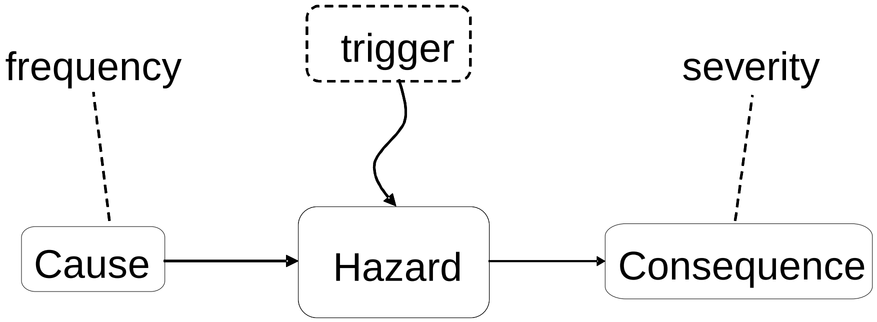
Hazard analysis

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

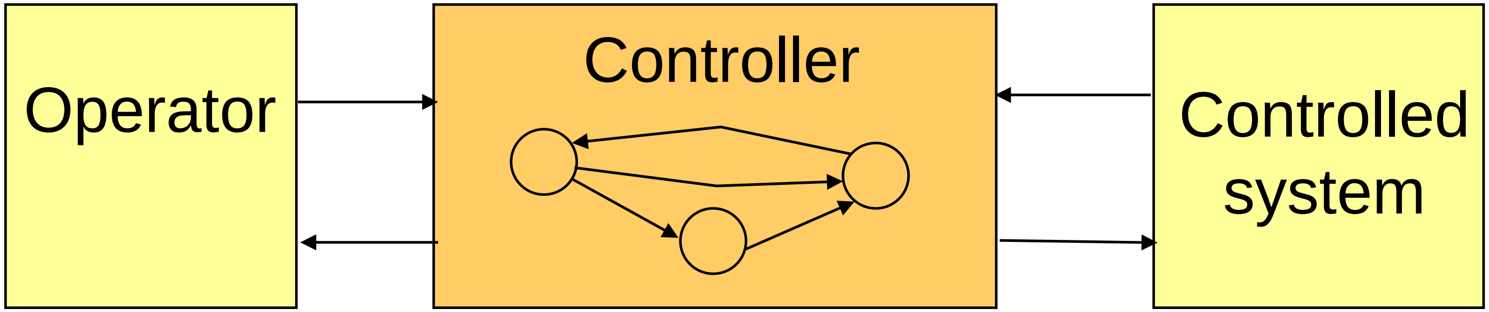
    78% (149/192) specification related failures, from which

    - 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:

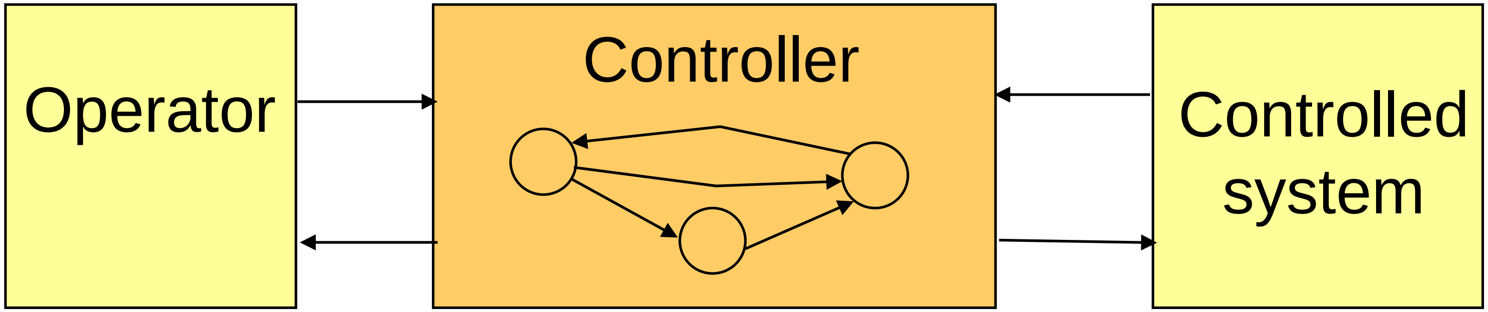
- State definition
- Inputs (trigger events)
- Outputs
- Relation of inputs (triggers) and outputs
- State transitions
- Human-machine interface



Example: Checklist for state machine specifications

- State definition
- Inputs (i...
- Outpu...
- Relat...
- State...
- Human-machine interface

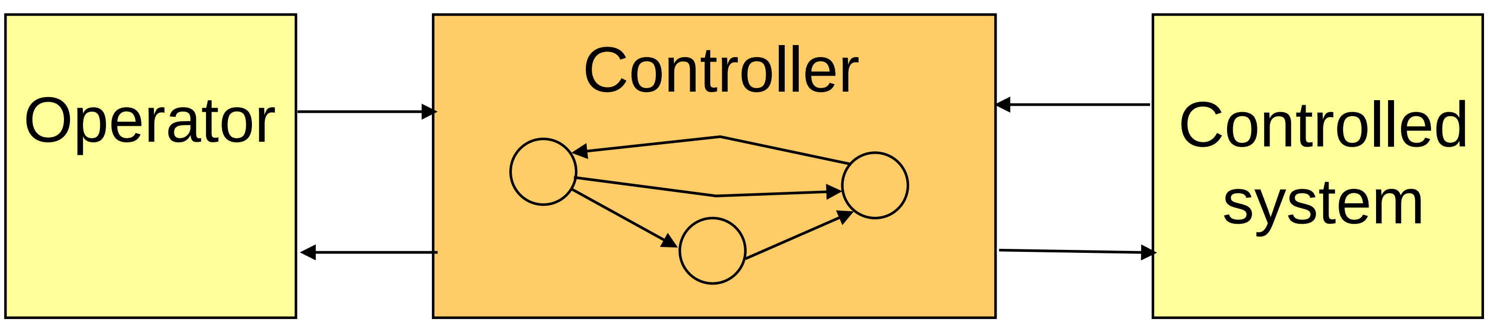
- Safe initial state
- Actualization of the internal model: timeout and transition to “invalid” state if input events are missing; output is not allowed in this state



Example: Checklist for state machine specifications

- State definition
- Inputs (trigger events)
- Outpu
- Relat
- State
- Huma

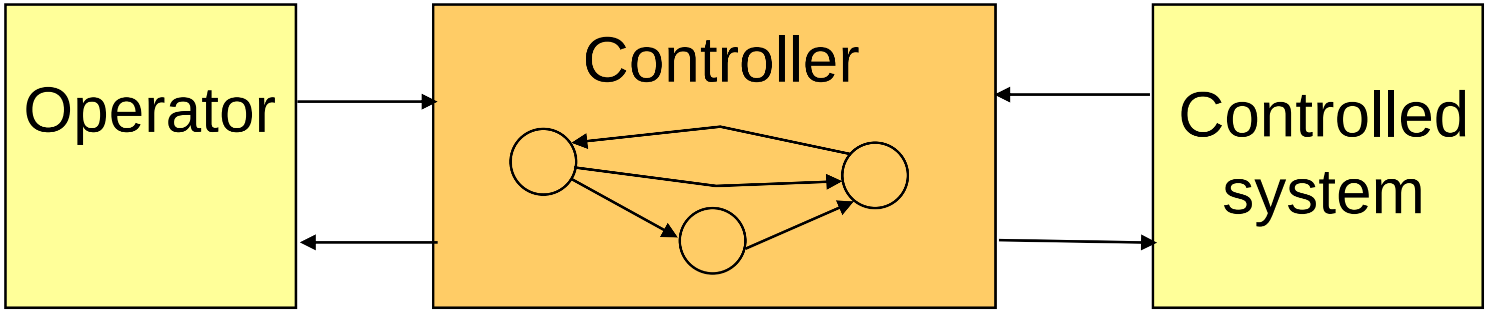
- Reaction to each potential input event
- Deterministic reactions
- Input checking (value, timing)
- Handling of invalid inputs
- Limited rate of interrupts



Example: Checklist for state machine specifications

- State definition
- Inputs (trigger events)
- Outputs
- Relati
- State
- Huma

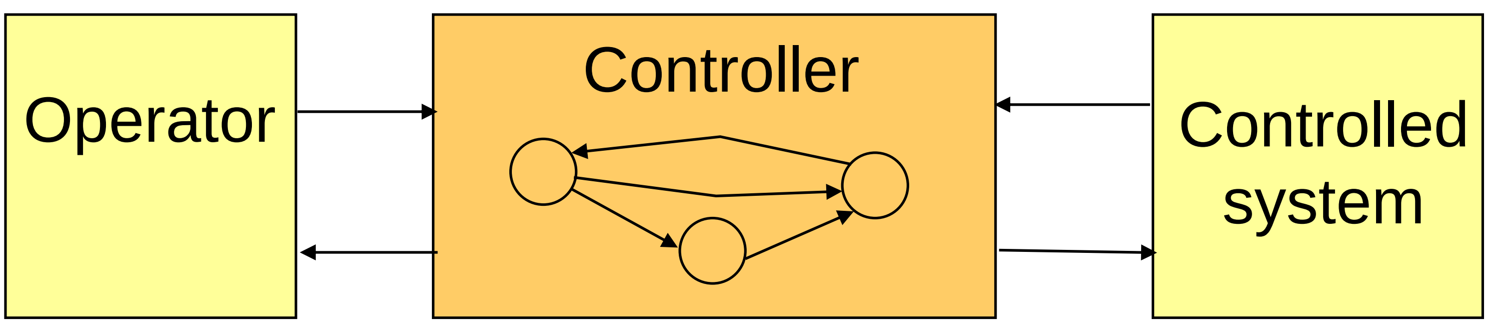
- Acceptance checking on the output
- There are no unused outputs
- Compliance with the limitations of the environment



Example: Checklist for state machine specifications

- State def
- Inputs (tr
- Outputs
- Relation of inputs (triggers) and outputs
- State transitions
- Human-machine interface

