NWEN303 : Concurrent Programming

***Marco Servetto* & Qiang Fu**

Lecture 1: Introduction

- Course organization
- Motivation
- Basic concepts
- Resources

Course Organization

Before the break:

Lectures:

Tuesday - 13:10- 14:00 Hugh Mackenzie LT103

Thursday - 13:10-14:00 Hugh Mackenzie LT103

Friday - 13:10-14:00 Hunter LT220

Labs:

Thursday - 12:00 to 13:00 and 13:00 to 14:00 CO238

Some weeks we have 3 lectures, some weeks we have a lab instead.

After the break:

Lectures:

Tuesday - 13:10- 14:00 Hugh Mackenzie LT103

Thursday - -14:00 Hugh Mackenzie LT103

Labs:

Monday - 12:00 to 13:00 and 13:00 to 14:00 CO238

Must:

- achieve across the assignments an average of at least 40%.
- achieve at least a 'D' grade in the two term tests.

Assessment:

2 small assignments	16% (8% each)
2 large assignments	30% (15% each)
Fist Term test	24%
Second Term Test	30%

People and Contact Information

- Lecturer: Marco Servetto: Co258, 463 5820 (marco.servetto@ecs.vuw.ac.nz)
  Office hours: Tue 12-13pm, Fri 11-13pm
- Lecturer: Qiang Fu:414 Alan MacDiarmid, 463 5233 (qiang.fu@vuw.ac.nz)
- School Office: Co358, 463 5341 (office@ecs.vuw.ac.nz)
- Tutors: Harrison Cook and Hayden Andersen
- Class rep: to decide

Lecture Handouts and Recording

- Lecture notes will be posted online

- Lectures will be recorded.
  - Access via Blackboard
  - Expect some recording issues to happen

Web Resources

http://ecs.victoria.ac.nz/Courses/NWEN303_2021T1

- Course outline
- Lecture notes, old exams,
- Additional reference material
- Forum
- Submission

What to expect from this course

Theory: terms to learn to think concurrently and applications of those terms on examples

Practice: Coding in Java correct solution to problems involving concurrency. A correct solution is not just a solution that happens to work when you casually run your code.

Most of the focus is going to be about designing correct solutions. Once you understand what to do, actually coding it is often trivial using high level libraries.

Nothing of: C/C++/Linux pthread library

Guidance: most assignments are designed so that you have some time to attempt them individually, a lab to support you with any problem you may encounter, and a little bit of time after the lab to conclude your work and submit.
We will also have a mock term test to prepare for the real deal.

Model solutions for most assignments are discussed in class.

Terminology

Program: an independent executable unit of code
    'Game.jar'

Process: a specific running execution of a Program.
  It have its own private paginated memory (false?)
    'java -jar Game.jar'

Thread: low-level technique for parallel execution.
  All threads share the same memory
    'main thread, gc thread, thread pool'

Worker: high level abstraction for concurrent execution.
  run over a Process, or a Tread, or even move between threads/processed.
    'NPC possible moves are computed by workers'

What is Concurrency?

- Several processes/threads/workers:
  - Executing “simultaneously”
  - Interacting and/or sharing resources
- Examples:
  - Operating Systems (I/O, file system...)
  - Database systems (multiple clients)
  - Control systems
  - Interactive systems
  - Large scale computation: forecasting, modeling, animation

