

Capita Selecta Information Systems
Formal Methods

Erik Poll

Digital Security group
Radboud University Nijmegen

Goals of this part of the course

- What are **formal methods** ?
 - formal methods vs informal methods,
formal **models** vs informal models
- Why would you use them ?
- Some practical experience with formal methods and formal method **tools**

(In)formal methods for architects

(In)formal methods for architects

- Which methods & models are used when constructing a building ?
 - Why are they used ?
 - By whom are they used ?
 - Are they formal or informal?
-
- How does all this compare/relate to the constructing of information systems?

Models

- 3D drawings
- computer animation
- scale model
- life size model
- plans
 - 2D floor plan or cross-sections
 - of specific aspects (electric wiring, plumbing, heating, ventilation, ...)

The roles of models

- **Communication**, eg
 - between architect and client, or architect and builder
- **Validation**, eg
 - is this what customer wants?
 - is it "ok" in other respects?
 - Can doors open? Is there enough daylight & ventilation? Does it look ok? ...
- **Calculation**, eg
 - calculation weight of building to decide thickness of walls
- **Specification**, eg
 - precisely telling that builder what to do
- **Verification**
 - checking that builder did it right, and if not getting them to fix it or sueing them

The role of *formal* models

- *Formal* models have a precise and unambiguous format & meaning
- This is useful for
 - precise communication & specification
 - validation & verification, using mathematical reasoning or calculation
 - quality
 - all successful engineering disciplines use formal models

What is a formal method?

- **formal language**, with a precisely defined **syntax** and **semantics**, that allows some **formal reasoning or calculation**
 - essentially, this is the definition of **mathematics**
- can be used to describe - ie to **model** - something
 - that already exists in the real world, or
 - that we want to construct in the real world, ie. as a specification or requirement
- models only capture some aspects of the real world, abstracting away some details

Example formal methods and models

- differential equations used to model all kinds of physical phenomena
- maps and plans
 - eg used to calculate routes
- balance sheet is a formal model of a company

Formal methods in computer science

For computer science, it is not so clear

- what useful models and methods are
 - what kind of mathematics we should use
- and there is little consensus about this

Probably not geometry or differential equations...

