Hardware Vulnerabilities

Paolo PRINETTO
Director
CINI Cybersecurity National Laboratory
Paolo.Prinetto@polito.it
Mob. +39 335 227529

CYBER CHALLENGE.IT

CYBERSECURITY NATIONAL LABORATORY

https://cybersecnatlab.it

License & Disclaimer

License Information

Disclaimer

- We disclaim any warranties or representations as to the accuracy or completeness of this material.
- Materials are provided “as is” without warranty of any kind, either express or implied, including without limitation, warranties of merchantability, fitness for a particular purpose, and non-infringement.
- Under no circumstances shall we be liable for any loss, damage, liability or expense incurred or suffered which is claimed to have resulted from use of this material.

Goal of the presentation

- Providing:
  - a taxonomy of Hardware Vulnerabilities
  - some significant examples
  - some suggestions for the activation of proper countermeasures.

Prerequisites

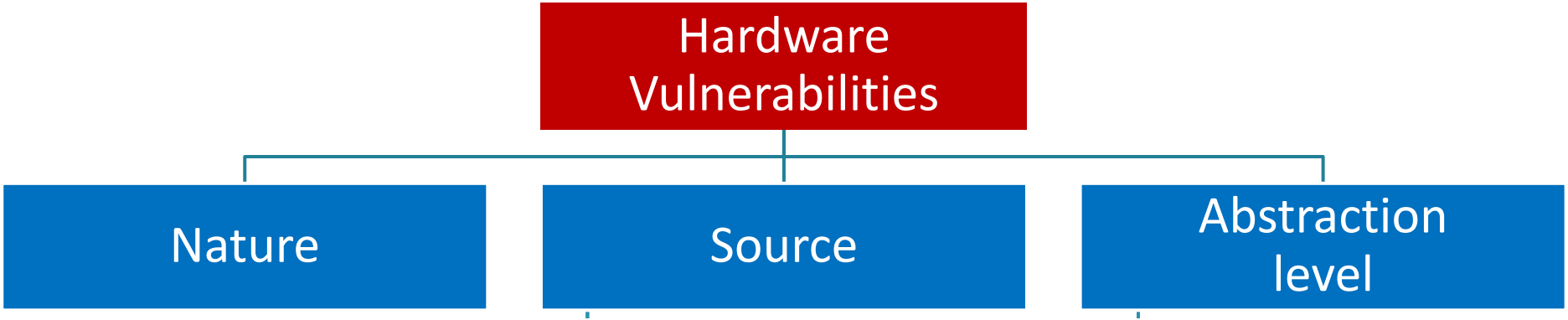
- Lecture:
  - *CS_1.04 – Cybersecurity: Vulnerabilities*

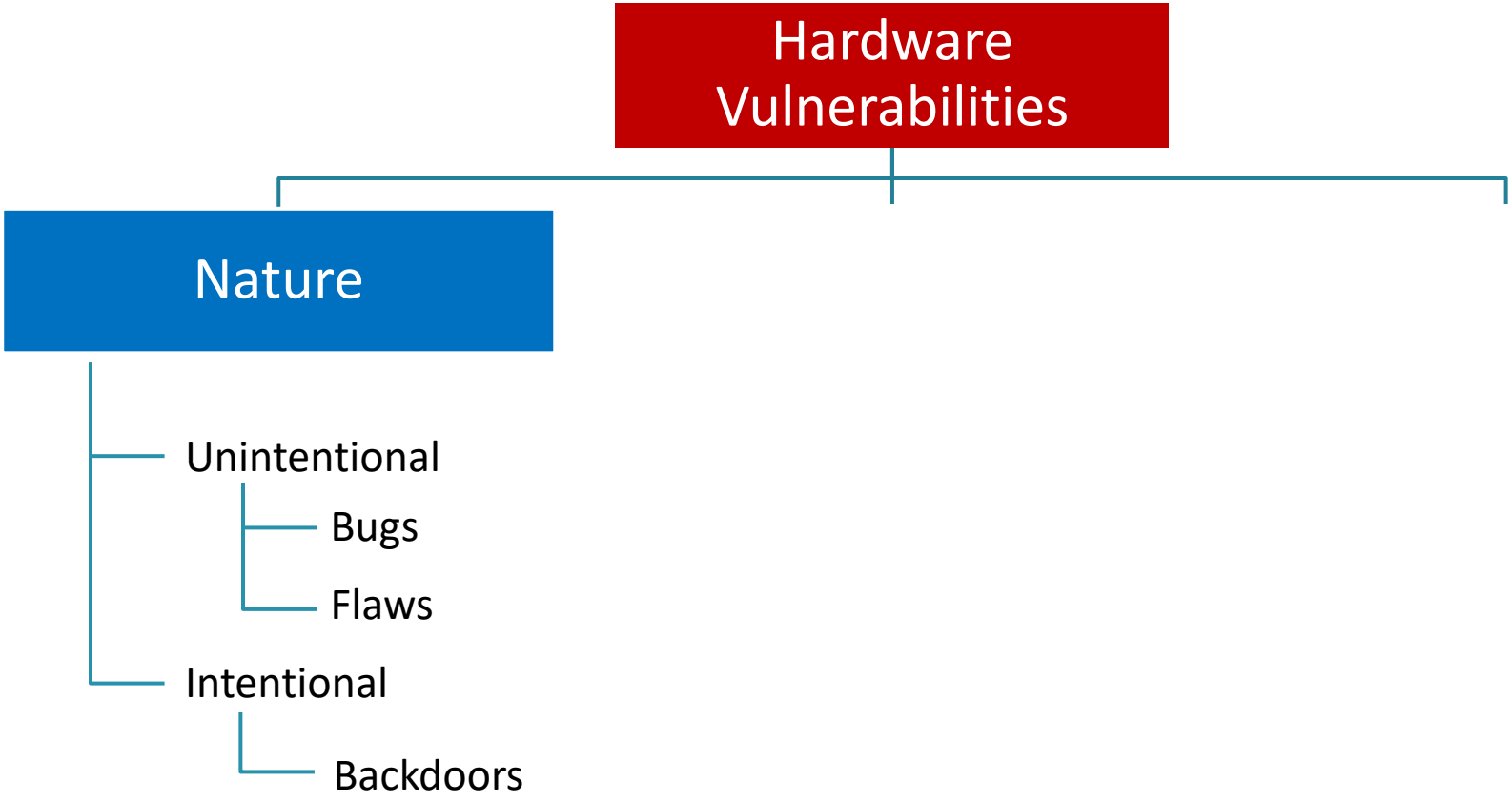
Insights for further details

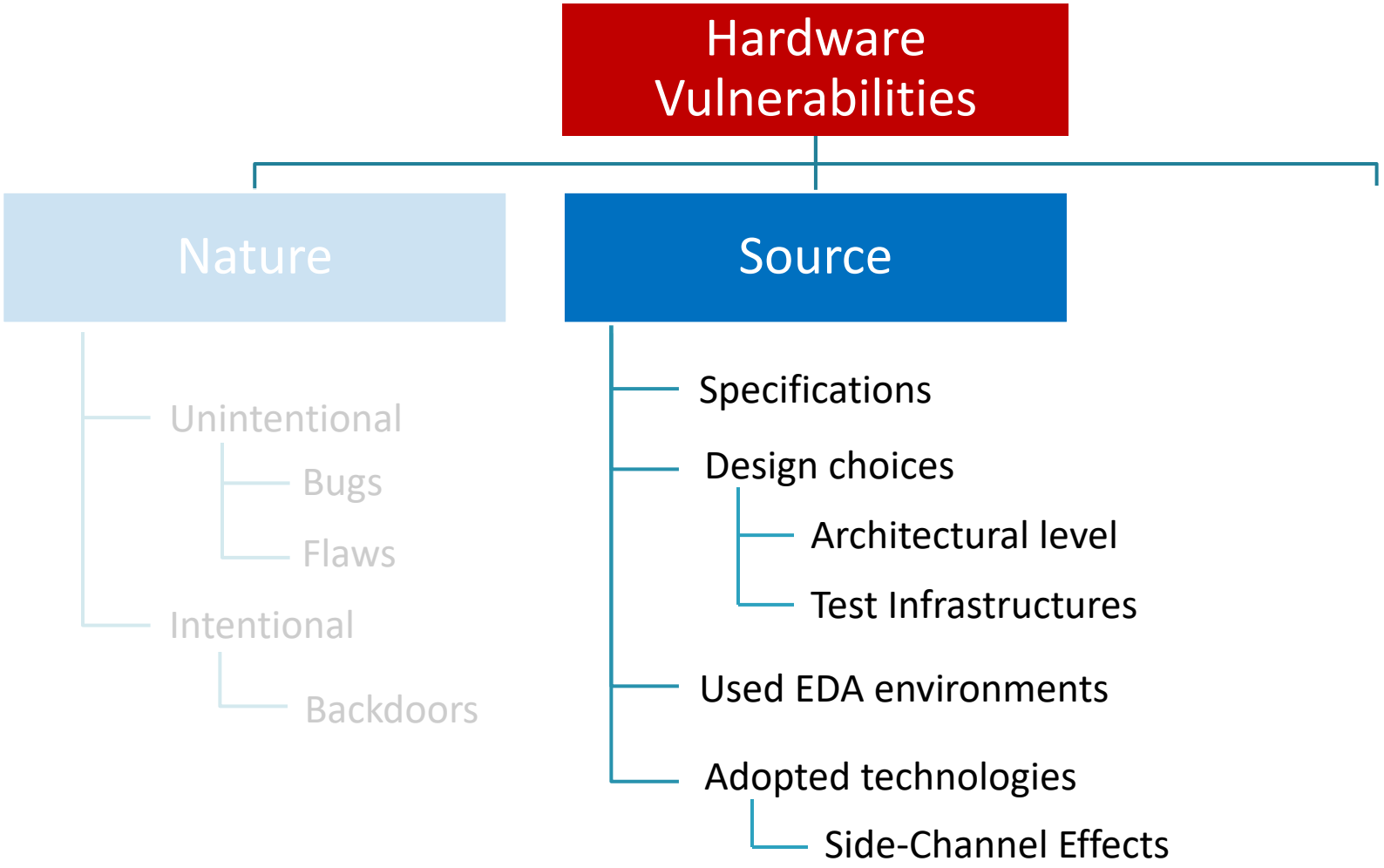
- After the last slide of the lecture, few slides have been added to present Insights related to some of the topics covered.

