technische universität
dortmund

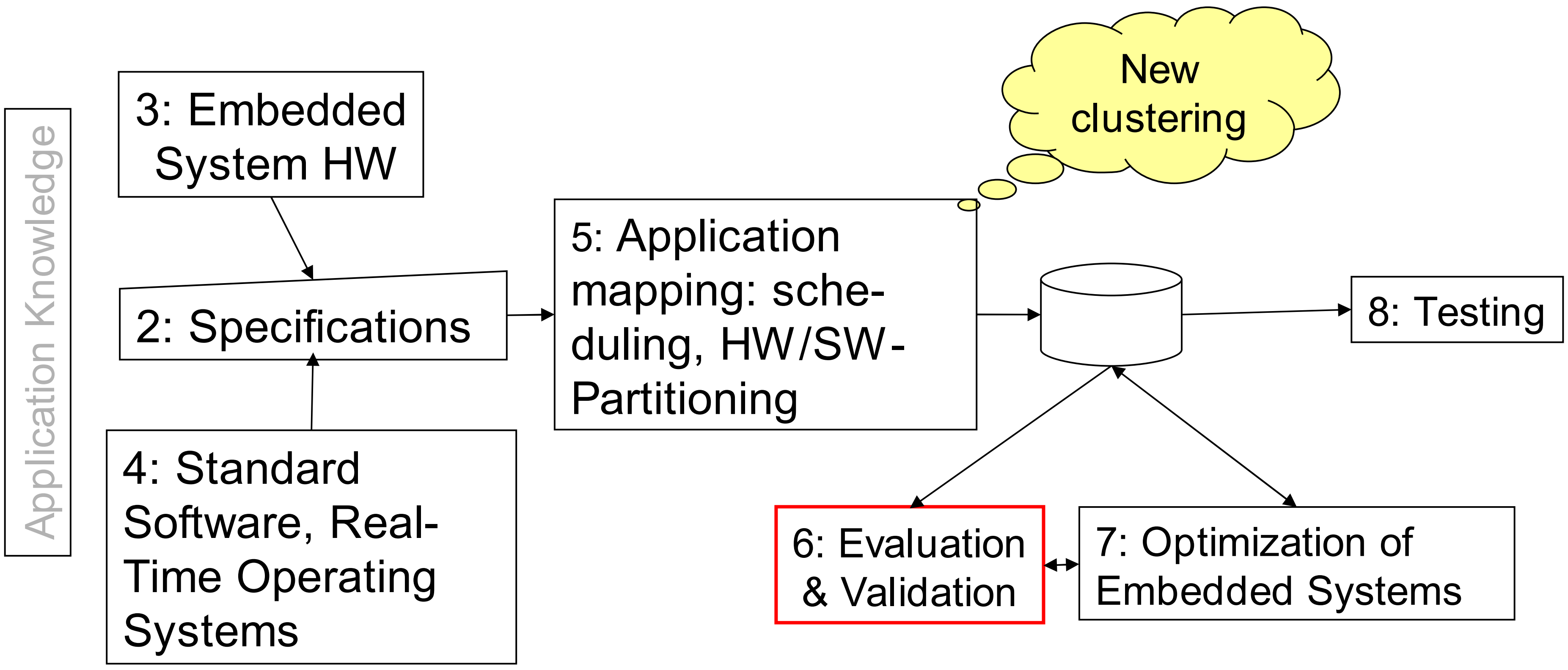
fakultät für informatik
informatik 12

Evaluation and Validation

Peter Marwedel
TU Dortmund, Informatik 12
Germany

2008/06/25

Structure of this course


Application Knowledge
3: Embedded System HW
2: Specifications
4: Standard Software, Real-Time Operating Systems
5: Application mapping: sche-duling, HW/SW-Partitioning
New clustering
8: Testing
6: Evaluation & Validation
7: Optimization of Embedded Systems

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*.

How to evaluate designs according to multiple criteria?

