Practical Parallel and Concurrent Programming

Performance and Correctness on the Multi-core Computer with .NET 4

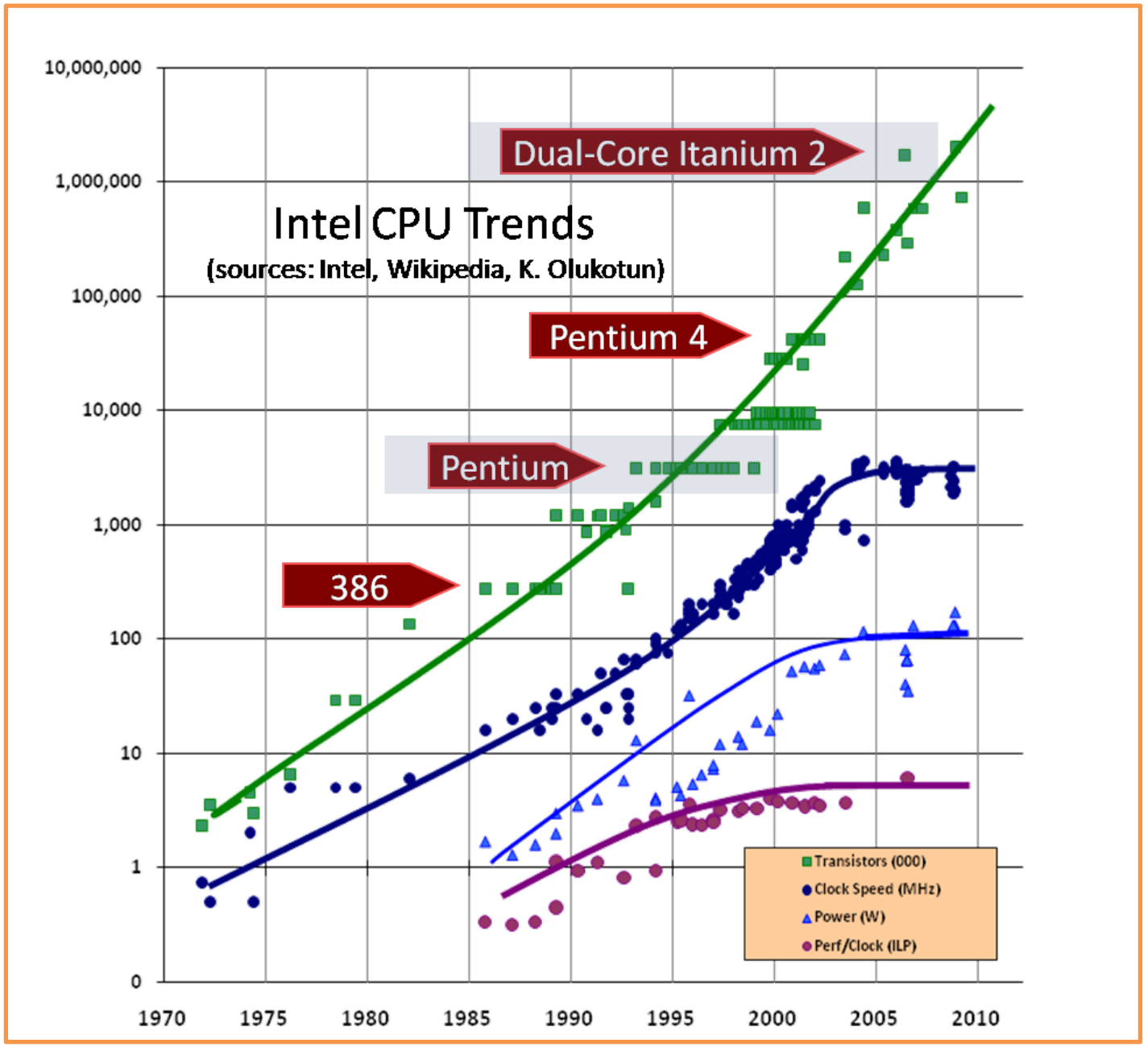
http://ppcp.codeplex.com/

Microsoft Research
University of Utah
University of California – Santa Cruz

Thomas Ball, Sebastian Burckhardt, Ganesh Gopalakrishnan, Joseph Mayo, Madan Musuvathi, Shaz Qadeer, Caitlin Sadowski

Outline

- Context, motivation
  - Parallelism *and* concurrency
  - Performance *and* correctness
- PPCP courseware
  - What is it?
  - Seven principles

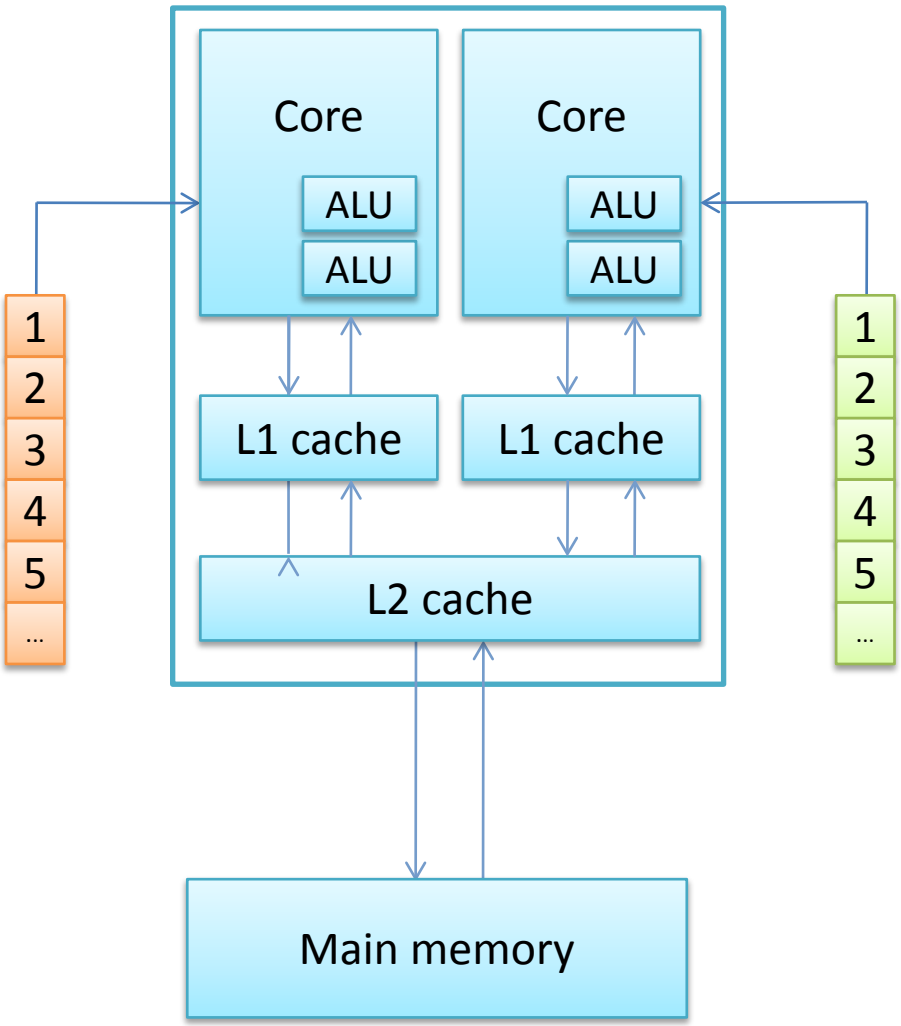


from Burton Smith

Power wall + ILP wall + memory wall = **BRICK WALL**

- *Power wall*
  - we can't clock processors faster
- *Memory wall*
  - many workloads' performance is dominated by memory access times
- *Instruction-level parallelism (ILP) wall*
  - we can't find extra work to keep functional units busy while waiting for memory accesses

