Introduction
to
Formal Methods (Part I)

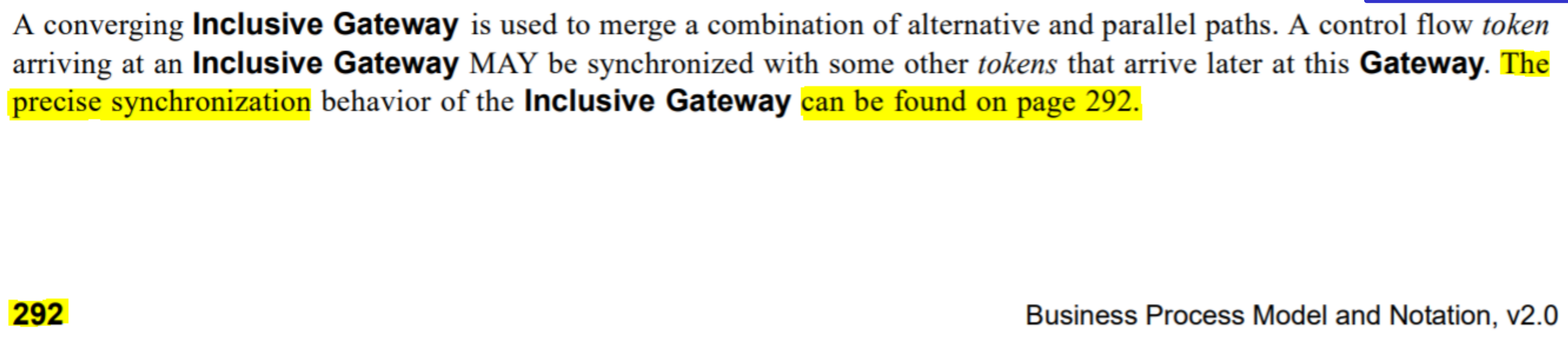
Academic year 2024/2025

.

Emilio Tuosto
emilio.tuosto@gssi.it
https://cs.gssi.it/emilio.tuosto

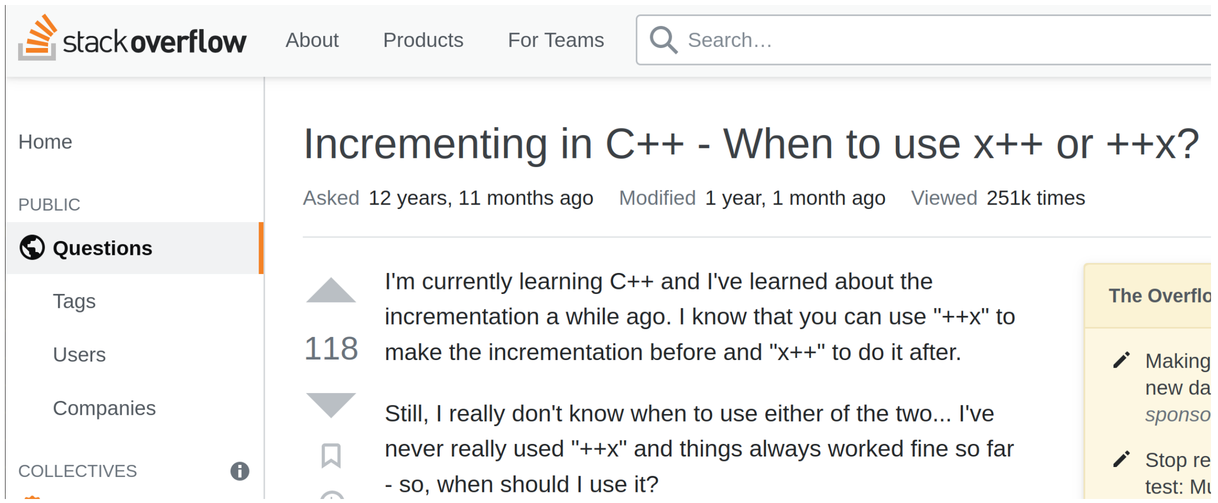
A couple of resasons to be rigorous

[https://www.omg.org/spec/BPMN/2.0/]


A converging **Inclusive Gateway** is used to merge a combination of alternative and parallel paths. A control flow *token* arriving at an **Inclusive Gateway** MAY be synchronized with some other *tokens* that arrive later at this **Gateway**. The precise synchronization behavior of the **Inclusive Gateway** can be found on page 292.

