Microsoft®

tech·ed

North America | 2010

JUNE 7-10, 2010 | NEW ORLEANS, LA

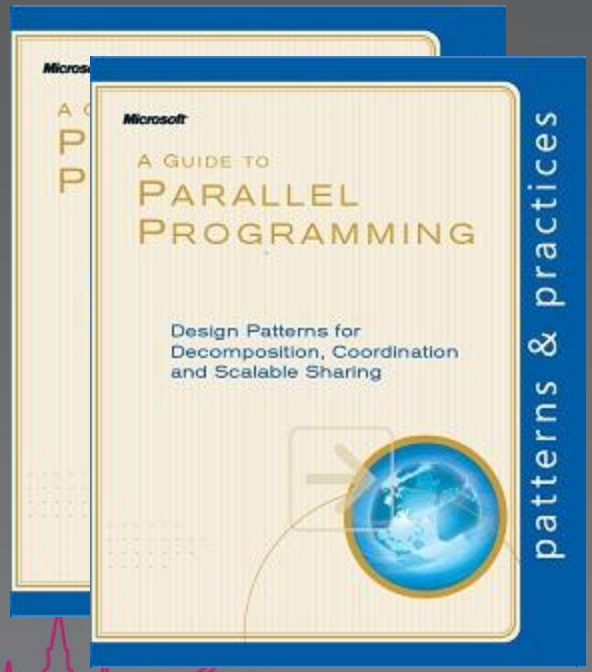
Microsoft®

Patterns of Parallel Programming

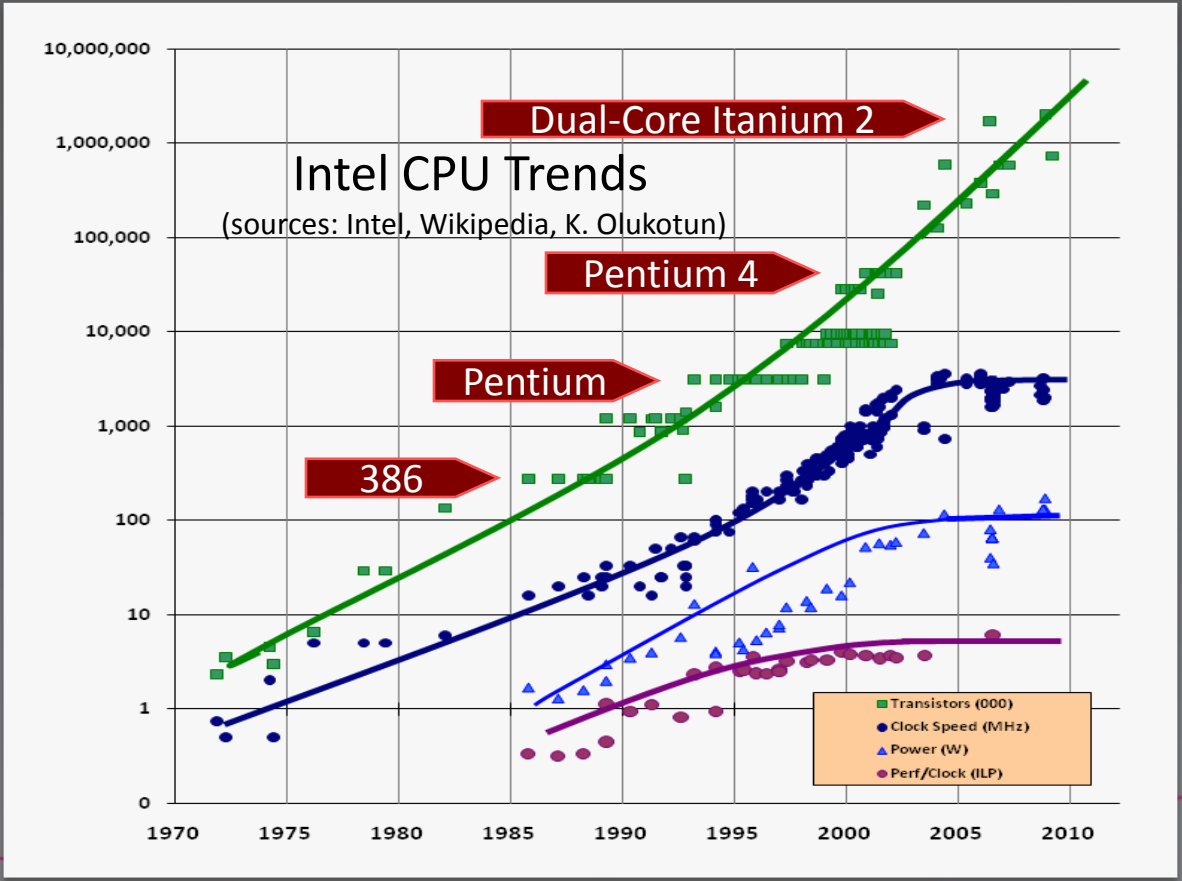
Ade Miller (adem@microsoft.com)
Senior Development Manager
Microsoft patterns & practices

Introduction

- Why you should care
- Where to start
- Pattern walkthrough
- Conclusions (and a quiz)



Why Should You Care?



Then:
Faster clocks

Now:
More cores

End of the Free Lunch

The End of The Free Lunch

- Although driven by hardware changes,
  the parallel revolution is **primarily a software revolution.**
- Parallel hardware is not “more of the same.”
- Software is the gating factor.
- Software requires the most changes to regain the “free lunch.”
- Hardware parallelism is coming,
  more and sooner than most people yet believe.

Where Should I Start?

- “Avoid multithreaded code”
- “Parallel programming is hard”
- “It’s for the experts”

- How do we succeed in this new parallel world?

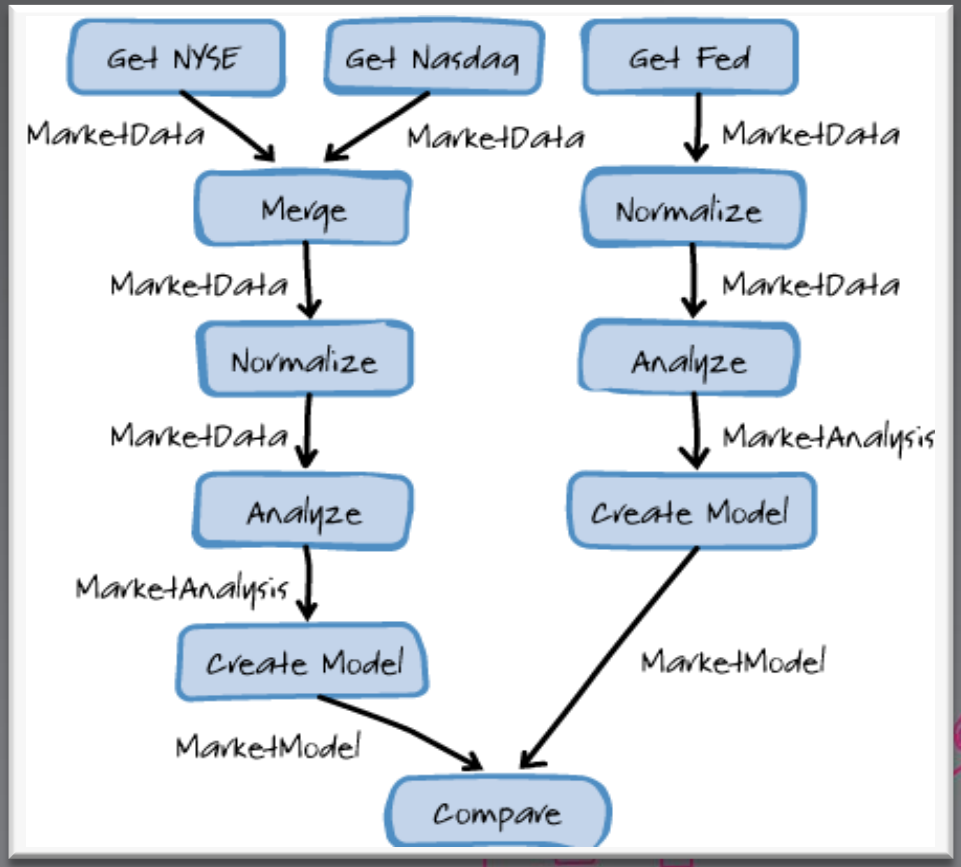
- Let’s look at an application and see…

An Example: Adatum-Dash

- A Financial application for portfolio risk analysis
- Look at large chunks of recent and historical data
- Compare models with market conditions

