MONITORING, ANALYSIS AND TUNING OF PARALLEL APPLICATIONS

l'Autònoma

Contents

- Instrumentation tools
- Tracing and Profiling tools
- Visualization tools
- Analysis tools
- Tuning tools

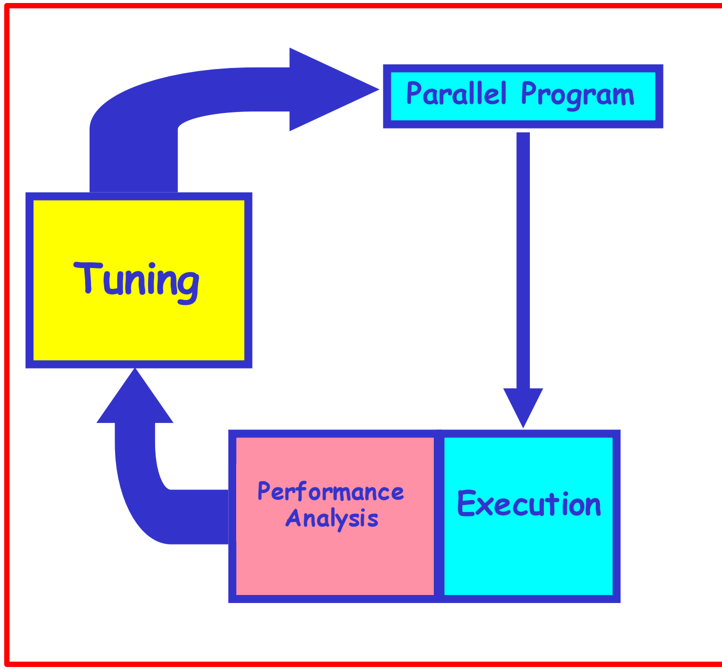
l'Autònoma

Performance Tuning. Why?

- **Reducing execution time**
- **Increasing "Throughput"**
  - Get more results in less time
- **Investment return**
  - Get more from the money invested in resources (HW & SW)

l'Autònoma

Performance Analysis and Tuning



Measurement

↓

Execution Data

↓

Analysing the Data

↓

Detecting Performance Problems

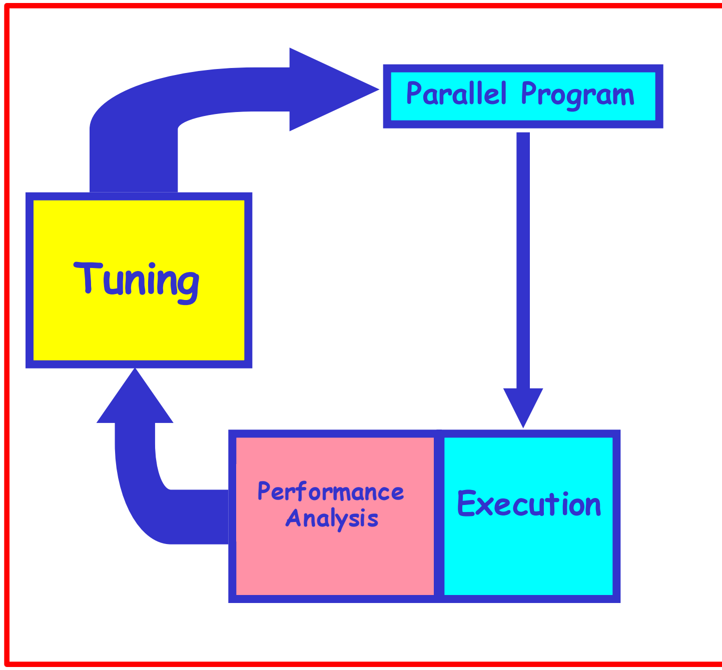
↓

Finding their sources

↓

Tuning

Performance Analysis and Tuning



Measurement

Execution Data

Analysing the Data

Detecting Performance Problems

Finding their sources

Tuning

Measurement Techniques

- **Everything based on capturing events:**
  - Each event is produced in a certain processor and at some specific time
  - Each event has a type:
    - Cache miss
    - Memory access
    - Start communication
    - ...

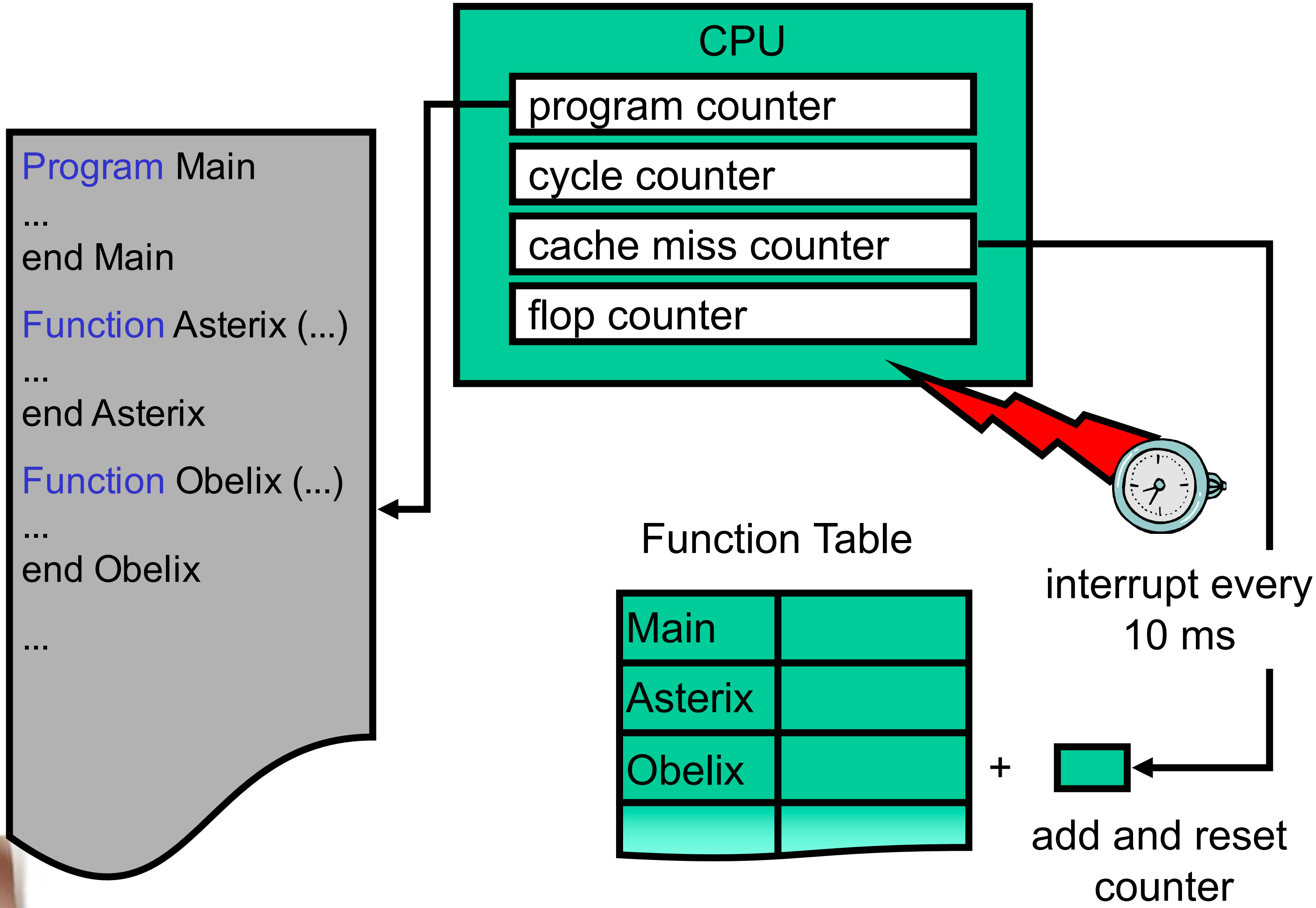
Measurement Techniques

- **"Profiling": We get aggregated information for different events.**
  - Sampling: Statistical Information
  - Instrumentation: Catching every occurrence

- **"Tracing": Saving every event.**
  - Instrumentation

l'Autònoma

Sampling


CPU
program counter
cycle counter
cache miss counter
flop counter
Program Main
...
end Main
Function Asterix (...)
...
end Asterix
Function Obelix (...)
...
end Obelix
...
Function Table
Main
Asterix
Obelix
+
interrupt every
10 ms
add and reset
counter

Performance API (PAPI)

- Innovative Computing Laboratory (U. Tennessee)
- PAPI specifies a standard interface for accessing current processors' performance counters
- These counters count hardware events (L1 or L2 cache misses, FLOPs, memory access, etc.)
- Correlating these counters can provide very useful performance information (also for compiler optimizations)

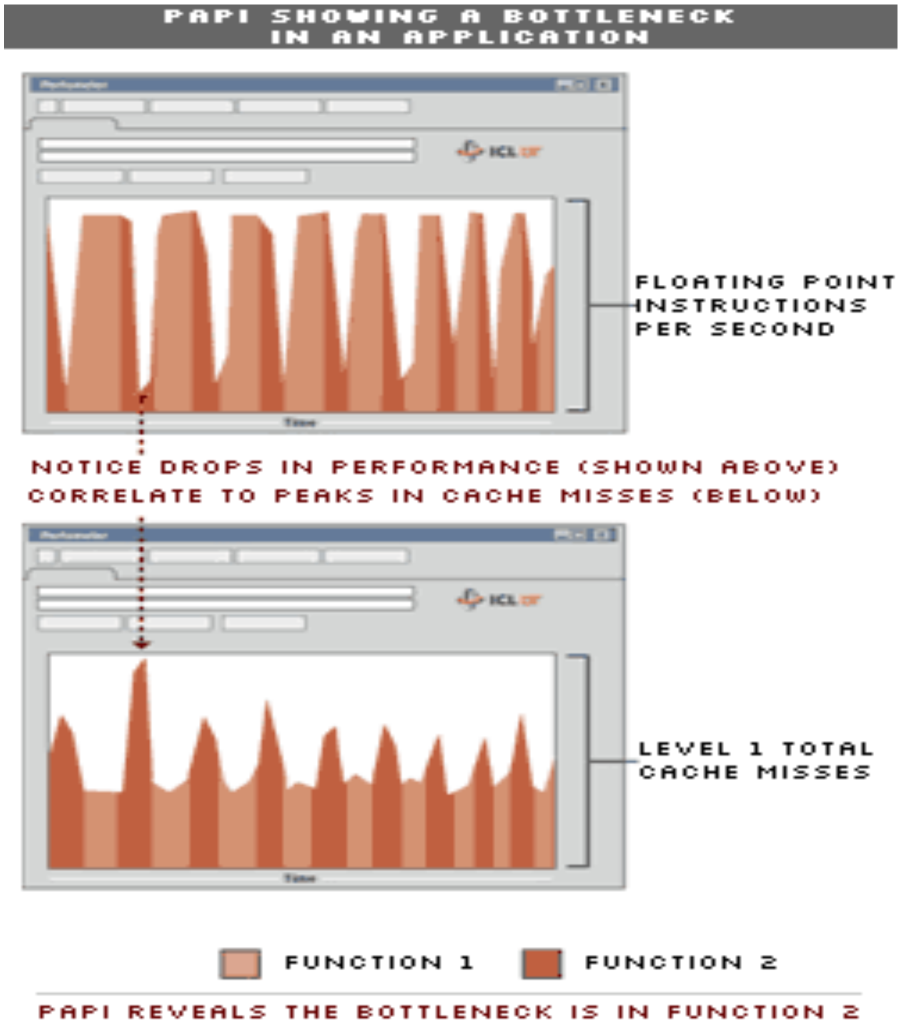
l'Autònoma

Performance API (PAPI)

- **High level functions:**
  - PAPI_num_counters() – number of available counters
  - PAPI_flips() - Mflips/s (floating point instruction rate)
  - PAPI_flops() - Mflops/s (floating point operation rate)
  - PAPI_ipc() – instructions per cycle
  - PAPI_accum_counters() – accumulate current values and initialize counters
  - PAPI_read_counters() – copy current values and initialize counters
  - PAPI_start_counters() – start counting
  - PAPI_stop_counters() – stop counting + return counters

l'Autònoma

Performance API (PAPI)



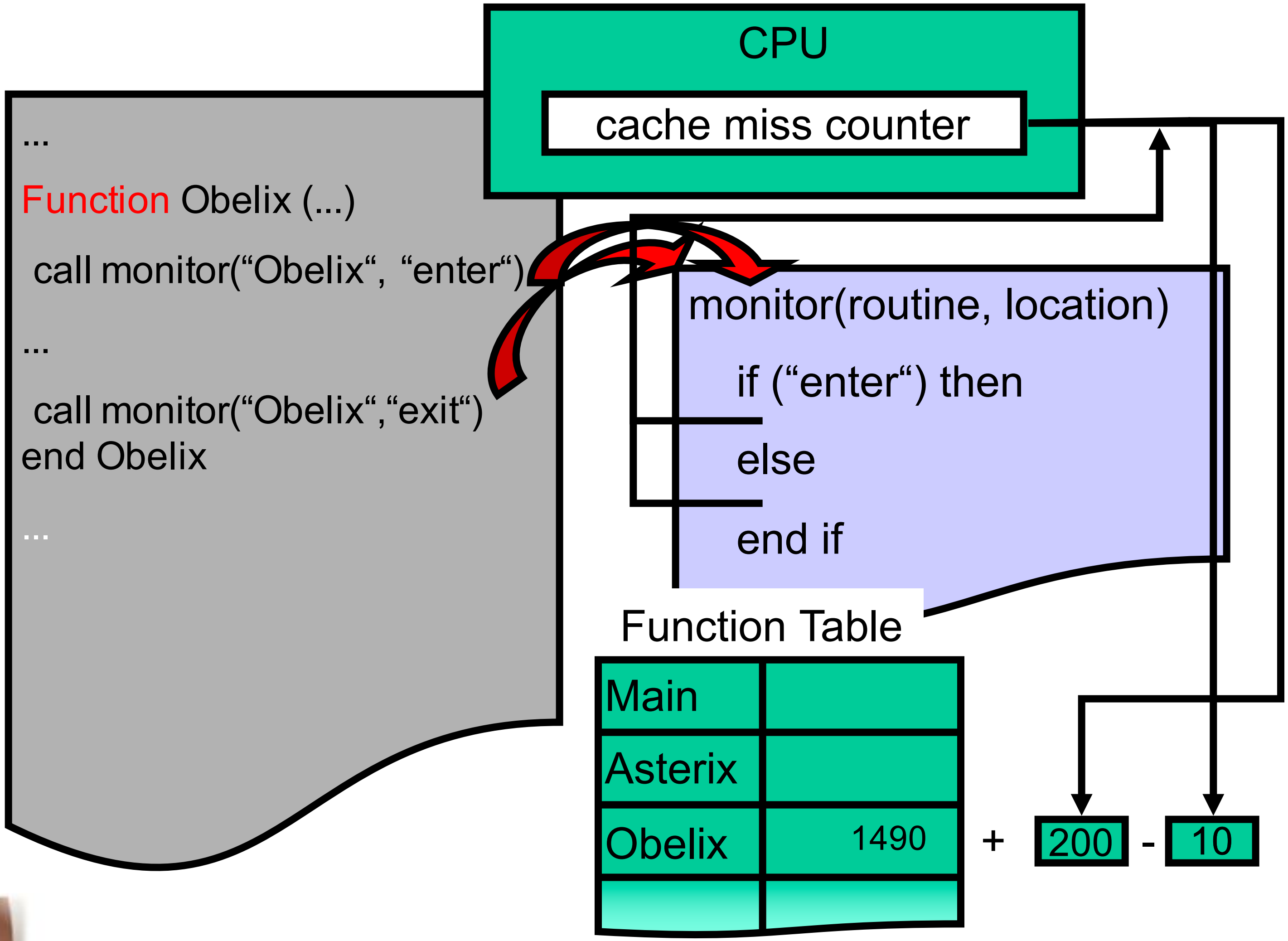
l'Autònoma

Performance API (PAPI)

