Safety-critical systems

Simin Nadjm-Tehrani

www.ida.liu.se/~rtslab

Department of Computer & Information Science

Linköping University, Sweden

and

University of Luxembourg

Linköping group @NODES

- Real-time systems laboratory
  - Dependability, Distributed systems, Formal analysis
  - Four PhD students, 5 examined PhDs in 2005-07
  - Recruiting 2 PhD students and a post doc …

- Intelligent information systems laboratory
  - Security, P2P systems, databases & web information systems
  - Five PhD students, 5 examined PhDs in 2005-07

Dependability

- How can we produce computer systems that do their job, and how to *prove or measure* how well they do their jobs?

Engineers: Fool me once, shame on you – fool me twice, shame on me

Software developers: Fool me N times, who cares, this is complex and anyway no one expects software to work...

FT - June 16, 2004

- "If you have a problem with your Volkswagen the likelihood that it was a software problem is very high. Software technology is not something that we as car manufacturers feel comfortable with.”

Bernd Pischetsrieder, chief executive of Volkswagen

October 2005

- “Automaker Toyota announced a recall of 160,000 of its Prius hybrid vehicles following reports of vehicle warning lights illuminating for no reason, and cars' gasoline engines stalling unexpectedly.”

  Wired 05-11-08

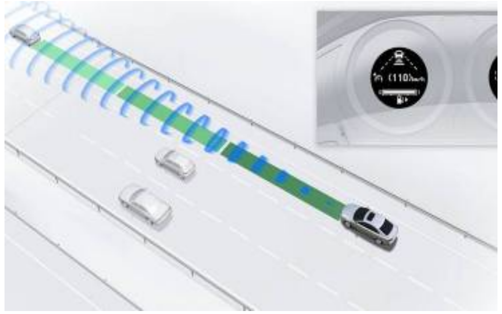
- The problem was found to be an embedded software bug

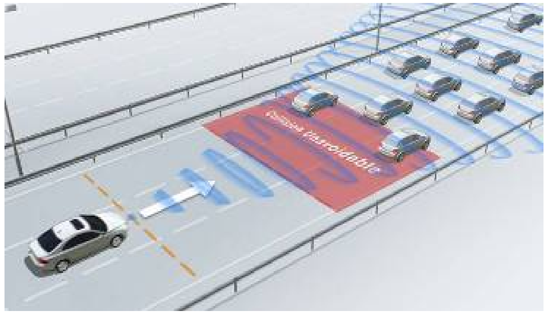
February 2, 2004

- Angel Eck, driving a 1997 Pontiac Sunfire found her car racing at high speed and accelerating on Interstate 70 for 45 minutes, heading toward Denver

- ... with no effect from trying the brakes, shifting to neutral, and shutting off the ignition.

Driver support: Volvo cars





1984	ABS Anti-lock Braking System	2004	Blind Spot Information system (BLIS)
1998	Dynamic Stability and Traction Control (DSTC)	2006	Active Bi-Xenon lights
2002	Roll Stability Control (RSC)	2006	Adaptive Cruise Control (ACC)
2003	Intelligent Driver Information System (IDIS)	2006	Collision warning system with brake support

Early space and avionics

- During 1955, 18 air carrier accidents in the USA (when only 20% of the public was willing to fly!)

- Today’s complexity many times higher

Airbus 380

- Integrated modular avionics (IMA), with safety-critical digital components, e.g.
  - Power-by-wire: complementing the hydraulic powered flight control surfaces
  - Cabin pressure control (implemented with a TTP operated bus)

What is safety?

- IFIP WG 10.4 definition:
  Safety: Absence of catastrophic consequences on the user(s) and the environment

  [Avizienis et al]

- Freedom from exposure to danger, or exemption from hurt, injury or loss

  [Bowen and Stavridou]

Programs are always safe!

- According to these definitions software can only **contribute** to unsafe behaviour

- Safety is a system level property, and can be claimed/assured at system level
- Differs from reliability

- Closely related to **risk**

System safety & Hazards

- Safety: achieved by anticipating accidents, and eliminating their causes

- Hazards are potential causes of accidents

  Conditions in a system which together with other factors in the environment inevitably cause accidents.

Fault to Accident

- Fault
- Error
- Failure
- Hazard
- Accident

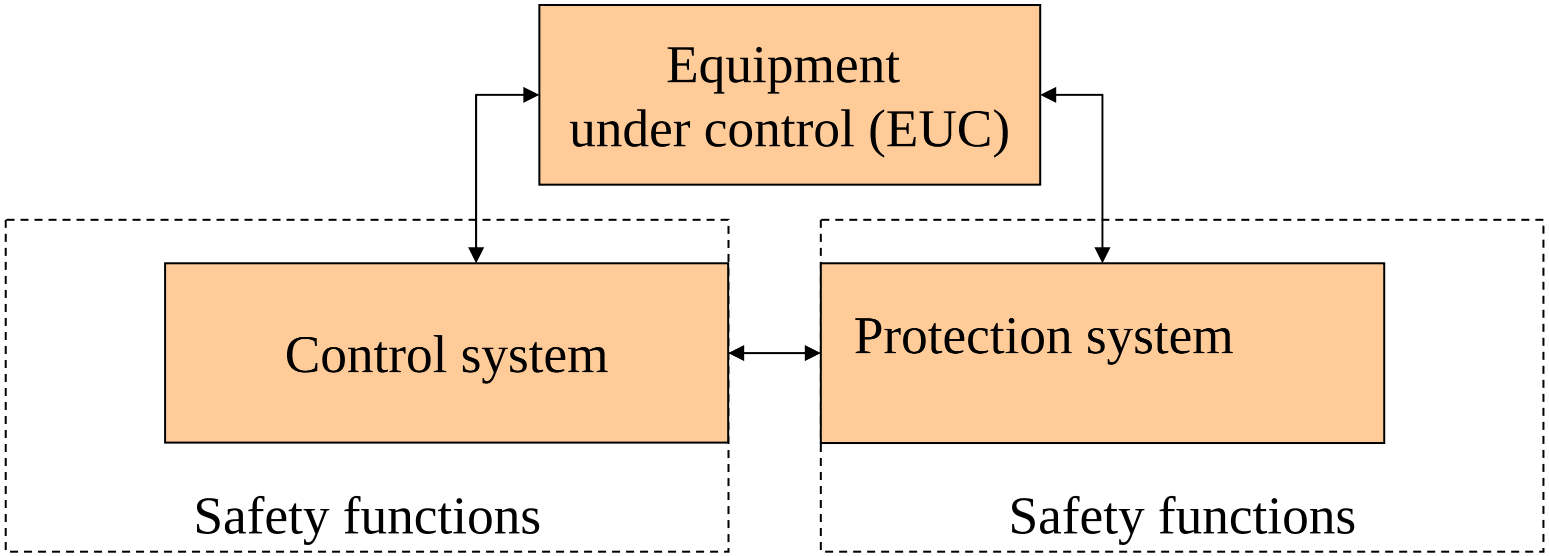
Safety & risk management

- Means anticipating accidents…
- hence anticipating hazards …
- which means quantifying/classifying the potential …
- Must reduce risks which are not tolerable!
- Result: construction of Safety Case

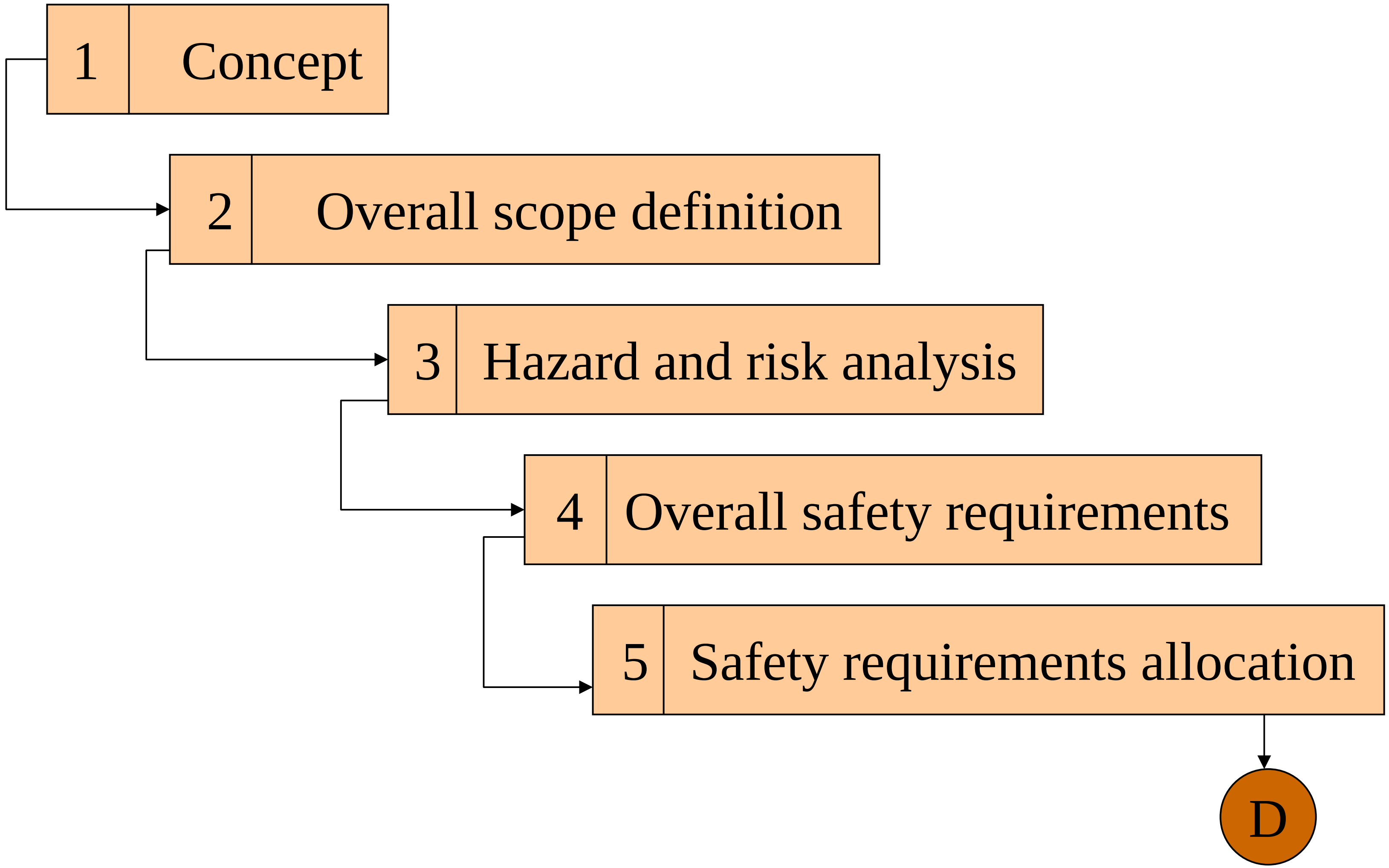
Think negative!

