C++ Development in the Safety Critical Domain

An introduction & lessons learned from a large automotive product

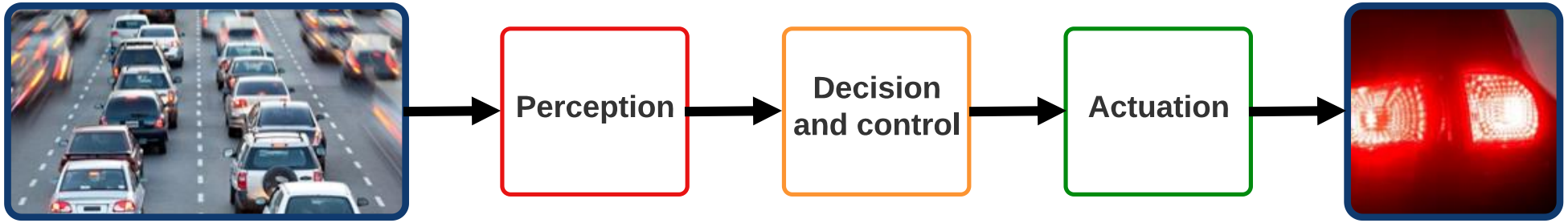
GbgCpp
Gothenburg, Sweden / 2023.04.05

David Friberg,
Technical Expert, Zenseact

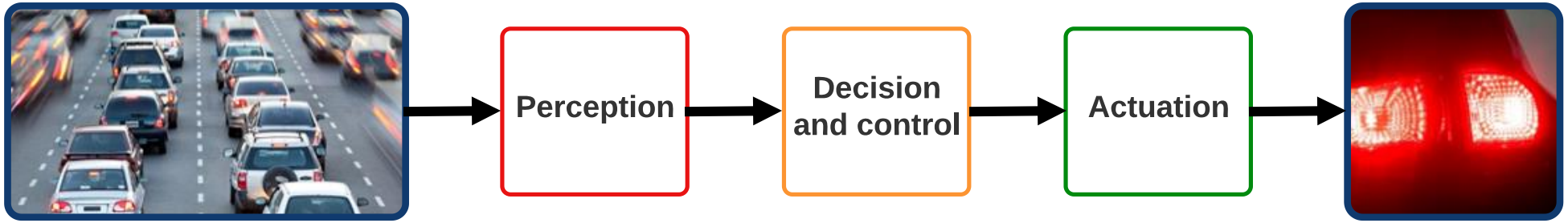
Agenda

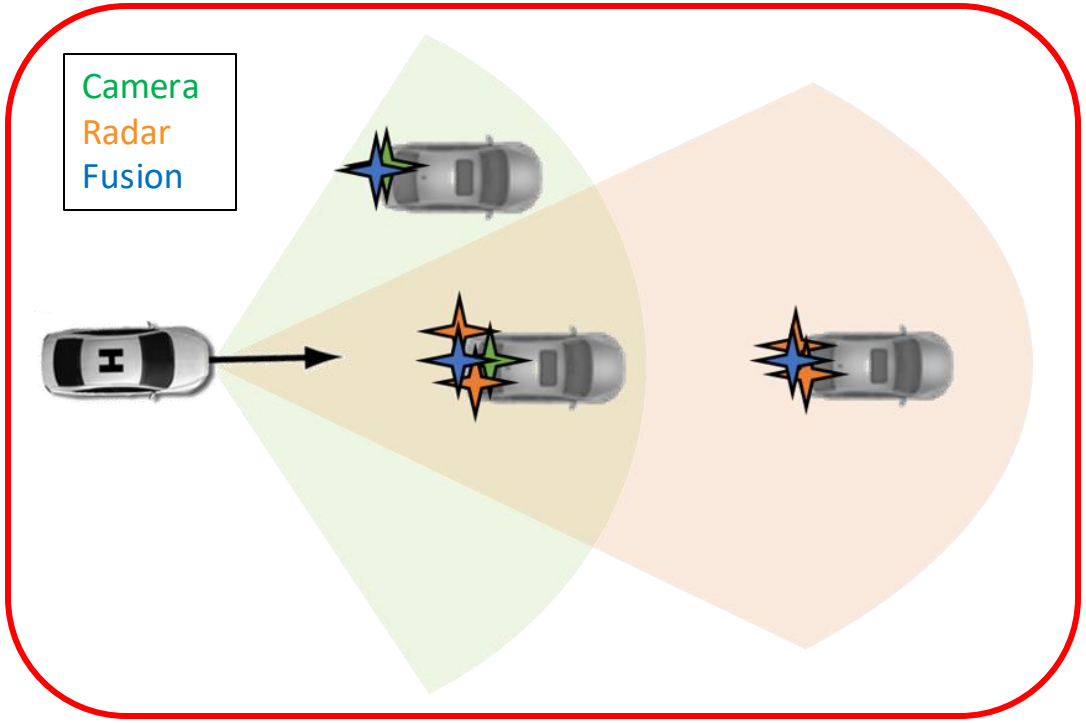
- ✓ A brief introduction *functional safety* and a safety-critical product
- ✓ Architectural levels, activities and associated effort
- ✓ *Language safety*: safety critical C++ & language subsets
- ✓ Memory safety
- ✓ Core libraries
- ✓ Unit testing and the search/struggle for test sufficiency
- ✓ A few words on requirements
- ✓ Human factors and safety culture
- ✓ Safety vs software engineering: perceived contradictions

A brief introduction to a (simplified) safety-critical product

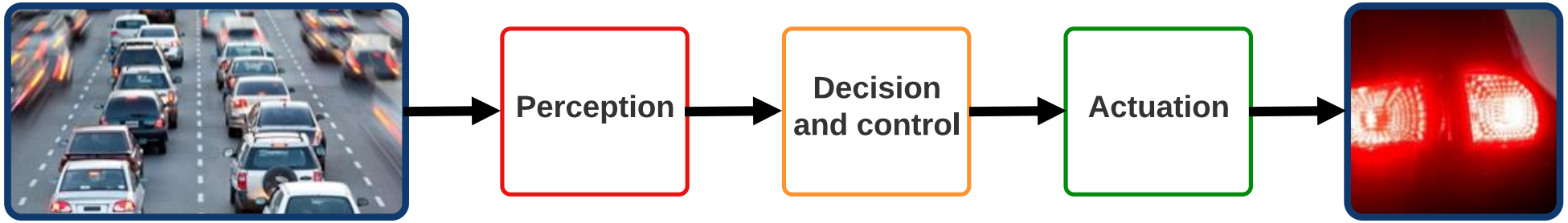


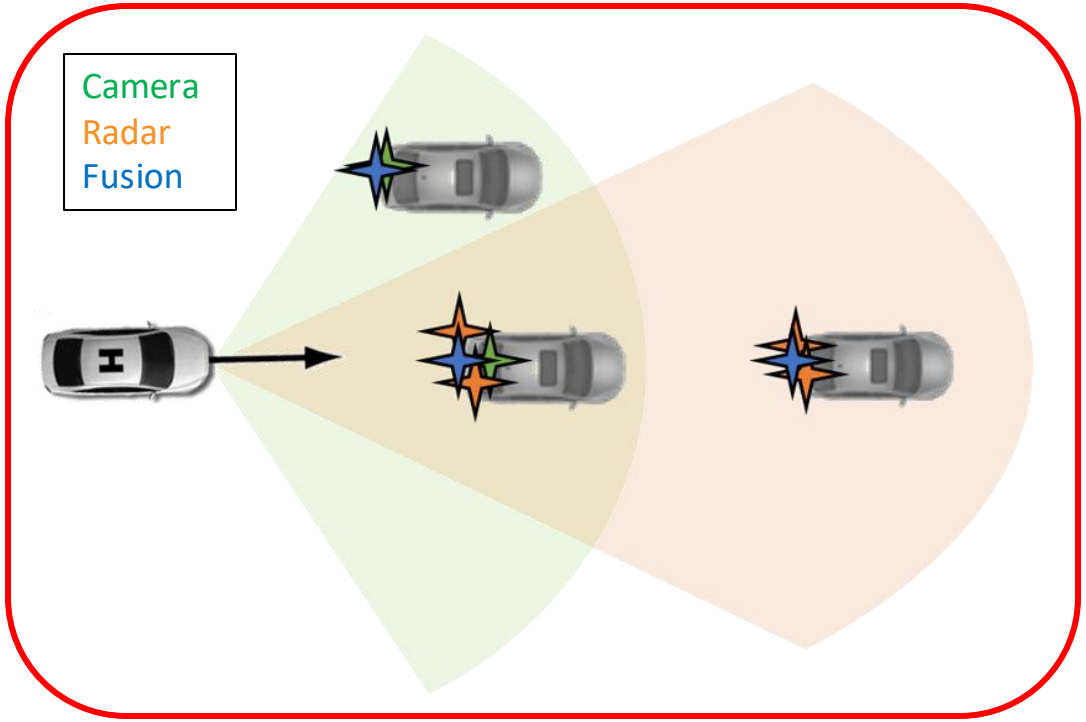
A brief introduction to a (simplified) safety-critical product

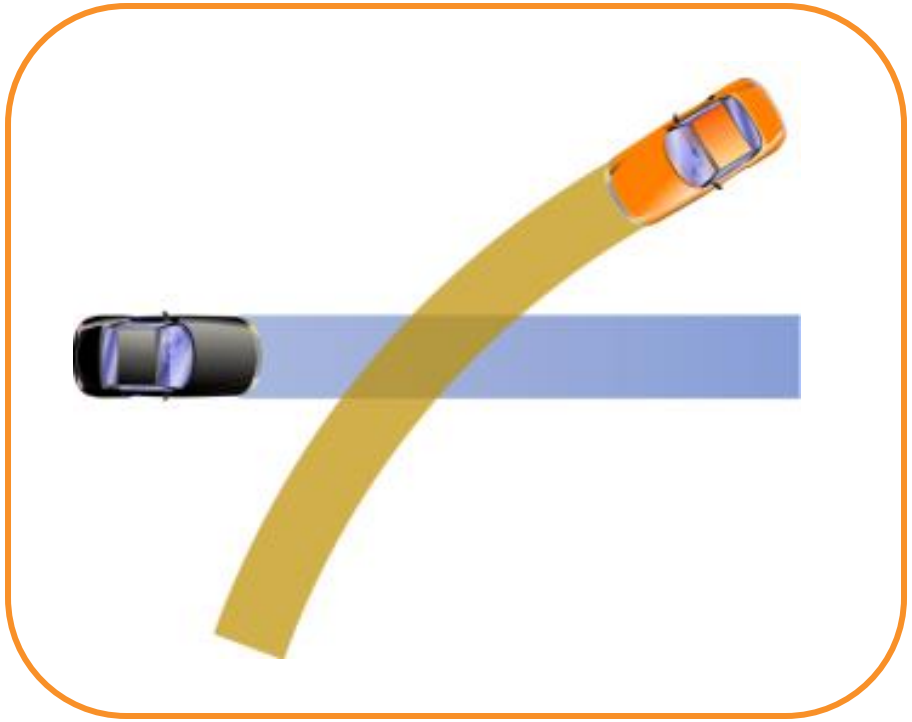




A brief introduction to a (simplified) safety-critical product







Functional safety: end-to-end (system & environment)

Perception

Decision and control

Actuation

Feature at the vehicle level

Hazard Analysis & Risk Assessment (**HARA**)

Any resulting **Safety Goals**?

No

Yes

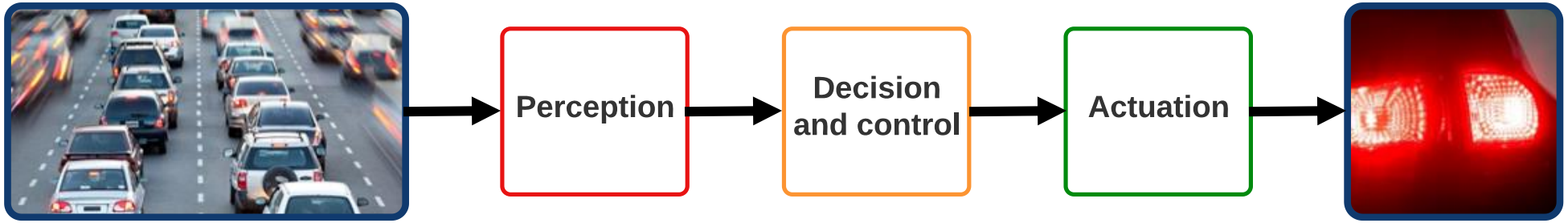
Requirements

QM (Quality Management)

ASIL (Automotive Safety Integrity Level)

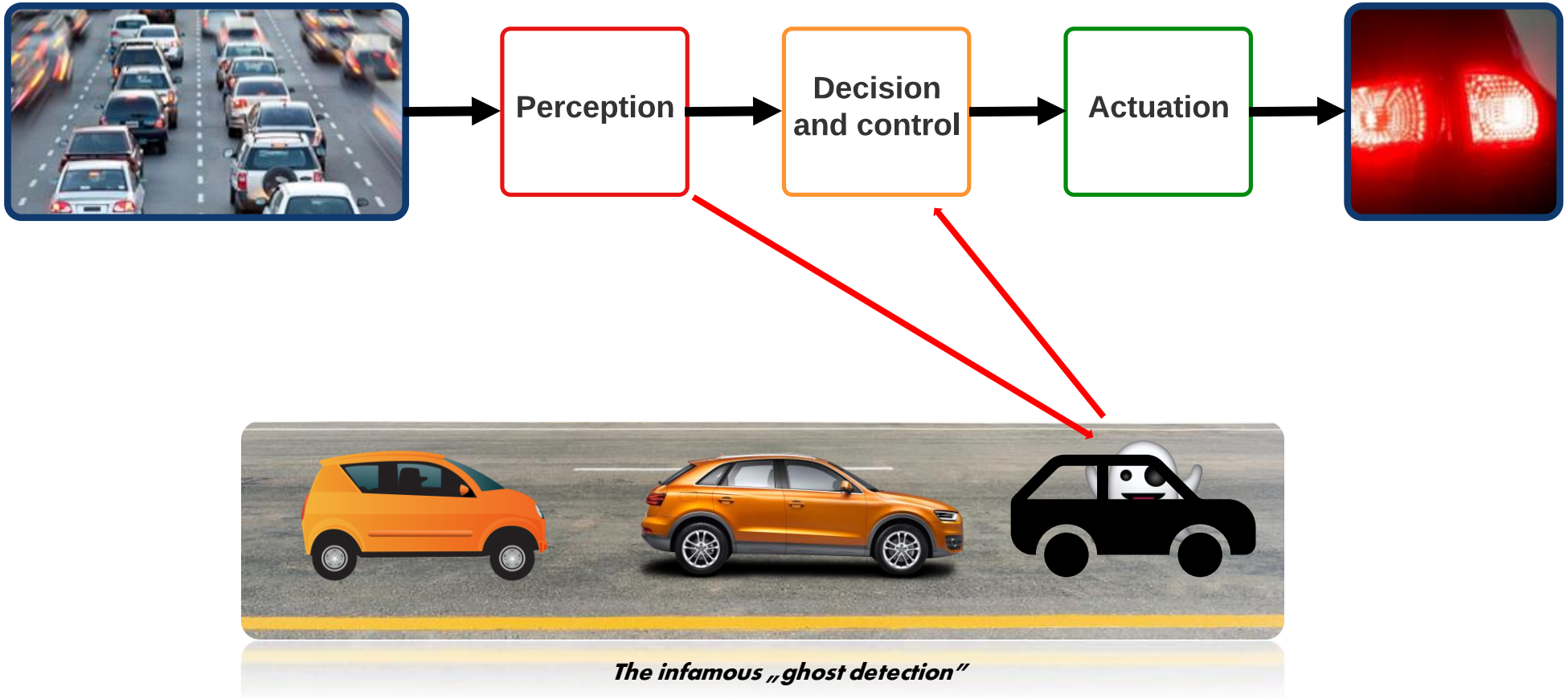
Safety **Requirements**

Functional safety: end-to-end (system & environment)



