

---

# Verification of Software Architecture Security Properties using a Knowledge Graph

**Jeisson Vergara-Vargas, Salah Sadou, Chouki Tibermacine, Felipe Restrepo-Calle**

Journée commune au GDR RSD, GPL (GT GLSEC) et SI (GT SSLR) sur la sécurité des piles réseau

---

**Orléans, France**  
September 30, 2024

# Motivation

## Problem Statement

- For efficiency and cost reasons it is important to **ensure** (validate) software's **properties** at the **earlier stages** of the development life cycle. (Tuma et al., MODELS'20)
- **Security** as a software **property** needs also to be addressed at **design** stage by the architect. However, they do not have the efficient means for that. (Mallouli, ICSTW'22)
- A **Secure by Design** problem. (Dan Bergh Johnsson et al., 2019)

# Motivation

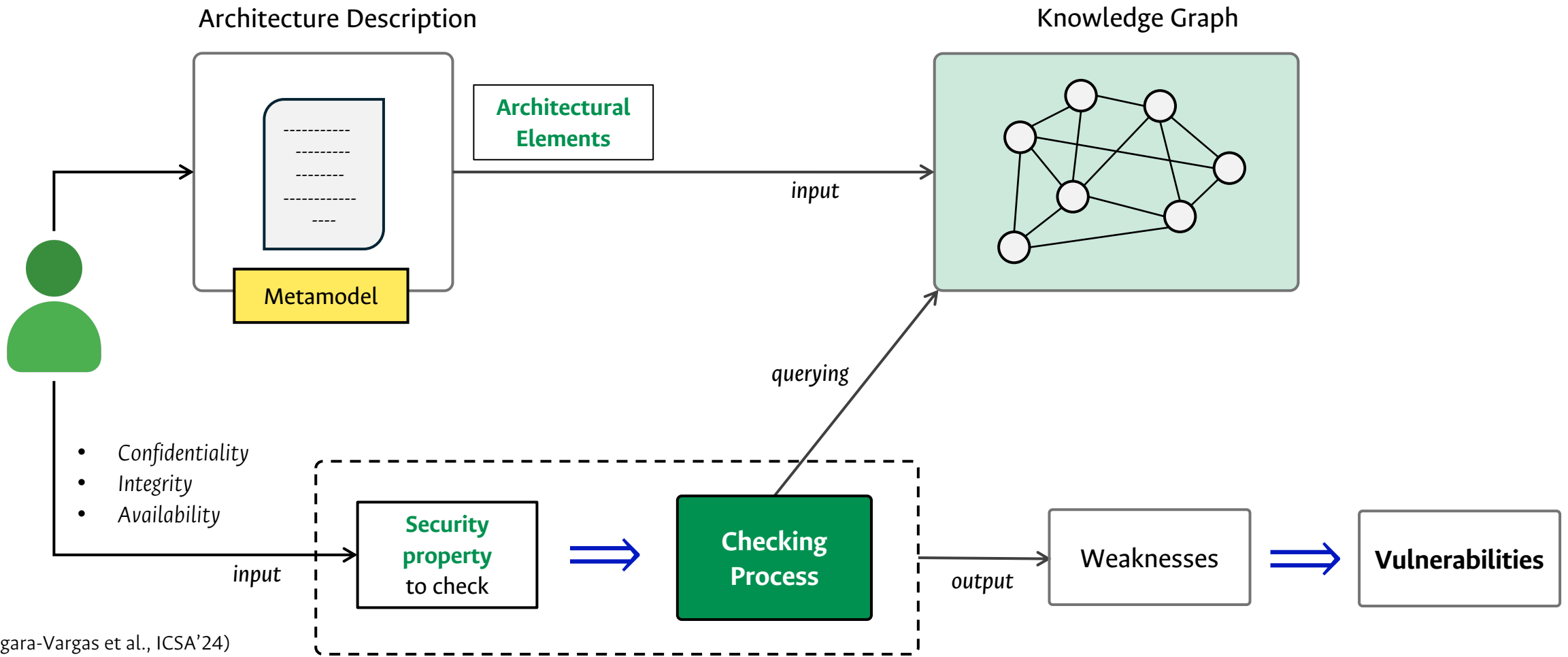
## Research Question

Is it possible to **check security properties** from the **software architecture description** of a software system?

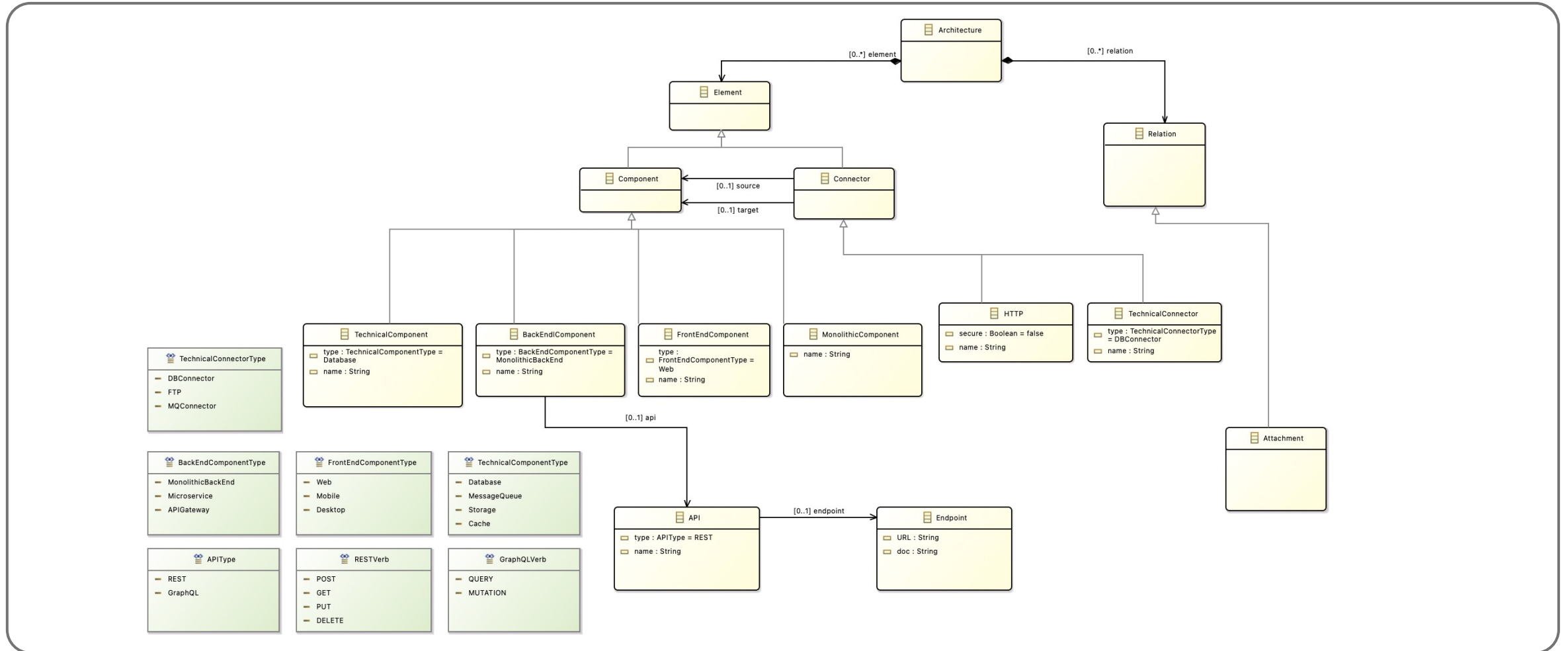


Method for **verifying security properties** at the **software architectural** level, using a **knowledge graph**.

