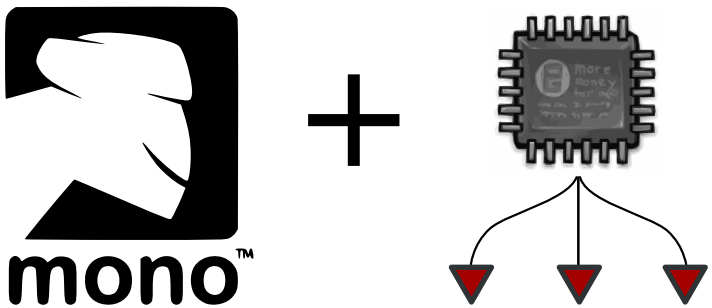
ParallelFx

Bringing Mono applications in the multicore era

Jérémie Laval
jeremie.laval@gmail.com
http://neteril.org



Outline

Why are we bothering with parallelization

Everyone loves his single thread

“The ideal number of thread you should use is 1”

– Alan Mc Govern

Why are we bothering with parallelization

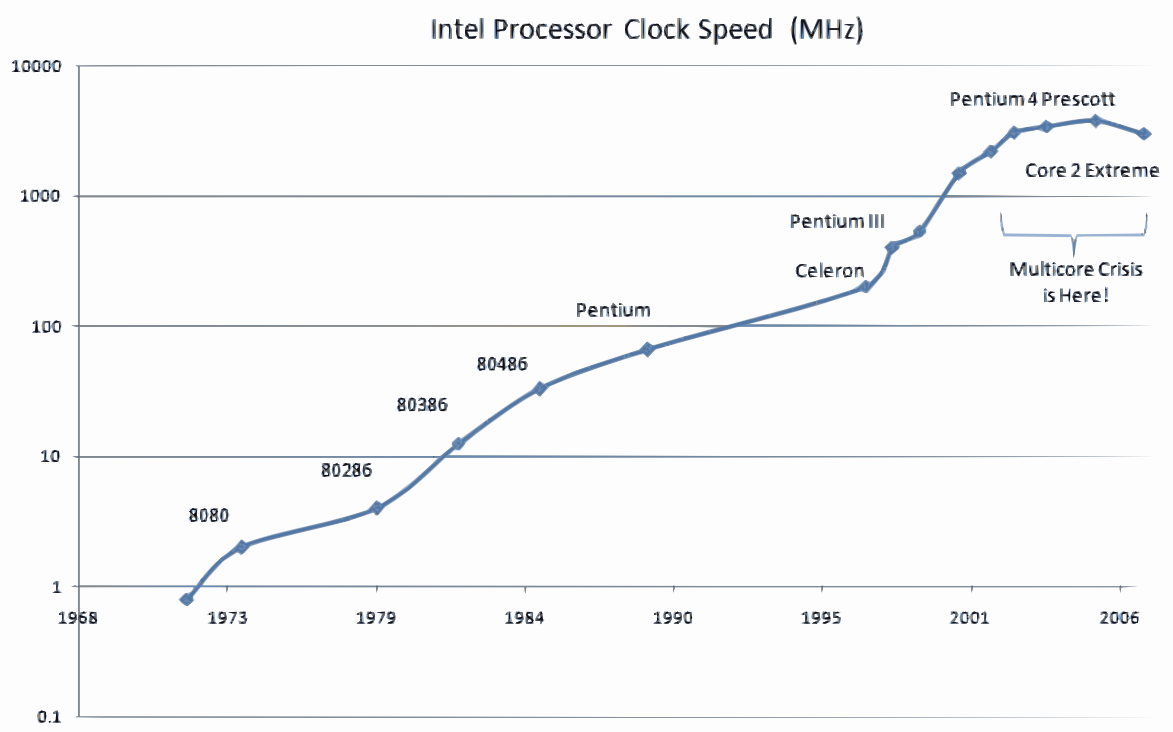
Because the free lunch is over!

Source: Sidsel Jensen

$\rightarrow$ We need to change the way we code

Free lunch ?

