Threads in Java and C++

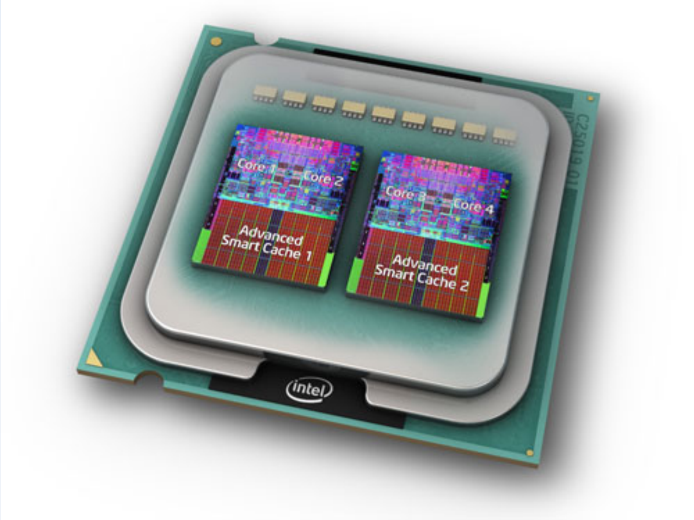
Niklas Meinzer

February 2, 2011

Introduction

Why Multitasking?

- The speed of a single CPU core is limited
  → multiple core machines



- Internet applications

•No active waiting for I/O operations

Threads vs Processes

Both are methods of parallelization, but on a different level

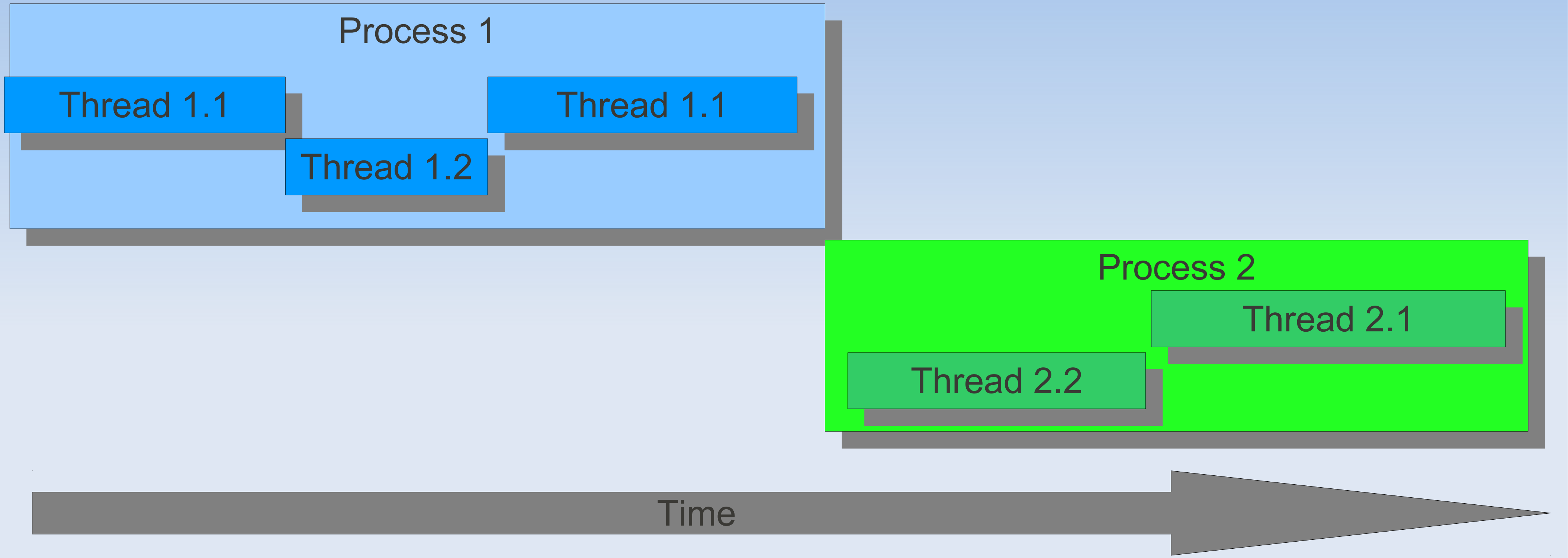
Processes

- independet instance
- private memory space
- Inter-process-communication via OS

