

# Safety Critical Systems Design:

Patterns and Practices for  
Designing Mission and Safety-  
Critical Systems\*

**Bruce Powel Douglass, Ph.D.**

**bpd@ilogix.com**

***Chief Evangelist, I-Logix***

\* Portions adopted from the author's book *Doing Hard Time: Developing Real-Time Systems with UML, Objects, Frameworks, and Patterns*, Addison-Wesley Publishing, 1999.

**I-Logix**

# Agenda

- Basic Safety Concepts
- Elements of Safety Designs
- How can the UML Help?
  - Safety Architectures in UML
  - Safety Qualities of Service in UML

# Why Care About Safety?

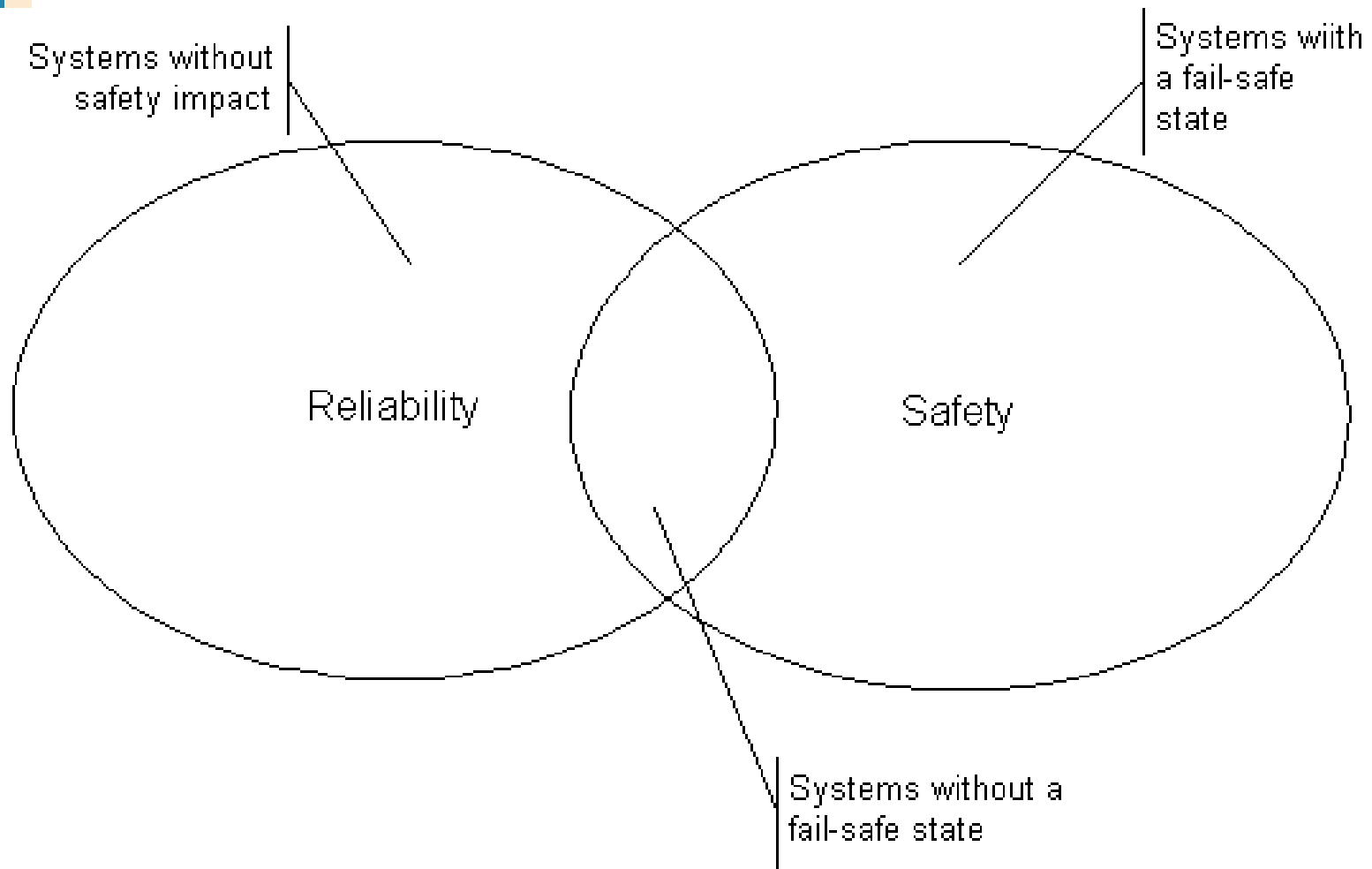
- Safety is not discussed in the literature
- Safety is not taught in the colleges
- Yet without training or guidance,

*Embedded systems are assuming more safety roles every day.*

# What is Safety?

- Safety is *freedom from accidents or losses*.
- Safety is not reliability!
  - Reliability is *the probability that a system will perform its intended function satisfactorily*.
- Safety is not security!
  - Security is *protection or defense against attack, interference, or espionage*.

# Safety is not Reliability!



# Safety-Related Concepts

- *Accident* is a loss of some kind, such as injury, death, or equipment damage
- *Risk* is a combination of the likelihood of an accident and its severity:  
$$risk = p(a) * s(a)$$
- *Hazard* is a set of conditions and/or events that leads to an accident.

# Safety-Related Concepts

- A *failure* is the nonperformance of a system or component, a random fault
  - A random failure is one that can be estimated from a pdf,
  - *Failures are events*
  - e.g., a component failure
- An *error* is a systematic fault
  - A systematic fault is an design error
  - *Errors are states or conditions*
  - e.g., a software bug
- A *fault* is *either* a failure or an error

# Safety-Related Concepts

- Safety must be considered in the context of the system, not the component or the software
- It is less expensive and far more effective to *build in safety early than try to tack it on later*
- The *Hazard Analysis* ties together hazards, faults, and safety measures

# ROPES Process: Eight Steps to Safety

1. Identify the Hazards
2. Determine the Risks
3. Define the Safety Measures
4. Create Safe Requirements
5. Create Safe Designs
6. Implement Safety
7. Assure the Safety Process
8. Test, Test, Test

# Eight Steps to Safety

1. Identify the Hazards
2. Determine the Risks *Safety Analysis*
3. Define the Safety Measures
4. Create Safe Requirements
5. Create Safe Designs
6. Implement Safety
7. Assure the Safety Process
8. Test, Test, Test

# Safety Analysis

- You must identify the hazards of the system
- You must identify faults that can lead to hazards
- You must define safety control measures to handle hazards
- These culminate in the Hazard Analysis
- The Hazard Analysis feeds into the Requirements Specification

# Eight Steps to Safety

1. Identify the Hazards

2. Determine the Risks

3. Define the Safety Measures

4. Create Safe Requirements

5. Create Safe Designs

6. Implement Safety

7. Assure the Safety Process

8. Test, Test, Test

# Hazard Causes

- Release of Energy
- Release of Toxins
- Interference with life support or other safety-related function
- Misleading safety personnel
- Failure to alarm

# Types of Hazards

- **Actions**
  - inappropriate system actions taken
  - appropriate system actions not taken
- **Timing**
  - too soon
  - too late
- **Sequence**
  - skipping actions
  - actions out of order
- **Amount**
  - too much
  - too little

# Means of Hazard Control

- Obviation
- Education
- Alarming
- Active correction
- Interlock
- Safety equipment (goggles, gloves)
- Restrict access
- Labeling
- Fail-Safe

# Hazard Analysis

How long is the exposure to hazard?

What do you do about it?

How long to discover?

How Frequently?

How can this happen?

How long can it be tolerated?

How bad if it occurs?

