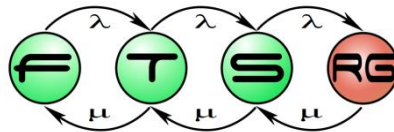


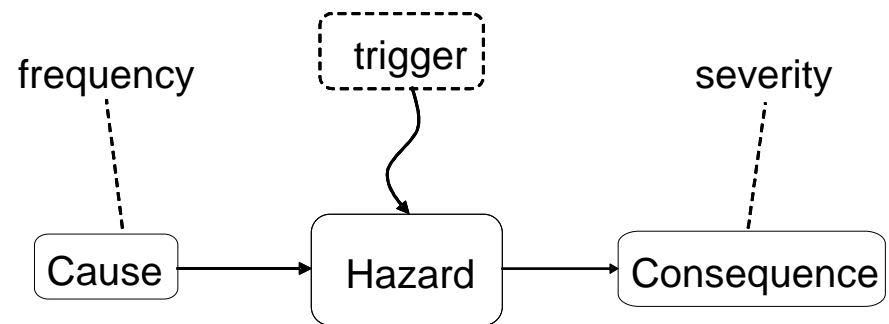
# Hazard Analysis

Ákos Horváth and István Majzik  
Budapest University of Technology and Economics  
Dept. of Measurement and Information Systems



# Hazard analysis

- Goal: Analysis of the fault effects and the evolution of hazards through dangerous states
  - What are the **causes** for a hazard?
  - What are the **consequences** of a component fault?
- Results:
  - Categorization of hazards
    - **Frequency** of occurrence
    - **Severity** of consequences
  - Hazard catalogue
  - **Risk matrix**
- These results form the basis for **risk reduction**



# Categorization of the techniques

- On the basis of the development phase (tasks):
  - **Design phase**: Identification and analysis of hazards
  - Delivery phase: Demonstration of safety
  - Operation phase: Checking the modifications
- On the basis of the analysis approach:
  - Cause-consequence view:
    - **Forward (inductive)**: Analysis of the **effects** of fault/events
    - **Backward (deductive)**: Analysis of the **causes** of hazards
  - System hierarchy view:
    - **Bottom-up**: From the components (subsystems) to system level
    - **Top-down**: From the system level towards the components
- **Systematic** techniques are needed

# Hazard analysis techniques (overview)

1. Checklists
2. Fault Tree
3. Event Tree
4. Cause-Consequence Analysis
5. Failure Modes and Effects Analysis (FMEA)

# 1. Checklists

- Basic approach
  - Collection of experiences about typical faults and hazards
  - Used as **guidelines** and as „rule of thumb“
- Advantages
  - Known sources of hazards are included
  - Well-proven ideas and solutions can be applied
- Disadvantages
  - Completeness is hard to achieve (checklist is incomplete)
  - False confidence about safety
  - Applicability in different domains than the original domain of the checklist is questionable

# Example: Checklist to examine a specification

- **Completeness**
  - Complete list of functions, references, tools
- **Consistency**
  - Internal and external consistency
  - Traceability of requirements
- **Realizability**
  - Resources are available
  - Usability is considered
  - Maintainability is considered
  - Risks: cost, technical, environmental
- **Testability**
  - Specific requirements
  - Unambiguous requirements
  - Quantitative requirements (if possible)



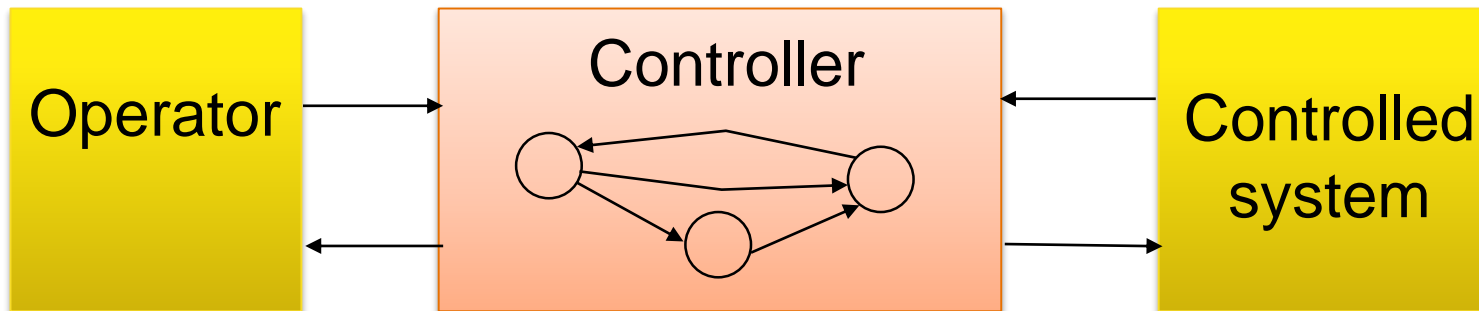
# Motivations to check the specification

- Experience: Hazards are often caused by **incomplete or inconsistent specification**
  - Example: Statistics of failures detected during the software testing of the Voyager and Galileo spacecraft
    - 78% (149/192) **specification related failures**, from which
      - 23% stuck in dangerous state (without exit)
      - 16% lack of timing constraints
      - 12% lack of reaction to input event
      - 10% lack of checking input values
- Potential solutions to avoid problems
  - Using a strong specification language
  - Applying correct design patterns
  - **Checking the specification**

# Example: Checklist for state machine specifications

## Completeness and consistency:

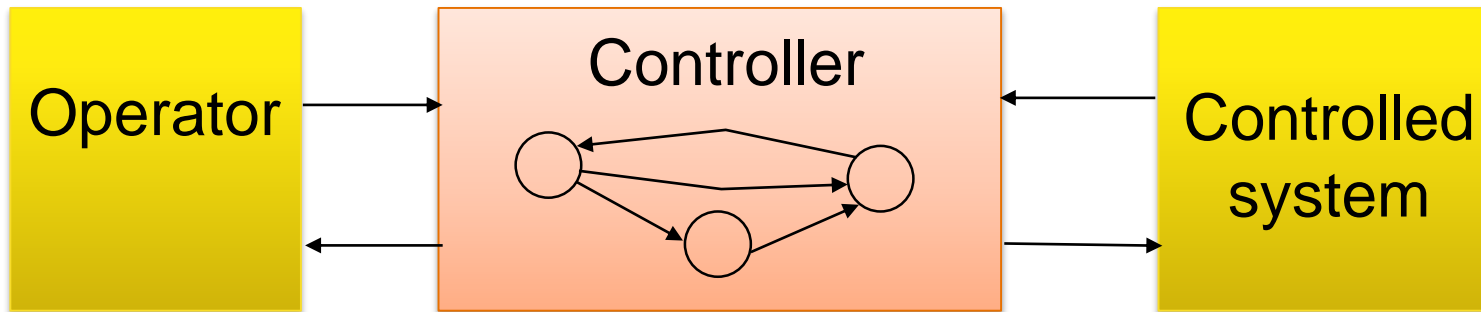
- State definition
- Inputs (trigger events)
- Outputs
- Relation of inputs (triggers) and outputs
- State transitions
- Human-machine interface





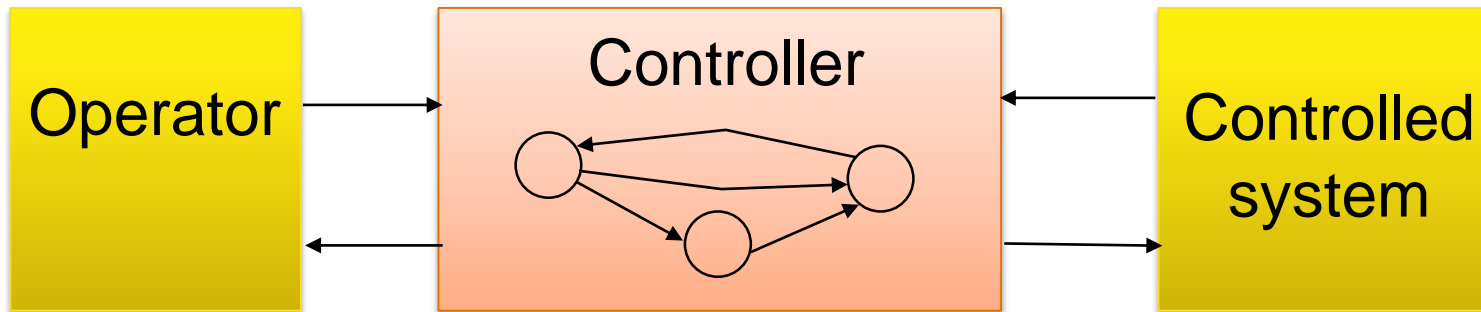
# Example: Checklist for state machine specifications

- State definition
- Inputs (triggers)
- Outputs
  - Safe initial state
  - Actualization of the internal model: timeout and transition to “invalid” state if input events are missing; output is not allowed in this state
- Relationship
- State transitions
- Human-machine interface



# Example: Checklist for state machine specifications

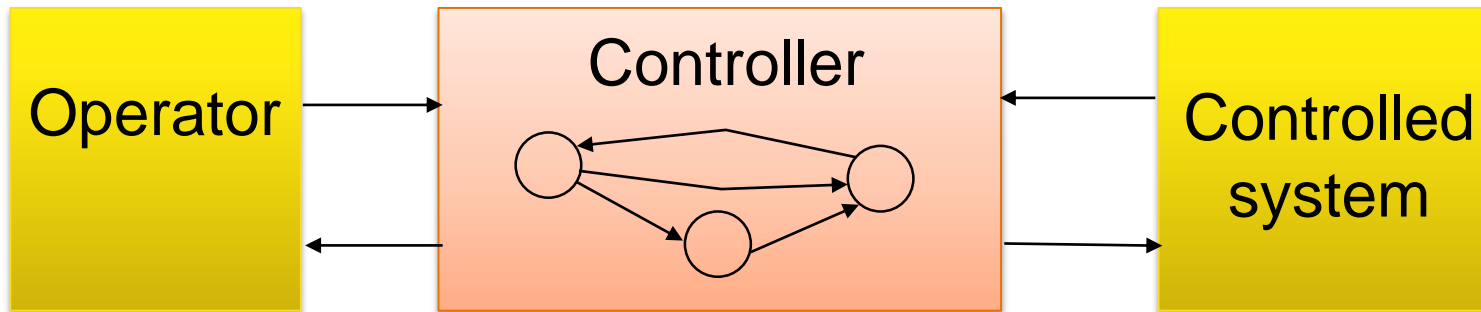
- State definition
- Inputs (trigger events)
- Outputs
  - Reaction to each potential input event
- Relations
  - Deterministic reactions
- State transitions
  - Input checking (value, timing)
  - Handling of invalid inputs
- Human factors
  - Limited rate of interrupts



# Example: Checklist for state machine specifications

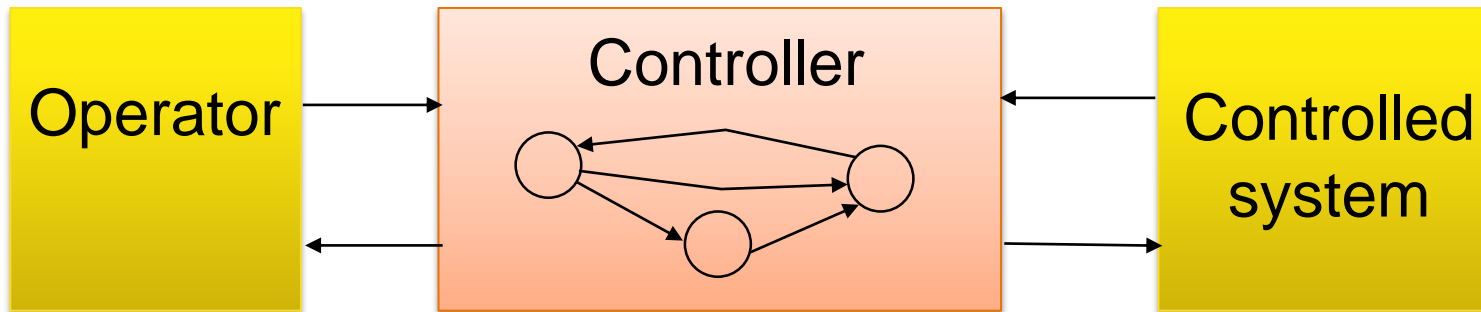
- State definition
- Inputs (trigger events)
- Outputs
- Relations
  - Acceptance checking on the output
- State transitions
  - There are no unused outputs
  - Compliance with the limitations of the environment
- Human

