

Microsoft®  
tech·ed  
中国 | 2010

1-3 December 2010 | Beijing, China



会议代码 : ARC-300-3

Microsoft®  
tech·ed  
中国 | 2010  
1-3 December 2010 | Beijing, China

# .NET 4 中的并行程序 模式

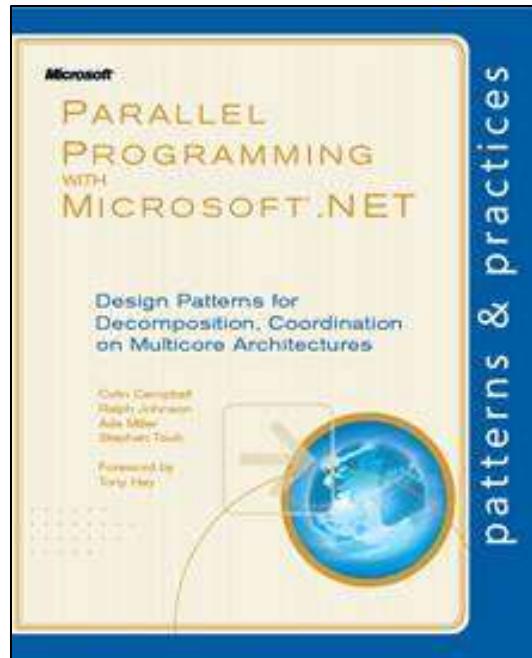


Ade Miller  
Principle Development Lead  
Microsoft patterns & practices

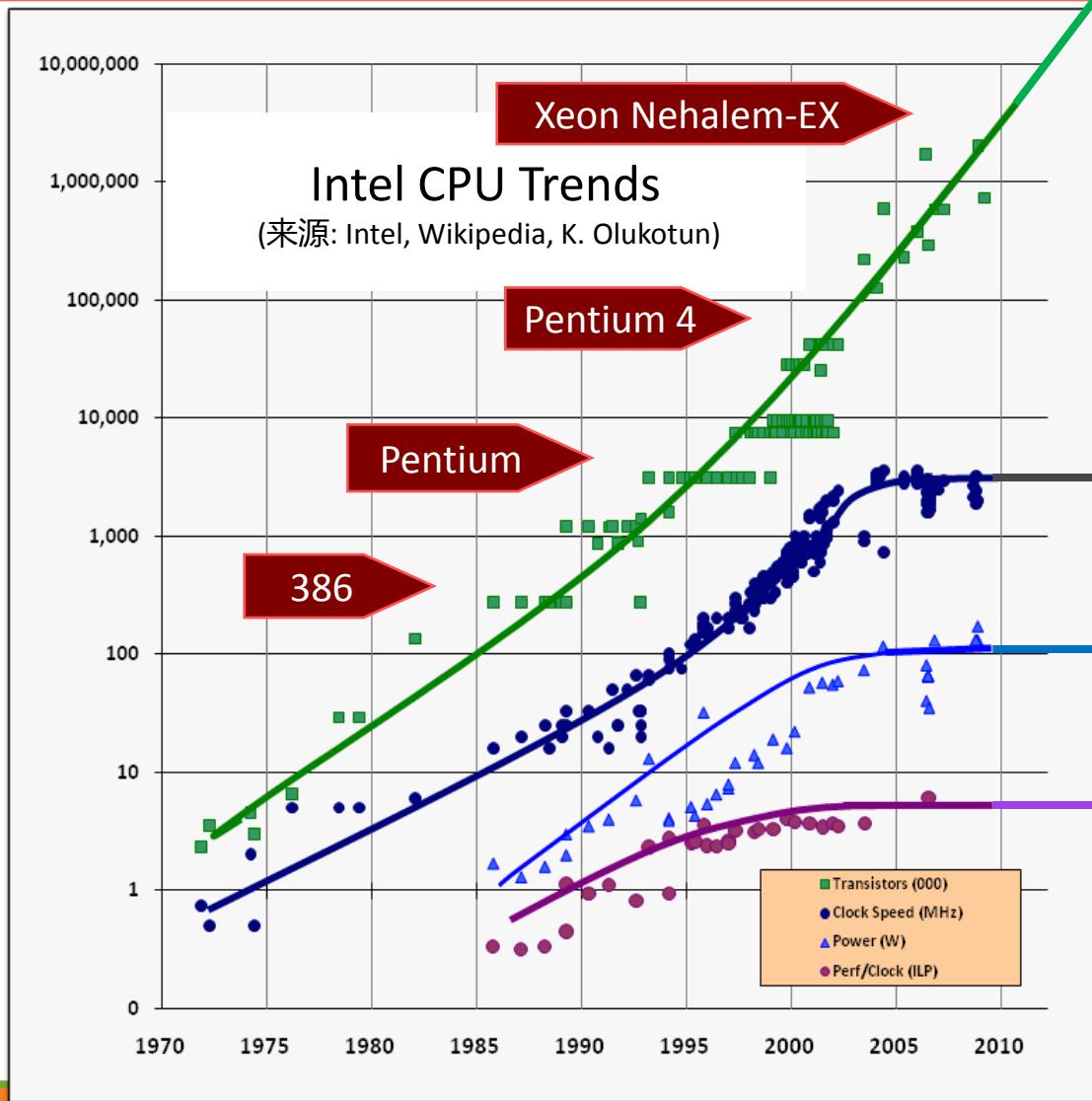


# 介绍

- 为何需要关注？
- 从哪里开始？
- 模式介绍
- 结论(以及简单的测试)



# 为何关注?



摩尔定律

时钟频率

能耗

指令级并行



# 免费午餐的终结

- 尽管来自于硬件变化的推动，并行革命主要是软件的革命
- 并行的硬件很多是不同的
- 软件是关键的因素
- 软件需要重大的变化以获得“免费的午餐”
- 硬件的并行化已经到来，更多的人很快就会相信这一点



# 从哪里开始?

- “避免多线程的代码”
- “并行程序编程是困难的”
- “这是为专家们准备的”
- 我们如何能够在新的并行世界里取得成功?
- 让我们看一个应用程序并了解更多...

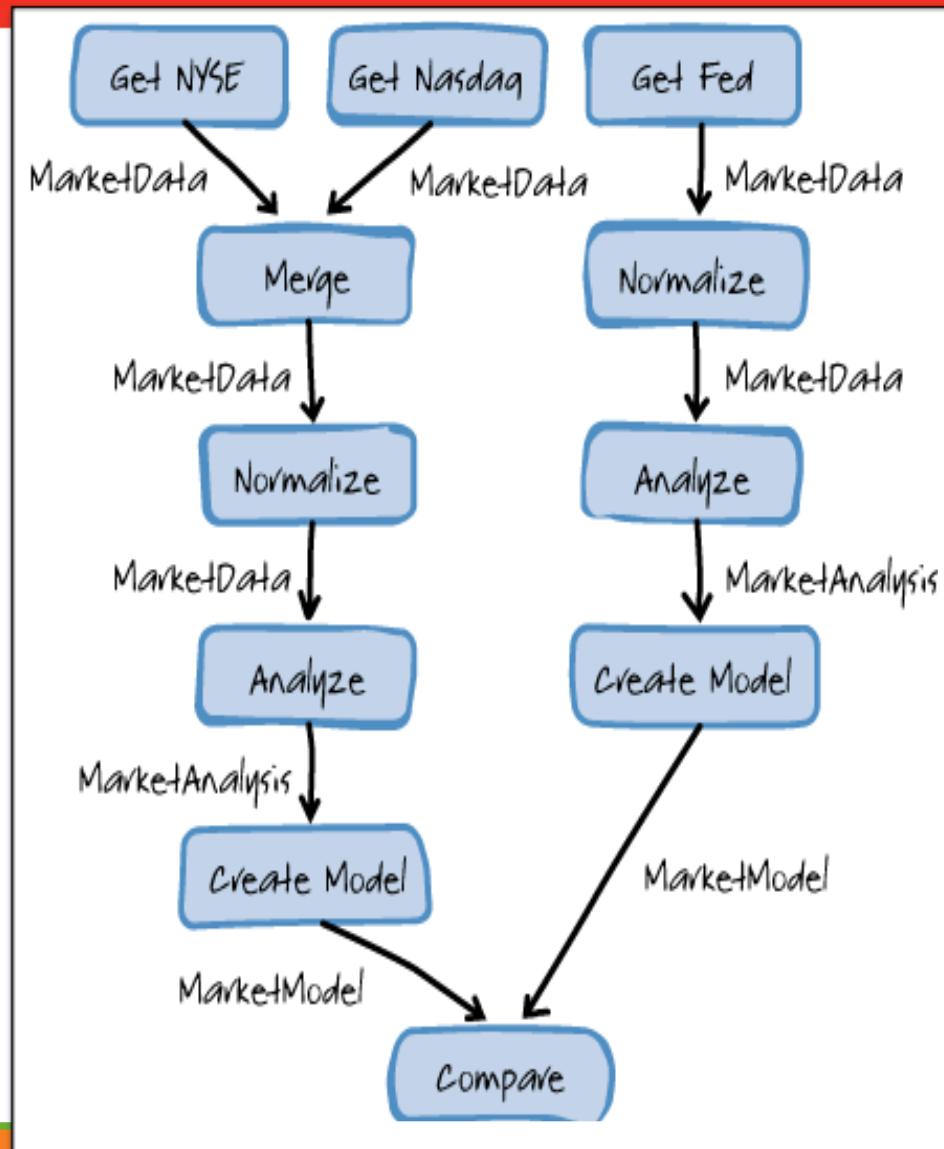


# 一个例子: Adatum-Dash

- 一个金融投资组合的风险分析程序
  - 需要查看大量的近期数据以及历史数据
  - 市场条件的比较模型
- 
- 源代码位于:  
<http://parallelpatterns.codeplex.com/>



