HPC.NRW

Performance Analysis 101

HPC.NRW Competence Network

Performance Analysis – Terms and Techniques

HPC.NRW Competence Network

Performance Analysis 101

Acknoledgements

- This presentation uses content from
  - VI-HPS (→ www.vi-hps.org) with content from
  - Bernd Mohr (Forschungszentrum Jülich) and
  - Luiz DeRose (Cray)

Goals of Performance Analysis

HPC.NRW

- Gain an understanding of the application behavior
- Identify potential performance problems of the application and their sources
  - You observe the symptoms, but need to fix the problem
- Present measurement and analysis data in a human understandable way
- Performance analysis tools do not automatically tune your application
  - Auto-tuning tools exist, but are not covered here

Performance factors of parallel applications

HPC.NRW

- “Sequential” performance factors
  - Computation
    - Choose right algorithm, use optimizing compiler
  - Cache and memory
    - Tough! Only limited tool support, hope compiler gets it right
  - Input / output
    - Often not given enough attention

- “Parallel” performance factors
  - Partitioning / decomposition (→ load balance)
  - Communication (i.e., message passing)
  - Multithreading
  - Synchronization / locking
    - More or less understood, good tool support

(Source: VI-HPS)

Tuning basics

HPC.NRW

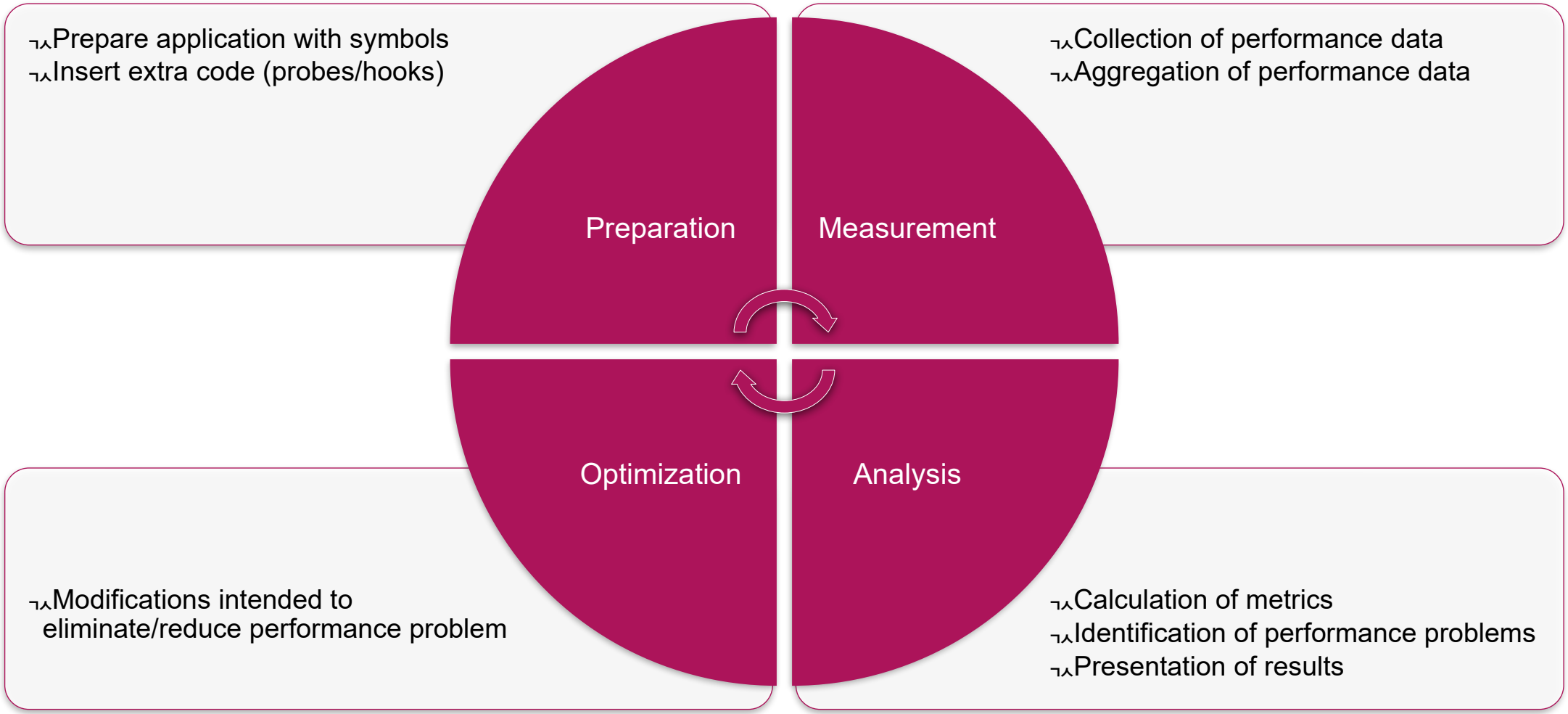
- Successful performance engineering is a combination of
  - Careful setting of various tuning parameters
  - The right algorithms and libraries
  - Compiler flags and directives
  - …
  - Thinking !!!

- Measurement is better than guessing
  - To determine performance bottlenecks
  - To compare alternatives
  - To validate tuning decisions and optimizations
    - After each step!

