The Role of Software Tracing in Software Maintenance

Abdelwahab Hamou-Lhadj, PhD.
Software Behaviour Analysis (SBA) Research Lab
Department of Electrical and Computer Engineering
Concordia University, Montreal, QC, Canada
www.ece.concordia.ca/~abdelw

CIIA'13, Saida, Algeria
May 5, 2013

Software Maintenance

- is defined as the modification of a software system after delivery
- accounts for 75% of the time of the software life cycle
- tends to be a human resource intensive process
- incurs very high costs: SW maintenance is estimated to a multi-billion dollar market

Issues with exiting software

- More than 100 billion lines of code in production in the world
- A large portion of it is unstructured, patched, and badly documented
- Initial design and architecture can no longer be trusted
- High turn-over causes initial developers to move from one company to another
- SW industry tends to be a poorly regulated industry

As a result

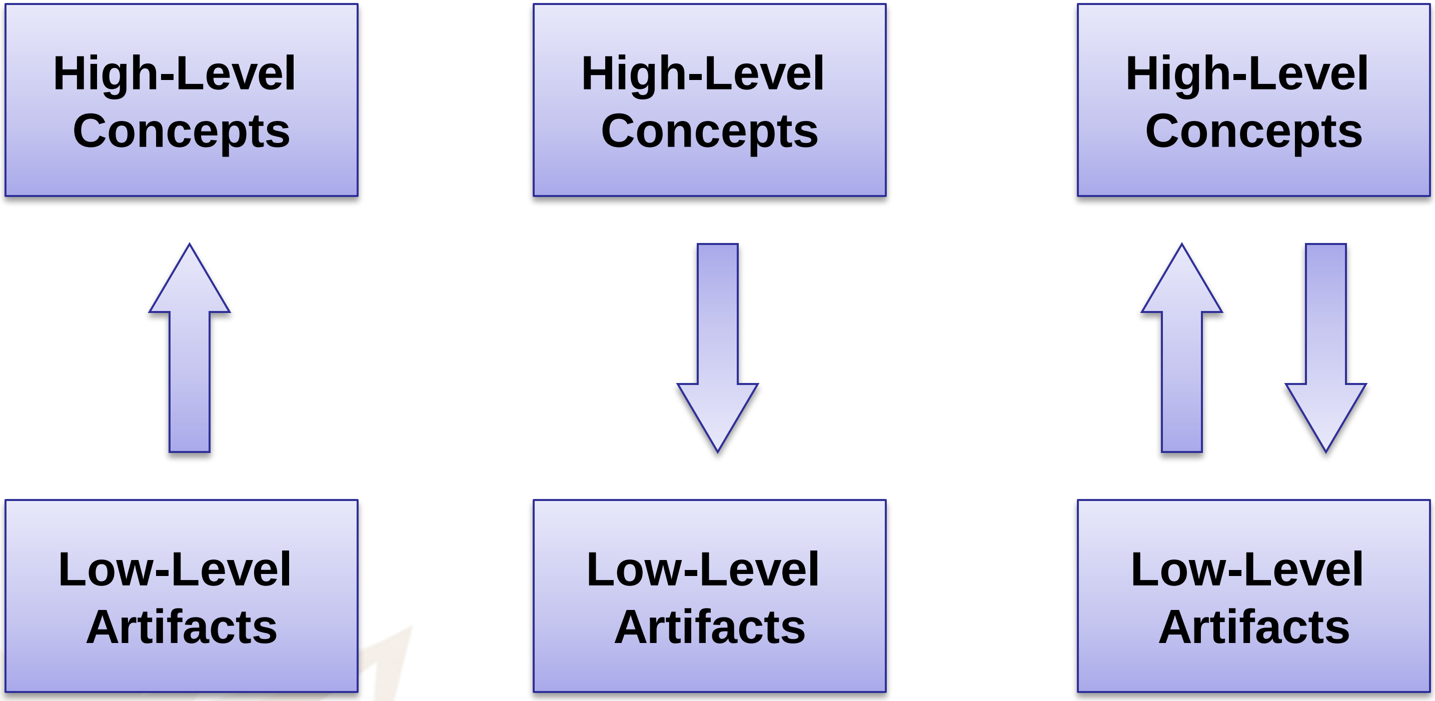
Software engineers must spent a considerable amount of time to understand the system before making any changes to it

As a result

Software engineers must spent a considerable amount of time to understand the system before making any changes to it

How do programmers understand programs?

Program Comprehension Models



High-Level Concepts

Low-Level Artifacts

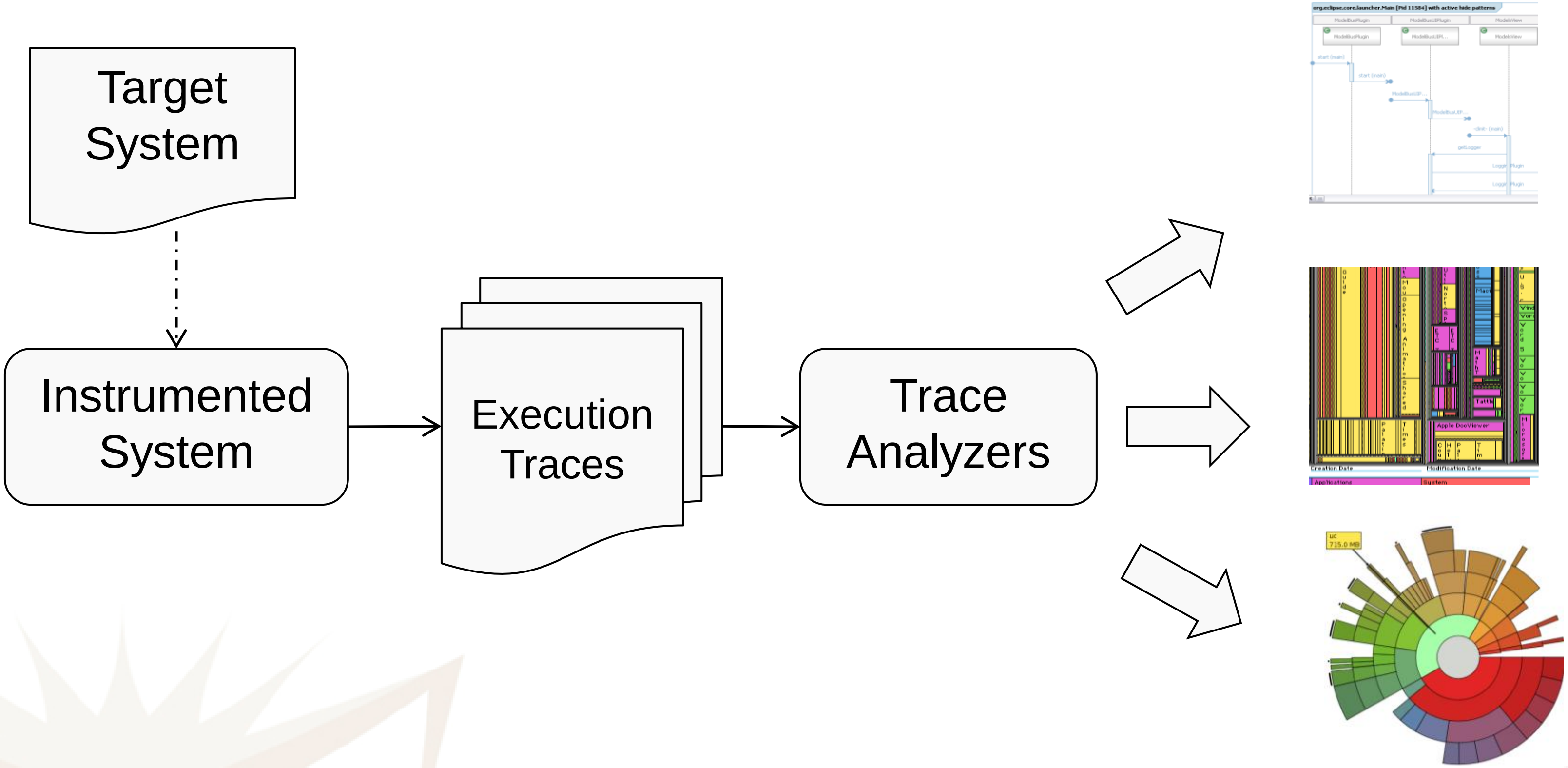
High-Level Concepts

Low-Level Artifacts

High-Level Concepts

Low-Level Artifacts

Understanding System Behavior Using Trace Analysis



Examples of execution traces

- Traces of routine (method) calls
- Traces of inter-process communication
- Traces of statement execution
- Traces of communication among subsystems
- Etc.

Trace Analysis (cont’d)

- Advantages:
  - High focus and resolution
  - Mapping of program inputs to outputs
  - Source code is not needed
- Challenges:
  - Tracing adds overhead to the system
  - Traces are overwhelmingly large
  - Different types of traces may require different processing techniques

Applications of Trace Analysis: Industrial Projects

Project 1: Tracing and Monitoring Tools for Distributed Multi-Core Systems

Project 2: Diagnostics for Real Time Distributed Multi-Core Architecture in Avionics

Project 3: Finding Faulty Functions from Traces of Field Failures

Tracing and Monitoring Tools for Distributed Multi-Core Systems

Develop techniques and algorithms to provide a software architecture for low overhead trace generation and analysis tools for complex distributed multi-core systems

Project Partners

ERICSSON

DEFENCE R&D DÉFENSE

NSERC
CRSNG

ÉCOLE
POLYTECHNIQUE
MONTRÉAL

UNIVERSITÉ
Concordia
UNIVERSITY

uOttawa

L'Université canadienne
Canada's university

UNIVERSITÉ
LAVAL

Trace Generation

- Research thread led by Dr. Michel Dagenais from Polytech de Montreal
- Objectives:
  - Build a tracer with low overhead and no disturbance on the system
  - Offer support for synchronisation in a multi-core environment
  - Offer support for system and user space tracing

LTTng: Linux Trace Toolkit New Generation

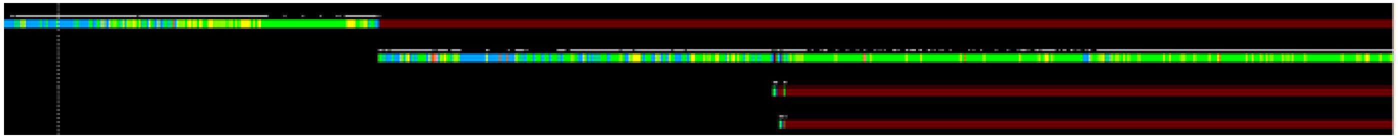
- Instruments the Linux kernel
- Adds 2% overhead to the kernel in the worst case scenario
- Is free and open source
- Is being integrated with the Linux kernel

Trace Analysis

- Objectives:
  - Simplify the understanding and analysis of very large traces
  - Extract high-level views from raw events
  - Identify the main components that implement the traced scenario
  - Correlate user space and system space traces

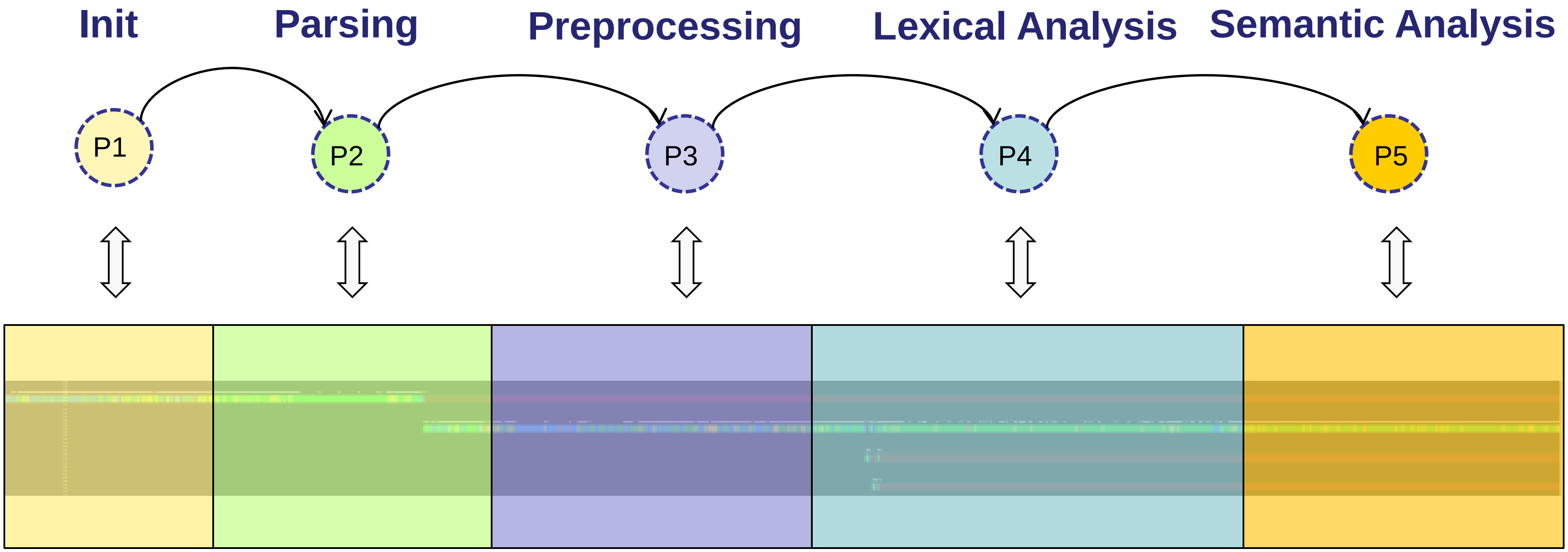
Motivating example

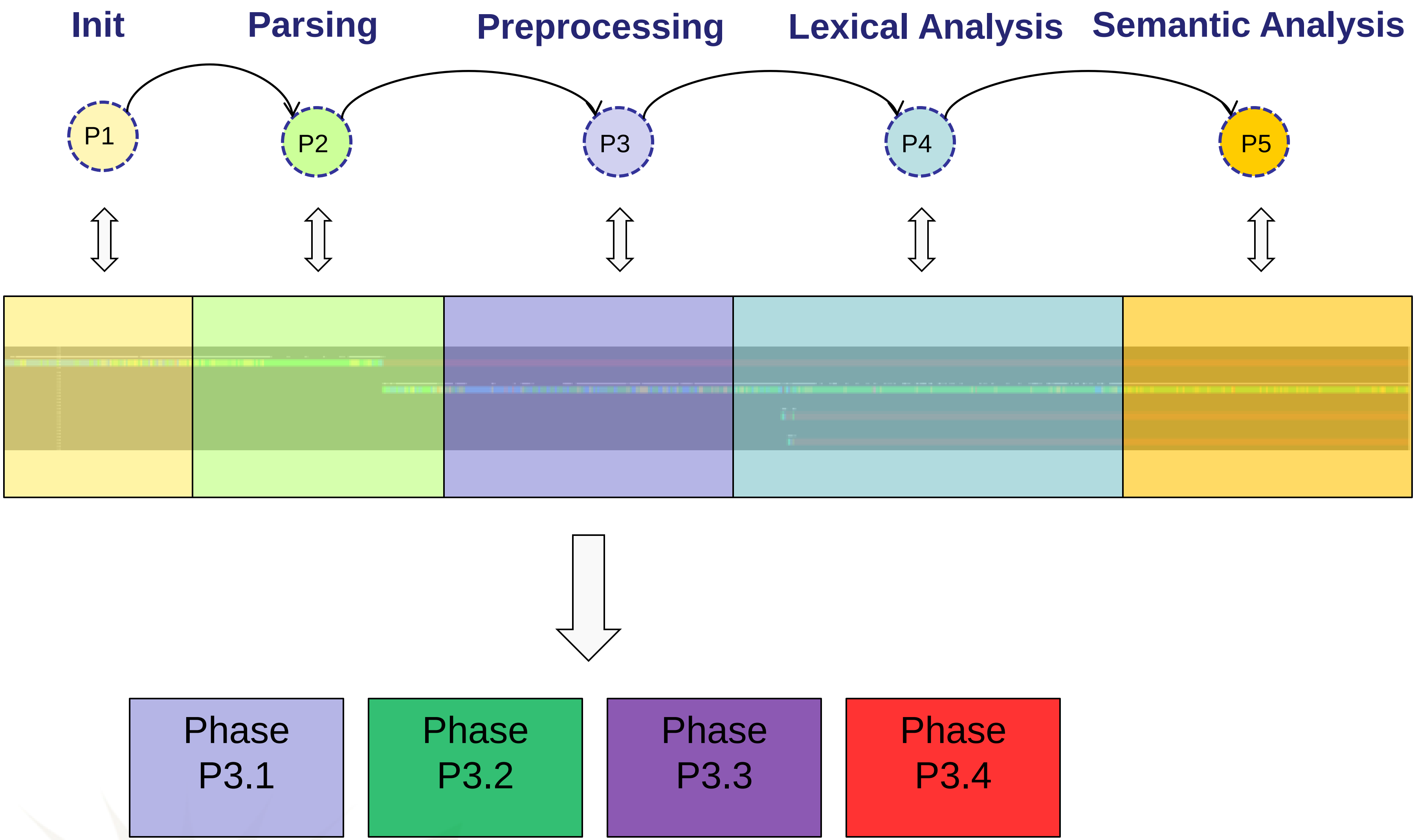
- A trace generated from a compiler:
  - parsing, preprocessing, lexical analysis, semantic analysis
- In most trace visualization tools, it will look like:



- But how can we tell what happens where?

