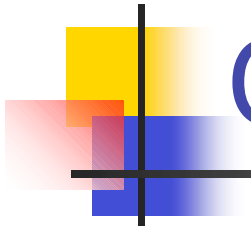
ATPG and Fault Simulation

Marie-Lise Flottes (flottes@lirmm.fr)

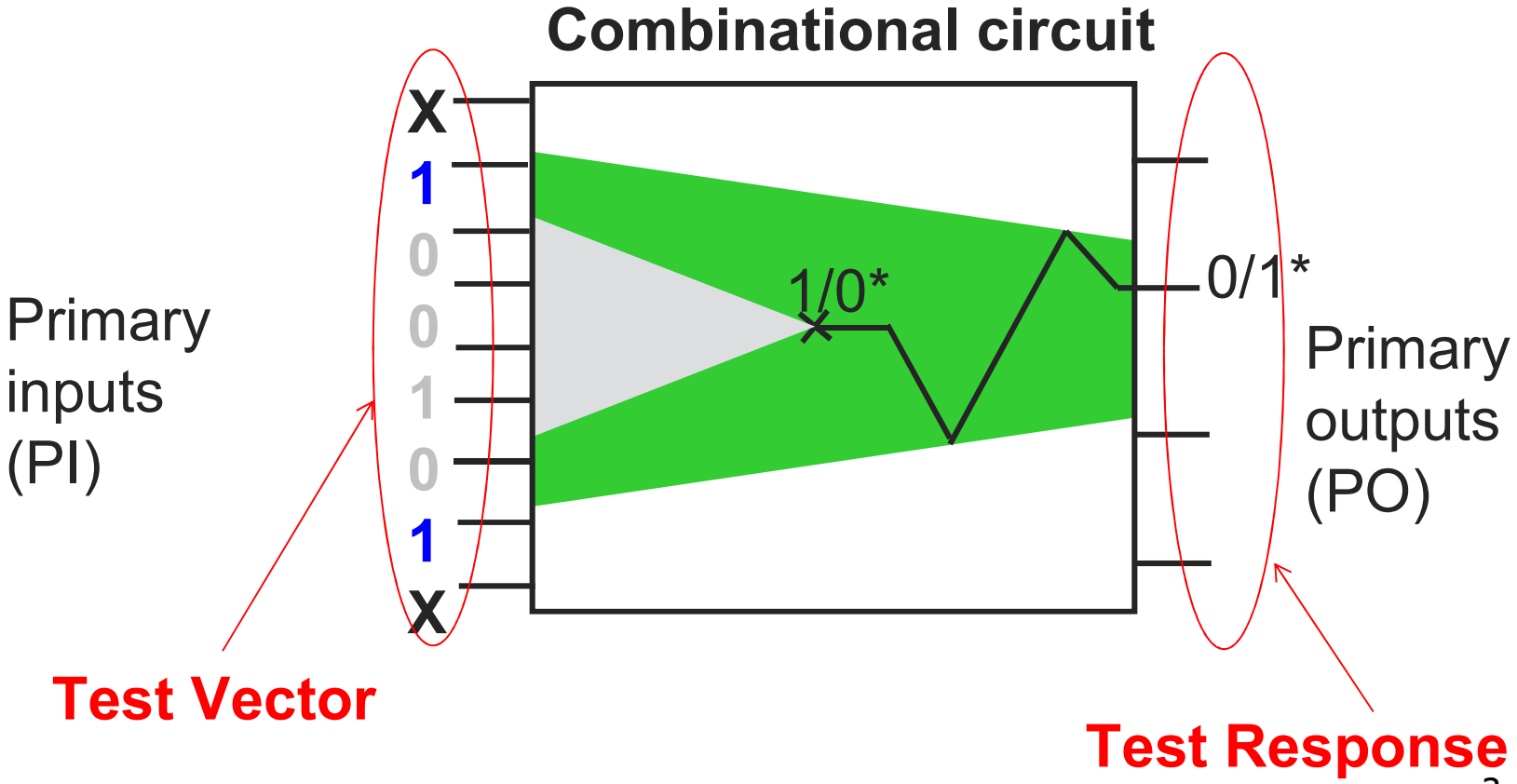


Outline

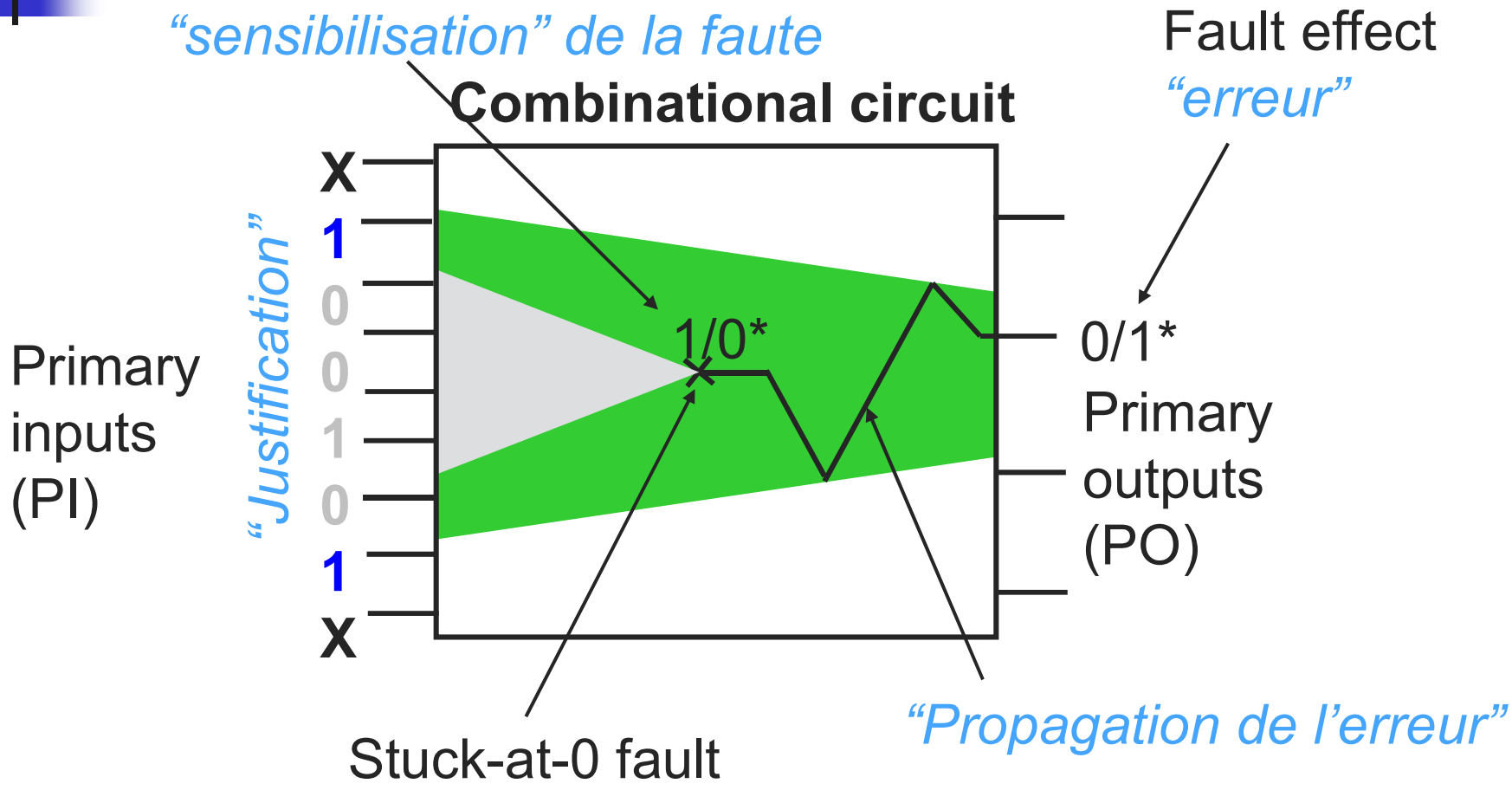
- Introduction (reminder)
- ATPG: process and tools
- Simulation

What is a test?

- *"Essai partiel de fonctionnement"*

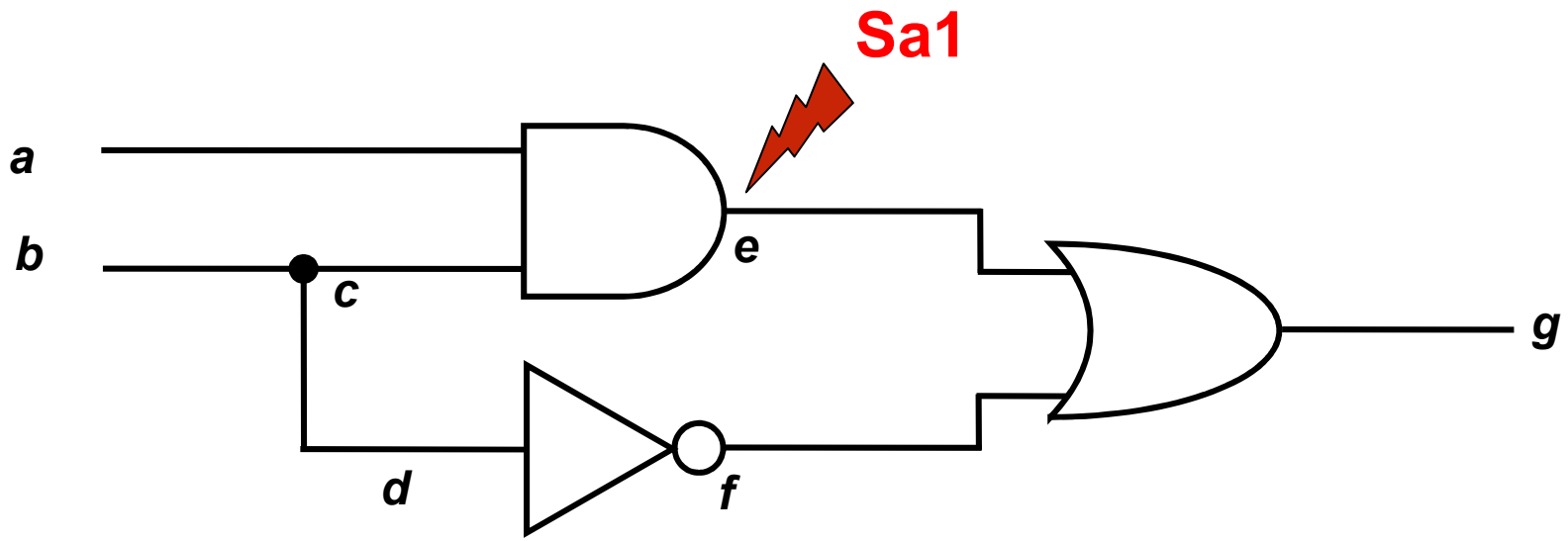


What is a test generation?