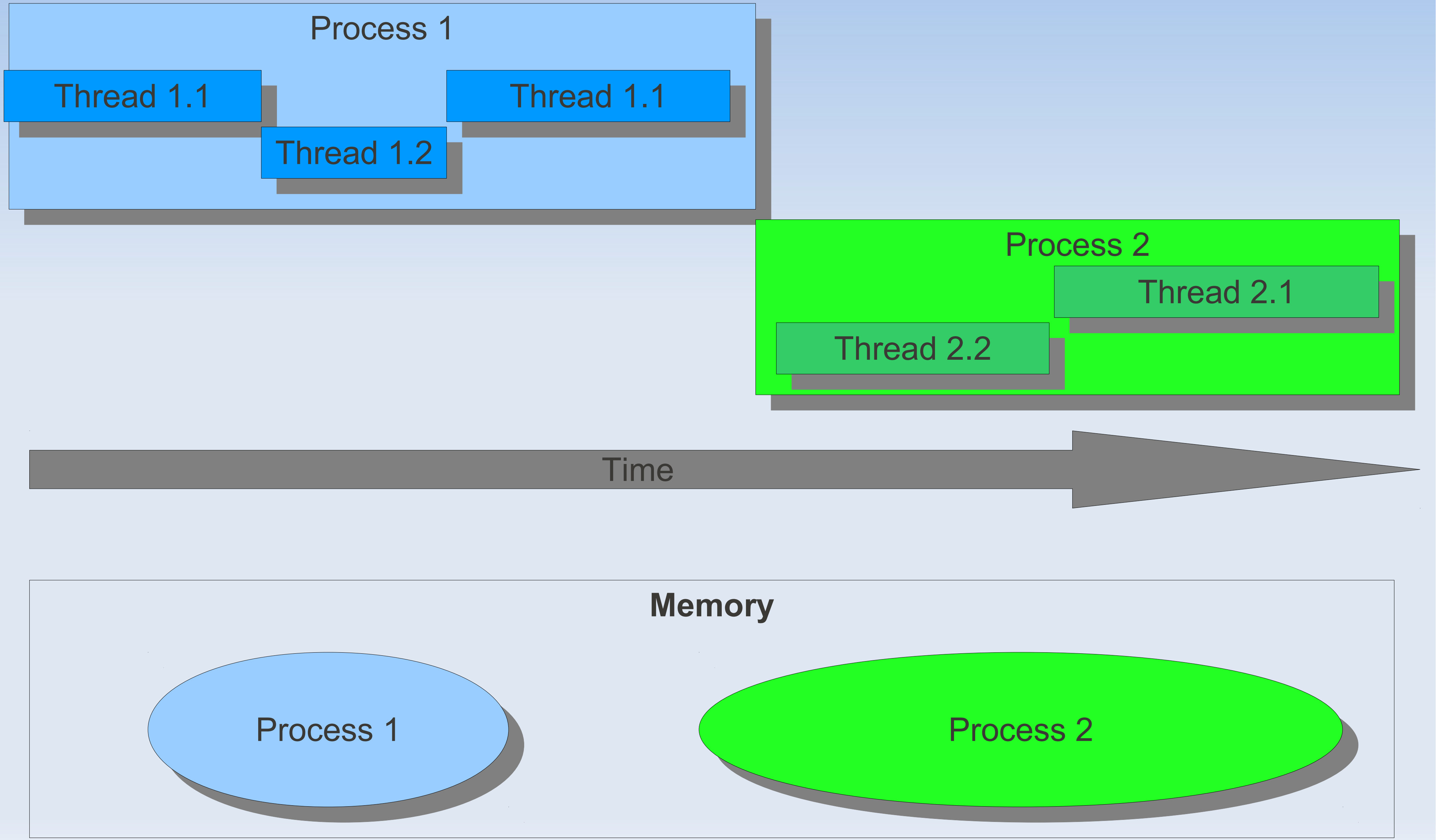
Threads

- subset of a process
- shared memory
- communication via process
- scheduled by OS

Threads vs Processes



Threads vs Processes



Threads vs Processes

- **Each thread has its own:**
  - Stack Pointer
  - Program Counter
  - Registers
  - Scheduling Properties

- **All Threads within a process share:**
  - The program code
  - The heap
  - Files

Threads vs Processes

Trade off

Processes	Threads
”Heavyweight”	”Lightweight”
Stability	one Thread can bring all down
Communication more complicated	Easy communication via shared memory

Problems

- Memory conflicts
- Thread interference
- Deadlock

Threads in Java and C++

Java

- supported ever since
- improvements in Java 5.0 (2004) with

  java.util.concurrent

C++

- no thread support in standard
- different solutions available
- plans to include concurrency in future releases

Java

- Every Java Program consists of at least one thread – the main thread
- Can spawn more Threads using *Thread* or *Runnable* objects
- Syncronisation can be used to prevent memory consistency errors

java.lang.Thread

- Classes that extent *Thread* can be run concurrently
- calculation must be done in *run()* method
- Instances launch a new Thread using Thread.start()

```
class MyThread extends Thread
{
  public void run()
   {
      // Do something
   }
}
```

```
public static void main(...)
{
  MyThread t = new MyThread();
  t.start(); // Start the Tread
  // Continue with something else
}
```

Interface Runnable

- All classes that are intended to be used as a Thread must implement *Runnable* (even *Thread*)
- more flexible

```
class MyRunnable
extends someSuperClass
implements Runnable
{
  public void run()
   {
      // Do something
   }
}
```

```
public static void main(...)
{
  Thread t = new Thread(new MyRunnable());

  t.start(); // Start the Tread
  // Continue with something else
}
```

Mutex: Object Locks

- To ensure mutual exclusion Java uses Object Locks
- Every Object has a corresponding monitor that can only be aquired by one thread at one time
- there are three different ways of using Object Locks in Java

Synchronized Methods

- Can only be executed by one Thread at a time
- Before a Thread calls a synchronized method it must aquire the corresponding Objects monitor

```
class myArray
{
  // …

  public synchronized void initialize()
  {
    // Initialize Array
  }
}
```

Synchronized Static Methods

- Like synchronized methods, but with *static* keyword
- In this case no other instance can call the method

```
class myArray
{
  // …

  public static synchronized void initialize()
  {
    // Initialize Array
  }
}
```

Synchronized Blocks

- Synchronized blocks offer programmers more fine tuning of synchronization
- The Object that provides the lock must be specified explicitly

