# 17-423/723: Software System Design

## Design for Robustness

Mar 23, 2026

# Logistics

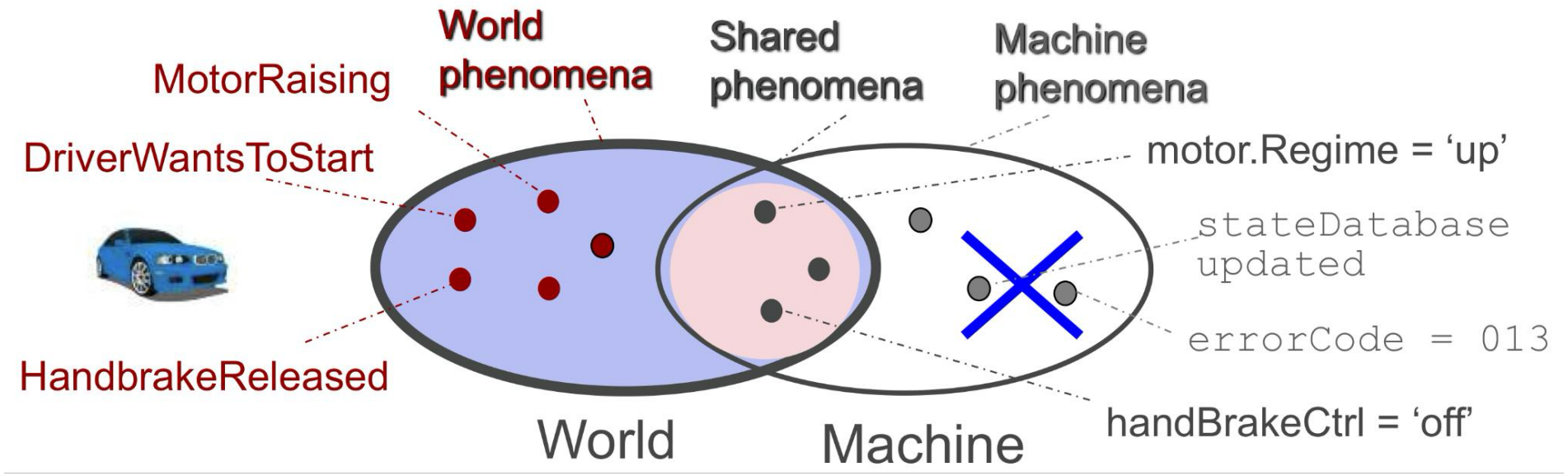
- M4 released
- Two parts & deadlines
  - April 1: Implement a prototype of your service
  - April 8: Integrate the services into an end-to-end system
- **Start early!** Your service needs to be up & running by April 8

# Learning 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
- Apply design patterns to improve the robustness of a system

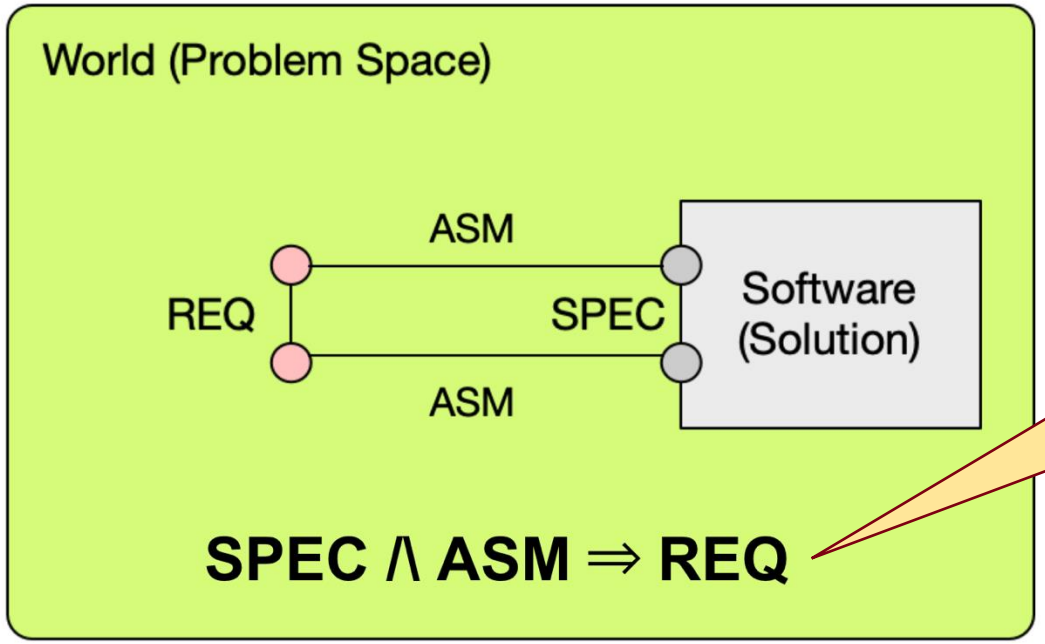
# What can possibly go wrong with my system?

# 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

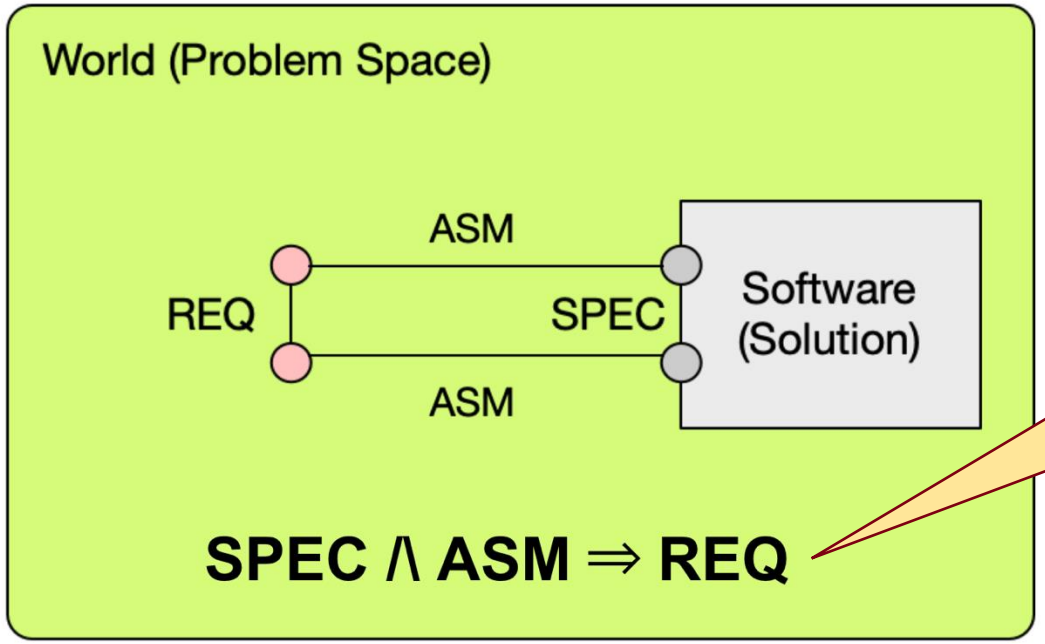
# Recall: Satisfaction Argument




"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

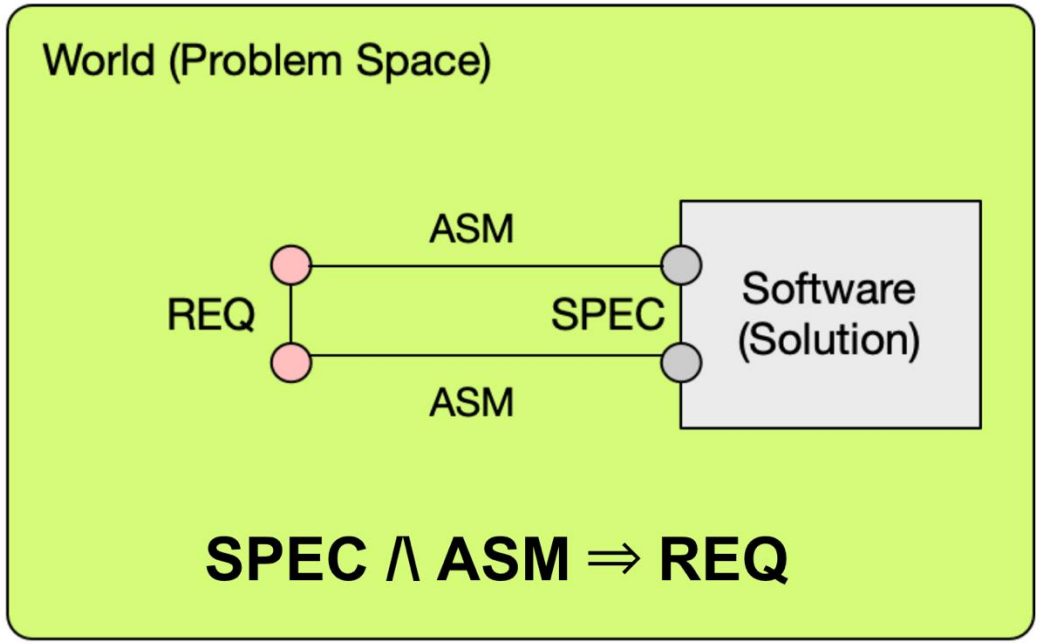
# What can go wrong in my system?




