www.bsc.es

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

ARM-based systems and software stack for HPC

Workshop on Computational Science Infrastructure and Applications for Academic Development

5 Oct 2015 - Trieste

Filippo Mantovani

With the contribution of:

Dani Ruiz, Luna Backes, Oriol Vilarrubi, Nikola Rajovic

SESSION 1
Introduction to ARM-based scientific computing

Outline

- Brief introduction to supercomputing
  - For what do we use supercomputers?
  - Difference between a PC and a supercomputer
  - How to “describe” a supercomputer: metrics
- Analyzing history of HPC with different metrics
  - From vector CPUs to commodity components
- What is commodity nowadays?
  - Overview of current trends for mobile CPUs
- Prototypes based on mobile embedded technology @ BSC
  - Mini-clusters
  - The Mont-Blanc prototype
- Looking ahead – Mont-Blanc project

Outline

- Brief introduction to supercomputing
  - For what do we use supercomputers?
  - Difference between a PC and a supercomputer
  - How to “describe” a supercomputer: metrics
- Analyzing history of HPC with different metrics
  - From vector CPUs to commodity components
- What is commodity nowadays?
  - Overview of current trends for mobile CPUs
- Prototypes based on mobile embedded technology @ BSC
  - Mini-clusters
  - The Mont-Blanc prototype
- Looking ahead – Mont-Blanc project

For what do we use supercomputers?

- Classical scientific method
  - Make experiments that are reproducible
  - Collect measurements during the experiments
  - Use the measurements for (in)validate the scientific theories
    - Hypothesis
    - Prediction
    - Testing
    - Analysis
- There are theories where experiments are “difficult”
  - Different scale (astronomy)
  - Too expensive (zero gravity)
  - Wide space phase (drugs docking)
  - …

Difference between a PC and a supercomputer

Your PC is like a standard car:

~140 km/h
~20 km/l

- Comfortable
  - graphical interface
  - screen, mouse, touch pad, …
  - printer, speaker
- Versatile
  - Can play music
  - Can connect to internet
  - Can do a bit of scientific computation
- Largely available
  - Cheap
  - OK, I assume you do not have a Ferrari…

A supercomputer is like a F1 car:

~300 km/h
~2 km/l

- Oriented to performance
  - Execution of floating point operations is the main target
  - No graphical interfaces, mouse, etc…
  - Only remotely accessible
- Very specialized
  - Small sets of workloads
  - Scientific simulations
  - Parallel computation
- Produced in few units
  - Very expensive

How to “describe” a supercomputer

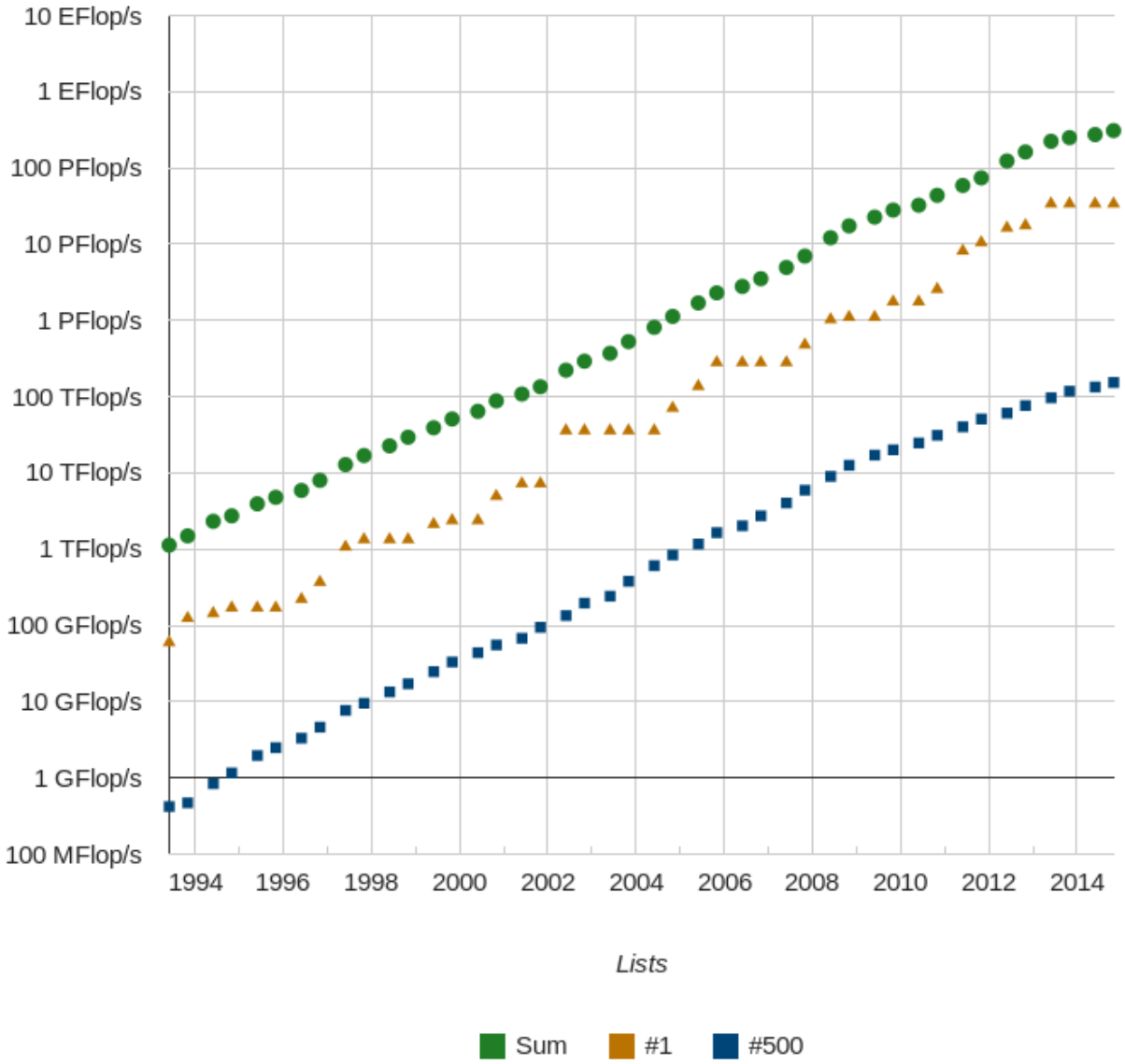
- Components
  - Hardware
    - CPU
    - Accelerator (GPU, MIC, FPGA)
    - Memory
    - Network
    - Storage
    - Cooling
    - Integration
  - Software
    - Operating System
    - Driver
    - Libraries
    - Compilers
    - Cluster management
- Metrics/Categories
  - Floating point performance [FLOPS]

  - Power efficiency [FLOPS/W]

  - Cost efficiency [FLOPS/$]
  - Others

~ … GFLOPS/W
~ … GFLOPS/$

Top500 evolution: pure performance metric



The metric of the power efficiency

- Reorder of the 500 most powerful supercomputers
- Using the same benchmark (HPL)
- Sorted by power efficiency

FLOPS/W

FLoating point Operations Per Seconds
per unit of power (Watt)

A bit similar to km/l in the cars!

The metric of the cost efficiency

Take the total price of a supercomputer and divide it by its performance:

FLOPS/$

FLoating point Operations Per Seconds per Price

NOTE:

- There is no “official” ranking for this metric
- Introduced for this talk: does not pretend to be “smart”

Similar to the cost of the speed: You can go faster or slower wasting more or less money (ignoring gasoline)…

Outline

- Brief introduction to supercomputing
  - For what do we use supercomputers?
  - Difference between a PC and a supercomputer
  - How to “describe” a supercomputer: metrics
- Analyzing history of HPC with different metrics
  - From vector CPUs to commodity components
- What is commodity nowadays?
  - Overview of current trends for mobile CPUs
- Prototypes based on mobile embedded technology @ BSC
  - Mini-clusters
  - The Mont-Blanc prototype
- Looking ahead – Mont-Blanc project

In the beginning... there were only supercomputers

Availability:

- Very few of them
  (why is called top500?)
- Very expensive

Market:

- Some units sold
- Very few companies
  (market mostly “drugged” by states)

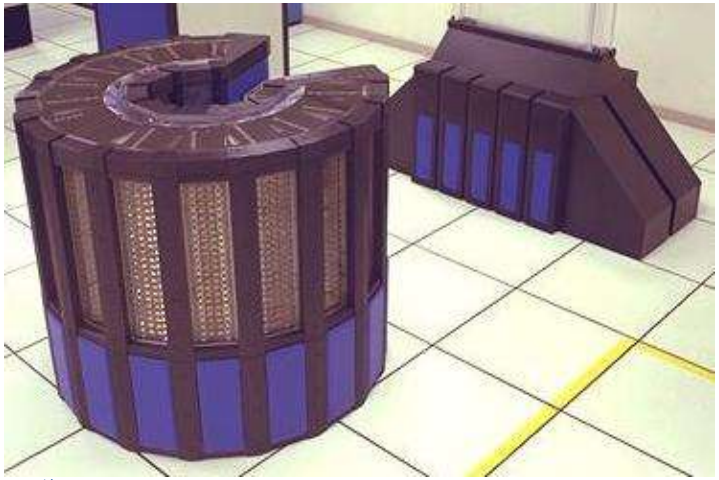
Features:

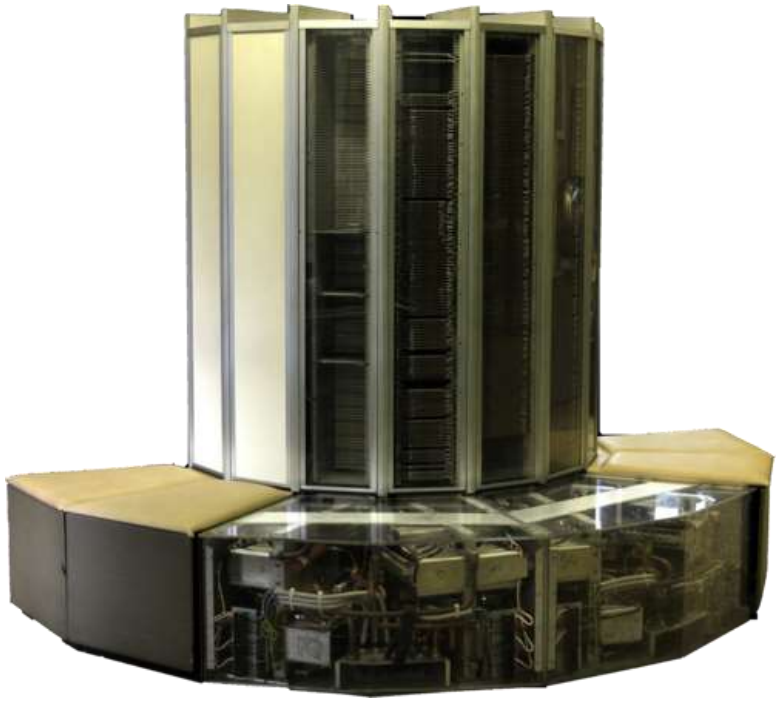
- Special purpose hardware
- Vector operations/processors

Examples:

- Cray-1
  1975 - 160 MFLOPS, 80 units, 5-8 M$, 115 kW
- Cray X-MP
  1982, 800 MFLOPS
- Cray-2
  1985, 1.9 GFLOPS
- Cray Y-MP
  1988, 2.6 GFLOPS

~ 1.4 KFLOPS/W
~ 10^{-5} KFLOPS/$





Then, commodity took over special purpose

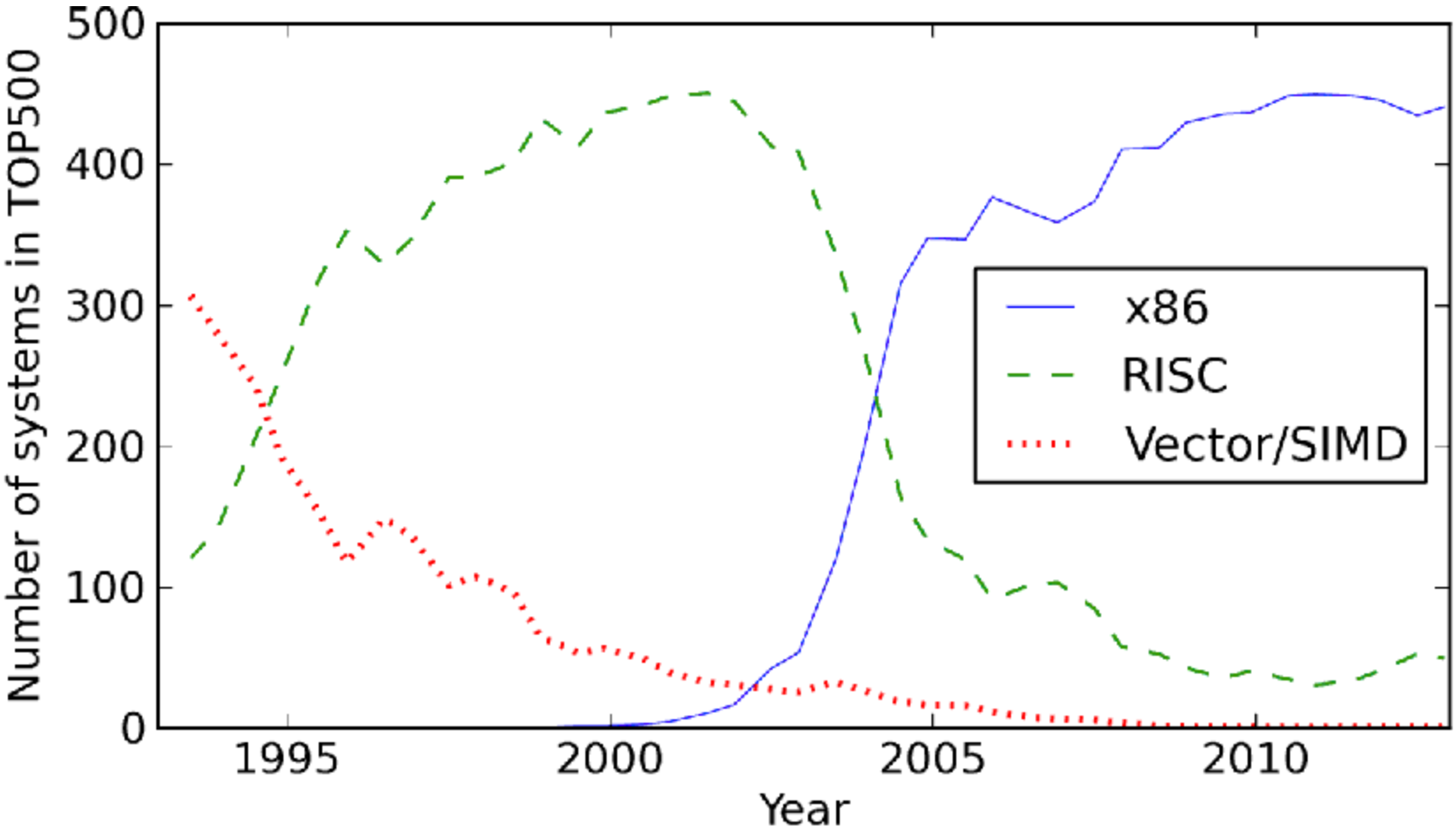
- **1997 - ASCI Red, Sandia**
  - 1 TFLOPS, 9298 cores @ 200 MHz Intel Pentium Pro, 850 kW
  - upgraded to Pentium II Xeon in 1999 (3.1 TFLOPS).

