Lecture 2: Introduction to Concurrency

Learning Goals

- Describe the differences between **single-tasking** and **multi-tasking**
- Identify **challenges introduced** by adding concurrency
- Describe an example of a multitasking system (perhaps with a diagram)
- Identify whether or not a program will benefit from concurrency, either in terms of **efficiency** or **scalability**
- Describe various methods of implementing multitasking, and list some advantages/disadvantages of each
- Define and describe **time-slicing** and how it differs from **pseudo-multitasking**

Single-Task vs Multi-Task

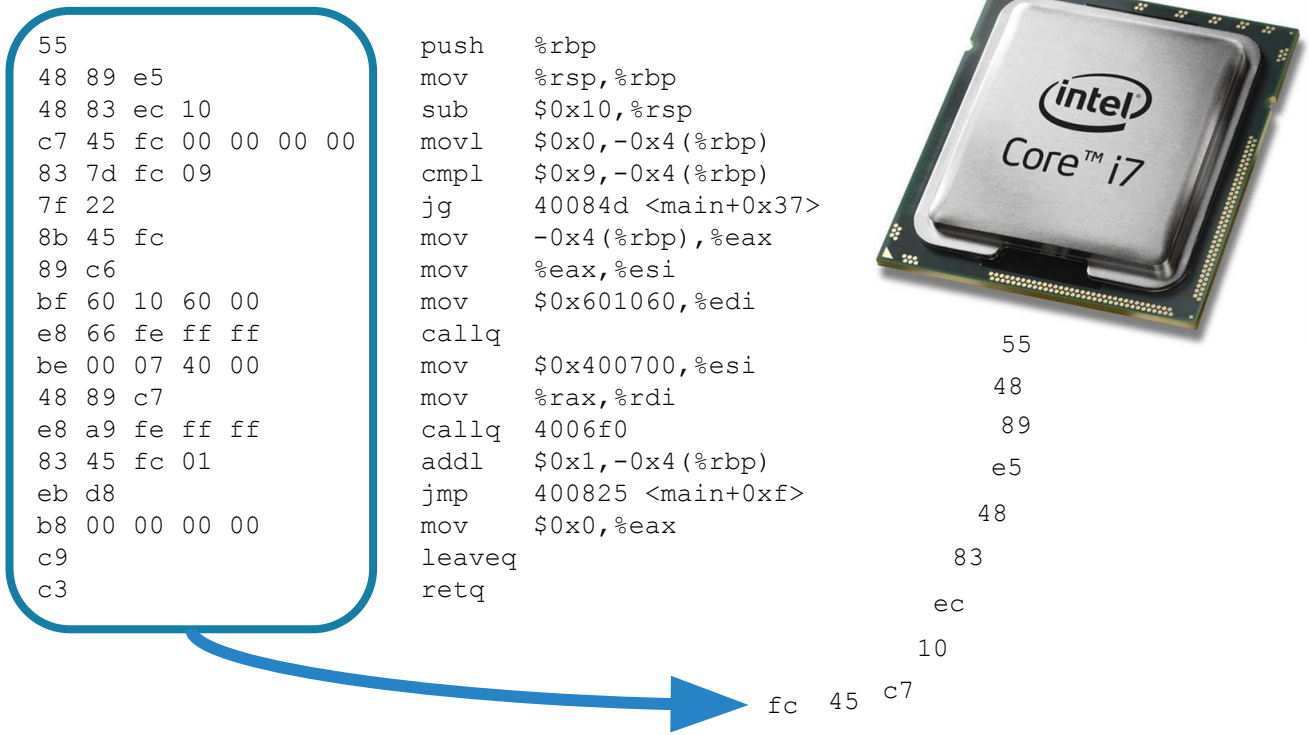
Single Tasking: a single program (thread) that runs sequentially

```
using System;
using System.Threading;


namespace concurrency
{
    class Program
    {
        static void Main(string[] args)
        {
            for(int i = 0; i < 3; i++) {
                Console.WriteLine("Some process takes 4s!");
                Thread.Sleep(4000);

            }


        }
    }
}
```