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

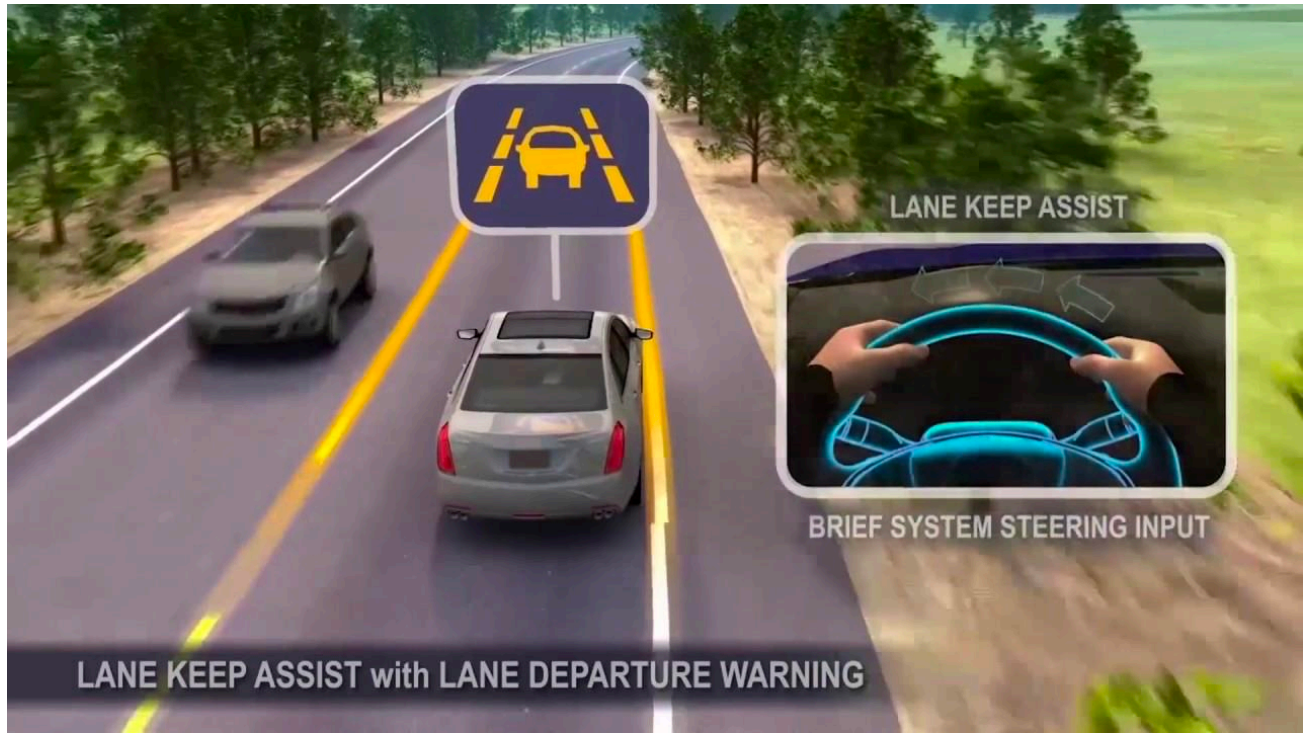
- **Q. What are some ways in which the system may fail to satisfy this argument?**

# 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)

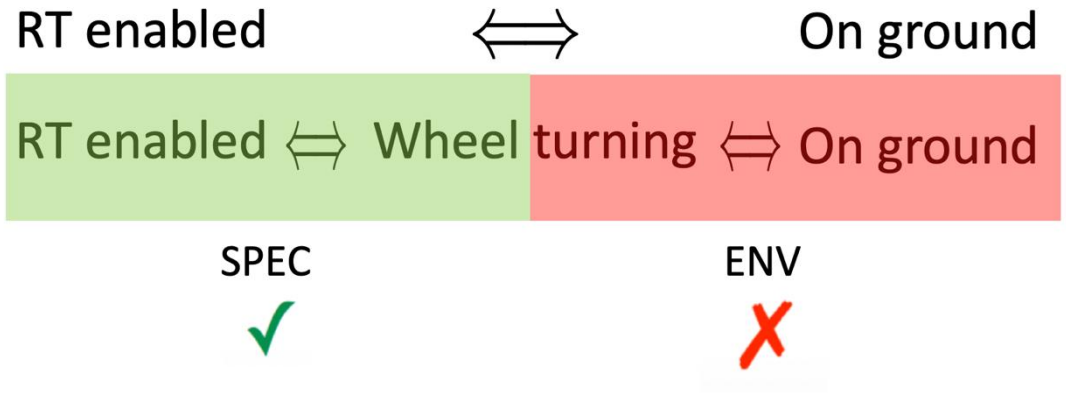
# Example: Lane Keeping Assist




**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

BUSINESS

# Major Internet outage along East Coast causes large parts of the Web to crash — again

By Timothy Bella

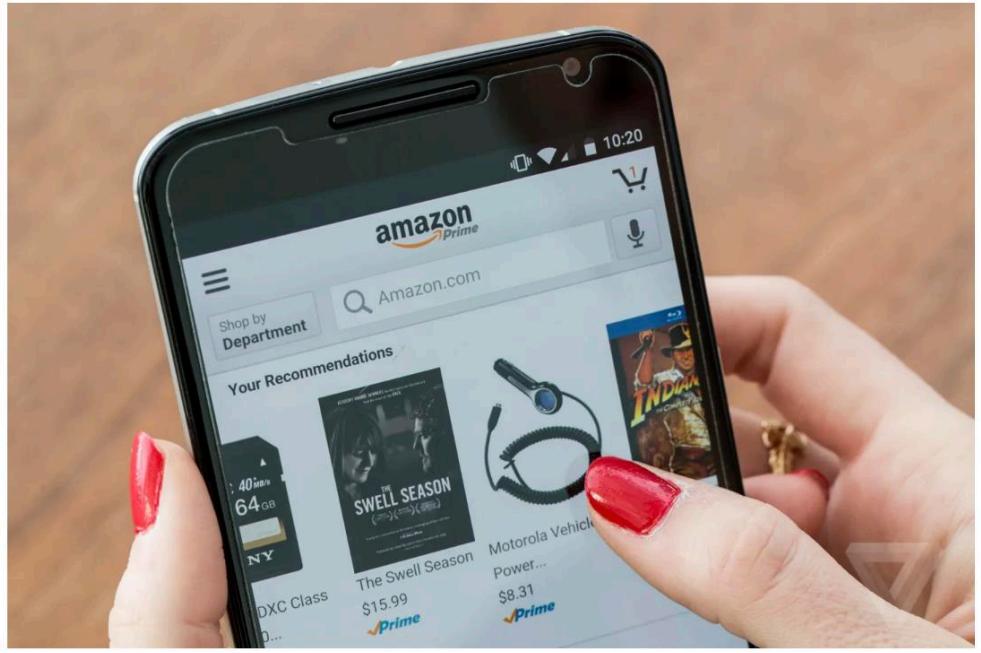
July 22, 2021 at 2:02 p.m. EDT


"...the websites of UPS, USAA, Home Depot, HBO Max and Costco were also among those affected. The websites of British Airways, GoDaddy, Fidelity, Vanguard and AT&T were among those loading slowly.

The cause of the outage, the latest major Internet outage this summer, was linked to Akamai Technologies, the global content delivery network based in Cambridge, Mass."

TECH

# How a typo took down S3, the backbone of the internet




/ Hello, operator

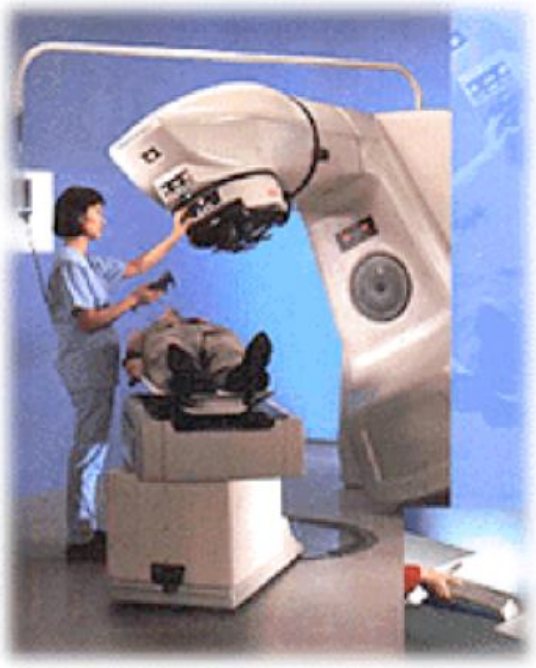
by **Casey Newton**
Mar 2, 2017, 1:24 PM EST

