Standards We Love
(*for assuring and verifying safety-critical systems*)

Heidy Khlaaf

Adelard

About Me

Consultant

Formal Verification
Evaluating, designing, specifying, and verifying safety-critical systems

Safety & Security Assessments
Carrying out safety and security assessments of clients' systems

Production of Standards
Production of standards and guidelines for safety and security related applications and their development

Background

- Comp Sci PhD at UCL 2017– Temporal logic model-checking and verification of software systems.
- As a PhD student, I collaborated with Microsoft Research Cambridge to create and extend the T2 tool to support temporal property verification.
- Focused on verifying Windows device-drivers in hopes of preventing a multitude of errors.

Safety-Critical Systems

What are they?

Systems whose failure or malfunction may result in the following outcomes: death or serious injury to people, loss or severe damage to equipment/property, and environmental harm.

- **Aviation**
- **Nuclear Energy**
- **Automative & Rail**
- **Defense, Medical, Finance, Autonomous Systems**

Formal Verification

“Program testing can be used to show the presence of bugs, but never to show their absence!” - Edsger W. Dijkstra

Formal Verification

Definition

The process of establishing whether a system satisfies some requirements (properties), using formal methods of mathematics.

Model

Formally model a system's implementation in the appropriate framework.

Specify

Formally specify a system's requirement in an expressive specification language.

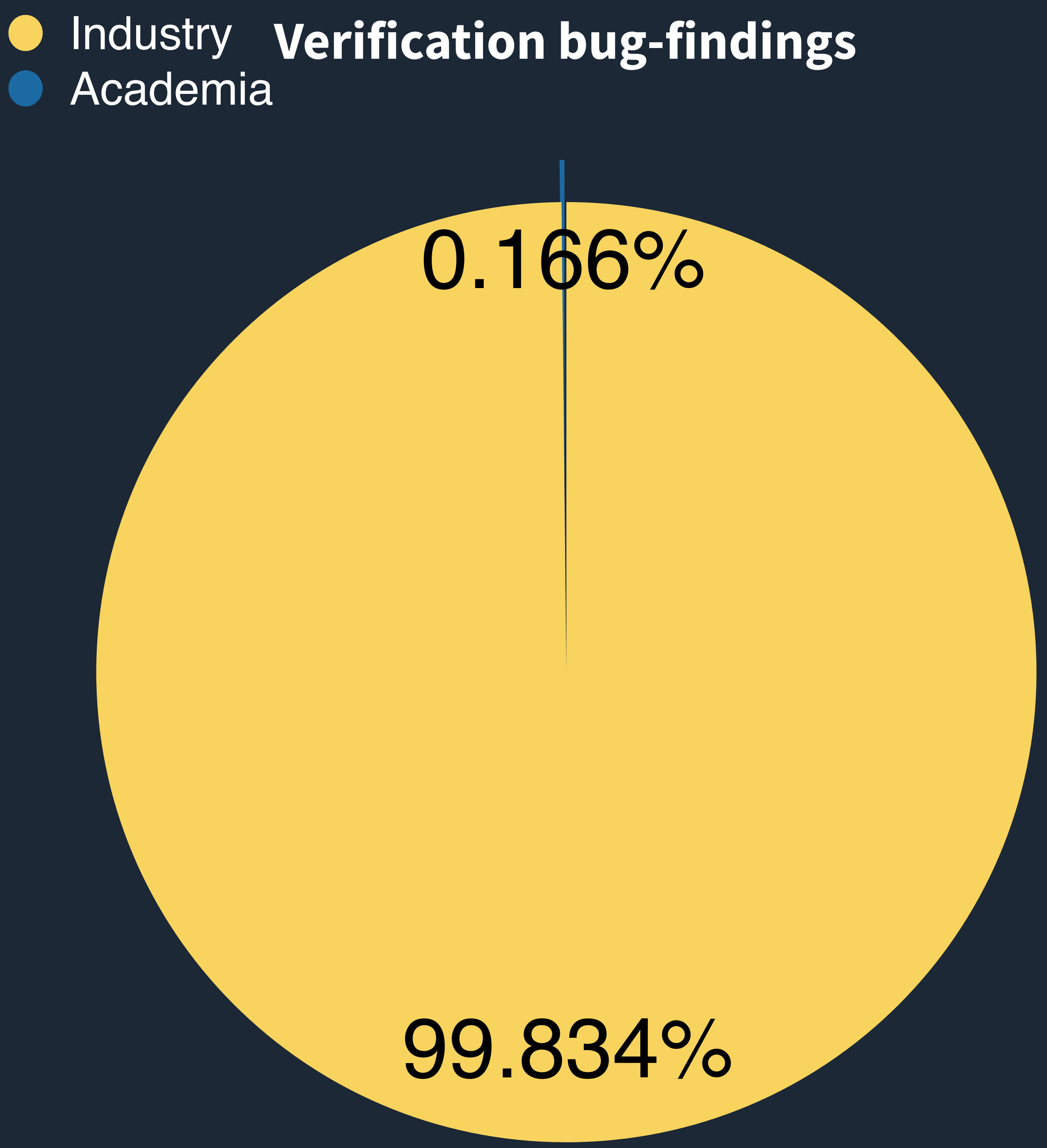
Model-Check

Exhaustive exploration of a system's mathematical model against its specifications.

Analyze

If not all specifications were satisfied, analyze findings and their consequences.

Bug Findings



01 **Academia**
Invested in producing more nuanced and expressive formal frameworks, despite lack of scalability and applicability.

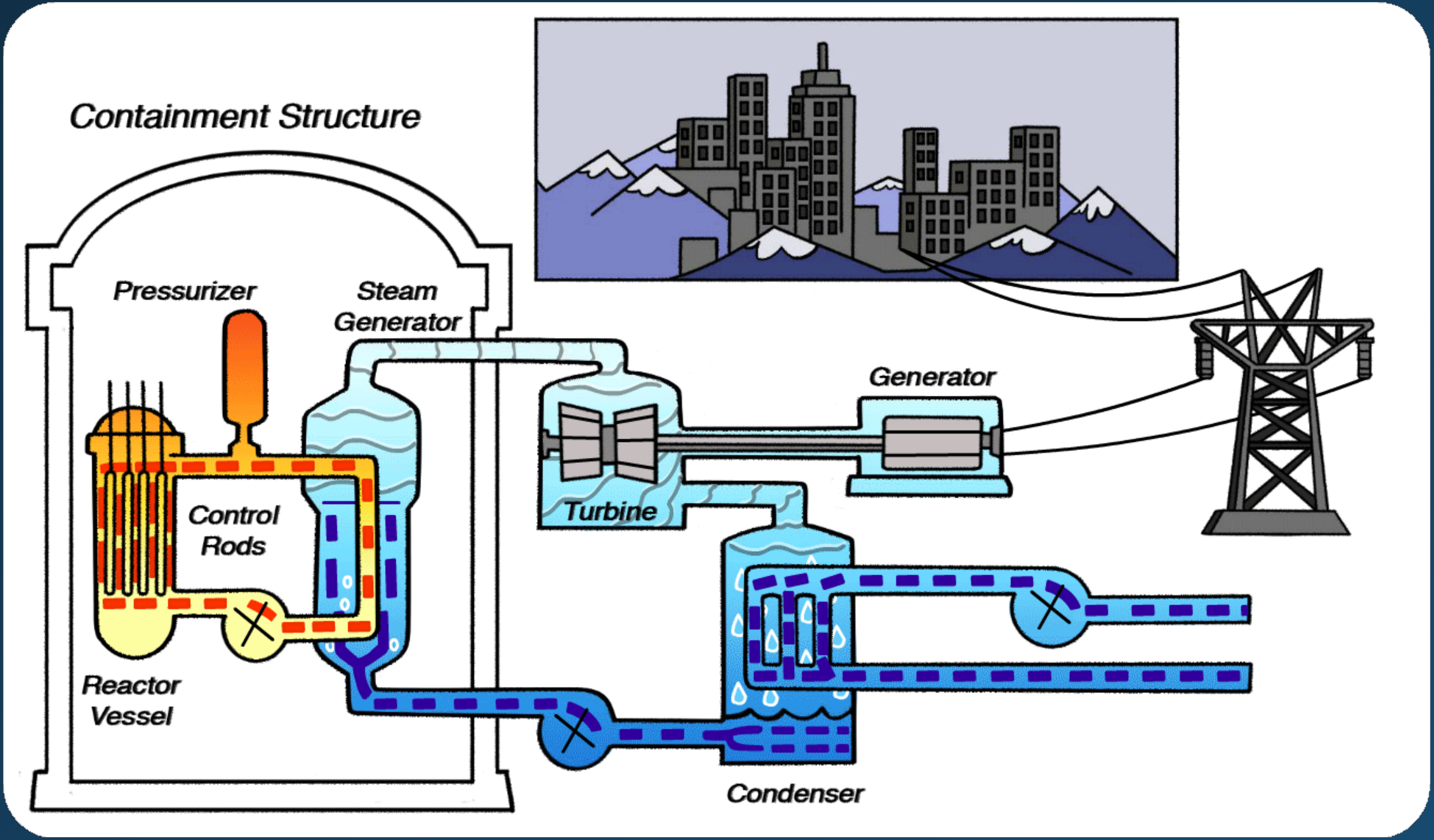
02 **Industry**
Industrial formal verification tools are adept in finding thousands of bugs on millions of lines of code.

How are we not dead yet?

