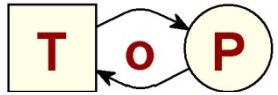
11.11.11 Eindhoven
Kees van Hee Symposium

From *Programs as Models* to *Models as Programs*



Theory of Programming

Wolfgang Reisig

Humboldt-Universität zu Berlin

Prof. Dr. W. Reisig

… from the IBM-Lab Zürich (2007)

Business Driven Development

code only	*code visualization*	*Round trip Engineering*	*model centric*	*the model is the code*
	model	*model*	*model*	*model*
code	*code*	*code*	*code*	

This talk:

60 years models of *behavior*

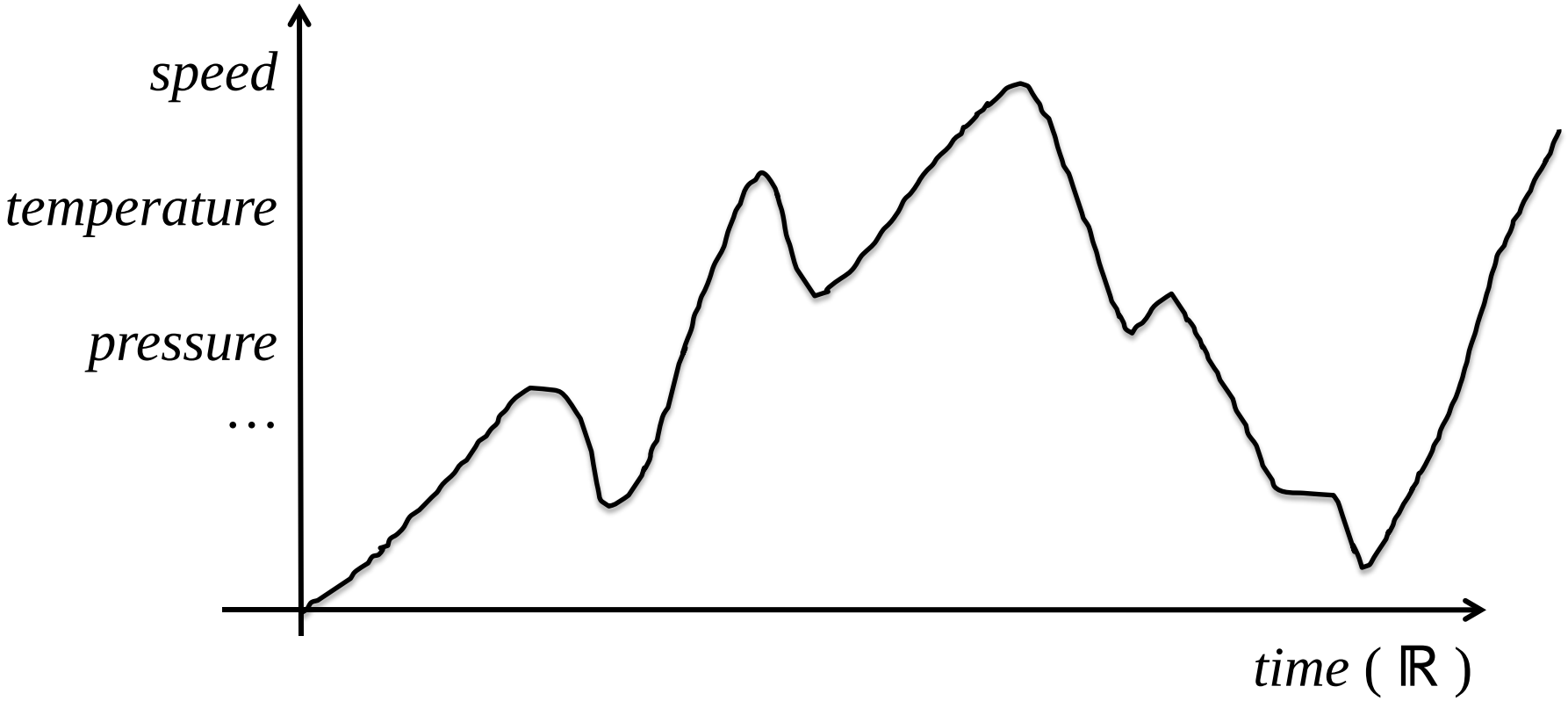
one highlight per decade

what *I* find relevant

what I hope Kees may find enlightening …

until 1950ies:

Modeling of behavior in physics and technics:
A model is a continuos function



the early years of Kees …

How model (represent) Euclid‘s algorithm?

as a program! *“pseudocode”*

```
function gcd(a, b)
if  a = 0  return b
while  b ≠ 0
        if  a > b  then  a := a − b
                   else  b := b − a
return a
```

1950ies:
Programs as models

Ingrediences:
finite set *var* of stores ("*variables*");
set *val* of "representable" *values;*
states: s: *var* $\rightarrow$ *val;*
steps: s $\xrightarrow{t}$ s' updates s at one argument, "effectively" realizable;

A *run* is a finite sequence
$s_0 \xrightarrow{t1} s_1 \xrightarrow{t2} s_2 \xrightarrow{t3} \ldots \xrightarrow{tn} s_n$
of steps;

A *system (model)* describes a set of *runs.*

How model (represent) the pebble game ?

Every 5 year child can play that game.

The rules:

Repeat as long as possible:

(1) remove 2 pebbles, a, b

return one pebble, c .

