17-423/723: Designing Large-scale Software Systems

Design for Robustness

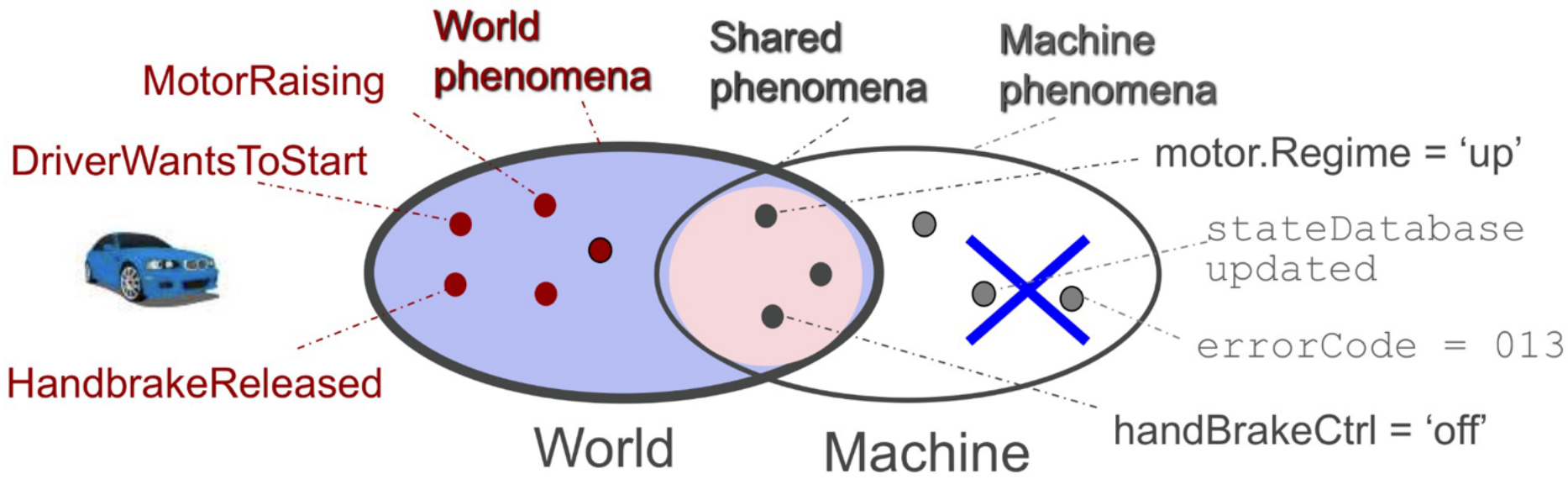
Mar 25, 2024

Leaning Goals

- Understand different ways in which a system may fail to meet its requirements and quality attributes
- Specify robustness as a quality attribute of a system
- Describe the differences between robustness, fault-tolerance, resilience, and reliability
- Apply fault tree analysis to identify possible root cause of a system failure
- Apply HAZOP to identify possible component failures and their impact on the system

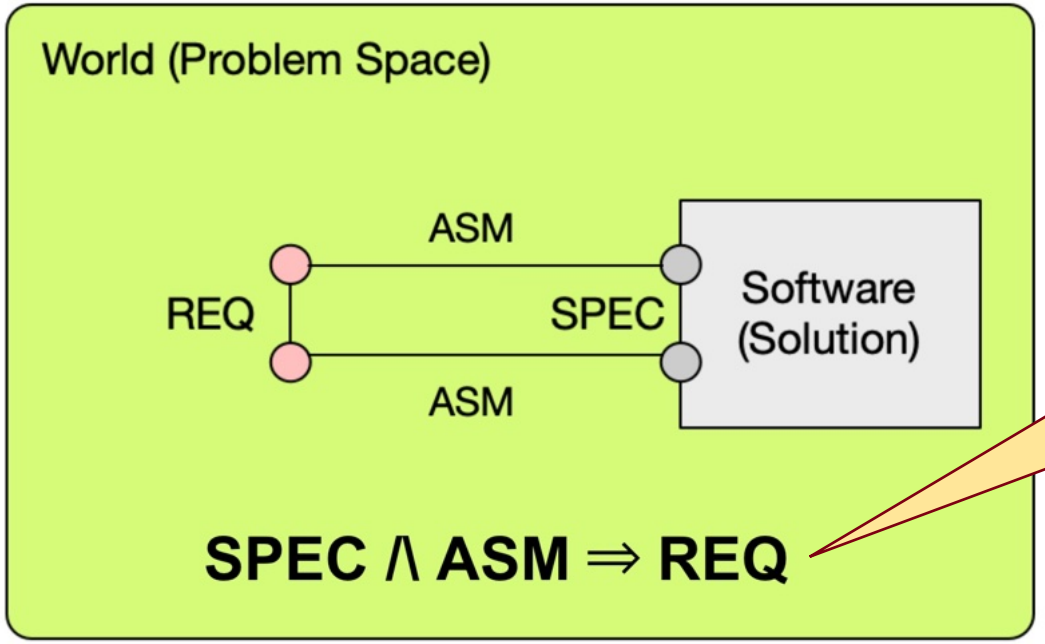
What can possibly go wrong with my system?

Recall: World vs. Machine



- **Shared phenomena**: Interface between the world & software
- Software can influence the world **only through the shared interface**
- Beyond this interface, we can only **assume** how the entities in the world will behave

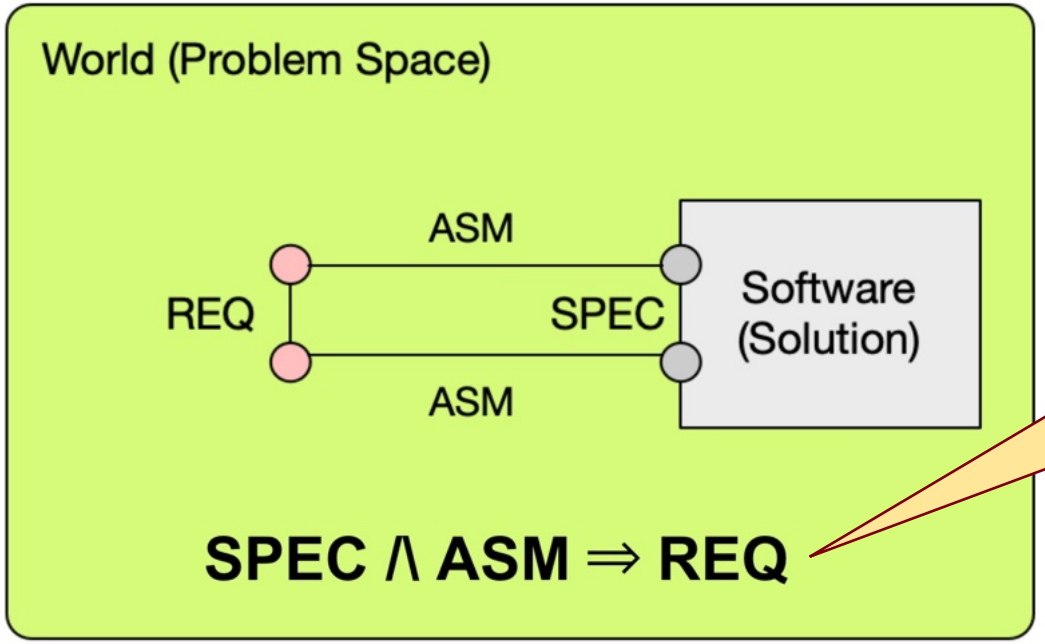
Recall: Satisfaction Argument



"If my software is implemented correctly (SPEC) and the world behaves as assumed (ASM), then the system should fulfill its requirement (REQ)"

- **Requirement (REQ):** What the system must achieve, in terms of desired effects on the world
- **Specification (SPEC)**: What software must implement, expressed over the shared interface
- **Domain assumptions (ASM)**: What's assumed about the world; bridge the gap between REQ and SPEC