- Acceptance checking on the output  
- There are no unused outputs  
- Compliance with the limitations of the environment



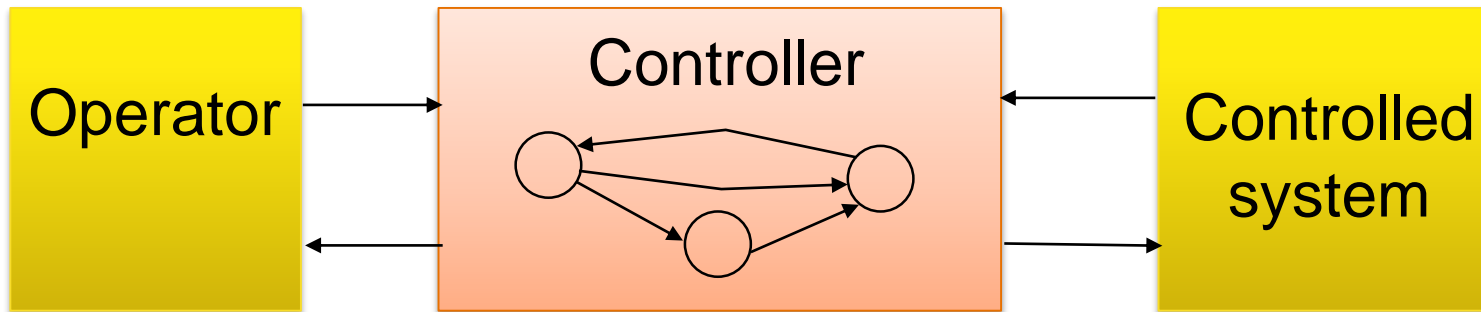
# Example: Checklist for state machine specifications

- State definition
  - The effects of outputs are checked through processing the inputs
- Inputs (triggers)
  - Stability of the control loop is preserved
- Outputs
- Relation of inputs (triggers) and outputs
- State transitions
- Human-machine interface



# Example: Checklist for state machine specifications

- State
  - Each state is reachable (static reachability)
- Input
  - Transitions are reversible (reverse path exists)
- Output
  - Multiple transitions from dangerous to safe states
  - Confirmed transitions from safe to dangerous states
- Relationship between states, safety, and compliance
- State transitions
- Human-machine interface



# Example: Checklist for state machine specifications

- State definition

- Input Well-specified outputs towards the operator:

- Ordering (with priorities)

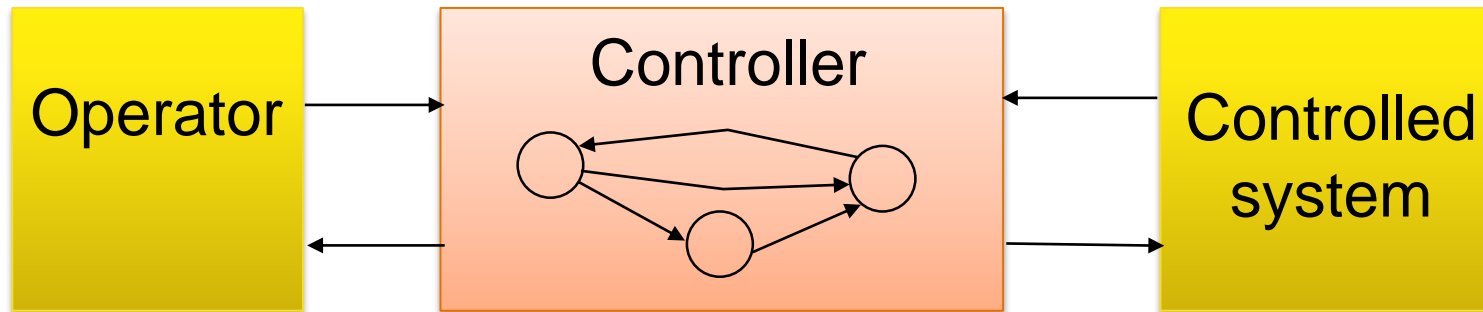
- Output - Update frequency

- Timeliness

- Reliability

- State transitions

- Human-machine interface



# Example: Static checking of the source code

- Goal: Finding dangerous constructs
  - Basis: Language subset (allowed constructs)
- Tool support
  - Finding typical faults (e.g., Lint for C)
    - **Data related faults**: Lack of initialization, ...
    - **Control related faults**: Unreachable statements, ...
    - **Interface related faults**: Improper type, lack of return value, ...
    - **Memory related faults**: Lack of releasing unused memory, ...
  - Semantic analysis (e.g., PolySpace tool)
    - Analysis of the function call hierarchy
    - Checking data flow (relations among variables)
    - Checking the ranges of variables
    - Checking coding rules (e.g., code complexity metrics)

# Example: Output of the analysis in PolySpace

```
static void Square_Root_conv (double alpha, float *beta_pt, float *gamma)
{
  *beta_pt = (float)((1.5 + cos(alpha))/5.0);

  if(*beta_pt < 0.3)
    *gamma = 0.75;
}

static void Square_Root (void)
{
  double alpha = random_float();
  float beta;
  float gamma;

  Square_Root_conv (alpha, &beta, &gamma);

  if(random_int() > 0){
    gamma = (float)sqrt(beta - 0.75);
  }
  else{
    gamma = (float)sqrt(gamma - beta);
    if(beta > 1)
      alpha = 0;
  }
}
```

## The Colors of PolySpace

Each function and operation is verified for **all** possible values, and then colored according to its reliability.

**Green** Proven safe under all operating conditions. Focus your efforts elsewhere.

**Red** Proven definite error each time the operation is executed.

**Orange** Unproven.

**Grey** Proven unreachable code. May point to a functional issue.

- Static analysis and code colouring: Identification of dangerous constructs



# 2. Fault tree analysis

Analysis of the **causes** of system level hazards

- **Top-down** analysis
- Identifying the component level combinations of faults/events that may lead to hazard

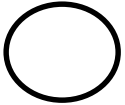
Construction of the fault tree

1. Identification of the foreseen **system level hazard**: on the basis of environment risks, standards etc.
2. Identification of **intermediate events (pseudo-events)**: Boolean (AND, OR) combinations of lower level events that may cause upper level events
3. Identification of **primary (basic) events**: no further refinement is needed/possible

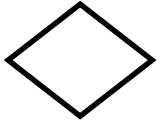
# Set of elements in a fault tree



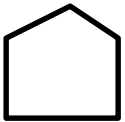
Top level or intermediate event



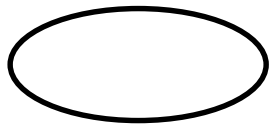
Primary (basic) event



Event without further analysis



Normal event (i.e., not a fault)



