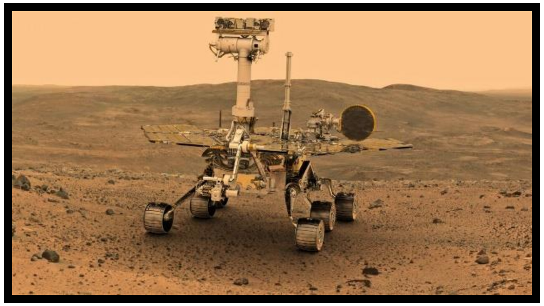
A fault-injection methodology for the system-level reliability analysis of computing systems modeled in SystemC

Antonio Miele – antonio.miele@polimi.it

Motivations

- Widespread adoption of complex computing systems (e.g. multi-cores) in mission-/safety-critical applications



Images downloaded from google

Motivations

- Widespread adoption of complex computing systems (e.g. multi-cores) in mission-/safety-critical applications

- Necessity to perform a reliability-aware design of computing systems

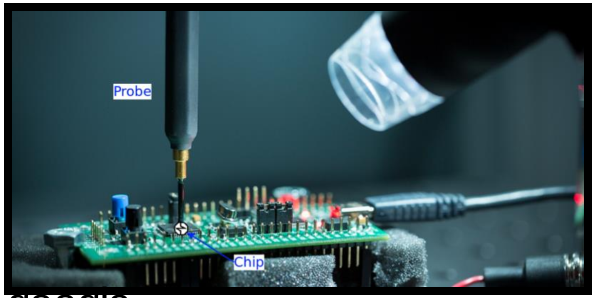
Images downloaded from google

Motivations

- Widespread adoption of complex computing systems (e.g. multi-cores) in mission-/safety-critical applications

- Necessity to perform a reliability-aware design of computing systems

- Fault injection is the common tool used for reliability analysis

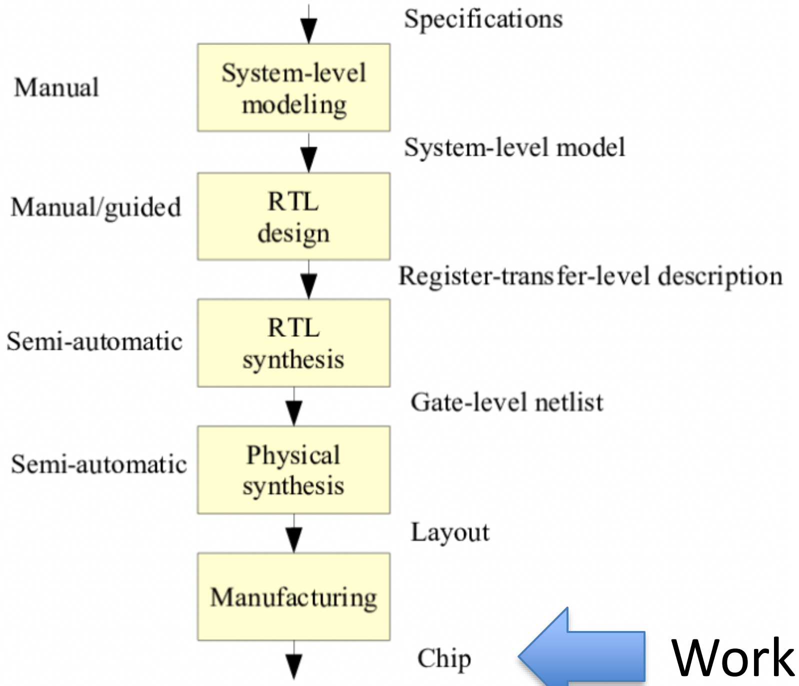


Images downloaded from google

Motivations

Limitations in common practices for fault injection:

– Performing fault injection only in the late design stages



Working on the final chip

Images downloaded from google

Motivations

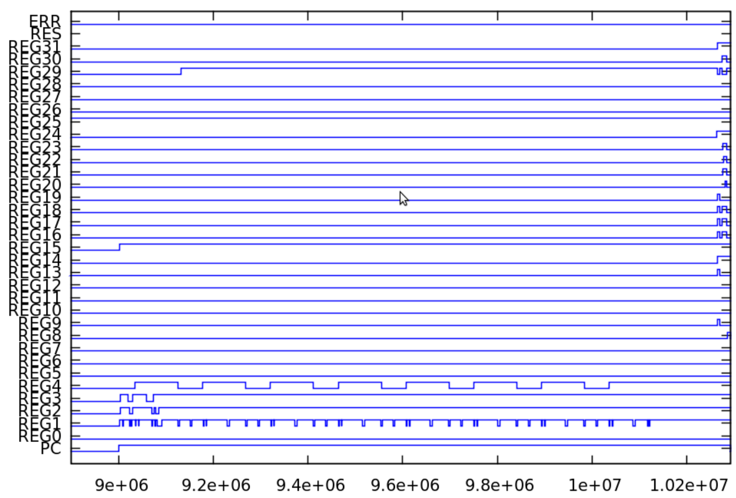
Limitations in common practices for fault injection:

- Performing fault injection only in the late design stages
- Analyzing manly final results

Motivations

Limitations in common practices for fault injection:

- Performing fault injection only in the late design stages
- Analyzing manly final results
- When performing monitoring on internal memory elements, analyzing raw traces



Goal of the work

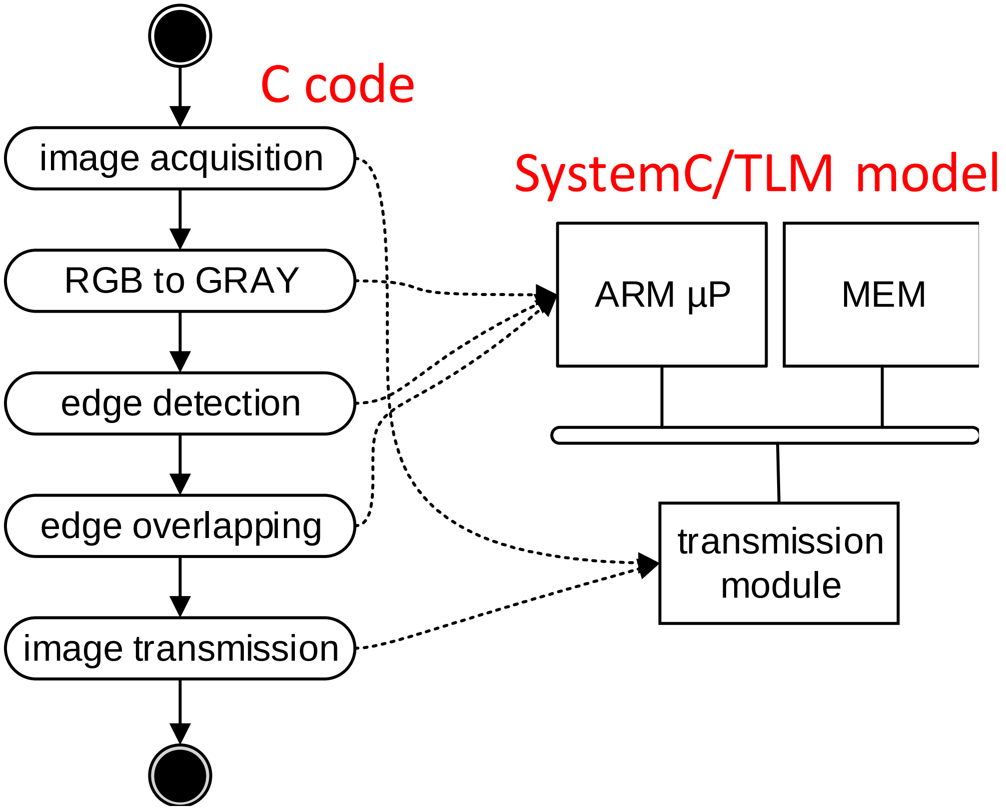
A framework and a methodology for **system-level fault-injection-based reliability analysis** in multi-cores specified in SystemC/TLM

Key points:

- Support for an accurate definition of the fault campaign
- Capability to perform error monitoring at both architecture and application level
- Customizable error analysis and classification approach

Reference system

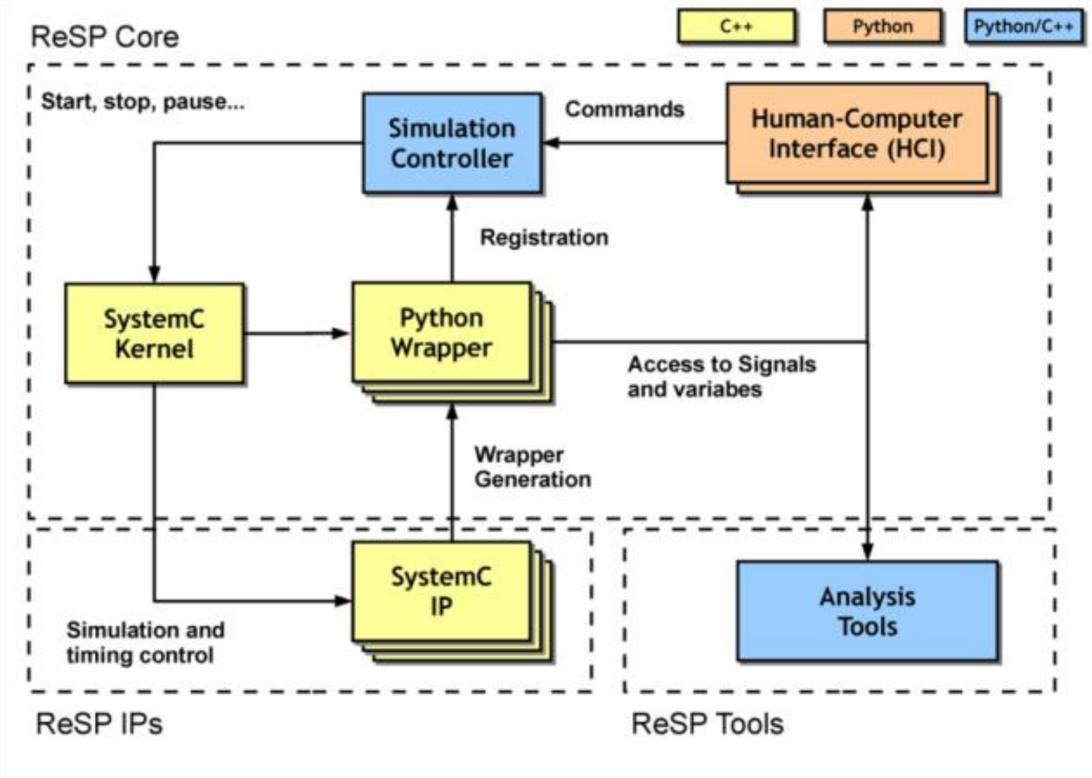
- The hardware is composed of one or various processors and HW modules
- The application is organized into tasks/functions
- Tasks are mapped on the various units