- **2001 - ASCI White, LLNL**
  - 7.3 TFLOPS, 8192 cores @ 375 MHz, IBM Power 3
  - 110 M$
  - 3 MW for computational power
  - 3 MW for cooling

~ 1.2 GFLOPS/W
~ 10^{-5} GFLOPS/$

From **vector parallelism** to **message passing** programming models...

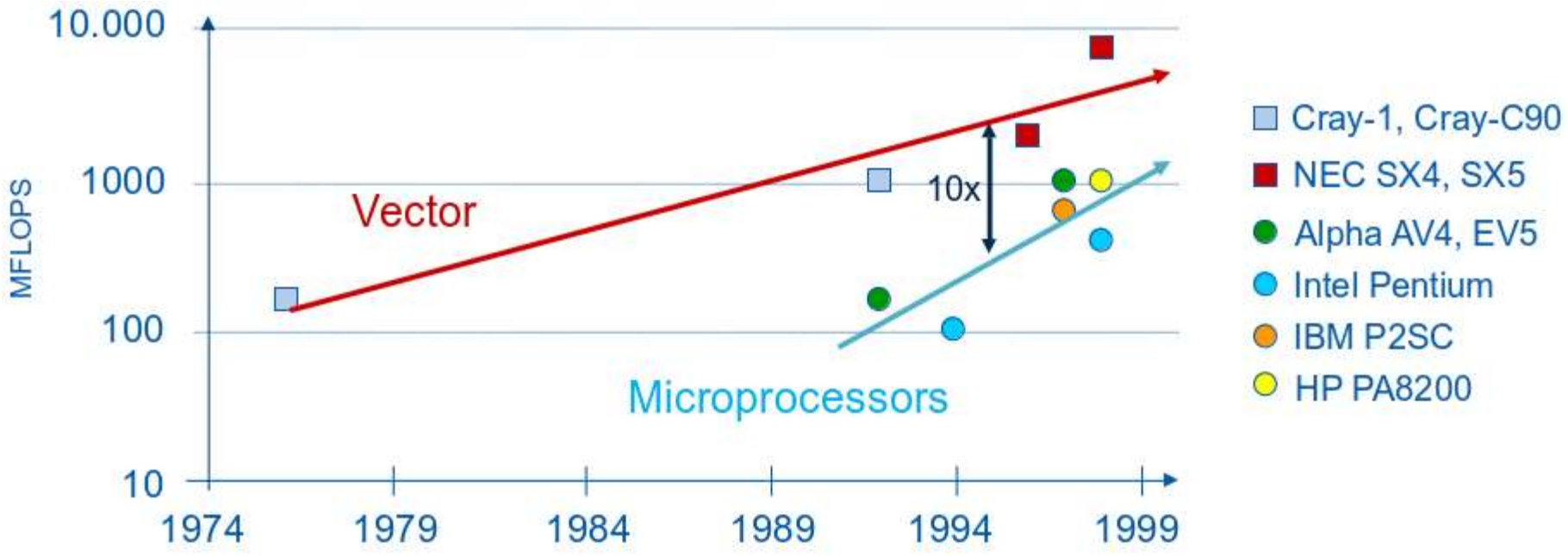
And now commodity components drive HPC



- RISC processors replaced vectors
- x86 processors replaced RISC
  Vector processors survive as (widening) SIMD extensions

Commodity hardware on stage

- Microprocessors killed the vector-based supercomputers
  They were not faster but they were significantly **cheaper** and **greener**
- 10 microprocessors ≈ 1 vector CPU
  SIMD vs. MIMD programming paradigms



Commodity hardware + commodity software

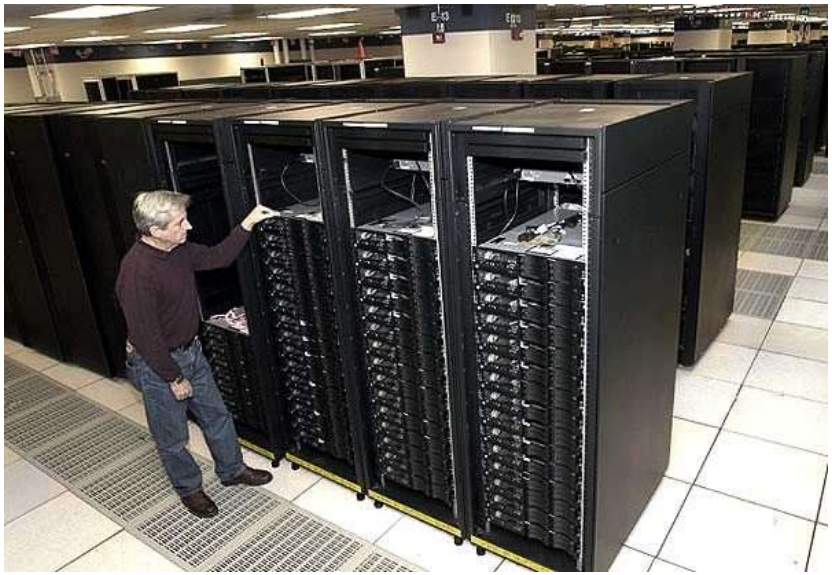
- MareNostrum
  - Nov 2004, #4 Top500
    - 20 TFLOPS, Linpack
    - 31 TFLOPS, Peak
    - 3564 cores
    - 1 MW + 1 MW
  - IBM PowerPC 970 FX
    - Blade enclosure
  - Myrinet + 1 GbE network
  - SuSe Linux

~ 0.01 GFLOPS/W
~ 0.002 GFLOPS/$

Even more commodity with game technology

- Los Alamos National Laboratory (USA)
- Hybrid architecture
  - 1 x AMD dual-core Master blade
  - 2 x PowerXCell 8i Worker blade
- 296 racks
  - 6.480 Opteron processors
  - 12.960 Cell processors
    - 128-bit SIMD
- Infiniband interconnect
  - 288-port switches
- 2.35 MWatt

2008: First PFLOPS supercomputer
IBM RoadRunner



0.425 GFLOPS/W
0.0137 GFLOPS/$

2009: 1.75 PFLOPS - Cray Jaguar

- Oak Ridge National Laboratory (USA)
- 230 racks
- 224.256 AMD Opteron processors
  - 6 cores / chip
- Cray Seastar2+ interconnect
  - 3D-mesh using AMD Hypertransport
- 7 MWatts
- 104 M$

0.257 GFLOPS/W
0.0167 GFLOPS/$

Commodity market driven (once more, game market)

- DOE/SC/Oak Ridge National Laboratory
  - Jaguar GPU upgrade

2012: 17.6 PFLOPS
Cray Titan

- 200 racks
- 224.256 Cray XK7 nodes
  - 16-core AMD Opteron
  - Nvidia Tesla K20X GPU

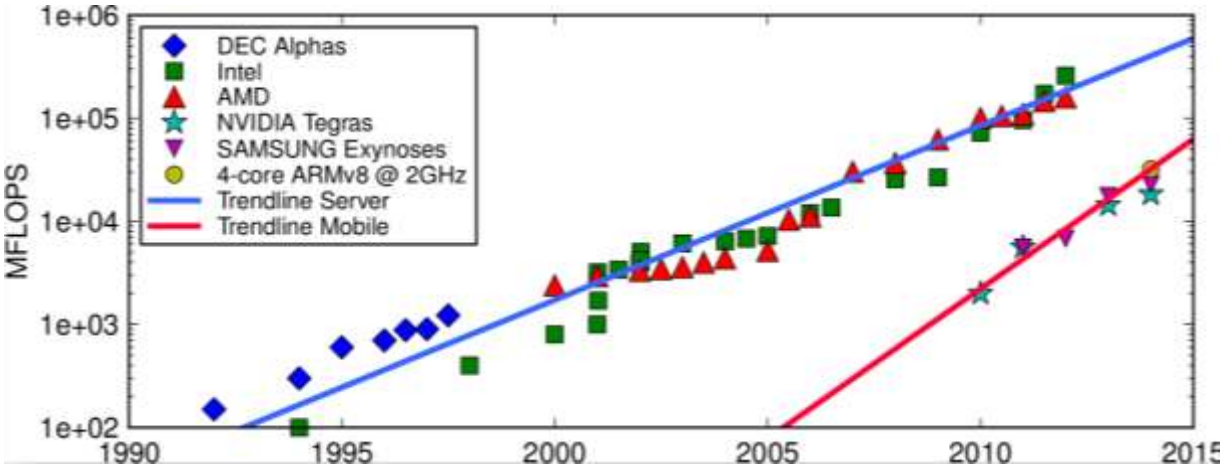
- 8.2 MWatts
- 97 M$

2.1 GFLOPS/W
0.18 GFLOPS/$

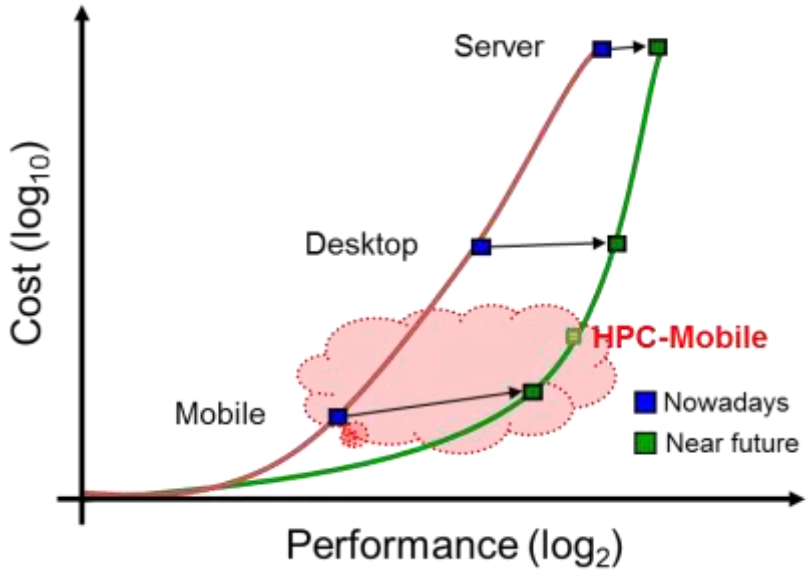
Outline

- Brief introduction to supercomputing
  - For what do we use supercomputers?
  - Difference between a PC and a supercomputer
  - How to “describe” a supercomputer: metrics
- Analyzing history of HPC with different metrics
  - From vector CPUs to commodity components
- What is commodity nowadays?
  - Overview of current trends for mobile CPUs
- Prototypes based on mobile embedded technology @ BSC
  - Mini-clusters
  - The Mont-Blanc prototype
- Looking ahead – Mont-Blanc project

Next commodity in the chain?



