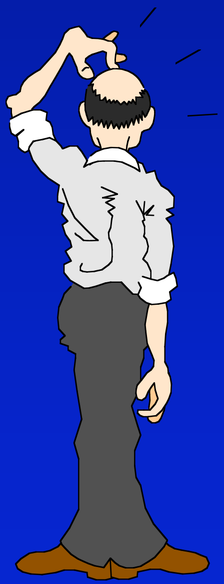
Abstraction and Modelling
a complementary partnership

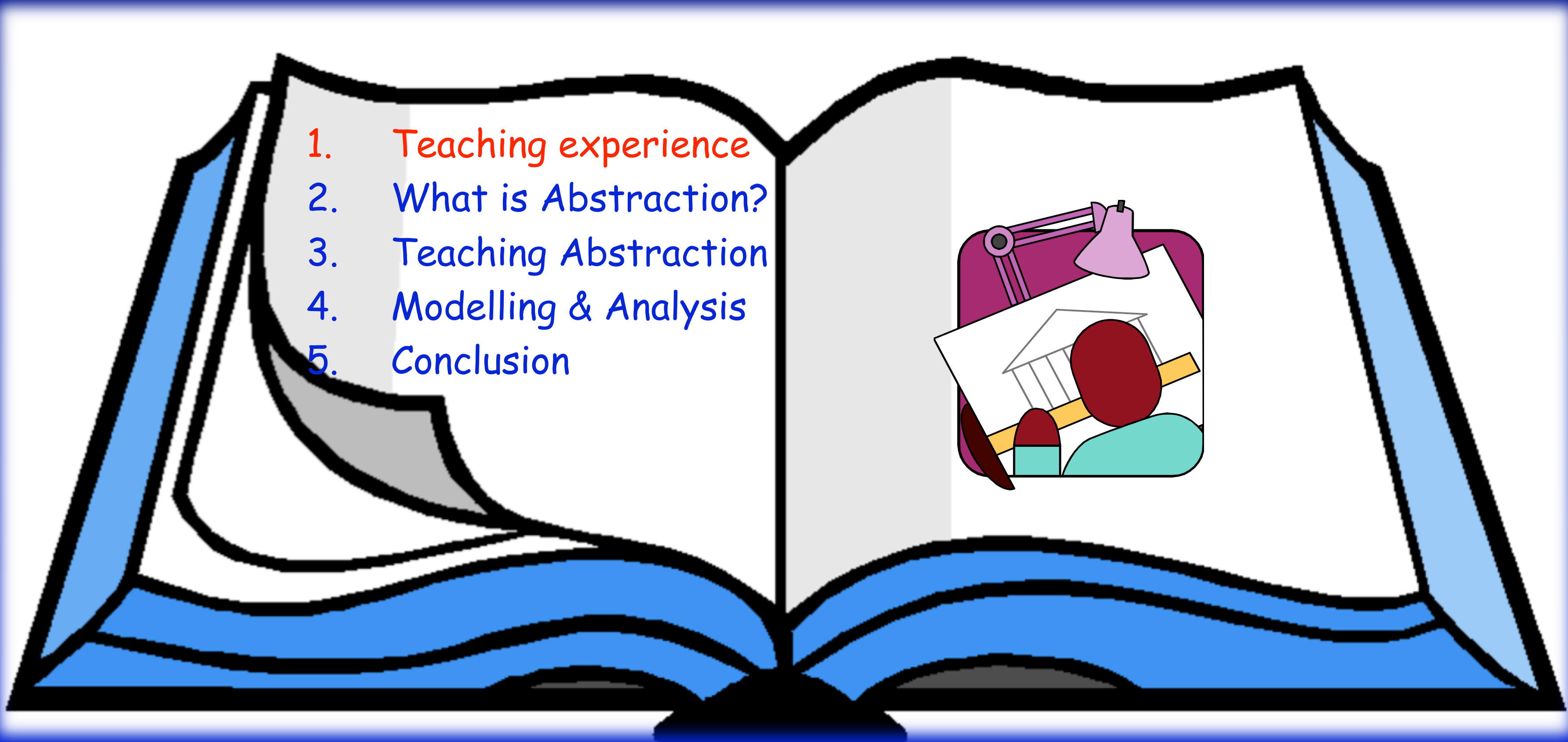


Jeff Kramer

Imperial College London

"Is Abstraction the key to Computing?" CACM April 2007

Chapter 1. My teaching experience



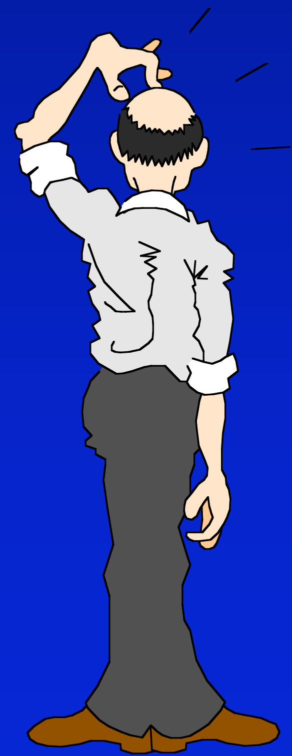
Teaching experience

Courses:

- software engineering,
- distributed systems,
- distributed algorithms,
- programming,
- concurrency,
- ……

Skills:

- problem solving,
- conceptualization,
- modelling,
- analysis,
- ……



Experience: the **better** ones….

Some students are able to produce **elegant** designs and solutions.

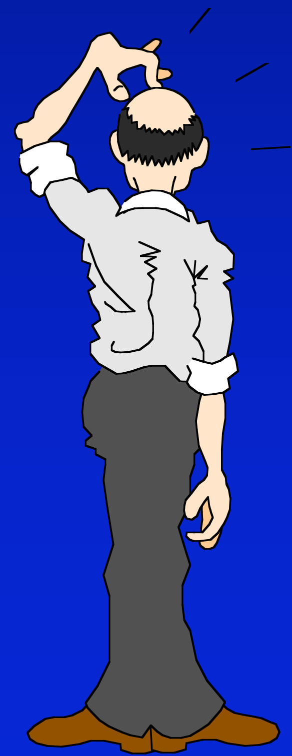
Generally the same students are also able to comprehend the complexities of distributed algorithms, the applicability of the various modelling notations, and so on.

Experience: the **others** ….

A number of others are **not so able**.

They tend to find distributed algorithms very difficult, do not appreciate the utility of modelling, find it difficult to know what is important in a problem, produce convoluted solutions which replicate the problem complexities, ……

Why ?

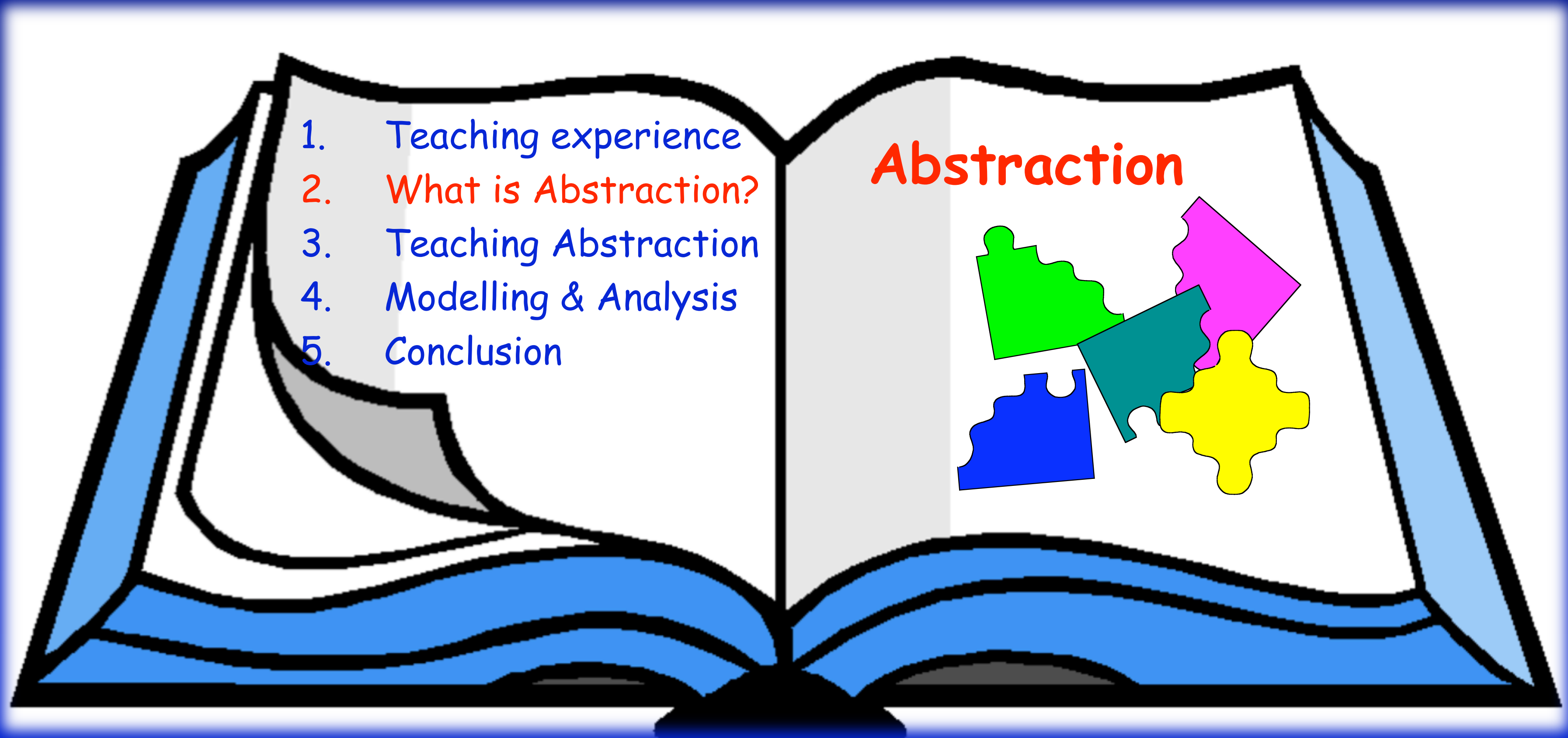


I believe …..

… that the heart of the problem lies in a difficulty in dealing with

Abstraction

Chapter 2. What is it? Why is it so important?



Definitions

- the act of withdrawing or removing something
- the act or process of leaving out of consideration one or more properties of a complex object so as to attend to others

=> Remove detail (simplify) and focus (selection)

- a general concept formed by extracting common features from specific examples
- the process of formulating general concepts by abstracting common properties of instances

=> generalisation (core or essence)

Abstraction in other disciplines

Art

Music

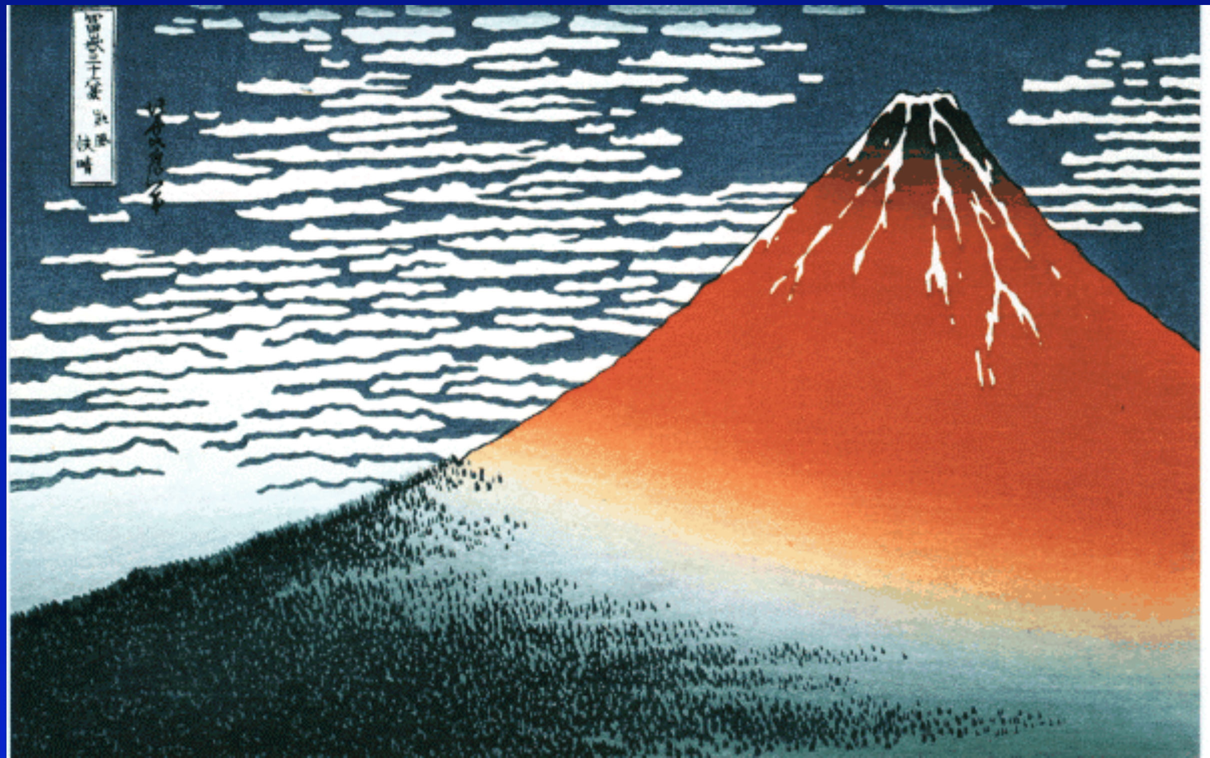
Maps

Matisse – guess what ….

representation
of the essence
of the subject
&
removal of
detail

Katsushika Hokusai – guess what ….

