Principles of AI Engineering Chapter 5: Requirements and risk

Prof. Dr. Steffen Herbold

Credit:

Based on contents from Christian Kästner (https://github.com/ckaestne/seai)

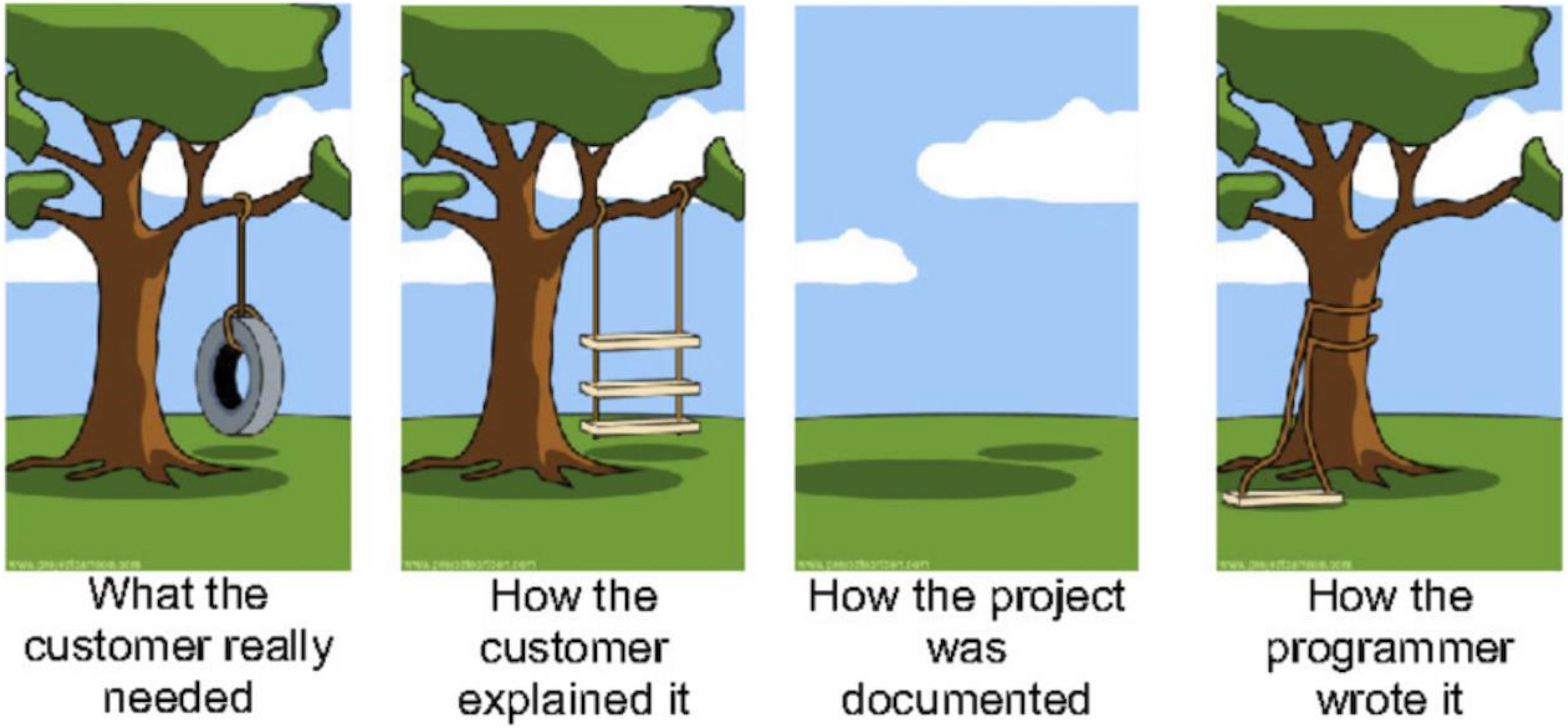
Contents

- Software requirements
- Risk analysis
- Strategies for handling faults in ML-based systems
- Constraints and trade-offs

Software requirements

Software requirements

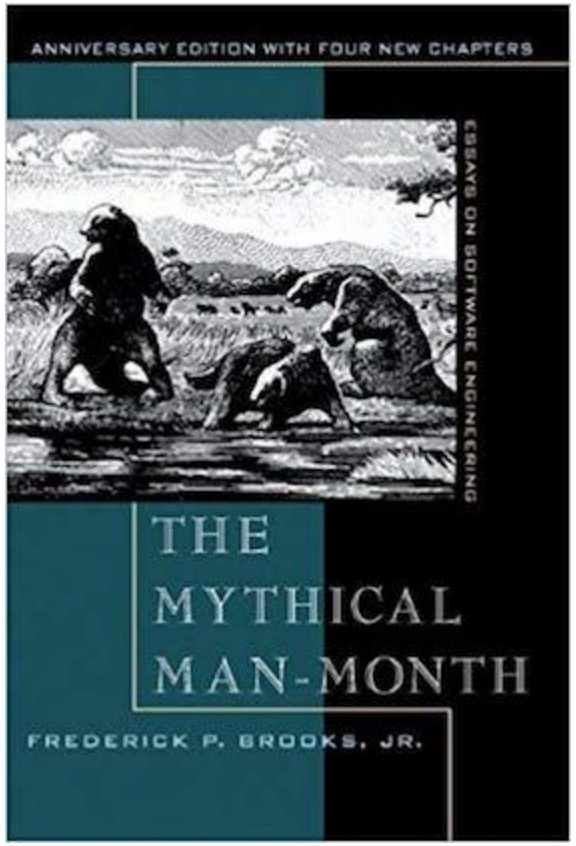
- Describe what the system should do, in terms of the services that it provides and their qualities (safety, reliability, performance)
- Gathered through requirements elicitations from stakeholders, standards, systems in use, …



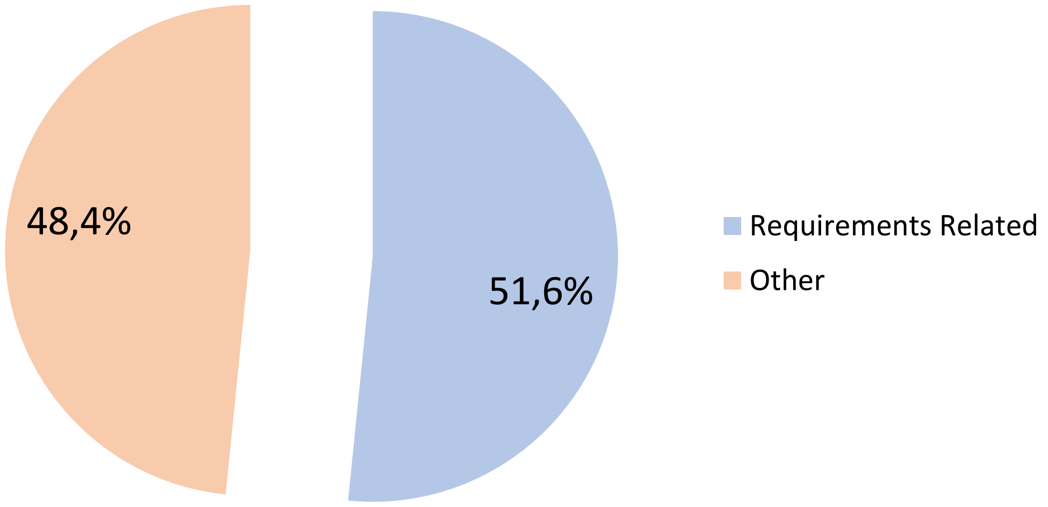
Importance of requirements

The hardest single part of building a software system is deciding precisely what to build. No other parth of the work so cripples the resulting system if done wrong.

Fred Brooks, The Mythical Man Month (1975)