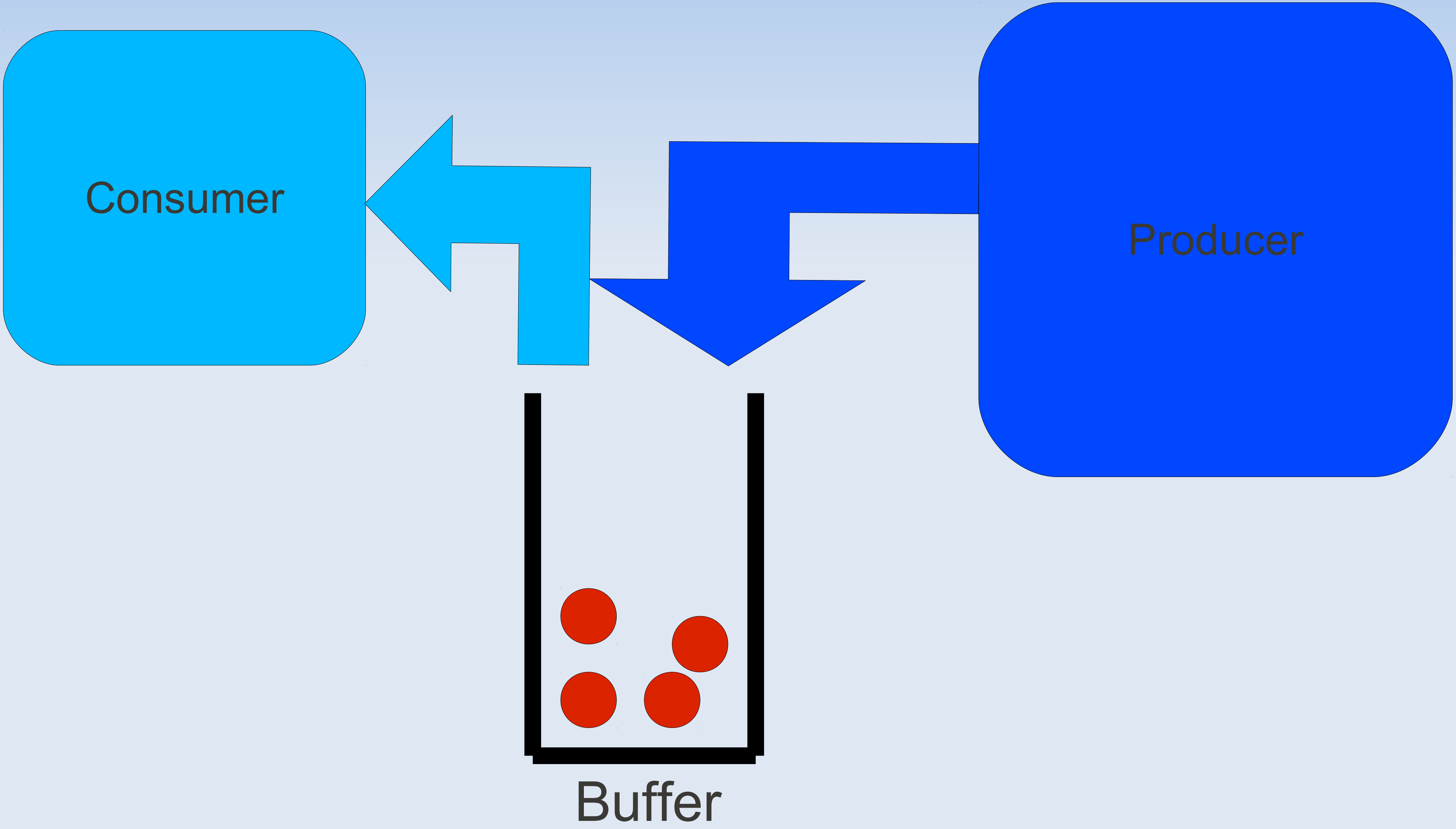
```
Object myLock = new Object();

/* Some operations that
*  are not critical
*/

synchronized(myLock)
{
  // critical code
}

// More non critical code
```

Collaboration of Threads

- wait()
- notifyAll()

- Serve the coordination of Threads and save time through ”smart” scheduling
- can only be called within synchronized code

Producer-Consumer-Problem



wait() & notifyAll()

- if a Thread executes wait() it will go to sleep
- notifyAll() activates all sleeping threads
- there is no way of waking up a specific thread

C++

- No threading in current standard
- Thread libraries:
  - pthreads
  - Boost Threads
- Threads will be included in the next standard (C++0x)

pthreads

- C style library
- Uses IEEE POSIX 1003.1c standard (1995) → pthreads

pthreads

- very low level
- e.g lets user define stack size and adress
- but features most commonly used thread tools
  - mutexes
  - signal and wait

