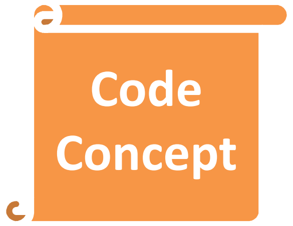
Imperative Data Parallelism (Correctness)

Unit 1.b

Acknowledgments

- Authored by
  - Thomas Ball, MSR Redmond

Concepts



- Parallel.Invoke
- Parallel.ForEach

- Schedules and determinism
- Assertions/Invariants
- Unit Testing

tjb2

Parallel.Invoke

```
static void Invoke(params Action[] actions);
```

```
int x = 0;
Parallel.Invoke(
   () => { x=1; },
   () => { x=2; }
   );
Console.WriteLine(“x={0}”, x);
```

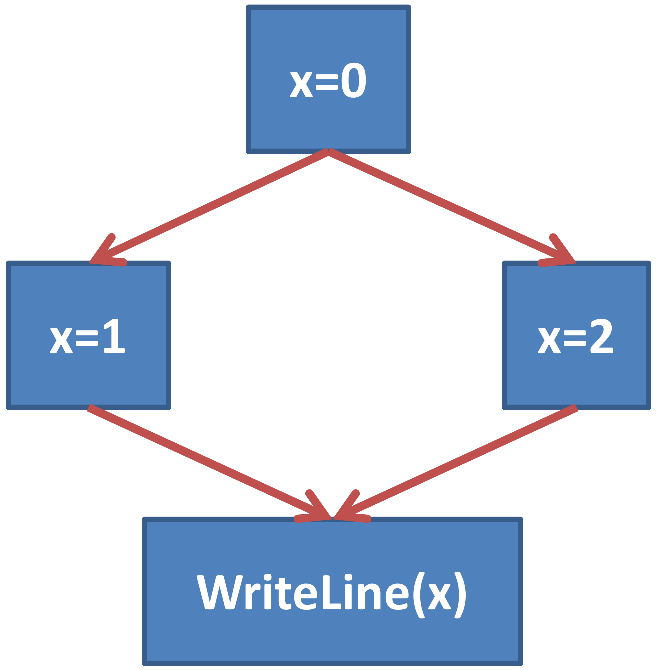
ParallelSamples.cs

Slide 4

tjb2 Maybe motivate with a prior slide with a more realistic example where Parallel.For is not quite what we want. i.e. what if we have two specific things we want to do in parallel?

Then have this simple example where stuff breaks.
Tom Ball, 8/14/2010

Parallel DAG and Happens-before Edges


$$\begin{tikzcd}
 & x=0 \arrow[dl] \arrow[dr] & \\
x=1 \arrow[dr] & & x=2 \arrow[dl] \\
 & \text{WriteLine(x)} &
\end{tikzcd}$$

