# Foundations of Software Engineering

Lecture 11: Architecture & Security

Eunsuk Kang

# Leaning Goals

- Understand key elements of security architecture and analysis
- Understand the major challenges of achieving security in practice
- Understand implications of architectural decisions on security
- Apply architectural principles & mechanisms to build security into the design of a system

# Key Elements of Security

- Security requirements (policies)
  - What needs to be protected?
- Threat model
  - What are capabilities & intents of an attacker?
- Attack surface
  - Which interfaces are exposed to an attacker?
- Protection mechanisms
  - How do we prevent an attacker from compromising a security requirement?

# Security Analysis Question

Having identified:

- Security requirements
- Threat model
- Attack surface
- Protection mechanisms

Does my system deploy sufficient **protection mechanisms** to establish its **security requirements** in the presence of an **attacker** who may attempt to compromise the system through its **attack surface**?

# What is security so hard?

- Security requirements
  - Often implicit, conflicting views of security
- Threat model
  - Uncertain, evolving attacker model
- Attack surface
  - Multiple interfaces across system layers
- Protection mechanisms
  - Human factors; no foolproof mechanisms

# Examples of Security Failures

# Wrong Threat Model


# Wrong Threat Model


**Maginot Line (1930s)**

Built by France to deter invasion; state-of-the-art engineering

Germans reformulated plans after WWI; cut across Belgium

# Unidentified Attack Surface

**Château Gaillard (1200s, Normandy, literally "Strong Castle")**

Impervious; under siege for 6 months by Phillip II (France)

Conquered by entering through toilet chute

# Insufficient Protection Mechanism

**Trojan Horse (Greeks vs Troy; 12th BC?)**
Disguised as a harmless trophy; hidden payload inside
**Lesson**: Treat all system inputs as potentially malicious

# Wrong Security Requirement

NEWS

**Ransomware takes Hollywood hospital offline, $3.6M demanded by attackers**

**Hollywood Presbyterian ransomware attack (2016)**

Computer systems frozen; patients transferred

Availability of critical services, not data exposure

# Strategies for Secure Design

- Security requirements
  - Elicitation & precise documentation
- Threat model
  - Principle of least privilege: Assume the worst
- Attack surface
  - Isolation: Separate the critical components
- Protection mechanisms
  - Defense in depth: Mitigate the weakest link

# Security Requirements

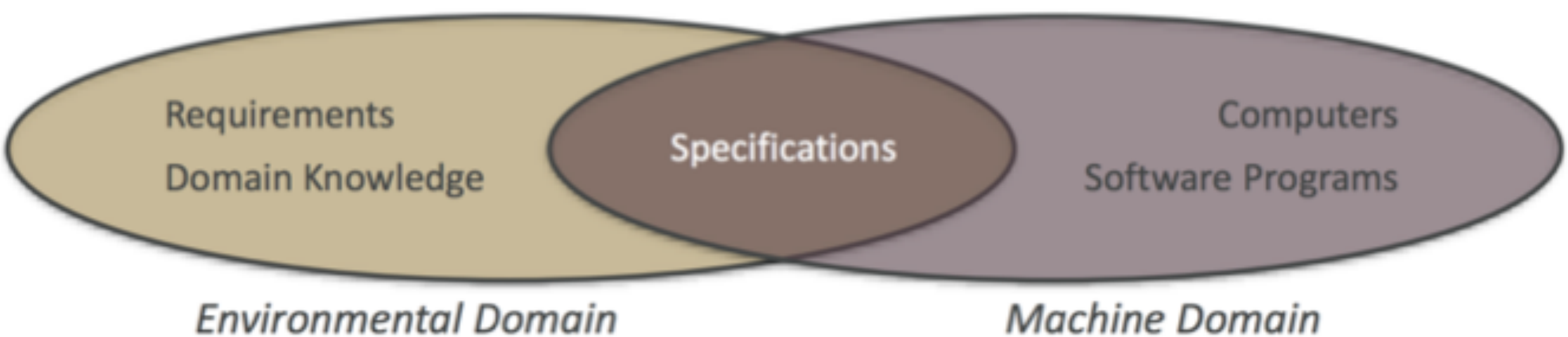
# Security Requirements

- **C**onfidentiality
    - Sensitive data accessible to authorized parties only
- **I**ntegrity
    - Sensitive data modifiable by authorized parties only
- **A**vailability
    - Services made available when needed by clients
- (Non-repudiation)

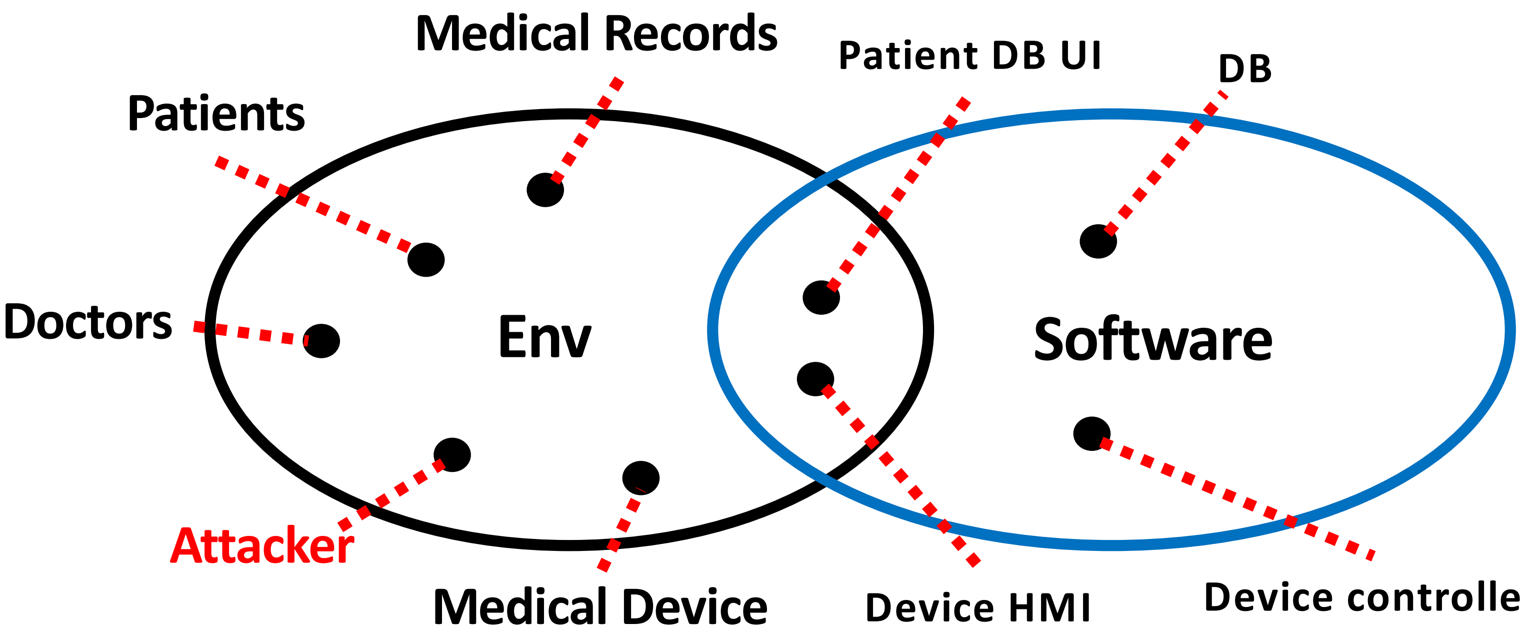
**“CIA triad”**

# Machine vs Environment

- Requirements are about phenomena in the **environment**
- Machines (software) are built to act on **shared** phenomena
- Same for security; must start from the environment!



# Example: Hospital




**Critical Req.** Patients must be given timely treatments by doctors based on their medical conditions

**Q.** What kind of security guarantee should software provide? Confidentiality? Integrity? Availability?

# Exercise: Graduate Admission System

FEATURE

## Hacker helps applicants breach security at top business schools

Among the institutions affected were Harvard, Duke and Stanford

Using the screen name "brookbond," the hacker broke into the online application and decision system of ApplyYourself Inc. and posted a procedure students could use to access information about their applications before acceptance notices went out. The hack was posted in a *Business Week* online forum mainly frequented by business students, said Len Metheny, CEO of the Fairfax, Va.-based ApplyYourself.

# Exercise: Graduate Admission System

**Q.** What are key security requirements of
the CMU graduate admission system?

# Architectural Design for Security

Slides adapted from: John Mitchell, Stanford

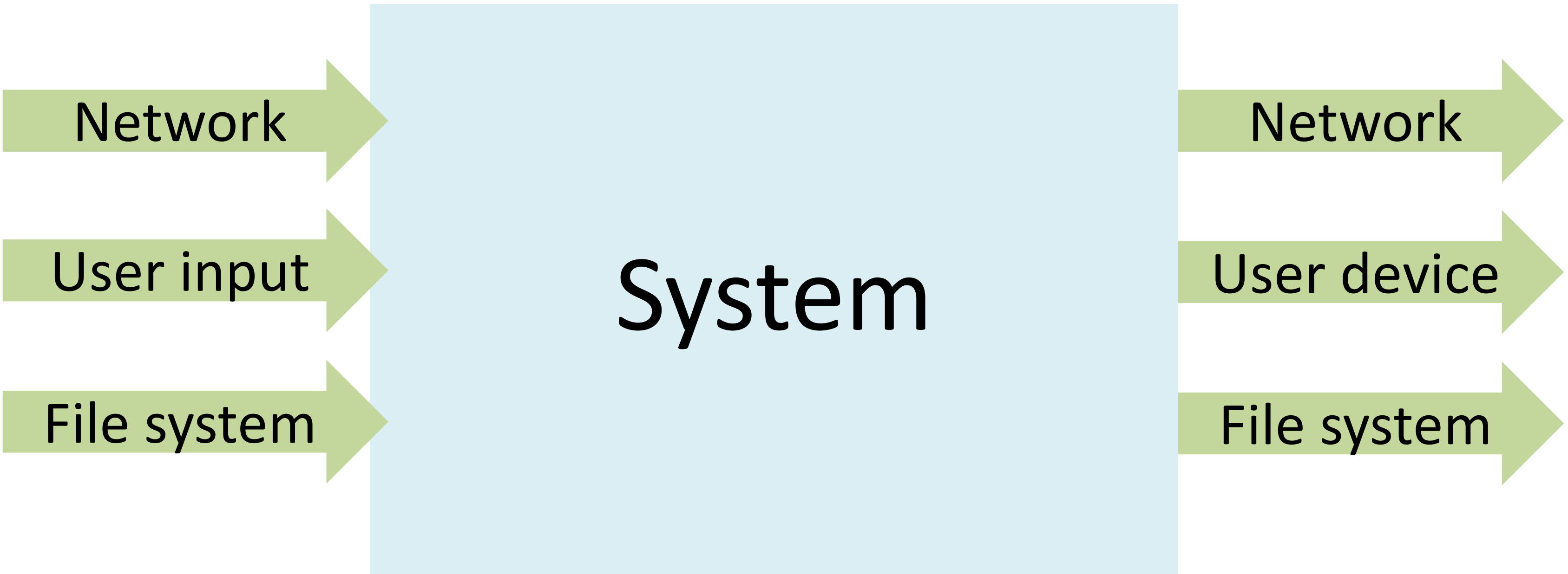
# Architectural Strategies for Security

- Principle of Least Privilege
  - A component should be given the **minimal** privileges needed to fulfill its functionality.
  - Goal: Minimize the impact of a compromised component.
- Isolation
  - Components should be able to interact with each other **no more than necessary**.
  - Goal: Reduce the size of **trusted computing base** (TCB)

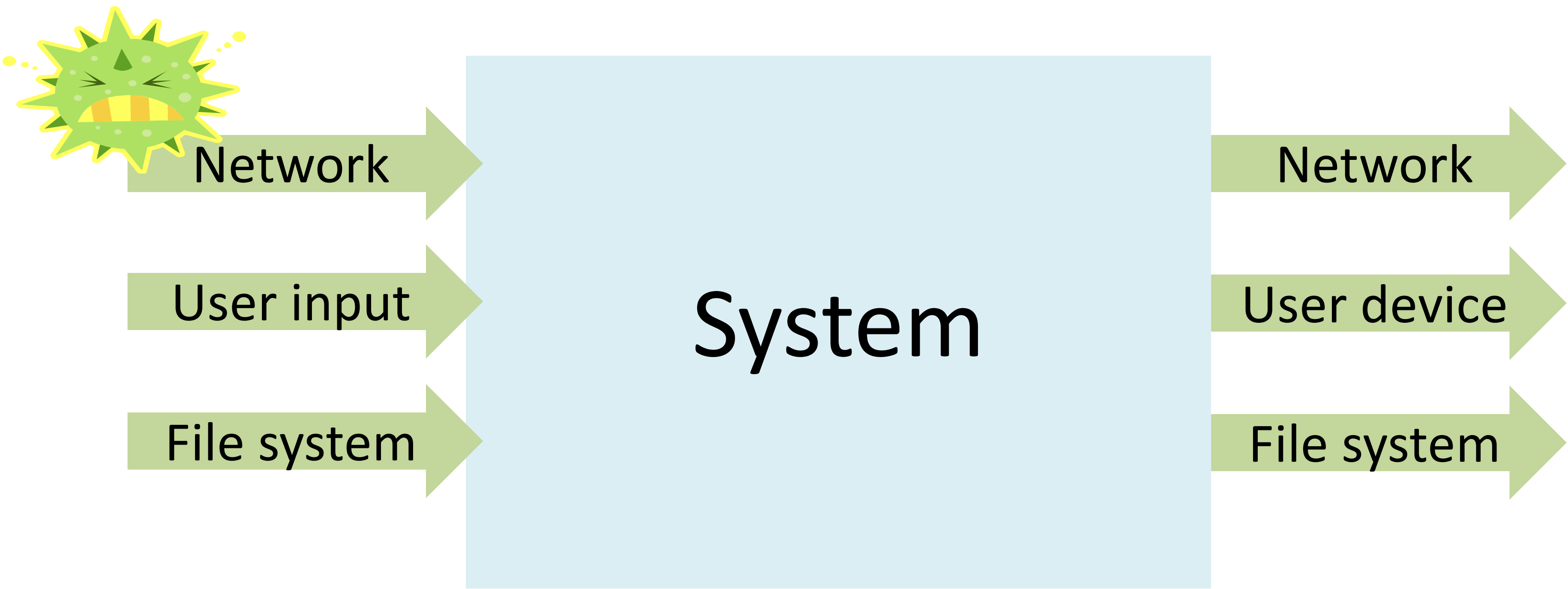
# Trusted Computing Base (TCB)