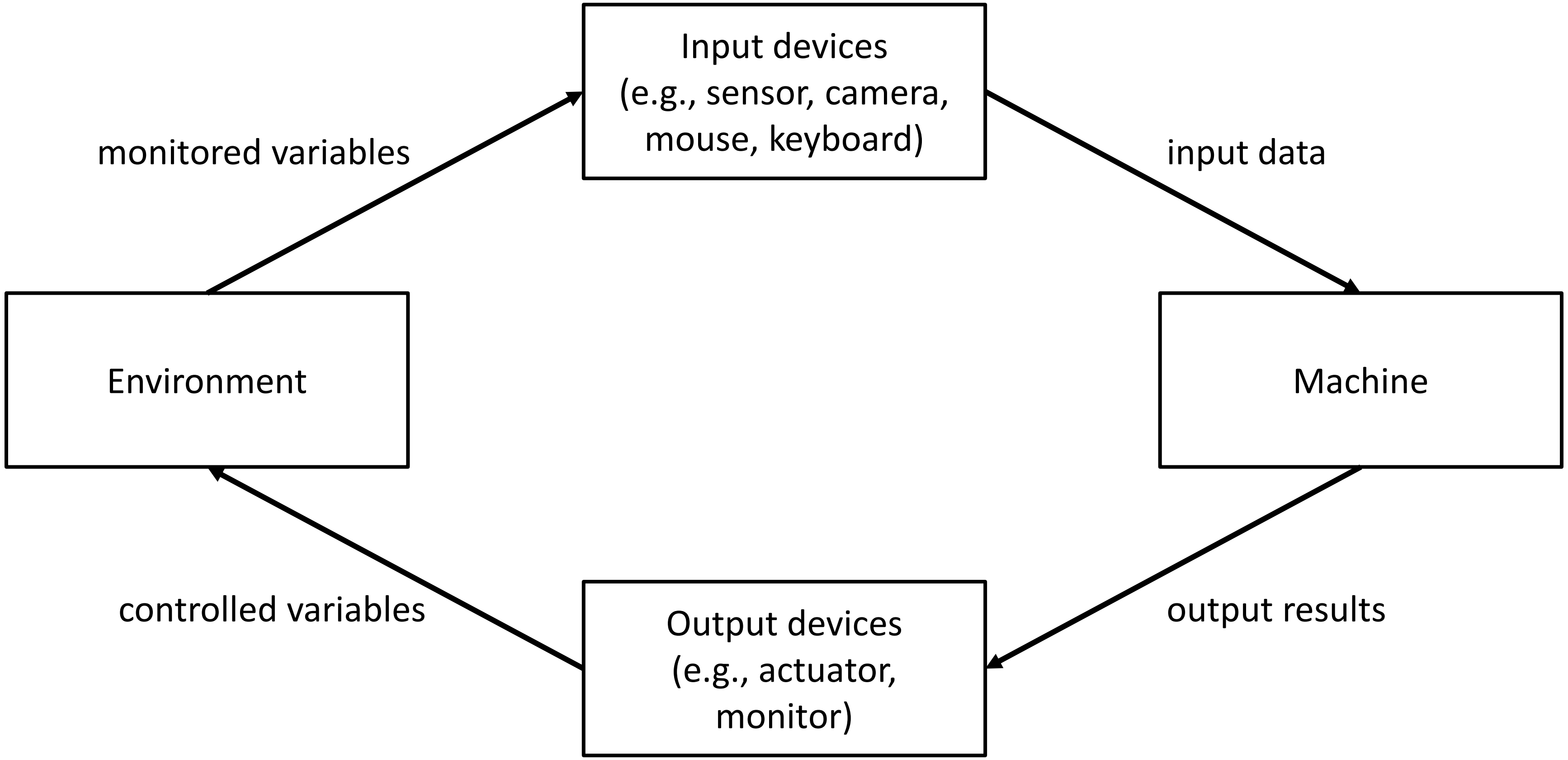
Reasons for project failure



Source: Standish Group 1995 (www.standishgroup.com)

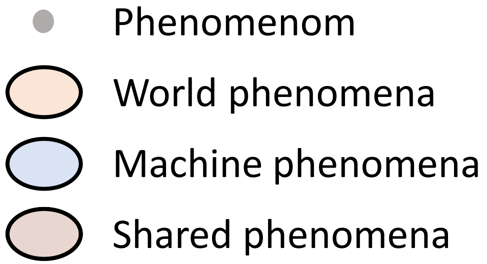
… there are also newer studies with similar results

Software does not live in a vacuum



Requirements describe desired states of the world

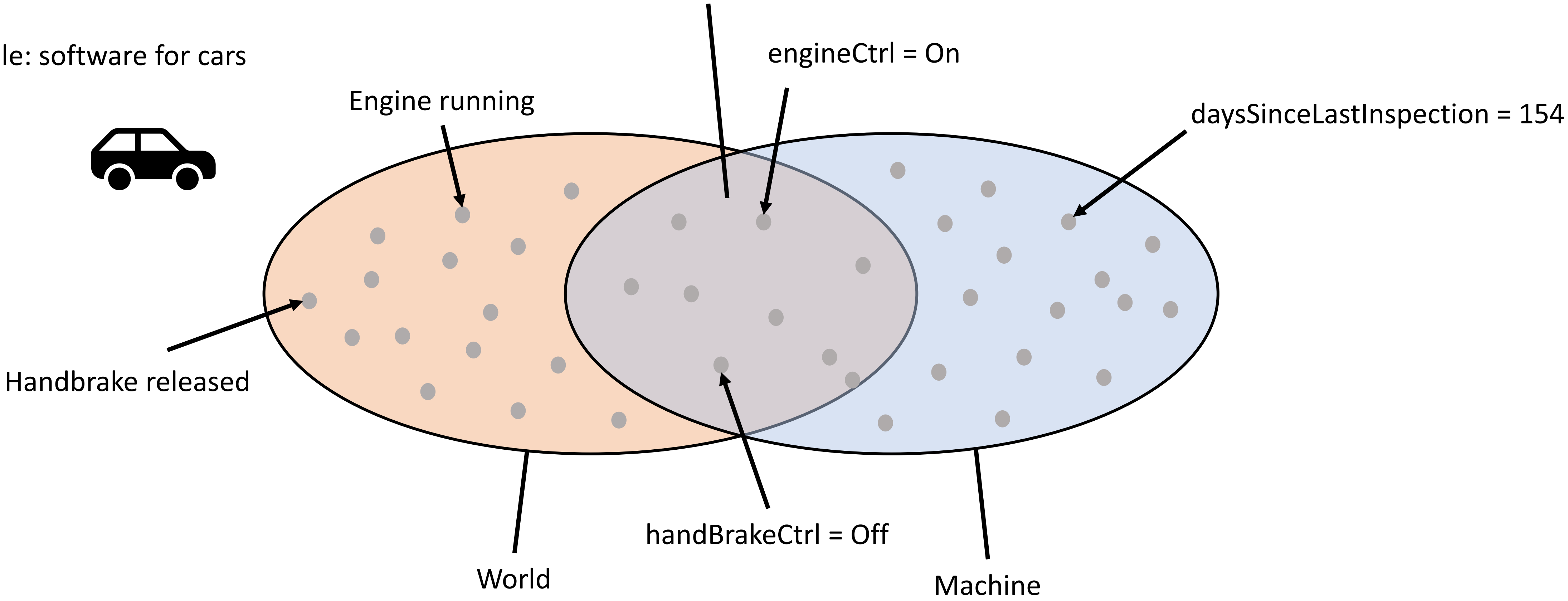
Software/machine is designed to sense and manipulate state of the world



Shared phenomena

Shared phenomena = interface between environment and software

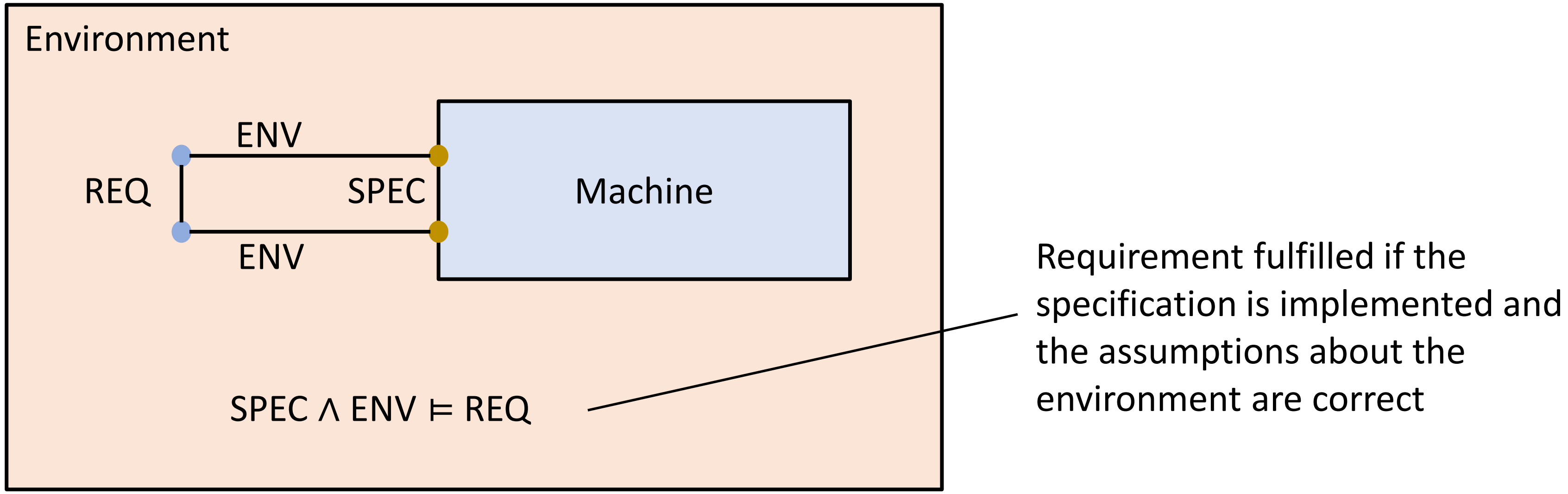
Example: software for cars



Software can influence environment only through the shared interface!

Unshared parts beyond control, we can only assume behavior!

Environment, specification, and requirements



Requirement (REQ): What the system must ensure, in terms of desired effects on the environment
Specification (SPEC): What the software must implement, expressed over shared phenomena
Environment (ENV): What‘s assumed about the behavior/properties of the environment. Bridges the gap between SPEC and REQ

Software cannot satisfy requirements on its own and must rely on environment!

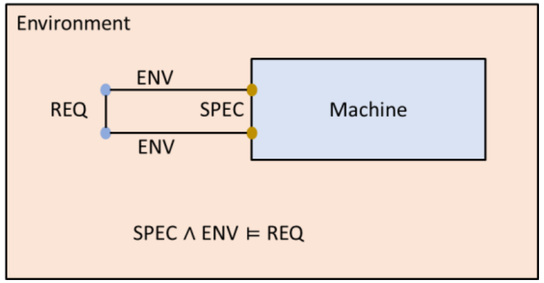
Example: Lane keeping assistance system with lane departure warning

REQ: The vehicle must be prevented from veering of the lane.

SPEC: Lane detector accurately identifies lane markings in the input image.
The controller generates correct steering commands.

ENV: Sensors provide accurate information about the environment.
Driver responds when given warning.
Steering wheel is functional.

What could go wrong?



- REQ: Wrong, inconsistent or infeasible requirements
- ENV: Missing or incorrect assumptions about the environment
- SPEC: Wrong or violated specifications
- SPEC ∧ ENV: Inconsistency between specification and assumptions about the environment

Lufthansa 2904 Crash

- REQ: Reverse thrust is enabled if and only if the plane is on the ground
- SPEC: Reverse thrust is enabled if and only if the wheel is turning
- ENV: Wheel is turning if and only if the plane is one the ground. Wheel is turning when plane moves on the ground.

