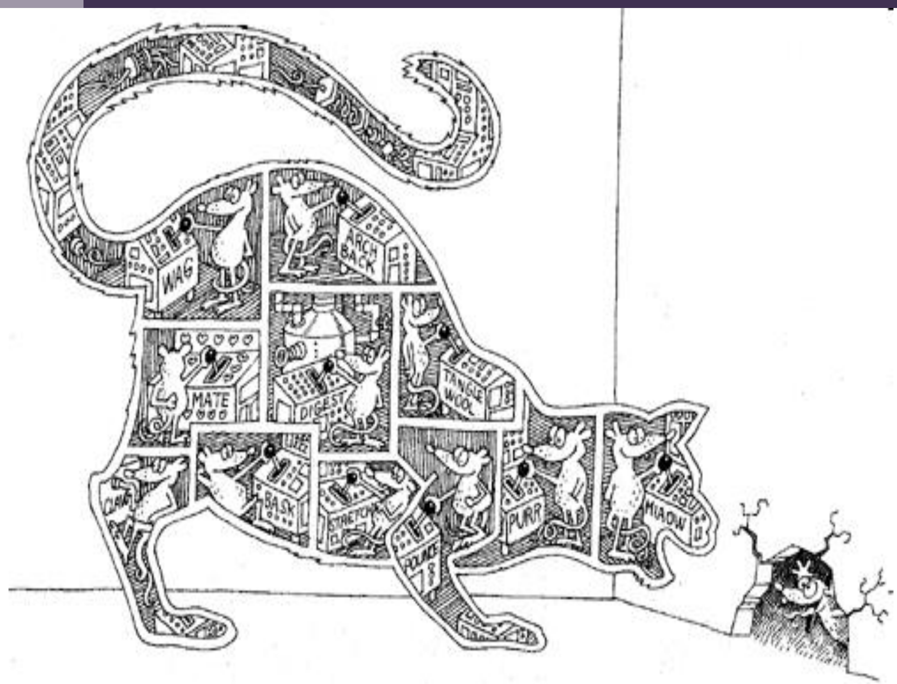
Department of Computer Science
University of Bristol

COMS20001 - Concurrent Computing

www.cs.bris.ac.uk/Teaching/Resources/COMS20001



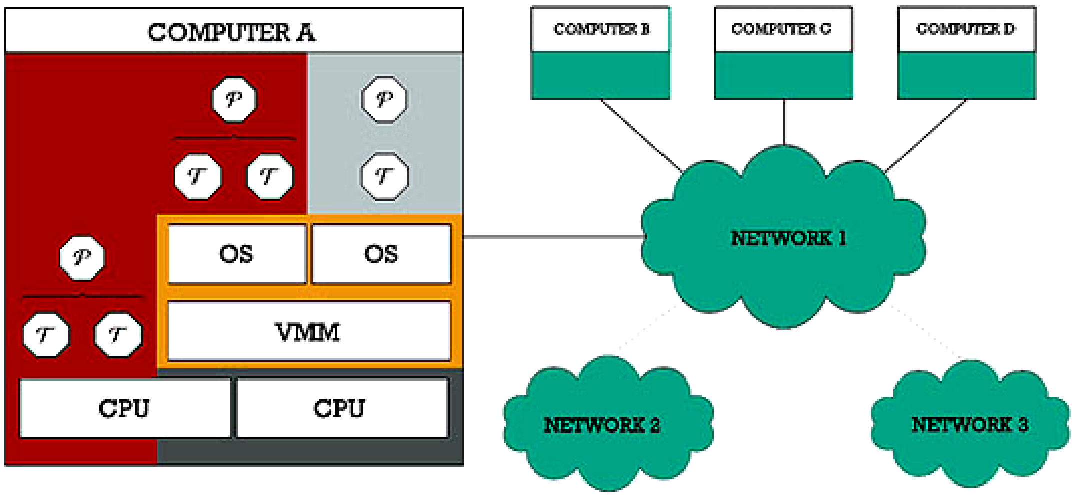
Lecture 01

Introduction and History

Sion Hannuna | sh1670@bristol.ac.uk
Dan Page | daniel.page@bristol.ac.uk

41 Slides

Unit Components of COMS20001



Year 1 Computer Architecture
Year 1 Programming & Algorithms

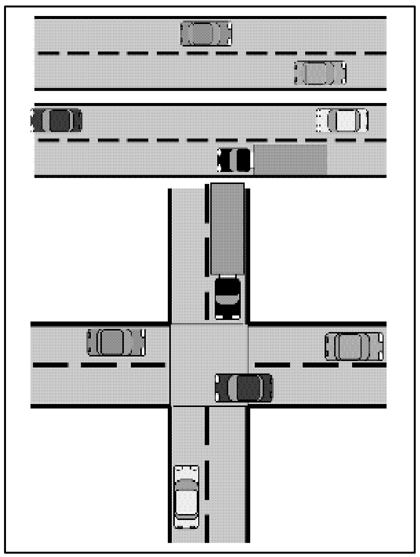
COMS20001 CONCURRENT COMPUTING:
PART 1: Concurrency Principles and Reasoning
PART 2: Concurrent Programming
PART 3: Operating Systems
PART 4: Interconnection Networks

Weeks 01-12	**Sion Hannuna**	Concepts of Concurrent Computing
	Tilo Burghardt	Concurrent Programming in XC Formal Reasoning about Concurrency
13-18	**Dan Page**	Operating Systems
19-24	**Dan Page**	Interconnection Networks

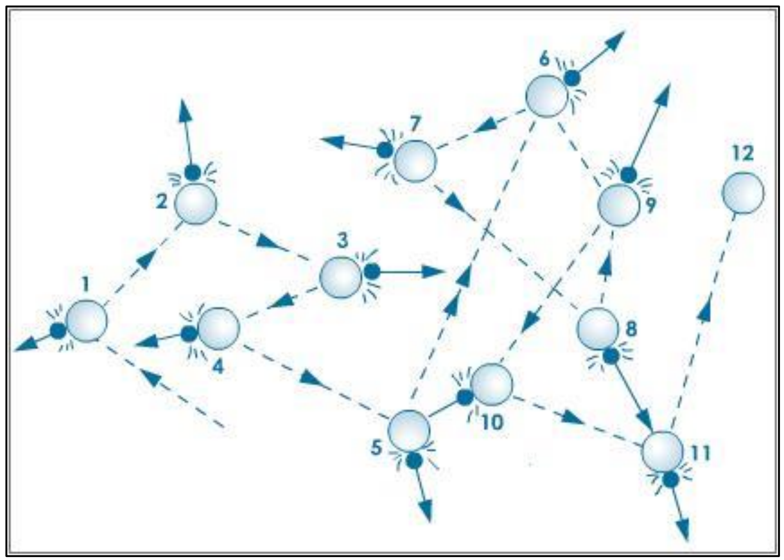
What exactly is meant by **Concurrency?**

Concurrency is a property given to systems (e.g. software and/or hardware) in which **multiple sequential processes** are running **simultaneously, interacting** with each other.

SION



a traffic network



Brownian particle motion

the human brain

Examples of "naturally" concurrent systems

What are the implications of **Concurrency?**

Theoretical/Conceptual aspects and models

SION

- **History!**
- **Models of Process Communication:**
  Shared memory, synchronous & asynchronous channels
  Deadlocks
- **Paradigms of Hardware Parallelism**
  Flynn's Taxonomy: SISD, SIMD, MISD, MIMD
- **Paradigms of Logical Parallelism**
  Geometric Parallelisation
  Parallelisation via Farming
  Algorithmic Parallelisation
- **Performance issues**

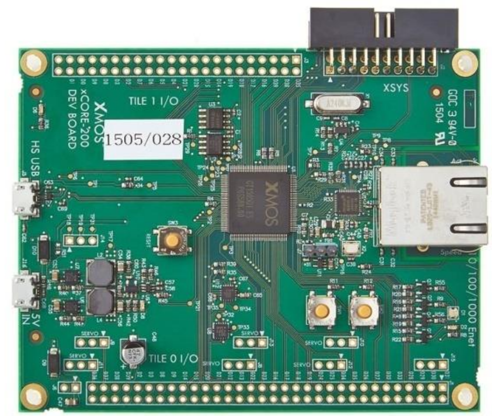
Concurrent Programming in XC on xCore200