Response #1: multi-core computer



From Tim Harris's

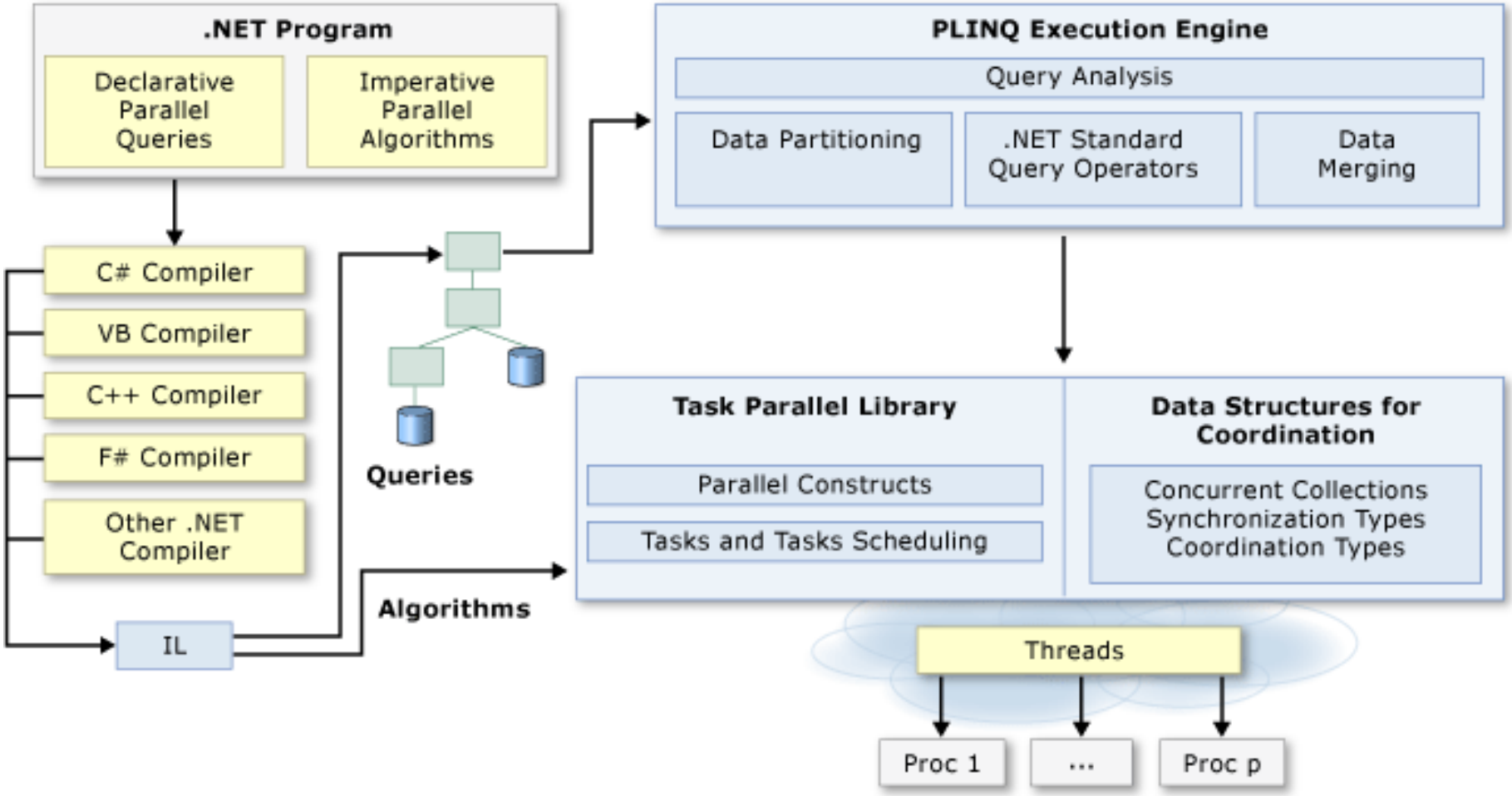
The Dream

Pleased Programmers Productively Playing with Parallel Platforms

Response #2: Libraries for Parallelism and Concurrency

- Boost
- Intel's Thread Building Blocks
- java.util.concurrent
- Microsoft's .NET 4.0
- …

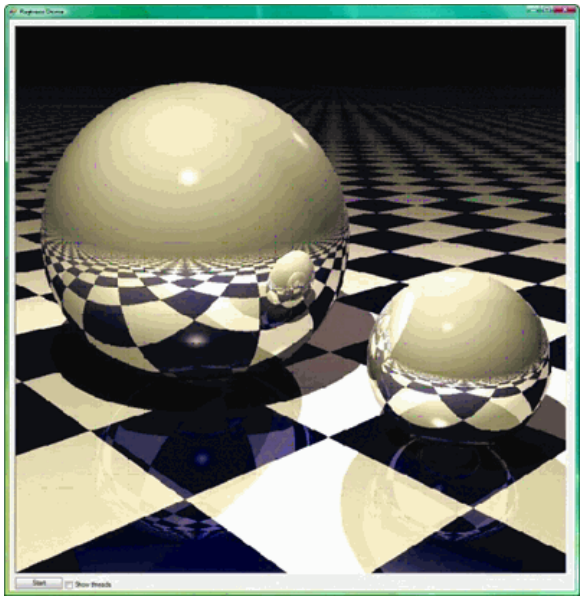
.NET 4 Libraries for Parallelism and Concurrency



Underlying Research

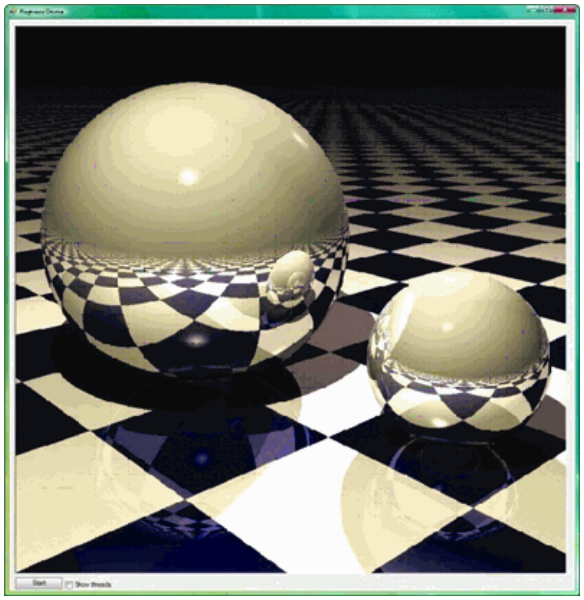
- Lock-free data structures
- Work stealing scheduling
- Parallel algorithms
- Patterns
- Languages

Demo



Sequential Ray Tracing

```
void Render(Scene scene, Color[,] rgb)
{
  for (int y = 0; y < screenHeight; y++)
  {
    for (int x = 0; x < screenWidth; x++) {
      rgb[x,y] = TraceRay(new Ray(scene,x,y));
    }
  }
}
```



Parallel Ray Tracing with **Parallel.For** and *Lambda*

```
void Render(Scene scene, Color[,] rgb)
{
  Parallel.For(0, screenHeight,  (y) =>
  {
    for (int x = 0; x < screenWidth; x++) {
      rgb[x,y] = TraceRay(new Ray(scene,x,y));
    }
  } );
}
```

Baby Data

```
class BabyInfo
{
    public string Name { get; set; }
    public string State { get; set; }
    public int Year { get; set; }
    public int Count { get; set; }
}
```

```
List<BabyInfo> _babies;
```

LINQ = Language Integrated Queries

```
IEnumerable<BabyInfo> _sequentialQuery =

    _babies
    .Where(b => b.Name.Equals(_userQuery.Name) &&
           b.State == _userQuery.State &&
           b.Year >= YEAR_START && b.Year <= YEAR_END)
    .OrderBy(b => b.Year);
```

PLINQ = LINQ + AsParallel

```
ParallelQuery<BabyInfo> _parallelQuery =

    _babies.AsParallel()
    .Where(b => b.Name.Equals(_userQuery.Name) &&
           b.State == _userQuery.State &&
           b.Year >= YEAR_START && b.Year <= YEAR_END)
    .OrderBy(b => b.Year);
```

“There is more to life than merely increasing its speed.”

Gandhi

Technology Trends

- Increasing *parallelism* in a "computer"
  - multi-core CPU (desktop, laptop, cellphone, …)
  - graphical processing unit (GPU)
  - cloud computing (Amazon, Google, Microsoft)

- Increasing *disk capacity*
  - we are awash in interesting *data*
  - data-intensive problems require *parallel processing*

Technology Trends (2)

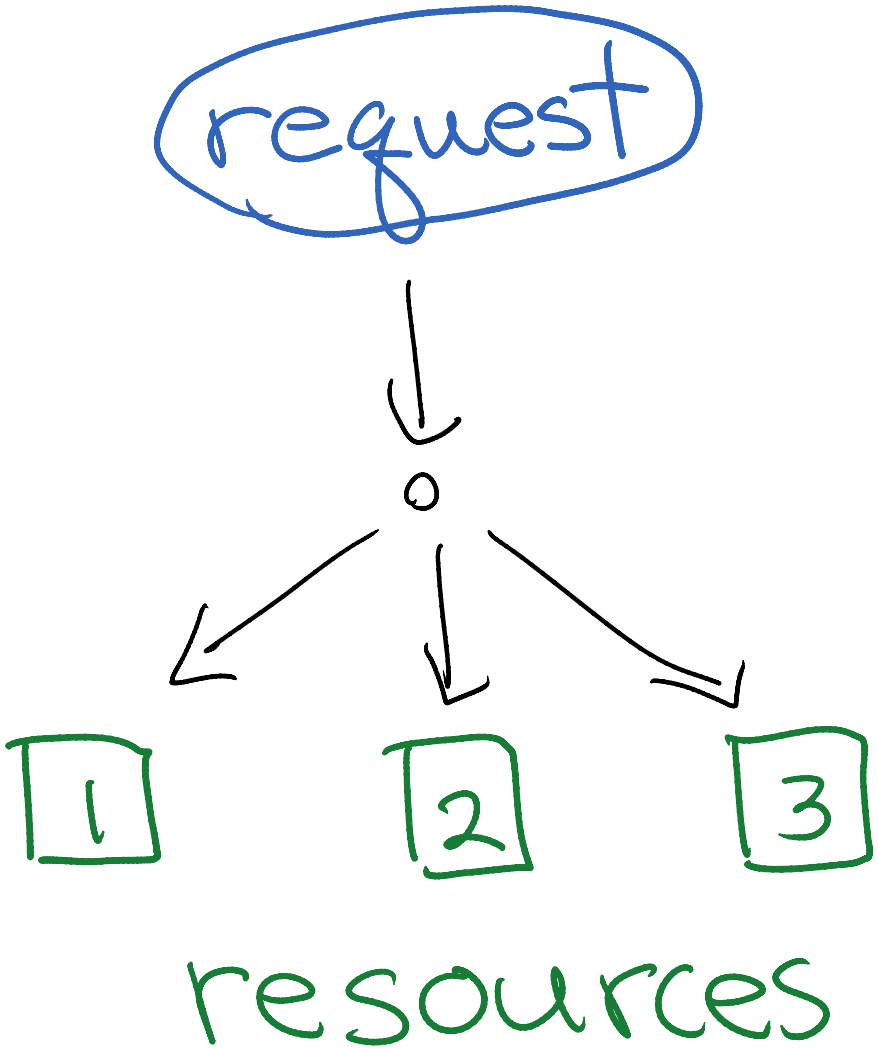
- Increasing *networks and network bandwidth*
  - wireless, wimax, 3G, …
  - collection/delivery of massive datasets, plus
  - real-time responsiveness to asynchronous events

