

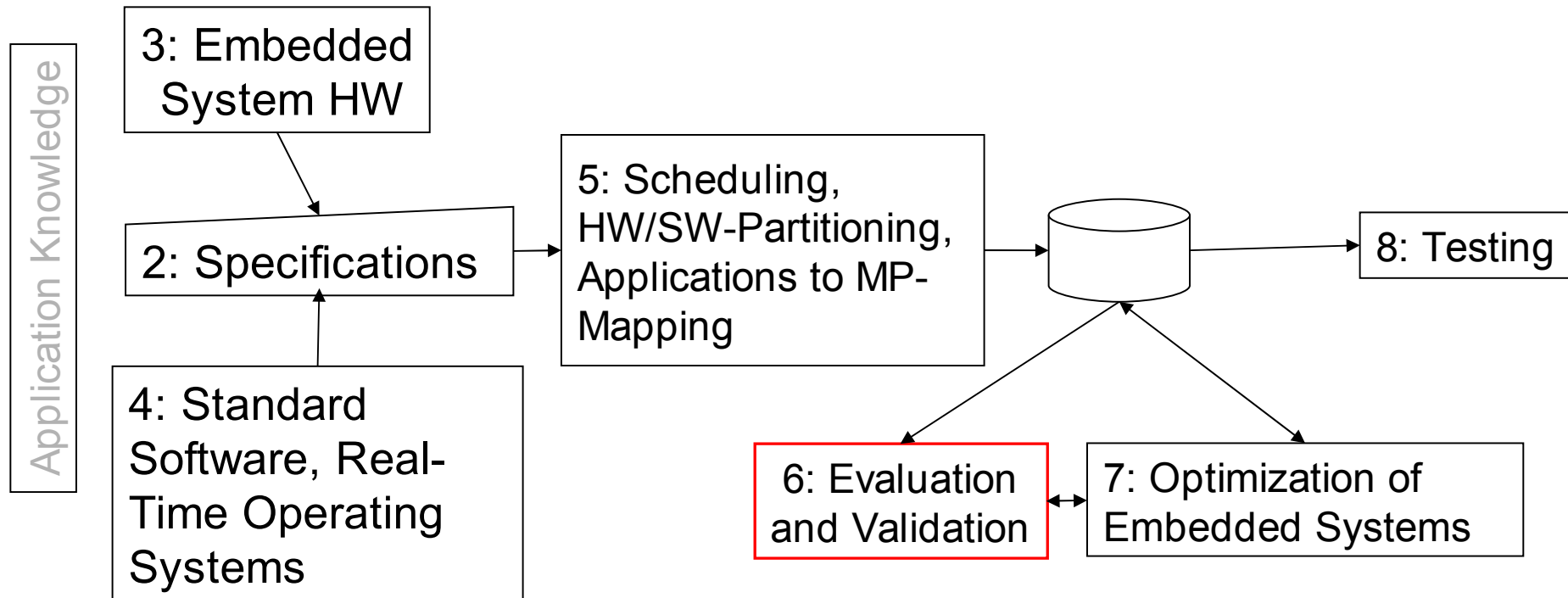
# Evaluation and Validation

Peter Marwedel  
TU Dortmund, Informatik 12  
Germany

2009/01/12



# Structure of this course



# Evaluation and Validation

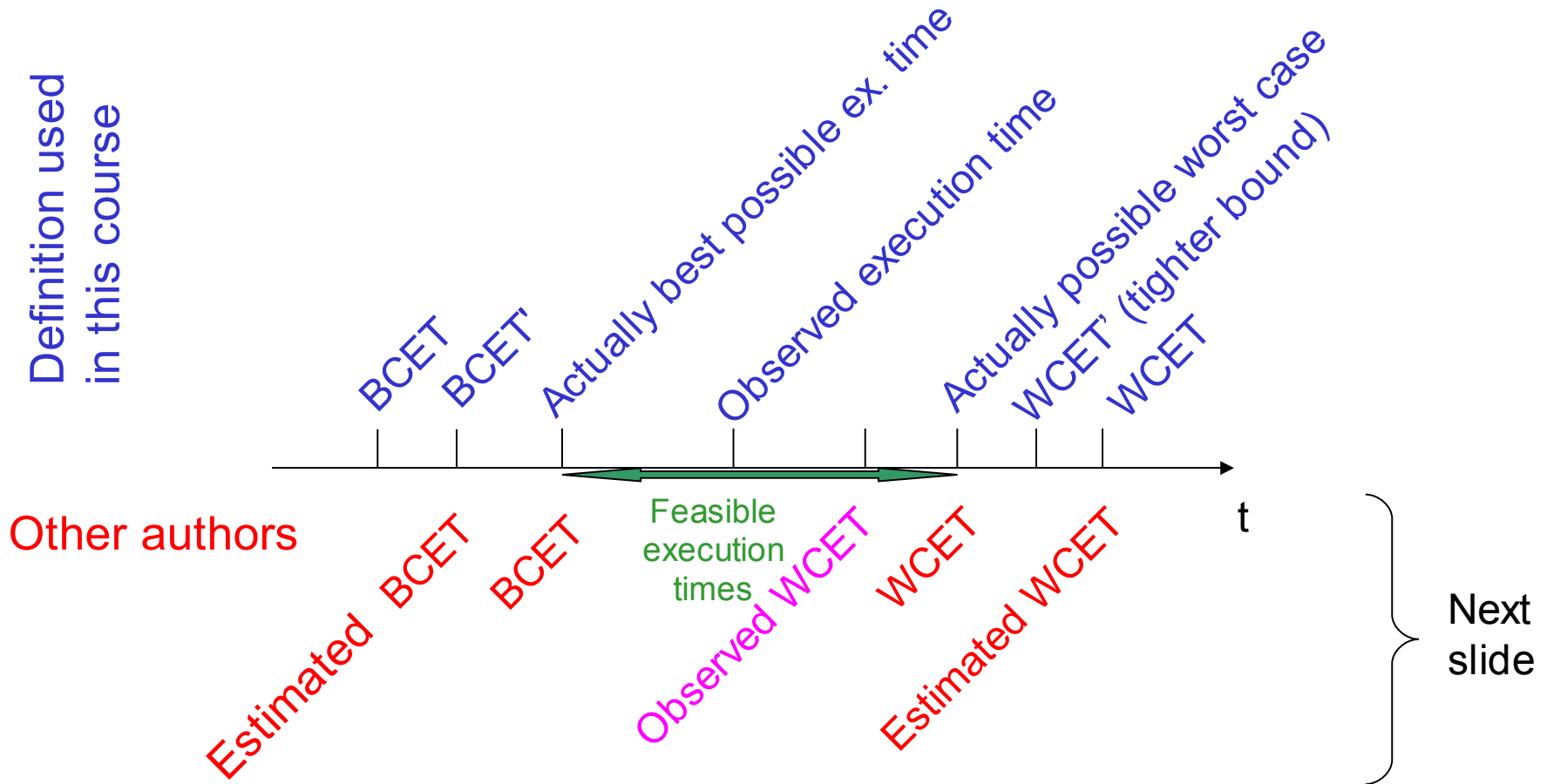
---

**Definition:** *Evaluation* is the process of computing quantitative information of some key characteristics of a certain (possibly partial) design.

