



Introduction to High Performance Computing and Performance Metrics and Tools

Aina Gomila-Olvera (BSC), Anke Visser (JSC), Katharina Haus (JSC)

HORIZON-EUROHPC-JU-2023-COE



EuroHPC
Joint Undertaking

1 January 2024– 31 December 2026

Grant Agreement No 101143931

Introducing the team!



Aina Gomila-Olvera (*aina.gomila@bsc.es*)

Barcelona Supercomputing Center

Best Practices for Performance and Programmability

Performance prediction of HPC workloads using ML

Katharina Haus (*k.haus@fz-juelich.de*)

Jülich Supercomputing Center

ATML Parallel Performance

Development of Scalasca Trace Tools



Anke Visser (*a.visser@fz-juelich.de*)

Jülich Supercomputing Center

ATML Parallel Performance

Development of Cube, education

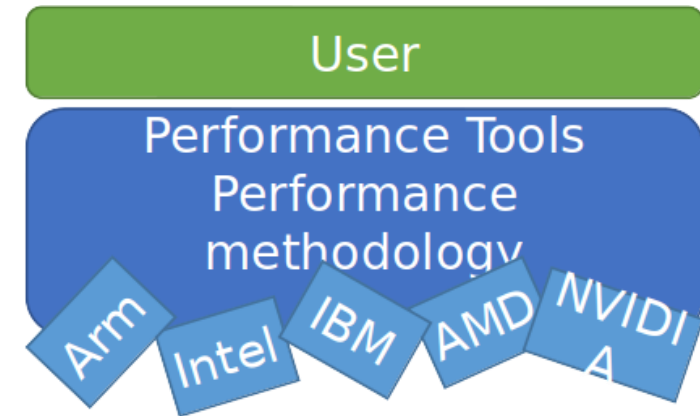
10/6/2026

- A **Centre of Excellence**
 - On **Performance Optimisation and Productivity**
 - Promoting **best practices in parallel programming**
- Providing **FREE Services**
 - Precise understanding of application and system behaviour
 - Suggestion/support on how to refactor code in the most productive way
- **Horizontal**
 - Transversal across application areas, platforms, scales
- **For (EuroHPC) academic AND industrial codes and users!**

Context and history



- **POP-CoE** starts in 2015
- Objectives:
 - To offer performance analysis services to European HPC users.
 - To develop a common methodology for performance analysis
- **POP1** runs from October 2015 to March 2018
- **POP2** runs from December 2018 to June 2022
- **POP3** will run between January 2023 and December 2026
- Upcoming **POP4** will run between January 2027 and December 2029



Partners



• Who?

- BSC, ES (coordinator)
- HLRS, DE
- INESC-ID, PT
- IT4I, CZ
- JSC, DE
- RWTH Aachen, IT Center, DE
- TERATEC, FR
- UVSQ, FR



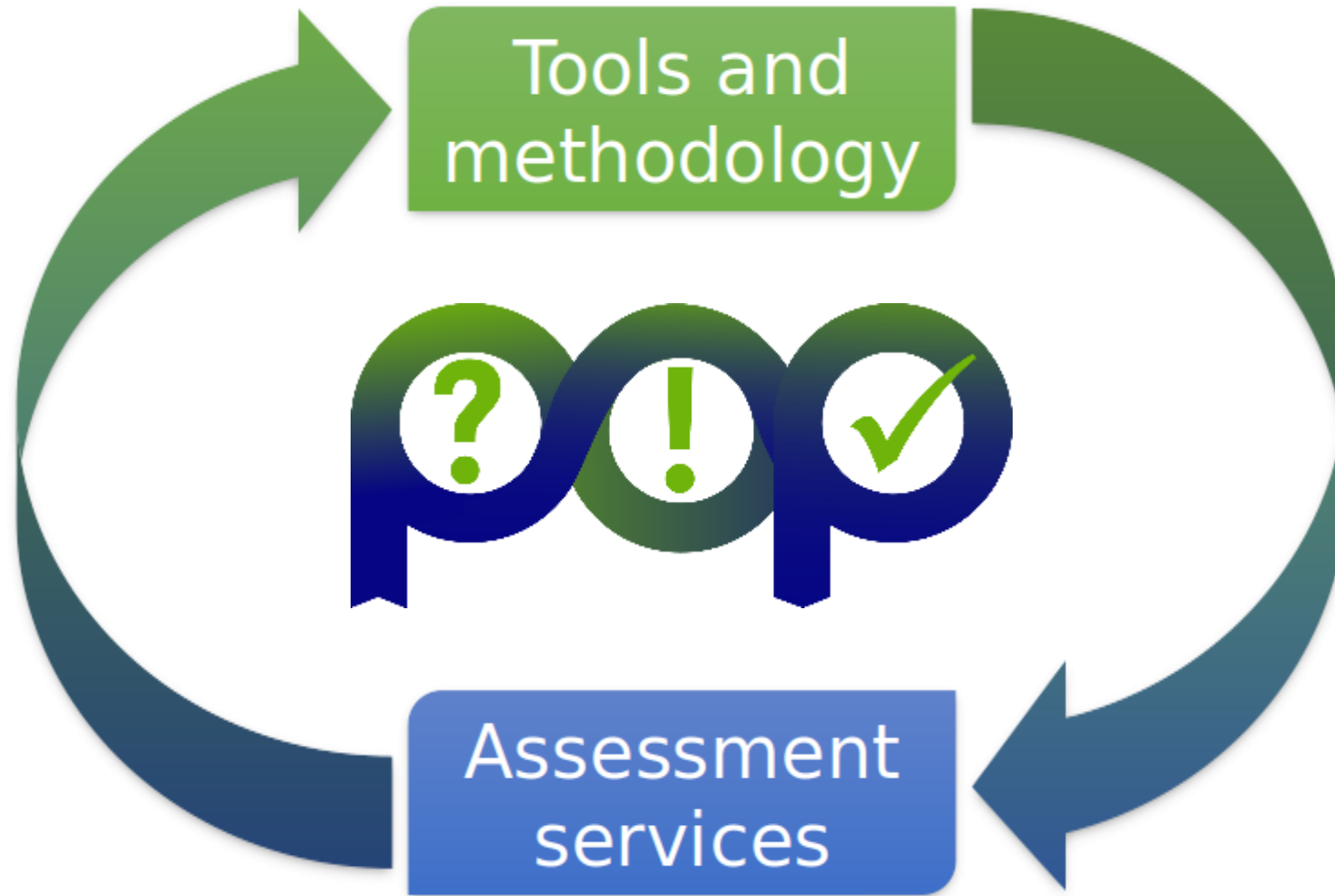
Why it works?

- Diversity of expertise to cover all areas
 - Some of them tool developers, some of them analysts, some customer advocate
- Diversity of tools for resiliency
- Diversity of backgrounds to design a common methodology

Objectives



Objectives



Background: WHPC chapters



Presence in
conferences (ISC,
HiPEAC, JURES)



- Objectives:
 - Empower and highlight women in HPC
 - Create a network of support among women in HPC



Organize BoF at ISC and
SC for several editions



Organize session at
EuroPAR since 2024

10/6/2026

Background: WHPC chapters



Contact us and stay up-to-date with WHPC!

- Website: <https://womeninhpc.org/>
- LinkedIn: <https://www.linkedin.com/company/women-in-hpc/>



To affiliate yourself to an existing chapter:

- Follow the steps for individual registration and select your regional chapter
- <https://womeninhpc.org/membership/individual>



To create a new WHPC chapter:

- Follow the steps for the “WHPC Chapter Application” form
- <https://whpctransfer.wpenginepowered.com/membership/chapters-and-affiliates/chapters>

- Introduction to High Performance Computing
 - Anatomy of a supercomputer and how to program and use a supercomputer
- Introduction to performance analysis
 - Performance concepts and metrics
- Performance analysis in action
 - Performance analysis tools, profiling versus tracing and demonstration of a performance analysis



Introduction to High Performance Computing

Aina Gomila-Olvera (BSC)

HORIZON-EUROHPC-JU-2023-COE



EuroHPC
Joint Undertaking

Grant Agreement No 101143931

1 January 2024– 31 December 2026



What is a supercomputer?

Supercomputers



- **Big, fast and powerful** computers.
- How big? They take up a whole room!



The 1st Supercomputer was made back in the 1960s, from Seymour Roger Cray at the Control Data Corporation (CDC)

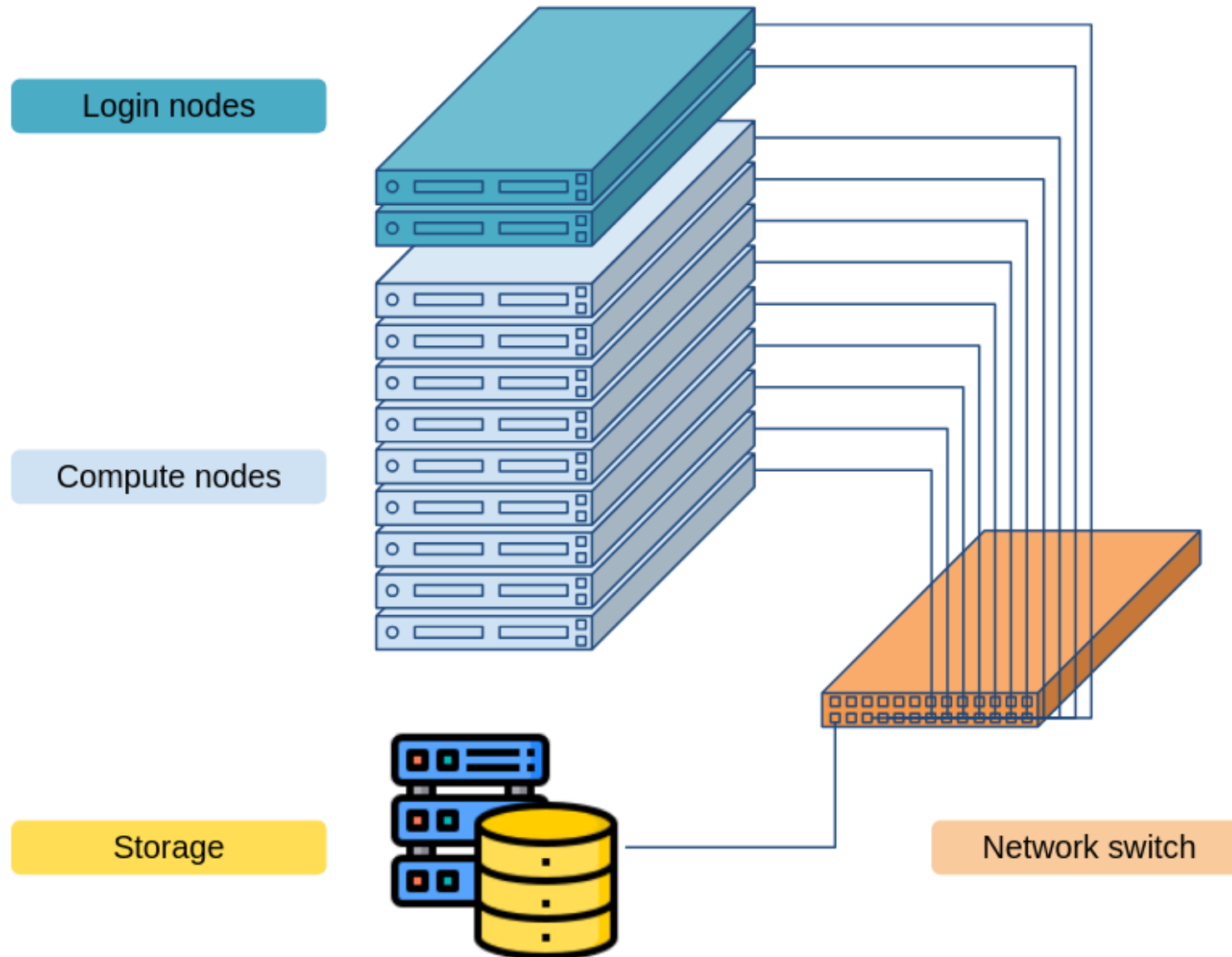




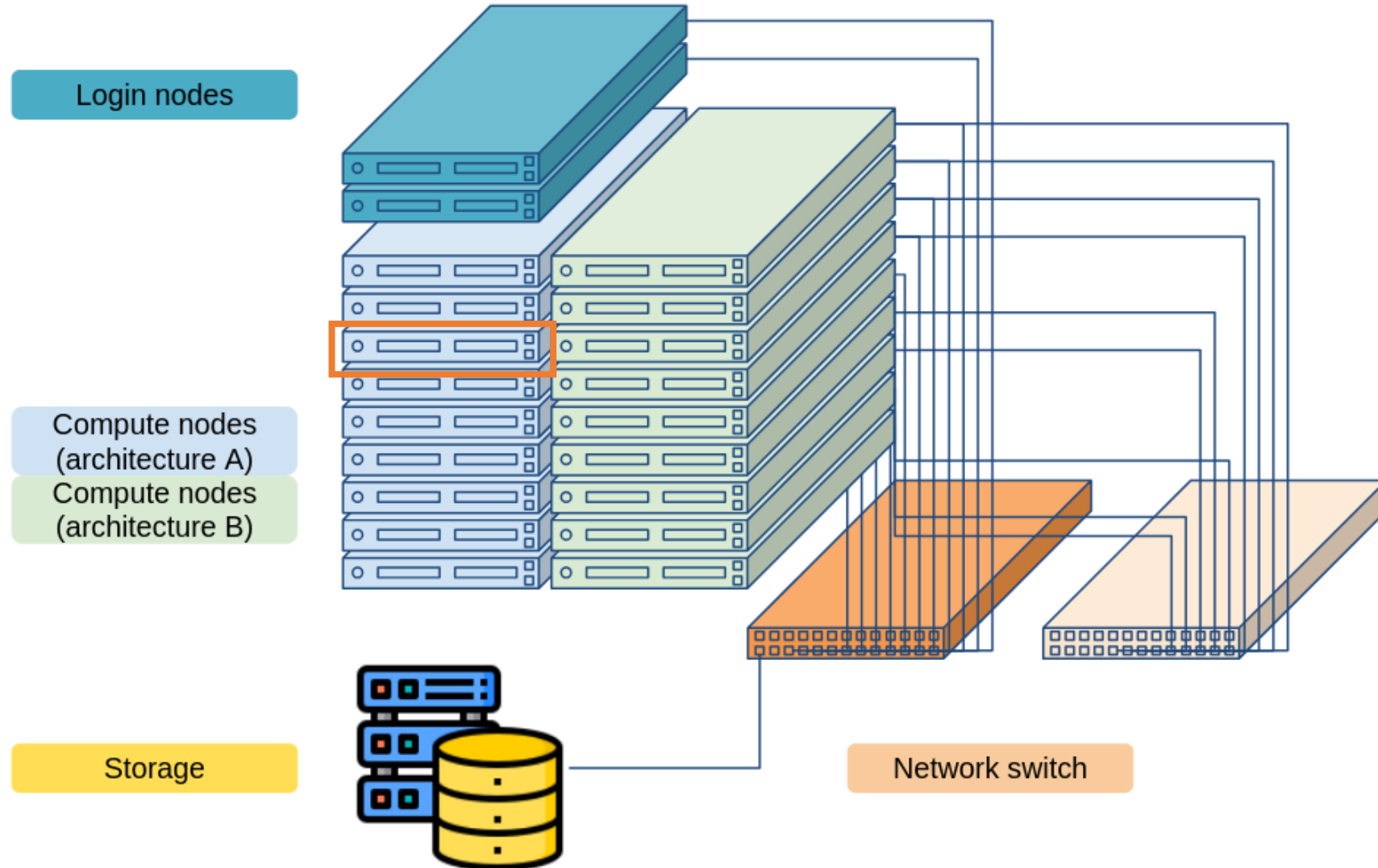


Anatomy of a supercomputer

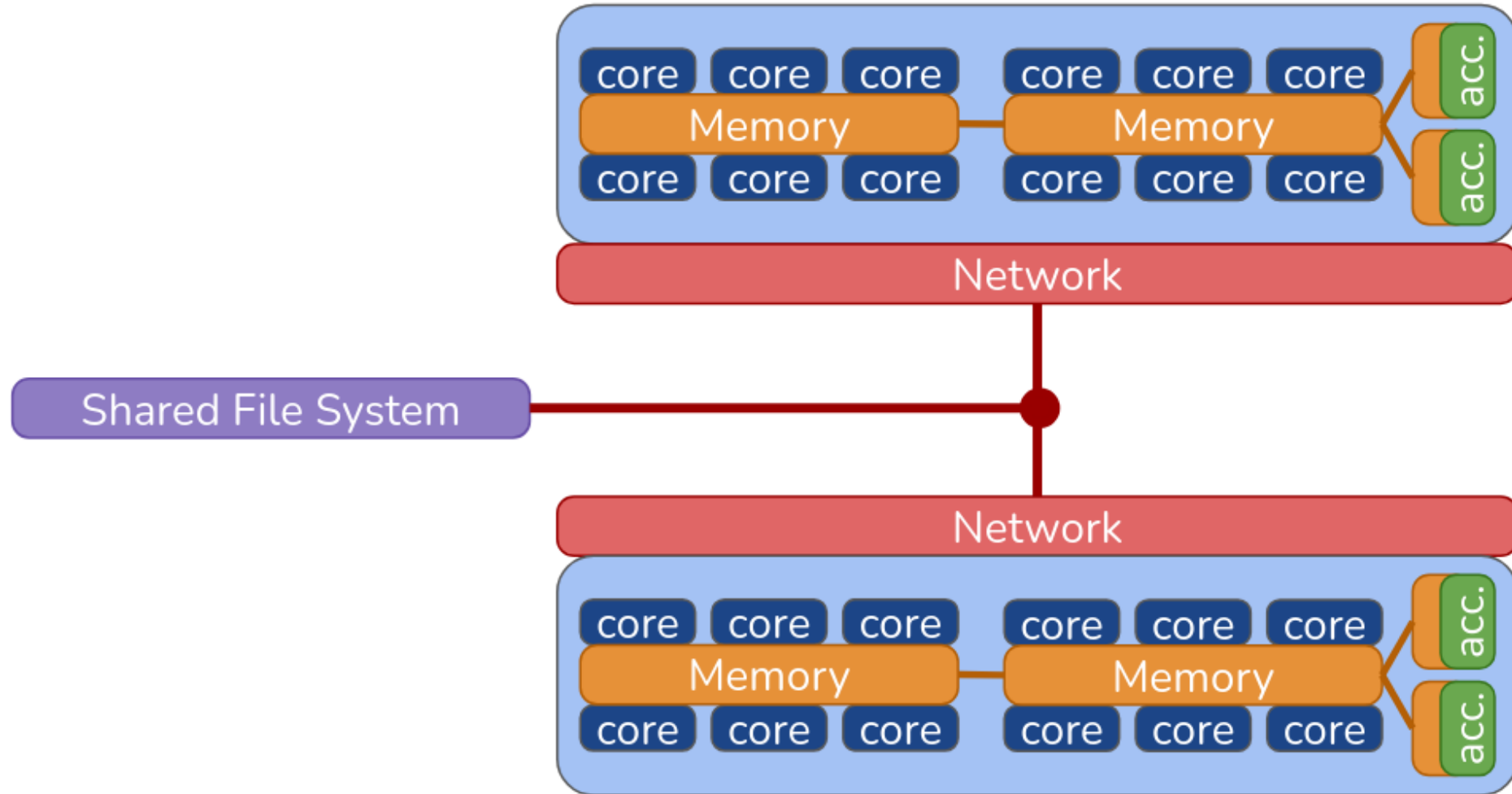
Architecture of an HPC cluster



Some clusters have 2 partitions!



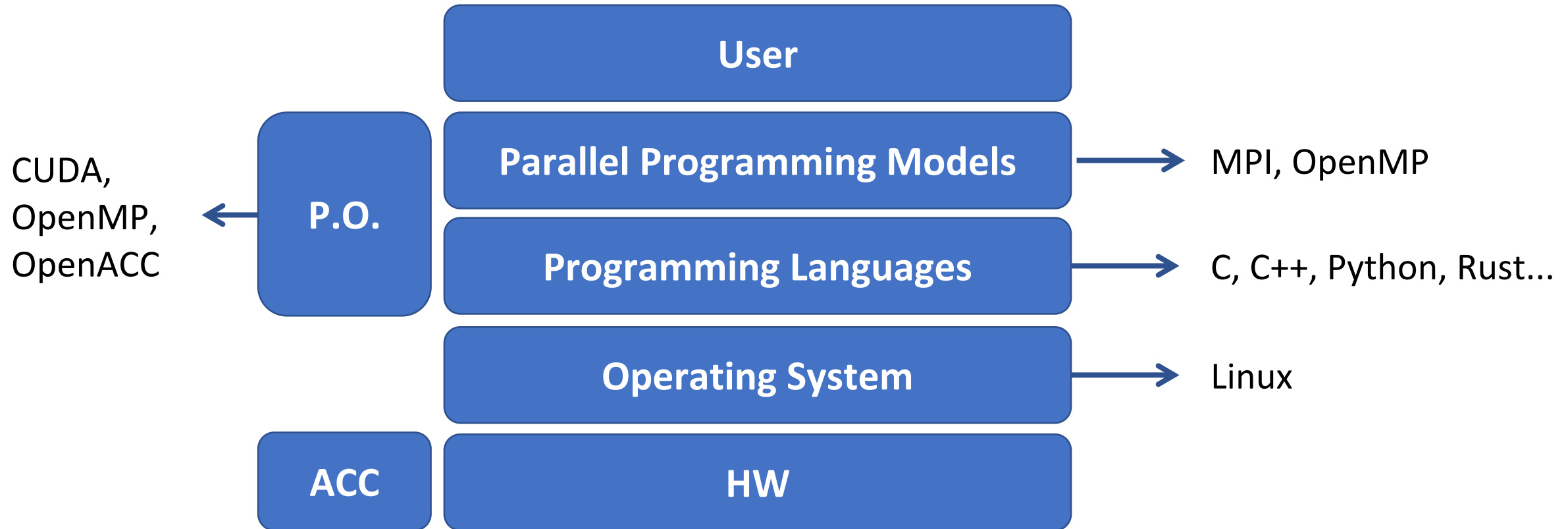
Inside the cluster





How to program a supercomputer?