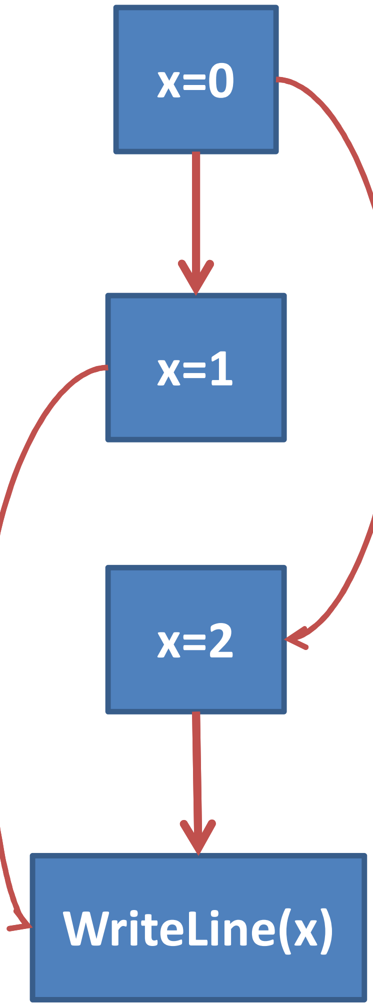
Schedule, Informally

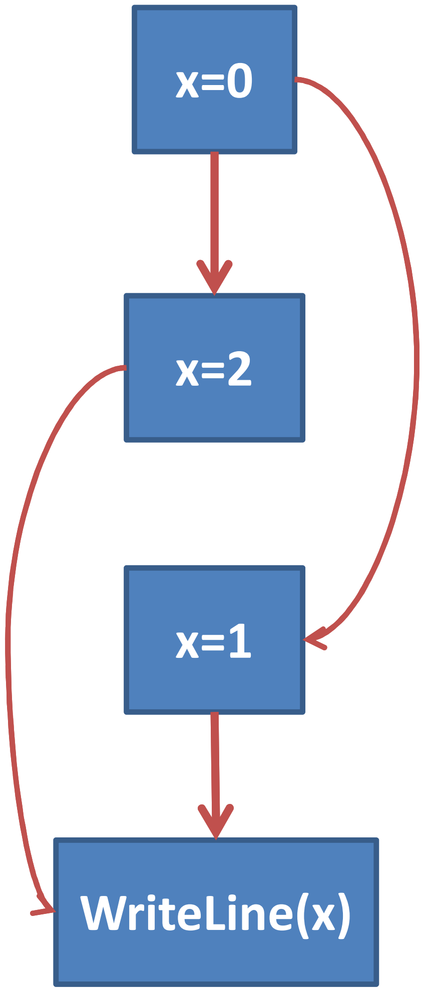
A topological sort (serialization) of the nodes in a parallel DAG

-

A <u>sequential</u> ordering of the nodes that respects the happens-before edges

Different schedules, different outputs





```
x = 2
```

```
x = 1
```

CS9

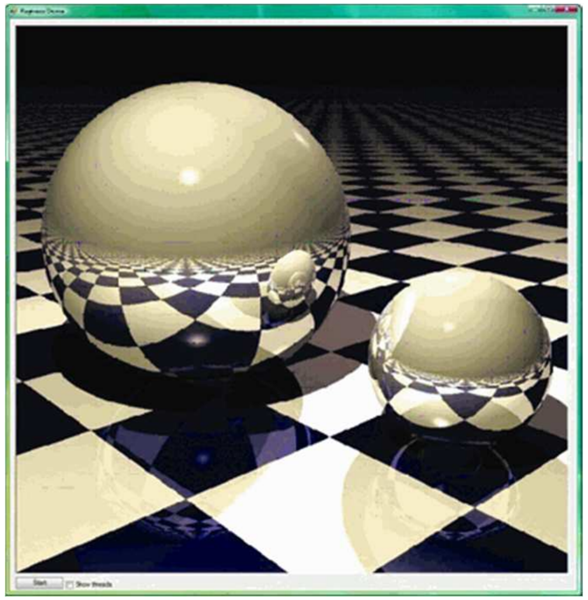
Determinism

- For the same initial state,
  observe the same final state,
  regardless of the schedule

- Determinism desirable for most data-parallel problems

Slide 8

CS9 How is determinism reflected on a happens-before graph?
Caitlin Sadowski, 7/8/2010



Parallel Ray Tracing: Deterministic

```
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));
    }
  });
}
```

Unit Testing

- The goal of *unit testing* is to isolate each part of the program and show that the individual parts are correct

- A unit test is
  - a closed program that
  - sets up conditions to run
  - a program unit and
  - check the results

System vs. Unit Testing

- System Testing
  - Test entire application
  - Needed to find integration errors
  - Does not put much stress on individual components

- Unit Testing
  - Better coverage, but more work
  - <u>Necessity</u> for libraries and frameworks
  - <u>Good idea </u>for tricky parallel/concurrent components

Checking Determinism

- How can we test the correctness of the parallel Ray Trace application?
- Create unit test to compare
  - the parallel version
  - the sequential version
- Should we be satisfied with such tests?
- Do unit tests work well for parallel programs?

