Tracing and Monitoring Distributed Multi-Core Systems Project
- Progress Meeting -

User Space Trace Abstraction Techniques

Heidar Pirzadeh and Wahab Hamou-Lhadj
Software Behaviour Analysis Lab
{s_pirzad, abdelw}@ece.concordia.ca

Montreal, QC
Dec. 09, 2011

Progress

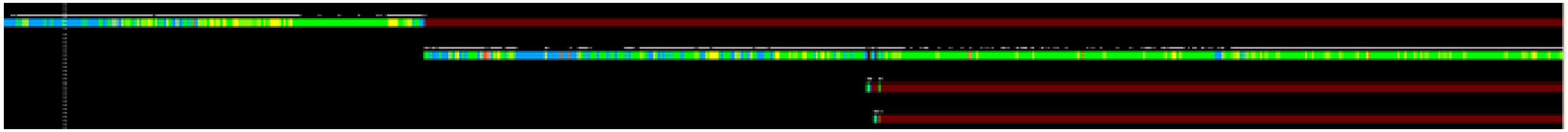
- Trace abstraction:
  - We continued to develop trace abstraction techniques for user space traces
  - Explored the use of state information in trace abstraction and exploration
  - Developed techniques for automatically extracting important content from a trace
- Anomaly detection:
  - Investigation of different tracing mechanisms
  - Reduction of learning time in building models
  - Reduction of false positives
  - Development of a taxonomy of attacks on the Linux kernel

Our Approach for Trace Abstraction

- Based on the extraction of execution phases from large traces
- What is an execution phase?
  - A segment of program's execution that performs a specific task
- Trace Segmentation: Automatically divide a trace into phases
  - Allow SW engineering to browse traces as a flow of execution phases rather than mere sequence of events

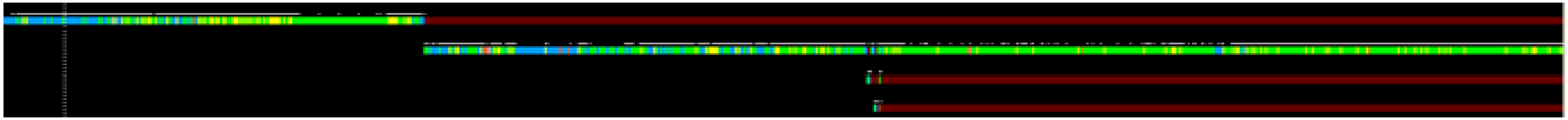
Example

- A trace generated from a compiler will contain the various compiler's phases including parsing, preprocessing, lexical analysis, semantic analysis, etc.
- In most visualization tools, it will look like:

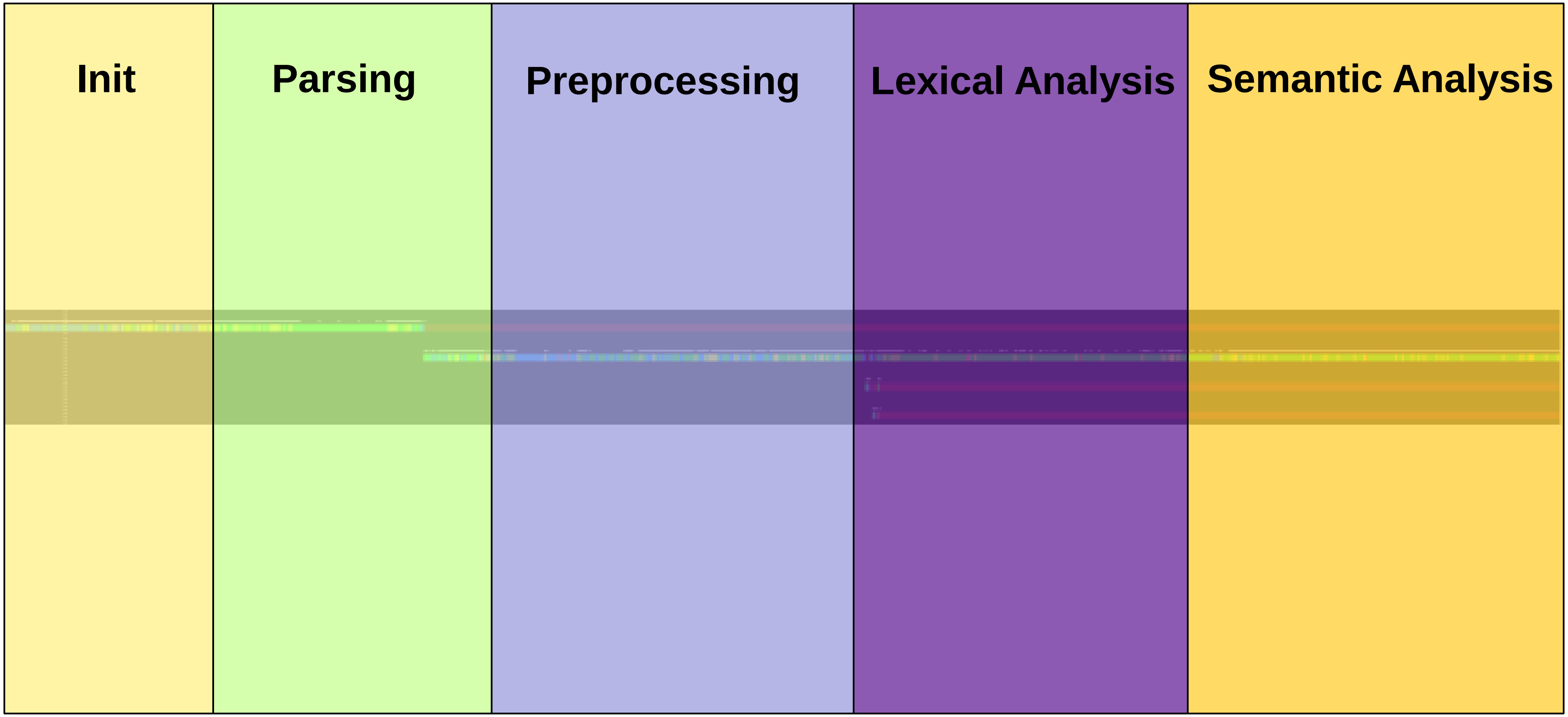


- But how can we tell what happens where?

Visually…



Visually…



Init

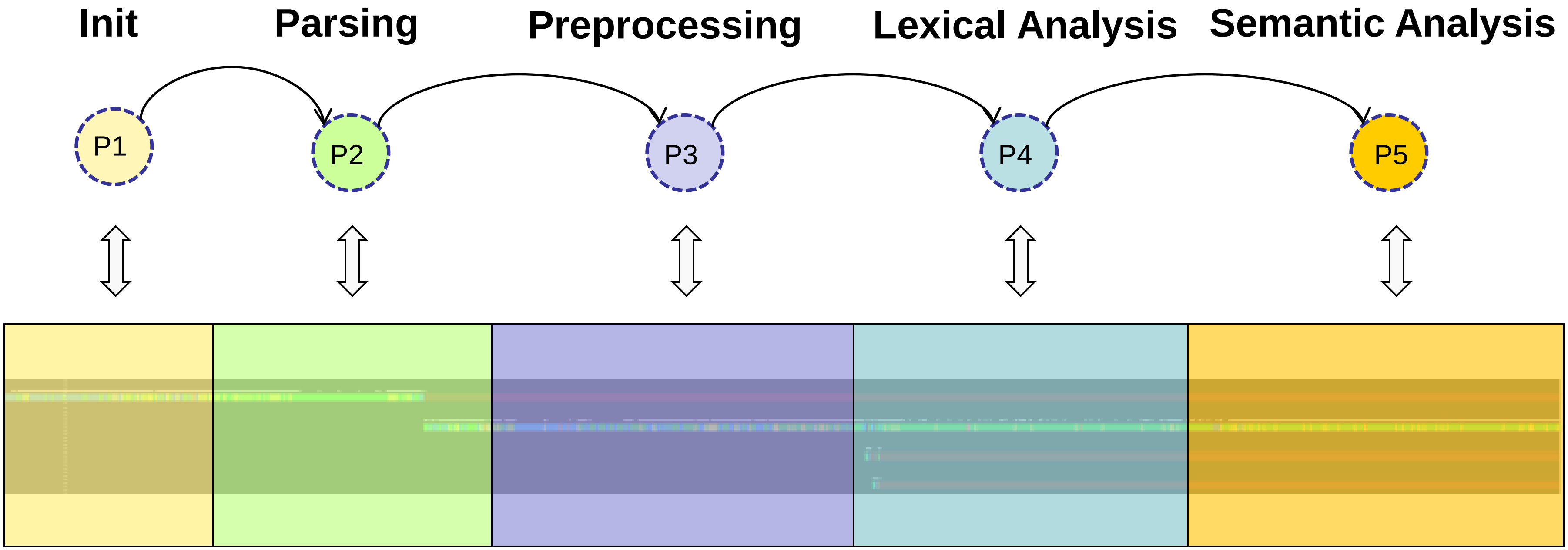
Parsing

Preprocessing

Lexical Analysis

Semantic Analysis

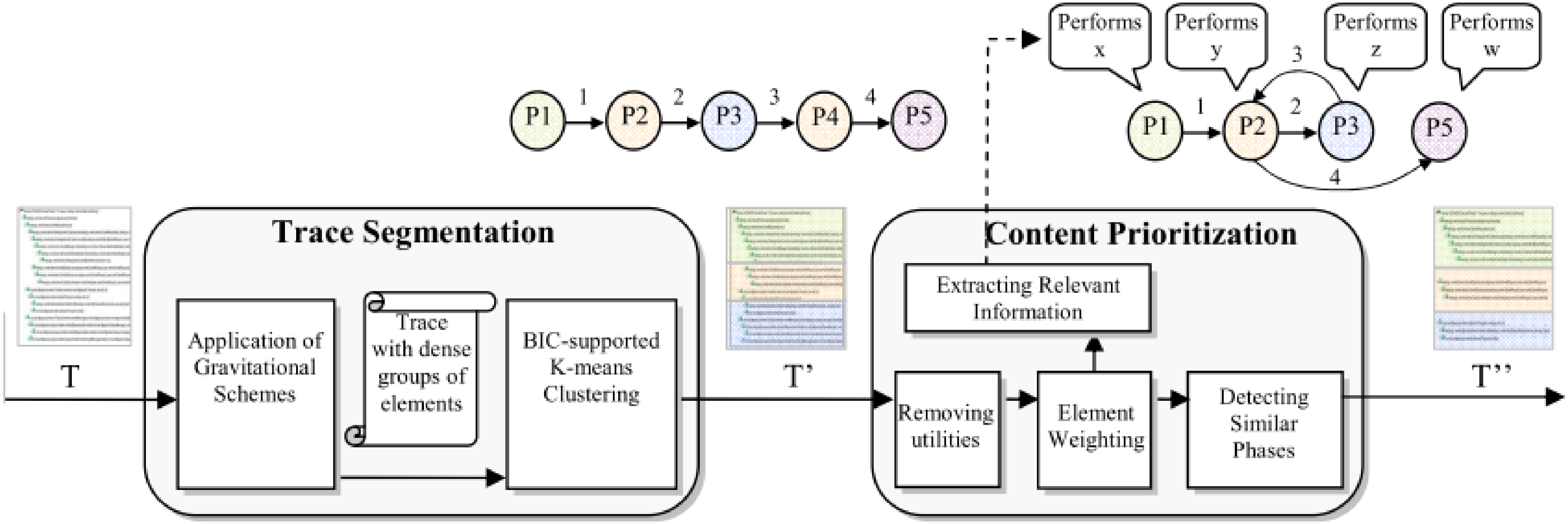
A different view…
Nested phases can be added