(Source: VI-HPS)

Performance engineering workflow

HPC.NRW



(Source: VI-HPS)

The 80/20 rule

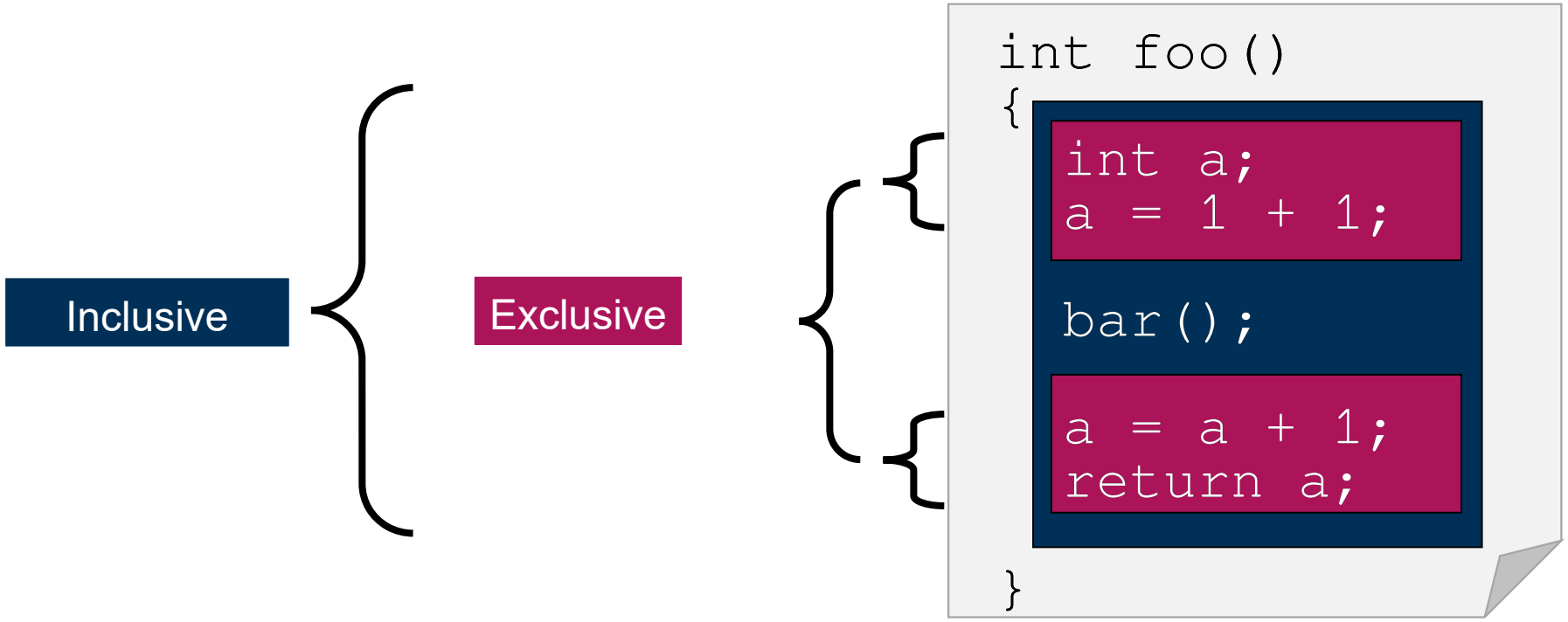
HPC.NRW

- 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!

(Source: VI-HPS)

Inclusive vs. Exclusive values

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



(Source: VI-HPS)