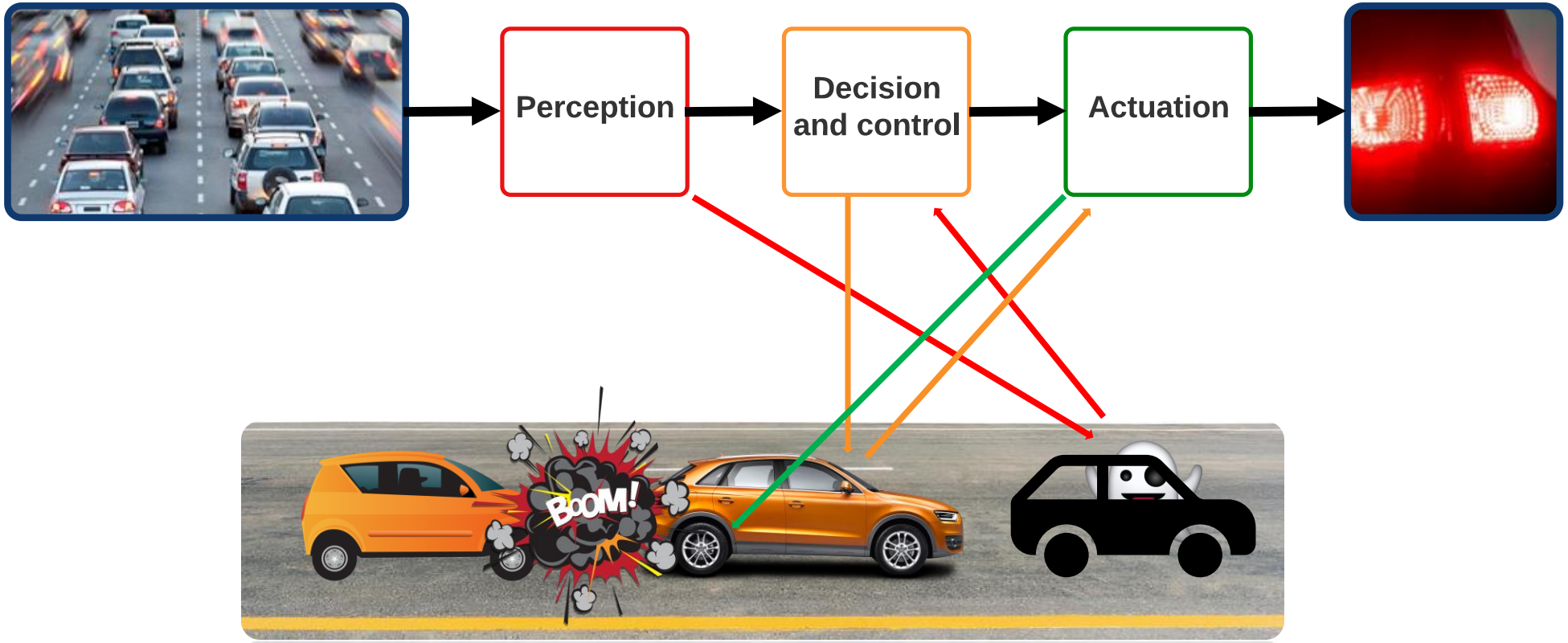
The infamous „ghost detection"

Functional safety: end-to-end (system & environment)



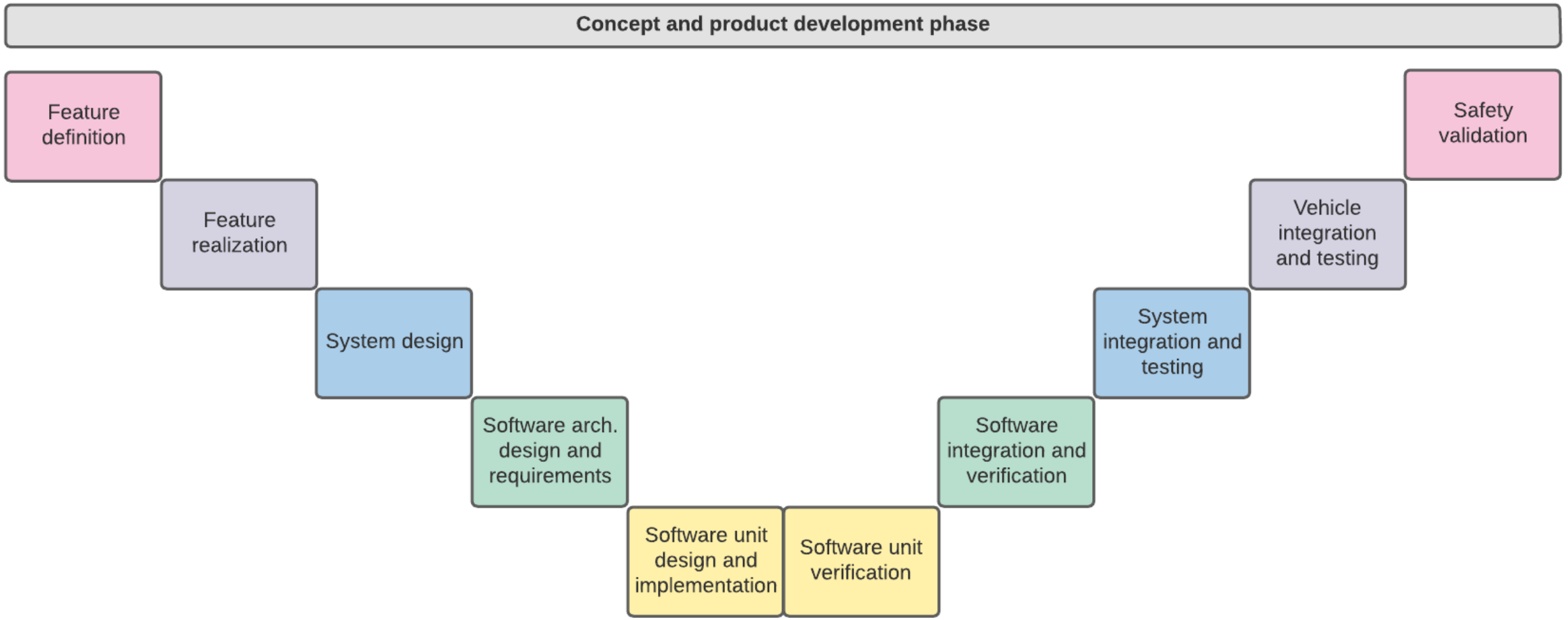
The infamous „ghost detection"

Functional safety: end-to-end (system & environment)

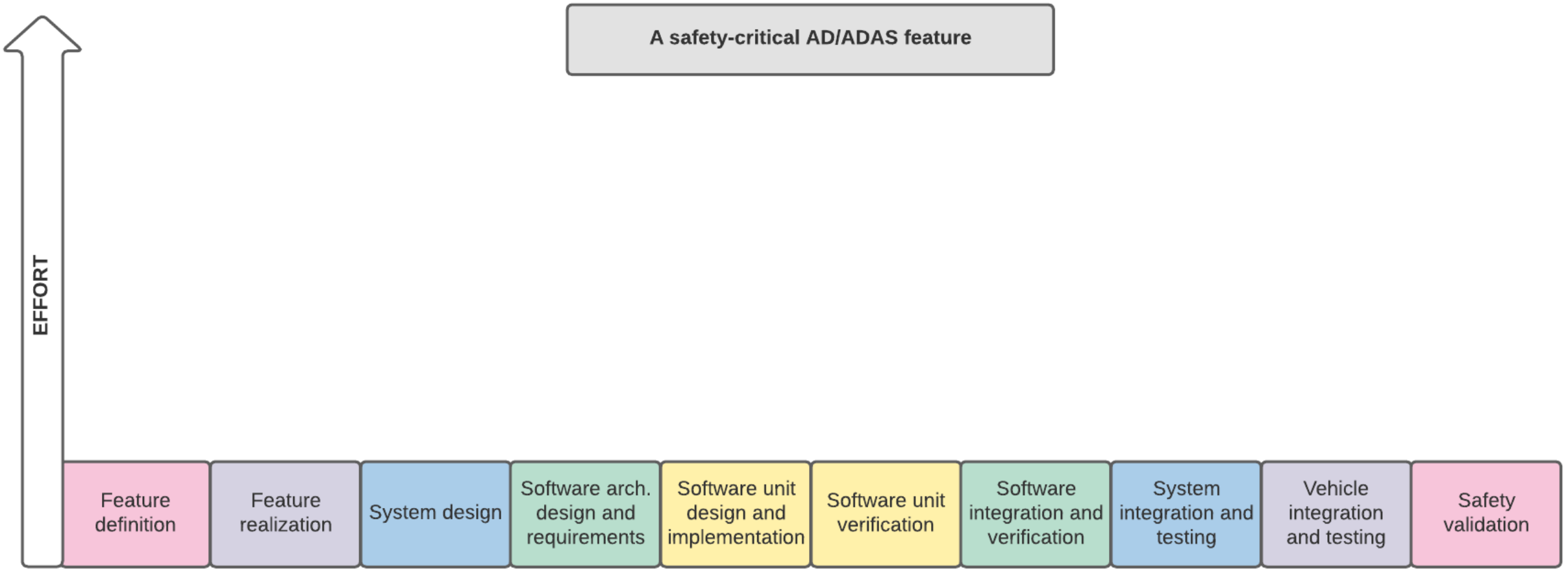


The infamous „ghost detection"

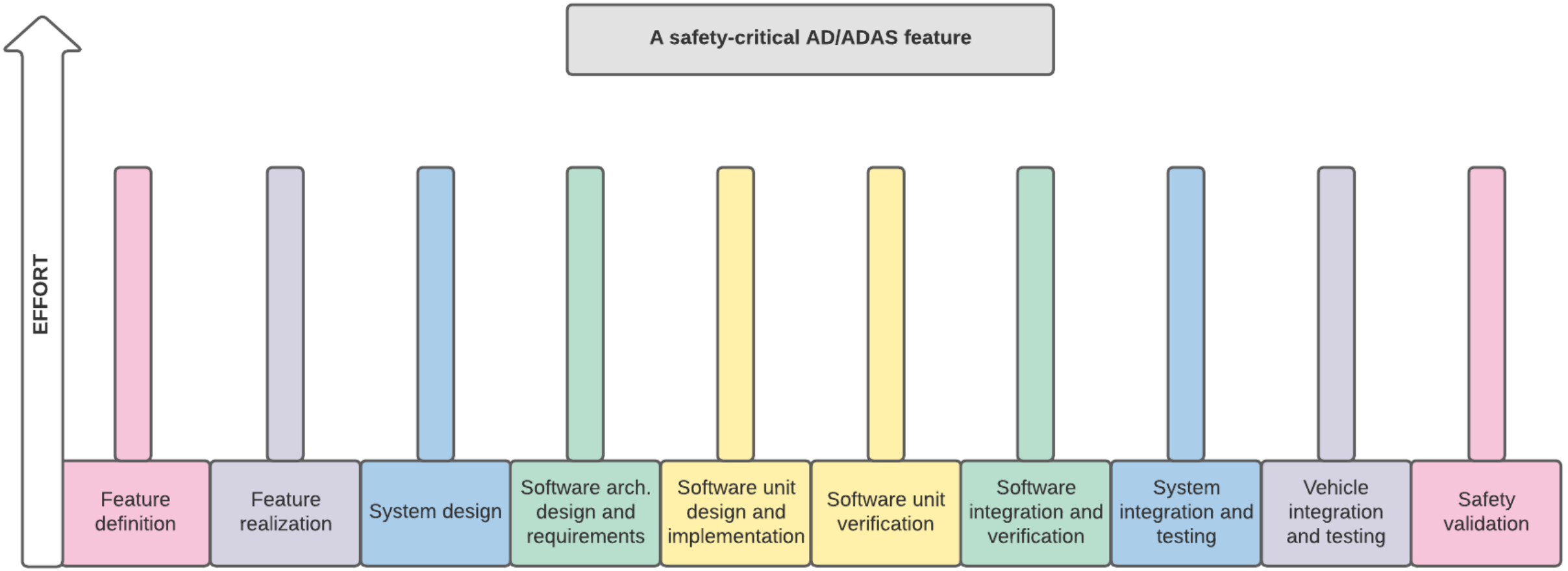
Architectural levels, activities and associated effort



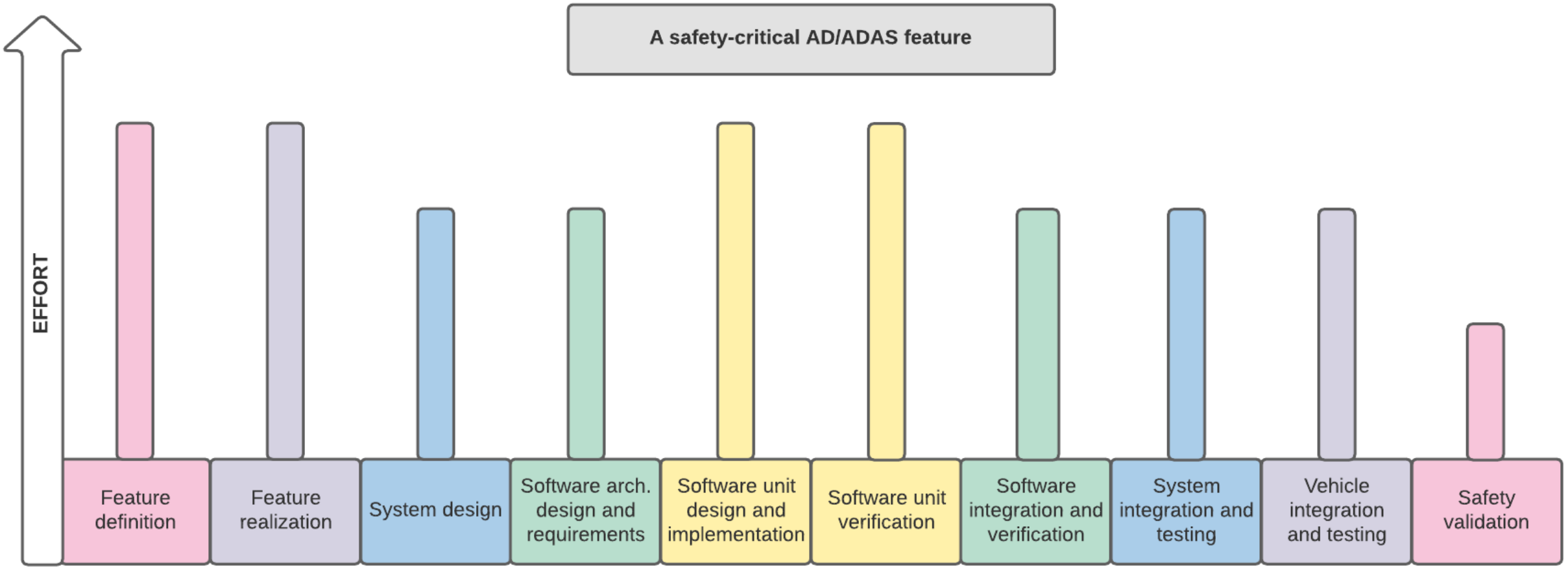
Architectural levels, activities and associated effort



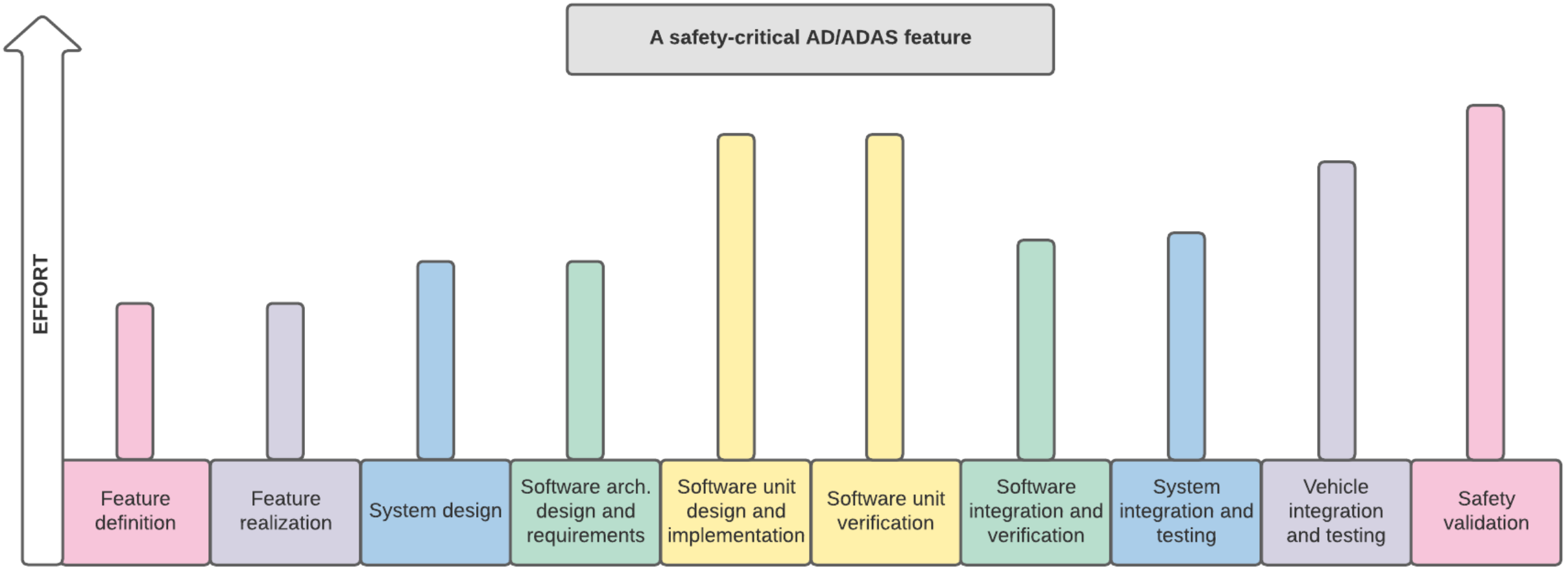
Architectural levels, activities and associated effort