Background: ReSP

ReSP is a system-level simulation platform for multicores

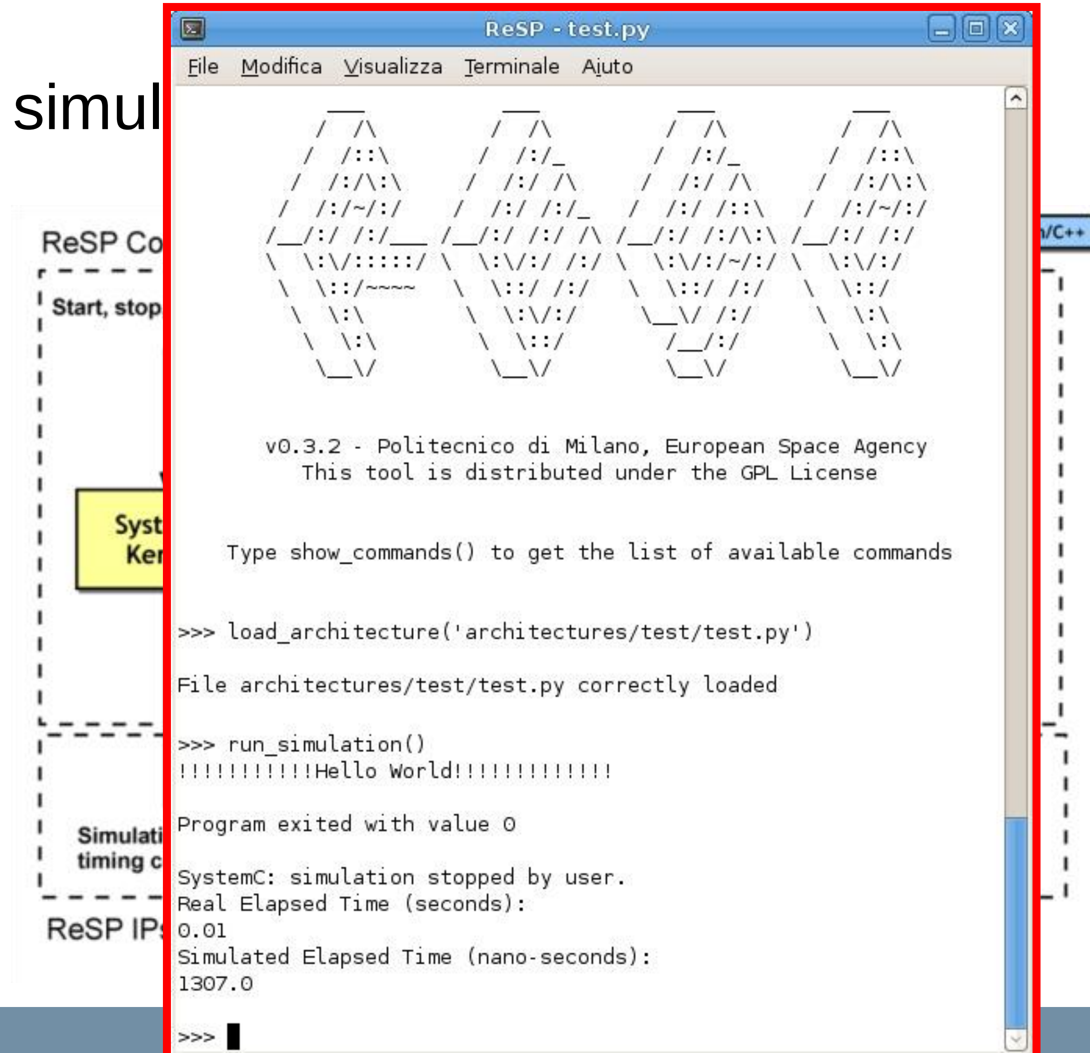
- HW components modeled in **SystemC/TLM**
- Features a functional model generator for microprocessors



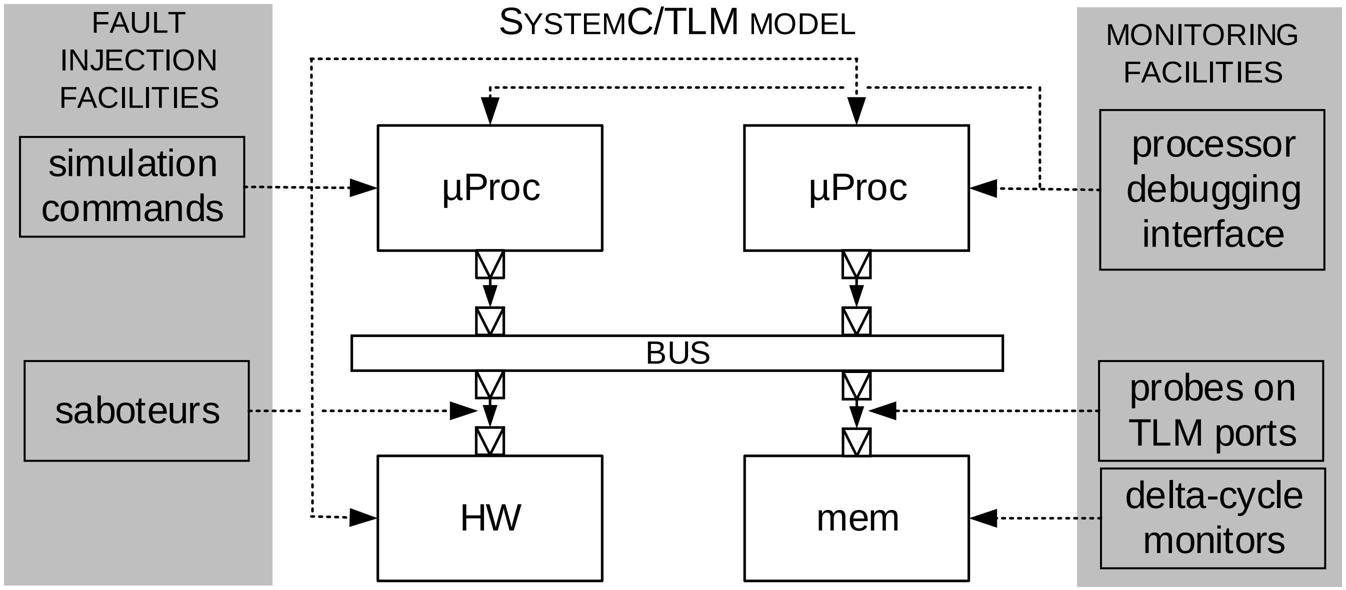
Background: ReSP

ReSP is a system-level simul

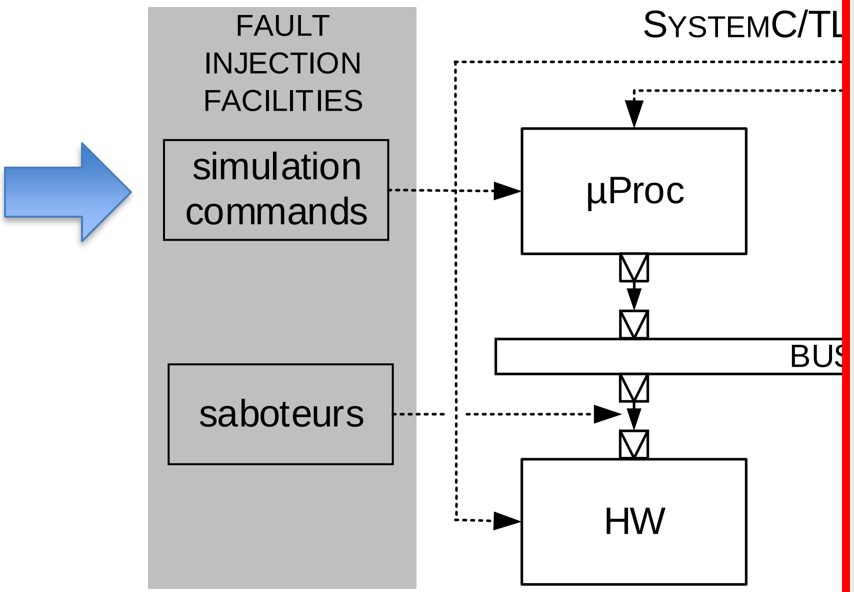
- **Python** offers introspection and scripting capabilities:
  - non-intrusive visibility into the components
  - Run-time composition and management of the specification



