Enabling Parallelism for Productivity Programmers with Patterns, Frameworks and Languages

Ade Miller
Principal Program Manager
Microsoft

Microsoft
patterns & practices
proven practices for predictable results

A Bit About Me…

- Currently working with the Technical Computing Group
  - HPC and parallel computing platform(s)
- Previously I was part of the patterns & practices team
  - Helping developers be successful on Windows
  - Including guidance on parallel programming

Questions I've Been Asking…

- How do we express parallelism through:
  - Languages?
  - Frameworks?
  - Patterns?
- What's the best way to help developers be successful?
- What's missing in the things we give them today?

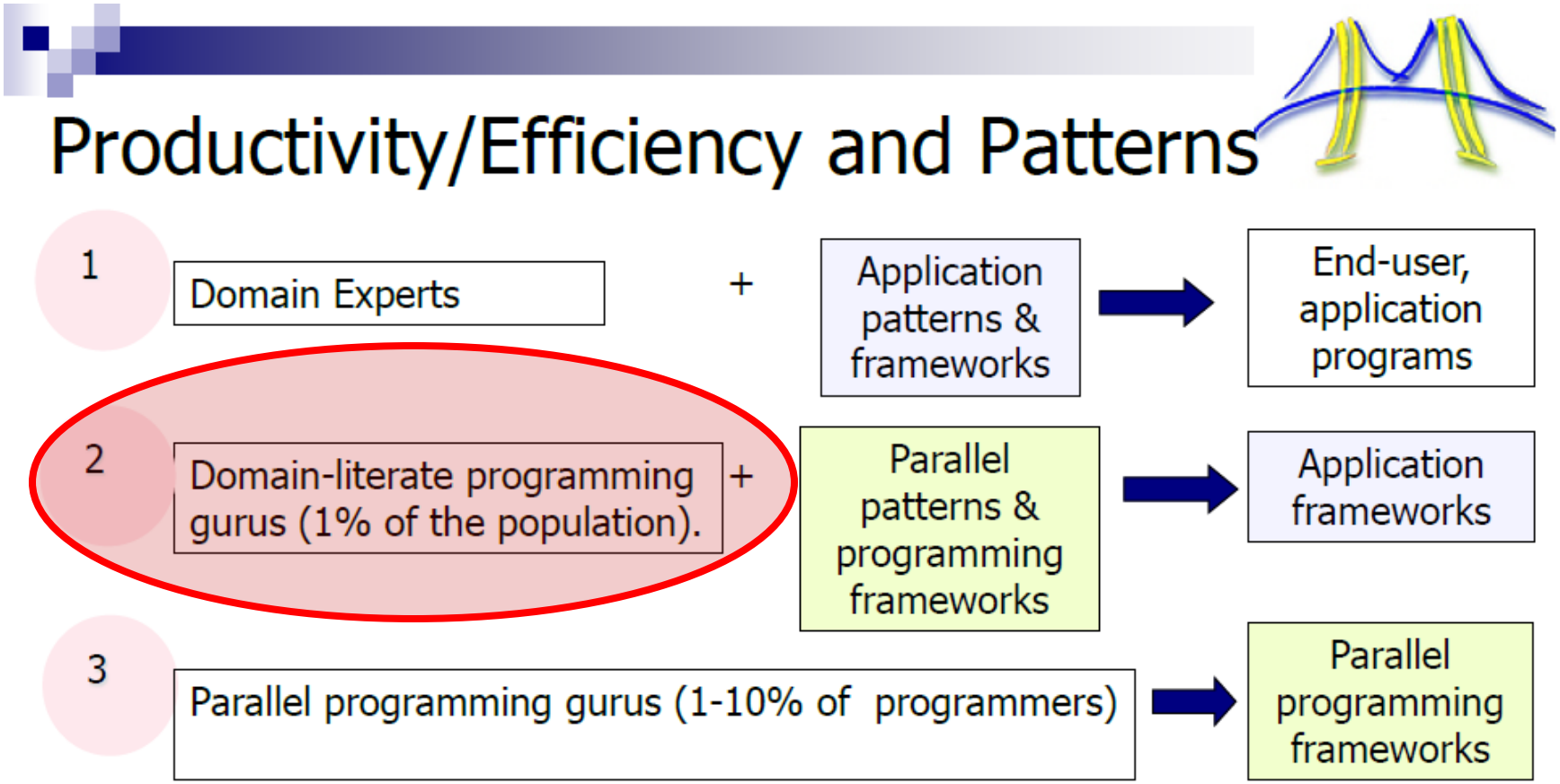
Agenda

- Who are Windows developers?
- Languages
  - New language features
- Frameworks
  - .NET
  - Native code
- Patterns
- Conclusions

Developer Ecosystem

- Managed .NET Languages
  - C#
  - VB.NET
  - F#
  - (Managed C++/CLI)
  - Iron Python
- Native Languages
  - C++
  - CPython

Where Do They Fit?



Overview of the UC Berkeley Par Lab – Patterson, 2009

.NET Programmers

- Not all solving traditional HPC problems
  - Some are; e.g. Financial applications
- Some focused on concurrency not parallelism
  - Building more responsive user interfaces
- Looking to improve performance
  - 3x speedup on 4 cores is “OK”
- New to parallel programming
  - ~5% of developers familiar with parallel programming

Native C++ Programmers

- Performance programmers
  - Numerical computation with MPI & OpenMP
- Large software vendors (ISVs)
  - Large applications where performance matters
- Targeting new devices
  - Tablets & phones

LANGUAGES

.NET Language Features

- Lambdas – unnamed functions

```
(a, b) => { return a + b; }
```

- Extension methods

```
static bool Find(this IEnumerable<T> me)
{
  // ...
}
```

- **`async`** and **`await`** keywords
  - Language level support for futures

C++ Language Features

- Lambdas rather than function objects

```
[](int a, int b){ return a + b; }
```

- Type inference with the **`auto`** keyword

```
auto func = [](int a, int b){ … }
```

FRAMEWORKS

Frameworks in Visual Studio 2010

- Task Parallel Library (TPL) for .NET 4
  - Task based parallel programming
- Parallel LINQ (PLINQ) for .NET 4
  - Declarative query based parallelism