- What happened?
  - The wheels did not start moving immediately when the plane touched the ground due to a wet runway
  - Reverse trust could not be used because because the assumption about the environment was wrong
  - The plane failed to brake in time and two people died.

Assumption violations in ML-based system

- Unrealistic or missing assumptions
  - Poorly understood impact of wheather condition on sensor accuracy, missing pedestrian behavior, assumptions about how users interact with a user interface, …
- Concept drift
  - Environment evolves over time and the underlying distributions change
  - Adversaries intentionally cause concept drift by poisoning data
- A malicious actor deliberately tries to violate or invalidate assumptions on the environment
  - Placing fake traffic signs or manipulating traffic signs
  - Initiating feedback loops
- System itself may change environment over time and invalidate assumptions
  - System interacts with environment and can change behavior

Example: Lane keeping assistance system with lane departure warning

Live exercise: What could go wrong?

Generic process to establish requirements

- Identify environmental entities and machine components
- State a desired requirement (REQ) over the environment
- Identify the interface between the environment and the machine
- Identify the environment assumptions (ENV)
- Develop specifications (SPEC) that are sufficient to establish REQ
- Check if $\text{SPEC} \land \text{ENV} \vDash \text{REQ}$
- If not, go back to the beginning and repeat

Risk analysis

What is risk analysis

- What can possibly go wrong in my system and what are potential impacts on the requirements?

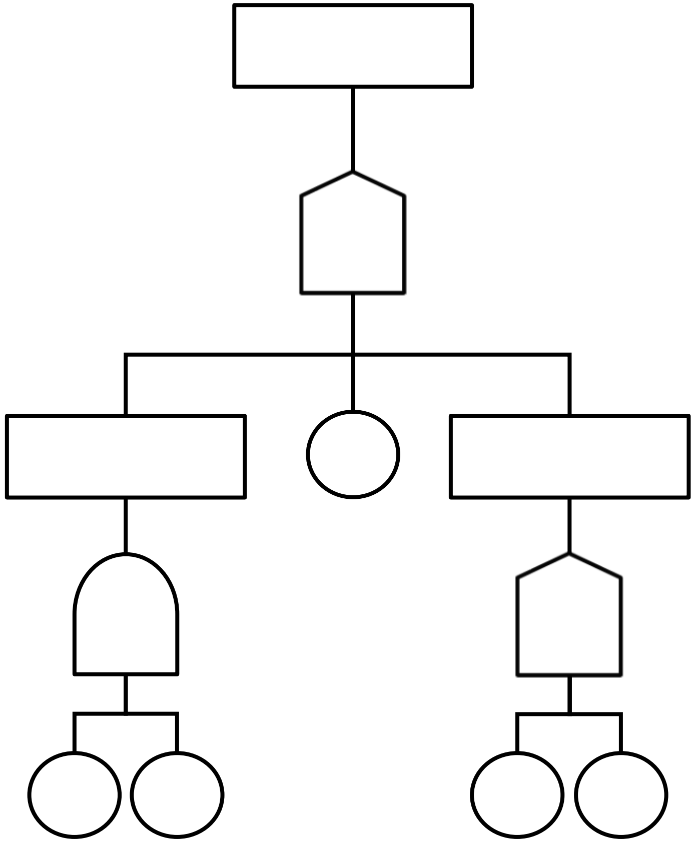
- Risk = Likelihood · Impact

- Methods for risk analysis
  - Failure mode & effects analysis (FMEA)
  - Hazard analysis
  - Why-because analysis
  - Fault tree analysis (FTA) ⟵ We focus on FTA

Fault tree analysis (FTA)

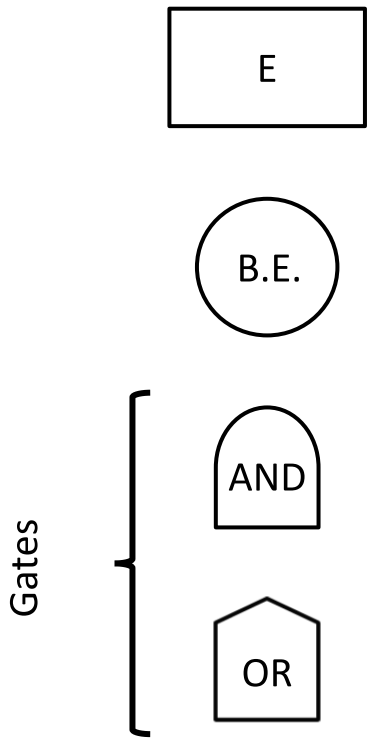
- Top-down diagram of the relationship between a system failure (= requirement violation) and its potential causes
- Can be used to identify sequences of events that result in a failure
- Allows prioritization of contributors leading to failures
- Informs decisions about how to design the system to be fault tolerant and/or fail safely
- Allows investigation of accidents and the identification of the root causes
- Often used for safety and reliability analysis, but can be used for other requirements as well

Fault tree analysis & ML



- ML is increasingly used in safety critical domains
  - Automotive, aeronautics, industrial control systems, …
- ML models are part of a larger system
- ML models will eventually make mistakes
  - Output wrong predictions/values
  - Fail to adapt to the changing environment
  - Confuse users
  - …
- Fault trees can help to understand how mistakes by ML contribute to system failures
  - Prerequesite to ensure that mistakes do not result in catastrophic outcomes

Basic building blocks of fault trees



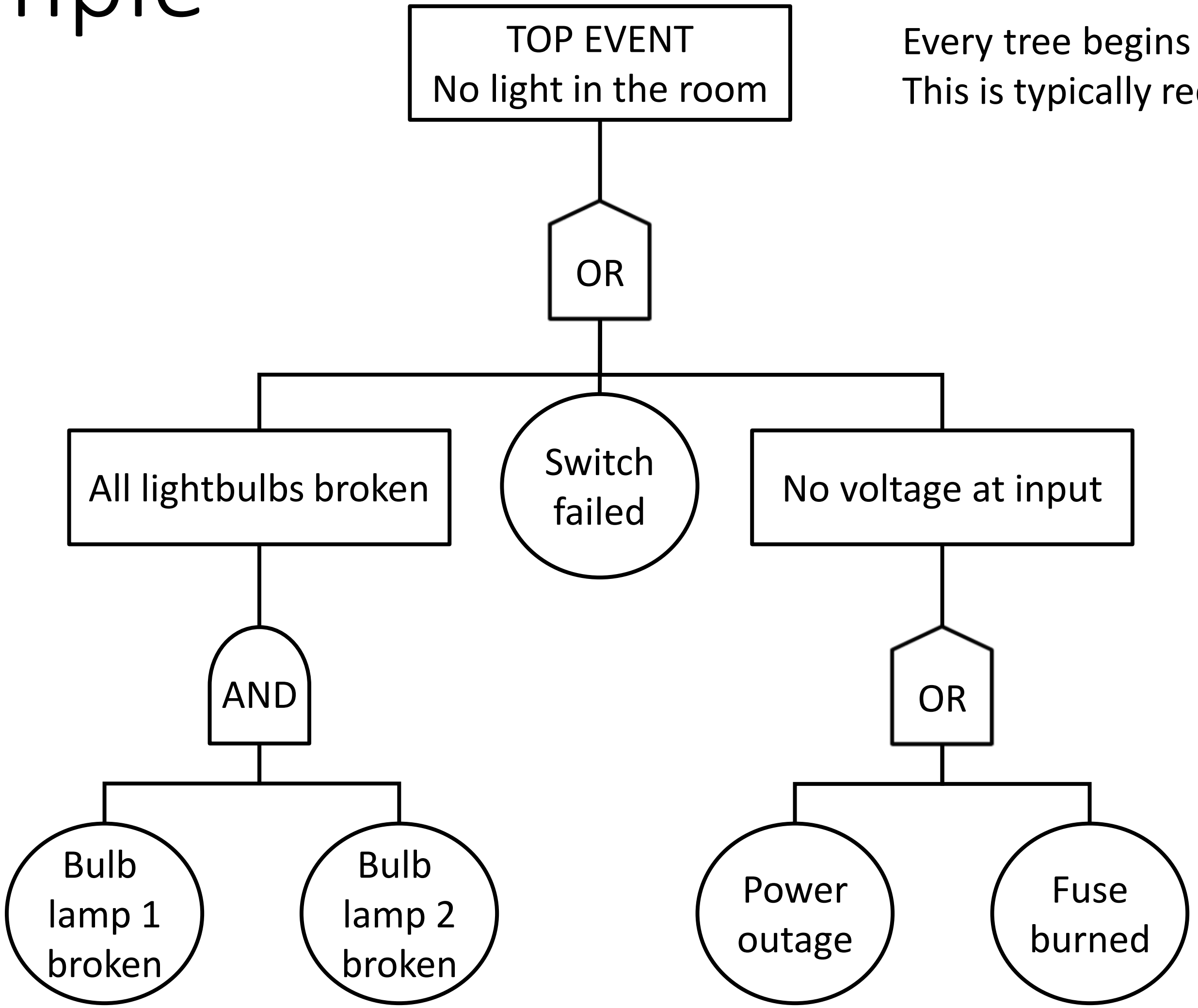
Event: an occurrence of a fault or an undesirable action

Basic event: no further development or breakdown (leaves of the tree)

AND: all of the sub-events must take place

OR: any of the sub-events may result in the parent event

Fault tree example



Every tree begins with a top event
This is typically requirement violation

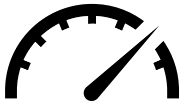
Every branch must terminate in a basic event

Figure from *Fault Tree Analysis and Reliability Block Diagram* (2016), Jaroslav Menčík.

