Introduction to Performance Engineering

Michael Gerndt
Technische Universität München

(with content used with permission from tutorials
by Bernd Mohr/JSC and Luiz DeRose/Cray)

Performance Analysis and Tuning is Essential

Performance factors of parallel applications

- “Sequential” 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” factors
  - Partitioning / decomposition
  - Communication (i.e., message passing)
  - Multithreading
  - Synchronization / locking
    - ☞More or less understood, good tool support

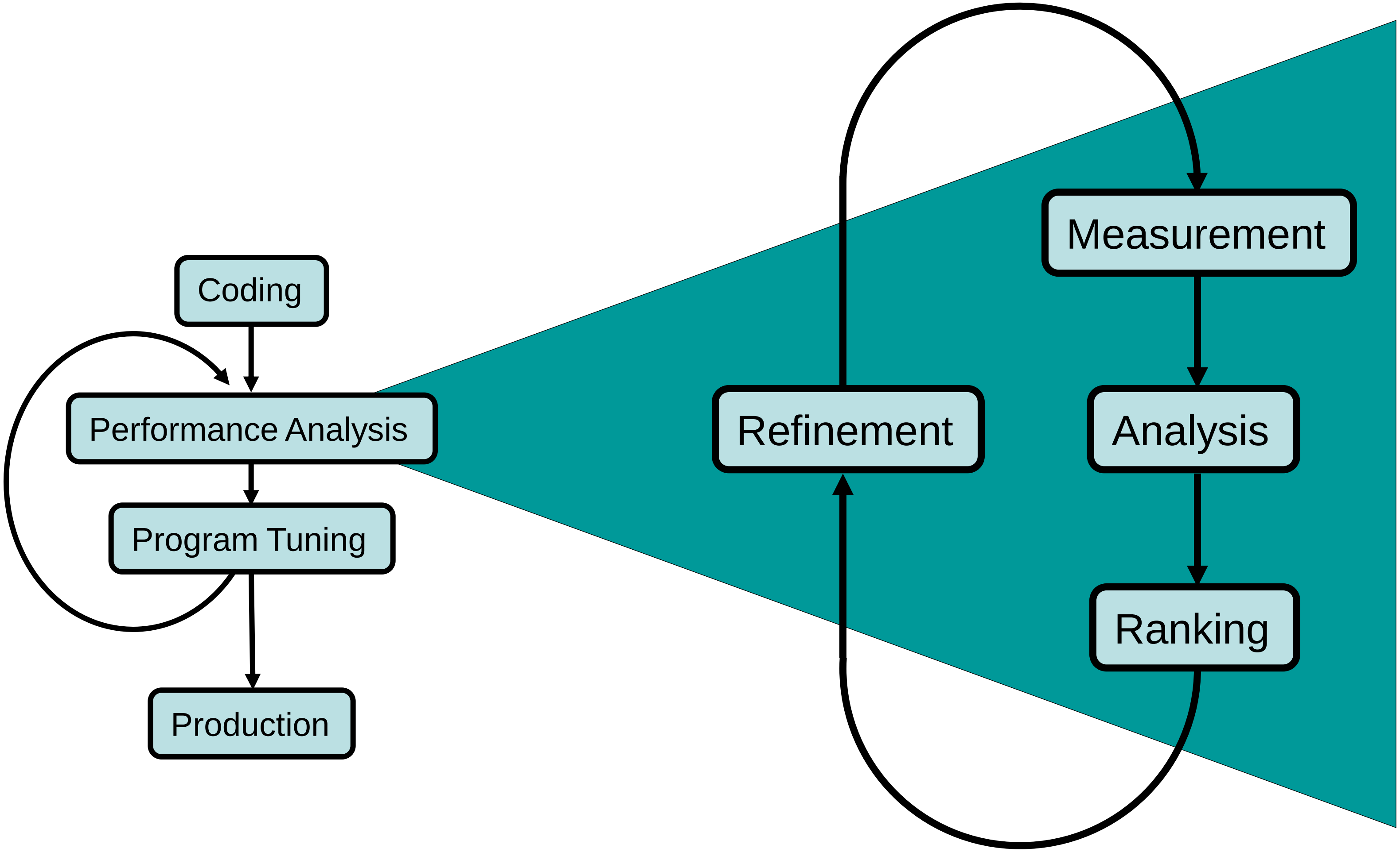
Tuning basics

- Successful engineering is a combination of
  - 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 Analysis Process



