DCML-CPS - Module 3

Testing Mechanisms

Tommaso Zoppi
University of Trento - Povo (IT)
tommaso.zoppi@unifi.it
tommaso.zoppi@unitn.it

Course Map

1. Basics and Metrology
2. Monitoring

Monitoring

Testing

3. Fault Injection
4. Robustness Testing

5. Data Analysis
6. Supervised ML
7. Unsupervised ML

Anomaly Detection

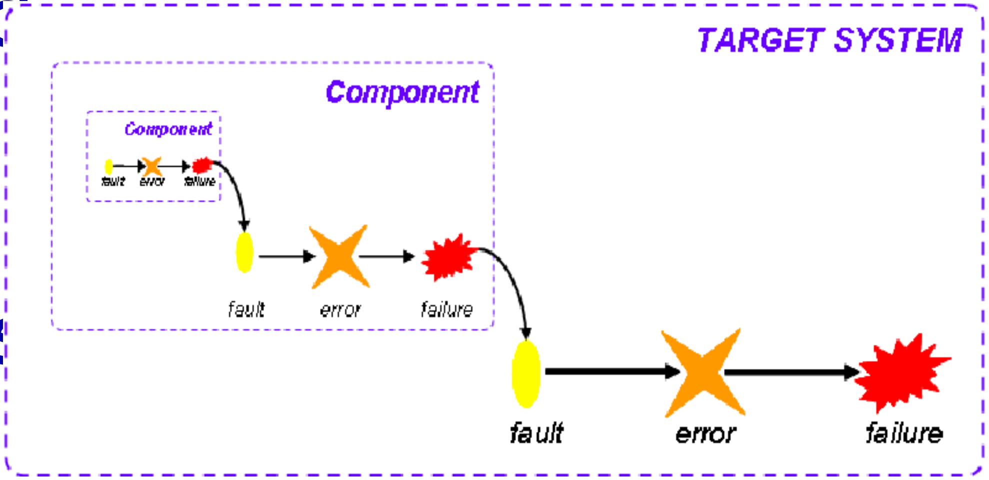
8. Meta-Learning
9. Error/Intrusion Detection

Tools & Libs

Deep Learning

Remember …

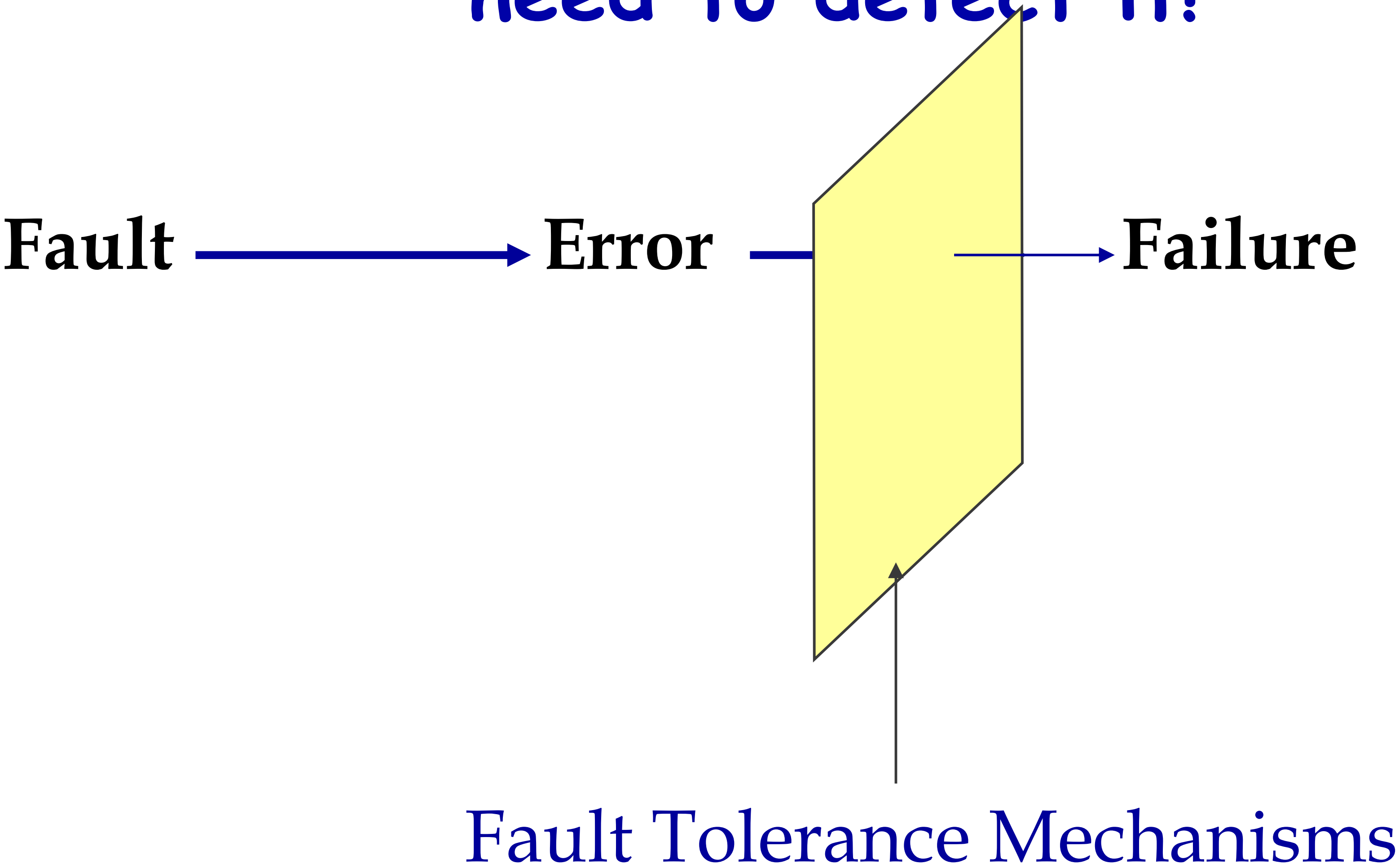
- An **error** is the part of the system state that may cause a subsequent failure
- A **failure** occurs when an error reaches the service interface and alters the service.
- A **fault** is the adjudged or hypothesized cause of an error. A fault is active when it produces an error; otherwise it is dormant.



Avizienis, Algirdas, et al. "Basic concepts and taxonomy of dependable and secure computing." *IEEE transactions on dependable and secure computing* 1.1 (2004): 11-33.

Remember: Faults, errors and failures

- Nice but… To tolerate a fault we first need to detect it!



Monitoring for what?

- We discussed how to monitor a system

- What is the aim of monitoring?

UNIVERSITÀ DEGLI STUDI FIRENZE
DIMAI DIPARTIMENTO DI MATEMATICA E INFORMATICA "ULISSE DINI"

Monitoring for what?

- We discussed how to monitor a system

- What is the aim of monitoring?
  - Observing key indicators of the system through probes
  - Collecting and storing data for further analyses
- OK, but, how can we discover the undesired behaviours of the system?
  - And, more importantly, how can we understand if our monitoring system is collecting useful data?

RCL RESILIENT COMPUTING LAB

Monitoring for what?

- What is the aim of monitoring?
  - Observing key indicators of the system through probes
  - Collecting and storing data for further analyses
- OK, but, how can we discover the undesired behaviours of the system?
  - And, more importantly, how can we understand if our monitoring system is collecting useful data?
    - Observing system when faults activate
    - Understanding if the measures we monitor fluctuate due to the fault

Waiting for Faults to Happen

- So… we just need to wait for faults to manifest.
- Easy, but…
  - You have to detect them
  - It may take a looooooong time
- Possible ideas?

Waiting for Faults to Happen

- So… we just need to wait for faults to manifestate.
- Easy, but…
  - You have to detect them
  - It may take a looooooong time
- Possible ideas?
  - List all the possible (KNOWN) faults
  - Artificially injecting them
    - Observe the reaction of the system

Basics of Fault Injection

Monitoring and Fault/Error Detection

- Which faults may impact our system?

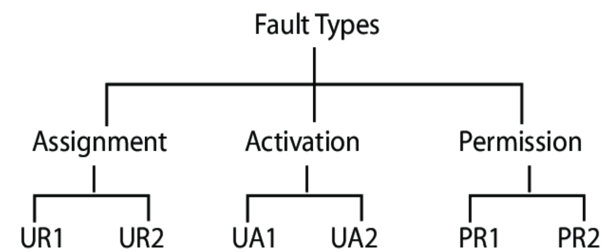
- How to detect faults?

- How to understand if our detectors are effective?

Monitoring and Fault/Error Detection

- Which faults may impact our system?

Studying the system to derive the main hazards **(fault model)**



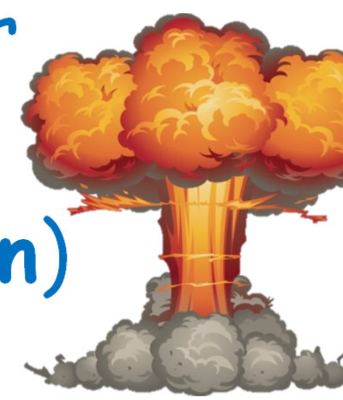
- How to detect faults?

Monitoring! At least we know how to do this!

- How to understand if our detectors are effective?

