Stochastic models & analyses for embedded systems evaluation

Sören Kemmann

Fraunhofer
IESE

Content

- Reliability Block diagrams (RBD)
- Fault Trees (FT)
  - Component Fault Trees (CFT)
  - Dynamic Fault Trees (DFT)
- Stochastic Petri nets (SPN)
  - GSPN, DSPN
  - PN analysis
- A comparison of the expressive power of model types

RBD

- Reliability Block Diagrams

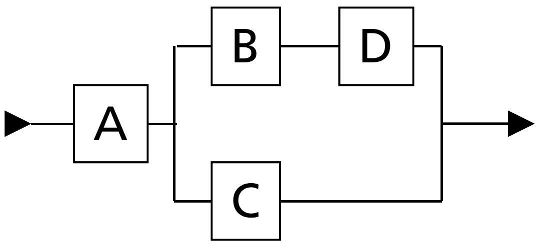
Reliability Block Diagrams (RBD)

- “Which elements of the system may fail without causing system failure?”

- Models necessary components for system functions
- Compare Fault Trees:
  - Fault Tree: Which basic events are necessary for a given failure?
  - RBDs: Which components must be available for correct operation

Reliability Block Diagrams (RBD)

- Example:

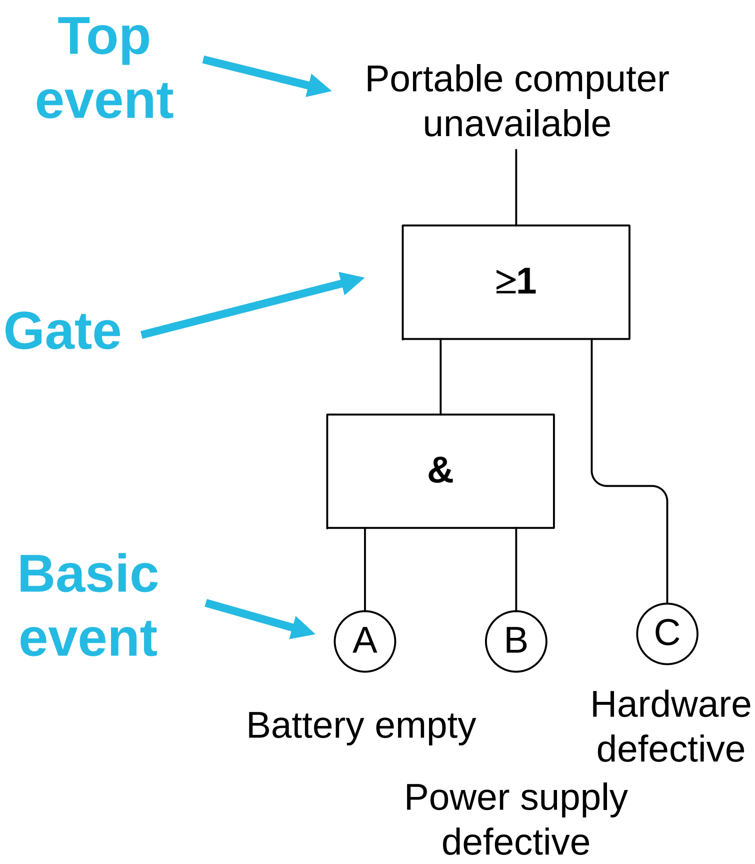


- Correct system operation is given when there is a path from one side to the other
- The system works if A, B and D are available or A and C are available.
- RBD allows easy identification of A as single point of failure.

FT

- Fault Trees

Fault trees



Analysis method for dependability properties

Recursive, deductive decomposition of causes for a given hazard or failure in the form of a DAG

- Root (top event) = hazard/failure
- Leaves (basic events) = elementary causes
- Logical gates (And, Or, …) explain interaction of causes

Fault tree analysis

Use

- Search for all relevant causes for hazards and failures

Qualitative analysis

- Listing all combinations of basic events that are necessary and sufficient to cause a top event
- Search for **single points of failure** (with minimal cut sets, MCS)

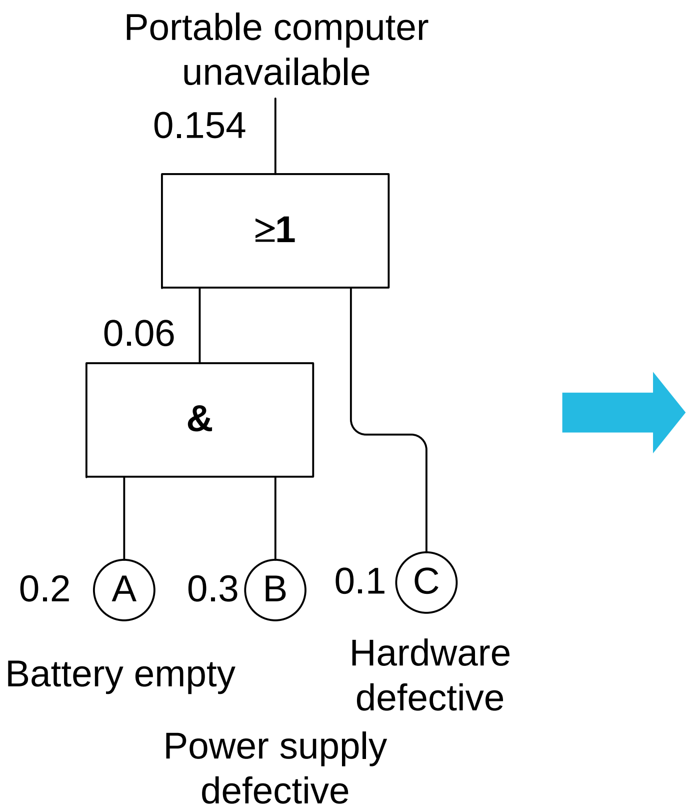
Quantitative Analysis

- Calculation of hazard or failure probabilities from given probabilities for elementary causes

Other measures

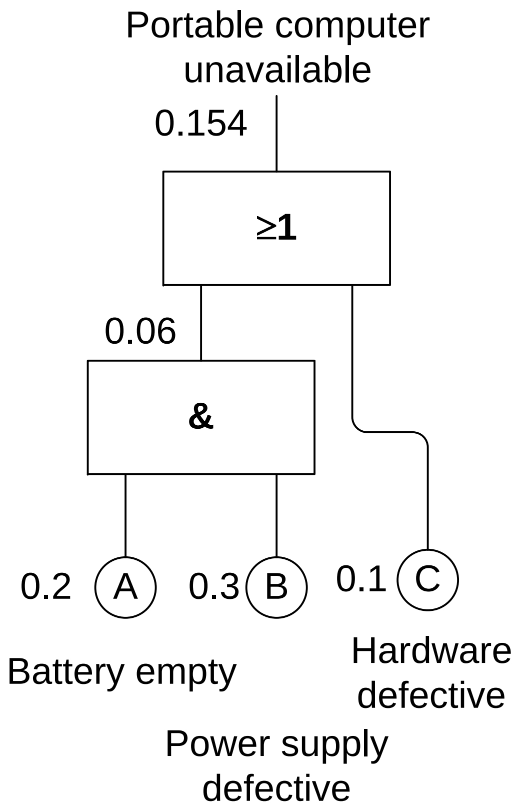
- Mean time to failure (MTTF)
- Influence/importance measures

Qualitative FTA



- Determine MCS
  - Find minterms/implicants
    - ABC, AB~C, A~BC, ~ABC, ~A~BC
  - Remove negated variables
    - ABC, AB, AC, BC, C
  - Minimise
    - **{A, B}, {C}**

