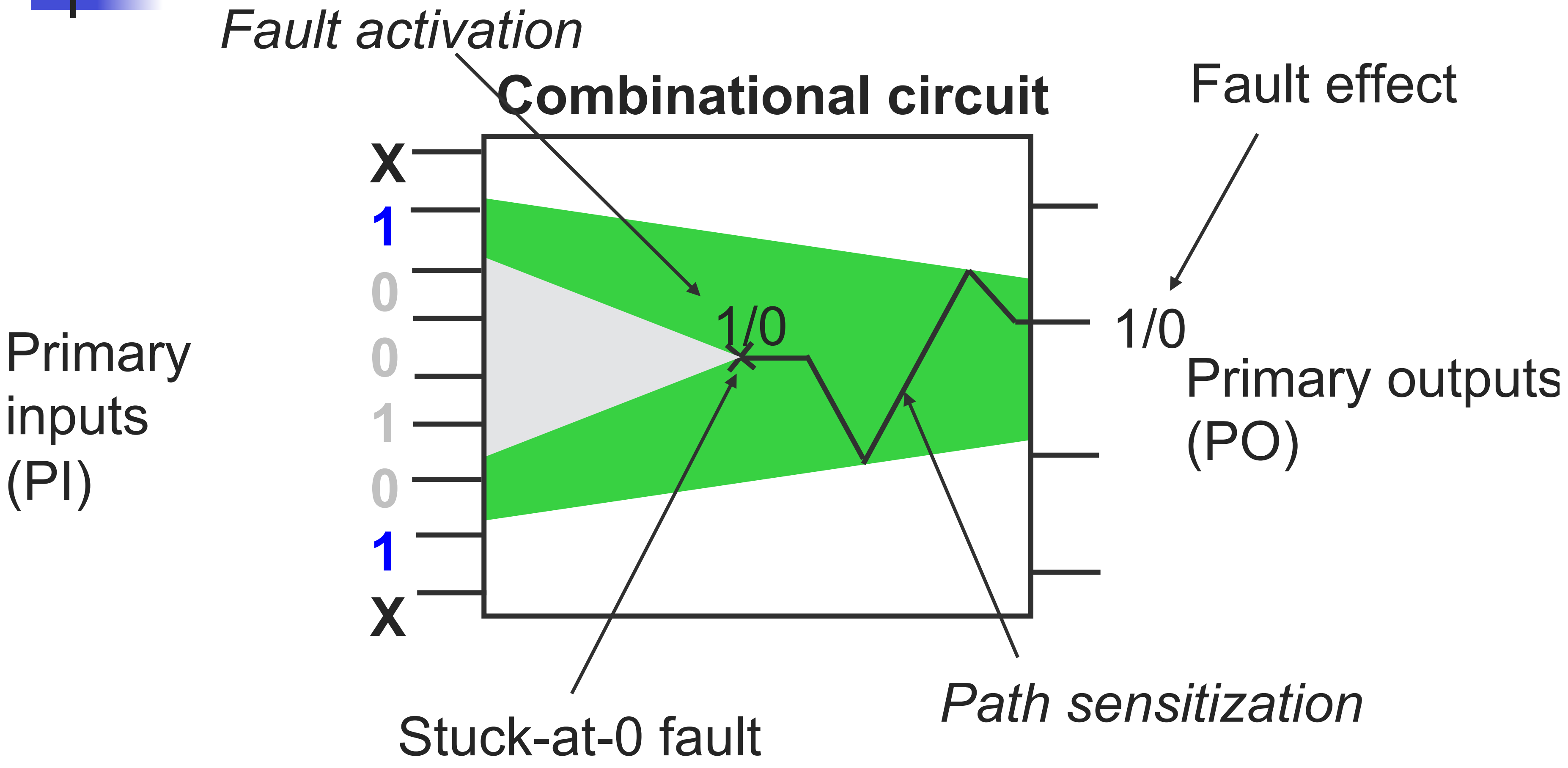
ATPG and Fault Simulation

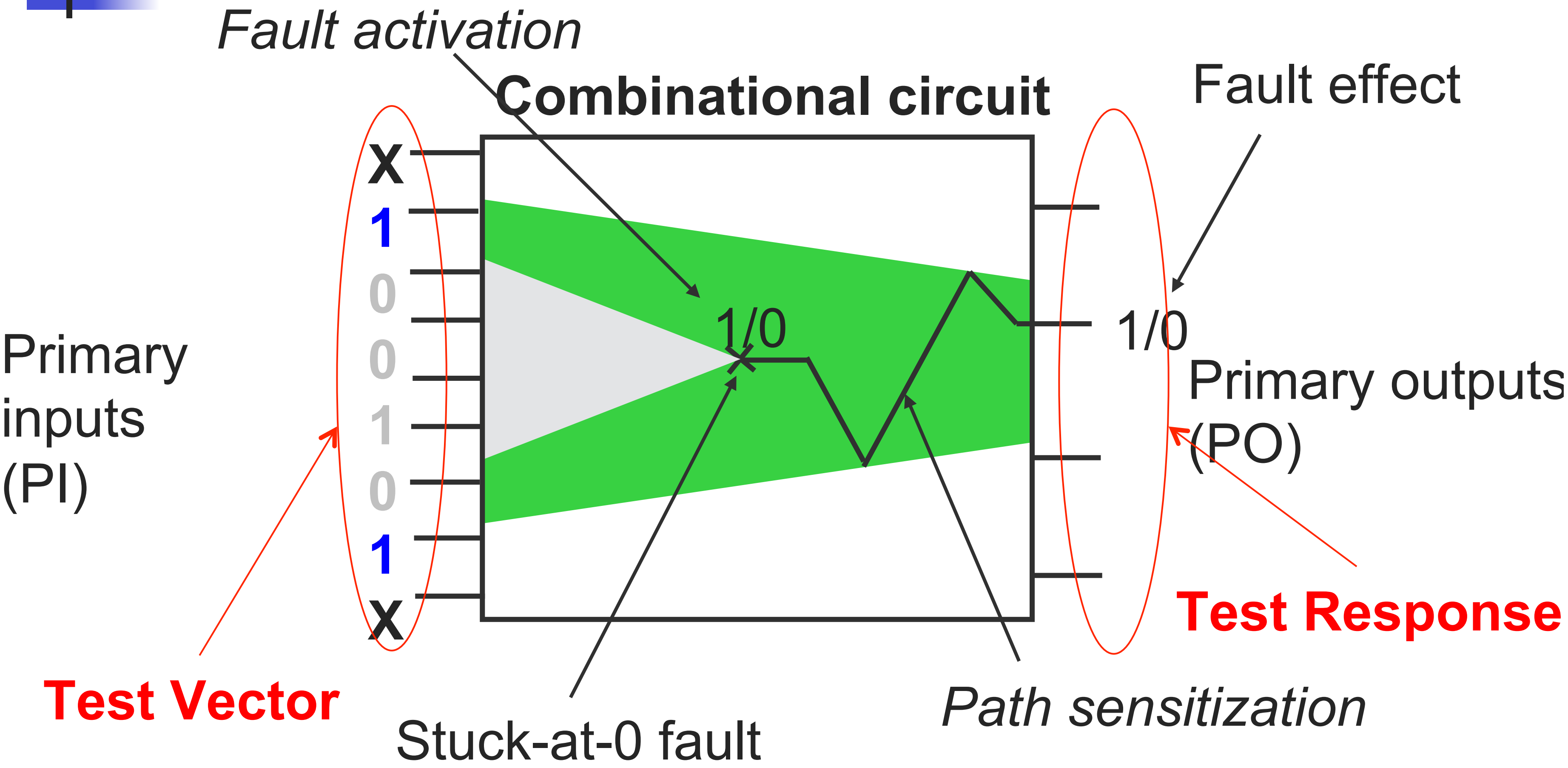
Alberto Bosio

bosio@lirmm.fr

What is a test?

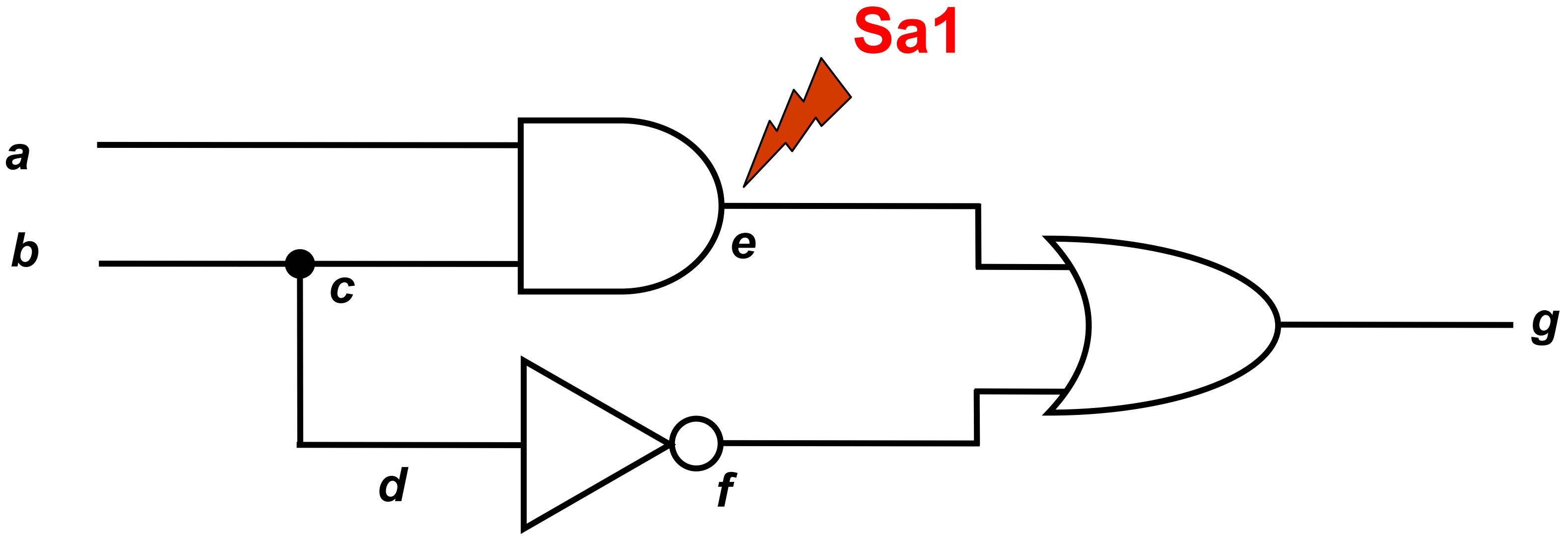


What is a test?