An execution phase based view





Init

Parsing

Preprocessing

Lexical Analysis

Semantic Analysis

P1

P2

P3

P4

P5

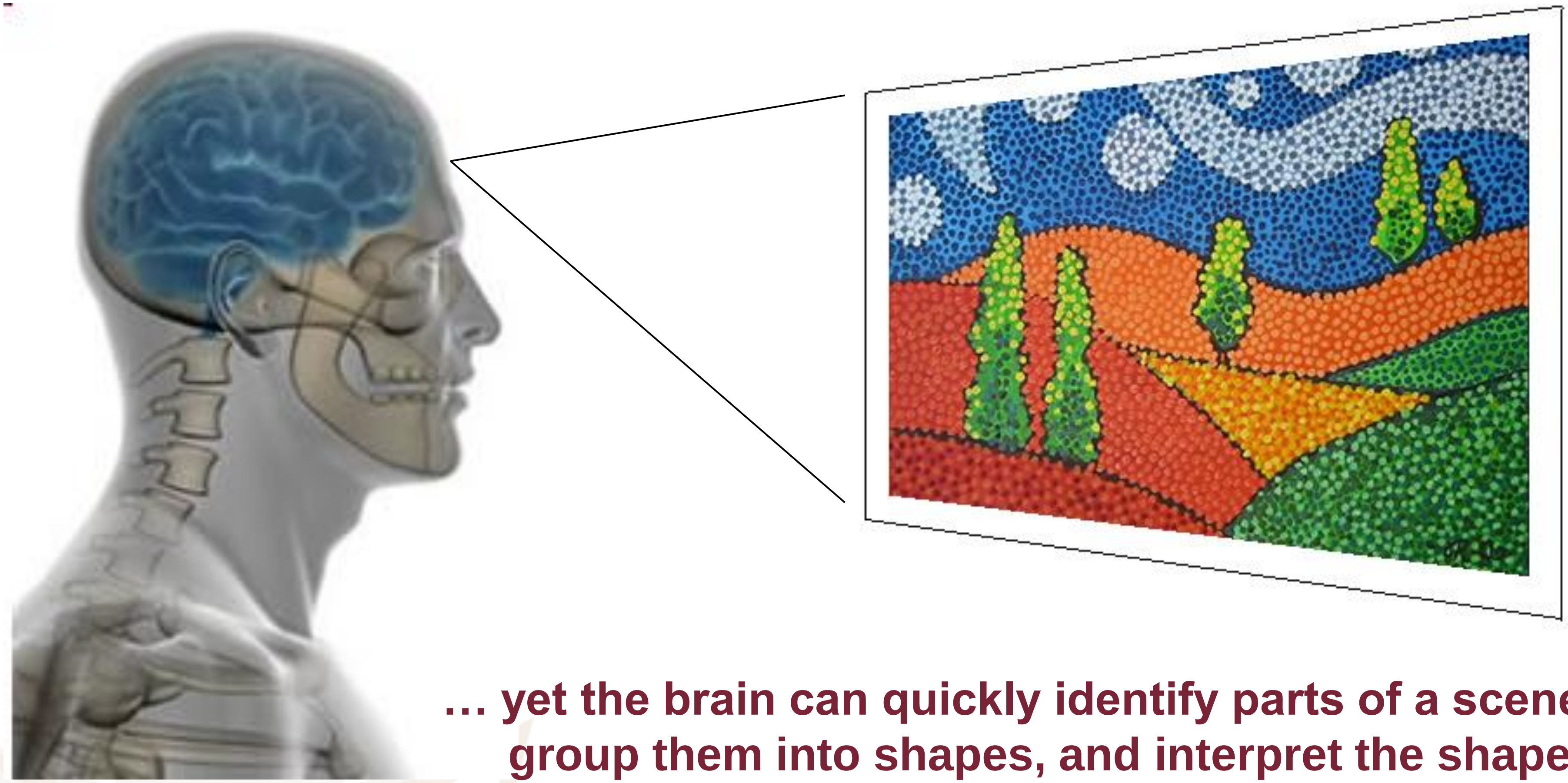
Phase P3.1

Phase P3.2

Phase P3.3

Phase P3.4

Same problem, different domain: The human perception system



… yet the brain can quickly identify parts of a scene, group them into shapes, and interpret the shapes

Segmentation:

The perceptual system segments local elements against their context and integrates them as objects and regions

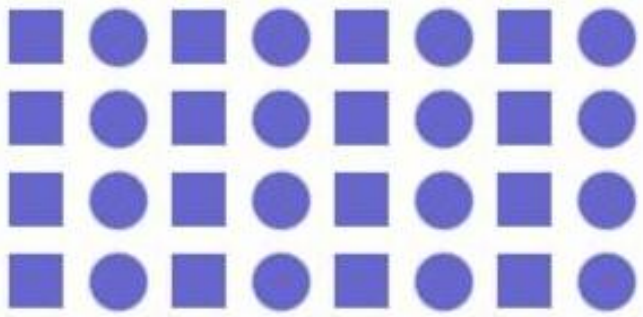
Global Perception:

The segmented scene is then quickly scanned with eye movements so as the brain obtains an overall impression of it

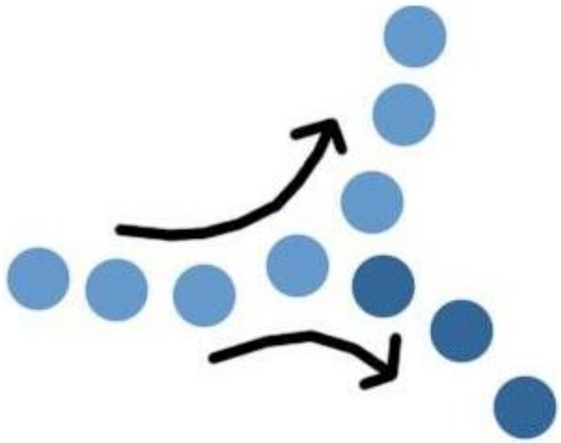
Preattentive Process:

The scene is analyzed in more detail by visiting the regions in a certain order. The pop-out effect is an important factor in this process