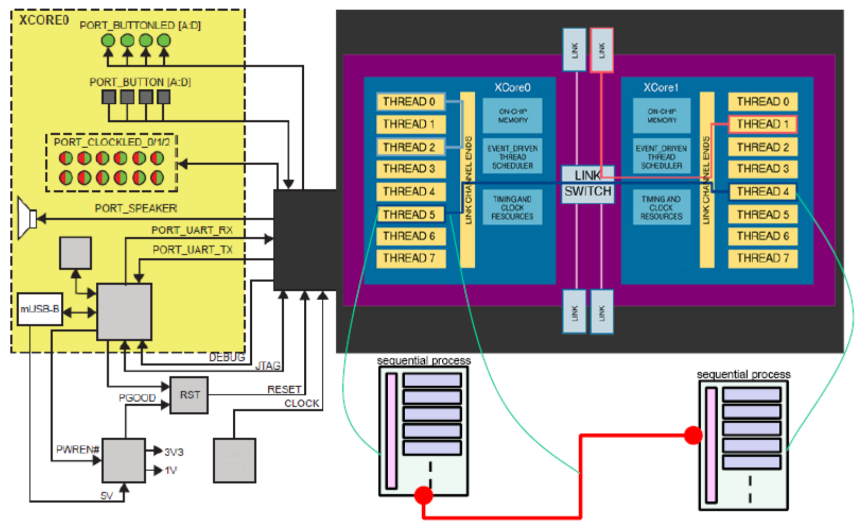
```
// process P1 (fragment)
void processOne(chanend ch) {
  point A[4]={...}; //init A
  point B[4]={...}; //init B
  int   C[4][4]={...}; //init C

  // send data to process two
  for (int i = 0; i<4; i++)
    ch <: A[i];
  for (int i = 0; i<2; i++)
    ch <: B[i];

  // calculate some distances
  for (int i = 0; i<4; i++)
    for (int j = 2; j<4; j++)
      C[i][j] = dist(A[i], B[j]);

  // receive results
  for (int i = 0; i<4; i++)
    for (int j = 0; j<2; j++)
      ch :> C[i][j];

  ... //output result C
}
```

```
// process P2 (fragment)
void processTwo(chanend ch) {
  point A[4]; //allocate local memory
  point B[2];
  int   C[4][2]={...};

  // receive data from process one
  for (int i = 0; i<4; i++)
    ch :> A[i];
  for (int i = 0; i<2; i++)
    ch :> B[i];

  // calculate some distances
  for (int i = 0; i<4; i++)
    for (int j = 0; j<2; j++)
      C[i][j] = dist(A[i], B[j]);

  // send results
  for (int i = 0; i<4; i++)
    for (int j = 0; j<2; j++)
      ch <: C[i][j];
}
```

channel ch

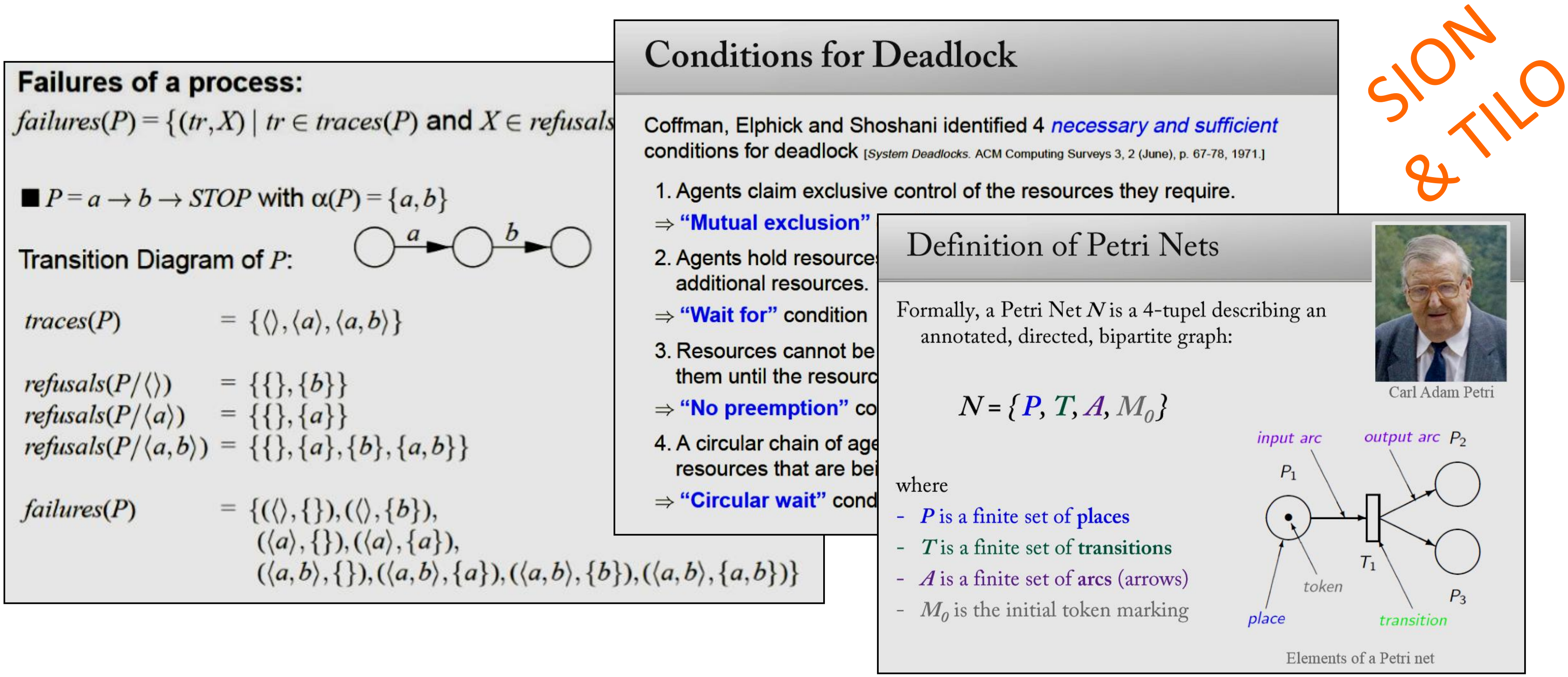
SION





- direct control of (parallel) hardware platform
- no operating system
- first hand experience of concurrency effects

Modelling with & Reasoning about Concurrent Systems

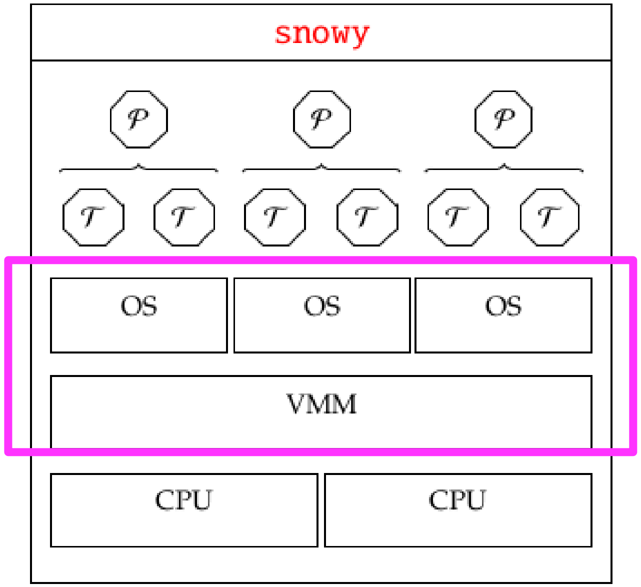


- formal modelling of concurrent processes using CSP basics
- reasoning about communicating sequential processes (e.g. safety, liveness, deadlock, divergence)
- understanding the impact of design choices on program properties (e.g. fairness, starvation, overheads)

Operating System Concurrency

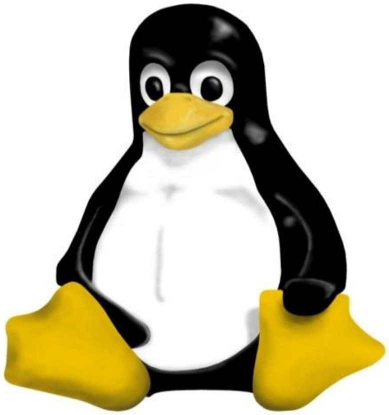
Dan

The OS is the "Middle-man" of a computing system. It coordinates concurrent operations on a hardware platform.



There are many types of concurrent operations which are managed by an operating system, including:

- Process management
- Memory management
- Device management



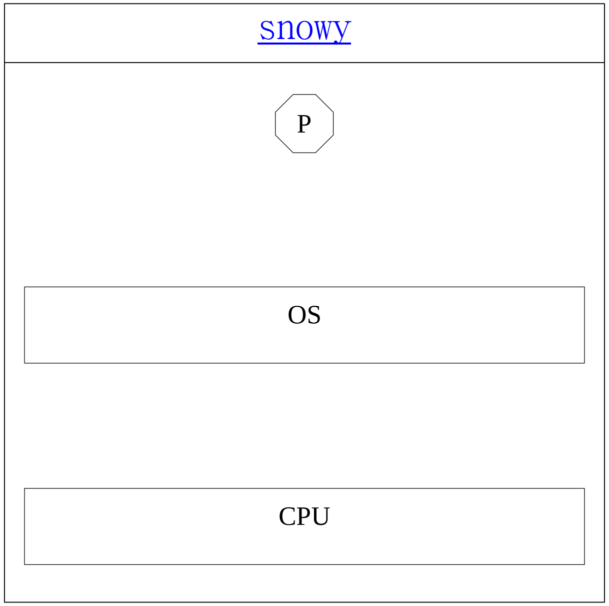
Operating System Concurrency

Dan

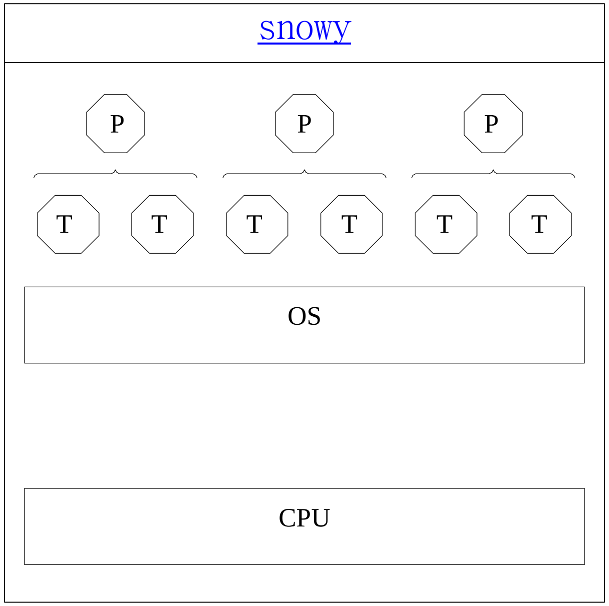
In the OS part of the course, we will:

- Look at what concurrency resources are provided by hardware
- Investigate how these are managed by an OS to provide concurrent execution
- Build a multi-tasking OS

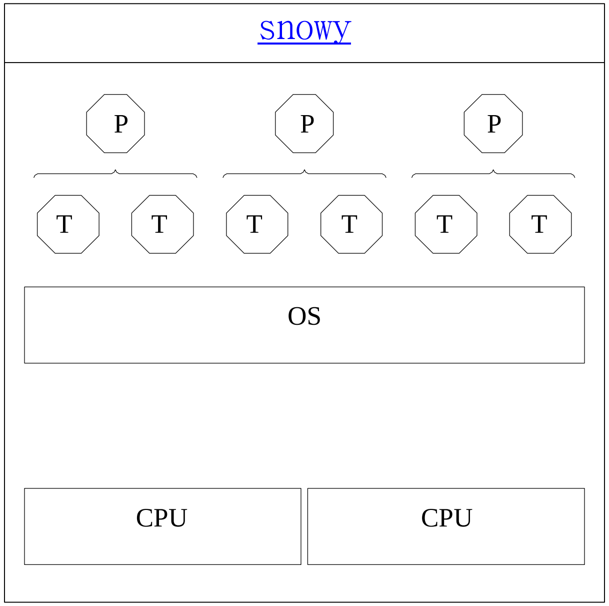
Concept of concurrent computing exists at various scales



Concept of concurrent computing exists at various scales

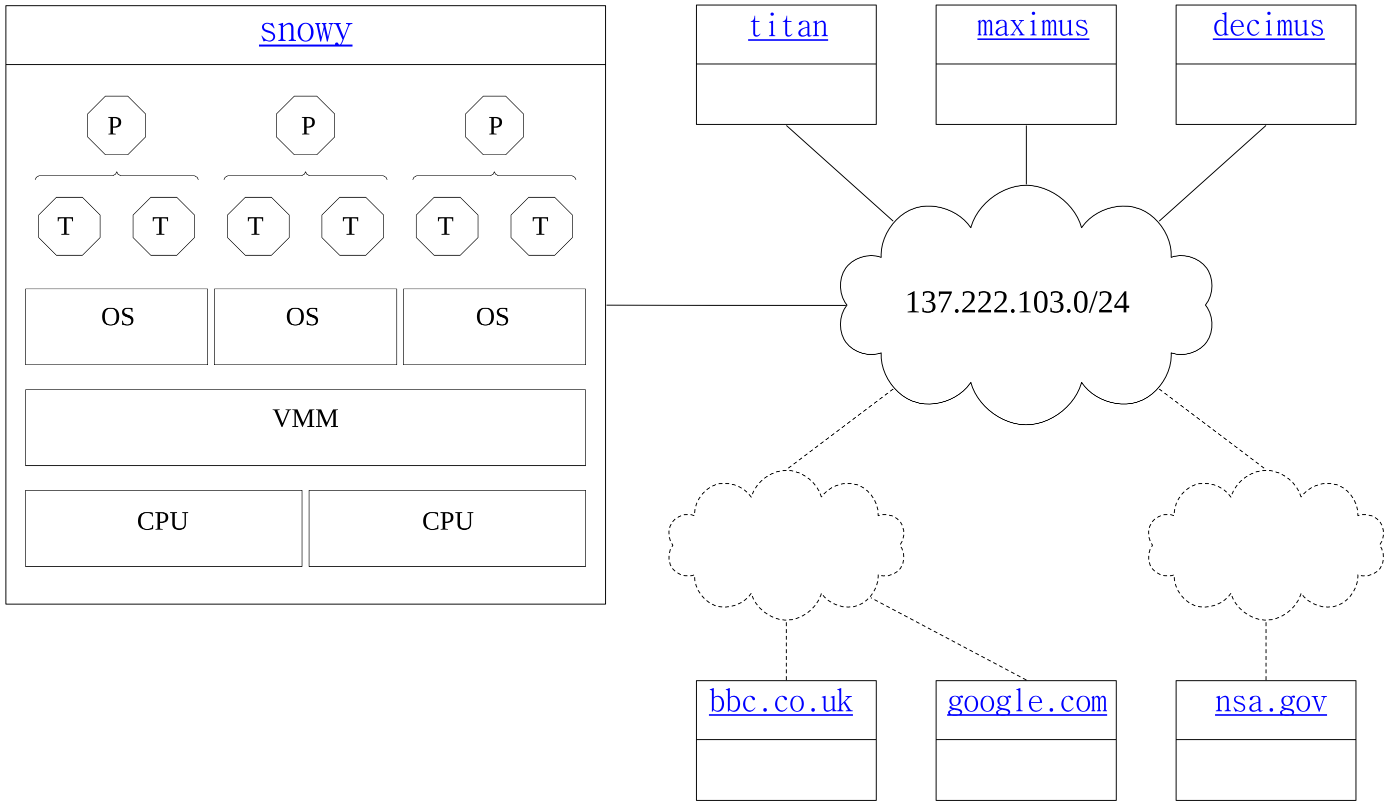


Concept of concurrent computing exists at various scales



Concept of concurrent computing exists at various scales

Dan



Bottom-up Understanding of Network Environments

Dan

1. link layer
   transmission, modulation, multiplexing, metrics
   802.3 (Ethernet), 802.11 (WiFi)
2. internet layer
   addressing, forwarding, fragmentation
   IP, ICMP, DHCP, ARP
3. transport layer
   connection management, routing, flow and congestion control
   UDP, TCP, NAT, DNS
4. application layer

... while emphasising general concepts as applied in specific technologies (using a running example) ...

... or, explain how the Internet works (and doesn't) so you can

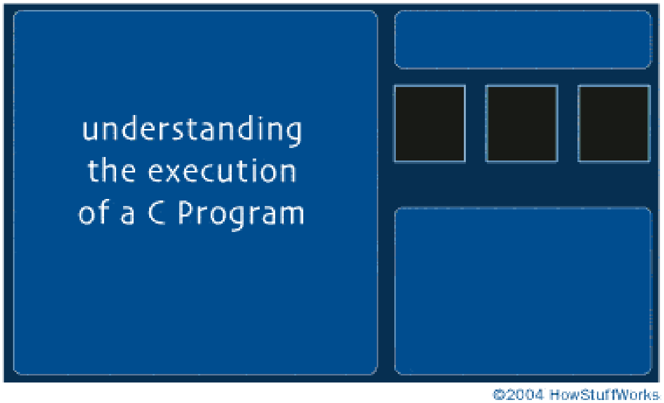
1. make effective use of existing technologies, but also
2. understand and apply similar concepts in next-generation technologies (of your own design).

So far on your degree programme…

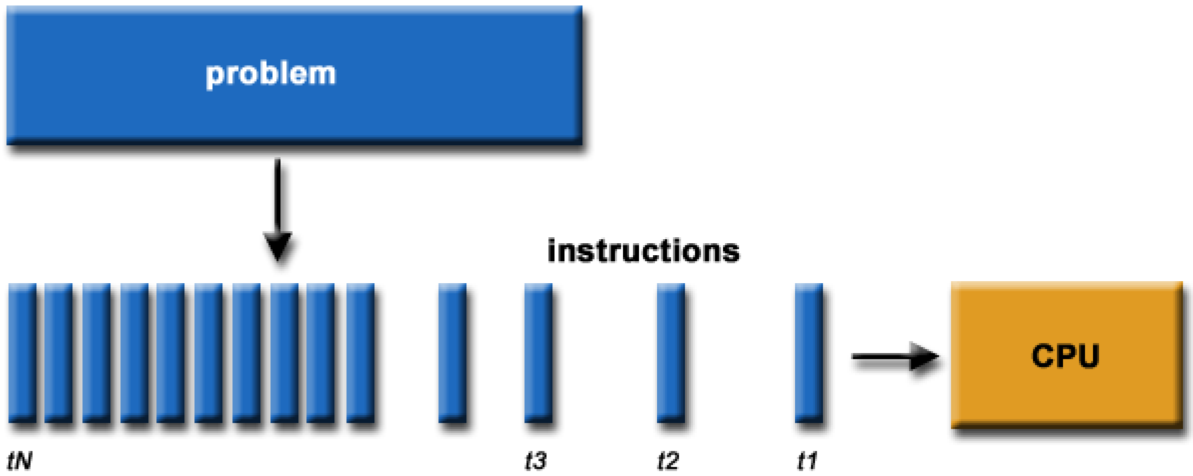
- …you learned about principles and programming basics…
  - C (procedural programming),
  - Java (object oriented programming) and
  - Haskell (functional programming)

- …however, you essentially wrote ***sequential*** programs…
  - where programs were **deterministic** entities (single, defined sequence of state changes)
  - where **memory** (code & data) was accessed **sequentially** (instruction by instruction) and **available globally** (apart from scope)

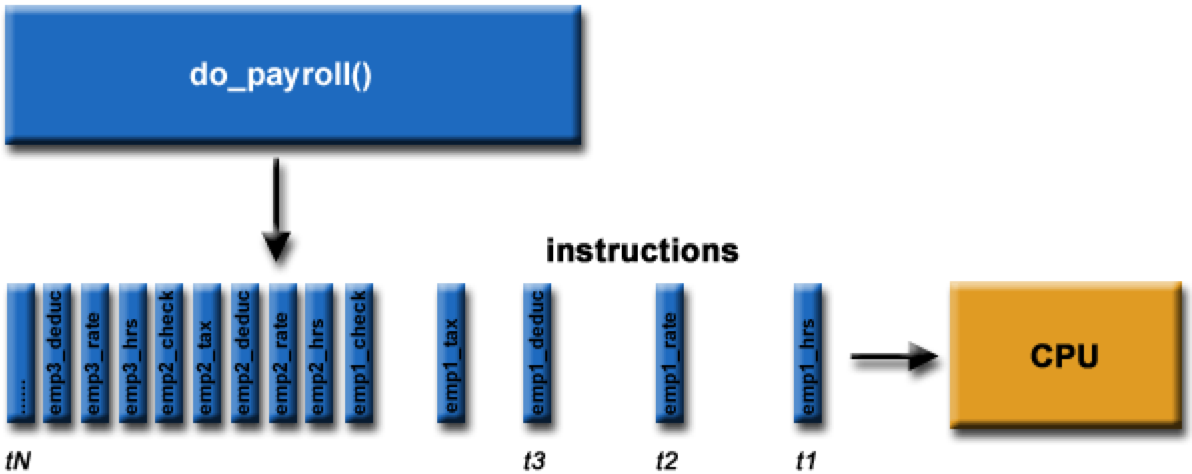
IN THIS PART OF COMS20001…
What happens if multiple programs are run <u>simultaneously</u> on multiple CPUs, with or without shared resources (memory banks etc)?



