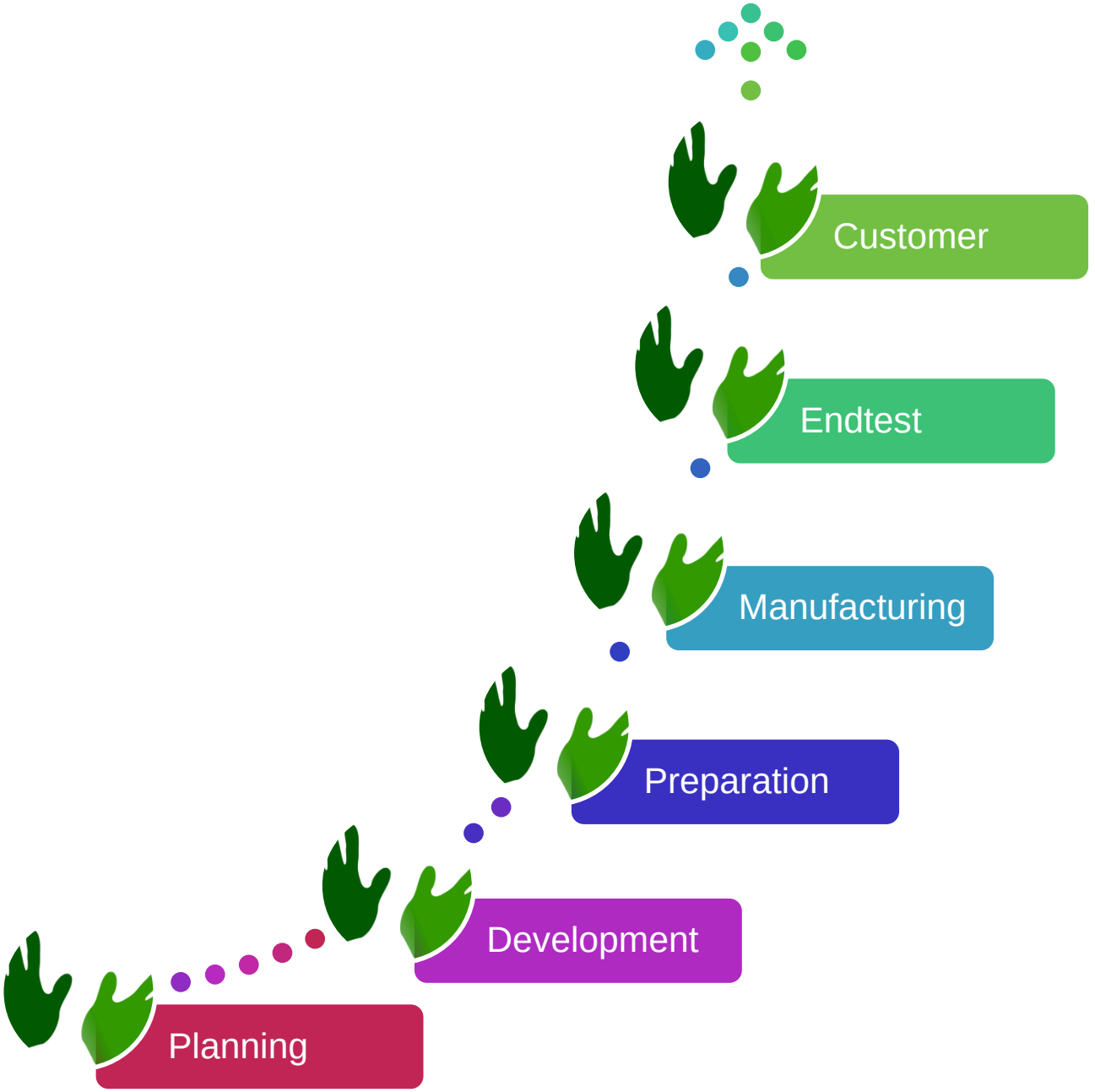
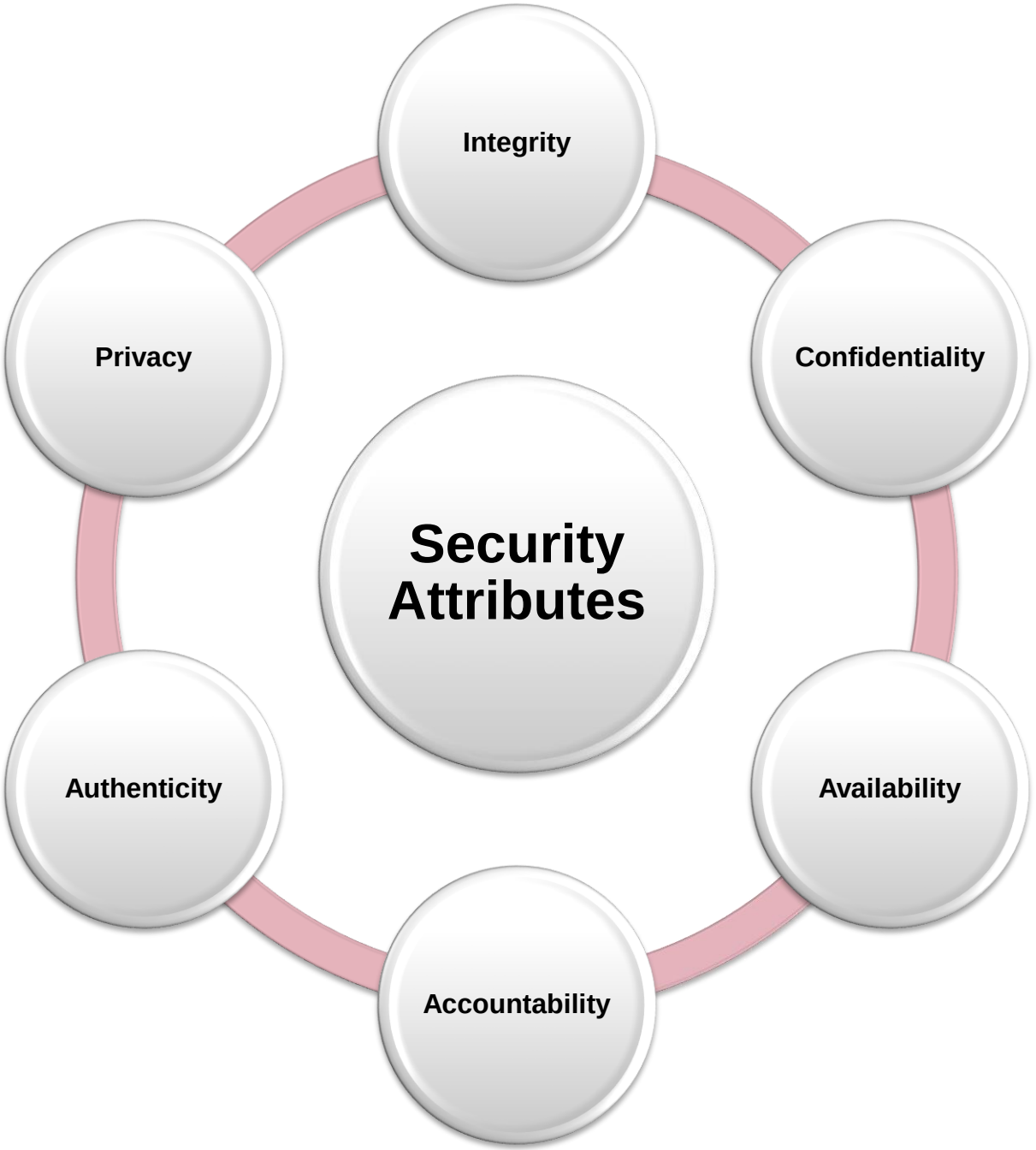
Secure Design Patterns

