

A Flexible Approach to Staged Events

Tiago Salmito

tsalmito@inf.puc-rio.br

Ana Lúcia de Moura

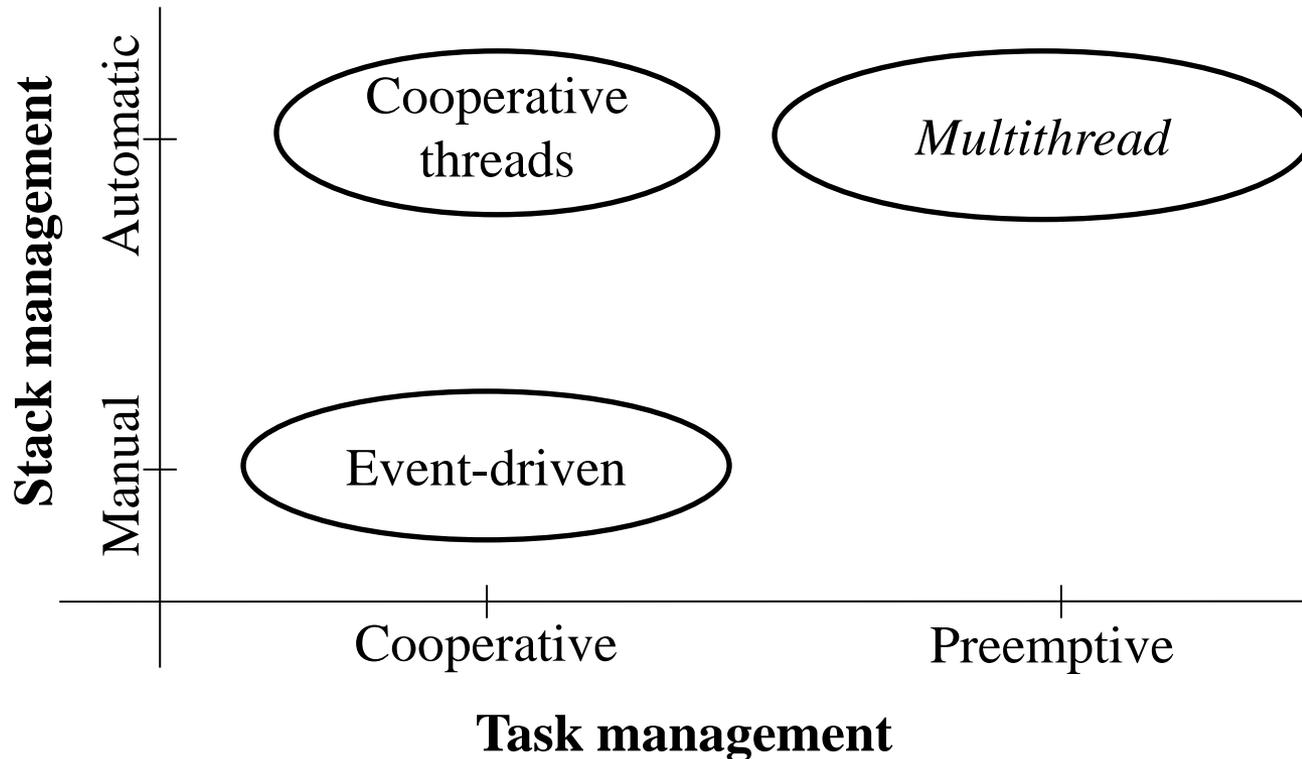
Noemi Rodriguez

6th International Workshop on Parallel Programming Models and
Systems Software for High-End Computing (P2S2)