...and we are still ignoring tablets: >200M



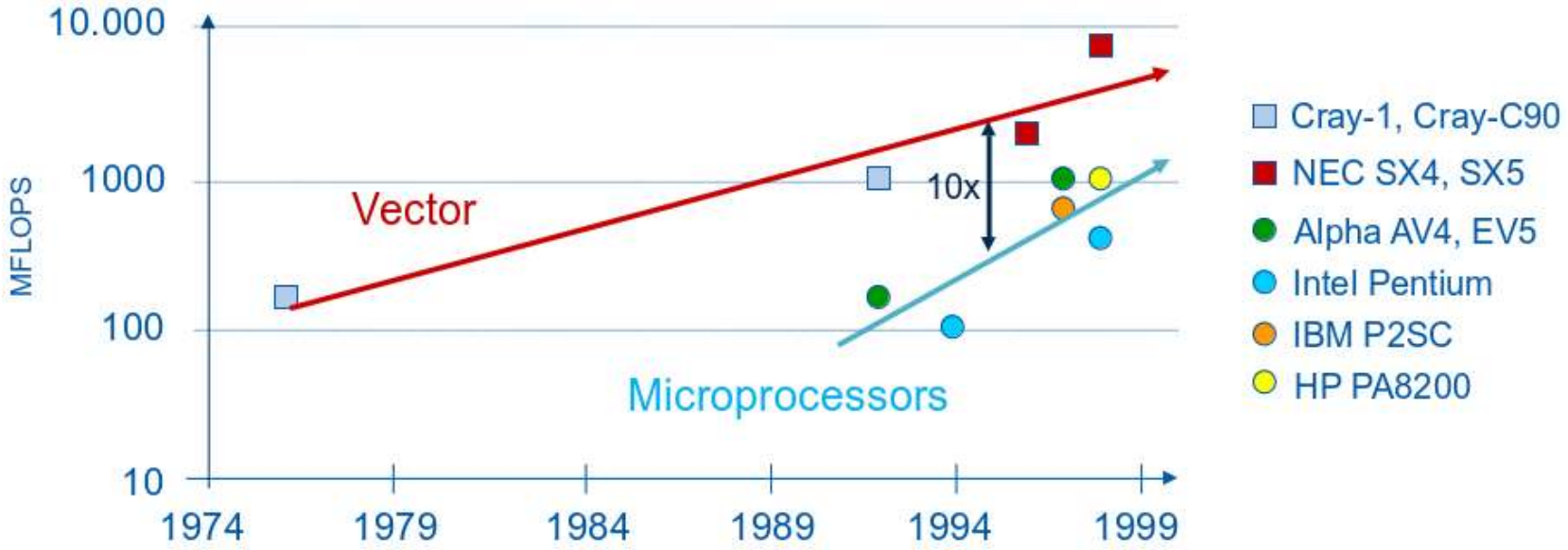
HPC
Jun 2015: 25 M cores

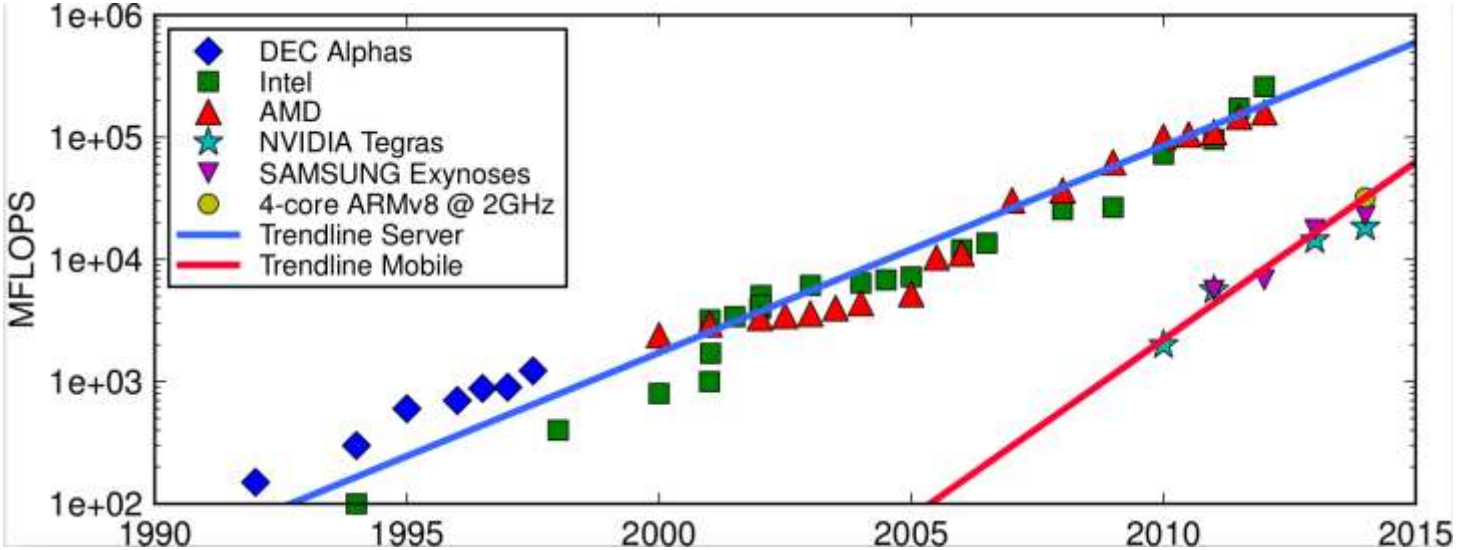
Server (+3%)
2013: 9.0 M
2014: 9.3 M

PC (-1 %)
2013: 316 M
2014: 314 M

Smartphone (+30%)
2013: 1000 M
2014: 1300 M

In case the history repeats itself, we want to be ready…





Performance vs. price

153 GFLOPS — 1500$

30x

5.2 GFLOPS — ~20$

70x

10x

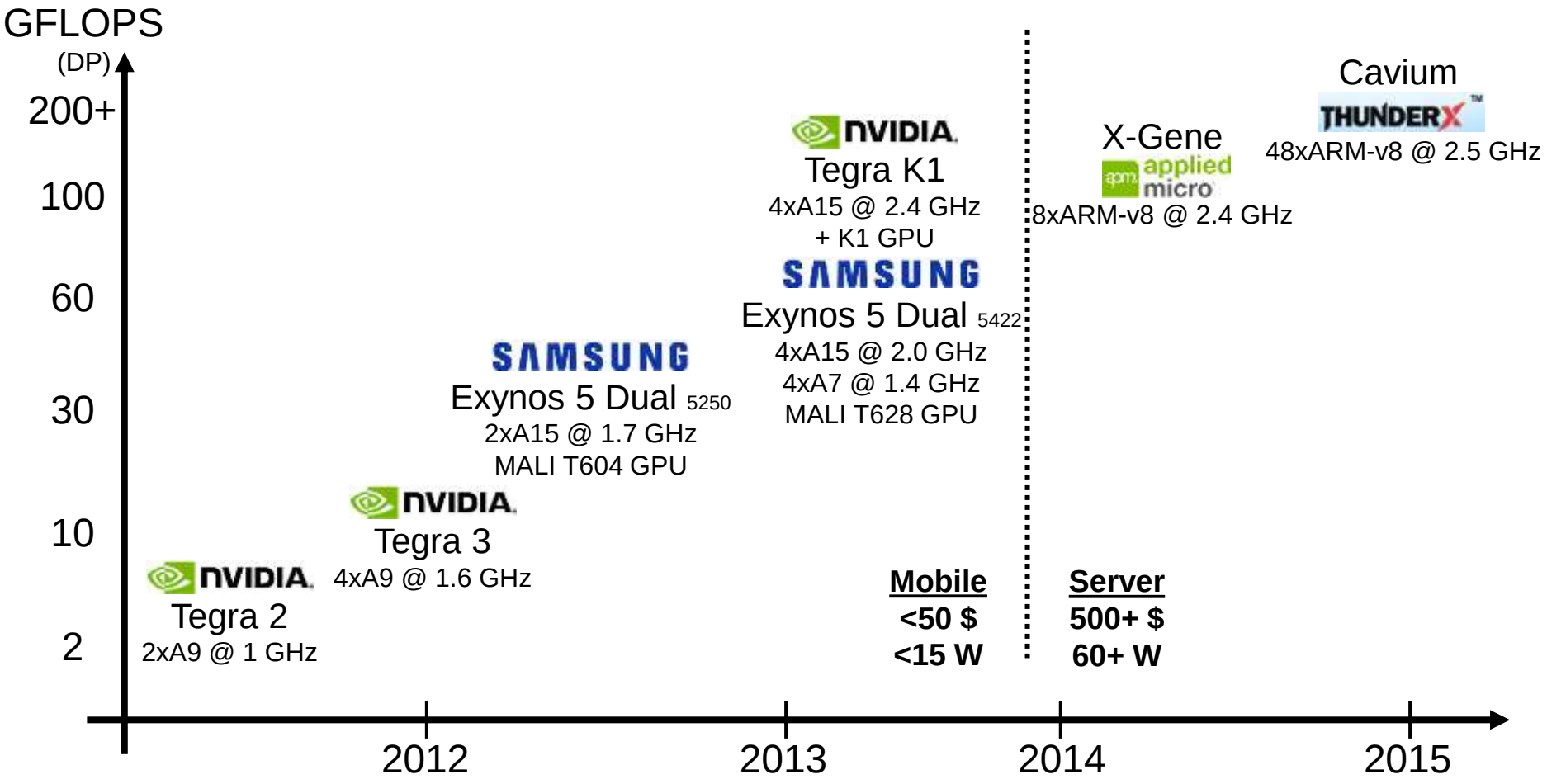
15.2 GFLOPS — ~20$

70x ?

5x

7+25 GFLOPS — ~20$

Cheap technology evolving fast



WARNING: Having an ISA or an IP licensed does not guarantee that you will have a chip!

Disclaimer: Numbers from public sources

Outline

- Brief introduction to supercomputing
  - For what do we use supercomputers?
  - Difference between a PC and a supercomputer
  - How to “describe” a supercomputer: metrics
- Analyzing history of HPC with different metrics
  - From vector CPUs to commodity components
- What is commodity nowadays?
  - Overview of current trends for mobile CPUs
- Prototypes based on mobile embedded technology @ BSC
  - Mini-clusters
  - The Mont-Blanc prototype
- Looking ahead – Mont-Blanc project

The BSC ARM-based prototype ecosystem

Prototypes are critical to accelerate software development

System software stack + applications

PRACE PRACE PRACE

MONT BLANC MONT BLANC

Tibidabo:
ARM multicore

Carma:
ARM +
external
mobile GPU

Pedraforca:
ARM +
HPC GPU

Arndale:
ARM + embedded GPU

Odroid:
ARM bigLITTLE
In-kernel switcher

Odroid Octa:
ARM bigLITTLE
Heterogeneous
multi-processing

NVIDIA Jetson
ARM 4+1 + K1 GPU

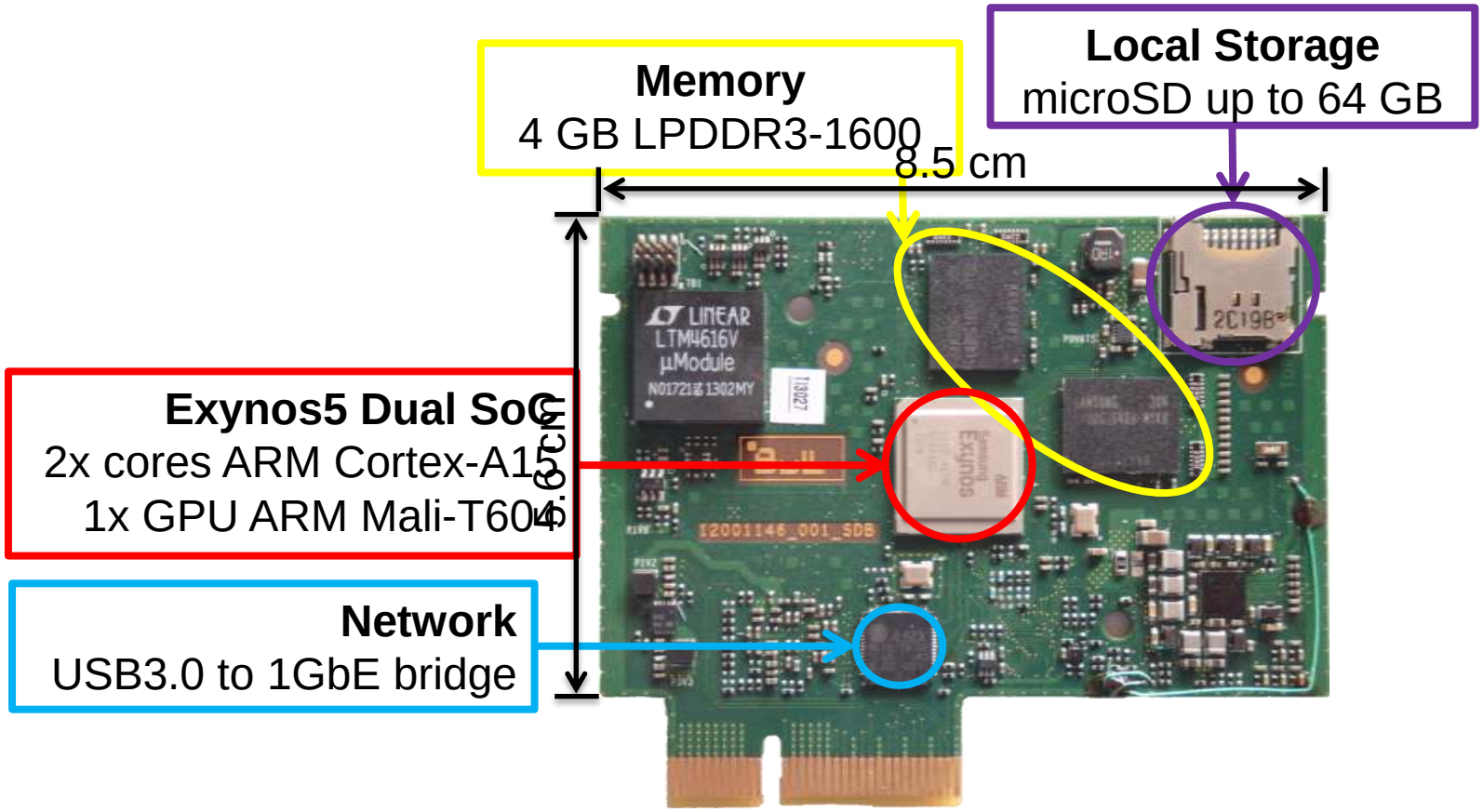
Mont-Blanc protoype:

2011 2012 2013 2014 2015

Mont-Blanc Server-on-Module (SoM)

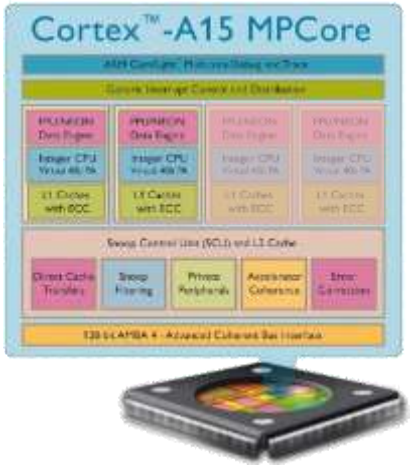
CPU + GPU + Memory + Local Storage + Network

Form factor: 8.5 x 5.6 cm



Exynos 5 Dual: ARM + GPU platform

- Dual-core ARM Cortex-A15 @ 1.7 GHz
  - VFP for 64-bit Floating Point
    - 6.8 GFLOPS (1 FMA / cycle)
  - NEON for 32-bit floating point SIMD
- Quad-core ARM Mali T604
  - Compute capable
    - **OpenCL 1.1**
    - 68 GFLOPS (SP)
    - 25.5 GFLOPS (DP)
- Shared memory between CPU and GPU



~1.6 GFLOPS/$ *

*** Cost efficiency of the chip only**

The Mont-Blanc prototype

Exynos 5 compute card
2 x Cortex-A15 @ 1.7GHz
1 x Mali T604 GPU
6.8 + 25.5 GFLOPS
15 Watts
2.1 GFLOPS/W

Carrier blade
15 x Compute cards
485 GFLOPS
1 GbE to 10 GbE
300 Watts
1.6 GFLOPS/W

Blade chassis 7U
9 x Carrier blade
135 x Compute cards
4.3 TFLOPS
2.7 kWatts
1.6 GFLOPS/W

Rack
8 BullX chassis
72 Compute blades
1080 Compute cards
2160 CPUs
1080 GPUs
4.3 TB of DRAM
17.2 TB of Flash