- **Tools using PAPI:**
  - PerfSuite (NCSA-U. of Illinois)
  - HPCToolkit (CS Dept – Rice U. Texas)
  - Tuning and Analysis Utilities (TAU) (U. Oregon, Julich Research Center)
  - Kojak (U. of Tennessee and Julich Research Center)
  - Open|SpeedShop (Department of Energy, LLNL, LANL, Sandia)
  - OMPP (OpenMP Profiler) (U. California at Berkeley)
  - Periscope (TUM)

l'Autònoma

Instrumentation



Instrumentation

- **Instrumenting source code:**
  - Manually or introduced by the compiler
    - \+ Portability
    - \+ Easy to relate to the source code
    - \- Recompilation is necessary for changing instrumentation
    - \- Libraries and external modules cannot be instrumented

```
...
Function Obelix (...)
 call monitor(“Obelix“, “enter“)
...
 call monitor(“Obelix“,“exit“)
end Obelix
...
```

l'Autònoma

Instrumentation

- **Instrumenting object code:**
  - Using hooks
    - + No recompilation needed
    - + It is possible to instrument applications without their source code
    - - Portability
    - - It is more difficult to relate events with source code

l'Autònoma

DynInst

- DynInst (Dynamic Instrumentation) library, developed by the University of Wisconsin and the University of Maryland, for dynamically inserting code into an application

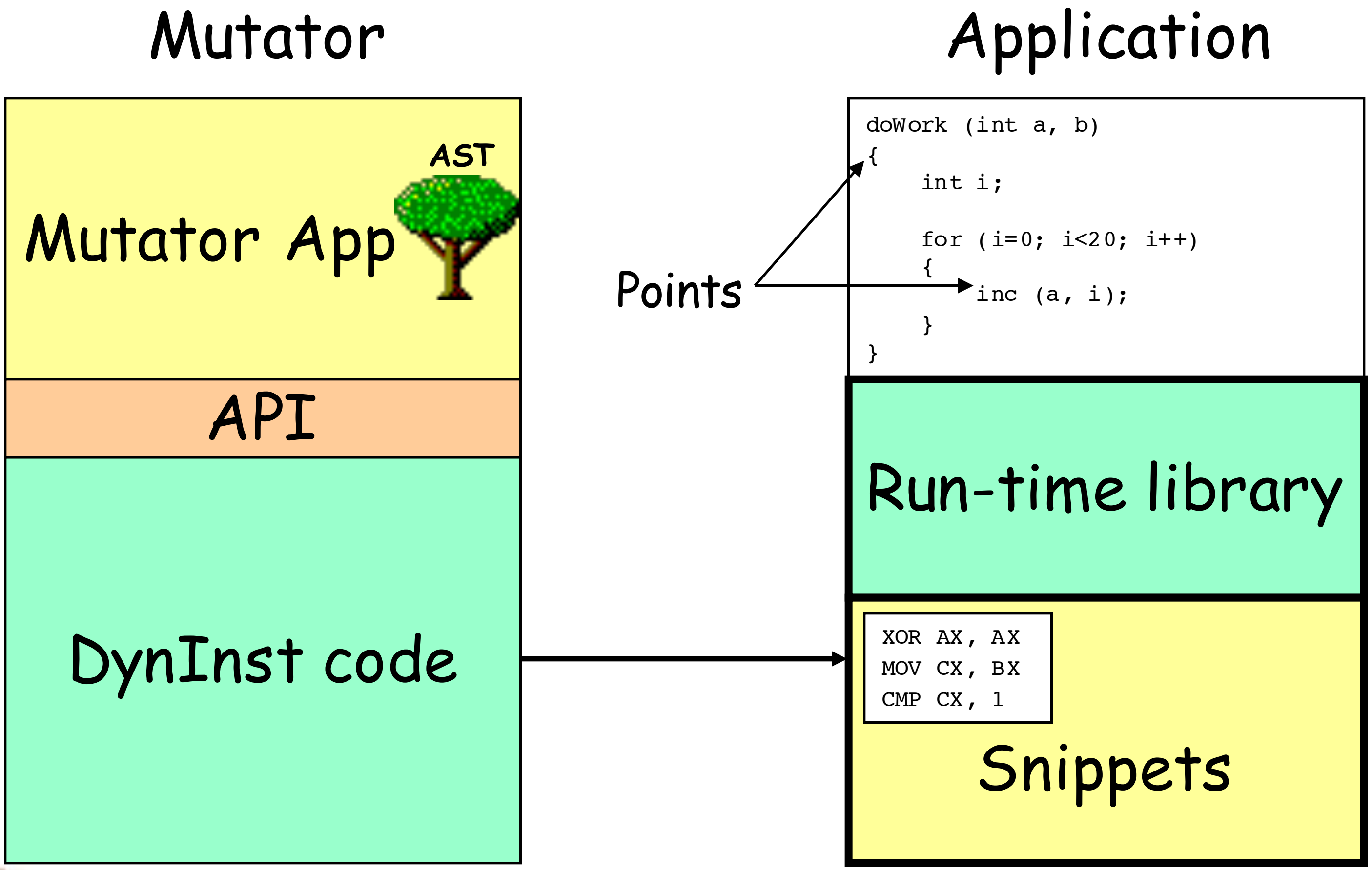
- DynInst defines an API for inserting code at runtime

- The modified application is not recompiled, relinked or re-executed

DynInst

- **Dyninst allows for:**
  - Creating a new process or to attach one process (mutator) to another (mutatee)
  - Creating a new fragment of code
  - Inserting a piece of code into a running process

DynInst


Mutator
Application
AST
Mutator App
API
DynInst code
Points
doWork (int a, b)
{
int i;
for (i=0; i<20; i++)
{
inc (a, i);
}
}
Run-time library
XOR AX, AX
MOV CX, BX
CMP CX, 1
Snippets

Profiling vs Tracing

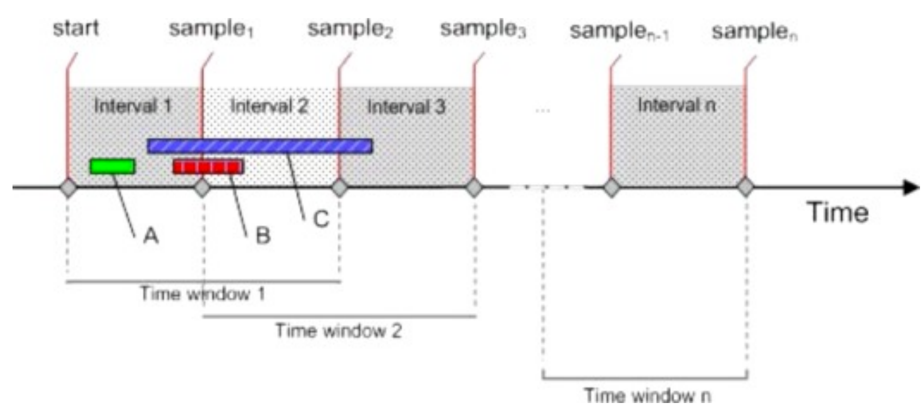
- **"Profiling": aggregating information for different events.**
  - Sampling: Statistical Information
  - Instrumentation: Catching every occurrence

- **"Tracing": saving every event.**
  - Instrumentation

l'Autònoma

Profiling vs Tracing

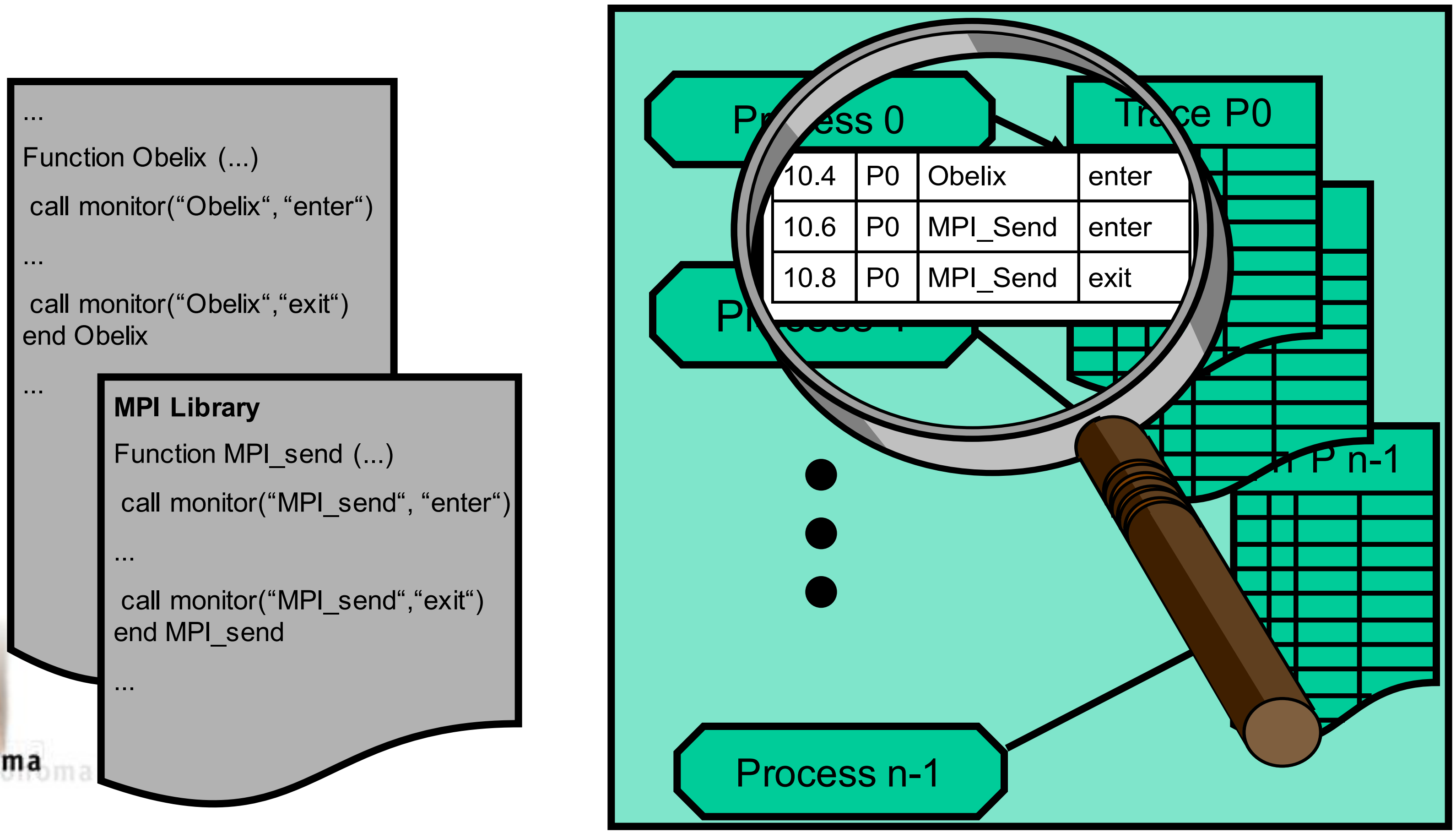
- **Profiling**
  - Summarized information
  - Quick global vision
  - Hints possible "bottlenecks"
  - Smaller files



l'Autònoma

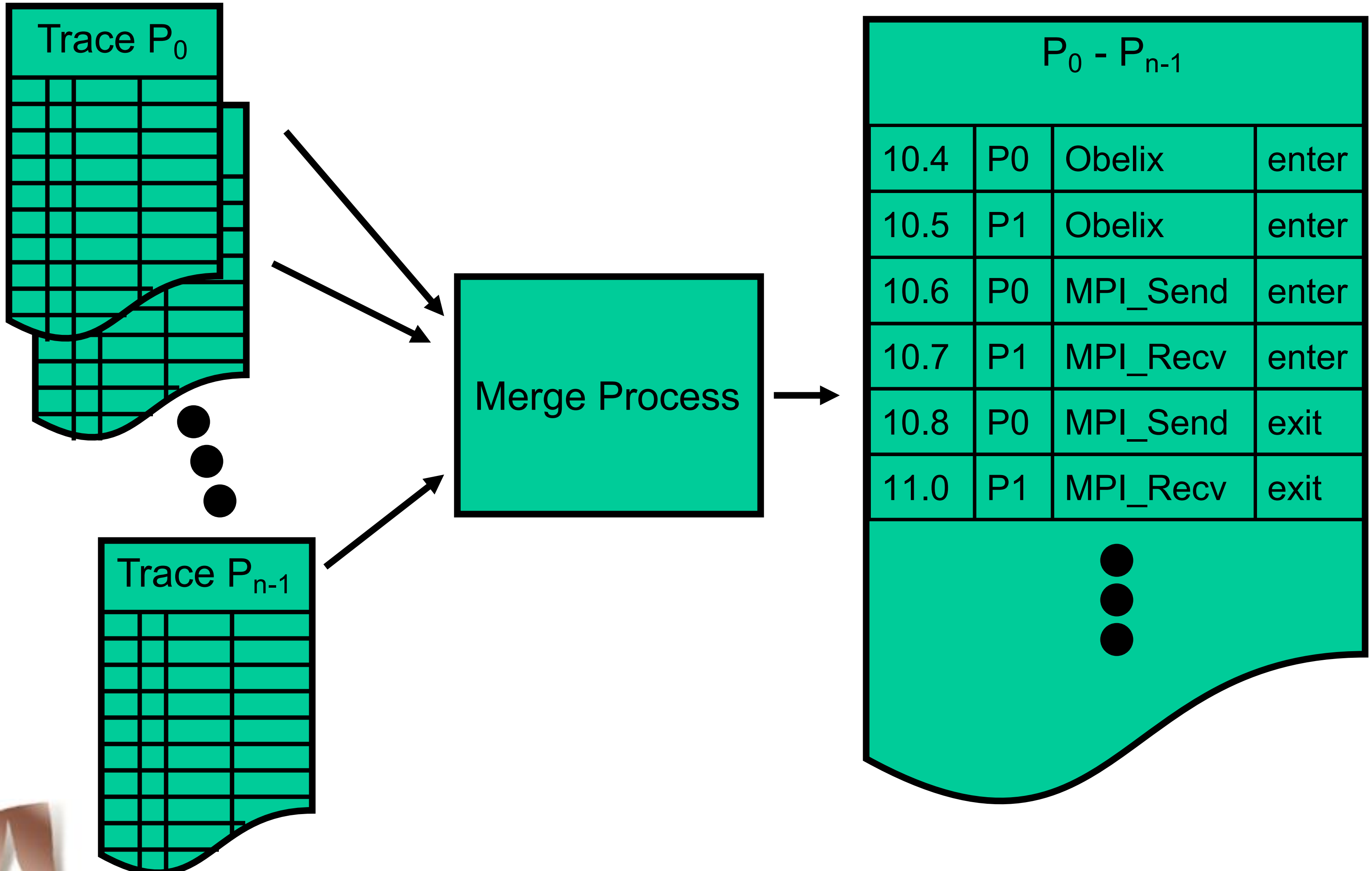
Profiling vs Tracing

- **Tracing**
  - The information of each event is available
  - The behavior of the application can be reproduced
  - Huge amounts of data can be generated (potentially)



l'Autònoma

Profiling vs Tracing



l'Autònoma

Wall clock time vs CPU time

CPU time is not wall time

CPU time is the total execution time or runtime for which the CPU was dedicated to a process.

