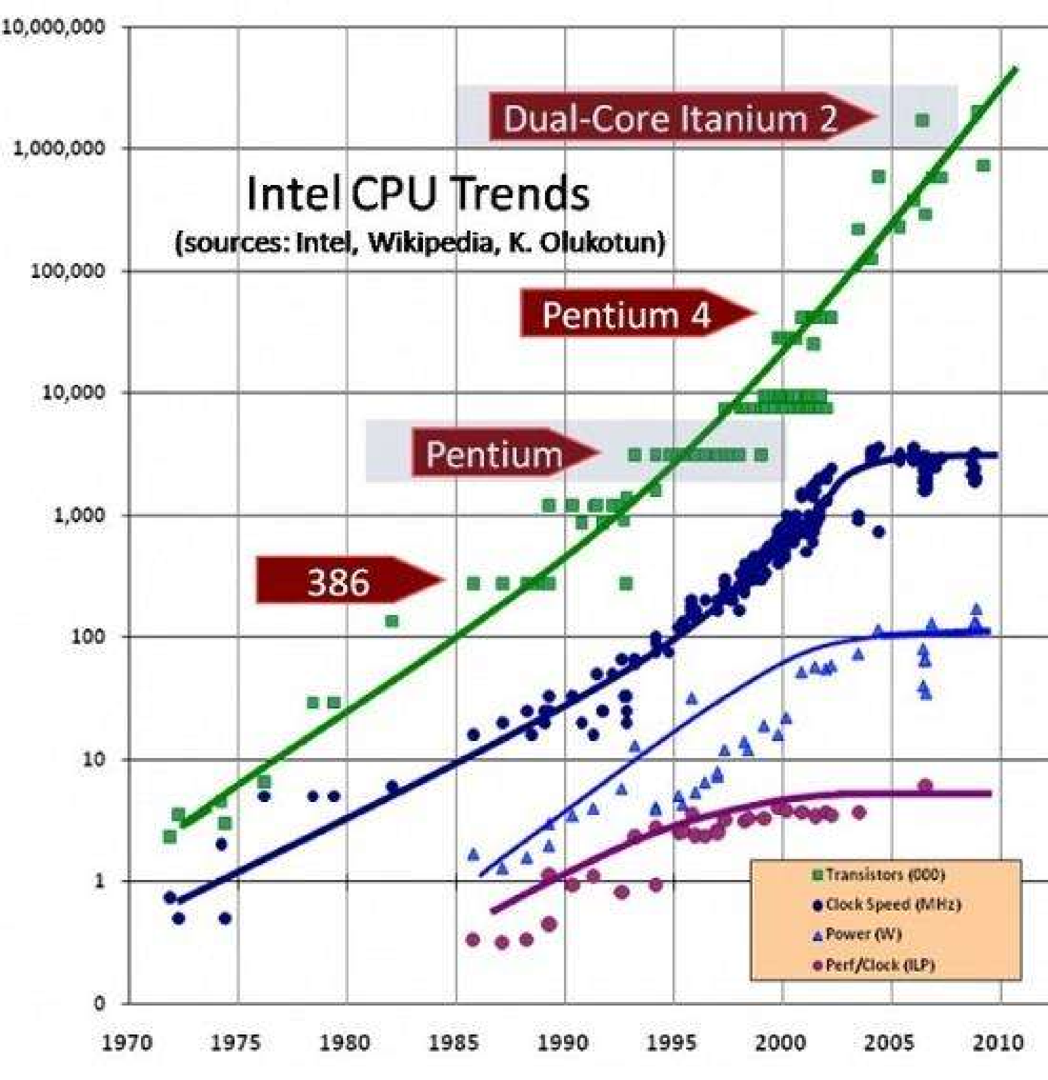
Le C++ à la rescousse du Raspberry Pi 3