Metrics of performance

- 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 these send calls
  - The **size** of some parameter
    - E.g., the number of bytes transmitted by these calls

- Derived metrics
  - E.g., rates / throughput
  - Needed for normalization

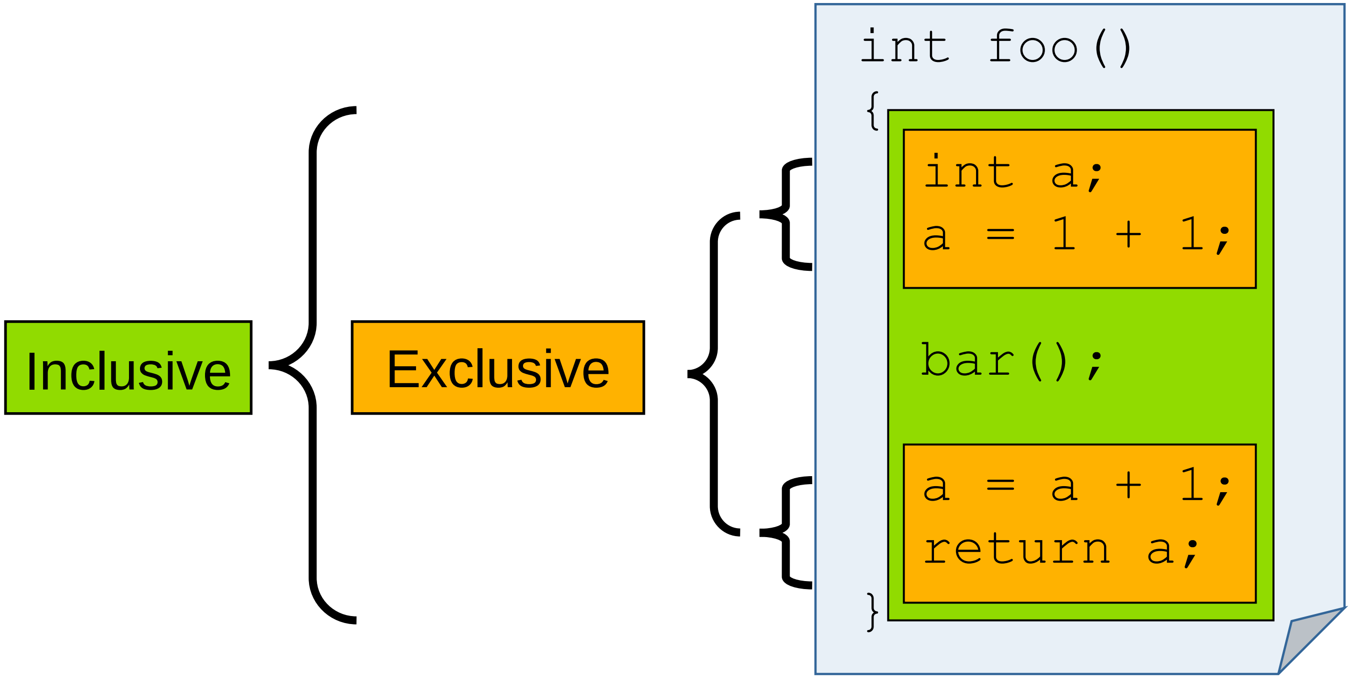
Example metrics

- Execution time
- Number of function calls
- CPI
  - CPU cycles per instruction
- FLOPS
  - Floating-point operations executed per second

