An Introduction to Formal Modelling

DI/UM, 2002-06

J.N. Oliveira

Cover Story

Excerpt of article in the CAMBRIDGE EVENING NEWS:

Computer Scientist Gets to the "Bottom" of Financial Scandal

*A Cambridge computer professor, Simon Peyton Jones, has made an interesting discovery regarding the **Enron** collapse. (...) Enron's collapse was due to a nearly impenetrable web of financial contracts that disguised the true financial state of the company (...)*

Cover Story (cont.)

(...) Accountants find that even when they are scrupulously honest about the valuation of such contracts there can still be sharp disagreements in regard to the worth of trading reserves, debts, and other components.

Enter Professor Peyton Jones. As part of his research at ***Microsoft*** *in Cambridge, he developed a computer language for describing and valuing financial contracts. (...)*

Cover Story (cont.)

*(...) With colleagues Jean-Marc Eber and Julian Seward, they developed a language capable of **accurately** describing and valuing even the most complex financial instruments. (...)*

> *"While accountants find financial derivatives to be mysterious and difficult, for us they are just ordinary **recursive equations**,"*

says Jones.

Cover Story (cont.)

(...) "We have been dealing with these for many years and have developed a wide range of techniques for handling them."

(...) According to Peyton Jones, his success in the financial world comes from years of research in ***Haskell*** *(...)*

"Without the tools developed by the ***Haskell*** *community I would never have been able to do what I've done. It's a jolly wonderful way to program computers"*

he stated. (...)

Cover Story (conclusion)

(...)

The Arthur Anderson accounting firm is rumored to have made overtures to Peyton Jones. (...) But Professor Peyton Jones plans to remain where he is.

*"I'm flattered that my research has finally been of use to someone but I'm quite happy working on **Haskell**. Besides, I don't want to have to wear a suit to work every day."*

Cover Story (conclusion)

(...)

...hur Anderson accounting firm...
...ored to have made overtures to Peyto...
ones. (...) But Professor Peyton Jones
plans to remain where he is.

*"I'm flattered that my research has finally been of use to someone but ...m quite happy working on **Haskell**... I don't want to have to...*

(CAMBRIDGE EVENING NEWS,

Cover Story (conclusion)

(...)

Arthur Anderson accountin
mored to have made overtures t
Jones. (...) But Professor Peyton Jo
plans to remain where he is.

"I'm flattered that my research
ally been of use to some

(CAMBRIDGE EVENING NEWS, 1st of April (!) 2002)

Prof. Peyton Jones' "magic words"

- . . . language capable of **accurately** describing and valuing . . .
- . . . just ordinary recursive **equations** . . .
- . . . tools developed by the **Haskell** community

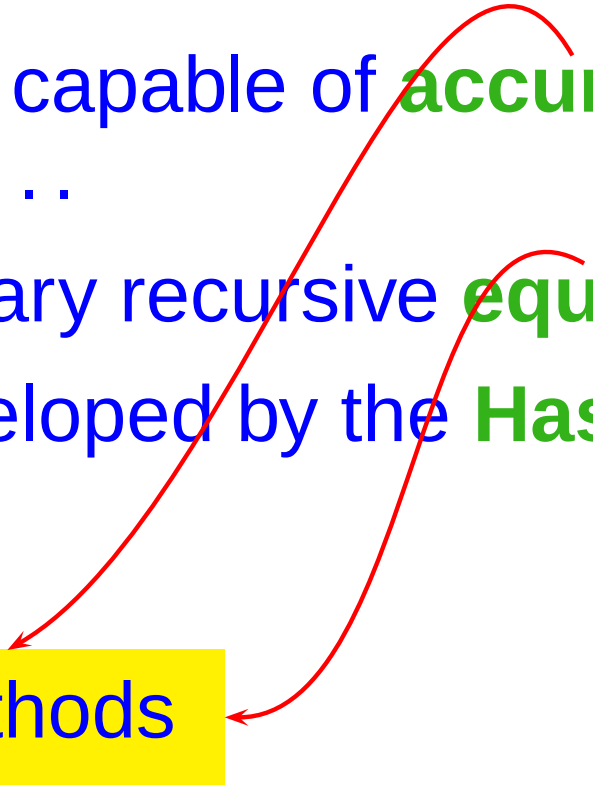
In other words:

Prof. Peyton Jones' "magic words"

- . . . language capable of **accurately** describing and valuing . . .
- . . . just ordinary recursive **equations** . . .
- . . . tools developed by the **Haskell** community

In other words:

- formal methods



Prof. Peyton Jones' "magic words"