How to program a supercomputer



Graphic Processing Unit (also known as Accelerators)

Main differences with CPU:

- Designed for massive parallel computation
- Good for matrix operations, simulations, ML, etc
- GPUs need an offload mechanism to offload work to accelerator and usually data too.

In HPC systems, **CPU and GPU work together**: CPU orchestrates and GPU computes efficiently.

GPUs fall out of scope of this tutorial

Parallel programming models allow to **use the computational resources in parallel** to solve a single problem.

- Shared memory: Computational resources “see” the same data.
 - Most popular: OpenMP
- Distributed memory: Computational resources use network to share data.
 - Most popular: MPI



Distributed memory programming

MPI

What is MPI?



MPI: Message Passing Interface

- standardizes what the MPI implementation provides and what the application can and cannot expect
- each system can implement it differently as long as the semantics match

MPI is not...

- a language or compiler specification
- a specific implementation or a product

Introduction to MPI



- Distributed-memory programming model
- Process level parallelism
- Explicit process communication
- Programming via MPI calls
- Code is written with MPI execution in mind from the very start

```
#include <mpi.h>
#include <stdio.h>

int main(int argc, char* argv[]) {
    int nranks, myrank;

    MPI_Init(&argc, &argv);

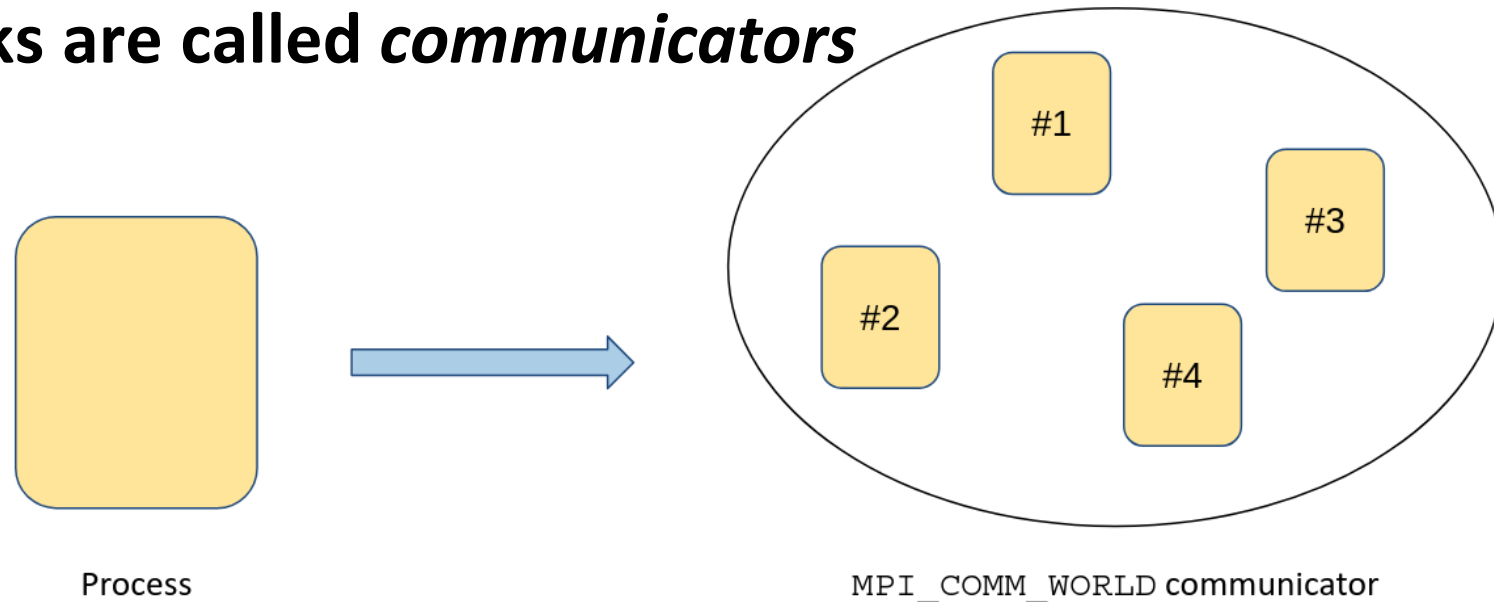
    MPI_Comm_size(MPI_COMM_WORLD, &nranks);
    MPI_Comm_rank(MPI_COMM_WORLD, &myrank);

    if (myrank == 0)
    {
        printf("Master rank prints\n");
    }

    MPI_Finalize();
}
```

Process $\simeq$ Rank

- Program in execution
- Has a Process Identifier (PID)
- Has its own memory space
- **Has a rank id**
- **Groups of ranks are called *communicators***



MPI Process communication



Point-to-point

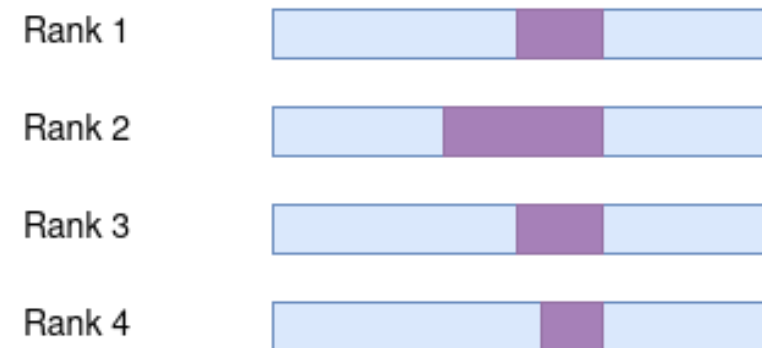
- Pair of processes
- MPI_Send(...)
- MPI_Recv(...)



□ Useful computation □ MPI_Send □ MPI_Recv

Collective

- All processes in a communicator
- MPI_Barrier(...)
- MPI_Allreduce(...)



□ Useful computation □ MPI_Allreduce

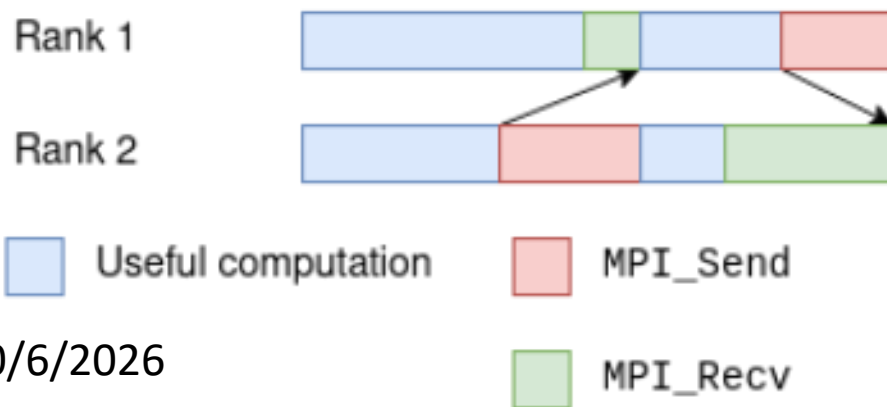
MPI Process communication



Blocking

- Wait until communication has finished

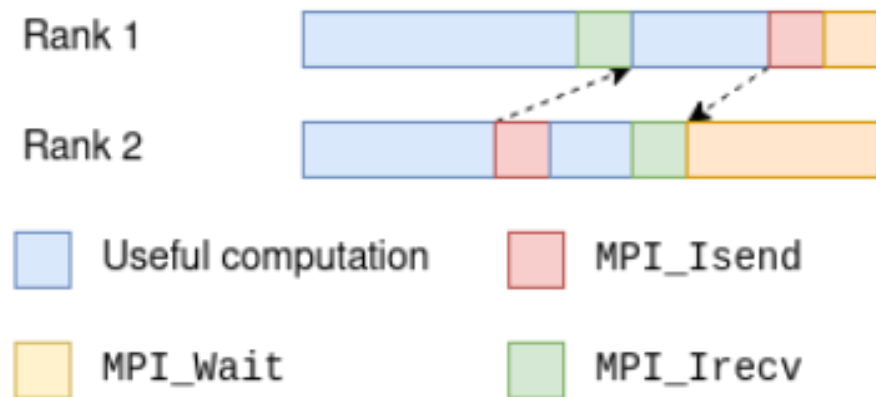
- `MPI_Send(...)`
- `MPI_Recv(...)`



Non-blocking

- Continue until you really need the data

- `MPI_Isend(...)`
- `MPI_IRecv(...)`
- `MPI_Wait(...)`





Shared memory programming

OpenMP

The OpenMP specification



OpenMP provides an **Application Programming Interface (API)** composed of:

➤ Runtime routines

➤ Compiler directives

➤ Environment variables

```
int main(void) {  
    omp_set_num_threads(NUM_THREADS);  
    #pragma omp parallel for  
    for (int i=LB; i<UB; ++i) {  
        a[i] = VALUE;  
    }  
    return SUCCESS;  
}
```

```
sroyuela@royubsc:~/tmp$ clang++ -fopenmp OMP_PROC_BIND=close -o test test.cpp
```



Introduction to OpenMP

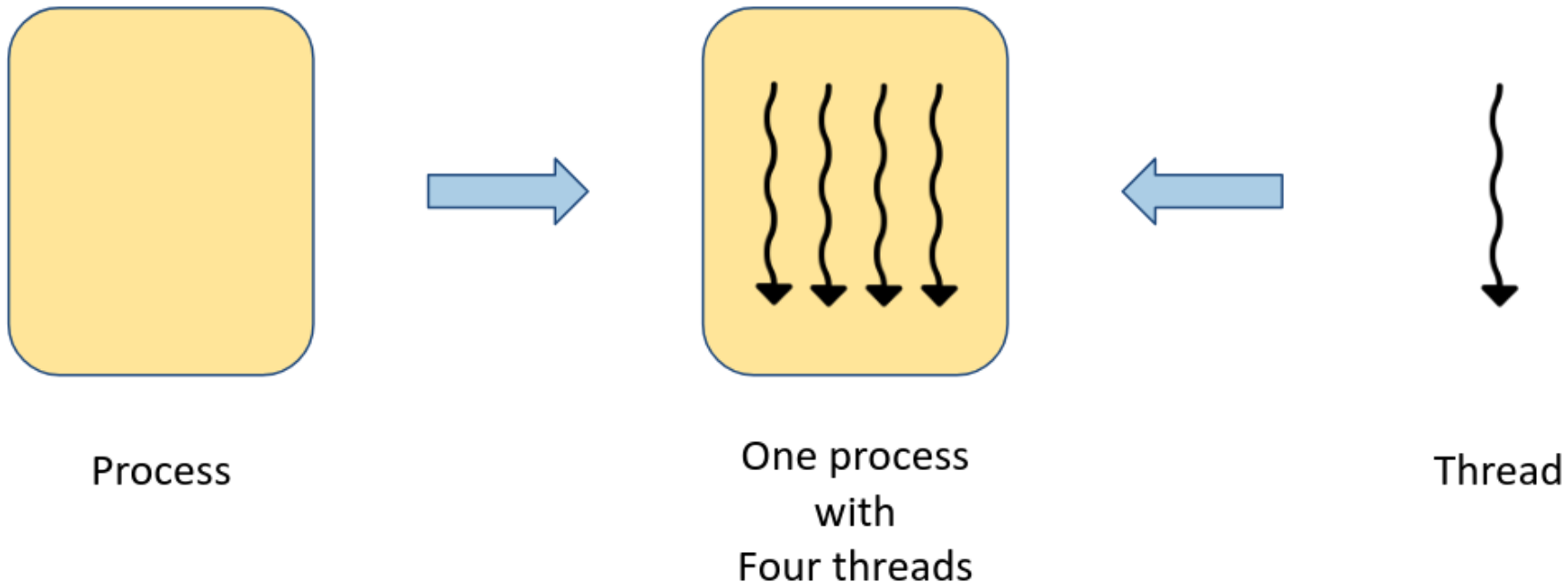


Process

- Program in execution
- Has a Process Identifier (PID)
- Has its own memory space

Thread

- Sequence of instructions
- Part of a thread pool
- Shares memory with other threads of same process

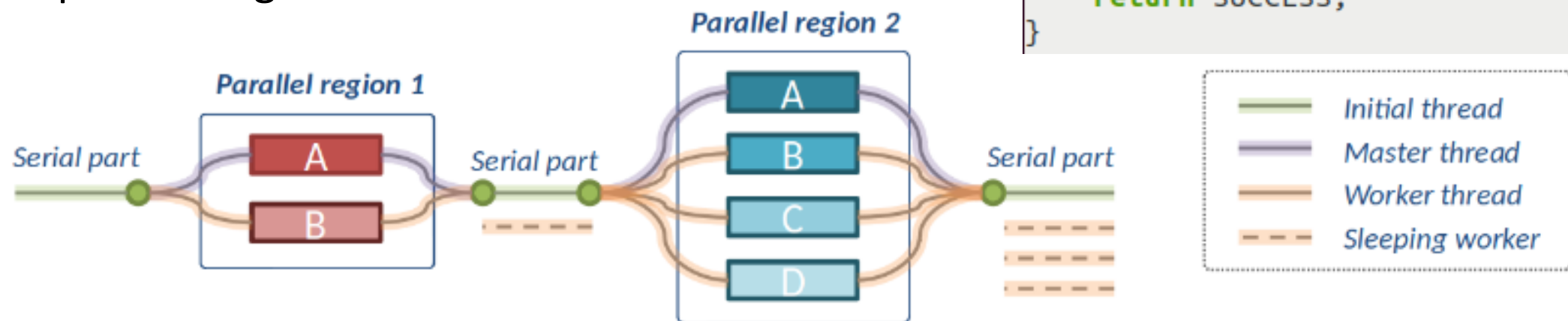


Fork-join execution model



- OpenMP programs start sequentially with a unique **initial thread**
- **Worker threads** are spawned in parallel regions
 - The thread encountering a parallel region becomes the **master thread**
 - Together with the workers, they form a **team of threads**
 - Workers are destroyed or put to sleep between parallel regions

```
int main(void) {  
    omp_set_num_threads(NUM_THREADS);  
  
    #pragma omp parallel for  
    for (int i=LB; i<UB; ++i) {  
        a[i] = VALUE;  
    }  
  
    return SUCCESS;  
}
```



- **Race condition:** It happens when two or more threads or processes try to access and change the same shared data at the same time
- Mechanisms
 - **Atomic Operations:** Hardware-level commands that perform a read-modify-write action without interference from other threads or cores
 - **Serial Execution:** Enforcing an ordered sequence where tasks wait for the previous task to finish completely
 - **Dependencies:** Specify the order in which the updates should happen



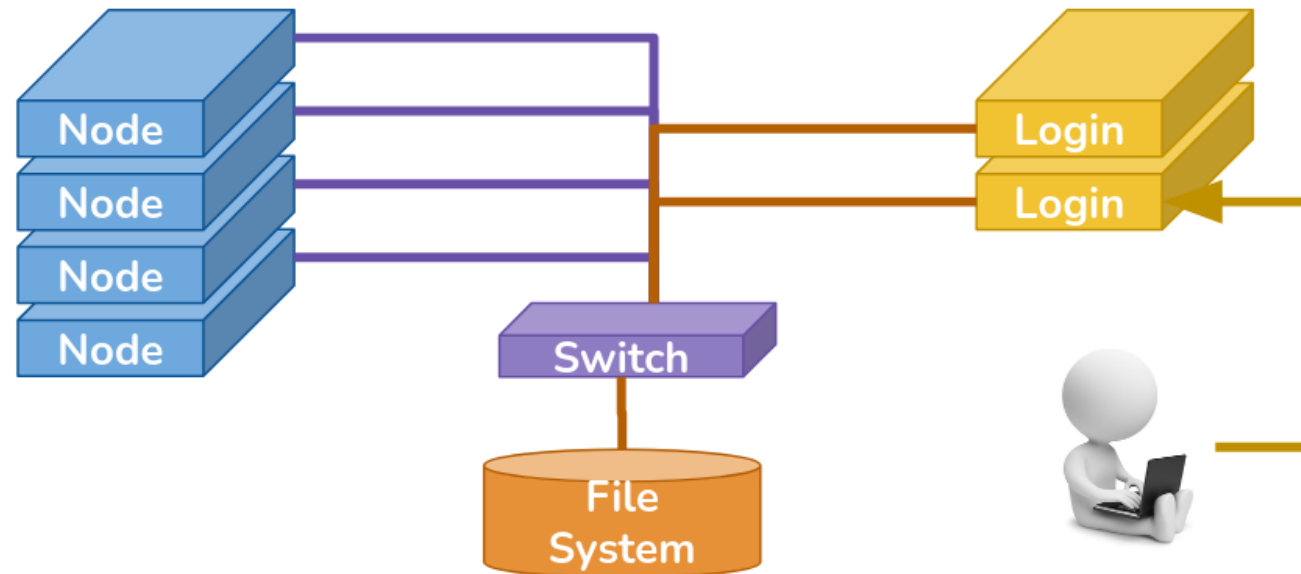
How to use a supercomputer

The batch scheduler

Logging in into the supercomputer



- User logs in to login node.
- Login nodes can be used to edit files, compile... should not be used for running.
- Access to compute nodes achieved through batch scheduler





Batch scheduler

What's a "Batch Scheduler"?



... or a workload manager or a job scheduler is like the steward for your cluster

May I have
your ticket?



What's a "Batch Scheduler"?



... or a workload manager or a job scheduler is like the steward for your cluster



Here it is:
submit.sh

May I have
your ticket?



What's a "Batch Scheduler"?



... or a workload manager or a job scheduler is like the steward for your cluster



Here it is:
submit.sh

May I have
your ticket?