# Adatum Dash 的场景



1-3 December 2010 | Beijing, China

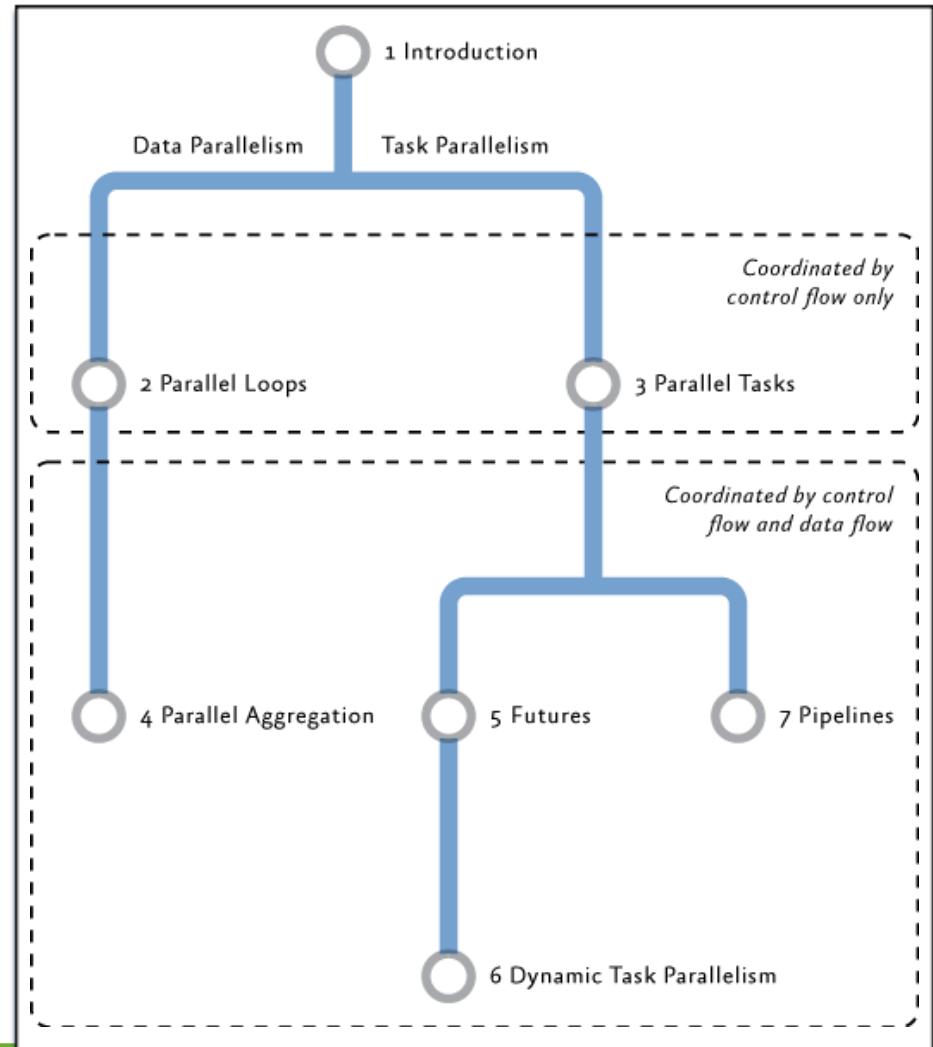
# 演示

## Adatum Dash



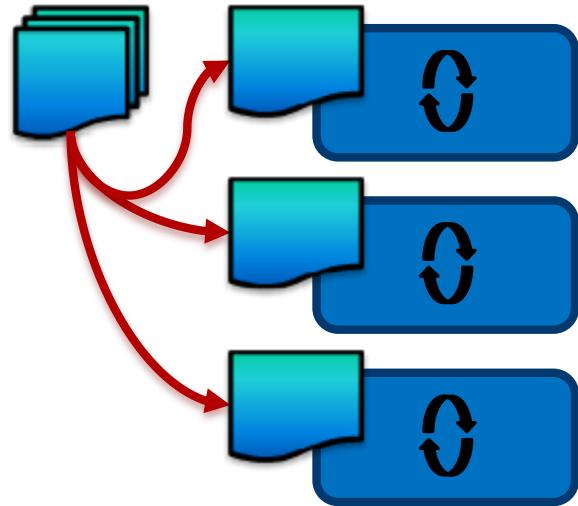
# 寻找潜在的并行

- 任务 vs. 数据
- 控制流
- 控制与数据流



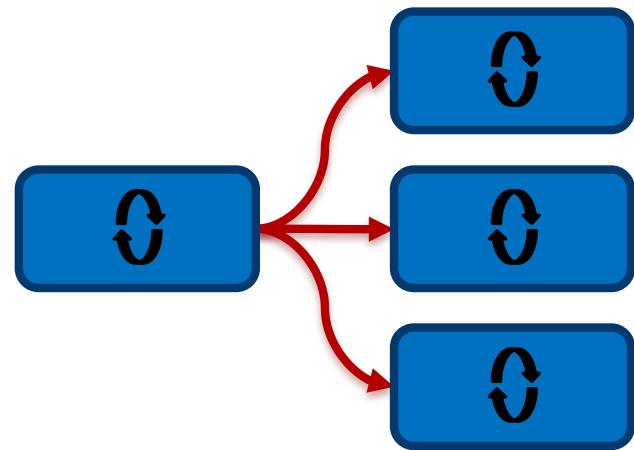
# 数据并行

- 数据“块”的大小?
  - 太大 – 不能充分利用
  - 太小 – 抖动
- 块布局?
  - 缓存以及缓存行的大小
  - 假的缓存共享
- 数据依赖?



# 任务并行

- **任务充分?**
    - 太多 - 抖动
    - 太少 - 利用不充分
  - **每个任务的工作?**
    - 小的工作负荷
    - 变化的工作负荷
  - **任务间的依赖?**
    - 可移除的
    - 可分离的
    - 只读或者读/写



# 控制与数据流

- **任务约束**

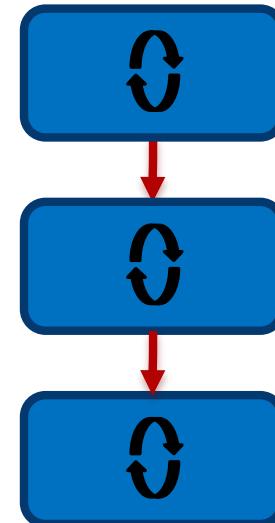
- 临时的:  $A \rightarrow B$
- 同时的:  $A \leftrightarrow B$
- 无约束:  $A \parallel B$

- **外部约束**

- I/O 读或写的顺序
- 消息或队列的输出顺序

- **线性的与无规则的顺序**

- 管道 (Pipeline)
- 预期的 (Futures)
- 动态任务 (Dynamic Tasks)