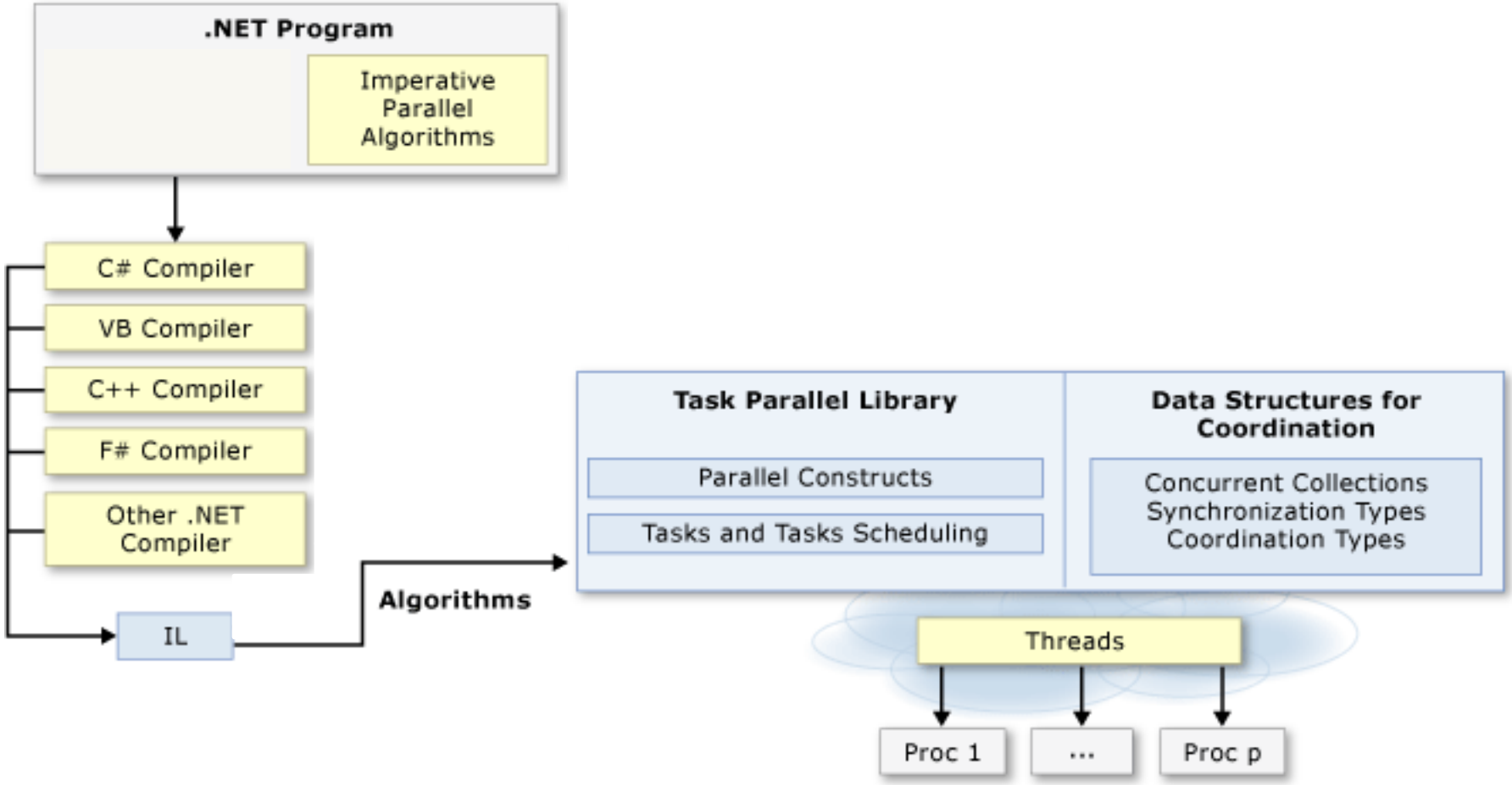
- Parallel Patterns Library for Visual C++
  - STL-like library of containers and algorithms
- Asynchronous Agent Library for Visual C++
  - Programming with agents, buffers and messages

Task Parallel Library for .NET

Express *potential parallelism* with…

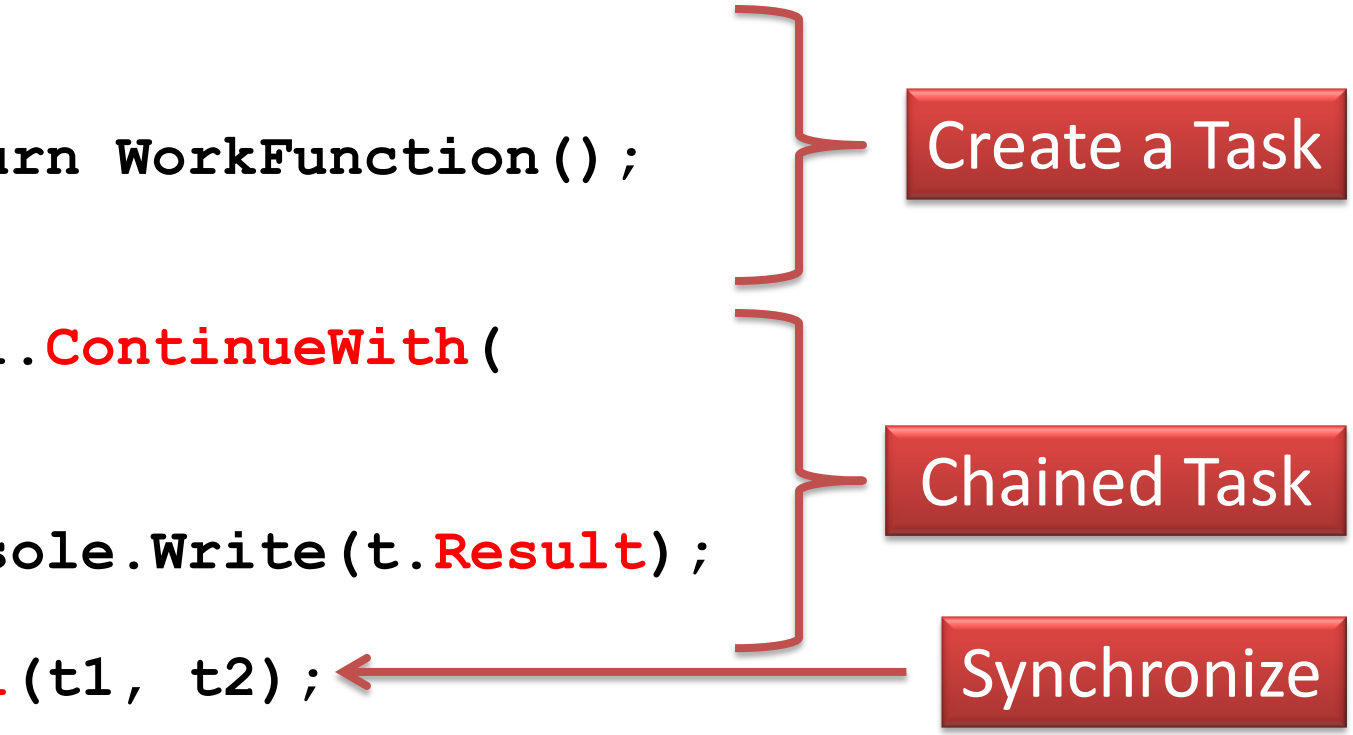
- Task based abstraction over a thread pool and scheduler
  `Task.Factory.StartNew(…)`
- Additional support for parallel loops
  `Parallel.ForEach(…)`
  `Parallel.For(…)`
- Provides concurrent containers like:
  `ConcurrentQueue<T>`
  `BlockingCollection<T>`

