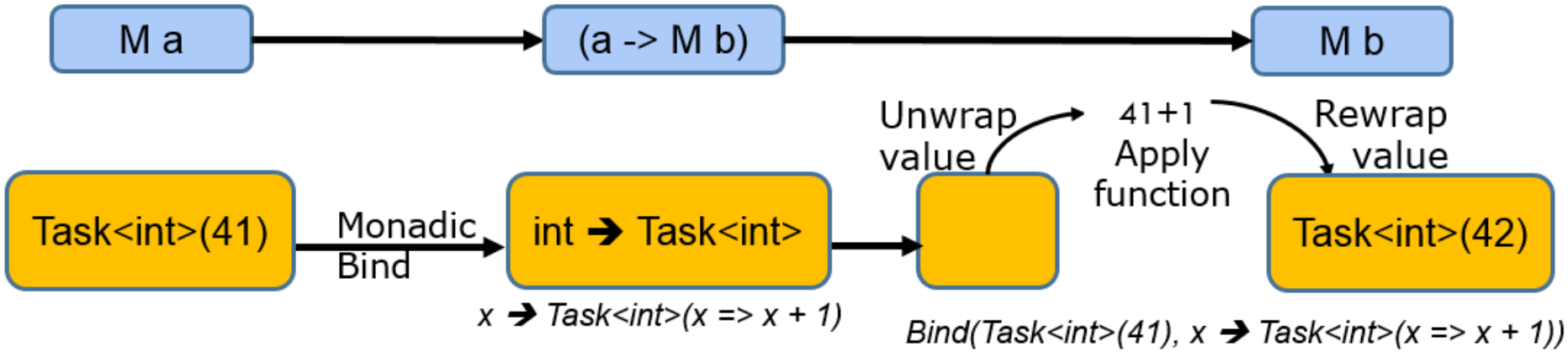


Functional Design Patterns – Monad

Generic type

With function (>>=): bind : M<‘a> -> (‘a -> M<‘b>) -> M<‘b>

With function: return : ‘a -> M<‘a>



Task Bind and Task Return

```
// 'T -> M<'T>
static member Return (value : 'T) : Task<'T>

 // M<'T> * ('T -> M<'U>) -> M<'U>
static member Bind (input : Task<'T>, binder : 'T -> Task<'U>) : Task<'U>
```

```
 // M<'T> * ('T -> M<'U>) -> M<'U>
static member SelectMany (input : Task<'T>, binder : 'T -> Task<'U>) : Task<'U>
```

SelectMany

```
// M<T> * (T -> M<U>) -> M<U>
IEnumerable<R> SelectMany<T, R>( this IEnumerable<T> source,
                                 Func<T, IEnumerable<R>> selector )

// M<T> * (T -> M<U>) -> (T -> M -> U) -> M<U>
IEnumerable<R> SelectMany<T, M, R>( this IEnumerable<T> source,
              Func<T, IEnumerable<M>> collectionSelector, Func<T, M, R> selector )

// M<T> * (T -> M<U>) -> M<U>
IEnumerable<R> Select<T, R>( this IEnumerable<T> source, Func<T, R> selector )
```

The hidden Functor – Map

Map : (T -> R) -> [T] -> [R]

```
Select(this IEnumerable<T> task, Func<T , R> map) …

Select(this Task<T> task, Func<T , R> map) …
```

Composing Tasks

```
from image in Task.Run<Emgu.CV.Image<Bgr, byte>()
from imageFrame in Task.Run<Emgu.CV.Image<Gray, byte>>()
from faces in Task.Run<System.Drawing.Rectangle[]>()
select faces;
```

LAB : SelectMany Tasks
&
Run Tasks as they complete

Task Bind and Task Return

```
// 'T -> M<'T>
static member Return value : Task<'T> = Task.FromResult<'T> (value)

 // M<'T> * ('T -> M<'U>) -> M<'U>
static member Bind (input : Task<'T>, binder : 'T -> Task<'U>) =
      let tcs = new TaskCompletionSource<'U>()
      input.ContinueWith(fun (task:Task<'T>) ->
         if (task.IsFaulted) then
             tcs.SetException(task.Exception.InnerExceptions)
         elif (task.IsCanceled) then tcs.SetCanceled()
         else
              try
                 (binder(task.Result)).ContinueWith(fun
(nextTask:Task<'U>) -> tcs.SetResult(nextTask.Result)) |> ignore
              with
              | ex -> tcs.SetException(ex)) |> ignore
      tcs.Task
```

Task SelectMany

```
 // M<'T> * ('T -> M<'U>) -> M<'U>
 static member SelectMany(input : Task<'T>, binder : 'T -> Task<'U>) =
       let tcs = new TaskCompletionSource<'U>()
       input.ContinueWith(fun (task:Task<'T>) ->
          if (task.IsFaulted) then
            tcs.SetException(task.Exception.InnerExceptions)
          elif (task.IsCanceled) then tcs.SetCanceled()
          else
               try
                  (binder(task.Result)).ContinueWith(fun
(nextTask:Task<'U>) -> tcs.SetResult(nextTask.Result)) |> ignore
               with
               | ex -> tcs.SetException(ex)) |> ignore
       tcs.Task
```

Task async/await is a monad

```
static Task<T> Return<T>(T task)=> Task.FromResult(task);


static async Task<R> Bind<T, R>(this Task<T> task,
    Func<T, Task<R>> cont)
     => await cont(await task.ConfigureAwait(false))
                                 .ConfigureAwait(false);

static async Task<R> Map<T, R>(this Task<T> task, Func<T, R> map)
=> map(await task.ConfigureAwait(false));
```

Task async/await is a monad

```
static async Task<R> SelectMany<T, R>(this Task<T> task,
        Func<T, Task<R>> then) => await Bind(await task);

static async Task<R> SelectMany<T1, T2, R>(this Task<T1> task,
         Func<T1, Task<T2>> bind, Func<T1, T2, R> project)
        {
            T taskResult = await task;
            return project(taskResult, await bind(taskResult));
        }

static async Task<R> Select<T, R>(this Task<T> task, Func<T, R> project)
            => await Map(task, project);

static async Task<R> Return<R>(R value) => Task.FromResult(value);
```