**Definition:** *Validation* is the process of checking whether or not a certain (possibly partial) design is appropriate for its purpose, meets all constraints and will perform as expected (yes/no decision).

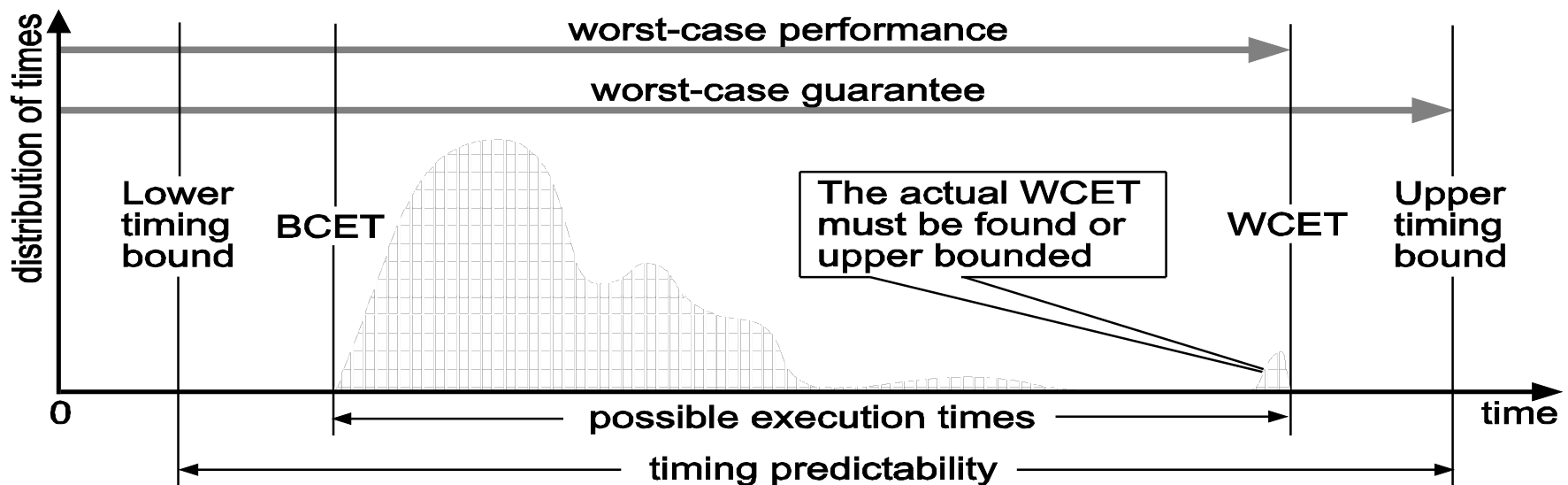
**Definition:** Validation with mathematical rigor is called *(formal) verification*.

# Worst/best case execution times (1)



# Estimation of Worst-Case Execution Times

**Solution:** Estimation of upper bounds for the actual (unknown) WCET



**Requirements on WCET estimates:**

- *Safeness:*  $WCET \leq WCET_{EST}$ !
- *Tightness:*  $WCET_{EST} - WCET \rightarrow \text{minimal}$

# Worst case execution times (2)

---



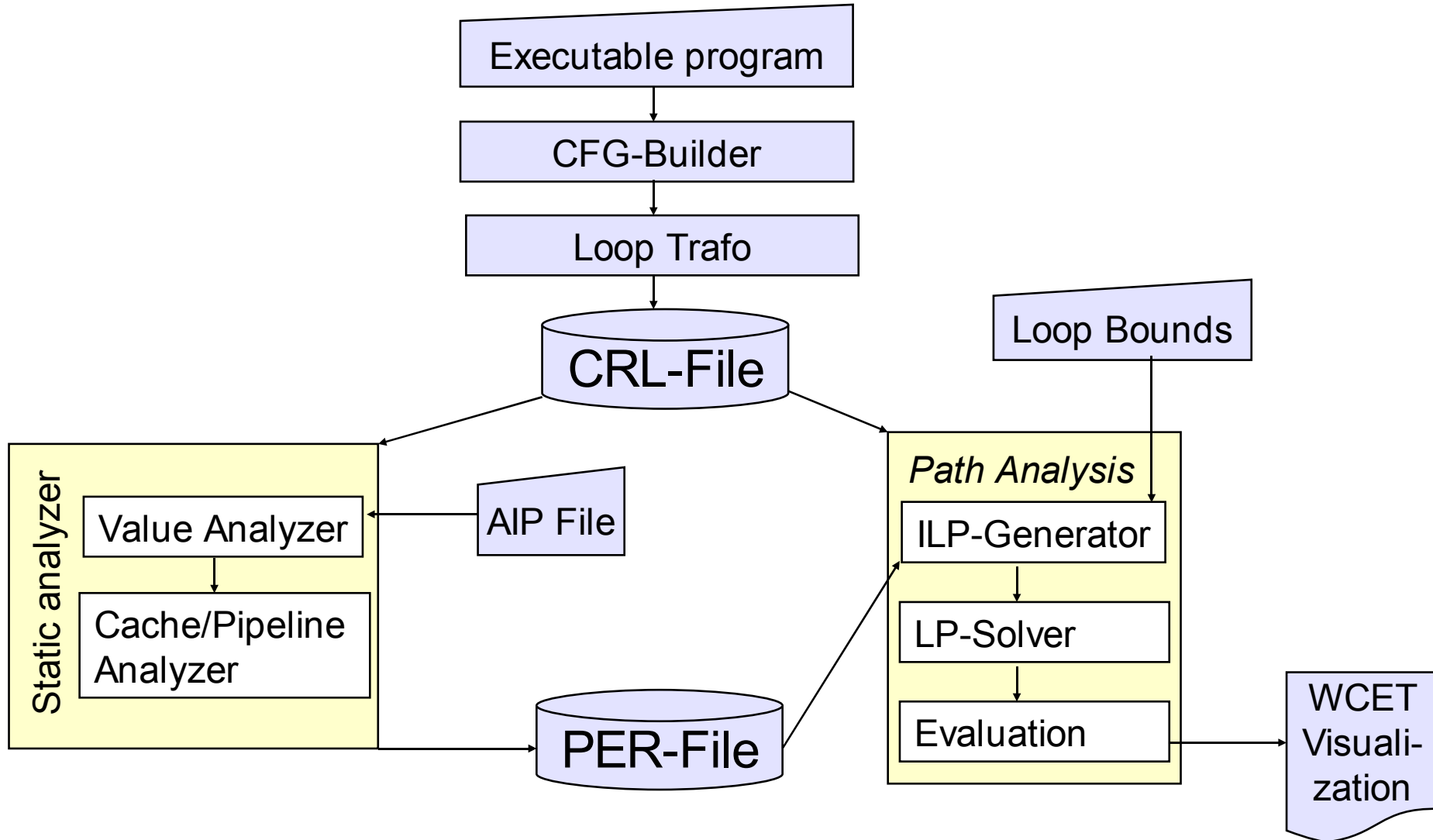
## Complexity:

- in the general case: undecidable if a bound exists.
- for restricted programs: simple for “old“ architectures, very complex for new architectures with pipelines, caches, interrupts, virtual memory, etc.