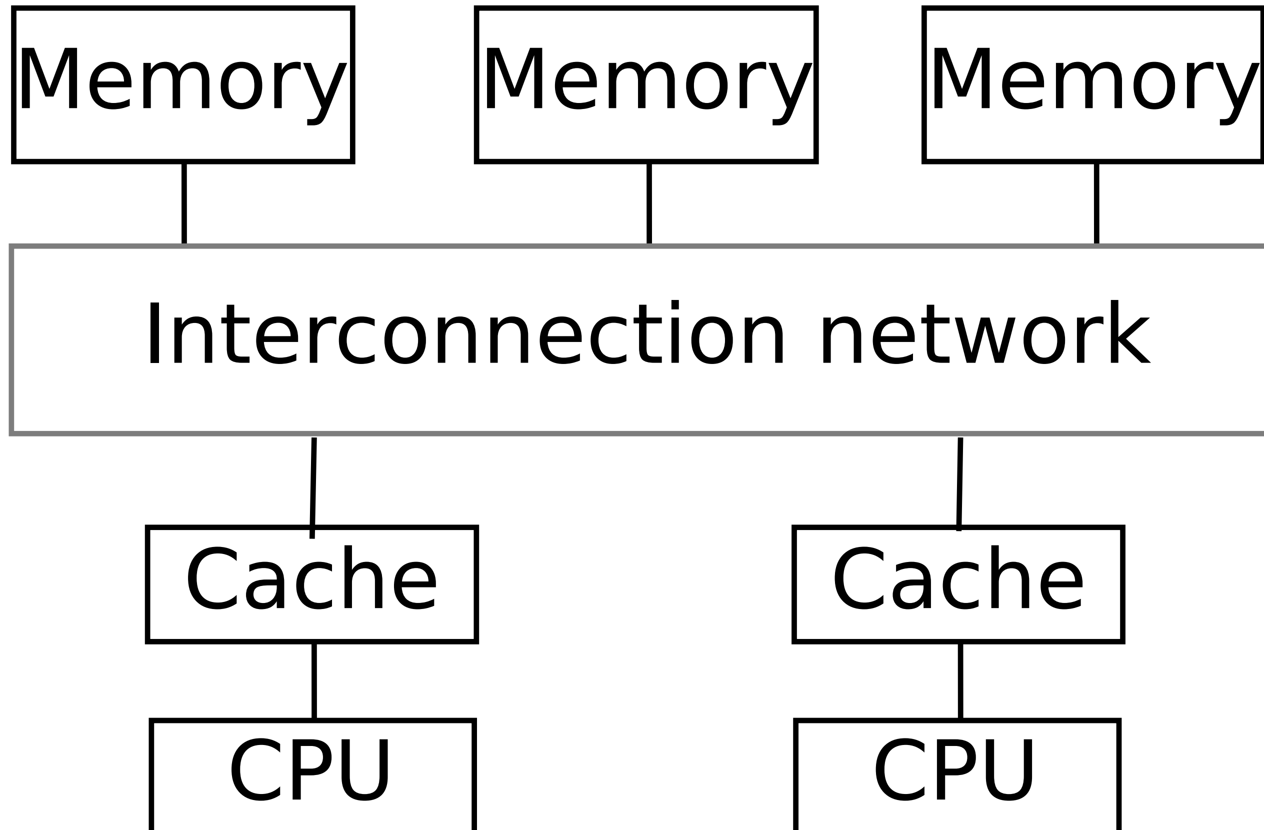
Why Use Concurrency?

- Communicating between physically separate machines
- Utilizing multiple processors/cores – speed up, e.g. pipeline
- Ensuring responsiveness in interactive systems, e.g. web applications, games

Major Paradigms

- Shared memory (first half)
  Workers interact by modifying locations in shared memory.
- Message passing (~second half)
  Workers interact by sending/receiving messages.
- Synchronous, data parallel
  Workers execute the same instructions, at the same time, on different parts of the data.