The Performance Analysis Tool Landscape

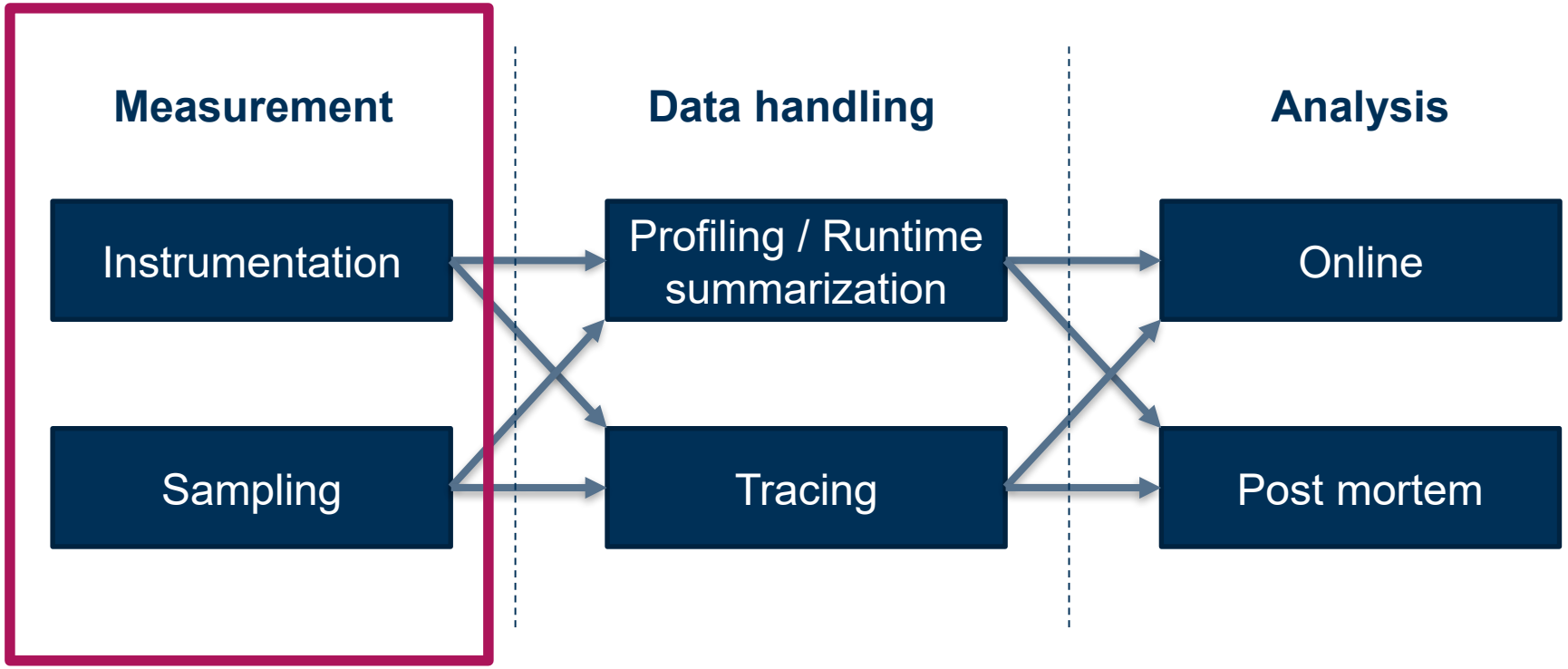
HPC.NRW

- Vast number of different tools available

- Which tool to use?
  - Purpose may dictate the type of tool
  - Tools often specialized for a specific purpose (Theading, Cache, Communication, …)

- Coarse-grained (technical) classification along three dimensions
  - Measurement
    - How is measurement data obtained? (Instrumentation vs. Sampling)
  - Data handling
    - How is measurement data handled at runtime? (Runtime summarization vs. Tracing)
  - Analysis
    - When is analysis data obtained / presented? (Online vs. Post-mortem)

Tool Classification



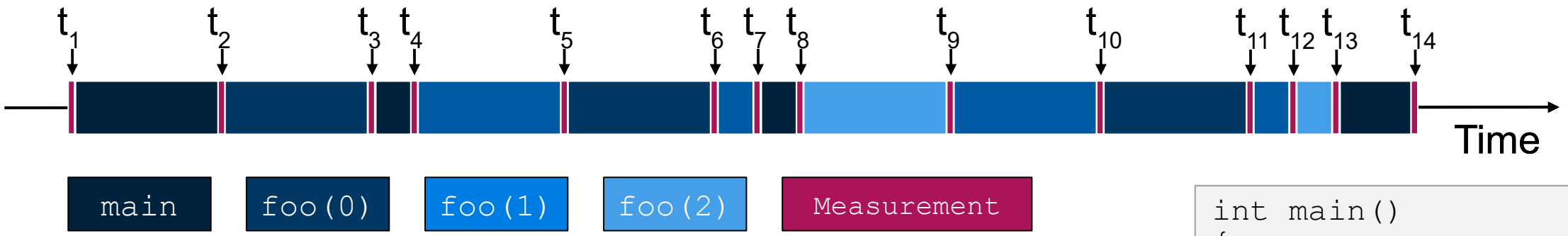
Instrumentation

HPC.NRW

- Insert measurement code into the application at specific locations in the application
  - State transitions (events)
    - Entering/Leaving a function call (provides context)
    - Communication and synchronization points (provides interaction and dependency information)

- Can be done …
  - Before compilation (using source-to-source translation)
  - During compilation (using compiler hooks)
  - During linking (using pre-instrumented libraries)
  - After compilation (using binary instrumentation / binary rewriting)