(2) If a *and* b *are coloured alike:*

c *ist black;*

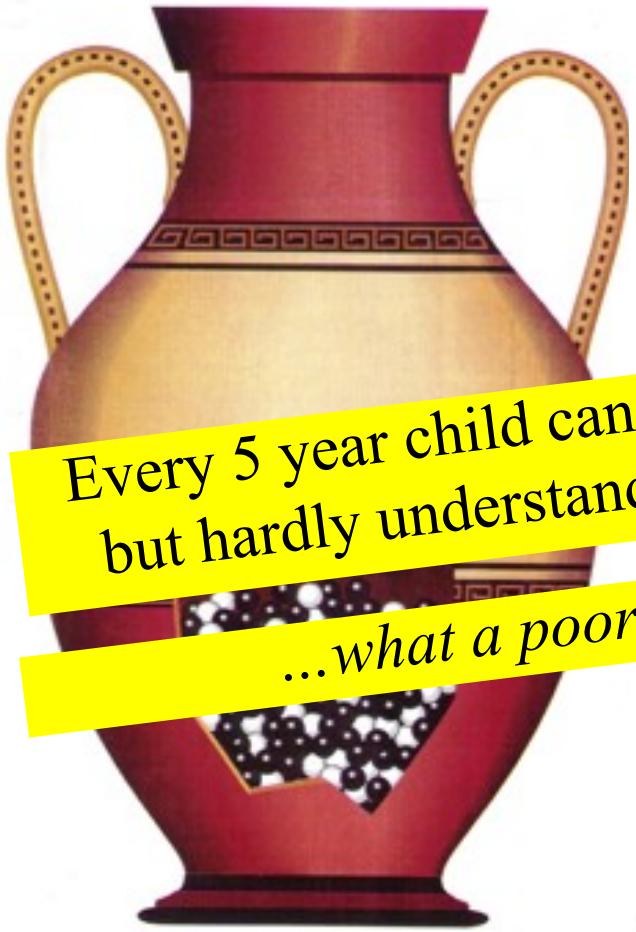
otherwise, c *is white.*

… how would you model this behavior ?

How model (represent) the pebble game ?

initial number of black [white] pebbles

integer variables



b := B; w := W

; do

to run this program
you must know B and W
are *variables*,
computing …
urn, no *pebbles*,
no *remove*, no *return* …

w := w − 2; b := b + 1

od

Every 5 year child can play that game
but hardly understands the program

…what a poor model!

remember

1950ies: Programs as models

Ingrediences:

finite set *var* of stores ("*variables*");

set *val* of "representable" *values;*

states: s: *var* $\rightarrow$ *val;*

steps: s $\xrightarrow{t}$ s' updates s at one argument effectively" realizable;

… we need far more general models!

far too special!

A *run* is a finite sequence

$s_0 \xrightarrow{t1} s_1 \xrightarrow{t2} s_2 \xrightarrow{t3} \ldots \xrightarrow{tn} s_n$

of steps;

A *system (model)* describes a set of *runs*.

some obvious observations

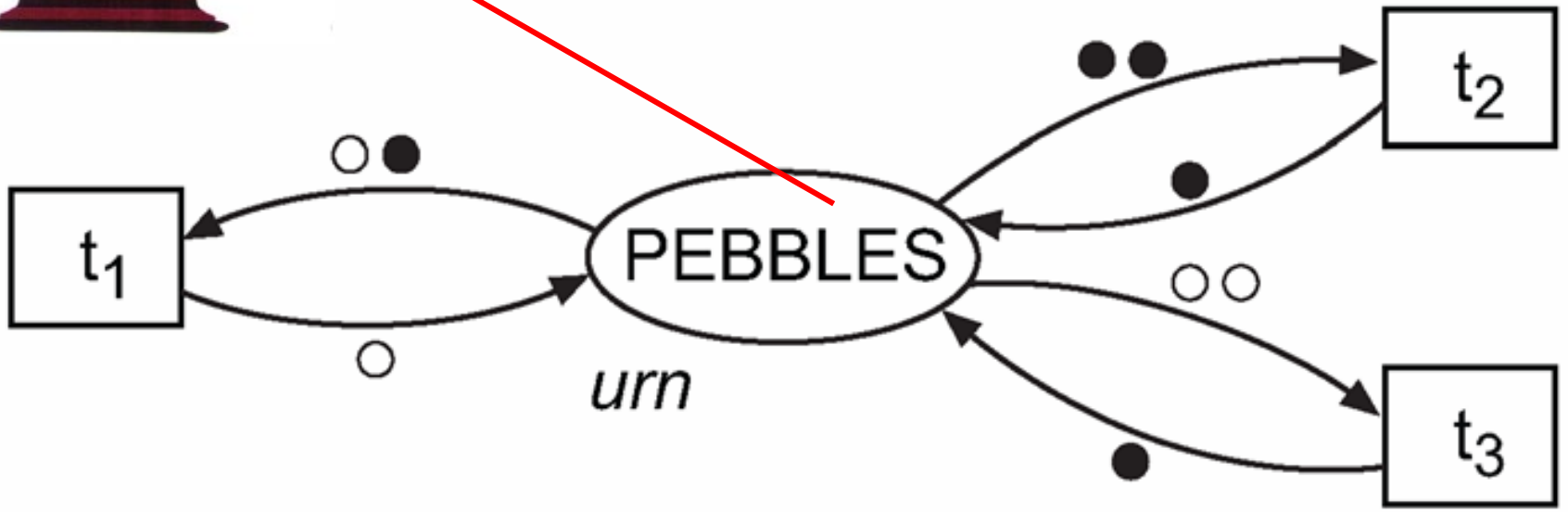
- A system is usually *distributed* !
- An event has *locally confined* causes and effects.
- *Global states* are a (misleading) fiction.
- An event requires no operating system!
  („*e happens*“ ; not „*e is executed*“)
- A basic step preserves information (is *reversible*).