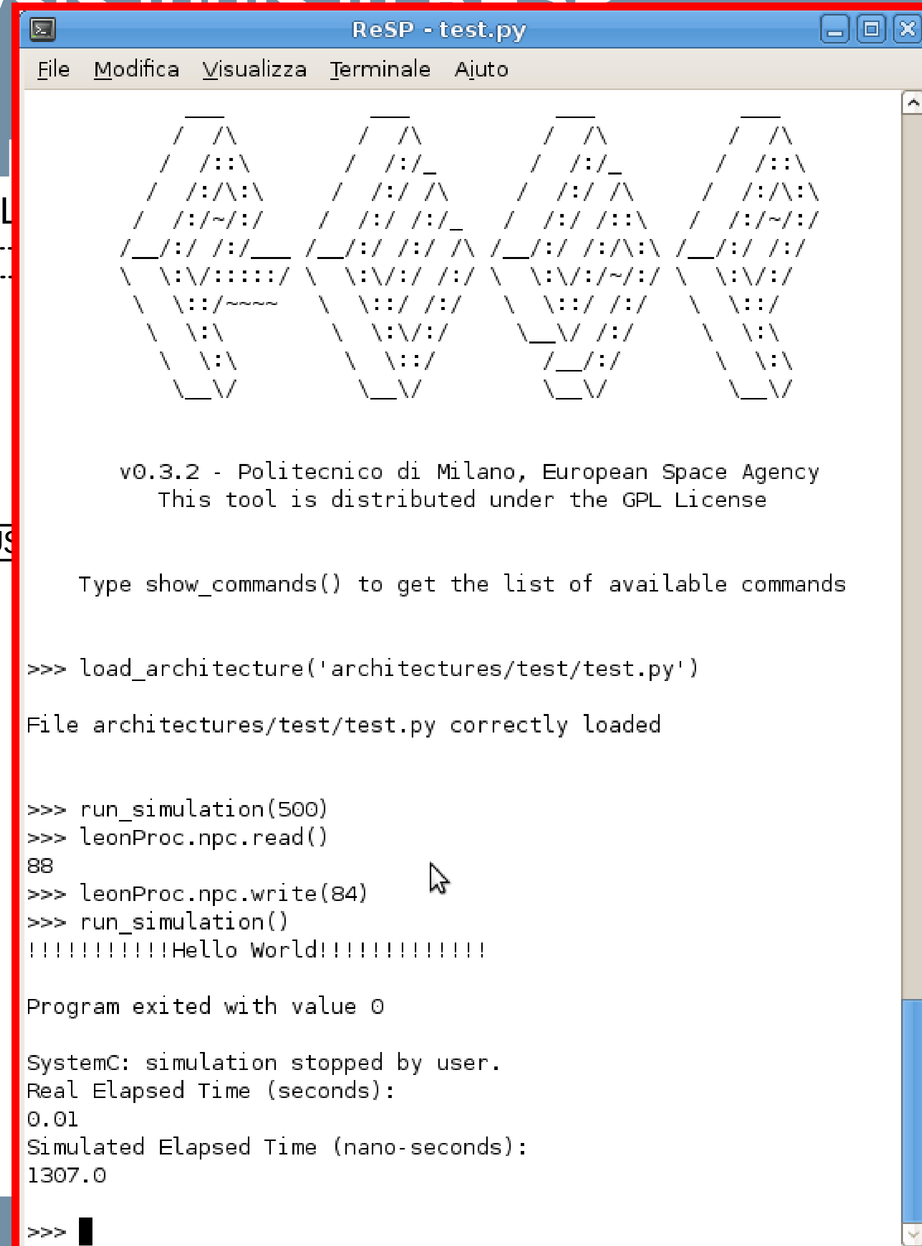
Fault injection and analysis tools in ReSP



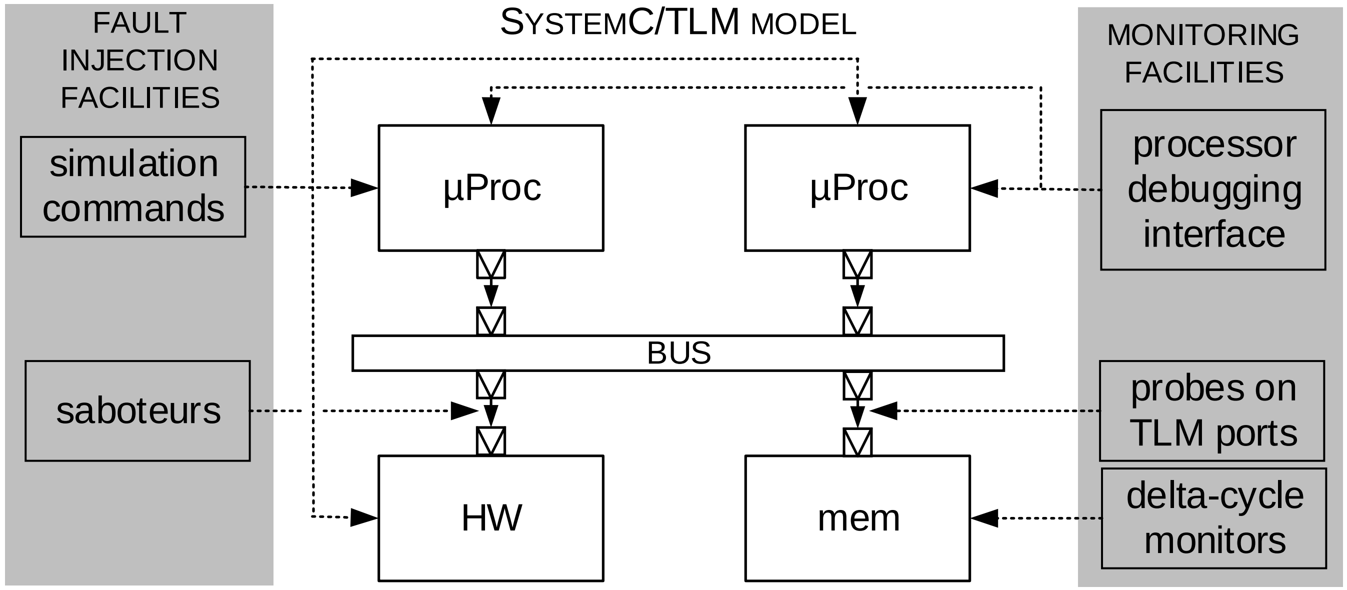
Fault injection and analysis tools in ReSP



- The console stops the simulation, injects the fault and resumes the simulation



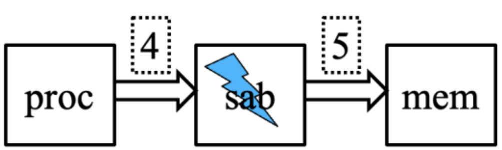
Fault injection and analysis tools in ReSP



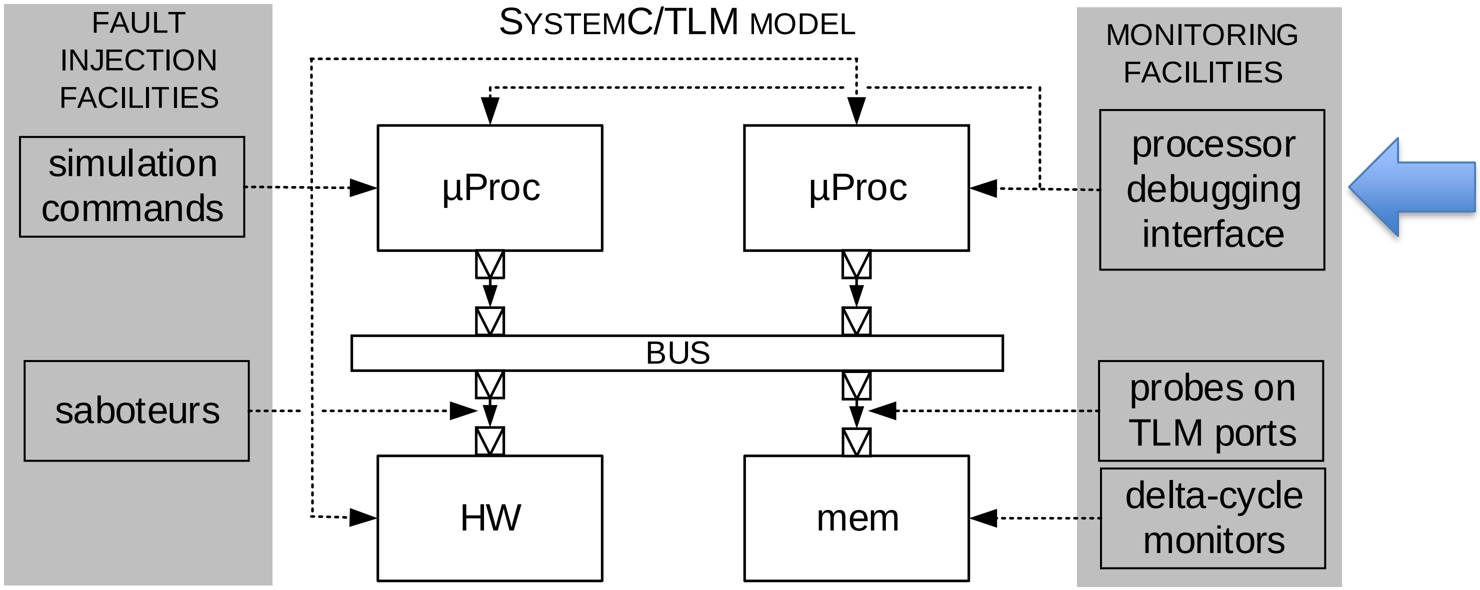
- Saboteurs can be automatically inserted in the specification

```
SC_MODULE(saboteur) {
  sc_in<int> in;
  sc_in<int> out;
  sc_in<int> fault_activate;

  void work();

  SC_CTOR(saboteur) {
    SC_METHOD(work);
    sensitive << in << fault_activate;
  }
```

```
void saboteur::work() {
  if ( fault_activate.read()==1 ){
    faulty_output = apply_mask(in.read());
    out.write(faulty_output);
  } else
    out.write(in.read());
  ...
}
```

