Performance Analysis Tools

Prof. Dr. Michael Gerndt
Lehrstuhl für Rechnerarchitektur und
Parallele Systeme

Speedup Limited by Overheads

$$\text{Speedup} \leq \frac{\text{Sequential Work}}{\textit{Max}\ (\text{Work} + \text{Synch Time} + \text{Comm Cost} + \text{Extra Work})}$$

Load Balance

- Limit on speedup:
  - Work includes data access and other costs
  - Not just equal work, but must be busy at same time
- Four parts to load balance
  1. Identify enough concurrency
  2. Decide how to manage it
  3. Determine the granularity at which to exploit it
  4. Reduce serialization

$$\text{Speedup}(p) \leq \frac{\text{Sequential Work}}{\text{Max Work on any Processor}}$$

Reducing Synch Time

- Reduce wait time due to load imbalance
- Reduce synchronization overhead

Reducing Synchronization Overhead

- Event synchronization
  - Reduce use of conservative synchronization
    - e.g. point-to-point instead of barriers, or granularity of pt-to-pt
  - But fine-grained synch more difficult to program, more synch ops.
- Mutual exclusion
  - Separate locks for separate data
    - e.g. locking records in a database: lock per process, record, or field
    - lock per task in task queue, not per queue
    - finer grain => less contention/serialization, more space, less reuse
  - Smaller, less frequent critical sections
    - don't do reading/testing in critical section, only modification
    - e.g. searching for task to dequeue in task queue, building tree

Reducing Inherent Communication

- Communication is expensive!
- Measure: communication to computation ratio
- Focus here on inherent communication
  - Determined by assignment of tasks to processes
  - Actual communication can be greater
- Assign tasks that access same data to same process
- Hide communication time
- But simple heuristic solutions work well in practice
  - Applications have structure!

$$\text{Speedup} \leq \frac{\text{Sequential Work}}{\textit{Max}\ (\text{Work} + \text{Synch Time} + \text{Comm Cost})}$$

Reducing Extra Work

- Common sources of extra work:
  - Computing a good partition
  - Using redundant computation to avoid communication
  - Task, data and process management overhead
    - applications, languages, runtime systems, OS
  - Imposing structure on communication
    - coalescing messages, allowing effective naming
- Architectural implications:
  - Reduce need by making communication and orchestration efficient

$$\text{Speedup} \leq \frac{\text{Sequential Work}}{\textit{Max}\ (\text{Work} + \text{Synch Wait Time} + \text{Comm Cost} + \text{Extra Work})}$$

A Lot Depends on Sizes

- Application parameters and no. of procs affect inherent properties
  - Load balance, communication, extra work, temporal and spatial locality
- Memory hierarchy
  - Interactions with organization parameters of extended memory hierarchy affect artifactual communication and performance
- Effects often dramatic, sometimes small: application-dependent

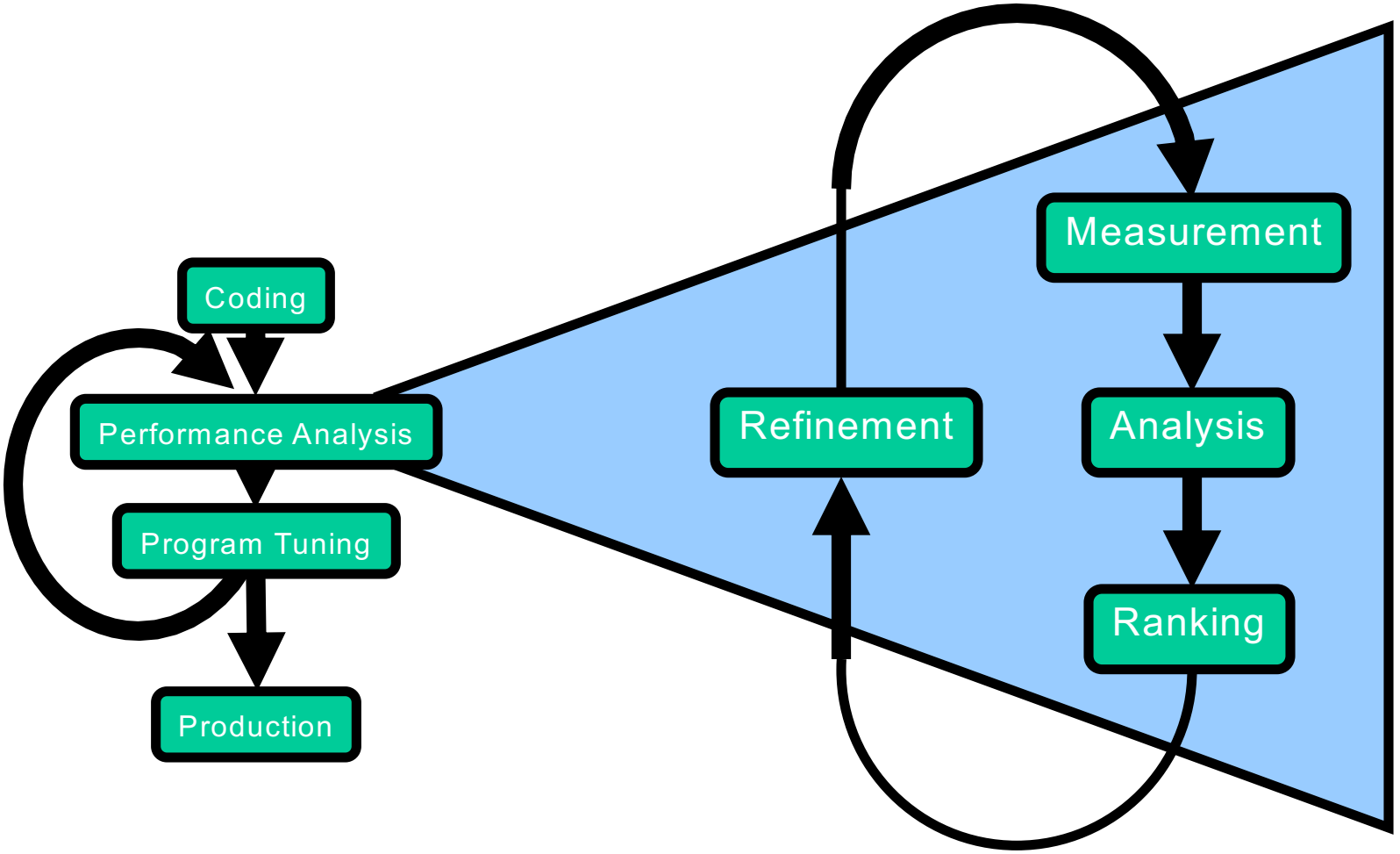
Scaling Down Problem Parameters

- Some parameters don't affect parallel performance much, but do affect runtime, and can be scaled down
  - Common example is no. of time-steps in many scientific applications
    - need a few to allow settling down, but don't need more
    - may need to omit cold-start when recording time and statistics
  - First look for such parameters