South Wind,
Clear Skies
(Red Fuji)



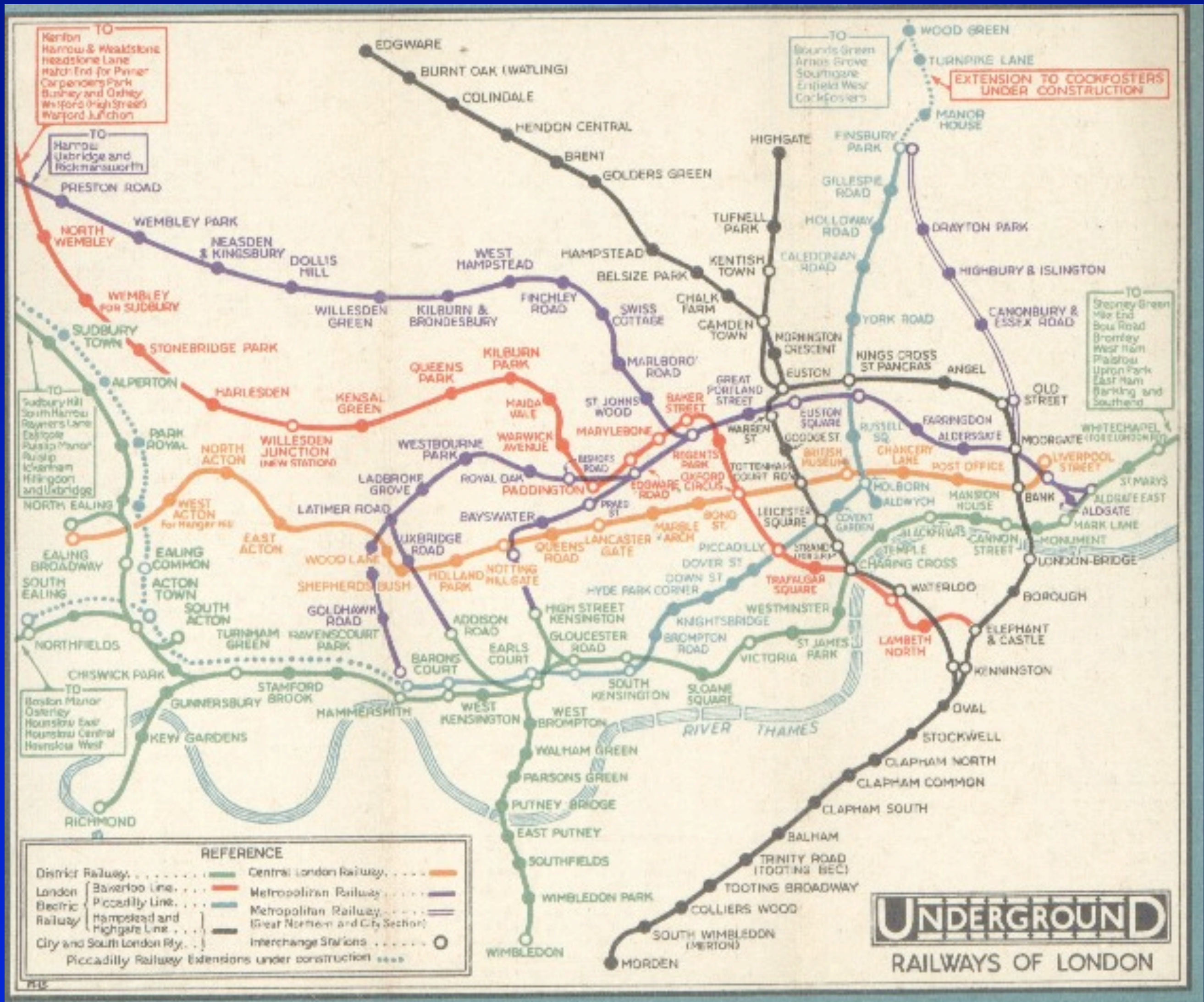
…" balance of colour and composition rendering an abstract form of the mountain to capture its essence" - art commentator

Jazz

jazz musician on the process of abstraction –

"It is easy to make something simple sound complex, however its more difficult to make something complex sound simple".

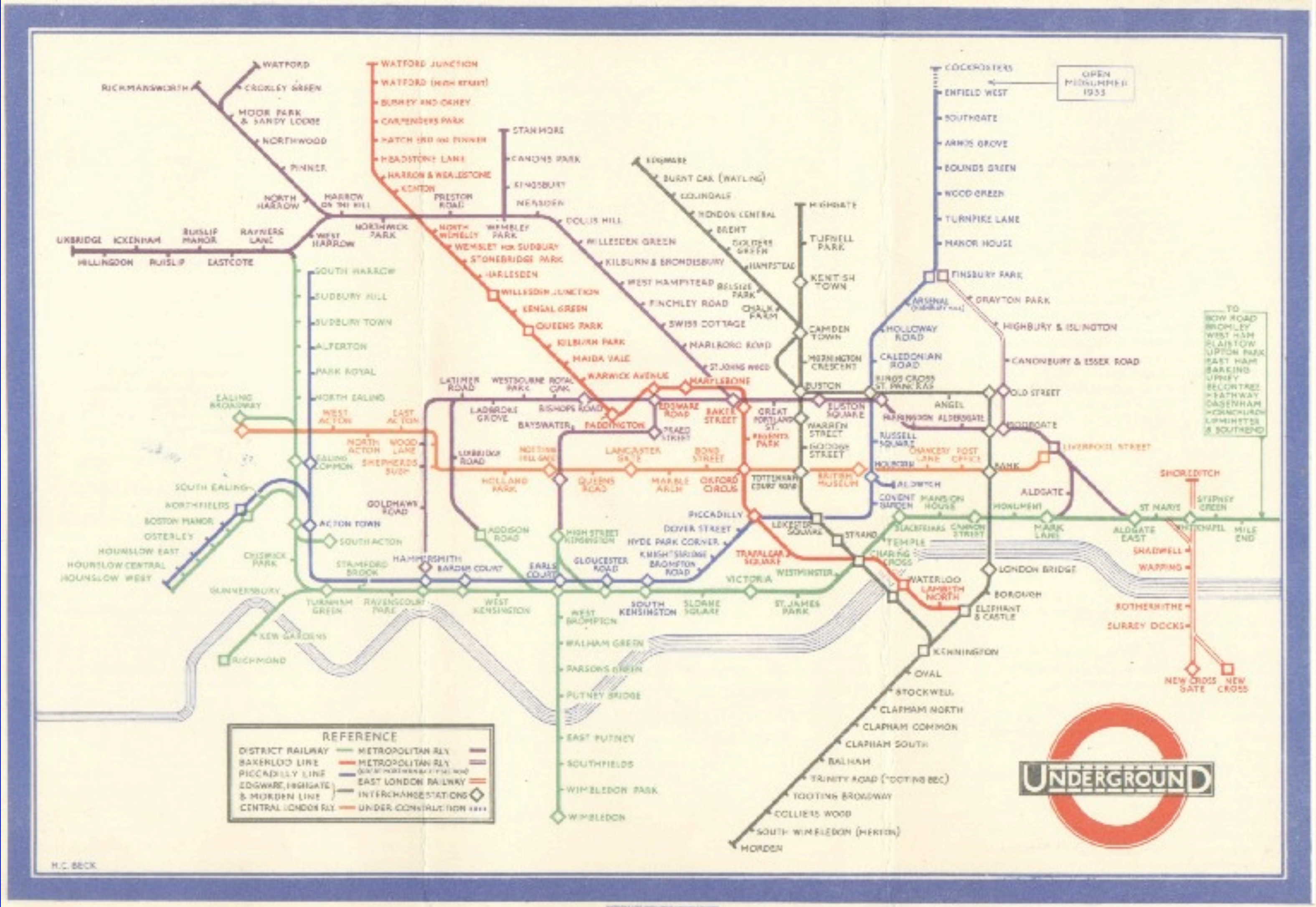
1930 – London Underground map



“Fit for purpose?”

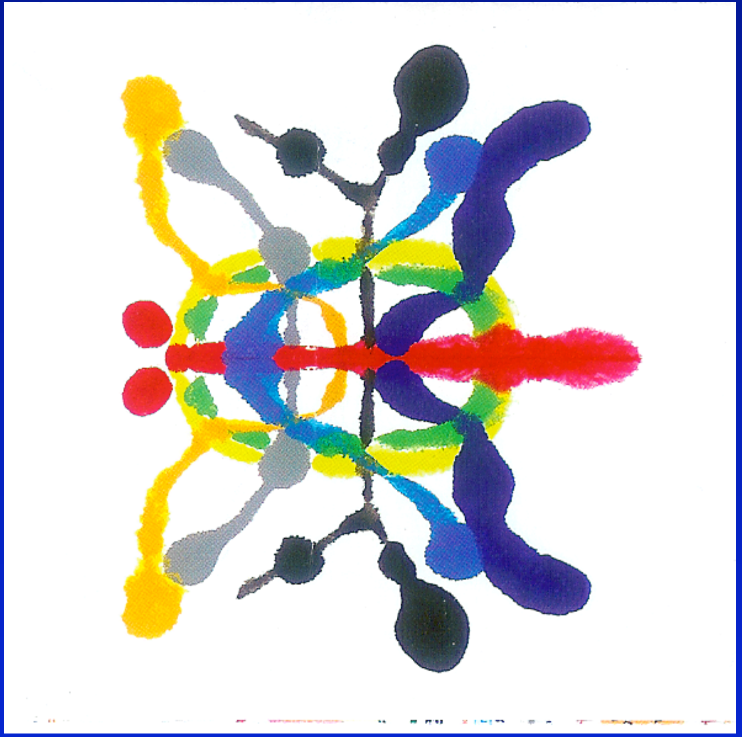
Relationship between stations and interchanges, not actual distances

1932 – Harry Beck (1st schematic image map)

