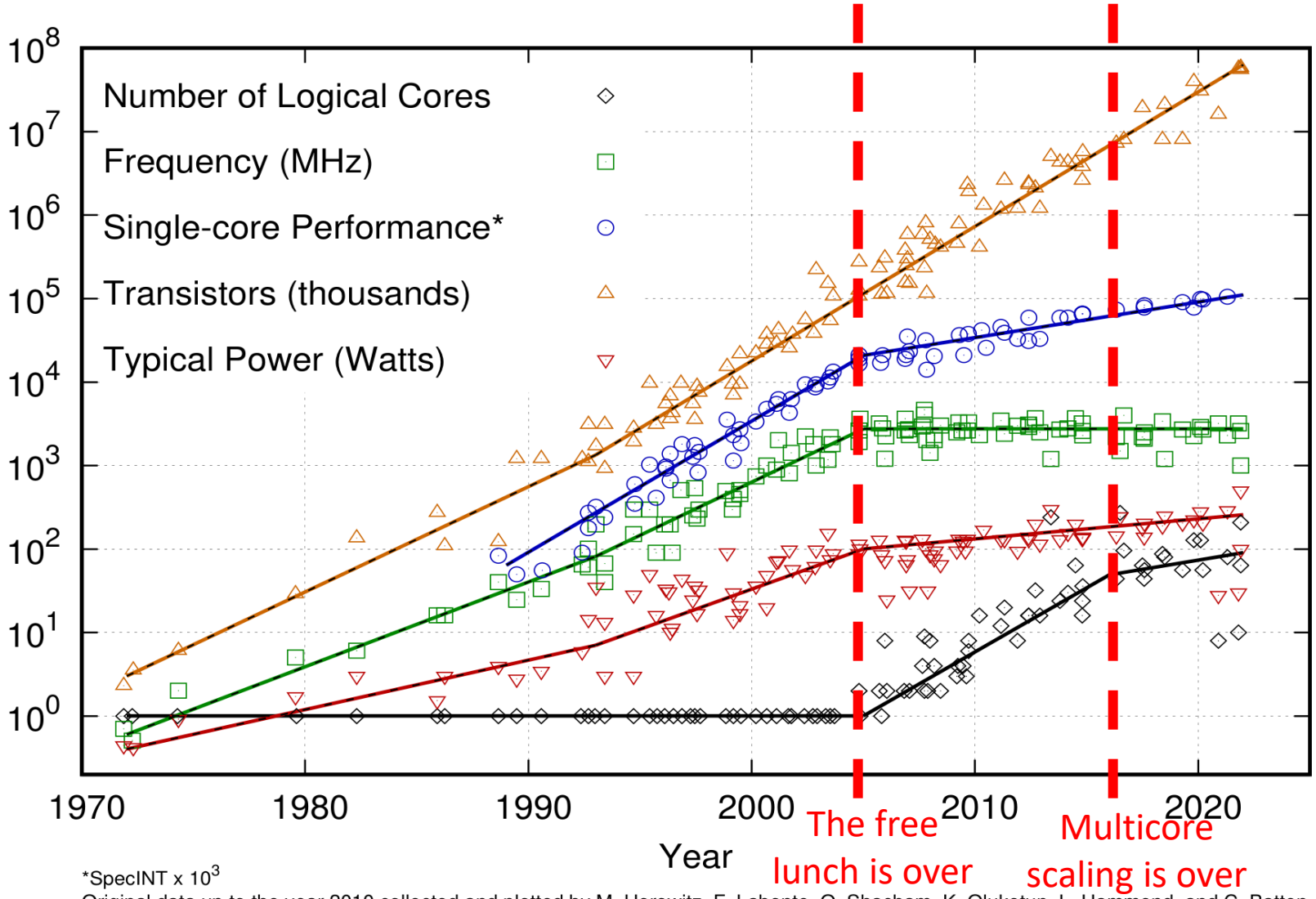
Programmazione concorrente

Laurea Magistrale in Ingegneria Informatica
Università Tor Vergata
Docente: Romolo Marotta

Introduction

Trend in processor technology



*SpecINT x 10^3

Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten

New plot and data collected for 2010-2018 by K. Rupp

Trend in processor technology

- Multicore is a standard and established technology
- Applications should be AT LEAST scalable on homogenous cores
  - Necessarily when remote computing power is not available
  - Ideally able to exploit different “kinds” of computing units
- Concurrent and parallel programming is a requirement to exploit current and future hardware

Parallel programming

Ad-hoc concurrent programming languages

- Development tools
  - Compilers
  - MPI, OpenMP, libraries
  - Tools to debug parallel code (gdb, valgrind)
- Writing parallel code is an art
  - There are approaches, not prepackaged solutions
  - Every machine has its own singularities
  - Every problem to face has different requisites
  - The most efficient parallel algorithm might **not** be the most intuitive one

A classical example

INIT

```
1. Buffer b;
```

PRODUCER

```
1. while(1) {
2.
3.
4.    <Write on b>
5.
6.
7. }
```

CONSUMER

```
1. while(1) {
2.
3.
4.    <Read from b>
5.
6.
7. }
```

A classical example

INIT

```
1. Buffer b;
2. Semaphore p = 0;
```

PRODUCER

```
1. while(1) {
2.   wait(p);
3.
4.   <Write on b>
5.
6.   signal(p);
7. }
```

CONSUMER

```
1. while(1) {
2.   wait(p);
3.
4.   <Read from b>
5.
6.   signal(p);
7. }
```

A classical example

INIT

```
1. Buffer b;
2. Semaphore p = 0;
3. Semaphore c = 0;
```

PRODUCER

```
1. while(1) {
2.   wait(p);
3.
4.   <Write on b>
5.
6.   signal(c);
7. }
```

