MATE and DMA: Tools for Dynamic Performance Analysis

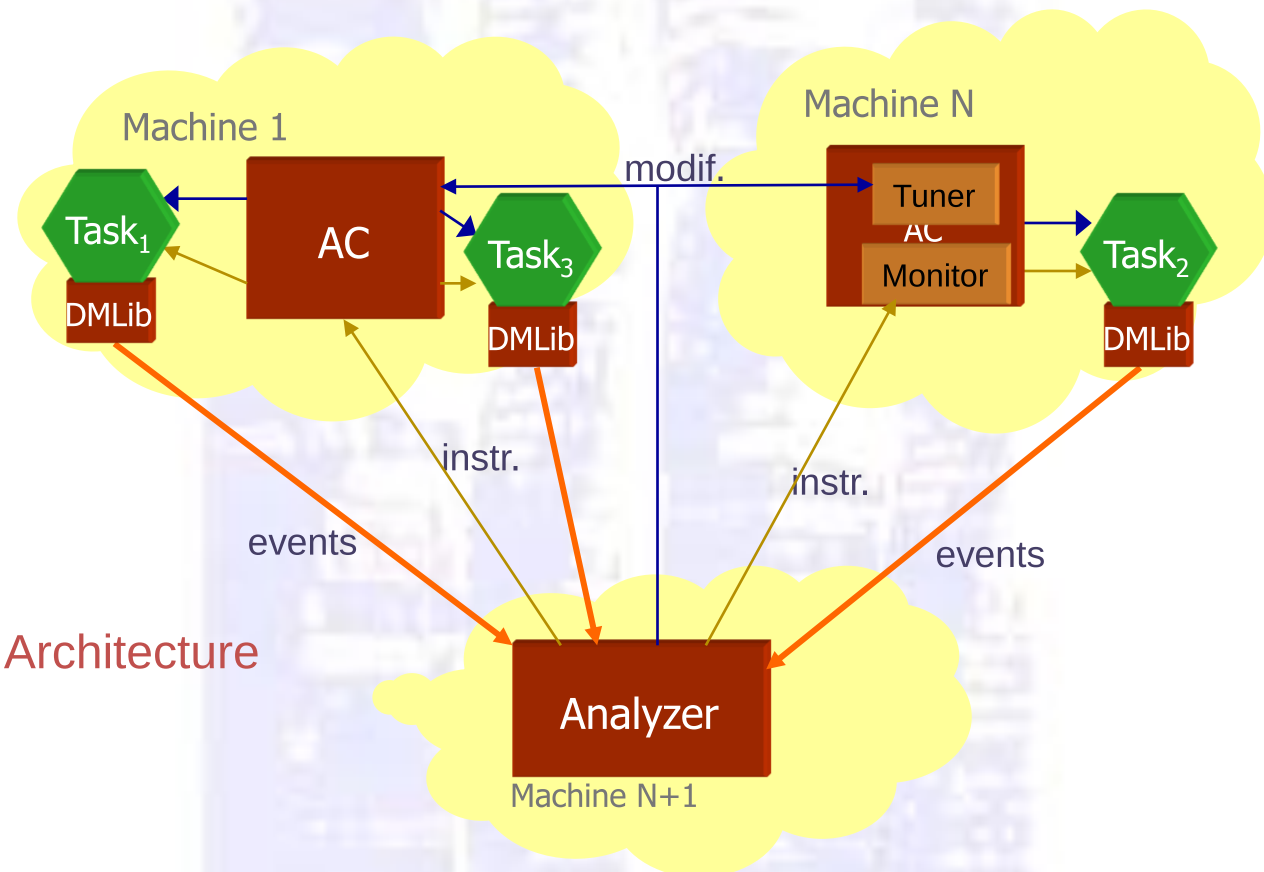
Eduardo Cesar
Computer Architecture and OS Department UAB

Contents

- MATE
  - Overview
  - Components
- DMA
  - Overview & building the model (TAG and PTAG)
  - Performance Analysis

