System Modelling

Fault Modelling

(based on slides from MAJZIK István and MICSKEI Zoltán)

Budapest University of Technology and Economics
Department of Measurement and Information Systems

Contents

- Concept of service dependability
- Factors affecting service dependability
- Tools of service dependability
- Service dependability analysis

Motivation: Failure Free Operation

- Service Level Agreements (SLA):
  - Characteristics required by the client
  - TelCo service systems („carrier grade”):
    „Five nines”: 99,999% (5 mins/year outage)
- Safety critical systems:
  - Standard specifications of the frequency of errors
  - Safety Integrity Levels → THRs (Tolerable Hazard Rates)

	…n failure / hour
	$10^{-6} \leq THR < 10^{-5}$
	$10^{-7} \leq THR < 10^{-6}$
	$10^{-8} \leq THR < 10^{-7}$
	$10^{-9} \leq THR < 10^{-8}$

If the life-span is 15 years, then 1 out of 750 devices will fail in that time

Failure free operation ~ 11.000 years??

Inevitable: Faults

Design process → **Operational product**

- Design faults
- Implementation faults

- Hardware faults
- Configuration faults
- Operator faults

Inevitable: Faults

Design process → **Operational product**

- Design faults
- Implementation faults

- Hardware faults
- Configuration faults
- Operator faults

Characteristics of design process:

- Better quality assurance, better methodologies
- But increasing complexity, more difficult verification

Usual estimated values for 1000 line of code:

- Good manual development and testing: <10 faults remain
- Automated development: ~1-2 faults remain
- Using formal methods: <1 fault remains

Inevitable: Faults

Design process → **Operational product**

- Design faults
- Implementation faults

- Hardware faults
- Configuration faults
- Operator faults

Technological limits:

- Better parameters, better materials
- But increasing complexity (sensitivity)

Usual estimated values:

- CPU: 10^{-5}…10^{-6} faults/hour
- RAM: 10^{-4}…10^{-5} faults/hour
- LCD: ~ 2…3 years life-span

Inevitable: Faults

Design process → **Operational product**

- Design faults
- Implementation faults

- Hardware faults
- Configuration faults
- Operator faults

Verification and validation in design time

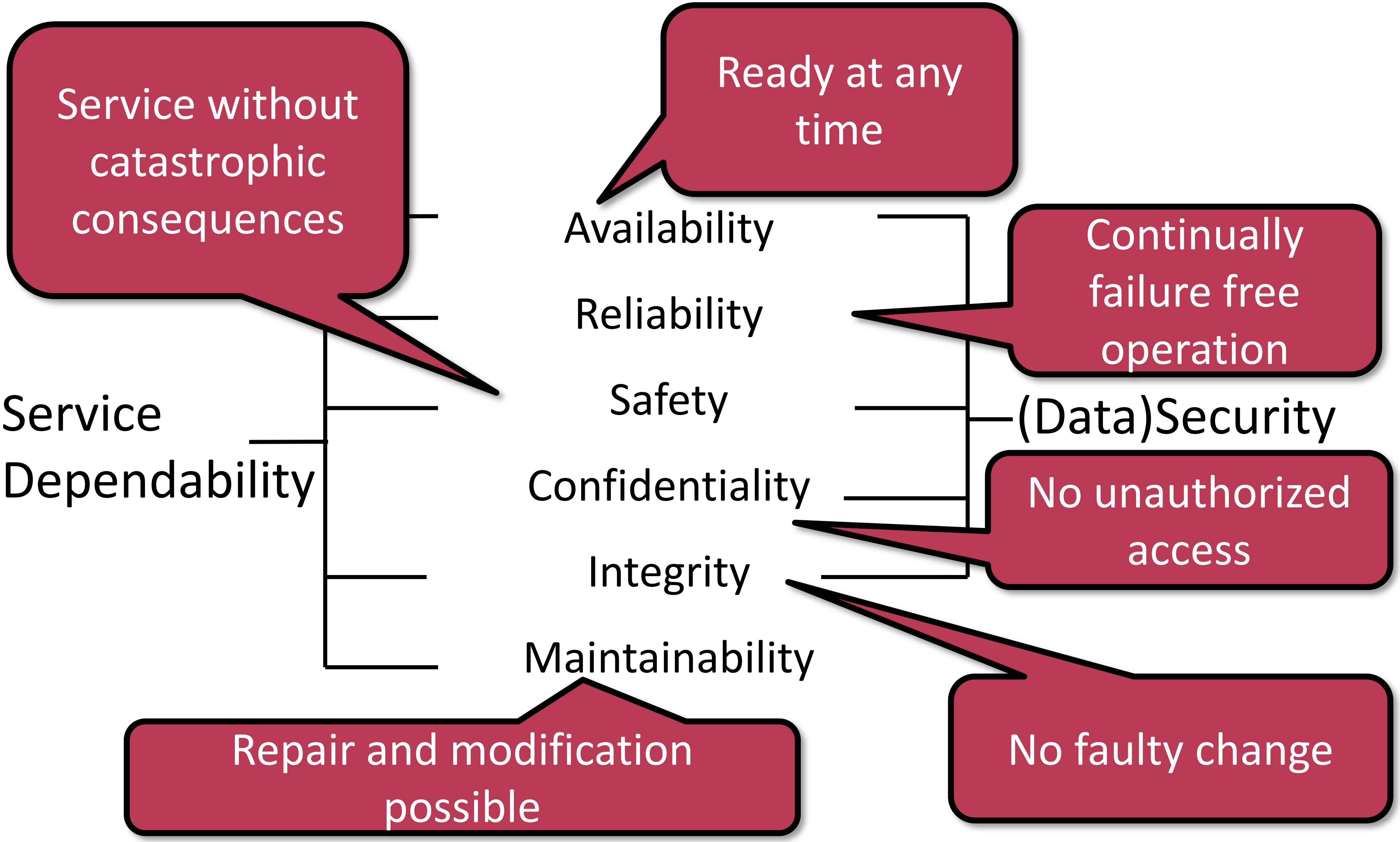
Fault tolerance during operation

Service Dependability

Dependability: is the ability to deliver service that can justifiably be trusted

- *justifiably*: based on analysis, measurements
- *trust*: service satisfies the demands

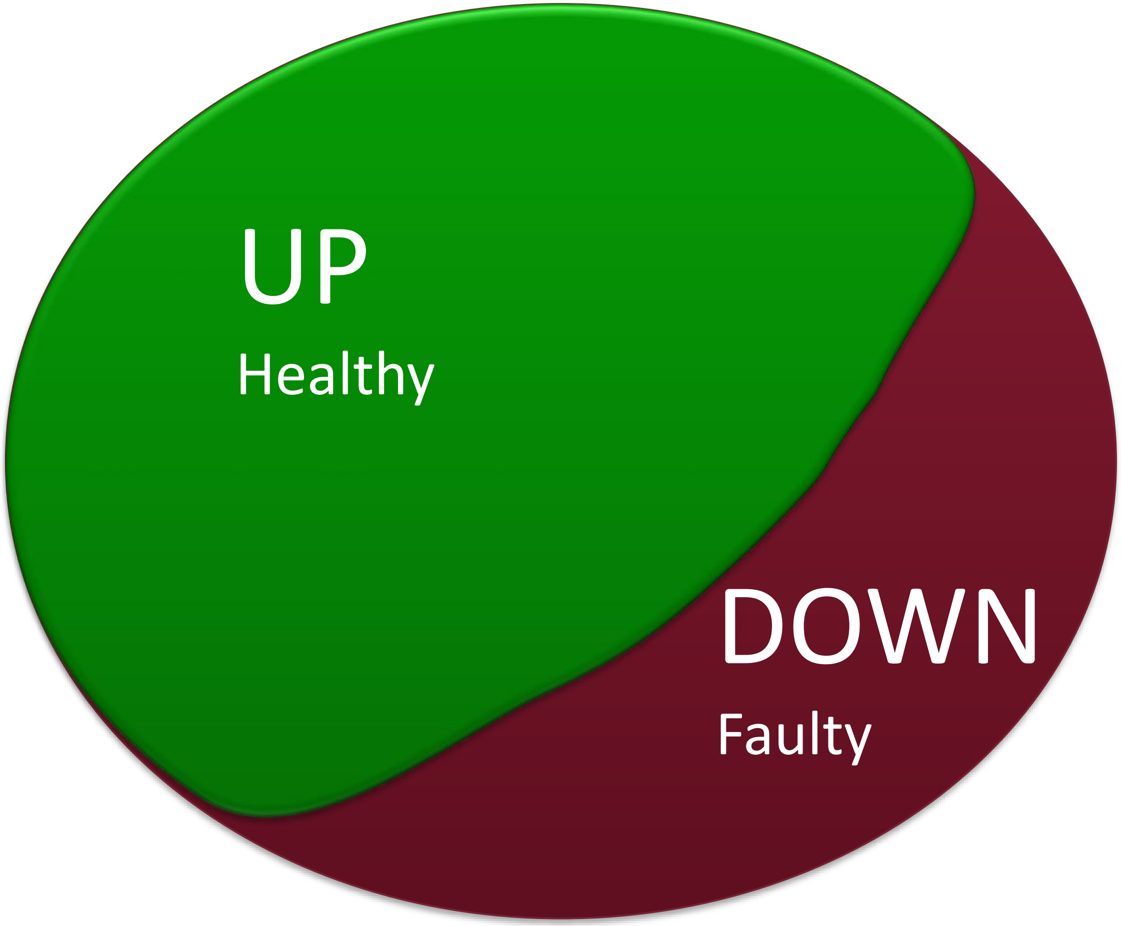
Attributes of Dependability



Laprie et. al.: Basic Concepts and Taxonomy of Dependable and Secure Computing

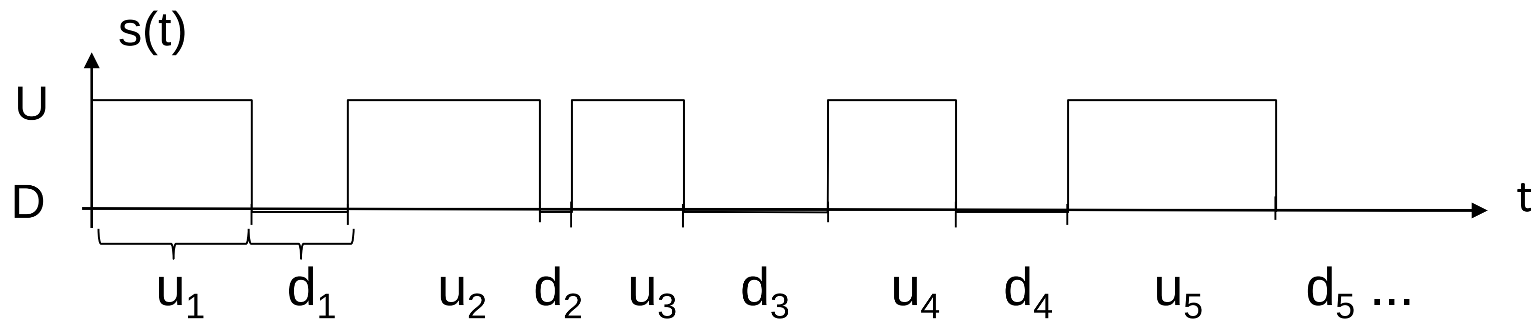
State Partitioning

- S: the state space of the system



Reliability Attributes

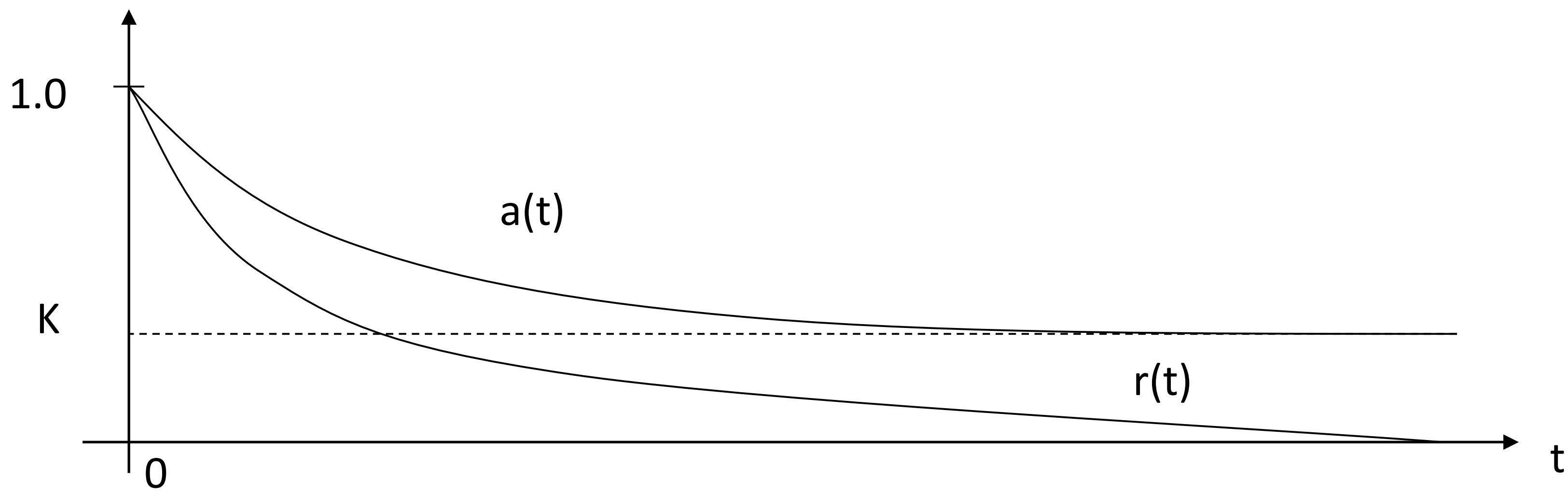
- State partitioning→ s(t) system state
  - Down (D) – Up (U) state partition



- Mean values:
  - Mean time to first failure: $MTFF = E\{u_1\}$ (sometimes MTTF)
  - Mean up time: $MUT = E\{u_i\}$
  - Mean down time: $MDT = E\{d_i\}$
  - Mean time between failures: $MTBF = MUT + MDT$

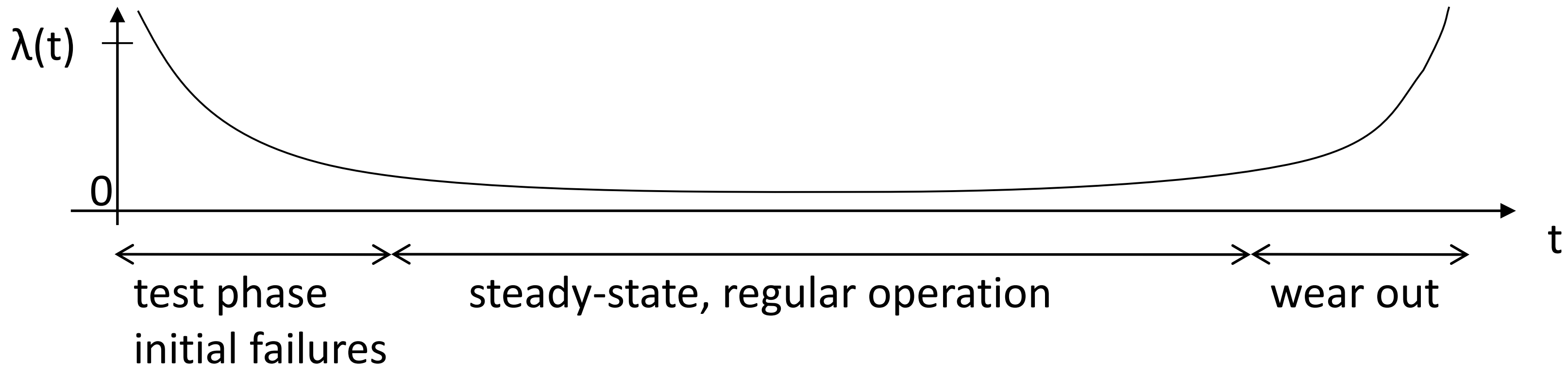
Probability Time Functions

- reliability:
  $r(t) = P\{\forall\ t' < t: s(t') \in U\}$ (can not go down)
- availability:
  $a(t) = P\{ s(t) \in U \}$ (may go down)
  - steady-state availability: $K = \lim_{t \to \infty} a(t) = \dfrac{MUT}{MUT+MDT}$



Reliability

- reliability: $r(t) = P\{ s(t') \in U, \forall t' < t \}$
- (first) failure rate: $-r'(t)$
  - The probability density function of the „time to first failure” probabilistic variable!
  - Mean value: MTTF
- failure rate: $\lambda(t) = -r'(t) / r(t)$ (probability of failure for one device during a period of time)
  - „bathtub curve”:



Reliability

- Approximation: steady-state, λ(t) = λ (const.)
  - „memoryless" property
  - May be true for a properly tested IT system: outdated before wear out
- Consequence: $r(t) = e^{-\lambda t}$
- -r'(t): time to failure is exponential distribution
  - with λ parameter
  - 1/λ mean value
  - Therefore: MTTF = 1/λ !