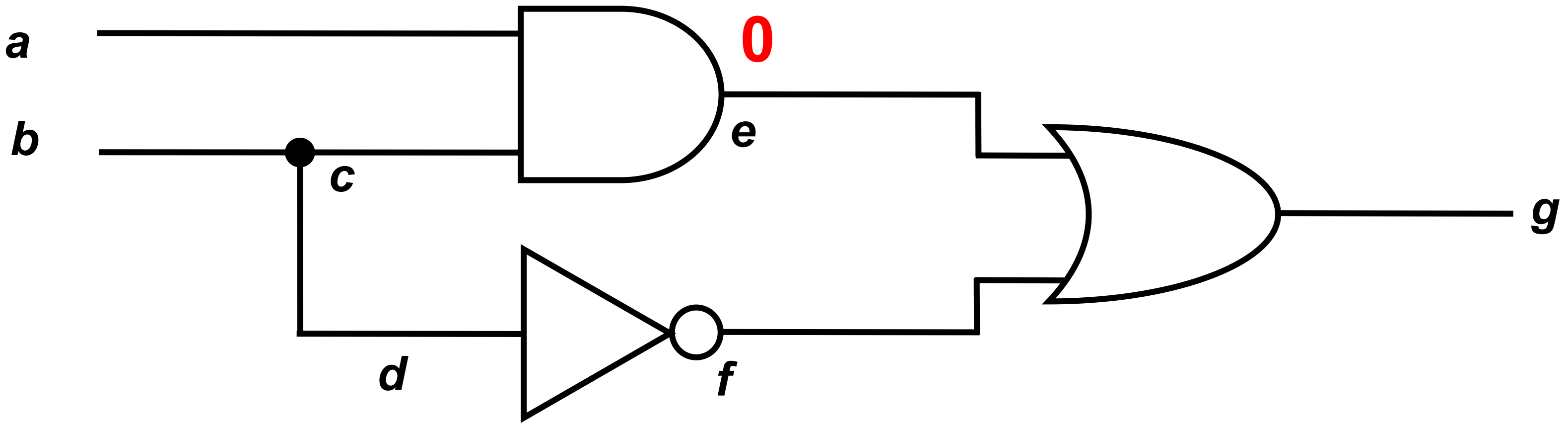
Example

- Generate a test for **e stuck-at-1**



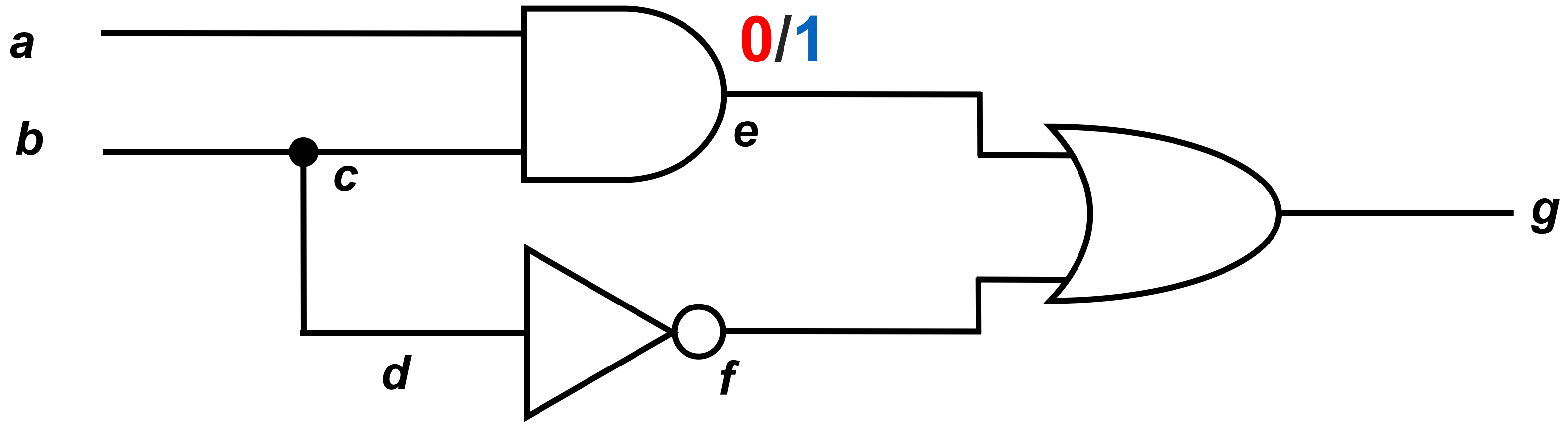
Example

- 1) Activate the fault



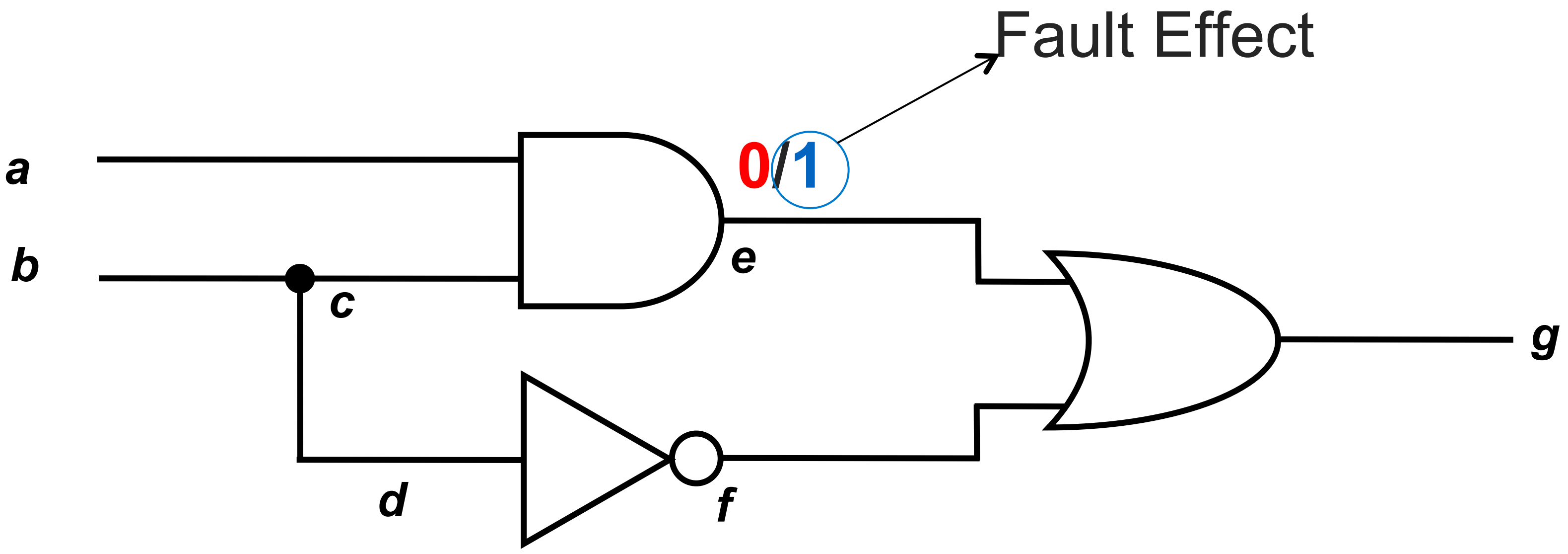
Example

- 1) Activate the fault



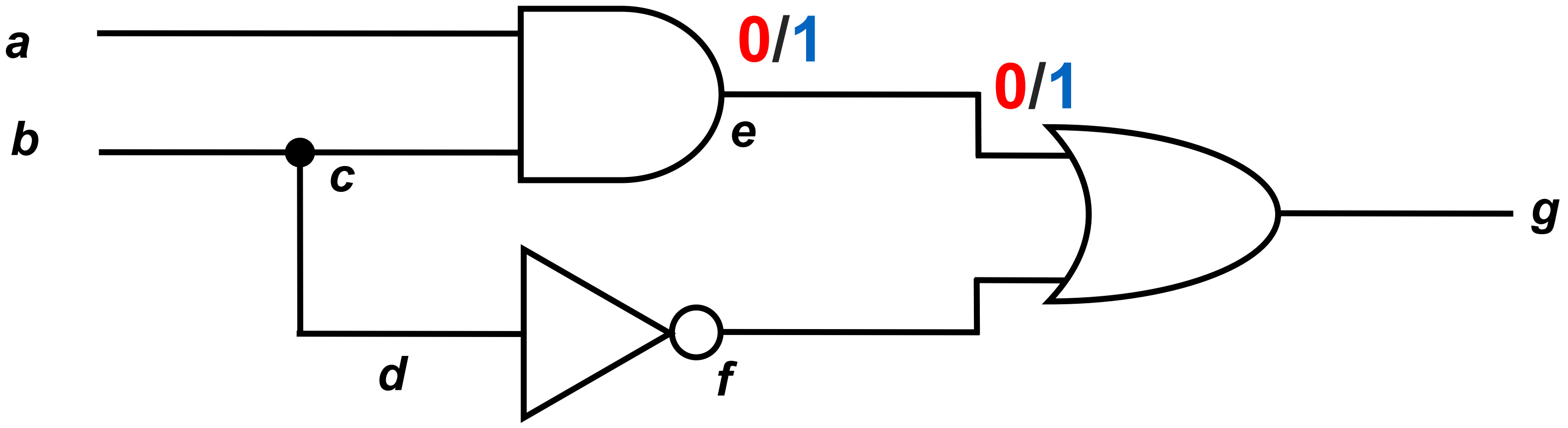
Example

- 1) Activate the fault



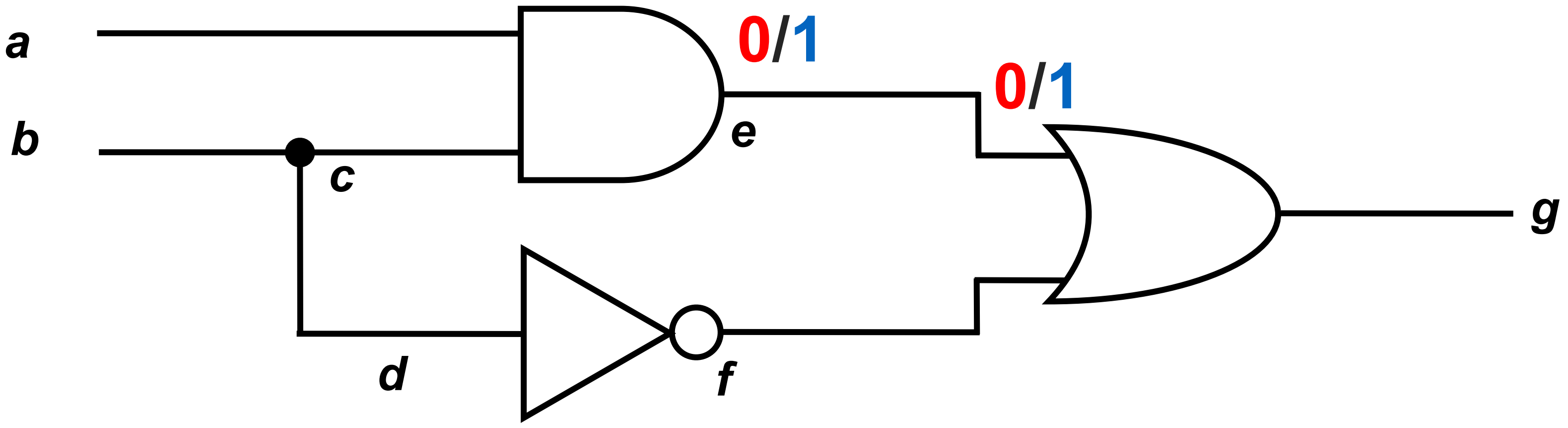
Example

- 1) Propagate the fault effect



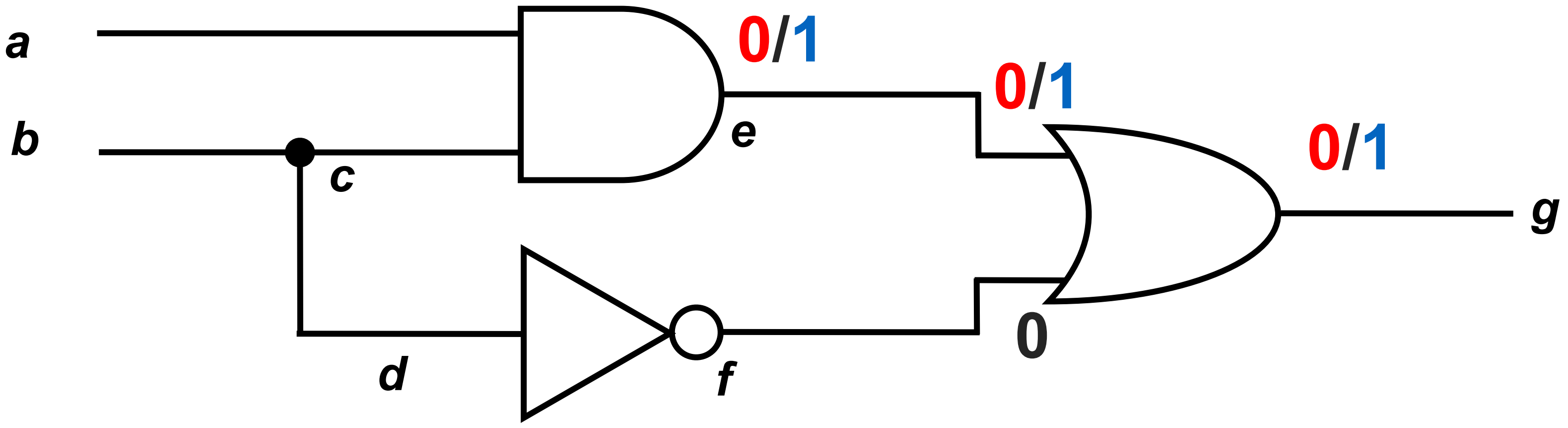
Example