Condition for a composite event



AND combination of events

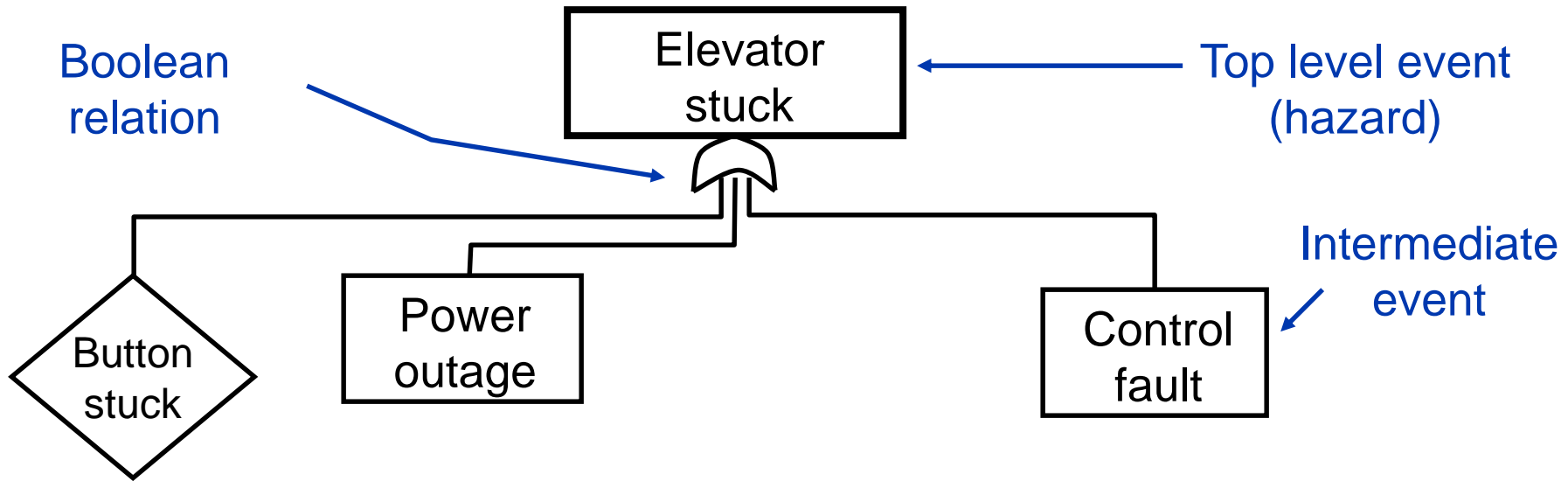


OR combination of events

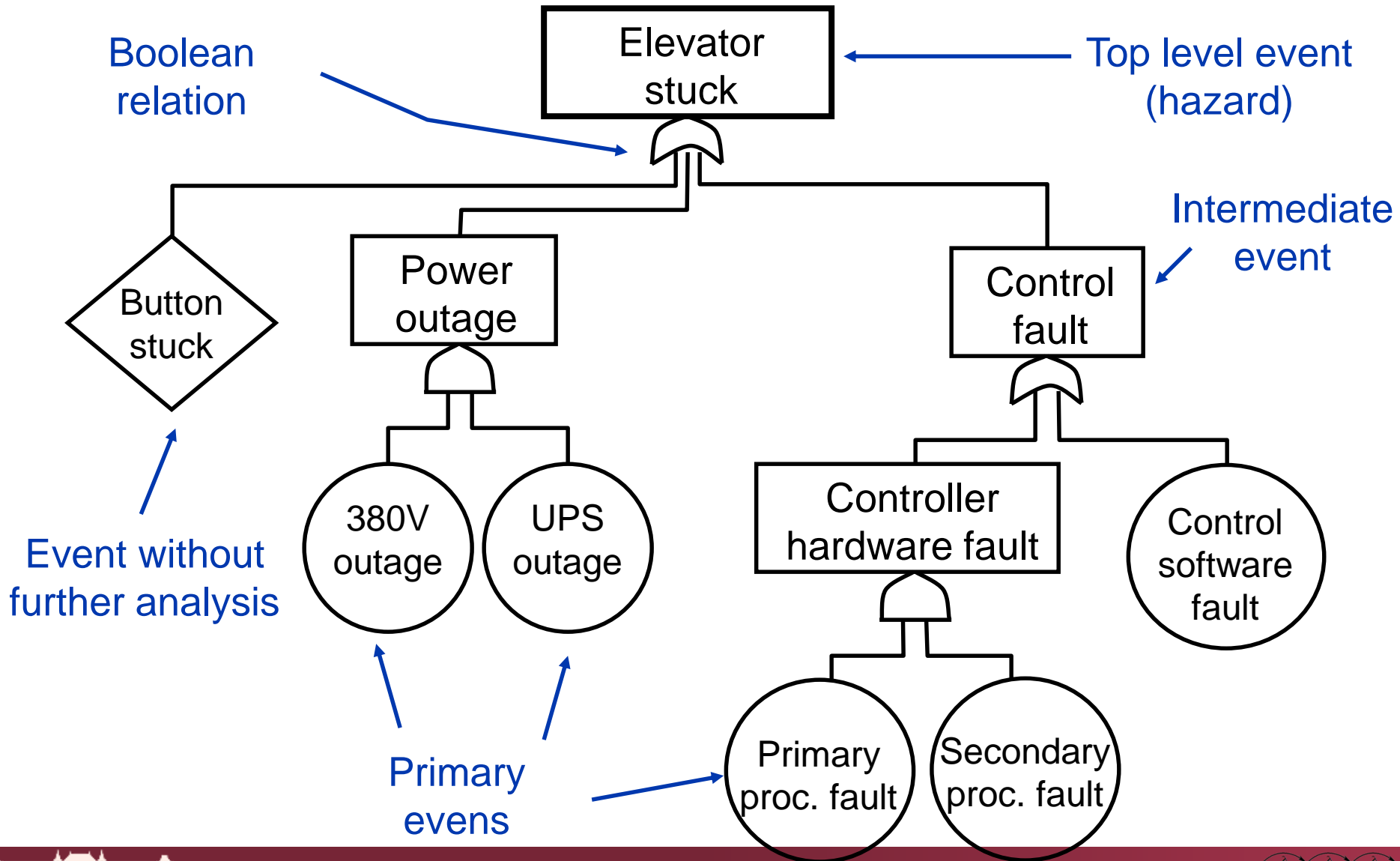
# Fault tree example: Elevator



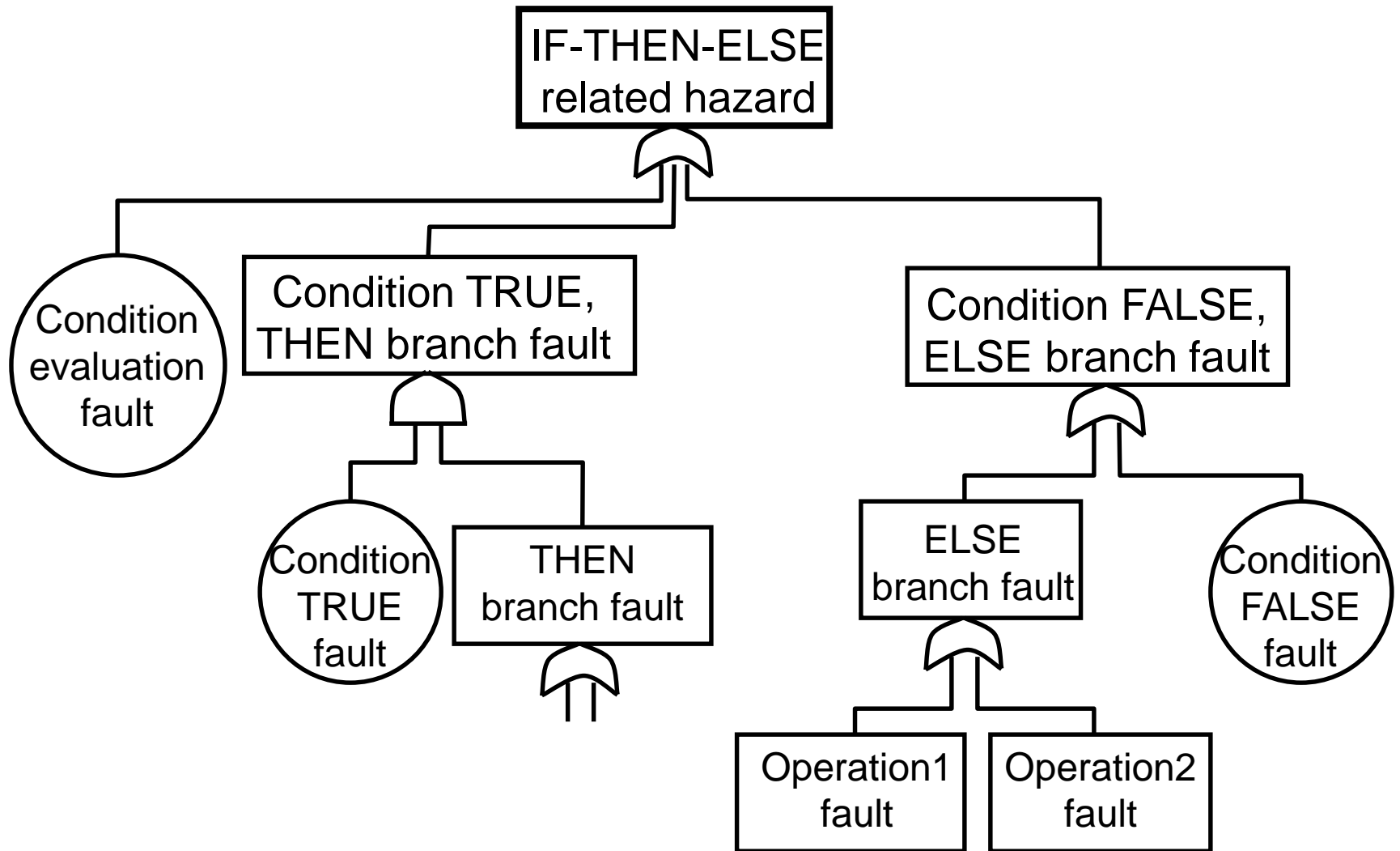
# Fault tree example: Elevator



# Fault tree example: Elevator



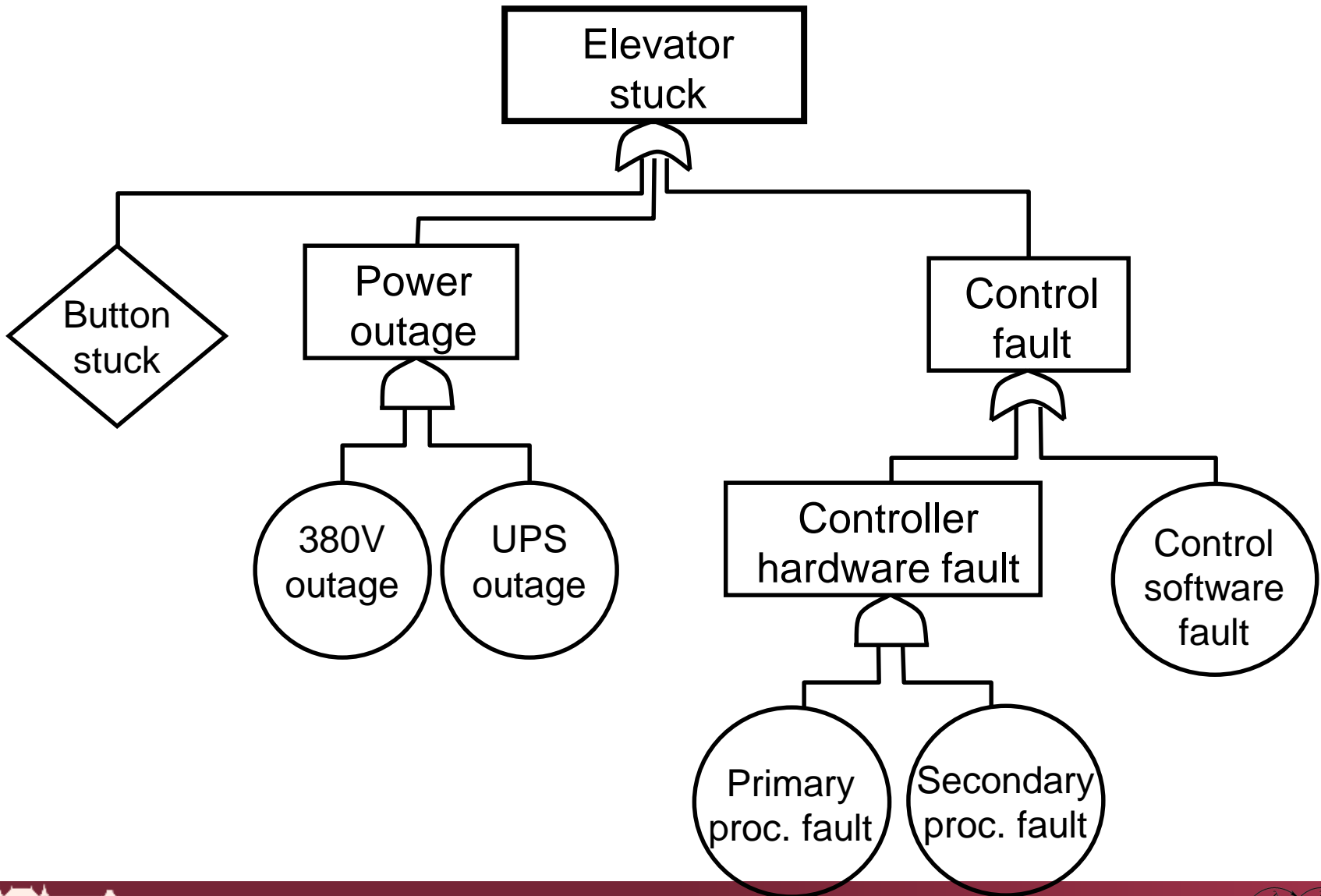
# Fault tree example: Software analysis



# Qualitative analysis of the fault tree

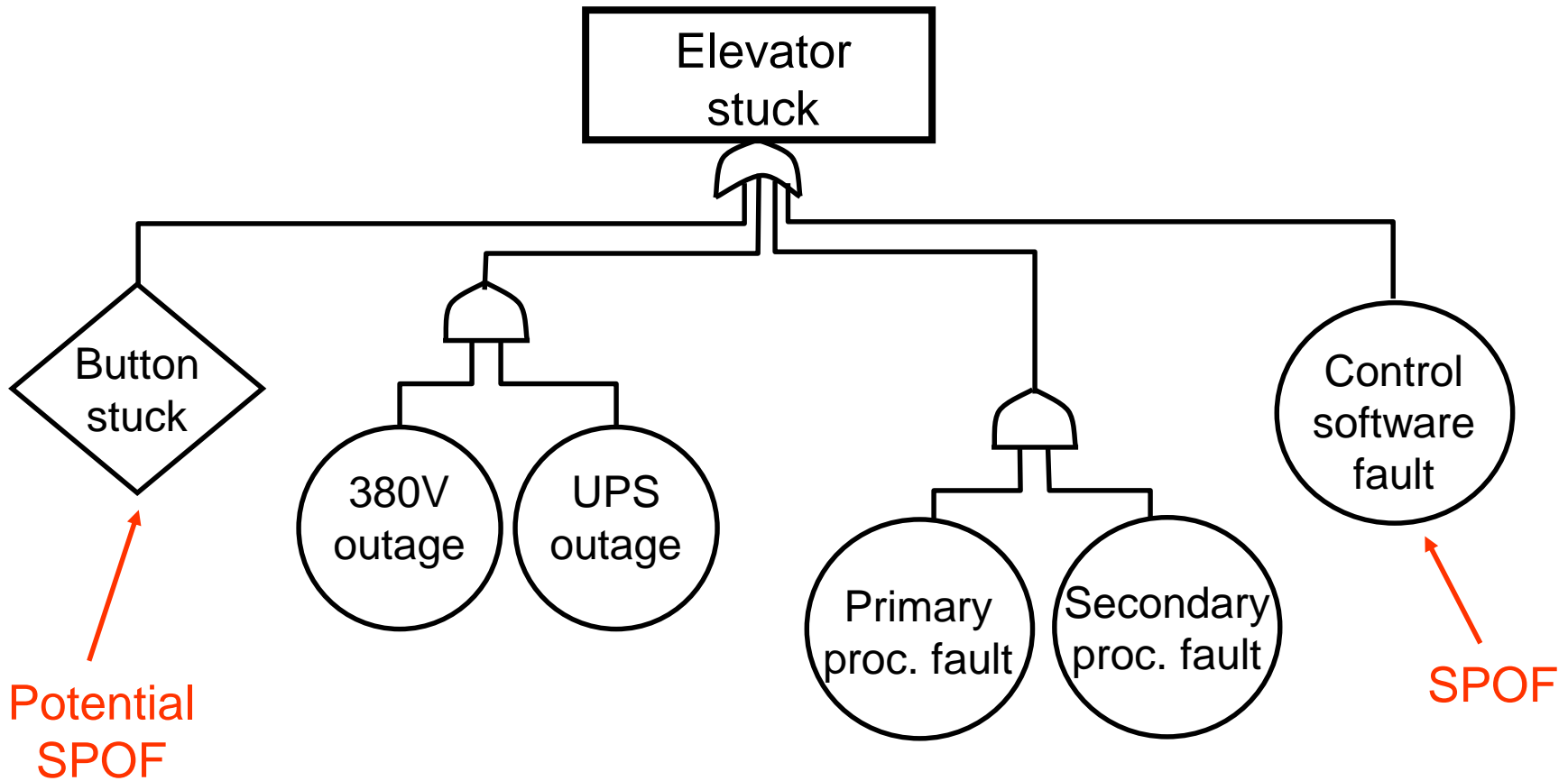
- Fault tree **reduction**: Resolving intermediate events/pseudo-events using primary events  
→ disjunctive normal form (OR on the top of the tree)
- **Cut** of the fault tree:  
AND combination of primary events
- **Minimal cut set**: No further reduction is possible
  - Minimal cut: There is no other cut that forms its subset
- Outputs of the analysis of the reduced fault tree:
  - **Single point of failure** (SPOF)
  - Critical events that appear in several cuts