70's - 2005 : Moore's law in action

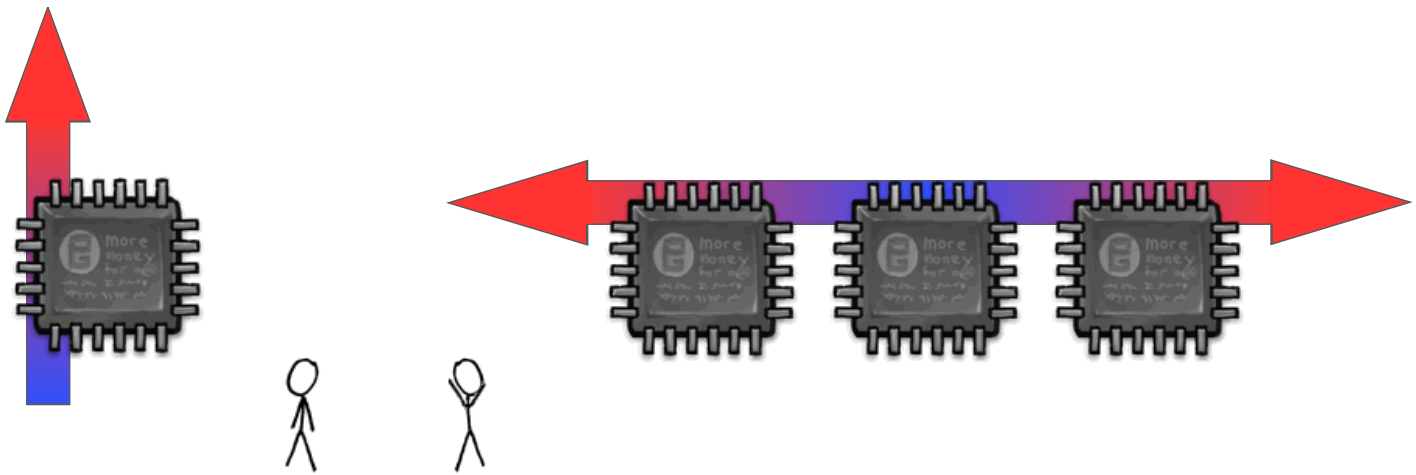


Source: Smoothspan blog

Bottom line : clock speed war is a no-go

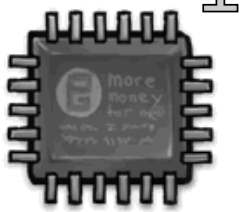
The awesome idea

Can’t scale vertically ? Scale horizontally !

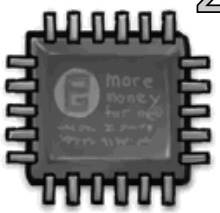


Trend of things

Number of cores

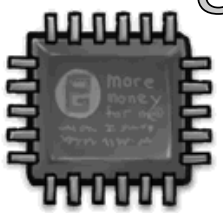


Pentium

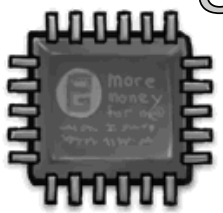


Core Duo

Core 2 Quad



i7



Intel prototype

Solution

Let's break up work...

....and share it among cores

Errr...


Well, it all started
with a loose thread
ICANHASCHEEZBURGER.COM

Parallelization is difficult

- Hard: is that stuff really thread safe ?
- Tedious: whose turn is it to debug deadlocks ?
- Inefficient: how many thread to use ?
  - Too few: it's not scaling
  - Too much: context-switching hurts performance
  - What if the number of core changes ?