- Increasing *number and variety of computers*
  - smaller and smaller, and cheaper to build
  - generating streams of asynchronous events

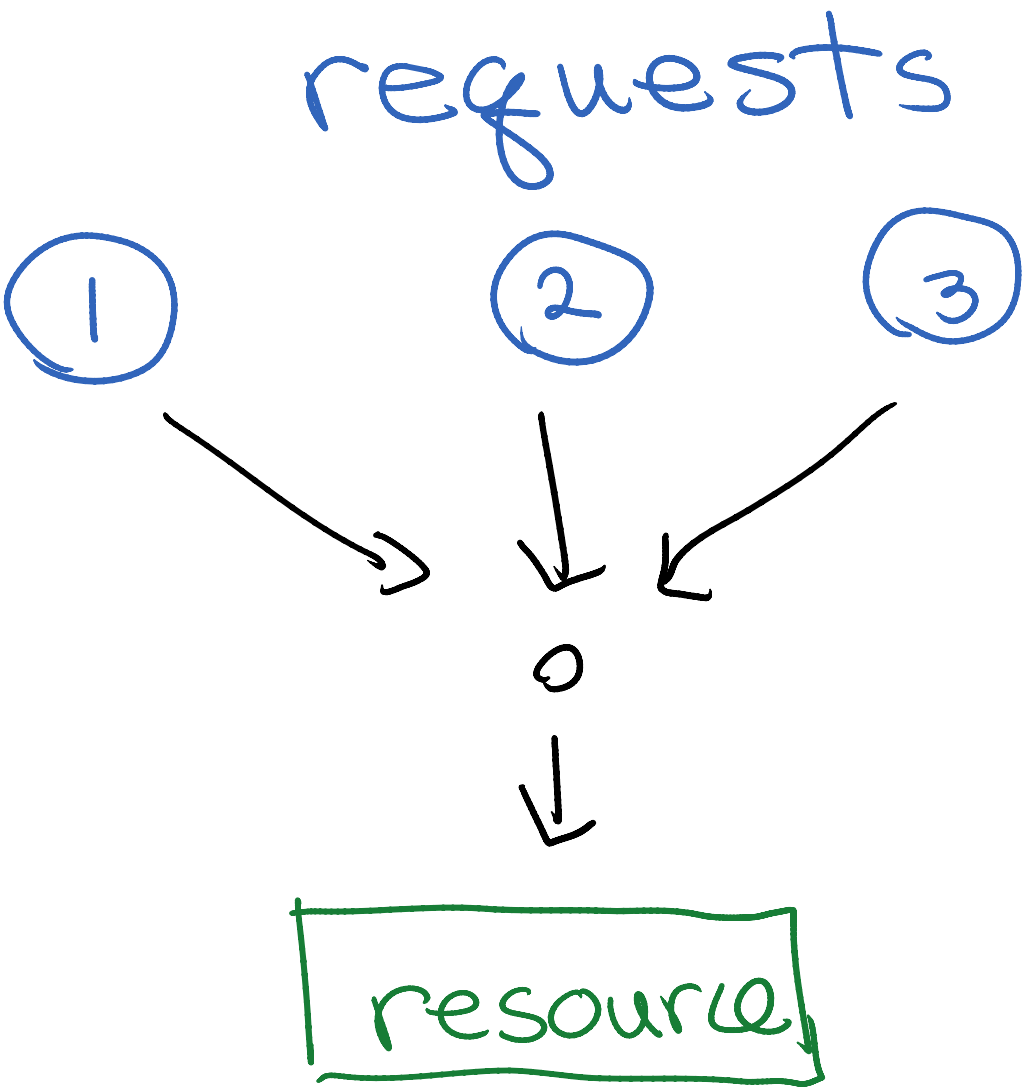
Mind Your P's and C's

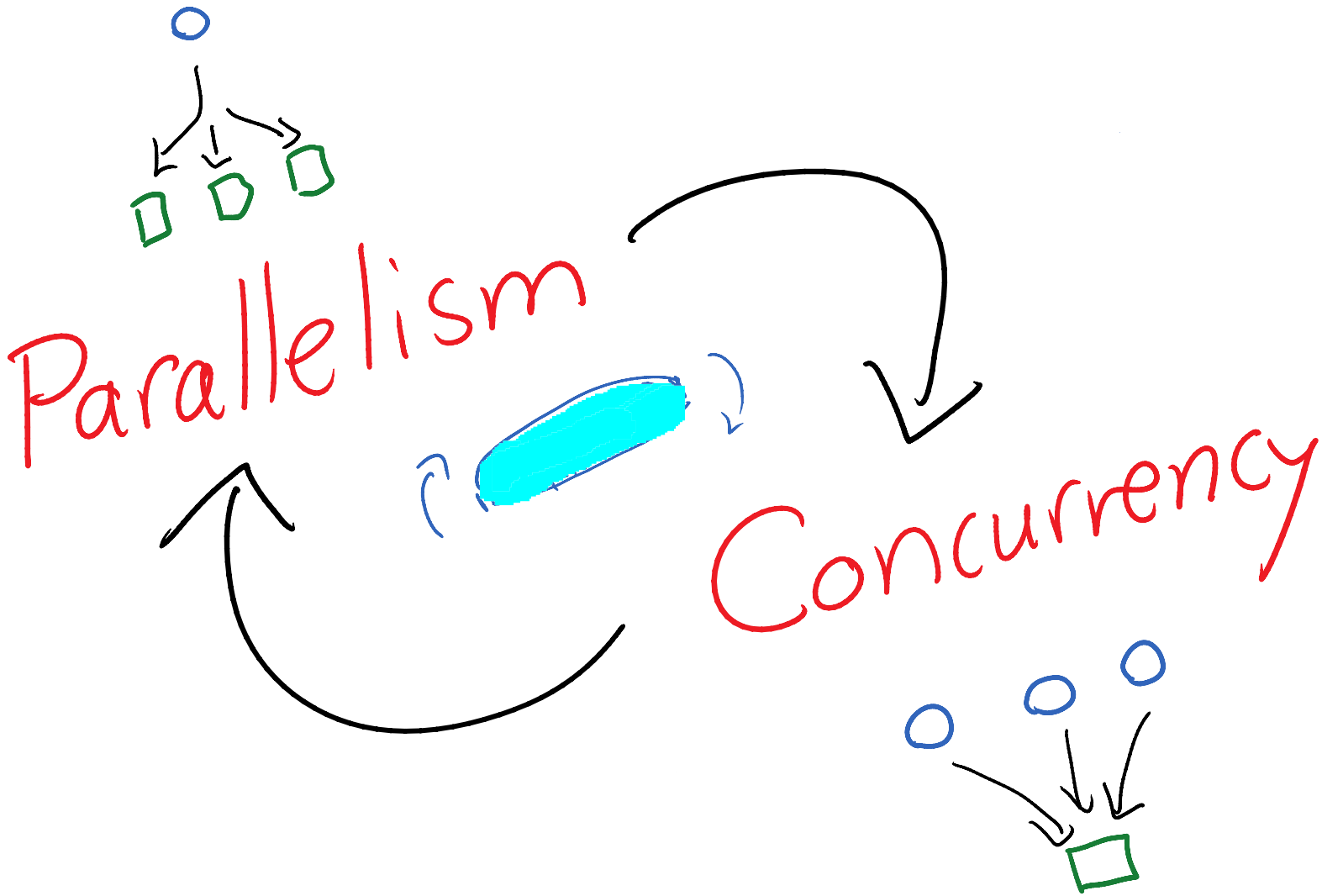
	Parallelism	**Concurrency**
Performance	*Speedup*	*Responsiveness*
Correctness	Data race freedom, Determinism, Deadlock, Livelock, Linearizability, …	

Parallelism



Concurrency




Parallelism
Concurrency

Performance & Correctness

"There are two ways of constructing a software design: One way is to make it so *simple* that there are obviously no deficiencies, and the other way is to make it so *complicated* that there are no obvious deficiencies. The first method is far more difficult."

Tony Hoare

Simple

slow

Correct

complex

fast

buggy

Photoshop Scalability: keeping It Simple

Clem Cole & Russell Williams

"As processors get more cores and grow in complexity, the need will only intensify for new tools and better programming primitives for hiding the complexity from developers and allowing them to code at higher levels of abstraction."

CACM oct. 2010

The PPCP Courseware is ...