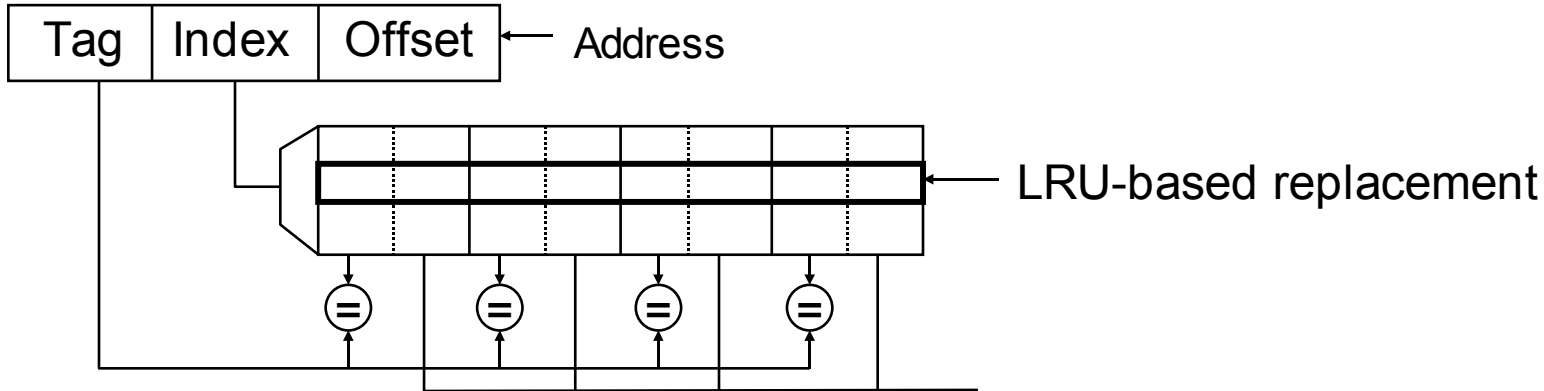
## Approaches:

- for hardware: requires detailed timing behavior
- for software: requires availability of machine programs; complex analysis (see, e.g., [www.absint.de](http://www.absint.de))

