Research Directions in Secure Software Engineering

Riccardo Scandariato

Katholieke Universiteit Leuven

Belgium

System of patterns

Instrument Suite

- Pattern template for homogenizing description
- Labels (~quality attributes) for each pattern
  - Allow making trade-offs
- Classifications of the patterns
  - Place in design process
  - According to security objective
- Relations between patterns

Instrument Suite

Pattern template

Pattern Name

Intent
Also known as (optional)
Applicability
Security objective
Labels
Relationships
- *Dependencies*
- *Impairments*
- *Conflicts*
- *Benefits*
- *Alternatives*

1. Problem
   - *Forces*
2. Example
3. Solution
   - *Structure*
   - *Dynamics*
   - *Participants*
   - *Collaborations*
4. Implementation (optional)
5. Pitfalls (optional)
6. Consequences
7. Related patterns
8. Known uses

Instrument Suite

Labels

ISO 9126

- Dependability
- Portability
- Maintainability
- Performance
- Usability

CC

- Manageability
- Auditability

Security Objectives

- Confidentiality
- Integrity
- Accountability
- Anonymity
- Privacy
- Non-repudiation

- Cost

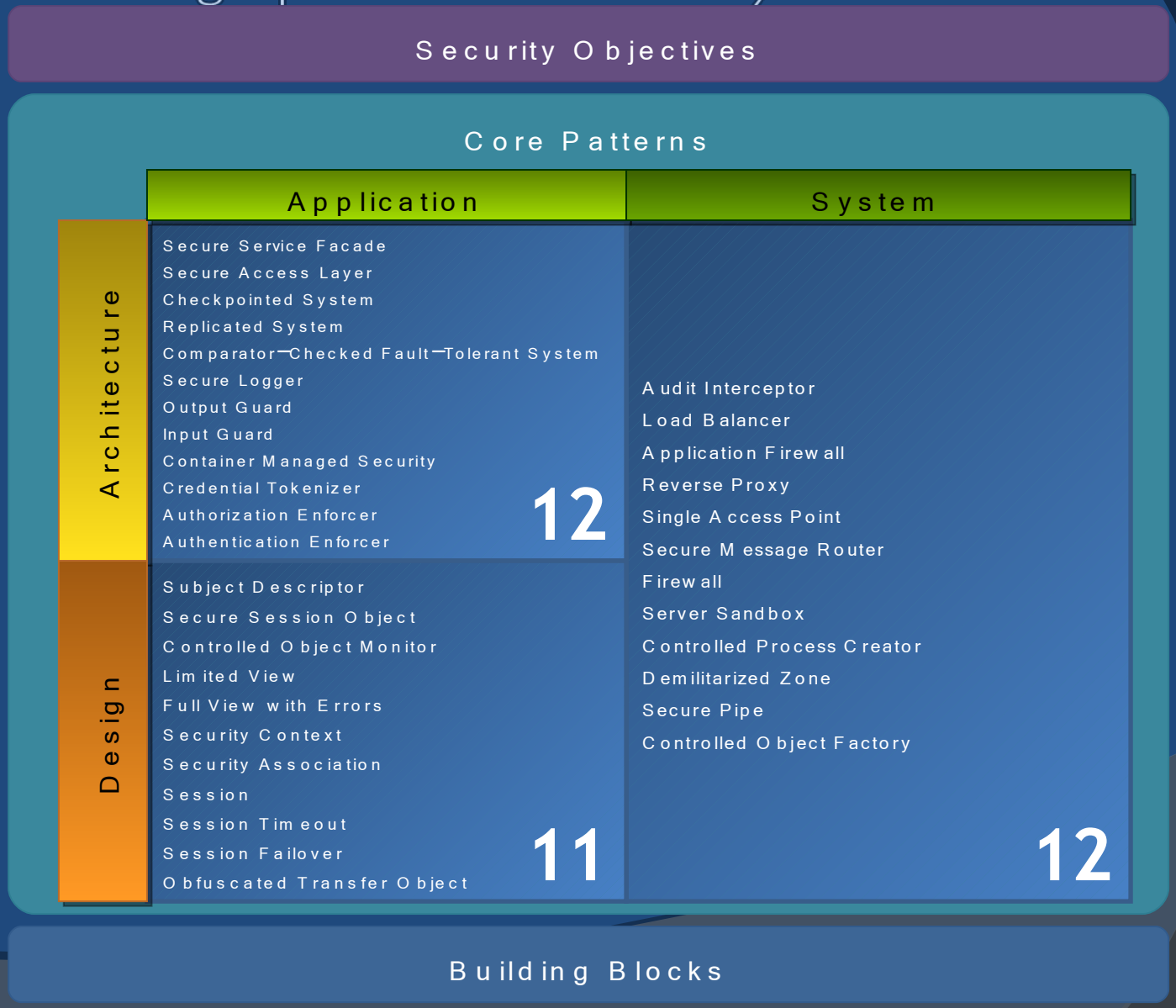
Instrument Suite

Labels - Examples

- Secure Logger
  - + Manageable (Centralized)
  - – Performance
- Authentication Enforcer
  - + Maintainability
  - + Auditability
  - – Dependability