The CPU must serve many processes every second, so your process only gets slices in between processing other requests.

Each of those small task slices is counted toward the total execution time.

While the CPU is processing someone else's request, is NOT counted towards CPU time of your process.

```
$ time gzip test.log
real  0m2.125s
user  0m1.920s
sys  0m0.170s
```

2.125 seconds of wall time ("real"),
1.920 seconds of CPU time ("user"),
0.170 seconds were spent in kernel mode ("sys").
0.035 seconds were time sliced to other processes.

l'Autònoma

Profiling tools (perf)

- **Linux profiling tool (perf)**

  https://perf.wiki.kernel.org/index.php/Main_Page

- **Tool integrated into Operating System**

- **Basic commands:**
  - perf stat

Profiling tools (perf)

$perf stat ./nn-vo-openmp

```
Performance counter stats for './nn-vo-openmp':

    130912,236653    task-clock (msec)          #    7,950 CPUs utilized
            10943    context-switches           #    0,084 K/sec
               15    cpu-migrations             #    0,000 K/sec
             3071    page-faults                #    0,023 K/sec
     424539090475    cycles                     #    3,243 GHz
     318924431738    stalled-cycles-frontend    #   75,12% frontend cycles idle
     194378774879    stalled-cycles-backend     #   45,79% backend  cycles idle
     203641975626    instructions               #    0,48  insns per cycle
                                                #    1,57  stalled cycles per insn
      21741577916    branches                   #  166,078 M/sec
          7475865    branch-misses              #    0,03% of  all branches

     16,467364814 seconds time elapsed
```

l'Autònoma

Profiling tools (perf)

```
$ perf stat -e cache-misses:u,cache-references:u,instructions:u  ./nn-vo-openmp

Performance counter stats for './nn-vo-openmp':

       331748390      cache-misses:u          #   41,436 % of all cache refs
       800626900      cache-references:u
    203364229778      instructions:u
    17,162008777 seconds time elapsed
```

```
$ perf stat -e cache-misses:u,cache-references:u,instructions:u –r 5 ./nn-vo-openmp

Performance counter stats for './nn-vo-openmp' (5 runs):

 307386670      cache-misses:u          #   40,068 % of all cache refs      ( +-  2,30% )
 767159580      cache-references:u                                          ( +-  1,22% )
 203154642251      instructions:u                                           ( +-  0,27% )
     16,247986563 seconds time elapsed                                      ( +-  1,44% )
```

l'Autònoma

Profiling tools (likwid)

- A set of tools developed by Erlangen Regional Computing Center (RRZE), University of Erlangen-Nuremberg
  https://github.com/RRZE-HPC/likwid/wiki

- Basic commands:
  - likwid-topology : A tool to display the thread and cache topology
  - likwid-perfctr : A tool to measure hardware performance counters
  - likwid-mpirun : Script enabling simple and flexible pinning of MPI and MPI/threaded hybrid applications.
- In lab you have to load a module first

  $module load likwid/4.0.1

l'Autònoma

Profiling tools (likwid)

```
$ likwid-topology
-------------------------------------------------------------
CPU name: Intel(R) Core(TM) i7 CPU        950  @ 3.07GHz
CPU type: Intel Core Bloomfield processor
CPU stepping: 5
*************************************************************
Hardware Thread Topology
*************************************************************
Sockets: 1
Cores per socket: 4
Threads per core: 2
-------------------------------------------------------------
HWThread Thread Core Socket Available
0          0      0    0        *
1          0      1    0        *
2          0      2    0        *
…
-------------------------------------------------------------
Socket 0: ( 0 4 1 5 2 6 3 7 )
-------------------------------------------------------------
```

l'Autònoma

Profiling tools (likwid)

```
********************************************************************************************
Cache Topology
********************************************************************************************
Level: 1
Size: 32 kB
Cache groups: ( 0 4 ) ( 1 5 ) ( 2 6 ) ( 3 7 )
-------------------------------------------------------------------------------------------
Level: 2
Size: 256 kB
Cache groups: ( 0 4 ) ( 1 5 ) ( 2 6 ) ( 3 7 )
-------------------------------------------------------------------------------------------
Level: 3
Size: 8 MB
Cache groups: ( 0 4 1 5 2 6 3 7 )
-------------------------------------------------------------------------------------------
```

l'Autònoma

Profiling tools (likwid)

```
$ likwid-perfctr -c 0-3 -g CACHE ./nn-vo-openmp
# profile for cache (-g) of cores 0-3 (-c)
```

...

Metric	Core 0	Core 1	Core 2	Core 3
Runtime (RDTSC) [s]	7.843182e+01	7.843182e+01	7.843182e+01	7.843182e+01
Runtime unhalted [s]	8.406984e+01	2.931435e-03	7.048604e-03	1.526573e-03
Clock [MHz]	3.368855e+03	3.036108e+03	3.104102e+03	2.854519e+03
CPI	1.367530e+00	3.646795e+00	4.814578e+00	3.093170e+00
Data cache misses	3741958252	53223	107572	19656
Data cache request rate	4.792796e-01	4.525067e-01	4.958291e-01	4.140127e-01
Data cache miss rate	1.958493e-02	2.130379e-02	2.364181e-02	1.281468e-02
Data cache miss ratio	4.086327e-02	4.707950e-02	4.768138e-02	3.095238e-02

...

l'Autònoma

Profiling tools (TAU)

TAU (Tuning and Analysis Utilities)

https://www.cs.uoregon.edu/research/tau/home.php

- A portable profiling and tracing toolkit supporting parallel programs written in Fortran, C, C++, Java, and Python.
- Instrumentation of functions, methods, basic blocks, and statements.
- The TAU API goes toward performance analysis, not only profiling and tracing

l'Autònoma

Profiling tools (TAU)

Provides two instrumentation options:

- **Dynamic through library preloading (I/O, MPI, Memory, CUDA, and OpenCL)**
  - The libraries chosen for pre-loading determine the scope of instrumentation.
  - MPI instrumentation is included by default, the others are enabled by command-line options to `tau_exec`.

- **Compiler based or Fortran 90, C, and C++**
  - TAU provides scripts: `tau_f90.sh`, `tau_cc.sh`, and `tau_cxx.sh` to instrument and compile Fortran, C, and C++ programs respectively.

l'Autònoma

Profiling tools (TAU)

- It can be installed for OpenMP, PAPI, MPI, CUDA

- For each installation a corresponding wrapper is generated

  Makefile.tau-openmp-opari
  Makefile.tau-pthread-openmp
  Makefile.tau-mpi
  Makefile.tau-mpi-openmp-papi

- An application is compiled and isntrumented using a chosen wrapper

  $export TAU_MAKEFILE=</path/to/the-wrapper>
  $tau_cc.sh <flags> application

- The application is executed normally and performance data are generated automatically

Profiling tools (TAU)

- **Textual visualization with pprof**

$ pprof

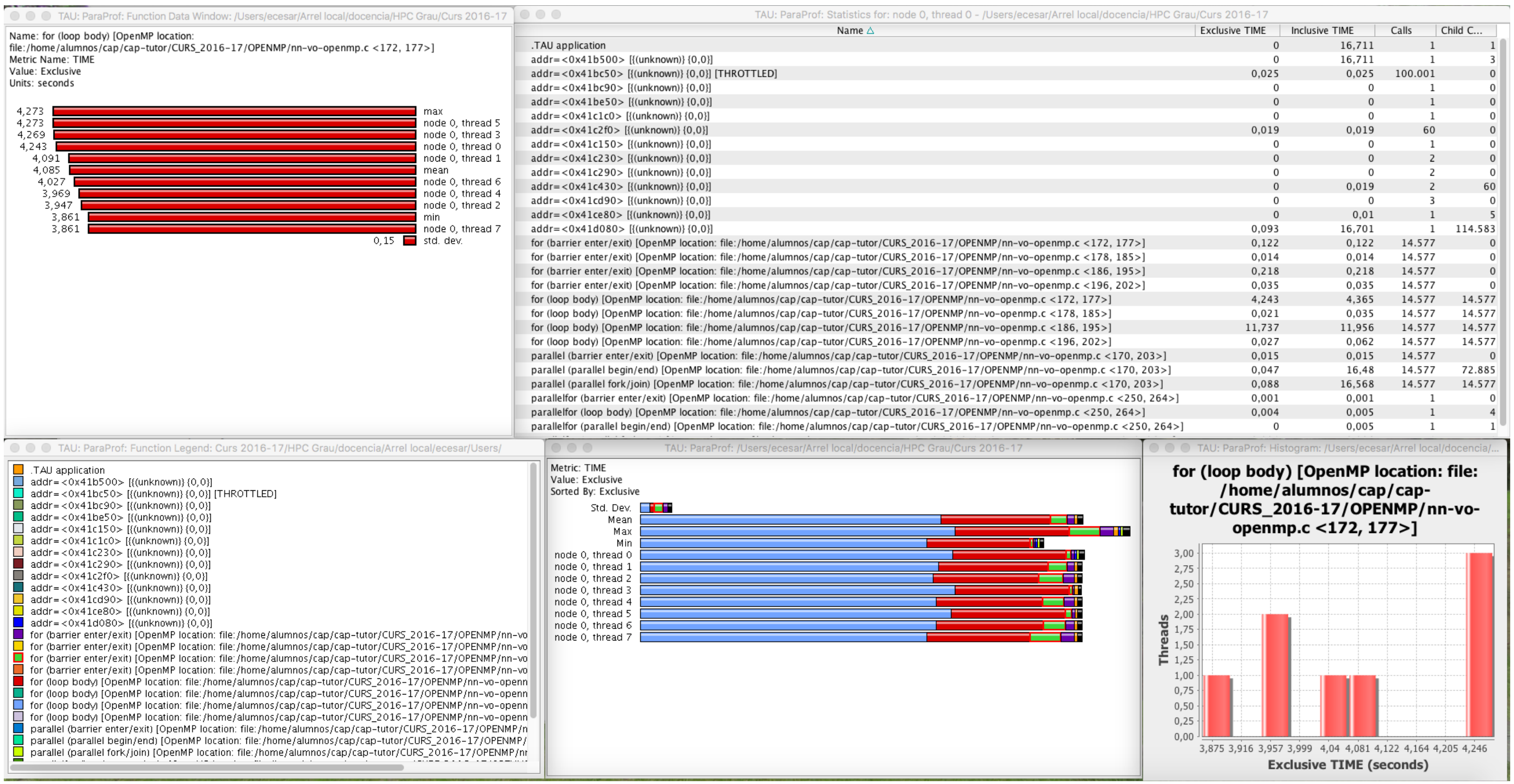
```
…
FUNCTION SUMMARY (mean):
---------------------------------------------------------------------------------------
%Time    Exclusive    Inclusive       #Call      #Subrs  Inclusive Name
              msec   total msec                          usec/call
---------------------------------------------------------------------------------------
100.0           99       16,614           1     12755.9   16614707 .TAU application
 99.2           50       16,481       14577       72885       1131 parallel (parallel begin/end) [OpenMP
location: file:/home/alumnos/cap/cap-tutor/CURS_2016-17/OPENMP/nn-vo-openmp.c  <170, 203>]
 72.0       11,307       11,955       14577       14577        820 for (loop body) [OpenMP location:
file:/home/alumnos/cap/cap-tutor/CURS_2016-17/OPENMP/nn-vo-openmp.c  <186, 195>]
 26.3        4,084        4,364       14577       14577        299 for (loop body) [OpenMP location:
file:/home/alumnos/cap/cap-tutor/CURS_2016-17/OPENMP/nn-vo-openmp.c  <172, 177>]
 12.6       0.0198        2,088       0.125       0.375   16711043 addr=<0x41b500>
 12.6           11        2,087       0.125     14322.9   16700640 addr=<0x41d080>
…
```

l'Autònoma

Profiling tools (TAU)

› Graphical visualization with paraprof

$ paraprof



Tracing tools (Vampirtrace)

- **Developed at TU Dresden**
- **Generates traces for MPI applications in OTF (open trace format)**
- **Information about:**
  - Point to point communications
  - Collective communications
  - I/O operations in MPI-2.
  - Source code
  - User defined events

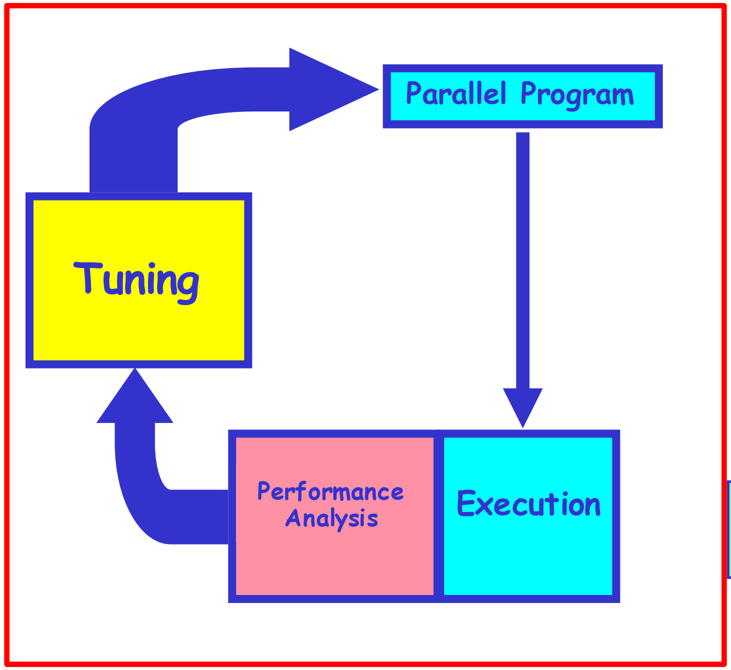
l'Autònoma

Performance Data

- Profiling and Tracing capture the behavior of the application.
- They must be used to improve the application performance.
- Consequently, this information must be appropriately presented to the user.