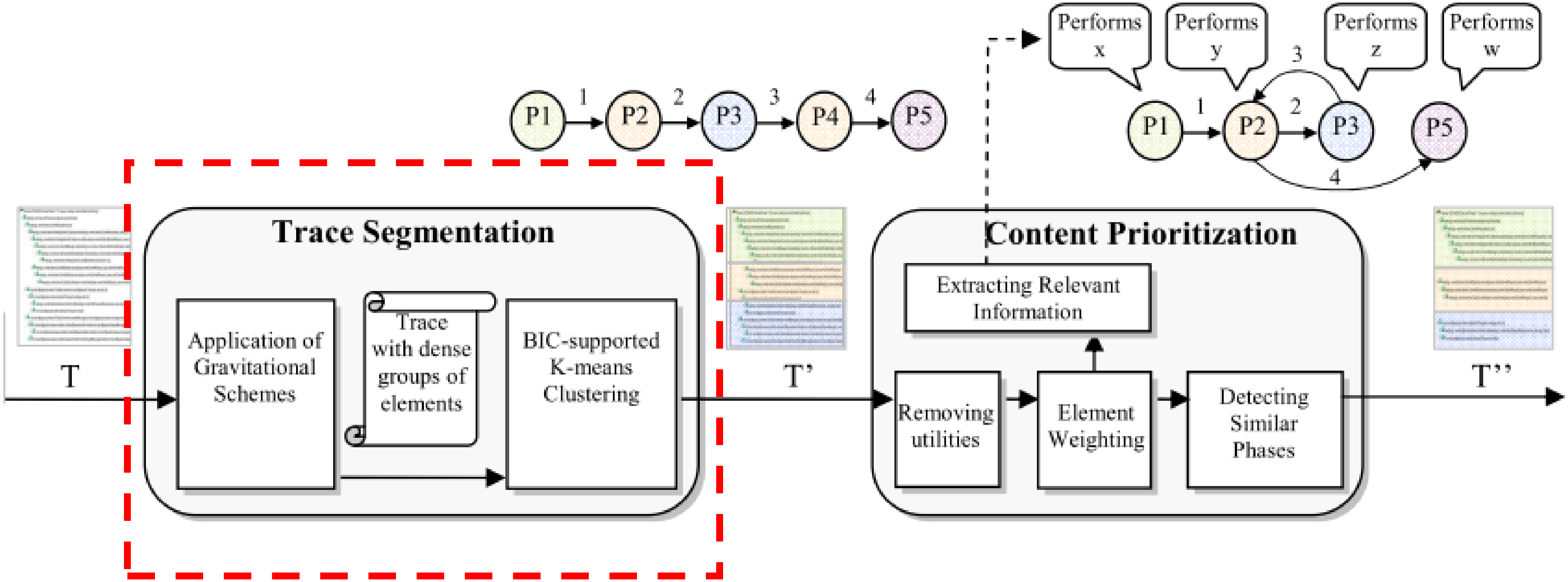
Research Questions?

- How can we automatically extract execution phases from a trace?
- What additional information states can reveal about execution phases?
- How can we extract the main components that implement a specific phase?
- Can we use execution phases to further reduce the size of traces?

Approach: Trace Abstraction Framework



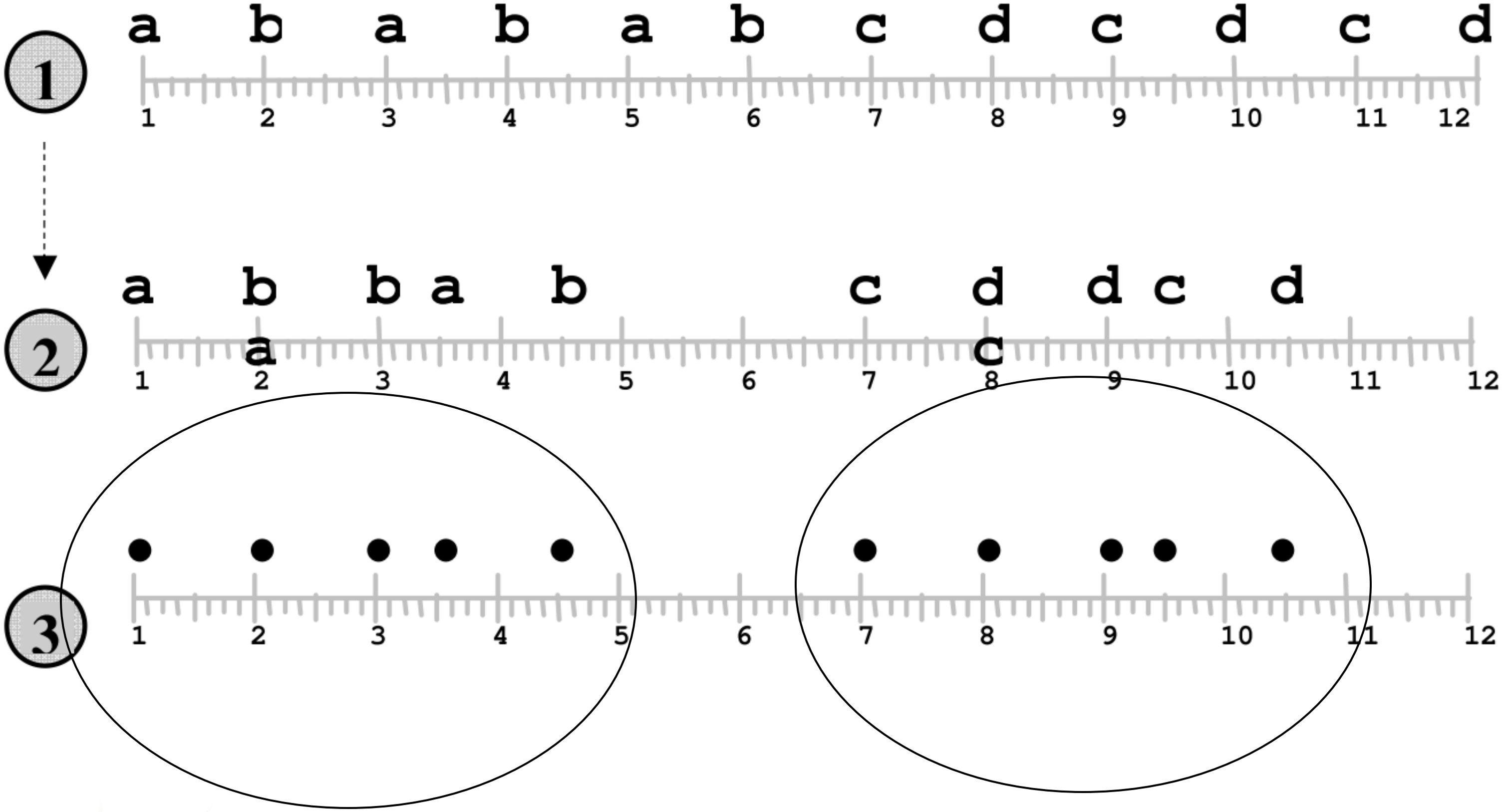
Our Approach: Trace Abstraction Framework



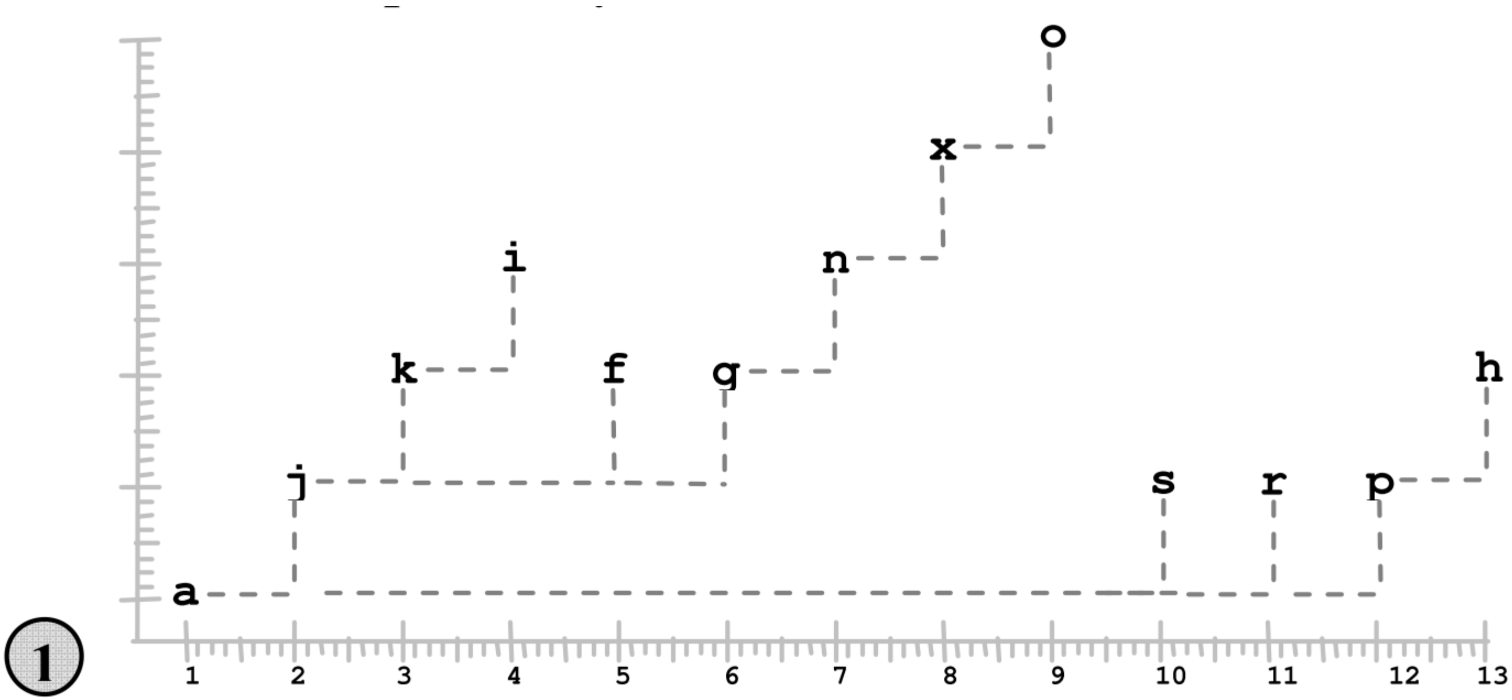
Trace Segmentation Approach

- The scientific foundation comes from the study of the human perception system
  - The ability for humans to group similar items to form objects and shapes
  - Explained using the Gestalt laws of similarity and continuity

