Universitat Autònoma de Barcelona

Performance analysis and tuning of parallel/distributed applications

Anna Morajko

Anna.Morajko@uab.es

26-05-2008

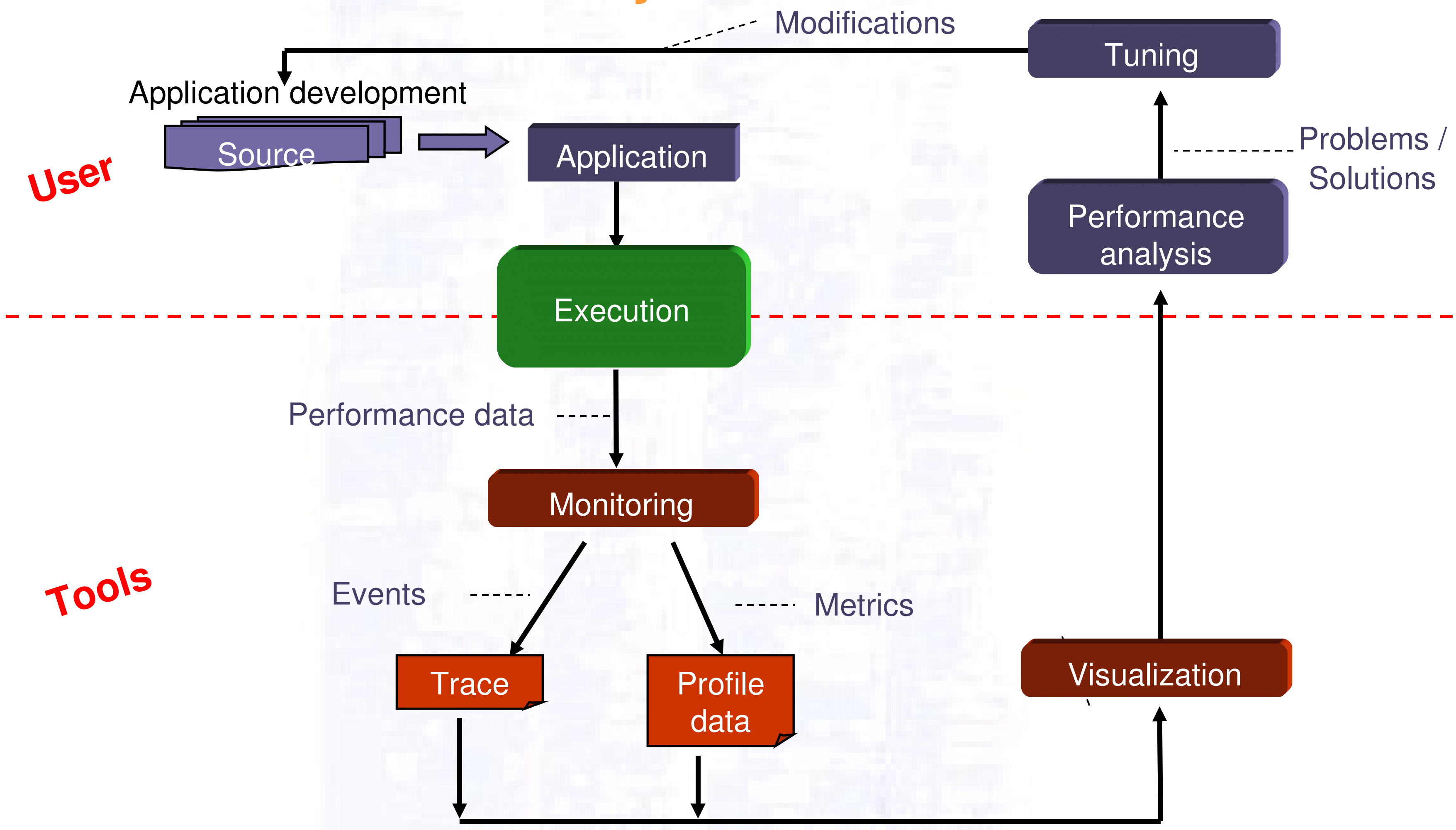
Introduction

Main research projects

- Develop techniques and tools for application performance improvement
- Monitoring, analysis, and tuning tools:
  - Kappa-Pi – post-mortem analysis
  - DMA – automatic analysis at run-time
  - MATE – automatic tuning at run-time
- Performance models
  - For applications based on frameworks
  - For scientific libraries