Example

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

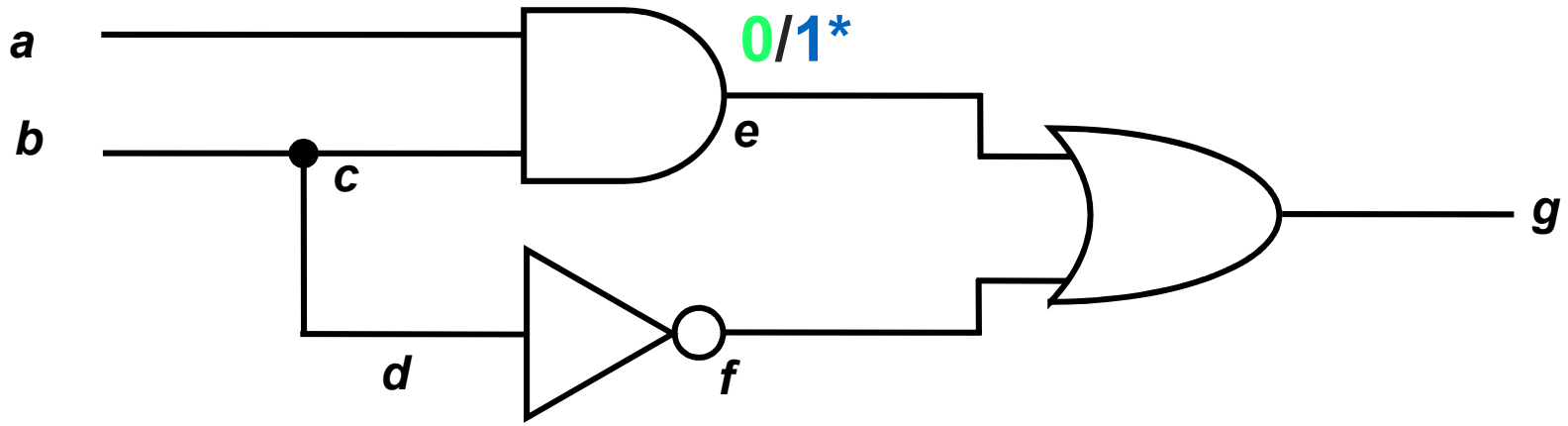


Example

- 1) Activate the fault (*"sensibilisation"*)

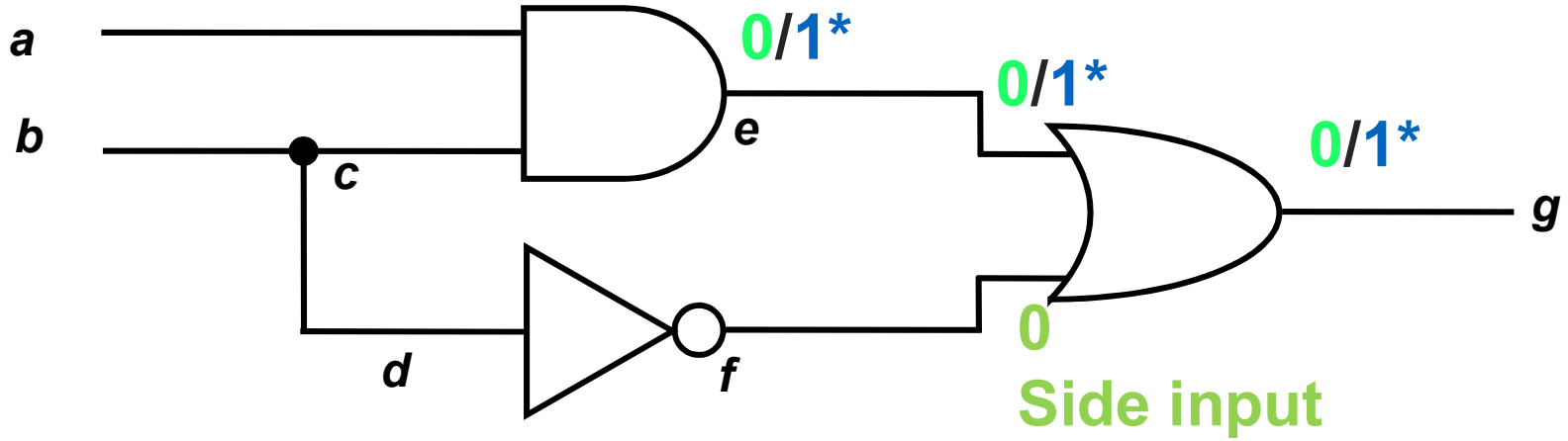
0 valeur que l'on cherche à assigner sur e

1* valeur réelle de e si Stuck-at-1/e (fault effect)



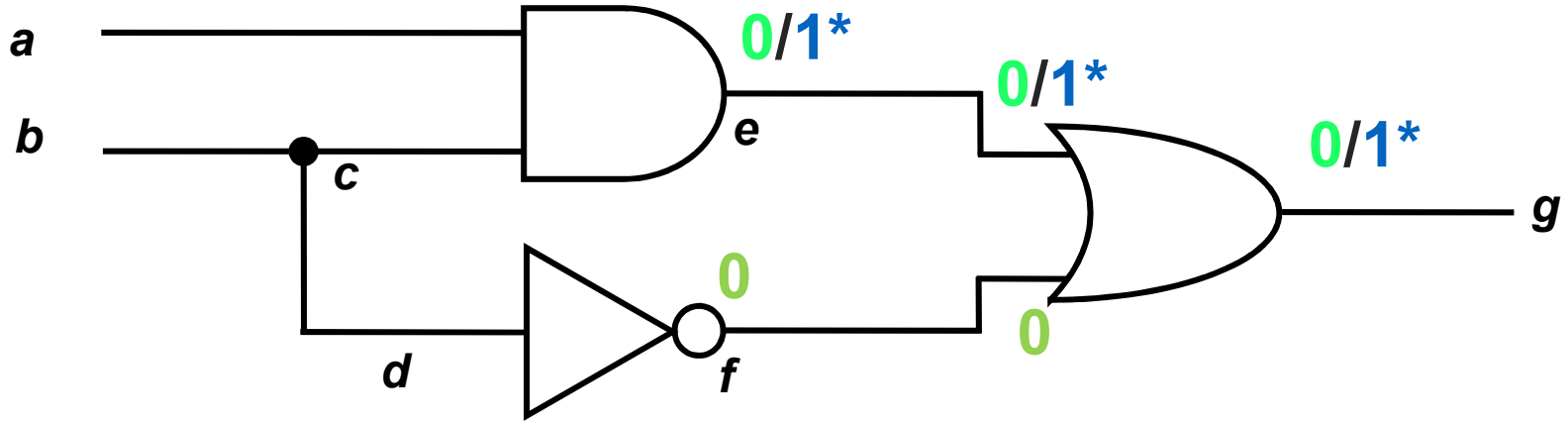
Example

- 2) Propagate the fault effect *("propagation")*



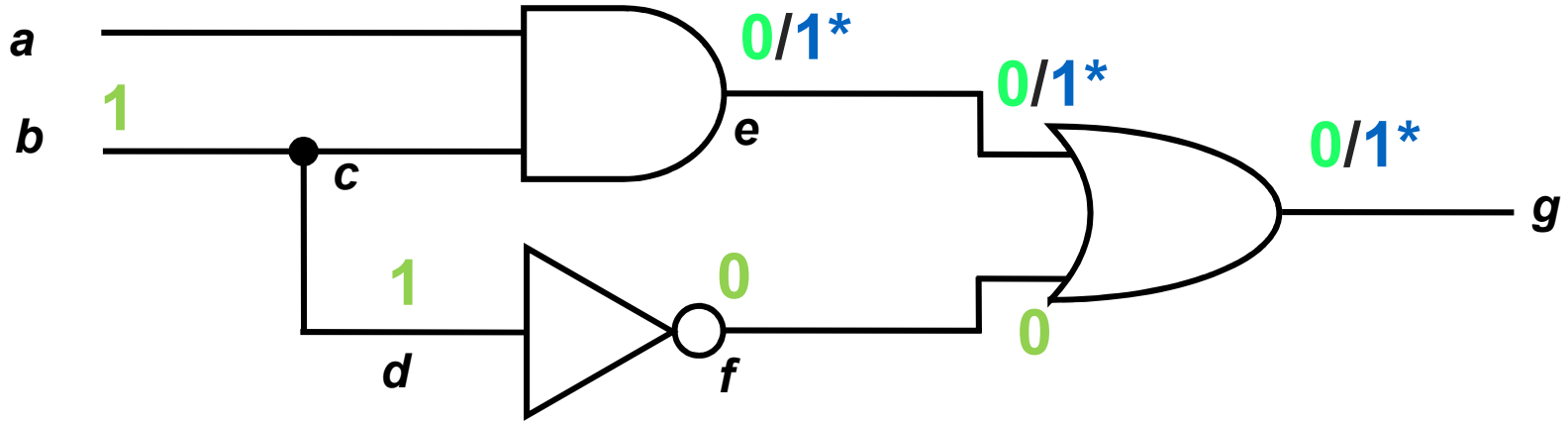
Example

- 3) Justify expected values on side-inputs and inputs of the gate under test *("justification")*

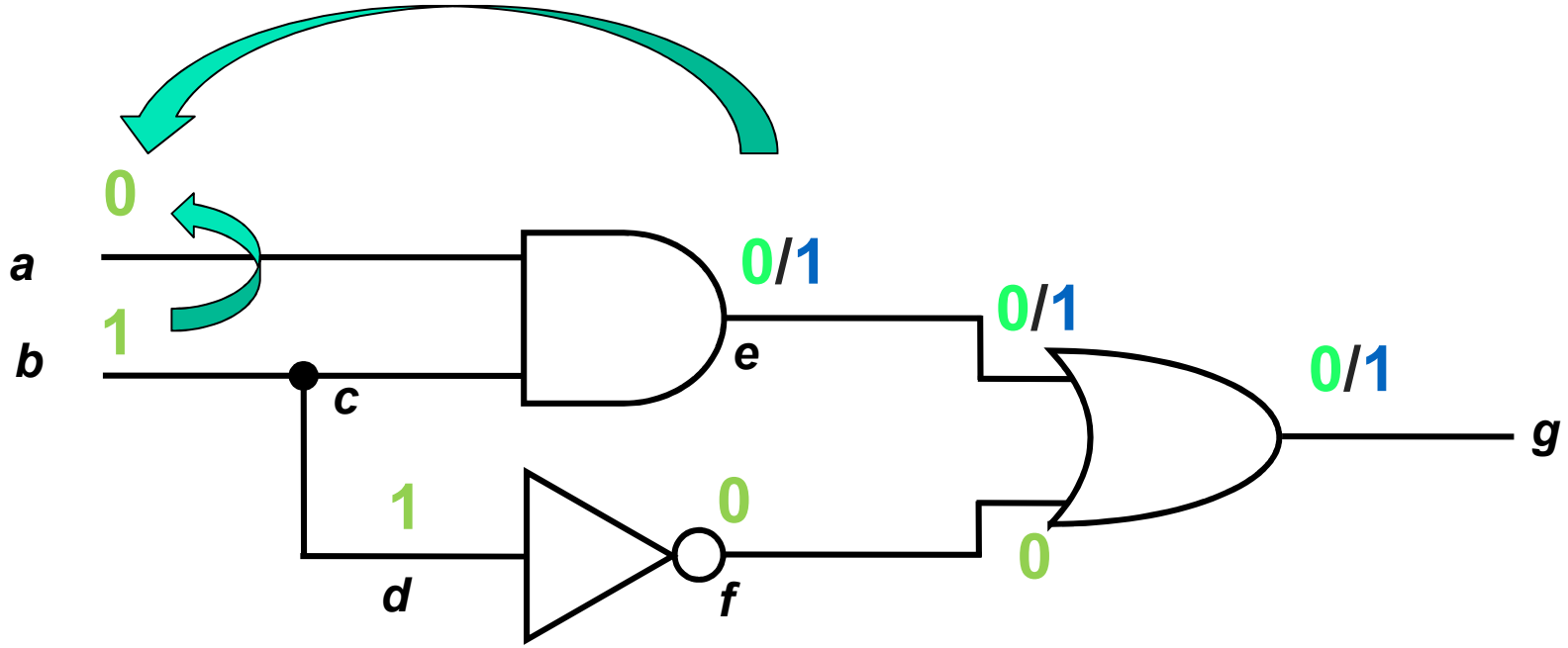


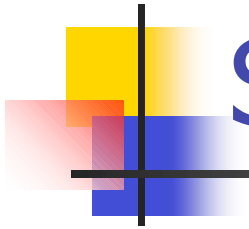
Example

- 3) Justify expected values on side-inputs and inputs of the gate under test *("justification")*