Wurzelbehandlung für Security

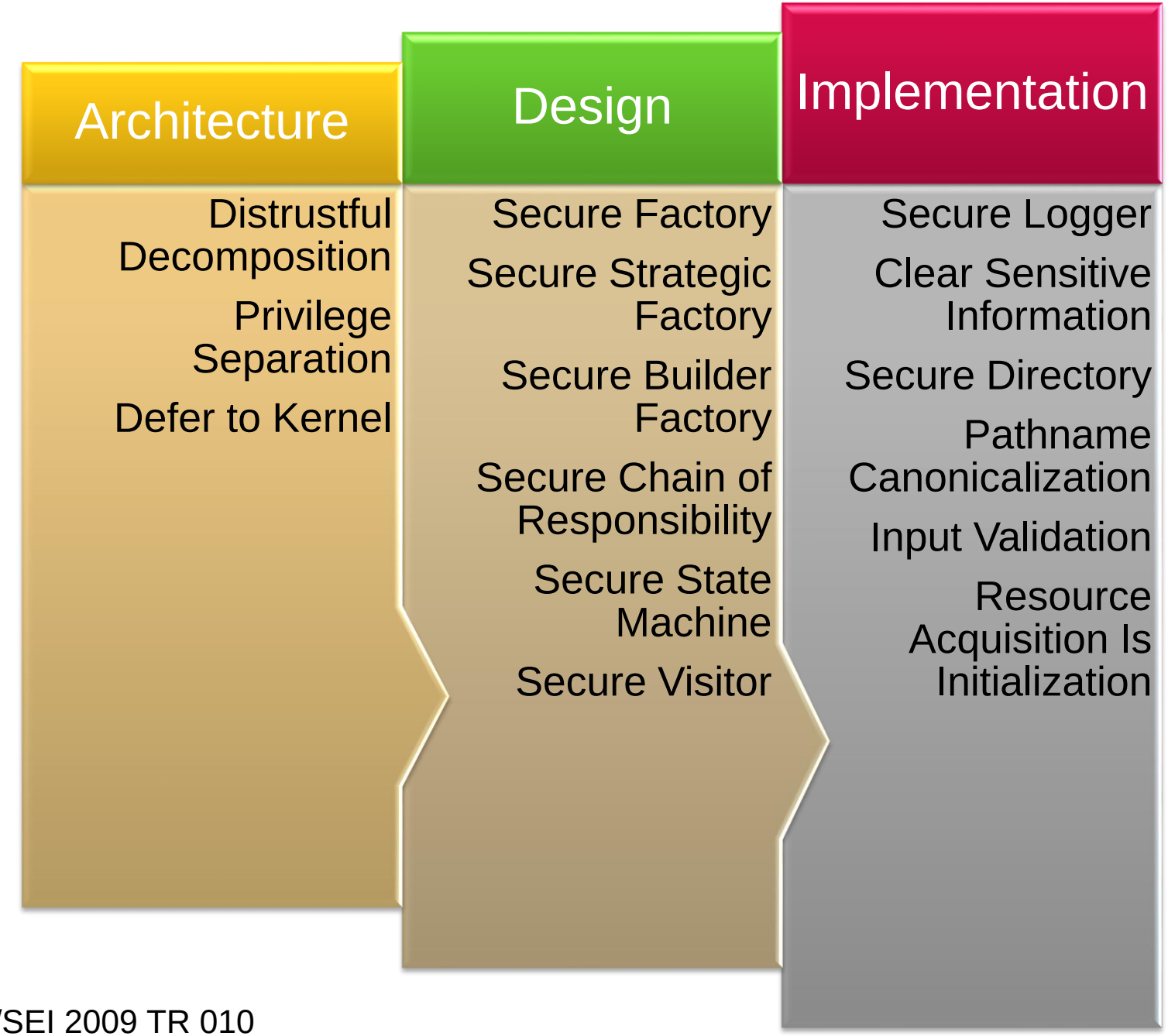
Marcus Gößler
MicroConsult GmbH




Integrity
Confidentiality
Availability
Accountability
Authenticity
Privacy
Security Attributes

Security Level (SL) as a result of Threat Level and Impact Level

Security Level (SL)	Impact Level (IL)					
Threat Level (TL)		0	1	2	3	4
	0	QM	QM	QM	QM	Low
	1	QM	Low	Low	Low	Medium
	2	QM	Low	Medium	Medium	High
	3	QM	Low	Medium	High	High
	4	Low	Medium	High	High	Critical