- Often called the Assembler of threaded programming

pthreads usage

- pthread_create(thread, attr, **function***, **arg***)
  - creates and launches a new thread
  - **function*** is a pointer to a function that will be run by the thread
  - **arg***: pointer to functions arguments

Boost

- Large C++ library collection

- A lot of libraries for all kinds of purposes
- **Boost::thread** provides threading infrastructure

Boost Threads usage

- Boost thread can launch procedures as new threads

- The constructor takes one function as argument and immediately starts the thread

```
void myFunction()
{
   // do something
}

int main(int argc, char* argv[])
{
   boost::thread myThread(myFunction); // Thread starts
   // Do something else
}
```

Funktor

- An easy way to create a threadable Object
- An object that overloads the ( ) operator and can thus be called like a function
- The boost::thread constructor will call the () function and run it as a thread

Functor example

```
class TSP
{

public:
  void addNode{
   //...
  }
  void addEdge{
   //...
  }

  void operator()()
  {
    // solve TSP in a seperate thread
  }

private:
  Node *nodeList;
}
```

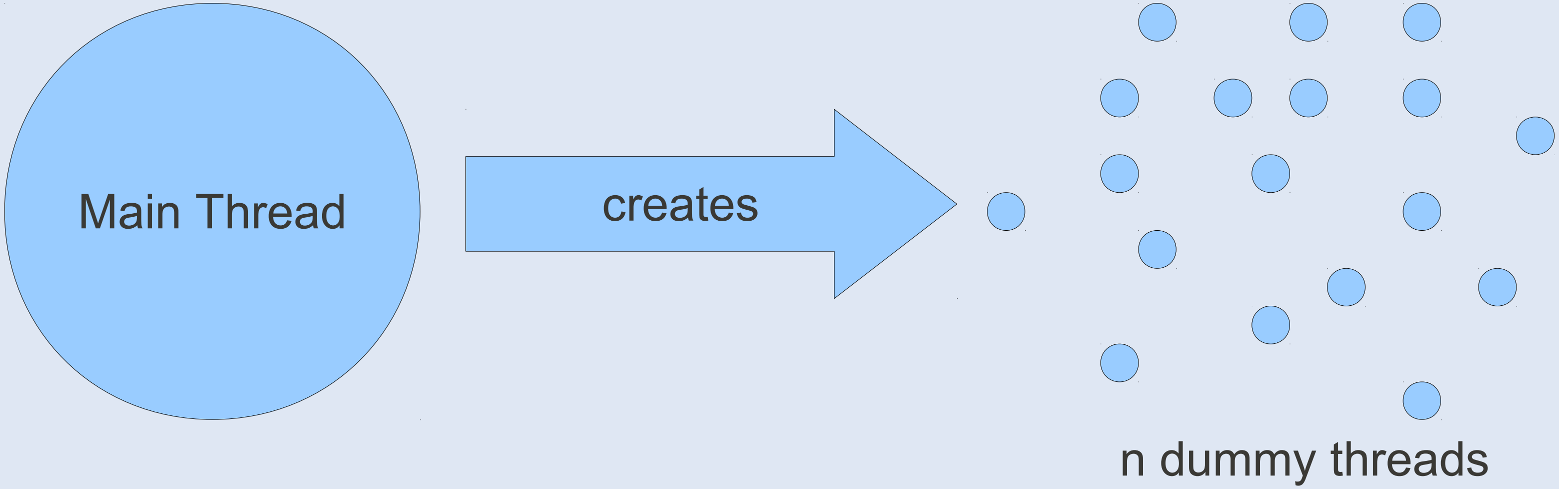
C++0x

- next C++ standard
- will include std::thread
- very similar to boost

Experiments

Test 1: Thread creation

- Create n threads that count up to 1000

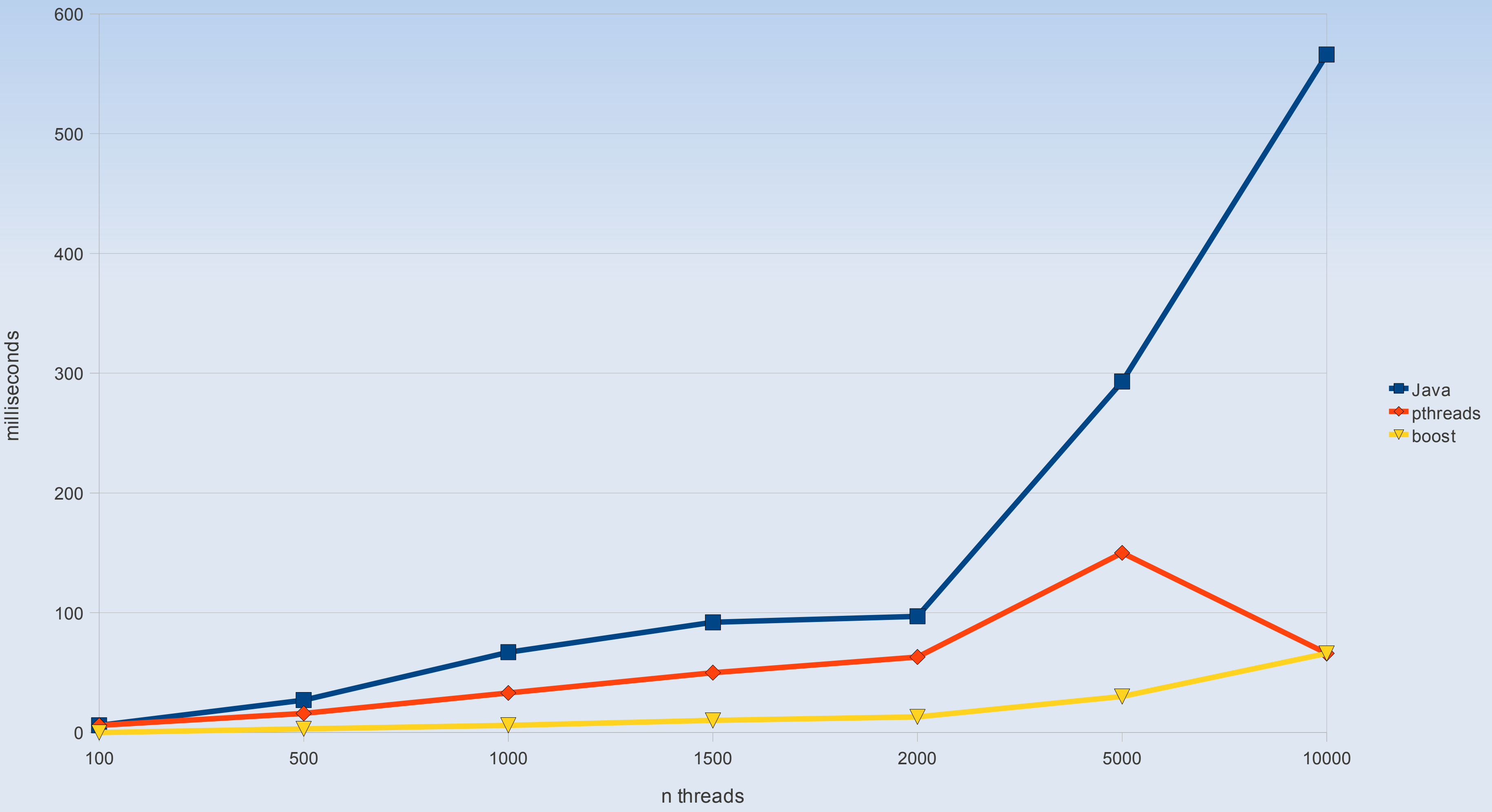


Test results (thread creation)