Quantitative FTA



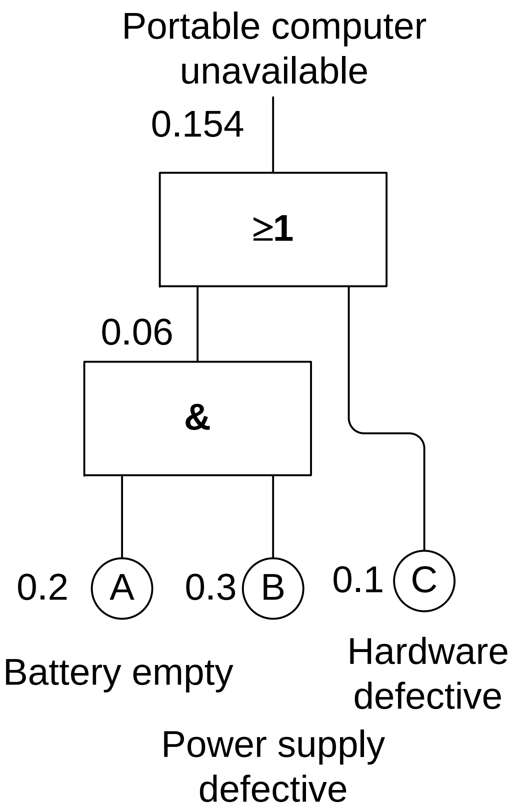
- Apply gate formulae bottom up
- Result when reaching the top event

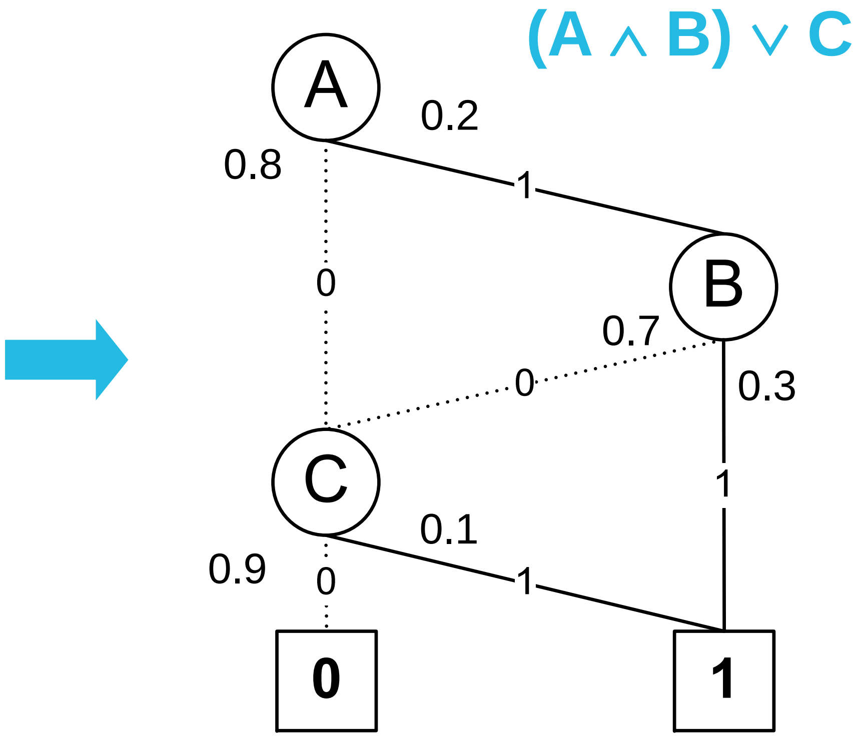
Bottom-up calculation is inefficient for large FT. There are two main alternatives…

- Minimal cut set algorithm
- BDD-based algorithm

(BDD = binary decision diagram)

Top event probability calculation – BDD method

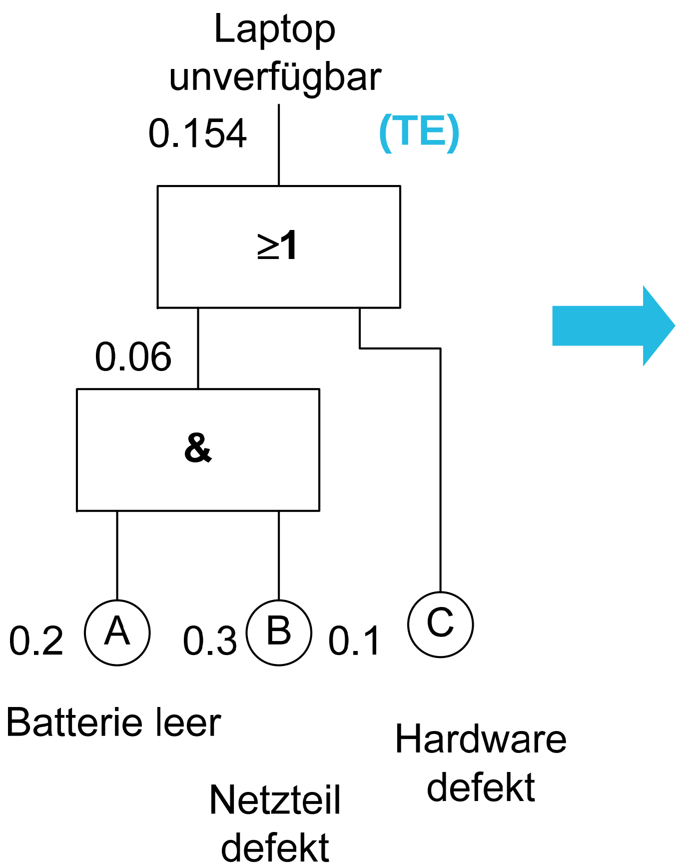




P(TE) = AB + A~BC + ~AC = 0.154

P(TE) = 0.06 + 0.014 + 0.08 = 0.154

Top event probability calculation – MCS method



MCS = {{A, B}, {C}}

Calculation of top event probability as sum of MCS probabilities

P(A)*P(B) = 0.06

P(C) = 0.1

Σ = 0.16 = P(TE)

BDD method: P(TE) = 0.154

- MCS method yields approximation

Deficiencies of conventional fault trees

No compositionality

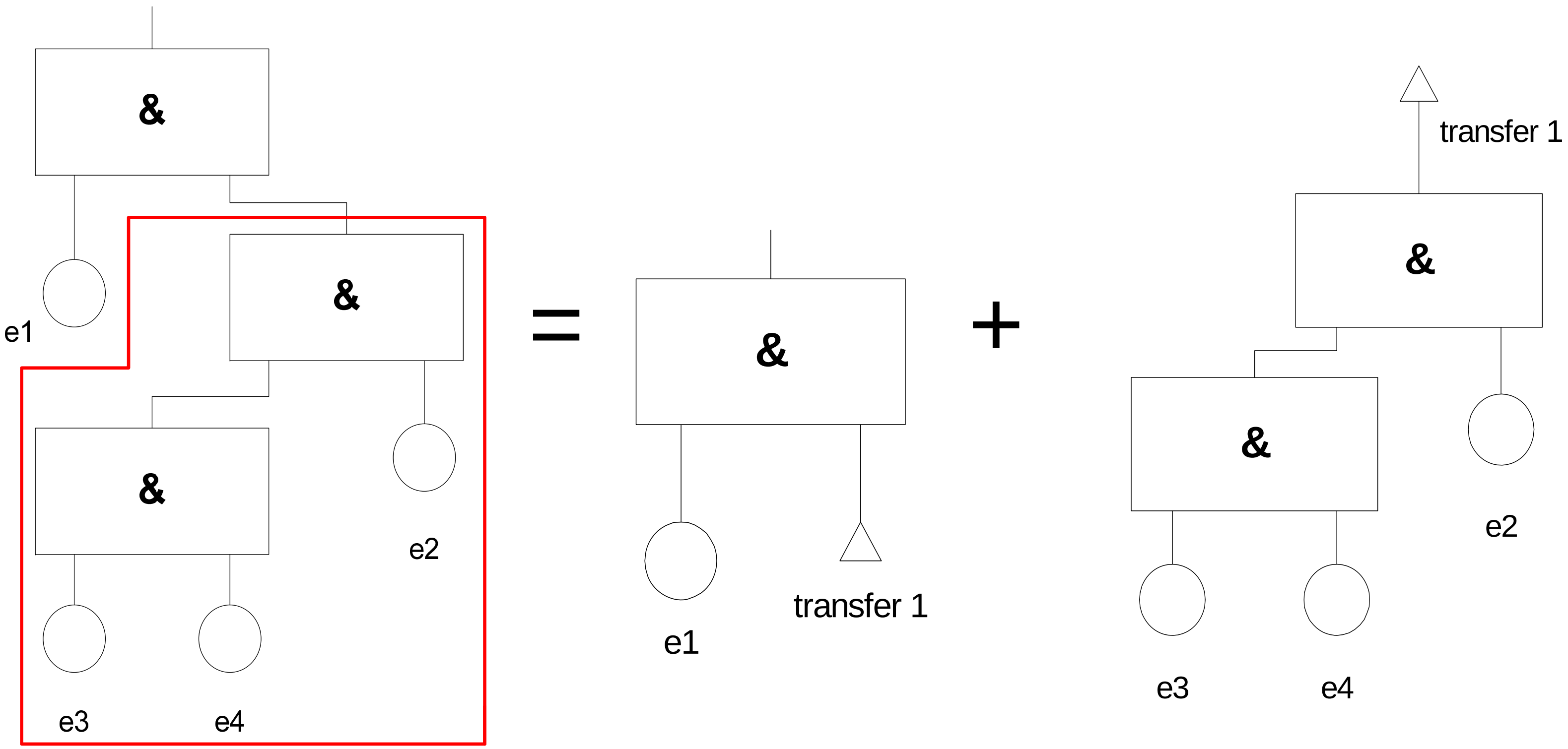
- Technical and software(-controlled) systems are made of components.
- Software design models are often compositional → lack of integration.