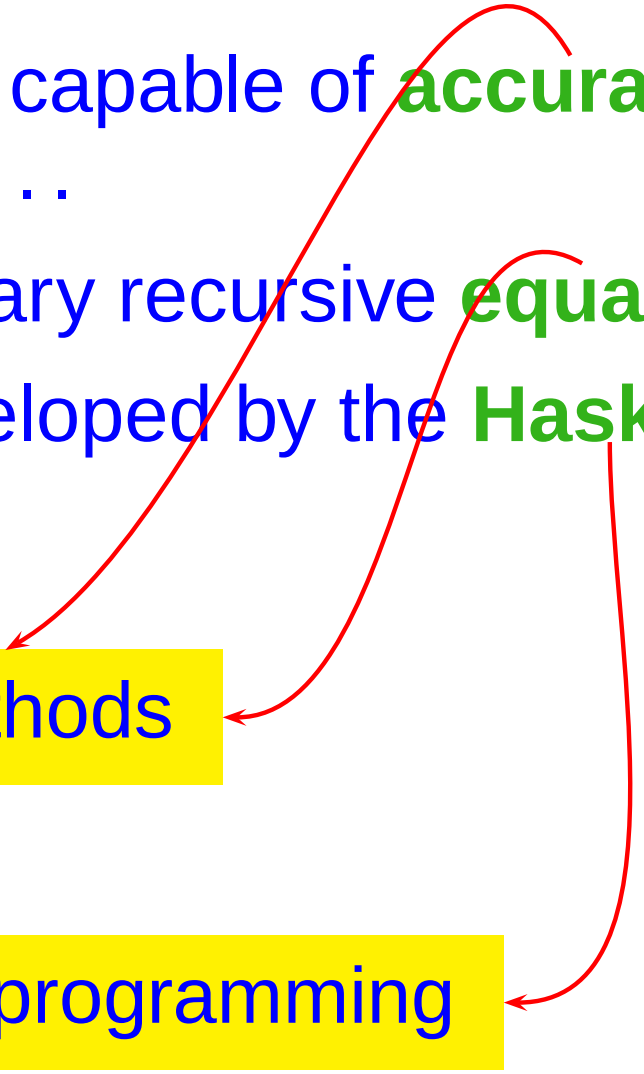
- . . . language capable of **accurately** describing and valuing . . .
- . . . just ordinary recursive **equations** . . .
- . . . tools developed by the **Haskell** community

In other words:

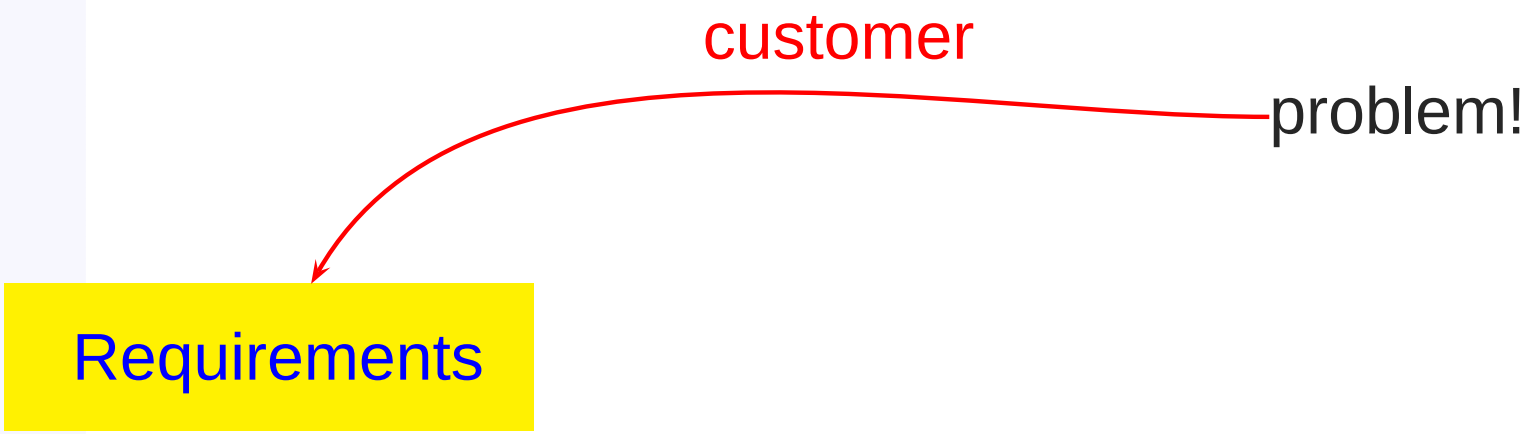
- formal methods

and

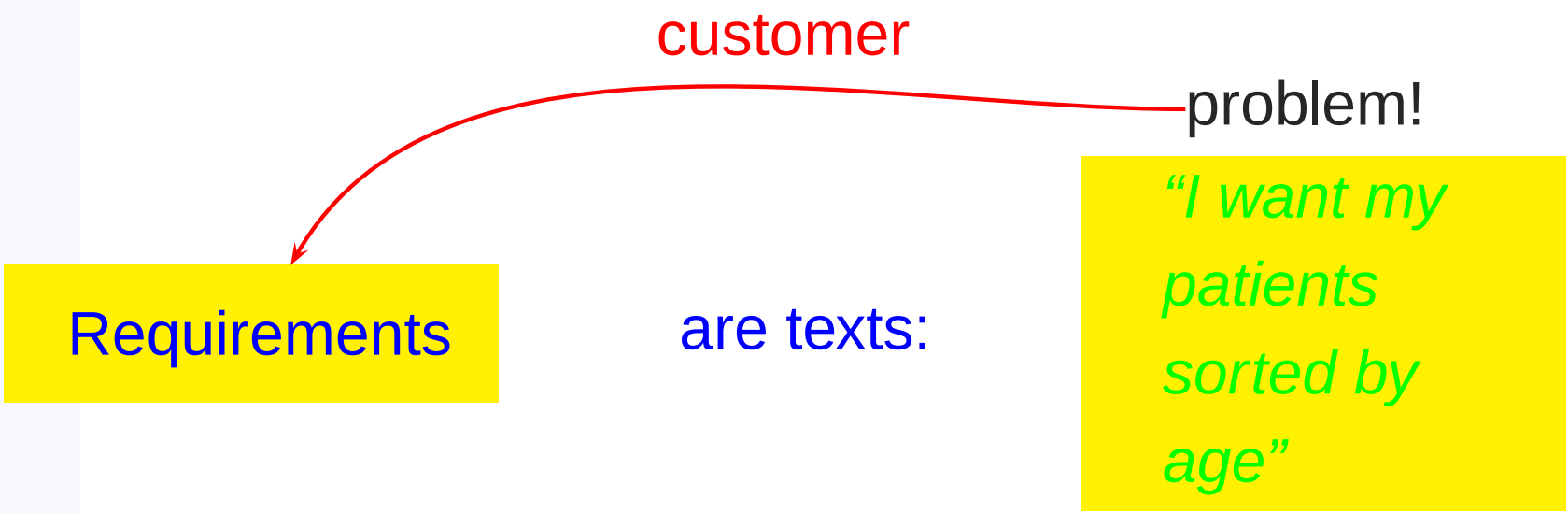
- functional programming