35 TFLOPS
24 kWatt

	Mont-Blanc [GFLOPS/W]	Green500 [GFLOPS/W]
Nov 2011	0.15	2.0
Nov 2014	1.5	5.2

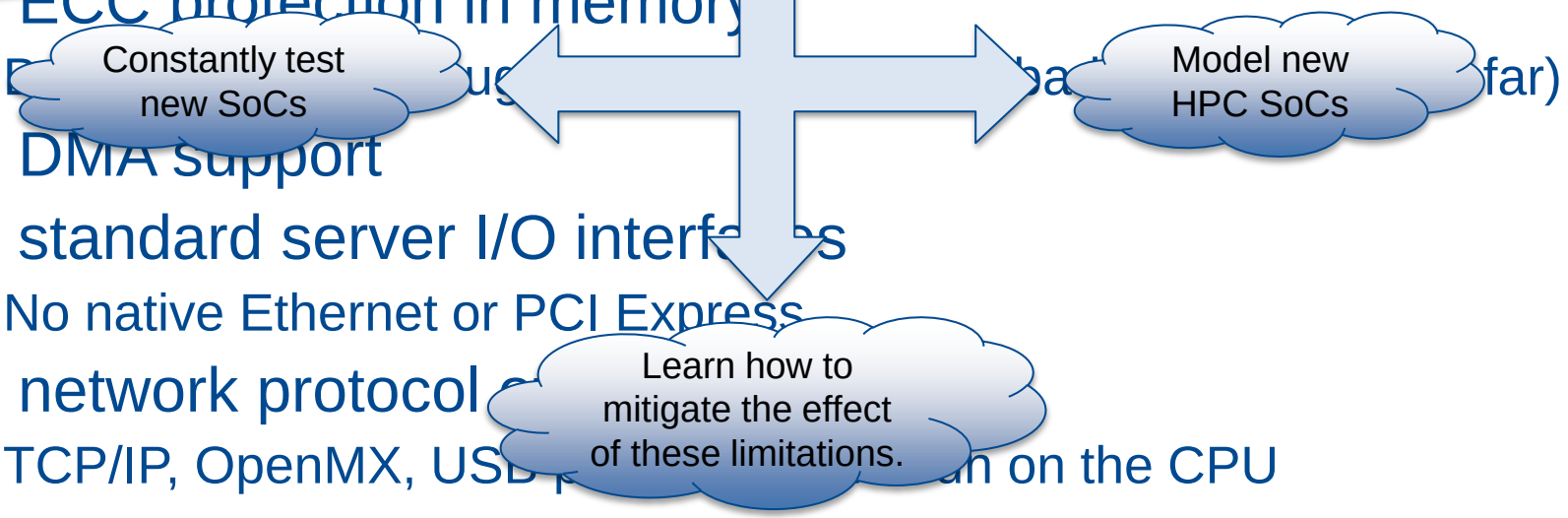
~10x (Mont-Blanc, Nov 2011 → Nov 2014)
~10x (Nov 2011, Mont-Blanc → Green500)
~3.5x (Nov 2014, Mont-Blanc → Green500)
~2.5x (Green500, Nov 2011 → Nov 2014)

Everything perfect then? Not really…

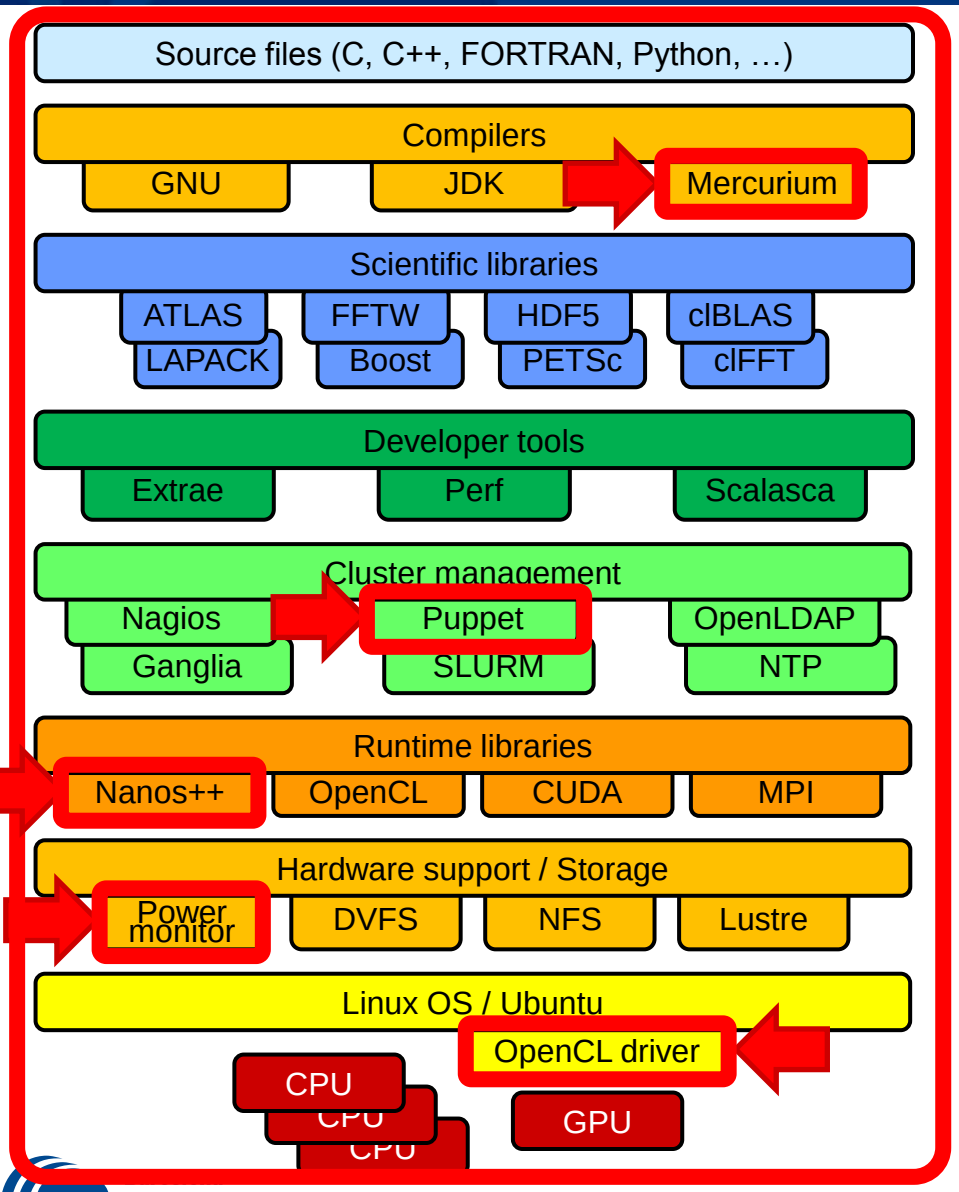
- Only dual core
  - Limiting factors for app that
-
- No ECC protection in memory
  - E
- No DMA support
- No standard server I/O interfaces
  - No native Ethernet or PCI Express
- No network protocol
  - TCP/IP, OpenMX, USB
- Thermal package not designed for sustained full-power operation

Implementation decisions, not unsolvable problems

The only need is a business case to justify the cost of including the new features (e.g. the HPC and server markets).

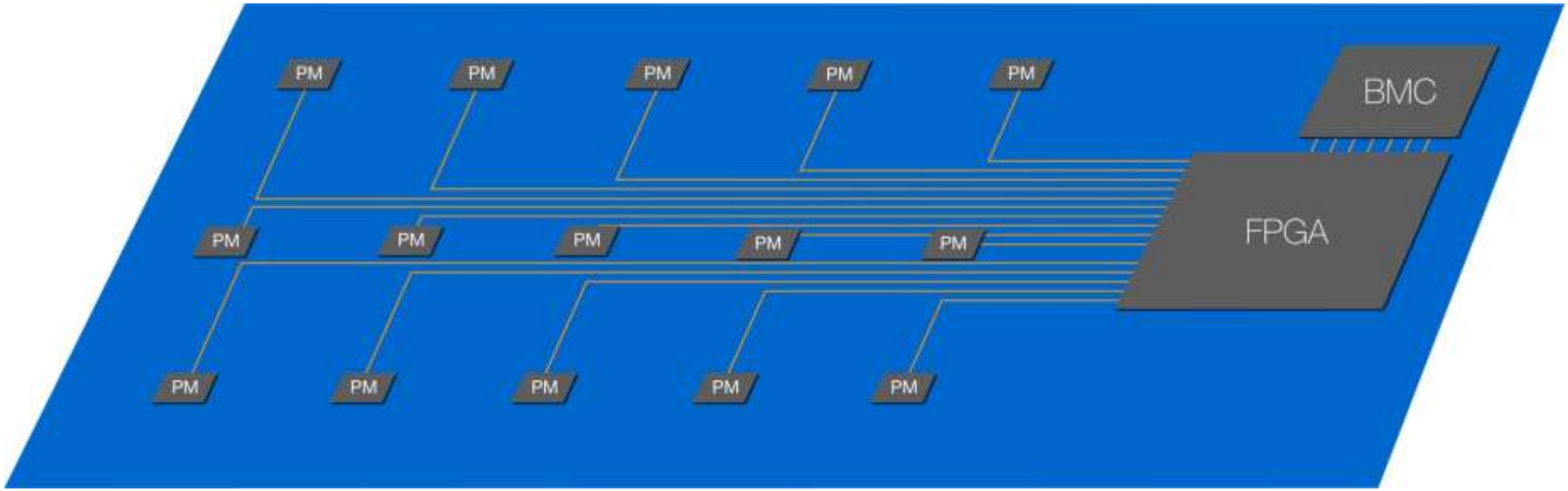


Full system software stack for ARM



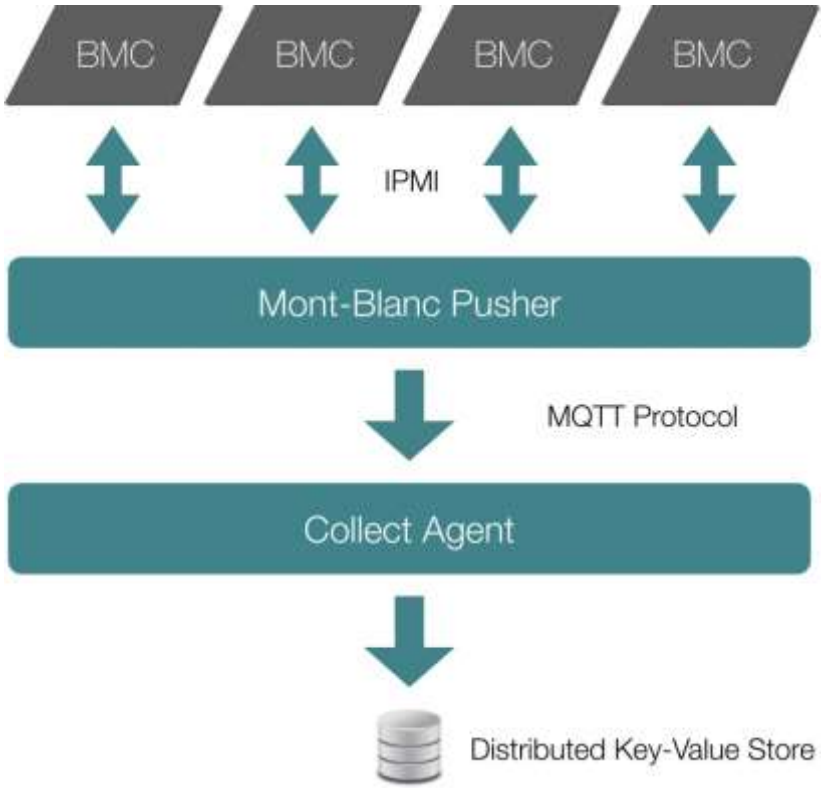
1. Fine grained power monitoring tool
2. OmpSs programming model ported to ARM + OpenCL support + FORTRAN support
3. Automatically deployed through Puppet and distributed through github
4. Tested on commercial ARM-based platforms

Power monitor – HW infrastructure

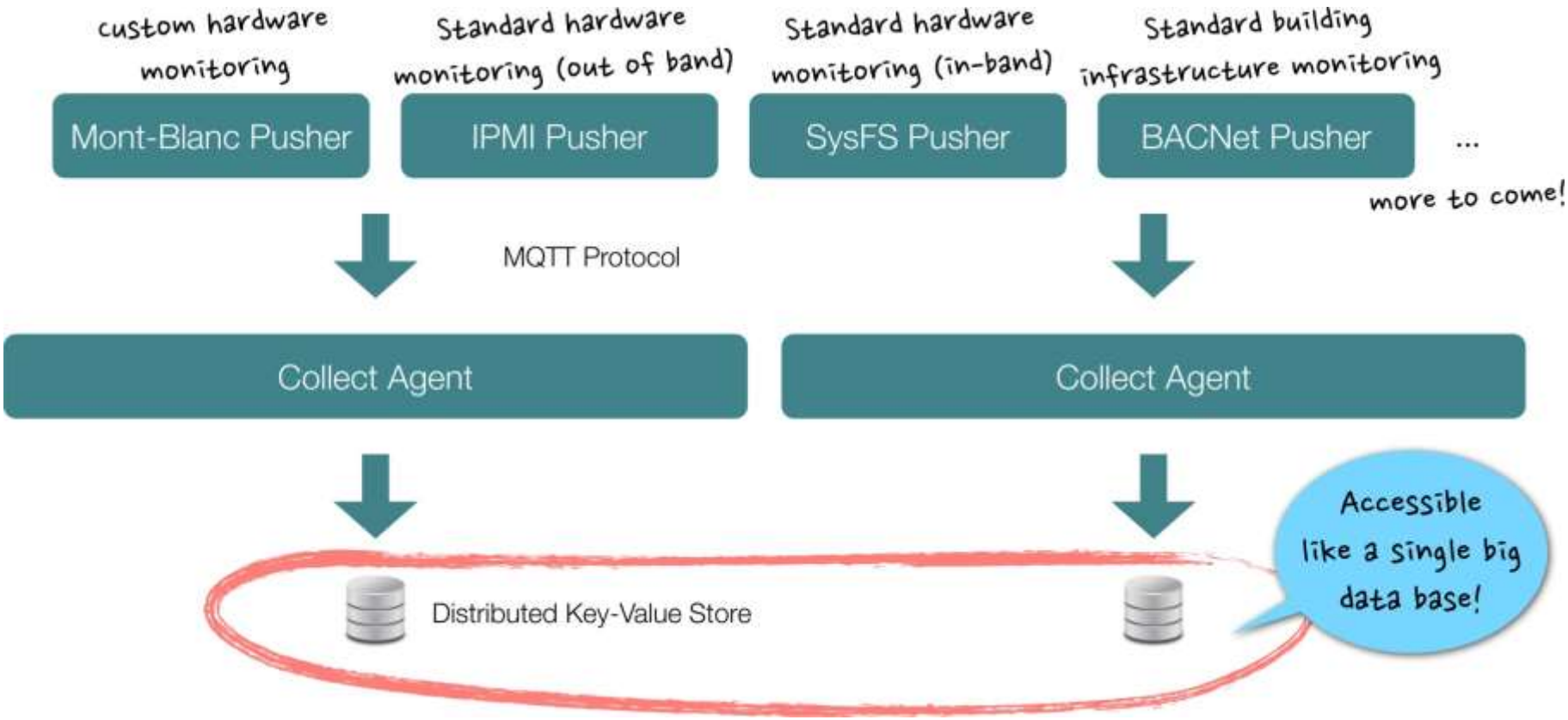


Power monitor – HW / SW interface

- Field Programmable Gate Array (FPGA)
  - Collects power consumption data from all 15 power measurement / sample interval: 70ms
- Board Management Controller (BMC)
  - Collects 1s averaged data from FPGA
  - Stores measurement samples in FIFO
- Mont-Blanc Pusher
  - Collects measurement data from multiple BMCs using custom IPMI commands
  - Forwards data using MQTT protocol through Collect Agent into key-value store