So essentially, we want to go from this...

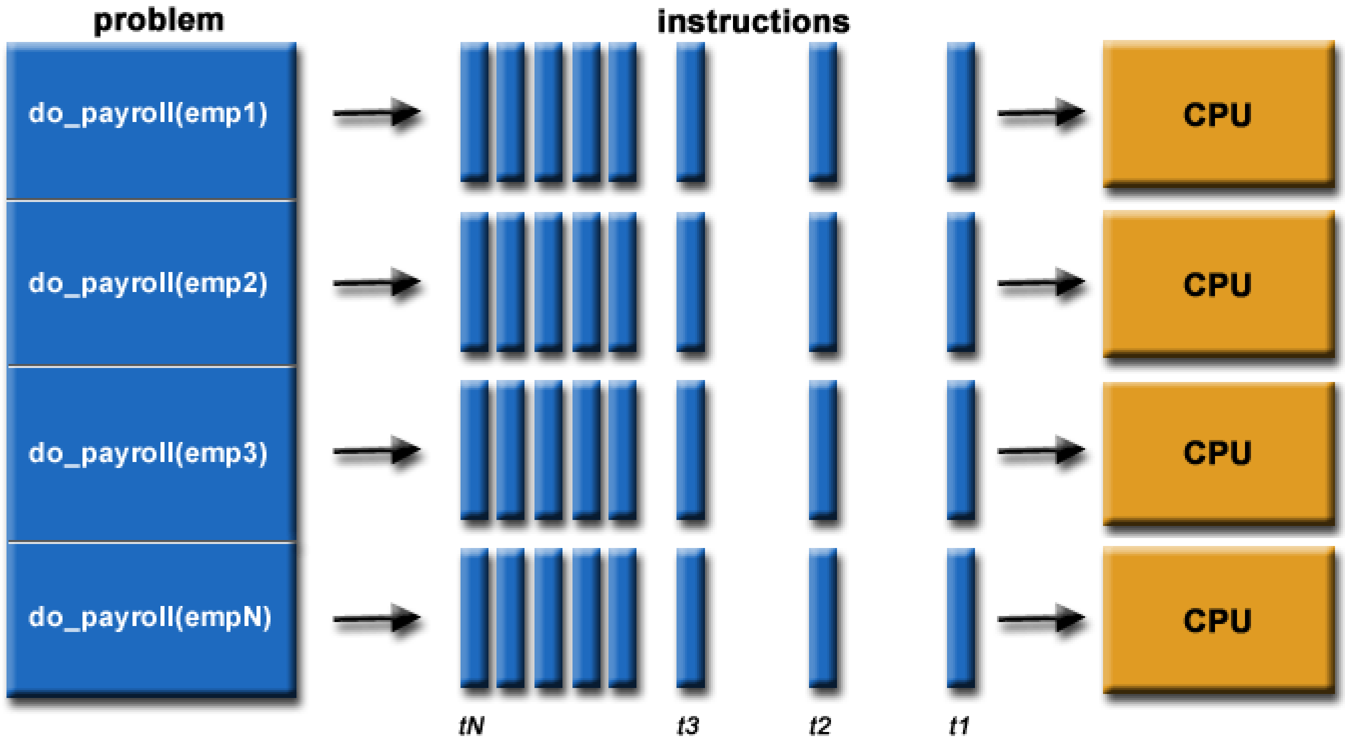


For example:



To this...

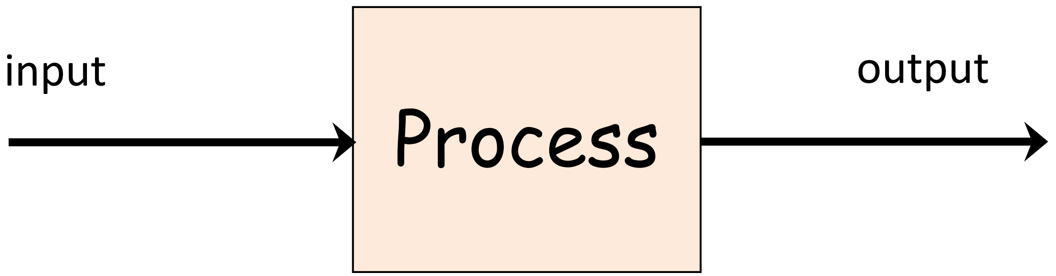
For example:



Disclaimer: this is a simple scenario – in practice things are a lot more complex...

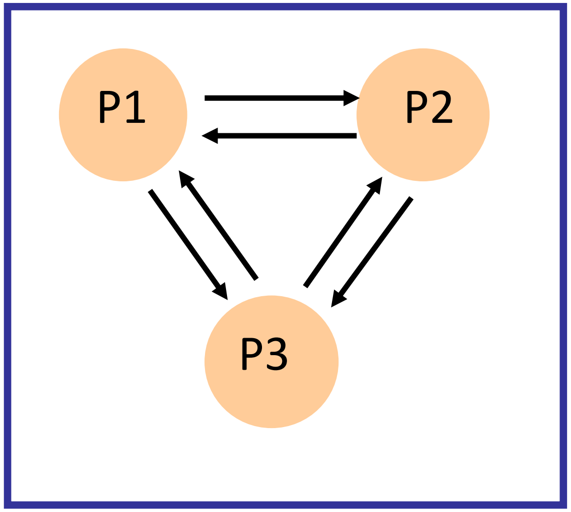
What is a ‘**Sequential Process**’?

- A ***sequential process*** is a unit of sequential execution.
- We structure complex systems as sets of simpler activities, each represented as a ***sequential process***.
- So processes are the basic building block of concurrent systems.
- Processes can overlap or be parallel, so as to reflect the concurrency inherent in the physical world, to offload time-consuming tasks, to manage communications or other devices

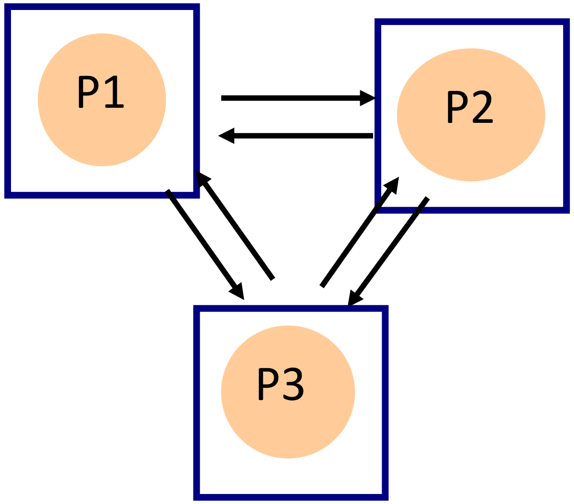


Is it really parallel?

3 parallel processes running . . .



. . . on the same CPU



. . . on 3 different CPUs

Concurrency? Parallelism?

- Concurrency
  - Logically simultaneous processing.
    - Does not imply multiple processing elements (PEs).
    - Interleaved execution on a single PE.
- Parallelism
  - Physically simultaneous processing.
    - Involves multiple PEs
    - Independent device operations.

Both concurrency & parallelism require controlled access to shared resources.

We shall use the terms parallel and concurrent interchangeably, and will only distinguish between real and pseudo-parallel execution if necessary.

Why is ‘Concurrency’ essential today?

Multiprocessing hardware → parallelism

- Performance gain
- Increased application throughput
  - an I/O call need only block one thread
- Increased application responsiveness
  - e.g. high priority processes for user requests…
- More appropriate structure for multitasking applications
  - e.g. one which interacts with the environment, controls multiple activities and handles multiple events.

Concurrency is ~~becoming~~ hard to ignore!