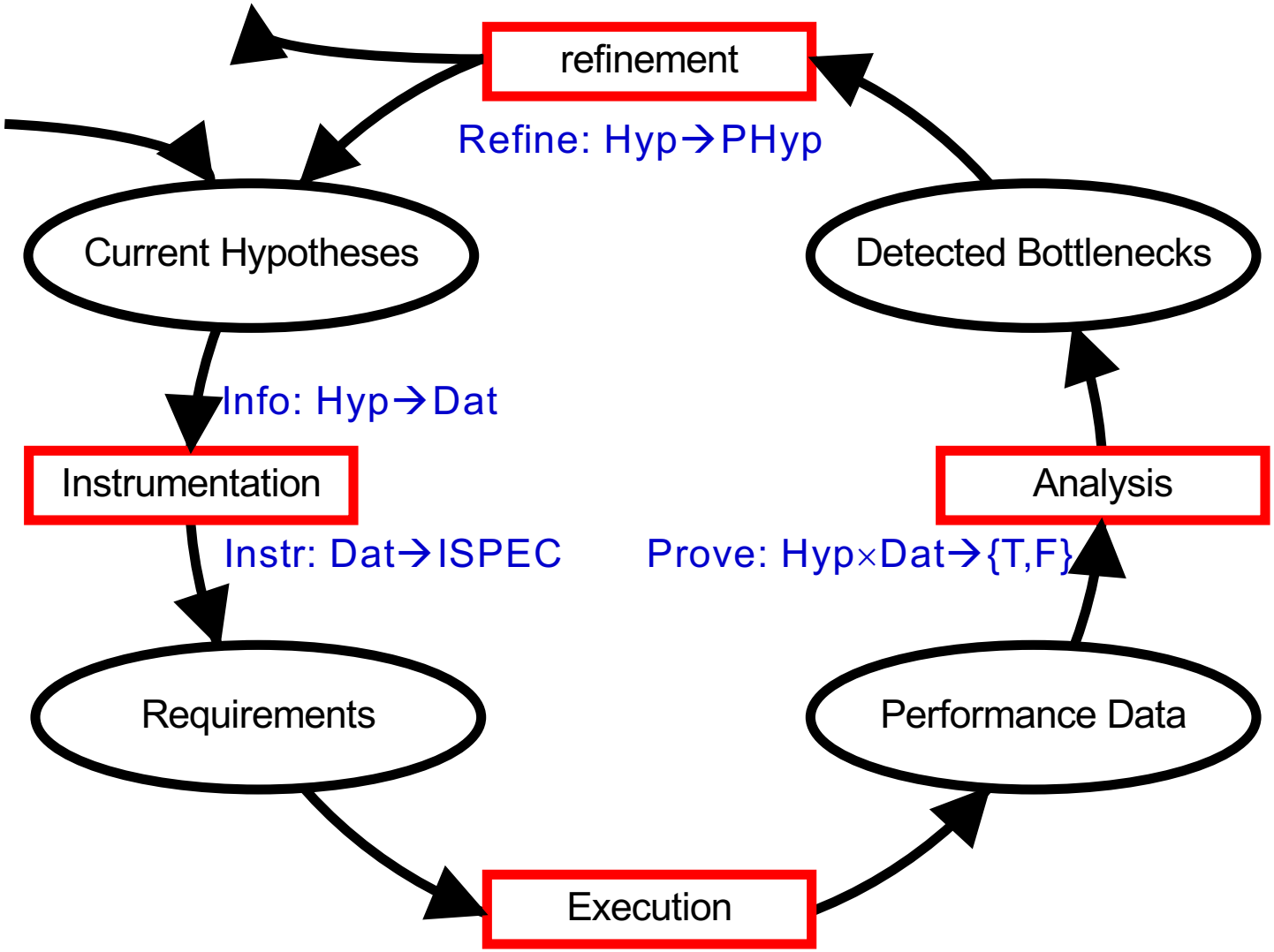
- But many application parameters affect key characteristics
  - Scaling them down requires scaling down no. of processors too
  - Otherwise can obtain highly unrepresentative behavior

Performance Analysis Tools

Performance Analysis Process



Performance Analysis



Performance Measurement Techniques

- 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
    - ...

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
- Binary instrumentation
  - „patching“ the executable to insert hooks (like a debugger)
    - inverse pros/cons
  - Offline
  - Online

Monitoring

- Profiling: Recording accumulated performance data for events
  - Sampling: Statistical approach
  - Instrumentation: Precise measurement
- Tracing: Recording performance data of individual events