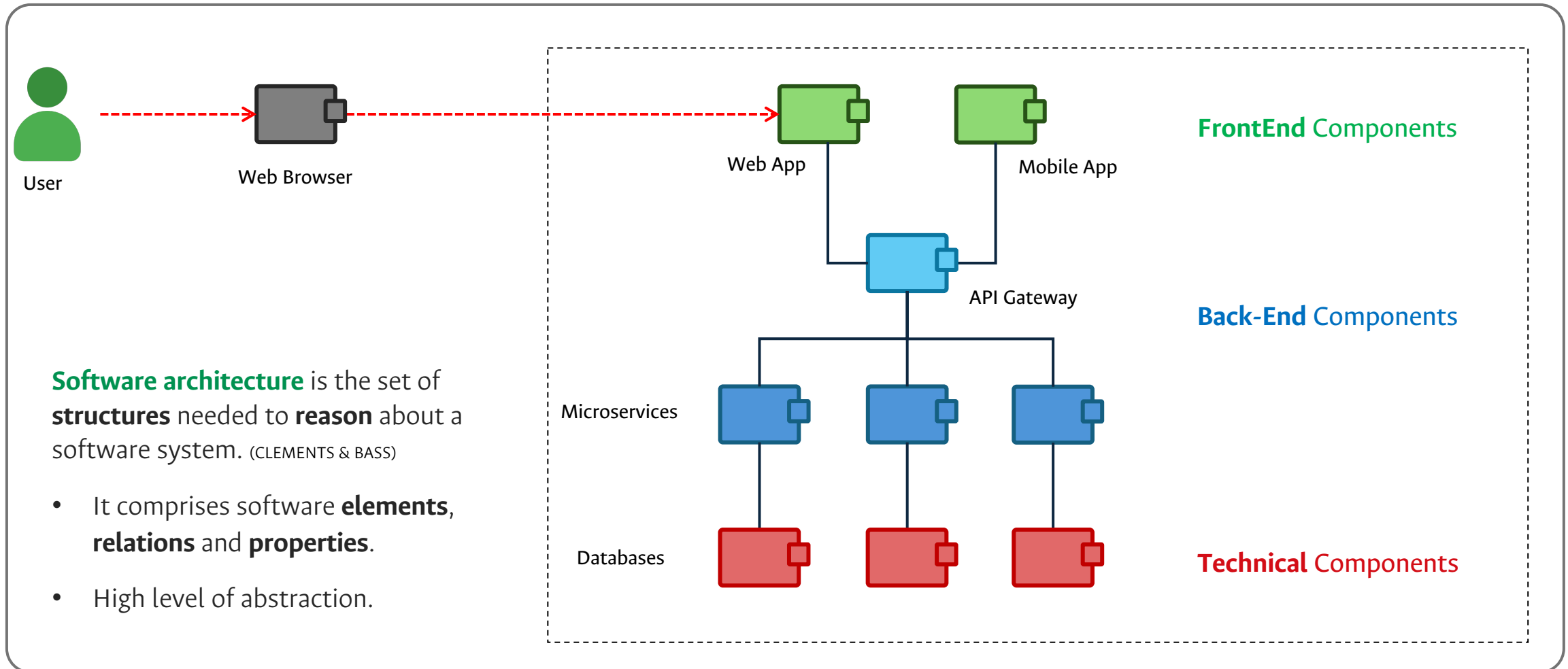
# General Approach for Sarch-Checks



# Architecture Description: Metamodel



# Architecture Description: Model

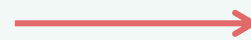


## Knowledge Graph: Sarch-Knows

There exist some **contributions** in the identification of **security** aspects related to the **architecture** of software systems.

[Santos et al., ICSA'17]

However



There is no contribution that provides a **comprehensive description** between security **concepts** and architectural **concepts**.

# Knowledge Graph: Sarch-Knows

## Observations:

- Architectural concepts are stable.
- Security concepts are evolving.

## Needs:

- An evolutive representation containing both sets of concepts.
- Ability to check properties and to look for evidences.

⇒ Use of a knowledge graph with a query language.

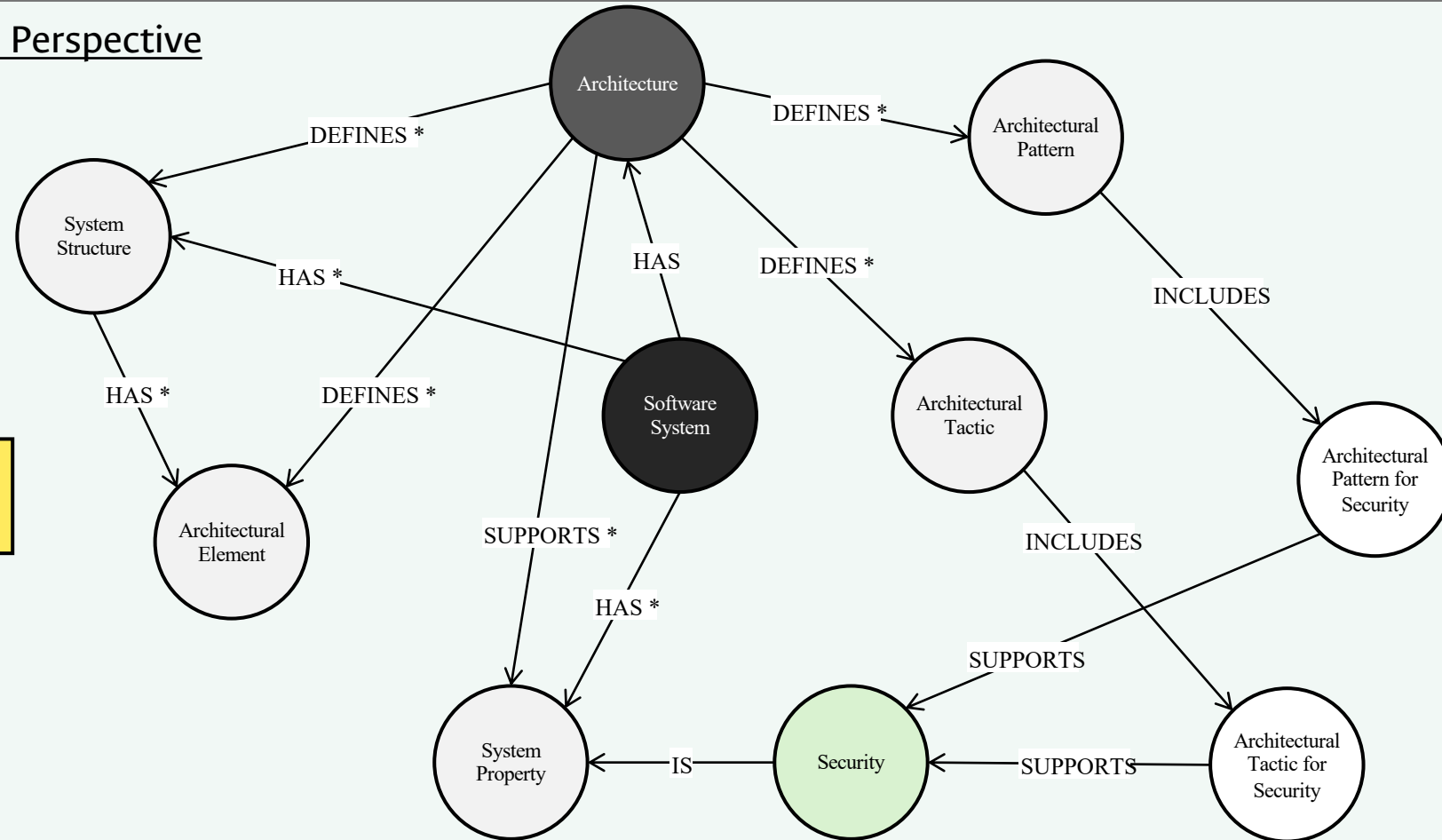
E.g. 