Task Inline - ExecuteSynchronously

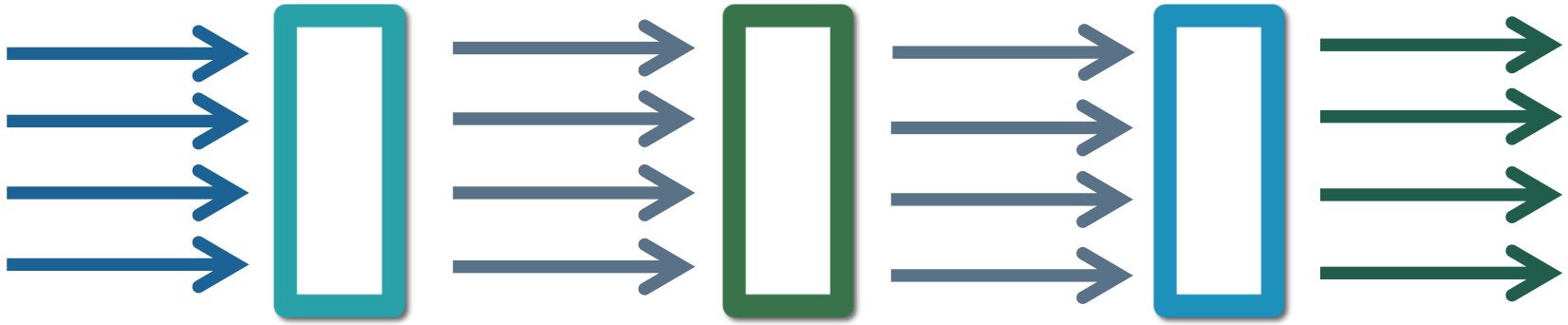
```
public static Task<TOut> SelectMany<TIn, TOut>(this Task<TIn> first, Func<TIn,
Task<TOut>> next)
{
    var tcs = new TaskCompletionSource<TOut>();
    first.ContinueWith(delegate
    {
        if (first.IsFaulted) tcs.TrySetException(first.Exception.InnerExceptions);
        else if (first.IsCanceled) tcs.TrySetCanceled();
        else
        {
                next(first.Result).ContinueWith(t =>
                {
                    if (t.IsFaulted) tcs.TrySetException(t.Exception.InnerExceptions);
                    else if (t.IsCanceled) tcs.TrySetCanceled();
                    else tcs.TrySetResult(t.Result);
                }, TaskContinuationOptions.ExecuteSynchronously);
    }, TaskContinuationOptions.ExecuteSynchronously);
    return tcs.Task;
}
```

Handle Tasks as they complete

```
string[] stocks = new[] { "MSFT", "FB", "AAPL","GOOG", "ORCL" };

List<Task<Tuple<string, StockData[]>>> stockHistoryTasks =
    stocks.Select(ProcessStockHistory).ToList();

while (stockHistoryTasks.Count > 0)
{
    Task<Tuple<string, StockData[]>> stockHistoryTask =
        await Task.WhenAny(stockHistoryTasks);

     stockHistoryTasks.Remove(stockHistoryTask);

    Tuple<string, StockData[]> stockHistory = await stockHistoryTask;

    ShowChartProgressive(stockHistory);
}
```

Lab : Task Pipeline

Pipeline Processing



- A pipeline is a set of data processing elements connected in series, so that the output of one element is the input of the next one. The elements of a pipeline are often executed in parallel or in time-sliced fashion; in that case, some amount of buffer storage is often inserted between elements