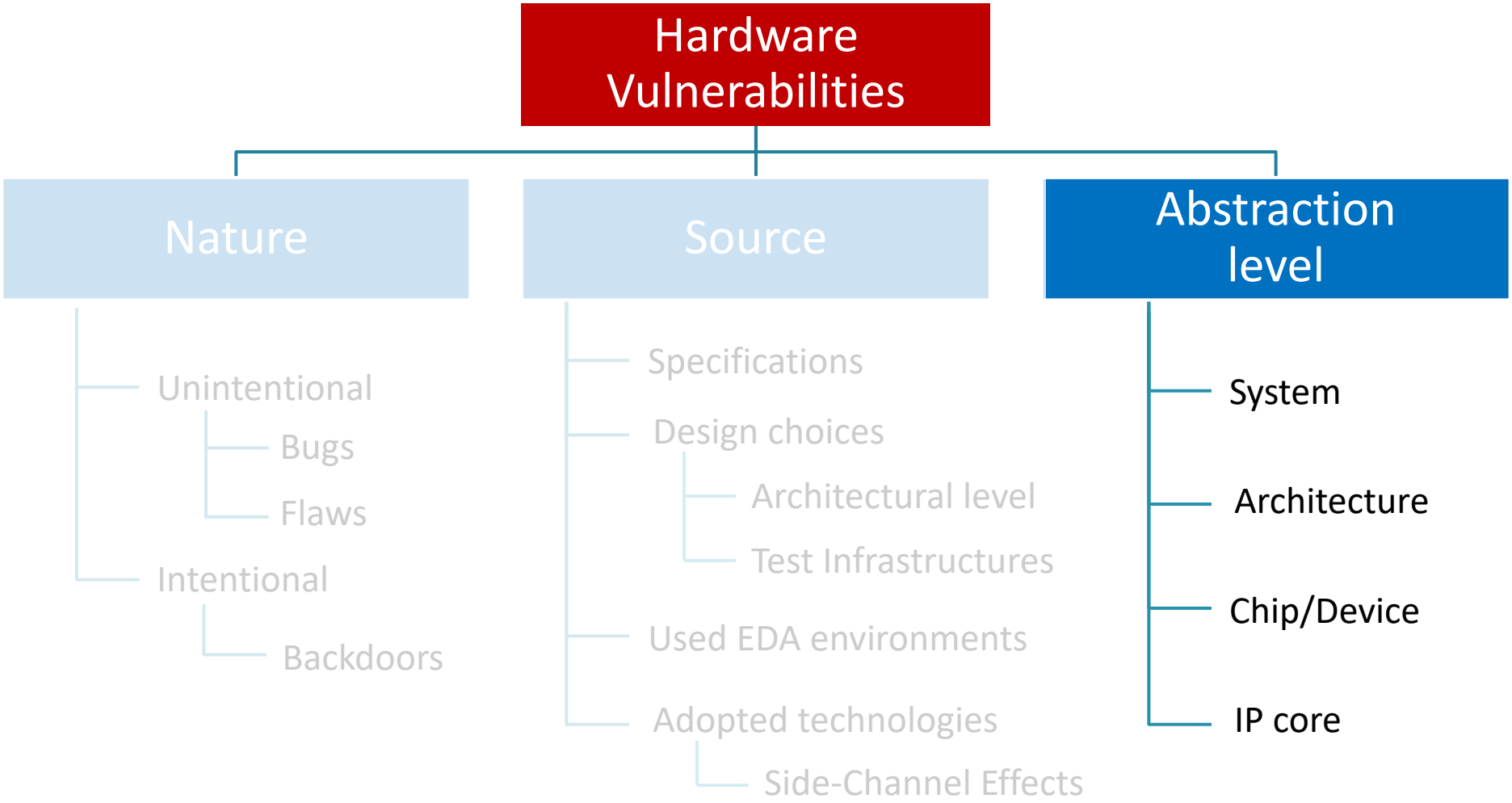
Hardware Vulnerabilities

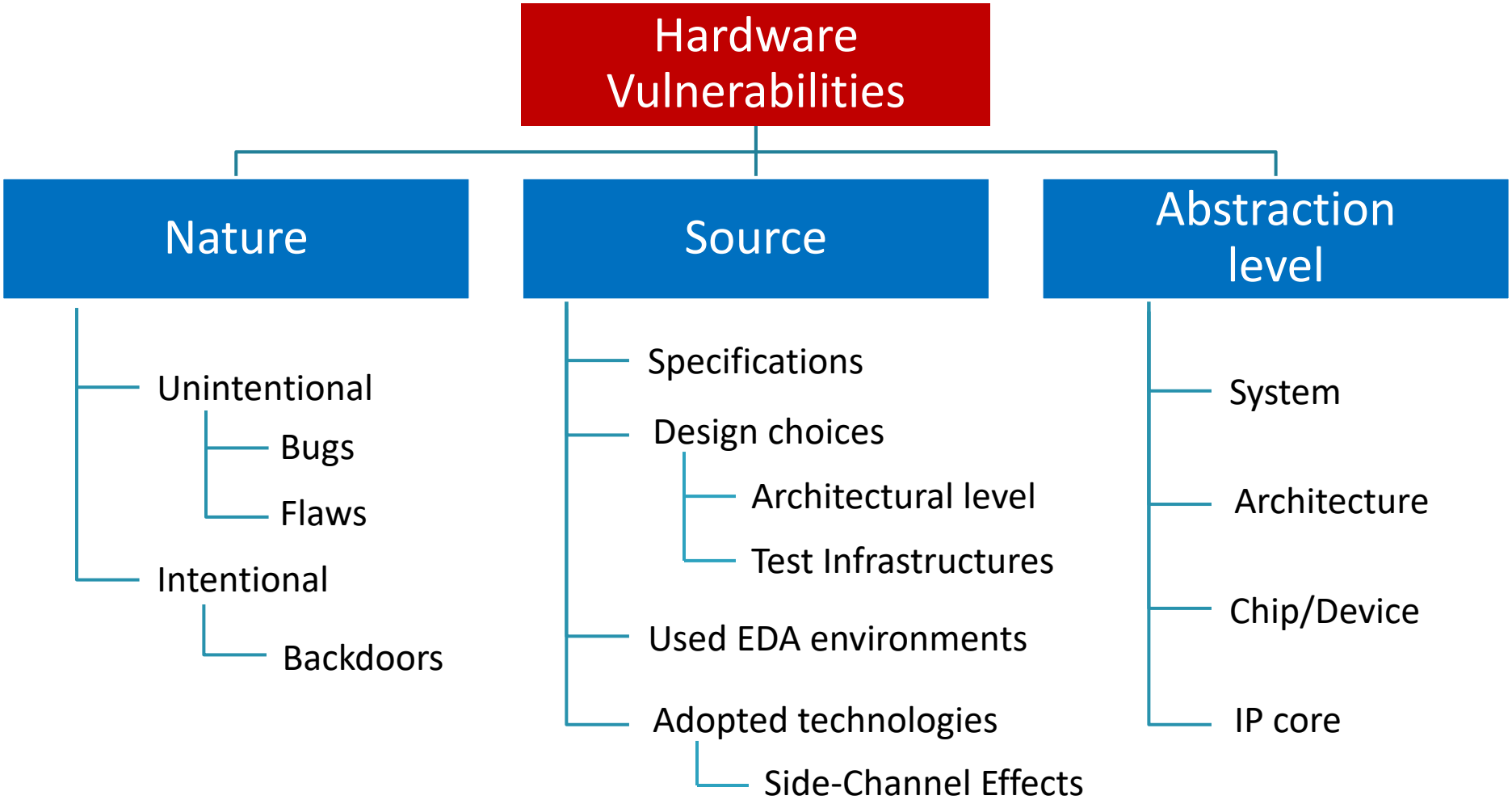
- Can be clustered according to several orthogonal dimensions
- In the sequel we are going to focus on 3 of them:
  - vulnerability *nature*
  - vulnerability *source*
  - vulnerability *abstraction level*