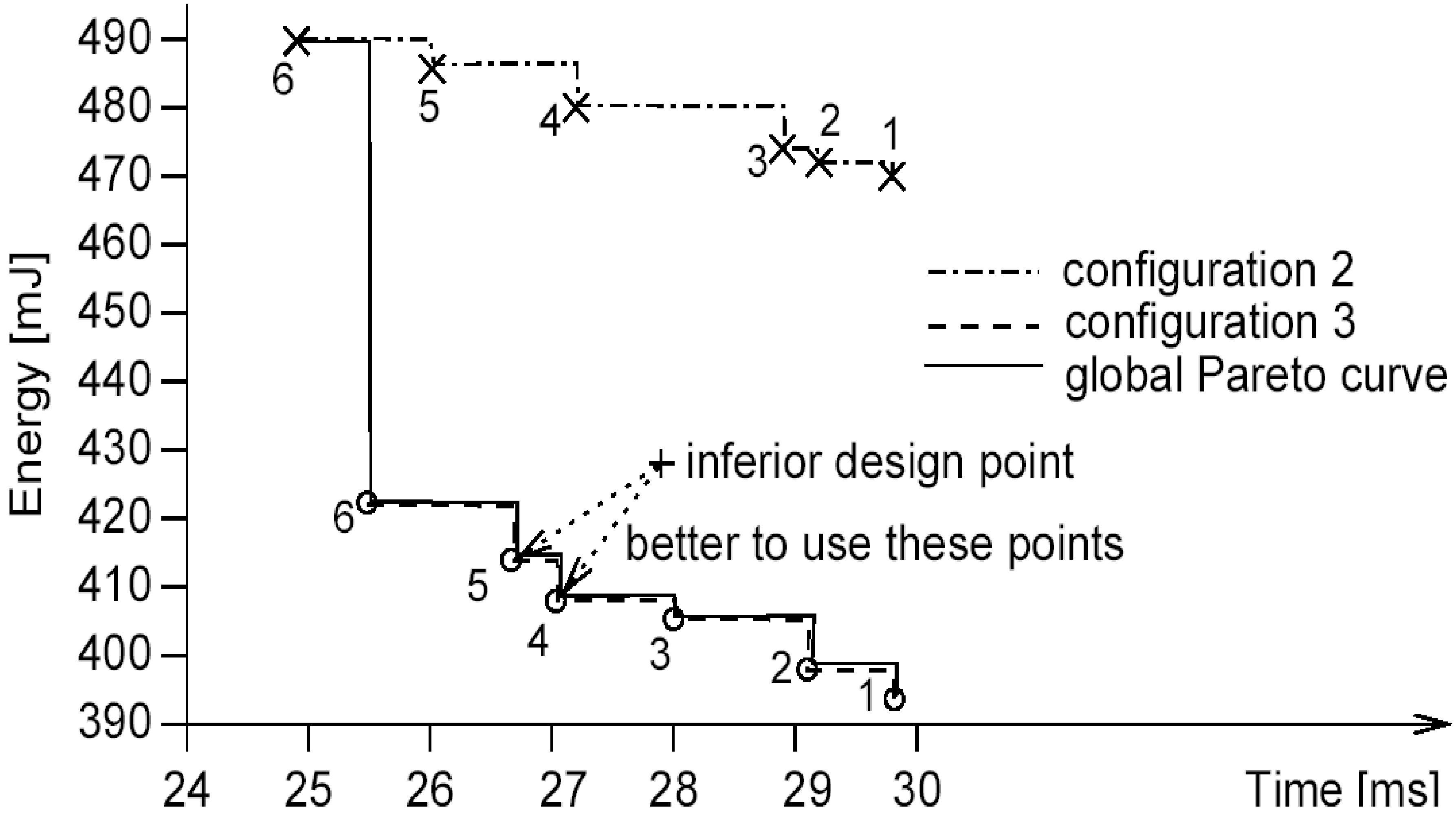
In practice, many different criteria are relevant for evaluating designs:

- (average) speed
- worst case speed
- power consumption
- cost
- size
- weight
- radiation hardness
- environmental friendliness ….

How to compare different designs? (Some designs are “better” than others)

Chapter 5.6.2

Pareto curves



Pareto points

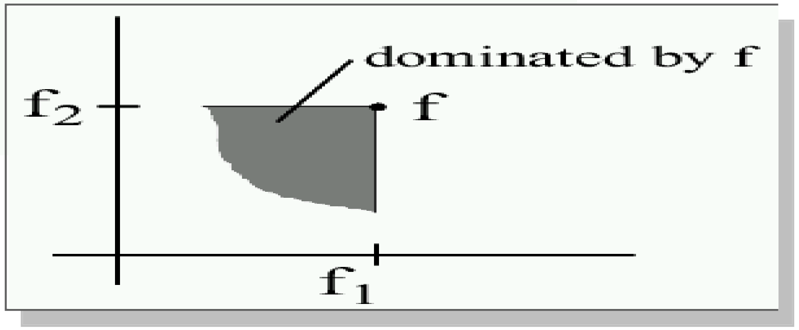
Definition: A (design) point J_i is **dominated** by point J_k, if J_k is equal or better than J_i in each criterion ($J_i \leq J_k$).

Definition: A (design) point is **Pareto-optimal** or a **Pareto point**, if it is not dominated by any other point.

Multi-objective Optimization

Definition 1 (Dominance relation)
Let $f, g \in \mathbb{R}^m$. Then f is said to dominate g, denoted as $f \succ g$, iff

1. $\forall i \in \{1, \ldots, m\} : f_i \geq g_i$
2. $\exists j \in \{1, \ldots, m\} : f_j > g_j$