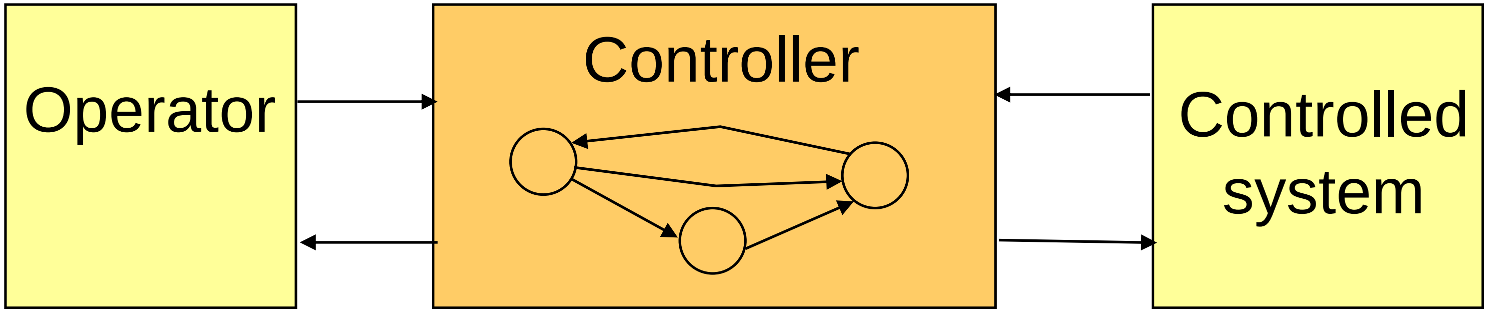
- The effects of outputs are checked through processing the inputs
- Stability of the control loop is preserved



Example: Checklist for state machine specifications

- St
- In
- Ou
- Re
- State transitions
- Human-machine interface

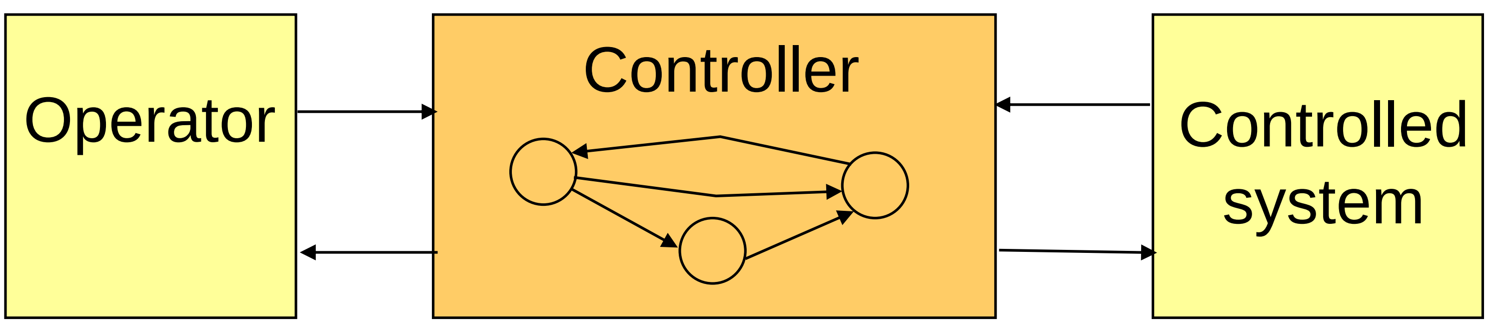
- Each state is reachable (static reachability)
- Transitions are reversible (reverse path exists)
- Multiple transitions from dangerous to safe states
- Confirmed transitions from safe to dangerous states



Example: Checklist for state machine specifications

- Sta
- Inp
- Ou
- Re
- State tran ns
- Human-machine interface

Well-specified outputs towards the operator:
- Ordering (with priorities)
- Update frequency
- Timeliness