# Original fault tree of the elevator example





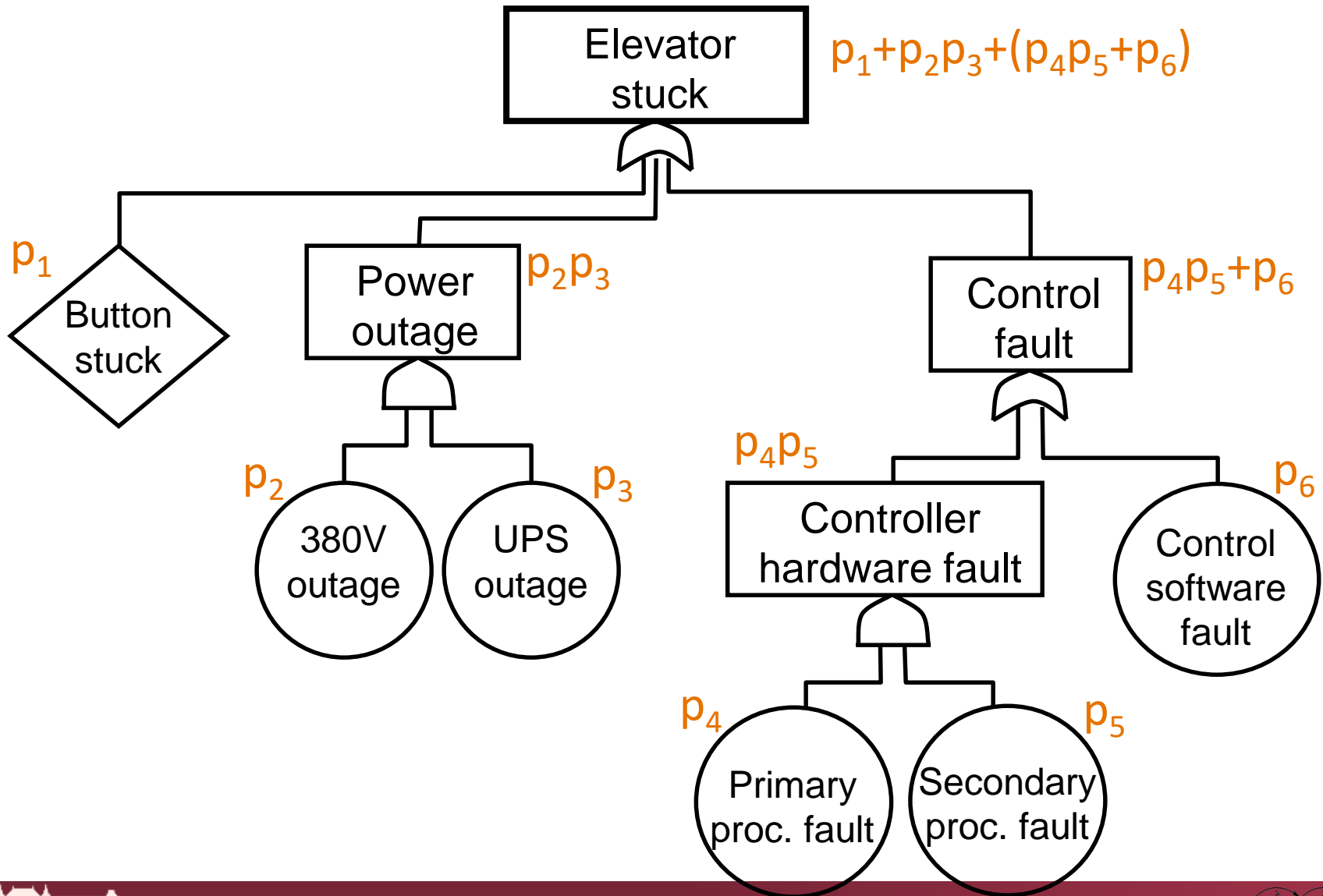
# Reduced fault tree of the elevator example



# Quantitative analysis of the fault tree

- Basis: **Probabilities** of the primary events
  - Component level data, experience, or estimation
- Result: Probability of the system level **hazard**
  - Computing probability on the basis of the probabilities of the primary events, depending on their combinations
  - AND gate: **product** (if the events are independent)
    - Exact calculation:  $P\{A \text{ and } B\} = P\{A\} \cdot P\{B|A\} \leq P\{A\} \cdot P\{B\}$
  - OR gate: **sum** (worst case estimation)
    - Exactly:  $P\{A \text{ or } B\} = P\{A\} + P\{B\} - P\{A \text{ and } B\} \leq P\{A\} + P\{B\}$
- Typical problems:
  - Correlated faults (not independent)
  - Handling of fault sequences

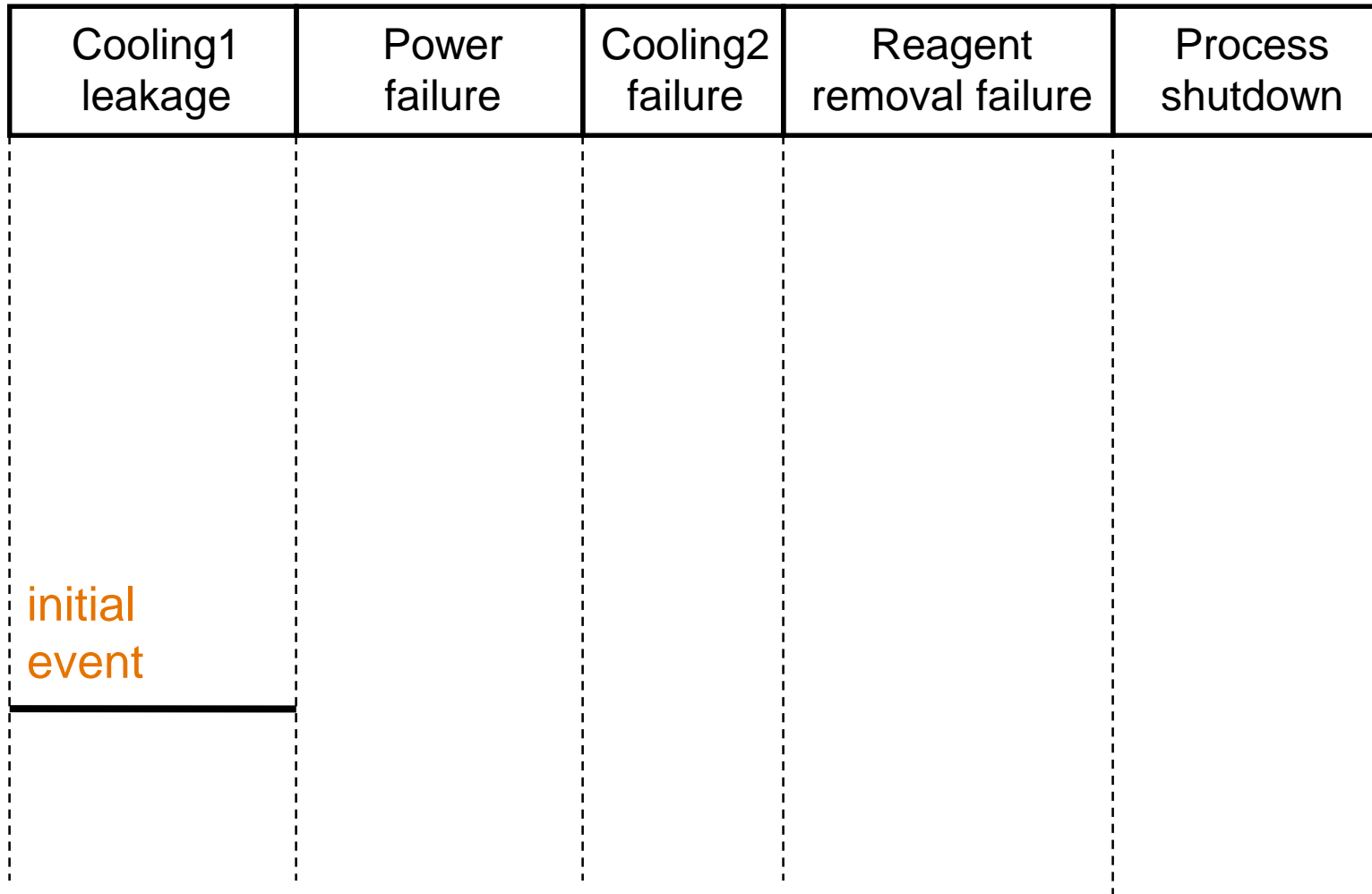
# Fault tree of the elevator with probabilities