Instrument Suite

Place in design process: *Taxonomy*



Instrument Suite

Place in design process: *Taxonomy*

Security Objectives

Information Obscurity

Authorization

Known Partners

Controlled Execution Environment

Secure Communication

Core Patterns

Logger

Network Authentication Protocol

Encrypted Storage

Minefield

Error Detection and Correction

Controlled Virtual Address Space

Password Synchronizer

File Authorization

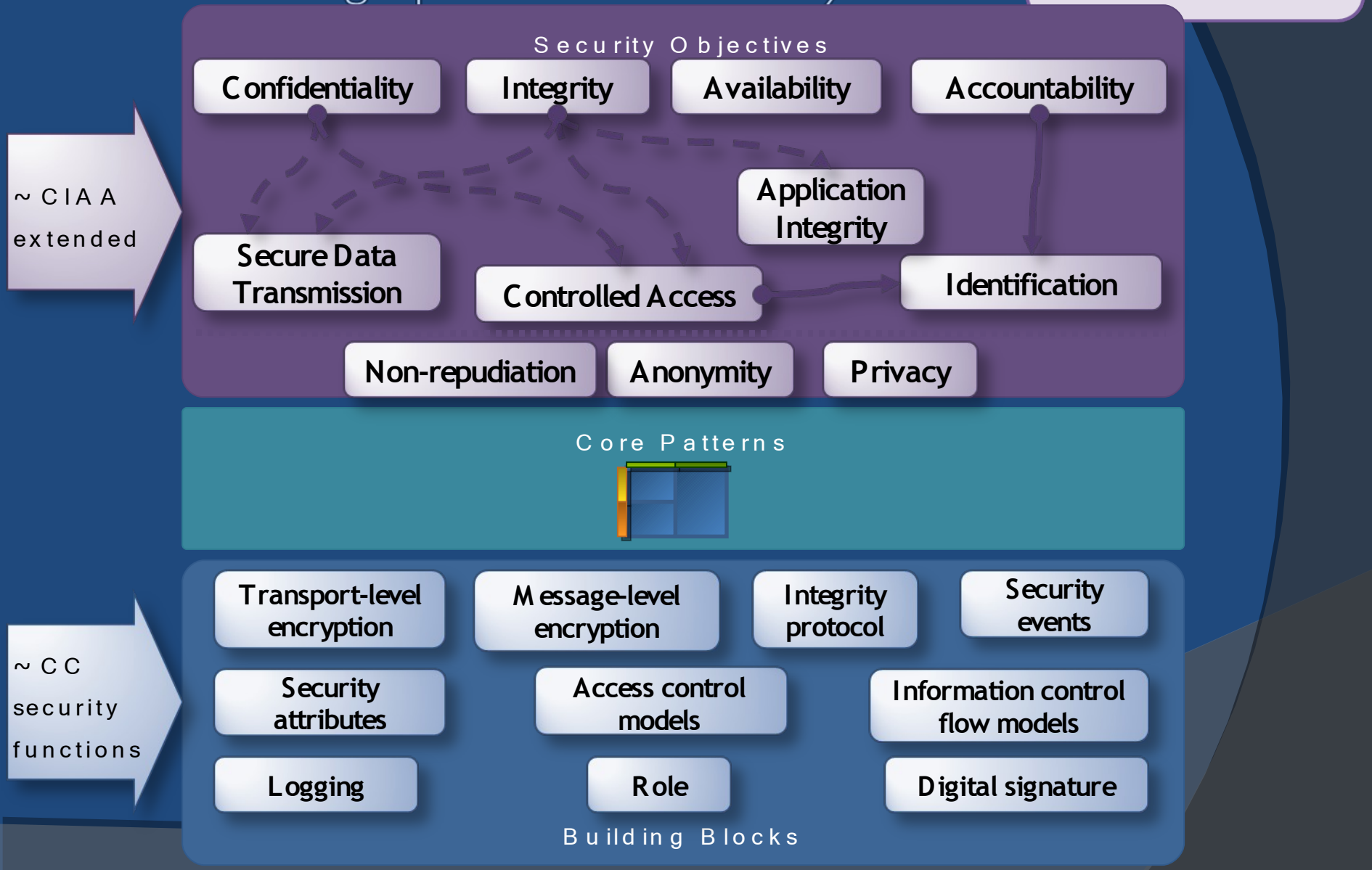
Roles

Multilevel Security

Building Blocks

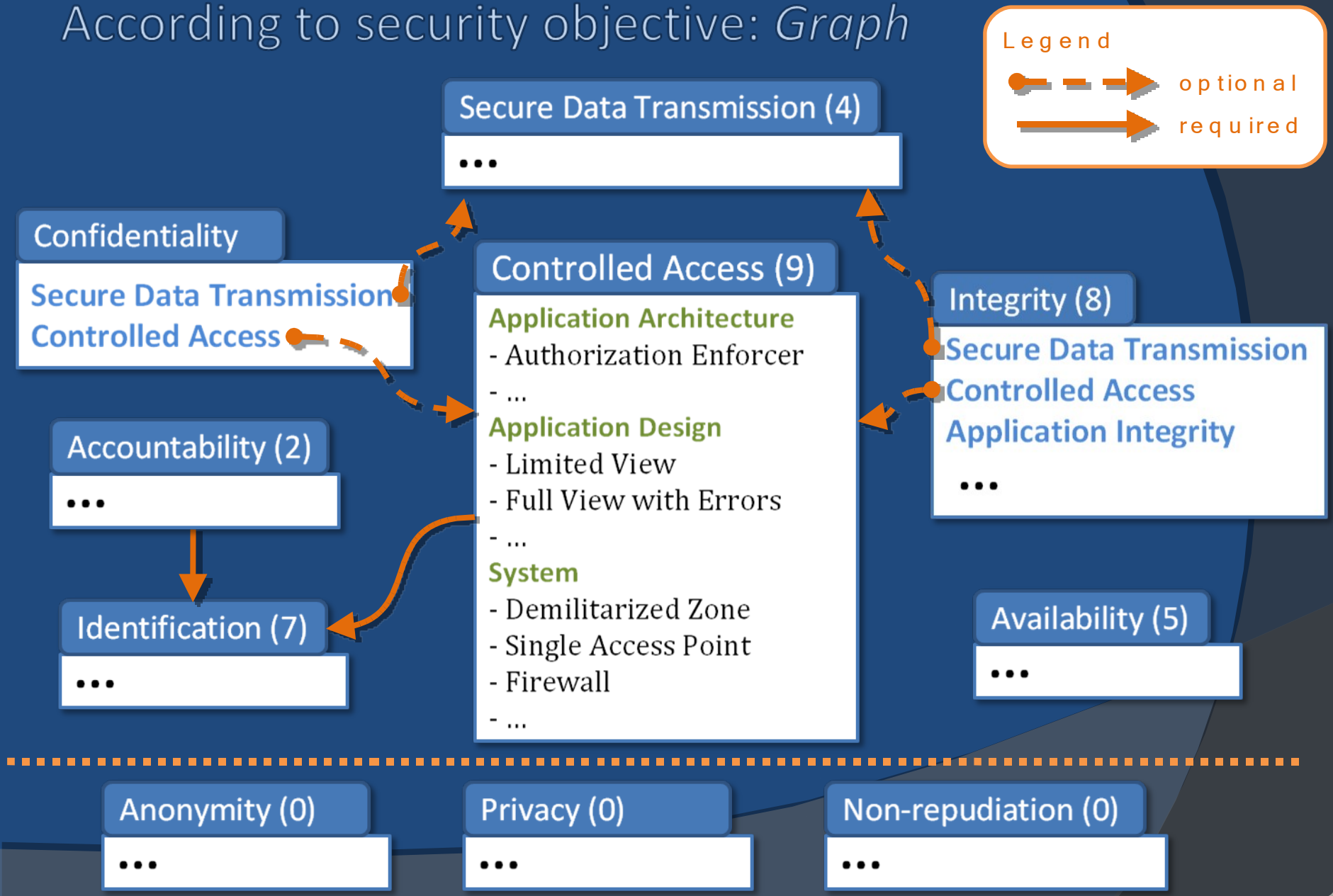
Instrument Suite

Place in design process: *Taxonomy*



Instrument Suite

According to security objective: *Graph*



Instrument Suite

Relations between patterns: *Table*

D: Depends on
B: Benefits from
I: Impairs
C: Conflicts with
A: Alternative

	System	Firewall	Single Access Point	*Appl. Architecture*	Authent. Enforcer	Authoriz. Enforcer	Secure Logger	*Applic. Design*	Security Association	Limited View	Full View w/ Errors	Session
System												
Demilitarized Zone		D										
Secure Pipe									B			
Load Balancer												I
Audit Interceptor							D					
Application Architecture												
Authentication Enforcer			B			B						
Authorization Enforcer			B		B							
Application Design												
Limited View											A,C	
Full View with Errors										A,C		