MATE: Overview

MATE - Monitoring, Analysis and Tuning Environment

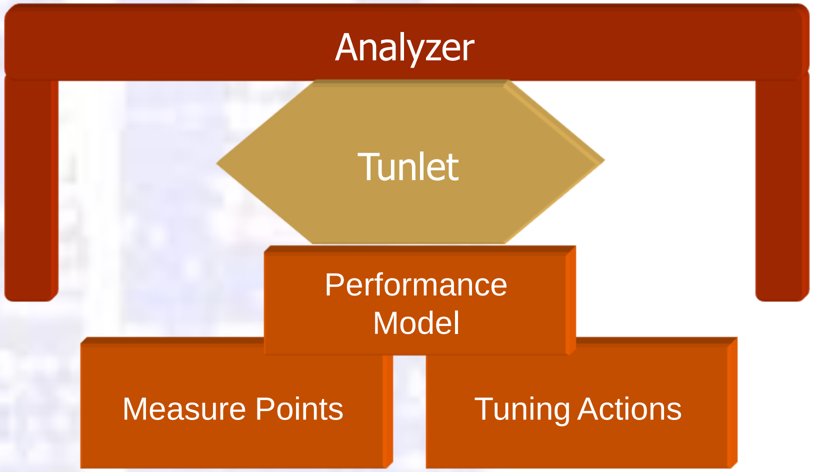


MATE Architecture

MATE: Overview

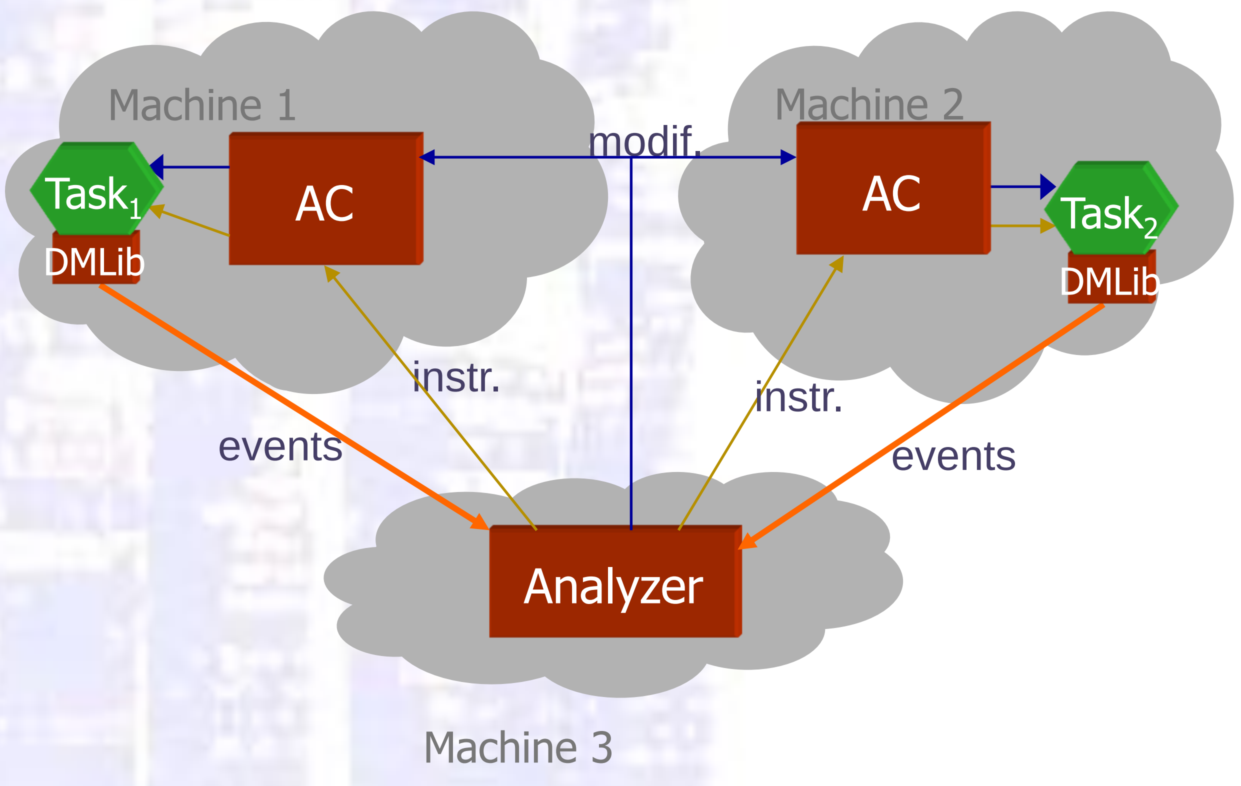
- The user specified model is made of:
  - Measure Points – where to insert instrumentation
  - Performance Model- how is the application analyzed
  - Tuning Actions– how to overcome performance bottlenecks (and when)

- All this knowledge is provided in the form of a **tunlet** – a user provided piece of coded integrated to the Analyzer.



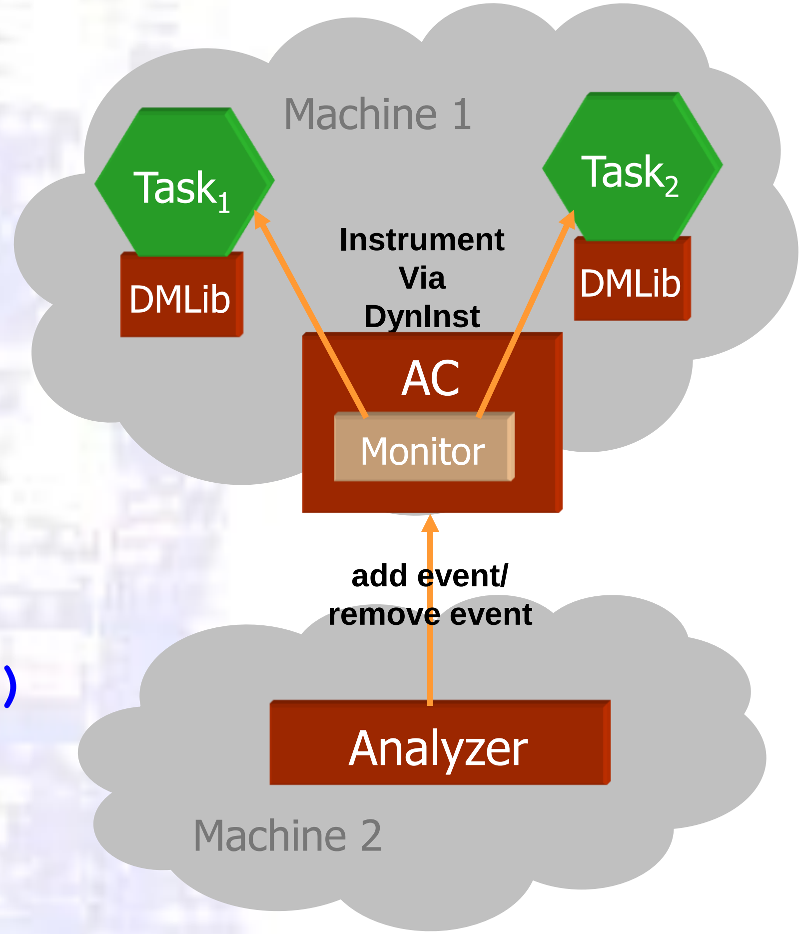
MATE: Components

- Application Controller – AC (Monitor/Tuner)
- Dynamic Monitoring Library – DMLib
- Analyzer