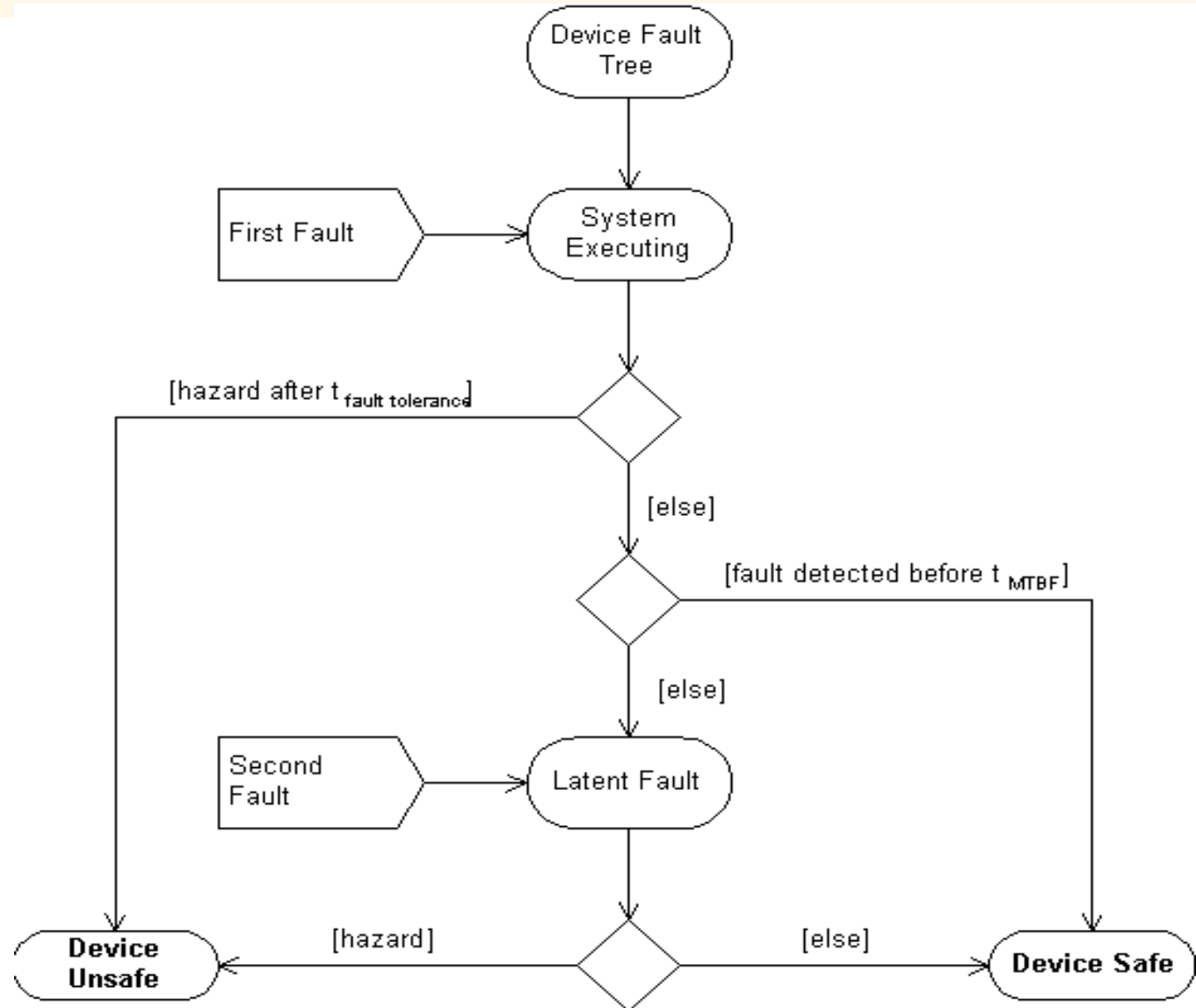
Hazardous Condition

Hazard	Level of Risk	Tolerance Time $T_1$	Fault	Likelihood	Detection Time	Control Measure	Exposure Time
Hypo-ventilation	Severe	5 min	Ventilator Fails	rare	30 sec	Independent pressure alarm, action by doctor	1 min
			Esophageal Intubation	often	30 sec	CO <sub>2</sub> sensor alarm	1 min
			User misattaches breathing circuit	often	0	Noncompatible mechanical fasteners used	0
Overpressure	Severe	250 ms	Release valve failure	rare	50 ms	Secondary valve opens	55 ms

# When is a System Safe Enough?

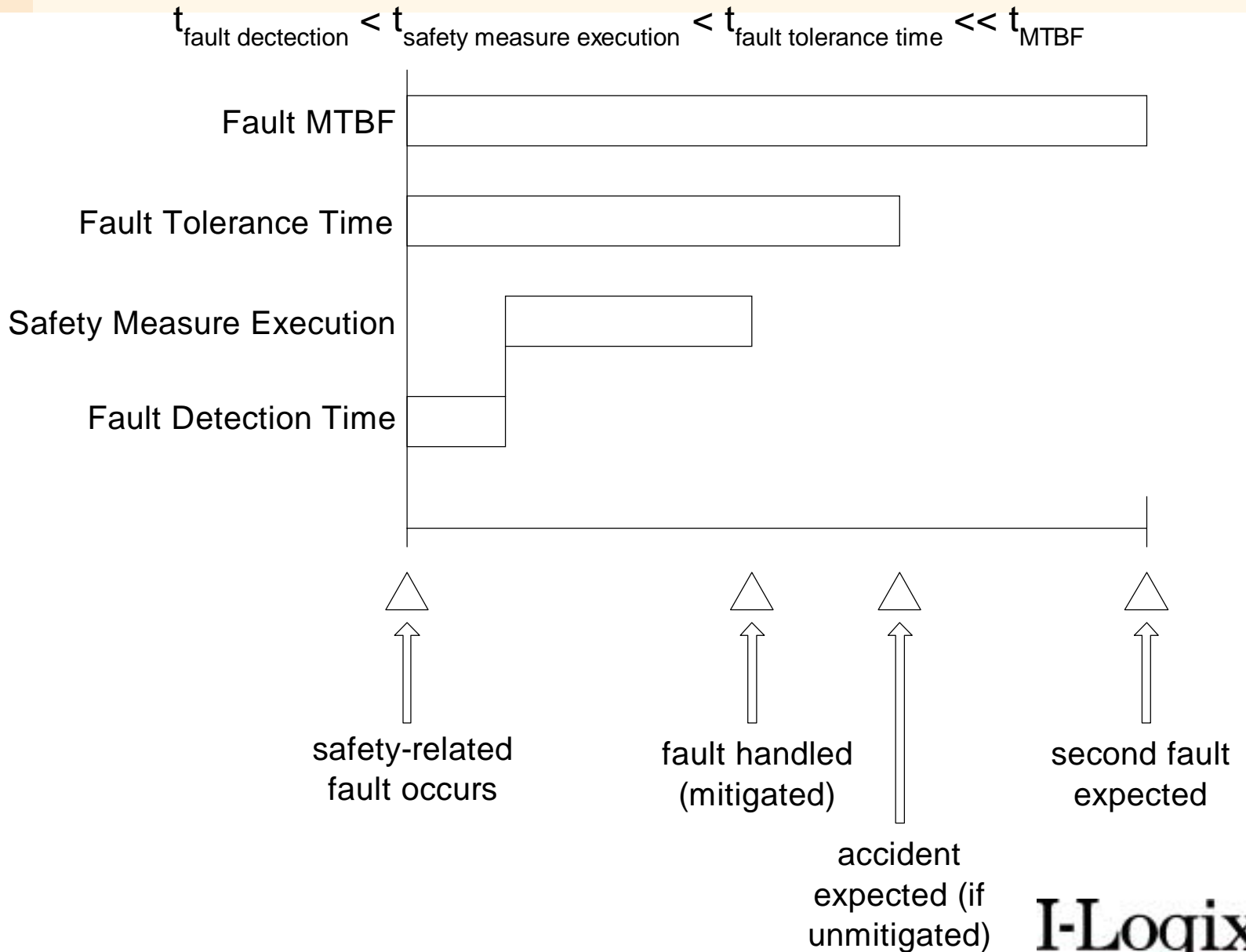
- (Minimal) No hazards in the absence of faults
- (Minimal) No hazards in the presence of any single point failure
  - A **common mode failure** is a single point failure that affects multiple channels
  - A **latent fault** is an undetected fault which allows another fault to cause a hazard
- Your mileage may vary depending on the risk introduced by your system

# TUV Single Fault Assessment



From VDE 0801: Principles of Computers in Safety-Related Systems

# Safety Fault Timeline



# Fail-Safe States

- Off
  - Emergency stop -- immediately cut power
  - Production stop -- stop after current task
  - Protection stop -- shut down without removing power
- Partial Shutdown
  - Degraded level of functionality
- Hold
  - No functionality, but with safety actions taken
- Manual or External Control
- Restart

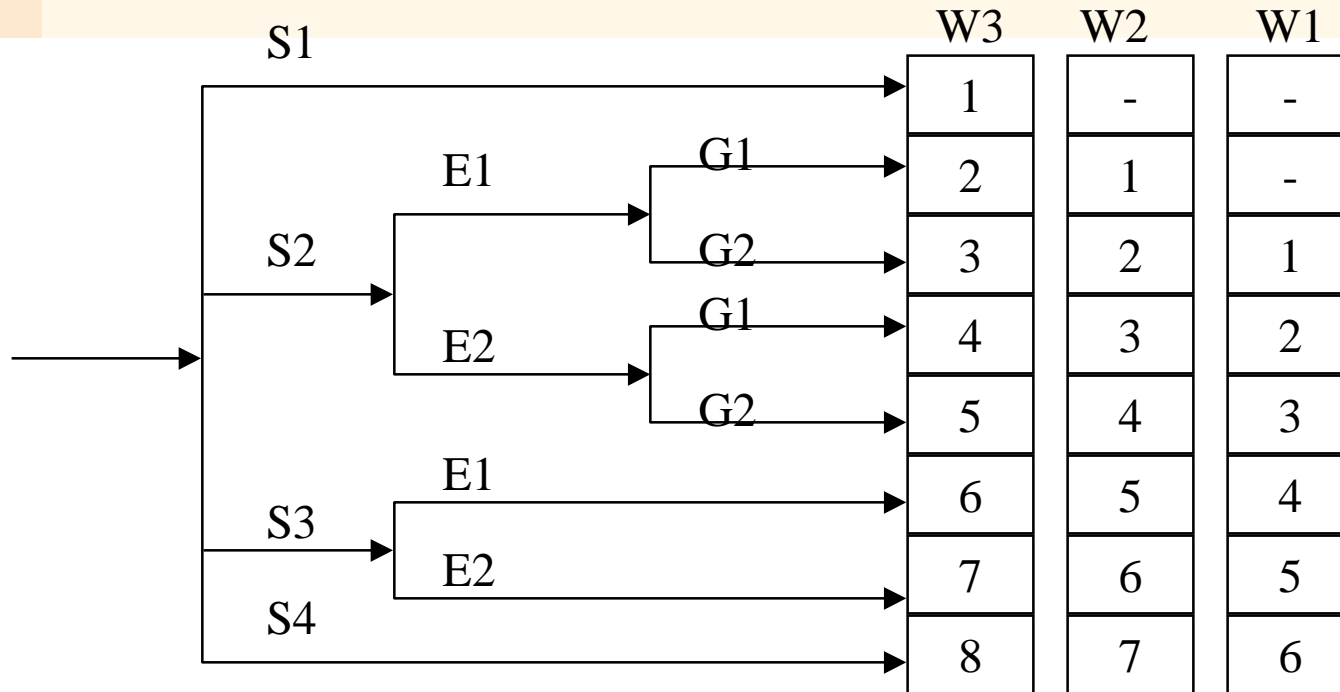
# Eight Steps to Safety

1. Identify the Hazards
2. Determine the Risks
3. Define the Safety Measures
4. Create Safe Requirements
5. Create Safe Designs
6. Implement Safety
7. Assure the Safety Process
8. Test, Test, Test

# Risk Assessment

- For each hazard
  - Determine the potential severity
  - Determine the likelihood of the hazard
  - Determine how long the user is exposed to the hazard
  - Determine whether the risk can be removed

# TUV Risk Level Determination Chart \*



Risk Parameters:

S: Extent of Damage

S1: Slight injury

S2: Severe irreversible injury to one or more persons or the death of a single person