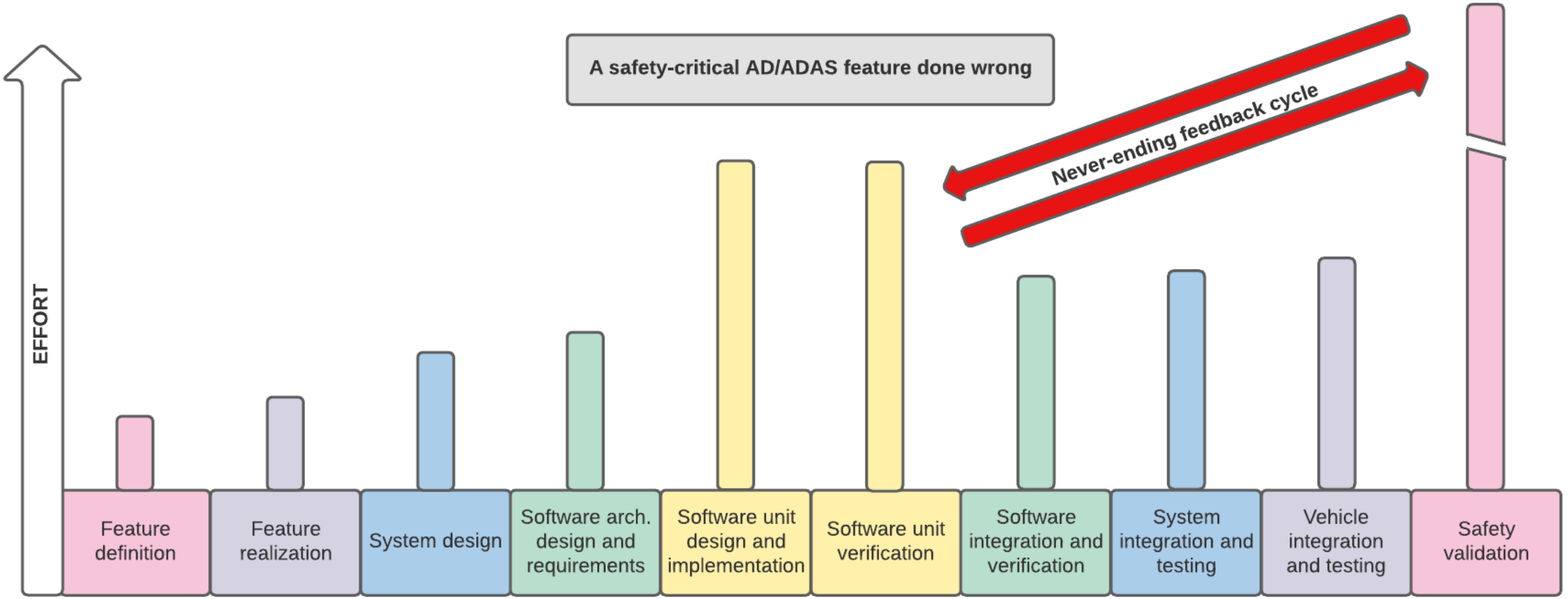
Architectural levels, activities and associated effort



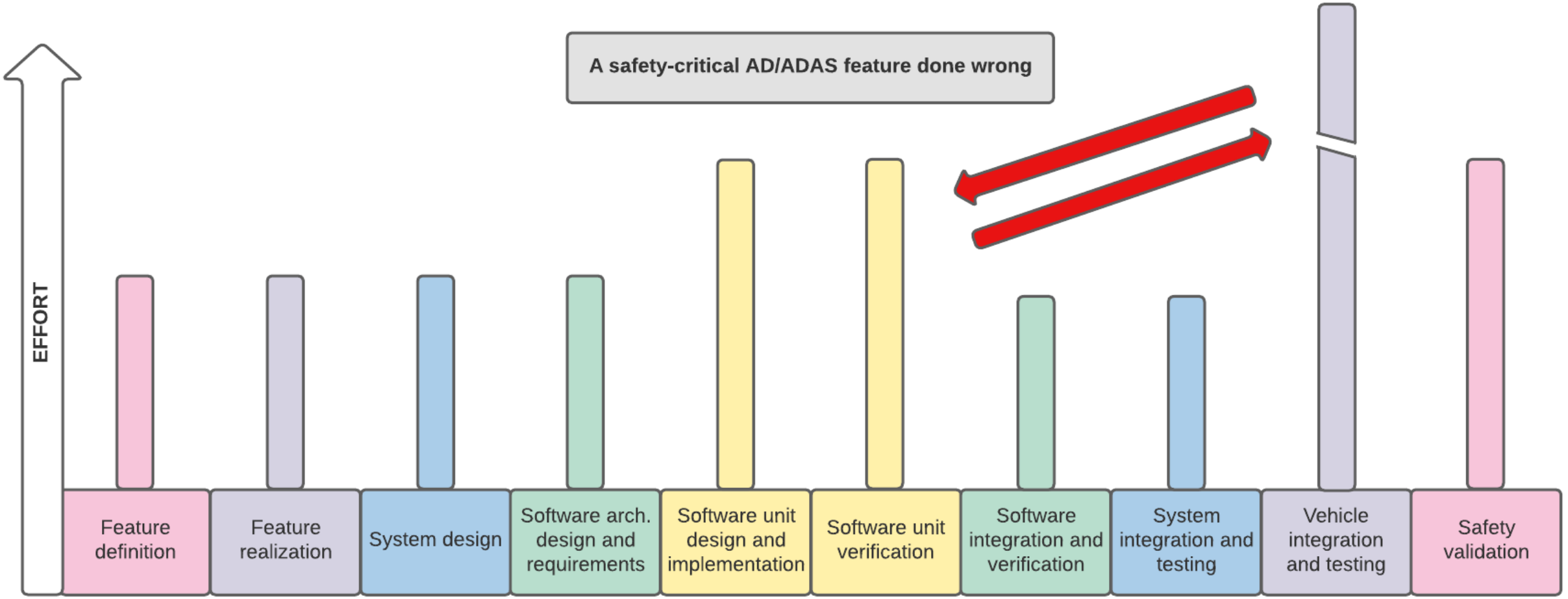
Architectural levels, activities and associated effort



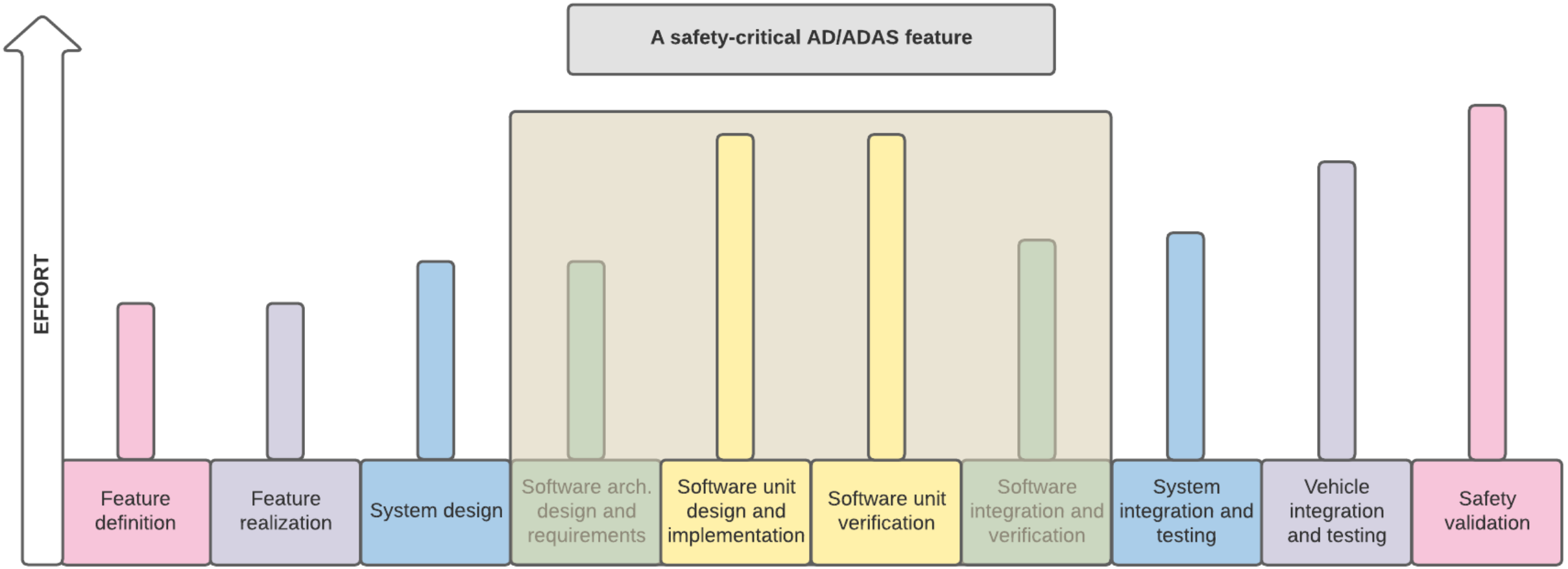
Architectural levels, activities and associated effort



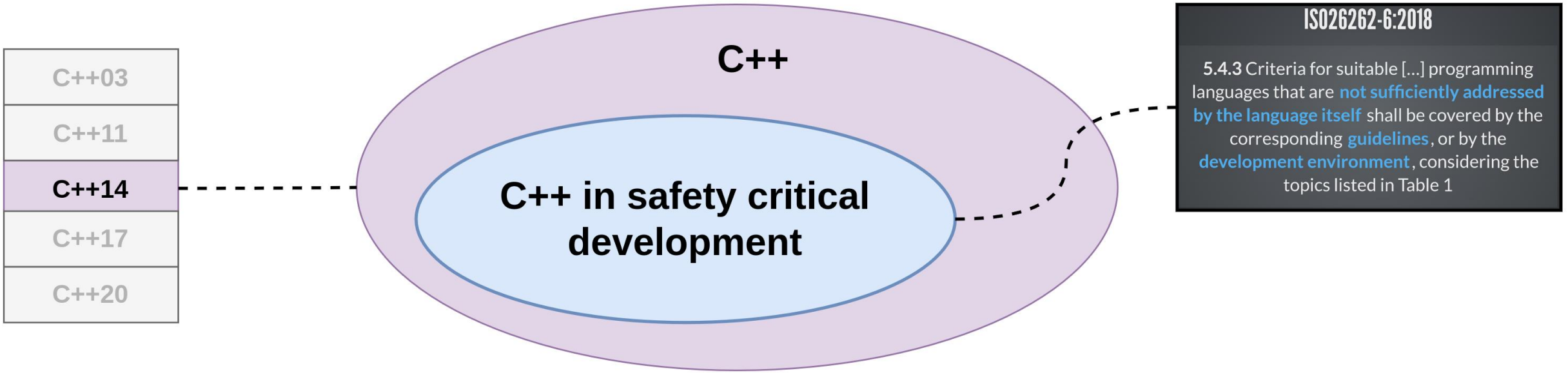
Architectural levels, activities and associated effort

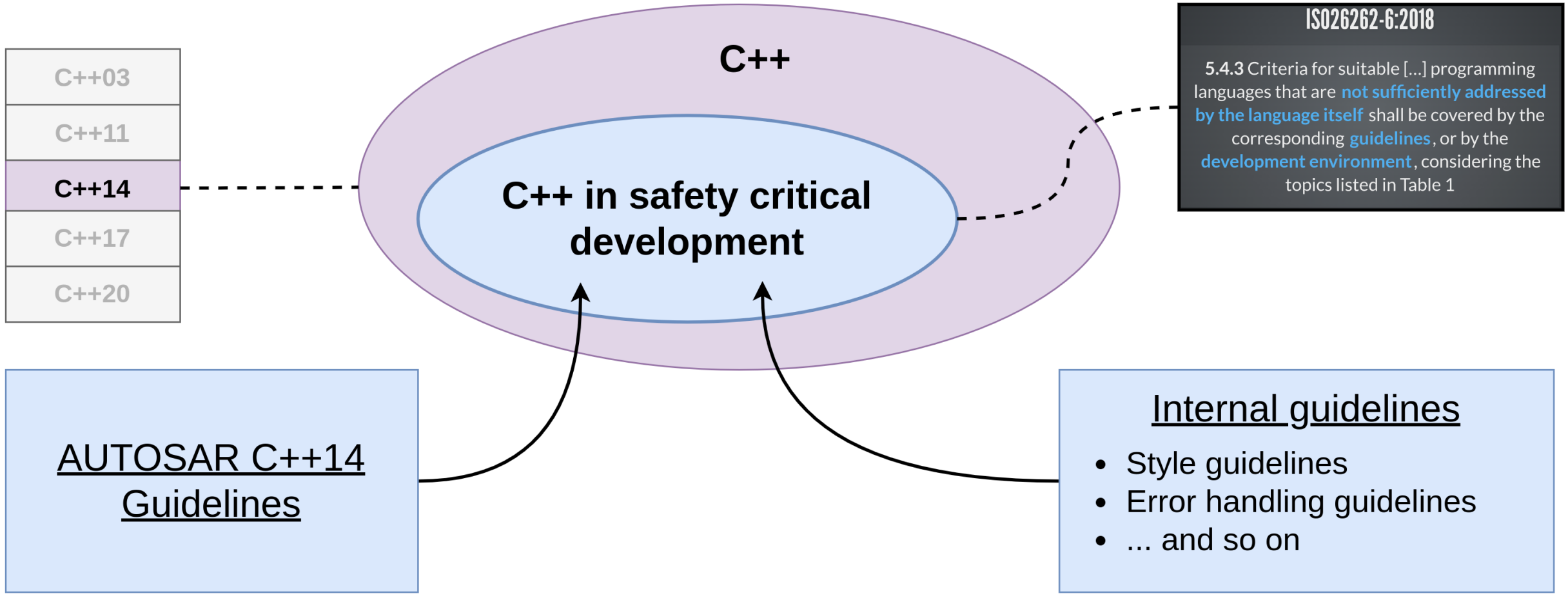


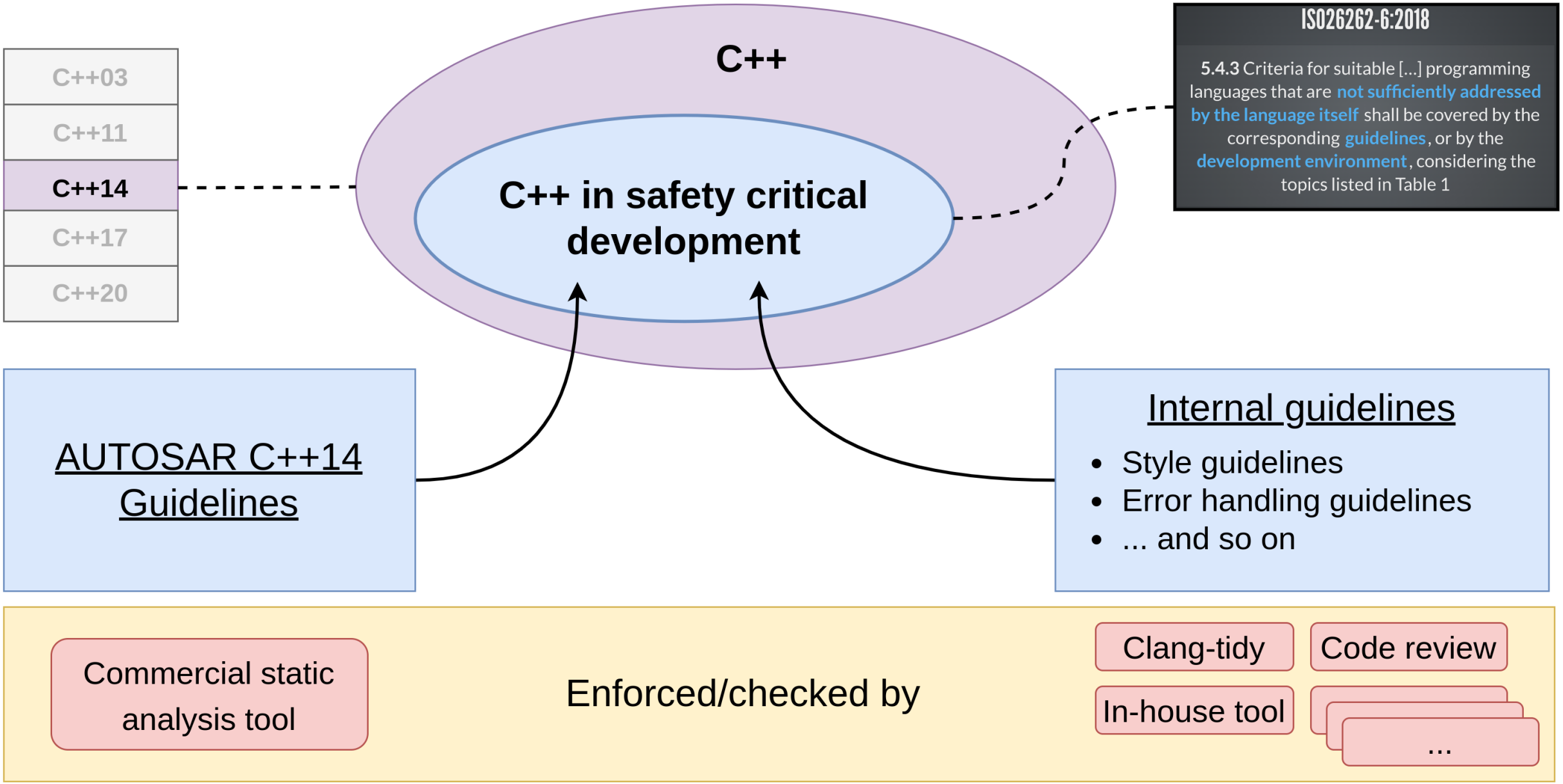
Architectural levels, activities and associated effort