No integration with other (aspects of) software/embedded systems (ES) design models, such as statecharts, Matlab/Simulink models etc.

CFT

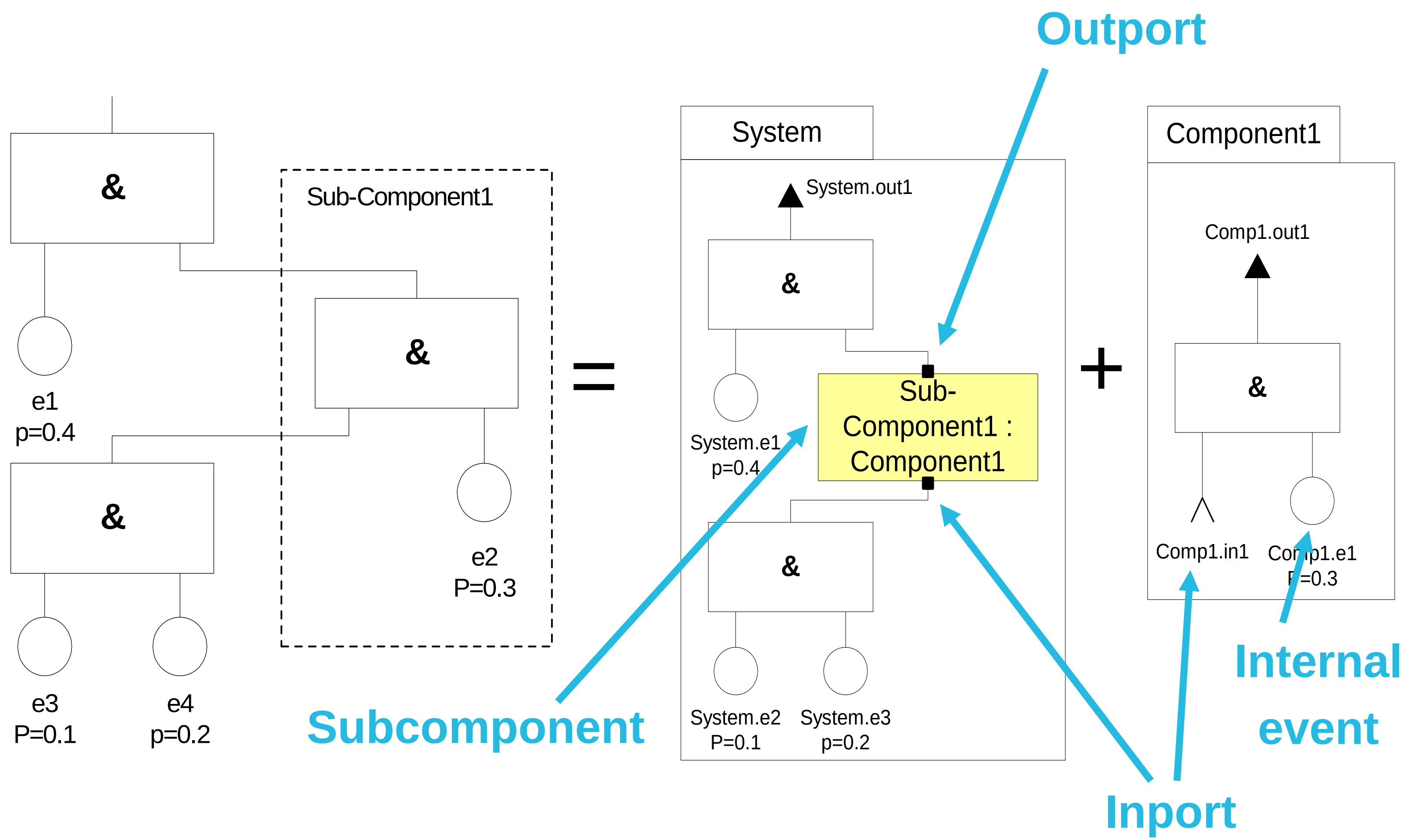
- Component fault trees

Traditional FT decomposition by modules



Traditionally, “modules” are independent subtrees.

Component fault trees

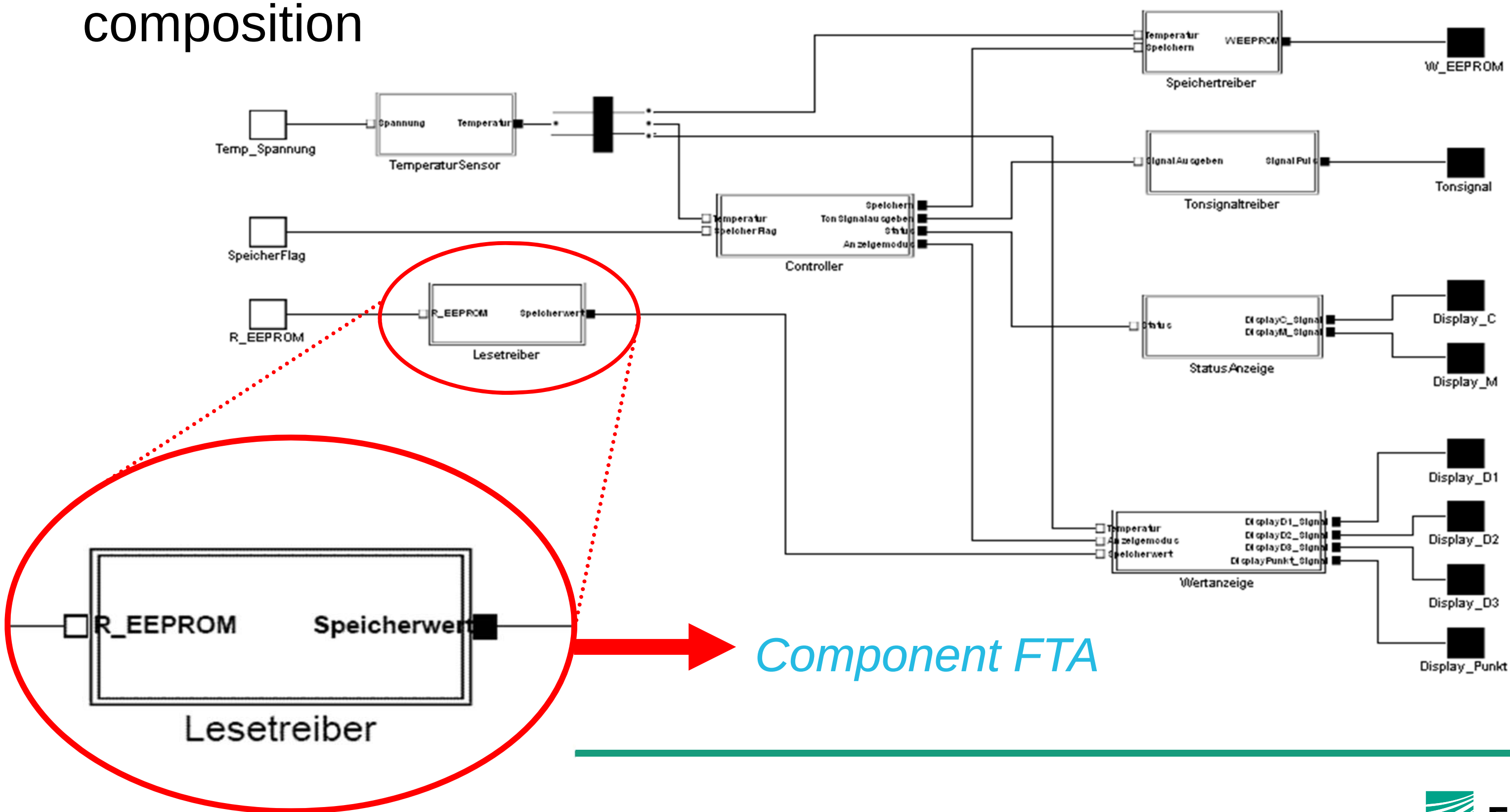


CFT component corresponds to technical component.
Components have specification/realisation with in- and outports.