- Source code (beta) available: http://parallelpatterns.codeplex.com/

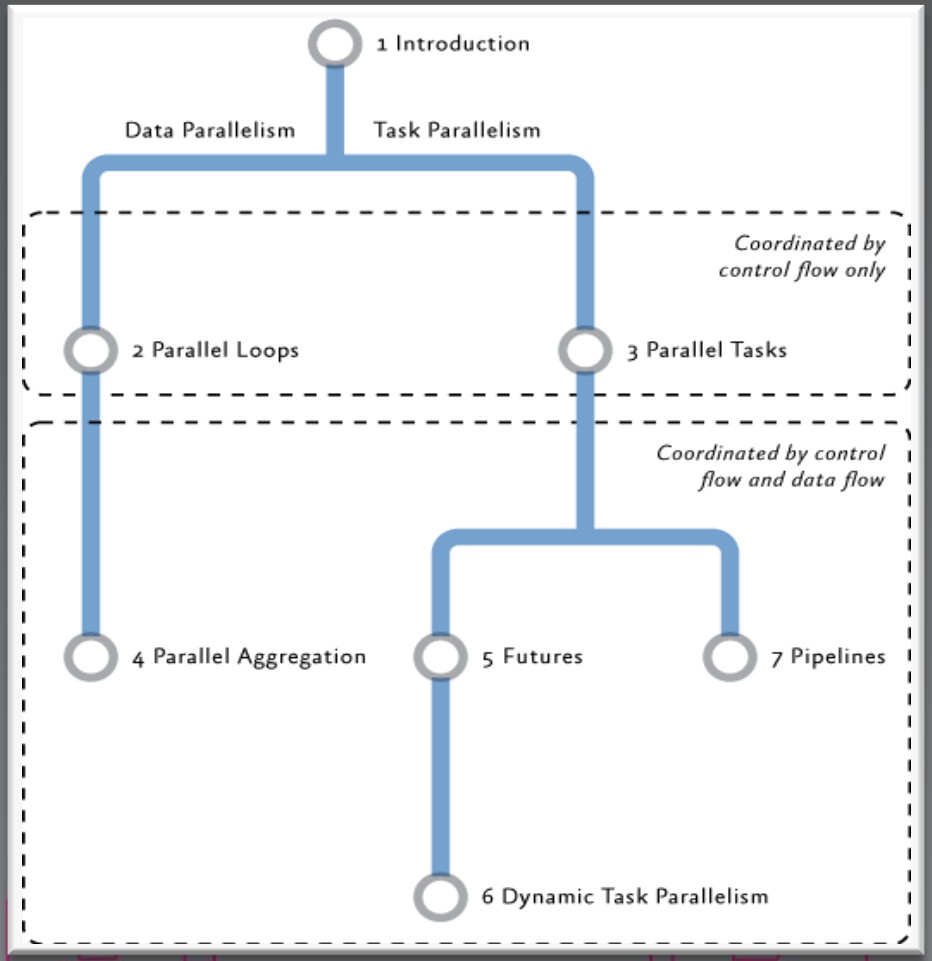
The Adatum Dash Scenario



Adatum Dash

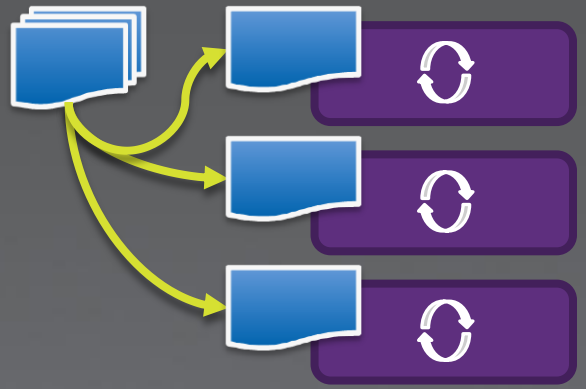
Finding Potential Parallelism

- Tasks vs. Data
- Control Flow
- Control and Data Flow



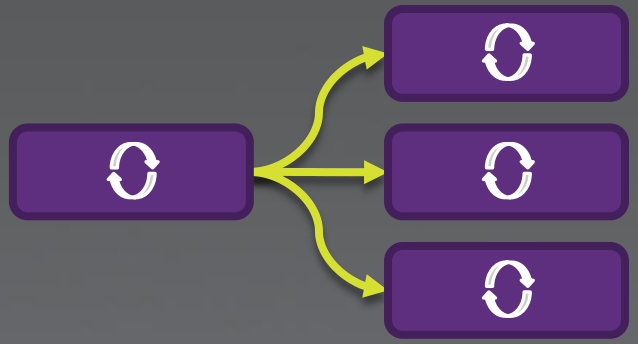
Data Parallelism

- Data “chunk” size?
  - Too big – under utilization
  - Too small – thrashing
- Chunk layout?
  - Cache and cache line size
  - False cache sharing
- Data dependencies?



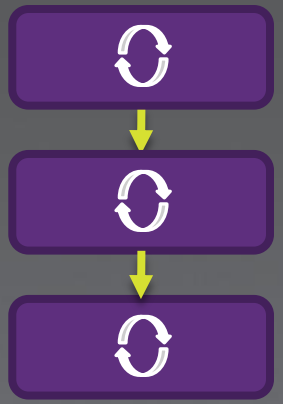
Task Parallelism

- Enough tasks?
  - Too many – thrashing
  - Too few – under utilization
- Work per task?
  - Small workloads
  - Variable workloads
- Dependencies between tasks?
  - Removable
  - Separable
  - Read only or read/write



Control and Data Flow

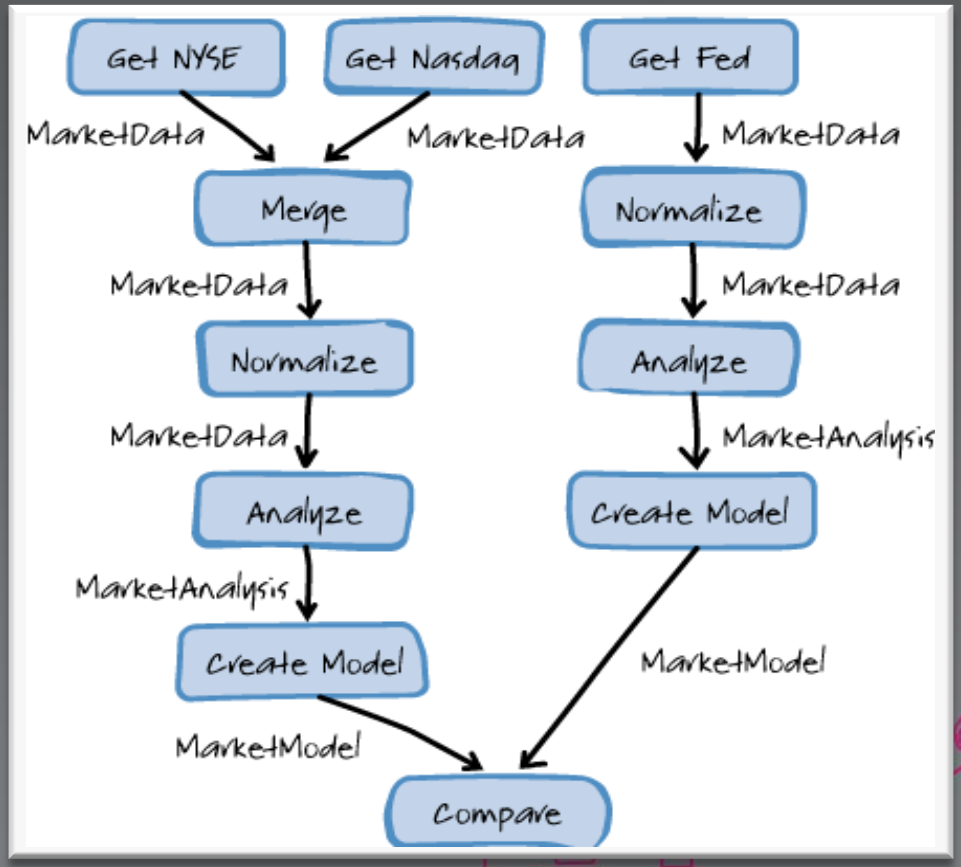
- Task constraints
  - Temporal: A → B
  - Simultaneous: A ↔ B
  - None: A B
- External constraints
  - I/O read or write order
  - Message or list output order
- Linear and irregular orderings
  - Pipeline
  - Futures
  - Dynamic Tasks