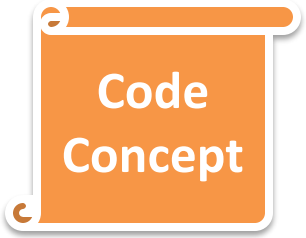
- **What**: 16 weeks (8 units) of material
  - Slides
  - Lecture notes
  - Quizzes, labs
  - Sample programs and applications
  - Tests and tools

- **Prerequisites:**
  - OO programming, data structures, operating systems

- **Dependencies**:
  - Visual Studio 2010 (includes .NET 4.0, C#, F#)

Concepts

- Amdahl's law
- Directed-acyclic graph (DAG) execution model
- Work and span



- C# lambdas
- Parallel.For

- Happens-before edges

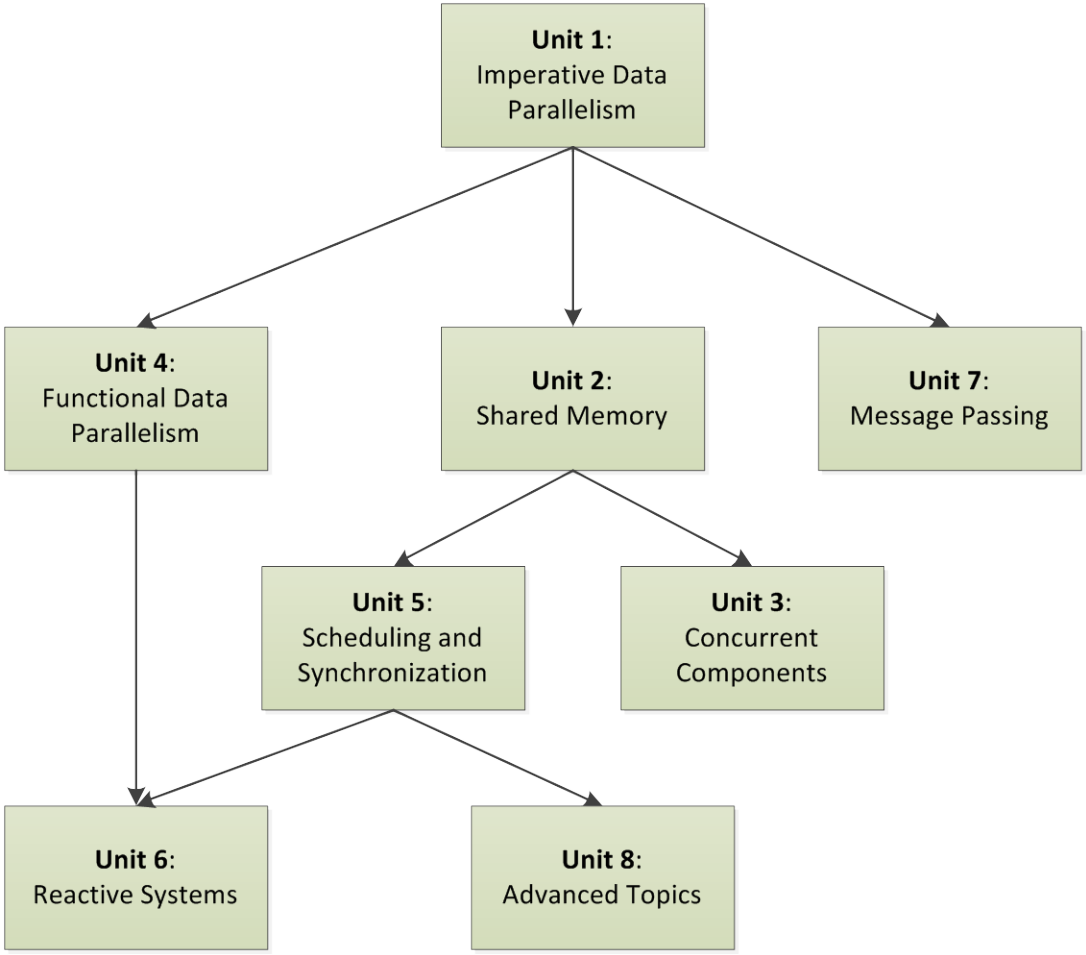
PPCP Units 1 – 4

- Unit 1: Imperative Data Parallel Programming
  - Data-intensive parallel programming (Parallel.For)
  - Concurrent Programming with Tasks
- Unit 2: Shared Memory
  - Data Races and Locks
  - Parallel Patterns
  - Cache Performance Issues
- Unit 3: Concurrent Components
  - Thread-Safety Concepts (Atomicity, Linearizability)
  - Modularity (Specification vs. Implementation)
- Unit 4: Functional Data Parallel Programming
  - Parallel Queries with PLINQ
  - Functional Parallel Programming with F#
  - GPU Programming with Accelerator

PPCP Units 5-8

- Unit 5: Scheduling and Synchronization
  - From {tasks, DAGs} to {threads, processors}
  - Work-stealing
- Unit 6: Interactive/Reactive Systems
  - Asynchronicity
  - Event-based programming
- Unit 7: Message Passing
  - Conventional MPI-style programming
- Unit 8: Advanced Topics
  - Memory models, lock-free data structures, optimistic concurrency

Eight Units: Flexibility



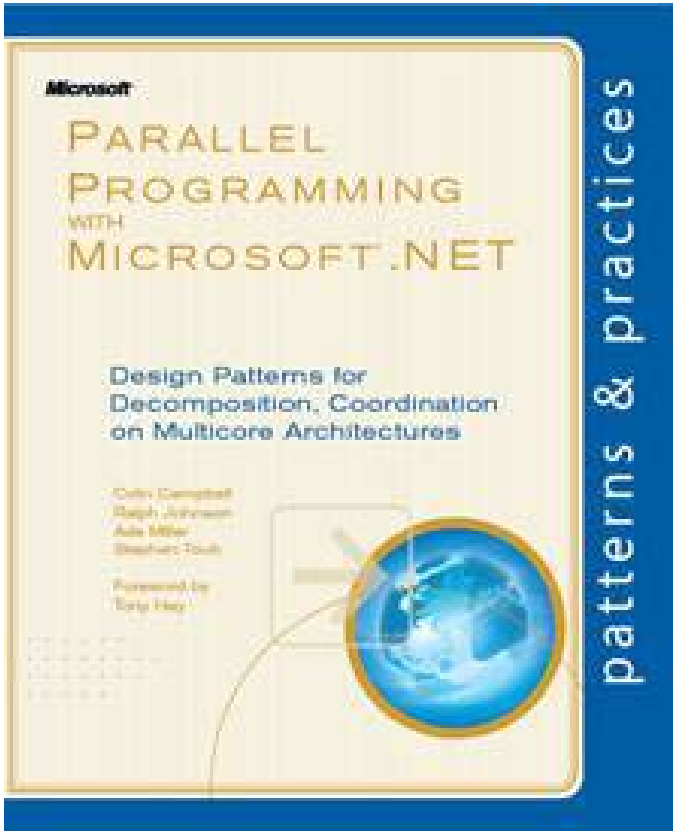
IDE, Libraries, Tools, Samples, Book

- **Visual Studio 2010**
  - C# and F# languages
  - **.NET 4:** Libraries for multi-core parallelism and concurrency
- **Other Libraries**
  - Accelerator
  - Code Contracts
  - Rx: Reactive Extensions for .NET

- **Alpaca**
  - A lovely parallelism and concurrency analyzer
  - Source code
- Code for all units, with Alpaca tests

- **Parallel Extensions Samples**
- **Book: Parallel Programming with Microsoft .NET**

Parallel Programming with Microsoft .NET



Design Patterns for Decomposition and Coordination on Multicore Architectures

Colin Campbell, *Ralph Johnson*, Ade Miller and Stephen Toub

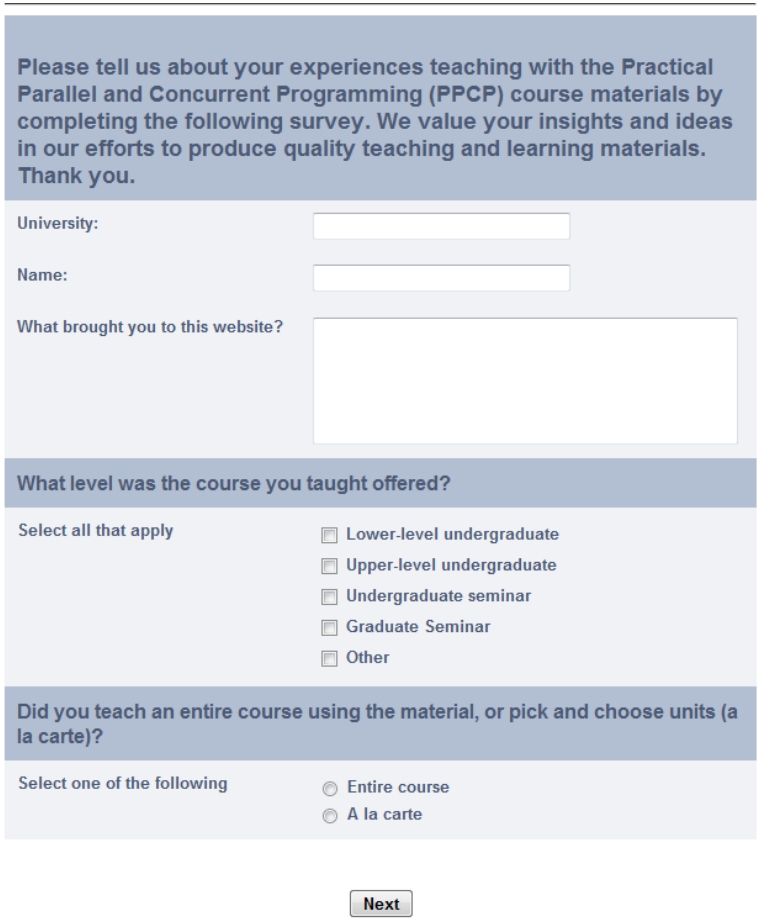
Teaching and Feedback

– Now:

THE UNIVERSITY OF UTAH

– Winter:

University of Washington
Computer Science & Engineering



Please tell us about your experiences teaching with the Practical Parallel and Concurrent Programming (PPCP) course materials by completing the following survey. We value your insights and ideas in our efforts to produce quality teaching and learning materials. Thank you.