Methodology

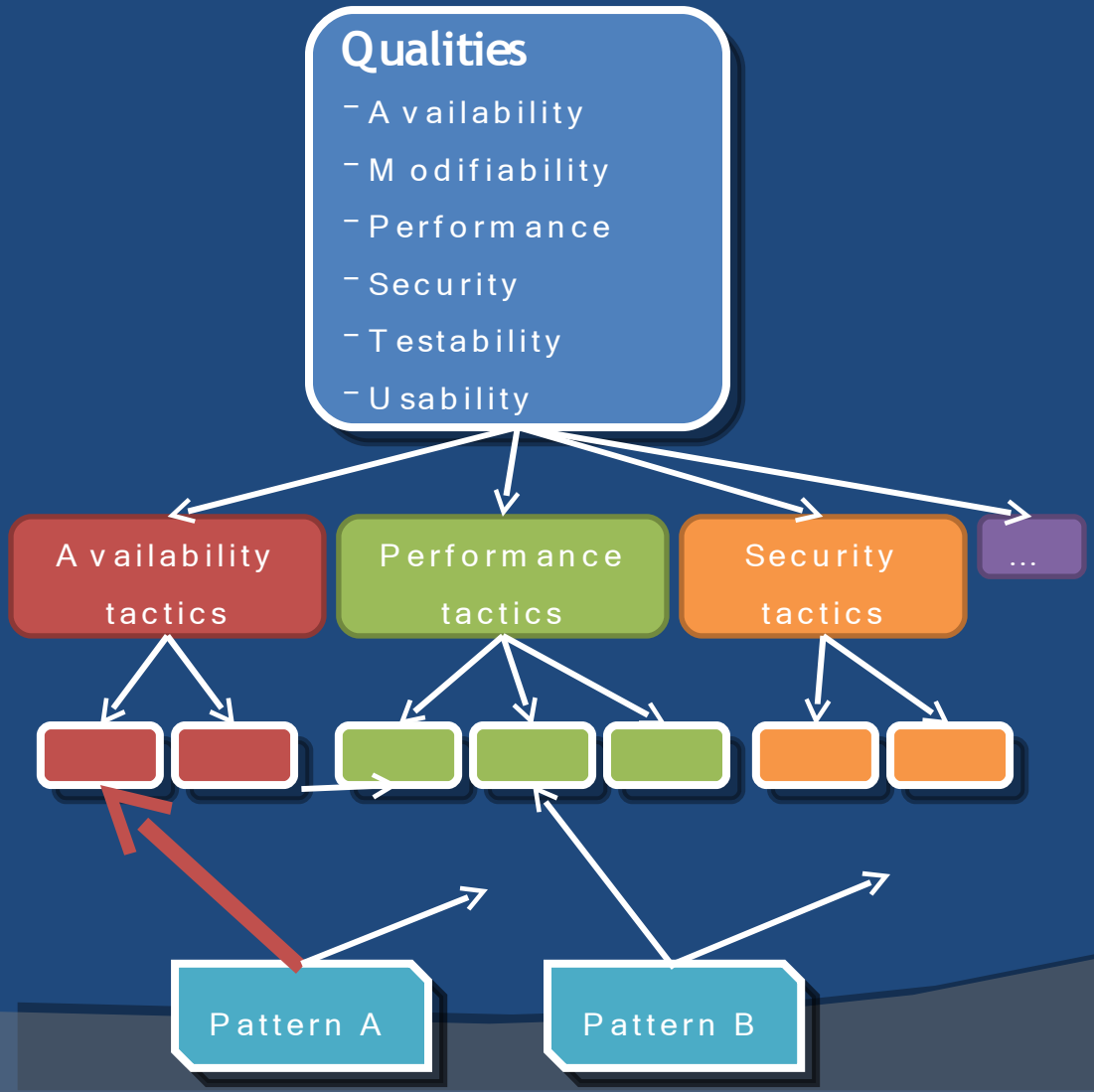
Methodology

- Step-by-step refinement
- Based on Attribute-driven design[1]
  - Use quality attributes to decompose the system
- But security as the main focus
  - Security *is* yet another quality
  - However, it needs end-to-end approach

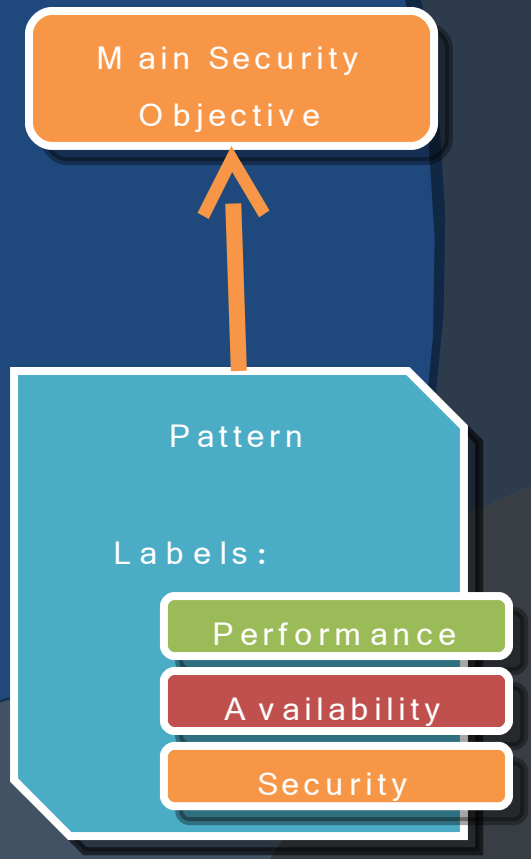
[1] Software Architecture in Practice, L. Bass, P. Clements, R. Kazman