- **Model those aspects!**
- **Exploit them in analysis**

1960ies: Petri nets

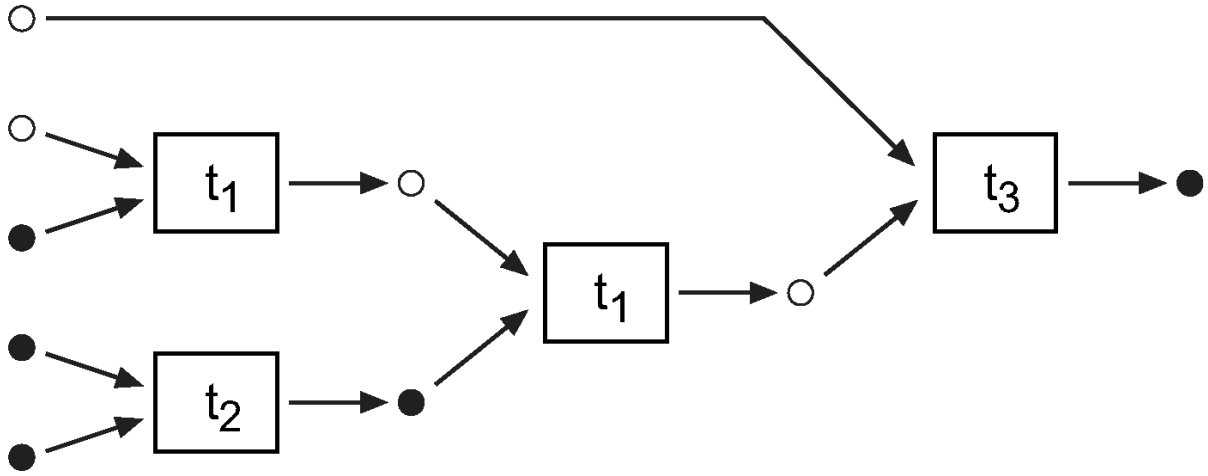
example: the pebble game

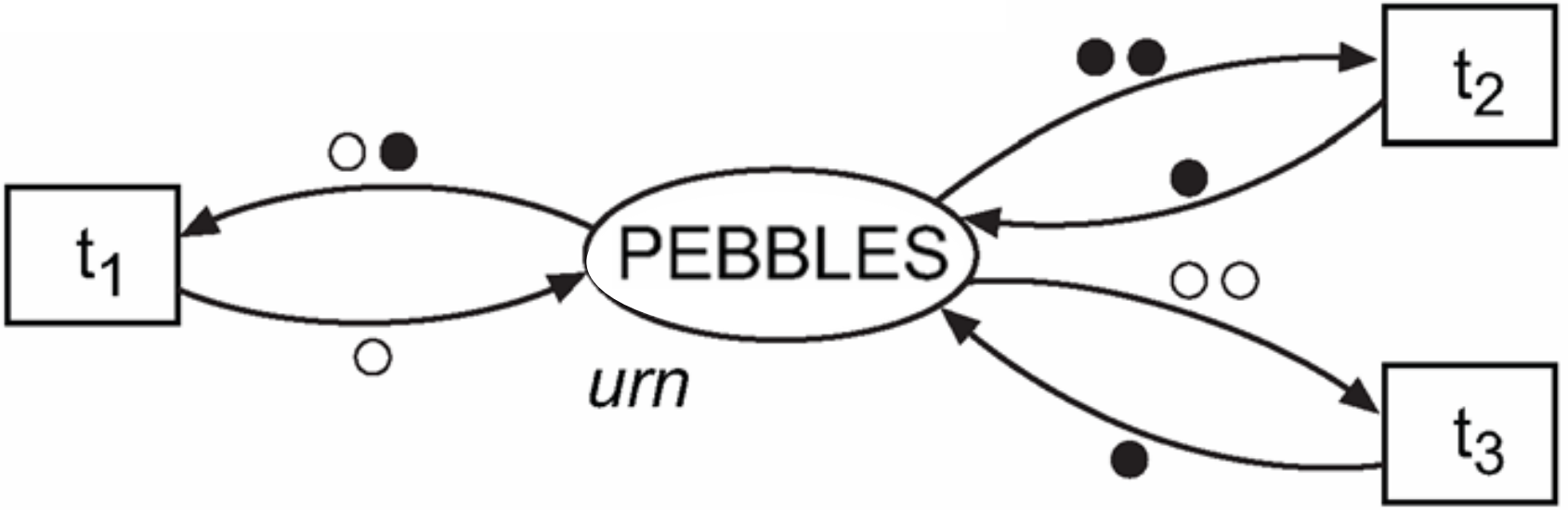
A schema for *infinitely many* Petri nets:
PEBBLES : a symbol, to be interpreted by



A typical run

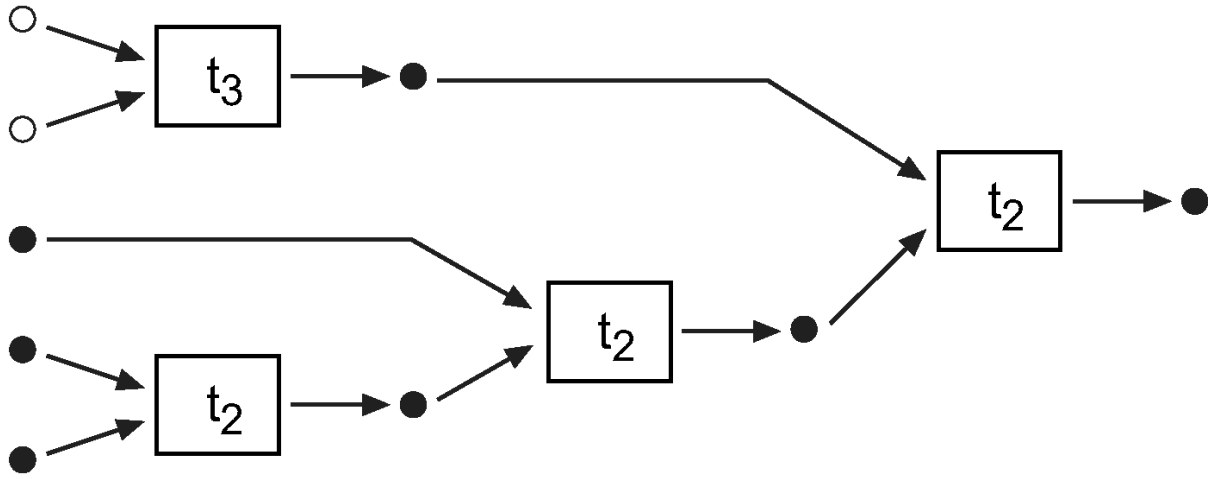
Each single run is *distributed*!

