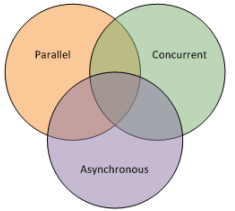
Choose the concurrency model in .NET

"I have no special talents. I am only passionately curious."

- Albert Einstein

Riccardo Terrell

Agenda



What is concurrency – parallelism – asynchronous - multithreading

Different programming models (pro & cons) – with code sample

Code Examples

- Threading & Parallel Programming
- Asynchronous Workflows
- Actor model
- Alternative STM – Channel …

Introduction - Riccardo Terrell

- Originally from Italy, currently - Living/working in Washington DC ~10 years
- +/- 19 years in professional programming
  - C++/VB → Java → .Net C# → Scala → Haskell → C# & F# → ??
- DC F# User Group - Organizer
- Working @ statmuse
- *Polyglot programmer - believes in the art of finding the right tool for the job*

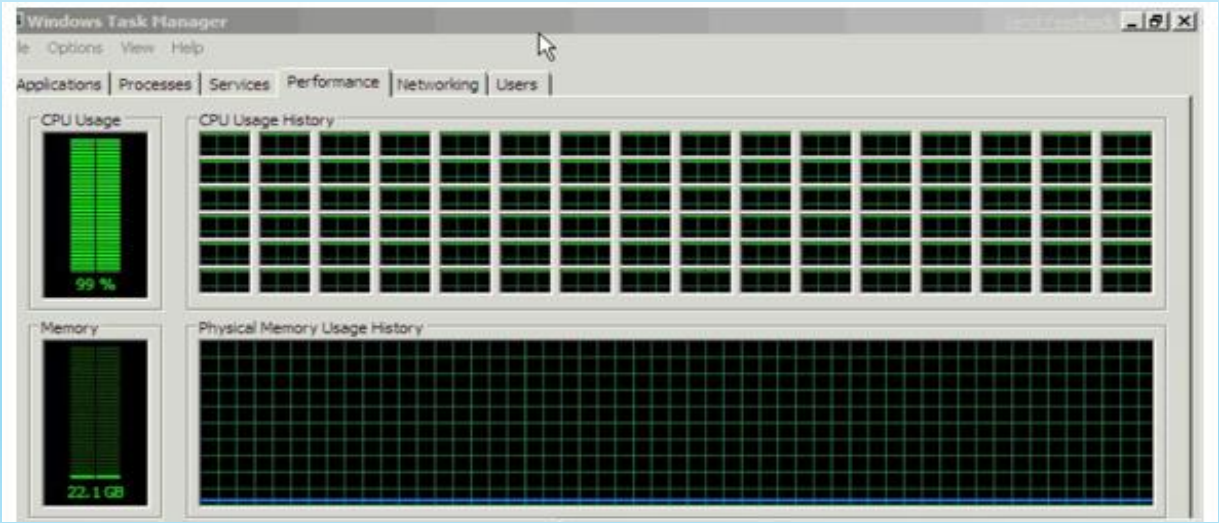
@trikace *rickyterrell.com* *tericcardo@gmail.com*

MVP Microsoft® Most Valuable Professional


What about you?

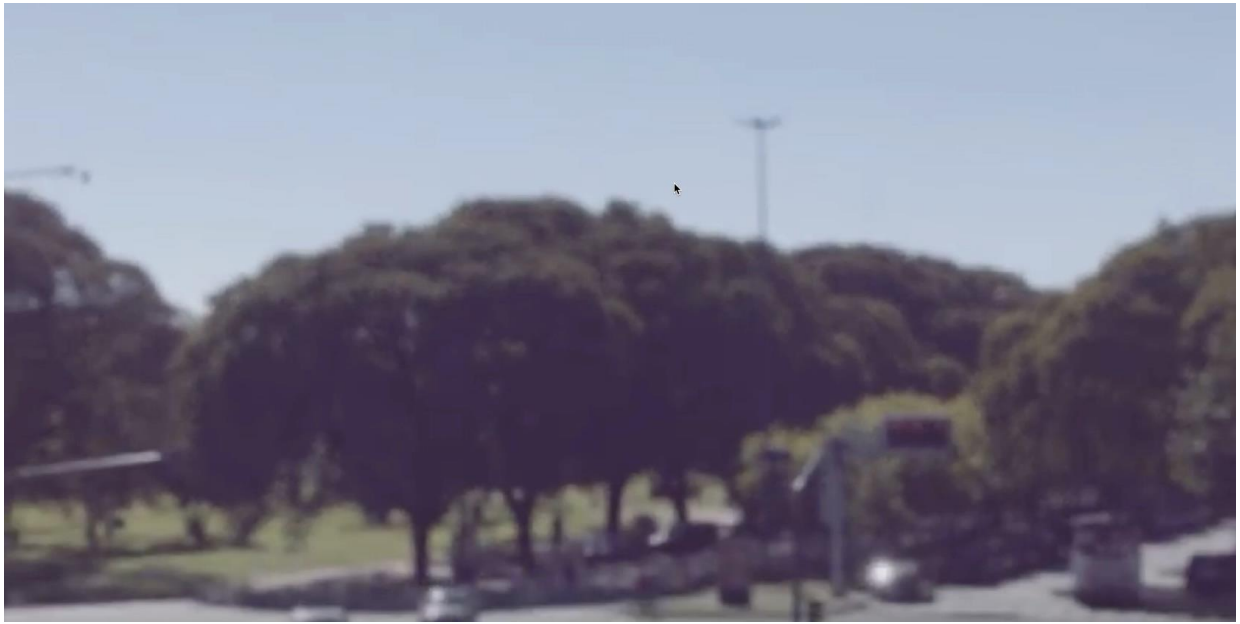
Moore's law - The Concurrency challenge

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

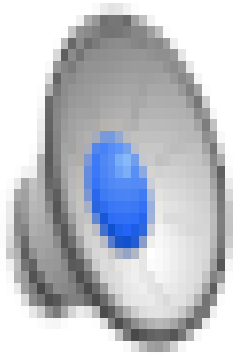


The need parallelism

Multithreading in practice

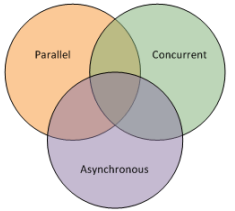


The issue is Shared of Memory



- 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

Goals



Embrace Parallelism exploiting the potential of multicore CPUs

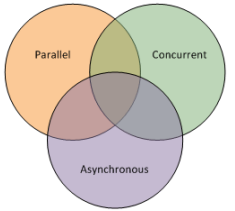
No single solution…

You must combine different programming models

(*right tool for the job*)

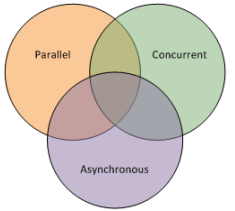
FP really **shines** in the area of concurrency

Definition



- Concurrent
- Multithreaded
- Parallel
- Asynchronous

Definition

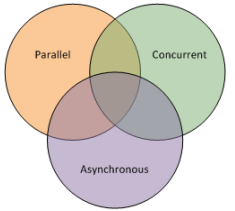


- Concurrent
- Multithreaded
- Parallel
- Asynchronous

Several things happening at once

TPL – Async – Agent - RX

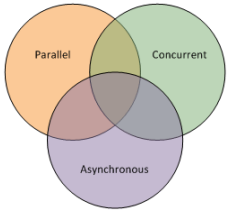
Definition



- Concurrent
- Multithreaded
- Parallel
- Asynchronous

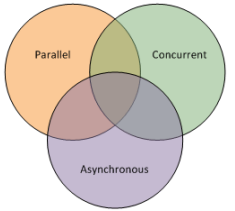
- Several things happening at once
  TPL – Async – Agent - RX
- Multiple execution context

Definition



Concurrent	Several things happening at once TPL – Async – Agent - RX
Multithreaded	Multiple execution context
Parallel	Multiple simultaneous computations TPL – Async – Agent – RX - GPU
Asynchronous	

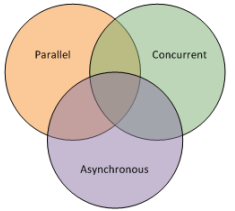
Definition



Concurrent	Several things happening at once TPL – Async – Agent - RX
Multithreaded	Multiple execution context
Parallel	Multiple simultaneous computations TPL – Async – Agent – RX - GPU
Asynchronous	Asynchronous AsyncWorkflow – TAP – Agent

Concurrent Programming Models

Sequential Fuzzy Match

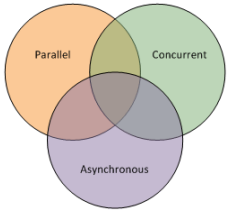



```
List<string> matches = new List<string>();
foreach (var word in WordsToSearch)
{
      var localMathes = JaroWinklerModule.bestMatch(Words, word);
      matches.AddRange(localMathes.Select(m => m.Word));
}
```

Fuzzy match 7 words against 13.4 Mb of text

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

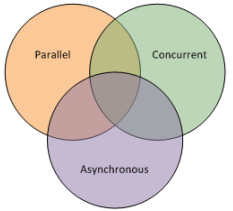
Two Threads Fuzzy Match




```
List<string> matches = new List<string>();
    var t1 = new Thread(() =>{
        var take = WordsToSearch.Count / 2;

        foreach (var word in WordsToSearch.Take(take))  {
            var localMathes = FuzzyMatch.JaroWinklerModule.bestMatch(Words, word);
            matches.AddRange(localMathes.Select(m => m.Word));
        }
    });
    var t2 = new Thread(() =>{
        var start = WordsToSearch.Count / 2;
        var take = WordsToSearch.Count - start;

        foreach (var word in WordsToSearch.Skip(start).Take(take)) {
            var localMathes = FuzzyMatch.JaroWinklerModule.bestMatch(Words, word);
            matches.AddRange(localMathes.Select(m => m.Word));
        }
    });
    t1.Start();                    t2.Start();
    t1.Join();                     t2.Join();
```