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

```c
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

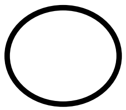
- Top-down analysis
- Identifying the component level combinations of faults/events that may lead to hazard

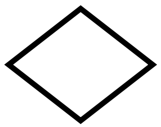
Construction of the fault tree

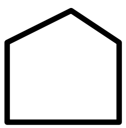
1. Identification of the foreseen system level hazard: on the basis of environment risks, standards etc.
2. Identification of intermediate events (pseudo-events): Boolean (AND, OR) combinations of lower level events that may cause upper level events
3. Identification of primary (basic) events: no further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

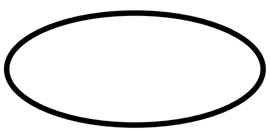
Set of elements in a fault tree

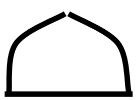

Top level or intermediate event

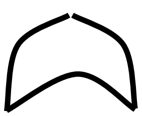

Primary (basic) event


Event without further analysis


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

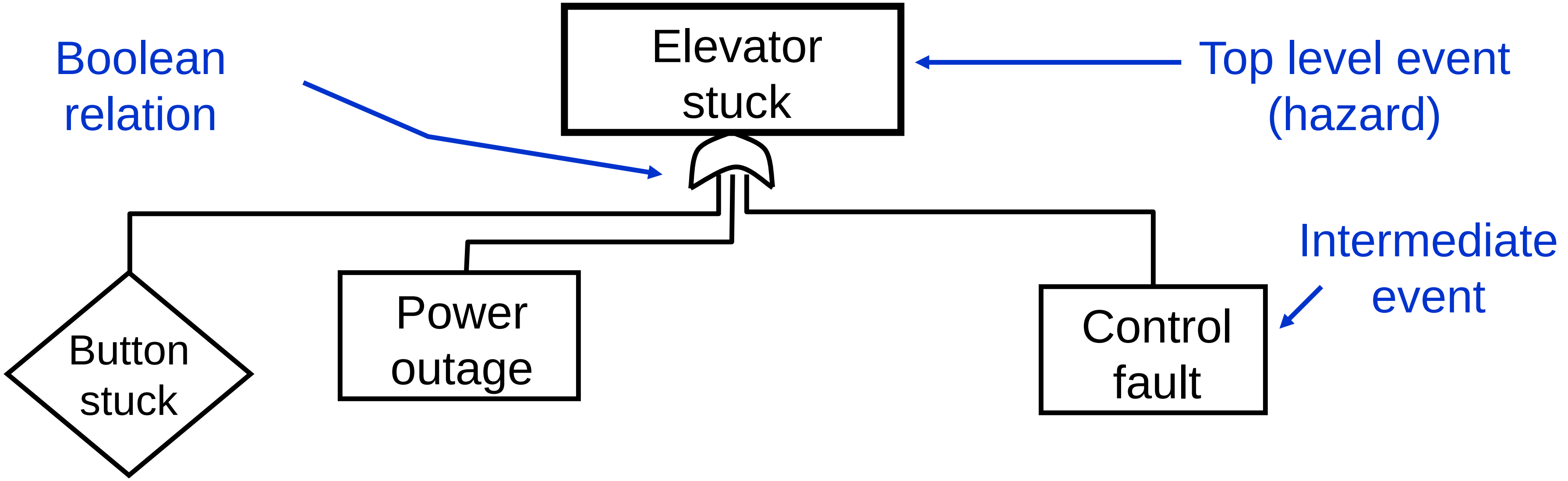

Condition for a composite event


AND combination of events

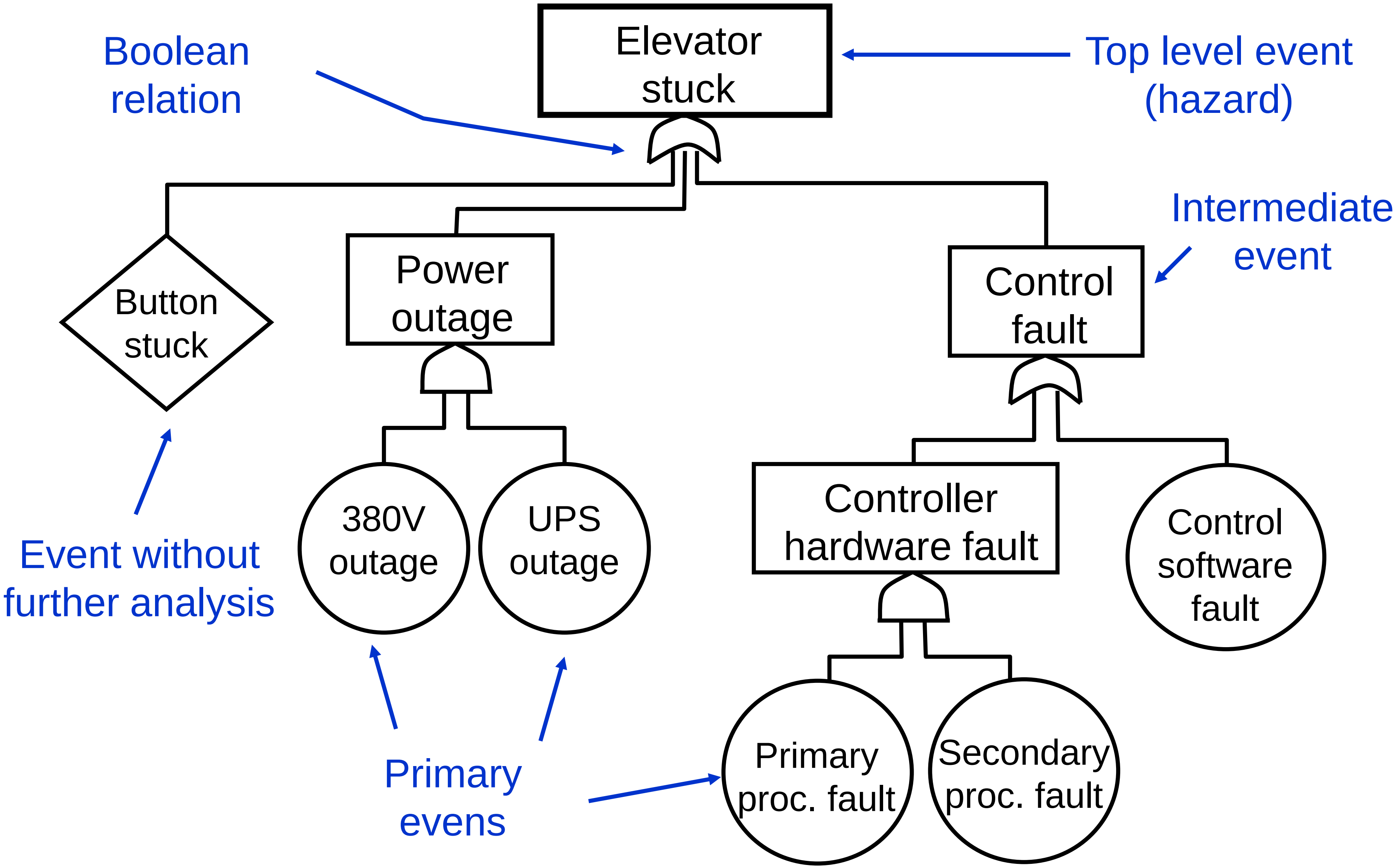

OR combination of events

Fault tree example: Elevator

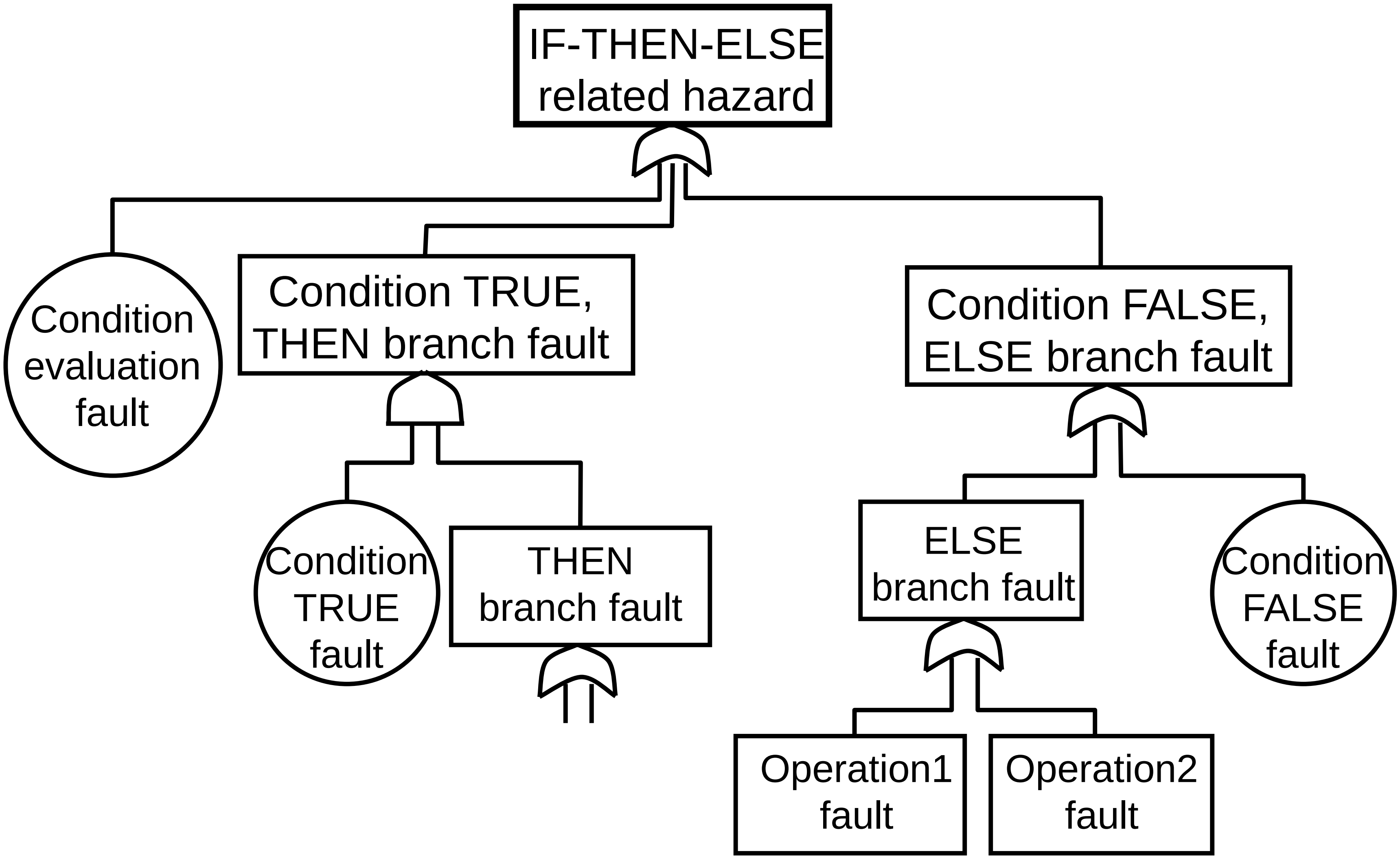
Fault tree example: Elevator