Example



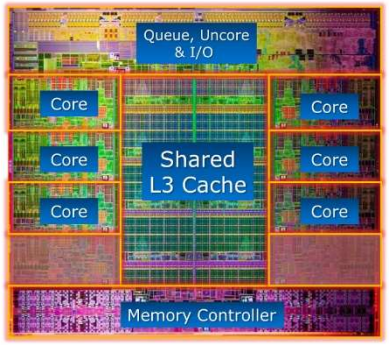


Some Considerations

- Test generation on Combinational circuit is easy
- But….

Some problems (the complexity)

Intel® Core™ i7-3960X Processor Die Detail

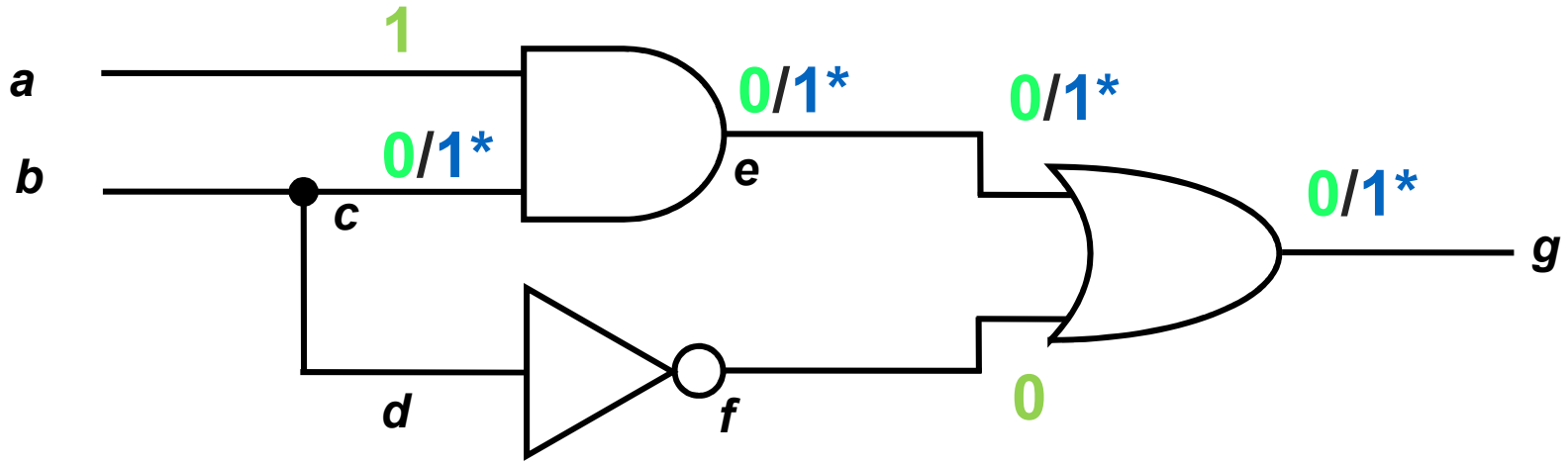


2011 : 2.2 Billion Transistors

2022 : sur 1 mm², Intel intègre environ 80 millions de *transistors* avec le procédé Intel 7(10nm) et en gravera 160 millions avec Intel 4 (7nm), trillion de transistors avant 2030.

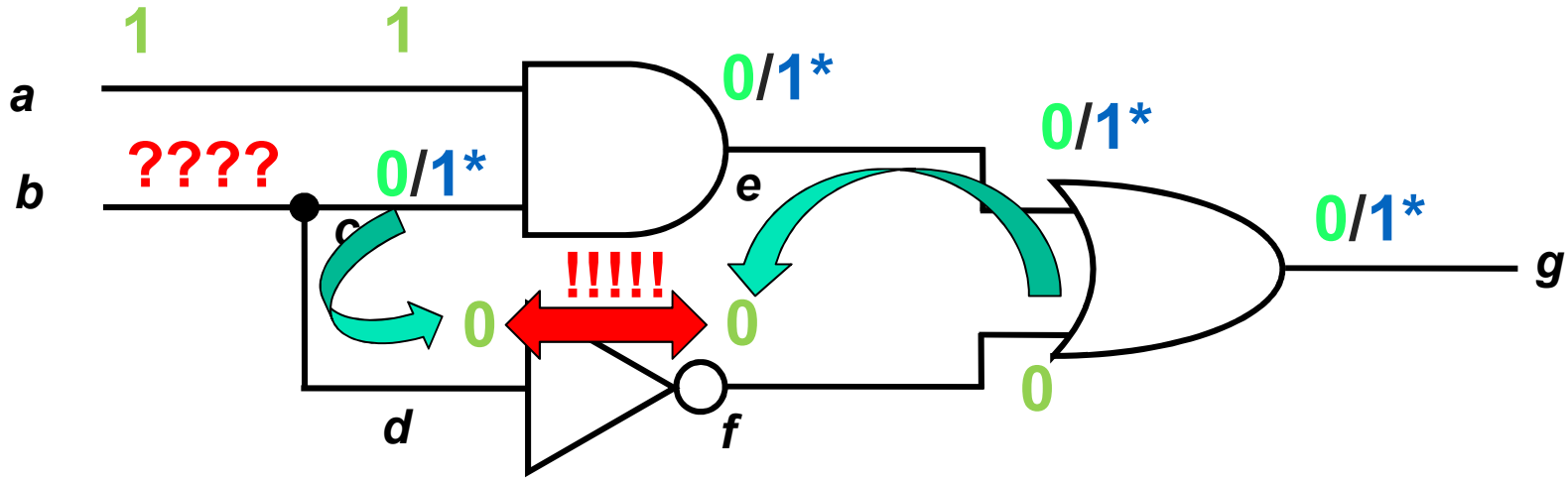
Some problems (the circuit)

- Redundant faults
  - E.g. Generate a test for **c stuck-at-1**
  - *Sensibilisation + propagation*

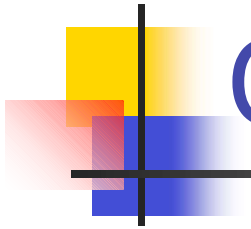


Some problems (the circuit)

- Redundant faults
  - E.g. Generate a test for **c stuck-at-1**
  - *Justification (no solution)*

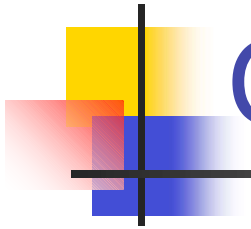


- **c stuck-at-1** is an untestable fault



Outline

- Introduction (reminder)
- ATPG: process and tools
- Simulation

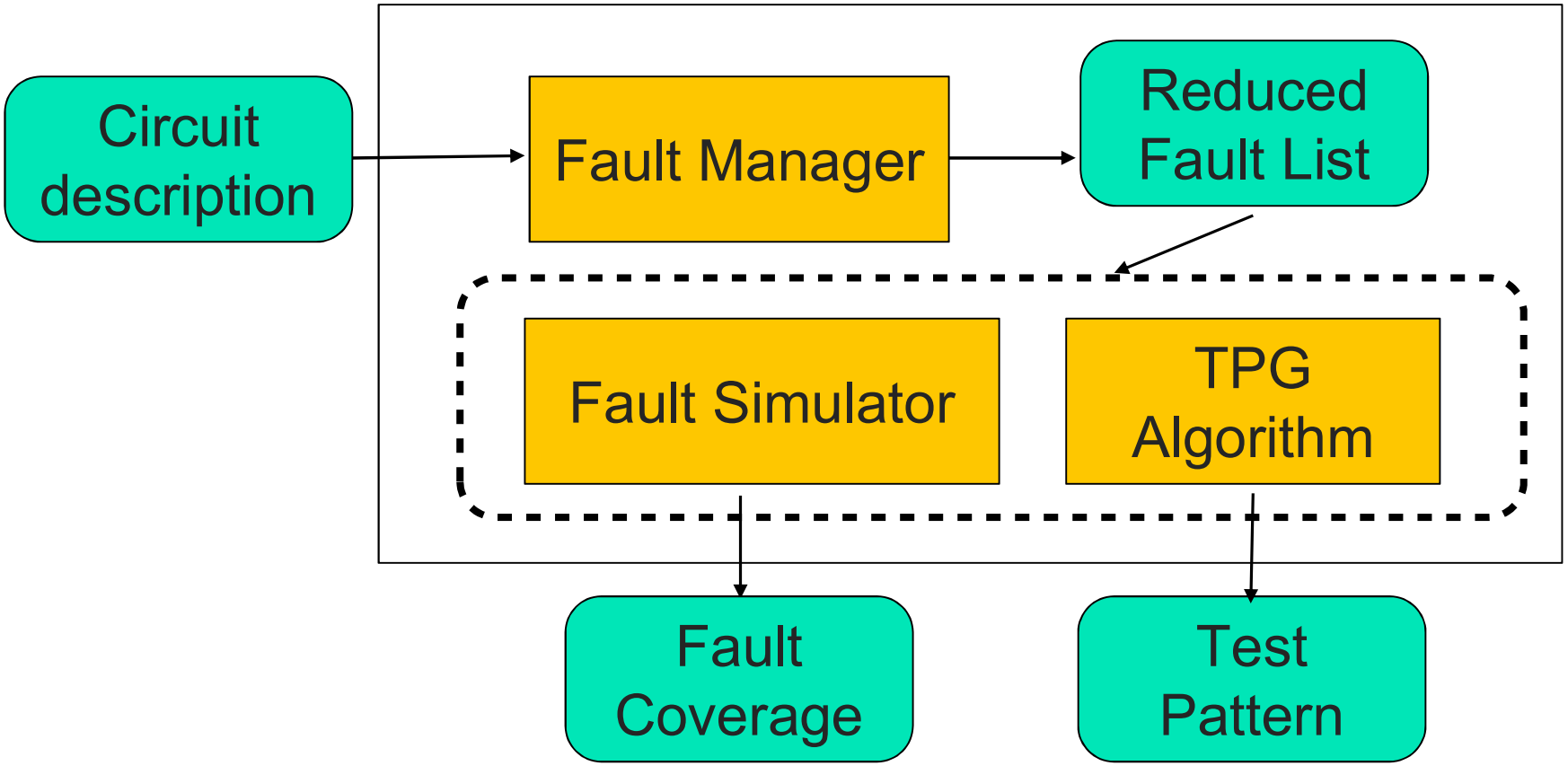


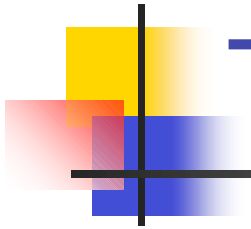
Goals

- Generate as quickly as possible the shortest test sequence for a fault coverage (FC%) target
- Reminder : FC = % # Detected Faults / #Total Faults