Time execution : **15,436** ms

Multi Thread Fuzzy M tch

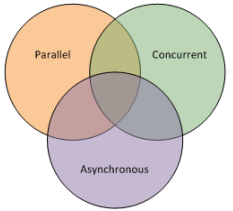



```
List<string> matches = new List<s                    ();
var threads = new Thread[Environm

for (int i = 0; i < threads.Leng

      threads[index]

      var take = Word                                    ads.Length));
      var start =                                        index * take;

      foreach (va                                        e)) {
                var lo                                   e.bestMatch(Words, word);
                match                                    ect(m =>      ord));

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

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

WRONG!!!

Time execution : **6,857** ms

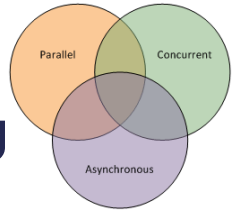
Multi Thread Fuzzy Match




```
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

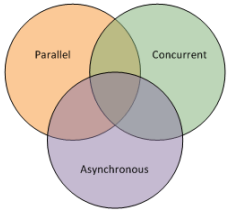
Task Parallel Library (TPL) - Task Programming



- Efficient and scalable use of system resources..
  - When used correctly
- Programmatic fine grain control
- Model largely applicable
- Easy to integrate into existing programs

- Hard to solve complex problems
- Sometimes requires the introduction of locking primitives

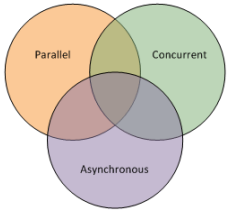
Parallel Loop Fuzzy Match




```
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 LINQ (PLINQ)




```
var query =
    from i in Enumerable.Range(1, 10).AsParallel()
```

LINQ to Objects
LINQ to XML
not LINQ to SQL,EF

Extension method in System.LINQ

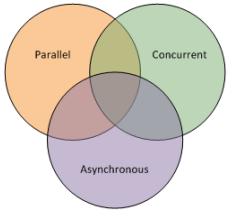
Extends **IEnumerable<T>**

"Usually" would be **Enumerable**

Results in a **ParallelQuery<T>**

Select() | Where() | Etc.

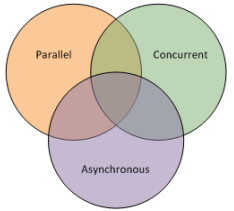
PLINQ Fuzzy Match

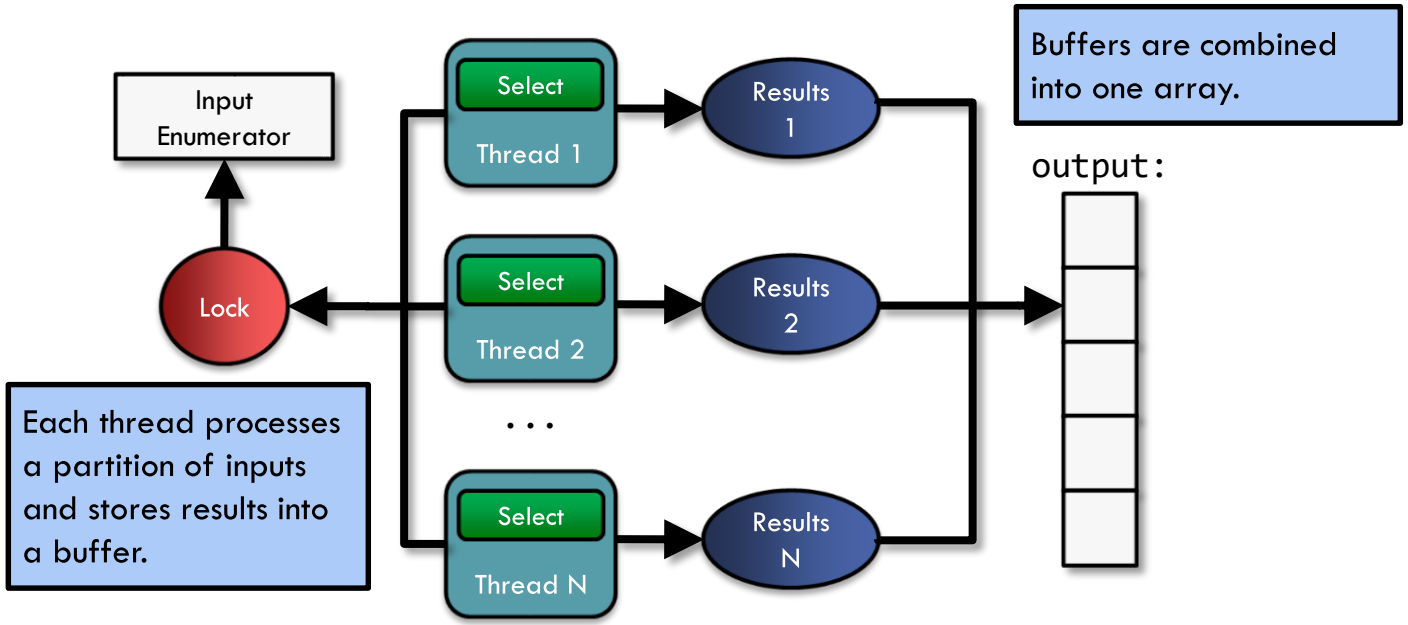



```
ParallelQuery<string> matches =
       (from word in WordsToSearch.AsParallel()
        from match in JaroWinklerModule.bestMatch(Words, word)
        select match.Word);
```

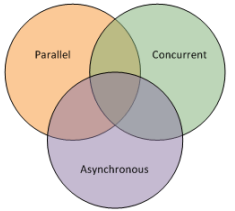
Time execution : **6,347** ms

PLINQ under the hood



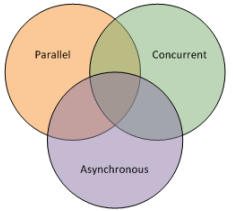


PLINQ options




```
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);
```

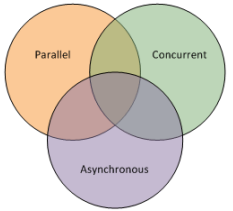
Multi Task Fuzzy Match




```
var tasks = new List<Task<List<string>>>();
var matches = new ThreadLocal<List<string>>(() => new List<string>());

foreach (var word in WordsToSearch) {

        tasks.Add(Task.Factory.StartNew<List<string>>((w) => {
        List<string> localMatches = matches.Value;
        var localMathes = FuzzyMatch.JaroWinklerModule.bestMatch(Words, w);
        localMatches.AddRange(localMathes.Select(m => m.Word));
        return localMatches;
       }, word));
}
Task.Factory.ContinueWhenAll(tasks.ToArray(), (ts) =>
      return new List<string>(tasks.SelectMany(t => t.Result).Distinct())
).Wait();
```

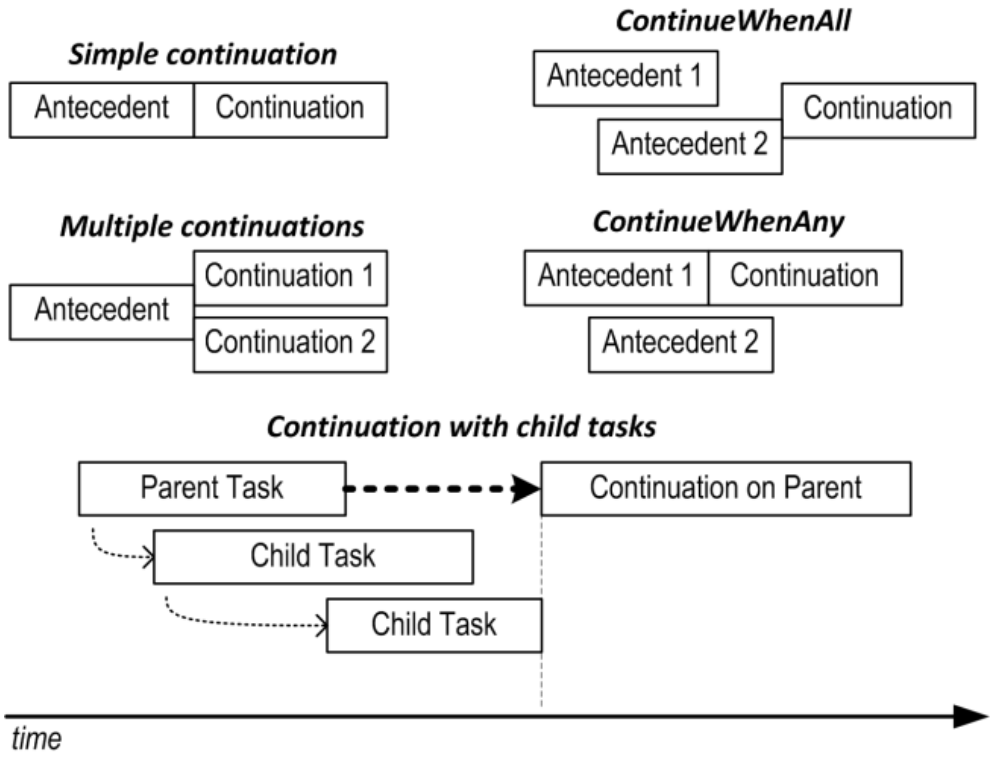
Time execution : **4,192** ms

PLINQ Fuzzy Match



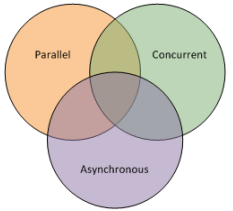

```
Create a load-balancing partitioner, specify false for static partitioning.