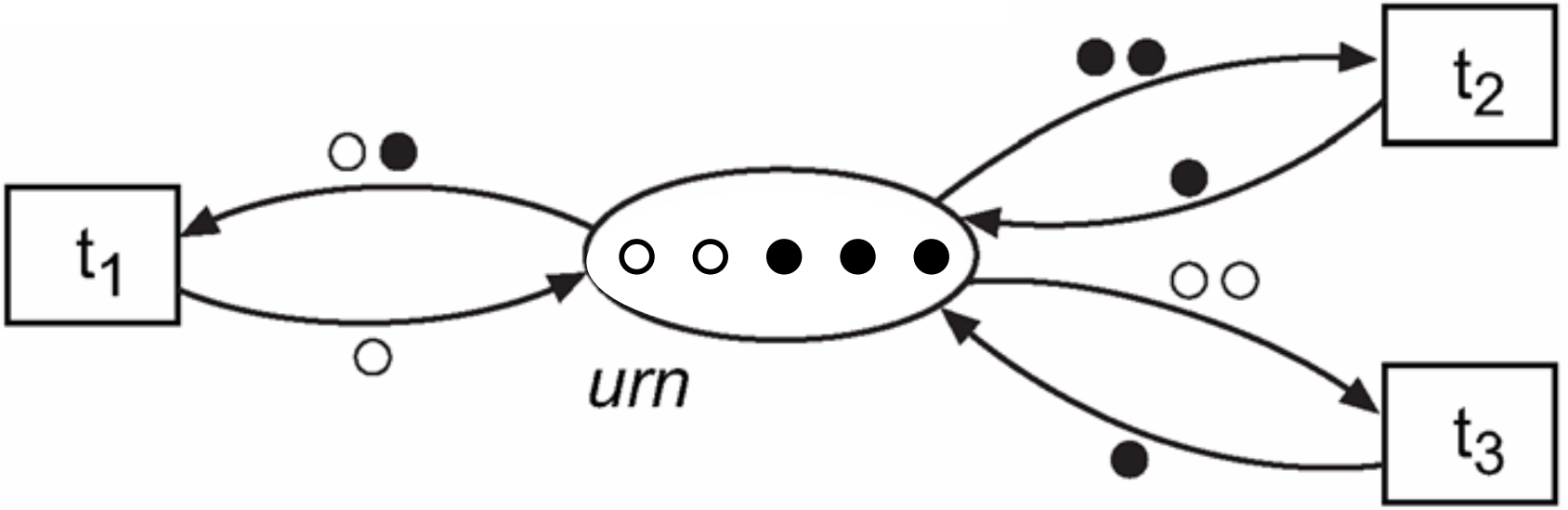




Another run

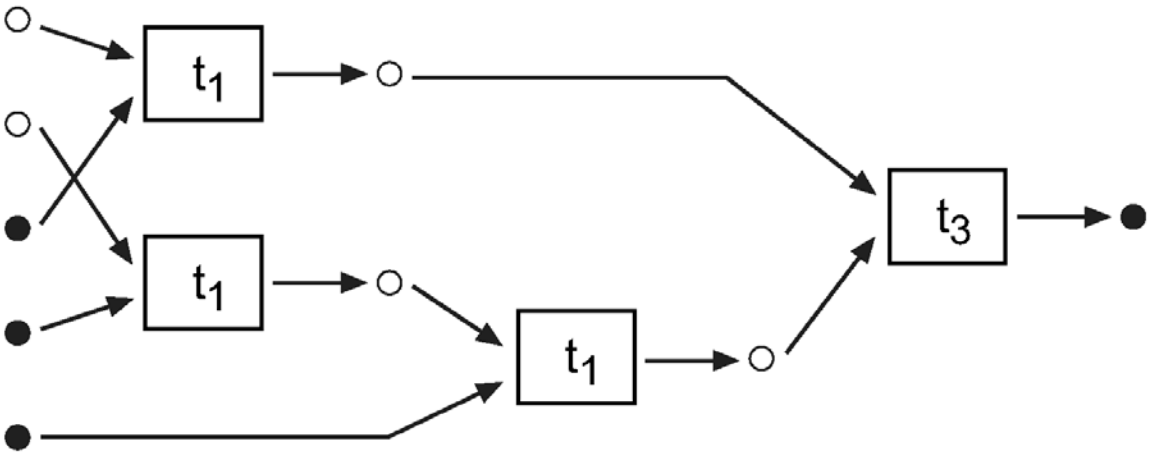
Each single run is *distributed*!