Wait until the fault manifests to evaluate our countermeasures, or

Artificially generate the fault (**fault injection**)



Fault Injection

- **Fault Injection** is an approach for dependability analysis complementary to the model-based analysis.
  - It is the deliberate introduction of faults within a system, to analyze its behavior in the presence of faults

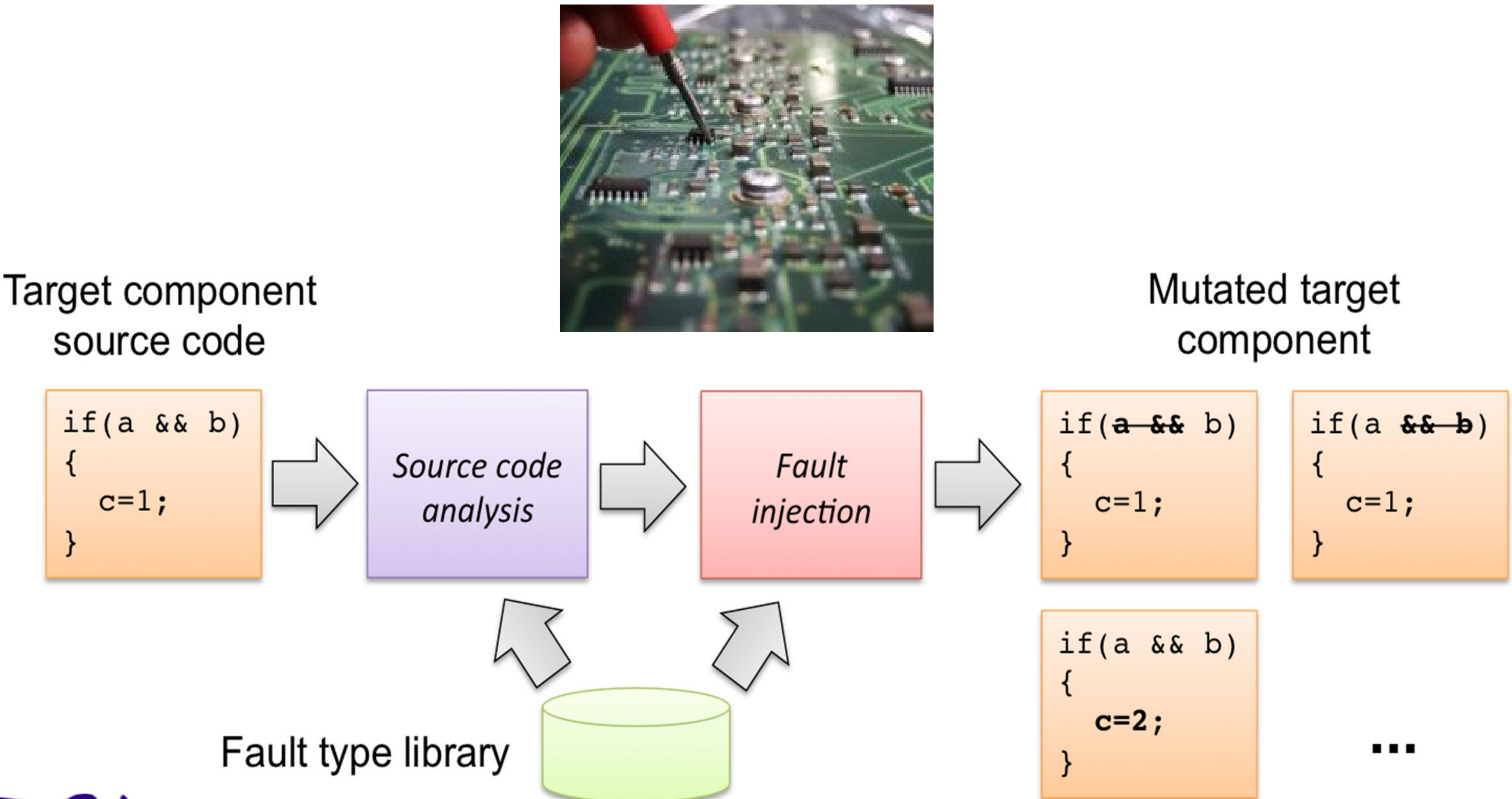
- To perform Fault Injection we need to specify a **Fault Model**:
  - **Types** and **Frequency** of the faults that could affect the correct execution of the system,
    - **Only realistic faults provide realistic measures**

Fault Injection

- Guess how to do it?

Fault Injection

- Guess how to do it?

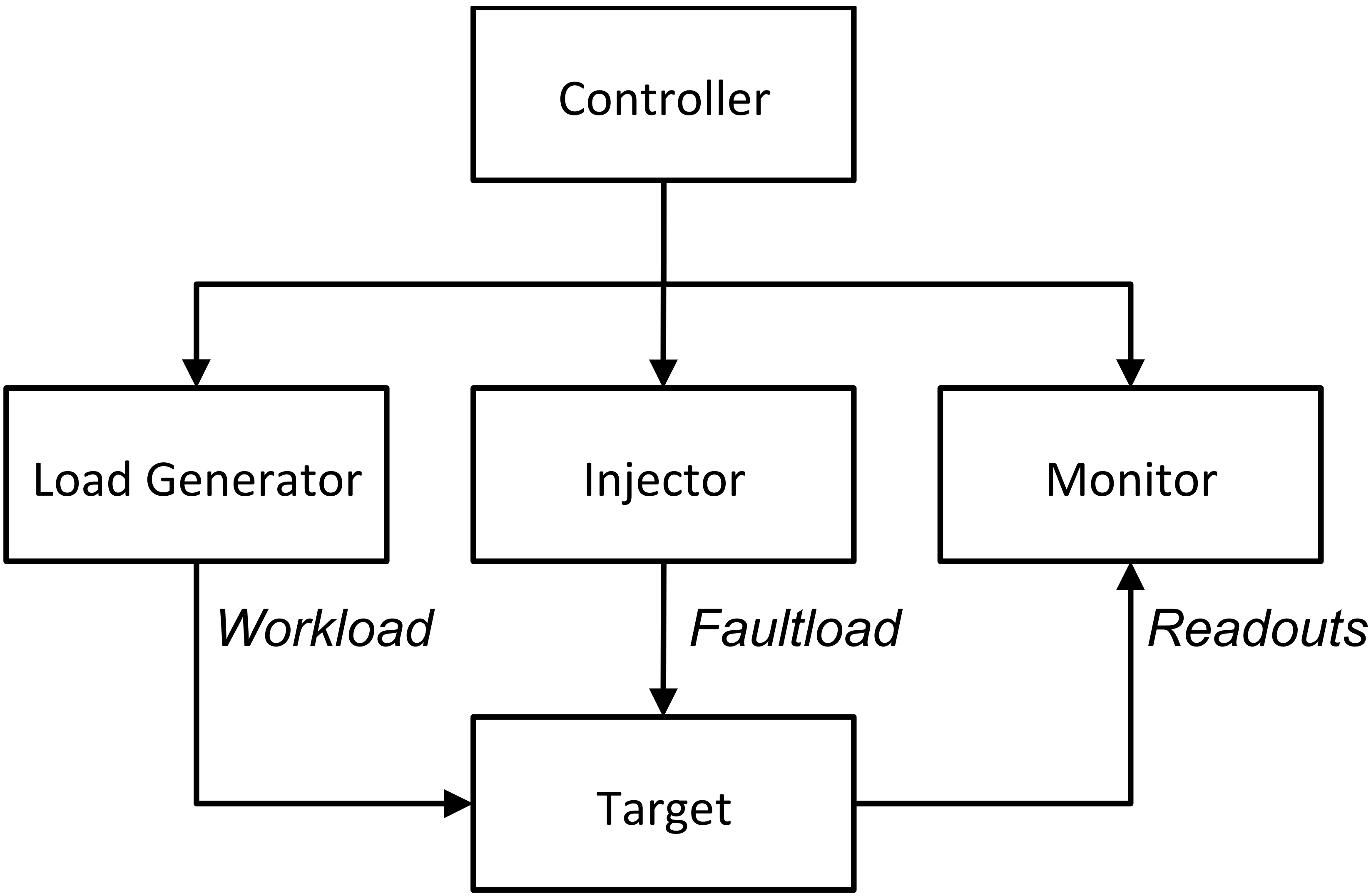


Phases

- A fault is **activated** when it causes an error
- An error can produce more errors (**propagation**), which could finally lead to a failure

- <u>The fault injection does not necessarily lead to a failure</u>
  - in fault-tolerant systems, an error can be detected and corrected by specific mechanisms
  - an incorrect value can be overwritten, masking the error
  - the fault may not be activated, or the error can propagate within the duration of the experiment