- Components responsible for establishing a security requirement(s)
  - If any compromised => security violation
  - Conversely, a flaw in non-TCB component => security preserved
- Design goal: **Minimize TCB**
  - Smaller TCB, less software to inspect & verify
  - In poor designs, TCB = entire system

# Monolithic Design



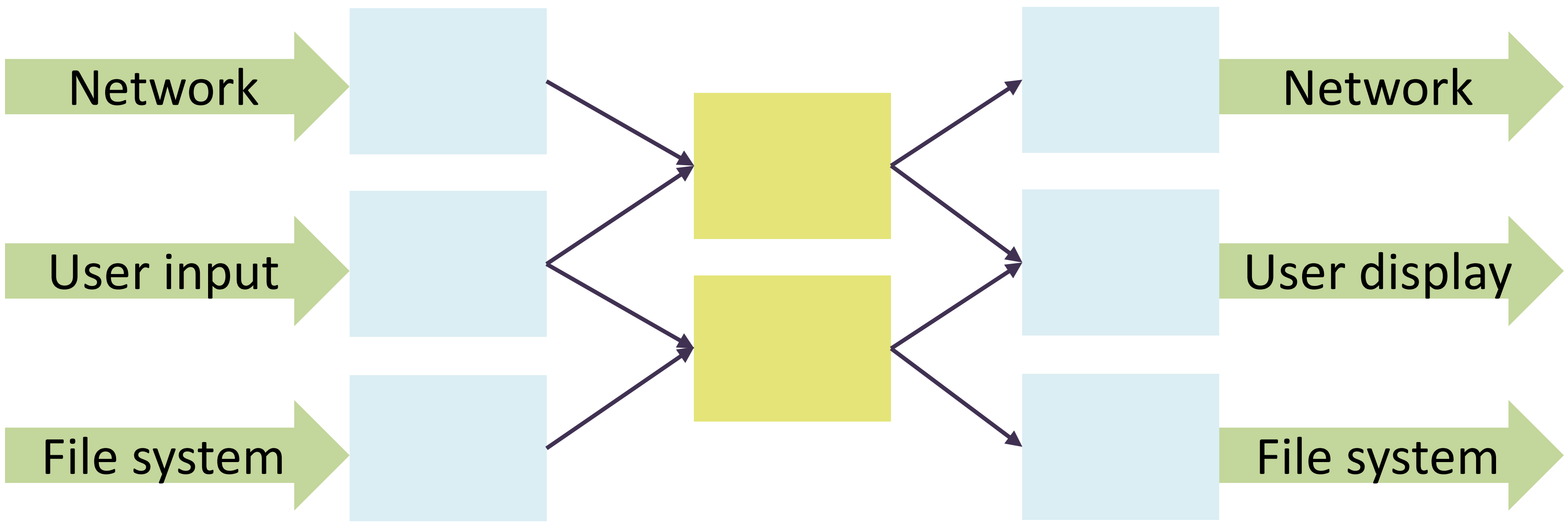
# Monolithic Design



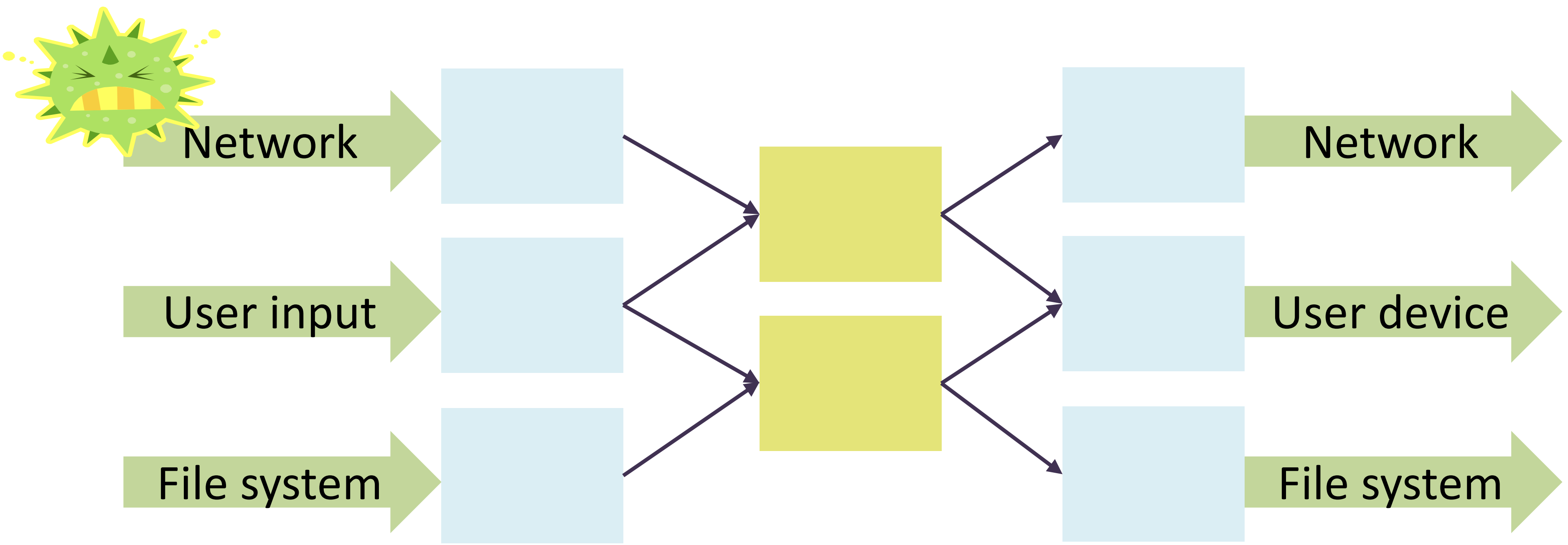
# Monolithic Design


Flaw in any part of the system =>
Potential security failure!

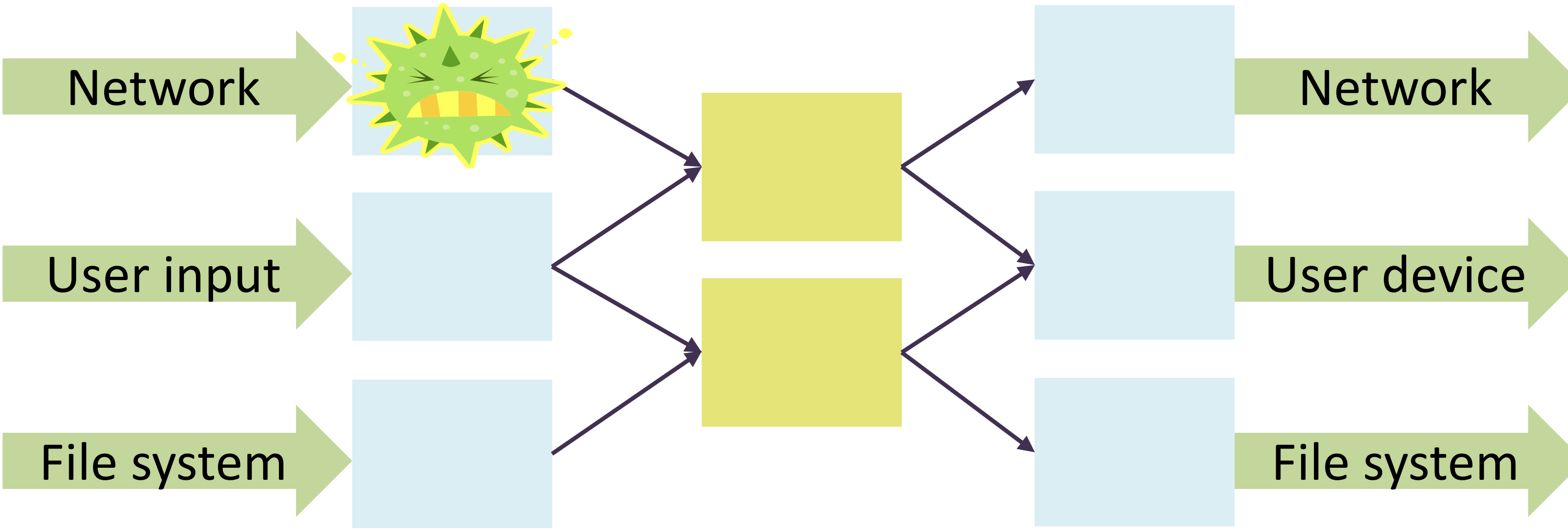
# Component Design



# Component Design



# Component Design




Flaw in one part of the system =>
Limited impact on security!

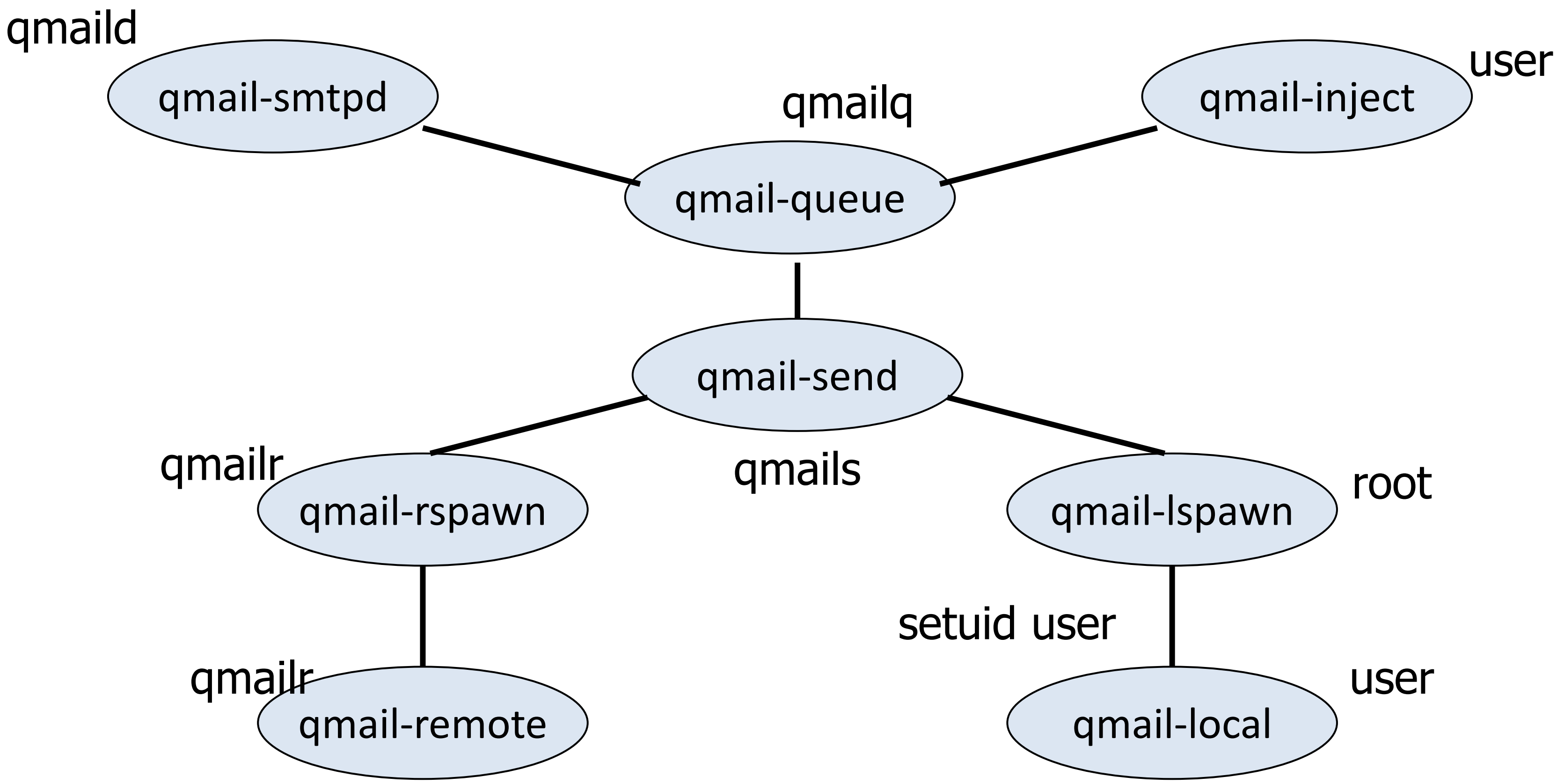
# Example: Mail Agent

- Requirements
  - Receive & send email over external network
  - Place incoming email into local user inbox files
- Sendmail
  - Traditional Unix
  - Monolithic design
  - Historical source of many vulnerabilities
- Qmail
  - “Security-aware” mail agent
  - Compartmentalized design