RayTracerTest.cs

IEnumerable and Parallel.ForEach

- Parallel.ForEach is not limited to integer ranges and arrays!
- Generic enumerations
  - `IEnumerable<T>`
  - Lists, sets, maps, dictionaries, …

CS12

Parallel.ForEach

```
public static ParallelLoopResult
    ForEach<TSource>(
        IEnumerable<TSource> source,
        Action<TSource> body
    );
```

Slide 14

CS12 Could we add a graphic?
Caitlin Sadowski, 7/8/2010

Speedup Demo: Antisocial Robots

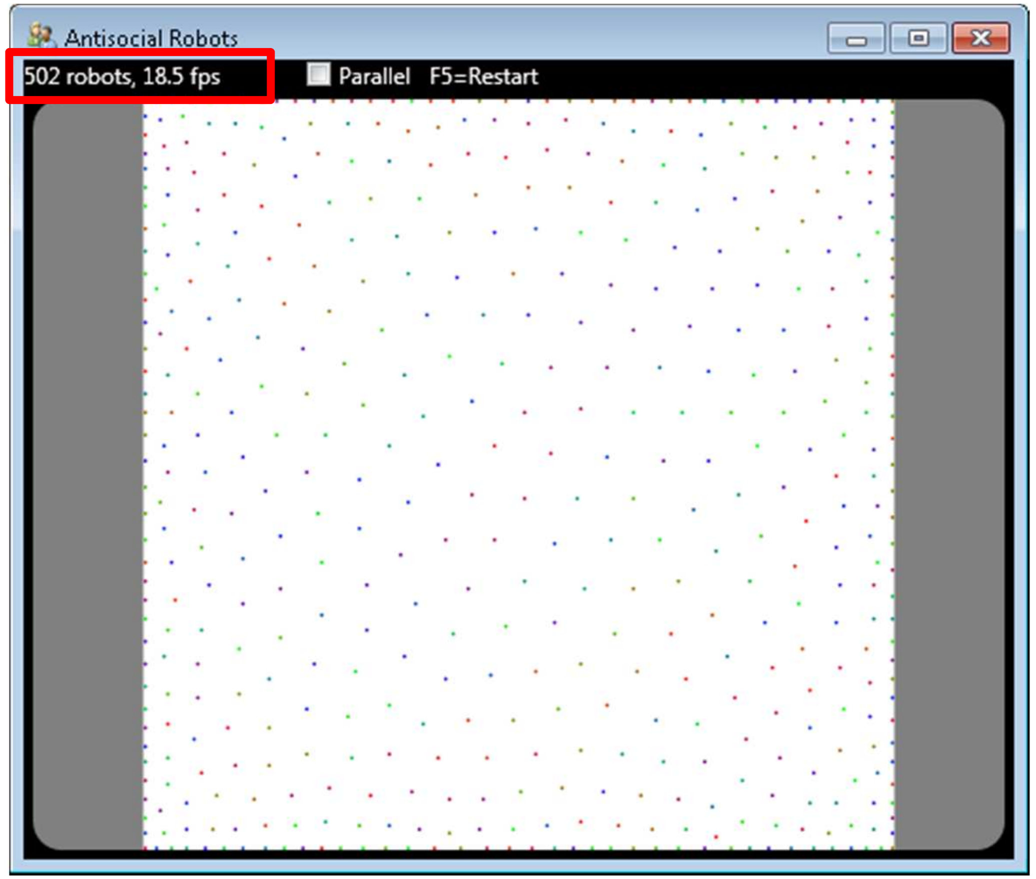
fps

=

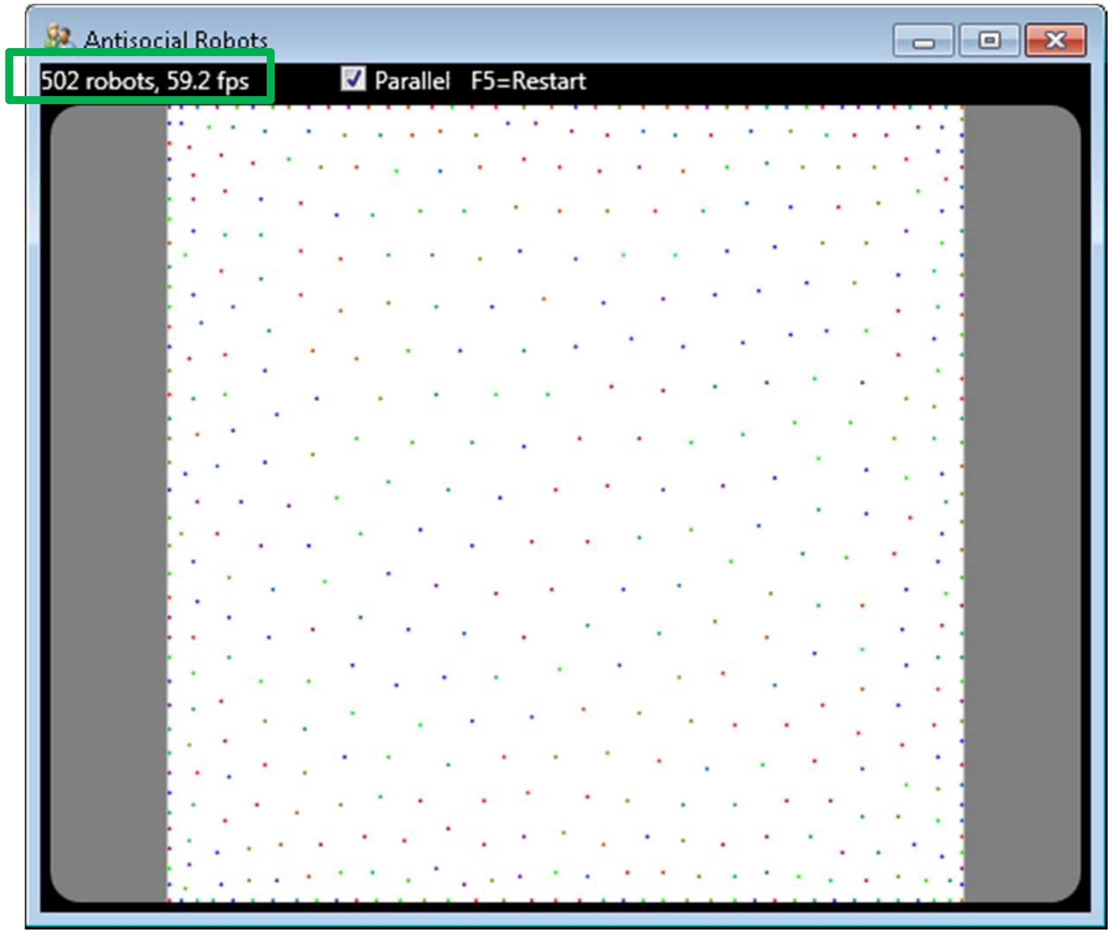