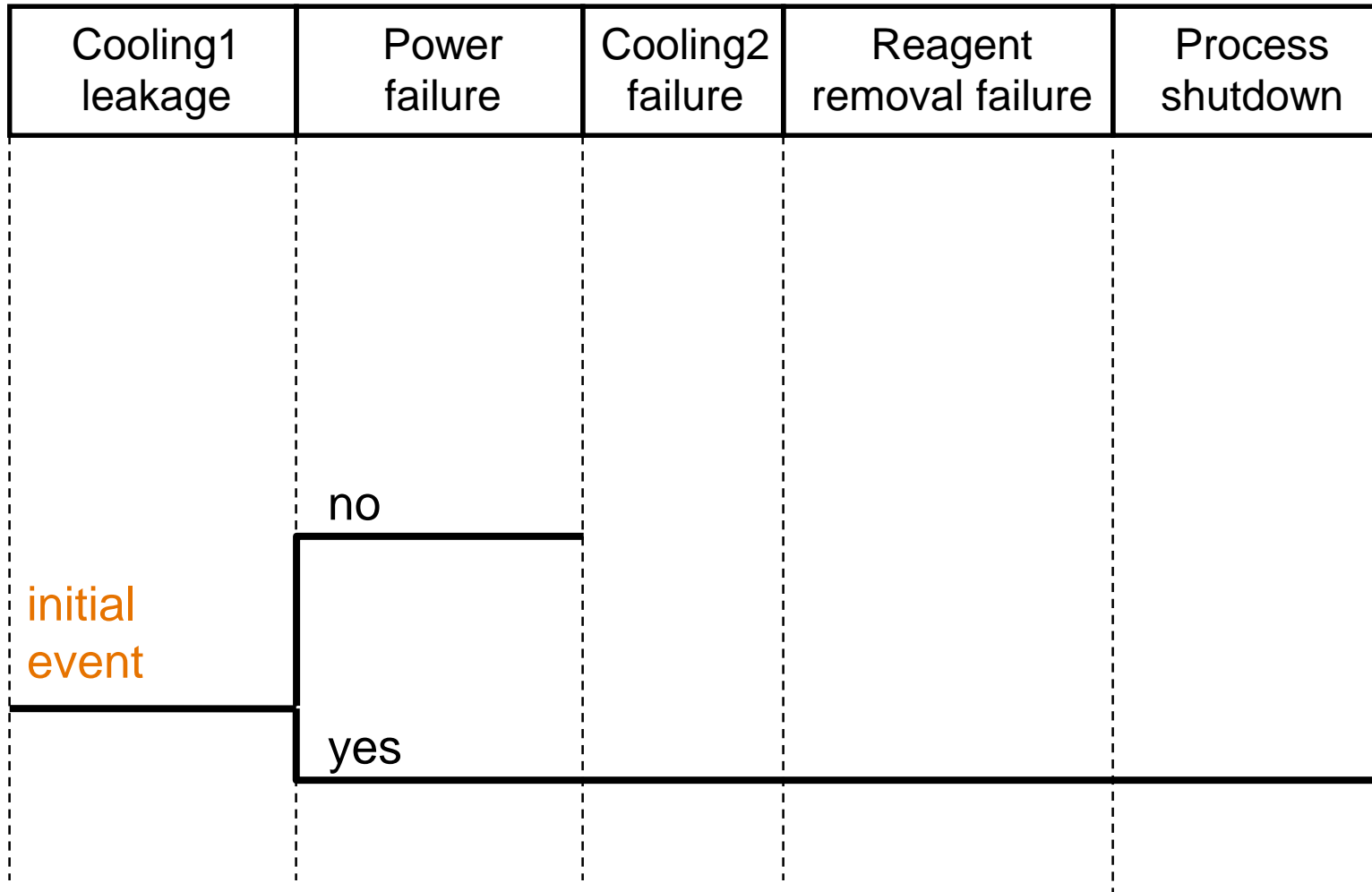
# 3. Event tree analysis

- Forward (inductive) analysis:  
Investigates the **effects** of an initial event
  - **Initial event:** component level fault/event
  - Related events: faults/events of other components
  - Ordering: causality, timing
  - Branches: depend on the occurrence of events
- Investigation of **hazard occurrence „scenarios“**
  - Path **probabilities** (on the basis of branch probabilities)
- Advantages: Investigation of **event sequences**
  - Example: Checking protection systems (protection levels)
- Limits: Complexity, multiplicity of events