Qualities, objectives, patterns, tactics and labels

- Bass, Clements, Kazman

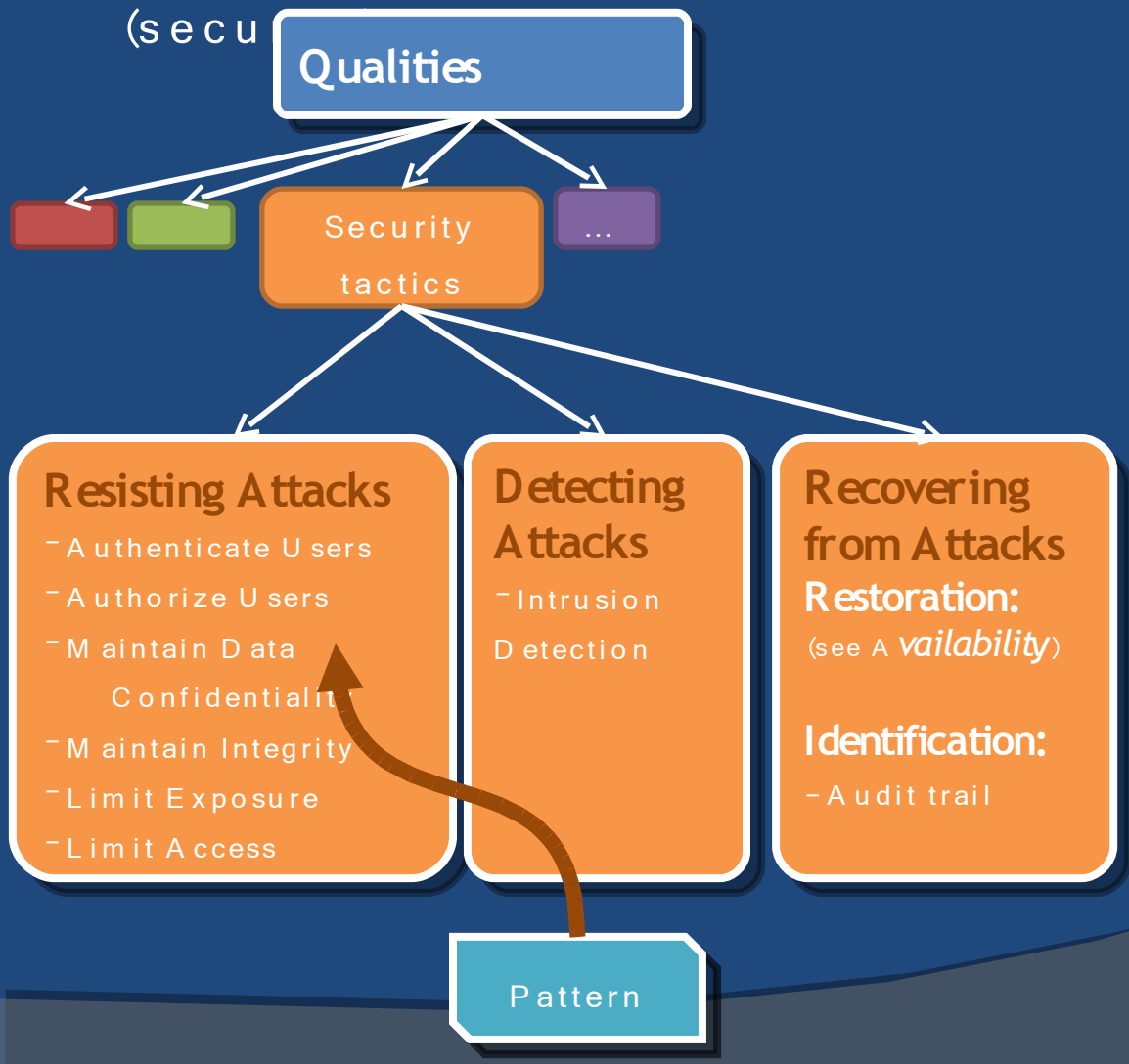


- Our approach

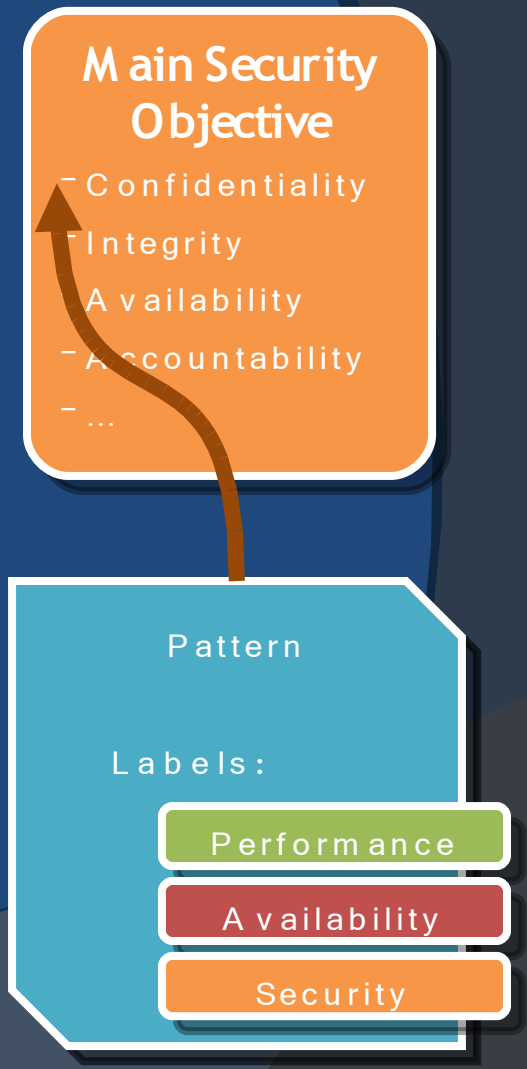


Qualities, objectives, patterns, tactics and labels

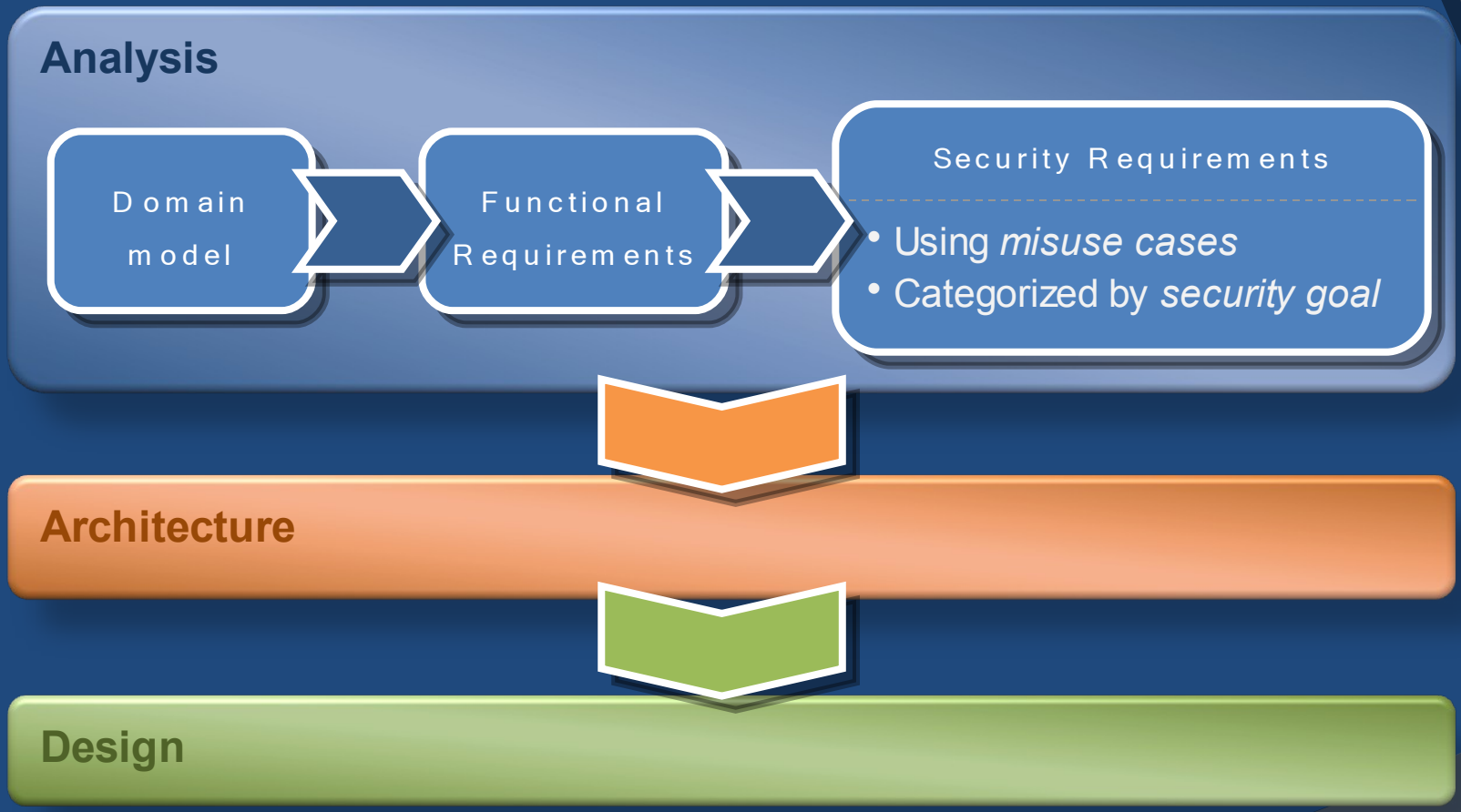
- Bass, Clements, Kazman (secu



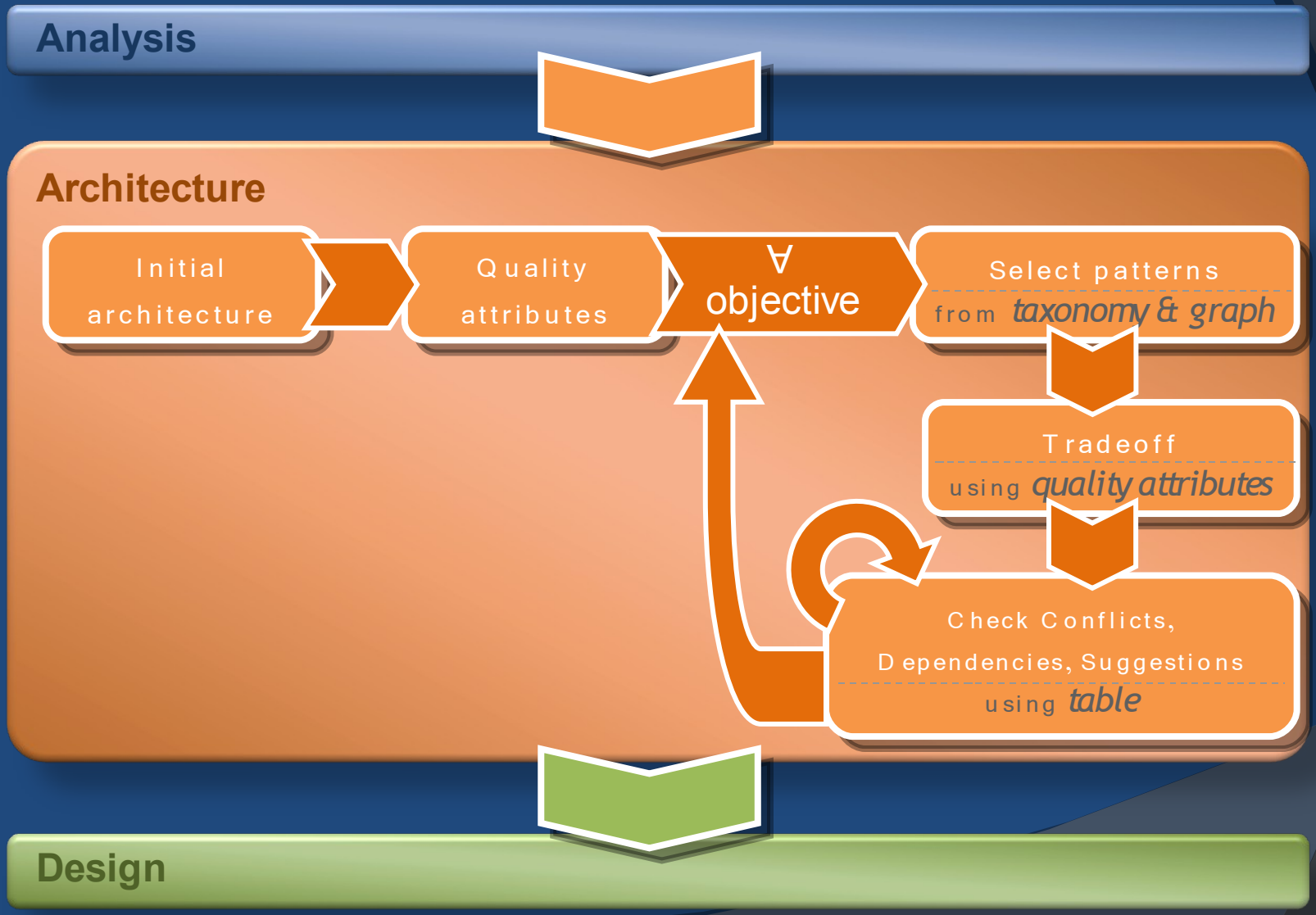
- Our approach



Methodology



Methodology



Methodology