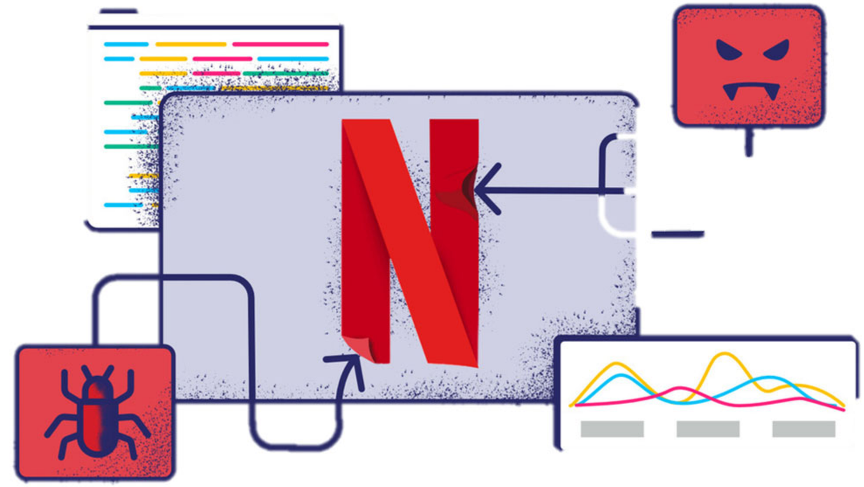
Fault Injection Experiment

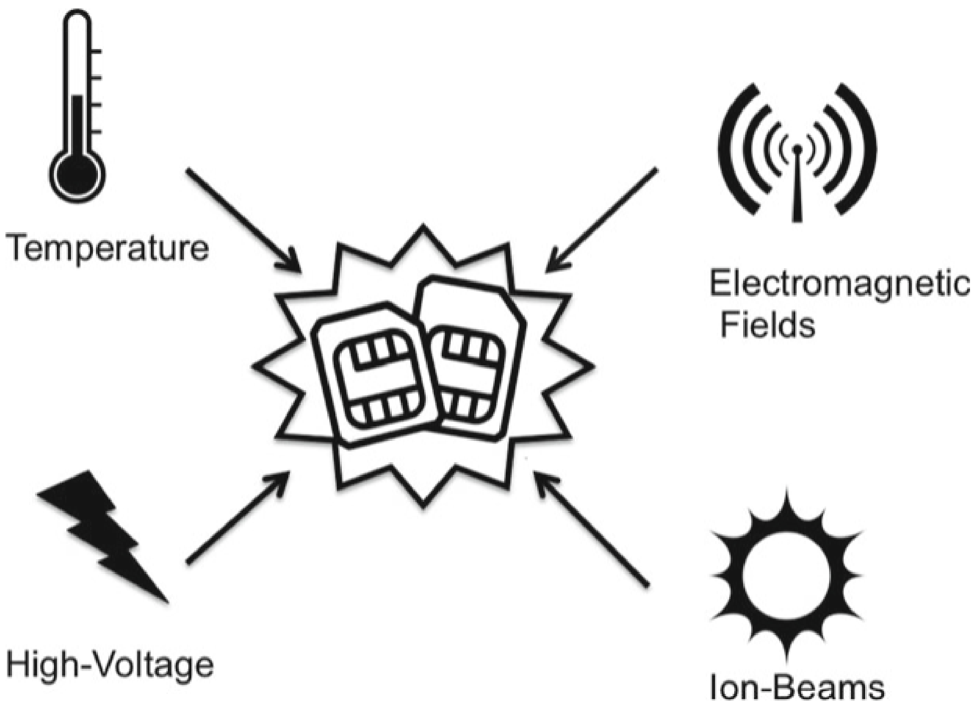


Fault Injection Components

- Target System: The system to be analyzed
- Workload Generator: provides the system with the inputs to be processed during the fault injection experiment (**Workload**)
- Injector: introduces a fault into the system during an experiment, altering the system structure or the system state
- Monitor: collects raw data from the system (using **Probes**) to process measures
- Controller: coordinates the operation of the other components and iterate the experiments (**Experimental Campaign**)
- Faultload: it is the set of faults - taken from the **Fault Model** - injected in a campaign

Fault Injection Made Right



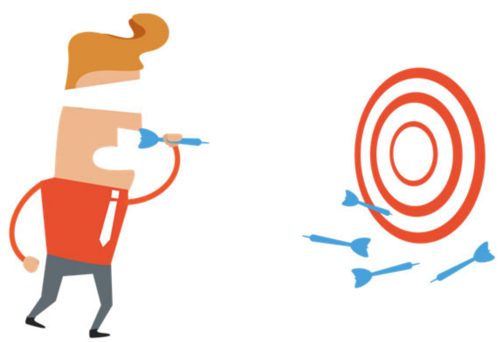


Fault Injection Made …

On the Analysis of the Failure Modes

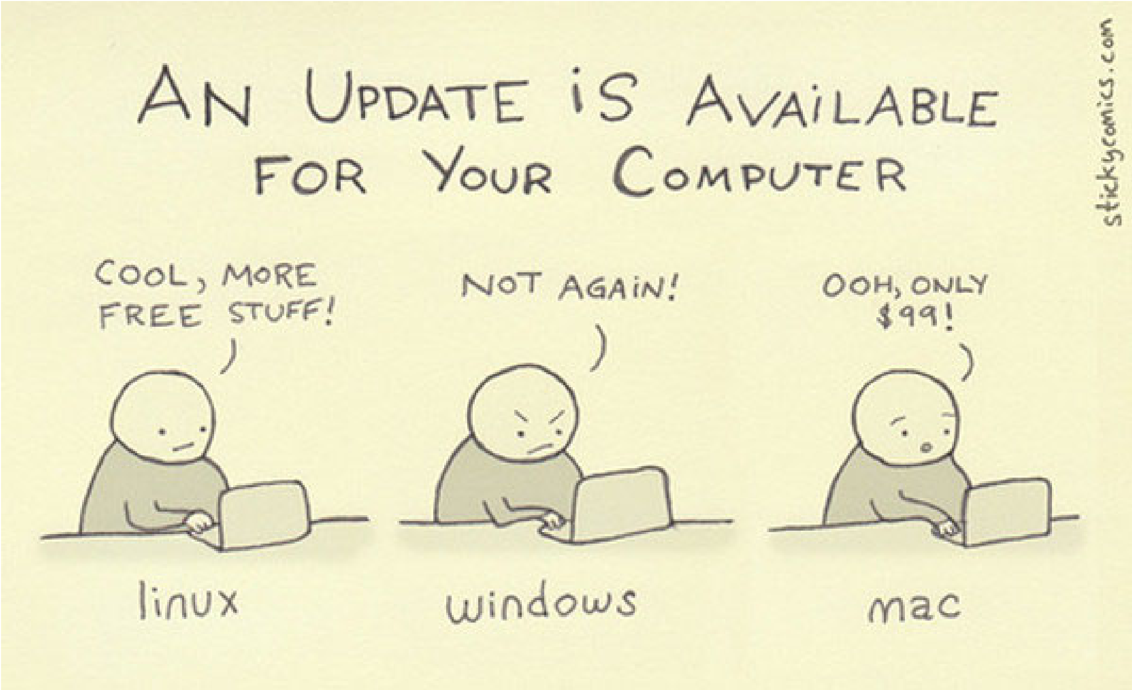
Analysis of the failure modes

- The Fault Injection allows
  - To identify what are the possible effects of the faults on the system's behavior (failure modes)
  - To predict possible safety-critical component failures
  - To suggest possible system countermeasures to avoid/mitigate safety-critical failures (e.g., introducing fault tolerant mechanisms)



Analysis of the failure modes

- Example: comparison of the failure modes of different Microsoft OSs, considering the **software faults** on the **device driver**
  - The driver represents the major cause of OS failures



Failure Modes of OSs

- The failure modes of the OSs can be classified considering 3 different points of view:
  - **Availability**: the most critical failure modes cause the unavailability of the system (or of a system function)
  - **Feedback**: the most critical failure modes provide less info on the system state to the user
  - **Stability**: the most critical failure modes are those for which the system is working but in an incorrect way

Sorting the Failure Modes by Severity

Availability	Feedback	Stability
A1 La macchina è inutilizzabile (non disponibile)	F1 Perdita di dati senza alcun avviso	S1 Perdita di dati senza alcun avviso
A2 La macchina è utilizzabile se non si prova a utilizzare il driver guasto	F2 Il sistema fallisce senza dare alcuna informazione sul malfunzionamento	S2 Il sistema sembra funzionare ma non lo è, e può esserci perdita di dati
A3 I sottosistemi che interagiscono col driver guasto diventano indisponibili	F3 Il driver guasto non è subito identificato dal sistema, ma il fallimento del sistema o dell'applicazione potrebbe permettere di identificare il problema	S3 Come S2, ma con meno effetti collaterali (il sistema è influenzato solo in parte dal guasto)
A4 La macchina è utilizzabile eccetto che per il driver guasto	F4 Sebbene l'utente riceva delle informazioni, potrebbe non riuscire a identificare il driver guasto	S4 Il sistema si rifiuta di eseguire, prevenendo ulteriori conseguenze
A5 L'intero sistema è disponibile	F5 Il driver guasto è identificato al primo utilizzo	S5 Il sistema evita l'utilizzo delle parti malfunzionanti
	F6 Il driver guasto non ha effetto sul SO oppure è identificato, permettendo rapide misure correttive	S6 Il sistema si comporta normalmente

Fault Model

Fault Model

- Fault Injection is a general methodology that has to be specialized for a particular system, considering the types of faults to be reproduced and the properties to be measured

- The characterization of the faults to be injected (in terms of **types** and **frequency**) is called ***Fault Model***

- The Fault Model of a system considers
  - the system requirements,
  - the environment in which it will operate, and
  - the technologies with which the system is realized