Anaysis of fault trees

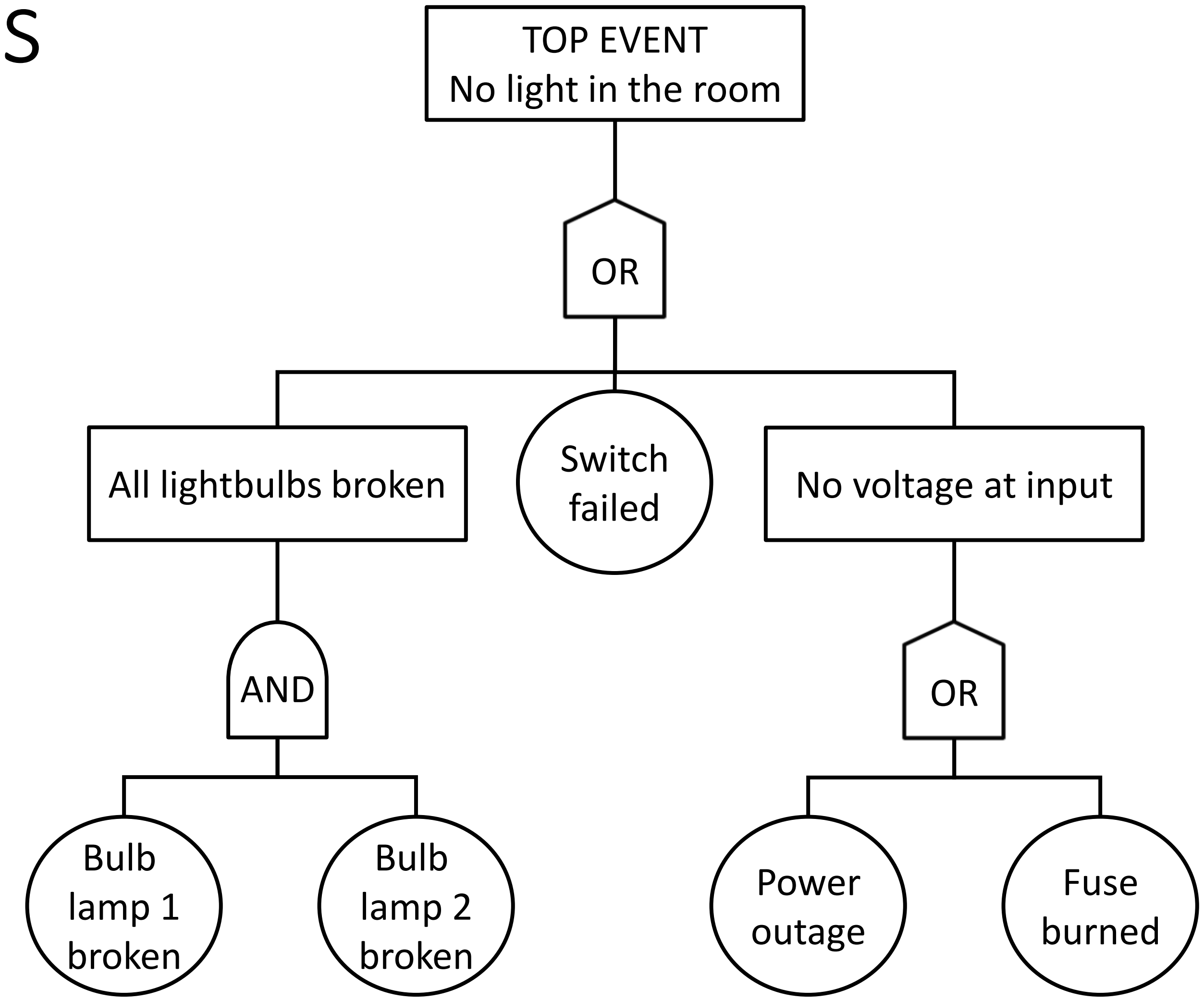
- Qualitative analysis
  - Determine potential root causes of failure through *minimal cut analysis*

- Quantitative analysis
  - Compute probability of failure



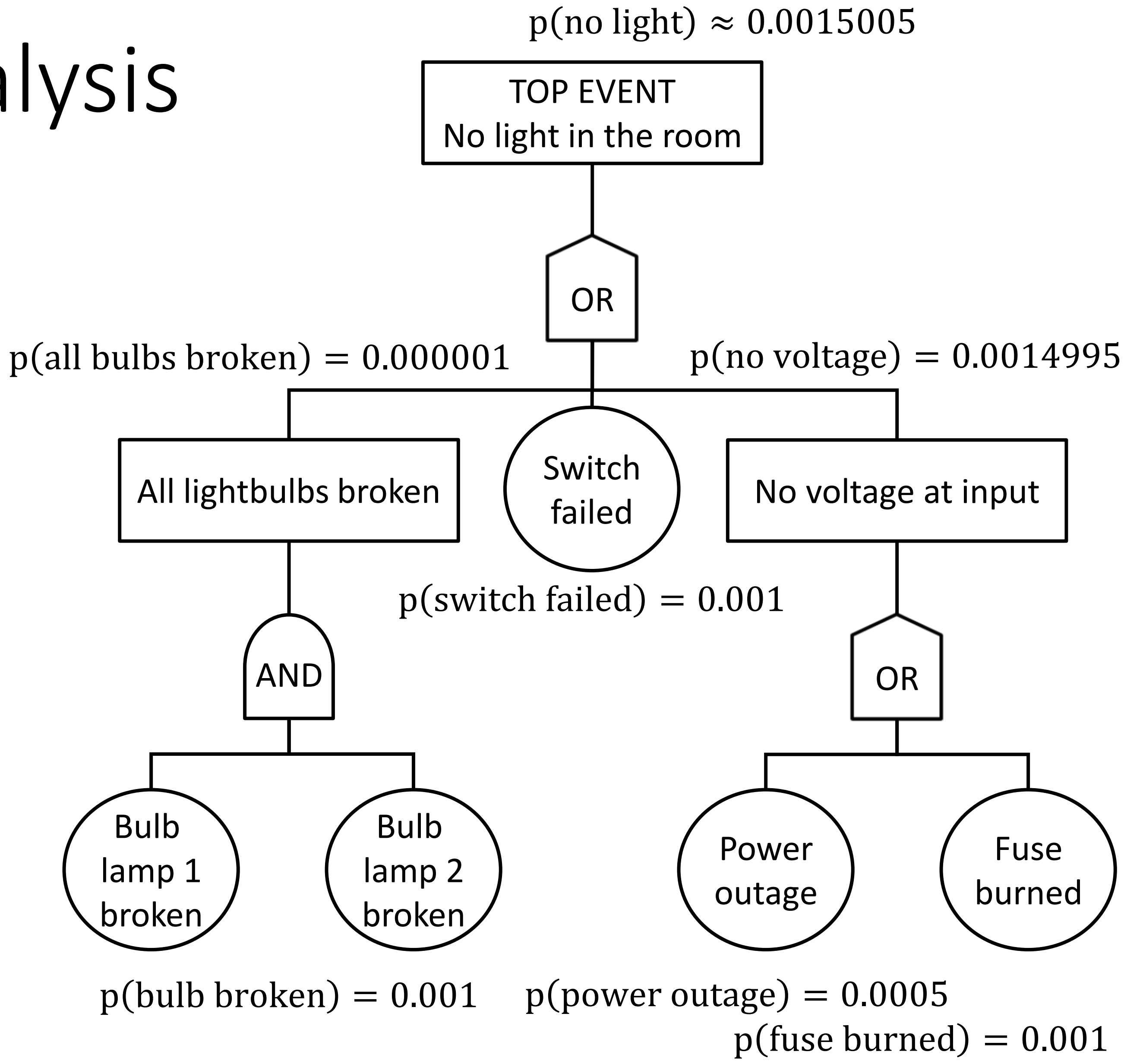
Minimal cut set analysis

- Cut set
  - A set of basic events whose simultaneous occurence is sufficient to guarantee that the top event occurs
- Minimal cut set
  - A cut set from which a smaller cut set cannot be obtained by removing basic events
- Minimal cut sets for the example
  - Bulb lamp 1 broken, bulb lamp 2 broken
  - Switch failed
  - Power outage
  - Fuse burned