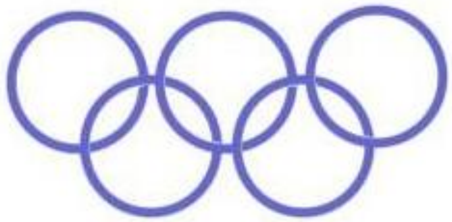
Gestalt Laws



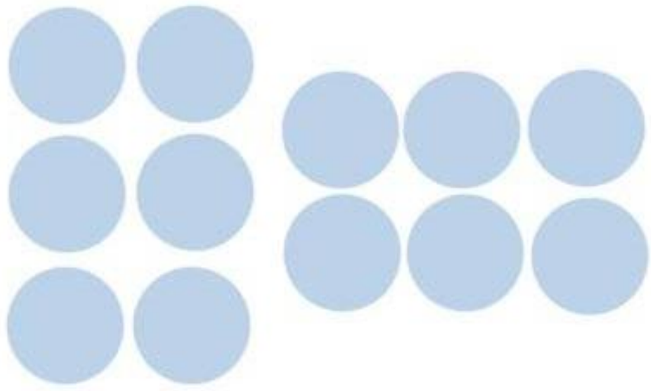
Law of Similarity



Law of Continuity

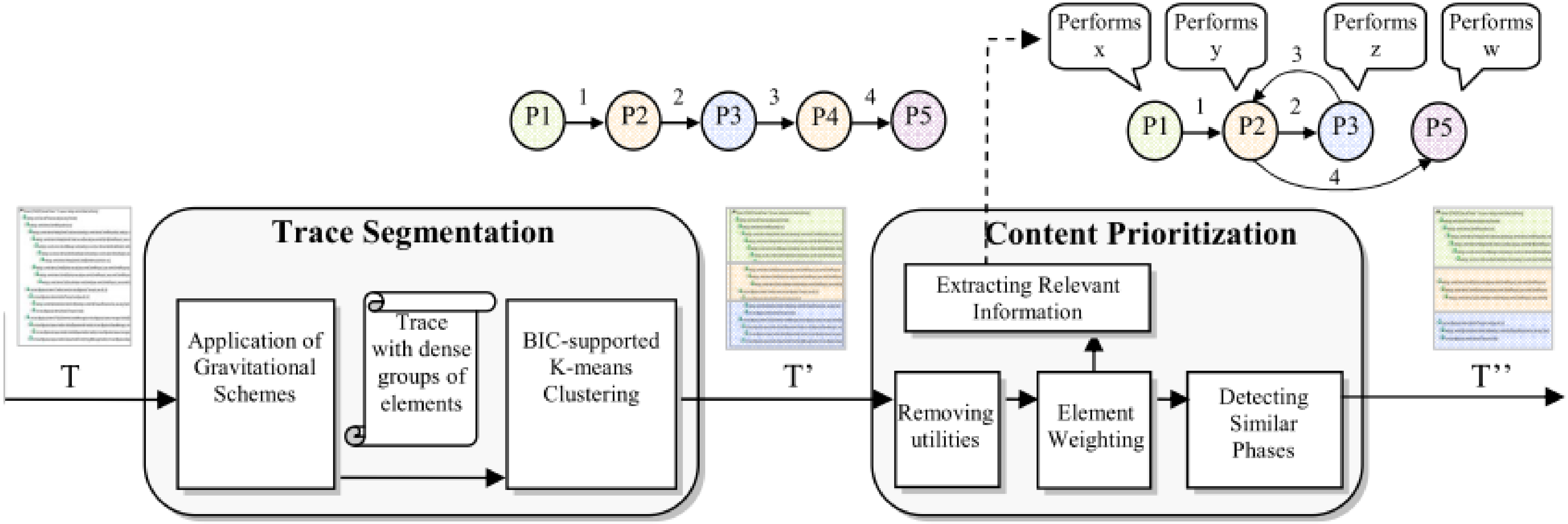


Law of Pragnanz



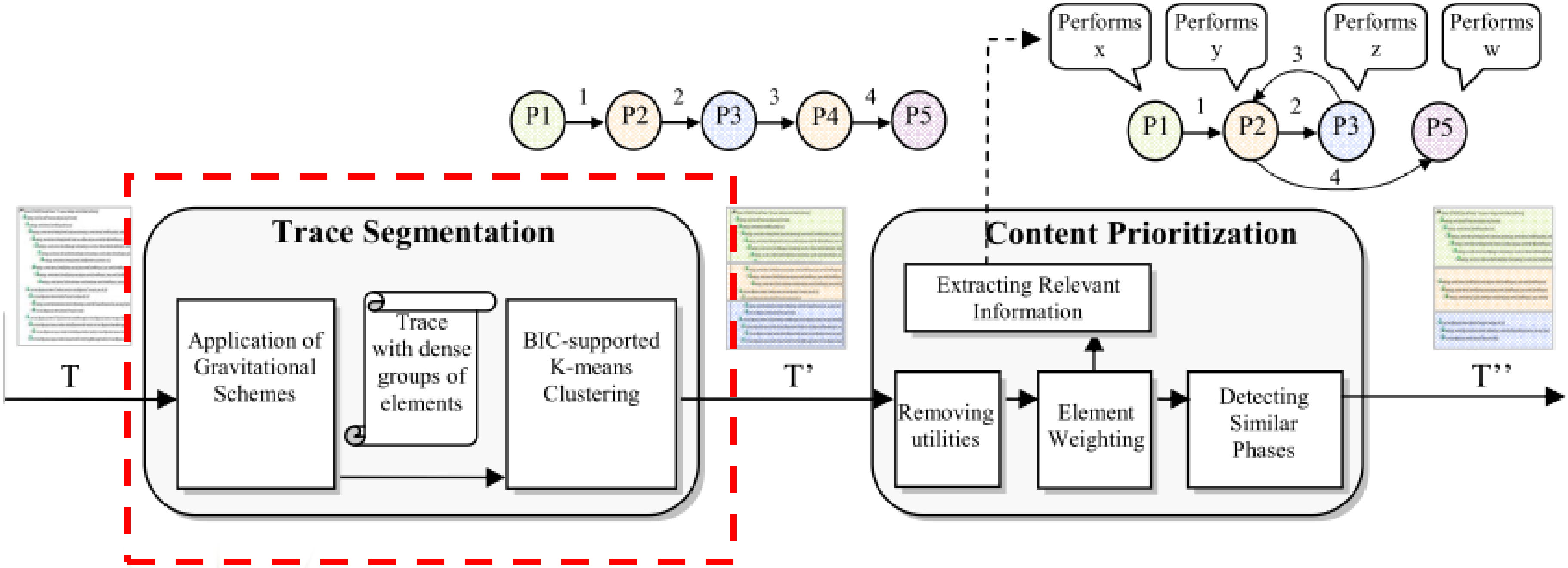
Law of Proximity

Trace abstraction framework

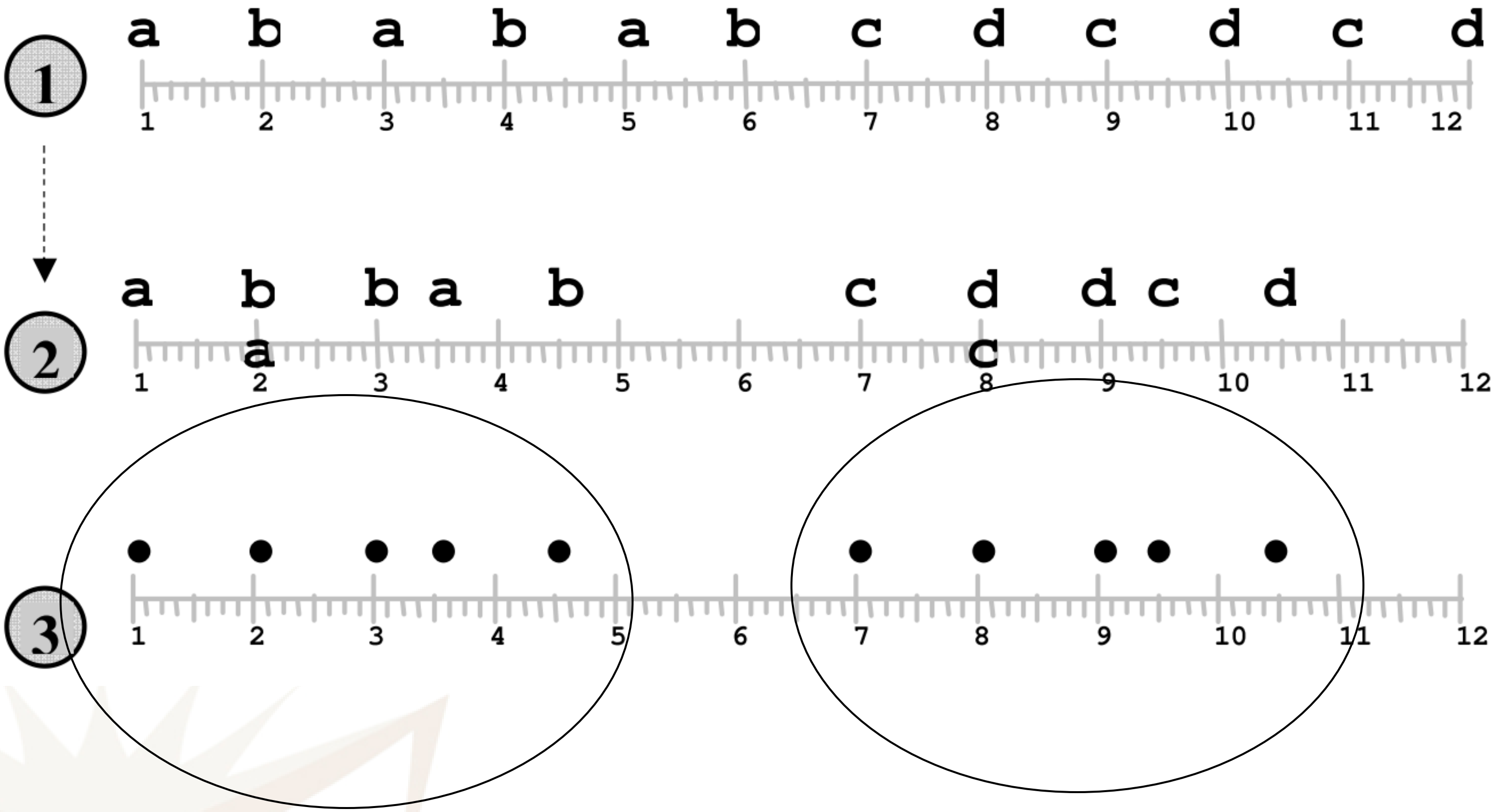


H. Pirzadeh, A. Hamou-Lhadj, ICSE 2010, ICPC 2011, ICECCS 2011

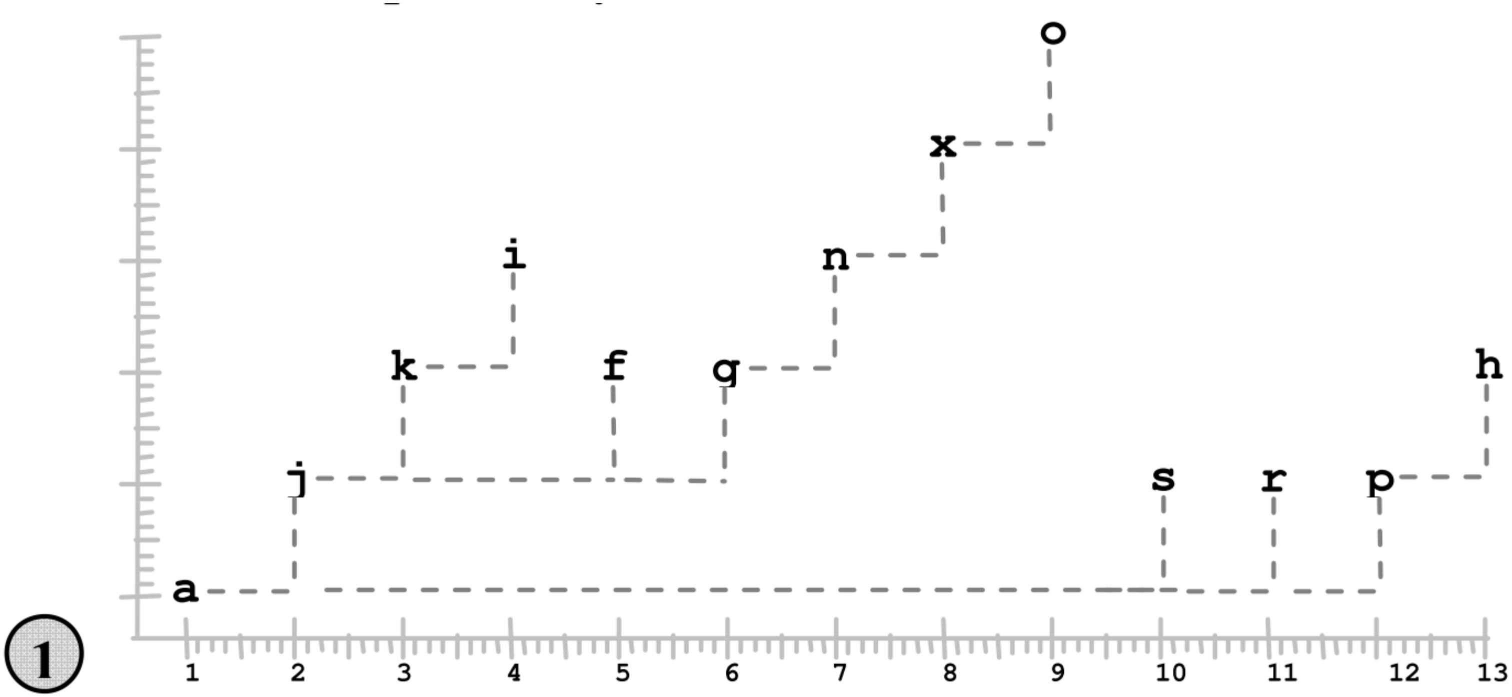
Trace abstraction framework



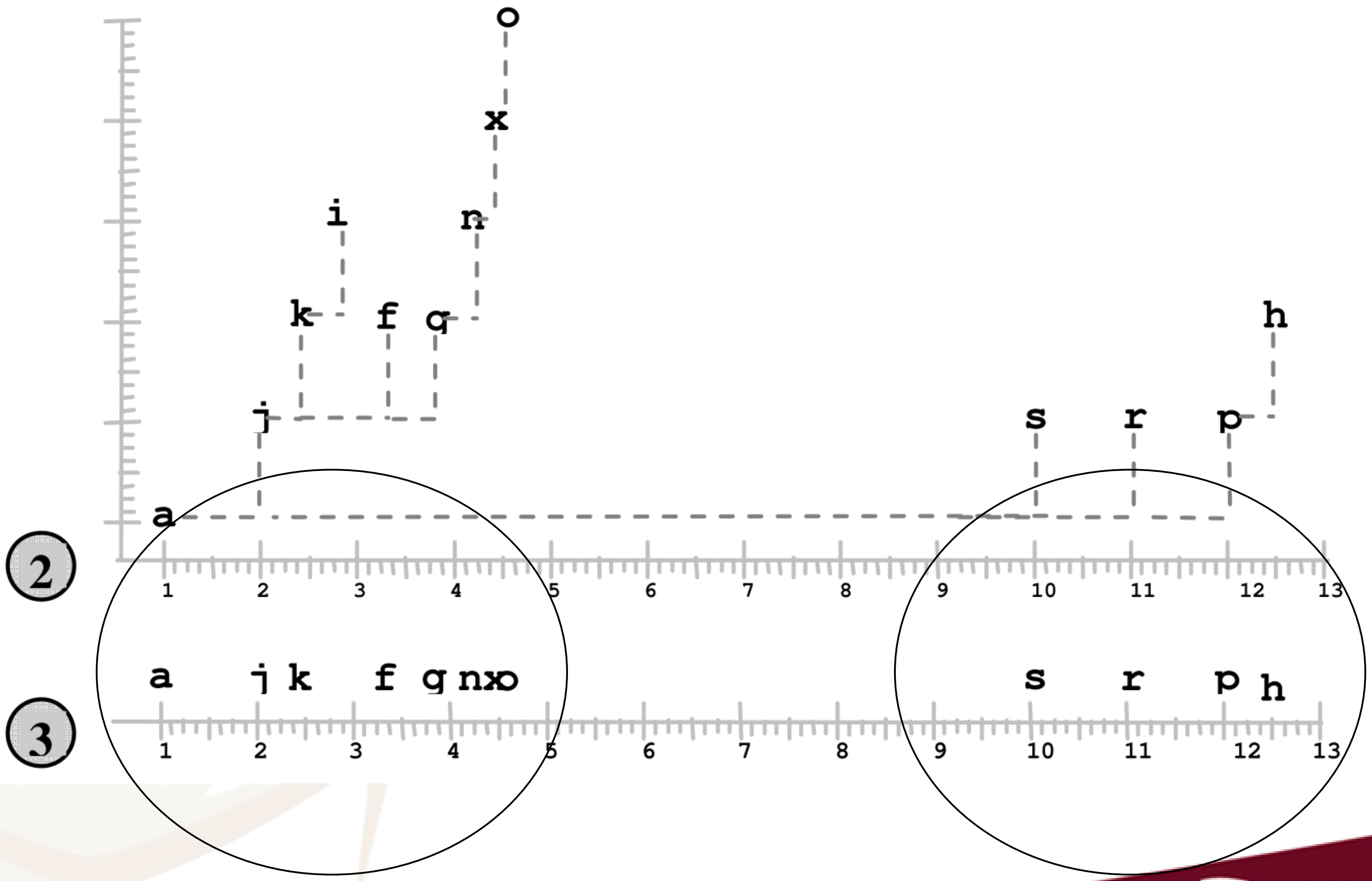
Repositioning trace events using similarity principle



Repositioning trace events using good continuation principle



Good continuation (cont.)

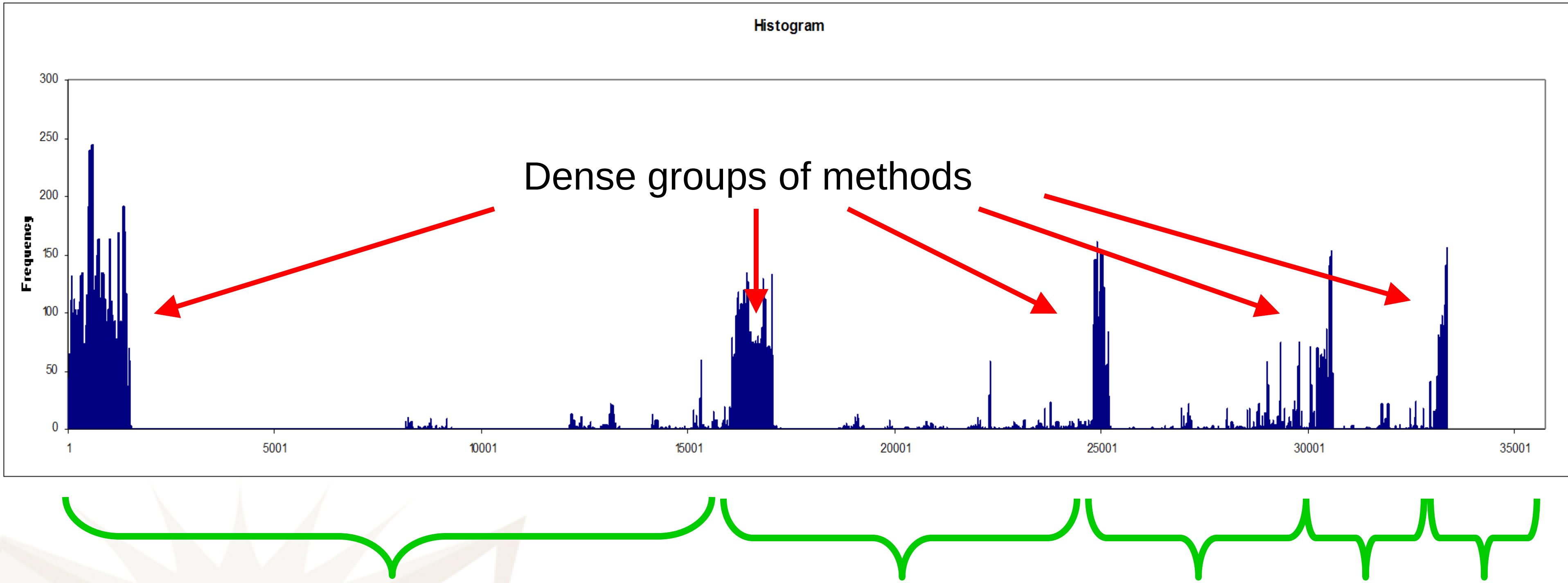


Evaluation

- Target System: ArgoUML
- Scenario: Starting up ArgoUML, drawing a class diagram, quitting ArgoUML
- Trace size: Hundred of thousands of function calls
- Number of distinct routines 2331 = ~33%

Application of trace segmentation

Starting up ArgoUML → Drawing a class diagram → Quitting ArgoUML



Dense groups of methods

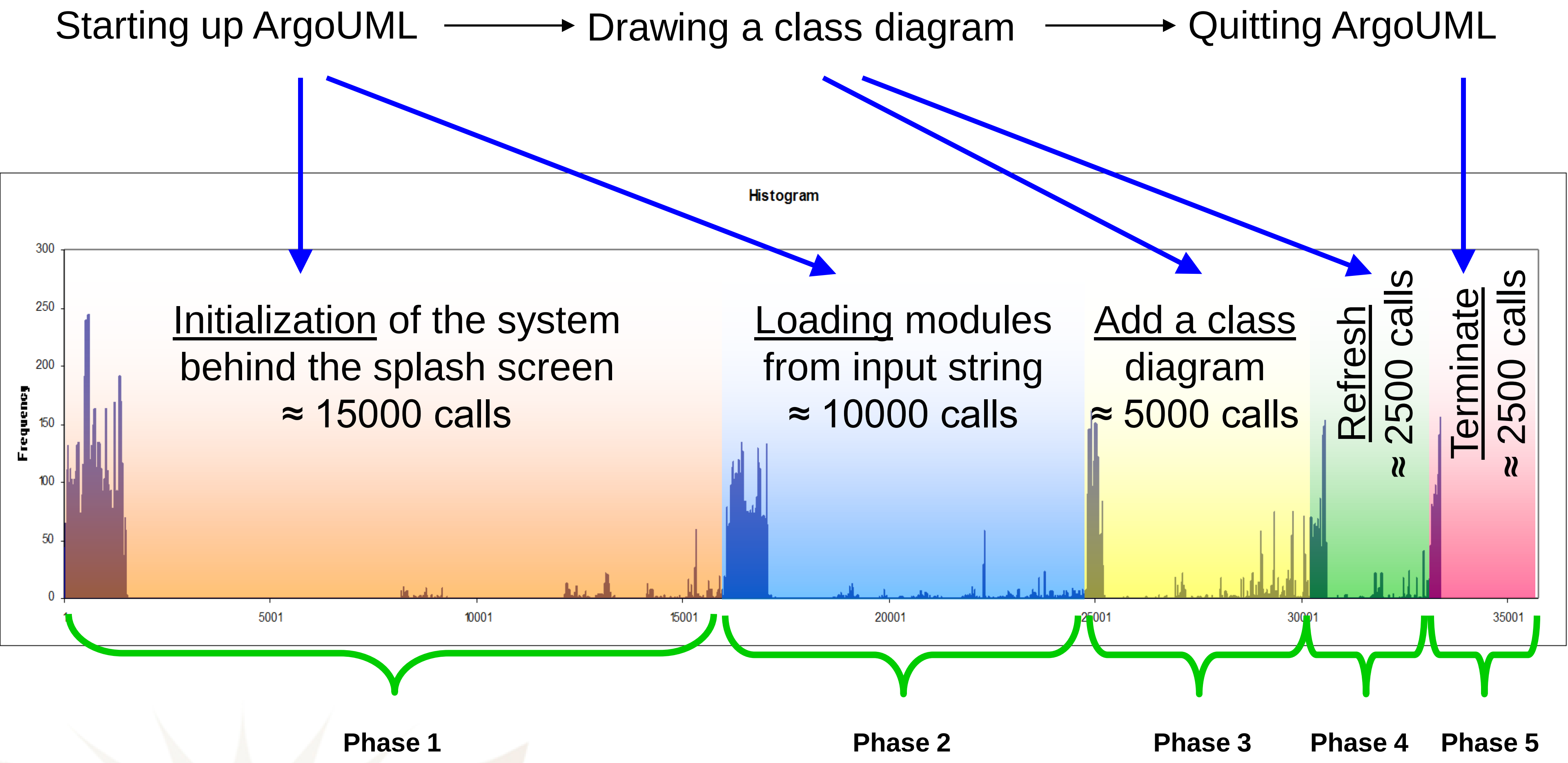
Phase 1

Phase 2

Phase 3

Phase 4

Phase 5



Starting up ArgoUML → Drawing a class diagram → Quitting ArgoUML

Initialization of the system behind the splash screen ≈ 15000 calls

Loading modules from input string ≈ 10000 calls

Add a class diagram ≈ 5000 calls

Refresh ≈ 2500 calls

Terminate ≈ 2500 calls

Phase 1

Phase 2

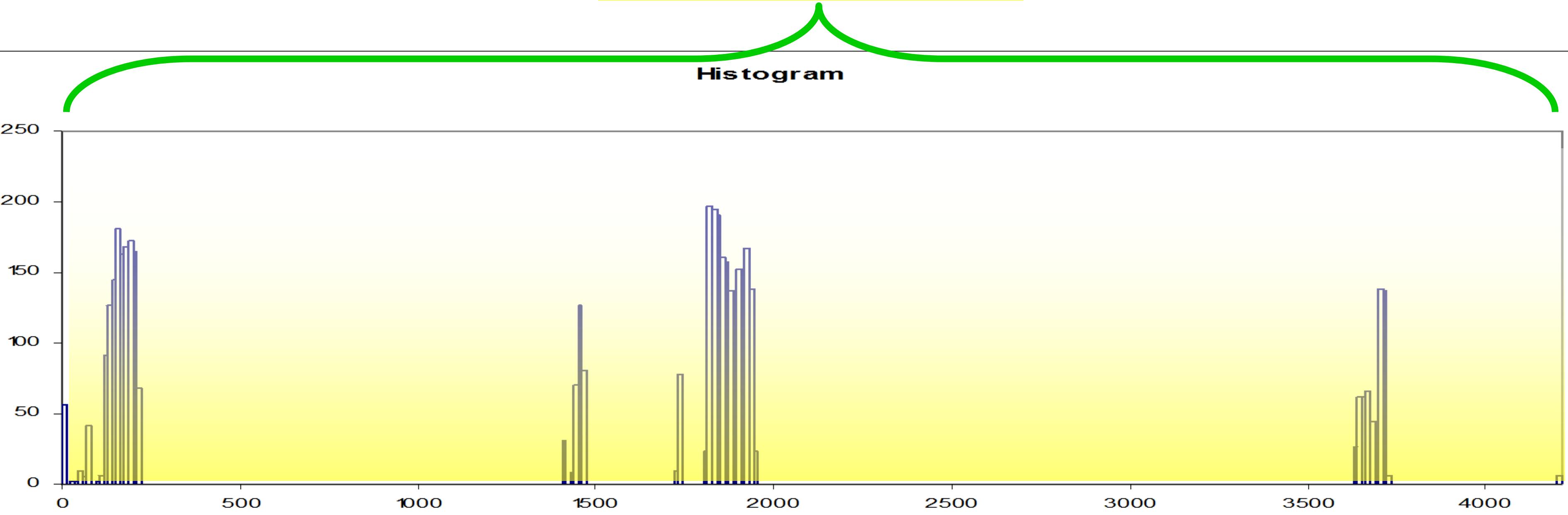
Phase 3

Phase 4

Phase 5

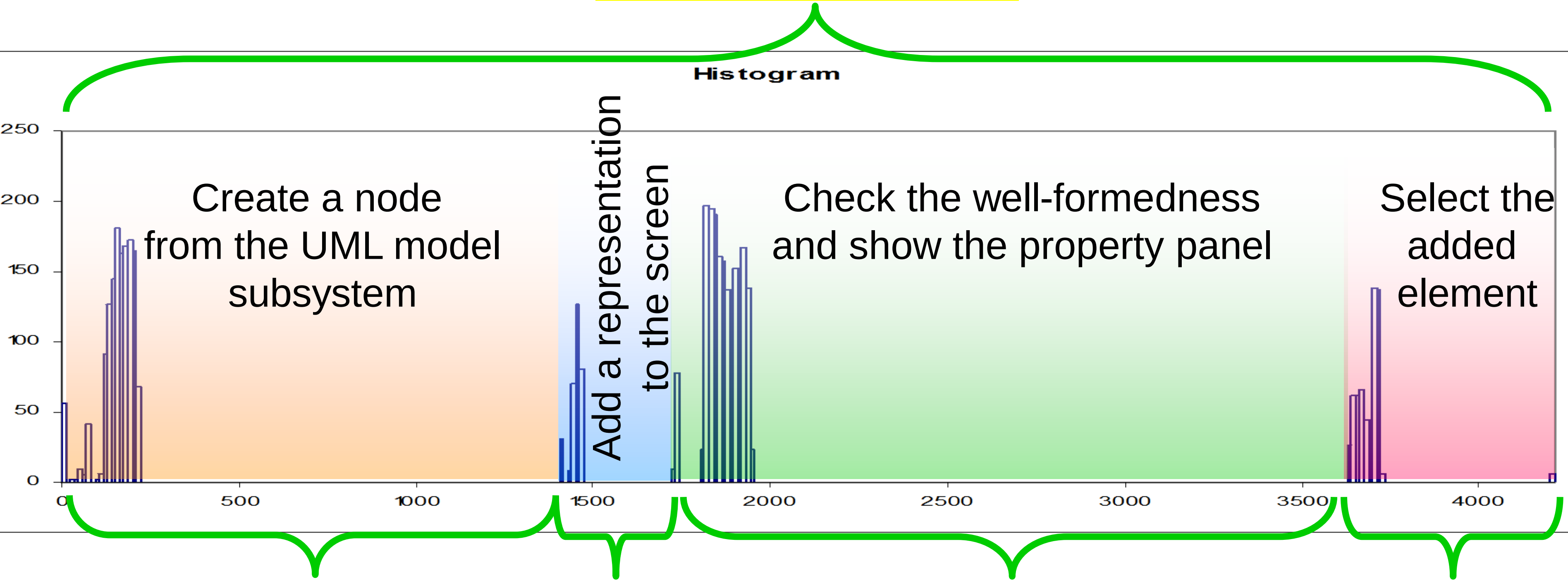
Starting up ArgoUML → Drawing a class diagram → Quitting ArgoUML