Approaches to Define a Fault Model

- The definition of a new fault model is required when there is not an adequate model for the system
- The two main approaches to define a fault model are:
  - **Field Failure Data Analysis (FFDA)**: analysis of failures from systems already in operation, to trace fault types and their frequency; It may take a long time, due to the low frequency of failures
  - **Failure Modes and Effects Analysis (FMEA)**: system analysis, decomposing the system into elementary parts and identifying
    - (i) possible failure modes and
    - (ii) possible causes, effects, countermeasures for each mode

    For each element, it assumes all the possible fault scenarios

Fault Model examples

- The first fault models concerned the **circuits**
  - the effects of wear, manufacturing defects and external electromagnetic interferences
  - failures can cause both permanent effects (for example, **stuck-at** of a logic port) and transient effects (for example, the **flip** of a memory bit)
- Example: a fault model for VLSI circuits is required to check coverage of the error detection mechanisms

Examples of Chip Defects

- ▶ Processing faults
  - Missing contact
  - Parasitic transistors
  - Oxide breakdown

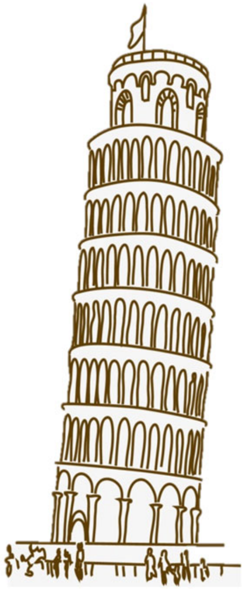
- ▶ Material defects
  - Bulk defects (cracks, crystal imperfections)
  - Surface impurities (ion migration)

- ▶ Time-dependent failures
  - Dielectric breakdown
  - Electro migration

- ▶ **Packaging failures**
  - **Contact degradation**
  - **Seal leaks**

Fault Model examples (2)

- More recent fault models include
  - software faults (e.g., programming defects)
  - faults due to human operators (e.g., configuration problems)
  - security vulnerabilities (which can lead to unauthorized access)



How to define realistic fault models?

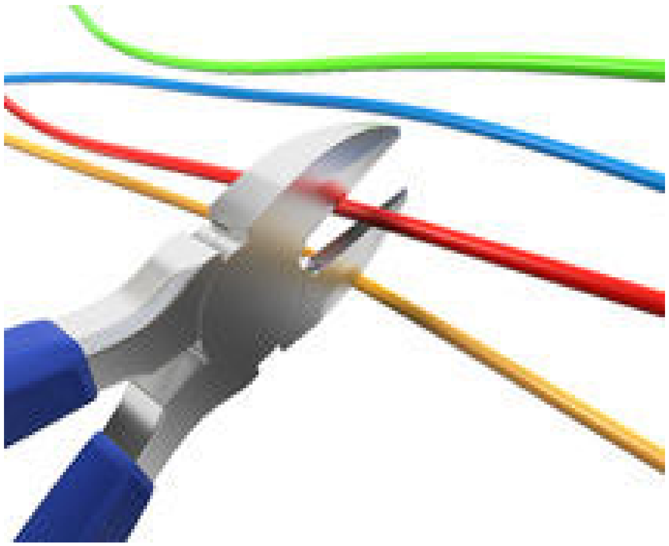
- Fault injection may aim to reproduce the effects of programming defects (bugs) in the software of a system
- With respect to hardware faults, the problem in injecting software faults is due to the **difficulty of defining realistic fault models** for the software

IDEAS?

HW Fault Injection

HW Fault Injection Techniques

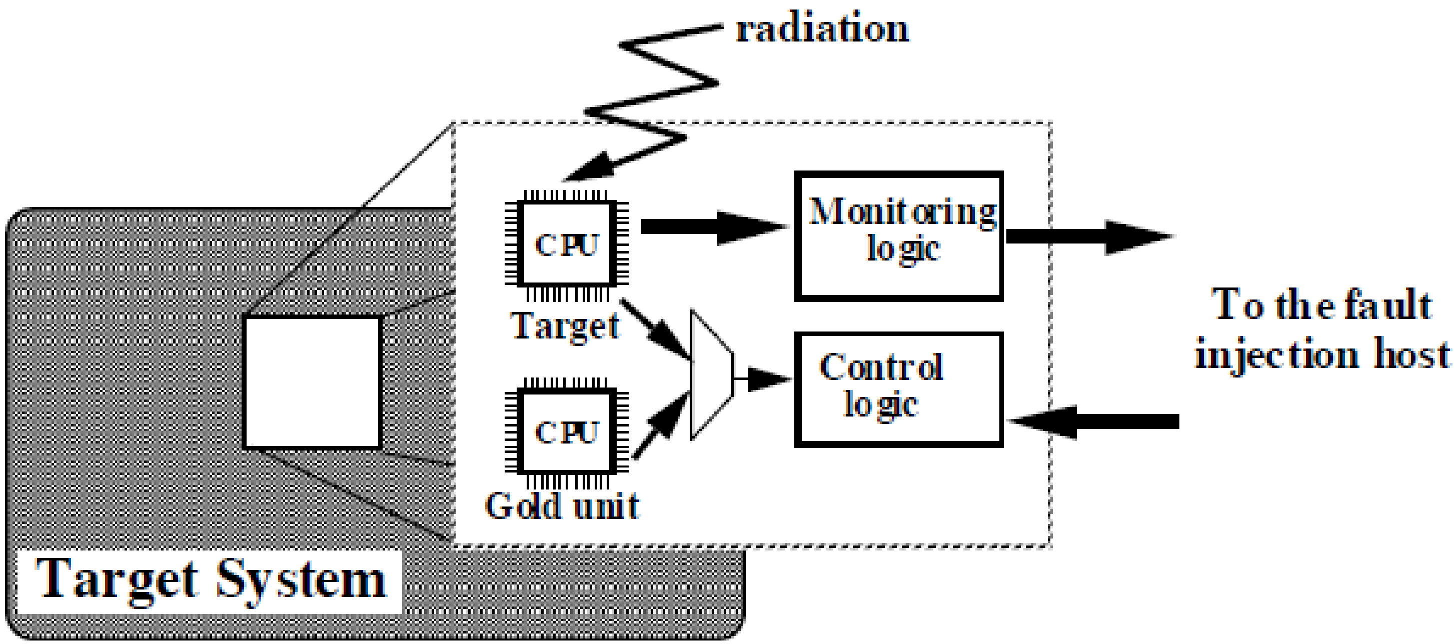
- Physical faults
  - Physical Fault Injectors
  - SoftWare Implemented Fault Injection (SWIFI)



First, Fault injection research focused mostly in the injection of physical faults.

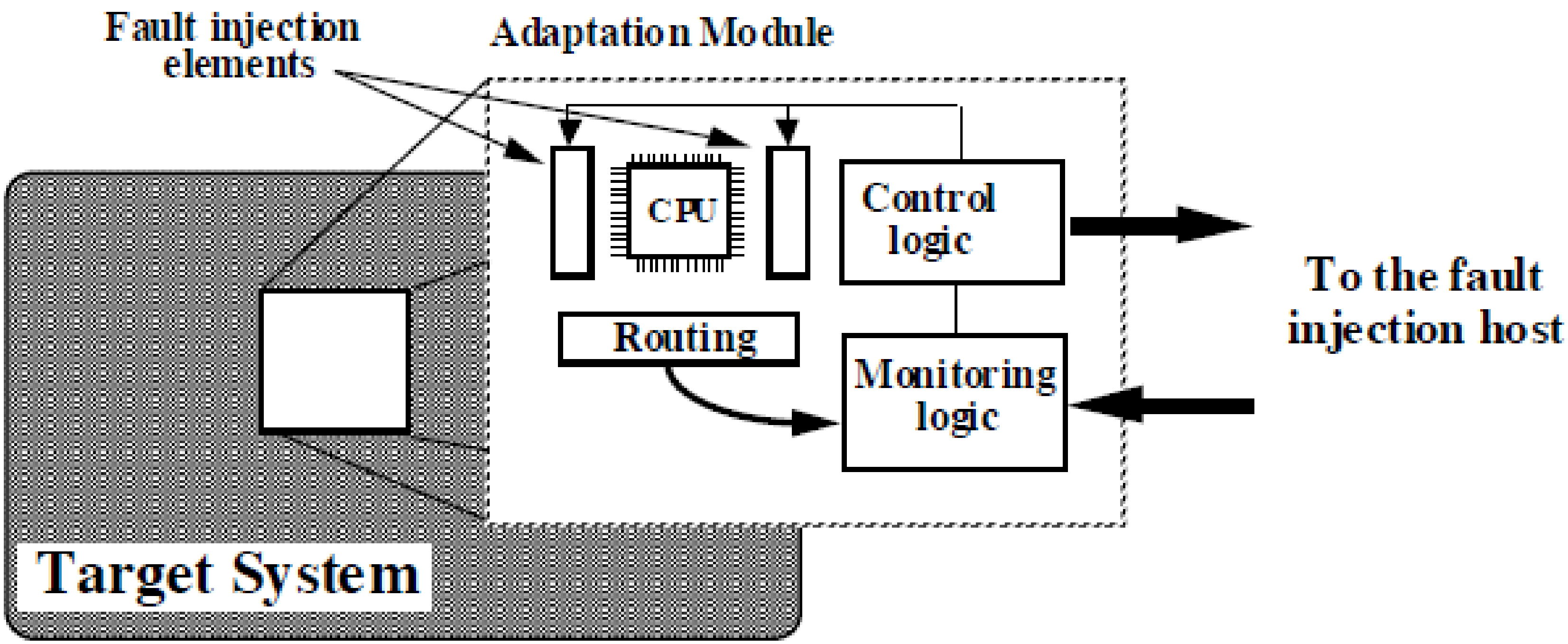
Physical Fault Injectors

- Possible approaches for the injection of hardware faults:
  - bringing the system near to a radiation source (**heavy-ion radiation**)