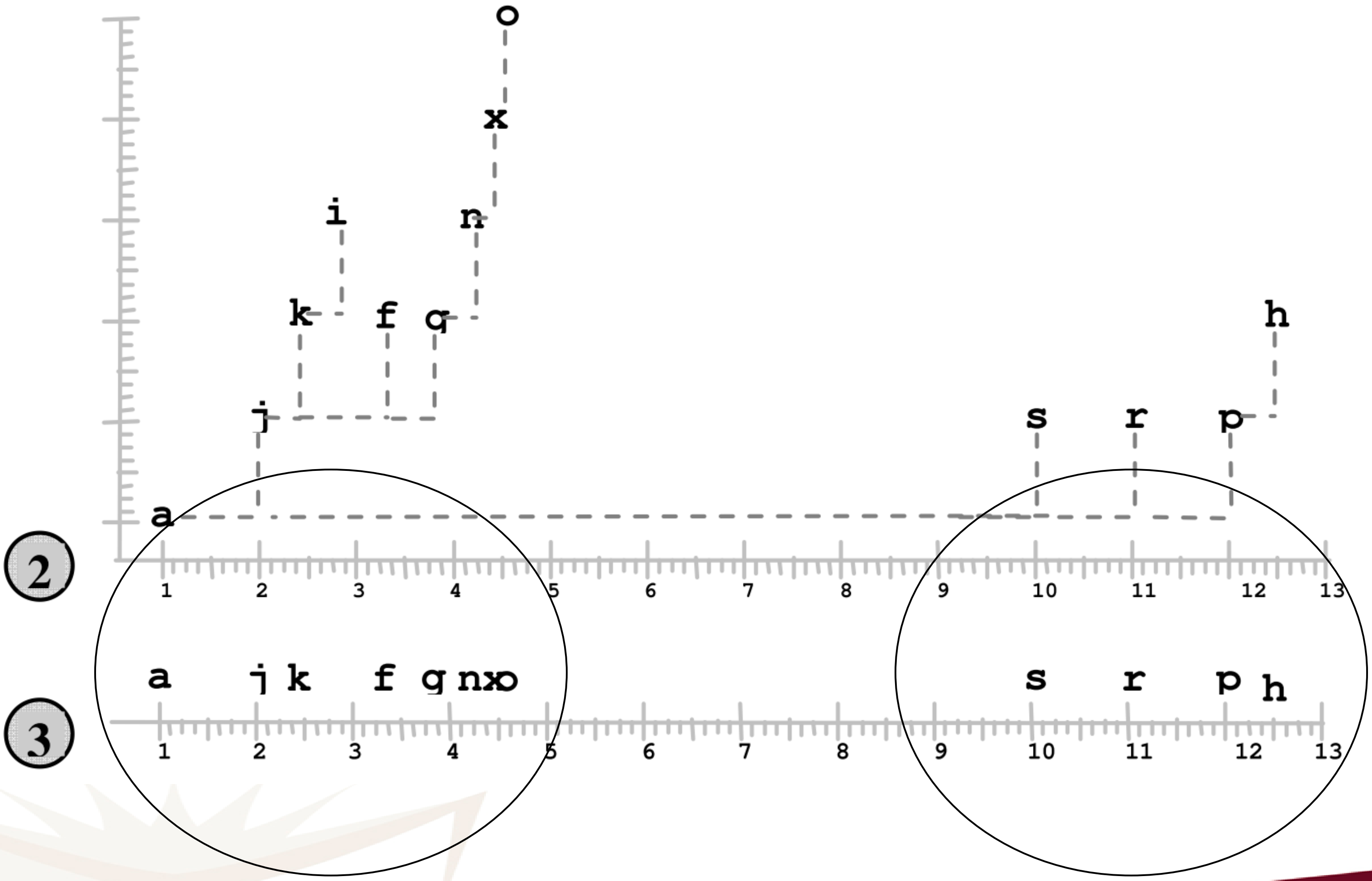
Measuring Similarity



Measuring Continuity in Traces with Nesting Levels

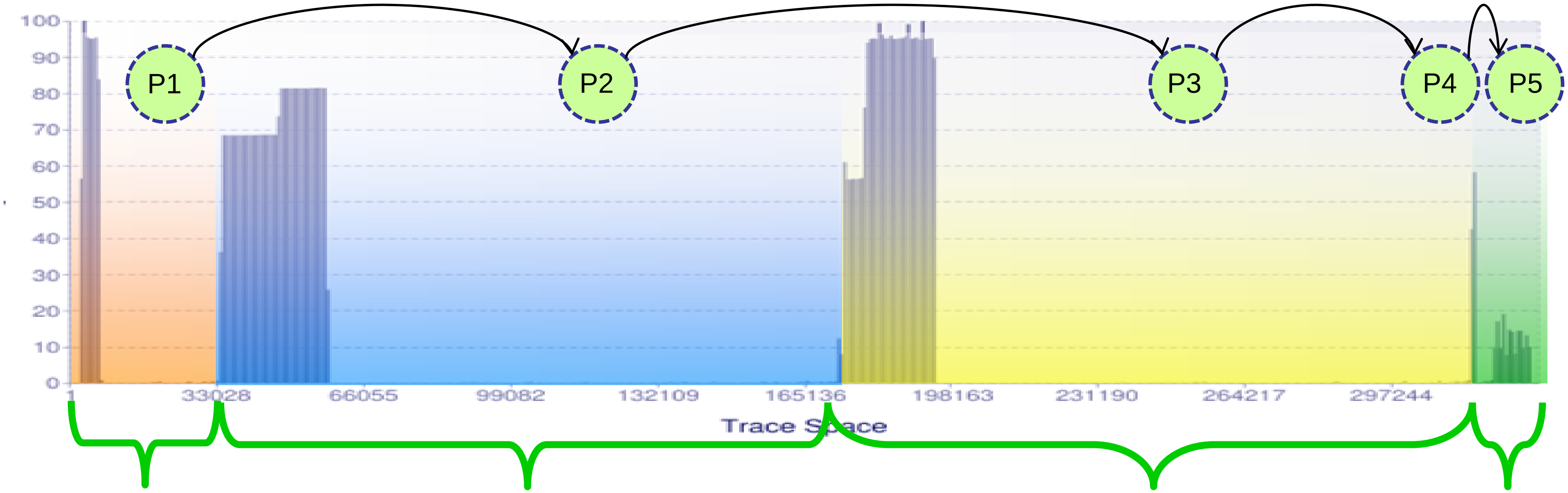


Measuring Continuity in Traces with Nesting Levels



Case Study

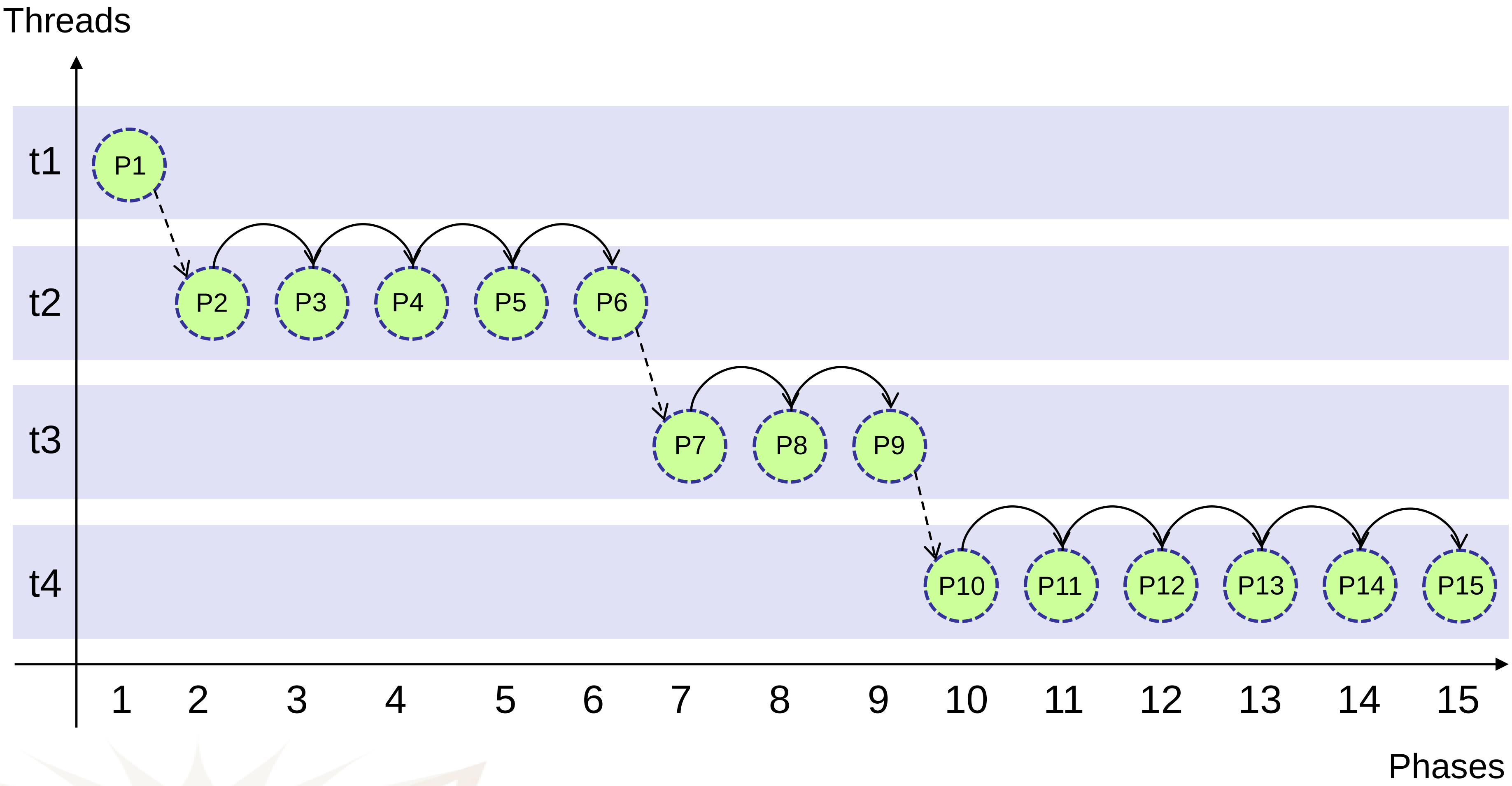
Program: WEKA 3.6.6
Scenario: building a decision tree learning algorithm for classifying data instances.
Trace: Multi-threads 1,571,214 events



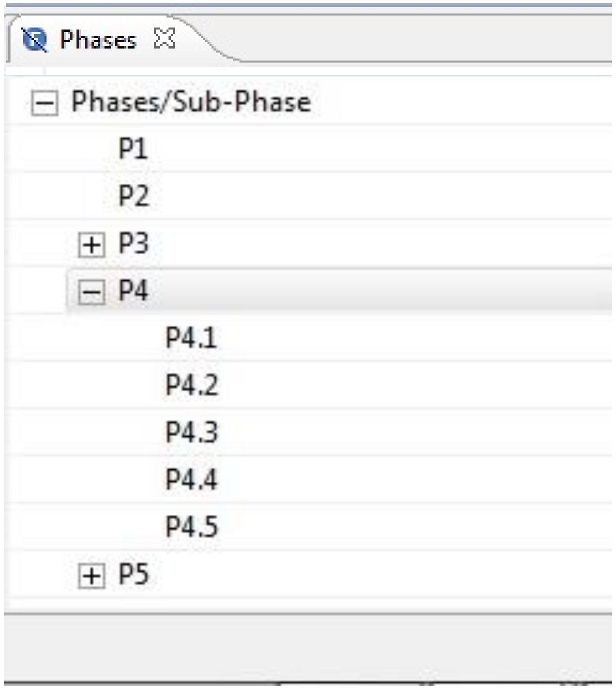
P1 P2 P3 P4 P5

Phase 1 Phase 2 Phase 3 Phases 4, 5

Phase flow diagram of a Weka trace

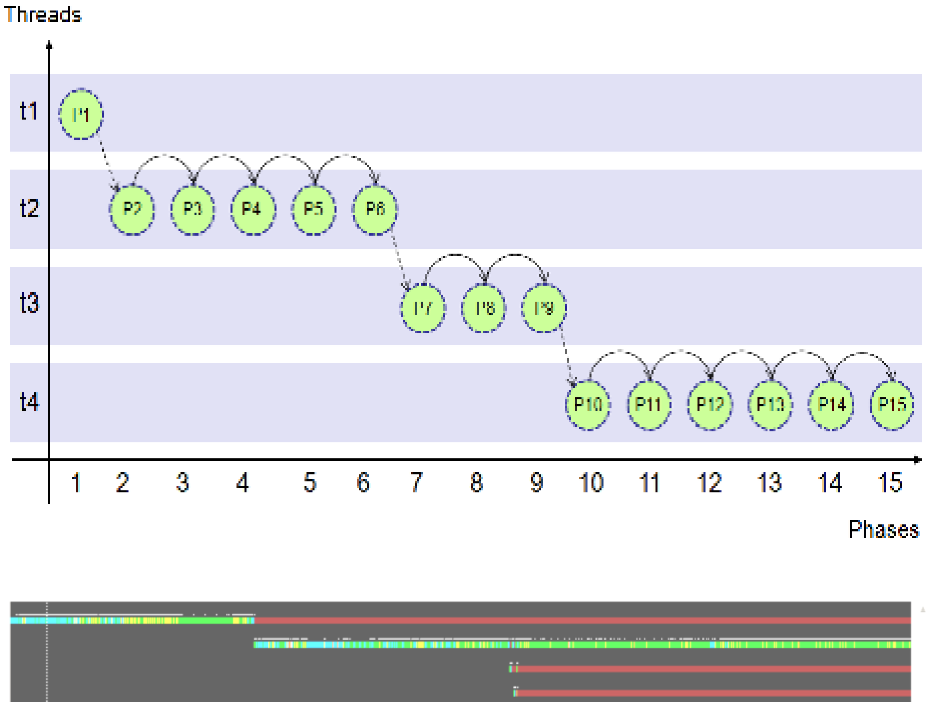


Adding phase views to a tool



Phases ⊠

- ⊟ Phases/Sub-Phase
  - P1
  - P2
  - ⊞ P3
  - ⊟ P4
    - P4.1
    - P4.2
    - P4.3
    - P4.4
    - P4.5
  - ⊞ P5



State Information

- What is a state?
  - The state of the system is the state value of every attribute in the system
  - State has a duration
  - State value, which can really be anything
- Attributes in the kernel-trace state system :
  - CPUs
  - CPUs/0
  - CPUs/0/current_thread
  - Etc.