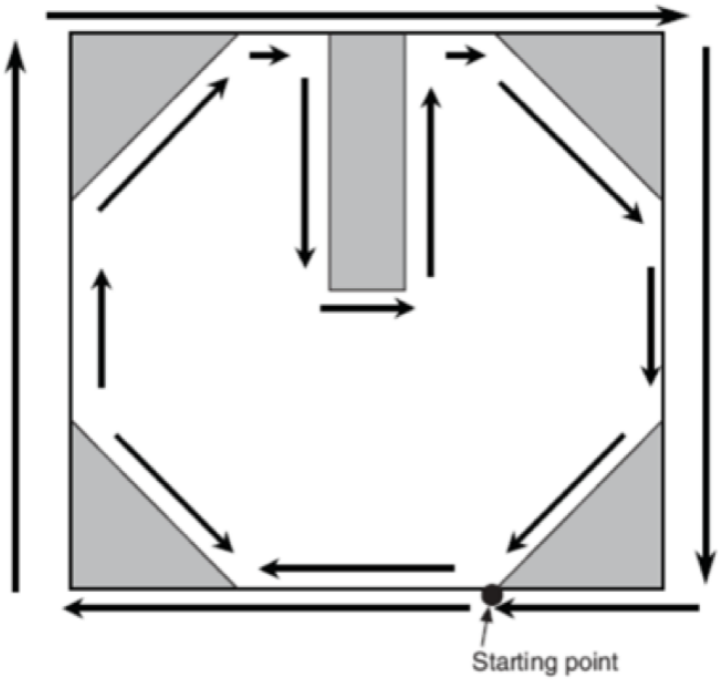
0 Comments

*"Unfortunately, one of the inputs to the command was entered incorrectly… The servers that were inadvertently removed supported two other S3 subsystems. One of these subsystems, the index subsystem, manages the metadata and location information of all S3 objects in the region."*

- A typo by a network engineer shuts down many S3 servers (2017)
- Major websites (Slack, Venmo, Trello…) down for 4 hours; $150M loss
- After the incident: Added safeguards to prevent similar failures & ensure fast recovery

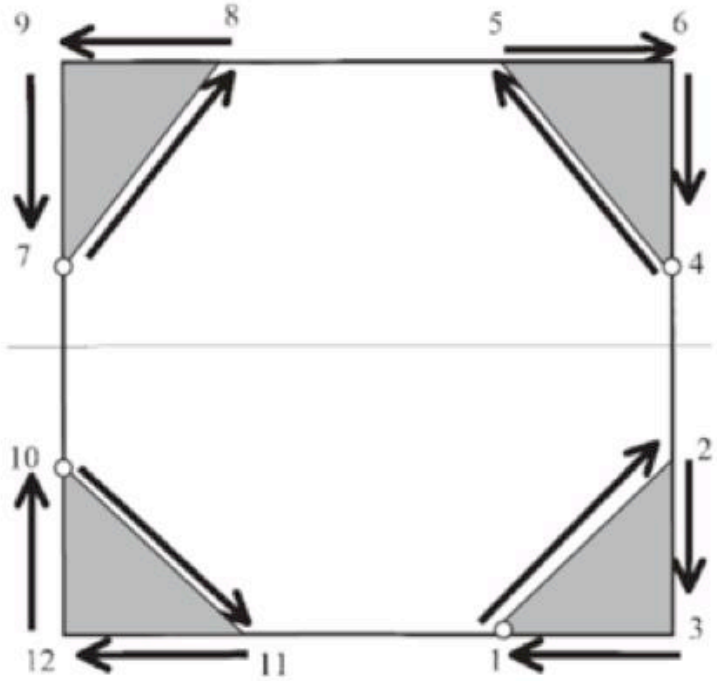
# Another Example: Panama City Hospital (2000)





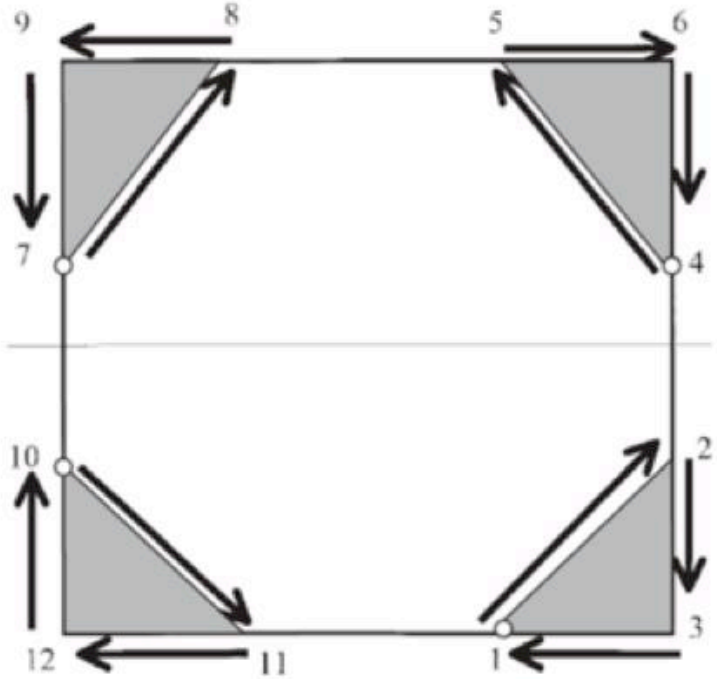

- Therapy planning software by Multidata Systems
- Theratron-780 by Theratronics (maker of Therac-25)
- **Shielding blocks**: Inserted into beam path to protect healthy tissue
- Therapist draws block shapes; software computes amount of radiation dose

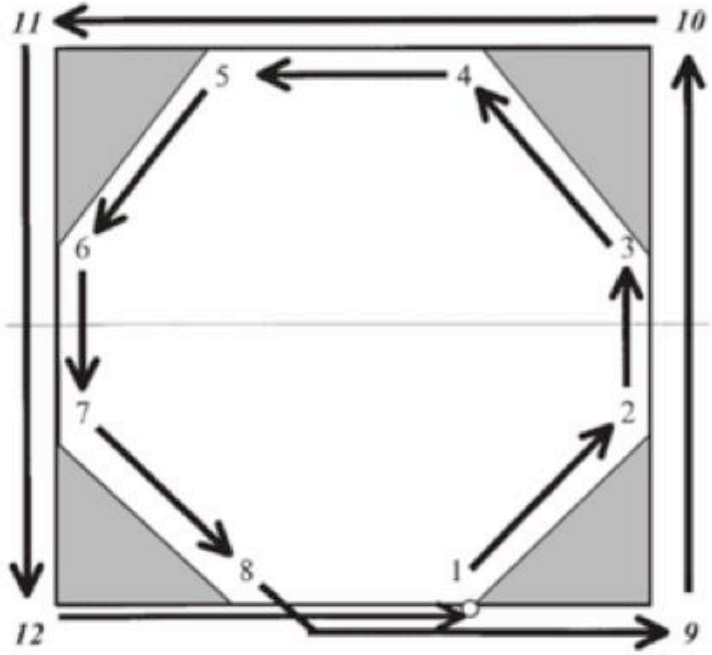
# Example: Panama City Hospital




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 “perfect”
  - The environment will sometimes behave in unexpected ways, violating assumptions (ASM)
  - Software will have bugs and fail from time to time, violating its specification (SPEC)
- 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 shut down, the system should continue to provide critical services
- **Q. How to design systems to be robust against abnormal events?**

# Robustness

# Robustness

- The ability of a system to provide an **acceptable level of service** even when it operates under **abnormal conditions**
- Acceptable service: Functionality or quality attribute (of high importance) to be preserved, such as:
  - **Safety**: “No unsafe level of radiation delivered to the patient”
  - **Performance**: “The 95$^{th}$-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”

# 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
  - Robustness covers both internal faults & external deviations
- 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 often necessary to achieve high reliability

# Specifying Robustness: Good & Bad Examples

- The radiation therapy system should never deliver more than a safe amount of radiation even under data entry errors ✅
- The autonomous vehicle must operate fully 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 number of customers spikes above an expected threshold ✅
- 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 ✅