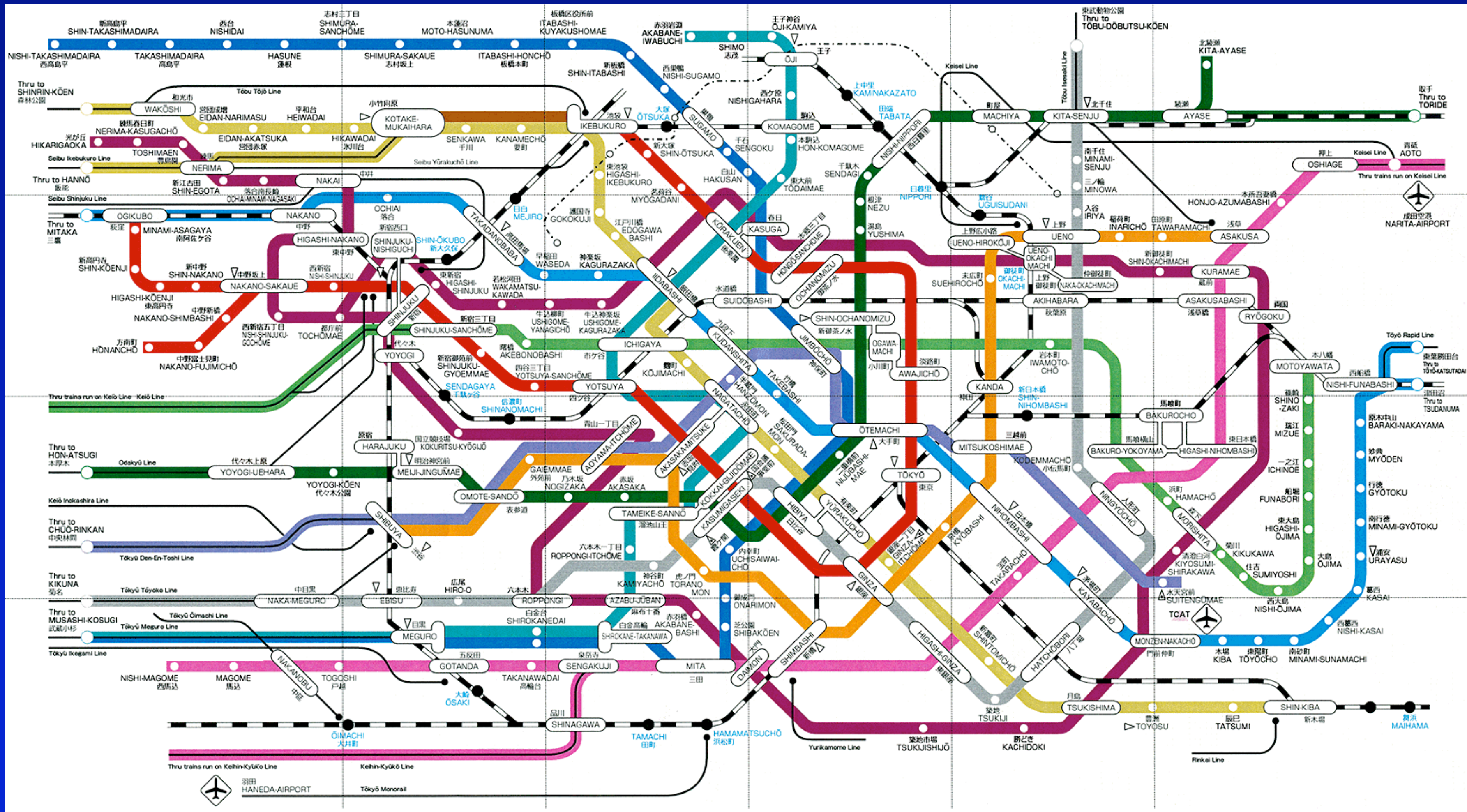


2001 – Fit for purpose ("mind the gap…")

Fit for purpose ?!



Fit for purpose – internationally!



Why is abstraction important in **Software Engineering**?

Software is abstract!

"Once you realize that computing is all about constructing, manipulating, and reasoning about abstractions, it becomes clear that an important prerequisite for writing (good) computer programs is the ability to handle abstractions in a precise manner."

Keith Devlin CACM Sept.2003

Why is abstraction important in **Software Engineering**?

Software is abstract!

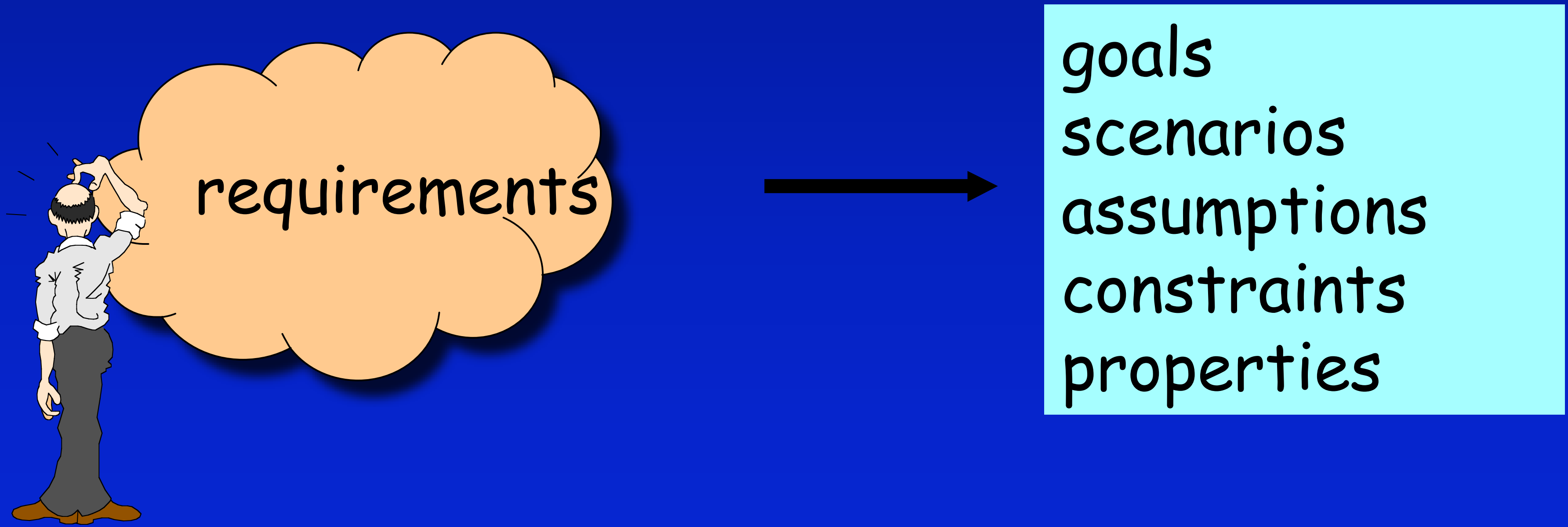
Requirements

Design

Programming

Why is it important? requirements engineering

Requirements - elicit the critical aspects of the environment and required system while neglecting the irrelevant.



"The act/process of leaving out of consideration one or more properties of a complex object so as to attend to others"

Why is it important? design

Design - articulate the software architecture and component functionalities which satisfy functional and non-functional requirements while avoiding unnecessary implementation constraints.

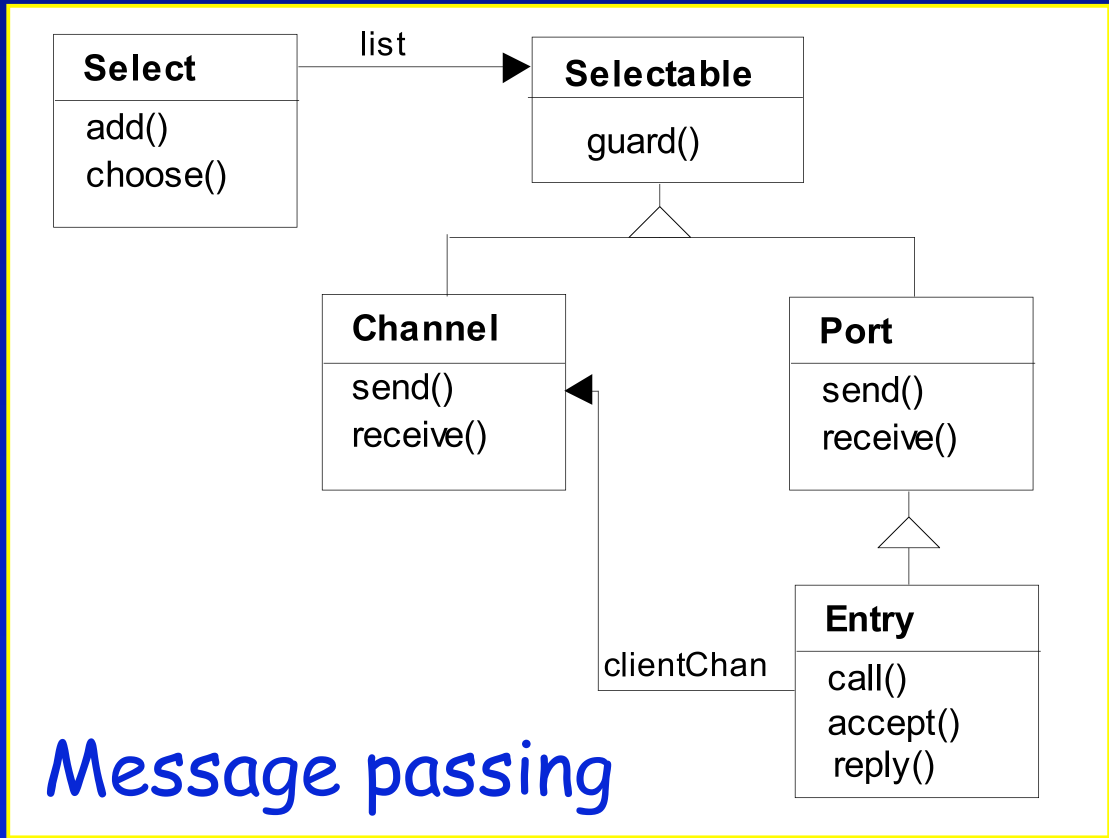
eg. Compiler design (Ghezzi):

- *abstract syntax* to focus on essential features of language constructs;
- design to generate intermediate code for an *abstract machine*

"The act/process of leaving out of consideration one or more properties of a complex object so as to attend to others"

Why is it important? programming

Programming - use data abstraction and classes so as to generalize solutions.



"the process of formulating general concepts by abstracting common properties of instances "

Why is it important? advanced topics

Abstract interpretation for program analysis - map concrete domain to an abstract domain which captures the semantics for the purpose at hand.

eg. Rule of signs for multiplication *

0*+ = 0*- = +*0 = -*0 = 0

+*+ = -*- = +

+*- = -*+ = -

Hankin

"the process of formulating general concepts by abstracting common properties of instances "

Abstraction

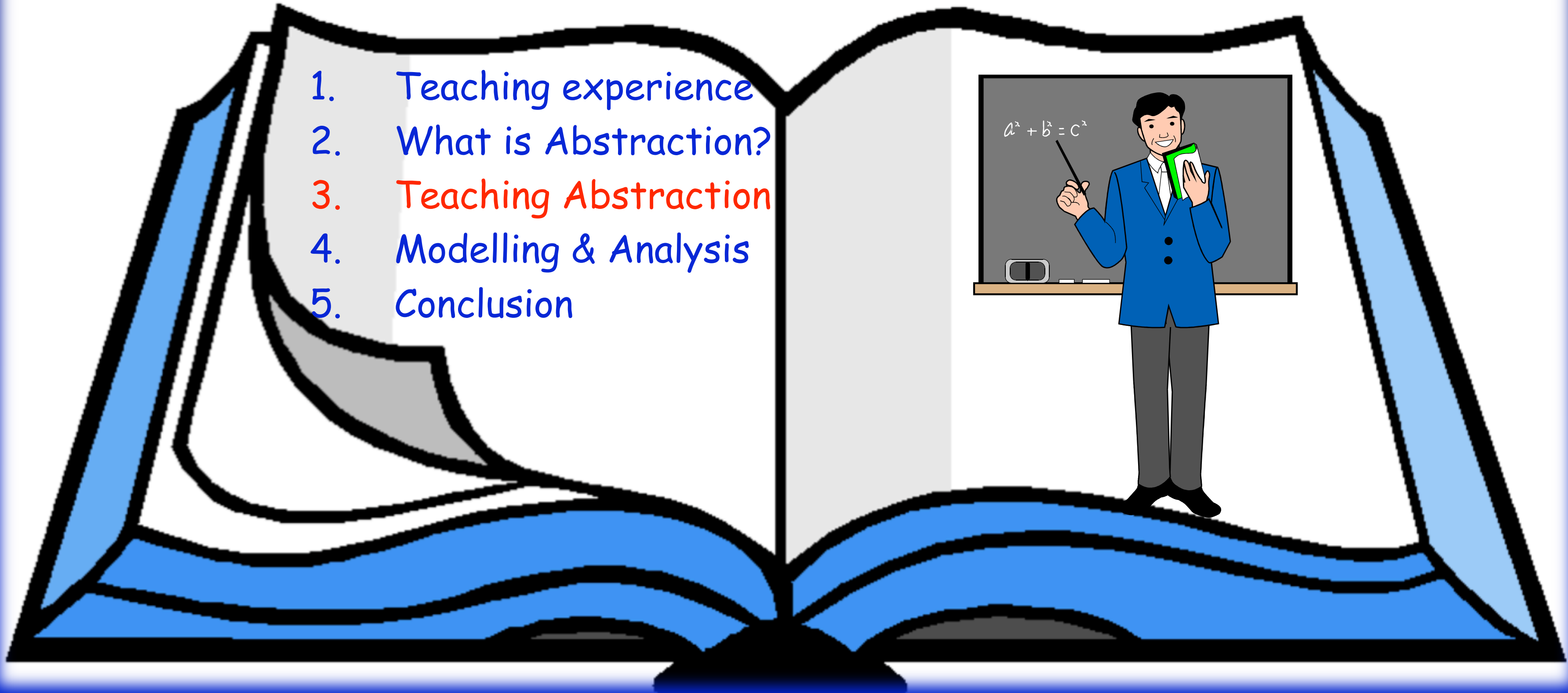
Abstraction is fundamental to Engineering in general, and to Software Engineering in particular !