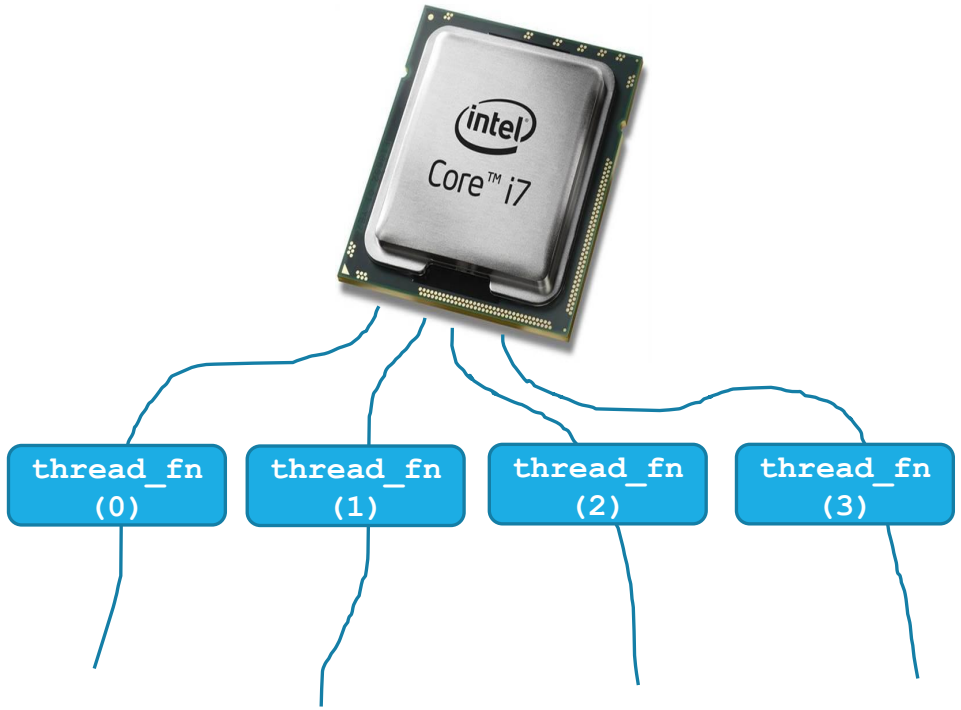
Single-Task vs Multi-Task

Multi Tasking: sections of code can be executed in **parallel**

```
using System;
using System.Threading;
namespace concurrency
{
    class Program
    {
        static void Main(string[] args)
        {
            for (int i = 0; i < 2; i++)
            {
                Thread thread = new Thread(MyProcess);

                // worker threads
                thread.Start();
            }

            Console.WriteLine("Do something that takes 4 seconds to process!");
            Thread.Sleep(4000);
            Console.WriteLine("Done!");
        }
        private static void MyProcess(object obj)
        {
            Console.WriteLine("Do something that takes 4 seconds to process!");
            Thread.Sleep(4000);
            Console.WriteLine("Done!");
        }
    }
}
```



Multi-Tasking

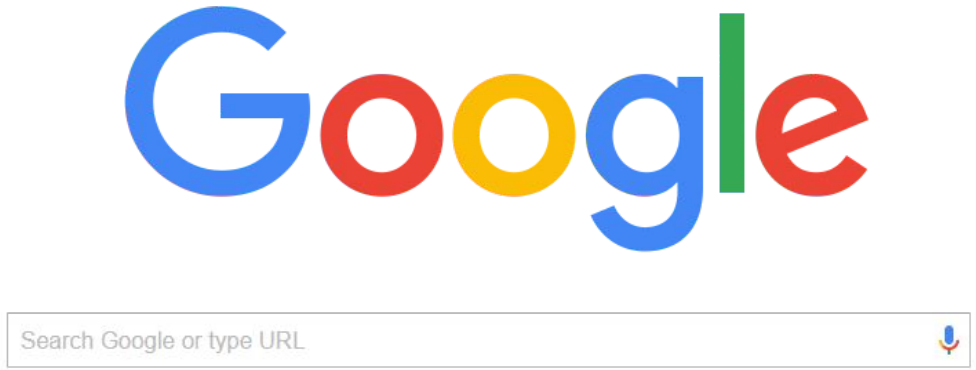
Advantages:

- greater **flexibility** – system can be distributed across several servers
- greater **scalability** – a single program can be instantiated as many times as needed

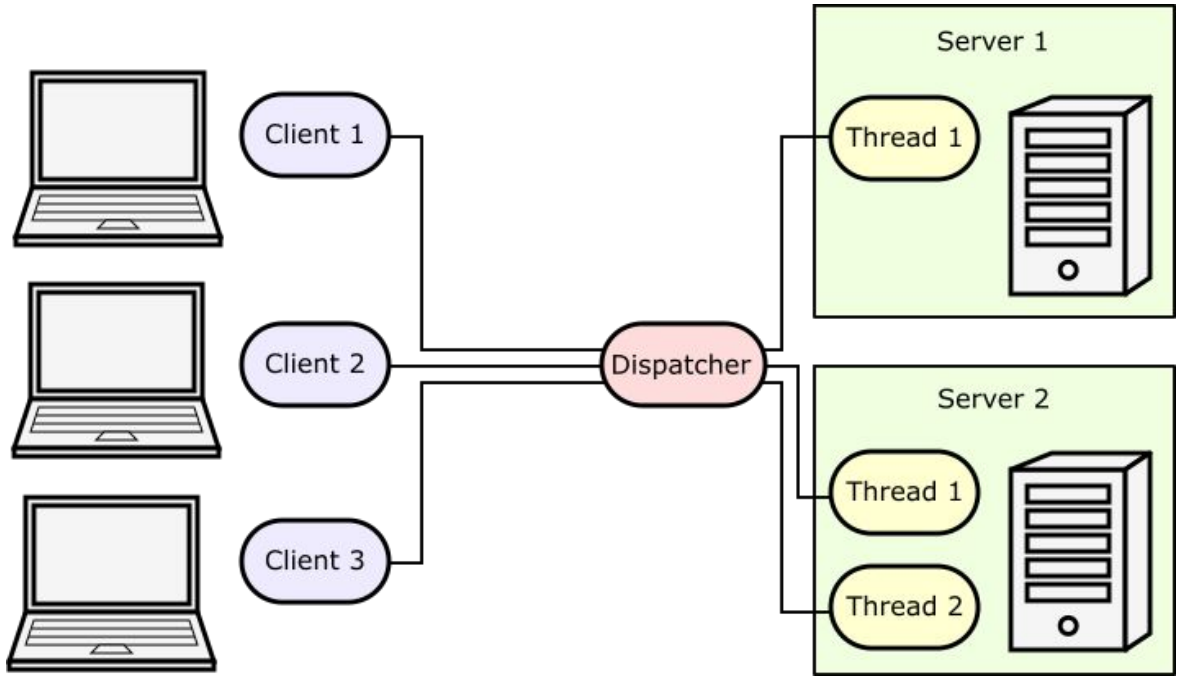
Disadvantages:

- challenges in **decomposing** system into appropriate concurrent units
- challenges in **communication** between parallel units
- challenges in **synchronization** between parallel units
- challenges in **debugging** and **testing**