Ref.: CMU/SEI 2009 TR 010

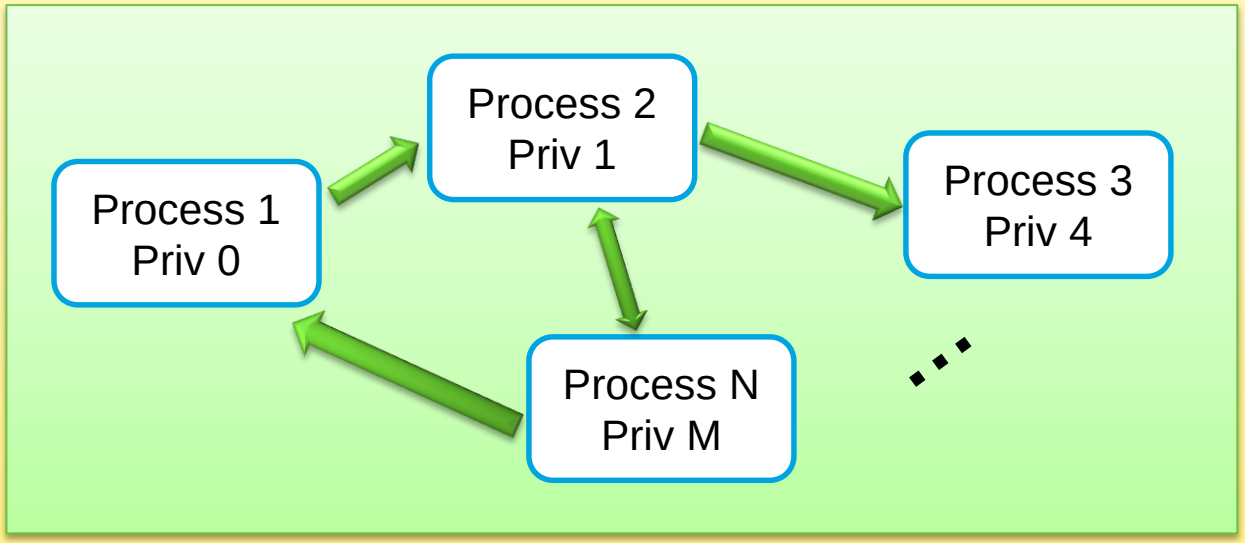
- No single pattern addresses all issues
- Patterns are not always applicable
- Patterns intend to close security flaws, but might open others

Objective

- Reduce individual attack surface by separate untrusting programs
- Expose less in case of one program hacked

Context

- System contains several higher-level functions
- Functions with different privilege levels

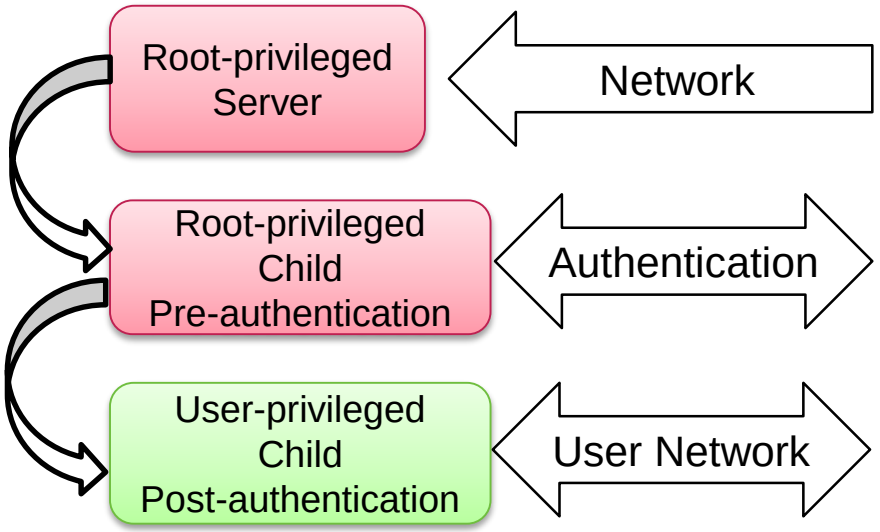


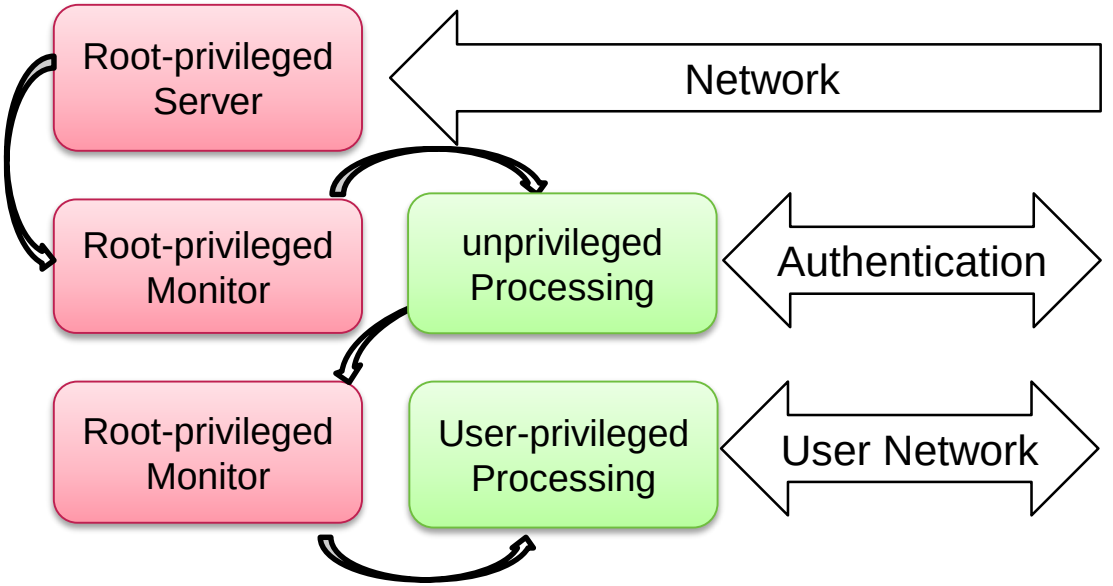
Objective

Reduce amount of code running with special (elevated) privilege

Context

- Function sets not requiring elevated privileges
- Functions with a lot communication to untrusted sources
- Functions made of complex (thus error prone) algorithms