Component fault trees

What is this good for?

- Composition enables integration of failure with design/architecture models.
- Example: signal flow graph can be used for automatic CFT composition



Fraunhofer IESE

DFT

- Dynamic fault trees

Dynamic fault trees (DFT)

- Problem
  - FTA cannot model the order in which components fail

- Solution
  - Dynamic fault trees (DFT) extend FTA to allow analysis of computer-based systems characterised by
    - Spares (cold, warm, pooled)
    - Functional and sequence dependences
    - Imperfect coverage and other common-cause failures

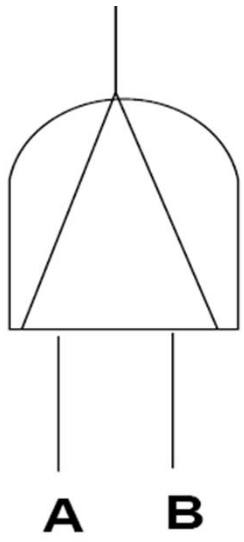
Dynamic fault trees (DFT)

- DFT has constructs (gates) for modelling
  - Sequence dependences (priority-And)
  - Functional dependences
  - Spares (hot, warm, cold)

- DFT model is divided into independent modules that are solved separately

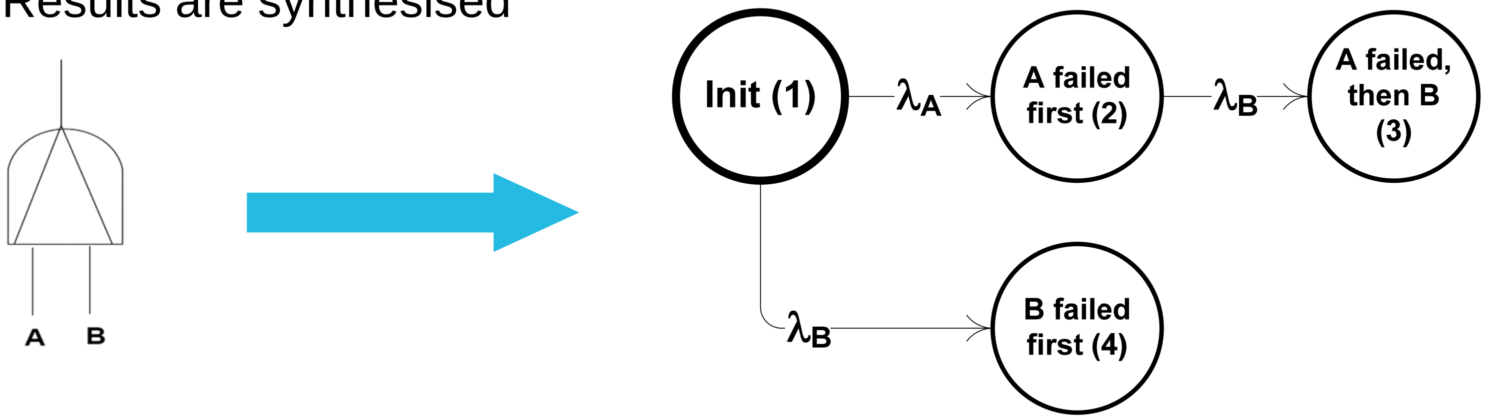
- Modules are classified as
  - static (containing only traditional gates) or
  - dynamic (containing at least one dynamic gate)

Priority-And
(A before B)



Dynamic fault trees (DFT)

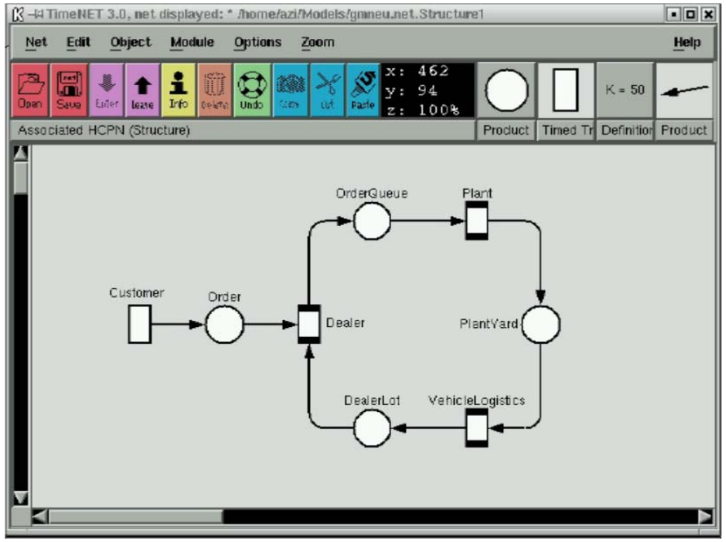
- Separate modules are solved using most appropriate means
  - Markov chain for dynamic modules
  - BDD for static modules
  - Results are synthesised



- Pros and cons
  - + Easier to use than Markov model directly
  - - State space largeness (can be exponential in number of basic events)

Petri nets

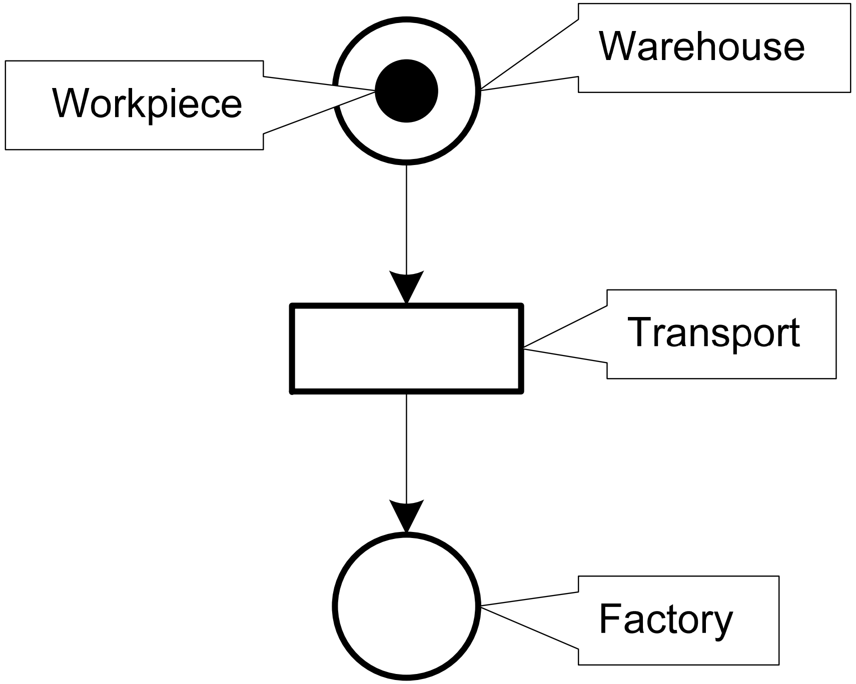
Petri nets



- Modelling of system behaviour
  - With focus on concurrency
- Large number of varieties
- Formal description and graphical representation
- Based on ideas of Carl Adam Petri (Dissertation 1962)