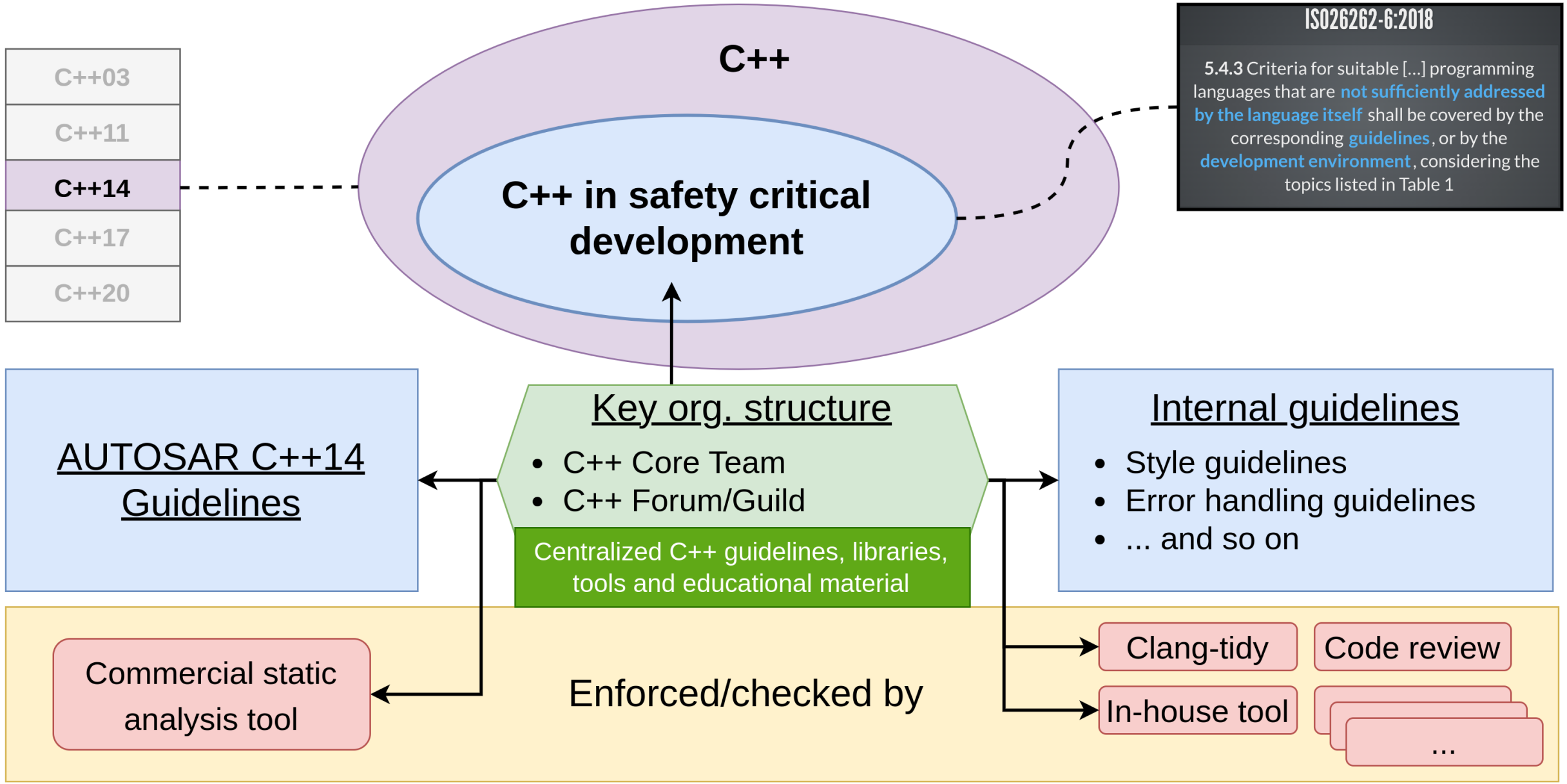


Programming language safety: C++ language subsets

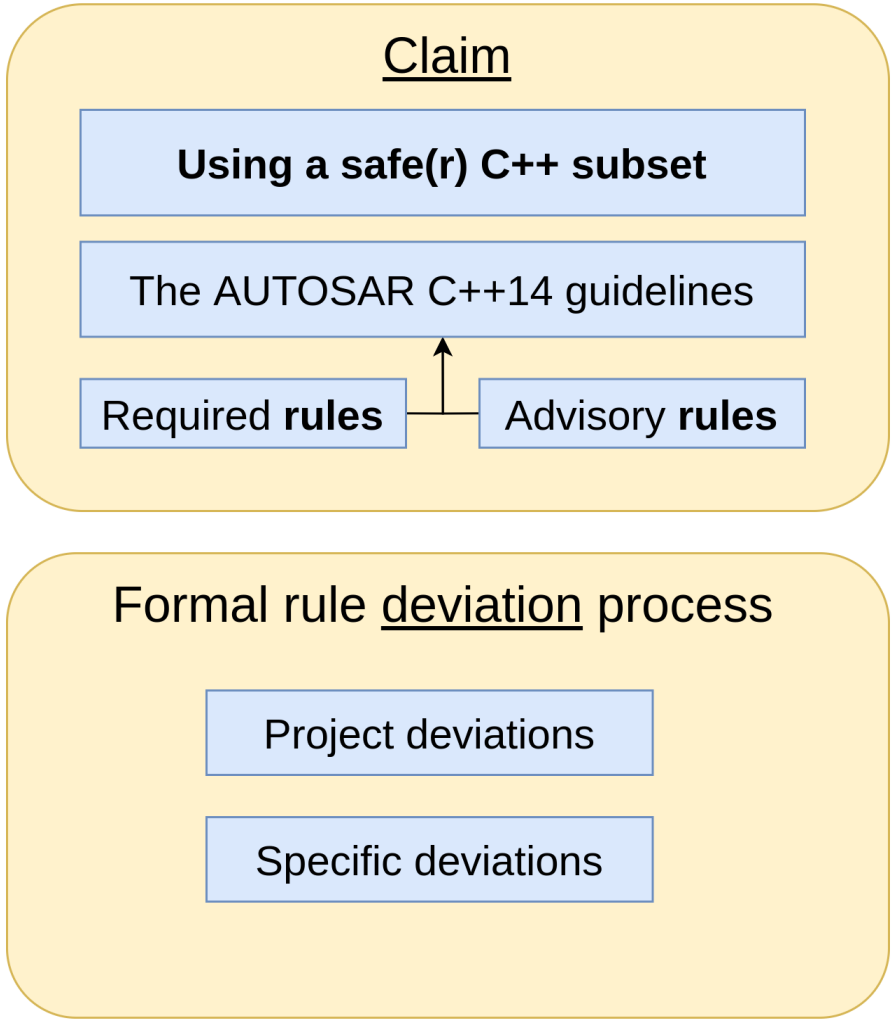

C++03
C++11
C++14
C++17
C++20
C++
C++ in safety critical development
ISO26262-6:2018
5.4.3 Criteria for suitable [...] programming languages that are not sufficiently addressed by the language itself shall be covered by the corresponding guidelines, or by the development environment, considering the topics listed in Table 1


C++03
C++11
C++14
C++17
C++20
C++
C++ in safety critical development
ISO26262-6:2018
5.4.3 Criteria for suitable [...] programming languages that are not sufficiently addressed by the language itself shall be covered by the corresponding guidelines, or by the development environment, considering the topics listed in Table 1
AUTOSAR C++14 Guidelines
Internal guidelines
• Style guidelines
• Error handling guidelines
• ... and so on

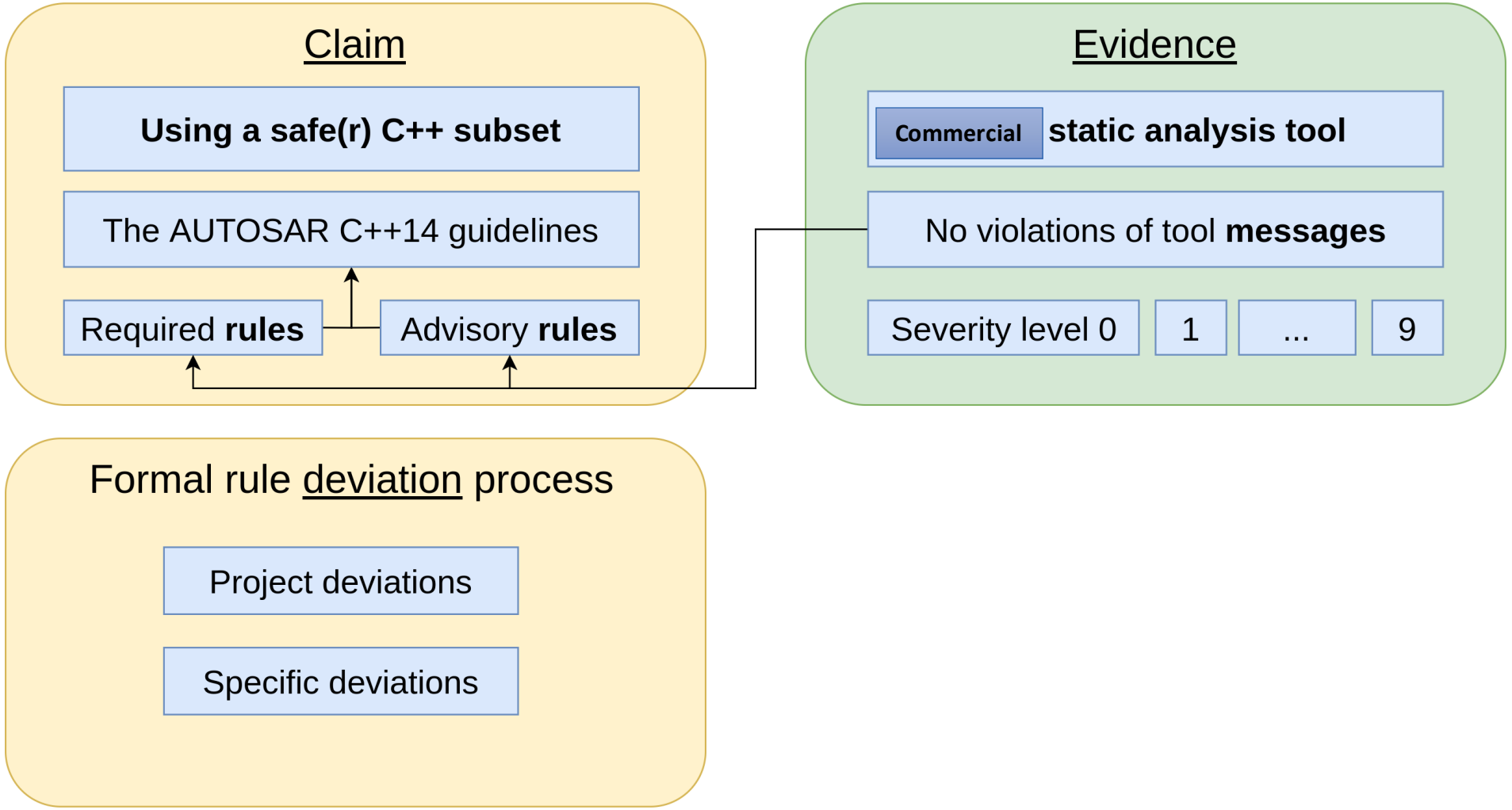

C++03
C++11
C++14
C++17
C++20
C++
C++ in safety critical development
ISO26262-6:2018
5.4.3 Criteria for suitable [...] programming languages that are not sufficiently addressed by the language itself shall be covered by the corresponding guidelines, or by the development environment, considering the topics listed in Table 1
AUTOSAR C++14 Guidelines
Internal guidelines
Style guidelines
Error handling guidelines
... and so on
Commercial static analysis tool
Enforced/checked by
Clang-tidy
Code review
In-house tool
...


C++03
C++11
C++14
C++17
C++20
C++
C++ in safety critical development
ISO26262-6:2018
5.4.3 Criteria for suitable [...] programming languages that are not sufficiently addressed by the language itself shall be covered by the corresponding guidelines, or by the development environment, considering the topics listed in Table 1
AUTOSAR C++14 Guidelines
Key org. structure
C++ Core Team
C++ Forum/Guild
Centralized C++ guidelines, libraries, tools and educational material
Internal guidelines
Style guidelines
Error handling guidelines
... and so on
Commercial static analysis tool
Enforced/checked by
Clang-tidy
Code review
In-house tool
...

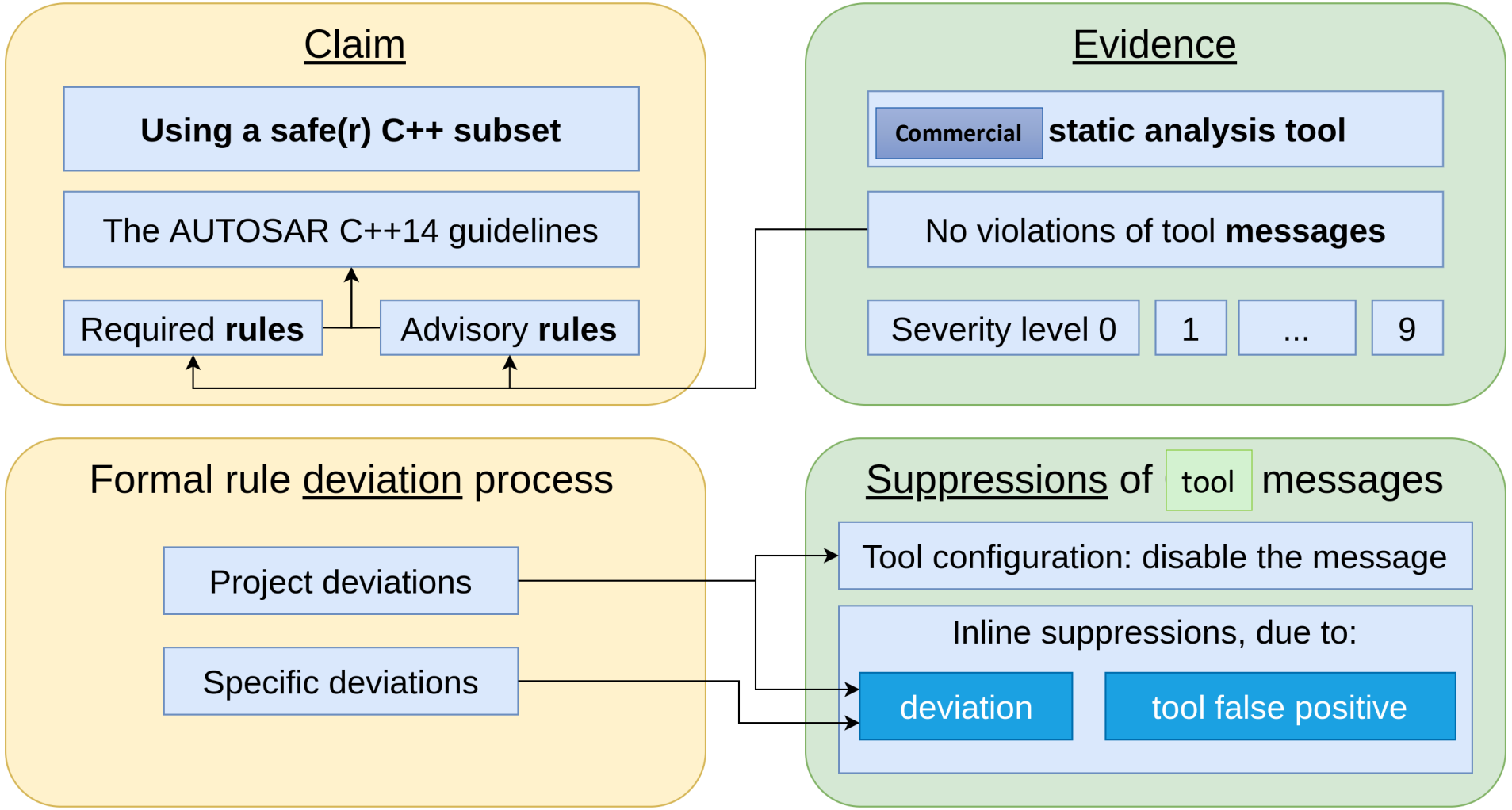
Rule compliance vs static analysis tools' diagnostics