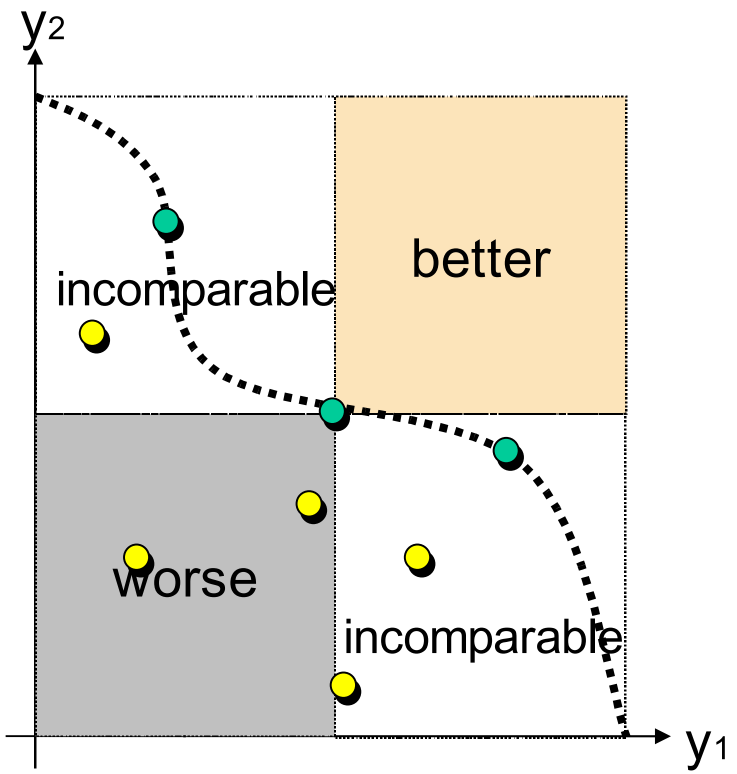


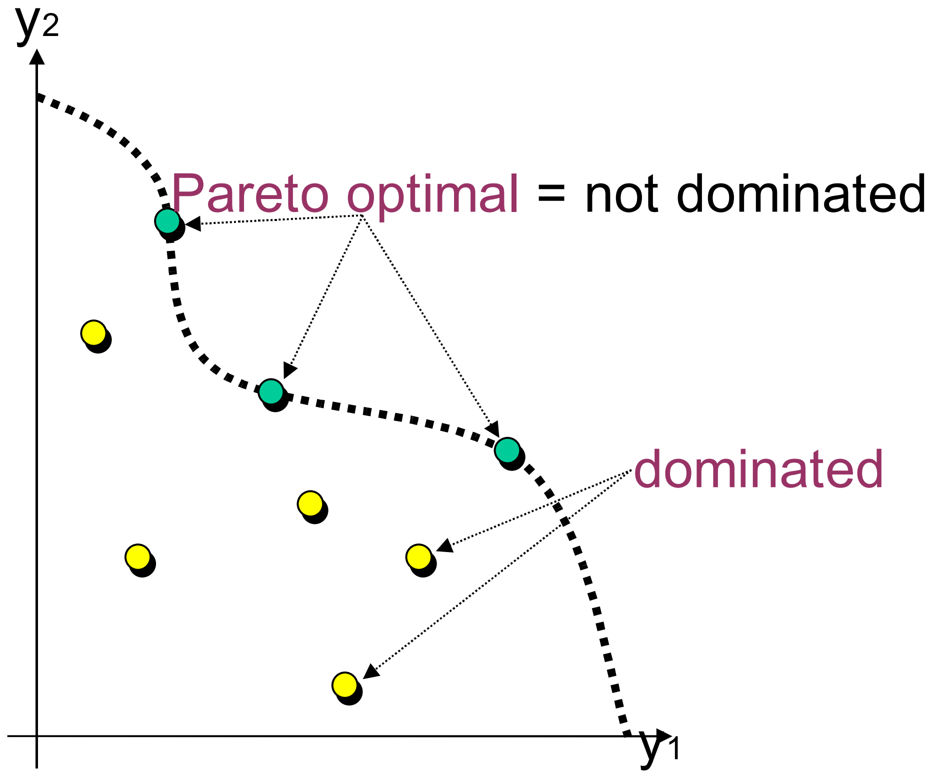
Definition 2 (Pareto set)
Let $F \subseteq \mathbb{R}^m$ be a set of vectors. Then the Pareto set $F^* \subseteq F$ is defined as follows: F^* contains all vectors $g \in F$ which are not dominated by any vector $f \in F$, i.e.

$$F^* := \{g \in F \mid \nexists f \in F : f \succ g\} \tag{1}$$

Multiobjective Optimization

Maximize $(y_1, y_2, \ldots, y_k) = f(x_1, x_2, \ldots, x_n)$





Pareto set = set of all Pareto-optimal solutions

Simulations

- Simulations try to imitate the behavior of the real system on a (typically digital) computer.
- Simulation of the functional behavior requires executable models.
- Simulations can be performed at various levels.
- Some non-functional properties (e.g. temperatures, EMC) can also be simulated.
- Simulations can be used to **evaluate** and to **validate** a design