# WCET estimation: AiT (AbsInt)

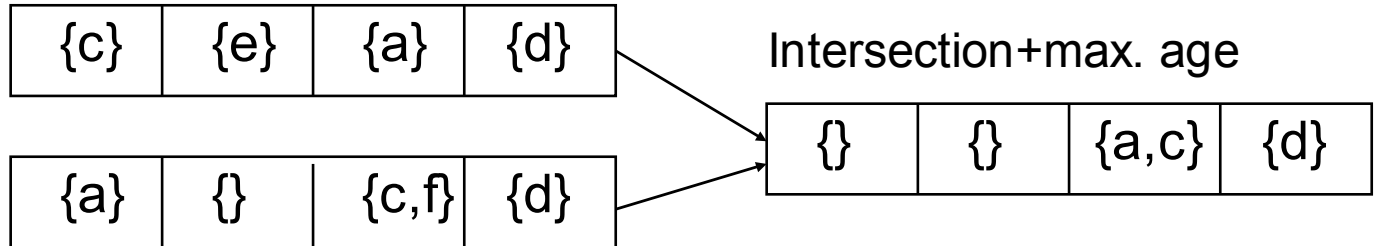


# WCET estimation for caches

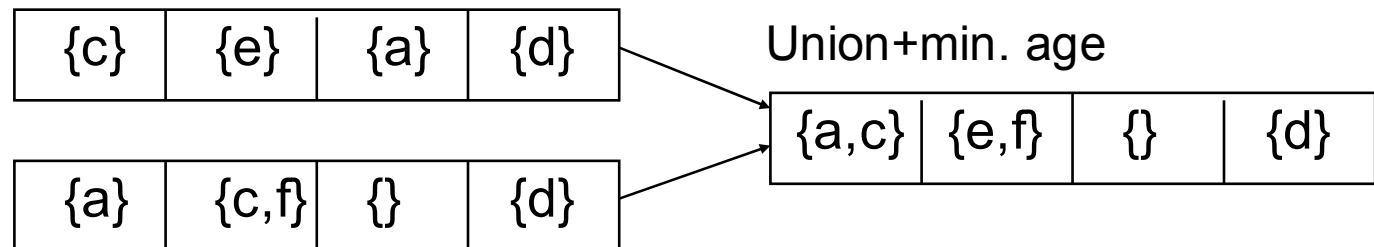


## *Behavior at program joins*

Worst case



Best case



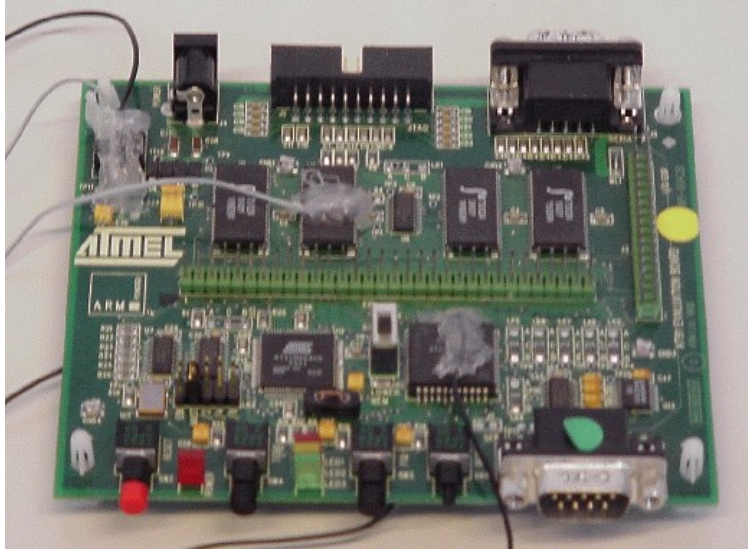


## Energy models

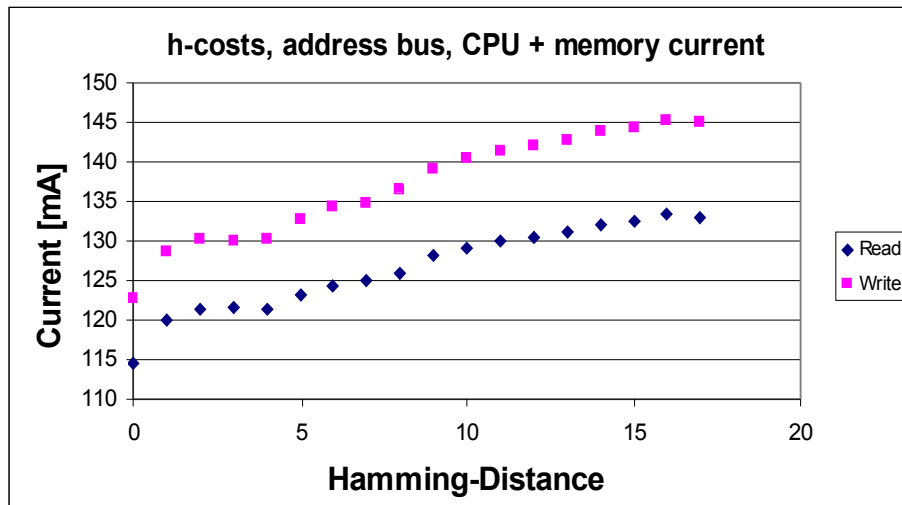
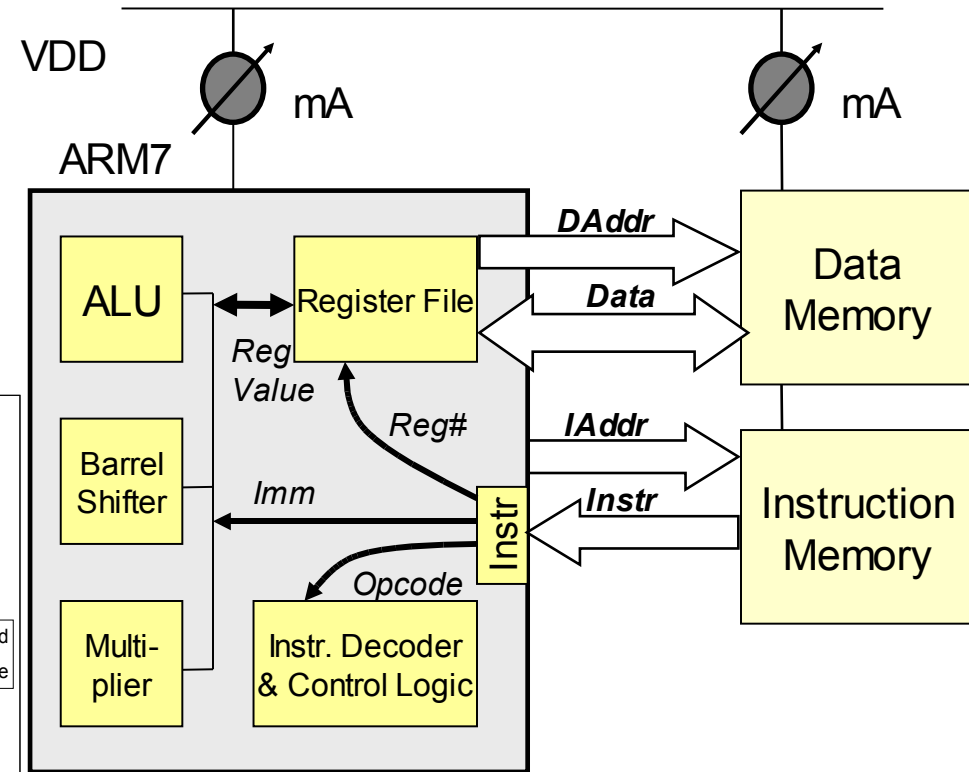
- Tiwari (1994): Energy consumption within processors
- Simunic (1999): Using values from data sheets. Allows modeling of all components, but not very precise.
- Russell, Jacome (1998): Measurements for 2 fixed configurations
- Steinke et al., UniDo (2001): mixed model using measurements and prediction
- Jouppi (1996): Energy consumption of caches predicted by CACTI.