- 1) Propagate the fault effect



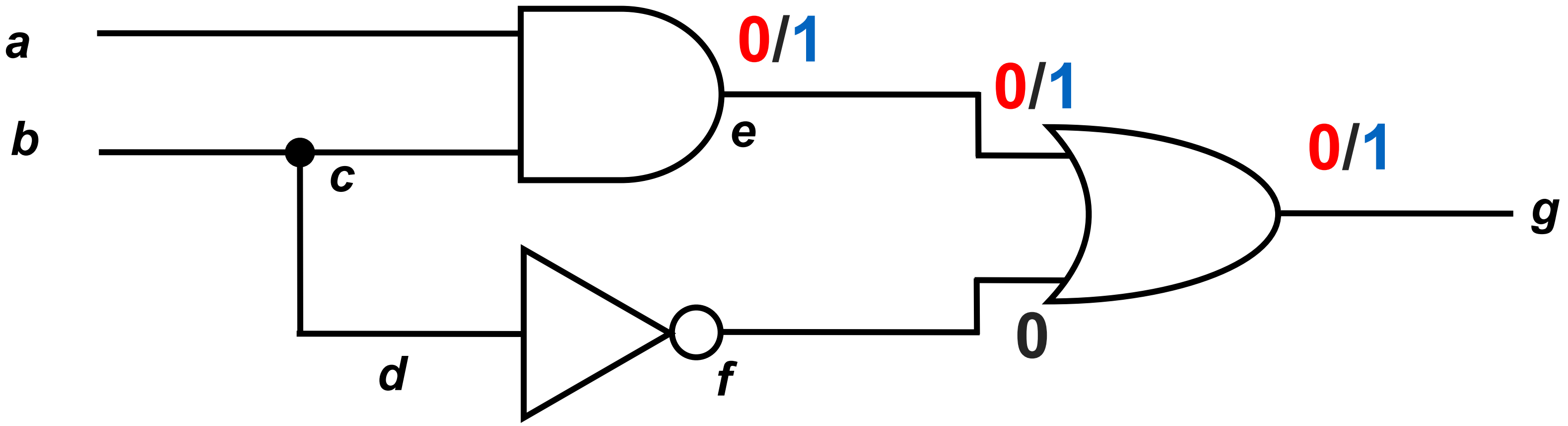
Example

- 1) Propagate the fault effect



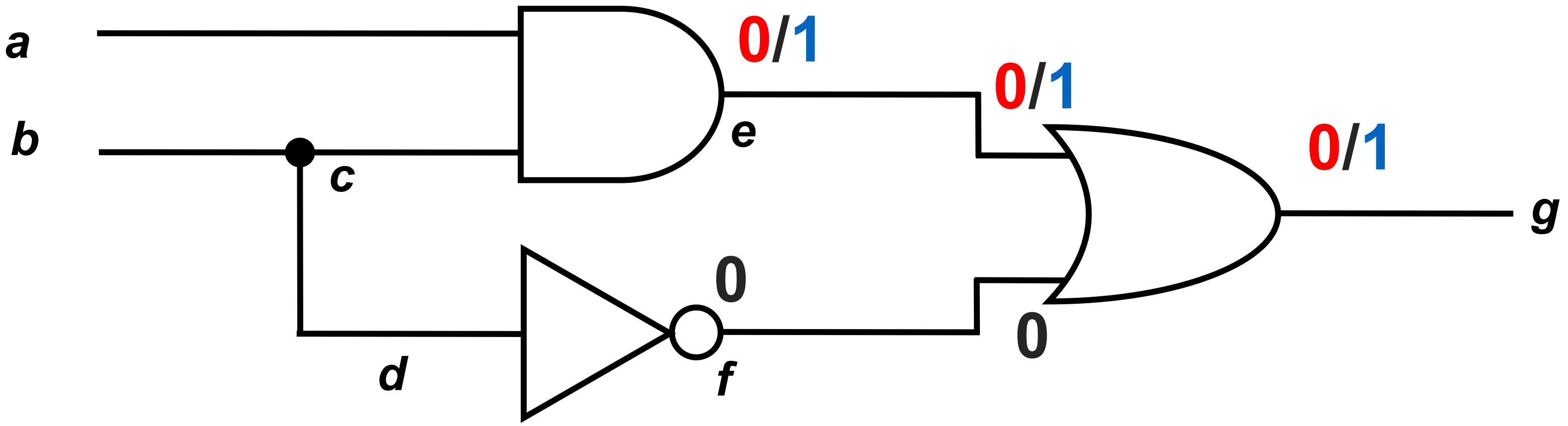
Example

- 1) Propagate the fault effect



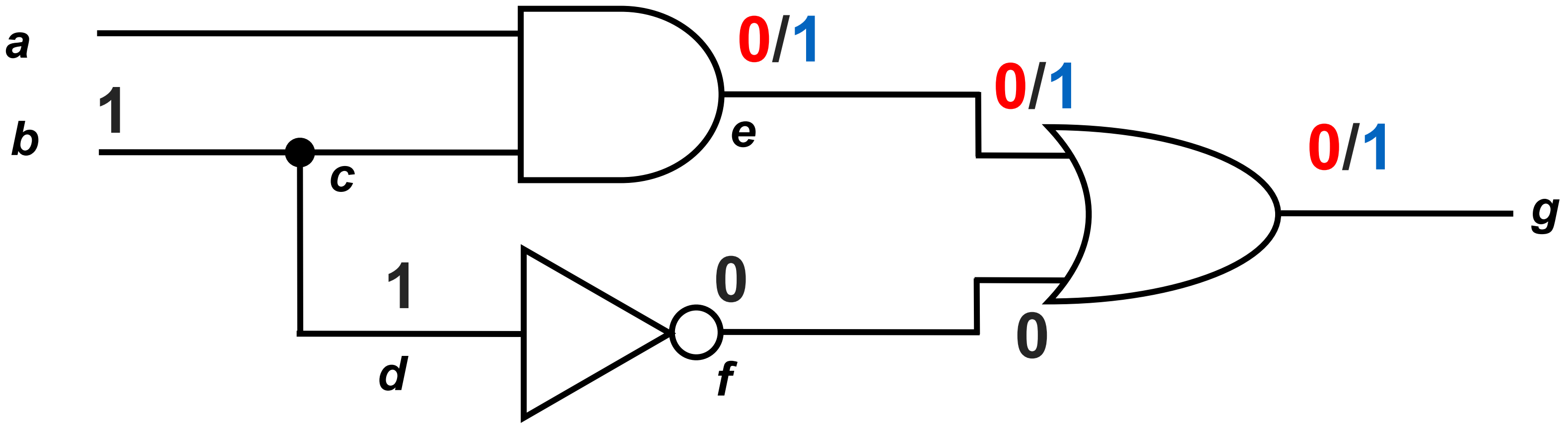
Example

- Justification



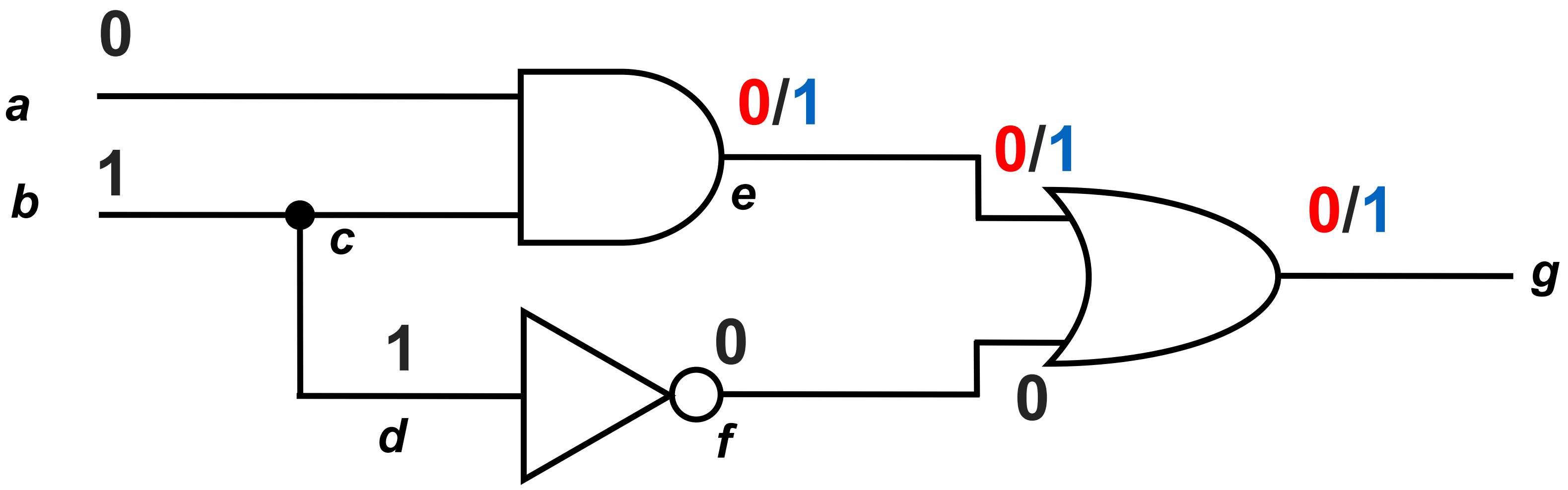
Example

- Justification



Example

- Justification

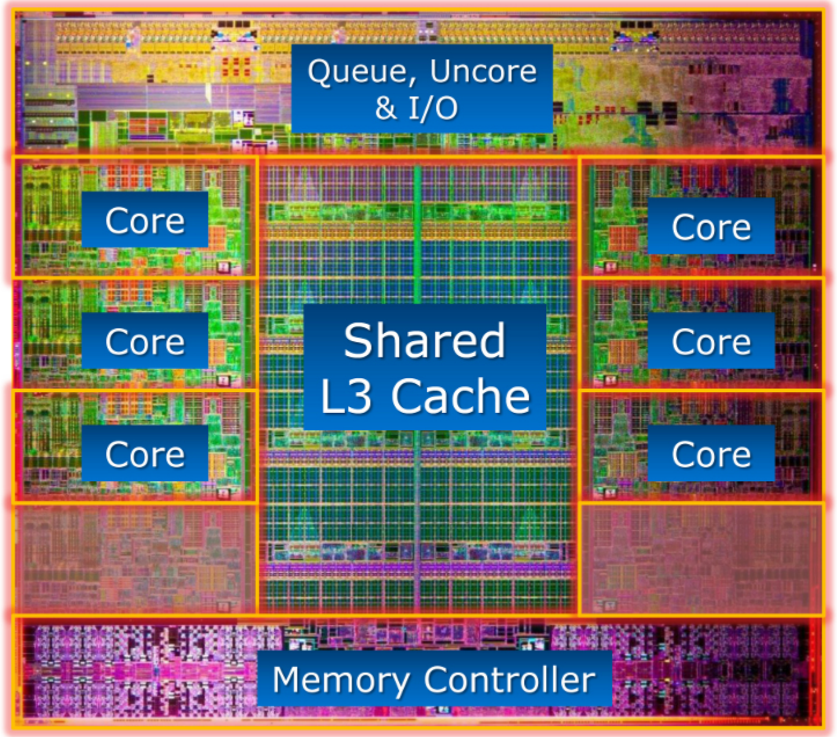


Some Considerations

- Test is easy
- But….