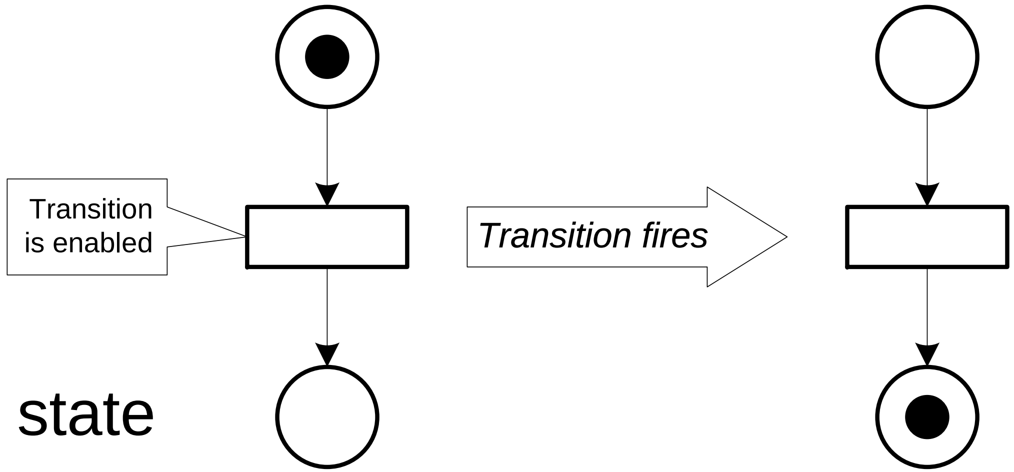
Petri nets

- Tokens
  - Entities
- Places
  - Location/state of entities
- Transitions
  - Activities
- Marking
  - System state



Petri nets

- Ongoing activity
  - Transition is enabled
  - All preconditions of transition have to be fulfilled
- Activity is finished
  - Transition fires
  - Firing is atomic
  - New marking/system state

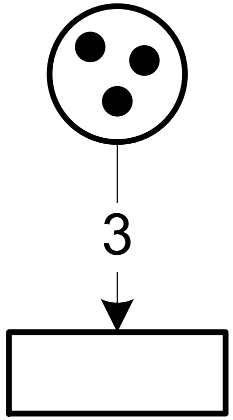


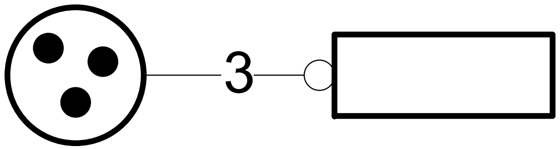
Petri nets

- Transition types
  - Immediate
    - Takes no time between enabling and firing
    - Can be prioritised for case of conflict
  - Timed
    - Takes time between enabling and firing
      - Exponentially timed
      - Deterministically timed

Petri nets

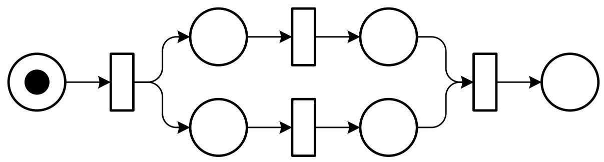
- Arc weights
  - For flow arcs
    - Minimum number of tokens on place to enable transition
    - Number of tokens consumed/produced
  - For inhibitor arcs
    - Min. number of tokens needed on place to disable transition
- Arc weights or priorities make PN Turing complete!



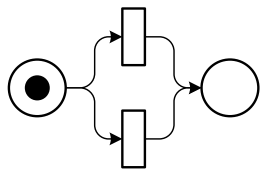


Petri nets – typical structures

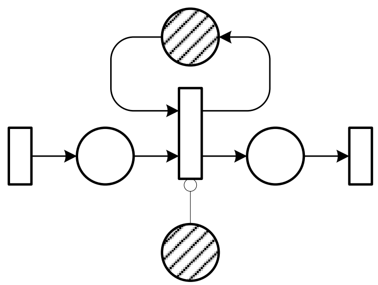
- Concurrency



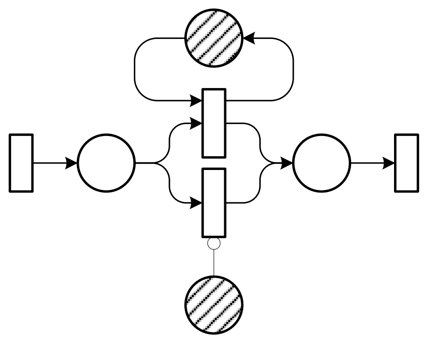
- Conflict



- Firing constraints



Conjunction



Disjunction

Petri Nets: Formal Definition

A marked Petri net is formally defined by the following tuple

$$PN = (P, T, F, W, M_0)$$

where

$P = (p_1, p_2, \ldots\ p_P)$ is the set of places

$T = (t_1, t_2, \ldots\ t_T)$ is the set of transitions

$F \subseteq (P \times T) \cup (T \times P)$ is the set of arcs

$W: F \rightarrow (1, 2, \ldots)$ is a weight function

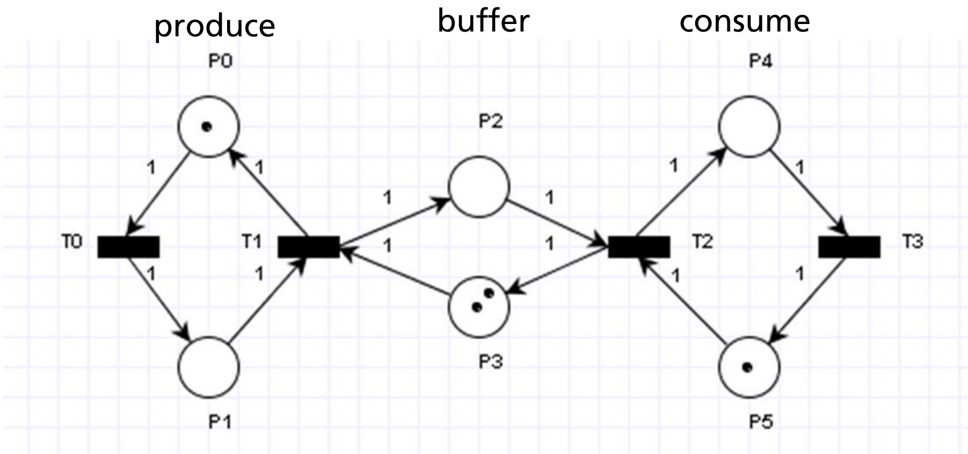
$M_0 = (m_{01}, m_{02}, \ldots\ m_{0P})$ is the initial marking

Combining the information provided by the flow relations and by the weight function, we obtain the *Incidence Matrix*

$$C = \begin{pmatrix} C_{11} & \cdots & C_{1T} \\ \vdots & \ddots & \vdots \\ C_{P1} & \cdots & C_{PT} \end{pmatrix}$$

Petri Nets: Simple example – Producer/Consumer

Petri net model:



Set of places:

Set of transitions: ( )

Initial marking:

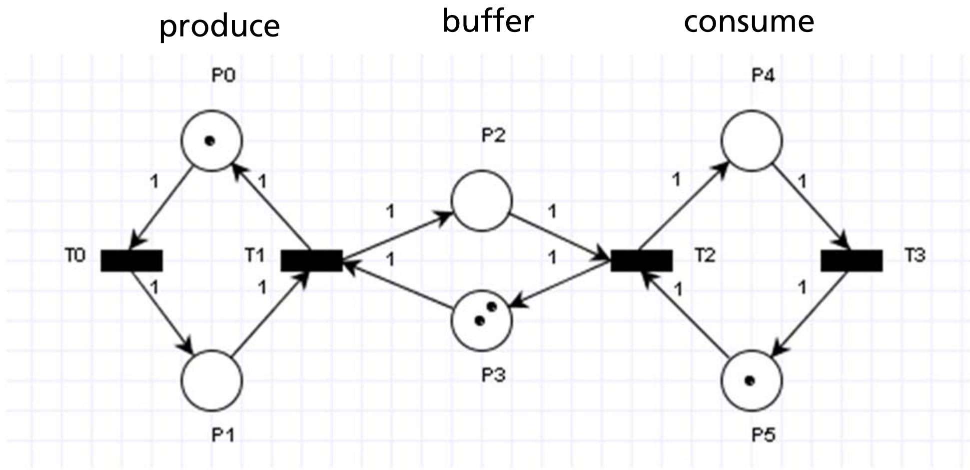
$$\begin{matrix} T0 & T1 & T2 & T3 \end{matrix}$$