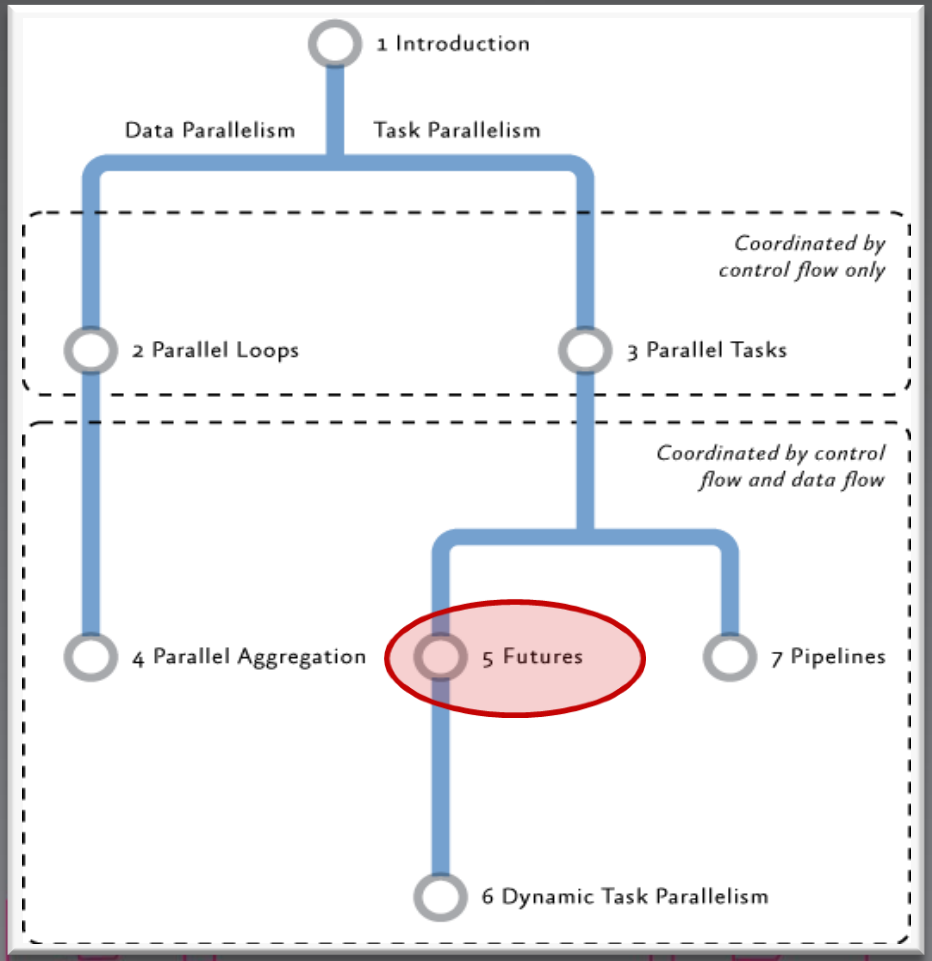
Solution Forces

- Flexibility:
  - Easy to modify for different scenarios
  - Runs on different types of hardware
- Efficiency:
  - Time spent managing the parallelism vs. time gained from utilizing more processors or cores
  - Performance improves as more cores or processors are added – Scaling
- Simplicity:
  - The code can be easily debugged and maintained

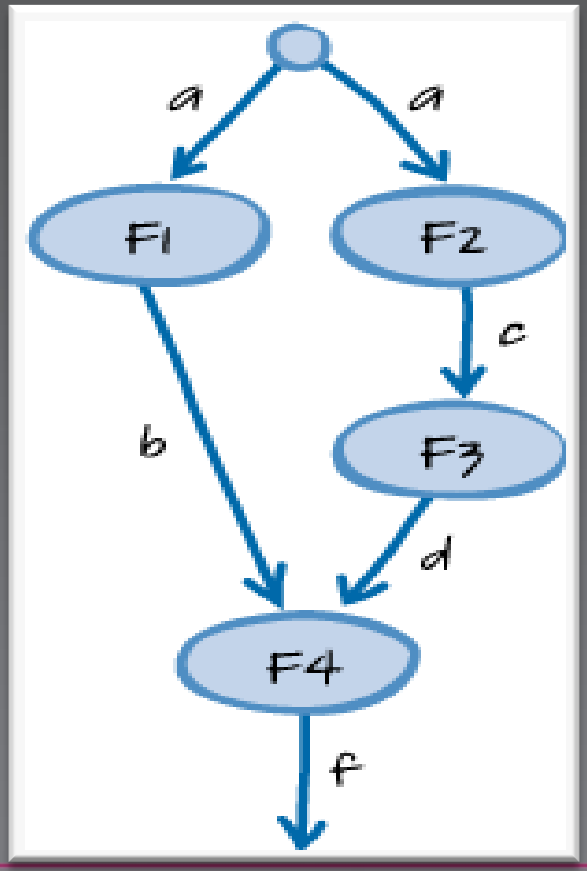
The Adatum Dash Scenario



The Futures Pattern



The Futures Pattern



"Does the ordering of steps in your algorithm depend on data flow constraints?"

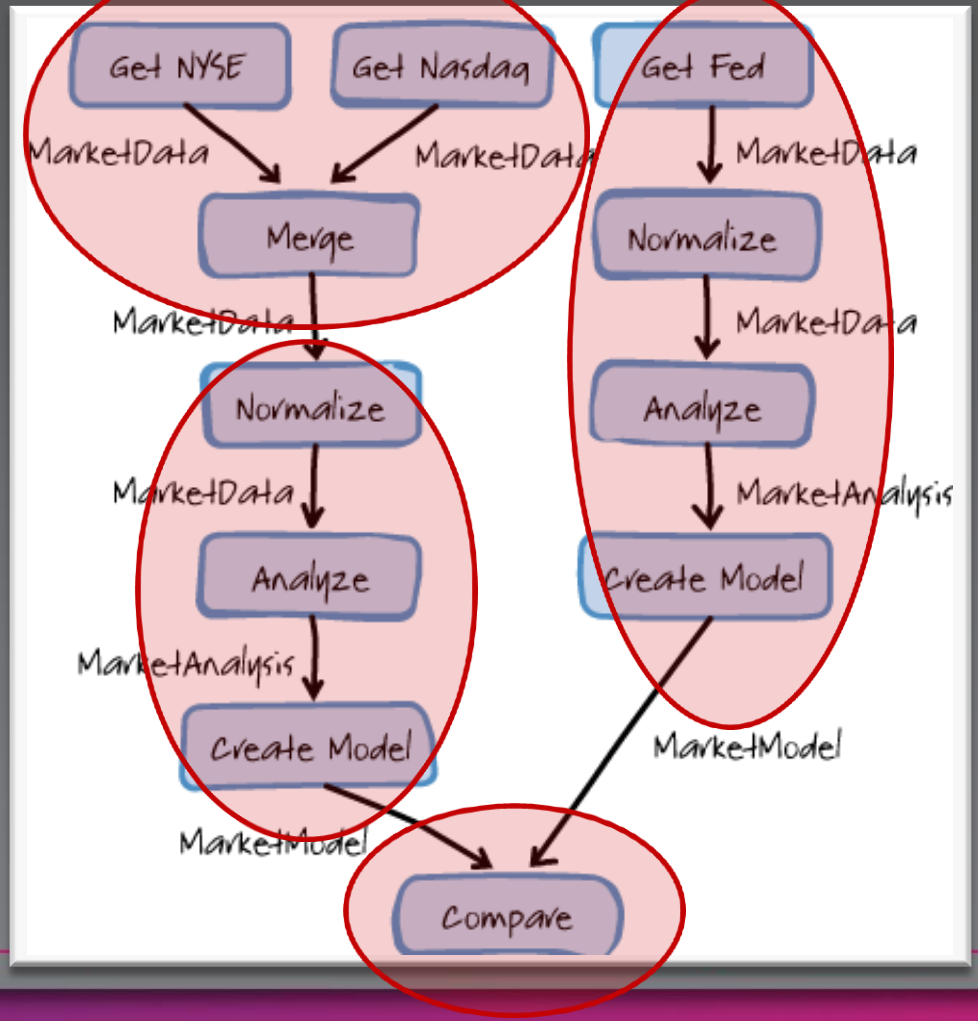
- Directed Acyclic Graph
- Dependencies between tasks
- F4 depends on the result of F1 & F3 etc

- Also called "Task Graph"