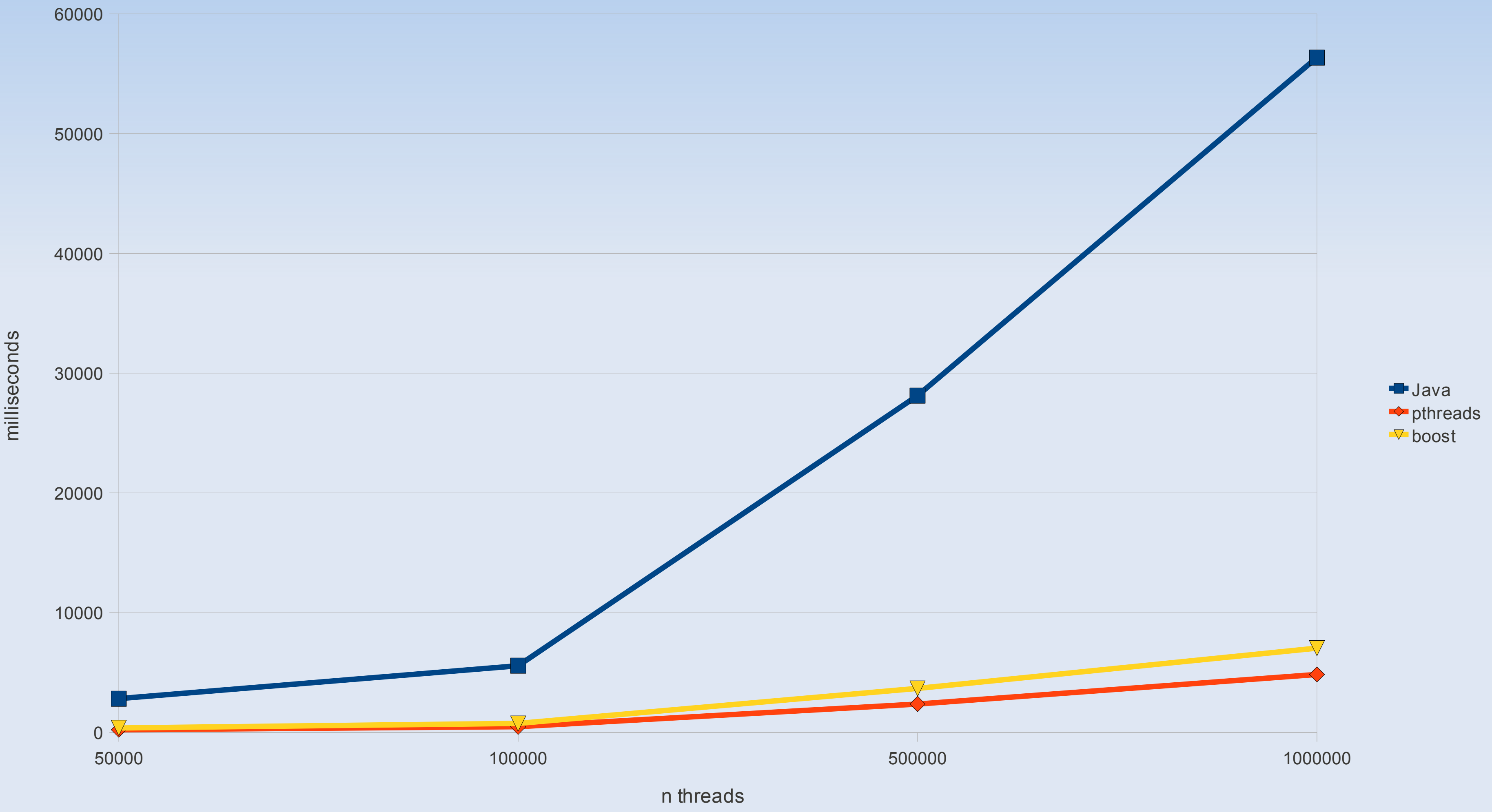
- Create n threads that count up to 1000

n	Java	pthreads	boost threads
1000	0.150 s	0.28 s	0.013 s
10,000	0.740 s	0.240 s	0.770 s
100,000	5.700 s	1.100 s	0.667 s
1,000,000	56.000 s	5.100 s	6.800 s