Thanks!
Distribute
yourself in nodes
045-050

Most popular one

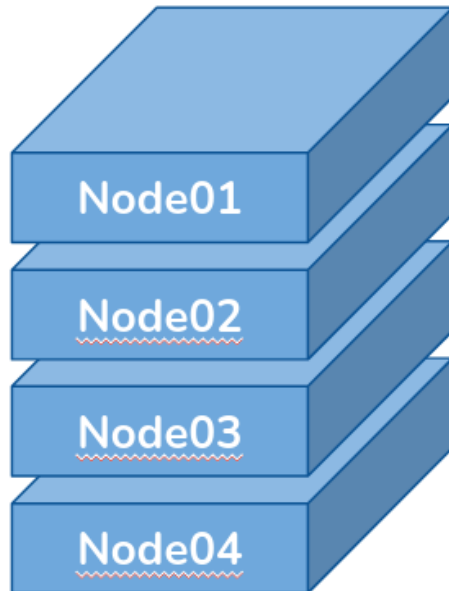


Simple Linux Utility for Resource Management

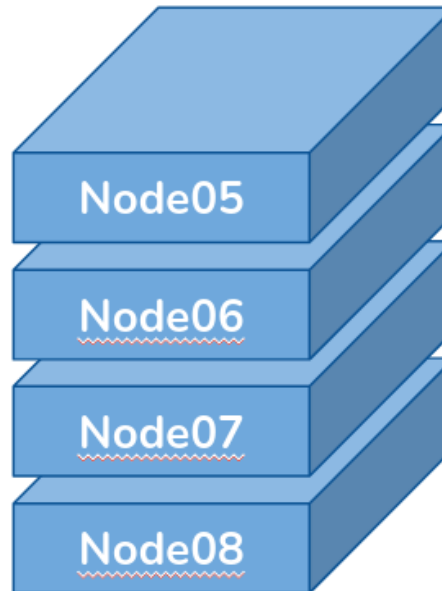
Resources and partitions



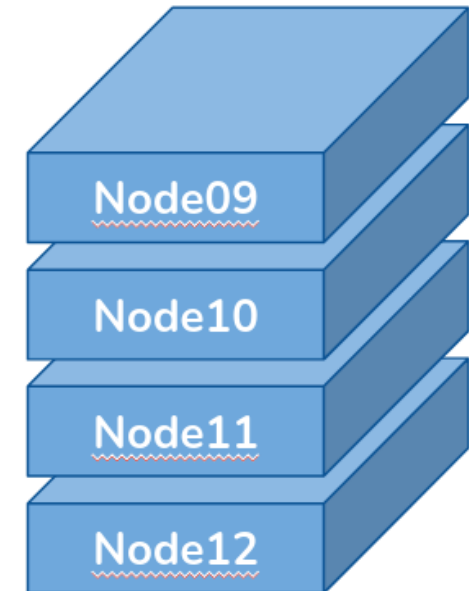
Partitions are similar to parking lots that I want to book for a given amount of time:



Fast jobs,
Low resources



Interactive apps
GPUs



Compute
intensive jobs

- A supercomputer is a big, fast and powerful computer.
- The batch scheduler manages jobs' access to resources
- Distributed memory means that computational resources use the network to share data
 - We use MPI: Needs explicit communication between processes
- Shared memory means that computational resources “see” the same data
 - We use OpenMP: Needs mechanisms to avoid race conditions when accessing shared data
- GPUs need an offloading mechanisms
 - They are programmed different than CPUs



Introduction to Performance Analysis

Katharina Haus (JSC)

HORIZON-EUROHPC-JU-2023-COE



EuroHPC
Joint Undertaking

Grant Agreement No 101143931

1 January 2024– 31 December 2026

Why Performance Analysis Matters



What do we mean by “Performance”

- Two complementary questions:
 - How fast is it
 - How efficiently does it use the resources it's given?
- Performance analysis is the discipline of measuring, understanding, and improving both

Performance Analysis

- Helps you decide: is my code ready to scale up?
- Helps you find bottlenecks (compute, memory, communication, I/O)
- Helps you make the case for more allocation/resources
- Helps compare algorithms, implementations, or hardware fairly

Successful engineering is a combination of

- Careful setting of various tuning parameters
- The right algorithms and libraries
- Compiler flags and directives
- ...
- Thinking



Measurement is better than guessing

- To determine performance bottlenecks
- To compare alternatives
- To validate tuning decisions and optimizations
 - ➔ After each step

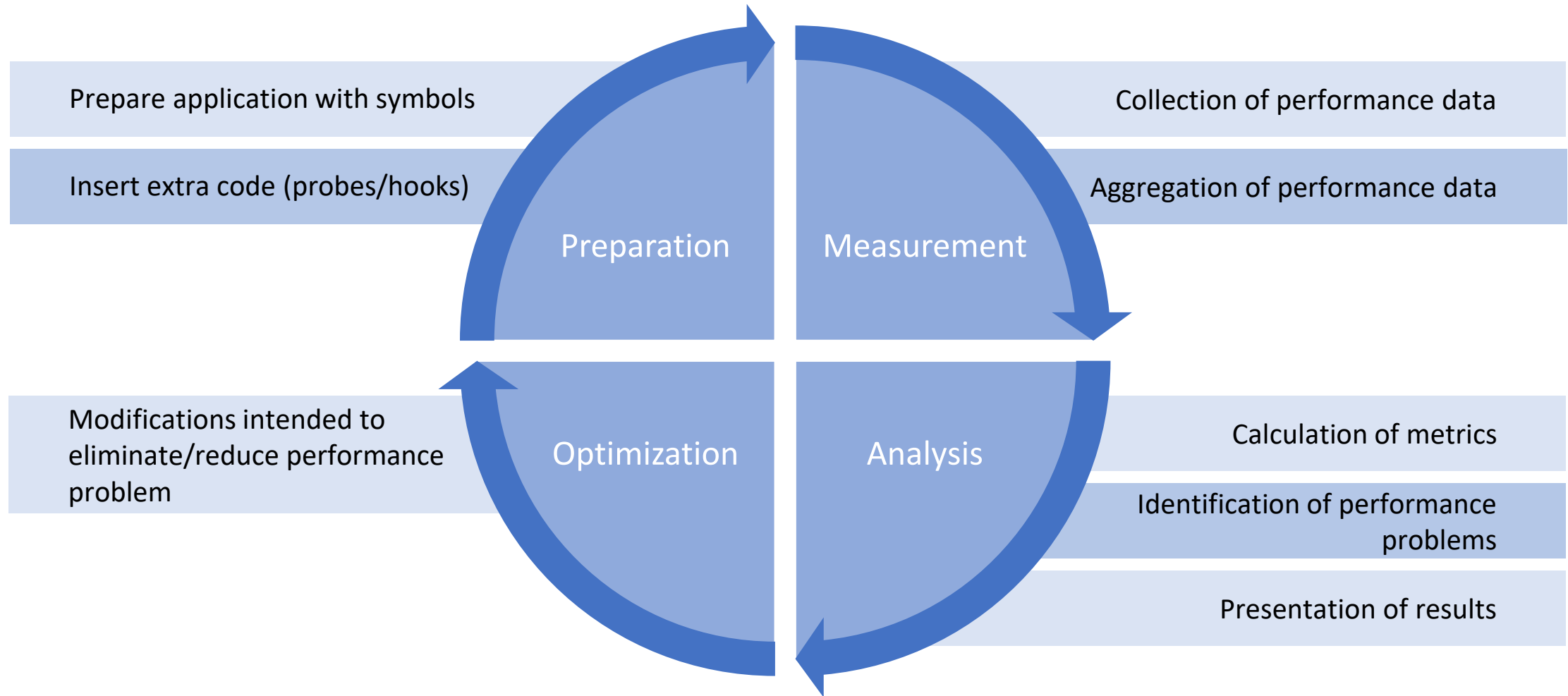


The 80/20 rule



- Programs typically spend 80% of their time in 20% of the code
- Programmers typically spend 20% of their effort to get 80% of the total speedup possible for the application
 - ➔ Know when to stop!
- Don't optimize what does not matter
 - ➔ Make the common case fast!

Performance Engineering Workflow



Core Performance Metrics



Execution time

FLOP/s (computational rate)

Speedup

Scalability (strong & weak)

Efficiency

Load Balance

- Wall-clock time
 - what the user actually experiences (most important in practice)
 - Includes waiting time: I/O, memory, other system activities
 - In time-sharing environments also the time consumed by other applications
- CPU time
 - Time spent by the CPU to execute the application
 - Does not include time the program was context-switched out
 - Problem: Does not include inherent waiting time (e.g., I/O)
 - Problem: Portability? What is user, what is system time?
- Problem: Execution time is non-deterministic
 - Use mean or minimum of several runs

Computational Rate: FLOP/s



- FLOP/s = floating-point operations per second
- Peak performance (theoretical max of the hardware) vs achieved/sustained performance
- % of peak as a sanity metric (real codes rarely exceed few tens of percent of peak)

Speedup



How much performance gain is achieved by parallelizing a given application over a sequential implementation

→ Captures the relative benefit of solving a problem in parallel

Formula: $S(p) = \frac{T(1)}{T(p)}$

Ideal (linear) speedup: $S(p) = p$

$T(1)$ = execution time on 1 processor

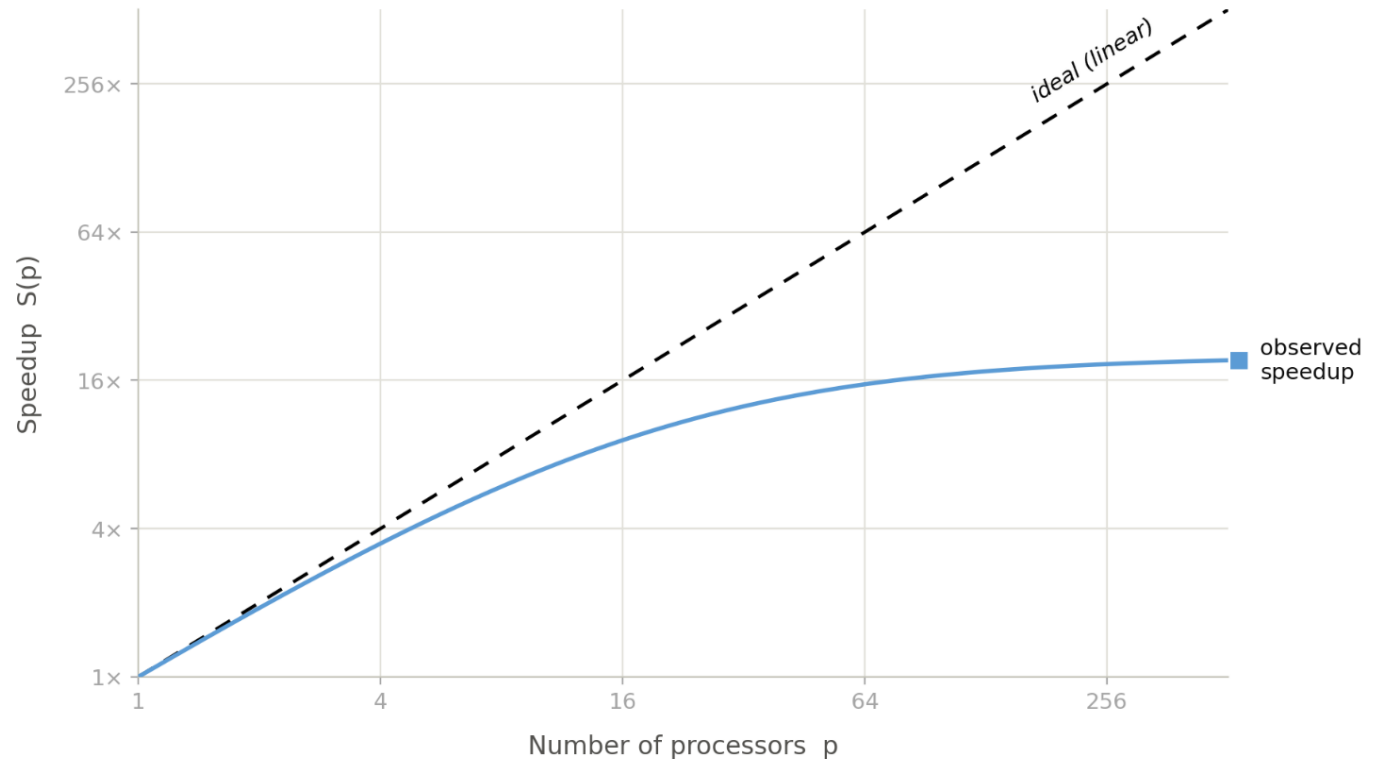
$T(p)$ = execution time on p processors

Speedup: The Reality



Why speedup falls short of linear:

- serial fraction
- communication overhead
- load imbalance
- synchronization



Amdahl's Law

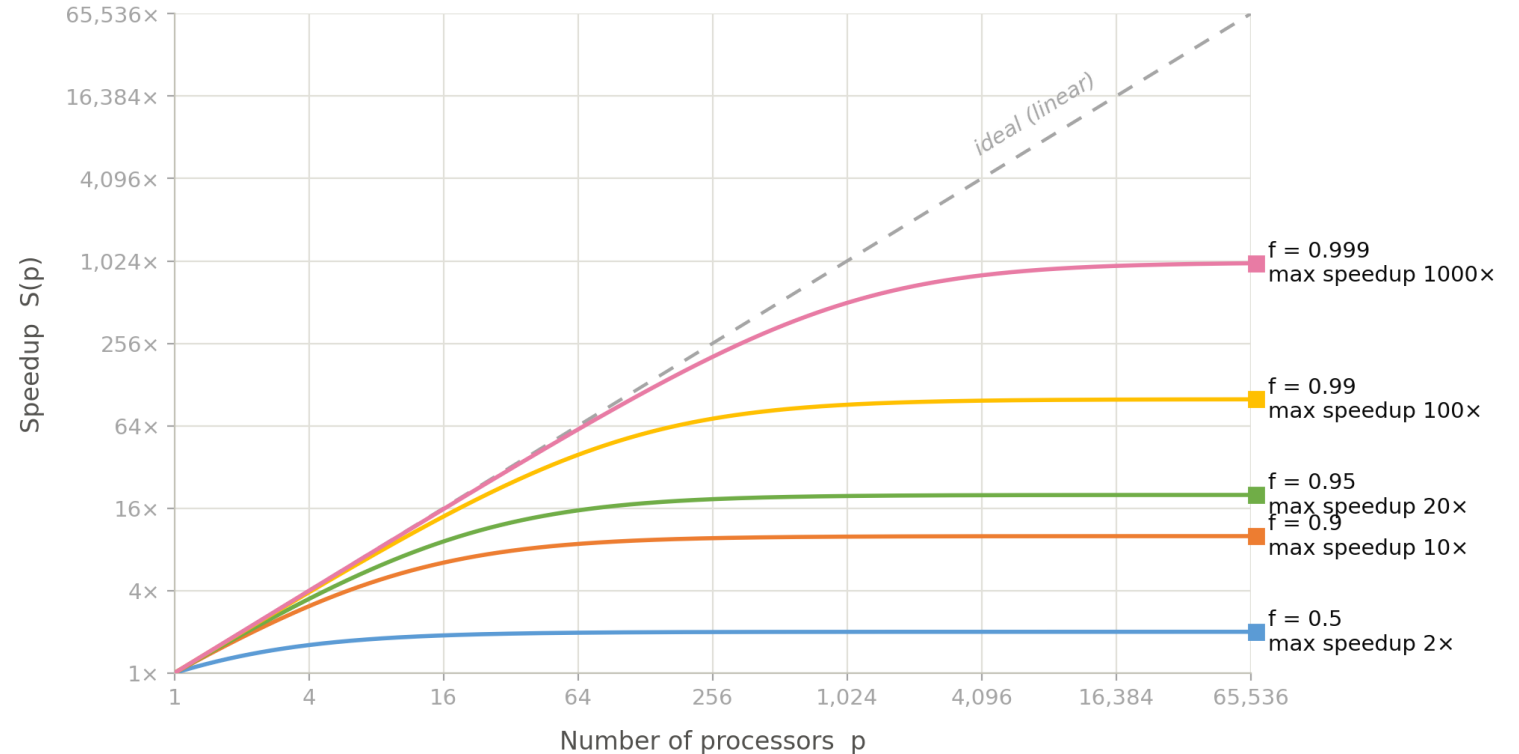


$$\text{Formula: } S(p) = \frac{1}{1-f + \frac{f}{p}}$$

f = parallelizable fraction,
 p = number of processors

$$\text{Limit: } \lim_{p \rightarrow \infty} S(p) = \frac{1}{1-f}$$

The serial fraction sets the
performance limit



! Strong Scaling is fundamentally limited by the part of the code that can't be parallelized

Strong Scaling vs Weak Scaling



Strong Scaling

- *How much faster can this exact problem be solved?*
- Fixed total problem size, increasing number of processors
- Directly governed by Amdahl's Law

Weak Scaling

- *Can a bigger problem be solved in the same time with more processors?*

- Governed by Gustafson's Law:

$$S(p) = p - (1 - f)(p - 1) = f \cdot p + (1 - f)$$

- Scaling looks more favorable: $S(p) \xrightarrow{p \rightarrow \infty} \infty$

f = parallelizable fraction,
 p = number of processors

Efficiency



- Compute time of process i : $c_i = \sum_{j=1}^3 c_i^j$
- Total compute time: $c = \sum_{i=1}^p c_i$



$$E(p) = \frac{S(p)}{p} = \frac{T(1)}{p \cdot T(p)} = \frac{c}{p \cdot T} = \frac{\text{blue}}{\text{blue} + \text{orange}}$$



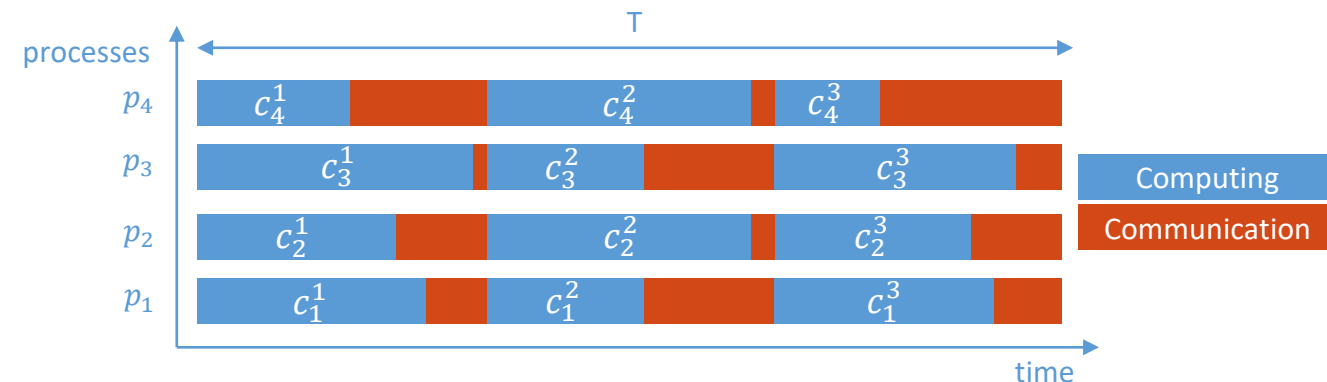
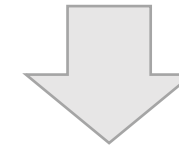
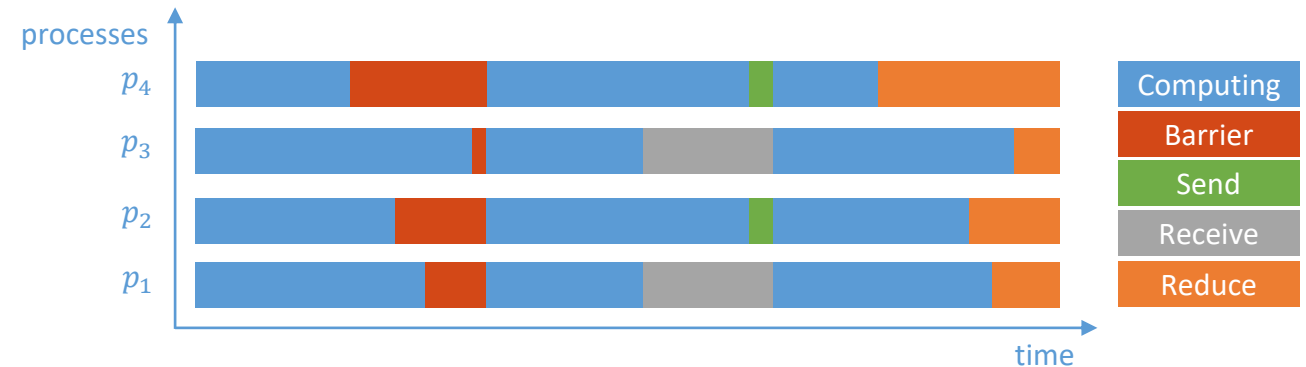
Ratio between computation time and consumed CPU time



Quantifies: the extent to which all resources in the system are kept active doing useful work



Interpretation: A low value indicates that a low fraction of the time consumed is used to do useful computation



Load Balance



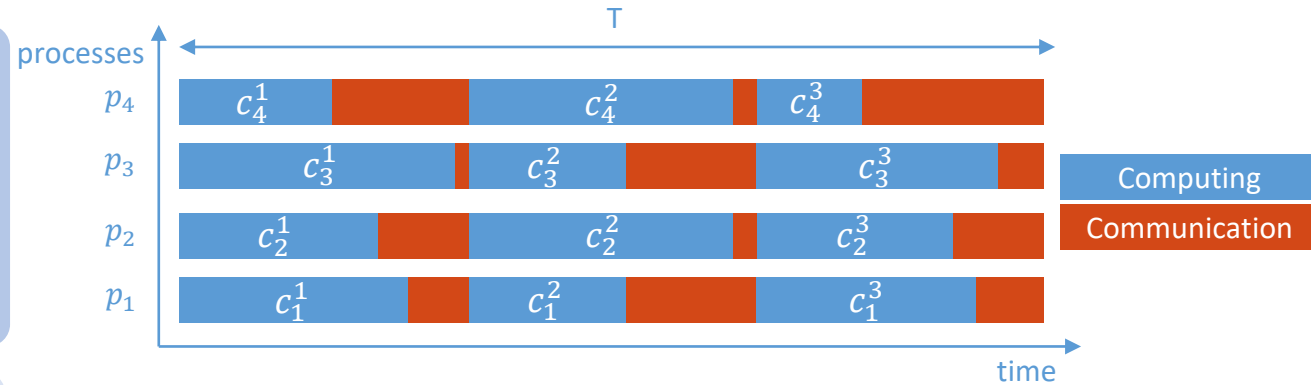
Ratio between time used to do useful computation and the useful computation of the most loaded process multiplied by the number of processes:

$$LB = \frac{c}{\max_{i=1}^p (c_i) \cdot p} = \frac{\text{blue}}{\text{blue} + \text{orange}}$$

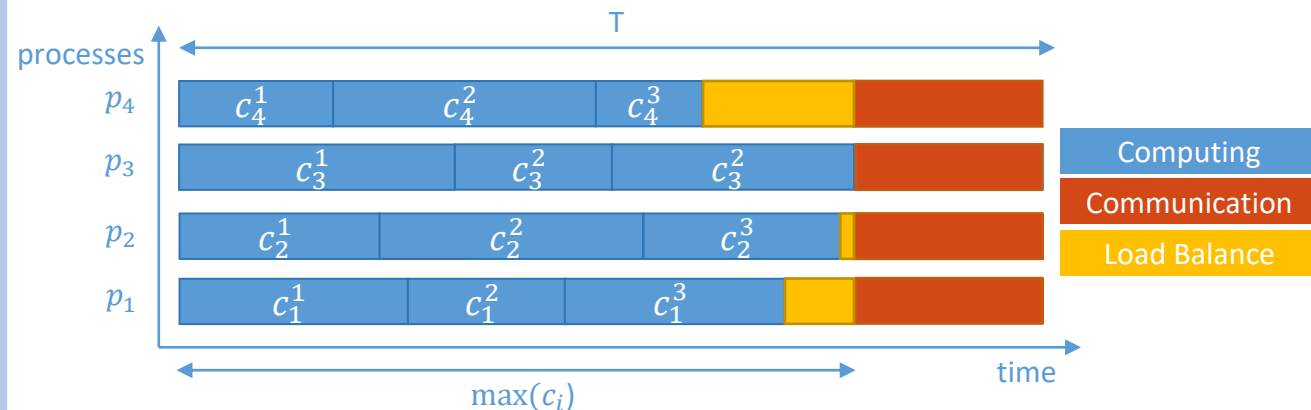
Quantifies: The efficiency loss due to the global distribution of work among processes.