292 Business Process Model and Notation, v2.0

[https://stackoverflow.com/questions/1812990/incrementing-in-c-when-to-use-x-or-x]



Incrementing in C++ - When to use x++ or ++x?

Asked 12 years, 11 months ago Modified 1 year, 1 month ago Viewed 251k times

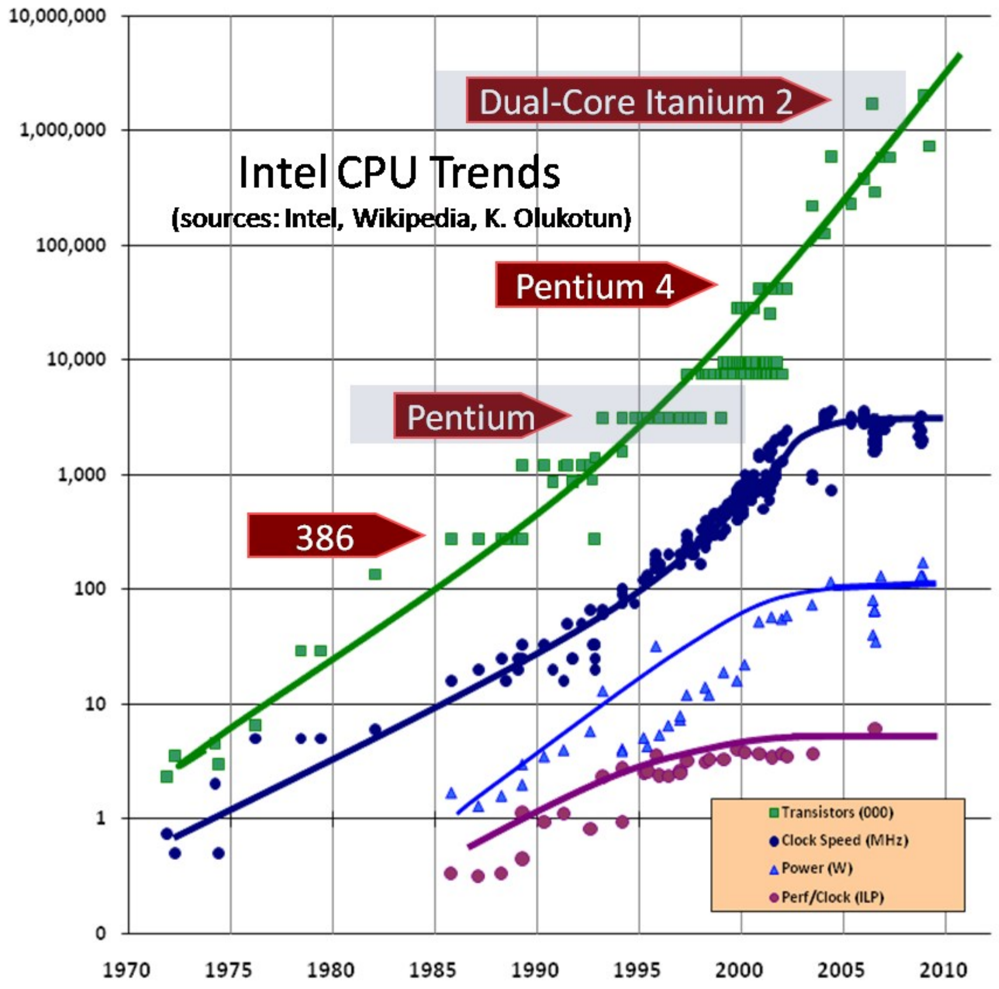
118

I'm currently learning C++ and I've learned about the incrementation a while ago. I know that you can use "++x" to make the incrementation before and "x++" to do it after.