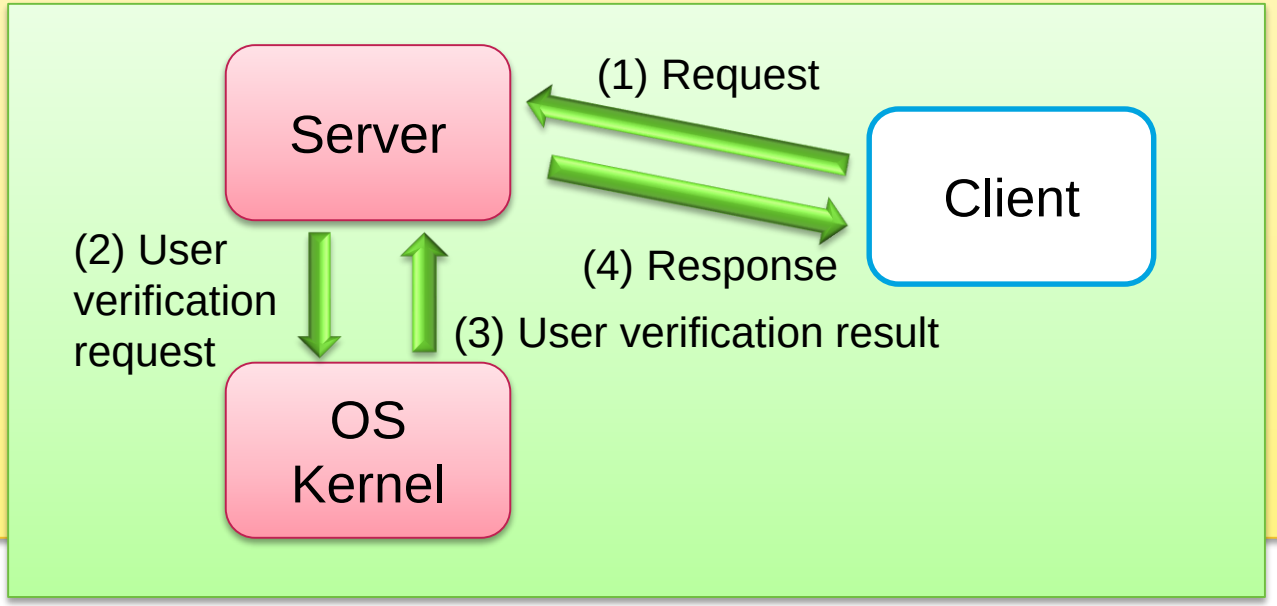
MICRO CONSULT

Objective

- Separate functions w/ elevated privileges from those w/o, and
- Utilize existing user verification functionality of the kernel

Context

- System of users w/o elevated privileges
- Some functionality requires elevated privileges
- System must verify permission of users to execute this functionality

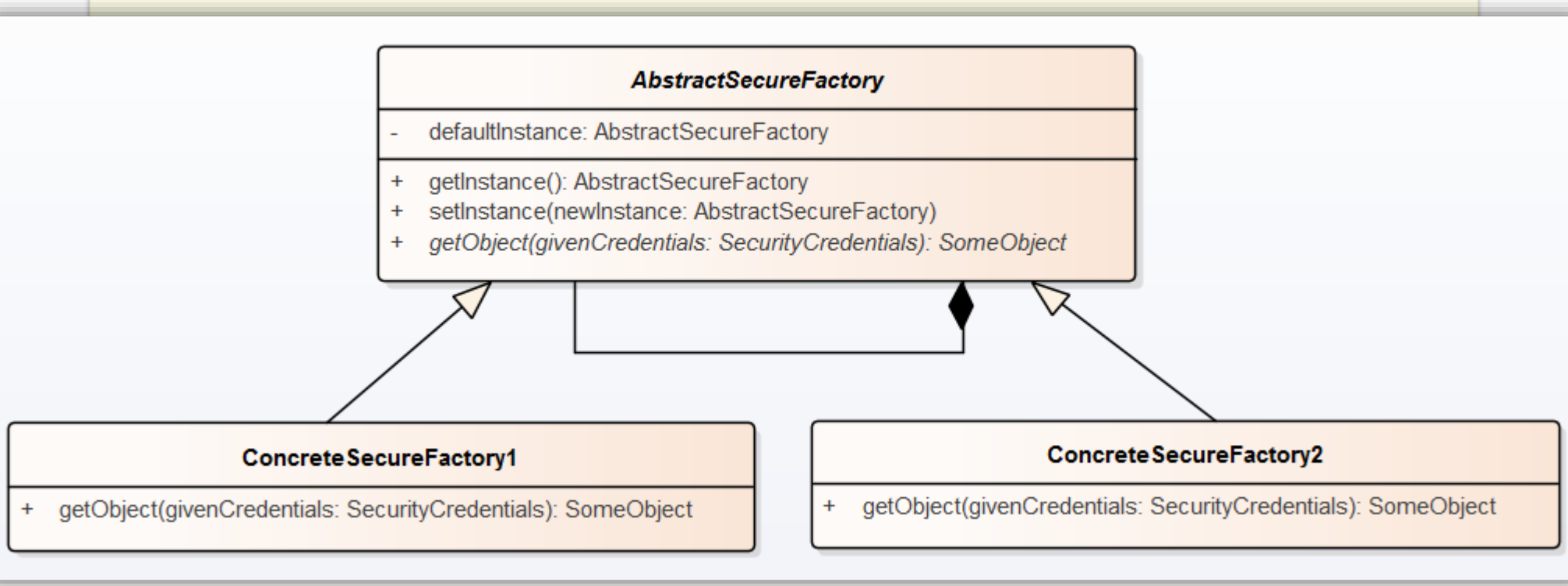


Objective

- Separate security-sensitive creation or selection of objects, *from*
- Functionality of created or selected objects

Context

- Different versions of objects based on security credentials
- Security credentials contain all needed information for creation/selection