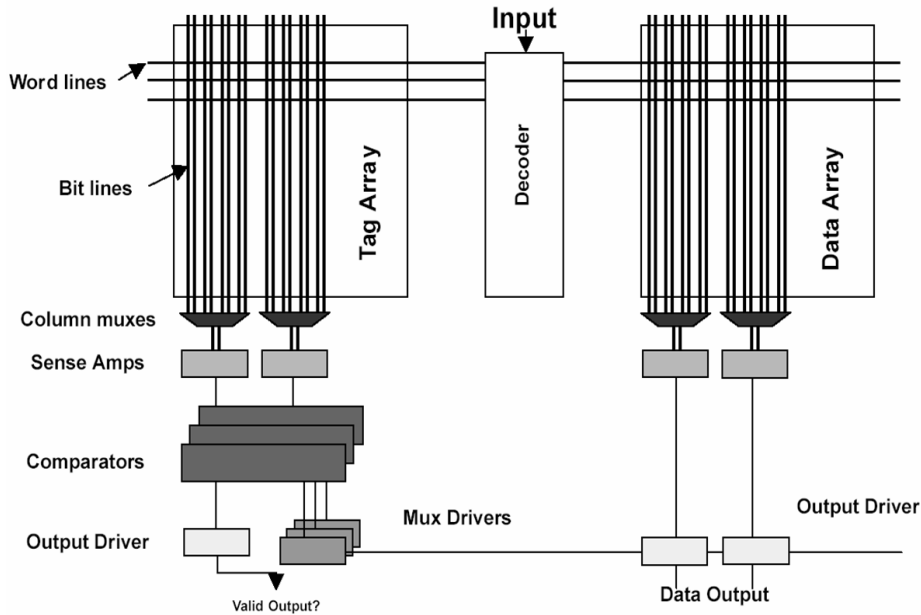
# Steinke's model



E.g.: ATMELEVAL board with ARM7TDMI and ext. SRAM

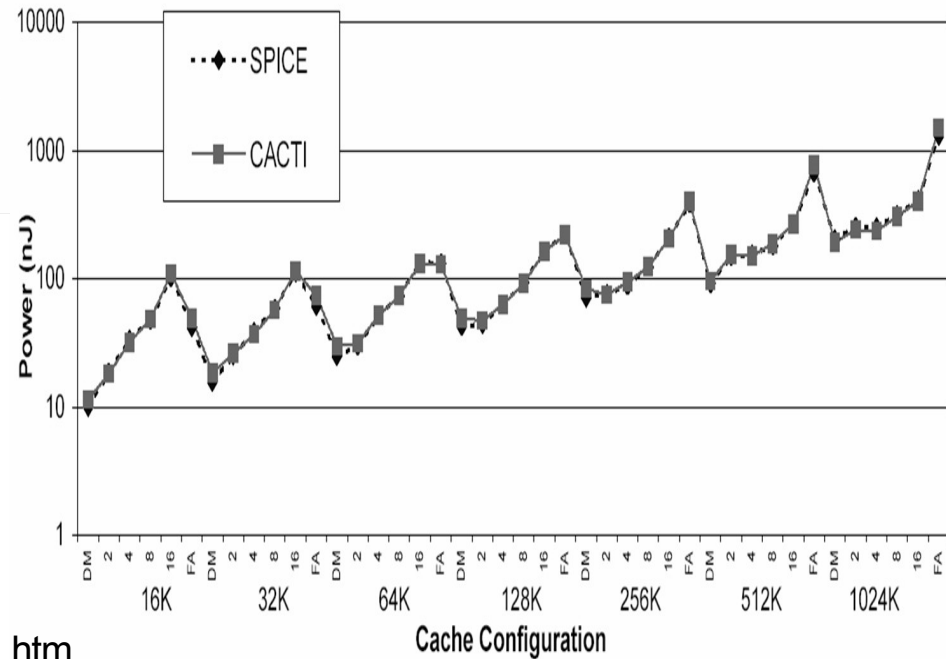


# CACTI model



Cache model used

## Comparison with SPICE

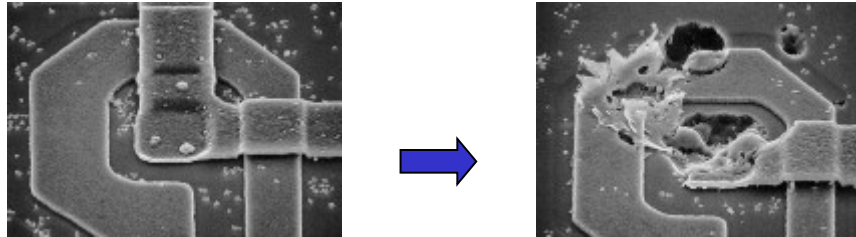


<http://research.compaq.com/wrl/people/jouppi/CACTI.htm>

# Risk- and dependability analysis

Example : metal migration @ Pentium 4

[www.jrwhipple.com/computer\\_hangs.html](http://www.jrwhipple.com/computer_hangs.html)



“ $10^{-9}$ “: For many systems, probability of a catastrophe has to be less than  $10^{-9}$  per hour  $\equiv$  one case per 100,000 systems for 10,000 hours.

FIT: failure-in-time unit for failure rate ( $=1/\text{MTTF} \approx 1/\text{MTBF}$ );

**1 FIT: rate of  $10^{-9}$  failures per hour**

Damages are resulting from hazards.

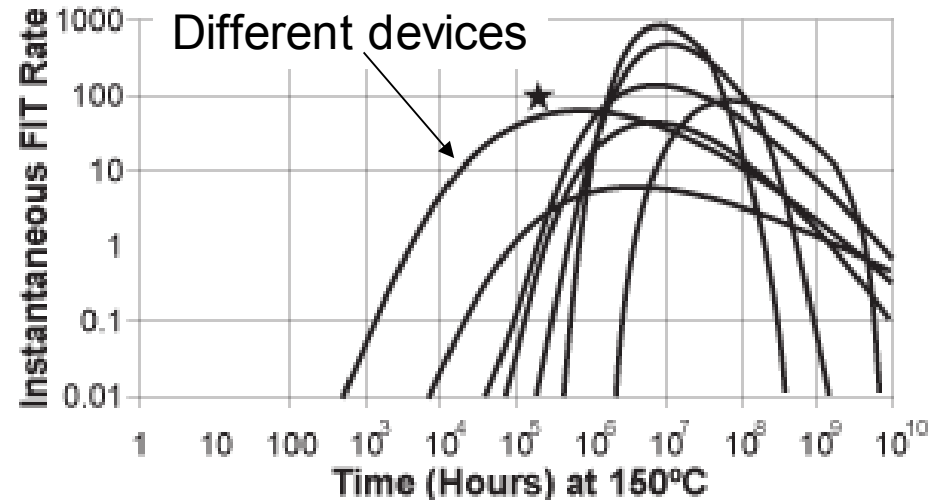
For every damage there is a severity and a probability.

Several techniques for analyzing risks.

# Actual failure rates

Example: failure rates less than 100 FIT for the first 20 years (175,300 hrs) of life at 150°C @ TriQuint (GaAs)

[\[www.triquint.com/company/quality/faqs/faq\\_11.cfm\]](http://www.triquint.com/company/quality/faqs/faq_11.cfm)



Target: Failures rates of systems  $\leq 1$ FIT

Reality: Failures rates of circuits  $\leq 100$  FIT

☞ redundancy is required to make a system more reliable than its components

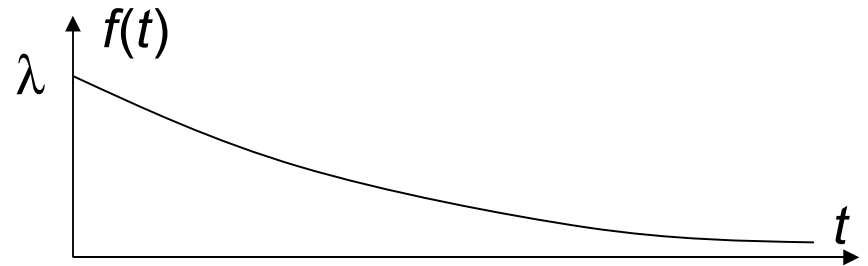
Analysis frequently works with simplified models ☞

# Reliability: $f(t)$ , $F(t)$

- Let  $T$ : time until first failure,  $T$  is a random variable
- Let  $f(t)$  be the density function of  $T$

Example: Exponential distribution

$$f(t) = \lambda e^{-\lambda t}$$

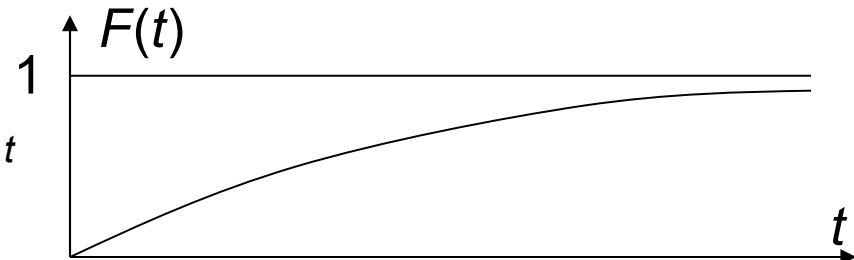


- $F(t)$  = probability of the system being faulty at time  $t$ :

$$F(t) = \Pr(T \leq t) \quad F(t) = \int_0^t f(x) dx$$

Example: Exponential distribution

$$F(t) = \int_0^t \lambda e^{-\lambda x} dx = -[e^{-\lambda x}]_0^t = 1 - e^{-\lambda t}$$



# Reliability: $R(t)$

- **Reliability**  $R(t)$  = probability that the time until the first failure is larger than some time  $t$ :