S3: Death of several persons

S4: Catastrophic consequences, several deaths

E: Exposure Time

E1: Seldom to relatively infrequent

E2: Frequent to continuous

G: Hazard Prevention

G1: Possible under certain conditions

G2: Hardly possible

W: Occurrence Probability of Hazardous Event

W1: Very Low

W2: Low

W3: Relatively High

\*adapted from DIN V 19250

# Sample Risk Assessments

<b>Device</b>	<b>Hazard</b>	<b>Extent of Damage</b>	<b>Exposure Time</b>	<b>Hazard Prevention</b>	<b>Probability</b>	<b>TUV Risk Level</b>
<b>Microwave oven</b>	<b>Irradiation</b>	<b>S2</b>	<b>E2</b>	<b>G2</b>	<b>W3</b>	<b>5</b>
<b>Pacemaker</b>	<b>Pace too slowly</b>	<b>S2</b>	<b>E2</b>	<b>G2</b>	<b>W3</b>	<b>5</b>
	<b>Pace too fast</b>	<b>S2</b>	<b>E2</b>	<b>G2</b>	<b>W3</b>	<b>5</b>
<b>Power Station Burner</b>	<b>Explosion</b>	<b>S3</b>	<b>E1</b>	<b>--</b>	<b>W3</b>	<b>6</b>
<b>Airliner</b>	<b>Crash</b>	<b>S4</b>	<b>E2</b>	<b>G2</b>	<b>W2</b>	<b>8</b>

# Eight Steps to Safety

1. Identify the Hazards
2. Determine the Risks
3. Define the Safety Measures
4. Create Safe Requirements
5. Create Safe Designs
6. Implement Safety
7. Assure the Safety Process
8. Test, Test, Test

# Safety Measures

- Safety measures do one of the following
  - Remove the hazard
  - Reduce the risk
  - Identify the hazard to supervisory personnel
- The purpose of the safety measure is to ensure the system remains in a safe state

# Risk Reduction

- Identify the fault
- Take corrective action, either
  - Use redundancy to correct and move on
    - feedforward error correction
  - Redo the computational step
    - feedback error detection
  - Go to a fail-safe state

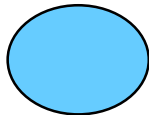
# Fault Identification at Run-time

- Faults must be identified (and handled) in  $< T_{\text{fault tolerance}}$
- Fault identification requires redundancy
- Redundancy can be in terms of
  - channel } *Architectural*
  - device }
  - data } *Detailed Design*
  - control }
- Redundancy may be either
  - Homogenous (random faults only)
  - Heterogeneous (systematic and random faults)

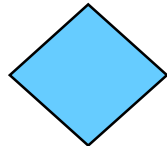
# Fault Tree Analysis Symbology



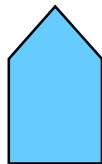
An event that results from a combination of events through a logic gate



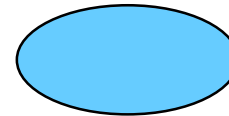
A basic fault event that requires no further development



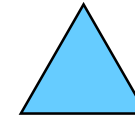
A fault event because the event is inconsequential or the necessary information is not available



An event that is expected to occur normally



A condition that must be present to produce the output of a gate



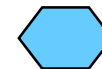
Transfer



AND gate

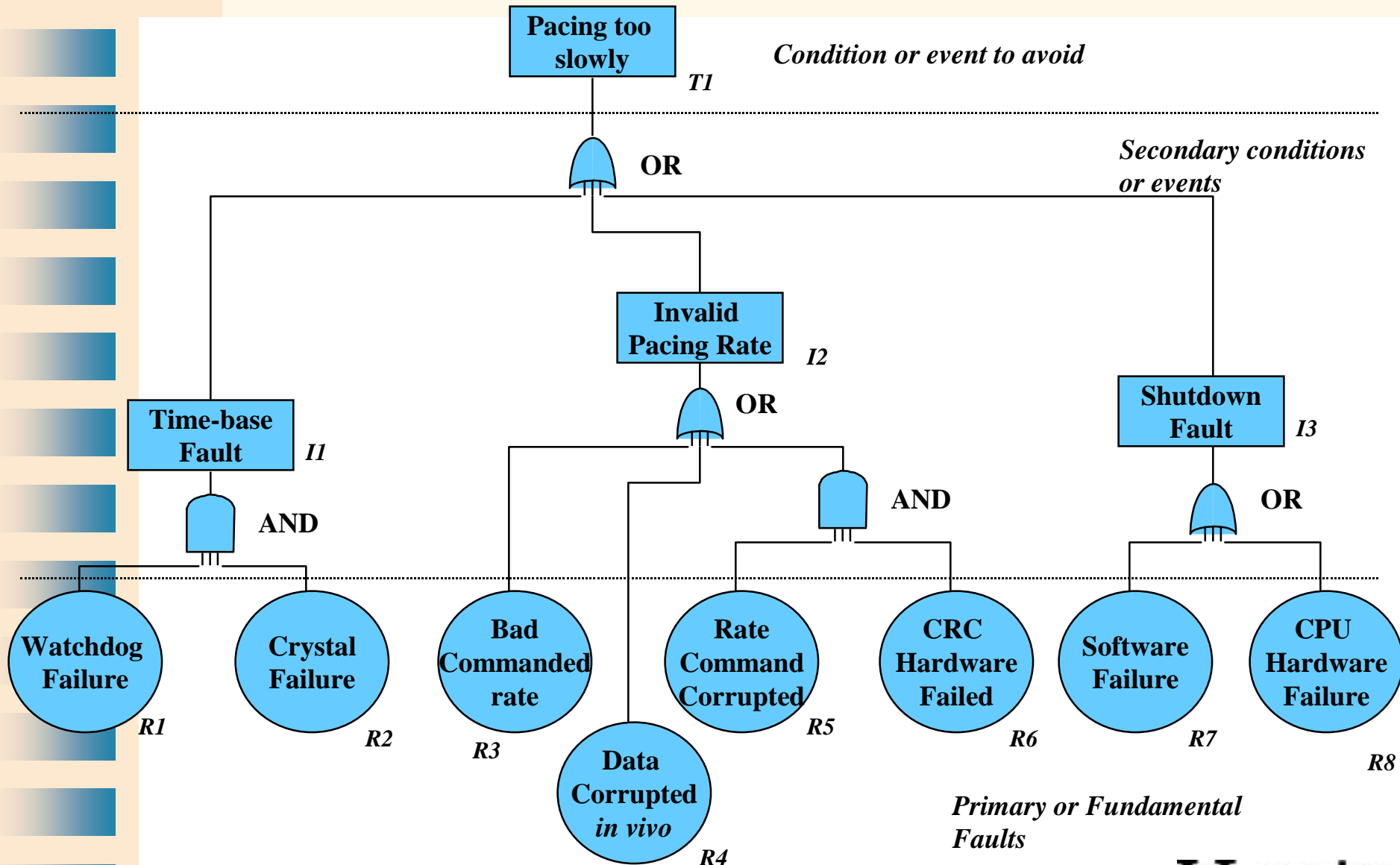


OR Gate



NOT Gate

# Subset of Pacemaker Fault Analysis



# Eight Steps to Safety

1. Identify the Hazards
2. Determine the Risks
3. Define the Safety Measures
4. Create Safe Requirements
5. Create Safe Designs
6. Implement Safety
7. Assure the Safety Process
8. Test, Test, Test

# Safe Requirements

- Requirements specification follows initial hazard analysis
- Specific requirements should track back to hazard analysis
- Architectural framework should be selected with safety needs in mind

# Eight Steps to Safety

1. Identify the Hazards
2. Determine the Risks
3. Define the Safety Measures
4. Create Safe Requirements
5. Create Safe Designs
6. Implement Safety
7. Assure the Safety Process
8. Test, Test, Test

# Isolate Safety Functions

- Safety-relevant systems are 300-1000% more effort to produce
- Isolation of safety systems allows more expedient development
- Care must be taken that the safety system is truly isolated so that a defect in the non-safety system cannot affect the safety system
  - Different processor
  - Different heavy-weight tasks (depends on OS)