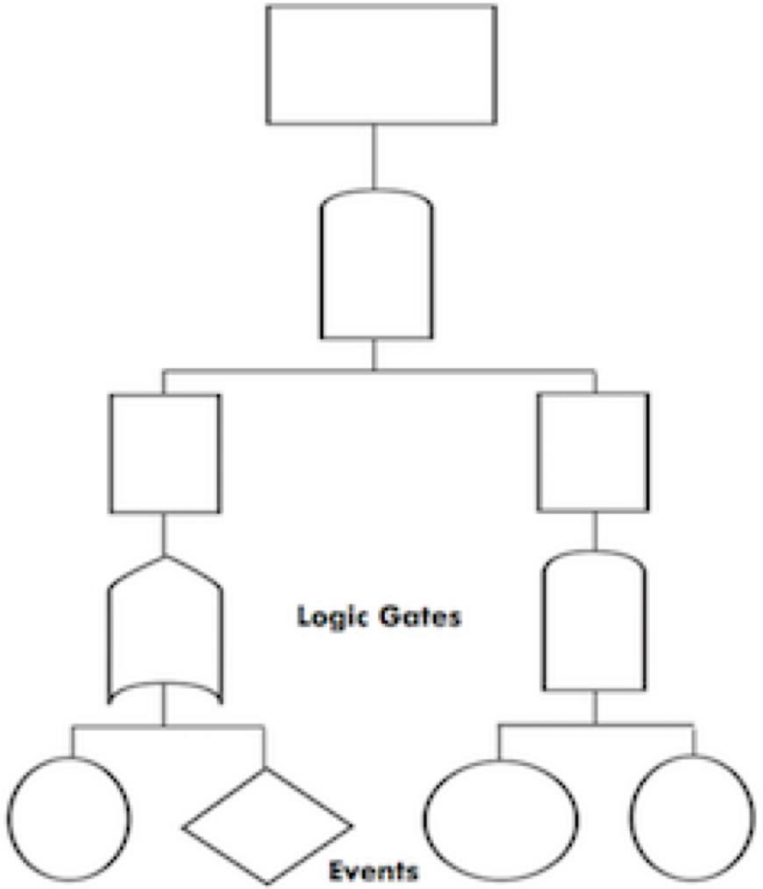
# Failure Analysis

# Failure Analysis

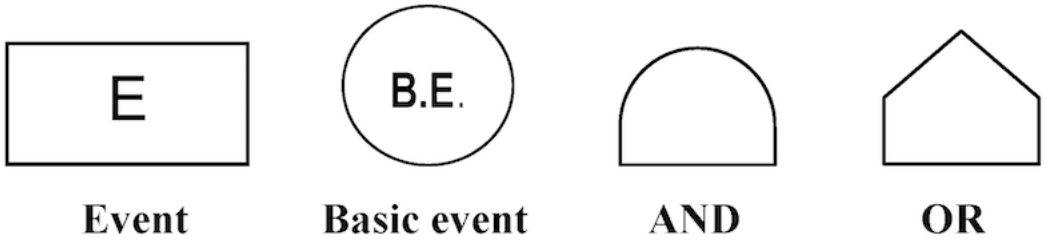
- *What can possibly go wrong in my system, and what is potential impact on system requirements?*
- Systematically analyze a design and identify different scenarios in which the system may fail to satisfy its requirements/QAs
- A number methods, developed and routinely applied in many engineering disciplines
  - **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…)