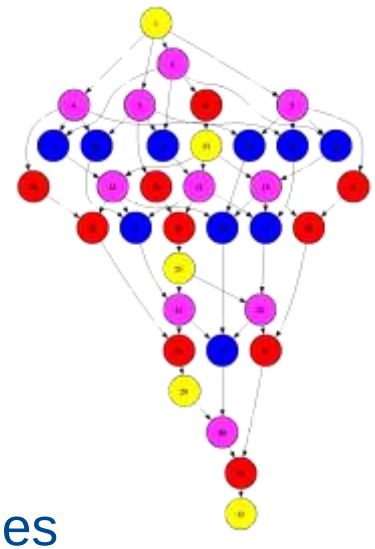


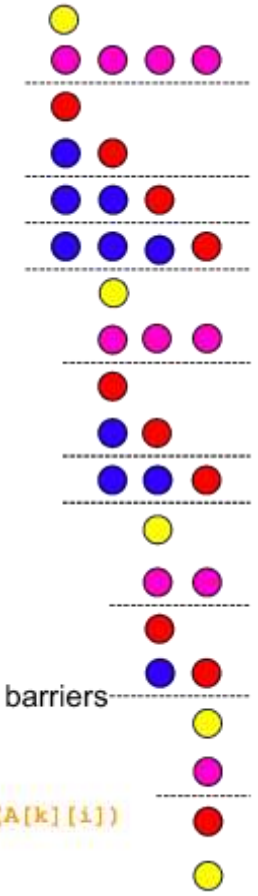
Power monitor – Block diagram



OmpSs programming model

- Programmer exposed to a simple architecture
  - Tasks
  - Data dependencies
  - Target devices (heterogeneity)
- Task graph provides look ahead
  - Exploit knowledge about the future
  - Allows exploration of scheduling policies
- It helps handling limitations of the hardware
  - Heterogeneity
  - Multiple address spaces
  - Low interconnect bandwidth
  - Synchronization





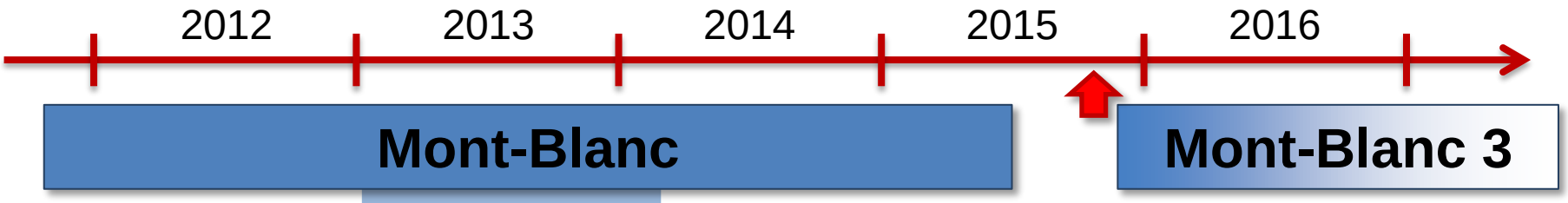

```
void Cholesky( float *A[NT][NT] ) {
int i, j, k;
for (k=0; k<NT; k++) {
   #pragma omp task inout (A[k][k])
   spotrf (A[k][k]) ;
   for (i=k+1; i<NT; i++) {
      #pragma omp task in (A[k][k]) inout (A[k][i])
      strsm (A[k][k], A[k][i]);
   }
   for (i=k+1; i<NT; i++) {
      for (j=k+1; j<i; j++) {
         #pragma omp task in (A[k][i], A[k][j]) inout (A[j][i])
         sgemm( A[k][i], A[k][j], A[j][i]);
      }
      #pragma omp task in (A[k][i]) inout (A[i][i])
      ssyrk (A[k][i], A[i][i]);
   }
}
```

Outline

- Brief introduction to supercomputing
  - For what do we use supercomputers?
  - Difference between a PC and a supercomputer
  - How to “describe” a supercomputer: metrics
- Analyzing history of HPC with different metrics
  - From vector CPUs to commodity components
- What is commodity nowadays?
  - Overview of current trends for mobile CPUs
- Prototypes based on mobile embedded technology @ BSC
  - Mini-clusters
  - The Mont-Blanc prototype
- Looking ahead – Mont-Blanc project

Is Mont-Blanc just a prototype?

No, Mont-Blanc is a EU project that leverages the fast growing market of mobile technology for scientific computation, HPC and non-HPC workload.



Mont-Blanc 2

- HPC prototype based on current mobile embedded technology
- Port and test real scientific applications
- Learn from the experience, plan for future architecture

Extend

- Support to hardware and system software
  - OmpSs programming model
  - Productivity tools
- New scientific and industrial applications
- Next generation Mont-Blanc architecture

Explore

- ARM 64-bit
- Fault tolerance and resiliency
- Market of ARM-based platforms for mini-clusters

Mont-Blanc project

montblanc-project.eu

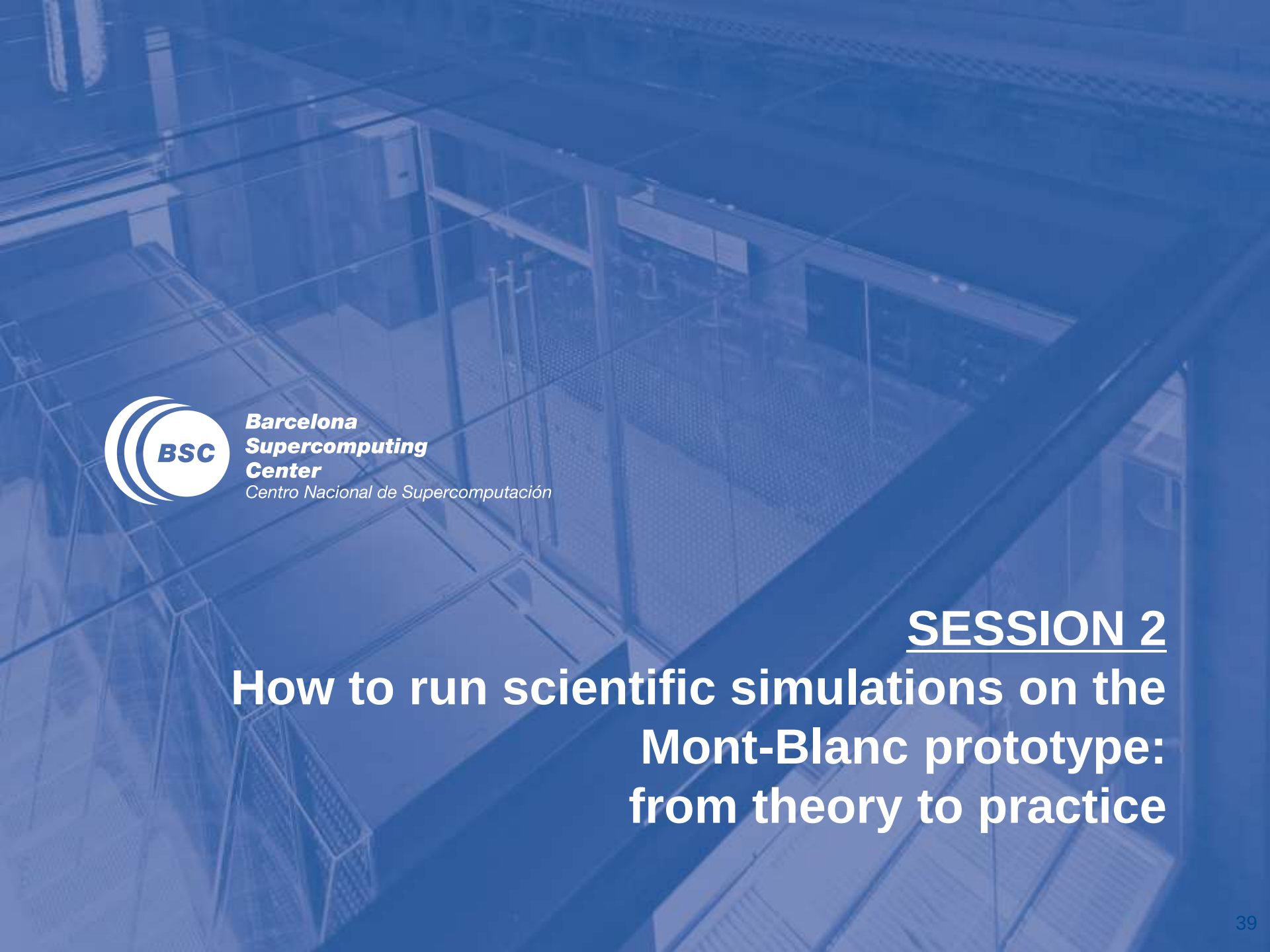
MontBlancEU

@MontBlanc_EU

filippo.mantovani@bsc.es

"The secret is to win going as slowly as possible."

Niki Lauda



SESSION 2

How to run scientific simulations on the Mont-Blanc prototype: from theory to practice

Session 2 - Outline

- HW resources
  - Cores
  - GPUs
  - Power monitoring
- SW resources
  - Compiler
  - Modules
  - Job scheduler
  - Power monitoring
- Exercises

The Mont-Blanc prototype

Exynos 5 compute card
2 x Cortex-A15 @ 1.7GHz
1 x Mali T604 GPU
6.8 + 25.5 GFLOPS
15 Watts
2.1 GFLOPS/W

Carrier blade
15 x Compute cards
485 GFLOPS
1 GbE to 10 GbE
300 Watts
1.6 GFLOPS/W

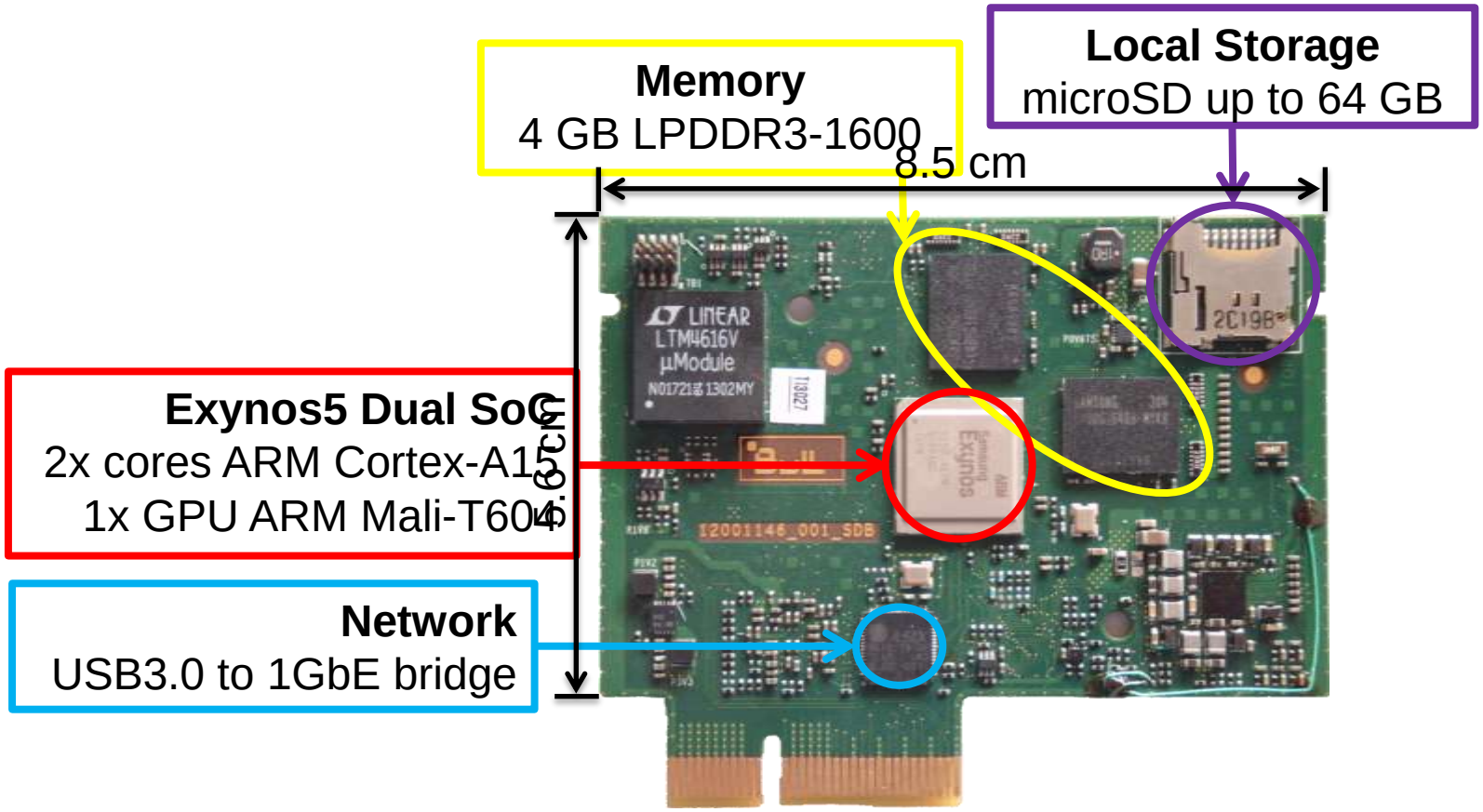
Blade chassis 7U
9 x Carrier blade
135 x Compute cards
4.3 TFLOPS
2.7 kWatts
1.6 GFLOPS/W

Rack
8 BullX chassis
72 Compute blades
1080 Compute cards
2160 CPUs
1080 GPUs
4.3 TB of DRAM
17.2 TB of Flash