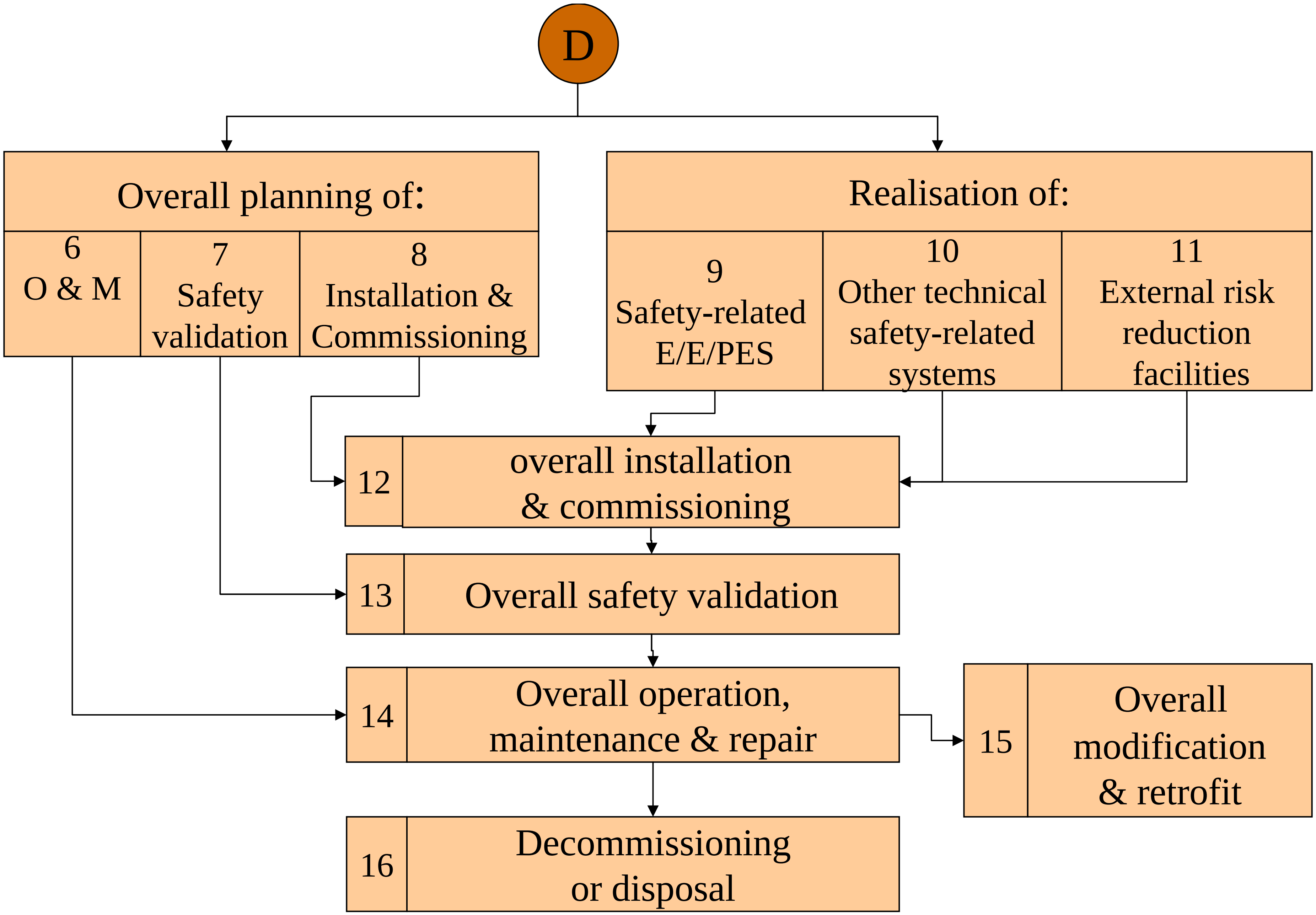
Structure of SC systems

IEC 61508



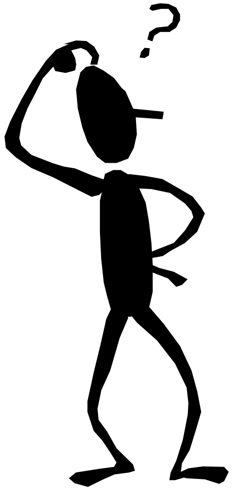
Overall safety lifecycle



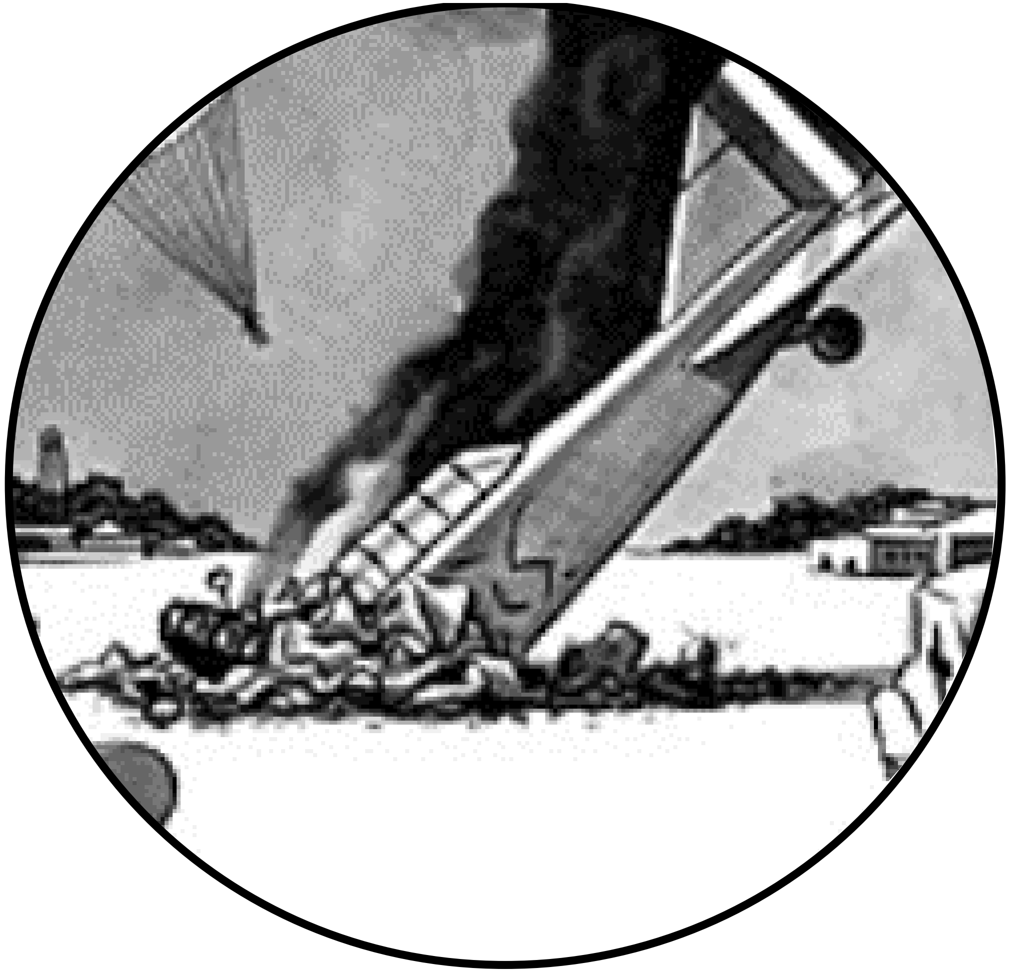


D
Overall planning of:
6
O & M
7
Safety validation
8
Installation & Commissioning
Realisation of:
9
Safety-related E/E/PES
10
Other technical safety-related systems
11
External risk reduction facilities
12
overall installation & commissioning
13
Overall safety validation
14
Overall operation, maintenance & repair
15
Overall modification & retrofit
16
Decommissioning or disposal

But how does this fit in classical (software) systems development process?