Phase 3: Add a class diagram



Starting up ArgoUML → Drawing a class diagram → Quitting ArgoUML

Phase 3: Add a class diagram



Create a node from the UML model subsystem

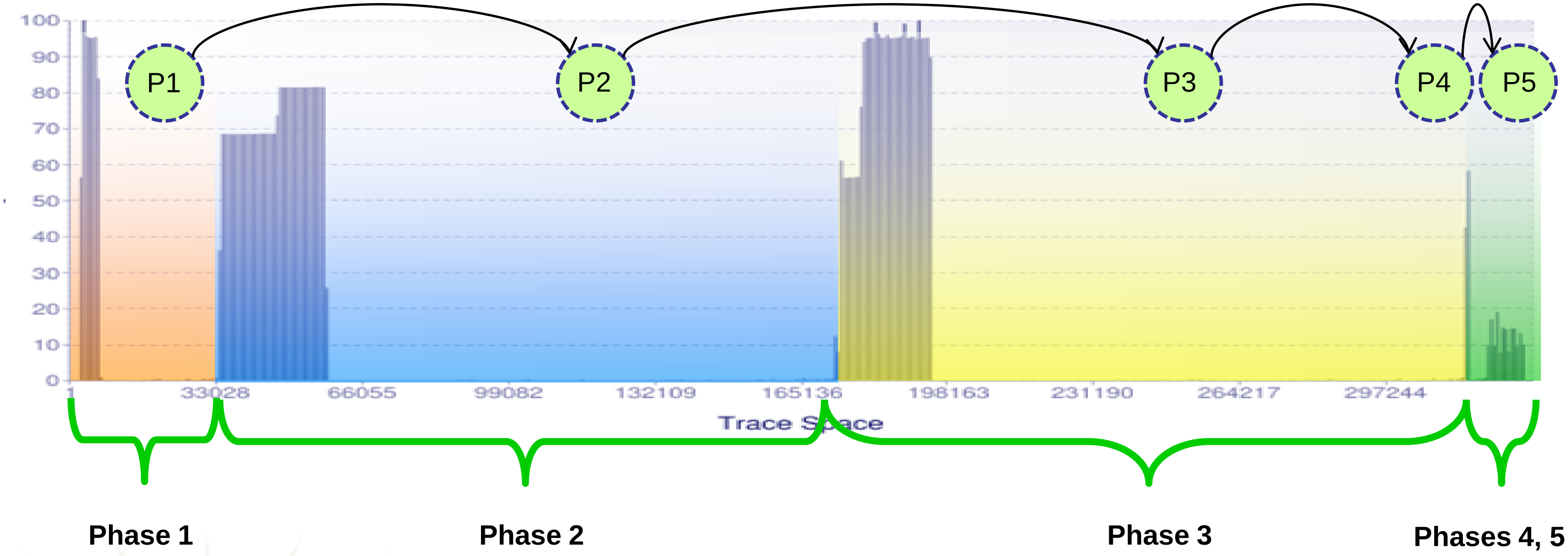
Add a representation to the screen

Check the well-formedness and show the property panel

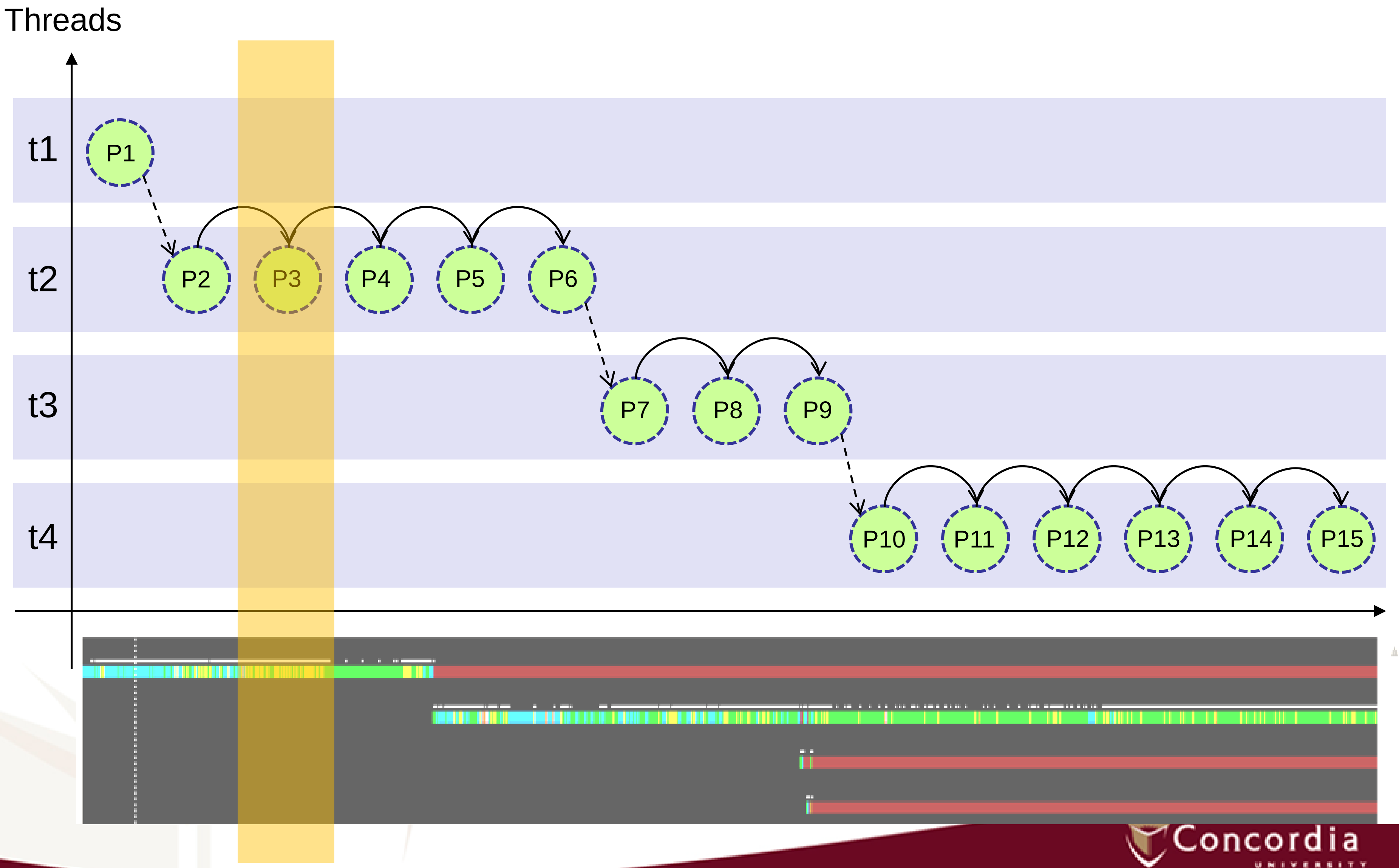
Select the added element

Sub-phase 1 **Sub-phase 2** **Sub-phase 3** **Sub-phase 4**

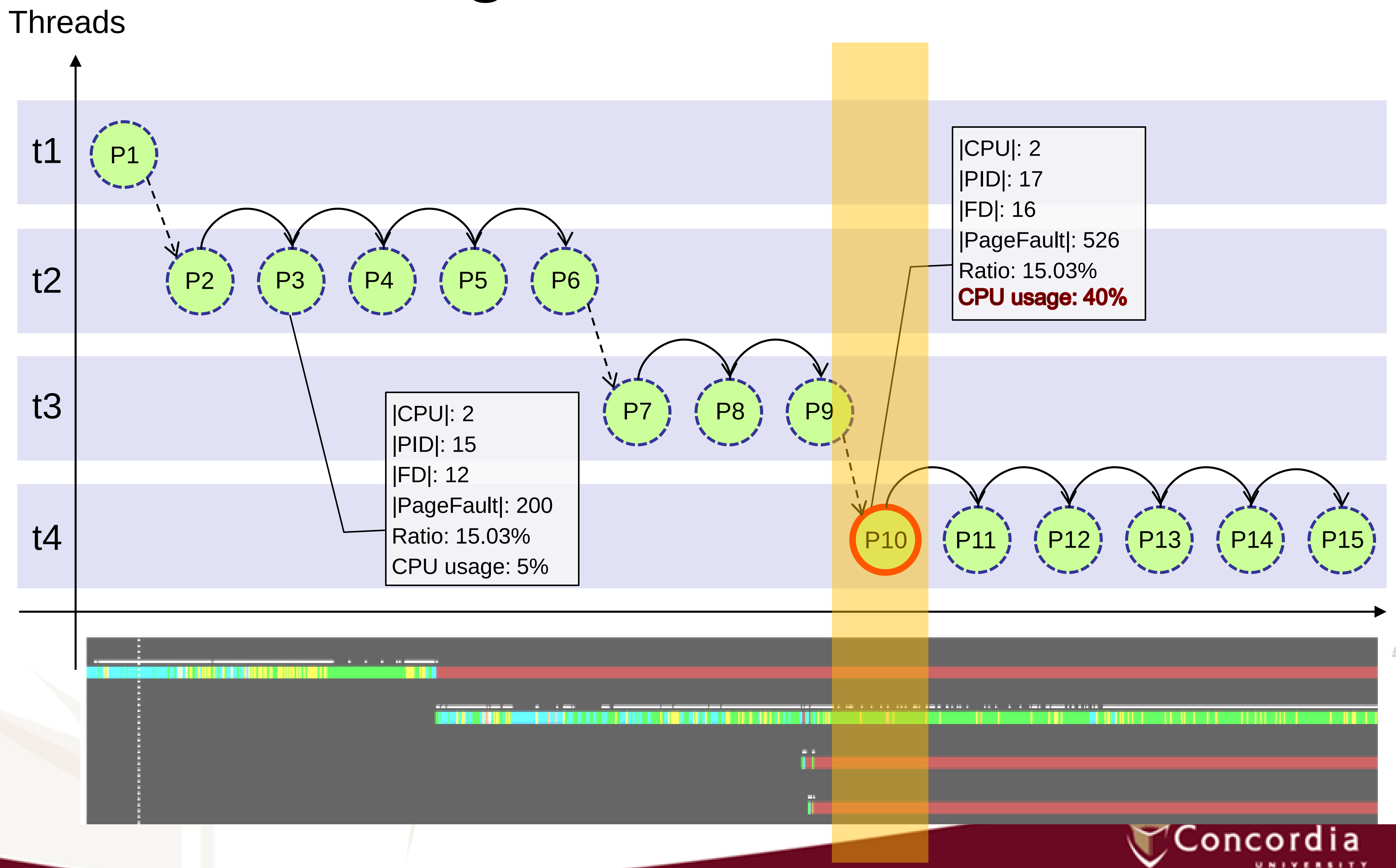
Phase flow diagram



Combining user and kernel space



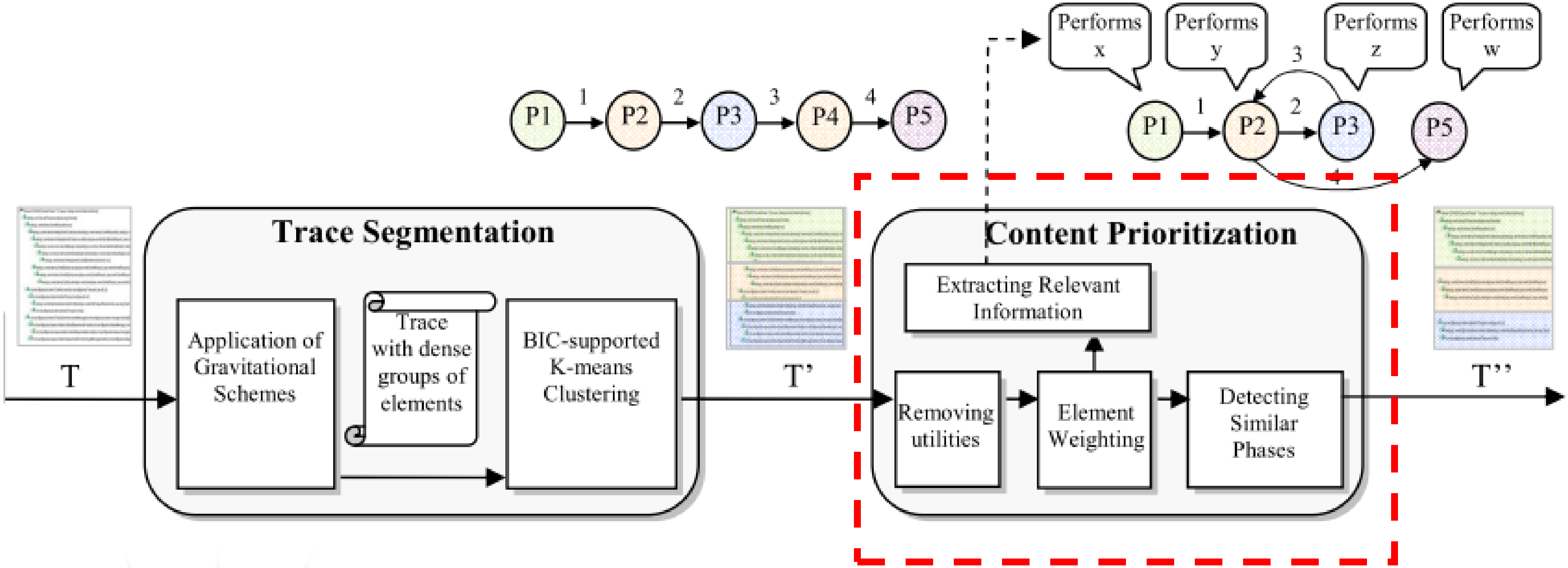
Adding state information



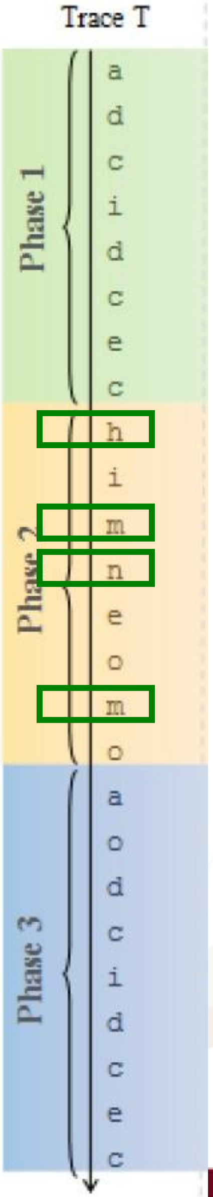
|CPU|: 2
|PID|: 15
|FD|: 12
|PageFault|: 200
Ratio: 15.03%
CPU usage: 5%