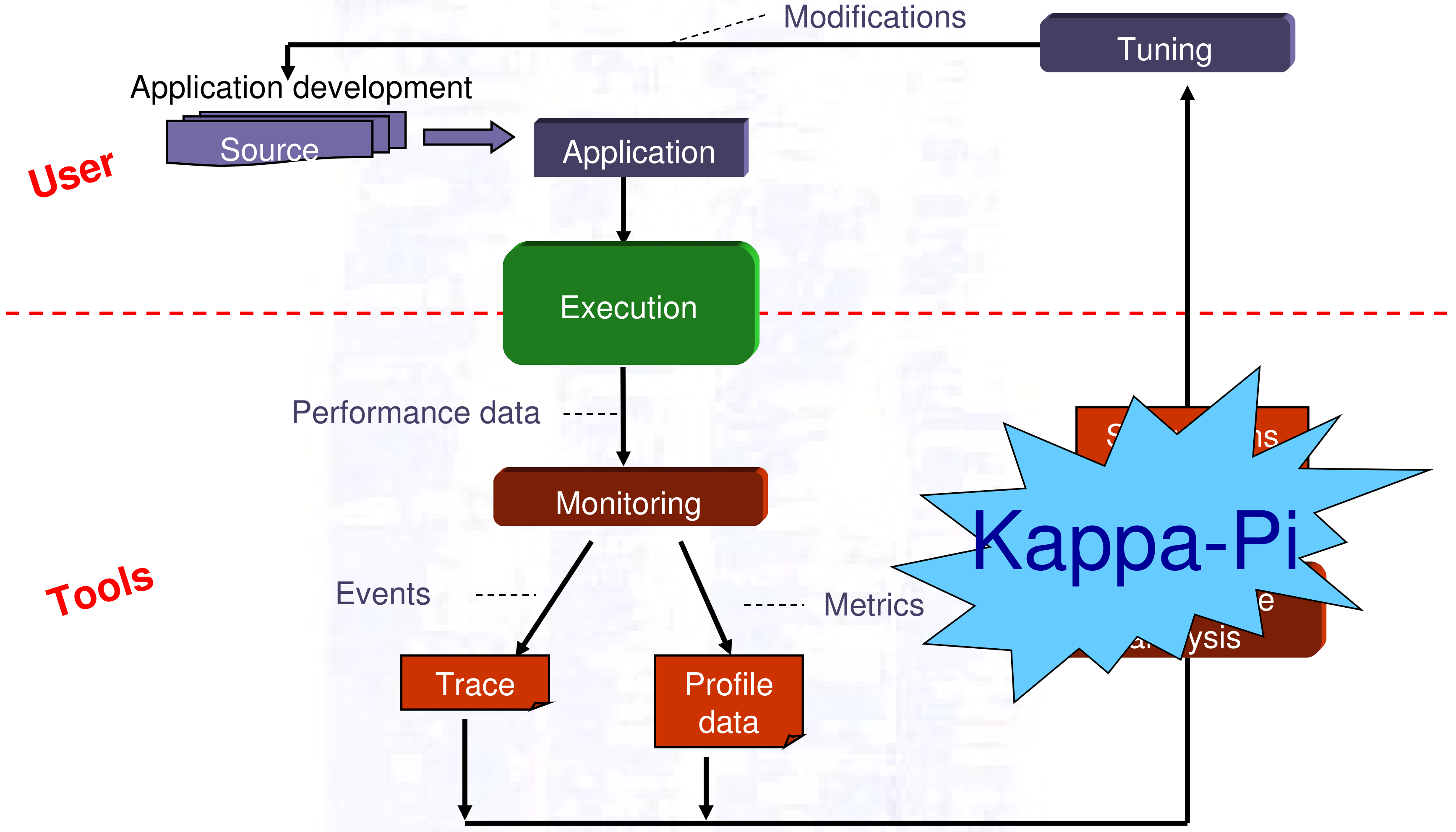
Performance analysis and tuning

Post-mortem manual analysis



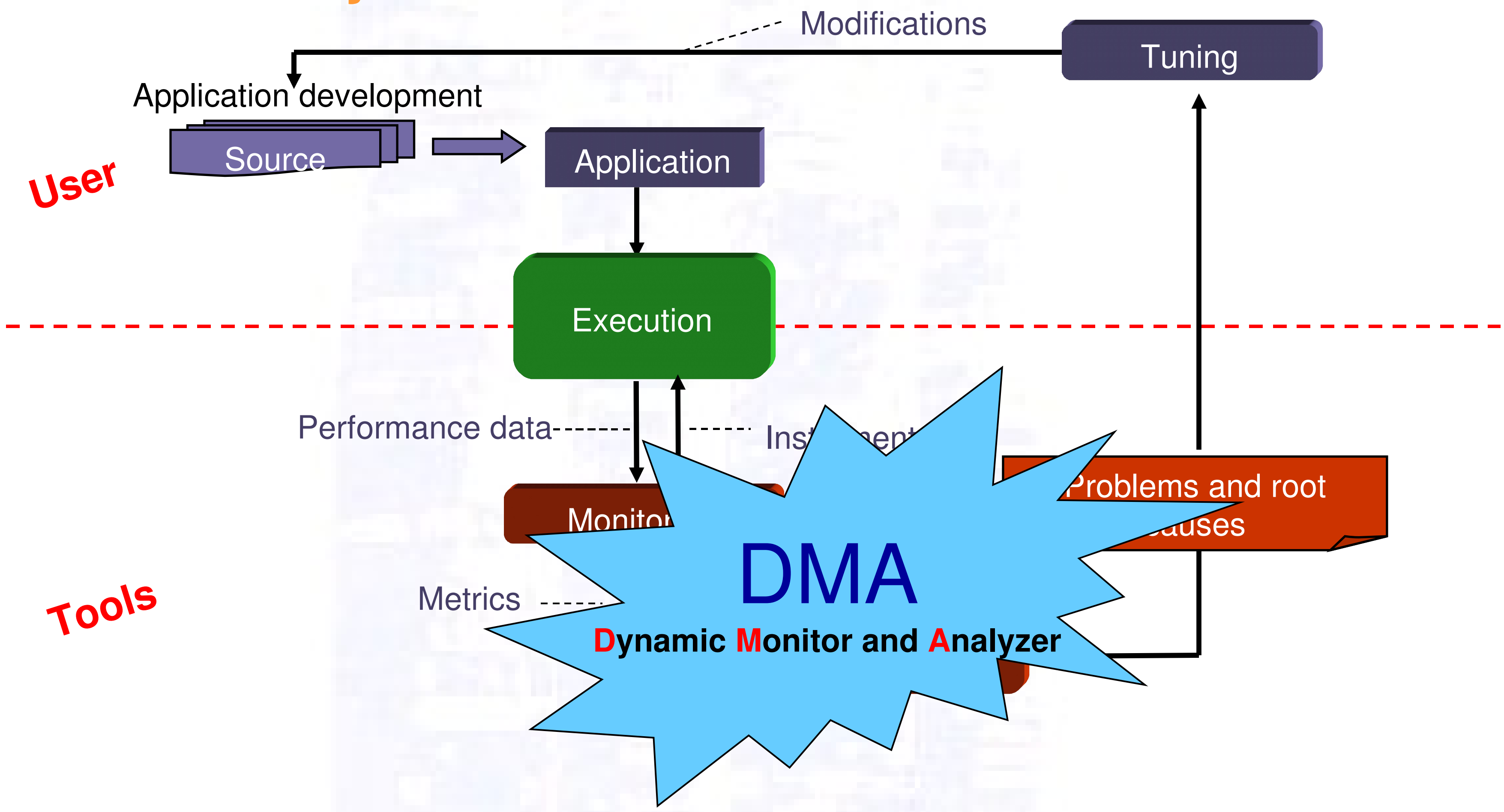
Performance analysis and tuning

Post-mortem automatic analysis



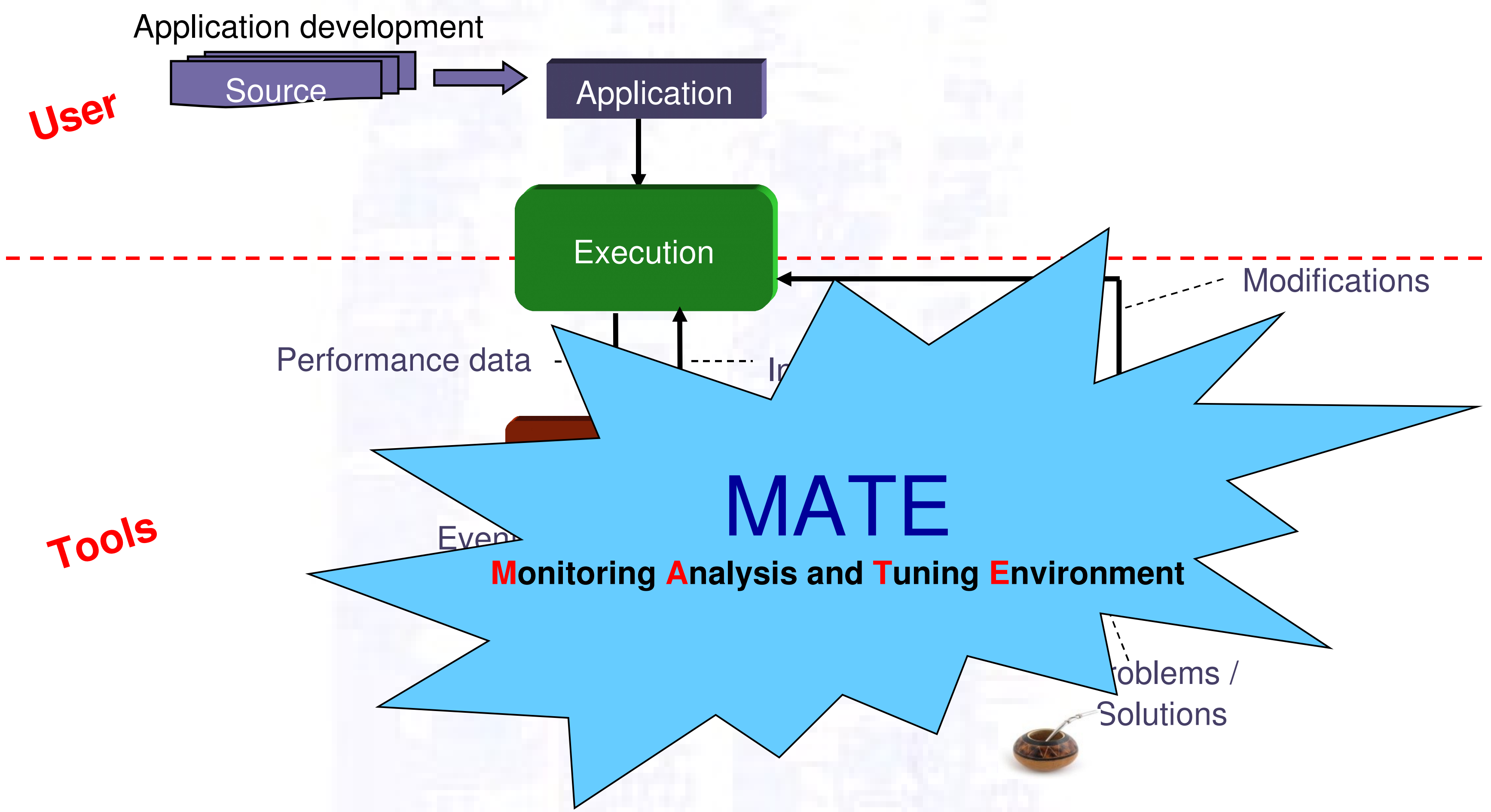
Performance analysis and tuning

Run-time analysis



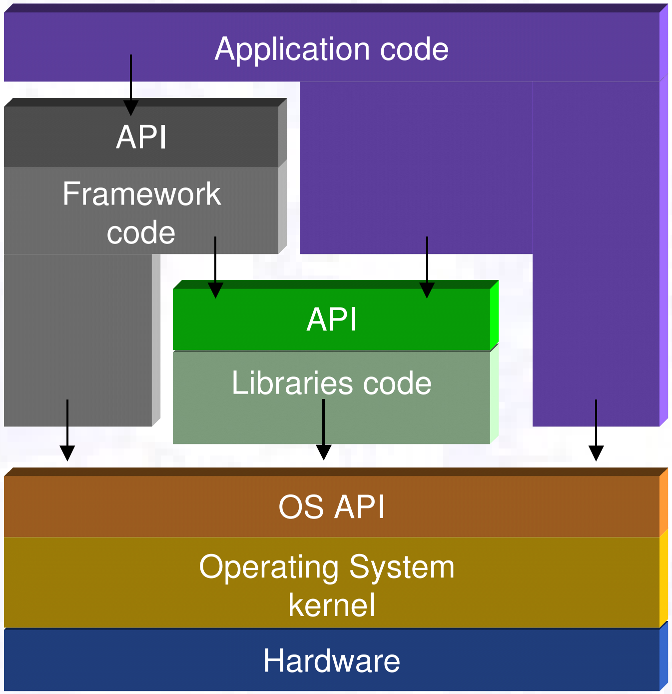
Performance analysis and tuning

Dynamic tuning



Performance analysis and tuning

What can be tuned in an application?