Some problems (the complexity)

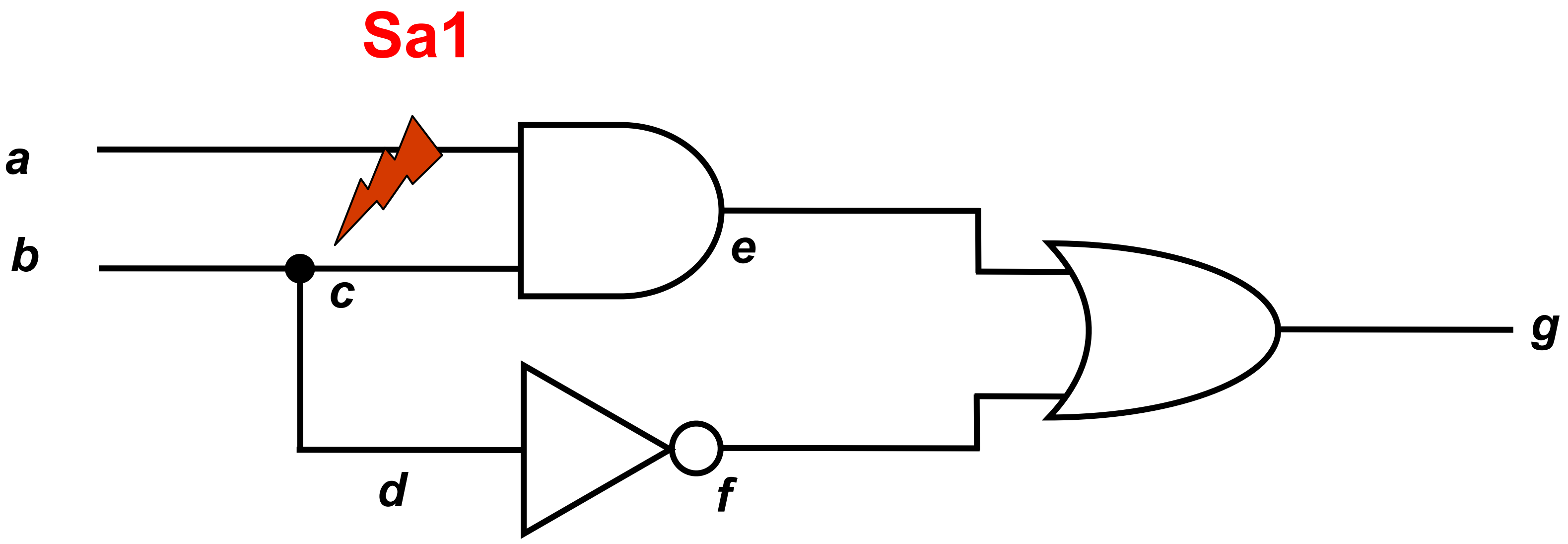
Intel® Core™ i7-3960X Processor Die Detail



2.2 Billion Transistors

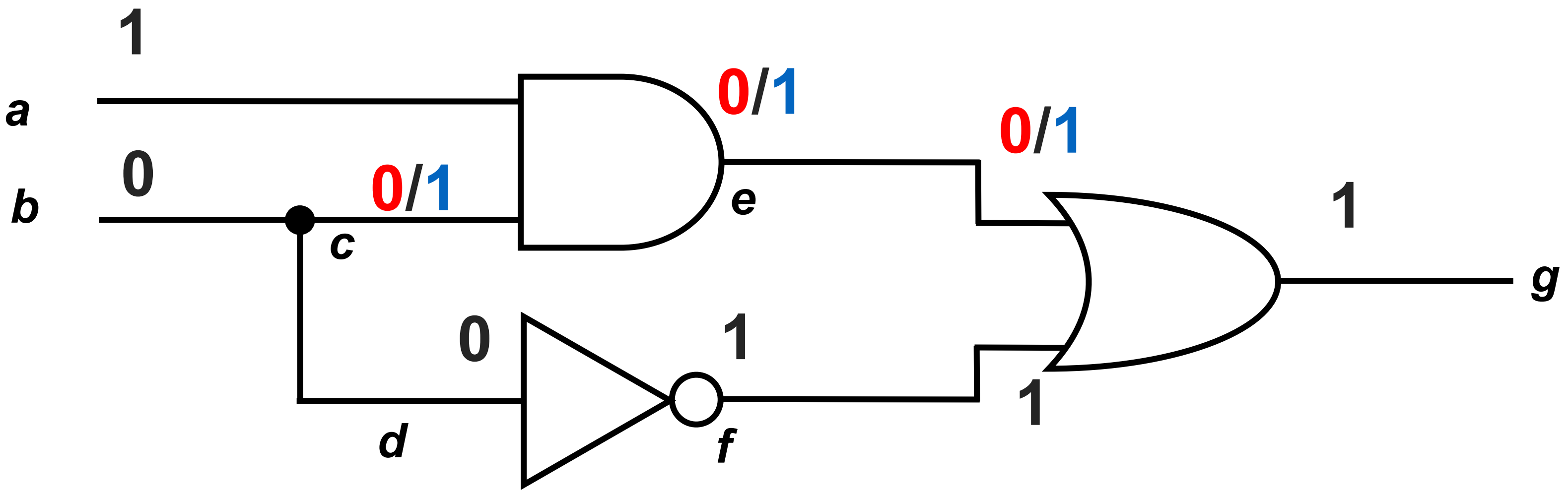
Some problems (the circuit)

- Generate a test for **c stuck-at-1**



Some problems (the circuit)

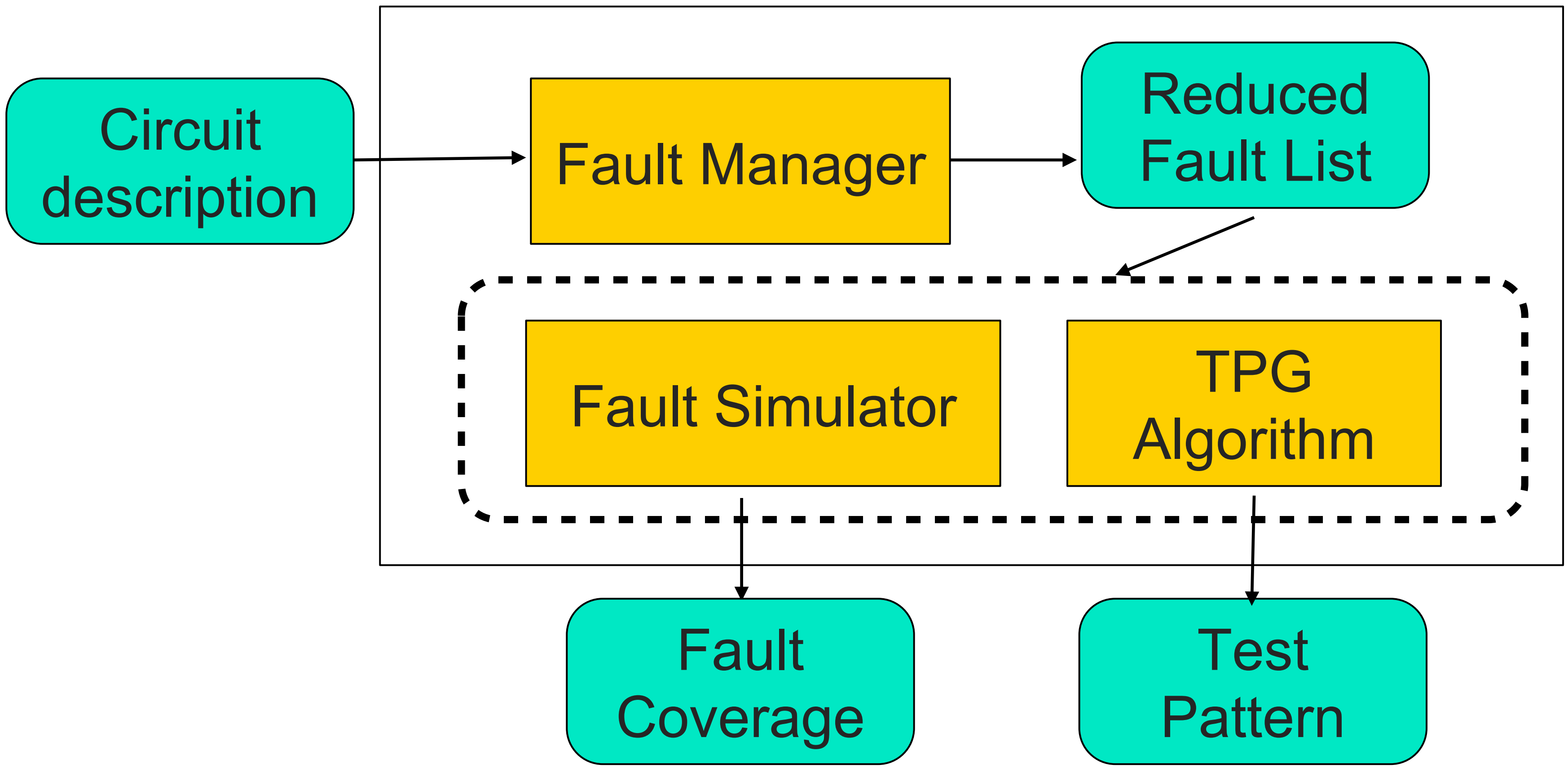
- **c stuck-at-1** is an untestable fault



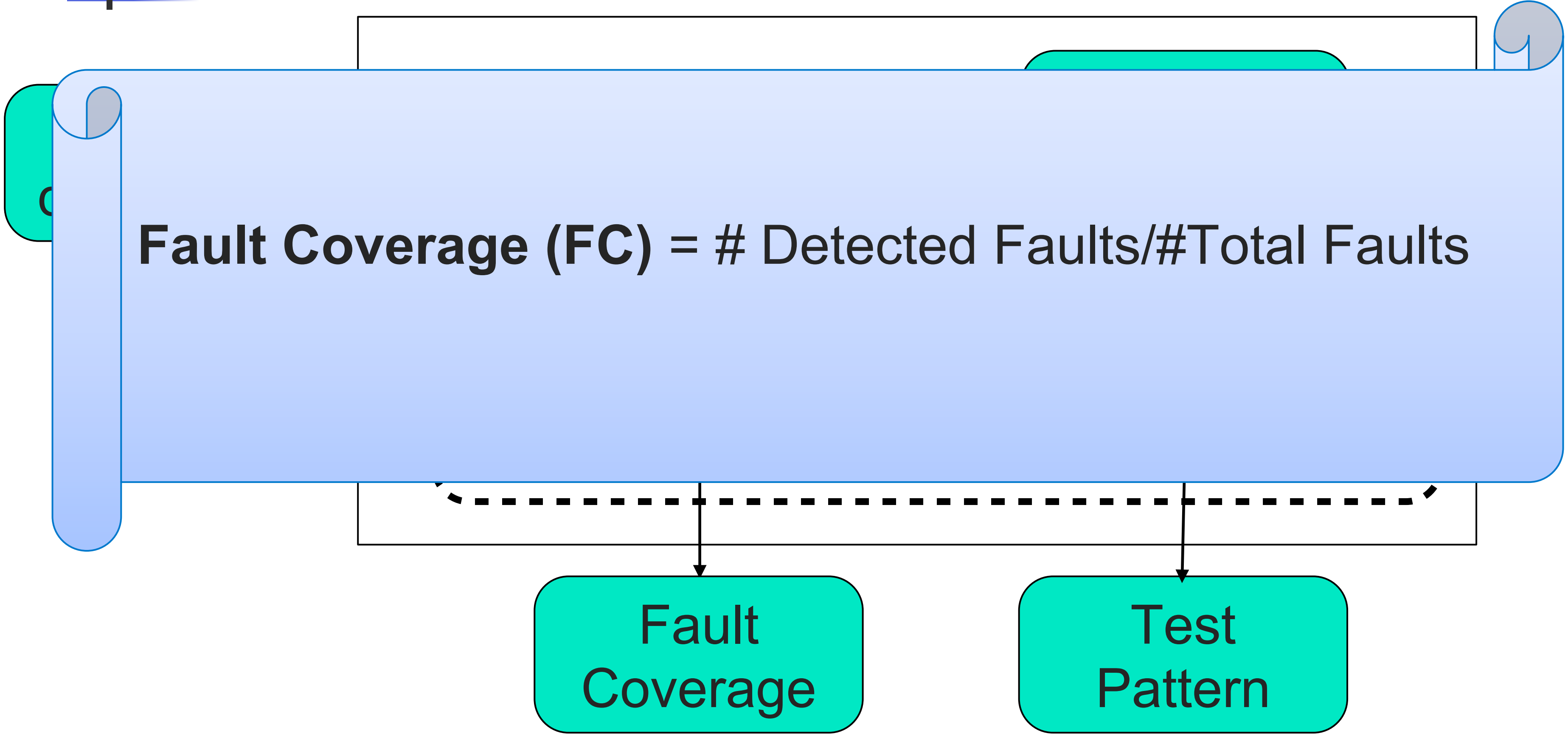
Goals

- You must use the appropriate tool
  - Automatic Test Pattern Generator (ATPG)