Message Passing Concurrency

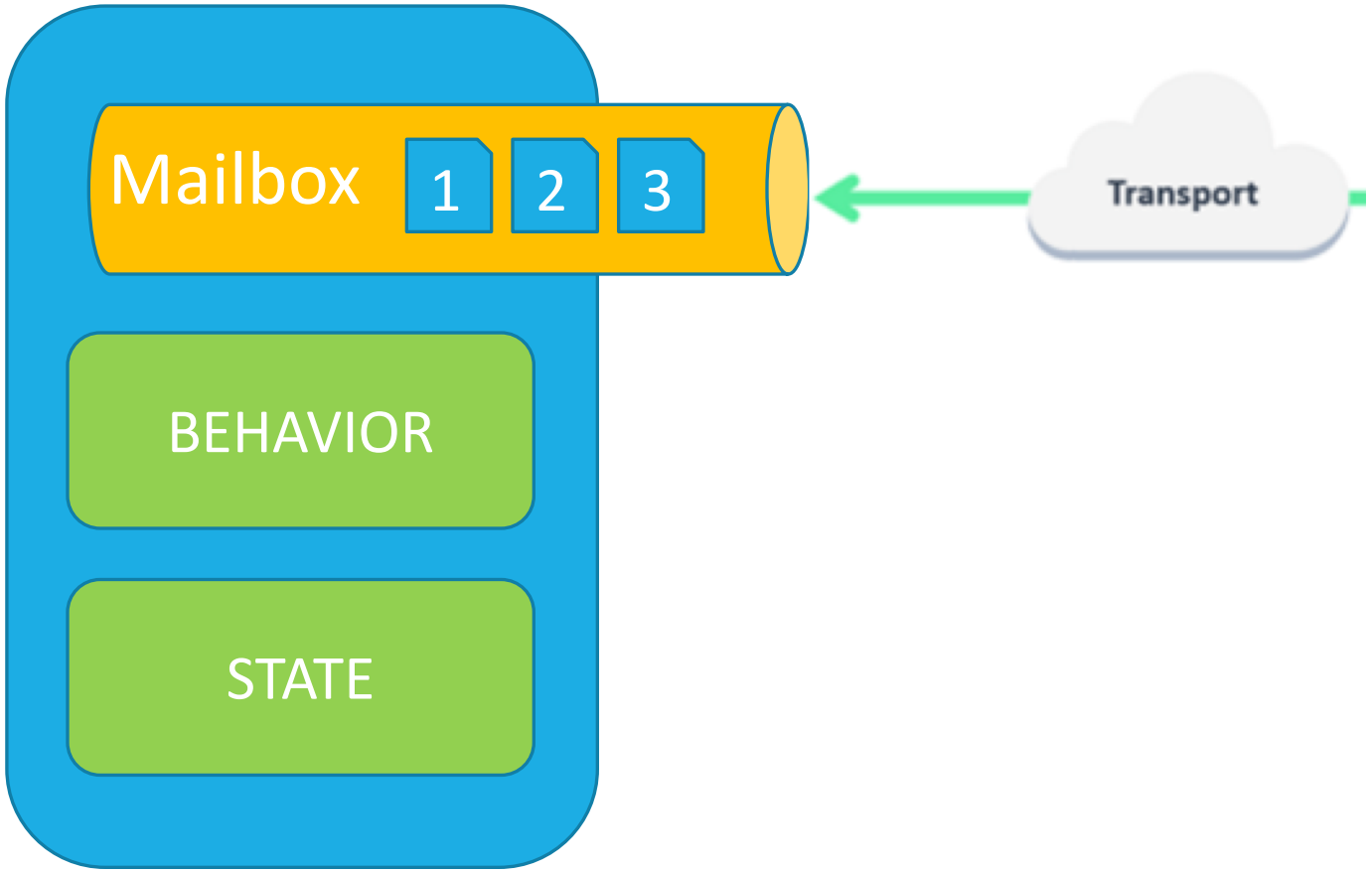
Agent motivations

- You can manage **shared data** and resources **without locks**.
- You can easily follow the **SRP**, because each **agent** can be designed to **do only one thing**.
- It encourages a "pipeline" model of programming with "producers" sending messages to decoupled "consumers“
- It is straightforward to **scale**
- Errors can be handled gracefully, because the decoupling means that agents can be created and destroyed without affecting their clients.

Agent is a single and independent unit of computation

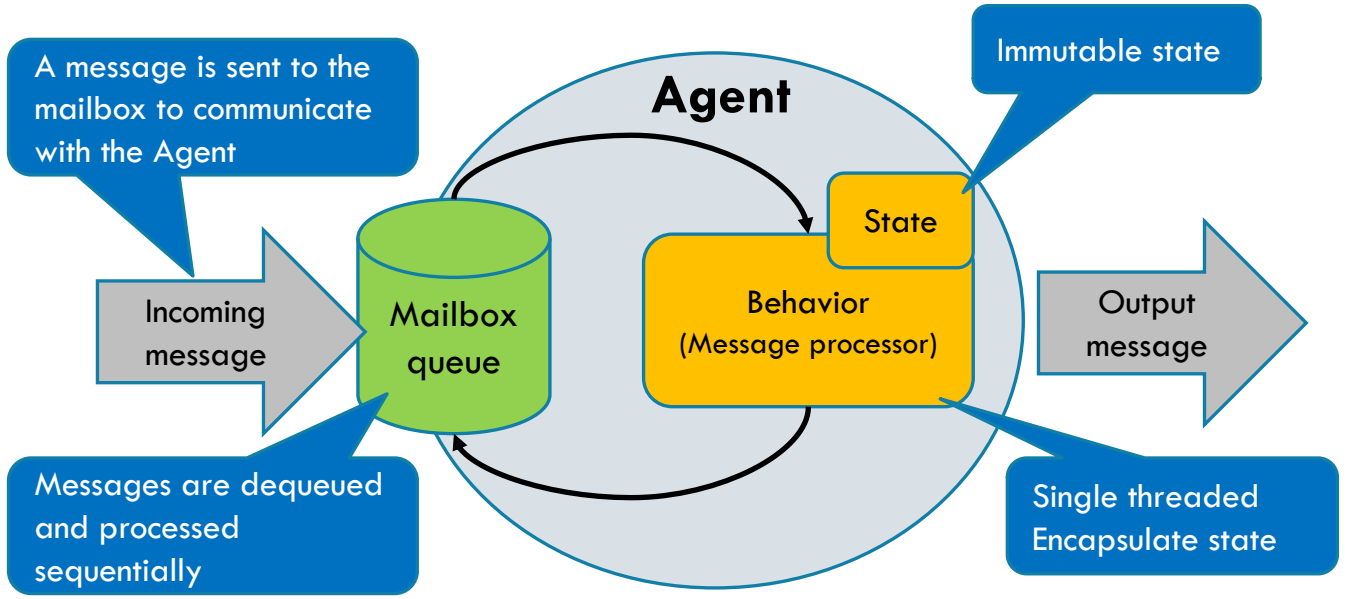
Message-Driven

Message Passing based concurrency



- Processing
- Storage – State
- Communication only by messages
- Share Nothing
- Message are passed by value
- Lightweight object
- Running on it’s own thread
- No shared state
- Messages are kept in mailbox and processed in order
- Massively scalable and lightening fast because of the small call stack

Agent anatomy

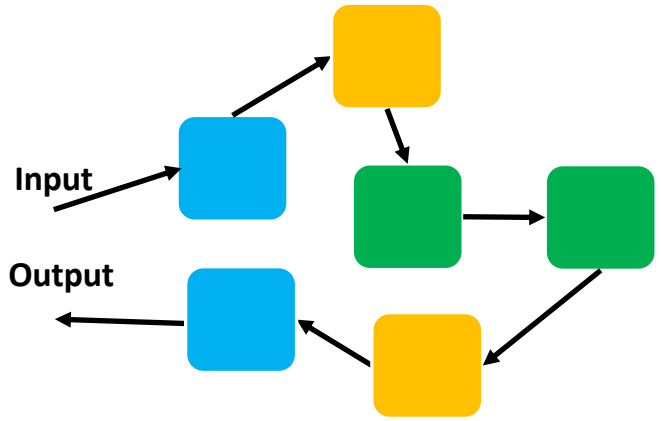


Share Nothing, Async Message Passing

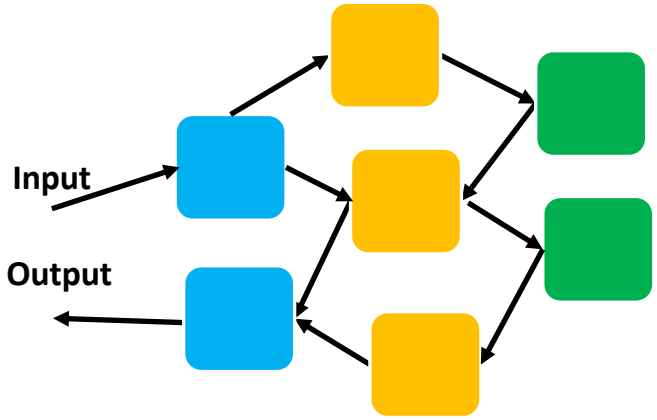
- Encourages shared-nothing process abstractions
  - mutable state - private
  - shared state - immutable
- Asynchronous message passing
- Pattern matching