Requirements for Availability

Availability rate	Maximum outage per year
2 nines (99%)	3,5 days
3 nines (99.9%)	9 hours
4 nines (99.99%)	1 hour
5 nines (99.999%)	5 minutes
6 nines (99.9999%)	32 seconds
7 nines (99.99999%)	3 seconds

Distributed systems (without fault tolerance, guiding figures):

- **1 computers : 95%**
- **2 computers : 90%**
- **5 computers : 77%**
- **10 computers : 60%**

Note for the Homework

- P(process$_{serial}$ ☺) = P(Task$_1$ ☺)*P(T$_2$ ☺)*…*P(T$_n$ ☺)
- P(Task$_n$ ☺) = $r_n(t_n) = e^{-\lambda_n * tn}$
- $\lambda_{total} * t_{total} = \sum_{i=0}^{n} \lambda_n * tn$
- $\lambda_i = 1/MTTF_i$
- Failure rate is a kind of costs
  - proportional to time
  - additive
- In homework: rescale for representation
  - $\lambda_n * tn$ should be a usable value (and t_n is small)
  - Result should be scaled back at the end!

Contents

- Concept of service dependability
- Factors affecting service dependability
- Tools of service dependability
- Service dependability analysis

Affecting Factors

- **Failure**:
  Service not conforming to specification
  - value / time, catastrophic / „beneficial"
- **Error**:
  System state leading to failure
  - latent $\rightarrow$ detected
- **Fault**:
  Presumed cause of error
  - effect: dormant $\rightarrow$ active
  - type: accidental or intentional, temporary or permanent
  - origin: physical/human, internal/external, development/operational

Software Faults

- Software fault: Permanent, developmental
- Activation is the function of operational profile
  - Input domain, trajectory
- Reliability is proportional to:
  Number of faults left after testing
- Number of fault left is proportional to:
  Faults detected during a period of time at the end of testing
  - Statistic testing: Measuring reliability
- Statistic techniques can estimate how long the testing process should be continued to reach a given reliability

Fault Chain

- Fault → Error → Failure
  - e.g. software:
    - fault: progr. fault: increase instead of decrease
    - error: control flow reaches it, variable value erroneous
    - failure: result of wrong calculation
  - e.g. hardware:
    - fault: cosmic radiation changes a bit
    - error: reading faulty memory cell
    - failure: robot arm hits the wall
- Function of system hierarchy level
  - lower level failure is fault on higher level
    - stuck output is failure in a chip
    - fault on system level (chip is the replaceable unit)

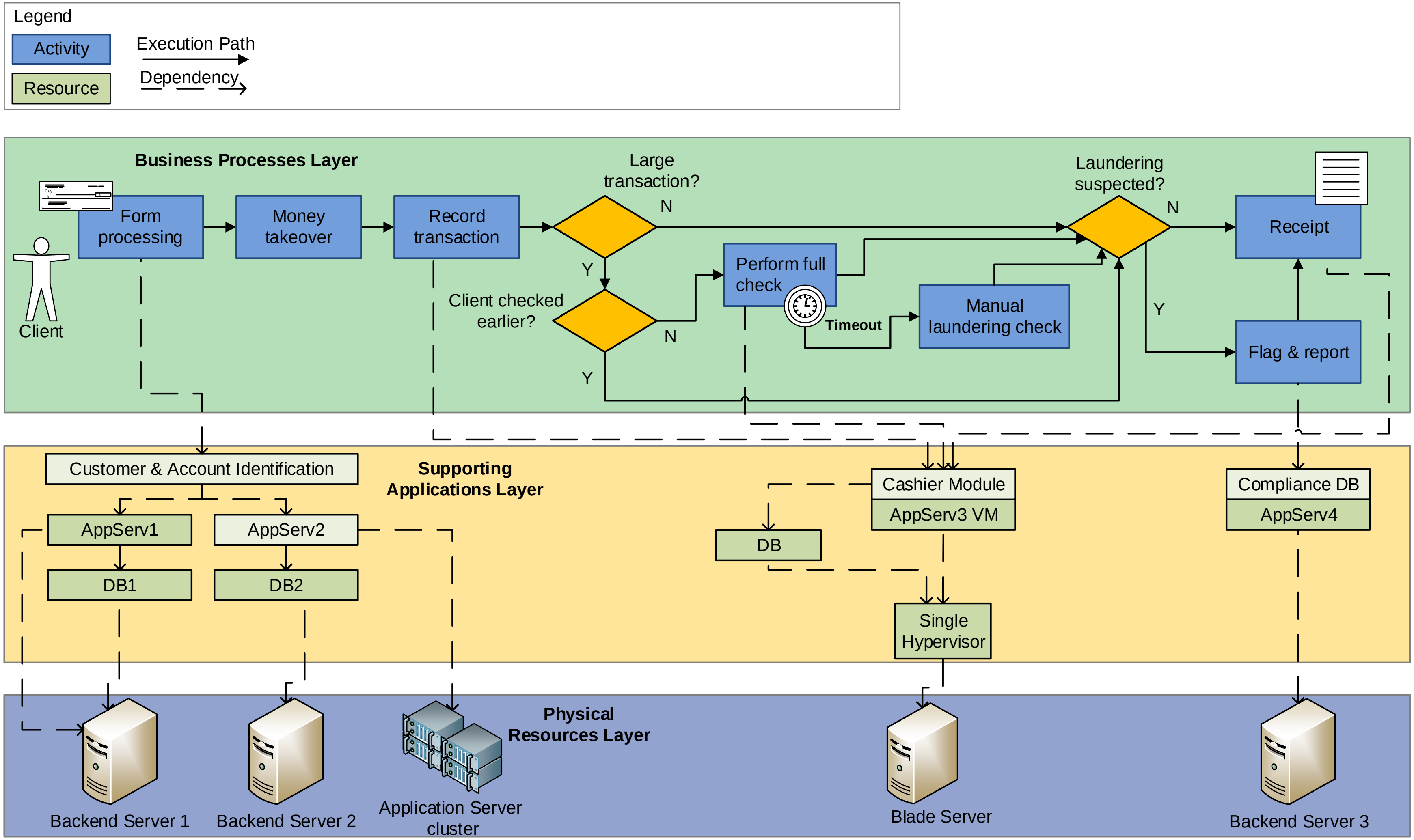
Fault Chain

- Affecting the fault chain
  - decrease failure rate
    - better quality components
    - stricter development process (verification, testing)

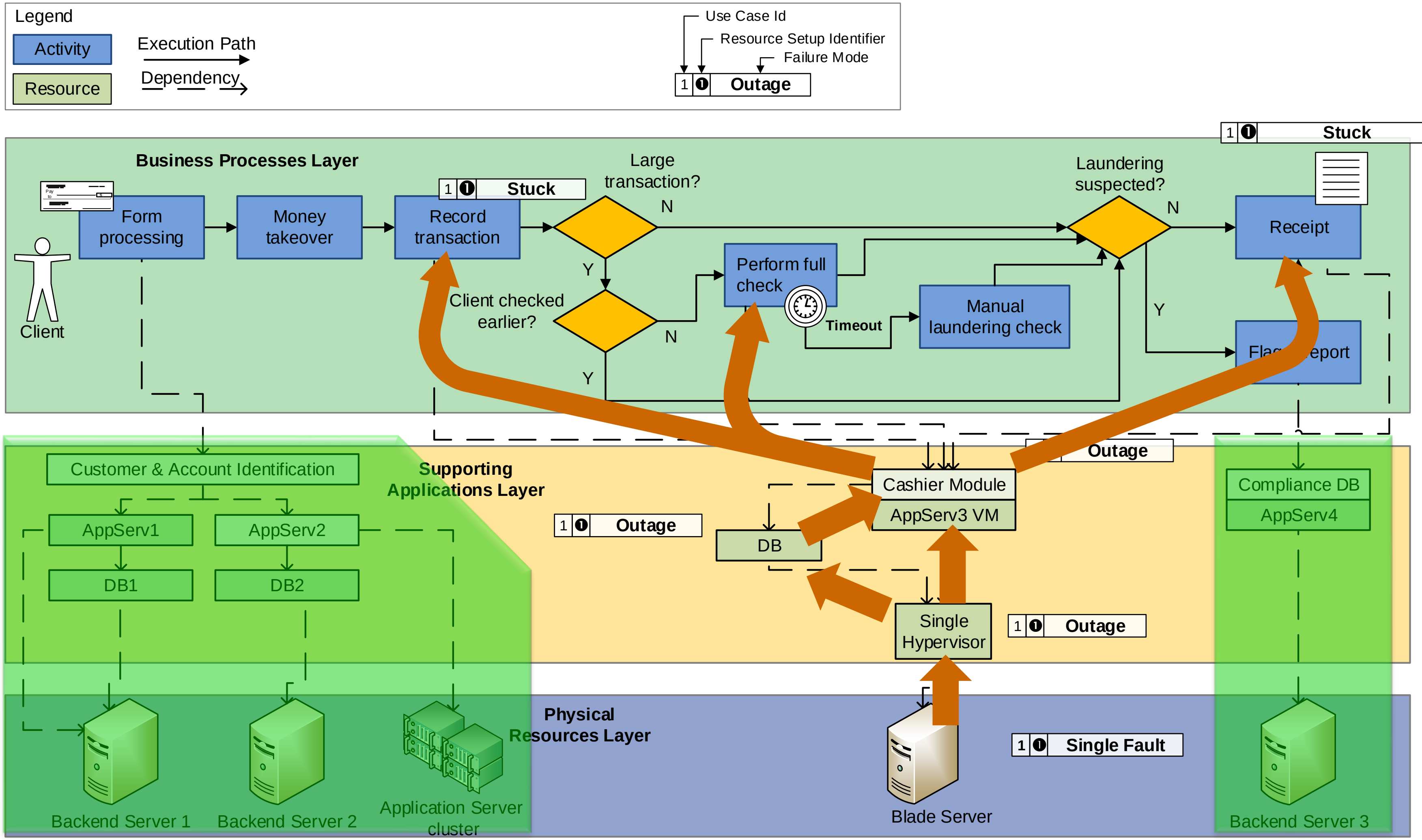
    Failure free operation can not be guaranteed (smaller chip size, more complex programs)
  - prevent emergence of failure
    - system structure design: redundancy
- Fault types:
  - faults considered in advance:
    optimal handling during design process
  - unforeseeable faults:
    requires appropriate system structure

Example: The Process

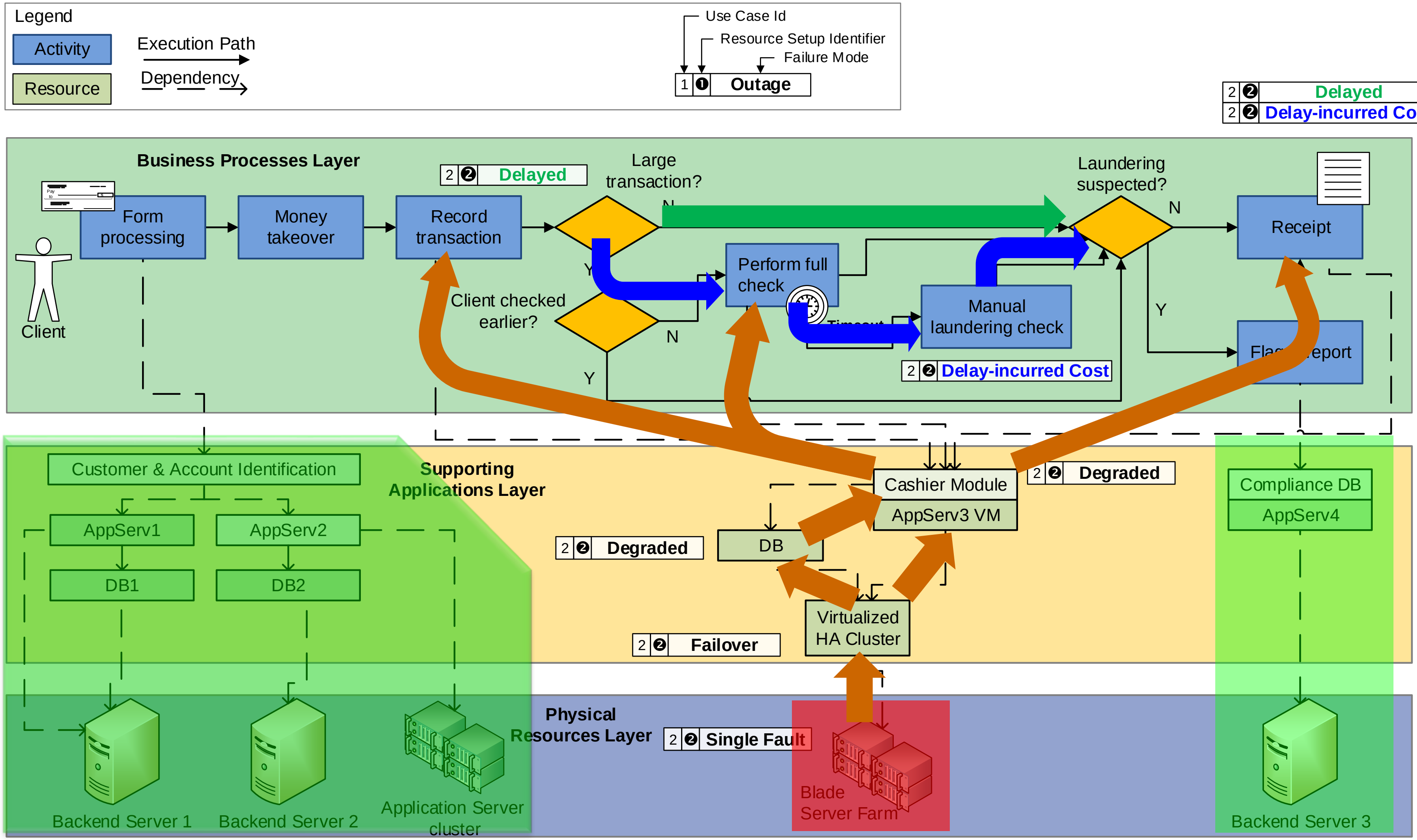
Gábor Urbanics – László Gönczy –Balázs Urbán – János Hartwig – Imre Kocsis:
Combined error propagation analysis and runtime event detection in process driven systems. In *Software Engineering for Resilient Systems*. 2014, Springer, 169–183. p.