- Industrial tool
  - Automatic Test Pattern Generator (ATPG)

ATPG Architecture





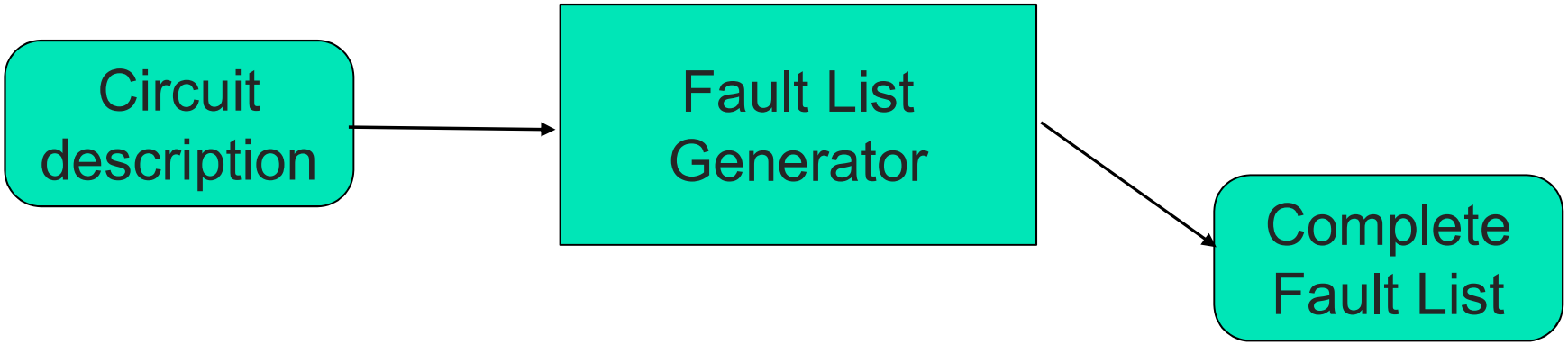
The test plan

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

Fault Manager

The test plan

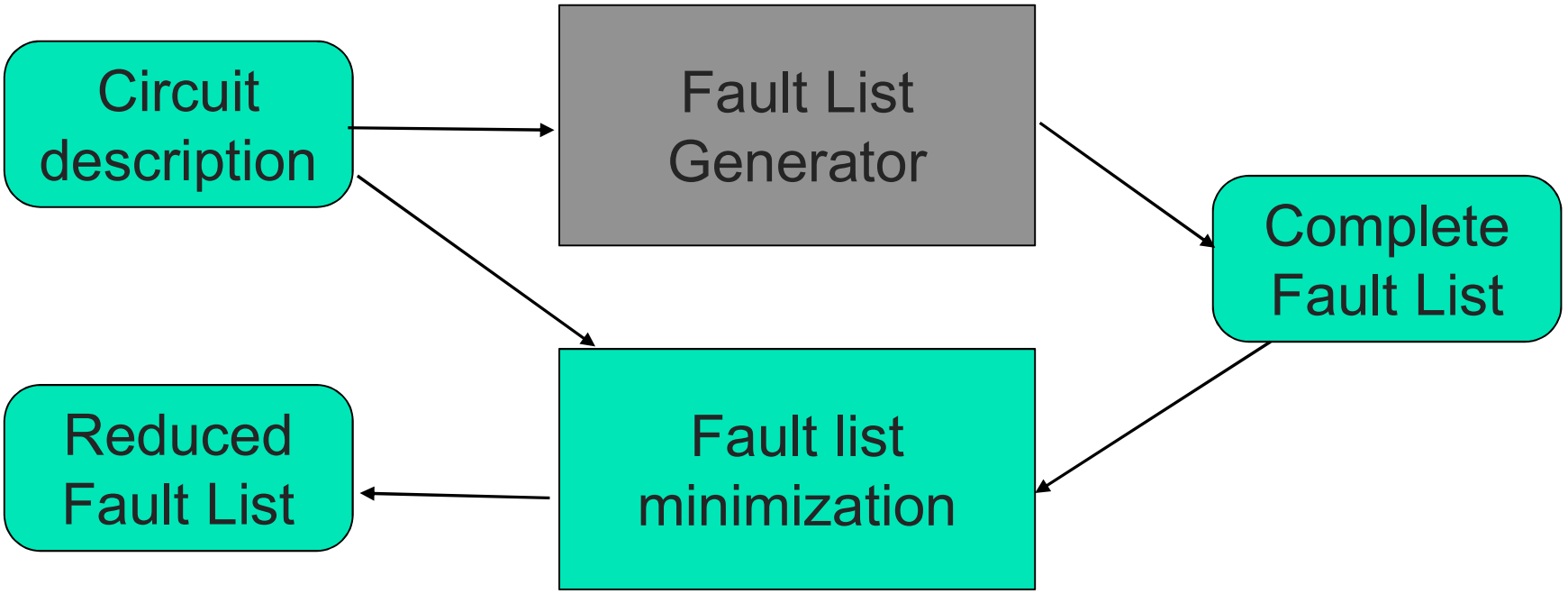
- Step 1: Fault Manager
  - Tools – Fault list generator



The test plan (cont'd)

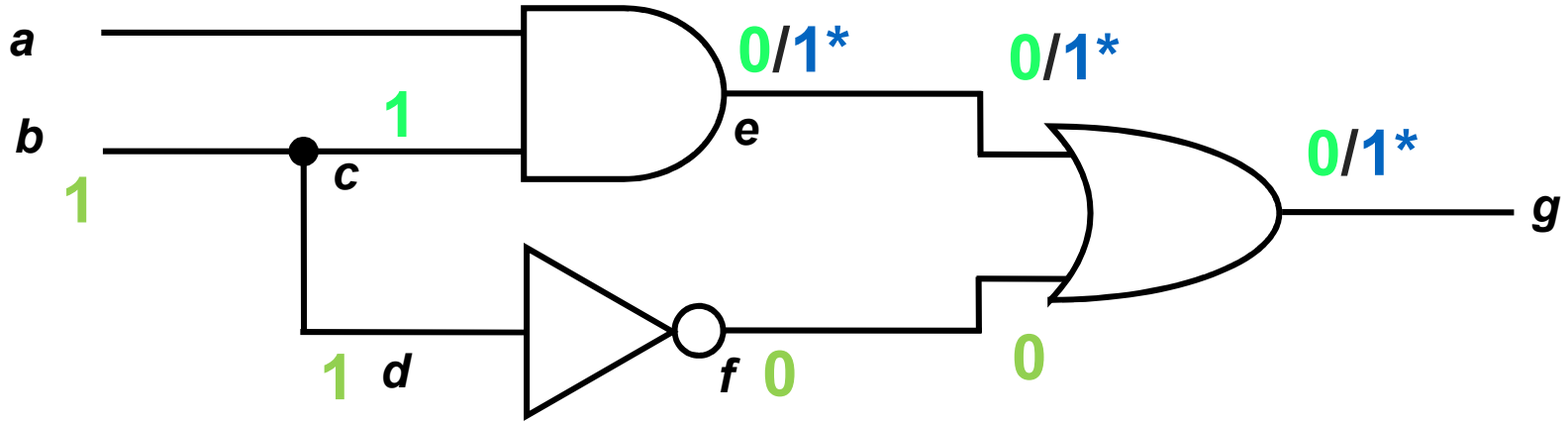
- Step 1:
  - Identify the set of target faults (complete fault list).
- Step 2: Fault Manager
  - Identify the minimum set of distinct faults to target (reduced fault list thanks to equivalence/implication rules)

The test plan (cont'd)



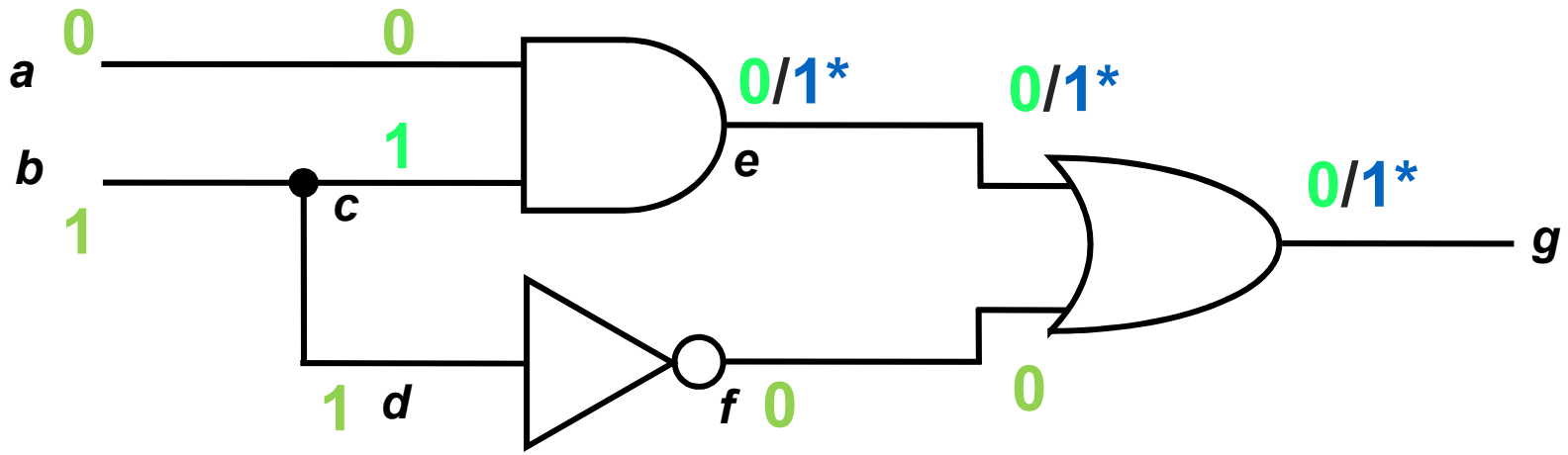
Fault List Minimisation e.g.