Hardware Vulnerabilities
Nature
Source
Abstraction level


Hardware Vulnerabilities
Nature
Unintentional
Bugs
Flaws
Intentional
Backdoors

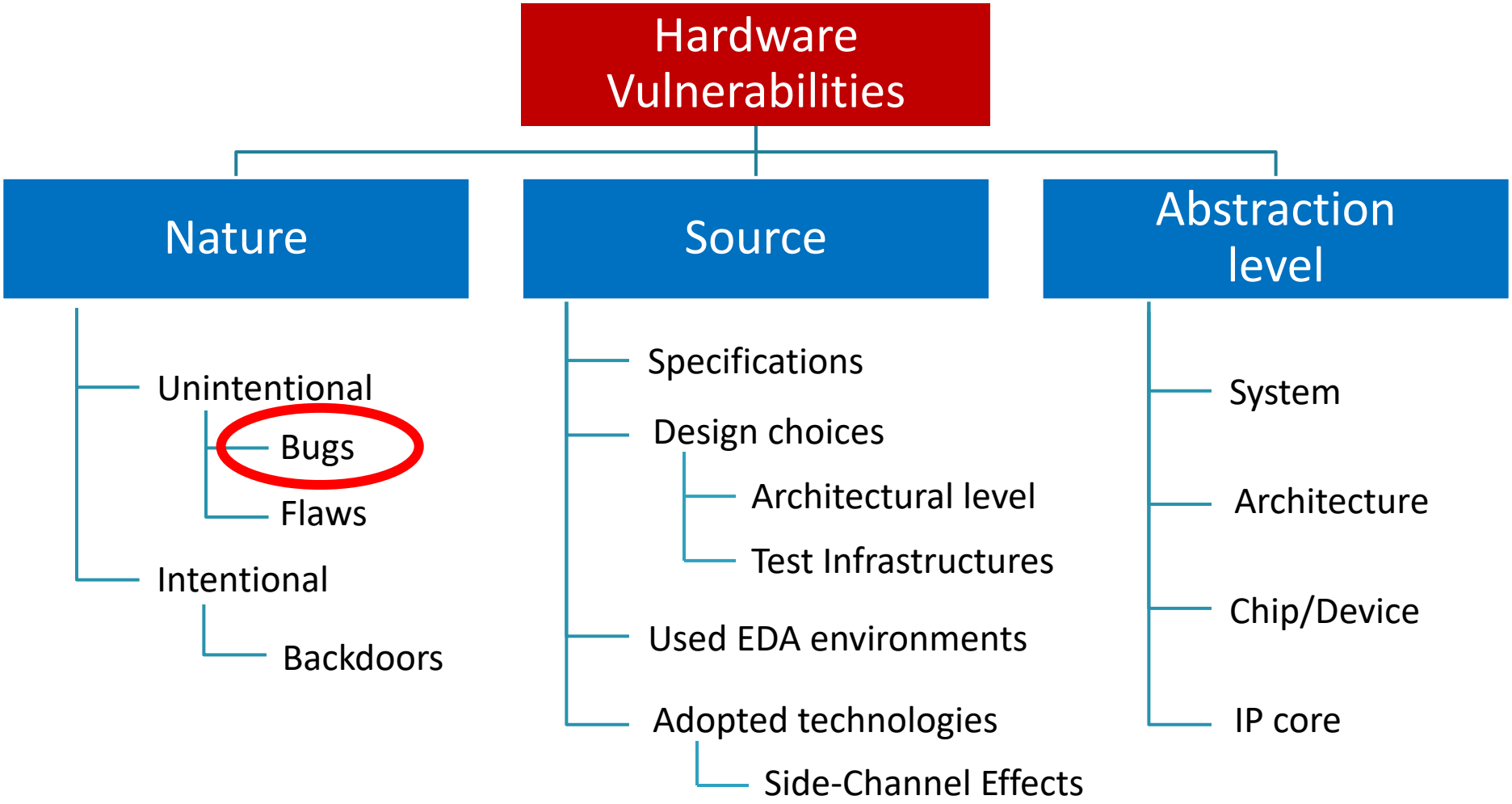

Hardware Vulnerabilities
Nature
Source
Unintentional
Bugs
Flaws
Intentional
Backdoors
Specifications
Design choices
Architectural level
Test Infrastructures
Used EDA environments
Adopted technologies
Side-Channel Effects


Hardware Vulnerabilities
Nature
Source
Abstraction level
Unintentional
Bugs
Flaws
Intentional
Backdoors
Specifications
Design choices
Architectural level
Test Infrastructures
Used EDA environments
Adopted technologies
Side-Channel Effects
System
Architecture
Chip/Device
IP core

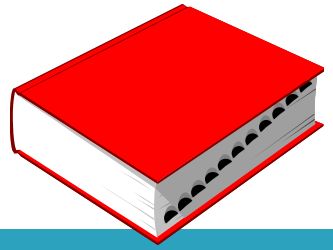

Hardware Vulnerabilities
Nature
Source
Abstraction level
Unintentional
Bugs
Flaws
Intentional
Backdoors
Specifications
Design choices
Architectural level
Test Infrastructures
Used EDA environments
Adopted technologies
Side-Channel Effects
System
Architecture
Chip/Device
IP core

Some details…

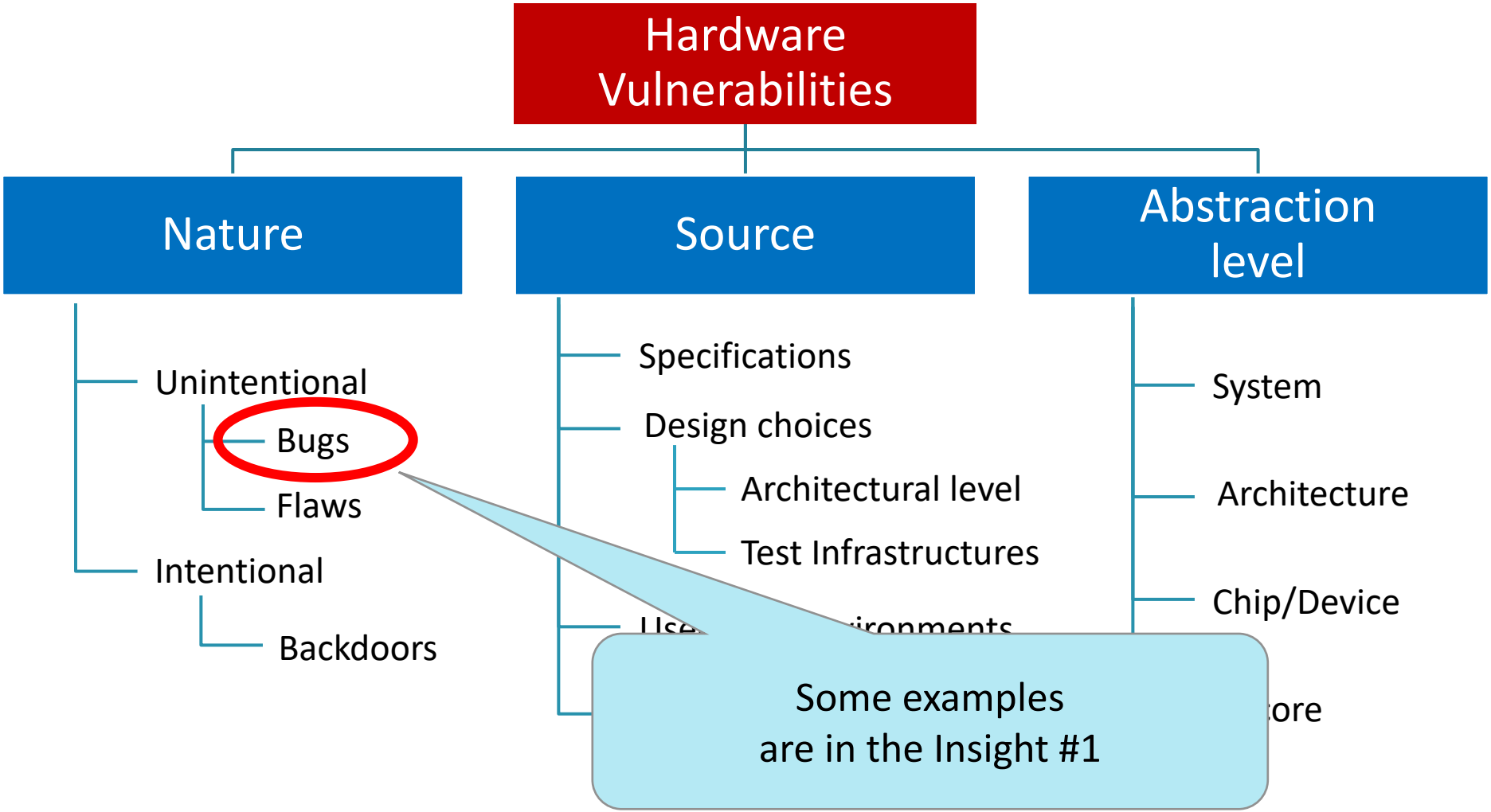
- In the sequel we shall focus of some significant cases…

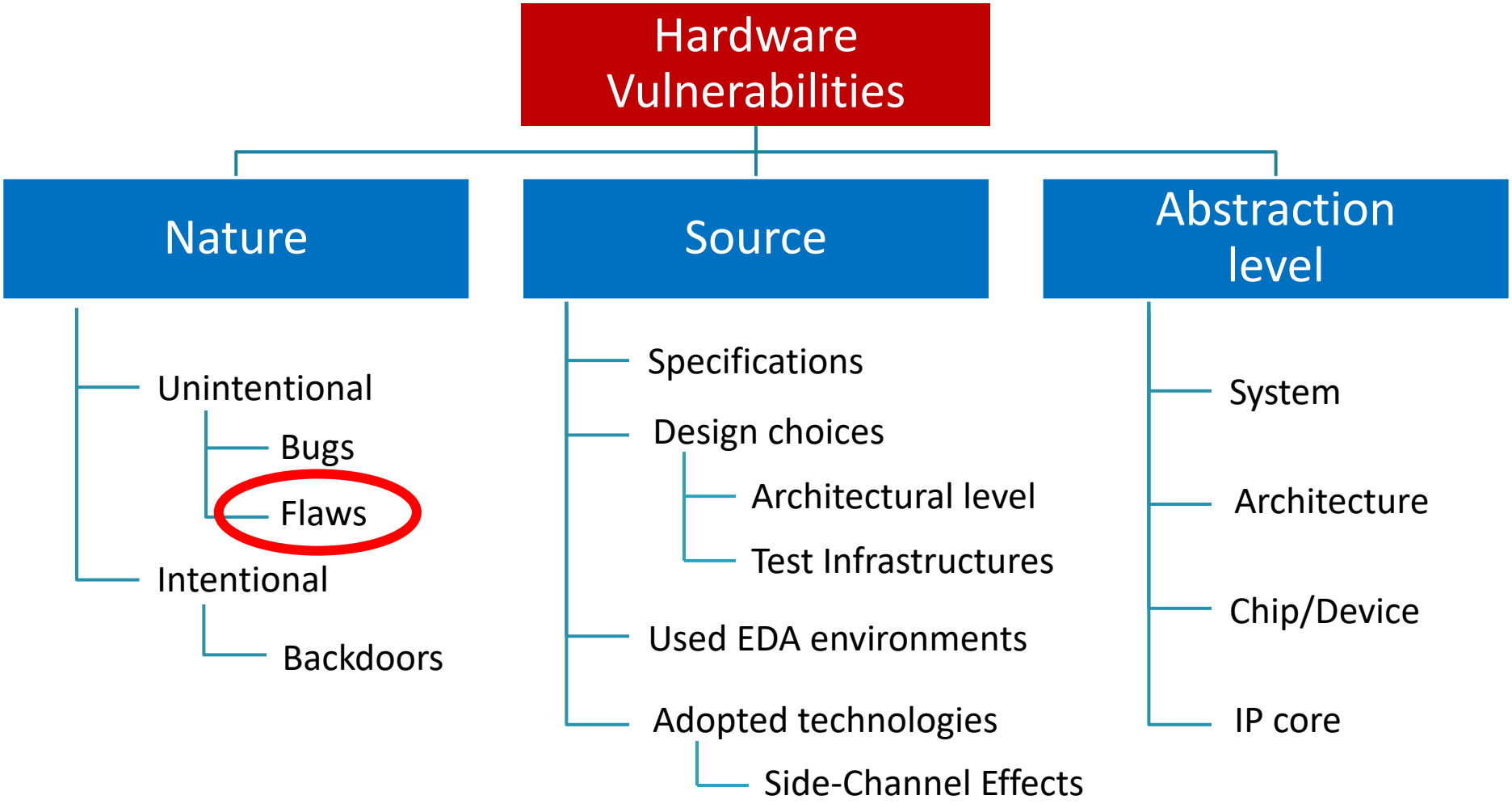

Hardware Vulnerabilities
Nature
Source
Abstraction level
Unintentional
Bugs
Flaws
Intentional
Backdoors
Specifications
Design choices
Architectural level
Test Infrastructures
Used EDA environments
Adopted technologies
Side-Channel Effects
System
Architecture
Chip/Device
IP core