Interpretation: A low value in this metric indicates that more highly loaded processes keep other processes idle for a significant amount of time

- A single highly loaded process will make this metric report a low value
- A single low loaded process won't have an effect on this metric



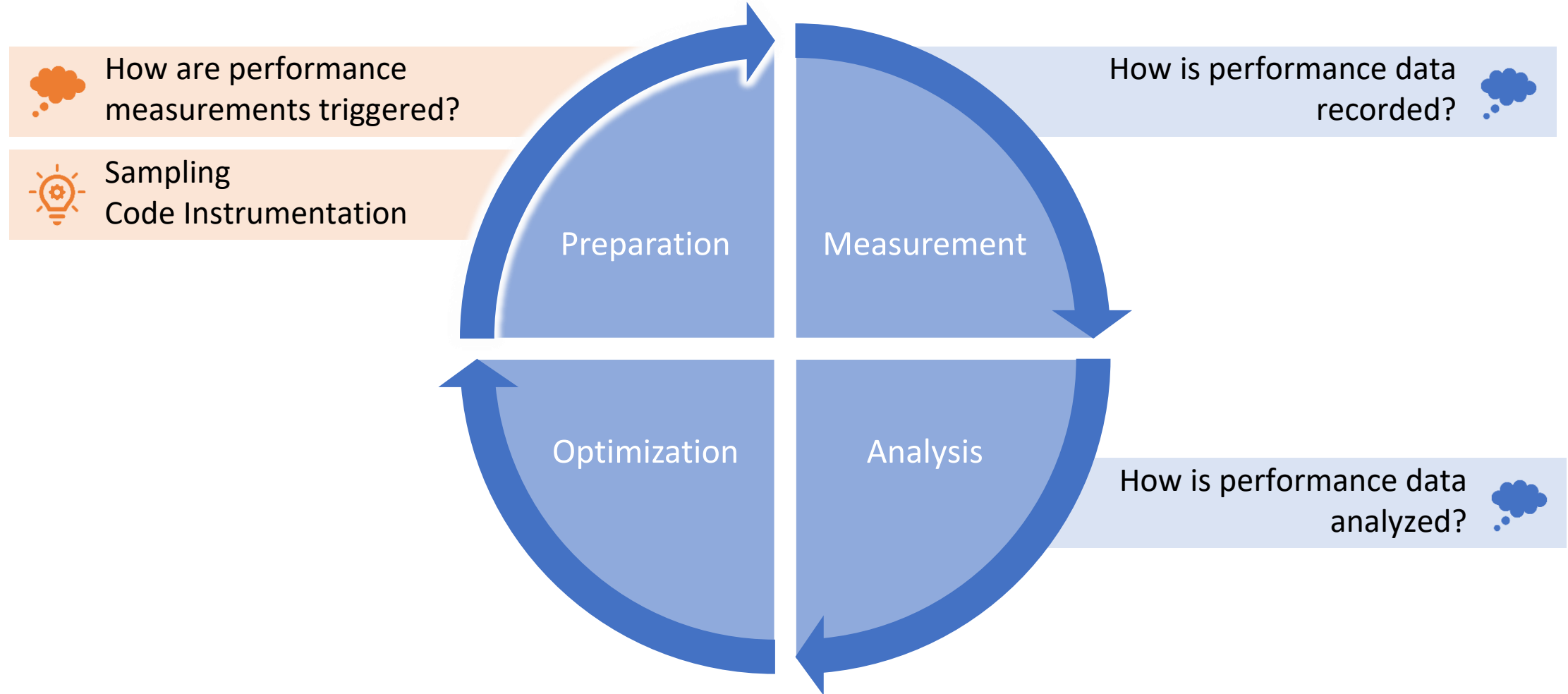
“Collapse”
computations



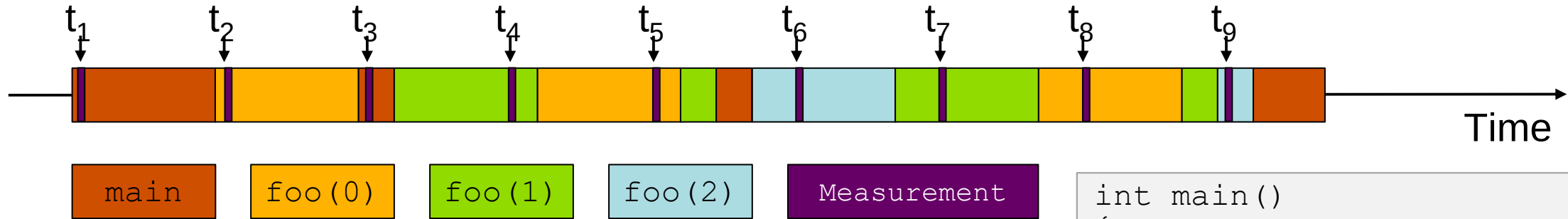
What can be measured

- A **count** of how often an event occurs
 - E.g., the number of MPI point-to-point messages sent
- The **duration** of some interval
 - E.g., the time spent in these send calls
- The **size** of some parameter
 - E.g., the number of bytes transmitted by these calls
- Calculate derived metrics

Classification of measurement techniques



Sampling



- Running program is periodically interrupted to take measurement
 - Timer interrupt, OS signal, or HWC overflow
 - Service routine examines return-address stack
 - Addresses are mapped to routines using symbol table information
- Statistical inference of program behavior
 - Not very detailed information on highly volatile metrics
 - Requires long-running applications
- Works with unmodified executables

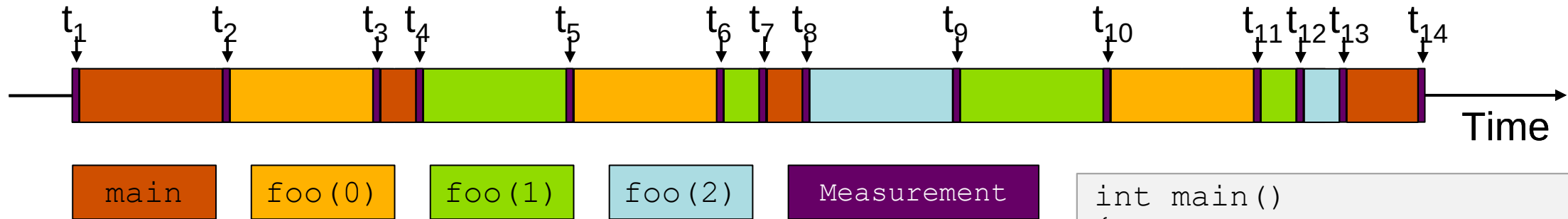
```
int main()
{
    int i;

    for (i=0; i < 3; i++)
        foo(i);

    return 0;
}

void foo(int i)
{
    if (i > 0)
        foo(i - 1);
}
```

Instrumentation



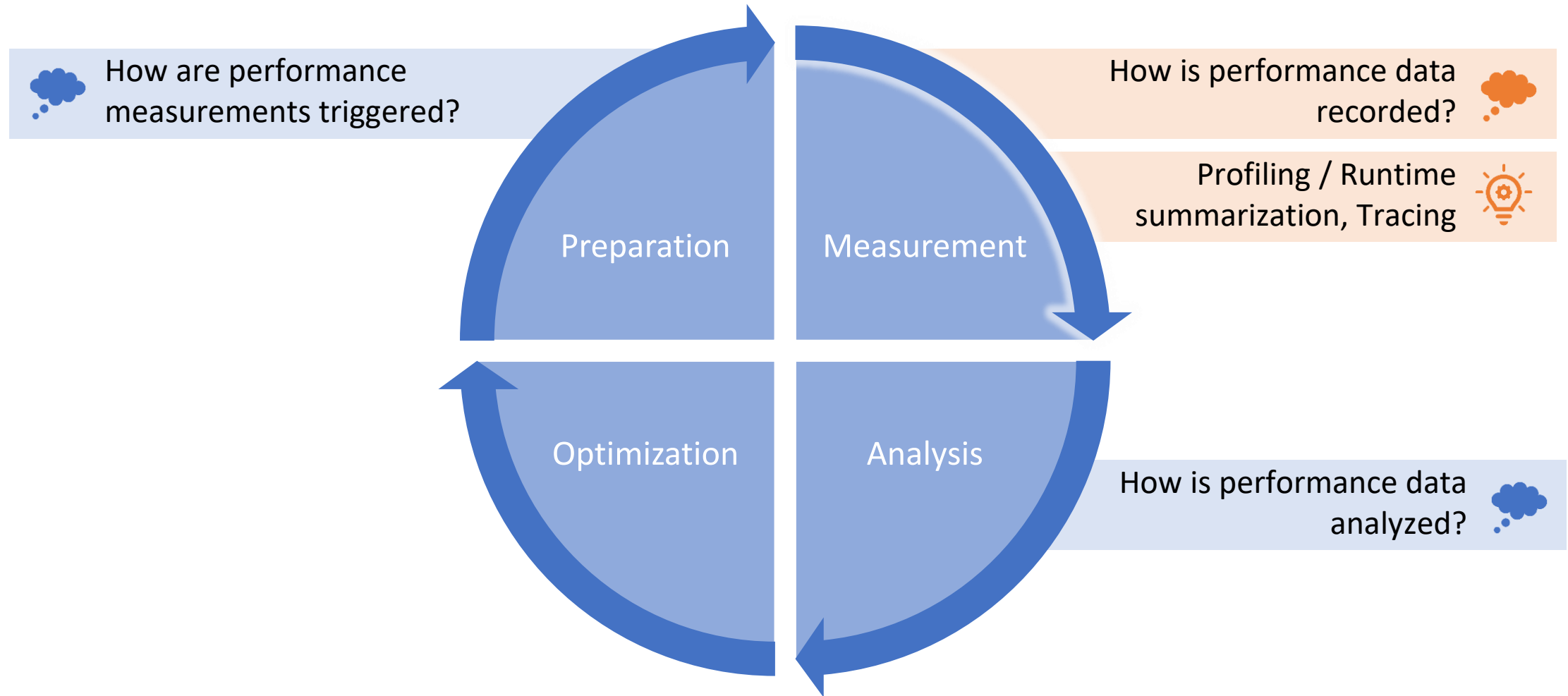
- Measurement code is inserted such that every event of interest is captured directly
 - Can be done in various ways
- Advantage:
 - Much more detailed information
- Disadvantage:
 - Processing of source-code/executable necessary
 - Large relative overheads for small functions

```
int main()
{
    int i;
    Enter("main");
    for (i=0; i < 3; i++)
        foo(i);
    Leave("main");
    return 0;
}

void foo(int i)
{
    Enter("foo");
    if (i > 0)
        foo(i - 1);
    Leave("foo");
}
```

- Accuracy
 - Intrusion overhead
 - Measurement itself needs time and thus lowers performance
 - Perturbation
 - Measurement alters program behavior
 - E.g., memory access pattern
 - Accuracy of timers & counters
- Granularity
 - How many measurements?
 - How much information / processing during each measurement?
- *Tradeoff: Accuracy vs. Expressiveness of data*

Classification of measurement techniques





- Recording of aggregated information
 - Total, maximum, minimum, ...
- For measurements
 - Time
 - Counts
 - Function calls
 - Bytes transferred
 - Hardware counters
- Over program and system entities
 - Functions, call sites, basic blocks, loops, ...
 - Processes, threads
- *Profile = summarization of events over execution interval*

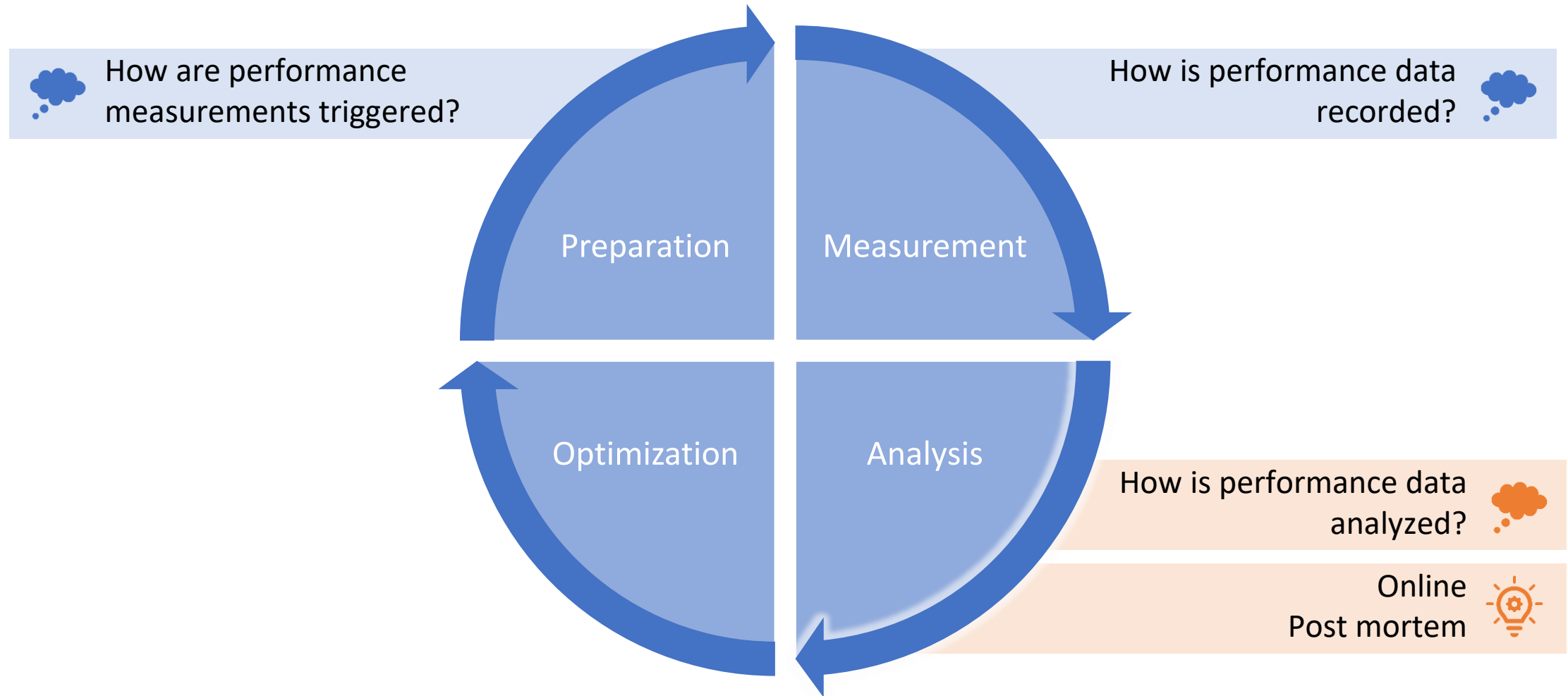
- Recording detailed information about significant points (events) during execution of the program
 - Enter/Leave of a region (function, loop, ...)
 - Send/Receive a message, ...
- Save information in event record
 - Timestamp, location, event type
 - Plus event-specific information (e.g., communicator, sender/receiver, ...)
- Abstract execution model on level of defined events
- *Event trace = Chronologically ordered sequence of event records*

Tracing Pros & Cons



- Tracing advantages
 - Event traces preserve the **temporal** and **spatial** relationships among individual event
 - Allows reconstruction of **dynamic** application behavior on any required level of abstraction
 - Most general measurement technique
 - Profile data can be reconstructed from event traces
- Disadvantages
 - Traces can very quickly become extremely large
 - Writing events to file at runtime may cause perturbation

Classification of Measurement Techniques



- Performance data is processed during measurement run
 - Process-local profile aggregation
 - Requires formalized knowledge about performance bottlenecks
 - More sophisticated inter-process analysis using
 - “Piggyback” messages
 - Hierarchical network of analysis agents
- Online analysis often involves application steering to interrupt and re-configure the measurement

Post-mortem Analysis



- Performance data is stored at end of measurement run
- Data analysis is performed afterwards
 - Automatic search for bottlenecks
 - Visual trace analysis
 - Calculation of statistics

Wait State & Critical Path

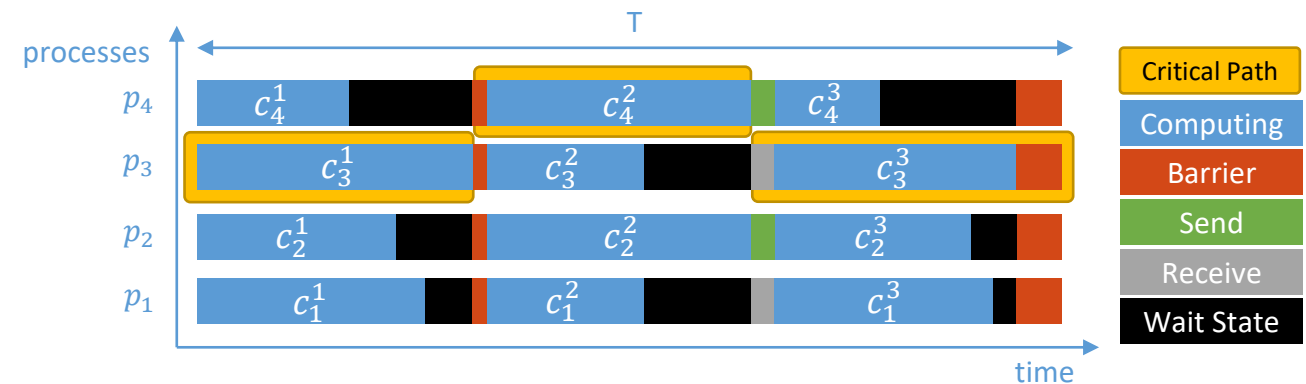
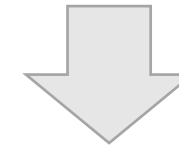


Wait State:

- Time a processor sits idle waiting for an action on a remote processor
- E.g., process blocks in receive call waiting for its peer process to call the send call

Critical Path:

- Shows call paths and processes/threads that are responsible for the program's wall-clock runtime
- Identifies good optimization candidates and parallelization bottlenecks
- Does not contain wait states



Typical Performance Analysis Procedure



- Do I have a performance problem at all?
 - Time / speedup / scalability measurements
- **What** is the key bottleneck (computation / communication)?
 - MPI / OpenMP / flat profiling
- **Where** is the key bottleneck?
 - Call-path profiling, detailed basic block profiling
- **Why** is it there?
 - Hardware counter analysis, trace selected parts to keep trace size manageable
- Does the code have scalability problems?
 - Load imbalance analysis, compare profiles at various sizes function-by-function
- No single solution is sufficient



Performance Analysis in Action

Anke Visser (JSC)

HORIZON-EUROHPC-JU-2023-COE

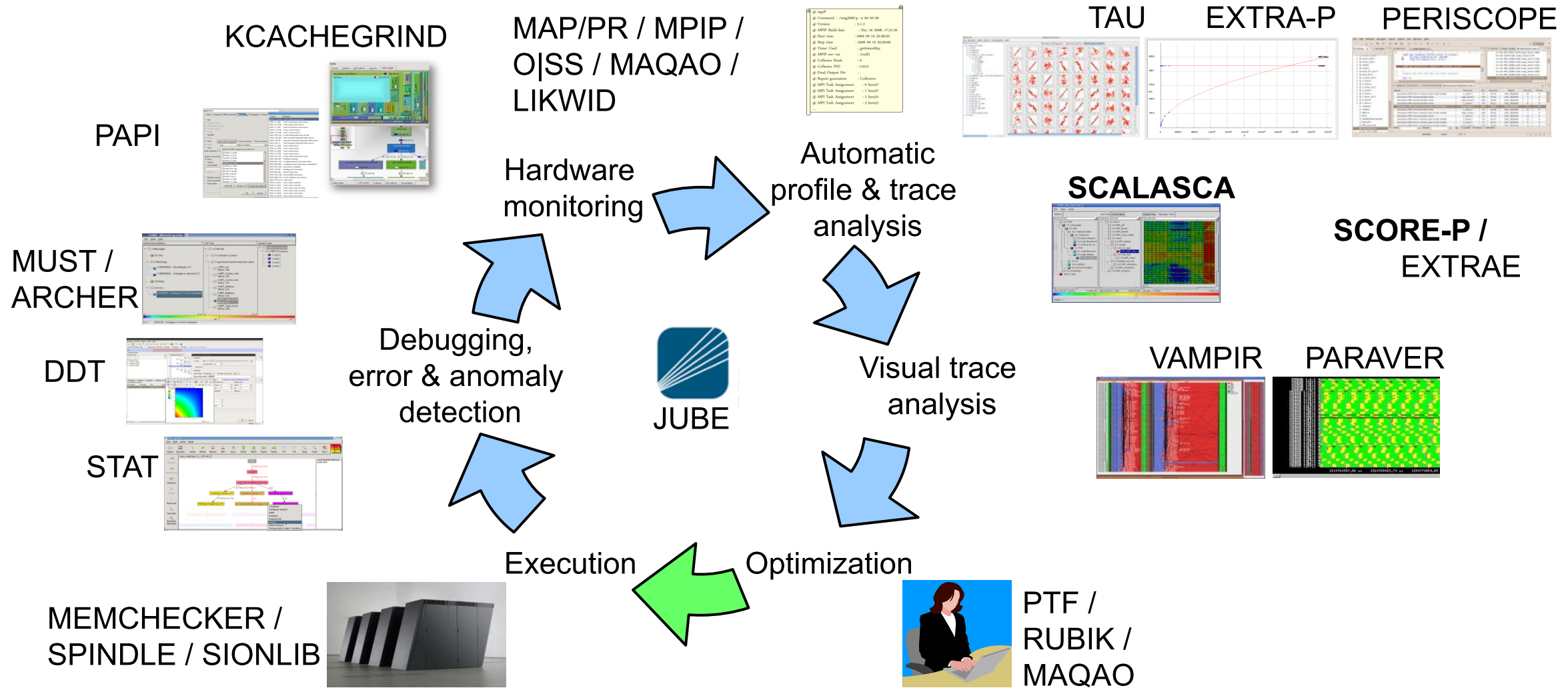


EuroHPC
Joint Undertaking

Grant Agreement No 101143931

1 January 2024– 31 December 2026

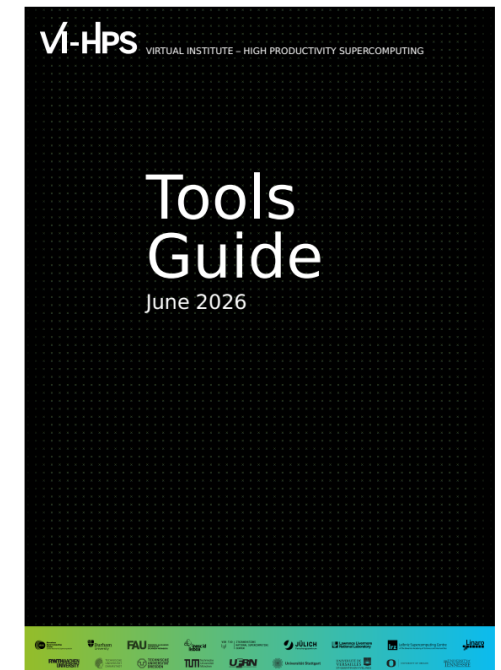
HPC Tools





- **Goal:** Improve the quality and accelerate the development process of complex simulation codes running on highly-parallel computer systems
- Start-up funding (2006–2011)
by Helmholtz Association of German Research Centres
- Activities
 - Development and integration of HPC programming tools
 - Correctness checking & performance analysis
- Academic workshops
- Training workshops
- Service
 - Support email lists
 - Application engagement