ATPG Architecture



ATPG Architecture



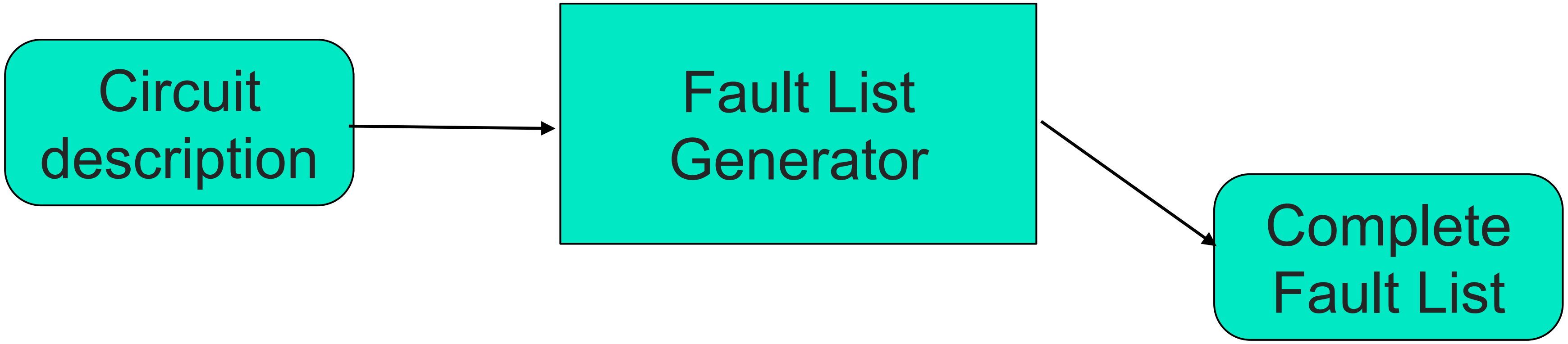
Fault Coverage (FC) = # Detected Faults/#Total Faults

The test plan

- Step 1:
  - Identify the set of target faults (complete fault list).

The test plan

- Step 1:
  - Tools – Fault list generator
    - (One of the components of the Fault Manager).



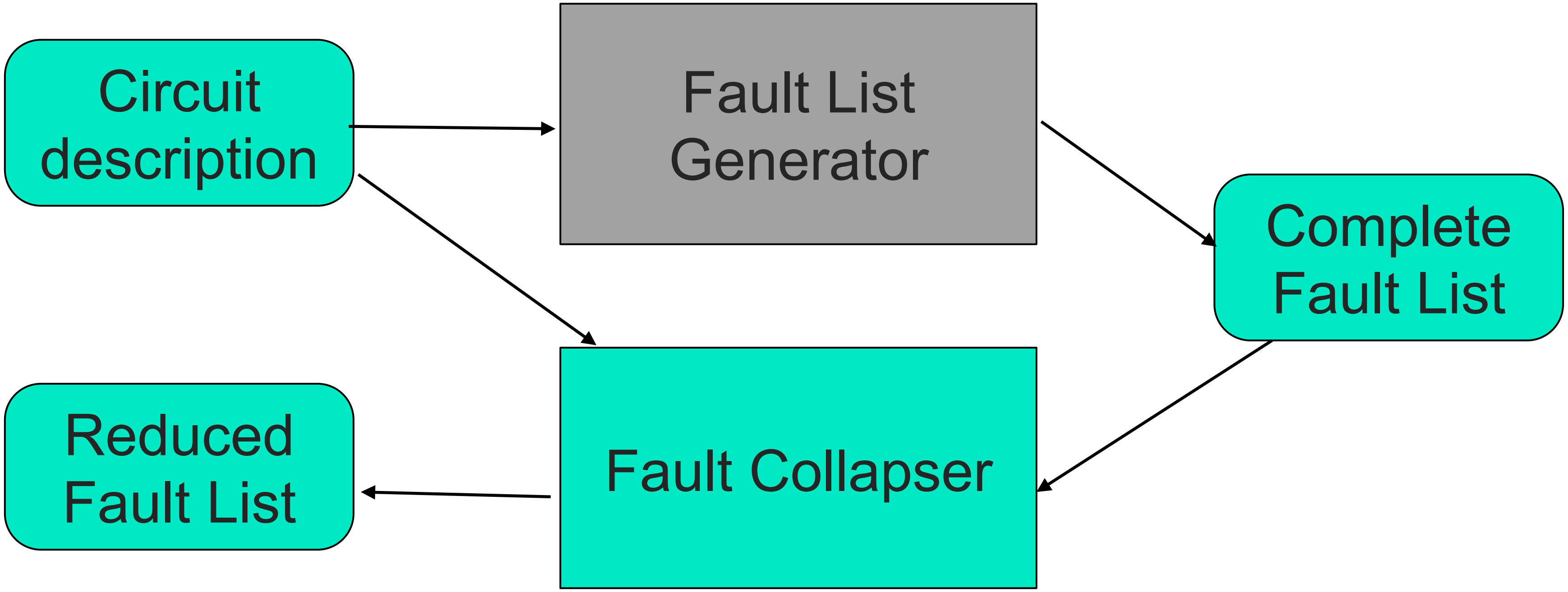
The test plan (cont'd)

- Step 1:
  - Identify the set of target faults (complete fault list).
- Step 2:
  - Identify the minimum set of distinct target faults (fault collapsing)

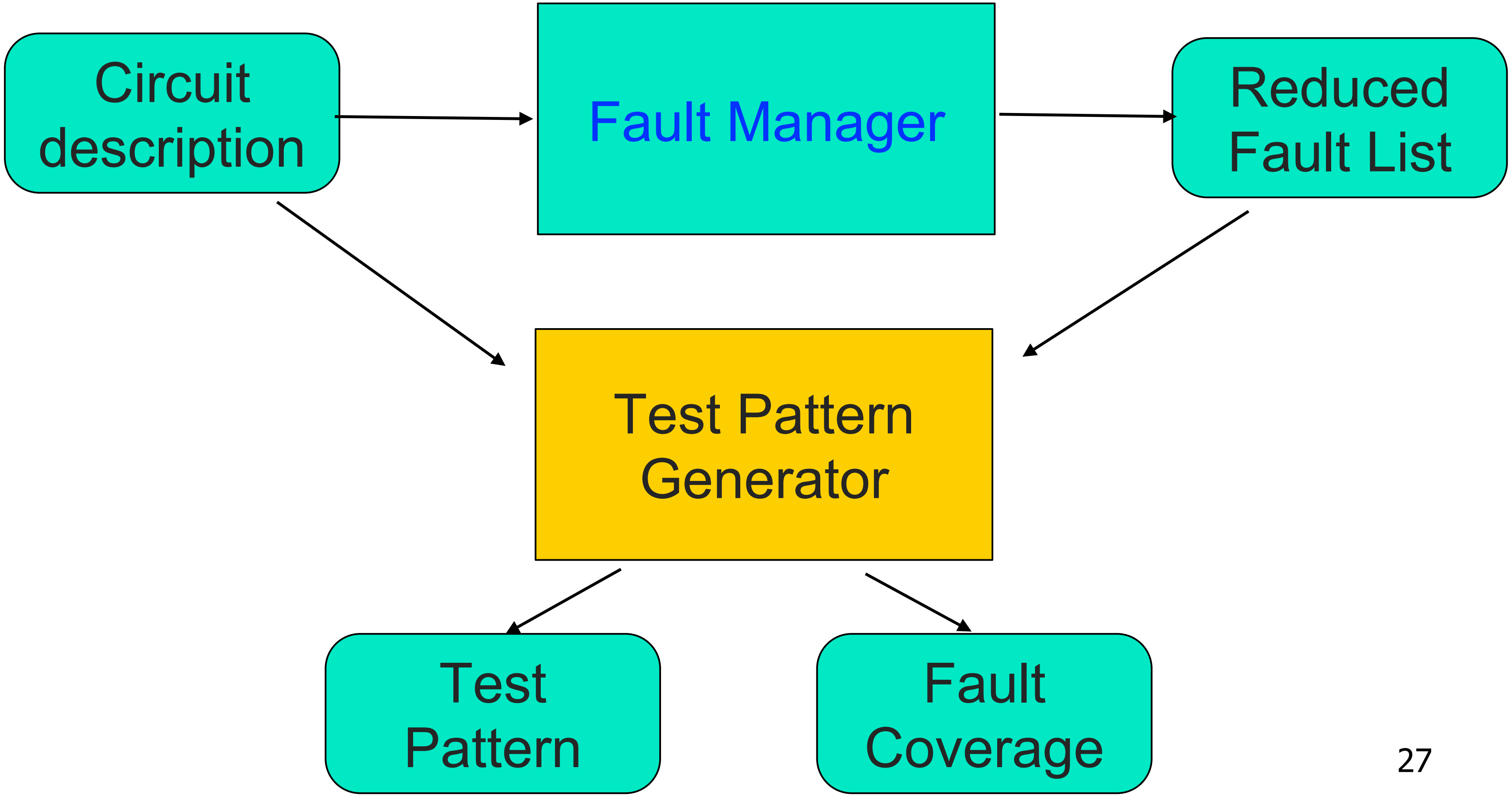
The test plan

- Step 2:
  - Tools – Fault collapser (One of the components of the Fault Manager).

Fault Manager



Fault Manager



The test plan (cont'd)

- Step 2:
  - Identify the minimum set of distinct target faults (fault collapsing)
- Step 3:
  - Generate, at no charge, an initial set of patterns (manually, from design validation, randomly, ...)