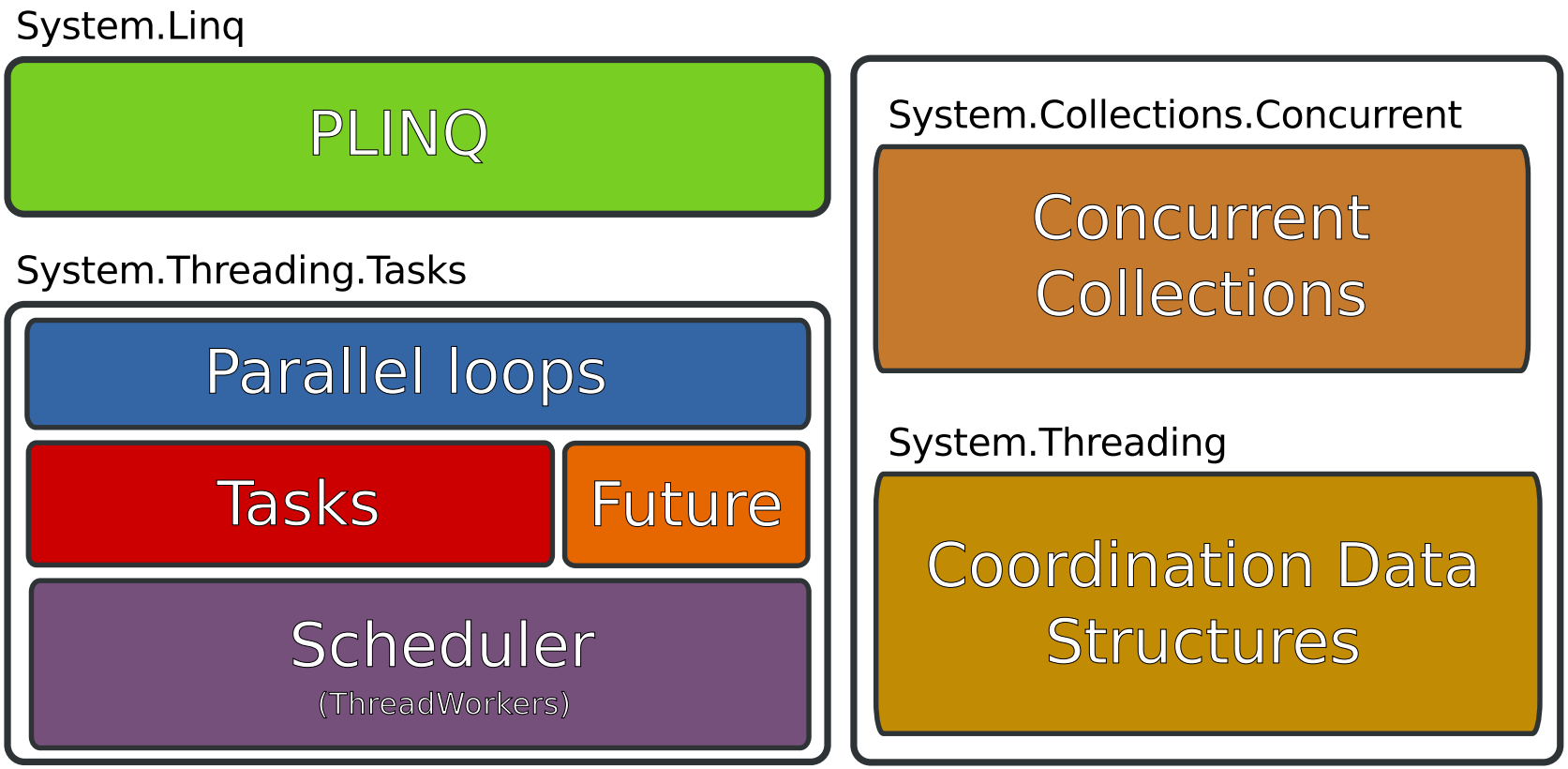
KISS time

We need something different :

- Automagically regulate thread usage at runtime
- As straightforward as possible
  - Mimicking familiar constructs
  - Reuse existing code with only slight modification