Task Parallel Library for .NET



Task Parallel Library Code

```
var factory = new TaskFactory();

Task<int> t1 = factory.StartNew<int>(
     () =>
     {
          return WorkFunction();
     });

Task t2 = t1.ContinueWith(
     (t) =>
     {
          Console.Write(t.Result);
     });
Task.WaitAll(t1, t2);
```



C# 5.0 **async** and **await**

```
async void OnClick(object sender,
  RoutedEventArgs e)
{
  string url = "http://...";
  string content = await new
    WebClient()
      .DownloadStringTaskAsync(url);

  Console.Write(content);
}
```

Create a Task

Process result

What is LINQ?

- Language Integrated Query
- Query support within .NET languages
- Back end providers for…
  - LINQ to SQL
  - LINQ to XML
  - LINQ to objects
  - …
- (Quaere is LINQ for Java)

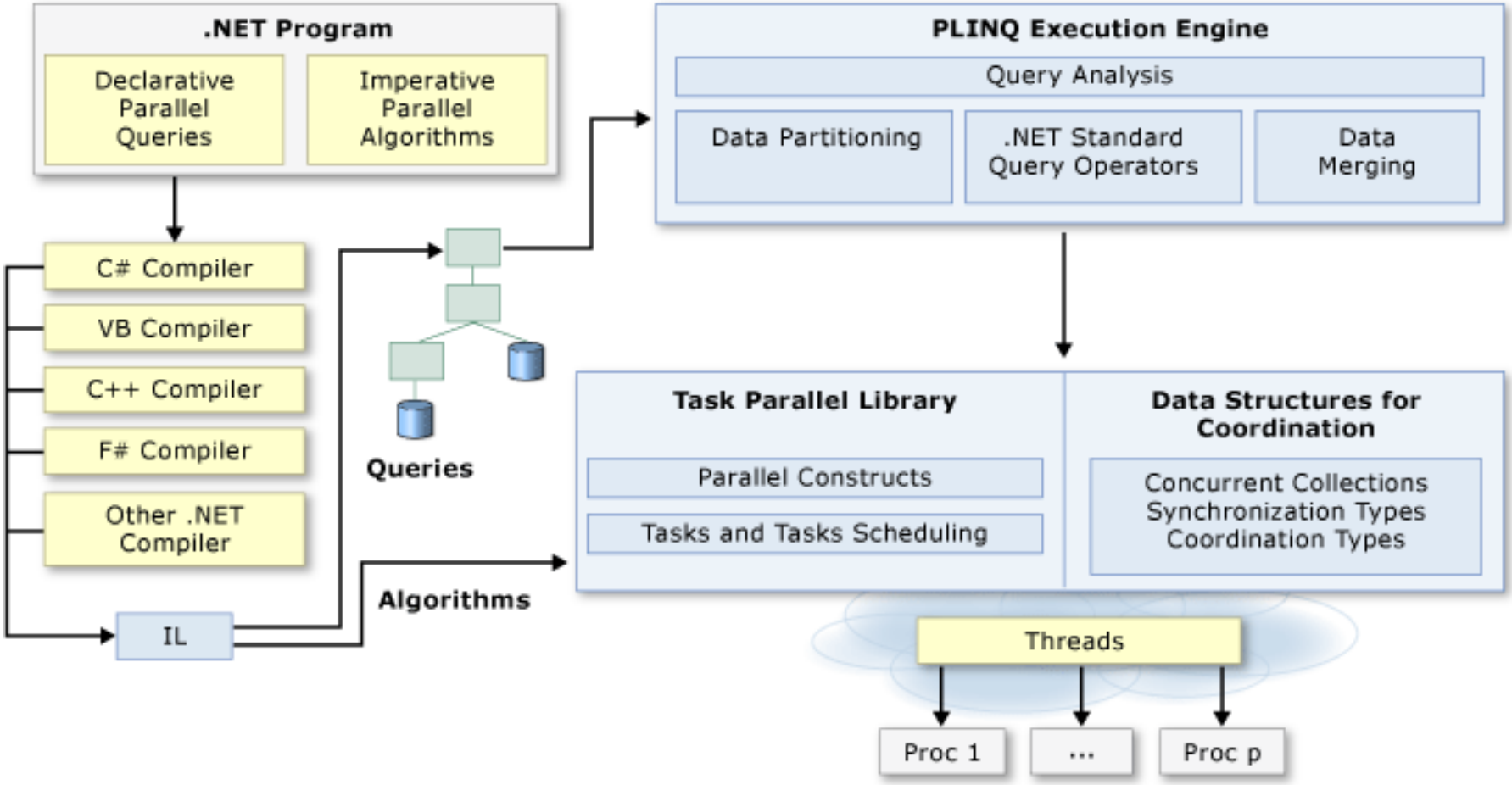
LINQ Code

```
string[] names = { "Burke", "Frank",
    "Everett", "Albert", "George", "David" };

IEnumerable<string> query =
  from name in names
    where (name.Length == 5)
    orderby name
    select name.ToUpper();

foreach (string item in query)
    Console.WriteLine(item);
```