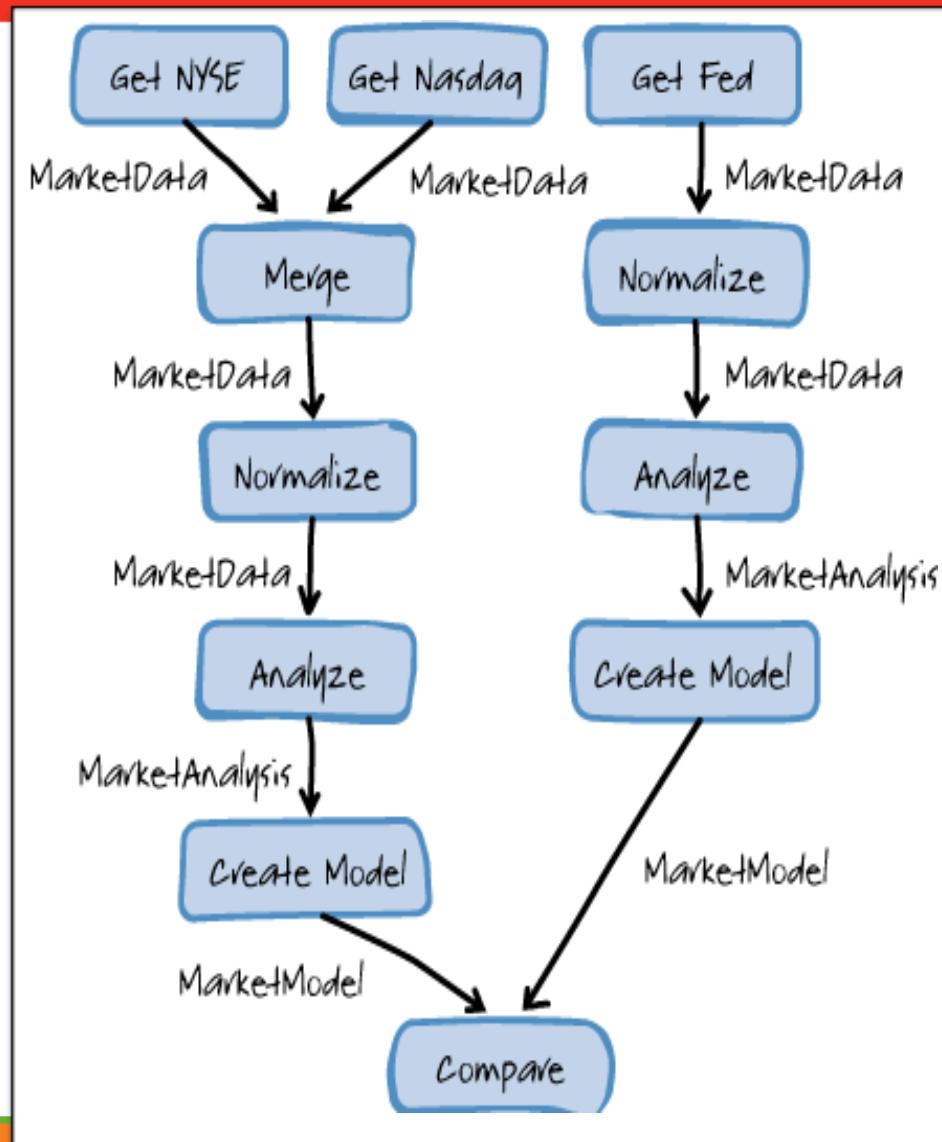


# 解决方案的重点

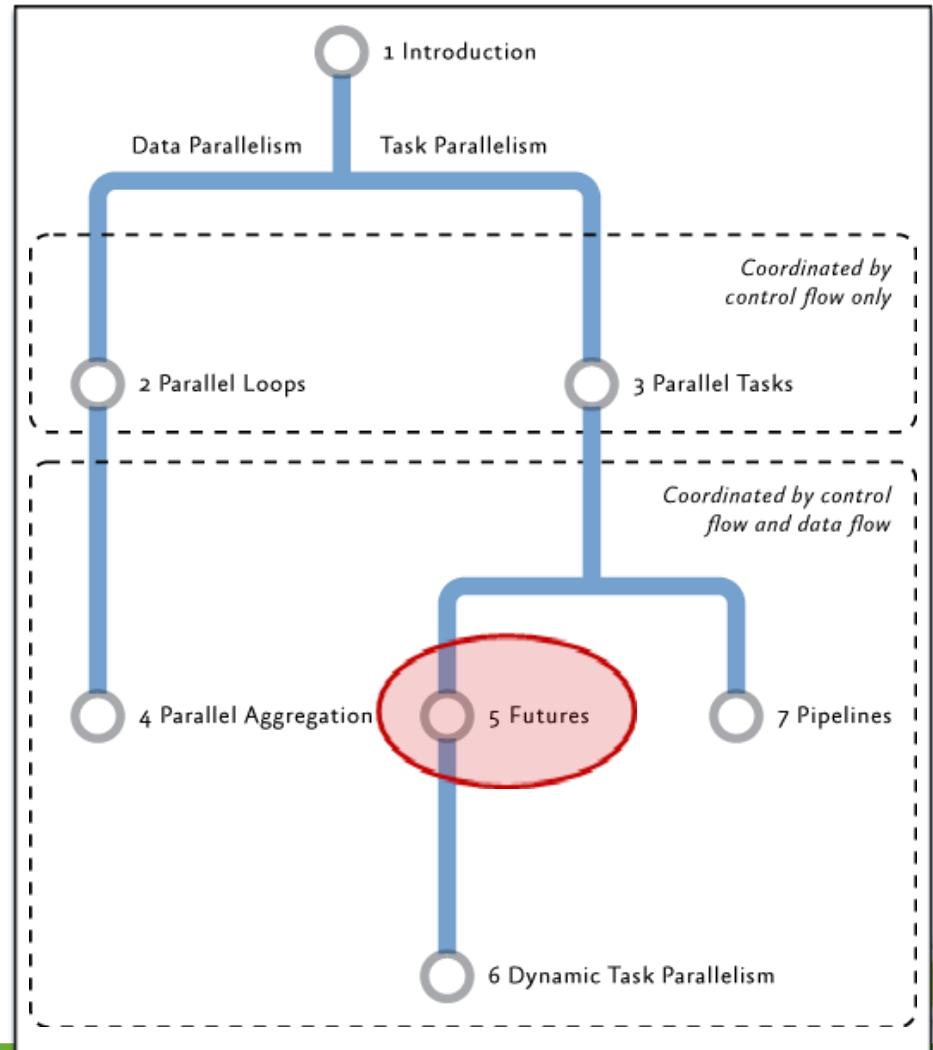
- **灵活性:**
  - 易于修改以适应不同的场景
  - 在不同的硬件上运行
- **效率:**
  - 并行时间开销管理 vs. 利用多处理器或者多核得到更多的时间
  - 随着增加更多的处理器或者核使得性能得到提升 – 扩展
- **简洁:**
  - 代码易于调试与维护



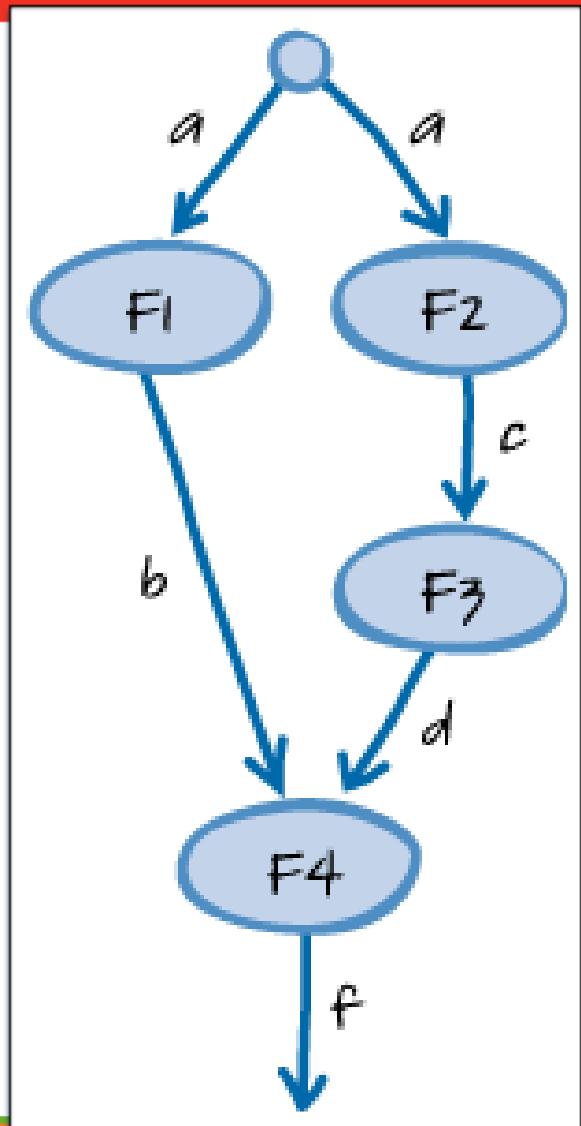
# Adatum Dash 的场景



# 主动对象(Futures) 模式



# 主动对象(Futures) 模式



“并行算法中的步骤顺序取决于  
数据流约束吗 ??”

- 有向无环图
- 任务间的依赖
- F4 依赖与 F1 & F3 的结果等等
- 也被称做 “任务图”