[1] http://www.dorsal.polymtl.ca/blog/alexandre-montplaisir/introduction-state-history

State Change

Consists of three things:

- timestamp
- attribute
- state value

The state of 'attribute' changed to 'state value'
at time 'timestamp'

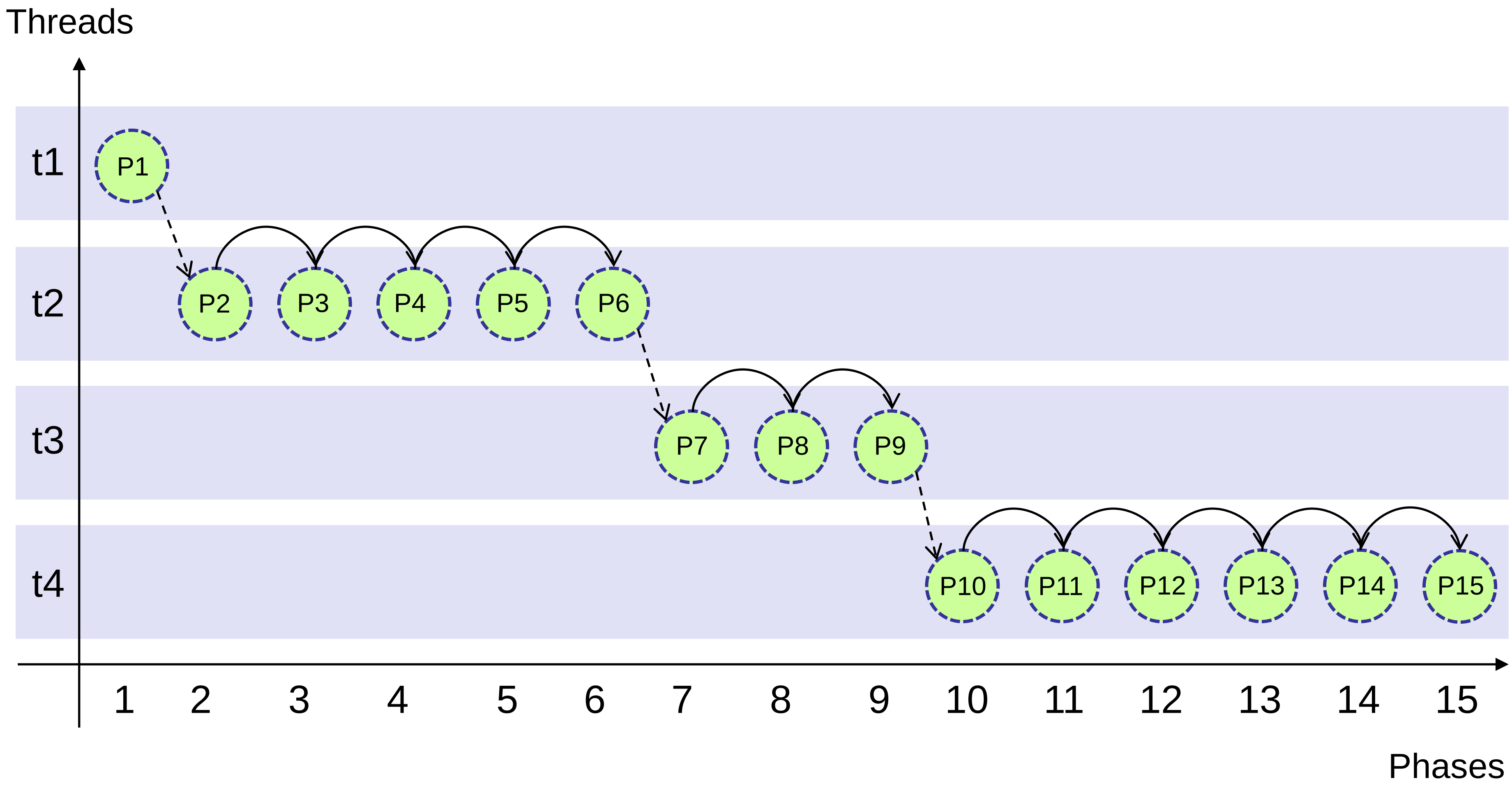
[1] http://www.dorsal.polymtl.ca/blog/alexandre-montplaisir/introduction-state-history

Existing Info

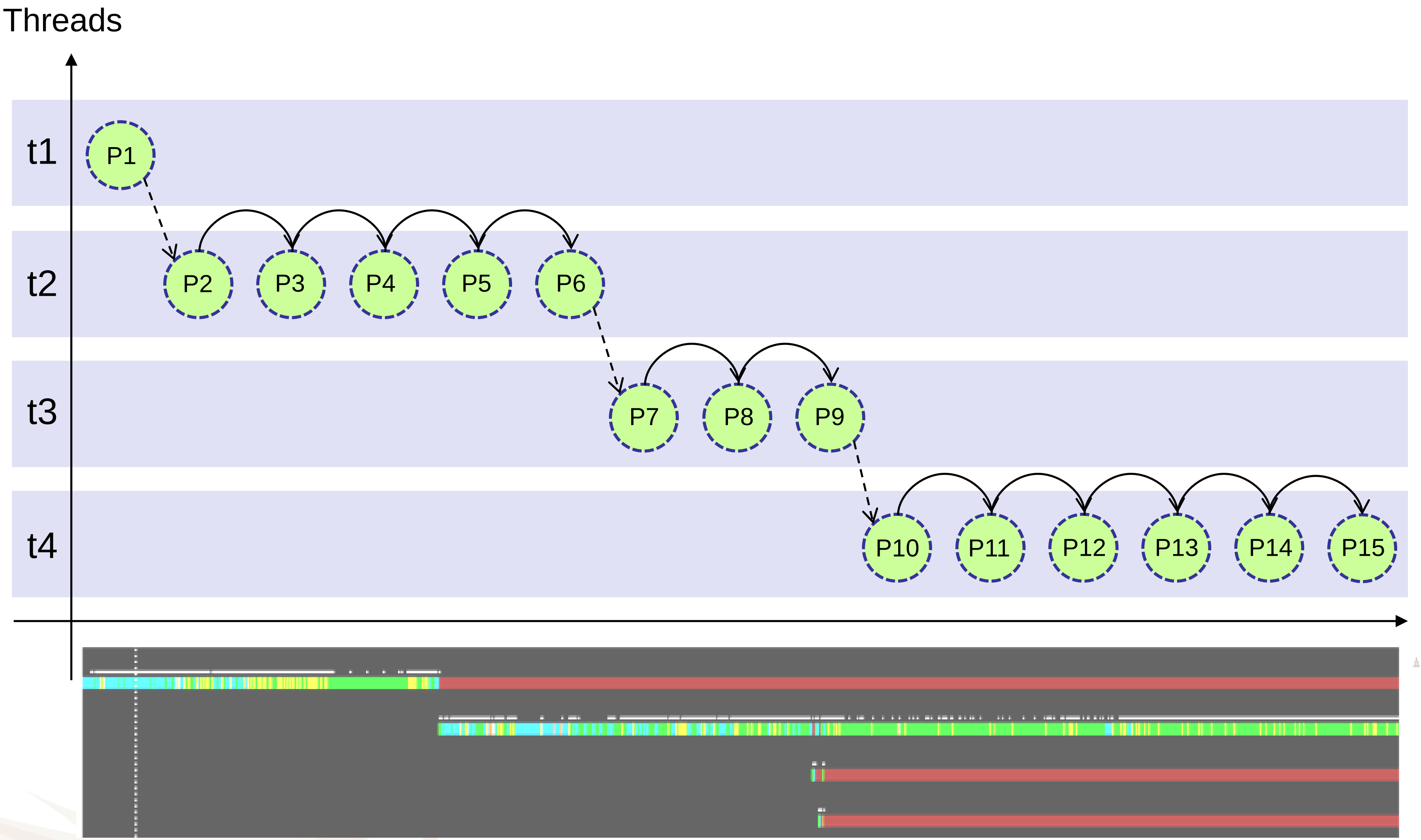
LTTNG Kernel Space Trace:

- Timestamp
- Event (page fault)
- Process ID
- CPU ID
- File Descriptor

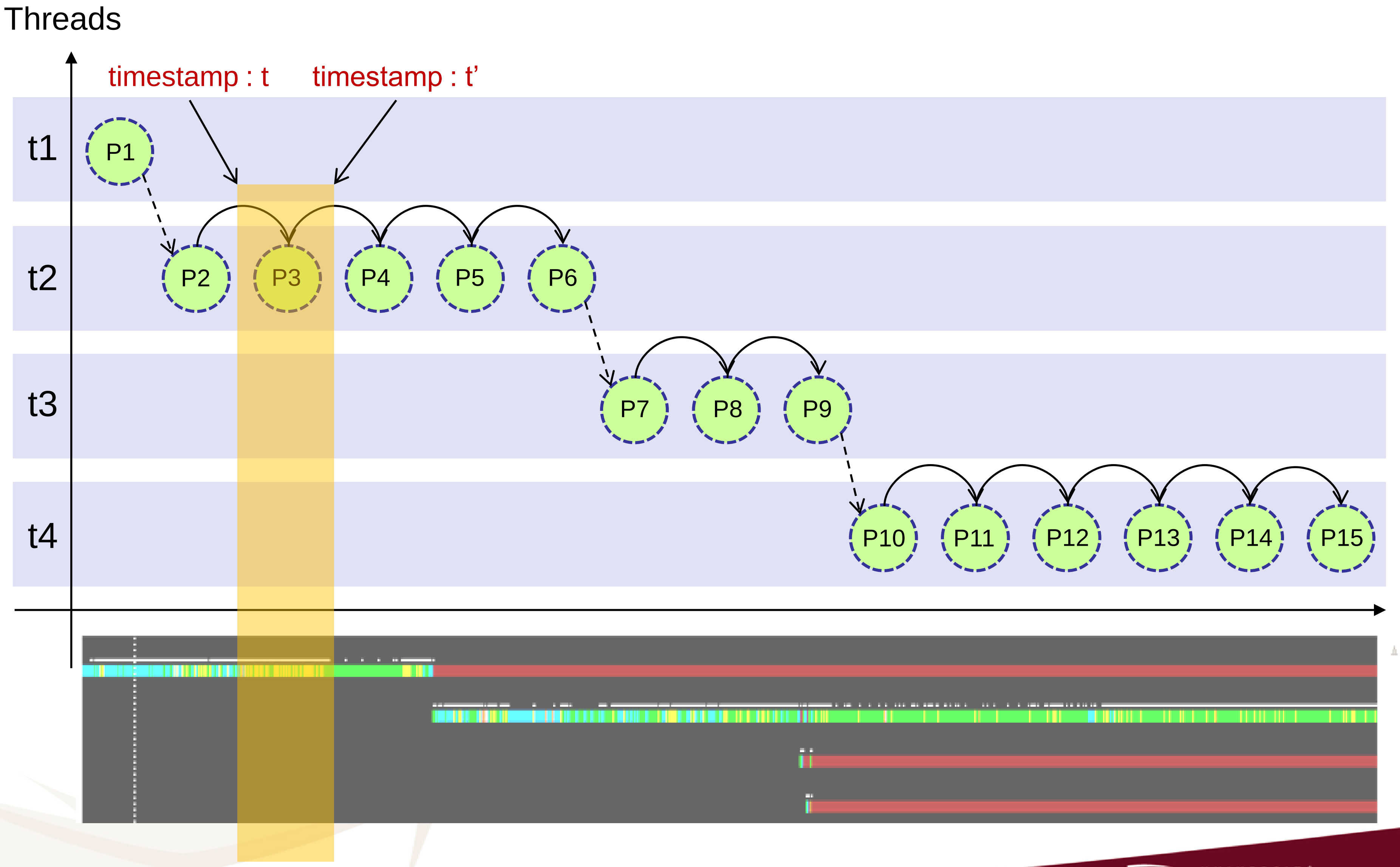
Phase Flow


Threads
t1
t2
t3
t4
P1
P2
P3
P4
P5
P6
P7
P8
P9
P10
P11
P12
P13
P14
P15
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
Phases