“math” Operations?
HW Operations?
HW Instructions?
32-/64-bit? …

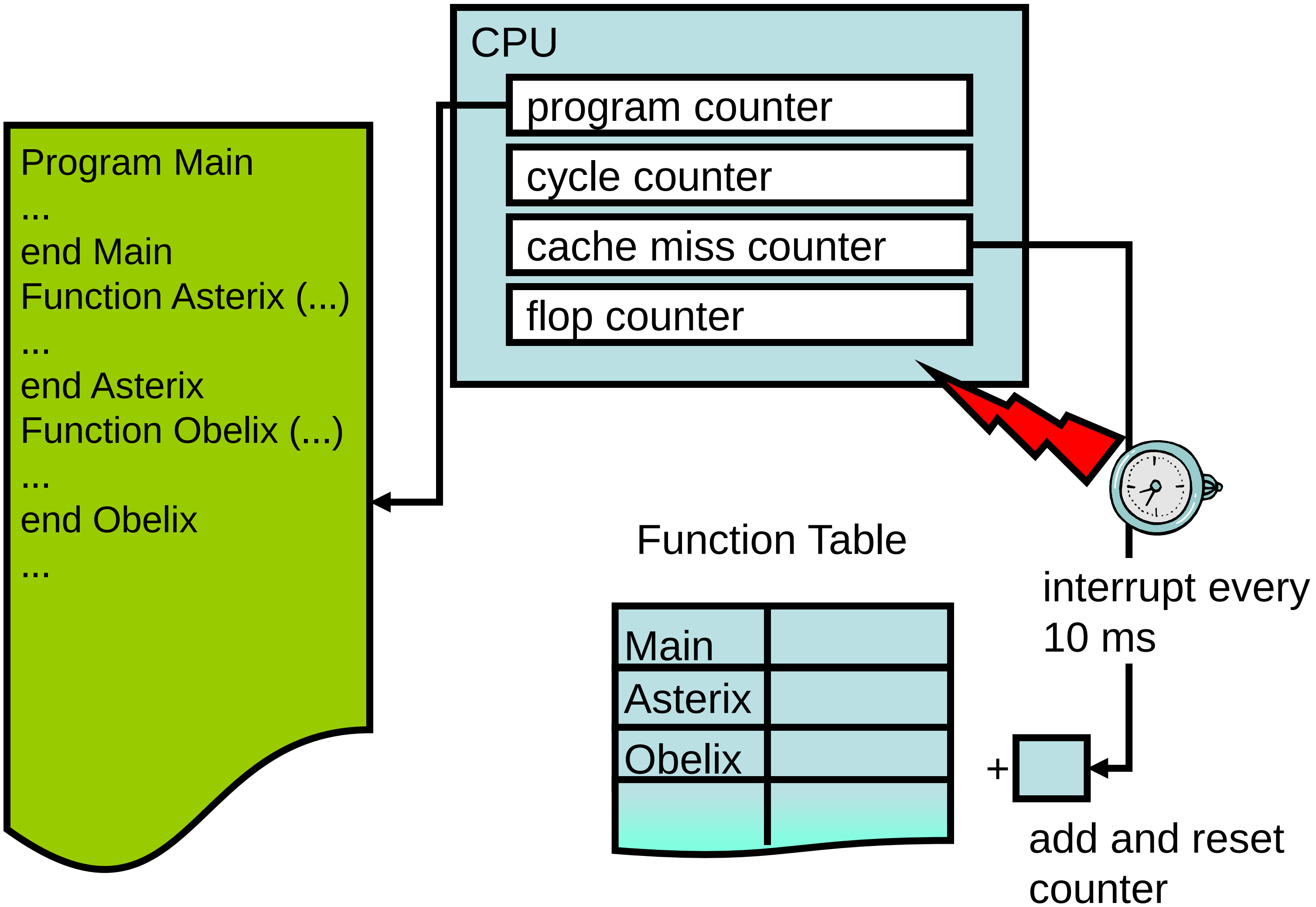
Inclusive vs. Exclusive values

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



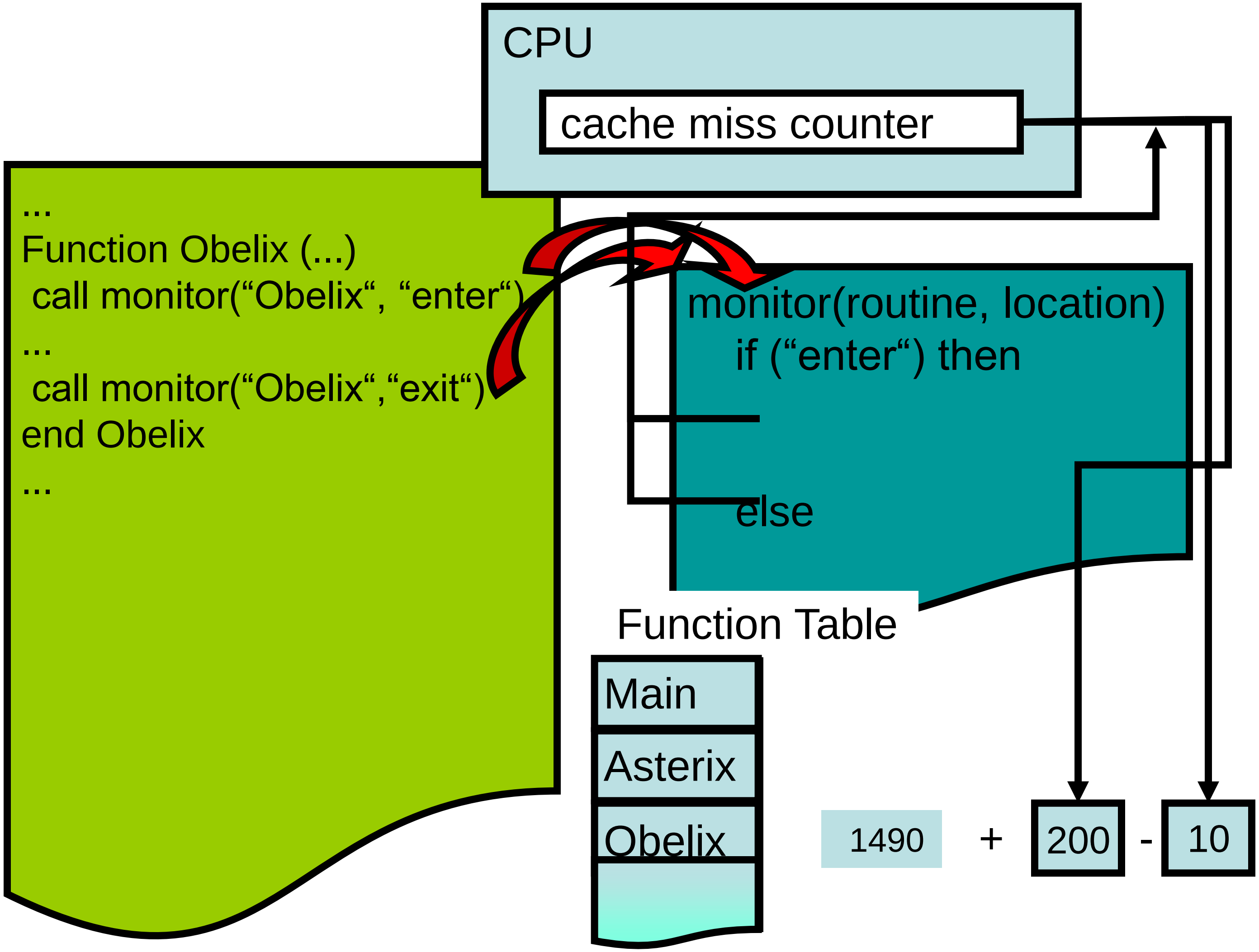
- Event model of the execution
  - Events occur at a processor at a specific point in time
  - Events belong to event types
    - clock cycles
    - cache misses
    - remote references
    - start of a send operation
    - ...
- Profiling: Recording accumulated performance data for events
  - Sampling: Statistical approach
  - Instrumentation: Precise measurement
- Tracing: Recording performance data of individual events