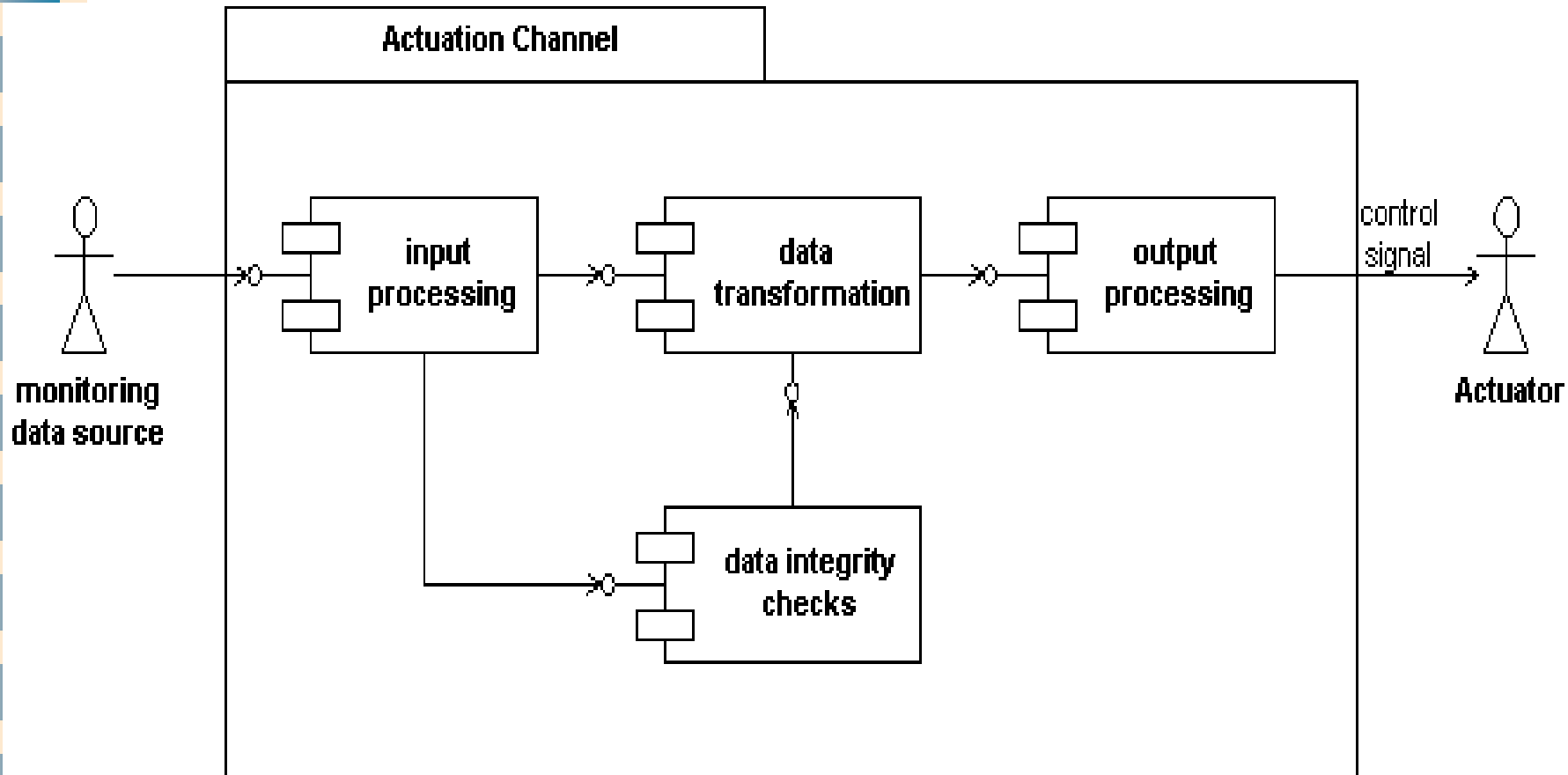
# Safety Architecture Patterns

- Protected Single-Channel Pattern
- Dual-Channel Patterns
  - Homogeneous Dual Channel Pattern
  - Heterogeneous Peer-Channel Pattern
  - Sanity Check Pattern
  - Actuator-Monitor Pattern
- Voting Multichannel Pattern

# Protected Single Channel Pattern

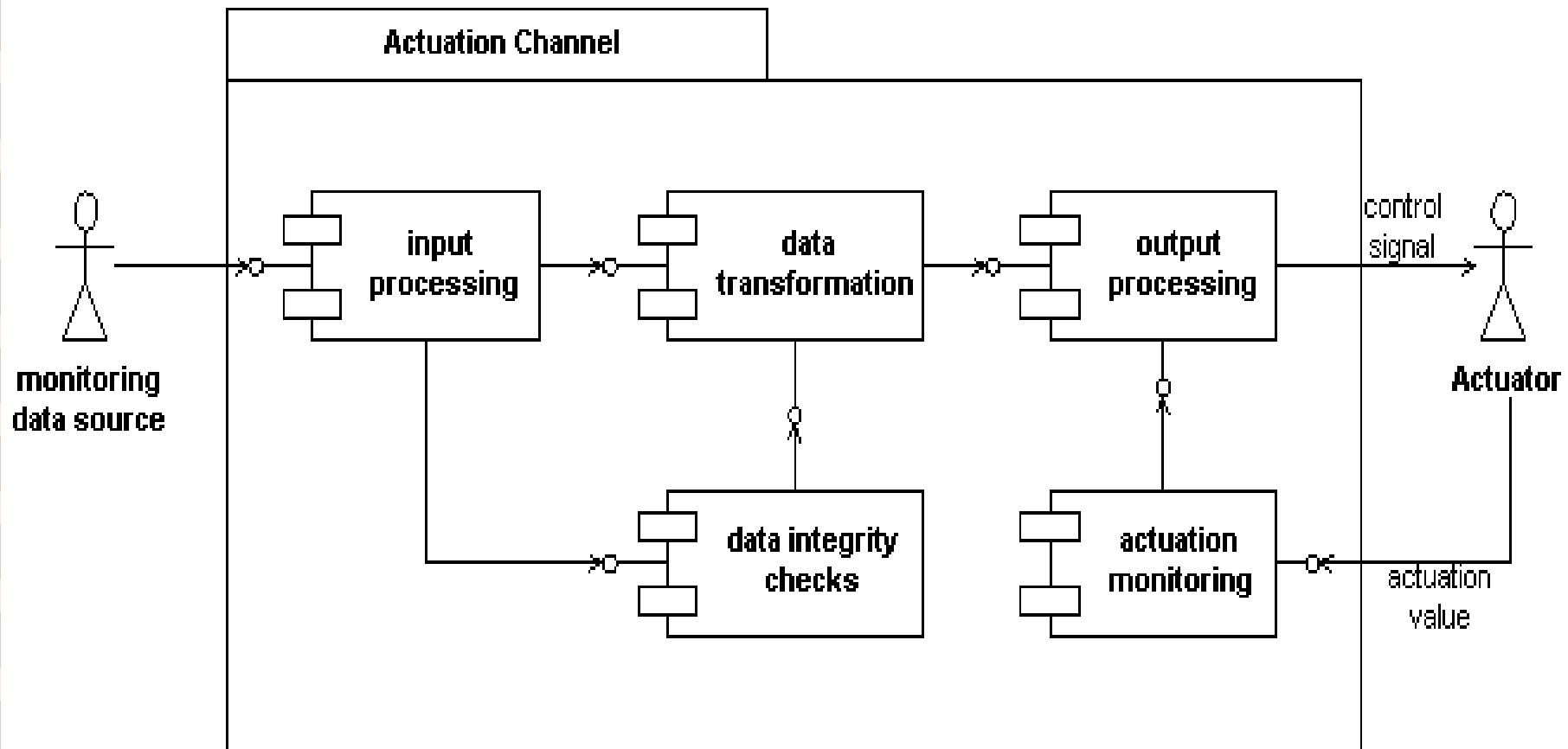
- Within the single channel, mechanisms exist to identify and handle faults
- All faults must be detected within the fault tolerance time
- May be impossible
  - To test for all faults within the fault tolerance time
  - To remove common mode failures from the single channel
- Generally,
  - Lower recurring system cost
  - Lower safety coverage
  - Cannot continue in the presence of a fault

# Single Channel Protected Architecture



Open Loop

# Single Channel Protected Architecture

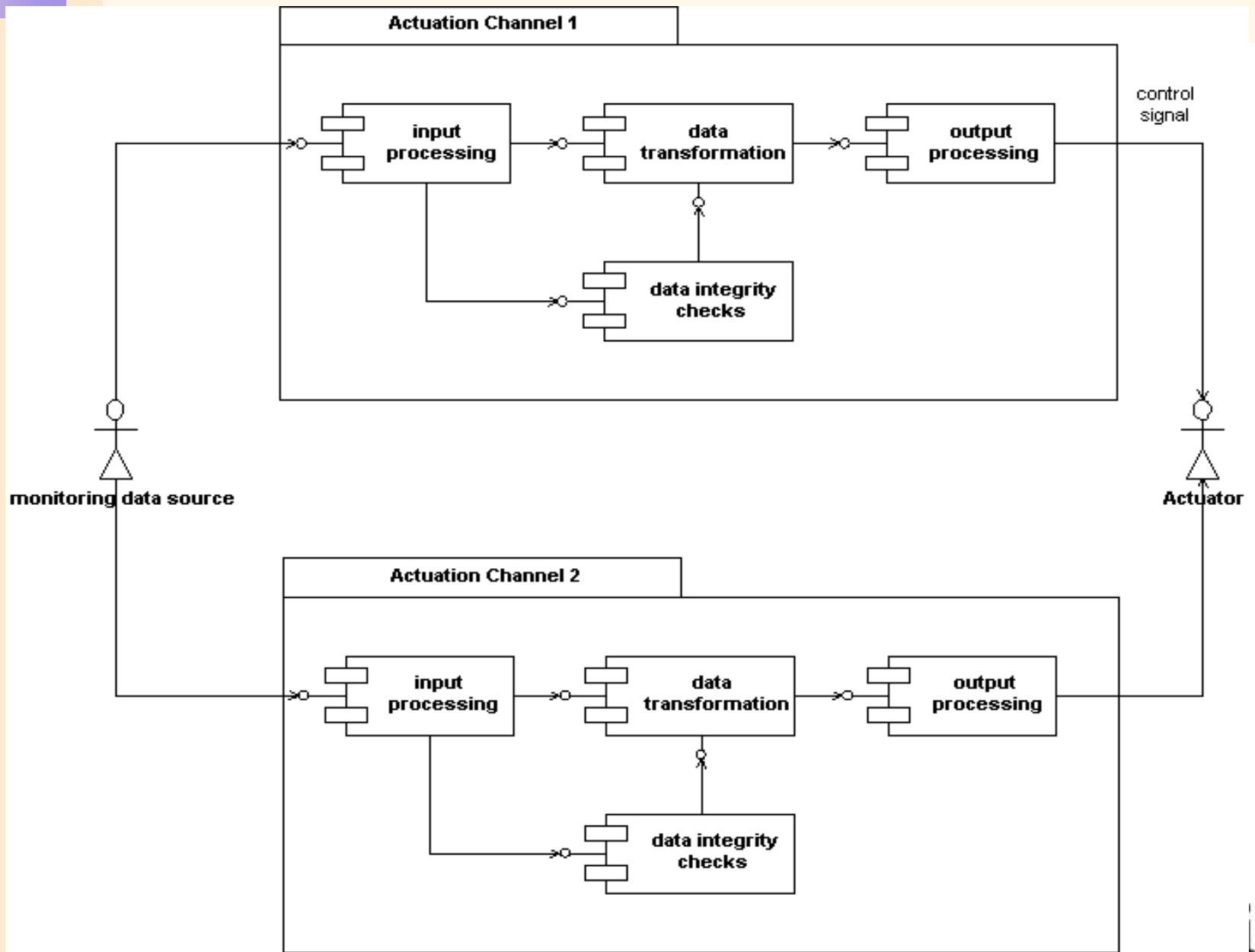


Closed Loop

# Dual Channel Architecture Patterns

- Separation of safety-relevant from nonsafety-relevant where possible
- Separation of monitoring from control
- Generally easier to meet safety requirements
  - Timing
  - Common mode failures
- Generally
  - Higher recurring system cost
  - Can continue in the presence of a fault