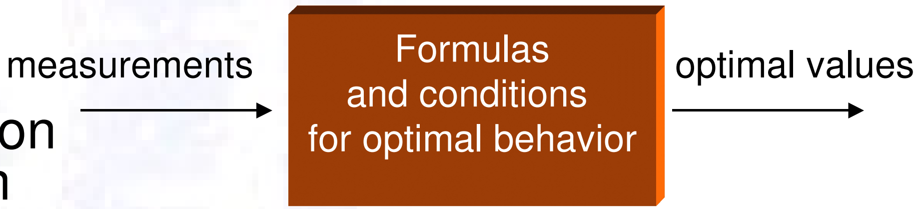


Performance analysis and tuning

Knowledge representation

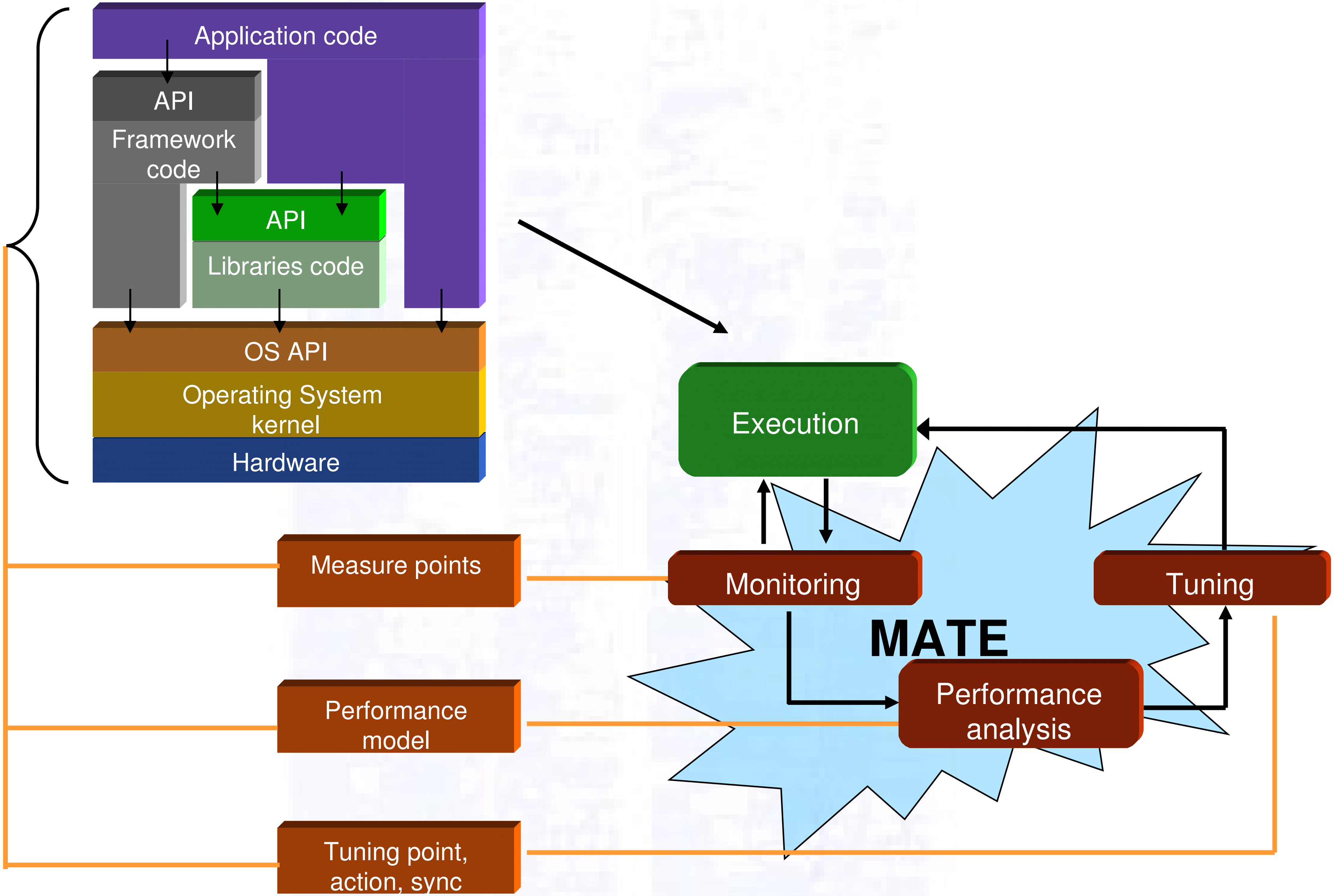
- Measure points
  - Where the instrumentation must be inserted to provide measurements

- Performance model
  - Determines minimal execution time of the entire application



- Tuning points/actions/synchronization
  - What and when can be changed in the application
    - point – element that may be changed
    - action – what operation to invoke on a point
    - synchronization – when a tuning action can be invoked to ensure application correctness

Performance analysis and tuning



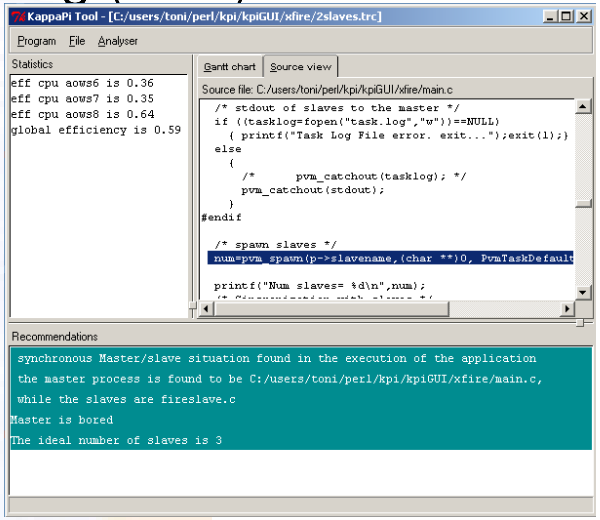
Outline

- **Post-mortem analysis**
- Online modeling and analysis
- Performance models
  - Framework-based applications
  - Scientific libraries
- MATE
- Future work

Post-mortem analysis

Kappa-Pi tool

- Knowledge-Based Automatic Parallel Program Analyser for Performance Improvement
  - Trace analyzer: automatically detects bottlenecks, relates them to source code and provides suggestions for a user
  - For PVM/MPI applications
  - Rule-based inference engine - detection based on decision trees
  - Declarative communication problems catalog (XML)
  - Problems declared as structured event patterns
  - Focused on analysis of inefficiency intervals
  - Basic source code analysis



Outline

- Post-mortem analysis
- **Online modeling and analysis**
- Performance models
  – Framework-based applications
  – Scientific libraries
- MATE
- Future work

Online modeling and analysis

DMA: Dynamic Modeler and Analyzer

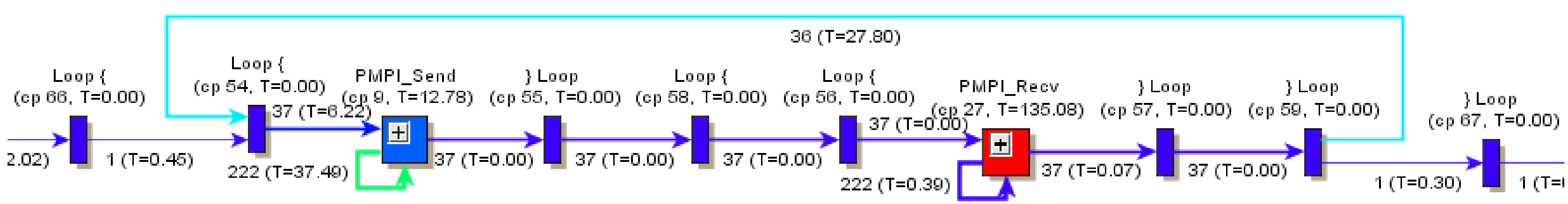
- Primary objective
  - Develop a tool that is able to analyze the performance of parallel applications, detect bottlenecks and **explain their reasons**

- Our approach
  - Dynamic on-the-fly analysis – DynInst library
  - Online performance modeling
    - captures the application structure and behavior
  - Root-cause performance analysis
  - Tool targeted to MPI programs and communication problems

Online modeling and analysis

Task Activity Graph (TAG)

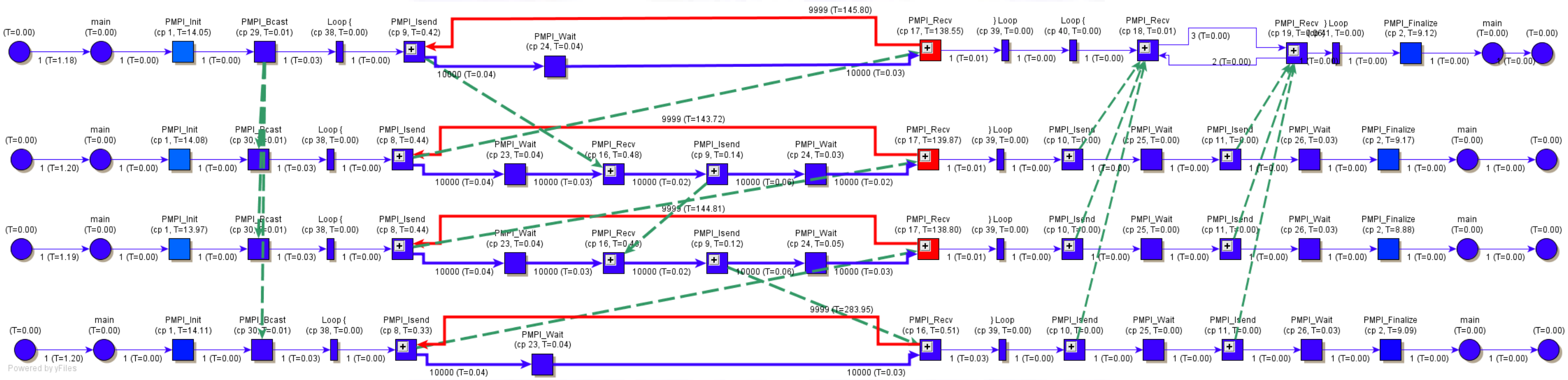
- Abstracts execution of a single task
- Execution is described by units that correspond to different activities
- **Nodes** reflect execution of communication activities and selected loops
- **Edges** represent sequential flow of execution (computation activities)
- Nodes and edges behavior is described by **execution profiles**
- Profiles contains **aggregated** performance **metrics**
- TAG maintains happens-before relationship between nodes and edges
- Model is constructed incrementally at run-time in memory of each task