Fault tree example: Elevator



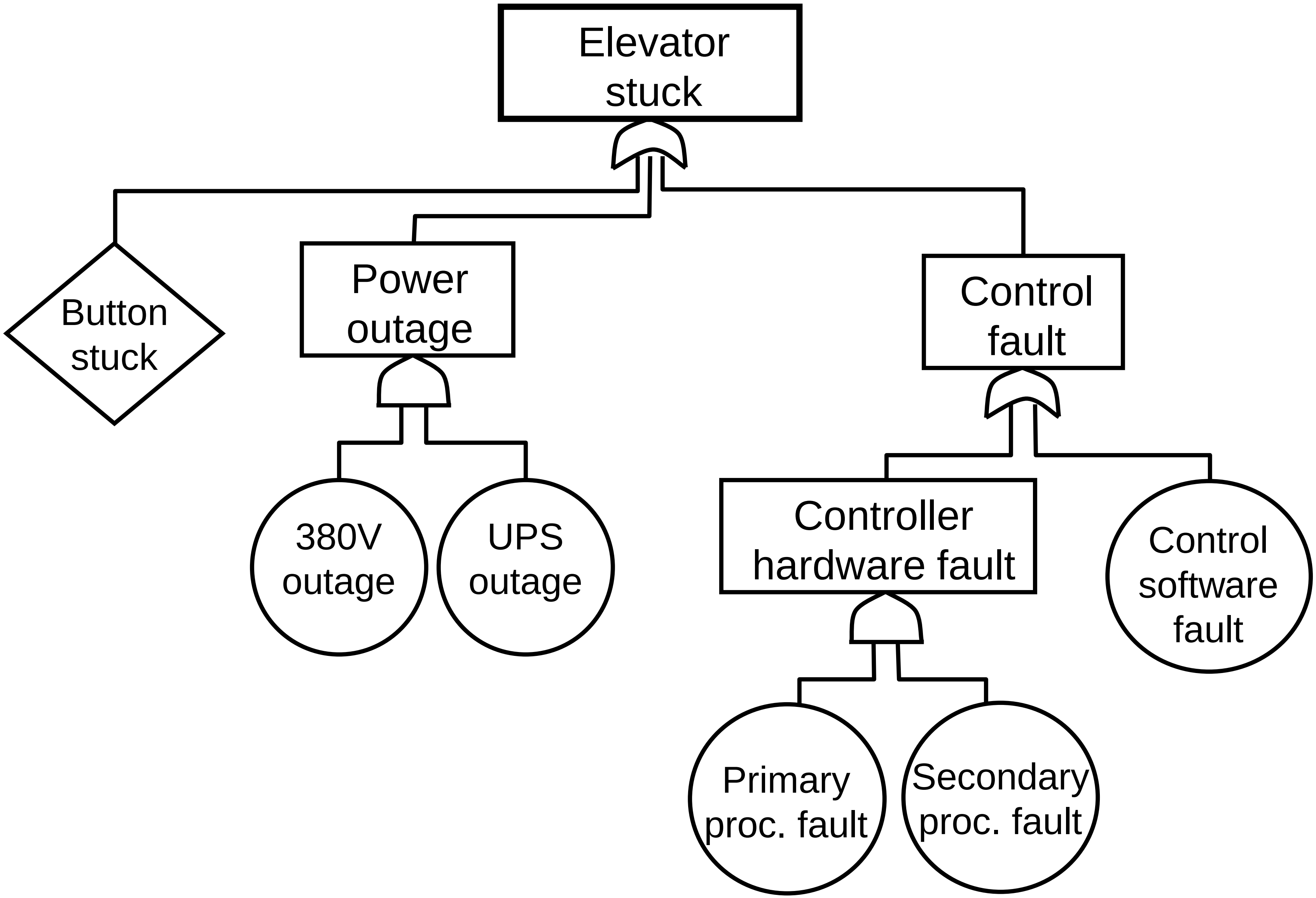
Fault tree example: Software analysis



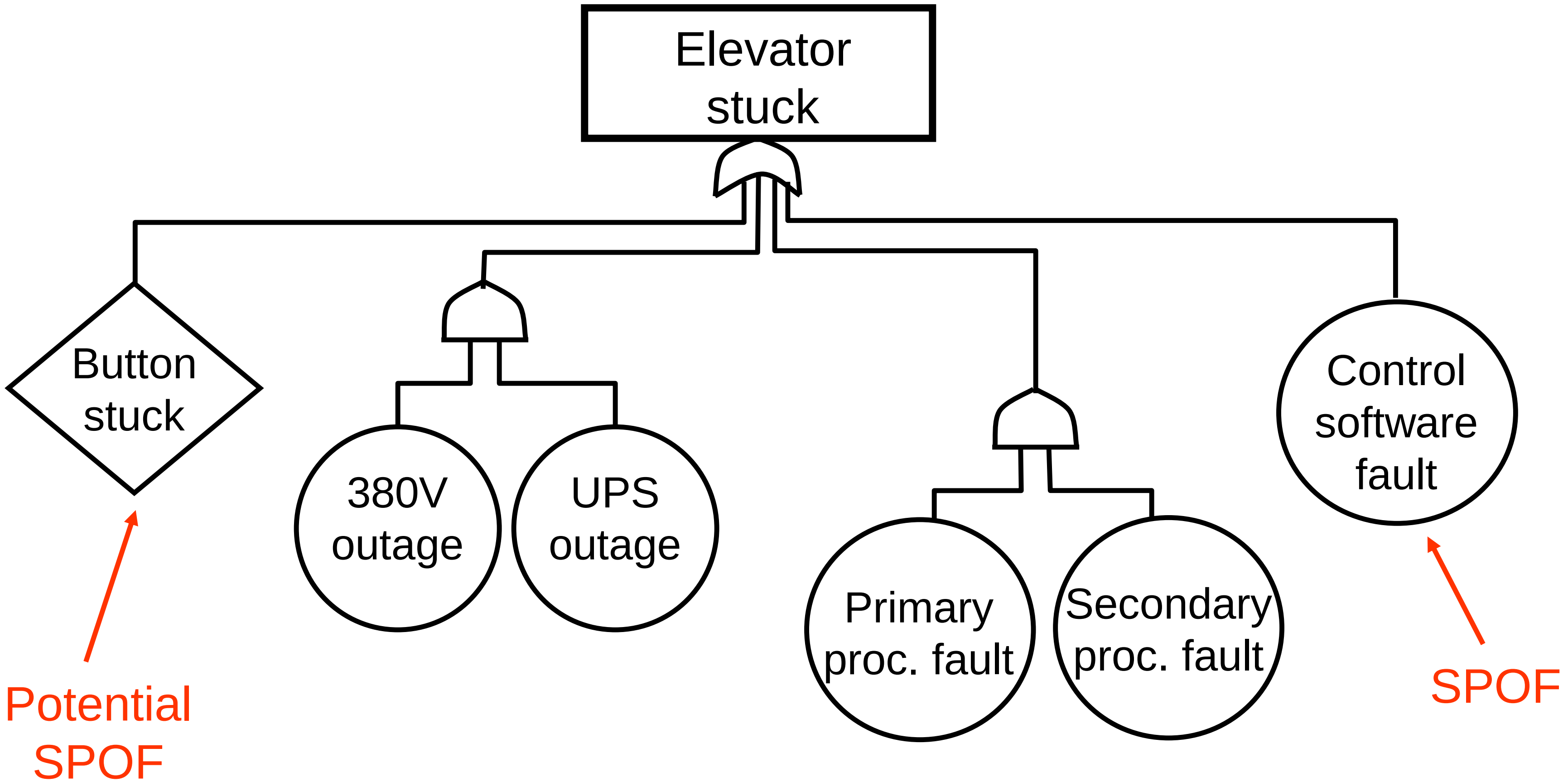
Qualitative analysis of the fault tree

- Fault tree reduction: Resolving intermediate events/pseudo-events using primary events
  $\rightarrow$ 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



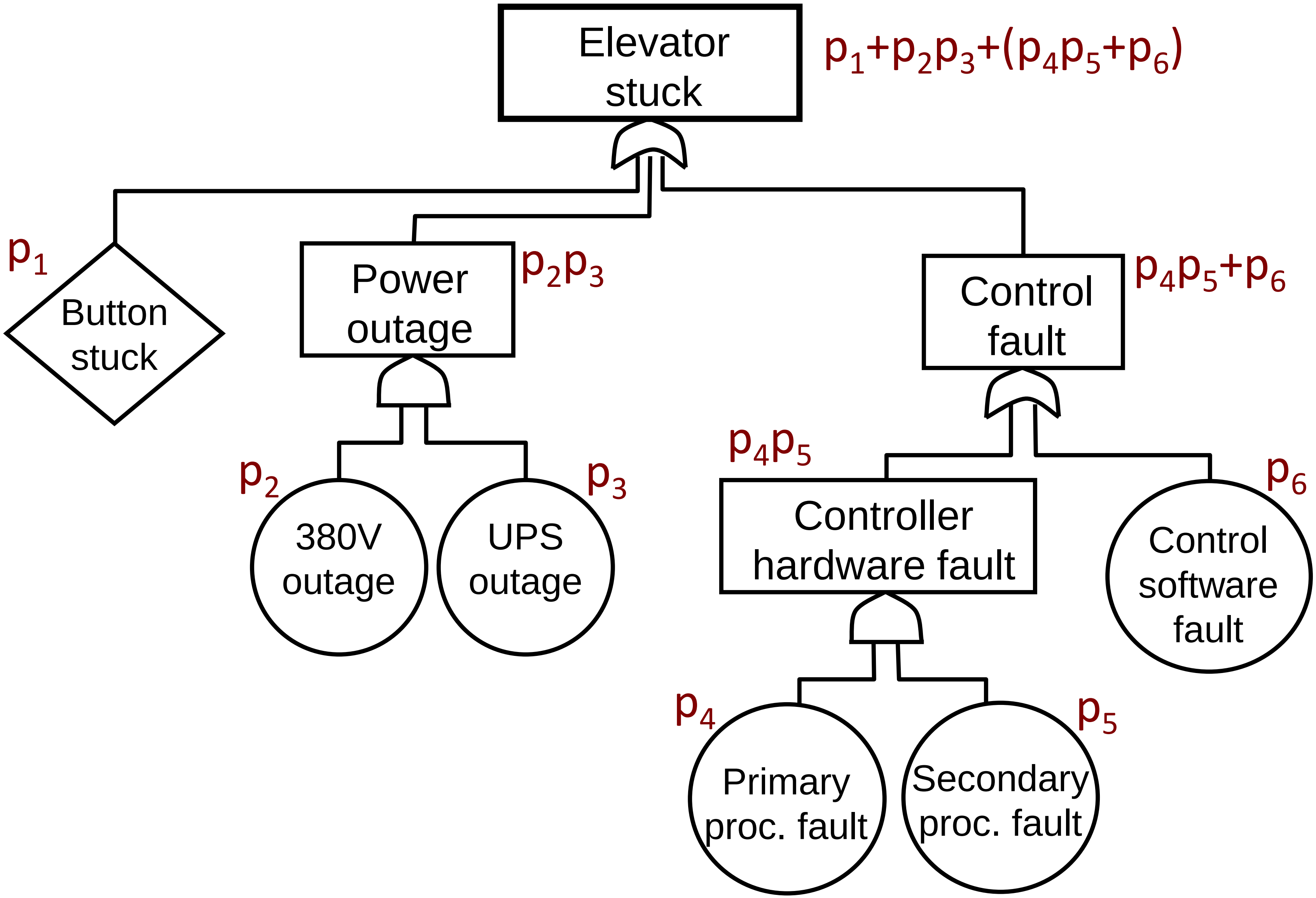
Reduced fault tree of the elevator example



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 and B} = P{A} · P{B|A}
  - OR gate: sum (worst case estimation)
    - Exactly: P{A or B} = P{A}+P{B}-P{A and B} <= P{A}+P{B}
- Typical problems:
  - Correlated faults (not independent)
  - Handling of fault sequences

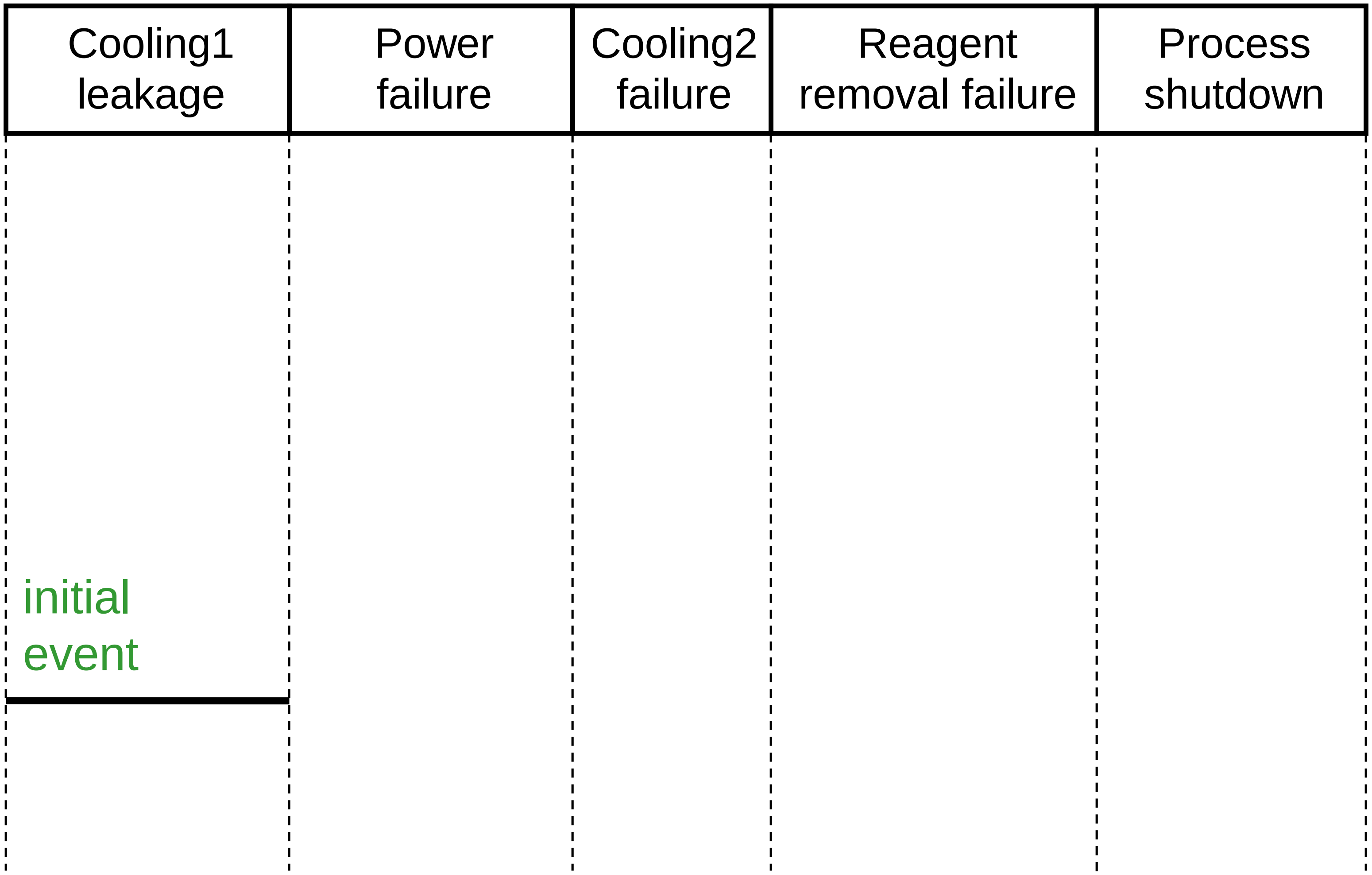
Fault tree of the elevator with probabilities



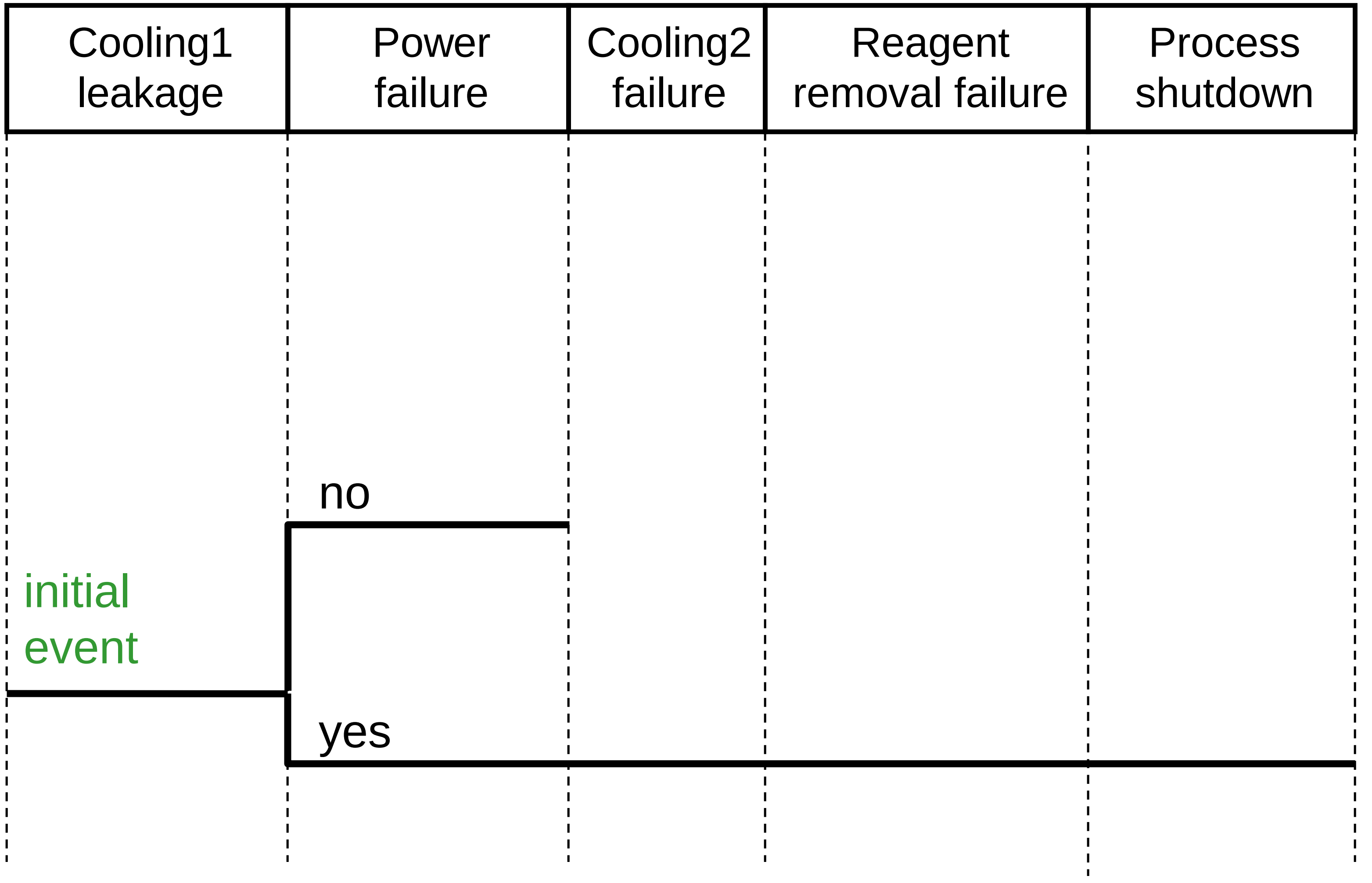
3. Event tree analysis

- 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

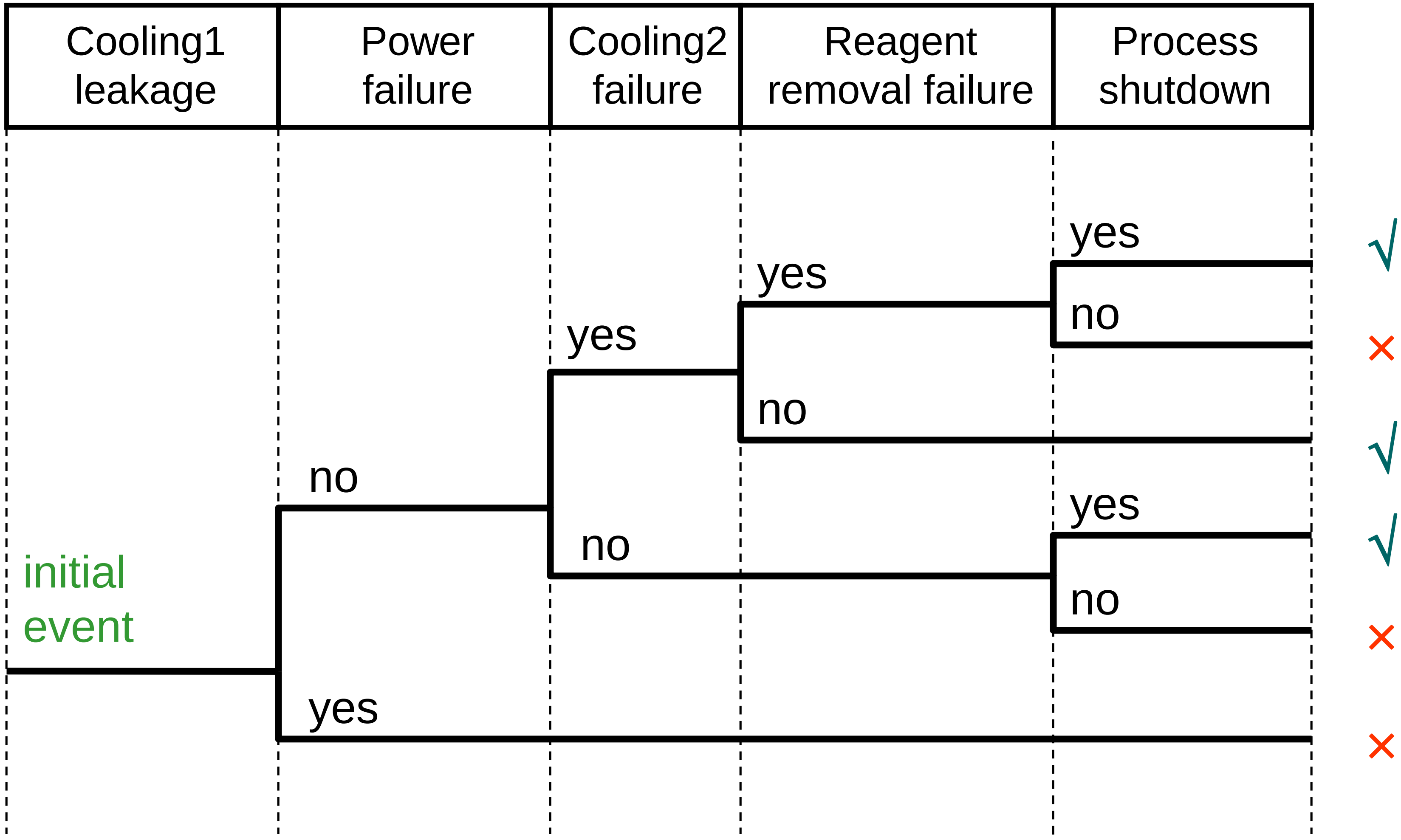
Event tree example: Reactor cooling



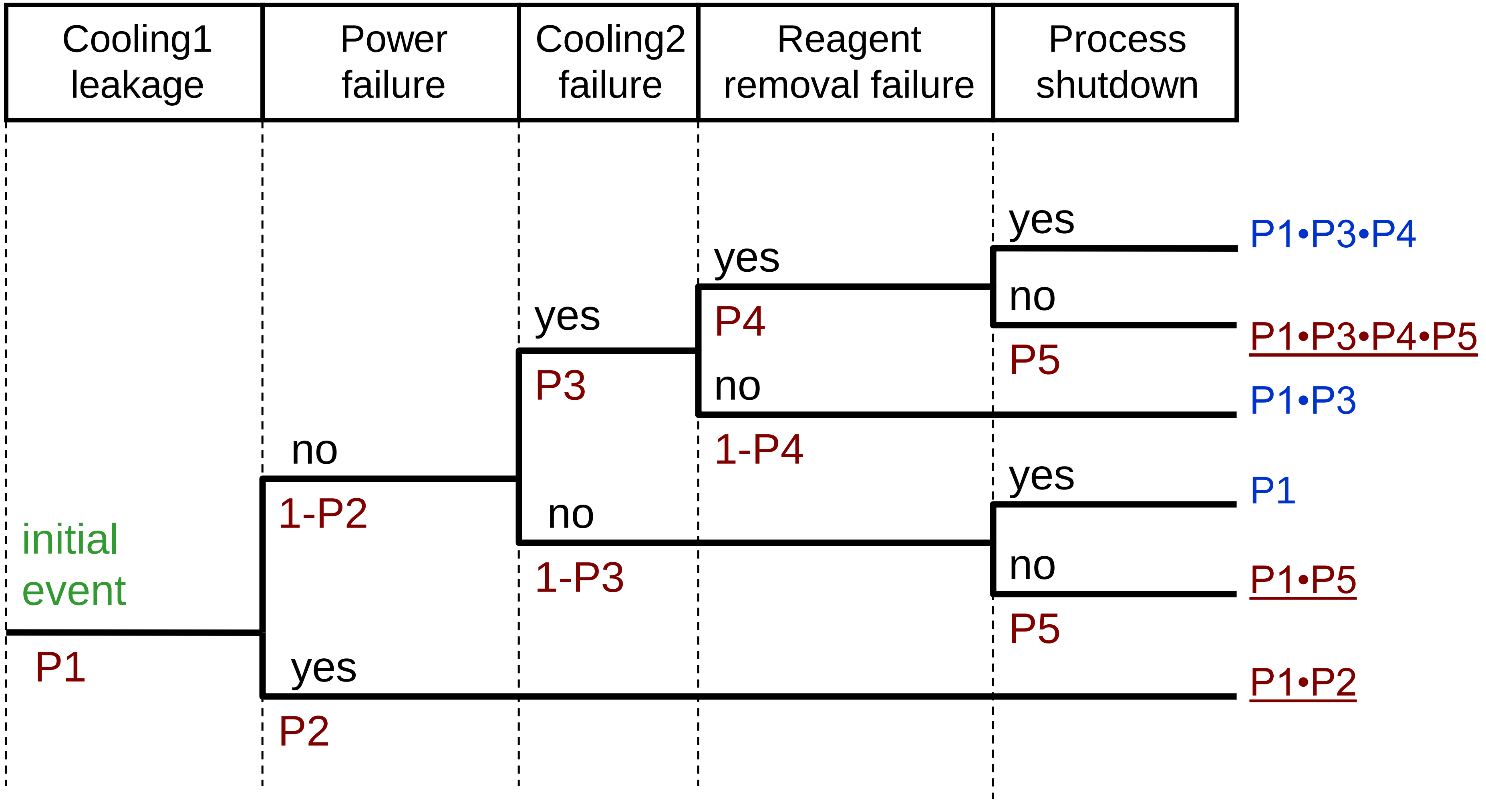
Event tree example: Reactor cooling



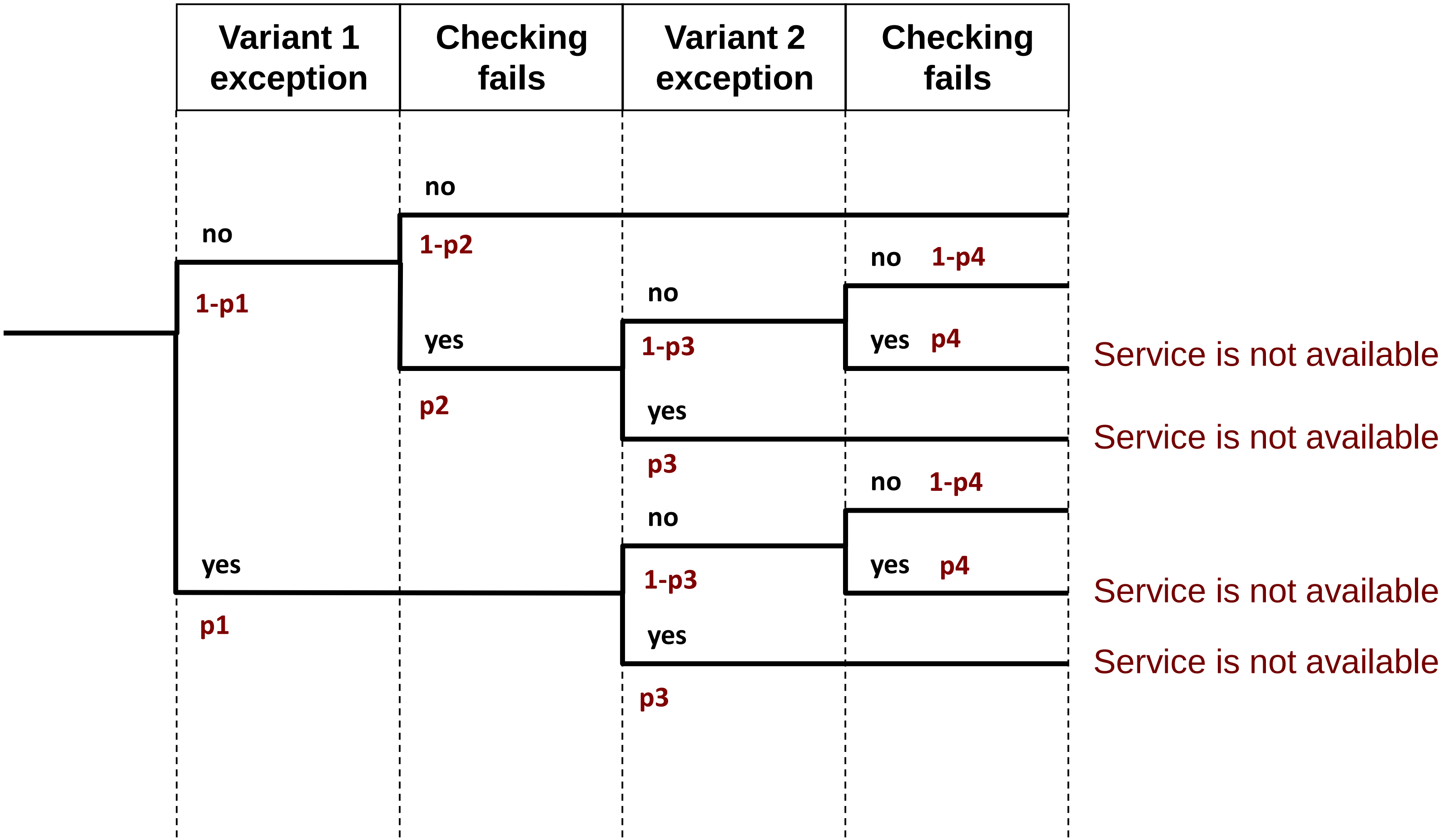
Event tree example: Reactor cooling



Event tree example: Reactor cooling



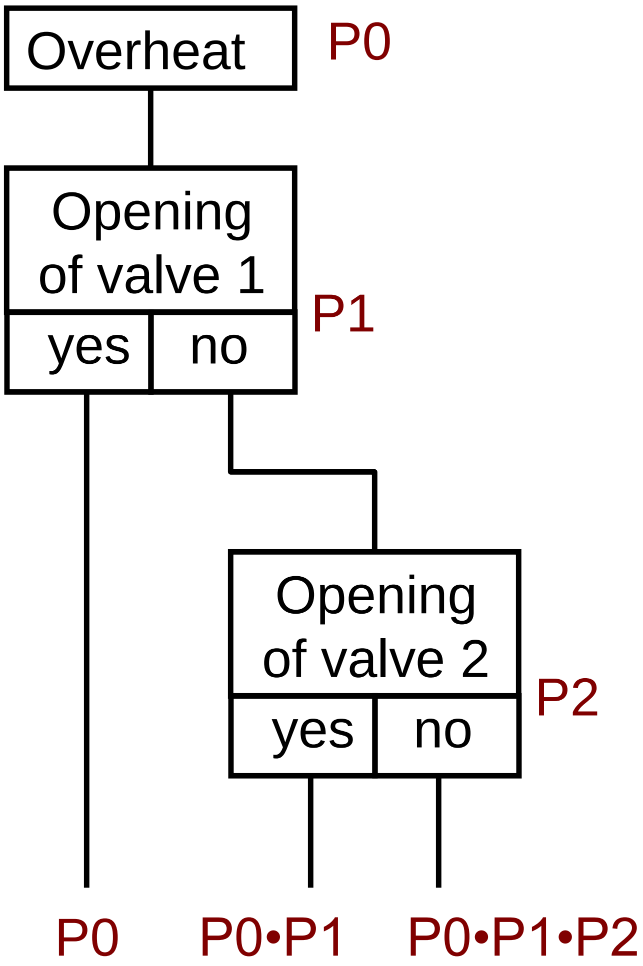
Event tree example: Recovery blocks (RB)



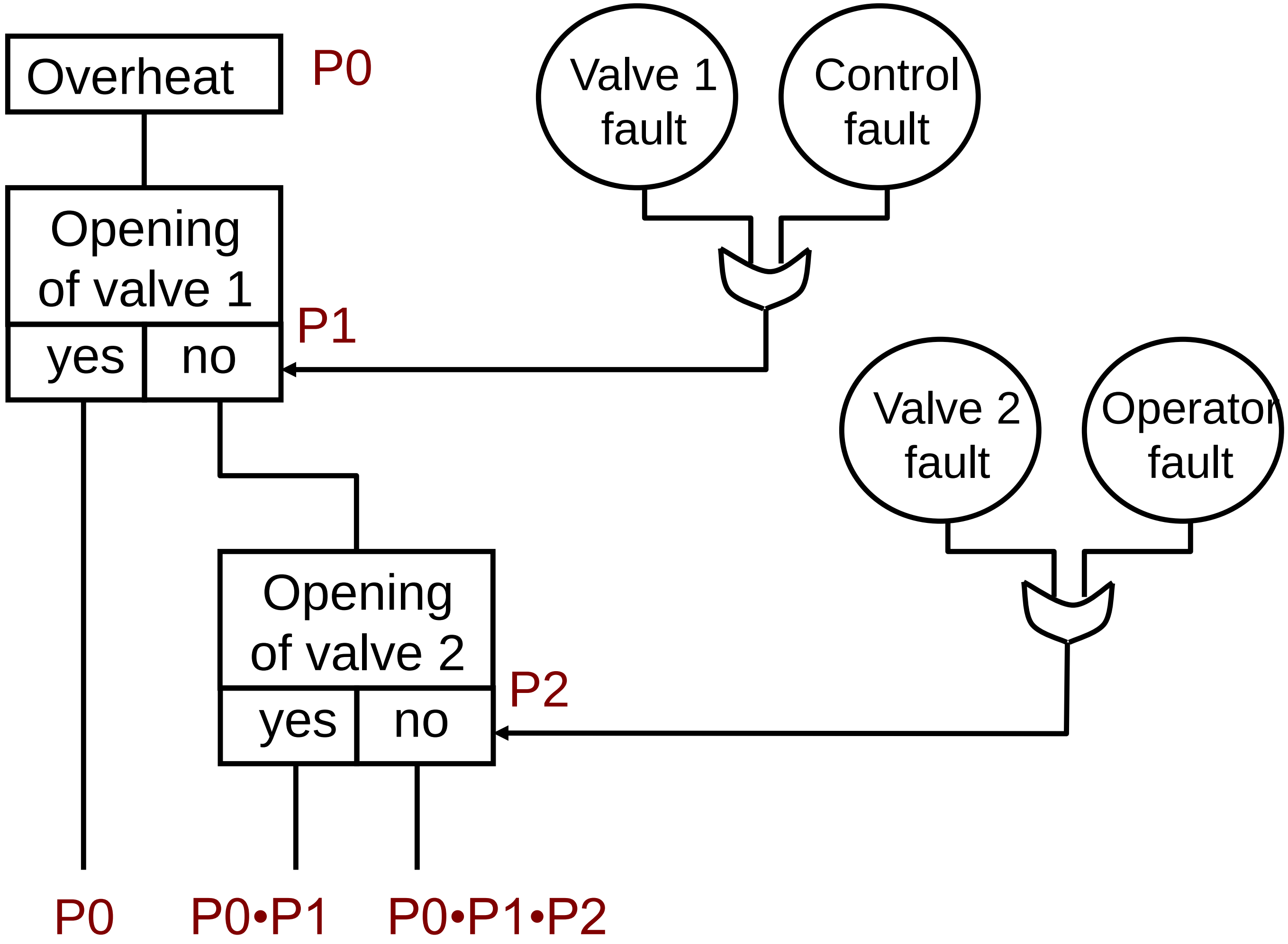