University:

Name:

What brought you to this website?

What level was the course you taught offered?

Select all that apply

- Lower-level undergraduate
- Upper-level undergraduate
- Undergraduate seminar
- Graduate Seminar
- Other

Did you teach an entire course using the material, or pick and choose units (a la carte)?

Select one of the following

- Entire course
- A la carte

Next

Seven PPCP Principles

1. Start with abstractions
2. Later, look inside them
3. Emphasize correctness
4. No more matrix multiply
5. Don’t discount libraries
6. Tools support abstractions and concepts
7. Expose students to new research

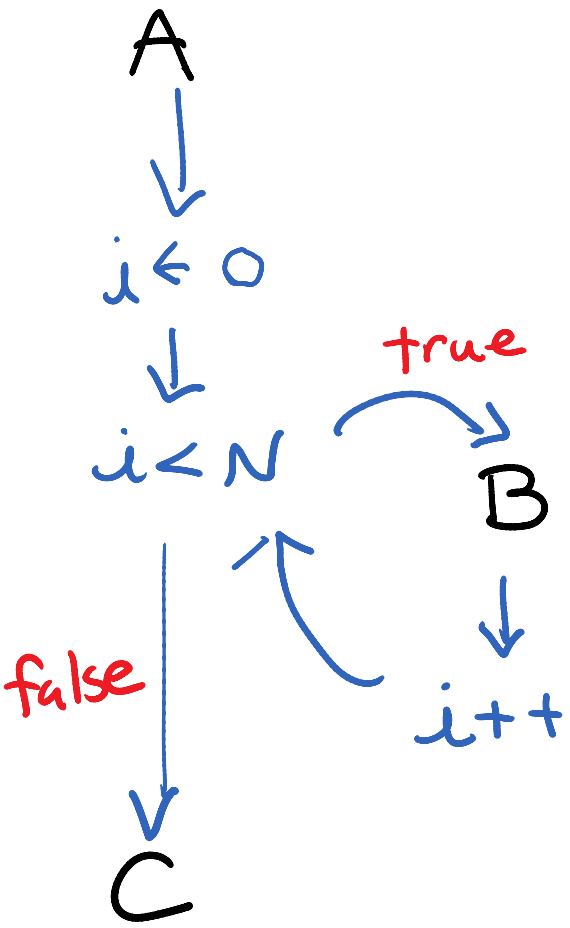
1. Start with Abstractions (Unit 1)

- Parallel speedup for data parallel computation
  - Highly motivating
  - Independent loops

- Emphasize productivity
  - high-level abstraction, not low-level primitives
  - coarse-grained task parallelism, not threads
  - array algorithms, not CUDA

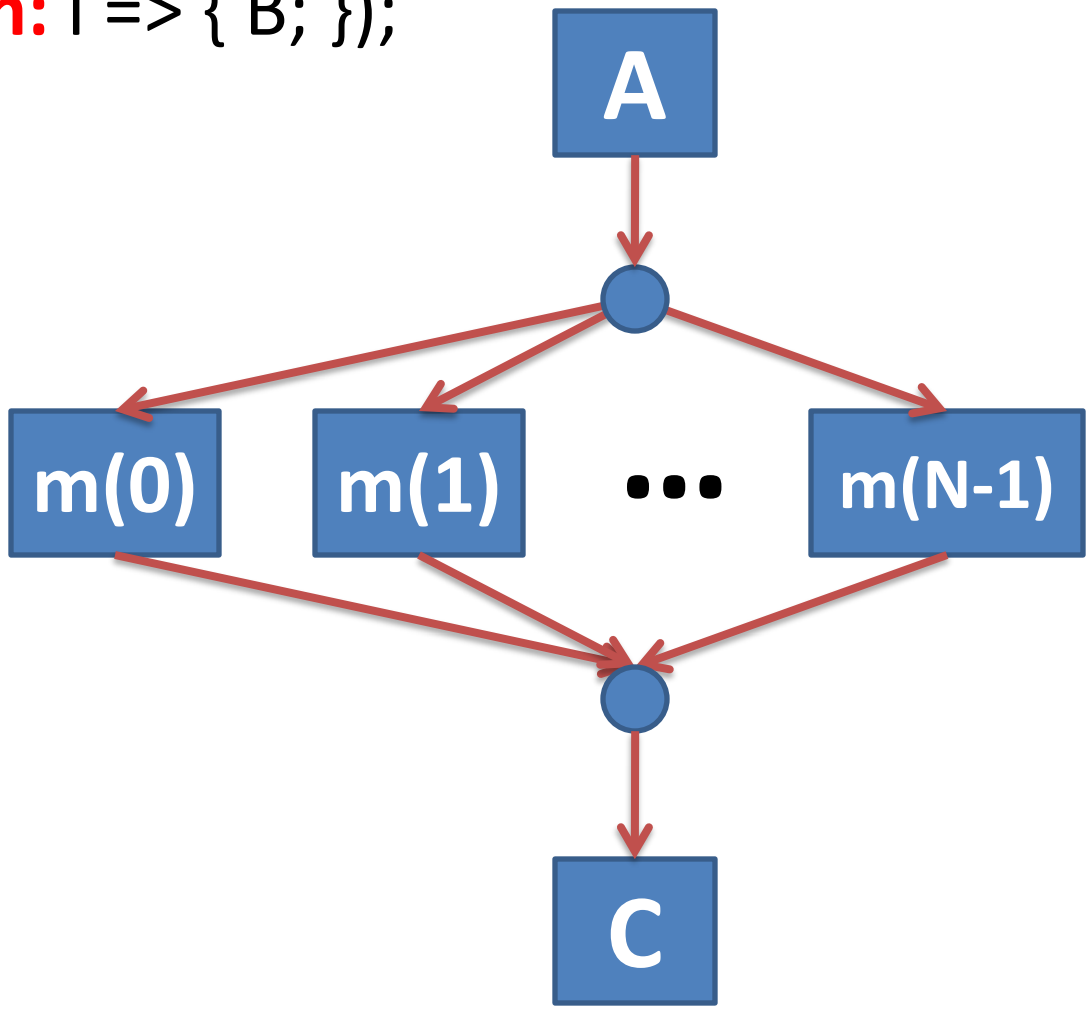
Sequential Loop

```
A;
for(int i=0; i<N; i++) {
    B
}
C;
```

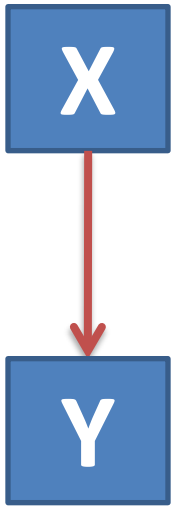


Parallel.For

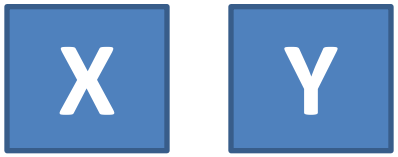
```
A;
Parallel.For(0, N, m: i => { B; });
C;
```



Parallel DAG Edges (Happens-before) Constrain Execution



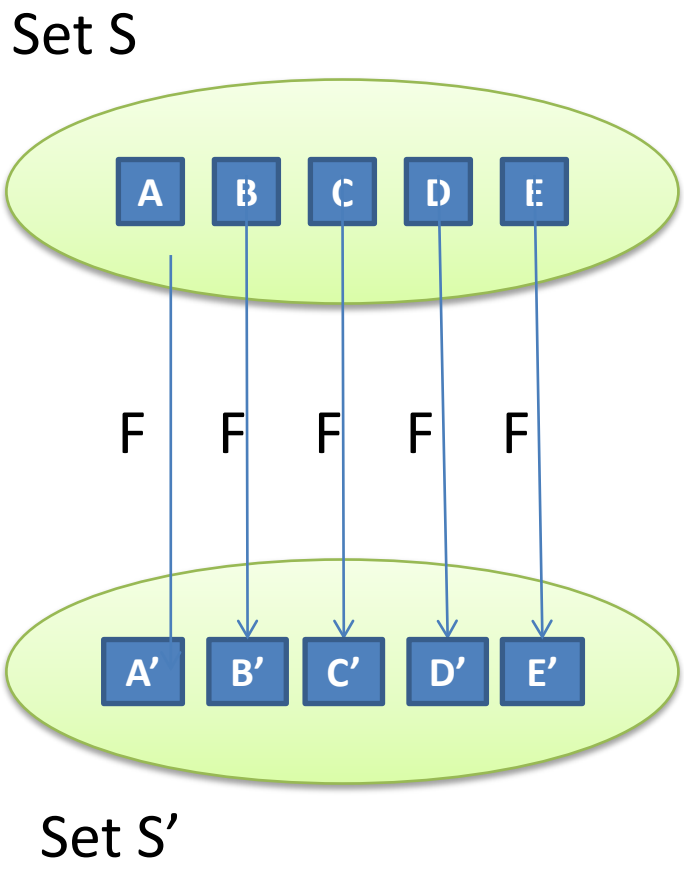
- Implementation guarantees
  - X completes execution before Y starts execution
  - All effects of X are visible to Y



- No directed path between X,Y, no guarantees
  - X,Y could happen in parallel, or
  - X could happen before Y, or
  - Y could happen before X

Sets and Parallelism

- A very nice match!
- Why?
  - sets abstract away from order of elements
  - each element is an independent entity
  - each element can be processed independently (by function F)



Functional/ Declarative Paradigms

- LINQ
  - Language Integrated Queries
- MapReduce
  - List.map
  - List.fold

2. Later, Look Inside Abstractions

- To understand performance
  - load balancing, work stealing (Unit 5)
  - data locality, false sharing, lock overhead (Unit 2)
  - memory bottlenecks

- To understand how to build higher-level abstractions
  - Example: threads, building a thread-safe buffer (Unit 5)

Understanding Parallel.For and PLINQ

- The platform
  - Assigns work to cores efficiently
  - Dynamically partitions
  - Balances load efficiently
  - Handles exceptions
- And much more…

3. Emphasize Correctness

- Multicore programming is hard
  - New bug types
  - Unpredictable bugs (not obviously deficient)
  - Severe bugs

"the single longest lived bug... in Photoshop ... hid out in [asychronous I/O] for about 10 years"

Example: Data Race Free Discipline

- **Data-Race-Free (DRF) Discipline**

  means we write programs that have NO data races (not even “benign” ones).

