Parallelism : The challenge for mainstream developers

Bringing Parallel Computing to the mass

Eric Vernié
Relation technique développeur
Microsoft France

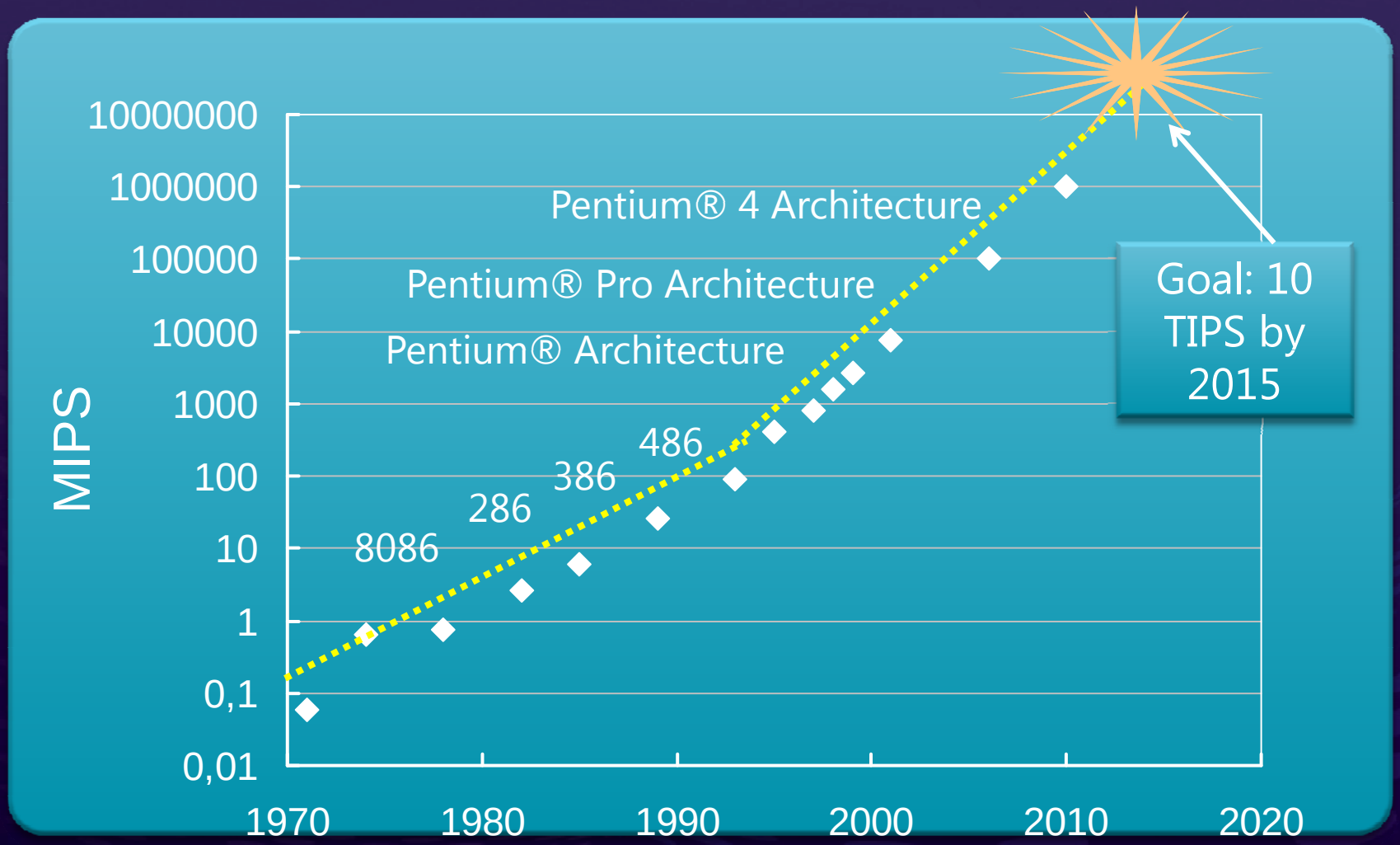
Microsoft

Visual Studio 2010

Gartner : 7 grand challenges

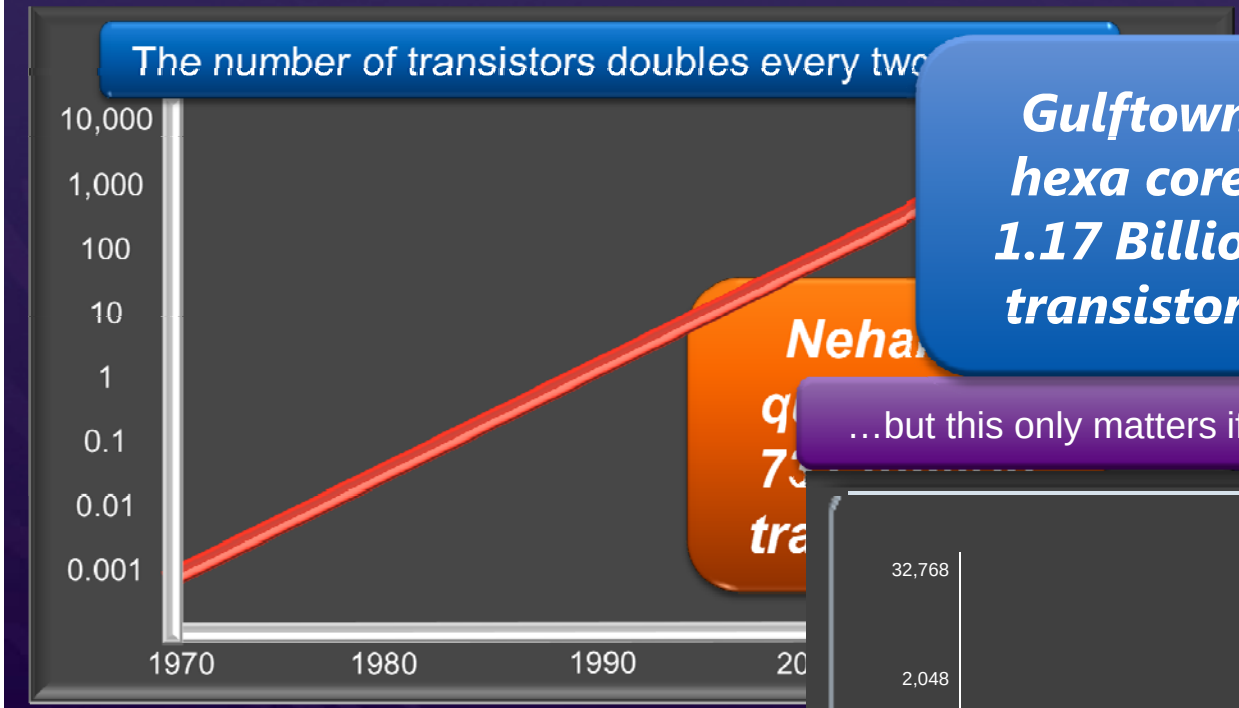
- Never having to manually recharge devices
- Parallel Programming
- Non Tactile, Natural Computing Interface
- Automated Speech Translation
- Persistent and Reliable Long-Term Storage
- Increase Programmer Productivity 100-fold
- Identifying the Financial Consequences of IT Investing