l'Autònoma

Performance Analysis and Tuning



Measurement

Execution Data

Analysing the Data

Detecting Performance Problems

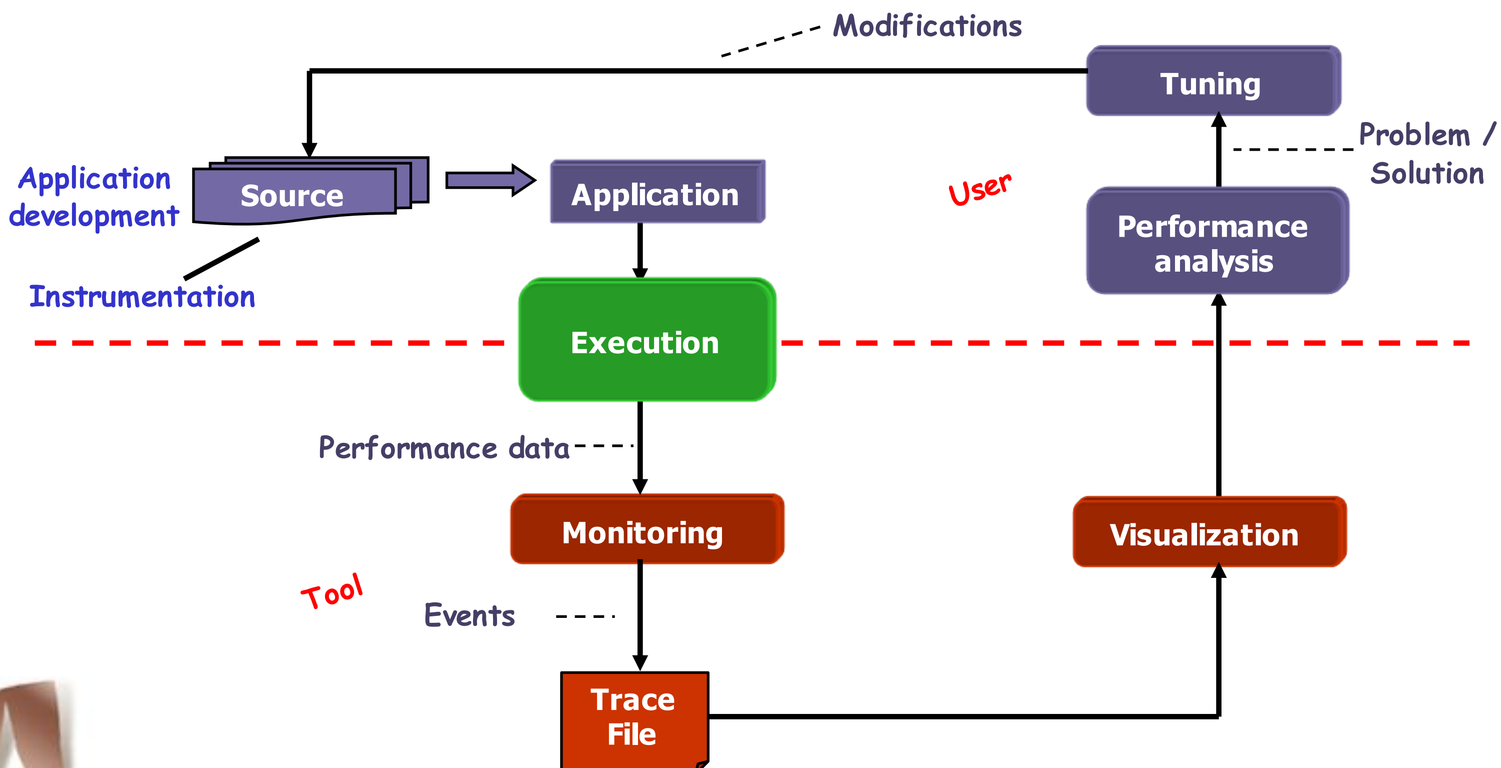
Finding their sources

Tuning

l'Autònoma

Performance Analysis: Approaches

- Classical: Obtaining multiple views of the data and its relationships using visualization tools such as Vampir or Paraver

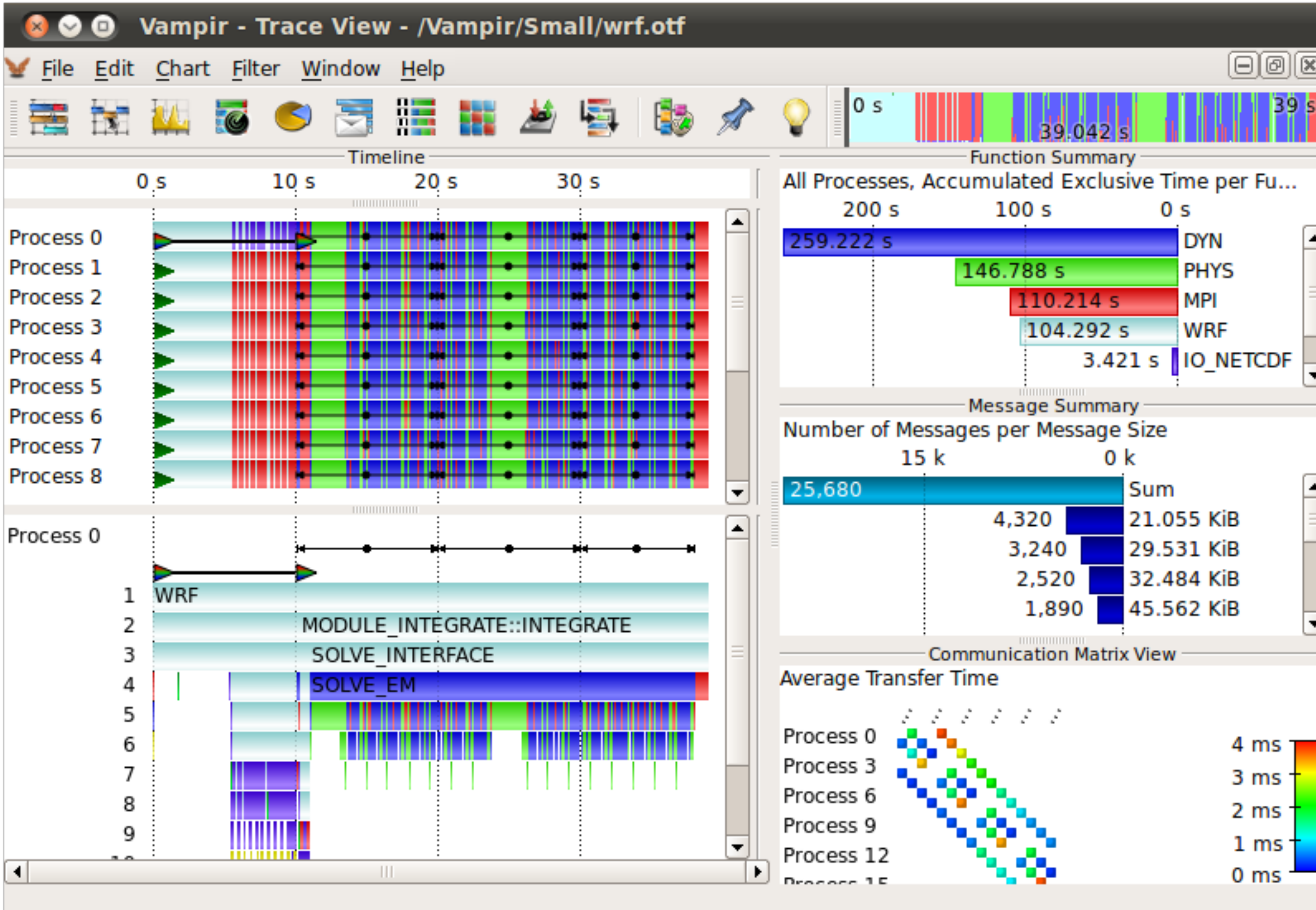


l'Autònoma

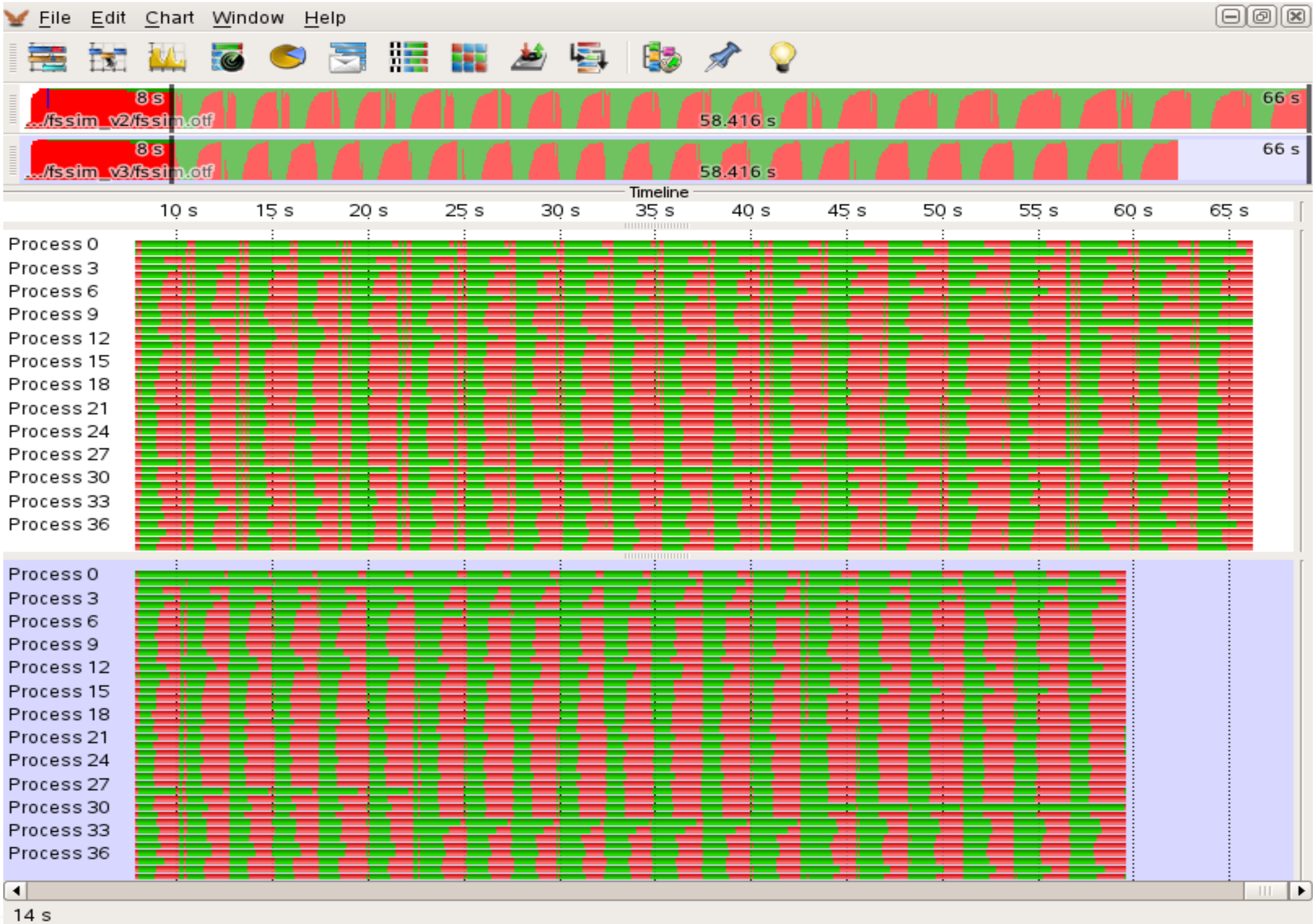
Classical: Vampir

- Visualization and Analysis of MPI pRograms
- Originally developed by Forschungszentrum Jülich.
- Currently maintained and commercialized by TU Dresden

Classical: Vampir



Classical: Vampir

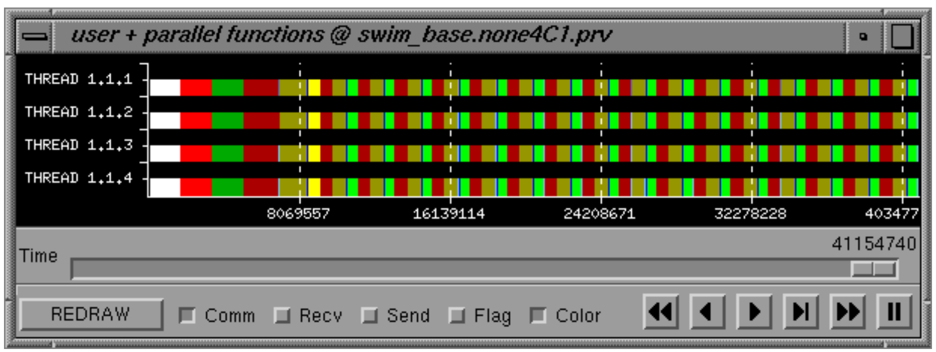


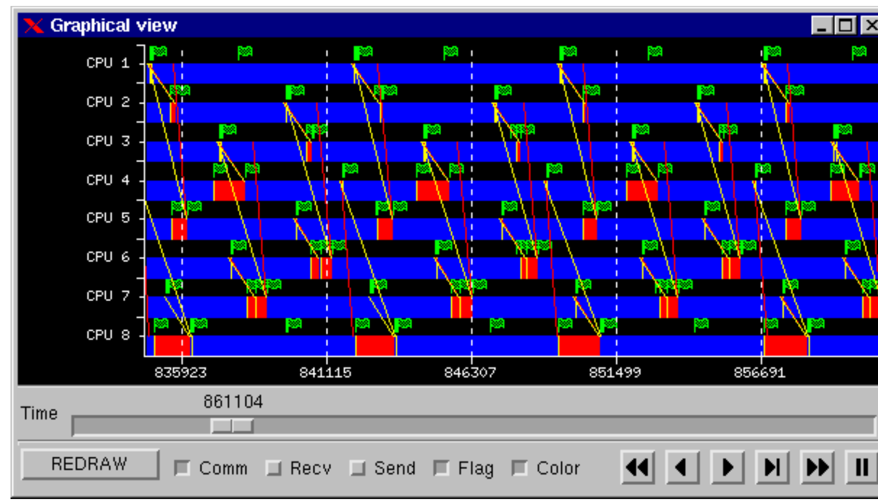
l'Autònoma

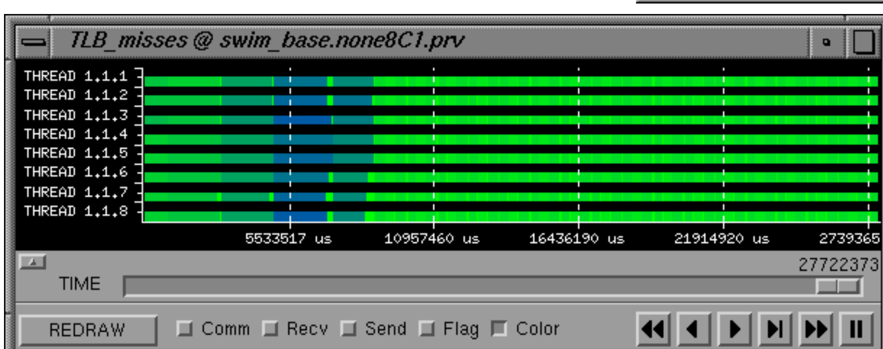
Classical: PARAVER

› Developed and maintained at the Barcelona Supercomputing Centre (BSC-UPC)

› It is a very powerful performance visualization tool based on traces.