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



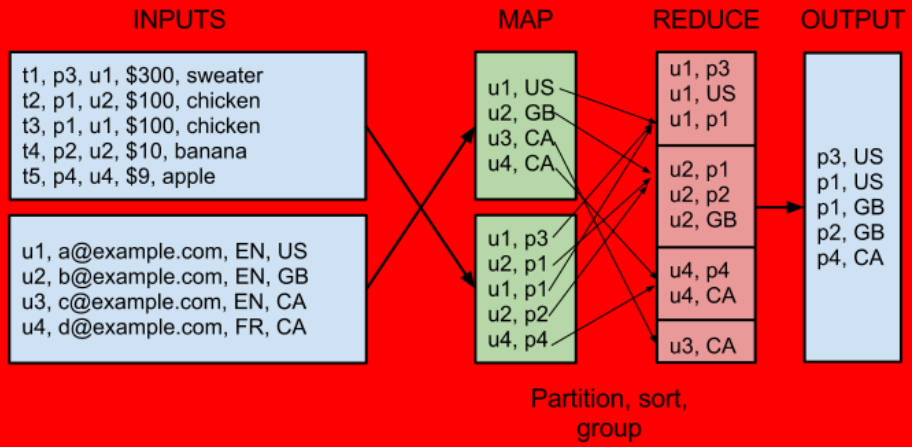
TaskContinuationOptions
AttachedToParent
ExecuteSynchronously
LazyCancellation
LongRunning
None
NotOnCanceled
NotOnFaulted
NotOnRanToCompletion
OnlyOnCanceled
OnlyOnFaulted
OnlyOnRanToCompletion
RunContinuationsAsynchronously

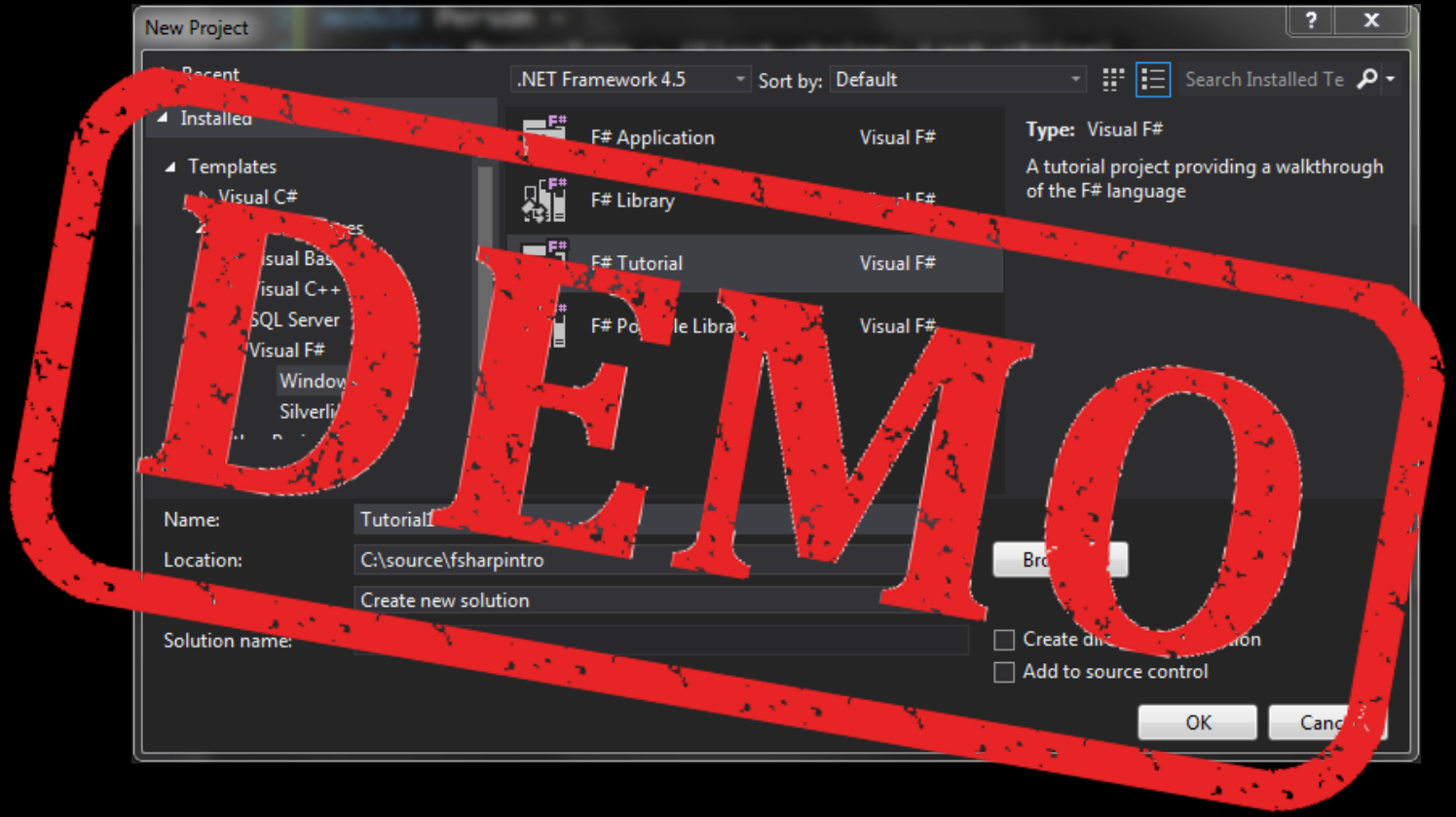
Task continuation




```
task1.ContinueWith(antecedent =>
        Console.WriteLine("Task #1 completion continuation."),
        TaskContinuationOptions.OnlyOnRanToCompletion);

task1.ContinueWith(antecedent =>
        Console.WriteLine("Task #1 cancellation continuation."),
        TaskContinuationOptions.OnlyOnCanceled);

task1.ContinueWith(antecedent =>
        Console.WriteLine("Task #1 on error continuation."),
        TaskContinuationOptions.OnlyOnFaulted);

task1.ContinueWith(antecedent =>
        Console.WriteLine("Task #1 continuation lomg running."),
        TaskContinuationOptions.LongRunning);
```

Data Parallelism & Lambda Architecture

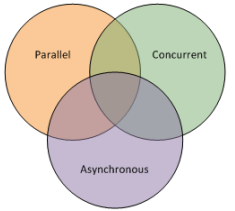



New Project
.NET Framework 4.5
Sort by:
Default
Search Installed Te
Installed
Templates
Visual C#
Visual C++
SQL Server
Visual F#
F# Application
Visual F#
F# Library
F# Tutorial
Visual F#
Type: Visual F#
A tutorial project providing a walkthrough of the F# language
Name:
Location:
C:\source\fsharpintro
Create new solution
Solution name:
Add to source control
OK
DEMO

Functional Programming in Concurrency

- declarative
- functions as values
- side-effects free
- referential transparency
- immutable
- composition

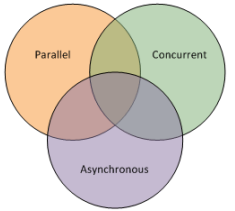
Immutability




```
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);
    }
}
```

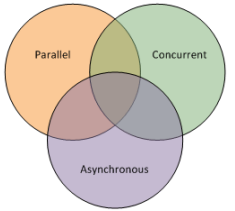
```
type Person = {FirstName:string; LastName:string; Age:int}
```

Compare And Swap - CAS



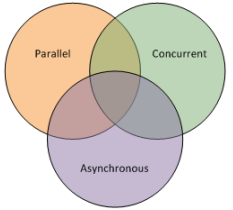

```
void DoWithCAS<T>(ref T location, Func<T, T> generator) where T : class
      {
         T temp, replace;
         do
         {
             temp = location;
             replace = generator(temp);
         } while (Interlocked.CompareExchange(ref location, replace, temp) != temp);
      }
```

Compare And Swap - CAS



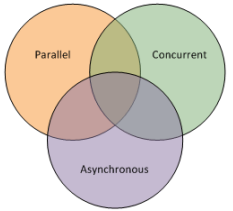

```
public class LockFreeStack<T>
{
    private class Node {
        public T Data;
        public Node Next;
    }
    private Node head;

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

Side Effects




```
int WordCounterLinq(string dirPath) {
    var wordTable= (from file in Directory.EnumerateFiles(dirPath, "*.*")
                    from line in File.ReadAllLines(file)
                    from word in Regex.Split(line, @"\W+")
                    where !string.IsNullOrEmpty(word) && word.Length > 0
                    select word.ToUpper())
                .GroupBy(s => s).ToDictionary(k => k.Key, v => v.Count());
    return wordTable;
}
```

Side Effects Free