Incidence matrix:

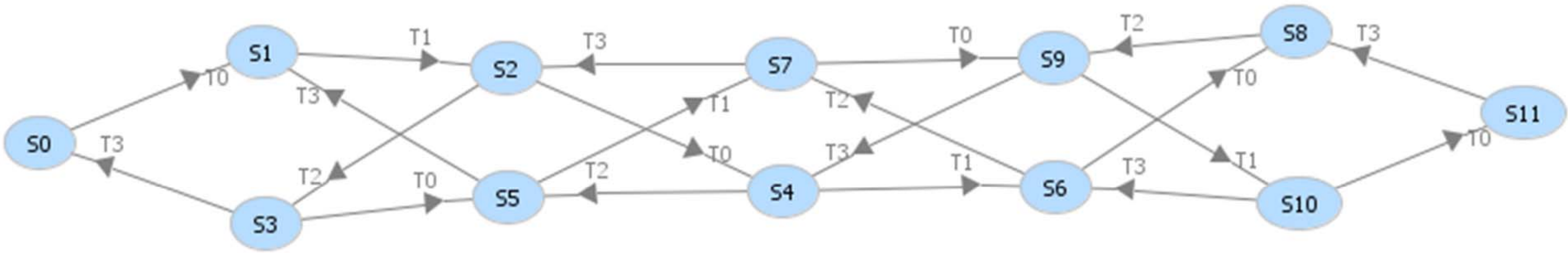
$$\begin{pmatrix} \quad & \quad & \quad & \quad \end{pmatrix} \begin{matrix} P0 \\ P1 \\ P2 \\ P3 \\ P4 \\ P5 \end{matrix}$$

Petri Nets: Simple example – Producer/Consumer

Petri net model:



Reachability Graph:



S4 [Vanishing State]

Marking: {0, 1, 1, 1, 0, 1}

Edges From: S2 (T0); S9 (T3)

Edges To: S5 (T2); S6 (T1)

Petri net types (untimed)

- Condition-event nets
  - At most one token per place
- Place-transition nets
  - Arbitrary number of tokens on places
- State machines
  - Transitions have exactly one input and output place
  - Can model finite state automata
- Marked graphs
  - Places have exactly one input and output transition
  - No conflicts possible
- Stochastic PN
- High-level PN
  - For example, coloured PN

Petri net analysis (untimed)

- PN properties
  - Behavioural properties (marking dependent)
    - Reachability → reachability graph (one node for every PN marking)
    - Liveness (deadlock free)
  - Structural properties (marking independent)
    - Concurrency
    - Synchronisation points
- Analysis
  - Incidence matrix
  - Graph-based methods → reachability graph

Time and Petri Nets

Petri net types (timed)

- Stochastic Petri nets (SPN)
  - All transition firing times are exponentially distributed
- Generalised stochastic Petri nets (GSPN)
  - Firing times are immediate or exponentially distributed
- Deterministic stochastic Petri nets (DSPN)
  - Immediate, exponentially distributed or deterministic

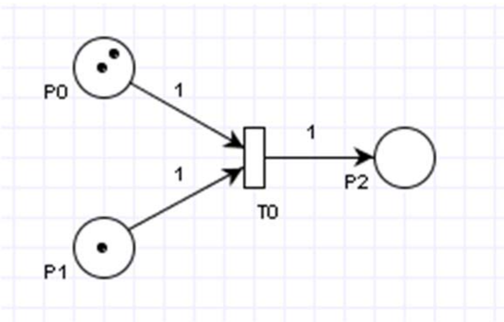
Timing Specifications

- Time is associated to places
- Time is associated to tokens
- Time is associated to arcs
- **Time is associated to transitions**

Timed transitions

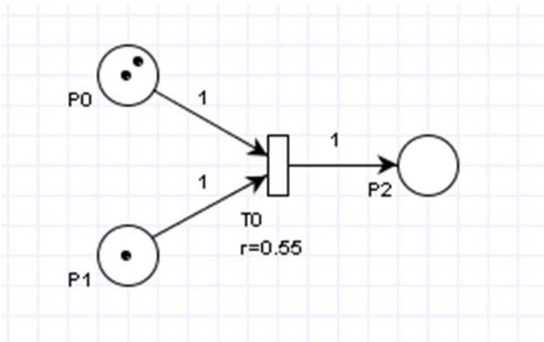
- Time is associated to transitions, that represent „activities“
  - Activity start corresponds to enabling
  - Activity end corresponds to firing

- Delay is associated with transitions



Stochastic (Exponential) Petri Nets

- The delay of a transition is a random variable
- Timed Transition PN with atomic firing and race policy in which transition delays are random variables *exponentially* distributed are called Stochastic Petri Nets (SPN)
- SPN is the name chosen by Molloy in 1982, but more adequate one could be Exponential Petri Nets



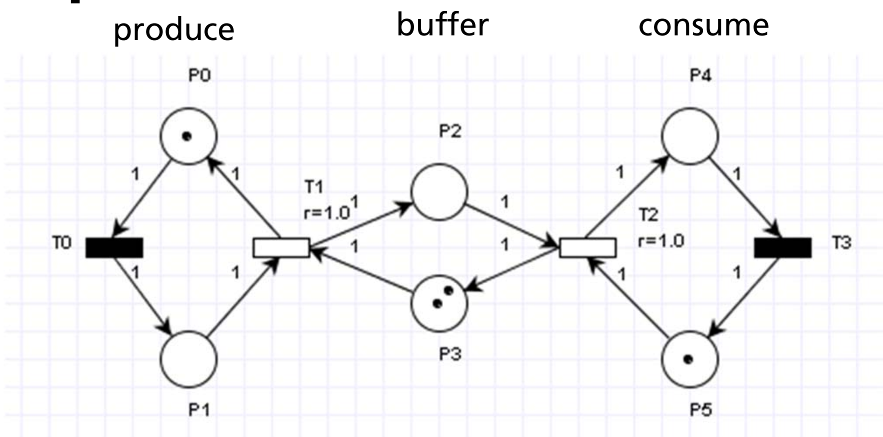
Generalized Stochastic Petri Nets

- Two types of transitions
  - Timed with an exponentially distributed delay
  - Immediate, with constant zero delay

- Why immediate transitions:
  - To account for instantaneous actions (typically choices)
  - To implement logical actions (e.g. emptying a place)

GSPN: Simple example – Producer/Consumer

GSPN model:



Set of places:

Set of transitions: ( )

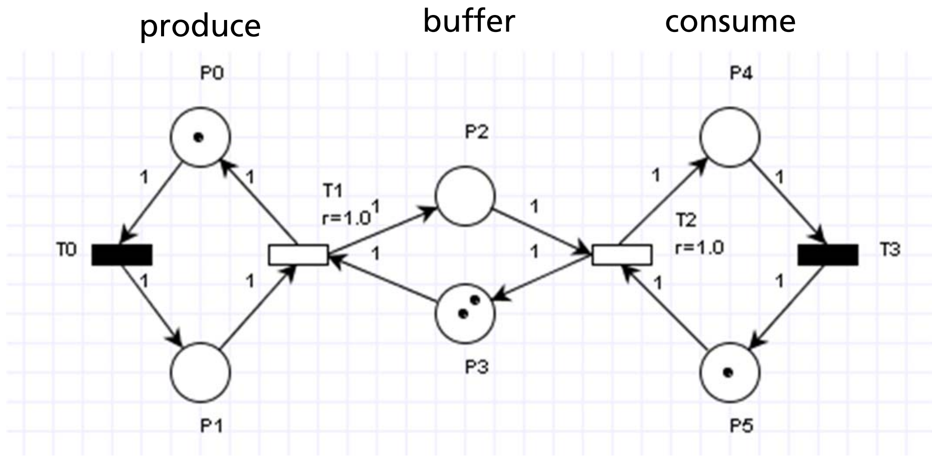
Initial marking:

Incidence matrix:

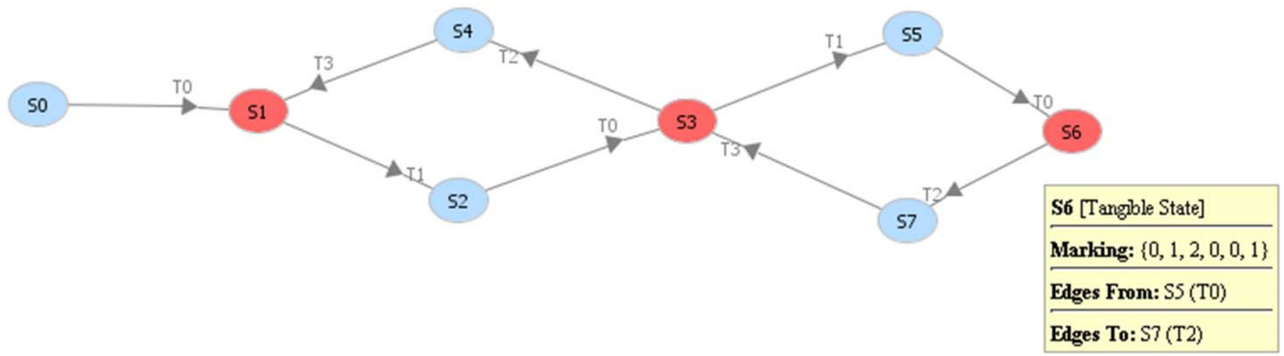
$$\begin{matrix} & T0 & T1 & T2 & T3 & \\ \Bigg(& & & & & \Bigg) \end{matrix} \quad \begin{matrix} P0 \\ P1 \\ P2 \\ P3 \\ P4 \\ P5 \end{matrix}$$

GSPN: Simple example – Producer/Consumer

Petri net model:

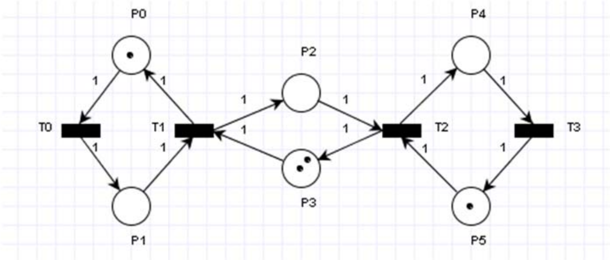


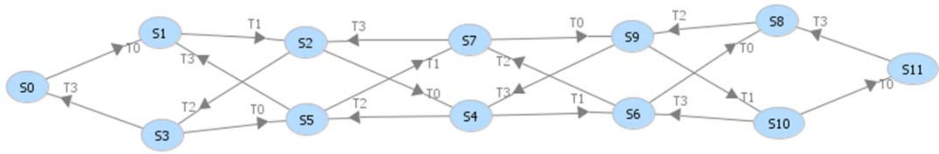
Reachability Graph:



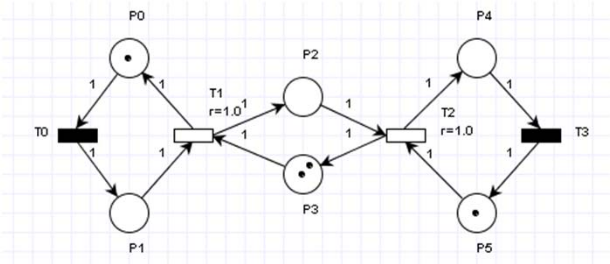
PN vs. GSPN

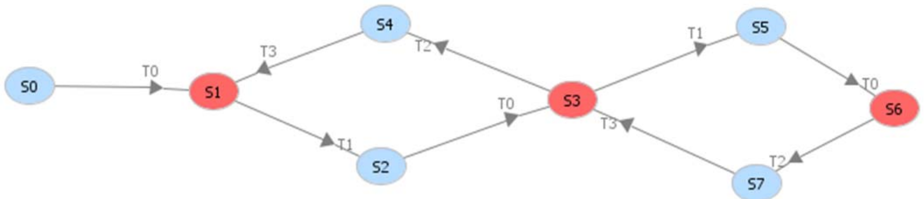
PN





GSPN





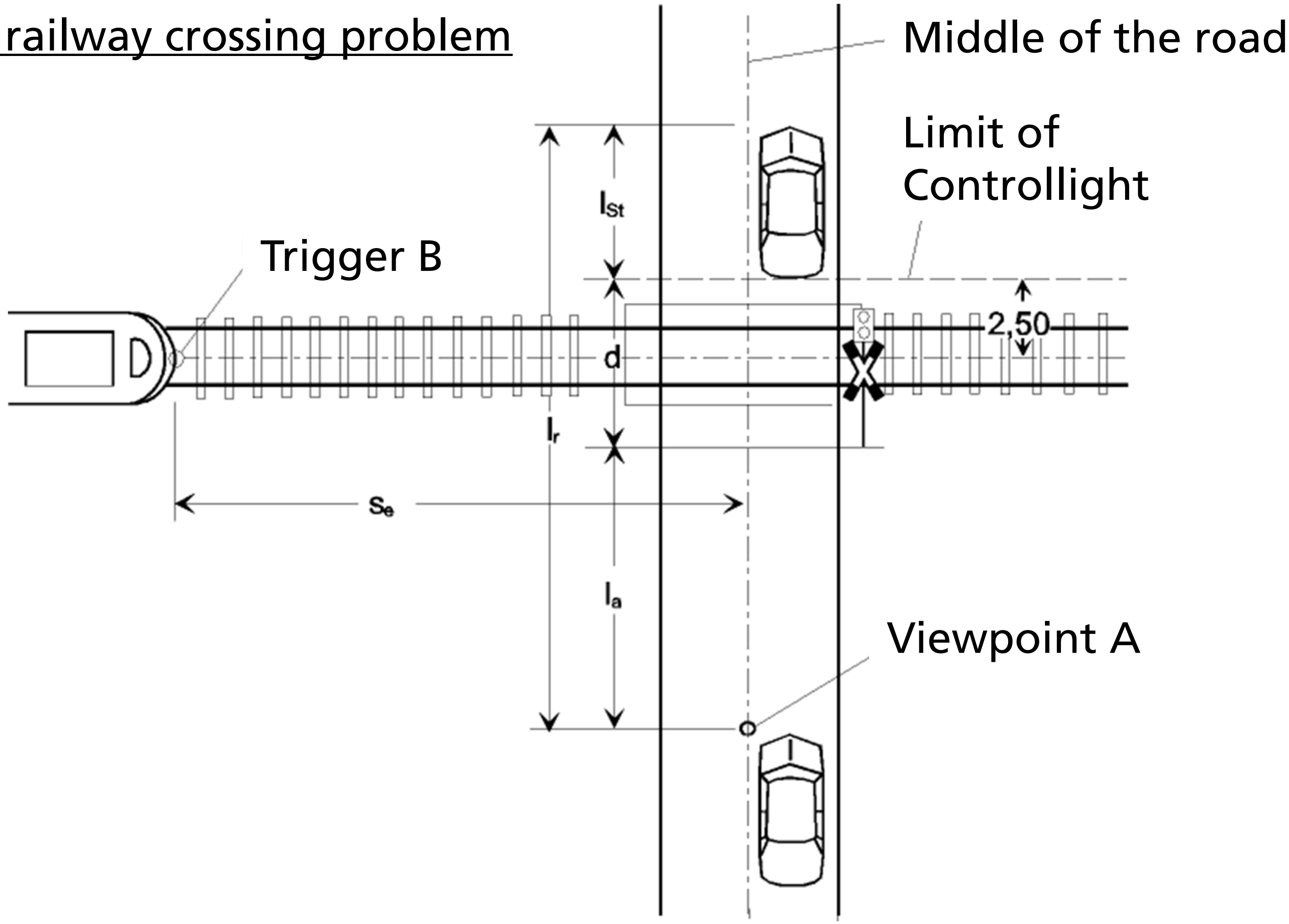
Petri net analysis (timed)