Comparison between Sequential, Task-based and Message passing programming

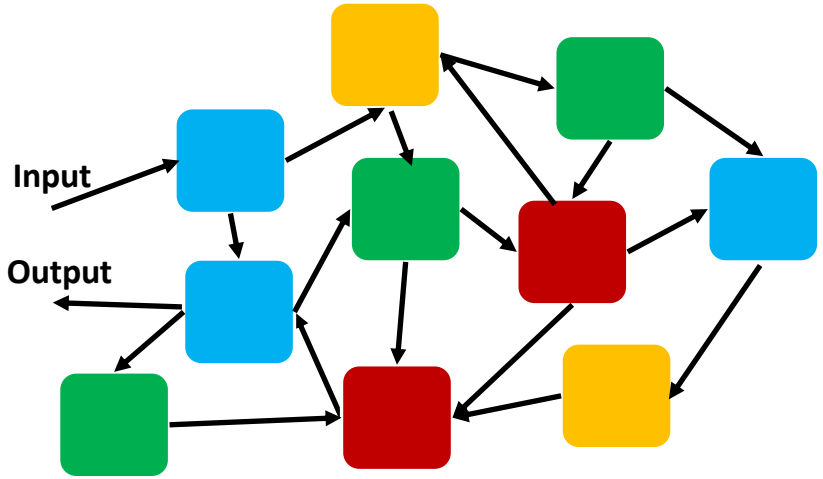
Sequential Programming



Task-Based Programming



Message-Passing Programming



Simple agent in F#

Receive message and say “Hello”

```
let hello = Agent.Start(fun agent -> async {
  while true do
    let! name = agent.Receive()
    printfn "Hello %s"  name
    do! Async.Sleep 500 })

hello.Post(“World!")
```

- Single instance of the body is running
- Waiting for message is asynchronous
- Messages are queued by the agent

Immutability OR Isolation

F#

```
let printerAgent = MailboxProcessor.Start(fun inbox->
    let list = new List<string>()
    // the message processing function
    let rec messageLoop() = async{
        // read a message
        let! msg = inbox.Receive()
        // process a message
        printfn "message is: %s" msg
        // loop to top
         list.Add(msg)
        return! messageLoop()
        }
    // start the loop
    messageLoop()
    )
```

Send message to agent

```
printerAgent.Post "hello"
printerAgent.Post "hello again"
printerAgent.Post "hello a third time"
```

Agent-based architecture

Lots of things going on!

- How to keep a big picture?

Using loosely coupled connections

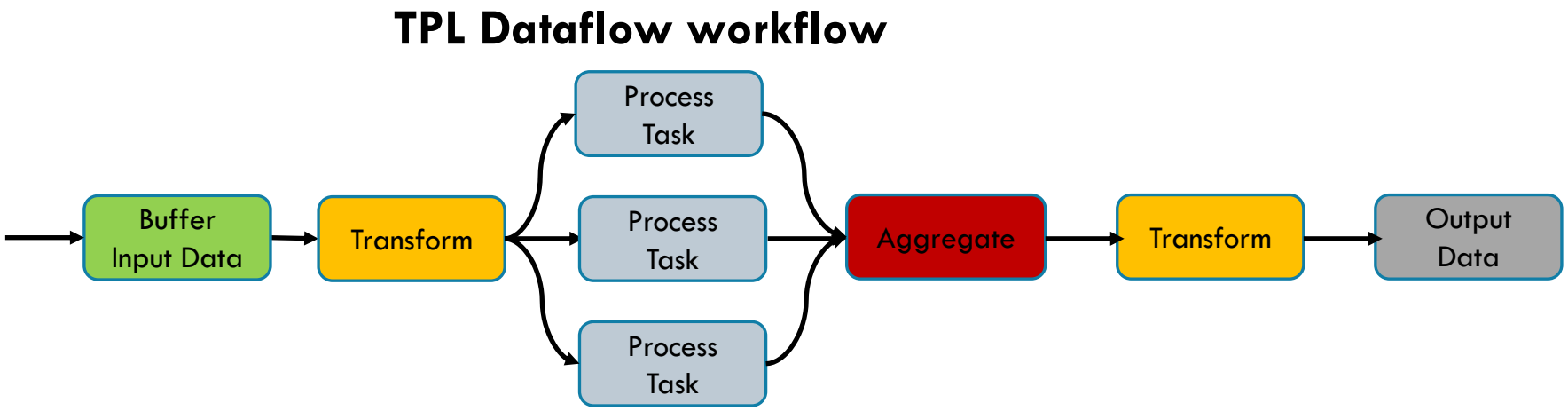
- Agents don’t reference each other directly

Common ways of organizing agents

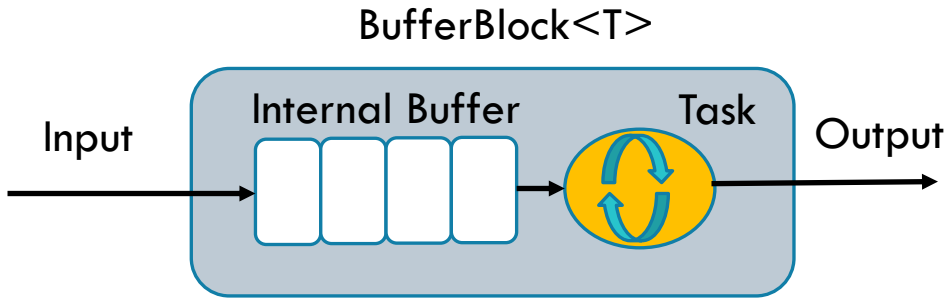
- **Worker agent** – Single agent does work in background
- **Layered network** – Agent uses agents from lower level
- **Pipeline processing** – Step-by-step processing

