Parallelism
Synchronous mechanism

Peter Levinsky IT, Roskilde

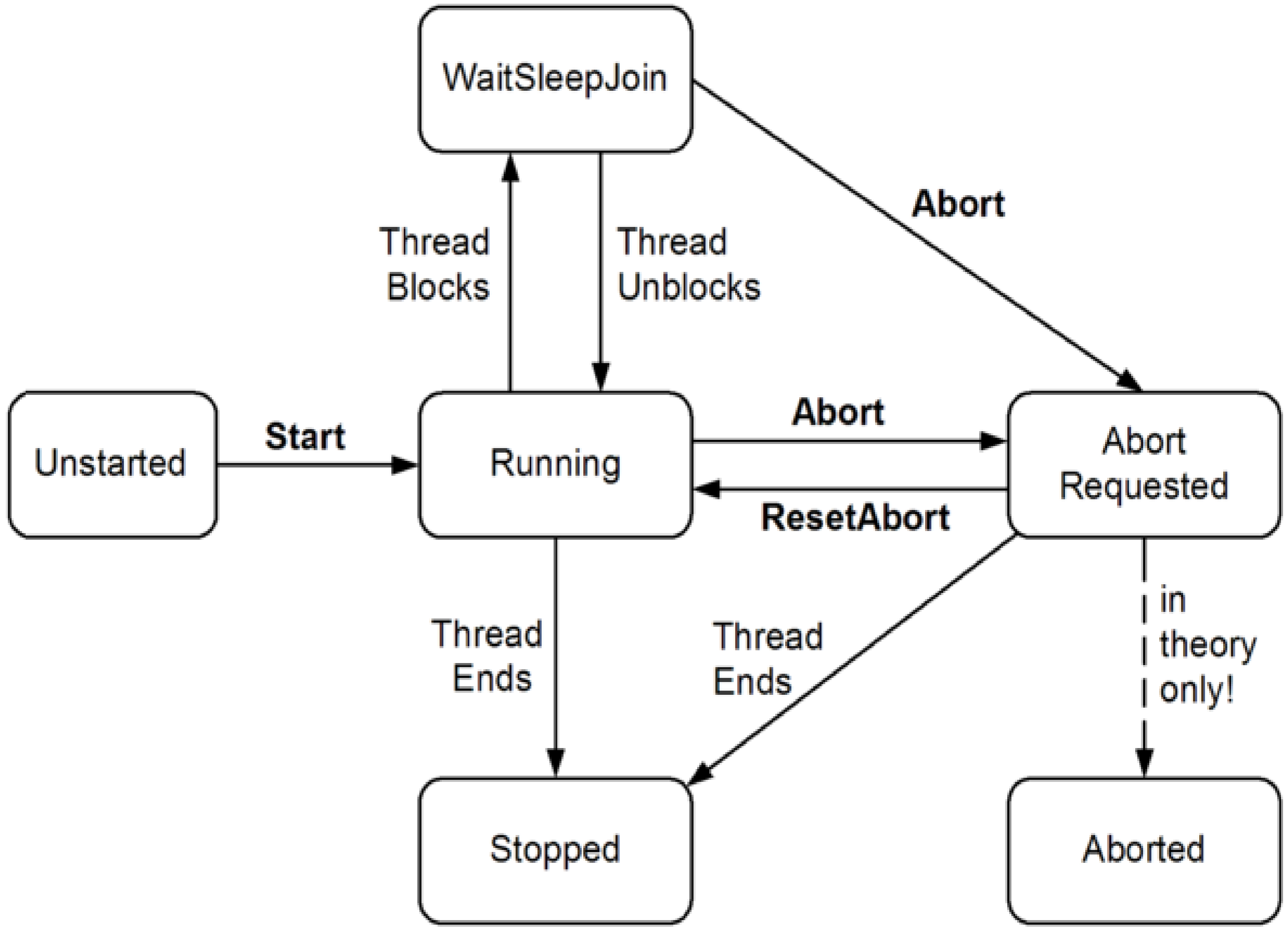
03.09.2024

Time consuming operations

Two categories

- CPU-bound operations
- I/O-bound operations

Thread Life cycle



Thread in C#

```
Thread t = new Thread (-- delegate Method --);
t.Start();
…
t.Join(); // wait here until t is completed
```

? Delegate Method

Delegate in C#

- Like you have references to objects
- **A delegate is a reference to a method**

<u>How to define:</u>
public delegate <<returnType>> MethodName(<<parameter list>>); // MethodName often xxxMethodType

<u>How to declare:</u>
xxxMethodType methodReferenceName;

<u>How to instantiate:</u>
methodReferenceName = 1) NameOfMethod
2) Lambda expression

<u>How to use:</u>
ReturnType var-name = methodReferenceName(parameter values);