Create Query

Enumerate

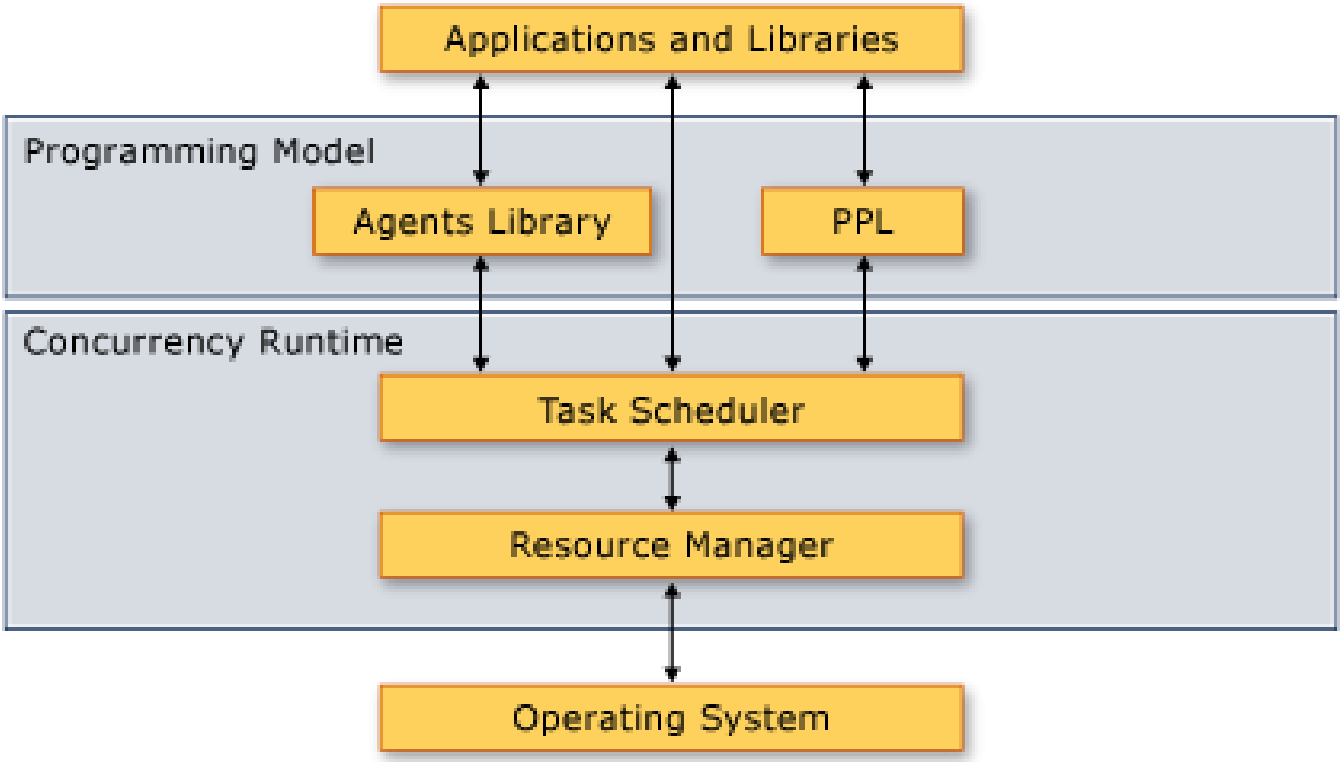
Parallel LINQ for .NET



Parallel LINQ Query Code

```
IEnumerable<string> query =
  from name in names.AsParallel()
    where (name.Length == 5)
    orderby name
    select name.ToUpper();
```

Concurrency Runtime for C++

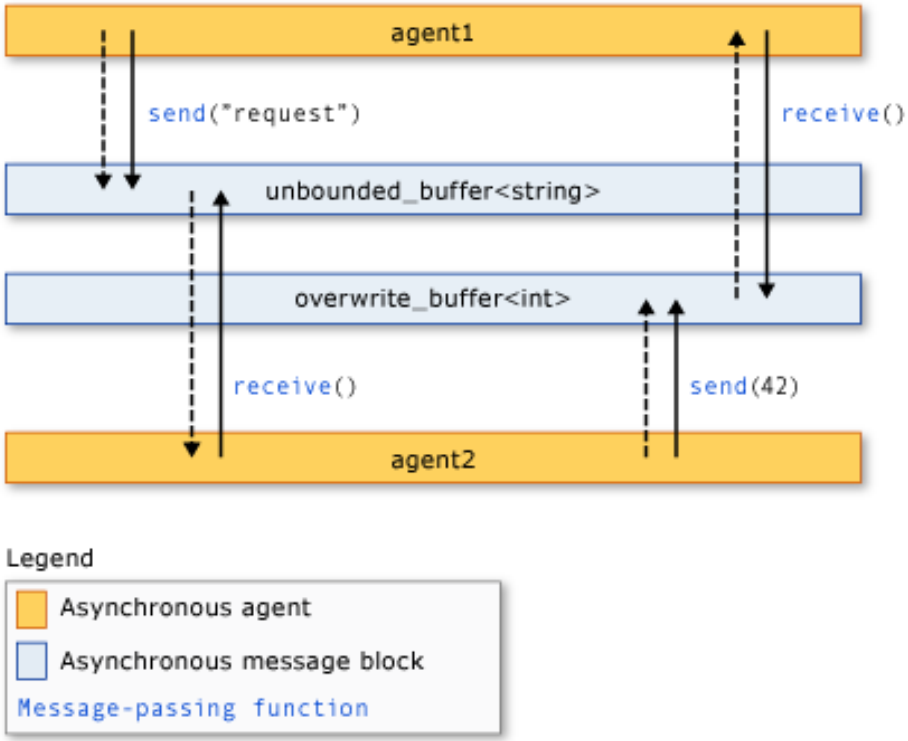


Parallel Patterns Library for C++

- Task based abstraction over a thread pool and scheduler
  `task_group.run(…)`
- STL-like algorithms
  `parallel_for_each(…)`
  `parallel_for(…)`
- Asynchronous Agent model
  `agent.start(…)`
- Provides concurrent collections like:
  `concurrent_vector<T>`
  `unbounded_buffer<T>`

Async Agents Library for C++

- Isolated agents
- Async communication
  - Message blocks
  - Message passing



Async Agents Example

```
class MyAgent : public agent
{
private:
    ITarget<wstring>& m_outputBuffer;

public:
    MyAgent(ITarget<wstring>& phraseOutput) {}

    void run()
    {
        wstring inputPhrase;
        do
        {
            wstring output = DoWork();
            asend(m_outputBuffer, output);
        }
        while (output != "");
        done();
    }
};
```