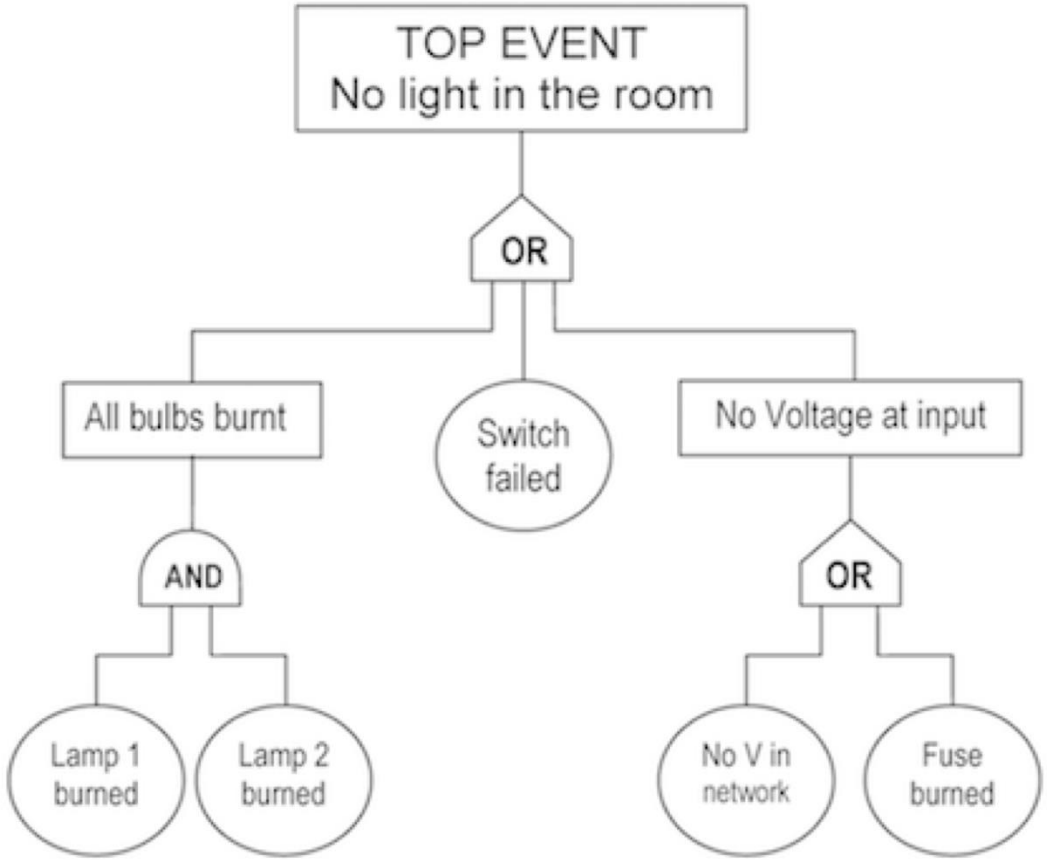
# Elements of Fault Trees




- 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

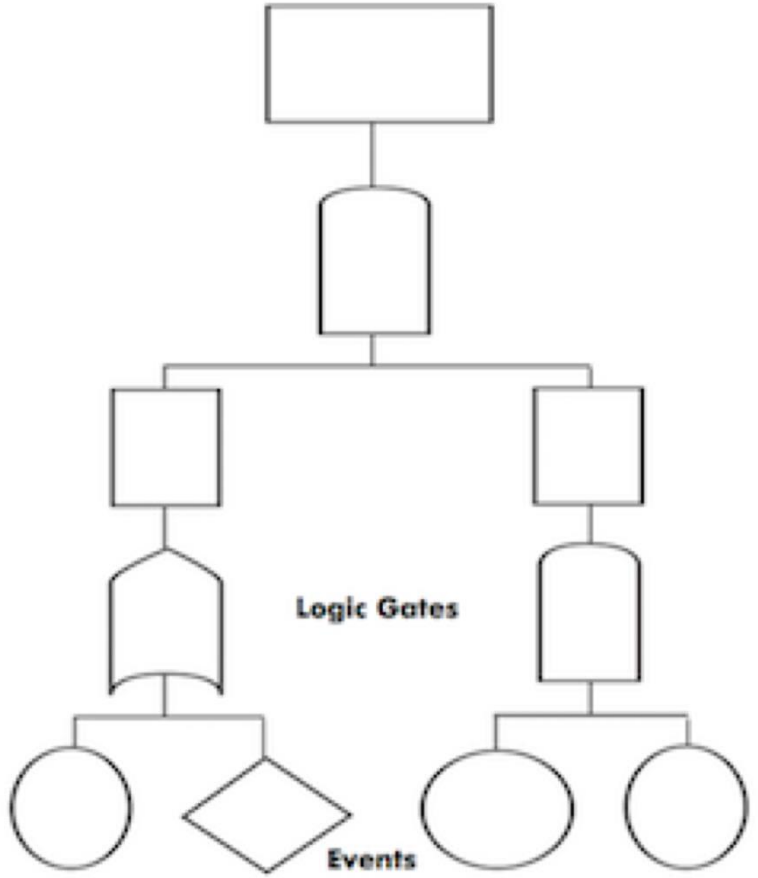
# Elements of Fault Trees

- 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

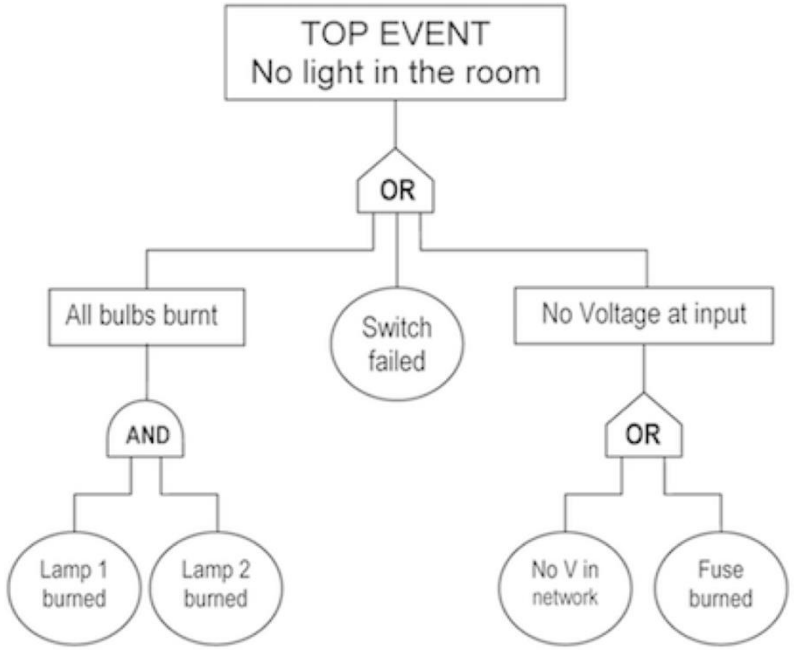


# What can we do with FTA?

- **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



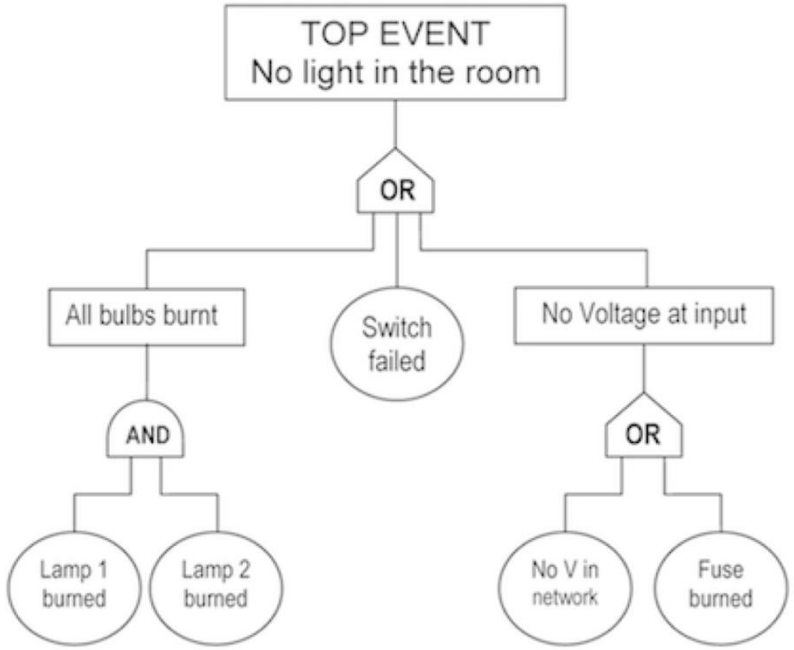

Minimal cut sets = {

??

}

- **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

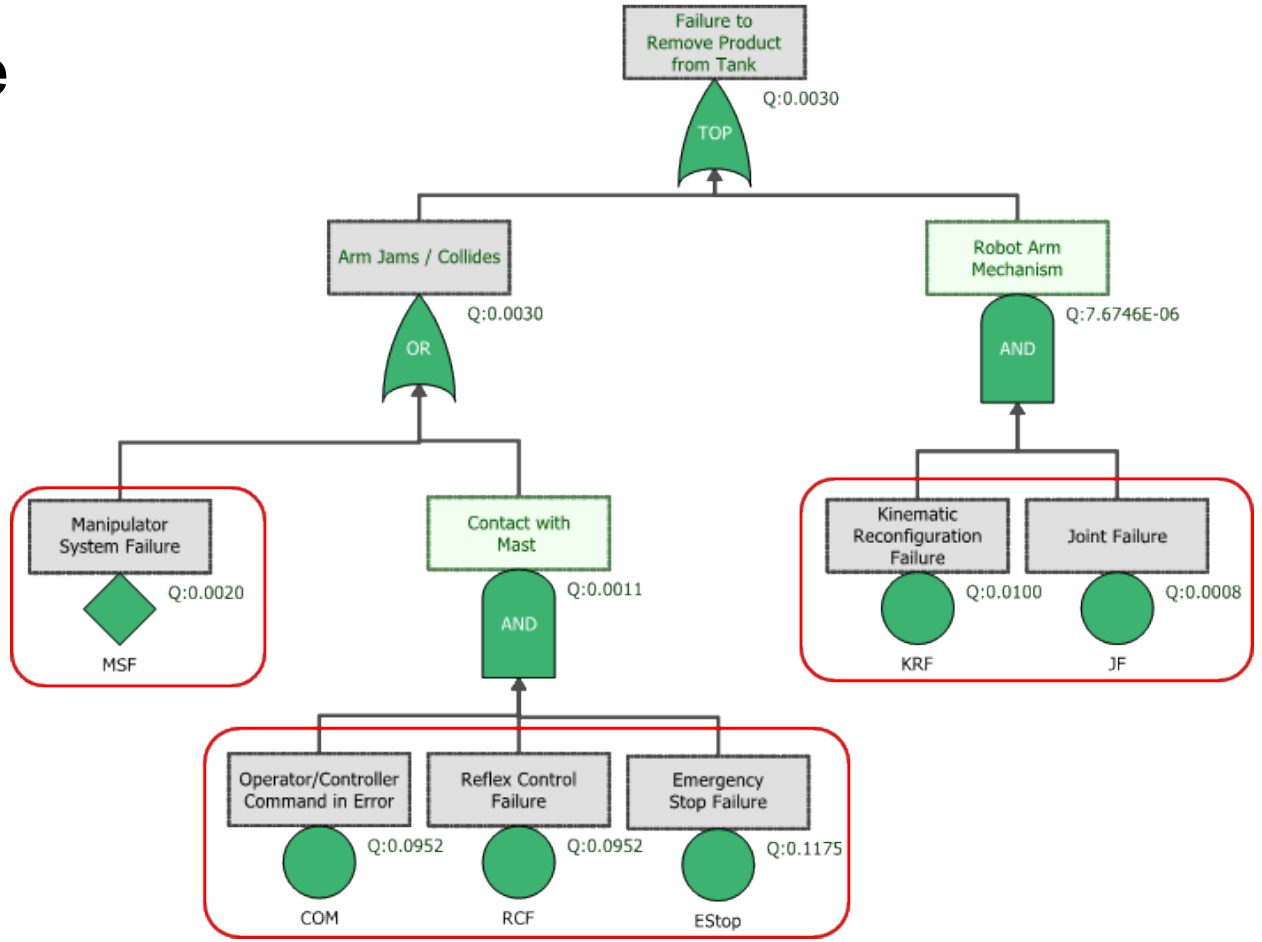



Minimal cut sets = {
{Lamp 1 burned, Lamp 2 burned},
{Switch failed},
{No V in network},
{Fuse burned}
}