Delegate build-in method types in C#

C# has a lot of build-in method types

- **Action**: a set of methods with no return types (i.e. void)

  ex. Action<int, string> is equal to `public delegate void XX(int i, String str)`

- **Func**: a set of methods with return types (the LAST type is the return type)

  ex. Func<int> is equal to `public delegate int XX()`
  ex. Func<int,string,bool> is equal to `public delegate bool XX(int i, String str)`

- **Predicate**: a set of methods with bool return type and only one parameter

  ex. Predicate<string> is equal to `public delegate bool XX(String str)`

Zealand

Thread in C# - exceuting

```
class ThreadTest
{
  static bool done=false; // Static fields are shared between all threads

  static void Main()
  {
    new Thread (Go).Start();
    Go();
  }

  static void Go()
  {
    if (!done) { done = true; Console.WriteLine ("Done"); }
  }
}
```

Parallelism in C#

Levels of parallelism:

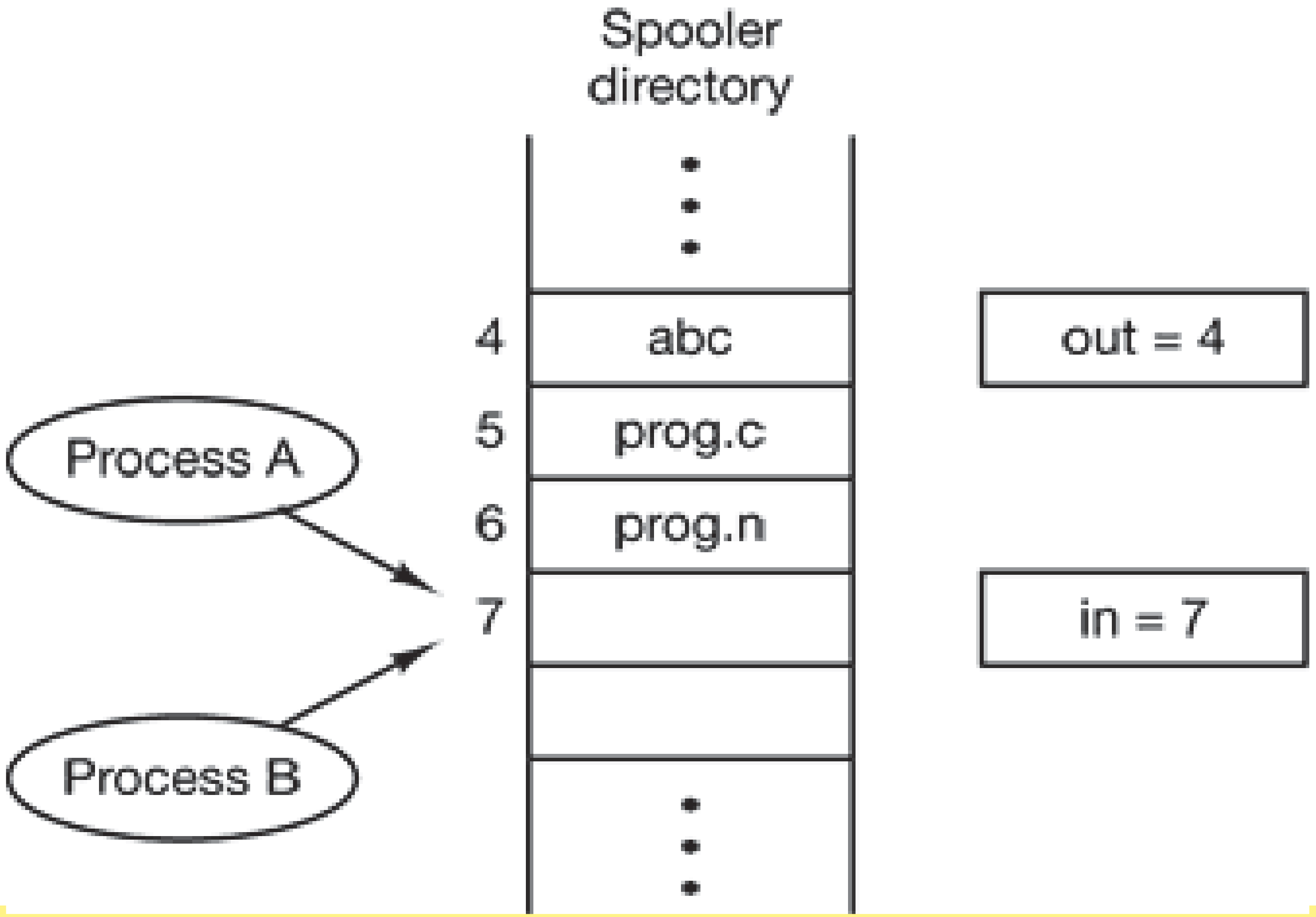
- Thread -- basic structure for parallelism (in most programming languages)
- Task -- C# smooth variant i.e. Task.Run(---)
- Parallel.Invoke -- Can start several threads (blocked until after all thread is completed)
- Parallel.For / Foreach -- Can start several threads in a loop (blocked until after all thread is completed)
- Plinq -- can execute a Linq expression in parallel