Do our powers of abstraction depend on our **genes** ?

Can we improve our abilities ? ...and if so, how ?

Is it possible to teach abstraction ?

Chapter 3. Teaching abstraction?



Cognitive Development

Changes in thinking by which mental processes become more complex and sophisticated.

Jean Piaget's fours stages of cognitive development:

1st & 2nd: sensorimotor and preoperational (0-7yrs)

3rd stage: concrete operational thought (7-12yrs)

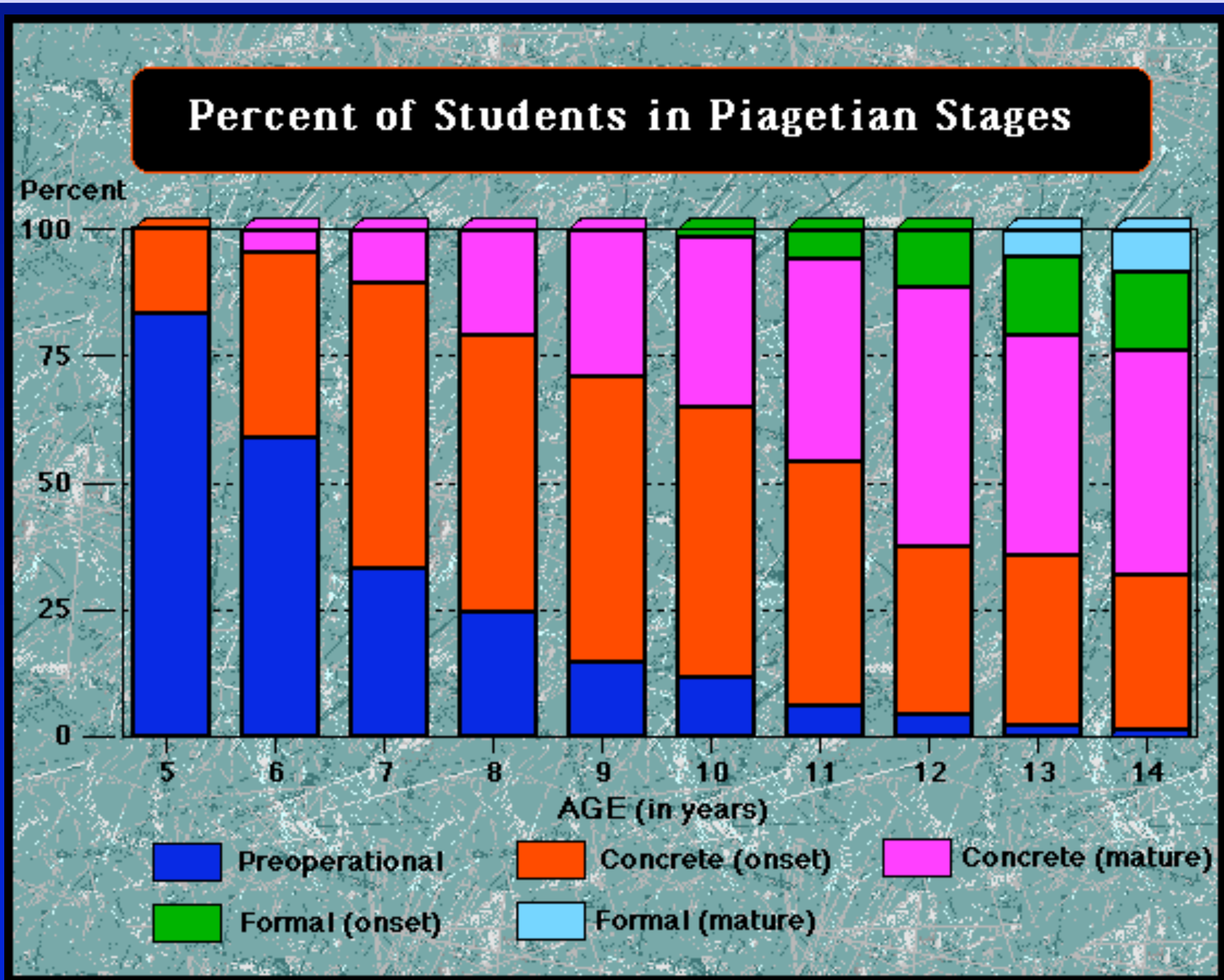
no abstract thought

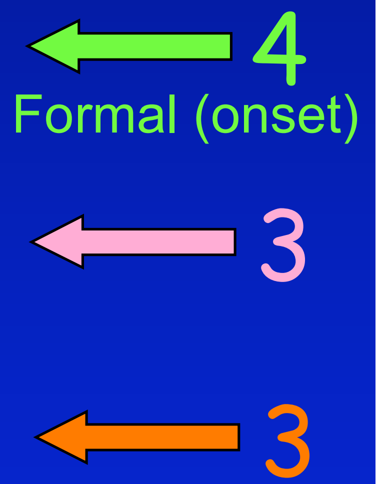
4th stage: formal operational period (12–adult)

think abstractly (logical use of symbols related to abstract concepts), systematically and hypothetically

Huitt & Hummel

Cognitive Development – formal operational thought





Kuhn et al

Cognitive Development

Changes in thinking by which mental processes become more complex and sophisticated.

Jean Piaget's fours stages of cognitive development:

1st & 2nd: sensorimotor and preoperational (0-7yrs)

3rd stage: concrete operational thought (7-12yrs)

good news

Some ability for abstraction with training

4th stage: formal operational period (12-adult)

bad news

Not reached by all individuals. Only 30% to 35% of adolescents exhibit ability for abstact thought, some adults never do!

Courses on **Abstraction**?

1st Year (all required):

Declarative Programming I

Databases 1

Declarative Programming II

Discrete Mathematics 1

Discrete Mathematics 2

Hardware

Programming I

Logic

Reasoning about Programs

Programming II

Computer Systems

Mathematical Methods and Graphics

2nd Year (most required):

Algorithms, Complexity and Computability

Architecture II

Compilers

Artificial Intelligence I (optional)

Operating Systems II

Computational Techniques (optional)

Software Engineering - Design I

Concurrent Programming (optional)

Statistics

Networks and Communications

Software Engineering - Design II

Imperial College MEng in Software Engineering

Courses on Abstraction?

3rd Year (most optional):

Advanced Databases
Advanced Computer Architecture
Advances in Artificial Intelligence
Computational Finance
Computational Logic
Custom Computing
Distributed Systems
Introduction to Bioinformatics
Knowledge Management Techniques
Decision Analysis
Operations Research
Graphics
Quantum Computing
Management - Organisation and Finance (required)
Simulation and Modelling
Multimedia Systems
Software Engineering - Methods (required)
Performance Analysis
The Practice of Logic Programming
Robotics
Type Systems for Programming Languages

4th Year (most optional):

Advances in Artificial Intelligence
Advanced Graphics and Visualization
Advanced Issues in Object Oriented
Programming Automated Reasoning
Advanced Operations Research
Complexity
Computer Vision
Computing for Optimal Decisions
Intelligent Data and Probabilistic Inference
Domain Theory and Exact Computation
Modal and Temporal Logic
Grid Computing
Models of Concurrent Computation
Knowledge Representation
Natural Language Processing
Management - Economics and Law
Network Security
Multi-agent Systems
Program Analysis
Parallel Algorithms
Software Engineering - Environments

Courses on **Abstraction**?

Which courses rely on or utilise the powers of abstraction to

- explain
- model
- specify
- reason
- solve .... ?

List of courses which do **NOT** make use of **Abstraction**?

So ....?

Abstract is essential but is taught indirectly.

So ....?

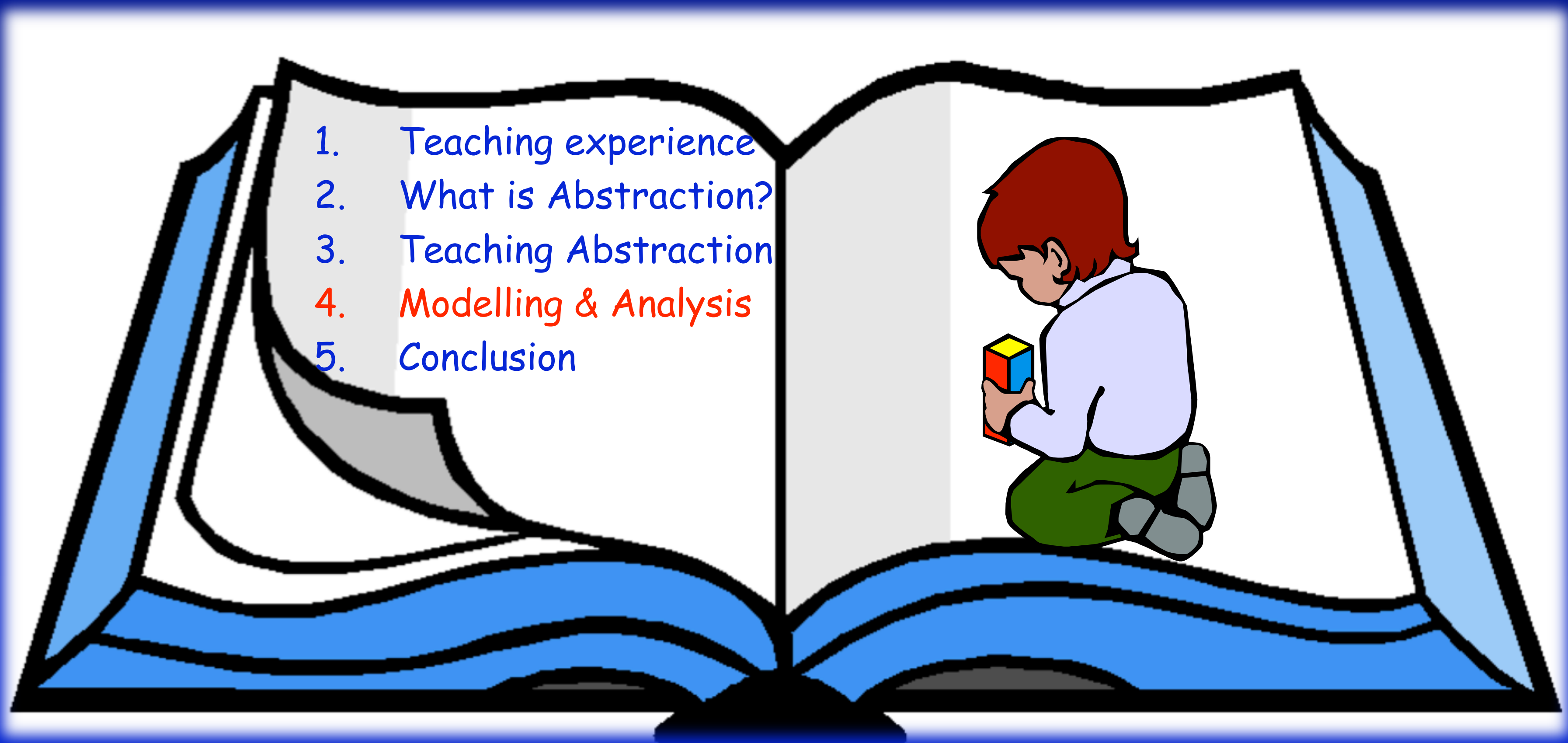
How should we ensure that students can understand and make use of **abstraction** ?

