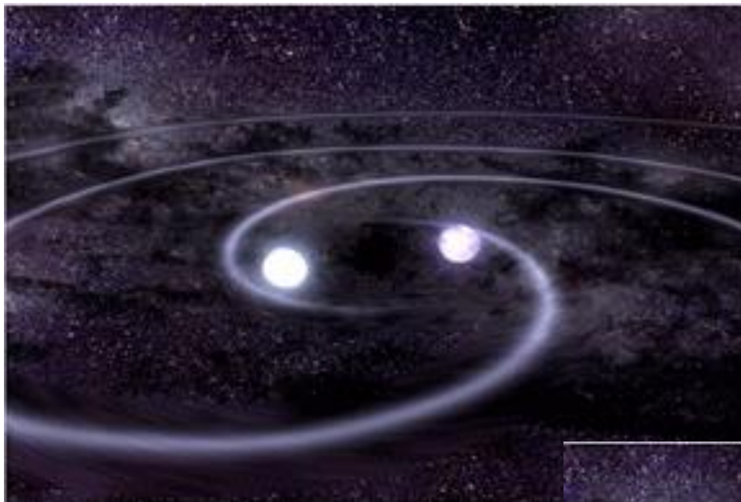
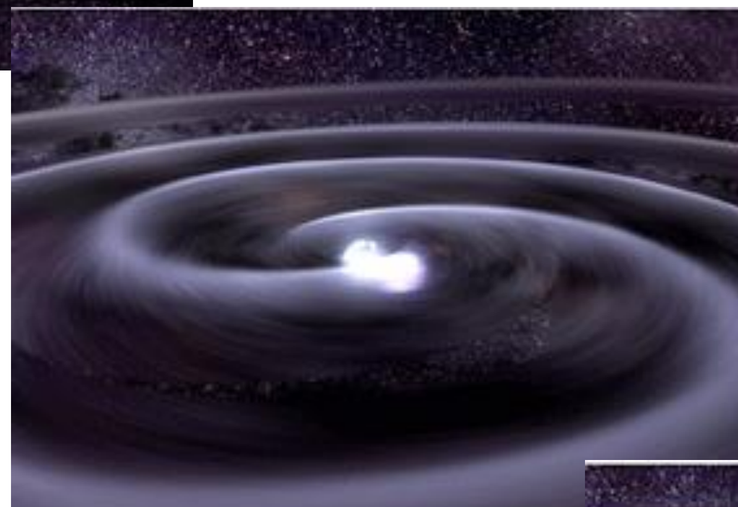
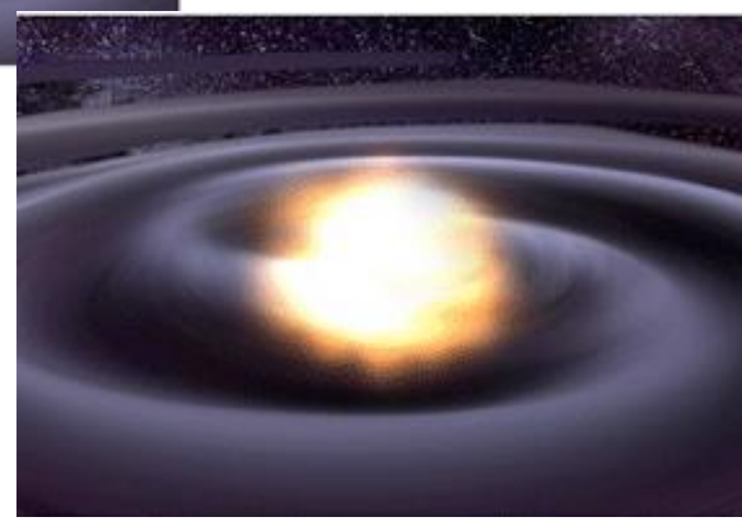
Parallel Systems