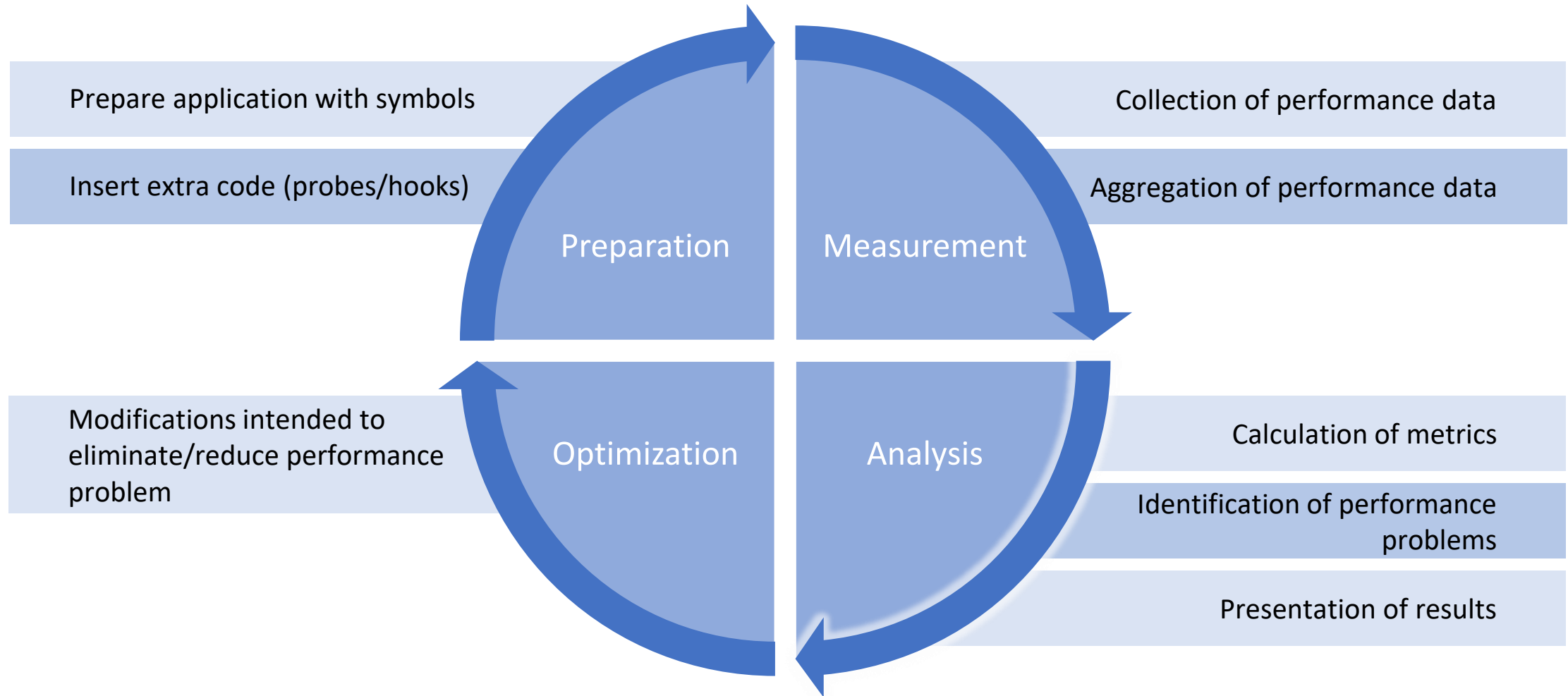
<http://www.vi-hps.org>



Tools will ***not*** automatically make you,
your applications or computer systems
more productive.

However, they can help you understand
how your parallel code executes and
when / where it's necessary to work on
correctness and performance issues.

Performance Engineering Workflow



JupyterLab - Supercomputing in your Browser



The screenshot displays the JupyterLab web interface. The browser address bar shows the URL: `jupyter.jsc.fz-juelich.de/user/a.visser_at_fz-juelich.de/j1b57726f4eb448...`. The interface includes a top menu bar with options: File, Edit, View, Run, Kernel, Git, LaTeX, Tabs, Settings, Help. The left sidebar contains 'FAVORITES' (HOME, PROJECT, SCRATCH) and 'FILE BROWSER' (cubelib). The central terminal window shows the command prompt `visser1@jrlogin04:~/cubelib` and the output of `ls`, listing files like `aclocal.m4`, `AUTHORS`, `autom4te.cache`, `bootstrap`, `build`, `build-ac`, `build-config`, `build-frontend`, `build-includes`, `ChangeLog`, `CMakeLists.txt`, `cmake-modules`, `common`, `configure`, `configure~`, `configure.ac`, `COPYING`, `doc`, `examples`, `HACKING`, `include`, `INSTALL`, `INSTALL -cmake`, `Makefile.am`, `Makefile.in`, `modules.txt`, `OPEN_ISSUES`, `README`, `share`, `src`, `test`, and `vendor`. The right sidebar shows a 'Launcher' with icons for Notebook, Python 3 (ipykernel), Bash, C++2b, Cube (4.9.1), Java -21, and Julia 1.12. The bottom status bar indicates 'Simple' mode, 3 tabs, 0 errors, and system resources: CPU: 0.00% | Mem: 242.10 / 257836.55 MB.

10/6/2026

76

Instrumentation and performance measurement



- Infrastructure for instrumentation and performance measurements
- Instrumented application can be used to produce several results:
 - Call-path profiling: CUBE4 data format used for data exchange
 - Event-based tracing: OTF2 data format used for data exchange
- Supported parallel paradigms:
 - Multi-process: MPI, SHMEM
 - Thread-parallel: OpenMP, Pthreads
 - Accelerator-based: CUDA, OpenCL, OpenACC, Kokkos
- Open Source; portable and scalable to all major HPC systems
- Initial project funded by BMBF
- Further developed in multiple 3rd-party funded projects

Score-P project idea



- Start a community effort for a common infrastructure
 - Score-P instrumentation and measurement system
 - Common data formats OTF2 and CUBE4
- Developer perspective:
 - Save manpower by sharing development resources
 - Invest in new analysis functionality and scalability
 - Save efforts for maintenance, testing, porting, support, training
- User perspective:
 - Single learning curve
 - Single installation, fewer version updates
 - Interoperability and data exchange
- Project funded by BMFTR
- Close collaboration PRIMA project funded by DOE

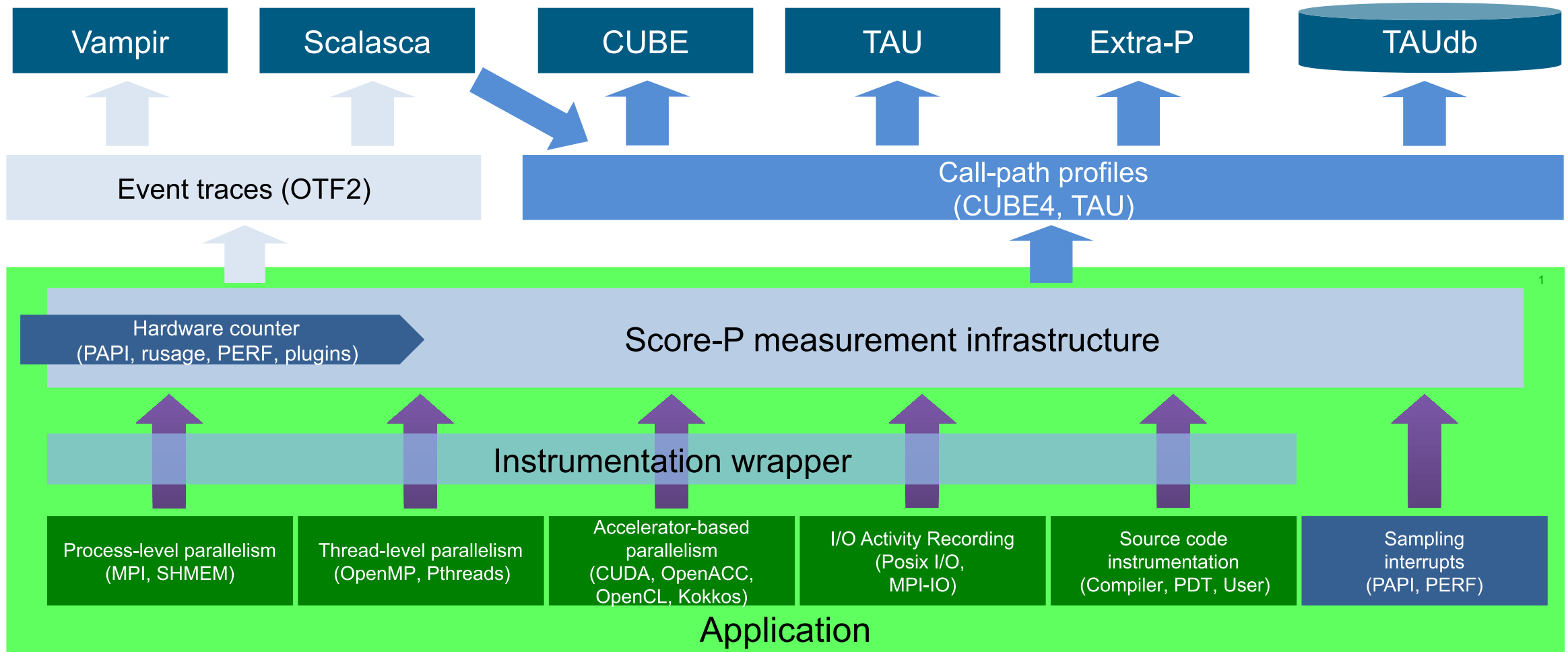
With funding from the:



Federal Ministry
of Research, Technology
and Space



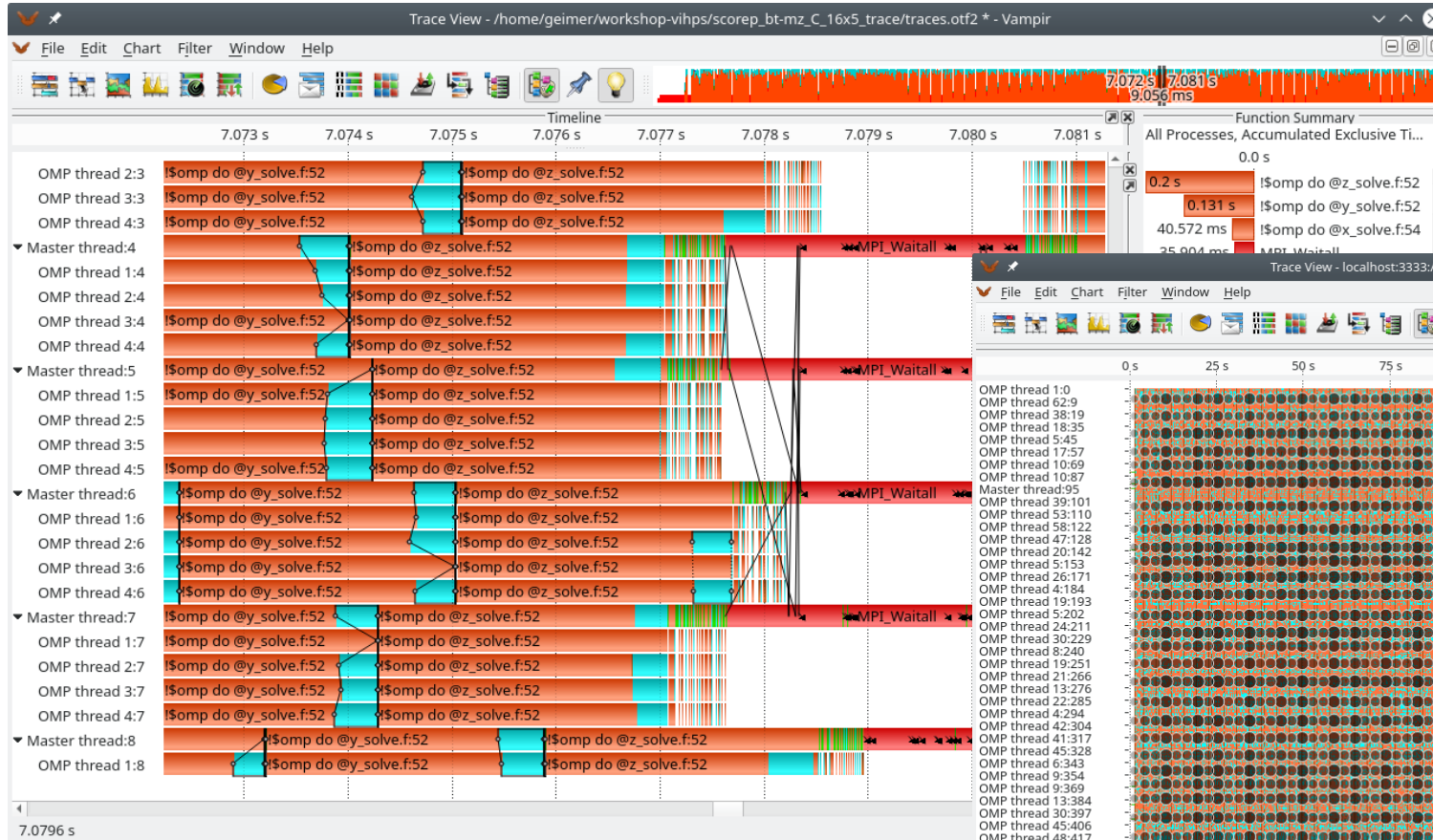
Score-P overview



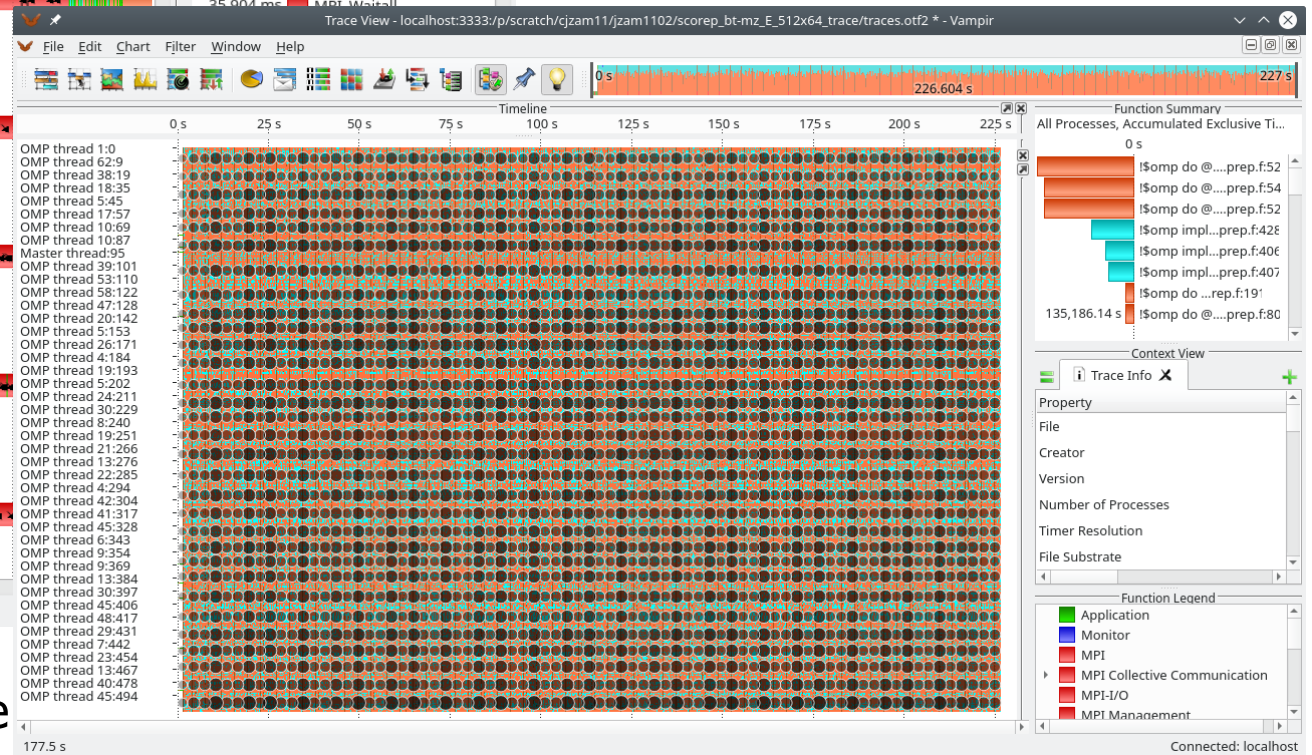
Automatic trace analysis



Trace Analysis



While interactive trace visualization is often intuitive...



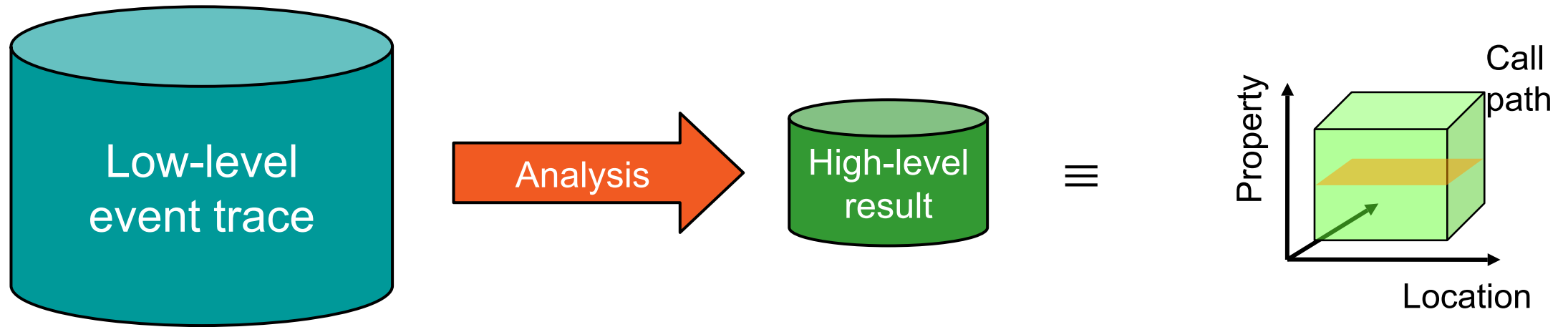
...finding bottlenecks may become the search for needles in a haystack...