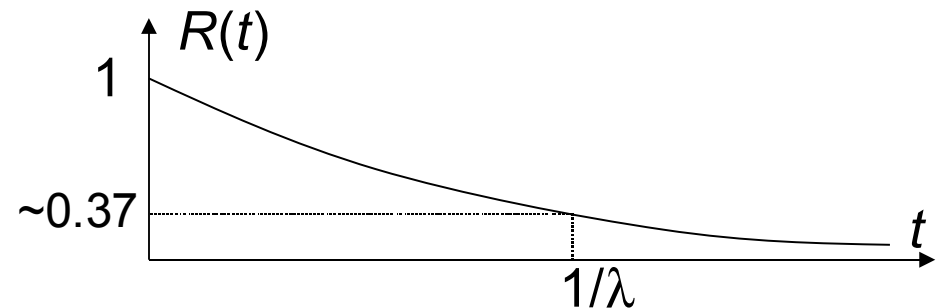
$$R(t) = \Pr(T > t), t \geq 0 \quad R(t) = \int_t^{\infty} f(x) dx$$

$$F(t) + R(t) = \int_0^t f(x) dx + \int_t^{\infty} f(x) dx = 1$$

$$R(t) = 1 - F(t) \quad f(t) = - \frac{dR(t)}{dt}$$

Example: Exponential distribution

$$R(t) = e^{-\lambda t}$$



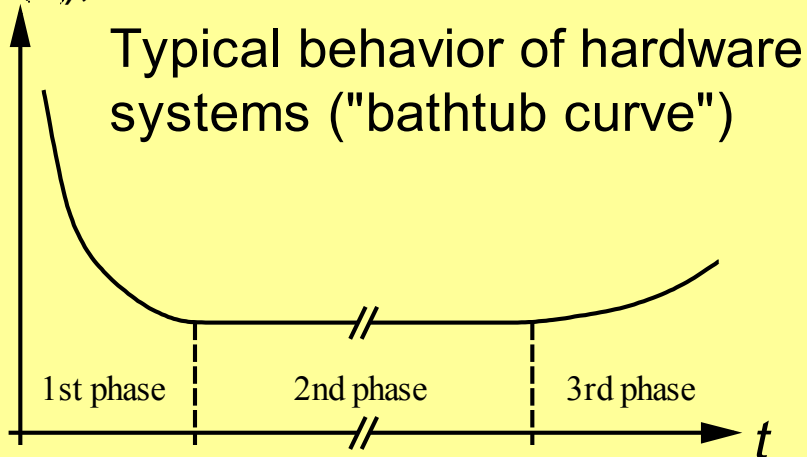
# Failure rate

The failure rate at time  $t$  is the probability of the system failing between time  $t$  and time  $t+\Delta t$ :

$$\lambda(t) = \lim_{\Delta t \rightarrow 0} \frac{\Pr(t < T \leq t + \Delta t | T > t)}{\Delta t} = \lim_{\Delta t \rightarrow 0} \frac{F(t + \Delta t) - F(t)}{\Delta t R(t)} = \frac{f(t)}{R(t)}$$

Conditional probability  
("provided that the system works at  $t$ ");

$$P(A|B) = P(AB)/P(B)$$



For exponential distribution:

$$\frac{f(t)}{R(t)} = \frac{\lambda e^{-\lambda t}}{e^{-\lambda t}} = \lambda$$

FIT = expected number of failures in  $10^9$  hrs.



# MTTF = $E\{T\}$ , the *statistical mean value* of $T$

---

$$\text{MTTF} = E\{T\} = \int_0^{\infty} t \cdot f(t) dt$$

According to the definition of the statistical mean value

Example: Exponential distribution

$$\text{MTTF}_{\text{exp}} = \int_0^{\infty} t \cdot \lambda e^{-\lambda t} dt = -\left[ \cancel{t \cdot e^{-\lambda t}} \right]_0^{\infty} + \int_0^{\infty} e^{-\lambda t} dt$$

$$\int u \cdot v' = u \cdot v - \int u' \cdot v$$

$$\text{MTTF}_{\text{exp}} = -\frac{1}{\lambda} \left[ e^{-\lambda t} \right]_0^{\infty} = -\frac{1}{\lambda} [0 - 1] = \frac{1}{\lambda}$$

MTTF is the reciprocal value of failure rate.

# MTTF, MTTR and MTBF

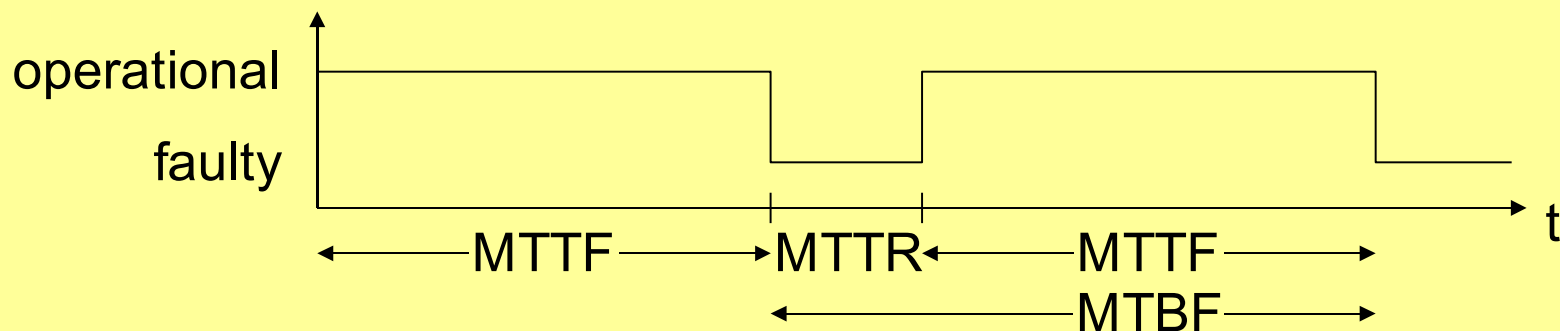
MTTR = mean time to repair

(average over repair times using distribution  $M(d)$ )

MTBF\* = mean time between failures = MTTF + MTTR

$$\text{Availability } A = \lim_{t \rightarrow \infty} A(t) = \frac{\text{MTTF}}{\text{MTBF}}$$

Ignoring the statistical nature of faults ...