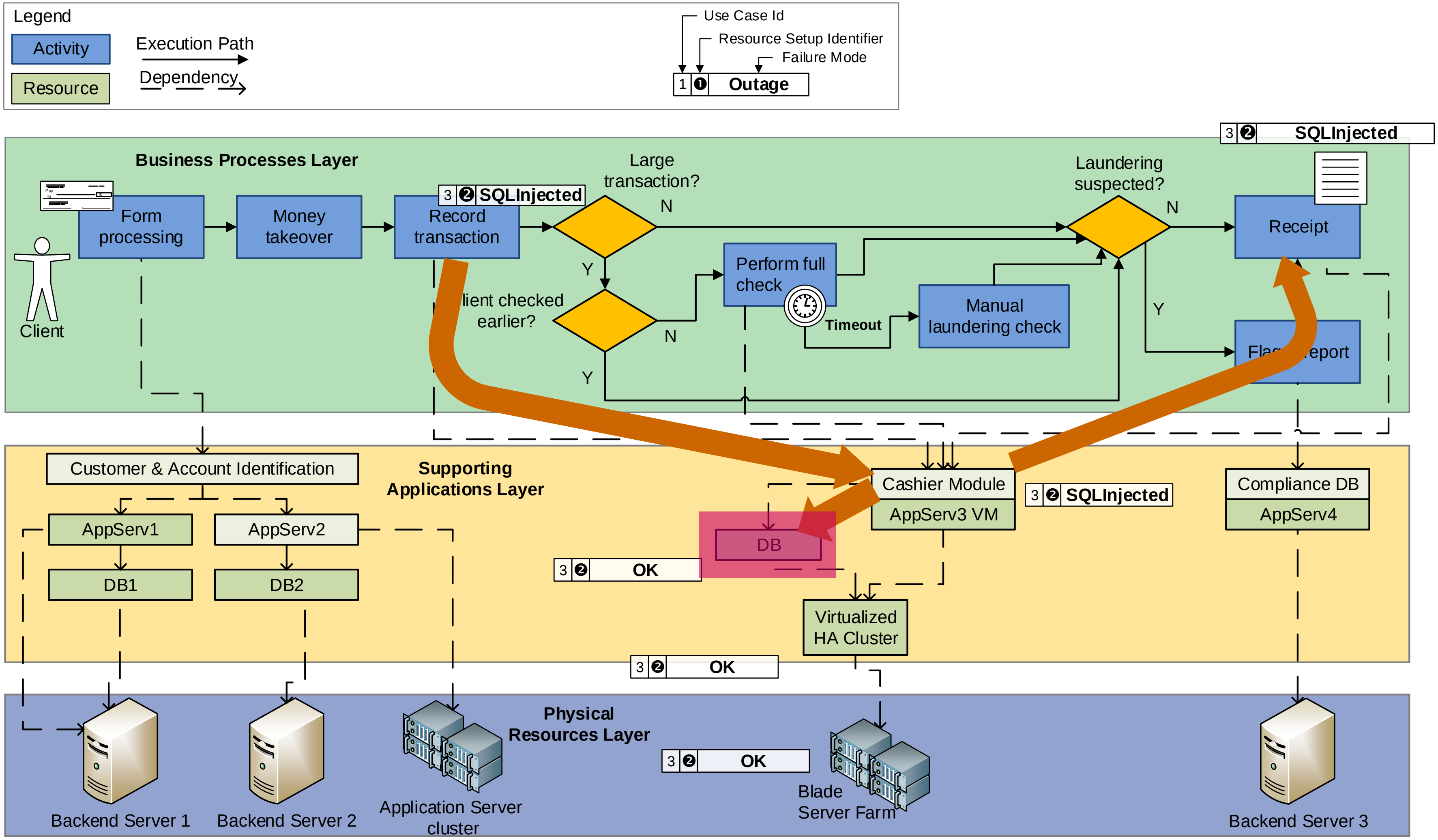
Single (Hardware) Fault



Effects of the Single Fault



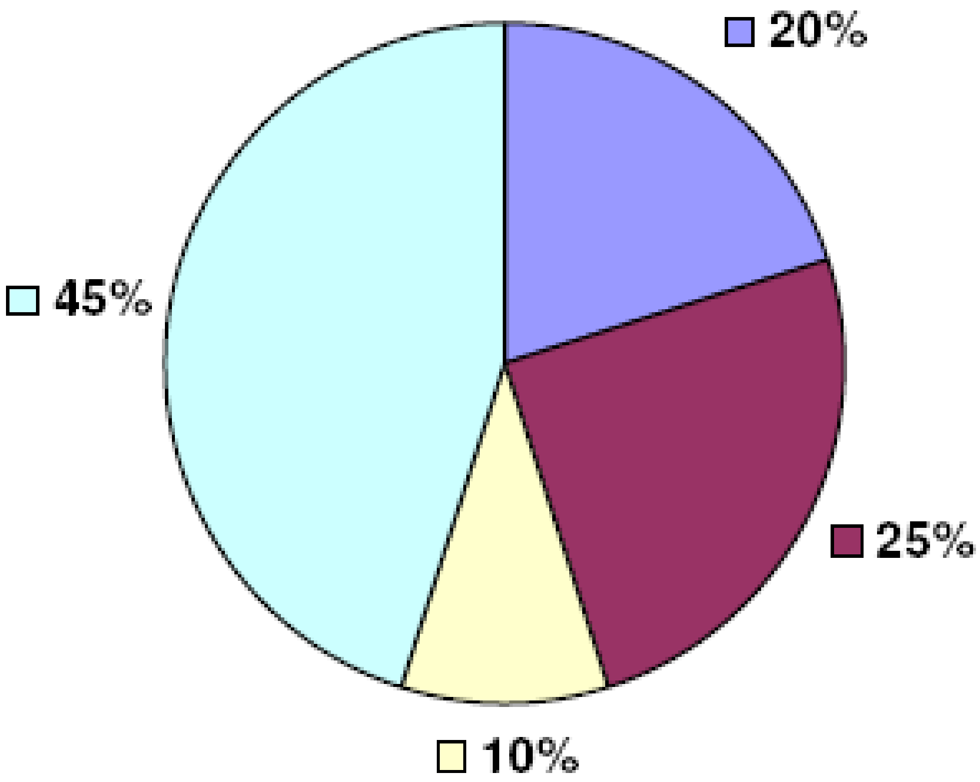
Propagation of the Fault



Categorizing Faults

- Hardware faults
  - base system (motherboard, processor, memory)
  - power (power supply, UPS)
  - storage subsystem
  - network
- Software faults
  - operating system faults
  - application faults
  - driver faults
- …
- Human-made faults
  - administrator faults
  - non-malicious fault of users
  - malicious fault of users
  - attack of an outsider
- Natural faults
  - interference of operation environment, eg. failing air conditioning, bomb alarm, pipe break
  - natural disasters

Causes of IT System Failures



- Operating system
- Software fault
- Hardware fault
- Other (human,natural, configuration) (PEBCAK, PICNIC ;)

Contents

- Concept of service dependability
- Factors affecting service dependability
- Tools of service dependability
- Service dependability analysis

Means of dependability

- **Fault prevention**: prevent the occurrence of faults
  - physical faults: good quality components, shading, ...
  - design faults: verification
- **Fault removal**:
  - prototype phase: testing, diagnostics, repair
  - in operation: monitoring, repair
- **Fault tolerance**: provide service even in the presence of faults
  - in operation: fault handling, redundancy
- **Fault forecasting**: estimate number and consequence of faults
  - measurement and „prediction", preventive maintenance

Fault-tolerant Systems

- However good is the verification during design, dependability can not be guaranteed:
  - temporary hardware faults (see disturbance sensitivity)
  - non-tested software faults
  - non-considered complex interactions

  → We must prepare for in-operation faults!
- Fault tolerance: provide service even in the presence of faults
  - autonomic fault handling in operation
  - intervention to fault → failure chain
  - system-based solutions (+ dependable components)
- Main condition: Redundancy (spares)
  - spare resources to replace faulty components

Appearance of Redundancy

1. Hardware redundancy
   - excess hardware resources
     - already in the system (distributed system)
     - designed for fault tolerance (spare)
2. Software redundancy
   - excess software modules
3. Information redundancy
   - excess information for fault removal
     - error correction coding (ECC)
4. Time redundancy
   - repeated execution, surplus time of fault handling

Simultaneous appearance!

Type of Redundancy

- Cold reserve (passive redundancy):
  - passive in normal operation, activated when fault occurs
  - slow failover (starting, state updating,...)
  - e.g. spare computer
- Warm reserve:
  - secondary functions in normal operation
  - faster failover (starting is not needed)
  - e.g. logging machine takes up critical functions
- Hot reserve (active redundancy):
  - active in normal operation, executes the same tasks
  - failover immediately
  - e.g. duplicated, multiplicated

1. Hardware Redundancy

- Multiplication
  - Duplication
    - fault detection: e.g. master-checker setup
    - fault tolerance only with diagnostics support and failover
  - TMR: Triple-modular redundancy
    - fault masking with voting
    - voter is critical component (but simple)
  - NMR: N-modular redundancy
    - fault masking with majority vote
    - MTFF lower, but higher chance of surviving mission time
    - airplane on-board devices: 4MR, 5MR
- Typical: high-availability clusters

Level of Multiplication

- Computer (server) level: Loosely coupled
  - high-availability clusters
    e.g. Sun Cluster, HA Linux, Windows Failover Cluster
  - software support: state synchronization, transactions
- Card level:
  - runtime reconfiguration, “hot-swap”
    e.g. compactPCI, HDD
  - software support: configuration management
- Component level: tightly coupled
  - component level multiplication
    pl. TMR, self-checking circuit

2. Software Redundancy

Usage:

1. In case of software design faults:
   - repeated execution doesn't help...
   - redundant modules: different design is required
     variants: same specification, but
     - different algorithm, data structure
     - different development environment, programming language
     - isolated development
2. In case of temporary (hardware) faults:
   - fault does not appear after repeated execution
   - fault prevention important

3. Information Redundancy

- Error correction coding
  - memory, disk, data transfer
  - e.g. Hamming-code, Reed-Solomon code
- Limited fault removal ability
  - long-term data stability can be bad (faults “pile up”)
  - disk: “memory scrubbing” continuous read and corrected write
- Redundant (multi instance) databases:
  - ensure access consistency
  - one instance serialization

4. Time Redundancy

- Clear case: retry execution
  - low-level hardware: processor instruction
  - effective against temporal faults
- Time redundancy is “companion” of others types

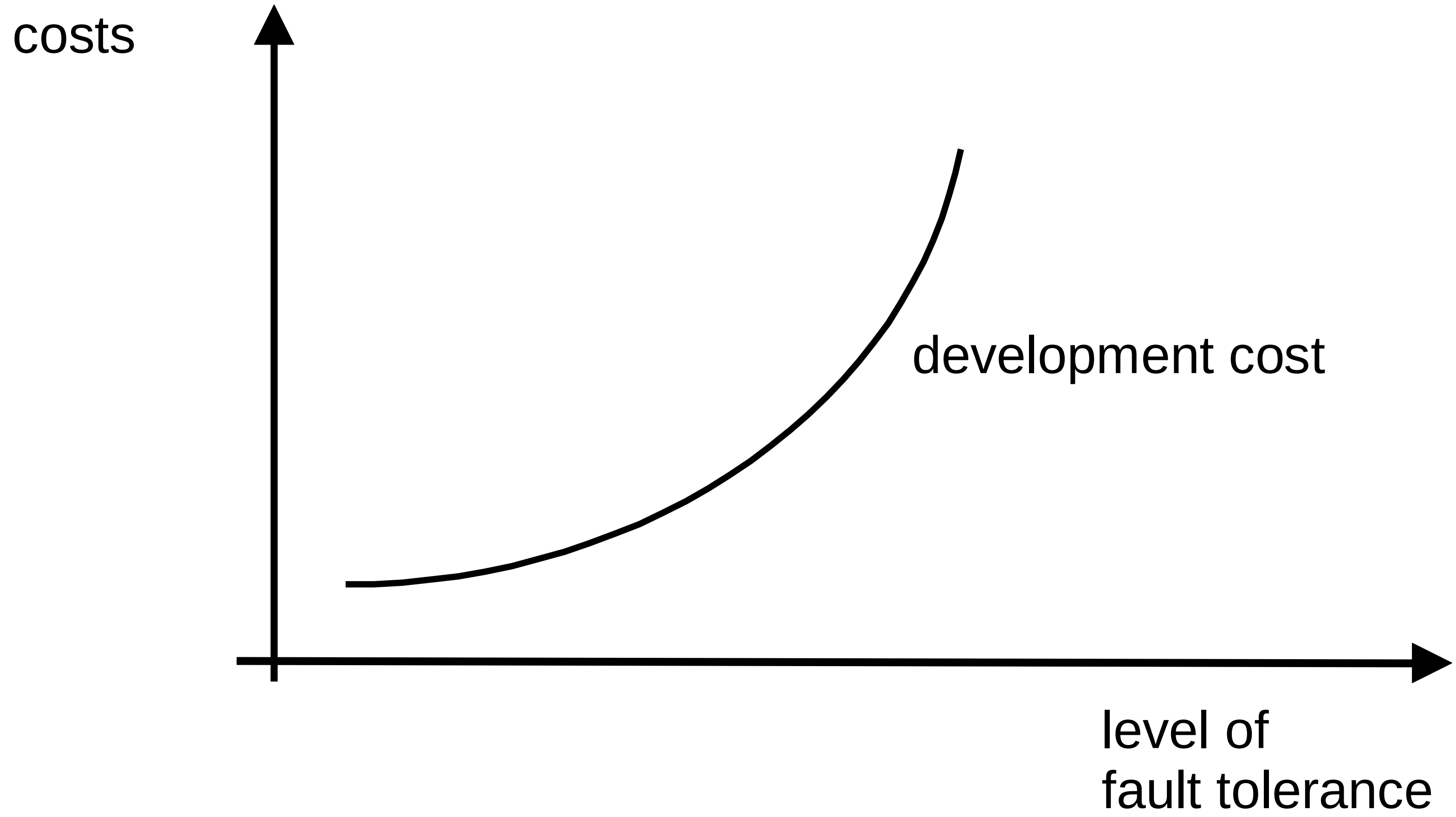
  Real-time systems: design consideration

  is the time of fault handling guaranteed?
  - permanent hardware faults: masking, hot reserve
  - temporal hardware faults: roll-forward recovery
  - software design faults: N-version programming