Profiling

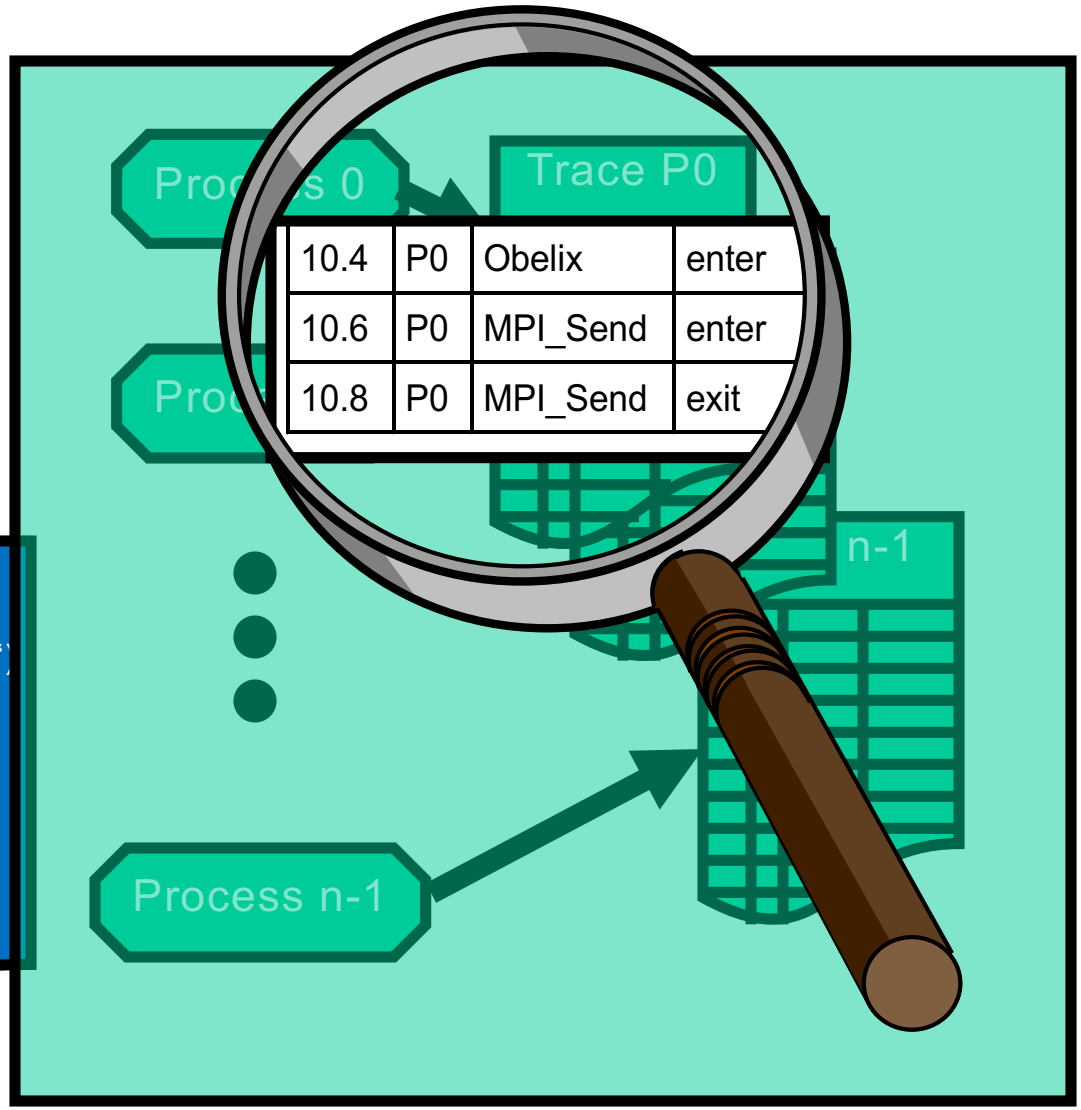
- Flat profile
  - List of function with summed up metrics
- Callpath profile
  - Tree of callpaths
  - Routines can appear multiple times
  - Summed up values per callpath

Tracing

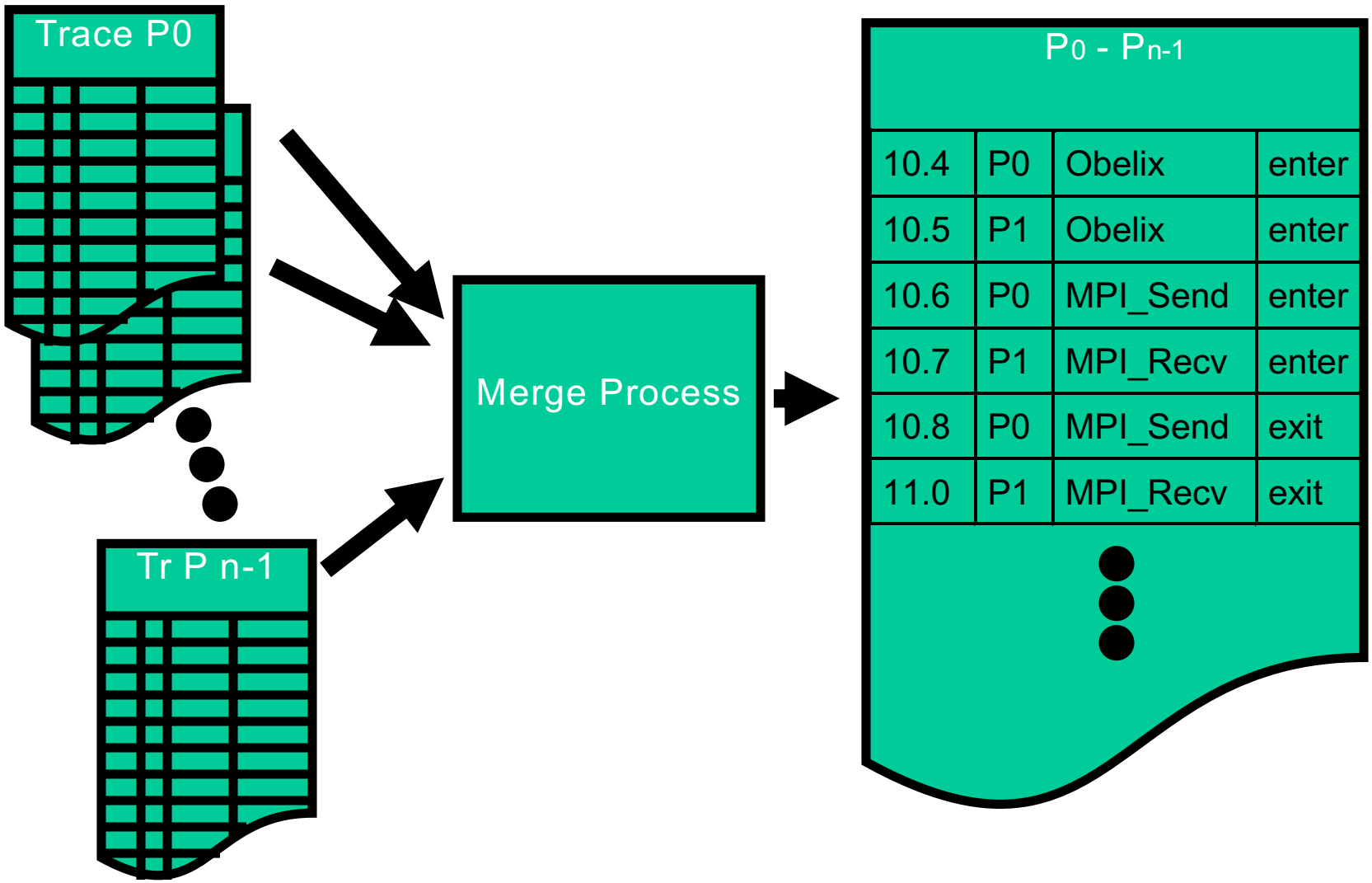
```
...
Function Obelix (...)
call monitor("Obelix", "enter")
...
call monitor("Obelix","exit")
end Obelix
...
```