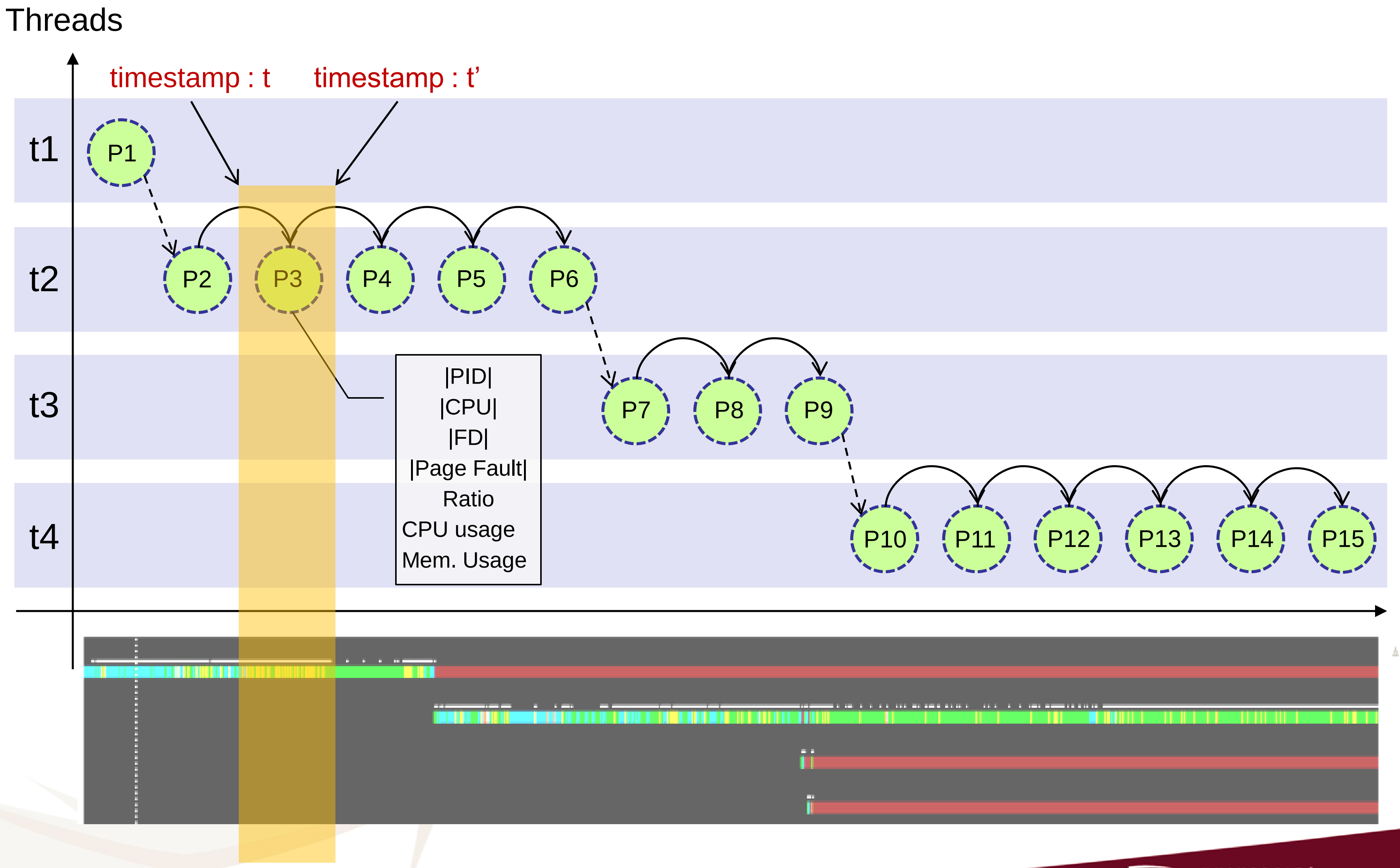
Phases Mapped to Kernel Space Trace



Phases Mapped to Kernel Space Trace

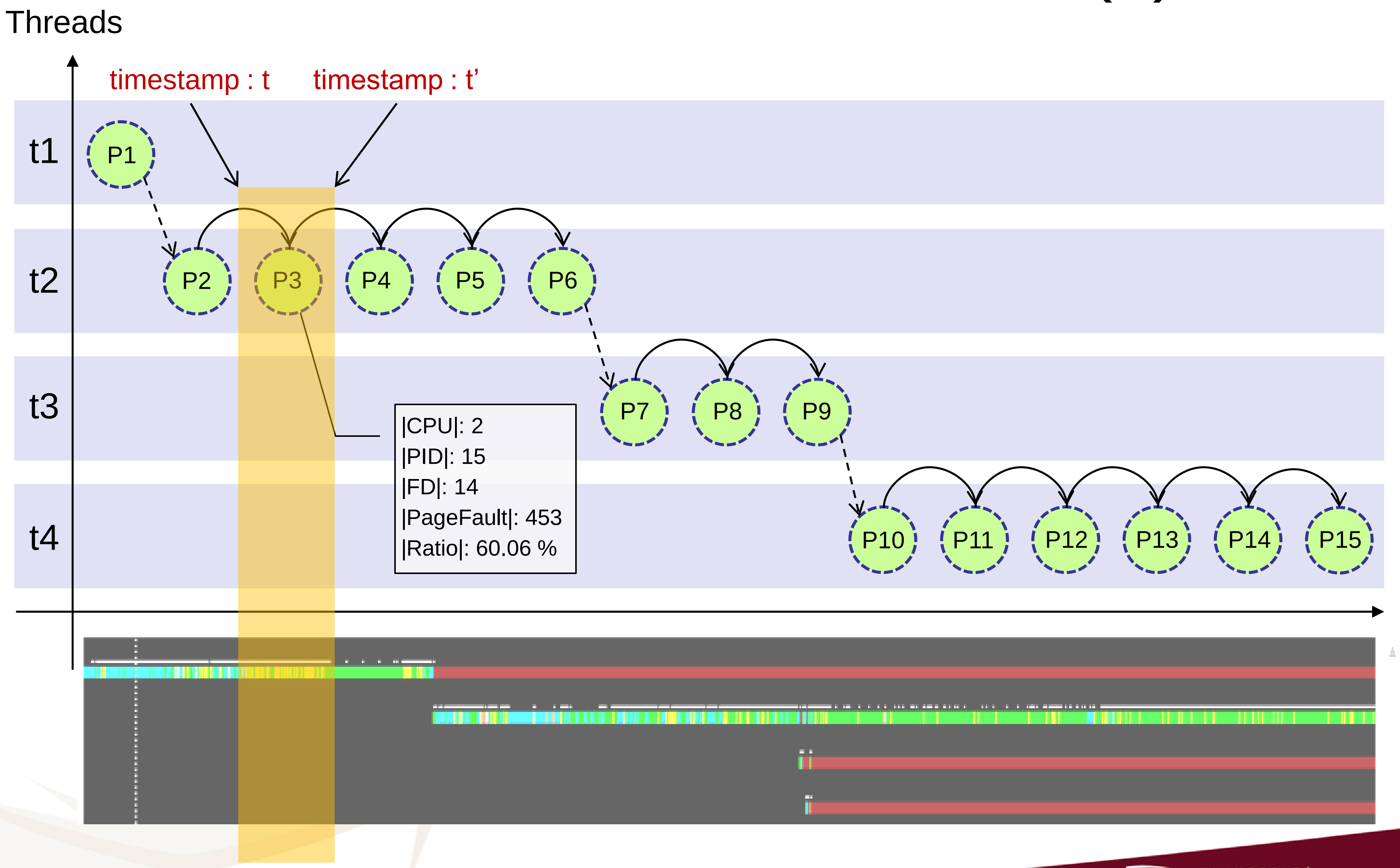


Phases Enriched with State Info



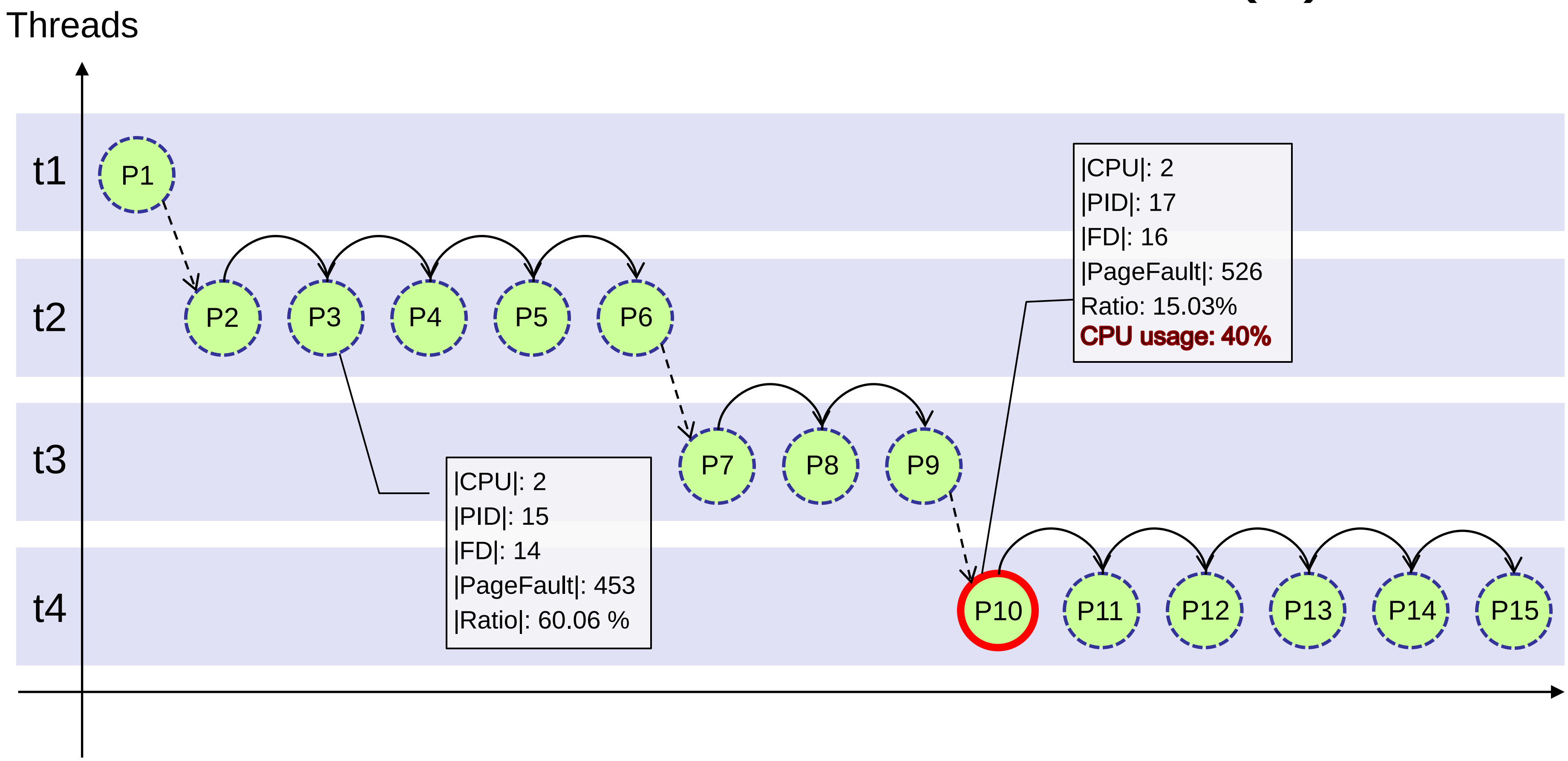
|PID|
|CPU|
|FD|
|Page Fault|
Ratio
CPU usage
Mem. Usage