# Basic Dual-Channel Pattern



# Homogeneous Dual-Channel Pattern

- Identical channels used
- Channels may operate simultaneously (Multichannel Vote Pattern)
- Channels may operate in series (Backup Pattern)
- Good at identifying *random* faults but not *systematic* faults
- Low R&D cost, higher recurring cost

# Heterogeneous Peer-Channel Pattern

- Equal-weight, differently implemented channels
  - May use algorithmic inversion to recreate initial data
  - May use different algorithm
  - May use different teams (not fool-proof)
- Good at identifying both *random* and *systematic* faults
- Generally safest, but higher R&D and recurring cost

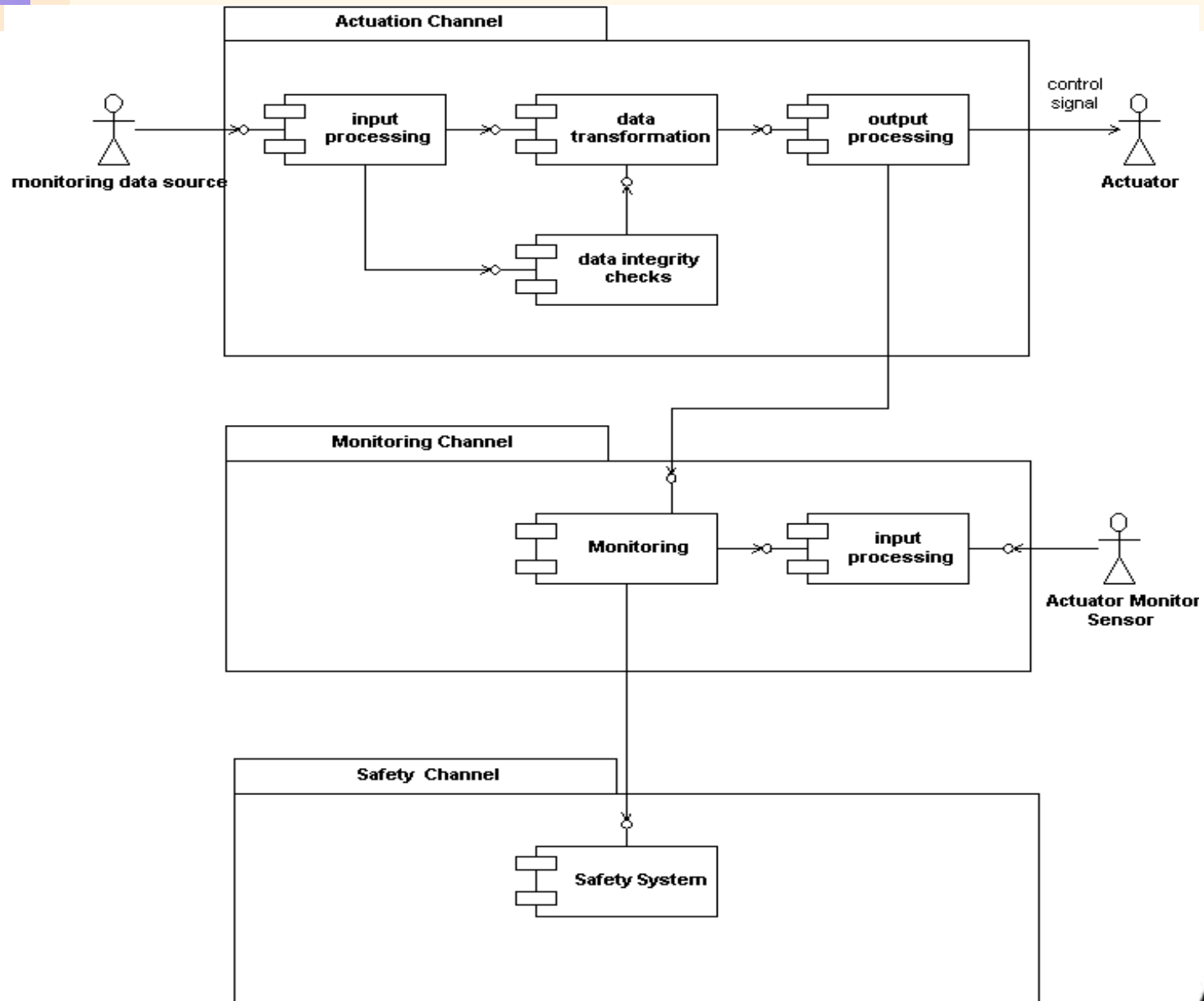
# Sanity Check Pattern

- A primary actuator channel does real computations
- A light-weight secondary channel checks the reasonableness of the primary channel
- Good for detection of both random and systematic faults
- May not detect faults which result in small variance
- Relatively inexpensive to implement, lower coverage, cannot continue in the presence of fault

# Monitor-Actuator Pattern

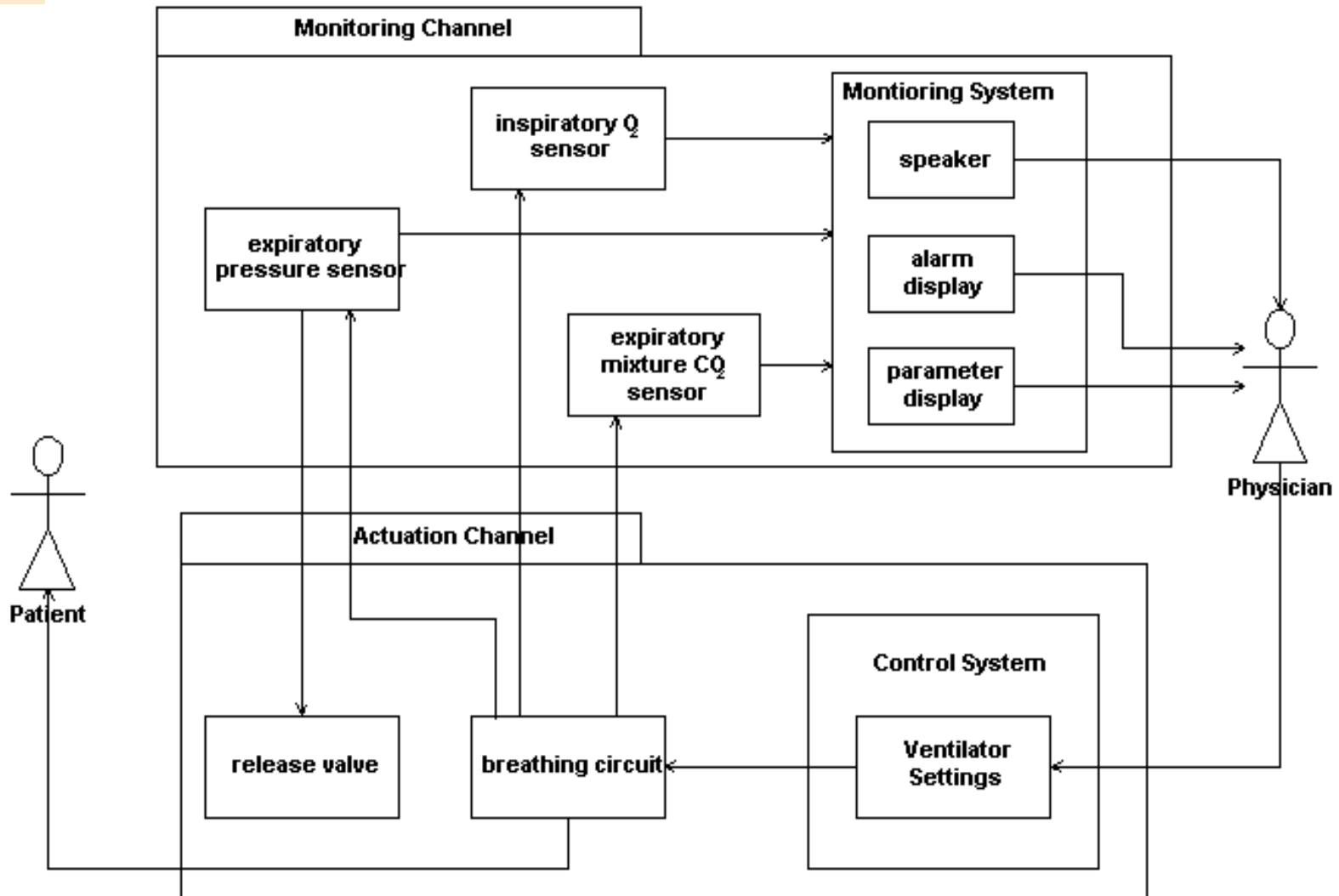
- Separates actuation from the monitoring of that actuation
- If the actuator channel fails, the monitor channel detects it
- If the monitor channel fails, the actuator channel continues correctly
- Requires fault isolation to be single-fault tolerant
  - Actuator channel **cannot** use the monitor itself