Physical Fault Injectors (2)

- Possible approaches for the injection of hardware faults:
  - alter the voltage at the ends of a circuit (**pin-level injection**)



HW Fault Injection Techniques

- Many problems today:
  - Hardware is too complex
  - **Poor controllability**
  - **Poor observability of the effects of the faults**
  - Huge development efforts
  - **Low portability**

SW fault injection

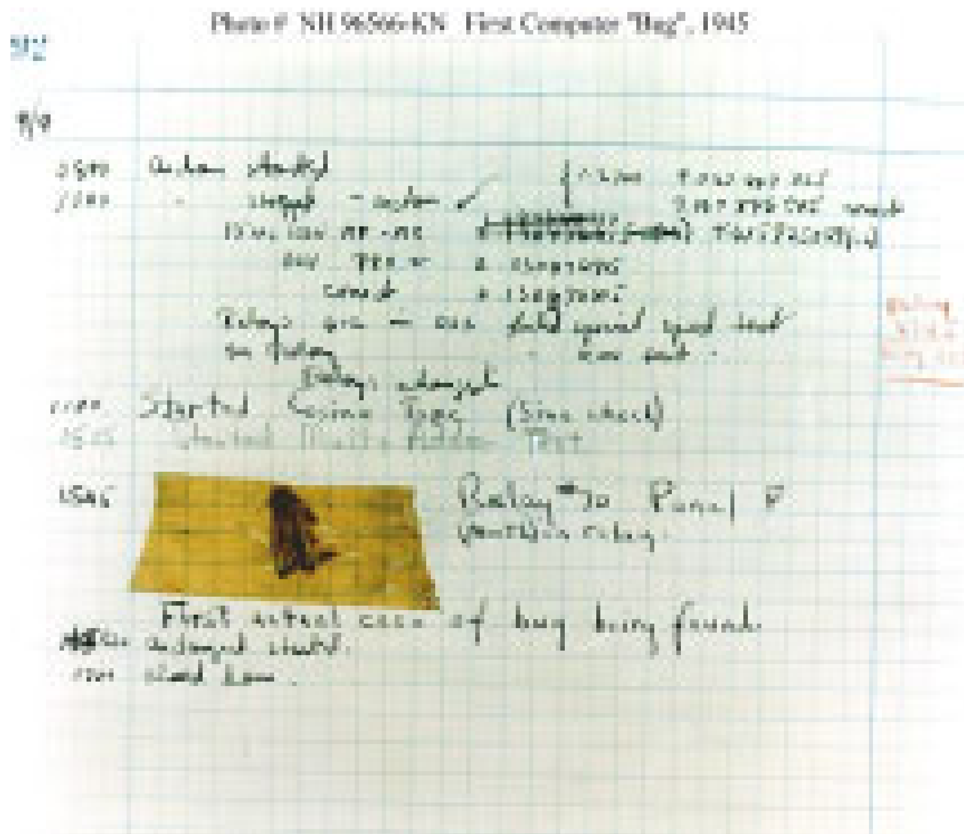
Software faults: the main cause of computer failures

Software faults (i.e., defects or bugs) are the major cause of computer failures

- The increasing complexity of software, the pressure to shrink **time-to-market**, and high cost of software testing contribute to keep bugs as the main computer failure cause.
  - Many failure reports available in the Internet
  - http://www.teach-ict.com/news/news_stories/news_computer_failures.htm
  - Cost thousands of millions of euros every year (occasionally software bugs cost human lives)

The first "bug"

- **Harvard University Mark II Aiken Relay Calculator**
  - "On the 9th of September, 1947, when the machine was experiencing problems, an investigation showed that there was a moth trapped between the points of Relay #70, in Panel F.
  - The operators removed the moth and affixed it to the log. The entry reads: "First actual case of bug being found."



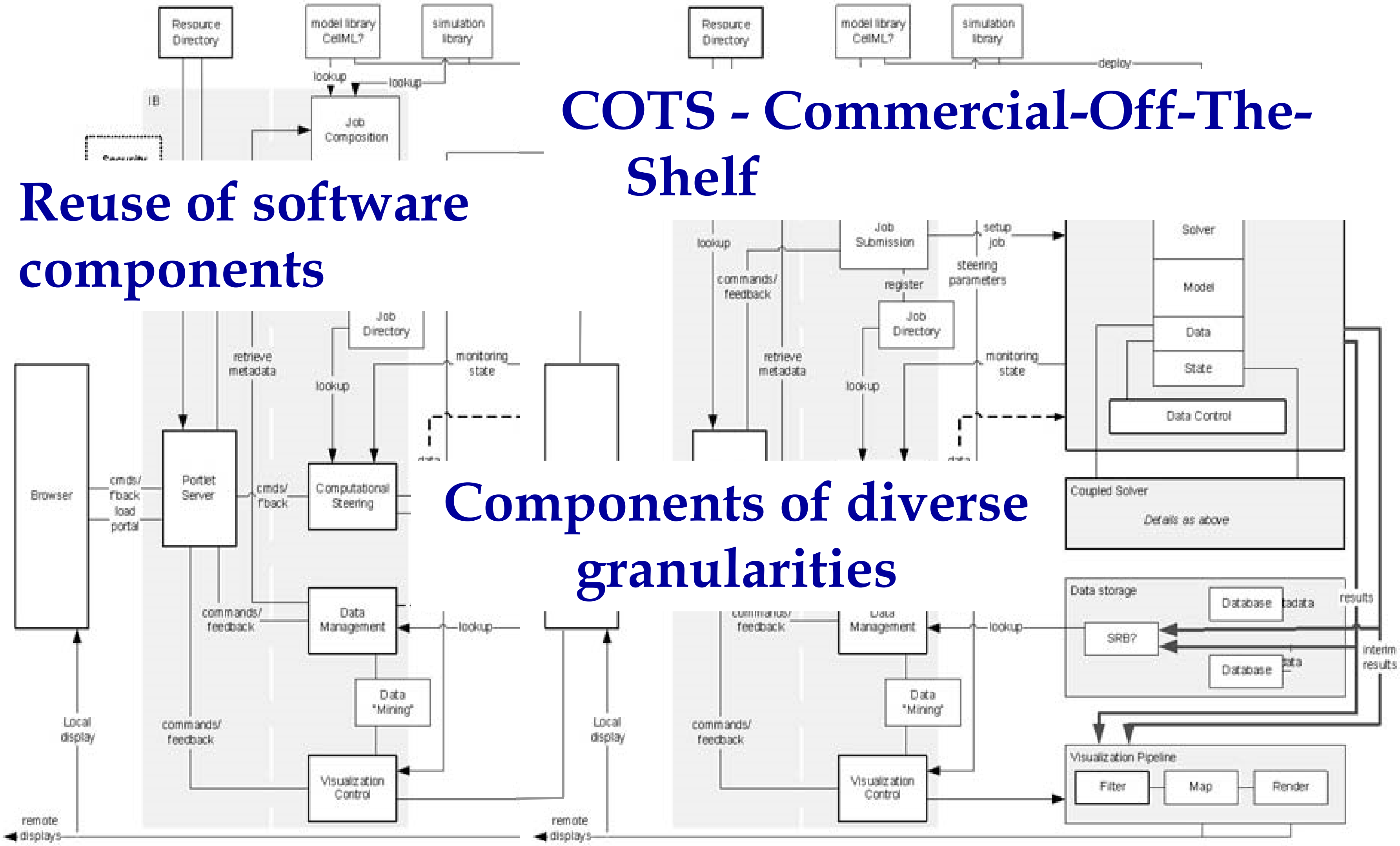
http://www.jamesshuggins.com/h/tek1/first_computer_bug.htm

ONCE FOUND, Bugs are oddly Simple

- Project Mercury's FORTRAN code had the following fault:
  - DO I=1.10 instead of ... DO I=1,10

- An F-18 crashed because of a missing exception condition:
  - if ... then ... without the else clause that was thought could not possibly arise
- In simulation, an F-16 program bug caused the virtual plane to flip over whenever it crossed the equator, as a result of a missing minus sign to indicate south latitude.
- The Bank of New York (BoNY) had a $32 billion overdraft as the result of a 16-bit integer counter that went unchecked.