- Equivalent faults
  - **e stuck-at-1**



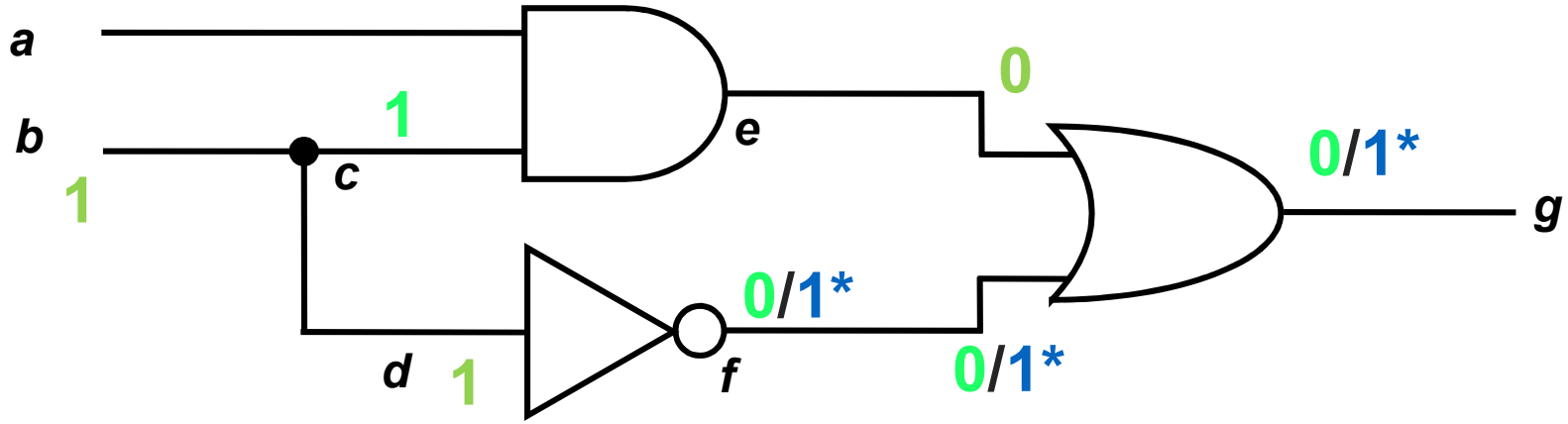
Fault List Minimisation e.g.

- Equivalent faults
  - **e stuck-at-1**
  - **Test vector (a,b) = (0,1)**



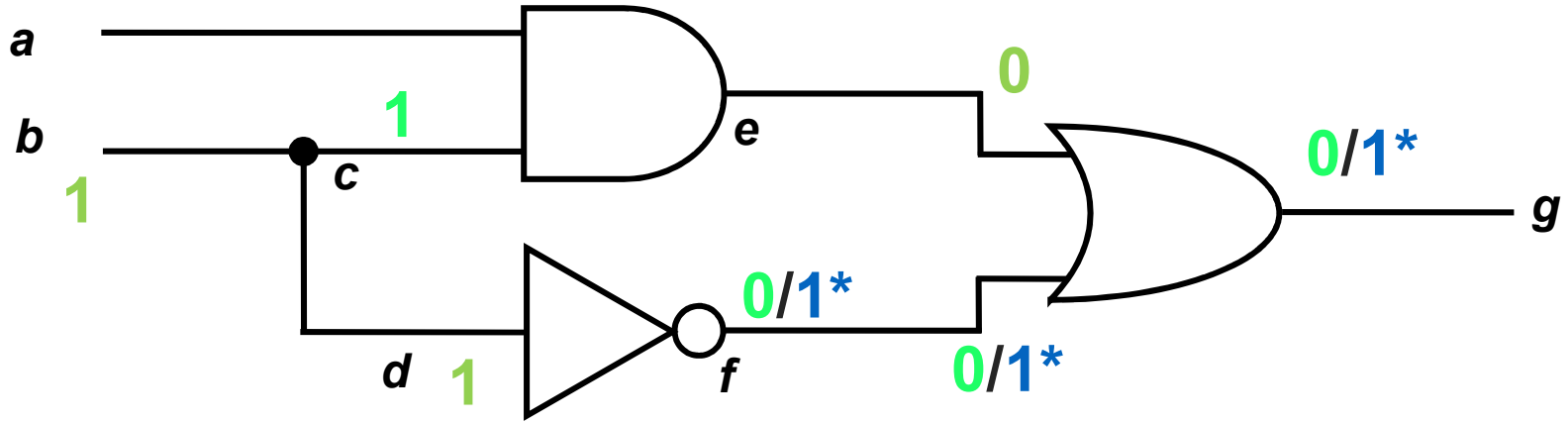
Fault List Minimisation e.g.

- Equivalent faults
  - **e stuck-at-1 or f stuck-at-1**
  - **Test vector (a,b) = (0,1)**



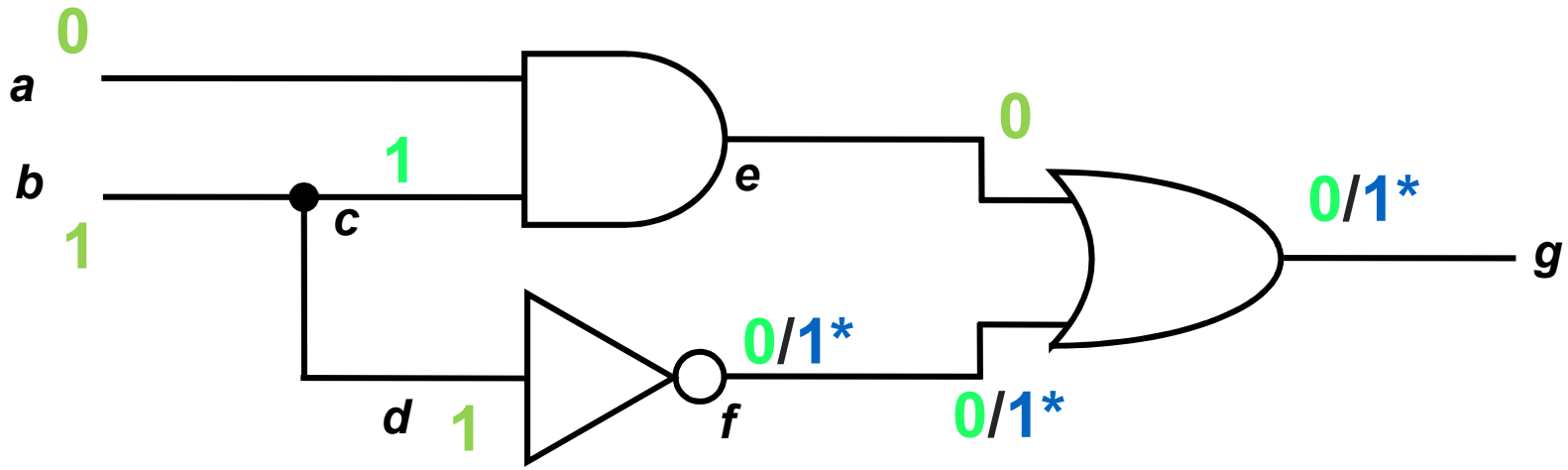
Fault List Minimisation e.g.

- Equivalent faults
  - **e stuck-at-1 or f stuck-at-1**
  - **Test vector (a,b) = (0,1)**

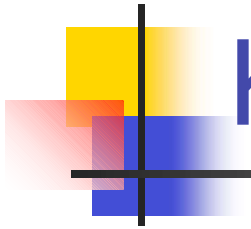


Fault List Minimisation e.g.

- Equivalent faults
  - **e stuck-at-1 (S-A-1) or f S-A-1**
  - **Test vector (a,b) = (0,1)**



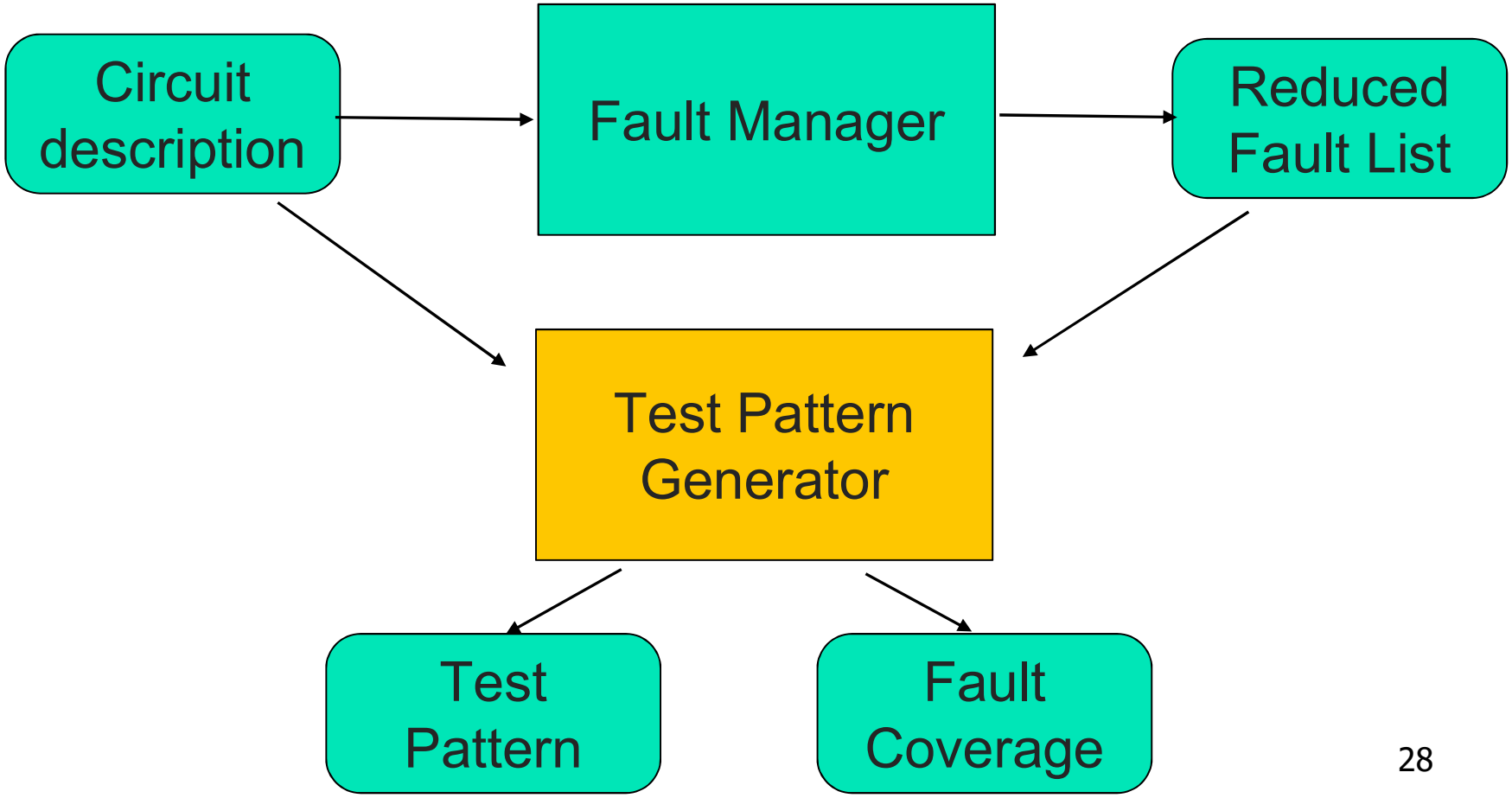
- **Rmk: S-A-1 on both inputs of an OR gate are equivalent (gate level relationship)**



Key takeaways

- Th1: In a combinatorial tree (without divergence) implemented with conventional gates (no XOR), any test that detects all stuck-at faults on the primary inputs also detects all stuck-at faults.
- Th2: In a combinational circuit implemented with conventional gates (no XOR), any test that detects all stuck-at faults on the primary inputs and divergence branches (checkpoints) also detects all stuck-at faults.
- After applying Theorem 1 or 2, the list of faults can still be reduced by using the equivalence rules at the gate level.