Sampling



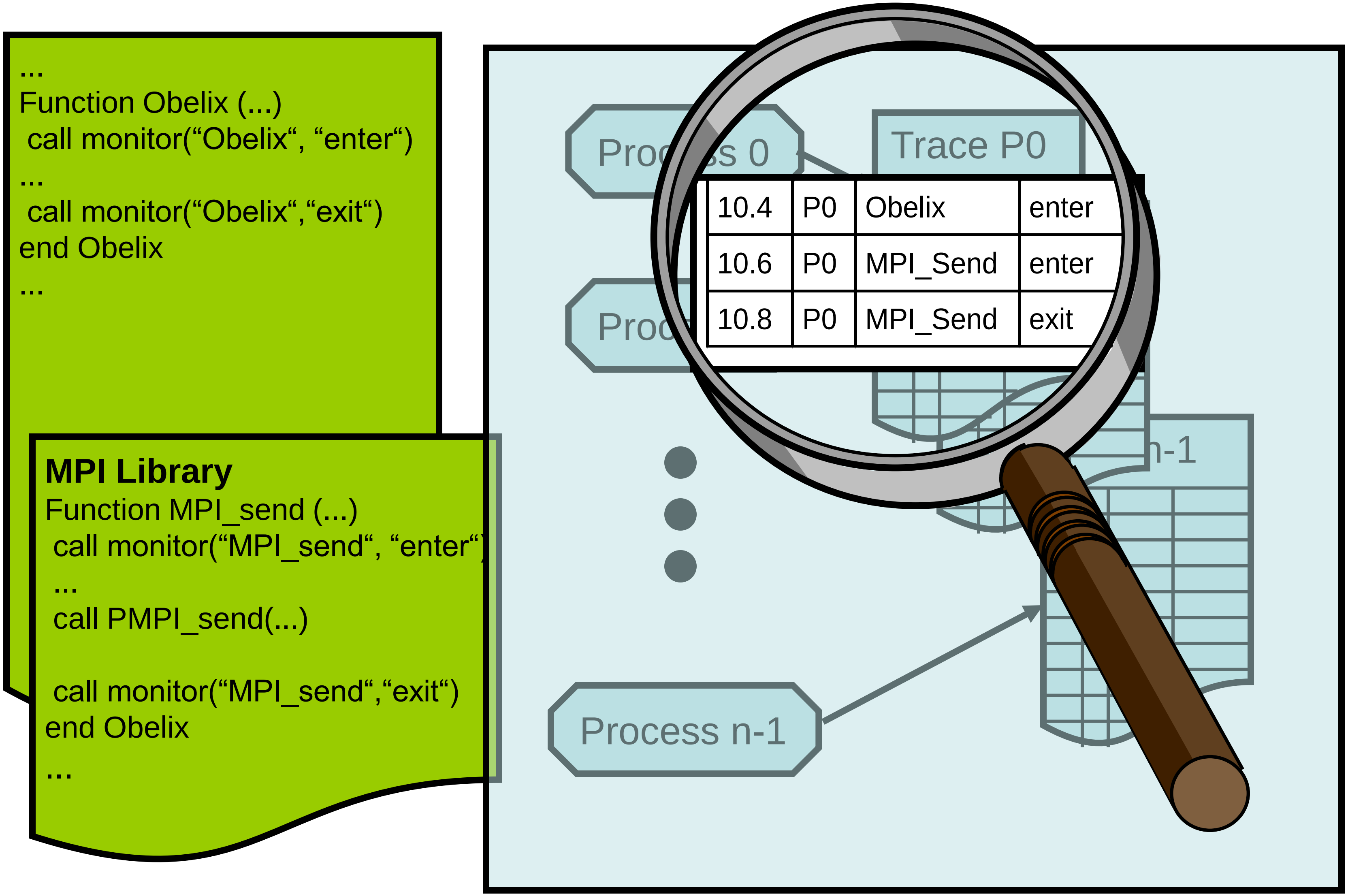
Program Main
...
end Main
Function Asterix (...)
...
end Asterix
Function Obelix (...)
...
end Obelix
...
CPU
program counter
cycle counter
cache miss counter
flop counter
Function Table
Main
Asterix
Obelix
interrupt every 10 ms
+
add and reset counter