Failure probability analysis

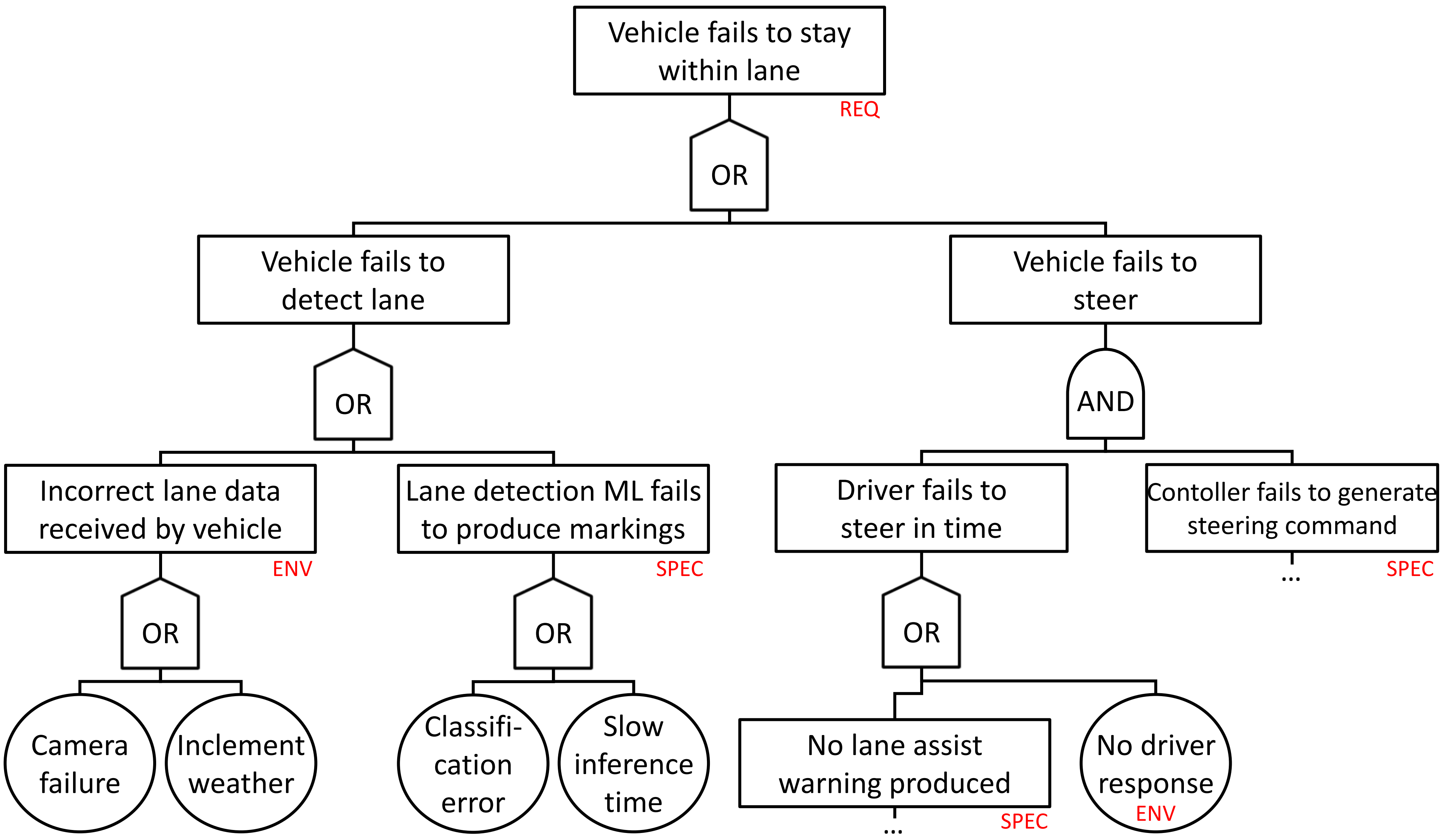
- Goal: compute the probability of the top event
- Assign probabilities to basic event
  - Based on domain knowledge
  - If possible, use measurements
- Apply probability theory to compute probalities of intermediate events through AND and OR gates
- Simplified but less accurate approach:
  - Compute probability as the sum as the probabilities of the minimal cut sets



Fault tree analysis process

- Specify the system structure
  - Environment entities and machine components
  - Assumptions (ENV) and specifications (SPEC)
- Identify the top even as a requirements violation (REQ)
- Construct the fault tree
  - Derive intermediate events from a violation of ENV or SPEC
  - Decompose intermediate events further down based on the knowledge of the domain or components
- Analyze the tree
  - Identify all possible minimal cut sets
- Consider design modifications
  - Try to elimate cut sets or
    - Elimates risk factor
  - Increase the size of cut sets
    - Makes the event less likely
- Repeat (possibly with new requirement)

Example: FTA for Lane assistant



FTA caveats

- In general, building a complete fault tree is impossible
  - There are probably some faulty events missed (u*nknown unknowns*)

- Domain knowledge is crucial for improving coverage
  - Talk to domain experts
  - Augment the tree as you learn more

- FTA is still very valuable for risk reduction!
  - Forces you to think about and explicitly document possible failure scenarios
  - A good starting point for designing mitigations

Strategies for handling faults in ML-based systems

Elements of fault-tolerant design

- Assume that …
  - software/ML components will make mistakes at some point
  - the environment evolves, violating some of its assumptions

- Goal: minimize the impact of mistakes/violations on the overall system

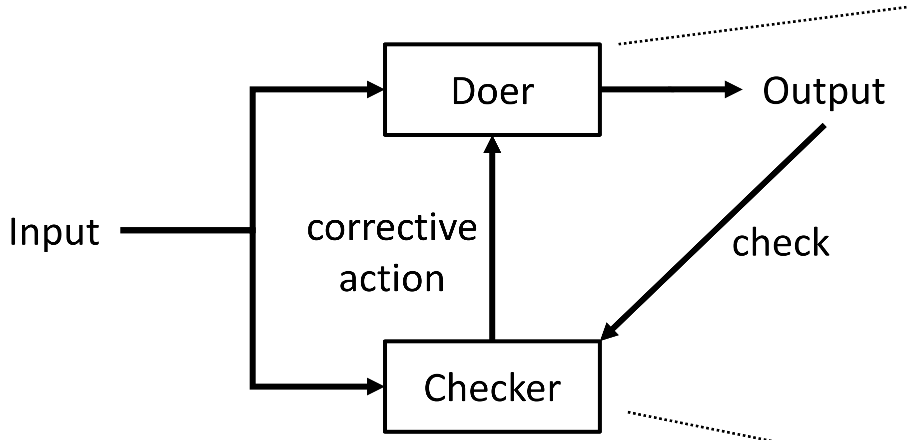
- Approaches:
  - Detection
  - Redundancy
  - Response
  - Containment

Detection: Monitoring

- Goal: detect when a component failure occurs
- Monitor: periodically checks the output of a component for errors
- Challenge: recognizing errors
  - E.g., corrupt sensor data, slow or missing responses, low ML confidence, …

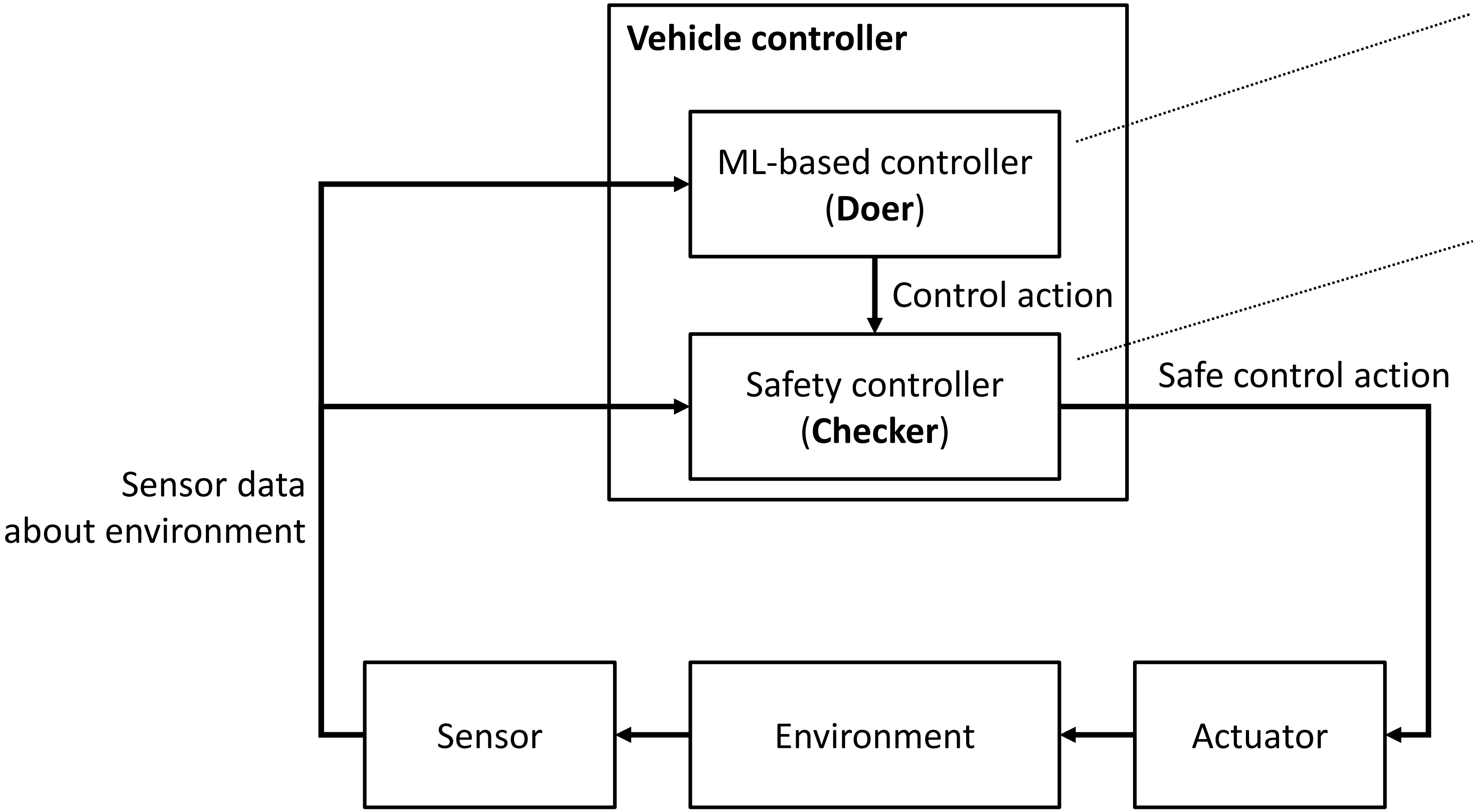
- Doer-Checker pattern

Perform primary functions
Untrusted and potentially faulty



If doer output is faulty, perform a corrective action.
Should be trustworthy!