Rule compliance vs static analysis tools' diagnostics



Rule compliance vs static analysis tools' diagnostics



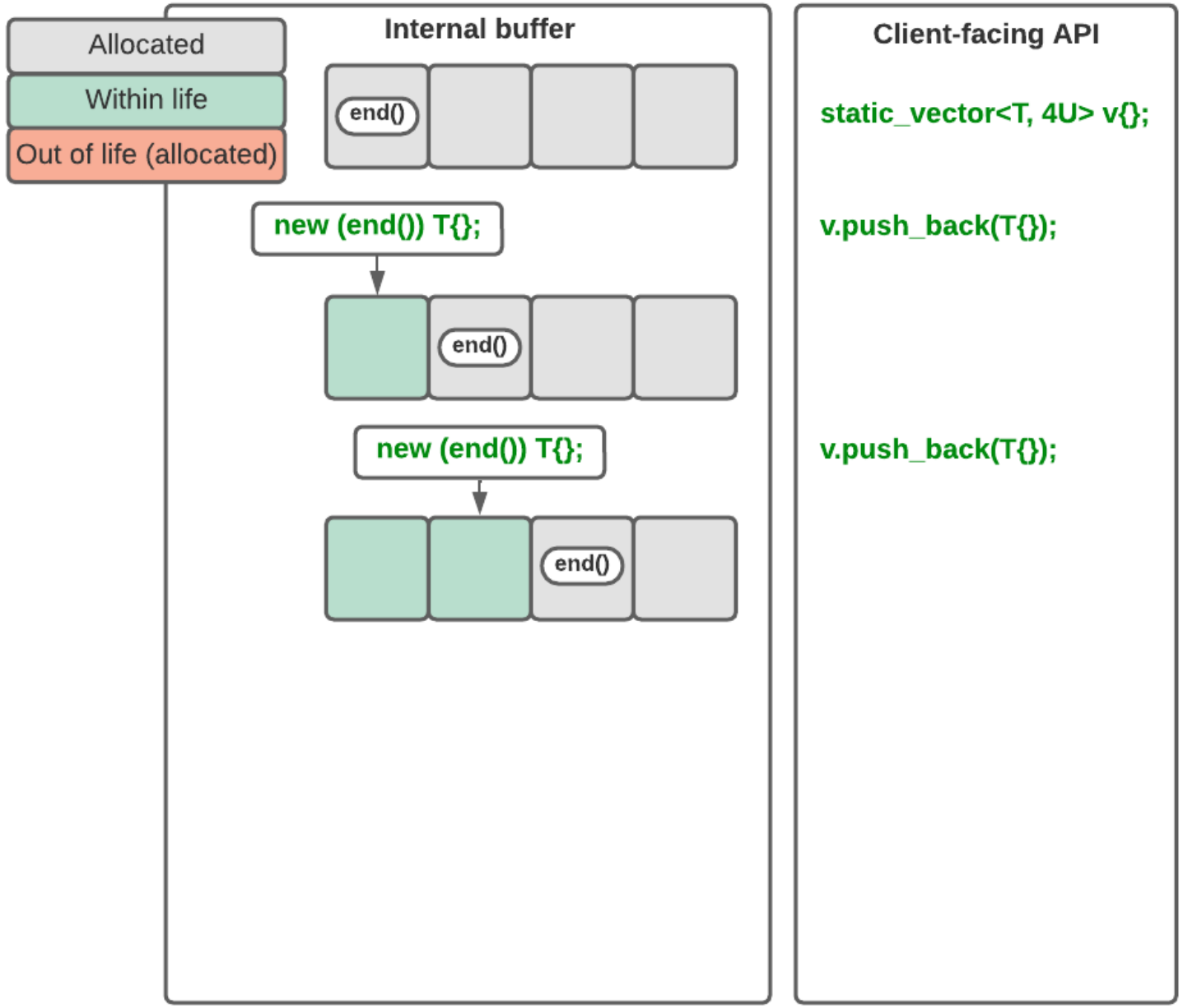
Memory safety

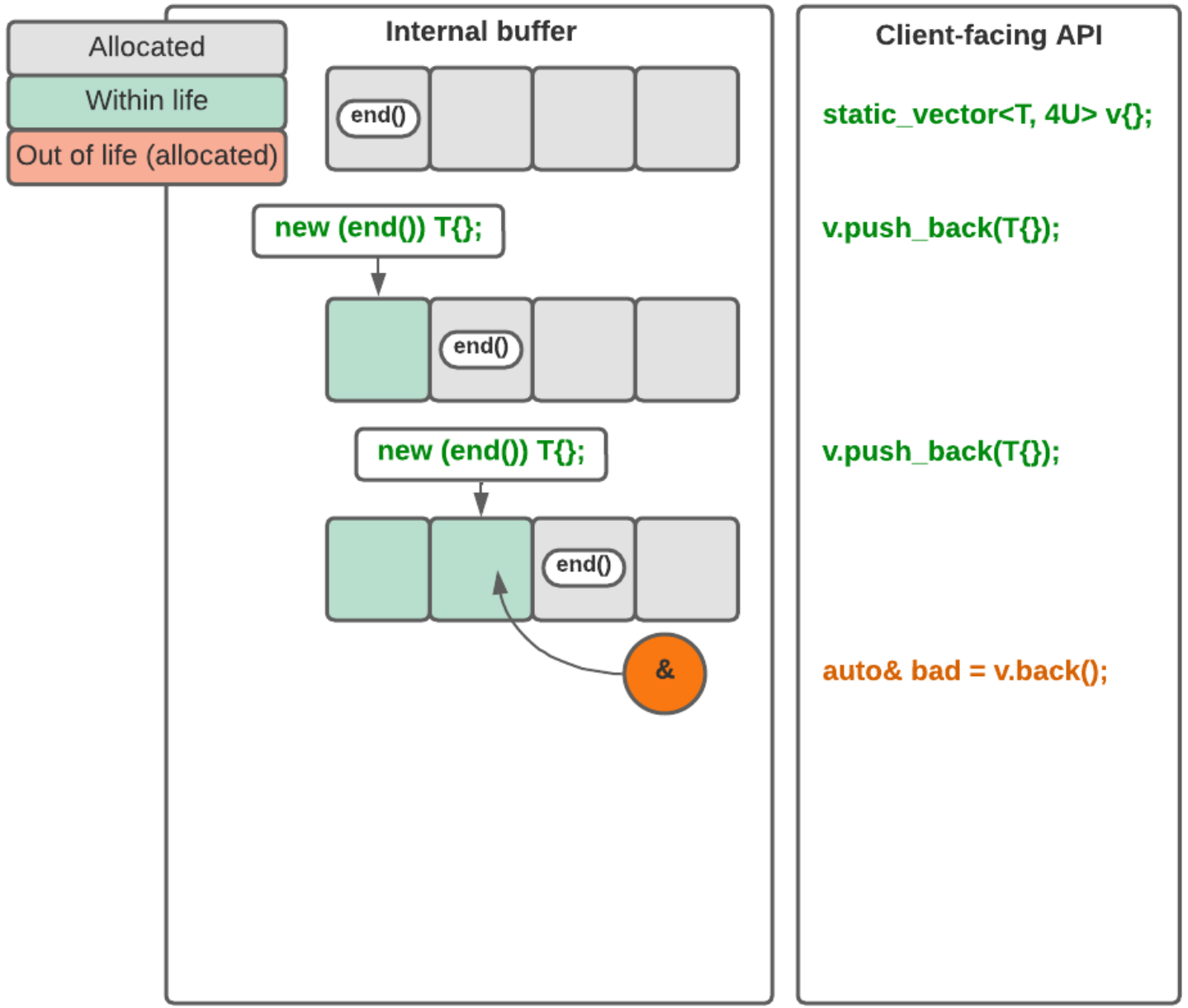
- ✓ Dynamic memory allocations?
  - ✓ Init-phase vs run-phase
  - ✓ In-house mallblocker tool

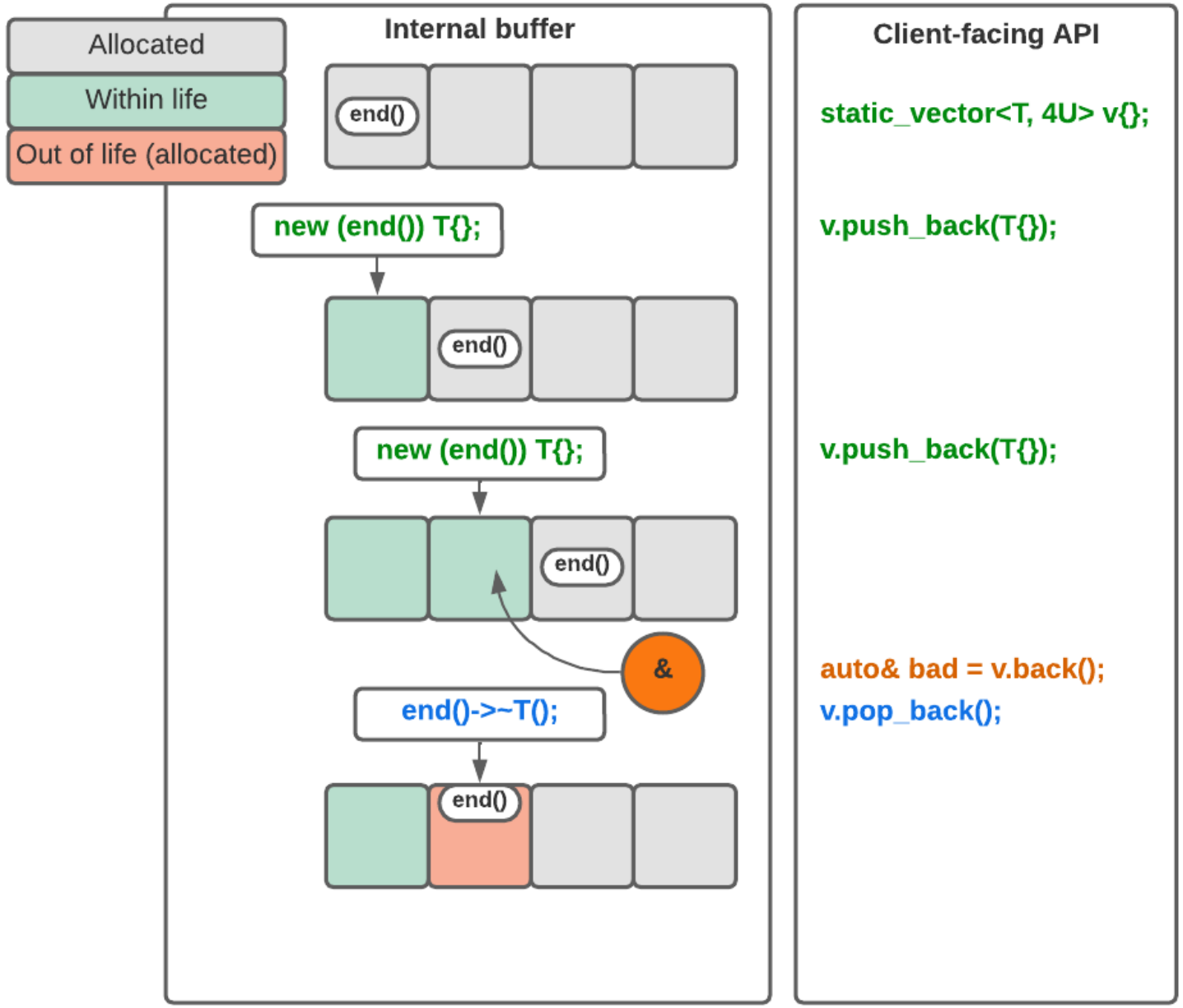
Memory safety

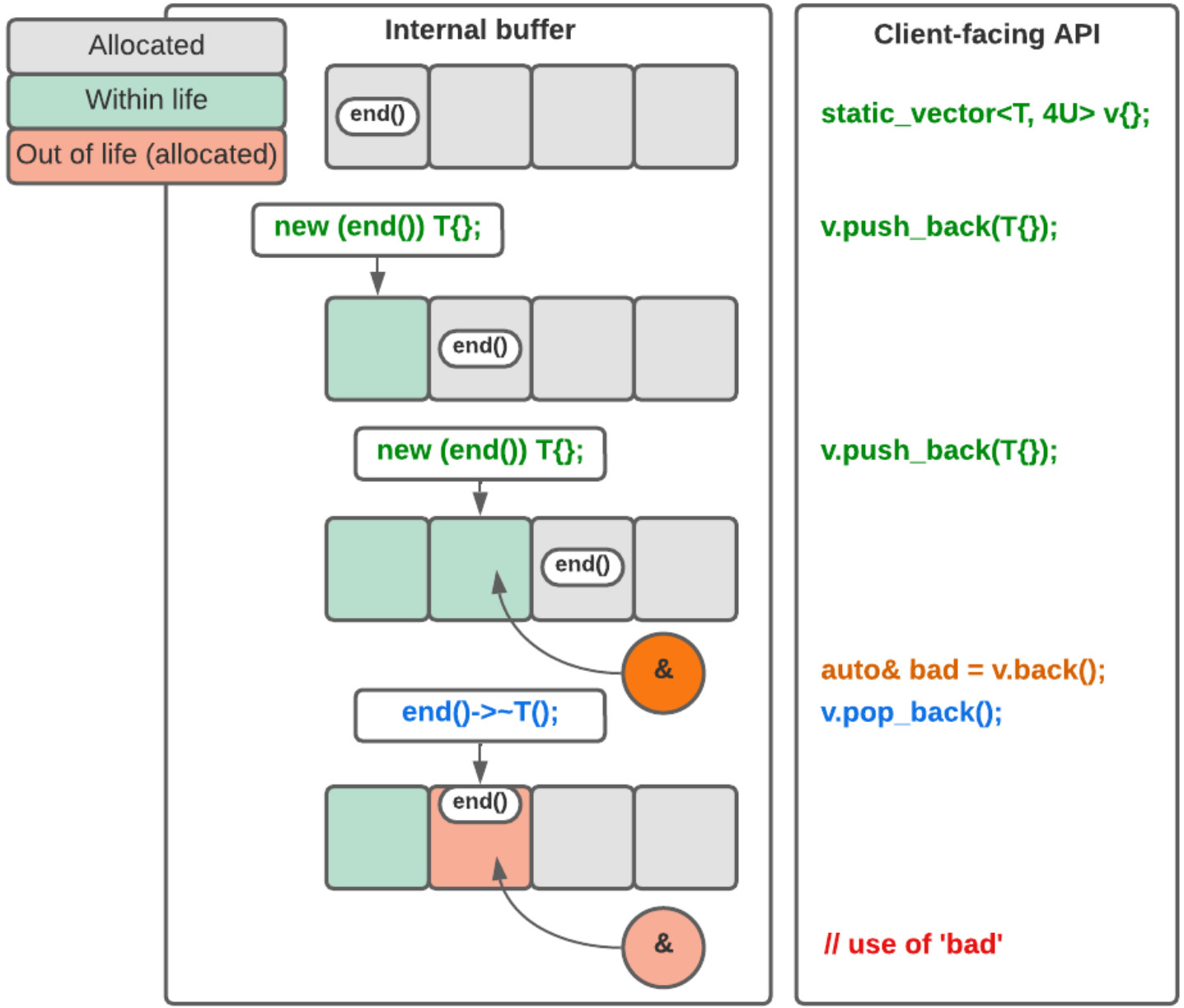
- ✓ Dynamic memory allocations?
  - ✓ Init-phase vs run-phase
  - ✓ In-house mallblocker tool
- ✓ Valgrind & AddressSanitizer (ASan)
  - ✓ (Example) Needs some help in lack to capture a wider scope of error

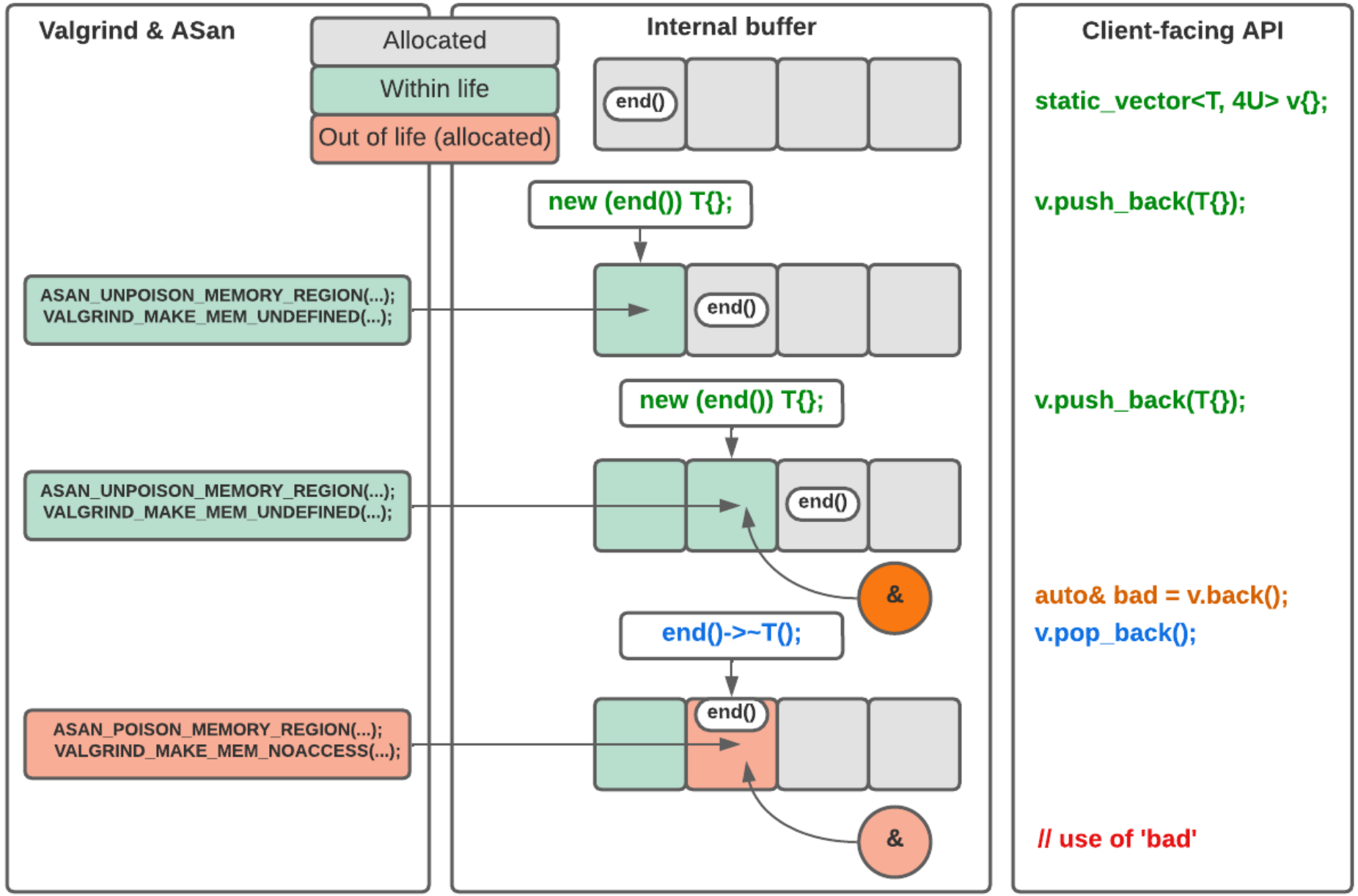

Allocated
Within life
Out of life (allocated)
Internal buffer
Client-facing API


Allocated
Within life
Out of life (allocated)
Internal buffer
end()
new (end()) T{};
end()
new (end()) T{};
end()
Client-facing API
static_vector<T, 4U> v{};
v.push_back(T{});
v.push_back(T{});