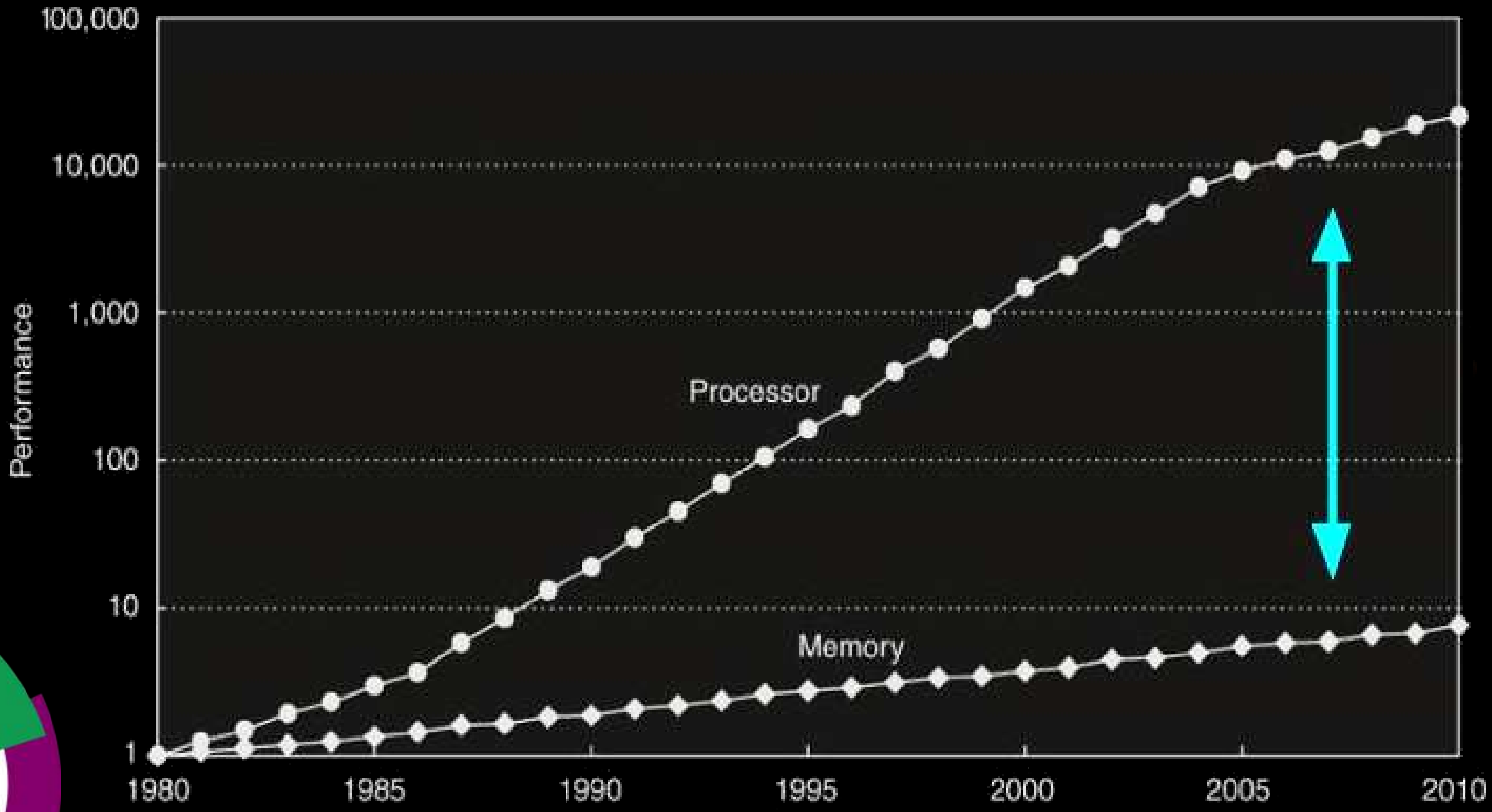
More transistors but same:

- Clock speed
- Power consomption
- Instructions per cycle



Intel CPU Trends

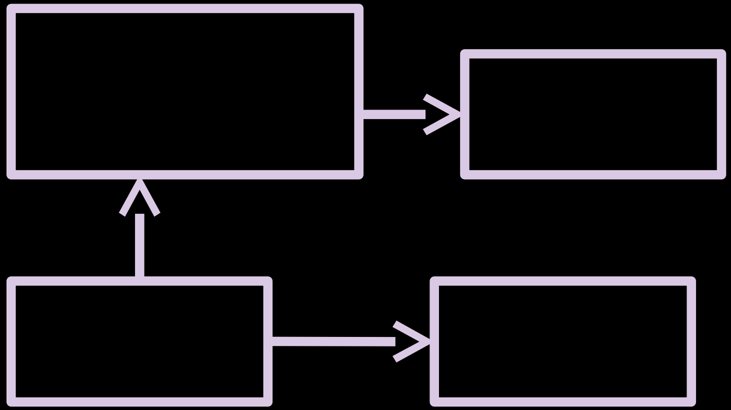
(sources: Intel, Wikipedia, K. Olukotun)


100,000
10,000
1,000
100
10
1
Performance
Processor
Memory
1980
1985
1990
1995
2000
2005
2010

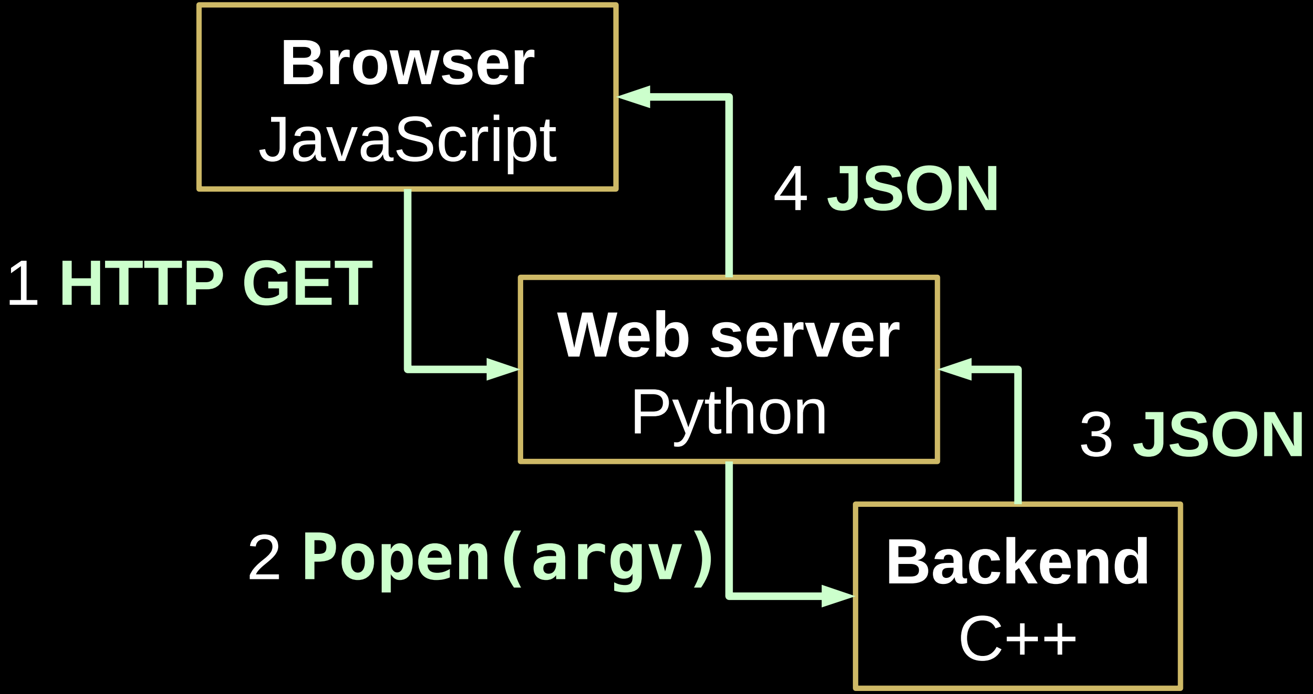
	Raspberry pi BCM2837 64 bit	**Intel i5**
Price	< 35 $	200 $
Frequency	1,2 GHz	2,66 GHz
Cores	4 (ARM Cortex-A53)	4
L1 cache	512KB shared	per core: 32KB data 32KB instructions
L2 cache		256 KB /core
L3 cache		8 MB shared
RAM	1 GB	1 TB …
Consomption	1,5W idle 6,7W under stress	90W