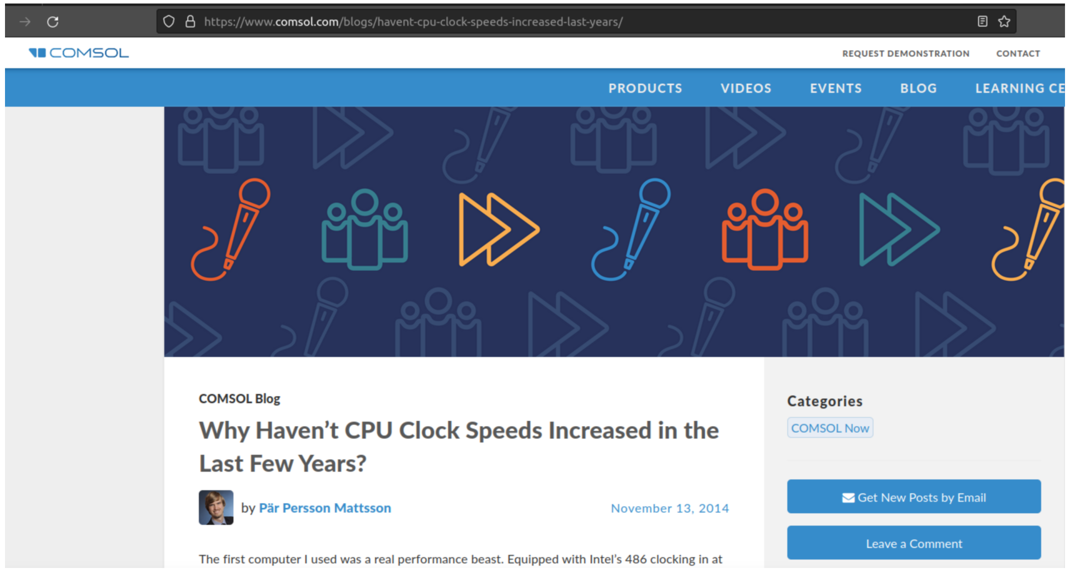
Still, I really don't know when to use either of the two... I've never really used "++x" and things always worked fine so far - so, when should I use it?

A reson to go concurrent

[https://i.extremetech.com/imagery/content-types/03zc6ghfKswe41smvPXi8Zh/images-6.jpg]



[https://www.comsol.com/blogs/havent-cpu-clock-speeds-increased-last-years]



COMSOL Blog

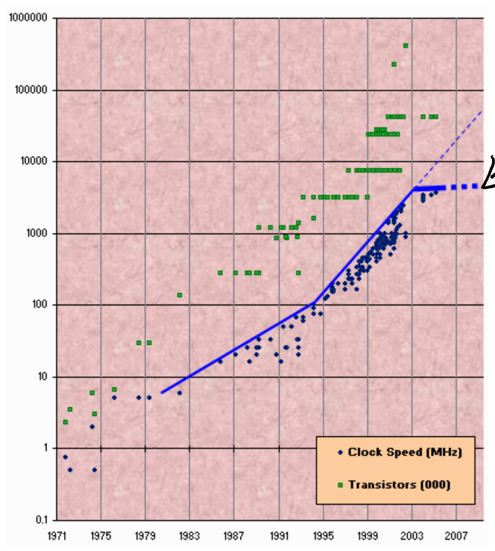
Why Haven't CPU Clock Speeds Increased in the Last Few Years?

by Pär Persson Mattsson November 13, 2014

The first computer I used was a real performance beast. Equipped with Intel's 486 clocking in at

The art _ multi-processor programming

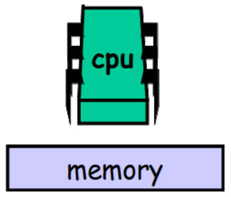
Hw $\xrightarrow{\text{Efficiency is no longer an "hw thing"}}$ Sw