- **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?**

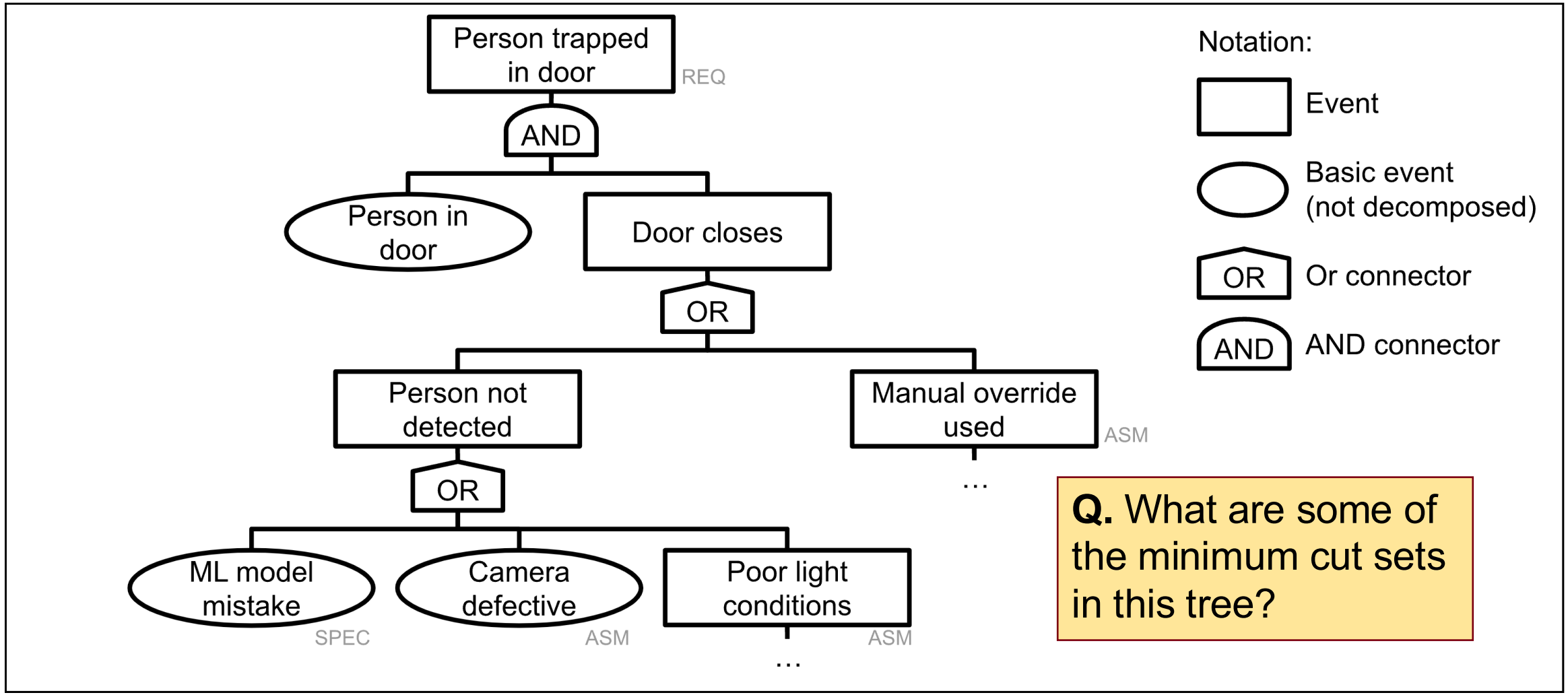


# Example: Autonomous Train

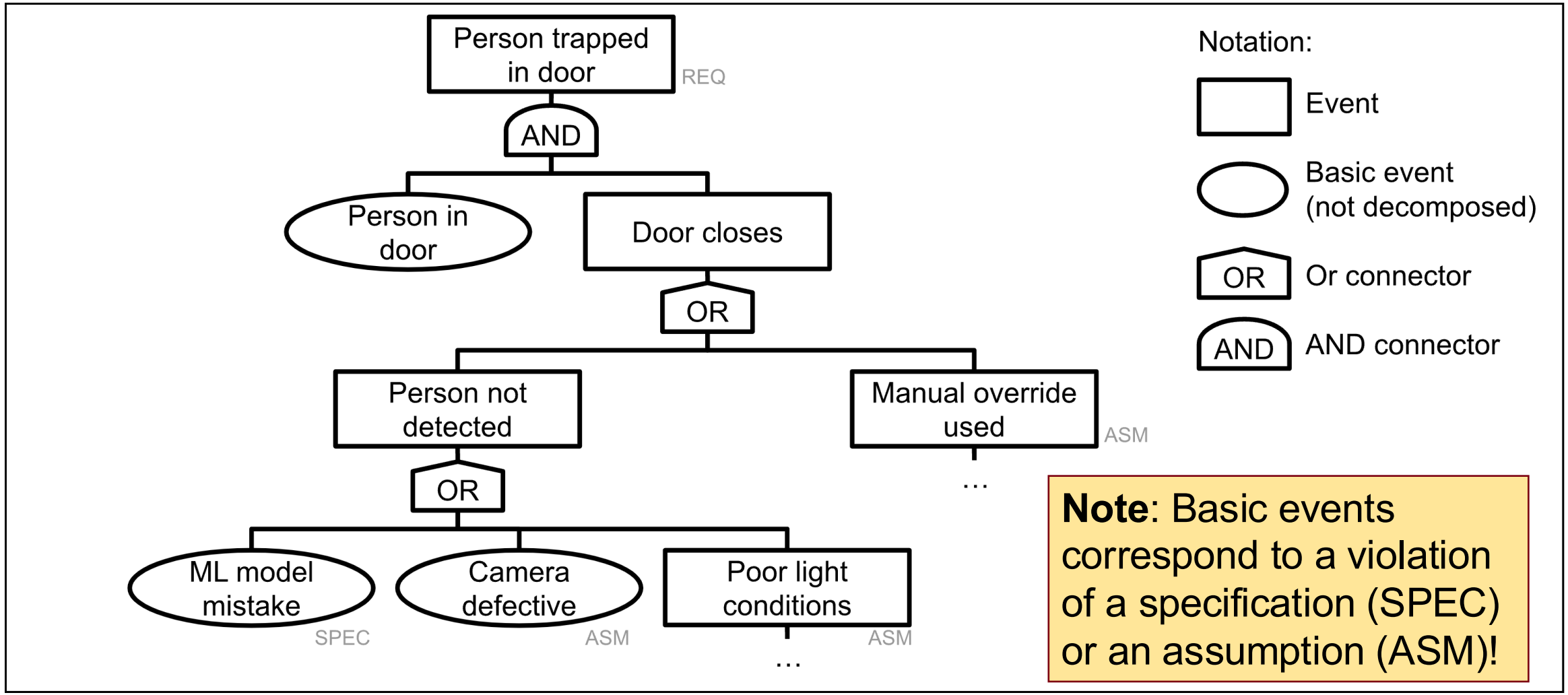
# Example: Autonomous Train

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

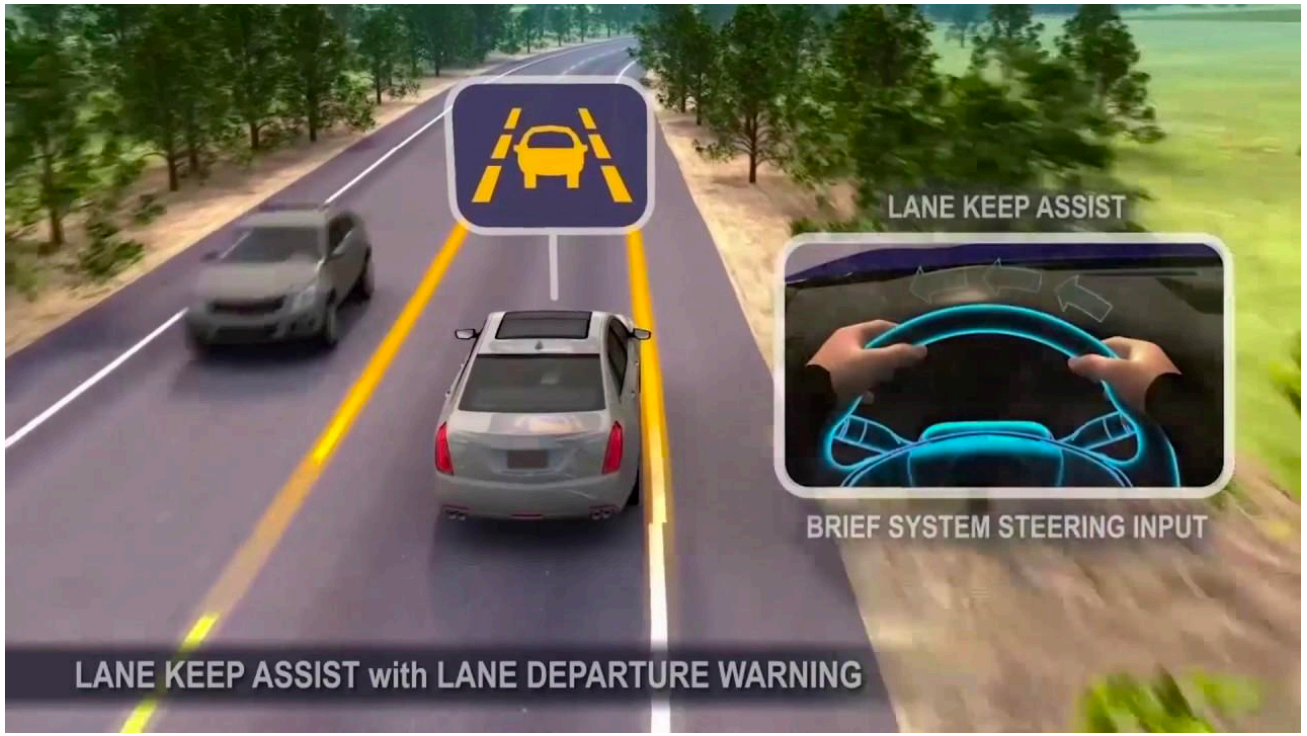
# FTA Example: Autonomous Train



# FTA Example: Autonomous Train

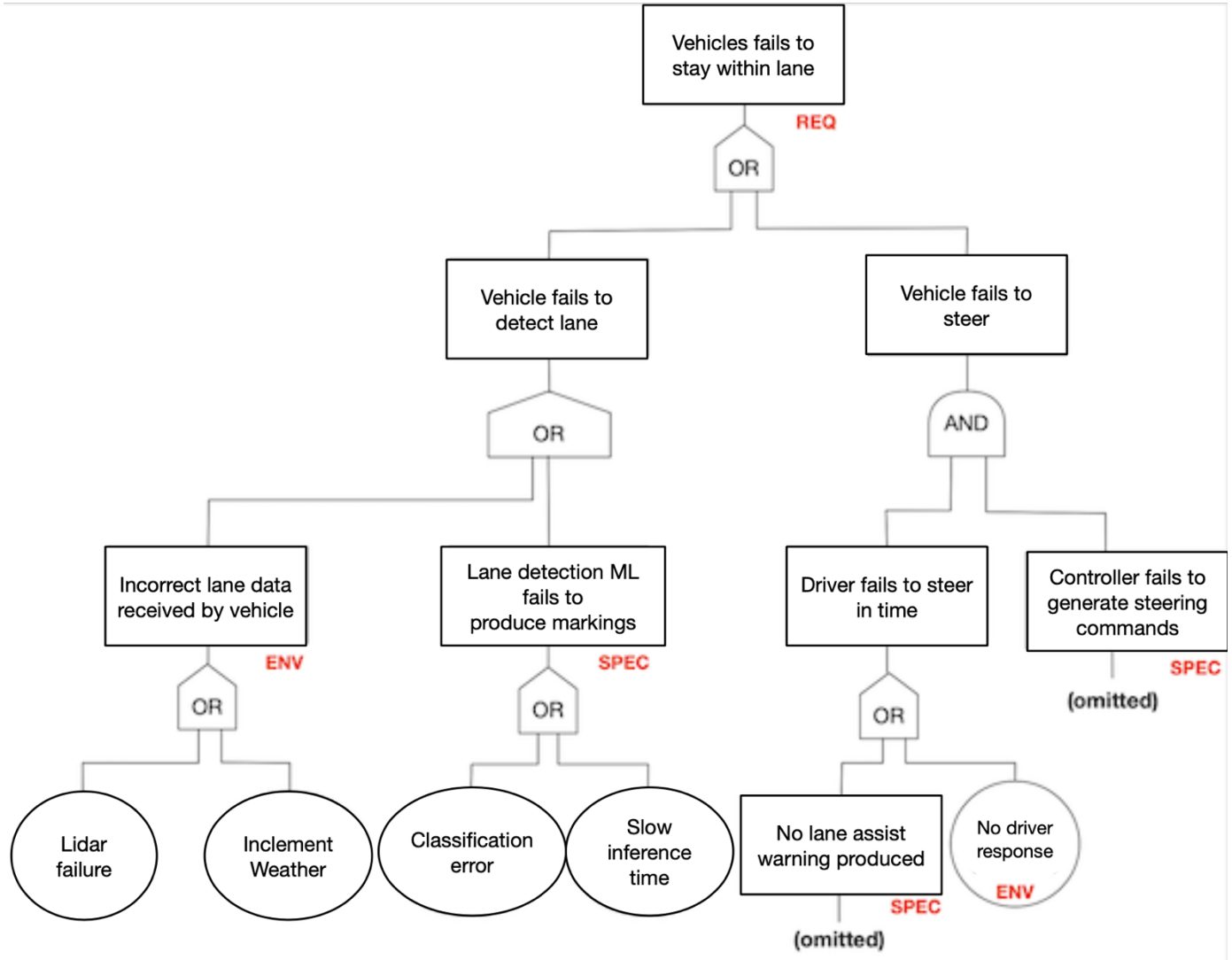


# FTA Exercise: Lane Keeping Assist




- **Requirement**: The vehicle must be prevented from going off the lane.
- **TOP event:** “Vehicle fails to stay within the lane”
- Apply FTA to identify possible causes of this failure

# FTA Exercise: Lane Keeping System



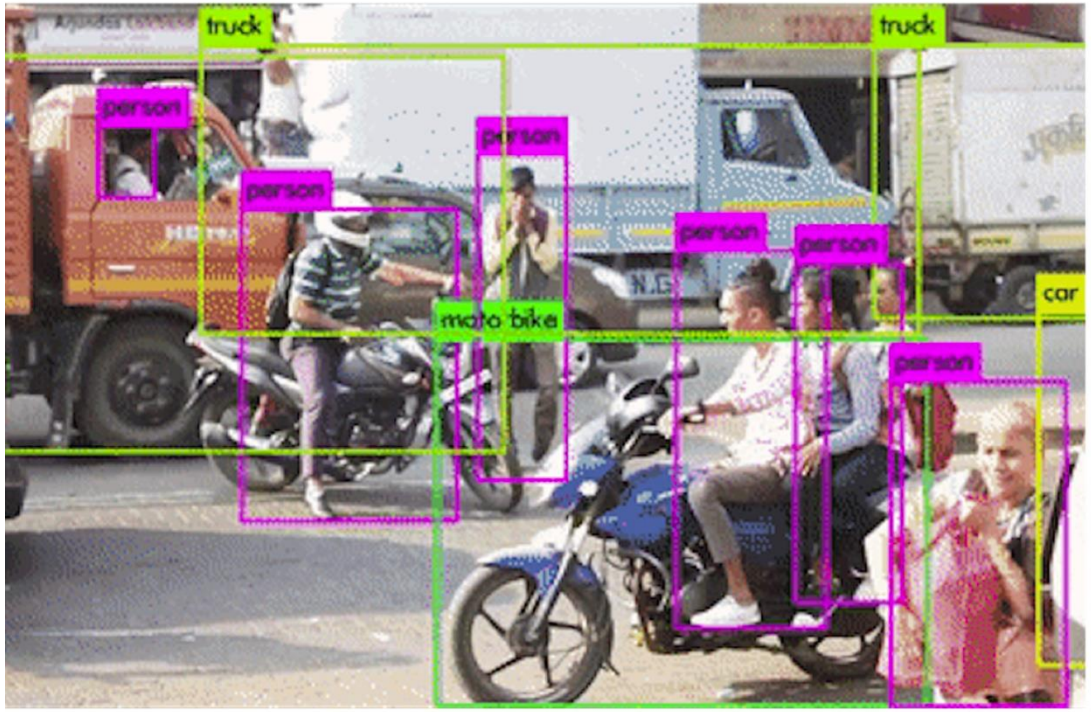
# FTA: Benefits & Caveats

- 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 designing robust systems!
  - Forces you to think about & explicitly document possible failure scenarios