# Qmail Design

- Isolation based on OS process isolation
  - Separate modules run as separate “users” (UID)
  - Each user only has access to specific resources (files, network sockets, …)
- Least privilege
  - Minimal privileges for each UID
  - Mutually untrusting components
  - Only one “root” user (with all privileges)
    - In comparison, entire sendmail runs as root

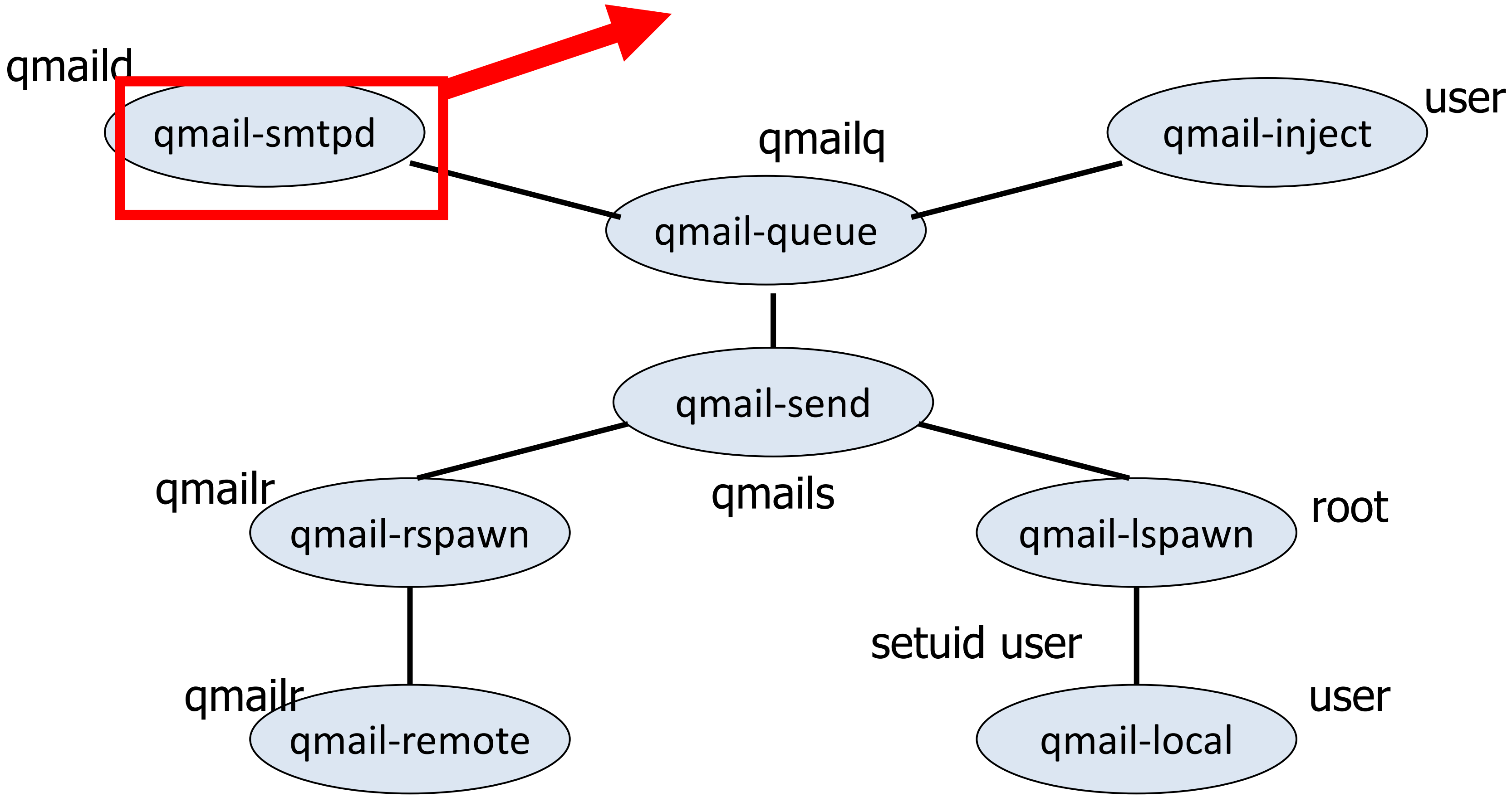
# Qmail Design



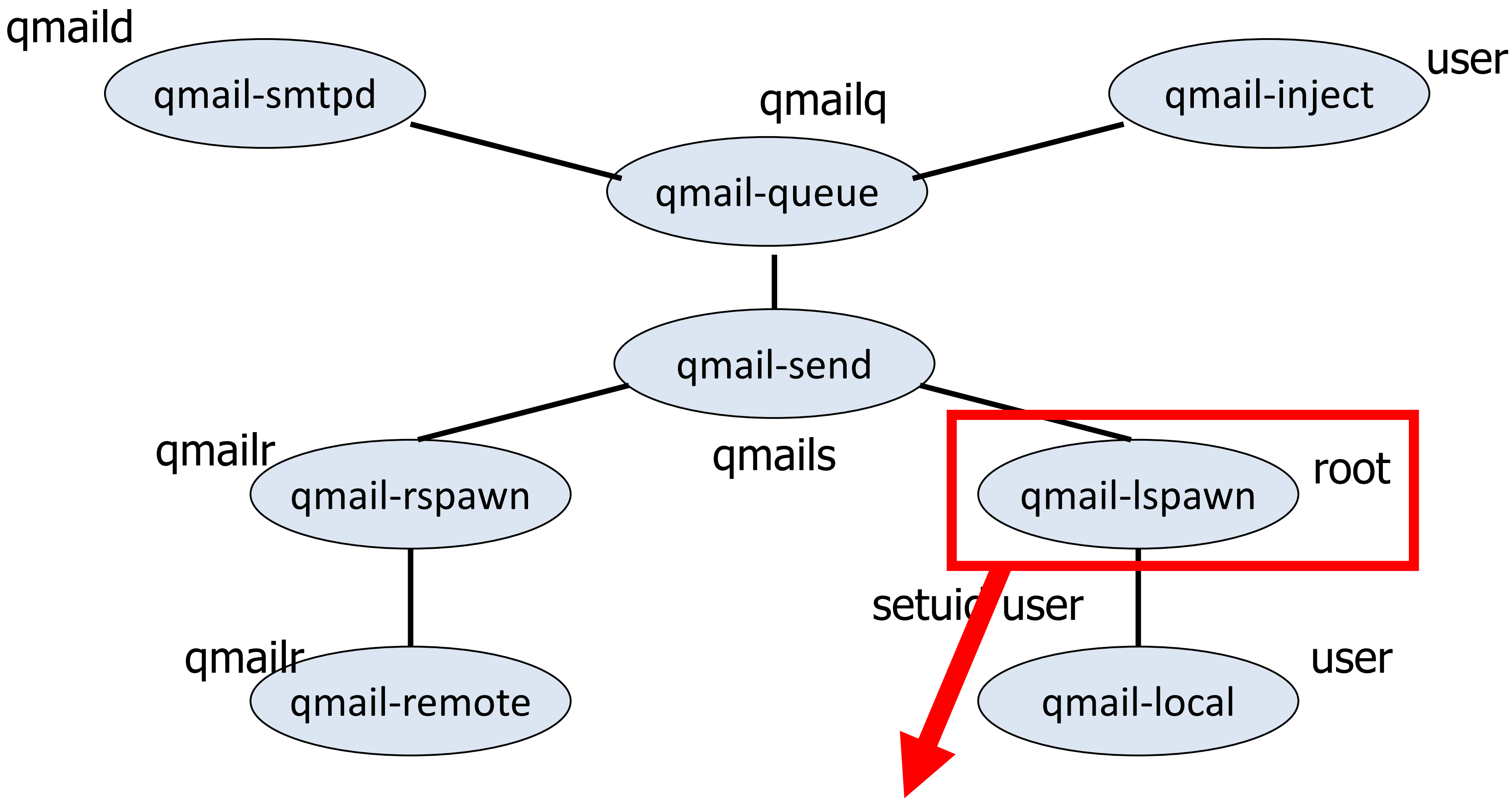

For more details: "The Security Architecture of qmail", Hafiz, Johnson, Afandi (2004)

# Qmail Design

Receives incoming external emails
Even if compromised, limited power
(vs. sendmail: runs as root)



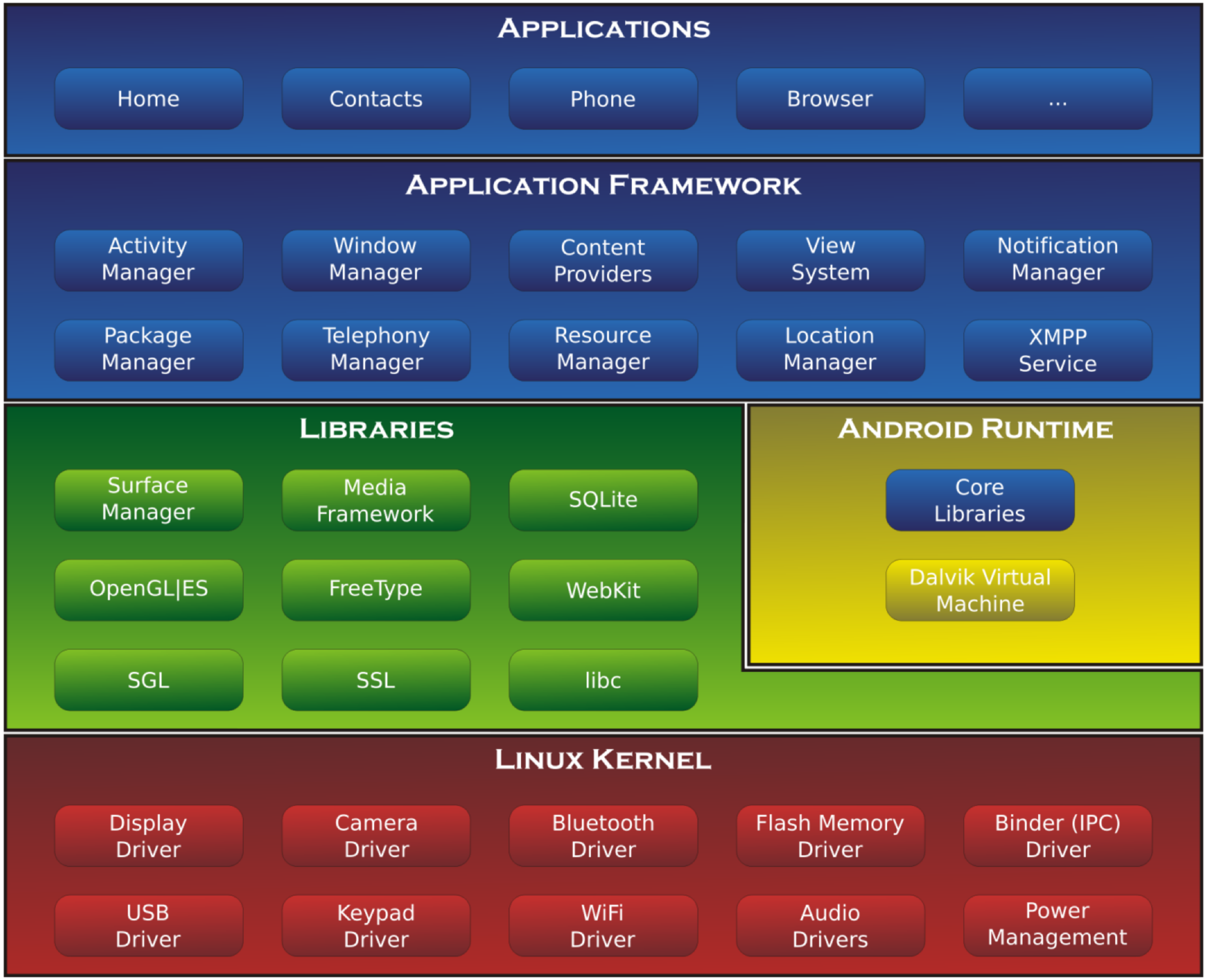
# Qmail Design




< 500 LOC
(vs. ~67K LOC in sendmail)