Doer-Checker for an autonomous vehicle

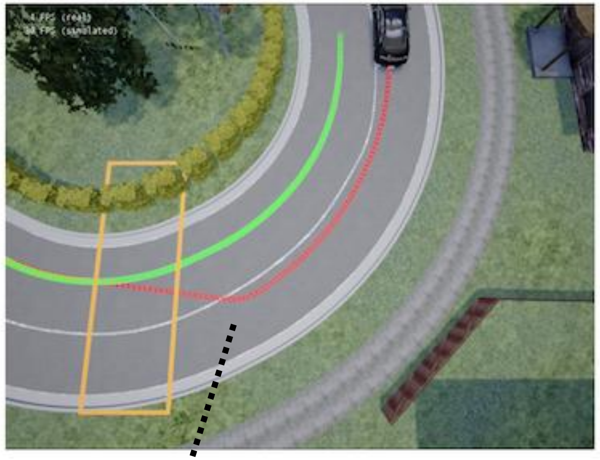


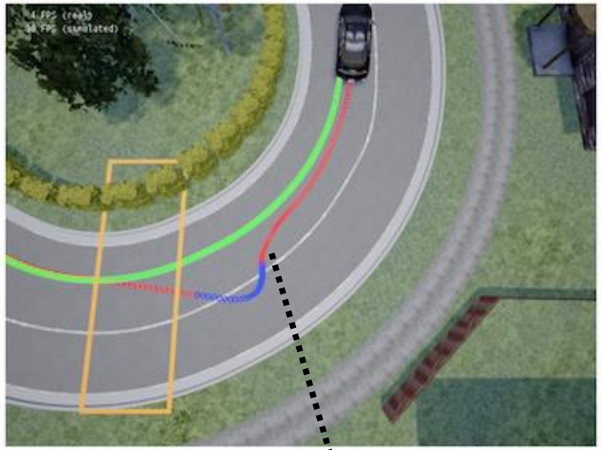
Generate commands to steer vehicle
E.g., complex neural network

Checks commands from ML controller.
Overrides commands with a safe default if the action is deemed risky
Simpler, based on transparent and verifiable logic. Conservative behavior.

The example in action

Green: optimal behavior
Red: generated by ML controller
Blue: correction by safety controller



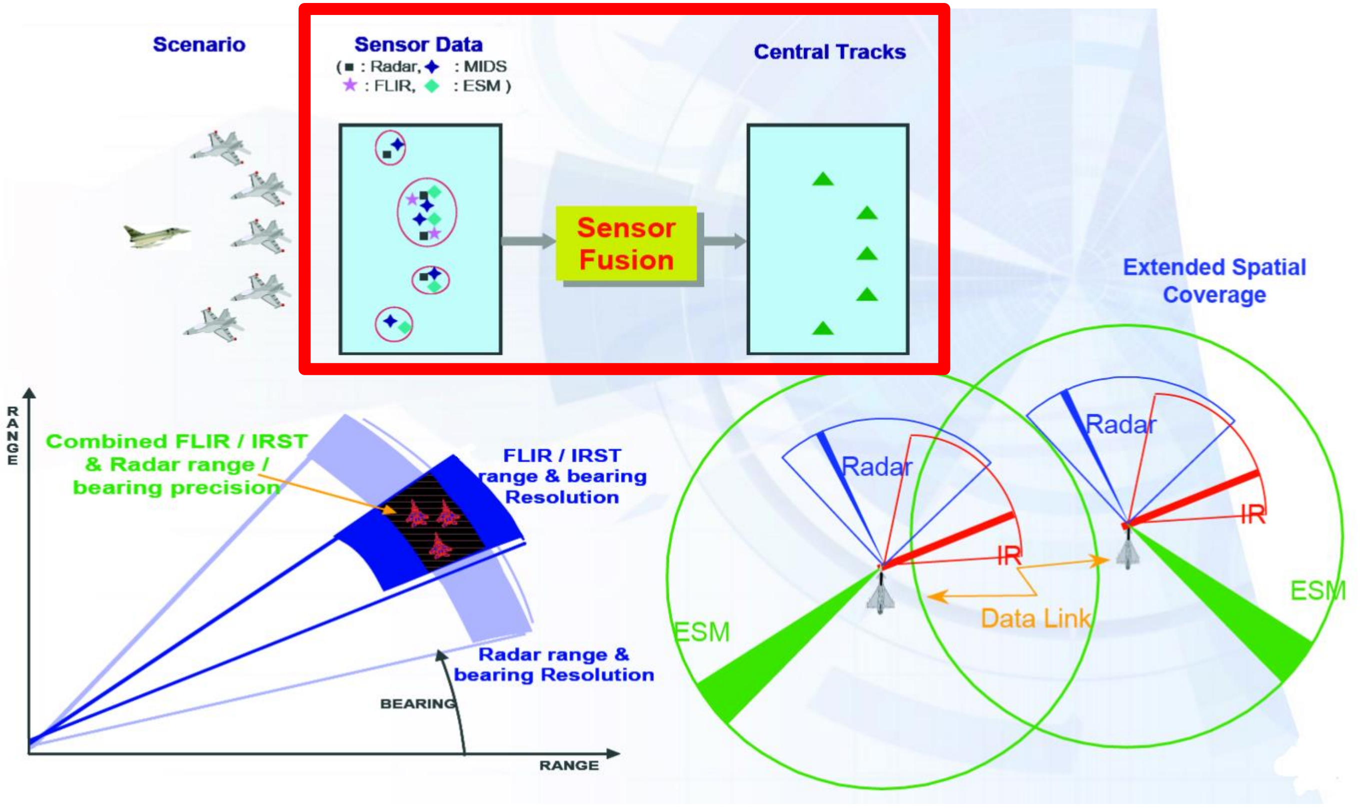


Safe actions directly generated by ML controller.
No action by safety controller required.

Unsafe steering in wrong direction by ML controller.
Safety controller overrides commands to achieve safe trajectory

Weichao Zhou, Ruihan Gao, BaekGyu Kim, Eunsuk Kang, Wenchao Li. Runtime-Safety-Guided Policy Repair, Intl. Conference on Runtime Verification (2020)

Redundancy: Sensor data fusion



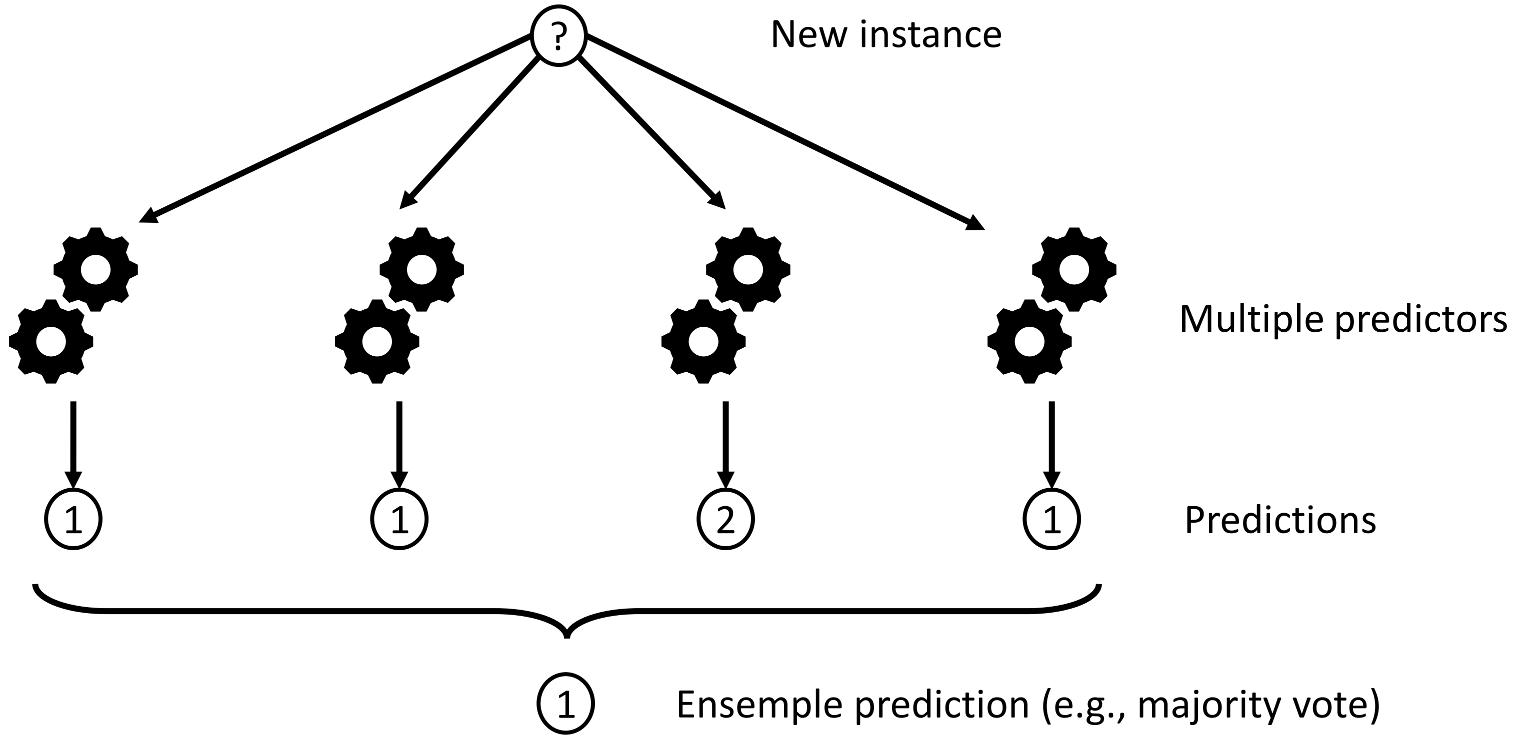
https://en.wikipedia.org/wiki/Sensor_fusion#/media/File:Eurofighter_sensor_fusion.png

Combine data from a wide range of sensors

Provides partial information even when (some) sensors are faulty

Common approach for all modern tracking technology

Redundancy: Ensemble learning



Only fails if multiple predictors are failing.

Response: Graceful degradation (fail-safe)

- Goal: when a component failure occurs, achieve system safety by reducing functionality and performance
- Relies on a monitor to detect failures!
- Example: computer vision failures
  - Failure: one lidar sensor fails, making position estimates less reliable
  - Response: switch to lower-quality detection and give more conservative estimations of positions and required minimal distances.