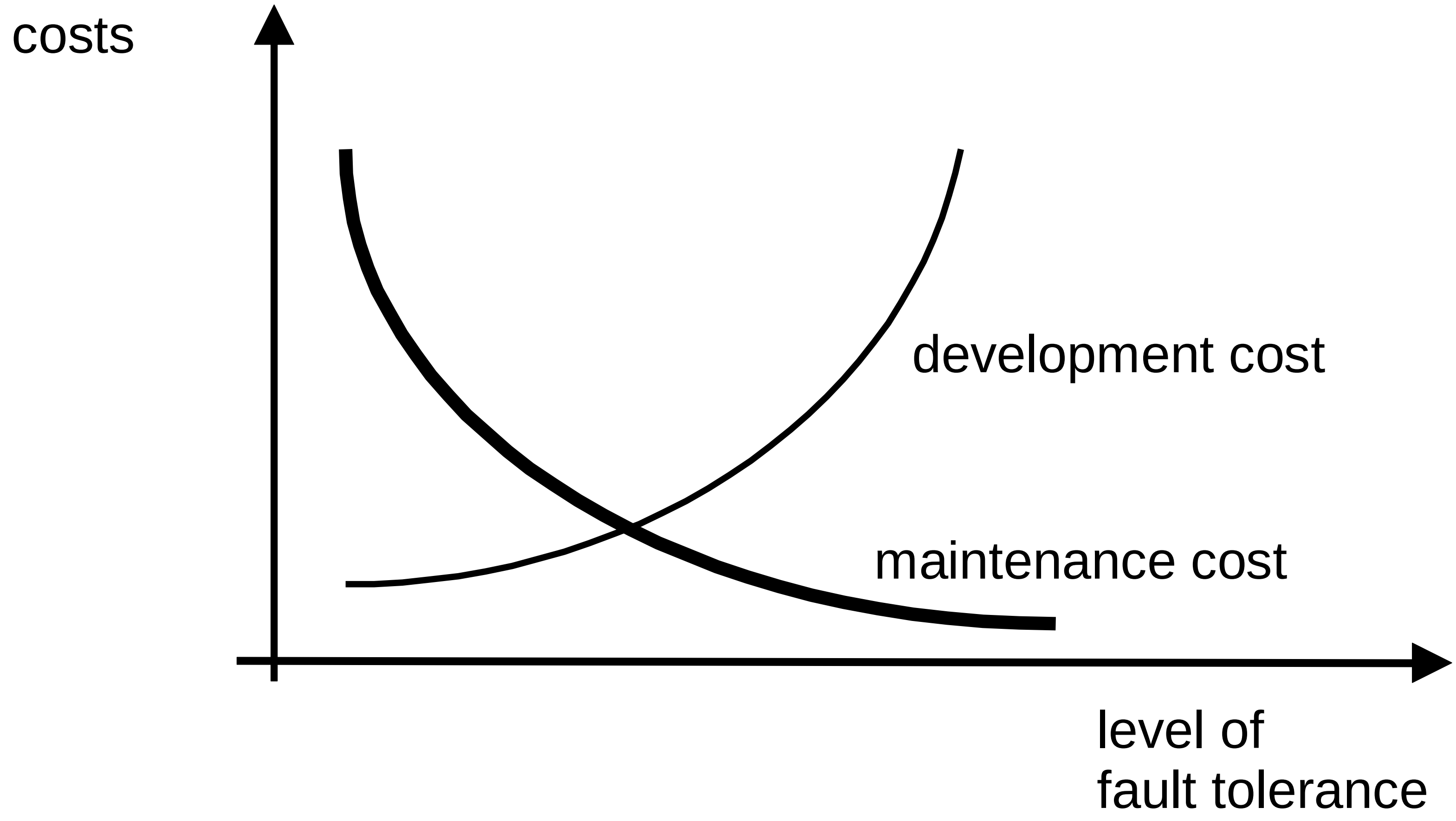
Fault Handling

- Hardware design faults (< 1%):
  - not taken into consideration (see properly tested components)
- Hardware permanent operation faults (10%):
  - hardware redundancy (e.g. spare processor)
- Hardware temporal operation faults (70-80%):
  - time redundancy (e.g. repeated execution)
  - information redundancy (e.g. error correction)
  - software redundancy (e.g. state save and recovery)
- Software design faults (10-20%):
  - software redundancy (e.g. separately designed modules)

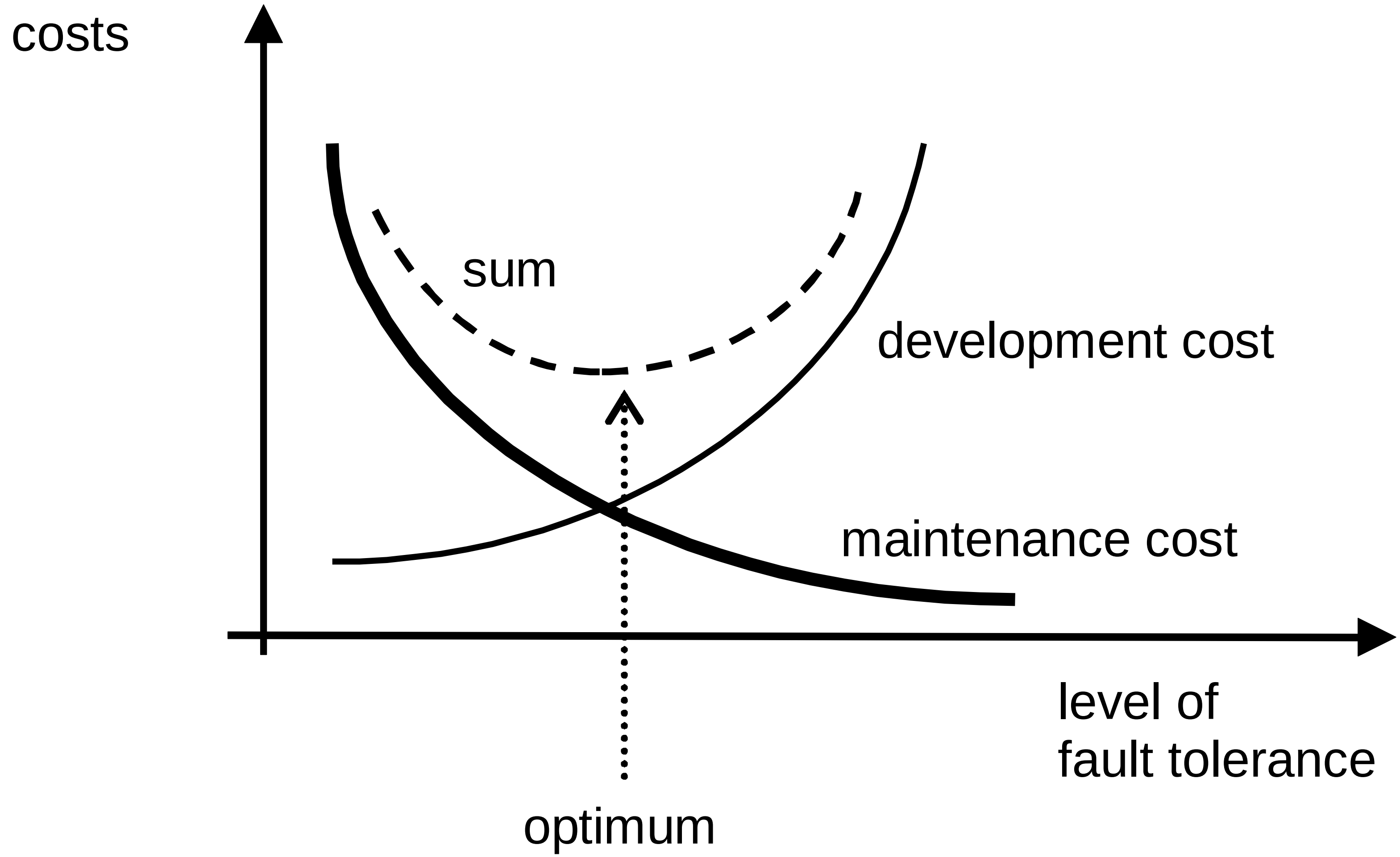
Cost Optimization



Cost Optimization



Cost Optimization



Contents

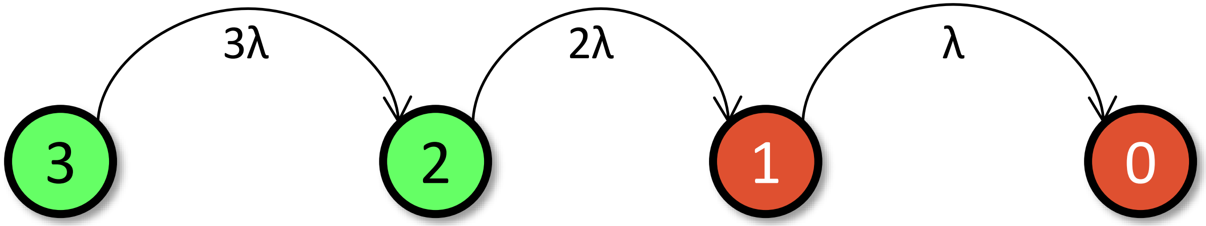
- Concept of service dependability
- Factors affecting service dependability
- Tools of service dependability
- Service dependability analysis

Dependability Analysis

- Why is analysis needed?
  - Is it not enough to provide bountiful redundancy?
- Redundancy is expensive ☹
- Only a properly **designed** redundancy achieves it’s goal!
  - Amount
  - Cold / hot
  - Recovery
  - …

Example

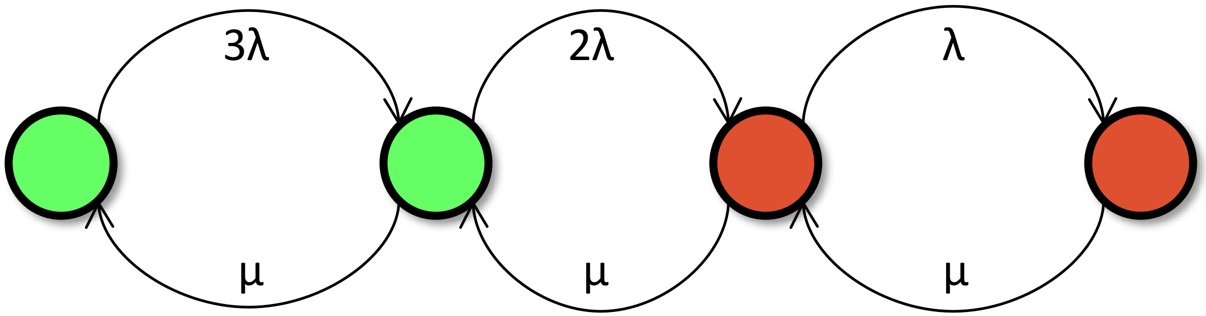
- 3 hard disk RAID-5 array
  - Two disk size usable space, plus parity
  - Tolerates the failure of one disk
  - Rate of first failure: 3λ
    (hot swap → all three disk may fail!)



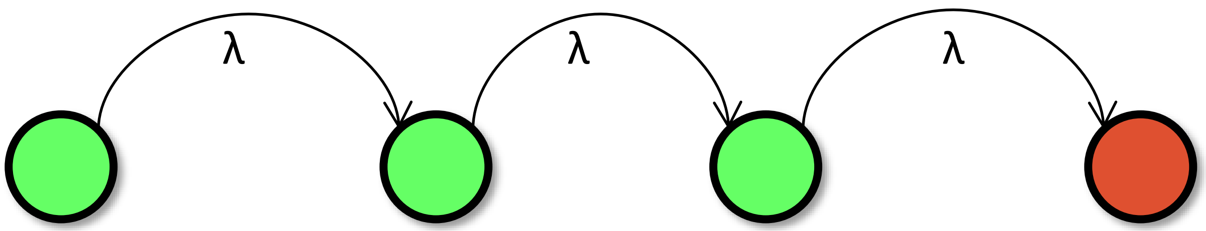
  - MTTF: $1/3\lambda + 1/2\lambda = 5/6\lambda$
  - $5/6\lambda < 1/\lambda$ – one disk is better!

Example

- Redundancy worsened dependability!
- Solution: failed disk must be changed quickly
  - Include repair process in Markov-chain



- Other example: three light bulbs, cold reserve



Dependability Analysis

- Tasks:
  - Identify fault modes, failures
  - Analysis: **qualitative and quantitative**
- Methods
  - Check lists
  - Tables
    (e.g. FMEA: Failure Mode and Effect Analysis)
  - Fault trees
  - State-based approaches (e.g. Petri nets)
  - …

Check List

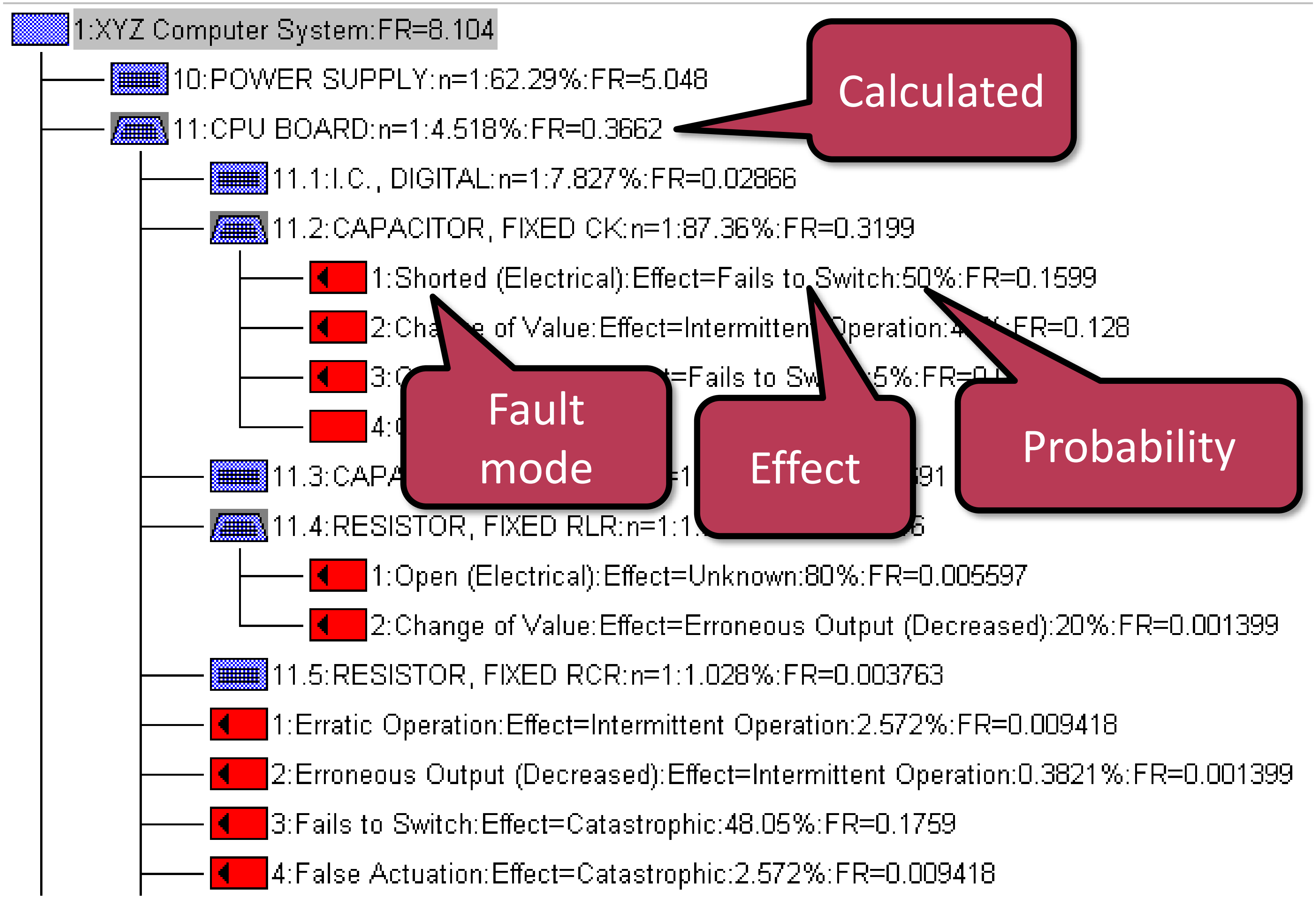
- Technique:
  - Organized collection of experience
  - Use as „rules of thumb”
- Assures:
  - Known fault sources not ignored
  - Employs tried practices
- Disadvantages:
  - List is <u>not complete</u> and difficult to extend
  - Gives false sense of security
  - Usability in other areas is questionable