Hardware Bug

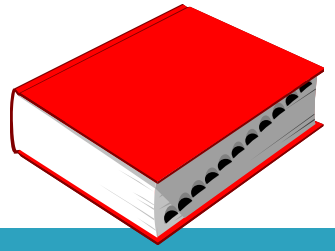


- An inconsistency between a specification and its actual implementation, introduced by a mistake during the design and not detected during the *Validation & Verification* (V&V) phases.


Hardware Vulnerabilities
Nature
Source
Abstraction level
Unintentional
Bugs
Flaws
Intentional
Backdoors
Specifications
Design choices
Architectural level
Test Infrastructures
System
Architecture
Chip/Device
Some examples are in the Insight #1


Hardware Vulnerabilities
Nature
Source
Abstraction level
Unintentional
Bugs
Flaws
Intentional
Backdoors
Specifications
Design choices
Architectural level
Test Infrastructures
Used EDA environments
Adopted technologies
Side-Channel Effects
System
Architecture
Chip/Device
IP core

Hardware Flaw



- A non-primary feature of the hardware device that:
  - does not constitute an inconsistency w.r.t. its specs
  - mostly stems from a misconception of the designer who did not take into consideration its potential dangerousness.

Hardware Flaw: example

- *Speculative Execution* in modern processors
  - On branch instructions, both branches are executed before condition check
  - At commit time, only the correct execution is validated

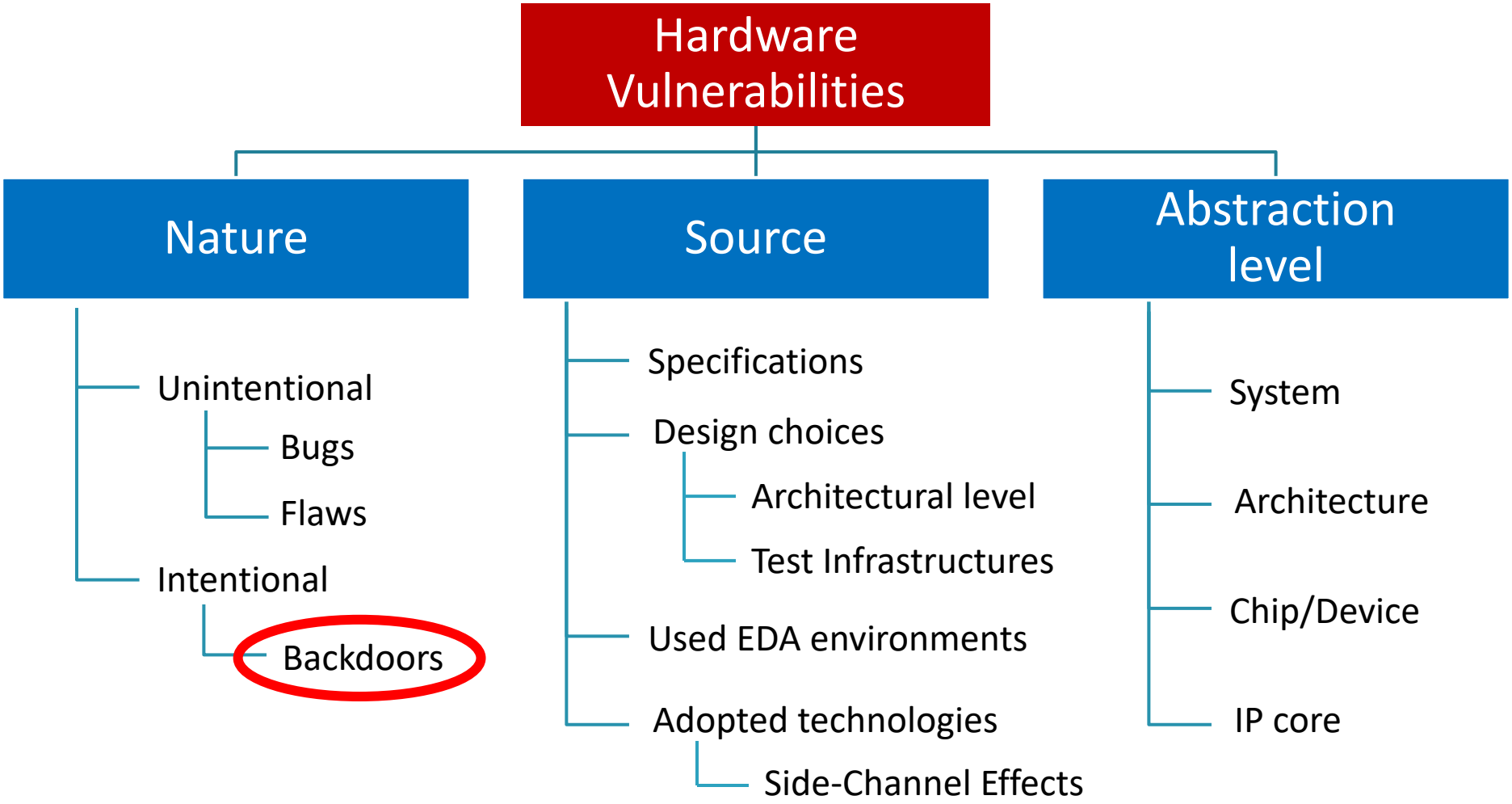
Hardware Flaw: example

- *Speculative Execution* in modern processors
  - On branch instructions, both branches are executed before condition check
  - At commit time, only the correct execution is validated
- Great for performance, but …
  - Commitment does not delete completely non-valid path
  - Traces of discarded execution may leak information

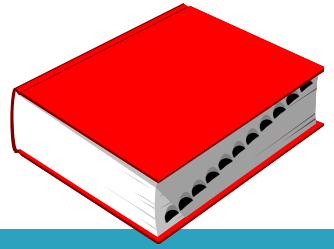
Examples: Microarchitectural Flaws

- Processors do not enter an error state but reveal private information!
- They usually allow a concurrent (aggressor) program to fraudulently access private data and keys of a victim program

- Spectre (2018)
- Meltdown (2018)
- Foreshadow (2018)
- ZombieLoad (2018)
- Spoiler (2019)


Hardware Vulnerabilities
Nature
Source
Abstraction level
Unintentional
Bugs
Flaws
Intentional
Backdoors
Specifications
Design choices
Architectural level
Test Infrastructures
Used EDA environments
Adopted technologies
Side-Channel Effects
System
Architecture
Chip/Device
IP core

Intentional Vulnerabilities



- When a vulnerability is inserted intentionally, it can be defined as a *backdoor* (or *trapdoor*), since the person who inserts it wants to guarantee, for her/himself or someone else, the possibility of access or subsequent use outside the set of intended use cases.

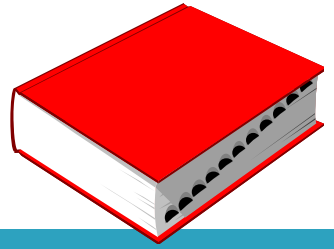
Hardware backdoor -- Examples

- *Undocumented machine instructions*
- *Hardware Trojan*

Undocumented CPU Instructions

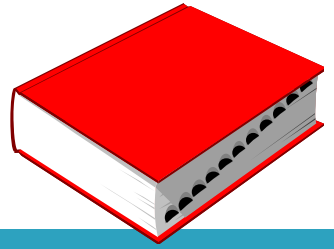
- An undocumented machine instruction has been detected in some CPUs x86 manufactured by VIA Technologies
- The instruction ALTINST (0F 3F) forces the CPU to execute an alternative ISA (Instruction Set Architecture) and directly accessing the RISC core available within the CPU by executing a JMP EAX, i.e., a jump at the memory location whose address is stored into the EAX register