C++ side = Boxes and links position

Algo	Description
Bombix	Computes geometry of links
Latuile	Computes translation of boxes



Implementation



Browser
JavaScript

1 **HTTP GET**

Web server
Python

2 **`Popen(argv)`**

Backend
C++

3 **JSON**

4 **JSON**

Web server in few python lines

```
import http.server
import socketserver

handler = http.server.SimpleHTTPRequestHandler
httpd = socketserver.TCPServer(("", 8080), handler)
httpd.serve_forever()
```

or command line:

```
python -m http.server 8080
```

HTTP GET URL encodes the input

Rectangle	Width	hexa	Height	hexa
1	52	0x**34**	36	0x**24**
2	68	0x**44**	45	0x**2D**
...	...	...	...	...

Link	From	hexa	To	hexa
lk1	5	0x**05**	2	0x**02**
lk2	1	0x**01**	4	0x**04**
...	...	...	...	...

URL : **3424442D05020104**

C++ called as a command

```
(argv[], printf(json)) as (Input,Output)
```

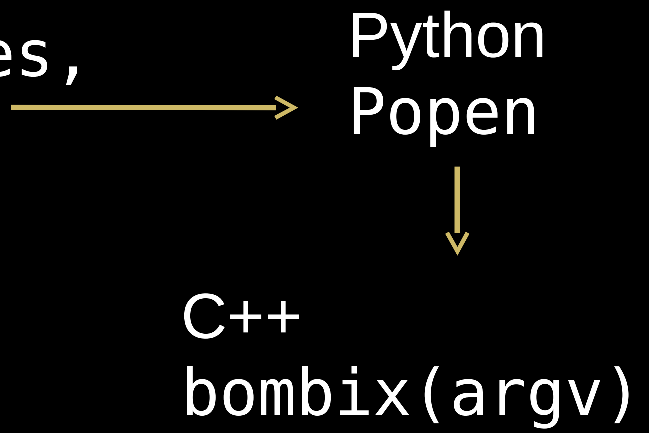
```
argv[]={
   bombix,
   --rectangles,
   3424442D,
   --links,
   05020104
}
```

Bombix Latuile

JSON

Python parses the URL, and run the command using Popen(argv[])

```
argv[]={
   bombix,
   --rectangles,
   342442D,
   --links,
   05020104
}
```



Python Popen

C++ bombix(argv)

Command returns JSON
Python forwards it (HTTP)
JavaScript receives a response

Performance drop Pi vs i5

Algo	**Pi** GCC-5.4	**i5** MSVC-2015
Bombix	1 mn 30	0.05 sec
Latuile	15 sec	0.05 sec

Replacing hash table by vector

Before	After	Benefit	Drawback
1 mn 30	1.5 sec	Faster access (no need to compute hash). Better cache coherence.	Vector size must be known and cannot be infinity

OpenMP multi-threading

Before	After	Comment	Benefit	Drawback
1.5 sec	0.5 sec	Good result for macro jobs. Can things be computed in parallel ?	Very light impact on code structure compared to sequential.	Overhead makes it not suitable to run small jobs in parallel

Function inlining

Before	After	Benefit	Drawback
15 sec	8 sec	Faster than function call	Executable size might increase

Struct size reduction

Before	After	Comment	Benefit	Drawback
8 sec	**9 sec**	Replace `int` by `int8_t`	Struct requires less memory. Easier to store in cache.	CPU performs operation on 32 or 64 bit integers only

Cache coherence

Before	After	Comment	Benefit
7 sec	6 sec	Example: array passed to `std::push_heap()` and `std::pop_heap()`	Isolation of hot information

Overview of optimization gains

Technique	Before	After
OpenMP	1.5 sec	0.5 sec
Hash table --> vector	1 mn 30	1.5 sec
Function inlining	15 sec	8 sec
Struct size reduction	8 sec	**9 sec**
Cache coherence	7 sec	6 sec

Performance drop Pi vs i5

Algo	Pi before	Pi after	i5
Bombix	1 mn 30	0.5 sec	0.05 sec
Latuile	15 sec	6 sec	0.05 sec