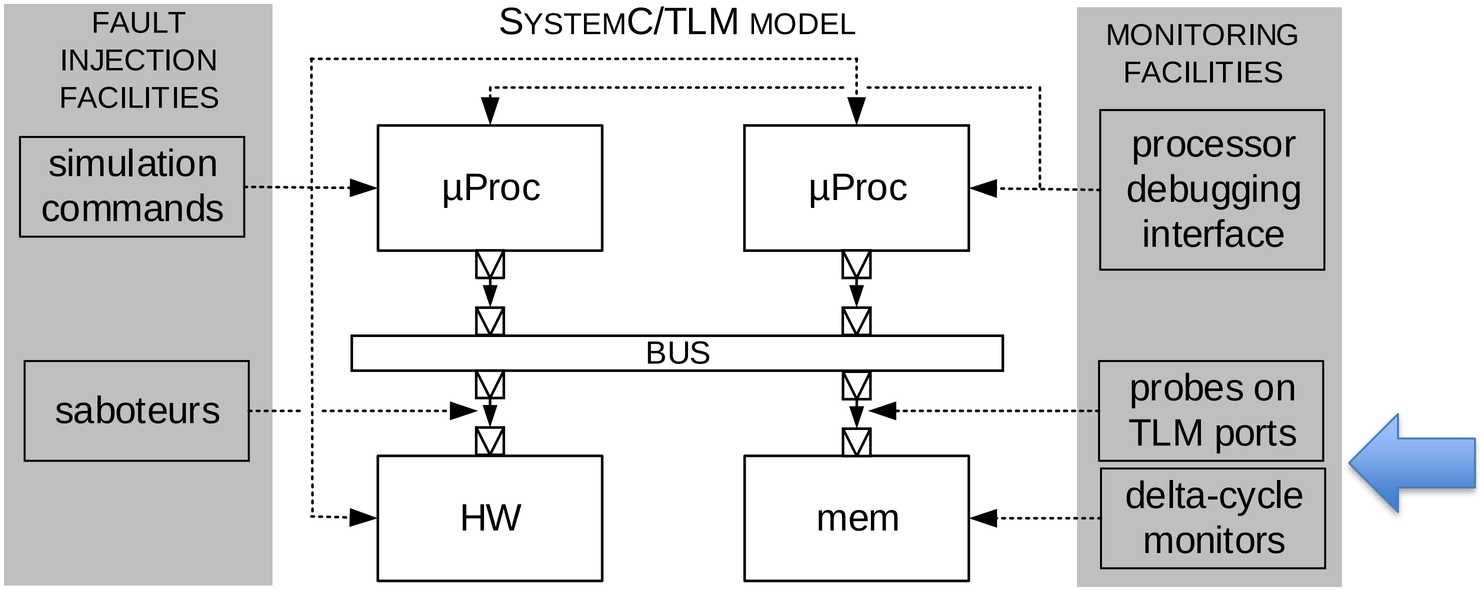


Fault injection and analysis tools in ReSP



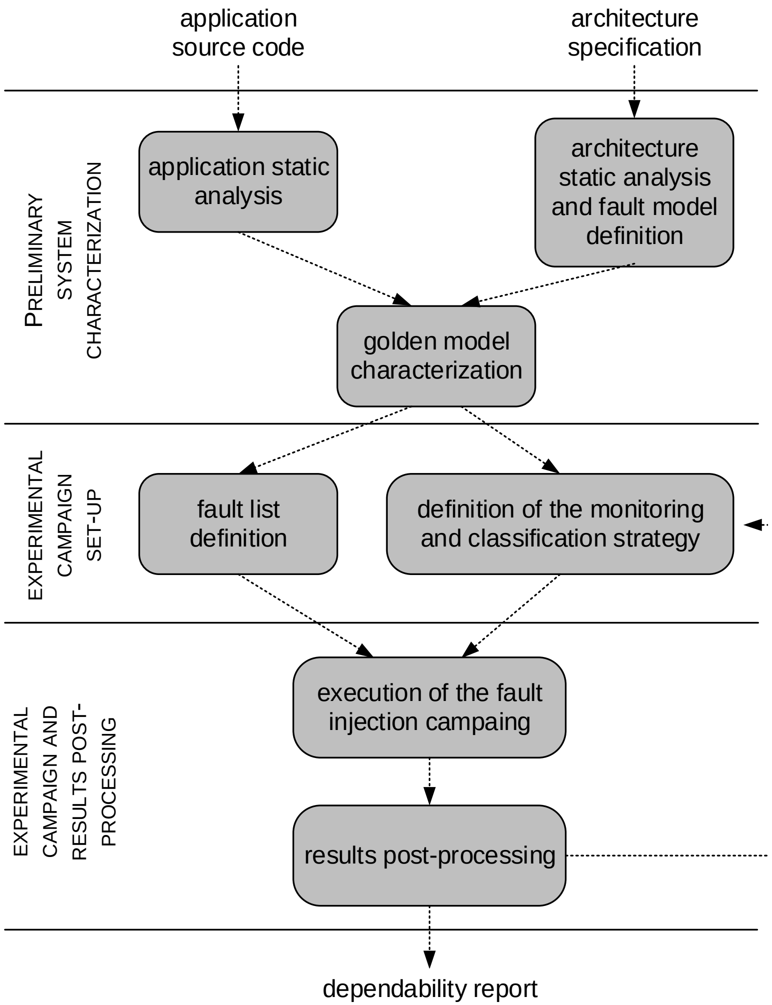
- Processor models expose a configurable debugging interface
- Custom C++/Python functions can analyze the execution
- Application Binary Interface (ABI) can be exploited to interpret raw data (in particular, on function calls/returns)

Fault injection and analysis tools in ReSP

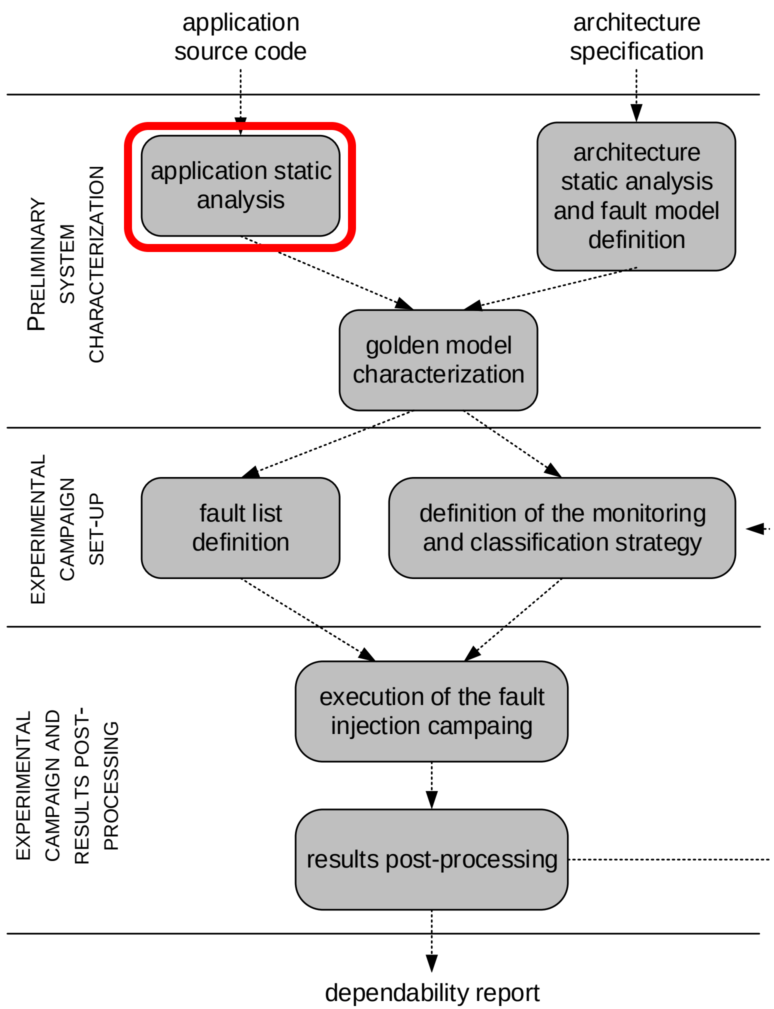


- Probes (similar to saboteurs) can be used to analyze transmitted data
- Custom Python functions can be used to monitor the internal status of components (called every scheduler delta cycle)

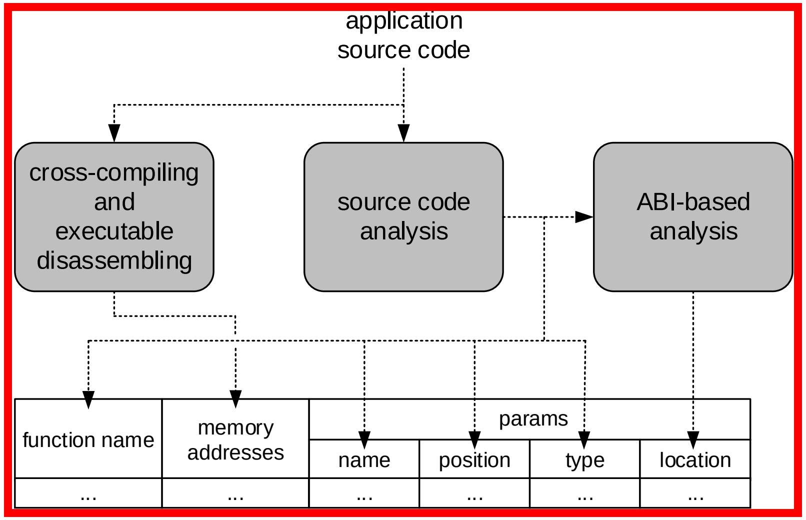
System-level reliability analysis methodology