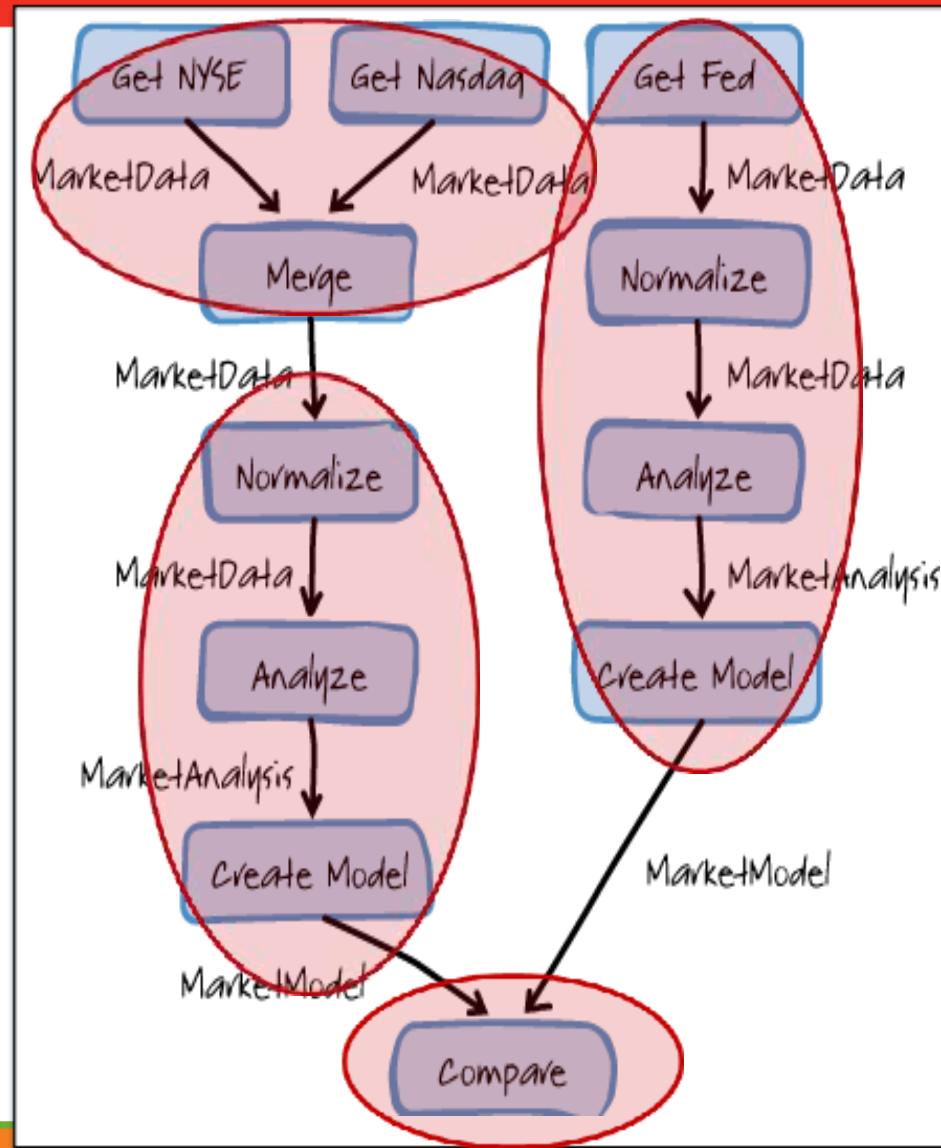


# 任务尺寸与粒度

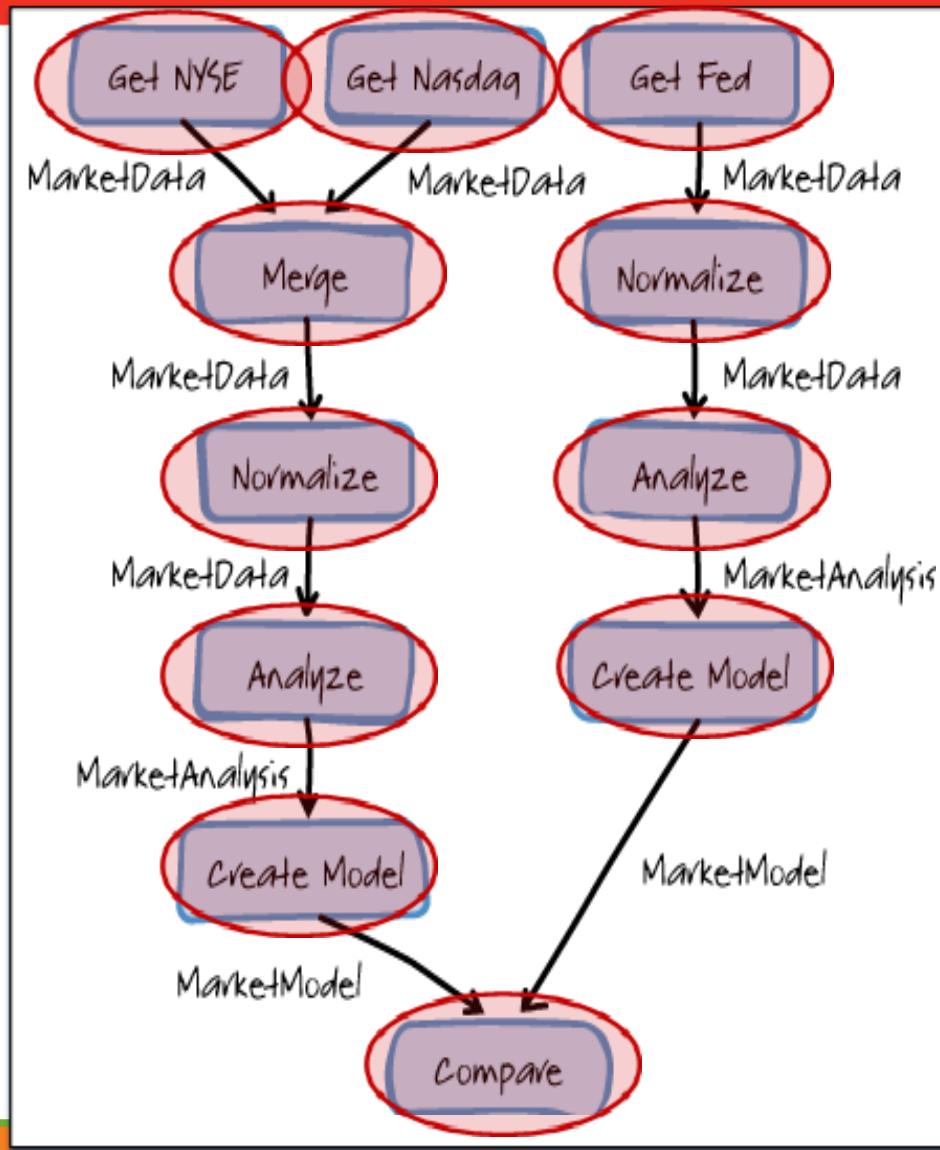
- 可变尺寸的任务 – 难于平衡
- 小任务 – 更多的开销; 管理与通讯
- 大任务 – 缺少提升利用率的潜力
- 特定的硬件 – 任务数多于核的数量



# 过程粒度分区



# 细粒度分区



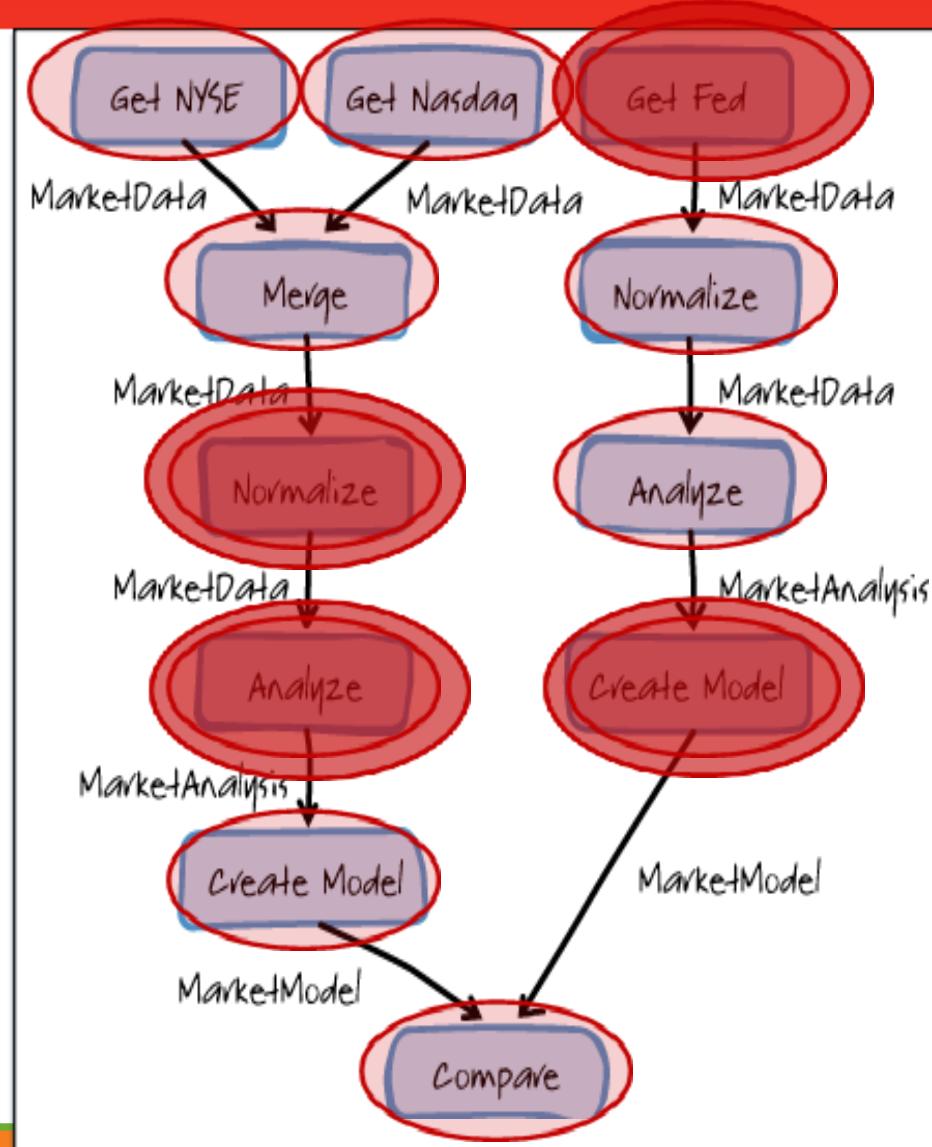
1-3 December 2010 | Beijing, China

# 演示

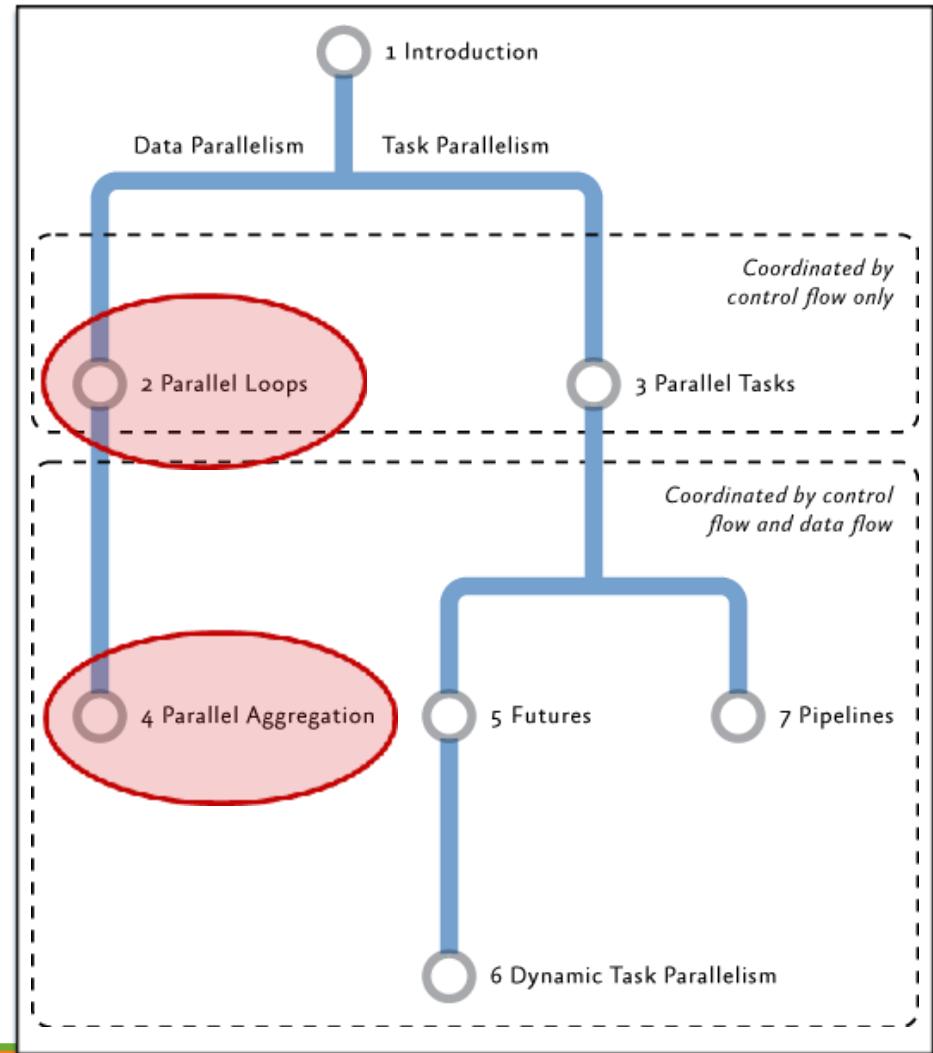
## 主动对象(Futures)