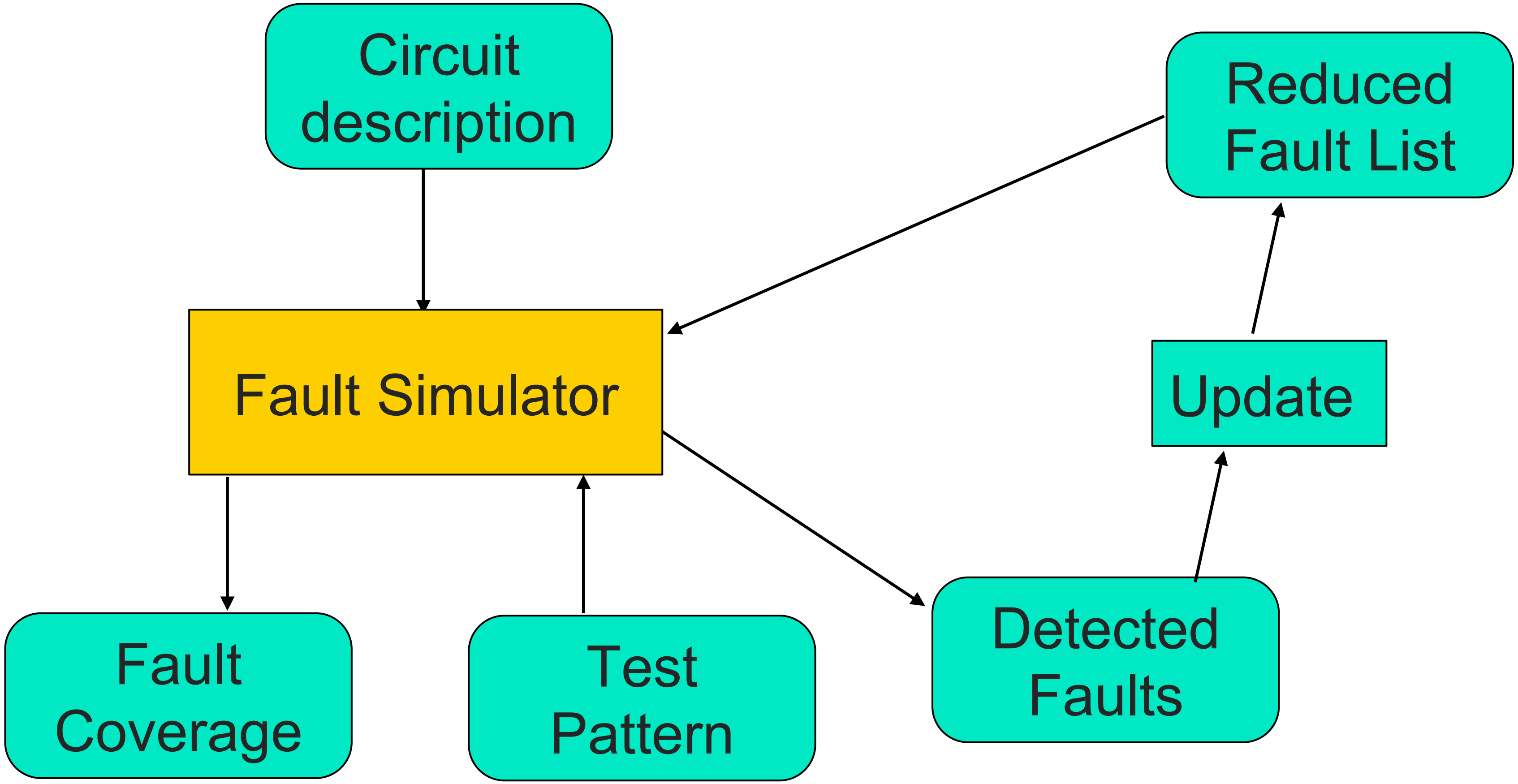
The test plan (cont'd)

- Step 3:
  - Generate, at no charge, an initial set of patterns (manually, from design validation, randomly, ...)
- Step 4:
  - Update the list of detected faults (fault simulation)

Step 4

- Tools
  - Fault Simulators: identify the set of faults covered by each test pattern.

Fault Simulator



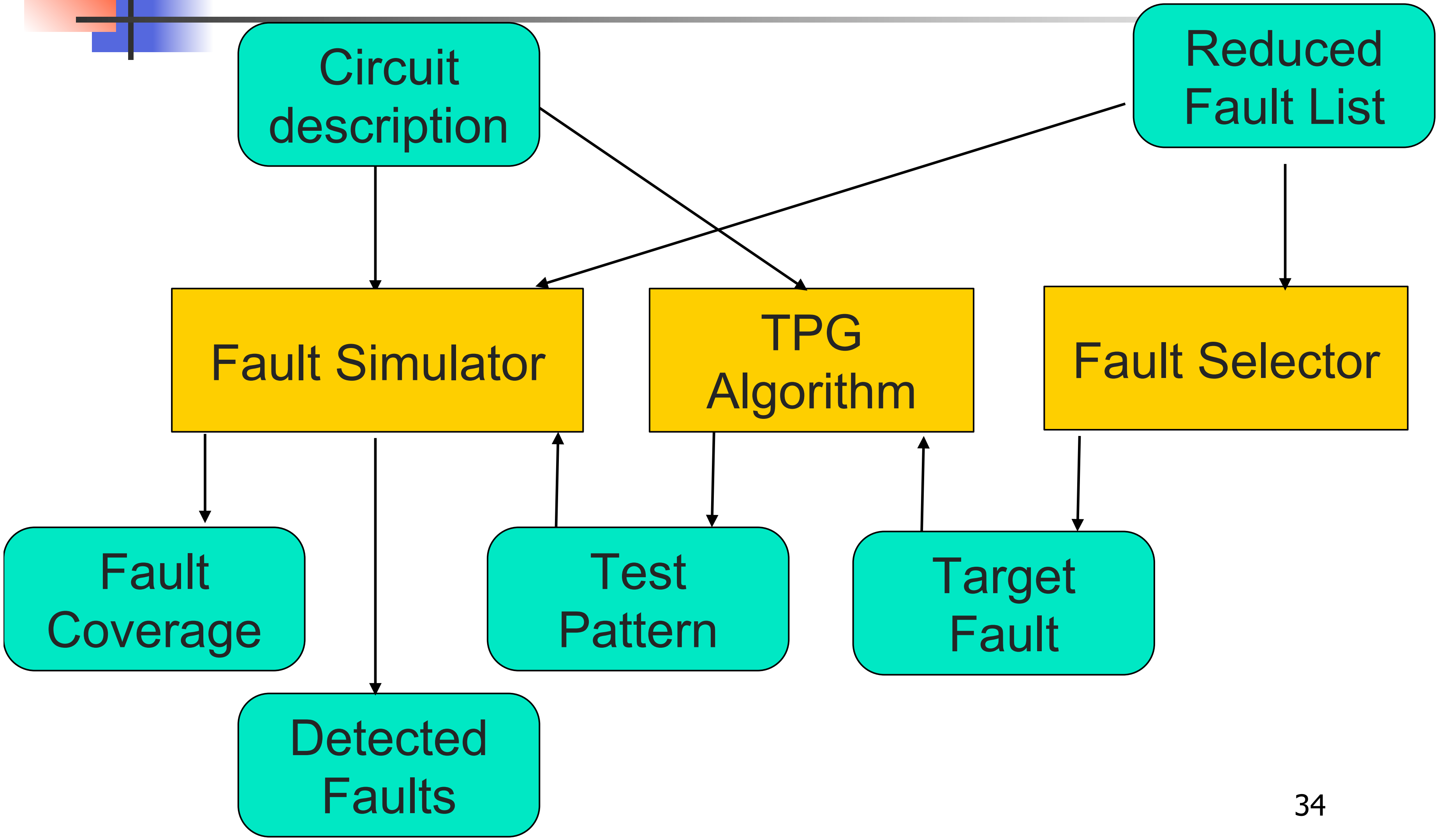
The test plan (cont'd)

- Step 4:
  - Update the list of detected faults (fault simulation)
- Step 5:
  - Generate a set of patterns to cover the uncovered faults (TPG)

Step 5

- Tools
    - ATPG: Automatic Test Pattern Generator

TPG


Circuit description
Reduced Fault List
Fault Simulator
TPG Algorithm
Fault Selector
Fault Coverage
Test Pattern
Target Fault
Detected Faults

- They cycle through three sub-phases:
  - target fault selection
  - Pattern generation
  - Covered fault list updating.

The test plan (cont’d)

- Step 5:
  - Generate a set of patterns to cover the uncovered faults (TPG)
- Step 6 (optional):
  - Testability analysis
- Step 7 (optional):
  - Compact test pattern set.

Step 6

- Goal
  - Estimate the effort needed to test the UUT:
    - Pattern length
    - Fault coverage
    - CPU time
    - ...
    - Identify hard-to-test areas
- Tools
  - Testability Analyzer
  - **Experience**.

The test plan (cont’d)

- Step 5:
  - Generate a set of patterns to cover the uncovered faults (TPG)
- Step 6 (optional):
  - Testability analysis
- Step 7 (optional):
  - Compact test pattern set.

Testability Analyzer

- High trade-off between result accuracy and CPU time.

- A Circuit is testable when you ATPG can manage it!!!!!

Fault Simulation*

- Problem and motivation
- Fault simulation algorithms
  - Serial
  - Parallel
  - Deductive

*The lecture has been taken from Prof. Agrawal VLSI test course (http://www.eng.auburn.edu/~agrawvd/COURSE/E7250_06/course.html)

Problem and Motivation

- Given
  - A circuit
  - A sequence of test vectors
  - A fault model
- Determine
  - Fault coverage - fraction (or percentage) of modeled faults detected by test vectors
  - Set of undetected faults

Problem and Motivation

- Motivation
  - Determine test quality and in turn product quality
  - Find undetected fault targets to improve tests

Fault Simulation Scenario

- Circuit model: mixed-level
  - Mostly logic with some switch-level for high-impedance (Z) and bidirectional signals
  - High-level models (memory, etc.) with pin faults
- Signal states: logic
  - Two (0, 1) or three (0, 1, X) states for purely Boolean logic circuits
  - Four states (0, 1, X, Z) for sequential MOS circuits
- Timing:
  - Zero-delay for combinational and synchronous circuits
  - Mostly unit-delay for circuits with feedback

Fault Simulation Scenario (Continued)

- Faults:
  - Mostly single stuck-at faults
  - Sometimes stuck-open, transition, and path-delay faults; analog circuit fault simulators are not yet in common use
  - Equivalence fault collapsing of single stuck-at faults
  - Fault-dropping -- a fault once detected is dropped from consideration as more vectors are simulated; fault-dropping may be suppressed for diagnosis
  - Fault sampling -- a random sample of faults is simulated when the circuit is large

Fault Simulation Algorithms

- Serial
- Parallel
- Deductive
- ....

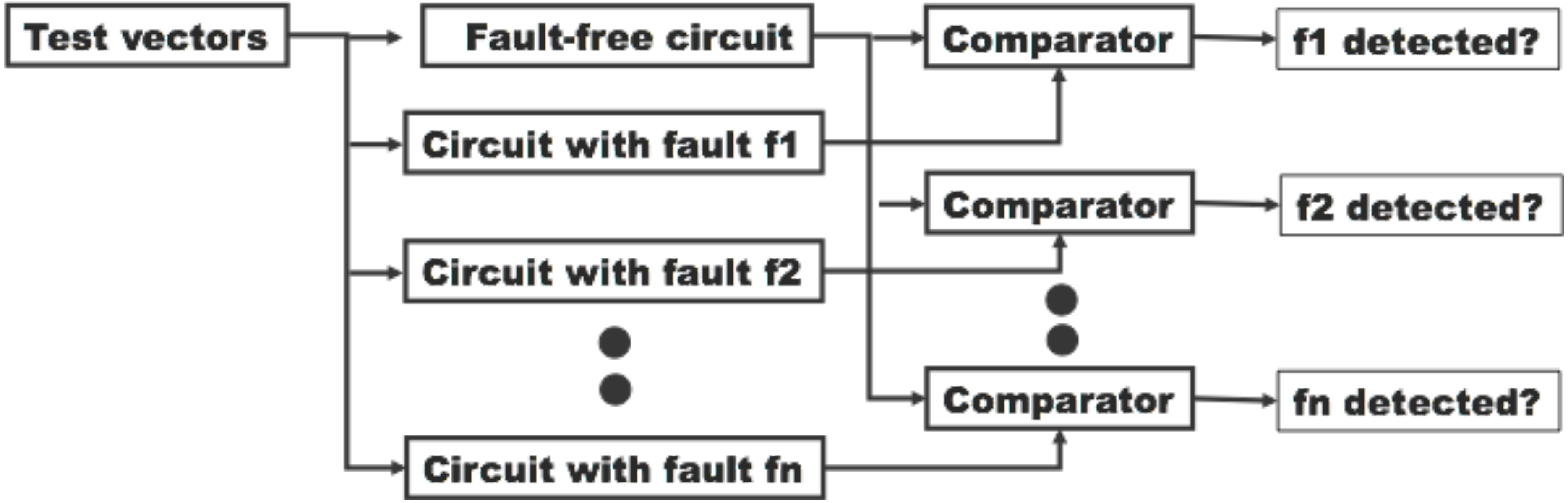
Serial Algorithm

- Algorithm: Simulate fault-free circuit and save responses. Repeat following steps for each fault in the fault list:
  - Modify netlist by injecting one fault
  - Simulate modified netlist, vector by vector, comparing responses with saved responses
  - If response differs, report fault detection and suspend simulation of remaining vectors
- Advantages:
  - Easy to implement; needs only a true-value simulator, less memory
  - Most faults, including analog faults, can be simulated