Hardware Trojan



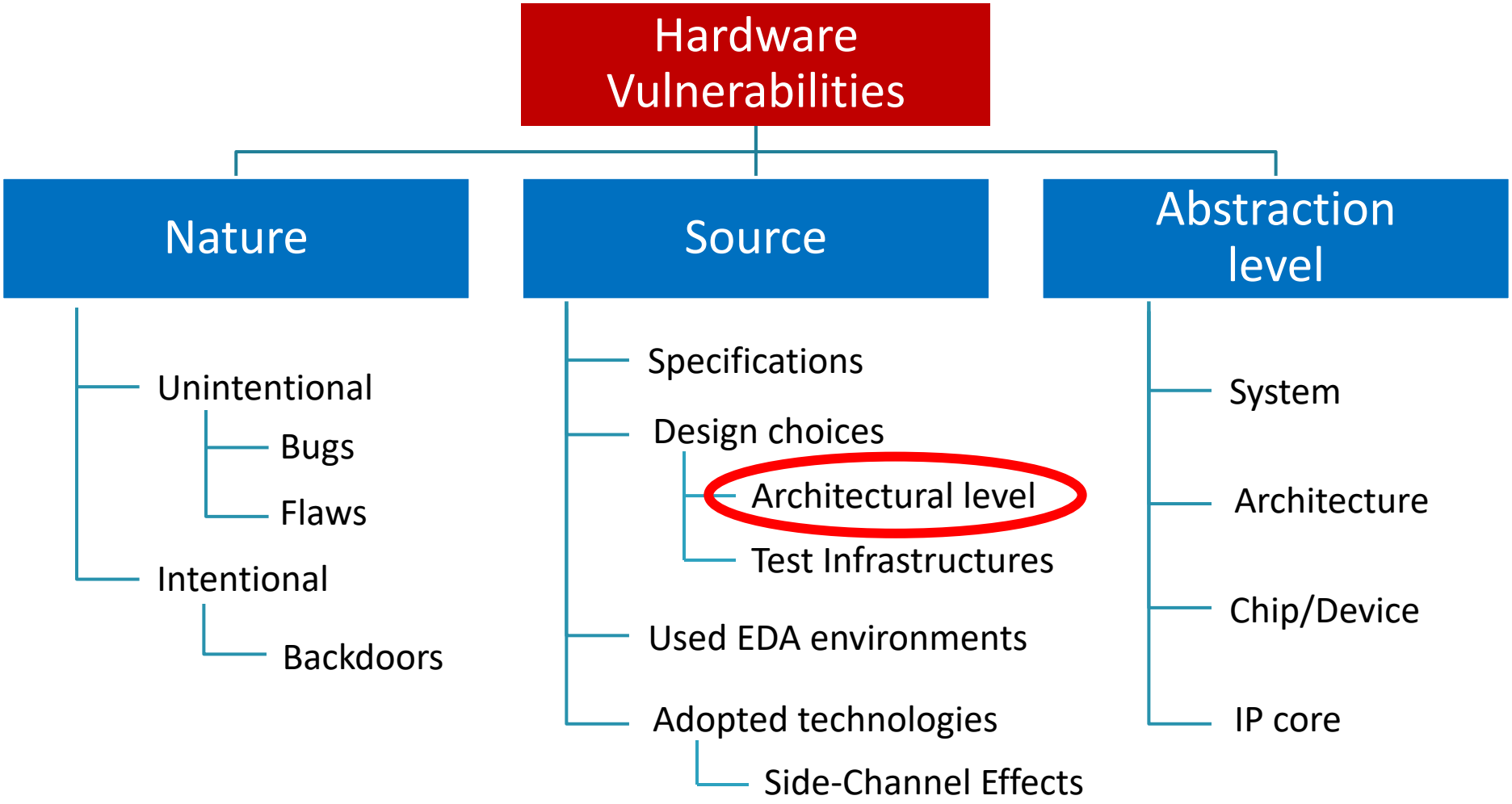
- A rogue piece of circuitry fraudulently inserted during the design or production phase, which can carry out unauthorized actions when its *triggering conditions* are satisfied.

Hardware Trojan



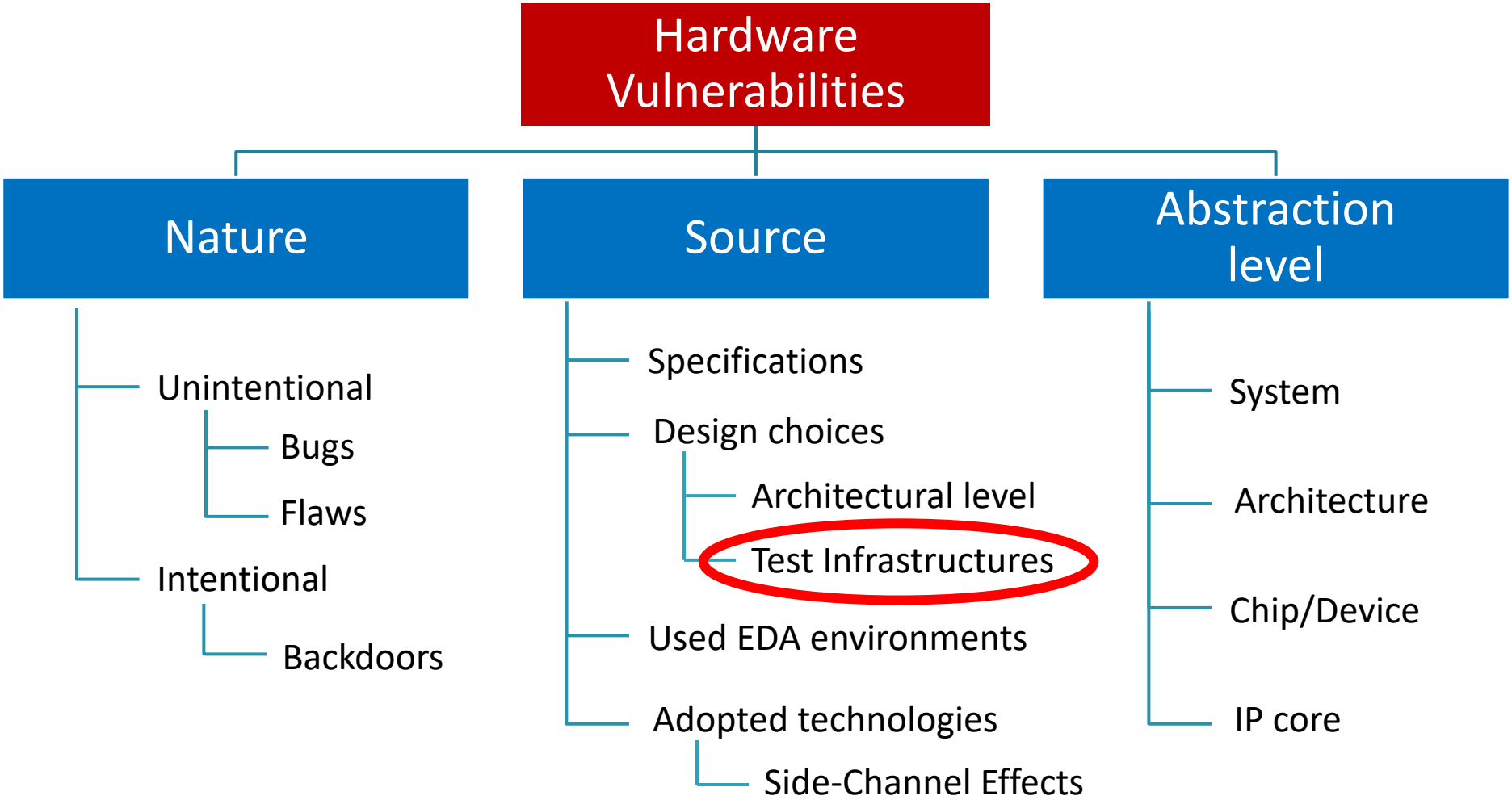
- A rogu piece of circuitry fraudulently inserted durin e design or production phase, which can carr unauthorized actions when its *triggering co s* are satisfied.

See lecture:
HS_1.7 - Hardware Trojans


Hardware Vulnerabilities
Nature
Source
Abstraction level
Unintentional
Bugs
Flaws
Intentional
Backdoors
Specifications
Design choices
Architectural level
Test Infrastructures
Used EDA environments
Adopted technologies
Side-Channel Effects
System
Architecture
Chip/Device
IP core

Examples

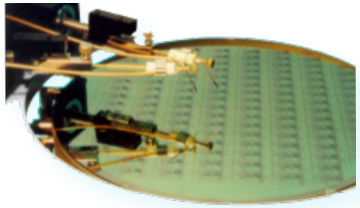
- Microarchitectural Flaws presented above all stem from choices made at the architectural level design


Hardware Vulnerabilities
Nature
Source
Abstraction level
Unintentional
Bugs
Flaws
Intentional
Backdoors
Specifications
Design choices
Architectural level
Test Infrastructures
Used EDA environments
Adopted technologies
Side-Channel Effects
System
Architecture
Chip/Device
IP core

Unprotected Test Infrastructures can Jeopardize the Security of the Entire System

Test vs Security

- Testing is mandatory to guarantee high quality of digital ICs
  - Increase controllability and observability



- At a same time, security fears testability
  - Test infrastructures used for attacks

Do we need to test secure circuits?

- Of course, yes!
- In general, we have to guarantee high quality
- In particular, a defective secure device may jeopardize the overall safety & security

Do we need to test secure circuits?