The physical laws change the rules



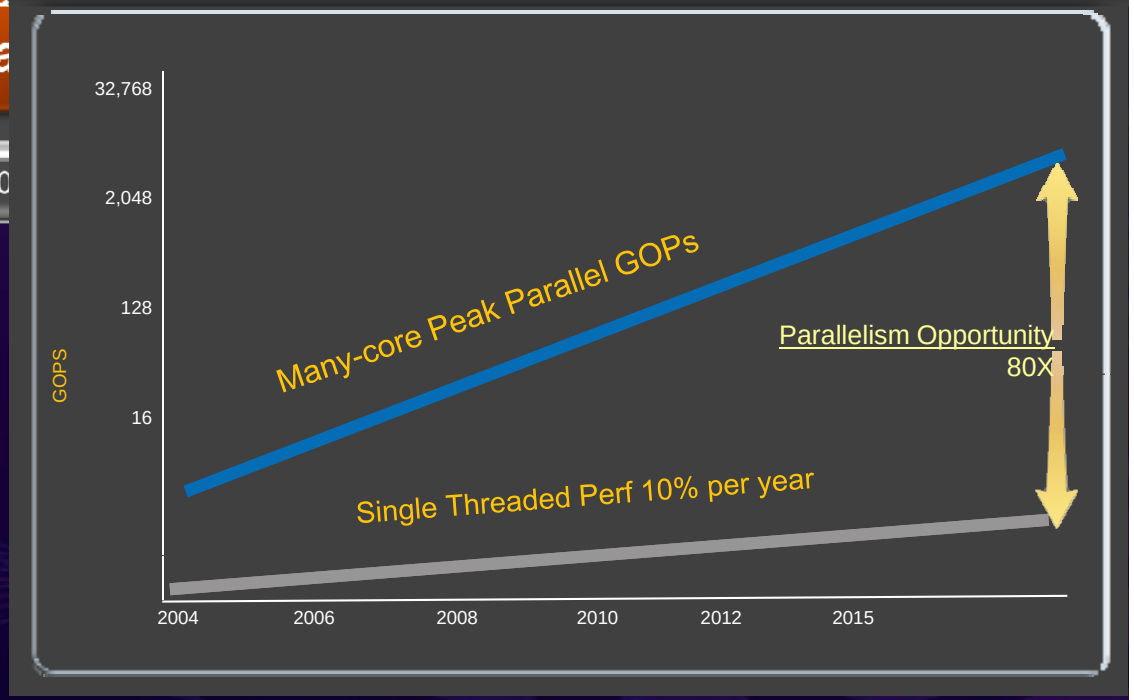
Source: Intel Juin 2006

The Manycore Shift



Gulftown hexa core: 1.17 Billion transistors

…but this only matters if developers embrace parallel computing!



In late 2008, only 3% of developers introduced parallelism in their applications

Evans Data

for mainstream developers, a new story begin…

The manycore shift is a ***disruption*** for the developers

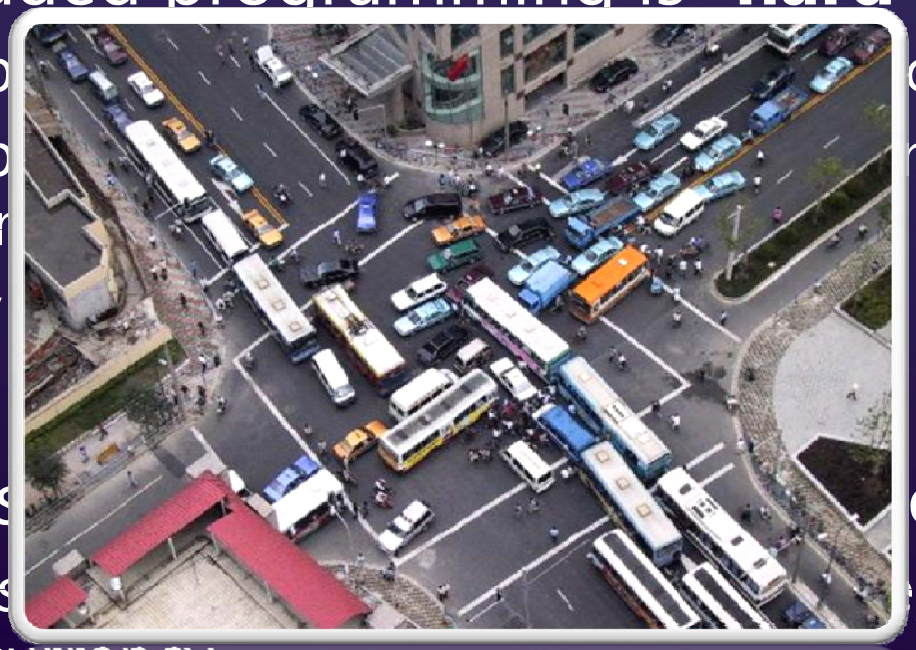
Free lunch is over

*If you haven't done so already, **now is the time** to take a hard look at the design of your application, determine what operations are CPU-sensitive now or are likely to become so soon, and **identify how those places could benefit from concurrency**."*

Herb Sutter, C++ Architect at Microsoft (March 2005)

What's the problem?

- Multithreaded programming is "**hard**" today
  - Doable b cialists
  - Parallel p nown, nor easy to impler
  - So many



- Businesses ep"
  - Best devs , not concurrency
  - Need simple ways to allow all devs to write concurrent code

So this is the challenge that we are focus on with our partner .