CONSUMER

```
1. while(1) {
2.   wait(c);
3.
4.   <Read from b>
5.
6.   signal(p);
7. }
```

A classical example

INIT

```
1. Buffer b;
2. Semaphore p = 1;
3. Semaphore c = 0;
```

PRODUCER

```
1. while(1) {
2.   wait(p);
3.
4.   <Write on b>
5.
6.   signal(c);
7. }
```

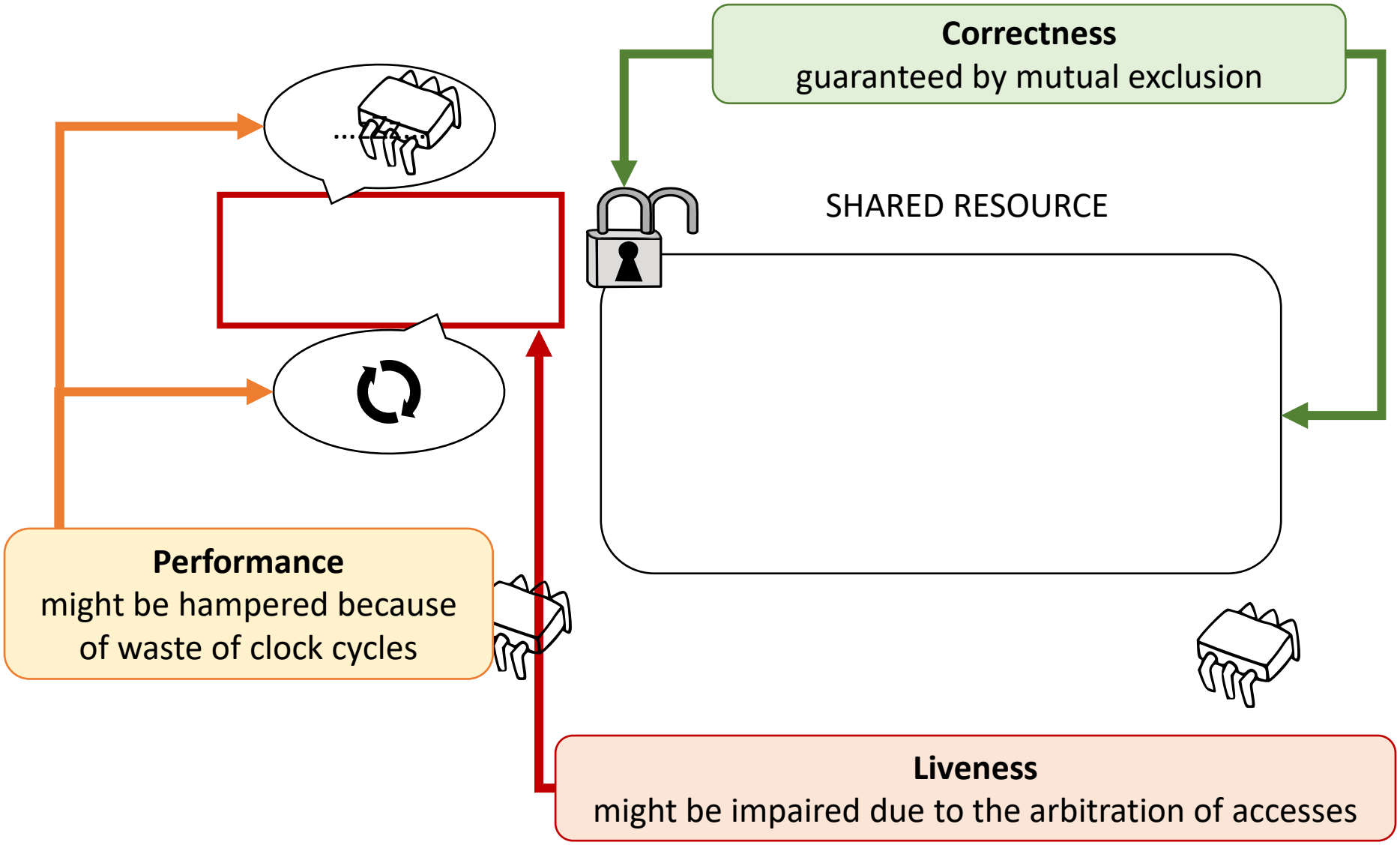
CONSUMER

```
1. while(1) {
2.   wait(c);
3.
4.   <Read from b>
5.
6.   signal(p);
7. }
```

Another example

- Challenge
  - Count primes between 1 and 10^8
- Given
  - N threads
  - 1 thread for each logical cpu
- Goals
  - Run N times faster

On concurrent programming



What do we want from parallel programs?

- **Safety:** *nothing wrong happens* (Correctness)
  - parallel versions of our programs should be correct as their sequential implementations
- **Liveliness:** *something good happens eventually* (Progress)
  - if a sequential program terminates with a given input, we want that its parallel alternative also completes with the same input
- **Performance**
  - we want to exploit our parallel hardware

A bit of terminology

- Hardware
  - Processor
  - CPU
  - CPU-Core
  - Logical Core
  - Hardware thread
- Software
  - Process
  - Thread
  - Fiber
  - Task
- Programs
  - Sequential
  - Concurrent
  - Parallel
  - Distrubuted
- Memory
  - Shared
  - Distributed

The system model

- Threads (aka processes)
- Cores (aka cpus)
- Shared memory
- Arbitrary long asynchronous delays
- Scheduler
  - A system component that decides which/when a thread runs on a given core