Violation of safety



Patterns for safety analysis?

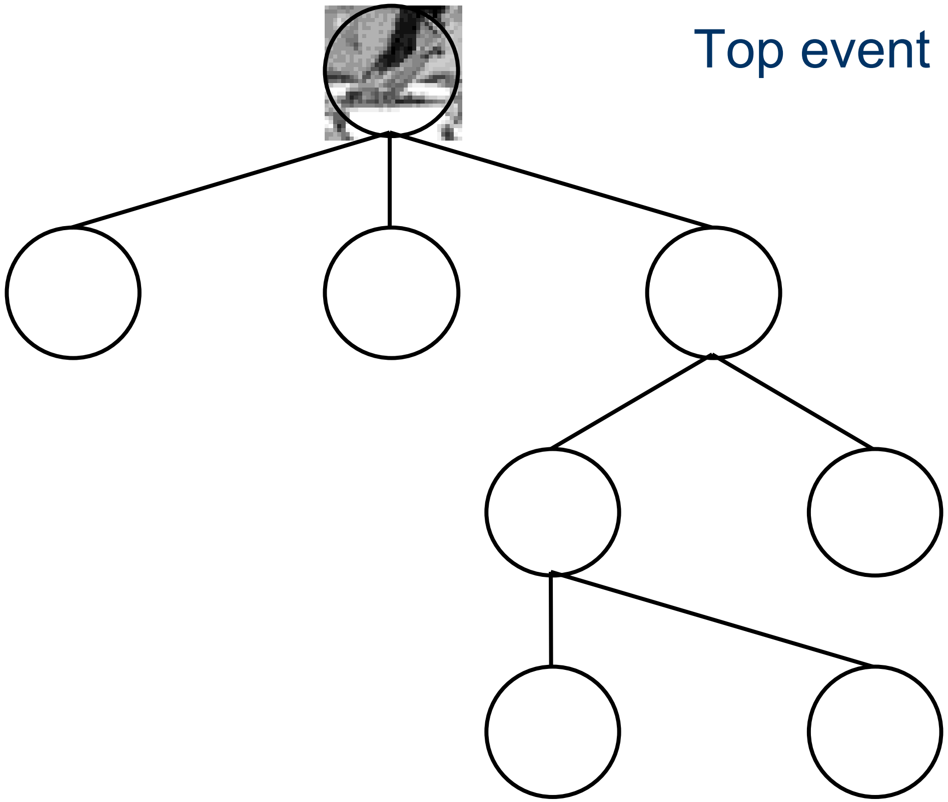
Traditional Safety Analysis

Fault

Tree

Analysis

(FTA)



Traditional Safety analysis

Failure modes and events analysis (FMEA):

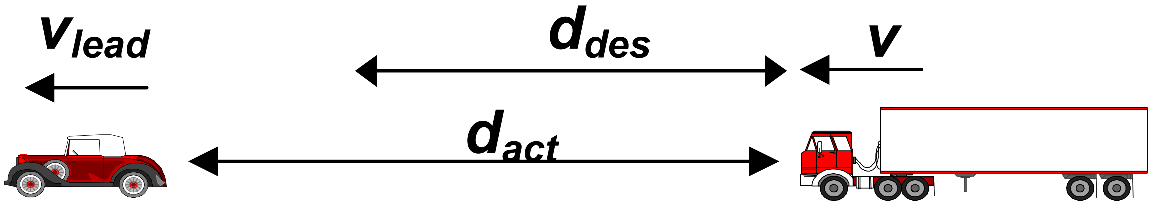
- What are the consequences of some particular component's failure?

Subsystem	Failure Mode	Effects of failure	Cause of failure	Actions	...
Sensor	Value Failure	?	Sensor Malfunction	Duplicate sensors	...
. .	. .	. .	. .	. .	. .

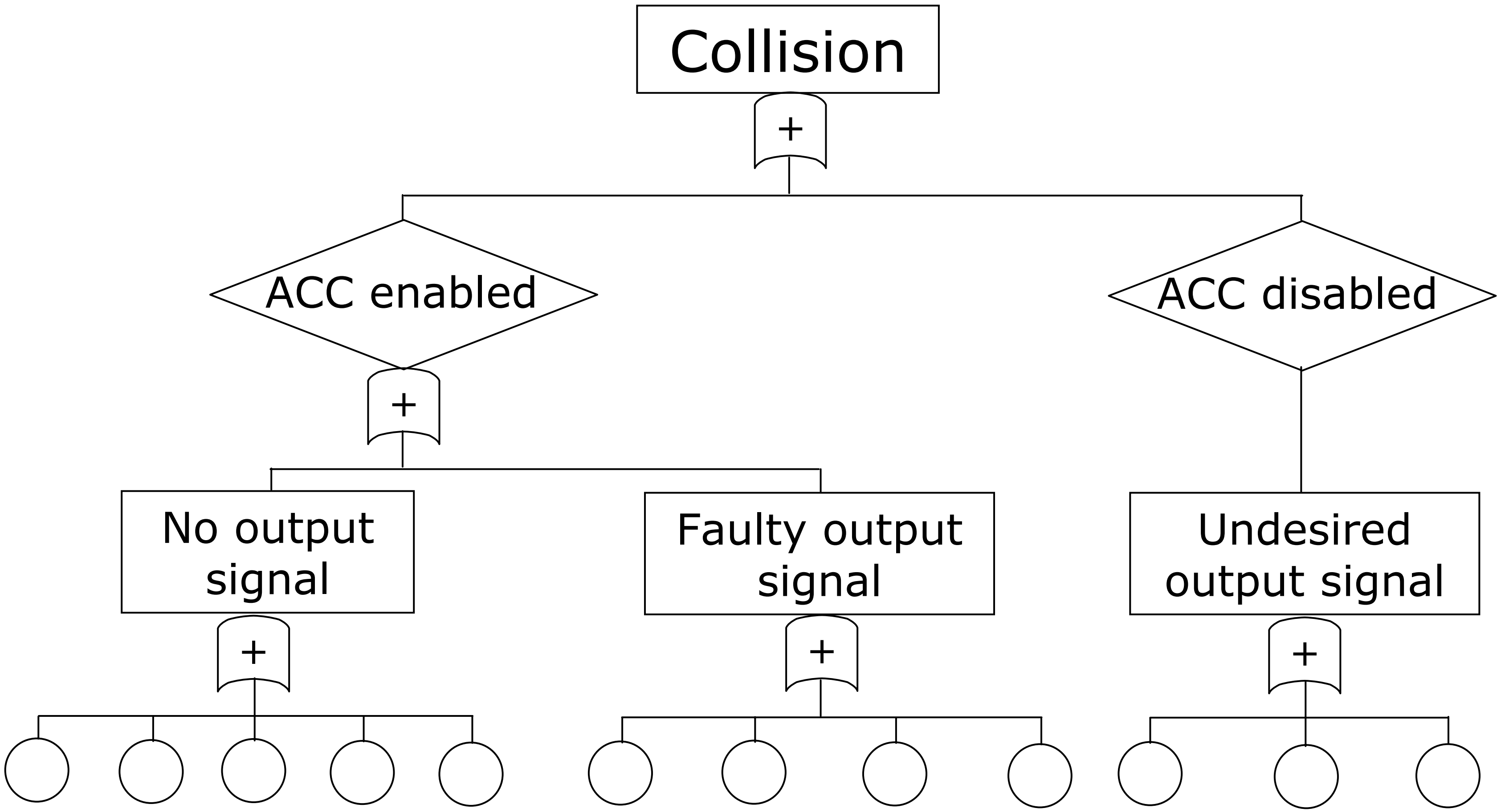
Example

- Adaptive Cruise Controller (ACC)
- Extension to a traditional cruise control
  - adapts vehicles speed to the speed and distance of the vehicle in front