4. Cause-consequence analysis

- 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

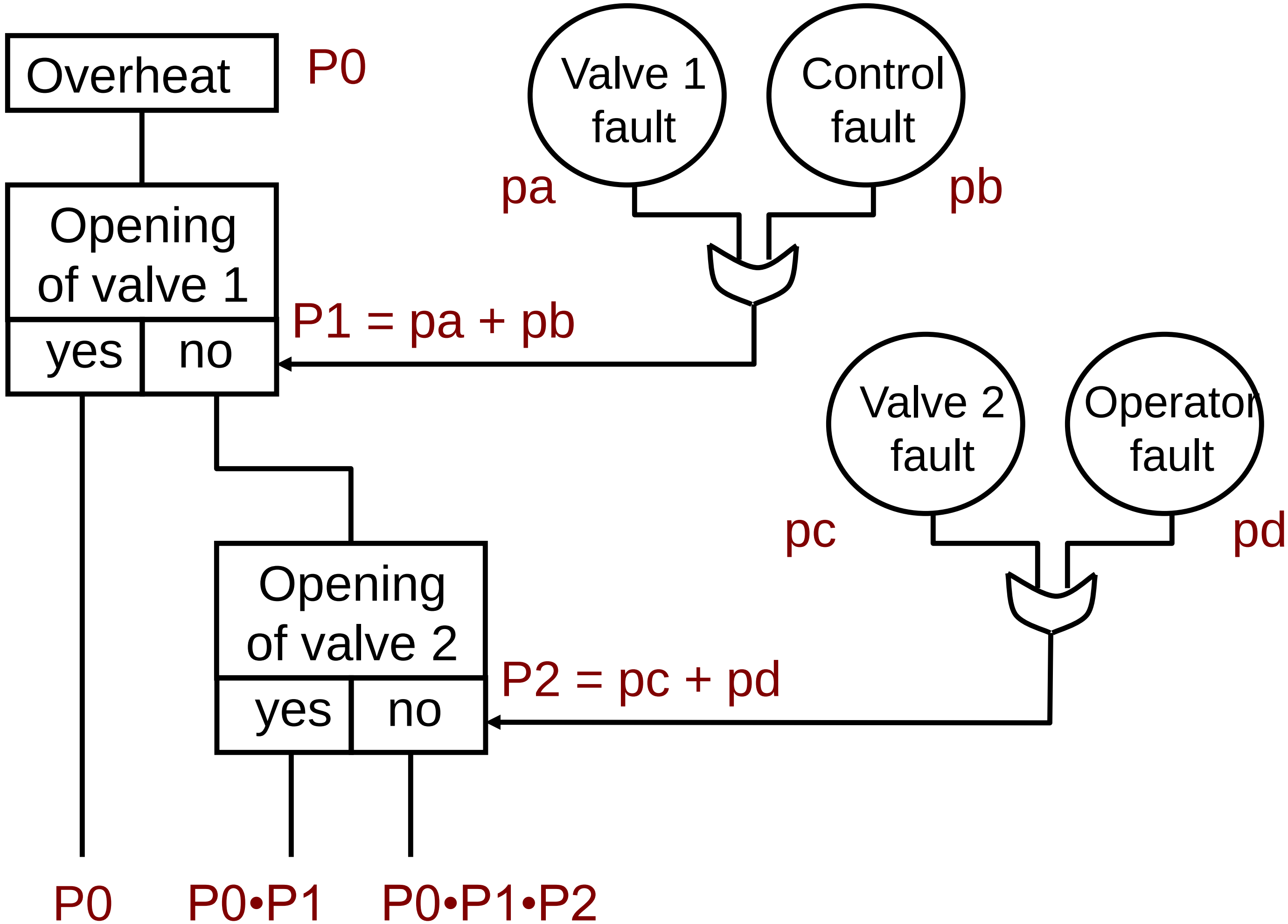
Cause-consequence analysis example



Cause-consequence analysis example



Cause-consequence analysis example

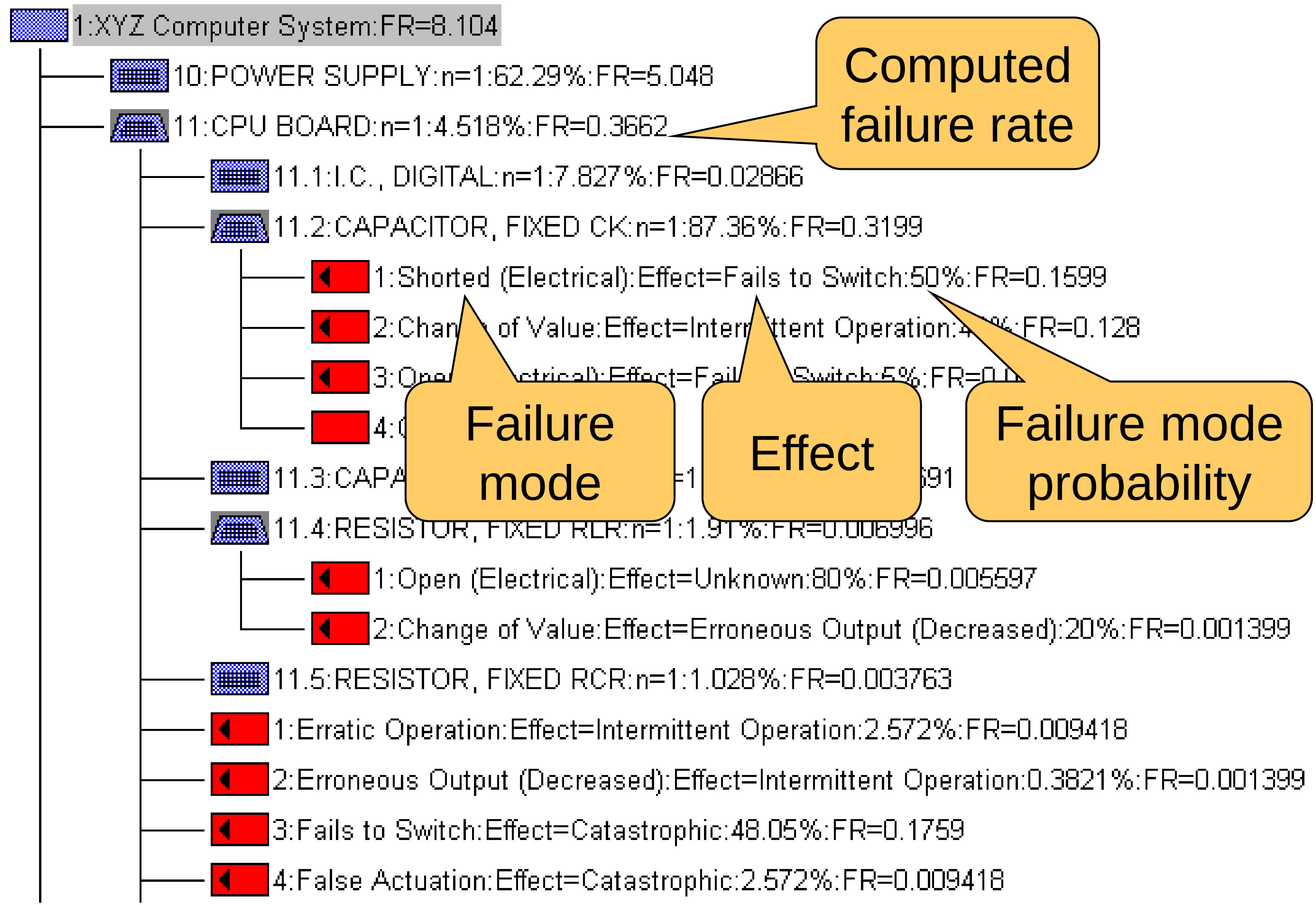


5. Failure modes and effects analysis (FMEA)

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

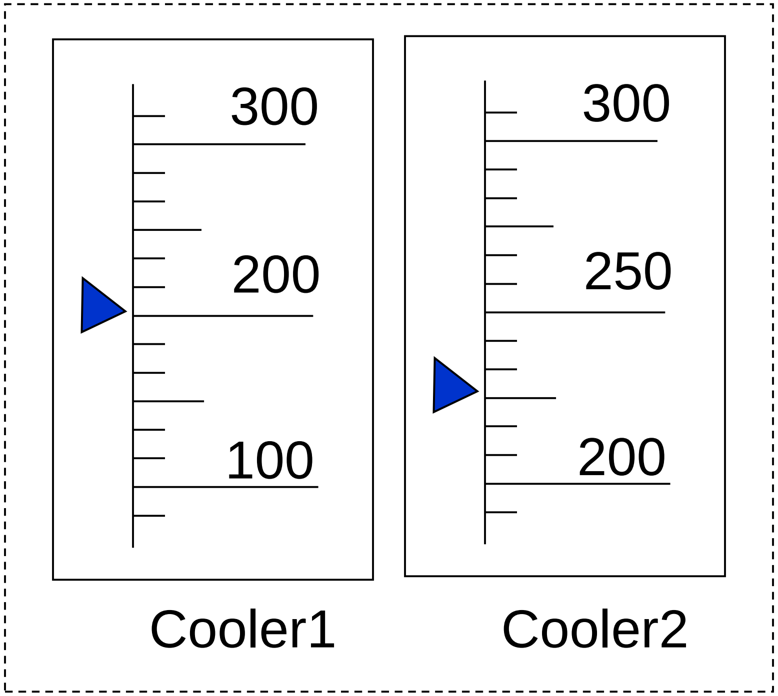
Component	**Failure mode**	**Probability**	**Effect**
D1 diode	open circuit short circuit	65% 35%	- over-heating - damaged product
...	...	...	...

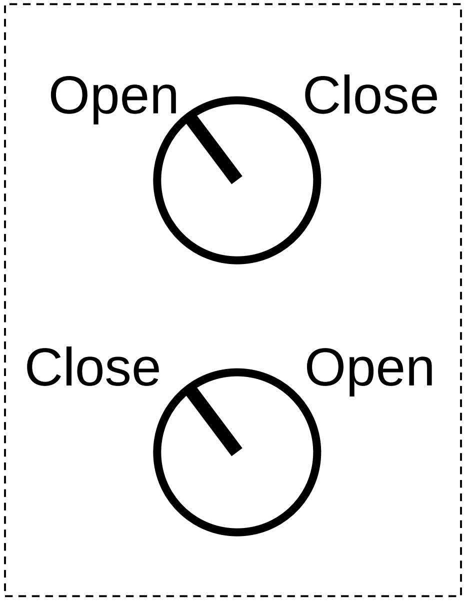
Example: Analysis of a computer system



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

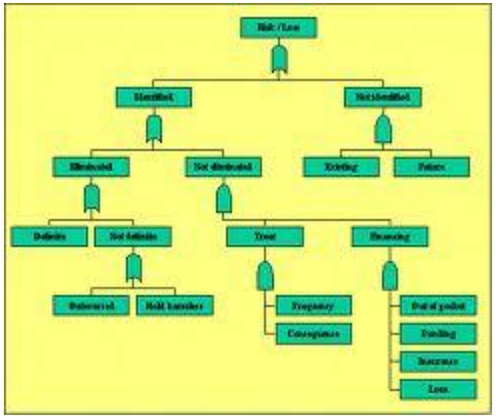
Example: Risk matrix (railway control systems)

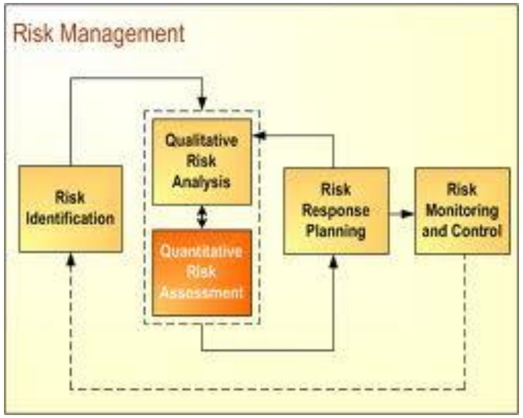
	Frequency of Occurrence of a Hazardous Event	RISK LEVELS			
Daily to monthly	**FREQUENT (FRE)**	Undesirable (UND)	Intolerable (INT)	Intolerable (INT)	Intolerable (INT)
Monthly to yearly	**PROBABLE (PRO)**	Tolerable (TOL)	Undesirable (UND)	Intolerable (INT)	Intolerable (INT)
Between once a year and once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

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

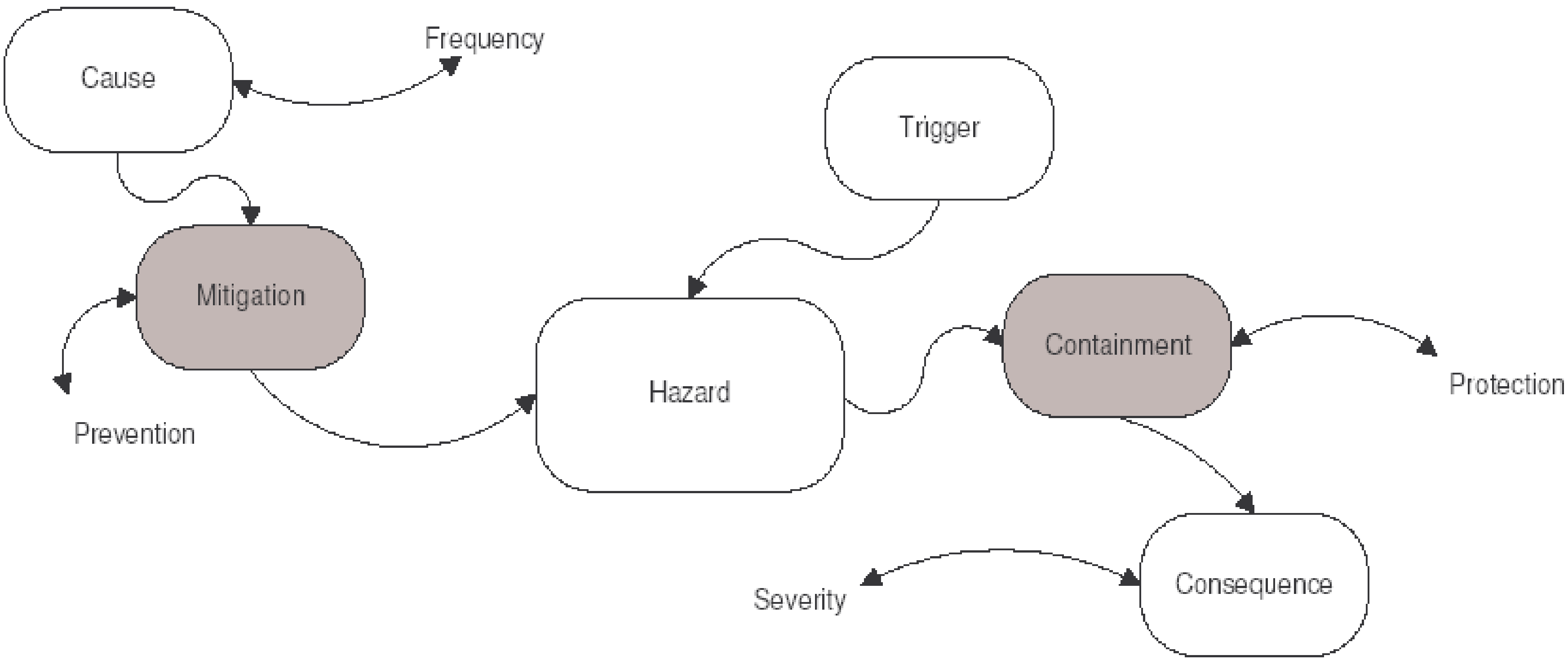
Risk reduction techniques