1. Teach enough Mathematics
2. Teach (formal) **modelling and analysis**

Caveat: Must be tool supported

Must feel the benefit

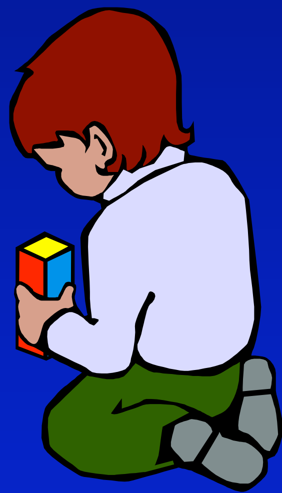
Chapter 4. Modelling and analysis



Models and Modelling?

- A model is a description from which detail has been removed in a systematic manner and for a particular purpose.

- A simplification of reality intended to promote understanding, reasoning and analysis.



- Models are the most important engineering tool; they allow us to understand and analyse large and complex problems.

Ockam's Razor

- William of Ockam (1285) formulated the famous "Rule of the Razor":

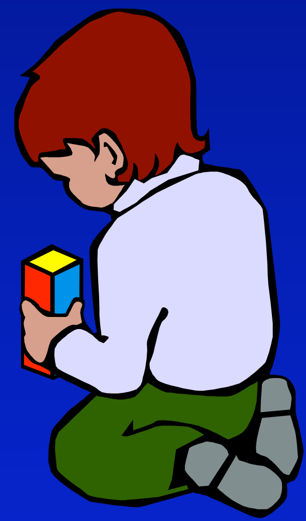
  Entia non sunt multiplicanda sine necessitate.

  Entities should not be multiplied without necessity.

- In other words a model should be as simple as possible, but no simpler - it should discard elements of no interest.

- **"Fit for purpose"**.

The challenge is to make modelling and analysis accessible and useful to software engineers.



I teach Concurrency – models and programs

Model-based approach

- Models
  - finite state models (FSP and LTS),
  - model checking for analysis (LTSA).
- Practice
  - Map into Java for concurrent programs.

component VOTER - behaviour

Component:

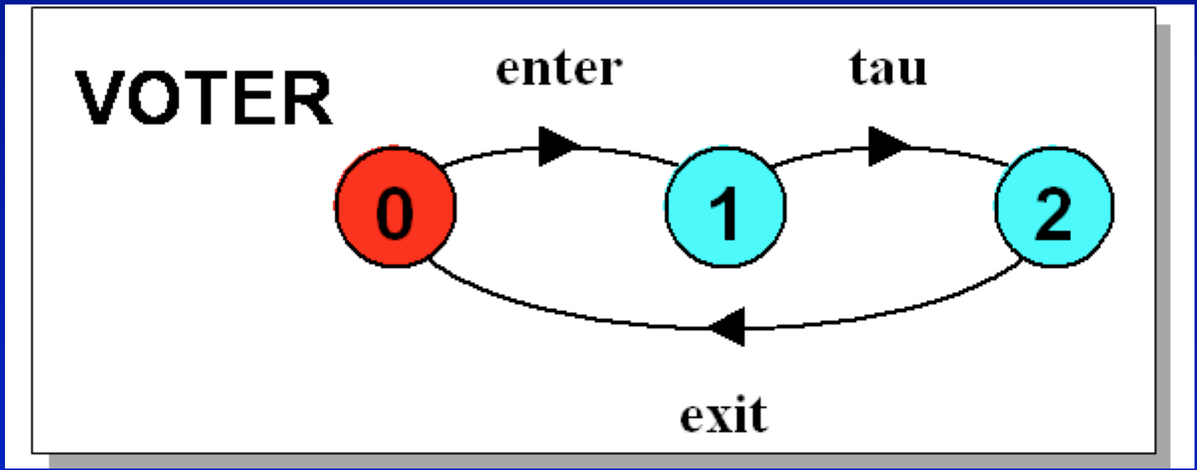
Process specification in FSP:

```
VOTER = (enter -> vote -> exit -> VOTER
        ) @{enter,exit}.
```

Actions `{enter,exit}` *are exposed,* `vote` *is hidden.*

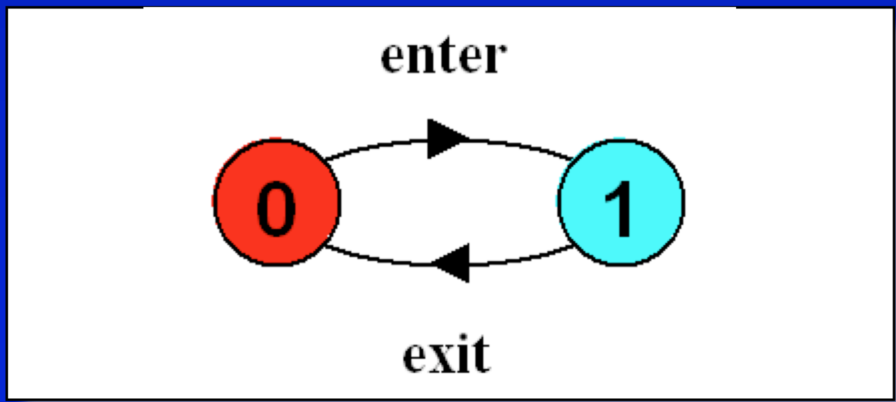
component USER - behaviour

Labelled transition system LTS:



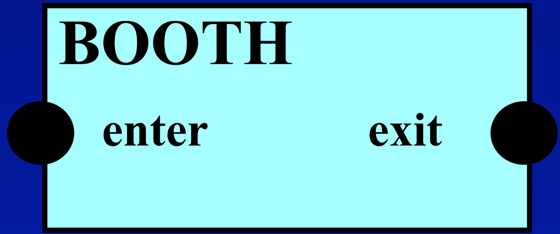
LTS Animation can be used to step through the actions to test specific scenarios.

VOTER can be minimised with respect to Milner's observational equivalence.



component BOOTH - behaviour

Component:



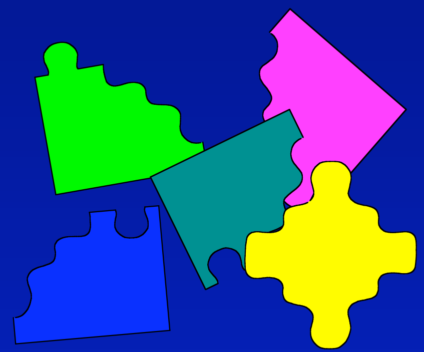
Voting booths used in Paris 2007 election.

Process specification in FSP:

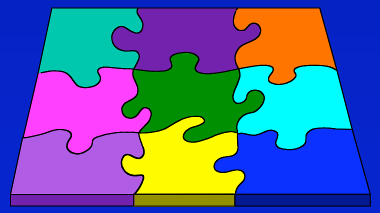
```
const Max = 3
range Int = 0..Max

BOOTH(N=Max) = BOOTH[N],
BOOTH[v:Int] =(when(v>0) enter -> BOOTH[v-1]
              |when(v<N) exit  -> BOOTH[v+1]
              ).
```

Modelling concurrent systems

Primitive components

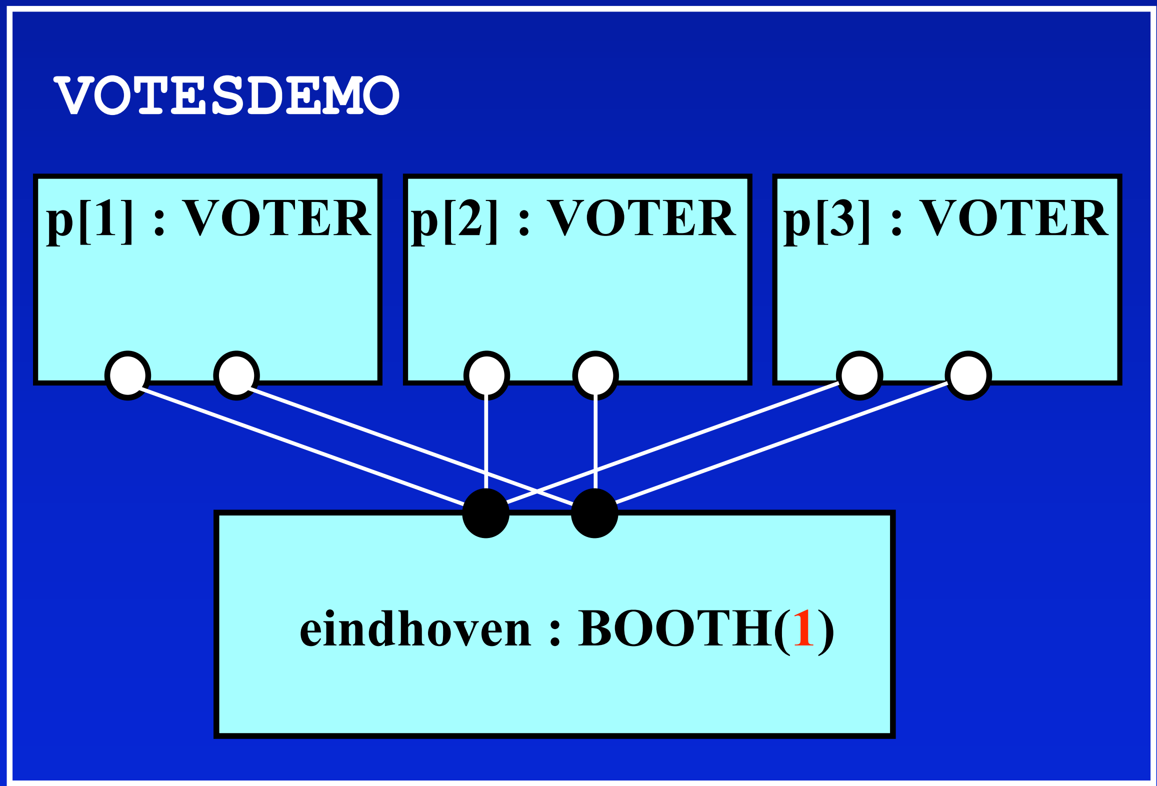


Composite components



Composite component behaviour

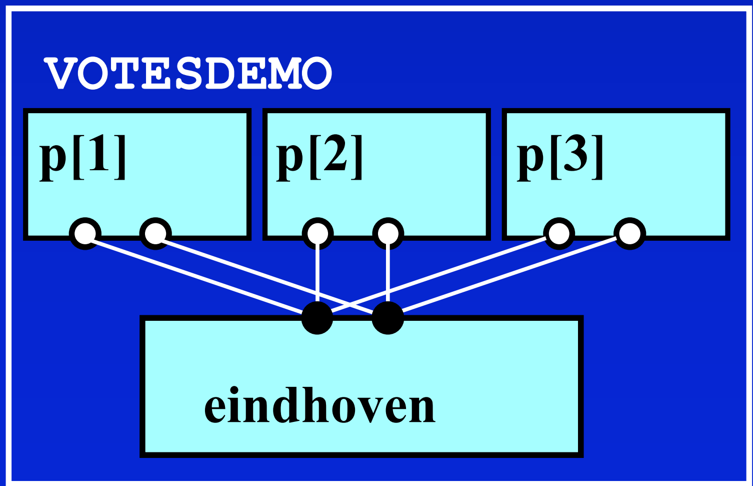
Three voters `p[1..3]` use a shared booth to register their vote. To ensure mutual exclusion ……