Phases Enriched: Statistics (1)



|CPU|: 2
|PID|: 15
|FD|: 14
|PageFault|: 453
|Ratio|: 60.06 %

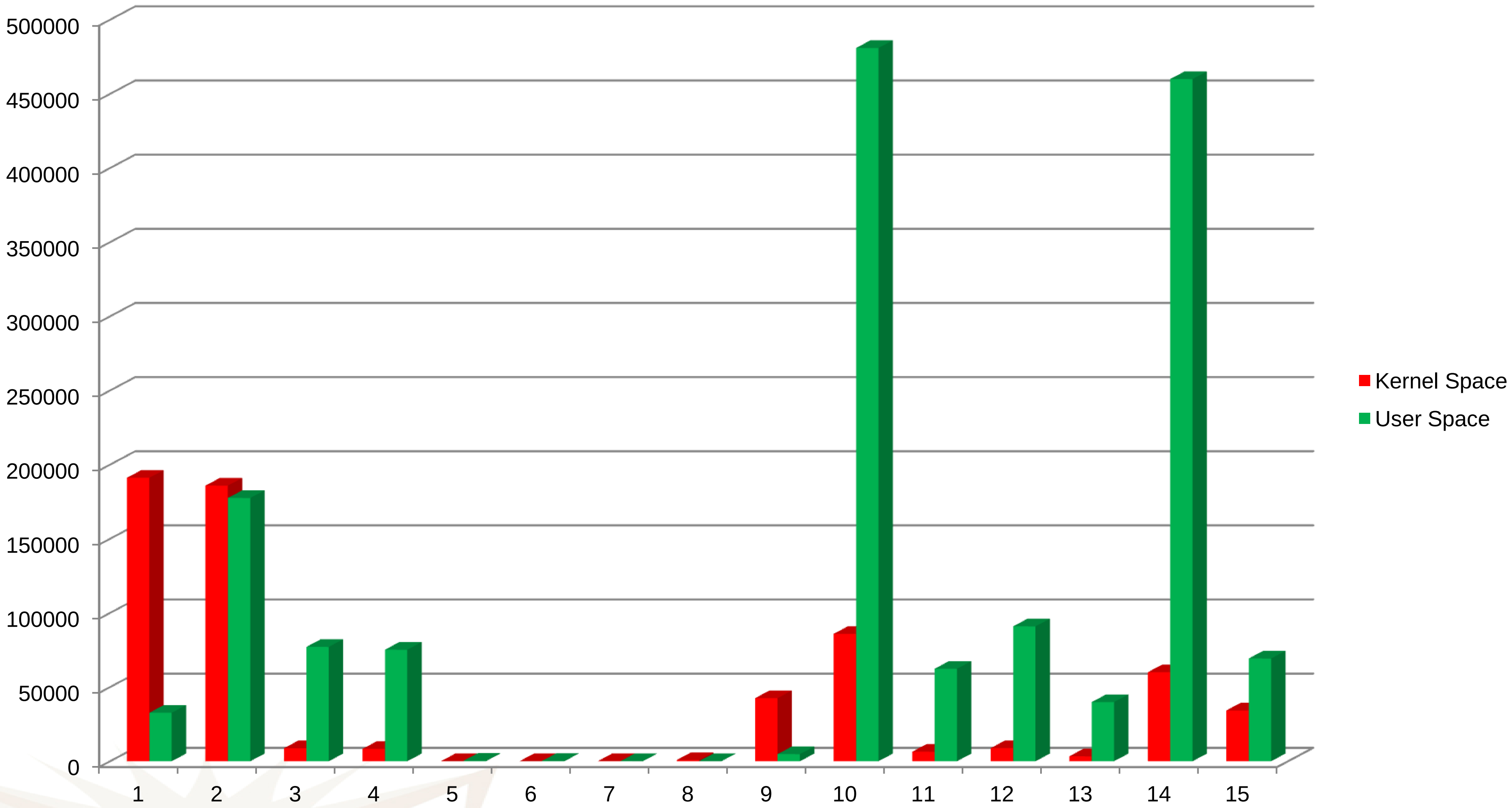
Phases Enriched: Statistics (2)



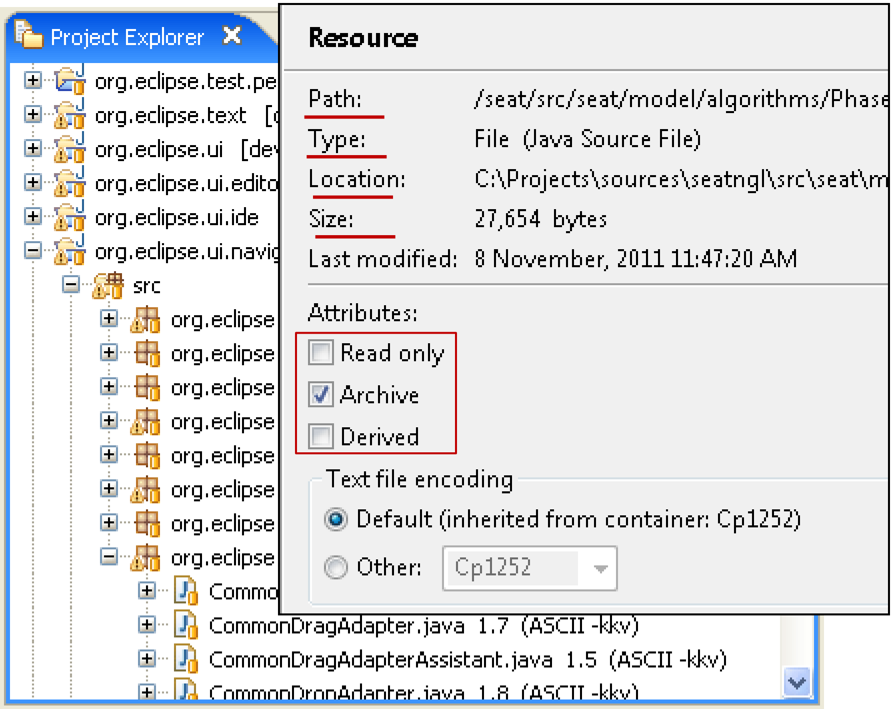
|CPU|: 2
|PID|: 15
|FD|: 14
|PageFault|: 453
|Ratio|: 60.06 %

|CPU|: 2
|PID|: 17
|FD|: 16
|PageFault|: 526
Ratio: 15.03%
CPU usage: 40%

Comparison: Kernel Space vs User Space



Enriched Phase View



Project Explorer

- org.eclipse.test.pe
- org.eclipse.text [d
- org.eclipse.ui [de
- org.eclipse.ui.edito
- org.eclipse.ui.ide
- org.eclipse.ui.navig
  - src
    - org.eclipse
    - org.eclipse
    - org.eclipse
    - org.eclipse
    - org.eclipse
    - org.eclipse
    - org.eclipse
    - org.eclipse
      - Commo
      - CommonDragAdapter.java 1.7 (ASCII -kkv)
      - CommonDragAdapterAssistant.java 1.5 (ASCII -kkv)
      - CommonDropAdapter.java 1.8 (ASCII -kkv)

Resource

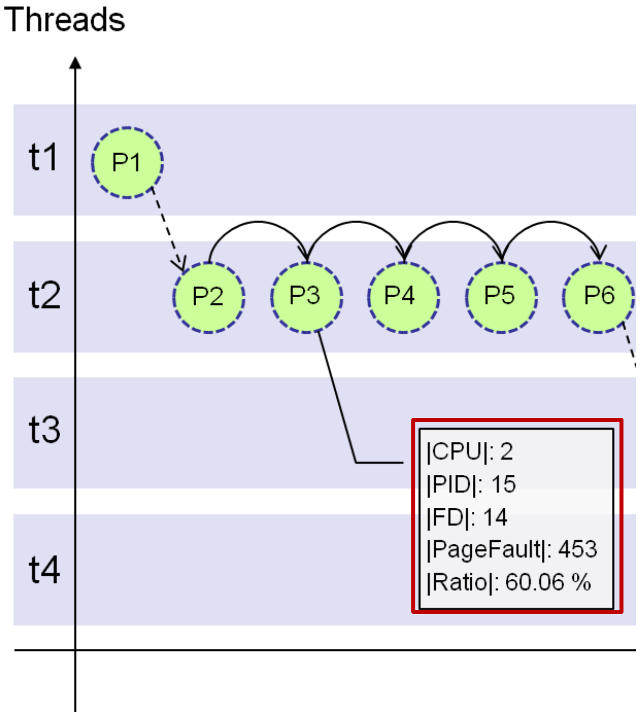
Path:	/seat/src/seat/model/algorithms/Phase
Type:	File (Java Source File)
Location:	C:\Projects\sources\seatngl\src\seat\m
Size:	27,654 bytes
Last modified:	8 November, 2011 11:47:20 AM

Attributes:

□ Read only
☑ Archive
□ Derived

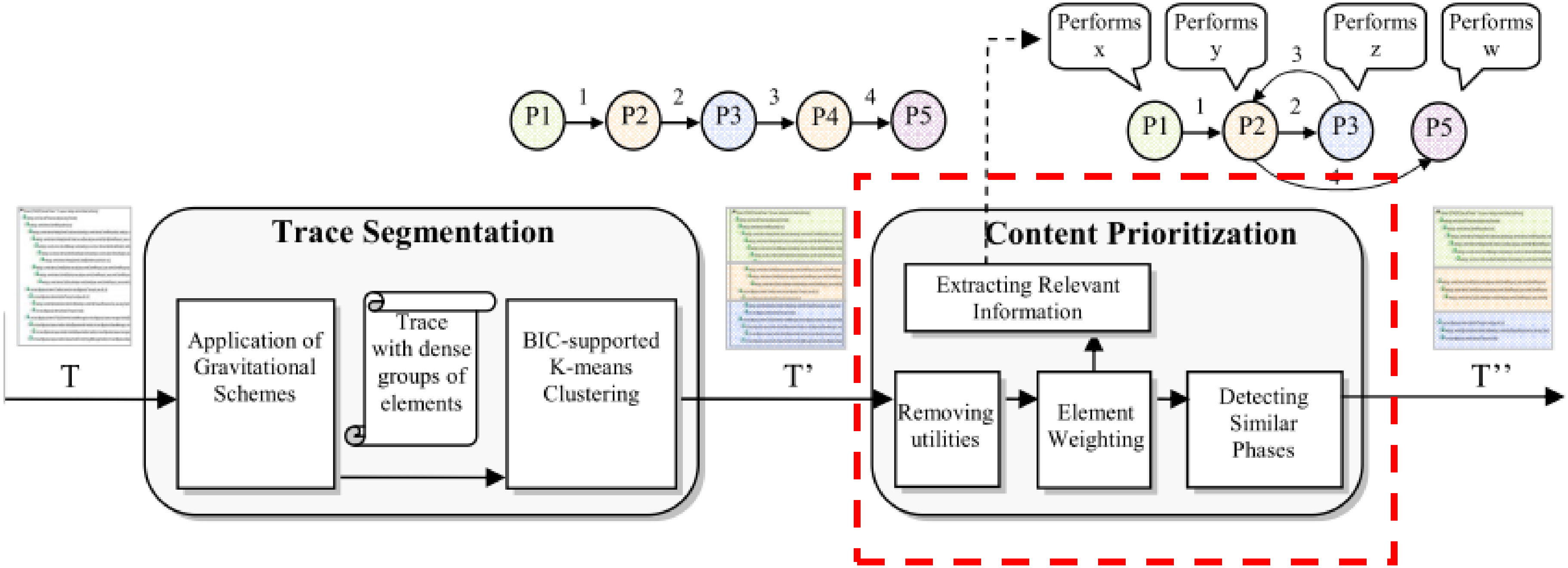
Text file encoding

● Default (inherited from container: Cp1252)
○ Other: Cp1252



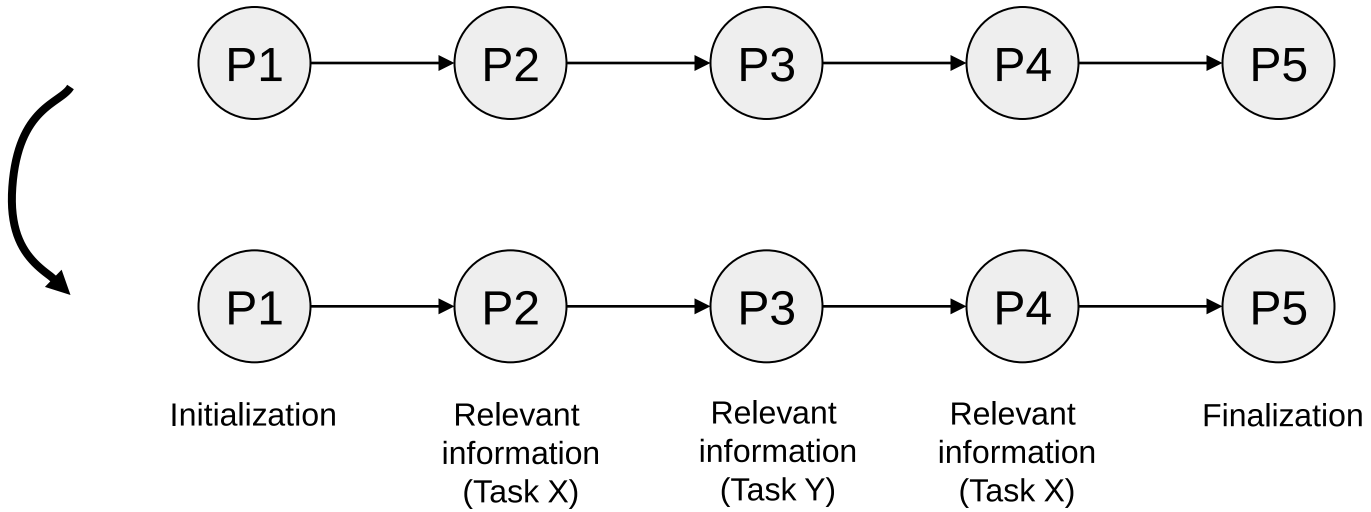
|CPU|: 2
|PID|: 15
|FD|: 14
|PageFault|: 453
|Ratio|: 60.06 %