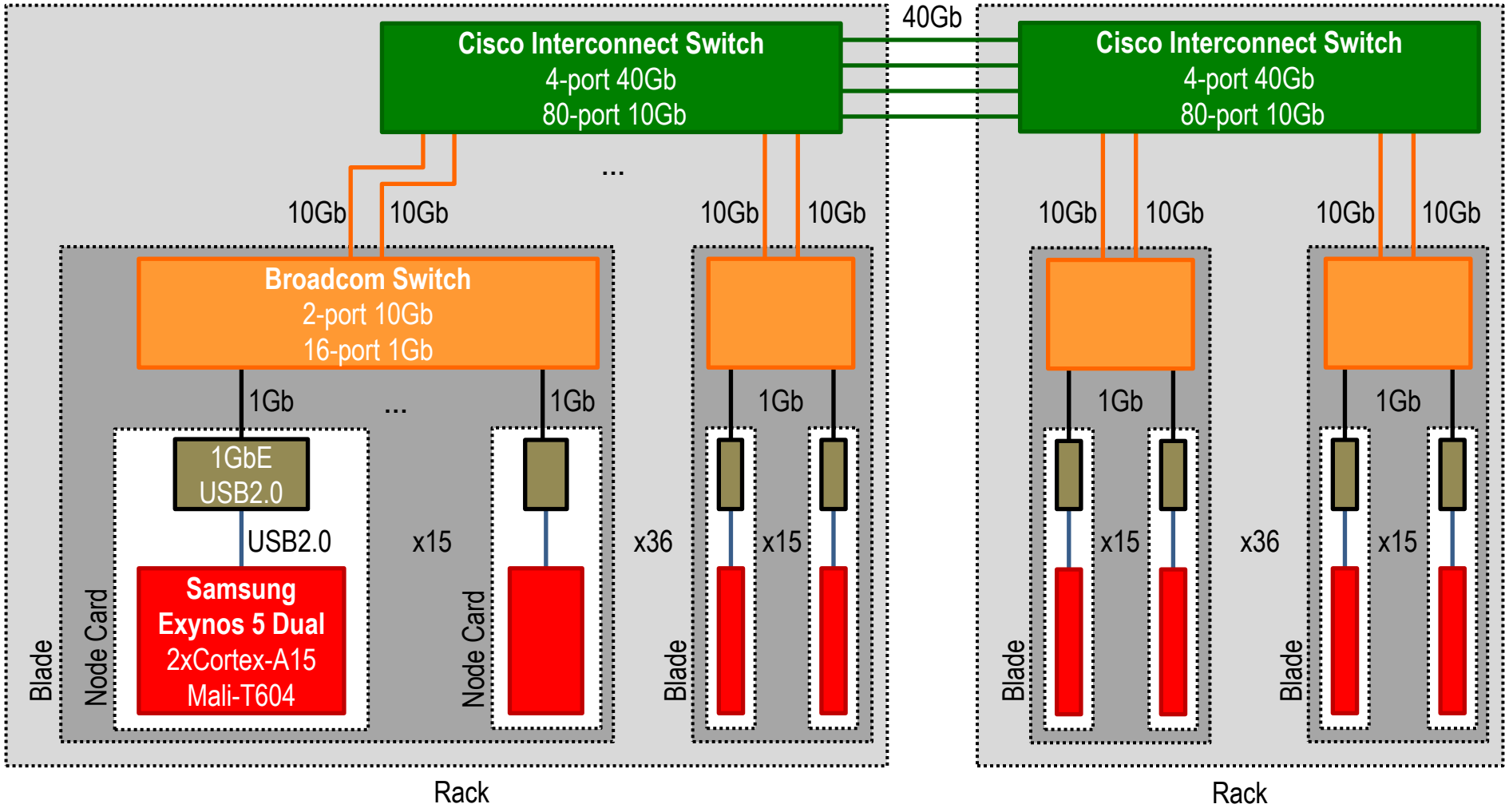
Mont-Blanc Server-on-Module (SoM)

CPU + GPU + Memory + Local Storage + Network

Form factor: 8.5 x 5.6 cm

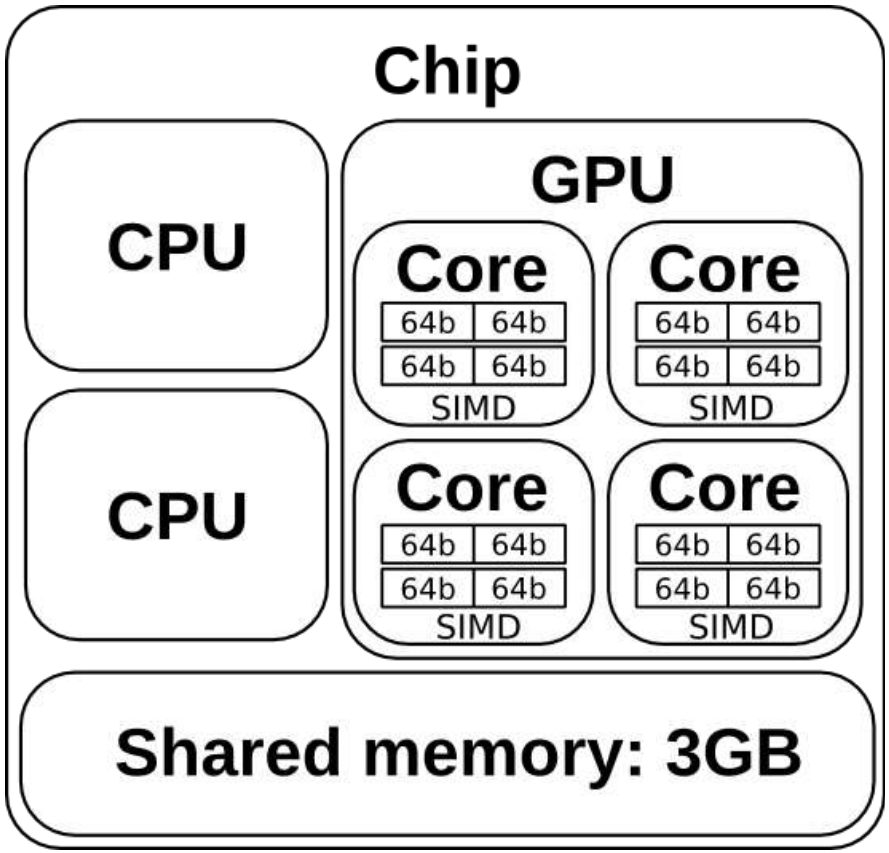


Interconnection network



Exynos 5 Dual: ARM + embedded GPU platform

- Dual-core ARM Cortex-A15 @ 1.7 GHz
  - VFP for 64-bit Floating Point
    - 6.8 GFLOPS (1 FMA / cycle)
  - NEON for 32-bit floating point SIMD
- Quad-core ARM Mali T604
  - Compute capable
    - **OpenCL 1.1**
    - 68 GFLOPS (SP)
    - 25.5 GFLOPS (DP)
- Shared memory between CPU and GPU
  - Possibility of avoiding host-device copies

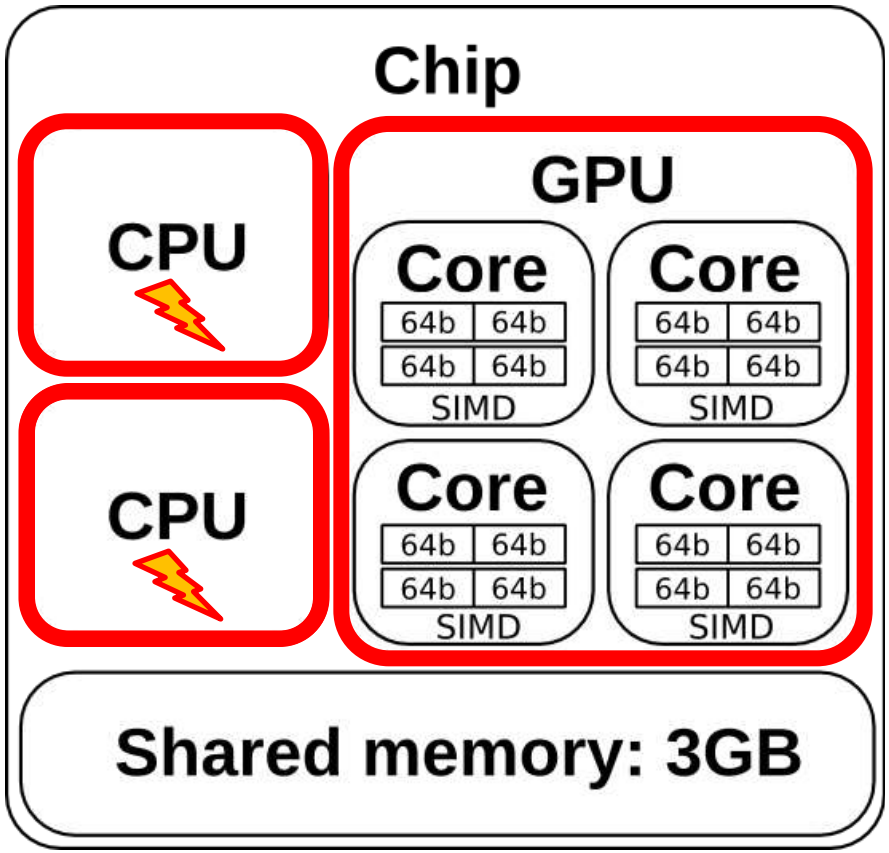


Taking advantage of the resources

By the end of the day we want to be able to run a small program on each of the following configurations:

- 1 CPU core (serial code)
- 2 CPU cores (OpenMP)
- 1 CPU core + GPU (OpenCL)

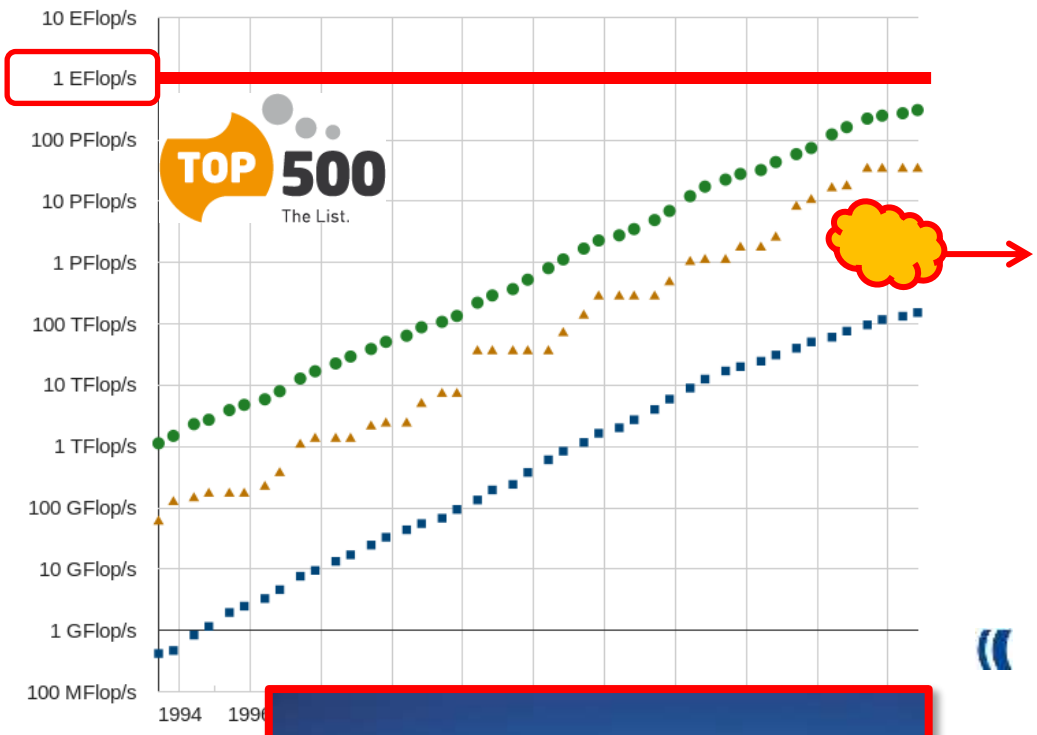
For each of these configurations we will change core frequency



We will study:

- Time to solution
- Energy to solution
- Power profile

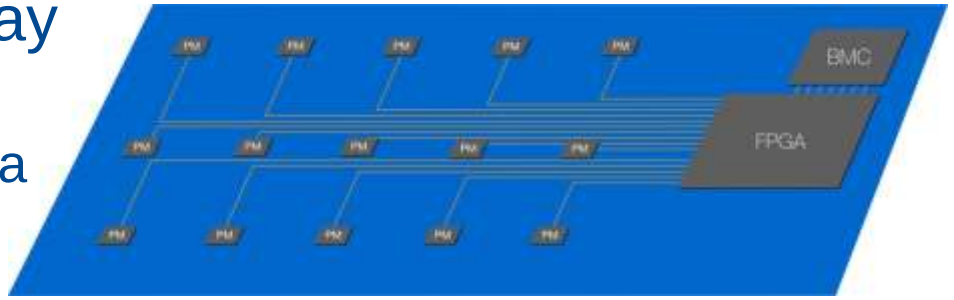
How important is power?