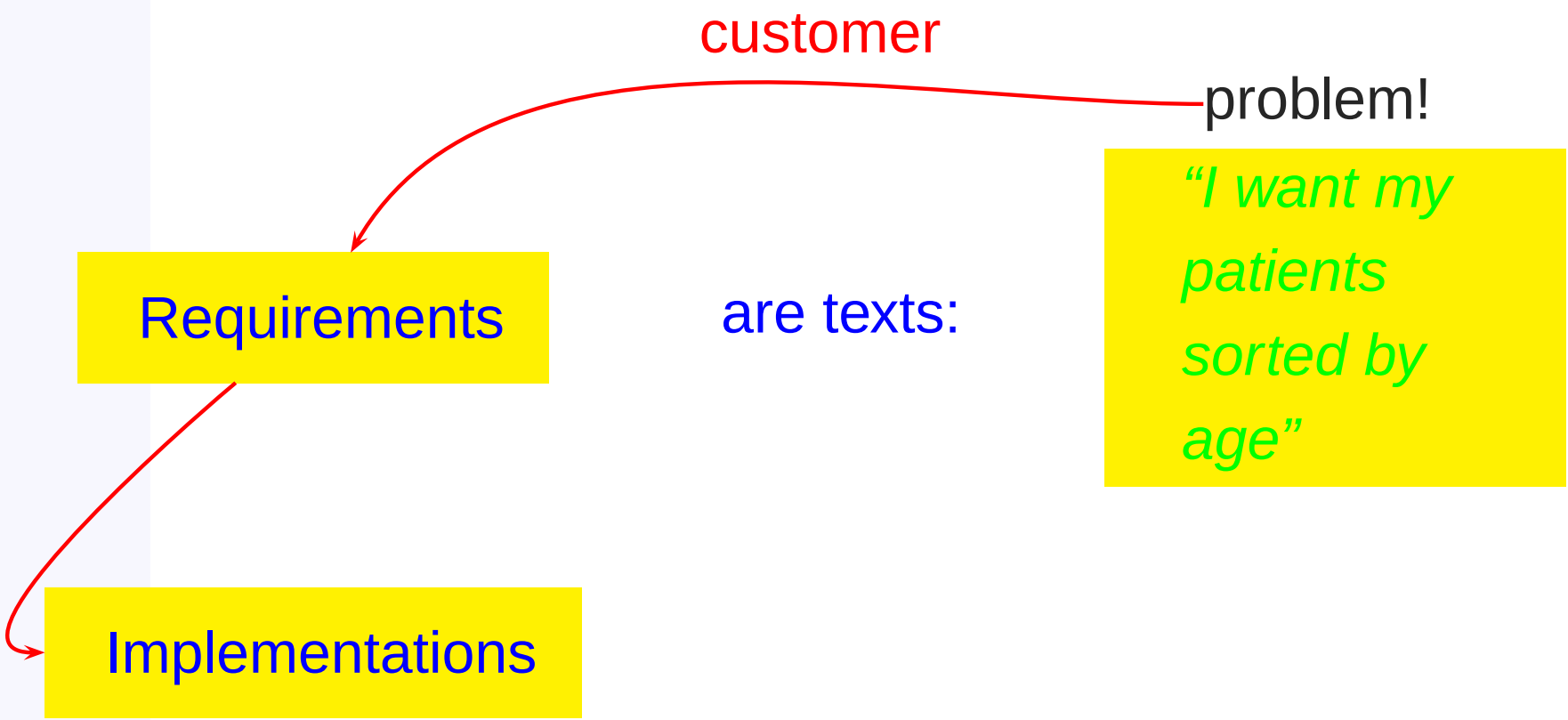
Notion of specification



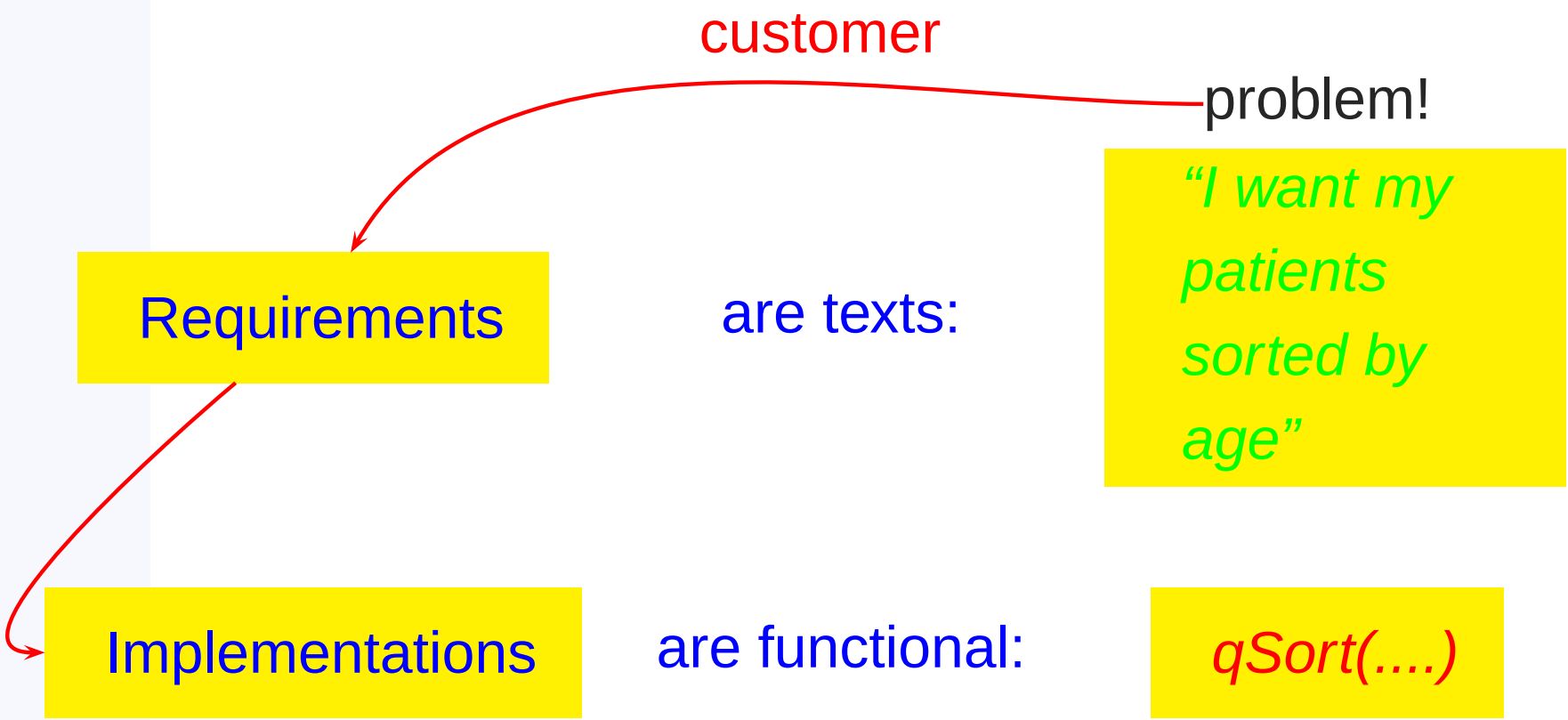
Notion of specification



