

Modelling the informatic future

I'INRIA à QUARANTE ANS

Robin Milner, Lille, December 2007

I gratefully acknowledge the Prefecture of the Île-de-France Region, for the award of a Blaise Pascal International Chair of Research.

PARTS OF THE TALK

- I The gulf between informatic **SCIENCE** and **ENGINEERING**
- II The challenge to understand **UBIQUITOUS COMPUTING**
- III The **TOWER** of informatic **MODELS**
- IV **BUILDING** the tower

PARTS OF THE TALK

- I** The gulf between informatic **SCIENCE** and **ENGINEERING**
- II** The challenge to understand UBIQUITOUS COMPUTING
- III** The tower of informatic MODELS
- IV** BUILDING the tower

Software: The gulf between engineering and science(2)

Software science has an established base in concepts:

universal machines, automata theory, formal language theory, automating logics, program semantics, specification disciplines, object orientation, type theories, process calculi, modal logics, calculi for mobile systems, intelligent agents, semi-structured data, game-theoretic models,

The impact of these on practice is low and haphazard. **Why?**

The pace of technological change and the ferociously competitive nature of the industrylead to the triumph of speed over thoughtfulness, of the maverick shortcut over discipline, and the focus on the short term.

G Robinson, quoted in RAEng/BCS report:
The Challenge of Complex IT Projects (2005)

Software: The gulf between engineering and science(4)

Example The **year-2000 fiasco**:

There was no disaster, but enormous sums wasted expecting one!
Appropriate theory to prevent it had been around for *two decades*.

Symptoms (or causes?) of the gulf:

- The **science** is neither complete nor unified
- Software houses design what **the market** demands, not what has been subjected to available theoretical analysis
- The software industry focusses on **managing software production**, not on understanding software itself.

A disconnect in the perception of informatics?

Software engineering often found faulty; does not match procedures in other engineering disciplines.

A recent UK report, **The Challenge of Complex IT projects***,

- exposes the fault in great depth
- and makes sound recommendations on **the software process**,

but *hardly mentions* models, or the science of software!

Implication: Software science and engineering are **disconnected**

A Grand Challenge: Establish modelling as the basis of informatics.

* jointly by the Royal Academy of Engineering and the British Computer Society

PARTS OF THE TALK

- I The gulf between informatic SCIENCE and ENGINEERING
- II The challenge to understand UBIQUITOUS COMPUTING**
- III The tower of informatic MODELS
- IV BUILDING the tower

Two visions of Ubiquitous Computing

Populations of computing entities will be a significant part of our environment, performing tasks that support us, and we shall be largely unaware of them. (after Mark Weiser, 1994)

*In the next five to ten years the computer will be erased from our consciousness. We will simply not talk about it any longer, we will not read about it, **apart from experts of course.***

(my emphasis)

Joseph Weizenbaum (2001)

..... and a third vision:

Ubiquitous computing will **empower us**, if we **understand it**.

Qualities of a ubiquitous computing system (UCS)

What is new about a UCS?

- It will continually make **decisions** hitherto made by us
- It will be **vast** – orders of magnitude larger than today's systems
- It must continually **adapt**, on-line, to new requirements
- Individual UCSs will **interact** with one another

Can the industry cope, using traditional software engineering?

Concepts for Ubicomp

Each ubicomp **domain**, hence each **model**, will involve several concepts. Here are a few:

provenance obligations
locality intentions specification self-management
beliefs continuous space data-protection
encapsulation mobility authorisation simulation
compilation policy continuous time role
delegation reflectivity failure
trust stochastics verification
security authenticity connectivity

Managing the conceptual overload



- Define a **model M** based on a small manageable subset, e.g. locality, connectivity, mobility, stochastics. **Implement M.**
- Build a **model tower M1, M2, ...** above **M** by adding more concepts.

But what is the meaning of ‘**model**’, ‘**implement**’, ‘**tower**’ ?