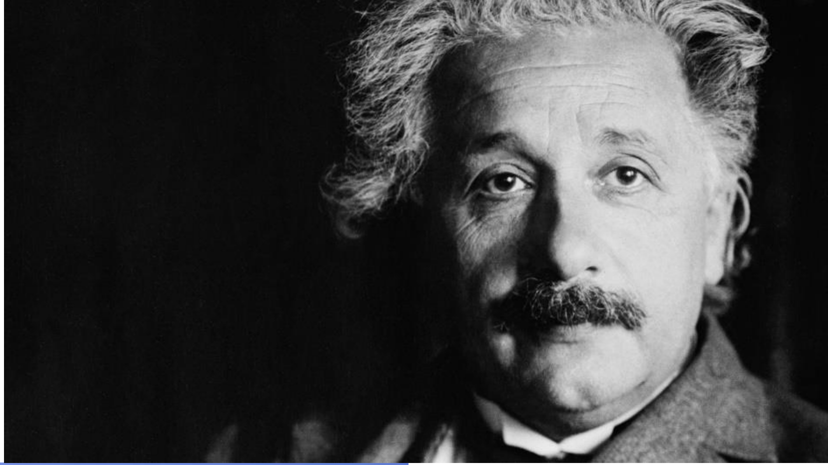
Introduction

Principles of Parallel Programming, Calvin Lin & Lawrence Snyder,
Chapters 1 & 2

Jan Lemeire
Parallel Systems
September - December 2017

Vrije Universiteit Brussel

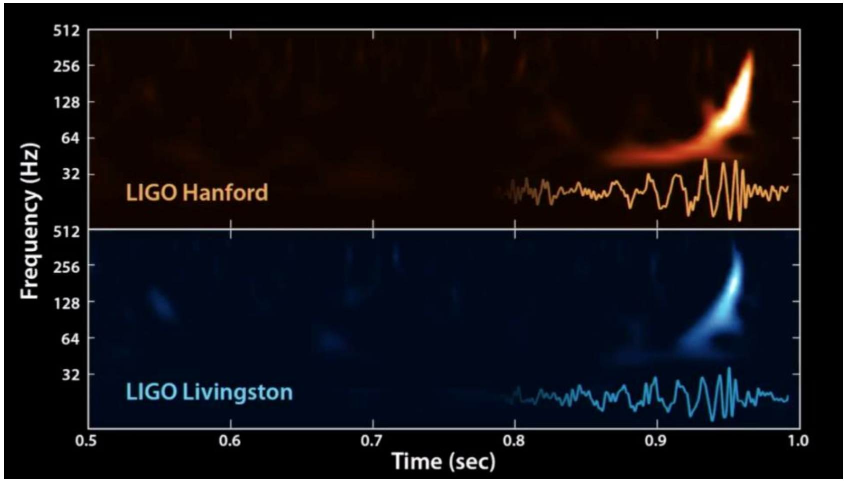




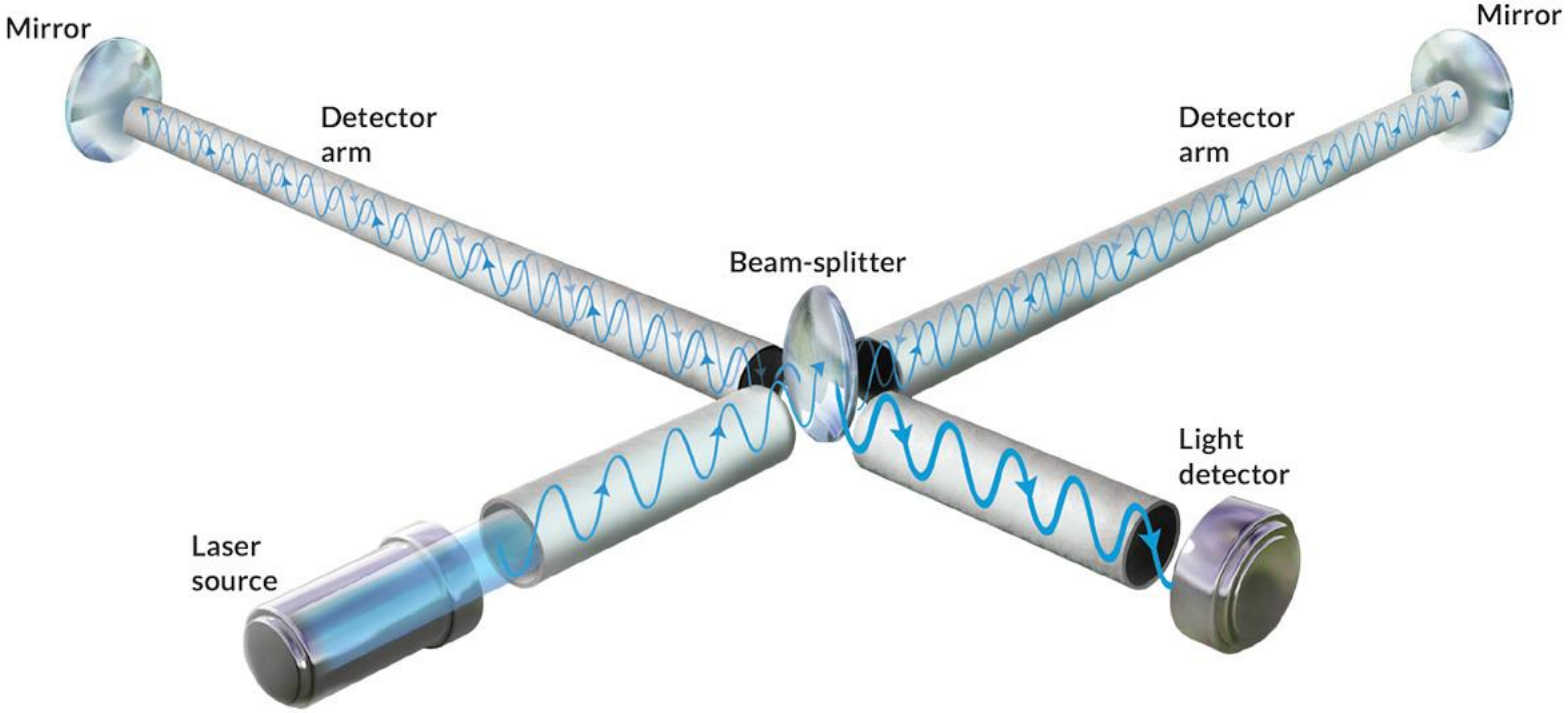


LIGO

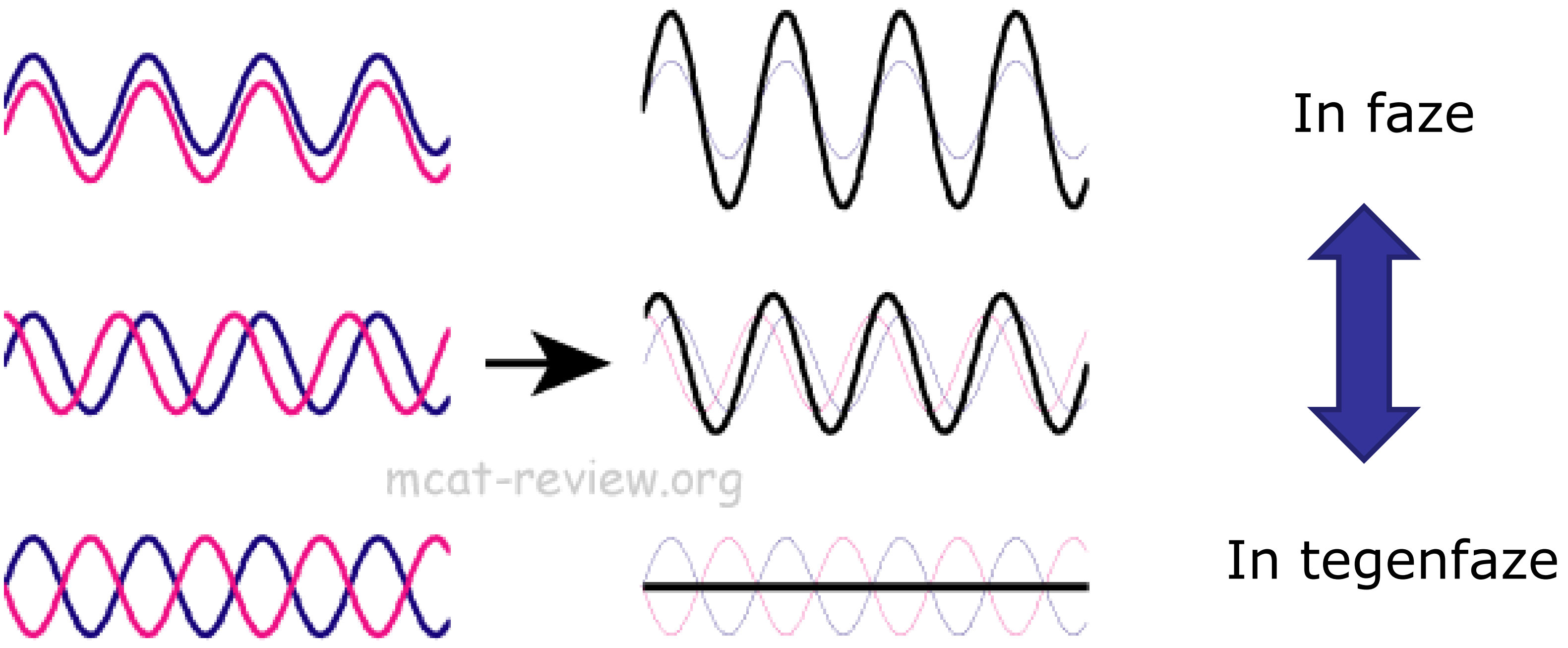
LIGO
The First Observation of Gravitational Waves

September 14, 2015

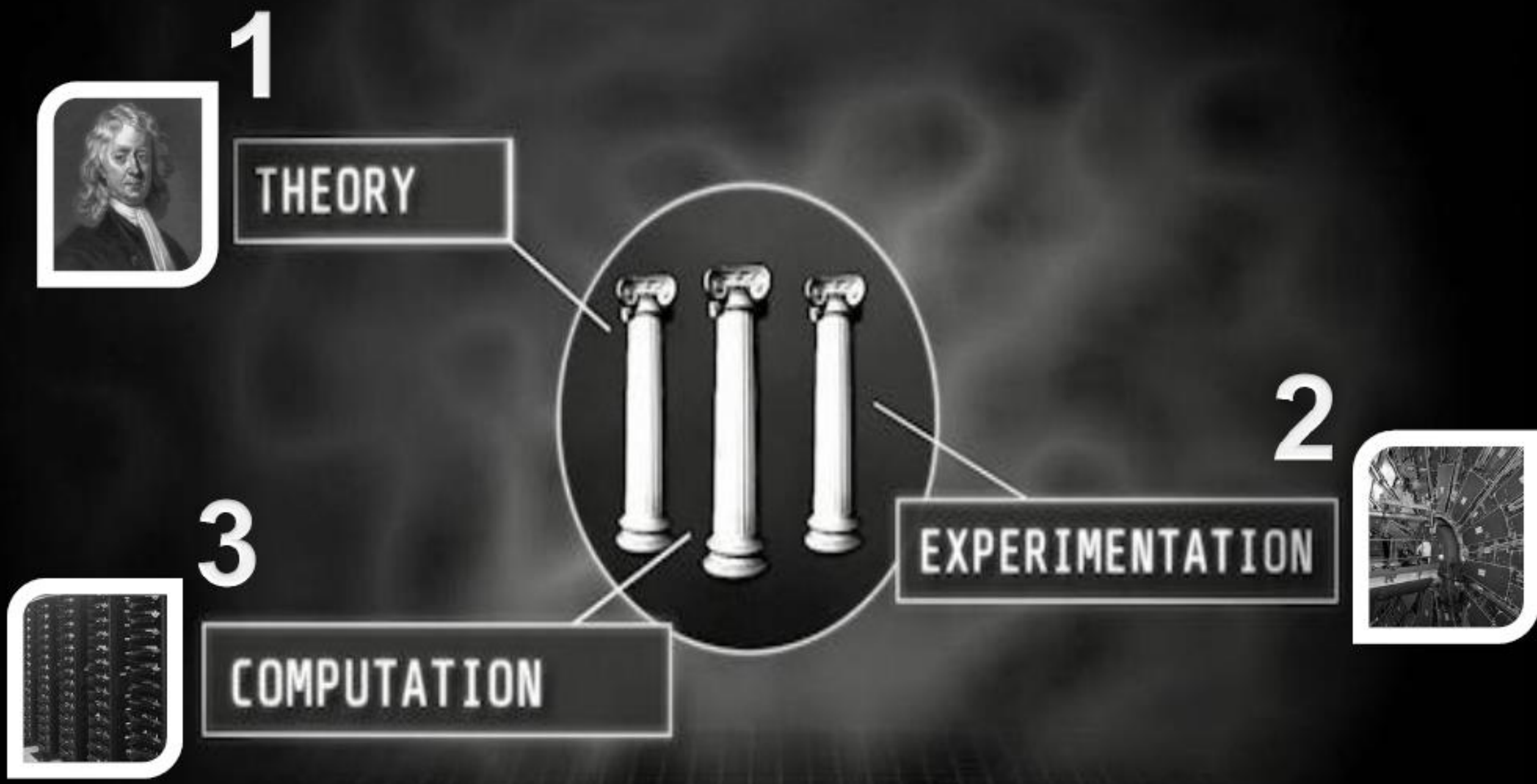




Superpositie van signalen



UPDATE: August 14, 2017
Detection of a gravitional wave by 3 detectors, also by the European Virgo.
1.8 billion years ago, collision of black holes of 25 and 31 times the mass of the sun



The third pillar
of the scientific world

Jan Lemeire (jan.lemeire@vub.ac.be)

- Civil Engineer - elektronics, 1994, VUB
  + additional masters in computer sciences (1995)
- Worked for 4 year in the private sector, 2 IT-consultancy companies
- 2000-2007: did PhD at the VUB as assistant
  - Thaught practica informatics
- Since 2008: professor at VUB
  - Course 'Parallel systems' in the masters
  - Since 2011: 'Informatics' first year bachelors engineer
- Since october 2013: teaching to engineers in industrial sciences
  - Computer architecture, electronics, informatics
- Research topics: parallel processing, gpu computing, processor architectures & data mining/machine learning/probabilistic models
- http://parallel.vub.ac.be

Goals of course

- Understand architecture of modern parallel systems.
- Employ software technologies for parallel programming.
- Design efficient and two-fold generic parallel solutions.
  - For a wide variety of parallel systems & broad class of similar algorithms.
  - Sharpen your low-level and high-level IT skills.
- Understand their performance.
- Make successful technology. Understand economics.