Shared-Memory



Shared-Memory

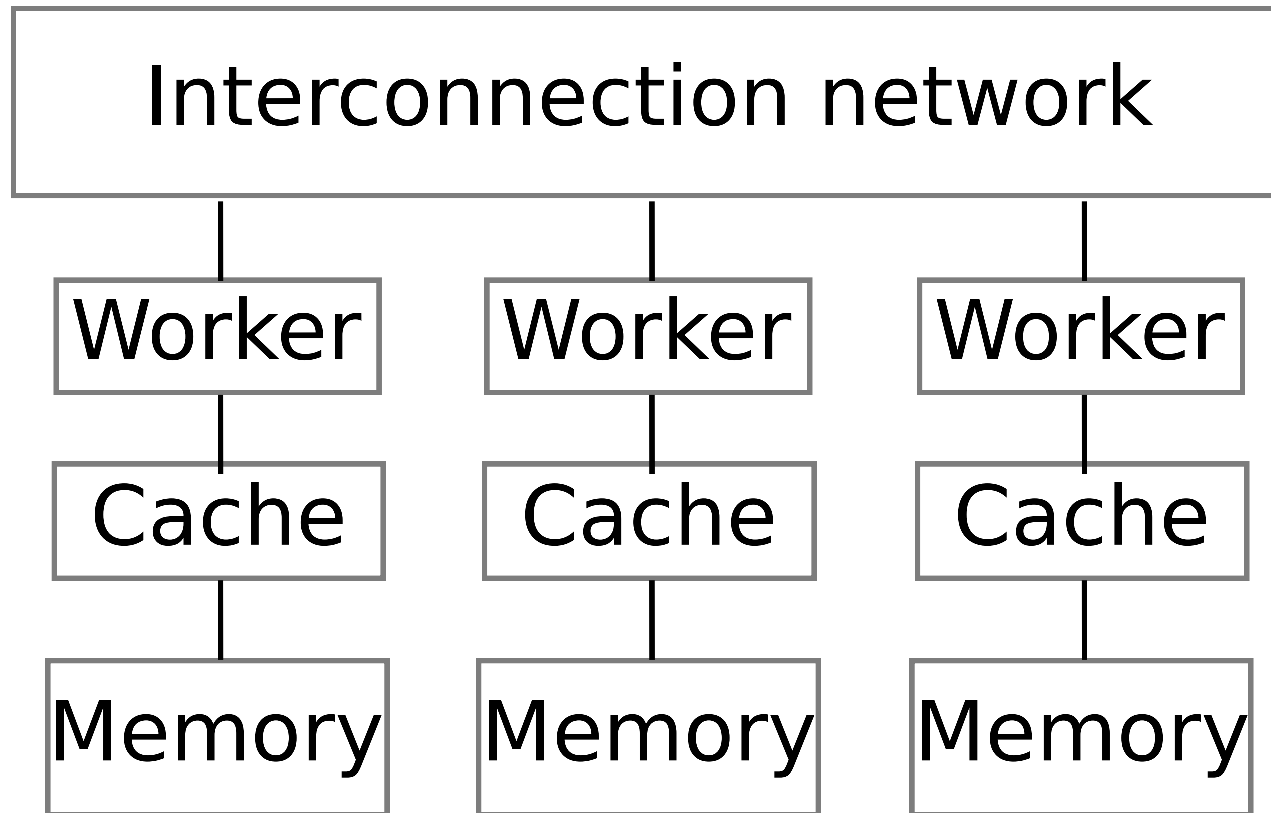
In a shared memory system:

- Workers may run on separate CPUs
  - May run at different speeds
- Workers may run on same CPU
  - Execution controlled by scheduler

In both cases, we can't predict the order in which reads and writes on shared memory take effect.

Typically they all run on the same physical machine

Distributed-Memory



Distributed-Memory

Distributed memory can be obtained by either using multiple physical machines, multiple processes or disciplined usage of threads.

Communication happens by **Message Passing.**

Can't predict order in which messages will be received.

Issues in Concurrency

- Software engineering/architecture:
  - Concurrency models
  - Identifying and implementing good primitives:
    - Libraries job!
- Performance: maximizing concurrency?
- Correctness
  - Testing is difficult (non-deterministic behavior)
  - Static analysis, Type systems, Patterns

Shared Memory Concurrency

- Concepts: atomic steps, interleaving, interference
- Concurrency in Java: ~~Threads~~, parallelStreams, synchronized, volatile, concurrency libraries, Java memory model
- Correctness: safety, liveness/progress, starvation, fairness, correctness conditions
- Designing concurrent algorithms and data structures