PARTS OF THE TALK

- I The gulf between informatic SCIENCE and ENGINEERING
- II The challenge to understand UBIQUITOUS COMPUTING
- III The tower of informatic **MODELS**
- IV BUILDING the tower

Bridging the gulf: a challenge

- Think in terms of a **Tower of inter-related models**.
- Informatic scientists have always built models—e.g. *a programming language, Petri nets, a security discipline, intelligent agents, ...*
- Software engineers begin to structure their work in terms of models (*Model-Driven Engineering, MDE*).
- But **What is a model?**

Tower of models

A **model** consists of some *entities*, and their *meaning*.

EXAMPLE: flowcharts, and how to execute them.

A **tower** of models is built by **explanation** and **combination** :

Model **A** **explains** model **B** if

A *abstracts from* or *specifies* **B**, or if

B *implements* or *refines* **A**.

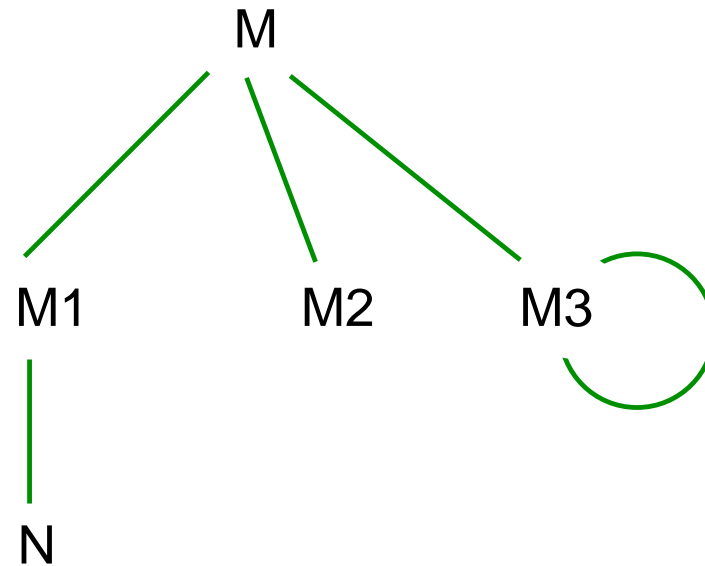
EXAMPLE: formulae in a specification logic specify programs.

Model **A** **combines** models **A1** and **A2** if

its entities and meanings join those of **A1** and **A2**.

EXAMPLE: combining distributed programs with networks.