50% Oral exam on theoretical part
50% Project: parallelize an algorithm with the 3 technologies

References

PPP

- "[Principles of Parallel Programming]" by Calvin Lin and Larry Snyder
  - Chapters 1-6, 7 (partly)

KUMAR

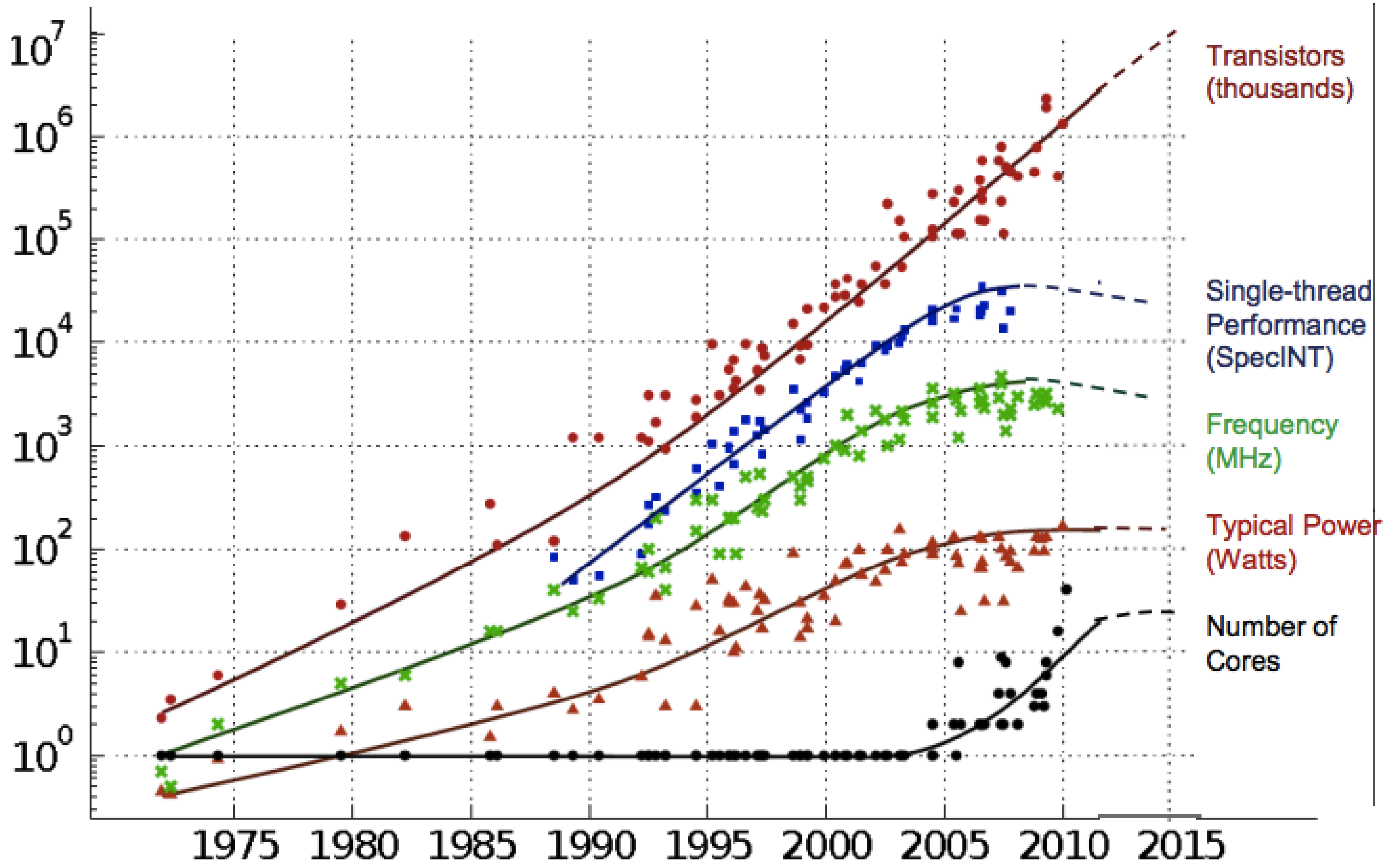
- "[Introduction to Parallel Computing]" by Grama, Gupta, Karypsis & Kumar.
  - Chapters 1-7, 8.2, 9, 11

- http://parallel.vub.ac.be/education/parsys

Parallel computing is hot

1. Urgent need
2. New technologies

The Free Lunch is Over



Chuck Moore, "DATA PROCESSING IN EXASCALE-CLASS COMPUTER SYSTEMS", The Salishan Conference on High Speed Computing, 2011.

The Free Lunch is Over

Why You Don't Have 10GHz Today?

- **heat/surface is the problem (power wall)**
- **12 nm would mean electric paths of 10 atoms wide!**

Moreover:

- **memory bottleneck**
- **instruction level parallelism (ILP) wall**

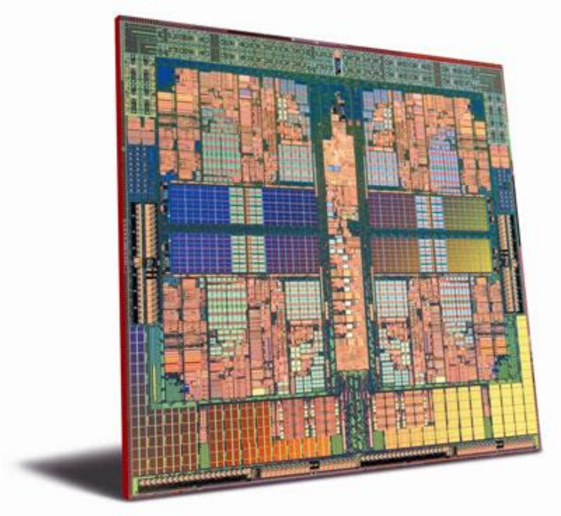
What about Moore's Law?

- ➢ **increase of Clock speed: stopped**
- ➢ **increase of Transistors: ongoing**

It's now about the number of cores!

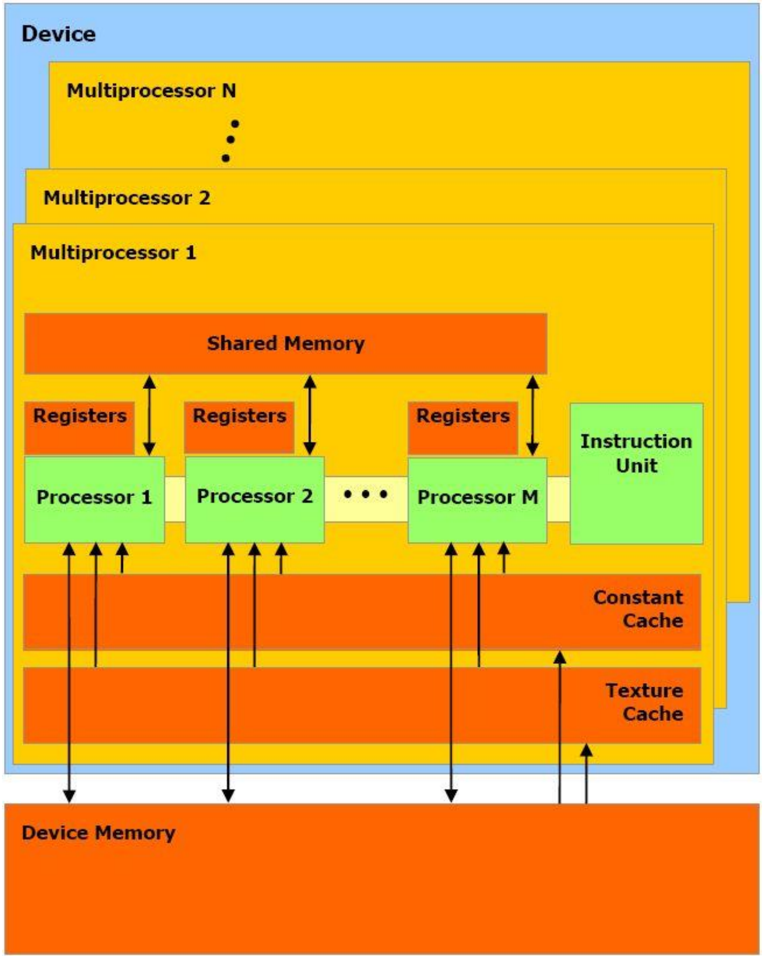
http://www.gotw.ca/publications/concurrency-ddj.htm

Multi- & manycores



Graphics Card Processors

Graphics card



Goals of this lesson

- What is a parallel system?
- Basics of parallel programming.
- Why are they harder to program than sequential computers?

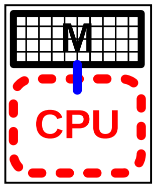
Overview

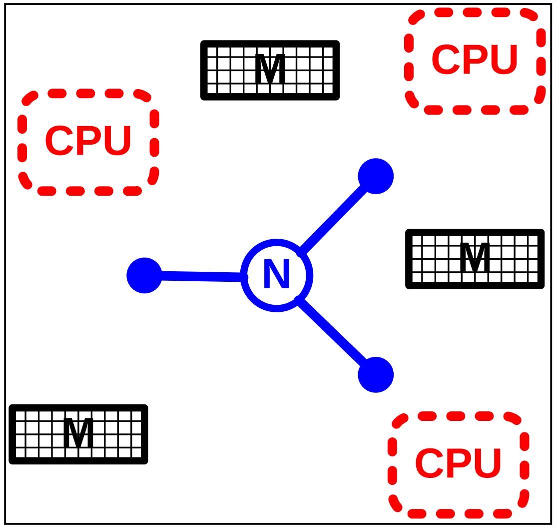
1. Definition
2. Why?
3. Parallel compiler?
4. Parallel architectures
5. Parallel Processing Paradigms
   - Multi-threading.
   - Message-passing.
6. End notes

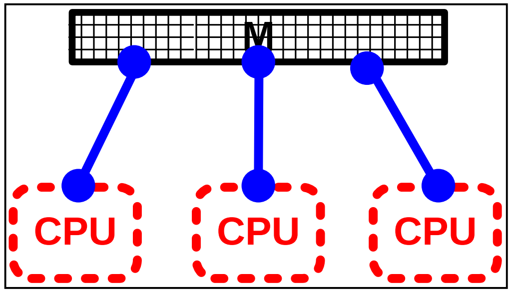
Overview

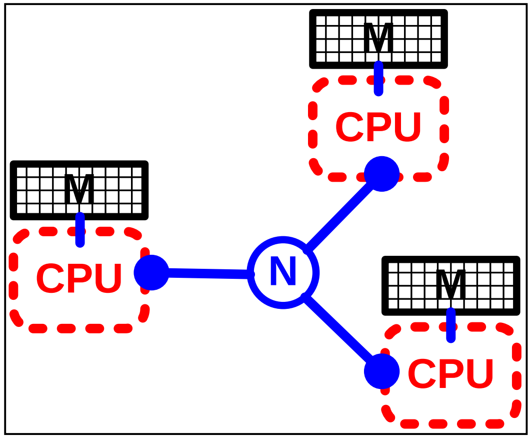
1. **Definition**
2. Why?
3. Parallel compiler?
4. Parallel architectures
5. Parallel Processing Paradigms
   - Multi-threading.
   - Message-passing.
6. End notes

What is a Parallel System?