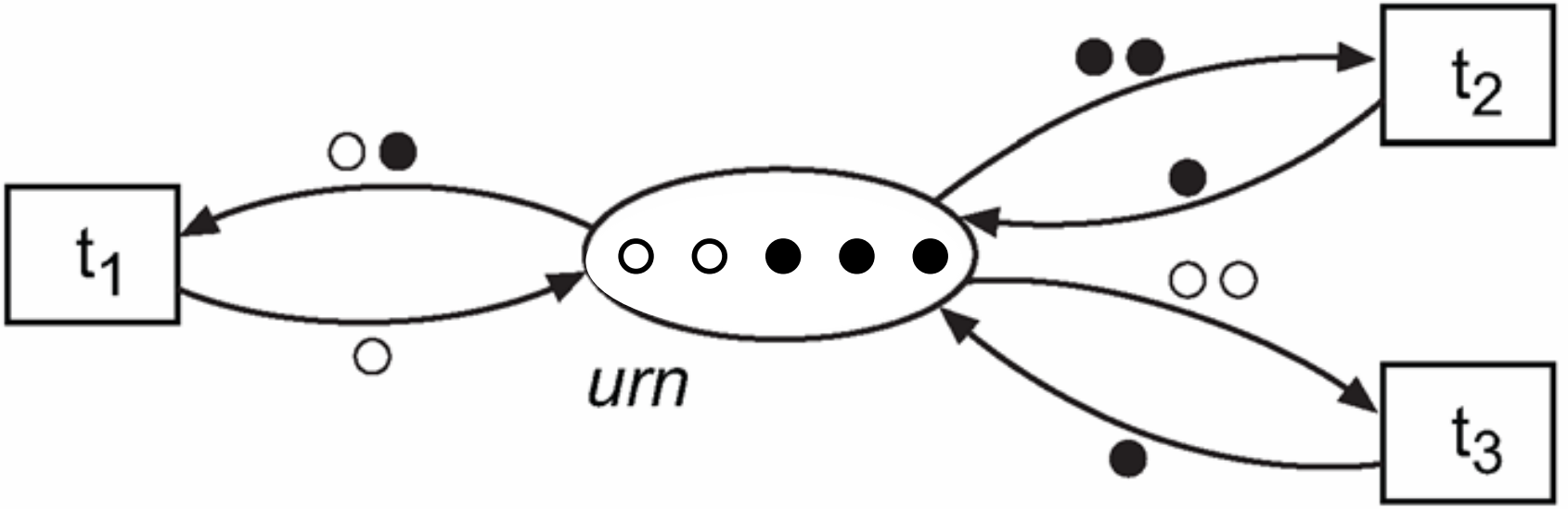




yet another one

Each singele run is *distributed*!





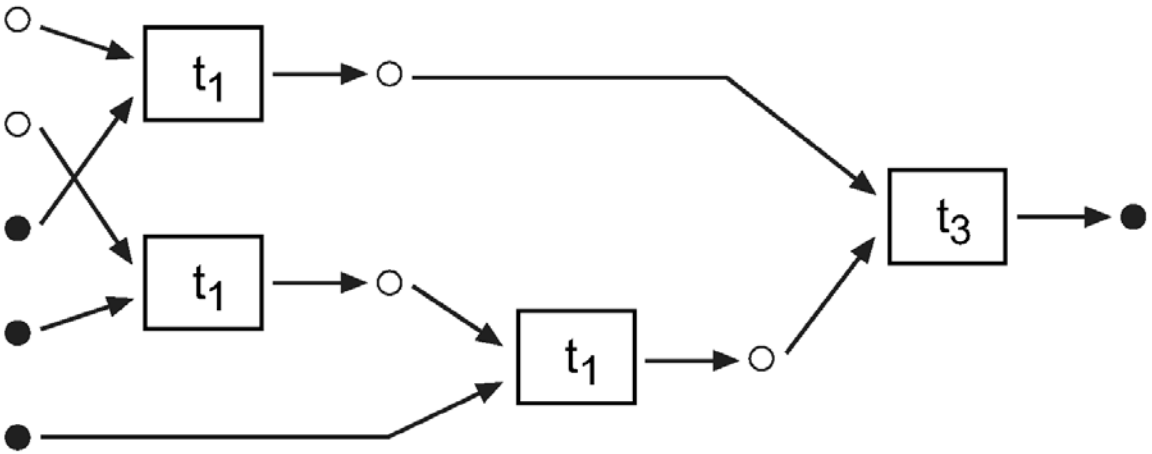
occurrences of transitions

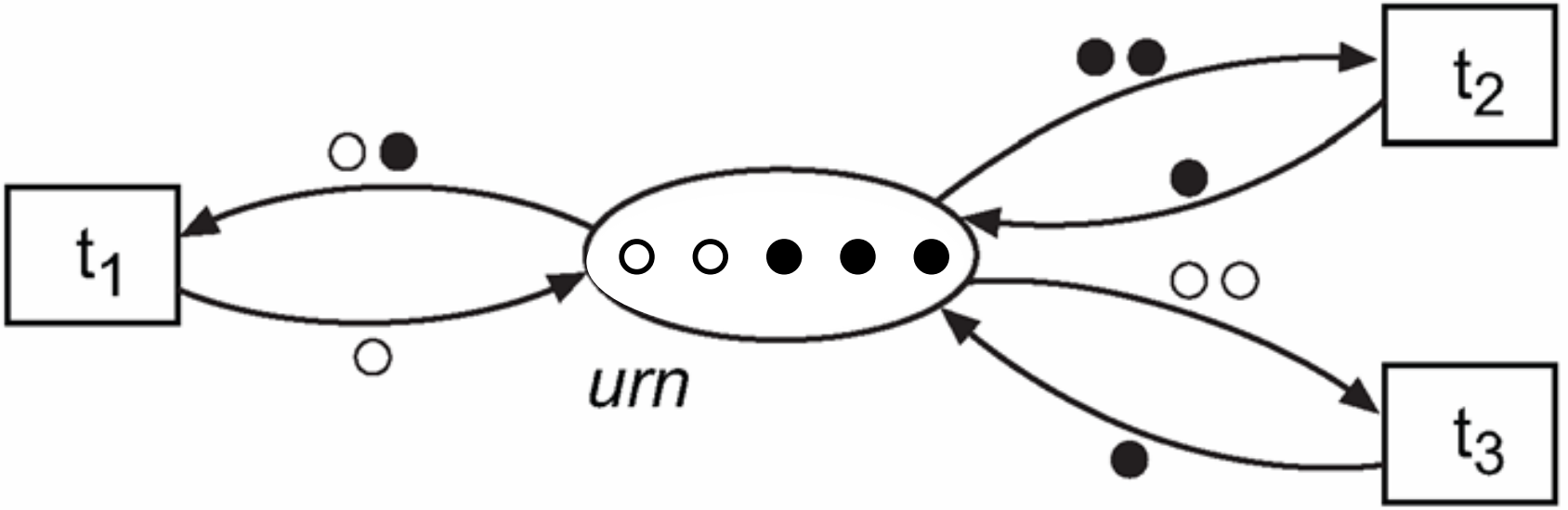
not:

