Example: Shared memory [Primality test (Herlihy & Shavit)]

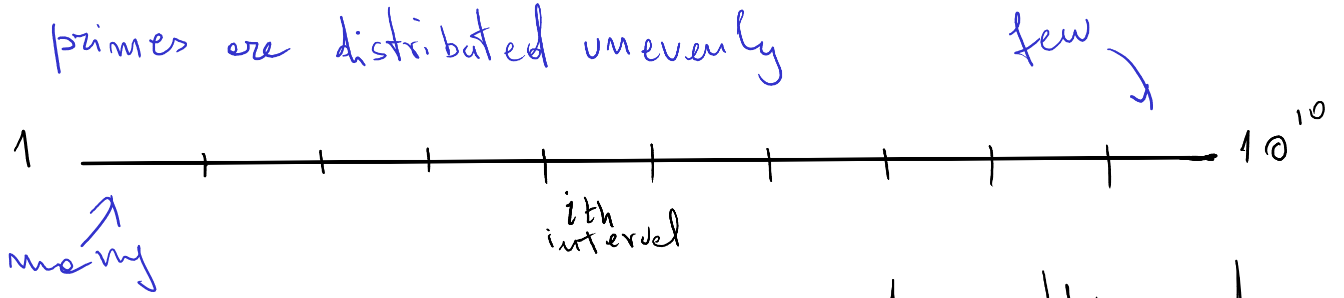
Print all prime integers between 1 & 10^{10}

```
void primeSeq {
  for (j = 1, j<10^10; j++) {
    if (isPrime(j))
      print(j);
  }
}
```

now let's try concurrently

Split the interval and launch a thread on each portion

primes are distributed unevenly




```
void primePrint(int i){ // i non-negative
  for (j = i*10^9+1, j<(i+1)*10^9; j++) {
    if (isPrime(j))
      print(j);
  }
}
```

- How good is this idea?
  - Uneven load
  - Is there an "optimal" split?

- coordination
  - shared memory

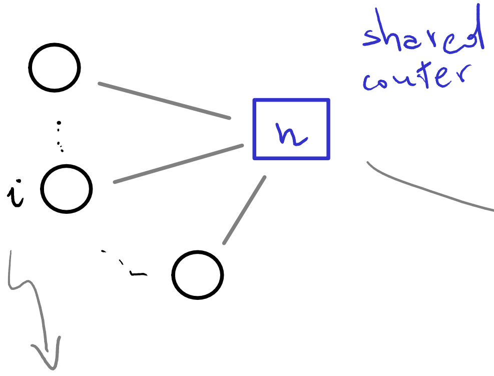
    historically this is the "first" and more used approach
  - communication

    this is becoming more & more popular
    - synchronous
    - asynchronous

    Message-Passing
    Event notif.
    Generative commun.

Exercise 0

Find a better multi-threaded program for the primality test




```
void primePrint( Counter counter ) {
  long j = 0;
  while (j < 10^10) {
    j = counter.getAndIncrement();
    if (isPrime(j))
      print(j);
  }
}
```

RACES

THIS IS NOT GOOD!

```
public class Counter {
  private long value;
     synchronised
  public long getAndIncrement() {
    return value++;
  }
}
```

Say something bad about JAVA?

temp := value
value ++
return Temp

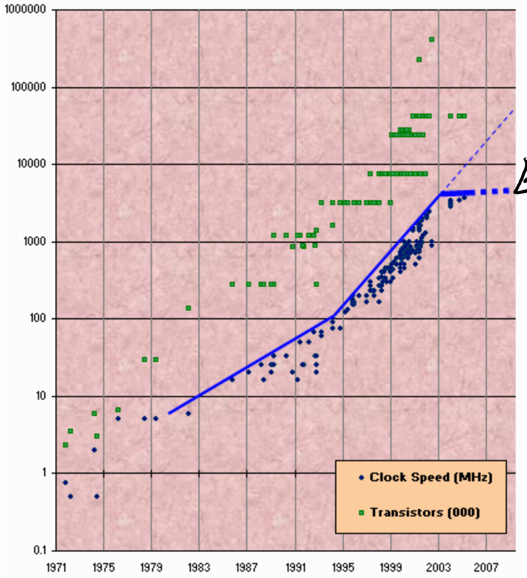
even better
WHY?

```
public long getAndIncrement() {
    synchronized {
      temp  = value;
      value = temp + 1;
    }
    return temp;
  }
}
```