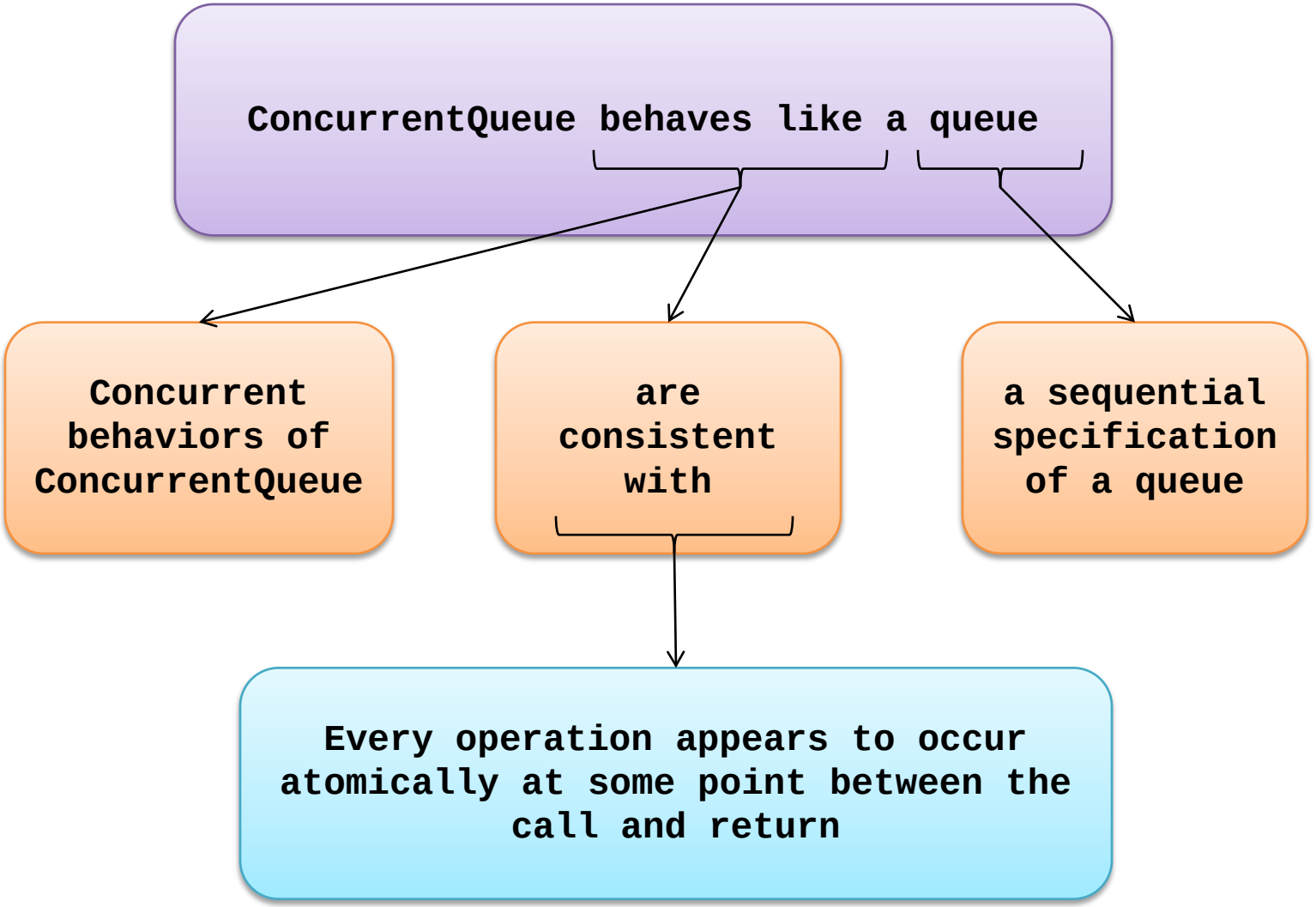
- Already “best practice” for many, but not all programmers.

remember buffer overflows?

Example: Atomicity

A statement sequence S is *atomic* if
S's effects appear to other threads as if
S executed without interruption

Example: Linearizability [Herlihy & Wing '90]

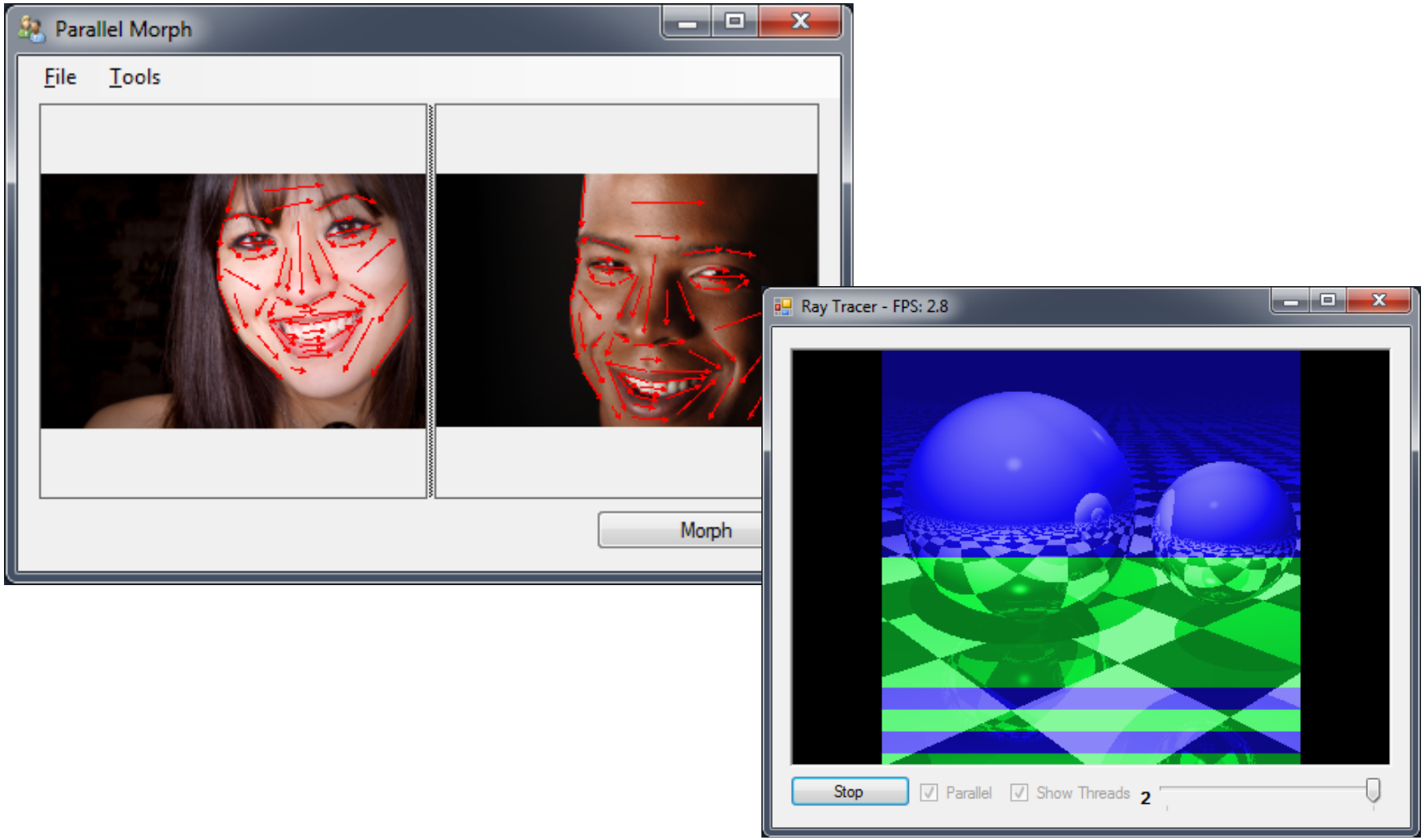


Determinism Contract

- Allows you to check parallel correctness
- Without having to specify the sequential contract
  - e.g. The output of the function does not depend on thread interleavings for a given input