Example: development based in components



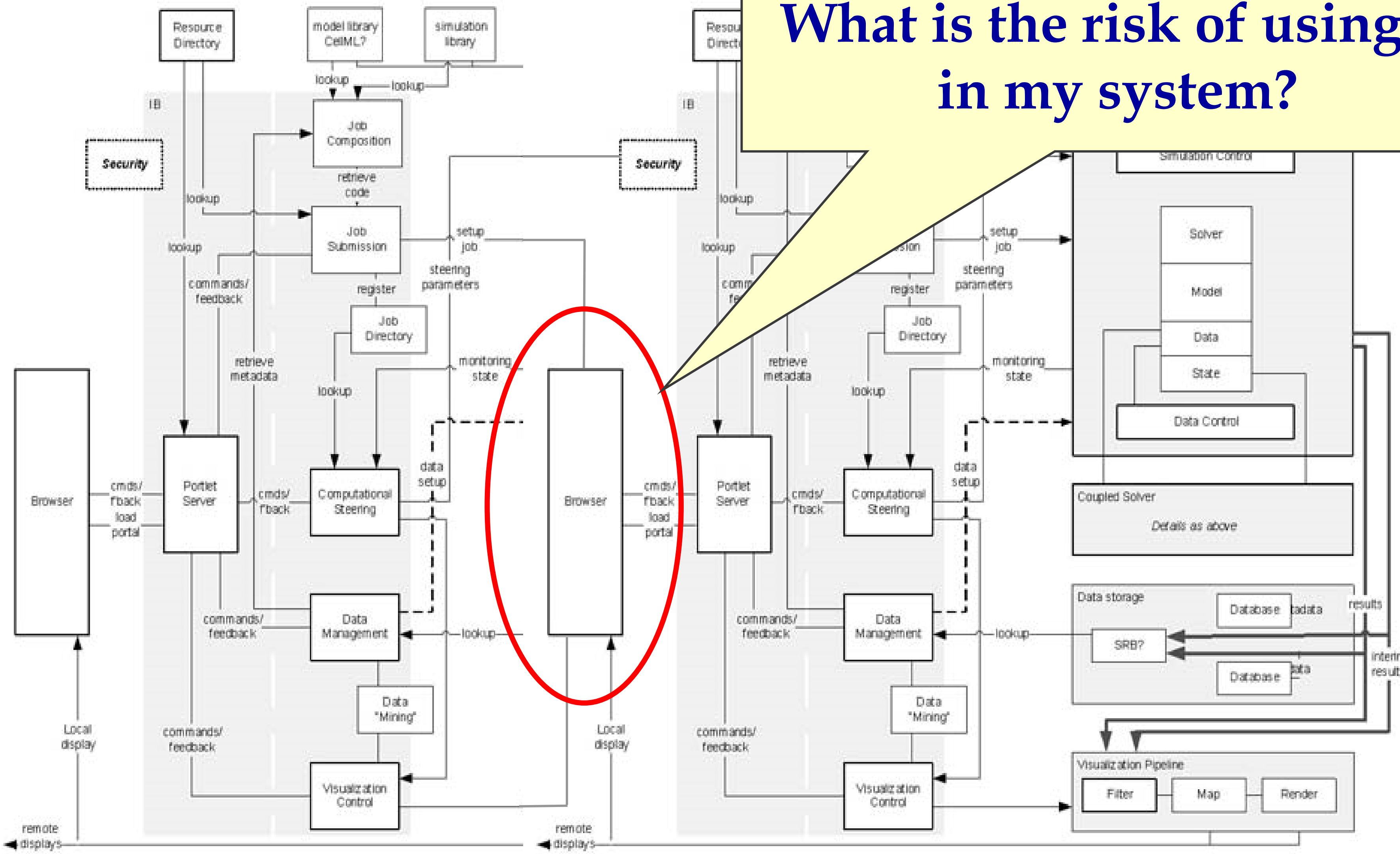
COTS - Commercial-Off-The-Shelf

Reuse of software components

Components of diverse granularities

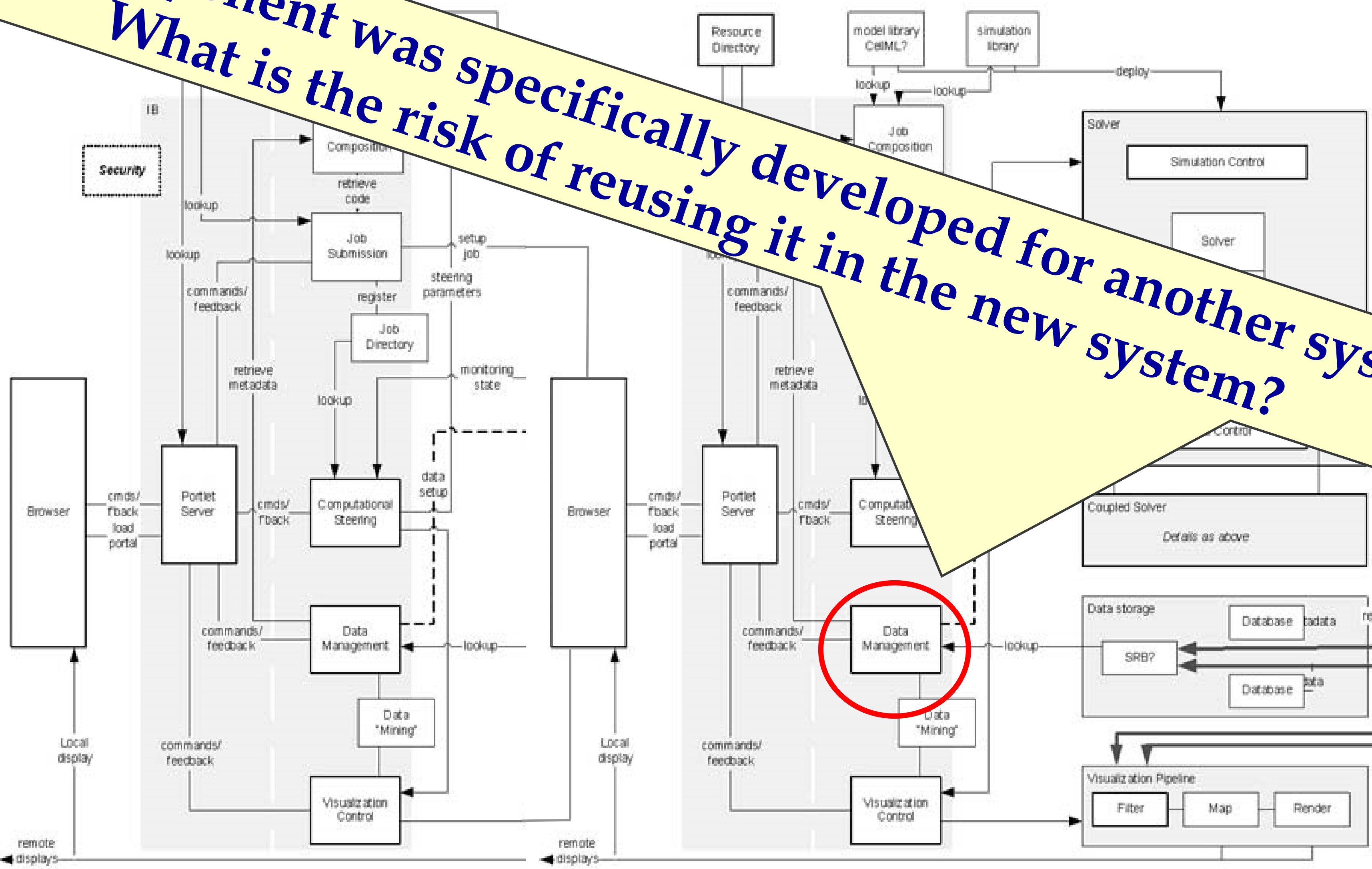
Example 1

This is a COTS component! What is the risk of using it in my system?