Basic idea for risk reduction

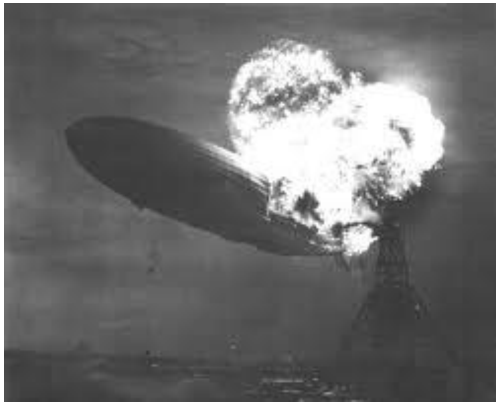
- Mitigation (or prevention) of causes
- Containment (or protection) of consequences



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
a. Substitution 	▪ Using safer material, component, technology, … E.g., substitution of flammable or toxic materials	▪ More safe programming language (e.g., SPARK Ada instead of C) ▪ Using well-tried modules (proven in use)

1. Hazard elimination

Generic method	Hardware solution	Software solution
b. Simplification 	▪ Reducing the number of components ▪ Reducing the number of operating modes Flexibility ↔ simplification Fault tolerance ↔ simplification	Simple program structure (testable, analyzable): ▪ Deterministic, static control ▪ Structured programming ▪ Simple interfaces ▪ Robust data structures

1. Hazard elimination

Generic method	Hardware solution	Software solution
c. Decoupling 	Elimination of dependences and unnecessary interactions (error propagation paths) E.g., firebreaks, overpasses and underpasses	“Loosely coupled” software: ▪ Modularization (safety kernel) ▪ Information hiding (well-defined interfaces) ▪ Separation of safety-critical and non-safety-critical functions

1. Hazard elimination

Generic method	Hardware solution	Software solution