Allocated
Within life
Out of life (allocated)
Internal buffer
end()
new (end()) T{};
end()
new (end()) T{};
end()
&
Client-facing API
static_vector<T, 4U> v{};
v.push_back(T{});
v.push_back(T{});
auto& bad = v.back();


Allocated
Within life
Out of life (allocated)
Internal buffer
end()
new (end()) T{};
end()
new (end()) T{};
end()
&
end()->~T();
end()
Client-facing API
static_vector<T, 4U> v{};
v.push_back(T{});
v.push_back(T{});
auto& bad = v.back();
v.pop_back();


Allocated
Within life
Out of life (allocated)
Internal buffer
end()
new (end()) T{};
end()
new (end()) T{};
end()
&
end()->~T();
end()
&
Client-facing API
static_vector<T, 4U> v{};
v.push_back(T{});
v.push_back(T{});
auto& bad = v.back();
v.pop_back();
// use of 'bad'


Valgrind & ASan
Allocated
Within life
Out of life (allocated)
Internal buffer
end()
new (end()) T{};
ASAN_UNPOISON_MEMORY_REGION(...);
VALGRIND_MAKE_MEM_UNDEFINED(...);
end()
new (end()) T{};
ASAN_UNPOISON_MEMORY_REGION(...);
VALGRIND_MAKE_MEM_UNDEFINED(...);
end()
&
end()->~T();
ASAN_POISON_MEMORY_REGION(...);
VALGRIND_MAKE_MEM_NOACCESS(...);
end()
&
Client-facing API
static_vector<T, 4U> v{};
v.push_back(T{});
v.push_back(T{});
auto& bad = v.back();
v.pop_back();
// use of 'bad'

Memory safety and beyond (language safety)

- ✓ Strong industry focus (beyond safety-critical domain) on memory safe languages
- ✓ C++ & WG21: P2816R0
  - ✓ Emphasis on type and resource safety
  - ✓ Many different notions of what "safety" encompasses
  - ✓ *Safety profiles*: "A cocktail of techniques" comes with a cost (cognitive complexity)

Memory safety and beyond (language safety)

- ✓ Strong industry focus (beyond safety-critical domain) on memory safe languages
- ✓ C++ & WG21: P2816R0
  - ✓ Emphasis on type and resource safety
  - ✓ Many different notions of what "safety" encompasses
  - ✓ *Safety profiles*: "A cocktail of techniques" comes with a cost (cognitive complexity)
- ✓ Not necessarily initially a focus on topics that are essential to the safety-critical domain
  - ✓ E.g.: **contracts** is a high-value target for safety-critical space, but not necessarily the highest prioritization for the C++ language safety space

Standard and core libraries

- ✓ Vendor-shipped
  - ✓ Pros: written by the pros, ...
  - ✓ Cons: need of qualification evidence, not aimed at safety-critical domain to begin with, ...

Standard and core libraries

- ✓ Vendor-shipped
  - ✓ Pros: written by the pros, ...
  - ✓ Cons: need of qualification evidence, not aimed at safety-critical domain to begin with, ...
- ✓ Inhouse versions
  - ✓ Pros:
    - ✓ Poor man's contracts
      - ✓ The Lakos Rule
    - ✓ Allows removing inherently unsafe APIs
    - ✓ Backporting yet-to-be implemented (e.g. in C++14) libs
    - ✓ Consolidation of in-house non-STL core libs and in-house STL implementations
  - ✓ Cons: reinventing the wheel, sub-par (non-vendor) implementations, ...

Standard and core libraries

- ✓ Open-source libs and STL backports?
  - ✓ Challenging to use in context of safety-critical applications
    - ✓ … even for the parts/software components that are not safety-critical (more on this later)
  - ✓ Bazel and Bazel meta-data can be used to protect from unintentional misuse

Unit testing & test sufficiency

- Some basic premises:
  - Dijkstra on **test sufficiency**: "Program testing can be used to show the presence of bugs, but never to show their absence"
  - Titus Winters on **test integrity**: "Tests should fail because the code under tests fails, and for no other reason"

Unit testing & test sufficiency

- Some basic premises:
  - Dijkstra on **test sufficiency**: "Program testing can be used to show the presence of bugs, but never to show their absence"
  - Titus Winters on **test integrity**: "Tests should fail because the code under tests fails, and for no other reason"
- Test integrity
  - Apply coding guidelines also for tests (to a reasonable extent)
  - -Werror on tests
  - Promote a testing first mentality (TDD, …)

Unit testing & test sufficiency

- ✓ Some basic premises:
  - ✓ Dijkstra on **test sufficiency**: "Program testing can be used to show the presence of bugs, but never to show their absence"
  - ✓ Titus Winters on **test integrity**: "Tests should fail because the code under tests fails, and for no other reason"
- ✓ Test integrity
  - ✓ Apply coding guidelines also for tests (to a reasonable extent)
  - ✓ -Werror on tests
  - ✓ Promote a testing first mentality (TDD, …)
- ✓ Test sufficiency: we need something other than the tests themselves

Unit testing & test sufficiency

- Using structural code metrics as a criteria for **test sufficiency**?
  - Goodheart's Law: "Any observed statistical regularity will tend to collapse once pressure is placed upon it for control purposes"

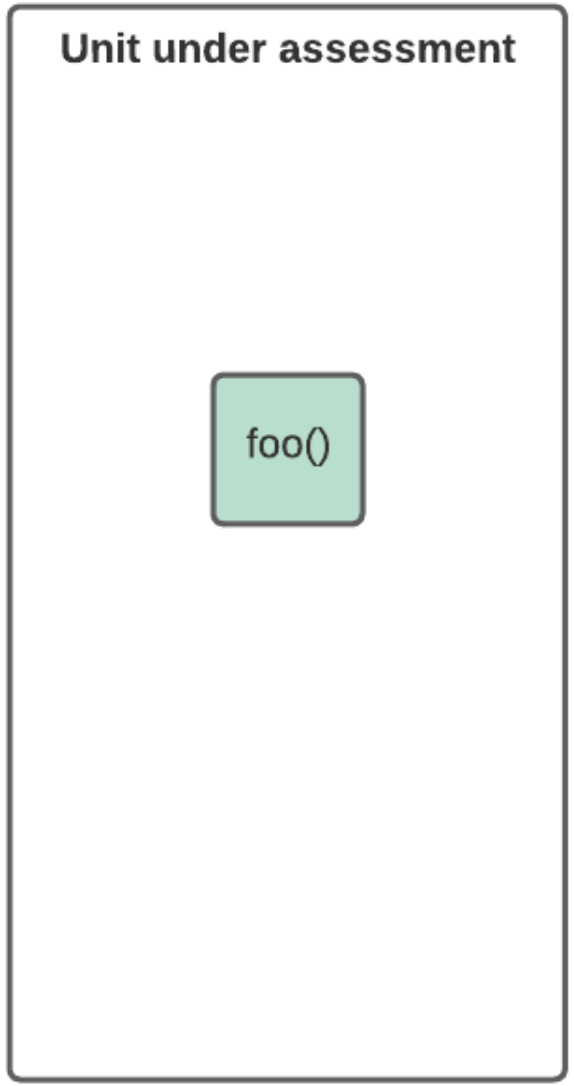
Unit testing & test sufficiency

- ✓ Using structural code metrics as a criteria for **test sufficiency**?
  - ✓ Goodheart's Law: "Any observed statistical regularity will tend to collapse once pressure is placed upon it for control purposes"
- ✓ Metric: code coverage
  - ✓ Not a silver bullet
  - ✓ Highly and arguably overly so relied on by safety standards

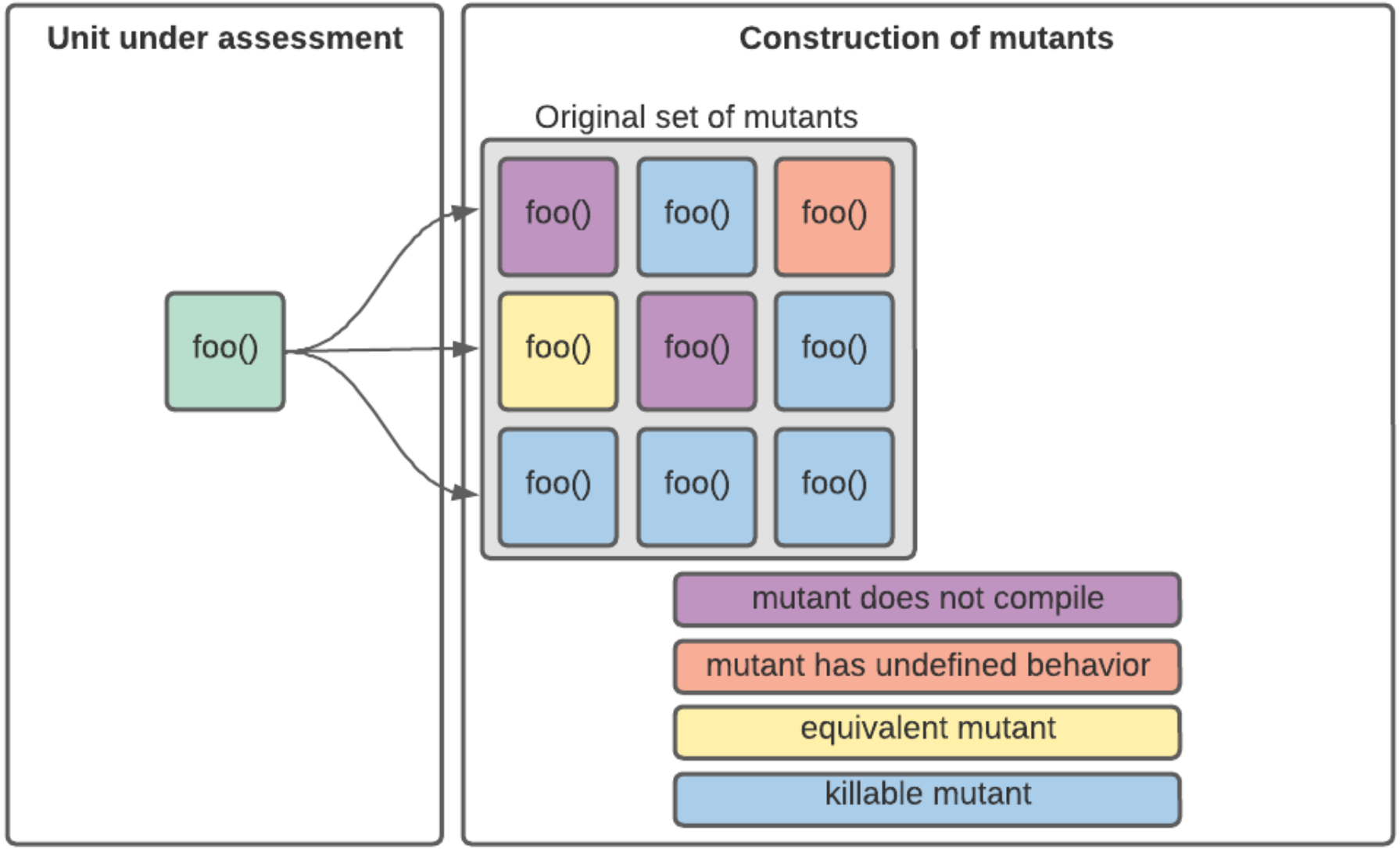
Unit testing & test sufficiency

- ✓ Using structural code metrics as a criteria for **test sufficiency**?
  - ✓ Goodheart's Law: "Any observed statistical regularity will tend to collapse once pressure is placed upon it for control purposes"
- ✓ Metric: code coverage
  - ✓ Not a silver bullet
  - ✓ Highly and arguably overly so relied on by safety standards
- ✓ Metric: mutation score (mutation testing)
  - ✓ Highest fault revelation among all common structural testing criteria
  - ✓ Difficult to use in practice

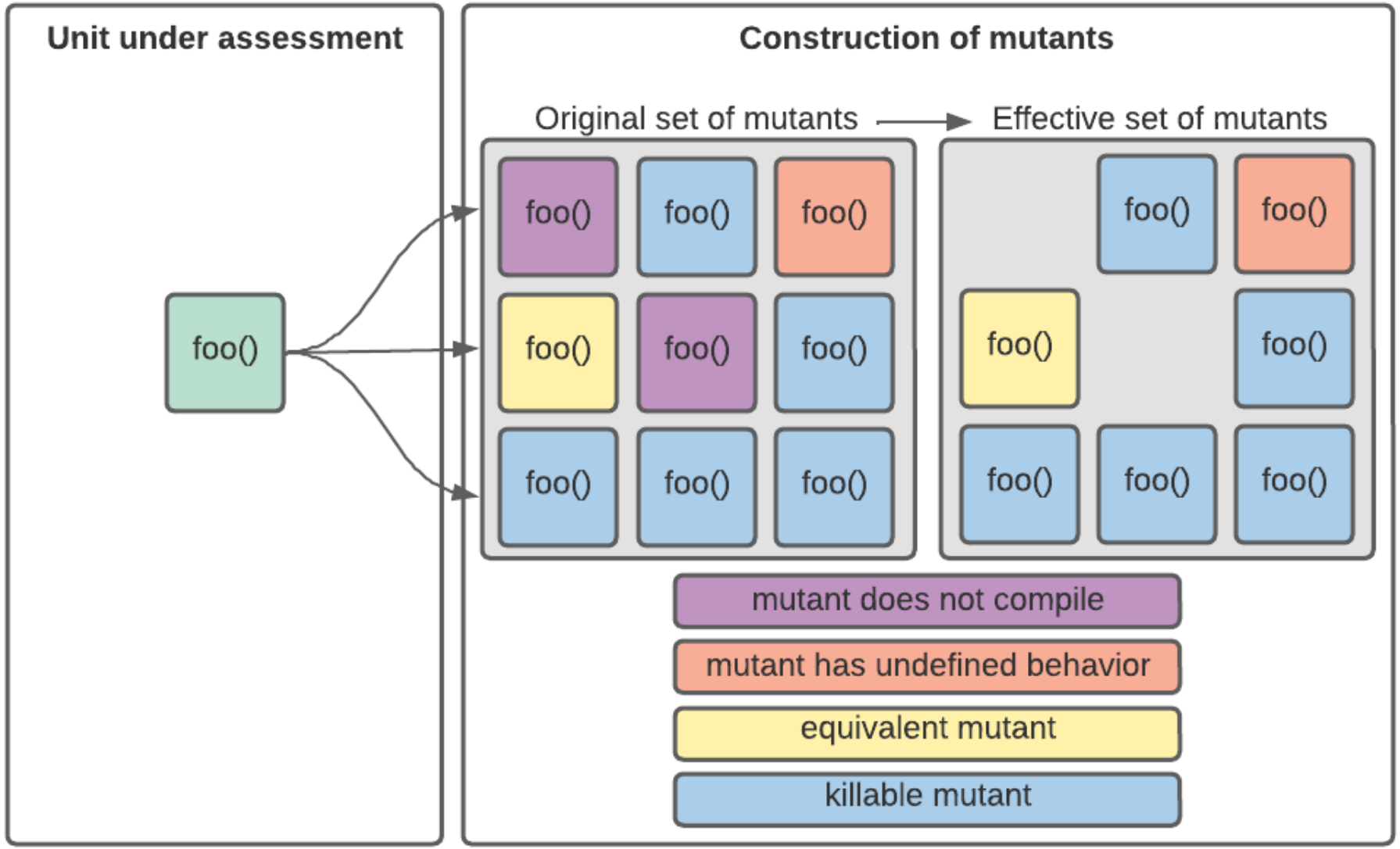
Unit testing & test sufficiency: mutation testing