"Enable people to do parallelism without the need to go deep and make parallelism accessible to a large number of software developers."

Steve Texeira
Product Unit Manager
Parallel Developer Tools
Microsoft Corporation

Microsoft® Parallel Computing Initiative

Microsoft's Parallel Computing Initiative encompasses the vision, strategy, and innovative technologies for delivering transformative, natural, and immersive personal computing experiences, that harness the computing power of manycore architectures.

The manycore is a ***disruption*** for the developer but it's also ***a great opportunity***

It begins with DEVELOPERS

Make it easier to express and manage the correctness, efficiency and maintainability of parallelism on Microsoft platforms for developers of all skill levels

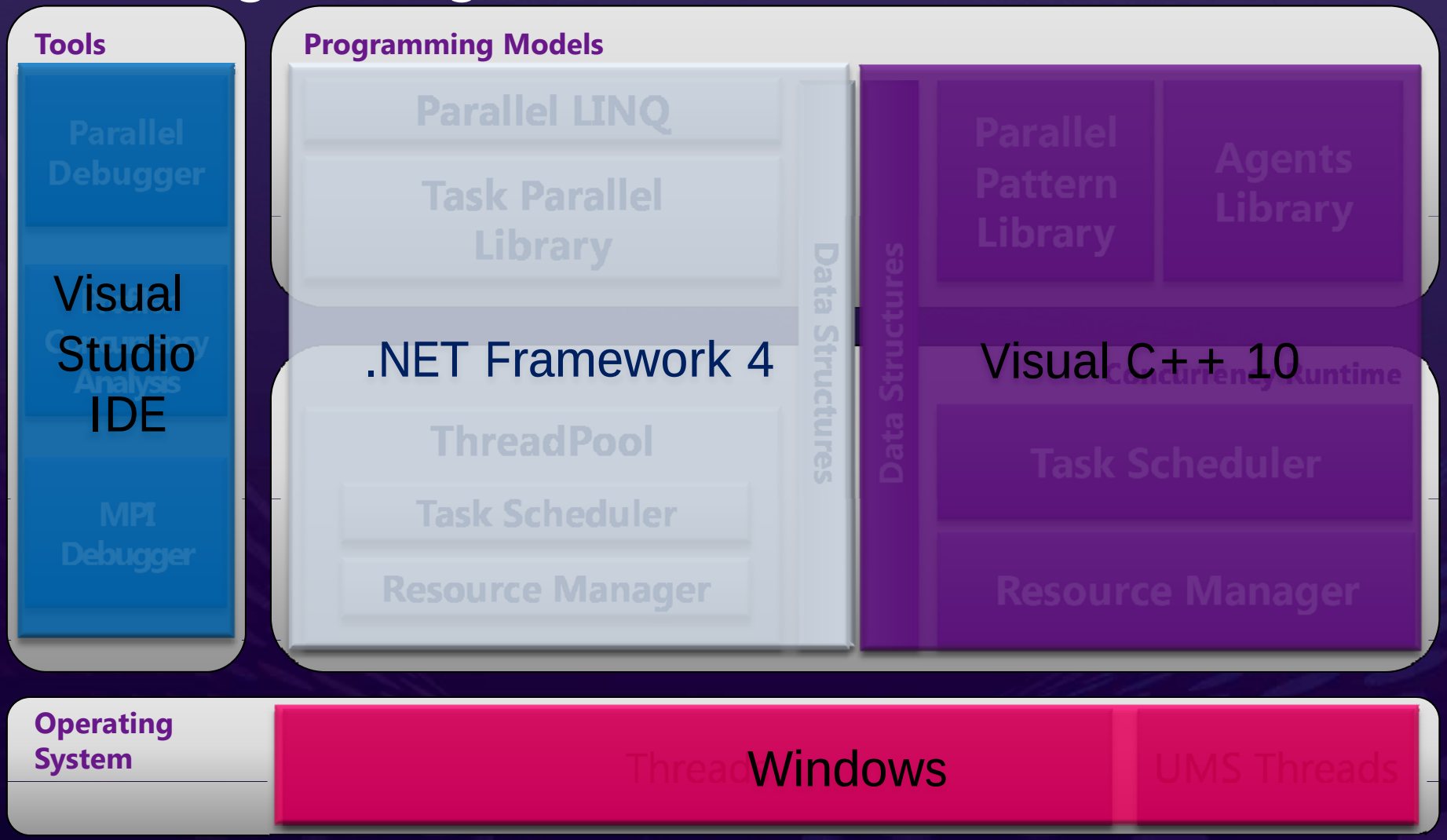
Enable developers to express parallelism easily and focus on the problem to be solved