# Knowledge Graph: Sarch-Knows

## Software Architecture Perspective

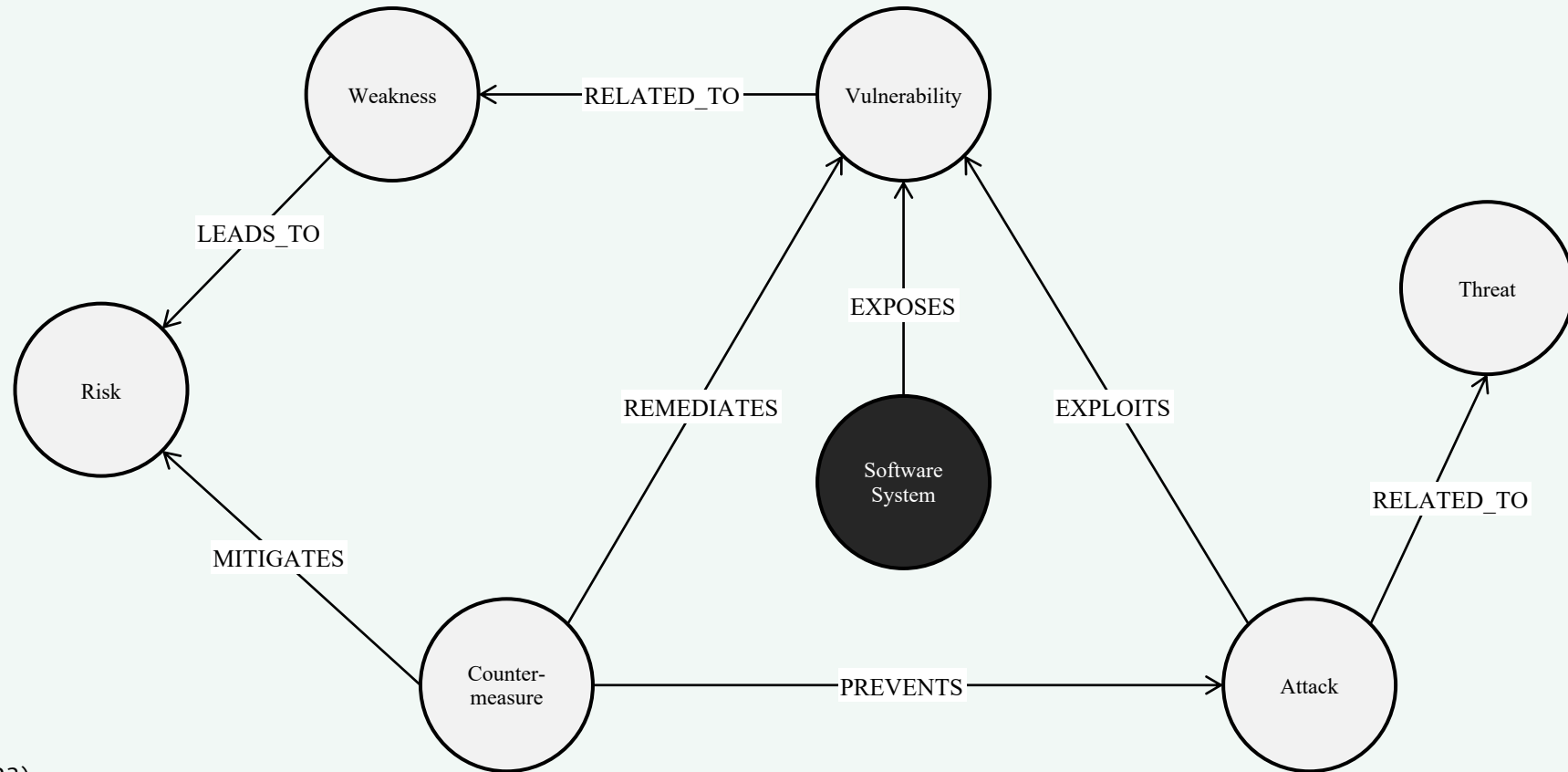
Consistency with the Metamodel



(Vergara-Vargas et al., ECSA'23)

# Knowledge Graph: Sarch-Knows

## Cybersecurity Perspective



(Vergara-Vargas et al., ECSA'23)

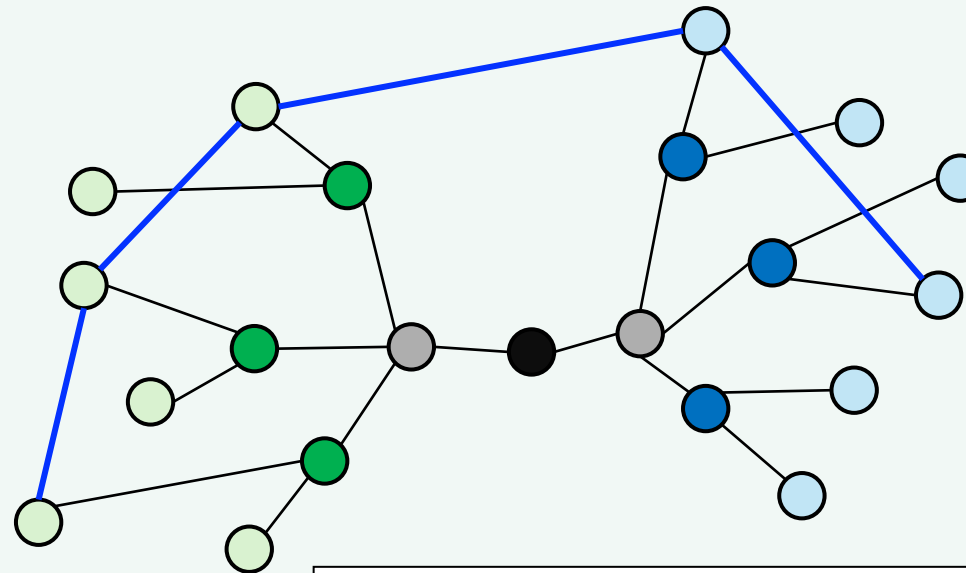
# Knowledge Graph: Sarch-Knows

Software Architecture

Cybersecurity

- Components
- Connectors
- Arch. Tactics
- Arch. Patterns
- ...