Failure Mode and Effect Analysis (FMEA)

- List faults and their effects

Component	Failure Modes	Probability	Effect
Webserver	HW fault	10%	Service outage, replace comp.
	SW update	90%	Temporal outage
SQL server	Disc full	20%	Only static content is available
…			

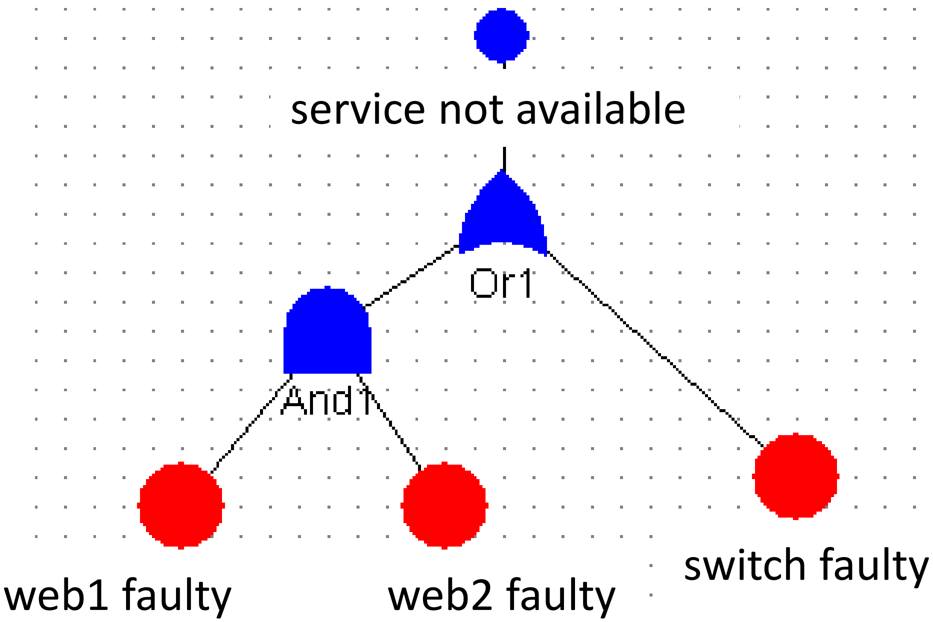
Example: Control electronics



Fault Tree

- How can the root failure occur?

- Components (partial)
  - AND gate
  - OR gate
  - Rectangle: subsystem
  - Circle: base level failure



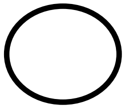
Fault Tree - Analysis

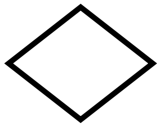
- Qualitative:
  - identify single point of failure (SPOF)
  - critical event: can cause failure on multiple paths

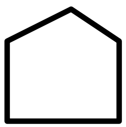
- Quantitative:
  - probability for basic failures (hard: where to get correct data?)
  - calculate properties of root (e.g. reliability)
  - Problems: not independent events…

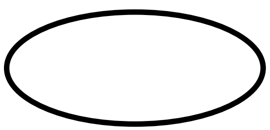
Graphical Component Set of Fault Trees

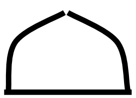

top level or intermediate event

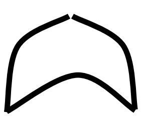

primary (base level) event


event not evaluated further

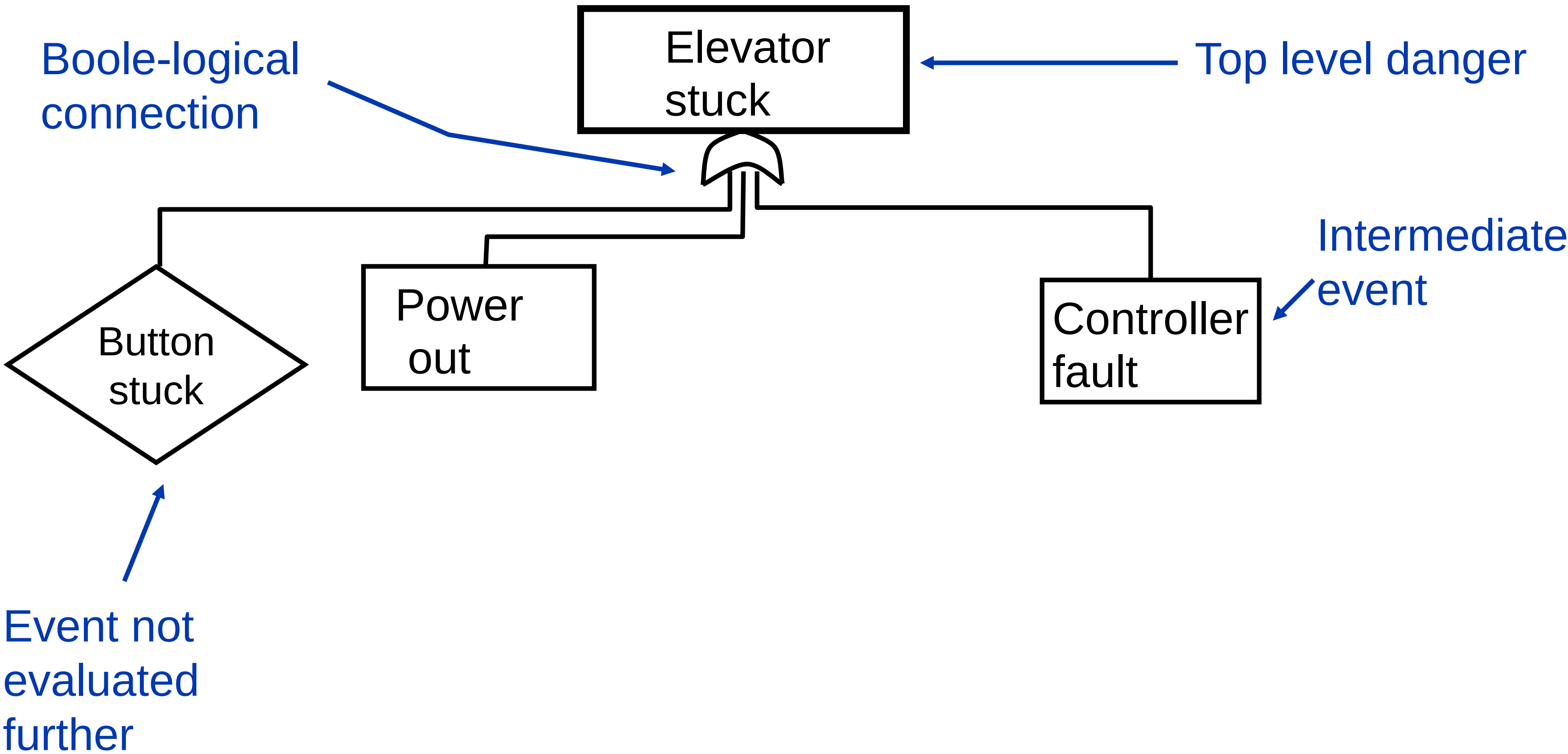

regular event (not fault or danger)