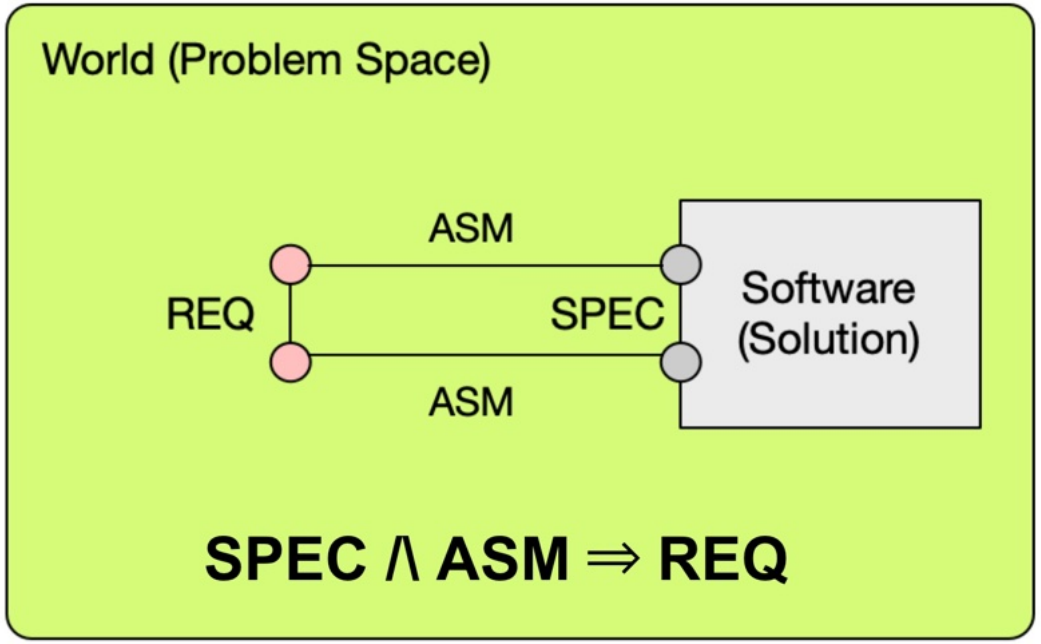
What can go wrong in my system?



"If my software is implemented correctly (SPEC) and the world behaves as assumed (ASM), then the system should fulfill its requirement (REQ)"

- **Q. What are some ways in which the system may fail to satisfy this argument?**

What can go wrong in my system?



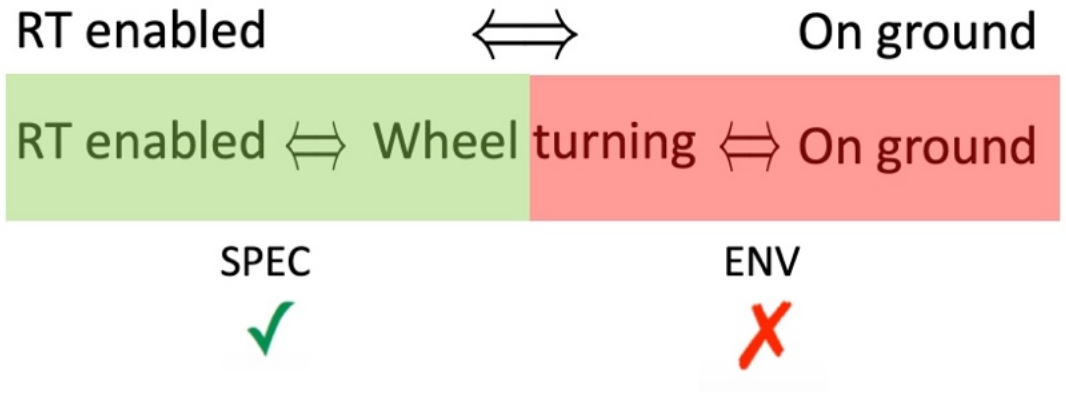
- Missing or incorrect specifications (SPEC)
- Violated specifications, due to bugs or faults in software (SPEC)
- Missing or incorrect assumptions (ASM)
- Missing or incorrect requirements (REQ)

Example: Lane Keeping Assist

Q. What can go wrong?

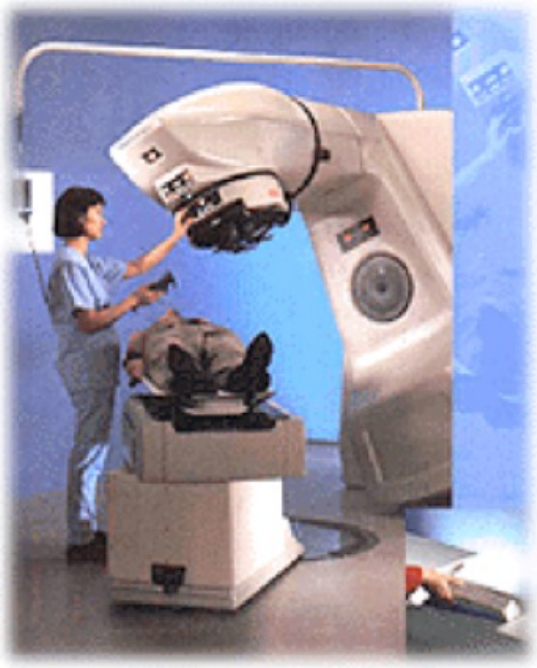
- **Requirement (REQ):** The vehicle must be prevented from veering off the lane.
- **Assumptions (ENV):** Sensors are providing accurate information about the lane; driver responses on time when given a warning; steering wheel is functional
- **Specifications (SPEC):** Lane detection accurately identifies the lane markings; controller generates correct steering commands to keep the vehicle within lane

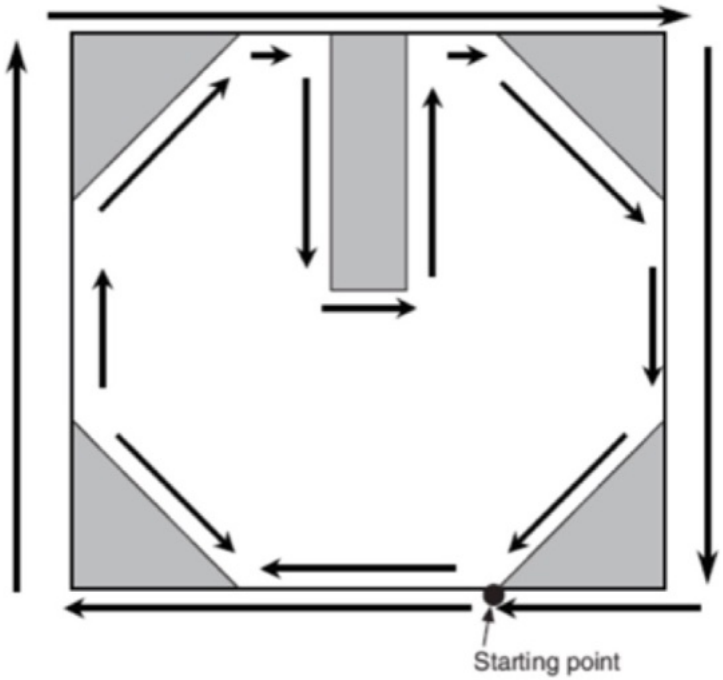
Recall: Lufthansa 2904 Runway Crash (1993)



- **Reverse thrust (RT):** Decelerates plane during landing
- What was required (REQ):
  RT is enabled if and only if plane is on the ground
- What was implemented (SPEC):
  RT is enabled if and only if wheel turning
- What was assumed (ENV):
  Wheel is turning if and only if it's on ground
- But runway was wet due to rain
  - Wheel failed to turn even when on ground
  - **Assumption (ENV) was incorrect!**
  - Pilot attempted to enable RT, but it was overridden by the software
  - Plane went off the runway and crashed

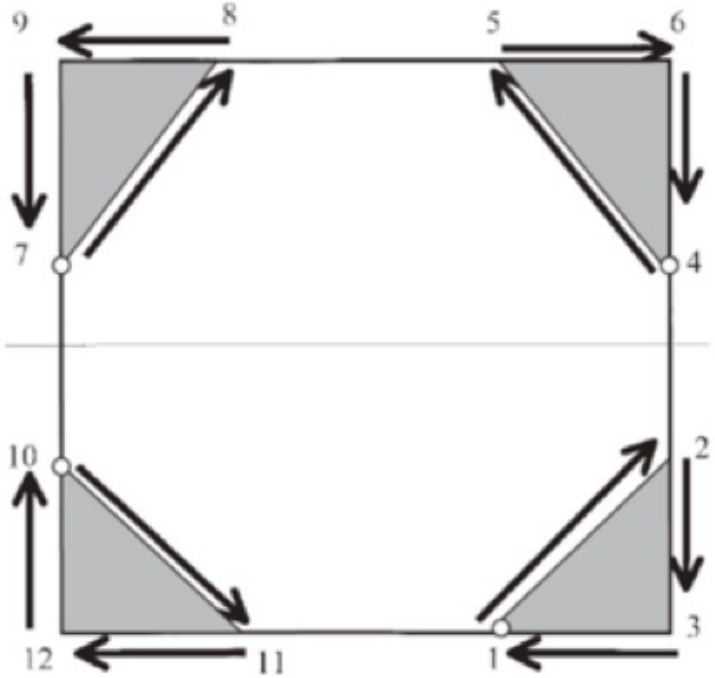
Example: Panama City Hospital (2000)