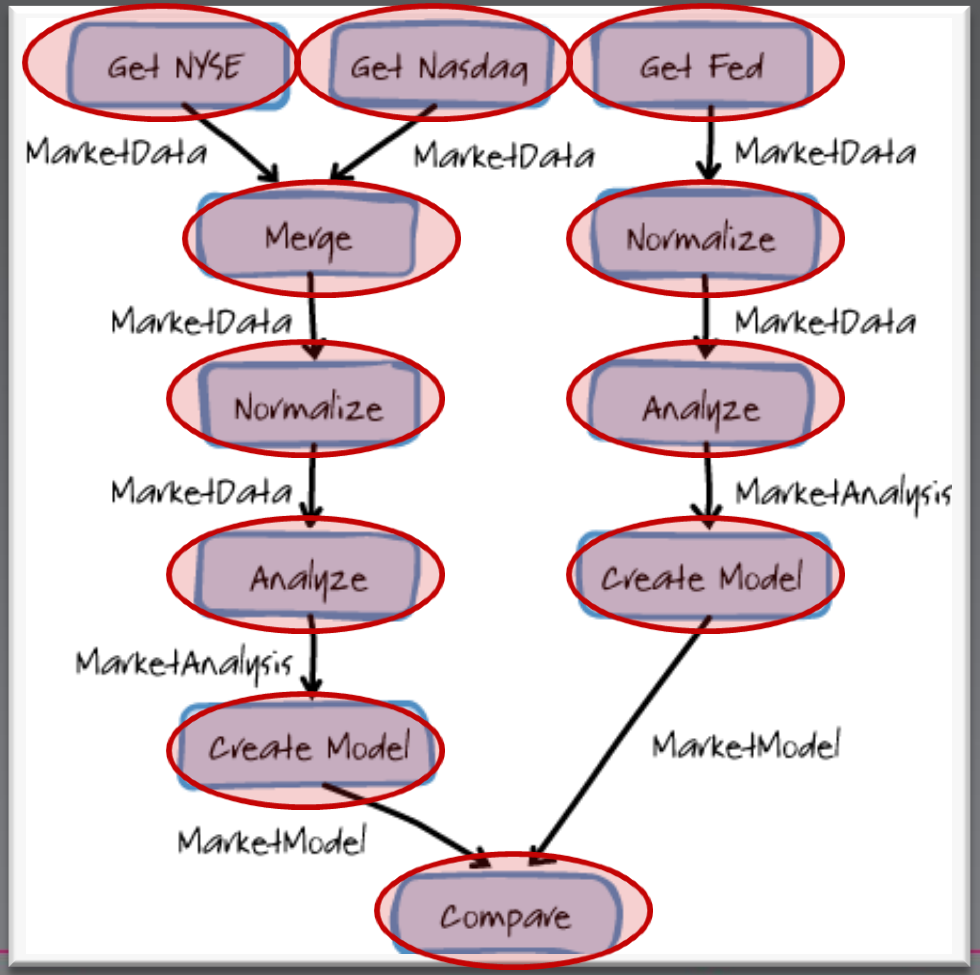
Task Size and Granularity

- Variable size tasks – harder to balance
- Small tasks – more overhead; management and communication
- Large tasks – less potential for utilization
- Hardware specific – more tasks than cores

Course Grained Partition

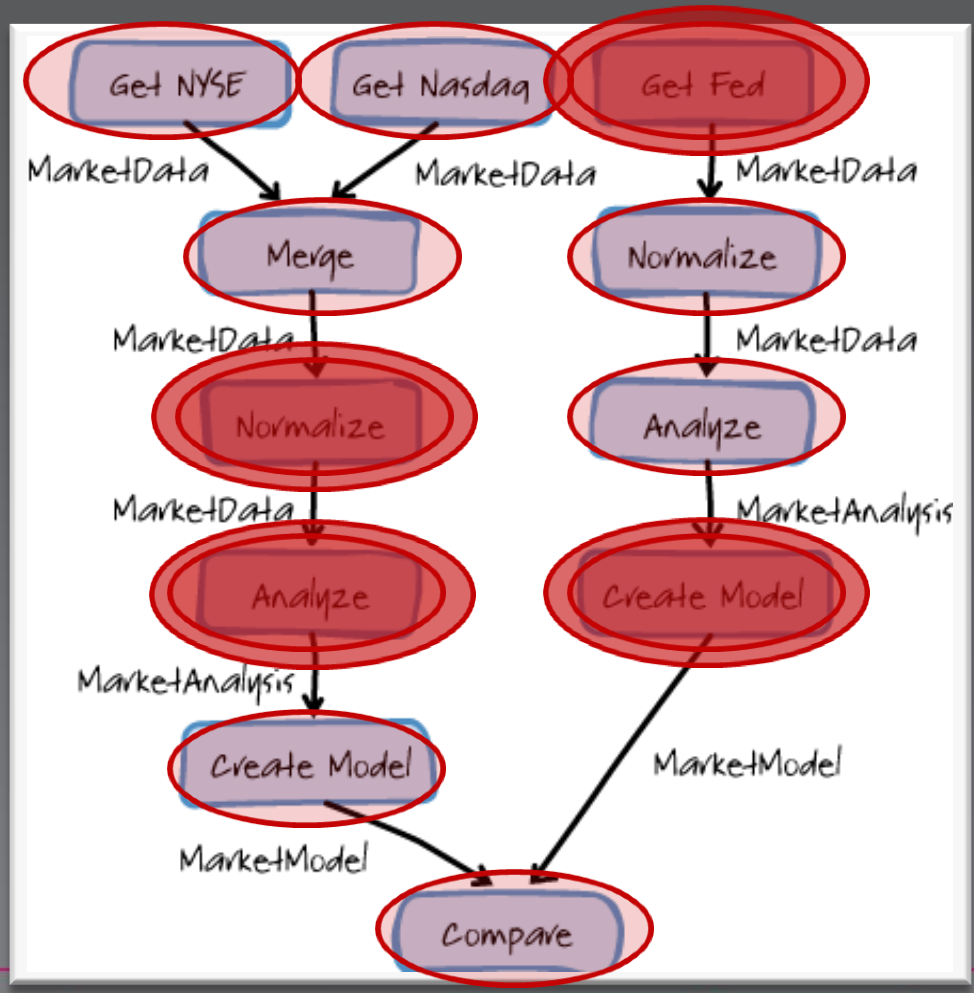


Fine Grained Partition

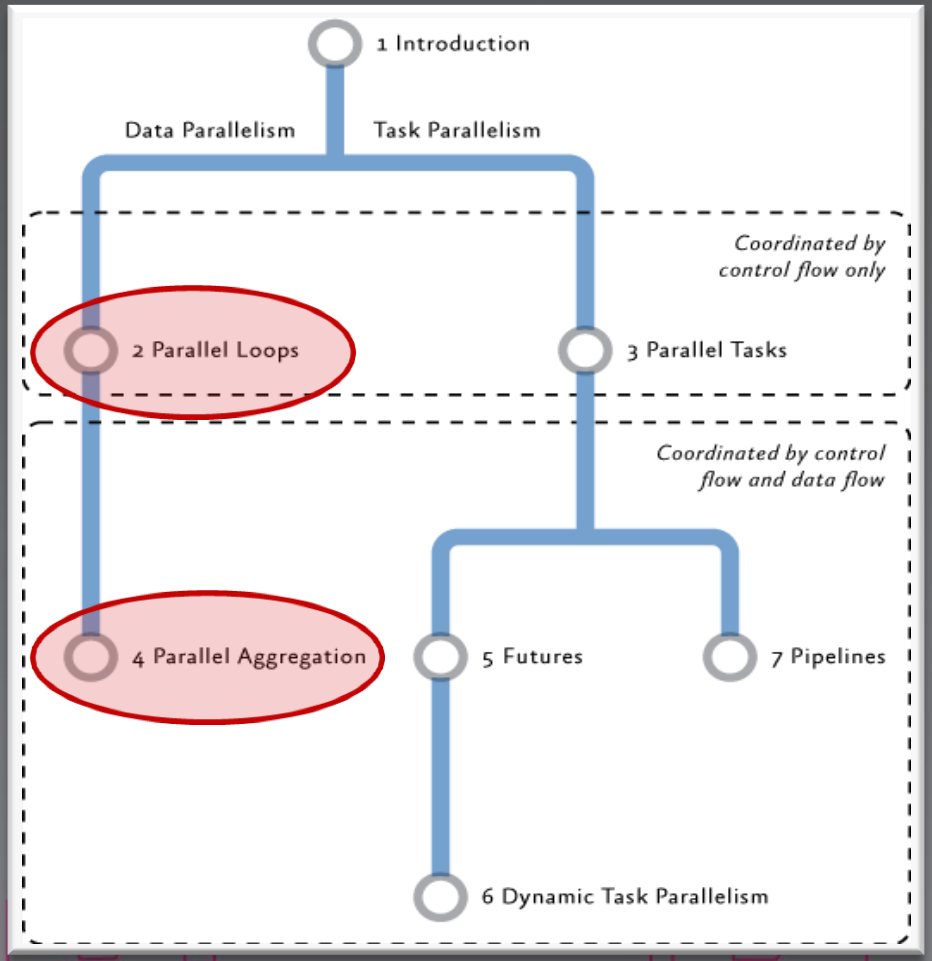


Futures

Finer Grained Partition



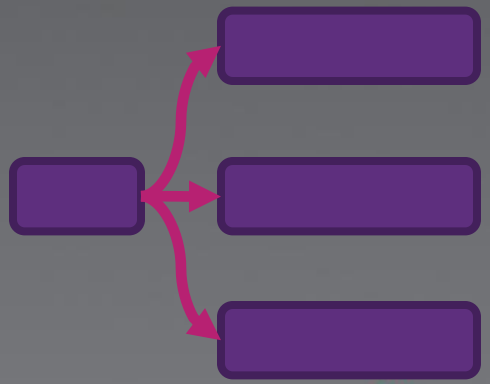
Data Parallelism Patterns



The Parallel Loop Pattern

“Do you have sequential loops where there's no communication among the steps of each iteration?”

- A very common problem!