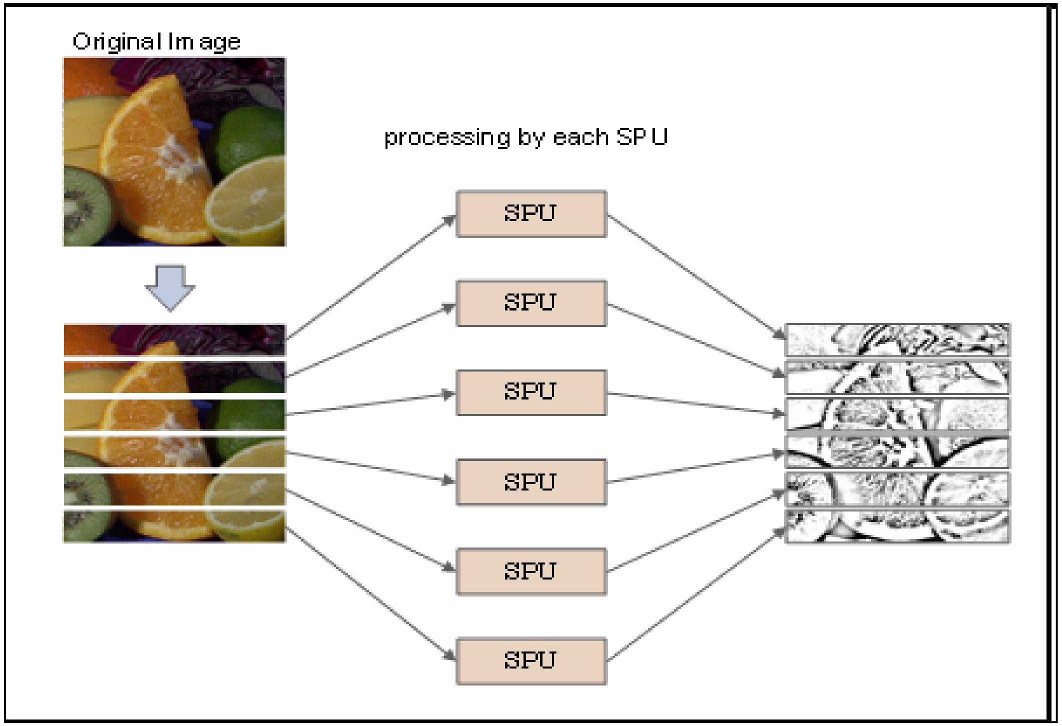
- Multicore chips are ~~increasingly~~ prevalent
  - Only multithreaded applications will see the performance benefits these chips offer

- Programming for the Web ~~often~~ requires concurrent programming (think image loading in Web browsers)

- Lots of domains in which concurrency is the norm, e.g.
  - Embedded software systems
  - Robotics
  - Simulation and modelling (e.g. weather prediction systems)

Example: image processing

- Trade ***processing over time*** for ***processing over space***
- Performance gain from multiprocessing hardware



Must handle concurrency properly!

Write efficient concurrent programs

- …understand **fundamental causes** why programs can produce different behaviours/results when run concurrently (e.g. deadlocks, race conditions, synchronisation etc)

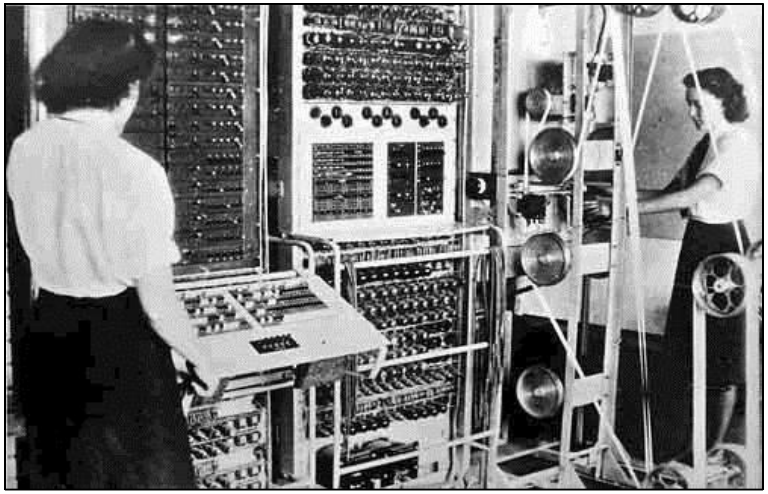
- …consider **systematic approaches** to control effects of concurrency: avoid ‘freezes’ and ‘unresponsiveness’ and optimize parallel execution performance

Therac-25: Computerized radiation therapy machine’s concurrent programming errors contributed to accidents causing deaths and serious injuries

Mars Rover: Problems with interaction between concurrent tasks caused periodic software resets reducing availability for exploration

It all started so innocently…

- until 1950s: Single-User Mode & Batch Processing
  - Single-User Mode: reserving the entire machine
    - e.g. 'I book from 9:00-13:00…'
    - debugging time was included in reserved time slot
  - Batch Processing: submitting jobs to a 'queue'
    - when program ends/fails next job is immediately executed
    - CPU still idle during I/O , low responsiveness of system



Colossus Mark 2 (1940s)

Universal Automatic Computer UNIVAC (1951)

First Steps towards Simultaneous Execution…

- 1960s: Era of Multi-Programming
  - multiple programs reside in memory at once
    - switching between programs using interrupts
    - introduction of concepts ‘memory protection’ and ‘virtual memory’
  - beginning of ‘theory of concurrent systems’
    - How to synchronize the processing of all these programs smoothly?
    - Communicating Sequential Processes (CSP - discussed later in the unit)

Leo I mainframe computer (1960s)

IBM mainframe (late 1960s)

Rise of Resource-Sharing Operating Systems

- 1970s: Time-Sharing and Memory-Sharing via OS
  - Fast Context Switching: time-slicing of a CPU between programs
    - illusion of using a (perhaps slower) machine alone
    - options of prioritising programs (in terms of resource use, rights etc)
    - but: switching overhead is significant
  - Shared Memory: programs share memory to communicate
    - enables information transfer between programs
    - but: mixing concepts of data storage and program communication

Mission control, Houston (1970s)

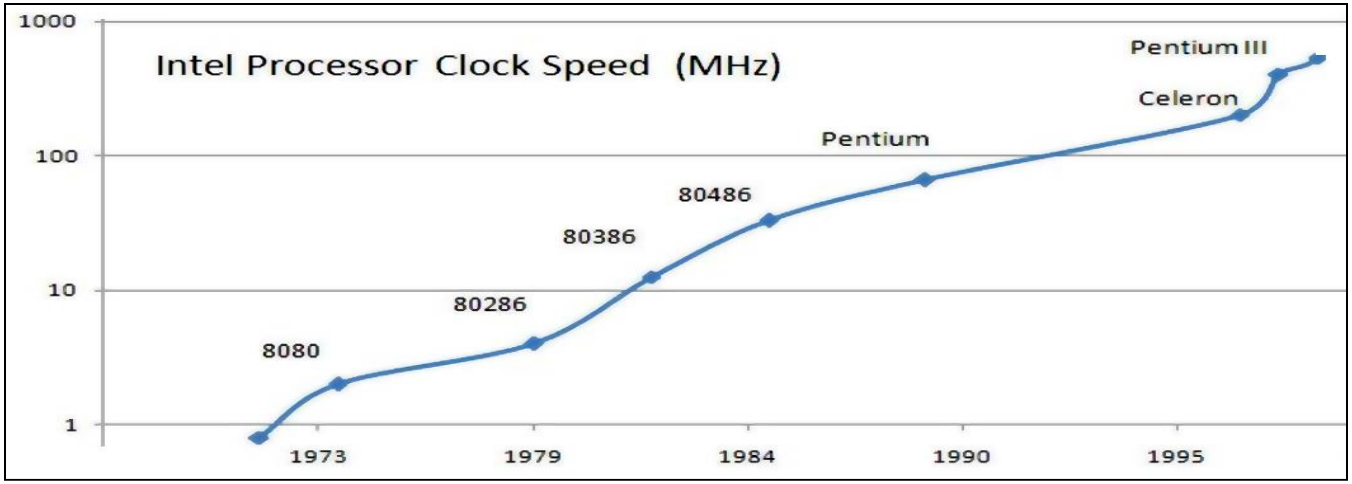
Apple II (1977)

Hardware Performance drives Efficiency Gains

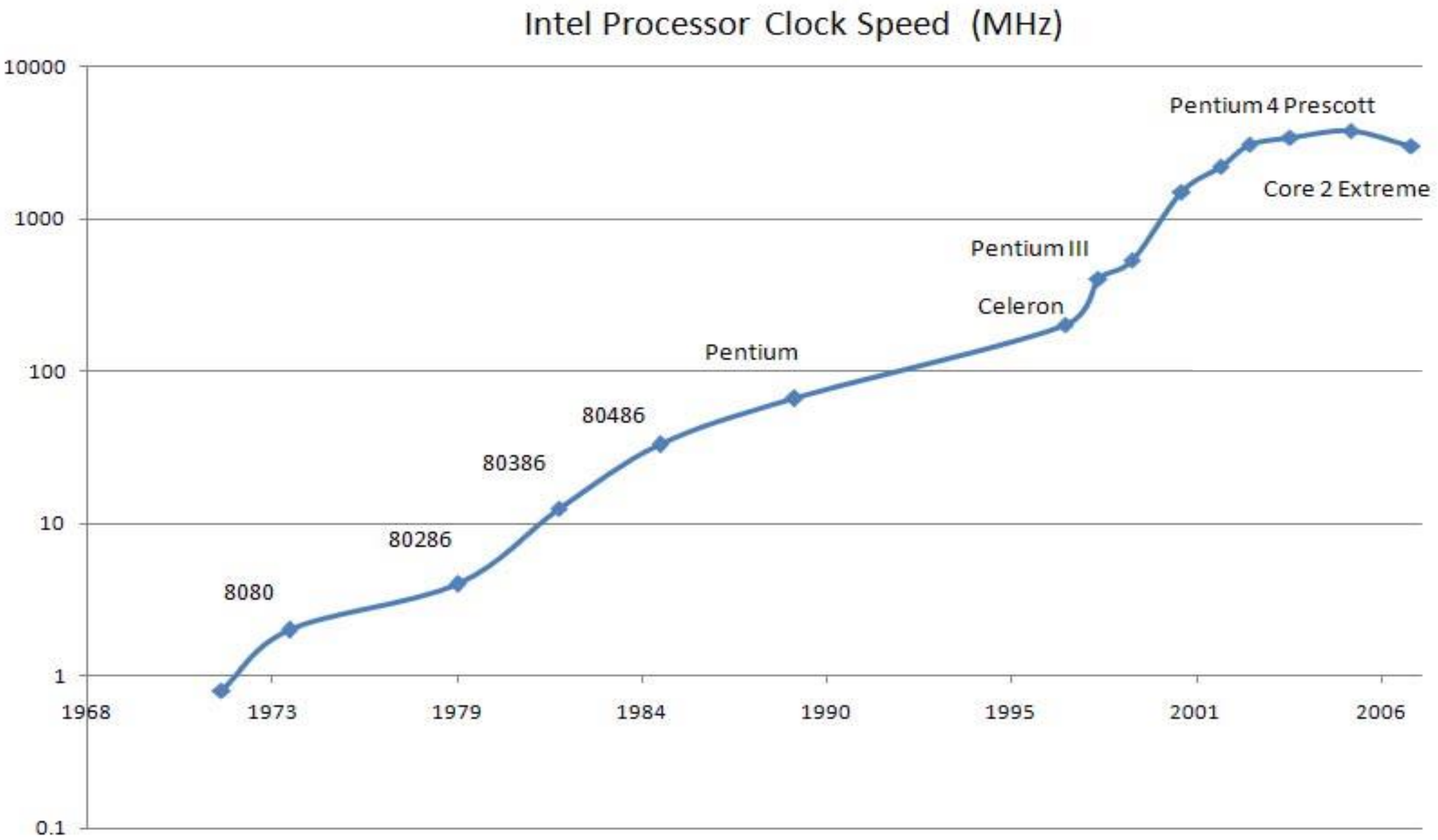
- 1980s: Era of Superscalar expansion (instruction level parallelism)
  - 50% annual improvement in performance
  - pipeline processor (10 CPI → 1 CPI) for implicit parallelism