Online modeling and analysis

Parallel model (PTAG)

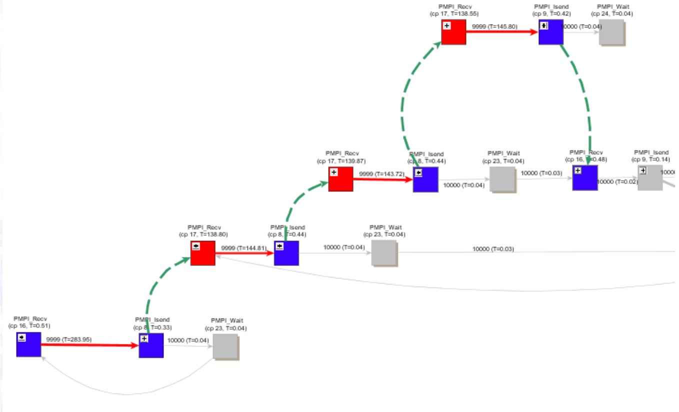
- Individual TAG models connected by **message edges** (P2P, Collective) enable construction of **Parallel-TAG** (PTAG)
- This can be done at run-time: sender call-path id sent with every MPI message
- PTAG is updated periodically by sampling and merging TAGs



Online modeling and analysis

How to find problems and their causes?

- **Root Cause Analysis (RCA)**
  - Online approach to performance analysis
  - Aims to find bottlenecks and their causes
  - Focuses on finding causes of latencies
  - Based on TAG/PTAG model
- RCA is a continuous analysis process that is divided in 3 phases:
  - Phase 1: Identify performance problems
  - Phase 2: Analyze individual problems
  - Phase 3: Correlate individual problems and find their root causes



Online modeling and analysis

Phase 1: Performance problems identification

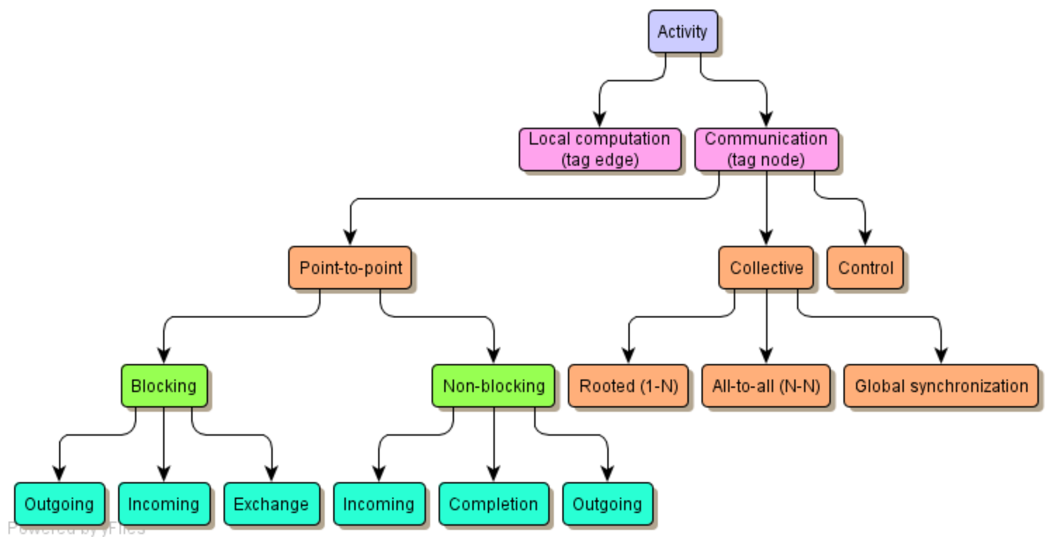
- A **bottleneck** is defined as an individual task activity that has accumulated a significant amount of execution time
- In the TAG model a single bottleneck is identified by a node or an edge
- The bottleneck can be a symptom of another problem or a root cause

- **Identification** (local / global)
  - Capture TAG/PTAG snapshot
  - Rank nodes and edges
  - Select top-k candidates
  - Periodic process (moving time-window)

Online modeling and analysis

Phase 2: Analyze individual problems

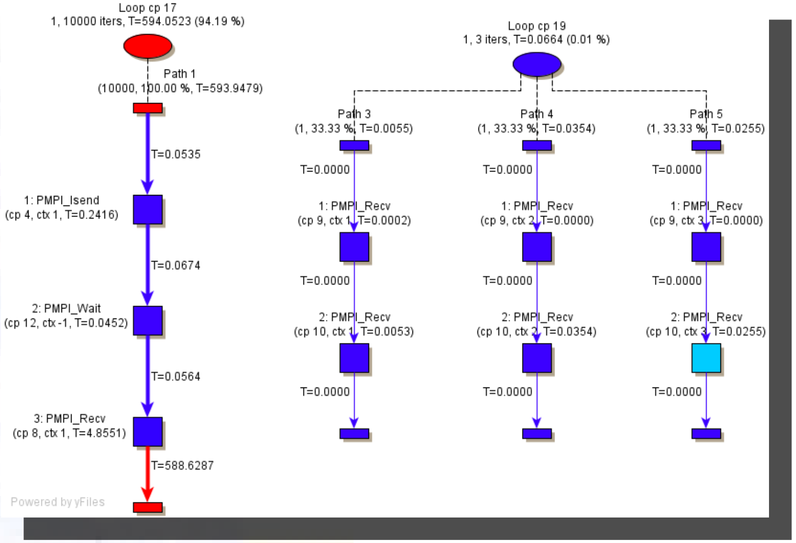
- For each problem detected in phase 1, we investigate the possible higher-level **causes** by exploring a **knowledge-based cause space**.
- We focus on determining causes that contribute most to the total problem time.
- For each activity we define a **performance model** that quantifies its cost
- Task activity classification used for better **problem understanding**



Online modeling and analysis

Phase 3: Causality paths

- **How to explain causes of latencies?** For example: why sender is late?
- Cause of waiting time between two nodes as the **differences** between their **execution paths**
- We can track activities before the problem occurs on both nodes and compare them
- We call them **causality paths** as they have happens-before property
- We consider two sources of causality:
  - Sequential flow
  - Message flow
- Detect causality paths in sender to explain latencies in receiver



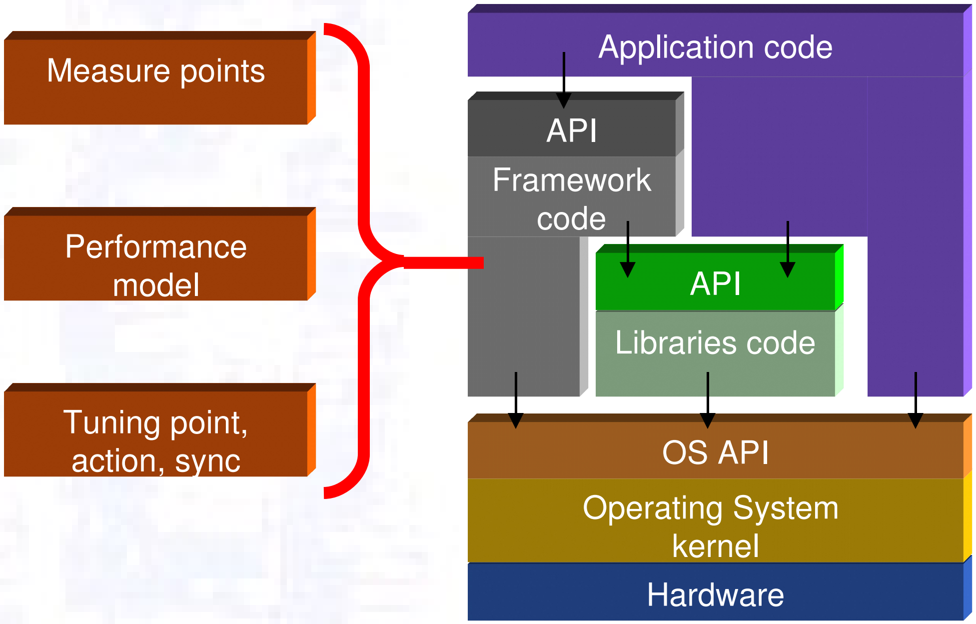
Outline

- Post-mortem analysis
- Online modeling and analysis
- **Performance models**
  - Framework-based applications
  - Scientific libraries
- MATE
- Future work