Multi-Tasking Example



- 60k searches per second
- 0.5 s to process each search

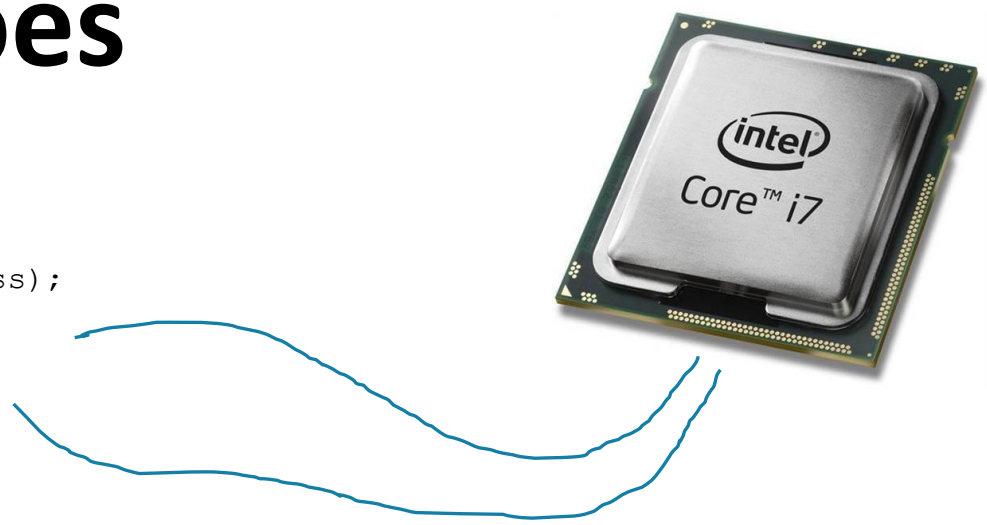


Multi-Tasking Woes

```
void increment() {

  int x = readMemoryValue(shared_memory_address);
  x = x + 1;
  writeMemoryValue(shared_memory_address, x);

}
```



Thread 1:

```
int x = readMemoryValue(shared_memory_address);
x = x + 1;
writeMemoryValue(shared_memory_address, x);
```

Thread 2:

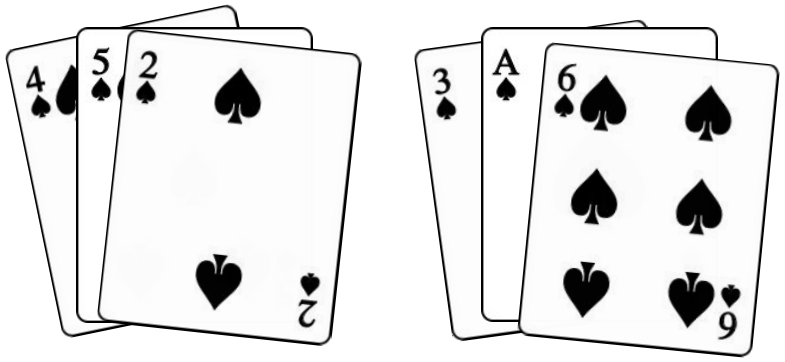
```
int x = readMemoryValue(shared_memory_address);
x = x + 1;
writeMemoryValue(shared_memory_address, x);
```

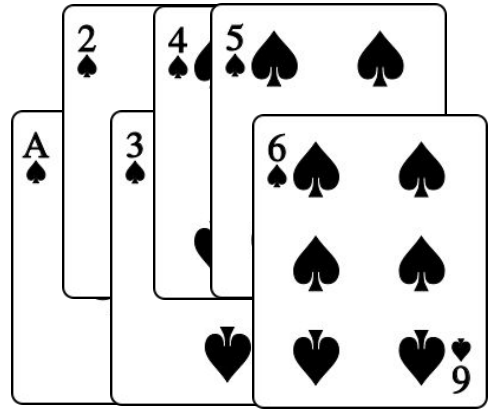
Parallel Programming

Multi-tasking has the **potential** to make your software faster

Parallel Programming: specialized area of concurrent programming that focuses on designing faster algorithms using concurrency.

e.g. parallel merge sort





Parallel Programming

In general, speeding up code is hard.

- Splitting system into parallel units introduces **data dependencies**
- Some sections need to **wait** for other portions to be complete
- Introduces need for **synchronization**, limits gains

Speed-up Factor: $S_p = \frac{T_1}{T_p}$

Merge Sort: 10 s

Parallel Merge Sort (p=4): 7 s

Speedup: 1.4x

Multitasking Implementations

- Pseudo-multitasking
- Multiple dedicated processors/cores
- Distributed systems
- Time-sliced processors/cores