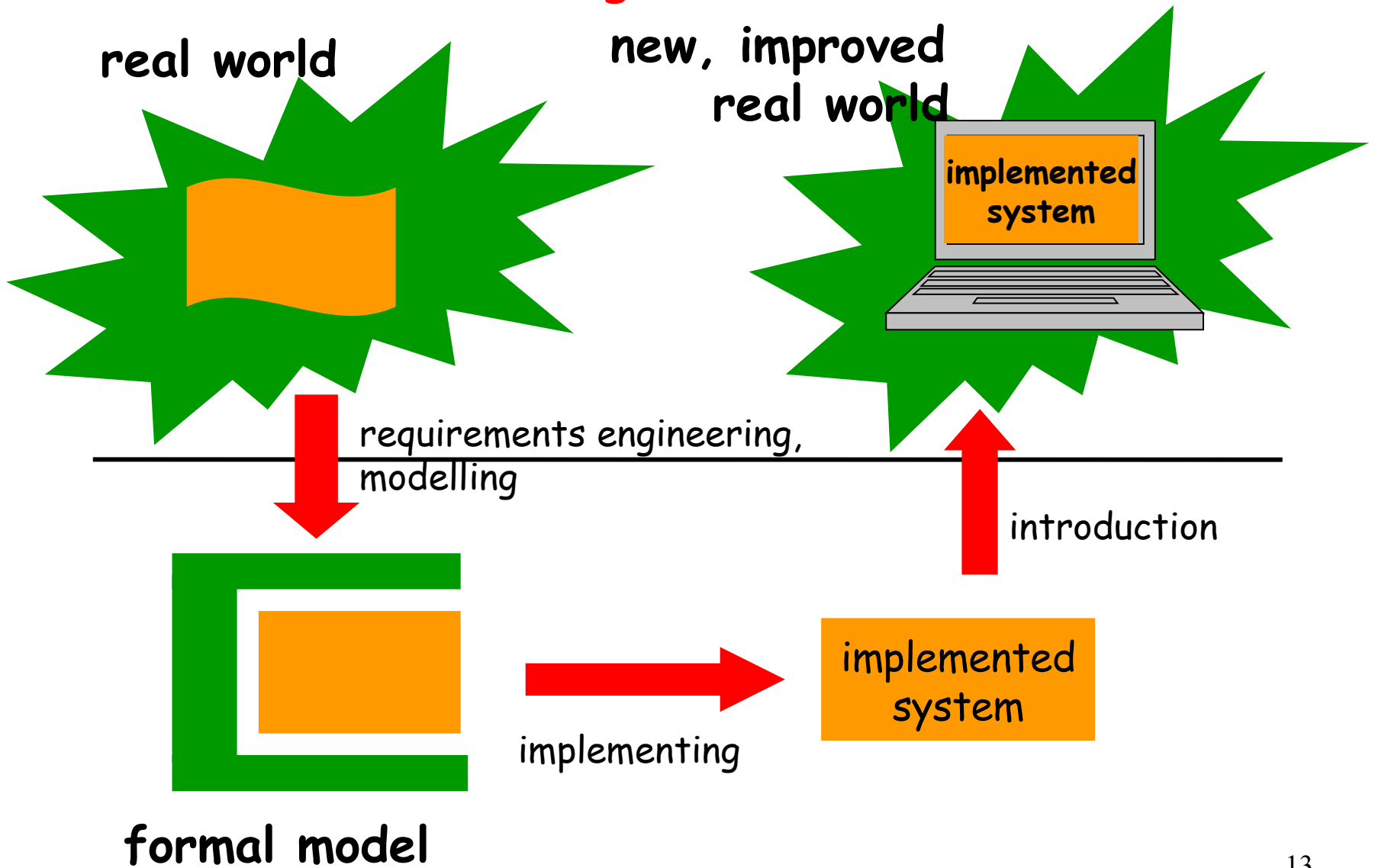
Example formal methods

- logics
 - eg proposition \sim , predicate \sim , temporal \sim , ...
- ORM, UML, OCL, Z, B, VDM, ...
- finite state machines, labelled transition systems, process algebras, Petri nets, YAWL,....

Some of these are at best only **semi-formal**

eg UML has (lots of!) well-defined syntax, but no well-defined, formal semantics

Idealised Software Development Life Cycle using formal methods



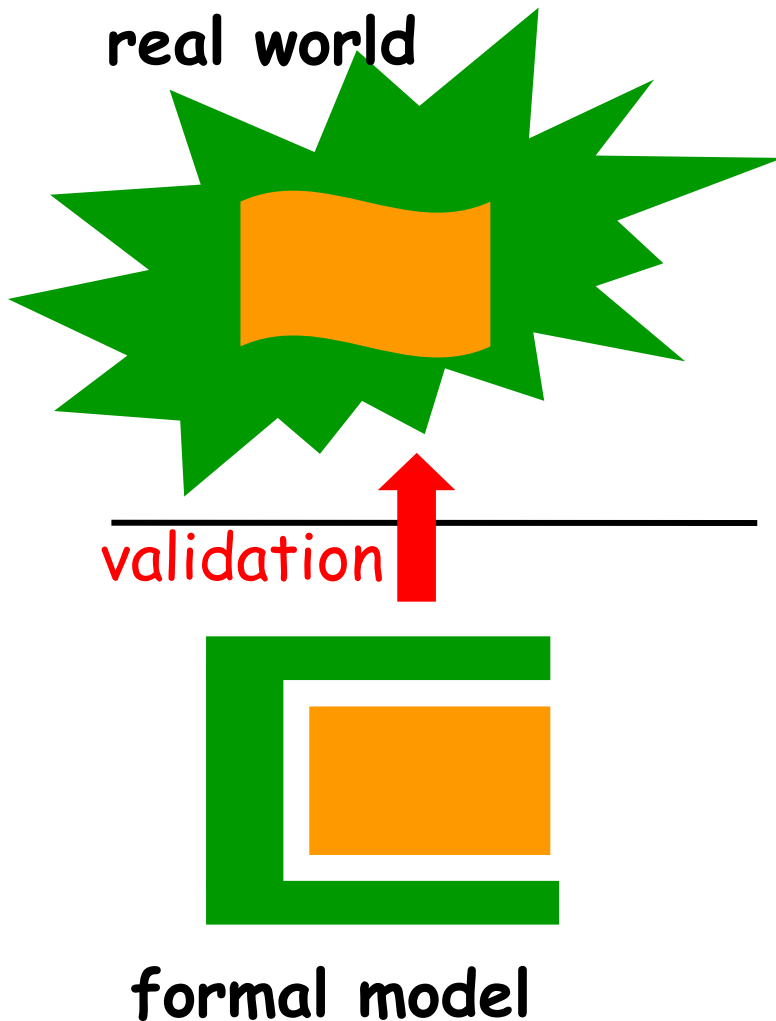
A formal method provides

- a formal language for defining models that can serve as specifications or requirements
- associated methods for
 - analysing such models
 - checking these models against implementations