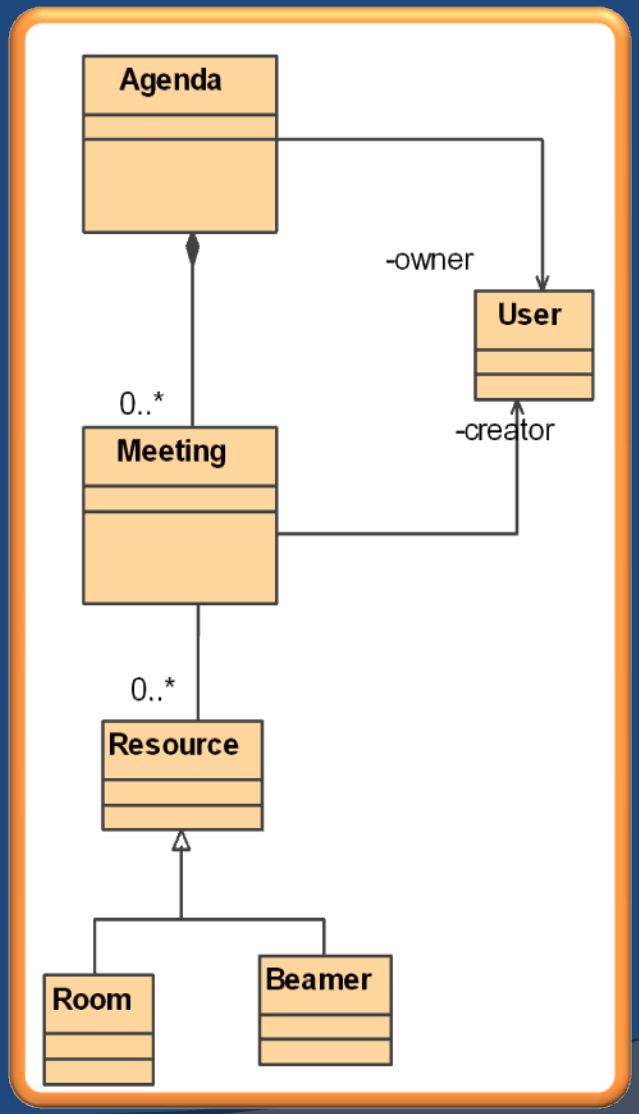
In practice…

Example: Calendar Application

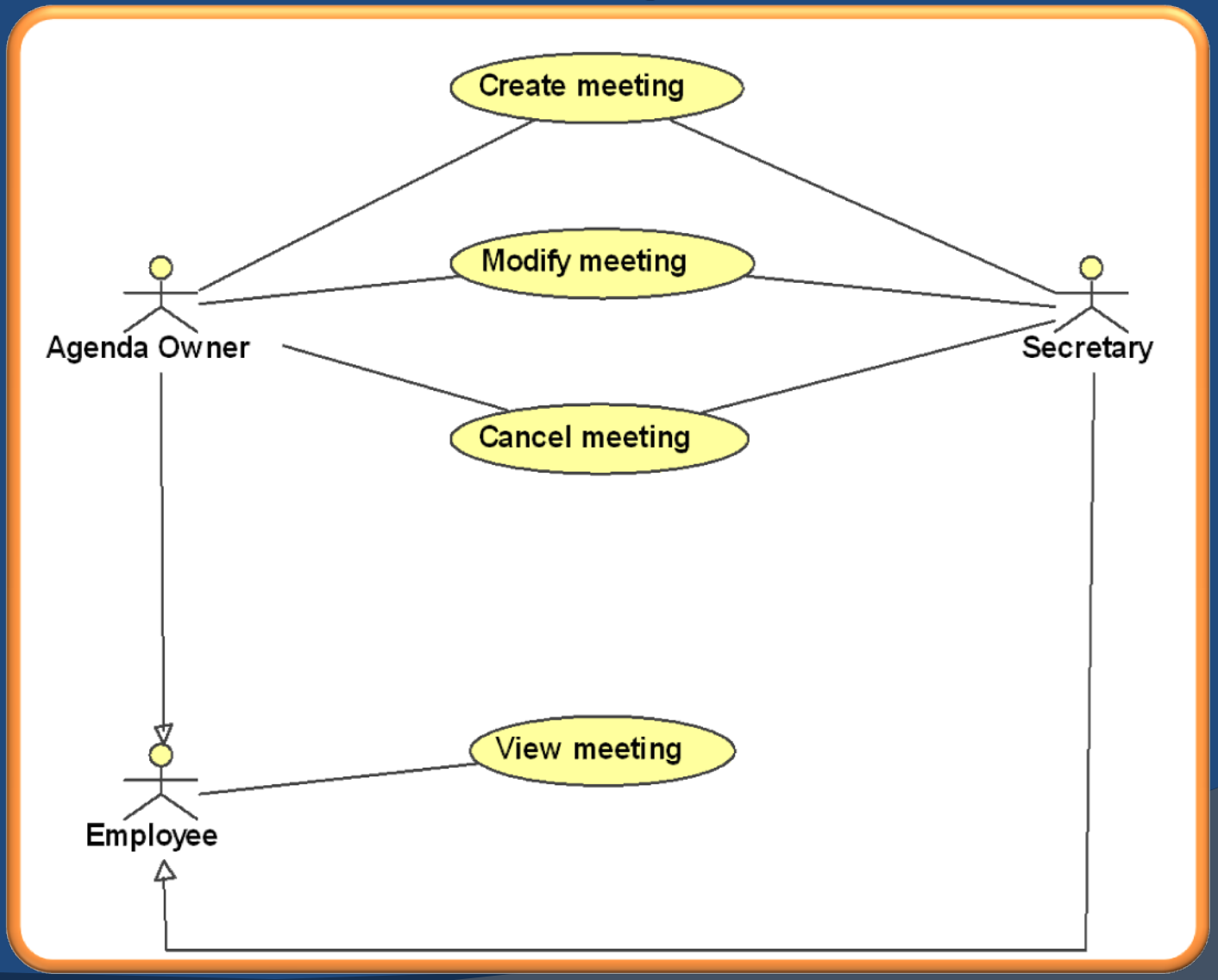
- Create a web-based variant of the well-known calendaring application using the proposed methodology

- Functionality:
  - Users having their own calendar
  - Entries can be added, modified and cancelled
  - Everyone can view entries of anyone
  - Secretaries can perform operations on behalf of users