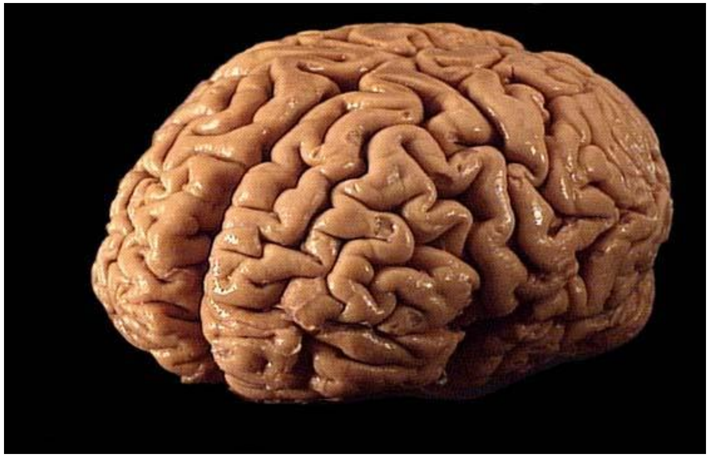




- Memory
- Processors
- Interconnect

Biggest Parallel System?

Internet



Brain

Frequency of brain waves: 10Hz
Number of neurons: 100 billion = 10^{11}

A bit of a History

- **1980s, early `90s: a golden age for parallel computing**
  - special parallel computers: **Connection Machine, MasPar, Cray (VUB also!)**
  - True supercomputers: incredibly exotic, powerful, expensive
  - Based on *vectorization* (see further)
- But…impact of data-parallel computing limited
  - Thinking Machines sold 100s of systems in total
  - MasPar sold ~200 systems

History II: now

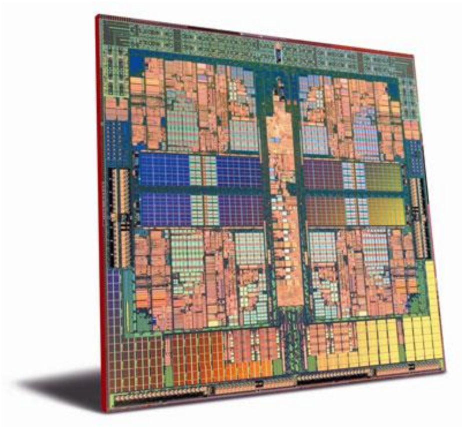
- Parallel computing steamrolled from behind by the inexorable advance of commodity technology
  - Economy of scale rules!
  - Commodity technology outperforms special machines
  - Massively-parallel machines replaced by *clusters* of ever-more powerful *commodity microprocessors*
  - Clusters: federates of standard pcs (MPI & OpenMP)

In this course we focus on widespread commodity parallel technology

More…

Supercomputer

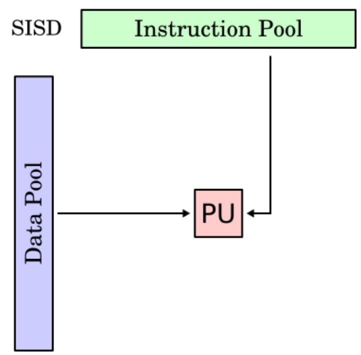
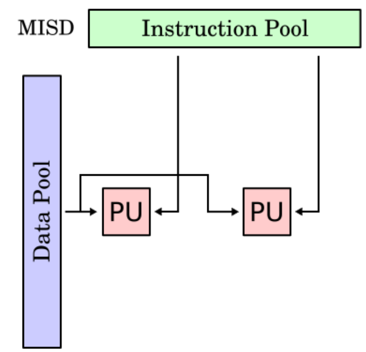
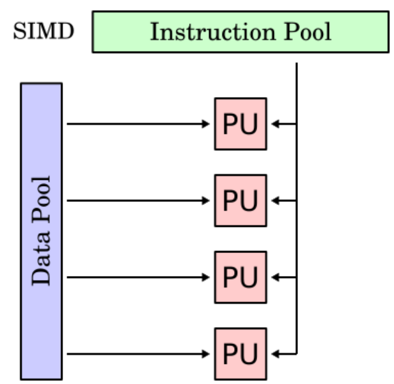
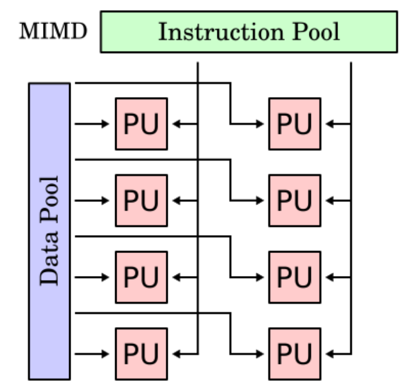
Cluster



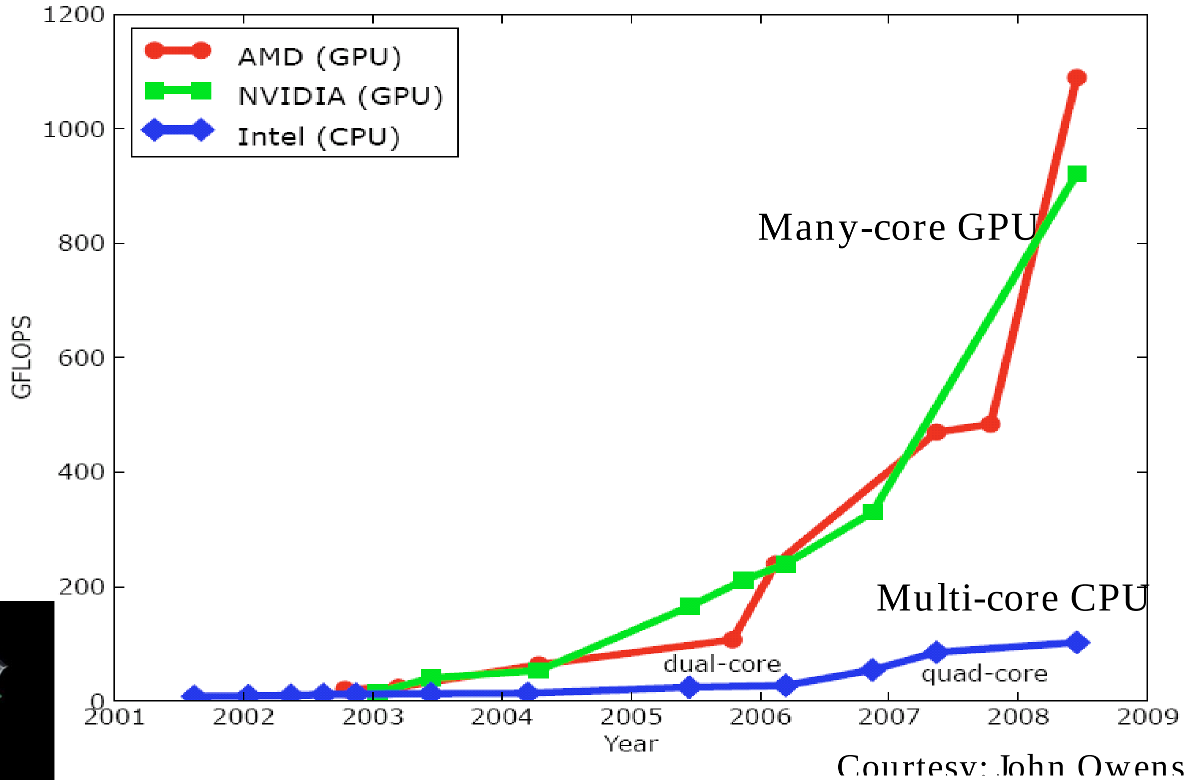
Multicore

But also a single core…

Flynn's taxonomy of architectures

<table>
<tr><th></th><th>Single Instruction</th><th>Multiple Instructions</th></tr>
<tr><td>Single Data</td><td>
</td><td>
</td></tr>
<tr><td>Multiple Data</td><td>
</td><td>
</td></tr>
</table>

Floating-Point Operations per Second for the CPU and GPU



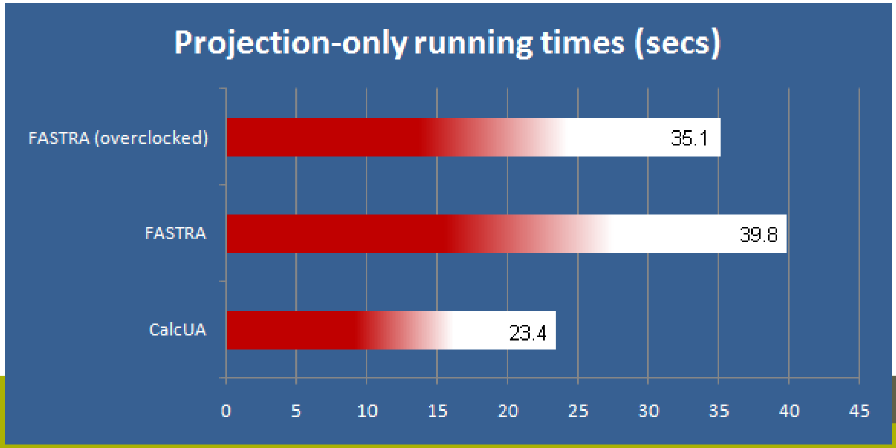
FASTRA at University of Antwerp

http://fastra.ua.ac.be

Collection of graphical cards

FASTRA 8 cards = 8x128 processors = 4000 euro

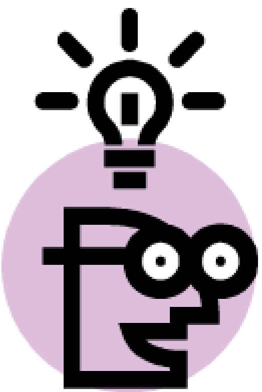
Similar performance as University's supercomputer (512 regular desktop PCs) that costed 3.5 million euro in 2005