frames

per

second



Speedup: Over 3x on a 4-core!



The Difference in the Code?

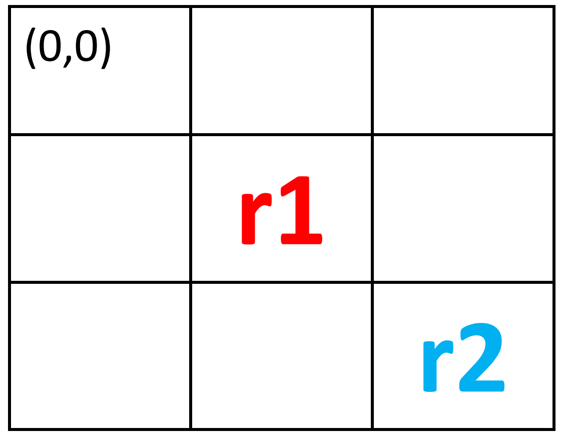
```
void PerformSimulationStep()
{
    if (naiveparallel.IsChecked.Value)
    {
        _robotSim.ParallelStep();
    }
    else
    {
        _robotSim.SequentialStep();
    }
    . . .
```

```
public void SequentialStep()
{
    foreach (Robot robot in _robots)
        SimulateOneStep(robot);
}
```

```
public void ParallelStep()
{
    Parallel.ForEach(_robots, r =>
        SimulateOneStep(r));
}
```