10/6/2026

Automatic trace analysis



- Automatic search for patterns of inefficient behavior
- Classification of behavior & quantification of significance
- Identification of delays as root causes of inefficiencies



- Guaranteed to cover the entire event trace
- Quicker than manual/visual trace analysis
- Parallel replay analysis exploits available memory & processors to deliver scalability

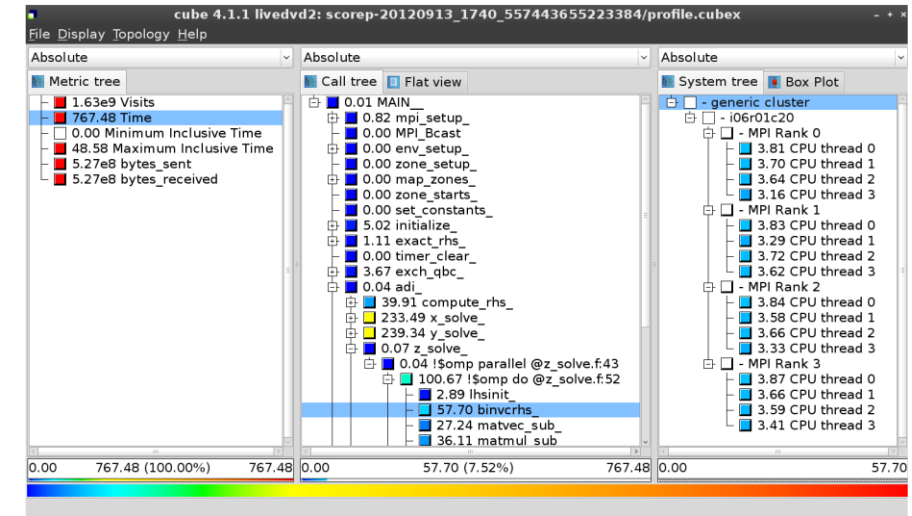
- Collection of trace-based performance tools
 - Specifically designed for large-scale systems
 - Features an automatic trace analyzer providing wait-state, critical-path, and delay analysis
 - Supports MPI, OpenMP, POSIX threads, and hybrid MPI+OpenMP/Pthreads
- Available under 3-clause BSD open-source license
- Documentation & sources:
 - <https://www.scalasca.org>
- Contact:
 - scalasca@fz-juelich.de



Analysis report examination with Cube



- Parallel program analysis report exploration tools
 - Libraries for XML+binary report reading & writing
 - Algebra utilities for report processing
 - GUI for interactive analysis exploration
 - Requires Qt5 or Qt 6
- Originally developed as part of the Scalasca toolset
- Now available as a separate component
 - Can be installed independently of Score-P, e.g., on laptop or desktop
 - Latest release: Cube v4.9.1 (April 2026)



Note:
Binary packages provided for Windows & MacOS,
from www.scalasca.org website in software/Cube-4x

Cube GUI (HPC system)

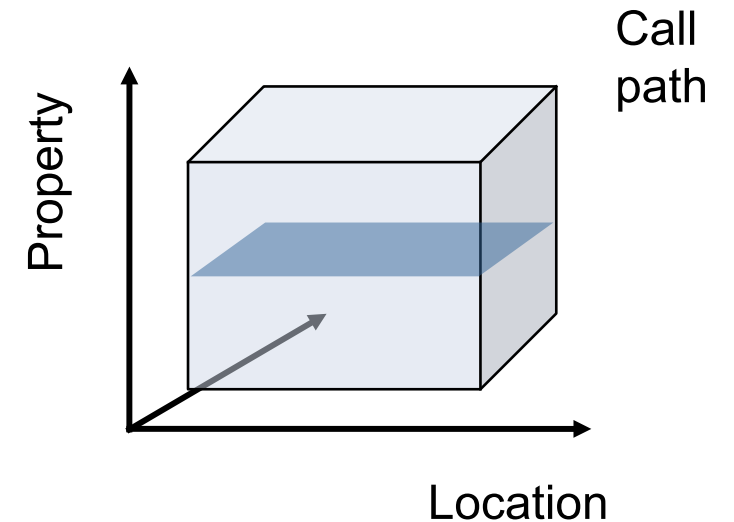


- Run on JupyterLab
 - Completely integrated: just click the icon
- Run **remote**
 - start X server (e.g., Xming) locally
 - connect to HPC system with X forwarding enabled
 - load cube module and start cube remotely
- Install & run **local**
 - install Cube GUI locally on desktop
 - binary packages available for MacOS & Windows and externally provided by OpenHPC and various Linux distributions
 - source package available for Linux, requires Qt
 - configure/build/install manually or use your favourite framework (e.g. Spack or EasyBuild)
 - copy .cubex file (or entire scorep directory) to desktop from remote system
OR locally mount remote filesystem
 - start cube locally

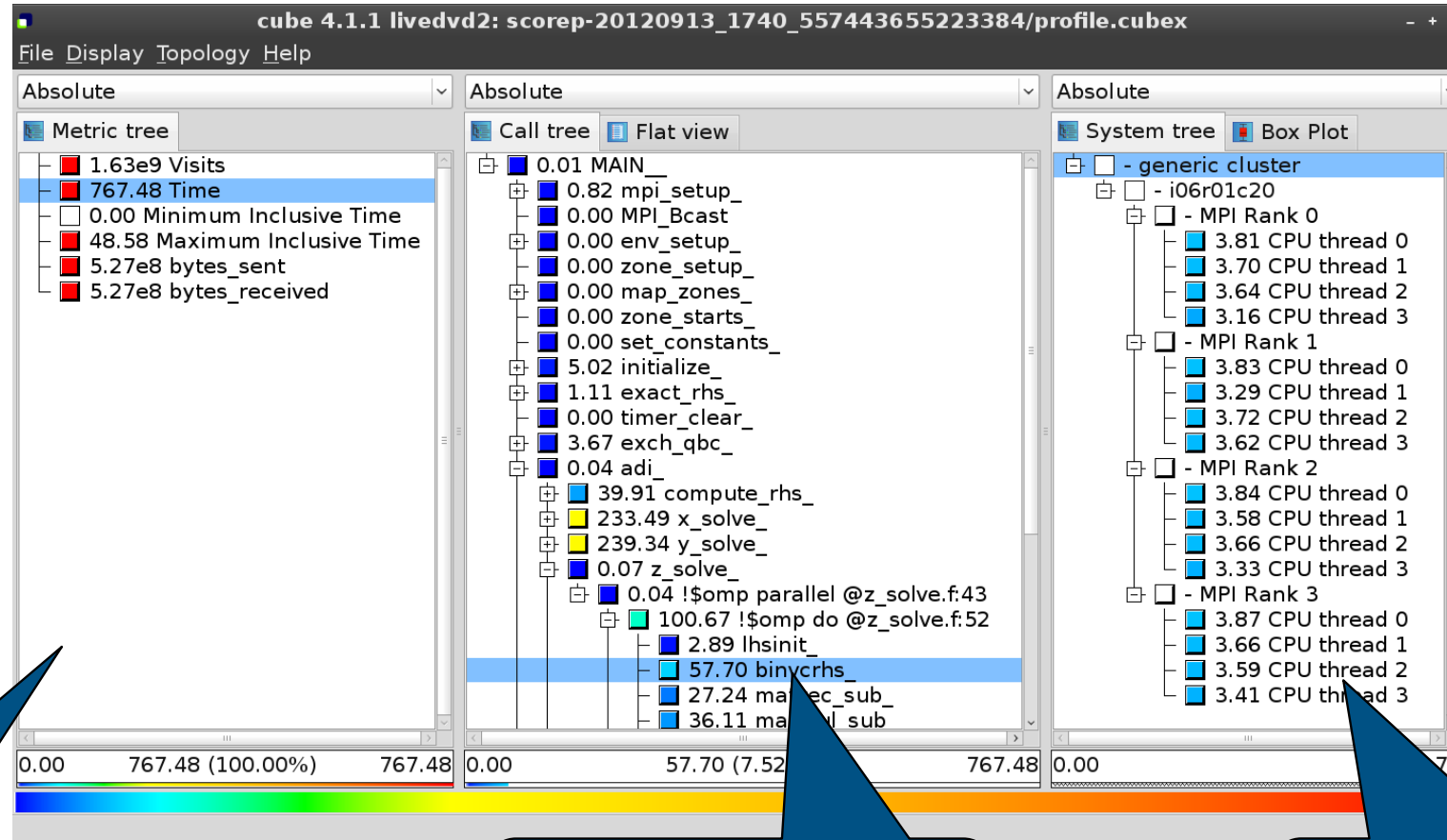


mailto: scalasca@fz-juelich.de

- Representation of values (severity matrix) on three hierarchical axes
 - Performance property (metric)
 - Call path (program location)
 - System location (process/thread)
- Three coupled tree browsers
- Cube displays severities
 - As *value*: for precise comparison
 - As *colour*: for easy identification of hotspots
 - Inclusive value when closed & *exclusive* value when expanded
 - Customizable via display modes



Analysis presentation



What kind of performance metric?

10/6/2026

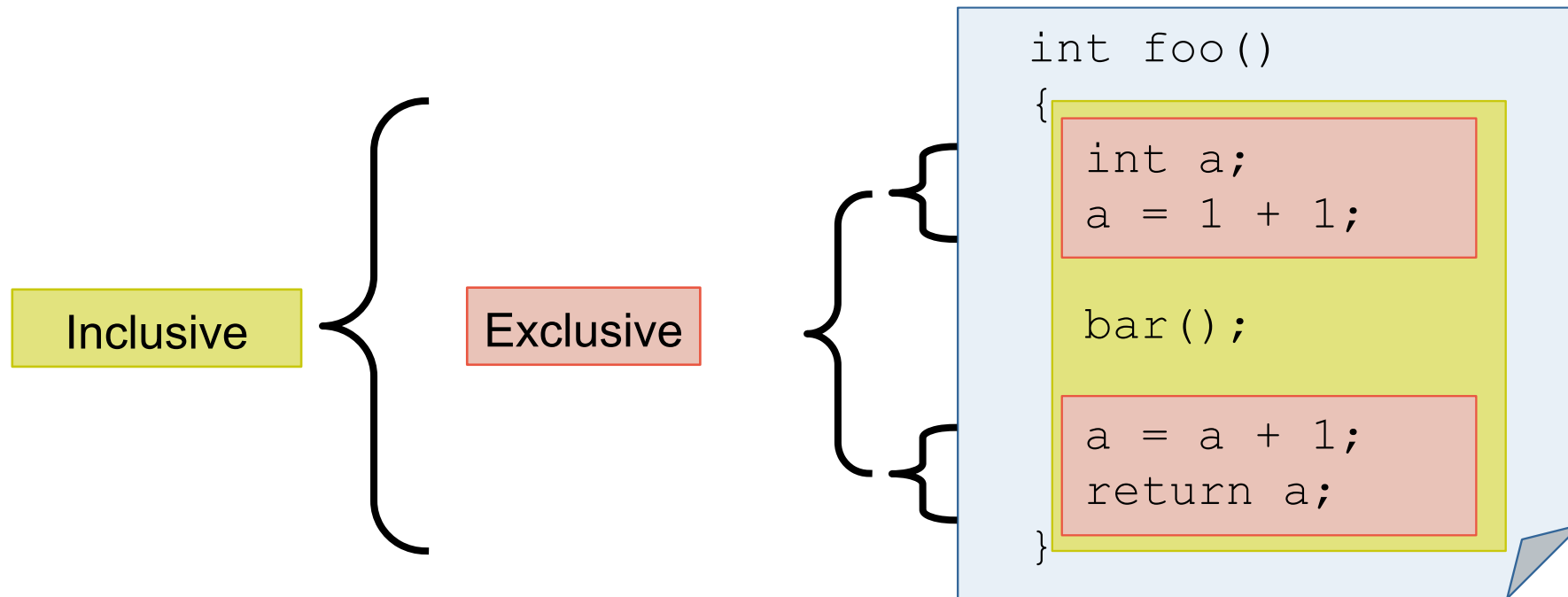
Where is it in the source code?
In what context?

How is it distributed across the processes/threads?

Inclusive vs. exclusive values

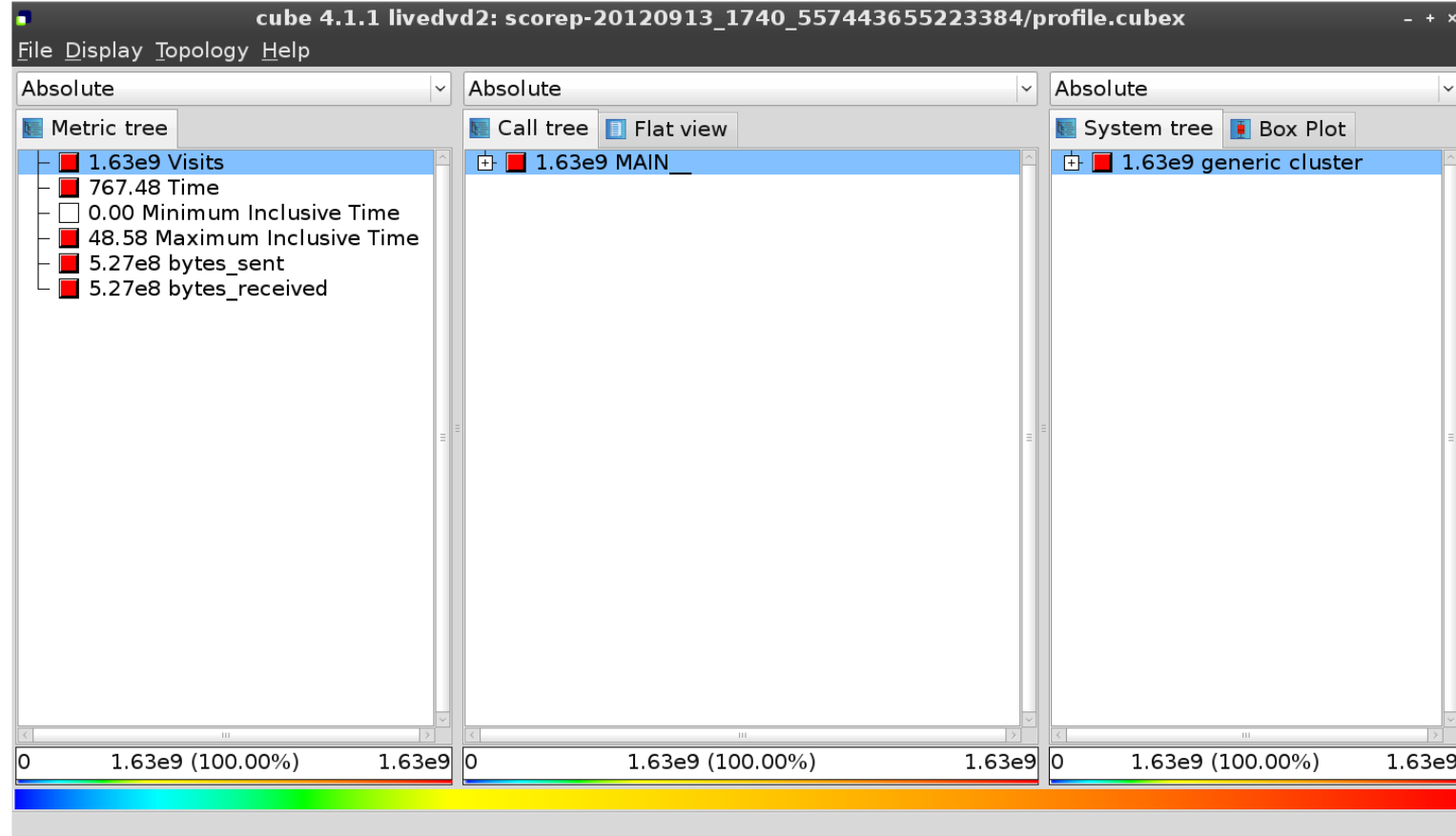


- Inclusive
 - Information of all sub-elements aggregated into single value
- Exclusive
 - Information cannot be subdivided further



Score-P analysis report exploration

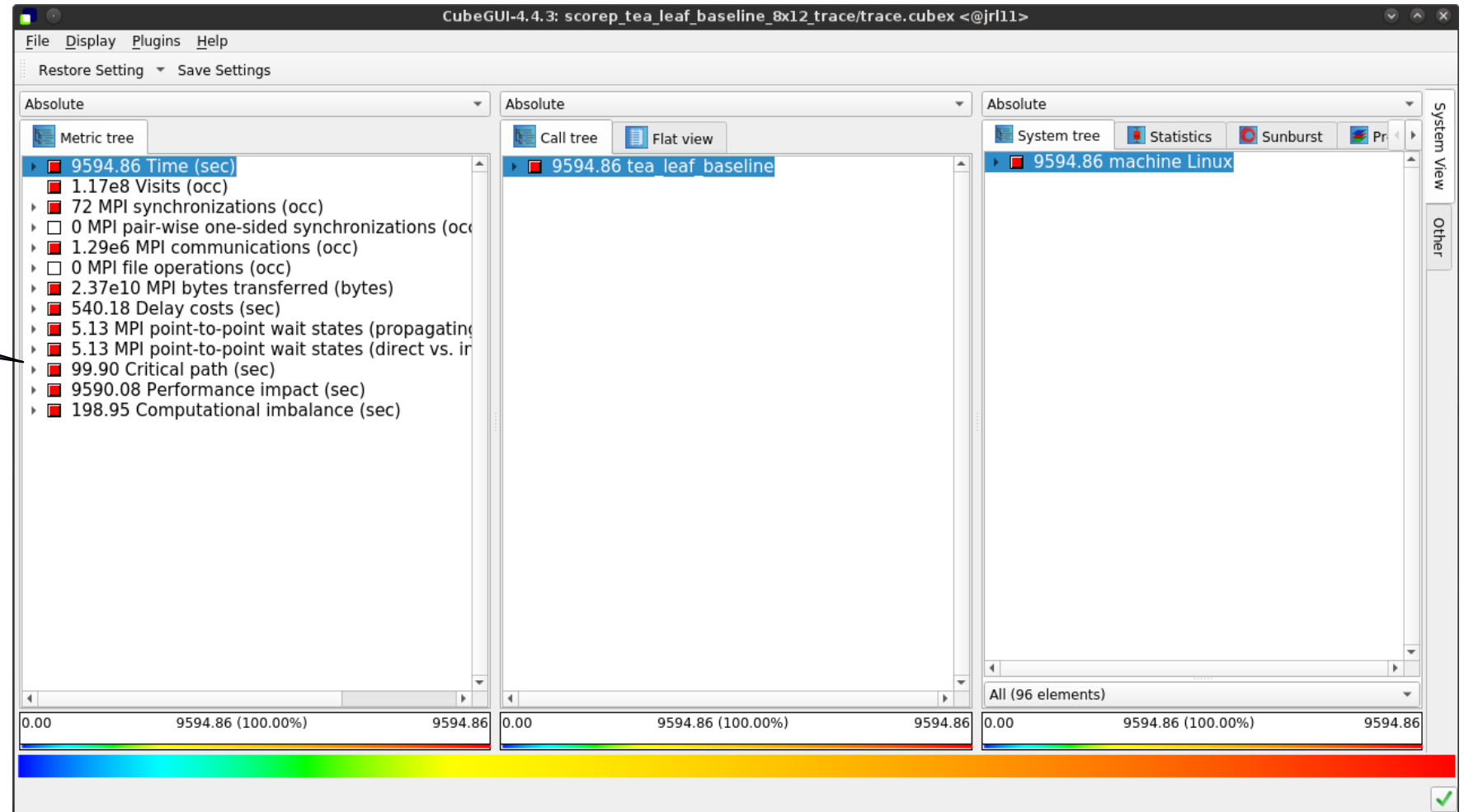
(opening view)



10/6/2026

Scalasca analysis report exploration

(Scalasca trace analysis)



Additional top-level metrics produced by the trace analysis...