Serial Algorithm

- Disadvantage: Much repeated computation; CPU time prohibitive for VLSI circuits
- Alternative: Simulate many faults together

Serial Algorithm



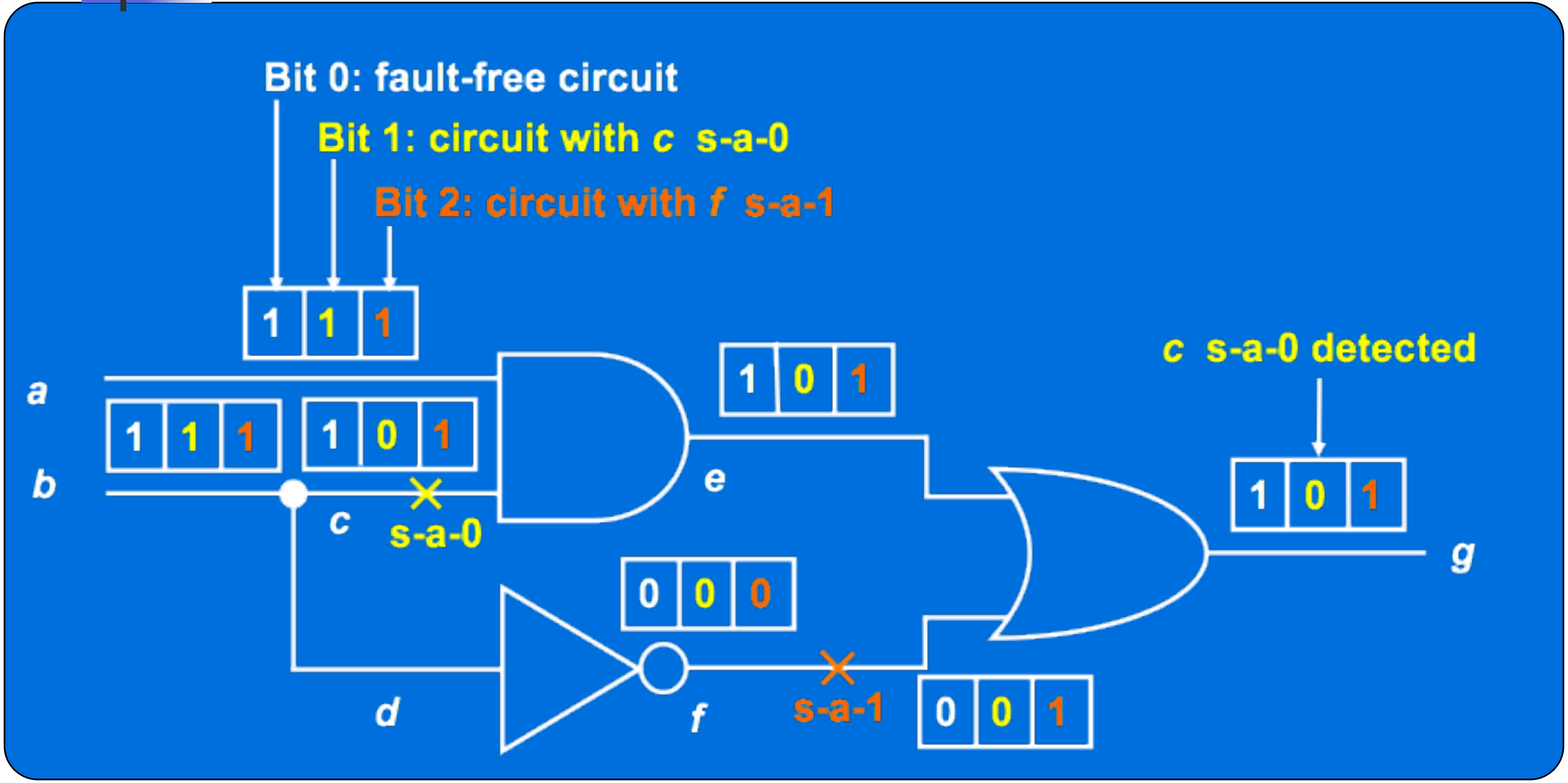
Serial algorithm

- + very simple
- - not efficient
  - Intel I7 is about ~10M gates
  - 20M faults, 1 simulation = 1s
  - 20Ms ~= **231 days**

Parallel Fault Simulation

- Compiled-code method; best with two-states (0,1)
- Exploits inherent bit-parallelism of logic operations on computer words
- Storage: one word per line for two-state simulation
- Multi-pass simulation: Each pass simulates w-1 new faults, where w is the machine word length
- Speed up over serial method ~ w-1
- Not suitable for circuits with timing-critical and non-Boolean logic

Parallel Fault Simulation


Bit 0: fault-free circuit
Bit 1: circuit with *c* s-a-0
Bit 2: circuit with *f* s-a-1
1 1 1
a
1 1 1
1 0 1
b
c
s-a-0
1 0 1
e
0 0 0
d
f
s-a-1
0 0 1
c s-a-0 detected
1 0 1
g

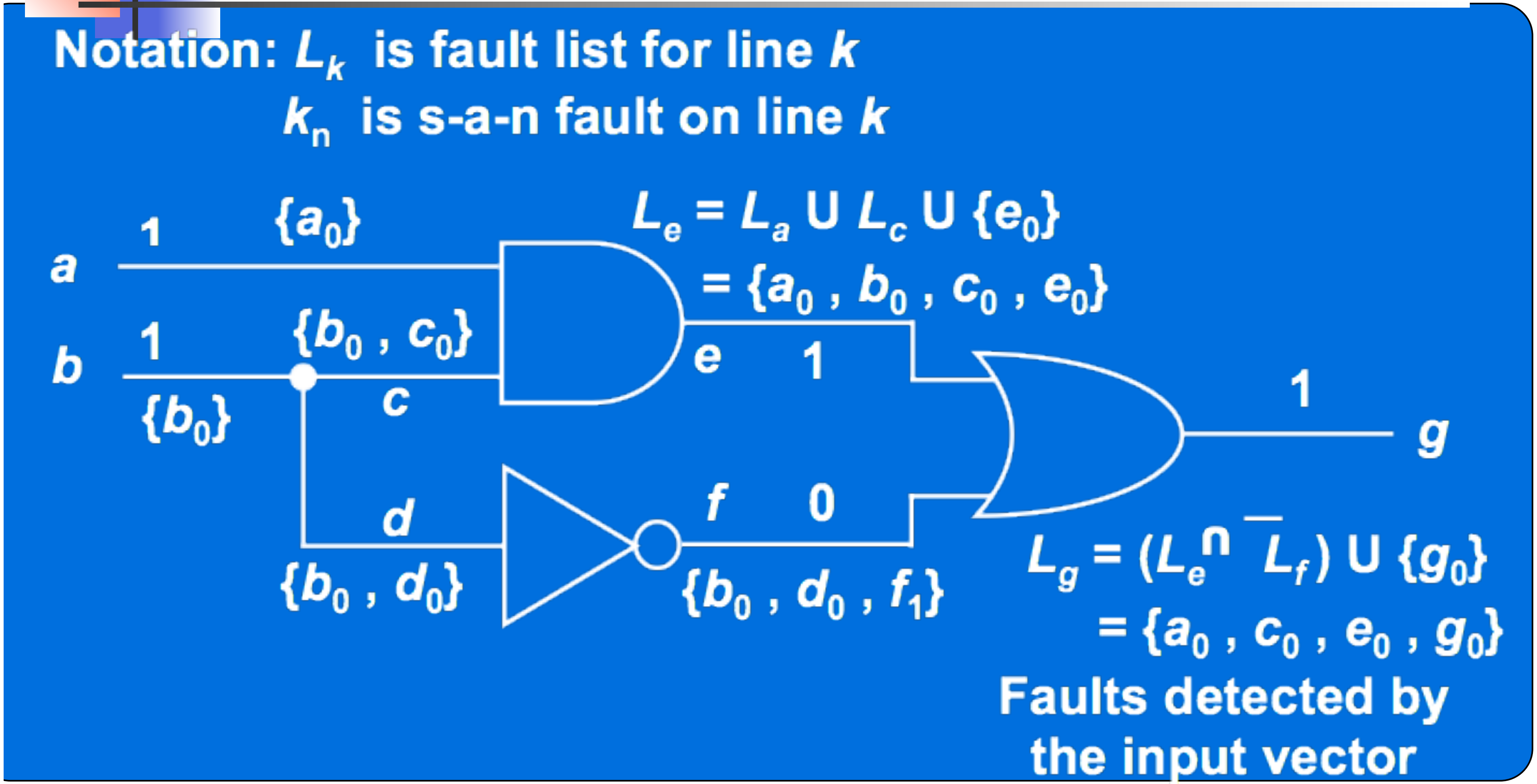
Parallel algorithm

- + still very simple
- + more efficient than serial
  - Intel I7 is about ~10M gates
  - 20M faults, 1 simulation = 1s
  - 20Ms ~= **231 days**
  - Using a 64bits machine
  - **231/63 ~= 4 days**

Deductive Fault Simulation

- One-pass simulation
- Each line k contains a list L_k of faults detectable on it
- Following true-value simulation of each vector, fault lists of all gate output lines are updated using set-theoretic rules, signal values, and gate input fault lists
- PO fault lists provide detection data
- Limitations:
  - Set-theoretic rules difficult to derive for non-Boolean gates
  - Gate delays are difficult to use