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");
}
```

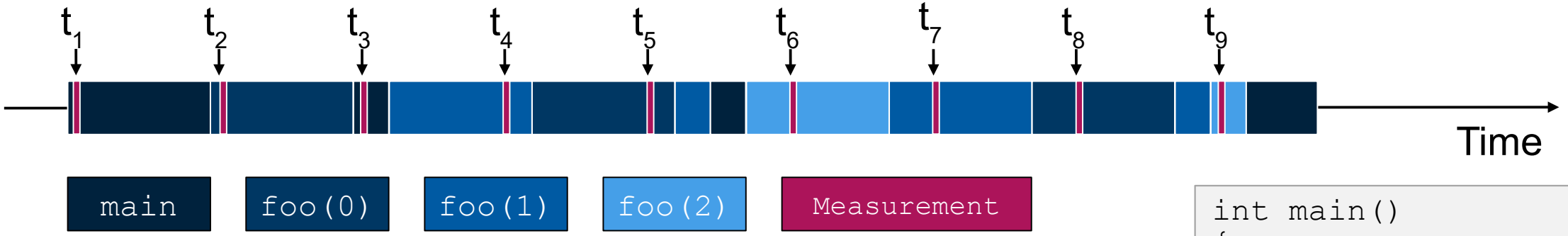
(Source: VI-HPS)

Sampling

HPC.NRW

- Measurement system is not part of the application
  - Application is interrupted by the measurement system to obtain data (a.k.a. “a sample”)
- Application is unmodified (debugging symbols may provide additional context)
  - All compiler optimizations stay in the code
- Behavior between samples remains a blackbox
  - Individual important state transitions may be missed / not be sampled
  - With sufficient measurement time data becomes more reliable
    - Statistically even rare events will be sampled at some point

Sampling

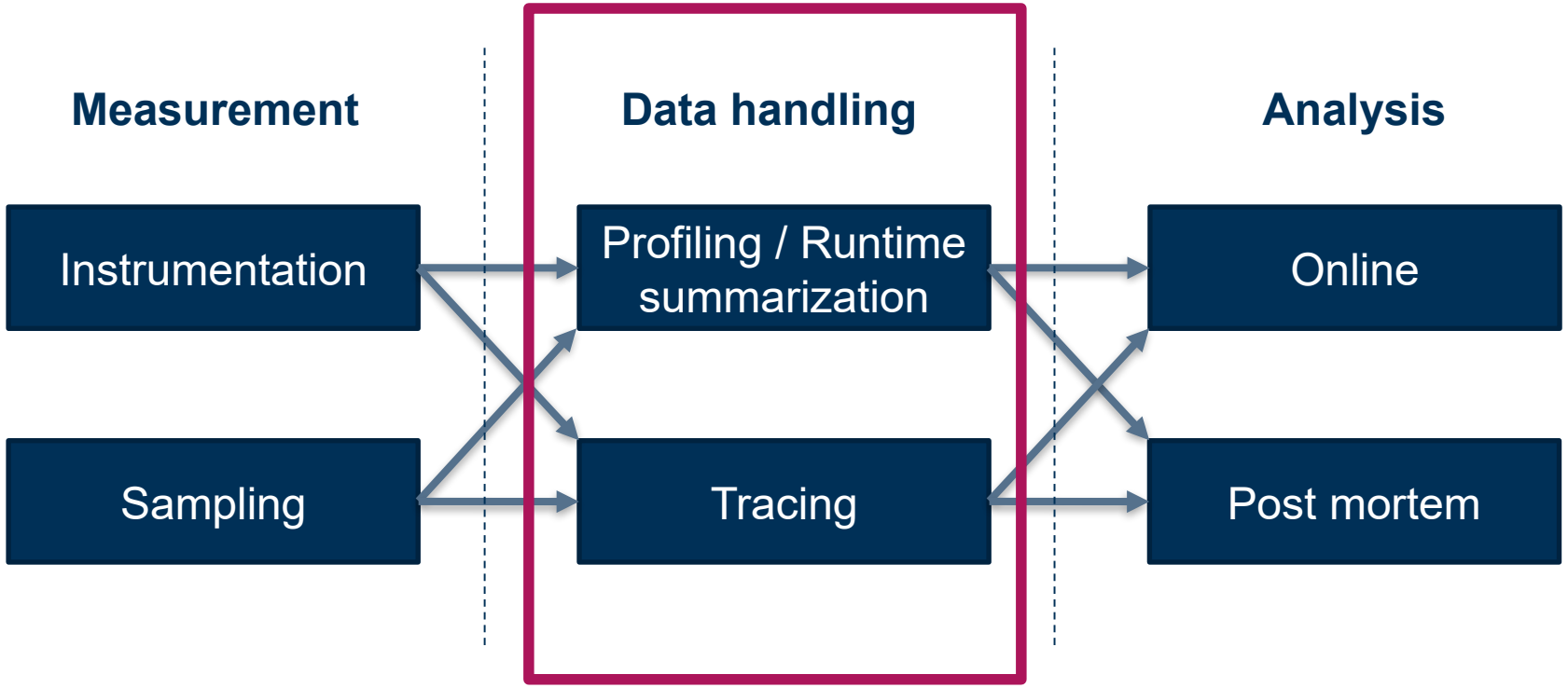


- Running program is periodically interrupted to take measurement
  - Timer interrupt, OS signal, or HWC overflow
  - Code position derived from debug information
- Statistical inference of program behavior
  - Not very detailed information on highly volatile metrics
  - Requires long(er)-running applications
- 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);

}
```

(Source: VI-HPS)

Tool Classification



Runtime Summarization (Profiling)

HPC.NRW

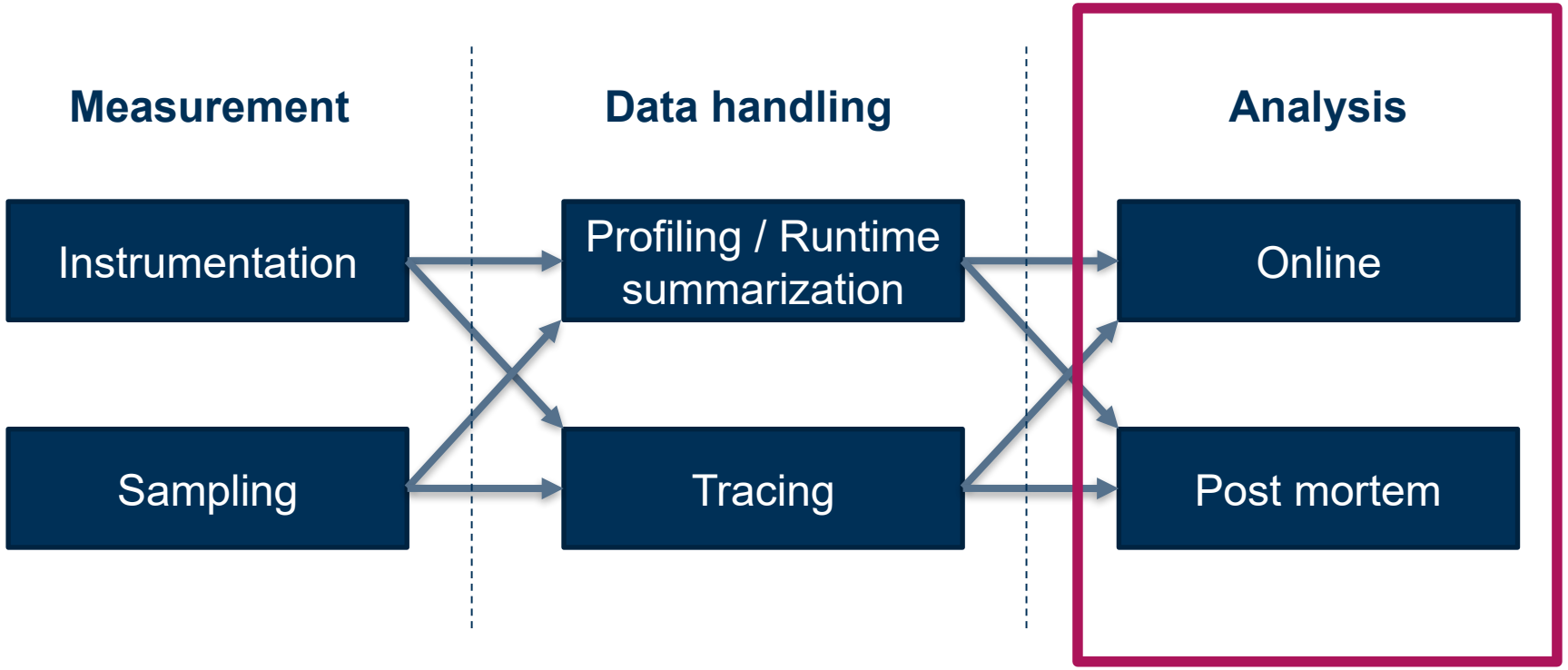
- Measurement data is processed and aggregated at runtime
  - Original measurement data instance is discarded after aggregation
- Memory footprint independent of execution time
  - Data is aggregated
  - Metric values accumulate over time
- Individual information of measurement data instances is lost
  - Some statistical metrics can be obtained at runtime
  - Some information is lost during processing