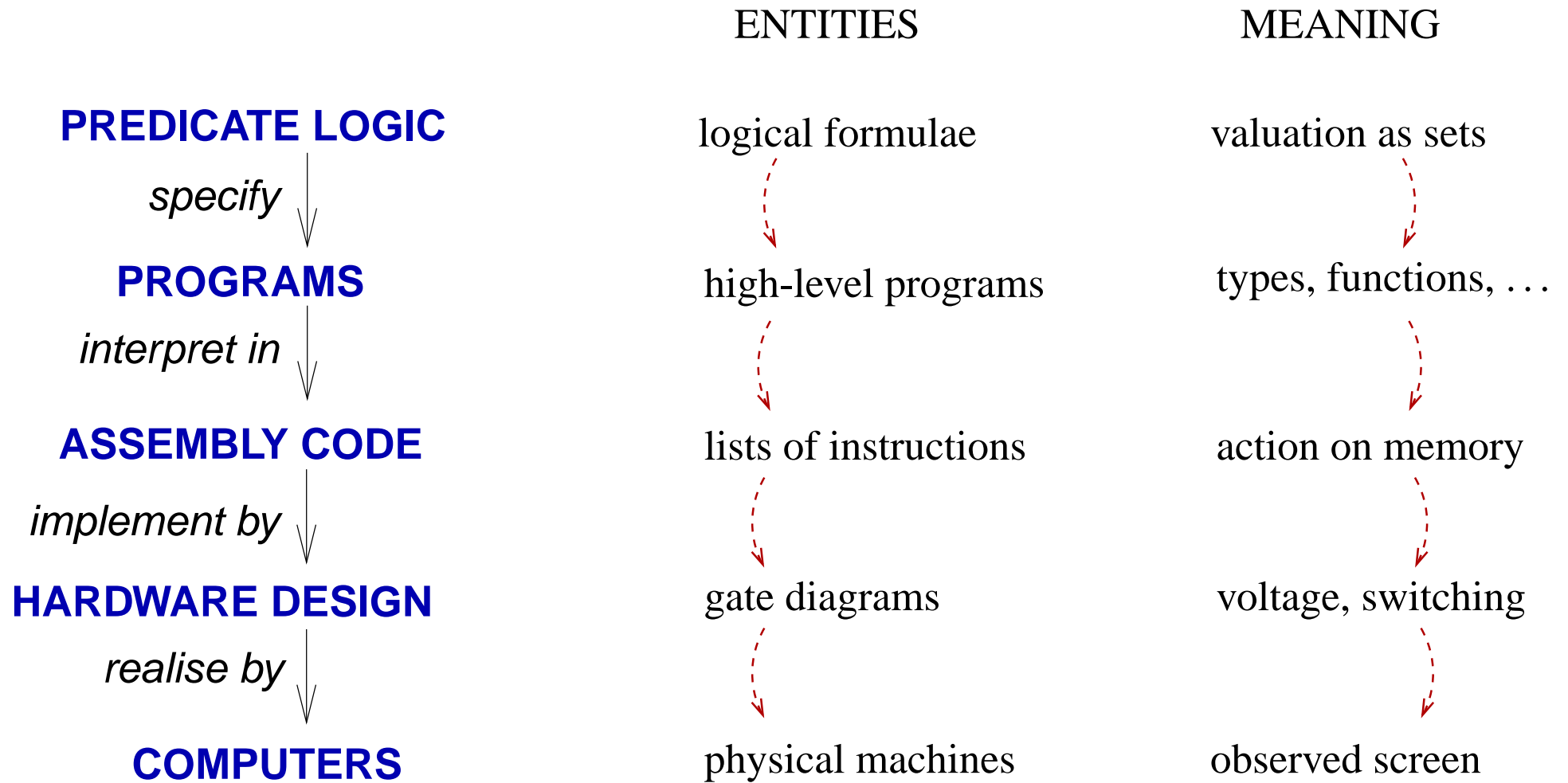
Example: a small tower of models



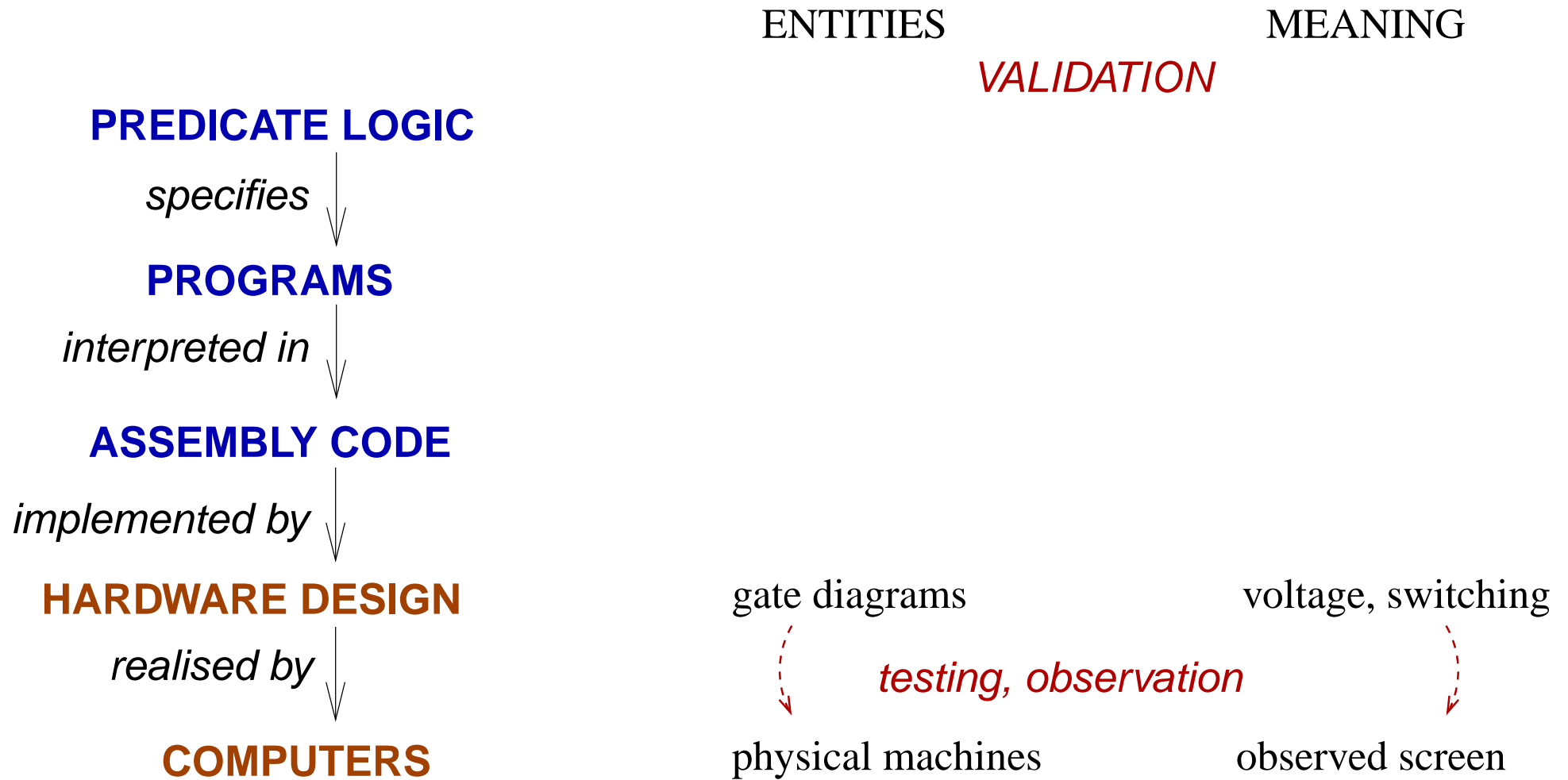
Model **M** is realised by a combination of **M1**, **M2** and **M3**.

M1 is realised by **N**. **M3** realises itself.