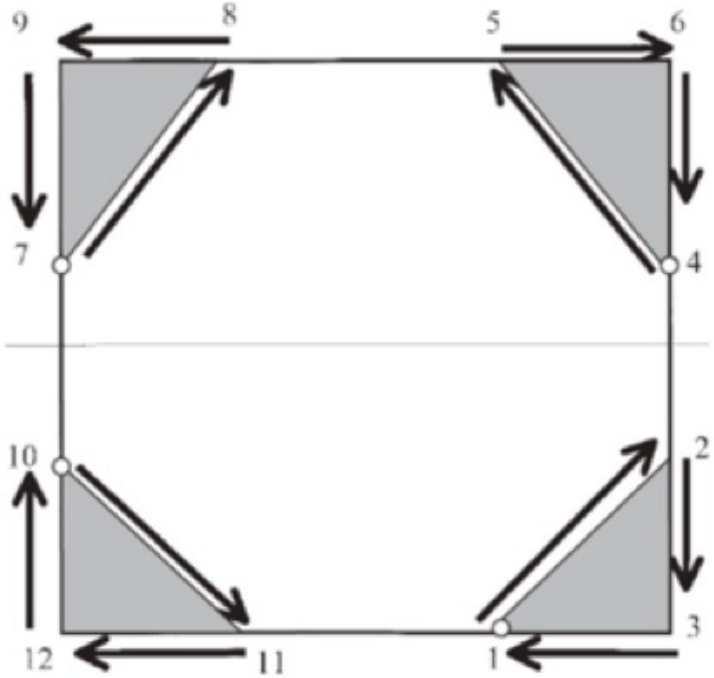
- Therapy planning software by Multidata Systems
- Theratron-780 by Theratronics (maker of Therac-25)
- **Shielding blocks**: Inserted into beam path to protect healthy tissue
- Therapist draws block shapes; software computes amount of radiation dose

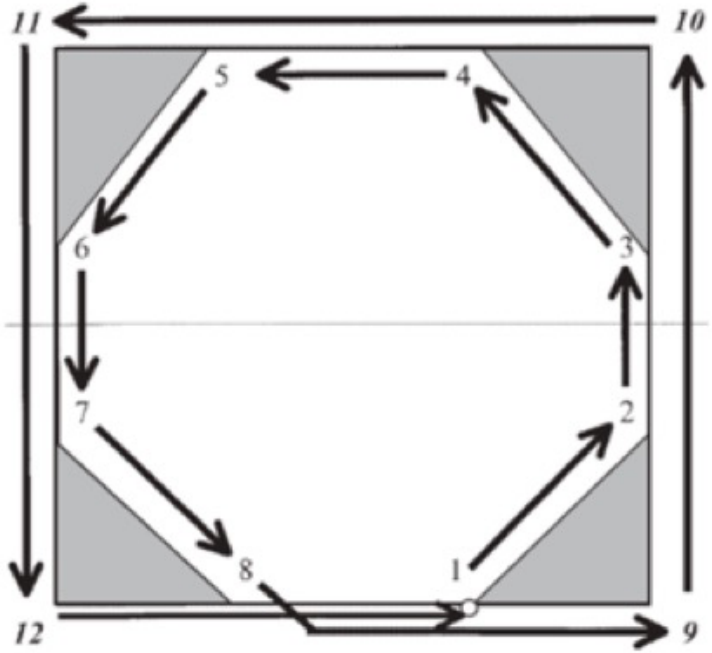
Example: Panama City Hospital



dose = D

Example: Panama City Hospital





dose = D

dose = $2D$

21 patients injured; 8 deaths

Blame the user or software?

- Lawsuits against the software company and hospital staff
- **Multidata Systems:**
  “Given [the input] that was given, our system calculated the correct amount, the correct dose. And, if [the staff in Panama] had checked, they would have found an unexpected result.”
- Three therapists charged & found guilty for involuntary manslaughter; barred from practice for several years

Being robust against possible failures

- No system will ever be “correct”
- The environment will often behave in unexpected ways, violating assumptions (ASM)
- Software will have bugs and the underlying hardware will sometimes fail; specifications (SPEC) will be violated
- Even when these abnormal events occur, we want our systems to behave in an acceptable manner
  - Even if a user makes a mistake, this should not lead to a safety disaster
  - An off-by-one error should not lead to an entire rocket crashing
  - Even if some of the servers shutdown, the system should continue to provide critical services
- How do we design systems to be robust against such failures?

Robustness

Robustness

- The ability of a system to provide an **acceptable level of service** even when it operates under **abnormal conditions**
- Acceptable level of service: Quality attribute (typically of high importance) to be preserved, such as:
  - Safety: “No unsafe level of radiation delivered to the patient”
  - Performance: “The 95th-tile response to client requests is at most 200ms”
  - Availability: “The patient record database is available 99% of the times”
- Abnormal conditions: An event or a condition that is outside of an expected, normal behavior, such as:
  - “The nurse deviates from the treatment instructions”
  - “The sensor provides an image with a significant amount of blur”
  - “The database is unresponsive and fails to store new appointments”

Robustness

- The ability of a system to provide an **acceptable level of service** even when it operates under **abnormal conditions**
- Acceptable level of service: Quality attribute (typically of high importance) to be preserved
- Abnormal conditions: An event or a condition that is outside of an expected, normal behavior
- **Q. Does this remind of you another quality attribute?**

Robustness

- The ability of a system to provide an **acceptable level of service** even when it operates under **abnormal conditions**
- Acceptable level of service: Functional requirement or quality attribute (typically of high importance) to be preserved
- Abnormal conditions: An event or a condition that is outside of an expected, normal behavior
- **Recall**: Scalability is the ability to handle growth in the amount of workload while maintaining an acceptable level of performance
  - Scalability can be thought of as one specific type of robustness!

Related Concepts

- Fault-tolerance: Ability of a system to provide acceptable service even when one or more of its components exhibit a faulty behavior
  - Typically about internal faults within a system
  - In this class, robustness covers both internal & external faults
- Resilience: Ability of a system to recover from an unexpected failure
  - Focus is on recovery instead of prevention
- Reliability: Ability of a system to provide acceptable level of service over a period of time
  - Typically measured as a “mean time between failures” (MTBF); e.g., 1 system failure over 1000 hours
  - Robustness is necessary to achieve reliability

Specifying Robustness: Good & Bad Examples

- The radiation therapy system should never deliver more than a maximum amount of radiation no matter what the nurse inputs ✅
- The autonomous vehicle must operate even under a severe weather ❌
- The scheduling app must process appointments even if the connection to the central database is lost ✅
- Amazon must provide provide a response time less than 100ms even when the amount of concurrent customers exceeds 2 million ✅
- The package delivery drone should never drop a package at a wrong location ❌
- The autonomous vehicle must avoid hitting a pedestrian even if an object detection model fails to recognize it ✅