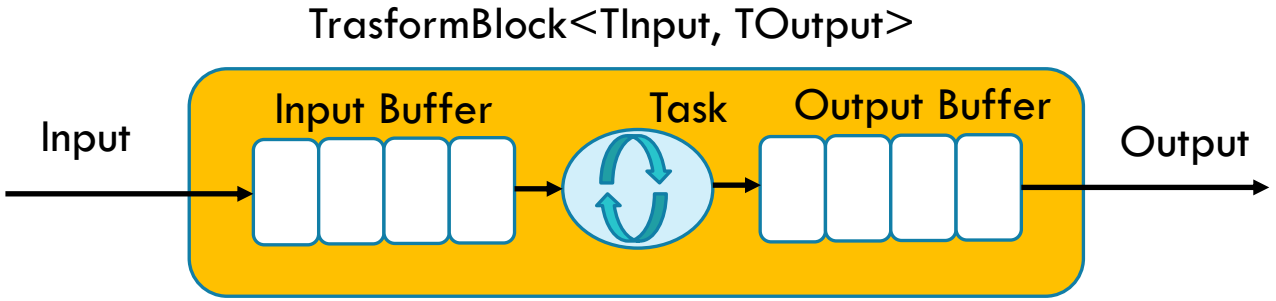
Agent in .NET and C#

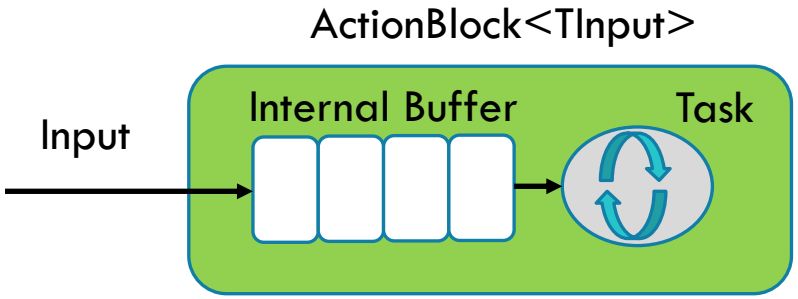
TPL DataFlow blocks –design to compose



Some Dataflow blocks



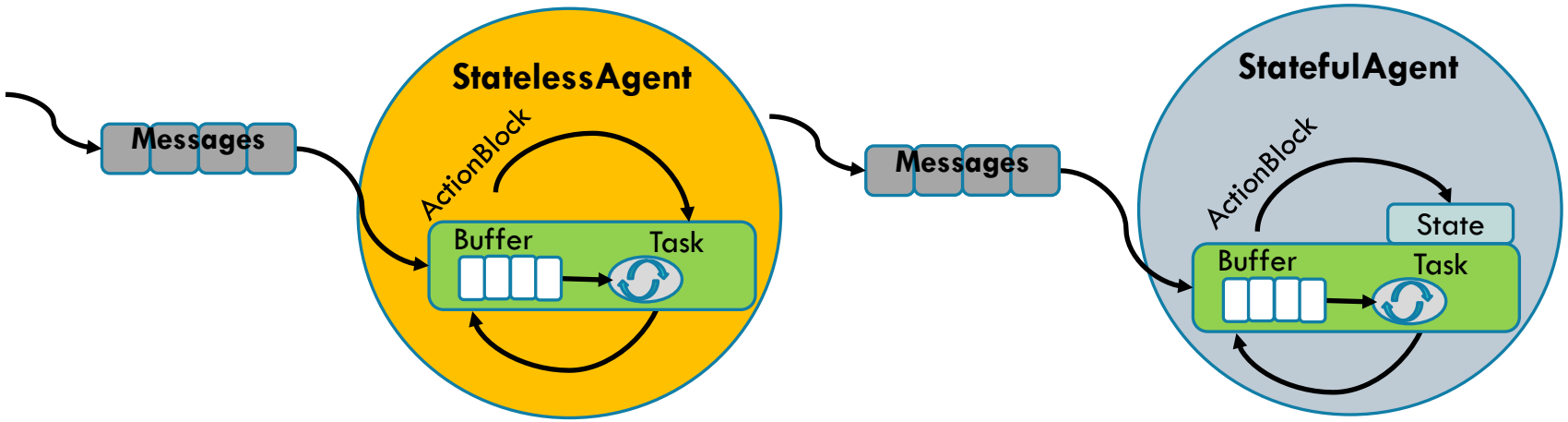




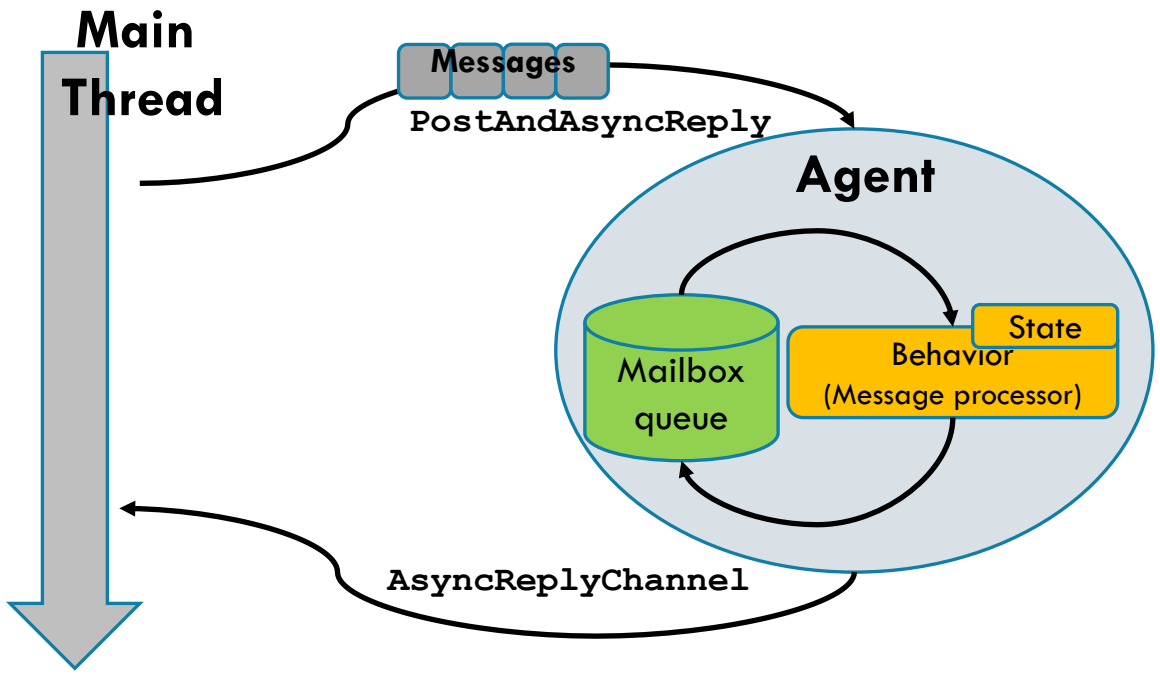
Simple producer-consumer

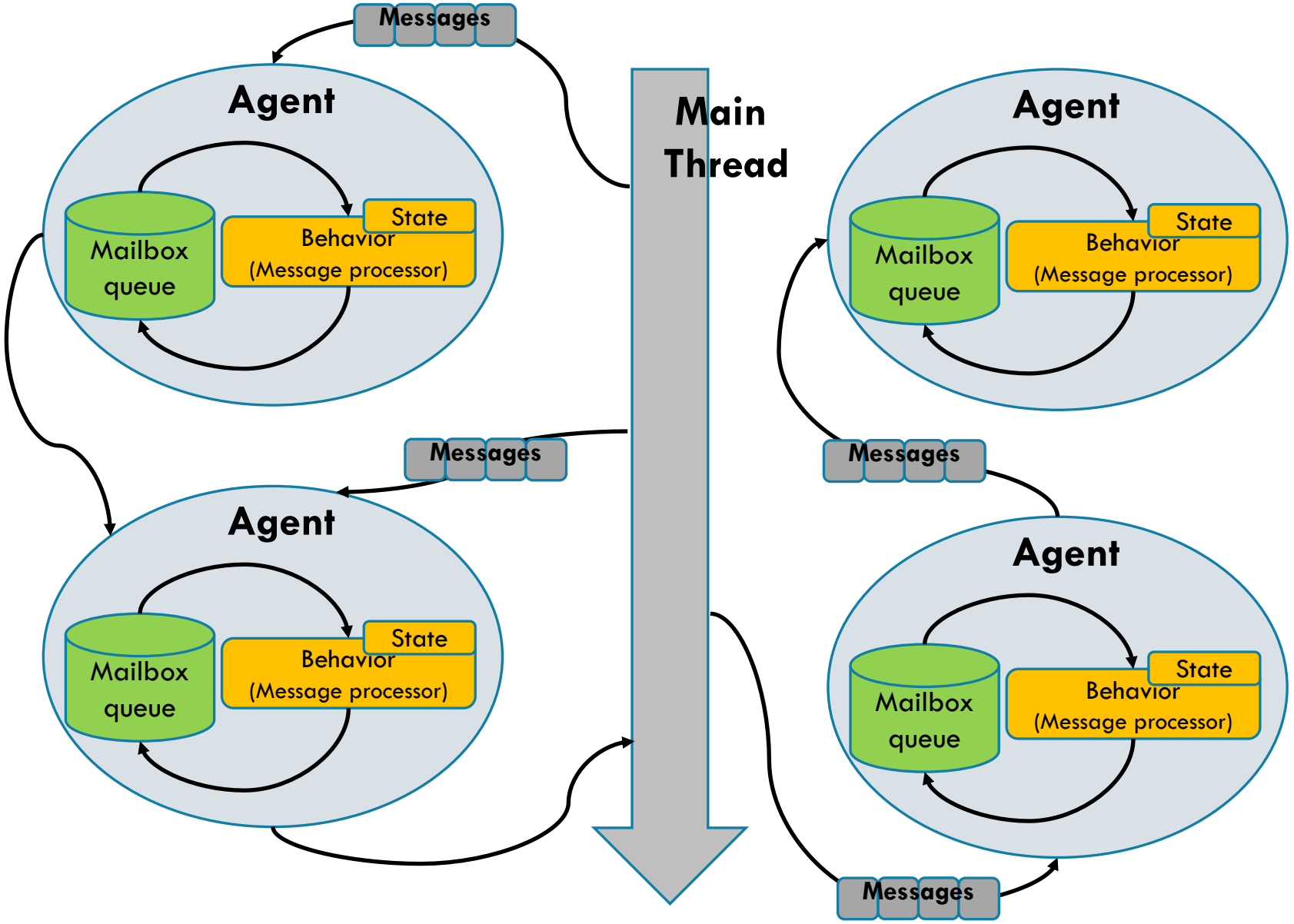
```
BufferBlock<int> buffer = new BufferBlock<int>();

async Task Producer(IEnumerable<int> values) {
       foreach (var value in values)
              buffer.Post(value);
       buffer.Complete();
}
async Task Consumer(Action<int> process)  {
       while (await buffer.OutputAvailableAsync())
             process(await buffer.ReceiveAsync());

}
async Task Run()  {
       IEnumerable<int> range = Enumerable.Range(0,100);
       await Task.WhenAll(Producer(range), Consumer(n =>
              Console.WriteLine($"value {n}")));
}
```

TPL DataFlow as Agent



Agent two-way communication




Messages
Agent
Mailbox queue
State
Behavior (Message processor)
Main Thread
Agent
Mailbox queue
State
Behavior (Message processor)
Messages
Messages
Agent
Mailbox queue
State
Behavior (Message processor)
Agent
Mailbox queue
State
Behavior (Message processor)
Messages