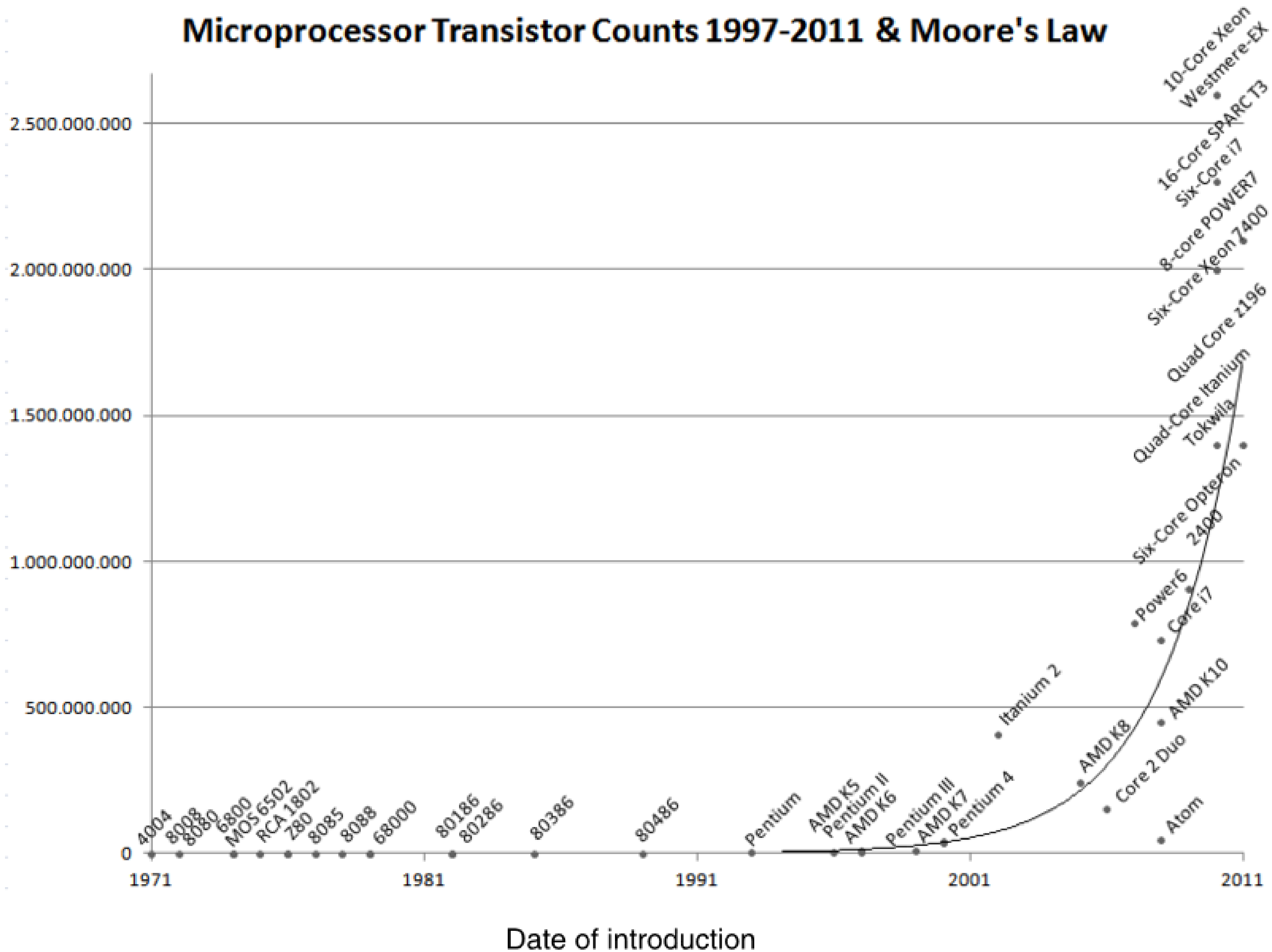
cycles per instruction

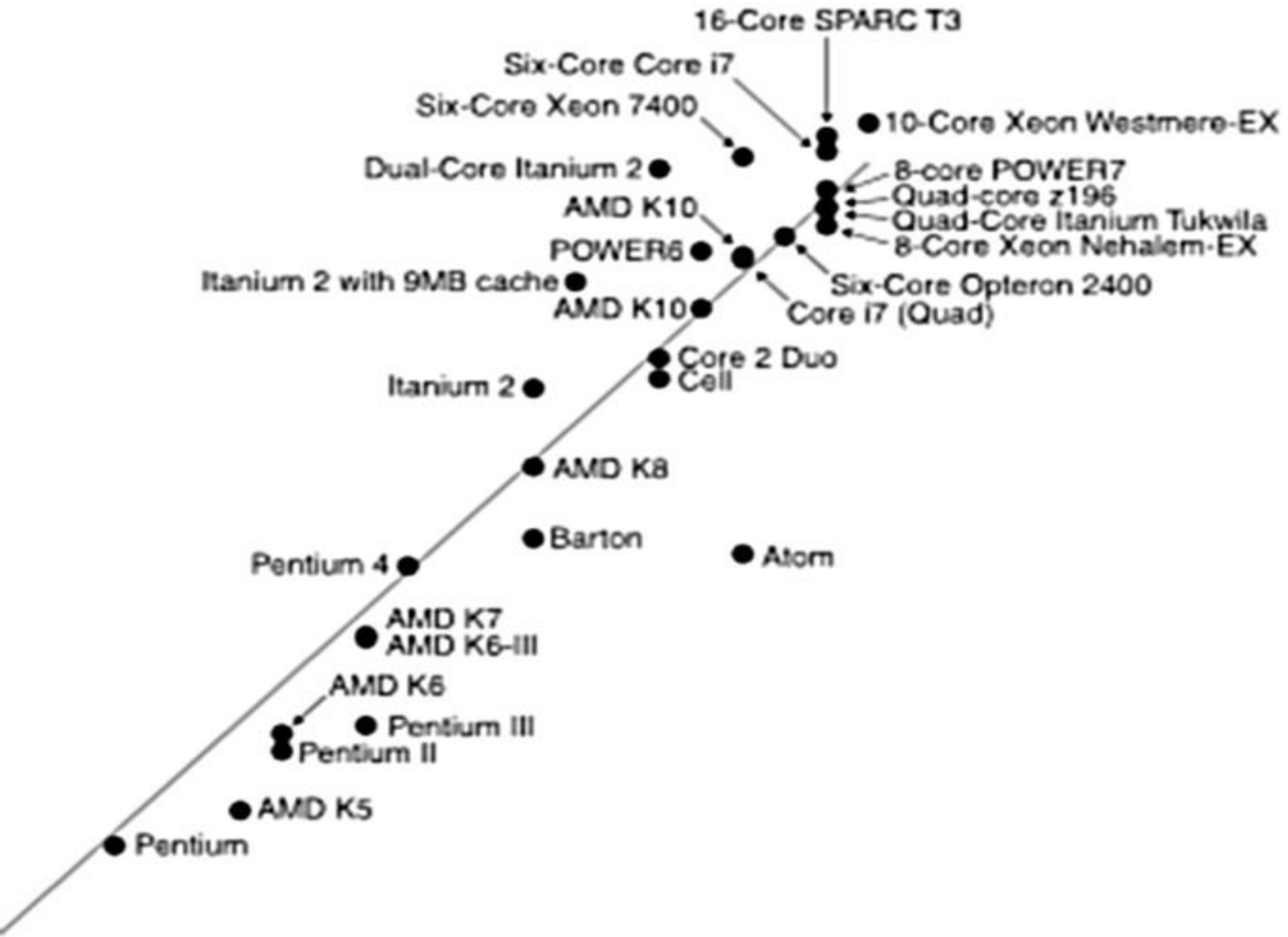
- 1990s: Era of Diminishing Returns
  - branch prediction etc. (1 CPI → 0.5 CPI)
  - performance below expectations