System-level reliability analysis methodology



- A binary analysis extracts a static characterization of the application



System-level reliability analysis methodology

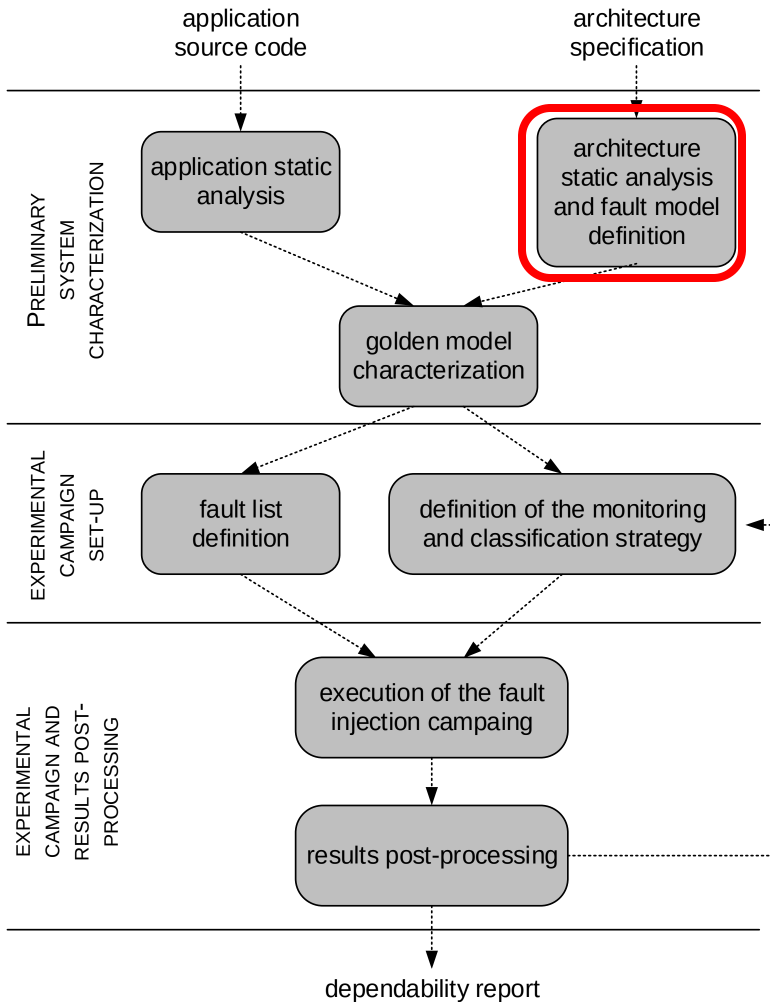
application source code

architecture specification

Function	Memory addresses	Parameters			
		Name	Position	Type	Location
main	0x1498 – 0x1640				
rgb2gray	0xea0 – 0xf64	inputImg	0	char* (dim = 17,280)	reg0
		outputImg	1	char* (dim = 5760)	reg1
		width	2	unsigned	reg2
		height	3	unsigned	reg3
edgeDetector	0xf68 – 0x12fc	inputImg	0	char* (dim = 5760)	reg0
		outputImg	1	char* (dim = 5760)	reg1
		width	2	unsigned	reg2
		height	3	unsigned	reg3
edgeOverlapping	0x1300 – 0x1480	inputImg	0	char* (dim = 17,280)	reg0
		edgeImg	1	char* (dim = 5760)	reg1
		outputImg	2	char* (dim = 17,280)	reg2
		width	3	unsigned	reg3
		height	4	unsigned	SP + 0x0
readBitmap	HW	inputImg	0	char* (dim = 17,280)	reg0
		width	1	unsigned	reg1
		height	2	unsigned	reg2
writeBitmap	HW	outputImg	0	char* (dim = 17,280)	reg0
		width	1	unsigned	reg1
		height	2	unsigned	reg2

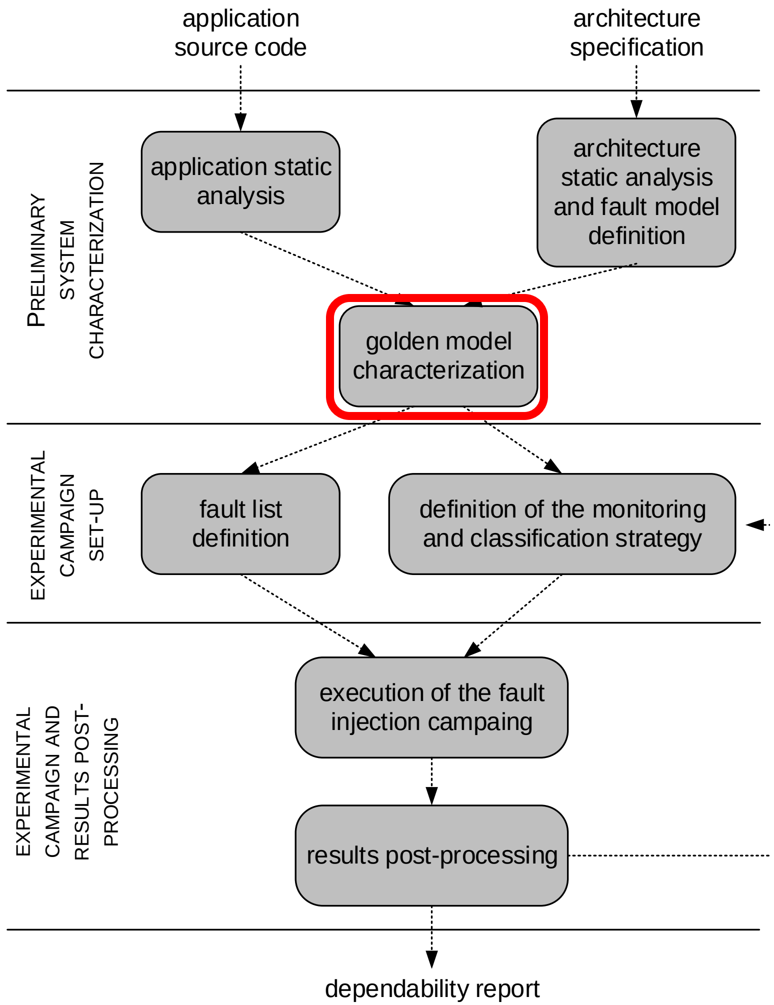
dependability report

System-level reliability analysis methodology



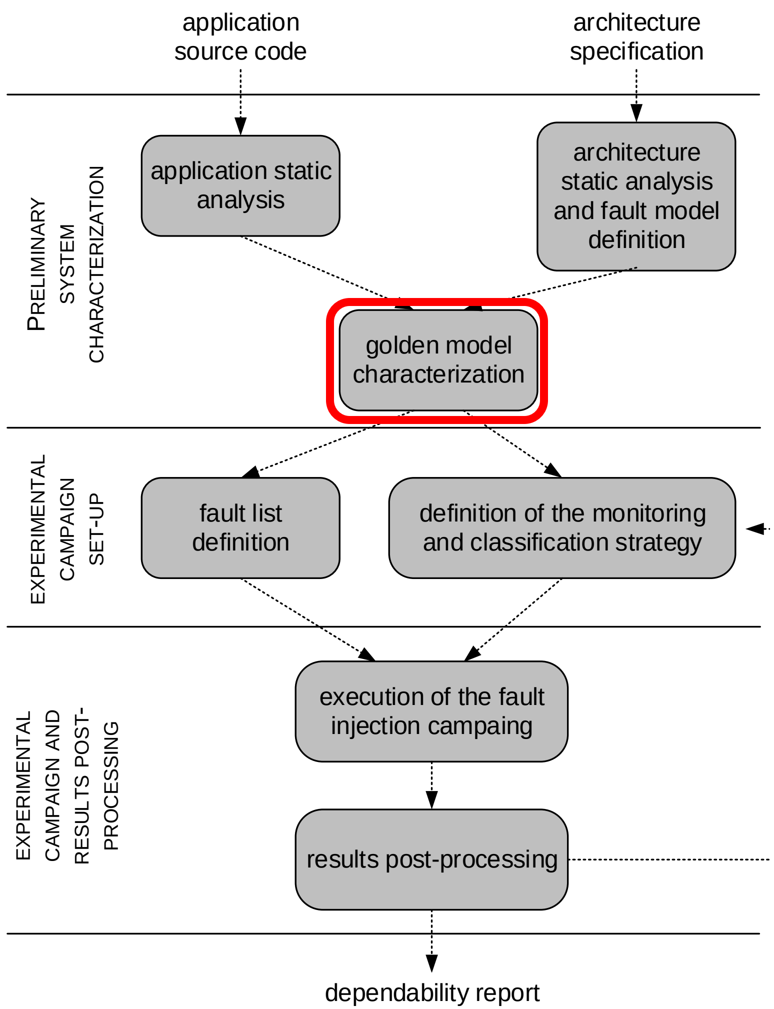
- A semi-automated analysis of components' SystemC specification
  - Identifies injection locations, and
  - Supports the definition of fault models