Tracing

HPC.NRW

- Record measurement data instances
  - No aggregation at runtime
  - No computation of metrics at runtime
- Memory footprint dependent on execution time
  - Can become prohibitive for long running applications
- Complex metrics can be computed from trace
  - Profile can be created from trace, but not vice versa
- Computation of metrics at temporally or spatially different location
  - Online but different computer
  - Post-mortem on the same (or different) computer

Tool Classification



Online Analysis

HPC.NRW

- Performance metrics can be investigated at runtime
  - Diagnostics screen
  - Visualization
- Useful in computational steering
  - Use performance information to guide simulation configuration
- Useful in automatic tuning & load balancing
  - Base tuning and data distribution decisions on performance metrics

Post-mortem Analysis

HPC.NRW

- Complex performance metrics are computed after measurement
  - Measurement is perturbed the least
- More complex metrics can be computed
  - Measurement will no longer be impaired by heavy-weight computation/communication
  - Multiple sources of data can be combined

The Measurement Environment (Application level)

HPC.NRW

- Data set
  - Representative of normal production runs
    - Different memory footprint may impact performance
  - Restrict runtime (5-15 minutes)
    - Should expose behavior
    - Should be repeated multiple times
  - Integrate data verification
    - Performance tools trigger hidden bugs

- Integrate a coarse-grained time measurement to your application
  - Helps to estimate the runtime dilation during a performance measurement run
  - High-level OpenMP, MPI, or UNIX time measurement suffices

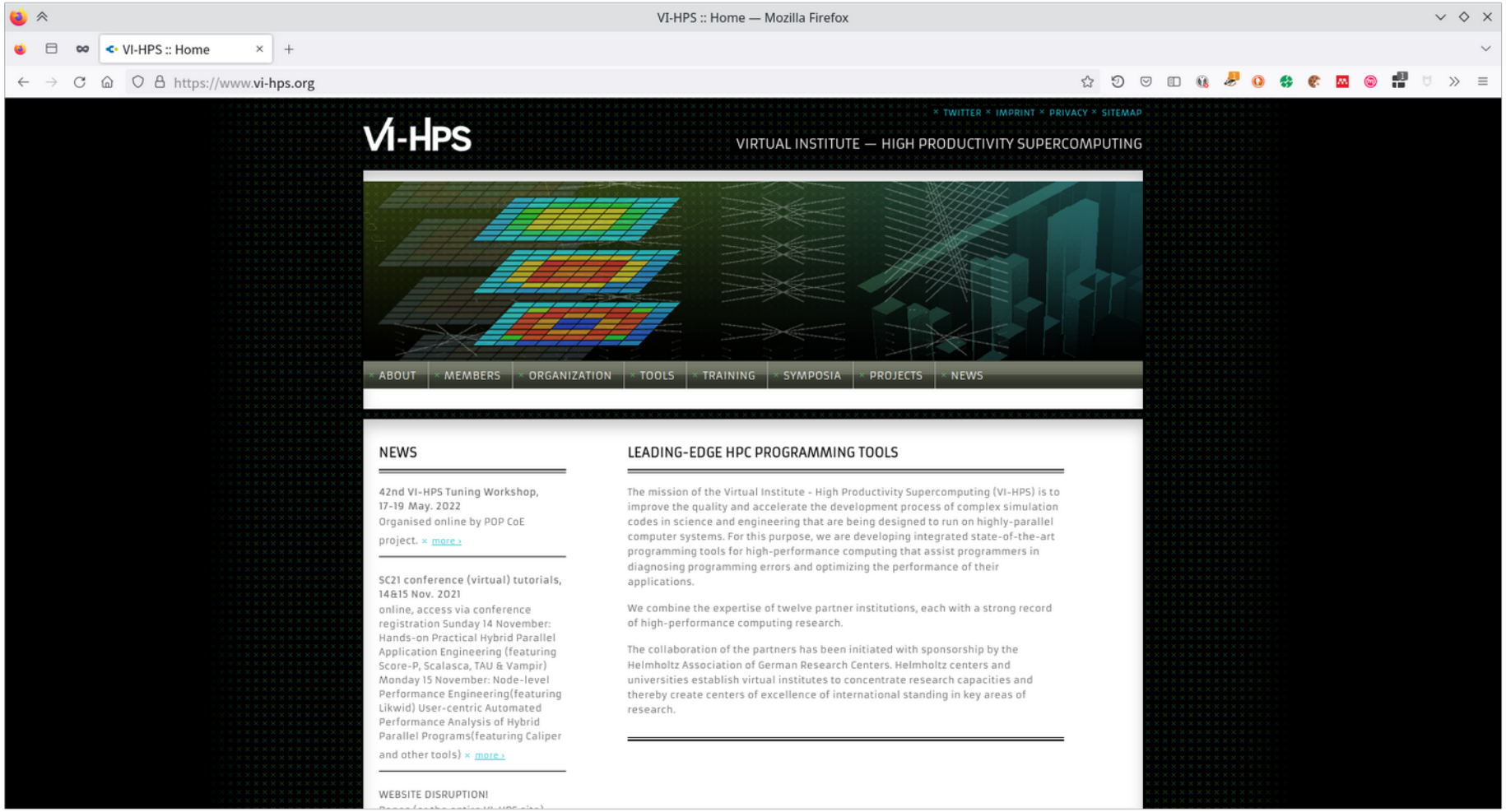
The Measurement Environment (System Level)