Validation

we can **validate** models
against the **real world**

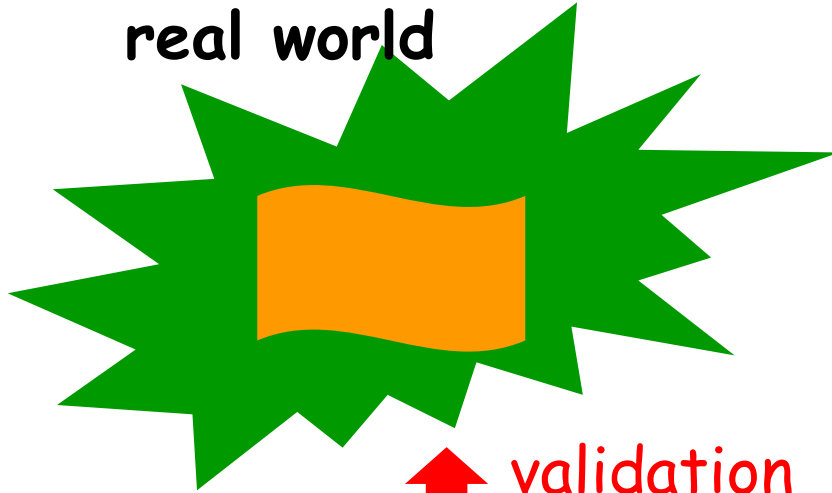
eg by talking to
domain experts



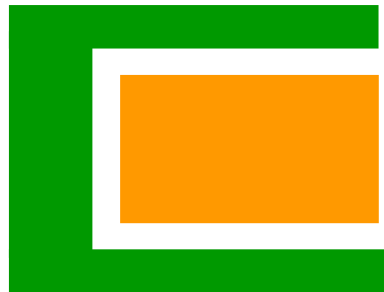
Analysis

- We can **analyse** the model by itself, eg to see
 - are there inconsistencies ?
 - check that it has desired properties ?
- We can use **tools** to automate this, because the model is formal. Such tools include **theorem provers** and **model checkers**

real world



validation



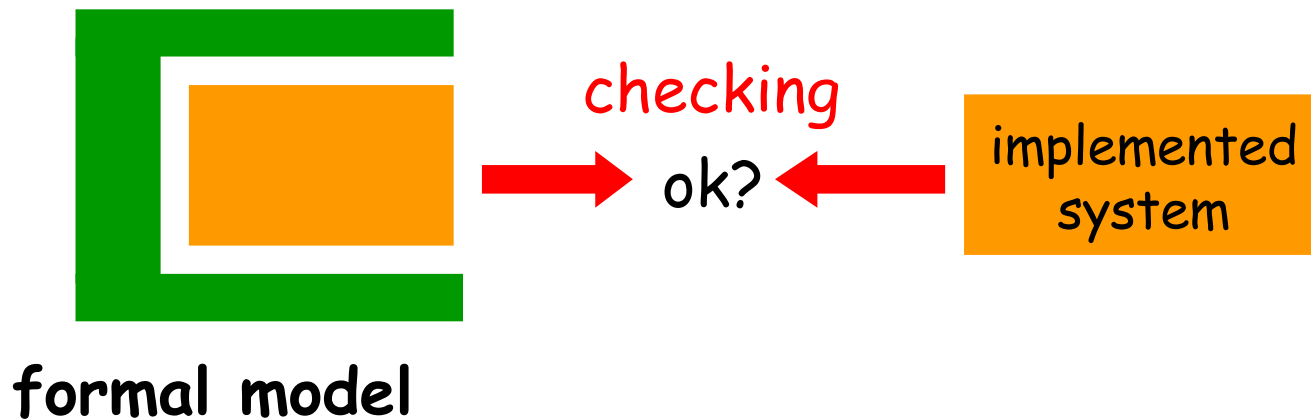
analysis formal model

Checking: verification and testing

We can compare formal model and implementation to check if implementation meets specification

- by **testing**, ie. model-based testing
- by (real) **verification**, ie. mathematical reasoning

NB because model is formal, we can use **tools** to automate this!