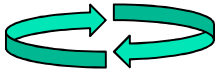
The test plan (cont'd)

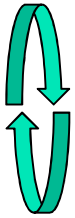


The test plan (cont'd)

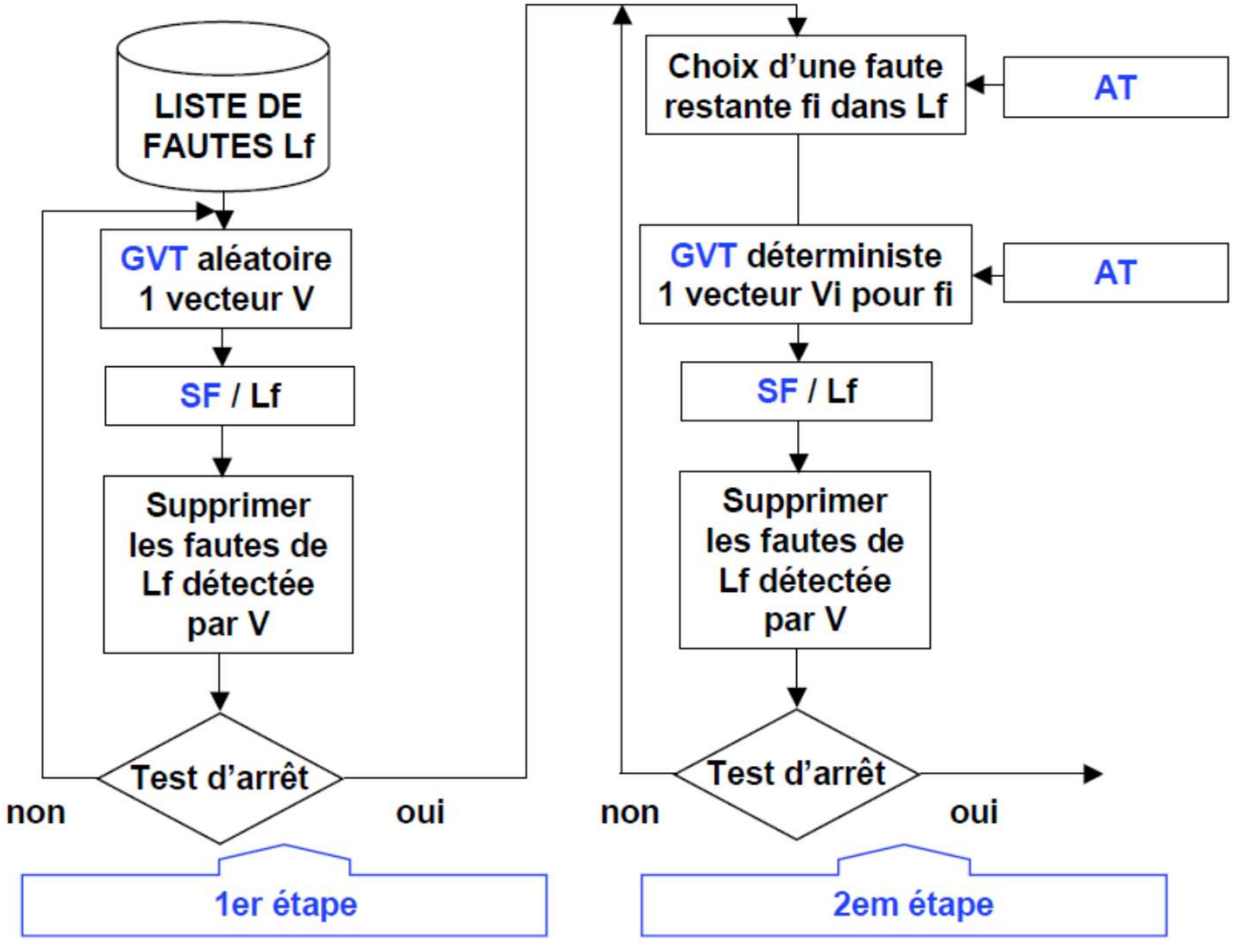
- Step 2:
  - Identify the minimum set of distinct target faults (fault collapsing)
- Step 3:
  - Generate test vectors for every target,
    - 3.1 generate an initial set of vectors at « no » cost (manually, from design validation, randomly, ...) less expensive than deterministic procedures
    - 3.2 generate vectors for remaining faults (use testability analysis for speeding up test generation : fault selection and vector generation)
    - 3.3 use fault simulation for excluding faults detected by the pattern generated so far

Iterative processes

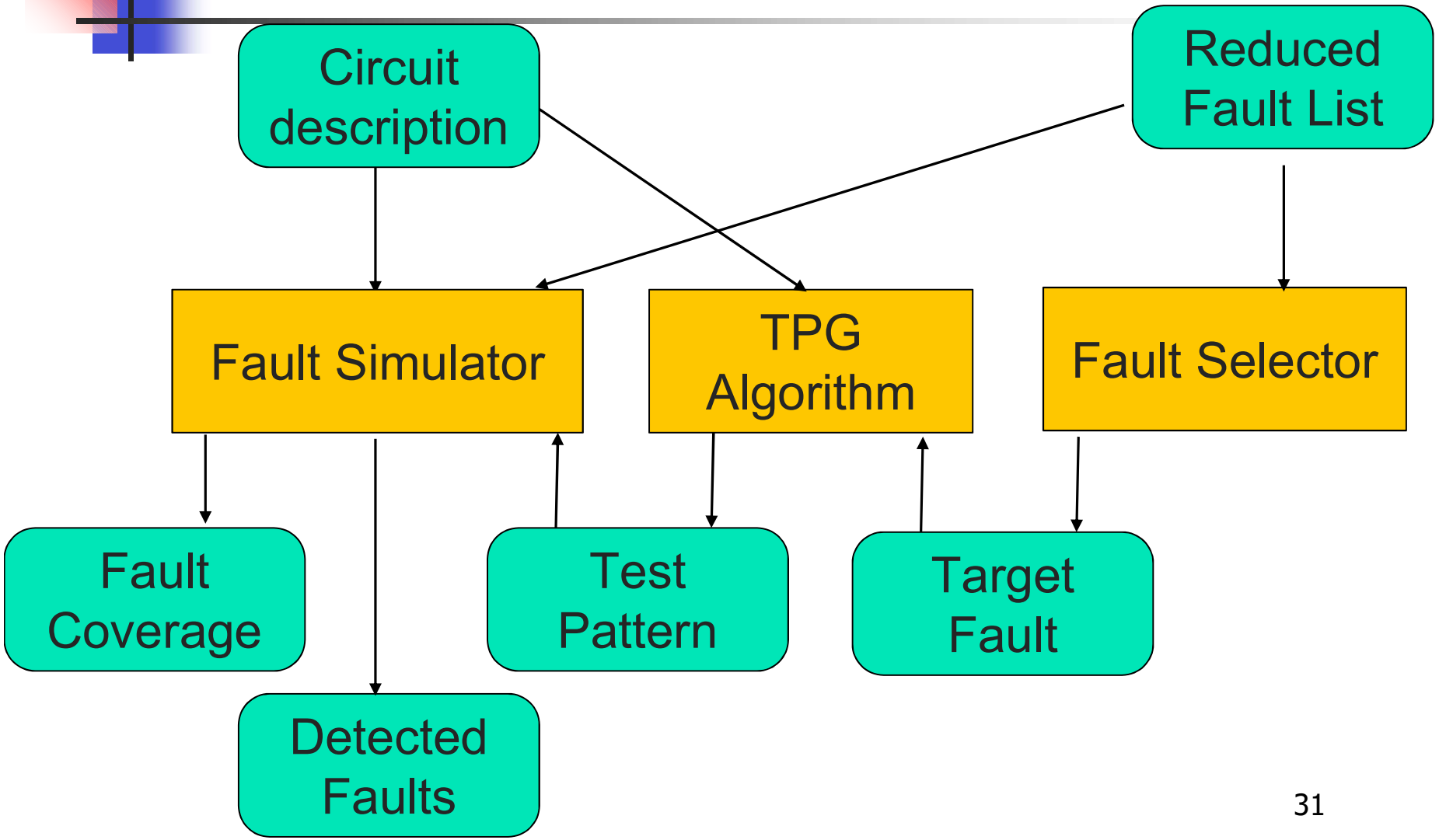


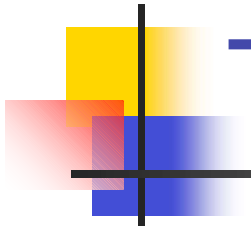


The test plan (cont'd)



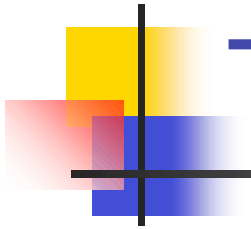
Tools


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



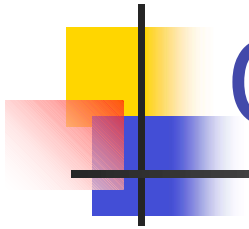
The test plan (cont'd)

- Step 4 (optional):
  - Compact the test pattern
    - Static compaction (compact after generation of the whole test pattern)
      - V1(e1,e2,e3)=(0,X,1) and V2(e1,e2,e3)=(0,0,X)
      - Vcompacted(e1,e2,e3)=(0,0,1), *covers V1 and V2*
    - Dynamic compaction
      - Start generation of next vector from PIs values assigned from the last generated vecor



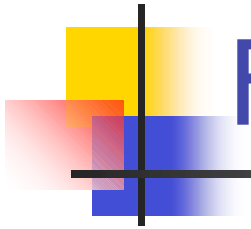
Testability Analyzer

- High trade-off between result accuracy and CPU time.
- A Circuit is testable when you ATPG can manage it!!!!!



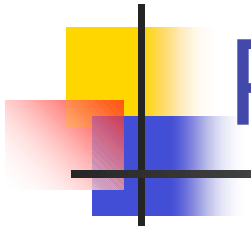
Outline

- Introduction (reminder)
- ATPG: process and tools
- Simulation



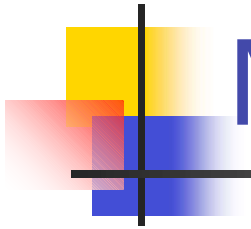
Fault Simulation

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



Problem

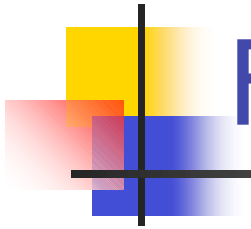
- Given
  - A circuit
  - A sequence of test vectors
  - A fault model
- Determine
  - Fault coverage
  - Set of undetected faults



Motivation

- Determine test quality (FC%)
- Identify undetected fault for further test improvement
- Fault dictionnary for diagnostic

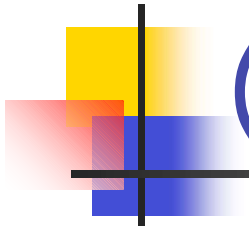
- Fault-dropping -- a fault once detected is dropped from consideration as more vectors are simulated; fault-dropping may be suppressed for diagnosis
- Fault sampling -- a random sample of faults is simulated when the circuit is large



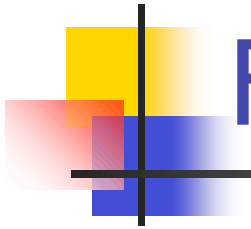
Fault Simulation Scenario

- Circuit model: mixed-level
  - Mostly logic with some switch-level for high-impedance (Z) and bidirectional signals
  - High-level models (memory, etc.)
- 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)

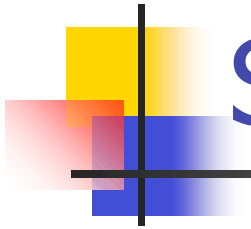


- Faults:
  - Mostly single stuck-at faults
  - Sometimes stuck-open, transition, and path-delay faults; analog circuit fault simulators are not yet in common use



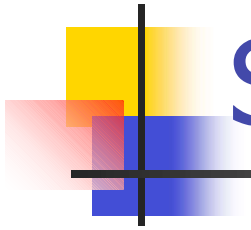
Fault Simulation Algorithms

- Serial
- Parallel
- Deductive
- ....