HPC.NRW

- Establish a “stable” testing environment to get repeatable performance results
  - Use exclusive nodes (minimize outside influence)
  - To test noise resiliency use deterministic noise sources/generators
  - Use thread binding
- Repeat application runs to eliminate outlier behavior (if possible)
  - Use appropriate statistical data analysis of performance results
    - E.g., mean, standard deviation, significance
- Automate measurement and analysis steps as much as possible
  - Use a workflow framework such as JUBE (https://www.fz-juelich.de/jsc/jube/)
    - Setup costs of integrating your application in a JUBE workflow will amortize over time

Virtual Institute – High Prodictivity Supercomputing (VI-HPS)

HPC.NRW



Using ARM Performance Reports

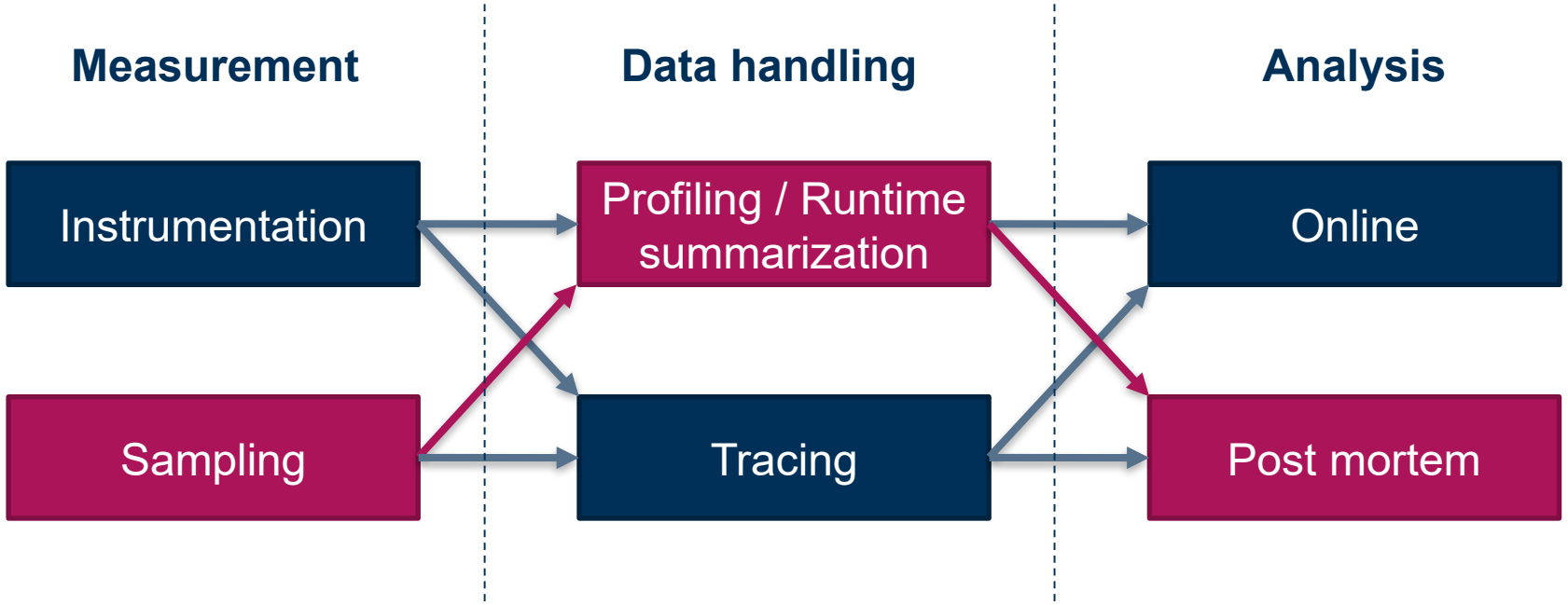
HPC.NRW Competence Network

Performance Analysis 101

ARM Performance Reports in the Tool Classification

HPC.NRW

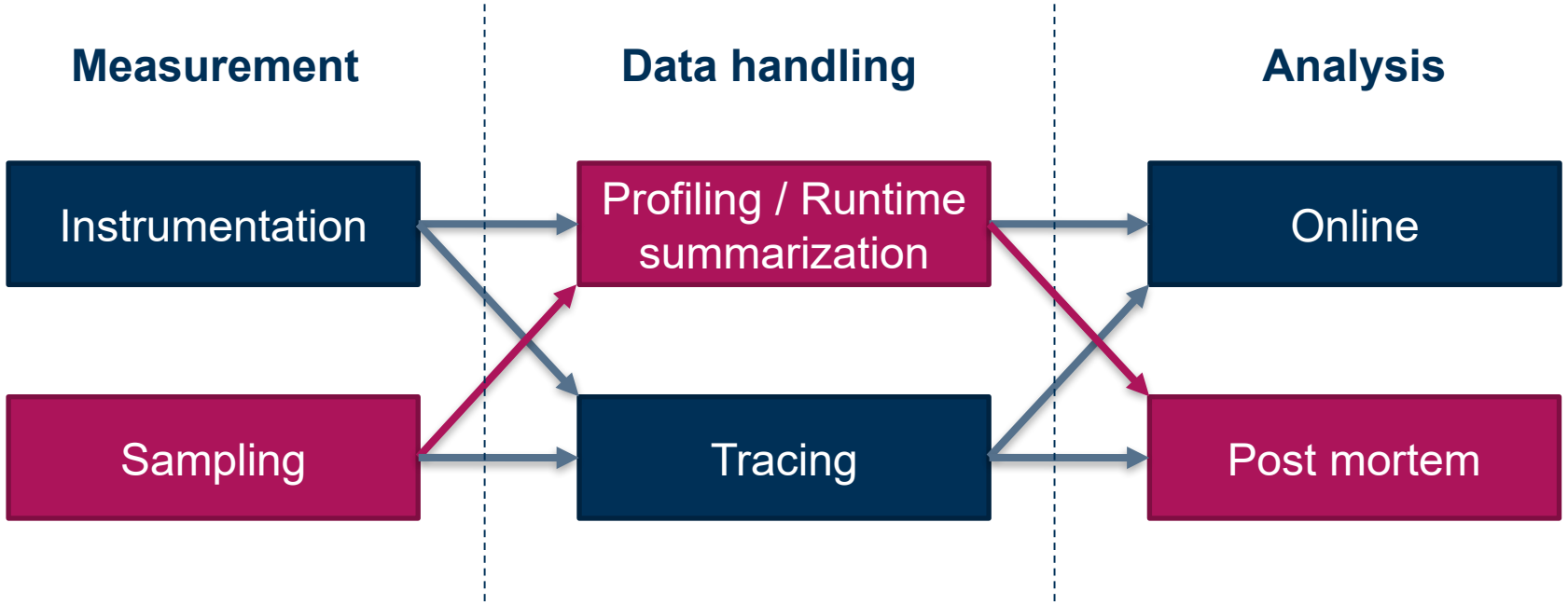
- ARM Performance Reports is a sampling profiler that writes a text/HTML report after measurement



ARM Performance Reports in the Tool Classification

HPC.NRW

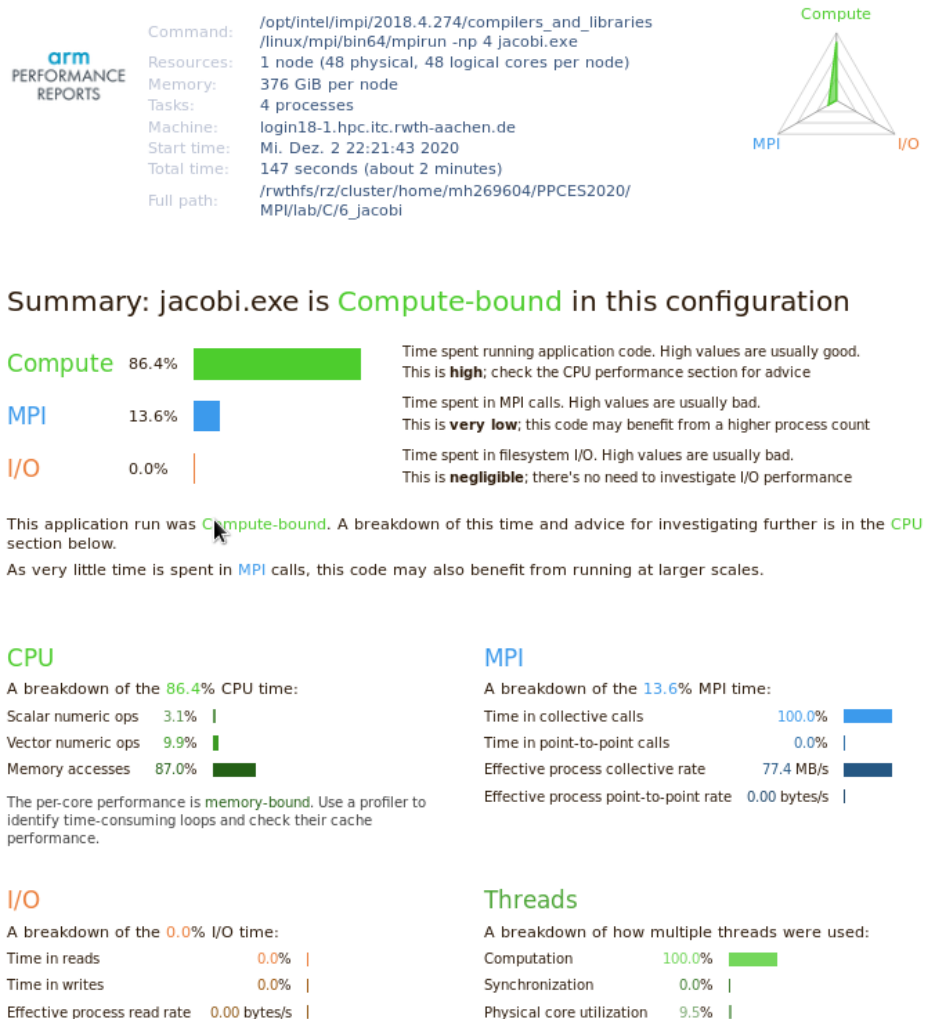
- ARM Performance Reports is a sampling profiler that writes a text/HTML report after measurement