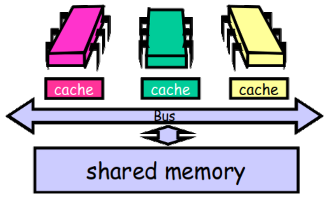
clock speed

# transistors grow by a factor of 10 every 10 years!

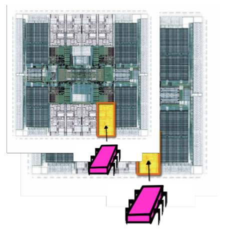
CPU speed is plateauing



uniprocessor



shared_memory processor



multicore

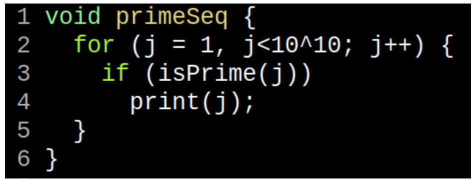
- programming constructs everywhere
  - "new" languages
    - Golang
    - Scala
    - Erlang/Elixir
    - Ballerina
    - Concurnas
- libraries (AKKA)
- Modelling languages
  - BPEL
  - BPMN

Job interviews and prime numbers

"On the first day of your new job, your boss asks you to ~~find~~ print all primes between 1 and 10^10 (never mind why), using a parallel machine that supports ten concurrent threads. This machine is rented by the minute, so the longer your program takes, the more it costs. You want to make a good impression. What do you do?"

[Herlihy, Shavit: The Art of Multiprocessor Programming. Elsevier, 2012.]

An example of shared memory concurrency



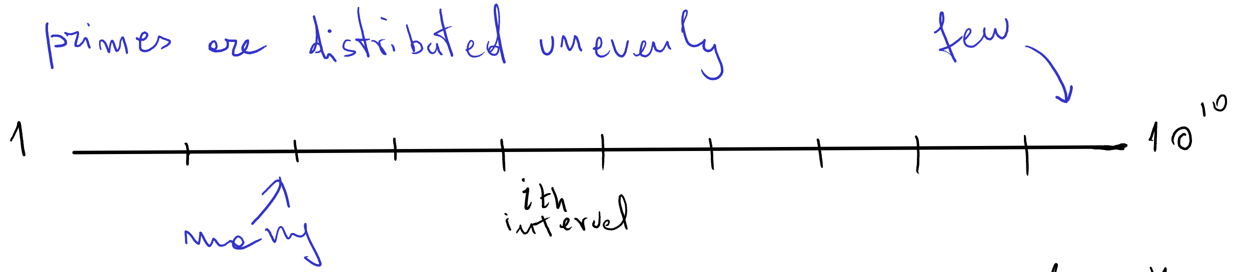
```
void primeSeq {
  for (j = 1, j<10^10; j++) {
    if (isPrime(j))
      print(j);
  }
}
```

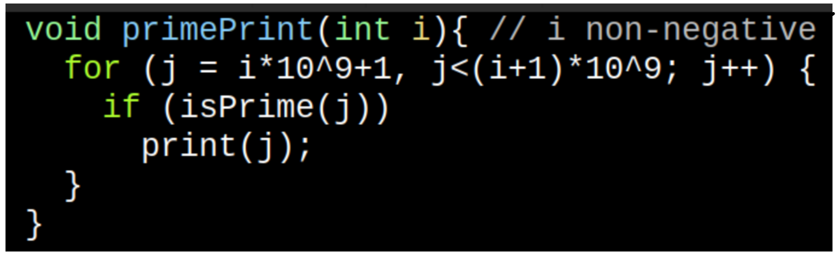
Printing all the prime numbers below 10^{10} ... sequentially