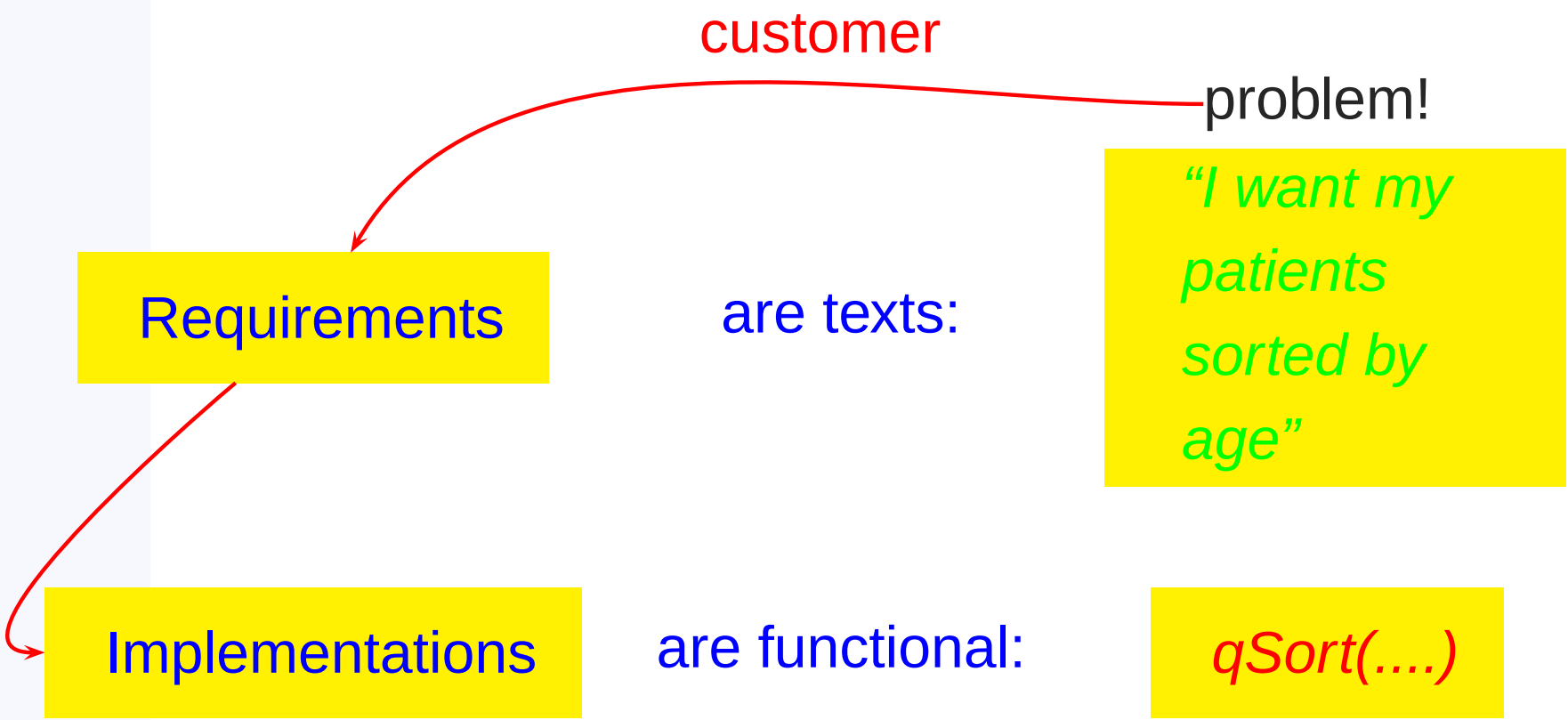
Notion of specification



Notion of specification

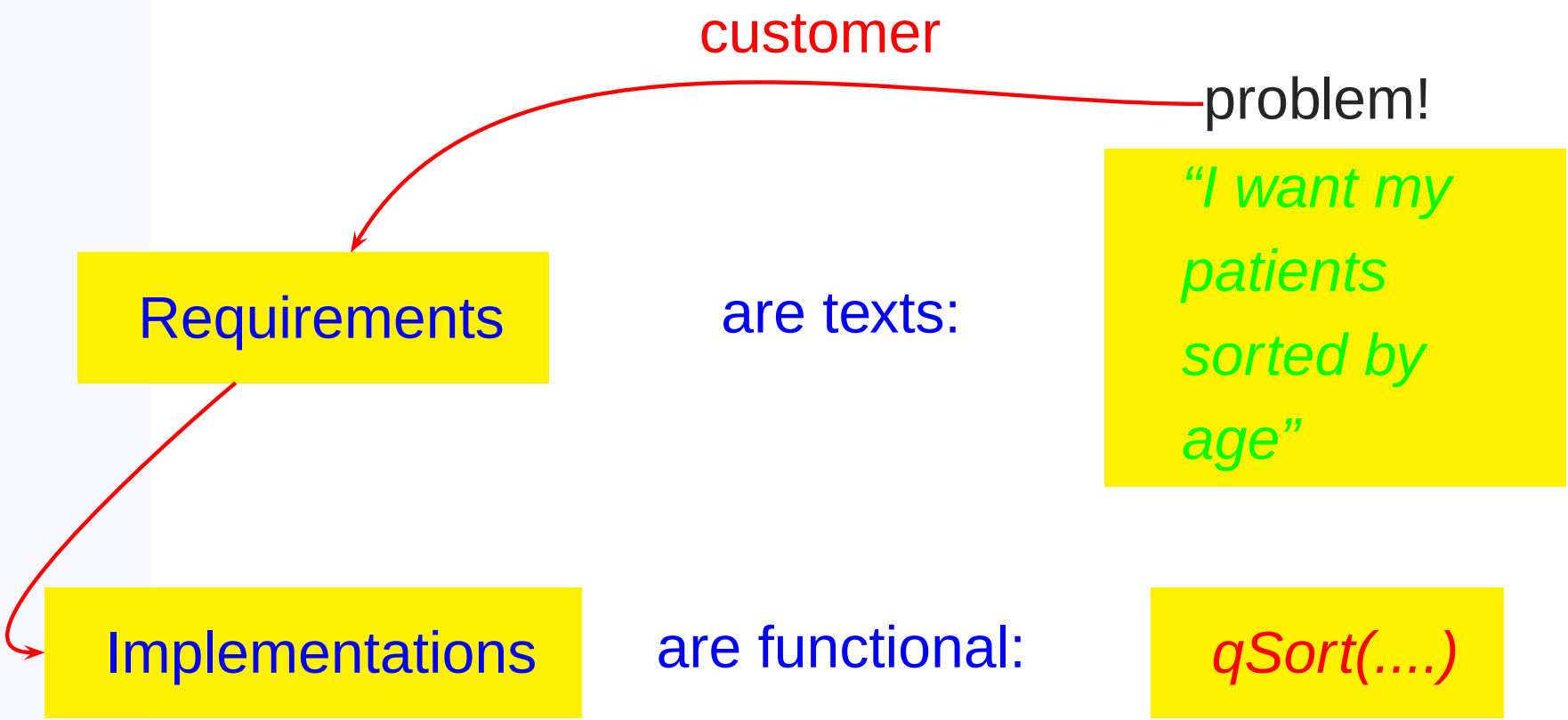


Notion of specification