Declare a buffer

Send message

Finish

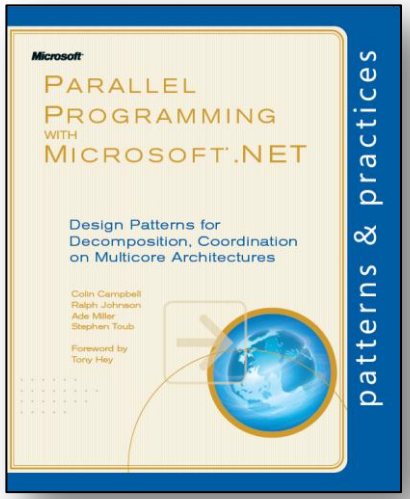
PATTERNS

Parallel Programming With…

.NET

- Task Parallel Library
- PLINQ

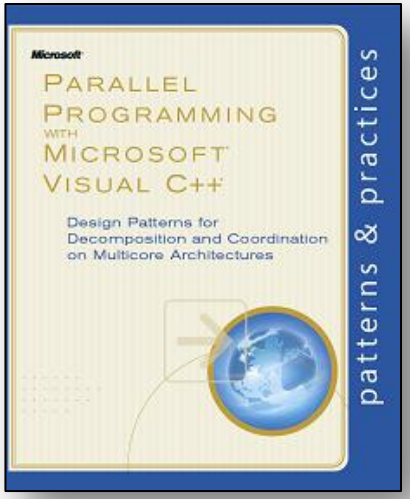
Samples for C#, VB & F#
(Available now)
http://parallelpatterns.codeplex.com/



Visual C++

- Asynchronous Agents Library
- Parallel Patterns Library

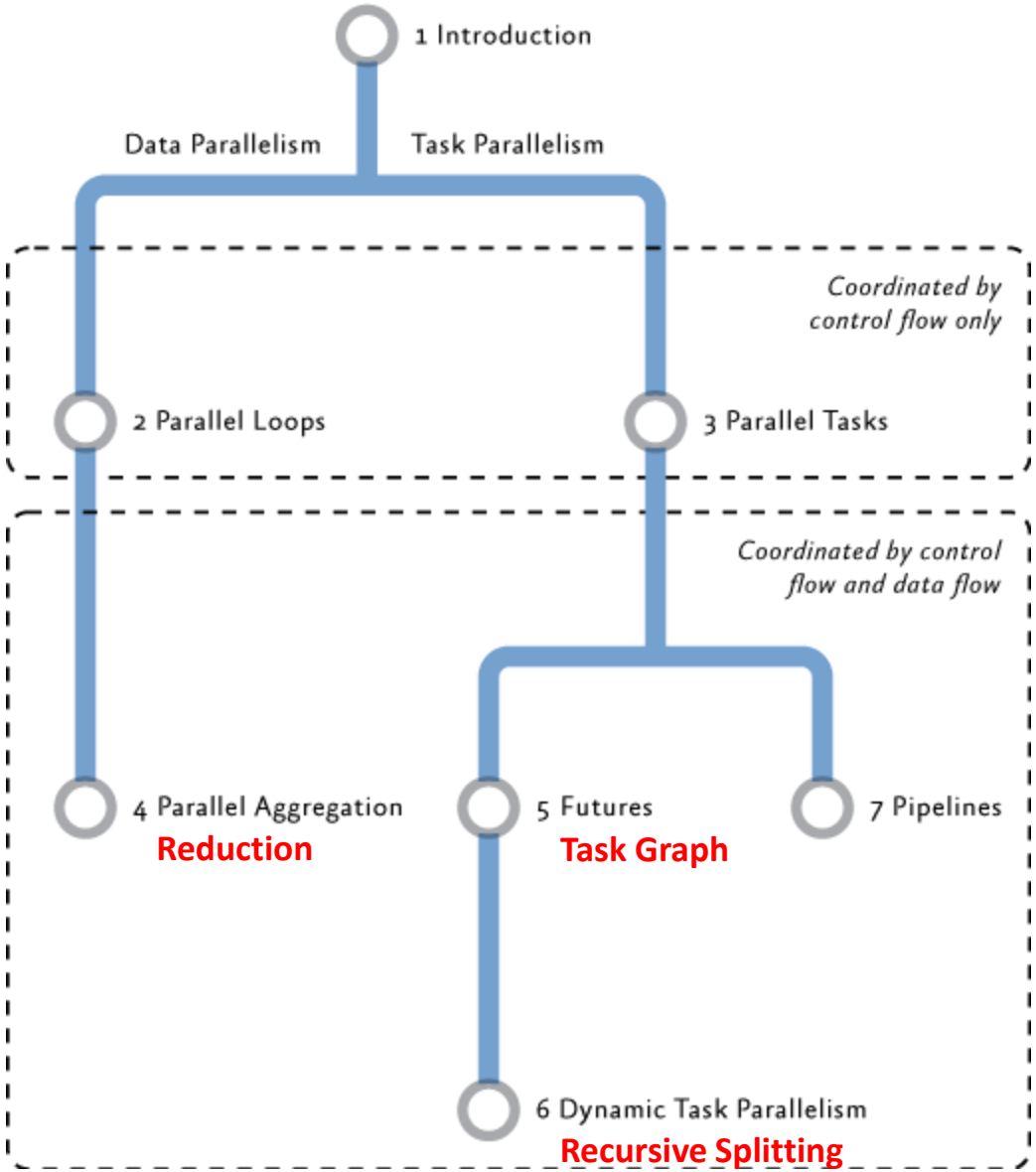
(Available April 2011)
http://parallelpatternscpp.codeplex.com/



Microsoft patterns & practices

The Patterns

- Taken from OPL
- Common Scenarios
- Supported by the frameworks



Aggregation

- Three different approaches

Parallel Aggregation with TPL

```
object lockObject = new object();
int[] data = ...
int sum = 0;

Parallel.ForEach(
    data,                                          // Pass in data
    () => 0,                                       // Initialize sum
    (x, loopState, partialSum) => {
       return Normalize(x) + partialSum;
    },
    (localPartialSum) =>
    {
       lock(lockObject) {
         sum += localPartialSum;
       }
    });
```

Partial Sums

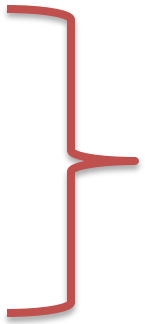
Aggregate

Parallel Aggregation with PLINQ

```
int[] data = ...
from x in data.AsParallel() select Normalize(x))
    .Aggregate(0.0d, (y1, y2) => y1 + y2);
```

Parallel Aggregation

```
vector<int> data;
combinable<int> sum([]()->int { return 0; });

parallel_for_each(data.cbegin(),data.cend(),
   [&](int x)
   {
     sum.local() += Normalize(x);
   });

return sum.combine(plus<int>());
```