Instrumentation and Monitoring

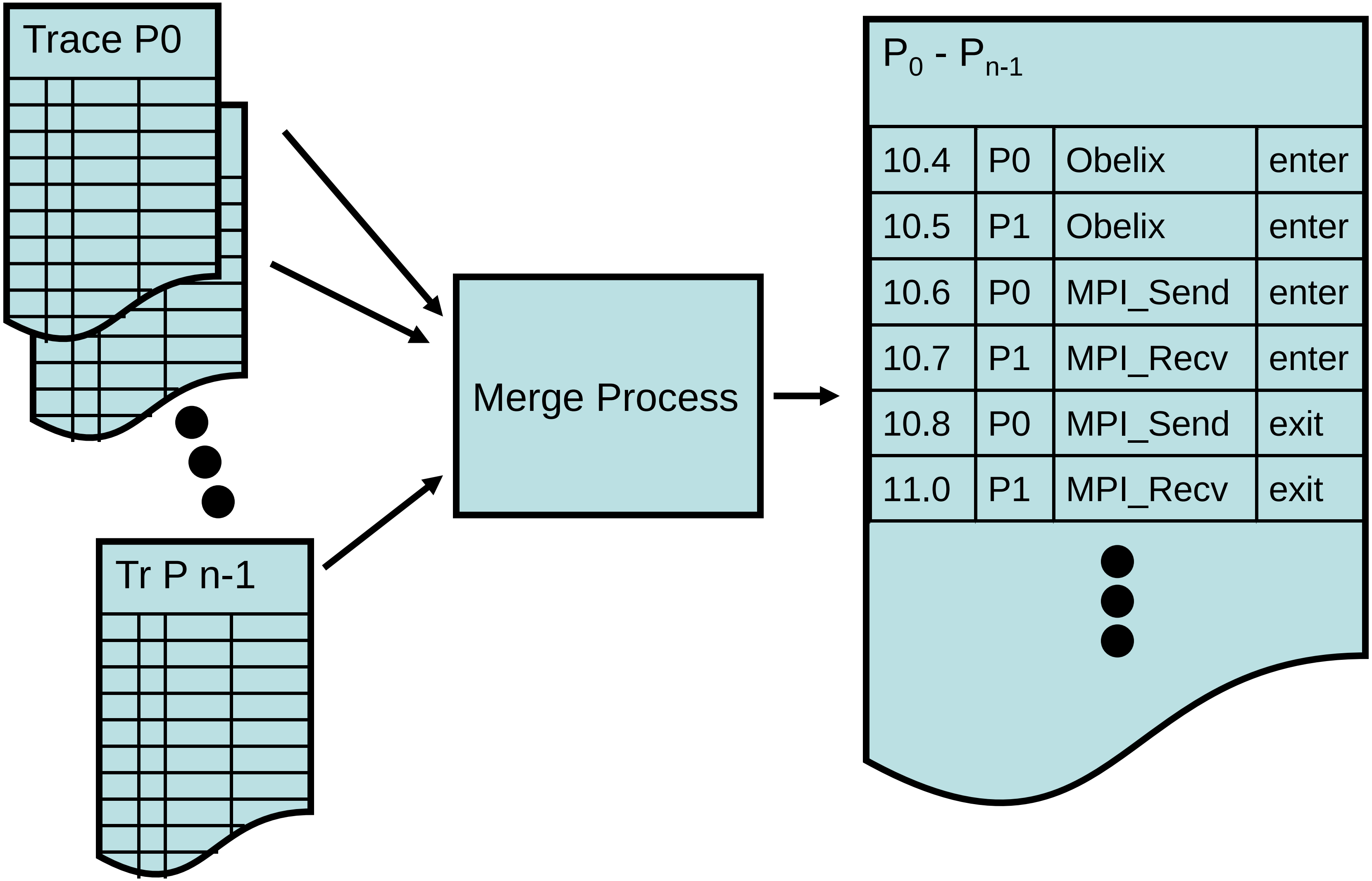


Instrumentation Techniques

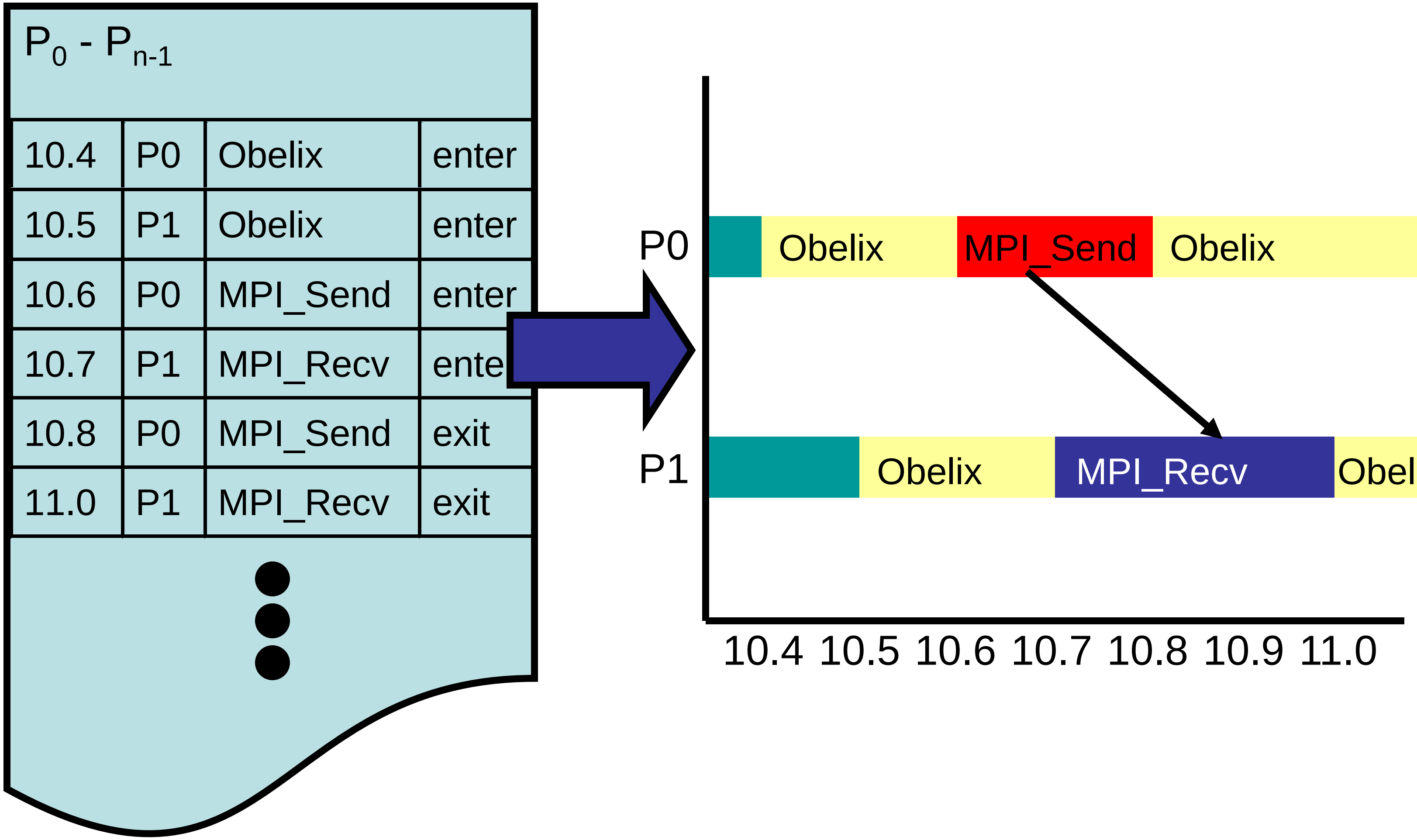
- Source code instrumentation
  - done by the compiler, source-to-source tool, or manually
    - + portability
    - + link back to source code easy
    - re-compile necessary when instrumentation is changed
    - difficult to instrument mixed-code applications
    - cannot instrument system or 3rd party libraries or executables
- Object code instrumentation
  - „patching“ the executable to insert hooks (like a debugger)
    - inverse pros/cons
  - Offline
  - Online