- How software in safety-critical systems is assured. 01
- Verification of a specific type of component that is one of the biggest challenges in the nuclear industry – smart devices. 02
- Lessons learned from standards, guidelines, and processes used in safety-critical verification and assurance. 03

Verifying Nuclear Power Plants

Pressurized Water Reactor



Smart Sensors

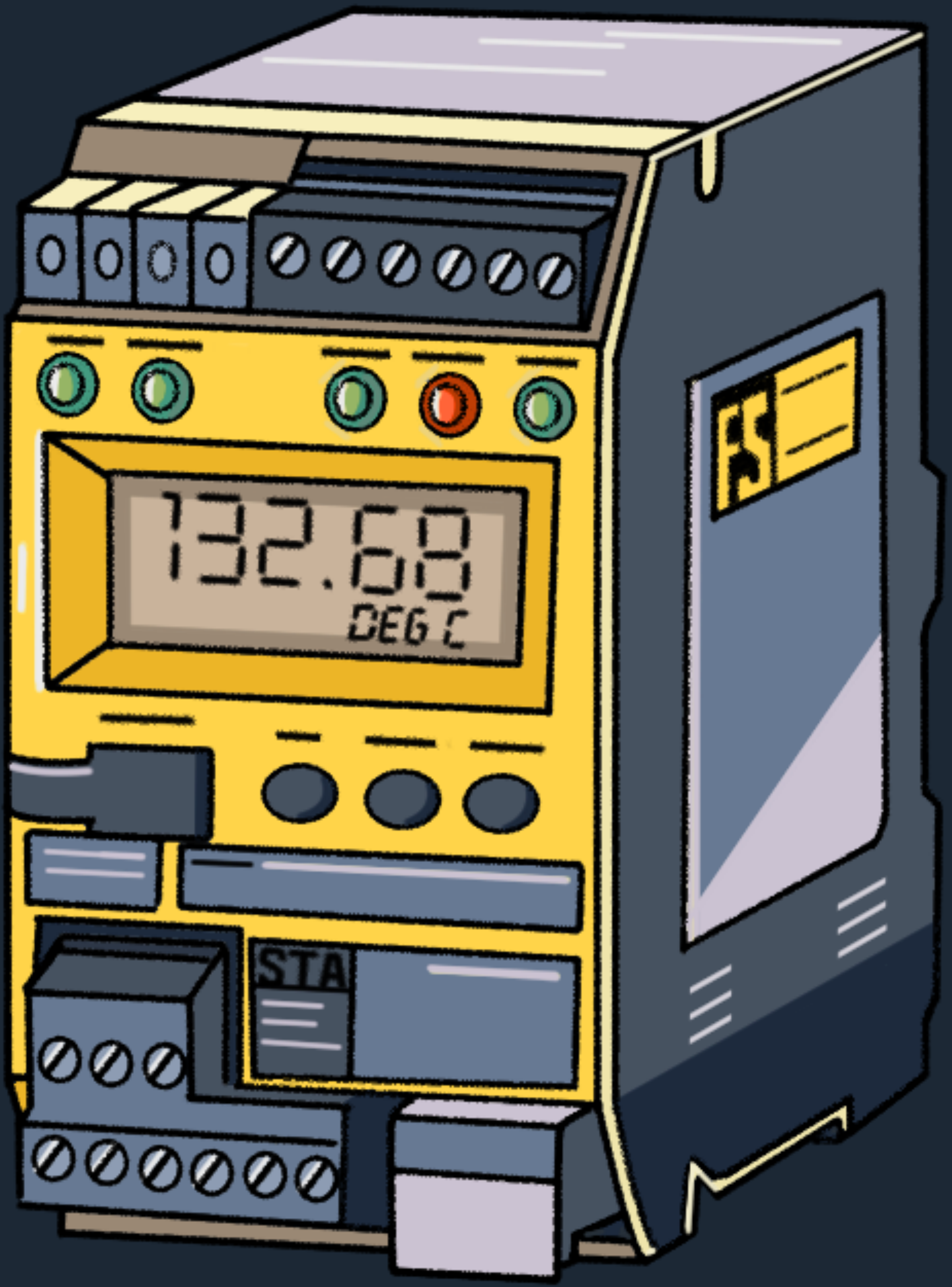
Embedded devices

Written in C & Assembly
- Can include FPGAs
- No underlying OS
- Interrupt driven (e.g., time-triggered architecture)
- Use of pointers
- Communication protocols

Use of compilers vary depending on processor and hardware

Not written to be verified
- Specification may not be available

10k, 100k, 1000k, 10000k, … LoC

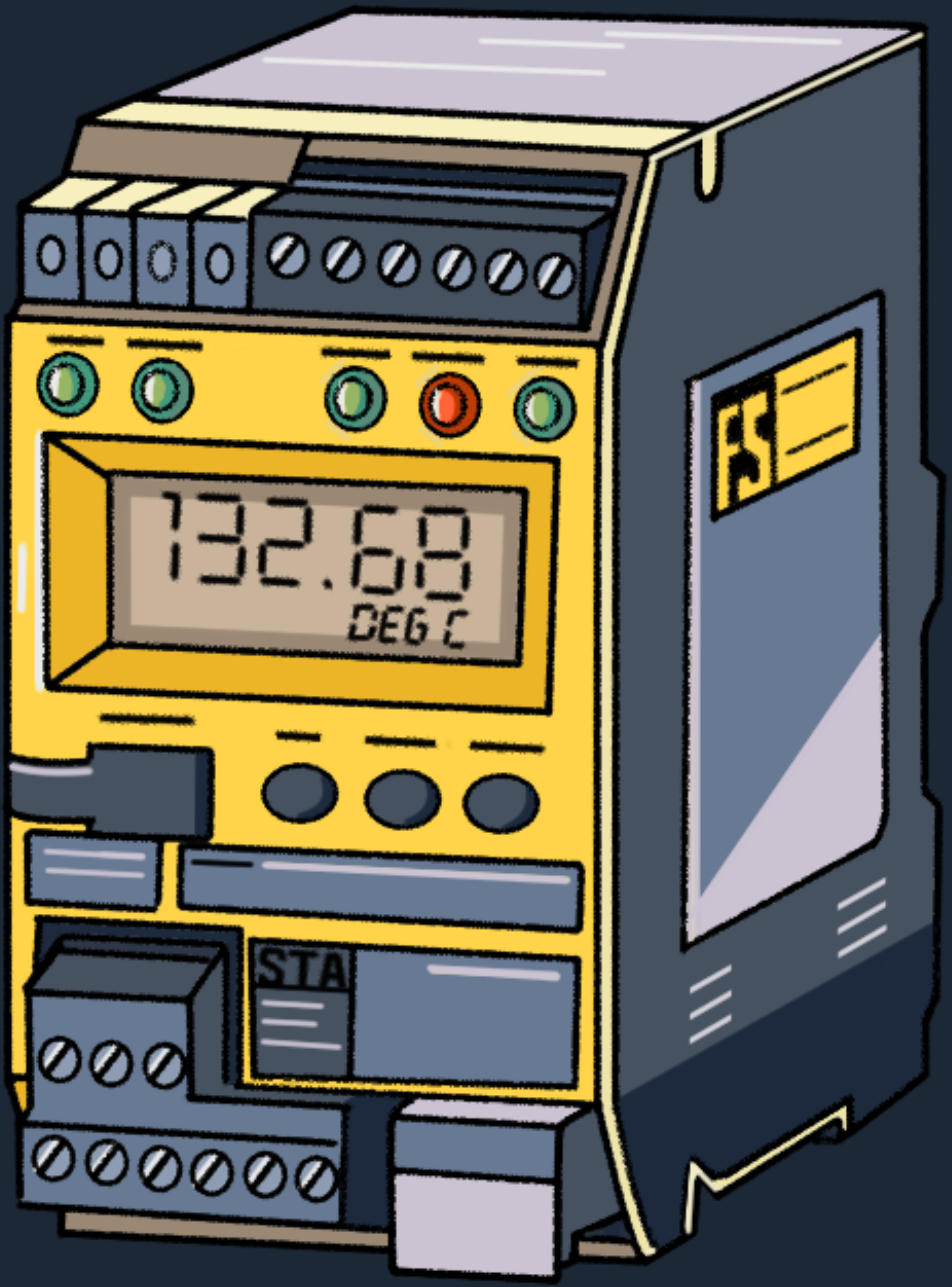


Smart Sensors

- Embedded applications: reading and writing port data, setting timer registers and reading input captures, etc.

- ARM C51 has special data types sfr and sbit that customise the compiler to the target processor:

```
sfr P0 0x80
sfr P1 0x81
sfr ADCON 0xDE
sbit EA 0x9F
…
timer0_int() interrupt 1 using 2 {
   unsigned char temp1 ;
   unsigned char temp2 ;
   ADCON = 0x08 ; /* Write data to register */
   P1 = 0xFF ; /* Write data to Port */
   io_status = P0 ; /* Read data from Port */
   EA = 1; /* Set bit(enable all interrupts)*/ }
```



Why Smart Sensors?