Improve the efficiency and scalability of parallel applications

Simplify the process of designing and testing parallel applications

Visual Studio 2010

Tools, Programming Models, Runtimes



Key: Managed | Native | Tooling

Tasks in .NET and C++

.NET 4.0

- Parallel.For(x, y, λ)
- Parallel.ForEach(IEnum, λ)
- Parallel.Invoke(λ, λ)
- Task
- Task.Factory.StartNew(λ)
- ThreadPool-based

Visual C++ 10

- parallel_for(x, y, step, λ);
- parallel_for_each(it, λ)
- parallel_invoke(λ, λ)
- task_group / task_handle
- task_group::run (λ)
- Native concurrency runtime

(and many overloads for the above)

Matrice Multiplication

```
void MatrixMult(
    int size, double** m1, double** m2, double** result)
{
    for (int i = 0; i < size; i++) {
        for (int j = 0; j < size; j++) {
            result[i][j] = 0;
            for (int k = 0; k < size; k++) {
                result[i][j] += m1[i][k] * m2[k][j];
            }
        }
    }
}
```

Manual solution

```
void MatrixM
    int siz      le** m1, double** m2,      ** result) {
  int N =
  int P =      S;
  int Chu
  HANDLE h      Event(NUL      , NULL);
  long coun
  for (int c      ++) {
    std::threa
      for (int
          i < (c      * Chunk); i++) {
        for (int j      {
          result[i]
          for (int k      k++) {
            result[i      * m2[k][j];
          }
        }
      }
      if (Inter      0)
        SetEv
    });
  }
  WaitFor      Event,INFI
  CloseH
}
```

"Nontrivial Software written with threads, semaphores, and mutexes are incomprehensible to humans and cannot and should not be trusted"

Edward A. Lee
UC Berkeley

```
void MatrixMult(
    int size, double** m1, double** m2, double** result)
{
    parallel_for(0, size, 1, [&](int i) {
        for (int j = 0; j < size; j++) {
            result[i][j] = 0;
            for (int k = 0; k < size; k++) {
                result[i][j] += m1[i][k] * m2[k][j];
            }
        }
    });
}
```

Example: “Race Car Drivers”

```
IEnumerable<RaceCarDriver> drivers = ...;
var results = new List<RaceCarDriver>();
foreach(var driver in drivers)
{
    if (driver.Name == queryName &&
        driver.Wins.Count >= queryWinCount)
    {
        results.Add(driver);
    }
}
results.Sort((b1, b2) =>
    b1.Age.CompareTo(b2.Age));
```

Manual Parallel Solution

```
IEnumerable<RaceCarDriver> drivers = ...;
var results = new List<RaceCarDriver>();
int partitionsCount = Environment.ProcessorCount;
int remainingCount = partitionsCount;
var enumerator = drivers.GetEnumerator();
try {
   using (var done = new ManualResetEvent(false)) {
      for(int i = 0; i < partitionsCount; i++) {
         ThreadPool.QueueUserWorkItem(delegate {
            while(true) {
               RaceCarDriver driver;
               lock (enumerator) {
                  if (!enumerator.MoveNext()) break;
                  driver = enumerator.Current;
               }
               if (driver.Name == queryName &&
                  driver.Wins.Count >= queryWinCount) {
                     lock(results) results.Add(driver);
               }
            }
            if (Interlocked.Decrement(ref remainingCount) == 0) done.Set();
         });
      }
      done.WaitOne();
      results.Sort((b1, b2) => b1.Age.CompareTo(b2.Age));
   }
}
finally { if (enumerator is IDisposable) ((IDisposable)enumerator).Dispose(); }
```