Cube: Further information

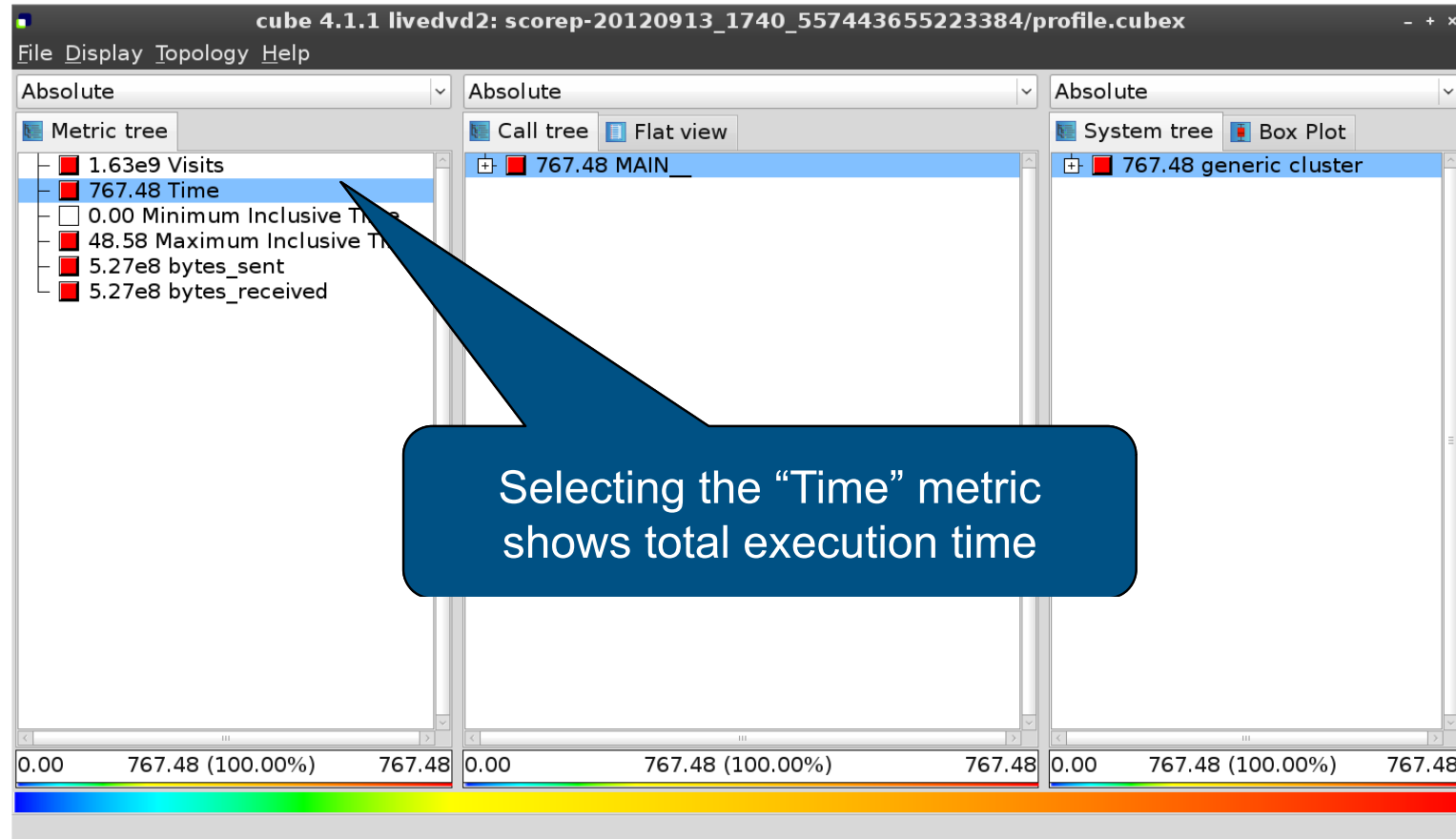


- Parallel program analysis report exploration tools
 - Libraries for Cube report reading & writing
 - Algebra utilities for report processing
 - GUI for interactive analysis exploration
- Available under 3-clause BSD open-source license
- Documentation & sources:
 - <http://www.scalasca.org>
- User guide also part of installation:
 - <prefix>/share/doc/CubeGuide.pdf
- Contact:
 - mailto: scalasca@fz-juelich.de

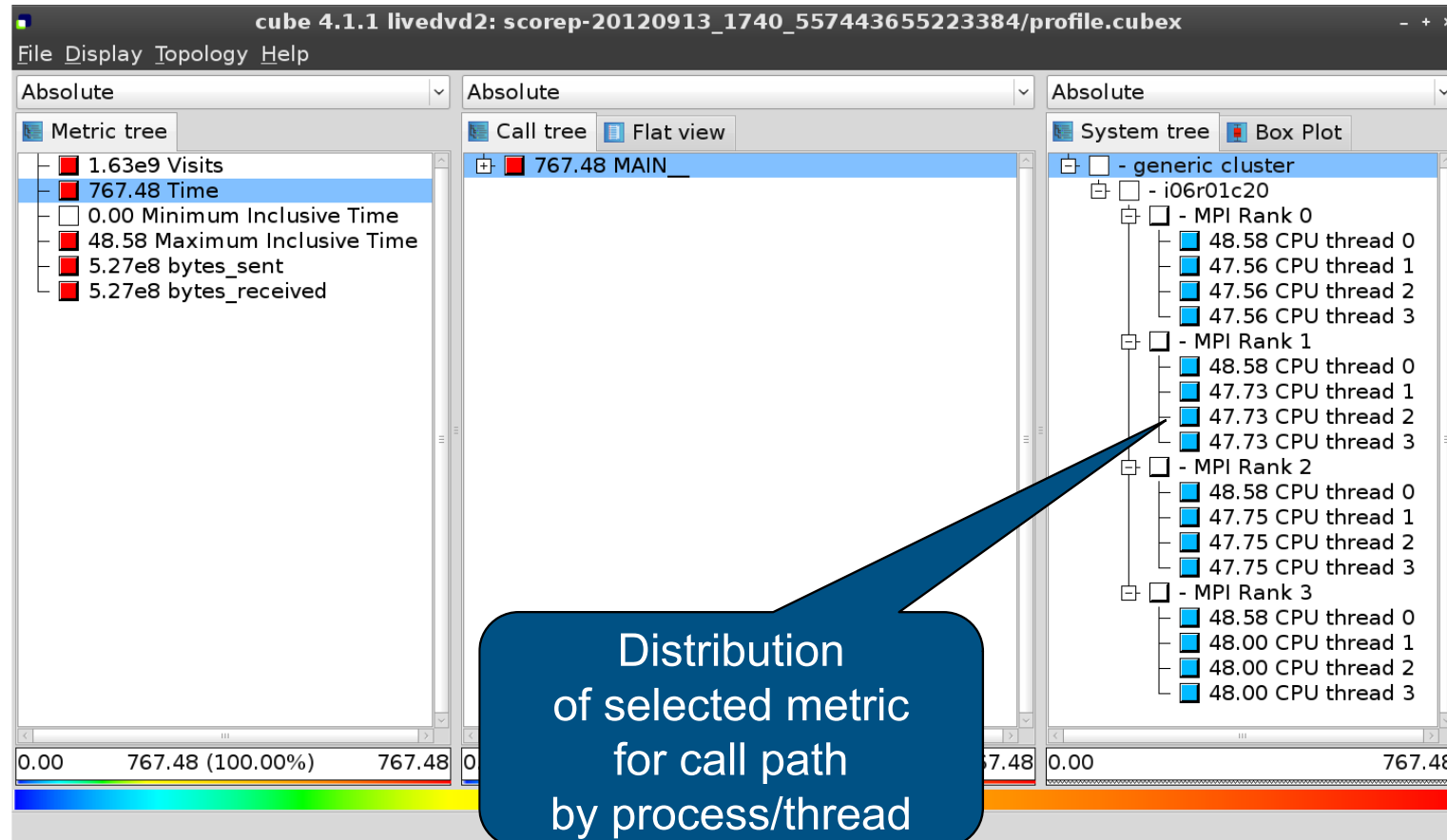




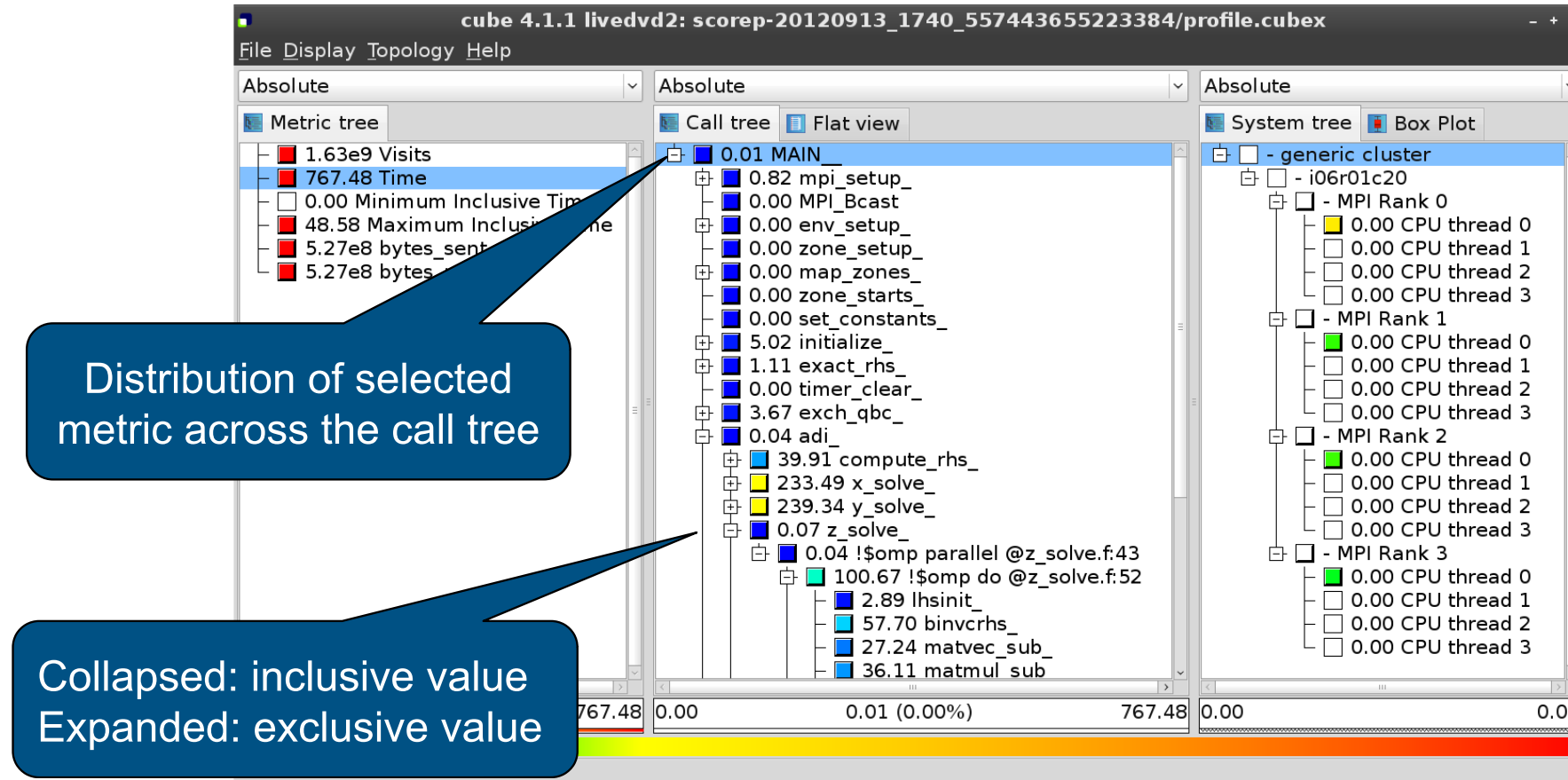
Metric selection

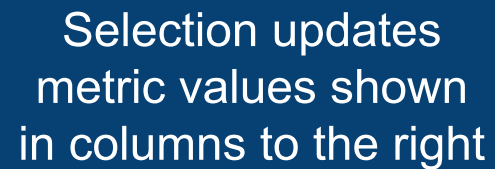


Expanding the system tree

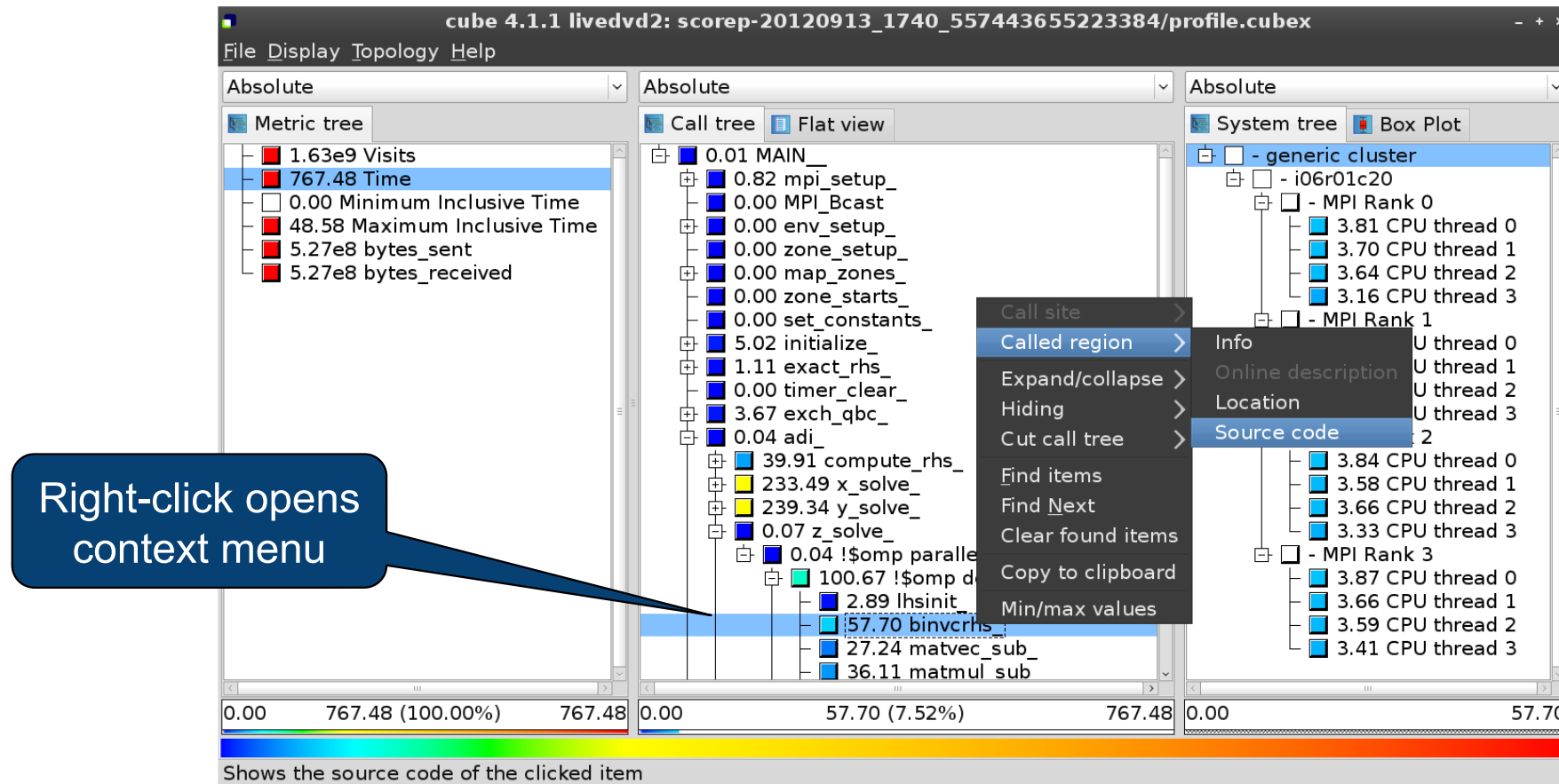


Expanding the call tree





Source-code View via Context Menu



Source-code view



```
/home/geimer/Projects/Tests/NPB3.3-MZ-MPI/BT-MZ/solve_subs.f
subroutine binvcrhs( lhs,c,r )
C-----
C-----
C-----
C
C-----

implicit none

double precision pivot, coeff, lhs
dimension lhs(5,5)
double precision c(5,5), r(5)

C-----
C
C-----

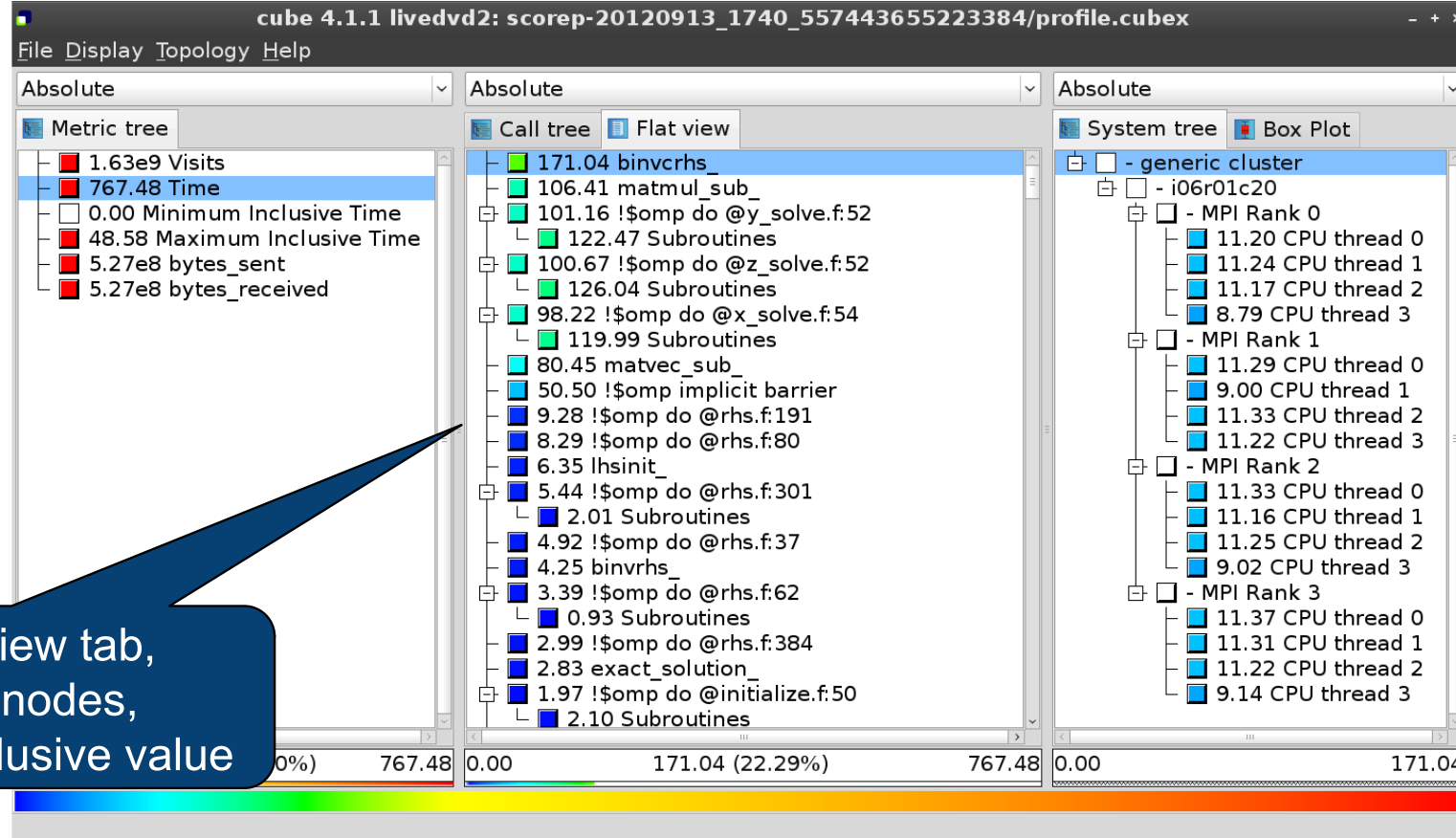
pivot = 1.00d0/lhs(1,1)
lhs(1,2) = lhs(1,2)*pivot
lhs(1,3) = lhs(1,3)*pivot
lhs(1,4) = lhs(1,4)*pivot
lhs(1,5) = lhs(1,5)*pivot
c(1,1) = c(1,1)*pivot
c(1,2) = c(1,2)*pivot
c(1,3) = c(1,3)*pivot
c(1,4) = c(1,4)*pivot
```

☒ Read only Save Save as Fo

Note:

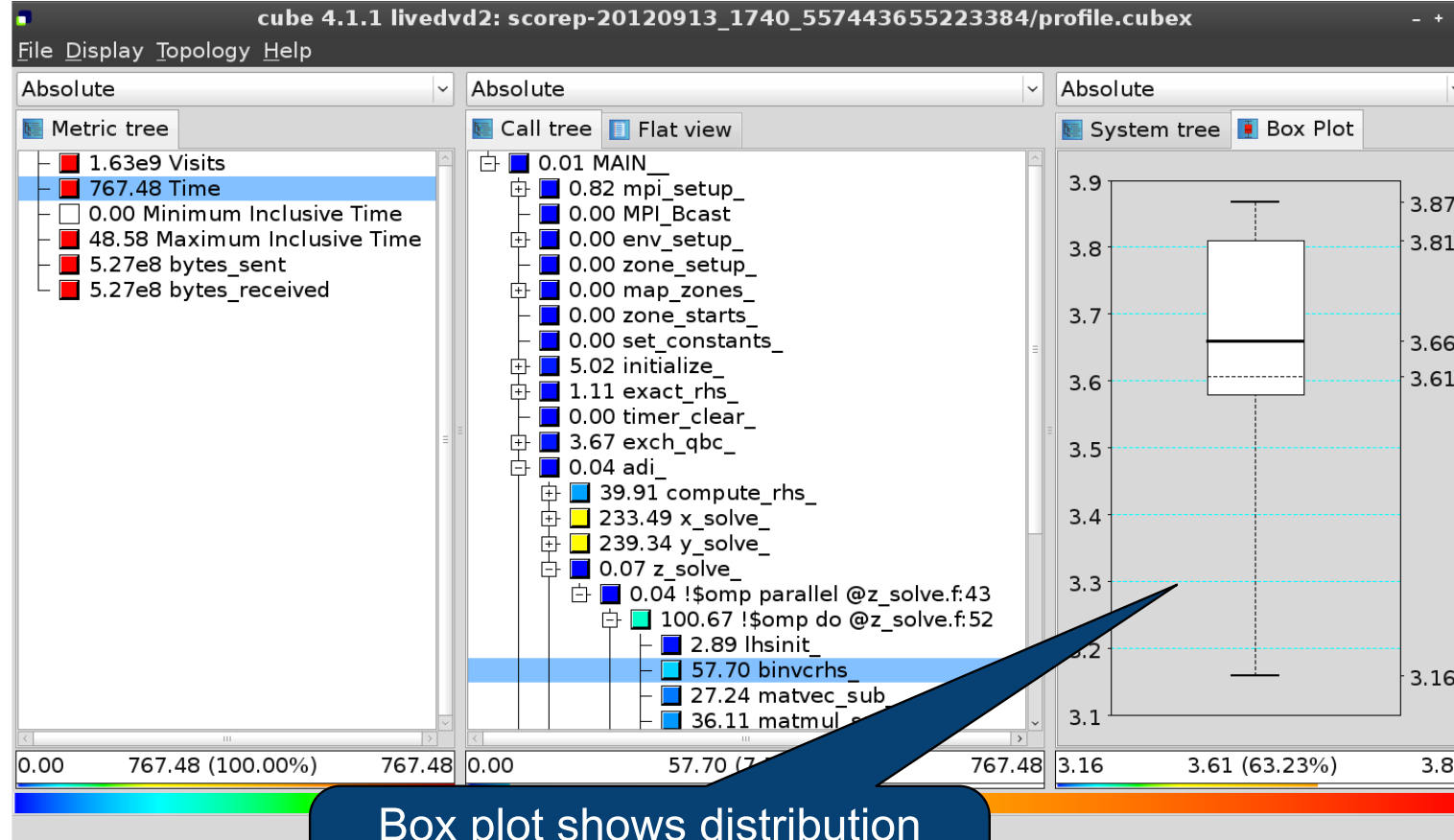
This feature depends on file and line number information provided by the instrumentation, i.e., it may not always be available