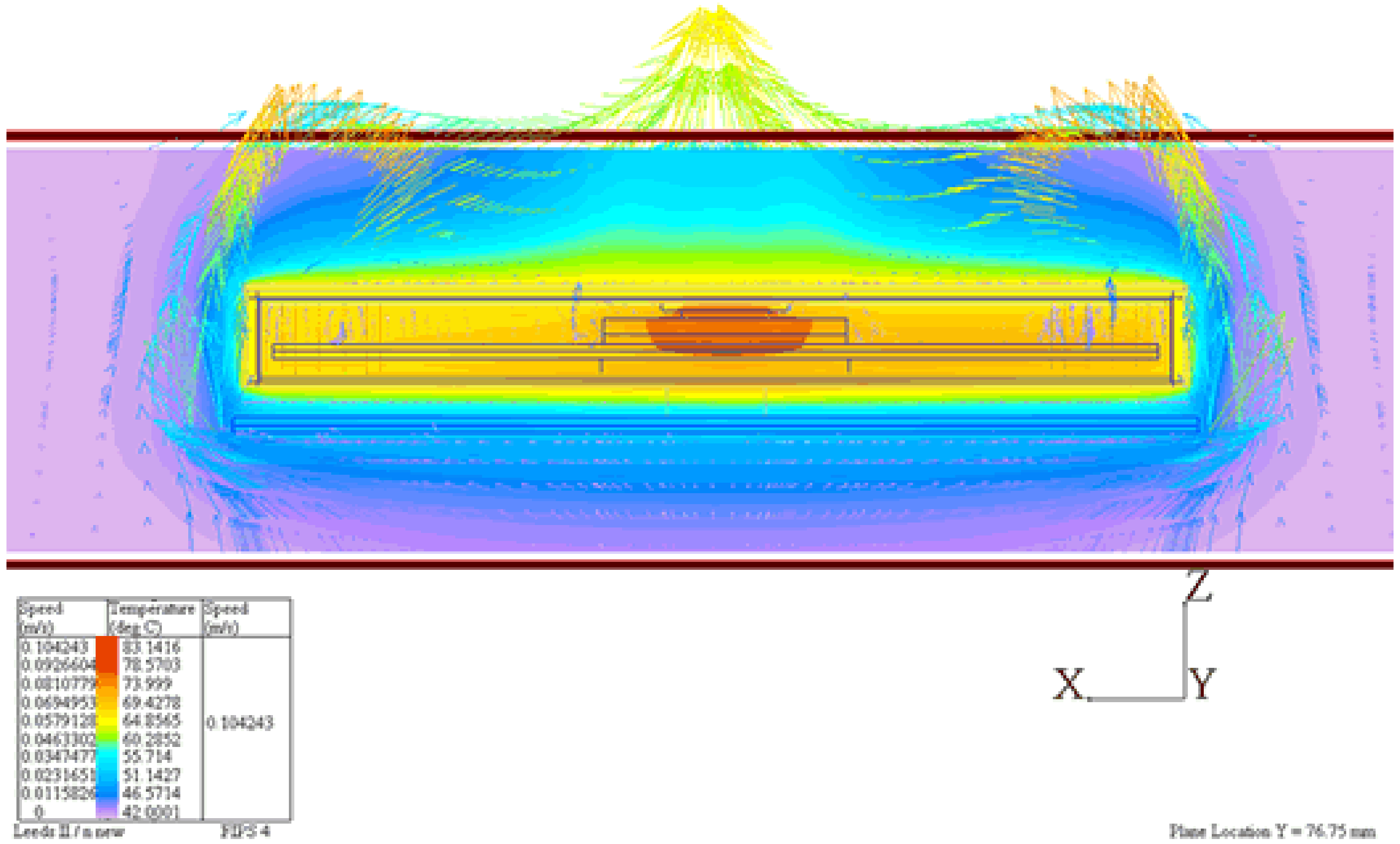
Validating functional behavior by simulation

Various levels of abstractions used for simulations:

- High-level of abstraction: fast, but sometimes not accurate
- Lower level of abstraction: slow and typically accurate
- Choosing a level is always a compromise

Non-functional behavior: Examples of thermal simulations (1)

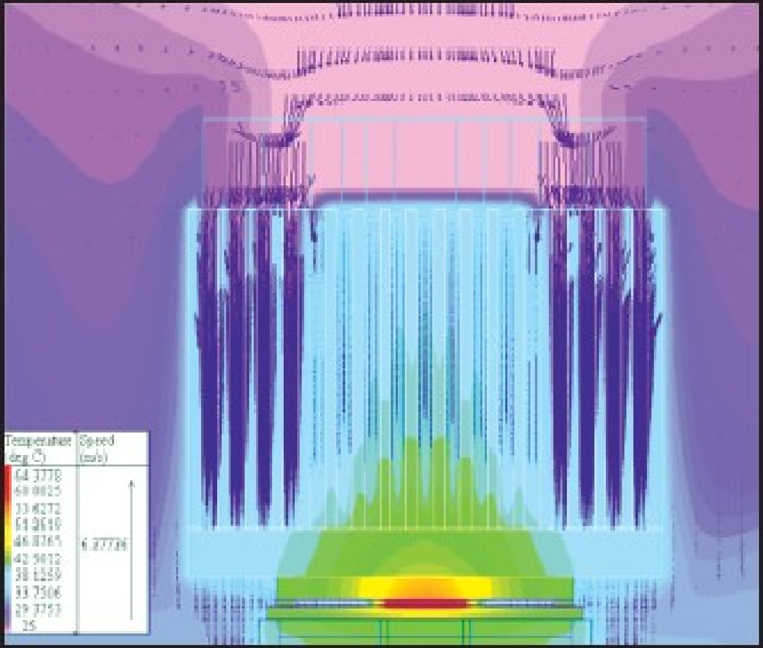
Encapsulated cryptographic coprocessor:



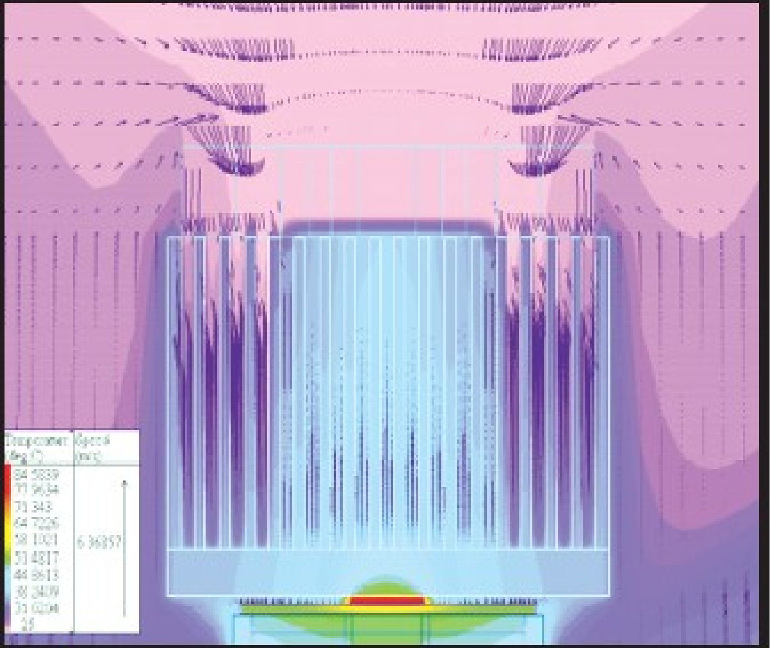
Source: http://www.coolingzone.com/Guest/News/NL_JUN_2001/Campi/Jun_Campi_2001.html

Examples of thermal simulations (2)

Microprocessor



▲ Flomerics image showing the thermal solution with a metal lid.

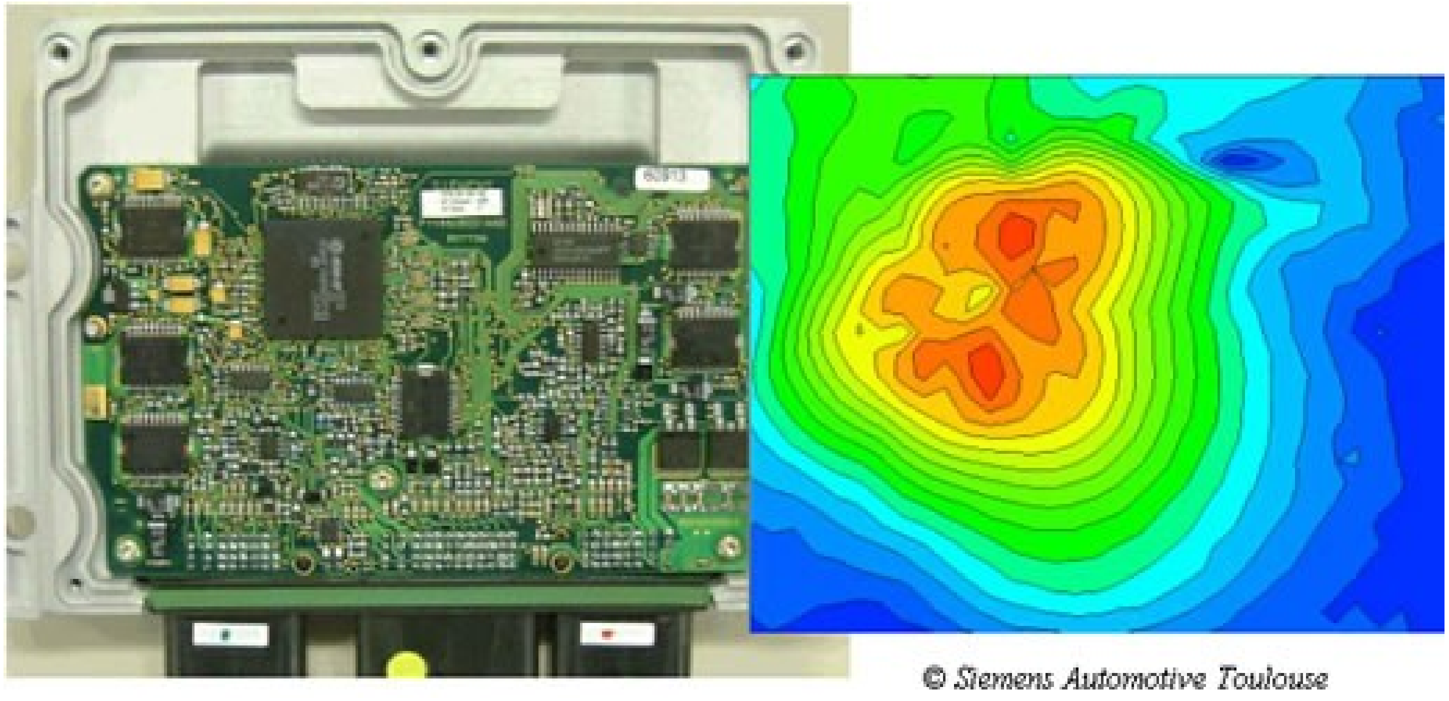


▲ Flomerics image showing the thermal solution without a metal lid.