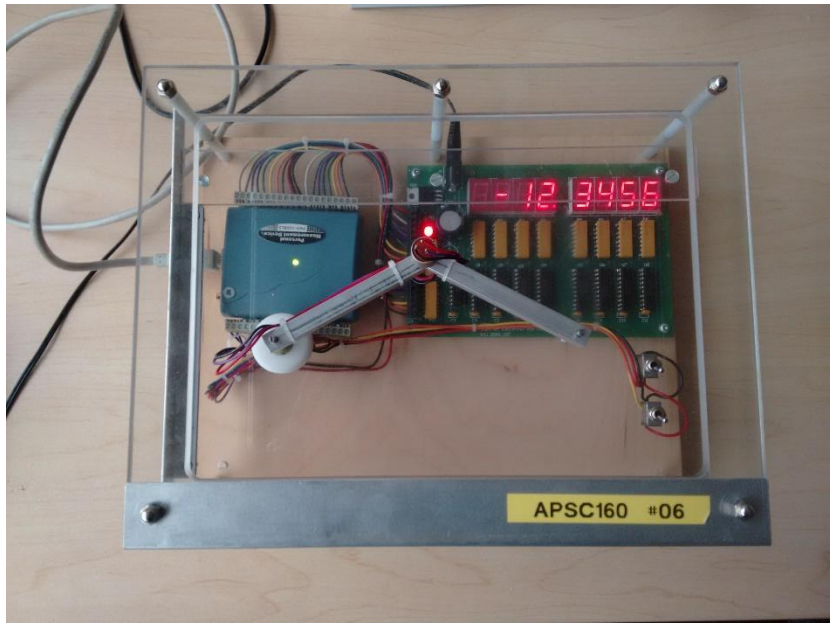
Pseudo Multitasking

Simply switch between a bunch of sequential tasks fast enough that it **looks** like they are running in parallel.

```
int main() {

   // continuously cycle through set of monitors
   // to give appearance of parallel operation
   while ( deviceIsOn() ) {
      monitorTemperature();
      monitorHumidity();
      monitorAtmosphericPressure();
      updateDisplays();
   }

   return 0;
}
```



Pseudo Multitasking

Advantages:

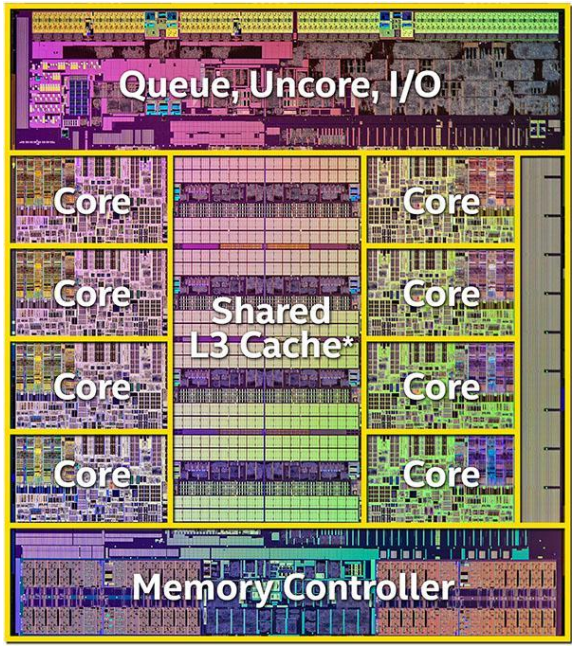
- simple to implement and debug
- no knowledge of operating system required
- communication and synchronization is trivial

Disadvantages:

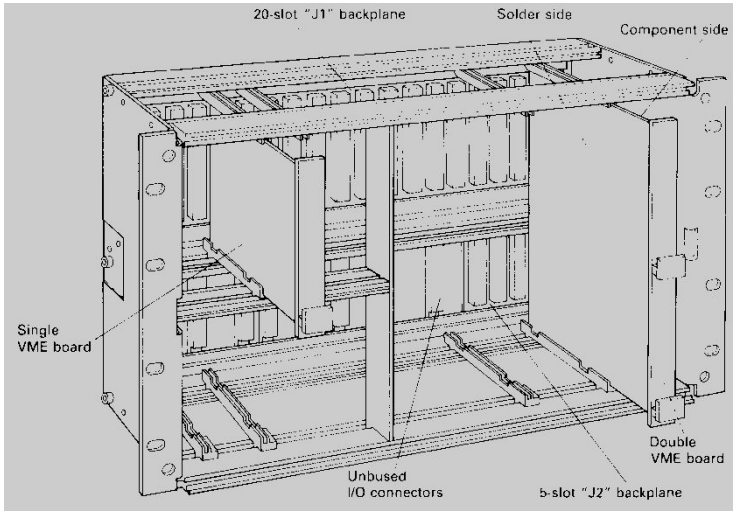
- tasks must be **brief**, preferably with guaranteed maximum execution times
- tasks must **never loop** on a condition or **wait** for input
- if a task crashes, brings down the entire system