LINQ Solution

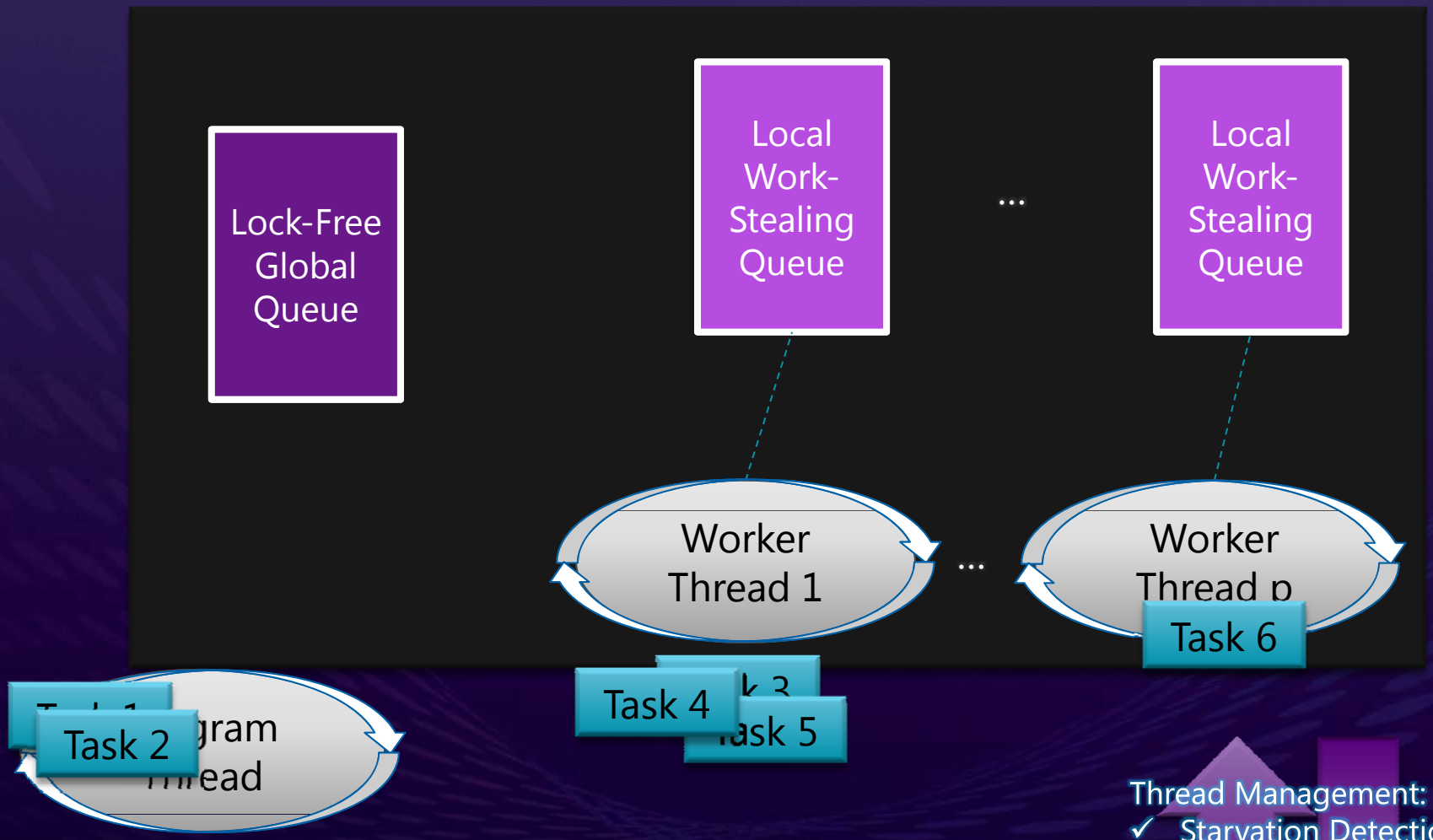
```
var results = from driver in drivers
              where driver.Name == queryName &&
                    driver.Wins.Count >= queryWinCount
              orderby driver.Age ascending
              select driver;
```

.AsParallel()