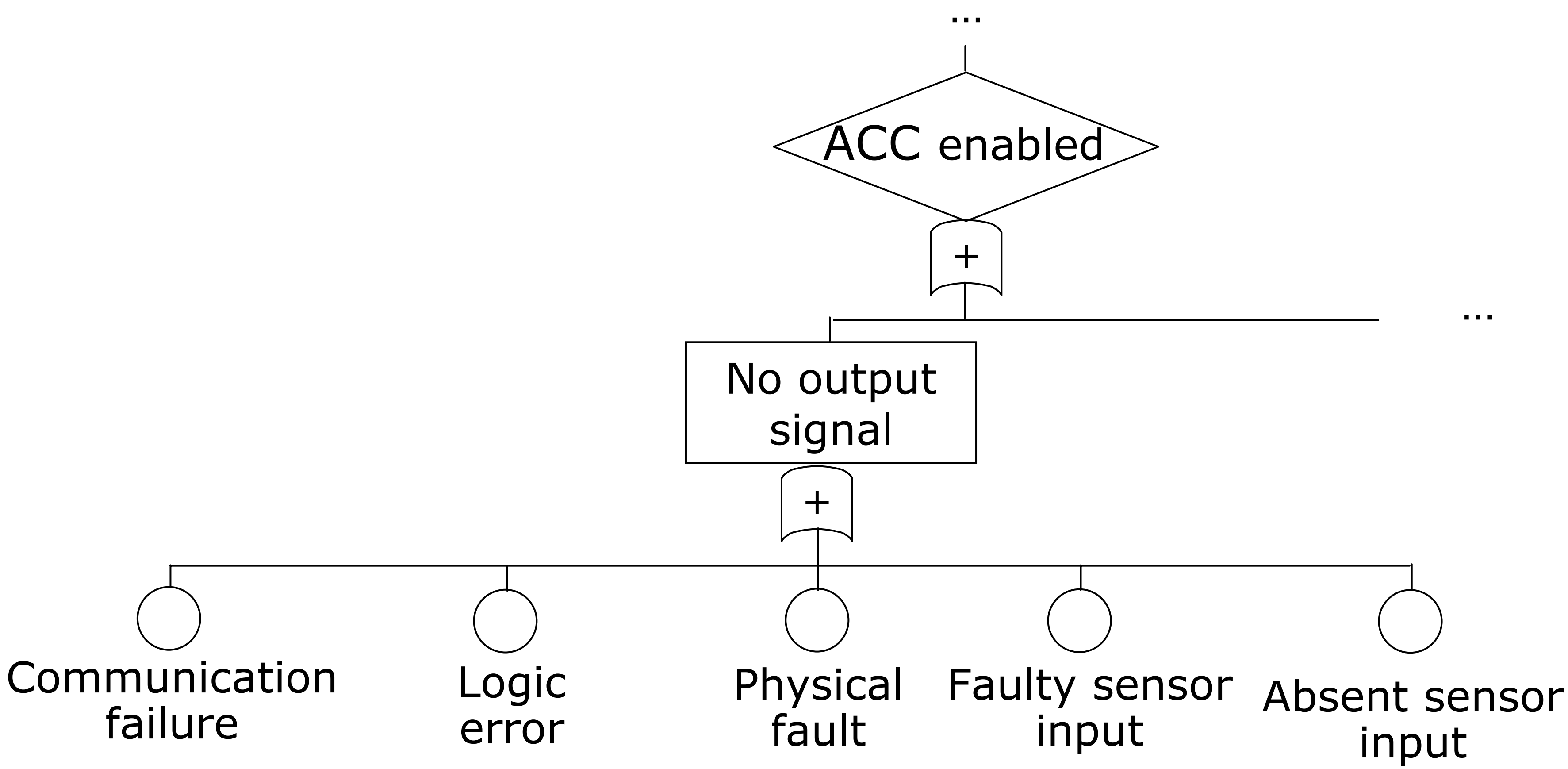
- Identify the hazards and their risks



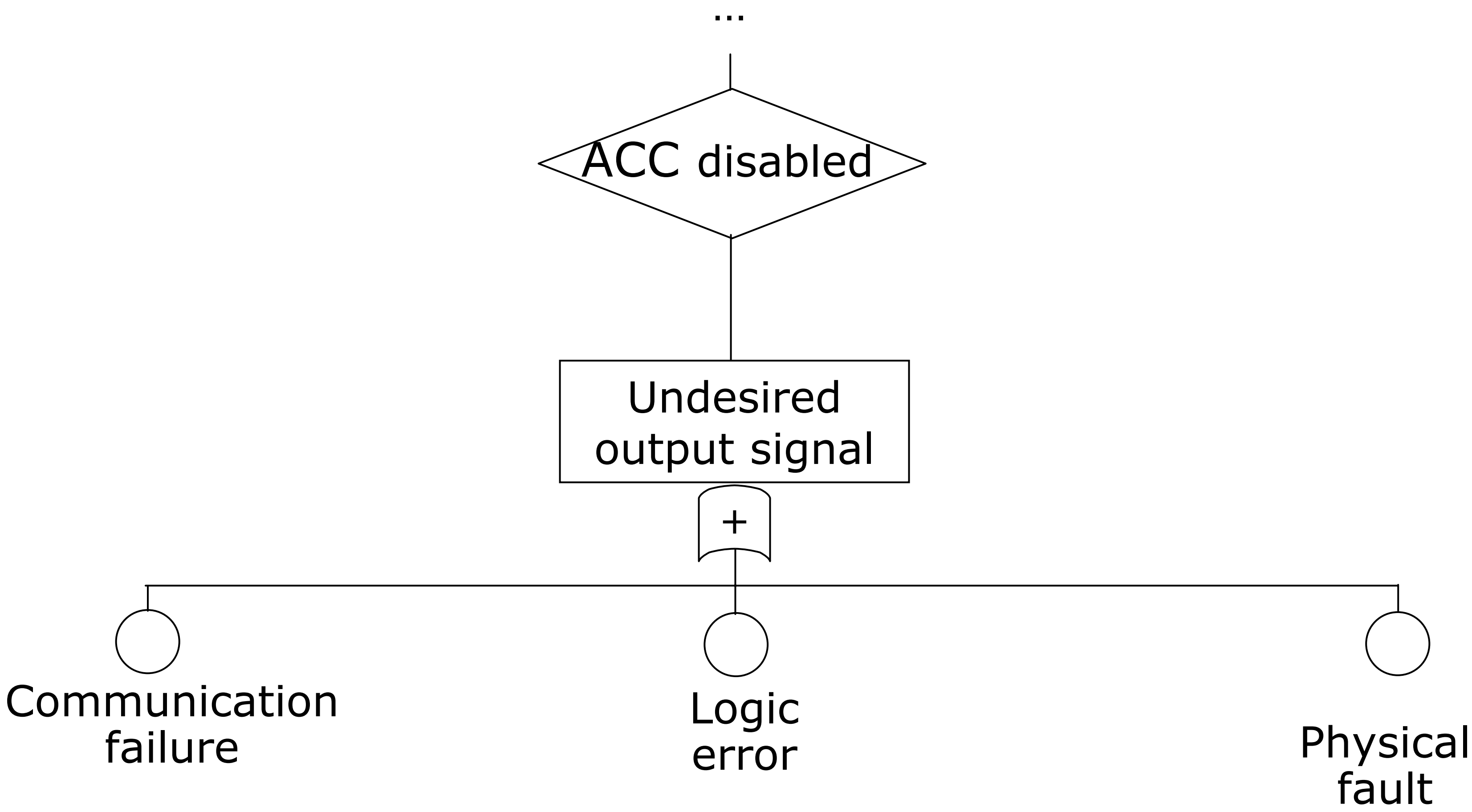
Fault tree analysis



No output signal

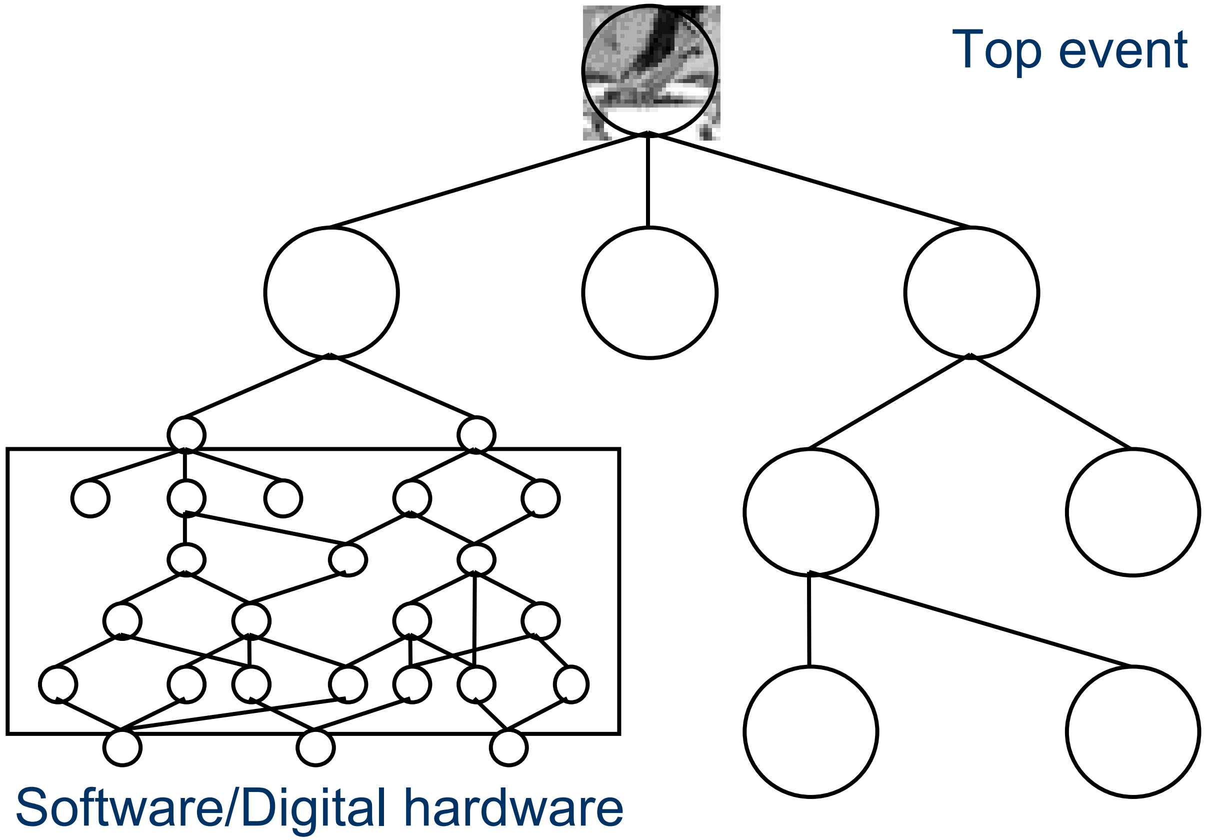


Undesired output signal



Growing complexity

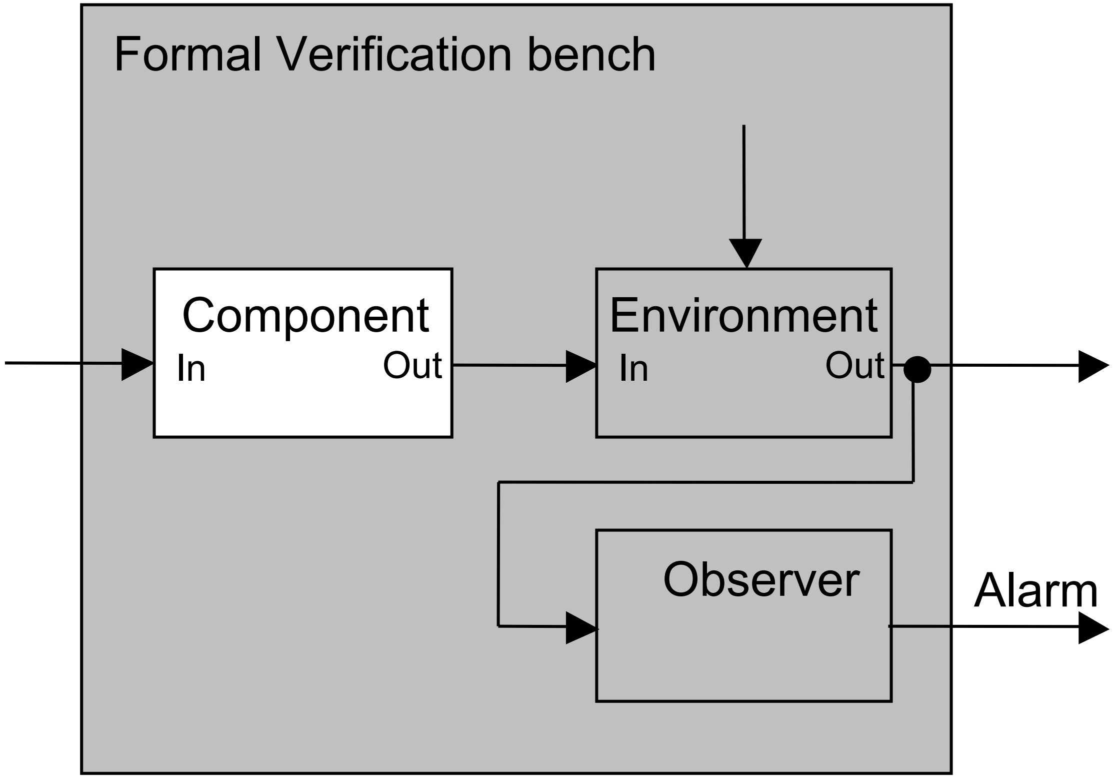
FTA:



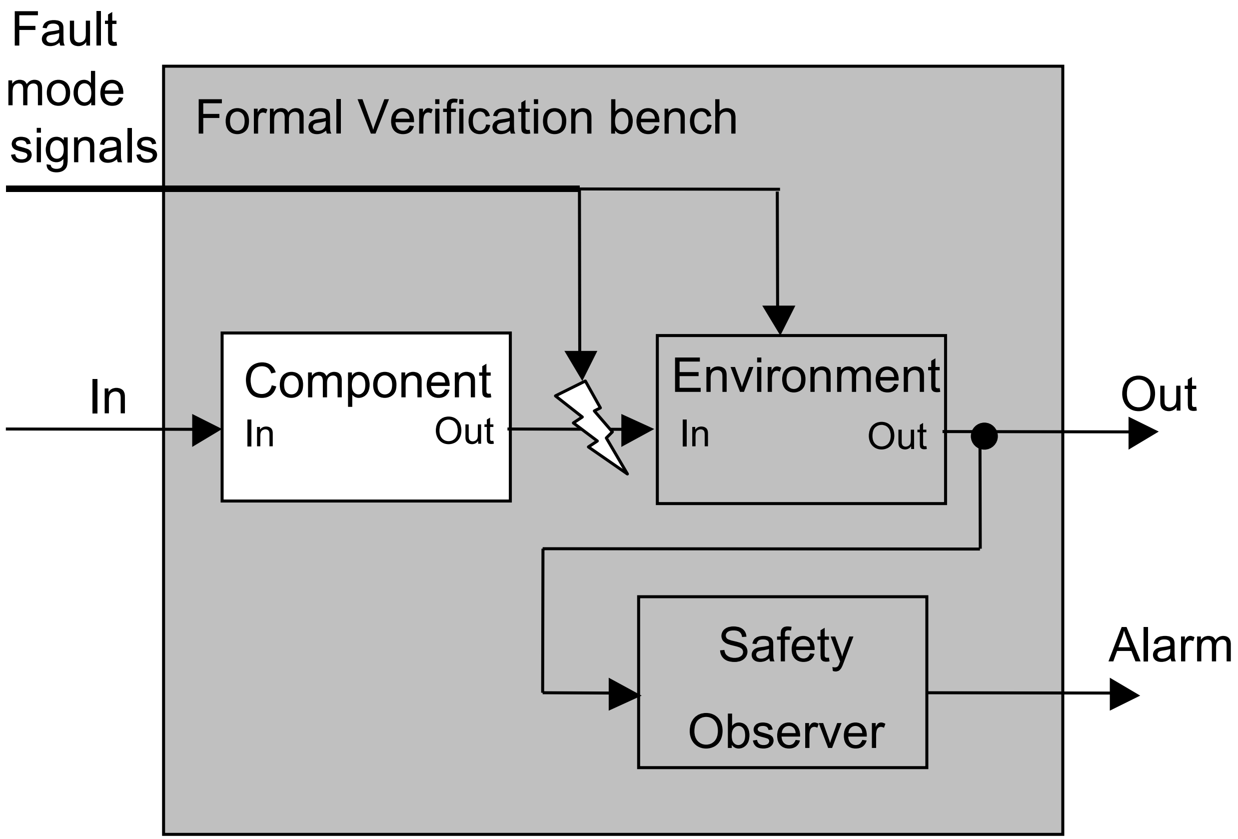
Focus on safety

- Faults that are probable and may cause failures that lead to hazards are in focus