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

Trace abstraction framework

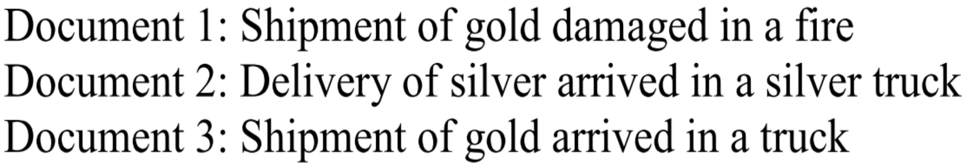


Extracting relevant components

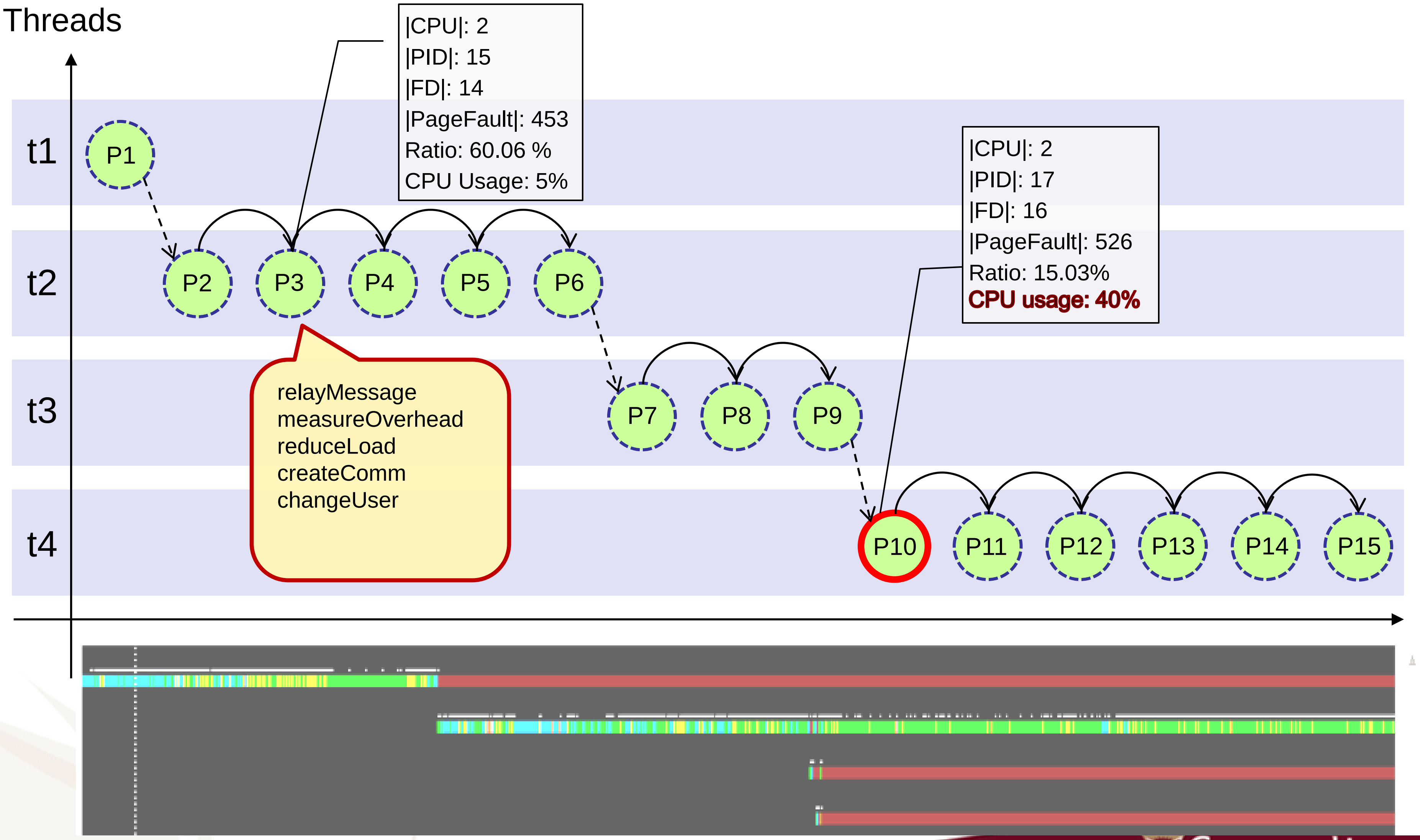


- Idea: Elements that are repeated in a phase but are not much shared between phases indicate their relevance to the phase