Test results (thread creation)

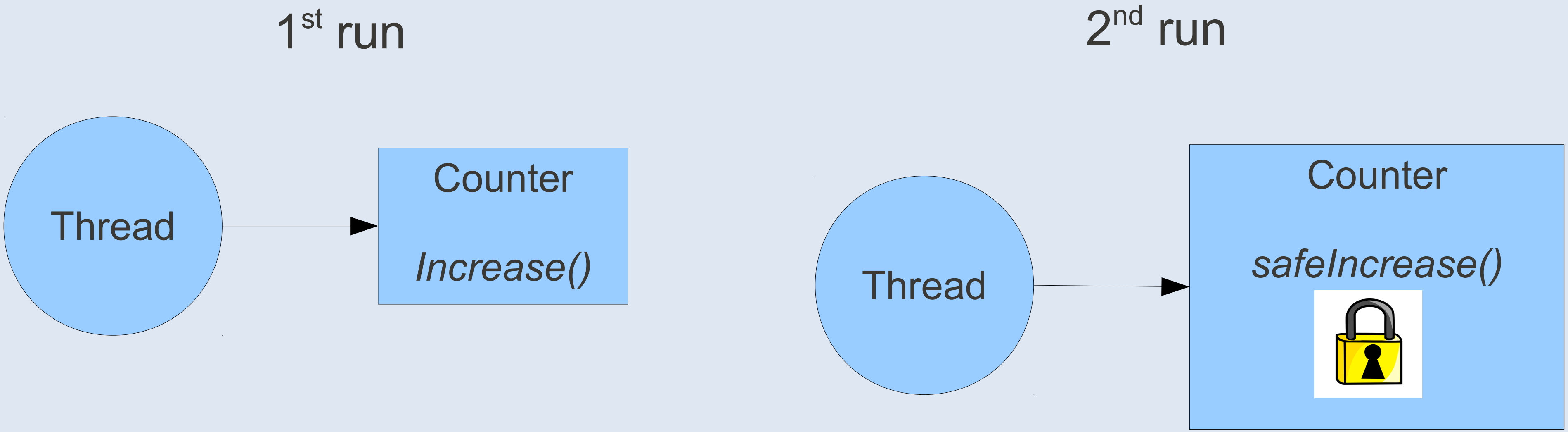


Test results (thread creation)



Test 2: Locking

- What is the overhead of making a function thread safe



1st run

Thread

Counter

Increase()

2nd run

Thread

Counter

safeIncrease()

Test results (Locking)

Goal	Java		pthreads		boost threads	
	unlocked	locked	unlocked	locked	unlocked	locked
100,000	1 ms	5 ms	1 ms	4 ms	1 ms	7 ms
500,000	2 ms	20 ms	5 ms	14 ms	4 ms	17 ms
1,000,000	3 ms	23 ms	5 ms	22 ms	5 ms	35 ms
10,000,000	23 ms	235 ms	50 ms	219 ms	58 ms	356 ms
100,000,000	255 ms	2253 ms	494 ms	2194 ms	524 ms	3599 ms
500,000,000	1182 ms	11140 ms	2481 ms	11010 ms	2629 ms	18108 ms
	~ factor 10		~ factor 4.5		~ factor 7	

Sources

- oracle.com – Java Tutorials
- *”Inside the Java Virtual Machine”* by Bill Venners

- computing.llnl.gov – Tutorials on POSIX Threads
- www.boost.org
- antonym.org – Boost threads tutorial