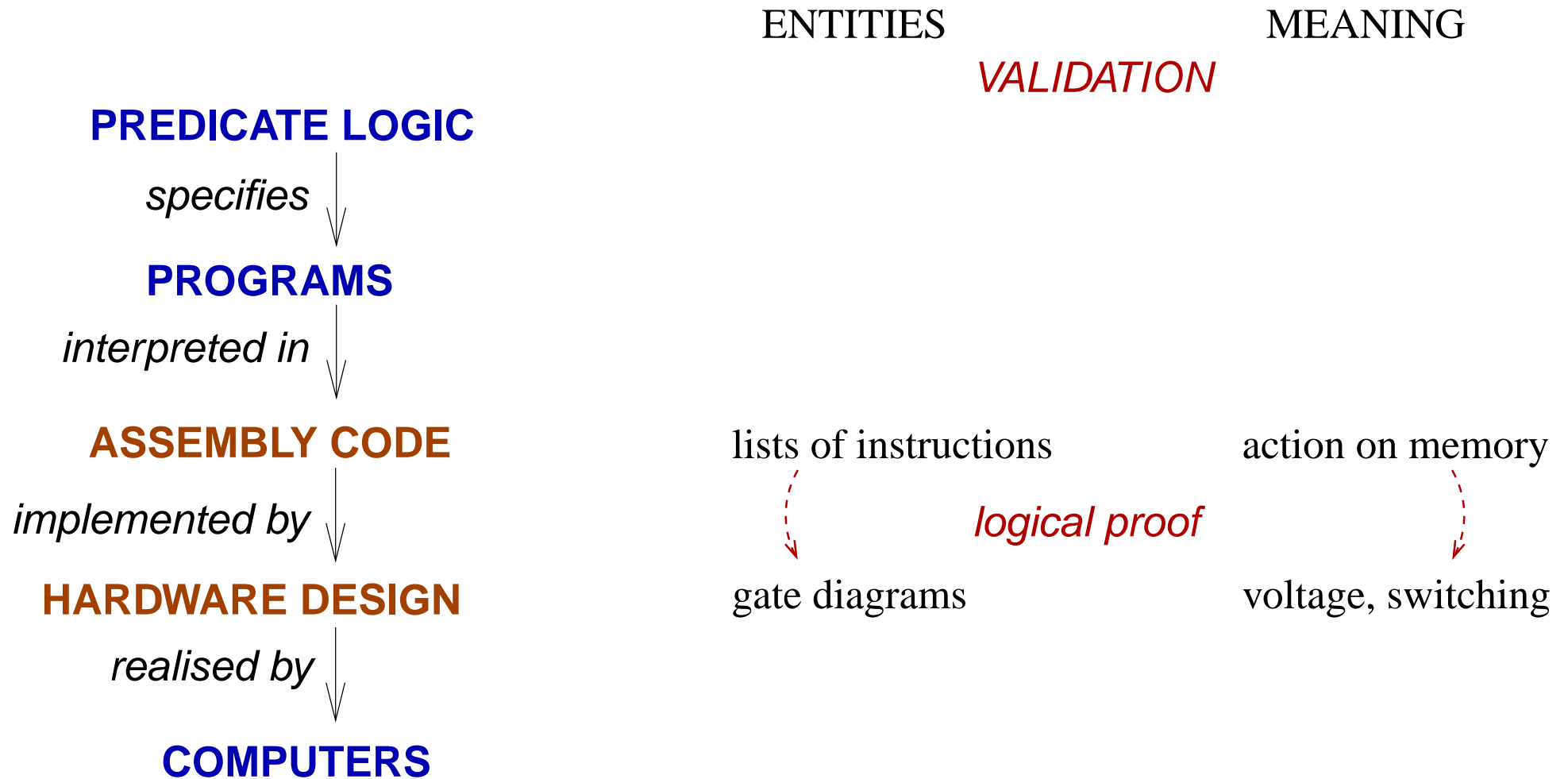
A tall tower, for programming



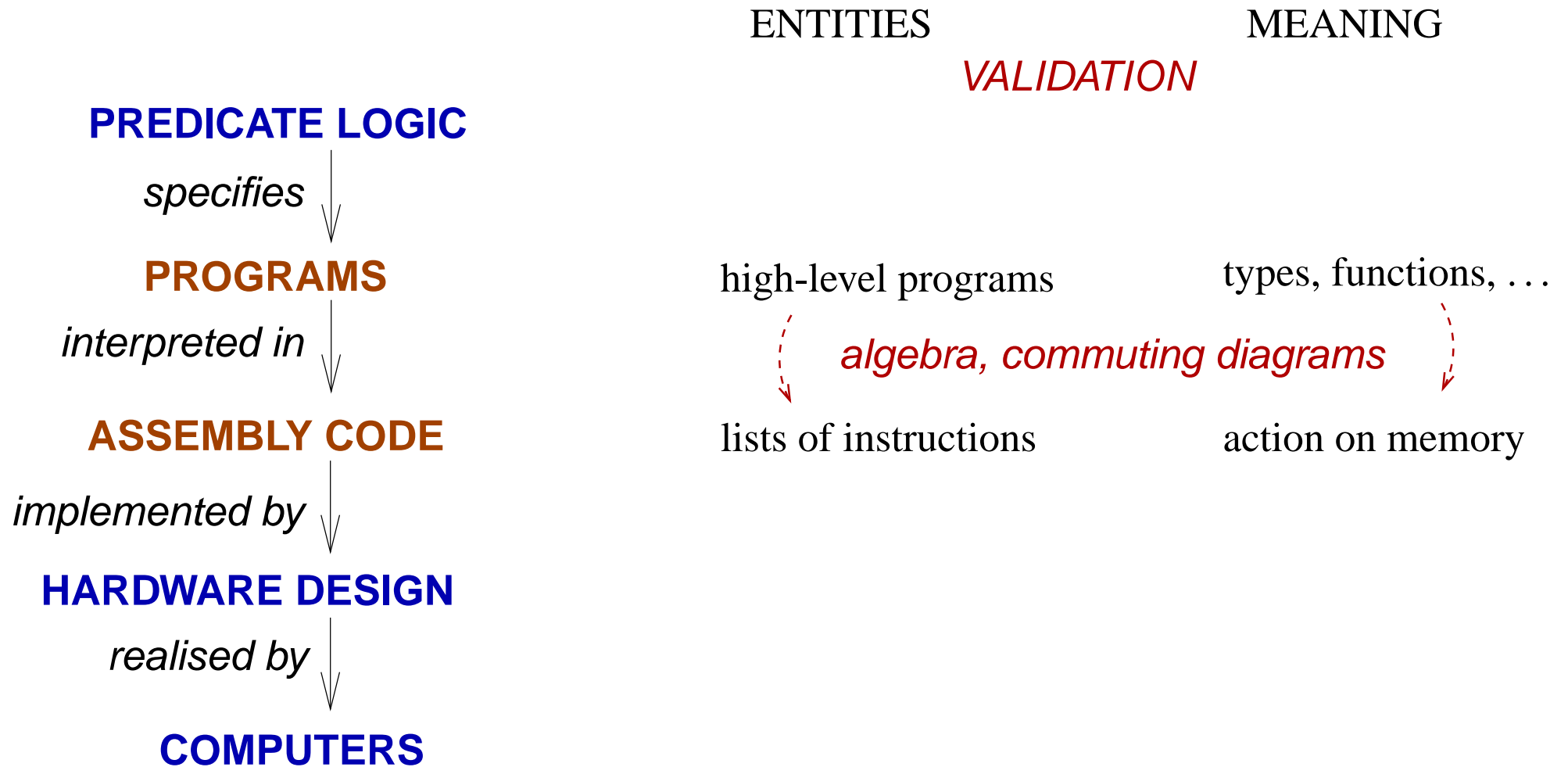
A tall tower, for programming



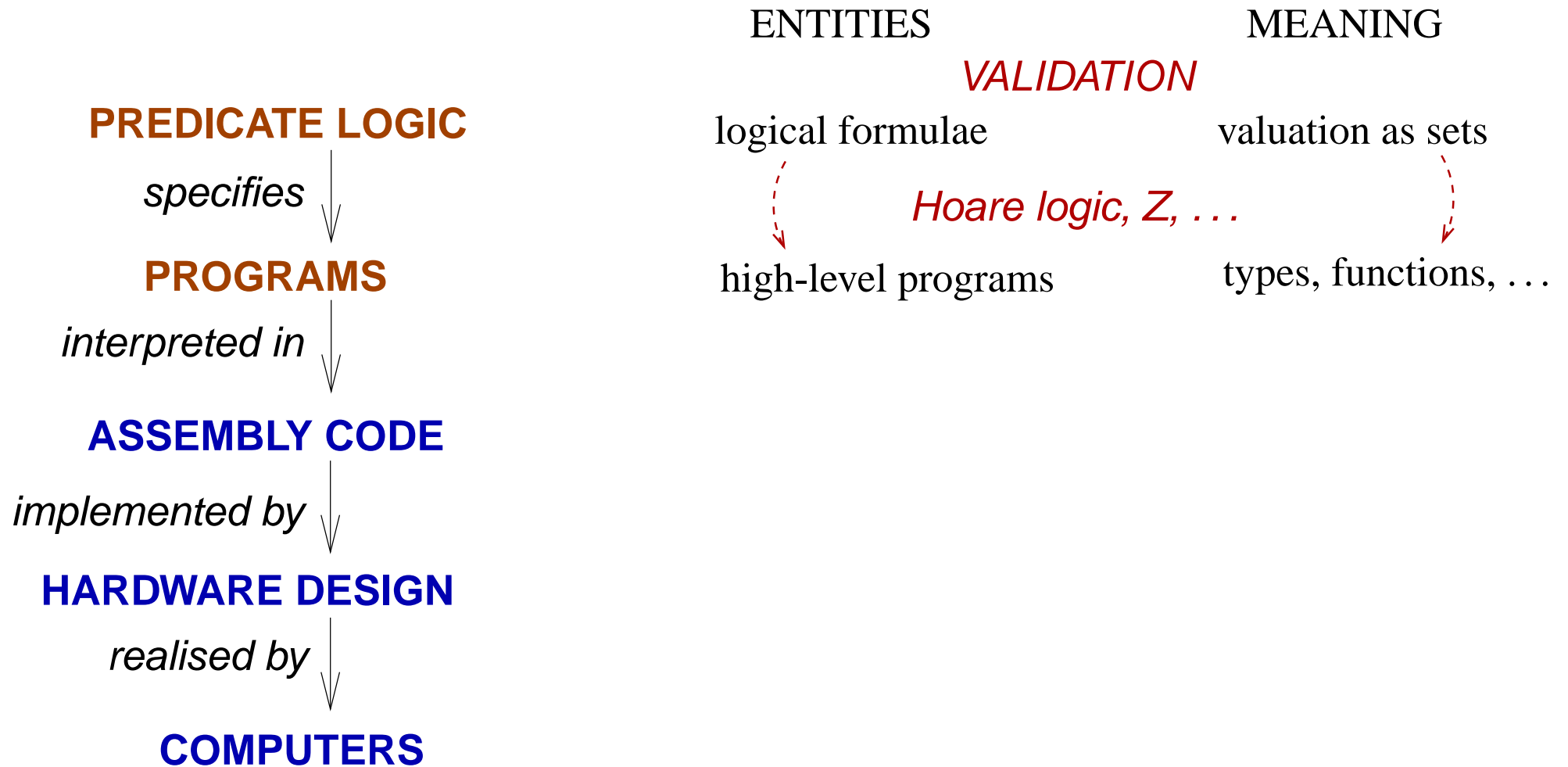
A tall tower, for programming



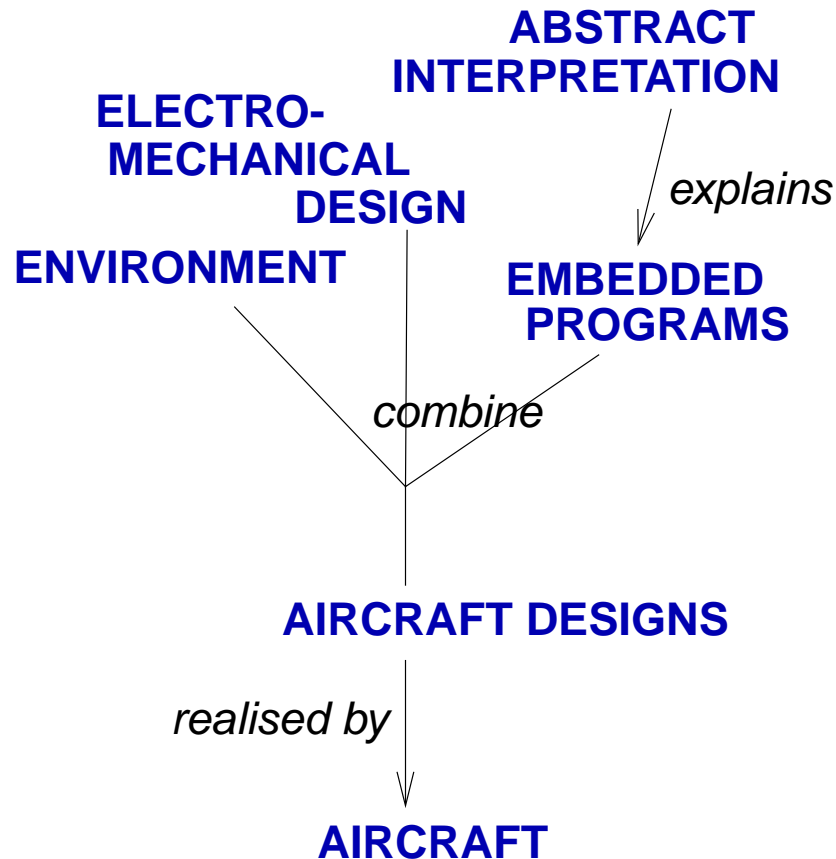
A tall tower, for programming



A tall tower, for programming



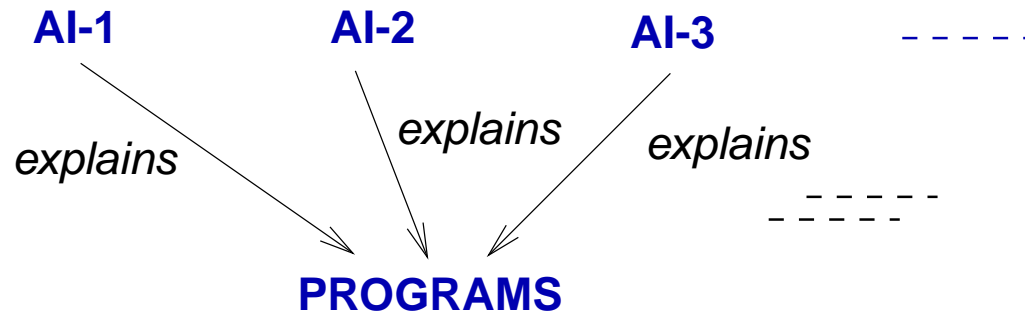
A wide tower for flight control



The combined model can give scientific answers to questions like *at what levels of turbulence will the aircraft maintain control?*

After the failure of the launch of **Ariane 5**, INRIA was committed by Scientific Director Gilles Kahn to analyse the embedded software for the **Airbus**.

Specialised Abstract Interpretation



- Each programming model demands many behavioural analyses: *type-checking, response time, numerical overflow,*