- The system should be shown to avoid hazardous failures even in presence of these faults

Pattern: Functional verification



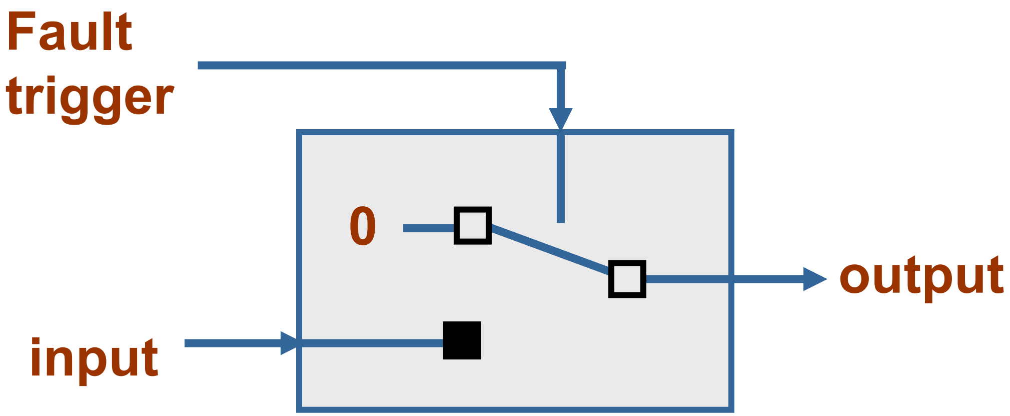
Pattern: Fault mode analysis

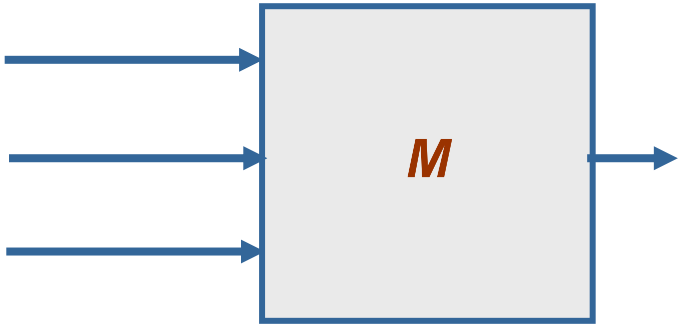


Fault Modelling

- A fault library can be created in design tools
- Fault mode classification:
  - Value faults
  - Omission faults
  - Commission faults