```
int WordCounterLinq(string[] lines) {
    var wordTable= (    from line in lines
                        from word in Regex.Split(line, @"\W+")
                        where !string.IsNullOrEmpty(word) && word.Length > 0
                        select word.ToUpper())
                   .GroupBy(s => s).ToDictionary(k => k.Key, v => v.Count());
    return wordTable;
}
```

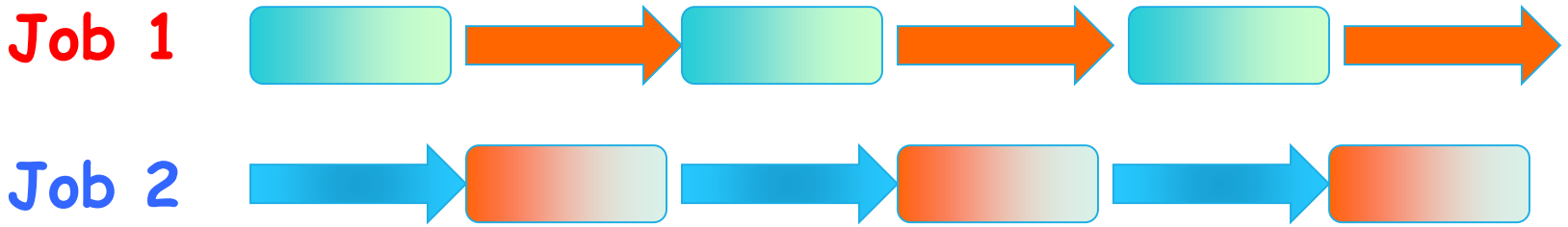
Asynchronous Programming

Asynchronous Programming

- *Software is often I/O-bound, it provides notable performance benefits*
  - Connecting to the Database
  - Leveraging web services
  - Working with data on disk
- *Network and disk speeds increasingly slower*
- *Not Easy to predict when the operation will complete* (no-deterministic)
- ***IO bound functions can scale regardless of threads***
  - ***IO bound computations can often "overlap"***
  - ***This can even work for huge numbers of computations***

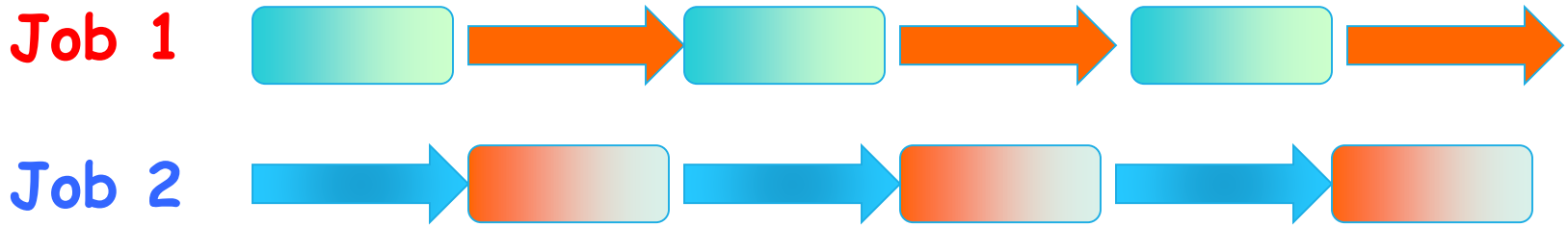
Classic Synchronous programming

- We are used to writing code linearly

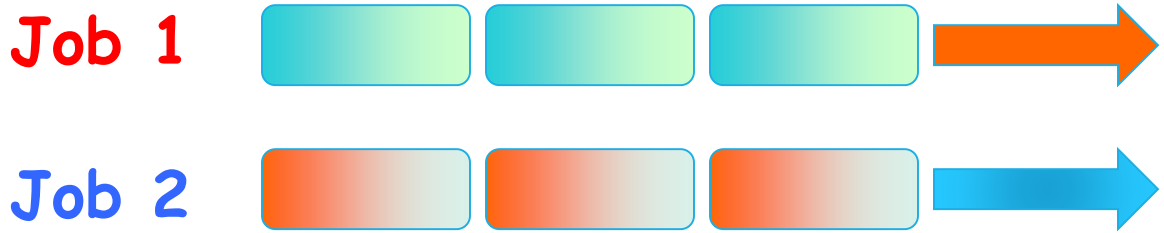


Classic Synchronous programming

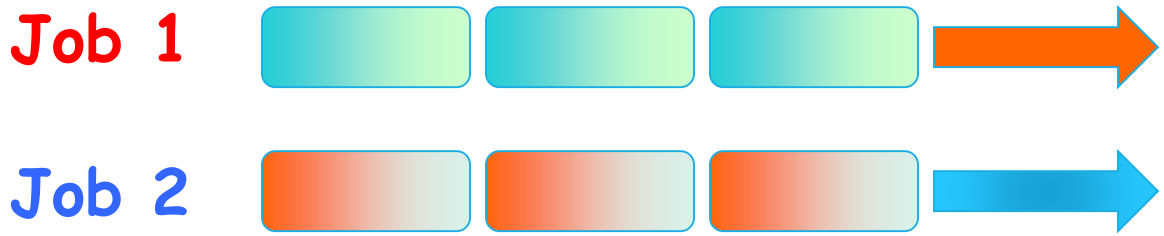
- We are used to writing code linearly



- Jobs executing in parallel



Classic Asynchronous programming



- Classic Asynchronous programming requires decoupling Begin from End
- Very difficult to:
  - Combine multiple asynchronous operations
  - Deal with exceptions, cancellation and transaction

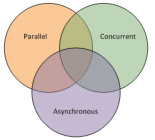
Classic Asynchronous Programming

```
let callBack (callBack:IAsyncRes
    let fs = callBack.As
    fs.EndRead(callBa
    fs.Dispose()
```

```
let fs =
    let
    let asyncR
        fs.Be    Read(d    0, da    ength,
                          callBac    s)
```

Where is my DATA ?
@#!$#!$?!!

Asynchronous Programming in C# 5.0




```
Task<string> GetTitleTplAsync(string url) {
            var w = new WebClient();
            Task<string> contentTask = w.DownloadStringTaskAsync(url);
            return contentTask.ContinueWith(t => {
                string result = ExtractTitle(t.Result);
                w.Dispose();
                return result; });
        }