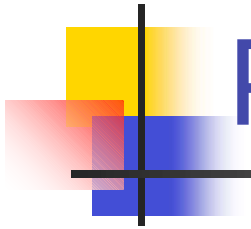
Serial Algorithm

- Based on logic simulation:
- 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 (to sum up)

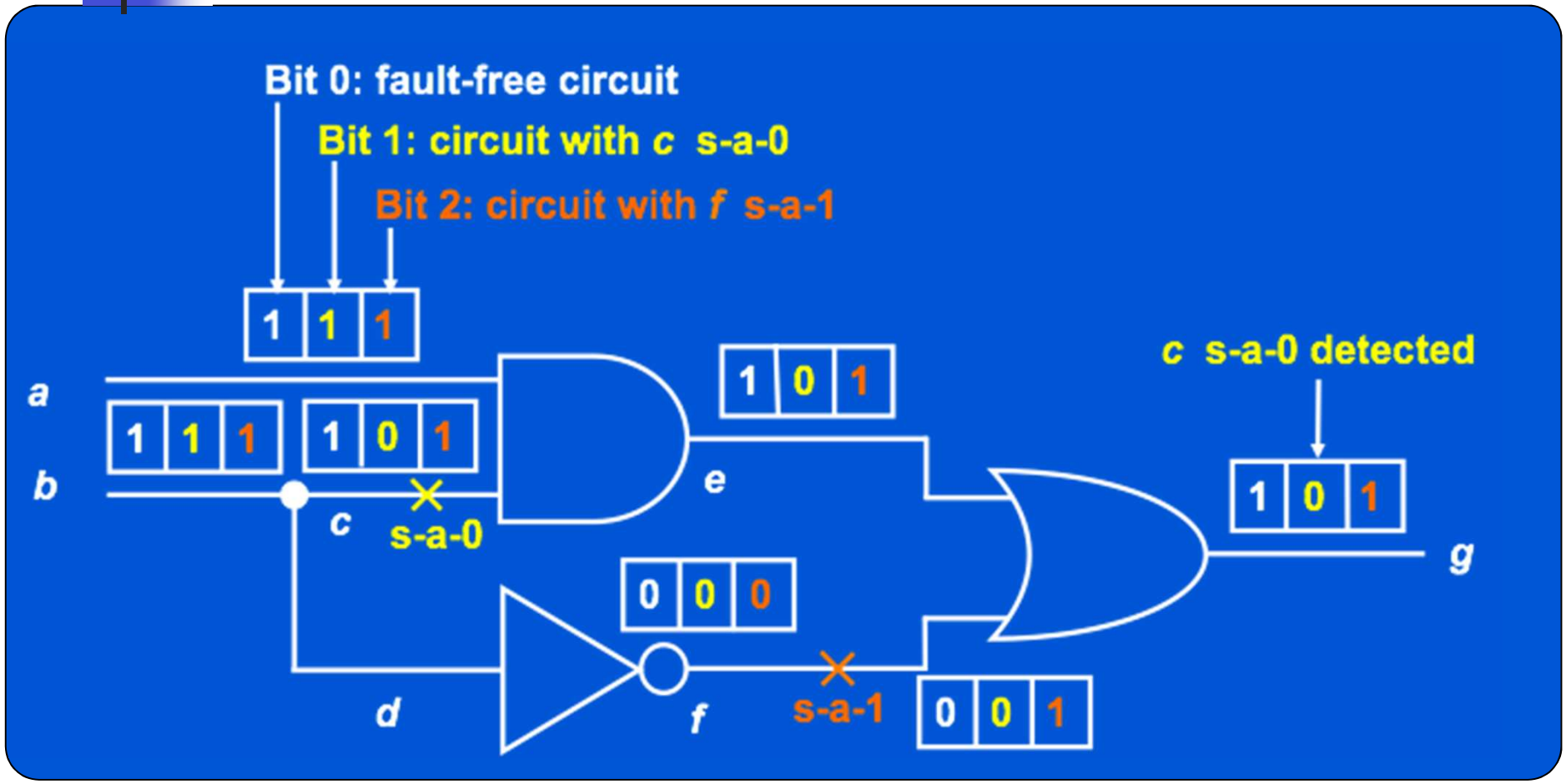
- + very simple
- - not efficient
  - Much repeated computation; CPU time prohibitive for VLSI circuits
  - Intel I7 is about ~10M gates
  - 20M faults, 1 simulation = 1s
  - 20Ms ~= **231 days**
- Alternative: Simulate many faults together

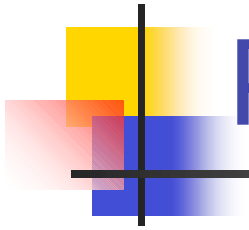


Parallel Fault Simulation

- Exploits inherent bit-parallelism of logic operations on computer words
- Storage: one word per line, one fault per bit (first word bit stores line fault-free value
- 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
- Best with 2 states (0,1), requires a second word per line for X encoding (unknows in sequential circuits)

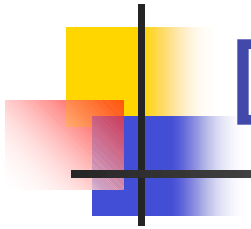
Parallel Fault Simulation





Parallel algorithm (to sum up)

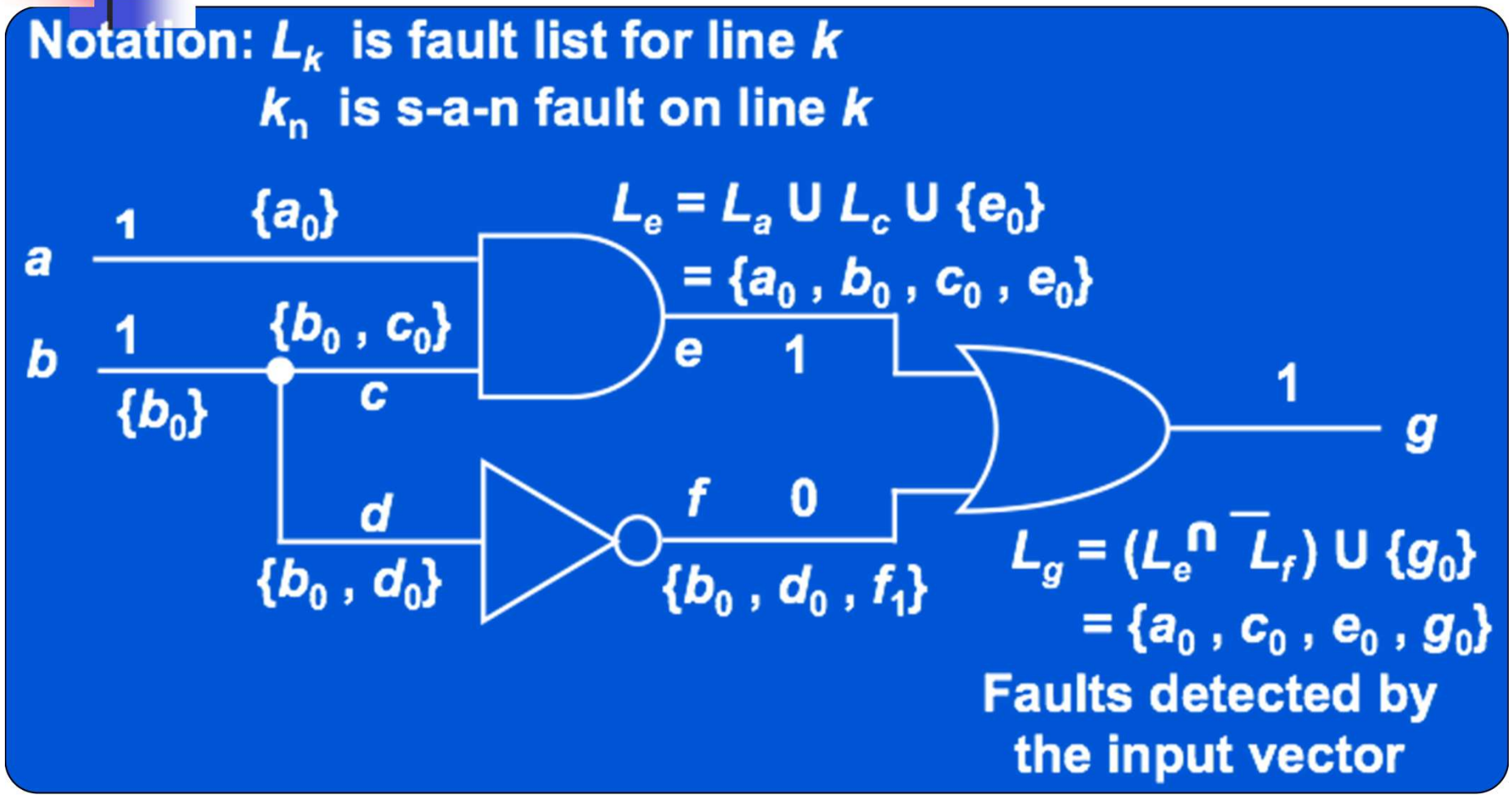
- + still very simple
- + Memory requirement known
- + more efficient than serial
  - Intel I7 is about ~10M gates
  - 20M faults, 1 simulation = 1s
  - 20Ms ~= **231 days**
  - Using a 64bits machine: **231/63 ~= 4 days**
- Requires several runs when #faults>word size (e.g.>64)