# 细密度分区

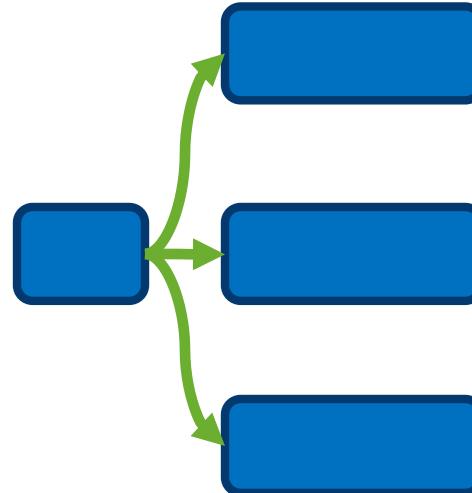


# 数据并行模式



# 并行循环模式

“你是否有这样的顺序循环，在每次迭代之间没有通信的步骤？”



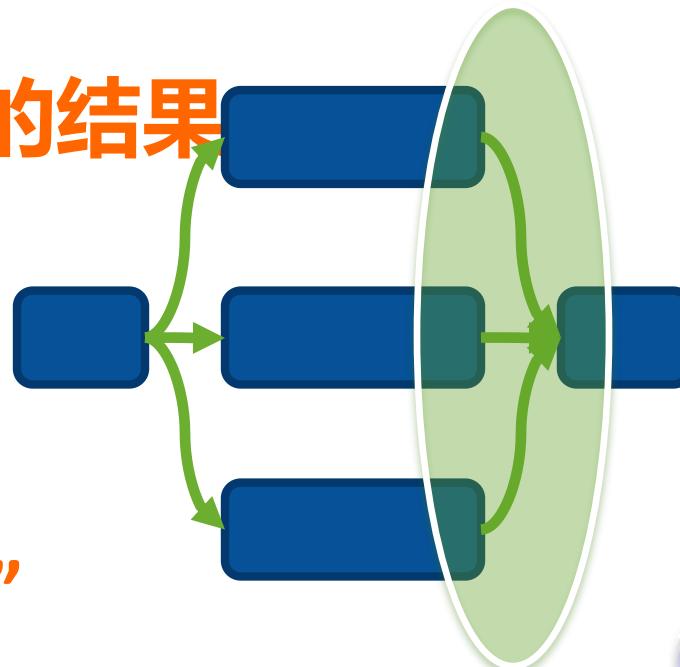
- 一个非常普遍的问题!



# 并行聚合模式

“你需要通过运用某种运算符组合来汇总数据？  
你的有步骤的循环不完全独立？”

- 计算每个任务中子问题的结果
- 合并处理后的结果
- 减少所需要的锁
- “减少”或者“映射/规约”



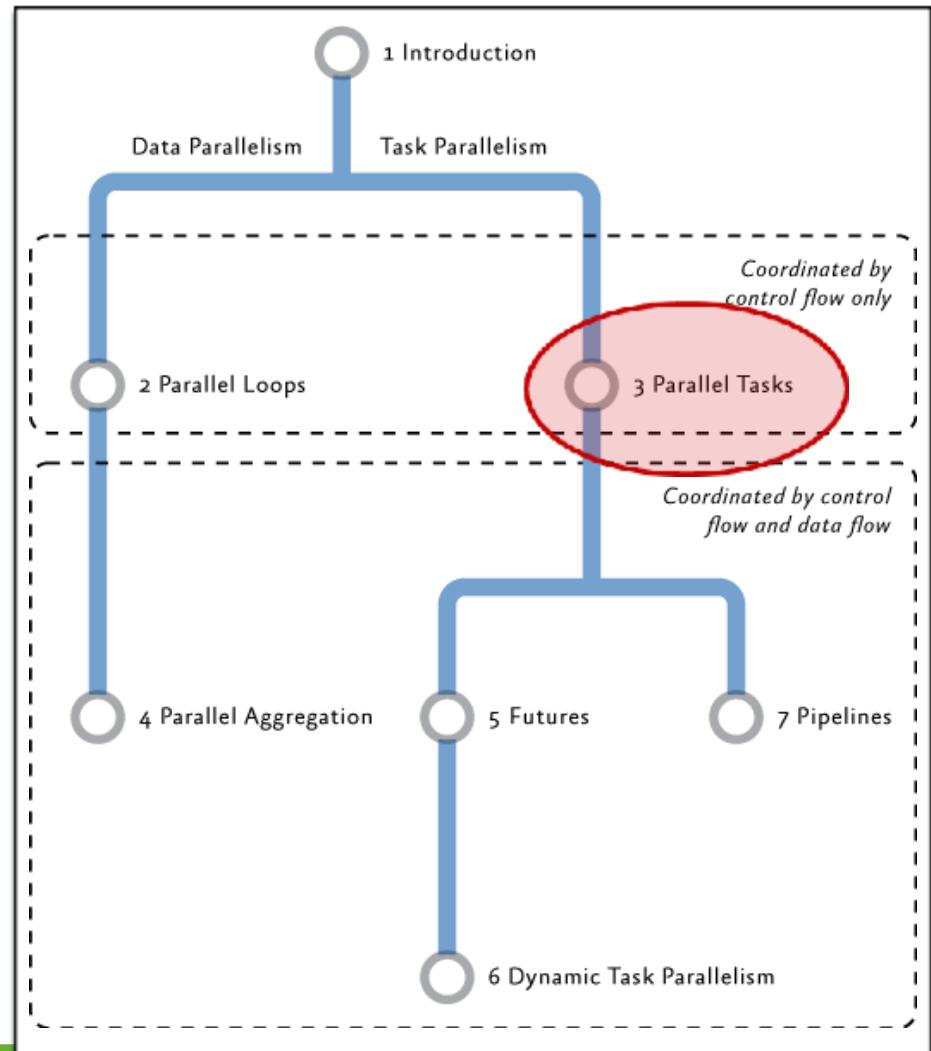
1-3 December 2010 | Beijing, China

# 演示

## 并行循环和聚合

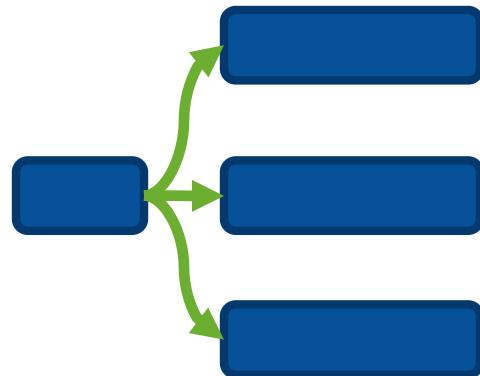


# 并行任务模式



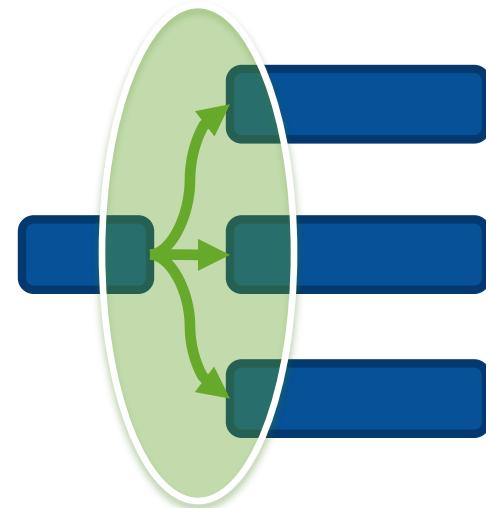
# 并行任务模式

“你是否有定义完整的控制依赖的工作单元？”



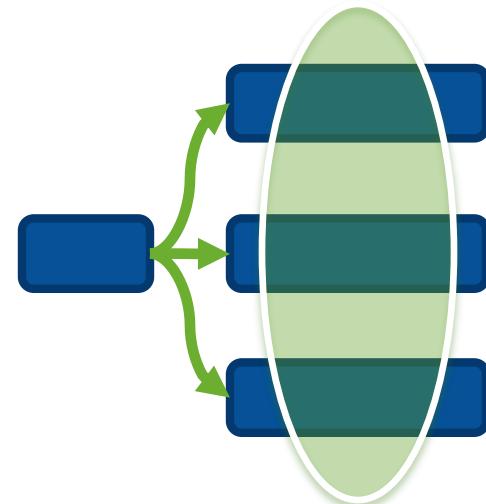
# 分割

- 如何分配工作量?
  - 固定的工作量
  - 可变的工作量
- 工作量的大小
  - 太大 – 难以平衡
  - 太小 – 通讯变得更重要



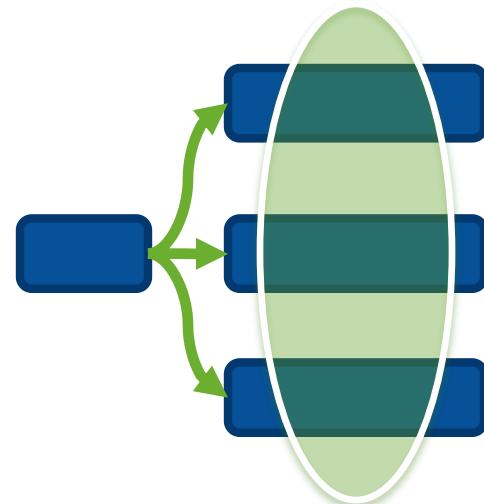
# 工作量的平衡

- **静态分配:**
  - 按照区块
  - 按照索引 (交错模式)
  - 指导
- **动态工作分配**
  - 已知的和未知的任务尺寸
  - 任务队列
  - 工作密取(Work stealing)
- **任务并行库 (TPL) 为你完成了很多工作**