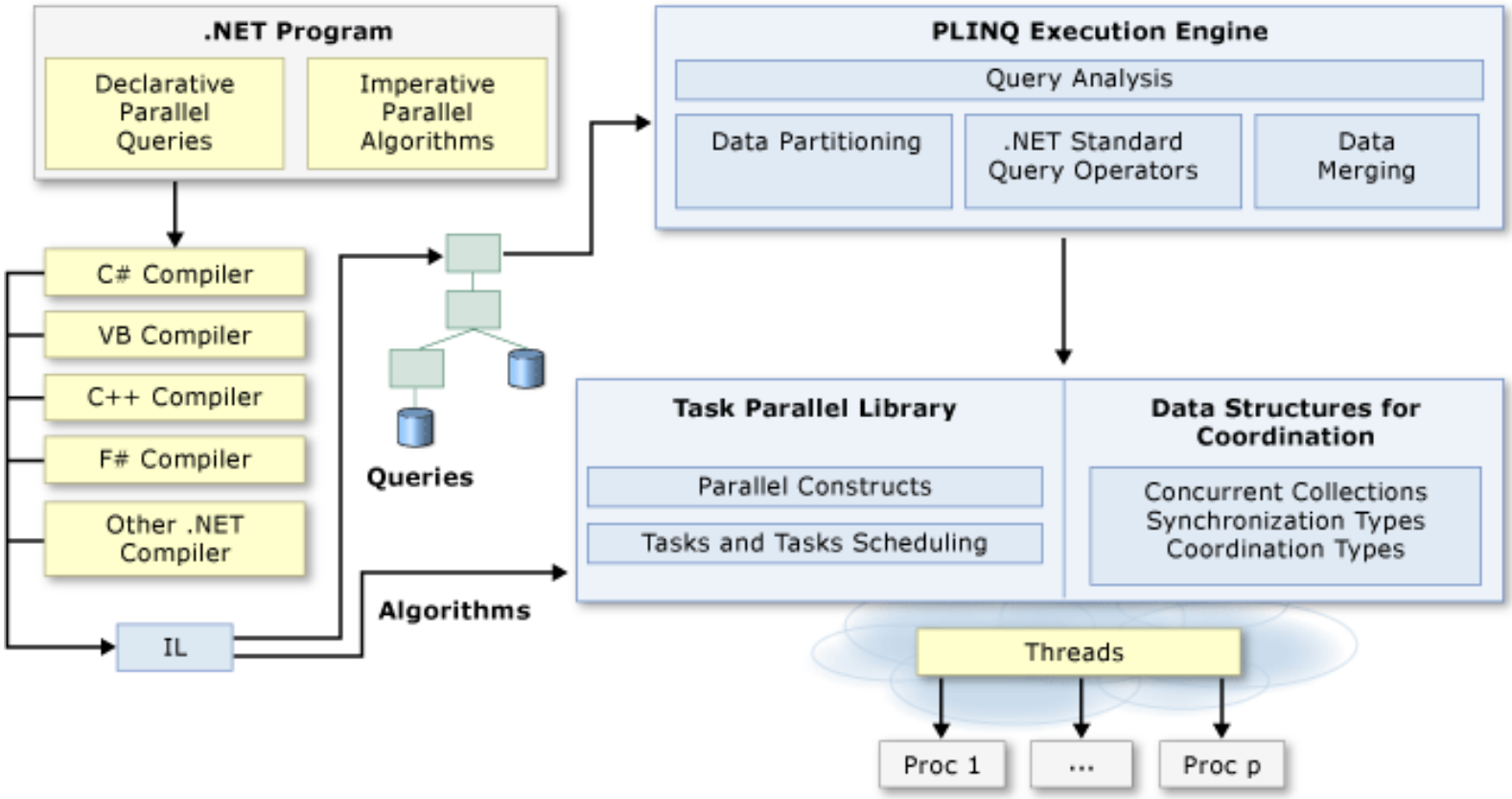
```
int ComputeSum ( IEnumerable<int> input)
{
  Contract.Requires ( input != null);

  IsDeterministic (
           Contract.Result<int>(), input );

   //implementation
}
```

4. No More Matrix Multiply: Parallel Extensions Samples



5. Don’t Discount Libraries

- Breadth, but embedded in .NET



Photoshop Scalability: keeping It Simple

Clem Cole & Russell Williams

"The more we can hide under more library layers, the better off we are."

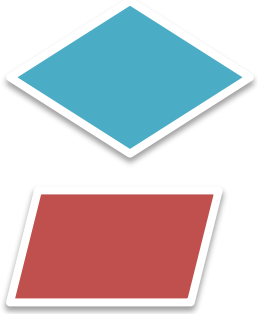
"...every parallel library implementation of common algorithms ... is greatly appreciated."

CACM oct. 2010

6. Tool-based Approach to Performance and Correctness

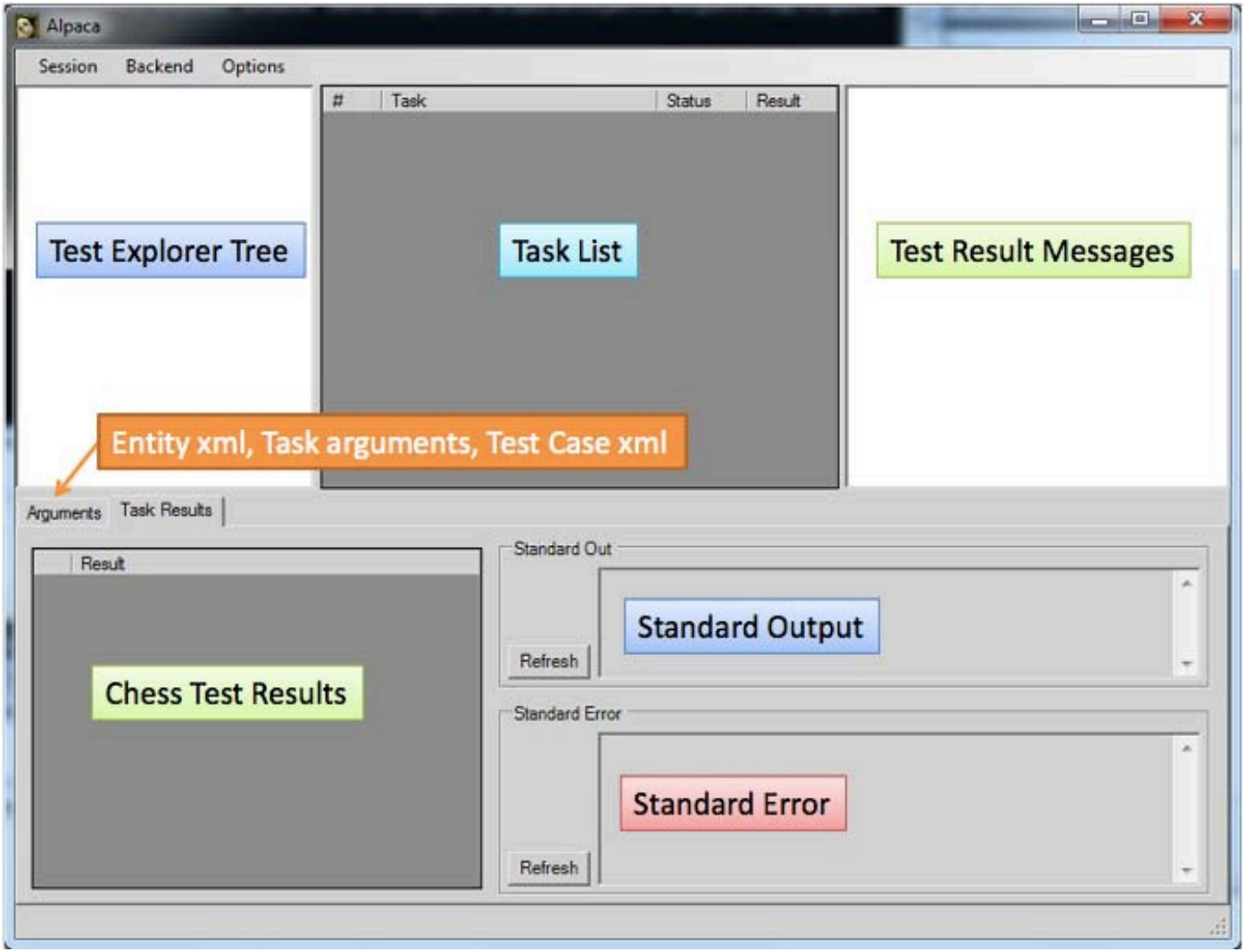
- Building understanding of concepts through experimentation
  - Data race, linearizability, deadlock detection, …
- Emphasize unit testing, performance testing
  - Alpaca tool
  - TaskoMeter