  - The outcome is a good starting basis for designing mitigations (more on this in the next lecture)

# 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/QAs
- 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
  - ...

# 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 |

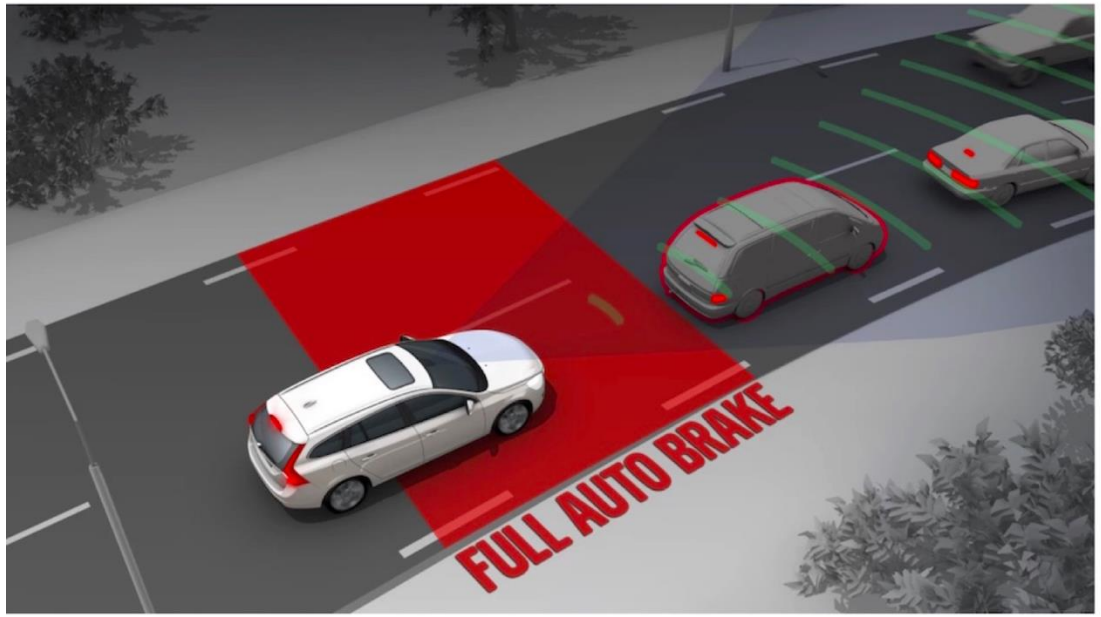
- **Goal**: Identify component faults and hazards (i.e., system failures ) through systematic, pattern-based inspection of component functions

# HAZOP

- HAZOP is a **bottom-up** method for identifying potential failures: It starts from individual components
  - vs. FTA, a **top-down** method
- 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 lead to 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 |

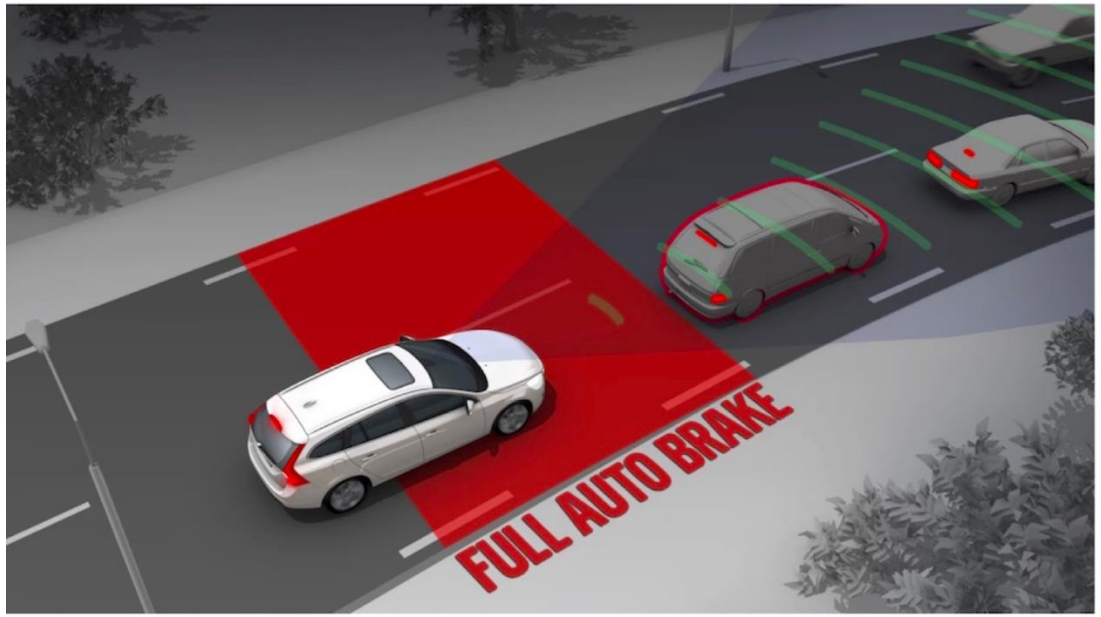
# HAZOP Example: Emergency Braking (EB)




| 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)