REFLECT about why this solution is better than splitting

Hw — Efficiency is no longer an hw thing → Sw

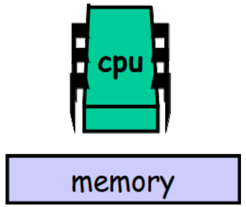
The art of multi-processor programming



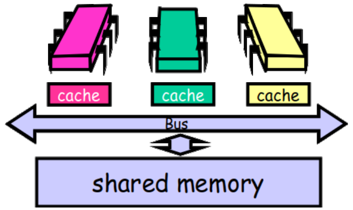
clock speed

transistors grows by a factor of 10 every 10 years

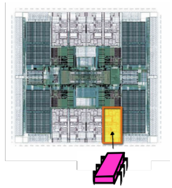
CPU speed is plateauing



uniprocessor



shared memory processor



multicore

- programming constructs in ALL languages
  - "new" languages
    - Go
    - Scala
    - Elixir / Erlang
    - Ballerina
    - Concurrnas
- supporting library, AKKA
- Modelling languages
  - BPEL
  - BPMN

Some terminology

Concurrency vs Parallelism

Concurrency

compose "independent" stuff

~

deal with a lot of stuff AT ONCE

GOAL: "good" composition

DESIGN!

break down problems
&
compose the pieces

Parallelism

run stuff symultaneously

~

do a lot of stuff AT ONCE

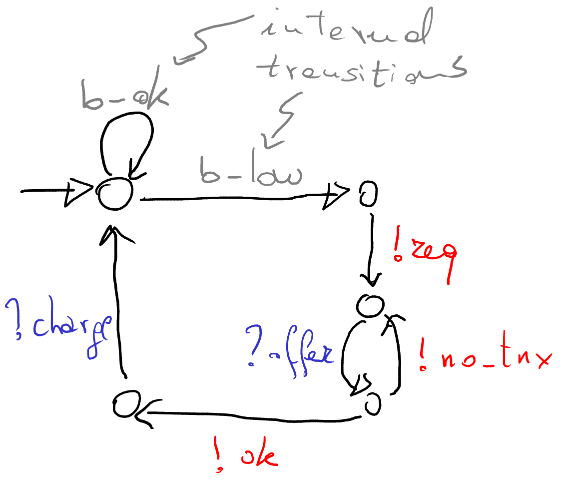
GOAL: "good" execution

PERFORMANCE

Communication Based Concurrency

< See slides with the example in Erlang>

Example: Some mobile robots need to manage their energy in order to accoplish their task (e.g., patrolling some premises). When their batteries deplete robots look for a recharge. Recharges are offered by recharge stations or other robots. We can model this behaviour using communicating machines. For instance, the behaviour of a robot seeking for a recharge is



(*)

Exercise 1 Give a communicating machine modelling the behaviour of a robot offering a recharge

Reflect about the compatibility between your solution and (*)

What do we mean by correct?

- SAFETY: "nothing bad happens"
  - If a number is printed then it is a positive prime less than 10^{10}
  - No deadlock
- LIVENESS: "something good eventually happens"
  - All robots looking for a recharge eventually find a charge station

BTW: One can think of sequential programs as multi-threaded ones with 1 thread only

But there're serious differences:

- testing is hard { poor reproducibility / failed tests hardly help bug localisation — heisenbugs
- non-determinism: blessing & curse

Modelling Behaviour

at some level of abstraction

The evolution of a system can be described in terms of state transitions
- states represent the possible configurations the system can be in
- transitions represent the possible evolution from a given configuration.

In its simplest form, such models can be mathematically rendered as binary relations

$Sys = (S, \rightarrow)$ where

- S is a set of states (aka configurations)
- $\rightarrow \subseteq S \times S$

$(s, s') \in \rightarrow$ "from the state s, Sys can evolve to s'"

Sys is deterministic if $\forall s, s', s'' \in S : \; s \rightarrow s', \; s \rightarrow s'' \implies s' = s''$

$$\begin{tikzcd} & s \arrow[dl] \arrow[dr] & \\ s' & = & s'' \end{tikzcd}$$

Of course this idea is hardly new and examples can be found in any book on automata or formal languages. Its application to the definition of programming languages can be found in the work of Landin and the Vienna Group [Lan,Oll,Weg].

[Lan] Landin, P.J. (1966) A Lambda-calculus Approach, Advances in Programming and Non-numerical Computation, ed. L. Fox, Chapter 5, pp. 97–154, Pergamon Press.
[Oll] Ollengren, A. (1976) Definition of Programming Languages by Interpreting Automata, Academic Press.
[Weg] Wegner, P. (1972) The Vienna Definition Language, ACM Computing Surveys 4(1):5–63.

Examples (Plotkin)

Finite Automata (finite)

$M = (Q, \Sigma, \delta, q_0, F)$

Q, Σ finite $\quad q_0 \in Q \quad F \subseteq Q$

$\delta \subseteq (Q \times \Sigma) \times Q \quad \delta : Q \times \Sigma \to 2^Q$

a corresponding TS is

- $Q \times \Sigma^*$
- $(q, w) \longrightarrow (q', w') \iff$

$\longrightarrow = \{((q, aw), (q', w)) \in (Q \times \Sigma^*)^2 \mid q' \in \delta(q, a)\}$