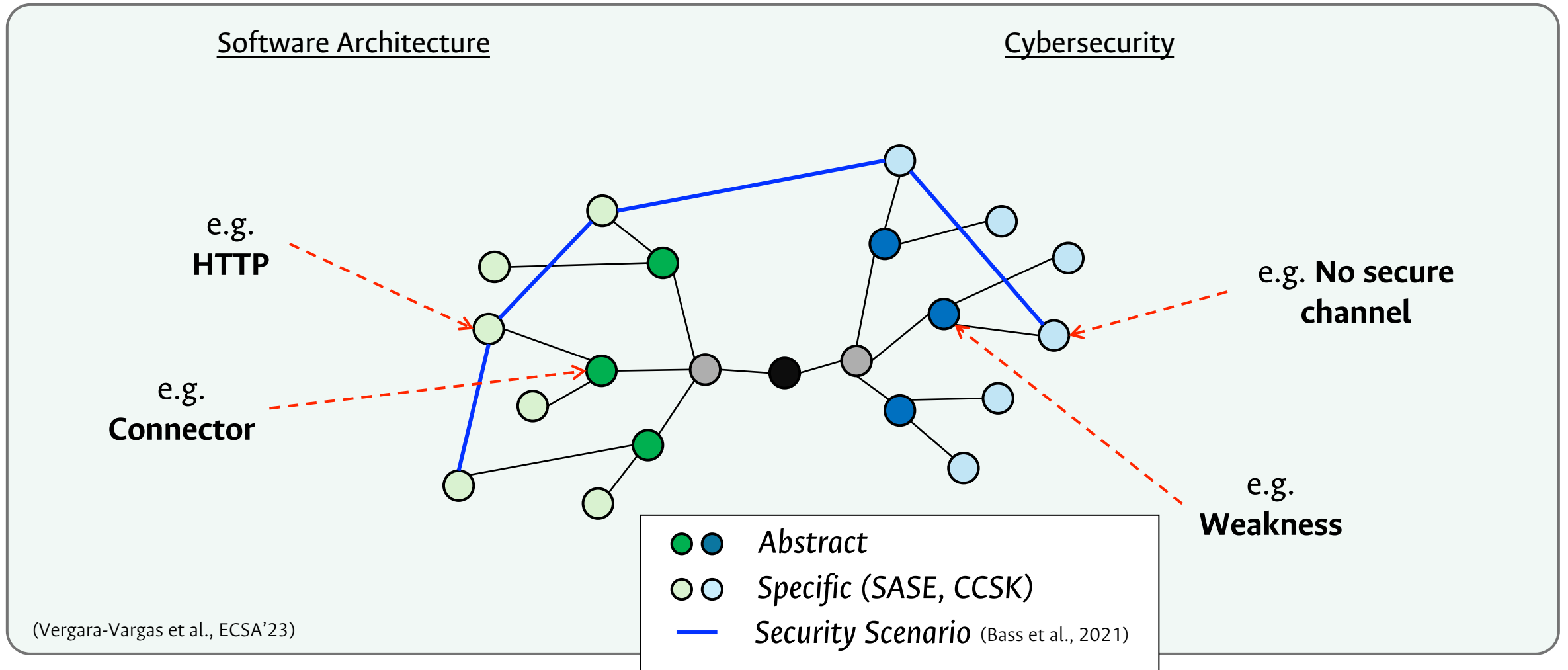
- Weaknesses
- Threats
- Risks
- Attacks
- Countermeasures
- ...



● ●	Abstract
○ ○	Specific (SASE, CCSK)
—	Security Scenario (Bass et al., 2021)

(Vergara-Vargas et al., ECSA'23)

# Knowledge Graph: Search-Knows



# Knowledge Graph: Sarch-Knows

Cypher queries

```
sw-arch-security$ match (n:concept {category: 'Core Element'}) return (n)
```

Result graph

Node properties

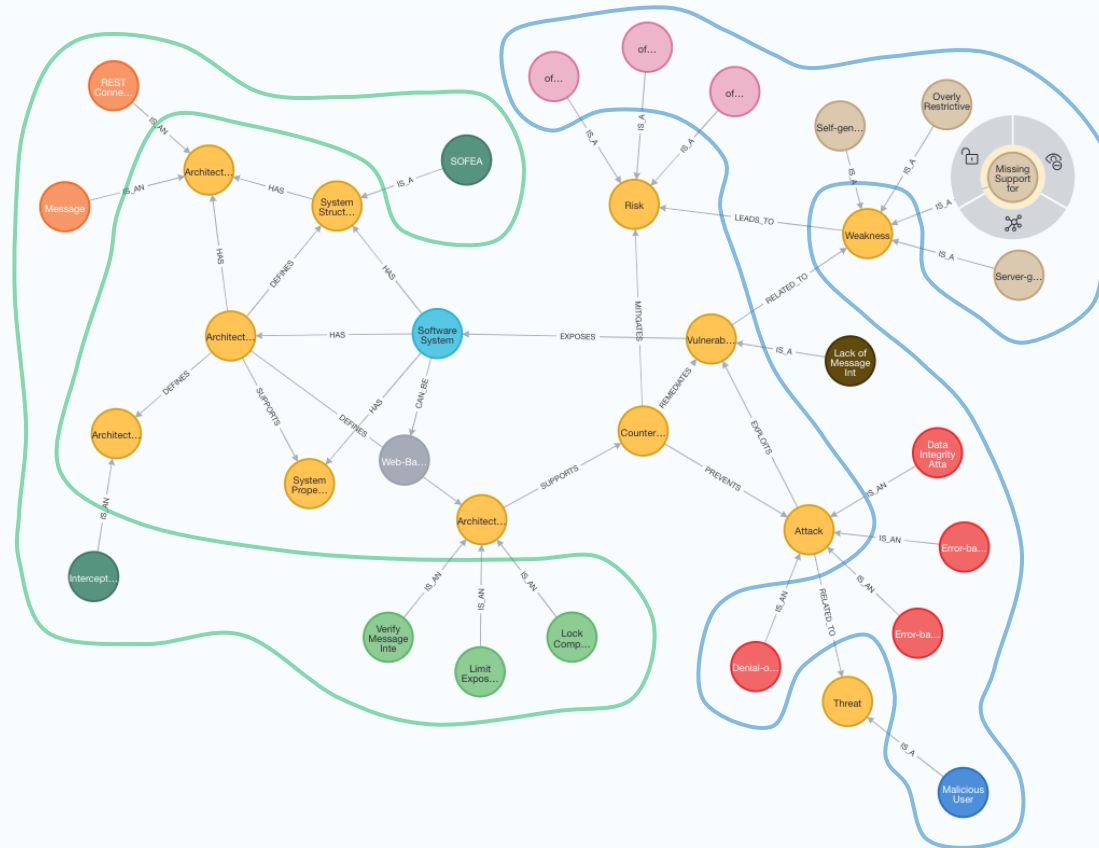
<b>concept</b>	
<b>&lt;id&gt;</b>	4
<b>category</b>	Core Element
<b>name</b>	Attack
<b>scenarios</b>	[1,2,3,4]