Alpaca: A lovely parallelism and concurrency analyzer

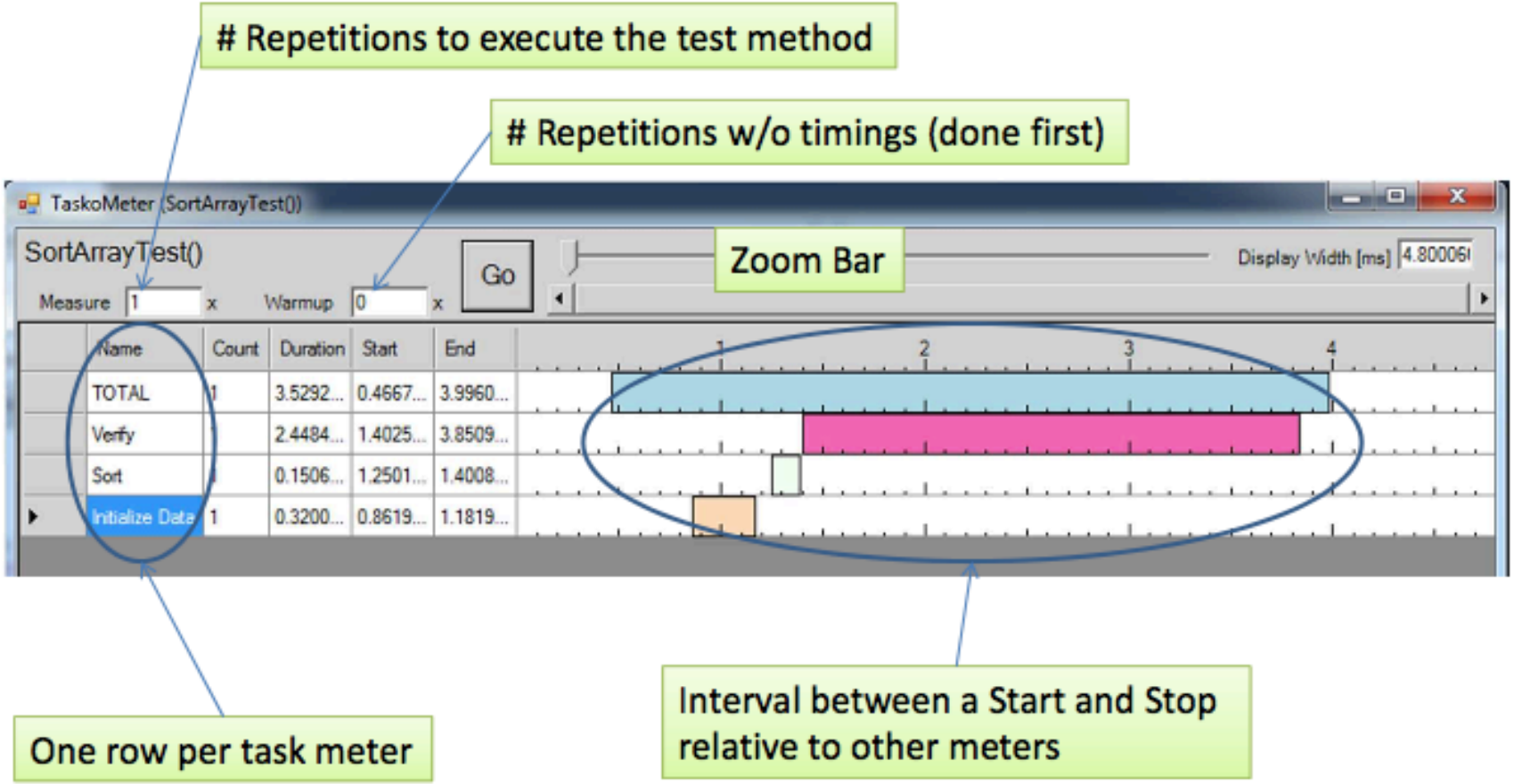


- Atttribute-based testing, for performance and correctness concepts
- `[UnitTestMethod]`
  - simply run this method normally, and report failed assertions or uncaught exceptions.
- `[DataRaceTestMethod]`
  - Run a few schedules (using CHESS tool) and detect data races.
- `[ScheduleTestMethod]`
  - Run all possible schedules of this method (with at most two preemptions) using the CHESS tool.
- `[PerformanceTestMethod]`
  - Like UnitTestMethod, but collect & graphically display execution timeline (showing intervals of interest)

Alpaca (A lovely parallelism and concurrency analyzer)



TaskoMeter


Repetitions to execute the test method
Repetitions w/o timings (done first)
TaskoMeter (SortArrayTest())
SortArrayTest()
Measure 1 x
Warmup 0 x
Go
Zoom Bar
Display Width [ms] 4.80006
Name
Count
Duration
Start
End
TOTAL 1 3.5292... 0.4667... 3.9960...
Verify 2.4484... 1.4025... 3.8509...
Sort 0.1506... 1.2501... 1.4008...
Initialize Data 1 0.3200... 0.8619... 1.1819...
1
2
3
4
One row per task meter
Interval between a Start and Stop relative to other meters

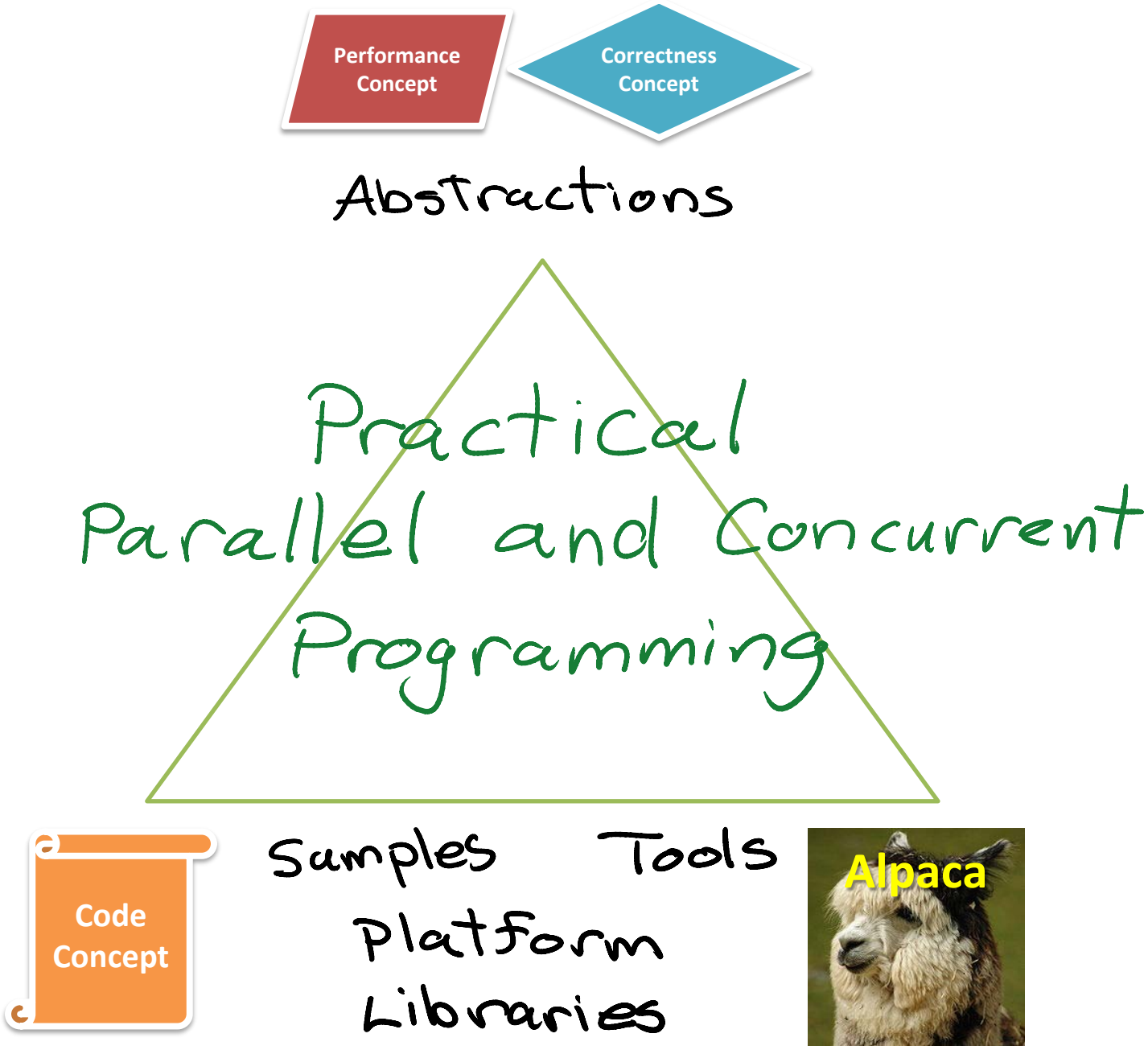
7. Expose Students to New Research

- CHESS
  - Deterministic scheduling
  - Checking concurrency correctness
- Code Contracts
  - Lightweight specifications
- Accelerator
  - Array/vector parallelism (GPU)
- Reactive Extensions (Rx)
  - Asynchronous and event-based computation
- Revisions
  - Deterministic parallelism

Alpaca/CHESS

http://research.microsoft.com/chess/

- ***Source code release***
  - chesstool.codeplex.com
- ***Preemption bounding [PLDI07]***
  - speed search for bugs
  - simple counterexamples
- ***Fair stateless exploration [PLDI08]***
  - scales to large programs
- ***Architecture [OSDI08]***
  - Tasks and SyncVars
  - API wrappers
- ***Store buffer simulation [CAV08]***
- ***Preemption sealing [TACAS10]***
  - orthogonal to preemption bounding
  - where (not) to search for bugs
- ***Best-first search [PPoPP10]***
- ***Automatic linearizability checking [PLDI10]***
- ***More features***
  - Data race detection
  - Partial order reduction
  - More monitors…


Performance Concept
Correctness Concept
Abstractions
Practical Parallel and Concurrent Programming
Code Concept
Samples
Tools
Platform
Libraries
Alpaca

Austin, USA
January 26-28

38th ACM SIGACT-SIGPLAN Symposium on
Principles of Programming Languages

Thank
you!