„t_1 *is executed*“

but:

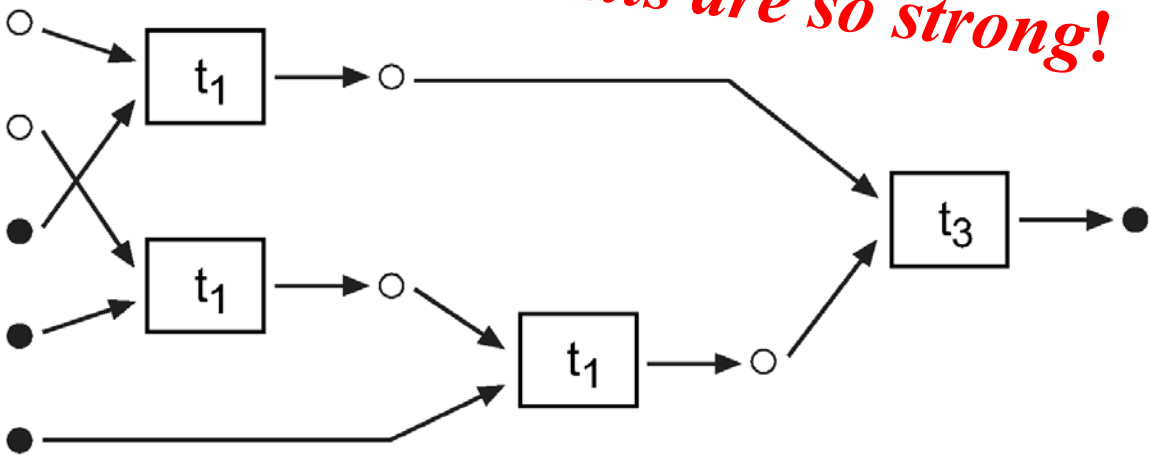
„ t_1 *occurs*“





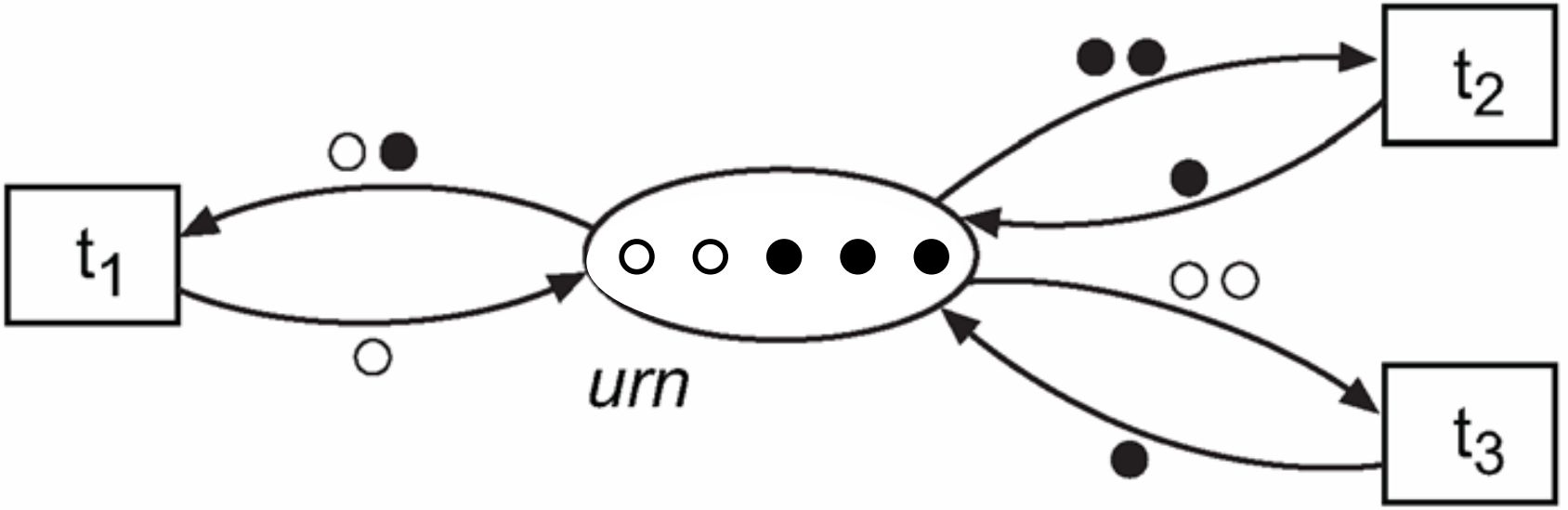
A basic step is *reversible*

This is why
Petri Net invariants are so strong!