async Task<string> GetTitleCsAsync(string url) {
             using (var w = new WebClient()){
                 string content = await w.DownloadStringTaskAsync(url);
                 return ExtractTitle(content);
             }
```

Anatomy of Async Workflows F#

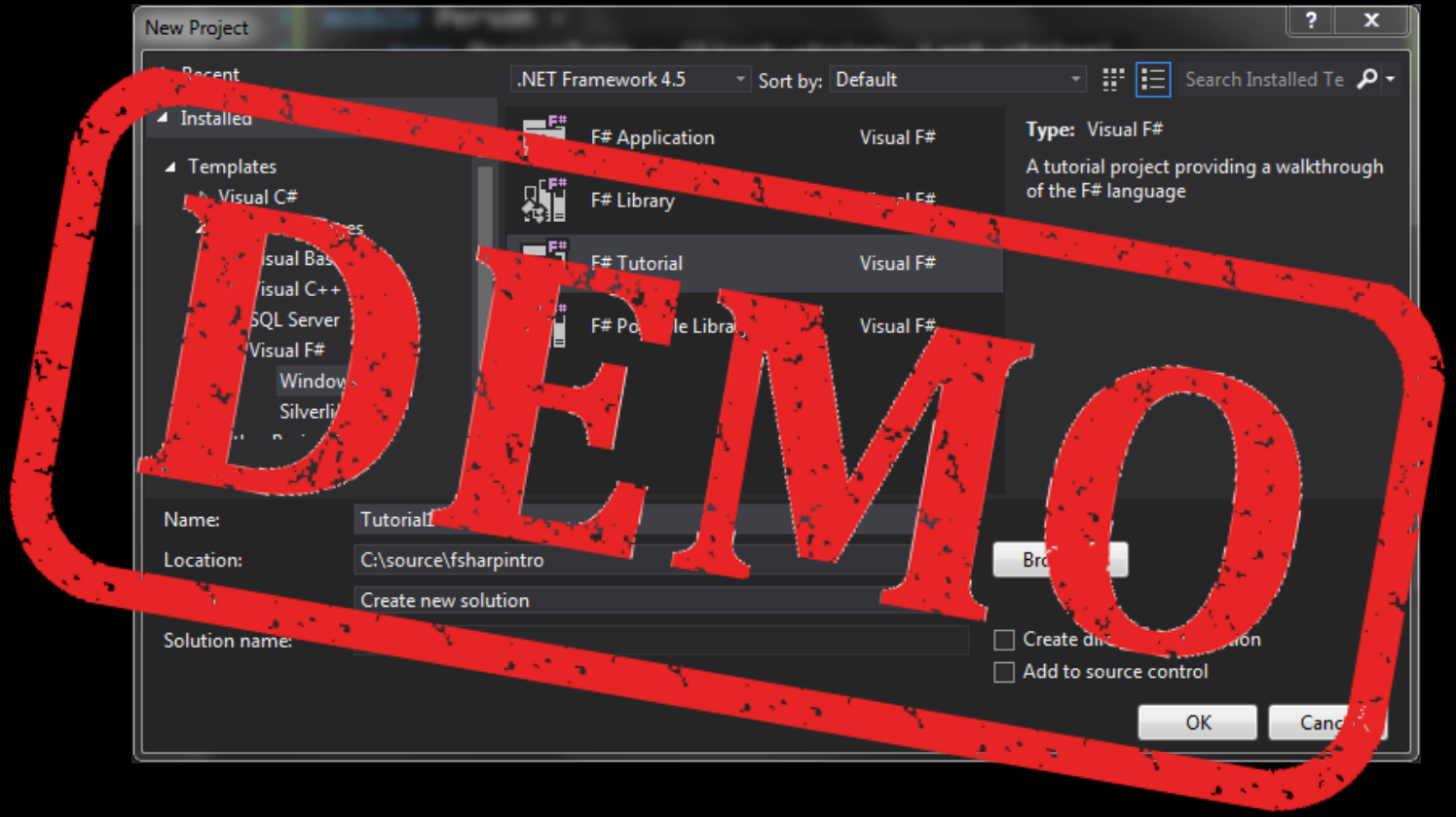
```
let readData path = async {
  let stream = File.OpenRead(path)
  let! data = stream.AsyncRead(stream.Length)
  return data }
```

- Easy transition from synchronous
  - Wrap in asynchronous workflow with the ***async*** keyword, use **let!** for async calls and add **return**
  - No need for explicit callback
  - Easy to debug
- Supports loops, recursion, exceptions, cancellation, resource management
- **Operation complete in the same scope**

Anatomy of Async Workflows

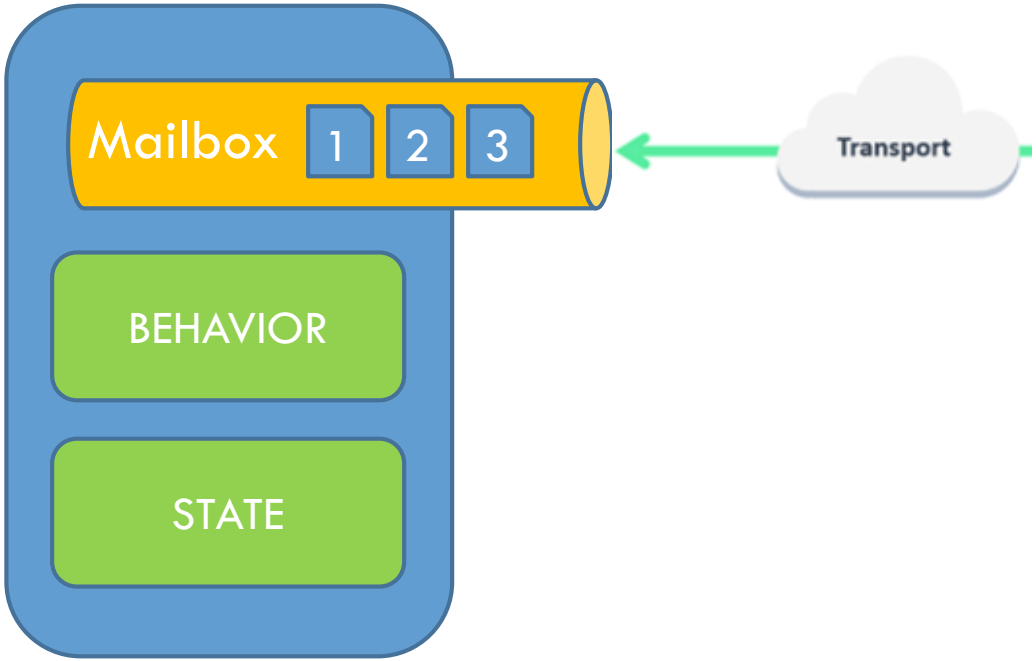
```
let readData path : Async<byte[]> = async {
  let stream = File.OpenRead(path)
  let! data = stream.AsyncRead(stream.Length)
  return data }
```

- Async defines a block of code which executes on demand
- Easy to compose


New Project
.NET Framework 4.5
Sort by:
Default
Search Installed Te
Installed
Templates
Visual C#
Visual C++
SQL Server
Visual F#
F# Application
Visual F#
F# Library
F# Tutorial
Visual F#
Type: Visual F#
A tutorial project providing a walkthrough of the F# language
Name:
Location:
C:\source\fsharpintro
Create new solution
Solution name:
Add to source control
OK
DEMO

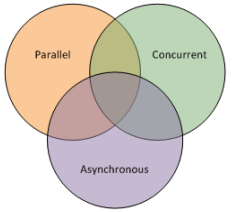
Message Passing
Agent (and Actor) model

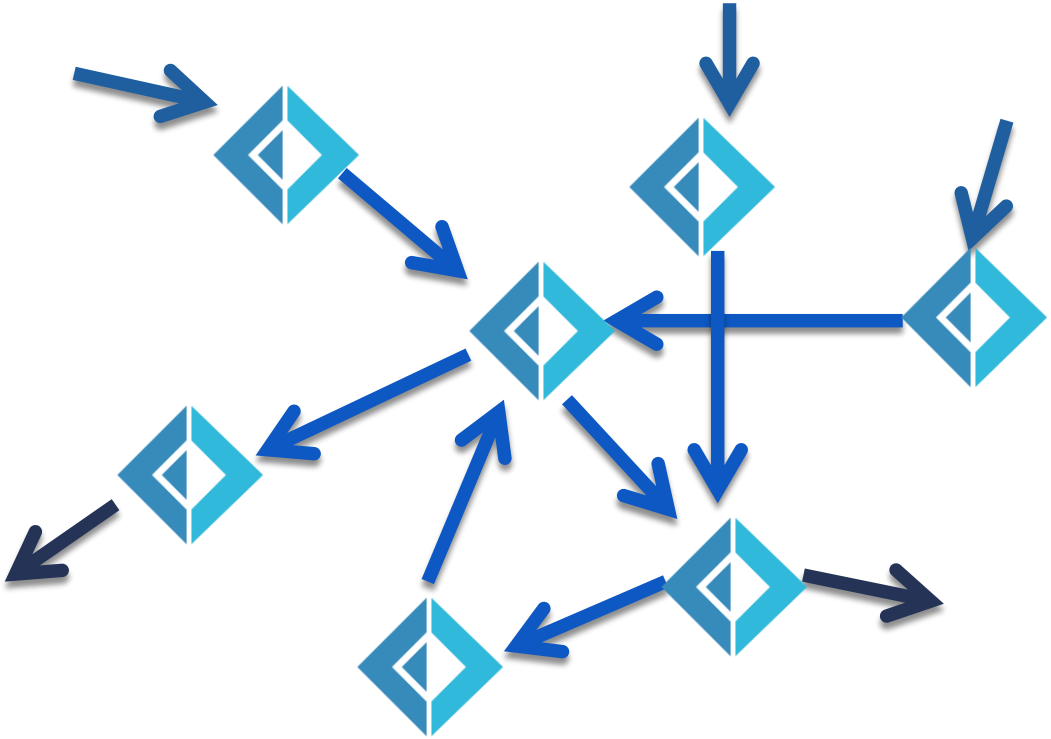
Message Passing based concurrency



- 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

Message Passing based concurrency





Agents **exchange** messages

Receive message and react

Reactive system

Handle inputs while running

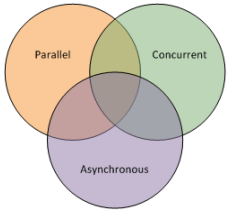
Emits results while running

The Solution is Immutability and Isolation

Immutability OR Isolation

```
let (lockfree:Agent<Msg<string,string>>) = Agent.Start(fun sendingInbox ->
    let cache = System.Collections.Generic.Dictionary<string, string>()
    let rec loop () = async {
        let! message = sendingInbox.Receive()
        match message with
            | Push (key,value) -> cache.[key] <- value
            | Pull (key,fetch) -> fetch.Reply cache.[key]
        return! loop ()
        }
    loop ())
```

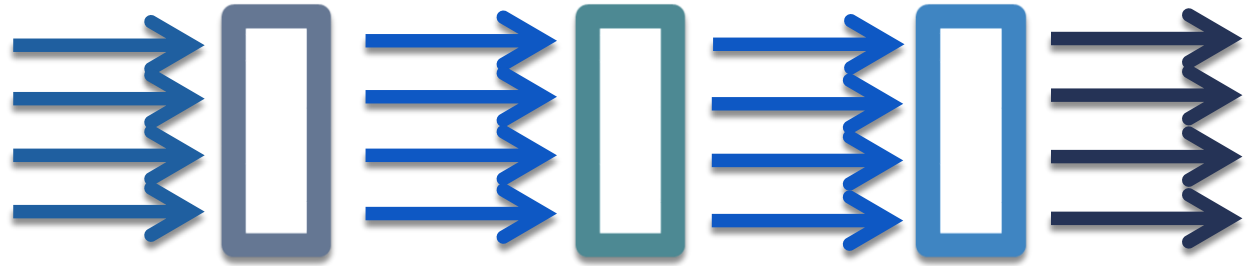
TPL DataFlow




```
var InputBlock = new BufferBlock<Tuple<string, string>>();
var splitLines = new TransformBlock<Tuple<string, st
        n => /* Code ... Transfirmation */    );
var splitWords = new TransformBlock<Tuple<string, st
        n => /* Code ... Transfirmation */    );
var fuzzyMatch = new TransformBlock<Tuple<string, st
        n => /* Code ... Transfirmation */    );
var finalBlocl = new ActionBlock<Tuple<string, strin
        n => /* Code ...  */    );