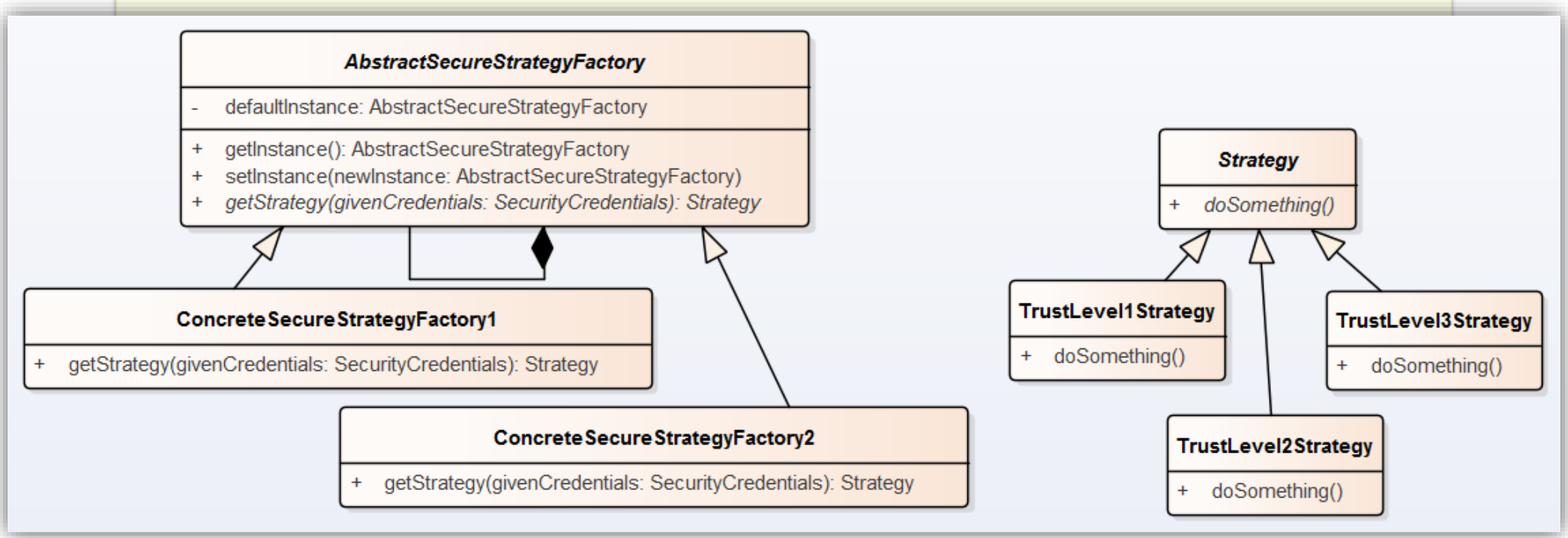


Objective

Easy selection and modification of appropriate strategy object

Context

- Varying security-credential-based specific behavior of a general function
- Variations can be implemented w/ classes using the Strategy pattern
- Selection of the specific behavior by security credentials alone

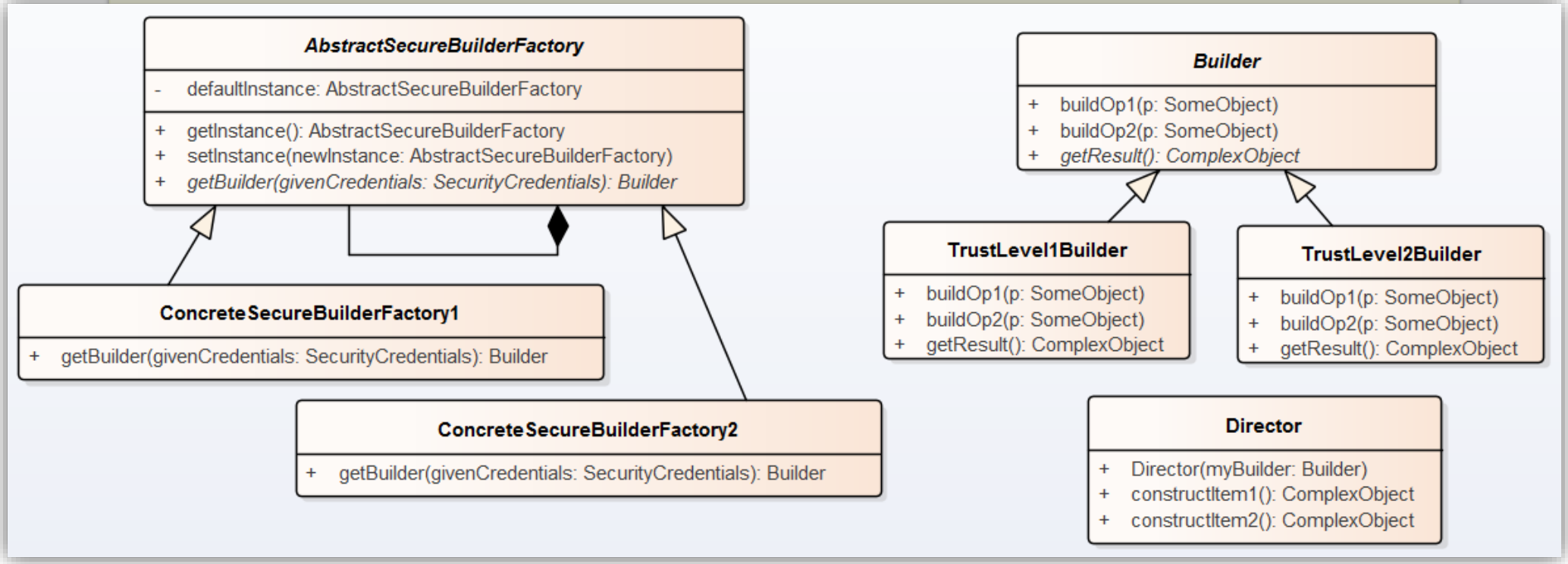


Objective

- Separate security-dependent rules to create a complex object, *from*
- Basic steps to actually create the object

Context

- Creation of complex objects
- Variations of complex object constructed based on security credentials
- Construction of complex object defined by security credentials alone