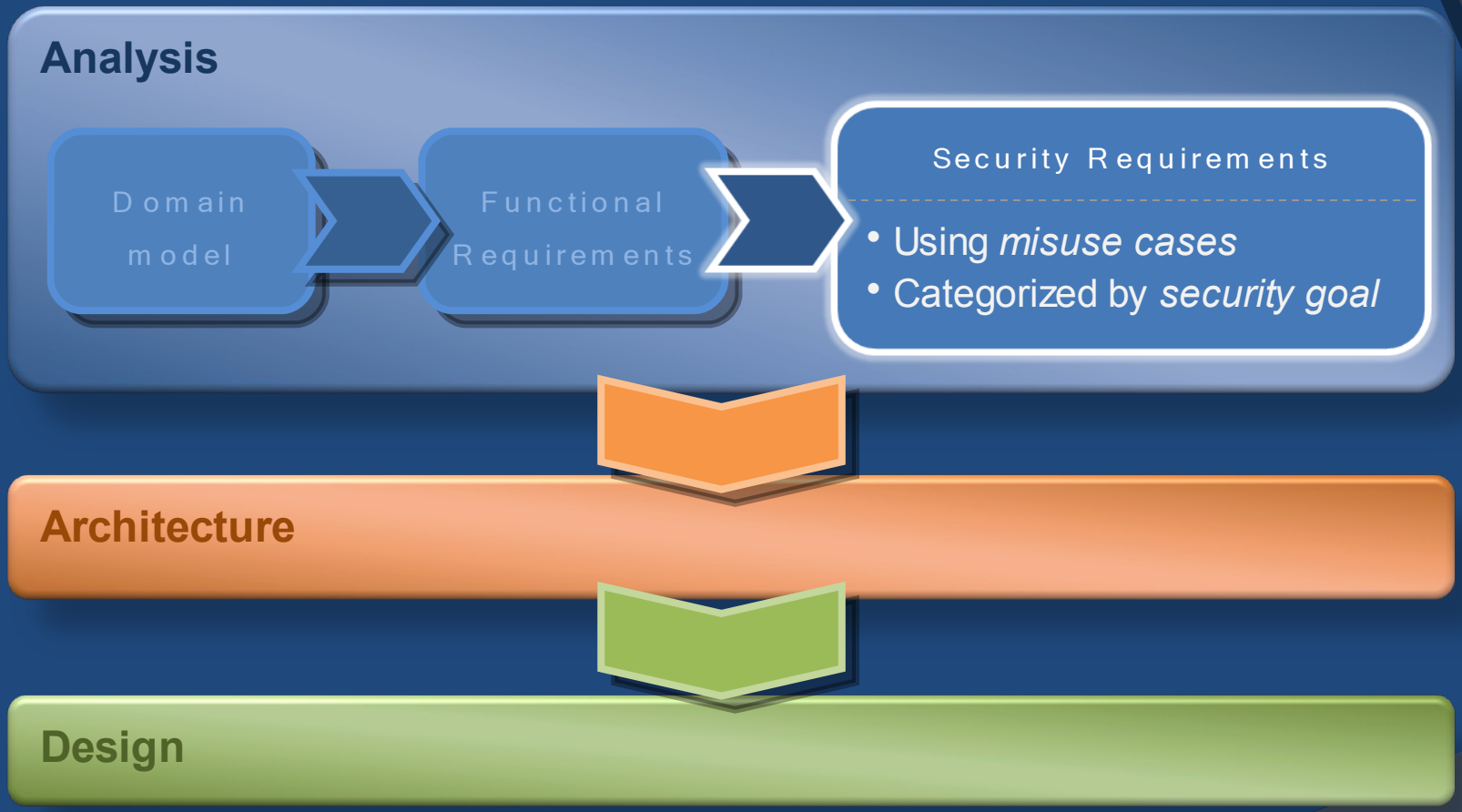
1. Domain Model/Vocabulary



2. Functional Requirements

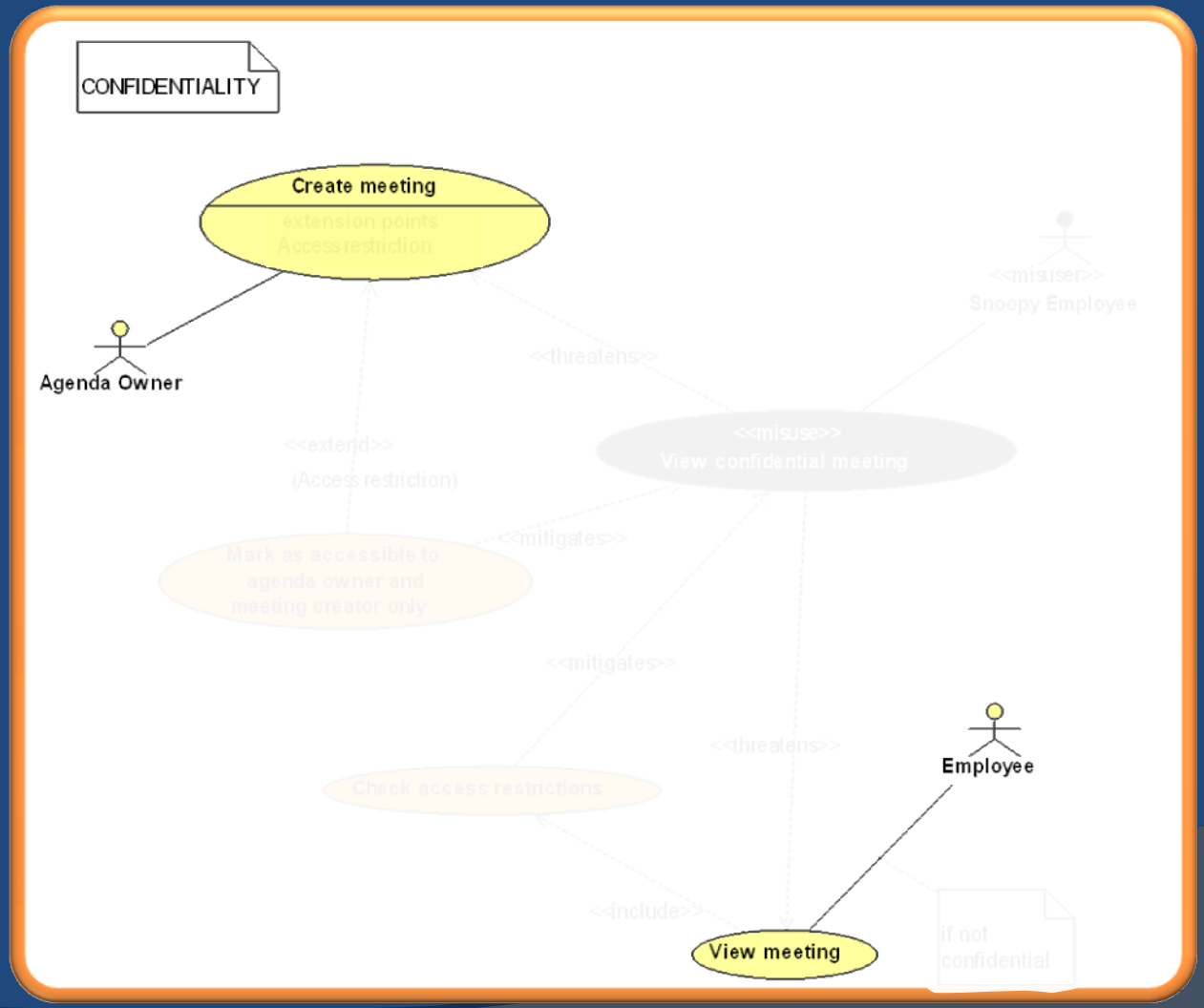


Methodology



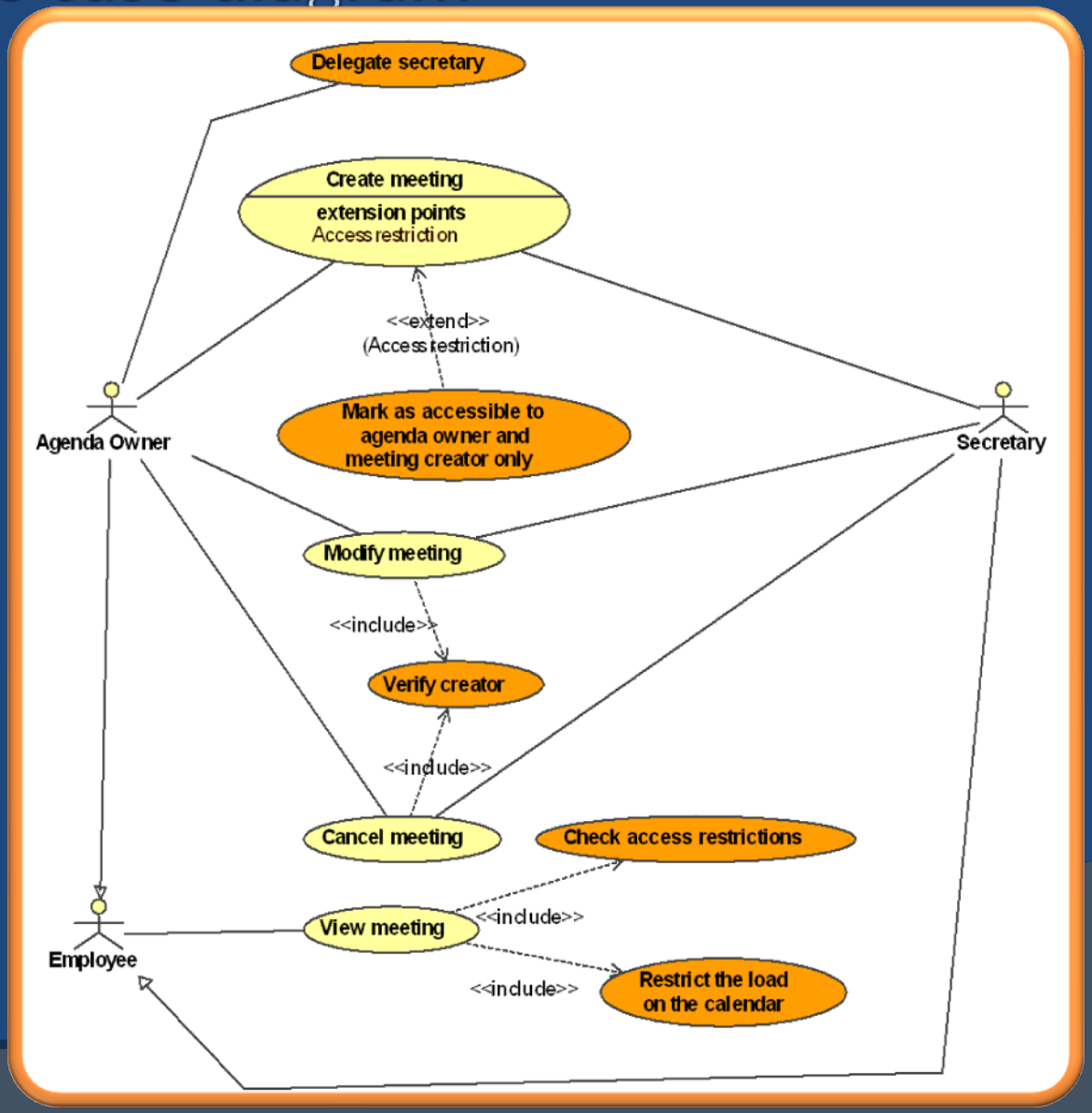
3. Security Requirements

Using misuse cases: Confidentiality