October 1st 2013 – Lyon, France



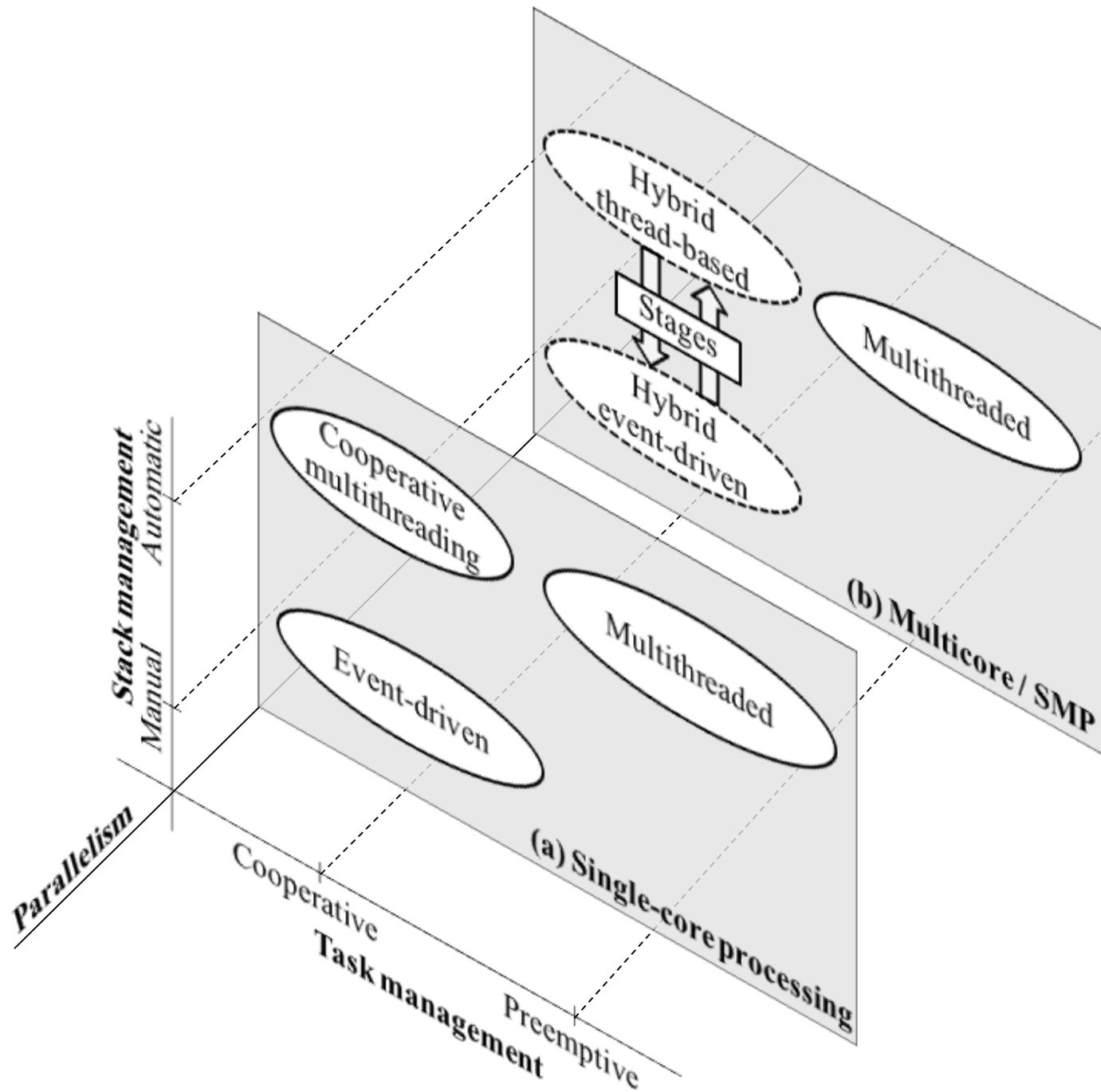
Concurrency Models



* A. Adya, J. Howell, M. Theimer, W. J. Bolosky and J. R. Douceur. *Cooperative Task Management Without Manual Stack Management*. (2002)