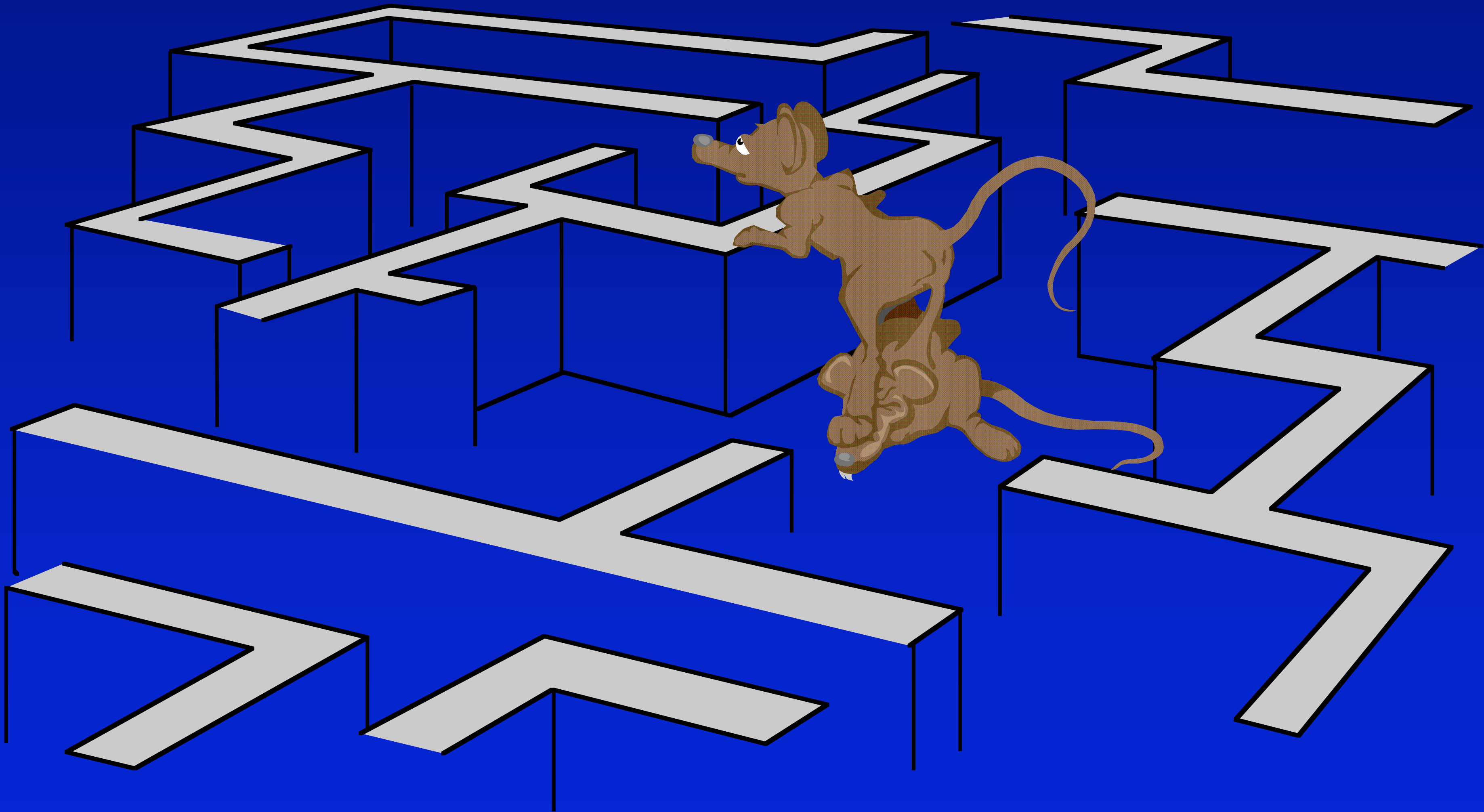
*… the number of spaces available in the booth must be **1**.*

Composite component behaviour

```
||VOTESDEMO = (  p[1..Max]:VOTER
               || eindhoven :BOOTH(1)
                )
    /{p[1..Max].enter/ eindhoven.enter,
      p[1..Max].exit/ eindhoven.exit}.
```

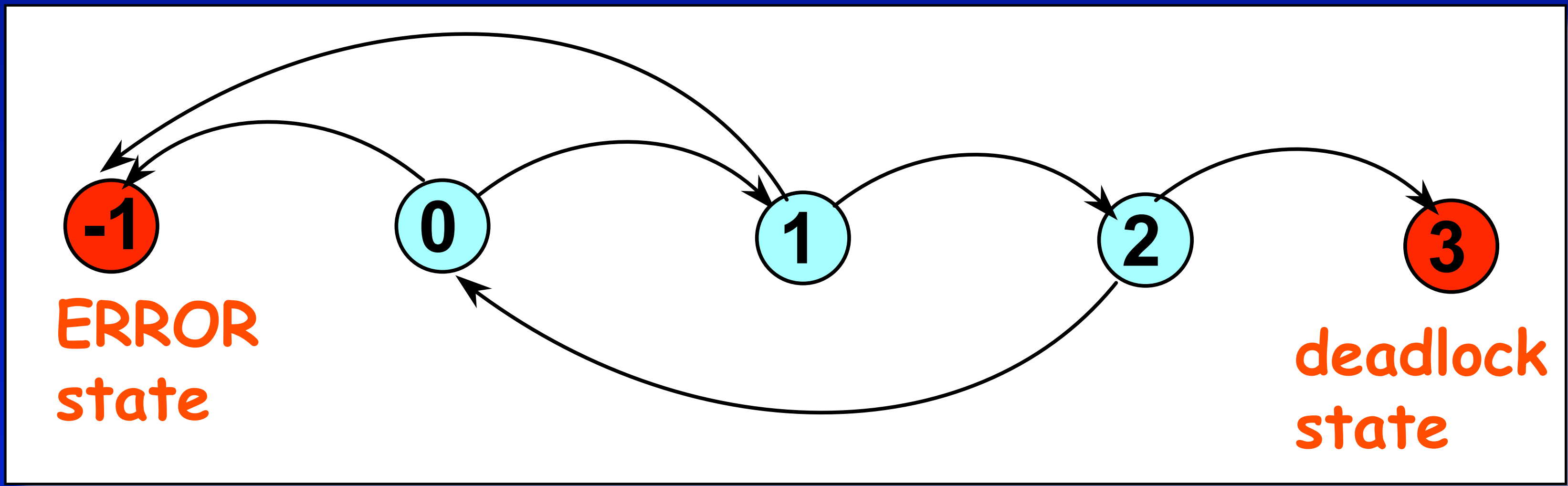


Benefit - behaviour analysis



Model checking

Safety properties are checked by searching the system state space for **deadlock** and **ERROR** states.



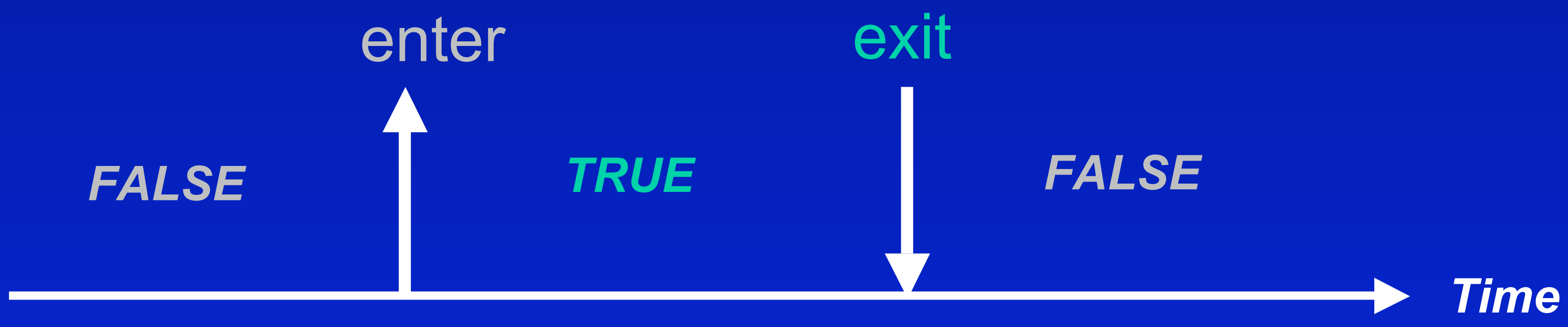
Deadlock is a state with no outgoing transitions.

ERROR (π) is a trap state for property violations.

Property specification

Fluents: abstract predicates or states over sequences of events (from the Event Calculus).

Defined in terms of sets of actions:




```
fluent
  VOTING[i:1..Max] = < p[i].enter , p[i].exit >
                                        initially False
```

[Magee & Giannakopoulou]

Safety Properties

```
assert
    EXCLUSION  = []!(exists[i:1..Max-1]
                  (VOTING[i] && VOTING[i+1..Max]))
```

Behaviour violations transition to the **ERROR** state. Safety properties are violated if the **ERROR** is reachable.

What if the initial value of the booth is 2 ? …or 0?

Liveness Properties

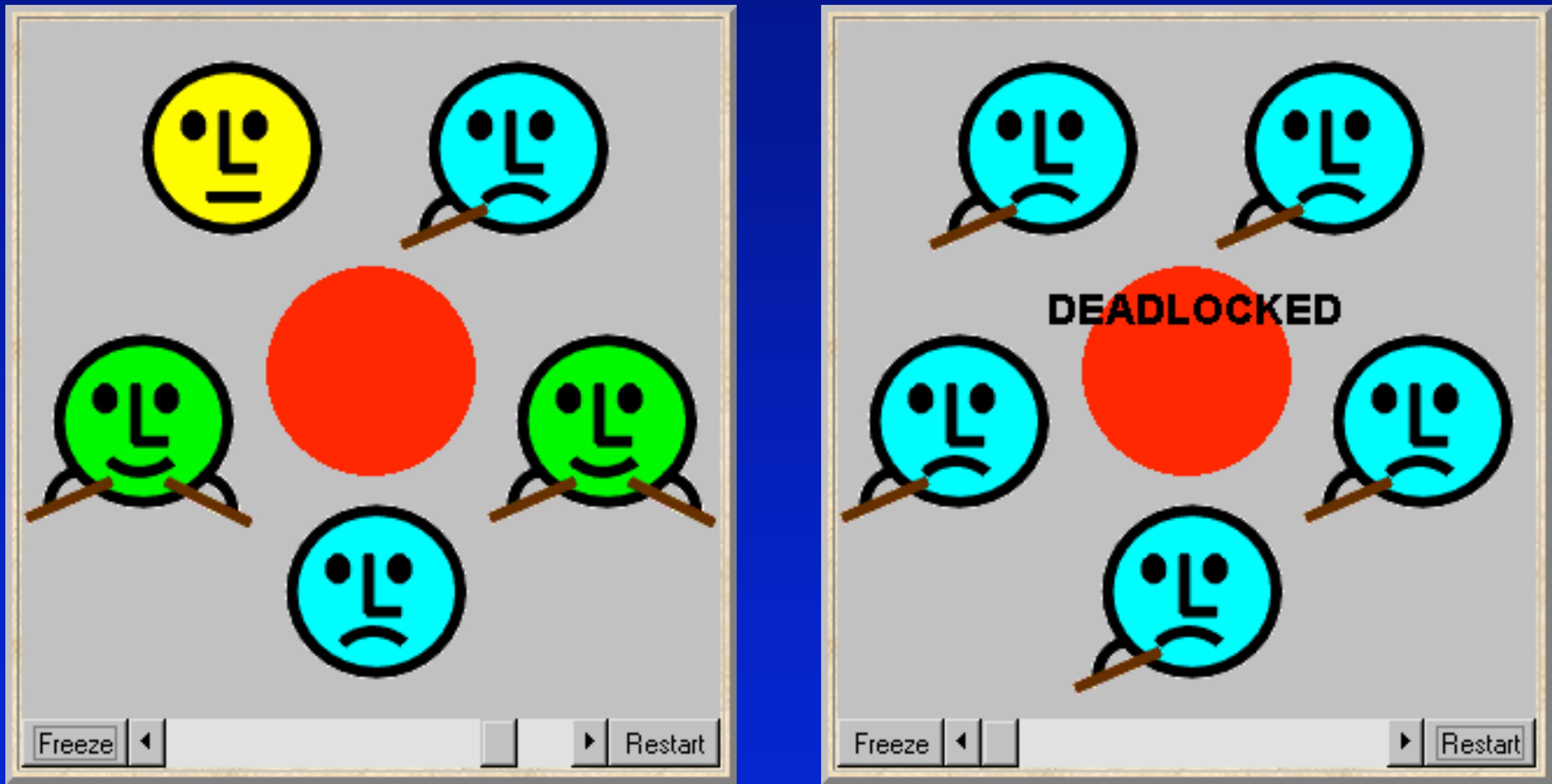
// action label denotes a fluent which is true when
// the action occurs and false immediately the next action occurs

```
assert
        OKtoVOTE = forall[i:1..Max] []<>p[i].enter
```

Use of Linear Temporal Logic LTL for liveness results in the use of Buchi Automata.