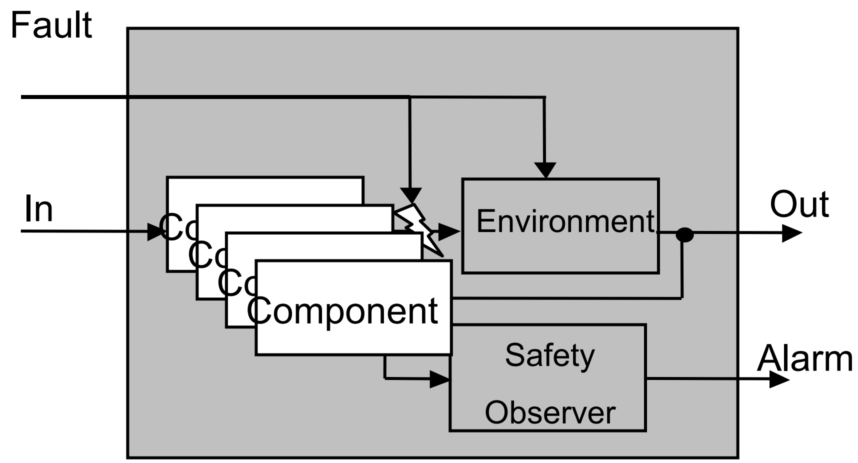
Examples of faults
- Stuck-at
- Bit-flips





Adding components (upgrades)

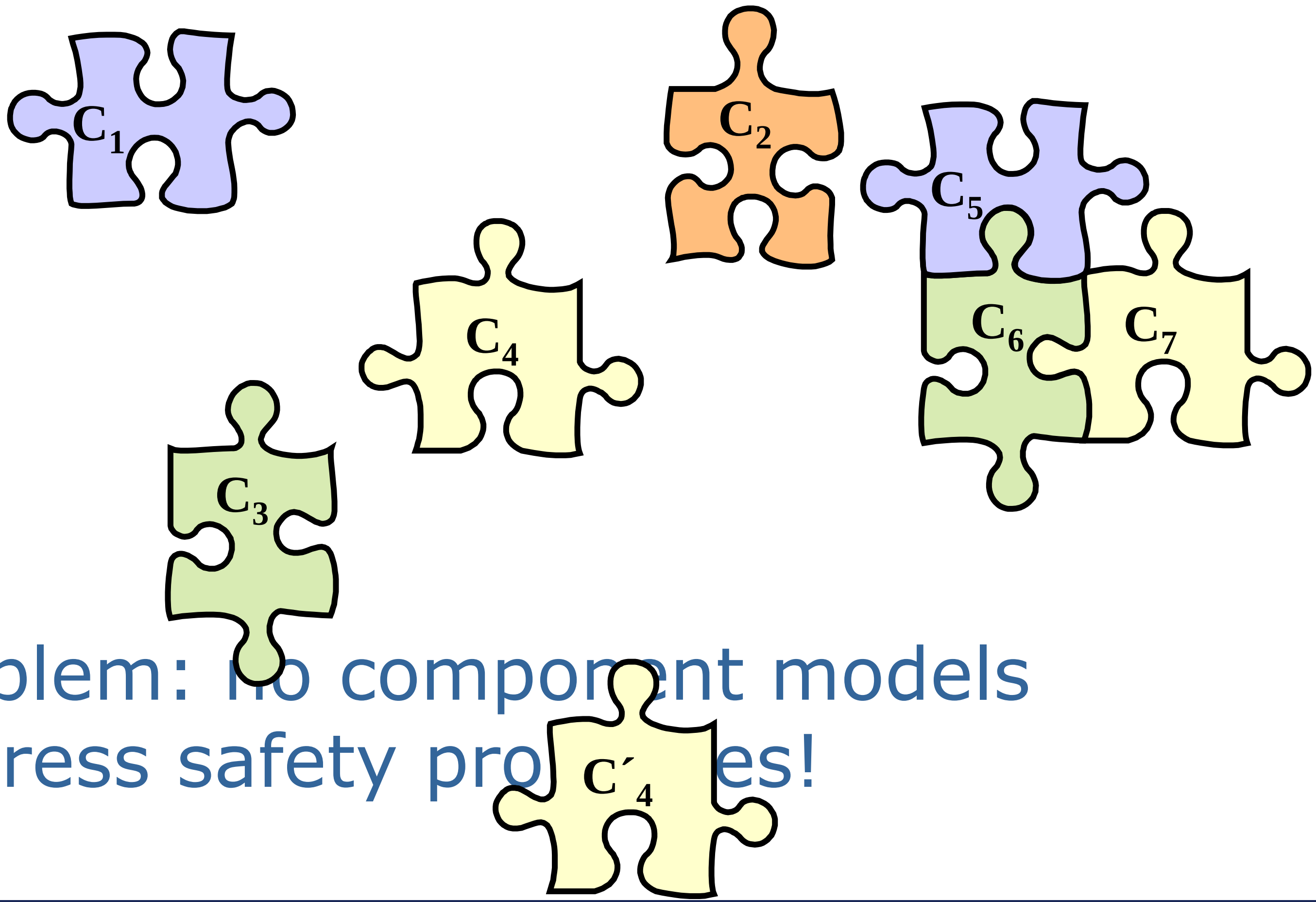
The pattern works if :



- The system is developed in one organisation
- All source code (all models) are available
- Formal analysis of the composition is not prohibitive (size, time)

Component-based Development

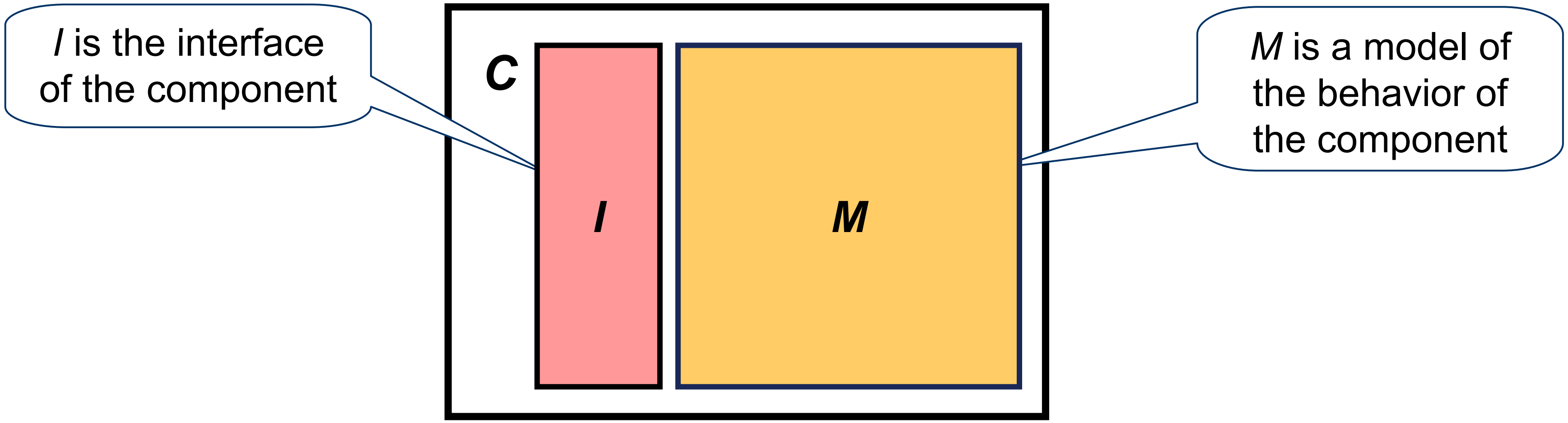
- CBD is an emerging trend in software systems



- Problem: no component models address safety properties!

Components & Interfaces

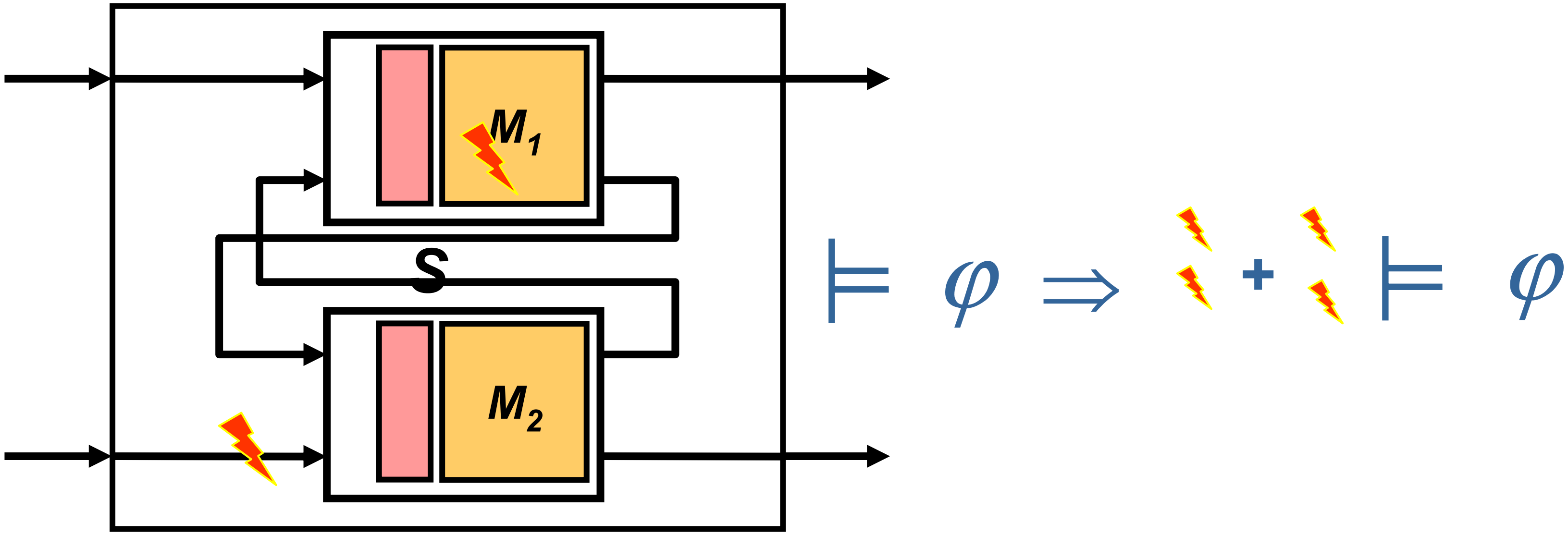
- *Software component Interfaces* provide all information needed for *composition*



- How should the interface look like in order to capture safety?