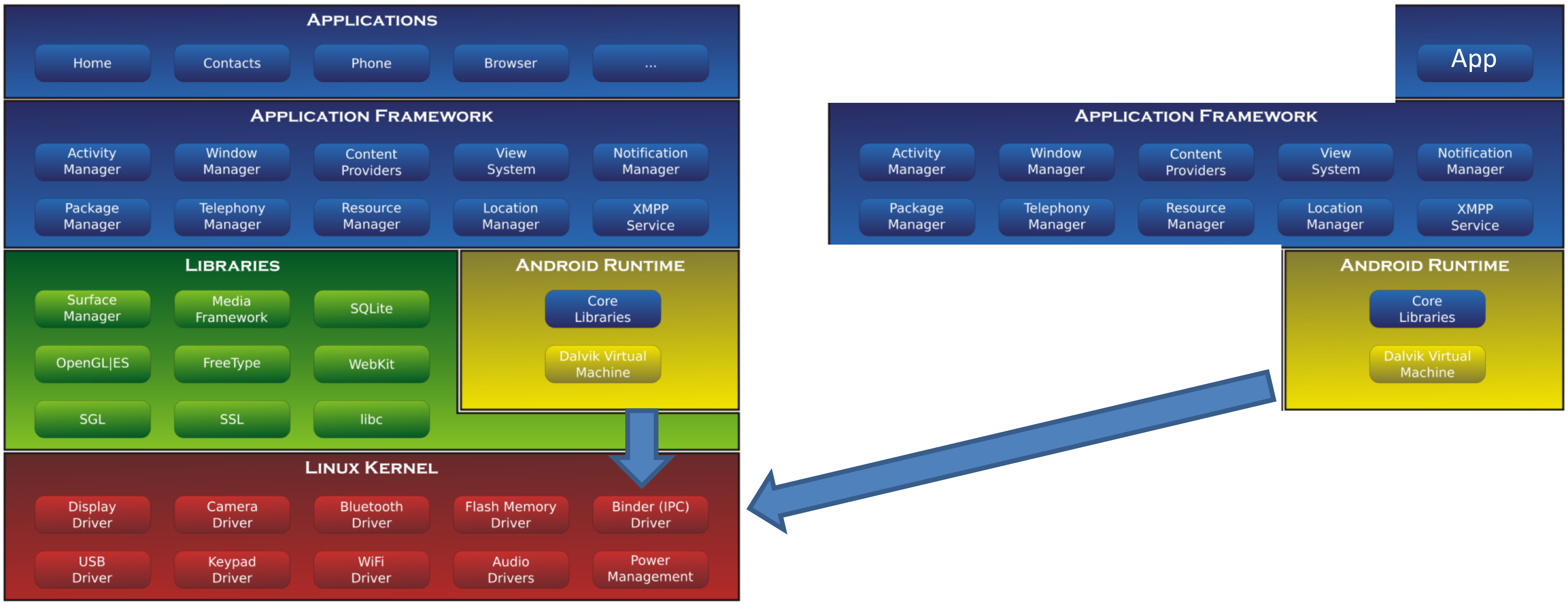
# Another Example: Android

- Isolation: Each app runs with its own UID & VM
  - Memory protection provided by OS
  - Inter-component communication: Permissions checked by reference monitor
- Least privilege
  - Application announces necessary permissions
  - User grants at install time

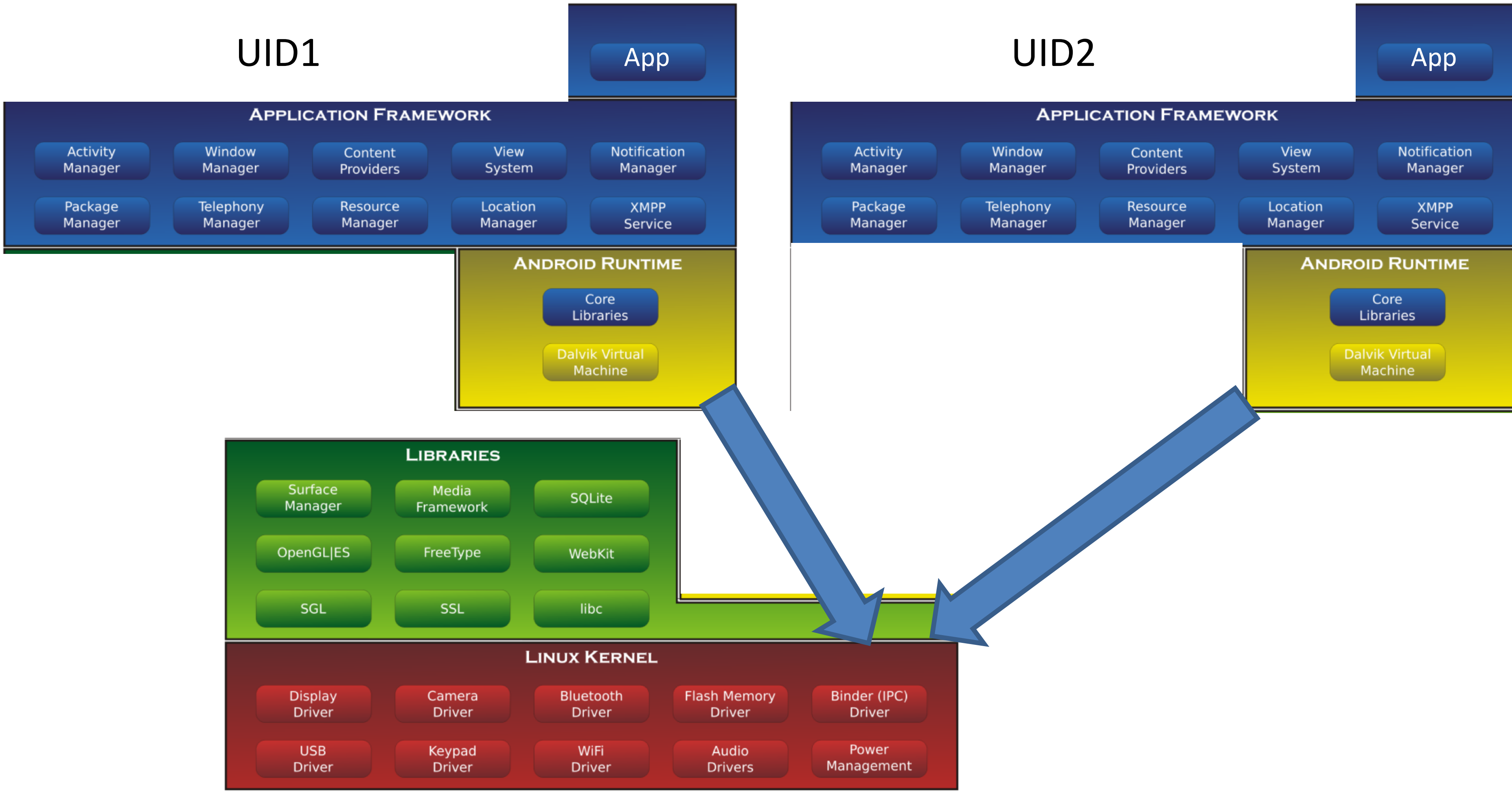


Applications
Home
Contacts
Phone
Browser
...
Application Framework
Activity Manager
Window Manager
Content Providers
View System
Notification Manager
Package Manager
Telephony Manager
Resource Manager
Location Manager
XMPP Service
Libraries
Surface Manager
Media Framework
SQLite
OpenGL|ES
FreeType
WebKit
SGL
SSL
libc
Android Runtime
Core Libraries
Dalvik Virtual Machine
Linux Kernel
Display Driver
Camera Driver
Bluetooth Driver
Flash Memory Driver
Binder (IPC) Driver
USB Driver
Keypad Driver
WiFi Driver
Audio Drivers
Power Management

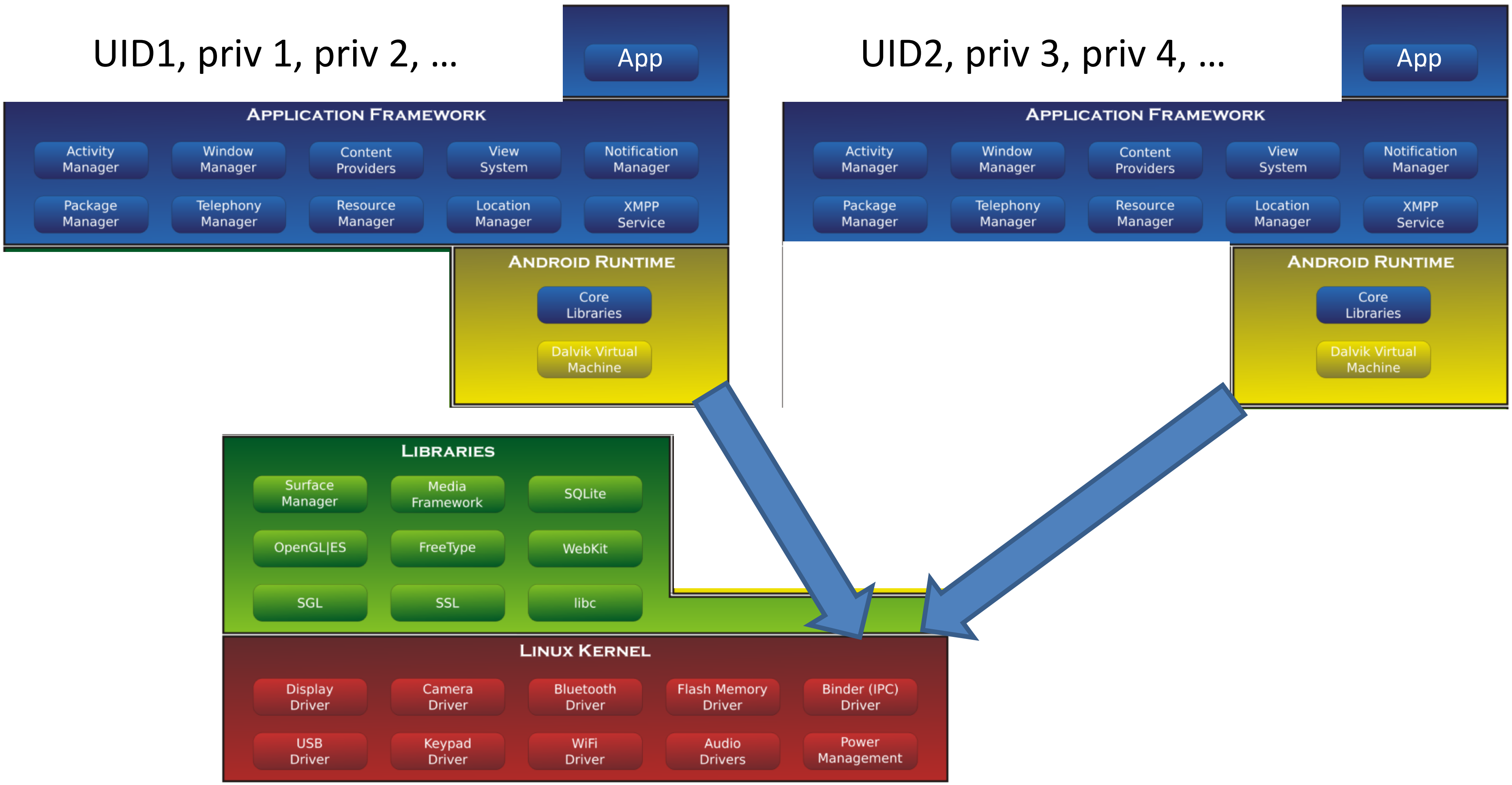
# Isolation: Different apps under different UIDs



# Isolation: Different apps under different UIDs



# Privileges set at install time



# Summary: Architectural Design for Security

- Monolithic vs compartmentalized design
- Principle of Least Privilege
  - A component should be given the **minimal** privileges needed to fulfill its functionality.
- Isolation
  - Components should be able to interact with each other **no more than necessary**.

# Questions during Architectural Design

- What are the major components of my system?
- What happens if a particular component is compromised?
- What is the TCB (i.e., components that are responsible for a security requirement)?
- Does any component have more privileges than needed?
- Is there sufficient isolation between critical & non-critical components?