Element properties

A **core (abstract) element** corresponds to a **fundamental concept** of software architecture or cybersecurity.

# Knowledge Graph: Sarch-Knows

Software Architecture Specific Elements  
**(SASE)**

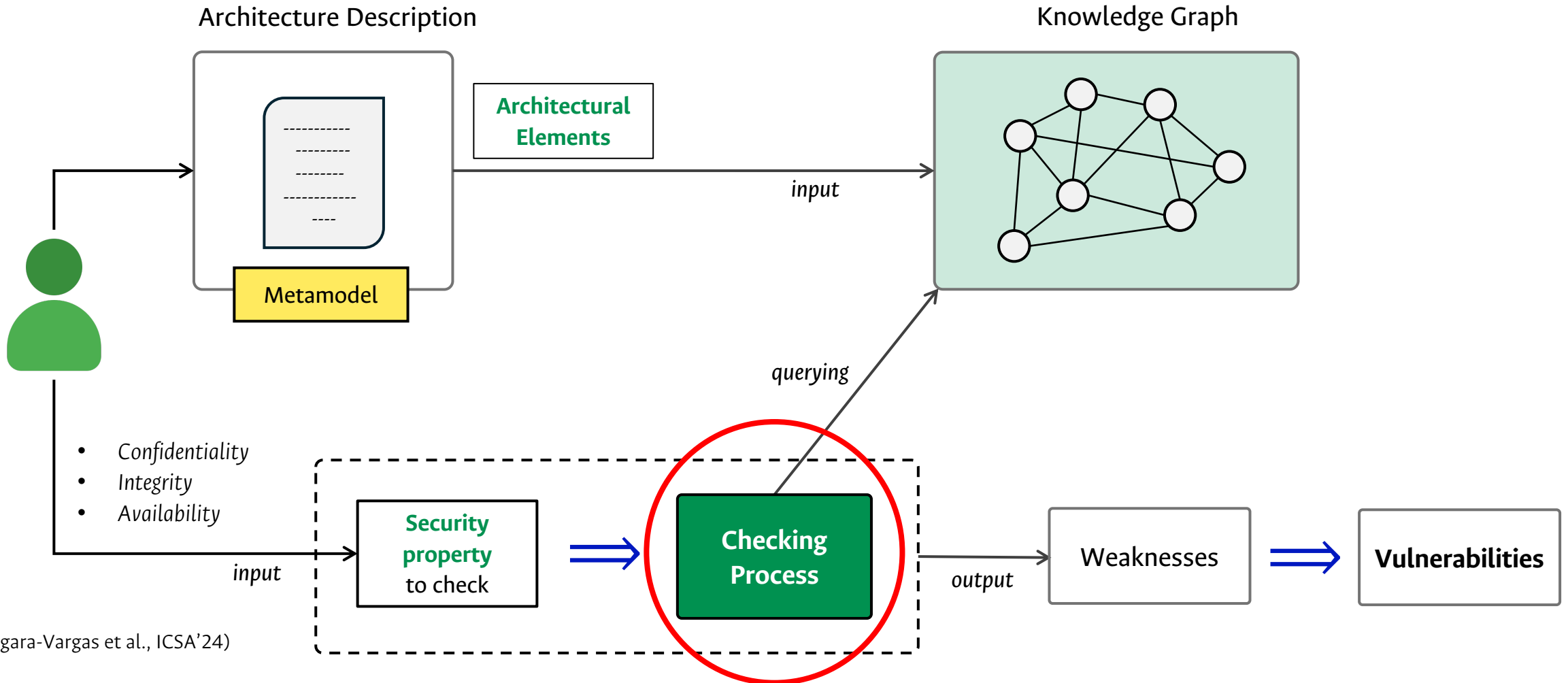


Current Common Security Knowledge  
**(CCSK)**

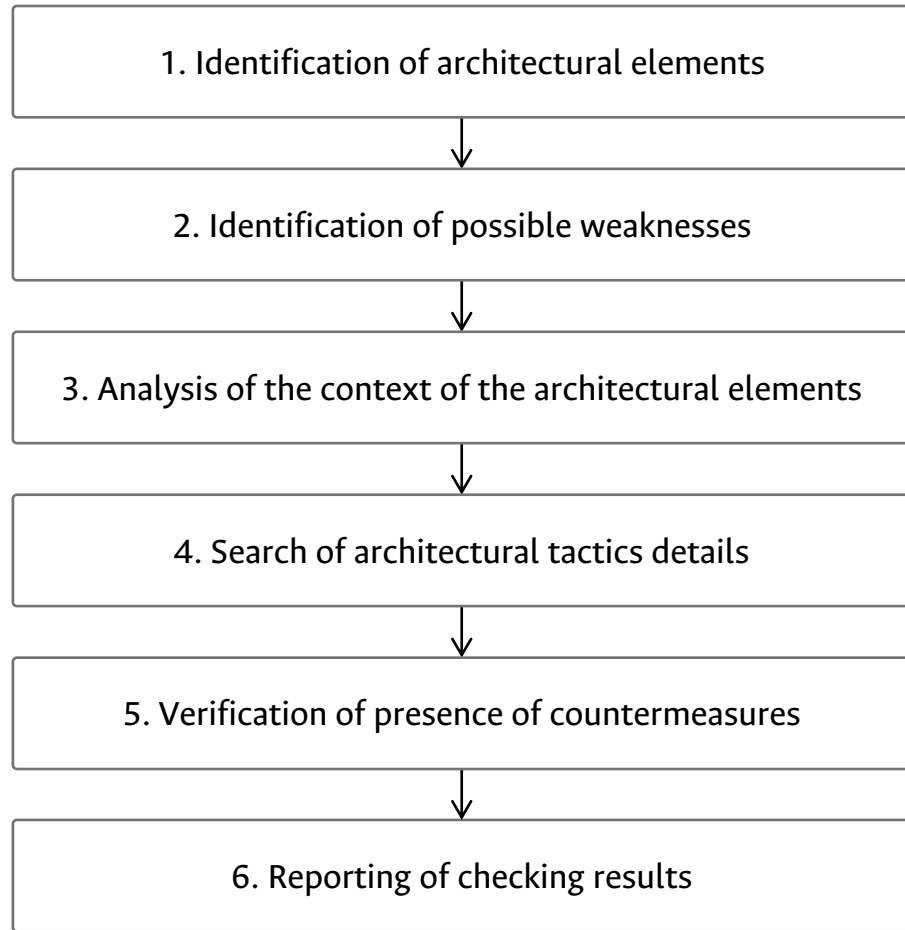
- (National Vulnerability Database **(NVD)** of **NIST**)
- (Common Weakness Enumeration **(CWE)** by **MITRE**)
- (Common Vulnerabilities and Exposures **(CVE)** by **MITRE**)
- (Architecture, Design and Threat Modeling **(ASVS)** by **OWASP**)