Hybrid Concurrency Models

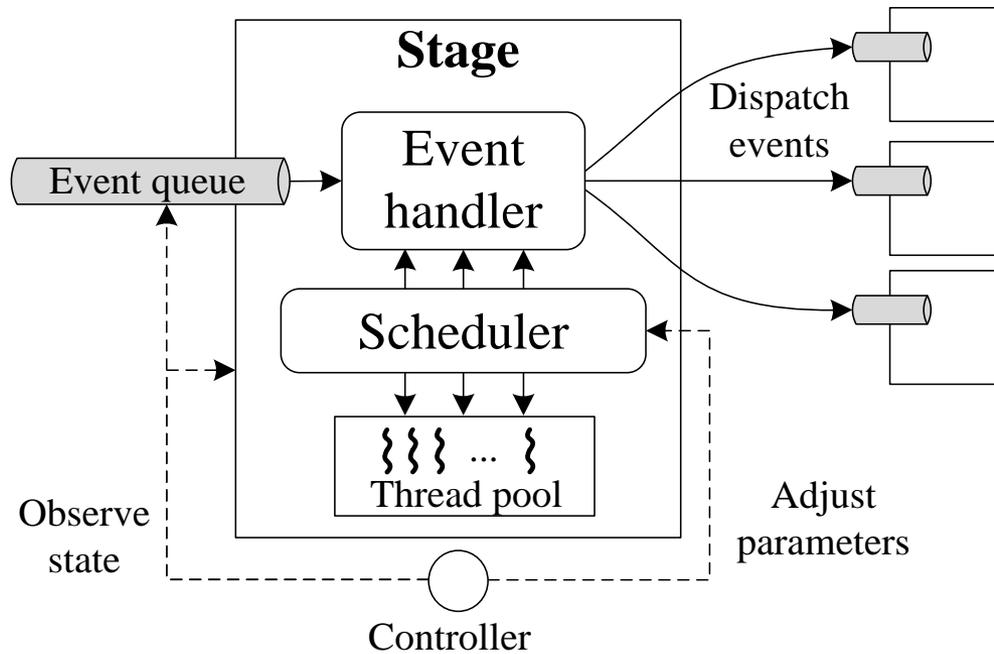
- Models combining threads and events
- Programing model bias:
 - Hybrid event-driven
 - More than one concurrent event loops
 - Hybrid thread based
 - Converts (user) threads to cooperative events during runtime
 - Staged event-driven
 - Does not have a clear bias towards events or threads
 - *Pipeline* processing



The Staged Model

- Inspired by SEDA
 - *Staged Event-Driven Architecture*
- Flexibility
 - Exposes both concurrency models
- Characteristics:
 - Applications are designed as a collection of stages
 - Stages are multithreaded modules
 - Asynchronous processing (event-driven communication)
- Decoupled scheduling
 - Local policies
 - Resource aware