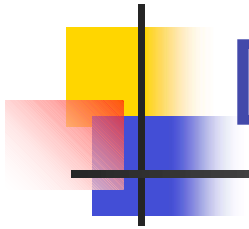


Deductive Fault Simulation

- 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





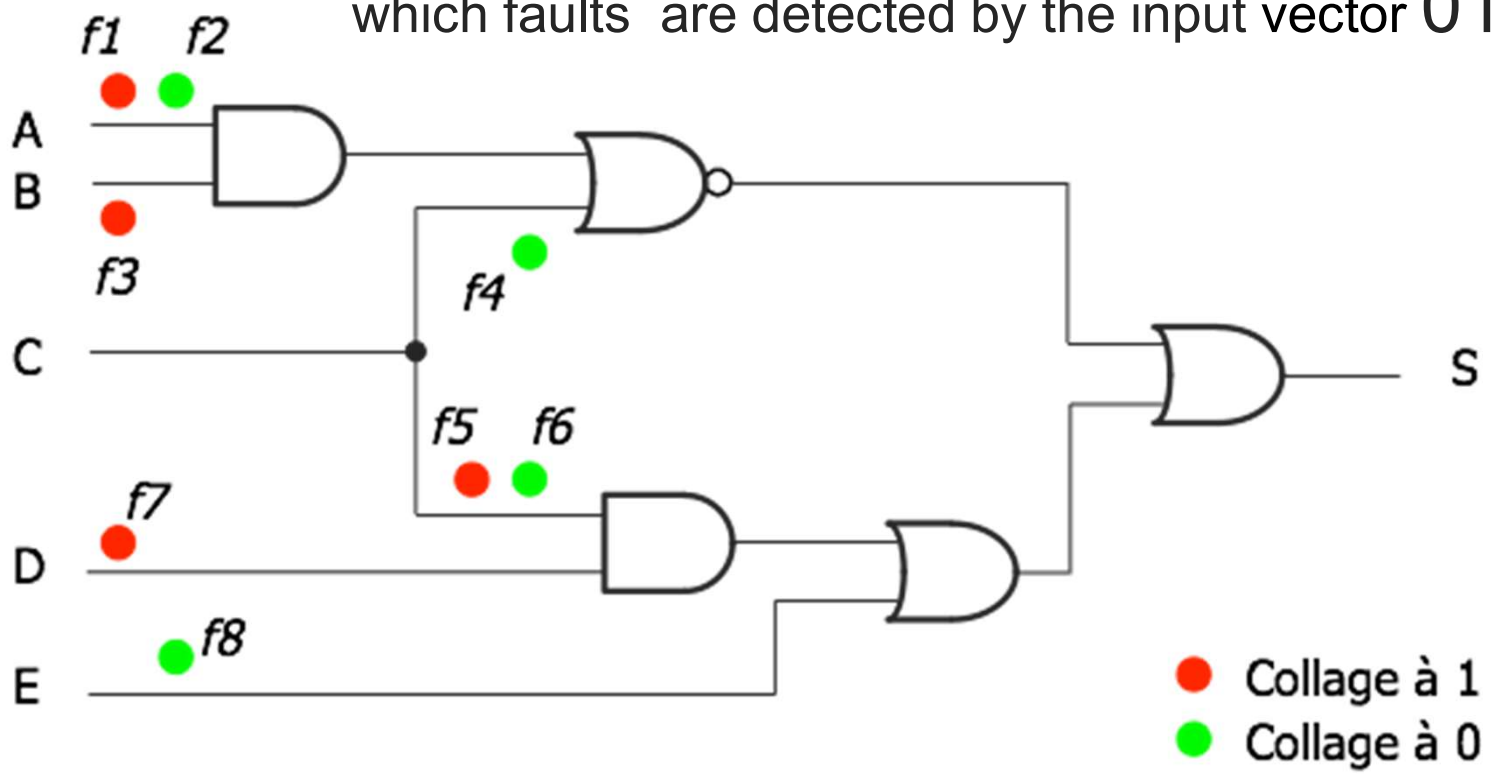
Deductive algorithm

- ++ more efficient than parallel
- Single iteration for all faults
  - Intel I7 is about ~10M gates
  - 20M faults, 1 simulation = 1s
  - 20Ms ~= **1s**
- - difficult to predict peak memory usage

Exercice

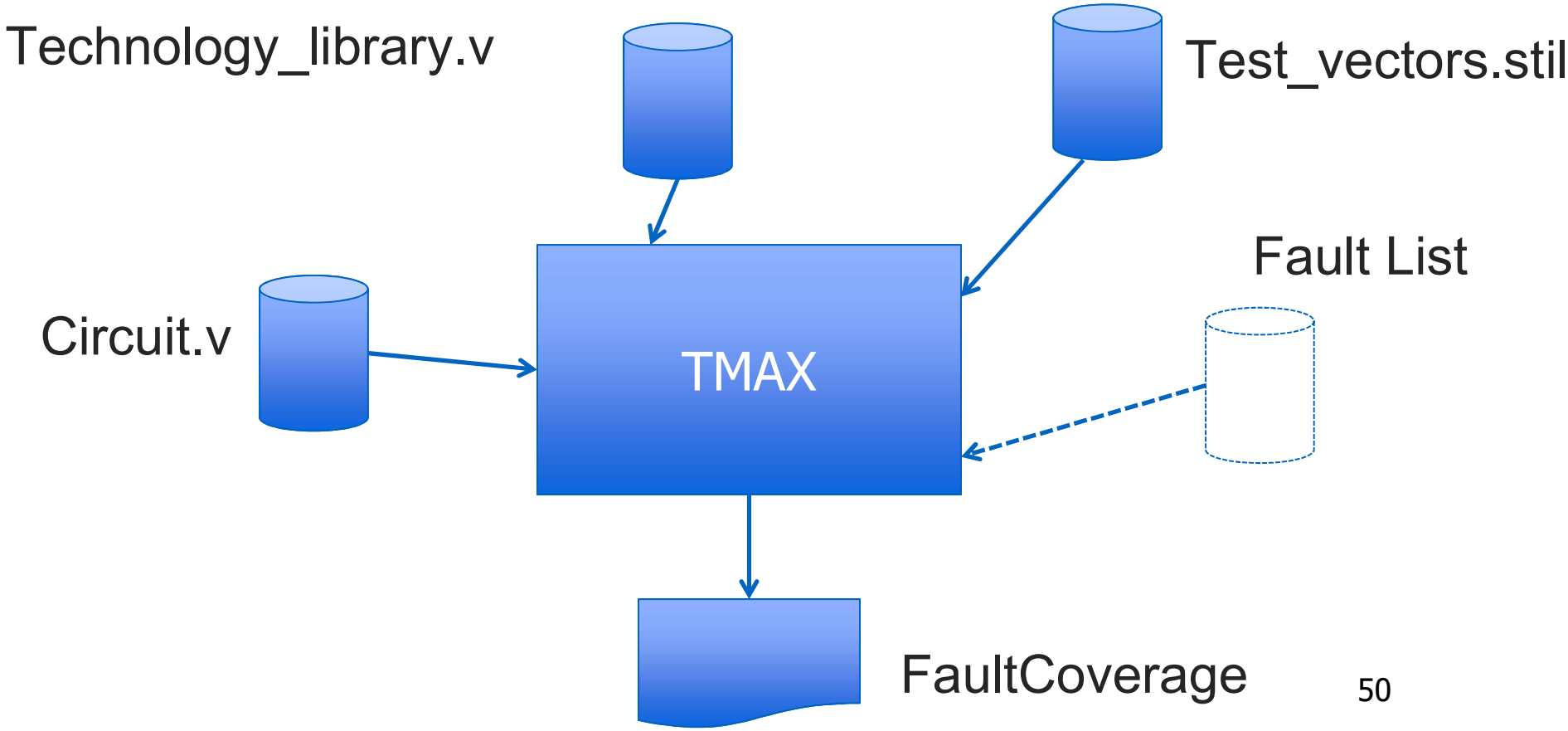
Considering the reduced list of faults below,

which faults are detected by the input vector 01110?



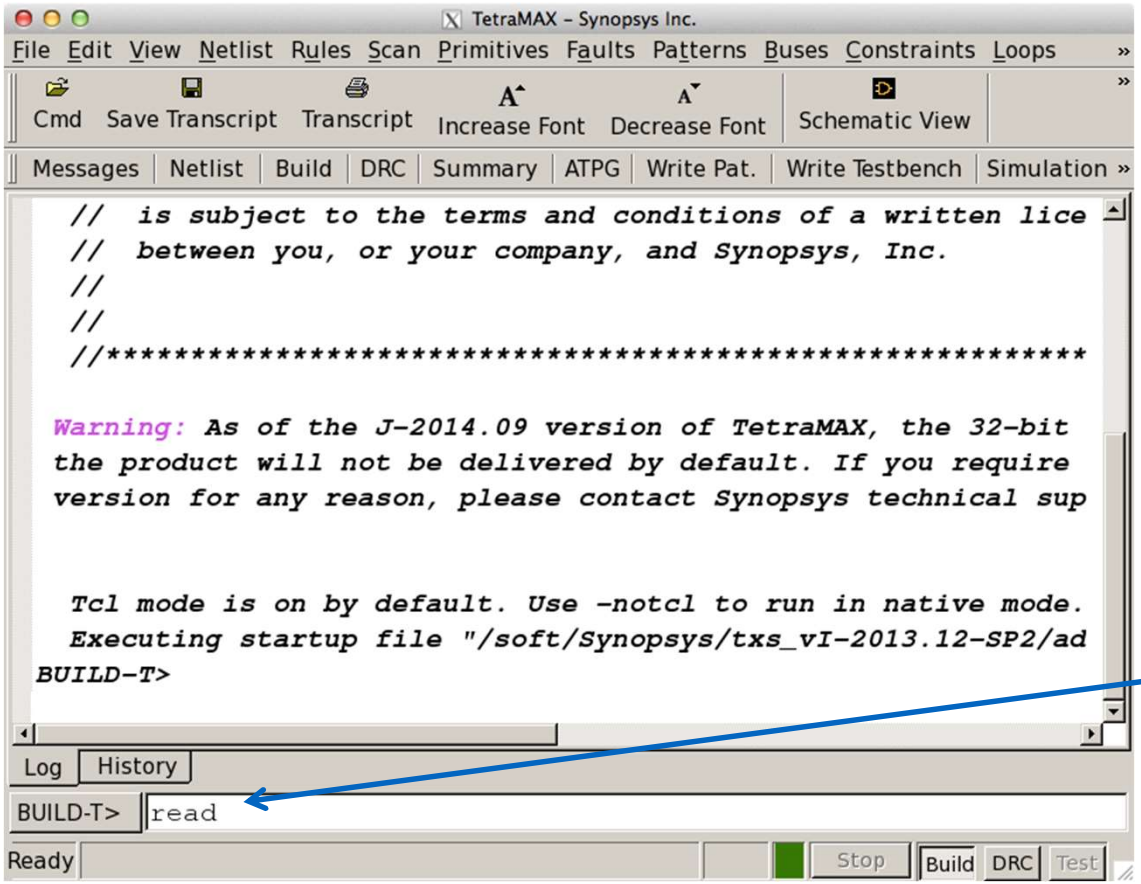
In practice

- Use of Synopsys© Tetramax (ATPG)

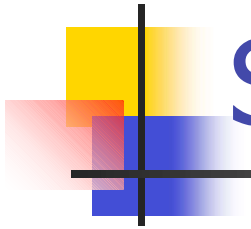


Invoking TetraMax

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



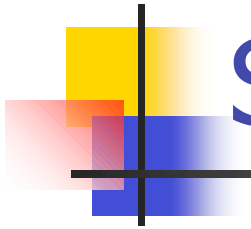
You can enter commands



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

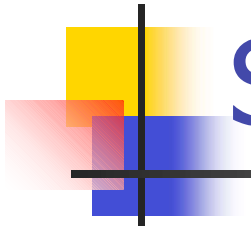
```
set_patterns -external example_exo1.stil
```

- 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
```