Performance models

Framework-based applications

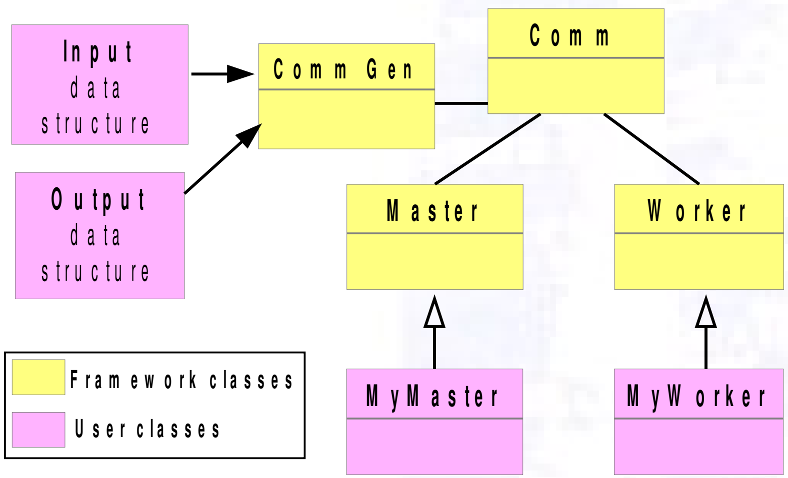
- Frameworks help users to develop parallel applications providing APIs that hide low level details (M/W, pipeline, divide and conquer)
- We can extract the knowledge given by frameworks to define performance models for automatic and dynamic performance tuning (MATE)
- Examples: M/W framework (UAB), eSkel (The Edinburg Skeleton library), llc (La Laguna Compiler)

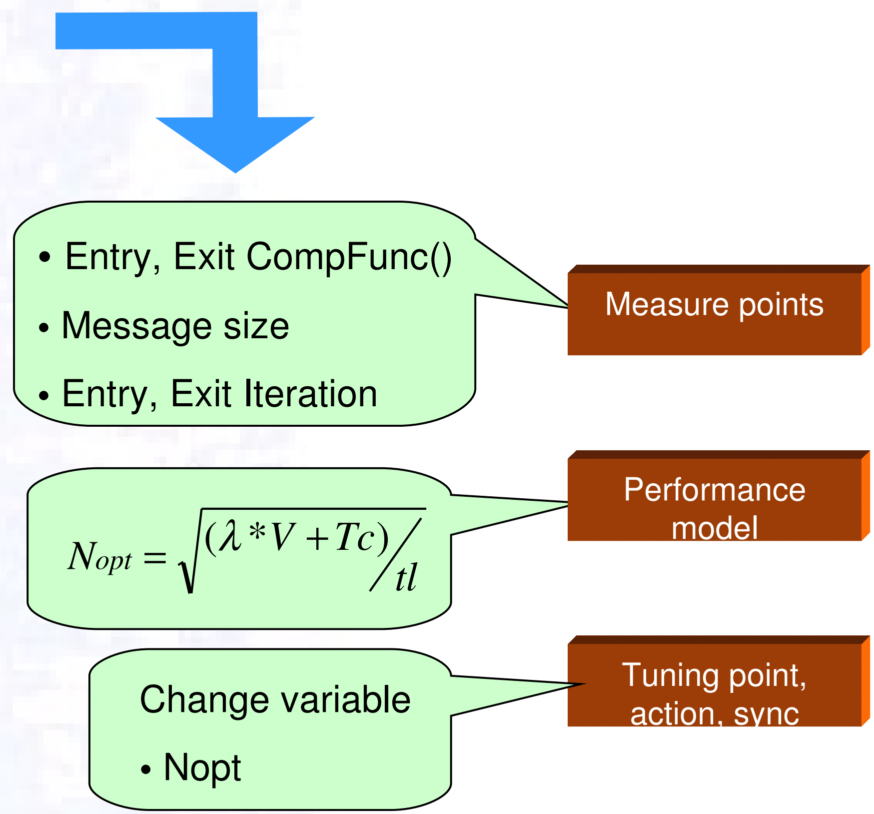


Performance models

Framework-based application example

- M/W application



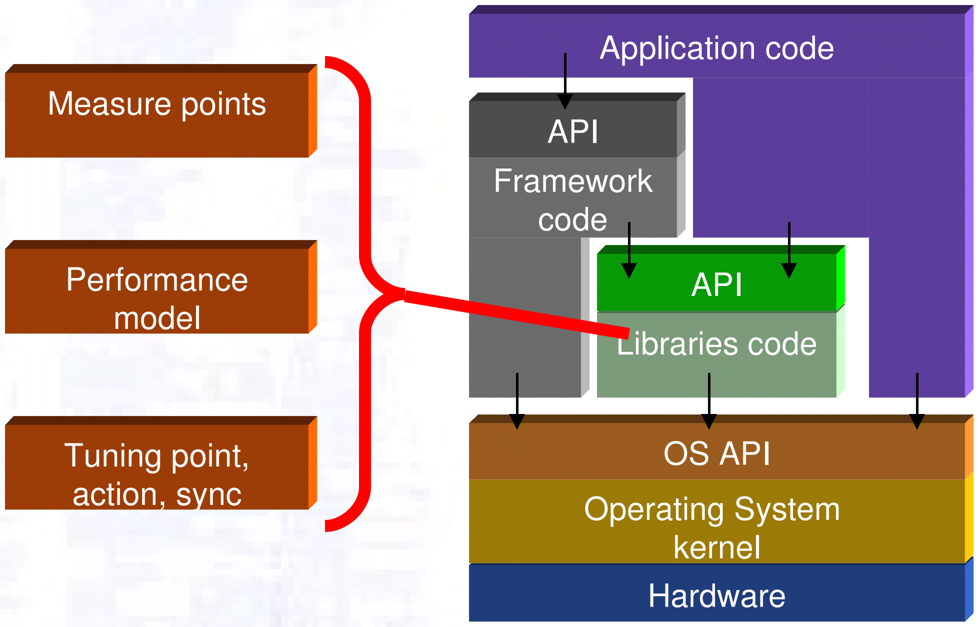


- Problems for tuning:
  - Load imbalance
  - Improper number of workers

Performance models

Scientific libraries modeling

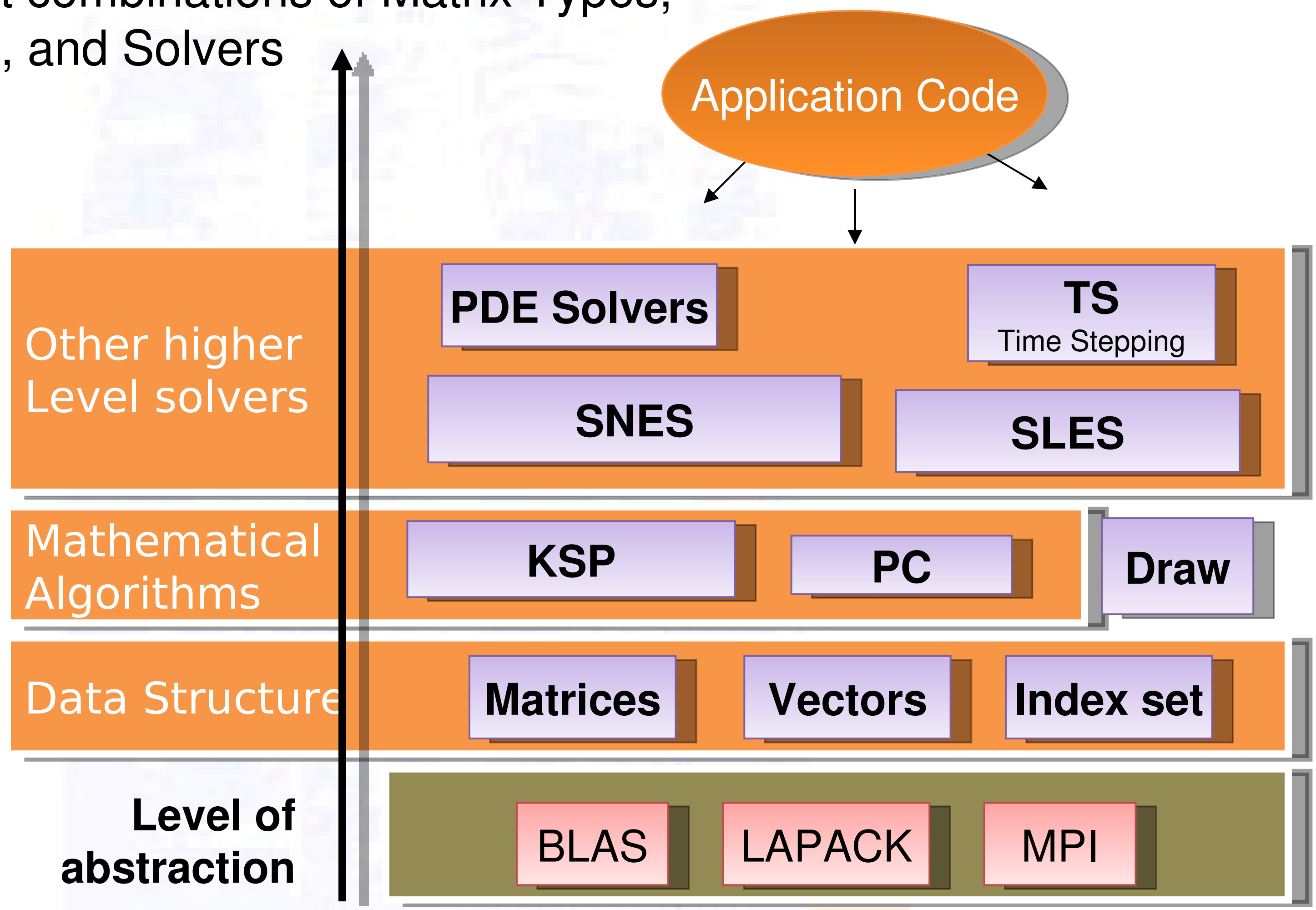
- Libraries help users to solve domain-specific problems
- Using the knowledge learned from scientific programming libraries to define performance models for automatic and dynamic performance tuning (MATE)
- Examples: mathematical library PETSc (Portable Extensible Toolkit for Scientific Computation)