...
Function Obelix (...)
call monitor("Obelix", "enter")
...
call monitor("Obelix","exit")
end Obelix
...
MPI Library
Function MPI_send (...)
call monitor("MPI_send", "enter")
...
call PMPI_send(...)
call monitor("MPI_send","exit")
end Obelix
...
Trace P0
10.4 P0 Obelix enter
10.6 P0 MPI_Send enter
10.8 P0 MPI_Send exit
Process n-1

Merging


Trace P0
Tr P n-1
Merge Process
P0 - Pn-1
10.4 P0 Obelix enter
10.5 P1 Obelix enter
10.6 P0 MPI_Send enter
10.7 P1 MPI_Recv enter
10.8 P0 MPI_Send exit
11.0 P1 MPI_Recv exit

Visualization of Dynamic Behaviour



Profiling vs Tracing

- Profiling
  - recording summary information (time, #calls,#misses...)
  - about program entities (functions, objects, basic blocks)
  - very good for quick, low cost overview
  - points out potential bottlenecks
  - implemented through sampling or instrumentation
  - moderate amount of performance data
- Tracing
  - recording information about events
  - trace record typically consists of timestamp, processid, ...
  - output is a trace file with trace records sorted by time
  - can be used to reconstruct the dynamic behavior
  - creates huge amounts of data
  - needs selective instrumentation

Common Performance Problems with MPI

- Single node performance
  - Excessive number of 3rd-level cache misses
  - Low number of issued instructions
- IO
  - High data volume
  - Sequential IO due to IO subsystem or sequentialization in the program
- Excessive communication
  - Frequent communication
  - High data volume

Common Performance Problems with MPI

- Frequent synchronization
  - Reduction operations
  - Barrier operations
- Load balancing
  - Wrong data decomposition
  - Dynamically changing load

Common Performance Problems with SM

- Single node performance
  - ...
- IO
  - ...
- Excessive communication
  - Large number of remote memory accesses
  - False sharing
  - False data mapping
- Frequent synchronization
  - Implicit synchronization of parallel constructs
  - Barriers, locks, ...
- Load balancing
  - Uneven scheduling of parallel loops
  - Uneven work in parallel sections

Analysis Techniques

- Offline vs Online Analysis
  - Offline: first generate data then analyse
  - Online: generate and analyze data while application is running
  - Online requires automation➔limited to standard bottlenecks
  - Offline suffers more from size of measurement information
- Three techniques to support user in analysis
  - Source-level presentation of performance data
  - Graphical visualization
  - Ranking of high-level performance properties

No single solution is sufficient!

☞ *A combination of different methods, tools and techniques is typically needed!*

- Analysis
  - Statistics, visualization, automatic analysis, data mining, ...
- Measurement
  - Sampling / instrumentation, profiling / tracing, ...
- Instrumentation
  - Source code / binary, manual / automatic, ...

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