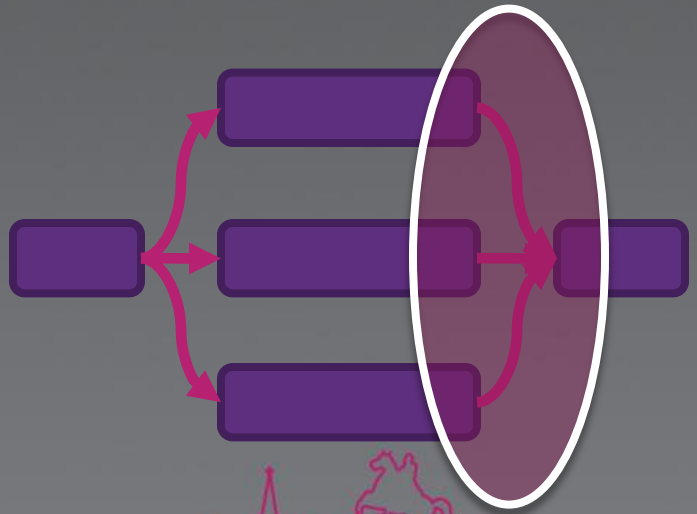


The Parallel Aggregation Pattern

"Do you need to summarize data by applying some kind of combination operator? Do you have loops with steps that are not fully independent?"

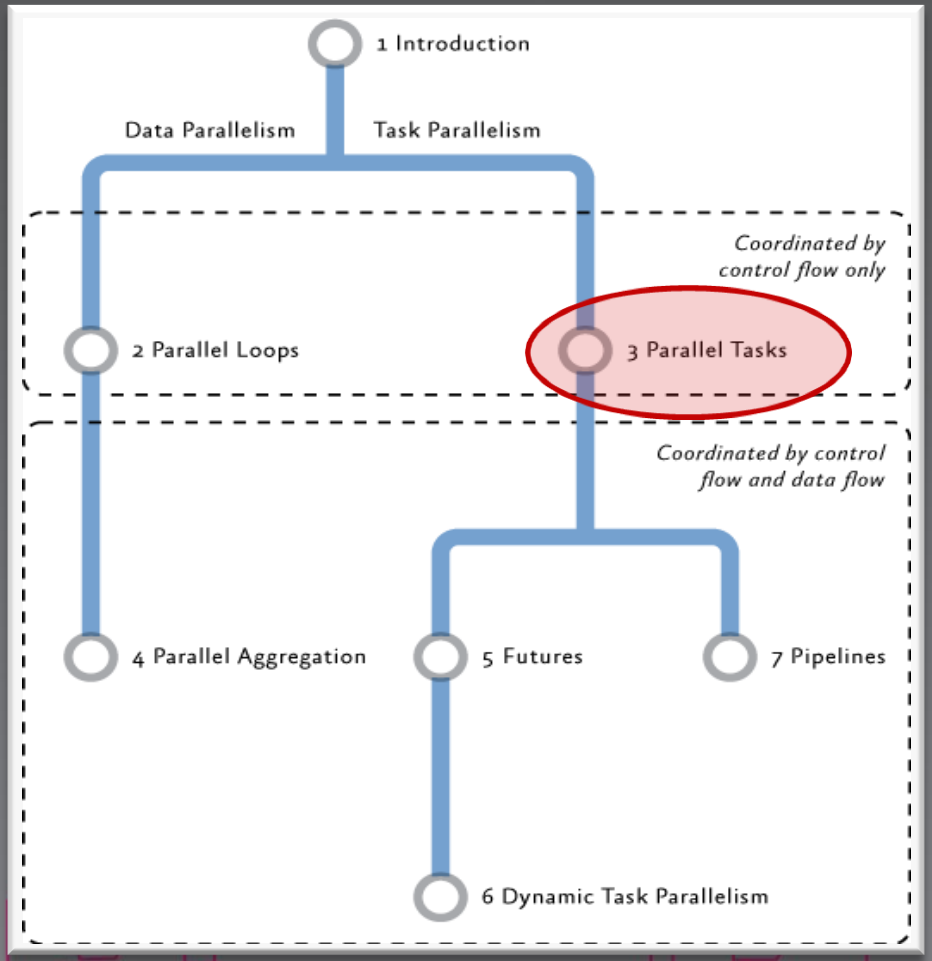
- Calculate sub-problem result per task
- Merge results later
- Reduces need for locking

- "Reduction" or "map/reduce"



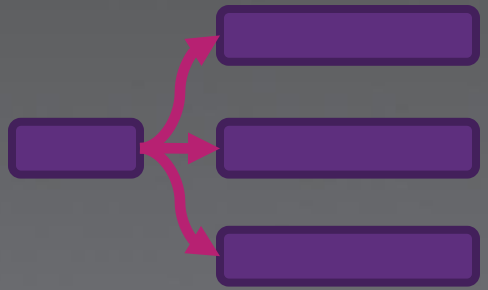
Parallel Loops and Aggregation

The Parallel Tasks Pattern



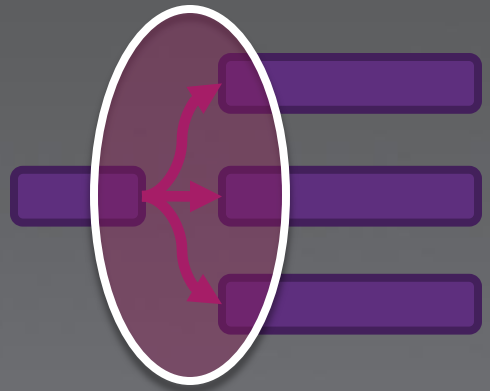
The Parallel Tasks Pattern

“Do you have specific units of works with well-defined control dependencies?”



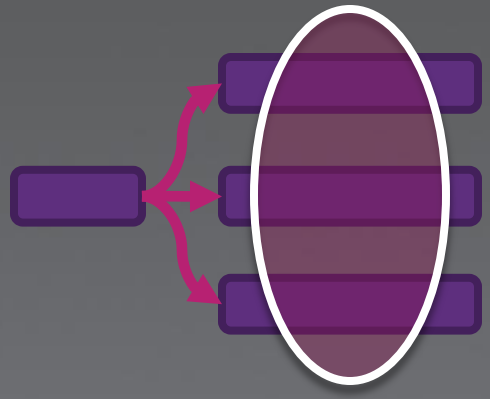
Partitioning

- How do we divide up the workload?
  - Fixed workloads
  - Variable workloads
- Workload size
  - Too large – hard to balance
  - Too small – communication may dominate



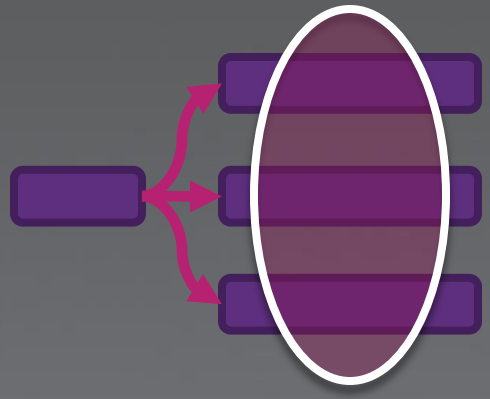
Workload Balancing

- Static allocation:
  - By blocks
  - By index (interleaved)
  - Guided
- Dynamic work allocation
  - known and unknown task sizes
  - Task queues
  - Work stealing
- The TPL does a lot of this work for you