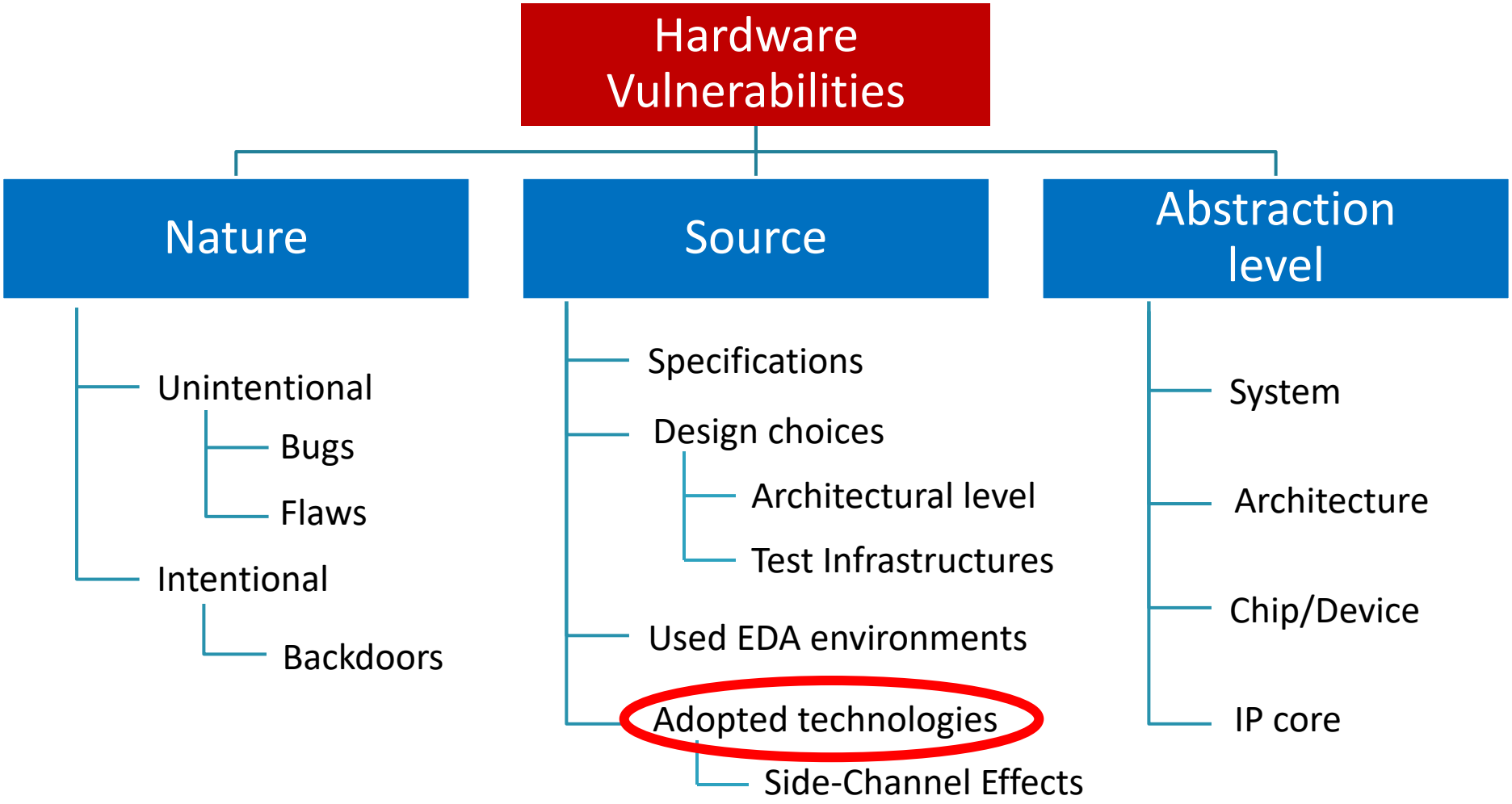
- Of course, yes!
- In general, we have uarantee high quality
- In particular, a defe secure device may jeopardize the ov ety & security

For further details, please refer to the lecture:
HW_S_0.7.1 – Hardware testing - Basic concepts

Potential Avenues of Attack

- Among the plethora of test infrastructures, we just mention here
  - Scan chains
  - Standard IEEE 1149.1
  - JTAG infrastructures
  - ATPG

See lecture:
HS_2.1 - Vulnerabilities in Test Infrastructures


Hardware Vulnerabilities
Nature
Source
Abstraction level
Unintentional
Bugs
Flaws
Intentional
Backdoors
Specifications
Design choices
Architectural level
Test Infrastructures
Used EDA environments
Adopted technologies
Side-Channel Effects
System
Architecture
Chip/Device
IP core

Consequences

- A same design implemented resorting to different technologies can show different vulnerabilities.

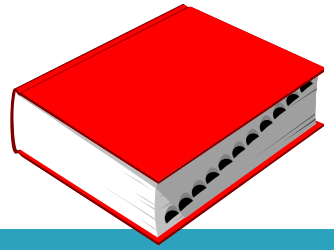
Examples

- *Row Hammer in DRAM memories*
- *Data remanence*

Row Hammer in DRAM memories

- Caused by the physical and intrinsic phenomenon of electric coupling between memory cells due to the used technology

Data remanence



- Is the residual information remaining on storage media after clearing, i.e., after the foreseen actions to remove or erase the target data.

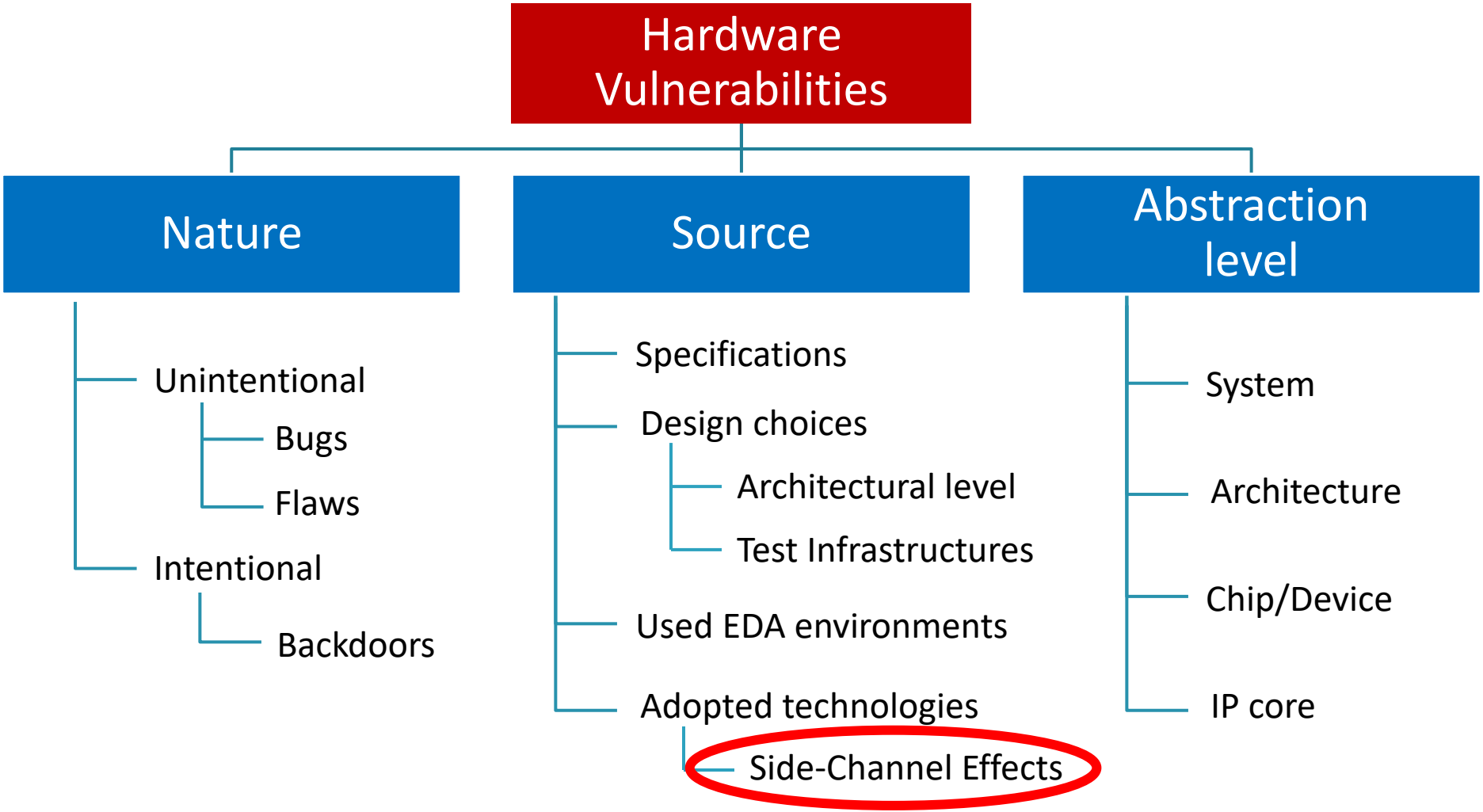
Residue origins

- This residue may result from:
  - data being left intact by a nominal file deletion operation
  - by reformatting of storage media that does not remove data previously written to the media
  - through physical properties of the storage media that allow previously written data to be recovered

Residue origins

- This residue may result from:
  - data being left intact by a nominal file deletion operation
  - by reformatting of storage media that does not remove data previously written to the media
  - through physical properties of the storage media that allow previously written data to be recovered

Some examples
are in the Insight #2


Hardware Vulnerabilities
Nature
Source
Abstraction level
Unintentional
Bugs
Flaws
Intentional
Backdoors
Specifications
Design choices
Architectural level
Test Infrastructures
Used EDA environments
Adopted technologies
Side-Channel Effects
System
Architecture
Chip/Device
IP core

Side-Channel Effects

- Hardware devices unintentionally release in the surrounding environment several clues:

Side-Channel Effects

- Hardware devices unintentionally release in the surrounding environment several clues:
  - Spent time
  - Spent energy
  - Emitted electromagnetic radiation
  - Emitted Noise
  - Emitted Light
  - …

Side-Channel Effects

- Hardware devices unintentionally release in the surrounding environment several clues:
  - Spent time
  - Spent energy
  - Emitted electromagnetic radiation
  - Emitted Noise

See lecture:
HS_3.1 - Side Channel Attacks

Conclusions

- Once a vulnerability has been identified, it has to be patched as soon and as effectively as possible
- The possible solutions are conditioned by different factors and must be studied and identified case by case.

Hardware Vulnerabilities -- How to fix them

- Except in very special cases, vulnerabilities found at the hardware level can be fixed only in later versions of devices
- Where possible, attempts are made to exploit software to mitigate the consequences.

Paolo PRINETTO

Director
CINI Cybersecurity
National Laboratory

Paolo.Prinetto@polito.it

Mob. +39 335 227529

CYBER CHALLENGE.IT

CYBERSECURITY NATIONAL LABORATORY

https://cybersecnatlab.it

Insight #1 – *Hardware Bug Examples*

- The *F00F Pentium P5 Bug*
- The *Cyrix coma bug*

Example: “F00F Pentium P5 Bug”

- Detected in 1997 in all Pentium P5 processors
- In the x86 architecture, the byte sequence F0 0F C7 C8 represents the instruction lock cmpxchg8b eax (locked compare and exchange of 8 bytes in register EAX) and does not require any special privilege.
- Hoverer, the instruction encoding is invalid. The cmpxchg8b instruction compares the value in the EDX and EAX registers (the lower halves of RDX and RAX on more modern x86 processors) with an 8-byte value in a memory location.
- In this case, however, a register is specified instead of a memory location, which is not allowed.