Concurrency: why

- Performance: In certain very specific scenarios, performance may be an issue.
  - "Programmers waste enormous amounts of time thinking about, or worrying about, the speed of noncritical parts of their programs, and these attempts at efficiency actually have a strong negative impact when debugging and maintenance are considered. We should forget about small efficiencies, say about 97% of the time: premature optimization is the root of all evil. Yet we should not pass up our opportunities in that critical 3%." --DonaldKnuth
- Avoid premature Optimization: write your code sequentially first, but in a way that makes easy/possible to inject parallelism on need using libraries.
- Thus, computation intensive components of your code should have a clear functional interface. No need to be functional inside, just the interface needs to be functional; then, if performance issues arises:
  - Profile your program to understand where are the points consuming most time. While still keeping your code sequential, try to optimize those points.
  - If you still need more performance, then you may consider using concurrency.
- Programming in this way have great advantages in testing, debugging and maintainability, dwarfing the performance benefit of concurrency.

Concurrency: why

- Reactiveness: Executing multiple actions at the same time means that (a) new input can be processed while former computation is still taking place, and that (b) partial results of a computation can be visible before the whole computation is completed.
- For example while clicking on a button in a GUI, (b) we can see the button pop up again even if the triggered computation is still ongoing.
- Moreover, (a) we can navigate in various menu while computation is being performed
- In most problem domains, good Libraries can completely hide the presence of concurrency. However sometimes for (potential) performance reasons, concurrency is still visible and requires care.
- Moreover, **writing such libraries** requires to handle concurrency directly.

Wisdom from Stack Overflow:

With Java 8 and lambdas it's easy to iterate over collections as streams, and just as easy to use a parallel stream. Two examples from the docs, the second one using parallelStream:

```
myShapesCollection.stream()//.parallelStream()
    .filter(e -> e.getColor() == Color.RED)
    .forEach(e -> System.out.println(e.getName()));
```

As long as I don't care about the order, would it always be beneficial to use the parallel? One would think it is faster dividing the work on more cores. Are there other considerations? When should parallel stream be used and when should the non-parallel be used?   -------------Matsemann

---

A parallel stream has a much higher overhead compared to a sequential one. Coordinating the threads takes a significant amount of time. I would use sequential streams by default and only consider parallel ones if

- I have a massive amount of items to process (or the processing of each item takes time and is parallelizable).
- I have a performance problem in the first place.
- I don't already run the process in a multi-thread environment (for example: in a web container, if I already have many requests to process in parallel, adding an additional layer of parallelism inside each request could have more negative than positive effects).

In your example, the performance will anyway be driven by the synchronized access to System.out.println(), and making this process parallel will have no effect, or even a negative one. Moreover, remember that parallel streams don't magically solve all the synchronization problems. If a shared resource is used by the predicates and functions used in the process, you'll have to make sure that everything is thread-safe. In particular, side effects are things you really have to worry about if you go parallel.

--------JB Nizet //best answer with 369 upvotes

Examples of Lambdas

```java
static void out(String s){ System.out.print(s); }

static <T,R> R call(T t,Function<T,R> f){ out("Call#"+t); return f.apply(t); }

static <T> T id(T t){ out("Id#"+t); return t; }

public static void main(String[]arg){
   int x0=id(id(5)+5);//What it prints?
   int x1=call(id(12),y->y*2);//What it prints?
   int x2=call(12,y->id(y)*2);//What it prints?
   int x3=call(12,y->id(y)*id(2));//What it prints?
   int x4=call(12,y->id(id(5)+5));//What it prints?
   int x5=call(12,y->call(42,z->y));//What it prints?
   }
```

Examples of Lambdas

```
static void out(String s){ System.out.print(s); }

static <T,R> R call(T t,Function<T,R> f){ out("Call#"+t); return f.apply(t); }

static <T> T id(T t){ out("Id#"+t); return t; }

public static void main(String[]arg){
   int x0=id(id(5)+5);//prints Id#5Id#10, and x0==10
   int x1=call(id(12),y->y*2);//
   int x2=call(12,y->id(y)*2);//
   int x3=call(12,y->id(y)*id(2));//
   int x4=call(12,y->id(id(5)+5));//
   int x5=call(12,y->call(42,z->y));//
   }
```