Stages



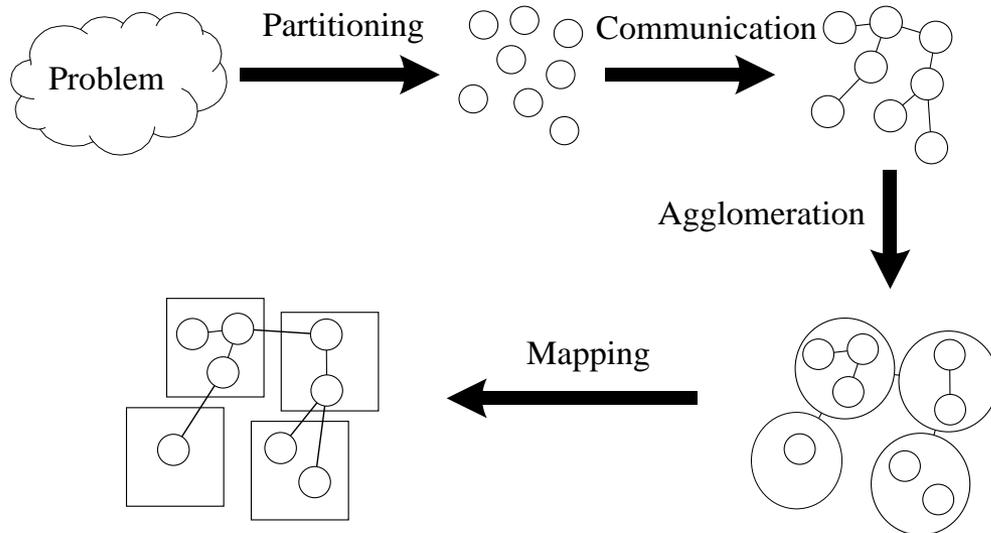
Stages: Some issues

- Coupling
 - Specification
 - Hinder reuse
 - Execution
 - Single (shared) address space
- Use of operating system threads
 - Thread sharing
- Local and global state sharing
 - Race conditions
 - Distributed resources

Extending the Staged Model

- Objective: Decoupling
 - Decisions related to the application logic and decisions related to the execution environment
- Characteristics
 - Stepwise application development
 - Stage composition and reuse
 - Cooperative execution with multiple threads

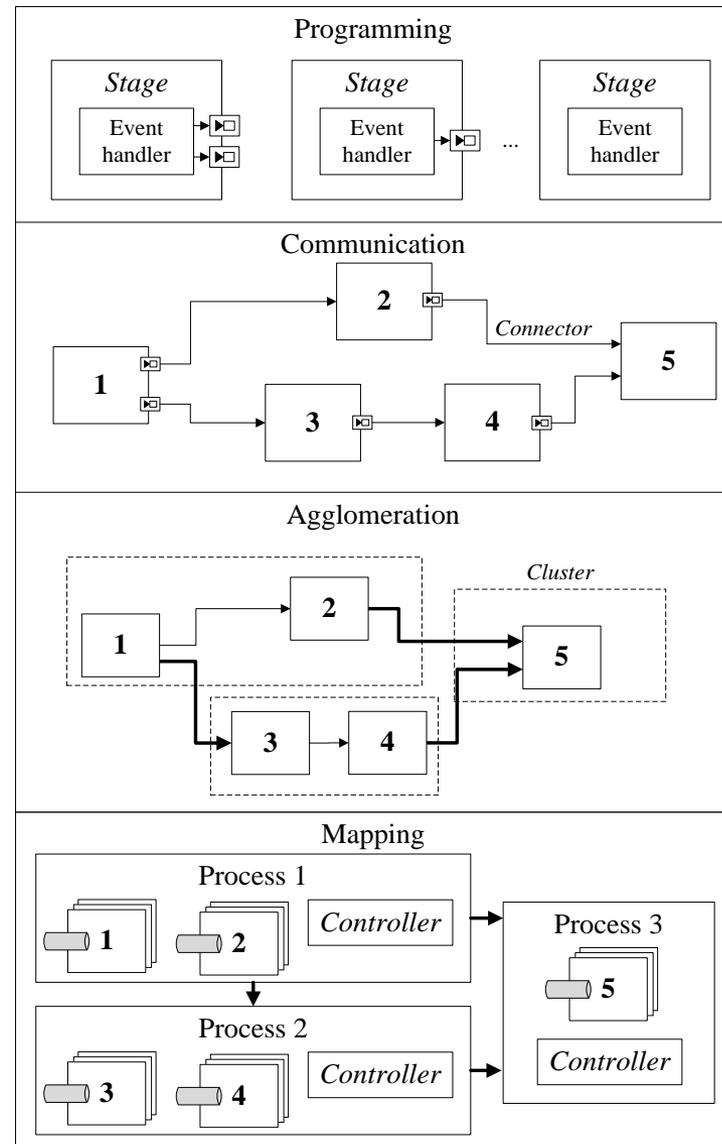
PCAM Design Methodology



- Partitioning
 - Functional or domain decomposition
- Communication
 - Data exchange
- Agglomeration
 - Processing and communication granularity
- Mapping
 - Mapping tasks to processors