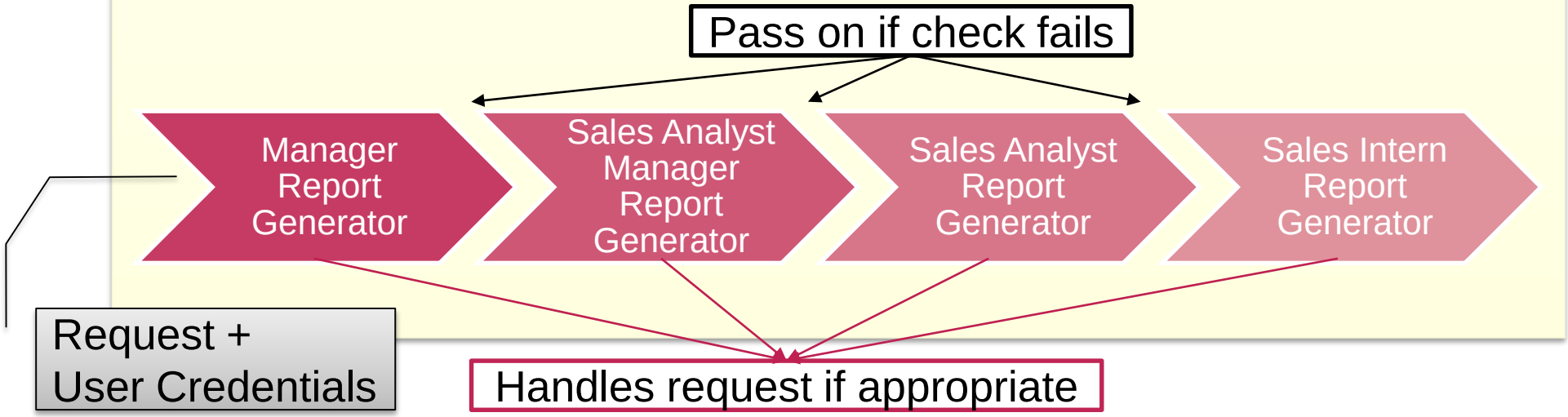


Objective

- Decouple trust dependent functionality from application requesting it
- Simplify trust dependent functionality and allow easy change

Context

- Varying security-credential based specific behavior of a general function
- Selection based on security-credentials
- Higher trust-levels are supersets of lower trust-levels
- Security-credentials determine level of handling a given request

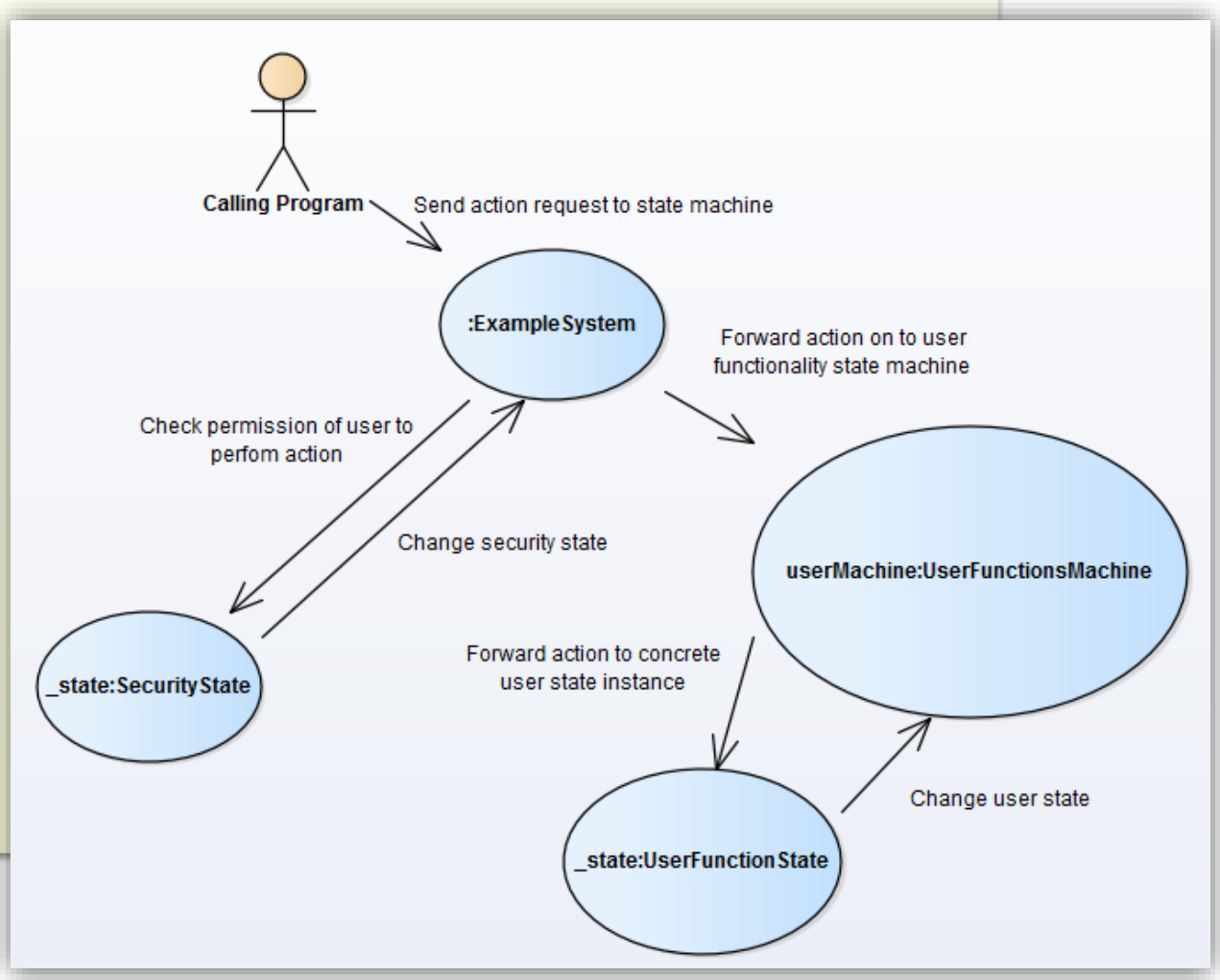


Objective

Separate security and user-level functionality as two separate state machines

Context

- User-level functionality can be cleanly represented as finite state machine
- Access control model for state transitions can also be represented as state machine