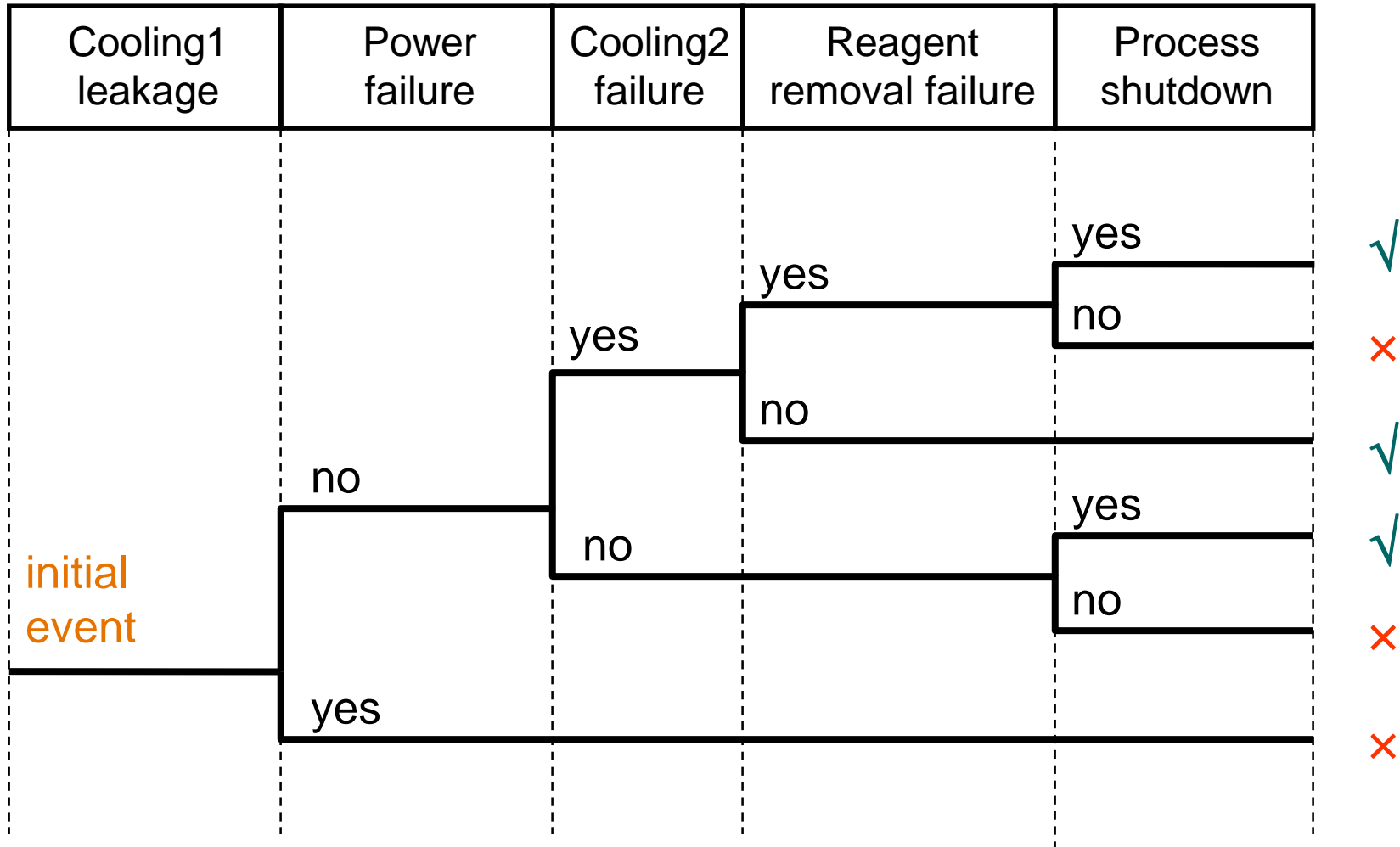
# Event tree example: Reactor cooling



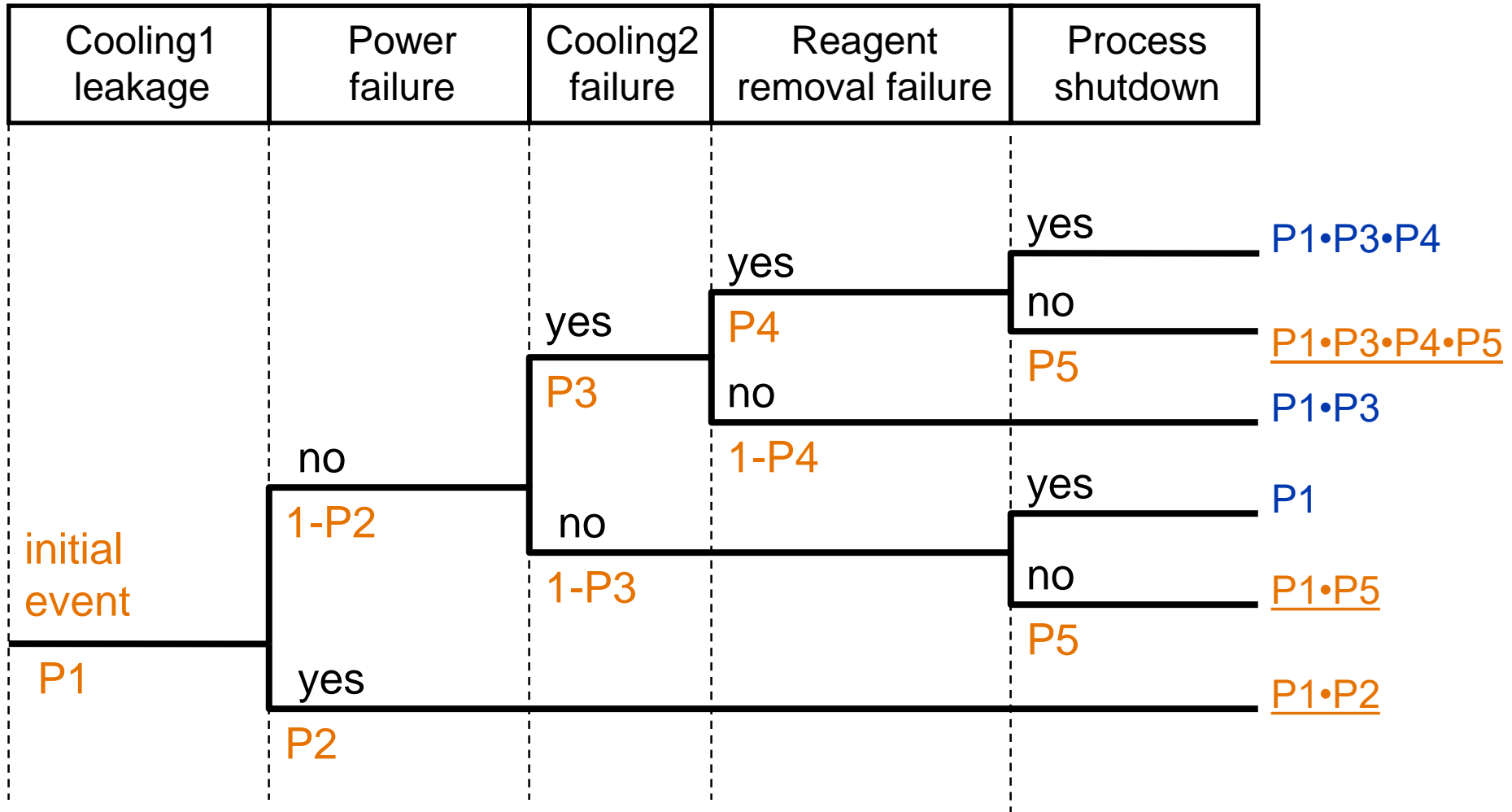
# Event tree example: Reactor cooling



# Event tree example: Reactor cooling

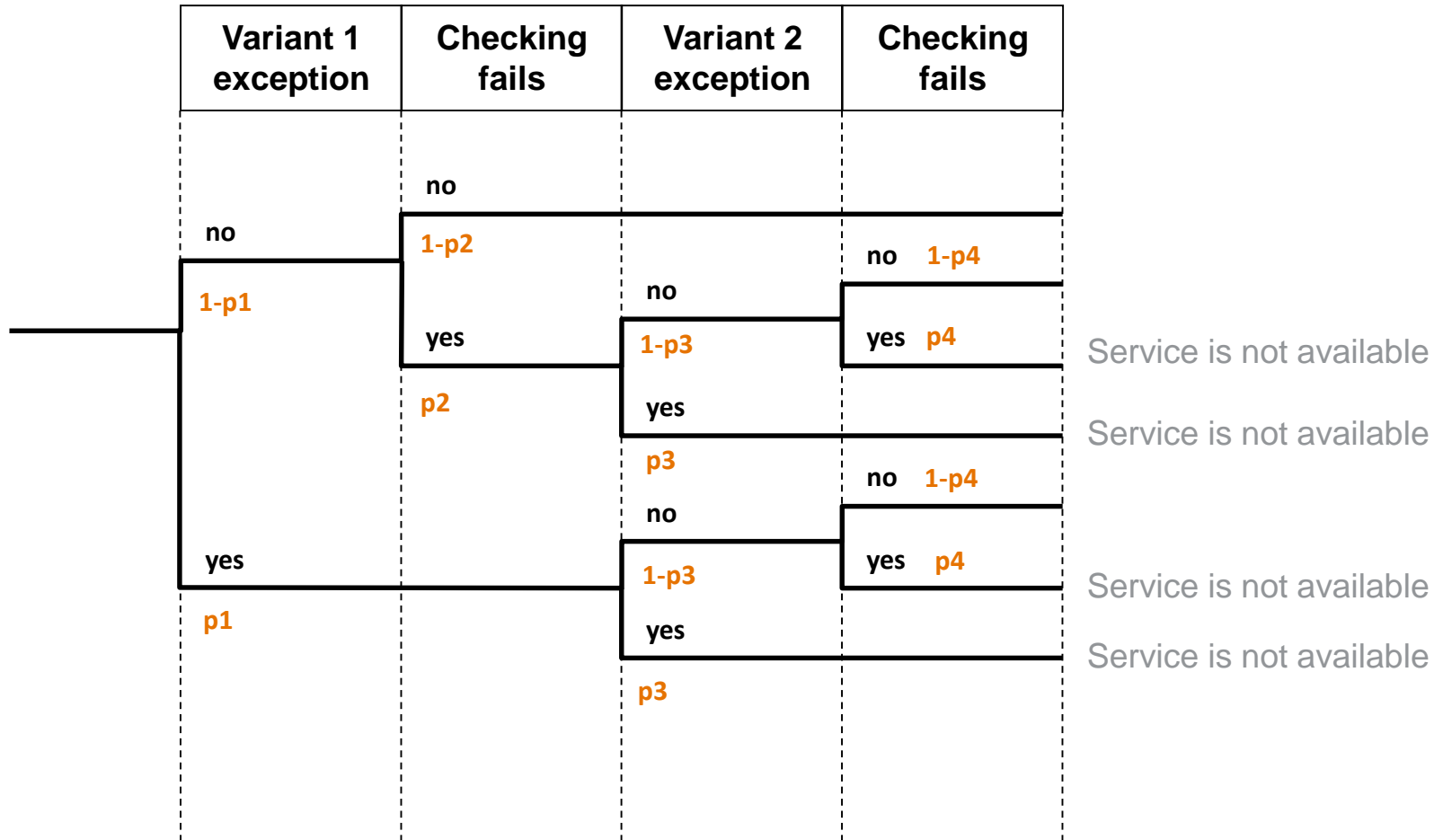


# Event tree example: Reactor cooling





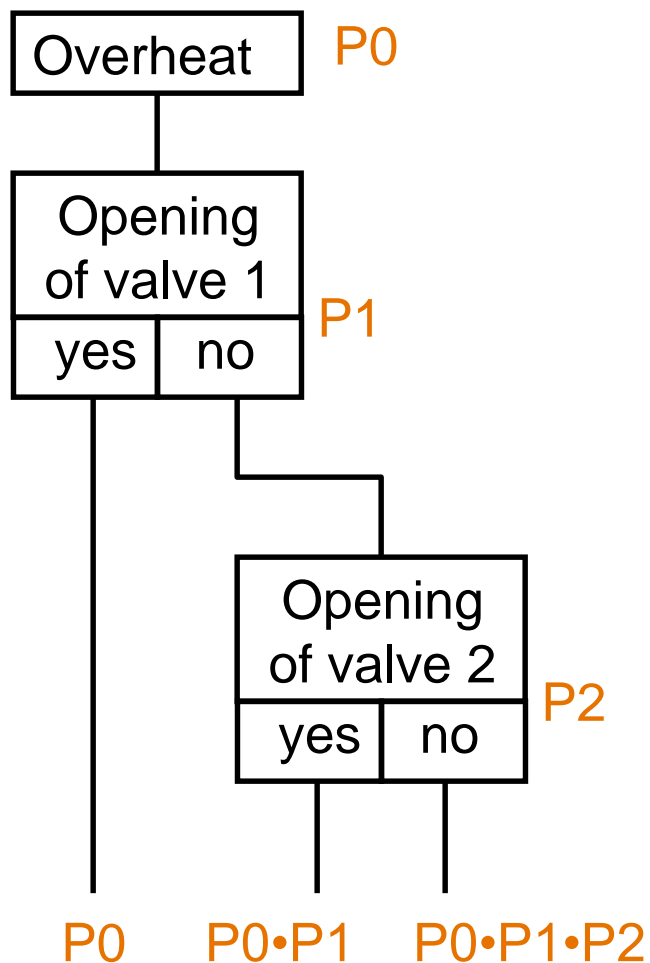
# Event tree example: Recovery blocks (RB)



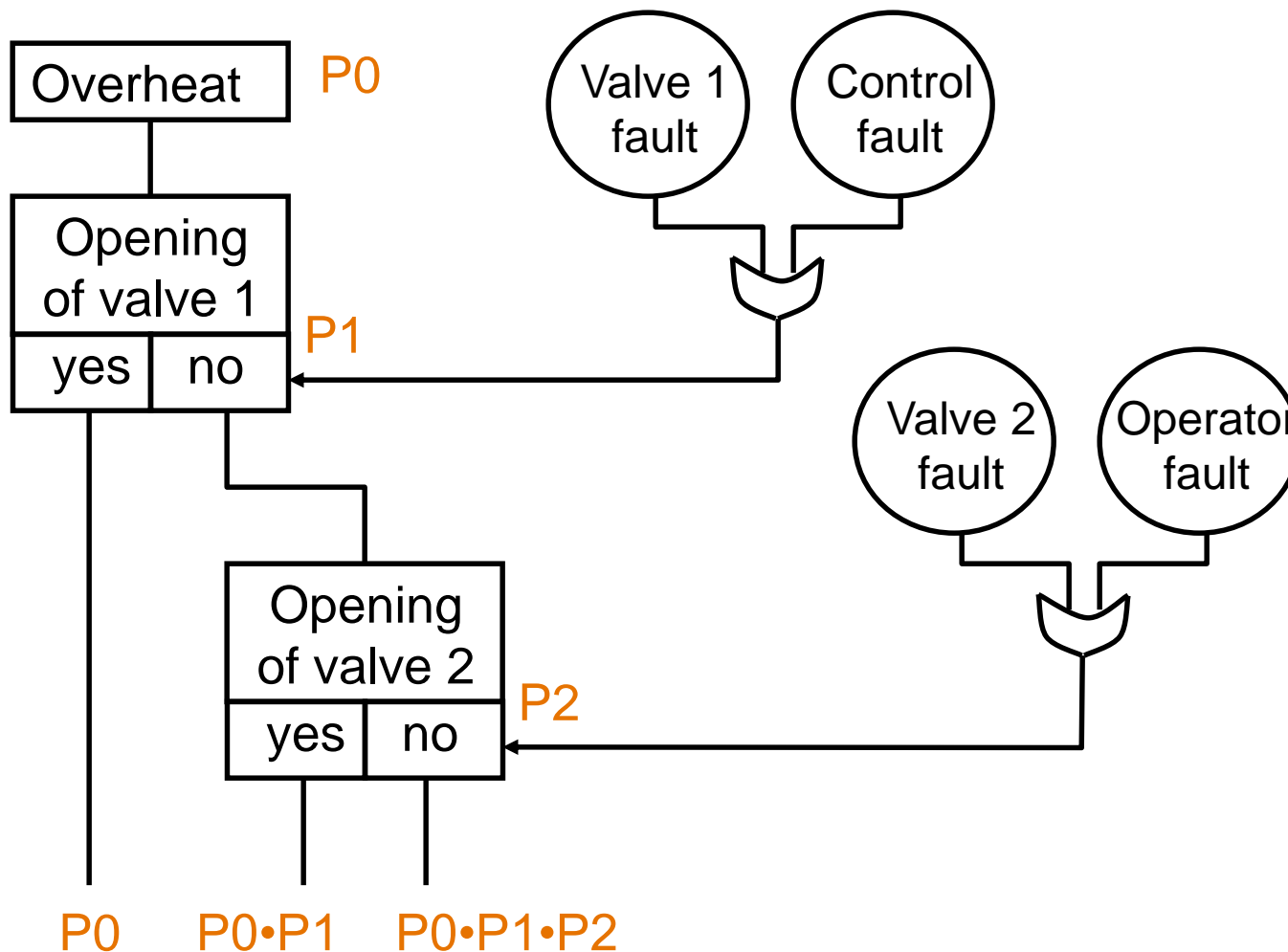
# 4. Cause-consequence analysis

- Integration of an **event tree** with **fault trees**
  - Event tree: **event sequences** (scenarios)
  - Attached fault trees: analysis of the **causes** of events
- Advantages:
  - **Event sequences** (forward analysis) and analysis of **causal relations** (backward analysis) together
- Limitations:
  - Separate diagram for each initial event
  - Complexity