Examples of Lambdas

```
static void out(String s){ System.out.print(s); }

static <T,R> R call(T t,Function<T,R> f){ out("Call#"+t); return f.apply(t); }

static <T> T id(T t){ out("Id#"+t); return t; }

public static void main(String[]arg){
  int x0=id(id(5)+5);//prints Id#5Id#10, and x0==10
  int x1=call(id(12),y->y*2);//prints Id#12Call#12, and x1==24
  int x2=call(12,y->id(y)*2);//
  int x3=call(12,y->id(y)*id(2));//
  int x4=call(12,y->id(id(5)+5));//
  int x5=call(12,y->call(42,z->y));//
  }
```

Examples of Lambdas

```
static void out(String s){ System.out.print(s); }

static <T,R> R call(T t,Function<T,R> f){ out("Call#"+t); return f.apply(t); }

static <T> T id(T t){ out("Id#"+t); return t; }

public static void main(String[]arg){
  int x0=id(id(5)+5);//prints Id#5Id#10, and x0==10
  int x1=call(id(12),y->y*2);//prints Id#12Call#12, and x1==24
  int x2=call(12,y->id(y)*2);//prints Call#12Id#12, and x2==24
  int x3=call(12,y->id(y)*id(2));//
  int x4=call(12,y->id(id(5)+5));//
  int x5=call(12,y->call(42,z->y));//
  }
```

Examples of Lambdas

```
static void out(String s){ System.out.print(s); }

static <T,R> R call(T t,Function<T,R> f){ out("Call#"+t); return f.apply(t); }

static <T> T id(T t){ out("Id#"+t); return t; }

public static void main(String[]arg){
   int x0=id(id(5)+5);//prints Id#5Id#10, and x0==10
   int x1=call(id(12),y->y*2);//prints Id#12Call#12, and x1==24
   int x2=call(12,y->id(y)*2);//prints Call#12Id#12, and x2==24
   int x3=call(12,y->id(y)*id(2));//prints Call#12Id#12Id#2, and x3==24
   int x4=call(12,y->id(id(5)+5));//
   int x5=call(12,y->call(42,z->y));//
}
```

Examples of Lambdas

```
static void out(String s){ System.out.print(s); }

static <T,R> R call(T t,Function<T,R> f){ out("Call#"+t); return f.apply(t); }

static <T> T id(T t){ out("Id#"+t); return t; }

public static void main(String[]arg){
  int x0=id(id(5)+5);//prints Id#5Id#10, and x0==10
  int x1=call(id(12),y->y*2);//prints Id#12Call#12, and x1==24
  int x2=call(12,y->id(y)*2);//prints Call#12Id#12, and x2==24
  int x3=call(12,y->id(y)*id(2));//prints Call#12Id#12Id#2, and x3==24
  int x4=call(12,y->id(id(5)+5));//prints Call#12Id#5Id#10, and x4==10
  int x5=call(12,y->call(42,z->y));//
}
```

Examples of Lambdas

```
static void out(String s){ System.out.print(s); }

static <T,R> R call(T t,Function<T,R> f){ out("Call#"+t); return f.apply(t); }

static <T> T id(T t){ out("Id#"+t); return t; }

public static void main(String[]arg){
    int x0=id(id(5)+5);//prints Id#5Id#10, and x0==10
    int x1=call(id(12),y->y*2);//prints Id#12Call#12, and x1==24
    int x2=call(12,y->id(y)*2);//prints Call#12Id#12, and x2==24
    int x3=call(12,y->id(y)*id(2));//prints Call#12Id#12Id#2, and x3==24
    int x4=call(12,y->id(id(5)+5));//prints Call#12Id#5Id#10, and x4==10
    int x5=call(12,y->call(42,z->y));//prints Call#12Call#42, and x5==12
    }
```