Performance models

Scientific libraries modeling: PETSc

Study of different combinations of Matrix Types, Pre-conditioners, and Solvers



Performance models

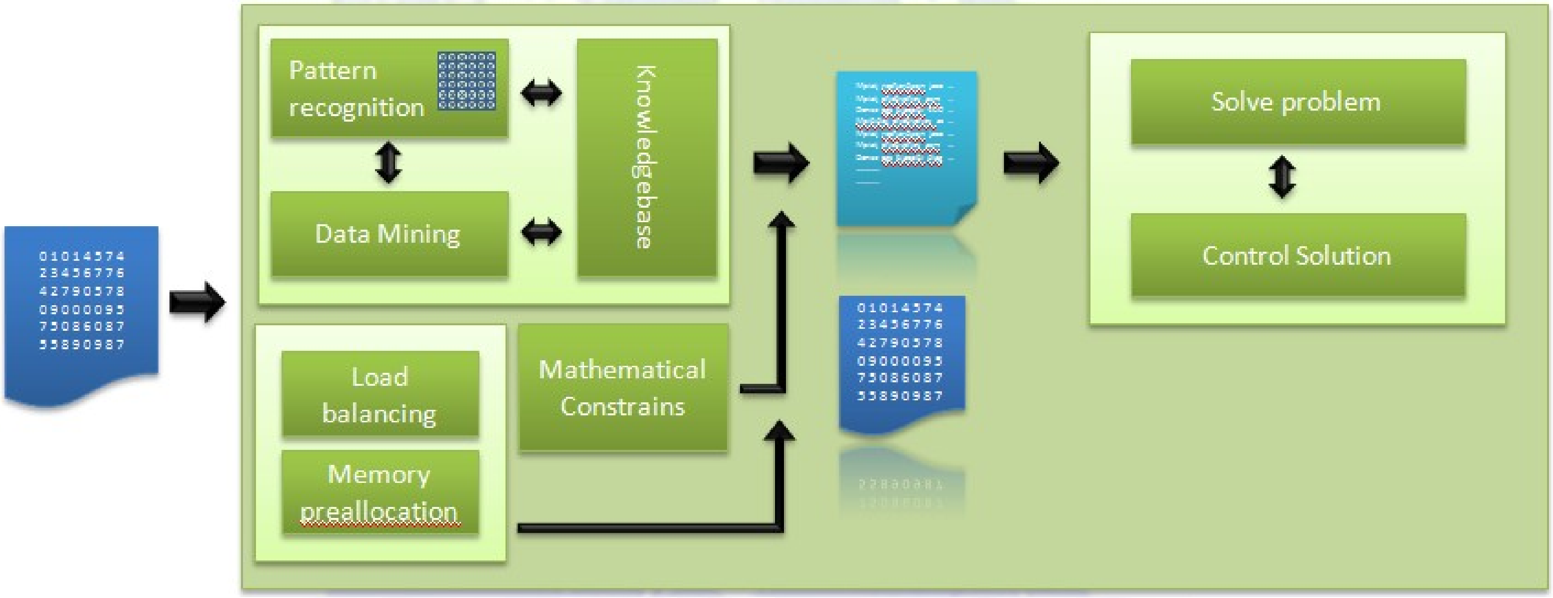
Scientific libraries modeling: PETSc

- PETSc supports different storage methods (dense, sparse, compressed, etc.) – no data structure is appropriate for all problems
- Linear system calculations involve matrix's preconditioner (PC) and Krylov Subspace method (KSP)
- Different mathematical algorithms for preconditioner (jacobi, LU, etc.) and KSP (GMRES, CG, etc.)
- Storage methods and algorithm selection can significantly affect the performance

Performance models

Scientific libraries modeling: PETSc

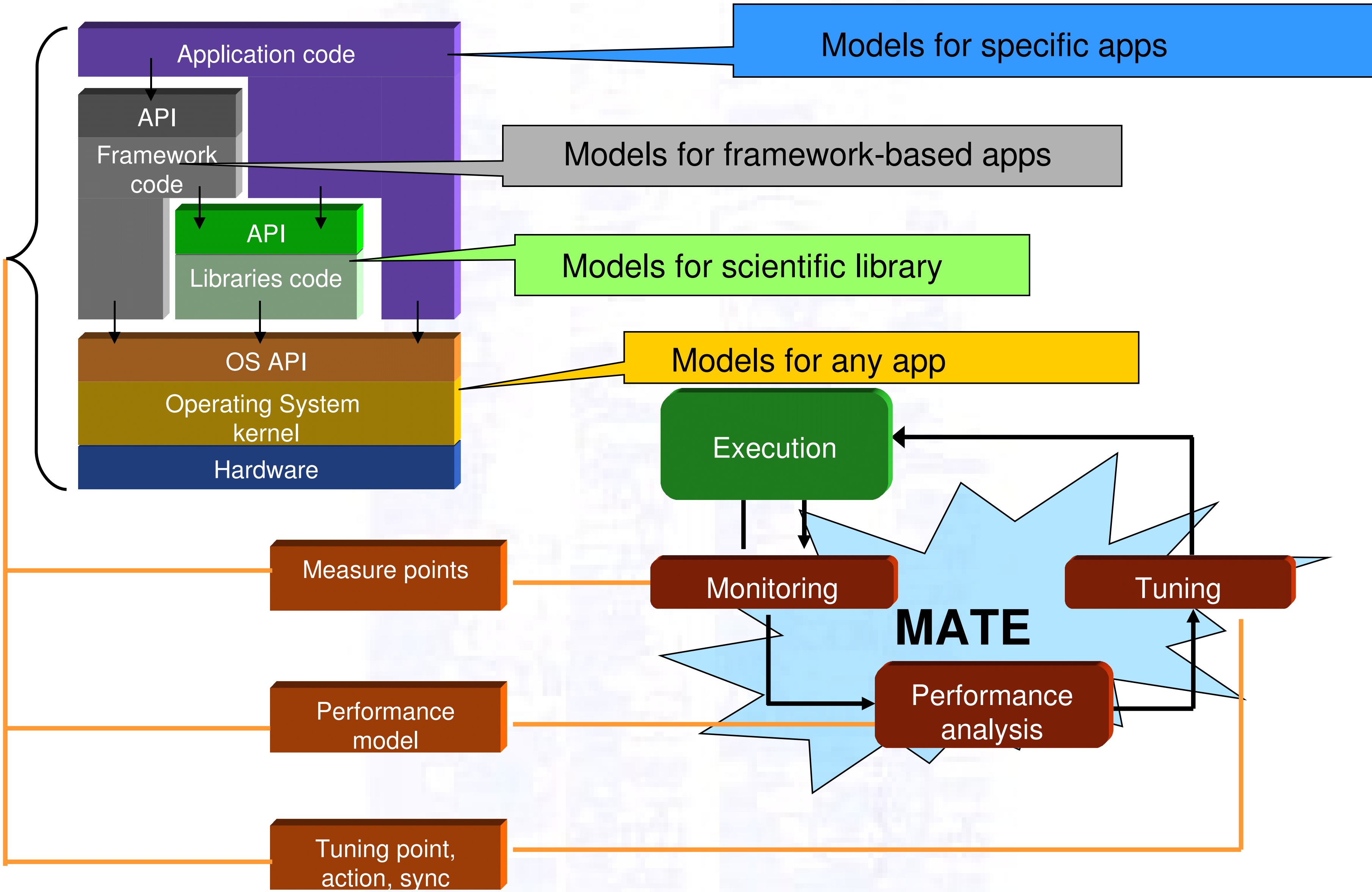
- We build a knowledge base that maps problems (different classes of matrices, matrix sizes, and other variables) to best configurations
- An input matrix is compared to the knowledge base using K-nearest-neighbors algorithm to select the most similar problem and its configuration
- Load balance applying different number of lines/values



Outline

- Post-mortem analysis
- Online modeling and analysis
- Performance models
  - Framework-based applications
  - Scientific libraries
- **MATE**
- Future work