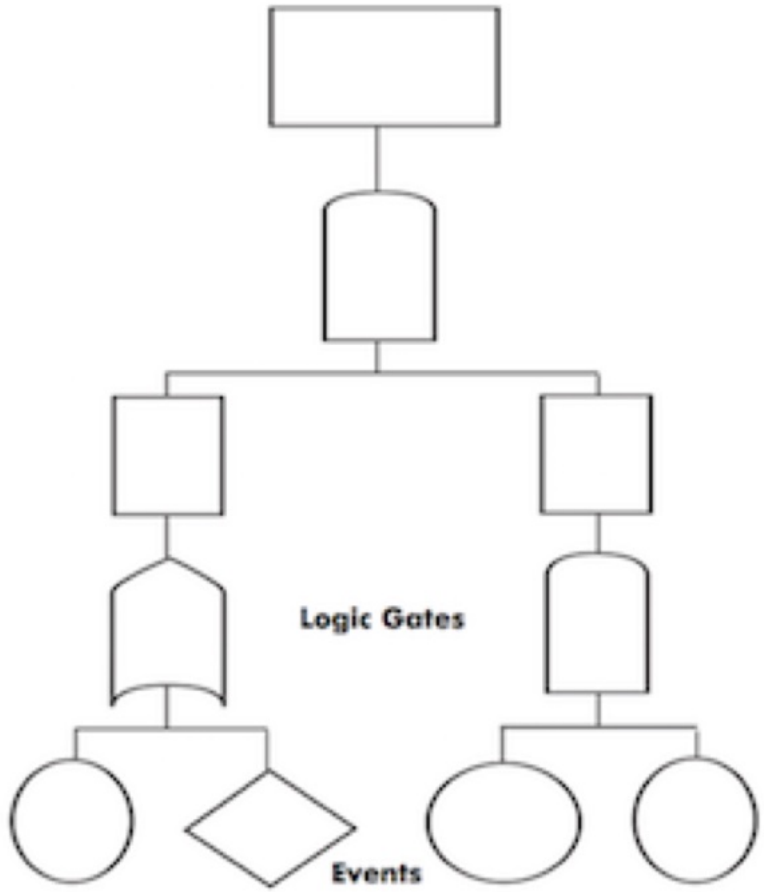
Failure Analysis

Failure Analysis

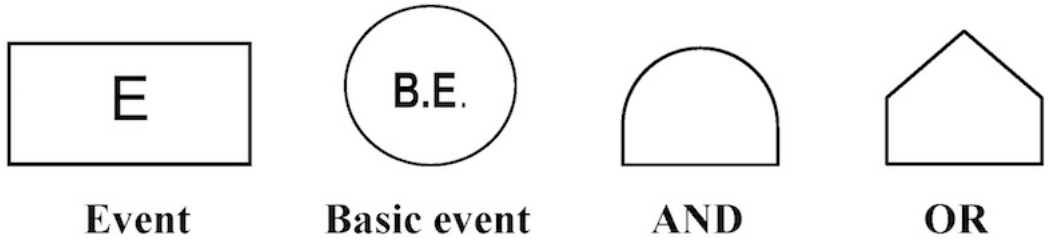
- *What can possibly go wrong in my system, and what is potential impact on system requirements?*
- Systematically analyze a design and identify different scenarios in which the system may fail to satisfy its requirements
- A number methods, developed and routinely applied in many engineering disciplines
  - **Fault tree analysis (FTA)**
  - **Hazard and operability study (HAZOP)**
  - Failure mode & effects analysis (FMEA)
  - Why-because analysis
  - ...

Fault-Tree Analysis (FTA)

- **Fault tree:** Specify relationships between a system failure (i.e., requirement violation) and its potential causes
  - Identify sequences of events that result in a failure
  - Prioritize the contributors leading to the failure
  - Inform decisions about how to (re-)design the system
  - Investigate an accident & identify the root cause
- Often used for safety & reliability, but can also be used for other types of QAs (e.g., poor performance, security attacks…)



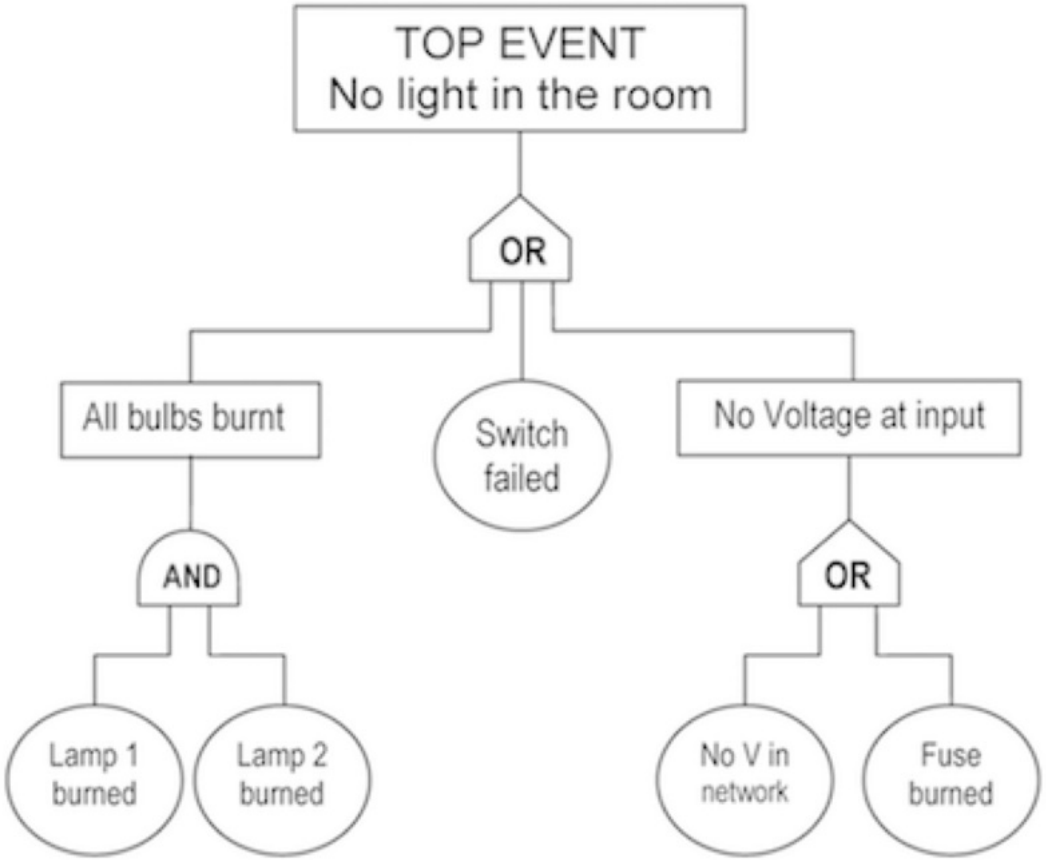
Elements of Fault Trees



- Event: A fault or an undesirable event
  - Non-basic event: An event that can be explained in terms of other events
  - Basic event: No further development or breakdown; leaf node in the tree
- Gate: Logical relationship between an event & its immediate subevents
  - AND: All of the sub-events must take place
  - OR: Any one of the sub-events may result in the parent event

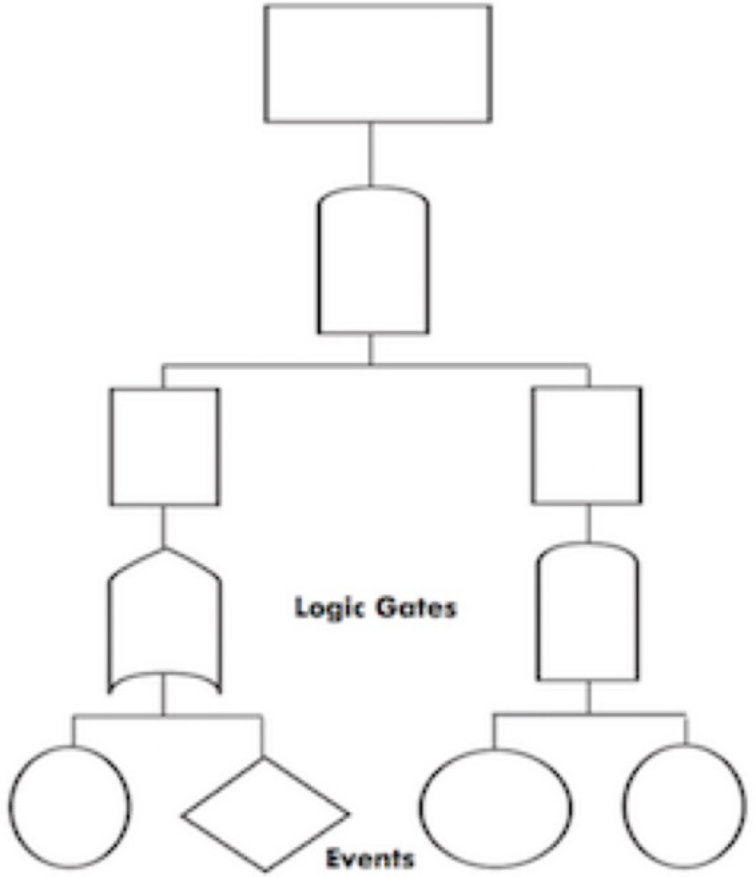
Elements of Fault Trees

- Every tree begins with a TOP event (typically a requirement violation or a hazardous event)
- Every non-basic event is broken into a set of child events and connected through an AND or OR gate
- Every branch of the tree must terminate with a basic event