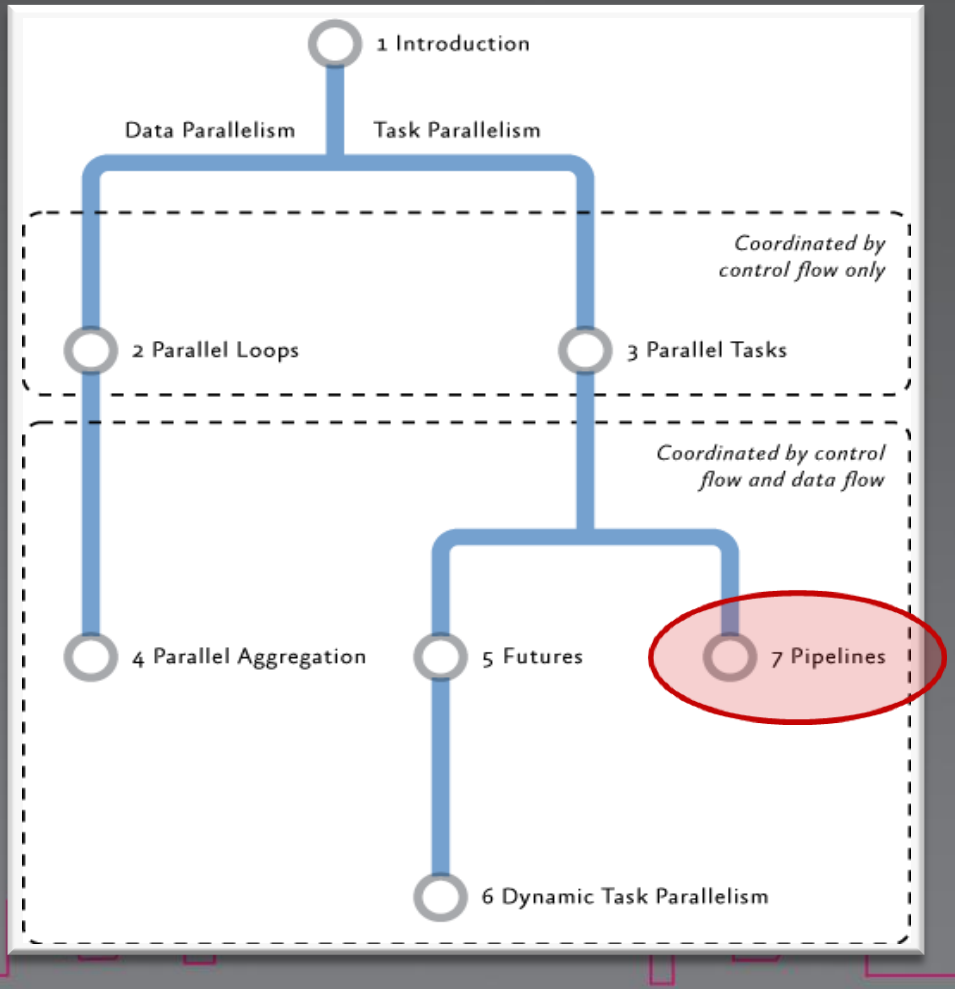
Sharing State and Synchronization

- Don’t share!
- Read only data
- Data isolation
- Synchronization



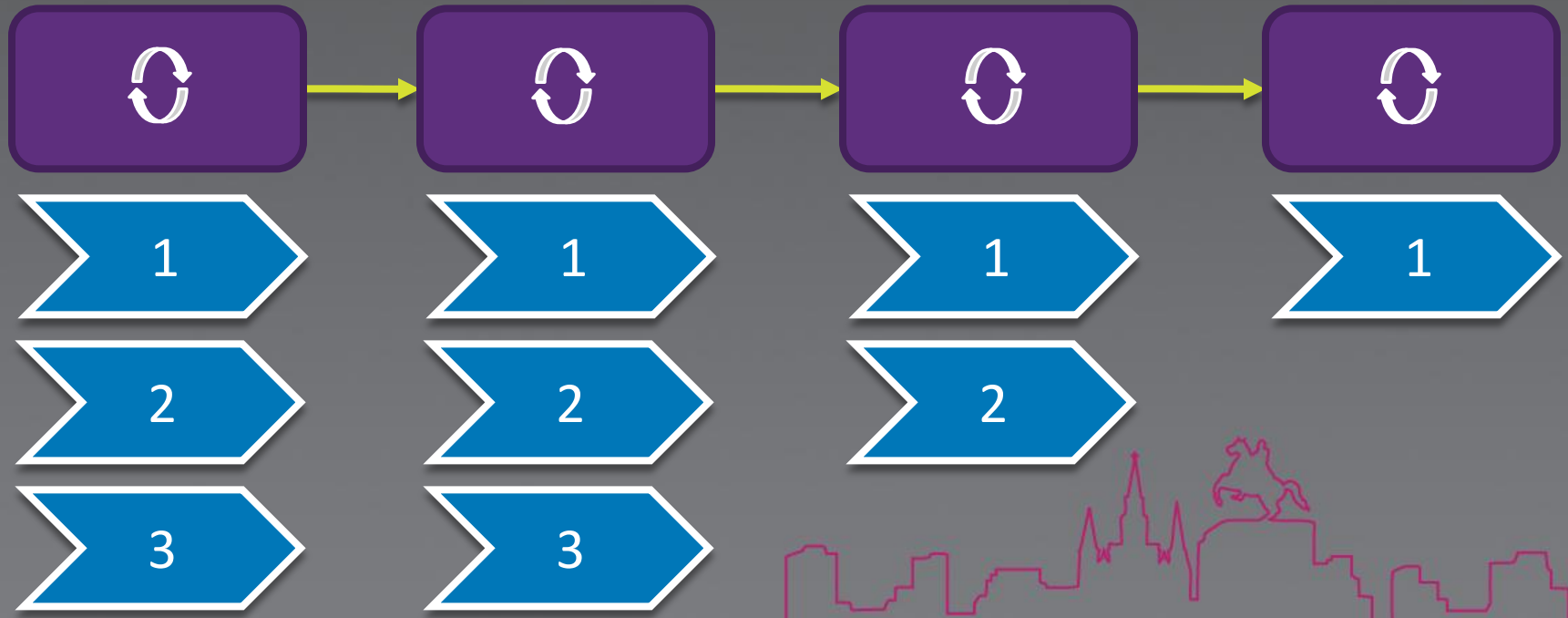
Parallel Tasks

The Pipeline Pattern



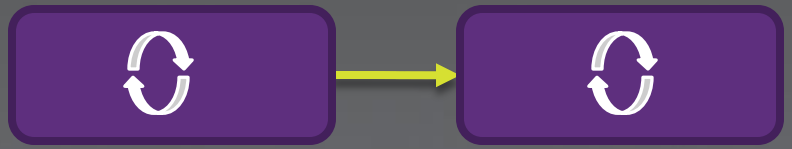
The Pipeline Pattern

“Does your application perform a sequence of operations repetitively? Does the input data have streaming characteristics?”



The Producer/Consumer Pattern

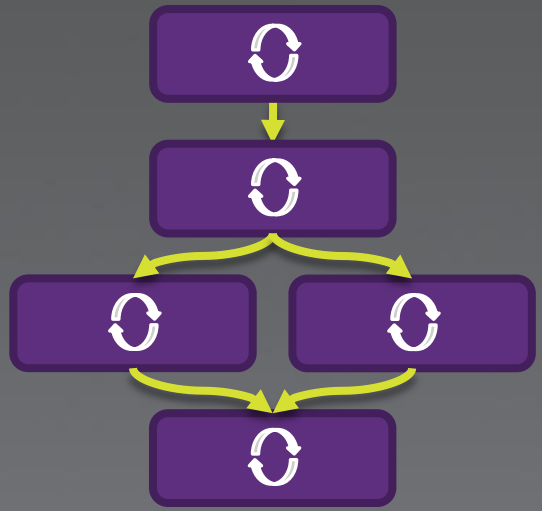
- Organize by Ordering



- Producers… produce!
  - Block when buffer full
- Consumers… consume!
  - Block when buffer empty

Workload Balancing

- Pipeline length
  - Long – High throughput
  - Short – Low latency
- Stage workloads
  - Equal – linear pipeline
  - Unequal – nonlinear pipeline

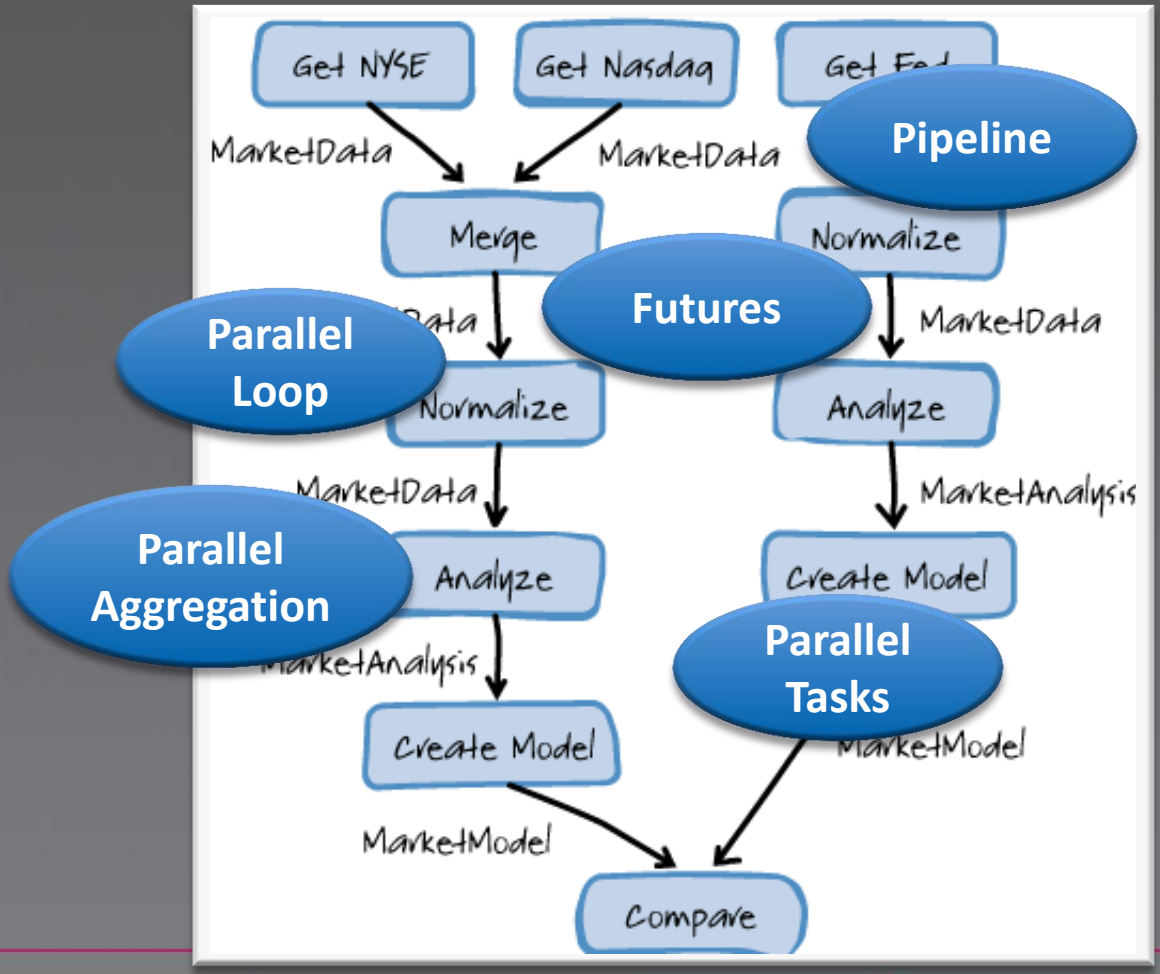


Passing Data Down the Pipe

- Shared queue(s)
  - Large queue items – under utilization
  - Small queue items – locking overhead