- Mapping to underlying stochastic process
- Reward measures derived from state probabilities of stochastic process
  - Determine reachability graph of SPN and GSPN
  - Convert reachability graph to Markov chain
  - May experience state space largeness problems
  - DSPN are mapped to *embedded* Markov chain

- Simulation
  - Statistical measures
  - No problems with state space size (except precision)

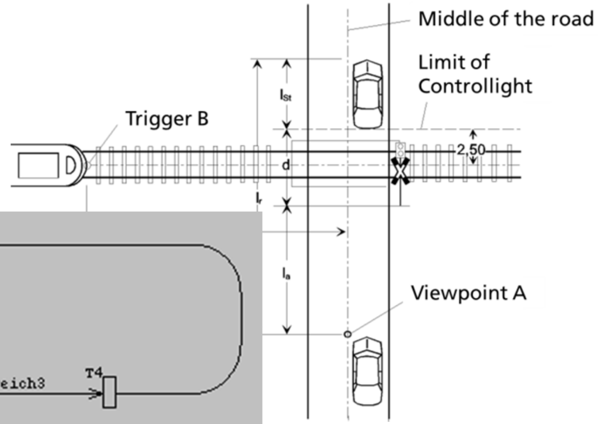
Petri nets – a practical view

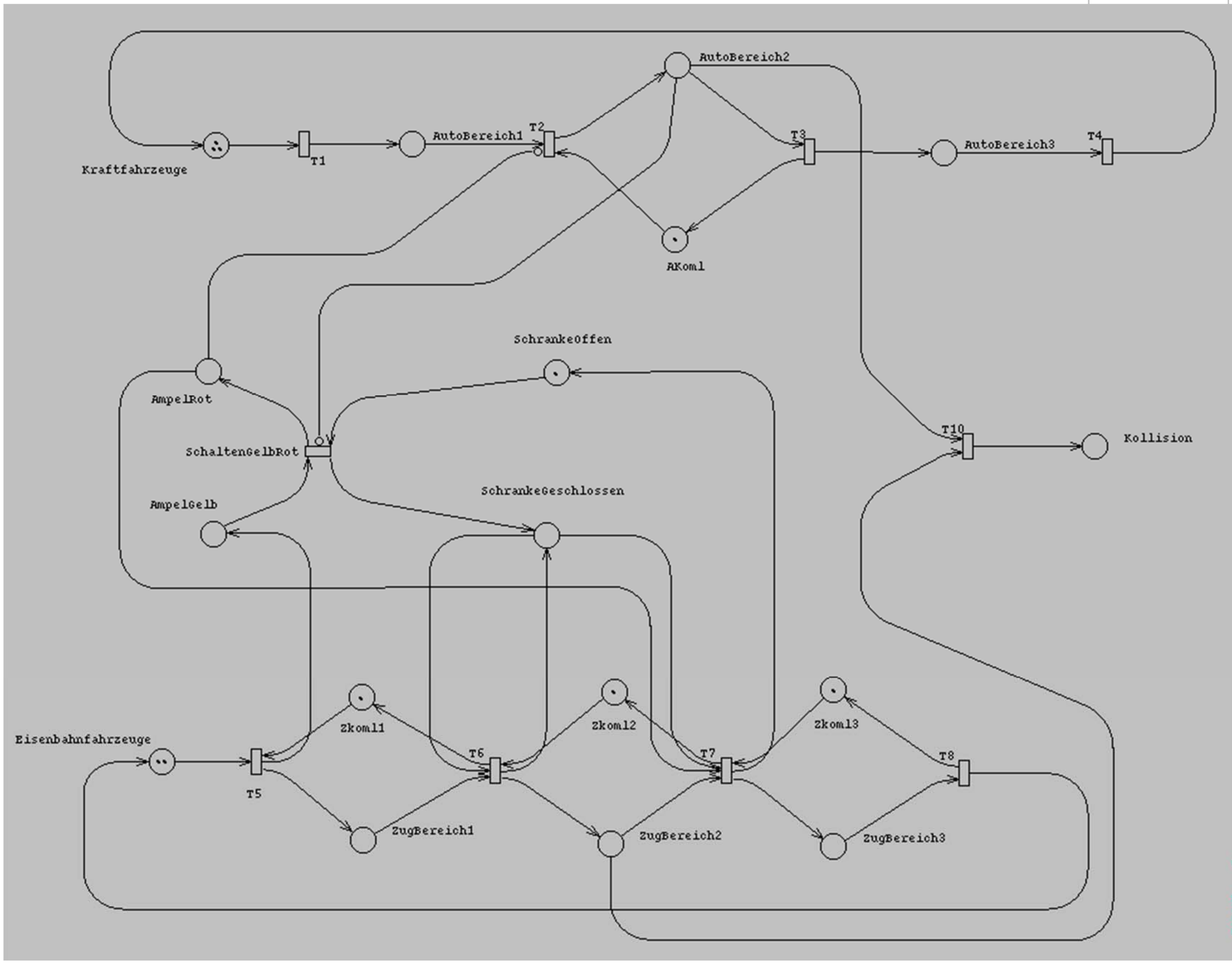
General railway crossing problem



Fraunhofer IESE

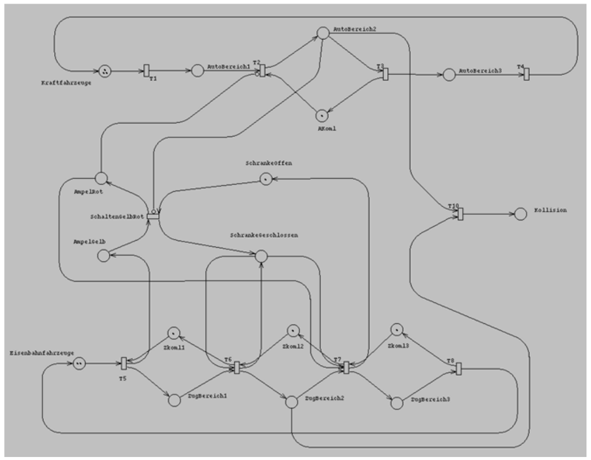
Petri nets – a practical view

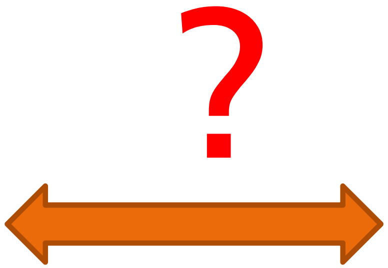


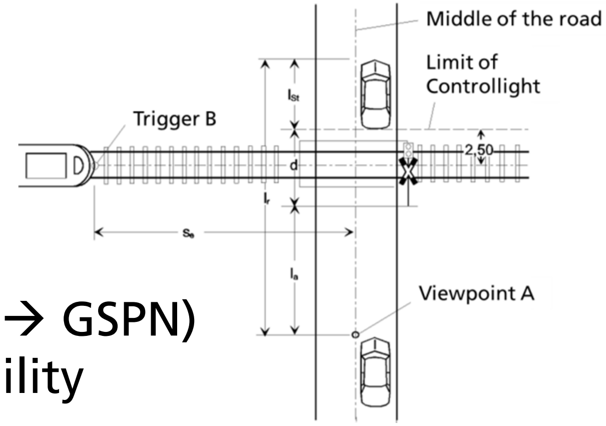


Fraunhofer
IESE

Petri nets – a practical view







- Mapping (F ←→ GSPN)
- Understandability
- Scalability
- *ilities

Expressive power of model types

Property	FME(C)A	FTA	ETA	RBD	Markov chain	Petri net
Direction of search	Inductive	Deductive	Inductive	Deductive	Inductive	Inductive/deductive
Sequence-dependent behaviour	No	FT extensions (DFT, SEFT)	No*	No	Yes	Yes
Deterministic dependences	No	FT extensions (SEFT)	No	No	No	Yes
Components	Yes	FT extensions (CFT, SEFT)	No	Yes	No**	No**
Semi-quantitative analysis (ordinal scale)	Yes	No***	No	No	No	No
Quantitative analysis	No	Yes	Yes	Yes	Yes	Yes

** → There are approaches to tackle components

*** → There are approaches for semi-quantitative Analysis