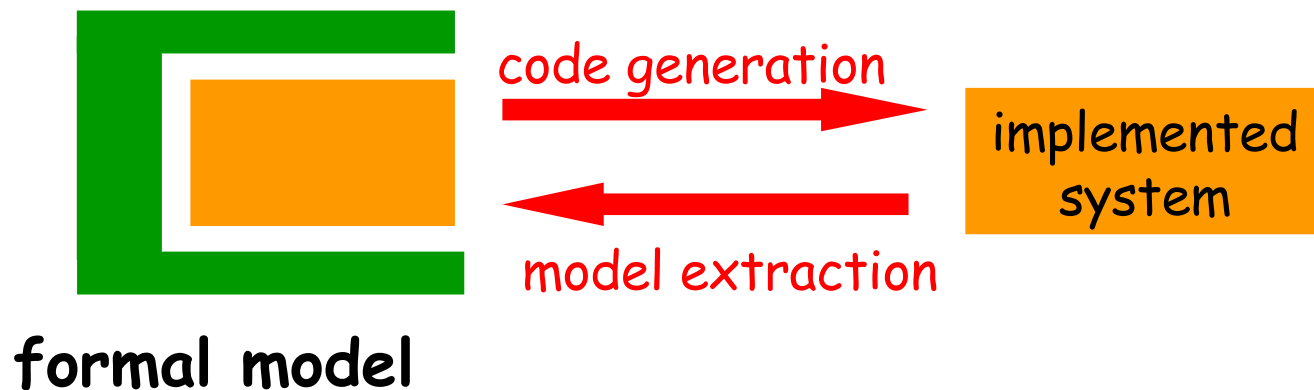
Verification vs testing

- verification can provide **stronger guarantees** than testing, as it considers all cases, whereas the guarantees obtained by testing are only as good as the test suite
- but ... verification is usually **a lot more work**
- Warning: some people, esp. in industry, call testing verification