# Architectural Security Analysis

# Security Analysis Question

Having identified:

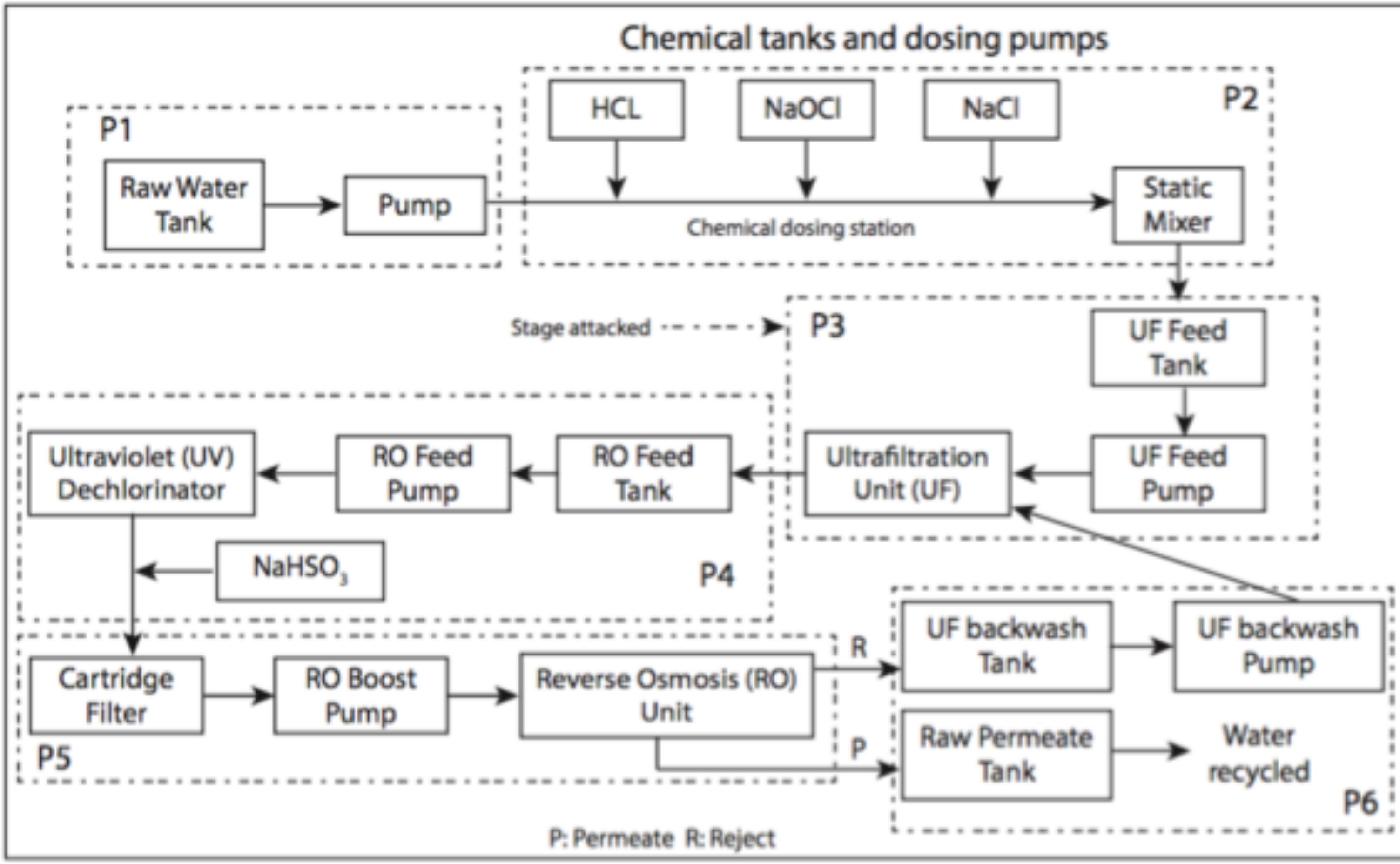
- Security requirements
- Threat model
- Attack surface
- Protection mechanisms

Does my system deploy sufficient **protection mechanisms** to establish its **security requirements** in the presence of an **attacker** who may attempt to compromise the system through its **attack surface**?

# Case Study: Water Treatment Plant

Fully functional plant (SUTD, Singapore)
Highly safety-critical! Failure may mean catastrophe

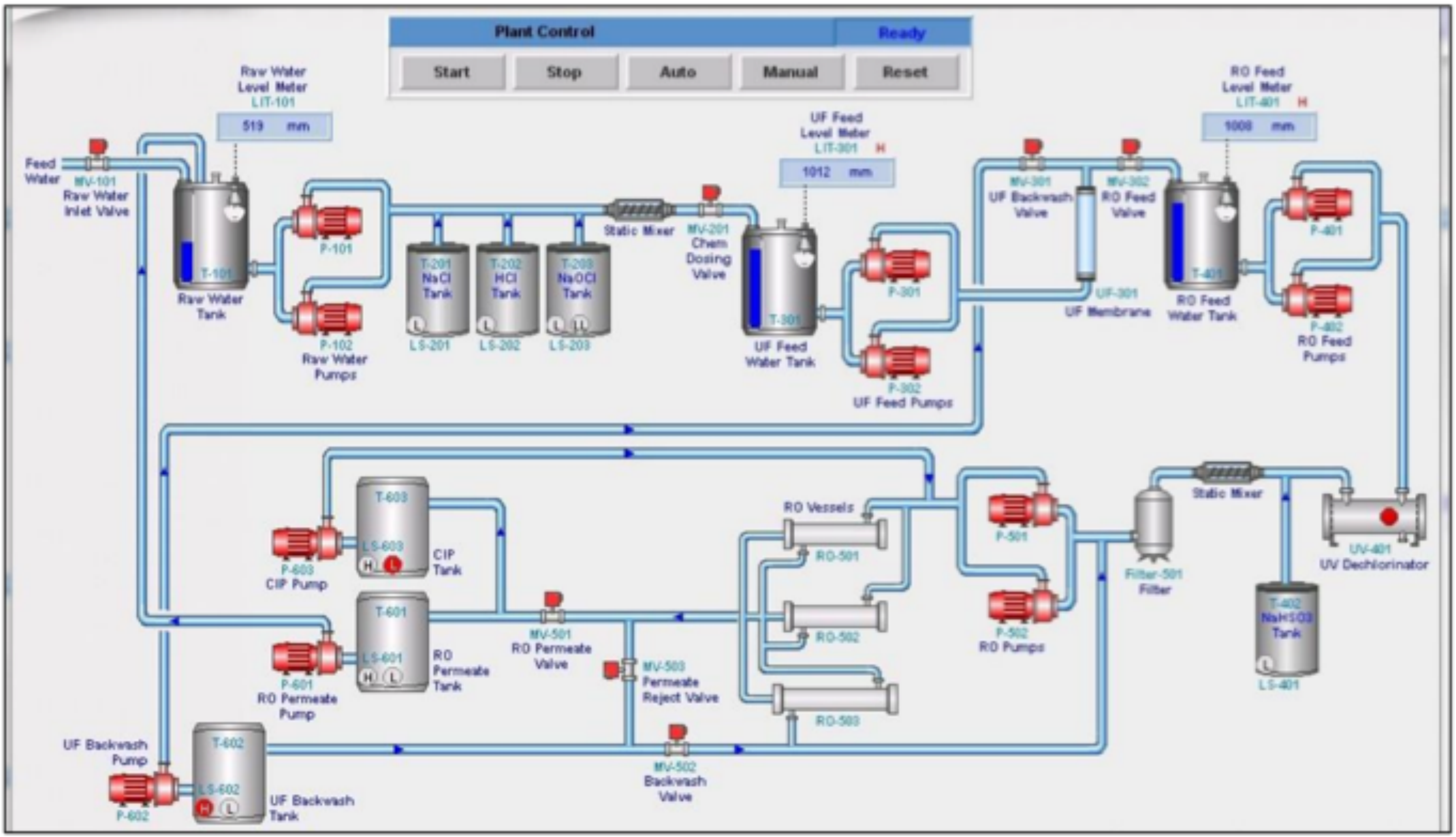
# Physical Architecture




Six different treatment stages (P1 – P6)

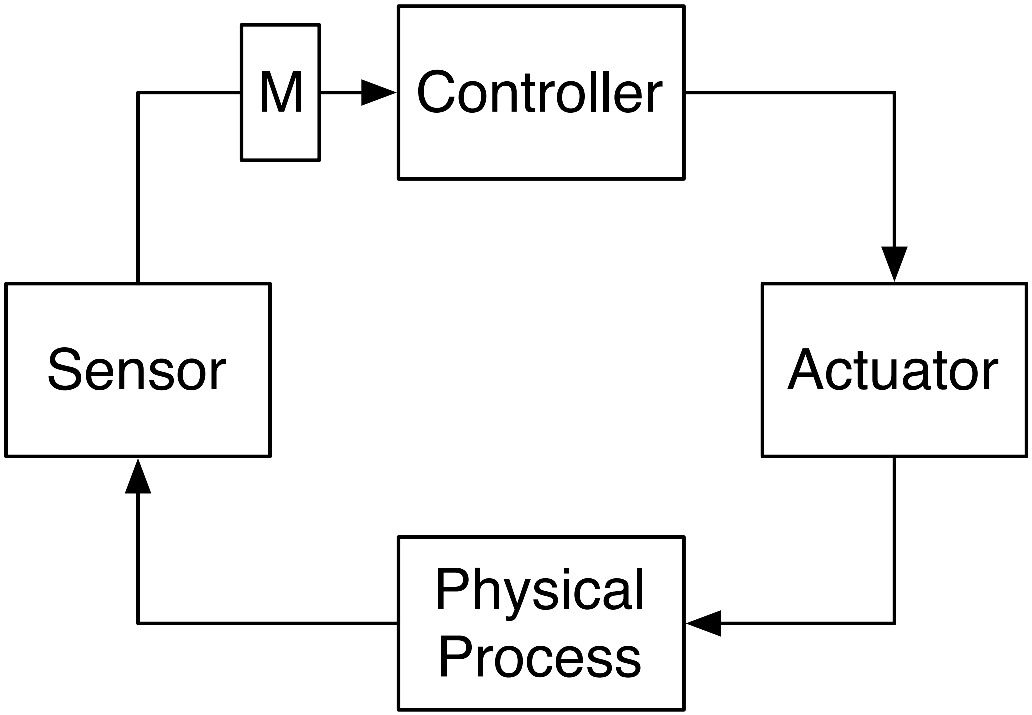
Each stage monitored by sensors for water quality

# SCADA Human-Machine Interface



# SCADA Human-Machine Interface

# Industrial Control System (ICS)

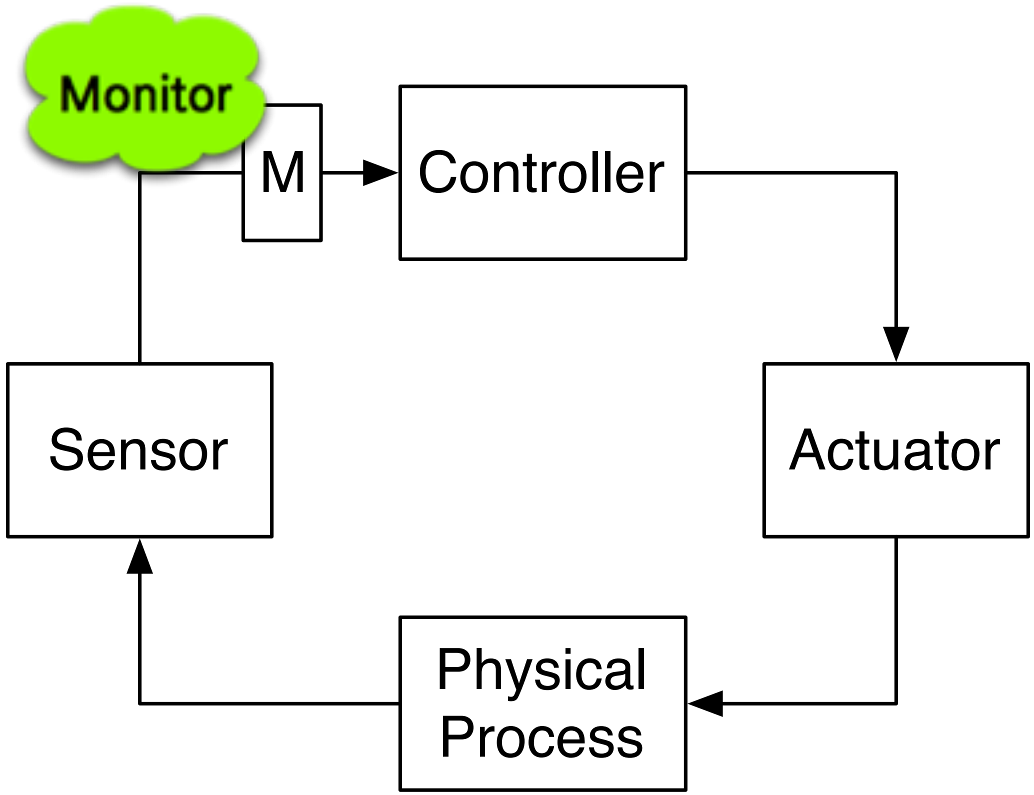



Physical Processes: Water tanks, pumps, valves

Controller: SCADA issues various actuator commands based on sensor readings

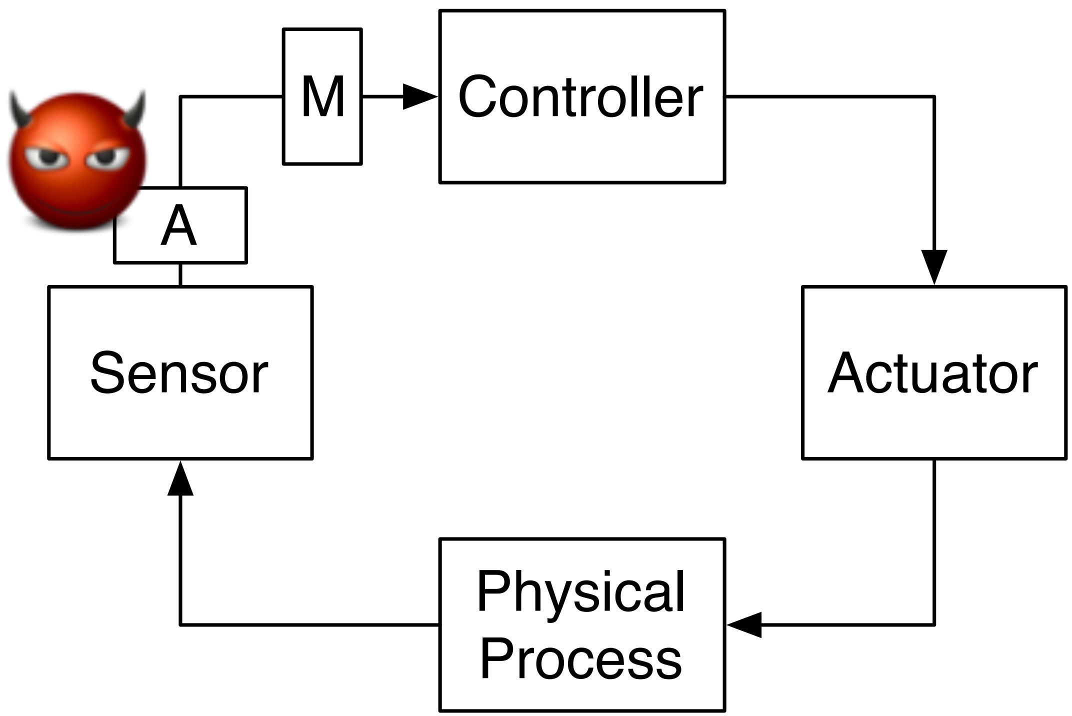
e.g., “Turn off pump to stop flow if tank level too high”