- 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

Identifying main content

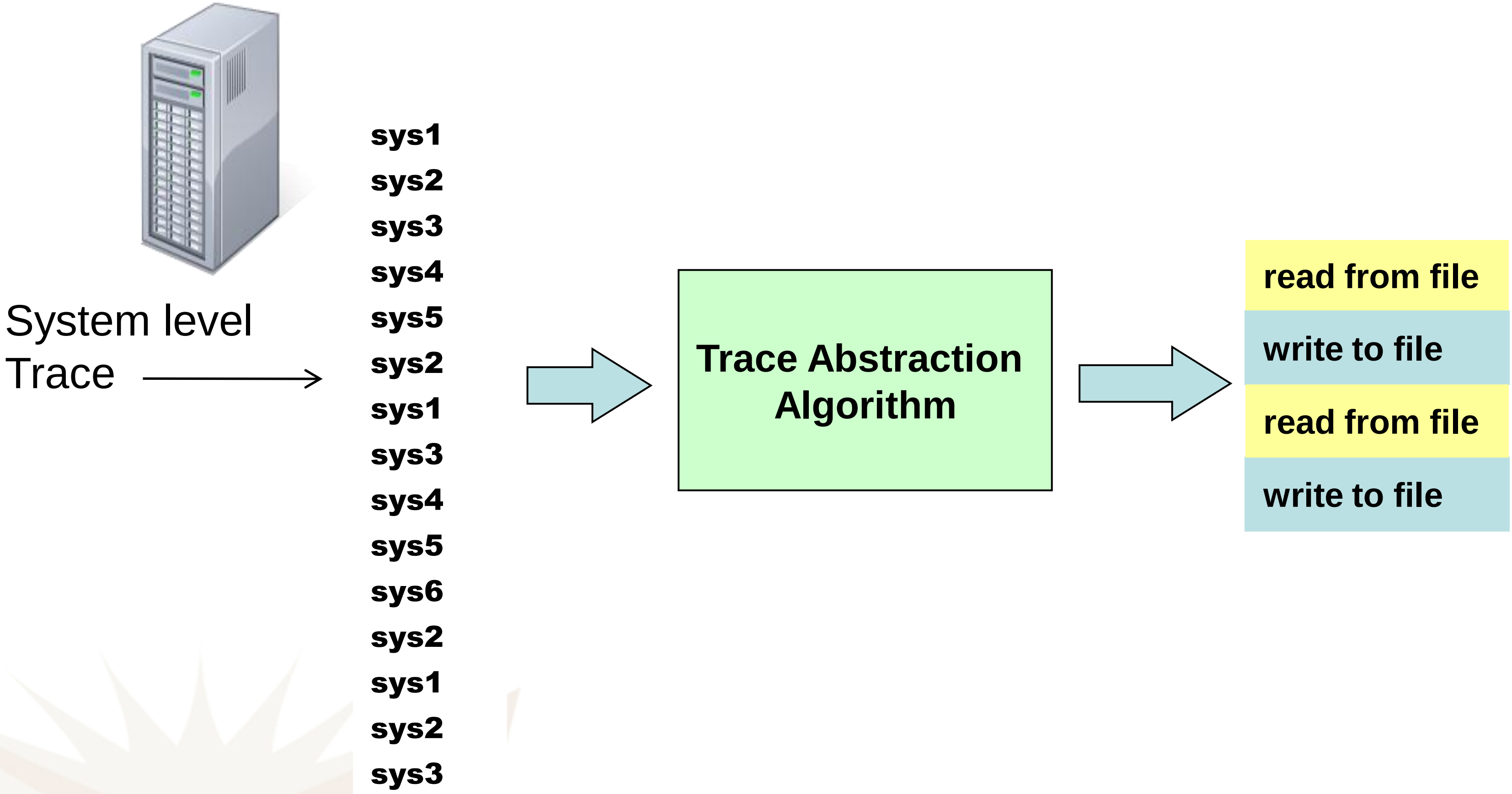


|CPU|: 2
|PID|: 15
|FD|: 14
|PageFault|: 453
Ratio: 60.06 %
CPU Usage: 5%

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

relayMessage
measureOverhead
reduceLoad
createComm
changeUser

Abstraction of System Call Traces



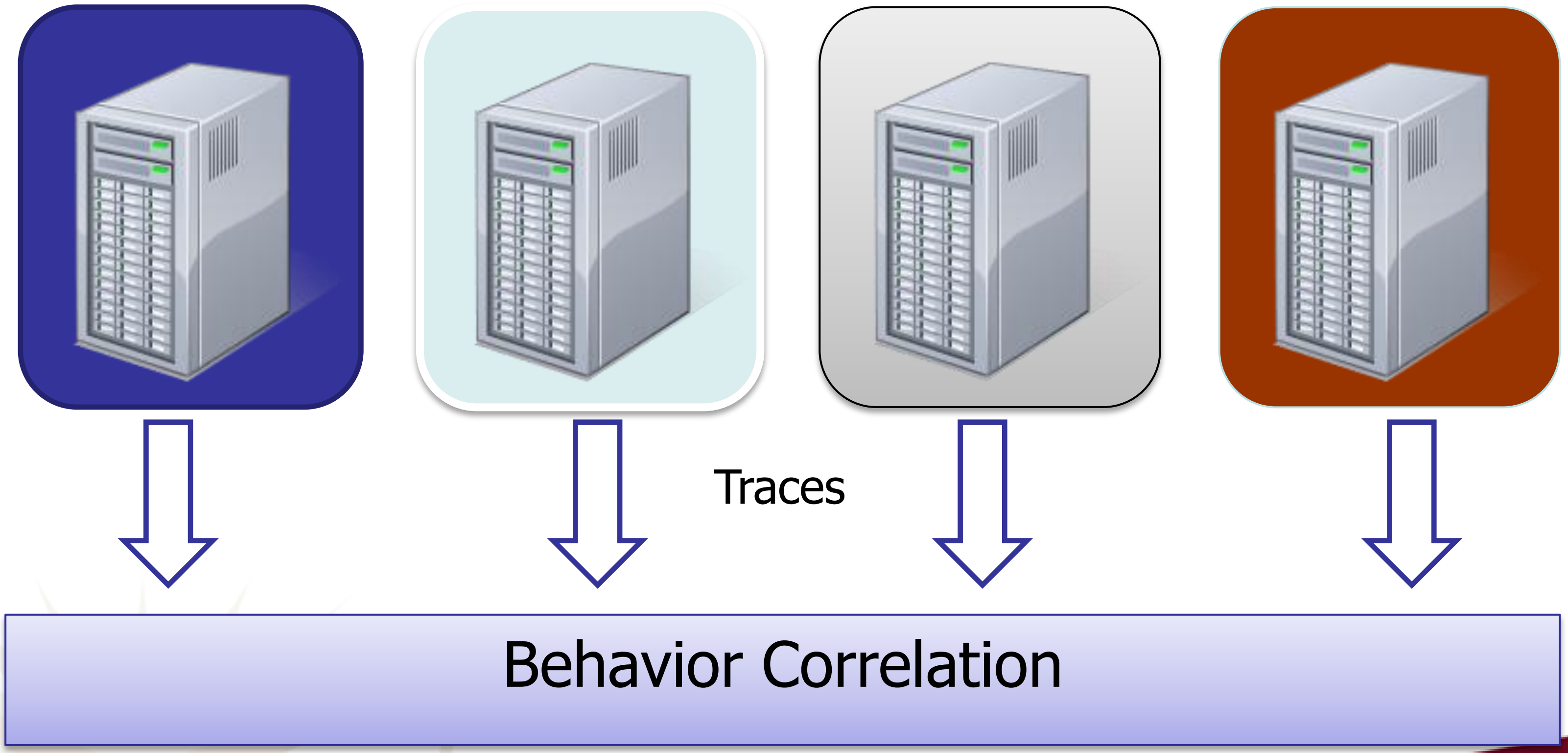
System level Trace

sys1
sys2
sys3
sys4
sys5
sys2
sys1
sys3
sys4
sys5
sys6
sys2
sys1
sys2
sys3

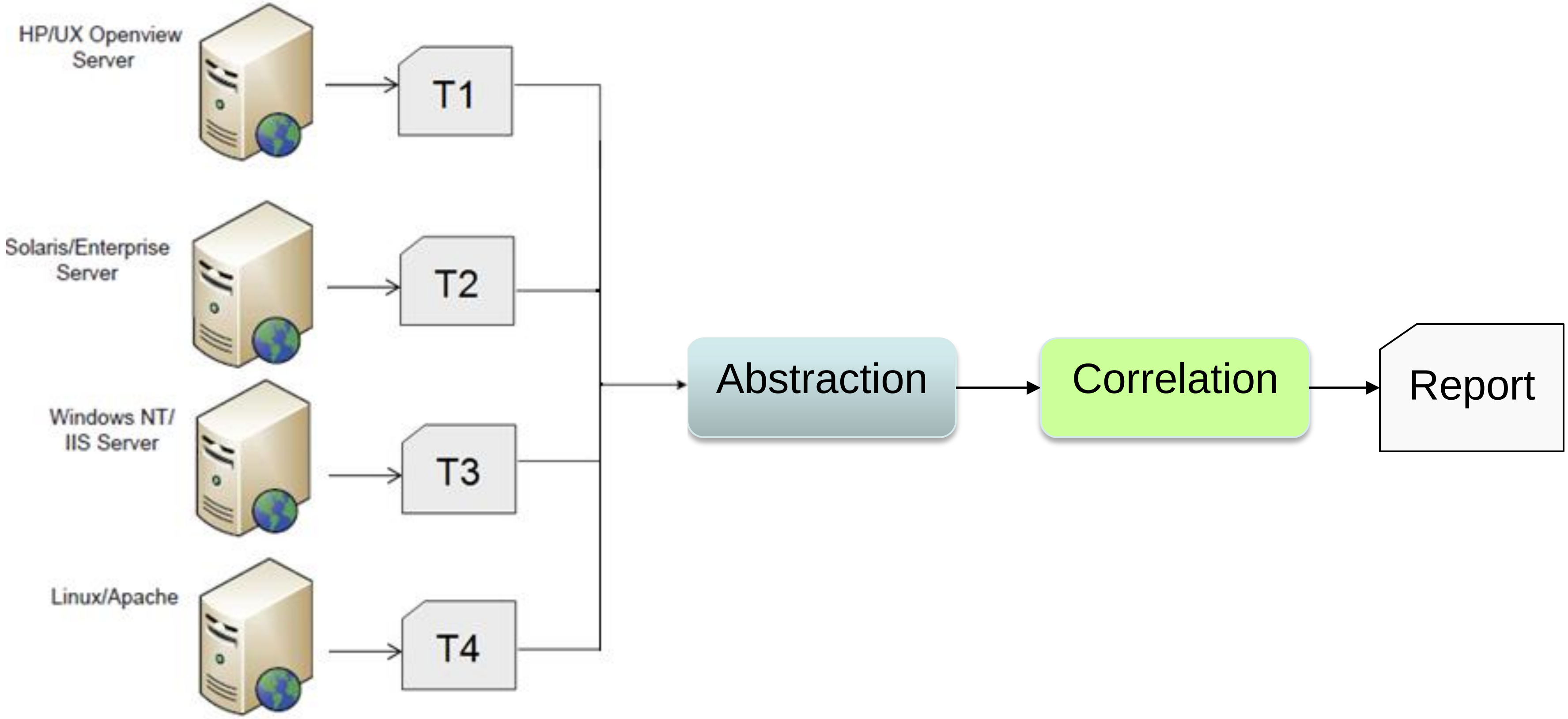
Trace Abstraction Algorithm

read from file
write to file
read from file
write to file

Fault Tolerance: Redundancy and Diversity

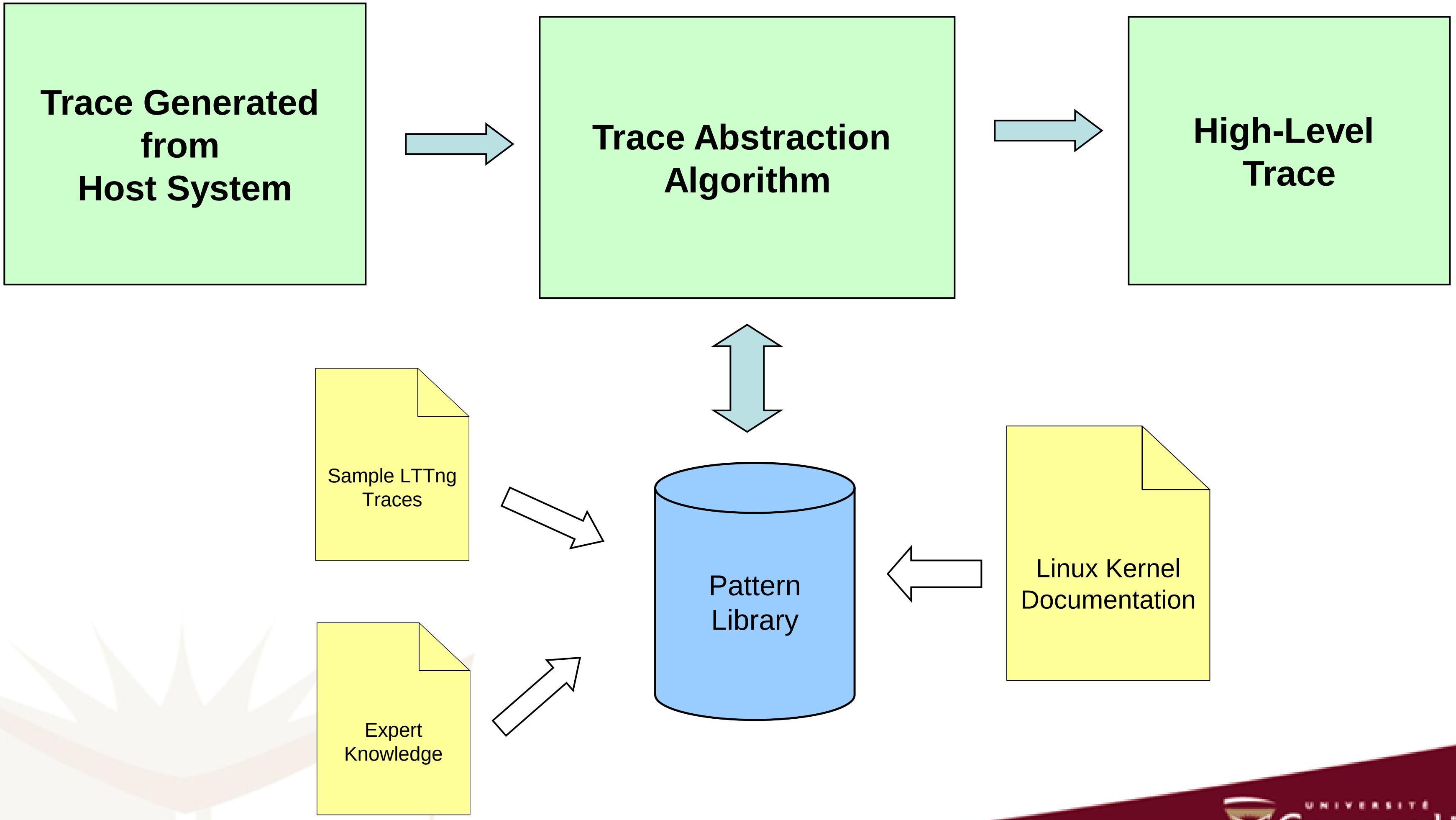


OS Diversity



A. Hamou-Lhadj et al., CISDA’12

Kernel-Level Trace Abstraction



Trace Generated from Host System

Trace Abstraction Algorithm

High-Level Trace

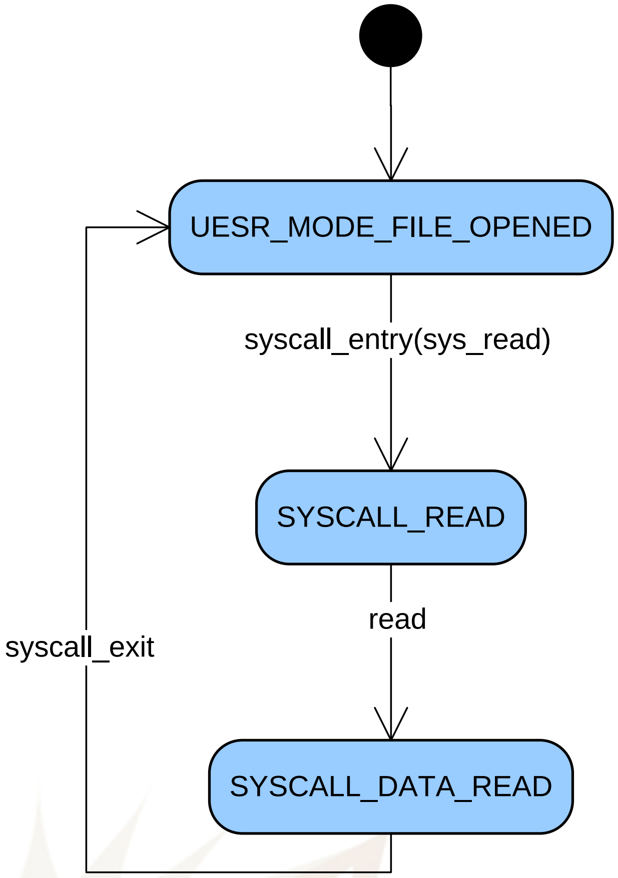
Sample LTTng Traces

Expert Knowledge

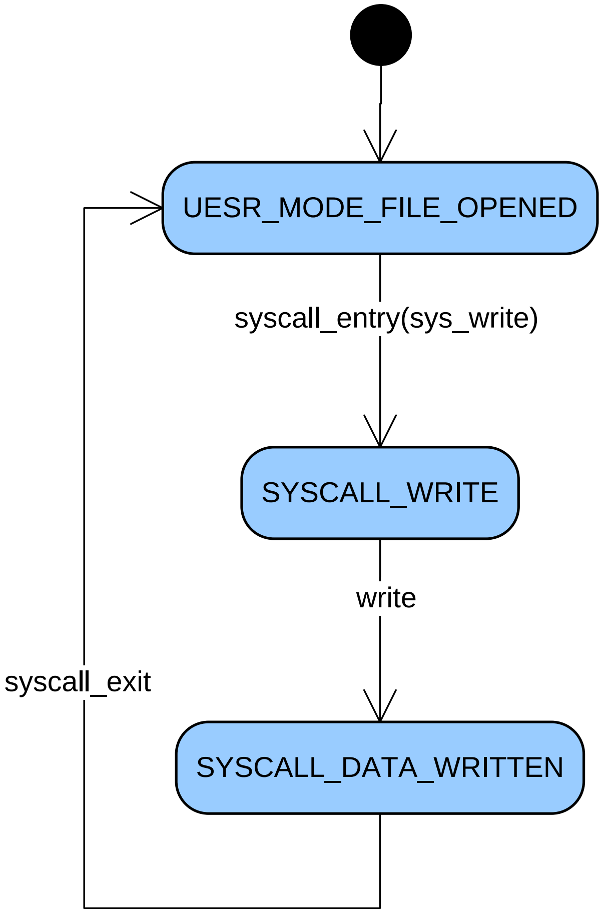
Pattern Library

Linux Kernel Documentation

File Read & Write Patterns



Write to File

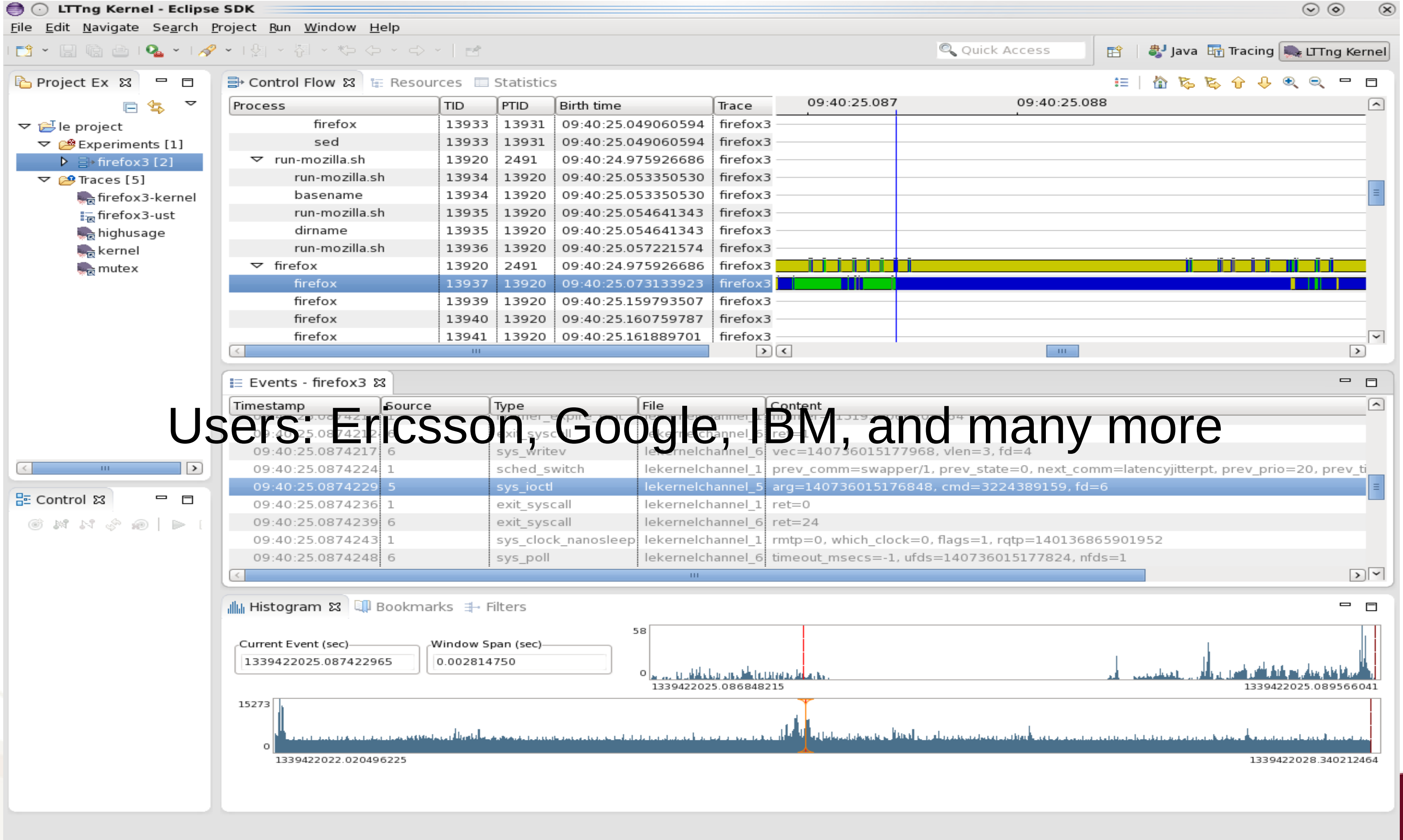


Read from File

Evaluation

- Two nodes: Linux and BSD
- Failures are simulated on BSD
- We are able to detect and recover from most failures
- Abstraction is a crucial step for behavior correlation to be effective
- Similarity based on pattern detection provides accurate measures

Tracing and Monitoring Framework



Process	TID	PTID	Birth time	Trace
firefox	13933	13931	09:40:25.049060594	firefox3
sed	13933	13931	09:40:25.049060594	firefox3
run-mozilla.sh	13920	2491	09:40:24.975926686	firefox3
run-mozilla.sh	13934	13920	09:40:25.053350530	firefox3
basename	13934	13920	09:40:25.053350530	firefox3
run-mozilla.sh	13935	13920	09:40:25.054641343	firefox3
dirname	13935	13920	09:40:25.054641343	firefox3
run-mozilla.sh	13936	13920	09:40:25.057221574	firefox3
firefox	13920	2491	09:40:24.975926686	firefox3
firefox	13937	13920	09:40:25.073133923	firefox3
firefox	13939	13920	09:40:25.159793507	firefox3
firefox	13940	13920	09:40:25.160759787	firefox3
firefox	13941	13920	09:40:25.161889701	firefox3

Timestamp	Source	Type	File	Content
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
09:40:25.0874212	6	exit_syscall	lekernelchannel_6	[illegible]
09:40:25.0874217	6	sys_writev	lekernelchannel_6	vec=140736015177968, vlen=3, fd=4
09:40:25.0874224	1	sched_switch	lekernelchannel_1	prev_comm=swapper/1, prev_state=0, next_comm=latencyjitterpt, prev_prio=20, prev_ti
09:40:25.0874229	5	sys_ioctl	lekernelchannel_5	arg=140736015176848, cmd=3224389159, fd=6
09:40:25.0874236	1	exit_syscall	lekernelchannel_1	ret=0
09:40:25.0874239	6	exit_syscall	lekernelchannel_6	ret=24
09:40:25.0874243	1	sys_clock_nanosleep	lekernelchannel_1	rmtp=0, which_clock=0, flags=1, rqtp=140136865901952
09:40:25.0874248	6	sys_poll	lekernelchannel_6	timeout_msecs=-1, ufds=140736015177824, nfds=1

Current Event (sec): 1339422025.087422965

Window Span (sec): 0.002814750

Users: Ericsson, Google, IBM, and many more

Diagnostics for Real Time Distributed Multi-Core Architecture in Avionics

Build efficient algorithms for low overhead, low disturbance tracing of real-time embedded multi-core systems and simulators

Develop special purpose performance analysis debugging, and feature location modules for avionic systems