Overview

Why use parallel systems

- Complete computation faster
- More (local) memory available



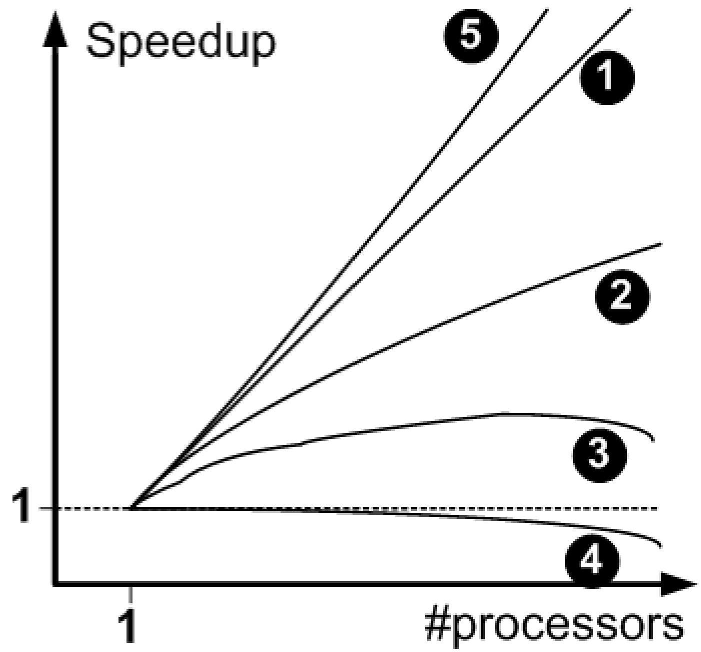
But… not simple!
Why?? Since a parallelizing compiler does not exist

Speedup

$$Speedup = \frac{T_{seq}}{T_{par}}$$

- Ideally: speedup = number of processors

Speedup i.f.o. processors



1) Ideal, linear speedup
2) Increasing, sub-linear speedup
3) Speedup with an optimal number of processors
4) No speedup
5) Super-linear speedup

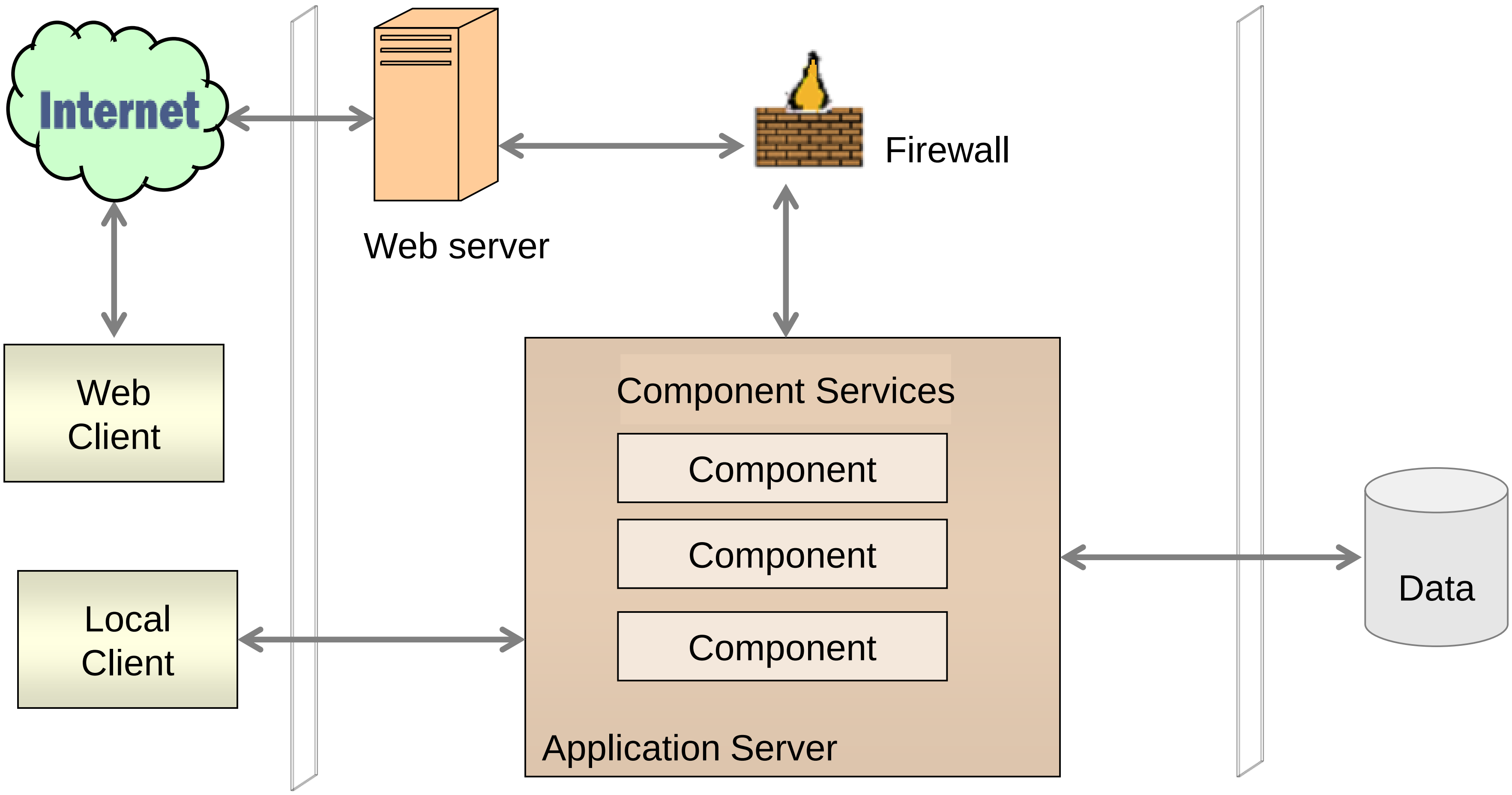
Parallel vs Distributed

OUR FOCUS

PPP 20-21

- Parallel computing: *provide performance*.
  - In terms of processing power or memory
  - To solve a single problem
  - Typically: frequent, reliable interaction, fine grained, low overhead, short execution time.
- Distributed computing: *provide convenience*.
  - In terms of availability, reliability and accessibility from many different locations
  - Typically: interactions infrequent, with heavier weight and assumed to be unreliable, coarse grained, much overhead and long uptime.

Example: Distributed 3rd generation web application



Overview

1. Definition
2. Why?
3. **Parallel compiler?**
4. Parallel architectures
5. Parallel Processing Paradigms
   - Multi-threading.
   - Message-passing.
6. End notes

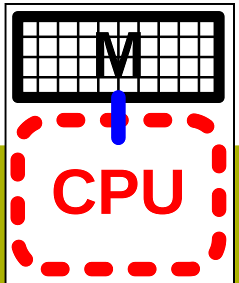
Sequential programming world

- Understand this to port it to the parallel world
- Write Once, Compile Everywhere
  - C, C++, Pascal, Modula-2, …
- Compile Once, Run Everywhere
  - Java, C#
- Sequential programming is close to our algorithmic thinking (> 2 GL).
- Von Neumann architecture provides useful abstraction

The Random Access Machine

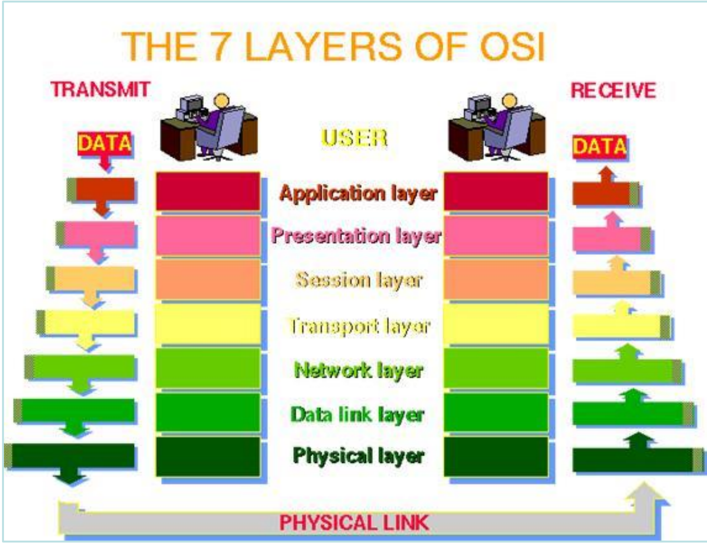
PPP 58-60

- Sequential computer = *device with an instruction execution unit and unbounded memory*.
  - Memory stores program instructions and data.
  - Any memory location can be referenced in ‘unit’ time
  - The instruction unit fetches and executes an instruction every cycle and proceeds to the next instruction.
- Today’s computers depart from RAM, but function *as if* they match this model.
- Model guides algorithm design.
  - Programs do not perform well on e.g. vector machines.