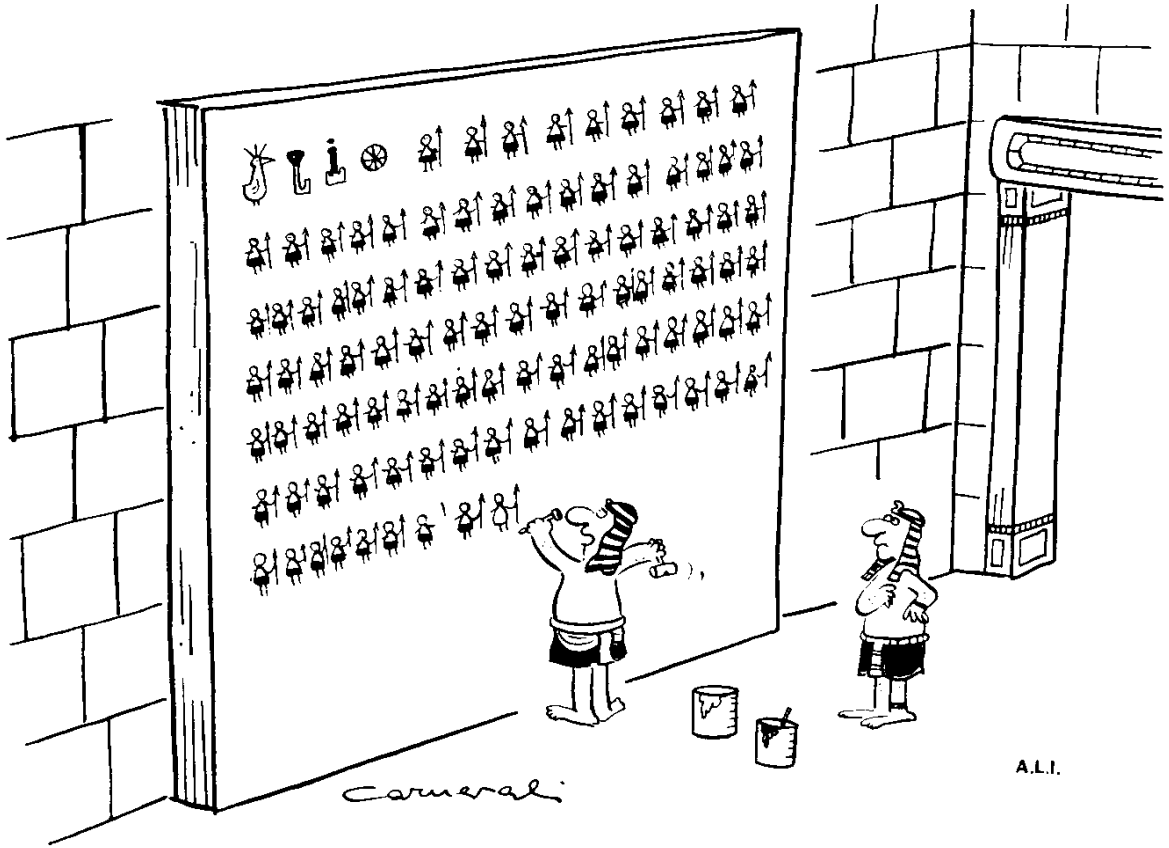
How do we accurately describe **sorting** without mentioning any sorting **algorithm?**