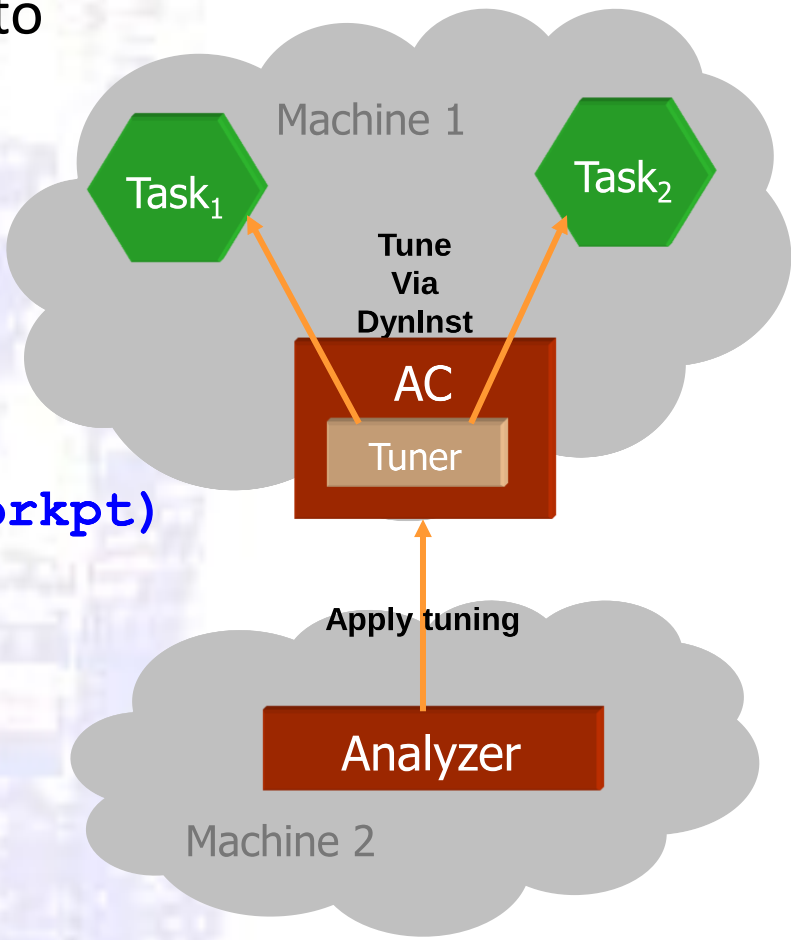
MATE: Components (Monitors)

- Instrumentation management via DynInst
  - Dynamically load DMLib
  - Generate monitoring snippets that call appropriate library functions
  - Insert/remove snippets in/from requested points
- API
  - `AddEventTrace(tid, eventId, funcName, instrPlace, attrs)`
  - `RemoveEventTrace(tid,eventId)`



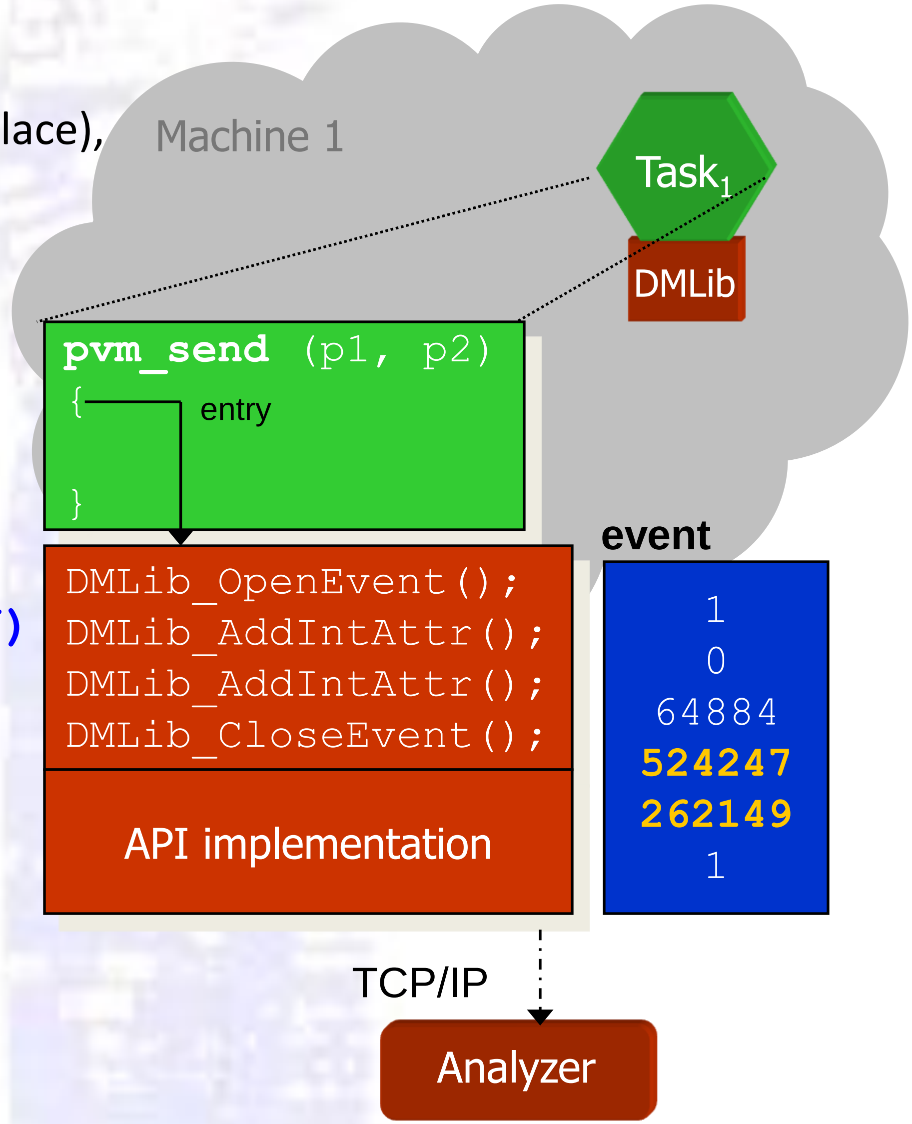
MATE: Components (Tuners)