InputBlock.LinkTo(splitLines, new DataflowLinkOption
splitLines.LinkTo(splitWords, new DataflowLinkOption
InputBlock.SendAsync(message);
```

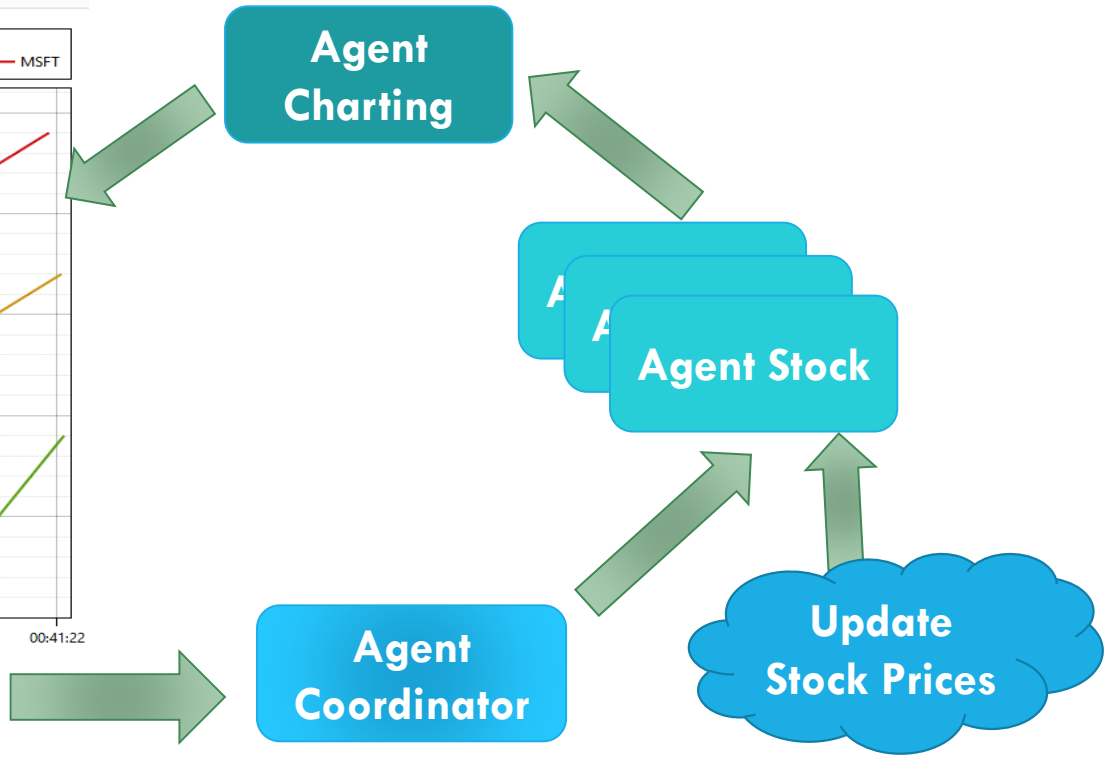
Platform	Dataflow support
.NET 4.5	✓
.NET 4.0	✗
Mono iOS/Droid	✓
Windows Store	✓
Windows Phone Apps 8.1	✓
Windows Phone SL 8.0	✓
Windows Phone SL 7.1	✗
Silverlight 5	✗

Pipeline Processing



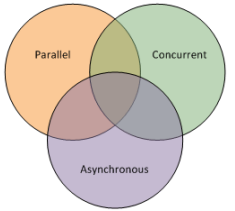
- Pipeline according to Wikipedia:
  - A pipeline is a set of data processing elements connected in series, so that the output of one element is the input of the next one. The elements of a pipeline are often executed in parallel or in time-sliced fashion; in that case, some amount of buffer storage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

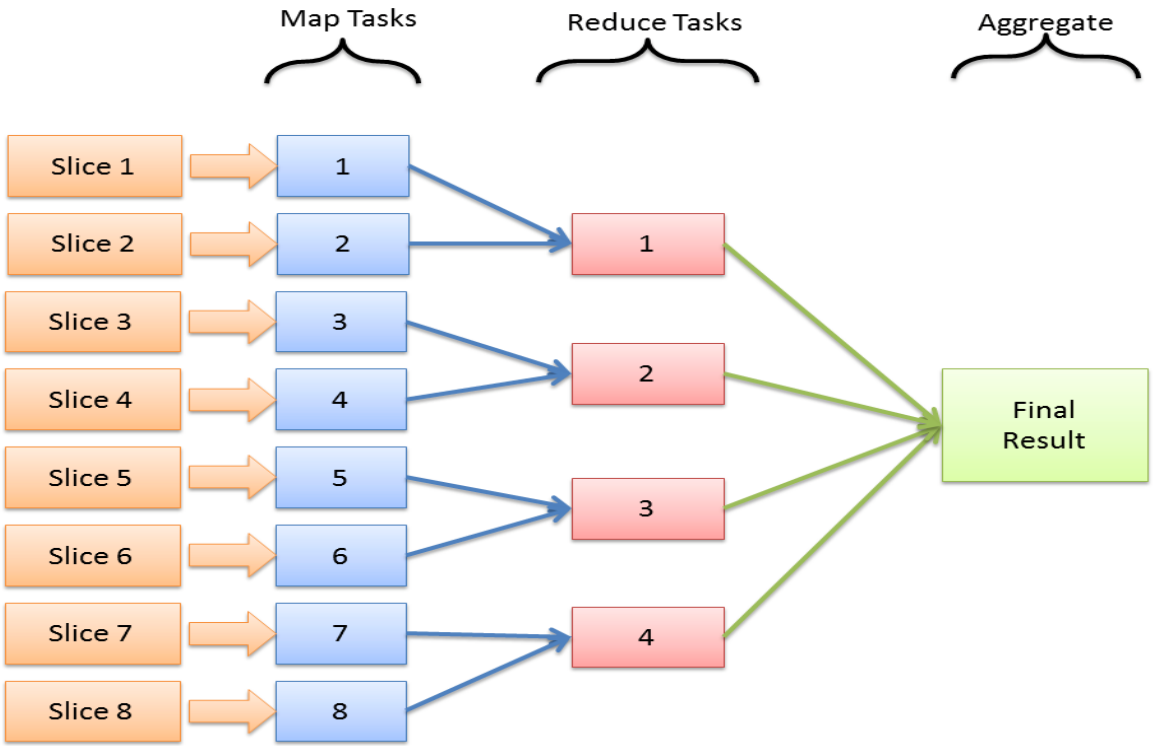
Agent Stock Ticker



Map >> Reduce

Map >> Reduce


Parallel
Concurrent
Asynchronous

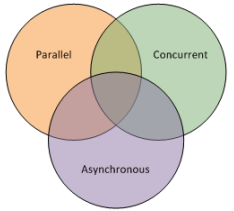

Map Tasks
Reduce Tasks
Aggregate
Slice 1
Slice 2
Slice 3
Slice 4
Slice 5
Slice 6
Slice 7
Slice 8
1
2
3
4
5
6
7
8
1
2
3
4
Final Result

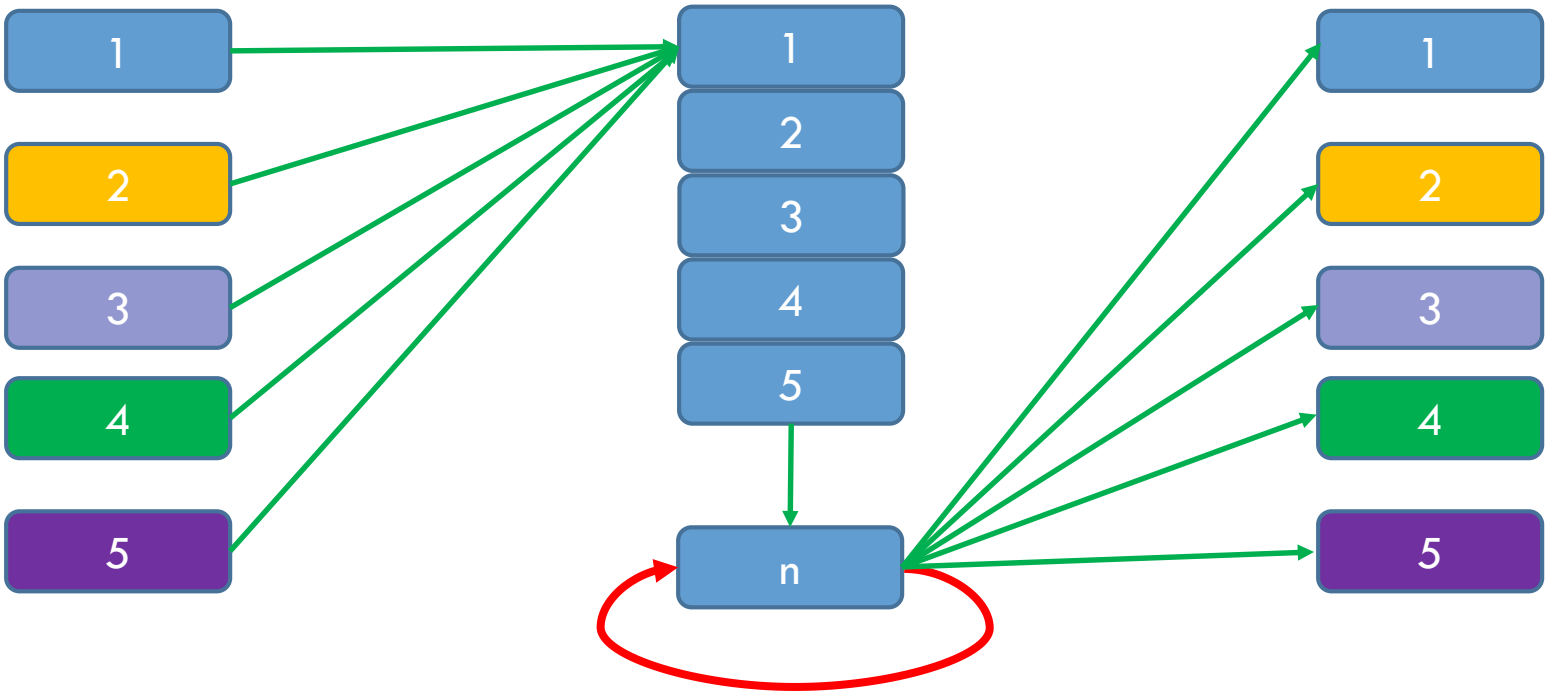
Map >> Reduce