# Monitor-Actuator Pattern

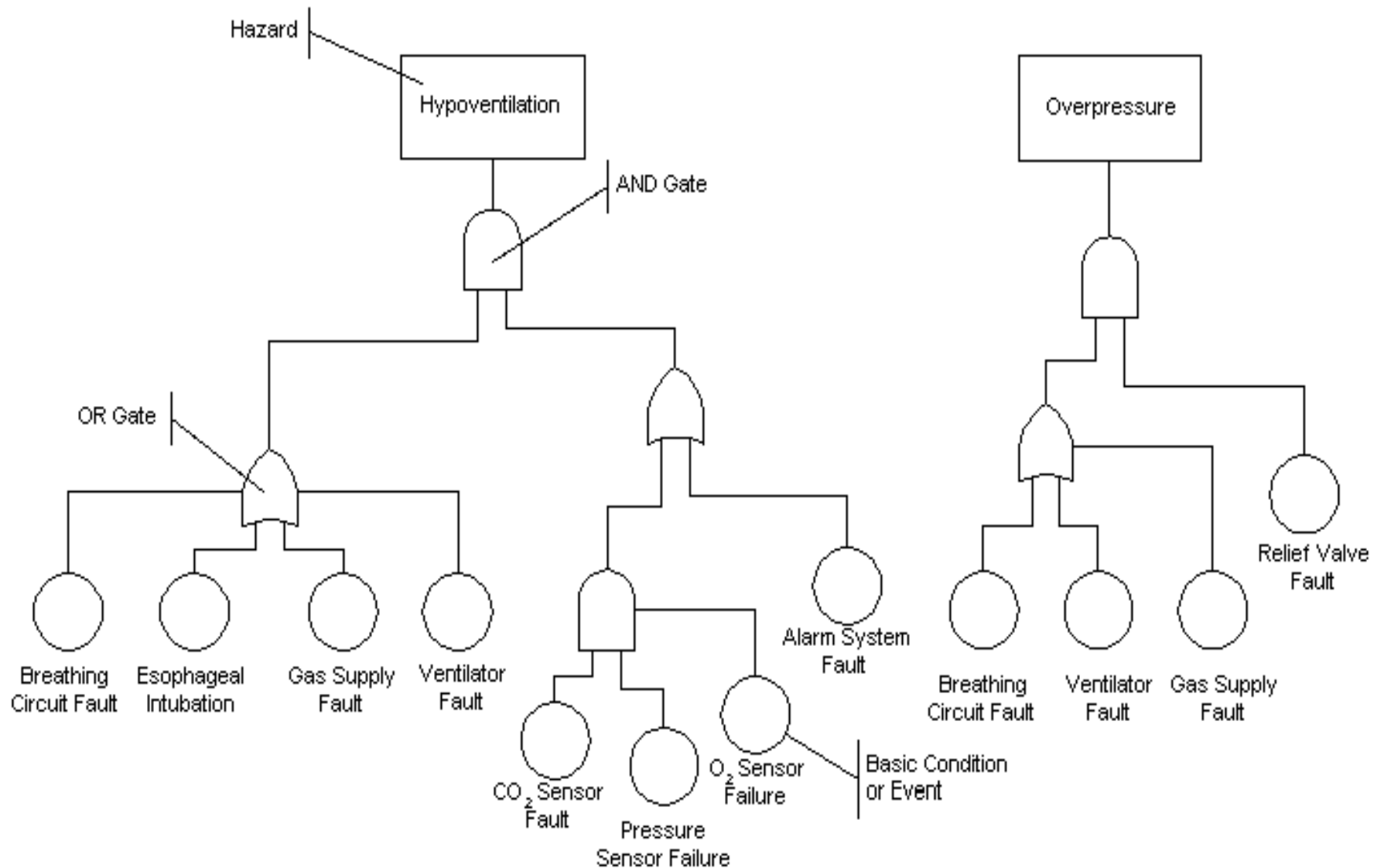


# Dual-Channel Design Architecture

## Dual Channel Ventilator Gas Delivery



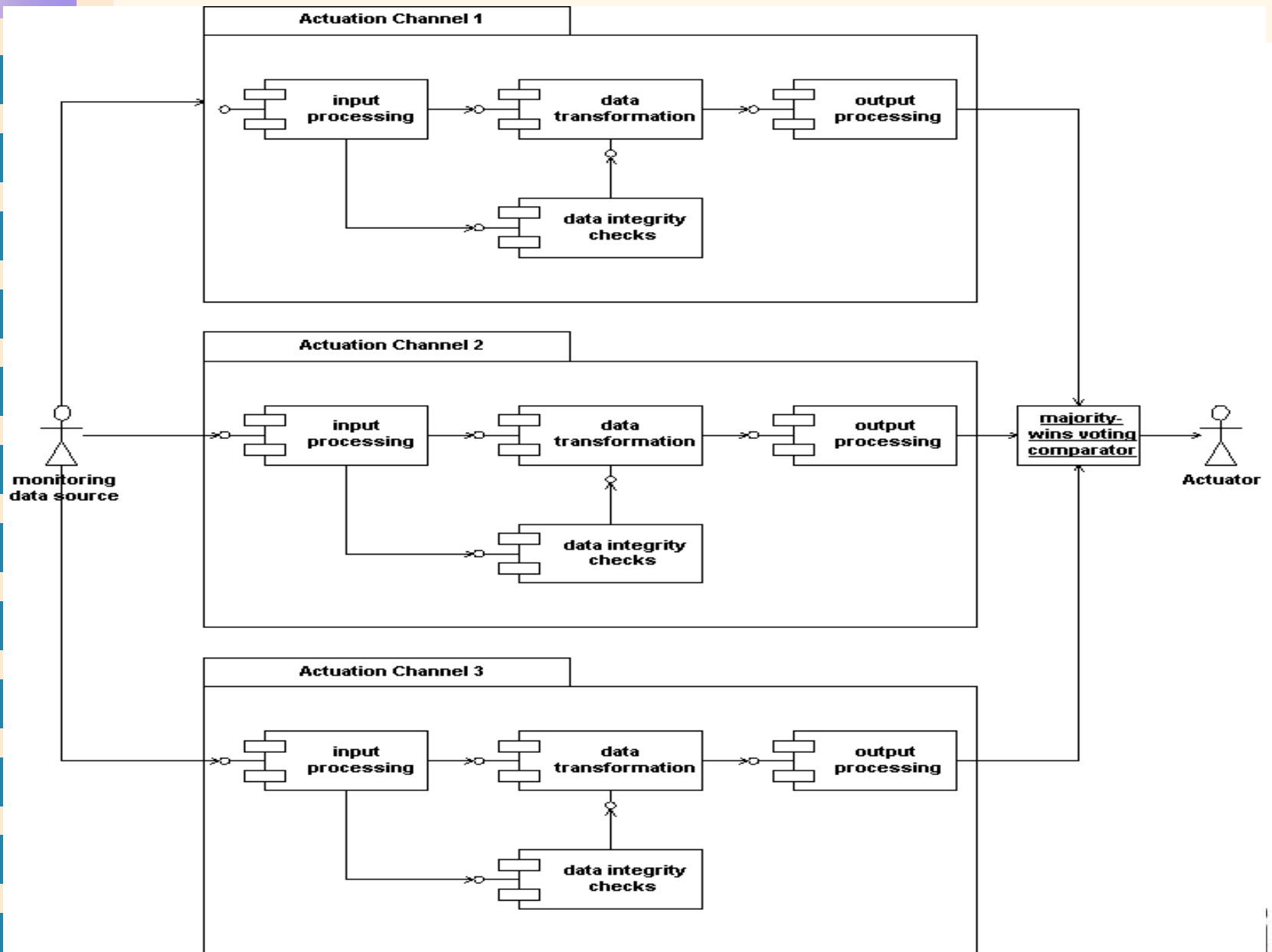
# Ventilator Fault Tree



# Multiple Channels with Voting

- Channels may be homogenous or heterogeneous
- Compare results of odd number of peer channels
- Primary channel with secondary reasonableness checks
  - May use algorithmic inversion to recreate initial data
  - May use different algorithm
  - May use different teams (not fool-proof)

# Triple Modular Redundancy



# Eight Steps to Safety

1. Identify the Hazards
2. Determine the Risks
3. Define the Safety Measures
4. Create Safe Requirements
5. Create Safe Designs
6. Implement Safety
7. Assure the Safety Process
8. Test, Test, Test

# Detailed Design For Safety

- **Make it right before you make it fast**
  - Simple, clear algorithms and code
  - Optimize only the 10%-20% of code which affects performance
  - Use “safe” language subsets
  - Ensure you haven’t introduced any common failure modes
- **Thoroughly test**
  - Unit test and peer review
  - Integration test
  - Validation test

# Detailed Design For Safety

- **Verify that it remains right throughout program execution**
  - Exceptions
  - Invariant assertions
  - Range checking
  - Index and boundary checking
- **When it's not right during execution, then make it right with corrective or protective measures**