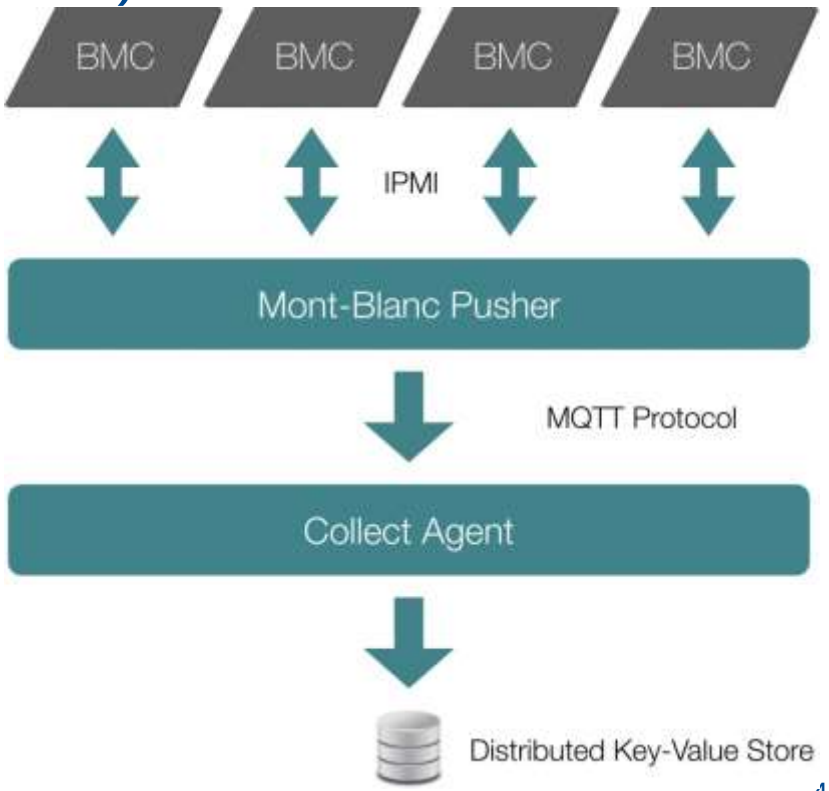
- Assuming (non-realistic) linear scale we need a system ~1000x larger to reach **one EFLOP**.
- This means ~1 GWatts for operating a supercomputer
- **NOTE:** Vandellòs Nuclear Power Plant in Catalunya: (144 km) is generating ~1 GWatts

Power monitor – HW / SW interface

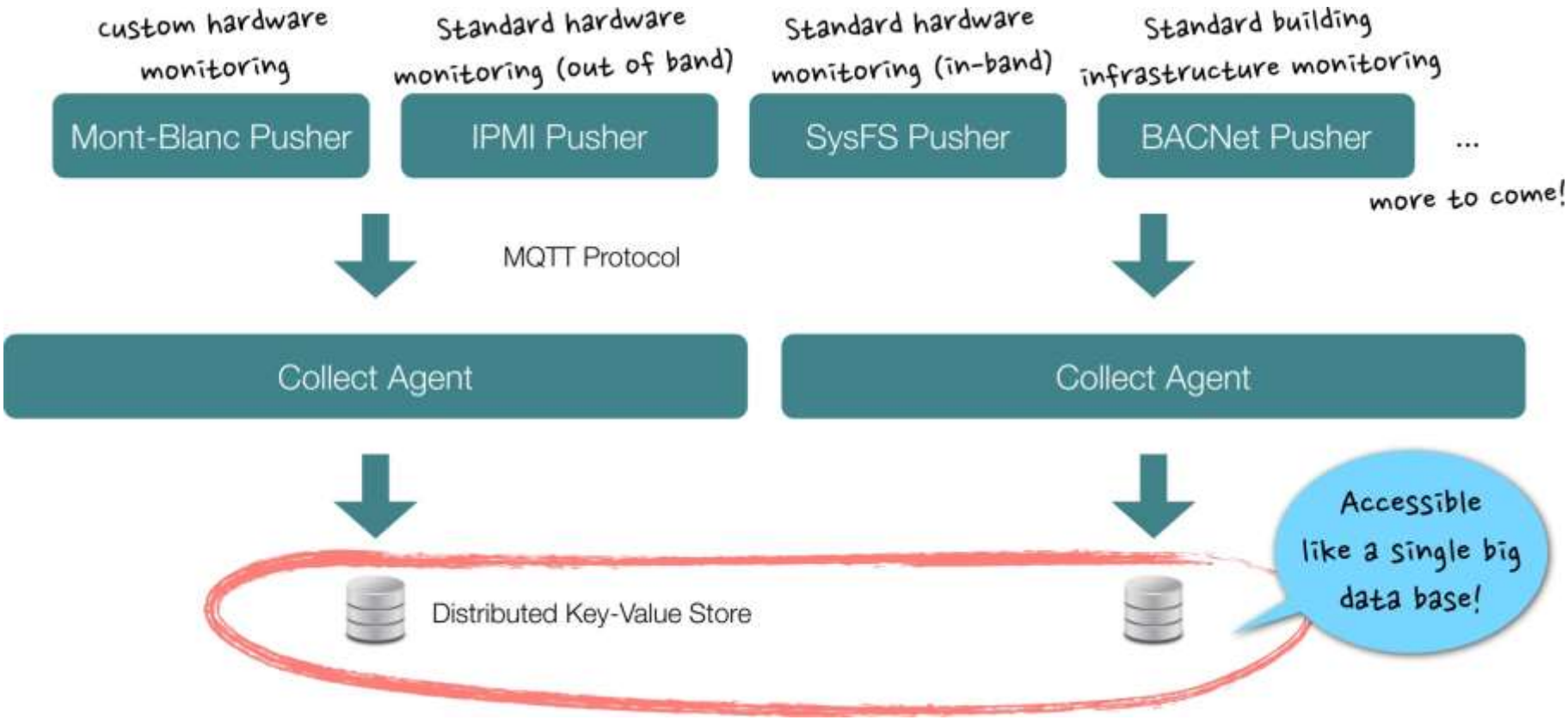
- Field Programmable Gate Array (FPGA)
  - Collects power consumption data from all 15 power measurement sample interval: 70ms



- Board Management Controller (BMC)
  - Collects 1s averaged data from FPGA
  - Stores measurement samples in FIFO
- Mont-Blanc Pusher
  - Collects measurement data from multiple BMCs using custom IPMI commands
  - Forwards data using MQTT protocol through Collect Agent into key-value store



Power monitor – Block diagram

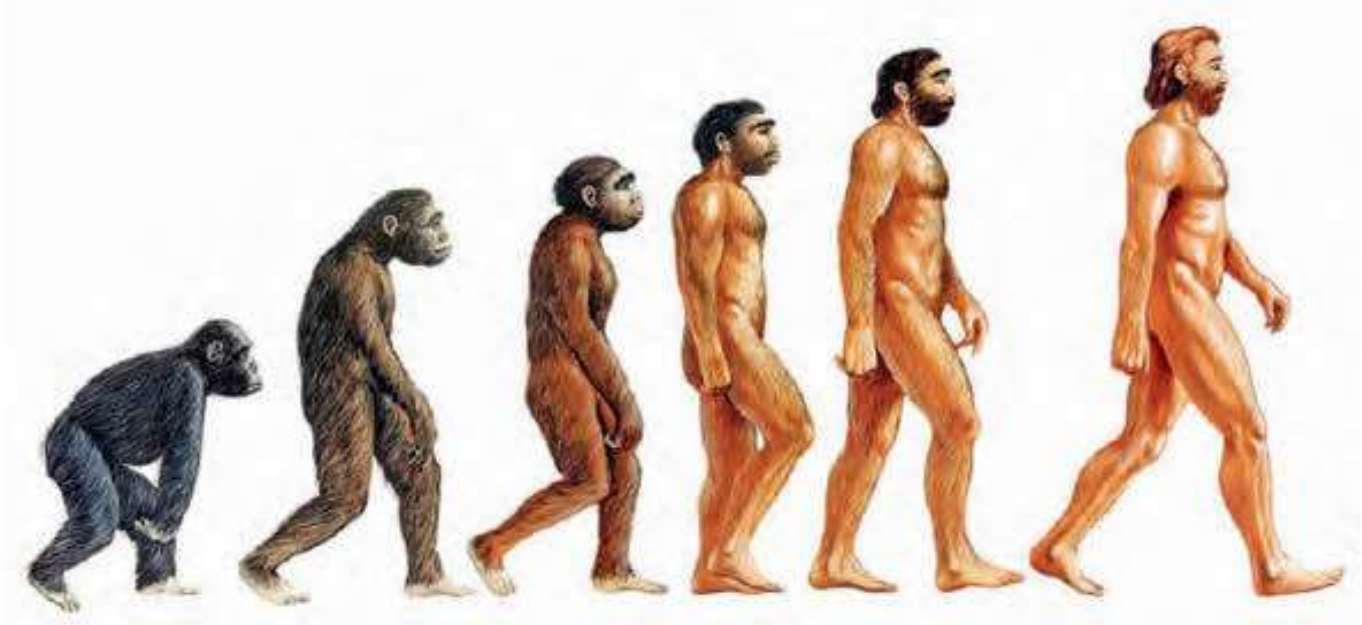


Session 2 - Outline

- HW resources
  - Cores
  - GPUs
  - Power monitoring
- SW resources
  - Compiler
  - Modules
  - Job scheduler
  - Power monitoring
- Runtime libraries… by examples
  - Serial code
  - OpenMP
  - OpenCL

Compilers

- Our ARM systems uses GNU compiler suite
  - gcc
  - gfortran
  - g++



Compilers: optimization flags

- **-march=arm*** - tells the compiler what kind of instructions can emit when generating assembly code
  - Binary portability across different ARM platforms
  - **-march=armv7-a**
  - **-march=native**

- **-mtune=name** - target specific ARM processors
  - Tune the code for a specific architecture
  - Often used together with -mcpu
  - **-mtune=cortex-a15**

Compilers: Floating point - ABI*

- -mfloat-abi={soft,softfp,hard}
  - soft - library calls for floating point emulation
    - Old ARM based SoCs did not include floating-point unit
  - softfp - allows the generation of code using the hardware floating-point instructions, but still uses soft-float calling convention
    - Binaries will benefit from dedicated hardware
  - hard - allows generation of floating-point instructions and uses FPU-specific calling convention
    - Noticeable improvement in floating-point performance compared to soft
    - Cortex-A15 (hands-on) uses hard

**** ABI: application binary interface***

Environment modules

- Package which provides dynamic modification of a user's environment via module files
  - Each module configures the shell for an application
  - Support for different versions of a single application
- How to use it?
  - module avail
  - module load ${module_name}/${version}
    e.g. module load power_monitor
  - module unload ${module_name}/${version}
- How it works?
  - It modifies environment variables
    - PATH
    - LD_LIBRARY_PATH
    - LD_RUN_PATH

Job scheduler: SLURM

- SLURM is an open source job scheduler and resource manager
  - Designed to operate in heterogeneous clusters with up to 64k nodes and >100k processors
  - Developed by Lawrence Livermore national Laboratory
  - Since 2010, maintained by SchedMD LLC
- SLURM features
  - Several scheduler policies (FIFO, backfilling, GANG)
  - Support for external schedulers (LSF, MOAB/MAUI)
  - Uses priorities and limits (queues)
  - Support for Heterogeneous systems (GPU)
  - DB (MySQL) for accounting management
- Most important: **it allows you to run on a subset of nodes of a large clusters without interferences**

Running/Handling jobs with SLURM

- sbatch <myscript.job>
  - myscript.job is a shell script with directives (resources, application, etc)
- squeue - Job queue control

```
fmantovani@mb-login-1:~$ squeue
JOBID   PARTITION   NAME     USER         ST   TIME   NODES   NODELIST
46      mb          myjob    fmantovani   R    0:24   16      mb-[1-15,30]
47      mb          myjob    fmantovani   R    0:15   20      mb-[31-50]
```

- scancel <job_id> - Delete a job
- sinfo - Ciew information about SLURM nodes and partitions

```
fmantovani@mb-login-1:~$ sinfo
PARTITION   AVAIL TIMELIMIT   NODES   STATE   NODELIST
mb*         up    30:00       900     idle    mb-[1-900]
mb*         up    30:00       30      alloc   mb-[901-930]
```

- scontrol show job <job_id> - Shows information regarding the job identified by job_id

Your first SLURM job script

```
#!/bin/bash

#SBATCH --partition=mb
#SBATCH --job-name=ictp-test
#SBATCH --output=ictp_%j.out
#SBATCH --error=ictp_%j.err
#SBATCH --ntasks=1
#SBATCH --nodes=1
#SBATCH --time=00:05:00
#SBATCH --gres=gpu
```

This is important… It requires an estimation of duration of the job execution: HH:MM:SS

If you want to run on the GPU you need this

srun **--cpu-freq=<FREQ-IN-KHz>** <app-to-run>

Power measurement acquisition

- Data needed:
  - **WHEN**: Interval of time while the job was running $T_START, $T_END
  - **WHERE**: List of nodes where the job was running $NODE

- How to get them
  - AT RUN TIME:
    Inside the job script: date +"%s" and echo $SLURM_JOB_NODELIST
  - POST-EXECUTION with SLURM command:
    export SLURM_TIME_FORMAT="%s"
    sacct -u $USER -o jobid,nodelist,start,end consumedenergy -j $JOB

- Query the power monitoring database
  - dcdbquery -h mb.mont.blanc -r ${NODE}-PWR $T_START $T_END

- Output is a csv file with the following format
  - "ID,Timestamps,PowerData[mW]"
  - You can import it in Excel, plot it with Gnuplot, etc.

Exercises

1. Submit your first job
   - Run ‘hostname’ on 1, 2 and 4 Mont-Blanc nodes
2. Test compiler flags
   - Get the code in $HOME/workshop-day6/es1.c and compile it with:
     - gcc es1.c -o es1-1 -lm
     - gcc -O3 es1.c -o es1-2 -lm
     - gcc -O3 -march=native es1.c -o es1-3 -lm
   - Write a job script that execute es1-1, es1-2 and es1-3
   - Execute the job script and analyze the output
3. Consider one of your previous jobs
   - Run it at 0.8GHz and 1.6GHz
   - Get energy to solution of it
   - Get power data of it

SESSION 3
How to use all resources

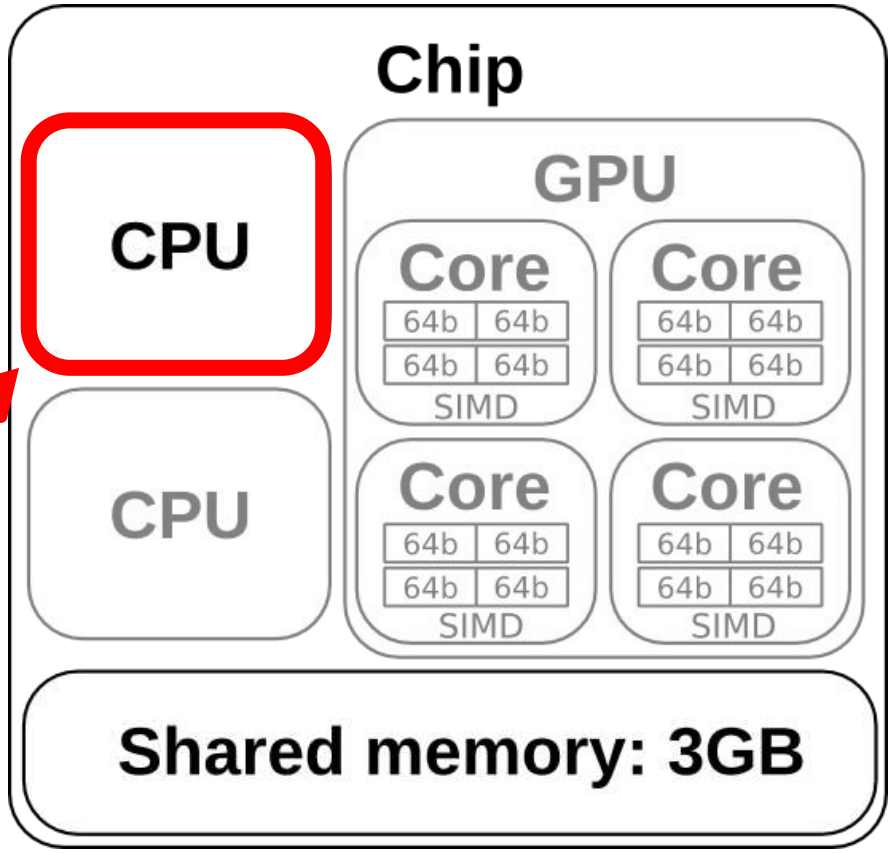
Session 3 - Outline

- Runtime libraries… by examples
  - Serial code
  - OpenMP
  - OpenCL

Synthetic workload

```
for (i = 0; i < arraySize; i++) {
    a2 = cbrt( inputA[i]*inputA[i] / M_PI ) ;
    b2 = cbrt( inputB[i]*inputB[i] * M_PI_2 ) ;
    output[i] = log(sqrt(a2+b2));
}
```