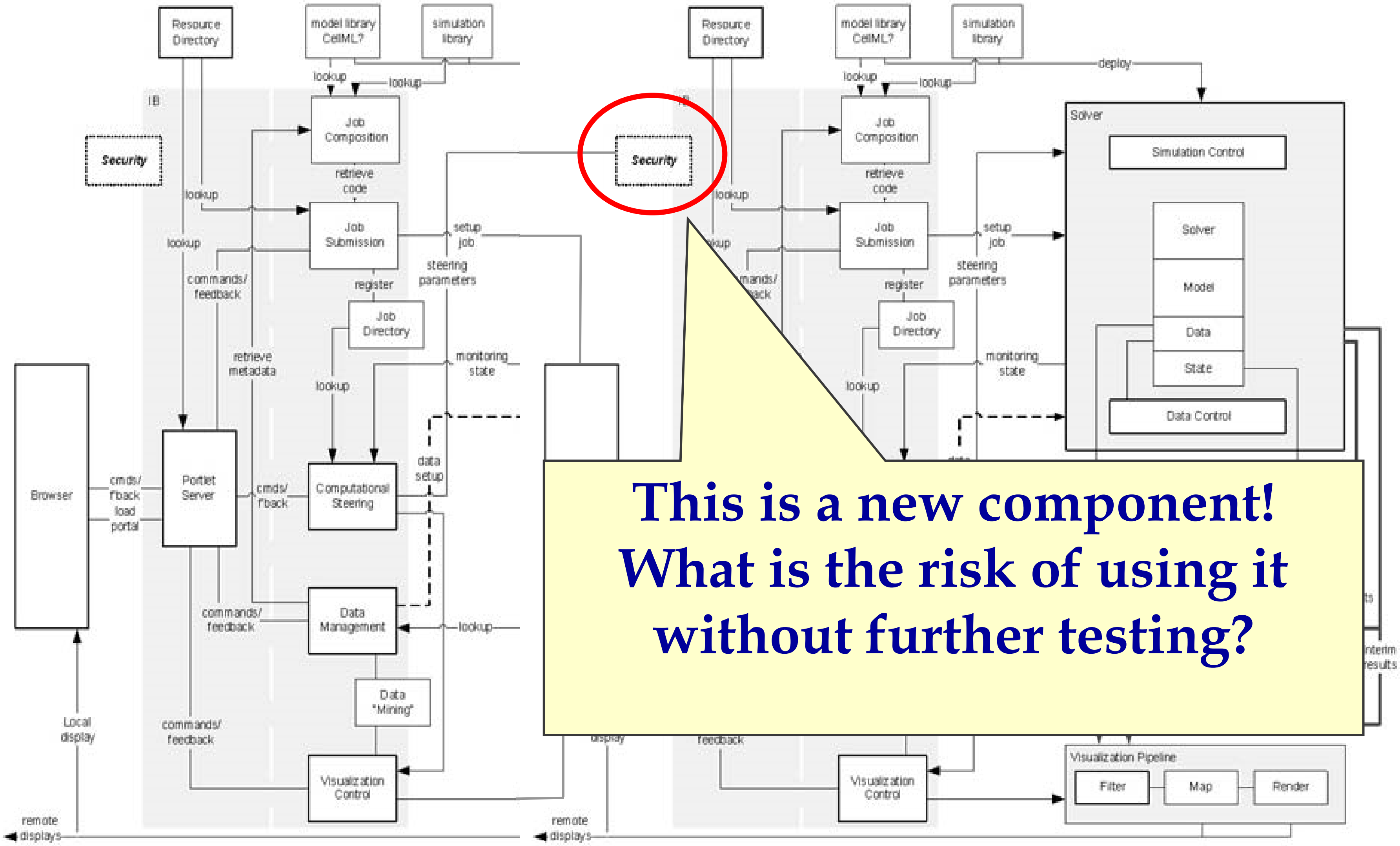
Example 2

This component was specifically developed for another system! What is the risk of reusing it in the new system?



UNIVERSITÀ DEGLI STUDI FIRENZE
DIMAI
DIPARTIMENTO DI MATEMATICA E INFORMATICA "ULISSE DINI"

Example 3



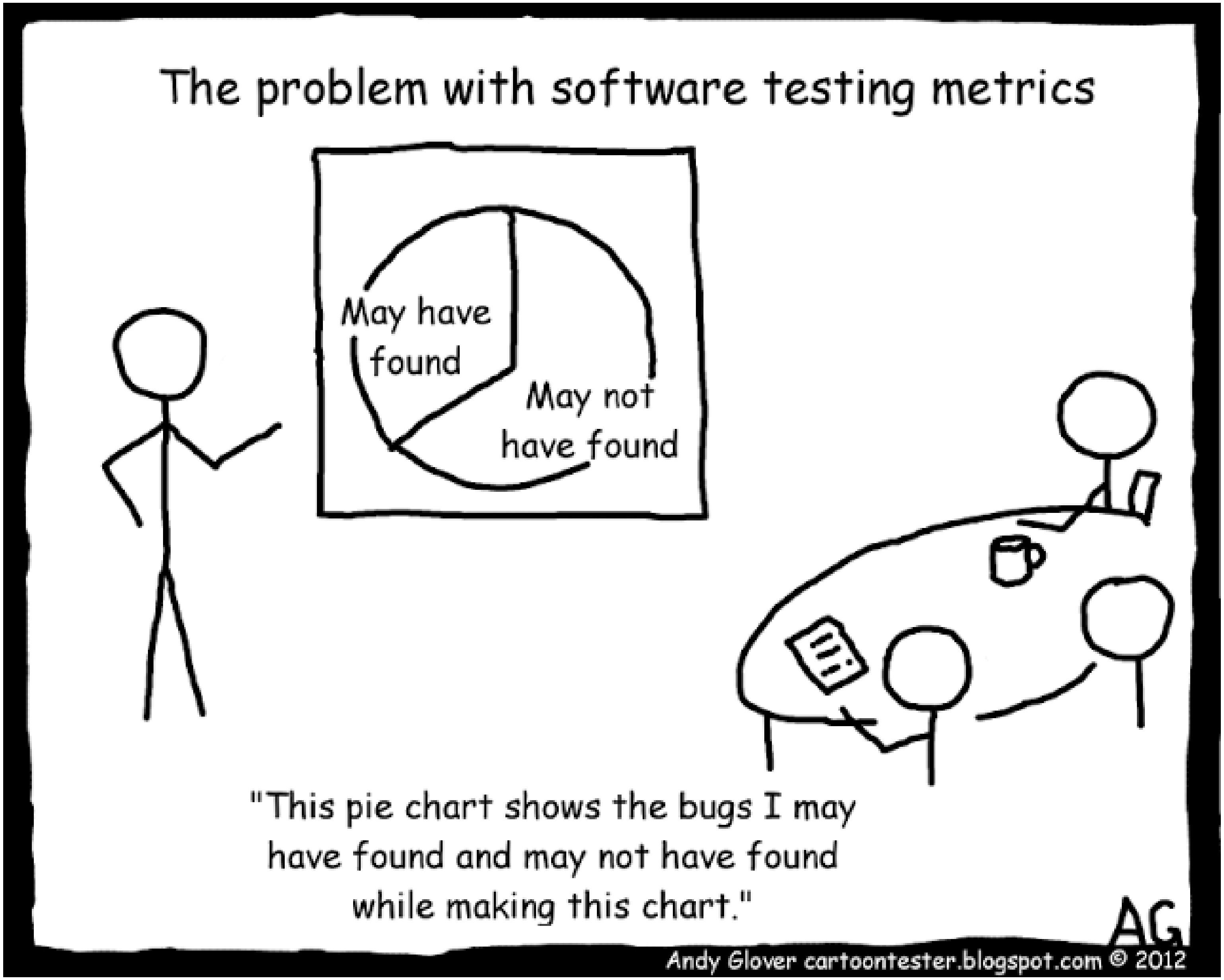
This is a new component!
What is the risk of using it without further testing?

SWFI pros and cons

- **Advantages**
  - Not much affected by the complexity of the target
  - Low complexity, cost and development effort
  - Reasonably portable, no physical interferences

- **Typical disadvantages**
  - Do not cover faults in peripheral devices, ASICS, etc
  - Limited monitoring capabilities
  - Tools may have a great impact on the target system behavior (i.e., change the target system's behavior by adding extra code for the fault injection tool)

To relax a bit…

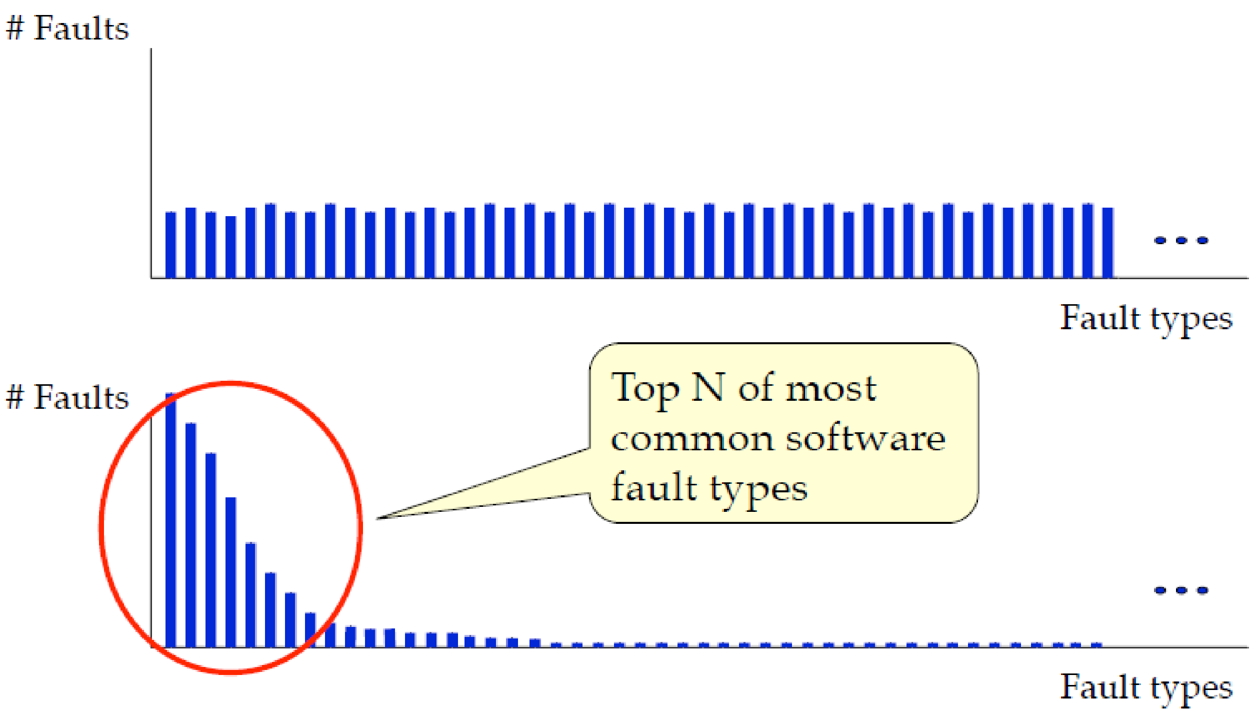


Orthogonal Defect Classification

- Paper about SW bugs

J. A. Durães and H. S. Madeira.

Emulation of Software faults: A Field Data Study and a Practical Approach. IEEE Transactions on Software Engineering, 32(11):849–867, 2006.