Approach: Trace Abstraction Framework



Content Prioritization

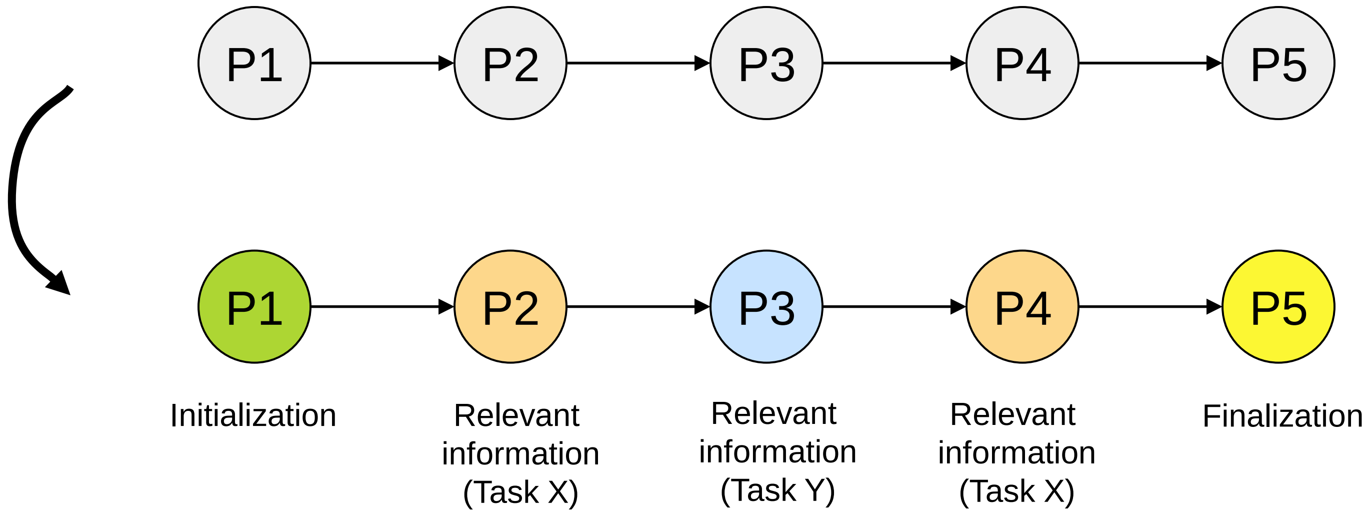
1. Extract representative elements of each phase


P1
P2
P3
P4
P5
P1
P2
P3
P4
P5
Initialization
Relevant information (Task X)
Relevant information (Task Y)
Relevant information (Task X)
Finalization

- Can give a hint about what is happening in a phase
- Uncover the most relevant elements that implement the traced scenario

Content Prioritization

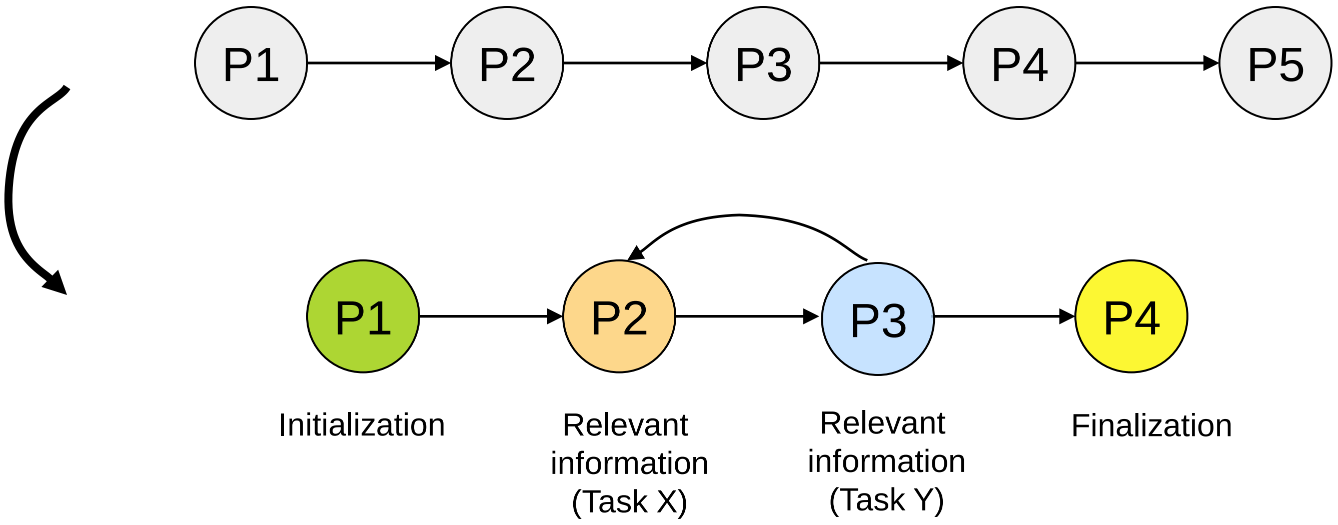
2- Finding similar phases


P1
P2
P3
P4
P5
P1
P2
P3
P4
P5
Initialization
Relevant information (Task X)
Relevant information (Task Y)
Relevant information (Task X)
Finalization

- Can give a hint about what is happening in a phase
- Uncover the most relevant elements that implement the traced scenario

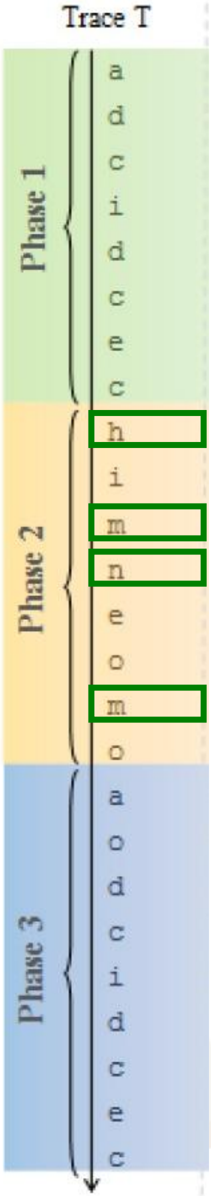
Content Prioritization

2- Finding similar phases



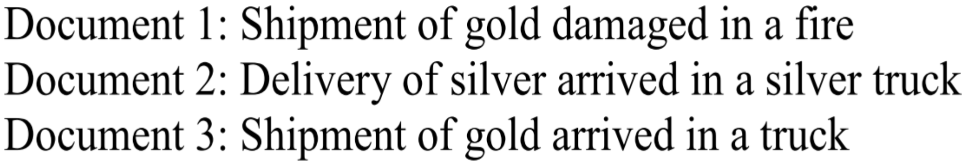
- Can give a hint about what is happening in a phase
- Uncover the most relevant elements that implement the traced scenario
- Optimized flow of phases

Extracting Relevant Components



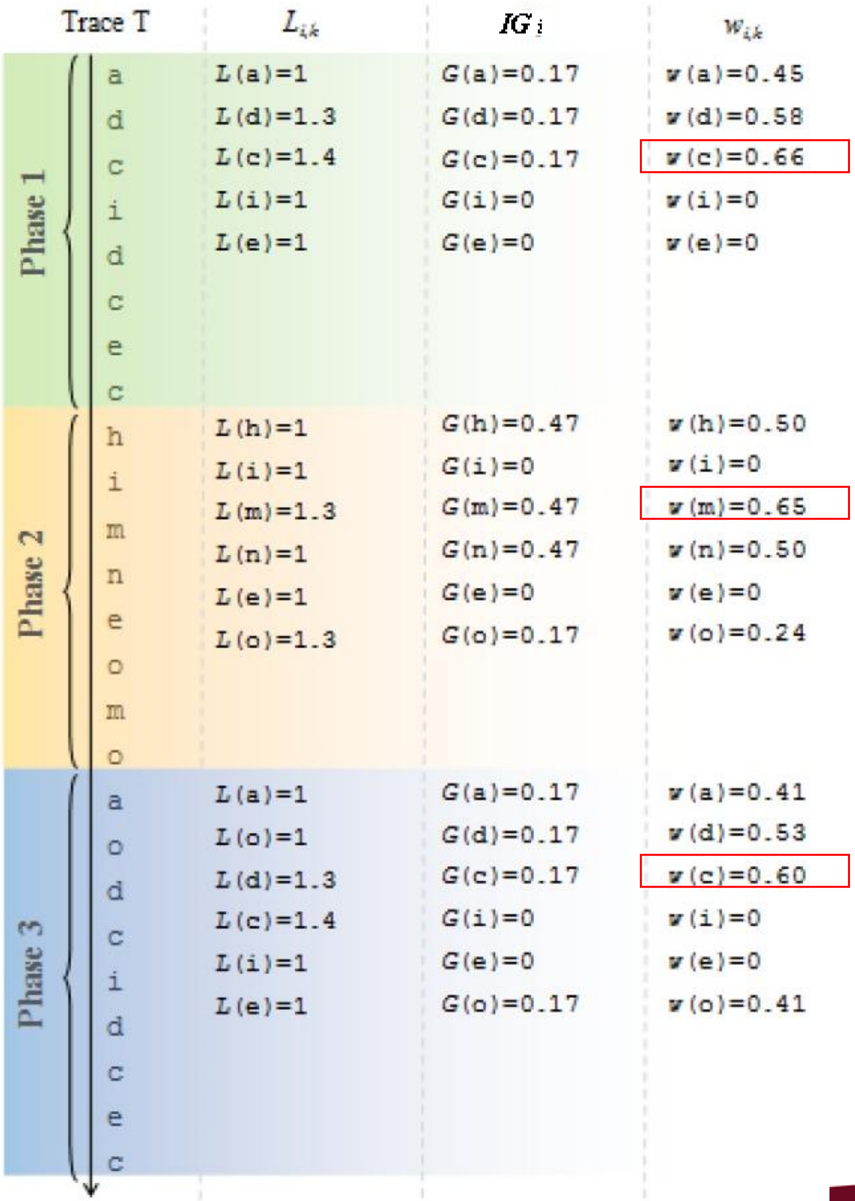
- Idea: Elements that are repeated in a phase but are not much shared between phases indicate their relevance to the phase

- This is similar to the concept of term frequency inverse document frequency in the text mining



Document 1: Shipment of gold damaged in a fire
Document 2: Delivery of silver arrived in a silver truck
Document 3: Shipment of gold arrived in a truck