Pipeline

Opportunities for Parallelism

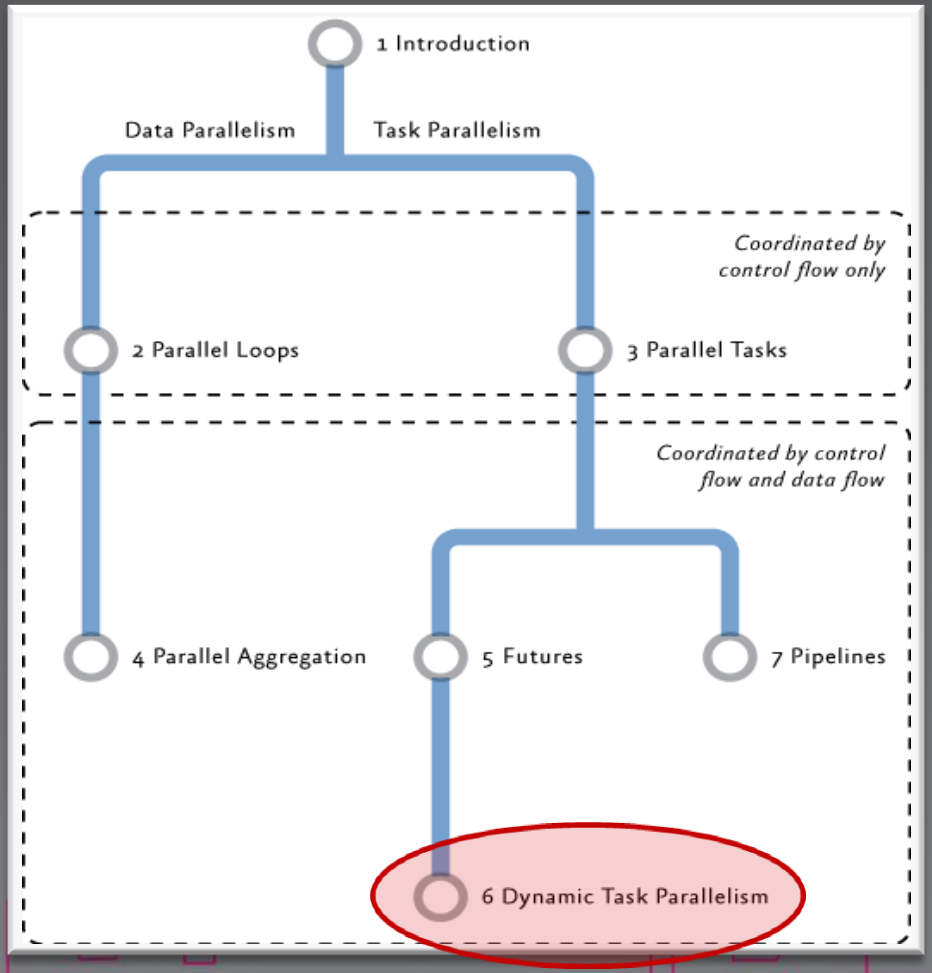


But Wait… There's More!

What About Recursive Problems?

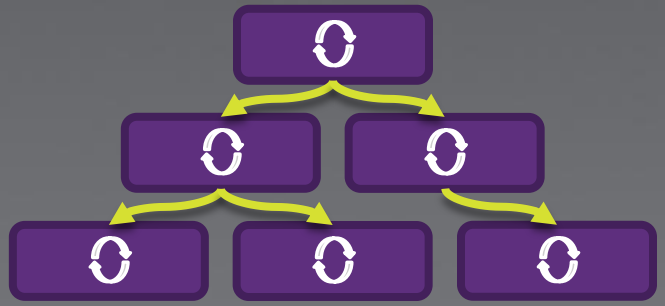
- Many problems can be tackled using recursion:
  - Task based: Divide and Conquer
  - Data based: Recursive Data

The Dynamic Task Parallelism Pattern



The Dynamic Task Parallelism Pattern

“Does your algorithm divide the problem domain dynamically during the run? Do you operate on recursive data structures such as graphs?”



Workload Balancing

- Deep trees – thrashing
  - Limit the tree depth
- Shallow trees – under utilization
- Unbalanced Trees – under utilization

Dynamic Task Parallelism Example

```
static void Walk<T>(Tree<T> root, Action<T> action)
{
    if (root == null) return;
    var t1 = Task.Factory.StartNew(() =>
          action(root.Data));
    var t2 = Task.Factory.StartNew(() =>
          Walk(root.Left, action));
    var t3 = Task.Factory.StartNew(() =>
          Walk(root.Right, action));
    Task.WaitAll(t1, t2, t3);
}
```

Conclusions

Success =

- Frameworks and runtimes
  - Task Parallel Library for .NET
  - Parallel Patterns Library for C++
- Tools
  - Visual Studio 2010

- **Guidance!**

Conclusions

QUIZ…

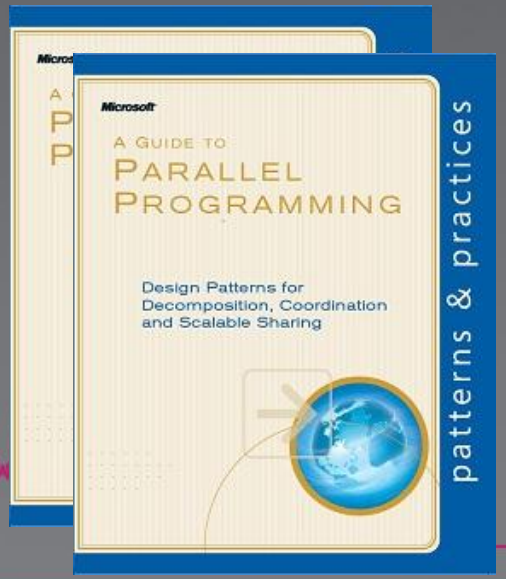
Our Book

A Guide to Parallel Programming: *Design Patterns for Decomposition, Coordination and Scalable Sharing*

Goal: Help developers make the most of the new parallel features in Visual Studio 2010:

Due for release late summer 2010.

http://parallelpatterns.codeplex.com/

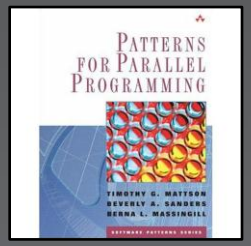


Your Help

- Take the book sample and read it!
- Come and give me feedback…
  - Architecture Area of the TLC (next to the Dev Info desk) - 10:30 to 12:30 on Wednesday
  - The p&p Booth
- **You don't need to be a "parallel expert" to provide feedback!**

- The first 40 people to read it and give me feedback @ TechEd will get a free copy!

Other Resources



Books

- Patterns for Parallel Programming – Mattson, Sanders & Massingill
- Design Patterns – Gamma, Helm, Johnson & Vlissides
- Head First Design Patterns – Freeman & Freeman
- Patterns of Enterprise Application Architecture – Fowler

Research

- A Pattern Language for Parallel Programming ver2.0
- ParaPLOP - Workshop on Parallel Programming Patterns

- My Blog: http://ademiller.com/tech/ (Decks etc.)