Formal model and implementation

More ambitious possibilities

- generating code from formal model
- or extracting formal models from code



Recap

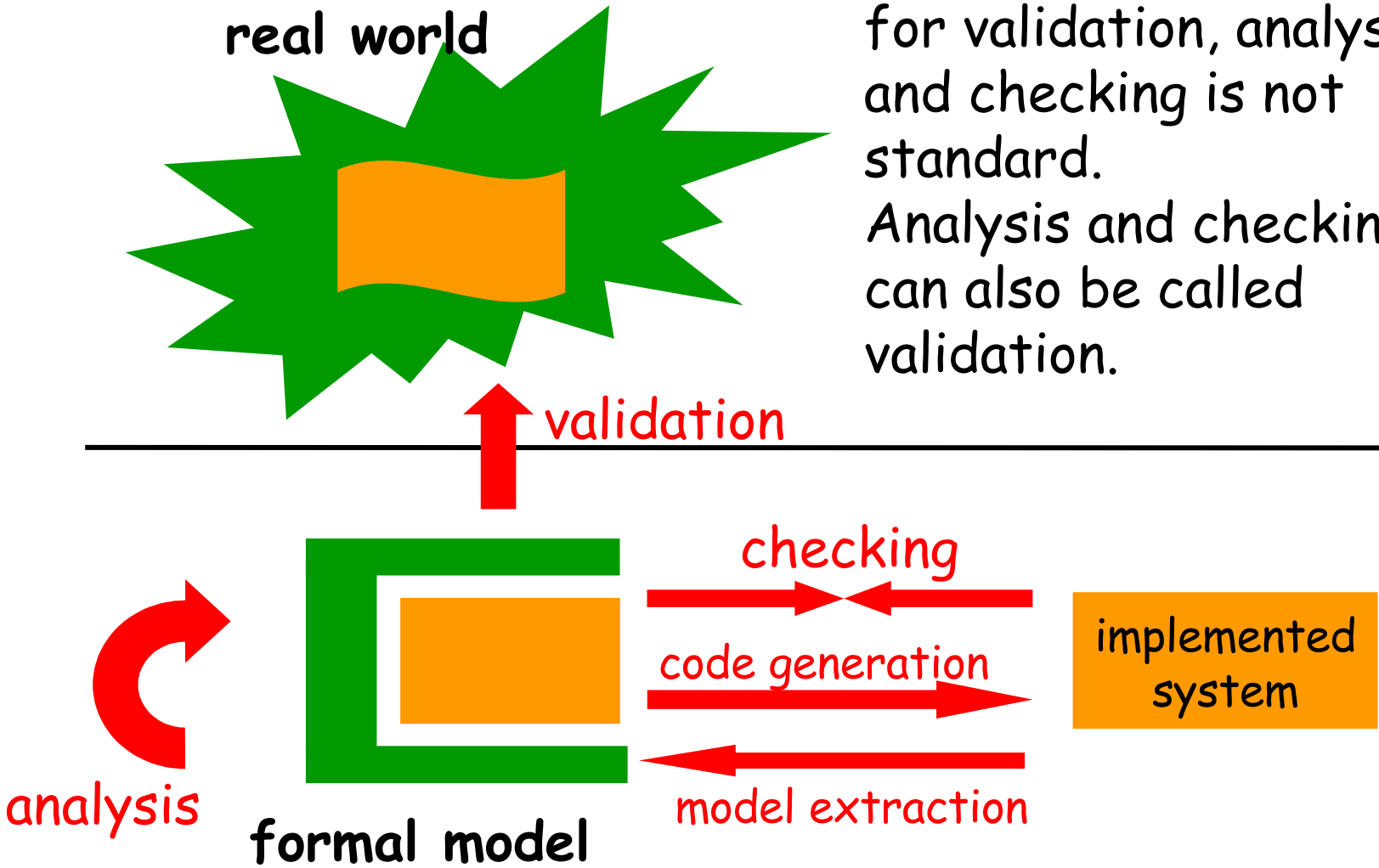
A formal method provides

- a formal language for defining models that can serve as precise & unambiguous specifications or requirements
- associated, possibly tool-supported, methods for
 - analysing such models
 - checking these models against implementations
 - by verification or testing

and possibly also for

- generating implementations from models
- extracting models from implementation

Warning: terminology for validation, analysis, and checking is not standard. Analysis and checking can also be called validation.