Source: http://www.flotherm.com/applications/app141/hot_chip.pdf

EMC simulation

Example: car engine controller

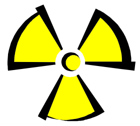


Red: high emission
Validation of EMC properties often done at the end of the design phase.

Source: http://intrage.insa-tlse.fr/~etienne/emccourse/what_for.html

Simulations Limitations

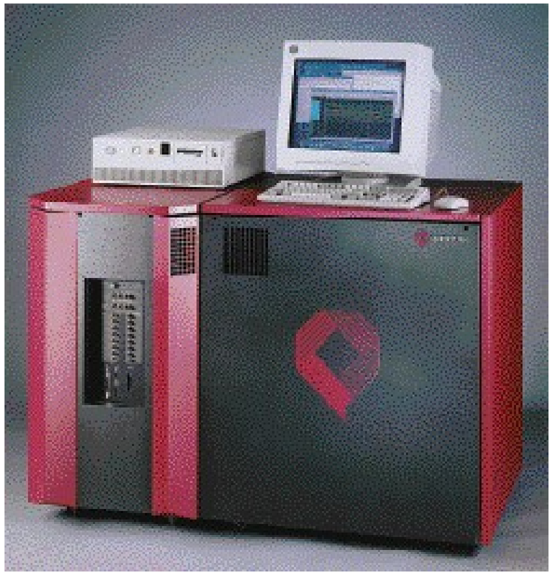
- Typically slower than the actual design.
  ☞ **Violations of timing constraints** likely if simulator is connected to the actual environment
- Simulations in the real environment may be **dangerous**
- There may be huge amounts of data and it may be impossible to simulate enough data in the available time.
- Most actual systems are too complex to allow simulating all possible cases (inputs).
  Simulations can help finding errors in designs, but they cannot guarantee the absence of errors.



Chapter 6.3

Rapid prototyping/Emulation

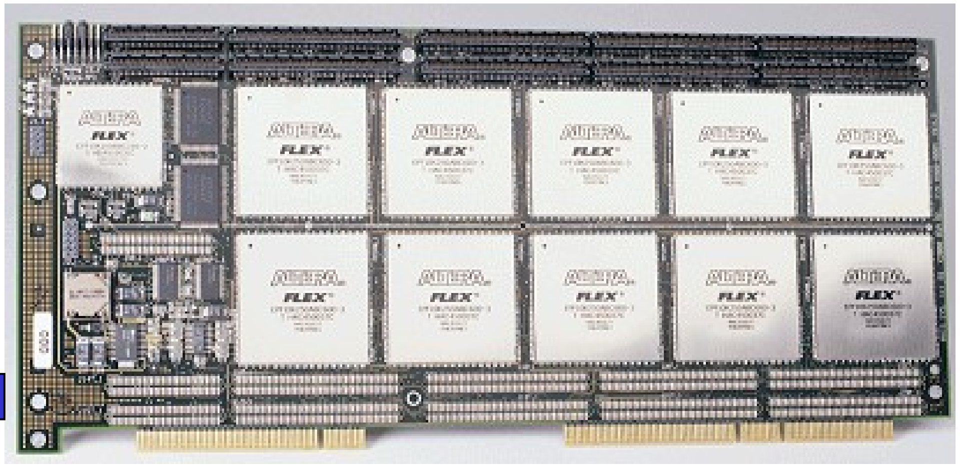
- Prototype: Embedded system that can be generated quickly and behaves very similar to the final product.
- May be larger, more power consuming and have other properties that can be accepted in the validation phase
- Can be built, for example, using FPGAs.



Example: Quickturn Cobalt System (1997), ~0.5M$ for 500kgate entry level system

Source & ©: http://www. eedesign. com/editorial/1997/ toolsandtech9703.html

Example of a more recent commercial emulator



[www.verisity.com/images/products/xtremep{1|3}.gif]

Chapter 6.6

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.

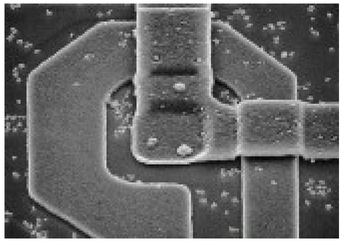
Same quality of results?

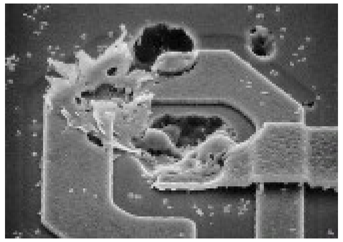
Chapter 6.7

Risk- and dependability analysis

Example : metal migration @ Pentium 4

www.jrwhipple.com/computer_hangs.html





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

FIT: failure-in-time unit for failure rate (=1/MTTF≈1/MTBF);