(Vergara-Vargas et al., ECSA'23)

# General Approach for Sarch-Checks



# Checking Process





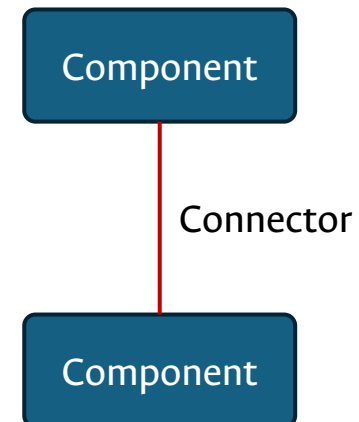
# Checking Process

## 1. Identification of architectural elements

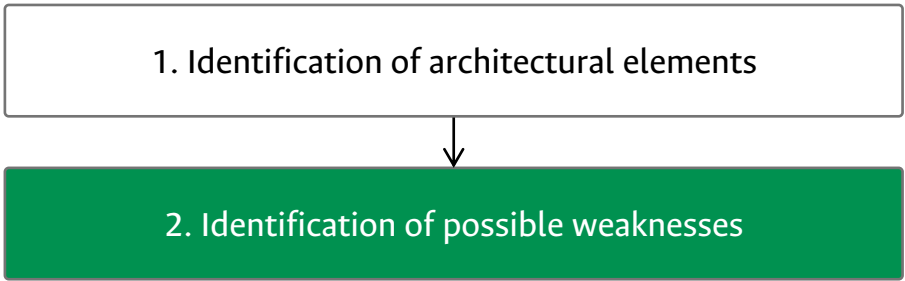
Elements to be analyzed:

- **Component-and-Connector View**

(Clements et al., 2010)



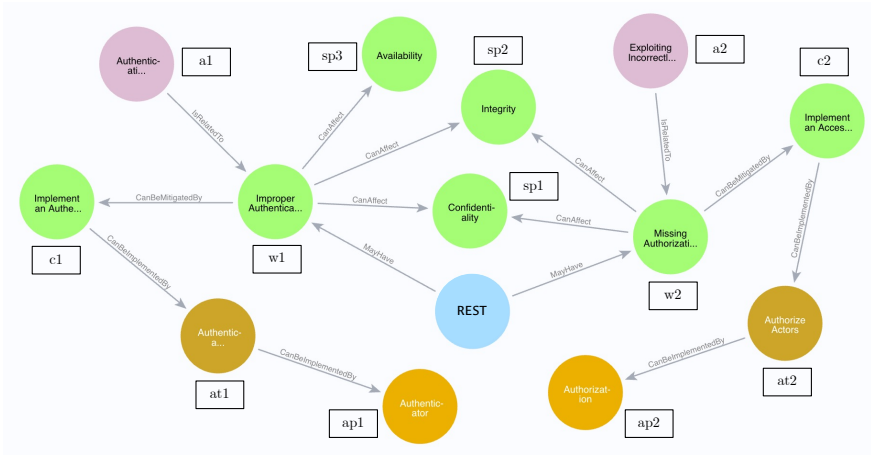
# Checking Process



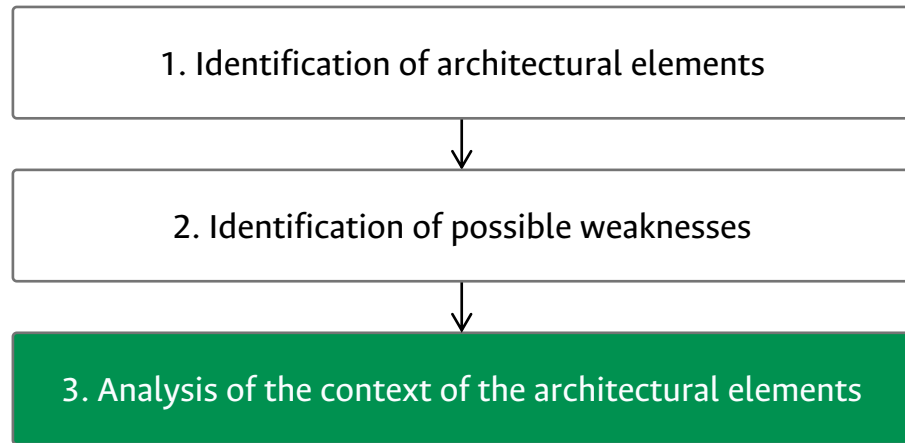
Search of possible **weaknesses** in the **knowledge graph**, using the Cypher Language of Neo4j:

```

    > MATCH subgraph = (a:specific {KEY: 'REST' }) - [:RELATIONSHIP*] - (b) RETURN subgraph;
  