MATE

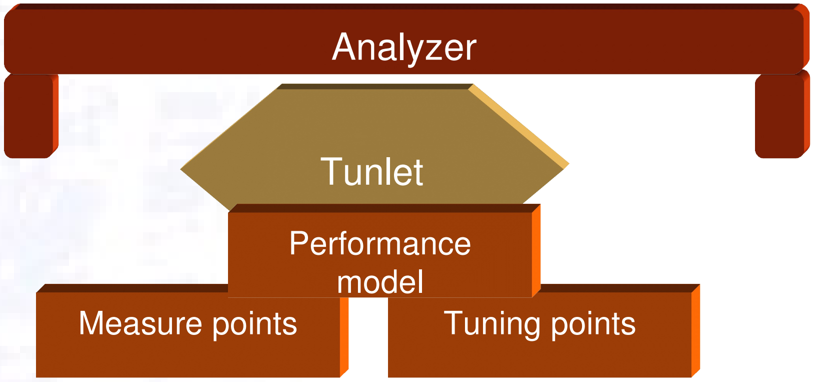


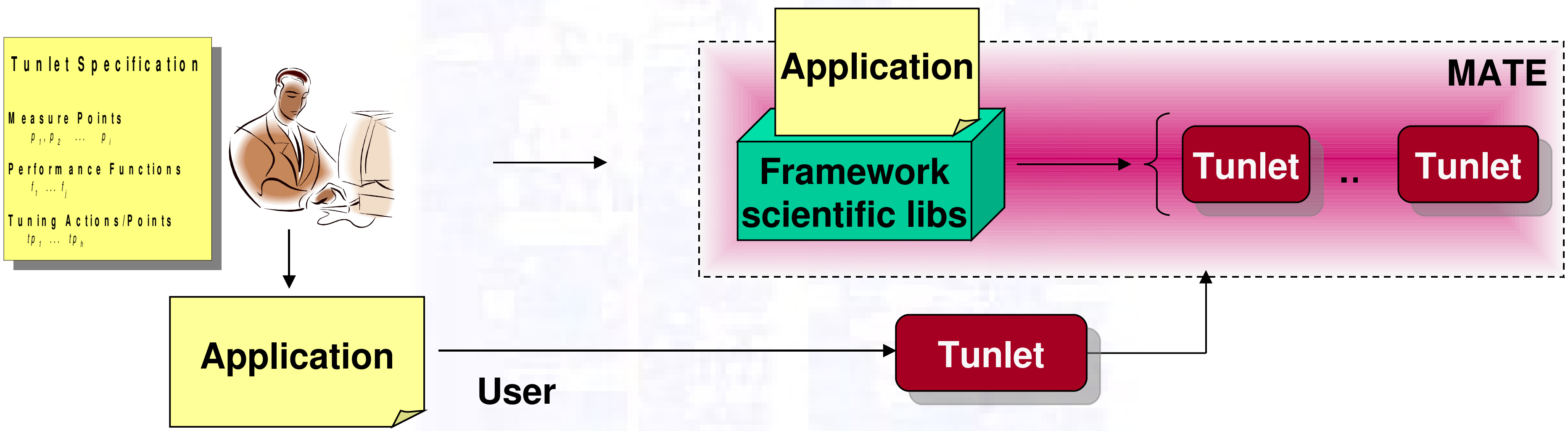
Application code
API
Framework code
API
Libraries code
OS API
Operating System kernel
Hardware
Models for specific apps
Models for framework-based apps
Models for scientific library
Models for any app
Execution
Monitoring
Tuning
MATE
Performance analysis
Measure points
Performance model
Tuning point, action, sync

MATE

Tunlets

- Mechanism for specifying a performance model
- Originally, each tunlet is written as C++ code library
- Simplify tunlet development:
  - Tunlet Specification Language
  - Automatic Tunlet Generator

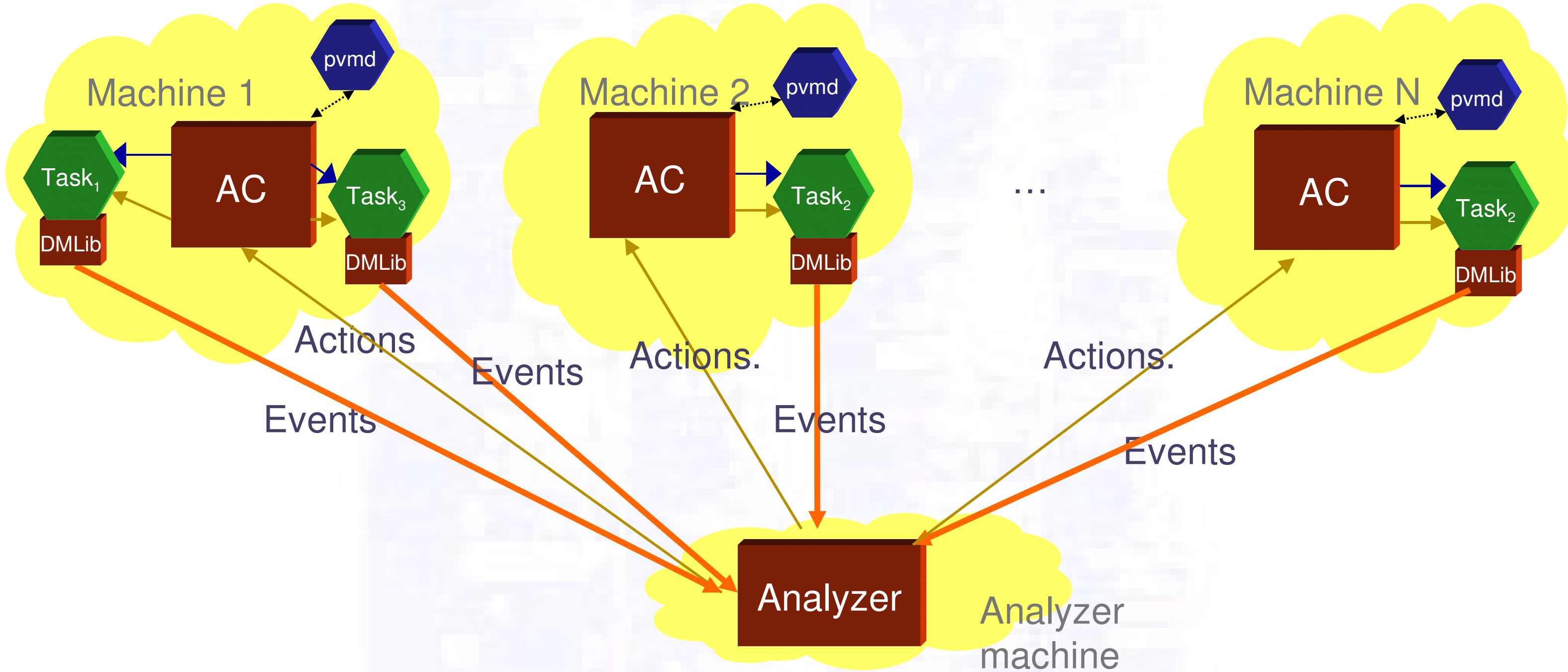




MATE

MATE scalability

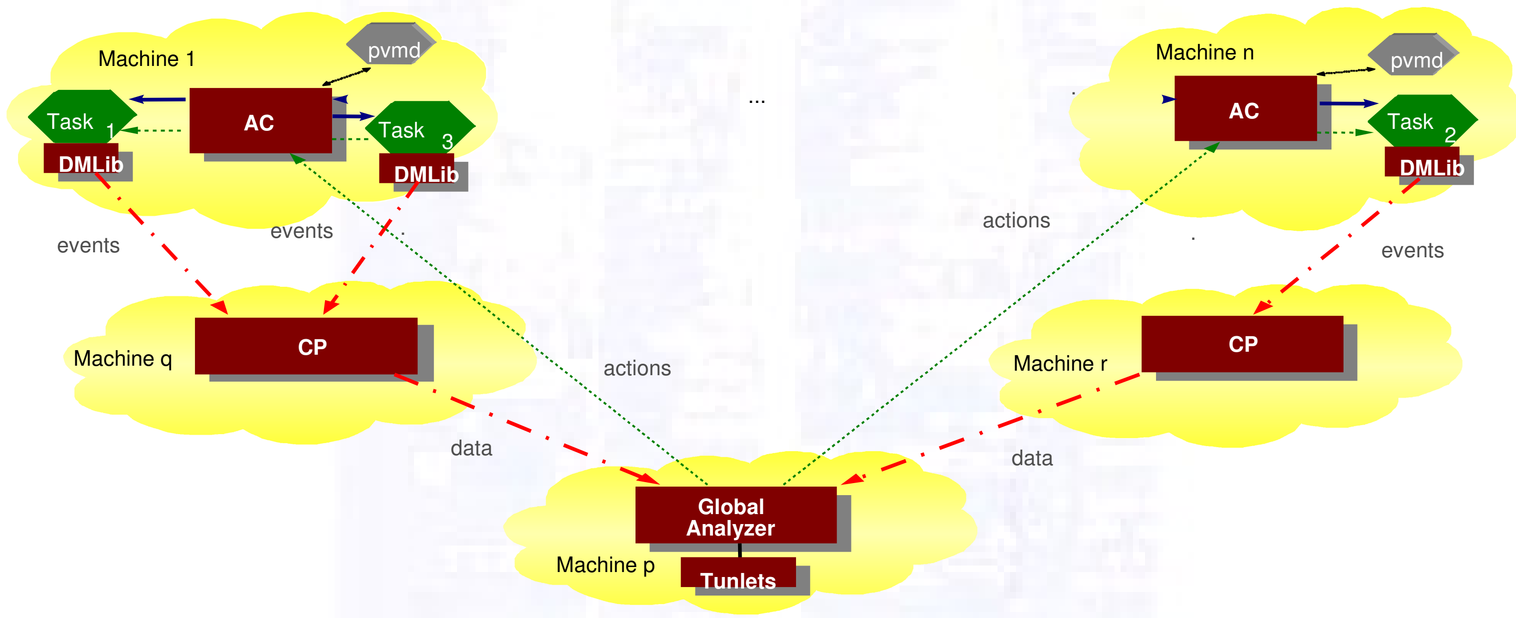
- Scalability problems with centralized analysis
  - The volume of events gets too high