Demo

Excerises C#Exercises Prog.4.1, 4.2

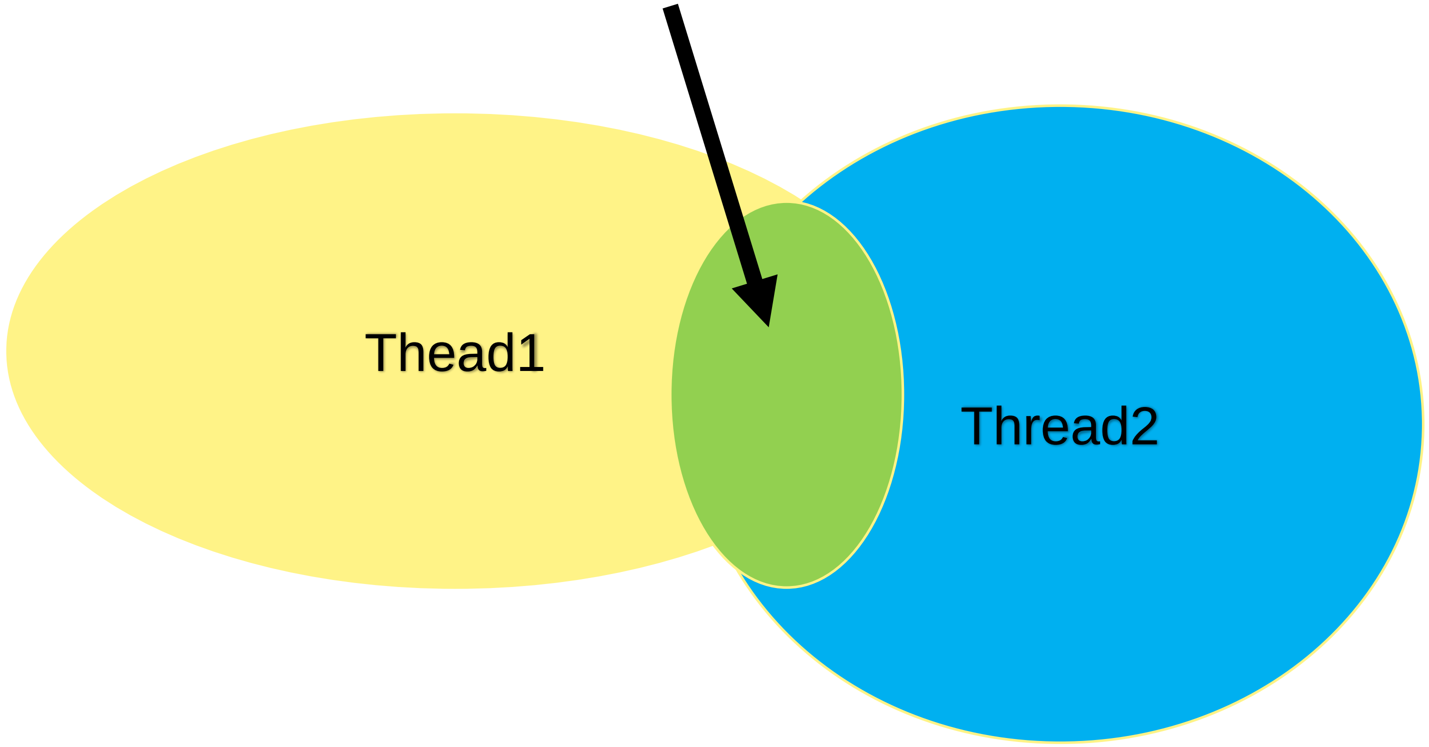
Synchronous Mechanism

Race Conditions:



Critical Regions

Common area (shared data) between several threads



Like ‘done’ in ThreadTest

Control of Critical Sections

A. Mutal Exclusion with busy waiting

   while (x != 0); // do nothing though loop again

   Petersons solution / TSL in machine language

B. Sleep and wakeup

   i. Lock

   ii. Semaphores

   iii. Mutex (binary semaphores)

   iv. Monitors (e.g. bounded buffer)

Sleep and Wait - Semaphore

Semaphore

Down for enter – count down by one if possible otherwise wait

Up for leave – increment by one if not reach roof (counting e.g. max 10)

C# waitOne, Release

Your turn

Prog 4.1-4.2 + Bounded Buffer