Safety and CBD

- A safety property φ is typically defined at *system-level*
- Our approach:
  - Interface captures information about behavior of component in presence of faults in the system



ACC example

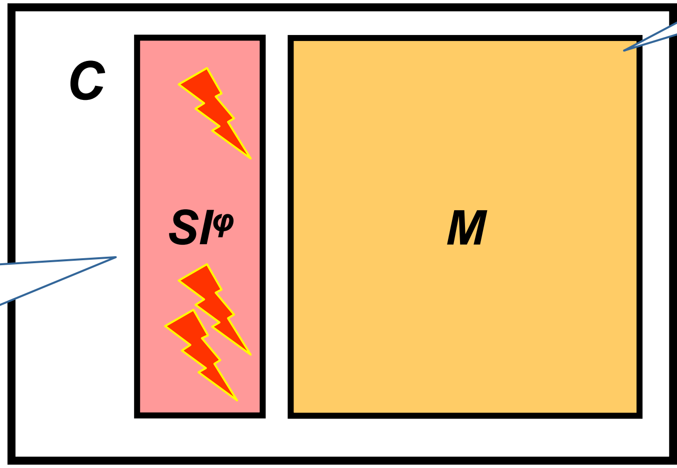
- φ : When the ACC is in ACC-Mode, the speed is higher than 50 km/h and there is a vehicle in front closer than 50 m, the ACC should not accelerate

- Formal definition: $C = \langle SI^{\varphi}, M \rangle$

For a given set of faults

How do single and double faults in the environment of *M* affect the safety property φ?

M is a model of the behavior of the component



- Given a set of faults **F**, a safety property φ, and a model M, the *safety interface* SI^{φ} describes the single and double faults in **F** that M is **resilient** to

Environment Abstraction

- Dilemma with CBD:
  - The fewer assumptions about the environment the more useful the notion of component
  - In order to guarantee something, assumptions must be made
- Solution: include some assumptions about the environment in the safety interface SI^{φ}

$$C = \langle SI^{\varphi}, M \rangle$$

$SI^{\varphi} = \langle E^{\varphi}, \mathtt{single}, \mathtt{double} \rangle$ where

$\mathtt{single} = \langle \langle F_1^s, A_1^s \rangle, \ldots, \langle F_m^s, A_m^s \rangle \rangle$ and

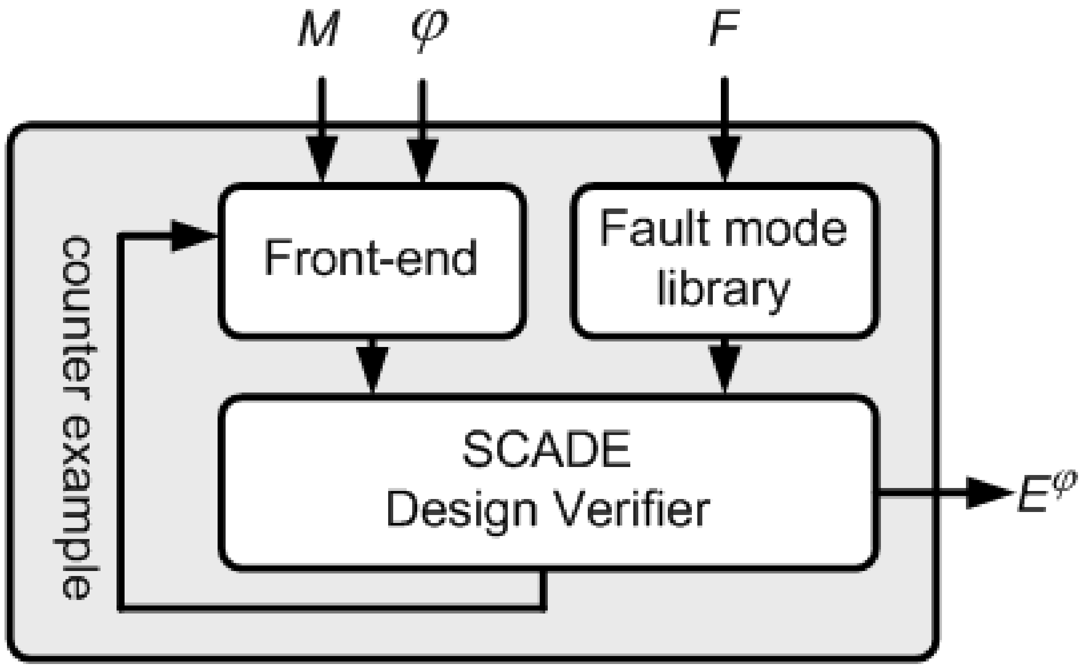
$\mathtt{double} = \langle \langle F_1^d, A_1^d \rangle, \ldots, \langle F_k^d, A_k^d \rangle \rangle$

And …

- Provide help in generating them!

Environment Generation Algorithm

- Support for computing the Interface implemented in SCADE



Environment Abstraction

E^{φ} is the weakest environment in which C will be "safe" with no faults

$$\boldsymbol{E^{\varphi} \,||\, M} \models \varphi$$

$C = \langle SI^{\varphi}, M \rangle$

SI^{φ} = < E^{φ}, single, double>

Environment Abstractions

F_i^d is a pair of faults

F_i^s is a single fault

$M\rangle$

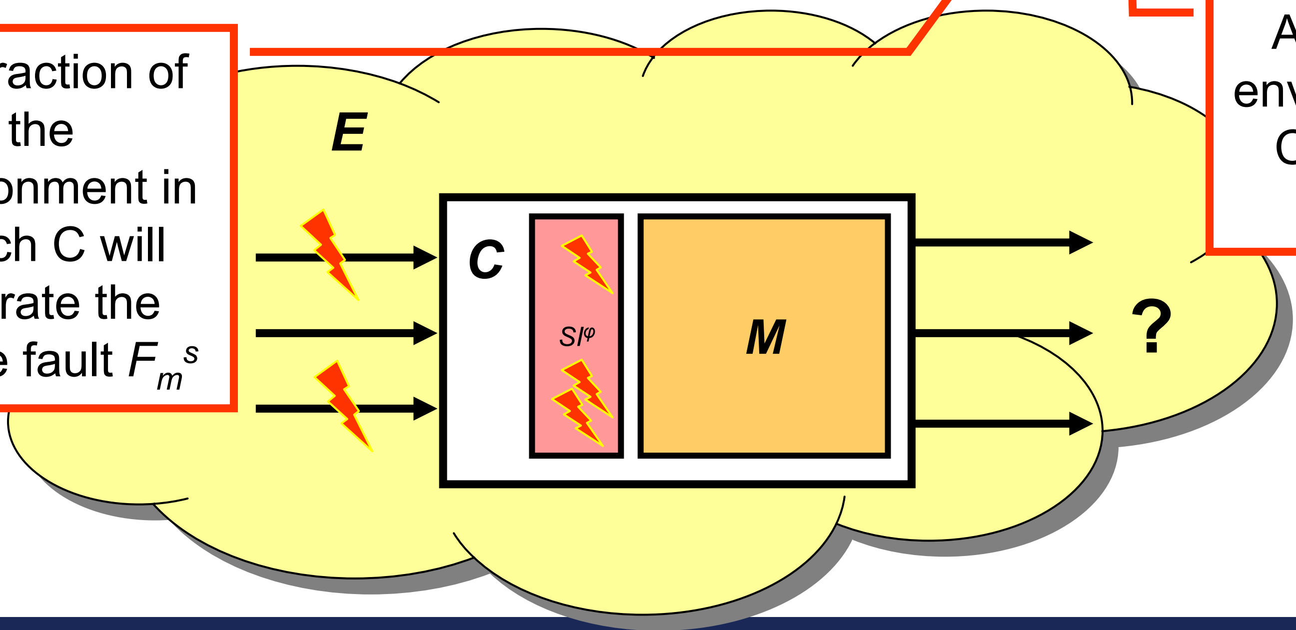
E^φ, single, double> where

single = $\langle \langle F_1^s, A_1^s\rangle, \ldots, \langle F_m^s, A_m^s\rangle \rangle$ and