condition for the occurrence of a complex event

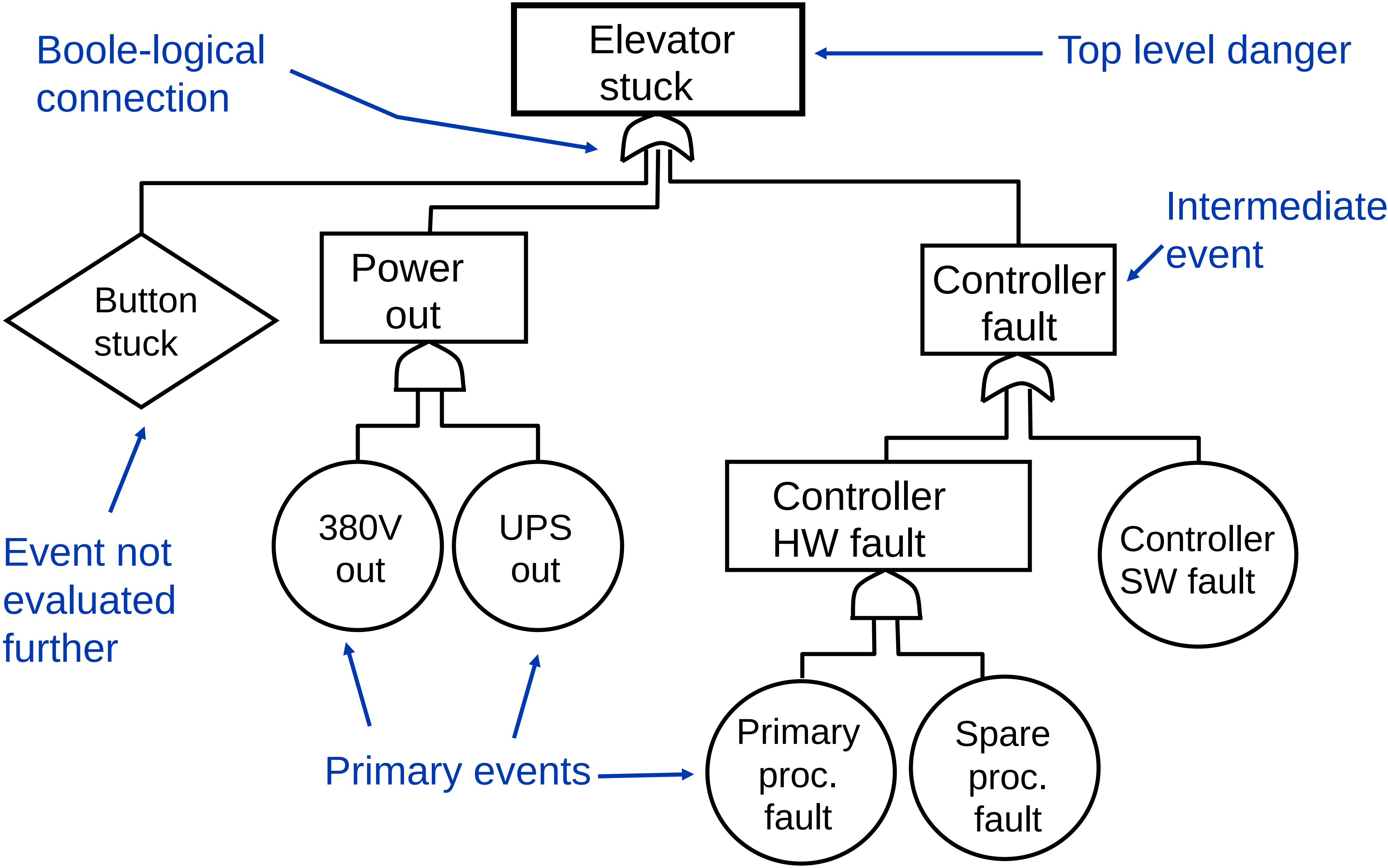

AND gate


OR gate

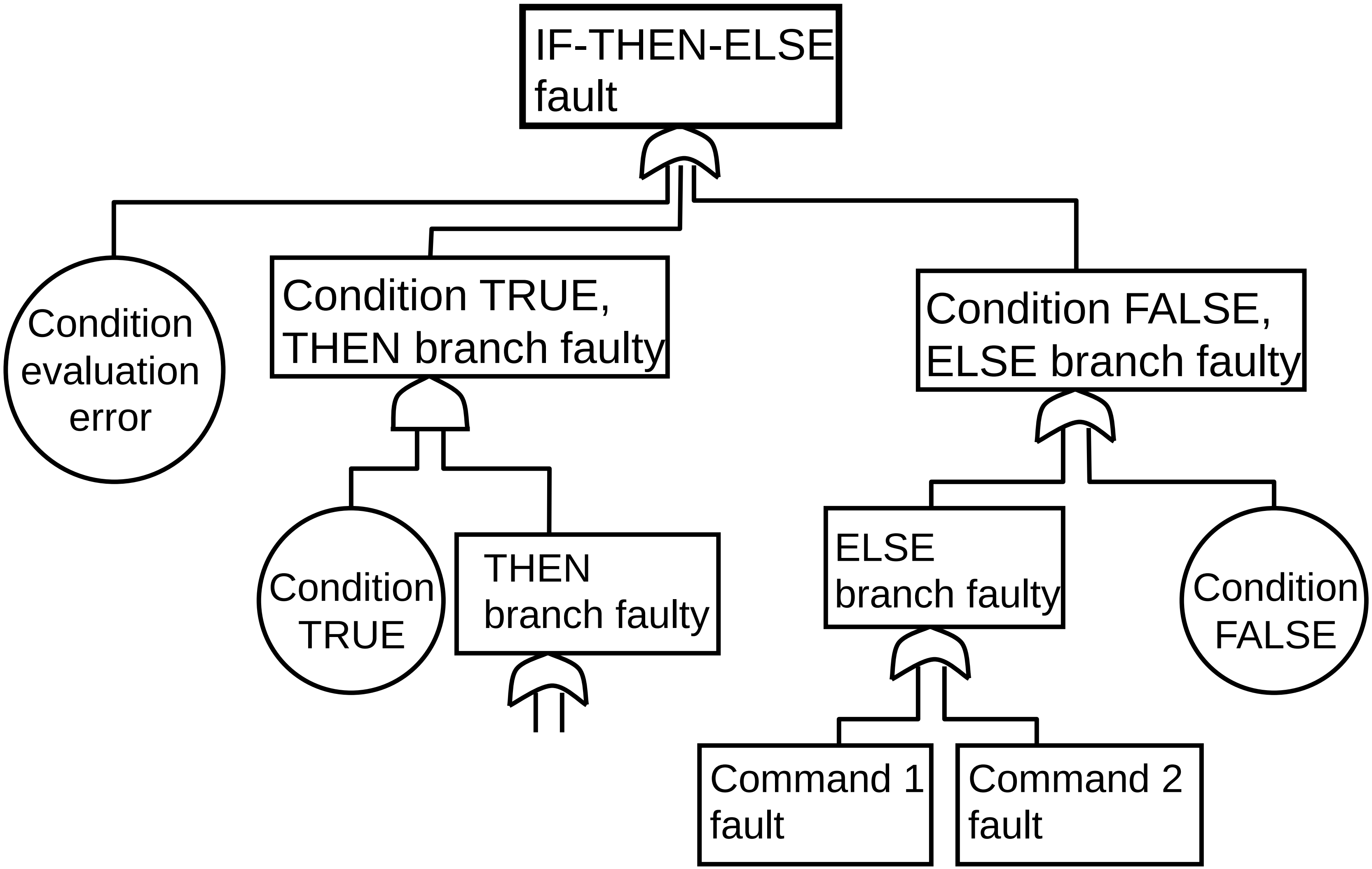
Fault Tree Example: Elevator



Fault Tree Example: Elevator



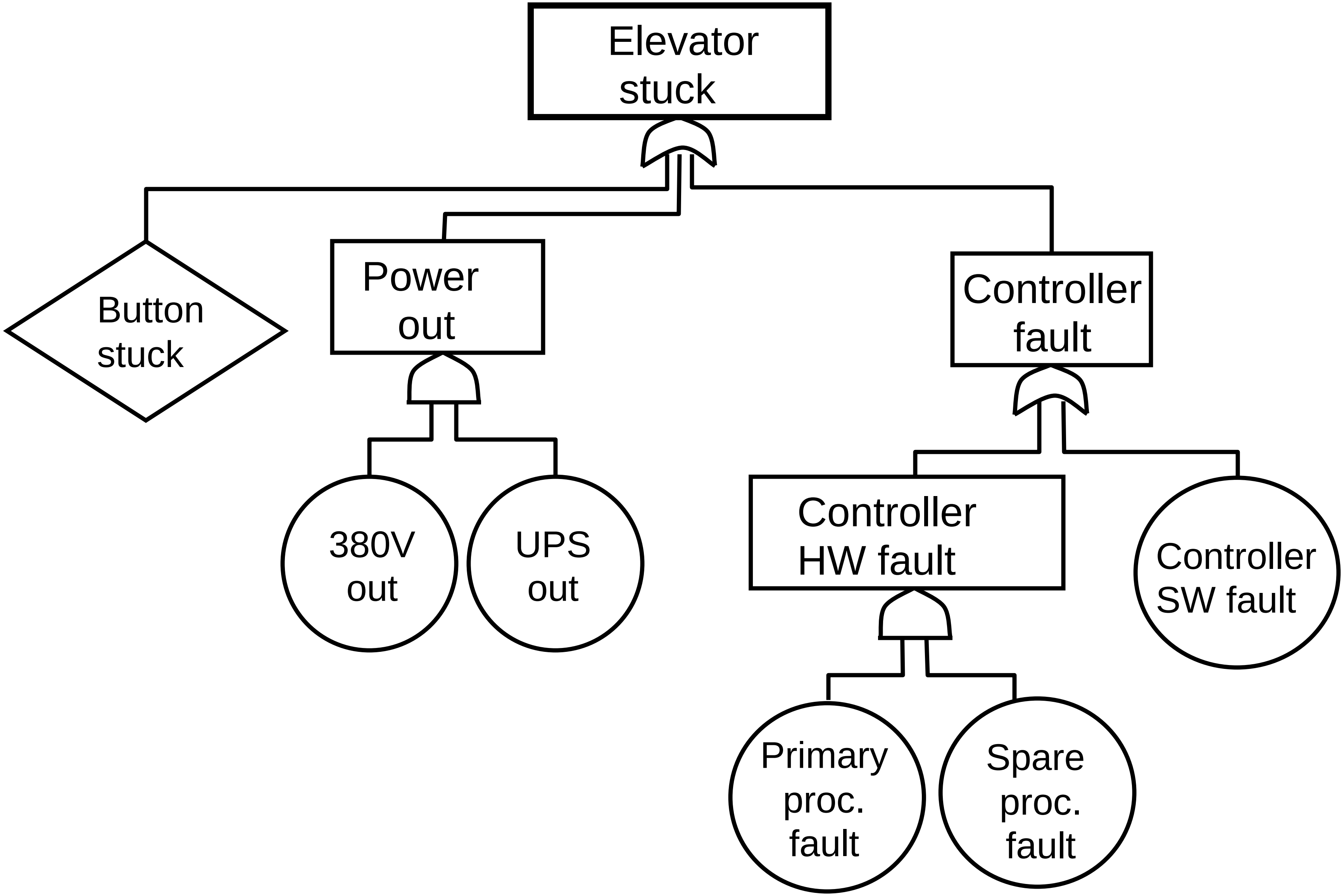
Fault Tree Example: Software Pattern



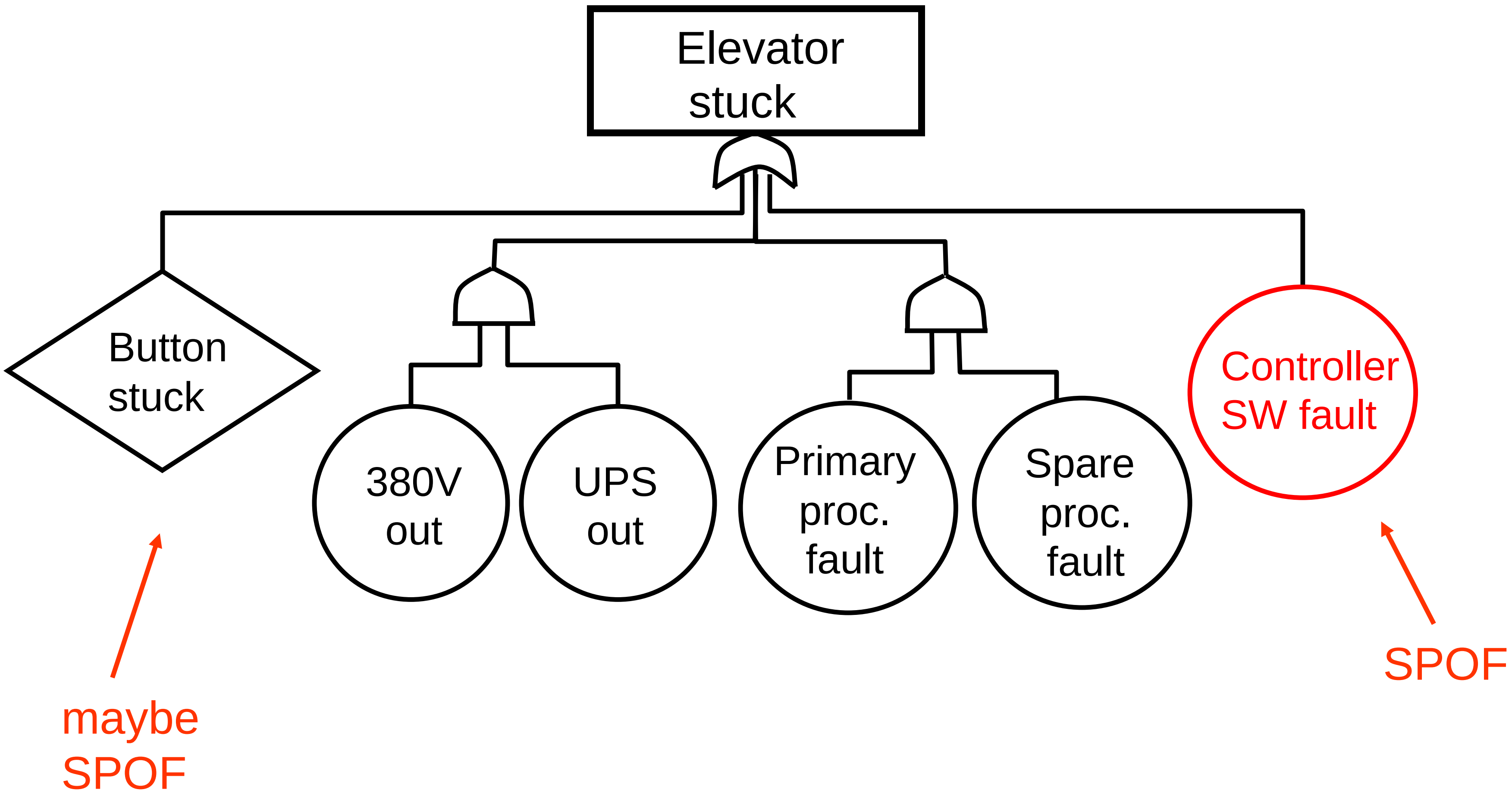
Qualitative Analysis

- Fault tree reduction: Resolve intermediate events and pseudo events
  $\rightarrow$ disjunctive normal form (OR on top)
- Cut:
  Primary events connected with AND gate
- Minimal set of cuts: Reduction not possible
  - No set, for which a subset can also be found
- Identifiable:
  - single point of failure (SPOF)
  - critical events (appears in more than one cut)

Fault Tree Example: Elevator



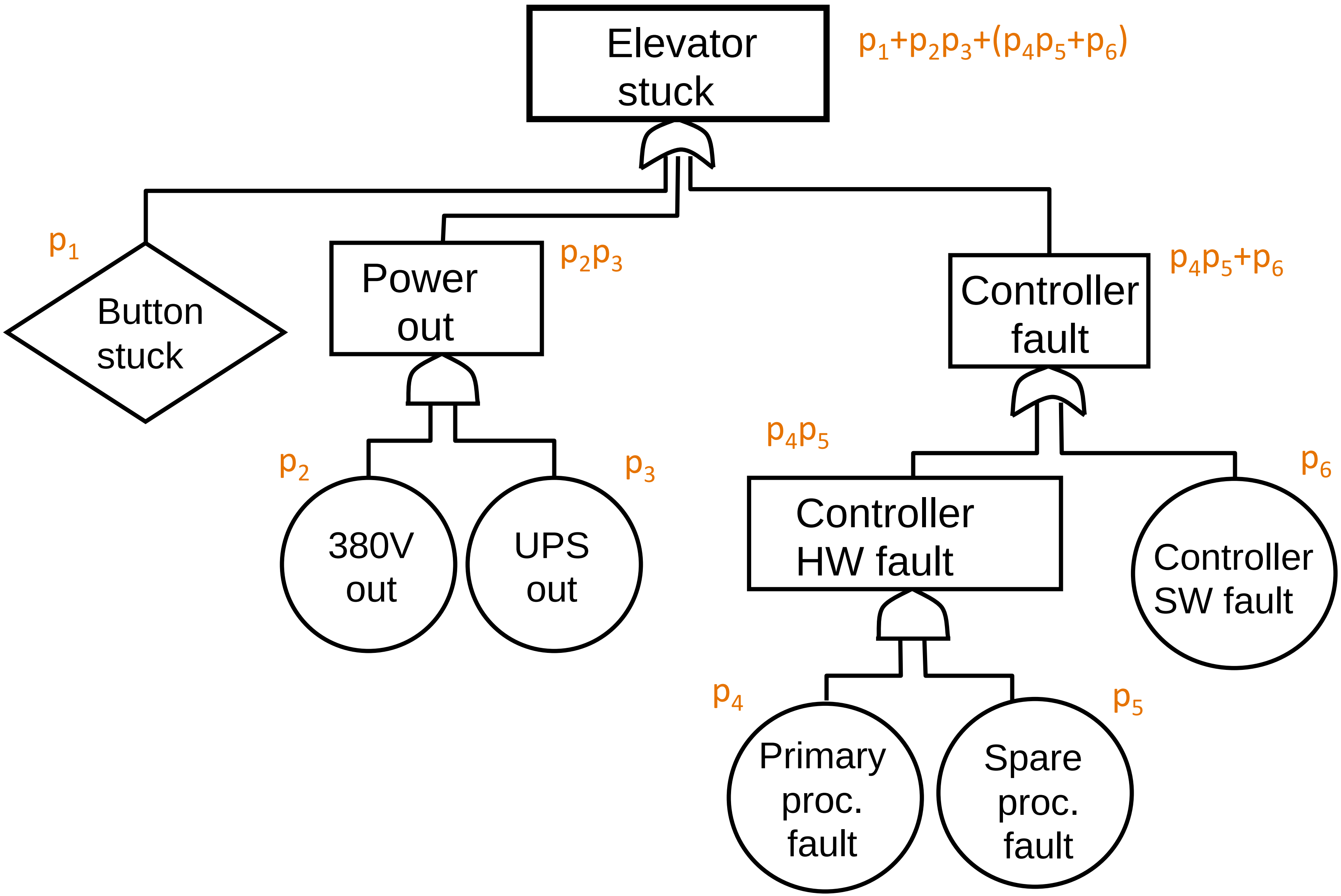
Reduced Fault Tree Example: Elevator



Quantitative Analysis

- Probabilities assigned to primary events
    - component data, experience, estimate
- Calculate probability of top-level danger
    - AND gate: product (if independent events)
      precise: P{A and B} = P{A}P{B|A}
    - OR gate: sum (over approximation)
      precise: P{A or B} = P{A}+P{B}-P{A and B}<=P{A}+P{B}
- Problems:
    - correlating faults
    - handling (fault) sequences over time

Fault Tree Example: Elevator

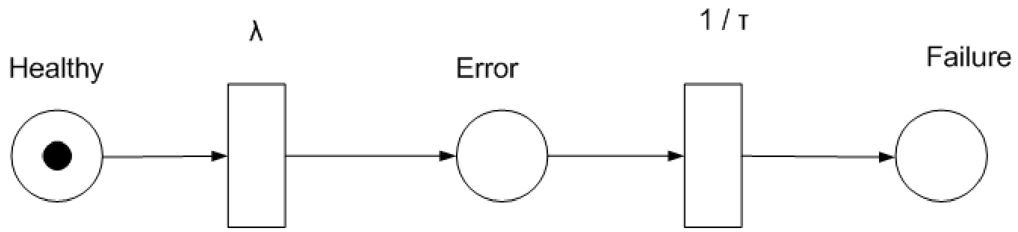


Failure Rates

- Basis of analysis: fault probabilities
- Where to get good data:
  - Estimate
  - Own monitoring system
  - External studies, numbers (credibility, precision?)
- Examples:
  - Cisco switch MTBF ~ 200000 hours (=22,8 years)
  - IBM S/390 mainframe MTTF 45 years
  - Windows XP MTTF 608 hours
  - web server MTTF ~ 16 days…

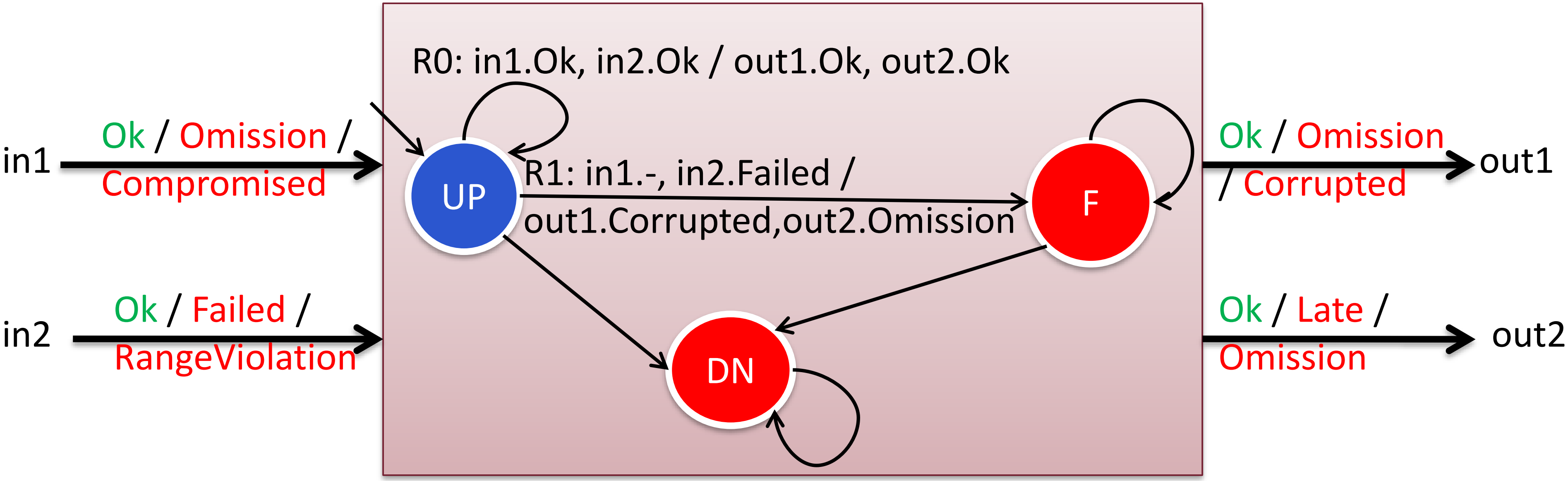
State Based Techniques

- Qualitative description of faults: discrete behavior model
  - State machine, data flow network, process, Petri-nets…
- Quantitative: timing for state transitions
  - Deterministic
  - Based on probability distribution: continuous time, markovian stochastic