Partial Sums

Aggregate

Pattern Implementation

	.NET	Native C++
Parallel Loops	`Parallel.For` `Parallel.ForEach`	`parallel_for` `parallel_foreach`
Parallel Aggregation	`PLINQ or` `Parallel.ForEach`	`combinable<T>`
Parallel Tasks	`Task`	`task_group`
Futures	`Task<T>`	`samples::Future<T>`
Dynamic Task Parallelism	`Task`	`task_group`
Pipelines	`Task` `ConcurrentQueue<T>`	`agent` `unbounded_buffer<T>`

CONCLUSIONS

Programmer Pitfalls

- Sharing state
- Tasks are not threads
- Cooperative cancellation
- Exception handling
- Understanding the scheduler

Shared State

- .NET and C++ don’t disallow shared state
- Provide features to make managing shared state easier
  - Collections
  - Messaging
  - Synchronization primitives
- Don’t prevent developers doing the wrong thing

Cooperative Cancellation with TPL

```
CancellationTokenSource cts = new CancellationTokenSource()
var options = new ParallelOptions { CancellationToken = cts.Token };
try
{
  Parallel.For(0, 1000, options, (i) =>
  {
    // ...
    if (cts.Token.IsCancellationRequested)
    {
      return;
    }
  });
}
catch (OperationCanceledException ex)
{
  // ... handle the loop cancellation
}
```

Handle cancel

Clean up

Exception Handling with TPL

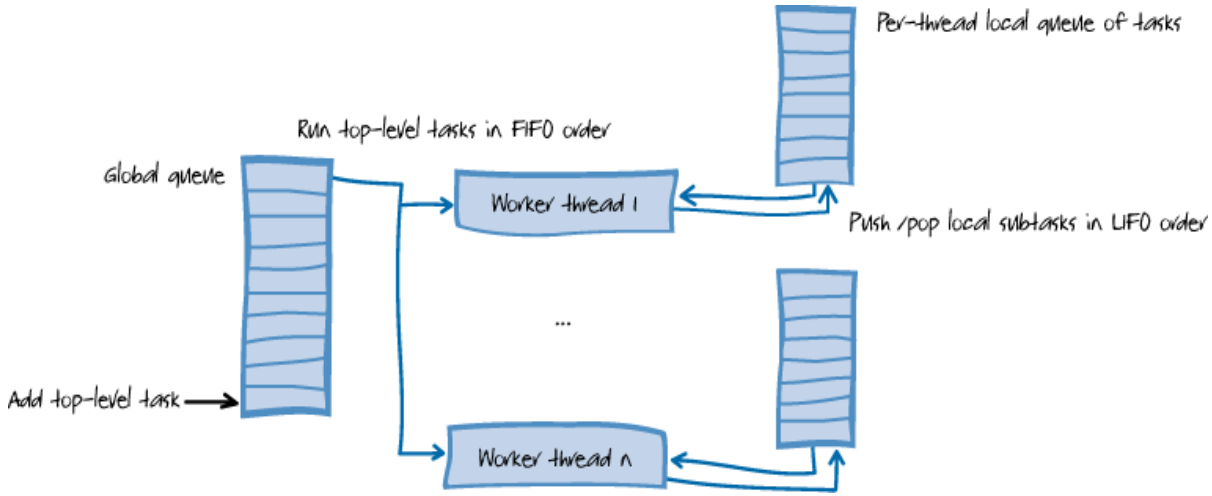
```
Task t1 = Task.Factory.StartNew(DoLeft);
Task t2 = Task.Factory.StartNew(DoRight);

try
{
  Task.WaitAll(t1, t2);
}
catch(AggregateException ae)
{
  // Handle exceptions...
  ae.Flatten().Handle(e => e is
    OperationCanceledException);
}
```

Exception observed

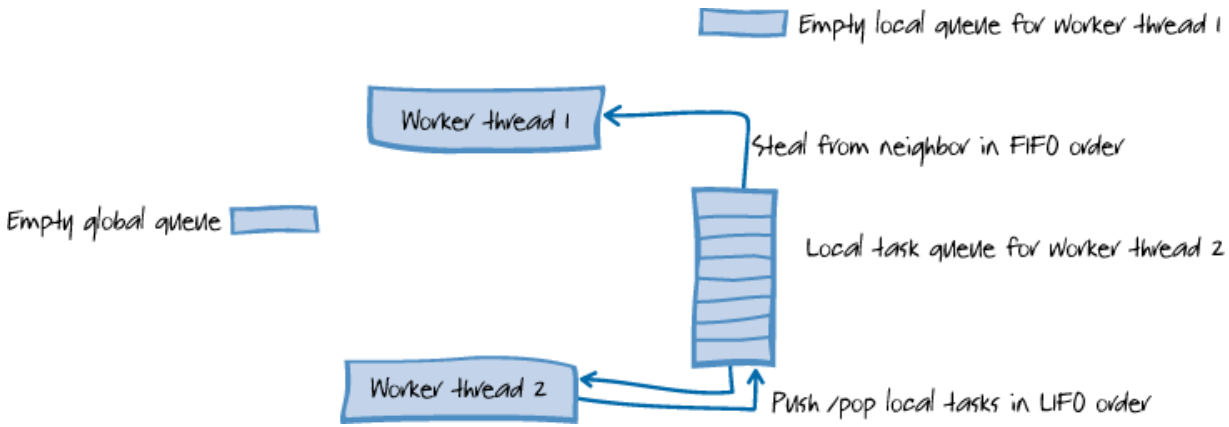
Handle exceptions

Understanding the Scheduler

Local thread queues:



Work stealing:

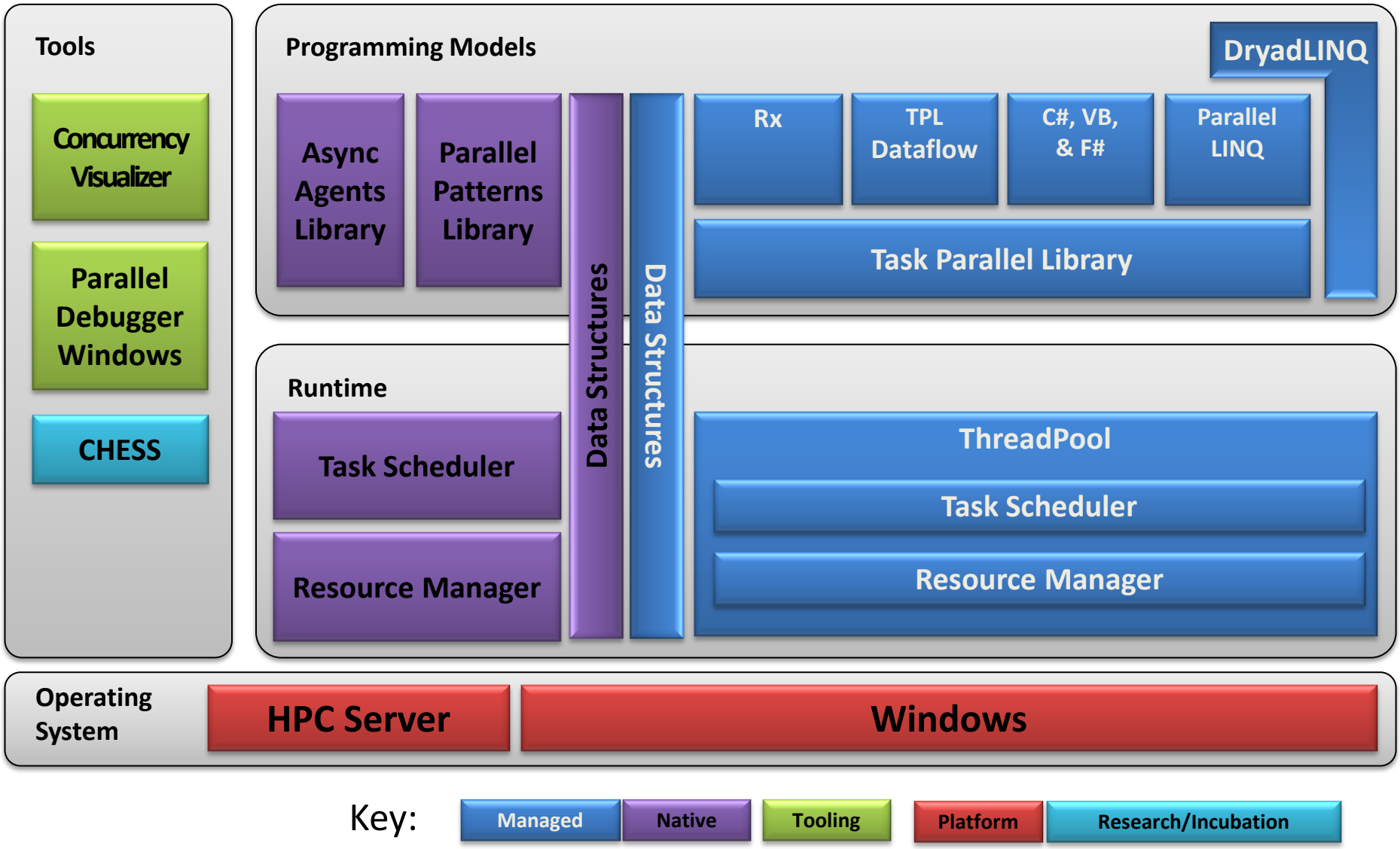


Conclusions

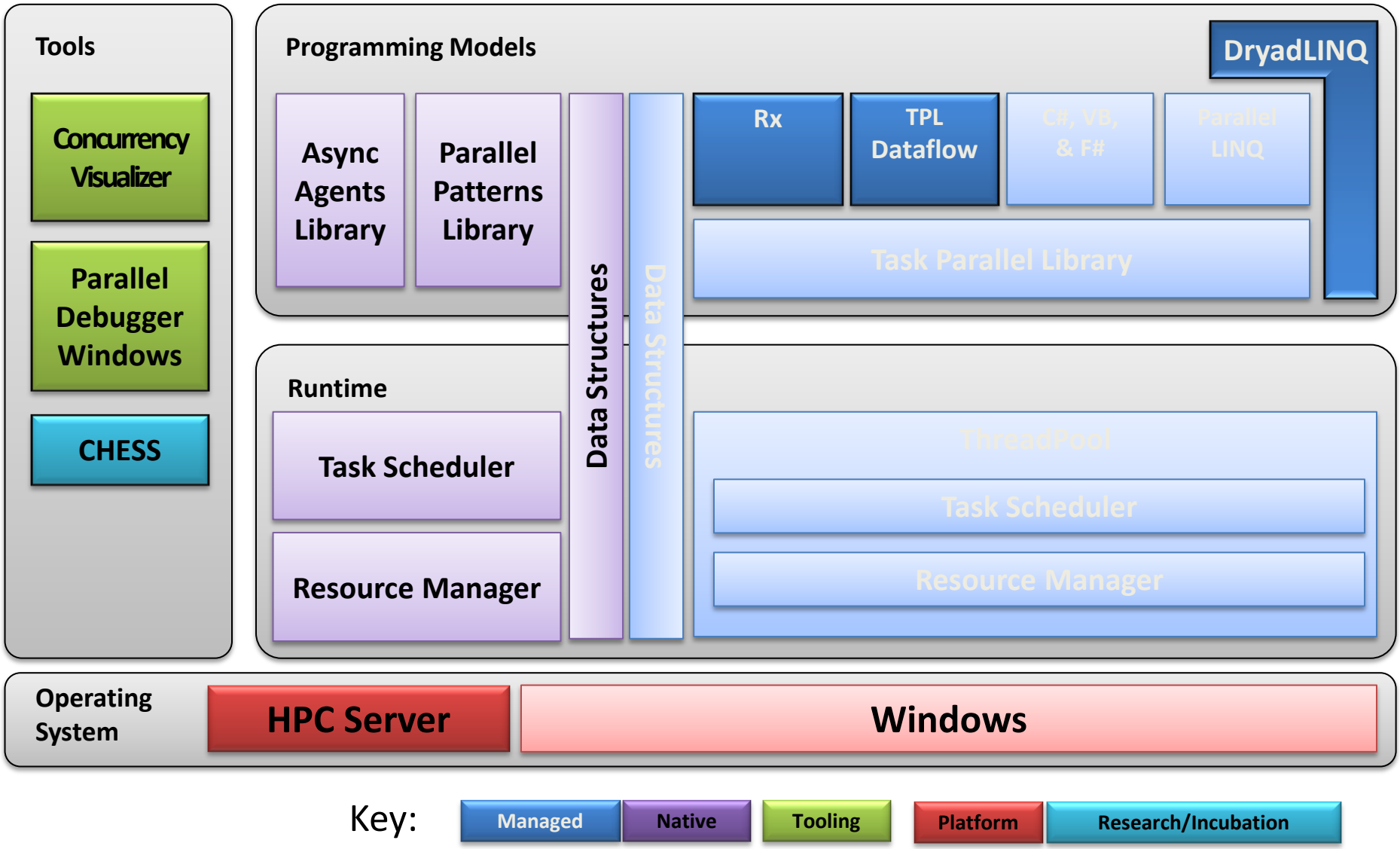
- Programming abstraction is still “leaky”
- We can add more frameworks and language features
- Patterns are a good place to start

- Maybe it’s time to remove some things?