- Tuning via DynInst
  - Generate tuning snippet according to the Analyzer's request
  - Inserting tuning snippet
- API
  - `LoadLibrary(tid,path)`
  - `SetVariableValue(tid,params,brkpt)`
  - `ReplaceFunction(…)`
  - `InsertFunctionCall(…)`
  - `OneTimeFunctionCall(…)`
  - `RemoveFunctionCall(…)`
  - `FunctionParamChange(…)`



MATE: Components DMLib

- Register event
  - **What, When, Where** – event type (id, place), global timestamp, task identifier
  - **Requested attributes**
- Deliver event to the Analyzer
- API
  - `DMLib_InitLogger(tid, analyzerHost,port,clockDiff)`
  - `DMLib_OpenEvent(id, nAttrs)`
  - `DMLib_AddIntAttr(value)`
  - `DMLib_AddFloatAttr(value)`
  - `DMLib_AddCharAttr(value)`
  - `DMLib_AddStringAttr(value)`
  - `DMLib_CloseEvent()`
  - `DMLib_DoneLogger()`



MATE: Components (Analyzer)

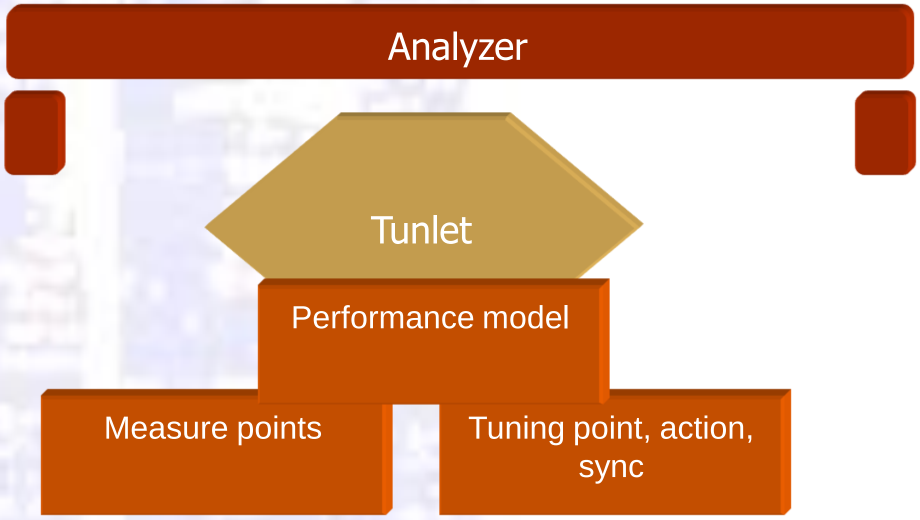
Services

- Automatic performance analysis on the fly
  - Request for events
  - Collect incoming events
  - Find bottlenecks among events applying the performance model
  - Find solutions that overcome bottlenecks
  - Send tuning request
- Analyzer is provided with the application *knowledge* about performance problems