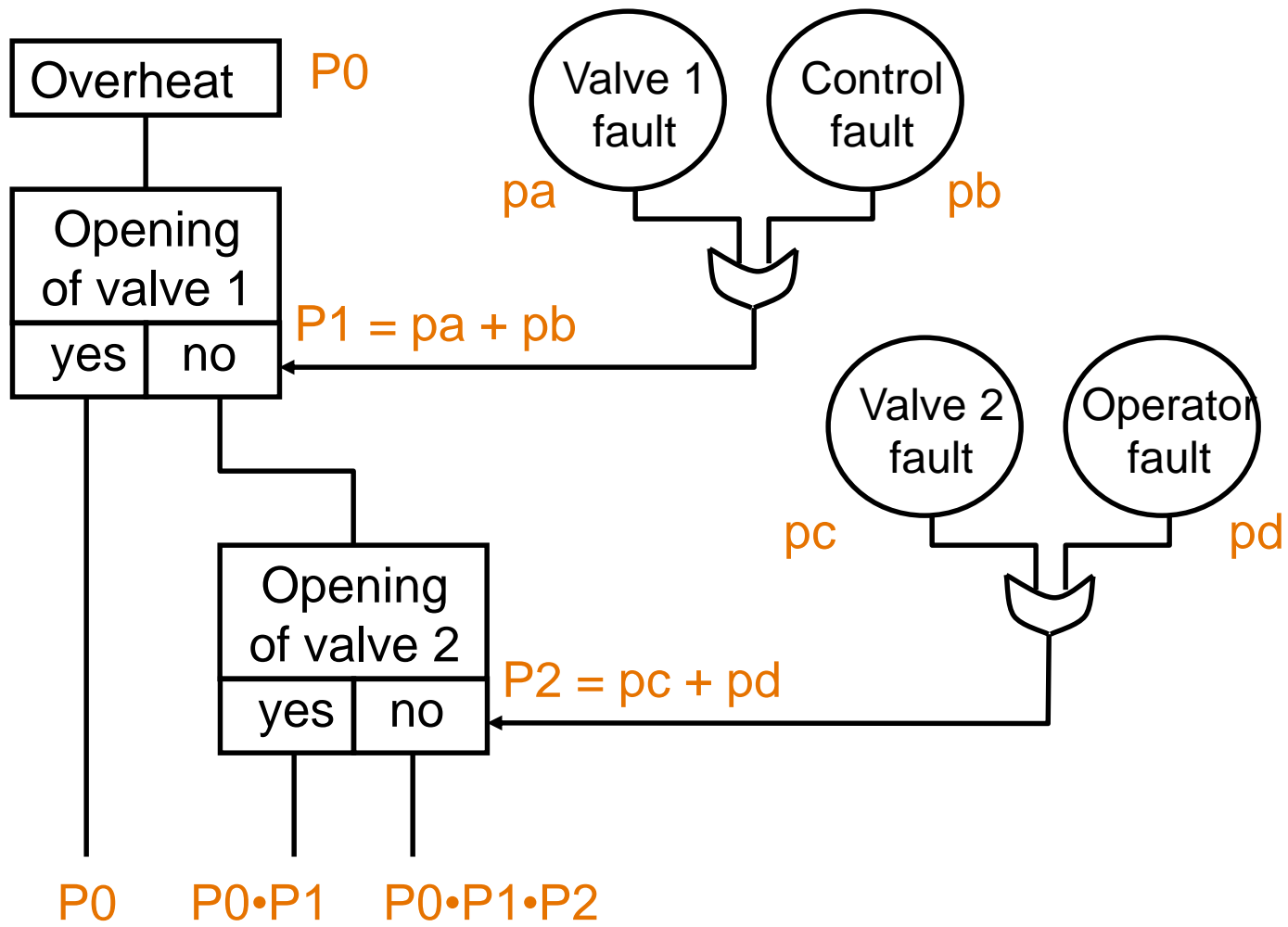
# Cause-consequence analysis example



# Cause-consequence analysis example



# Cause-consequence analysis example

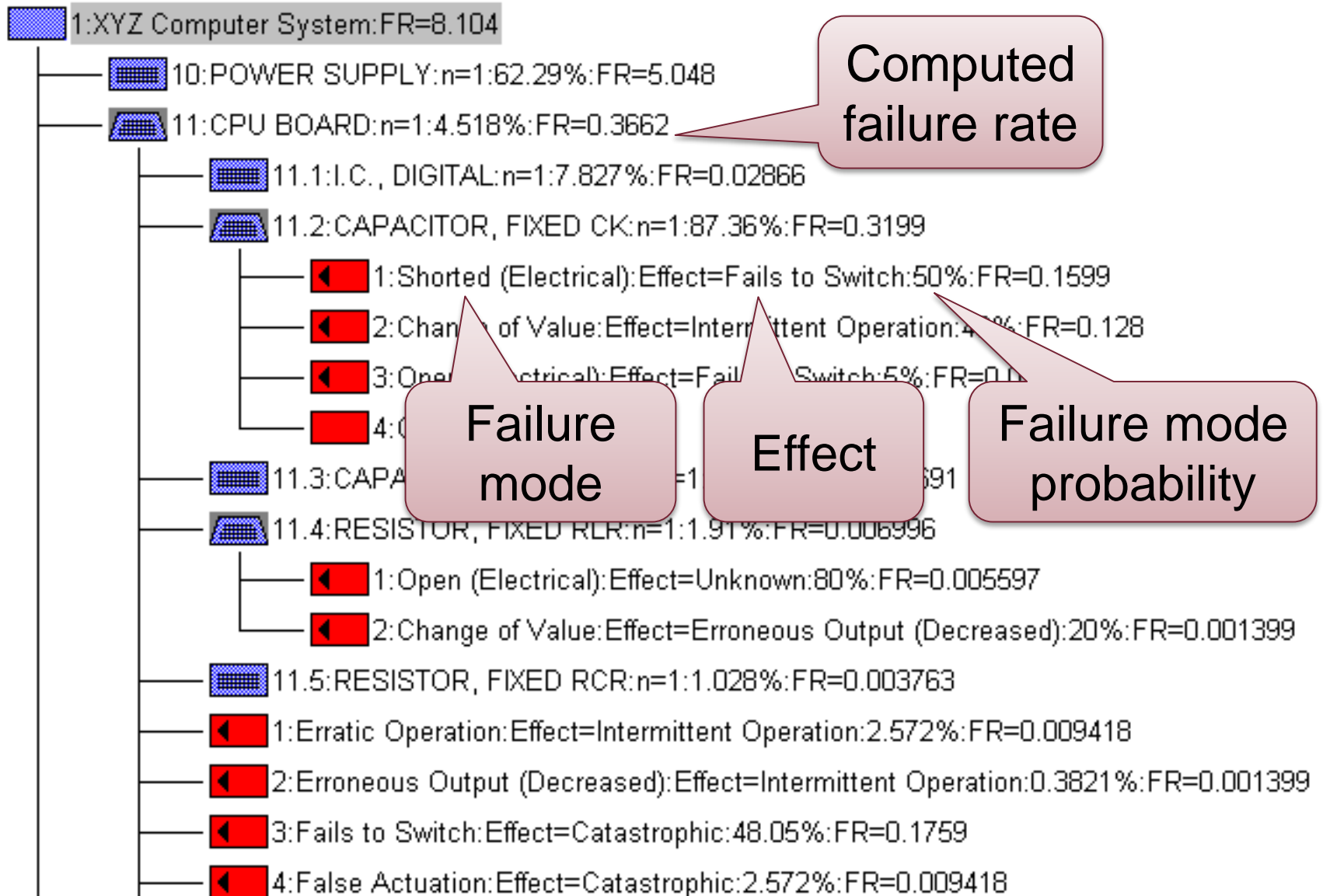


# 5. Failure modes and effects analysis (FMEA)

- Systematic investigation of component **failure modes** and their **effects**
- Advantages:
  - Known faults of components are included
  - Criticalities of effects can also be estimated (FMECA)

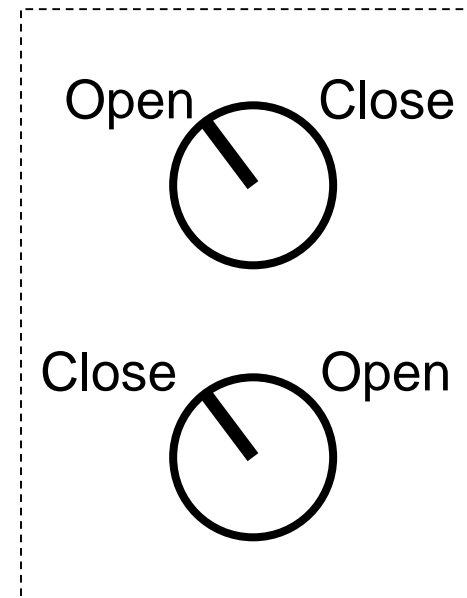
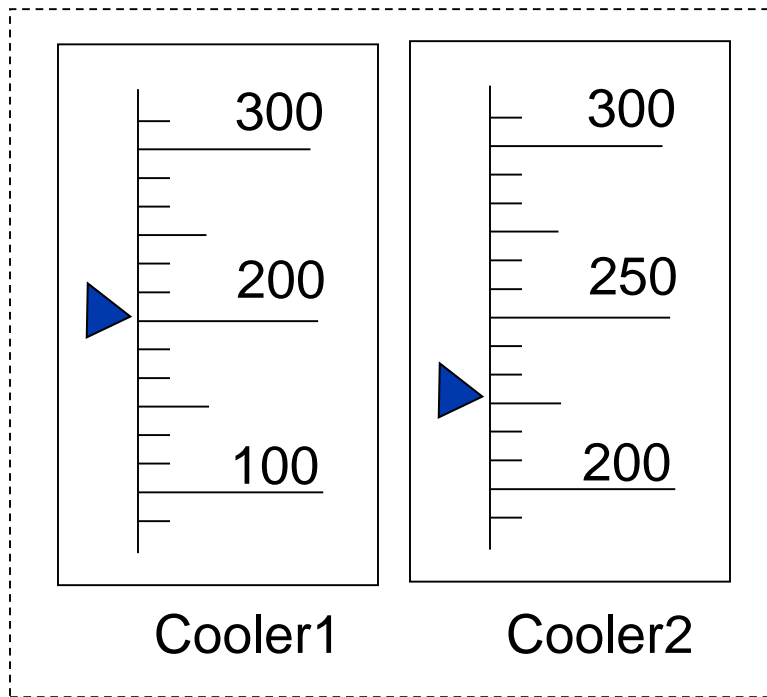
Component	Failure mode	Probability	Effect
D1 diode	open circuit	65%	- over-heating - damaged product
	short circuit	35%	
...	...	...	...

# Example: Analysis of a computer system



# Analysis of operator faults

- Qualitative techniques:
  - Operation – hazards – effects – causes – mitigations
  - Analysis of physical and mental demands
  - Fault causes ← human-machine interface problems





# Catalogue of hazards

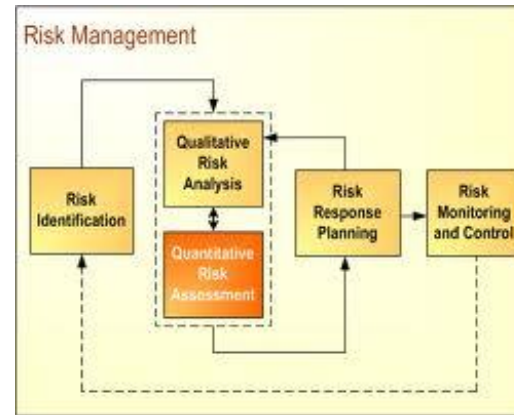
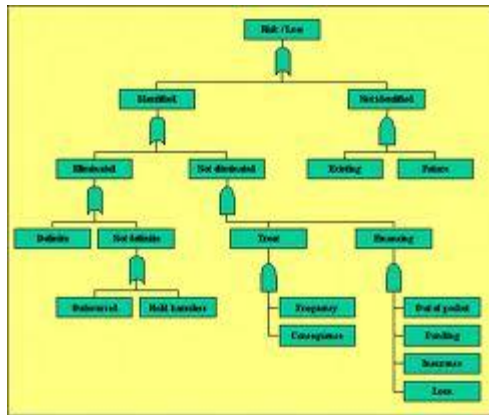
- Categorization of hazards on the basis of hazard analysis (e.g., MIL-STD-822b, NASA):
  - **Severity level** of hazard consequences:  
Catastrophic, critical, marginal, insignificant
  - **Frequency of occurrence** of hazards:  
Frequent, probable, occasional, remote, improbable, incredible
- Identification of **risks**
- Output of the severity/frequency analysis:
  - **Risk matrix**
  - **Protection level**: Identifies the risks to be handled

# Example: Risk matrix (railway control systems)

	Frequency of Occurrence of a Hazardous Event	RISK LEVELS			
Daily to monthly	<b>FREQUENT (FRE)</b>	Undesirable (UND)	Intolerable (INT)	Intolerable (INT)	Intolerable (INT)
Monthly to yearly	<b>PROBABLE (PRO)</b>	Tolerable (TOL)	Undesirable (UND)	Intolerable (INT)	Intolerable (INT)
Between once a year and once per 10 years	<b>OCCASIONAL (OCC)</b>	Tolerable (TOL)	Undesirable (UND)	Undesirable (UND)	Intolerable (INT)
Between once per 10 years and once per 100 years	<b>REMOTE (REM)</b>	Negligible (NEG)	Tolerable (TOL)	Undesirable (UND)	Undesirable (UND)
Less than once per 100 years	<b>IMPROBABLE (IMP)</b>	Negligible (NEG)	Negligible (NEG)	Tolerable (TOL)	Tolerable (TOL)
	<b>INCREDIBLE (INC)</b>	Negligible (NEG)	Negligible (NEG)	Negligible (NEG)	Negligible (NEG)
		<b>INSIGNIFICANT (INS)</b>	<b>MARGINAL (MAR)</b>	<b>CRITICAL (CRI)</b>	<b>CATASTROPHIC (CAT)</b>
		<b>Severity Levels of Hazard Consequence</b>			