\* Mixed up with MTTF, if starting in operational state is implicitly assumed

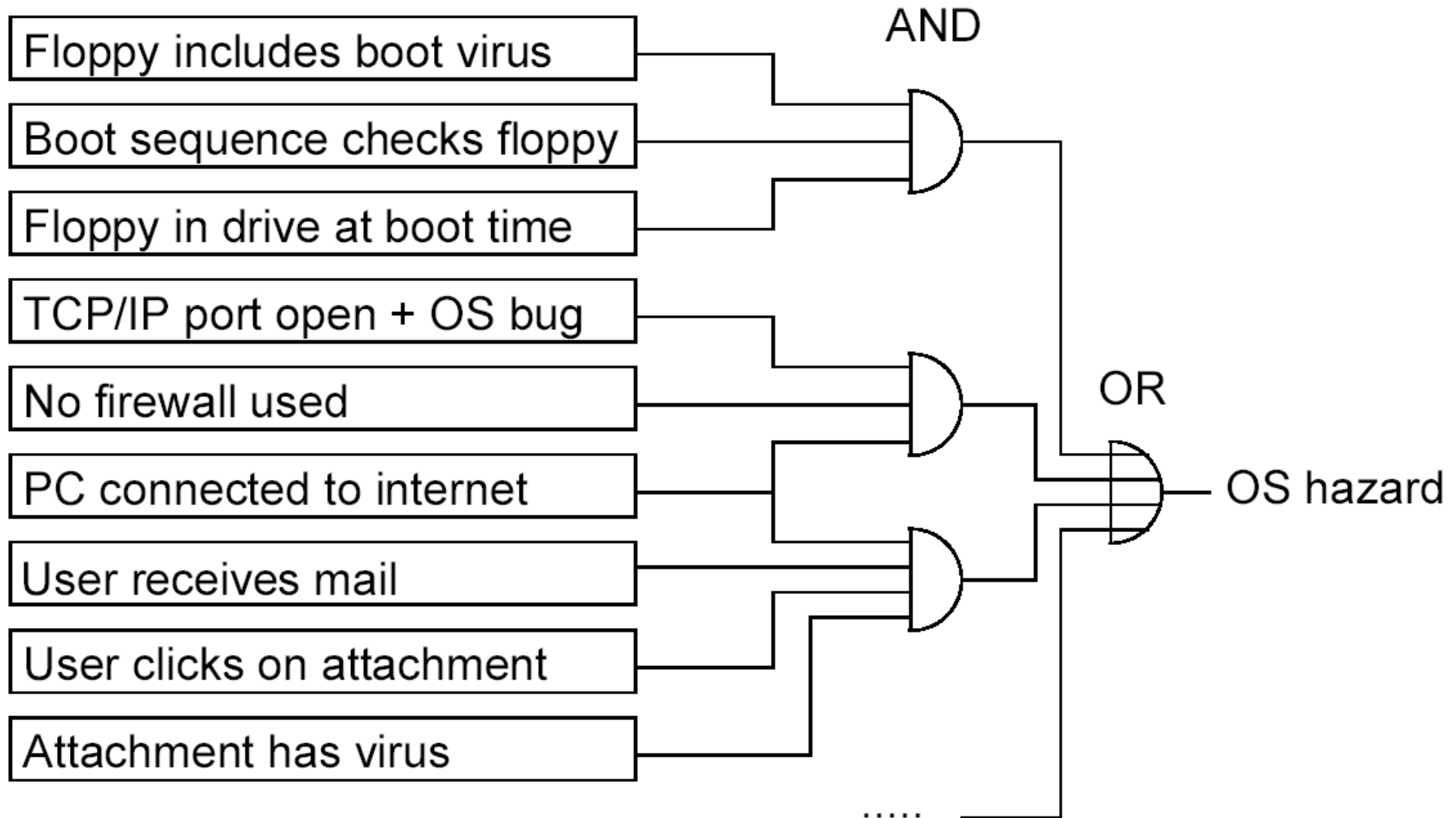
# Fault tree Analysis (FTA)

---

- FTA is a top-down method of analyzing risks. Analysis starts with possible damage, tries to come up with possible scenarios that lead to that damage.
- FTA typically uses a graphical representation of possible damages, including symbols for AND- and OR-gates.
- OR-gates are used if a single event could result in a hazard.
- AND-gates are used when several events or conditions are required for that hazard to exist.



# Example



# Limitations

---

The simple AND- and OR-gates cannot model all situations. For example, their modeling power is exceeded if shared resources of some limited amount (like energy or storage locations) exist.

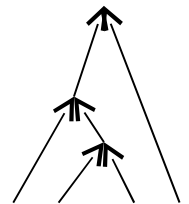
Markov models may have to be used to cover such cases.

# Failure mode and effect analysis (FMEA)

- FMEA starts at the components and tries to estimate their reliability. The first step is to create a table containing components, possible faults, probability of faults and consequences on the system behavior.

<i>Component</i>	<i>Failure</i>	<i>Consequences</i>	<i>Probability</i>	<i>Critical?</i>
Processor	metal migration	no service	$10^{-6}$ /h	yes
...	...	...	...	...

- Using this information, the reliability of the system is computed from the reliability of its parts (corresponding to a bottom-up analysis).



# Safety cases

---

Both approaches may be used in “safety cases”. In such cases, an independent authority has to be convinced that certain technical equipment is indeed safe.

One of the commonly requested properties of technical systems is that no single failing component should potentially cause a catastrophe.

## Formal verification

- Formal verification = formally proving a system correct, using the language of mathematics.
- Formal model required. Obtaining this cannot be automated.
- Model available → try to prove properties.
- Even a formally verified system can fail (e.g. if assumptions are not met).
- Classification by the type of logics.



**Ideally:** Formally verified tools transforming specifications into implementations (“*correctness by construction*”).

**In practice:** Non-verified tools and manual design steps → validation of each and every design required Unfortunately has to be done at intermediate steps and not just for the final design → Major effort required.



# Propositional logic (1)

---

- Consisting of Boolean formulas comprising Boolean variables and connectives such as  $\vee$  and  $\wedge$ .
- Gate-level logic networks can be described.
- Typical aim: checking if two models are equivalent (called **tautology checkers** or **equivalence checkers**).
- Since propositional logic is decidable, it is also decidable whether or not the two representations are equivalent.
- Tautology checkers can frequently cope with designs which are too large to allow simulation-based exhaustive validation.

# Propositional logic (2)

---

- Reason for power of tautology checkers: Binary Decision Diagrams (BDDs)
- Complexity of equivalence checks of Boolean functions represented with BDDs:  $O(\text{number of BDD-nodes})$  (equivalence check for sums of products is NP-hard).  $\#(\text{BDD-nodes})$  not to be ignored!
- Many functions can be efficiently represented with BDDs. In general, however, the  $\#(\text{nodes})$  of BDDs grows exponentially with the number of variables.
- Simulators frequently replaced by equivalence checkers if functions can be efficiently represented with BDDs.
- Very much limited ability to verify FSMs.

# First order logic (FOL)

---

FOL includes quantification, using  $\exists$  and  $\forall$ .

Some automation for verifying FOL models is feasible.

However, since FOL is undecidable in general, there may be cases of doubt.

# Higher order logic (HOL)

---

Higher order logic allows functions to be manipulated like other objects.

For higher order logic, proofs can hardly ever be automated and typically must be done manually with some proof-support.