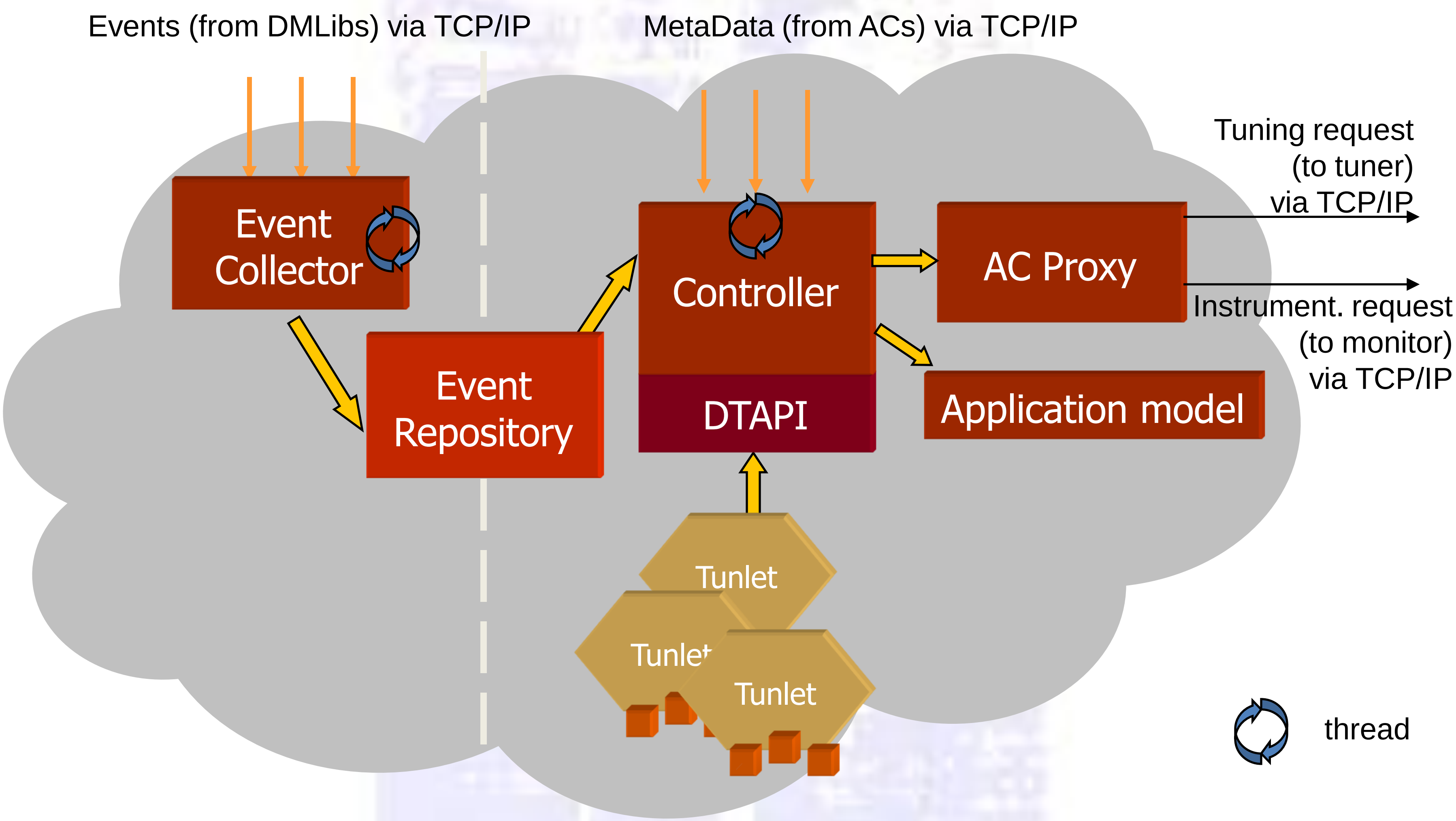
MATE: Components (Analyzer)

Tunlets

- This knowledge is provided as a set of **tunlets**
- A tunlet contains specific code related to a concrete performance problem
- A tunlet is a C/C++ library dynamically loaded into the Analyzer process



MATE: Components (Analyzer)



Contents

- MATE
  - Overview
  - Components
- DMA
  - Overview & building the model (TAG and PTAG)
  - Performance Analysis

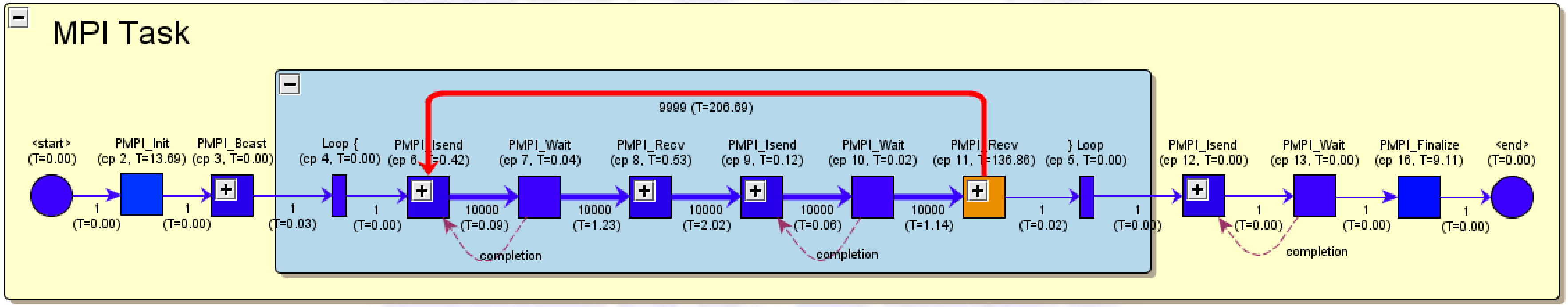
DMA: Overview

- **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***
  - ***Automatic modeling of application structure and behavior***
  - Root-cause analysis based on happens-before relationships
  - Tool primarily targeted to MPI-based parallel programs
  - Focus on communication problems
  - Applicable to wide range of MPI applications
  - ***Scalable to thousands and more CPUs***
  - Easy to use: no source code

DMA: Building de Model

Task Activity Graph (TAG)