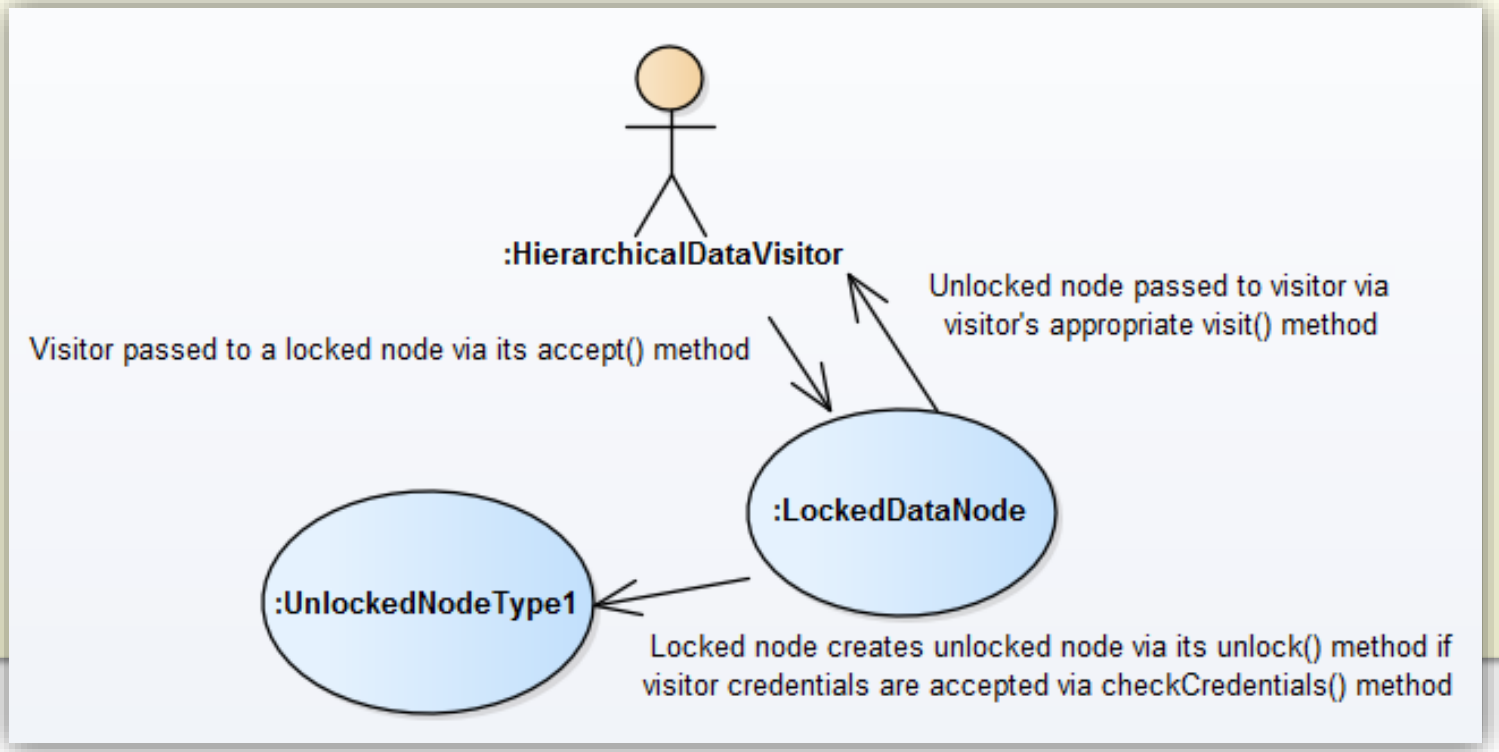


Objective

Allow data nodes to lock themselves unless visitor provides proper credentials for unlocking

Context

- System with hierarchical data
- Data nodes with different access privileges

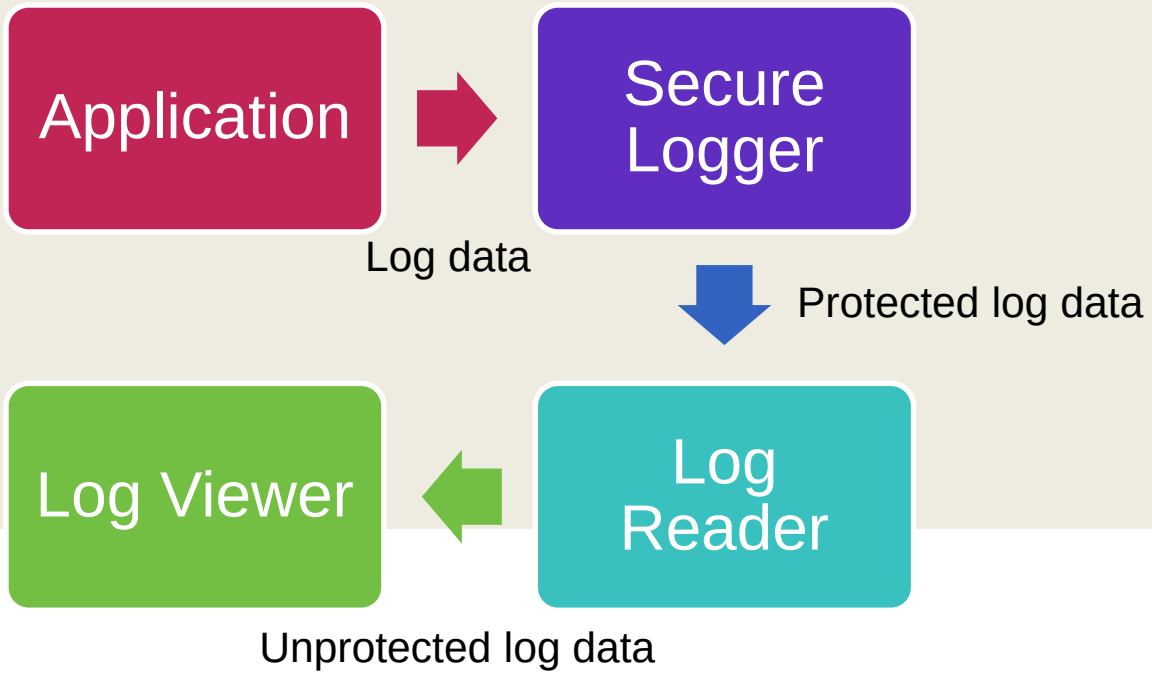


Objective

- Prevent gathering of sensitive information from system logs
- Prevent hiding actions from attackers editing system logs

Context

- System logs are used
- Information contained is sensitive and potentially used for an attack
- System logs are used to detect and diagnose attacks