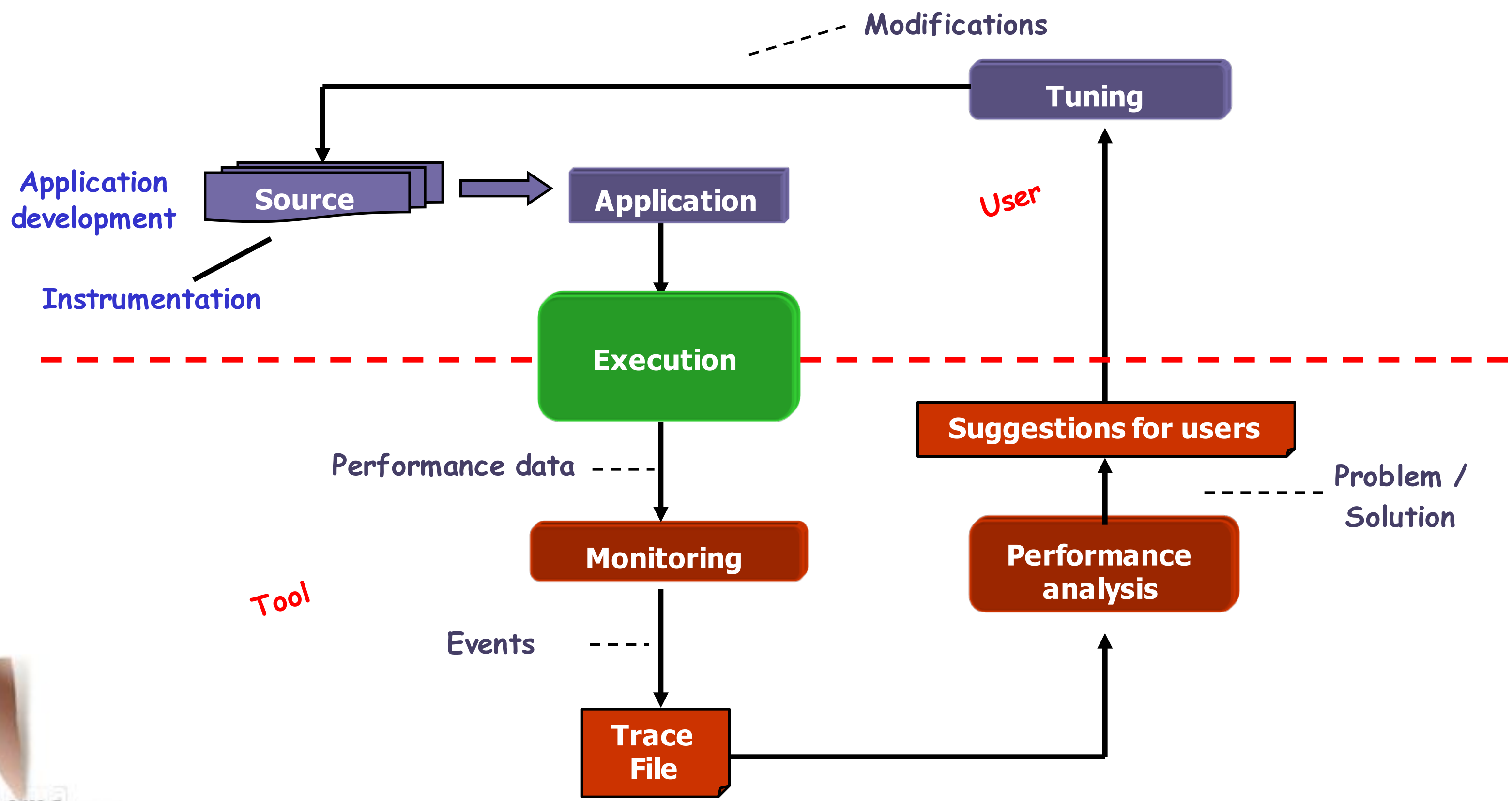
Visualization limits

- **Generally, visualization tools offer low level information:**
  - Communications
  - Cache misses
  - FLOPS
- **Many times it is quite difficult to understand an establish relationships among all the graphics**
- **It can be even more difficult to relate the visualized data with the source code**

l'Autònoma

Performance Analysis: Approaches

- Automatic: Obtaining information about performance inefficiencies using tools such as TAU, Dimemas, Scalasca or Periscope. It can be postmortem…



l'Autònoma

Automatic Postmortem: TAU

TAU instrumentation

Calls to the TAU API are made by probes inserted into the execution of the application via source transformation, compiler directives or by library interposition.

TAU profiling

After instrumentation and compilation are completed, the profiled application is run to generate the profile data files.

TAU tracing

Tracing the execution of a parallel program shows when and where an event occurred, in terms of the process that executed it and the location in the source code.

TAU Analysis

For a view of profile data – use **pprof** or **ParaProf**

For a view of trace data - use **JumpShot**

l'Autònoma

Automatic Postmortem: Scalasca

- It supports the performance optimization of parallel programs by measuring and analyzing their runtime behavior.

- The analysis identifies potential performance bottlenecks (communication and synchronization) and offers guidance in exploring their causes.

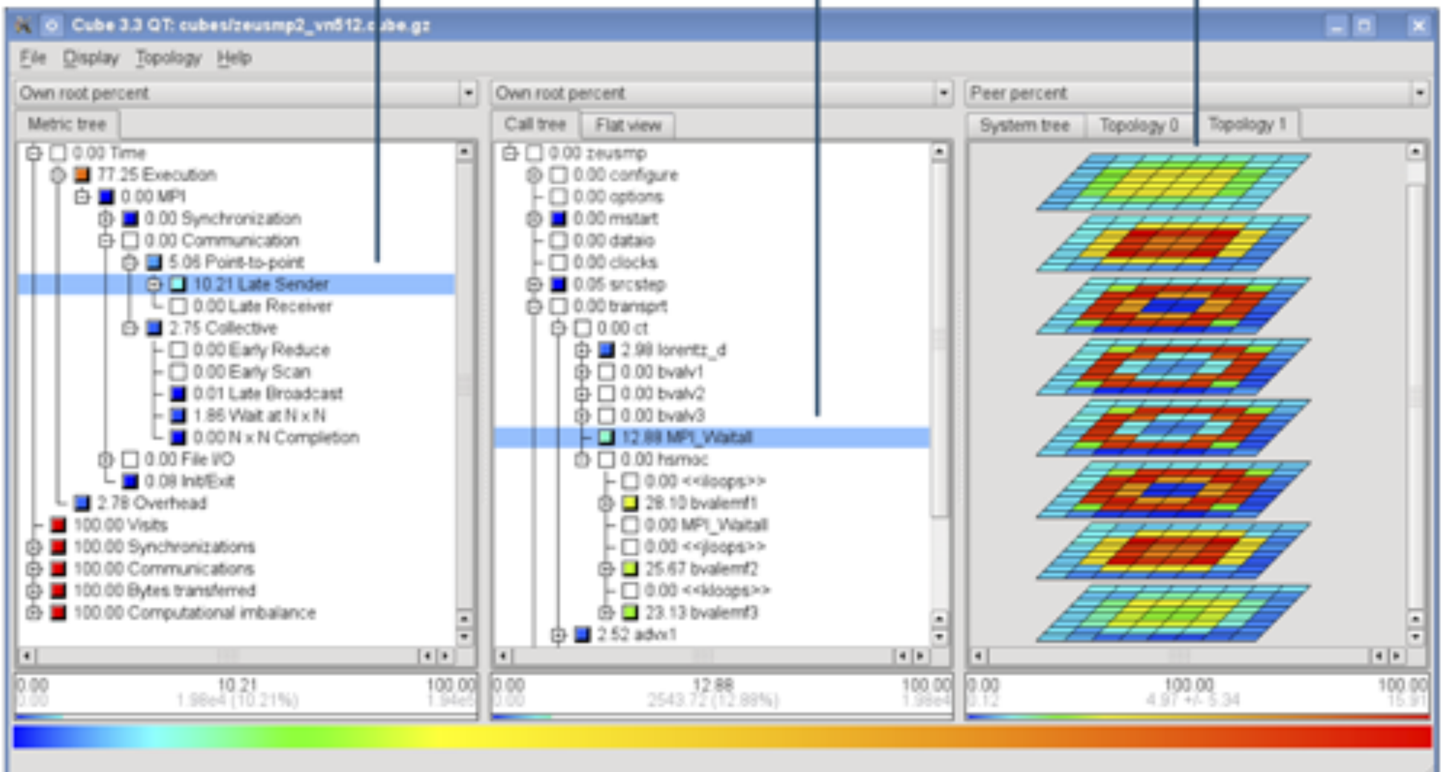
- Scalasca targets applications based on MPI and OpenMP.

- Scalasca is a joint project of:
  - Forschungszentrum Jülich, Jülich Supercomputing Centre
  - Technische Universität Darmstadt, Laboratory for Parallel Programming
  - German Research School for Simulation Sciences, Laboratory for Parallel Programming

l'Autònoma

Automatic Postmortem: Scalasca

Which performance problem?	Where in the program?	Where in the system?



Automatic Postmortem: Scalasca

1. Use the tool in profiling mode
2. Detect most inefficient call-paths
3. Insert instrumentation
4. Use the tool in event tracing mode: user can do things like comparing different traces to determine how effective an optimization has been

l'Autònoma

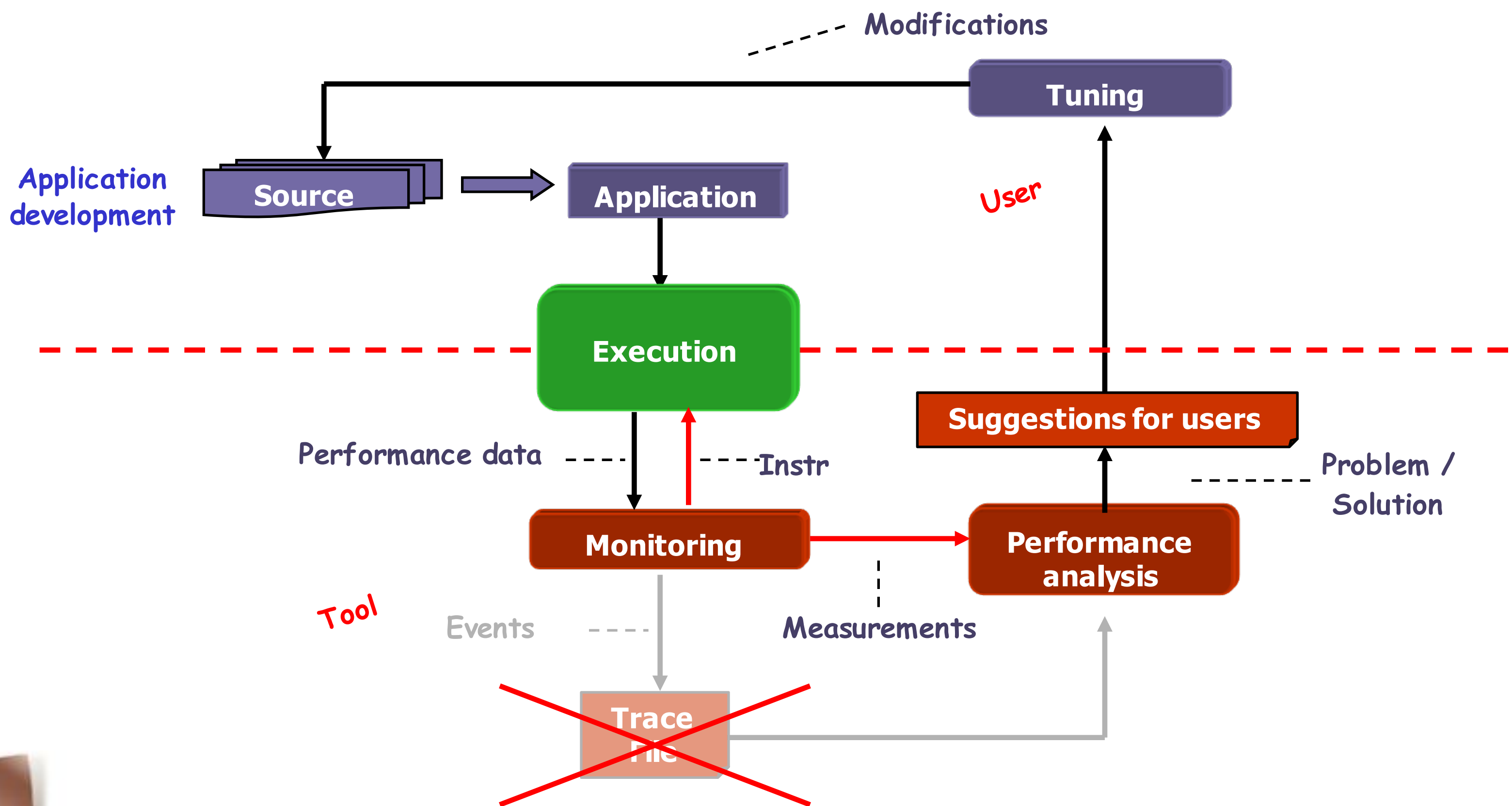
Automatic Postmortem: Dimemas

- Developed and maintained at the Barcelona Supercomputing Centre (BSC-UPC)
- Simulation tool for the parametric analysis of the behaviour of MPI applications on a configurable parallel platform.
- Based on one execution of the [MPI] application many configurations can be simulated with high precision.
- The tool produces traces that can be visualized using Paraver or Vampir.