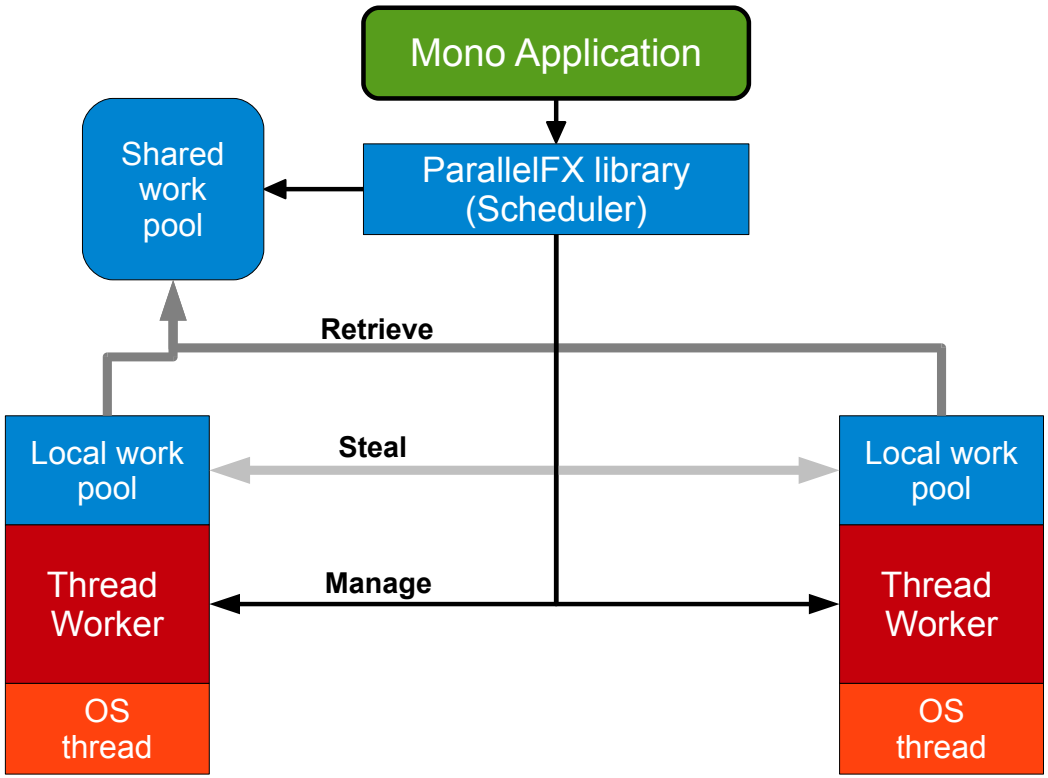
Enter ParallelFx

ParallelFx at a glance



At the heart

Work-stealing scheduler



Tasks

```
CancellationTokenSource source = new CancellationTokenSource ();
Task task = Task.Factory.StartNew (() => DoSomeStuff (), source.Token);
Task continuation = task.ContinueWith ((t) => Console.WriteLine ("task finished");
source.Cancel ();
t.Wait ();
```

Future (Task<T>)

```
static int SumParallel (Tree<int> tree, int curDepth)
{
  const int SequentialThreshold = 3;

  if (tree == null) return 0;

  if (curDepth > SequentialThreshold)
     return SumSequentialInternal (tree);

  int right = SumParallel (tree.Right, curDepth + 1);

  Task<int> left =
             Task.Factory.StartNew (() => SumParallel (tree.Left, curDepth + 1));

  return tree.Data + left.Value + right;
}
```

Parallel.For

```
Fractal fractal = new Fractal (width, height);

ColorChooser colorChooser = new ColorChooser ();

Parallel.For (0, width, (i) => {
  for (int j = 0; j < height; j++) {
    ProcessPixel (i, j, fractal, colorChooser);
  }
});
```

Demo: Parallel For

Demo: image processing with parallel loops

PLINQ

```
ParallelEnumerable.Range (1, 1000)./* ... */
ParallelEnumerable.Repeat (``Rupert'', 1000)./* ... */
enumerable.AsParallel ()./* ... */

var query = from x in Directory.GetFiles ("/etc/")
            .AsParallel ()
            where x.EndsWith (".conf")
            select x;

query.ForAll ((e) => {
  Console.WriteLine ("{0} from {1}", e,
                     Thread.CurrentThread.ManagedThreadId);
});
```

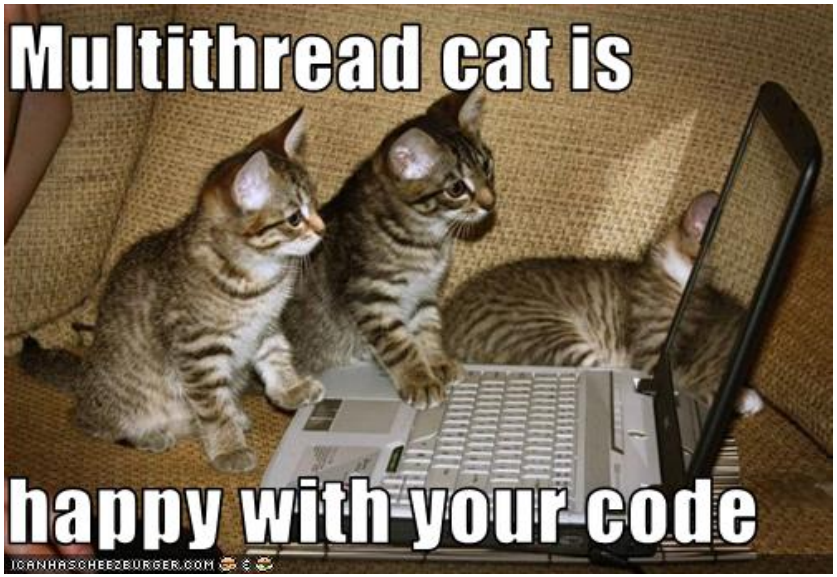
Demo: PLINQ

Demo: raytracing the PLinq way

State of things

- Mono 2.6 (December 2009) : .NET 4 beta 1
  - Task, Future, Parallel loops
  - Concurrent collections
  - Coordination data structures
- Mono trunk (1 week ago) : .NET beta 2
- Mono 2.8 ? : .NET 4 compliant
  - With (hopefully) PLINQ !

Questions



Any questions ?