Fault Modelling with Data Flow Networks

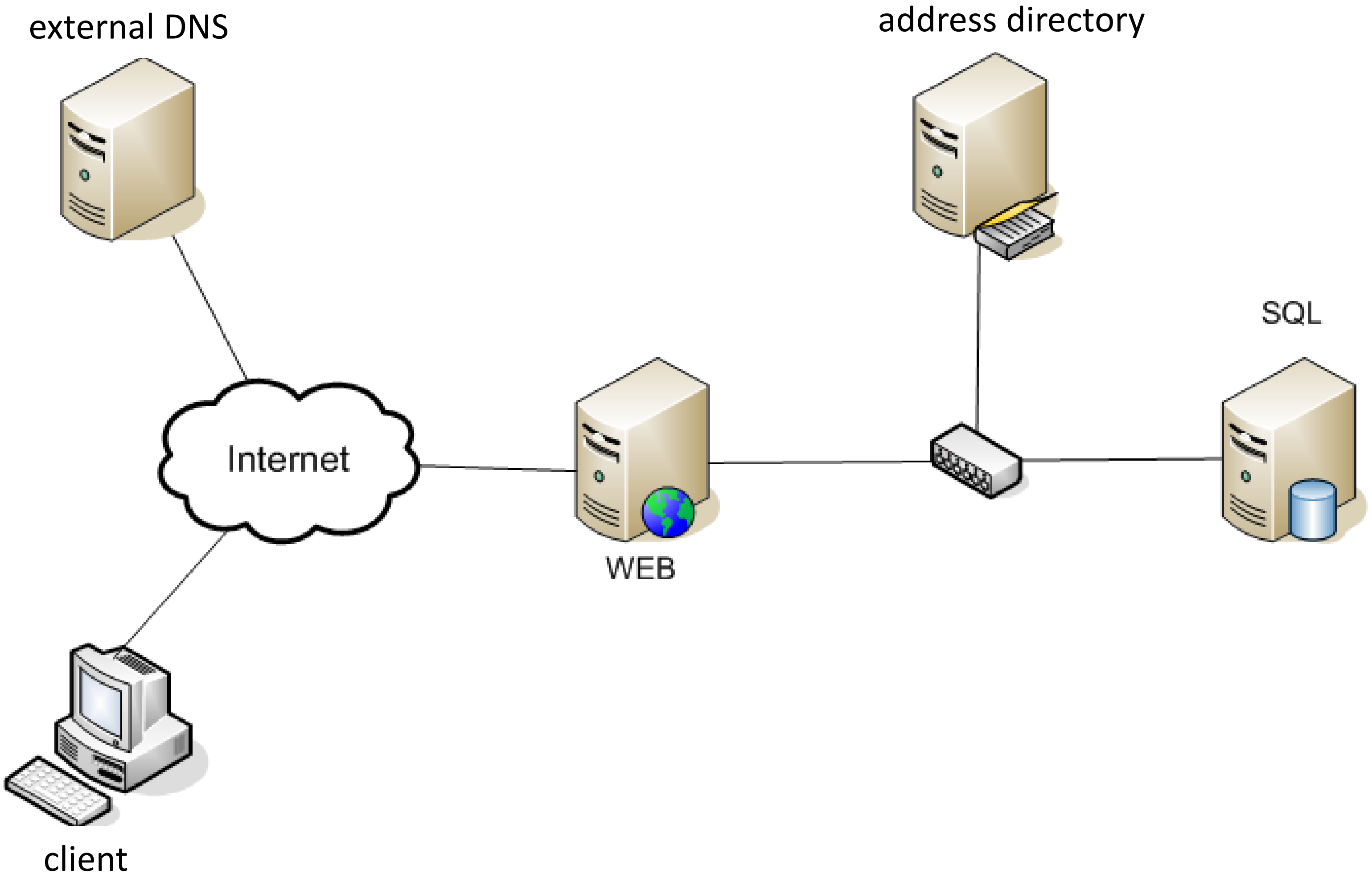
- Qualitative fault model → data flow network
  - Component→ data flow node
  - Internal fault modes → node states
  - Component connections → channels
  - Communication faults → channel tokens



Fault Modelling with Data Flow Networks

- Fault propagation
  - Faulty component state → faulty message
  - Faulty message → faulty component state
- Qualitative analysis
  - Forward: what is the consequence of an fault?
  - Backward: what is the cause of a failure?
- One (not complete) solution technique:
  - Constraint Satisfaction Problem (CSP)

Example: Dependability Analysis



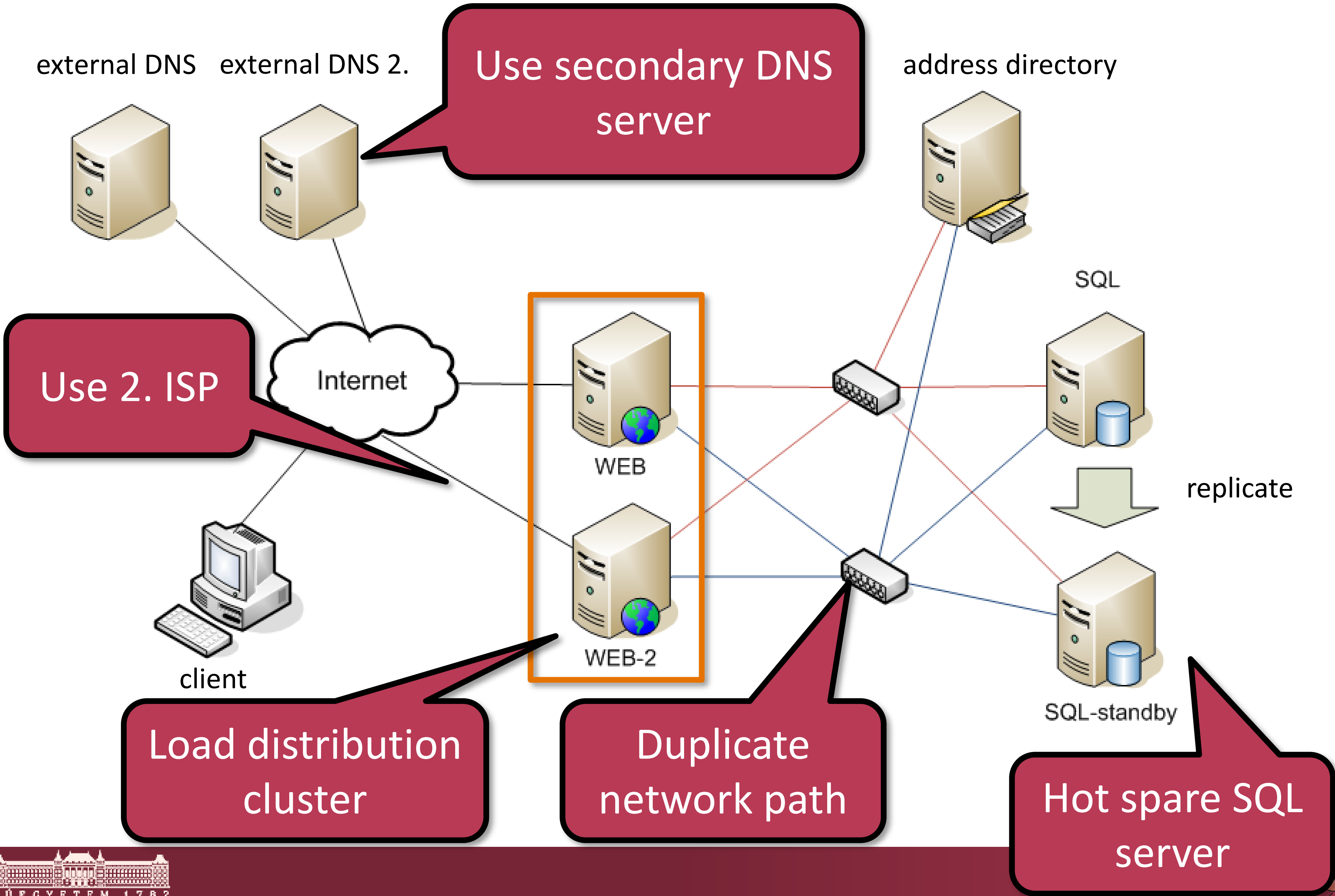
Task: What kind of faults will make the service unavailable (web store)?

Task: Identify Fault Modes

- What kind of faults will make the service unavailable (web store)?

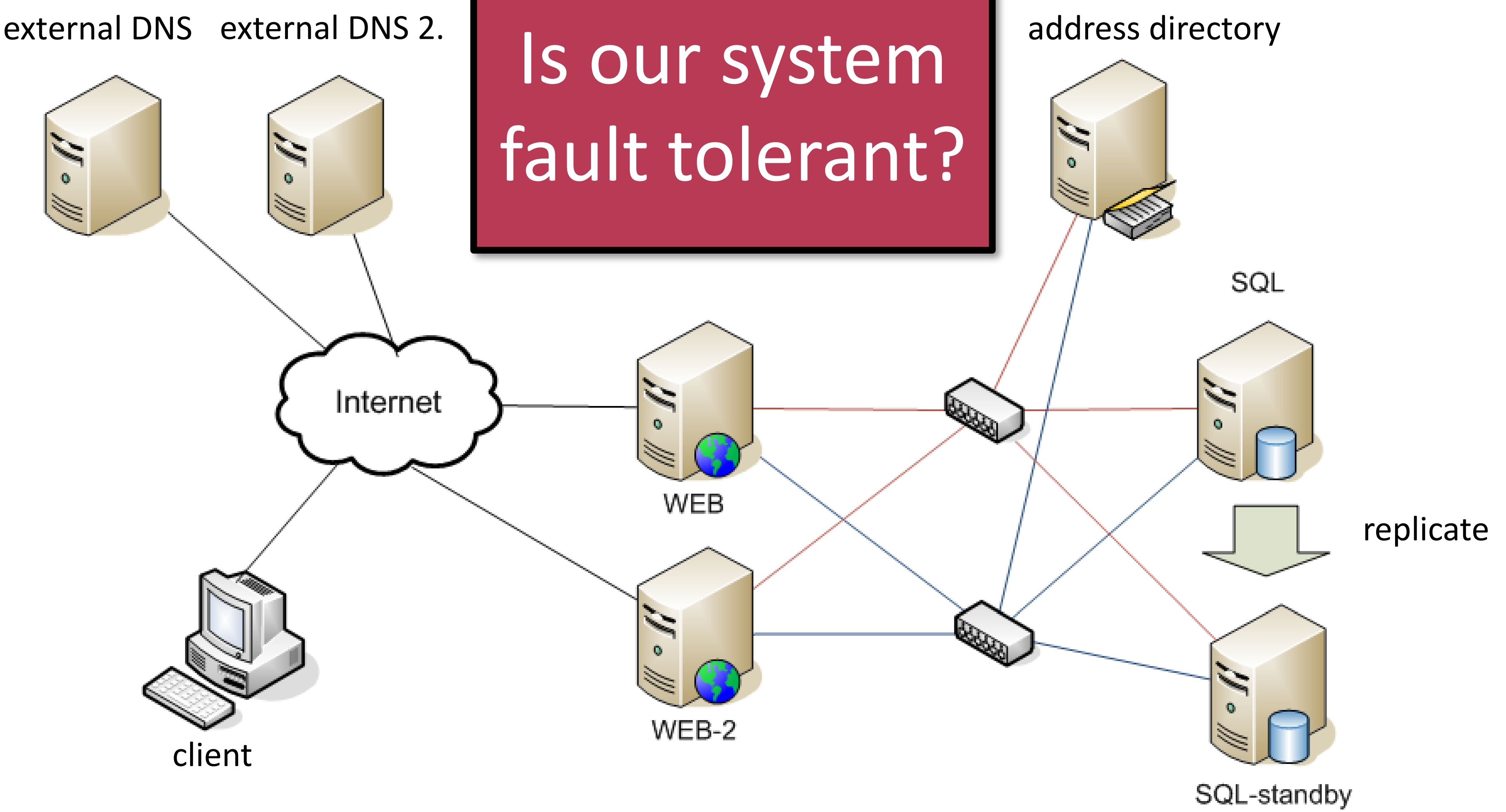
- Power outage, HW fault, network component/cable fault, server service faults, application fault, install update, overload, attack, misconfiguration, version incompatibility, virus…

Example: Incorporate Fault Tolerance



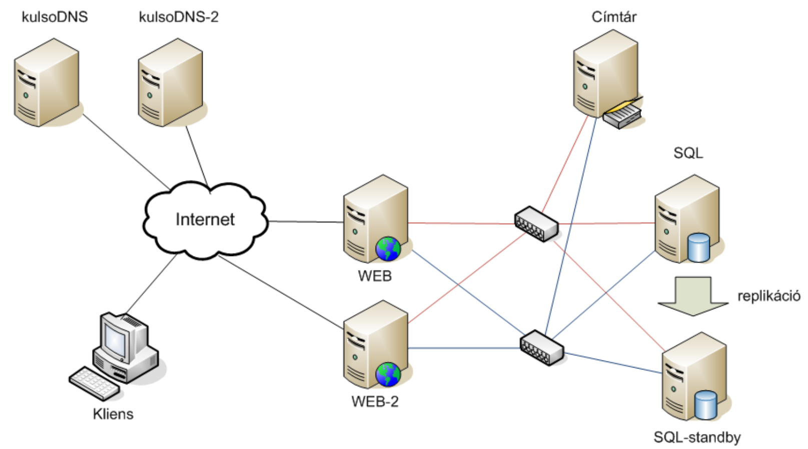
Example: Incorporate Fault Tolerance

Is our system fault tolerant?



Example: Incorporate Fault Tolerance

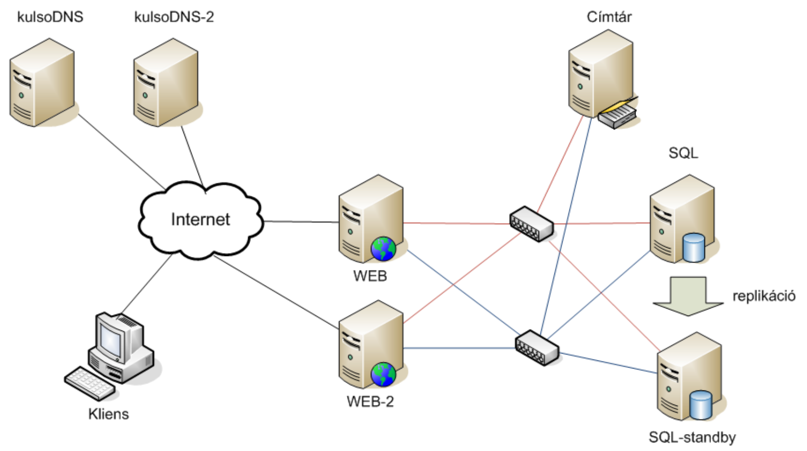
Is our system fault tolerant?