Notion of specification



How do we accurately describe **sorting** without mentioning any sorting **algorithm?** We need a **specification language**.

Why accurate (formal) notations?



Are you sure there isn't a simpler means of writing
'The Pharaoh had 10,000 soldiers?'

Trend for Notation Economy

- **Notation** — always a concern throughout the history of mathematics.
- In the 16th century,

$$12x^3 + 18x^2 + 27x + 17$$

would be written

12.cu.p̃.18.ce.p̃.27.co.p̃.17

— cf. **Libro de algebra** (1567) by Coimbra mathematician **Pedro Nunes** (1502-1578).

- Such notation was at its time replacing a even more obscure syntax.

Requirement analysis

From a mobile phone manufacturer:

> (...) For each list of calls stored in the mobile phone (eg. numbers dialed, SMS messages, lost calls), the store operation should work in a way such that (a) the more recently a call is made the more accessible it is; (b) no number appears twice in a list; (c) only the last 10 entries in each list are stored.

Requirement analysis

From a mobile phone manufacturer:

> (...) For each list of calls stored in the mobile phone (eg. numbers dialed, SMS messages, lost calls), the store operation should work in a way such that (a) the more recently a call is made the more accessible it is; (b) no number appears twice in a list; (c) only the last 10 entries in each list are stored.

Requirement analysis

From a mobile phone manufacturer:

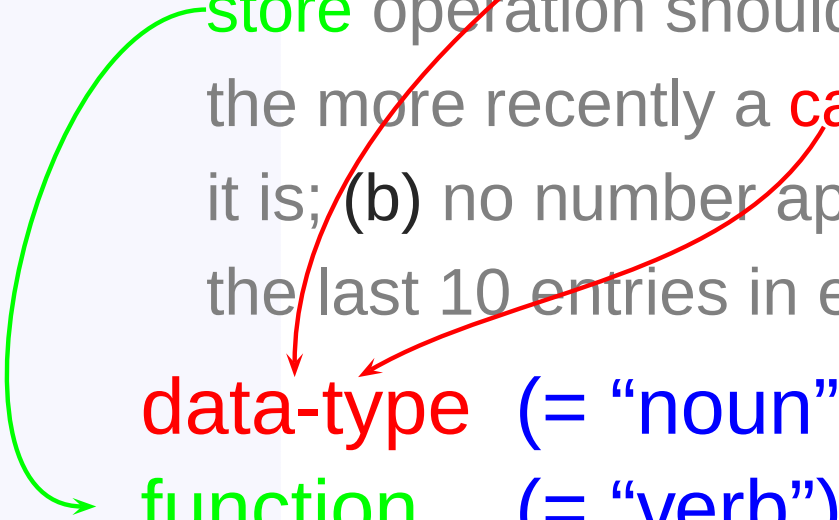
(...) For each list of calls stored in the mobile phone (eg. numbers dialed, SMS messages, lost calls), the store operation should work in a way such that (a) the more recently a call is made the more accessible it is; (b) no number appears twice in a list; (c) only the last 10 entries in each list are stored.

data-type (= “noun”);

Requirement analysis

From a mobile phone manufacturer:

(...) For each list of calls stored in the mobile phone (eg. numbers dialed, SMS messages, lost calls), the store operation should work in a way such that (a) the more recently a call is made the more accessible it is; (b) no number appears twice in a list; (c) only the last 10 entries in each list are stored.



data-type (= “noun”);

function (= “verb”);

Requirement analysis

From a mobile phone manufacturer:

> (...) For each list of calls stored in the mobile phone (eg. numbers dialed, SMS messages, lost calls), the store operation should work in a way such that (a) the more recently a call is made the more accessible it is; (b) no number appears twice in a list; (c) only the last 10 entries in each list are stored.

data-type (= “noun”);
function (= “verb”);
property (= “integrated sentence”);

Formal model

(...) For each list of calls stored in the mobile phone (eg. numbers dialed, SMS messages, lost calls), the store operation should work in a way such that (a) the more recently a call is dialed the more accessible it is; (b) no number appears twice in a list; (c) only the last 10 entries in each list are stored.

Formal model

(...) For each list of calls stored in the mobile phone (eg. numbers dialed, SMS messages, lost calls), the store operation should work in a way such that (a) the more recently a call is dialed the more accessible it is; (b) no number appears twice in a list; (c) only the last 10 entries in each list are stored.