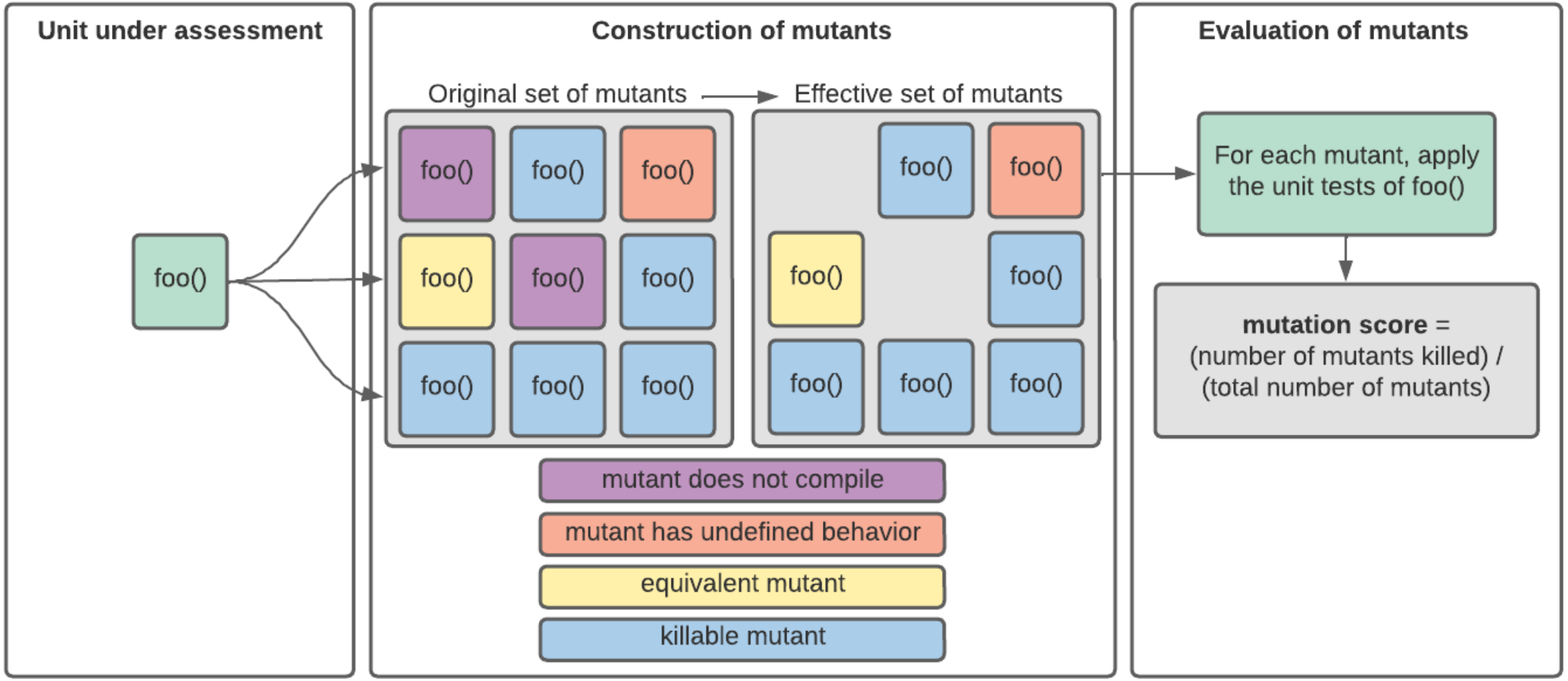
Unit testing & test sufficiency: mutation testing



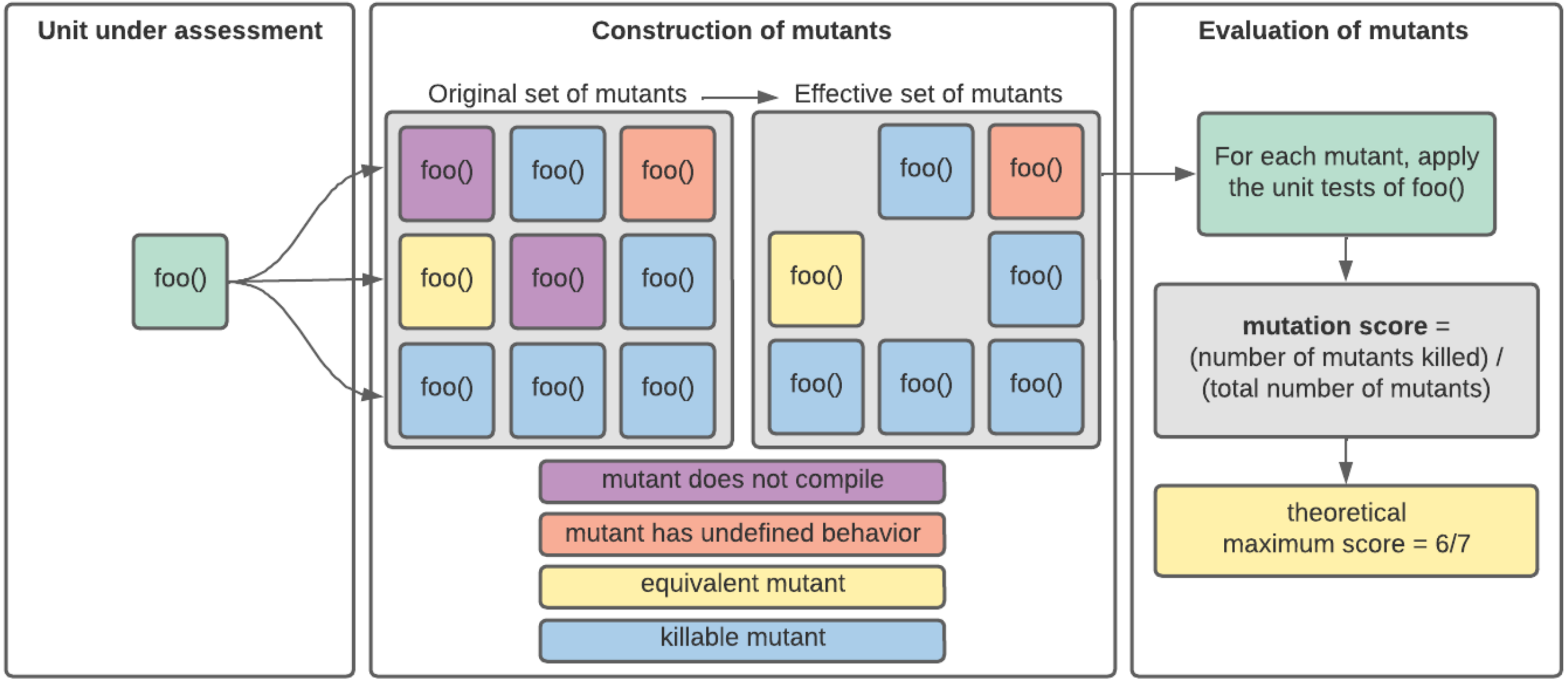
Unit testing & test sufficiency: mutation testing



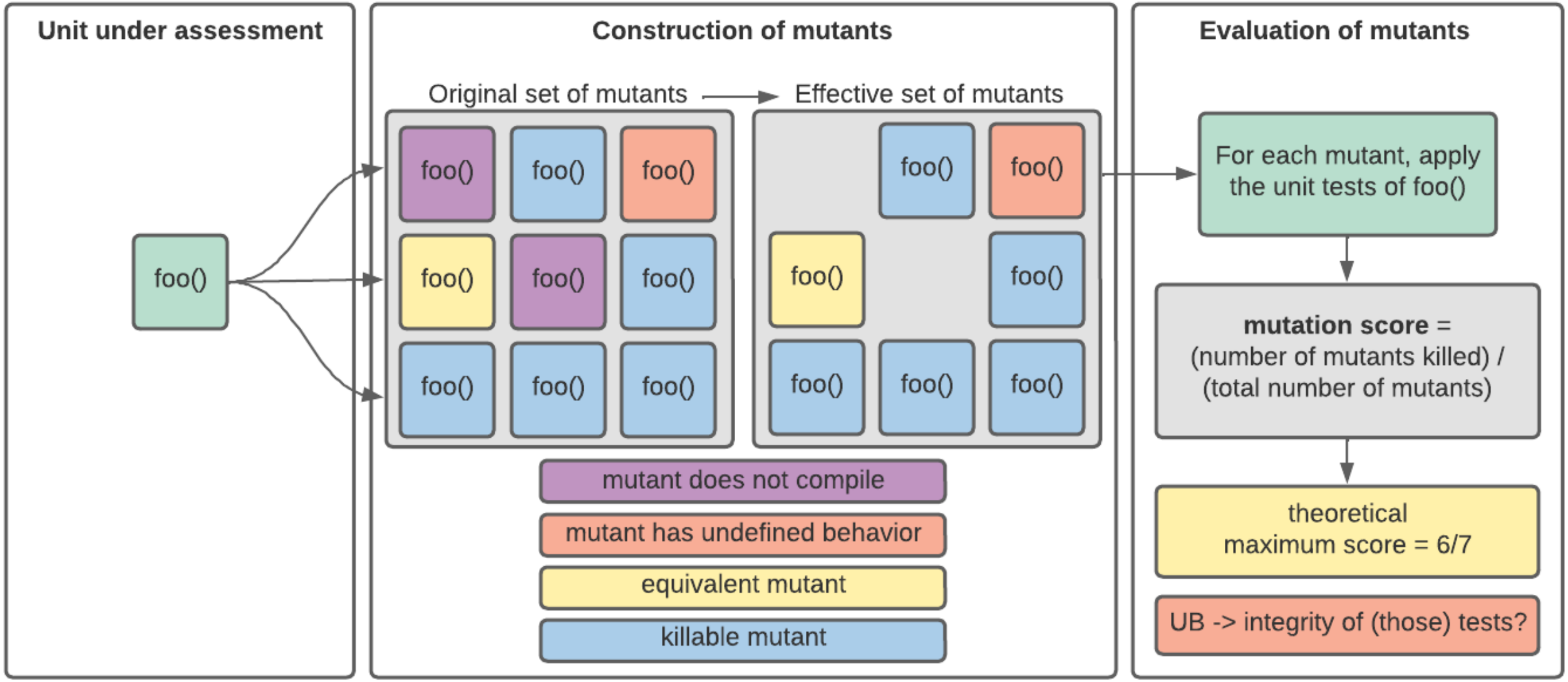
Unit testing & test sufficiency: mutation testing



Unit testing & test sufficiency: mutation testing



Unit testing & test sufficiency: mutation testing



Unit testing & test sufficiency

- ✓ Bleeding-edge methods
  - ✓ E.g.: finding hot spots where test sufficiency may be
    - ✓ … more likely to be subpar?
    - ✓ … of more importance than in other areas?

Unit testing & test sufficiency

- ✓ Bleeding-edge methods
  - ✓ E.g.: finding hot spots where test sufficiency may be
    - ✓ … more likely to be subpar?
    - ✓ … of more importance than in other areas?
- ✓ (Poor man's) Contracts
  - ✓ Equivalence class testing
  - ✓ Negative testing

Unit testing & test sufficiency

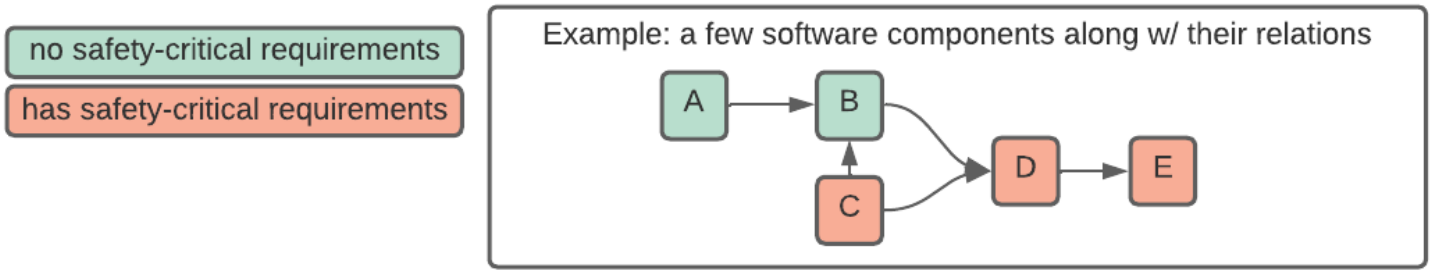
- ✓ Bleeding-edge methods
  - ✓ E.g.: finding hot spots where test sufficiency may be
    - ✓ … more likely to be subpar?
    - ✓ … of more importance than in other areas?
- ✓ (Poor man's) Contracts
  - ✓ Equivalence class testing
  - ✓ Negative testing
- ✓ Fault injection
  - ✓ Mutation testing
  - ✓ Out-of-contract testing

A few words on requirements and requirements testing

- ✓ (Derived) Requirements testing
  - ✓ Requirements coverage vs code coverage: common ISO 26262 misconceptions

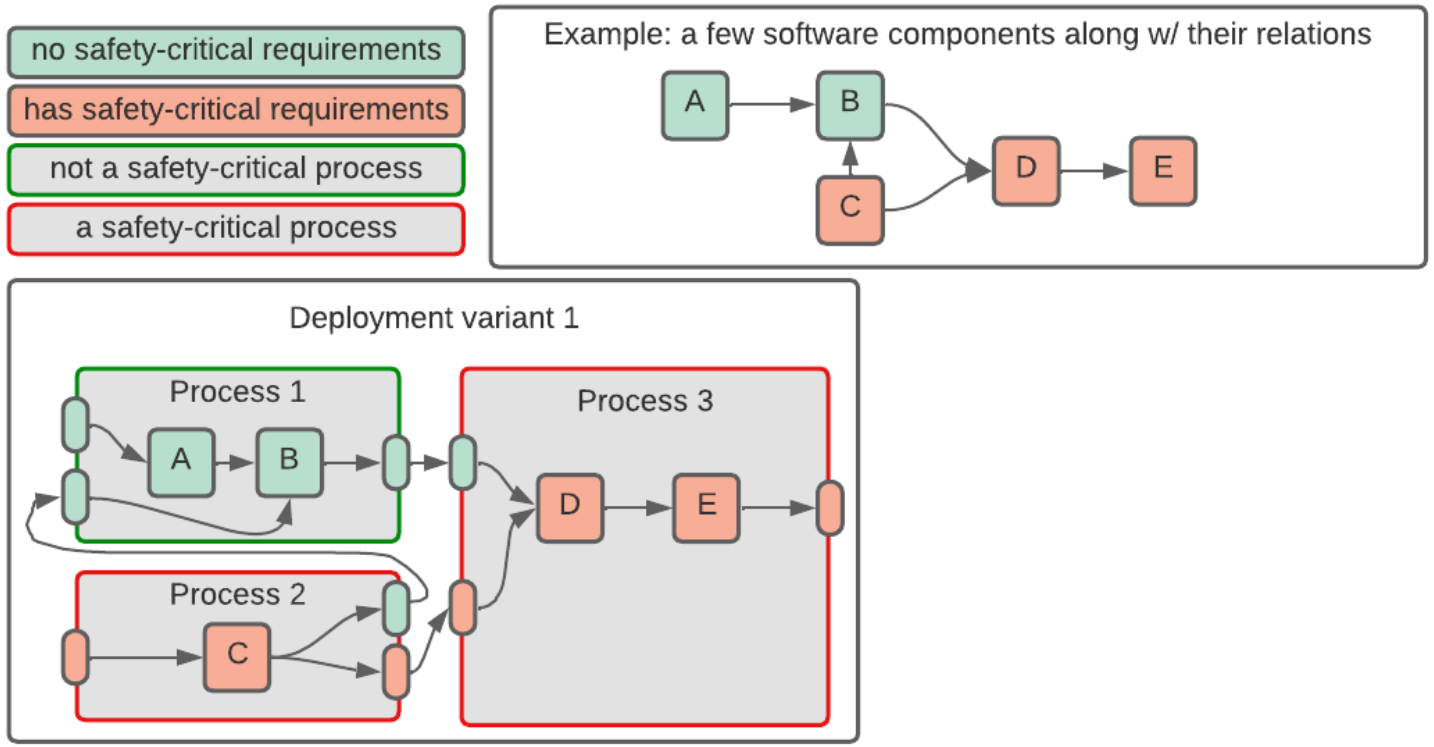
A few words on requirements and requirements testing

- ✓ (Derived) Requirements testing
  - ✓ Requirements coverage vs code coverage: common ISO 26262 misconceptions
- ✓ (Non-derived) Freedom from interference requirements



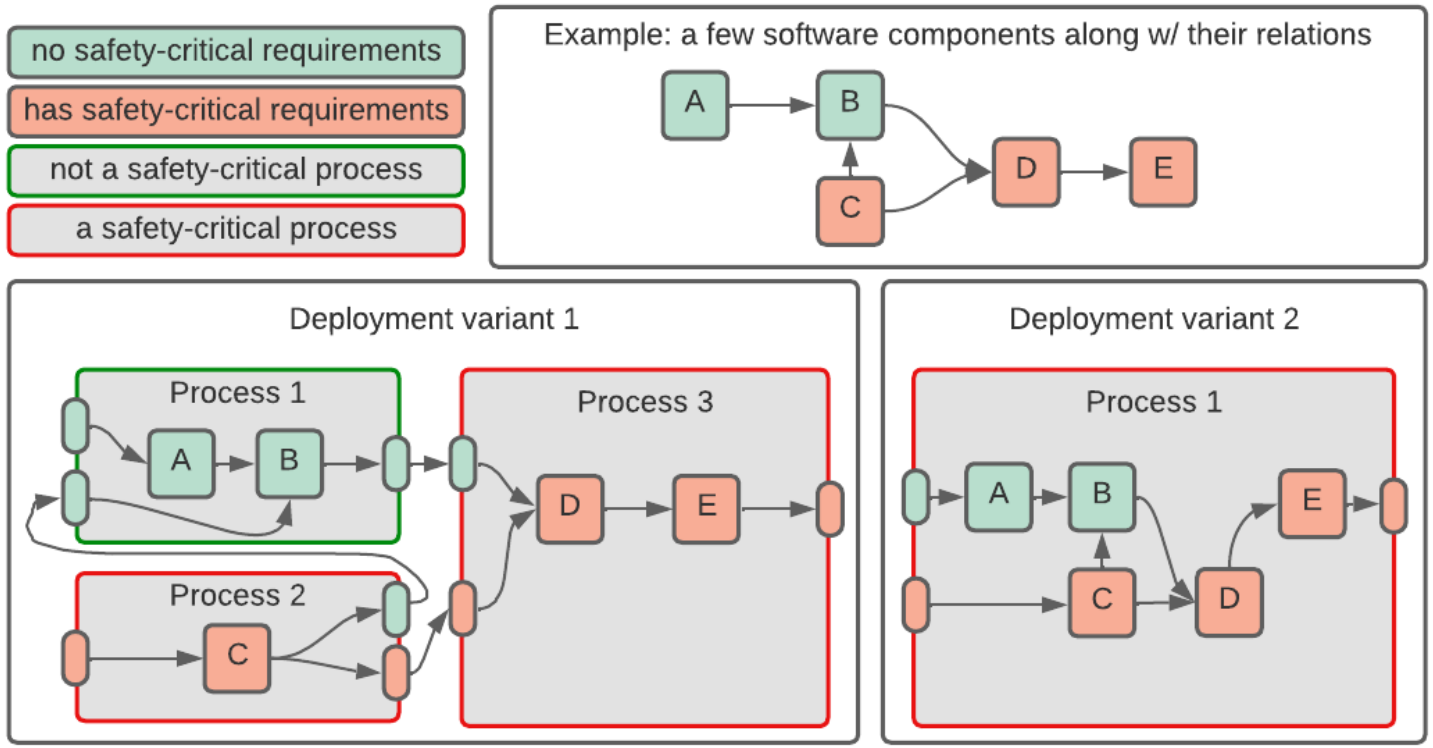
A few words on requirements and requirements testing