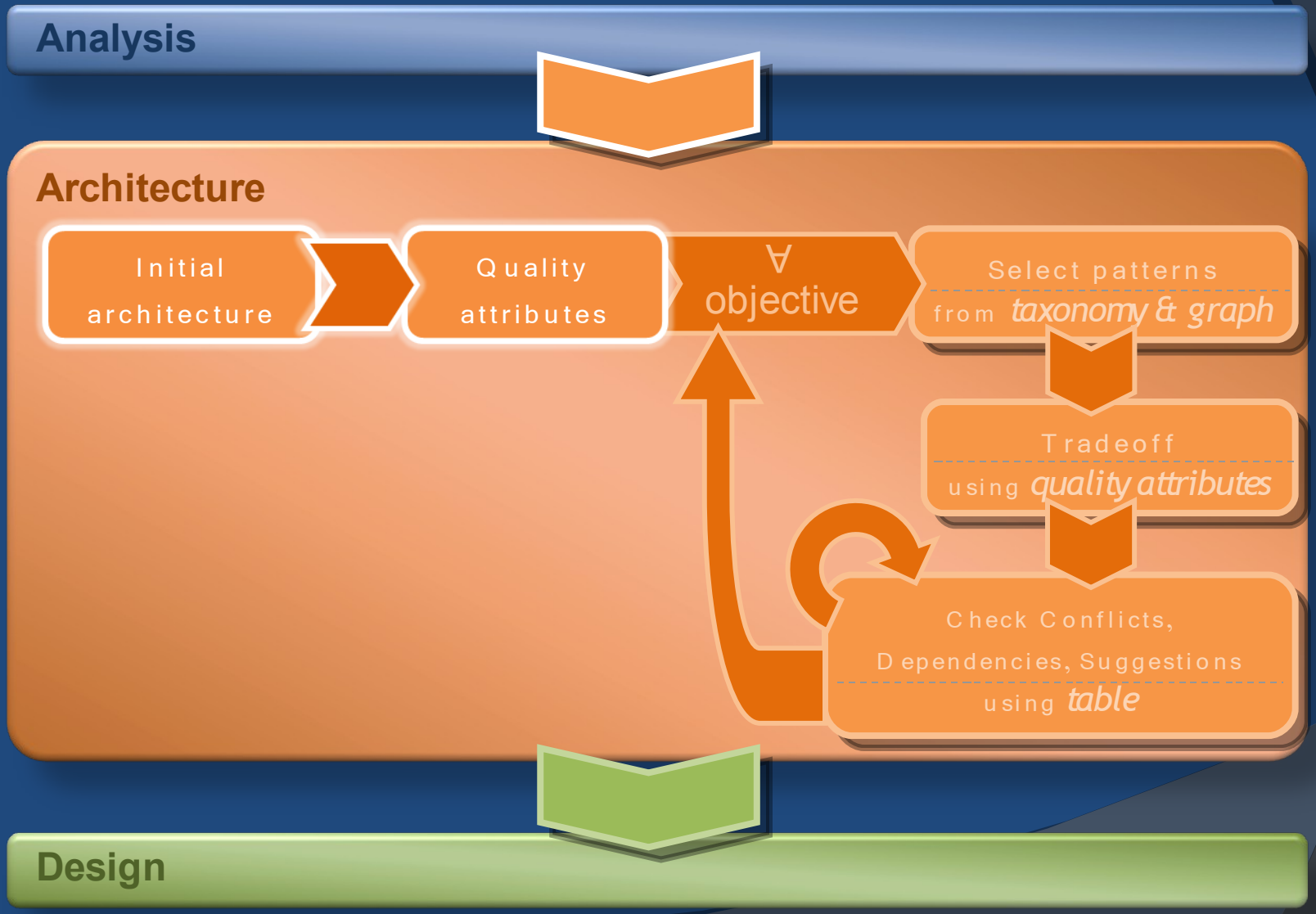


3. Security Requirements

Final use case diagram



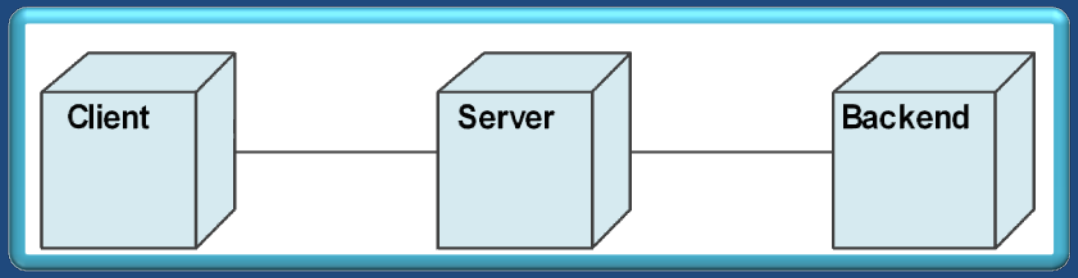
Methodology



4. Initial architecture

Web-based calendar

➔ standard 3-tier web architecture (naive)