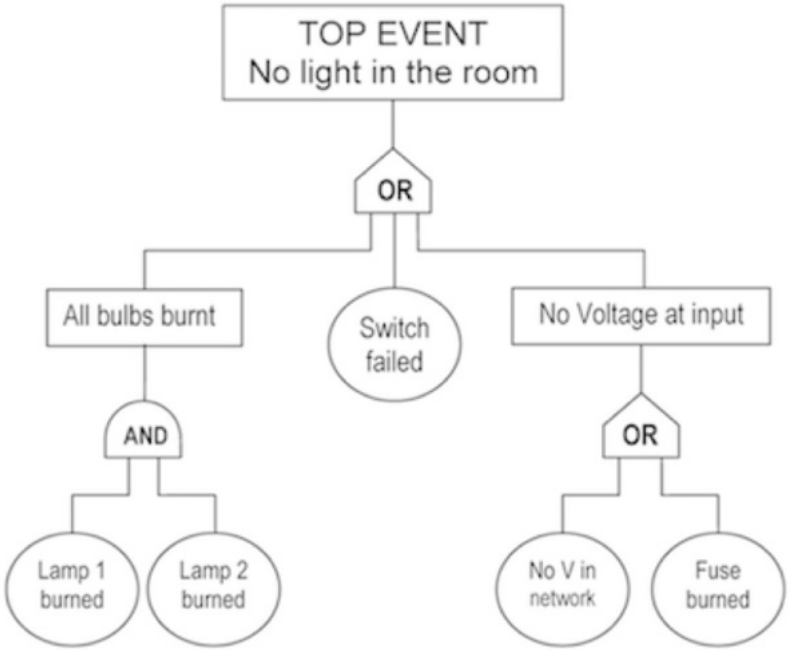


What can we do with FTA?

- **Qualitative analysis**: Determine potential root causes of a failure through **minimal cut set analysis**
- **Quantitative analysis**: Compute the probability of a failure based on the probabilities of the basic events



Minimum Cut Analysis



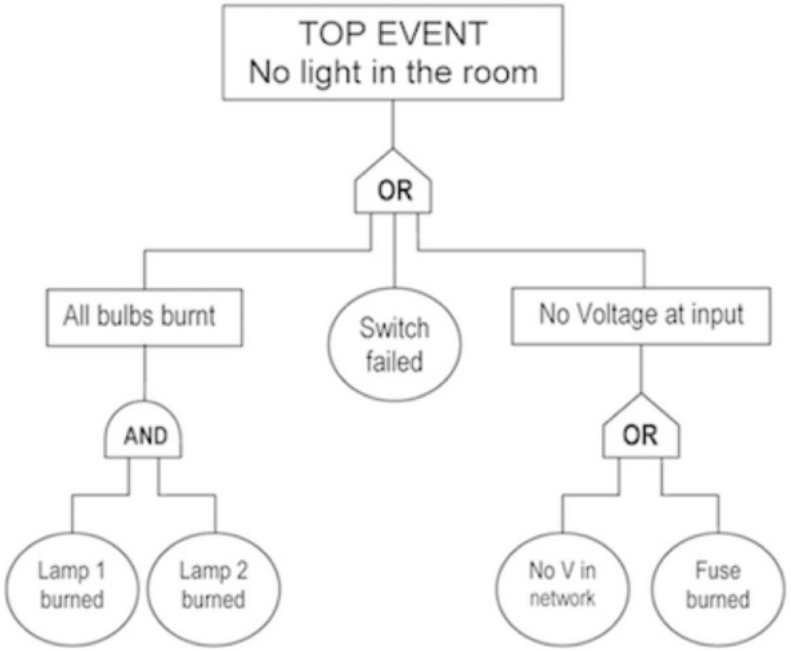
Minimal cut sets = {

??

}

- **Cut set**: A set of basic events whose simultaneous occurrence 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 a basic event.

Minimum Cut Analysis

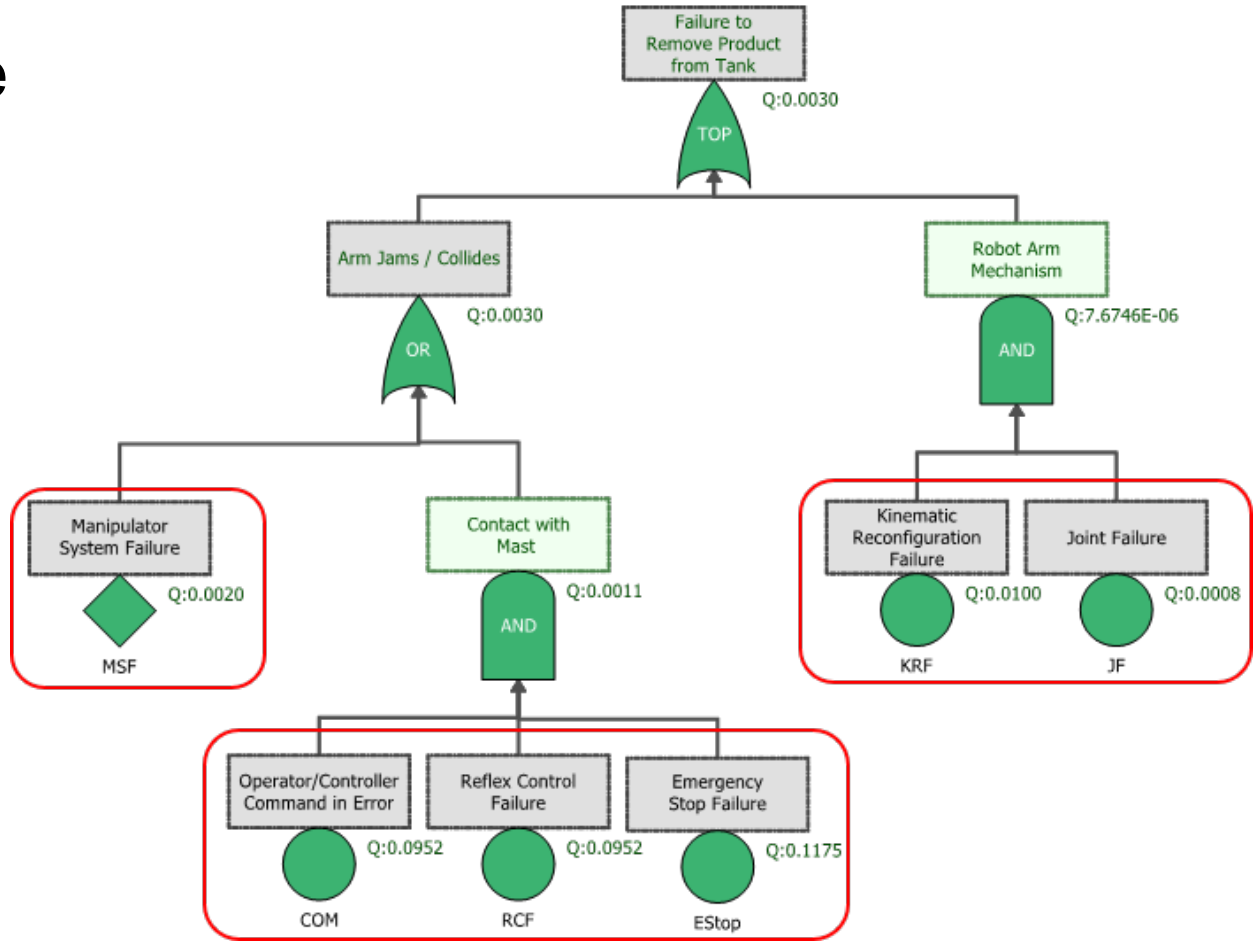


Minimal cut sets = {
{Lamp 1 burned, Lamp 2 burned},
{Switch failed},
{No V in network},
{Fuse burned}
}

- **Cut set**: A set of basic events whose simultaneous occurrence 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 a basic event.