# 状态共享与同步

- 不要共享!
- 数据只读
- 数据隔离
- 同步



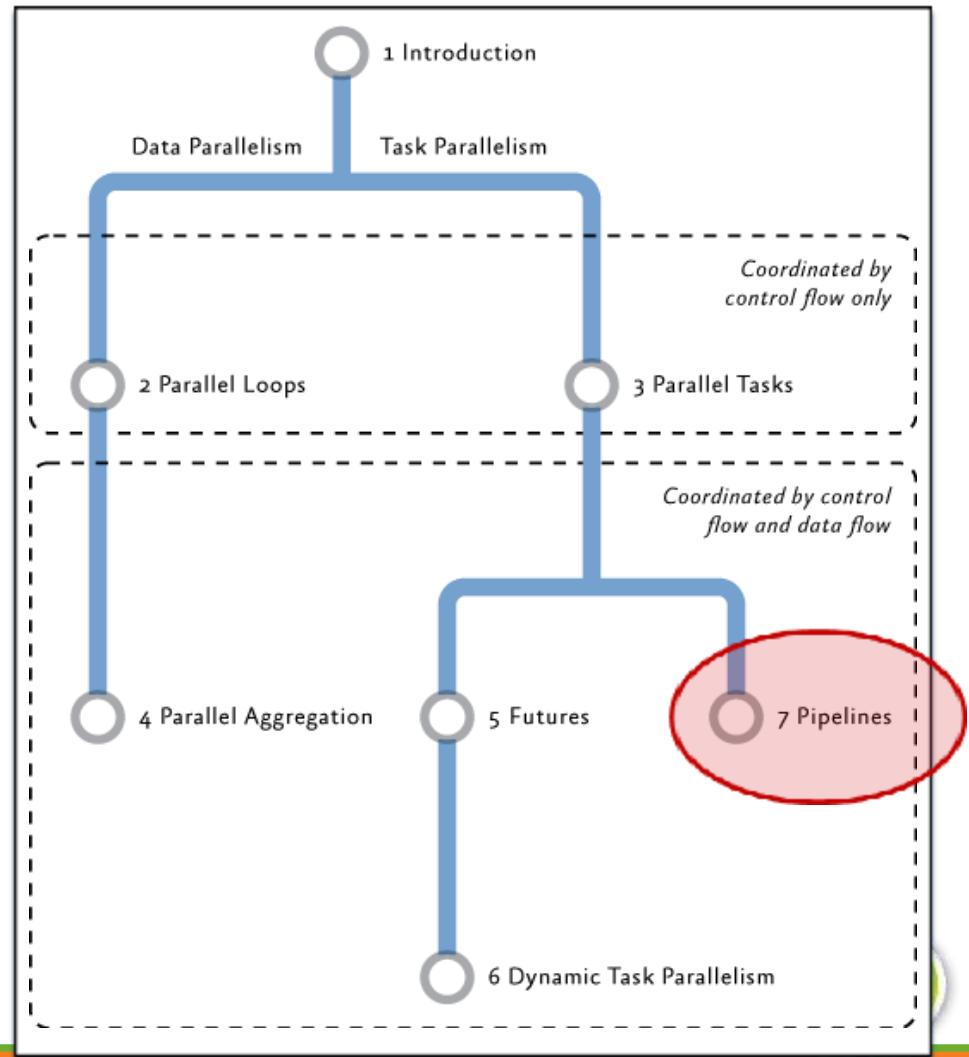
1-3 December 2010 | Beijing, China

演示

并行任务

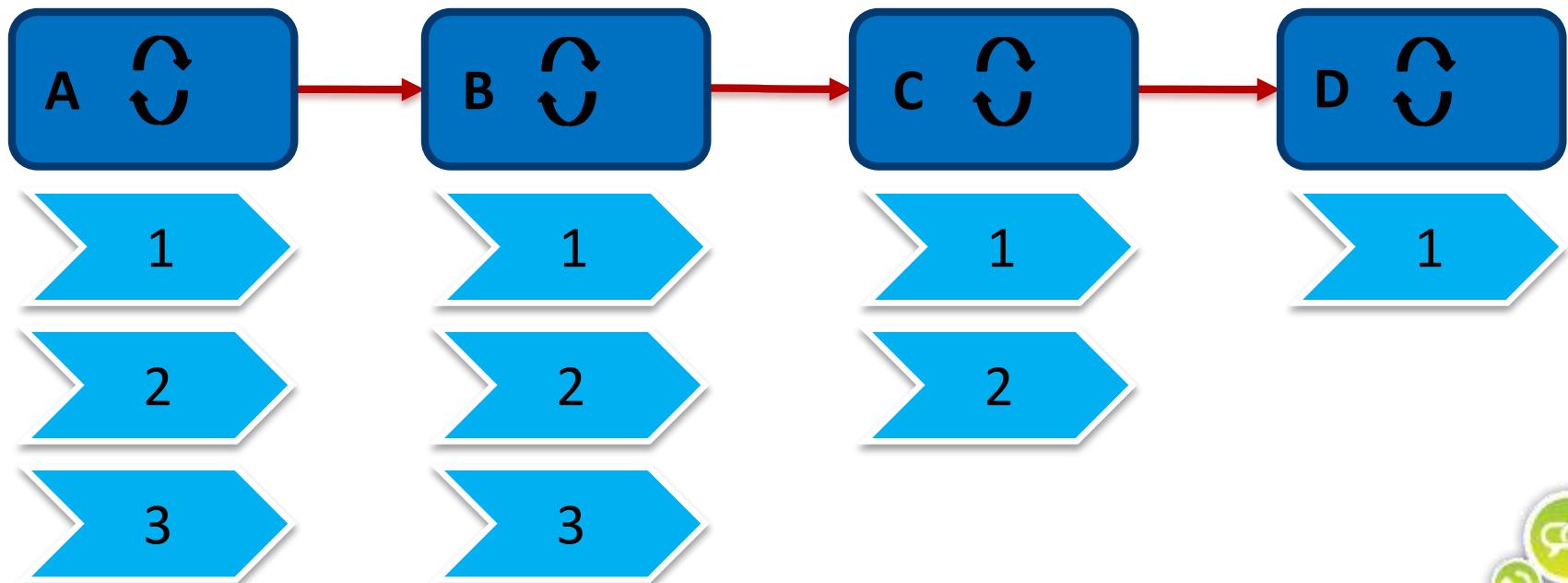


# 管道模式



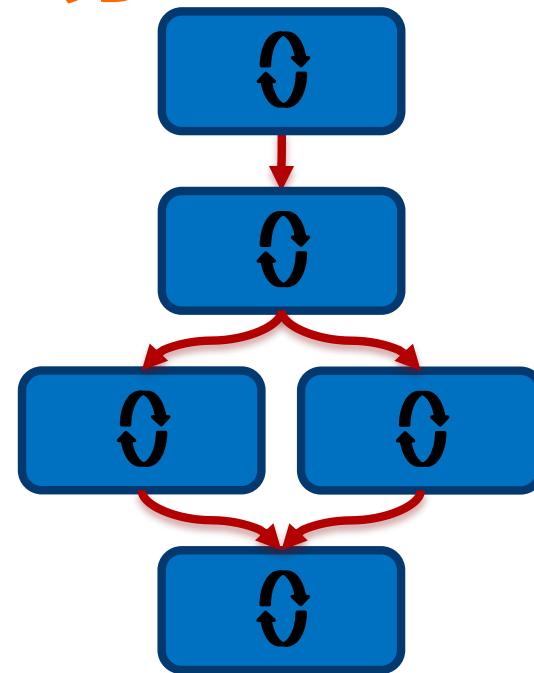
# 管道模式

“你的应用程序重复执行了操作序列? 输入的数据具有流的特点?”



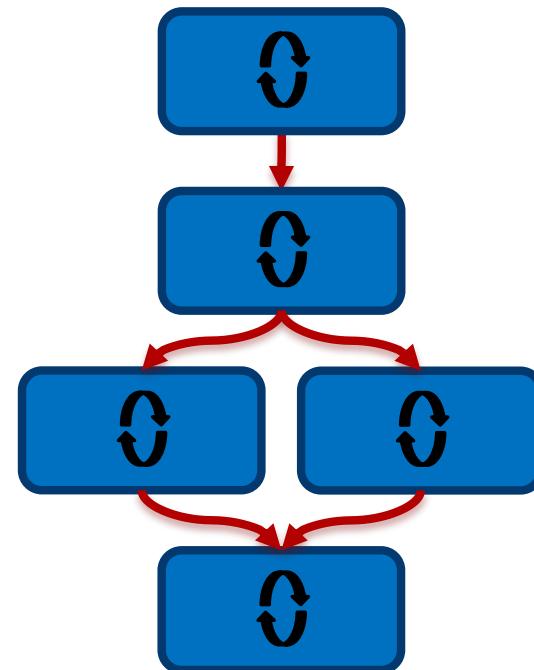
# 生产者/消费者 模式

- 先进先出(FIFO)的顺序组织任务
- 生产者... 生产!
  - 缓冲区满导致阻塞
- 消费者... 消费!
  - 缓冲区空导致阻塞
- 限制任务的进展.