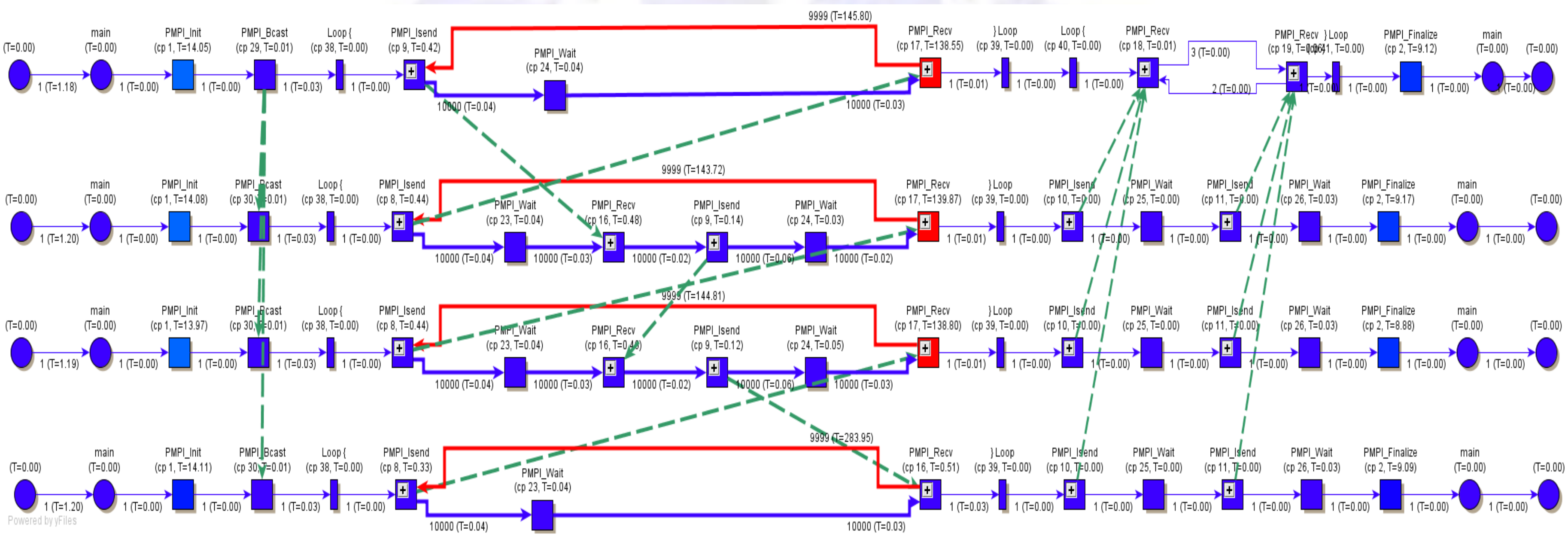
- Abstracts execution of a single task
- Execution is described by units that correspond to different activities
- **Nodes** reflect execution of communication activities and selected loops
- **Edges** represent sequential flow of execution (computation activities)
- TAG maintains **happens-before relationship** between nodes and edges