Now let's try concurrently

Split the interval & lauch a thread on each position

primes are distributed unevenly

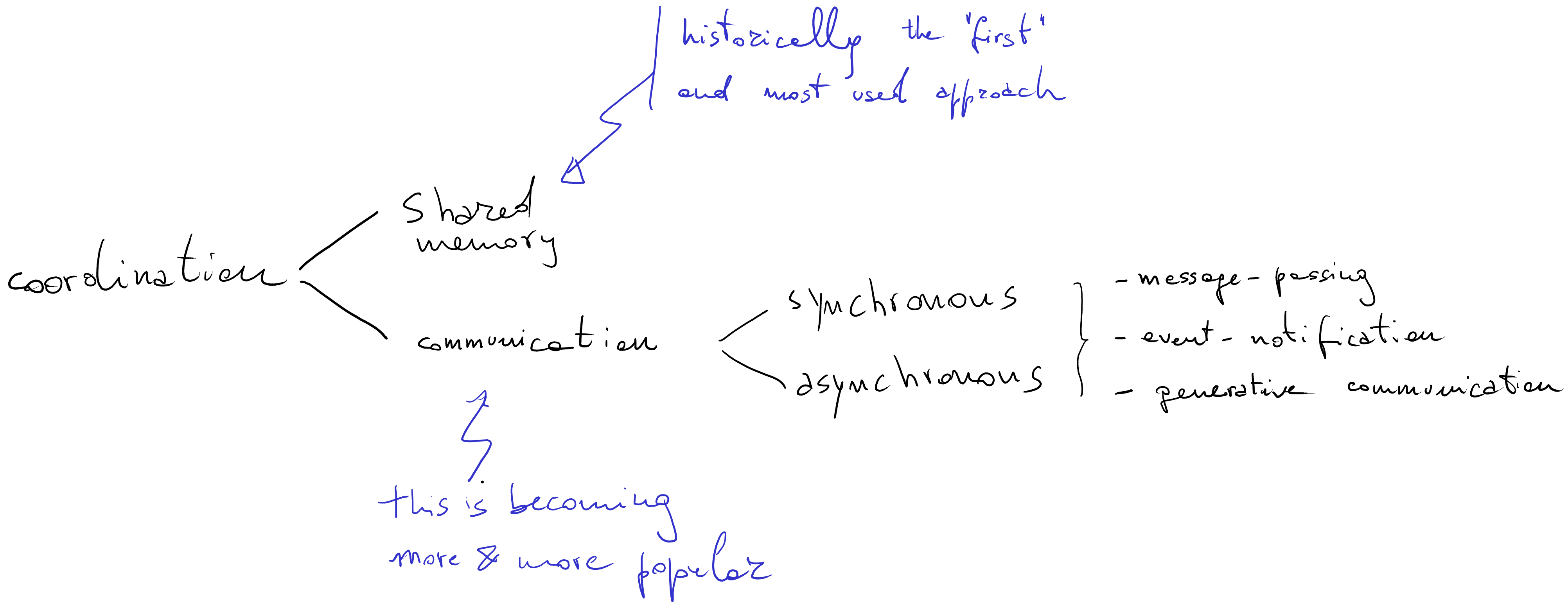




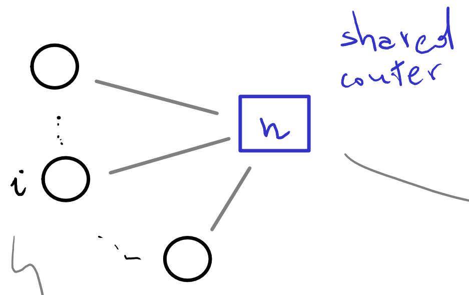
```
void primePrint(int i){ // i non-negative
  for (j = i*10^9+1, j<(i+1)*10^9; j++) {
    if (isPrime(j))
      print(j);
  }
}
```

How good is this idea?

- uneven load
- should we look for an optimal split?


historically the "first" and most used approach
coordination
shared memory
communication
synchronous
asynchronous
- message-passing
- event-notification
- generative communication
this is becoming more & more popular

Exercise 0 Find a better multi-threaded program for printing the first 10^{10} primes



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

RACES

THIS IS NOT GOOD!

synchronised!

```
public class Counter {
  private long value;

  public long getAndIncrement() {
    return value++;
  }
}
```

temp := value
value ++
return temp

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

even better

WHY?

REFLECT about why this solution is better than splitting

Some terminology

Concurrency vs Parallelism

compose "independent" stuff

~

deal with a lot of stuff
AT ONCE

GOAL: "good" composition

DESIGN!

break down problems
&
compose the solutions

run stuff symoltaneously

~

do a lot of stuff
AT ONCE

GOAL: "good" execution

PERFORMANCE