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

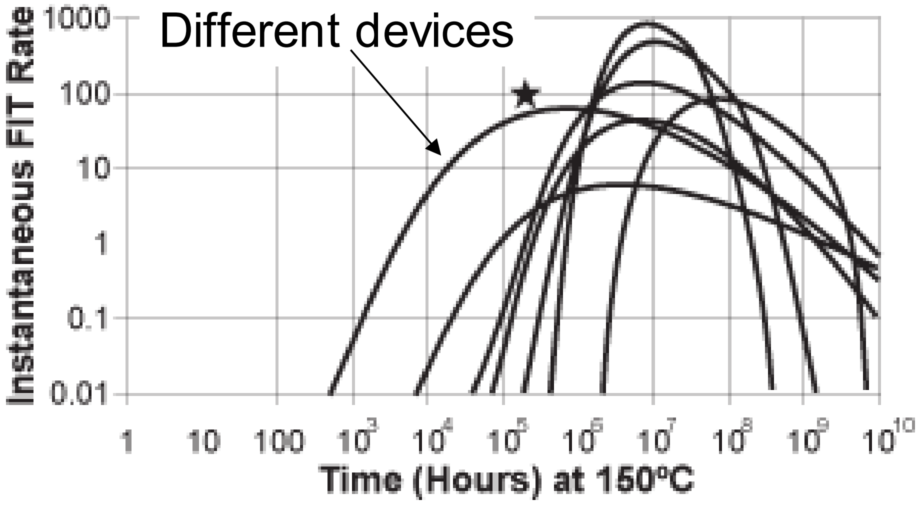
Damages are resulting from hazards.

For every damage there is a severity and a probability.

Several techniques for analyzing risks.

Actual failure rates

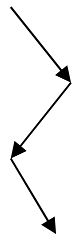
Example: failure rates less than 100 FIT for the first 20 years of life at 150°C @ TriQuint (GaAs)
[www.triquint.com/company/quality/faqs/faq_11.cfm]



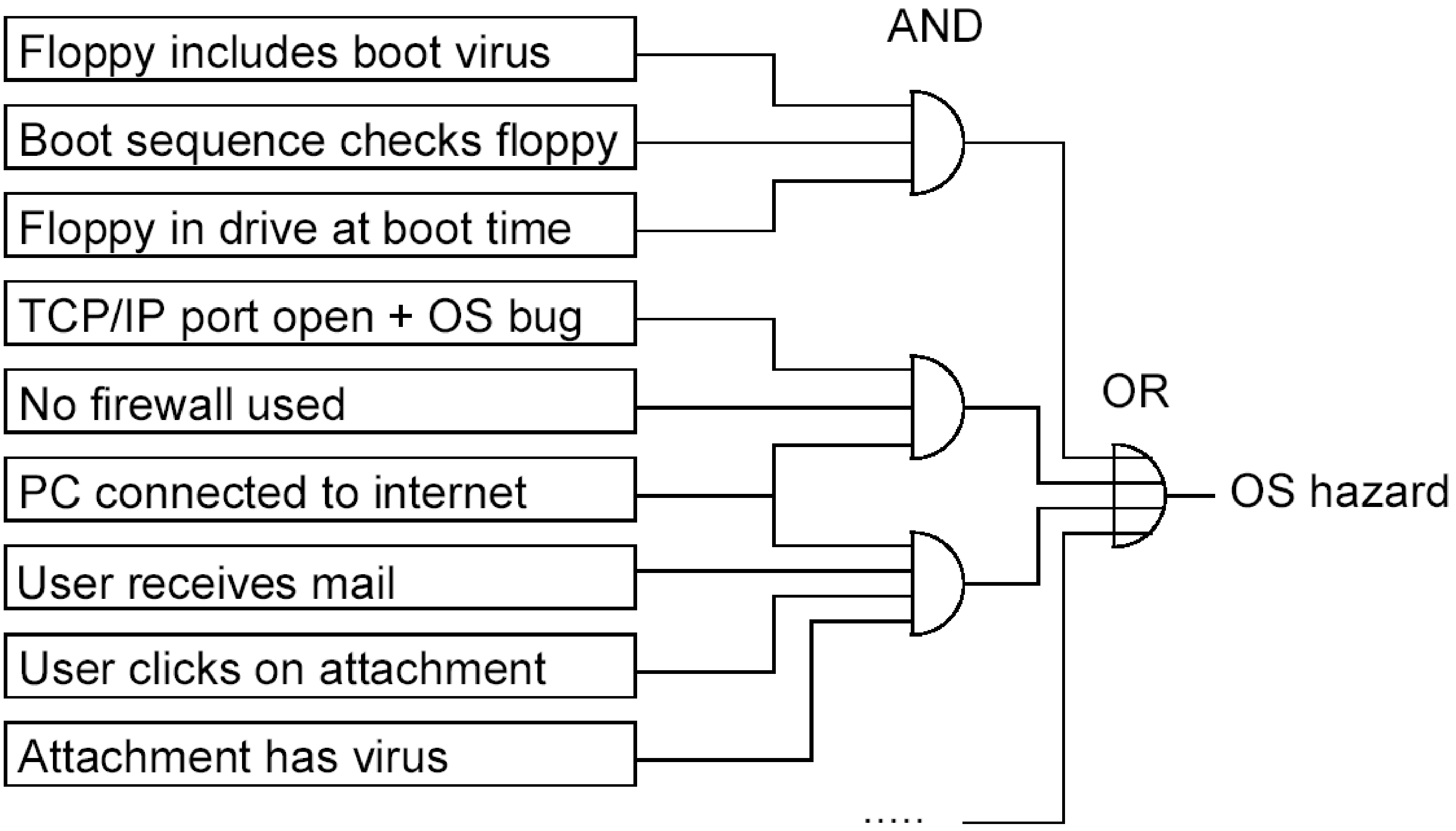
Target: Failures rates of systems ≤ 1FIT
Reality: Failures rates of circuits ≤ 100 FIT
☞redundancy is required to make a system more reliable than its components
Analysis frequently works with simplified models ☞