Deductive Fault Simulation



Notation: L_k is fault list for line k

k_n is s-a-n fault on line k

$$L_e = L_a \cup L_c \cup \{e_0\}$$
$$= \{a_0, b_0, c_0, e_0\}$$

$$L_g = (L_e \cap \overline{L_f}) \cup \{g_0\}$$
$$= \{a_0, c_0, e_0, g_0\}$$

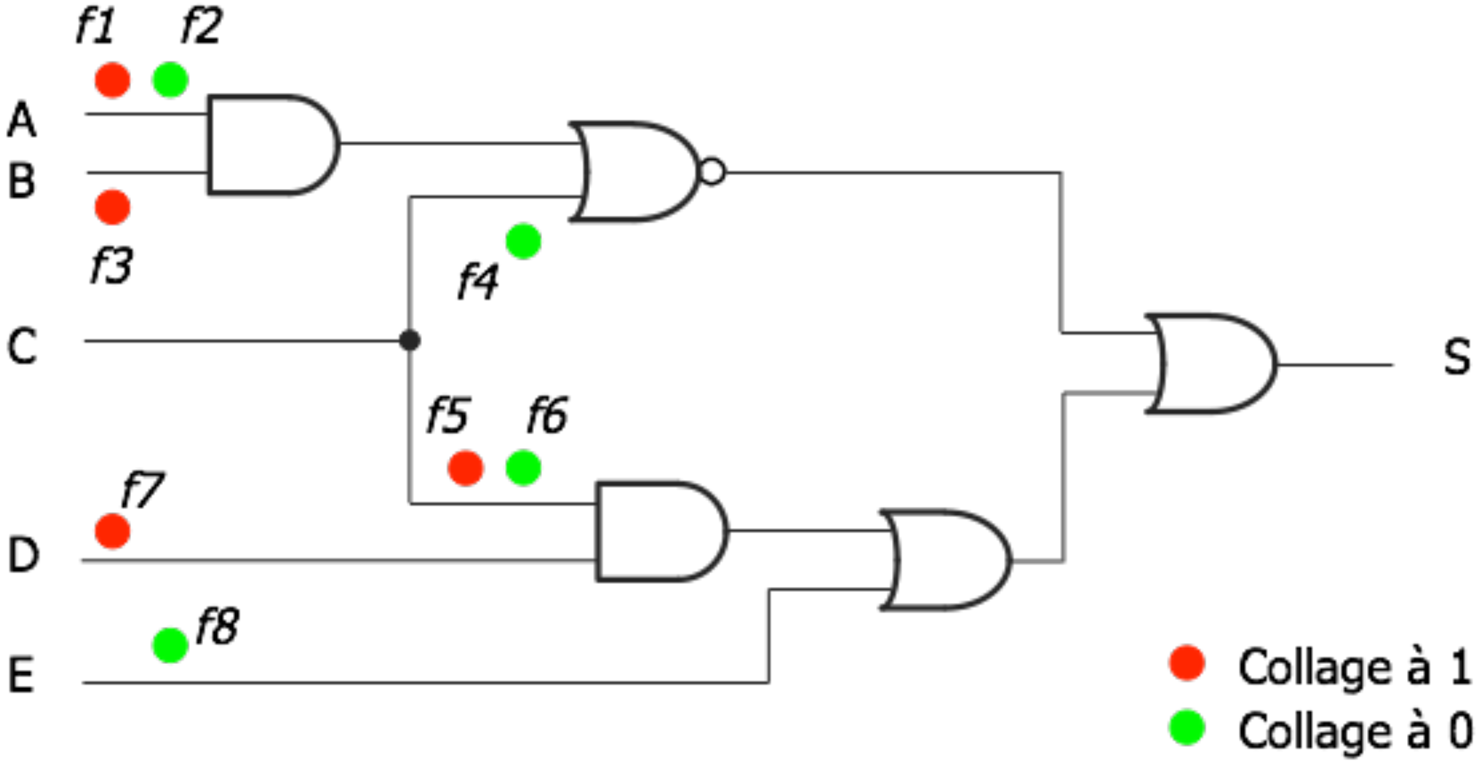
Faults detected by the input vector

Deductive algorithm

- - complex
- ++ more efficient than parallel
  - Intel I7 is about ~10M gates
  - 20M faults, 1 simulation = 1s
  - 20Ms ~= **1s**
- - it requires a lot of memory

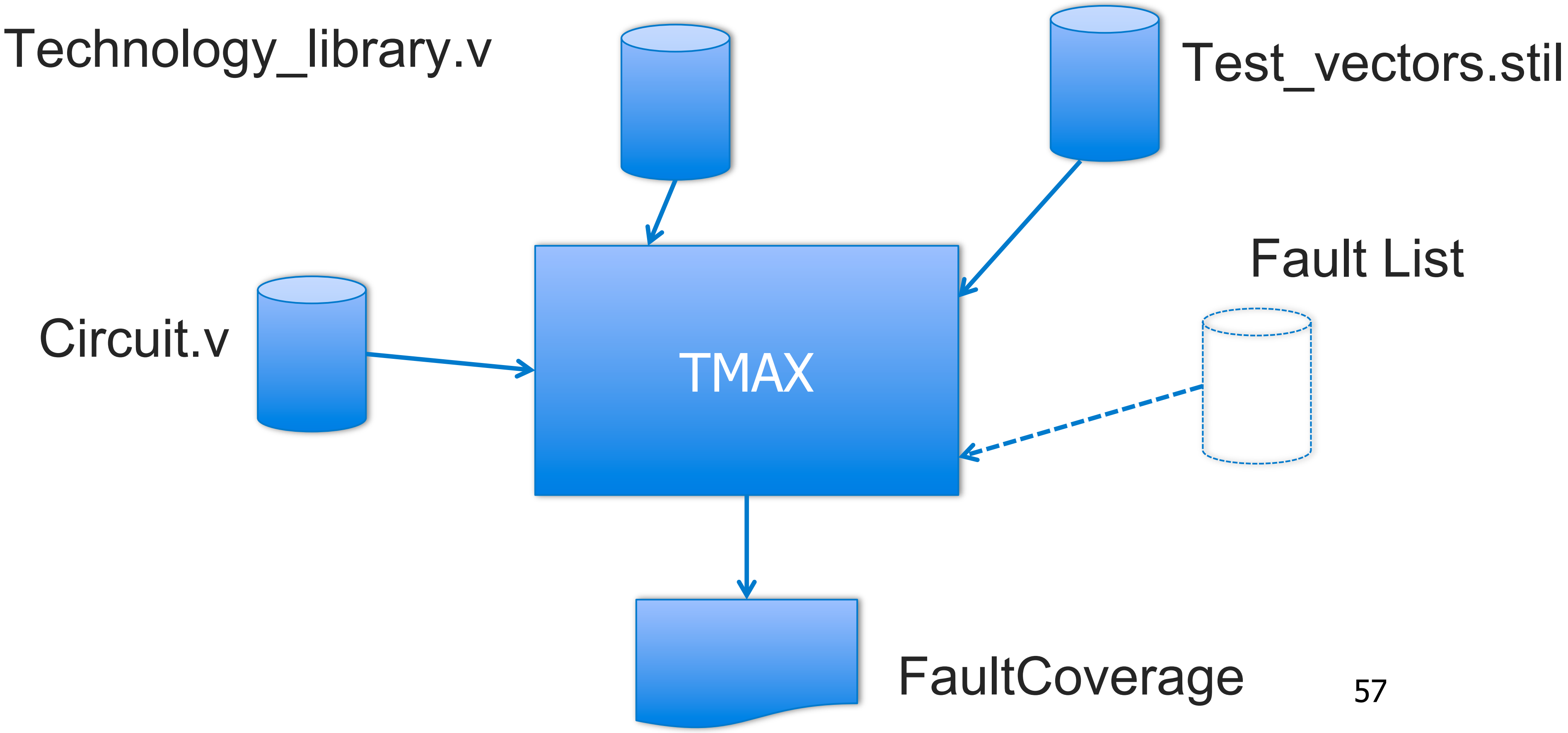
Exercice

which faults are detected by the input “01110”?



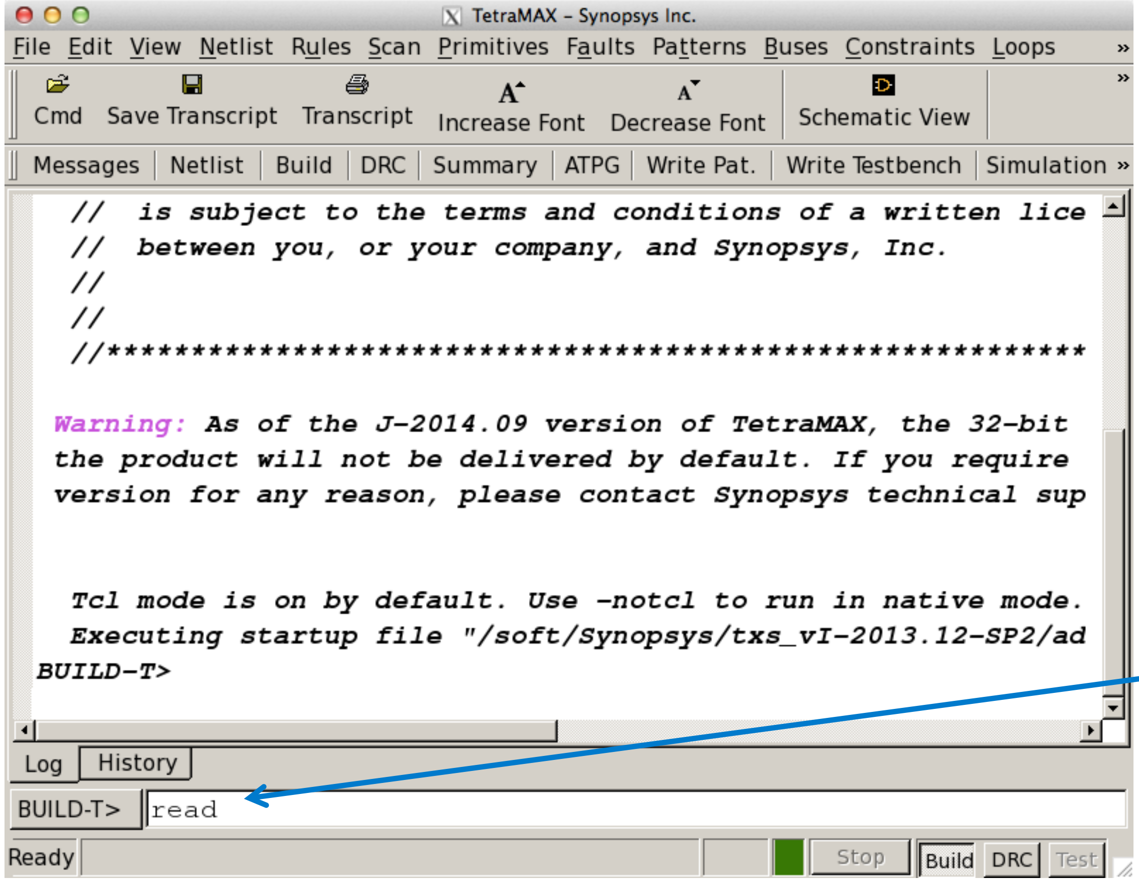
In practice

- Use of Synopsys© Tetramax



Invoking TetraMax

- source /soft/Synopsys/source_config/.config_tetramax_standalone_vI-2013.12
- tmax



TetraMAX - Synopsys Inc.
File Edit View Netlist Rules Scan Primitives Faults Patterns Buses Constraints Loops »
Cmd Save Transcript Transcript Increase Font Decrease Font Schematic View
Messages Netlist Build DRC Summary ATPG Write Pat. Write Testbench Simulation »
Log History
BUILD-T>
Ready Stop Build DRC Test

```
  //  is subject to the terms and conditions of a written lice
  //  between you, or your company, and Synopsys, Inc.
  //
  //
  //**********************************************************

 Warning: As of the J-2014.09 version of TetraMAX, the 32-bit
 the product will not be delivered by default. If you require
 version for any reason, please contact Synopsys technical sup


  Tcl mode is on by default. Use -notcl to run in native mode.
  Executing startup file "/soft/Synopsys/txs_vI-2013.12-SP2/ad
BUILD-T>
```

```
BUILD-T> read
```

You can enter commands

Step1

- Read and Compile the circuit description

```
read_verilog C35.v –library
read_verilog exo1.v
run_build_model
Run_drc
```

Step 2

- Generate the fault list

```
set_faults -model stuck
add_faults -all
```

Step 3

- Specify the test vectors to be simulated
  - We have to use the stil syntax
  - Look in the example

```
Pattern "_pattern_" {
   W "_default_WFT_";
   "precondition all Signals": C
{ "_pi"=0000; "_po"=XX; }

   "pattern 0": Call "capture" {
      "_pi"=1010; "_po”=LL; }
}
```

Step 3

- Import the test vector file

```
set_patterns -external example_exo1.stil
```

- Now we can run a simulation

```
run_simulation
```

- You will got errors:

```
TEST-T> run_simulation
 Begin good simulation of 1 external
patterns.
    0  S2  (exp=0, got=1)
 Simulation completed: #patterns=1,
#fail_pats=1(0), #failing_meas=1(0),
CPU time=0.00
```

Step 3

- Tmax has to calculate the gold outputs before running the fault simulation

```
run_simulation -override_differences
```

- Now you can run the fault simulation

```
run_fault_sim
```