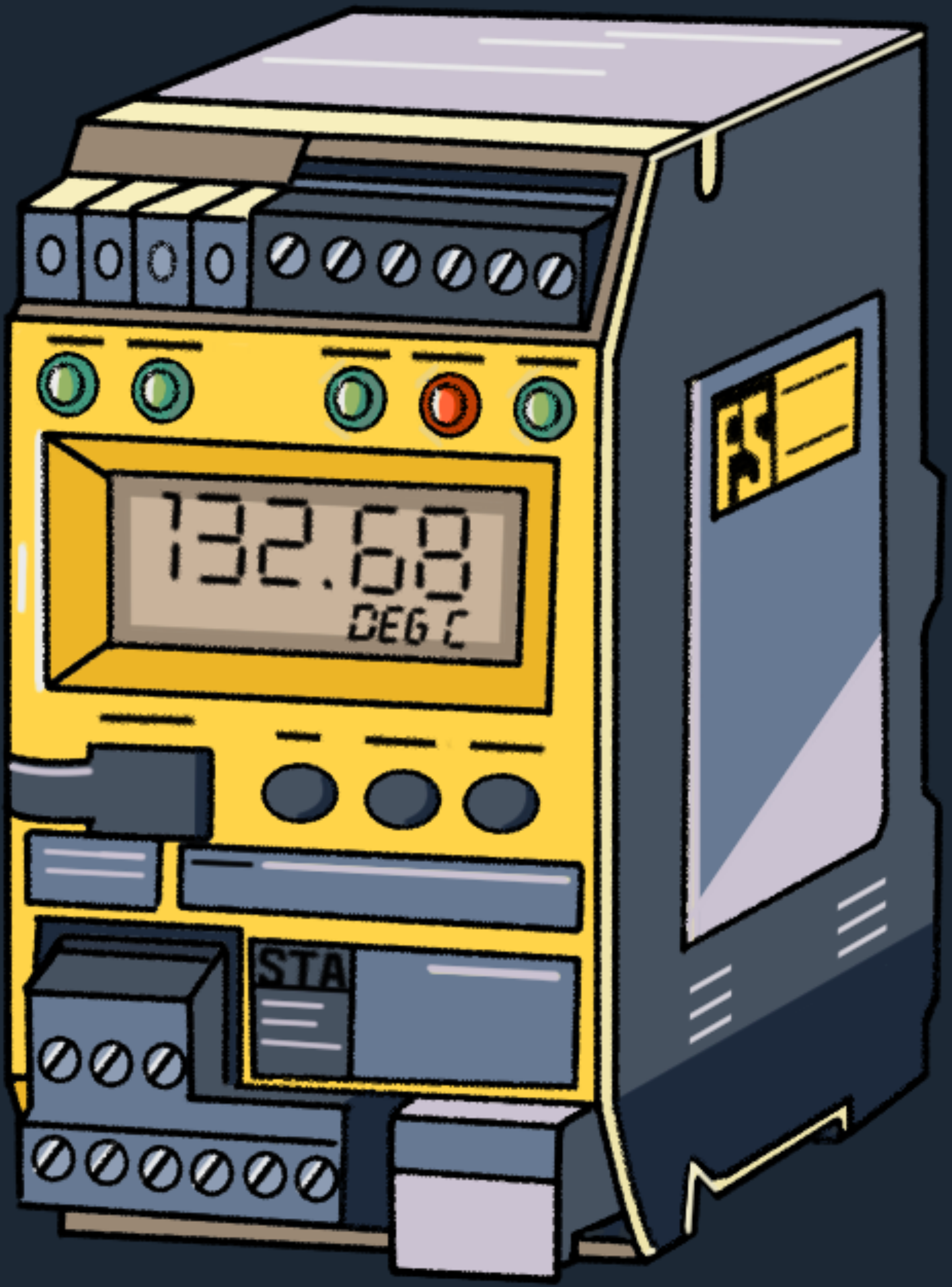
Pure analogue sensors disappearing

Improved functionality

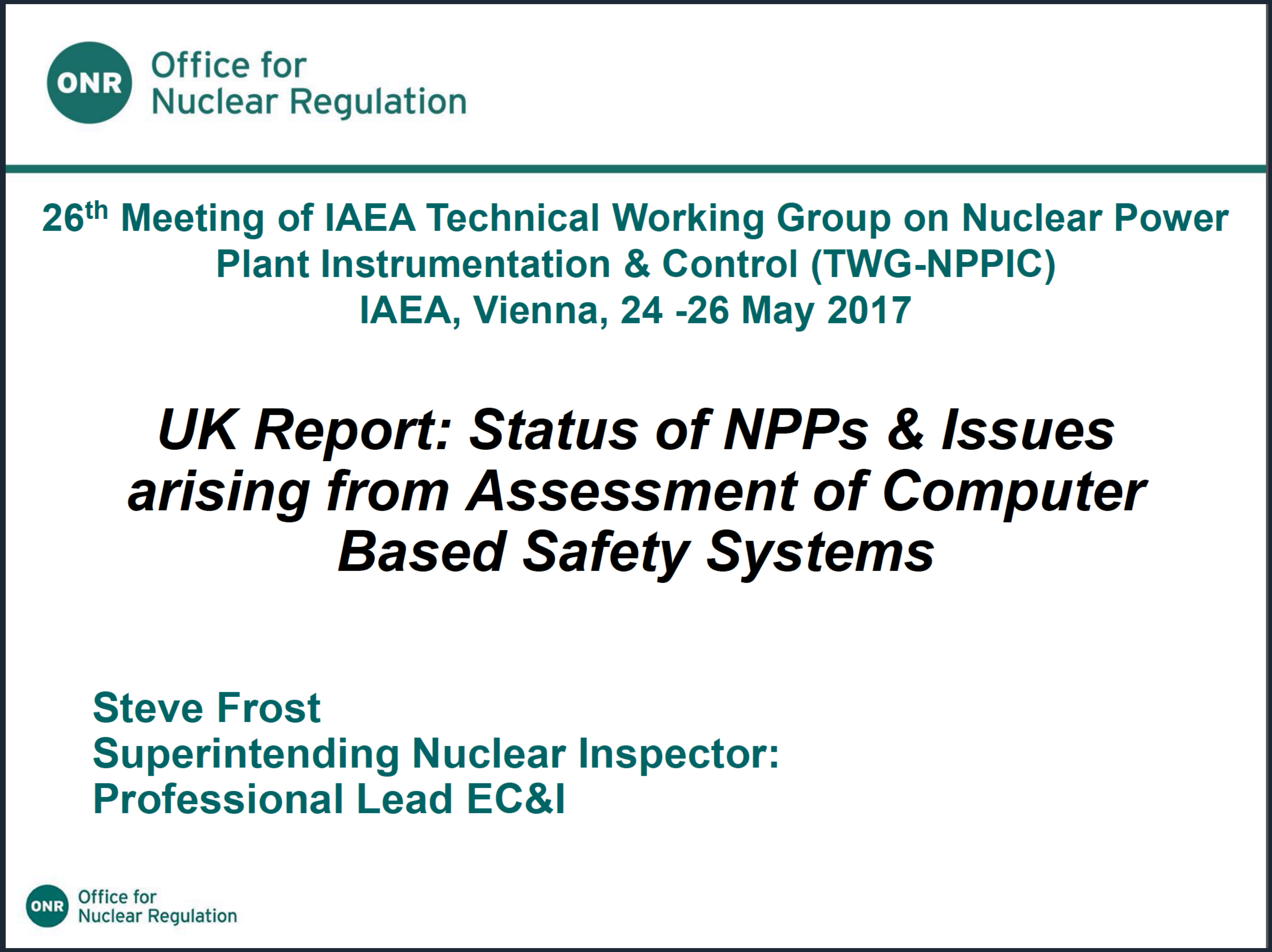
- Microprocessors or micro-controllers
- Better accuracy, calibration, diagnostics
- Configurable but not programmable
- Efficient and fast

Industrial embedded devices

- Commercial-Off-The-Shelf (COTS)
- Perform a defined function
- Examples include
  - Temperature transmitters
  - Pressure transmitters
  - Voltage regulators



Verifying Nuclear Power Plants



Office for
Nuclear Regulation

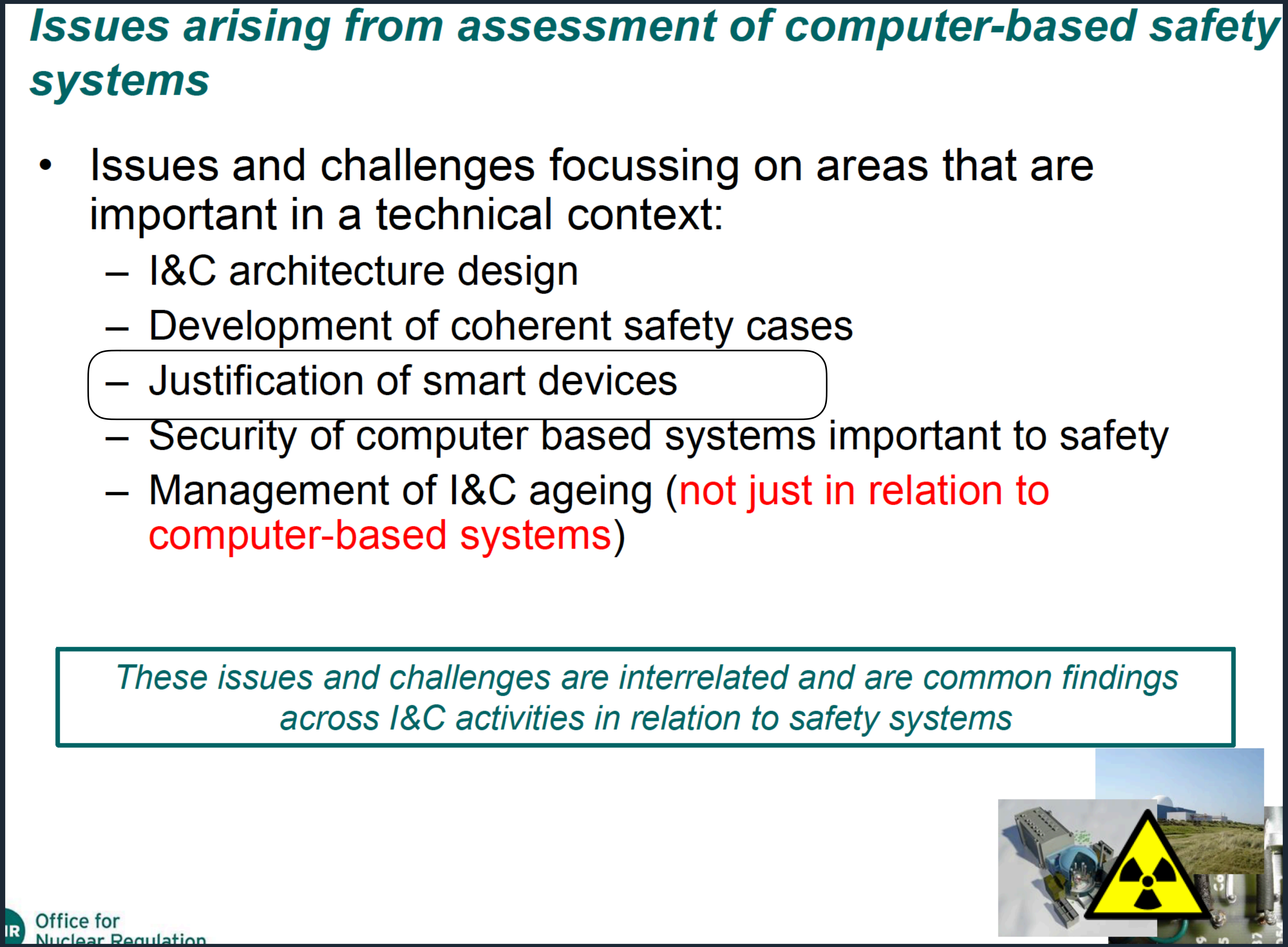
26th Meeting of IAEA Technical Working Group on Nuclear Power Plant Instrumentation & Control (TWG-NPPIC)
IAEA, Vienna, 24 -26 May 2017

UK Report: Status of NPPs & Issues arising from Assessment of Computer Based Safety Systems

Steve Frost
Superintending Nuclear Inspector:
Professional Lead EC&I

Office for
Nuclear Regulation

Verifying Nuclear Power Plants



Issues arising from assessment of computer-based safety systems

- Issues and challenges focussing on areas that are important in a technical context:
  - I&C architecture design
  - Development of coherent safety cases
  - Justification of smart devices
  - Security of computer based systems important to safety
  - Management of I&C ageing (not just in relation to computer-based systems)