```
var duplicates =
    Directory.GetFiles(path, "*.*", SearchOption.AllDirectories).AsParallel()
        .Select(f =>
            {   using (var fs = new FileStream(f, FileMode.Open, FileAccess.Read))
                    return new
                    {
                        FileName = f,
                        FileHash = BitConverter.ToString(SHA1.Create().ComputeHash(fs))
                    };
            })
        .GroupBy(f => f.FileHash, EqualityComparer<string>.Default)
        .Select(g => new { FileHash = g.Key, Files = g.Select(z => z.FileName)  })
        .Where(g => g.Files.Count > 1)
        //.Skip(1).SelectMany(f => f.Files)
        .ToList();
```

CHANNEL

Channel



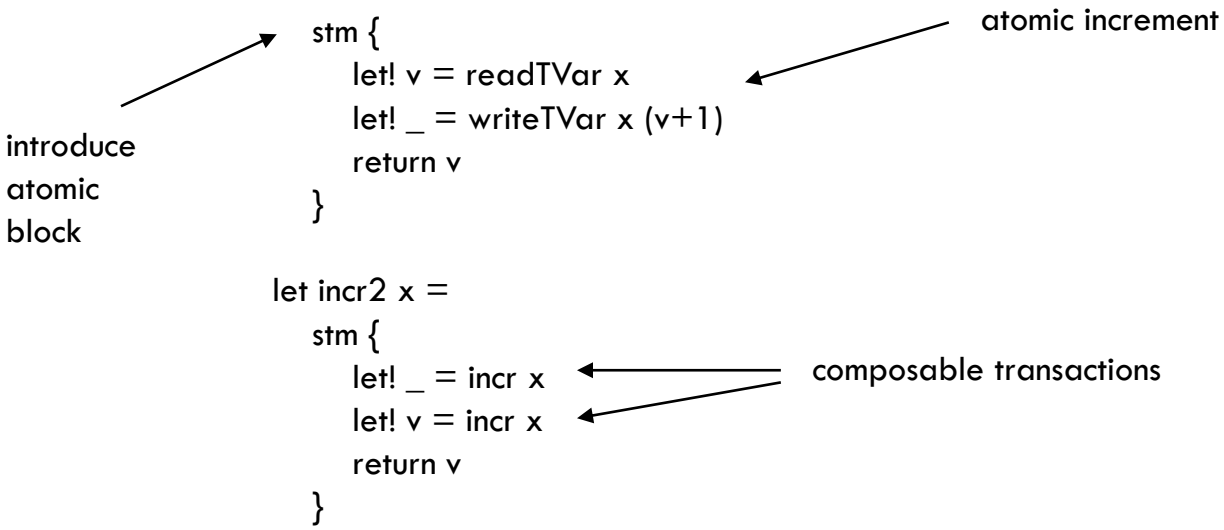


```
type [<Sealed>] Channel<'T>() =
    let channelAgent =
        MailboxProcessor<ChannelMsg<'T>>.Start(fun inbox ->
            let readers = Queue()
            let writers = Queue()
            let rec loop () = async {
                let! msg = inbox.Receive()

                match msg with
                | Read ok when writers.Count = 0 ->
                    readers.Enqueue ok
                    return! loop()
                | Read ok -> let value, cont = writers.Dequeue()
                             Pool.Spawn cont
                             ok value
                             return! loop()
                | Write (x,ok) when readers.Count = 0 ->
                          writers.Enqueue(x, ok)
                          return! loop()
                | Write (x,ok) ->  let cont = readers.Dequeue()
                                   Pool.Spawn ok
                                   cont x
                                   return! loop() }
            loop() )

    member this.Read(read) = channelAgent.Post (Read read)
    member this.Write(v, ok) = channelAgent.Post (Write (v,ok))

    member inline this.Read() = Async.FromContinuations(fun (ok, _, _) -> this.Read ok)
    member inline this.Write x = Async.FromContinuations(fun (ok, _, _) -> this.Write(x,ok))
```

atomic { STM }

Software Transactional Memory

```
val readTVar : TVar<'a> -> Stm<'a>
val writeTVar : TVar<'a> -> 'a -> Stm<unit>
val : atomically : Stm<'a> -> 'a

let incr x =
    stm {
        let! v = readTVar x
        let! _ = writeTVar x (v+1)
        return v
    }

let incr2 x =
    stm {
        let! _ = incr x
        let! v = incr x
        return v
    }
```



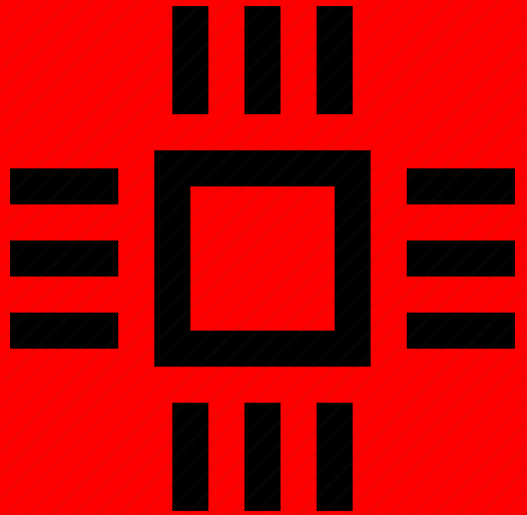
Software Transactional Memory

```
let take fork =    stm {
                      let! ref_fork = readTVar (fork)
                      do! check ref_fork
                      do! writeTVar (fork) false }

let acquire left_fork right_fork  =
              stm {
                  do! take left_fork
                  do! take right_fork } |> atomically