Concurrency Runtime

- Programming-Model-Independent
- Scheduler
  - Low-overhead scheduling and load balancing
  - Policy-driven
  - Multiple scheduler instances, cooperative blocking, supports UMS on Win7
  - APIs
    - Scheduler, SchedulerPolicy, ScheduleGroup, Context
- Resource Manager
  - Serves requests for physical processors, intra-process
  - Dynamic load balancing between scheduler instances
  - Interfaces
    - IResourceManager, IScheduler, IVirtualProcessorRoot, IThreadProxy, IExecutionContext

Work-Stealing Scheduler

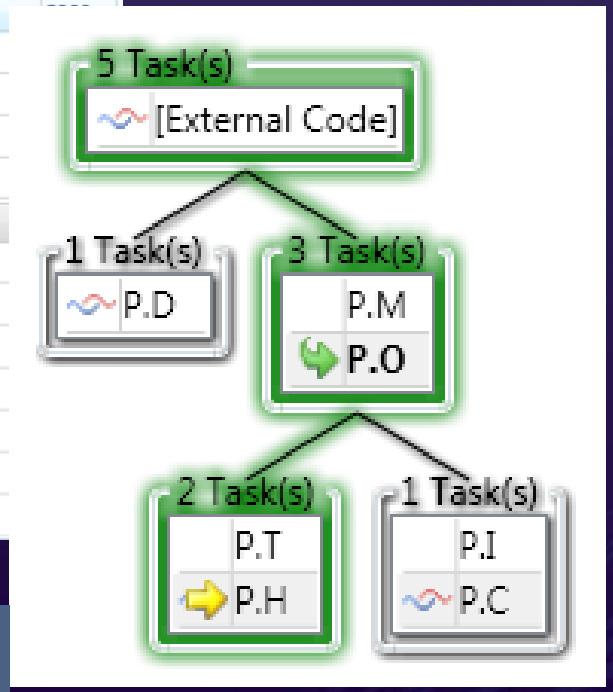


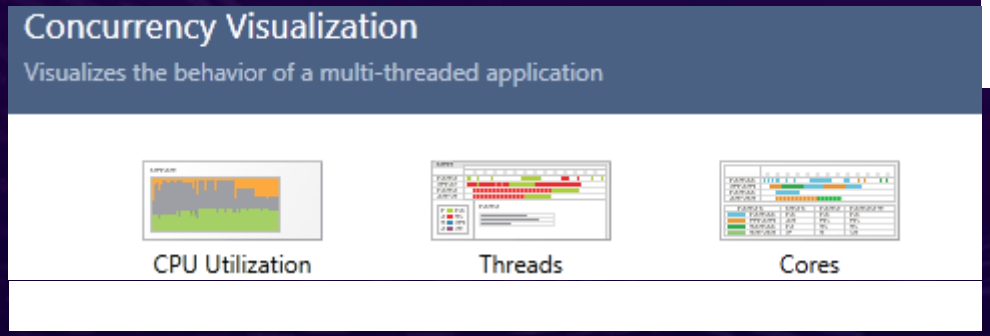
Thread Management:
- ✓ Starvation Detection
- ✓ Idle Thread Retirement
- ✓ Hill-climbing

Debugging & profiling tools

Tasks

Id.	Status	Location	Parent Task	Thread
1242864	Waiting	Concurrency::StructuredTaskPool::Wait		
1963792	Waiting	Concurrency::StructuredTaskPool::Wait	1242864	
1961324	Waiting	Concurrency::StructuredTaskPool::Wait	1963792	
1958136	Waiting	Concurrency::StructuredTaskPool::Wait	1961324	
1954948	Scheduled	0x004110be CChore::Payload(void *)	1958136	
1955128	Running	State::Remove	1958136	
1243044	Waiting	Concurrency::StructuredTaskPool::Wait		
1701972	Waiting	Concurrency::StructuredTaskPool::Wait	1243044	
1698964	Waiting	Concurrency::StructuredTaskPool::Wait	1701972	
1695776	Scheduled	0x004110be CChore::Payload(void *)	1698964	
1695956	Waiting	Concurrency::StructuredTaskPool::Wait	1698964	
1692108	Scheduled	0x004110be CChore::Payload(void *)	1695956	
1692288	Running	State::Solve	1695956	





Baby Step

Thinking Parallel

As thinking Parallel becomes intuitive, structuring problems to scale will become second nature

James Reinder

Chief Software Evangelist and Director of Software Development Products at Intel Corporation

question & answer