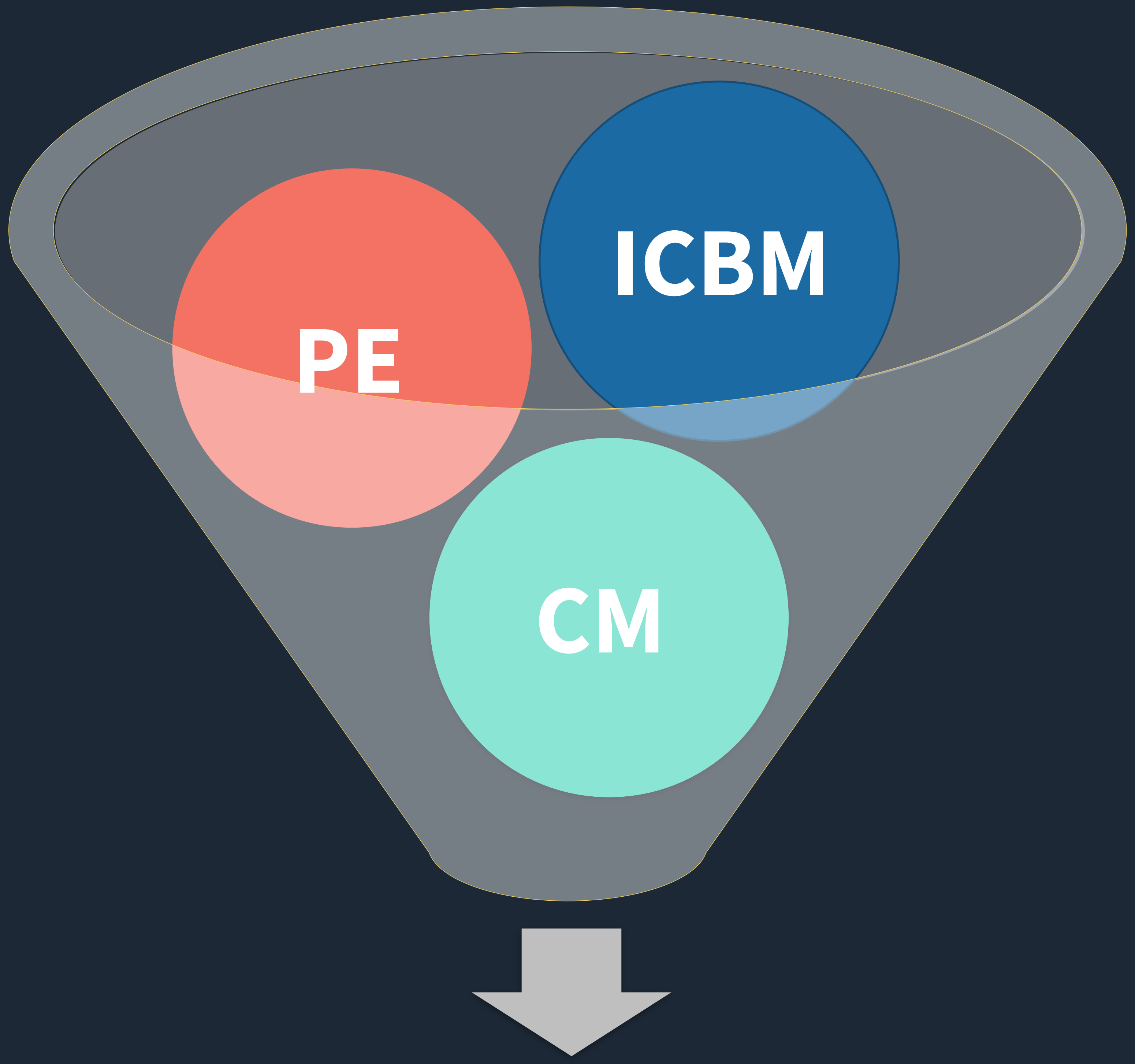
These issues and challenges are interrelated and are common findings across I&C activities in relation to safety systems

Office for Nuclear Regulation

Safety Assessment Principles

ONR SAPs

The primary principles that define the overall approach for nuclear installations in the UK



Justification of use of software

Production Excellence

Demonstration of excellence in all aspects of production from the initial specification through to the finally commissioned system

Compensation Measures

Weaknesses that are identified and are to be compensated for from PE

ICBM

Independent and thorough assessment of a safety system's fitness for purpose

Production Excellence

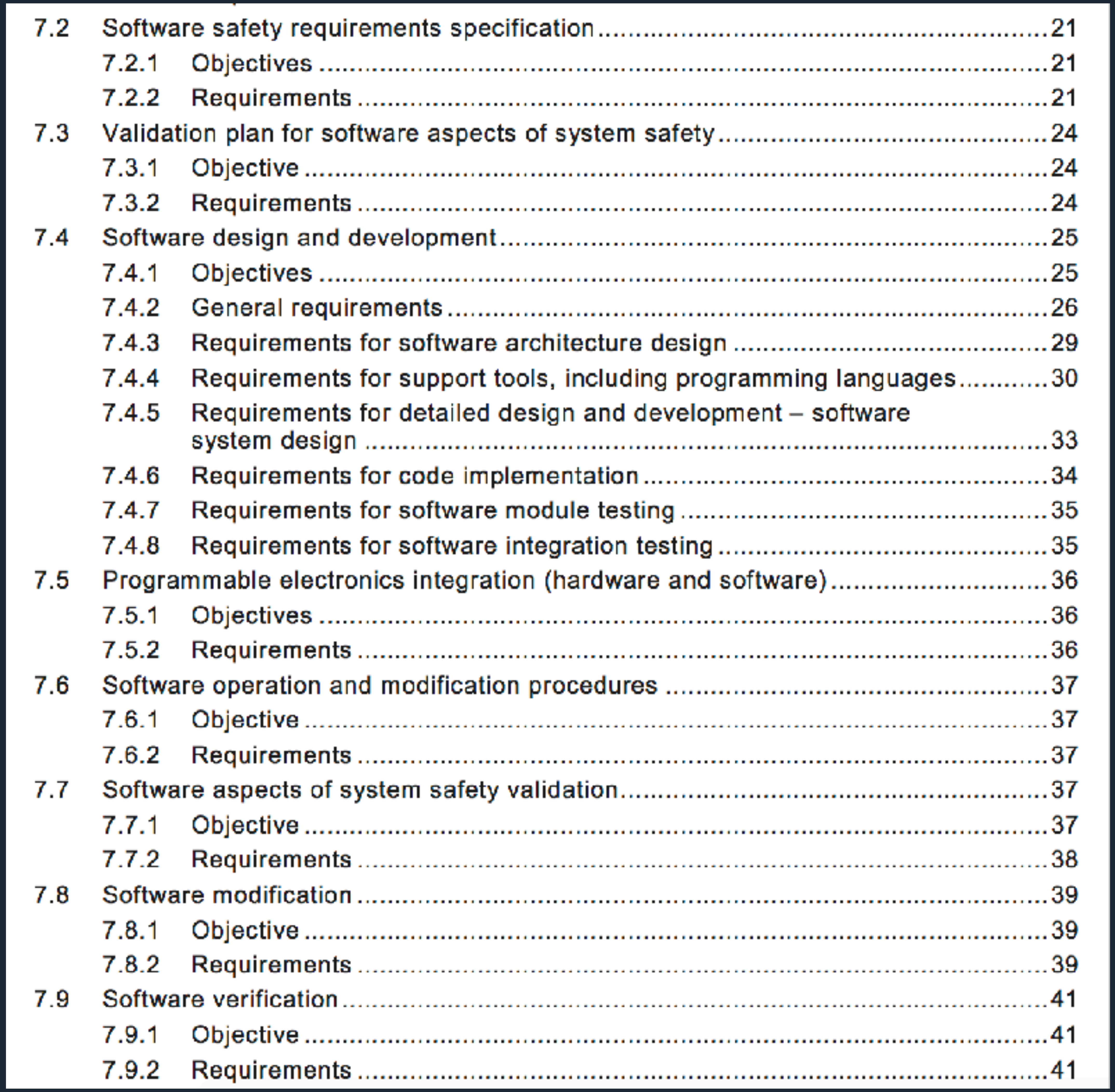
- Application of best design practice consistent with accepted standards for the development of software for computer-based safety systems (e.g, following standards such as *IEC 61508, IEC 61513, ISO 26262, DO 331*)
- Implementation of a modern standards quality management system (e.g., *ISO 9001, Github, ClearQuest, ClearCase, etc.*)
- Application of a comprehensive testing program formulated to check every system function

Independent Confidence-Building Measures

- Complete and diverse checking of the production software by a team that is independent of the systems suppliers (e.g., formal verification, static analysis)

- Independent assessment of the comprehensive testing program covering the full scope of the test activities