- Depends:
  - We are protected from some SPOFs
- BUT
  - many fault options are left
  - Delete data, destruction of complete server room, administrator faults, OS hotfix needs restart…

Example: incorporate fault tolerance

Is our system fault tolerant?



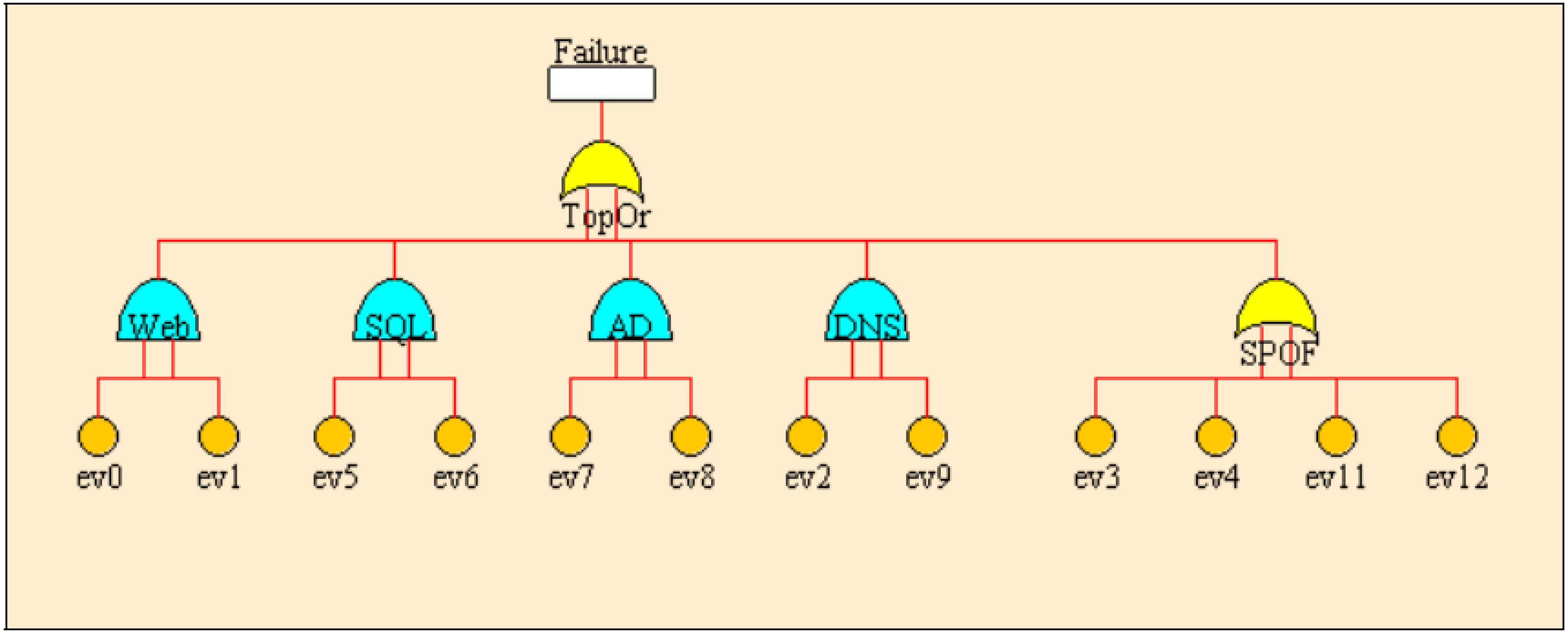
- De
  -
- BU
  -
  - n,

administrator faults, OS hotfix needs restart…

Moral: always know,
- against what you want to protect,
- what techniques do you have,
- is it worth to protect it

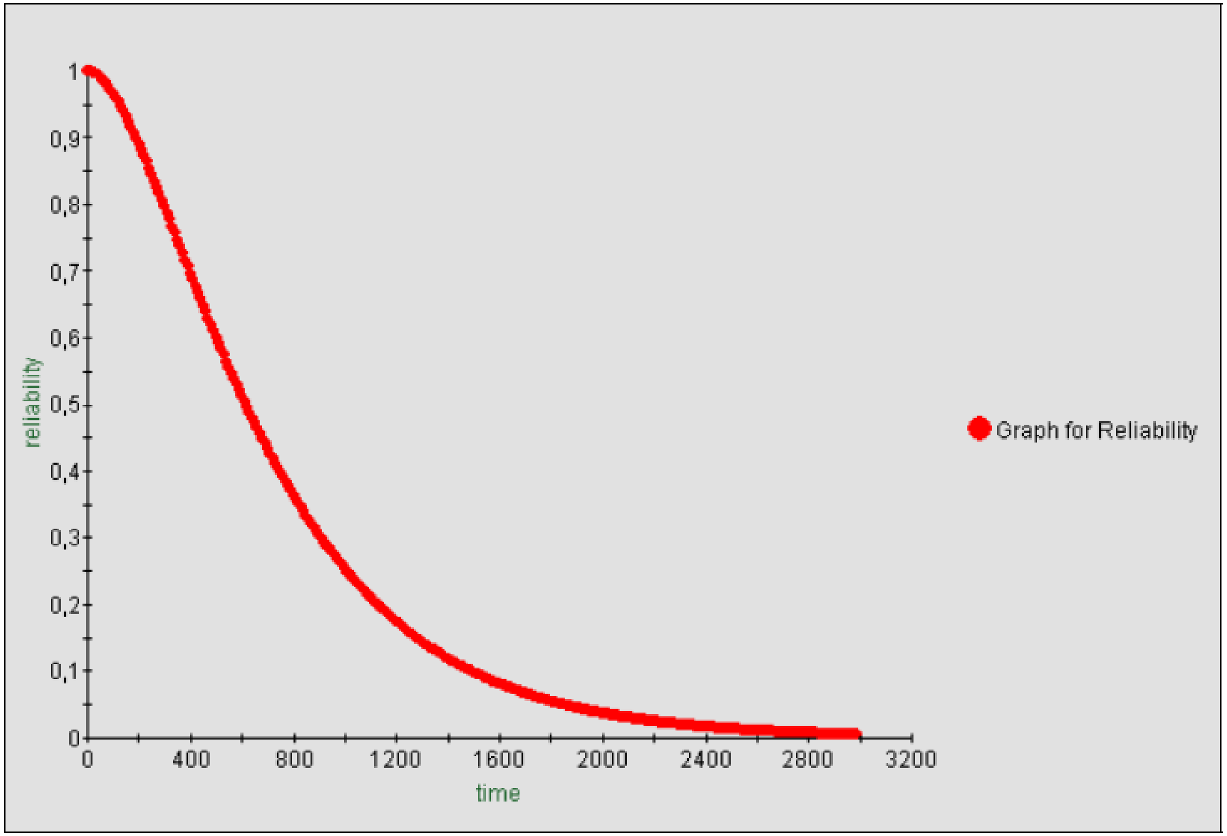
Analysis: Fault Tree

- SHARPE tool
- Draw fault tree



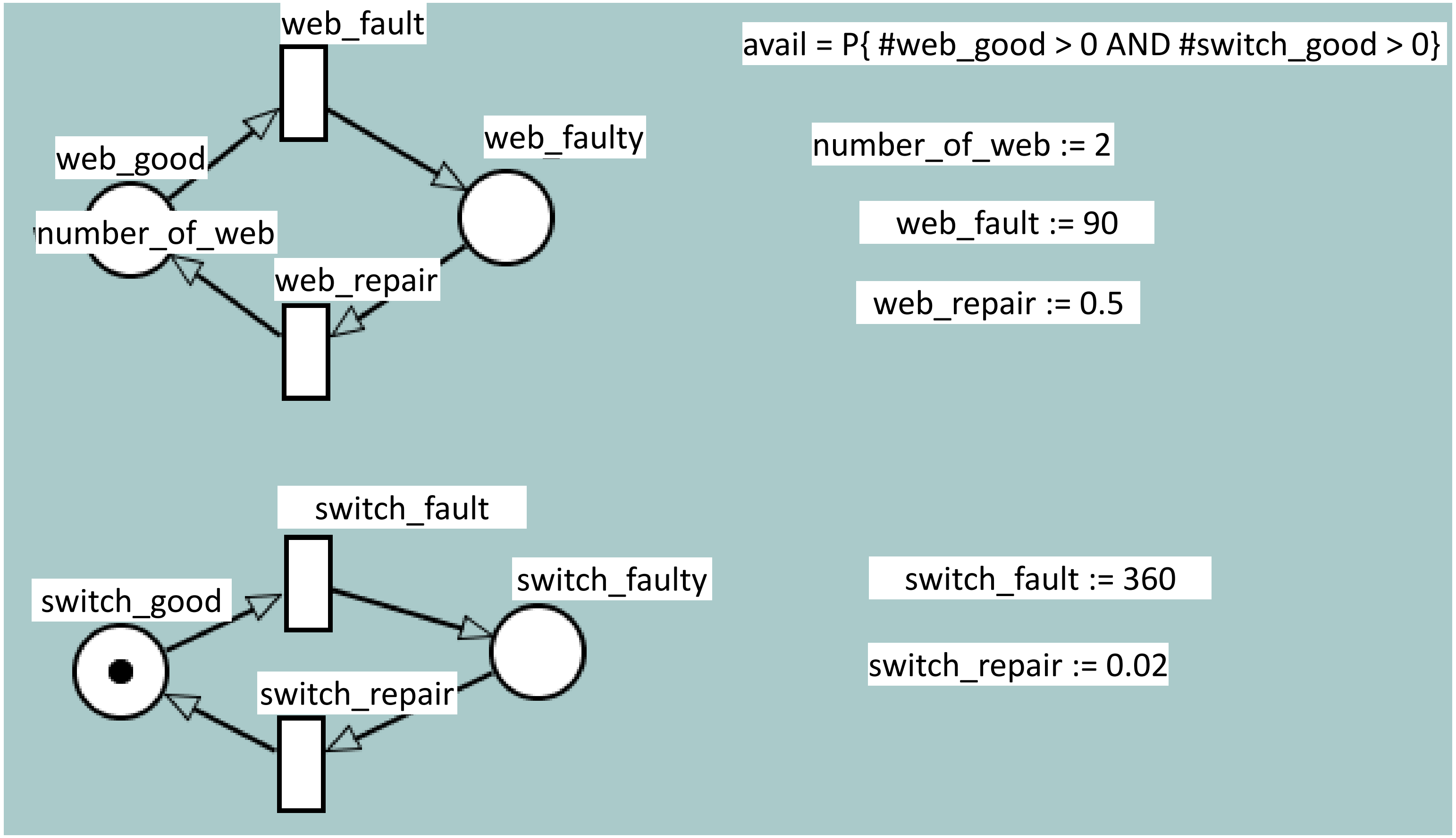
Analysis: Fault Tree

- Assigning occurrence probability to primary events
- Determining system reliability:



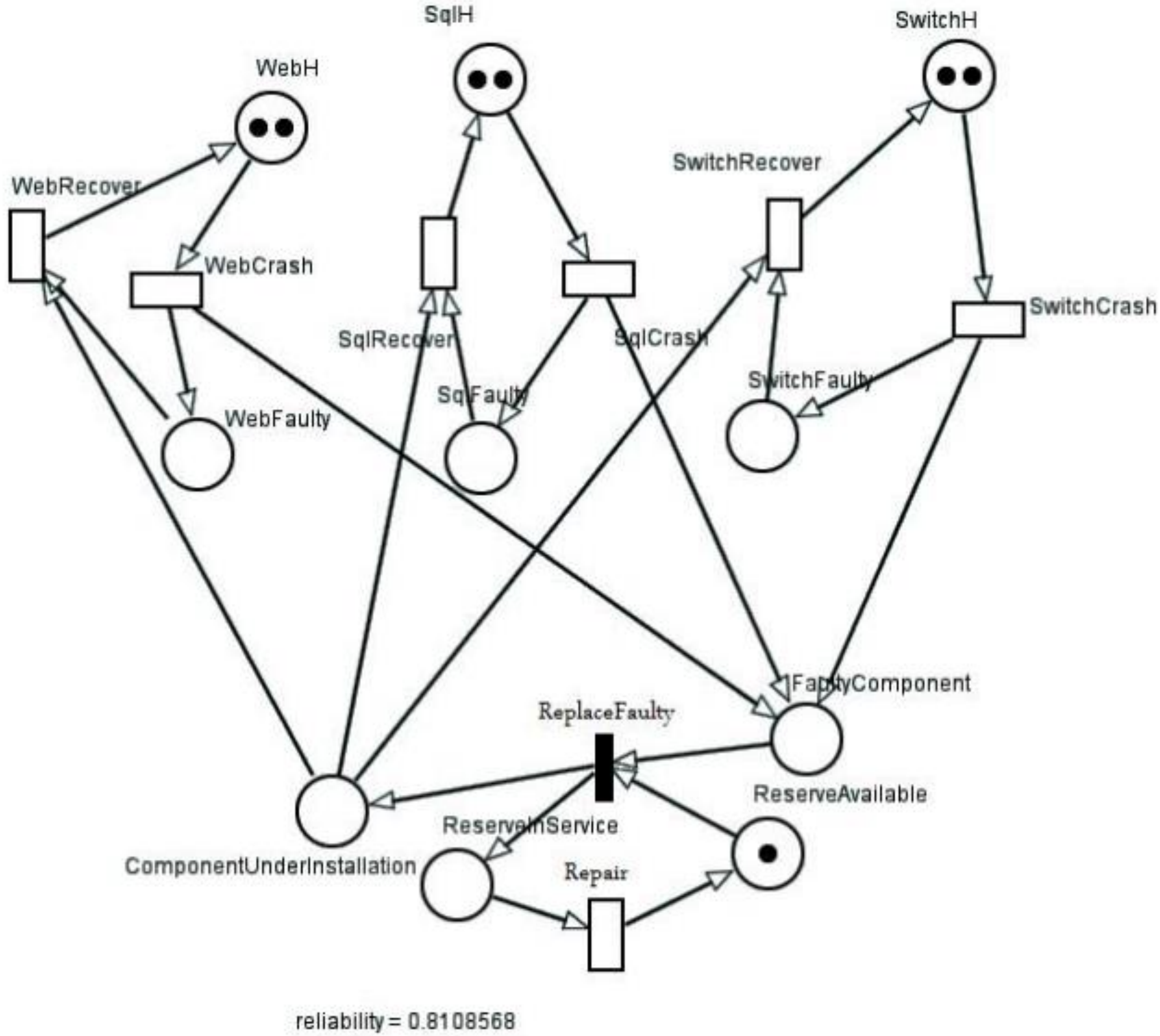
Analysis: Petri-net

- TimeNET tool
- Basic blocks and parameters



Analysis: Petri-net

Complete model:



Summary

- Dependability
  - Characteristics, propagation chain, tools
- Fault tolerance
  - Appearance of redundancy
- Analysis:
  - Technical and mathematical methods
  - Identification of fault modes
  - Select appropriate protection technique