Road Map


Tools
Concurrency Visualizer
Parallel Debugger Windows
CHESS
Programming Models
DryadLINQ
Async Agents Library
Parallel Patterns Library
Data Structures
Data Structures
Rx
TPL Dataflow
C#, VB, & F#
Parallel LINQ
Task Parallel Library
Runtime
Task Scheduler
Resource Manager
ThreadPool
Task Scheduler
Resource Manager
Operating System
HPC Server
Windows
Key:
Managed
Native
Tooling
Platform
Research/Incubation

Road Map

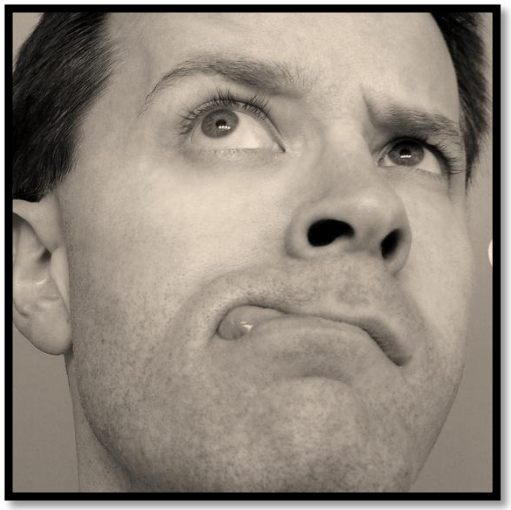


Key: Managed | Native | Tooling | Platform | Research/Incubation

Other Resources

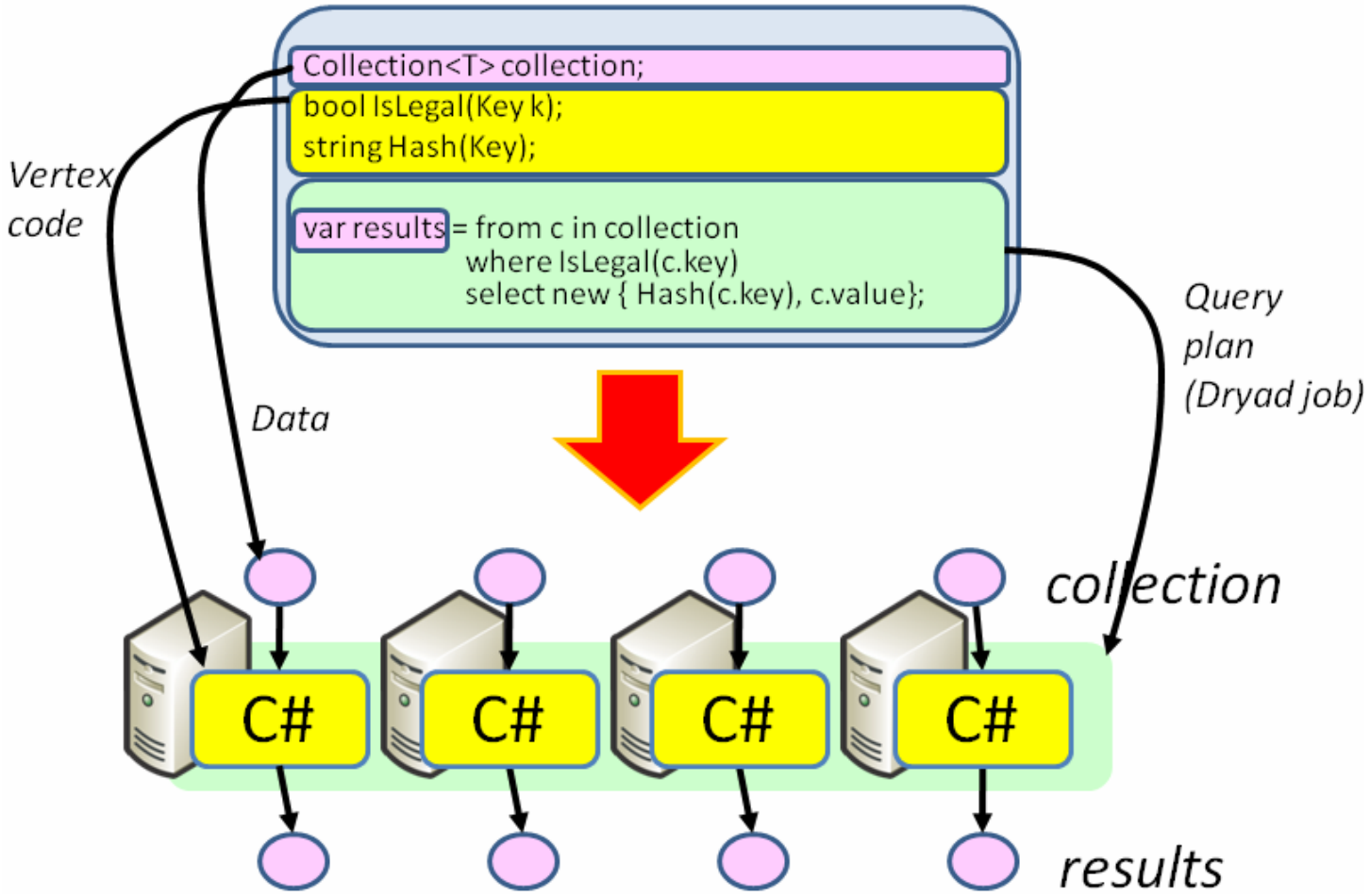
- It's all here… http://ademiller.com/tech/
  - Patterns books and code
    - http://parallelpatterns.codeplex.com/
    - http://parallelpatternscpp.codeplex.com/
  - Visual Studio
    - Python Tools: http://pytools.codeplex.com/
    - Async: http://msdn.microsoft.com/vstudio/async.aspx
  - Technical Computing Labs "Dryad"
    - http://msdn.microsoft.com/en-us/devlabs/tclabs
    - http://www.modelingtheworld.com/

Thoughts? Questions?


SUPPORTING SLIDES

Distributed LINQ with Dryad



Distributed LINQ Queries

```
IEnumerable<string> query =
    from name in connection.FromDsc("names")
        where (name.Length == 5)
        orderby name
        select name.ToUpper();
```