let release left_fork right_fork =
    stm {
        do! writeTVar left_fork true
        do! writeTVar right_fork true
    } |> atomically
```

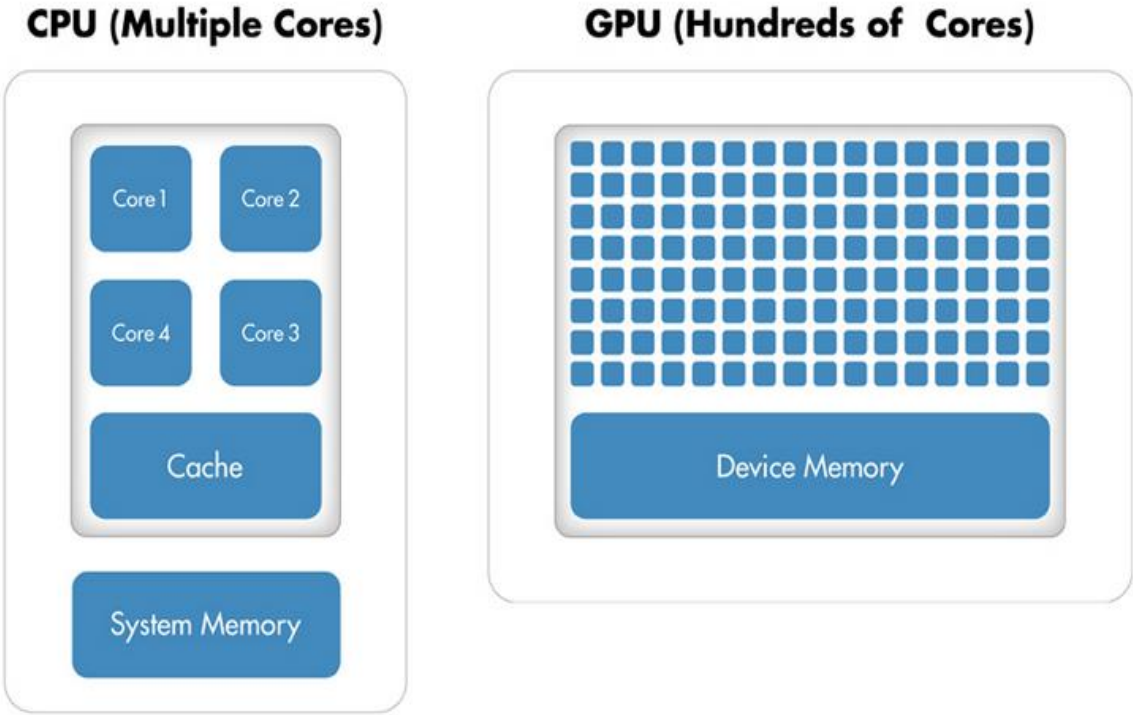
Software Transactional Memory

Hardly Successful

GP-GPU



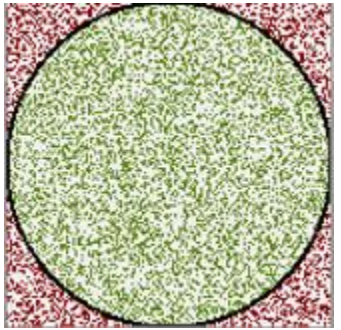
GPU


CPU (Multiple Cores)
Core 1
Core 2
Core 4
Core 3
Cache
System Memory
GPU (Hundreds of Cores)
Device Memory


AMD FireStream

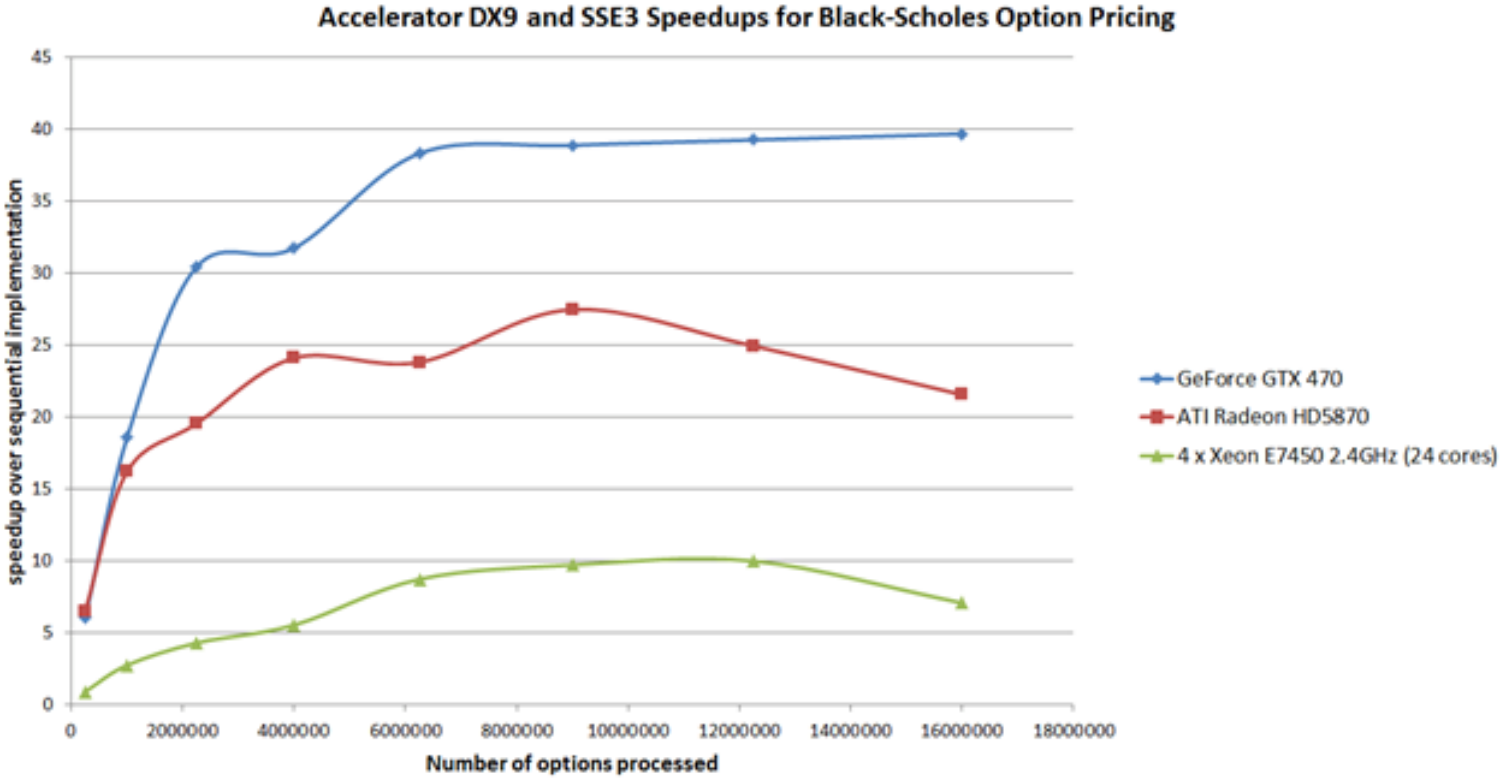
GPU – MSFT Accelerator

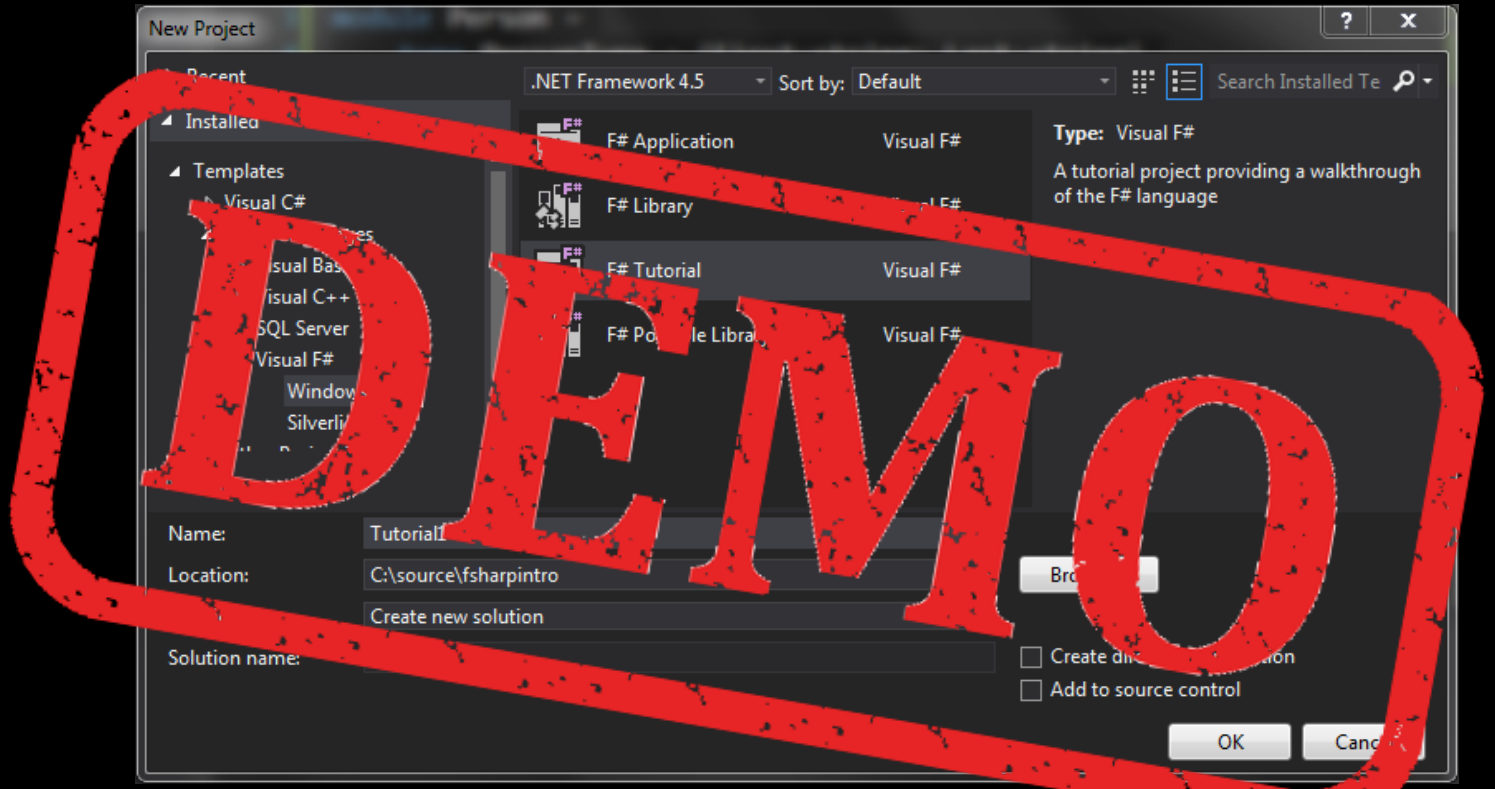
- Microsoft Accelerator
  - The project Microsoft Research Accelerator is a .Net library for developing array-based computations and executing them in parallel on multicore CPUs or more interestingly, using GPU shaders



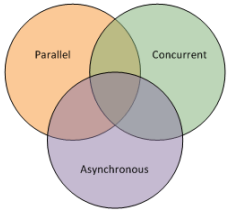
  - Accelerator handles all the details of parallelizing and running the computation on the selected target processor, including GPUs and multicore CPUs

GPU – MSFT Accelerator




New Project
.NET Framework 4.5
Sort by:
Default
Search Installed Te
Installed
Templates
Visual C#
Visual Basic
Visual C++
SQL Server
Visual F#
Windows
Silverlight
F# Application
Visual F#
F# Library
Visual F#
F# Tutorial
Visual F#
F# Portable Library
Visual F#
Type: Visual F#
A tutorial project providing a walkthrough of the F# language
Name:
TutorialF
Location:
C:\source\fsharpintro
Browse...
Create new solution
Solution name:
Create directory
Add to source control
OK
Cancel
DEMO

Summary



Embrace Parallelism exploiting the potential of multicore CPUs

No single solution…

You must combine different programming models

(*right tool for the job*)

FP really **shines** in the area of concurrency


That's all Folks!

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

-- Edsger Dijkstra

How to reach me

github.com/rikace/Presentations/AkkaActorModel

meetup.com/DC-fsharp

@DCFsharp @TRikace

tericcardo@gmail.com

https://www.surveymonkey.com/r/S6PV89W