Resources

Microsoft tech·ed Online

Sessions On-Demand & Community

www.microsoft.com/teched

Microsoft | Learning

Microsoft Certification & Training Resources

www.microsoft.com/learning

Microsoft TechNet

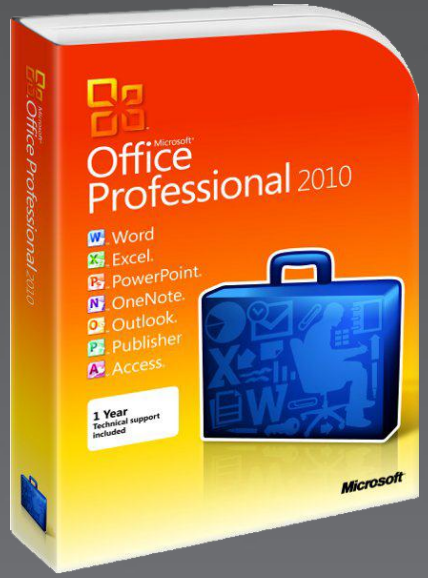
Resources for IT Professionals

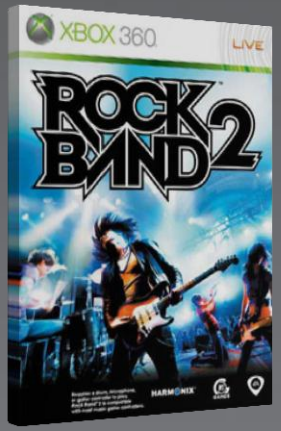
http://microsoft.com/technet

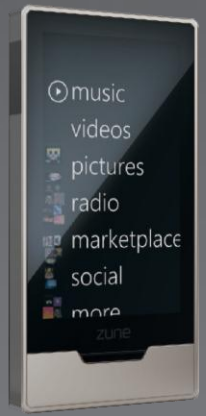
msdn

Resources for Developers

http://microsoft.com/msdn





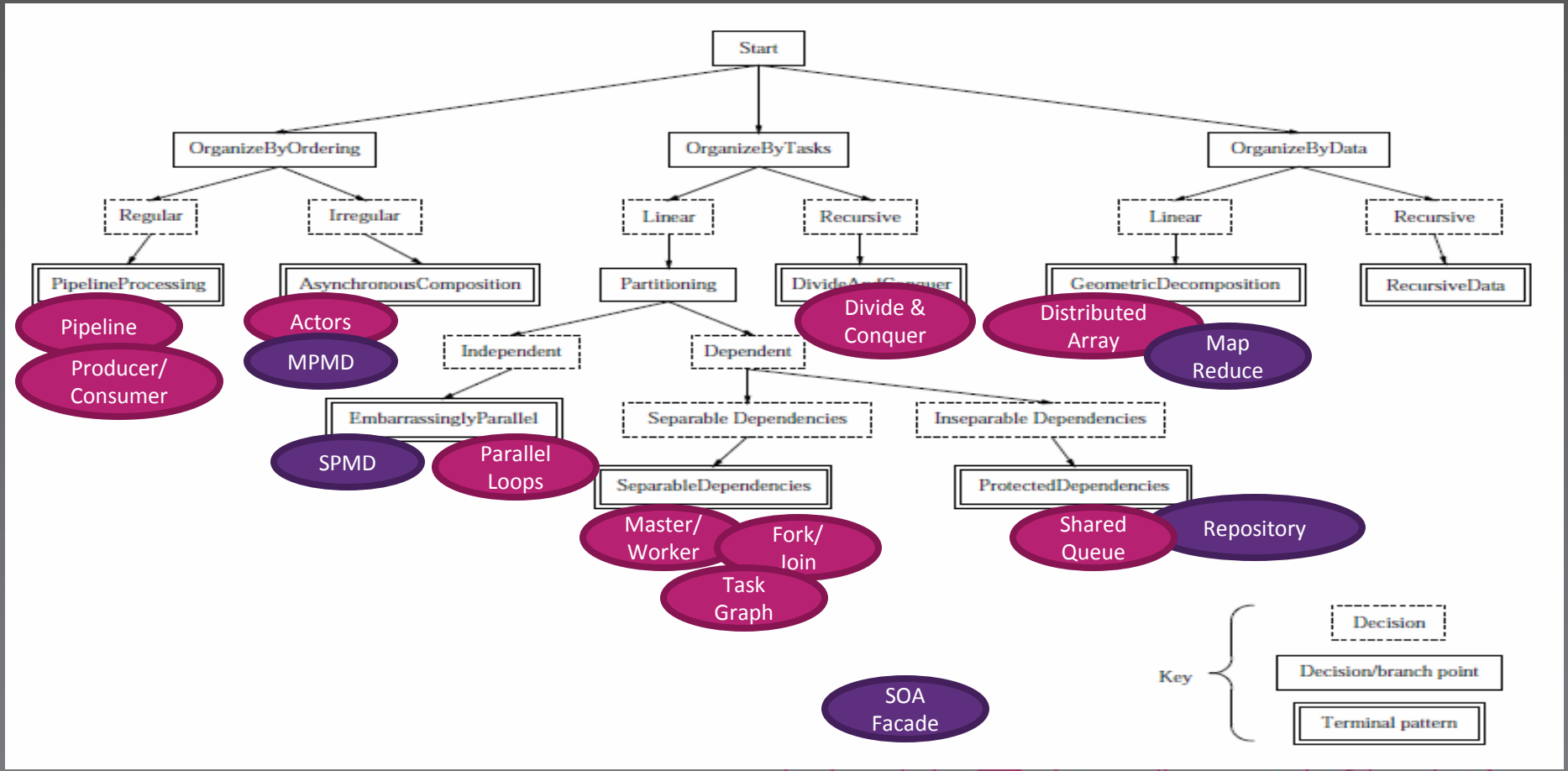


Complete an evaluation on CommNet and enter to win!

Sign up for Tech·Ed 2011 and save $500
starting June 8 – June 31st
http://northamerica.msteched.com/registration

You can also register at the
North America 2011 kiosk located at registration
Join us in Atlanta next year

Microsoft®



Source: More Patterns for Parallel Application Programs, Berna L. Massingill, Timothy G. Mattson and Beverly A. Sanders

Supporting Patterns

- “Gang of Four” Patterns
  - Façade
  - Decorator
  - Repository
- Shared Data Patterns
  - Shared Queue

The Façade Pattern

- Hide parallelism
- Optimize call granularity

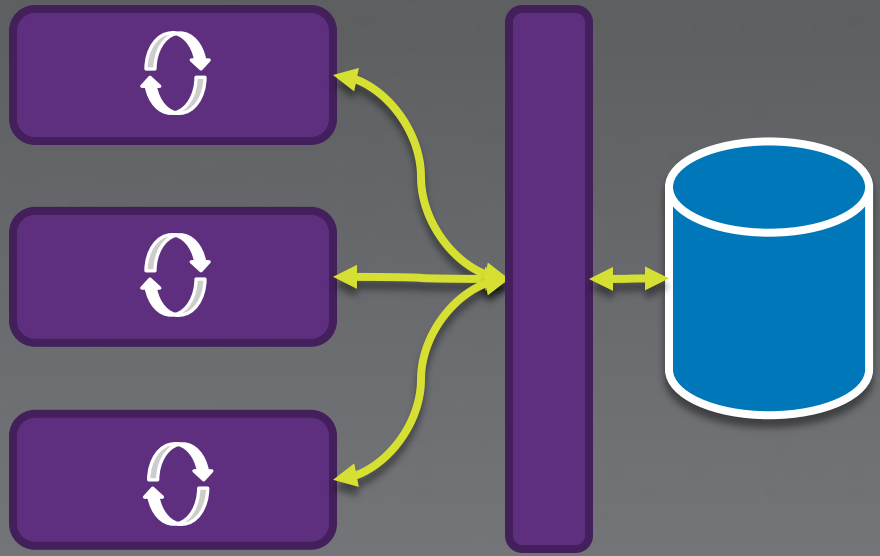
Source: Design Patterns – Gamma, Helm, Johnson & Vlissides
Patterns of Enterprise Application Architecture - Fowler

The Decorator Pattern

- Encapsulate parallelism
  - Calling code is parallel agnostic
- Add parallelism to an existing (base) class

Source: Design Patterns – Gamma, Helm, Johnson & Vlissides

The Repository Pattern



- Shared hash or queue
- Database
- Distributed cache

Source: Patterns of Enterprise Application Architecture - Fowler

The Shared Queue Pattern

Task Queue

- A decorator to Queue
  - Hides the locking which protects the underlying queue
- Facilitates Producer/Consumer pattern
  - Producers call:
    `theQueue.Enqueue()`
  - Consumers call:
    `theQueue.Dequeue()`

Source: Patterns for Parallel Programming – Mattson, Sanders & Massingill

Shared Queue Example

```
var results =
       new ConcurrentQueue<Result>();

Parallel.For(0, 1000, (i) =>
{
    Result result =
        ComputeResult(i);
    results.Enqueue(result);
});
```

Parallel With PLINQ

```
var accountRatings =
       accounts.AsParallel()
           .Select(item =>
               CreditRating(item))
         .ToList();
```

Loop Parallel Examples

```
Parallel.For (0, acc.Length, i =>
{
    acc[i].UpdateCreditRating();
});

#pragma omp parallel for
for (int i = 0; i < len; i++)
{
    acc[i].UpdateCreditRating();
  }
```

Parallel Tasks Example

```
Parallel.Invoke(
   () => ComputeMean(),
   () => ComputeMedian()
);

   parallel_invoke(
   [&] { ComputeMean(); },
   [&] { ComputeMedian(); },
);
```

Pipeline Examples

```
var input = new BlockingCollection<string>();

var readLines = Task.Factory.StartNew(() =>
{
    try {
        foreach(var line in
                    File.ReadAllLines(@"input.txt"))
                input.Add(line);
    }
    finally { input.CompleteAdding(); }
});

var writeLines = Task.Factory.StartNew(() =>
{
```

Pipeline Examples

```
Get-ChildItem C:\ |
   Where-Object {$_.Length -gt 2KB} |
      Sort-Object Length
```