d. Eliminating human errors 	Masterability, understandability, maintainability, checkability ▪ Ergonomic interfaces ▪ No interchangeable connectors ▪ Color codes ▪ …	Limiting fault prone features in language subsets ▪ Pointers, ▪ Implicit conversion, ▪ Overloading, … Simple human-machine interfaces: ▪ Clear operation modes ▪ Tolerable timing

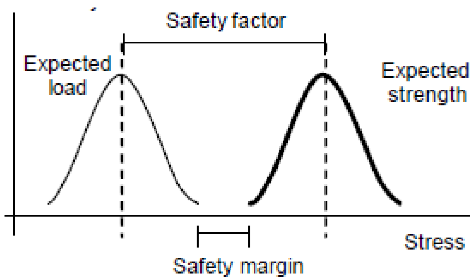
2. Hazard reduction

Generic method	Hardware solution	Software solution
a. Design for controllability (active hazard reduction) 	▪ Allowing actions to provide protection in case of hazards ▪ Detection, diagnosis and controlled response ▪ E.g., mechanical control systems (backup), multiple control modes, …	▪ Incremental control: Feedback and corrections ▪ Monitoring hazards and conditions: - Sanity check - Monitor-actuator - Watchdog - Safety executive architecture patterns

2. Hazard reduction

Generic method	Hardware solution	Software solution
b. Barriers (passive hazard reduction) 	▪ Lockout: Making access to dangerous state difficult (wall, fence) ▪ Lockin: Make leaving a safe state difficult (safe area) ▪ Interlock: Enforce a safe sequence of actions	▪ Lockout: Access control, authorization, acknowledgements ▪ Lockin: Checking inputs, requests, accesses ▪ Interlock: Checking call sequences, synchronization (baton)

2. Hazard reduction

Generic method	Hardware solution	Software solution
c. Failure minimization  	▪ Robust components ▪ Safety factors, safety margins (e.g., higher load does not cause failure) Safety factor: Ratio expected strength and expected (nominal) stress Safety margin: Difference of minimum probable strength and maximum probable stress	▪ Robustness ▪ Redundancy (diverse instances) ▪ Fault tolerance: Forward recovery is preferred (guarantees for execution)

3. Hazard control

Generic method	Hardware solution	Software solution
a. Reducing exposure	▪ Staying in higher risk state as short as possible ▪ Timely return to safe state	▪ Safe initial state ▪ Keeping synchronization with the environment to return to safe state
b. Isolation and containment	▪ Isolation in time and space	▪ Partitioning of safety functions
c. Protection systems	▪ Moving the system to safe state	▪ Control to safe state ▪ Challenge protocol for protection systems

4. Damage minimization

Generic method	Hardware solution	Software solution
a. Planning alarming and escape routes	▪ Alarm devices with periodic testing ▪ Fire escape, lifeboat, abandonment of products	▪ Software controlled alarm ▪ Complex devices with software support (e.g., airbag control)
2. Determining „point of no return”	▪ Turn to damage minimization instead of hazard control	

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