# Safety Monitor




Safety method in traditional ICS:
Expect random failures in sensors & actuators
Monitor for irregular behavior & alert operator
e.g., ”If water is flowing into tank, its level should rise”

# Safety vs Security




Traditional ICS retrofitted with modern network technology

**Different failure models in security!**

e.g., multiple sensors compromised at a time (not random)

Drop/inject/modify packets, bypass detection by monitor

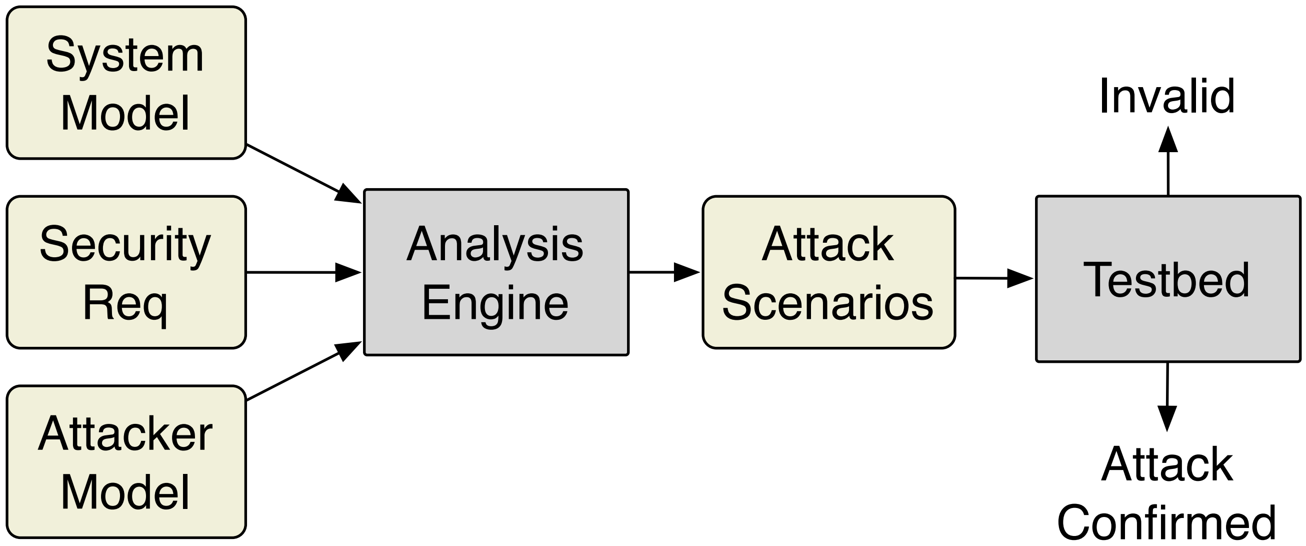
# Ingredients of Security Analysis

- Security requirement
  - **Integrity**: Information presented to the operator accurately reflects the status of the plant.
- Threat model
  - Has access to the building; intent to physically damage the plant to interrupt its operations
- Attack surface
  - Wireless network; open to eavesdrop & packet injection
- Protection mechanisms
  - Safety monitor to detect unusual water properties & tank levels

# Security Analysis

Does the safety monitor (**protection mechanism**) ensure accurate transmission of the plant status to the operator (**integrity requirement**) even when an intruder (**threat**) attempts to sabotage the plant through the wireless network (**attack surface**)?

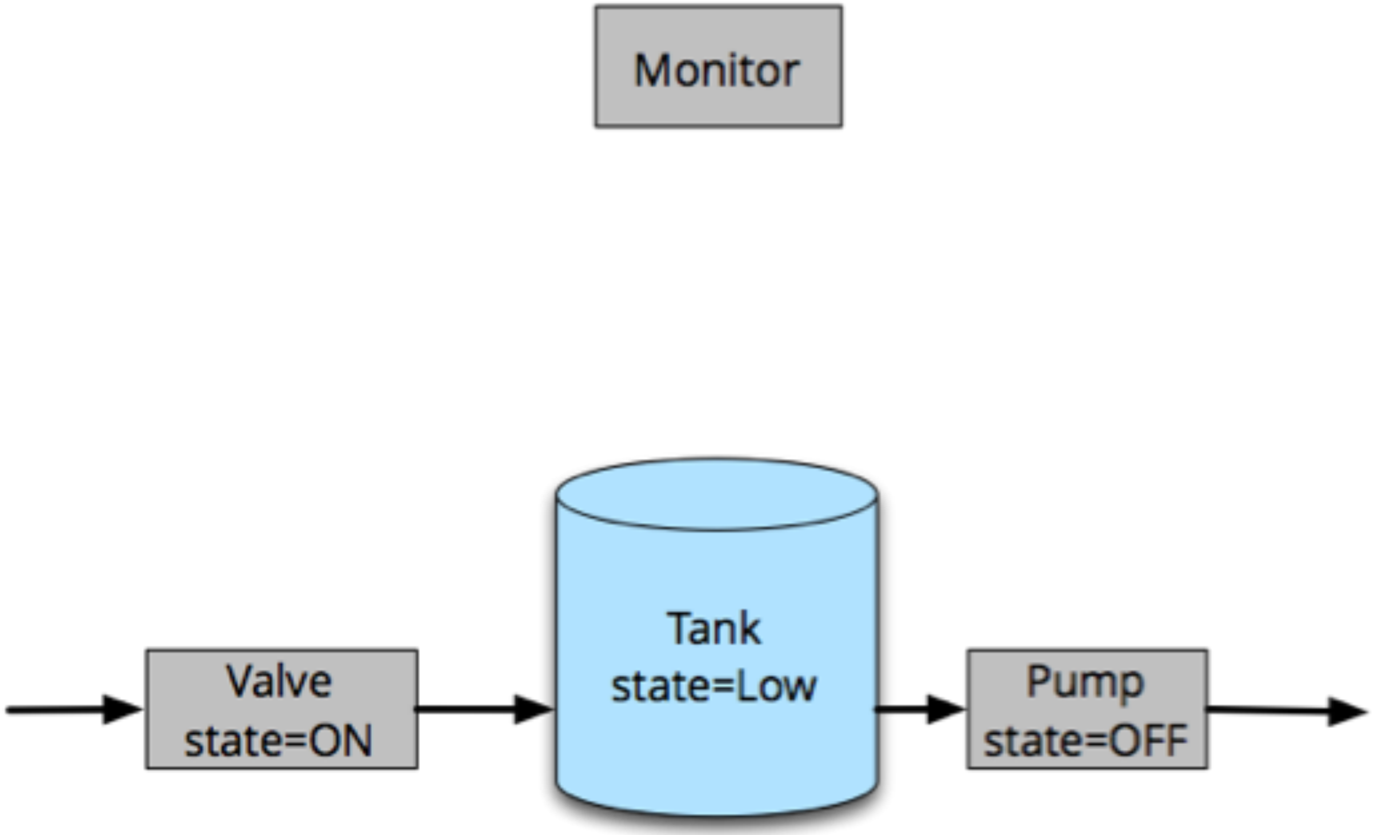
# Automating Security Analysis



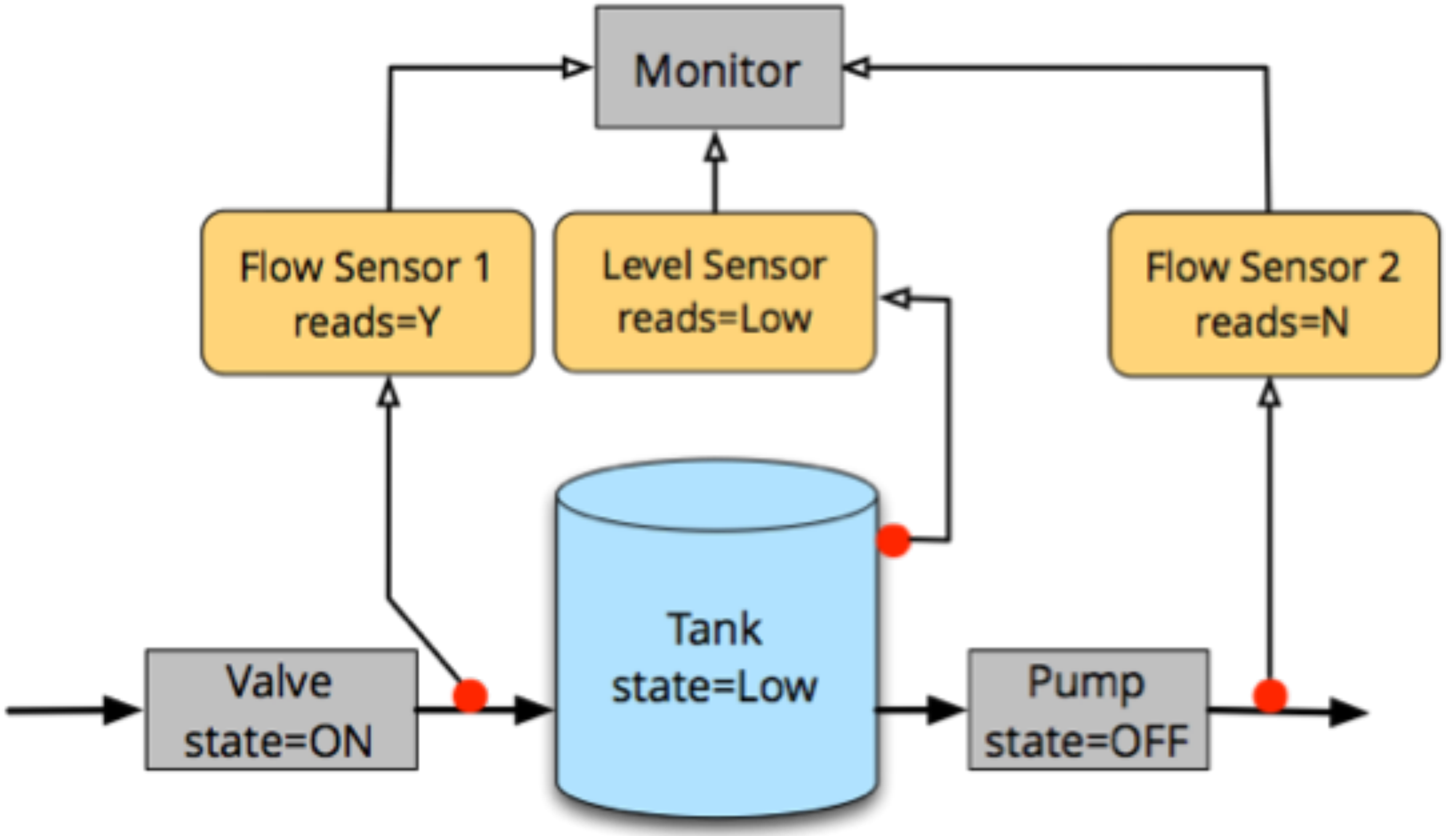

Build formal models of system & attacker (e.g., state machines)
Specify security req. using a formal notation (e.g., temporal logic)
Analyzer exhaustively explores all possible sequences of attacker actions (model checking)

Risk: But what if the models aren't accurate?