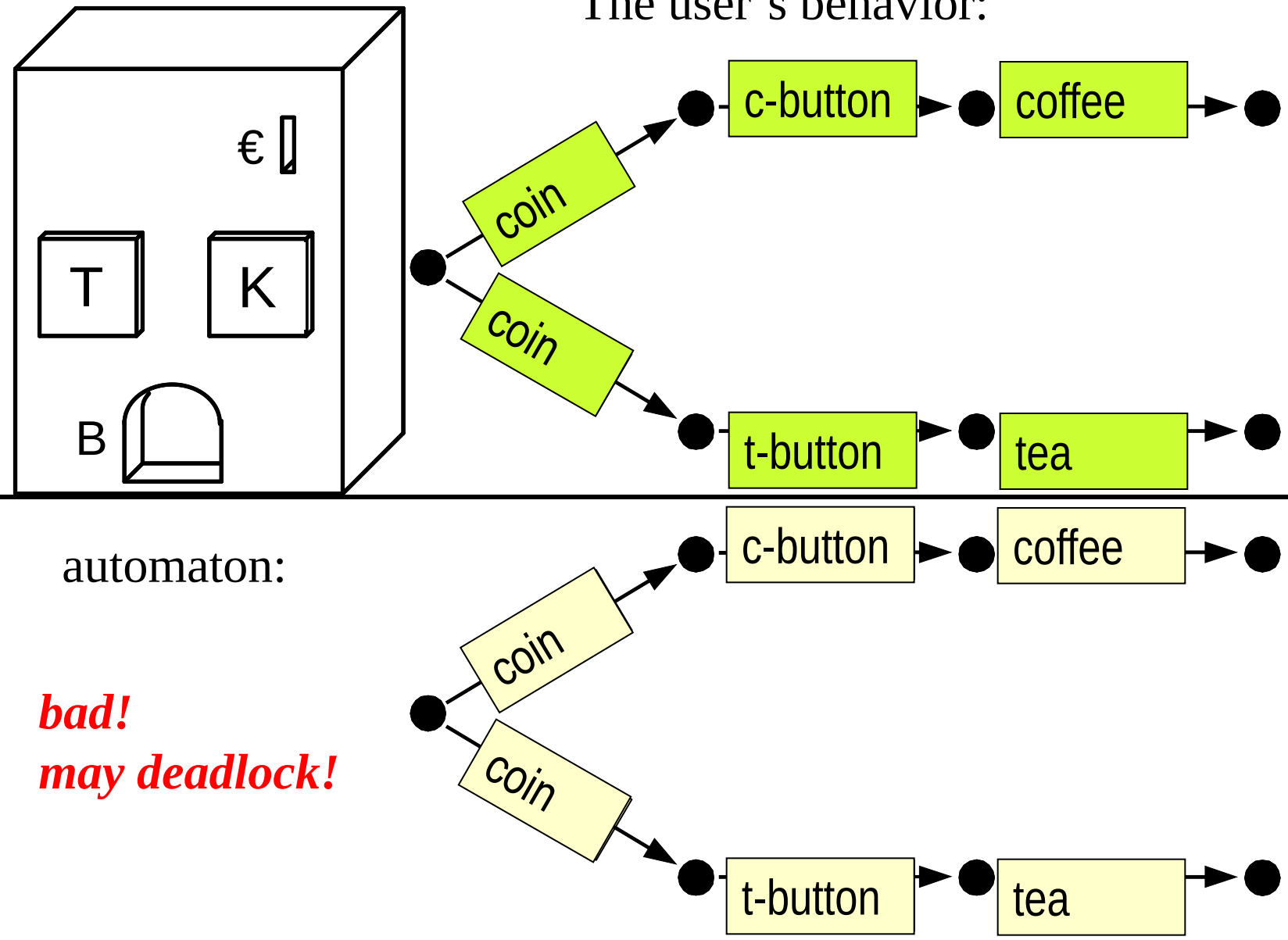
in a step $s_0 \xrightarrow{t} s_1$,
s_1 and t imply s_0 .

counterexample:
$s_0 \xrightarrow{x := y+z} s_1$



1970ies : Process algebra

The user‘s behavior:



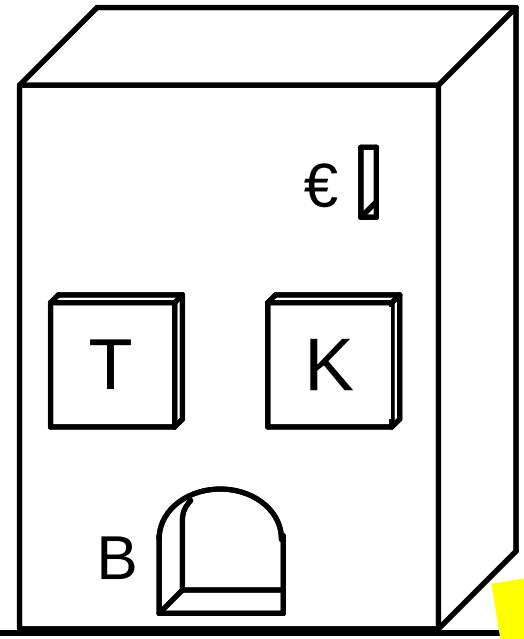
automaton:

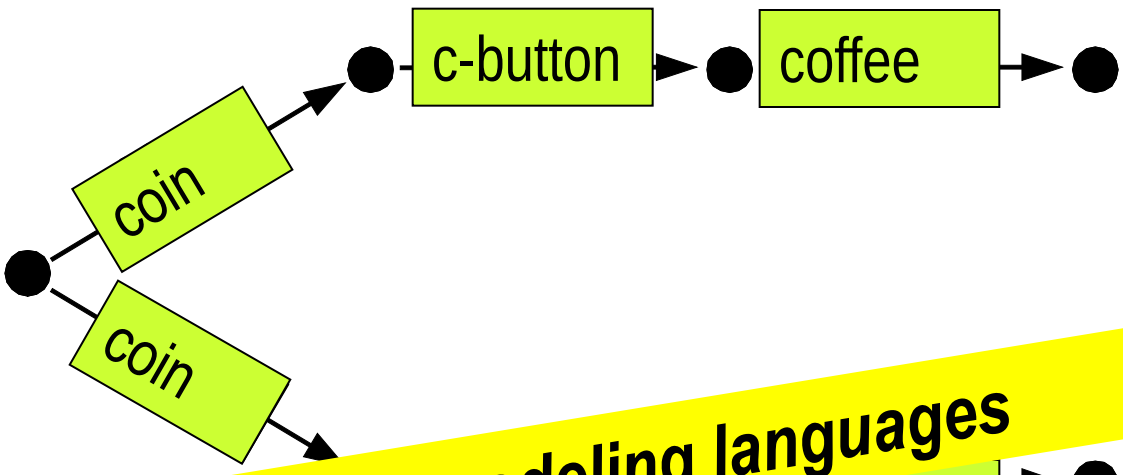
bad!

may deadlock!