Key Data Structures

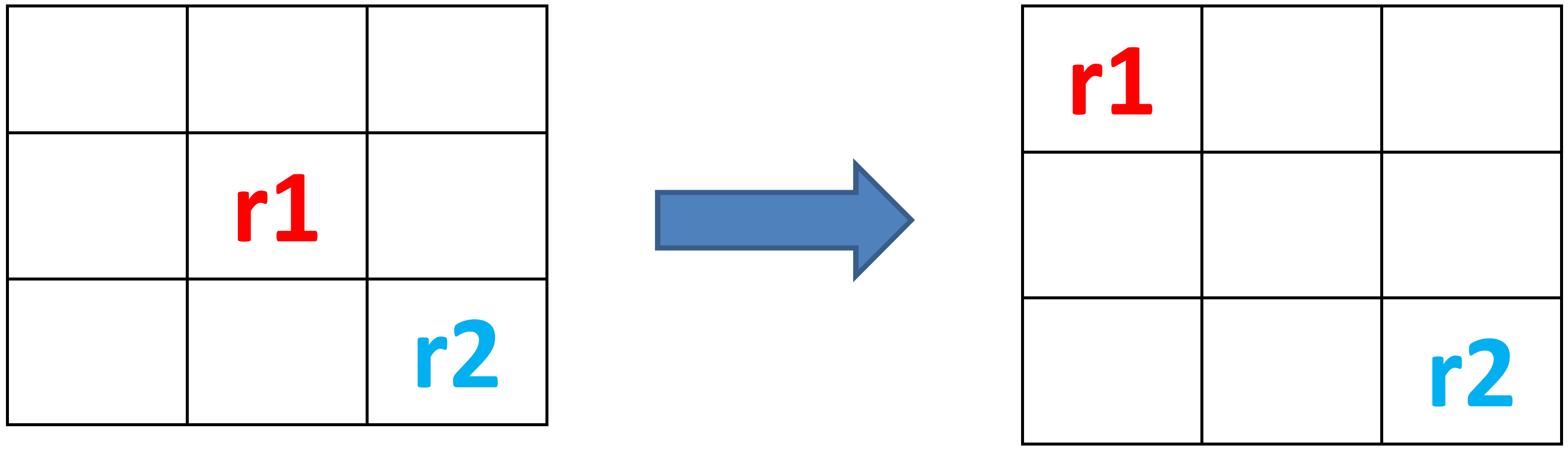
```
struct RoomPoint {
   public int X;
   public int Y;
}

class Robot {
   public RoomPoint Location;
}

List<Robot> _robots;
Robot[][] _roomCells;
```