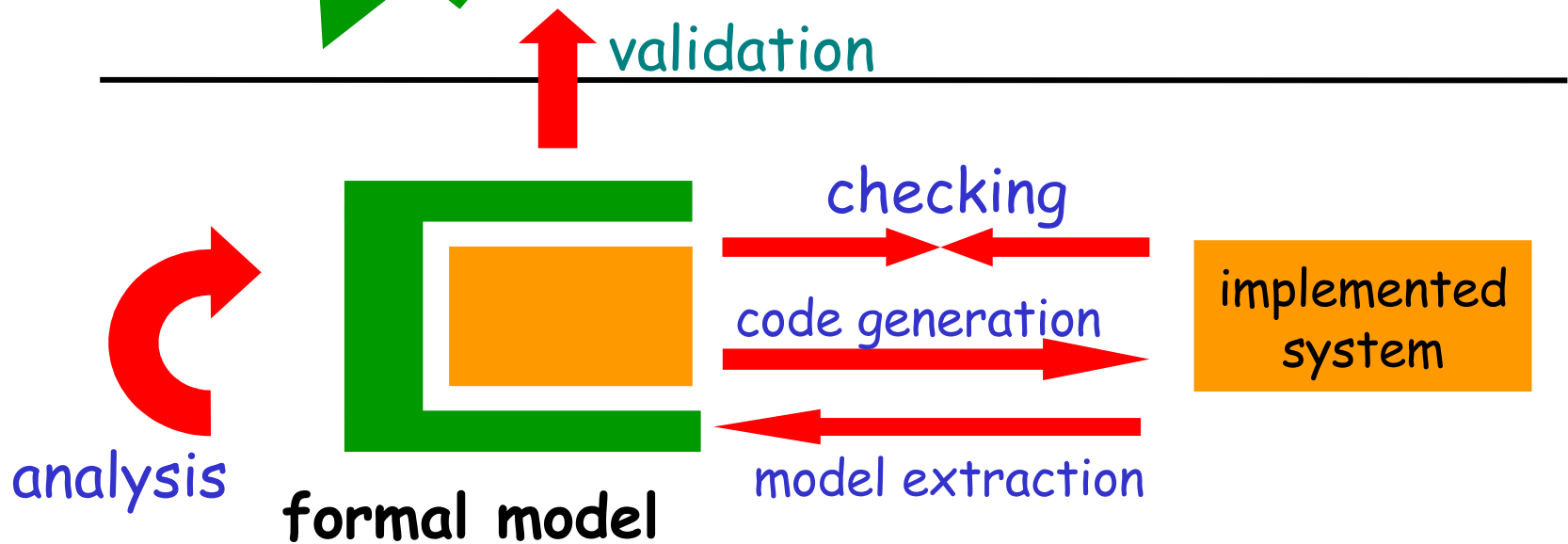
Terminology

- **Analysis** of the model is sometimes called **validation** or **verification** (of the model)
- **Verification** is sometimes called **validation** (of the implementation wrt. the formal model)
- **Testing** an implementation is sometimes called **verification**
 - common in industry, but academics do not approve

real world

Validation must involve humans.

Below the line we can use tools



Background for practical work
on Thursday

Overview

- The assignment involves
 - formalising some constraints, as used in ORM, in a formal specification language, JML
 - verifying these constraints for a Java program, using a verification tool, ESC/Java2
 - if a constraint is violated, fixing the program (or maybe the constraint, if that was wrong...)
- Goal (hopefully):
 - see how formal specifications and tools can help in improve the quality of a system
 - but also.. the price you have to pay for this

- The formalism that JML uses is essentially **first-order logic**
- The tool we will use, ESC/Java2, relies on an **automated theorem prover**.
 - Just like a calculator can do arithmetic, a theorem prover can do first-order logic (ie. prove formulae in first-order logic), and much faster & more reliable than any human
 - NB one of the benefits of formal methods of the possibility of such tools

Java recap & terminology

```
class Person {  
    int age;          // field  
    String name;  
    Bankaccount account;  
  
    // constructor  
    Person (String str) {  
        age = 0;  
        name = str;  
    }  
  
    // method  
    void birthday() {  
        age = age+1;  
    }  
}
```

JML

- JML is a formal specification language for Java
- It can be used to express properties in **first-order logic** (and more, but we won't use that), using a Java-like syntax
- We will only use two kinds of properties
 - **invariants**
 - **preconditions**

propositional logic JML syntax

AND	&&
OR	
NOT	!
=	==
≠	!=
⇒	==>
⇔	<==>

Eg $x \geq 22$ AND $x \neq 23 \Rightarrow C \geq 25$ OR $C = 25$

is written as

$x \geq 22 \ \&\& \ x \neq 23 \ ==> \ C \geq 25 \ || \ C == 25$

Invariants

- An **invariant** is a property that should always be true of an object.

```
class Person {  
    int age;  
    //@ invariant age >= 0;  
    ...  
}
```

- In ORM terminology, such a property would be called a **constraint**

Invariants

- When we say that an invariant should always be true , we mean
 - it has to be true at the end of the constructor
 - it has to be true at the beginning and end of each methods

So during a constructor or method, an invariant may temporarily be broken, but at the end it has to hold.

Preconditions

- A **precondition** is a requirement that must hold if a constructor or method is called. For example

```
class BankAccount{
    int balance;

    //@ requires amount >= 0;
    void debit(int amount) {
        balance = balance - amount;}
}
```

Often, preconditions are needed to prevent invariants from being broken!

- Long preconditions and invariants can be split over several lines using `/*@ ... @*/` instead of `//@`

For example

```
/*@ invariant age < 18
    ==>
    account.balance >= 0
@*/
```

(What is the informal meaning of this invariant?)

non-nullness

- Many preconditions and invariants are about references not being null, for example

```
class Person {  
    String name; //@ invariant name != null;  
  
    //@ requires str != null;  
    Person (String str) {  
        age = 0;  
        name = str;  
    }  
}
```

Why use JML ?

Advantages of *formal* specification in JML, instead of informal comments or javadocs:

- it is **precise** and **unambiguous**
 - eg do we mean amount > 0 or amount ≥ 0 ?
- **tool support**
 - **testing**: using the runtime assertion checker, violations of JML specs can be detected when executing Java code (**at runtime**)
 - **verification**: using program verification tools we can detect the possible violation without even executing the code (**at compile time**)