# Examples of risk reduction requirements

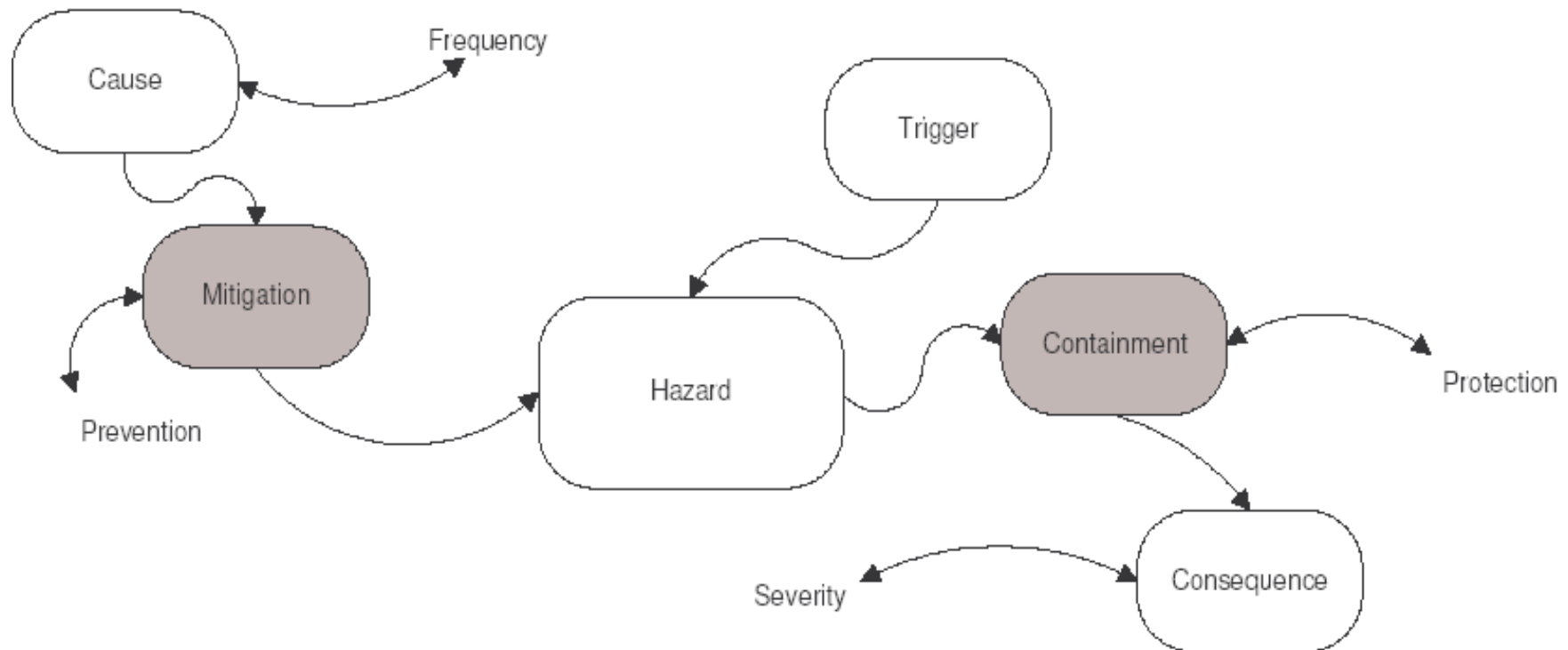
- In case of **catastrophic consequence**:
  - **Improbable** or lower frequency of occurrence is needed
- In case of **critical consequence**:
  - **Improbable** or lower frequency of occurrence is needed
- In case of **marginal consequence**:
  - **Remote** or lower frequency of occurrence is needed
- In case of **insignificant consequence**:
  - **Occasional** or lower frequency of occurrence is needed



# Risk reduction Techniques

# Basic idea for risk reduction

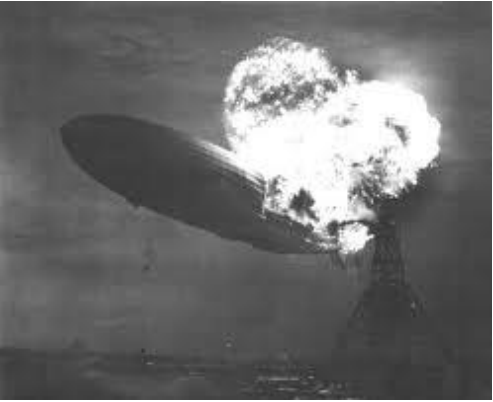
- Mitigation (or prevention) of causes
- Containment (or protection) of consequences




# Risk reduction principles (overview)

1. **Hazard elimination**: Assuring safety by eliminating hazards
  - Substitution
  - Simplification
  - Decoupling
  - Eliminating human errors
2. **Hazard reduction**: Reducing the occurrence rate of hazards
  - Design for controllability
  - Barriers: Lockouts, lockins, interlocks
  - Failure minimization: Safety margins, redundancy
3. **Hazard control**: Reducing the likelihood of an accident
  - Reducing exposure
  - Isolation and containment
  - Protection systems and fail-safe design
4. **Damage minimization**: Reducing the consequences
  - Planning alarming and escape routes
  - Determining “point of no return”

# 1. Hazard elimination


Generic method	Hardware solution	Software solution
<p data-bbox="48 396 488 454">a. Substitution</p> 	<ul data-bbox="672 396 1257 675" style="list-style-type: none"><li>■ Using safer material, component, technology, ...</li></ul> <p data-bbox="672 789 1180 996">E.g., substitution of flammable or toxic materials</p>	<ul data-bbox="1296 396 1779 1003" style="list-style-type: none"><li>■ More safe programming language (e.g., SPARK Ada instead of C)</li><li>■ Using well-tried modules (proven in use)</li></ul>

# 1. Hazard elimination


Generic method	Hardware solution	Software solution
<p data-bbox="48 396 537 461"><b>b. Simplification</b></p> 	<ul data-bbox="672 396 1161 846" style="list-style-type: none"><li>■ Reducing the number of components</li><li>■ Reducing the number of operating modes</li></ul> <p data-bbox="672 975 1064 1103">Flexibility ↔ simplification</p> <p data-bbox="672 1146 1145 1275">Fault tolerance ↔ simplification</p>	<p data-bbox="1296 396 1798 604">Simple program structure (testable, analyzable):</p> <ul data-bbox="1296 646 1734 1268" style="list-style-type: none"><li>■ Deterministic, static control</li><li>■ Structured programming</li><li>■ Simple interfaces</li><li>■ Robust data structures</li></ul>




# 1. Hazard elimination

Generic method	Hardware solution	Software solution
<p data-bbox="48 396 459 464">c. Decoupling</p> 	<p data-bbox="672 396 1168 753">Elimination of dependences and unnecessary interactions (error propagation paths)</p> <p data-bbox="672 882 1072 1092">E.g., firebreaks, overpasses and underpasses</p>	<p data-bbox="1296 396 1767 521">“Loosely coupled” software:</p> <ul data-bbox="1296 564 1825 1249" style="list-style-type: none"><li data-bbox="1296 564 1738 692">■ Modularization (safety kernel)</li><li data-bbox="1296 728 1825 935">■ Information hiding (well-defined interfaces)</li><li data-bbox="1296 971 1825 1249">■ Separation of safety-critical and non-safety-critical functions</li></ul>


# 1. Hazard elimination

Generic method	Hardware solution	Software solution
<p data-bbox="48 396 465 544">d. Eliminating human errors</p> 	<p data-bbox="672 396 1141 672">Masterability, understandability, maintainability, checkability</p> <ul data-bbox="672 715 1232 1186" style="list-style-type: none"><li>■ Ergonomic interfaces</li><li>■ No interchangeable connectors</li><li>■ Color codes</li><li>■ ...</li></ul>	<p data-bbox="1296 396 1827 594">Limiting fault prone features in language subsets</p> <ul data-bbox="1296 636 1850 879" style="list-style-type: none"><li>■ Pointers,</li><li>■ Implicit conversion,</li><li>■ Overloading, ...</li></ul> <p data-bbox="1296 908 1812 1036">Simple human-machine interfaces:</p> <ul data-bbox="1296 1079 1763 1300" style="list-style-type: none"><li>■ Clear operation modes</li><li>■ Tolerable timing</li></ul>

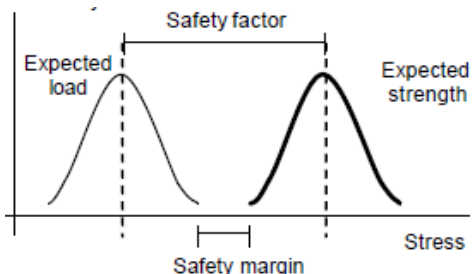

## 2. Hazard reduction

Generic method	Hardware solution	Software solution
<p data-bbox="48 396 473 725">a. Design for controllability (active hazard reduction)</p> 	<ul data-bbox="608 396 1188 1163" style="list-style-type: none"><li>■ Allowing actions to provide protection in case of hazards</li><li>■ Detection, diagnosis and controlled response</li><li>■ E.g., mechanical control systems (backup), multiple control modes, ...</li></ul>	<ul data-bbox="1246 396 1845 1220" style="list-style-type: none"><li>■ Incremental control: Feedback and corrections</li><li>■ Monitoring hazards and conditions:<ul data-bbox="1333 806 1835 1220" style="list-style-type: none"><li>- Sanity check</li><li>- Monitor-actuator</li><li>- Watchdog</li><li>- Safety executive architecture patterns</li></ul></li></ul>

## 2. Hazard reduction

Generic method	Hardware solution	Software solution
<p data-bbox="48 396 513 635">b. Barriers (passive hazard reduction)</p> 	<ul data-bbox="595 396 1155 1239" style="list-style-type: none"><li>■ Lockout: Making access to dangerous state difficult (wall, fence)</li><li>■ Lockin: Make leaving a safe state difficult (safe area)</li><li>■ Interlock: Enforce a safe sequence of actions</li></ul>	<ul data-bbox="1211 396 1875 1239" style="list-style-type: none"><li>■ Lockout: Access control, authorization, acknowledgements</li><li>■ Lockin: Checking inputs, requests, accesses</li><li>■ Interlock: Checking call sequences, synchronization (baton)</li></ul>

# 2. Hazard reduction

Generic method	Hardware solution	Software solution
<p data-bbox="48 392 444 542"><b>c. Failure minimization</b></p>  	<ul data-bbox="627 392 1207 842" style="list-style-type: none"><li>■ Robust components</li><li>■ Safety factors, safety margins (e.g., higher load does not cause failure)</li></ul> <p data-bbox="627 871 1207 1049">Safety factor: Ratio expected strength and expected (nominal) stress</p> <p data-bbox="627 1071 1207 1313">Safety margin: Difference of minimum probable strength and maximum probable stress</p>	<ul data-bbox="1246 392 1883 1013" style="list-style-type: none"><li>■ Robustness</li><li>■ Redundancy (diverse instances)</li><li>■ Fault tolerance: Forward recovery is preferred (guarantees for execution)</li></ul>

# 3. Hazard control

Generic method	Hardware solution	Software solution
a. Reducing exposure	<ul style="list-style-type: none"><li>■ Staying in higher risk state as short as possible</li><li>■ Timely return to safe state</li></ul>	<ul style="list-style-type: none"><li>■ Safe initial state</li><li>■ Keeping synchronization with the environment to return to safe state</li></ul>
b. Isolation and containment	<ul style="list-style-type: none"><li>■ Isolation in time and space</li></ul>	<ul style="list-style-type: none"><li>■ Partitioning of safety functions</li></ul>
c. Protection systems	<ul style="list-style-type: none"><li>■ Moving the system to safe state</li></ul>	<ul style="list-style-type: none"><li>■ Control to safe state</li><li>■ Challenge protocol for protection systems</li></ul>

# 4. Damage minimization

Generic method	Hardware solution	Software solution
a. Planning alarming and escape routes	<ul style="list-style-type: none"><li>■ Alarm devices with periodic testing</li><li>■ Fire escape, lifeboat, abandonment of products</li></ul>	<ul style="list-style-type: none"><li>■ Software controlled alarm</li><li>■ Complex devices with software support (e.g., airbag control)</li></ul>
2. Determining „point of no return”	<ul style="list-style-type: none"><li>■ Turn to damage minimization instead of hazard control</li></ul>	

# Summary

## ■ Hazard analysis

- Checklists
- Fault tree analysis
- Event tree analysis
- Cause-consequence analysis
- Failure modes and effects analysis (FMEA)

## ■ Risk matrix

- Severity level of hazard consequences
- Frequency of hazard occurrence

## ■ Risk reduction techniques

- Hazard elimination, hazard reduction, hazard control, damage minimization