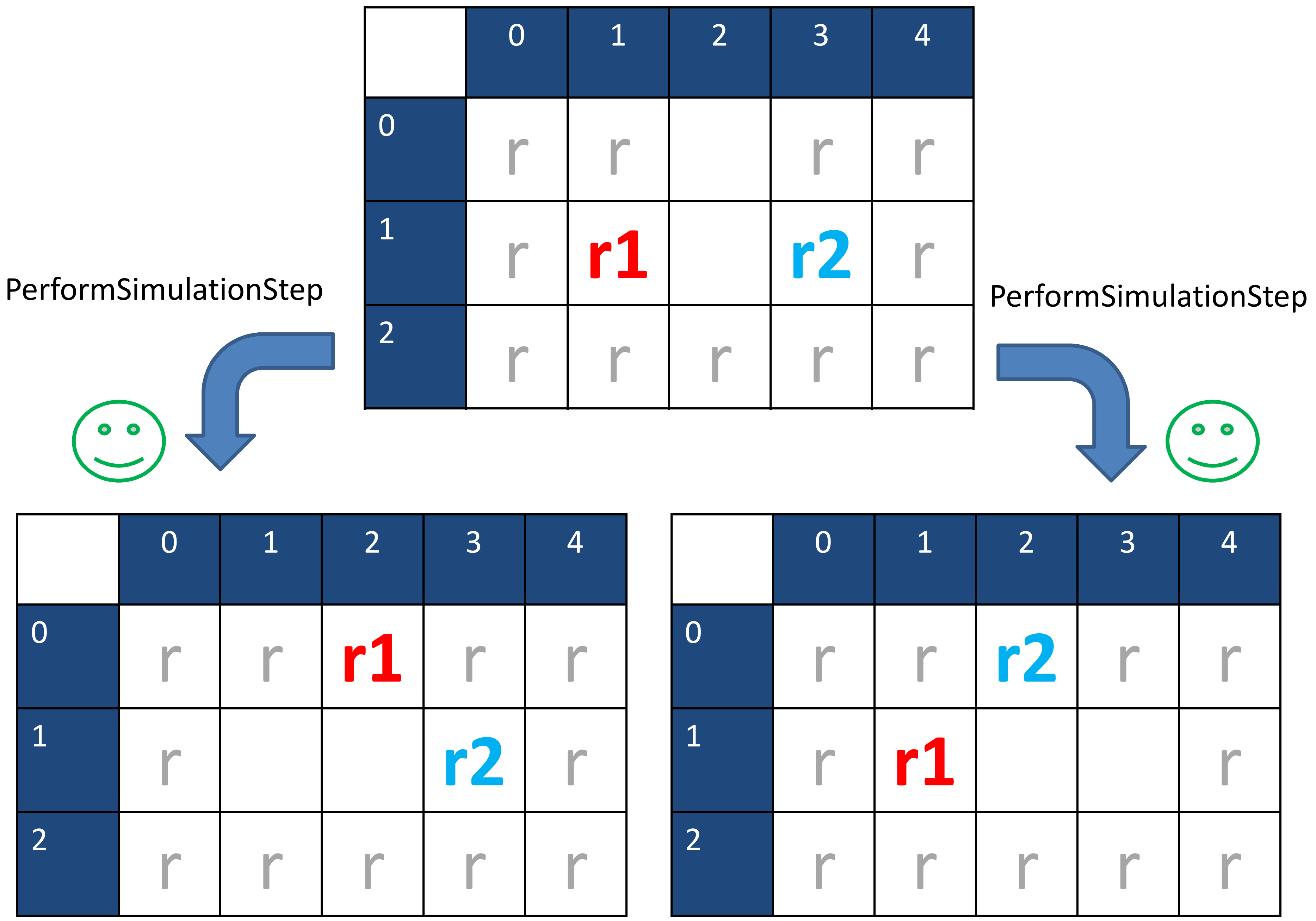


`_roomCells;`

SimulateOneStep(Robot r1)

- Determine new cell for **r1**
- Move **r1** to new cell, if not already occupied





	0	1	2	3	4
0	r	r		r	r
1	r	r1		r2	r
2	r	r	r	r	r

	0	1	2	3	4
0	r	r	r1	r	r
1	r			r2	r
2	r	r	r	r	r

	0	1	2	3	4
0	r	r	r2	r	r
1	r	r1			r
2	r	r	r	r	r

	0	1	2	3	4
0	r	r		r	r
1	r	r1		r2	r
2	r	r	r	r	r

	0	1	2	3	4
0	r	r	r2	r	r
1	r				r
2	r	r	r	r	r

Pigeonhole Principle

- “Two robots can’t occupy the same cell”

```
foreach (var in _robots)
  Debug.Assert(_roomCells[r.Location.X,r.Location.Y] == r,
               “Can’t have two robots in the same cell!”);
```

- If it is true before execution of `PerformSimulationStep` then it should be true afterward, regardless of sequential/parallel implementation

Assert Statement

- Assert(e)
  - e a Boolean expression (*state predicate*)
  - e should always evaluate true when statement executes; otherwise program has an error

- Helpful assertions have messages:
  - Assert(balance>=0,
                "account balance should be non-negative")