ARM Performance Reports – Overview

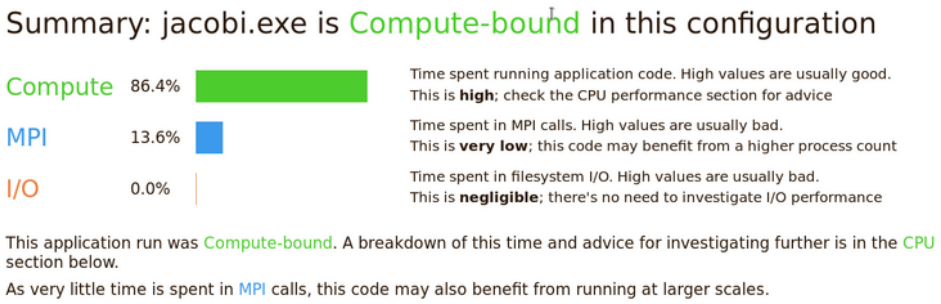
HPC.NRW

- General overview of application behavior
  - Suitable for initial performance screening
- Breakdown of behavior in different categories
- General advice on how to improve performance
  - Provides human readable, high-level information
  - May need experience to derive actions



ARM Performance Reports – Summary

- Identify where most wallclock time was spent
  - Computation
  - Communication (MPI)
  - I/O
- Explains high/low values and guides conclusions
- **Note:** Compute-bound here just means most time is spent with CPU related tasks
  - Does **NOT** refer to "*compute-bound* vs. *memory-bound*"



ARM Performance Reports – Summary

HPC.NRW

- How well is the code vectorized?
  - Scalar vs. Vector ops
- Is it computing or waiting for data?
  - Compute-bound vs. Memory-bound
- Information on vectorization of your code
  - High vectorization needed for peak performance
  - Not all codes can be vectorized!

CPU

A breakdown of the 86.4% CPU time:

Scalar numeric ops	3.1%
Vector numeric ops	9.9%
Memory accesses	87.0%

The per-core performance is memory-bound. Use a profiler to identify time-consuming loops and check their cache performance.

ARM Performance Reports – Summary

HPC.NRW

- Check for excessive communication time
  - Depends on application
    - Below 20% considered OK here
- Low transfer rates may indicate …
  - Waiting time
  - Communication overhead on small messages

MPI

A breakdown of the 13.6% MPI time:

Time in collective calls	100.0%
Time in point-to-point calls	0.0%
Effective process collective rate	77.4 MB/s
Effective process point-to-point rate	0.00 bytes/s

ARM Performance Reports – Summary

HPC.NRW

- Verify memory consumption
  - Overall consumption
- Mean vs. Peak
  - Are some processes using more than the rest?
- How much headroom is available?
  - Am I using the right scale for my dataset?
  - Estimate feasibility of tracing

Memory

Per-process memory usage may also affect scaling:

Mean process memory usage	1.55 GiB
Peak process memory usage	4.54 GiB
Peak node memory usage	28.0%

There is significant variation between peak and mean memory usage. This may be a sign of workload imbalance or a memory leak.

The peak node memory usage is very low. Running with fewer MPI processes and more data on each process may be more efficient.

ARM Performance Reports – Threads / OpenMP

HPC.NRW

- Name depending on detected threading model
- Computation
  - Time threads spent actually computing
- Synchronization
  - Time spent in locks and synchronization primitives
- Physical core utilization
  - Values > 100% indicates active hyper-threading
- System load
  - Values < 100% indicate unused resources

Threads

A breakdown of how multiple threads were used:

Computation	100.0%
Synchronization	0.0%
Physical core utilization	9.5%
System load	16.5%

Physical core utilization is low. Try increasing the number of threads or processes to improve performance.

ARM Performance Reports – Summary

HPC.NRW

- Check for excessive I/O time
  - Serialized I/O can induce a lot of waiting time
- Low transfer rates may indicate …
  - High meta data server load
    - HPC filesystems not build for many small ops
  - Filesystem contention
    - Writes to same block from multiple processes

I/O

A breakdown of the 0.0% I/O time:

Time in reads	0.0%
Time in writes	0.0%
Effective process read rate	0.00 bytes/s
Effective process write rate	0.00 bytes/s

No time is spent in I/O operations. There's nothing to optimize here!

ARM Performance Reports Summary

HPC.NRW

- Sampling profiler
  - Easy initial performance investigation
  - No special preparation of application needed
- Provides high-level information and advice
  - Key factors covered (Computation, MPI, Threading, IO, Memory)
- May serve for initial performance screening
  - Use report to decide which other tools to employ next