l'Autònoma

Performance Analysis: Approaches

- … or dynamic



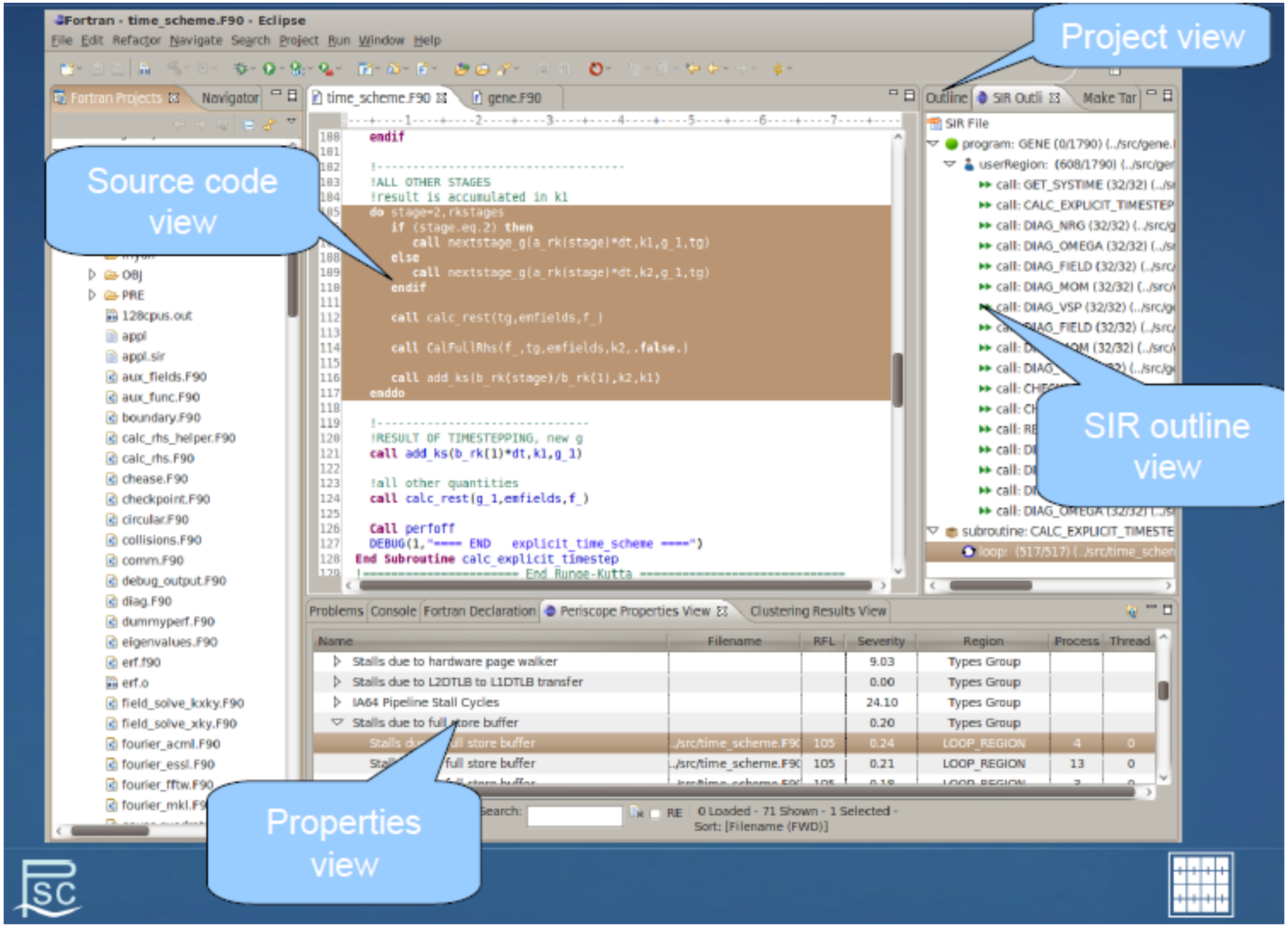
l'Autònoma

Automatic and Dynamic: Periscope

- **A scalable automatic performance analysis tool developed at Technische Universität München**
- **Iterative online analysis**
  - Measurements are obtained and evaluated on the fly
  - no tracing!
- **Distributed architecture**
  - Analysis performed by multiple distributed hierarchical agents
- **Automatic bottlenecks search**
  - Based on performance optimization experts' knowledge
- **Instrumentation**
  - Fortran, C/C++

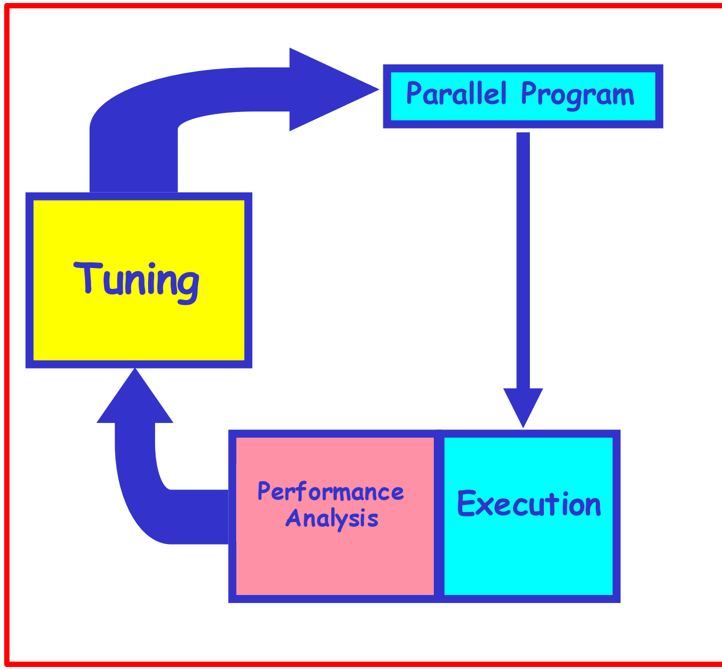
l'Autònoma

Automatic and Dynamic: Periscope



l'Autònoma

Performance Analysis and Tuning



Measurement

↓

Execution Data

↓

Analysing the Data

↓

Detecting Performance Problems

↓

Finding their sources

↓

Tuning

l'Autònoma

Performance analysis limits

- Fully instrumented application
- Can be trace file based analysis
- Difficult to relate bottlenecks to the source code
- Manual source code changes
- Recompilation, re-linking and restarting each time the source code has been changed

- Only for static applications
- Only for static environment

l'Autònoma

Performance Analysis and Tuning

- **Now we know the problems, so:**
  - Which are the causes?
  - Where the problem is in the source code?
  - How must this code be modified for solving the problem?
  - Can it be automated?

YES!

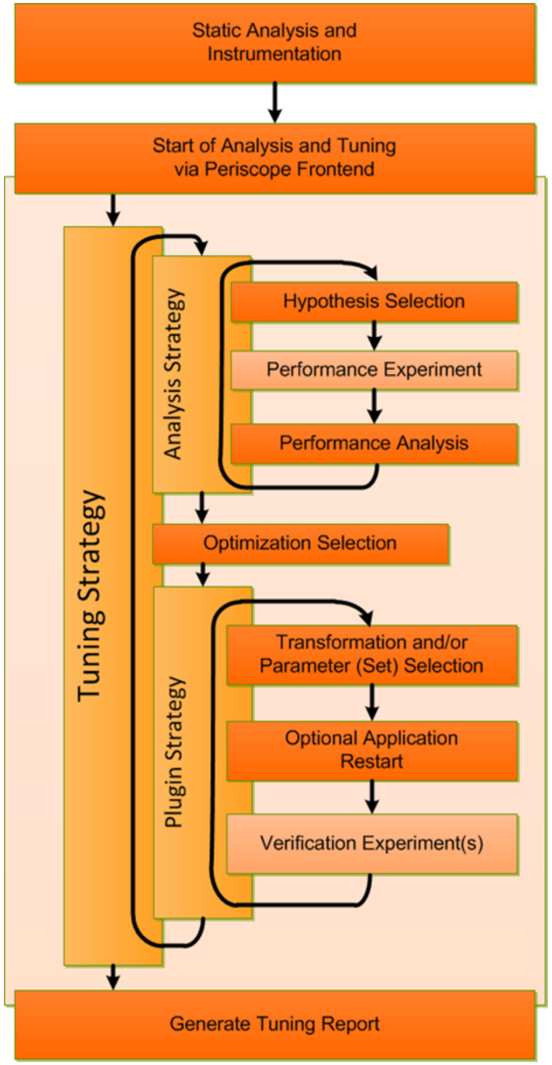
Autotuning

l'Autònoma

Autotuning: PTF

Periscope Tuning Framework (PTF)

- **Online**
  - Analysis and evaluation of tuned version in single application run
  - Multiple versions in single step due to parallelism in application
- **Result**
  - Tuning recommendation
  - Adaptation of source code and /or execution environment
  - Impact on production runs



l'Autònoma TUM universität wien CAPS UAB Universitat Autònoma de Barcelona lrz ICHEC SEVENTH FRAMEWORK PROGRAMME

Autotuning: PTF

- **Tackle complexity of HPC architectures**
  - Multicore, multisocket, accelerators, DVFS
- **Enable higher productivity via auto-tuning**
- **Focus on static tuning in pre-production phase**
  - Produce tuning recommendations
- **Leverage state of the art performance analysis**
- **Implement an extensible environment**
  - Support open and proprietary plugins

l'Autònoma

Autotuning: PTF

Partners:

- **UNIVIE**
  - High level Parallel Patterns for GPGPU Plugin
- **CAPS**
  - Hybrid Manycore Tuning: HMPP Codelet Tuning Plugin
- **LRZ**
  - Energy Consumption via CPU Frequency Tuning Plugin
- **UAB**
  - Master-Worker MPI Plugin
  - MPI Runtime Plugin
- **TUM**
  - Compiler Flag Selection Plugin
  - User-level Tuning Plugin
- **ICHEC**

l'Autònoma

Autotuning: PTF

Based on Periscope:

- **On-line**

  --no need to store trace files

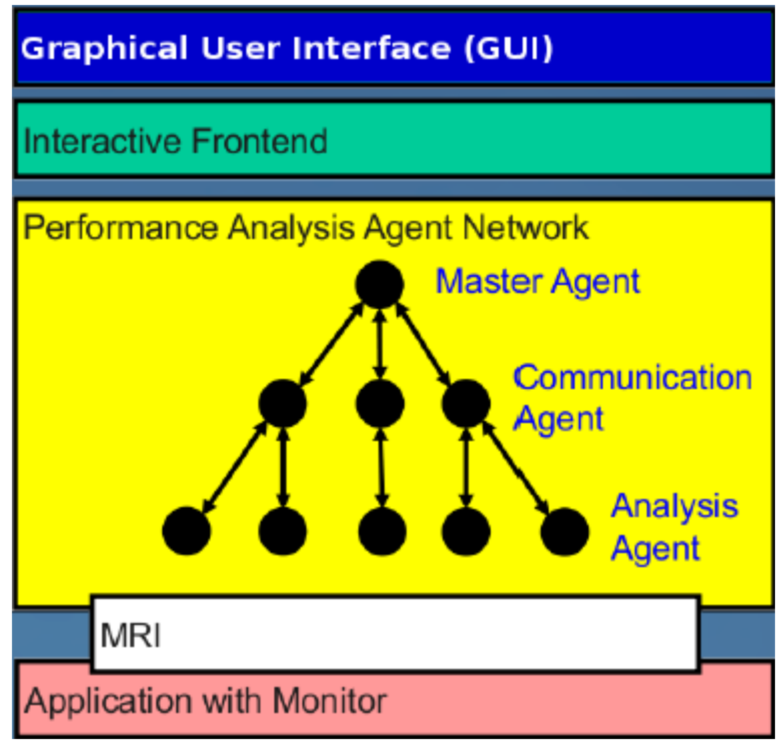
- **Distributed**

  -reduced network overhead

  -based on autonomous cooperating agents

- **Analyzes:**
  - MPI Communication
  - Single-node Performance
  - OpenMP Performance

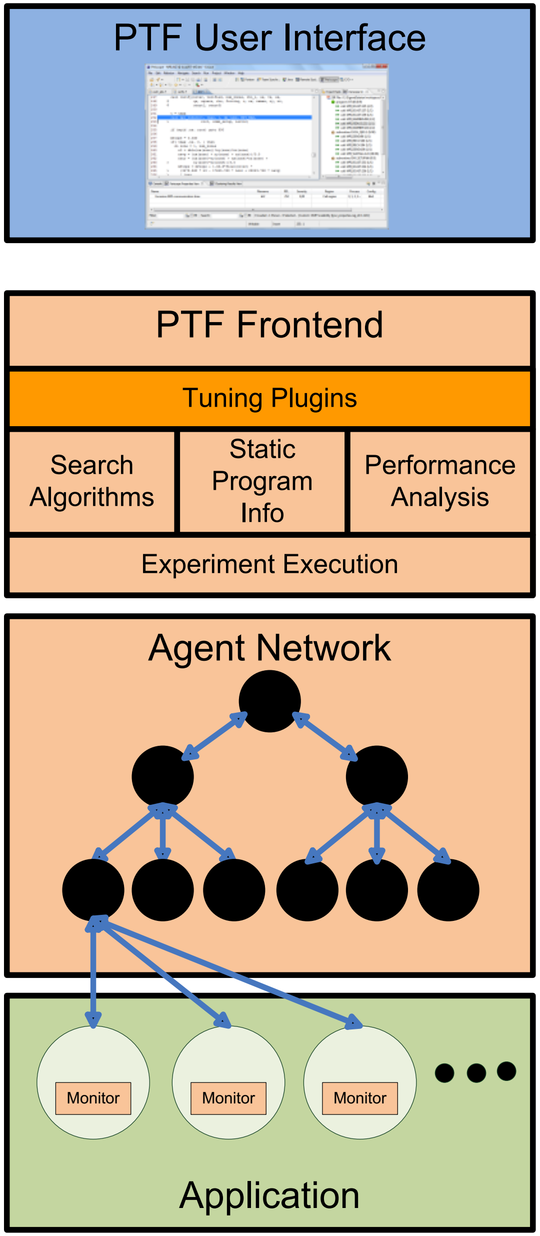
- **Supports: Fortran, C/C++**



l'Autònoma

Autotuning: PTF

- Extension of Periscope
- Online tuning process
  - Application phase-based
- Extensible via tuning plugins
  - Single tuning aspect
  - Combining multiple tuning aspects
- Rich framework for plugin implementation
- Automatic and parallel experiment execution



l'Autònoma

AUTOTUNE

www.autotune-project.eu

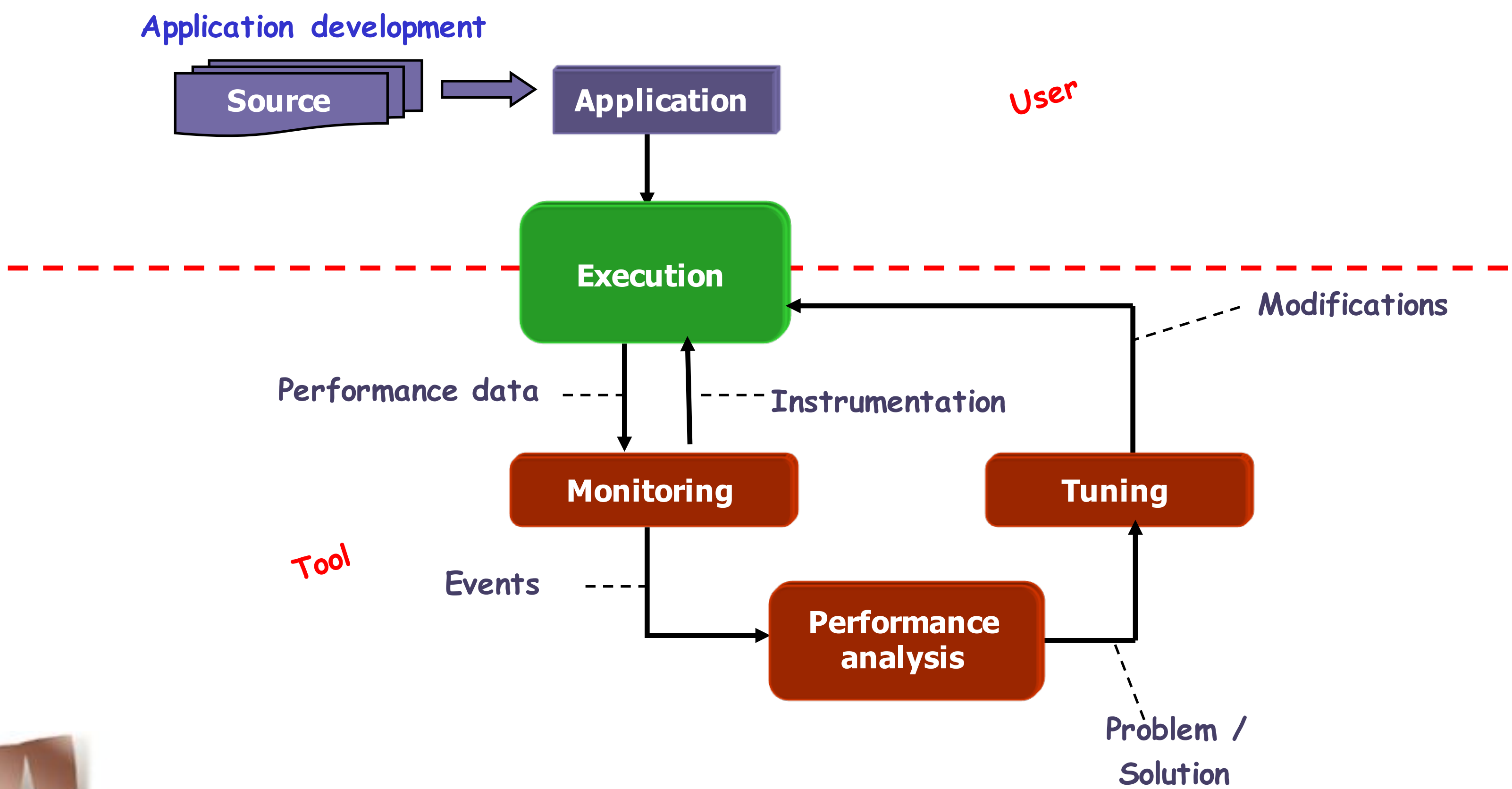
Performance Analysis and Tuning

- The behavior of the application changes from execution to execution or during a single execution, some parameters' values should be dynamically adapted to keep the application tuned

- Can the tuning process be automatic and dynamic?

l'Autònoma

Dynamic Tuning



l'Autònoma

Dynamic Tuning

- We need to be able to modify a program whilst it is running, without recompiling it, without stopping it

Well, luckily we still have Dyninst!