MPI Library

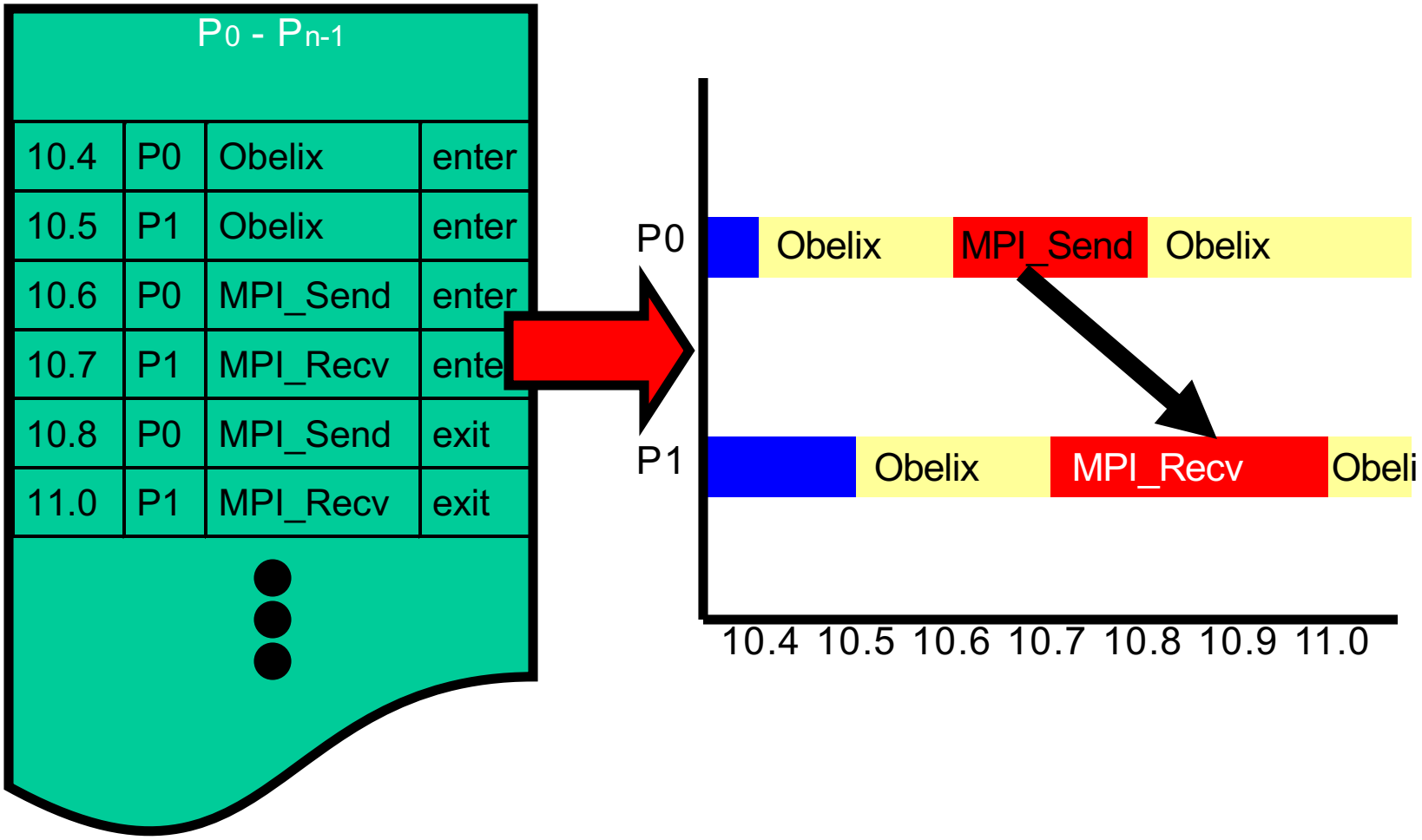
```
Function MPI_send (...)
call monitor("MPI_send", "enter")
...
call PMPI_send(...)

call monitor("MPI_send","exit")
end Obelix
...
```



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

Analysis Techniques

- Offline vs Online Analysis
  - Offline: first generate data then analyze
  - 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

Profiling based Tools

- Gprof – GNU profiling tool
  - Time profiling
  - Inclusive and exclusive time
  - Flat profile