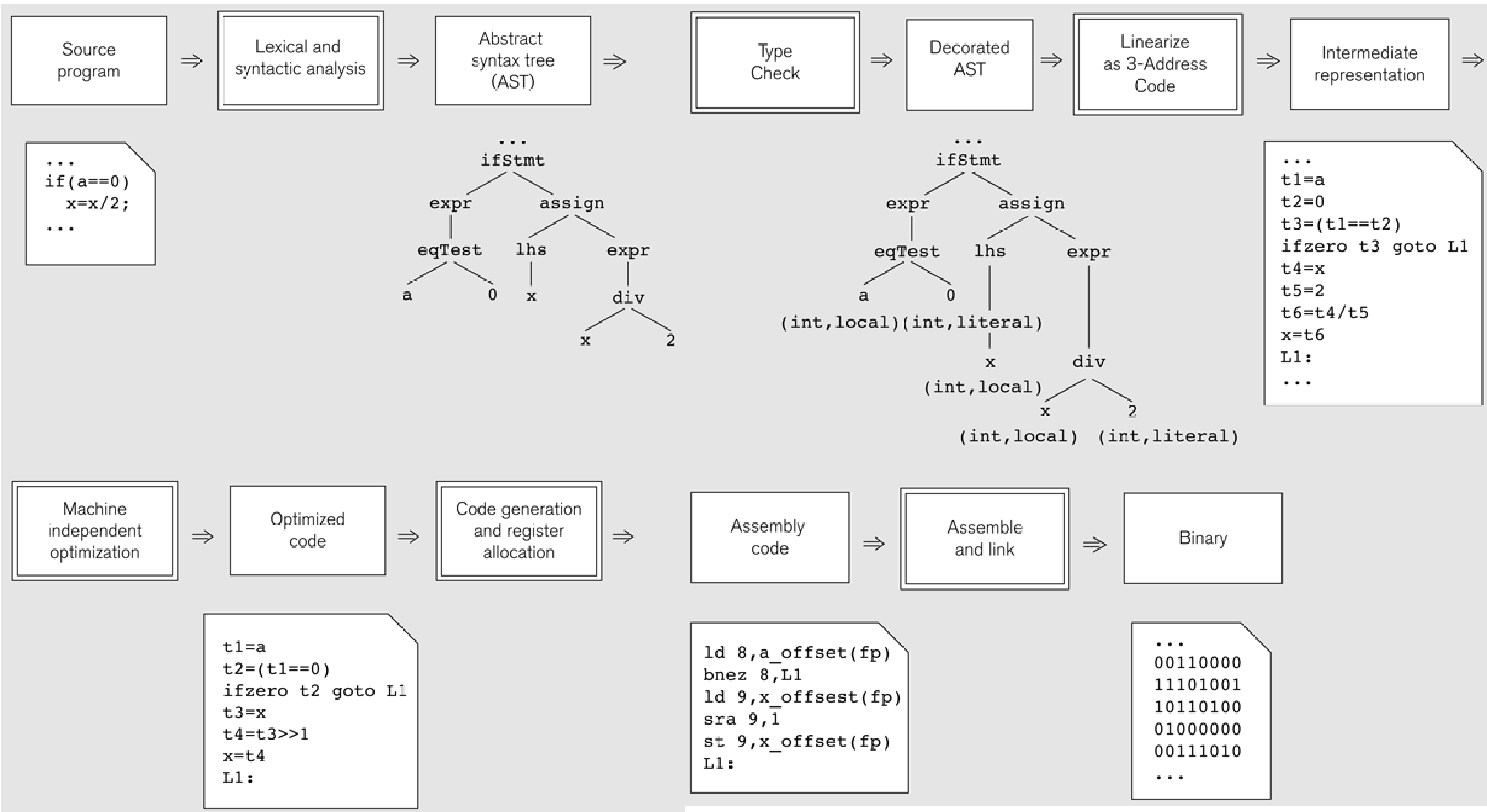
Software success relies on *abstraction & user transparency*

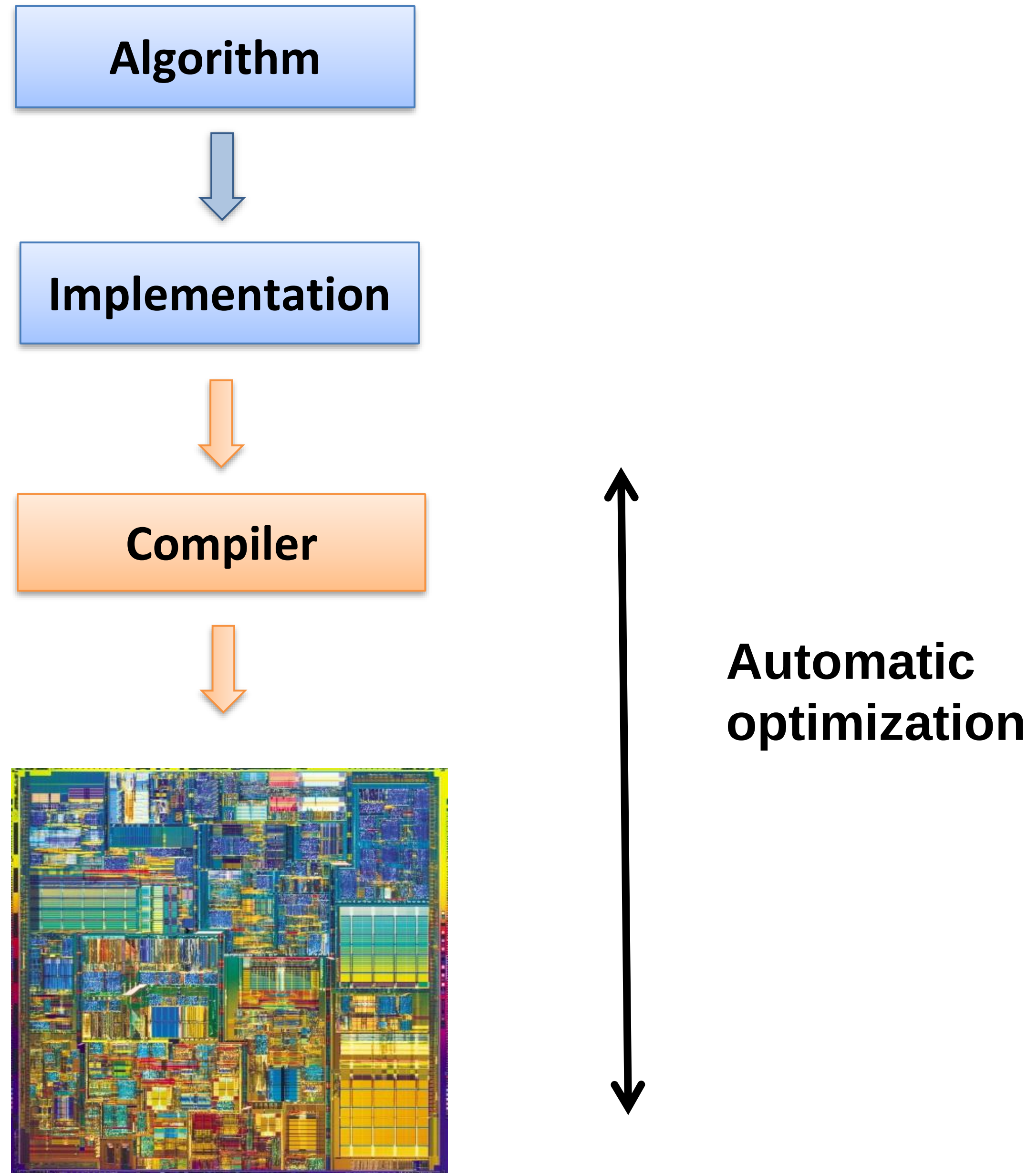
- A library offers a service and hides implementation details for you.
- Layered approaches such as the OSI model in telecommunication
- 3rd generation language => assembler => machine code => machine
  - Language hides hardware details
- Software engineering concepts



Generic Compilation Process

PPP 22-25




Algorithm
Implementation
Compiler
Automatic
optimization

Parallel compilers

PPP 22-23

- *Goal*: automatically compile sequential program into an efficient parallel program that does the same thing.
  - *Programmers would not have to learn special parallel constructs*
- *Is a dream that seems beyond reach…*
  - Many user-defined algorithms contain data dependencies that prevent efficient parallelization.
  - Automatic dependency analysis and algorithm transformation: still in their infancy, far from optimal. Real breakthrough not expected in the near future.
  - For efficient parallel programs, a simple hardware model such as the RAM model does not work.

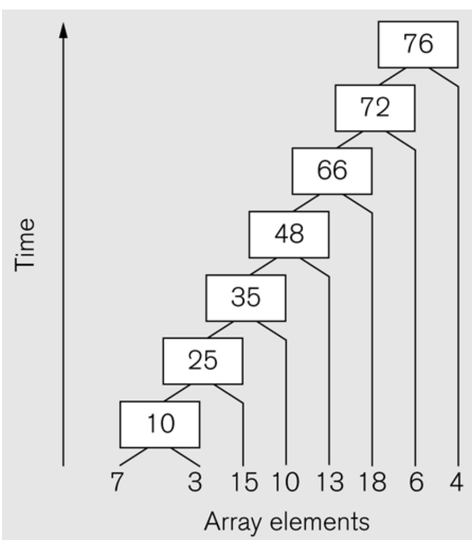
Example: Iterative Sum

PPP 23

n data values x_0, …, x_n in array *x*

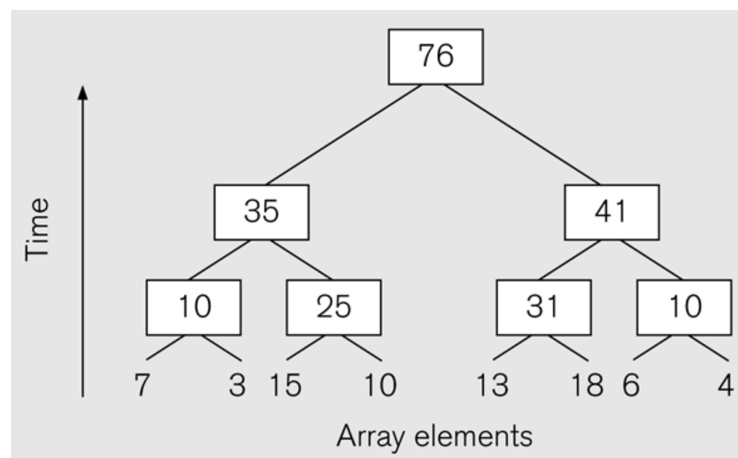
```
sum=0;
for (int i=0;i<n;i++)
  sum+=x[i];
```

- Parallelism? Independent computations needed.



By associativity of sum

Can this be done by a compiler??