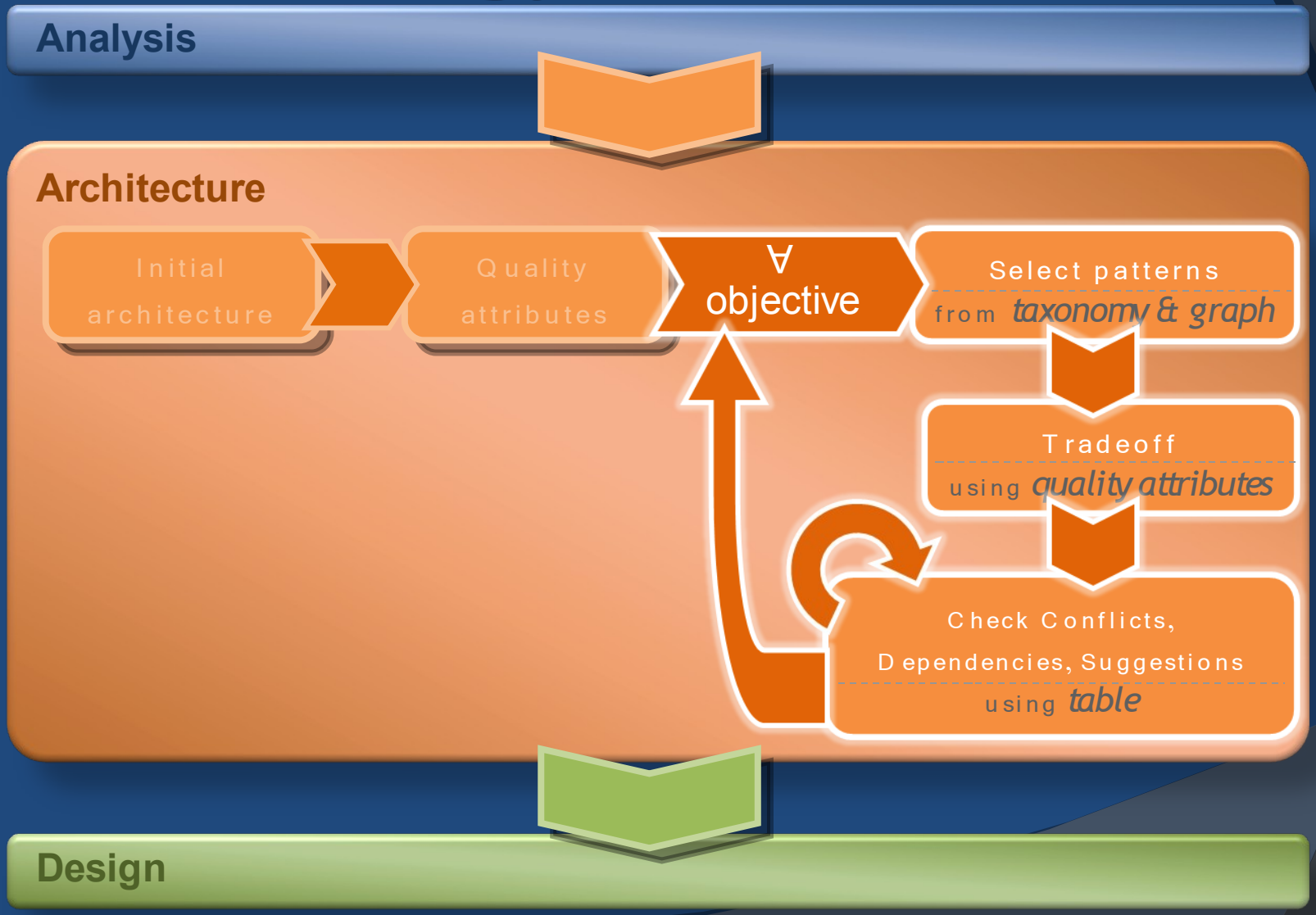


Note: for this example, we use the deployment view only

5. Prioritized quality attributes

1. Availability
   - ✓ Graceful degradation where calendars of important people fail later
2. Usability
   - ✓ Integration with existing Kerberos
3. Modifiability
   - ✓ Change the policy
   - ✓ Change the authentication method

Methodology



6. Iteration: Confidentiality (1)

1. Look at the use cases for confidentiality
   - ✓ Functional:
     - ✓ Create meeting
     - ✓ View meeting
   - ✓ Security:
     - ✓ Mark as accessible to agenda owner and meeting creator only (on "Create meeting")
     - ✓ Check access restrictions (on "View meeting")
2. Select candidate patterns (using graph)
   - ✓ Confidentiality = *Controlled Access* + *Identification*

6. Iteration: Confidentiality (2)

1. Select architectural patterns for *controlled access*

- ✓ Pick Reference monitor (modifiability)

- ✓ Benefits from Checkpoint

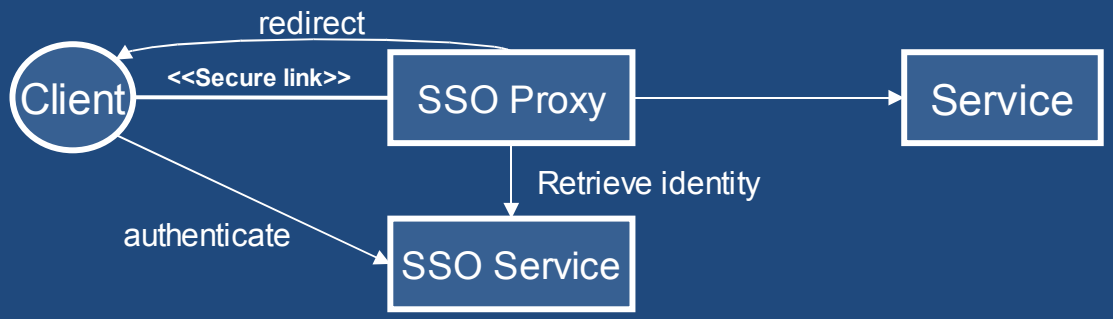
6. Iteration: Confidentiality (3)

1. Select patterns for *identification*

   - ✓ Pick Authenticator and SSO (usability)
   - ✓ Checking the table, SSO benefits from Credential Tokenizer

6. Iteration: Confidentiality (4)

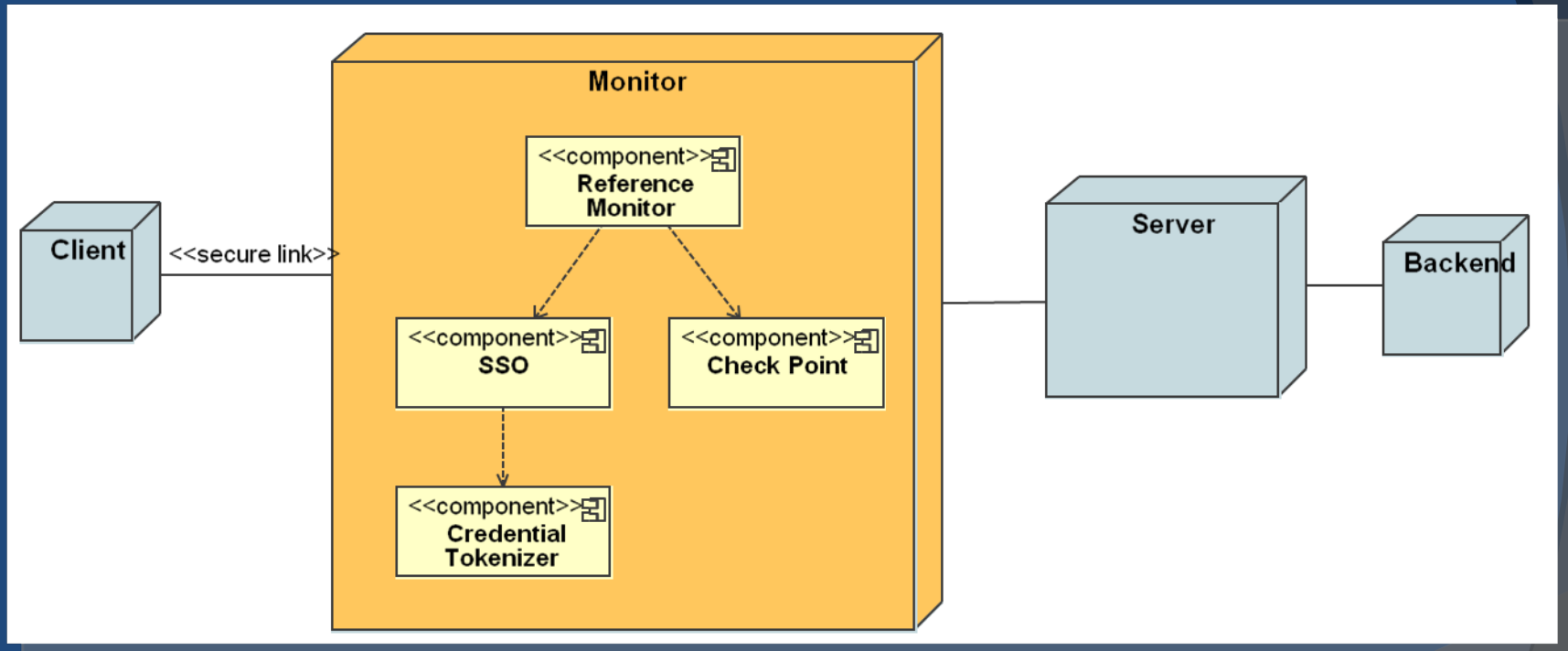
1. Using the description of the patterns: refine the architecture
   - Single Sign-On



   - Reference Monitor

6. Iteration: Confidentiality (4)

Result after this step



Final architecture