Stepwise development

- Programming Stages
 - Functional decomposition
 - State isolation
 - Transient state
 - Domain decomposition
 - Persistent state
 - Atomic execution
- Communication
 - Connectors: Application graph
 - Output ports and event queues
- Agglomeration
 - Clusters of stages
 - Scheduling domain
- Mapping
 - Execution locality



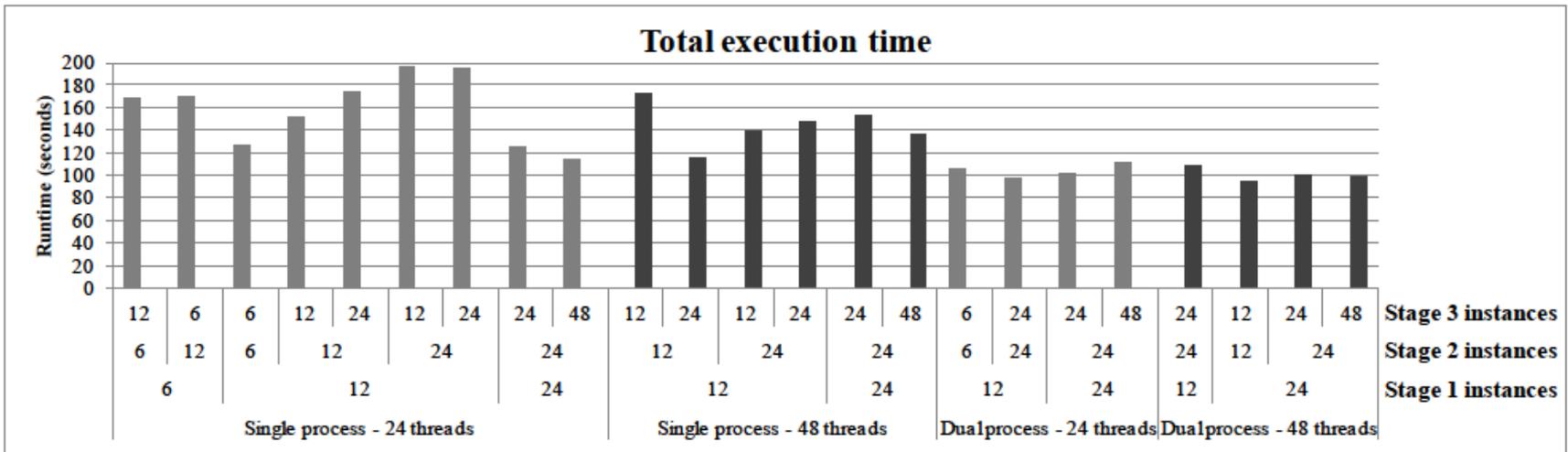
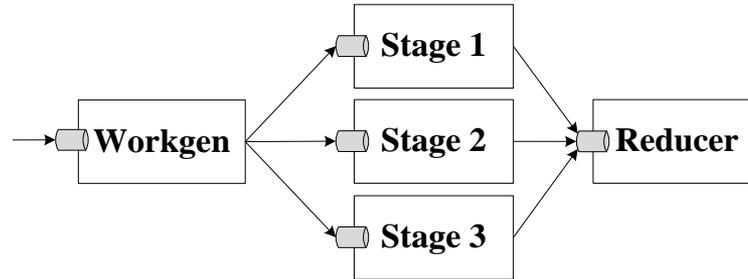
Leda

- Distributed platform for staged applications
- Implemented in C and Lua
 - Scripting environment
 - Use of C for CPU-intensive operations
- Declarative application description
 - Application graph
 - Execution configuration