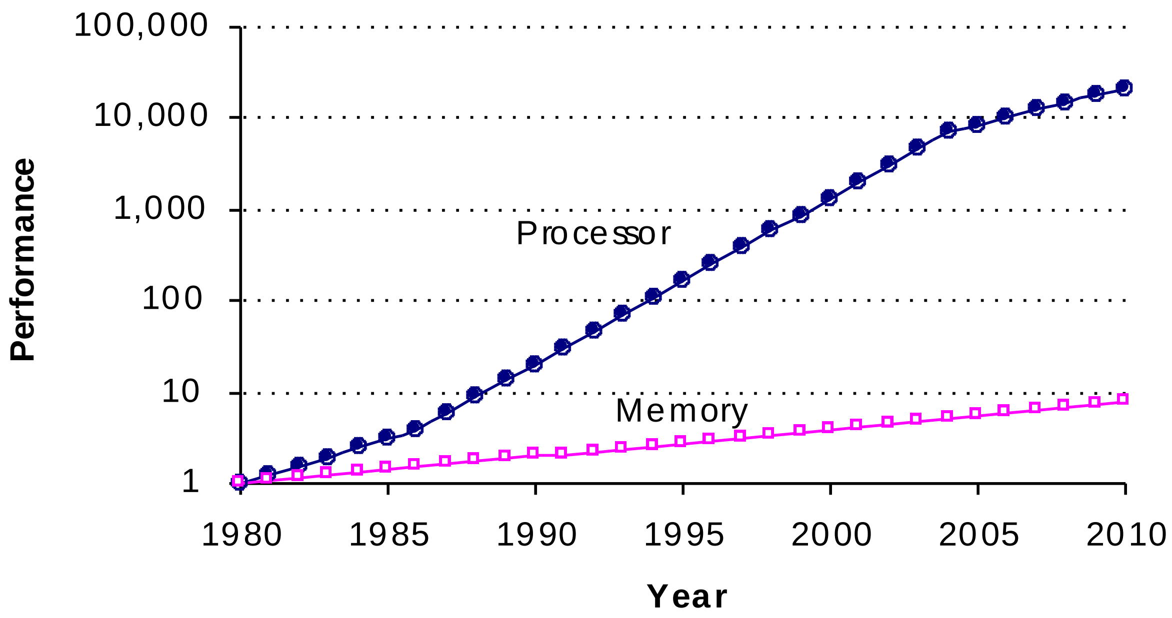
Overview

1. Definition
2. Why?
3. Parallel compiler?
4. **Parallel architectures**
5. Parallel Processing Paradigms
   - Multi-threading.
   - Message-passing.
6. End notes

PRAM: Parallel Random Access Machine

- Global memory of unbounded size that is uniformly accessible to all processors
- It fails by misrepresenting memory behavior.
  - Impossible to realize the unit-time single memory image
- Cf: memory is now the bottleneck, also in sequential computers

Memory has become the main bottleneck…



Pentium chip devoted about 10% of chip area to cache,
Pentium 4 devotes about 50%

Memory speed lags behind processor speed...

Memory Latency λ

PPP 63

- Memory Latency = *delay required to make a memory reference*.
- Relative to processor's local memory latency, ≈ unit time ≈ one word per instruction
  - Variable, due to cache mechanisms etc
- **Locality Rule**: *Fast programs maximize number of local memory references*.
  - Sometimes it is better to recalculate globals locally (e.g. random number)

1. Shared Memory

Natural extension of sequential computer: all memory can be referenced (single address space). Hardware ensures memory coherence.

- Easier to use
  - Through multi-threading
- Easier to create faulty programs
  - Race conditions
- More difficult to debug
  - Intertwining of threads is implicit
- Easier to create inefficient programs
  - Easy to make non-local references

2. Distributed Memory

Processors can only access their own memory and communicate through messages.

- Requires the least hardware support.
- Easier to debug.
  - Interactions happens in well-defined program parts
  - The process is in control of its memory!
- Cumbersome communication protocol is needed
  - Remote data cannot be accessed directly, only via request.

3. Fine-grain parallelism

Needs many small pieces that can be processed in parallel.

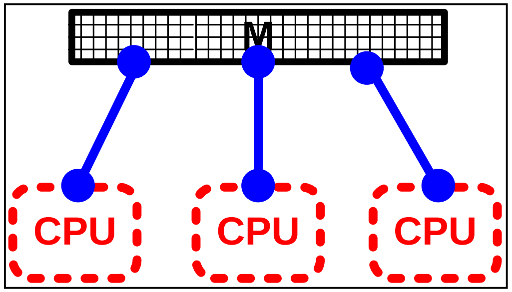
- Enormous processing power: vector processors, GPUs
- No single programming model
  - OpenCL versus vectorization
- Harder to program
- Independence & locality & high computational intensity needed to reach peak performance.

Overview

1. Definition
2. Why?
3. Parallel compiler?
4. Parallel architectures
5. **Parallel Processing Paradigms**
   - **Multi-threading.**
   - **Message-passing.**
6. End notes

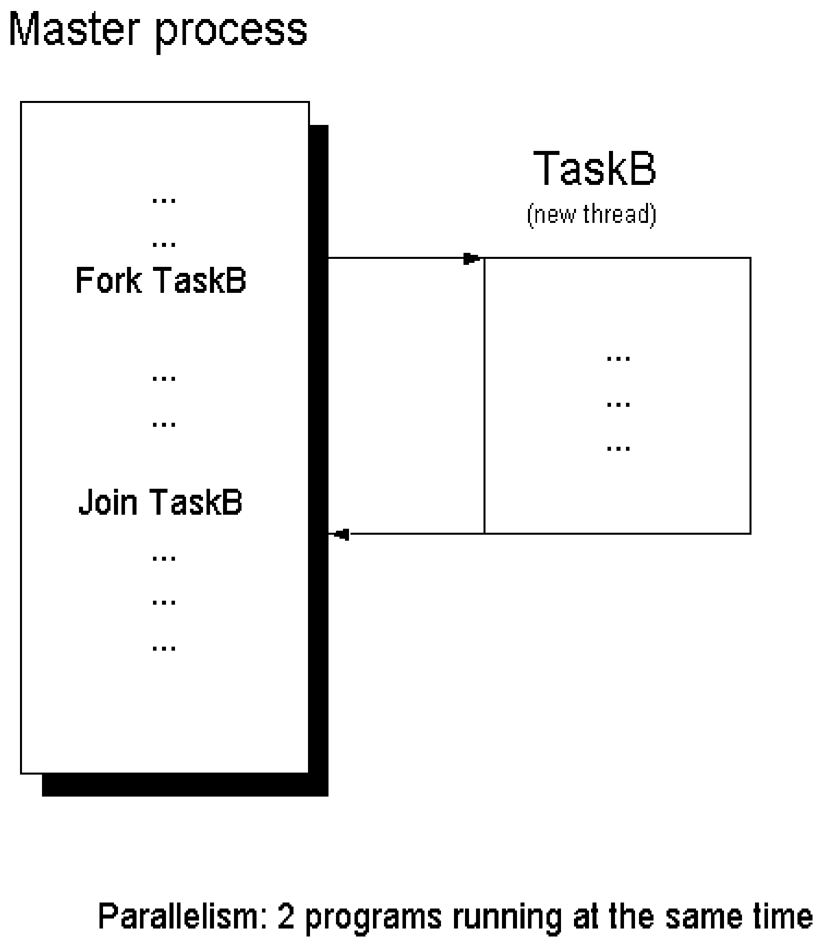
1. Multithreading

- One process is split into separate threads,
  - executing a different sequence of instructions
  - having access to the same memory
- = ***Shared address space approach***



Multi-threading primitives

- Fork & join



The Java Thread Class

PPP 202

```
public synchronized void start()
```

- Starts this `Thread` and returns immediately after invoking the `run()` method.
- Throws `IllegalThreadStateException` if the thread was already started.

```
public void run()
```

- The body of this `Thread`, which is invoked after the thread is started.

```
public final synchronized void join(long millis)
  throws InterruptedException
```

- Waits for this `Thread` to die. A timeout in milliseconds can be specified, with a timeout of 0 milliseconds indicating that the thread will wait forever.

```
public static void yield()
```

- Causes the currently executing `Thread` object to yield the processor so that some other runnable Thread can be scheduled.

```
public final int getPriority()
```

- Returns the thread's priority.

```
public final void setPriority(int newPriority)
```

- Sets the thread's priority.

LINK 2

Thread creation

```
class PrimeThread extends Thread
{
    long minPrime;
PrimeThread(long minPrime) {
    this.minPrime = minPrime;
}
public void run() {
// compute primes larger
// than minPrime
    . . .
}
}
```

```
PrimeThread p = new PrimeThread(143);
 p.start();
```

```
class PrimeRun implements Runnable
{
long minPrime;
PrimeRun(long minPrime) {
this.minPrime = minPrime;
}
public void run() {
// compute primes larger
// than minPrime
. . .
}
}
```

```
PrimeRun p = new PrimeRun(143);
new Thread(p).start();
```

Example: Counting 3s

PPP 29

n data values $x_0, \ldots, x_n$ in array *array*

```
count=0;
for (int i=0;i<array.length;i++)
  if (array[i] == 3)
    count++;
```

- Parallelism? Yes.
- Multithreaded solution: divide counting

Multithreaded Counting 3s

Note: this program is faulty
Will be discussed

```
count=0;
Thread[] threads = new Thread[nbrThreads];
for(int t=0;t<nbrThreads;t++){
   final int T = t;
   threads[t] = new Thread(){
        public void run(){
             int length_per_thread=array.length/ nbrThreads;
             int start=T*length_per_thread;
             for(int i=start;i<start+length_per_thread; i++)
                  if (array[i] == 3)
                      count++;
        }
   };
   threads[t].start();
}
// wait until all threads have finished
for(int t=0;t<nbrThreads;t++)
   try {
      threads[t].join();
   } catch (InterruptedException e) {}
```

Some advanced java…

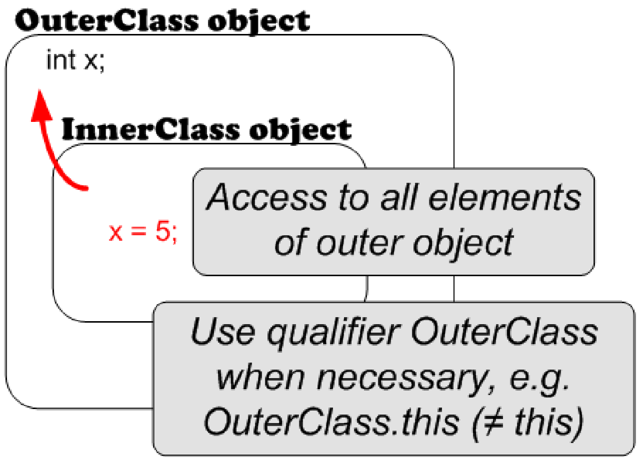
Inner class

```
class OuterClass{

   class InnerClass{
      ...
   }

   non-static method(){
      InnerClass a = new InnerClass();
   }
}
```