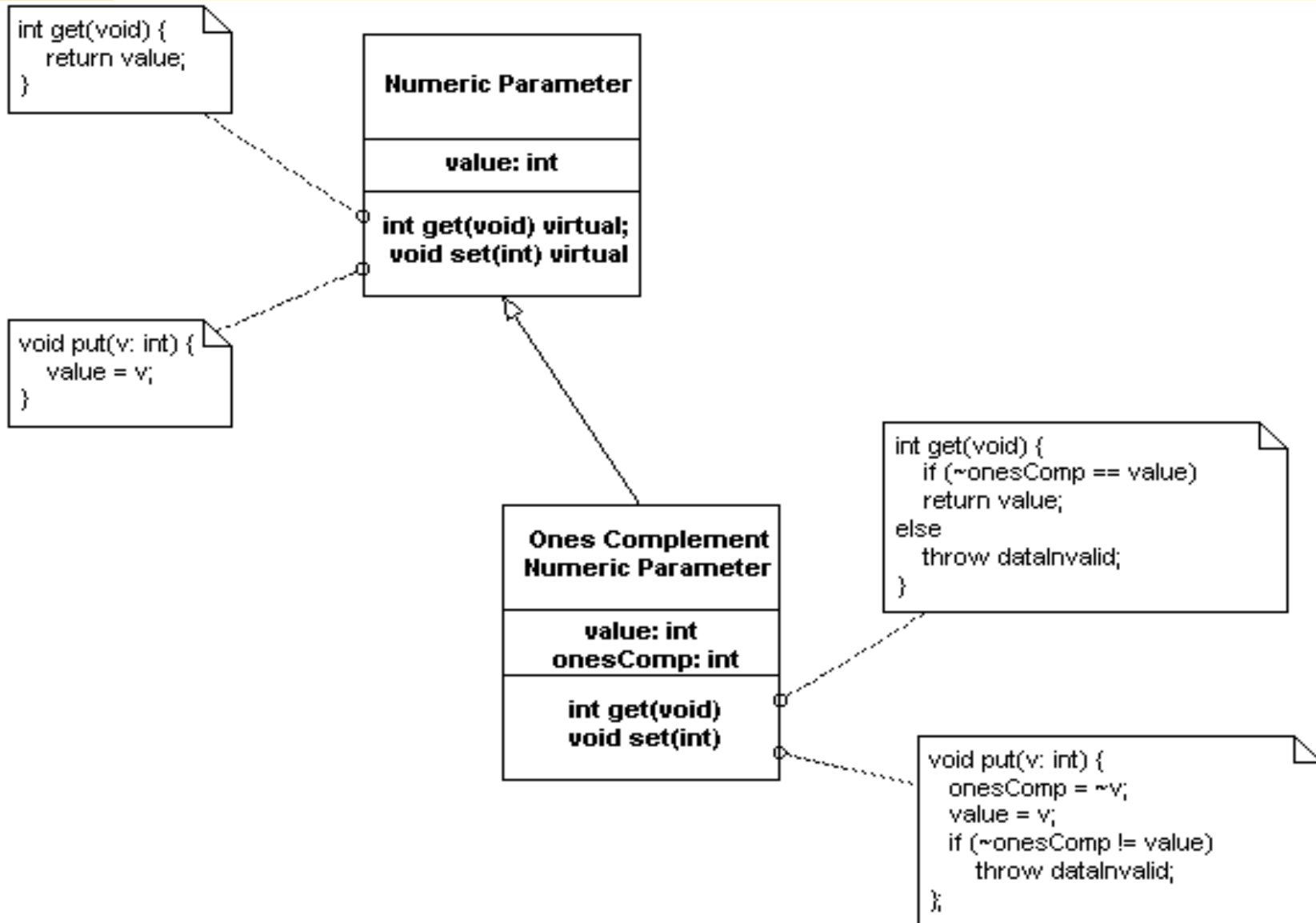
# Detailed Design For Safety

- **Use “safe” language subsets**
  - Strong compile-time checking
  - Strong run-time checking
  - Exception handling
  - Avoid error prone statements and syntax
- **Do not allow ignoring of error indications**
- **Separate normal code from error handling code**
- **Handle errors at the lowest level with sufficient context to correct the problem**

# Detailed Design For Safety

- Data Validity Checks
  - CRC (16-bit or 32-bit)
    - Identifies all single or dual bit errors
    - Detects high percentage of multiple bit errors
    - Table- or compute-driven
    - Chips are available
  - Checksum
  - Redundant storage
    - One's complement
- Redundancy should be set every write access
- Data should be checked every read access

# Detailed Design for Safety



# Eight Steps to Safety

1. Identify the Hazards
2. Determine the Risks
3. Define the Safety Measures
4. Create Safe Requirements
5. Create Safe Designs
6. Implement Safety
7. Assure the Safety Process
8. Test, Test, Test

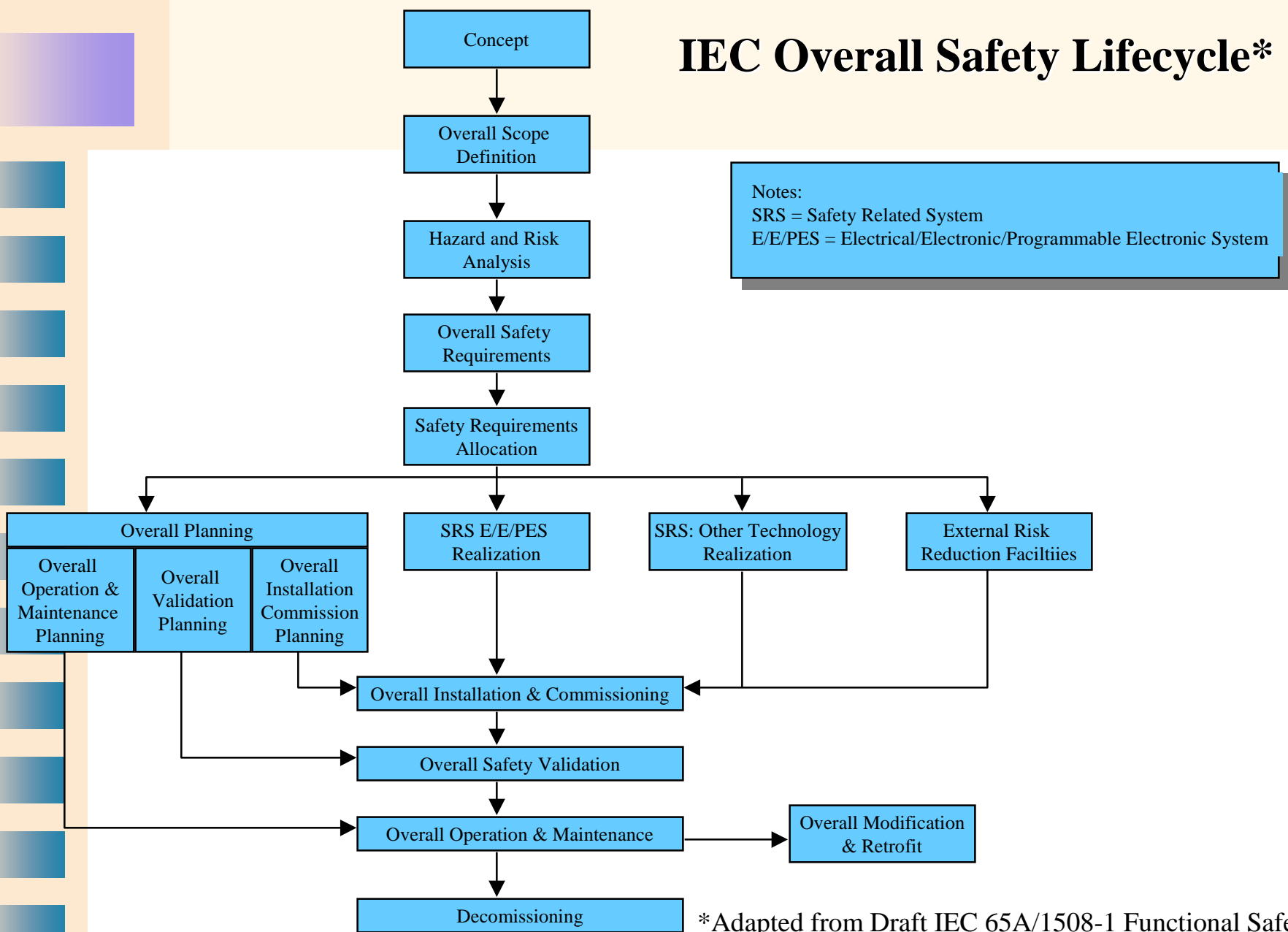
# Safety Process (Development)

- Do Hazard Analysis early and often
- Track safety measures from hazard analysis to
  - Requirements Specification
  - Design
  - Code
  - Validation Tests
- Test safety measures with fault seeding

# Safety Process (Deployment)

- Install Safely
  - Ensure proper means are used to set up system
  - Safety measures are installed and checked
- Deploy Safely
  - Ensure safety measures are periodically checked and serviced
  - Do not turn off safety measures (see Bophal)
- Decommission Safely
  - Removal of hazardous materials

# IEC Overall Safety Lifecycle\*



\*Adapted from Draft IEC 65A/1508-1 Functional Safety: Safety Related Systems. Part 1: General Requirements.

# Eight Steps to Safety

1. Identify the Hazards
2. Determine the Risks
3. Define the Safety Measures
4. Create Safe Requirements
5. Create Safe Designs
6. Implement Safety
7. Assure the Safety Process
8. Test, Test, Test

# Safety in Testing in R&D

- Use *fault-seeding*
  - **Unit (Class) testing**
    - White box
    - Procedural invariant violation assertions
    - Peer reviews
  - **Integration testing**
    - Grey box
  - **Validation testing**
    - Black box
    - Externally caused faults
    - (Grey box) Internally seeded faults

# Safety Testing During Operation

- Power On Self Test (POST)
  - Check for latent faults
  - All safety measures must be tested at power on and periodically
    - RAM (stuck-at, shorts, cell failures)
    - ROM
    - Flash
    - Disks
    - CPU
    - Interfaces
    - Buses

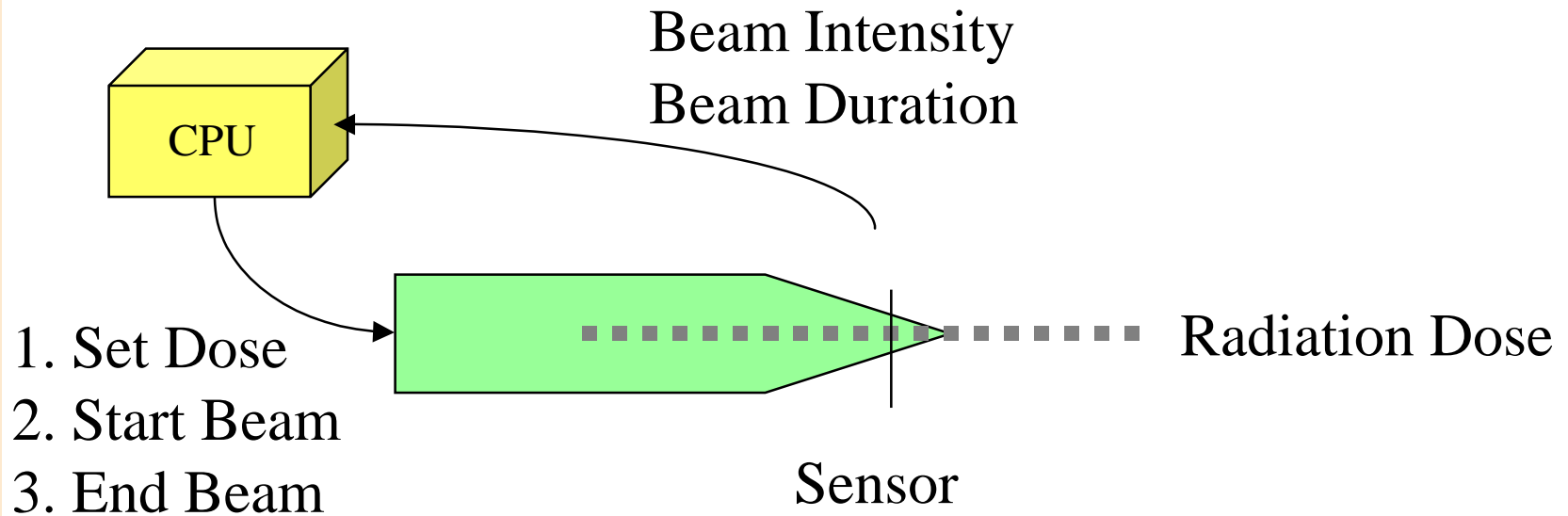
# Safety Testing During Operation

- Built-In Tests
  - Repeats some of POST
  - Data integrity checks
  - Index and pointer validity checking
  - Subrange value invariant assertions
  - Proper functioning
    - Watchdogs
    - Reasonableness checks (e.g. Sanity Check Pattern)
    - Lifeticks