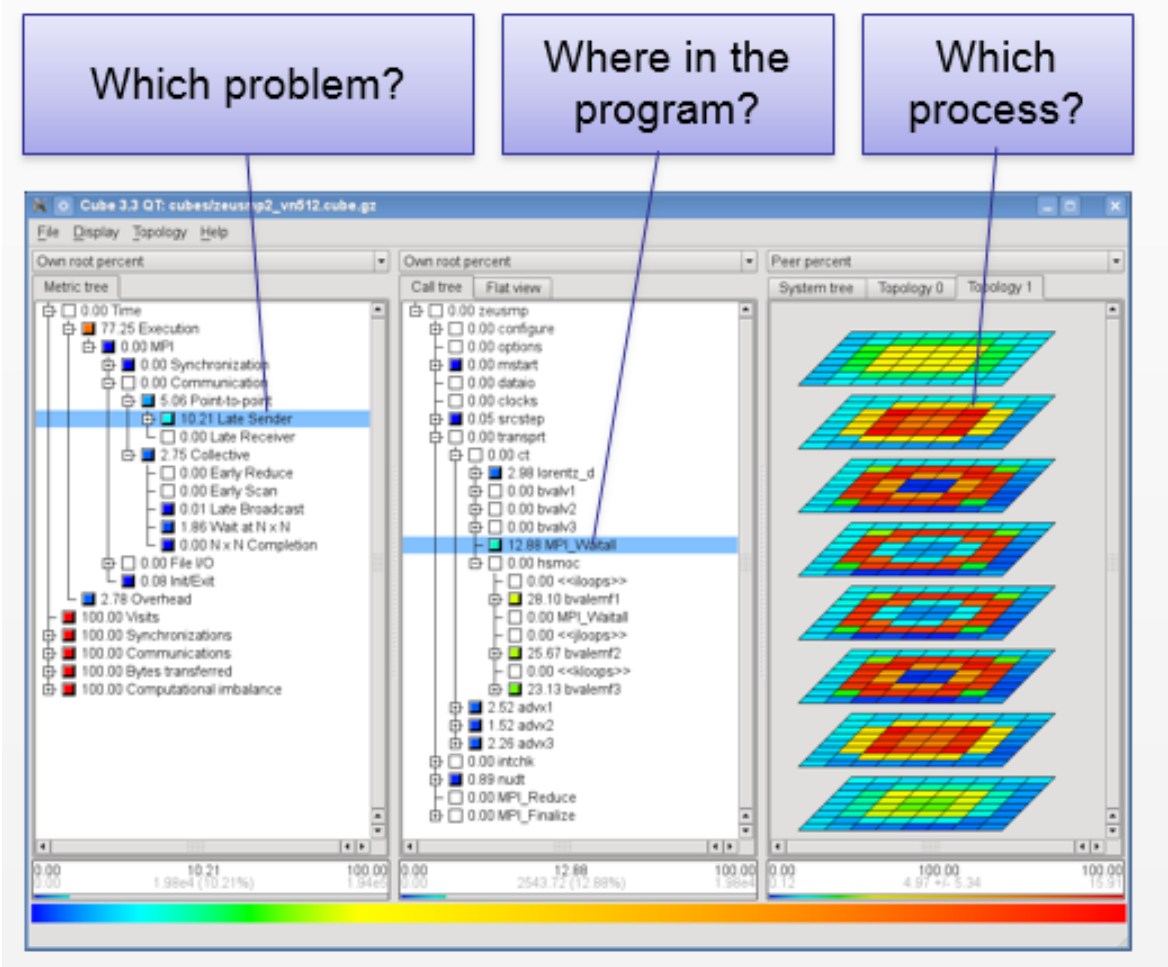
```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call  name
 33.34      0.02     0.02     7208     0.00     0.00  open
 16.67      0.03     0.01      244     0.04     0.12  offtime
 16.67      0.04     0.01        8     1.25     1.25  memccpy
 16.67      0.05     0.01        7     1.43     1.43  write
 16.67      0.06     0.01                             mcount
  0.00      0.06     0.00      236     0.00     0.00  tzset
  0.00      0.06     0.00      192     0.00     0.00  tolower
  0.00      0.06     0.00       47     0.00     0.00  strlen
  0.00      0.06     0.00       45     0.00     0.00  strchr
  0.00      0.06     0.00        1     0.00    50.00  main
  0.00      0.06     0.00        1     0.00     0.00  memcpy
  0.00      0.06     0.00        1     0.00    10.11  print
  0.00      0.06     0.00        1     0.00     0.00  profil
  0.00      0.06     0.00        1     0.00    50.00  report
```

- Call graph profile
- Based on instrumentation of function entry and exit
- Records were the call is coming from.

```
granularity: each sample hit covers 2 byte(s) for 20.00% of 0.05 seconds

index % time    self  children    called     name
                                                 <spontaneous>
[1]     100.0    0.00    0.05                 start [1]
                 0.00    0.05       1/1           main [2]
                 0.00    0.00       1/2           on_exit [28]
                 0.00    0.00       1/1           exit [59]
-----------------------------------------------
                 0.00    0.05       1/1           start [1]
[2]     100.0    0.00    0.05       1         main [2]
                 0.00    0.05       1/1           report [3]
-----------------------------------------------
                 0.00    0.05       1/1           main [2]
[3]     100.0    0.00    0.05       1         report [3]
                 0.00    0.03       8/8           timelocal [6]
                 0.00    0.01       1/1           print [9]
                 0.00    0.01       9/9           fgets [12]
                 0.00    0.00      12/34          strncmp <cycle 1> [40]
                 0.00    0.00       8/8           lookup [20]
                 0.00    0.00       1/1           fopen [21]
```