```

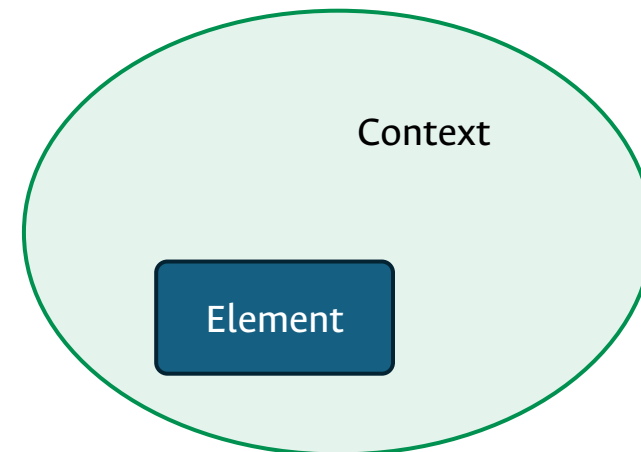


# Checking Process

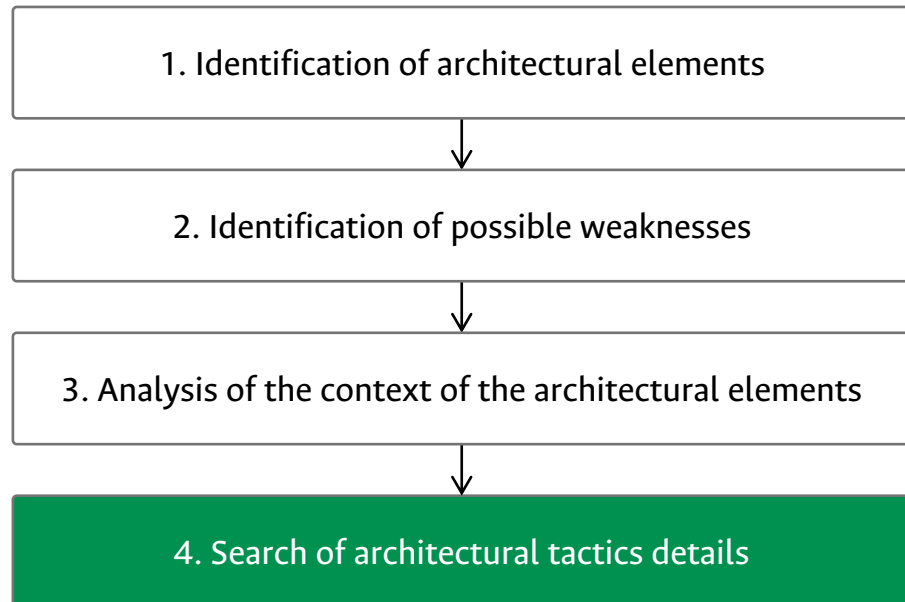


Identify the **conditions** in which the element is found in the architecture:

- Interactions with other elements.
- Interaction characteristics.
- Internal properties.



# Checking Process



Search of the **architectural tactics** details in the **knowledge graph**:

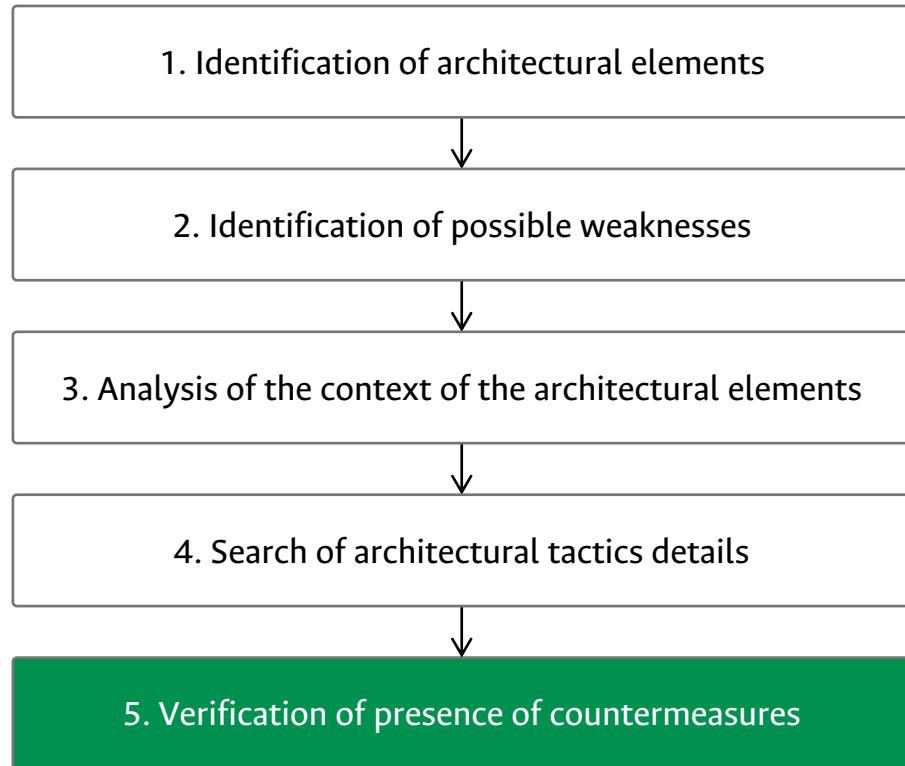
**Architectural Tactic** (Santos et al. ICSA'17)

*implementation*

**Architectural Pattern** (Zheng et al., SOSE'20)  
(for Security)

- **Countermeasure** -

# Checking Process



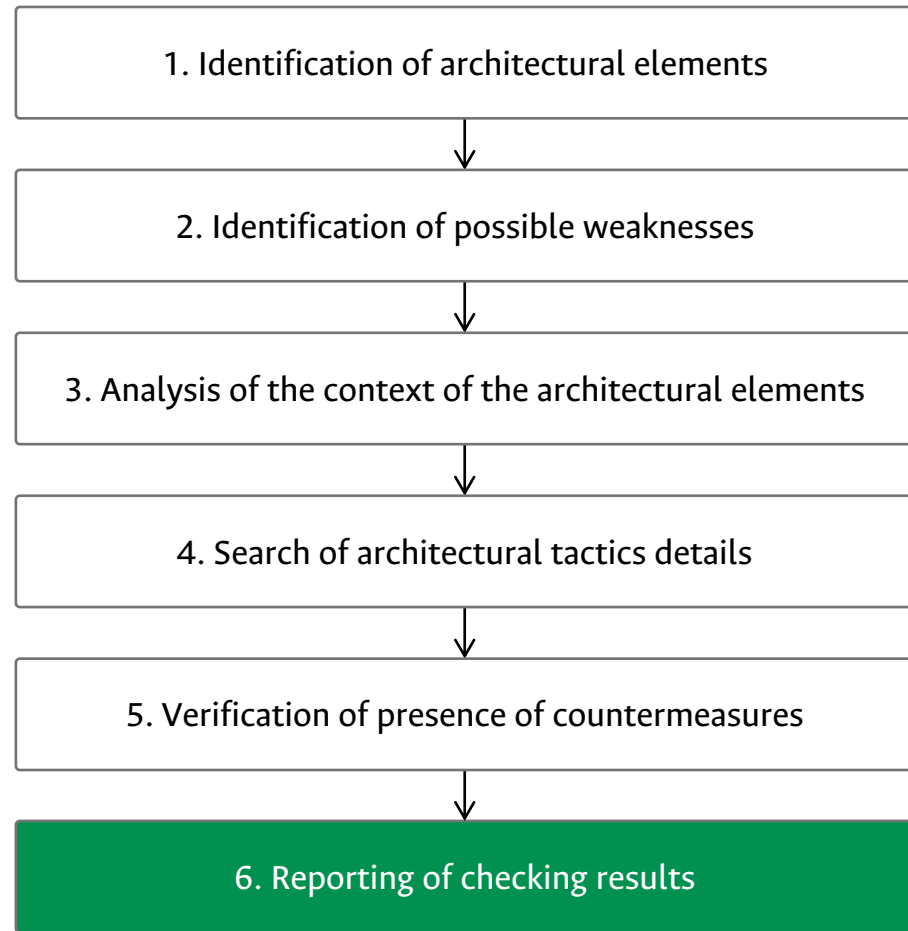
Execution of an **inspection** process:

- For each architectural element.
- Is there **evidence** of the presence of the related tactics?