# 工作量平衡

- 管道(Pipeline) 长度
  - 长 - 高吞吐
  - 短 - 低延迟
- 工作量化分
  - 平等 - 线性管道
  - 不等 - 非线性管道



# 通过管道向下传输管道

- **共享队列**

- 大队列的工作项 – 低利用率
- 小队列的工作项 – 锁的开销



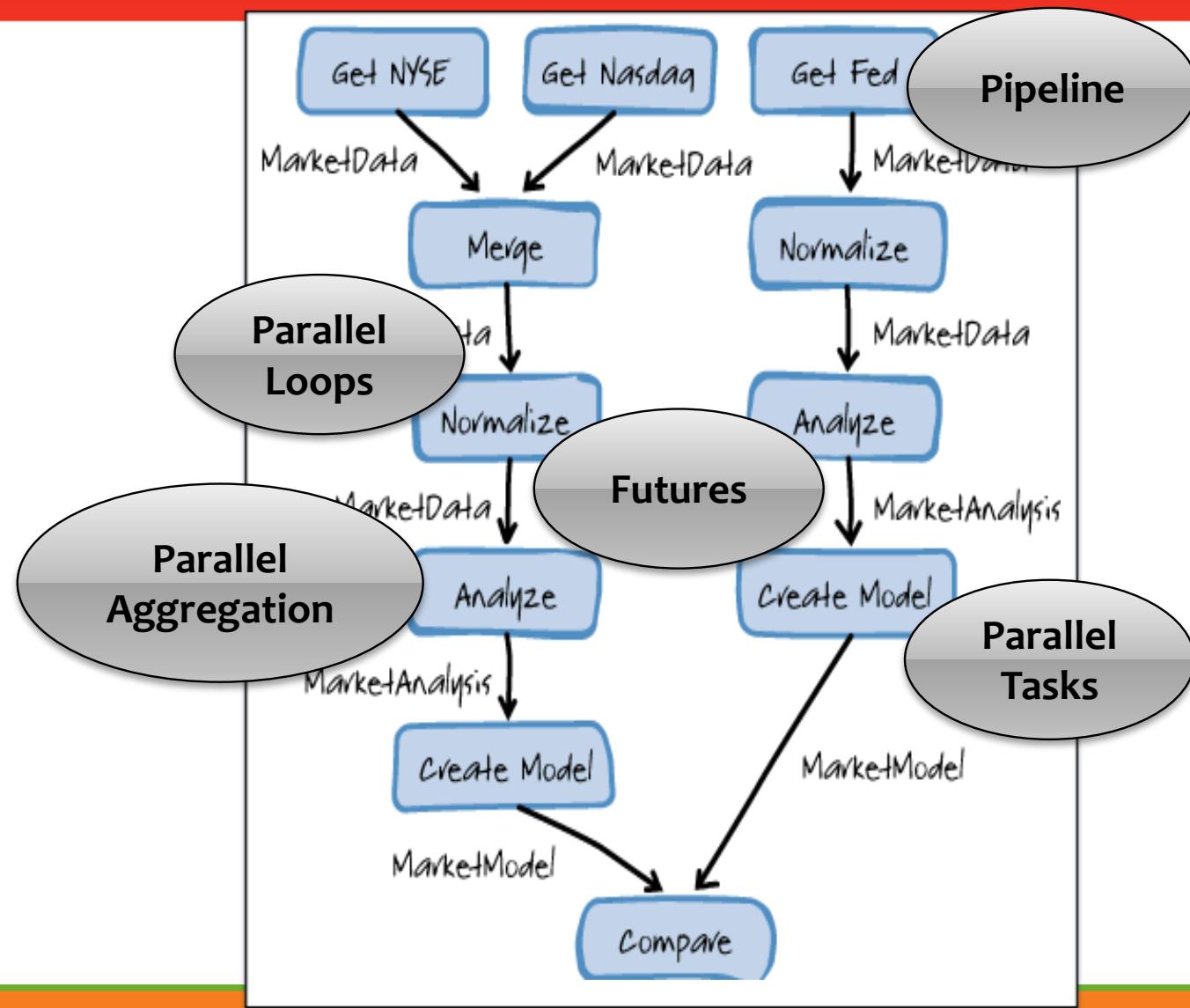
1-3 December 2010 | Beijing, China

# 演示

## 管道 (Pipeline)



# 并行的机遇



# 重要结论...



# 不仅仅是重写你的循环

- 在开始之前理解你的应用
  - 分析
  - 度量
- 给你的用户带来什么问题?
- 用并行化方法设计你的应用



# “并行化的架构”

- **发现潜在的并行**
  - 理解你的任务与数据
  - 理解它们的依赖性
- **使潜在的并行最大化**
  - 尽可能少的共享数据 (以及锁)
  - 更多的不可变的状态
- **找到模式**
  - 警惕那些看起来不适合的



# 密切注意...

- 计划和调度程序
- 异常与错误处理



# 现在读一下这本书吧!

## 利用Microsoft .NET的并行程序设计：

### 多核架构下的设计模式的分解和协调

Colin Campbell, Ralph Johnson,  
Ade Miller and Stephen Toub.  
Foreword by Tony Hey

<http://parallelpatterns.codeplex.com/>  
用 C#, Visual Basic 以及 F# 编写



1-3 December 2010 | Beijing, China

# 问 & 答

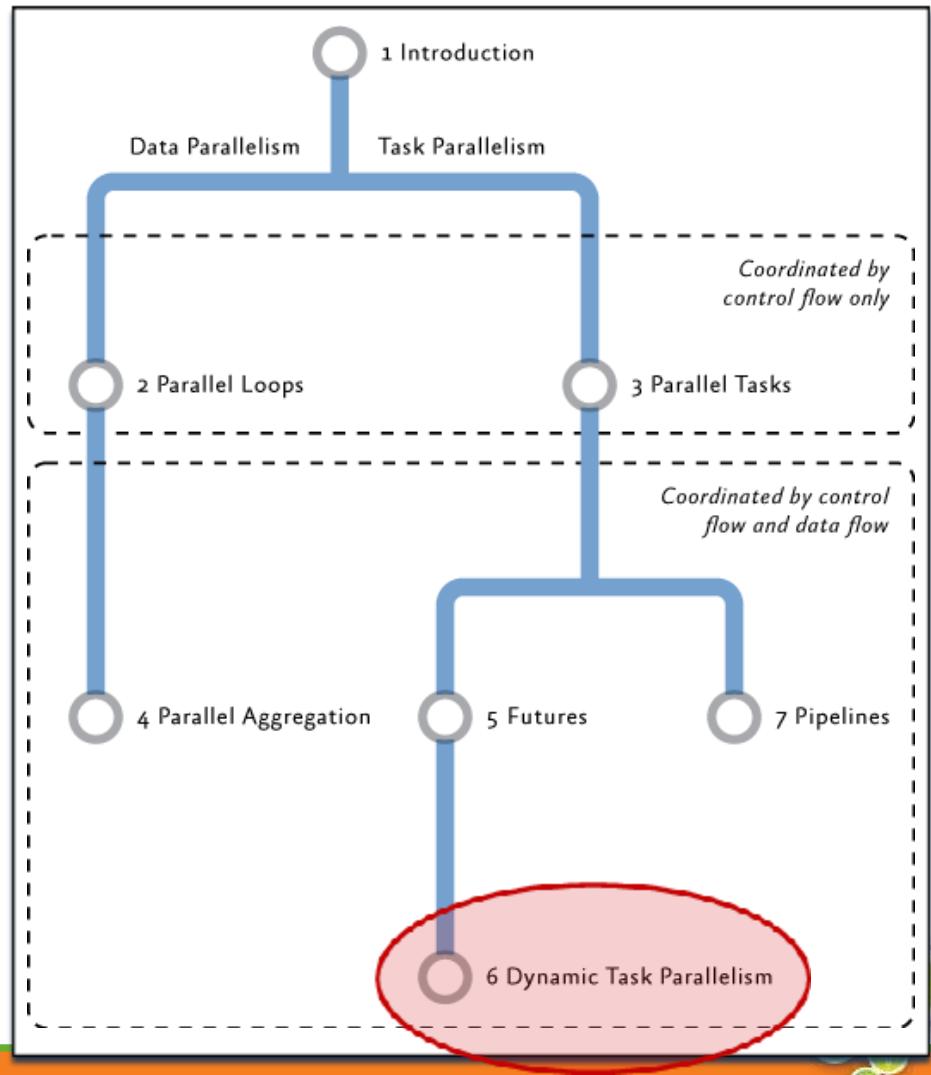


# What About Recursive Problems?

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

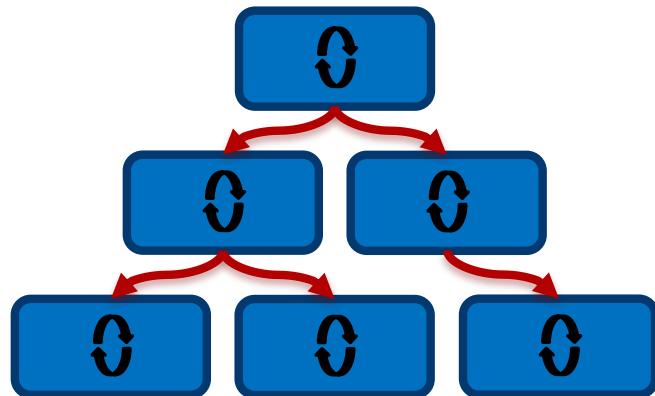


# Dynamic Task Parallelism Pattern



# 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

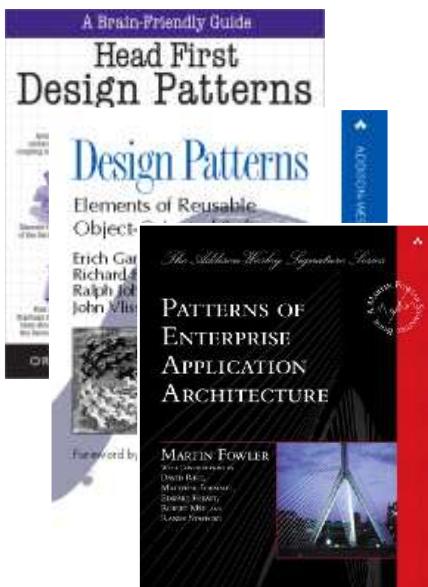
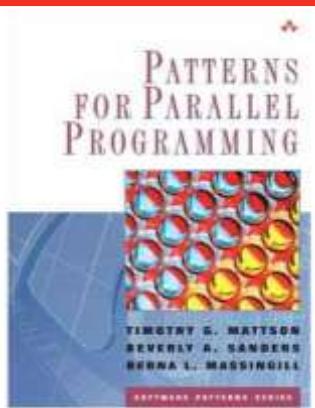