Flat profile view

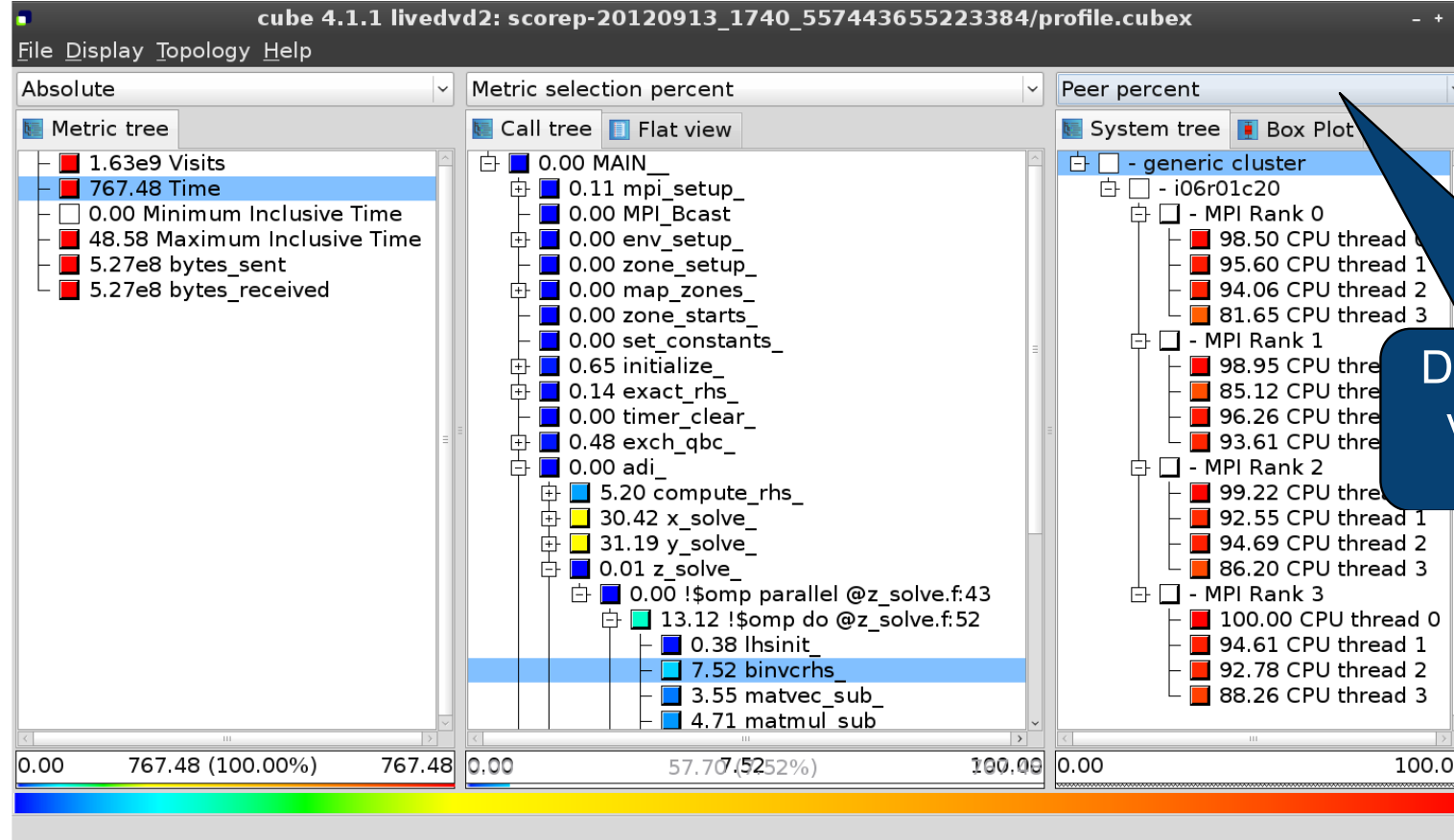


Select flat view tab,
expand all nodes,
and sort by exclusive value

Box plot view



Alternative display modes



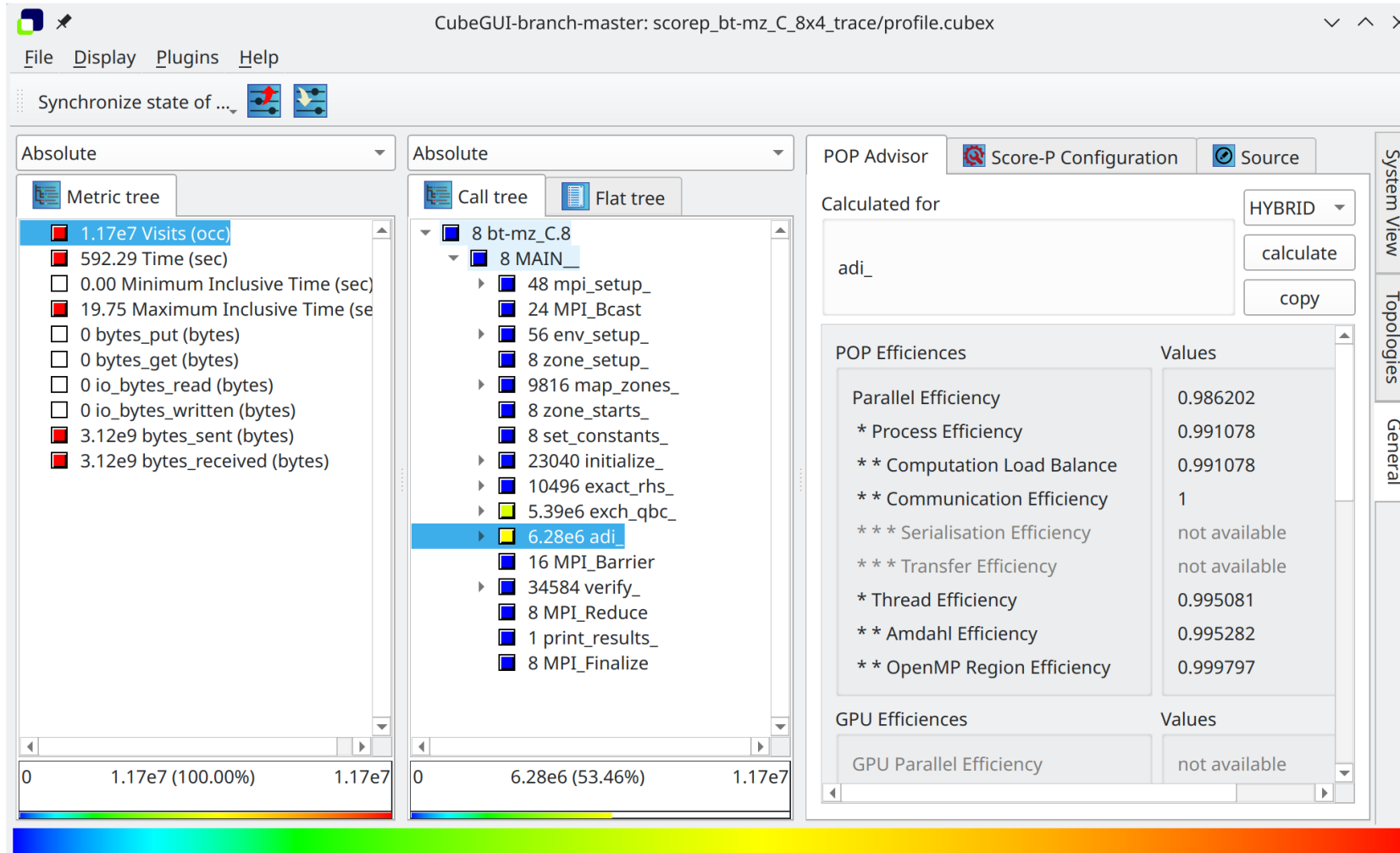
Data can be shown in various percentage modes

Important display modes



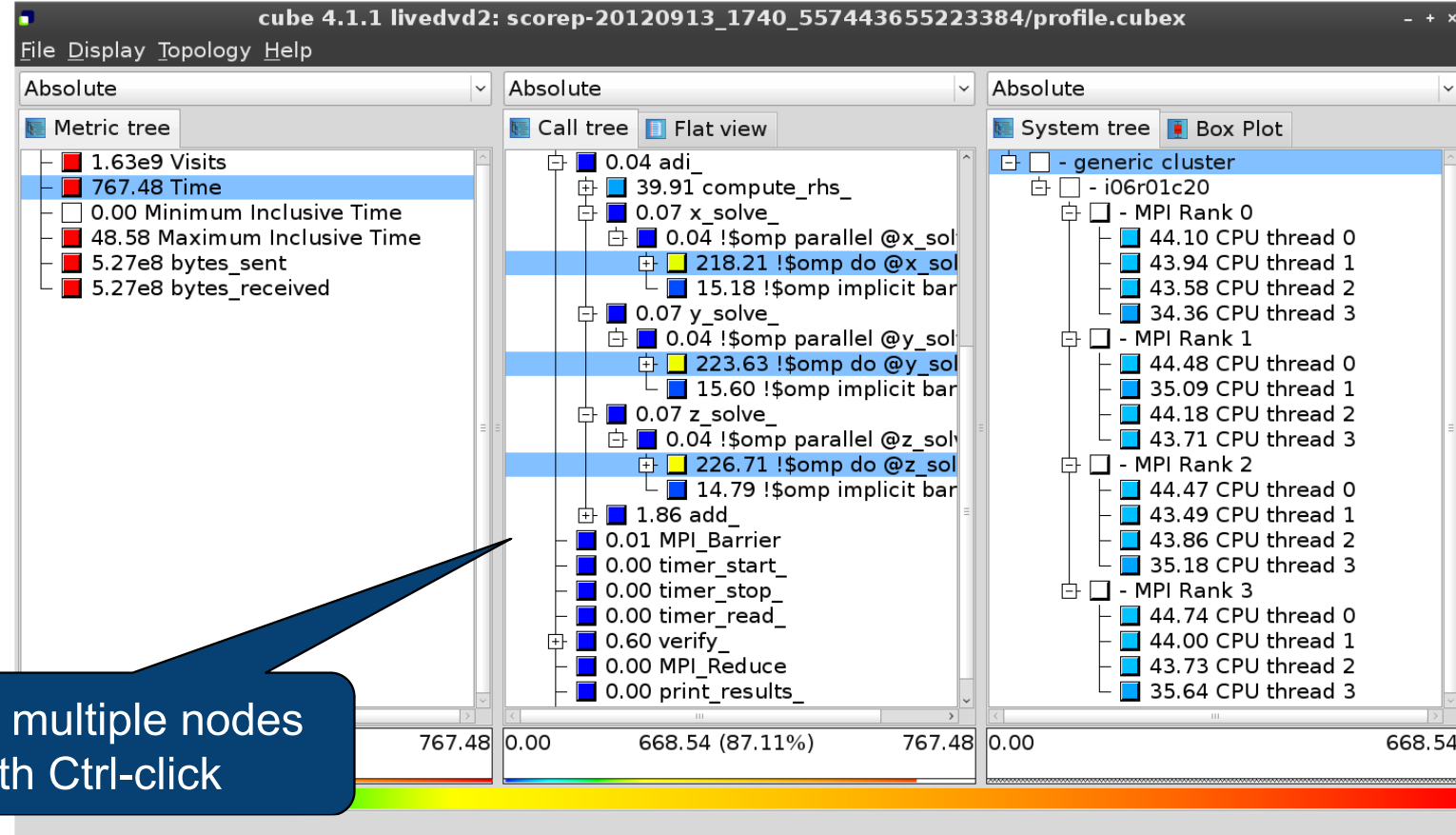
- Absolute
 - Absolute value shown in seconds/bytes/counts
- Selection percent
 - Value shown as percentage w.r.t. the selected node “on the left” (metric/call path)
- Peer percent (system tree only)
 - Value shown as percentage relative to the maximum peer value

POP Advisor

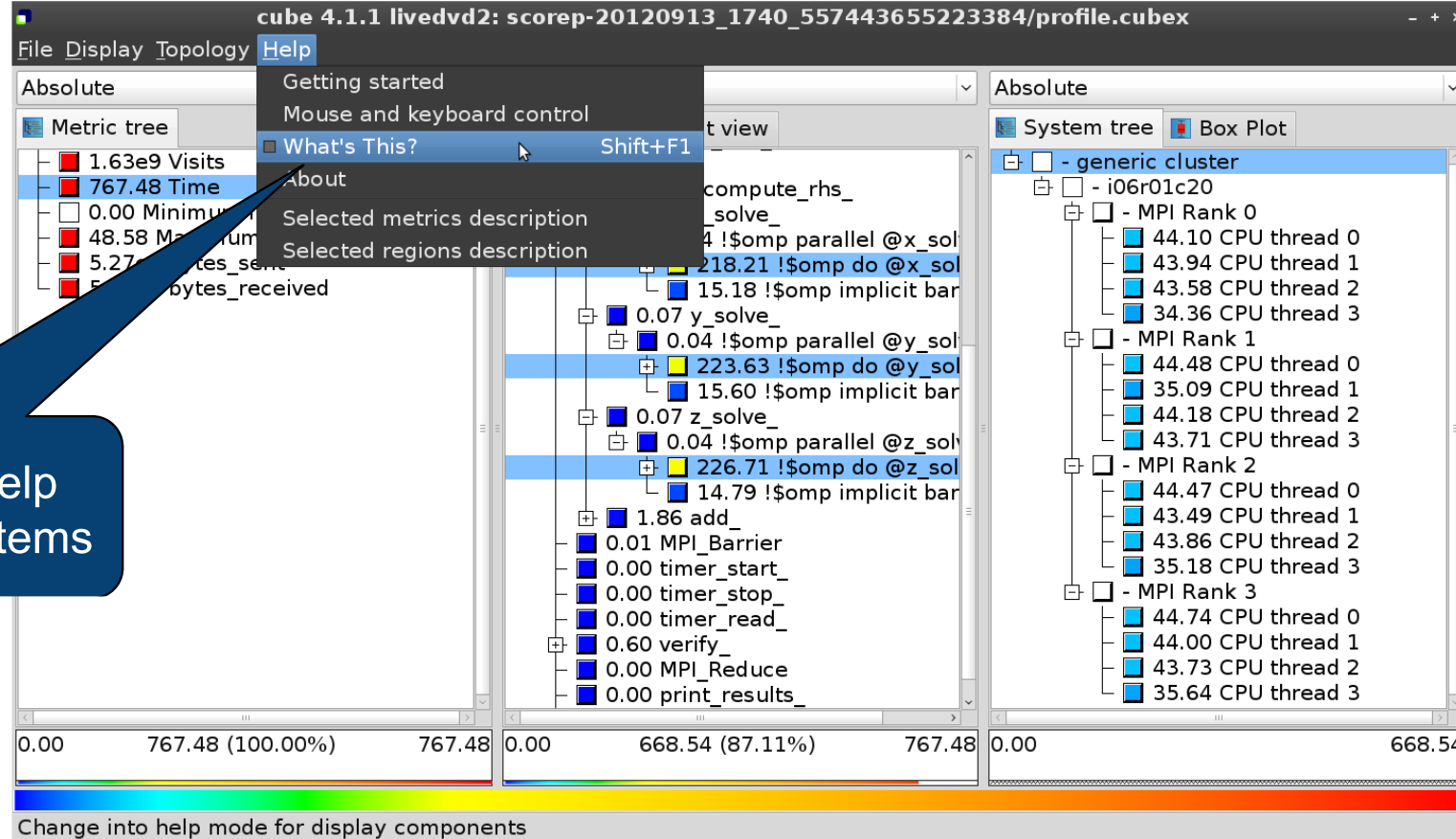


Advisor calculates several POP metrics

Multiple selection

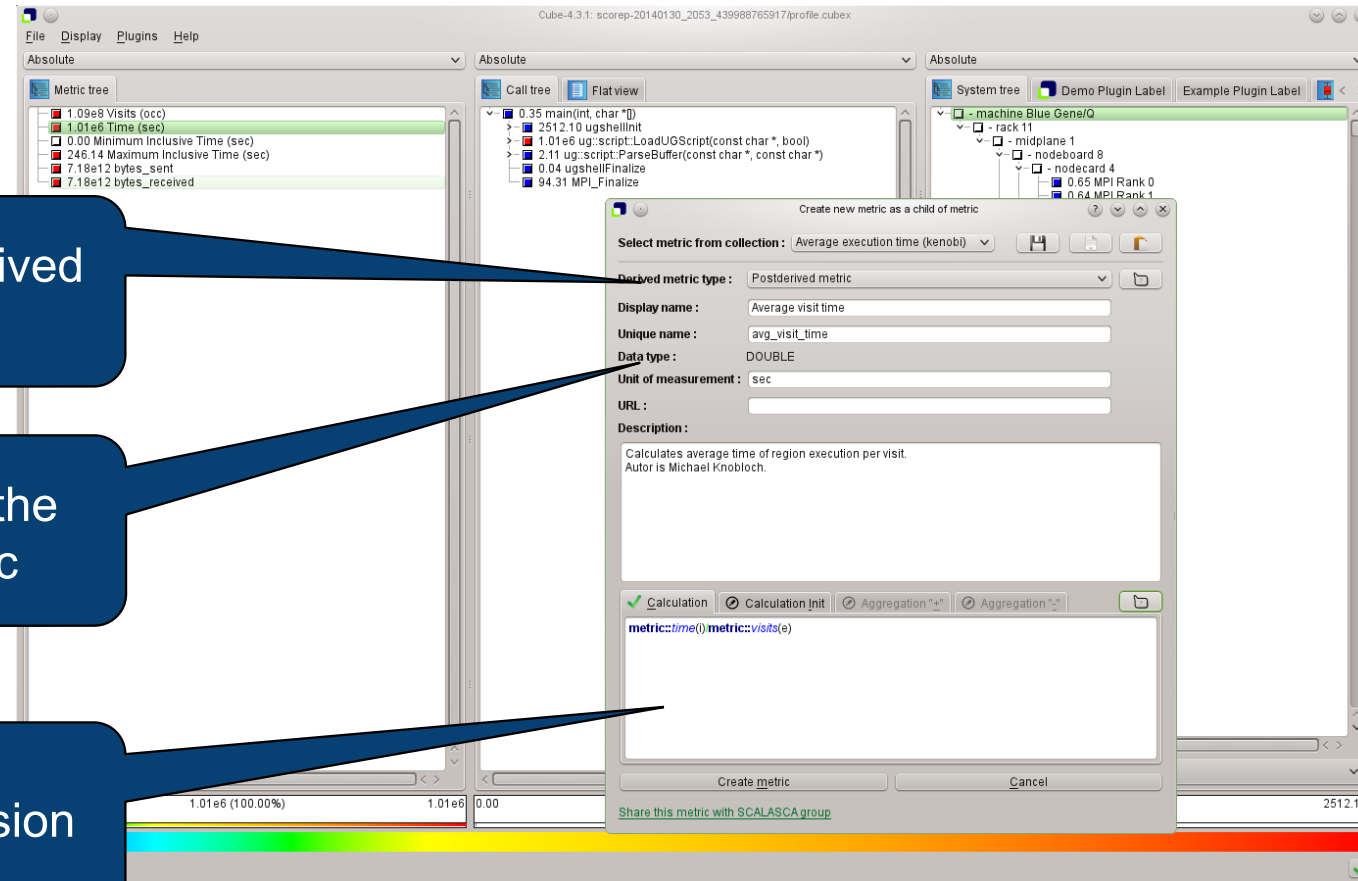


Context-sensitive help



Context-sensitive help
available for all GUI items

Derived metrics in Cube GUI



Collection of derived metrics

Parameters of the derived metric

CubePL expression

- Derived metrics are defined using CubePL expressions, e.g.:
`metric::time(i)/metric::visits(e)`
- Values of derived metrics are not stored, but calculated on-the-fly
- Types of derived metrics:
 - Prederived: evaluation of the CubePL expression is performed before aggregation
 - Postderived: evaluation of the CubePL expression is performed after aggregation
- Examples:
 - “Average execution time”: Postderived metric with expression
`metric::time(i)/metric::visits(e)`
 - “Number of FLOP per second”: Postderived metric with expression
`metric::FLOP()/metric::time()`

POP Tools Overview



More information at: <https://pop-coe.eu/partners/tools>



Performance Optimisation and Productivity 3

A Centre of Excellence in HPC

Contact:

 <https://www.pop-coe.eu>

 pop@bsc.es

 [@POP_HPC](#)

 [youtube.com/POPHPC](https://www.youtube.com/POPHPC)



This project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 101143931. The JU receives support from the European Union's Horizon Europe research and innovation programme and Spain, Germany, France, Portugal and the Czech Republic.