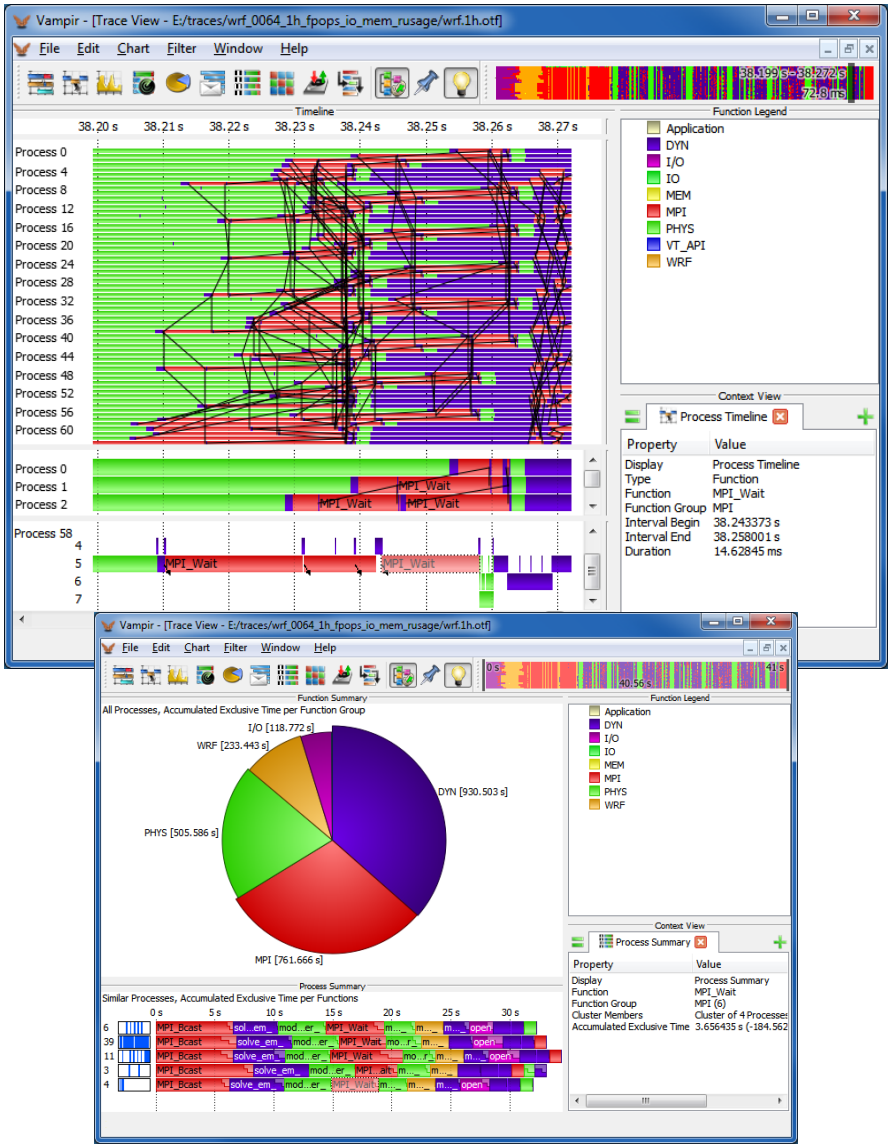
Profiling Tool based on Instrumentation

- Cube
  - Performance visualizer
  - Profiles based on Score-P
  - Call-path profiling



Trace-based Analysis Tools

- Vampir
  - Graphical views presenting events and summary data
  - Flexible scrolling and zooming features
  - OTF2 trace format generated by Score-P
  - Commercial license
  - www.vampire.eu

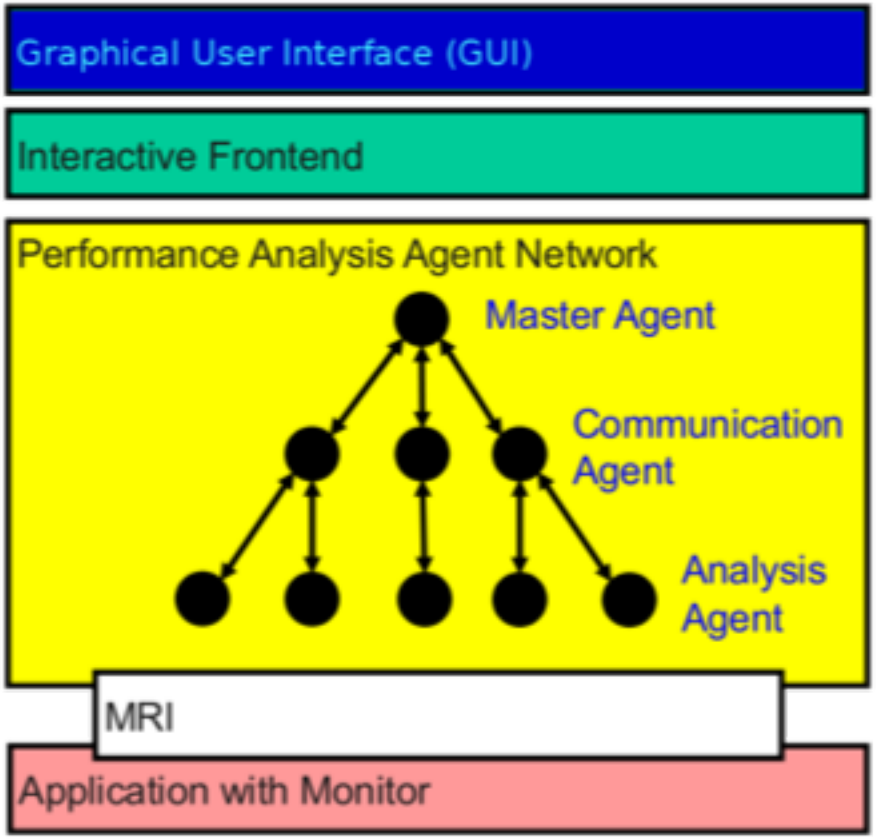


Automatic Analysis Tools

- Paradyn
  - University of Wisconsin Madison
- Periscope
  - TU München
  - Automatic detection of formalized performance properties
  - Profile data
  - Distributed online tool
- Scalasca
  - Search for performance patterns in traces
  - Post-mortem on parallel resources of the application
  - Visualization of patterns in CUBE