1970ies : Process algebra

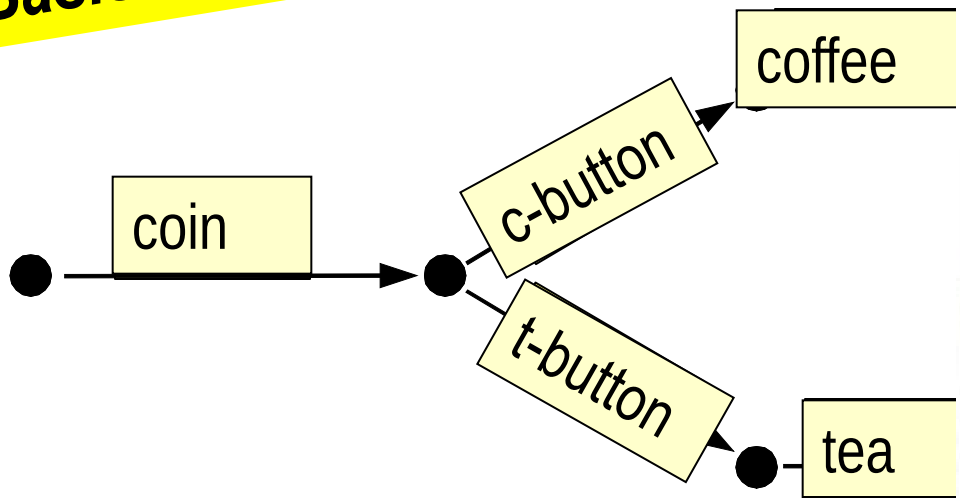
The user‘s behavior:



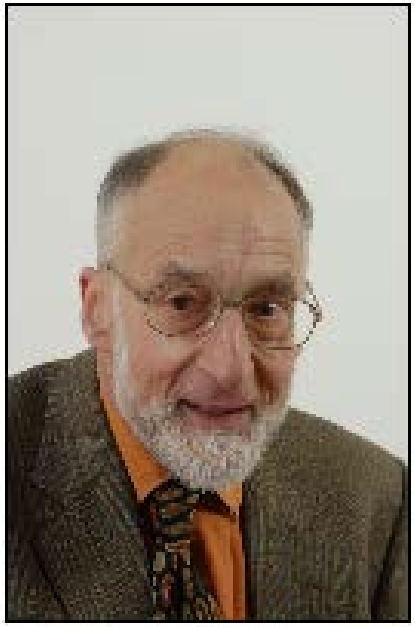


Basis of many modeling languages

automaton:



axcellent!
terminates
properly!



1980ies : Temporal Logic

- propositional logic +
  „always … “ „eventually… “ („next state … “) ,
  to be interpreted in a transition system

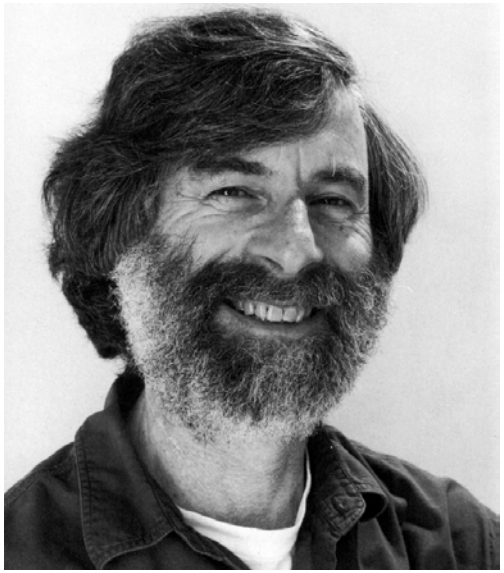
- Model Checking: amazingly successful

(Temporal) logic specifications: It would be phantastic ...

- „Composition is conjunction“
- „Implementiation (refinement) is implication“

… Why doesn‘t this function properly?

Leslie Lamport tried by means of TLA



a hour/minute clock

hr:	22	22	23	23	23	...	23	00	00	00	00	...
min:	58	59	00	01	02	...	59	00	01	02	03	...

If I only need an hour clock …

idea: mask the minute

a hour/minute clock

hr:	22	22	23	23	23 ...	23	00	00	00	00 ...
min:	58	59	00	01	02 ...	59	00	01	02	03 ...

If I only need an hour clock

idea: mask the minute

The hour/minute clock *stutters* in the hour dispaly!

The - intended – hour clock does not !

not too fascinating…

1990ies:
A model is not necessarily executable

. . . we should have achieved
a mathematical model of computation,
perhaps highly abstract
in contrast with the concrete nature
of paper and register machines,
but such that programming languages
are merely executable fragments
of the theory . . .

Robin Milner
2005

… is this an algorithm?

- cooking recipe
- pattern for knitting
- perpendicular at a point in a li
- solicit m
- elevator
- business

fundamental contribution to a theory of modeling
… still widely ignored

… *yes, of course!*

First order logic tells us!

"Pseudocode " has a formal basis

Abstract State Machies

Yuri Gurevich

2000ends : Loose coupling / SOC

voices from industry:

“*THE* most relevant emerging paradigm”

“A substantial change of view
as it happens at most once each decade”

“The next fundamental software revolution after OO”

“Much more than just an other type of software!”

“The foundational layer for tomorrow's information systems”

2000ends : Loose coupling / SOC

2006: Start of B.E.S.T.

B.E.S.T offspring

… to be continued!

2010ends: Models as programs. The ideal:

- Write down your model, as abstract as *YOU* wish
- validate (contrast your model with reality and intuition)
- generate executable code automatically.

Your model may / should

- be distributed ,
- locally confine causes and effects,
- avoid global states and “operating systems”,
- do with reversible basic steps
- reflect „Composition is conjunction“ and „Implementation (refinement) is implication“

so, there is still much work to be done!

From *programs as models* to *models as programs*

Thanks for your interest

Thank you, Kees,
for so many
inspiring, enjoyable
discussions

Theory of Programming

Prof. Dr. W. Reisig