- ✓ (Derived) Requirements testing
  - ✓ Requirements coverage vs code coverage: common ISO 26262 misconceptions
- ✓ (Non-derived) Freedom from interference requirements



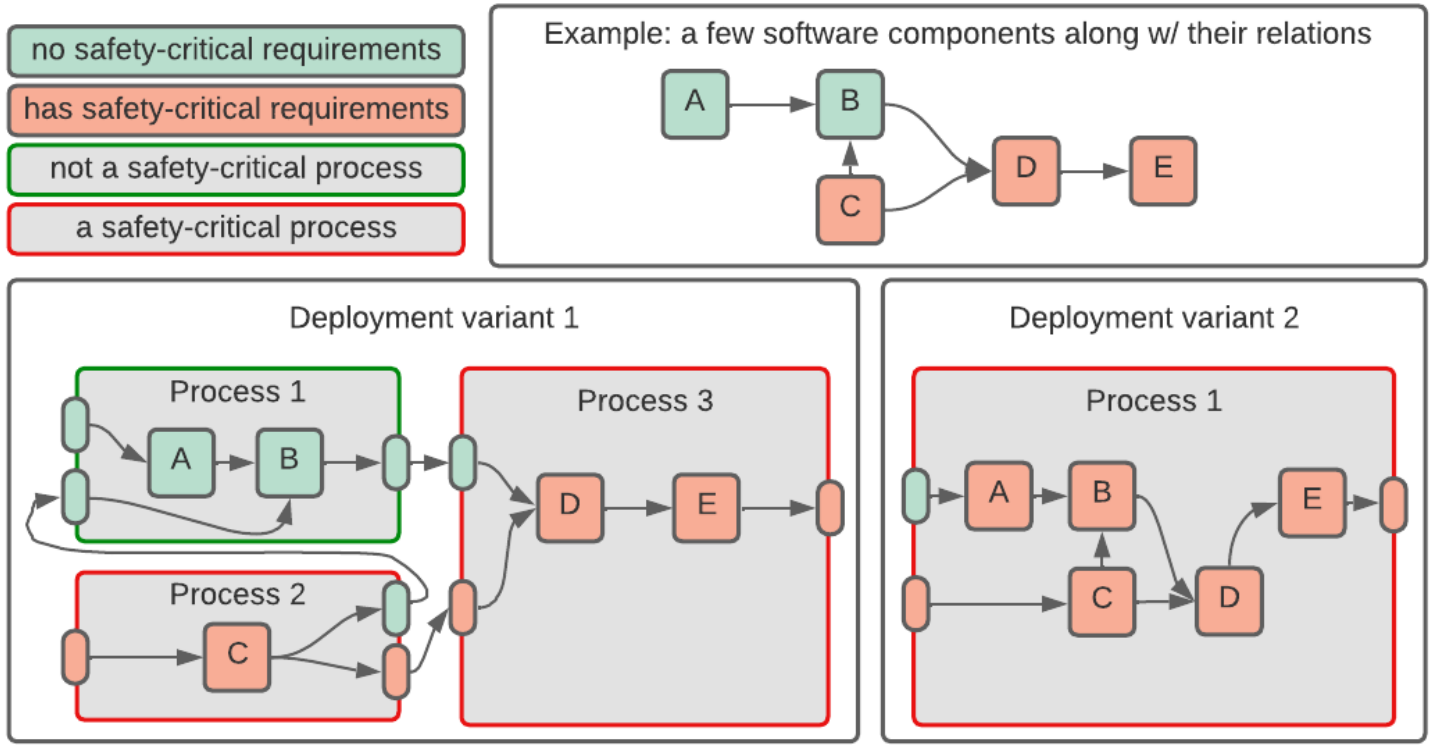
A few words on requirements and requirements testing

- ✓ (Derived) Requirements testing
  - ✓ Requirements coverage vs code coverage: common ISO 26262 misconceptions
- ✓ (Non-derived) Freedom from interference requirements



A few words on requirements and requirements testing

- ✓ (Derived) Requirements testing
  - ✓ Requirements coverage vs code coverage: common ISO 26262 misconceptions
- ✓ (Non-derived) Freedom from interference requirements



Human factors & safety culture

- ✓ Nothing replaces software safety culture

Human factors & safety culture

- ✓ Nothing replaces software safety culture
- ✓ Automation cannot replace critical thinking
- ✓ Code review is an essential part

Human factors & safety culture

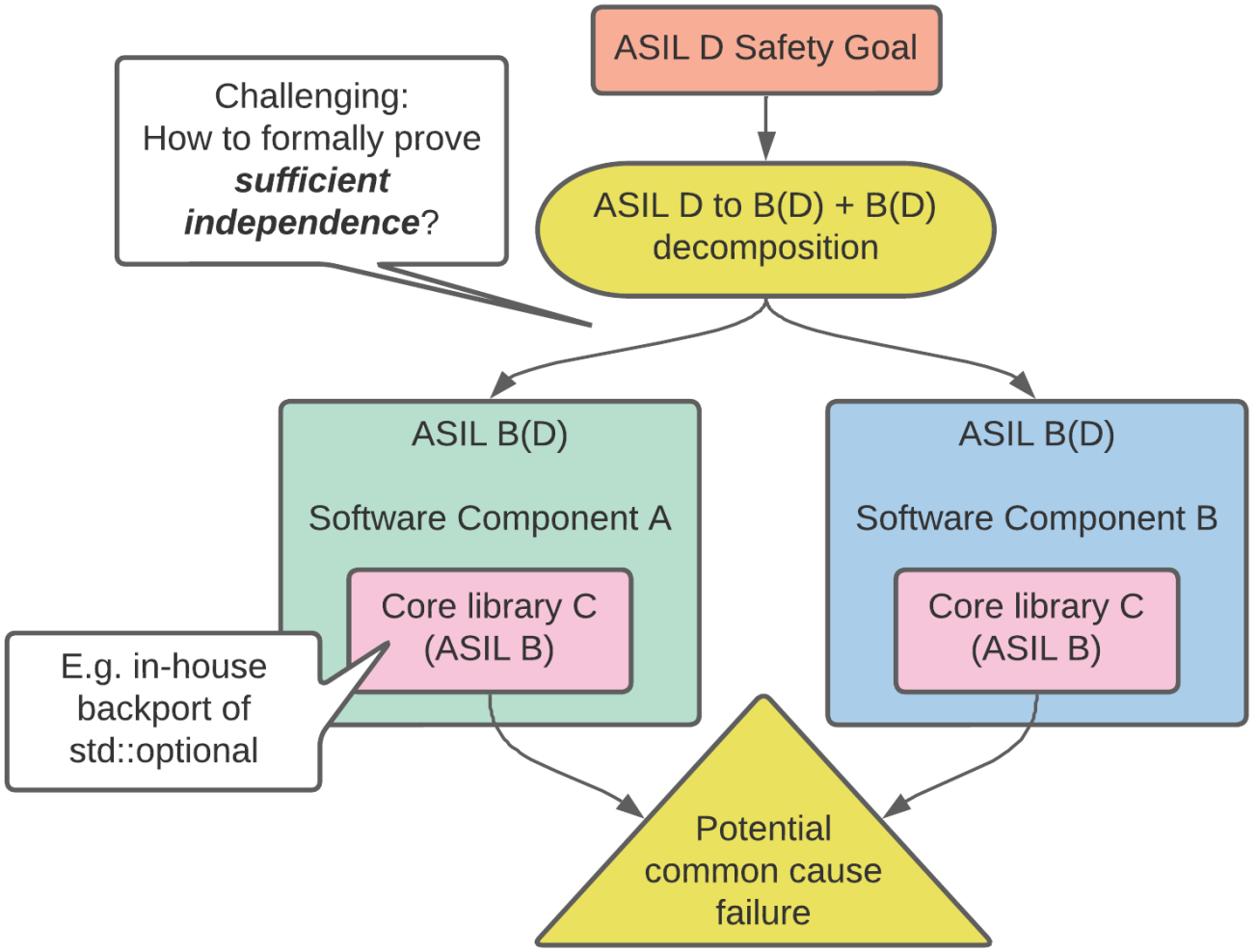
- ✓ Nothing replaces software safety culture
- ✓ Automation cannot replace critical thinking
- ✓ Code review is an essential part
- ✓ Developer complacency when reaching higher automation

Human factors & safety culture

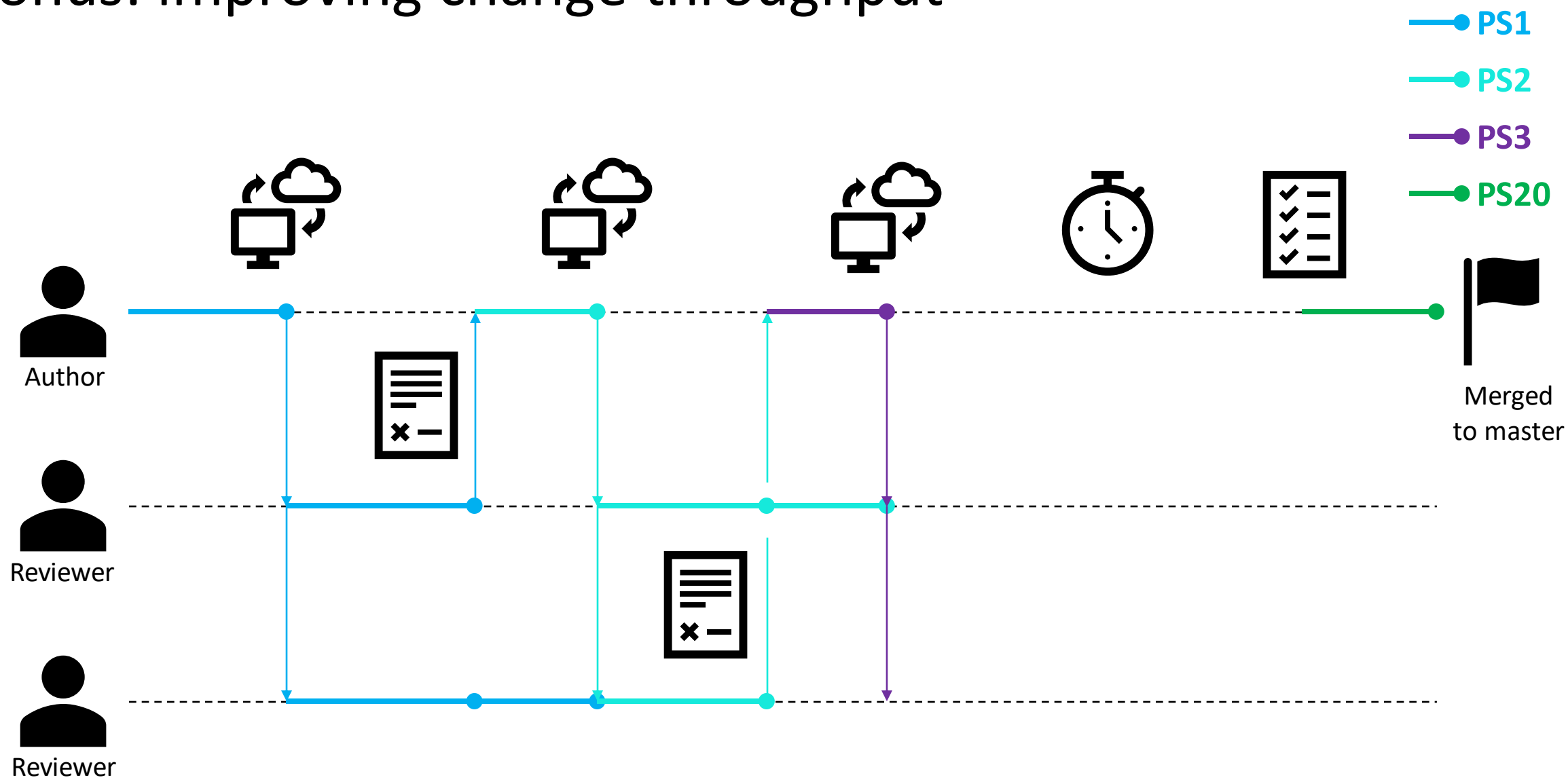
- ✓ Nothing replaces software safety culture
- ✓ Automation cannot replace critical thinking
- ✓ Code review is an essential part
- ✓ Developer complacency when reaching higher automation
- ✓ Repository uniformity facilitates safety culture
  - ✓ E.g.: within reasonable limits, consolidate coding requirements on safety- and non-safety-critical code
  - ✓ Additional pro: facilitates robustness towards higher-level safety and deployment architectural changes

Safety vs software engineering: perceived contradictions

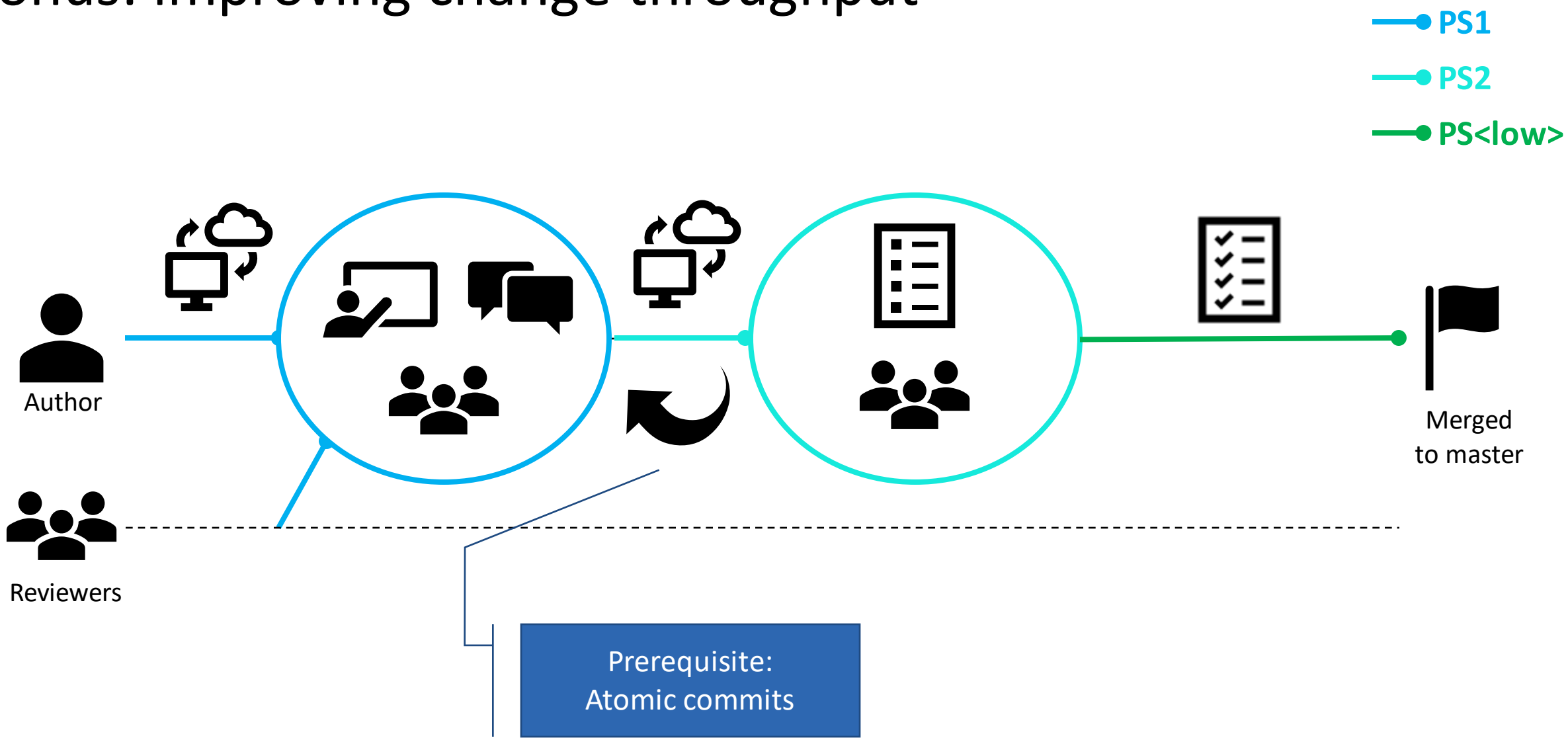
- ✓ Common cause failures vs code re-use



Bonus: improving change throughput



Bonus: improving change throughput



A few other interesting topics (out of scope today)

- ✓ How to handle debug-only code
- ✓ Error-handling guidelines on a unit/library level
- ✓ Floating point guidelines (**difficult**)
- ✓ Software architecture and fault handling on architectural elements' boundaries