Failure Probability Analysis

- To compute the probability of the top event:
  - Assign probabilities to basic events (based on data analysis or domain knowledge)
  - Apply probability theory to compute probabilities of intermediate events through AND & OR gates
- Alternatively, compute the top event probability as a sum of prob. of minimal cut sets
- **Q. This is difficult to do with software – why?**

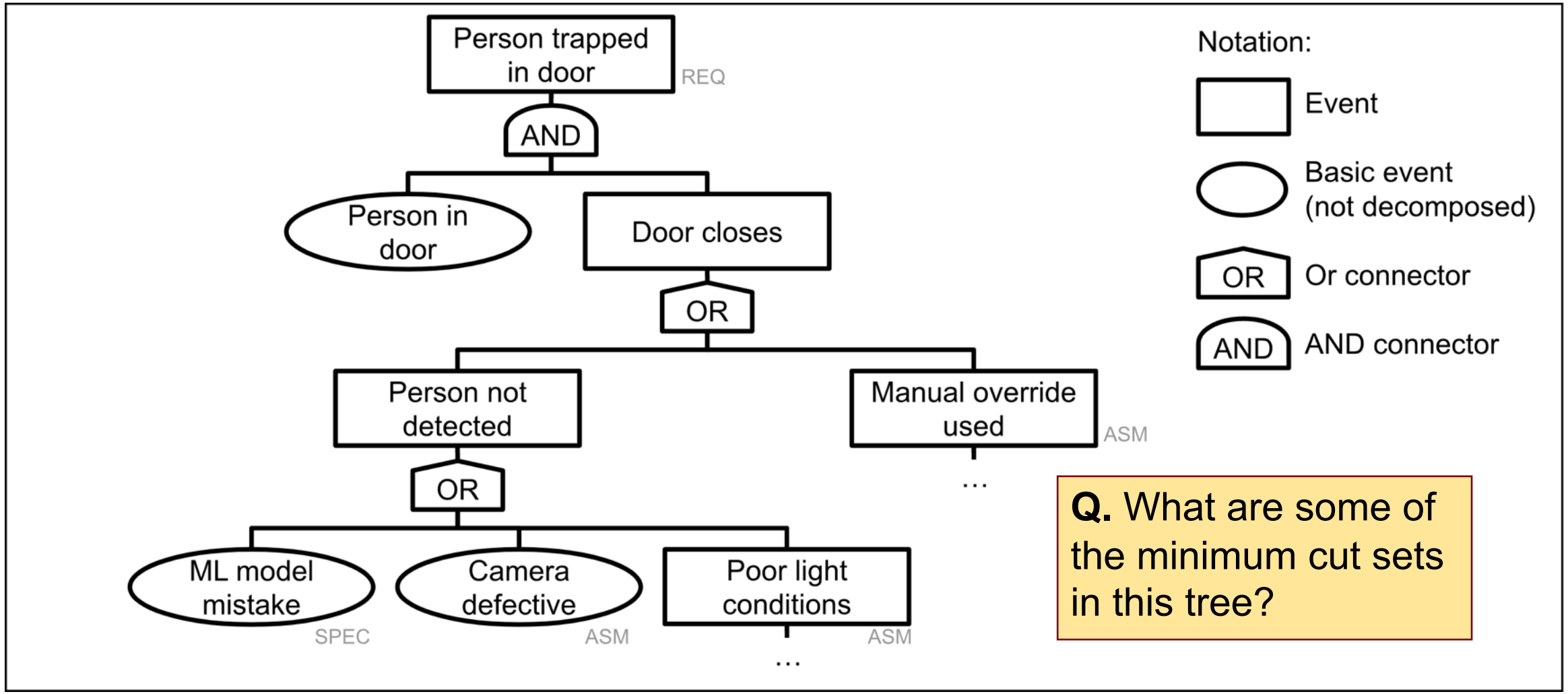


Example: Autonomous Train

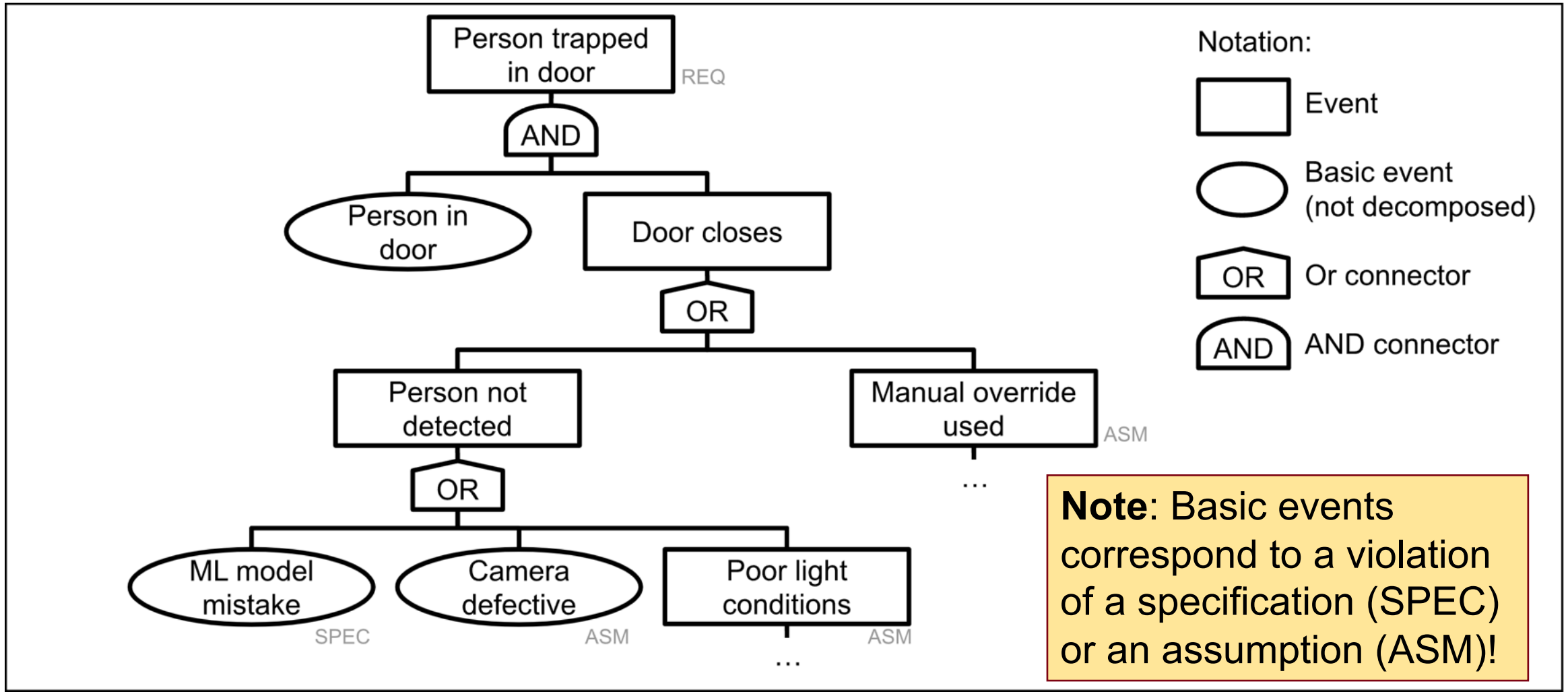
Example: Autonomous Train

- **Requirements**: The train shall not depart all doors are closed. The train shall not trap people between the doors.
- Train uses a vision-based system to identify people in the door
- Use a fault tree to identify possible ways in which the person may be trapped in a door.