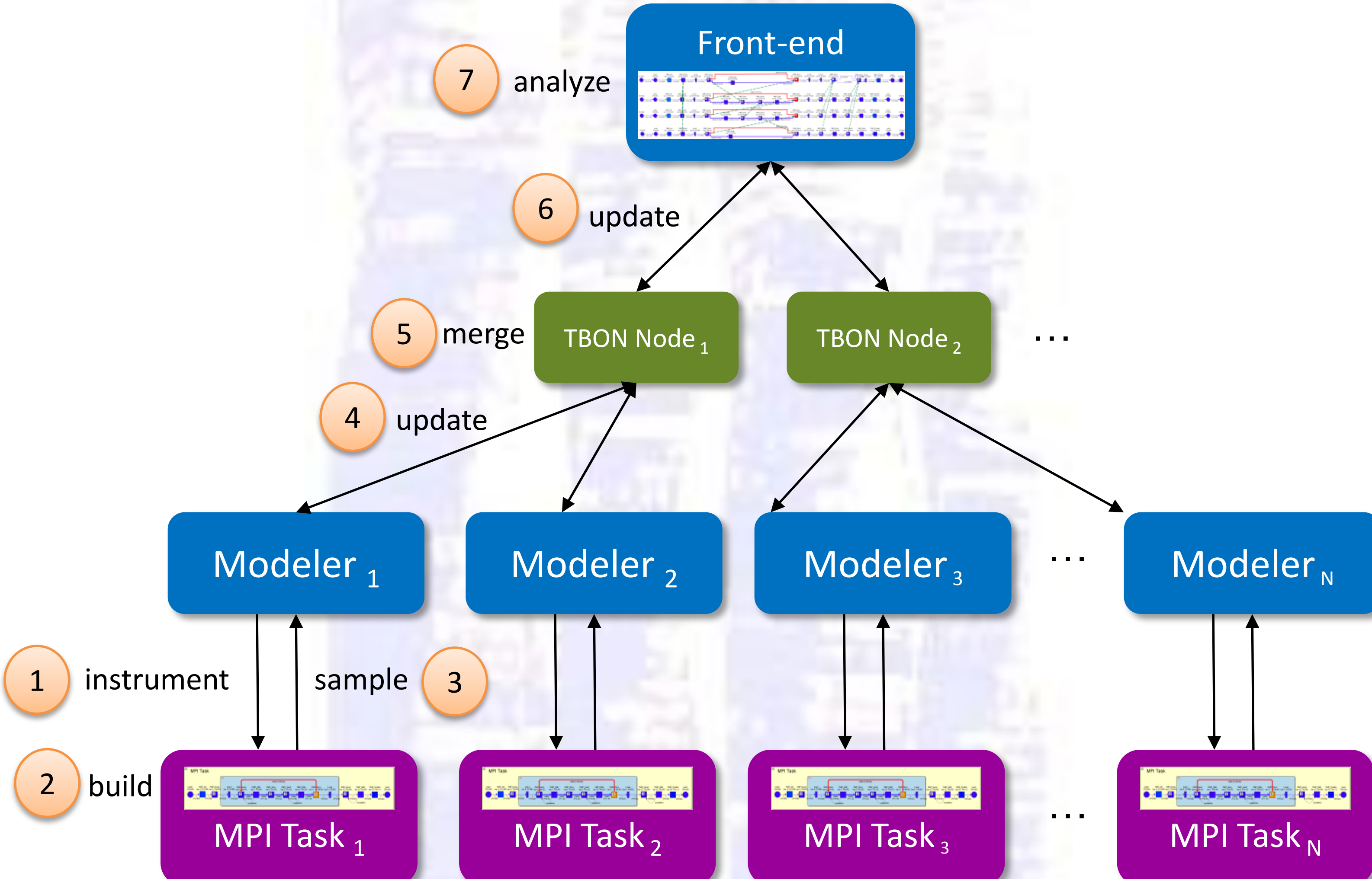
DMA: Building the Model

PTAG: Merging TAGs into parallel model

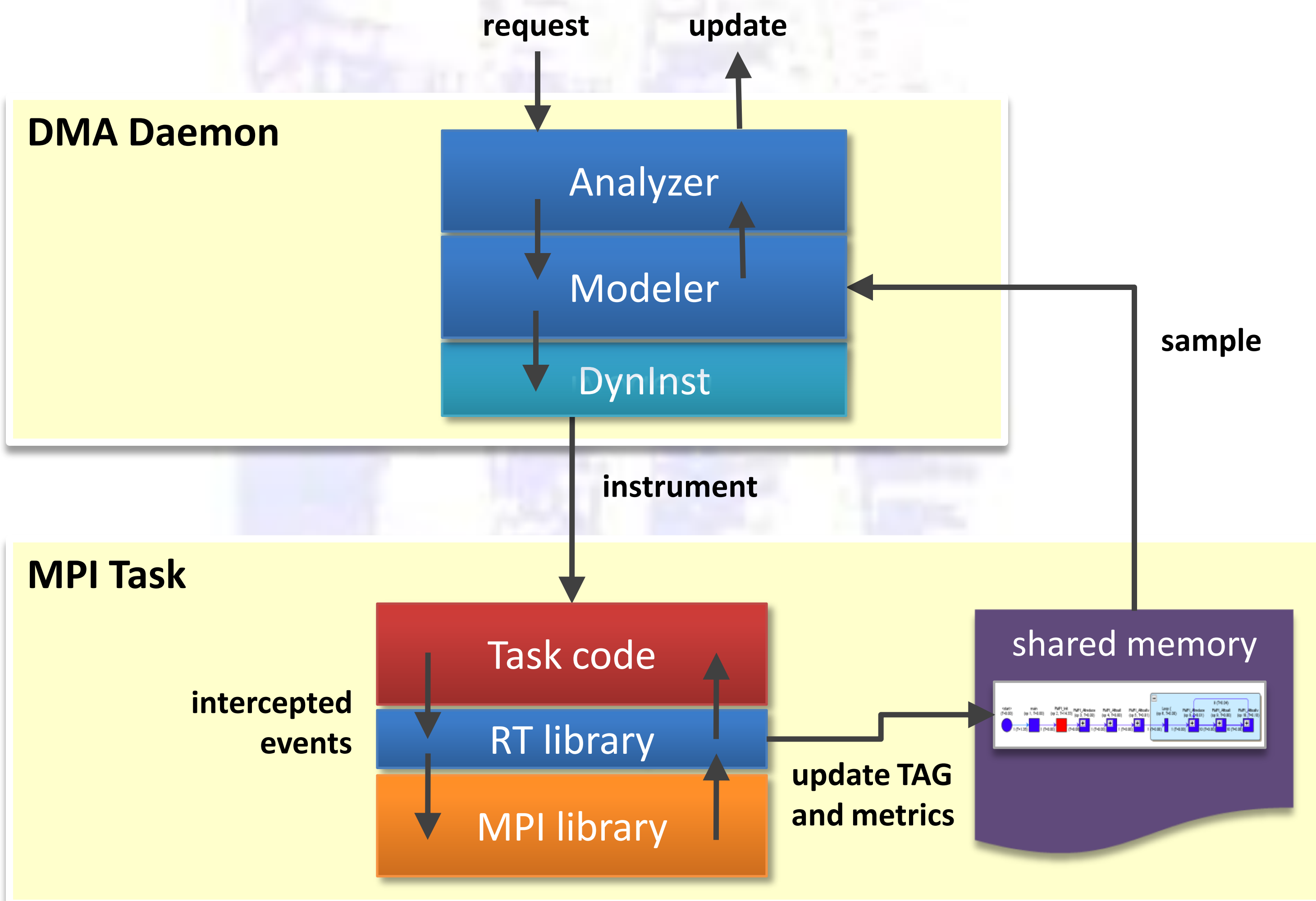
- Individual TAG models connected by **message edges** (P2P, Collective) enable construction of **Parallel-TAG** (PTAG)
- PTAG is updated periodically by sampling and merging TAGs