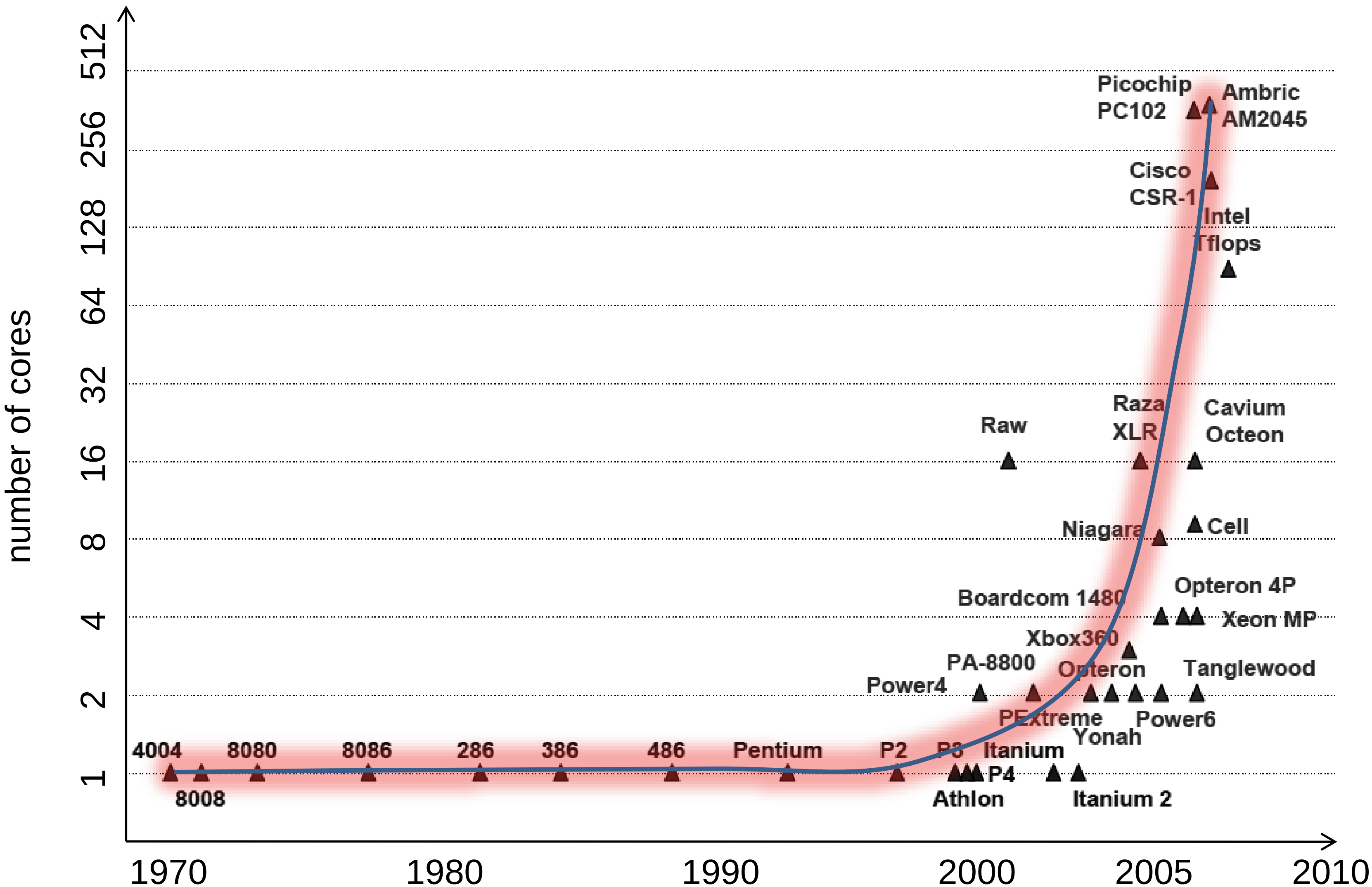
CRISIS: Clock Speeds hit Ceiling in 2000s




Microprocessor Transistor Counts 1997-2011 & Moore's Law
2.500.000.000
2.000.000.000
1.500.000.000
1.000.000.000
500.000.000
0
1971
1981
1991
2001
2011
Date of introduction
4004
8008
8080
6800
MOS 6502
RCA 1802
Z80
8085
8088
68000
80186
80286
80386
80486
Pentium
AMD K5
Pentium II
AMD K6
Pentium III
AMD K7
Pentium 4
Itanium 2
AMD K8
Core 2 Duo
Atom
AMD K10
Core i7
Power6
Six-Core Opteron 2400
Quad-Core Itanium Tokwila
Quad Core z196
Six-Core Xeon 7400
8-core POWER7
16-Core SPARC T3
Six-Core i7
10-Core Xeon Westmere-EX


16-Core SPARC T3
Six-Core Core i7
Six-Core Xeon 7400
10-Core Xeon Westmere-EX
Dual-Core Itanium 2
8-core POWER7
Quad-core z196
Quad-Core Itanium Tukwila
8-Core Xeon Nehalem-EX
AMD K10
POWER6
Itanium 2 with 9MB cache
Six-Core Opteron 2400
AMD K10
Core i7 (Quad)
Core 2 Duo
Cell
Itanium 2
AMD K8
Barton
Atom
Pentium 4
AMD K7
AMD K6-III
AMD K6
Pentium III
Pentium II
AMD K5
Pentium

A Multi-Core Revolution
Concurrent Programming is Back on the Menu!



Therefore: Concurrency is critical now!

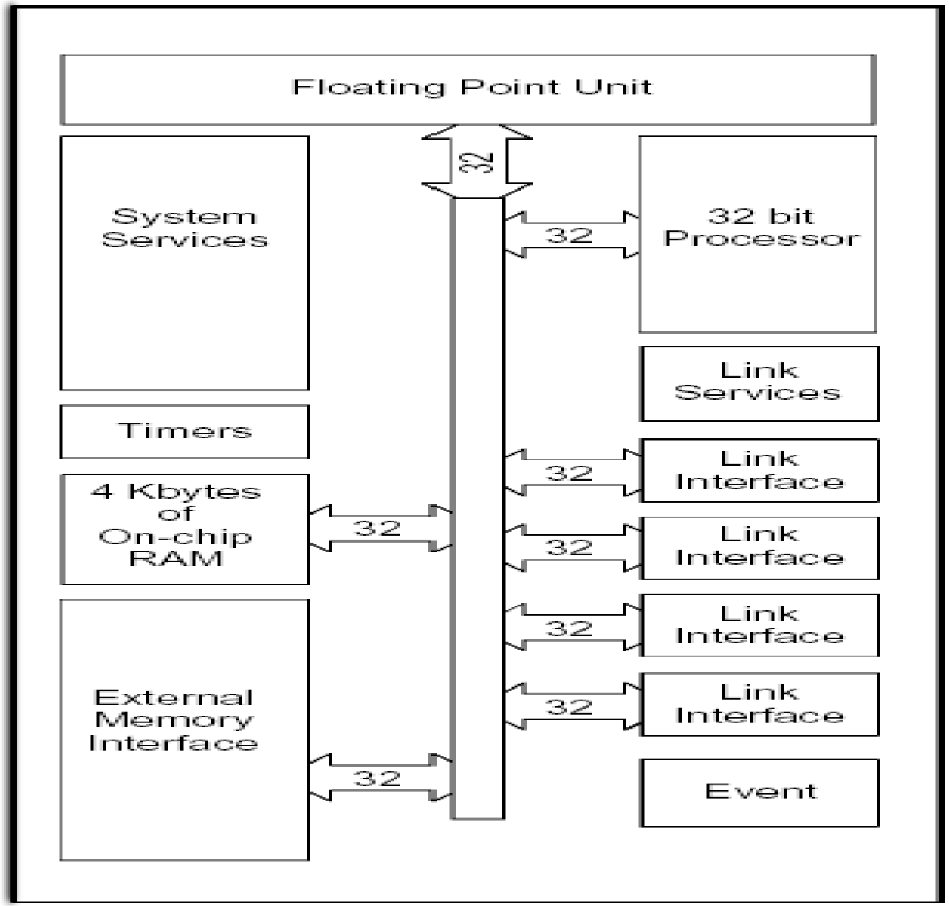
Multiprocessing hardware → parallelism

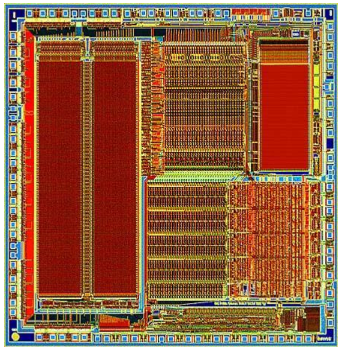
- Performance gain
- Increased application throughput
  - an I/O call need only block one thread
- Increased application responsiveness
  - e.g. high priority processes for user requests…
- More appropriate structure for multitasking applications
  - e.g. one which interacts with the environment, controls multiple activities and handles multiple events.