Orthogonal Defect Classification

- Christmansson and Chillarege (1996) have proposed a methodology for defining fault injection experiments of realistic sw faults
- ODC (*Orthogonal Defect Classification*): a measurement technique that classifies software faults as
  - based on the code change made to fix it, and
  - based on the system condition during which the fault has been activated.

Classifying faults from a fault injection perspective

- A software fault is a construction in a programing language that is incorrect due to one of the following reasons:

- Missing construction
  - Inexistent code (i.e., missing code in the program)
- Wrong construction
  - Syntactically correct code, but it is not adequate to fulfill the specification
- Extraneous construction
  - Existence of not needed code (that causes wrong behavior)

Software faults characterization

ODC type	Nature	Example
Assignment	**Missing**	Missing initialization of a variable
	Wrong	Assignment of a wrong value to a variable
	Extraneous	Extraneous (and not needed) assignment of a value to a variable
Checking	**Missing**	Missing *if* condition
	Wrong	Wrong logic expression
	Extraneous	Extraneous *if* condition
Interface	**Missing**	Missing parameter in a function call (or in the interface between components)
	Wrong	Wrong parameter value between functions
	Extraneous	Extraneous parameter in a function call
Algorithm	**Missing**	Part of the algorithm is missing (e.g., missing function call)
	Wrong	Wrong algorithm (e.g., wrong function is called)
	Extraneous	Too many instructions in the algorithm (that may affect the behavior)
Function	**Missing**	Missing components
	Wrong	The code structure needs to be restructured
	Extraneous	Extraneous parts of code in a given component

Fault distribution per ODC type

- Some types of faults are more representative than others (i.e. **more interesting for fault injection**): **Assignment**, **Checking**, **Algorithm**
- Other field studies have similar distributions

ODC Type	# of faults	ODC distribution (Coimbra study)	ODC distribution (IBM study)
Assignment	118	**22.1 %**	21.98 %
Checking	137	**25.7 %**	17.48 %
Interface	43	**8.0 %**	8.17 %
Algorithm	198	**37.2 %**	43.41 %
Function	36	**6.7 %**	8.74 %

Characterization of the nature of the faults per ODC type

ODC Types	Nature	# faults
Assignment	Missing	44
	Wrong	64
	Extraneous	10
Checking	Missing	90
	Wrong	47
	Extraneous	0
Interface	Missing	11
	Wrong	32
	Extraneous	0
Algorithm	Missing	155
	Wrong	37
	Extraneous	6
Function	Missing	21
	Wrong	15
	Extraneous	0

- **Missing** and **wrong** are the most frequent
- This trend is consistent across all the ODC fault types

Characterization of faults in the different programs

Nature of the faults	Software Programs									
	CDEX	Vim	FCiv	Pdf2h	GAIM	Joe	ZSNES	Bash	LKernel	Total
Missing	3	157	35	11	17	34	1	0	63	**321**
Wrong	8	85	18	9	6	41	2	2	24	**195**
Extraneous	0	7	0	0	0	3	0	0	6	**16**

- **Missing** faults are the most frequent
- **Extraneous** faults are quite rare
- This tendency is consistent across different programs

UNIVERSITÀ DEGLI STUDI FIRENZE
DIMAI
DIPARTIMENTO DI MATEMATICA E INFORMATICA "ULISSE DINI"

"Top-12" ODC Faults

Fault types	Description	% of total observed	ODC classes
MIFS	Missing "If (cond) { statement(s) }"	9.96 %	Algorithm
MFC	Missing function call	8.64 %	Algorithm
MLAC	Missing "AND EXPR" in expression used as branch condition	7.89 %	Checking
MIA	Missing "if (cond)" surrounding statement(s)	4.32 %	Checking
MLPC	Missing small and localized part of the algorithm	3.19 %	Algorithm
MVAE	Missing variable assignment using an expression	3.00 %	Assignment
WLEC	Wrong logical expression used as branch condition	3.00 %	Checking
WVAV	Wrong value assigned to a value	2.44 %	Assignment
MVI	Missing variable initialization	2.25 %	Assignment
MVAV	Missing variable assignment using a value	2.25 %	Assignment
WAEP	Wrong arithmetic expression used in function call parameter	2.25 %	Interface
WPFV	Wrong variable used in parameter of function call	1.50 %	Interface
	Total faults coverage	**50.69 %**	

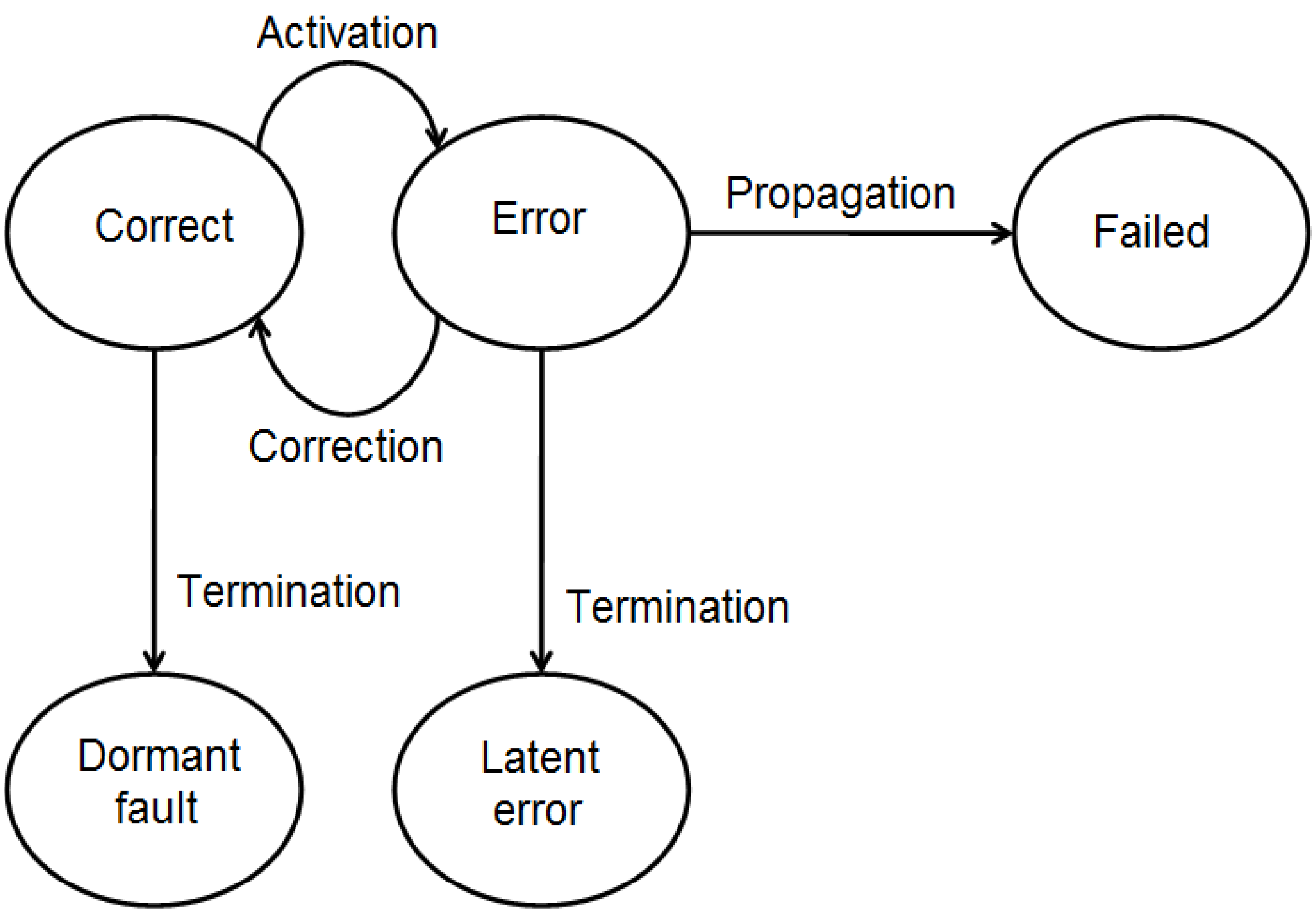
RCL
RESILIENT COMPUTING LAB

Error Injection

UNIVERSITÀ DEGLI STUDI FIRENZE
DIMAI DIPARTIMENTO DI MATEMATICA E INFORMATICA "ULISSE DINI"

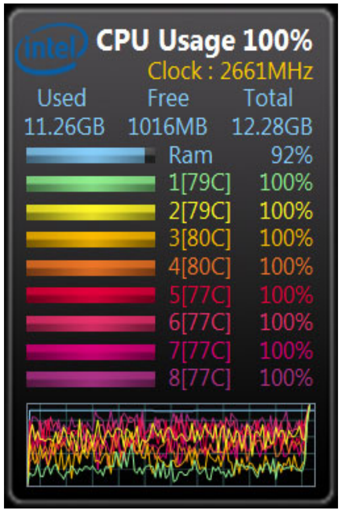
Phases of a Fault Injection experiment

Question: How can I be sure that an injected fault activates somewhere?



Error Injection

- The **Error Injection** is a Fault Injection approach that **accelerates** the occurrence of failures
  - *Avoid waiting for the activation of faults*
  - Facilitate the injection of certain types of fault
- **It forces directly the system into a wrong state**
  - for example, overwriting the value of an operation with an incorrect result



Question: What if my target application is a third-party application that can be viewed only as black-box?