Response: Human in the loop

- Use less forceful interaction, make suggestions, or ask for confirmation

- AI for prediction, human for judgement
  - AI good at statistics at scale with many factors
  - Human good at understanding context and data generation process

- Several risks involved
  - Notification fatigue
  - Complacency and just following predictions

- Lots of UI design and Human-Computer-Interaction (HCI) problems

Response: Undoable action

- Design the system to reduce the consequences of wrong predictions, allowing humans to override or undo

- Similar to human in the loop
  - No asking for confirmation
  - Instead allowing to undo already performed actions

Containment: Decoupling and isolation

- Faults in low-critical (LC) components should not impact high-critical (HC) components
  - A broken car radio software should not interfere with the engine!

- Apply principle of least privilege
  - LC components should be allowed access to the minimum of necessary functions
  - Limits interaction across critical boundaries

- Build barriers between LC and HC components
  - E.g., deploy in different networks
  - Add monitors/checks at interfaces

- Determine if ML component is performing LC or HC tasks
  - If HC: check if this can be degraded into LC tasks or be replaced with non-ML components

Experimental Security Analysis of a Modern Automobile, Koscher et al., IEEE Symposium on Security and Privacy, 2010

Example: Lane keeping assistance system with lane departure warning

Live exercise: How could we mitigate the risk?

… monitoring?

… redundancy?

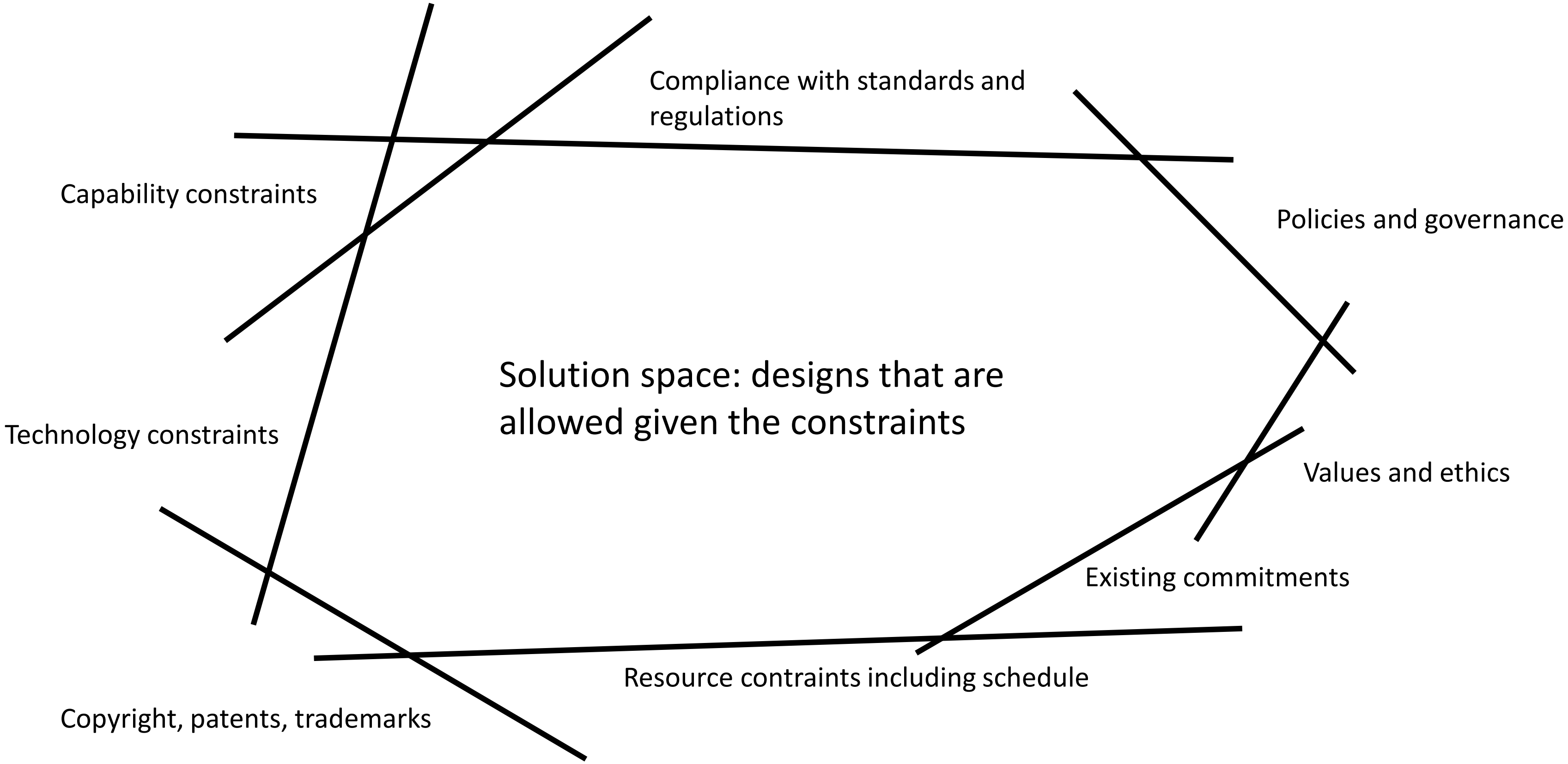
… response?

… containment?

Constraints and trade-offs

Constraints

- Define the space of attributes for valid design solutions



Types of Constraints

- Problem constraints
  - Minimal required quality assurance for an acceptable product
  - E.g., to guarantee safety or fairness

- Project constraints
  - Deadlines
  - Project budget
  - Team skills

- Design constraints
  - Type of ML task (e.g., clustering, classification, regression)
  - Available data
  - Available resources for training
  - Available resources for inference

Trade-offs between ML models

- Assume multiple ML methods satisfy the constraints: which one should be used?

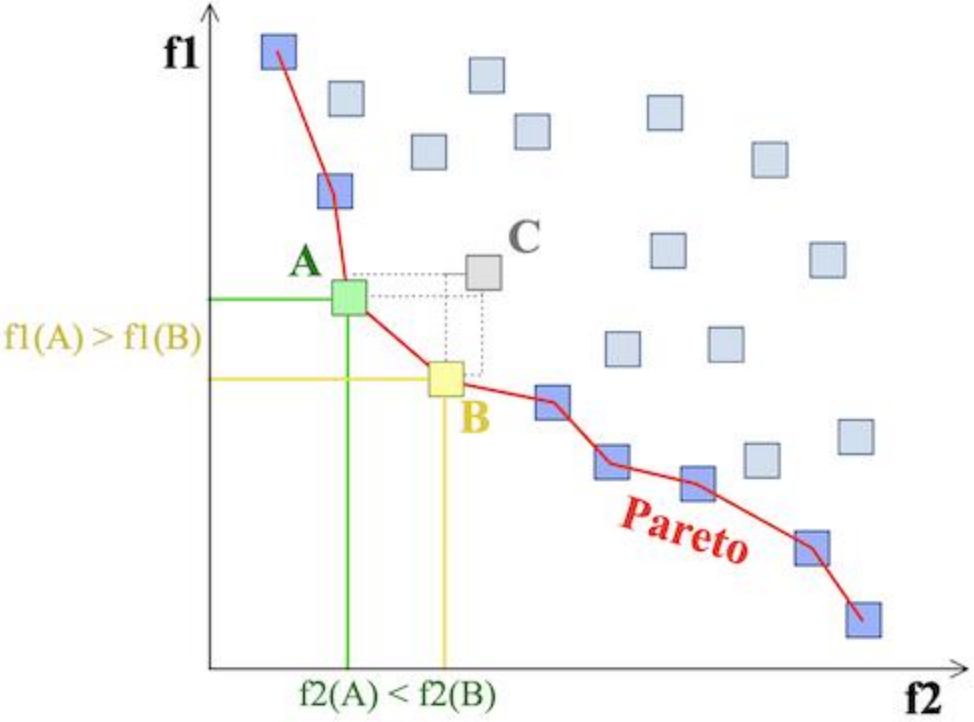
- Different ML qualities may be in conflict with each other
  - E.g., accuracy vs. interpretability

- Requires trade-offs between these qualities

- Decision between models requires
  - Understanding the impact of different ML models on the trade-offs
  - Understanding the importance of the different qualities: which one(s) do you care about most?