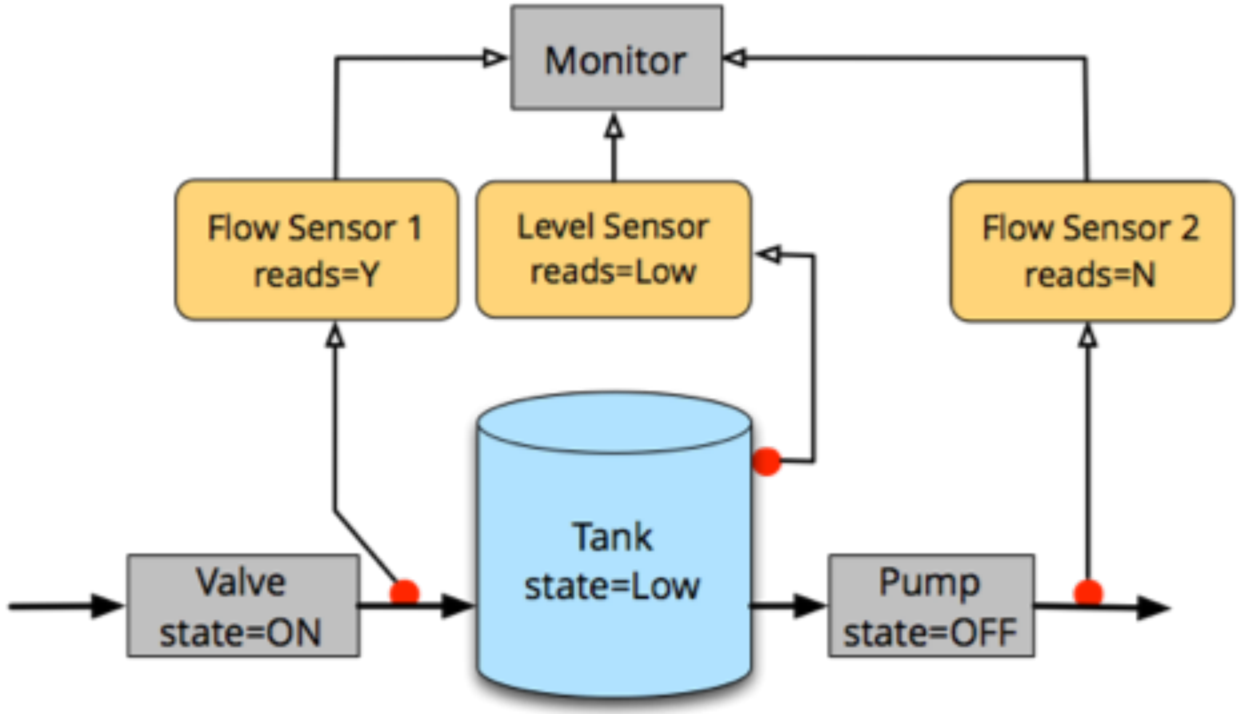
# Example Attack Scenario



# Example Attack Scenario

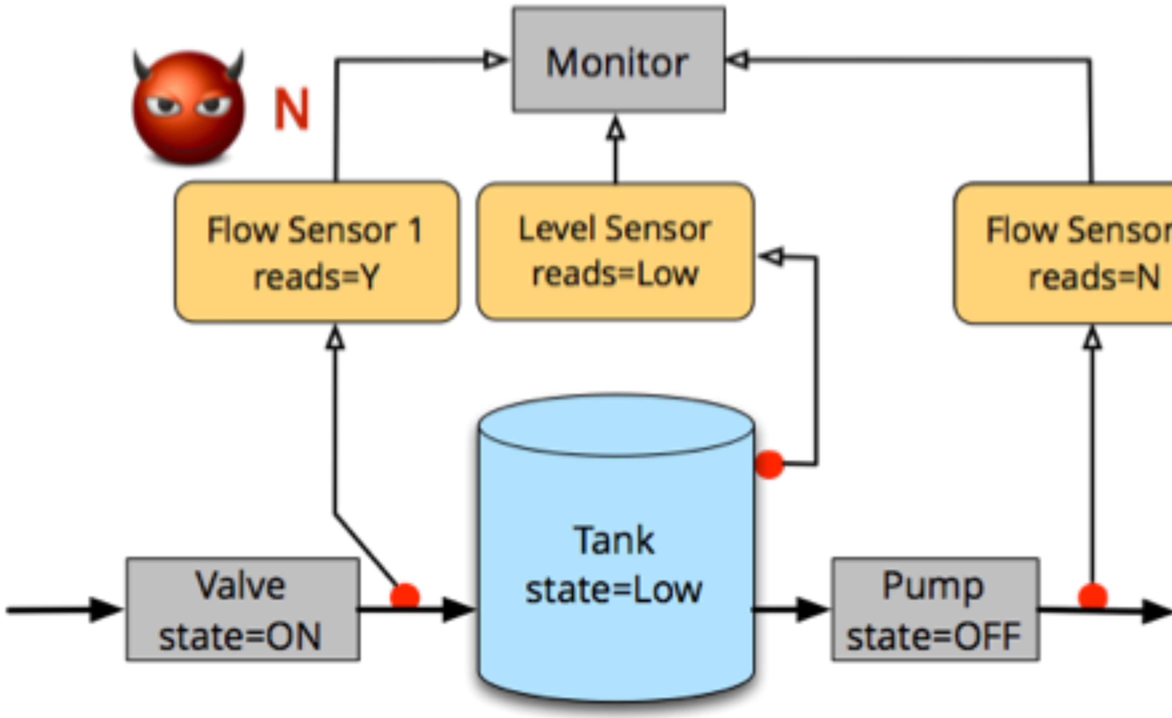


# Example Attack Scenario




t = ⟨ L=Low, F1=Y

# Example Attack Scenario

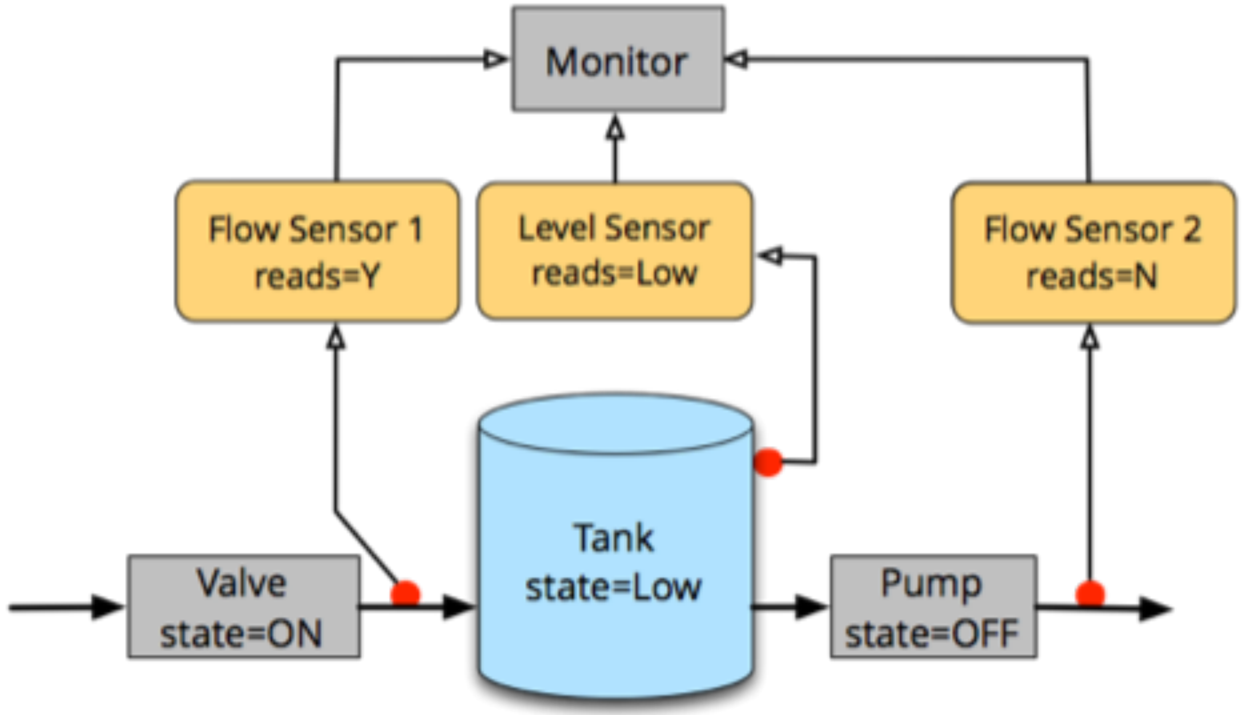



t = < L=Low, F1=Y

t'= < L=Low, F1=N

t = actual sensor readings; t′ = readings seen by the monitor

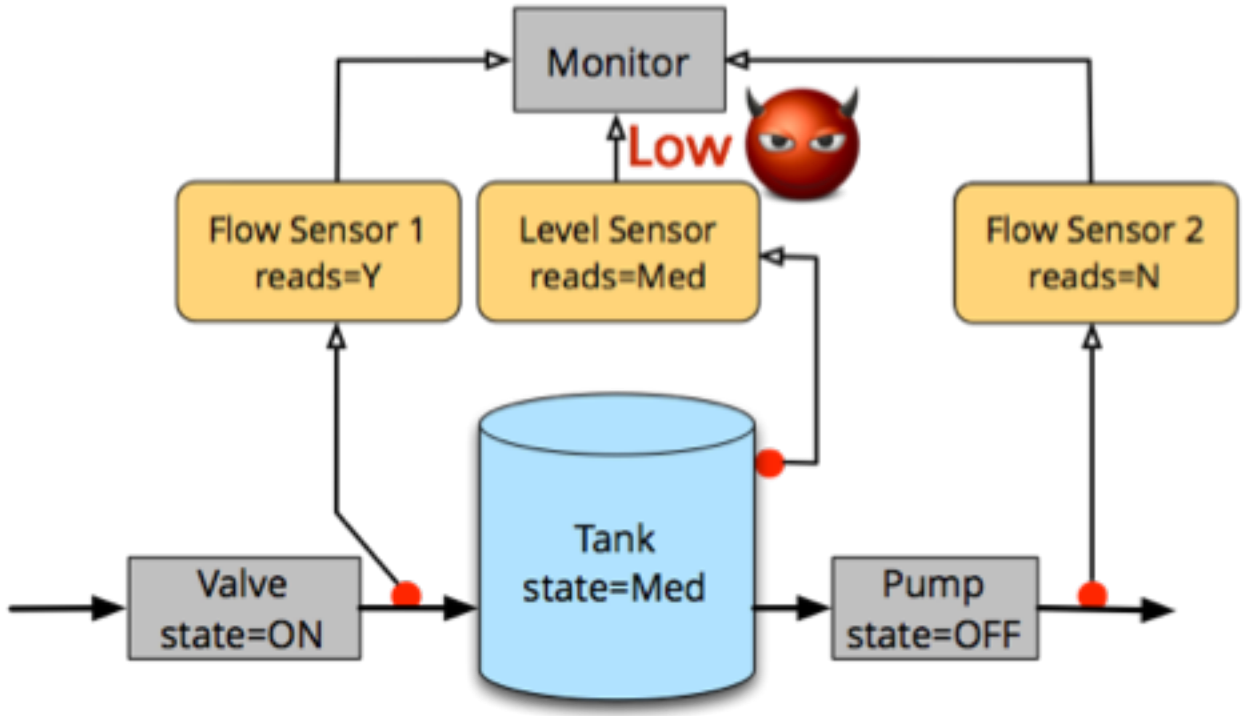
# Example Attack Scenario




t = < L=Low, F1=Y, F2=N

t'= < L=Low, F1=**N**, F2=N

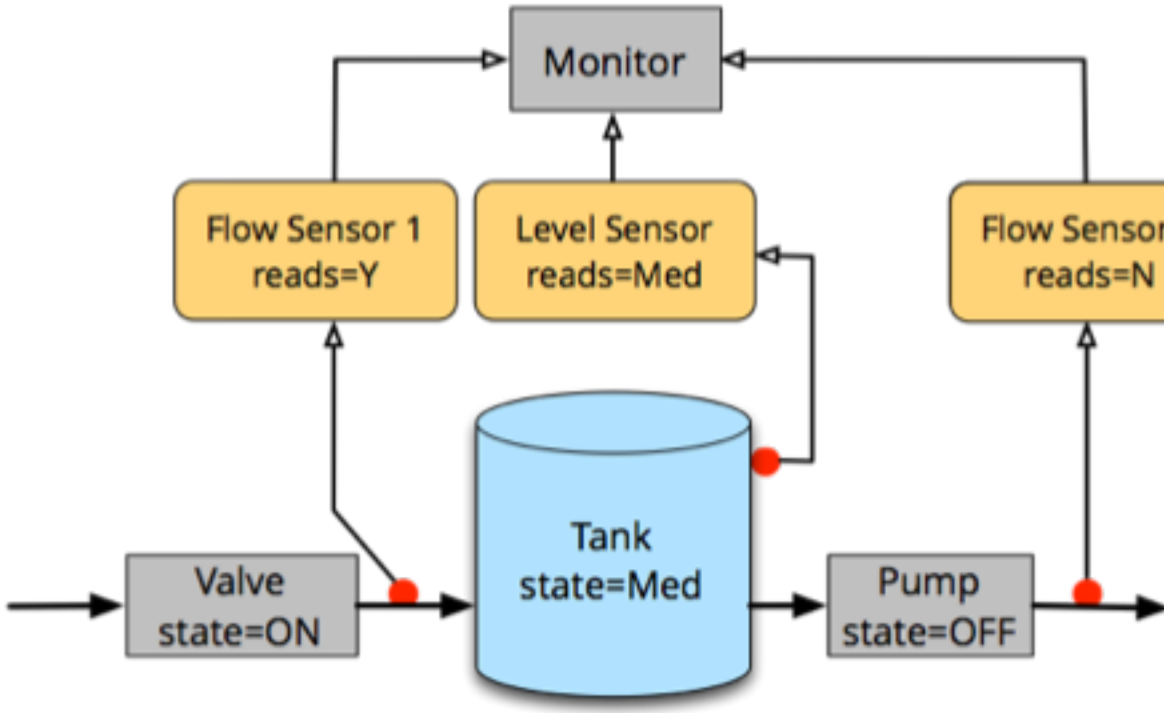
# Example Attack Scenario




t = 〈 L=Low, F1=Y, F2=N, L=Med

t'= 〈 L=Low, F1=N, F2=N, L=Low

# Example Attack Scenario



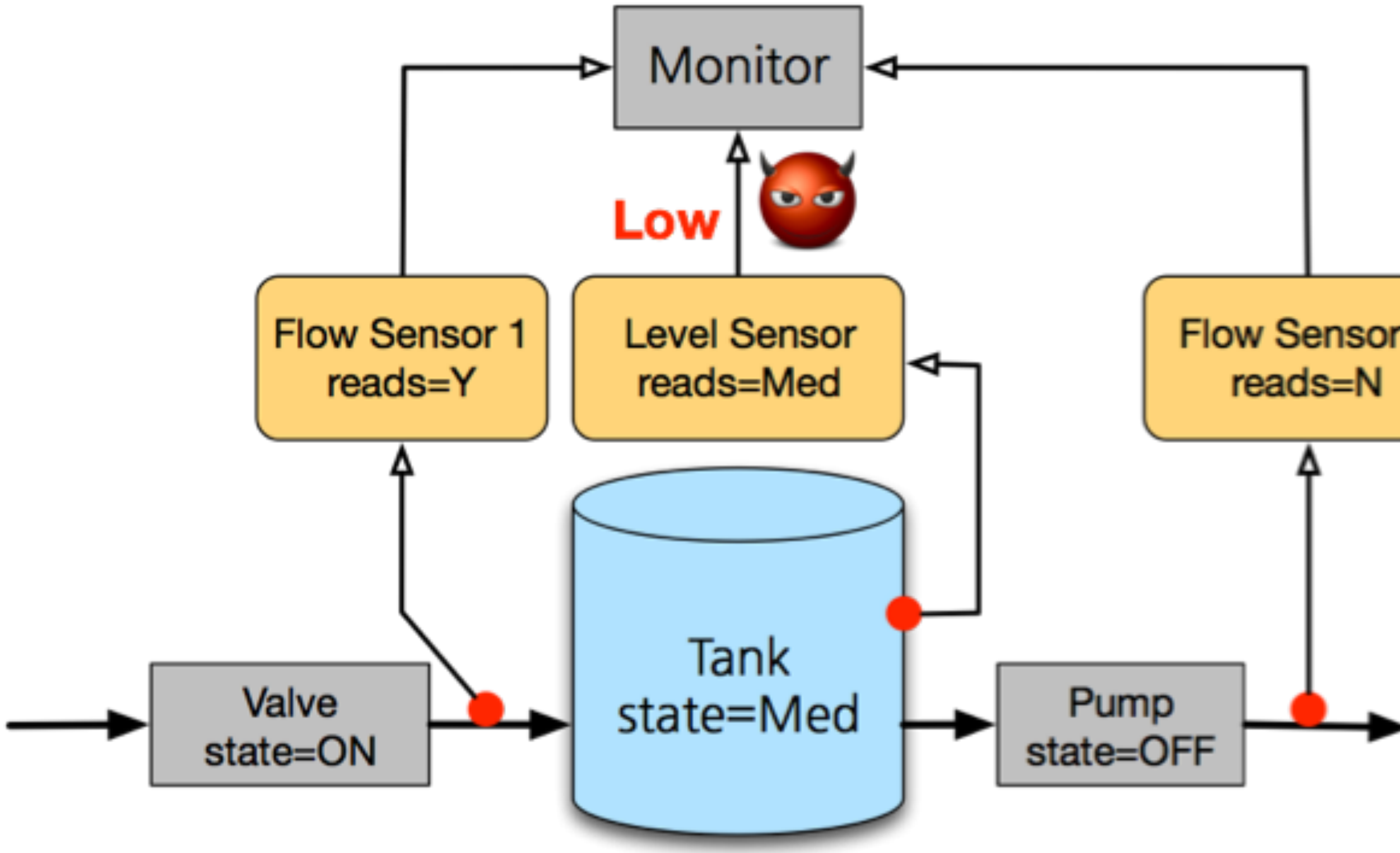

t = ⟨ L=Low, F1=Y, F2=N, L=Med, F1=Y, F2=N ⟩

↓ ↓

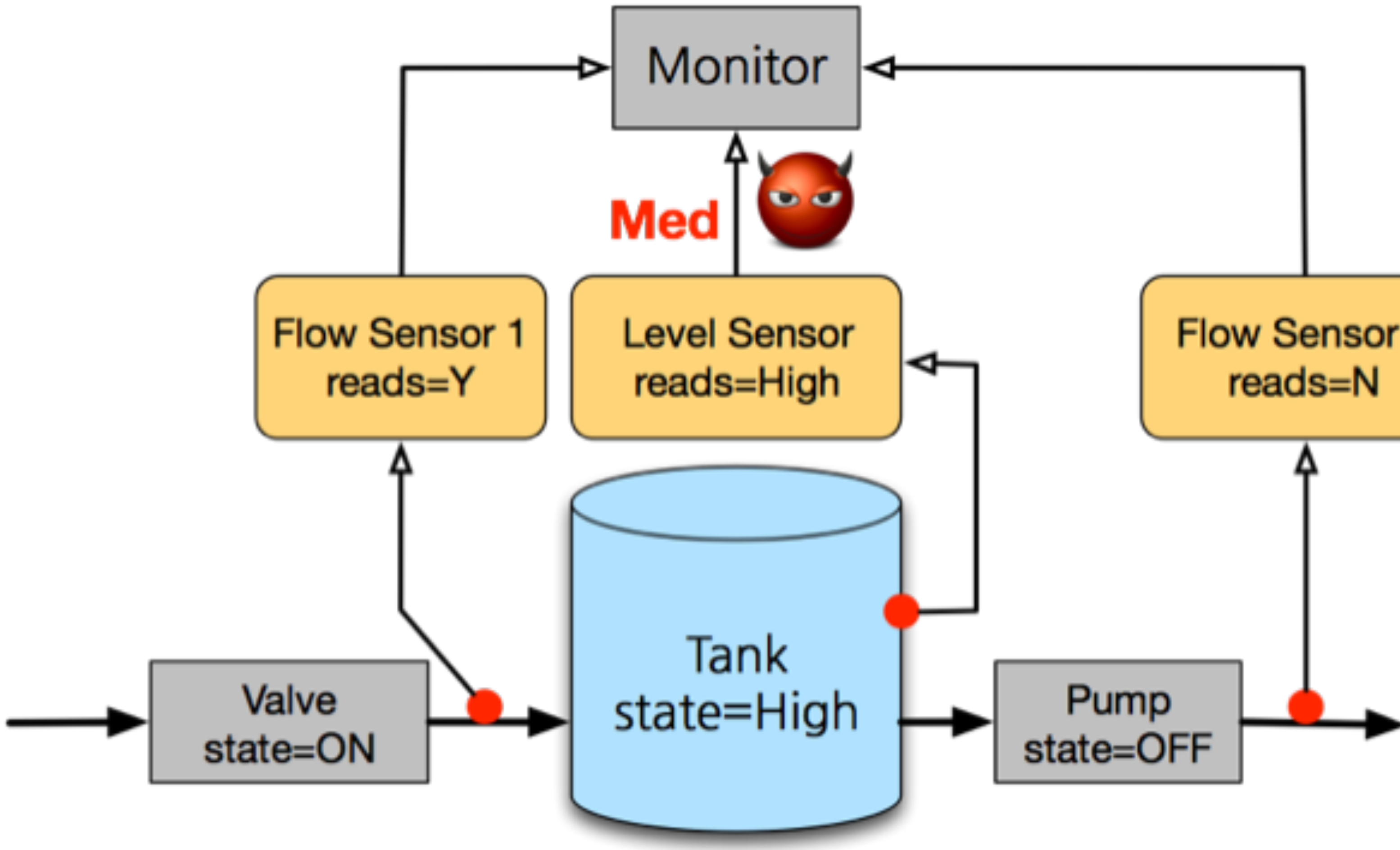
t'= ⟨ L=Low, F1=N, F2=N, L=Low, F1=Y, F2=N ⟩

**Monitor believes everything is OK!**

# Example Attack Scenario



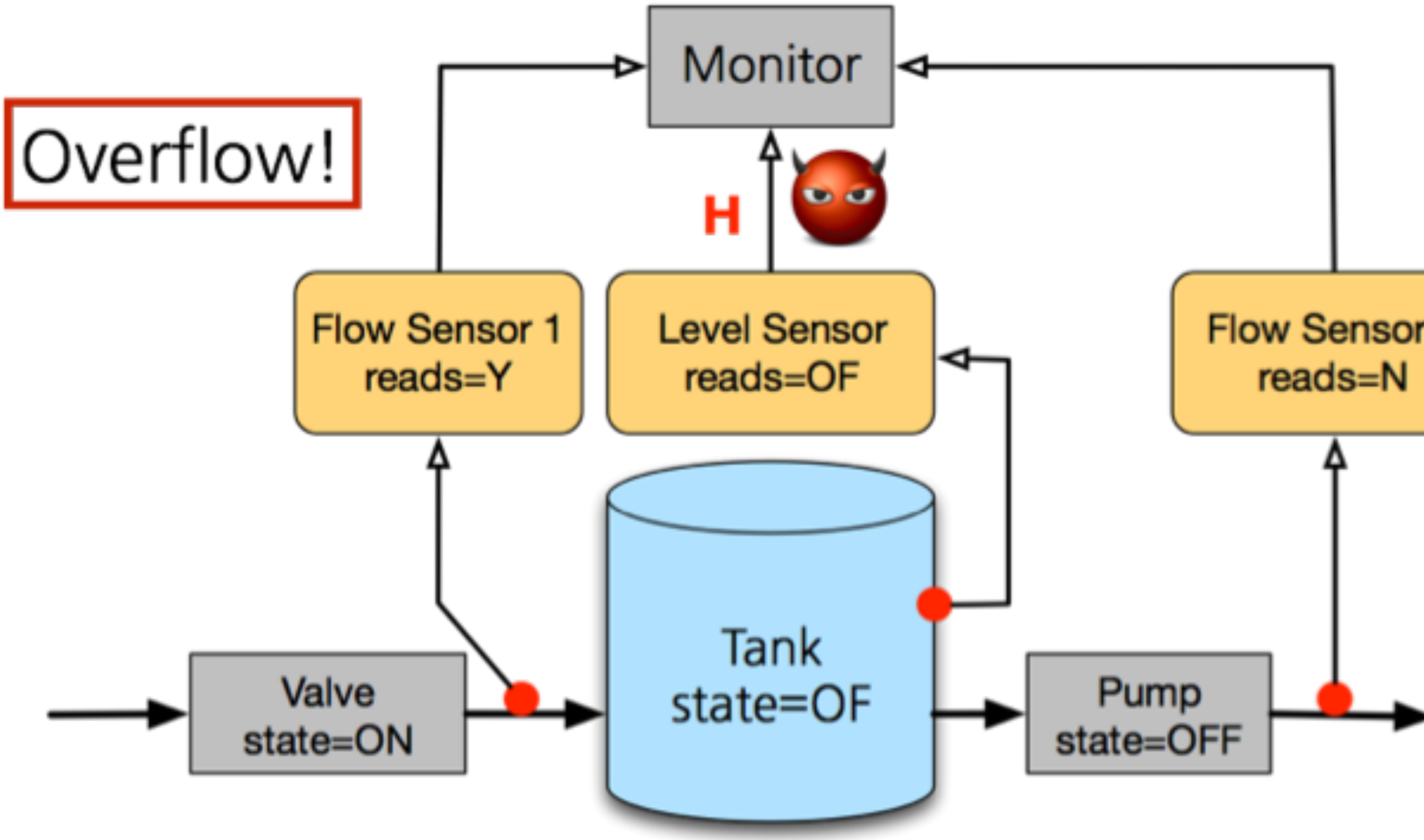
# Example Attack Scenario




**Water level high: Flow must be stopped**

But monitor fails to act, since it believes plant status is OK

# Example Attack Scenario



# Lessons

- New environment, new threats
  - Legacy ICS: Isolated, mostly physical failures
  - Modern cyber-physical system (CPS): Connected to the web, diverse threat models (e.g., Stuxnet)
  - Traditional safety methods are insufficient!
  - Ideally, redesign the system with security as a goal (but difficult to do in general)
- Analysis
  - Recent development in formal techniques for automated analysis & attack generation
  - But must still get the system & threat models right!

# What I haven't talked about today

- Protection mechanisms
  - Access control, capability-based models
  - Information flow control
- Human factors
  - Often the weakest link in the design!
  - Include users & operators as part of requirements elicitation & environment model
  - Clearly define user roles & their privileges
  - Treat all user inputs as potentially malicious

# Summary: Strategies for Secure Design

- Security requirements
  - Elicitation & precise documentation
- Threat model
  - Principle of least privilege: Assume the worst
- Attack surface
  - Isolation: Separate the critical components
- Protection mechanisms
  - Defense in depth: Mitigate the weakest link