**Subgraphs comparison:**

- Architectural element and its context.
- Architectural pattern implementation.

# Checking Process



Results:

- **Non-equivalence:** non-presence or bad implementation of the pattern. **Suggestion** of vulnerability.
- **Equivalence:** presence of pattern (tactic). **Suggestion** of property presence.

# Case Study

**A:** Design and Planning

**B:** Preparation and Collection of Data

**C:** Data Analysis

# Case Study

(Wohlin et al., 2012)

**A: Design and Planning**

**B: Preparation and Collection of Data**

**C: Data Analysis**

**1st Experiment**

**RQ:** Is it possible to check security properties from the architectural description of a software system?

1. Identification and selection of a **reported vulnerability**.
2. Reverse engineering the software system to **abstract** the **architecture**.
3. Description of the software system **architecture**.
4. Execution of the **checking** process.

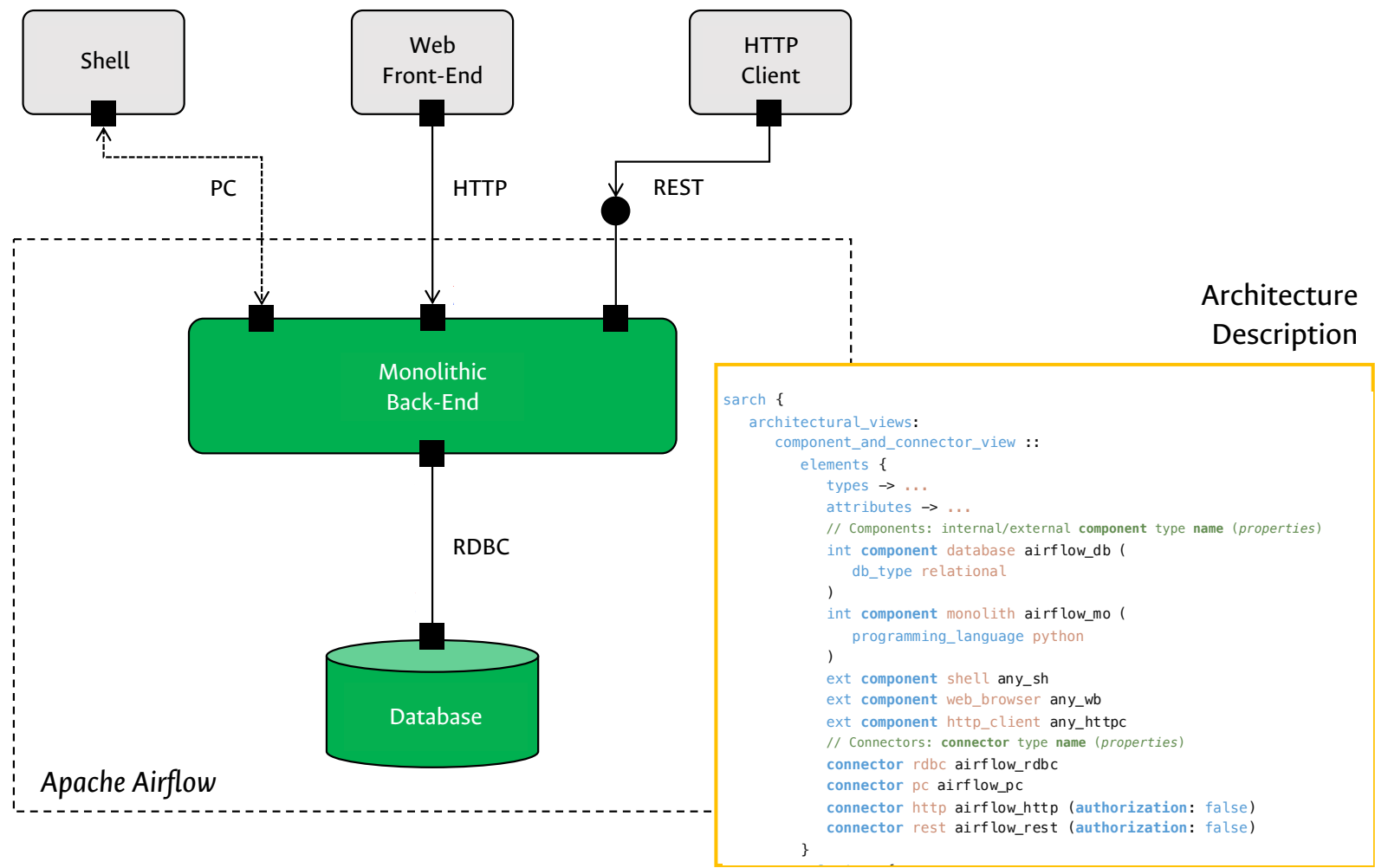


# Case Study

(Wohlin et al., 2012)

- A: Design and Planning
- B: Preparation and Collection of Data
- C: Data Analysis

- **Software System:**
  - Apache Airflow
- **Reported Vulnerability:**
  - CVE-2020-13927
- **CWE-287:**
  - Improper Authentication




# Case Study


(Wohlin et al., 2012)

**A: Design and Planning**

**B: Preparation and Collection of Data**

**C: Data Analysis**

 CVE vulnerability found by Sarch-Checks.

 Additional vulnerabilities found by Sarch-Checks.

Architectural Element	Architectural Element Type	Weakness	Security Properties	Architectural Tactic	Architectural Pattern	Property presence (suggestion)
HTTP	Connector	CWE-287: Improper Authentication	Confidentiality Integrity Availability	Authenticate Actors	Authenticator	Yes
		CWE-862: Missing Authorization	Confidentiality Integrity	Authorize Actors	Authorization	Yes
		CWE-353: Missing Support for Integrity Check	Integrity	Verify Message Integrity	Transport Layer Security	Yes
		CWE-354: Improper Validation of Integrity Check Value	Integrity	Verify Message Integrity	Transport Layer Security	Yes
REST		CWE-287: Improper Authentication	Confidentiality Integrity Availability	Authenticate Actors	Authenticator	No
		CWE-862: Missing Authorization	Confidentiality Integrity	Authorize Actors	Authorization	No
		CWE-353: Missing Support for Integrity Check	Integrity	Verify Message Integrity	Transport Layer Security	Yes
		CWE-354: Improper Validation of Integrity Check Value	Integrity	Verify Message Integrity	Transport Layer Security	Yes
Monolith	Component	CWE-250: Execution with Unnecessary Privileges	Confidentiality Integrity Availability	Limit Access	Secure Three-Tier Architecture	Yes
Database		CWE-250: Execution with Unnecessary Privileges	Confidentiality Integrity Availability	Limit Access	Secure Three-Tier Architecture	No

## Conclusions and Current Work

- The main contribution of this work is the use of an **agile representation** of the **security knowledge** to adapt to its continuous evolution.
- The process is based on a **validated knowledge** on security at the architectural level, accessible to everyone for verification.

### Future Work

- Collect a large number of software architectures for validation.
- Refine elements and data in the knowledge graph.