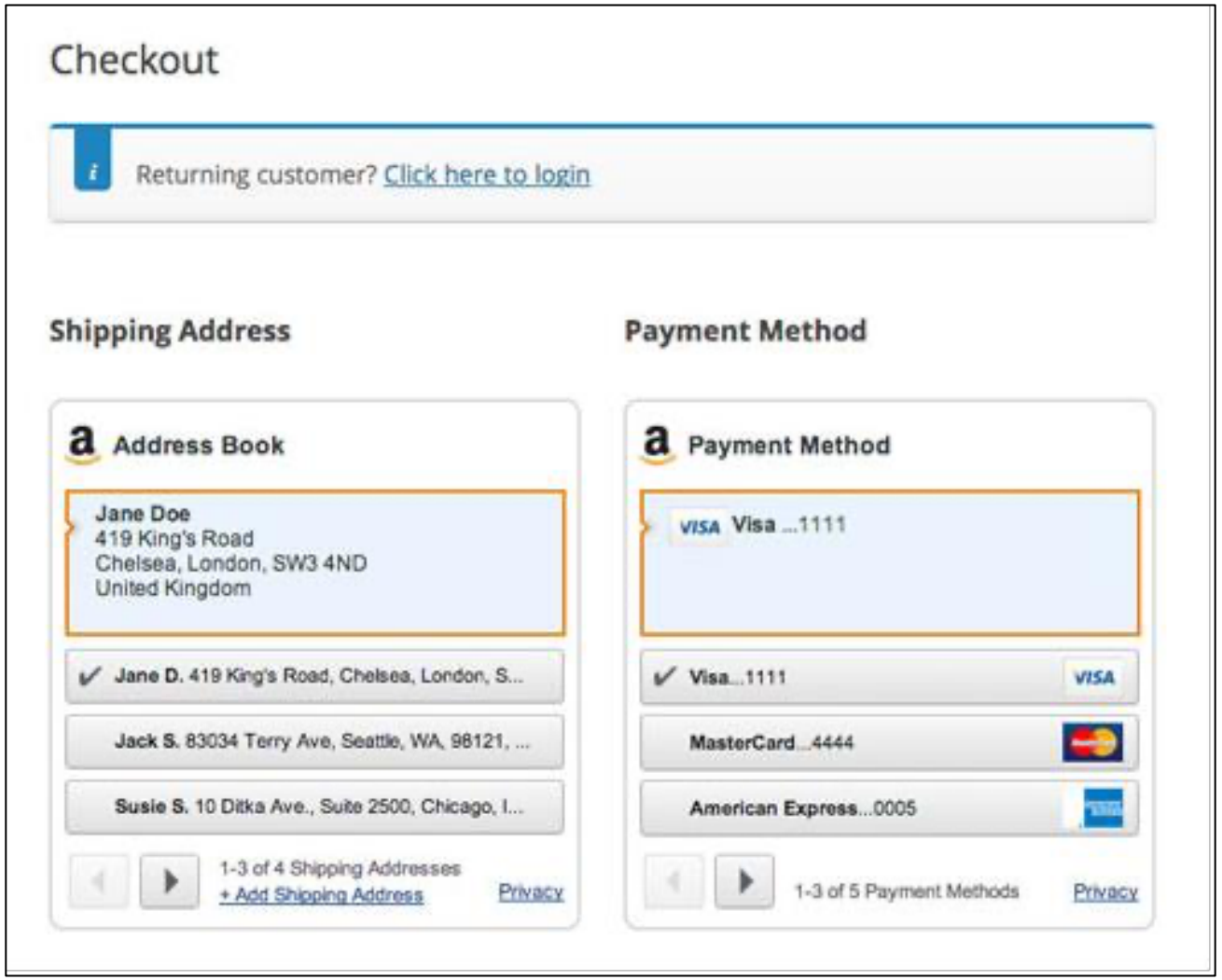



| Guide Word | Meaning |
|---|---|
| NO OR NOT | Complete negation of the design intent |
| MORE | Quantitative increase |
| LESS | Quantitative decrease |
| AS WELL AS | Qualitative modification/increase |
| PART OF | Qualitative modification/decrease |
| REVERSE | Logical opposite of the design intent |
| 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.

# Another Example: Payment Service

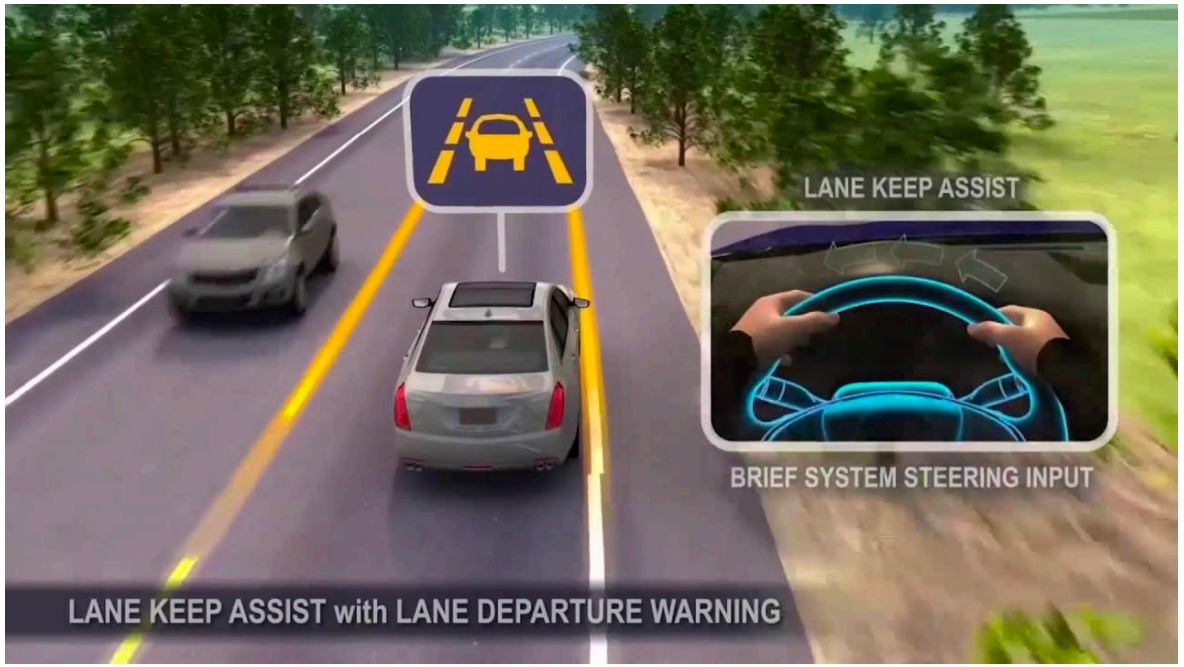
- **Component**: Payment service for a shopping site
- **Input**: Order details, customer billing information
- **Expected behavior**: Charge customer & update order status to “success”
- (Note: This is a post-condition for the component)



# Another Example: Payment Service

| Guide Word | Deviation | Possible Cause(s) | Impact | Design Change |
|---|---|---|---|---|
| NO/NOT | Payment not processed | Missing/incorrect payment info from customer | Customer complaint; possible sale loss | Input validation on the frontend UI |
| MORE | Duplicate payment processed | Accidental double click by customer | Double charge to the customer | Implement idempotency with unique request IDs |
| LATE | No response from payment service | 3rd party payment gateway timeout | Customer quits; possible sale loss | Implement retry; failover to backup gateway |
| INSTEAD | Wrong payment amount deducted | Program bug | Incorrect billing to the customer | Validate amount before processing |
| PART OF | Customer order status still “pending” | DB transaction failure | Customer complaint; possible sale loss | Use atomic transactions, rollback on failure |

# 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 and their impact
- Can be used to derive basic events, to be used for FTA
  - i.e., component faults are possible causes of a TOP event in FTA
- Guidewords are useful, but not perfect; they won't cover every possible component fault
- Like FTA, it requires human judgement & domain knowledge to:
  - Determine whether a particular guideword is relevant
  - Analyze the impact of a component fault on the overall system

# Summary

- Exit ticket!