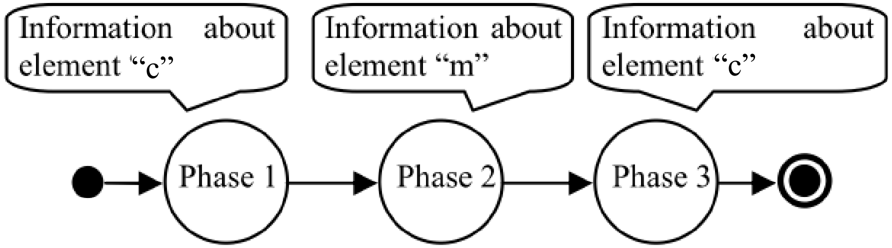
Extracting Representative Elements



	Trace T	$L_{i,k}$	IG_i	$w_{i,k}$
Phase 1	a	L(a)=1	G(a)=0.17	w(a)=0.45
	d	L(d)=1.3	G(d)=0.17	w(d)=0.58
	c	L(c)=1.4	G(c)=0.17	w(c)=0.66
	i	L(i)=1	G(i)=0	w(i)=0
	d	L(e)=1	G(e)=0	w(e)=0
	c			
	e			
	c			
Phase 2	h	L(h)=1	G(h)=0.47	w(h)=0.50
	i	L(i)=1	G(i)=0	w(i)=0
	m	L(m)=1.3	G(m)=0.47	w(m)=0.65
	n	L(n)=1	G(n)=0.47	w(n)=0.50
	e	L(e)=1	G(e)=0	w(e)=0
	o	L(o)=1.3	G(o)=0.17	w(o)=0.24
	m			
	o			
Phase 3	a	L(a)=1	G(a)=0.17	w(a)=0.41
	o	L(o)=1	G(d)=0.17	w(d)=0.53
	d	L(d)=1.3	G(c)=0.17	w(c)=0.60
	c	L(c)=1.4	G(i)=0	w(i)=0
	i	L(i)=1	G(e)=0	w(e)=0
	d	L(e)=1	G(o)=0.17	w(o)=0.41
	c			
	e			
	c			

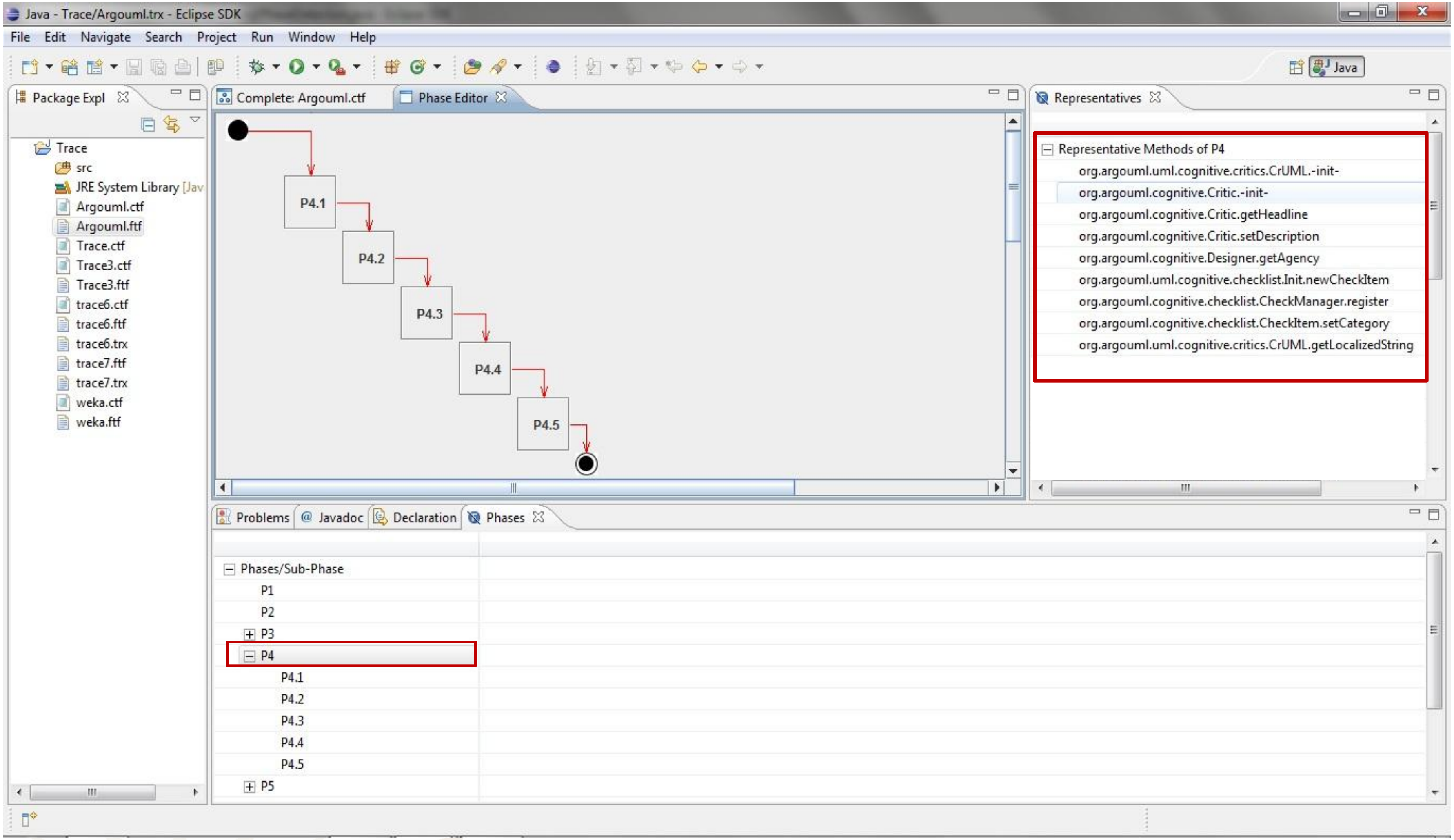
$$w_{i,k} = \underbrace{\frac{\overbrace{(\log(ef_{i,k})+1)}^{L_{i,k}} * \overbrace{\log(N/n_i)}^{IG_i}}{\sqrt{\sum_{j=1}^{e}\left[\left(\log\left(ef_{j,k}\right)+1\right) * \log\left(N/n_j\right)\right]^2}}}_{\frac{1}{N_k}}$$

$$w_{"d",Phase\ 1} = \frac{1.3 * 0.17}{\sqrt{(0.17)^2 + (0.22)^2 + (0.26)^2}} = 0.45$$



Information about element "c"
Information about element "m"
Information about element "c"
Phase 1
Phase 2
Phase 3

Relevant Events Snapshots



Java - Trace/Argouml.trx - Eclipse SDK

- Trace
  - src
  - JRE System Library [Jav
  - Argouml.ctf
  - Argouml.ftf
  - Trace.ctf
  - Trace3.ctf
  - Trace3.ftf
  - trace6.ctf
  - trace6.ftf
  - trace6.trx
  - trace7.ftf
  - trace7.trx
  - weka.ctf
  - weka.ftf

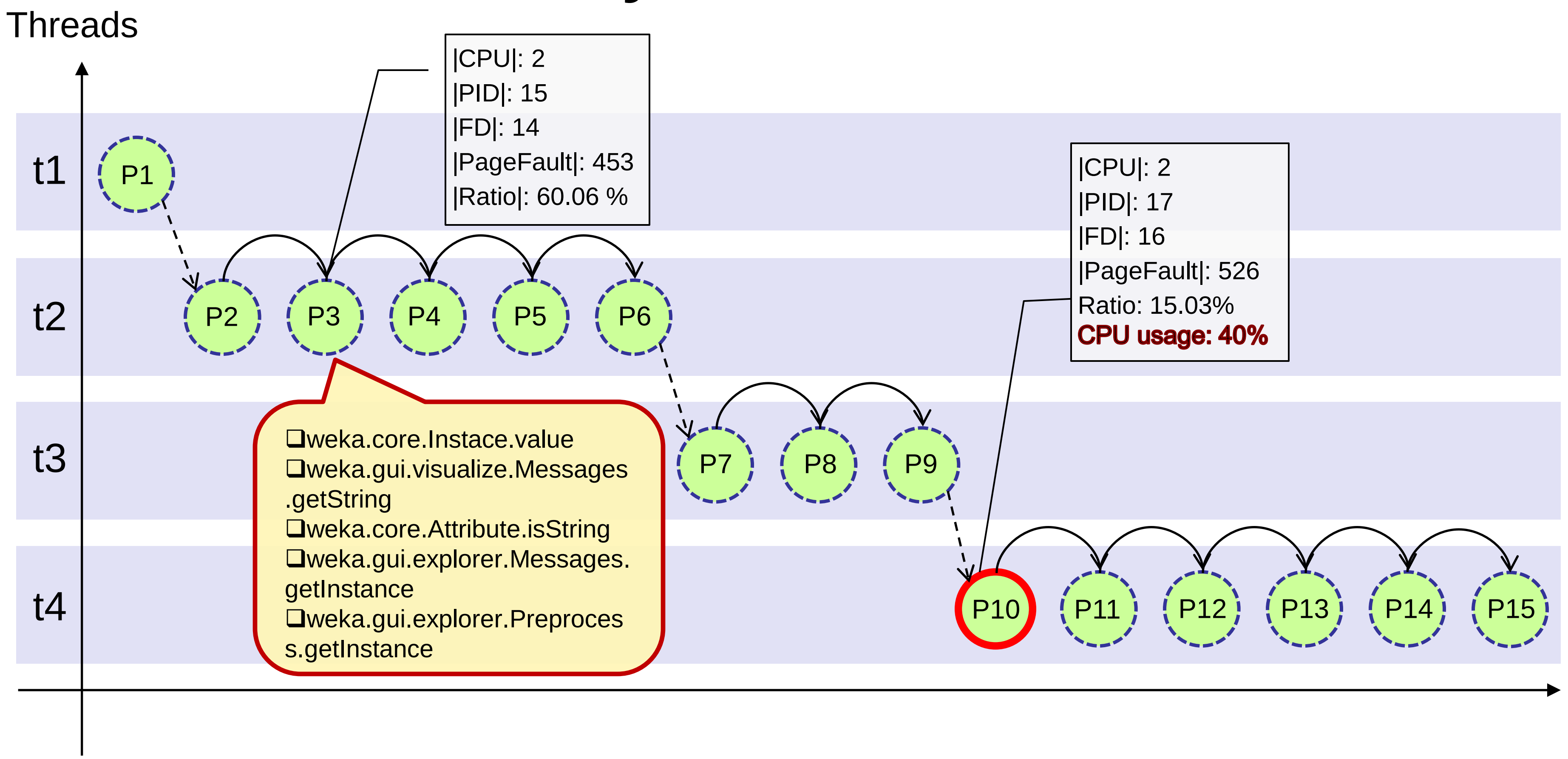
Representative Methods of P4

- org.argouml.uml.cognitive.critics.CrUML.-init-
- org.argouml.cognitive.Critic.-init-
- org.argouml.cognitive.Critic.getHeadline
- org.argouml.cognitive.Critic.setDescription
- org.argouml.cognitive.Designer.getAgency
- org.argouml.uml.cognitive.checklist.Init.newCheckItem
- org.argouml.cognitive.checklist.CheckManager.register
- org.argouml.cognitive.checklist.CheckItem.setCategory
- org.argouml.uml.cognitive.critics.CrUML.getLocalizedString

Phases/Sub-Phase

- P1
- P2
- P3
- P4
  - P4.1
  - P4.2
  - P4.3
  - P4.4
  - P4.5
- P5

Case Study: Relevant Events



|CPU|: 2
|PID|: 15
|FD|: 14
|PageFault|: 453
|Ratio|: 60.06 %

|CPU|: 2
|PID|: 17
|FD|: 16
|PageFault|: 526
Ratio: 15.03%
CPU usage: 40%

- ❑weka.core.Instace.value
- ❑weka.gui.visualize.Messages.getString
- ❑weka.core.Attribute.isString
- ❑weka.gui.explorer.Messages.getInstance
- ❑weka.gui.explorer.Preprocess.getInstance

Conclusions

- We showed trace abstraction techniques based on execution phases
- We added state information to extracted phases
- We presented techniques for identifying the most relevant components of each phase

Thank you!