FTA Example: Autonomous Train



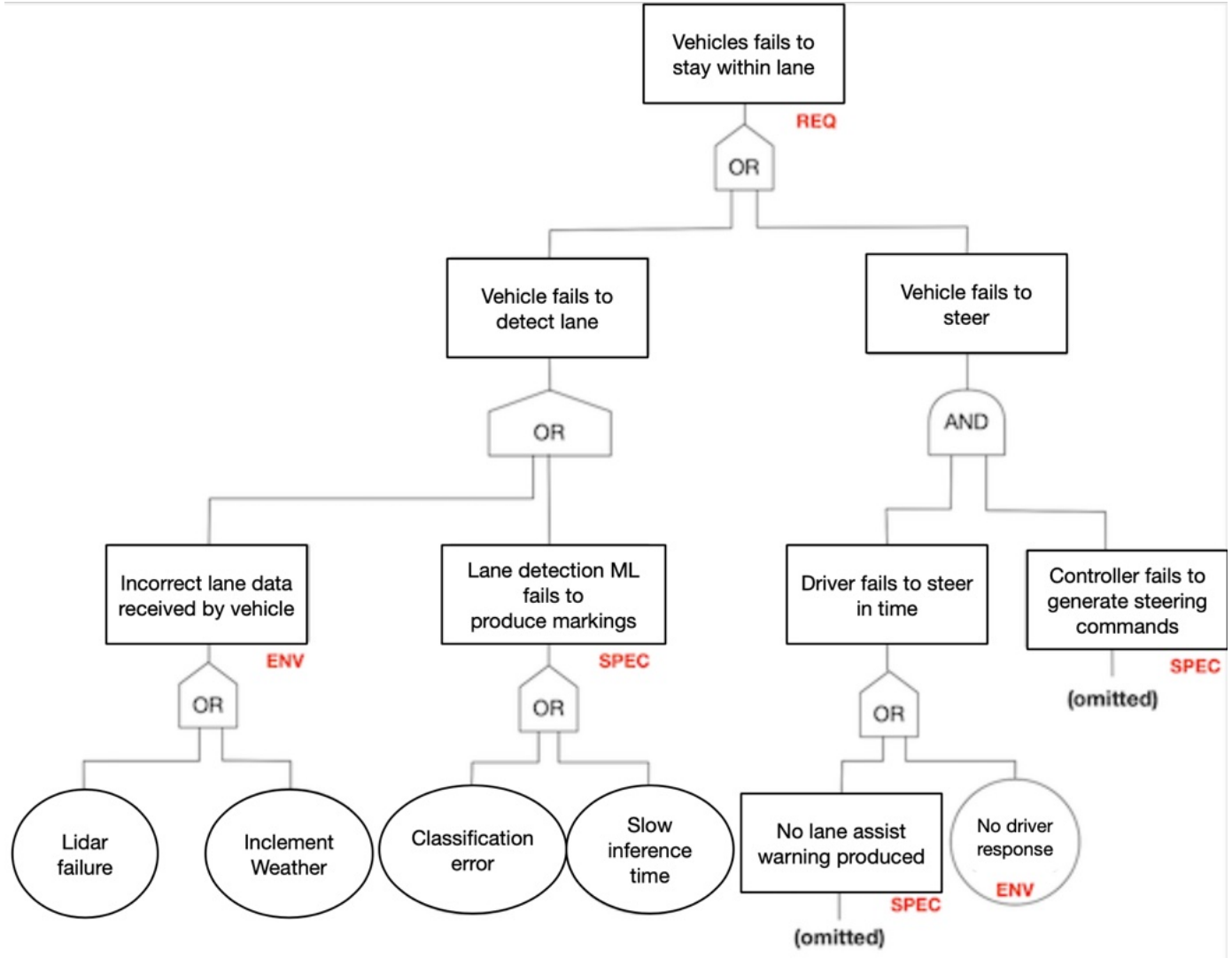
FTA Example: Autonomous Train



FTA Exercise: Lane Keeping Assist

- **Requirement**: The vehicle must be prevented from going off the lane.
- Use the failure to satisfy this as the TOP event
- Perform FTA to identify possible causes of this failure

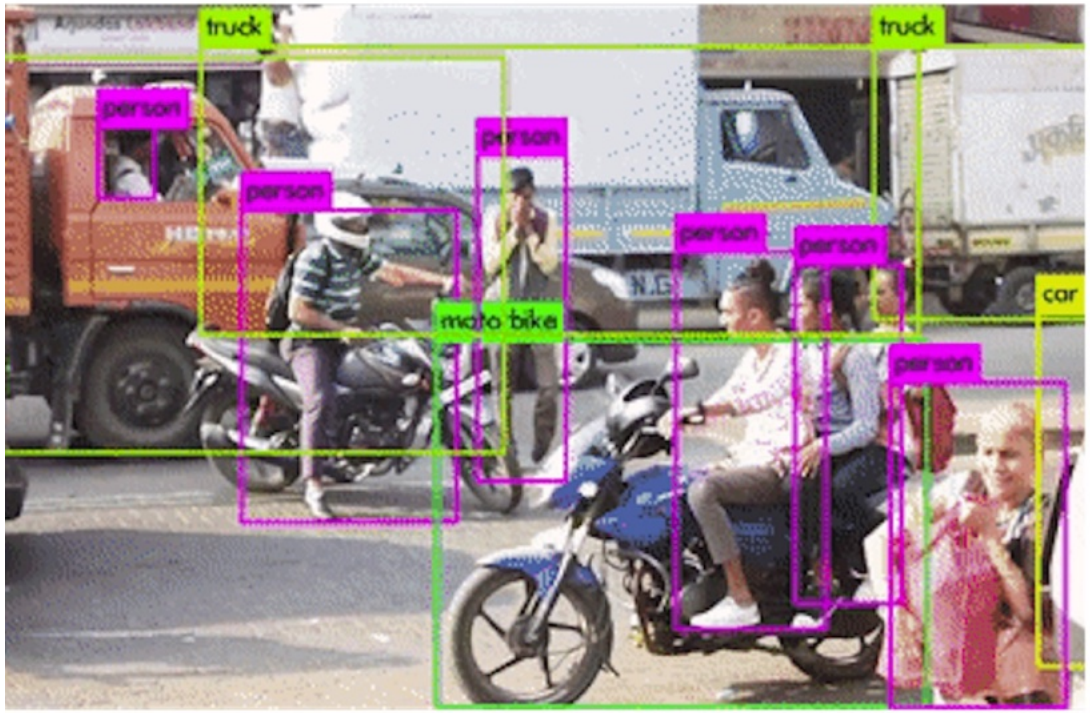
FTA Exercise: Lane Keeping System



FTA: Caveats

- In general, building a “complete” tree is impossible
  - There are probably some faulty events that you missed (i.e., ”unknown unknowns”)
- Domain knowledge is crucial for improving coverage
  - Talk to domain experts to identify important and common basic events for your application domain
- FTA is still very valuable for risk reduction!
  - Forces you to think about & explicitly document possible failure scenarios
  - A good starting basis for designing mitigations (more on this in the next lecture)

Hazard and Operability Study (HAZOP)



Guide Word	Meaning
NO OR NOT	Complete negation of the design intent
MORE	Quantitative increase
LESS	Quantitative decrease
AS WELL AS	Qualitative modification/increase
PART OF	Qualitative modification/decrease
REVERSE	Logical opposite of the design intent
OTHER THAN / INSTEAD	Complete substitution
EARLY	Relative to the clock time
LATE	Relative to the clock time
BEFORE	Relating to order or sequence
AFTER	Relating to order or sequence

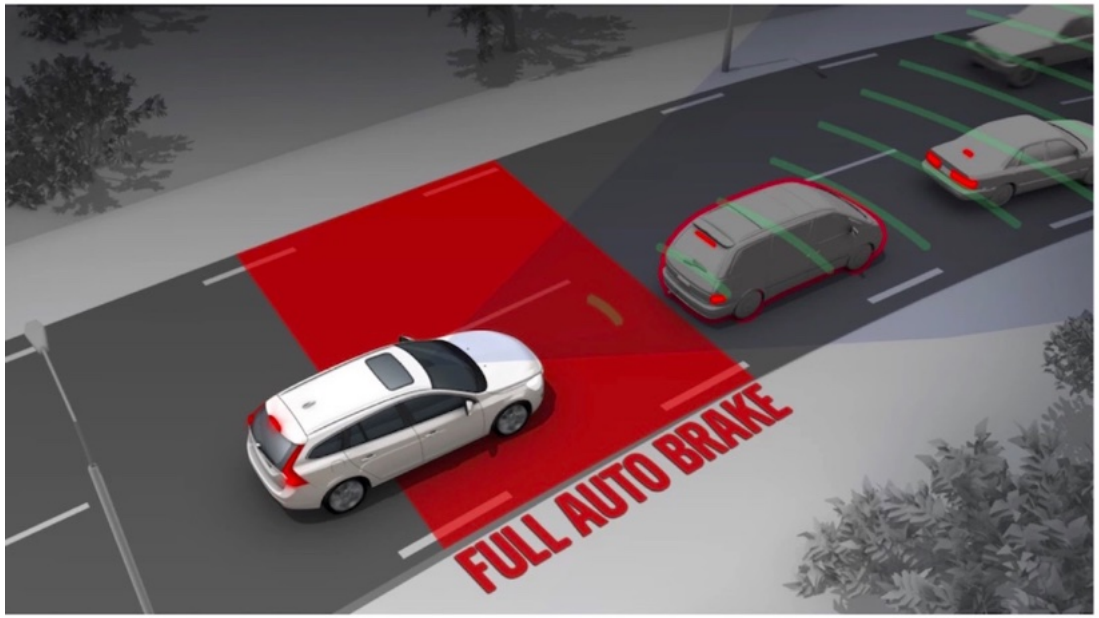
- **Goal**: Identify hazards and component faults through systematic, pattern-based inspection of component functions

HAZOP

- HAZOP is a **bottom-up** method to identify potential failures: It starts from individual components
  - FTA is a **top-down** method: It starts from a top-level failure and links it to component-level faults
- HAZOP process:
  - For each component, specify the expected behavior of the component (SPEC)
  - Use a set of **guide words** to generate possible deviations from expected behavior
  - Analyze the impact of each generated deviation: Can it result in a system-level failure?