Example: “F00F Pentium P5 Bug”

- Under normal circumstances, this would simply result in an exception
- However, when used with the lock prefix (normally used to prevent two processors from interfering with the same memory location), the CPU erroneously uses locked bus cycles to read the illegal instruction exception-handler descriptor.
- Locked reads must be followed by locked writes, and the CPU's bus interface enforces this by forbidding other memory accesses until both actions are completed.
- As none are forthcoming (since, due to the locking, write cannot take place), after performing these bus cycles all CPU activity stops, and the CPU must be reset to recover.
- The instruction can be exploited for a DoS attack.

Example: Cyrix coma bug

- The Cyrix coma bug is a design flaw in Cyrix 6x86, 6x86L, and early 6x86MX processors that allows a non privileged program to hang the computer.

Example: The Cyrix coma bug

- This C program (which uses inline x86-specific assembly language) could be compiled and run by an unprivileged user:

```
unsigned char
c[4] = {0x36, 0x78, 0x38, 0x36};
int main()
{
asm (
"         movl $c, %ebx\n"
"again:   xchgl (%ebx), %eax\n"
"         movl %eax, %edx\n"
"         jmp again\n"
     );
}
```

Example: The Cyrix coma bug

- On executing this program, the processor enters an infinite loop that cannot be interrupted.
- This allows any user with access to a Cyrix system with this bug to perform a DoS attack.

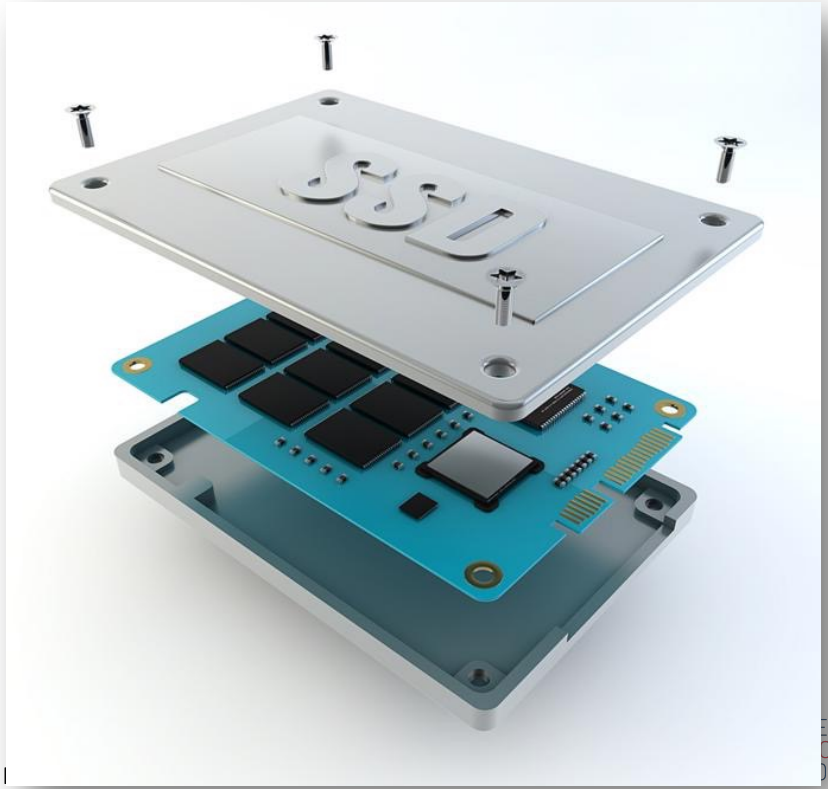
Insight #2 –
Data remanence Examples

Data Remanence Attacks

- We shall consider 3 technologies:
  - SSDs
  - Hard Disks
  - SRAMs & DRAMs

Data Remanence Attacks

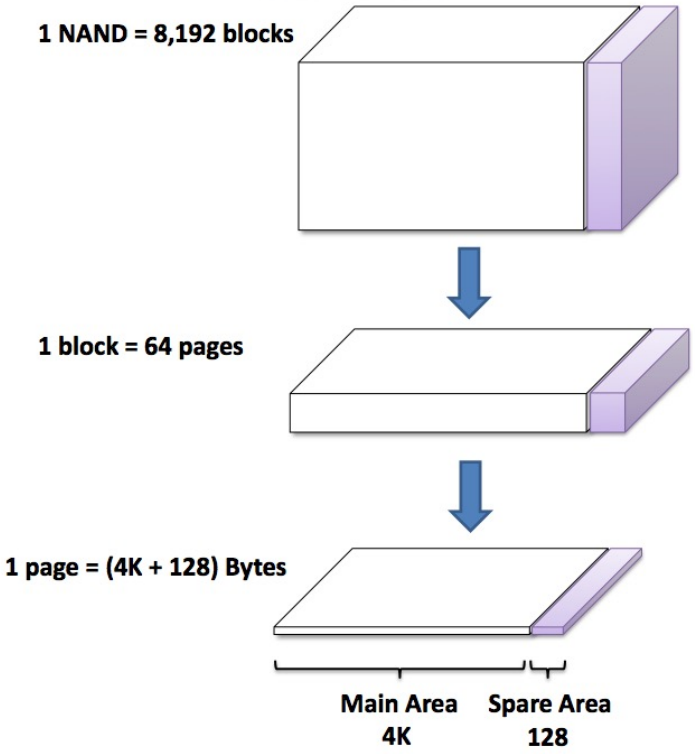
- We shall consider 3 technologies:
  - SSDs
  - Hard Disks
  - SRAMs & DRAMs



SSD (Solid State Drive) implementation

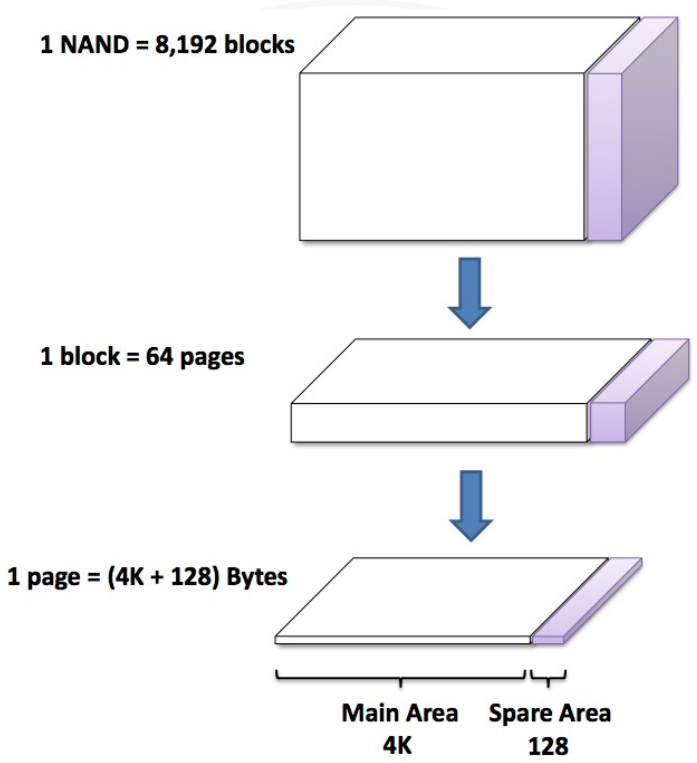
- SSDs rely on NAND Flash memories:
  - High-density
  - Low-cost data storage
  - Low-power consumption
  - Low endurance

Logical Organization of a NAND



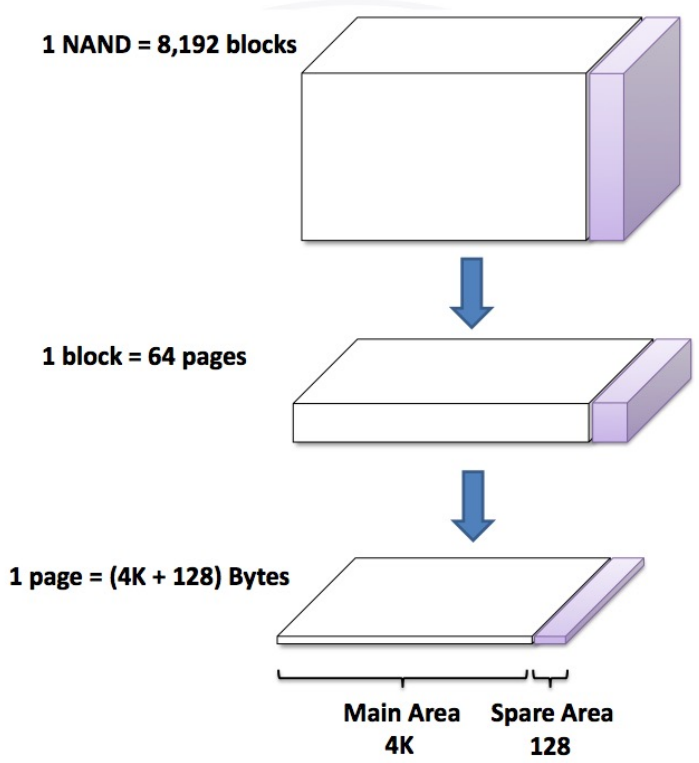
Flash Page

- Logical pages are composed of cells belonging to a same wordline
- The page is the smallest storage unit when performing read and programming operations