Old News: It has been there a long time

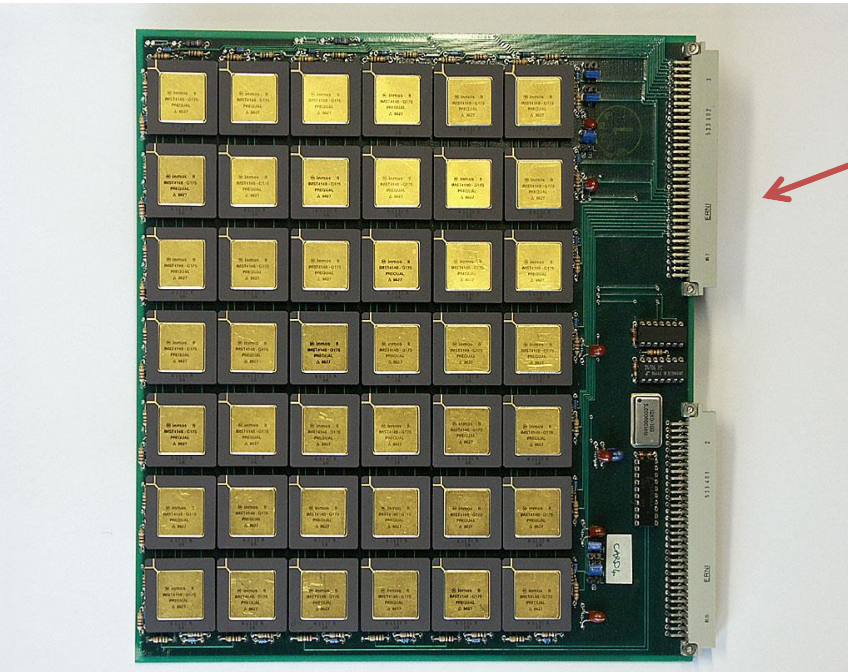
The Transputer – 1980s

- The first single chip computer designed for message-passing parallel systems, in 1980s, by INMOS
- *Transistor Computer.* Low cost, low power chip to form a complete processor
- It had a RISC type of instruction set



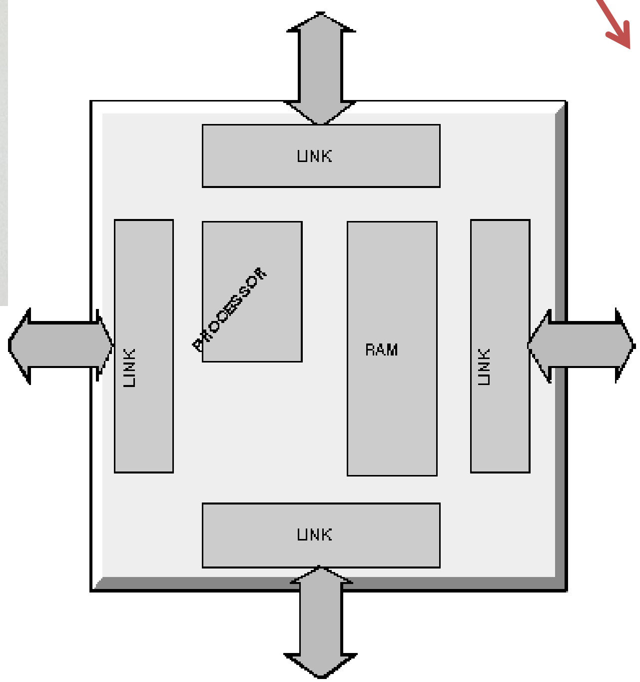


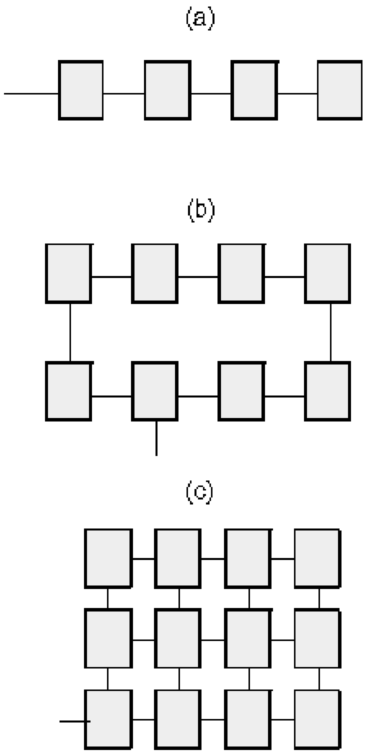
Transputer Network



Transputer Network Board

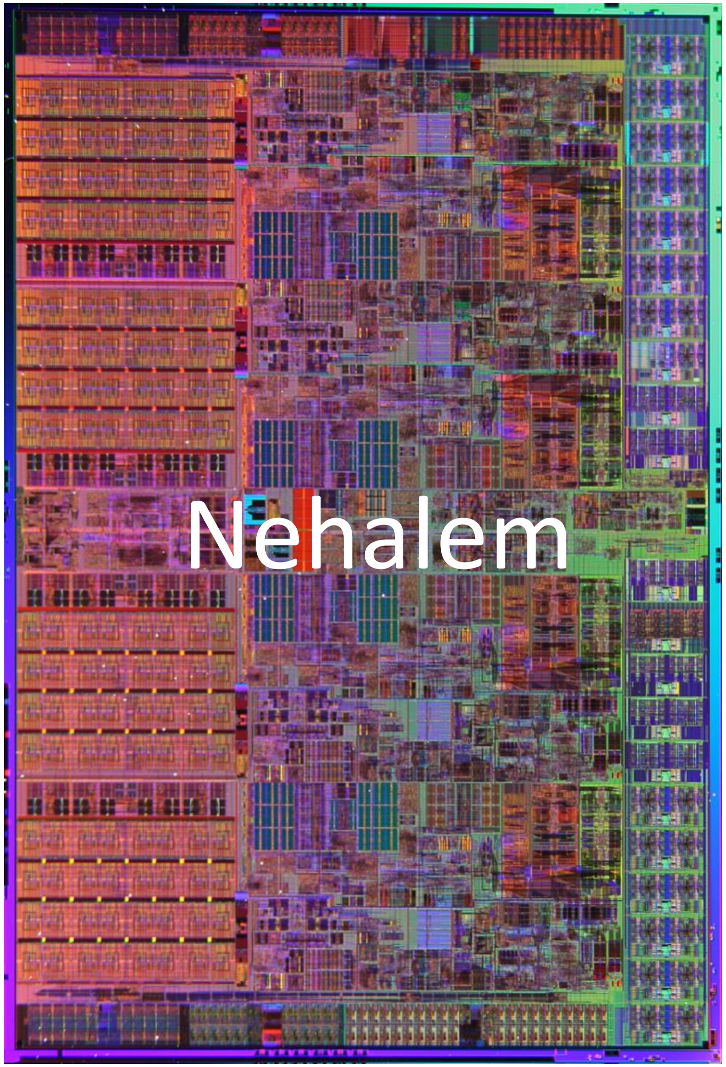
Transputer networks with specific connections





Technology constantly on the move!

- The Intel® Core™ i7 microprocessor (“Nehalem”)
  - 4 cores/chip
  - 45 nm, Hafnium hi-k dielectric
  - 731M Transistors
  - Shared L3 Cache - 8MB
  - L2 Cache - 1MB (256K x 4)
- Num of transistors not limiting factor
  - Currently ~ 2.5 billion transistors/chip
  - Problems:
    - Too much Power, Heat, Latency
    - Not enough Parallelism
- 3-dimensional chip technology?
  - Sandwiches of silicon
  - communication can be an issue



There is a Software side to the story…

First Software Crisis (1960s-1970s)
Problem: assembly language limits abstraction and portability
Solutions: ***high-level languages*** for uni-processors (C, Fortran etc)

Second Software Crisis (1980s-2000)
Problem: procedural languages limit maintainability and reusability
Solutions: ***object orientation*** (C++, Java, C# etc), software design process and tools

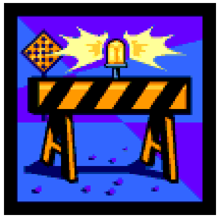
NOW: Third Software Crisis (2005-today)
Problem: sequential approach limits data throughput, real-time demand and distributed infrastructure
Challenge: explicit support for concurrency and communication…

- high-performance connectivity (Gbit networks, 4G, …)
- use physical parallelism (multicore, multiCPU, cloud…)
- ***logical parallelism*** (concurrent programming…)

Sharing Memory vs. Process Communication

Traditional: *Shared Memory Model*

- concurrent interaction via synchronized alterations of shared memory (e.g. C#, Java)
- problem of synchronization (controlling the concurrency) is left to the responsibility of the programmer!



Makes writing concurrent programs much more difficult than writing sequential programs!

More common Approach: *Message Passing Model*

- processes communicate by exchanging messages on channels (e.g. Occam, Ada , XC)
- can be efficiently implemented in existing multi-processor hardware with *or without* shared memory

Languages based on Parallel Execution: Occam

Occam: 1st language based on principles of **parallel execution**.

- automatic communication and synchronization between concurrent processes.
- Theoretical foundation based on CSP, first introduced by Tony Hoare in 1968. (CSP = Communicating Sequential Processes)
- Inmos developed Occam and the Transputer in the 1980s.

Philosophy behind the Occam language:
Minimalistic approach. Easy to learn.

14th century Franciscan friar William of Ockham:
"It is vain to do with more what can be done with fewer."

Languages based on Parallel Execution: **XC**

- C-based imperative language developed by XMOS
- shows semantic similarities to Occam in a familiar C syntax
- design was heavily influenced by CSP, which can be used to reason about XC programs
- supports explicit parallelism and channel communication (for versatility: it also includes guarded commands)
- directly supports timers and port access
- compiles directly to drive XMOS multi-core hardware

You will get your own xmos board kit for your coursework later

☺

TASK: Form Pair Programming Teams Now

- Within the next 48 hours: find a team partner and register on the course website for one of the two lab slots
- You will then work together during TB1 **as a pair** in your programming team

Lecture 2 Outlook

XC: Towards Concurrent Programming in xC