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


AND
Floppy includes boot virus
Boot sequence checks floppy
Floppy in drive at boot time
TCP/IP port open + OS bug
No firewall used
PC connected to internet
User receives mail
User clicks on attachment
Attachment has virus
OR
OS hazard
.....

Limitations

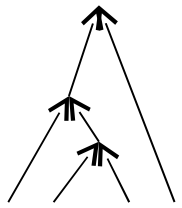
The simple AND- and OR-gates cannot model all situations. For example, their modeling power is exceeded if shared resources of some limited amount (like energy or storage locations) exist.
Markov models may have to be used to cover such cases.

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.

Component	*Failure*	*Consequences*	*Probability*	*Critical?*
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.

Chapter 6.8

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*(*number of BDD-nodes*) (equivalence check for sums of products is NP-hard). #(BDD-nodes) not to be ignored!
- Many functions can be efficiently represented with BDDs. In general, however, the #(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 ∃ and ∀.
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)

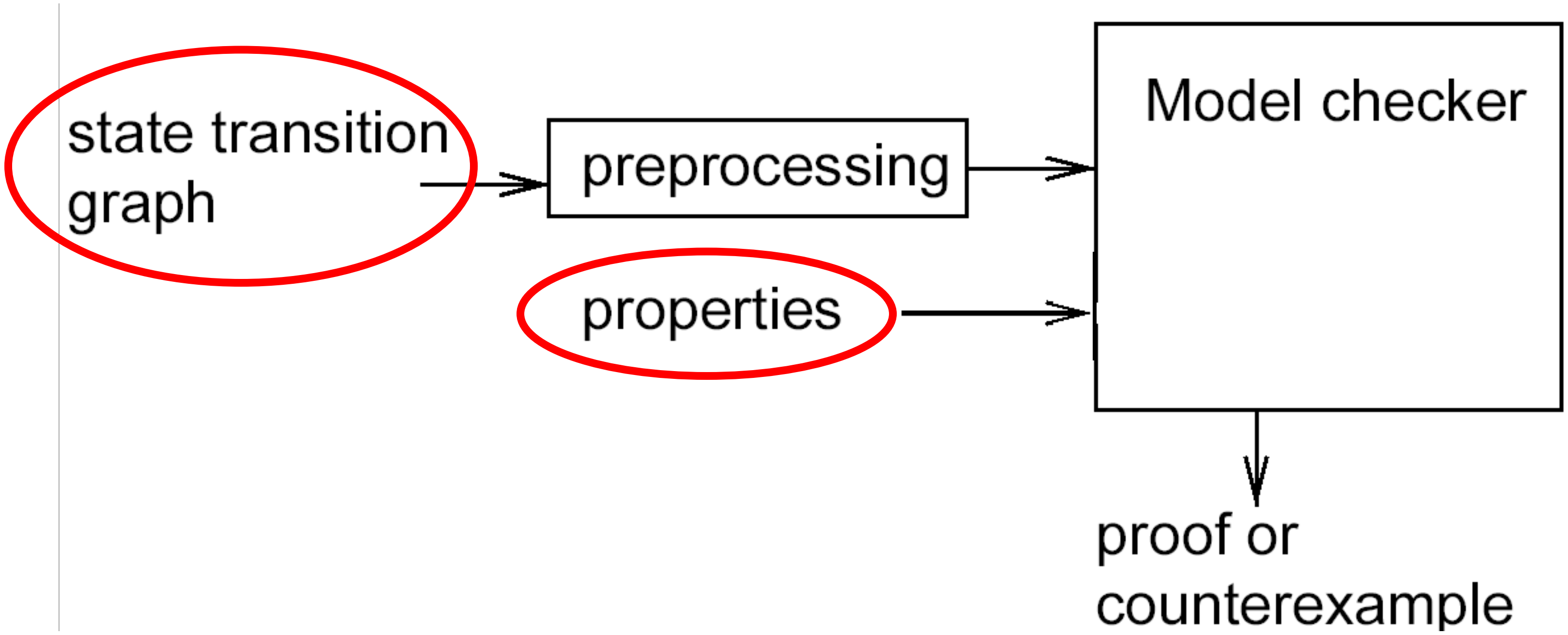
Higher order allows functions to be manipulated like other objects.
For higher order logic, proofs can hardly ever be automated and typically must be done manually with some proof-support.

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 $s°$ iff f_1 holds in all successor states of $s°$
- 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.

Summary

- Simulation
  - functional
  - non-functional validation
- Emulation
- Formal verification
  - …, Model checking