System-level reliability analysis methodology

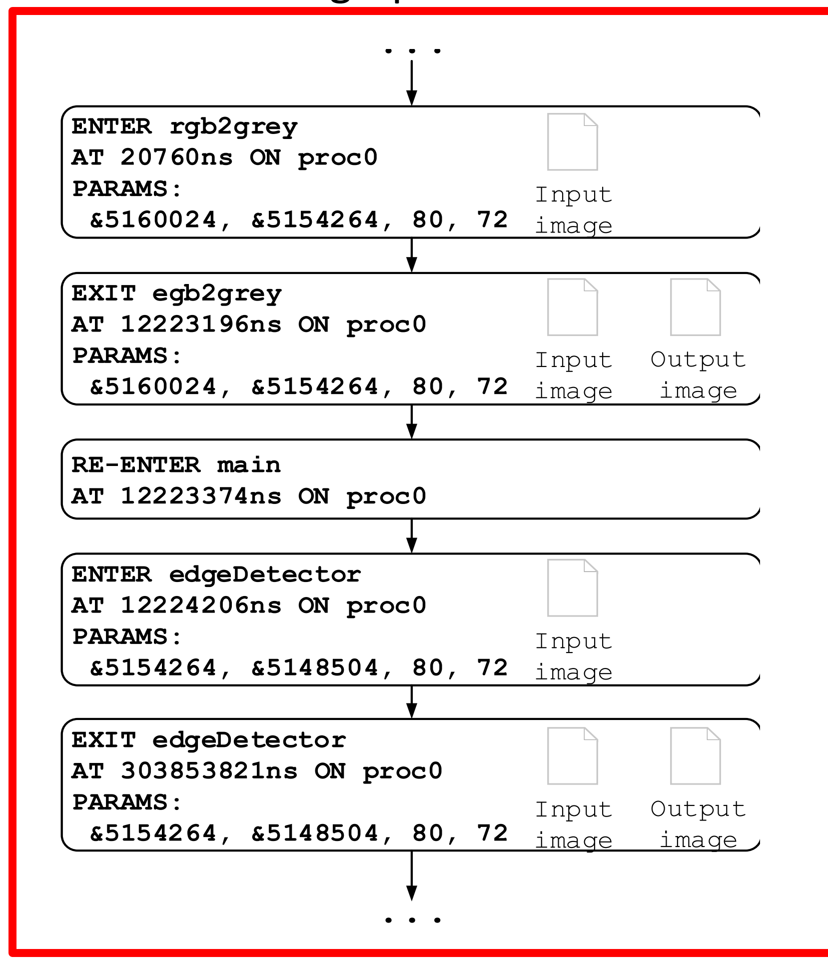


- A fault-free run is performed to characterize the golden execution
  - At architecture level:
  - Collect relevant traces
  - At application level: Execution flow graph

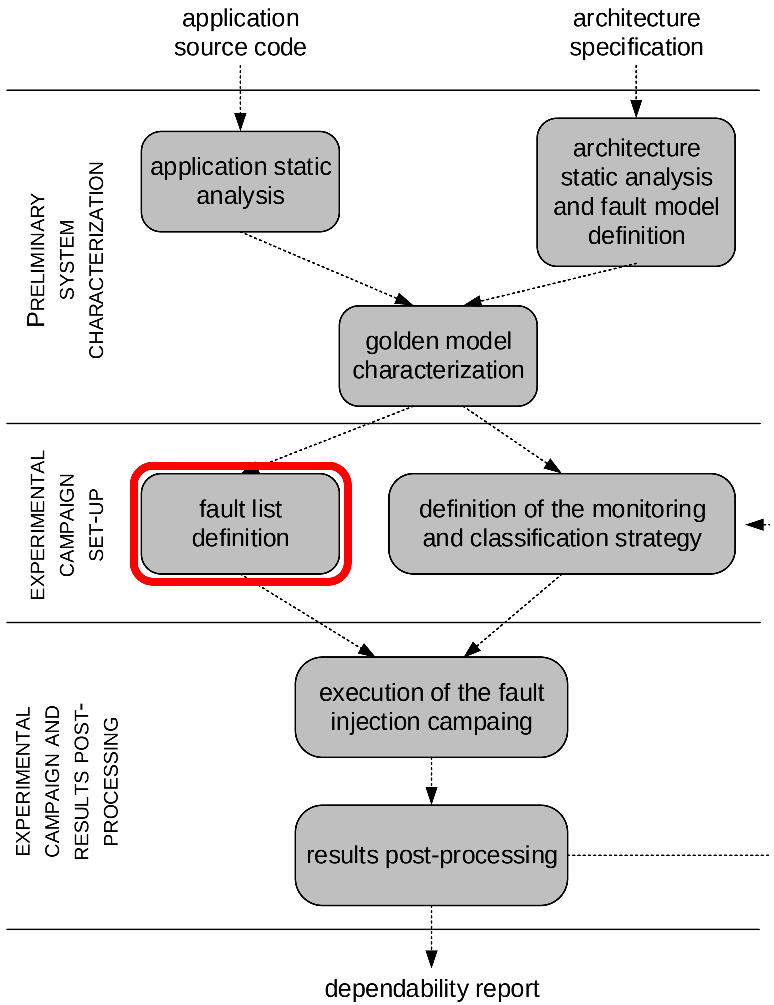
System-level reliability analysis methodology