- There are tools, such as Active Harmony or MATE, that can do these kinds of things

l'Autònoma

Dynamic Tuning: MATE

MATE – **Monitoring, Analysis and Tuning Environment (CAOS-UAB)**

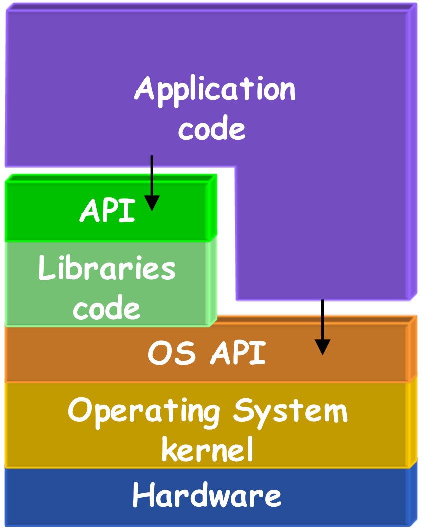
- **The tool implemented in C++**
- **Uses Dyninst**
- **For PVM and MPI applications**
- **Sun Solaris 2.x / SPARC y LINUX**

l'Autònoma

Dynamic Tuning: MATE

Tuning layers

- Application specific code
- Standard and custom libraries (API+code)
- Operating system libraries (API+code)
- Hardware



l'Autònoma

Dynamic Tuning: MATE

Application

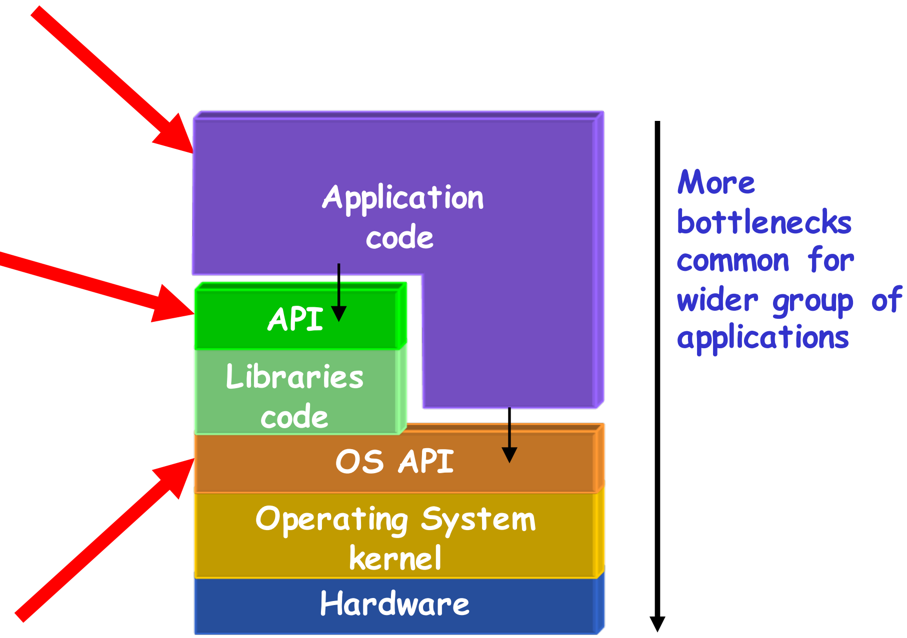
- **Application code changes**
  - Different bottlenecks that depend on the application implementation

Libraries

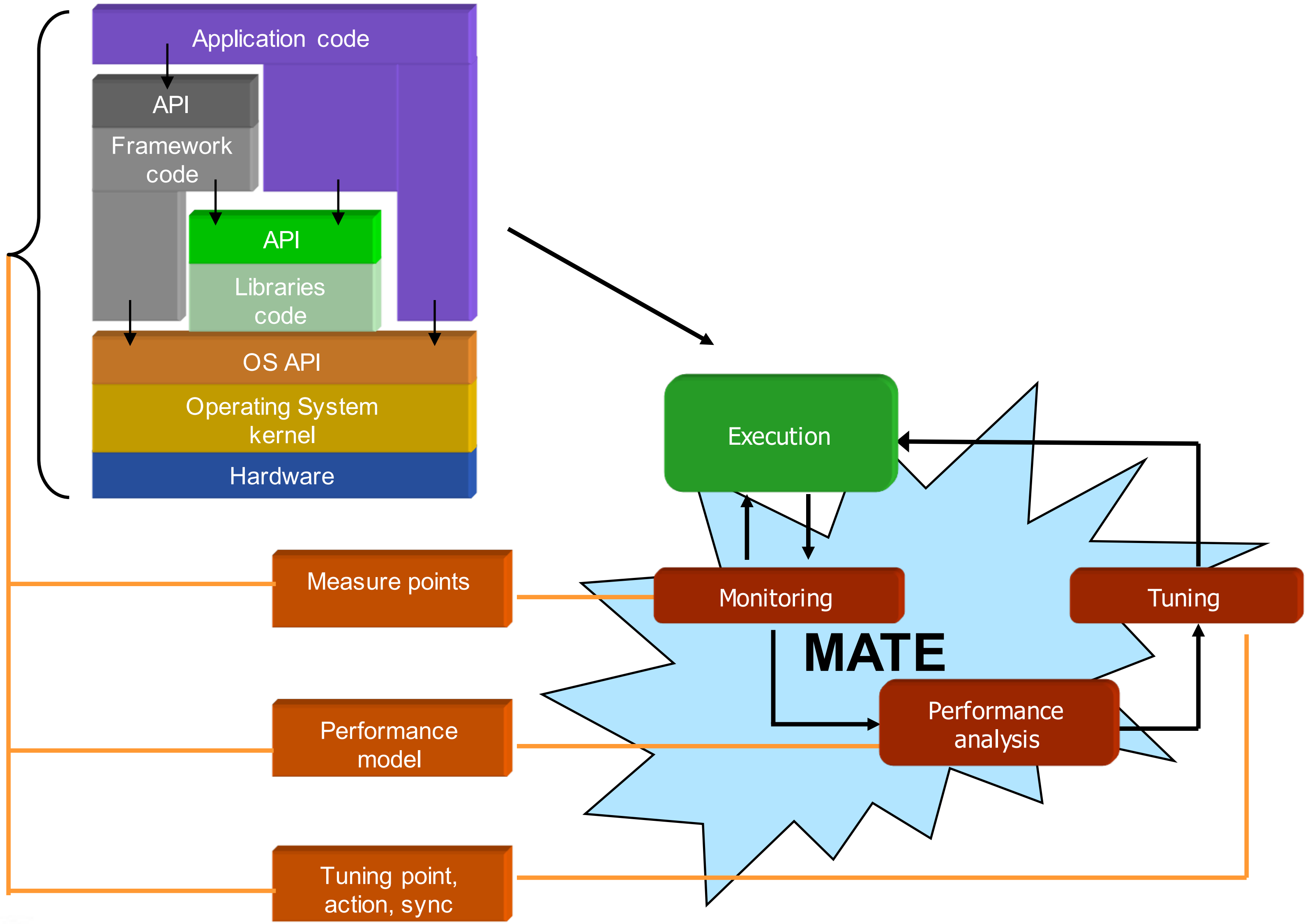
- **Library code changes**
- **API usage**
  - Standard
    - C/C++ library -> memory management, dynamic containers
  - Custom
    - PVM, MPI -> communication

OS

- **Kernel code changes**
- **API usage**
  - Adjustment of options (e.g. TCP/IP socket), I/O request grouping