Project Partners

CAE

OPAL·RT

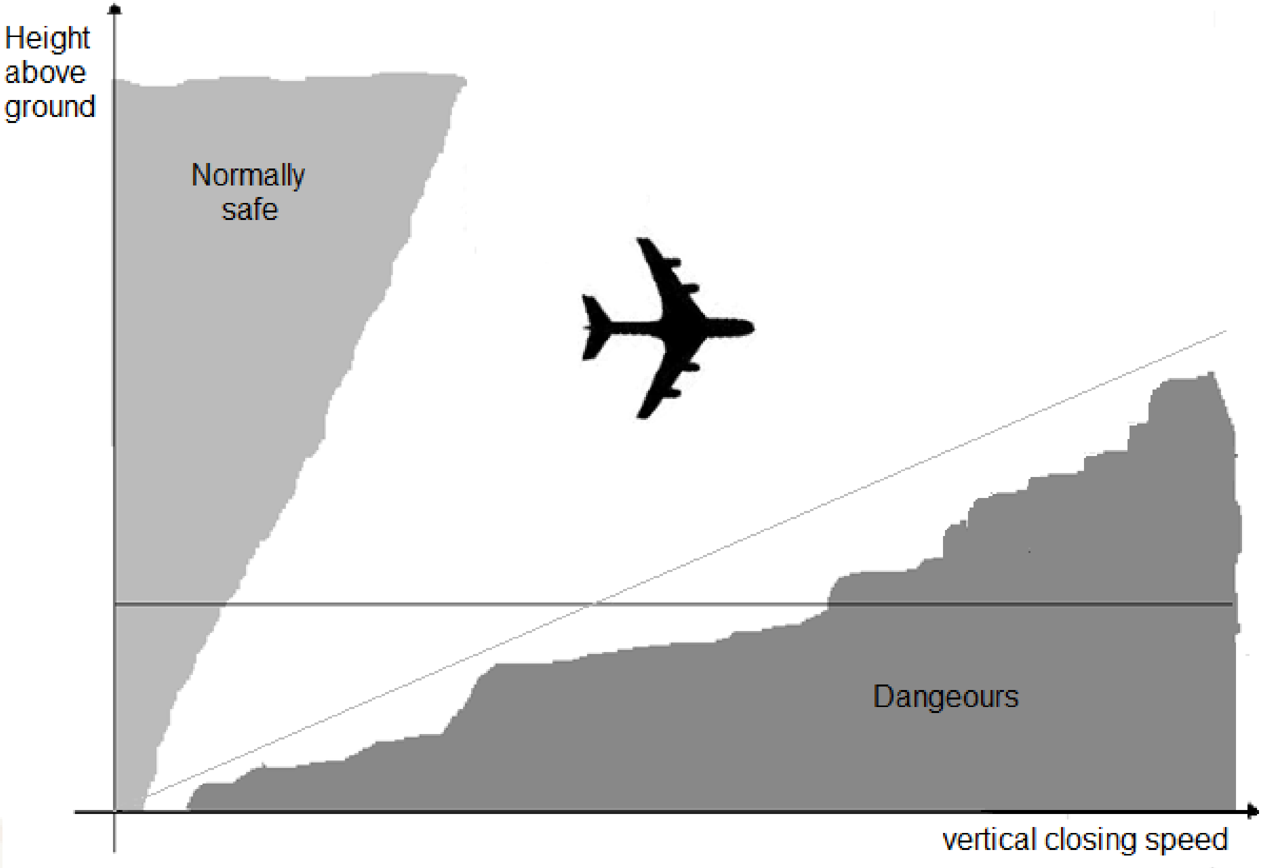
NSERC
CRSNG

CRIAQ
www.criaq.aero

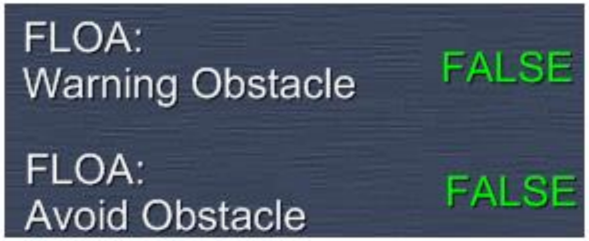
UNIVERSITÉ
Concordia
UNIVERSITY

ÉCOLE
POLYTECHNIQUE
MONTRÉAL

Motivating scenario



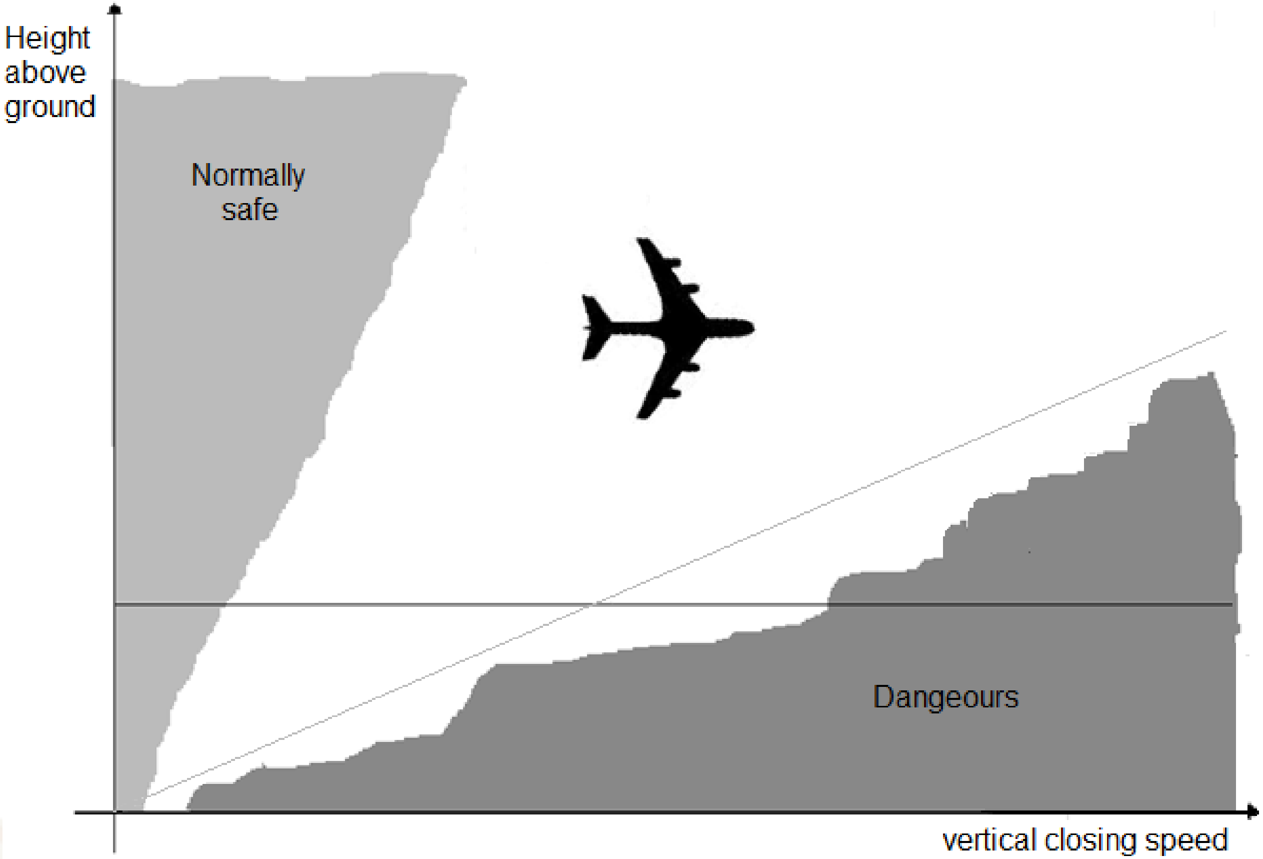
IOS

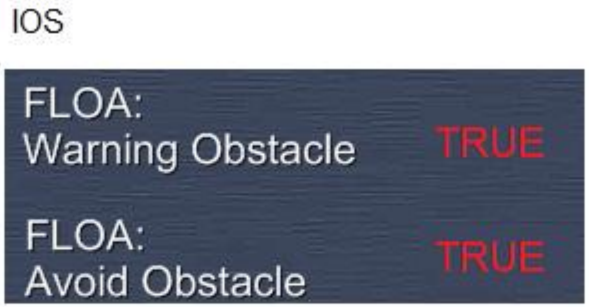


FLOA: Warning Obstacle FALSE

FLOA: Avoid Obstacle FALSE

Motivating scenario



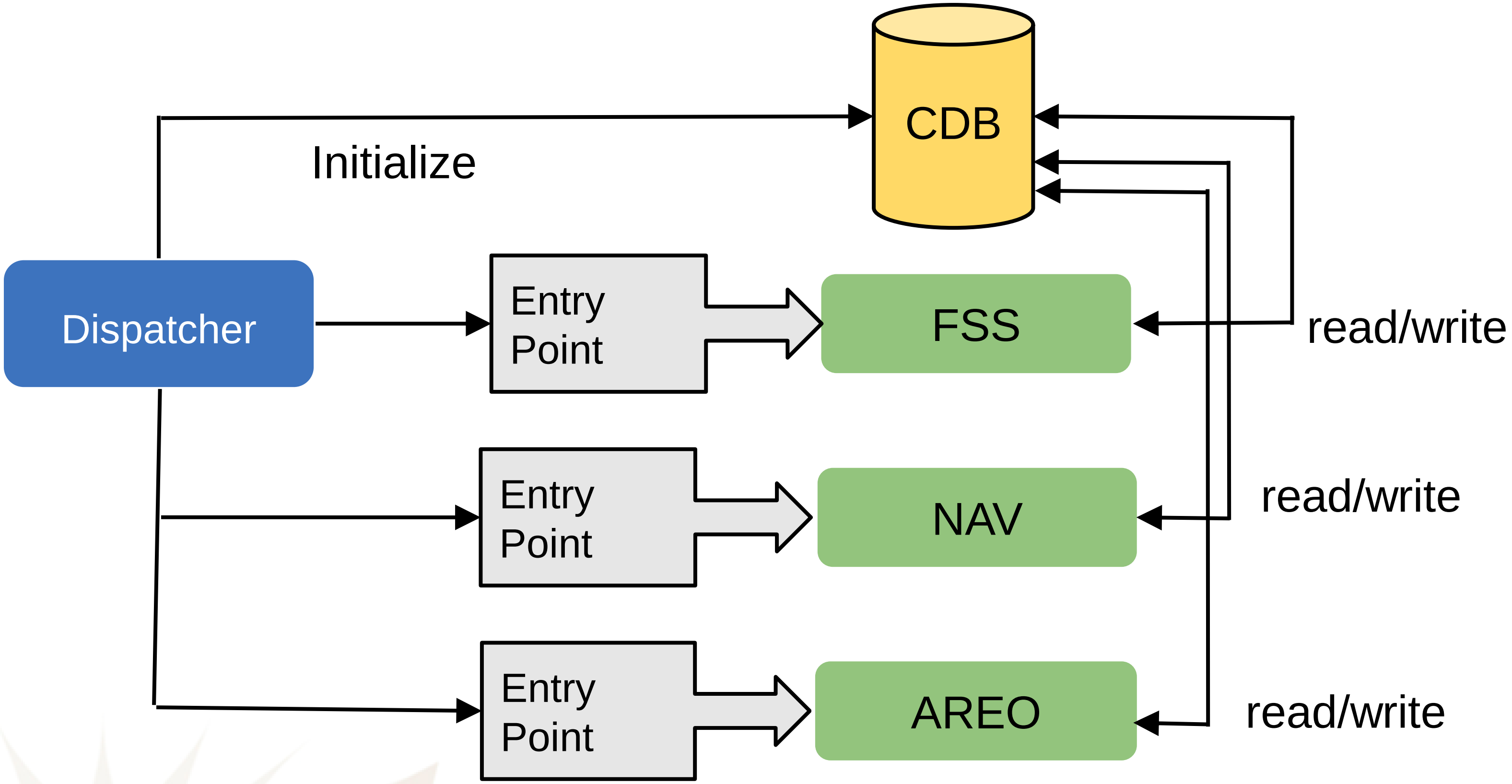


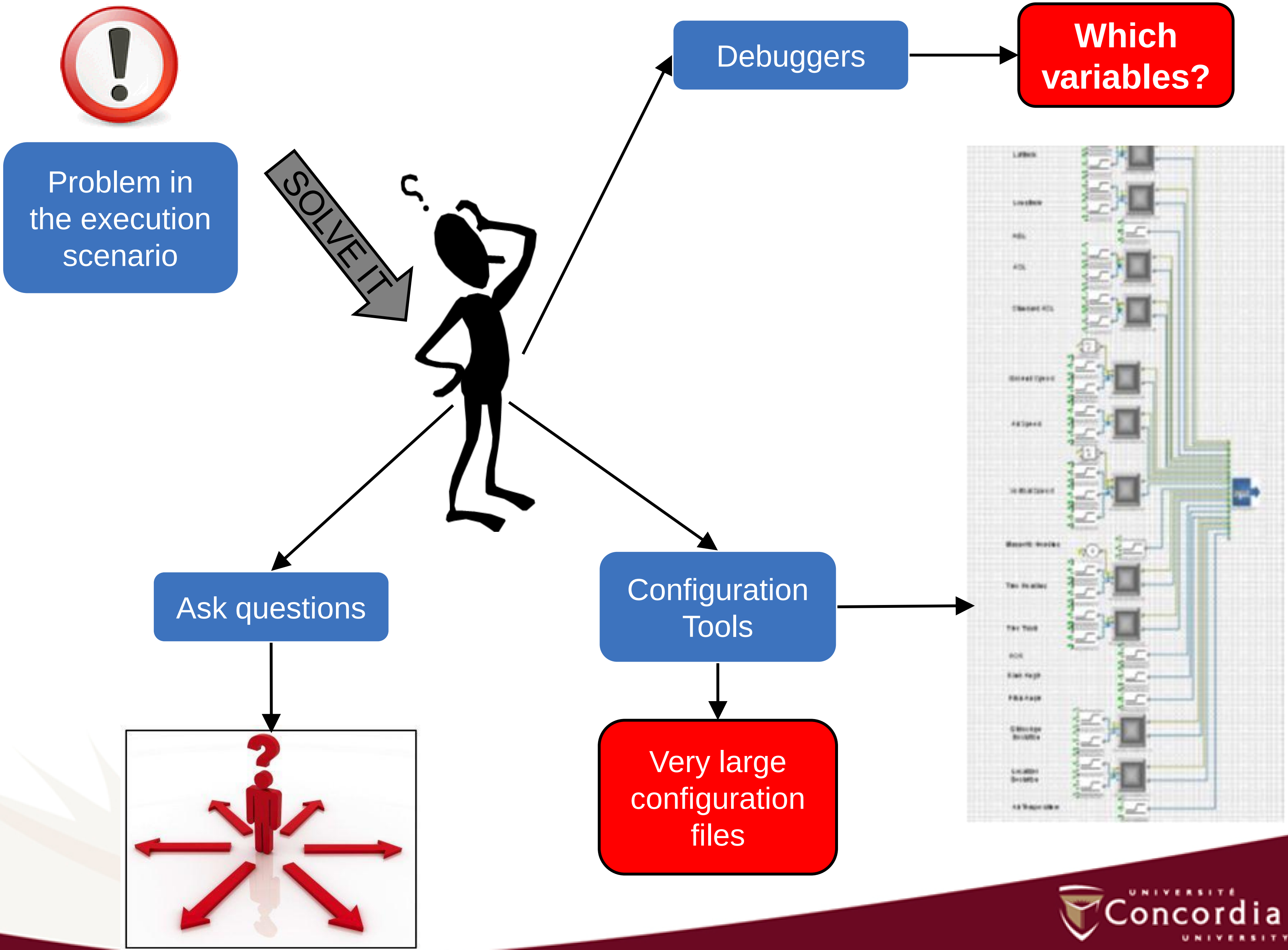
IOS

FLOA: Warning Obstacle TRUE

FLOA: Avoid Obstacle TRUE

CAE - Architecture





Problem in the execution scenario

SOLVE IT

Debuggers

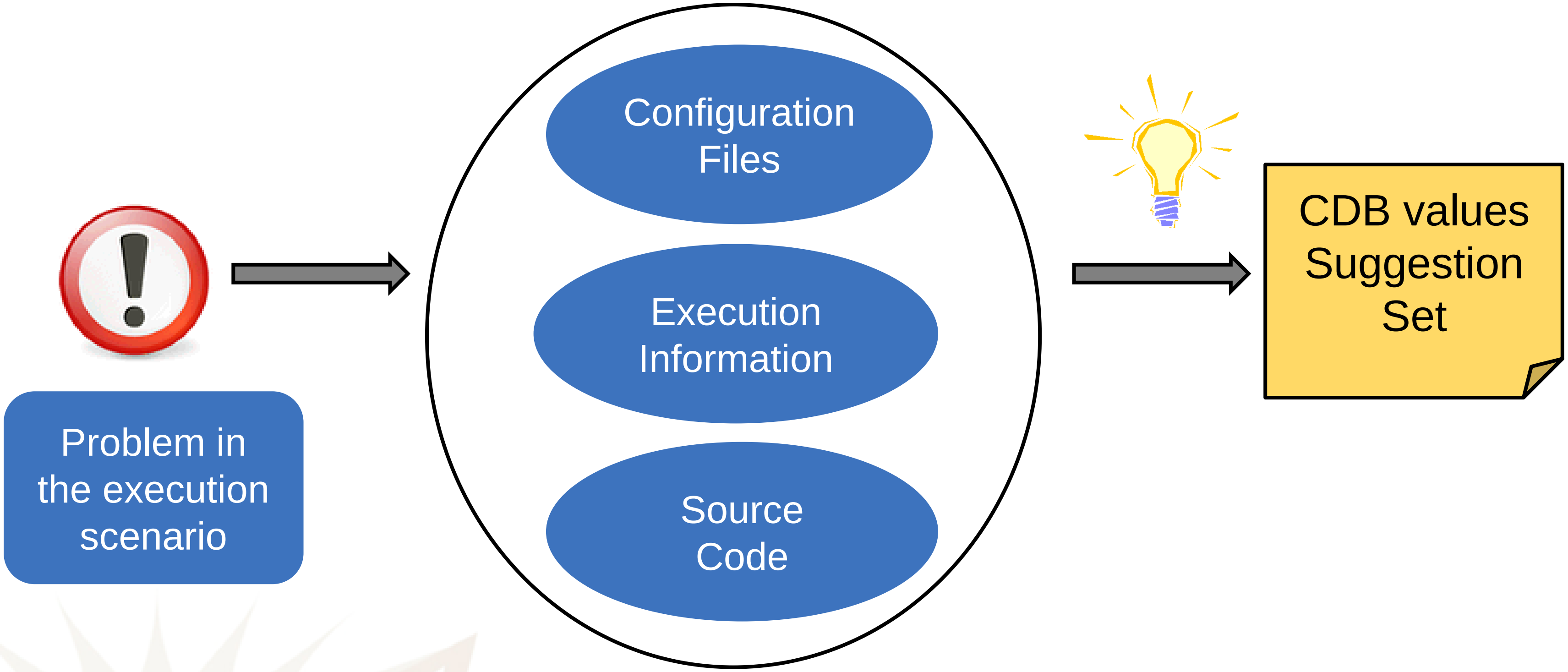
Which variables?

Ask questions

Configuration Tools

Very large configuration files

Need: Automatic identification of CDB values for a specific failure



Problem in the execution scenario

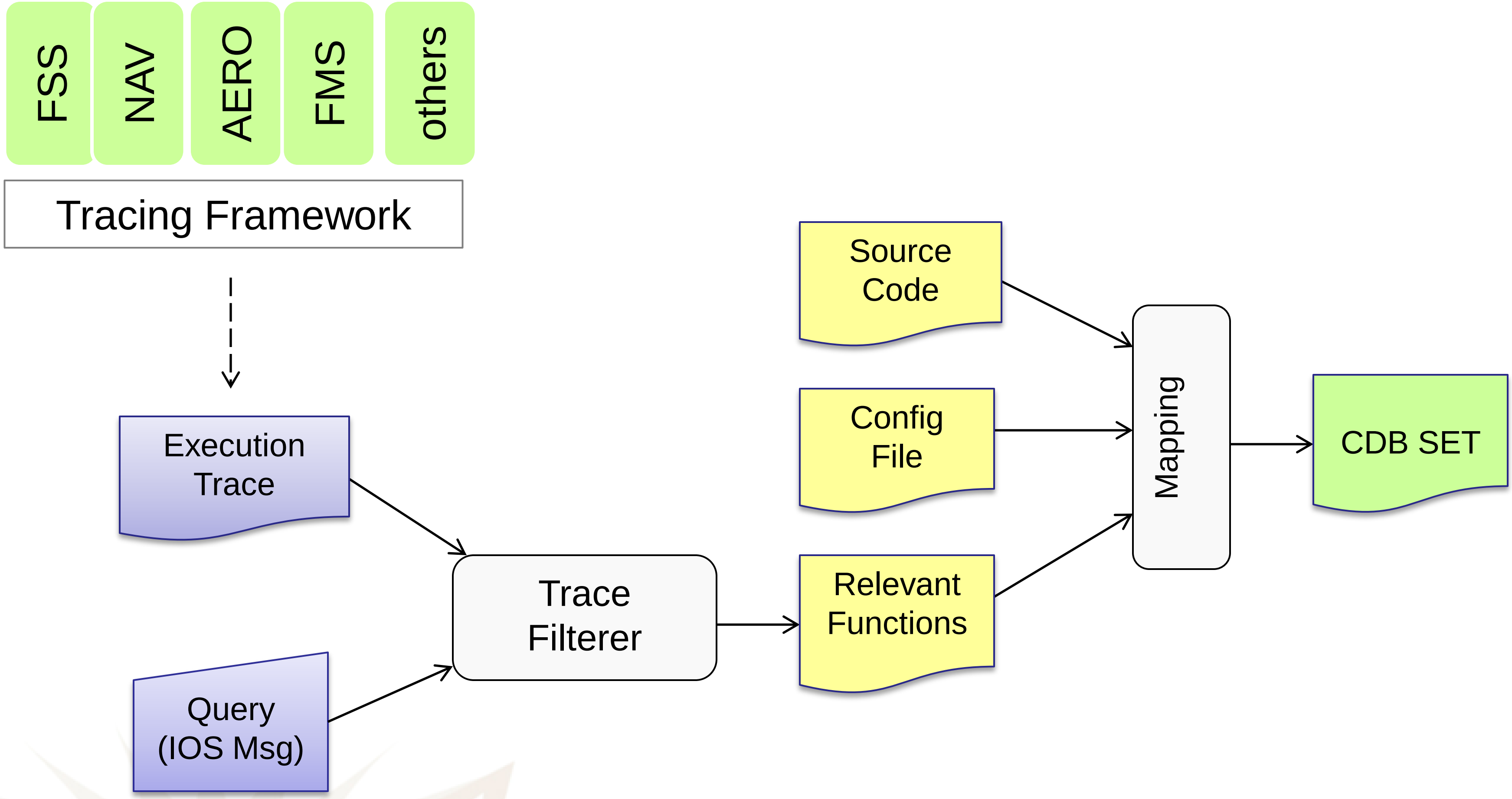
Configuration Files

Execution Information

Source Code

CDB values Suggestion Set

Proposed Solution



Evaluation

Scenario	Aircraft Condition	Trace size (millions)	# CDB vars in config.	Relevant CDBs	Retrieved CDBs	Precision	Recall
TAWS Mode1	Altitude: 900 feet Vertical speed: -3000 feet/min	20	1720	4	1	25%	50%
TAWS Mode4B	Altitude: 300 feet Airspeed: 50 knots Gears Position: down Flaps Position: in flight	8	1620	4	19	21%	100%
TAWS Mode4A	Altitude: 400 feet Airspeed: 50 knots Gears Position: up Flaps Position: landing	4	1499	5	28	19%	100%

Finding Faulty Functions from the Traces of Field Failures

Improve the troubleshooting process to increase the productivity of software engineers by reducing the number of field reports to be analysed