# Model checking

---

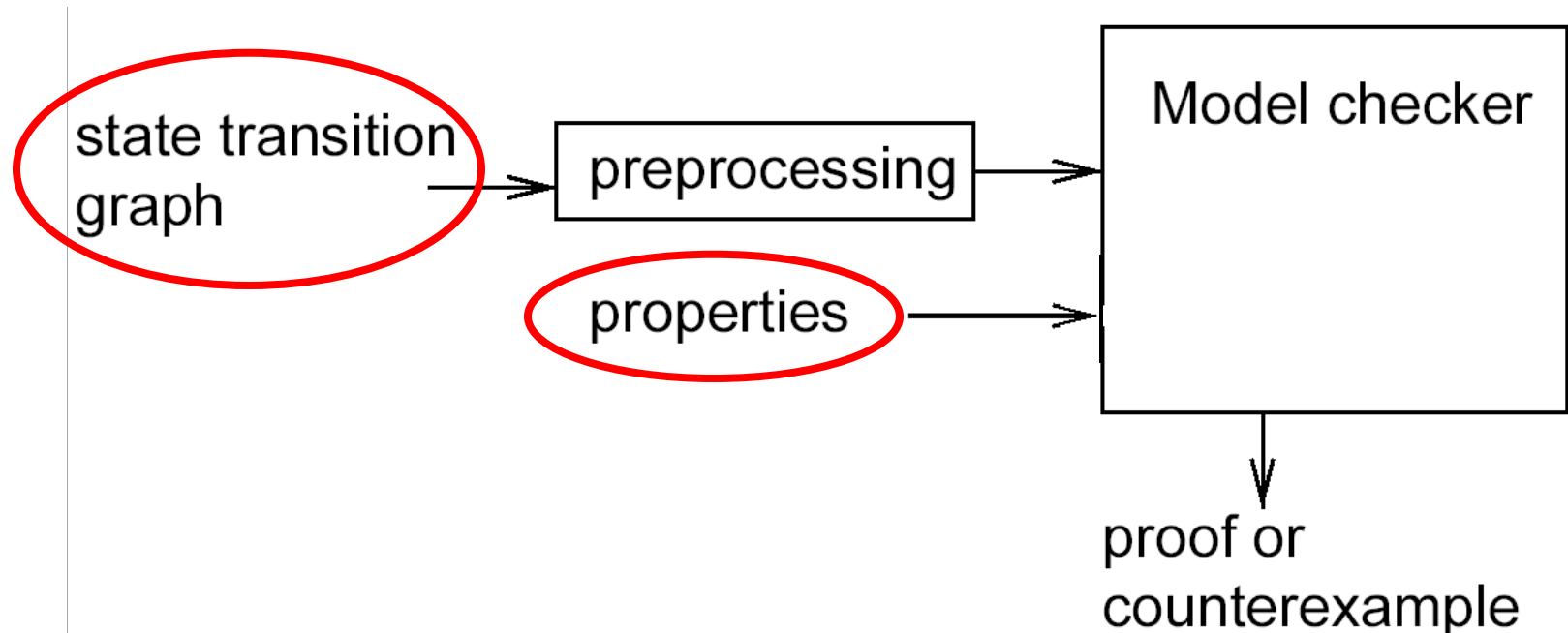
Aims at the verification of finite state systems.

Analyzes the state space of the system.

Verification using this approach requires three stages:

- generation of a model of the system to be verified,
- definition of the properties expected, and
- model checking (the actual verification step).

## 2 types of input



Verification tools can prove or disprove the properties. In the latter case, they can provide a counter-example.

**Example: Clarke's EMC-system**

# Computation tree logic (CTL)

Let  $V$  be a set of atomic propositions

CTL formulas are defined recursively:

1. Every atomic proposition is a formula
  2. If  $f_1$  and  $f_2$  are CTL formulas, then so are  $\neg f_1$ ,  $f_1 \wedge f_2$ ,  $AX f_1$ ,  $EX f_1$ ,  $A[f_1 U f_2]$  and  $E[f_1 U f_2]$
- $AX f_1$  means: holds in state  $s^\circ$  iff  $f_1$  holds in all successor states of  $s^\circ$
  - $EX f_1$  means: There exists a successor such that  $f_1$  holds
  - $A[f_1 U f_2]$  means: always until.
  - $E[f_1 U f_2]$  means: There exists a path such that  $f_1$  holds until is  $f_2$  satisfied.

Christoph Kern and Mark R. Greenstreet: Formal Verification In Hardware Design: A Survey, ACM Transactions on Design Automation of Electronic Systems, Vol. 4, No. 2, April 1999, Pages 123–193.

# Computational properties

---

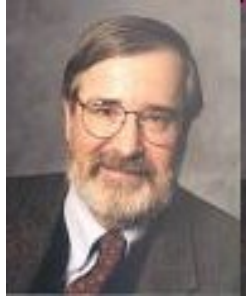
- Model checking is easier to automate than FOL.
- In 1987, model checking was implemented using BDDs.
- It was possible to locate several errors in the specification of the *future bus* protocol.
- Extensions are needed in order to also cover real-time behavior and numbers.



# ACM Turing award 2008

## granted for basic work on model checking

---



Edmund M. Clarke, CMU, Pittsburgh



E. Allen Emerson, U. Texas at Austin



Joseph Sifakis, VERIMAG, Grenoble

# Fault injection

Fault simulation may be too time-consuming

- ☞ If real systems are available, faults can be injected.




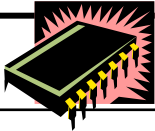

Two types of fault injection:

5. local faults within the system, and
6. faults in the environment (behaviors which do not correspond to the specification).

For example, we can check how the system behaves if it is operated outside the specified temperature or radiation ranges.

# Physical fault injection

Hardware fault injection requires major effort, but generates precise information about the behavior of the real system.  
3 techniques compared in the PDCS project on the MARS hardware [Kopetz]:

Injection Technique	Heavy-ion 	Pin-level 	EMI 
Controllability, space	Low	High	Low
Controllability, time	None	High/medium	Low
Flexibility	Low	Medium	High
Reproducibility	Medium	High	Low
Physical reachability	High	Medium	Medium
Timing measurement	Medium	high	Low

# Software fault injection

---

Errors are injected into the memories.

Advantages:

- **Predictability:** it is possible to reproduce every injected fault in time and space.
- **Reachability:** possible to reach storage locations within chips instead of just pins.
- **Less effort** than physical fault injection: no modified hardware.

Same quality of results?

# Summary

---

## Evaluation and Validation

- WCET estimation
  - Example: aiT (based on abstract interpretation)
- Energy models
  - Examples: Steinke's instruction set-based model, CACTI
- Risk and dependability analysis
  - Failure rates, reliability, MTBF, MTTF, MTTR
  - Fault tree analysis, FMEA
- Formal verification
  - Propositional, first order, higher order based techniques,
  - model checking
- Fault injection
  - Software and hardware-based techniques