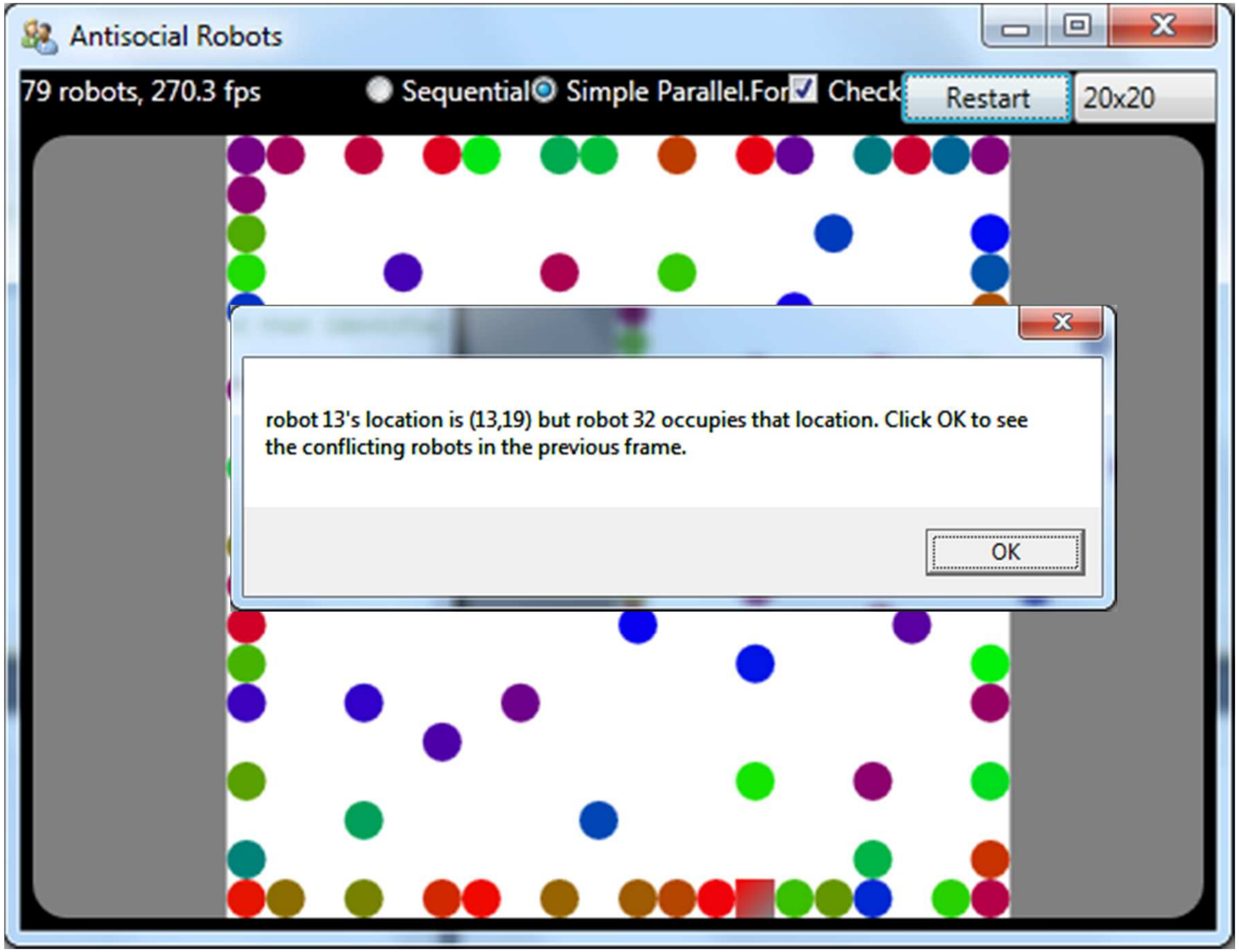
Invariant

- State predicate e is **invariant** to program fragment S provided that
  - **If** predicate e is true before execution of S then
  - **Then** predicate e is true after execution of S
- So,
  - State predicate
    - “Two robots can’t occupy the same cell”
  - Is invariant to
    - PerformSimulationStep

AntisocialRobots.csproj

1. Antisocial Robots has a Bug

2. It’s Hard to Expose Concurrency Bugs!



79 robots, 270.3 fps

Sequential [✓ Simple Parallel.For] [✓ Check]

robot 13's location is (13,19) but robot 32 occupies that location. Click OK to see the conflicting robots in the previous frame.