IEC 61508 - The “Golden” Boy Safety Standard



IEC 61508 - The "Golden" Boy Safety Standard

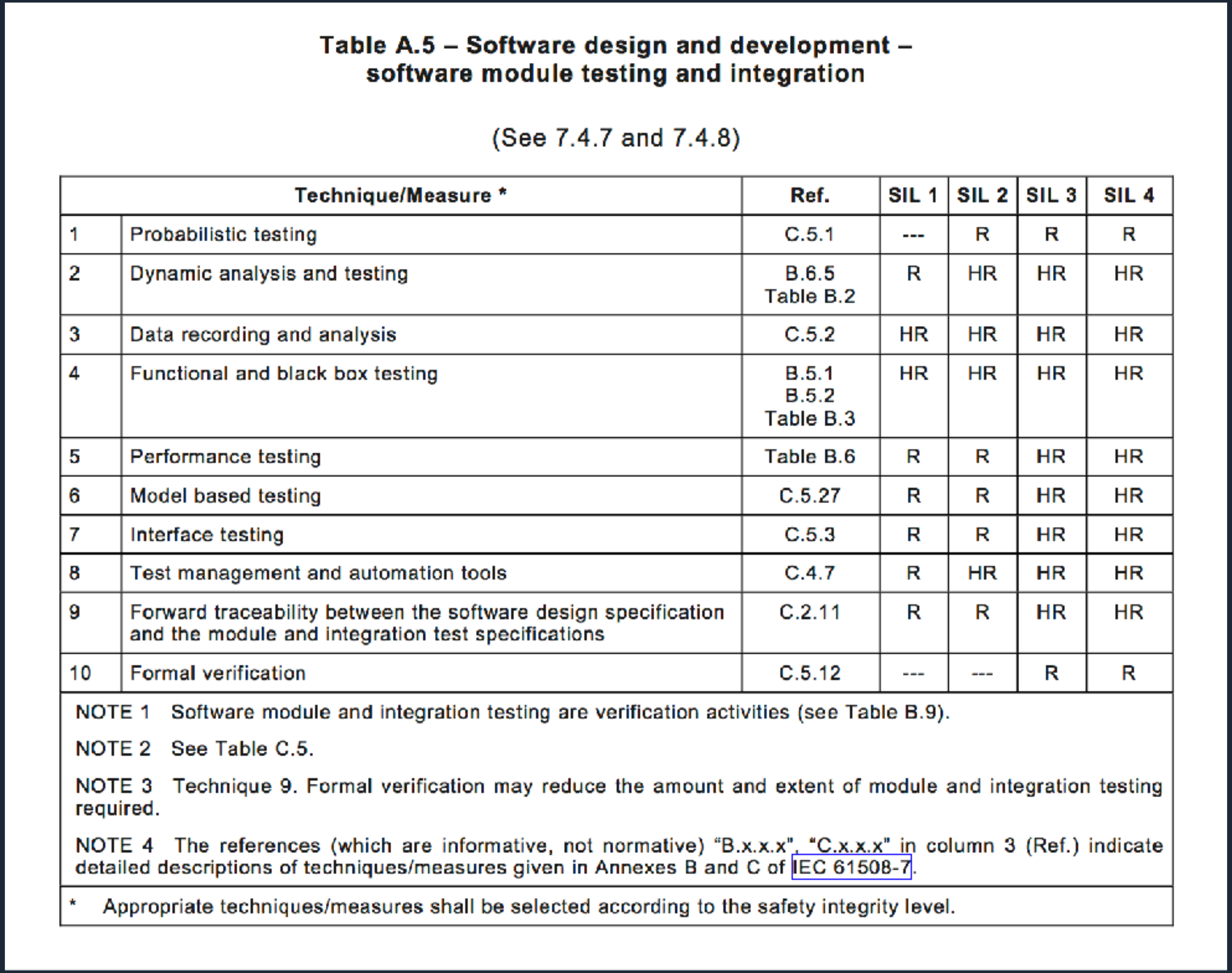


Table A.5 – Software design and development – software module testing and integration

(See 7.4.7 and 7.4.8)

	Technique/Measure *	Ref.	SIL 1	SIL 2	SIL 3	SIL 4
1	Probabilistic testing	C.5.1	---	R	R	R
2	Dynamic analysis and testing	B.6.5 Table B.2	R	HR	HR	HR
3	Data recording and analysis	C.5.2	HR	HR	HR	HR
4	Functional and black box testing	B.5.1 B.5.2 Table B.3	HR	HR	HR	HR
5	Performance testing	Table B.6	R	R	HR	HR
6	Model based testing	C.5.27	R	R	HR	HR
7	Interface testing	C.5.3	R	R	HR	HR
8	Test management and automation tools	C.4.7	R	HR	HR	HR
9	Forward traceability between the software design specification and the module and integration test specifications	C.2.11	R	R	HR	HR
10	Formal verification	C.5.12	---	---	R	R

NOTE 1 Software module and integration testing are verification activities (see Table B.9).

NOTE 2 See Table C.5.

NOTE 3 Technique 9. Formal verification may reduce the amount and extent of module and integration testing required.

NOTE 4 The references (which are informative, not normative) "B.x.x.x", "C.x.x.x" in column 3 (Ref.) indicate detailed descriptions of techniques/measures given in Annexes B and C of IEC 61508-7.

* Appropriate techniques/measures shall be selected according to the safety integrity level.

IEC 61508 - The "Golden" Boy Safety Standard

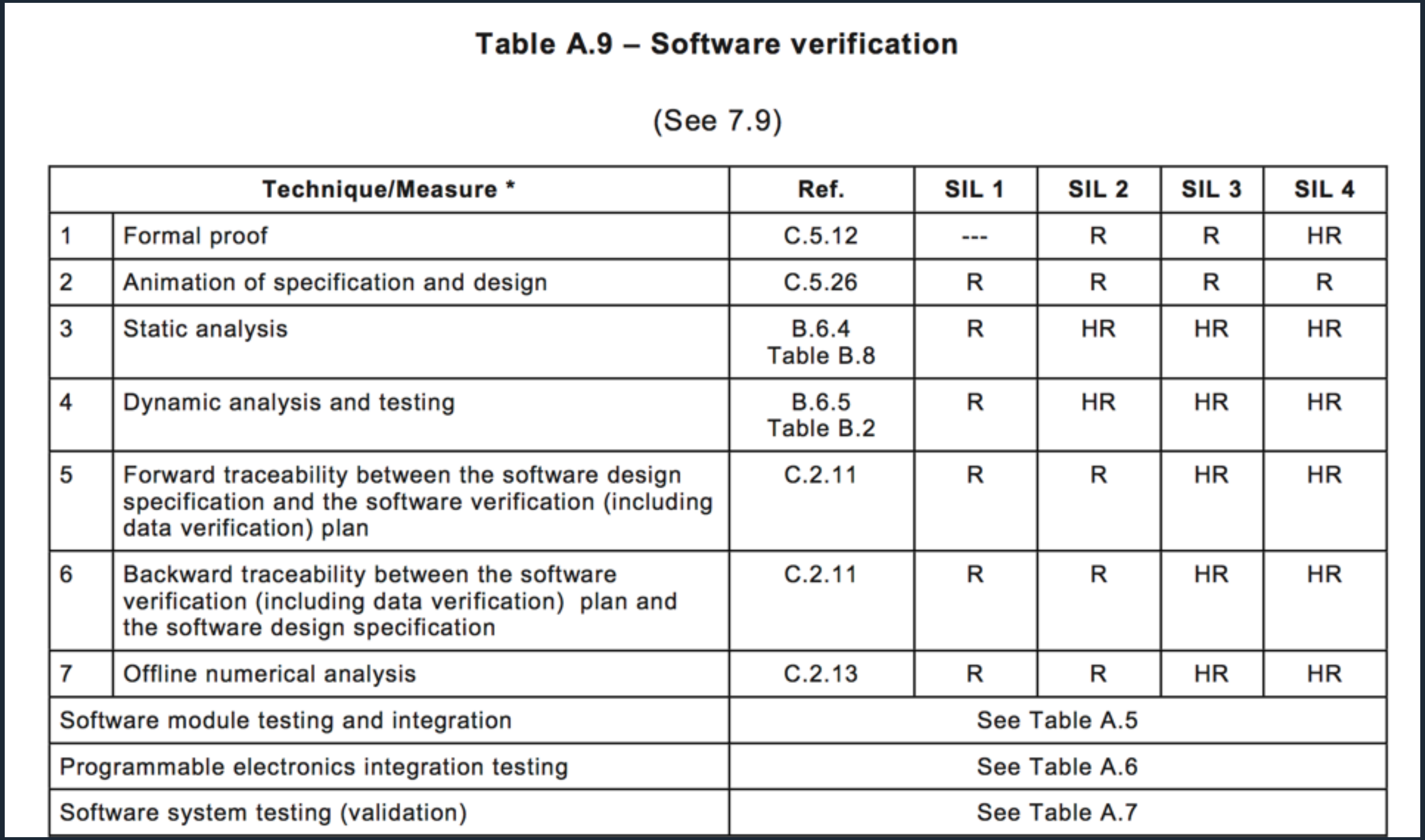


Table A.9 – Software verification

(See 7.9)

Technique/Measure *		Ref.	SIL 1	SIL 2	SIL 3	SIL 4
1	Formal proof	C.5.12	---	R	R	HR
2	Animation of specification and design	C.5.26	R	R	R	R
3	Static analysis	B.6.4 Table B.8	R	HR	HR	HR
4	Dynamic analysis and testing	B.6.5 Table B.2	R	HR	HR	HR
5	Forward traceability between the software design specification and the software verification (including data verification) plan	C.2.11	R	R	HR	HR
6	Backward traceability between the software verification (including data verification) plan and the software design specification	C.2.11	R	R	HR	HR
7	Offline numerical analysis	C.2.13	R	R	HR	HR
Software module testing and integration		See Table A.5				
Programmable electronics integration testing		See Table A.6				
Software system testing (validation)		See Table A.7				