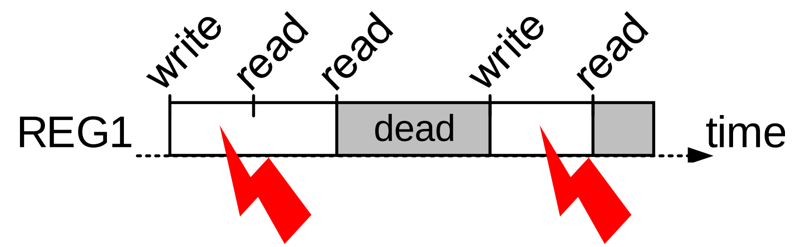
Execution flow graph



System-level reliability analysis methodology

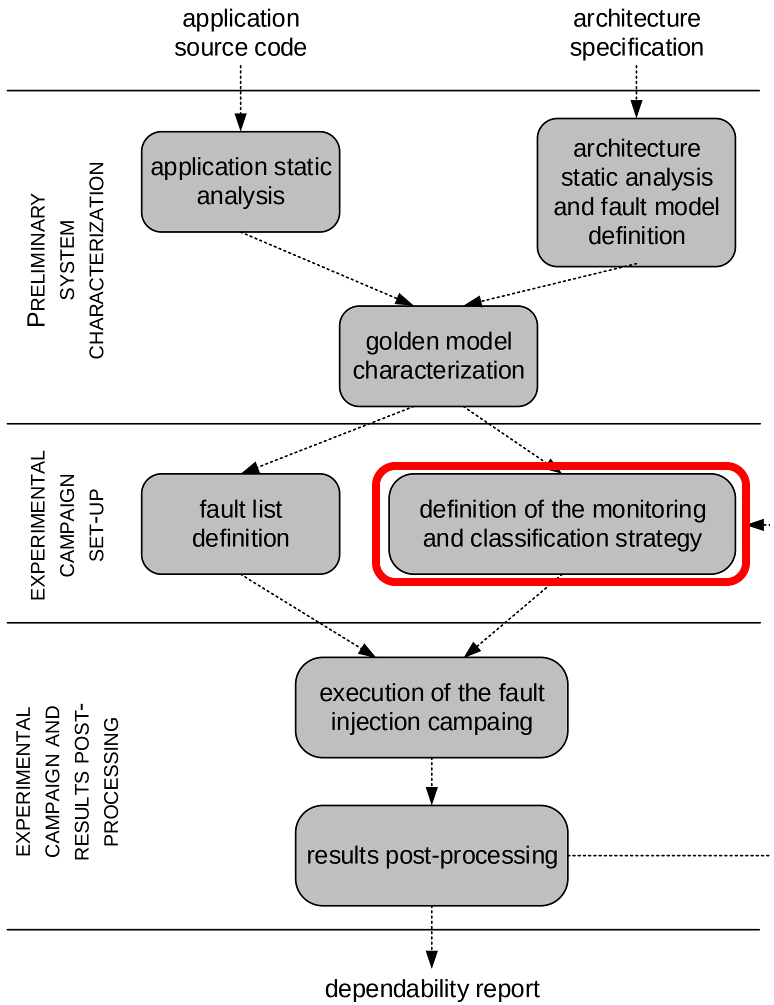


- Fault list is defined according to
  - The liveness analysis on raw traces

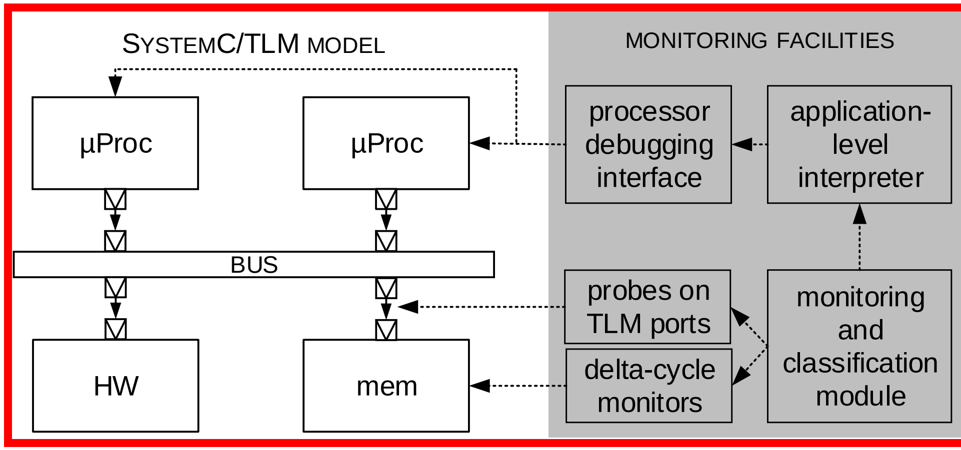


  - The function to corrupt

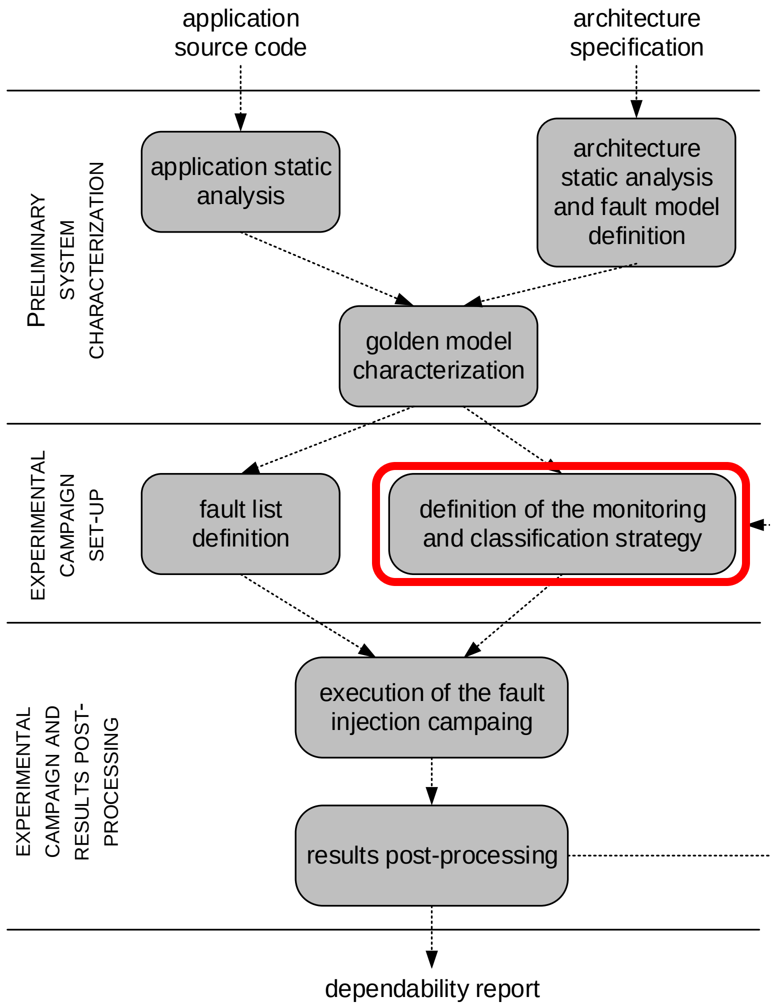
System-level reliability analysis methodology



- Definition of custom architecture/application-level monitoring and classification functionalities in C++/Python

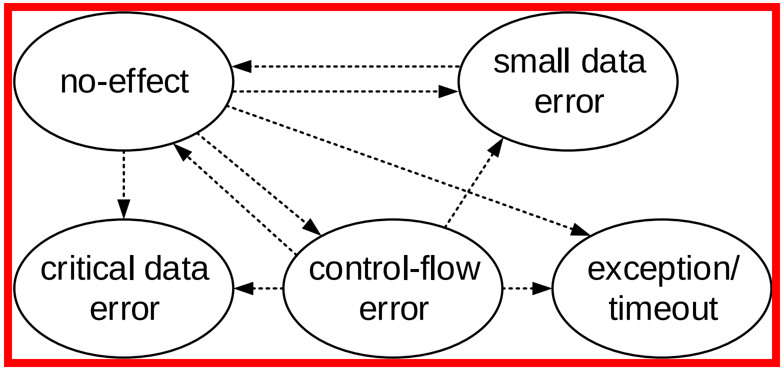


System-level reliability analysis methodology

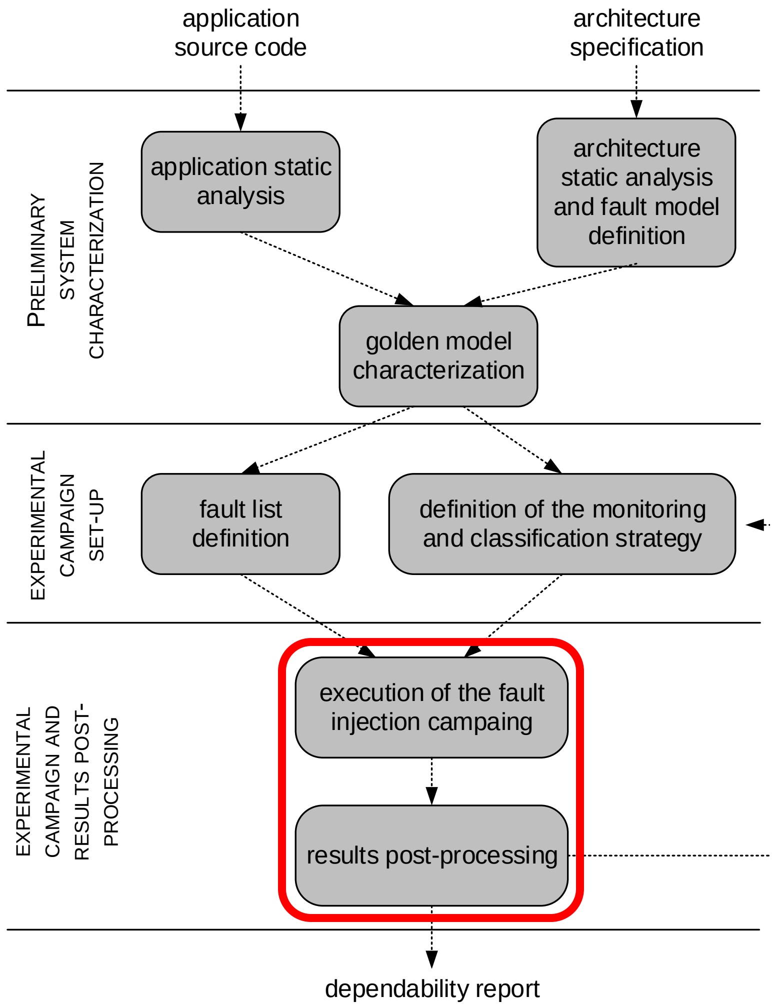


- Definition of custom architecture/application-level monitoring and classification functionalities in C++/Python

- Example of classifier for the edge detection application

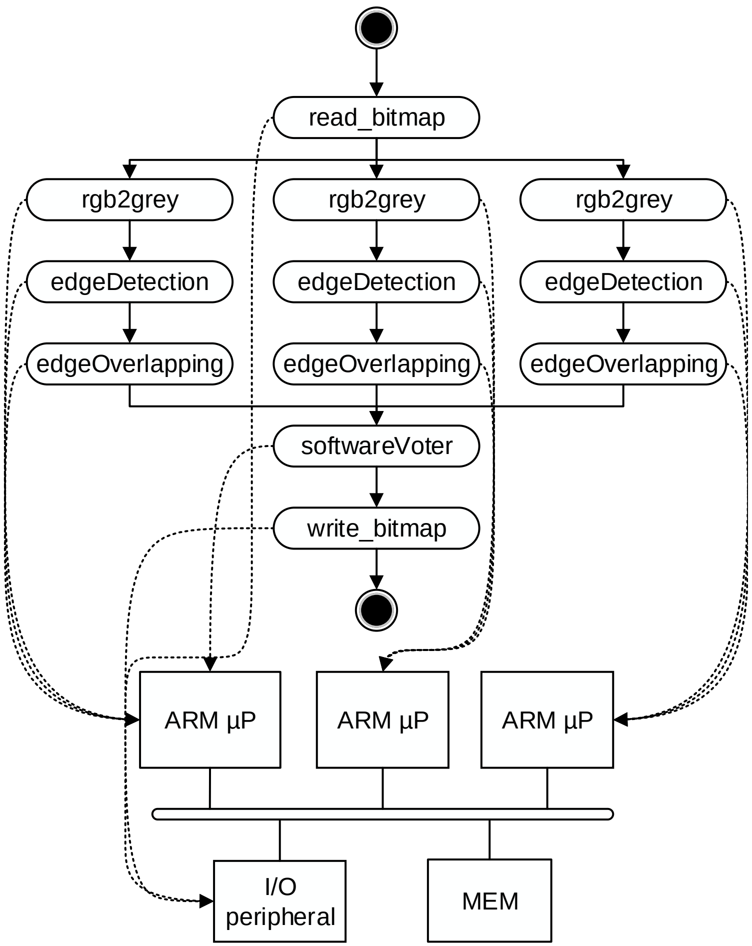


System-level reliability analysis methodology



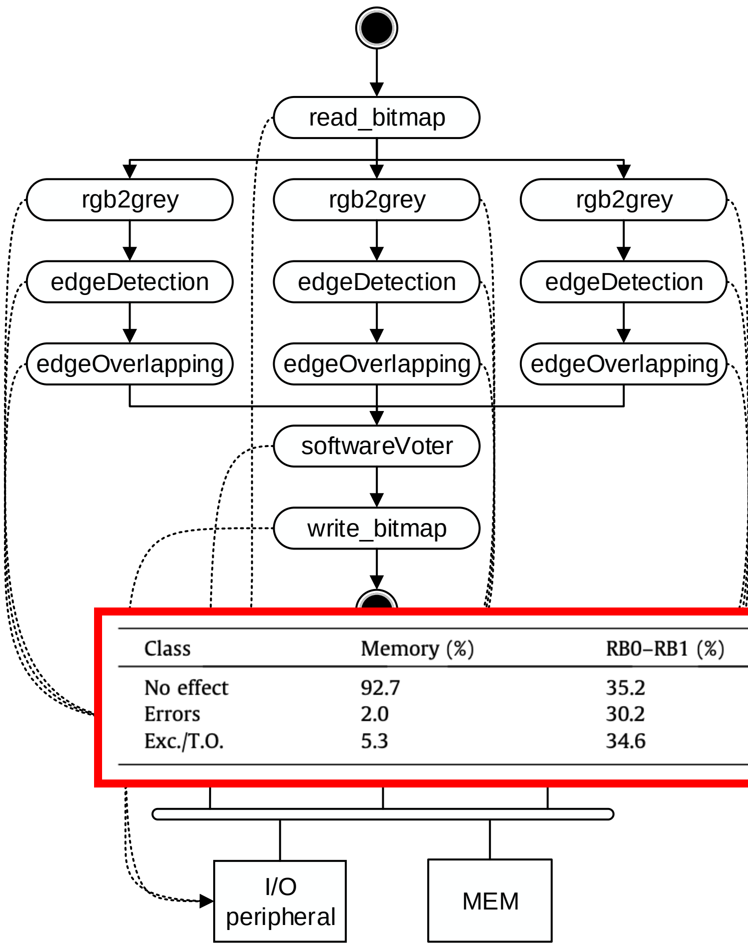
- Pretty standard execution flow of the fault injection campaign
- "Reacher" results allow more in-depth analysis
- Analysis results may provide a feedback for a refinement of the classification strategy