Guide Word	Meaning
NO OR NOT	Complete negation of the design intent
MORE	Quantitative increase
LESS	Quantitative decrease
AS WELL AS	Qualitative modification/increase
PART OF	Qualitative modification/decrease
REVERSE	Logical opposite of the design intent
OTHER THAN / INSTEAD	Complete substitution
EARLY	Relative to the clock time
LATE	Relative to the clock time
BEFORE	Relating to order or sequence
AFTER	Relating to order or sequence

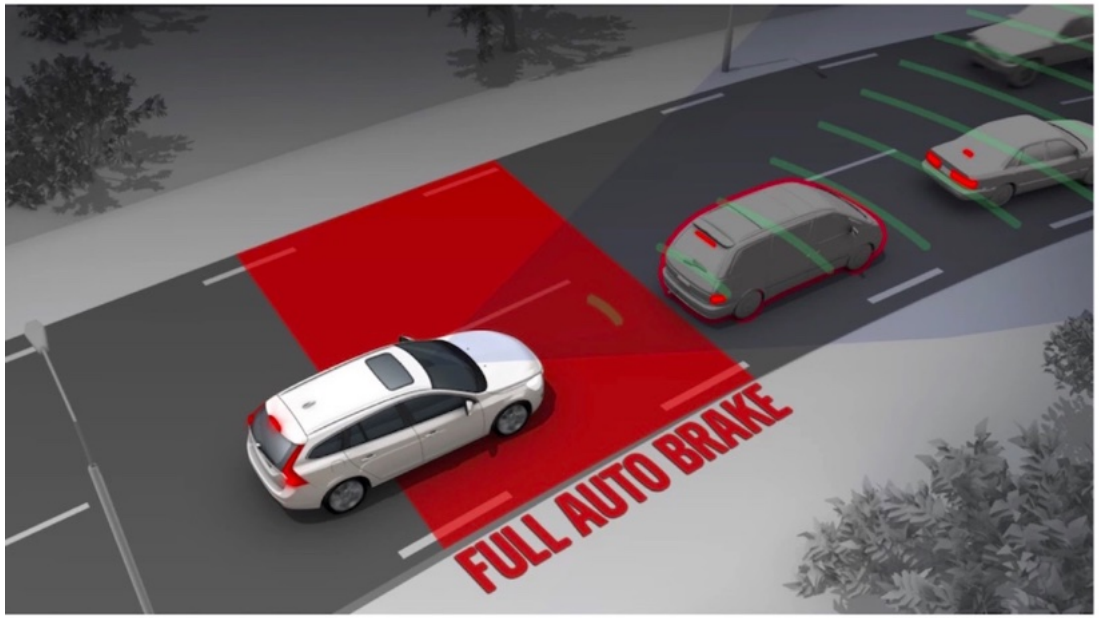
HAZOP Example: Emergency Braking (EB)



Guide Word	Meaning
NO OR NOT	Complete negation of the design intent
MORE	Quantitative increase
LESS	Quantitative decrease
AS WELL AS	Qualitative modification/increase
PART OF	Qualitative modification/decrease
REVERSE	Logical opposite of the design intent
OTHER THAN / INSTEAD	Complete substitution
EARLY	Relative to the clock time
LATE	Relative to the clock time
BEFORE	Relating to order or sequence
AFTER	Relating to order or sequence

- Component: Software controller for EB
  - Expected behavior (SPEC): If the ego vehicle is too close to the leading vehicle, generate a maximum amount of braking to prevent collision

HAZOP Example: Emergency Braking (EB)



Guide Word	Meaning
NO OR NOT	Complete negation of the design intent
MORE	Quantitative increase
LESS	Quantitative decrease
AS WELL AS	Qualitative modification/increase
PART OF	Qualitative modification/decrease
REVERSE	Logical opposite of the design intent
OTHER THAN / INSTEAD	Complete substitution
EARLY	Relative to the clock time
LATE	Relative to the clock time
BEFORE	Relating to order or sequence
AFTER	Relating to order or sequence

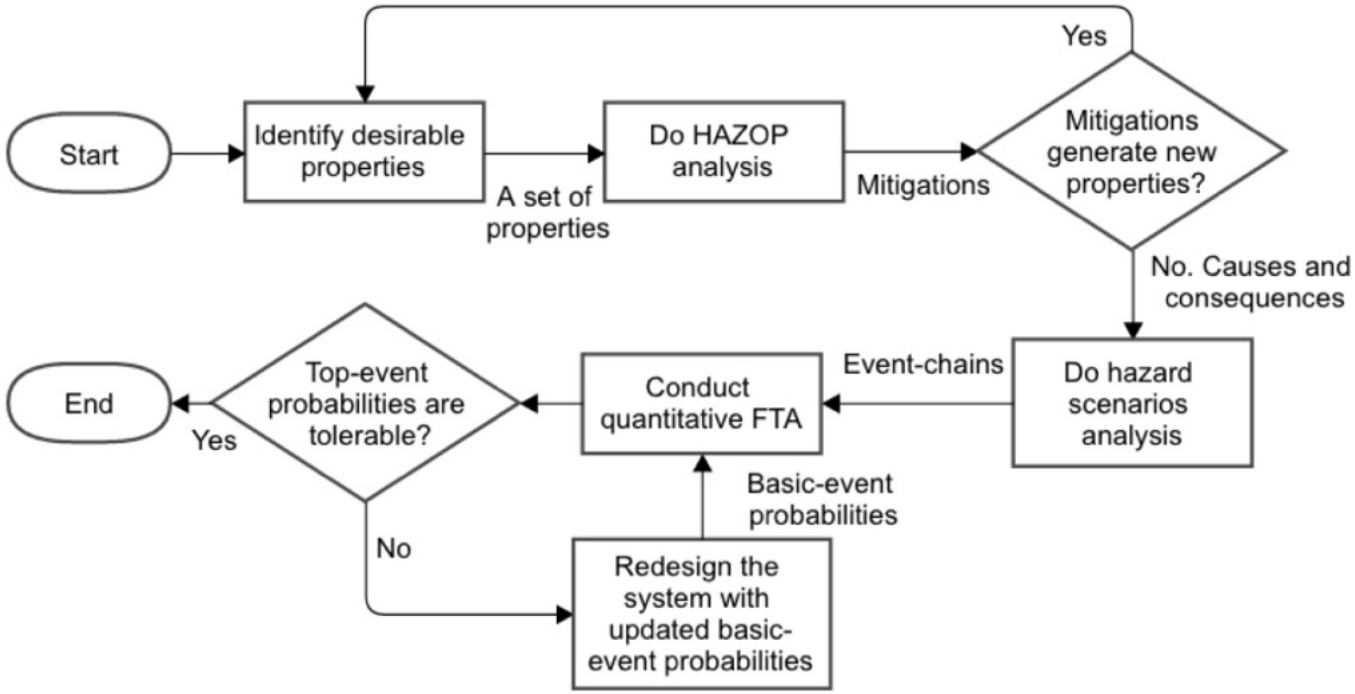
- Expected: EB must apply a maximum braking command to the engine.
- NO OR NOT: EB does not generate any braking command.
- LESS: EB applies less than max. braking.
- LATE: EB applies max. braking but after a delay of 2 seconds.
- REVERSE: EB generates an acceleration command instead of braking.

HAZOP Exercise: Lane Keeping Assist

Guide Word	Meaning
NO OR NOT	Complete negation of the design intent
MORE	Quantitative increase
LESS	Quantitative decrease
AS WELL AS	Qualitative modification/increase
PART OF	Qualitative modification/decrease
REVERSE	Logical opposite of the design intent
OTHER THAN / INSTEAD	Complete substitution
EARLY	Relative to the clock time
LATE	Relative to the clock time
BEFORE	Relating to order or sequence
AFTER	Relating to order or sequence

- **Component:** ML model for lane detection
  - **Expected behavior (SPEC):** Given a sensor image of the ground, the ML model detects the presence/absence of lane markings
- Apply HAZOP guidewords to identify different ways in which this component might deviate from expected behavior

HAZOP: Benefits & Limitations



- Encourages systematic reasoning about component faults
- Can be combined with FTA to generate faults (i.e., basic events in FTA)
- Potentially labor-intensive; relies on engineer's judgement
- Does not guarantee to find all failures (but this is true for every method!)

Summary

- Exit ticket!
- **Next lecture**: Principles and patterns for how to improve the robustness of a system