IEC 61508 - The “Golden” Boy Safety Standard

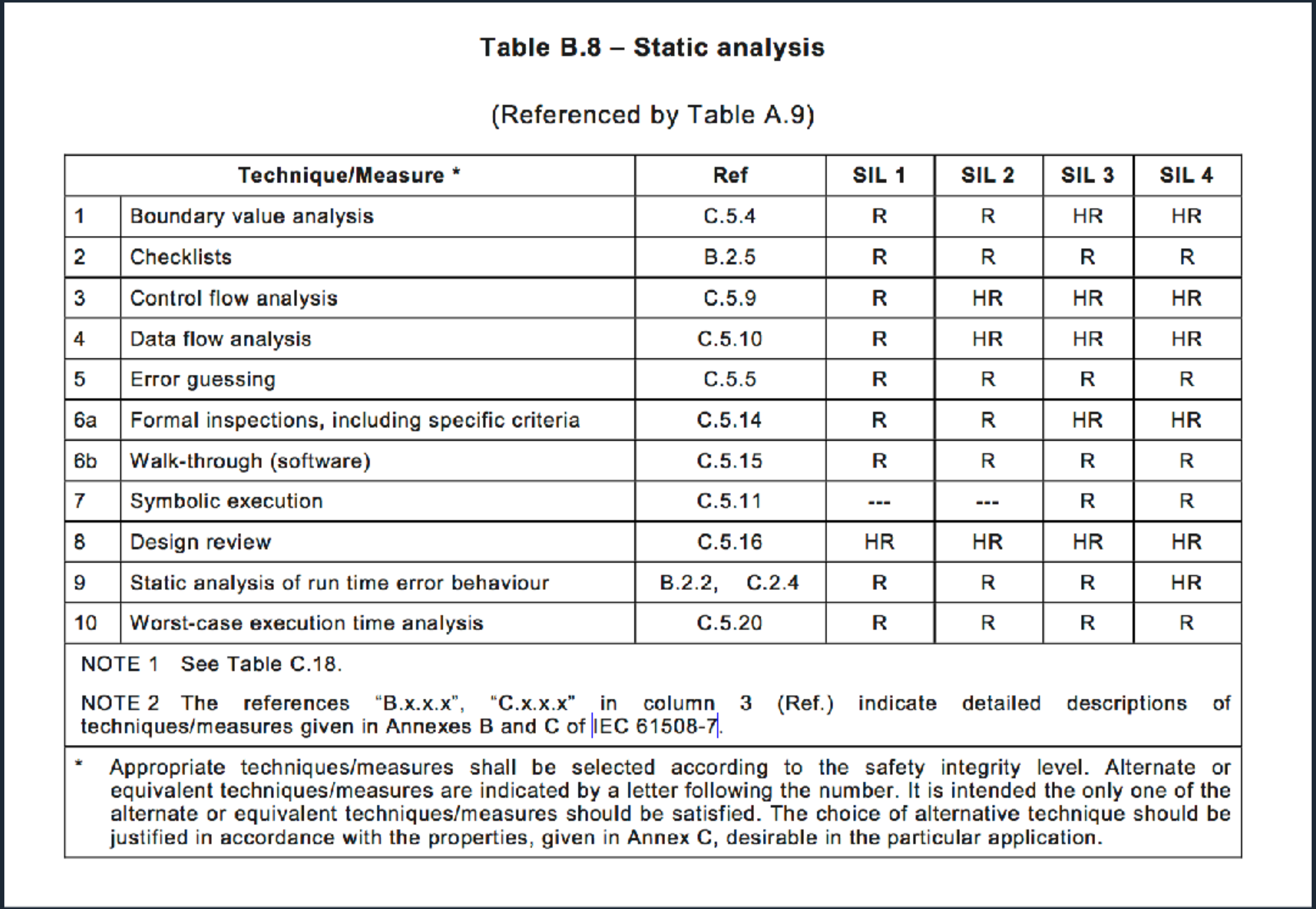


Table B.8 – Static analysis

(Referenced by Table A.9)

Technique/Measure *		Ref	SIL 1	SIL 2	SIL 3	SIL 4
1	Boundary value analysis	C.5.4	R	R	HR	HR
2	Checklists	B.2.5	R	R	R	R
3	Control flow analysis	C.5.9	R	HR	HR	HR
4	Data flow analysis	C.5.10	R	HR	HR	HR
5	Error guessing	C.5.5	R	R	R	R
6a	Formal inspections, including specific criteria	C.5.14	R	R	HR	HR
6b	Walk-through (software)	C.5.15	R	R	R	R
7	Symbolic execution	C.5.11	---	---	R	R
8	Design review	C.5.16	HR	HR	HR	HR
9	Static analysis of run time error behaviour	B.2.2, C.2.4	R	R	R	HR
10	Worst-case execution time analysis	C.5.20	R	R	R	R

NOTE 1 See Table C.18.

NOTE 2 The references “B.x.x.x”, “C.x.x.x” in column 3 (Ref.) indicate detailed descriptions of techniques/measures given in Annexes B and C of IEC 61508-7.

* Appropriate techniques/measures shall be selected according to the safety integrity level. Alternate or equivalent techniques/measures are indicated by a letter following the number. It is intended the only one of the alternate or equivalent techniques/measures should be satisfied. The choice of alternative technique should be justified in accordance with the properties, given in Annex C, desirable in the particular application.

IEC 61508 - The “Golden” Boy Safety Standard

Model Checking. E. M. Clarke, O. Grumberg, and D. A. Peled. MIT Press, 1999, ISBN 0262032708, 9780262032704

Systems and Software Verification: Model-Checking Techniques and Tools. B. Berard, M. Bidoit, A. Finkel, F. Laroussinie, A. Petit, L. Petrucci, Ph. Schnoebelen, and P. Mckenzie, Springer, 2001, ISBN 3-540-41523-8

The Spin Model Checker: Primer and Reference Manual. G. J. Holzmann. Addison-Wesley, 2003.

Using symbolic execution for verifying safety-critical systems. A. Coen-Porisini, G. Denaro, C. Ghezzi, M. Pezzé. Proceedings of the 8th European software engineering conference, and 9th ACM SIGSOFT international symposium on Foundations of software engineering. ACM, 2001.

Safety Standards

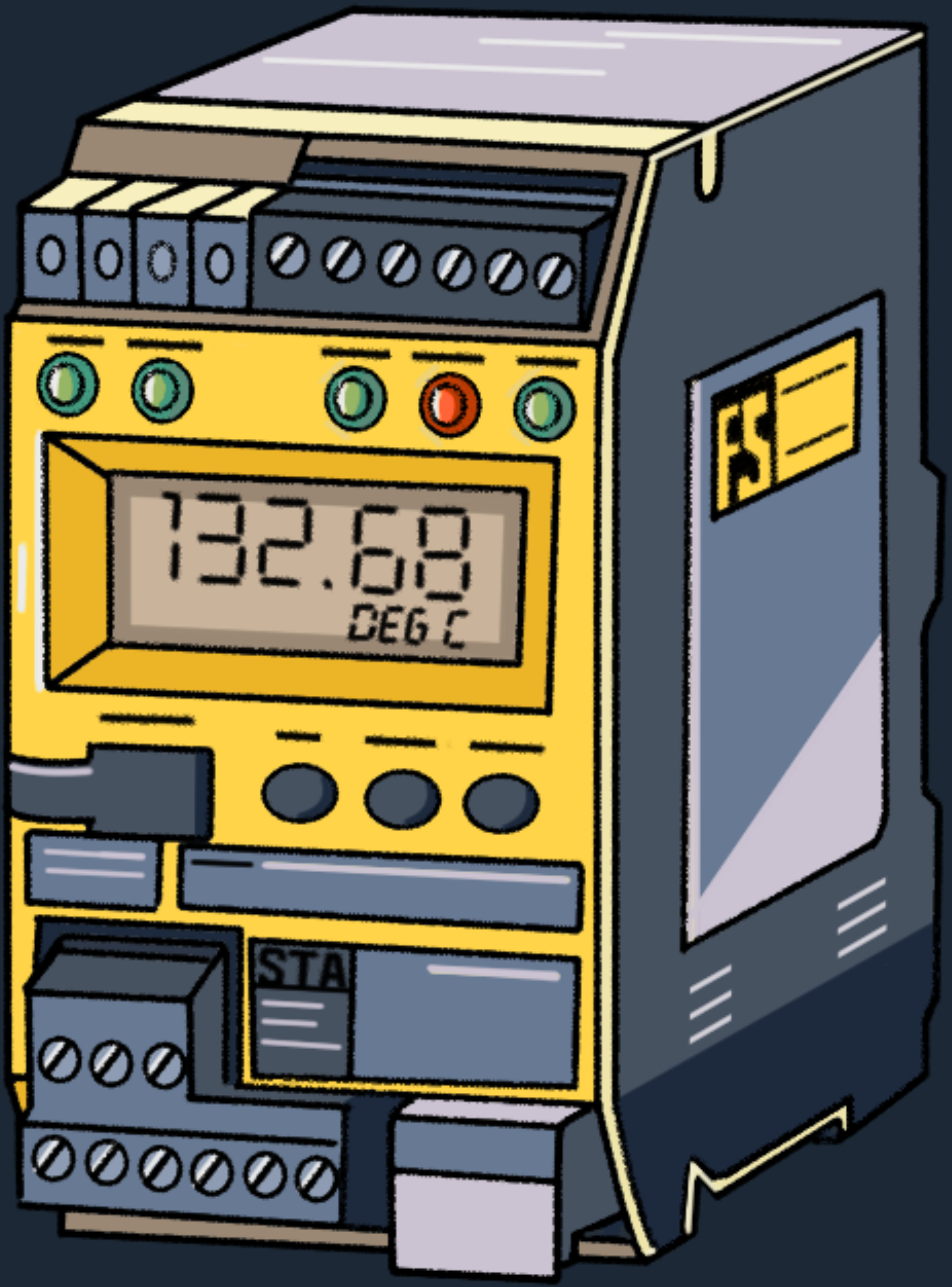
- Standards are a consensus building exercise , in particular, a consensus of a group of individuals at a specific moment in time.
- May work well for predictable systems, but defining a standardised way to build software is difficult.
- Software systems and verification technologies advance significantly faster than standard updates.
- Interpreting a standard against each unique software system may pose difficulties.
  - e.g., IEC 6108 provides no guidance on when and how to apply verification techniques. It’s a check-list approach.
  - Even if software could be standardised, software will still have bugs.

Verifying Smart Sensors

Industrial verification tools lack property-driven techniques

State-of-the-art formal verification techniques

- Assume standard C compilers (GCC/LLVM)
- Assume standard architectures (Intel x86)
- Assume underlying OS (scheduler, memory management, etc.)
- Little support for embedded devices
- No support for interrupt-driven systems
- Not scalable beyond 10k loc



Verifying Smart Sensors

Analysing Memory Resource Bounds for Low-Level Programs. Chin, WN et al., ISMM 2008.

- Show how memory resource bounds can be inferred for assembly-level programs.
- Fix-point analysis and formalised inference systems "for a small assembly language".
- Supports only subset of standard C. Example of code analyzed:

```
void f(c x, c y, d z) {
  x = new c();
  dispose(z);
  y = new c();
  dispose(x);
  dispose(y);
 }
```

- Assumes standard C, underlying OS, no interrupts, etc.

Verifying Smart Sensors

Analysing Memory Resource Bounds for Low-Level Programs. Byrolow, D. et al., ICSE 2001.

- Static checker for interrupt-driven Z86-based software with hard real-time requirements.
- Stack analysis and interrupt-latency analysis.
- Via modelling the program counter and the interrupt mask register.
  - Records whether interrupts are enabled or disabled.
- Build a control-flow graph in which each node represents the PC and the IMR.

- Not scalable, only 1K LoC.