MATE

MATE scalability

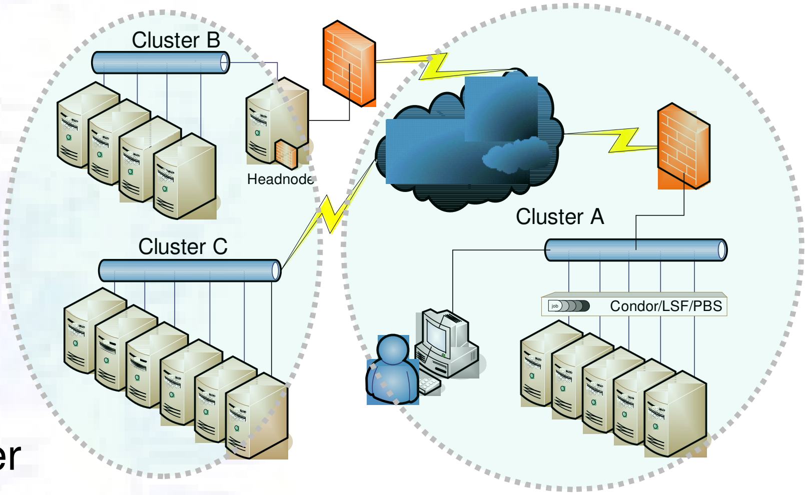
- Distributed-hierarchical collecting-preprocessing approach
  - Distribution of event collection
  - Collector-Preprocessor - preprocessing of certain calculations: cumulative and comparative operations
  - Global Analyzer – performance model evaluation



MATE

MATE in Grid Systems (GMATE)

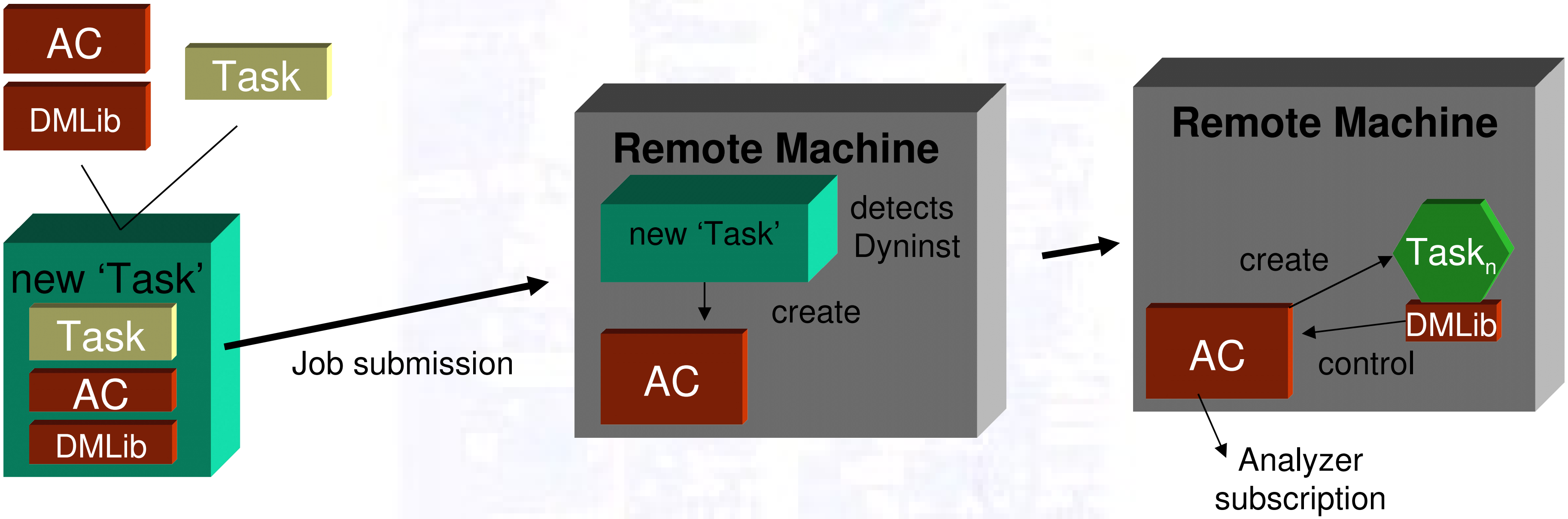
- Grid performance models
- Tool distribution architecture
  - Application Monitoring - transparent process tracking and control
    - AC should follow application process to any cluster
    - Inter-cluster communication with analyzer
  - Lower inter-cluster event collection overhead
    - Inter-cluster communications generally have high latency and lower bandwidth
    - Remote trace events should be aggregated
  - Analysis in Grid



MATE

G-MATE: Transparent process tracking

- **Application wrapping**
  - AC can be binary packaged with application binary



MATE

G-MATE: Analyzer approach

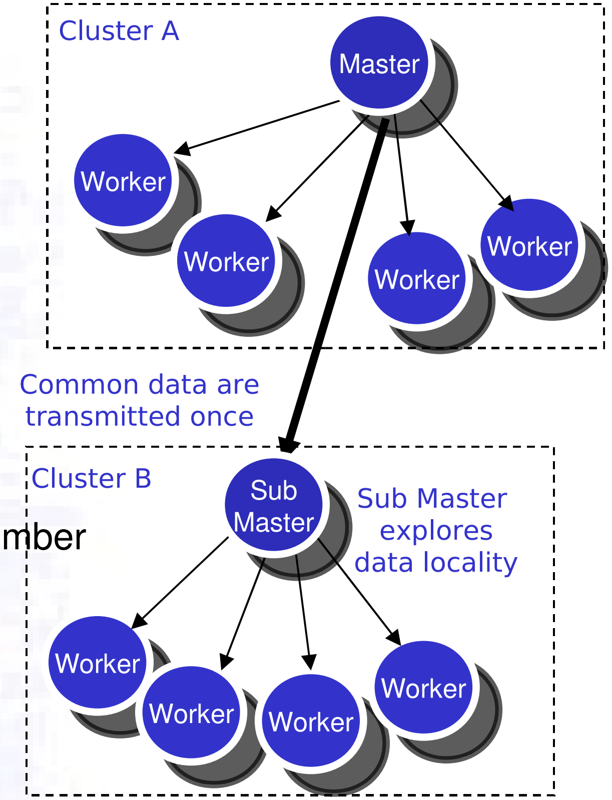
- **Central analyzer**
  - Collects performance data in a central site
- **Hierarchical analyzer**
  - Local analyzers process local data, generates abstract metrics and send to global analyzers
- **Distributed analyzer**
  - Each analyzer evaluates its local performance data and abstract global data cooperating with the rest of analyzers

MATE

Tuning scenario

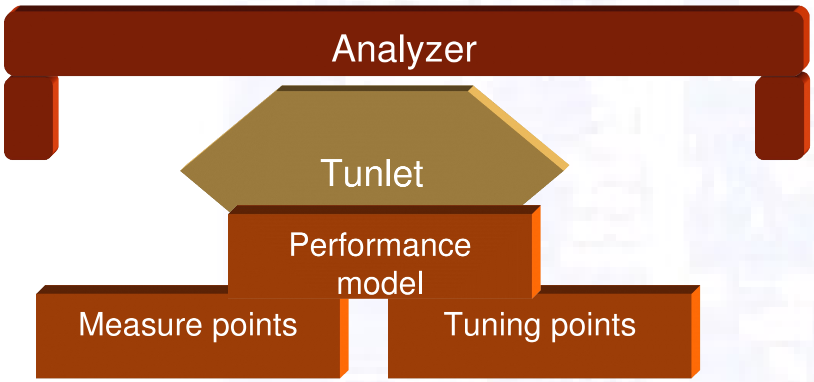
Distributed Grid Application

- Hierarchical Master/Worker Matrix Multiplication



Performance model for:

- Number of Workers
- Application Grain Size
  - Change compute/communication ratio - change maximum number of workers



Outline

- Post-mortem analysis
- Online modeling andanalysis
- Performance models
  - Framework-based applications
  - Scientific libraries
- MATE
- **Future work**

Future work

- DMA – new methodology final phase of development
- Development of performance models:
  - for applications based on different frameworks
    - Join both strategies: load balance and number of workers
    - Replication/elimination of pipeline stages
  - PETSc-based applications – under construction
  - for black-box applications
  - for grid applications – under construction
- Development of tuning techniques

Future work

- MATE’s analysis improvement
  - Performance model of the number of CPs
  - Hierarchy at the CPs level as the number of machines increases (MRNet?)
  - Root cause analysis?
- Instrumentation evaluation
- Co-execution of tunlets

Universitat Autònoma de Barcelona

Thank you very much

Anna Morajko

Anna.Morajko@uab.es

26-05-2008