Dynamic Tuning: MATE

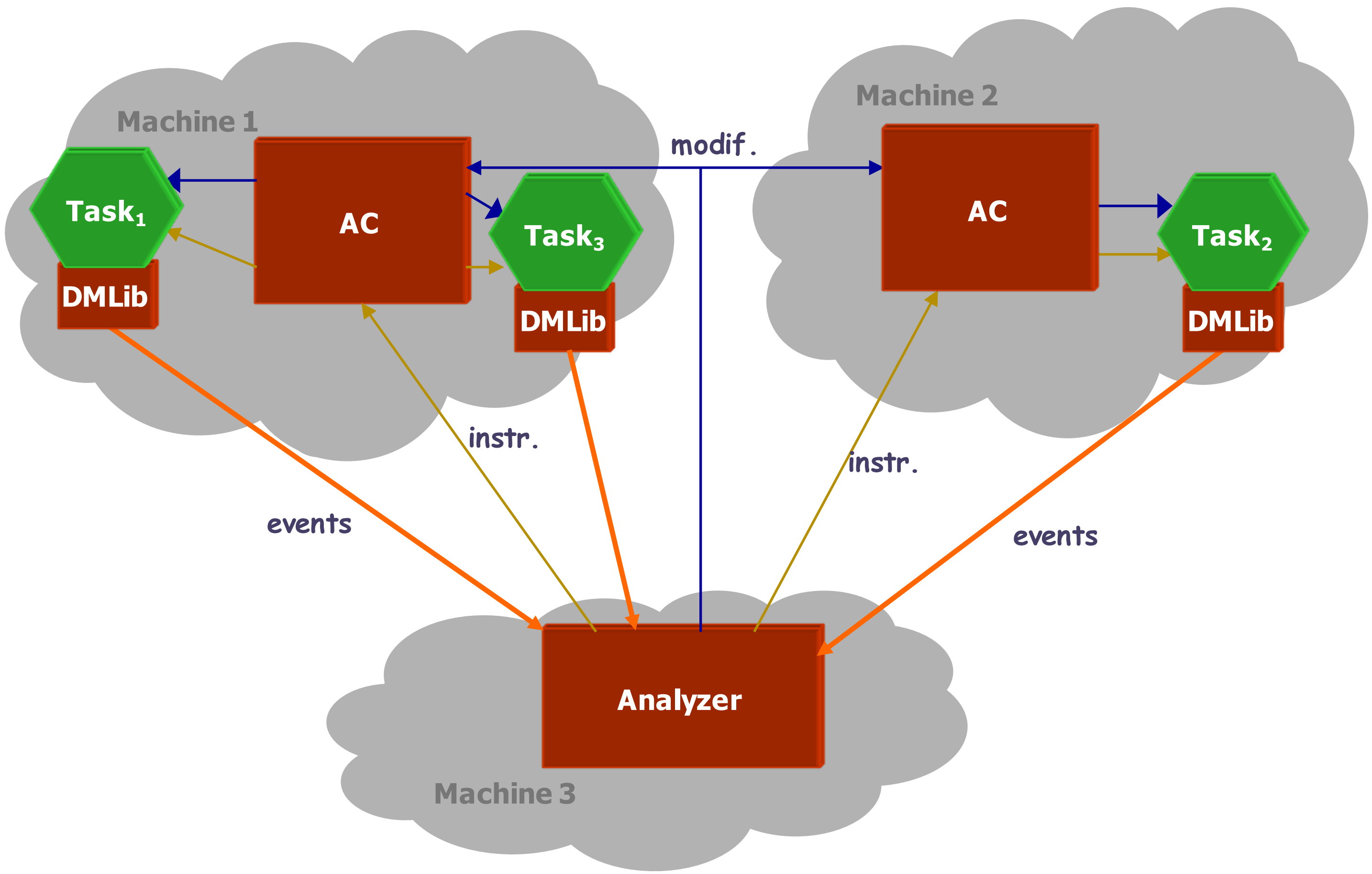


l'Autònoma

Dynamic Tuning: MATE

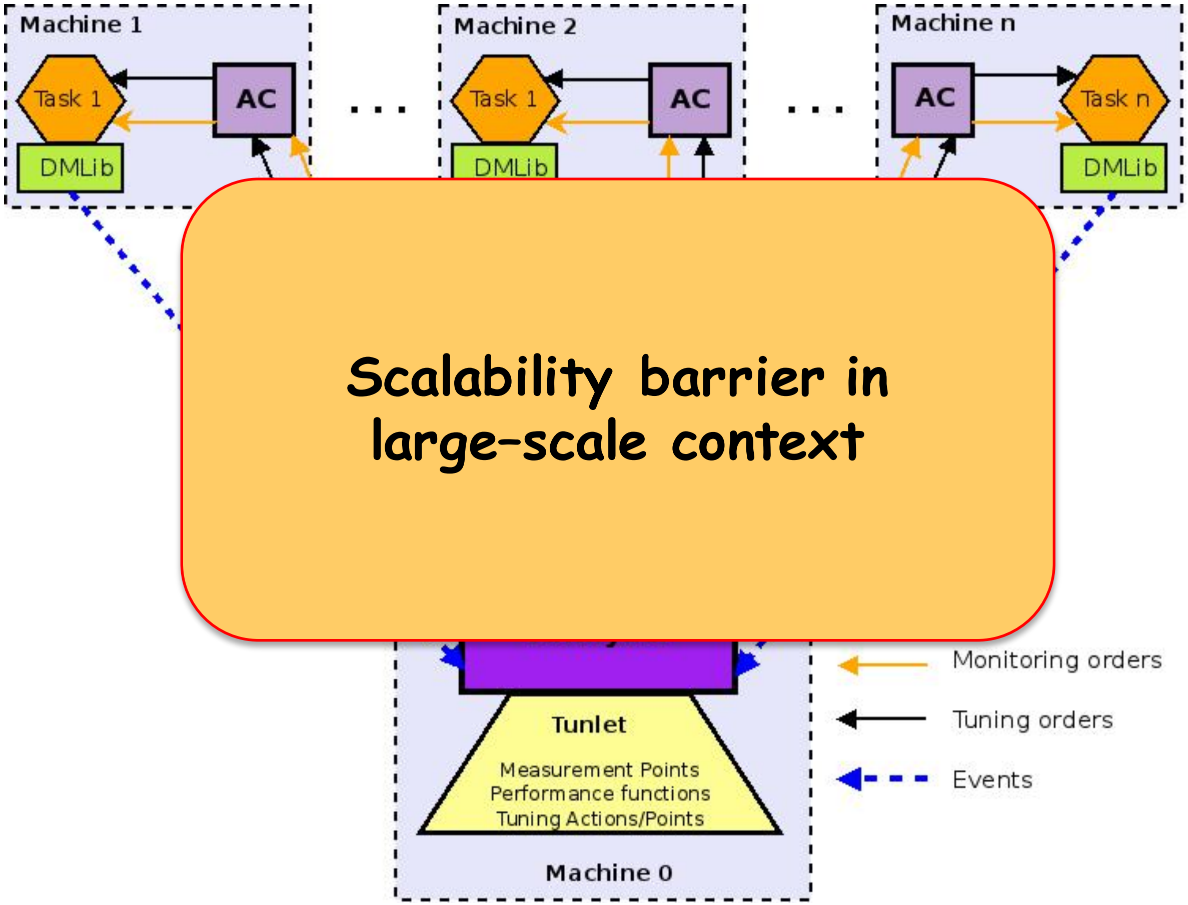
Architecture

- Application Controller - AC
- Dynamic Monitoring Library - DMLib
- Analyzer



l'Autònoma

Dynamic Tuning: limits

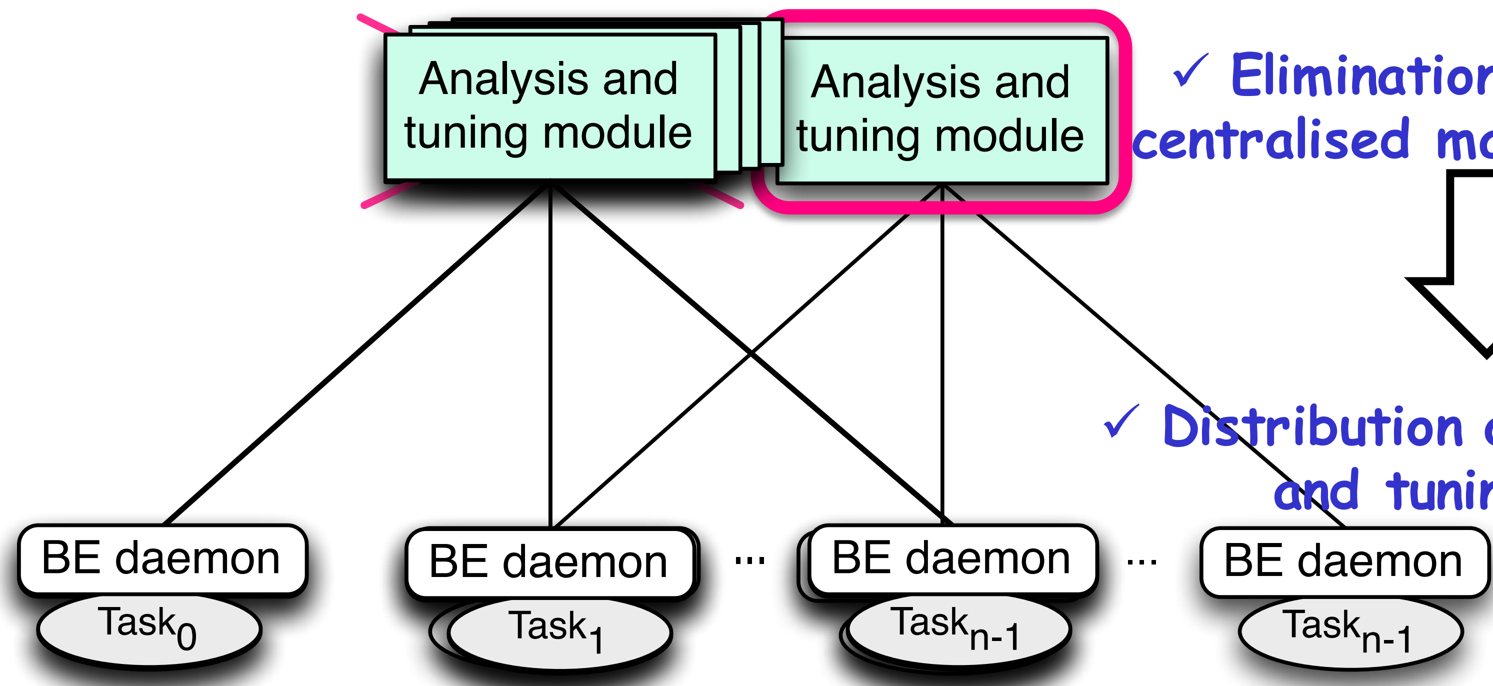


Scalability barrier in large-scale context

l'Autònoma

Dynamic Tuning: limits

Centralised Architecture of Tuning Tools

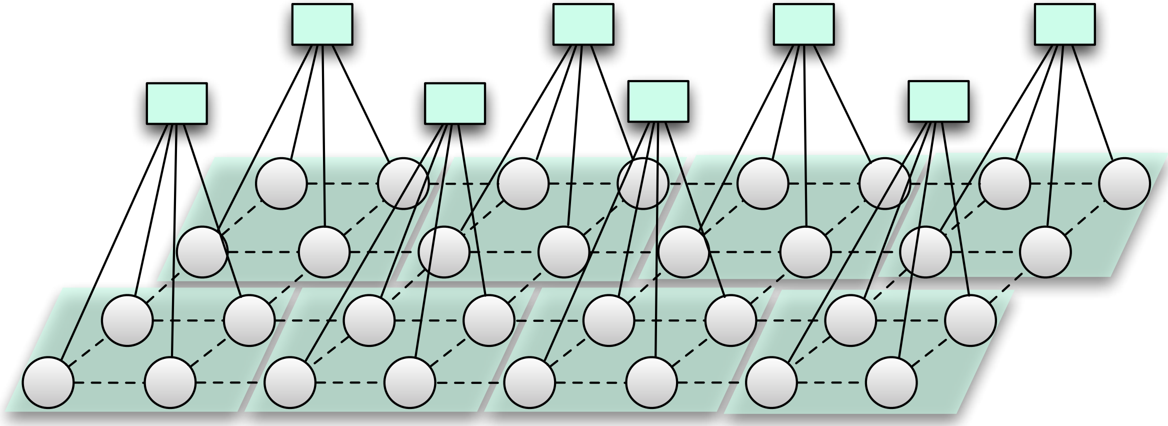


- ✓ Elimination of a single centralised management point
- ✓ Distribution of the analysis and tuning process

Performance Model
Monitoring Points
Performance Expressions
Tuning Points, Actions and Synchronisation Method

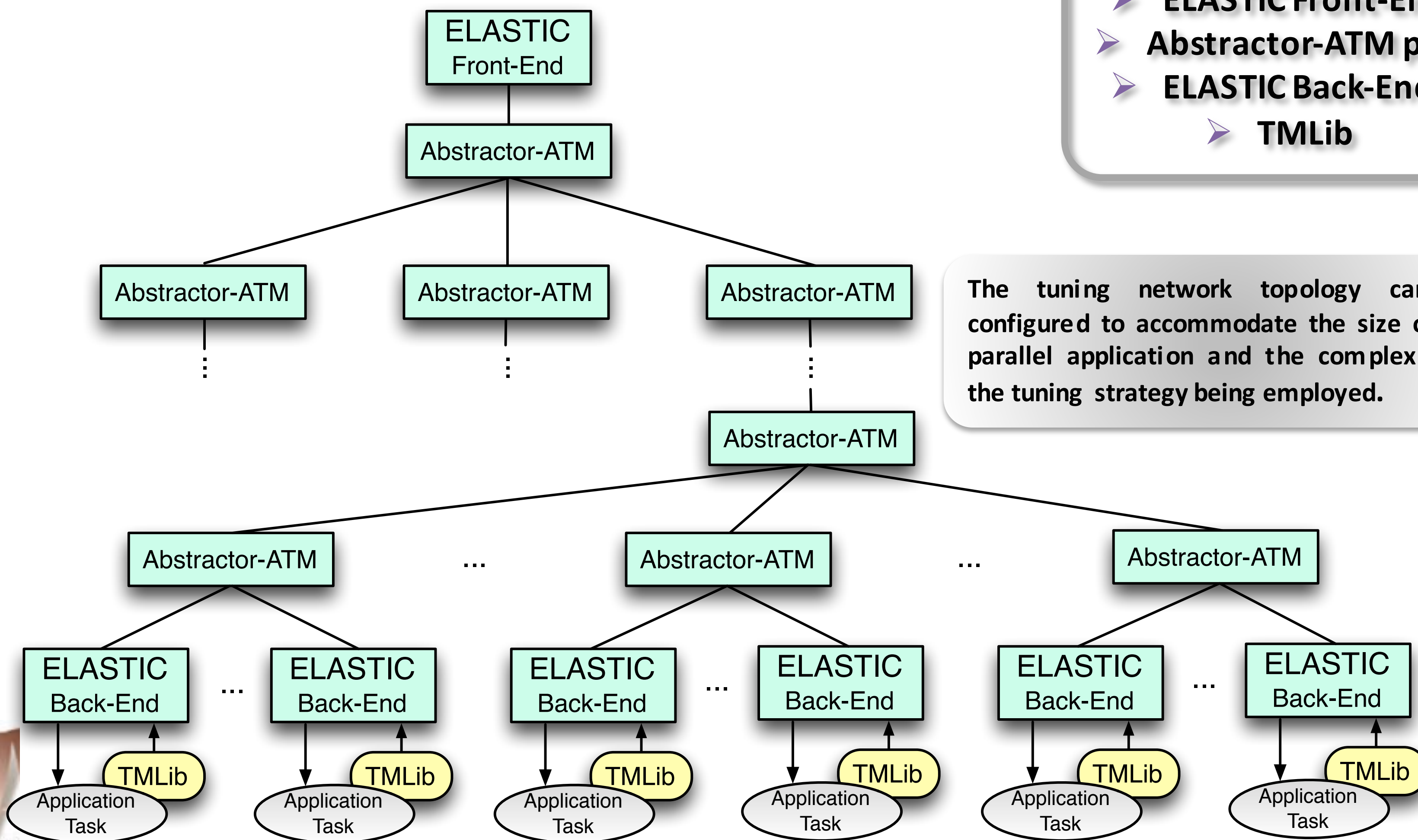
Dynamic Tuning: ELASTIC

- Hierarchical tuning network
- Representing the application tasks as a virtual parallel application



l'Autònoma

Dynamic Tuning: ELASTIC



- ELASTIC Front-End
- Abstractor-ATM pair
- ELASTIC Back-Ends
- TMLib

The tuning network topology can be configured to accommodate the size of the parallel application and the complexity of the tuning strategy being employed.

Dynamic Tuning: ELASTIC

PLUGIN ARCHITECTURE

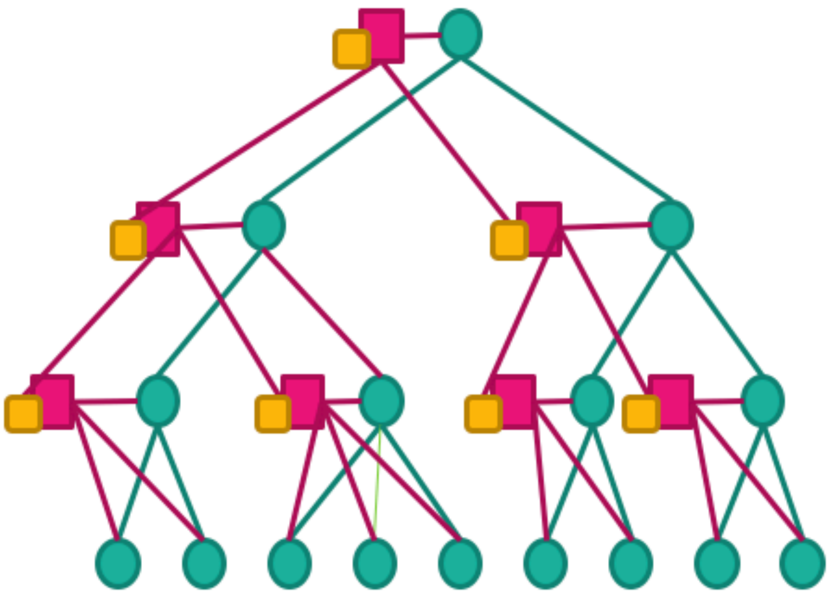
- Codification of the ELASTIC Package based on sub-classing Abstractor-ATM components.

This plugin architecture converts ELASTIC into a general purpose tuning tool and gives it the flexibility to tackle a wide range of performance problems

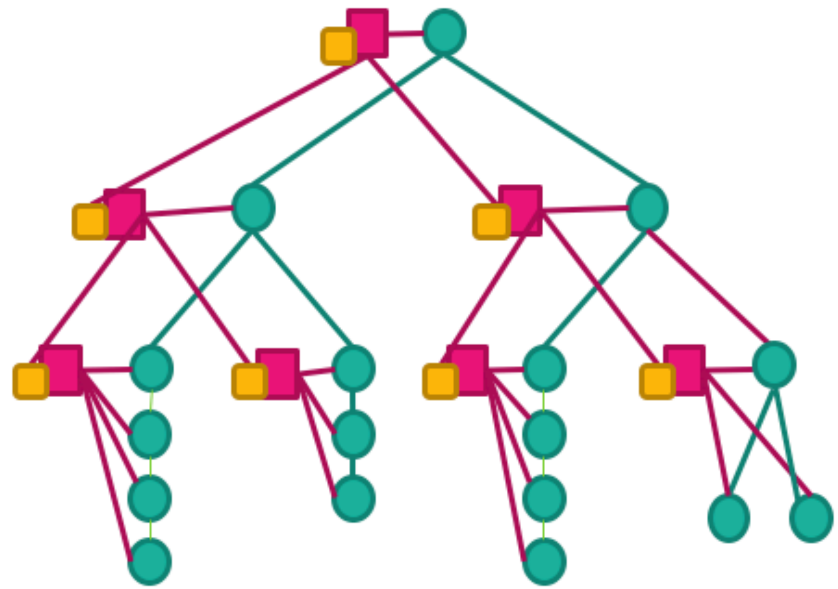
l'Autònoma

Dynamic Tuning: ELASTIC

Hierarchical Master/Worker



Master/Worker of pipelines



l'Autònoma