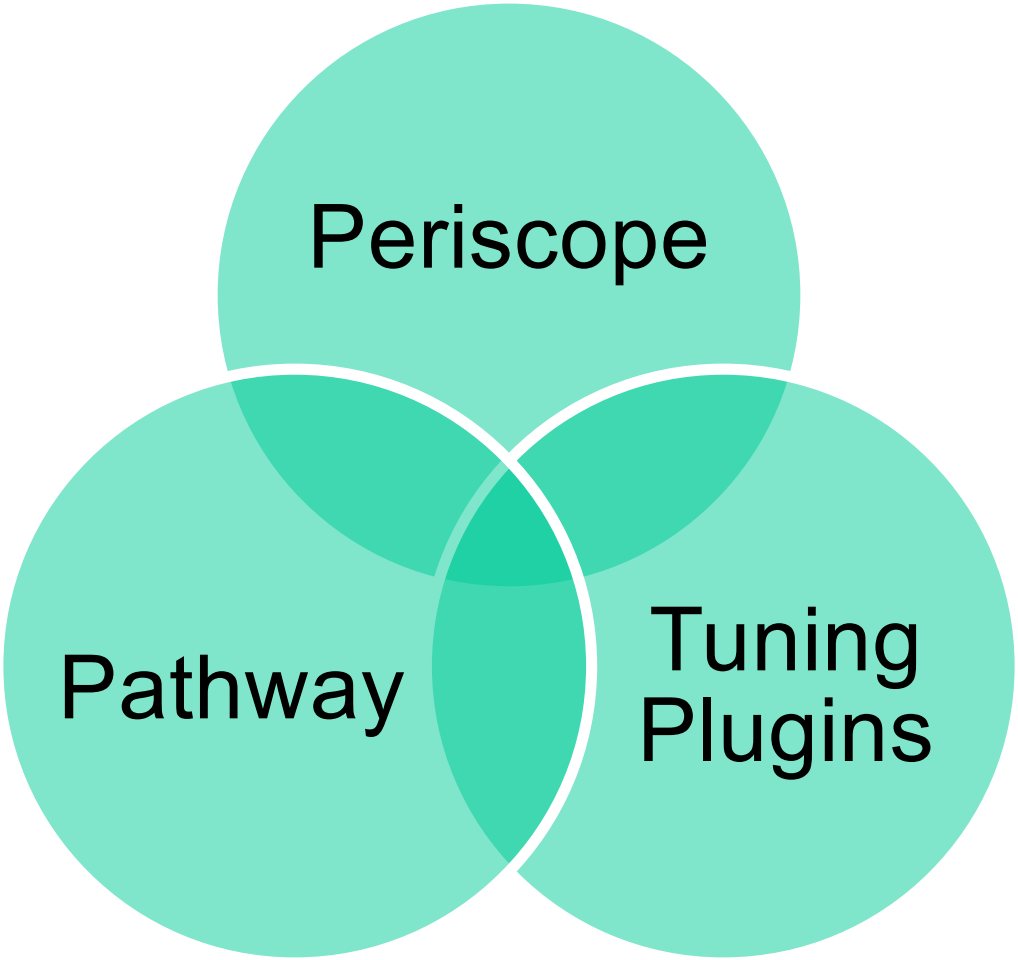
Periscope

- Online
  - no need to store trace files
- Distributed
  - reduced network utilization
- Scalable
  - Up to 100000s of CPUs
- Multi-scenario analysis
  - Single-node Performance
  - MPI Communication
  - OpenMP
- Portable
  - Fortran, C with MPI & OMP
  - Intel Itanium2, x86 based systems
  - IBM Power6, BlueGene P, Cray



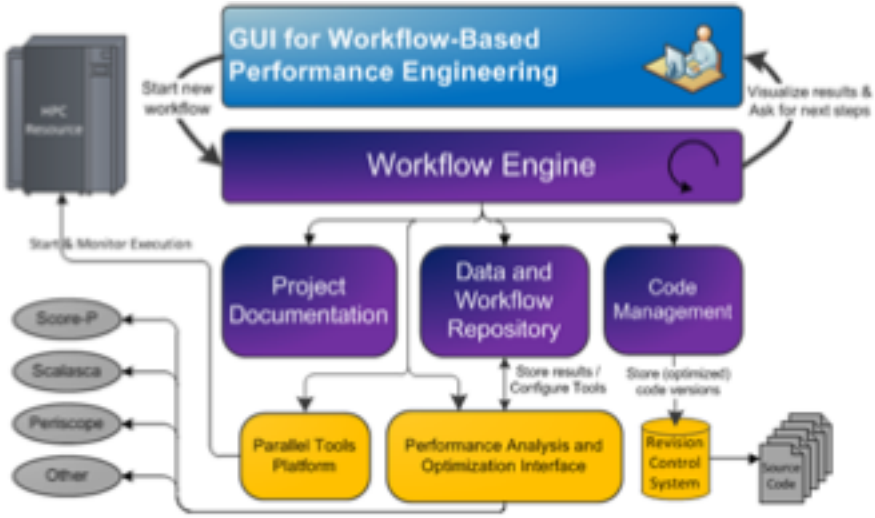
http://www.lrr.in.tum.de/periscope

Periscope Tuning Framework



Pathway

- Performance Engineering Workflows
  - Ph.D. thesis of Ventsislav Petkov (LMAC project)
- Formalization and automation
- Support for
  - Definition of workflows in Eclipse
  - Automatic/manual tasks
  - Transparent experiments
  - Experiment archive and documentation