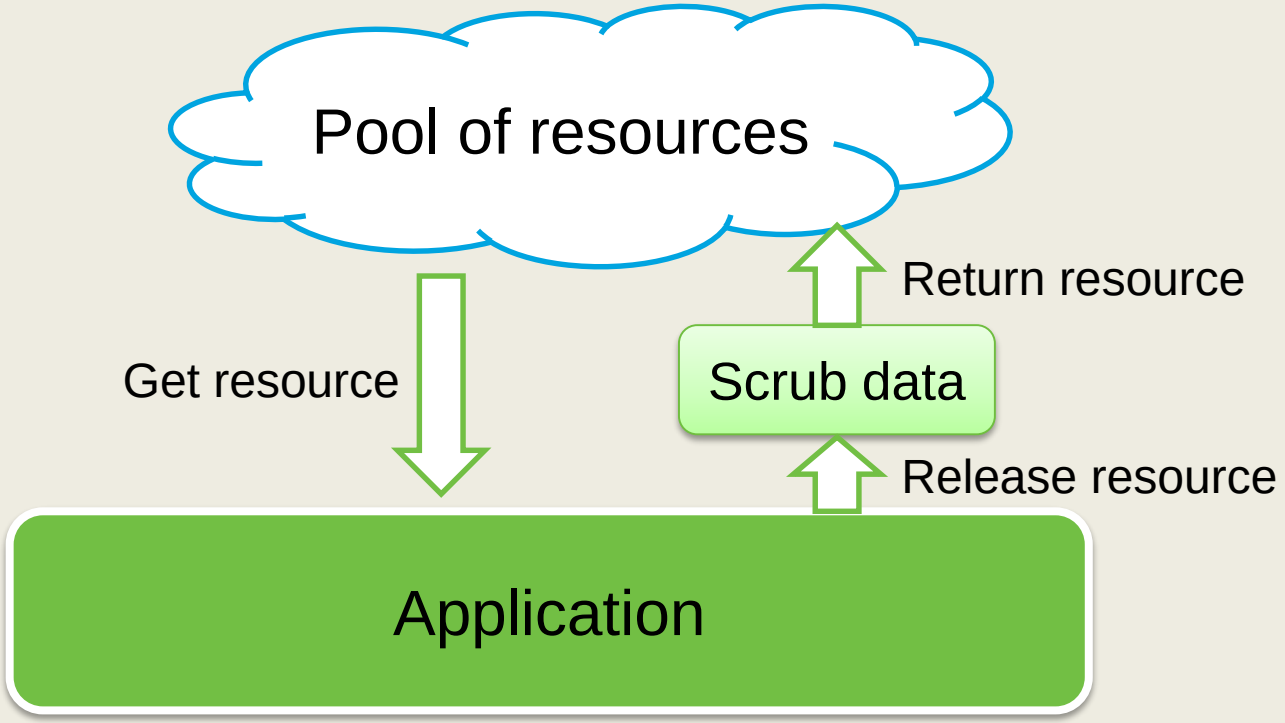


Objective

Sensitive information is not available in freed reusable resources

Context

Sensitive information is stored in reusable resources



Objective

Prevent manipulation of data (files) during its usage

Context

- Program is run in an insecure environment
- Program reads/writes data (files)
- Program execution is compromised if data is manipulated from outside

1. Find canonical pathname of directory
2. Check if directory is secure
    1. If secure → read/write
    2. If unsecure → Error, no read/write

Objective

- All data (files) are referred to by a valid path
- A valid path is the canonical path

Context

- Program accepts pathnames from untrusted sources
- An attacker could provide a pathname, that *non-obviously* refers to a protected directory
- Each data (file) has a unique canonical pathname

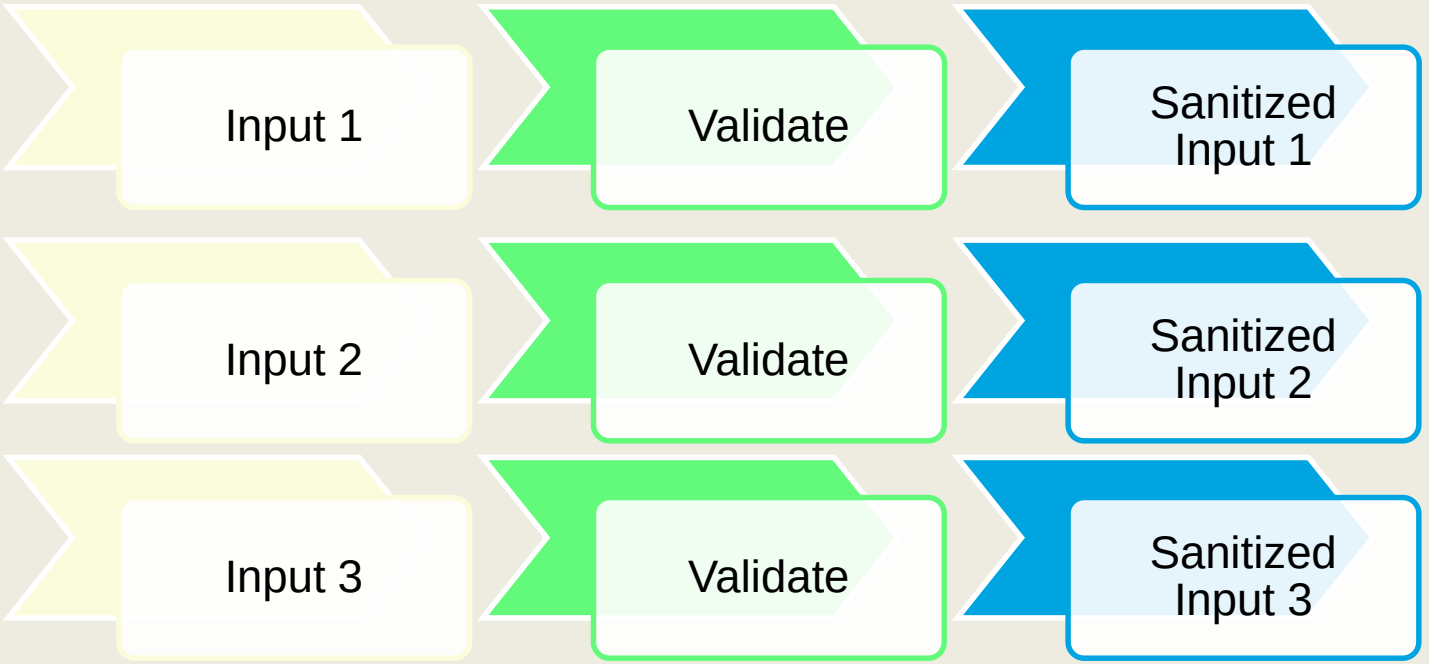
1. Utilize pathname-canonicalization function on given pathname prior to accessing/opening data (file)
2. Use canonical pathname when operating on data (file)

Objective

Validate data being input

Context

- Any software accepting data from untrusted sources
- Any data crossing security boundaries



Objective

Performing resource allocation and deallocation in an object's constructor and destructor

Context

- Systems using resources, which need to be acquired and released
- Systems with finite resources

≠