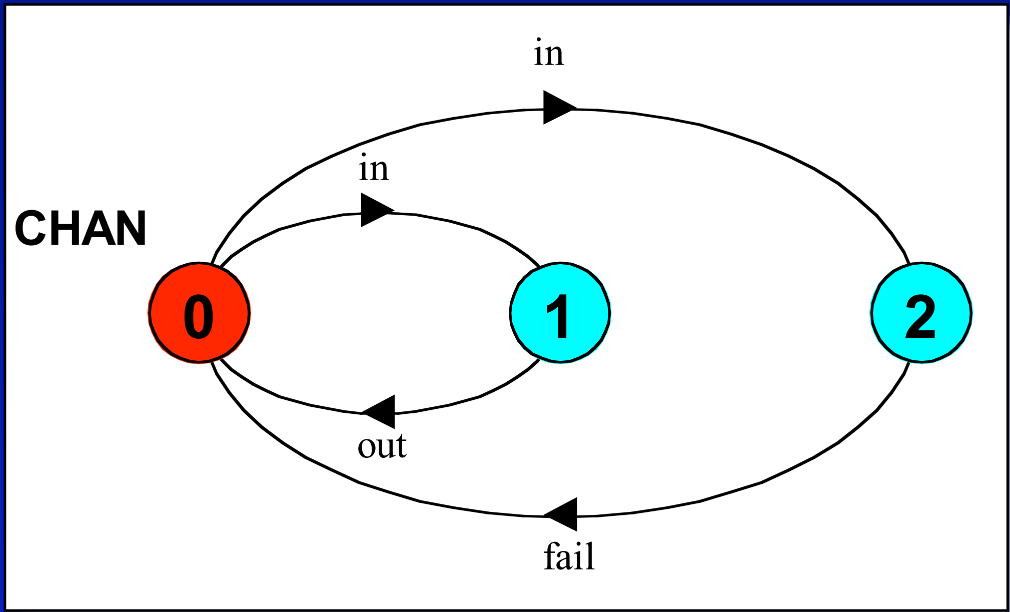
What if we give priority to one of the voters?

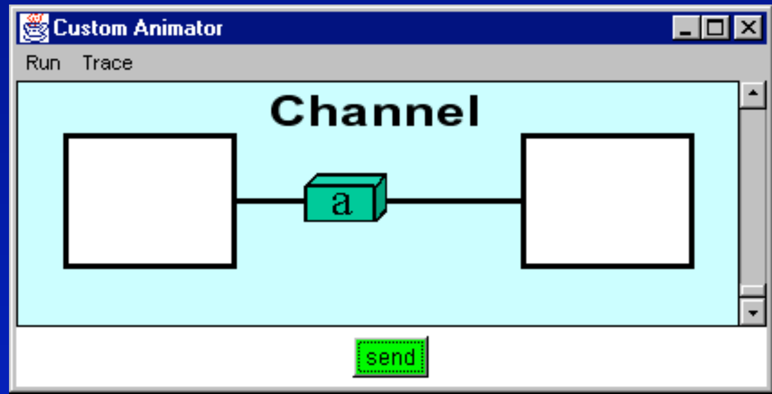
Deadlock – analysis Vs intuition



Dining Philosophers

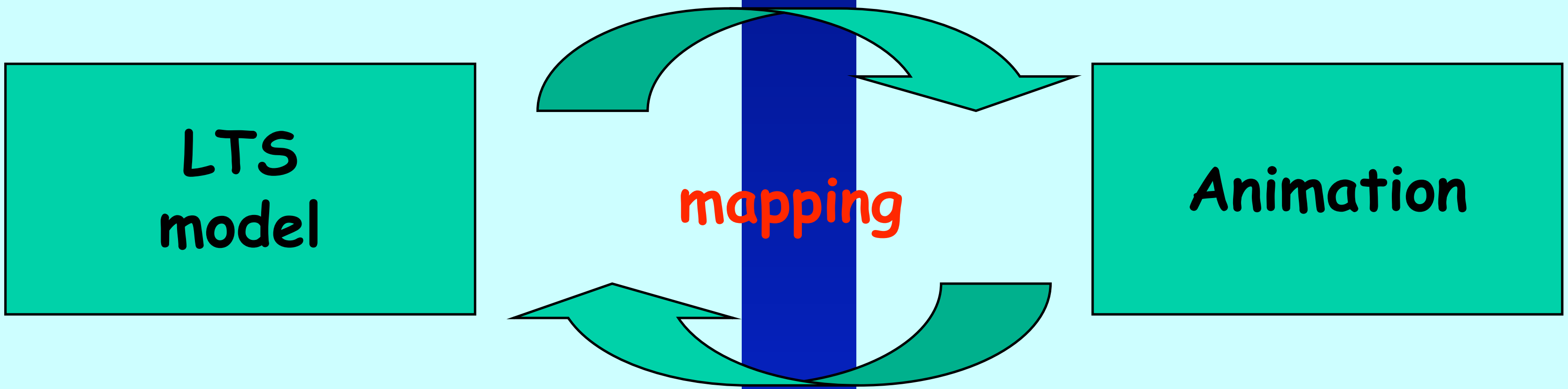
abstract models ⟺ concrete animations






```
CHAN = (in -> out  -> CHAN
       |in -> fail -> CHAN
       ).
```

Model interpretation ⟺ animations

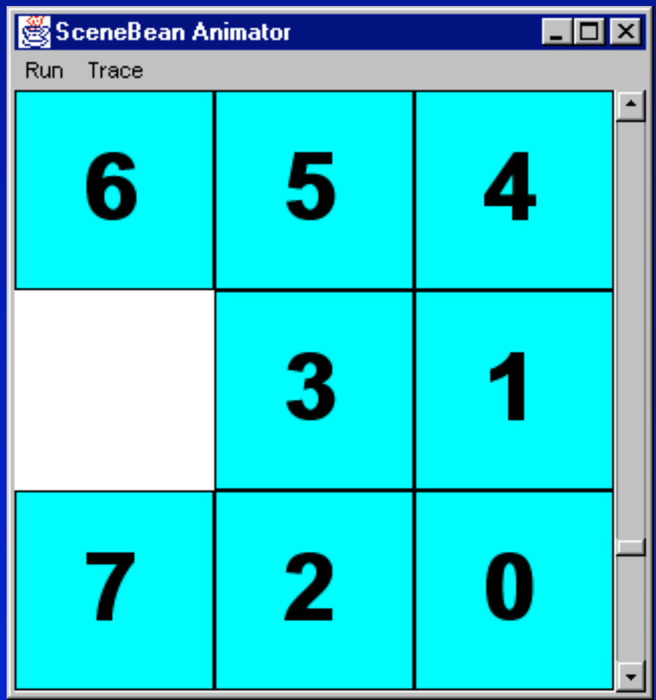


LTS Model checking

- safety properties
- progress properties
- compositional reachability
- abstraction & minimisation

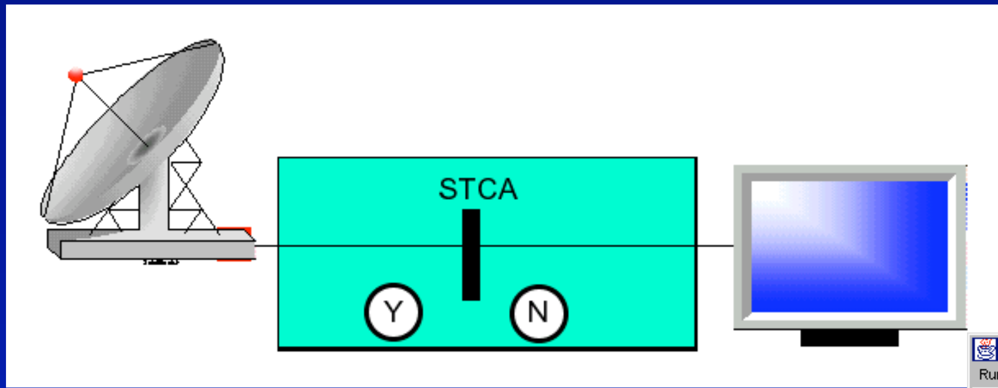
Separate graphic animation model which preserves the behaviour of the model and has sound semantics based on Timed Automata.

Puzzle



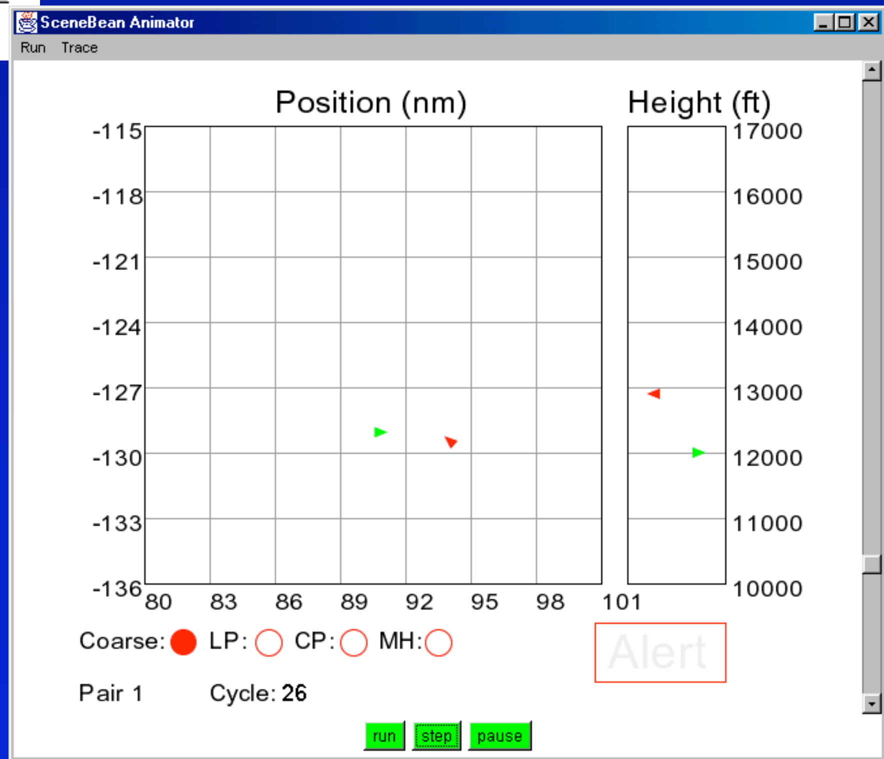
The animated model can be used to help understand the meaning of **counterexamples**.