Verifying Smart Sensors

Standards Compliance **01**

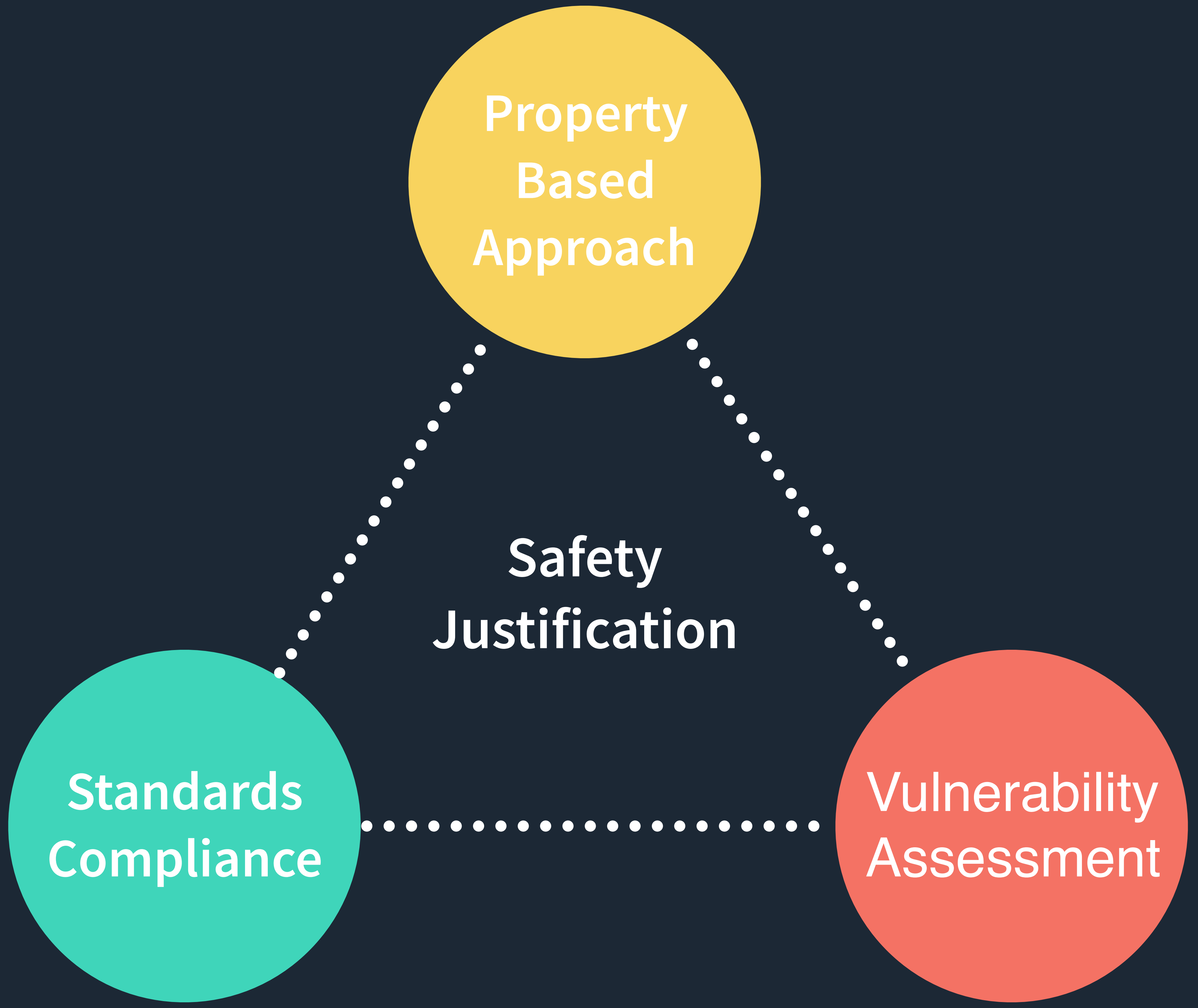
How do we manage to meet what's required by these standards, specifically when academic work can't guide us?

02 Adopting advancements

How can we adopt new techniques to help us assure systems when standards don't reflect them?

Verifying Smart Sensors

- Functionality, accuracy, timing, failure integrity, non-interference
- Data-races, deadlock freedom, stack/ buffer overflow, memory safety
- Coding standards, complexity metrics



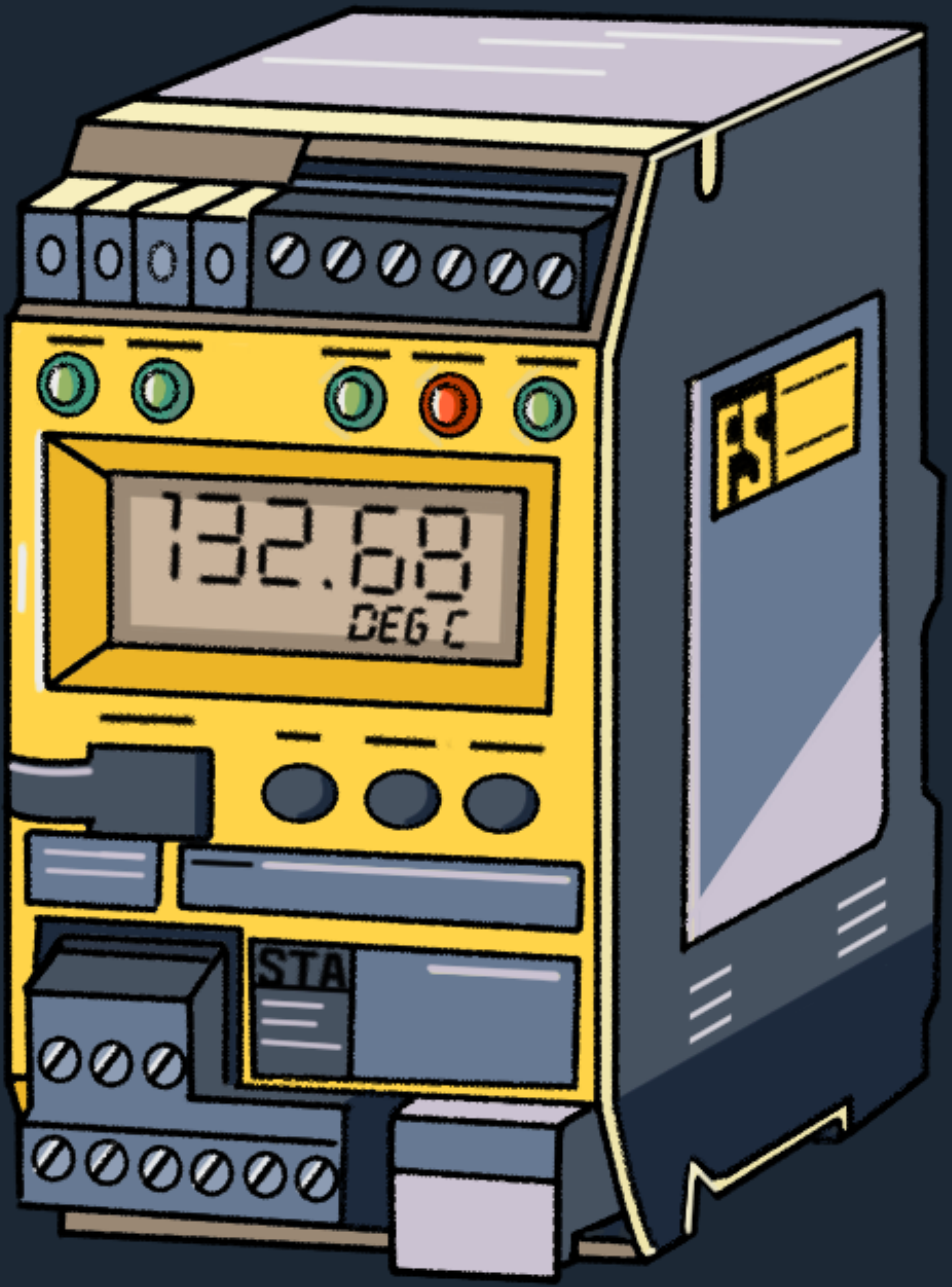
Verifying Smart Sensors

Start with available tools:

- Academia — *Frama-C* compatible only with standard C, but lucrative framework for verifying various properties
- Industry— *QA.C, Polyspace, CodeSonar, etc.*, support wider variety of compilers and architectures, but limited properties.
  - Based on Abstract Interpretation

Engineering to use tools:

- Code transformation to reduce programs to standard C
- Developing plugins for Frama-C



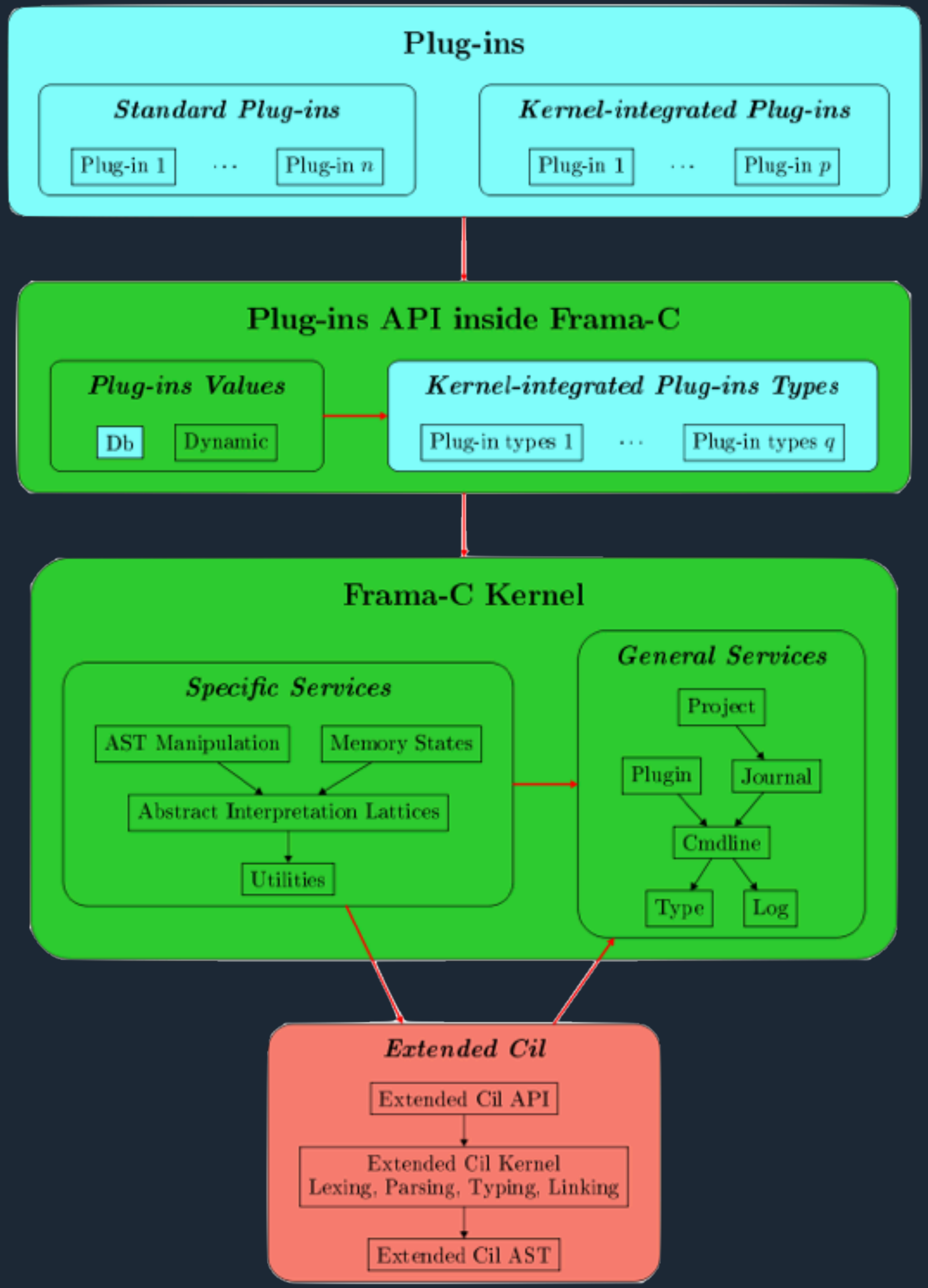
Verifying Smart Sensors

Frama-C

- Extensible through new plug-in that may use functionalities provided by existing plug-ins and kernel.
- Allows for plug-ins to be written with relatively little effort.
- Plug-ins written in OCaml, and passed as an .ml file through command-line.