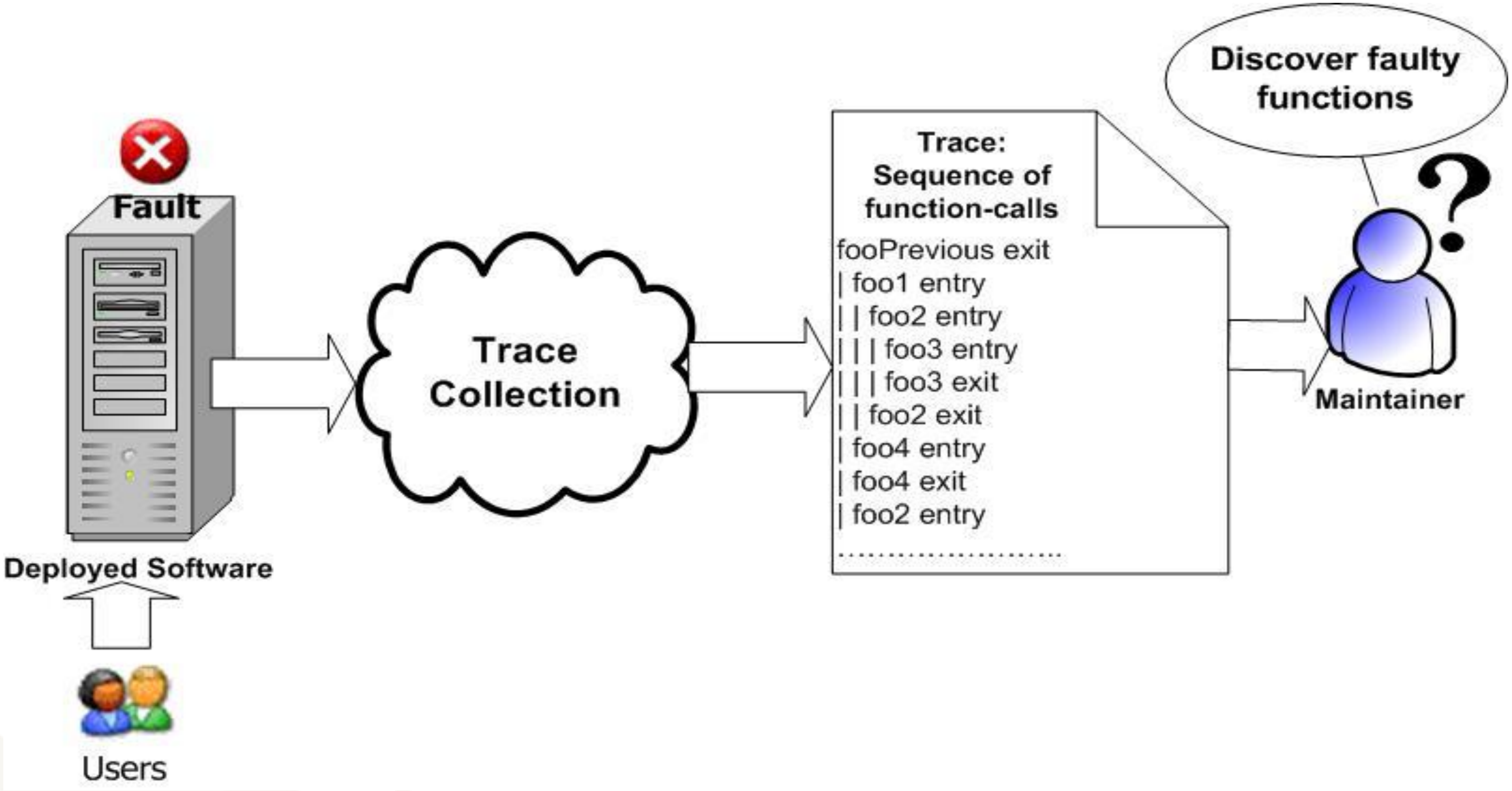
IBM®

NSERC CRSNG

Western
UNIVERSITY · CANADA

UNIVERSITÉ Concordia UNIVERSITY

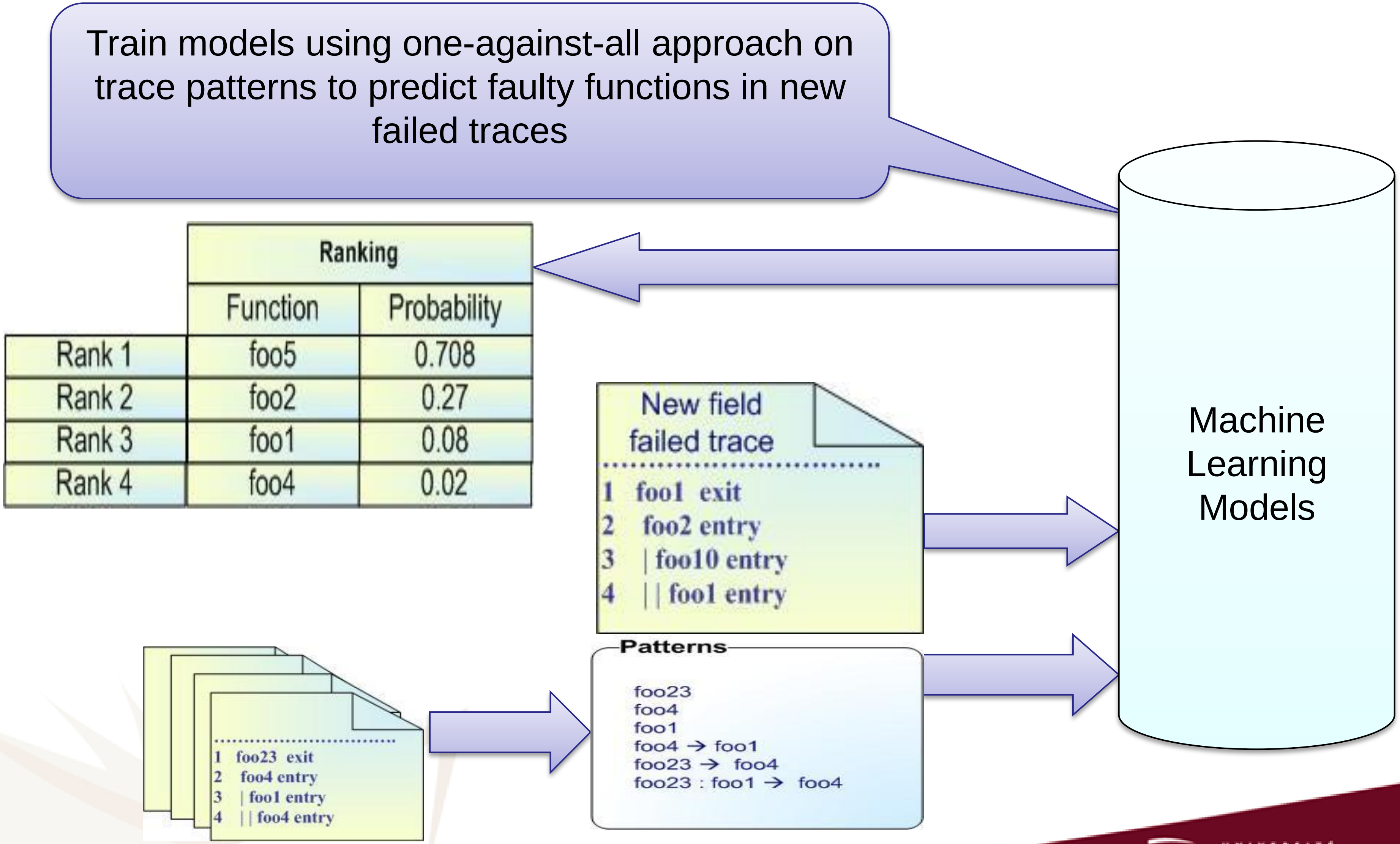
Finding Faulty Functions from the Traces of Field Failures



Trace:
Sequence of
function-calls

```
fooPrevious exit
| foo1 entry
| | foo2 entry
| | | foo3 entry
| | | foo3 exit
| | foo2 exit
| foo4 entry
| foo4 exit
| foo2 entry
........................
```

Approach



Train models using one-against-all approach on trace patterns to predict faulty functions in new failed traces

	Ranking	
	Function	Probability
Rank 1	foo5	0.708
Rank 2	foo2	0.27
Rank 3	foo1	0.08
Rank 4	foo4	0.02

New field failed trace

```
1  foo1  exit
2   foo2 entry
3    | foo10 entry
4    | | foo1 entry
```

Patterns

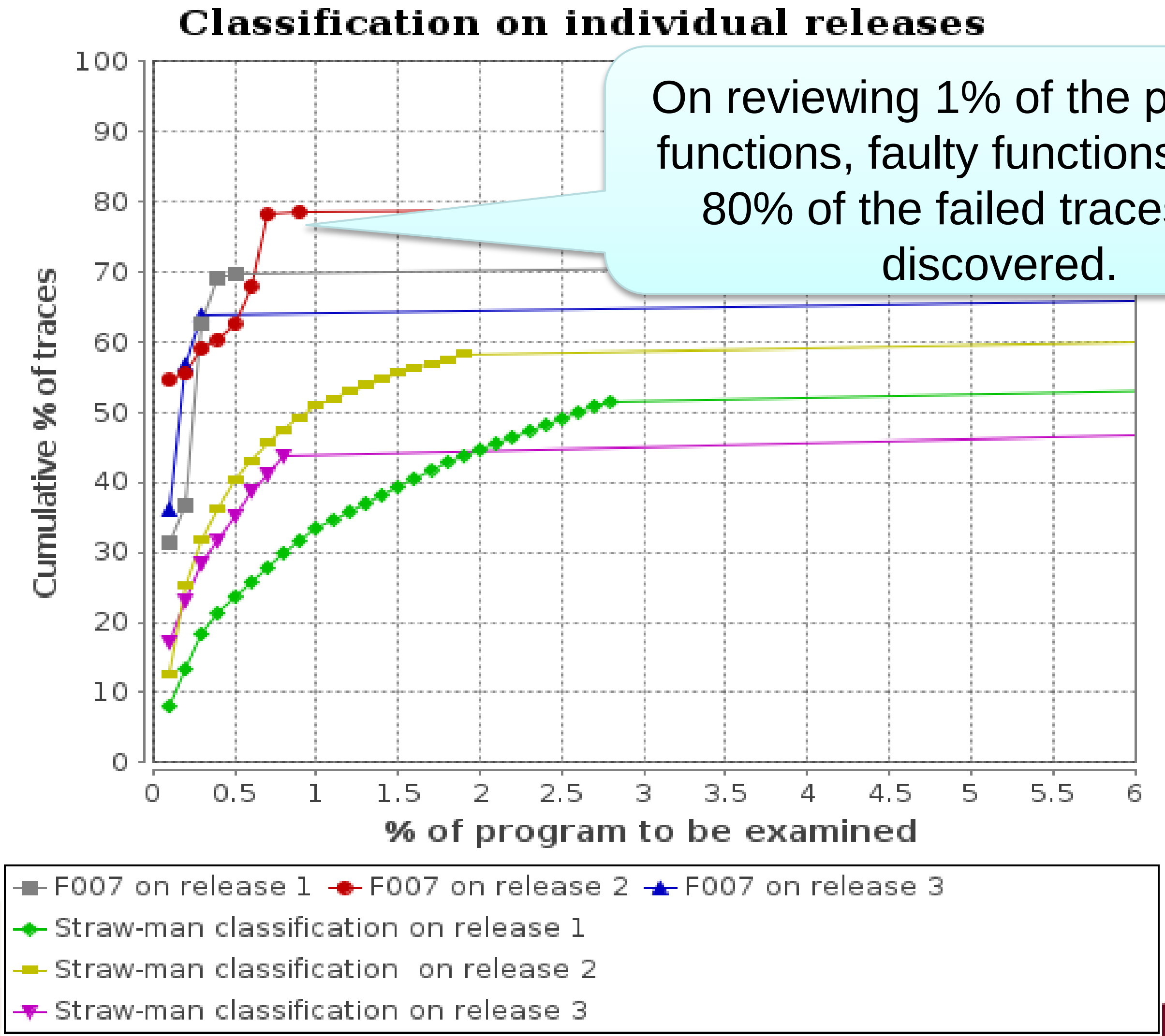
```
foo23
foo4
foo1
foo4 → foo1
foo23 → foo4
foo23 : foo1 → foo4
```

```
1  foo23  exit
2   foo4 entry
3   | foo1 entry
4   | | foo4 entry
```

Application to the IBM system

<table>
<tr><td colspan="4">20+ million LOC, 300+ components, approx. 200 K+ functions, traces of size up to 4GB (44 million function-calls), and 82% rediscoveries of field faults.</td></tr>
<tr><td></td><td># Failed Traces</td><td># Faulty Comp.</td><td># Faulty Func.</td></tr>
<tr><td>Release 1</td><td>269</td><td>52</td><td>65</td></tr>
<tr><td>Release 2</td><td>337</td><td>35</td><td>47</td></tr>
<tr><td>Release 3</td><td>99</td><td>30</td><td>31</td></tr>
<tr><td colspan="2">Total Distinct Faults (Union)</td><td>65</td><td>103</td></tr>
</table>

Results on the IBM system



On reviewing 1% of the program's functions, faulty functions in up to 80% of the failed traces were discovered.

From Data To Knowledge for Better System Maintenance - D2K Project

Enable and implement efficient use of analytical techniques to achieve revenue targets within risk limits by continuously improving the end-to-end software maintenance process

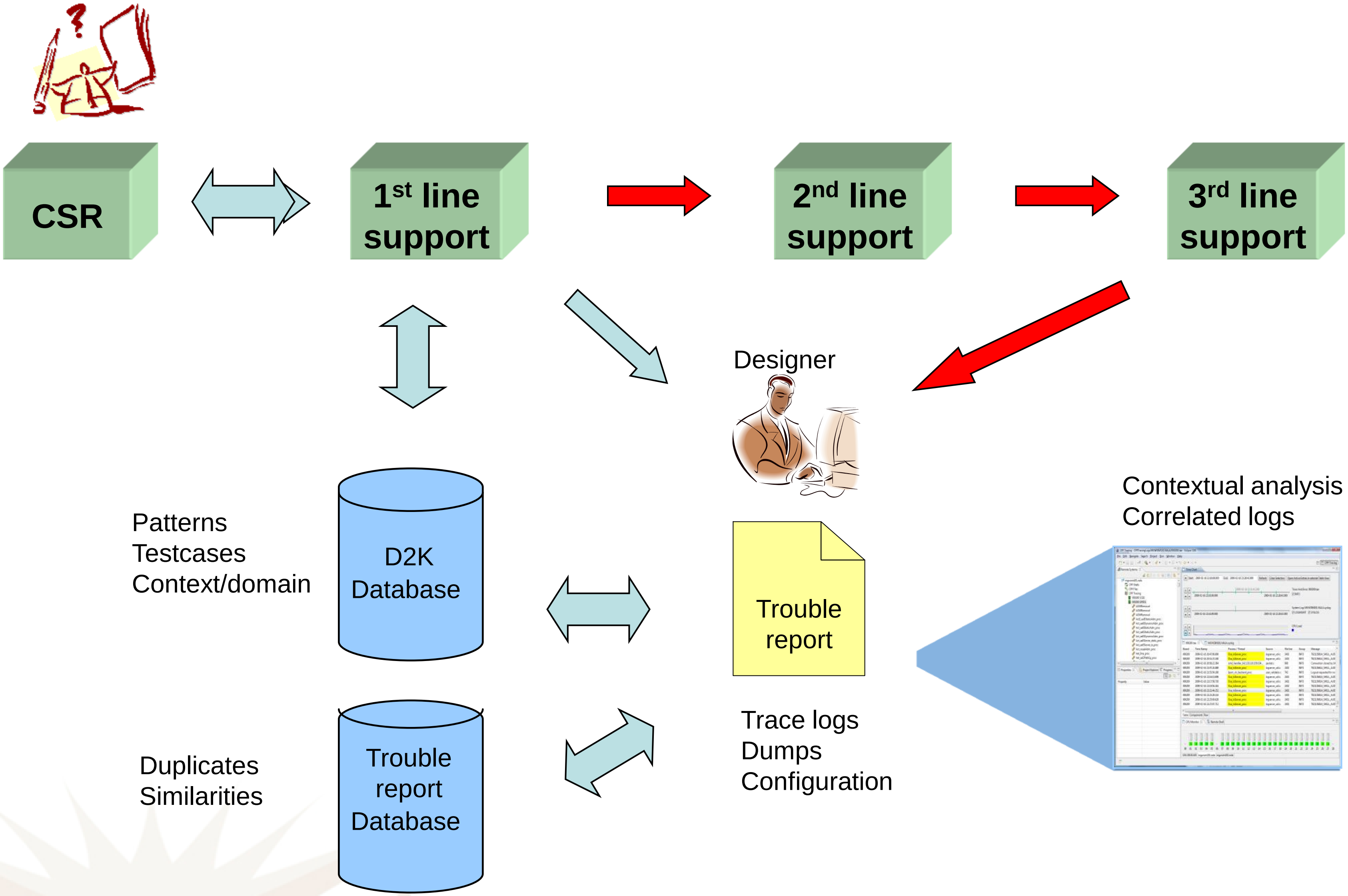
ERICSSON

Mitacs

UNIVERSITÉ Concordia UNIVERSITY

D2K Objectives

1. Identify changes to improve current software maintenance process and information flow
2. Investigate automated solutions for fault discovery, diagnosis, and prediction
3. Provide better analysis capabilities to software engineers
4. Help software engineers focus on the real problem rather than spending time on irrelevant information



CSR

1st line support

2nd line support

3rd line support

Designer

Patterns
Testcases
Context/domain

D2K Database

Trouble report

Contextual analysis
Correlated logs

Duplicates
Similarities

Trouble report Database

Trace logs
Dumps
Configuration

Conclusion

Trace analysis is useful for many software engineering applications including software maintenance and evolution, performance analysis, software resilience, and cyber security

Future

Invest in an end-to-end Enterprise Tracing Platform (ETP) for trace generation, modeling, abstraction, and analytics to support forward and background engineering tasks

Merci!