Run Alpaca [UnitTestMethod]

to get more reliable

reproduction of bug

RobotSimulationInterferenceTest.cs

High-level Problem

- SimulateOneStep(r1) and SimulateOneStep(r2) interfere with one another when
  - r1 wants to move to cell (X,Y), and
  - r2 wants to move to cell (X,Y)
- Sequential version: invariant is maintained
- Parallel version: invariant breaks!

Two Bugs in Three Lines: Updating Robot r's Location

```
SimulateOneStep(Robot r) {

 RoomPoint ptR;
 // compute new location of Robot r into ptR
 ...

 // update robot location
 if (((ptR.X != r.Location.X) || (ptR.Y != r.Location.Y))
 &&  (_roomCells[ptR.X, ptR.Y] == null))
 {
      _roomCells[r.Location.X, r.Location.Y] = null;
      _roomCells[ptR.X, ptR.Y] = r;
      r.Location = new RoomPoint(ptR.X, ptR.Y);
 }
```

Order of Statements Leading to Invariant Failure

	0	1	2	3	4
0	r	r		r	r
1	r	r1		r2	r
2	r	r	r	r	r

Time

SimulateOneStep(r1) **SimulateOneStep(r2)**

```
if (_roomCells[2,0] == null)
 _roomCells[1,1] = null;

                               if (_roomCells[2,0] == null)
                                _roomCells[3,1] = null;
 _roomCells[2,0] = r1;
 r1.Location = (2,0);

                                _roomCells[2,0] = r2;
                                r2.Location = (2,0);
```

Order of Statements Leading to Invariant Failure

	0	1	2	3	4
0	r	r	r2	r	r
1	r				r
2	r	r	r	r	r

SimulateOneStep(r1) **SimulateOneStep(r2)**

```
if (_roomCells[2,0] == null)
 _roomCells[1,1] = null;
                                if (_roomCells[2,0] == null)
                                 _roomCells[3,1] = null;

 _roomCells[2,0] = r1;
 r1.Location = (2,0);

                                 _roomCells[2,0] = r2;
                                 r2.Location = (2,0);
```

Question: What is the Second Bug?

- Think about the `struct RoomPoint`
- Come up with a scenario
  - Ordering of statements leading to invariant violation

Parallel.For/ForEach and Correctness

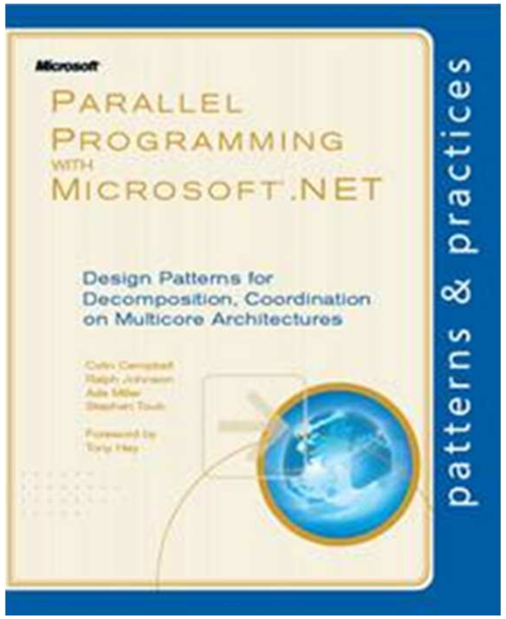
- No interference between delegates on different loop iterations
  - Avoid Writing to Shared Memory Locations
  - Avoid Calls to Non-Thread-Safe Methods

- No interference: implies determinism?

- Only the GUI thread can access GUI state
  - Don’t execute Parallel.For on the GUI thread

Parallel Programming with Microsoft .NET

- Chapter 2 (Parallel Loops) Parallel.For/ForEach
- Appendix B (Debugging and Profiling Parallel Applications)


Microsoft
Parallel Programming with Microsoft .NET
Design Patterns for Decomposition, Coordination on Multicore Architectures
patterns & practices