DMA: Tool architecture



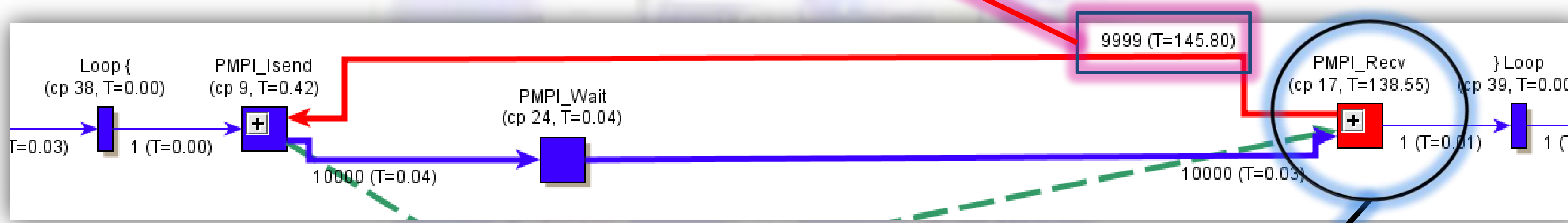
DMA: Tool daemon in-depth



DMA: Root-cause analysis

- Use TAG to identify bottlenecks in individual tasks
- Profile edges for non-communication problems
- Analyze transfer costs and synchronization issues for communication problems

CPU-bound activity
~45% time

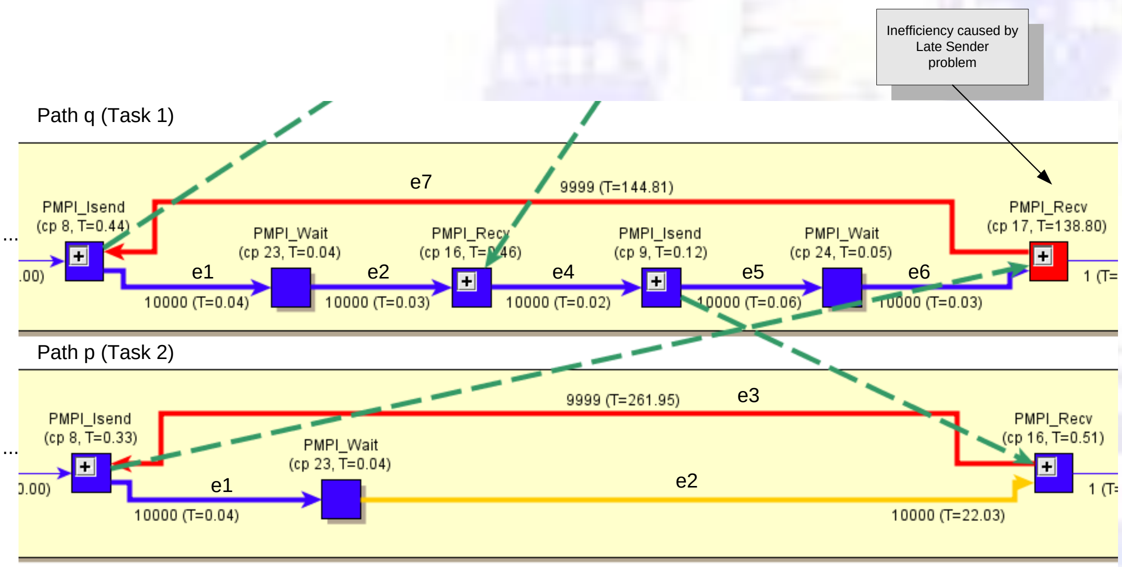


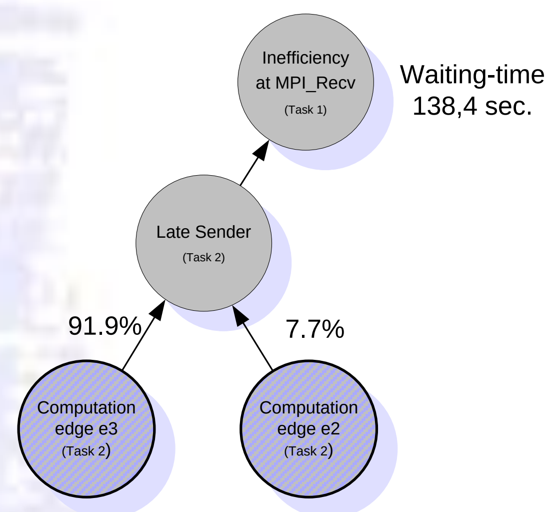
Blocked receive
~42% time

Communication or synchronization problem

DMA: Root-cause analysis

- Use PTAG to search for causes of communication latencies (nodes) by means of **cause-effect analysis**
- Latencies explained by differences in corresponding execution paths of communicating tasks





MATE and DMA

Installation

- GNU g++
- PVM 3.4 / Open MPI 1.2.x environment
- DynInst 5.1

- Contact:
  - MATE: Anna Morajko, e-mail: Anna.Morajko@uab.es
  - DMA: Oleg Morajko, e-mail: olegm@aia.ptv.es

Thank you for your attention