Lab :
Implement a state full agent using TPL DataFlow

Agent in C#

```
class StatefulDataflowAgent<TState, TMessage> : IAgent<TMessage>
{
    private TState state;
    private readonly ActionBlock<TMessage> actionBlock;


    public StatefulDataflowAgent(
        TState initialState,
        Func<TState, TMessage, Task<TState>> action,
        CancellationTokenSource cts = null)
    {
        state = initialState;
        var options = new ExecutionDataflowBlockOptions
        {
            CancellationToken = cts != null ?
                cts.Token : CancellationToken.None
        };
        actionBlock = new ActionBlock<TMessage>(
            async msg => state = await action(state, msg), options);
    }

    public Task Send(TMessage message) => actionBlock.SendAsync(message);
    public void Post(TMessage message) => actionBlock.Post(message);
}
```

Agent two-way communication in C#

```
public Task<TReply> Ask(TMessage message)
{
    var tcs = new TaskCompletionSource<TReply>();
    actionBlock.Post((message, Some(tcs)));
    return tcs.Task;
}

public StatefulReplyDataflowAgent(TState initialState,
Func<TState, TMessage, Task<TState>> projection,
Func<TState, TMessage, Task<(TState, TReply)>> ask,
CancellationTokenSource cts = null)
{
    state = initialState;
    actionBlock = new ActionBlock<(TMessage, Option<TaskCompletionSource<TReply>>)>(
        async message =>
        {
            (TMessage msg, Option<TaskCompletionSource<TReply>> replyOpt) = message;
            await replyOpt.Match(
                    none: async () => state = await projection(state, msg),
                    some: async reply => {
                        (TState newState, TReply replyresult) = await ask(state, msg);
                        state = newState;
                        reply.SetResult(replyresult);
                    });
```

TPL DataFlow caching web-sites downloaded

```
List<string> urls = new List<string> {
       ”http://www.google.com",
                    "http://www.microsoft.com",
                    "http://www.bing.com",
                    "http://www.google.com"
            };

var agentStateful = Agent.Start(ImmutableDictionary<string,string>.Empty,
   async (ImmutableDictionary<string,string> state, string url) => {
          if (!state.TryGetValue(url, out string content))
       using (var webClient = new WebClient()){
         content = await webClient.DownloadStringTaskAsync(url);
         await File.WriteAllTextAsync(createFileNameFromUrl(url), content);
         return state.Add(url, content);
      }
   return state;
   });

urls.ForEach(url => agentStateful.Post(url));
```

Agent fold-over state and messages (Aggregate)

```
Agent(ImmutableDictionary<string,string>.Empty,
        async (state, url) => {
if (!state.TryGetValue(url, out string content))
  using (var webClient = new WebClient())
  {
        content = await webClient.DownloadStringTaskAsync(url);
        await File.WriteAllTextAsync(createFileNamFromUrl(url),     content);
        return state.Add(url, content);
  }
  return state;
});
```

Agent fold-over state and messages (Aggregate)

```
urls.Aggregate(ImmutableDictionary<string,string>.Empty,
        async (state, url) => {
if (!state.TryGetValue(url, out string content))
  using (var webClient = new WebClient())
  {
        content = await webClient.DownloadStringTaskAsync(url);
        await File.WriteAllTextAsync(createFileNamFromUrl(url),     content);
        return state.Add(url, content);
  }
  return state;
});
```

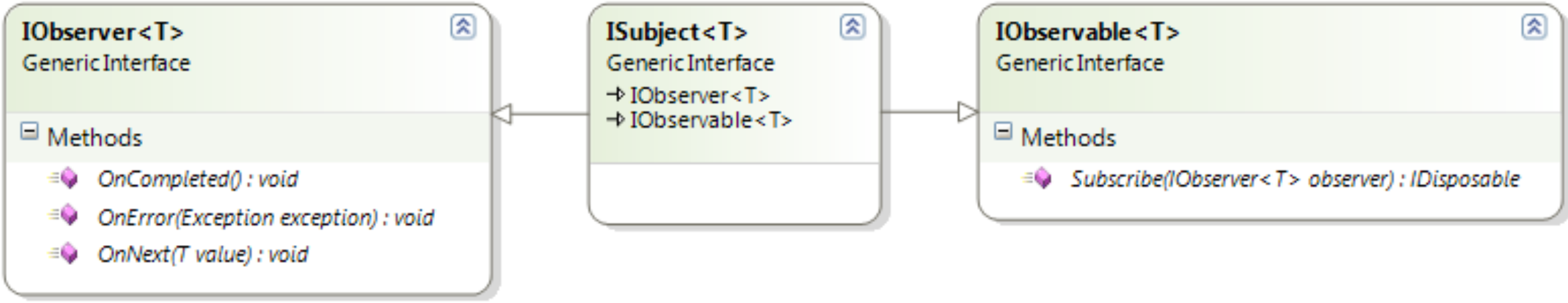
Lab :
Implement a state full agent using TPL DataFlow

TPL Dataflow & Reactive Extension

Reactive Extensions

What are Reactive Extensions

Rx is a library for composing asynchronous and event-based programs using observable sequences and LINQ-style query operators



IObserver & IObservable

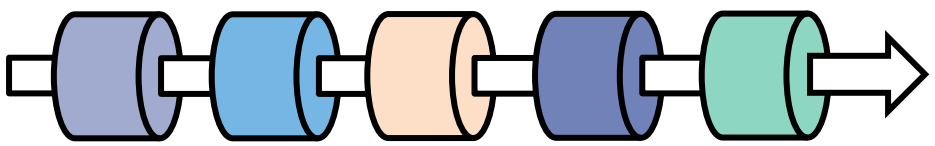
```
type IObserver<'a> = interface
    abstract OnCompleted : unit -> unit
    abstract OnError : exn -> unit
    abstract OnNext : 'a -> unit
end

type IObservable<'a> = interface
    abstract Subsrcribe : IObserver<'a> -> IDisposable
end
```

What is Rx?