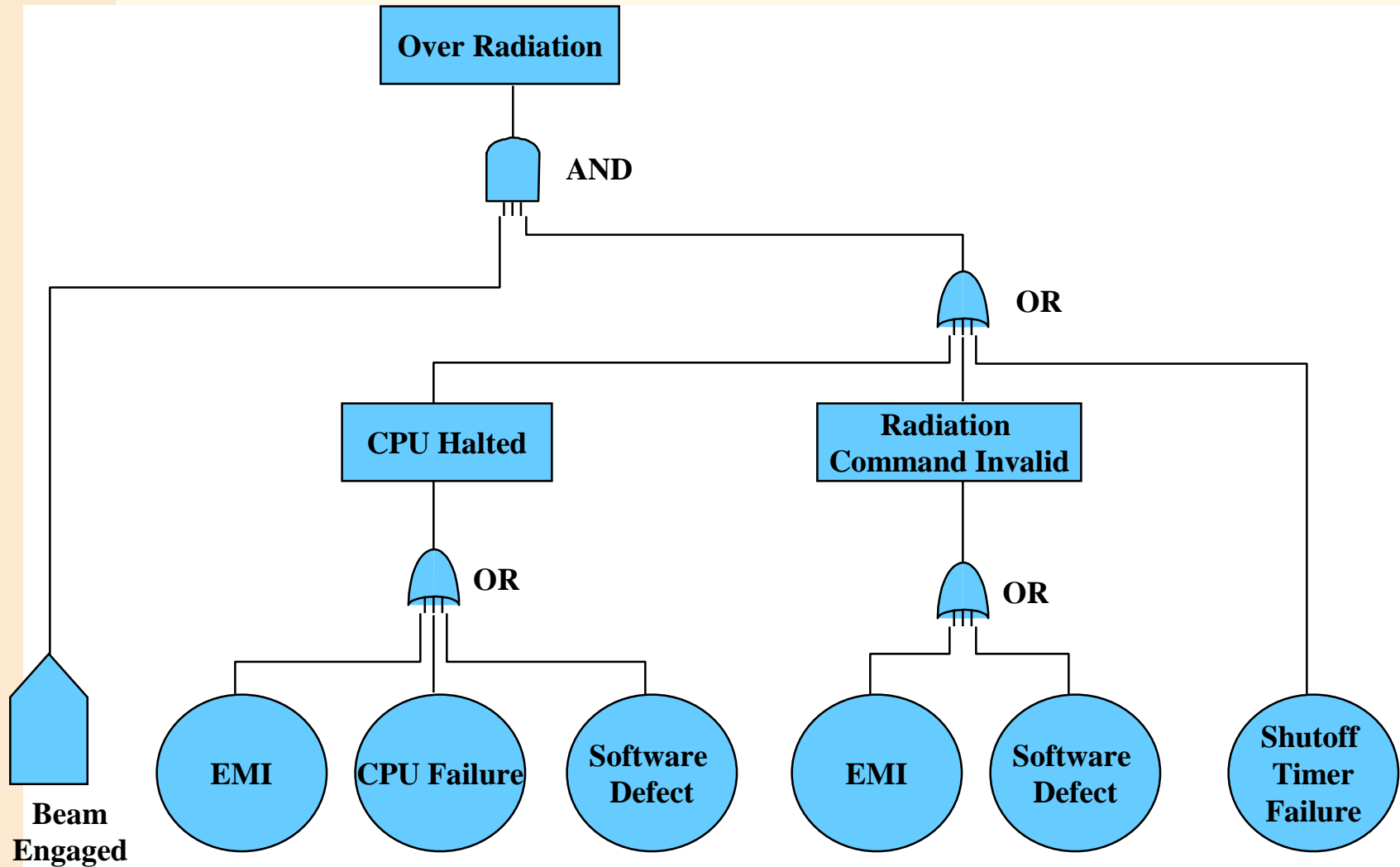


# **A simplified Example: A Linear Accelerator**

# Unsafe Linear Accelerator



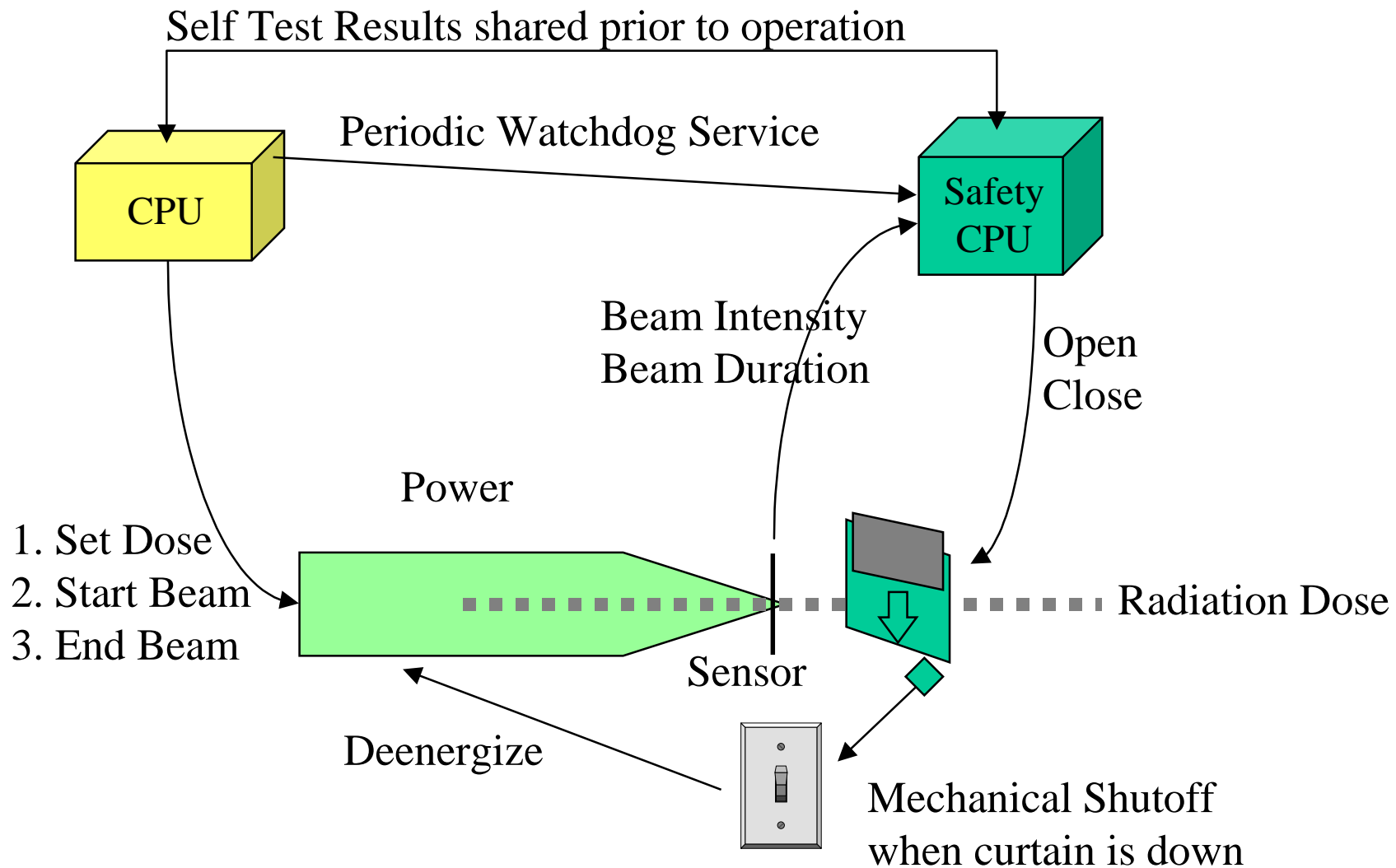
# Fault Tree Analysis



# Hazards of the Linear Accelerator

<b>Hazard</b>	<b>Level of Risk</b>	<b>Tolerance Time <math>T_1</math></b>	<b>Fault</b>	<b>Likelihood</b>	<b>Detection Time</b>	<b>Control Measure</b>	<b>Exposure Time</b>
<b>Over radiation</b>	<b>Severe</b>	<b>100 ms</b>	<b>CPU Locks up</b>	<b>rare</b>	<b>50 ms</b>	<b>Safety CPU checks lifetick @ 25 ms</b>	<b>50 ms</b>
			<b>Corrupt data settings</b>	<b>often</b>	<b>10 ms</b>	<b>32-bit CRCs on data checked every access</b>	<b>15 ms</b>
<b>Under radiation</b>	<b>Moderate</b>	<b>2 weeks</b>	<b>Corrupt data settings</b>	<b>often</b>	<b>10 ms</b>	<b>32-bit CRCs on data checked every access</b>	<b>15 ms</b>
<b>Inadvertent Radiation on power on</b>	<b>Severe</b>	<b>100 ms</b>	<b>Beam left engaged during power down</b>	<b>often</b>	<b>n/a</b>	<b>Curtain mechanically shuts at power down</b>	<b>0 ms</b>

# Safe Linear Accelerator



# Conclusion

- Safety is a system issue
- It is cheaper and more effective to include safety early on then to add it later
- Safety architectures provide programming-in-the-large safety
- Safe coding rules and detailed design provide programming-in-the-small safety



**i-Logix**

Embedded Systems and Software Design Automation

I-Logix