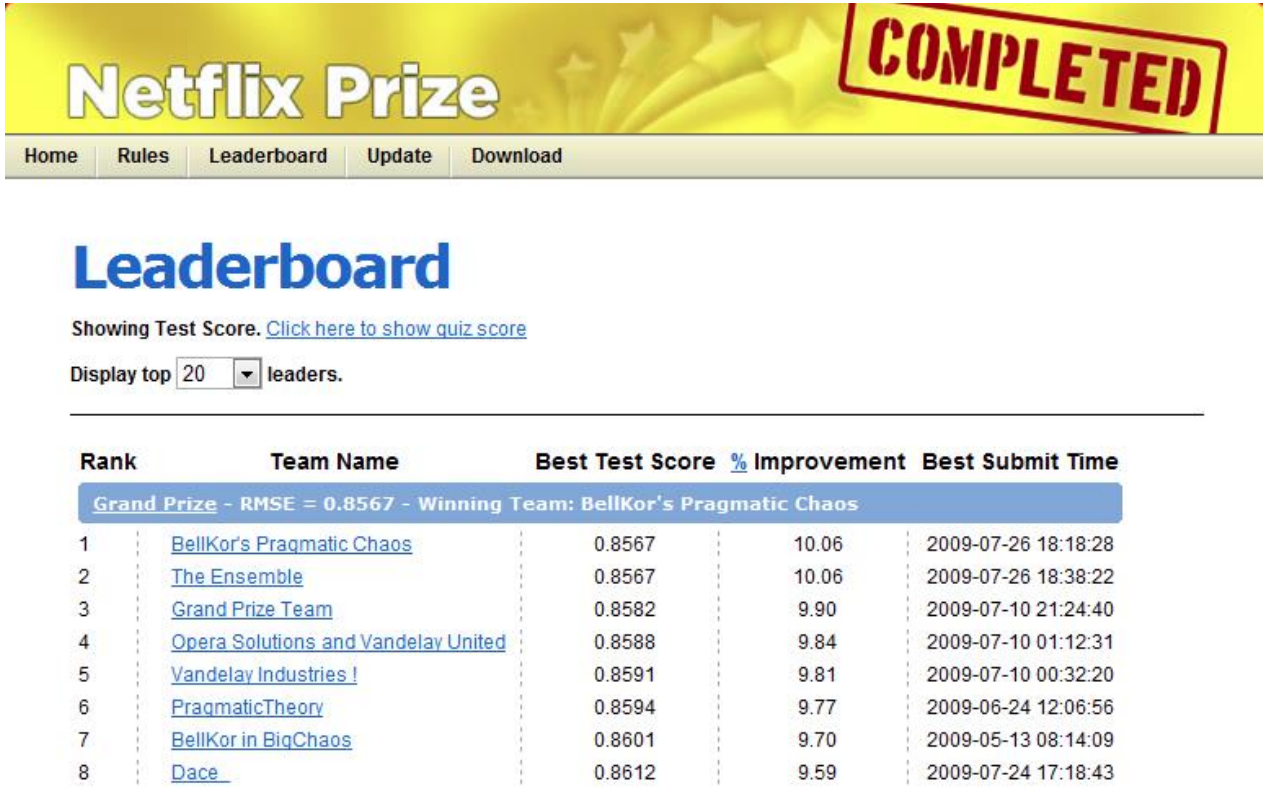
Pareto Fronts

- Multi-objective optimization:
  - Let $x \in X$ be a possible solution from a solution space X
  - Let $f_1, f_2, \dots, f_n$ be objectives that should be optimized (in our case min.)
- x is dominated, iff $\exists x' \in X \; s.t. \; \forall \, i = 1, \dots, n: f_i(x) > f_i(x')$
- x is Pareto optimal, iff $\nexists x' \in X \; s.t. \; \forall \, i = 1, \dots, n: f_i(x) > f_i(x')$
- The set of Pareto optimal solutions defines the Pareto front
- Each solution on the Pareto front is optimal for a different trade-off between the objectives



https://en.wikipedia.org/wiki/Multi-objective_optimization#/media/File:Front_pareto.svg

Examples for trade-offs: Cost vs. accuracy


Netflix Prize

COMPLETED

Home | Rules | Leaderboard | Update | Download

Leaderboard

Showing Test Score. Click here to show quiz score

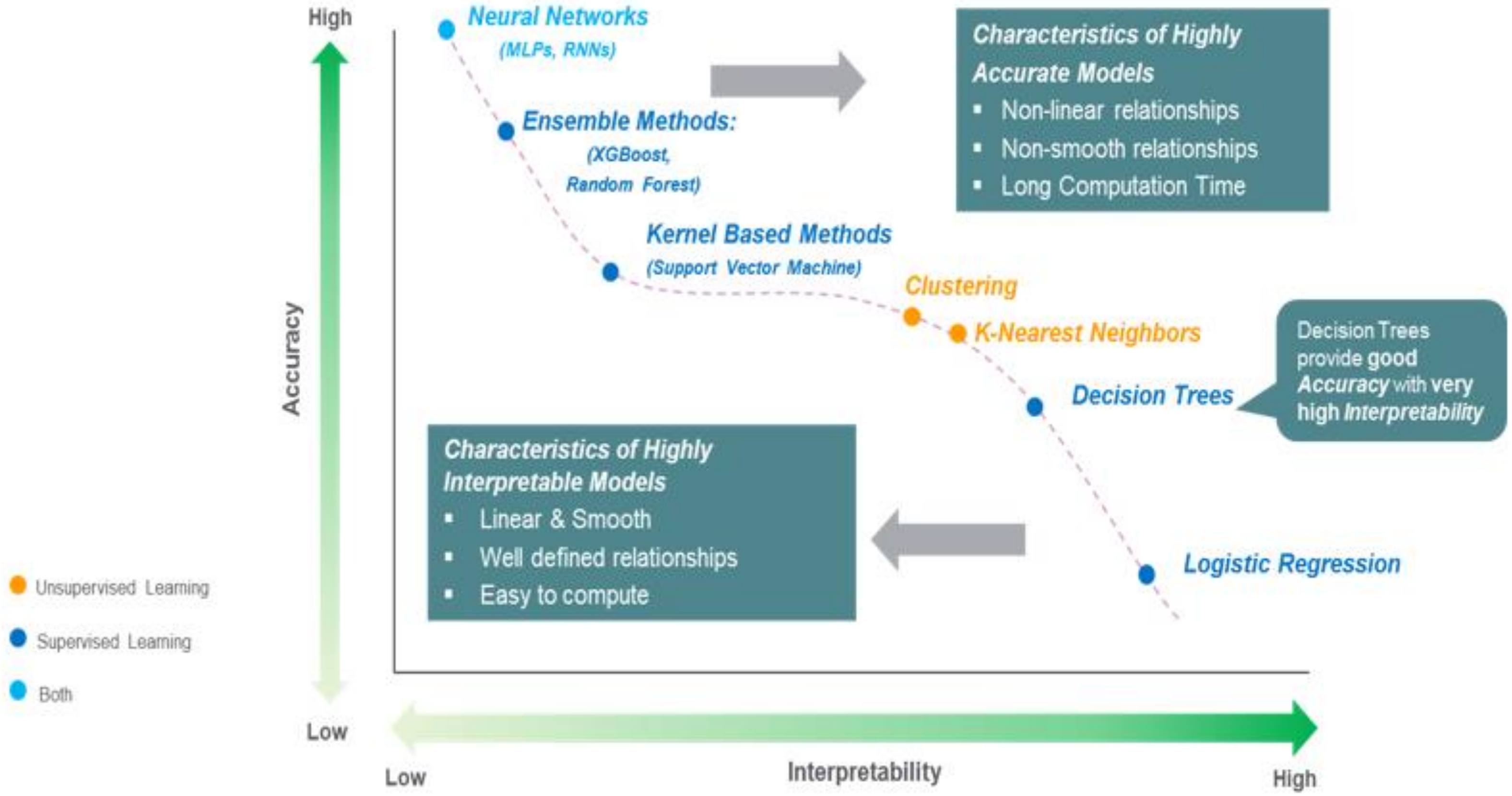
Display top 20 leaders.

Rank	Team Name	Best Test Score	% Improvement	Best Submit Time
Grand Prize - RMSE = 0.8567 - Winning Team: BellKor's Pragmatic Chaos				
1	BellKor's Pragmatic Chaos	0.8567	10.06	2009-07-26 18:18:28
2	The Ensemble	0.8567	10.06	2009-07-26 18:38:22
3	Grand Prize Team	0.8582	9.90	2009-07-10 21:24:40
4	Opera Solutions and Vandelay United	0.8588	9.84	2009-07-10 01:12:31
5	Vandelay Industries !	0.8591	9.81	2009-07-10 00:32:20
6	PragmaticTheory	0.8594	9.77	2009-06-24 12:06:56
7	BellKor in BigChaos	0.8601	9.70	2009-05-13 08:14:09
8	Dace	0.8612	9.59	2009-07-24 17:18:43

"We evaluated some of the new methods offline but the additional accuracy gains that we measured did not seem to justify the engineering effort needed to bring them into a production environment."

https://netflixtechblog.com/netflix-recommendations-beyond-the-5-stars-part-1-55838468f429

Examples for trade-offs: Accuracy vs. interpretability



https://towardsdatascience.com/the-balance-accuracy-vs-interpretability-1b3861408062

More trade-offs

- Accuracy vs. model size (memory footprint)
- Accuracy vs. inference time
- Dedicated hardware vs. standard hardware
- Fairness vs. costs
- Online vs. offline model
- Model update vs. accuracy loss
- …

Finding the right qualities for a product

- Interview stakeholders (customers, operators, developers, business experts, …)
  - Understand the problem, the kind of prediction needed
  - Understand the scope (target domain, frequency of change, …)

- Broadly understand quality needs from different views
  - Model view: direct expectation on the model
  - Data view: availability, quantity, and quality of the data
  - System view: system goals and the role of the ML model and interactions with the environment
  - Infrastructure view: training costs, reproducibility needs, serving infrastructure needs, monitoring needs, …
  - Environment/user view: external expectations on the system by users and society (fairness, safety, privacy, …)

- Collect and document needs, resolve conflicts, discuss and prioritize

Siebert, Julien, Lisa Joeckel, Jens Heidrich, Koji Nakamichi, Kyoko Ohashi, Isao Namba, Rieko Yamamoto, and Mikio Aoyama. "Towards Guidelines for Assessing Qualities of Machine Learning Systems." In International Conference on the Quality of Information and Communications Technology, pp. 17-31. Springer, Cham, 2020.

Formulating goals

- Set minimum accuracy expectations (functional requirement)
- Identify runtime needs
  - Number of predictions, latency requirements, cost budget, local vs. cloud deployment
- Identify evolution needs
  - Update and retraining frequency, expected concept drift, …
- Identify explainability needs
- Identify protected characteristics and possible fairness concerns
- Identify security and privacy requirements (both ethical and legal)
- Understand data availability and need
  - Quality, quantity, diversity, formats, provenance

Map to system goals

Vogelsang, Andreas, and Markus Borg. "Requirements Engineering for Machine Learning: Perspectives from Data Scientists." In Proc. of the 6th International Workshop on Artificial Intelligence for Requirements Engineering (AIRE), 2019.

Questions?