# Dynamic Task Parallelism Example

```
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);
}
```



# Other Resources



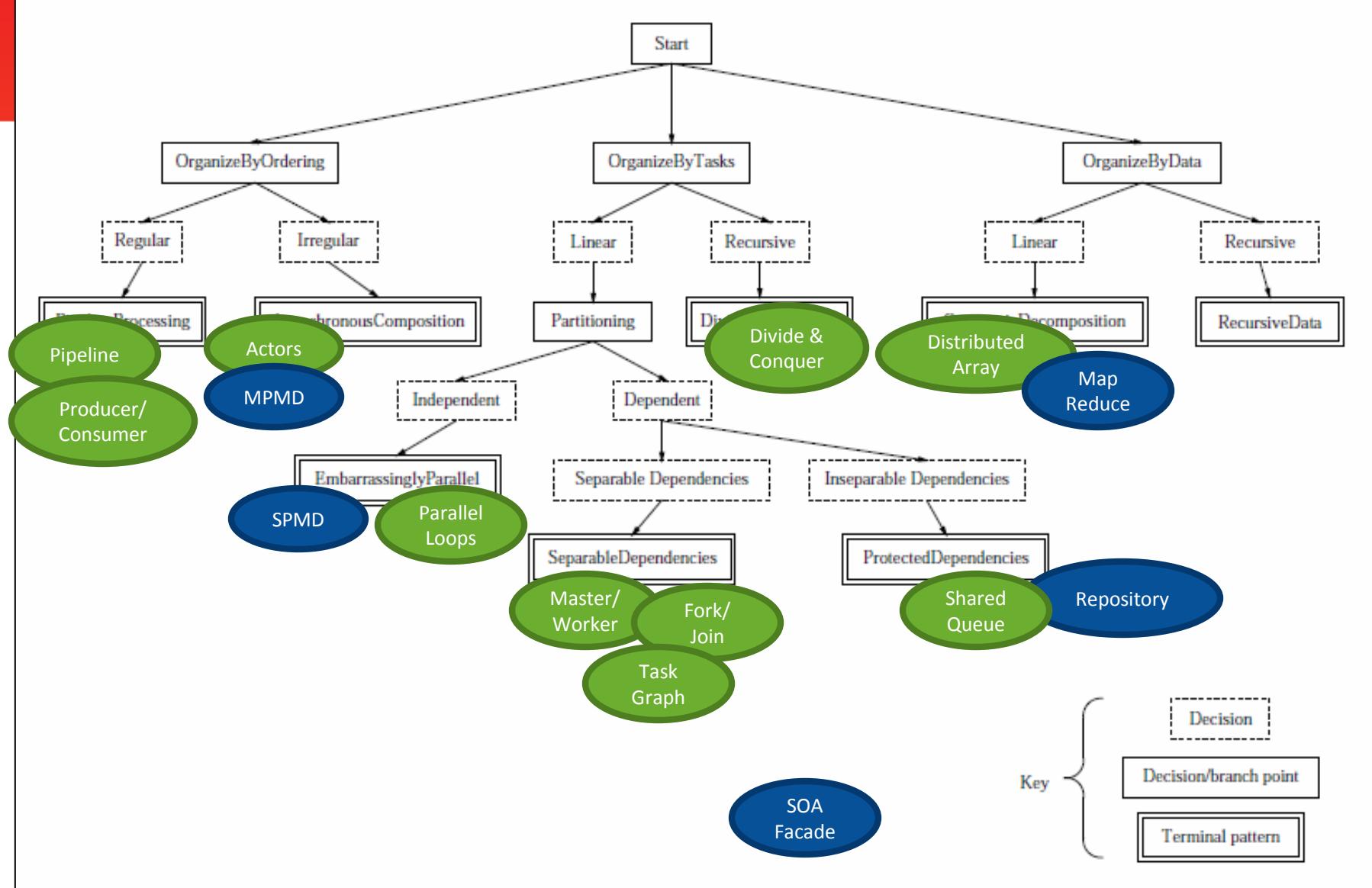
## Books

- [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/> \(Decks etc.\)](#)





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

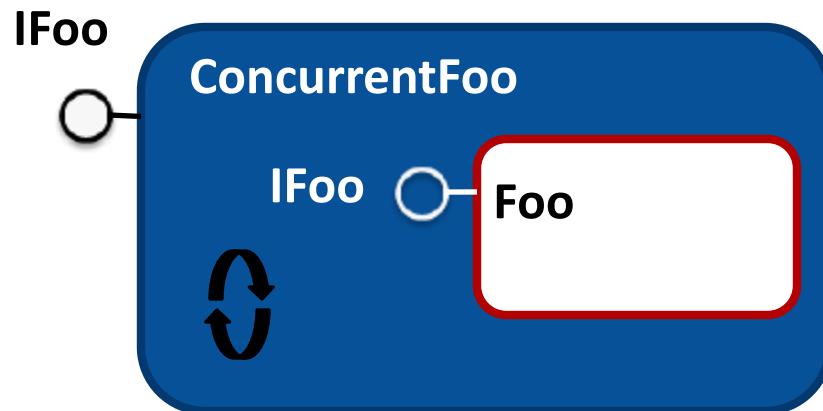
- Hide parallelism
- Optimize call granularity

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



# The Decorator Pattern

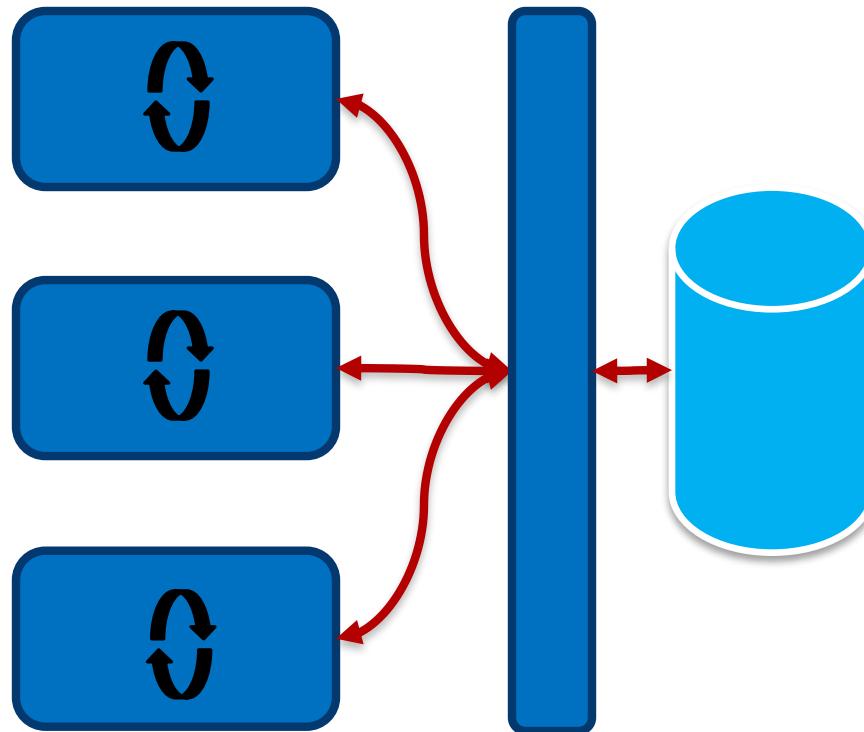
- **Encapsulate parallelism**
  - Calling code is parallel agnostic
- **Add parallelism to an existing (base) class**



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

or... 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()`



# 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 inp = new BlockingCollection<string>();  
  
var readLines = Task.Factory.StartNew(() =>  
{  
    try {  
        foreach(var line in File.ReadAllLines(@"input.txt"))  
            inp.Add(line);  
    }  
    finally { input.CompleteAdding(); }  
});  
  
var writeLines = Task.Factory.StartNew(() =>  
{  
    File.WriteAllLines(@"out.txt", inp.GetConsumingEnumerable());  
});  
Task.WaitAll(readLines, writeLines);
```



# Pipeline Examples

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





© 2008 Microsoft Corporation. All rights reserved. Microsoft, Windows, Windows Vista and other product names are or may be registered trademarks and/or trademarks in the U.S. and/or other countries.

The information herein is for informational purposes only and represents the current view of Microsoft Corporation as of the date of this presentation. Because Microsoft must respond to changing market conditions, it should not be interpreted to be a commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information provided after the date of this presentation. MICROSOFT MAKES NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, AS TO THE INFORMATION IN THIS PRESENTATION.