- You already used it (without knowing it) in es0…
- Classical C implementation produces a serial code



OpenMP

De facto standard API for writing shared memory parallel applications in C, C++, and Fortran

OpenMP API consists of:

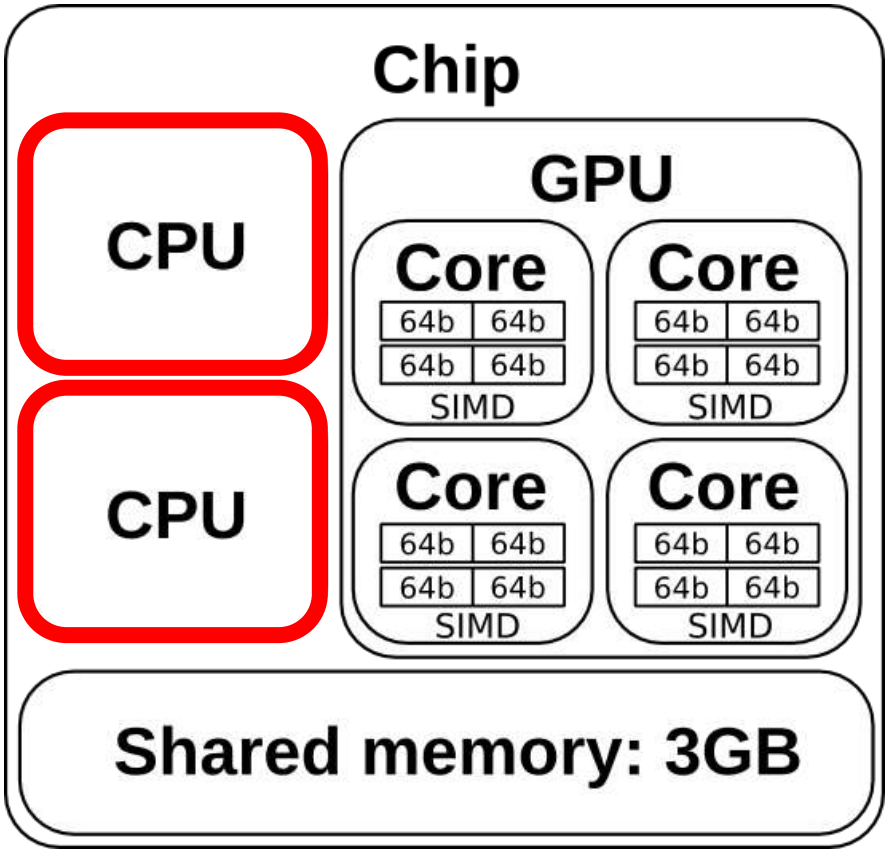
- Compiler Directives
- Runtime subroutines/functions
- Environment variables

gcc … -fopenmp …

```
#include <omp.h>
#pragma omp parallel for private (a2,b2)
  for (i = 0; i < arraySize; i++) {
     a2 = cbrt( inputA[i]*inputA[i] / M_PI ) ;
     b2 = cbrt( inputB[i]*inputB[i] * M_PI_2 ) ;
     output[i] = log(sqrt(a2+b2));
  }
```

```
export OMP_NUM_THREADS=2
```

with OpenMP



CPUs are fine, but how can we take advantage of the GPU?

OpenCL

- Framework for writing programs across heterogeneous platforms containing processing elements such as:
  - CPUs
  - GPUs
  - FPGAs
- Open standard
- Flexible, can be used from many languages such as:
  - C++
  - Java
  - Python
  - Perl
- Easy to try anywhere, even in a laptop without GPU

Memory types

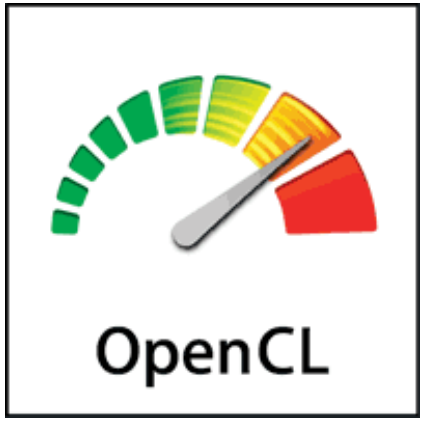
- Global memory: __global
  To refer to memory objects allocated from the global memory pool.
- Local memory: __local
  To describe variables that need to be allocated in local memory and are shared by all work-items of a work-group.
- Constant memory: __constant
  To describe variables allocated in global memory and which are accessed inside a kernel(s) as read-only variables. These can be accessed by all work-items of the kernel during its execution.
- Private memory: __private
  All variables inside a function or passed into the function as arguments are in the __private or private address space. Variables declared as pointers are considered to point to the __private address space if an address space qualifier is not specified.

Basic data structures

Variable	Type
Platform id	cl_platform_id
Device id	cl_device_id
Context	cl_context
Command Queue	cl_command_queue
Program	cl_program
Kernels	cl_kernel

Steps for porting to OpenCL

- Initialization
- Finalization
- Create kernel in OpenCL
- Create wrapper function
- Replace function call



Initialization (1)

- Get the platform IDs and select one
- Connect to a compute device of selected platform
- Create a compute context for the selected device
- Create a command queue for those context and device
- Open kernels file, read it and format it for OpenCL
- Create the compute program
- Build the program executable: at runtime!
  - Add OpenCL flags
  - Get information about kernel's compilation errors
- Create kernel

Initialization (2)

- clGetPlatformIDs()
- clGetDeviceIDs()
- clCreateContext()
- clCreateCommandQueue()
- (Open kernels file, read it and format it for OpenCL)
- clCreateProgramWithSource()
- clBuildProgram()
  - Add OpenCL flags
  - Get information about kernel's compilation errors!
- clCreateKernel()

Finalization

- Release kernel objects
  clReleaseKernel()
  - All of them!
- Release program
  clReleaseProgram()
- Release command queue
  clReleaseCommandQueue()
- Release context
  clReleaseContext()

Create kernel in OpenCL (1)

- Should be something with this shape:

```
__kernel nameOfKernel (__global const int
*inputVector, __global int *outputVector)
{
    const int i = get_global_id(0);
    outputVector[i] = inputVector[i] * 2;
}
```

- Like a look

```
for (i = 0; i < SIZE; i++)
    outputVector[i] = inputVector[i] * 2;
```

- Store it in a ".cl" file

Create kernel in OpenCL (2)

- Should be something with this shape:

```
__kernel nameOfKernel (__global const int *inputVector, __global int *outputVector)
{
        const int i = get_global_id(0);
        outputVector[i] = inputVector[i] * 2;
}
```

- If passing pointers as arguments, declare them with the `__global` prefix
- To allow double precision set this pragma at the start of the OpenCL file:

```
#pragma OPENCL EXTENSION cl_khr_fp64 : enable
```

Create wrapper function (1)

- Add this in your program where the function that will be replaced by the new kernel is
- Wrapper including:
  1. Create memory objects
  2. Create buffers
  3. Prepare arguments
  4. Set number of threads to partition the data
  5. Call the kernel
  6. Wait for the kernel execution to finish
  7. Get the results and mapping them to the CPU
  8. Release memory created by the buffers

Create wrapper function (2)

```
{
  int *variable = (int*)malloc(size);
  // Initialization
1 cl_mem memoryObject;
2 memoryObject = clCreateBuffer();
3 clSetKernelArg(..., memoryObject);
4 size_t globalWorkSize[1] = { size }; // #elems to process
4 size_t localWorkSize[1]  = { 1 };    // #elems per kernel
5 clEnqueueNDRangeKernel();
6 clFinish();
7 clEnqueueMapBuffer(); // Get output data (copy from GPU)
8 clReleaseMemObject();
  // Finalization
}
```

One example

/lustre/ICTP/session-3/es2.cpp

Exercises

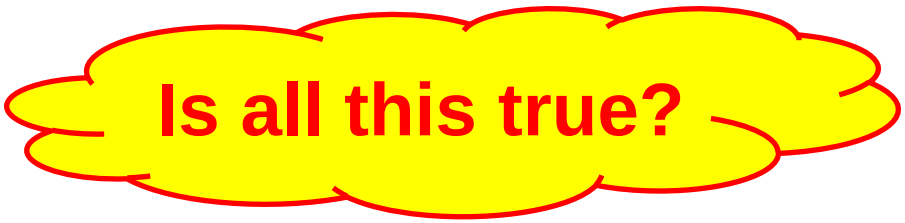
1. Transform es0.c into an OpenMP program (es1.c)
2. Run es1 with 1 and 2 threads @ 0.8 GHz and @ 1.6 GHz
3. Copy files in /lustre/ICTP/session-3/ to your home
   Recognize the part ported to OpenCL in es2.cpp
4. Compile es2.cpp
5. Run es2 @ 0.8 GHz and @ 1.6 GHz
6. Provide a ranking of the execution times of each of the 4 tests performed

SESSION 4

Becoming “energy aware programmers”

Requirements

- We have a workload implemented as
  - serial
  - parallel (OpenMP)
  - heterogeneous (OpenCL)
- We know how to run each of these on the Mont-Blanc prototype
  - we can submit/cancel/check jobs
  - we can run at different frequencies
- We know how to retrieve power data out of the Mont-Blanc prototype



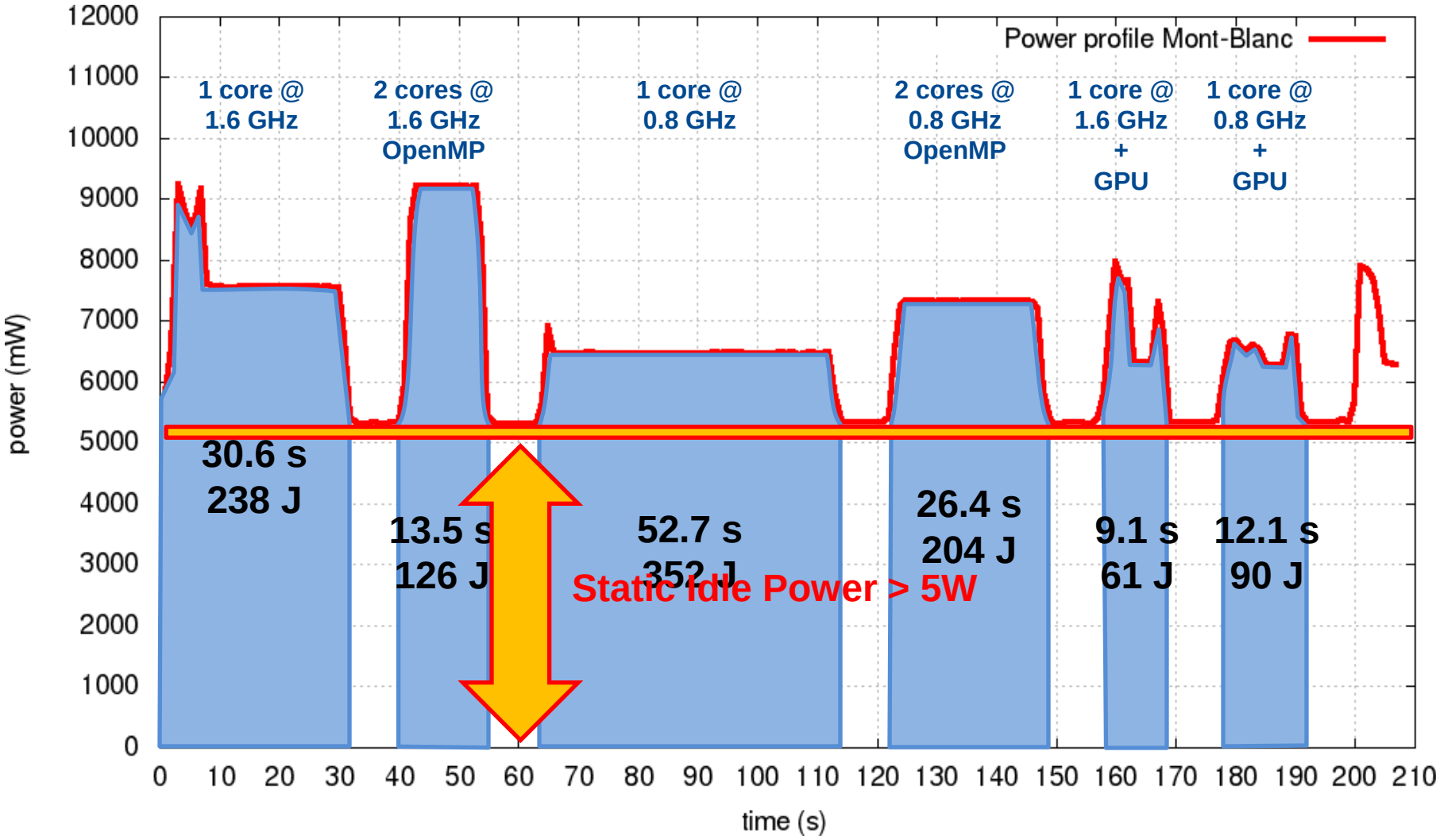
Exercises

1. Prepare a single job script that runs the following cases and fill the table:

1 core	2 cores	@1.6 GHz	@0.8 GHz	GPU	Time to solution	Energy to solution
X		X				
	X	X				
X			X			
	X		X			
X		X		X		
X			X	X		

2. Plot the power profile of the previous execution (csv)
   - Hint: insert a “srun sleep 10” between consecutive executions

Power profile study



Mont-Blanc project

montblanc-project.eu

MontBlancEU

@MontBlanc_EU

filippo.mantovani@bsc.es

"The secret is to win going as slowly as possible."
Niki Lauda