Multiple Dedicated Processors/Cores

Each processor/core is given one task to run on its own, synchronizes with others.



(2015) 8 Core I7 CPU with shared Cache and Memory controller



Custom server with shared backplane, communicate over a VMEbus

Multiple Dedicated Processors/Cores

Advantages:

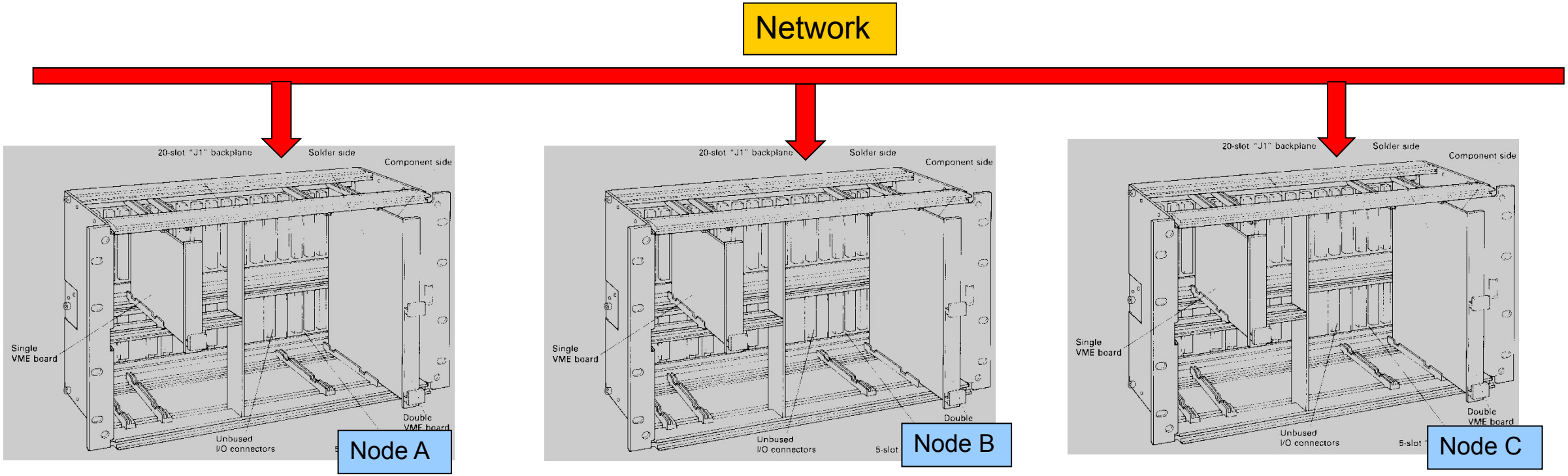
- **customizable** to the application
- extremely high **performance**

Disadvantages:

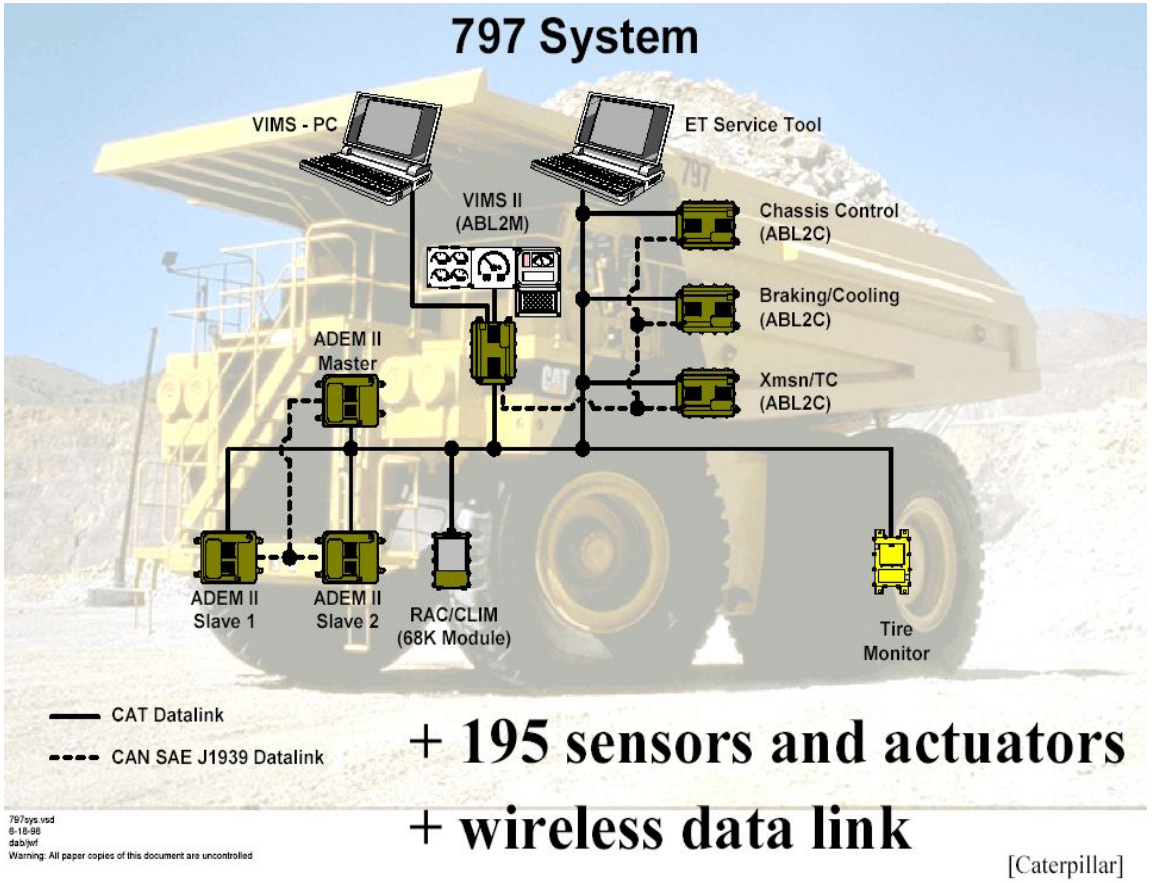
- highly **expensive**
- requires complex **hardware**
- **inflexible**, cannot accommodate dynamic process creation
- difficult to debug

Distributed Systems

Each processor/core is given one task to run on its own, processors can be in different machines/locations and communicate over a **network**.



Distributed Systems



+ 195 sensors and actuators

+ wireless data link

[Caterpillar]

Microsoft's data center in San Antonio, Texas

Distributed Systems

Advantages:

- **flexibility** in adding/removing resources
- if one node goes down, the rest can continue and pick up the slack

Disadvantages:

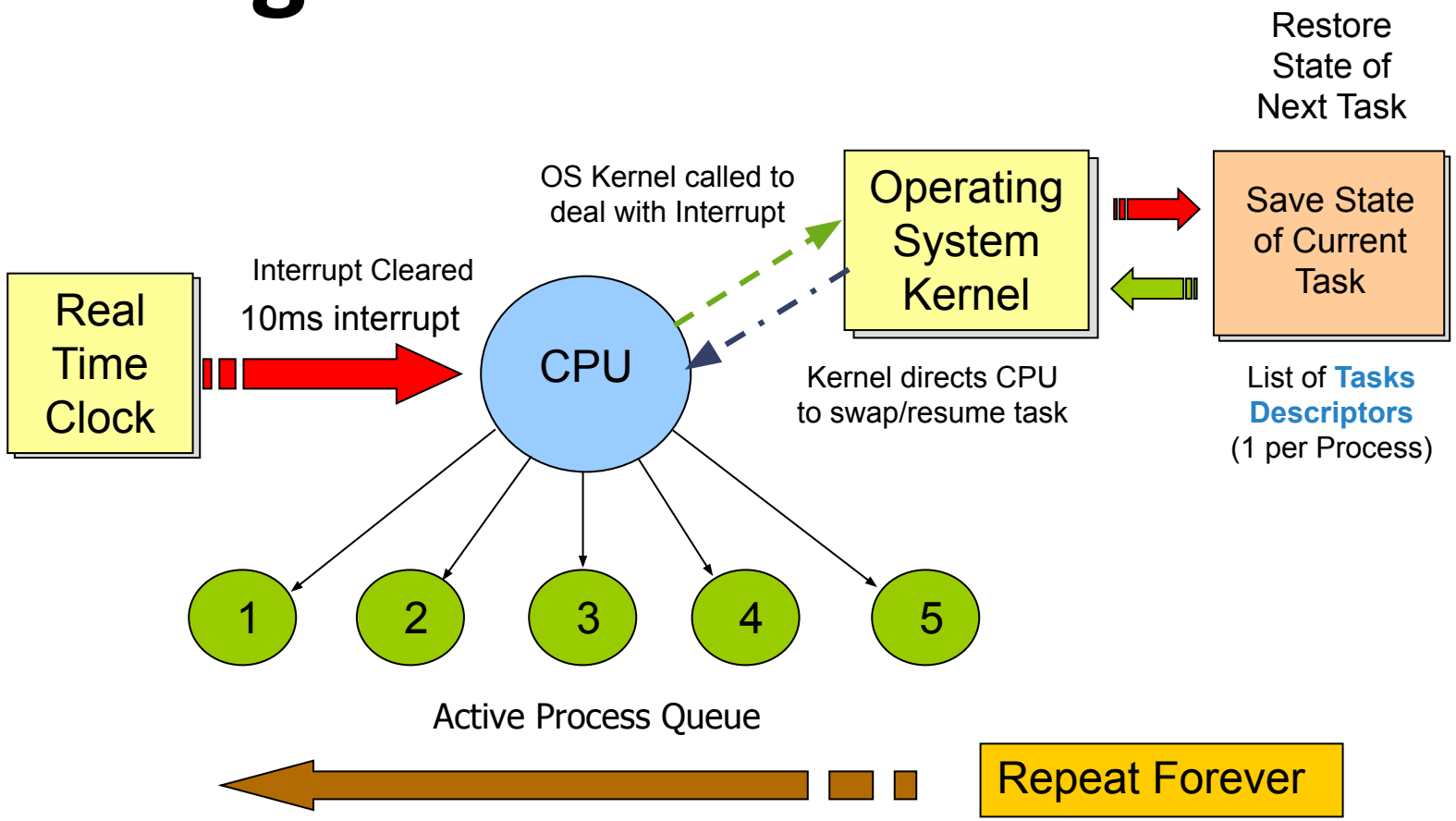
- network communication is slower and less reliable
- complexity in design, requires redundancy in case of disconnection or errors

Time-Slicing

The more practical and cheaper approach, used in most systems today. Relies on a **Multi-Tasking Operating System** (MTOS) in conjunction with a time-sliced CPU/core.

- processor's computation time is divided into **time-slices**
- OS **assigns** slices to programs/threads
- processor **swaps** between programs/threads rapidly, executing one time slice after the other, sequentially

Time-Slicing



Time-Slicing

Advantages:

- **splitting** of execution is handled automatically by the OS
- sub-tasks no longer need to be short, can wait on conditions or for input
- if one task goes down, others can still run

Disadvantages:

- context switch (saving current task's state and restoring next) adds overhead

Homework:

- Read **Introduction to Git**

- Create a free github account:
  https://github.com

- Review C# syntax: It is expected that you are familiar with C# programming language