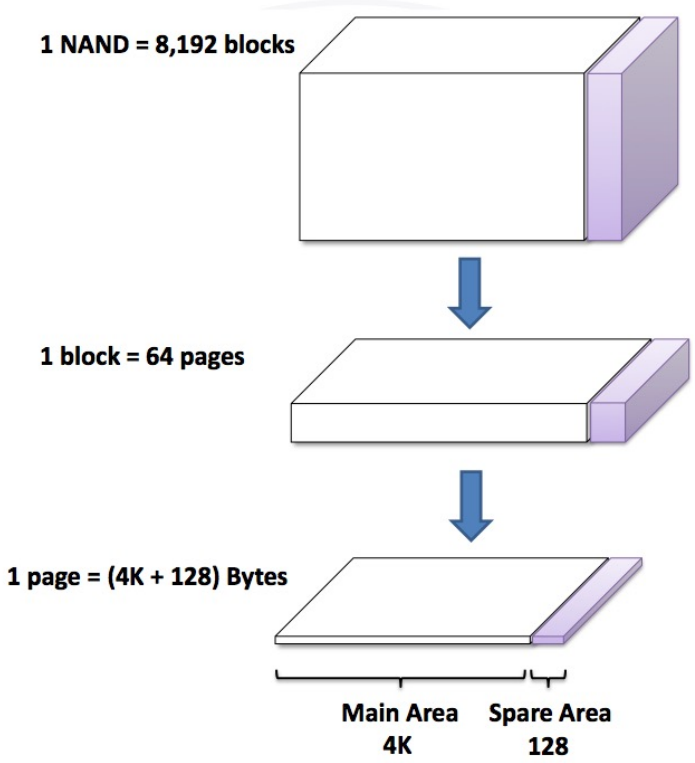
Flash Page

- Pages that already contain data must be erased before writing new data
- Typical page sizes:
  - 2 kB (data) + 64 B (spare) for SLC
  - 4 kB (data) + 128 B (spare) for MLC



Flash Block

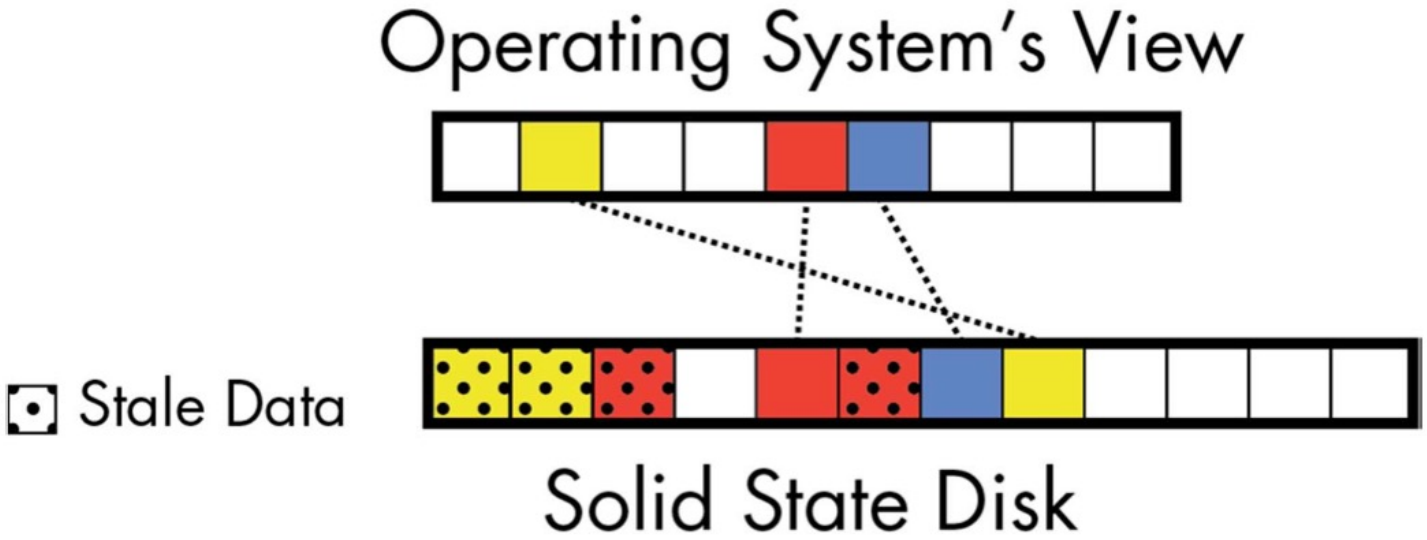
- A block is a set of Pages
- It's the minimum portion of the memory that can be erased
- To erase a page, you have to erase the whole Block it belongs to
- Typical Block size = 64 or 128 pages



Consequences

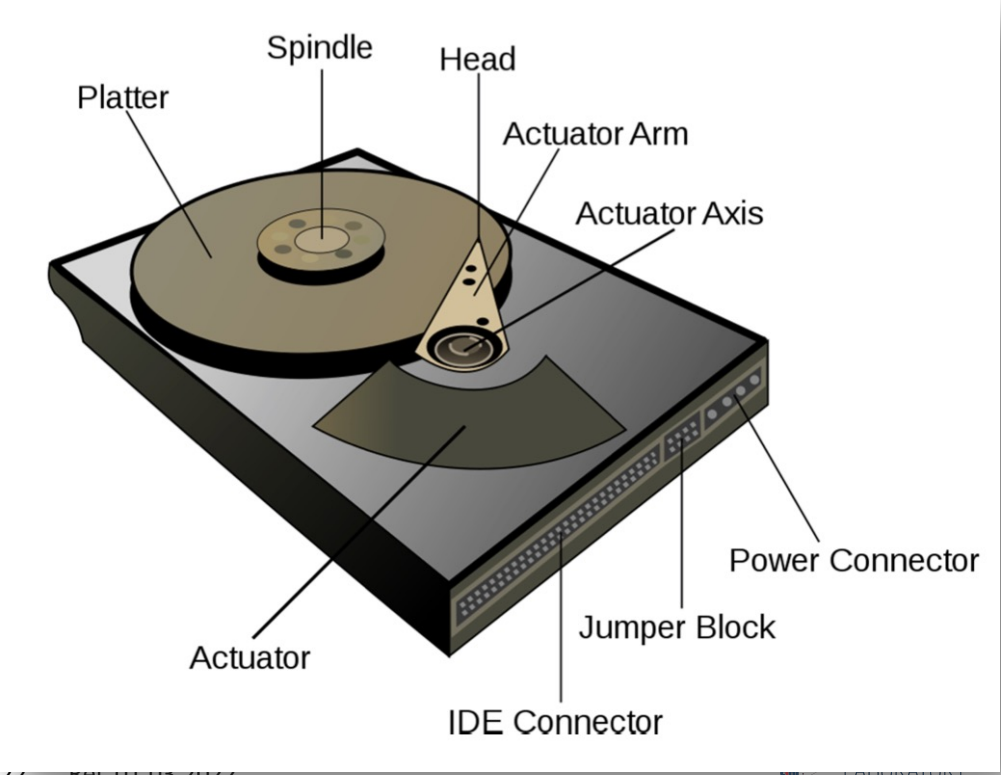
- A lot of *Stale data* on SSDs

Stale data on SSDs

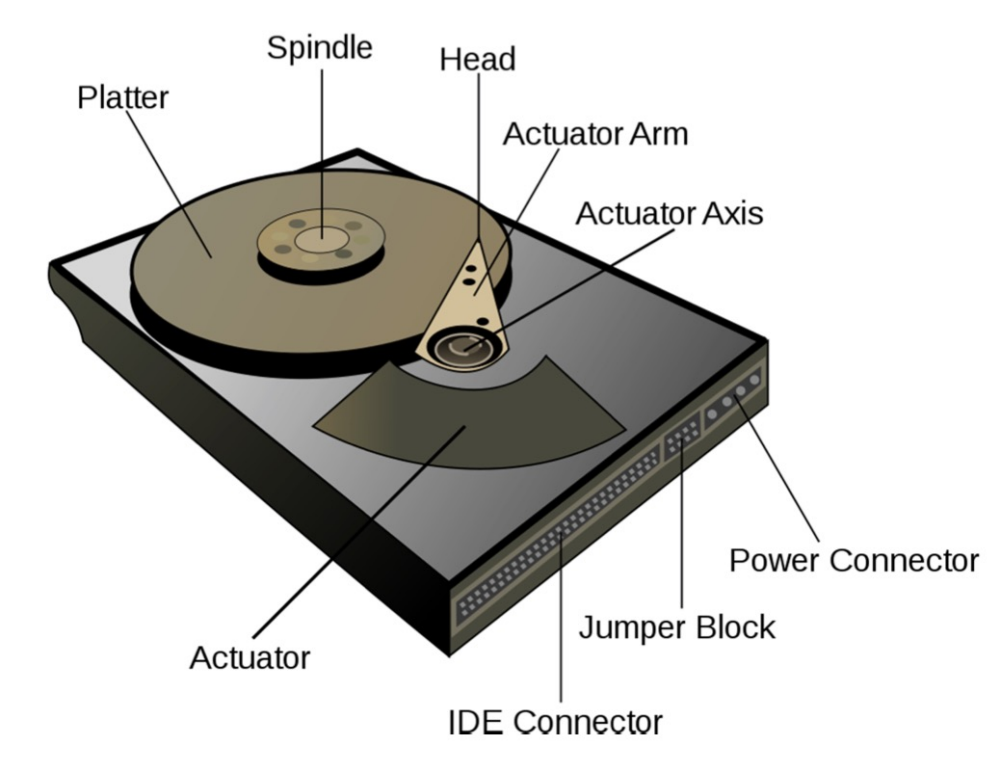


Data Remanence Attacks

- We shall consider 3 technologies:
  - SSDs
  - Hard Disks
  - SRAMs & DRAMs



Magnetic Hard Disks



Data on Hard Disk

- Write heads used on HDs differ slightly in position and width due to manufacturing tolerances.
- As a result, one writer might not overwrite the entire area on a medium that had previously been written to by a different device.
- Normal read heads will only give access to the most recently written data, but special high-resolution read techniques (e.g., magnetic-force microscopy) can give access to older data that remains visible near the track edges.

Data Remanence Attacks

- We shall consider 3 technologies:
  - SSDs
  - Hard Disks
  - SRAMs & DRAMs

Data in SRAM

- Data remanence has been observed in static random-access memory (SRAM), which is typically considered volatile (i.e., the contents degrade with loss of external power).
- In one study, data retention was observed even at room temperature.

Data in DRAM

- A study found data remanence in DRAM with data retention of seconds to minutes at room temperature and “a full week without refresh when cooled with liquid nitrogen”, i.e., at temperatures from -50° C down.