- For each analysis, adopt a **different abstract interpretation**.
- **Huge** efficiency gain, over a single all-purpose analyser.

PARTS OF THE TALK

- I The gulf between informatic SCIENCE and ENGINEERING
- II The challenge to understand UBIQUITOUS COMPUTING
- III The tower of informatic MODELS
- IV **BUILDING** the tower

Working with models + explanations

- Use them to present **existing knowledge** and **ongoing work**. Many examples can be seen as models linked by explanations

Explanations for programming

programming language	<i>implemented by</i>	assembly language
assembly language	<i>realised by</i>	binary code
binary code	<i>realised by</i>	machine
denotational semantics	<i>explains</i>	programming language
Z specification	<i>realised by</i>	programming language
abstract interpretation	<i>abstracts from</i>	programming language

Explanations for process and interaction

security disciplines

realised by

cryptology

higher-order logic

explains

security disciplines

cryptology

implemented by

programming language

intracommunication

realised by

intercommunicating processes

swim lanes

explain

intracommunication

CSP

refined by

CSP

Explanations for ubiquitous computing

intelligent agents	<i>realised by</i>	(O-O) programming
game theory	<i>explains</i>	intelligent agents
trust logic	<i>realised by</i>	authorisation protocols
P-to-P messaging	<i>realised by</i>	routing strategy
self-managed cells	<i>specify</i>	composable ubicomp systems
? modal/temporal logic	<i>explains</i>	self-managed cells

Working with models + explanations

- Use them to present **existing knowledge** and **ongoing work**. Many examples can be seen as models linked by explanations
- Where possible, identify a **field** of knowledge as a subtower. Ask how fields themselves are linked by explanation.
- Identify different **kinds** of explanation. We have many examples: what do they have in common?
- Explanation can often be automated, e.g. *model-checking, compiler verification, abstract interpretation*. **Software engineering** must automate any explanation made precise by **software science**.

What's the point of a Grand Challenge in informatics?

To make applications that startle the world?

(e.g. beating a grandmaster at chess)

OR

To develop organising principles for a science?

The first without the second may (or may not) spin off science

The two together will embed computing in our scientific culture

....oooo000000000000oooo....