Rapid prototyping and deployment

- In production: Stack analysis, resource analysis, and functional analysis.

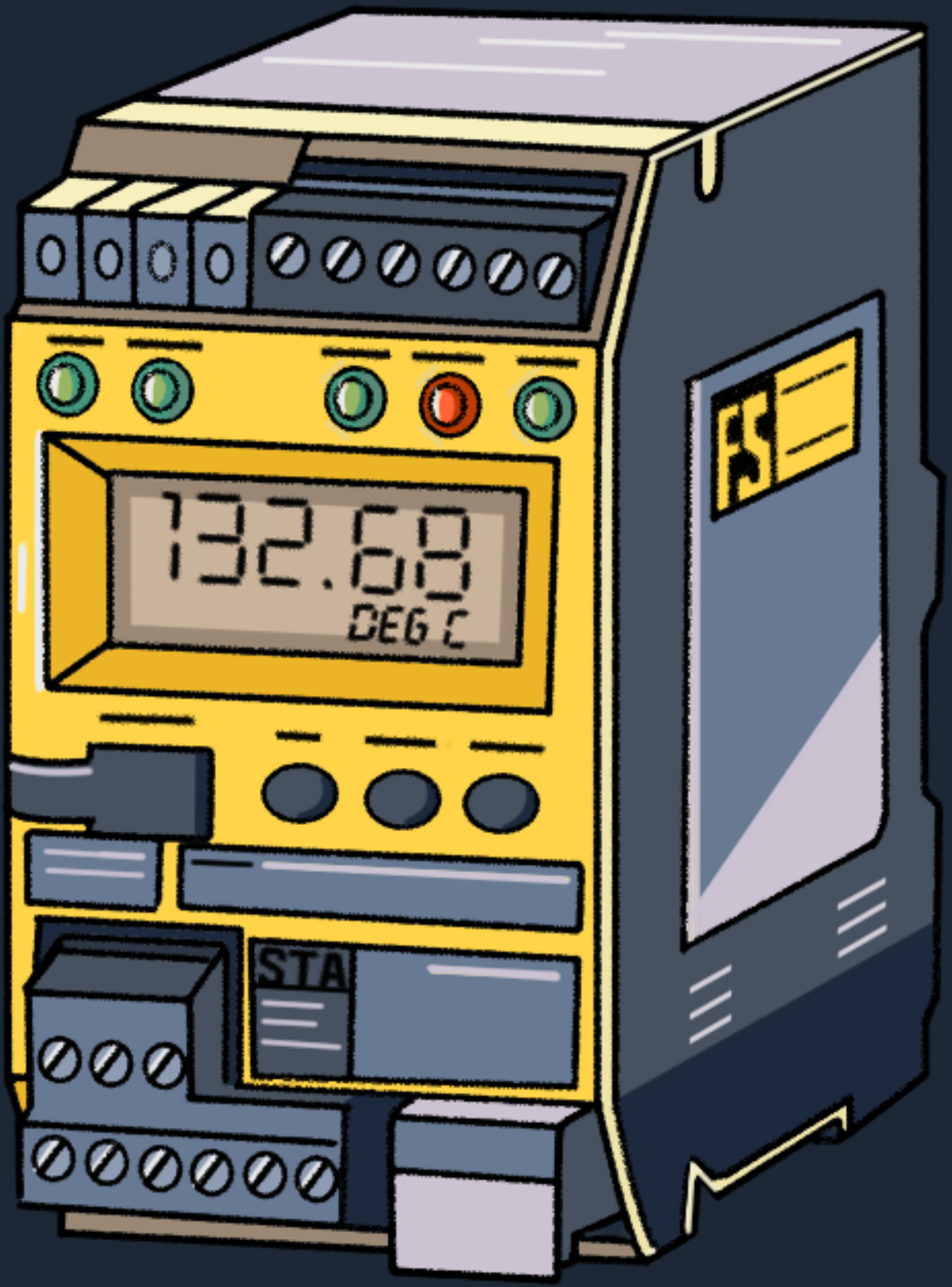


Verifying Smart Sensors

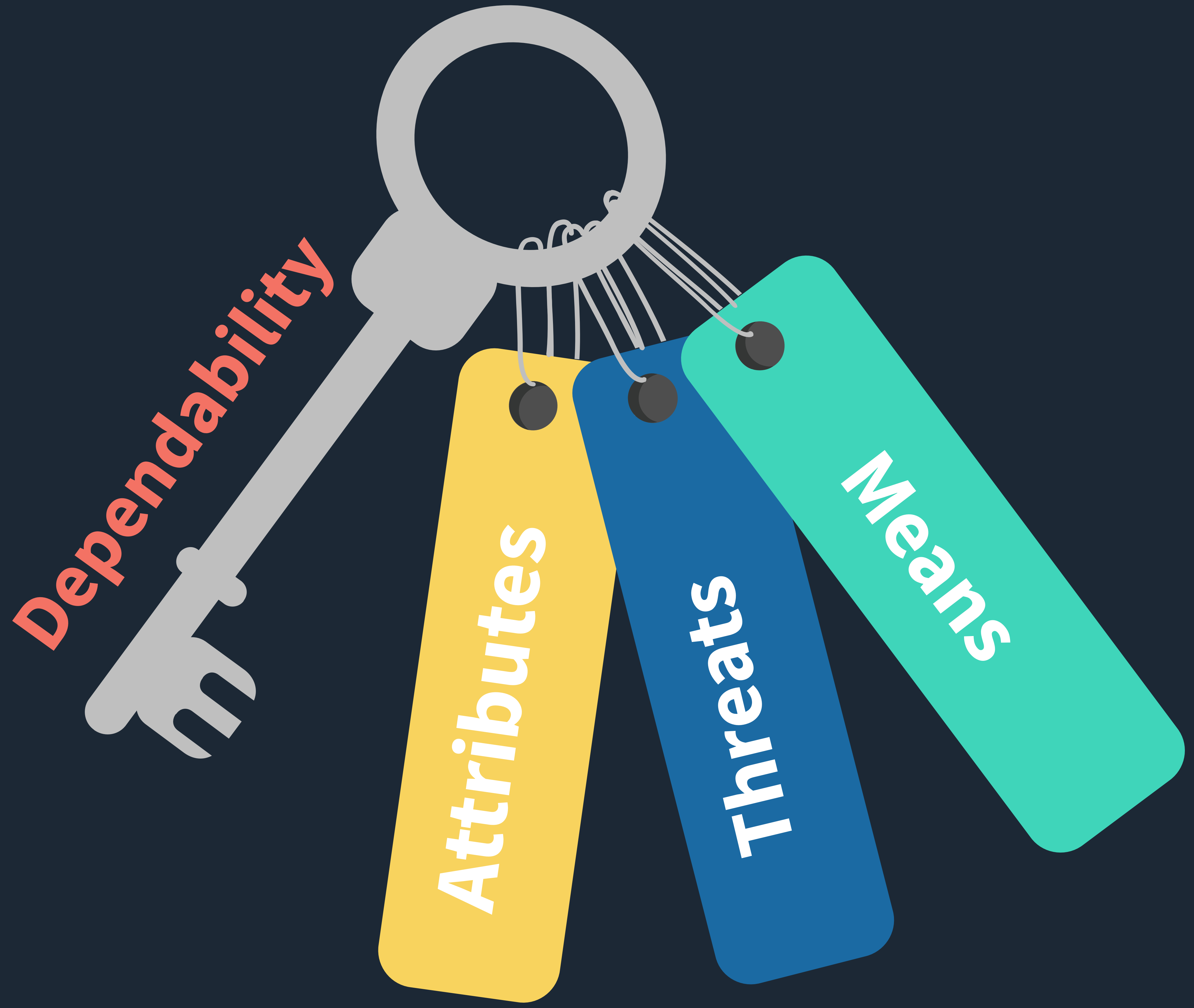
Even with automated tools, manual inspection and judgement is required

- Are the bugs false positives?
- Are they relevant to the safety and security of the system?
- Which class of bugs are more critical than others?
- What is the probability of the actual bug occurring?
- What is an acceptable level of failure?
- Can testing and field data demonstrate infeasibility of a bug?
- What efforts are required to repair a system?
- Do the repairs themselves pose a safety and security threat? What is the impact?

Hardware verification, specification correctness, standards compliance



Why Should You Care?



- Availability - readiness for correct service
- Reliability - continuity of correct service
- Safety - absence of catastrophic consequences on the user(s) and the environment
- Integrity - absence of improper system alteration
- Maintainability - ability for a process to undergo modifications and repairs

Opening a Dialogue

01 Consider these safety critical systems in the scope of future technologies and frameworks developed. If the appropriate technology is there, it will be adopted.

02 Recognise that what safety entails is always evolving, and will in fact affect our views on system dependability and security. Verification is just a small part of assurance.

03 Standards and guidelines already exist to increase systems' security and safety. Applying best practices wherever applicable will promote system dependability.

Thank You!

@heidykhlaaf