Definition inside a class

Object instantiation only in outer class => you have to create an outer object

Invisible outside of class



Anonymous class

Is also an inner class

```
final int I = i;
A a = new A("parameter"){

   @Override
   void method1(){
      ...
   }

   void method2(){
      int x = I;
   }
}
```

No specific constructor

Override methods or add new methods

Only access to final local variables of enclosing method

Create object of new subclass which is not given a separate name

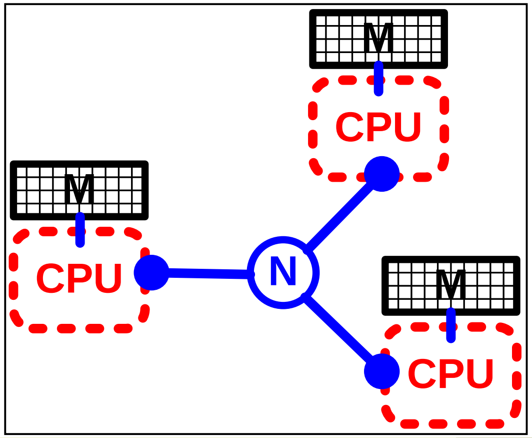
Counting 3s: experiments

On a dual core processor

```
Counting 3s in an array of 1000 elements and 4 threads:
 * Seq  : counted 100 3s in 234us
 * Par 1: counted 100 3s in 3ms 615us


Counting 3s in an array of 40000000 elements and 4 threads:
 * Seq  : counted 4000894 3s in 147ms
 * Par 1: counted 3371515 3s in 109ms
```

2. Message-passing

- Different processes
  - Communicate through messages
  - Got their own dedicated memory (and got full control over it)
- ***=Message-passing approach***



Messages...

- The ability to send and receive messages is all we need

  - void Send(message, destination)
  - char[] Receive(source)
  - boolean IsMessage(source)

- But... we also want performance!
  - More functions will be provided

Message-passing Counting 3s

```
int count3s_master(int[] array){
   int length_per_slave=array.length/nbr_slaves;
   for (slave: slaves)
      send integer subarray of length length_per_slave to slave;

   int sum=0;
   for (slave: slaves)
      sum+= receive integer from slave;
   return sum;
}

int count3s_slave(){
   int[] array = receive array from master;
   count=0;
   for (int i=0;i<array.length;i++)
      if (array[i] == 3)
         count++;
   send count  to master;
}
```

pseudo code

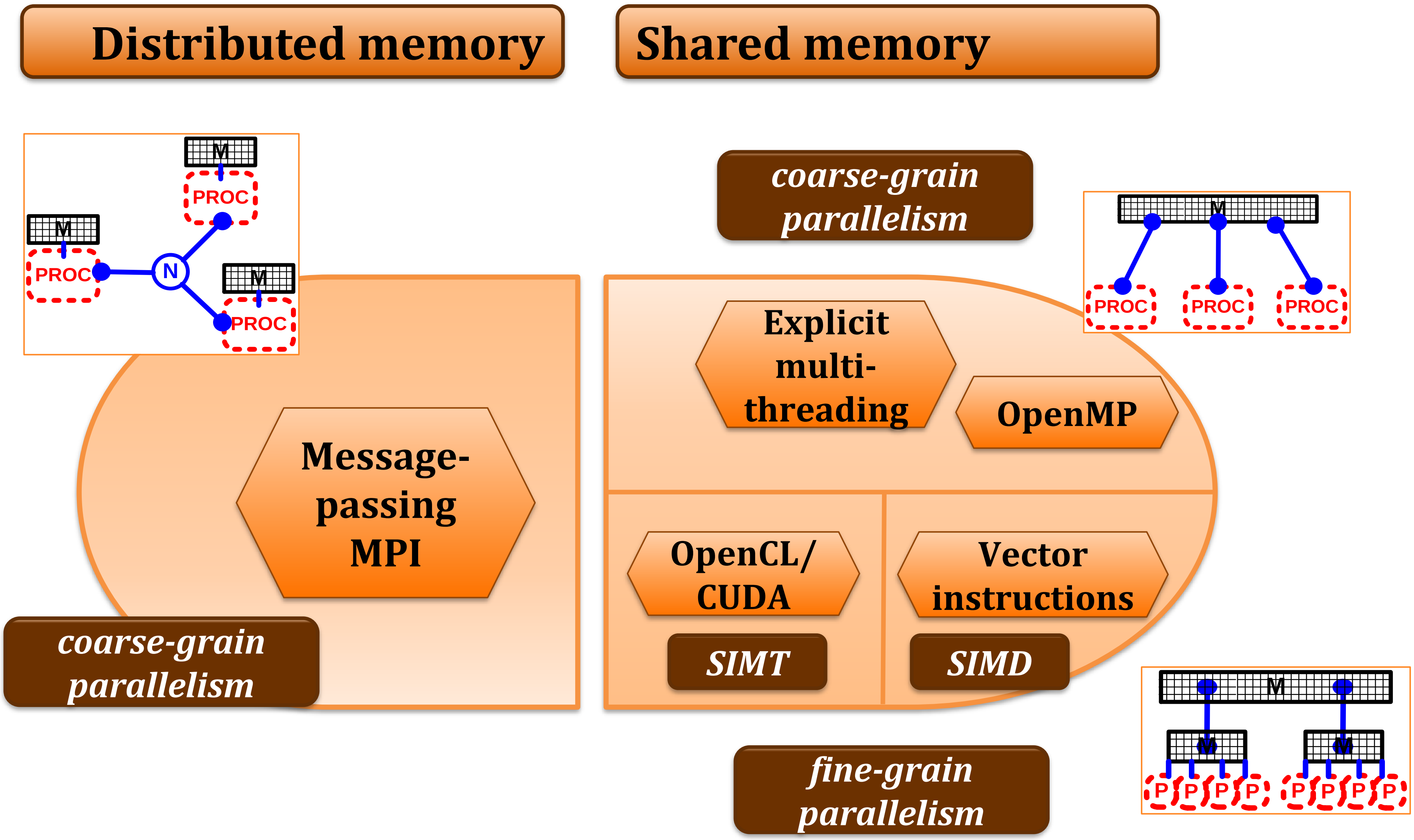
= sequential program!

Focus on low-level approaches

- MPI, multi-threading & OpenCL: low-level primitives
- Higher-level alternatives exist, but have not proven to be successfull for a wide variety of parallelization problems
  - Fail to hide low-level aspects

- We'll focus on the 3 main low-level approaches
- You will be able to learn/use the other approaches yourself

Parallel Systems



Course Overview

Parallel paradigms

Multi-threaded

- *architectures*: PRAM & cache coherence
- *synchronization*:
  - critical sections
  - conditional wait
  - composite constructs

⇔

Message-passing

- *architectures*:
  - communication networks
- *communication*: MPI
  - specific functions for specific structures!

GPU programming

- *clear strategy:*
  - transparent, powerful, dedicated hardware
  - write efficient programs taking structure into account

Parallel algorithms

Matrix Algorithms

- *structured mathematical operations:*
  - optimize for memory
  - structured communication and computation

Tree Search

- *advanced techniques:*
  - dynamic load balancing
  - termination detection

Sort Algorithms

- specific solutions are necessary
- inherent parallelism in sequential algorithm is not enough

Performance Analysis

- *understand overheads*
- *predict performance*
- *scalability*

Philosophy and conclusions in first & last session!

Overview

1. Definition
2. Why?
3. Parallel compiler?
4. Parallel architectures
5. Parallel Processing Paradigms
   - Multi-threading.
   - Message-passing.
6. **End notes**

The goals of this course

PPP 39-41

Learn to write good parallel programs, which

- Are **correct**
- Achieve **good performance**
- Are **scalable** to large numbers of processors
- Are **portable** across a wide variety of parallel platforms.
- Are **generic** for a broad class of problems.

To attain goals…

- Master low-level and high-level IT skills
  - Low-level: hardware and system
  - High-level: Software engineering
- Combine knowledge and inventivity
- Approach: look at it as a user who wants to know as little as possible

Instruction-level Parallelism (ILP): vector instructions

- X86 architecture: MMX or SSE instructions will perform an instruction on 4/8/16 floats (special registers) at once (SIMD)
- Example: compute difference between 2 images
  - Compute 8 pixels at once
- Fine-grain parallelism like GPUs
  - New Intel Xeon Phi architecture also requires vector computation
- Program: assembler instructions or special C-extensions
  - First move data to special registers
  - Not so easy!

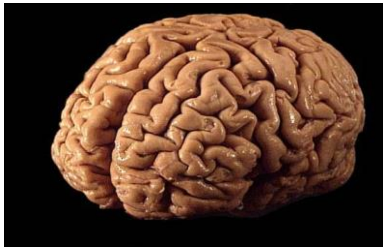
In practice, an ILP of 4 seems the maximum

Open questions...

- Automatic parallelizing compiler?
- Why is there no universal method to write parallel programs?
- How much do we need about the hardware to write good parallel programs?
  - Knowledge yield significant performance improvements
  - For portability, machine details should be ignored…
- How to make parallelism a success story?

Intelligence & Parallelism

- A lot of researchers, philosophers think that intelligence is caused by the highly parallel structure of our brain.
  - Only parallelism can give intelligence?



- I do not agree, every parallel program can be executed sequentially, if necessary by adding indeterminism

Parallelism $\not\Rightarrow$ **Intelligence**

Intelligence & Parallelism II

- On the other hand: intelligence makes efficient parallel processing possible.
  - Insight into all HW & SW aspects is necessary
  - Automated parallel processing only by an intelligent computer

Intelligence ➡ **Parallelism**