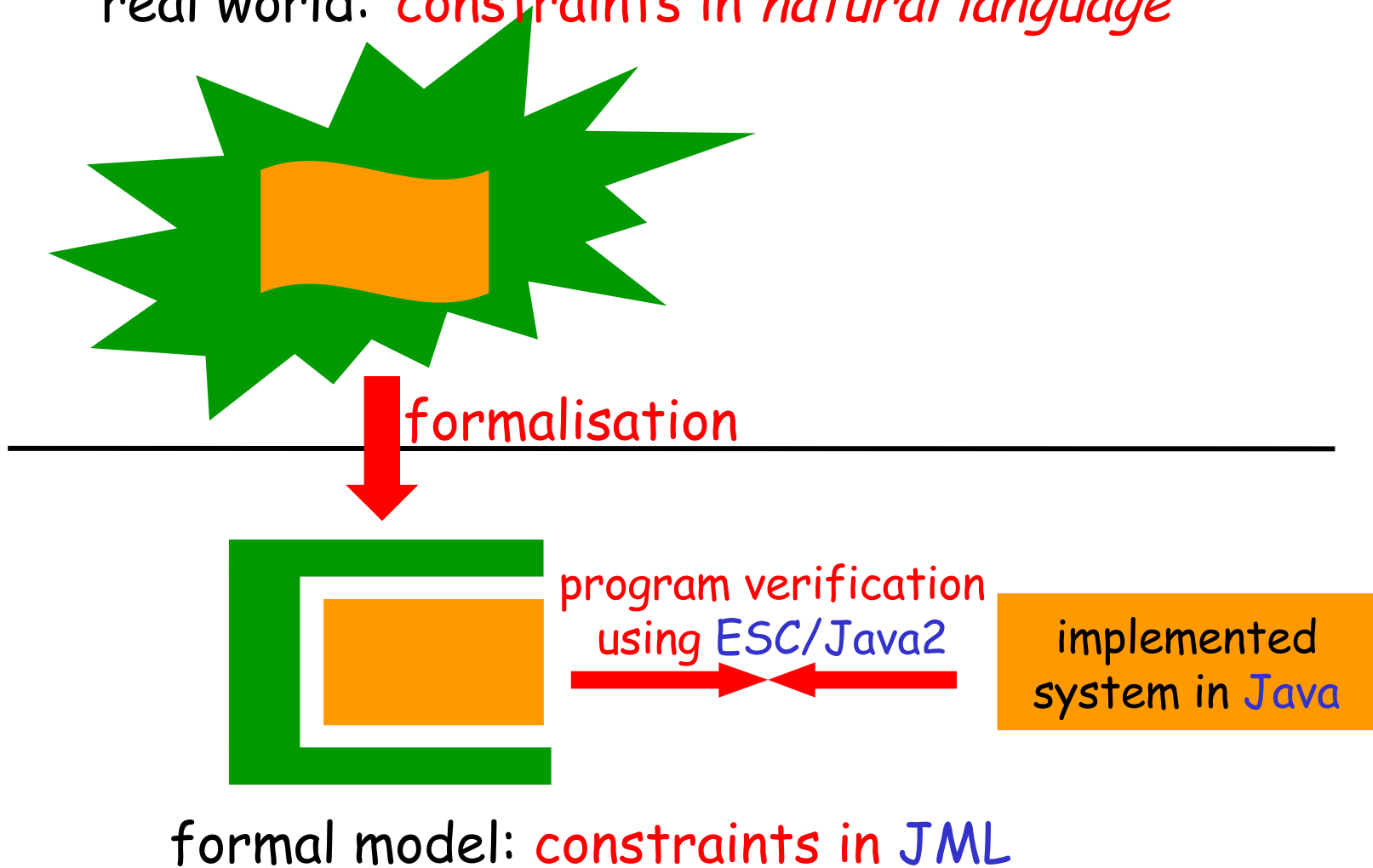
JML specification vs Java implementation

Relation with earlier discussion:

- ESC/Java2 is an automated tool that (mathematically) **verifies** if the Java code meets the JML spec
- Note: the formal spec in JML is (usually) only a **partial** specification

Practical assignment

real world: constraints in *natural language*



formal model: constraints in JML

Reflection after Thursday

Things I'll hope you notice

- formal specifications can help to reveal errors
- formalising can be difficult, but forces you to be precise
- some things may be so obvious that you forget to specify them (eg $\text{age} \geq 0$)
 - people make many *implicit* assumptions!
 - as an outsider in a domain, these may not be obvious for you
 - for the verification tool you have to make all such assumptions *explicit*
- *but how good is this formal method for communication, validation, computation, ...?*