Rx **compose pipeline** of operations over **single** or **multiple event's source**

Focus on what happens **between** the **Producer** and the **Consumer**

Rx is the **Pipeline**



Rx = Async Data Stream Composition

Rx = LINQ to Async Data Stream

TPL Dataflow & Reactive Extension

```
IPropagatorBlock<int, string> source =
            new TransformBlock<int, string>(i => (i + i).ToString());

IObservable<int> observable = source.AsObservable().Select(int.Parse);

IDisposable subscription =
              observable.Subscribe(i => $"Value {i} - Time
                                  {DateTime.Now.ToString("hh:mm:ss.fff")}".Dump());


for (int i = 0; i < 100; i++)
          source.Post(i);
```

TPL Dataflow & Reactive Extension

```
IPropagatorBlock<string, int> target =
                            new TransformBlock<string, int>(s => int.Parse(s));

IDisposable link = target.LinkTo(new ActionBlock<int>(i => $"Value {i} - Time
                                    {DateTime.Now.ToString("hh:mm:ss.fff")}".Dump())

IObserver<string> observer = target.AsObserver();
IObservable<string> observable = Observable.Range(1,20)
                                        .Select(i => (i *i).ToString());
observable.Subscribe(observer);

for (int i = 0; i < 100; i++)
        target.Post(i.ToString());
```

TPL DataFlow and Rx

```
inputBuffer.LinkTo(compressor, linkOptions);
compressor.LinkTo(encryptor, linkOptions);

encryptor.AsObservable()

          .Scan((new Dictionary<int, EncryptDetails>(), 0),
        (state, msg) => Observable.FromAsync(async() => {
         (Dictionary<int,EncryptDetails> details, int lastIndexProc) = state;
         details.Add(msg.Sequence, msg);

         return (details, lastIndexProc);
}) .SingleAsync())
.SubscribeOn(TaskPoolScheduler.Default).Subscribe();
```

Lab :
TPL DataFlow & Reactive Extensions

Summary

- The free lunch is over
- Applications must be built for concurrency
- Shared state concurrency is hard
- Alternative concurrency models
  - Message passing concurrency
  - Divide and Conquer

Survey

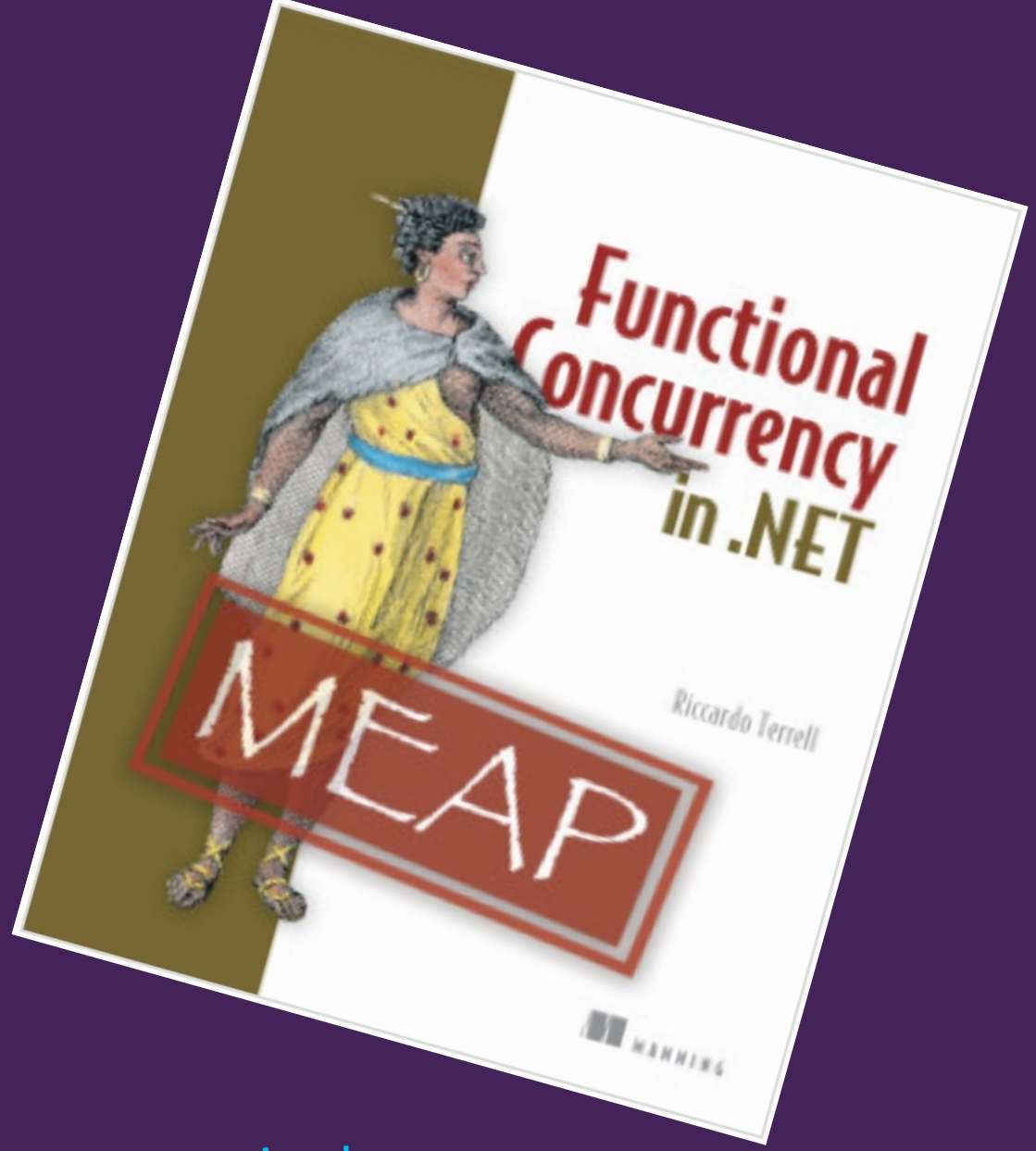
https://www.surveymonkey.com/r/6BPKBHJ

The tools we use have a profound (and devious!) influence on our thinking habits, and, therefore, on our thinking abilities.

-- Edsger Dijkstra

terrellpc

40% off Functional Concurrency in .NET
(all formats)



https://www.manning.com/books/functional-concurrency-in-dotnet

contacts

Source	https://github.com/rikace/ncrafts-ws
Twitter	*@trikace*
Blog	*www.rickyterrell.com*
Email	*tericcardo@gmail.com*
Github	*www.github.com/rikace/*


That's all Folks!