NATS – short term conflict alert (STCA)

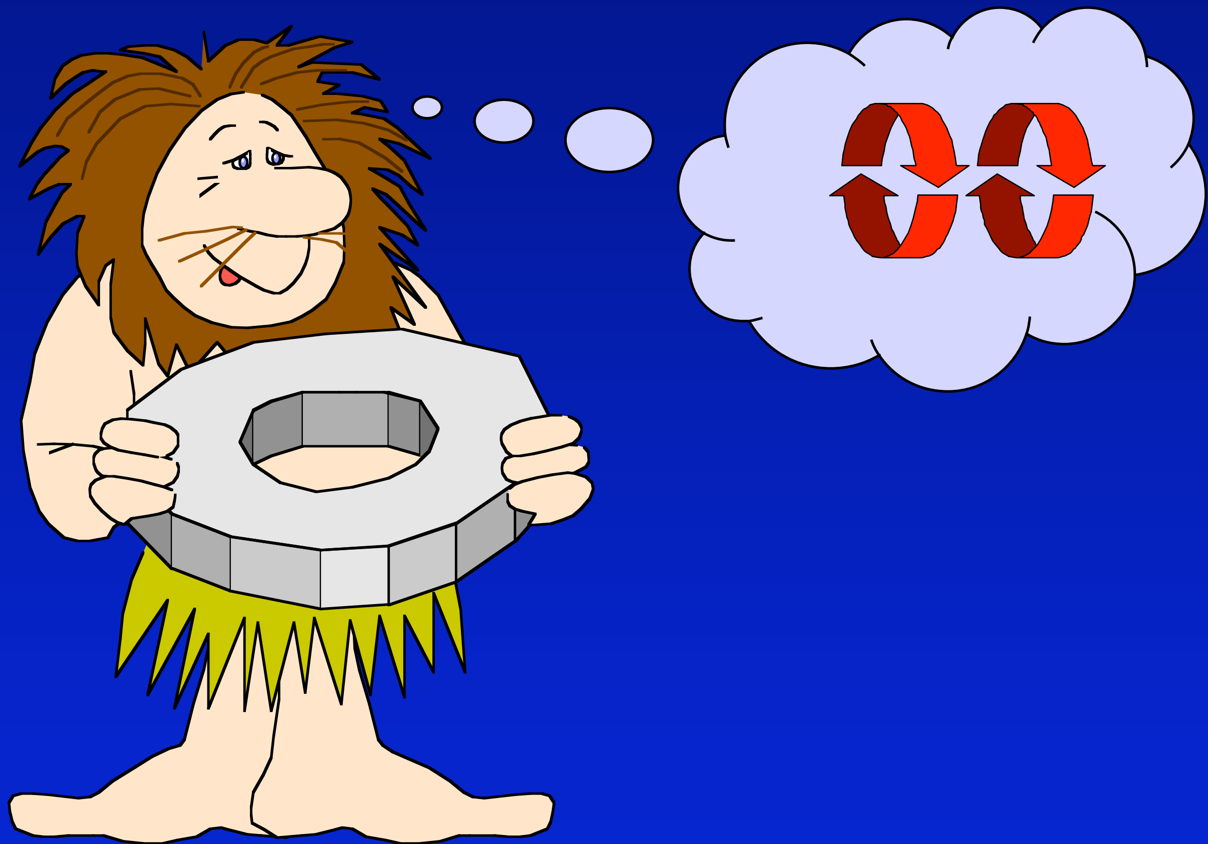


For each pair of aircraft determine potential conflict.

We can construct **hybrid models** that combine the discrete behavioural model with a real valued data stream.



Model based design of concurrent programs



© Kramer

http://www-dse.doc.ic.ac.uk/concurrency/

My teaching experience …..

- Generally very good - the students find the formal models relatively intuitive and helpful in clarifying the problem.
- Comprehension is facilitated by model animation, model checking and simulation.

- **However** – some still seem to find constructing models themselves, ab initio, to be very difficult!

Modelling

- It is not enough to think about **what** they want to model, they need to think about **how** they are going to use that model.
- … **fit for purpose** (Occam's Razor)

Jean-Raymond Abrial (IEEE Computer Sept 2009**)

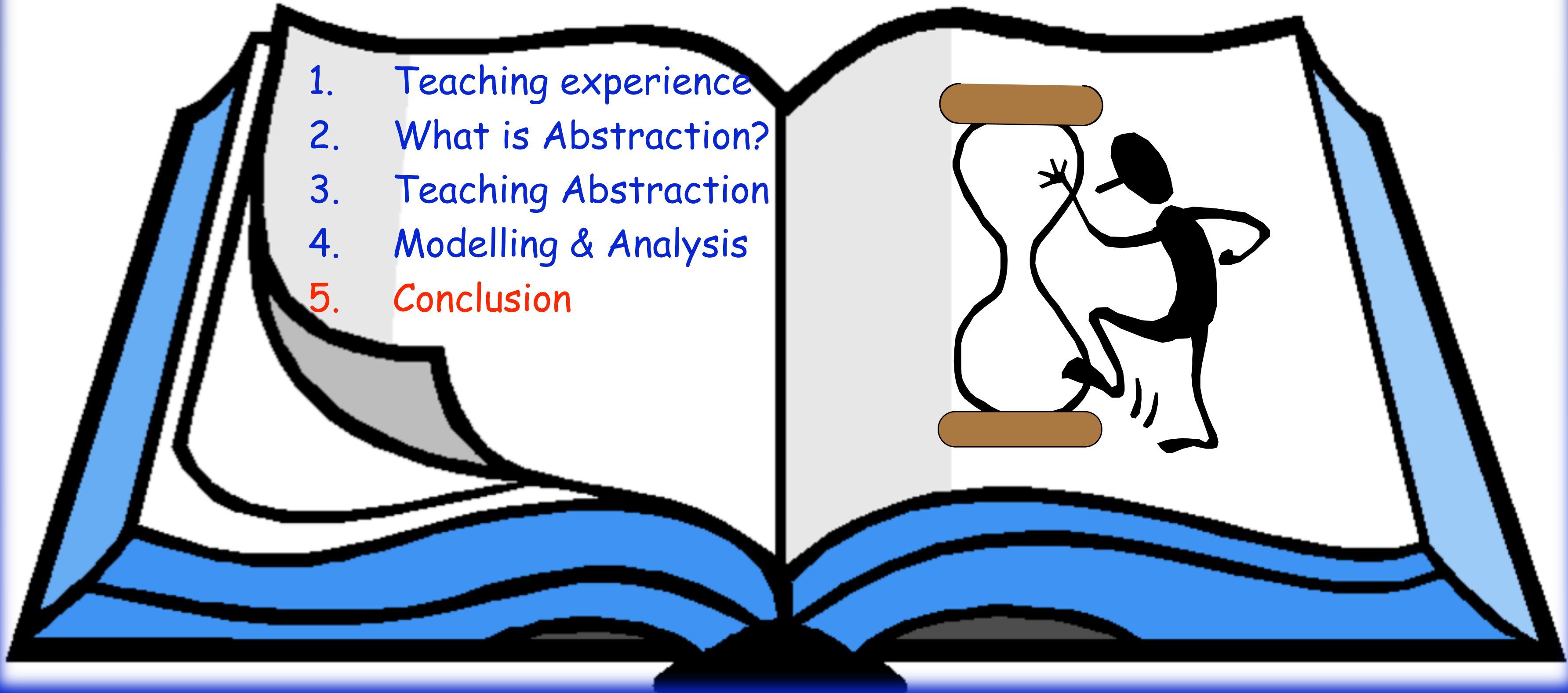
Focus on modelling the problem:

"we have no choice but to perform a complete modeling of our future system, including the software that will eventually be constructed and its environment"

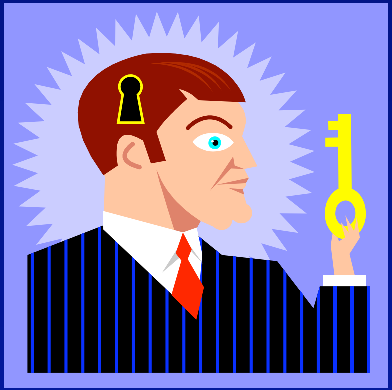
- Use mathematical models:
  - discrete transition systems and proofs
- Animation complements modelling:
  - "directly animating the model"
- Education ?
  - discipline of software engineers is (discrete) mathematics and advocates teaching requirements engineering and construction of mathematical models.

** "Faultless Systems: Yes We Can"

Chapter 5. Conclusion ...



ACM/IEEE Computing Curricula: Software Engineering



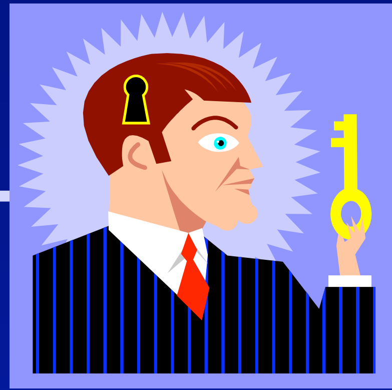
Abstraction:

- Generalization
- Levels of abstraction and viewpoints
- Data types, class abstractions, generics/templates
- Composition

Modeling:

- Principles of modeling
- Pre and post conditions, invariants
- Math models and specification languages
- Model development tools and model checking/validation
- Modeling/design languages (eg UML, OOD and functional)
- …

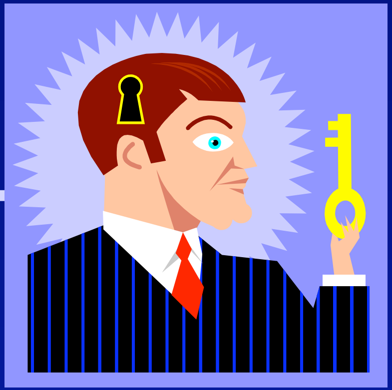
I believe that …



- Abstraction is rarely discussed directly, but is the key to modelling in Software Engineering.

- Students who can understand, appreciate and utilise abstraction produce the most elegant models and software.

Abstraction – the key to Computing?

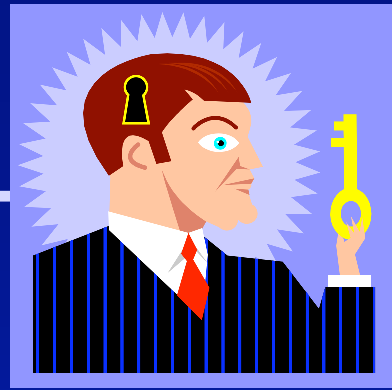


If we want the best Software Engineers, we need to …

- teach them abstraction skills
- consider selecting students for Computing based not only on their school grades, but also on their abstraction abilities?
- *Construct tests to assess formal operational thinking and abstraction*

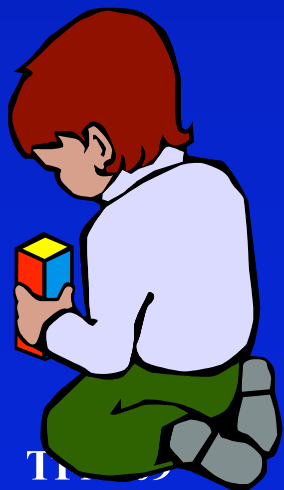
  (working with Orit Hazzan, Technion)

I believe that …



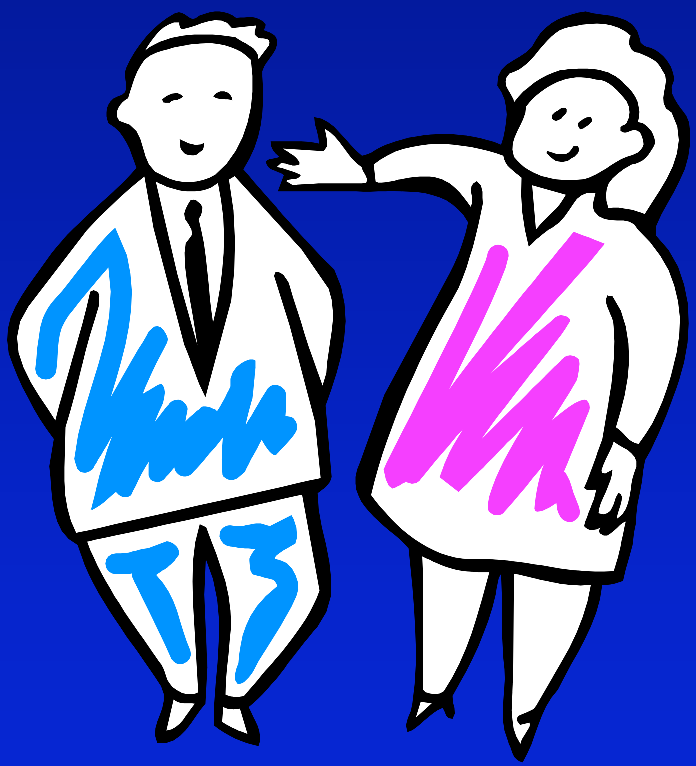
- Formal modelling is an excellent technique for teaching, practising and improving abstraction skills for Software Engineers.

- Abstraction and modelling are complementary.



Abstraction and Modelling

a complementary partnership **



Thank you.

** See "Is Abstraction the key to Computing?" CACM April 2007