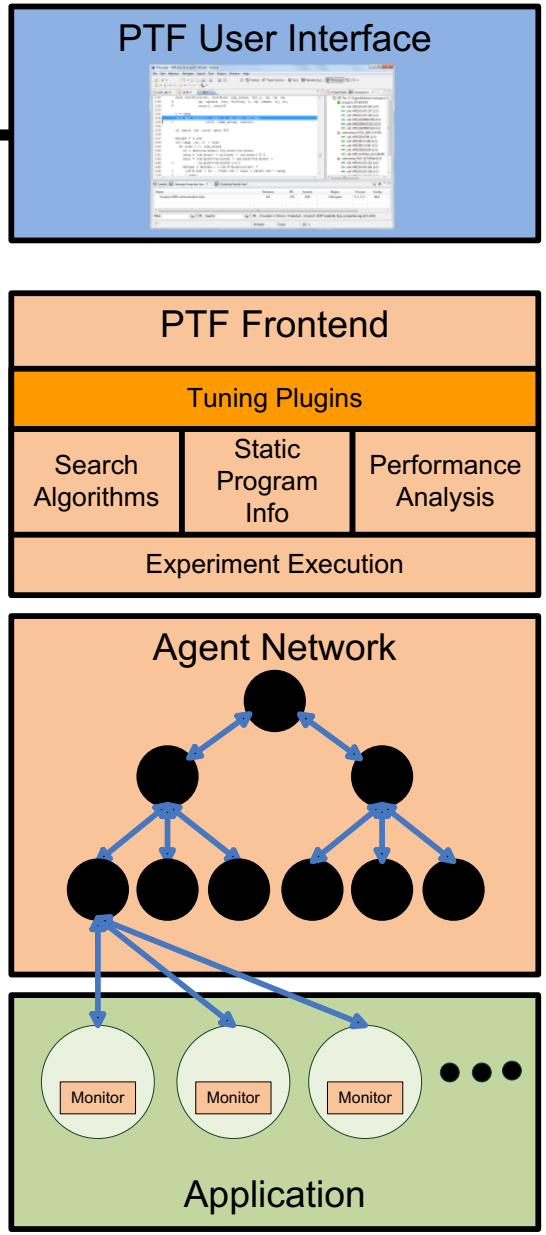


AutoTune Project

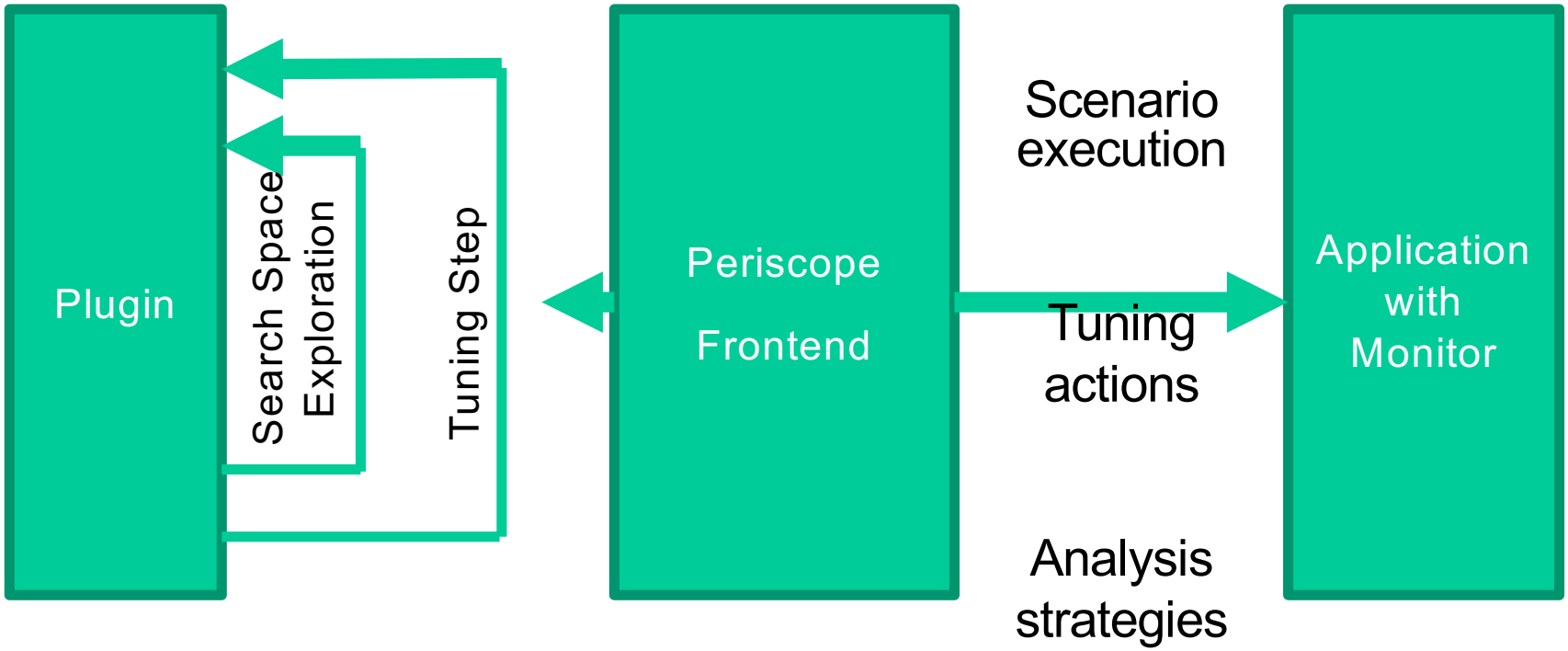
- Automatic tuning based on expert knowledge
- Beyond auto-tuning like Active Harmony
- Tuning plugins for Periscope
- Expert knowledge
  - When is it necessary to tune?
  - Where to tune?
  - What are the tuning parameters?
  - How to shrink the search space?

Periscope Tuning Framework

- 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



Tuning Plugin Interface



Tuning Plugins

- MPI parameters
  - Eager Limit, Buffer space, collective algorithms
  - Application restart or MPIT Tools Interface
- DVFS
  - Frequency tuning for energy delay product
  - Model-based prediction of frequency
  - Region level tuning
- Parallelism capping
  - Thread number tuning for energy delay product
  - Exhaustive and curve fitting based prediction

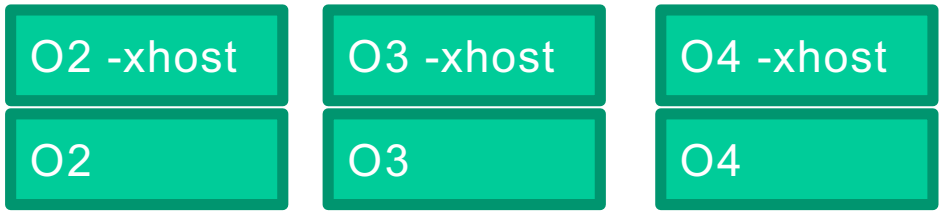
Search Algorithms

- Tuning Parameters

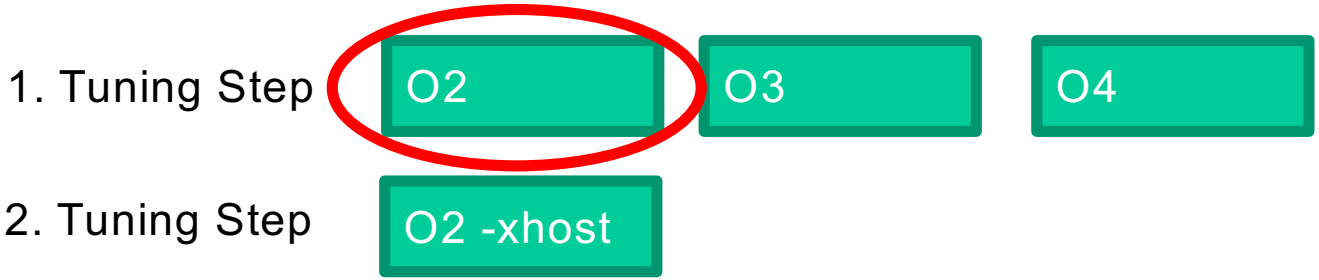
  TP1: O2, O3, O4 TP2: -xhost, none

- Exhaustive

Single tuning step



- Individual



Individual Tuning NPB (C)

							time		
	O	xhost	unroll	ip	ipo	opt-pref	min	max	incr. %
BT	O2	X		X	X	X	5,19	6,09	17
LU	O4	X	X		X		3,54	3,92	11
SP	O3				X		1,77	3,23	83
CG	O2			X			4,66	4,80	3
FT	O4	X				X	13,10	15,09	15
MG	O3						4,06	4,06	0
IS	O2	X					4,54	4,62	2

Performance Analysis

- Performance properties
- Performance analyis
- Sampling, instrumentation, profiling, tracing
- Manual and automatic analysis
- Periscope Tuning Framework