Example: echo server

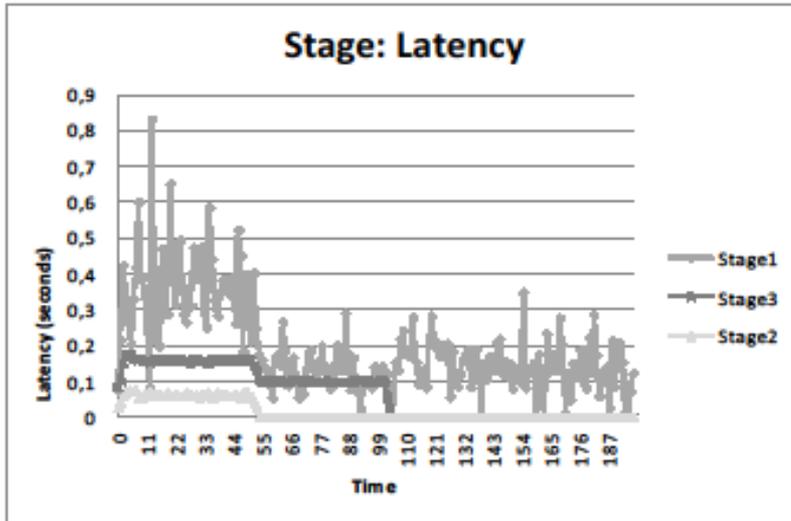
```
require 'leda'
local port=5000
local server=leda.stage{
  handler=function()
    local server_sock=assert(socket.bind("*", port))
    while true do
      local cli_sock=assert(server_sock:accept())
      leda.send("client",cli_sock)
    end
  end,
  init=function() require'leda.utils.socket' end,
}:push()
local reader=leda.stage{
  handler=function(sock)
    repeat
      local msg,err=sock:receive()
      leda.send("message",msg)
    until msg==nil
  end
}
local echo=leda.stage(function(msg) print(msg) end)
local graph=leda.graph{
  server"client"..reader,
  reader"message"..echo
}
graph:run()
```

Evaluation

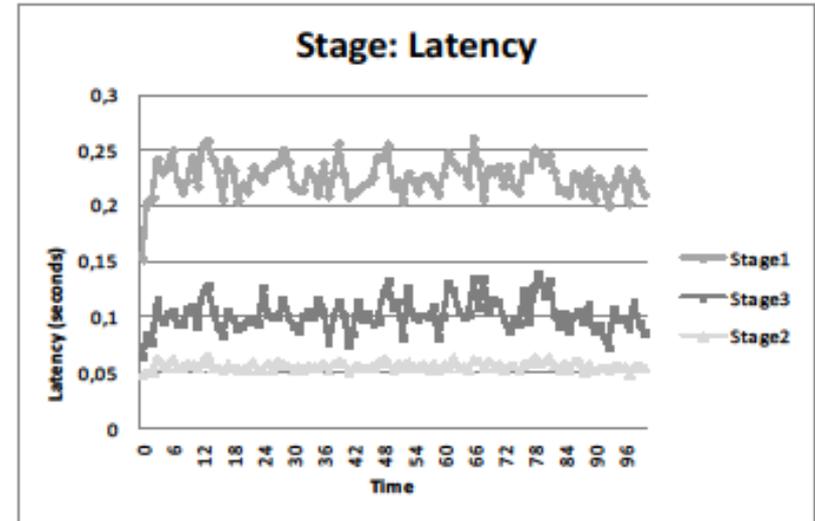


Internal statistics

Worst case scenario



Best case scenario



Final Remarks

- Hybrid concurrency
 - Event-driven, thread based or staged
- An extension to the staged model
 - Stepwise application development
- Implementation of a distributed platform for staged applications
 - Leda

A Flexible Approach to Staged Events

Tiago Salmito

tsalmito@inf.puc-rio.br

Ana Lúcia de Moura

Noemi Rodriguez

6th International Workshop on Parallel Programming Models and
Systems Software for High-End Computing (P2S2)

October 1st 2013 – Lyon, France



Extra: Runtime Architecture