A case study



- Reliability analysis of a thread-level TMRed edge detector running onto a multicore

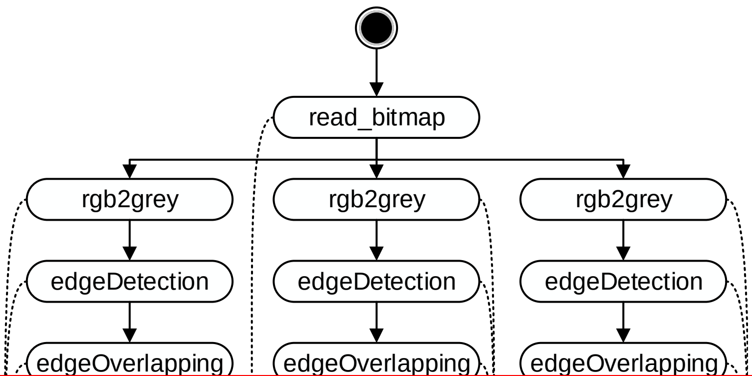
A case study



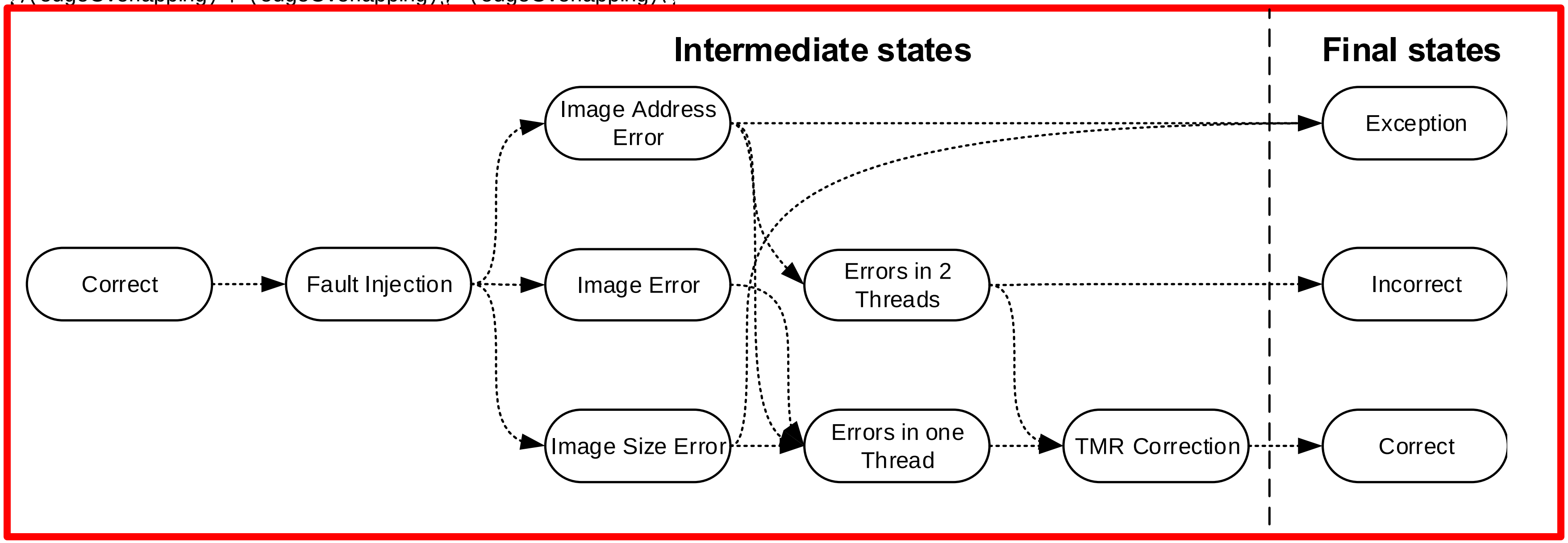
Class	Memory (%)	RB0–RB1 (%)	RB2–RB3 (%)	Link Reg. (%)	SP Reg. (%)	PC Reg. (%)
No effect	92.7	35.2	79.9	22.0	30.6	22.5
Errors	2.0	30.2	7.7	0	1.4	2
Exc./T.O.	5.3	34.6	12.4	78	68	75.5

- First analysis performed on the main results
- Classification:
  - No effect
  - Errors
  - Exception/timeout

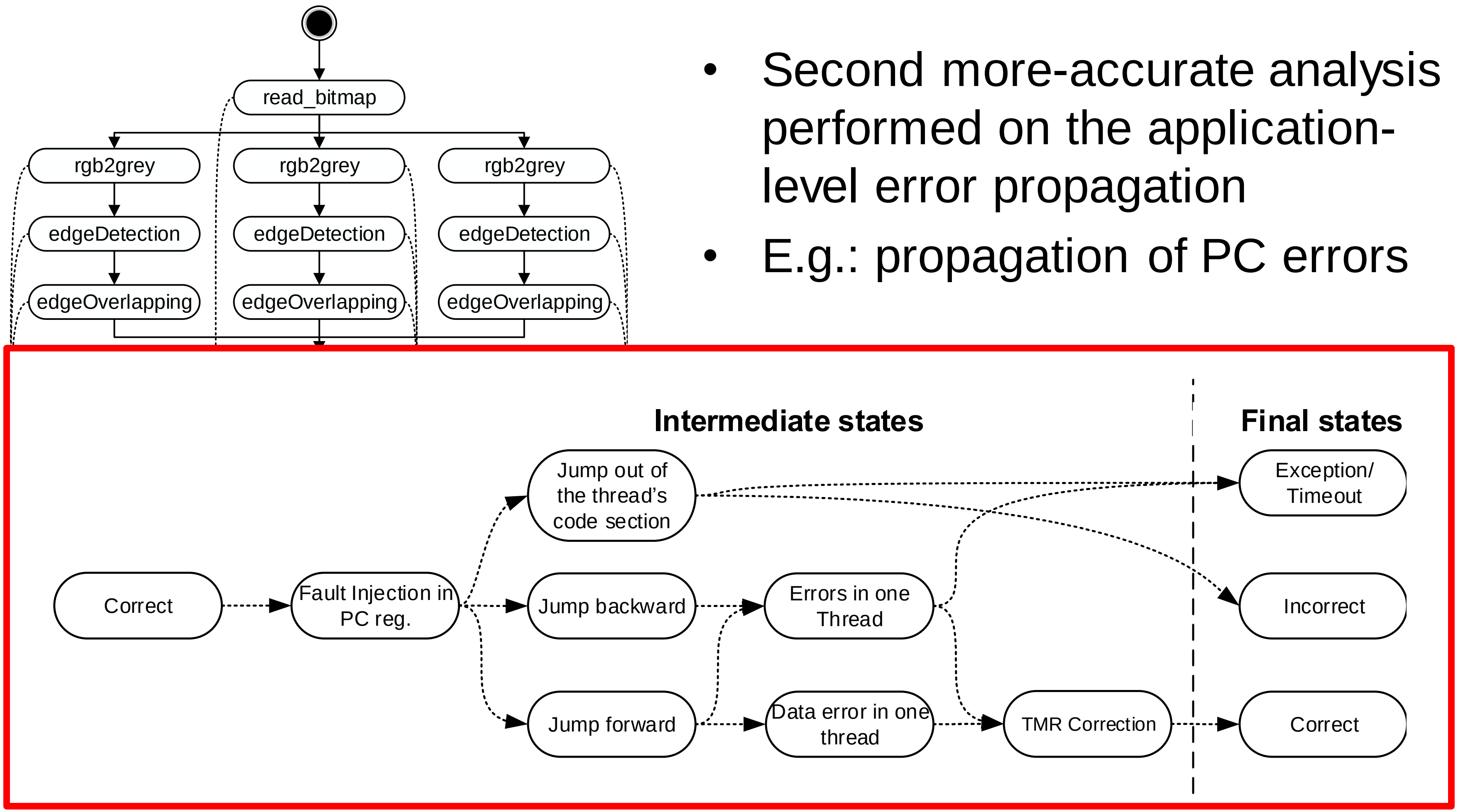
A case study



- Second more-accurate analysis performed on the application-level error propagation
- E.g.: propagation of data errors



A case study



- Second more-accurate analysis performed on the application-level error propagation
- E.g.: propagation of PC errors

Conclusions

The various aspects of the methodology have been presented in three scientific papers:

- A. Miele: **A fault-injection methodology for the system-level dependability analysis of multiprocessor embedded systems**. In Journal of Microprocessors and Microsystems, Elsevier, August 2014
- G. Beltrame, C. Bolchini, A. Miele: **Multi-level Fault Modeling for Transaction-level Specifications**. In Proc. of (GLSVLSI), 2009
- C. Bolchini, A. Miele, D. Sciuto: **Fault Models and Injection Strategies in SystemC Specifications**. In Proc. of IEEE Euromicro DSD, 2008

Thank you…

… questions?

Contact: antonio.miele@polimi.it