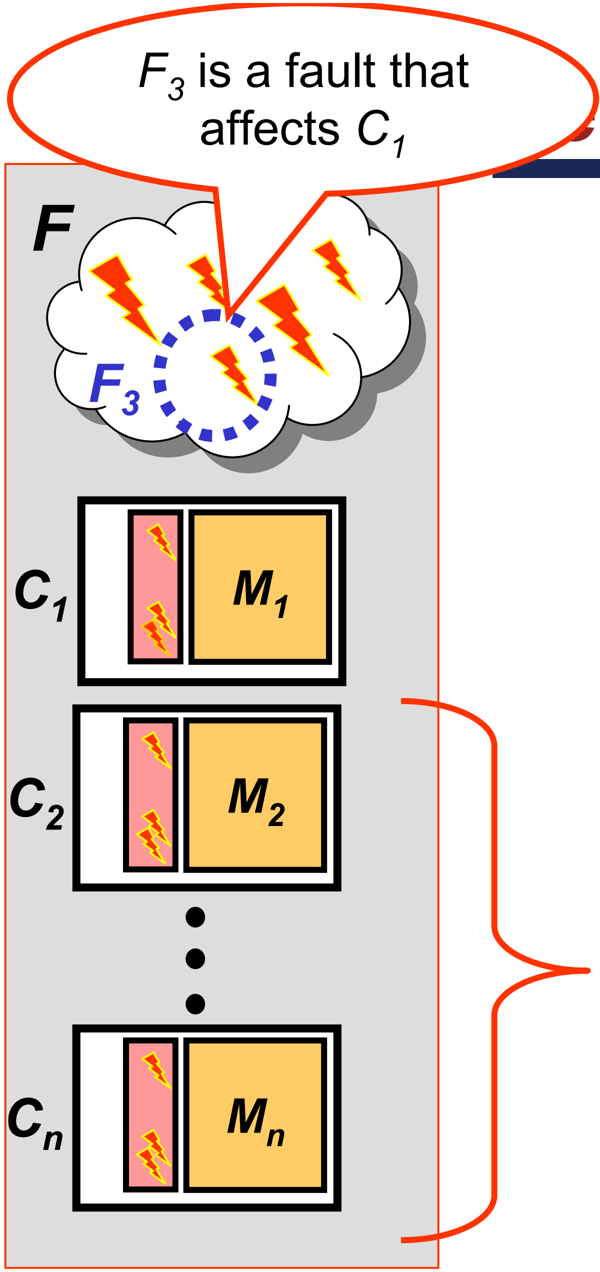
double = $\langle \langle F_1^d, A_1^d\rangle, \ldots, \langle F_k^d, A_k^d\rangle \rangle$

Abstraction of the environment in which C will tolerate the single fault F_m^s

Abstraction of the environment in which C will tolerate the double fault F_k^d



…nt-Based Safety Analysis



$\leq A_3$

$SI_1^{\varphi} = \langle\ E^{\varphi}\ ,$ `single`, `double` $\rangle$

`single` $= \langle F_1^s, A_1^s\rangle, \ldots, \langle F_m^s, A_m^s\rangle$

`double` $= \langle\langle F_1^d, A_1^d\rangle, \ldots, \langle F_k^d, A_k^d\rangle\rangle$

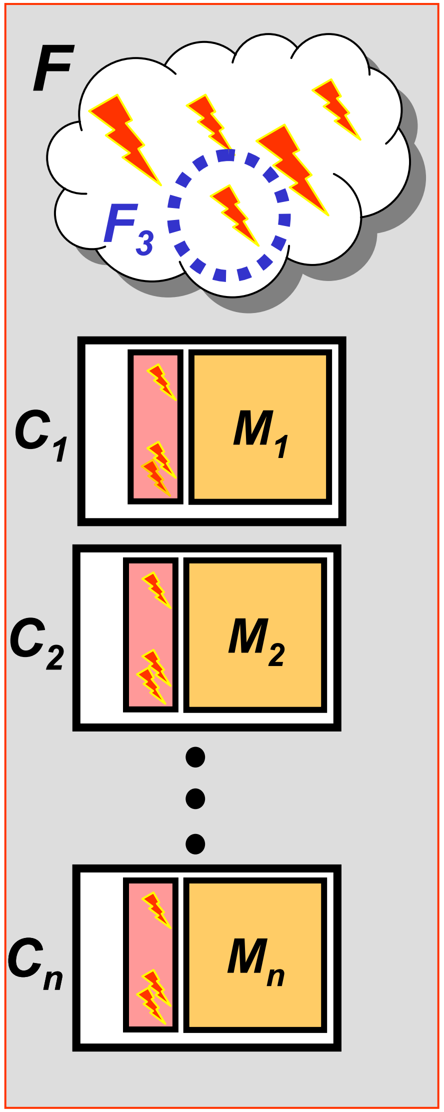
If F_3 appears in `single`, then it suffices to prove that the environment of M_1 is *more constrained* than A_3

- However, infeasible to compose all components and check

  $M_2 \mid\mid \ldots \mid\mid M_m \leq A_3$

- Solution: Assume-Guarantee reasoning

Assume-Guarantee reasoning



To show that environment of M_1 is more specific than A_3, show that:

– C_1 with the fault F_3 at its input still satisfies environment requirement of every other component

For all *j*: $A_3 \circ F_3 \,||\, M_1 \leq E_j^{\varphi}$

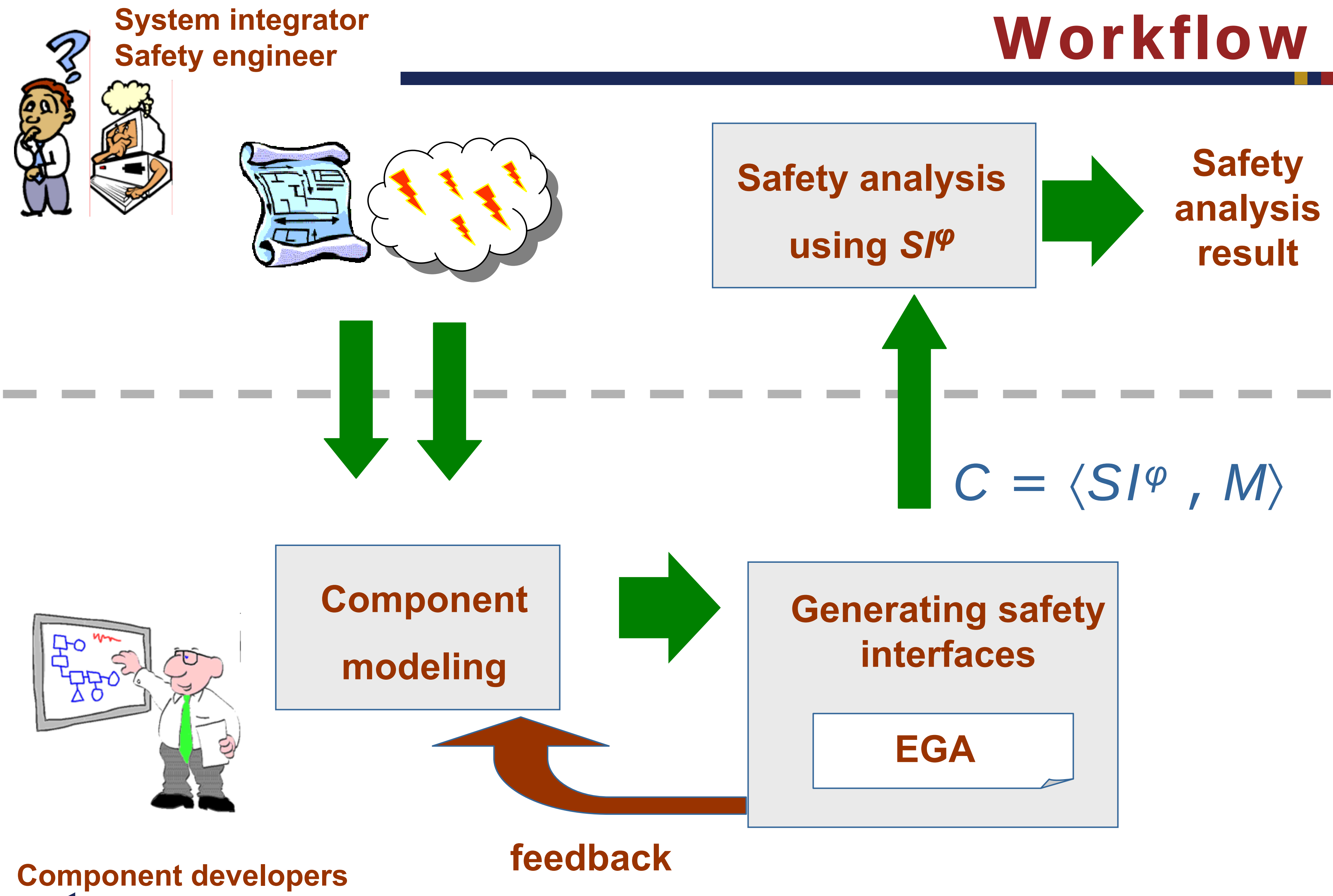
– individual components and their weakest environments are more specific than A_3

For all *j*: $M_j \,||\, E_j^{\varphi} \leq A_3$

Resilience to double faults

- At system level is proved similarly
- Proof rules that take account of:
  - Double faults in one component
  - Two single faults affecting two different components

Workflow


System integrator
Safety engineer
Safety analysis using SI^{φ}
Safety analysis result
$C = \langle SI^{\varphi}, M \rangle$
Component modeling
Generating safety interfaces
EGA
feedback
Component developers

ACC: Safety Analysis Result

- Of the 20 fault modes considered the ACC is resilient to:
  - 8 single faults
  - 2 double faults

- Parts of safety analysis from one fault can be reused later

- However: safety analysis is not finished here!