Formal model

(...) For each list of calls stored in the mobile phone (eg. numbers dialed, SMS messages, lost calls), the store operation should work in a way such that (a) the more recently a call is dialed the more accessible it is; (b) no number appears twice in a list; (c) only the last 10 entries in each list are stored.

In **Haskell** notation:

```
store :: Call -> [Call] -> [Call]
store c l = ...
```

Formal model

(...) For each list of calls stored in the mobile phone (eg. numbers dialed, SMS messages, lost calls), the store operation should work in a way such that (a) the more recently a call is dialed the more accessible it is; (b) no number appears twice in a list; (c) only the last 10 entries in each list are stored.

In **VDM-SL** notation:

```
store : Call -> seq of Call -> seq of Call
store (c)(l) = ...
```

Meeting the requirements

(...) such that (a) the more recently a call is made the more accessible it is; (b) no number appears twice in a list; (c) only the last 10 entries in each list are stored.

Meeting the requirements

(...) such that (a) the more recently a call is made the more accessible it is; (b) no number appears twice in a list; (c) only the last 10 entries in each list are stored.

In Haskell:

```
store :: Call -> [Call] -> [Call]
store c l = [ c ] ++ l
```

Notation: `x ++ y` means "x catenated with y", eg.

```
[ c ] ++ [ a,b,c ] = [ c,a,b,c ]
```

Meeting the requirements

(...) such that (a) the more recently a call is made the more accessible it is; (b) no number appears twice in a list; (c) only the last 10 entries in each list are stored.

```
store :: Call -> [Call] -> [Call]
store c l = [ c ] ++ filter (/=c) l
```

Notation: From the **Haskell** Prelude:

```
filter ::  (a -> Bool) -> [a] -> [a]
filter p l = [ a | a <- l, p a ]
```

Meeting the requirements

(...) such that (a) the more recently a call is made the more accessible it is; (b) no number appears twice in a list; (c) only the last 10 entries in each list are stored.

```
store' :: Call -> [Call] -> [Call]
store' c l = take 10 (store c l)
```

Notation:

```
take 0 _ = [] take _ [] = []
take n (x:xs) | n>0 = x : take (n-1) xs
              | otherwise = []
```

Writing it in VDM-SL

```
store': Call -> seq of Call -> seq of Call
store'(c)(l)= take(10)(store(c)(l));
store : Call -> seq of Call -> seq of Call
store (c)(l) = [ c ] ^
                    [ a | a <- l & a <> c ];
```

Notation: `x^y` is the VDM-SL equivalent of `x ++ y`.

Notation `[ a | a <- l & a <> c ]` is not valid **VDM-SL**. One has to write

```
[ l(i) | i in set inds l & l(i) <> c ].
```

Common practice, in eg. C#

```
public void store10(string phoneNumber)
{
  System.Collections.ArrayList auxList =
      new System.Collections.ArrayList();
  auxList.Add(phoneNumber);
  auxList.AddRange(
      this.filteratmost9(phoneNumber) );
  this.callList = auxList;
}
```

C# version of store (cont.)

```
public System.Collections.ArrayList filteratmost9(string n)
{
  System.Collections.ArrayList retList =
      new System.Collections.ArrayList();
      int i=0, m=0;
  while((i < this.callList.Count) && (m < 9))
  {
      if ((string)this.callList[i] != n)
      {
          retList.Add(this.callList[i]);
          m++;
      }
      i++;
  }
  return retList;
}
```

Comments on C# code

Even tolerating code verbosity ...

- How “good” is this implementation?

Comments on C# code

Even tolerating code verbosity ...

- How "good" is this implementation?
- Does it meet the 3 properties stated by the mobile phone manufacturer?

Comments on C# code

Even tolerating code verbosity ...

- How "good" is this implementation?
- Does it meet the 3 properties stated by the mobile phone manufacturer?

Obs.:

- The same requirements in an **FM** exam paper led to 5 kinds of answer, of which only one (!) was correct!

Comments on C# code

Even tolerating code verbosity ...

- How "good" is this implementation?
- Does it meet the 3 properties stated by the mobile phone manufacturer?

Obs.:

- The same requirements in an **FM** exam paper led to 5 kinds of answer, of which only one (!) was correct!
- Alternatively, **FMs** provide for correct program construction, eg. by calculation.

Programming by calculation

```
   store'(c)(l)
=  take(10)(store(c)(l))
=  take(10)([c]^[l(i)|i in set inds l & l(i)<>c])
=  [c]^take(9)([c]^[l(i)|i in set inds l&l(i)<>c]
=  [c]^filteratmost(9)(...l...)
=  ...
```

Notation: calculation stems from formal properties, eg.

```
take(m)(x^y) = take(m)(x)^take(m-len x)(y)
```

FMs = true software engineering

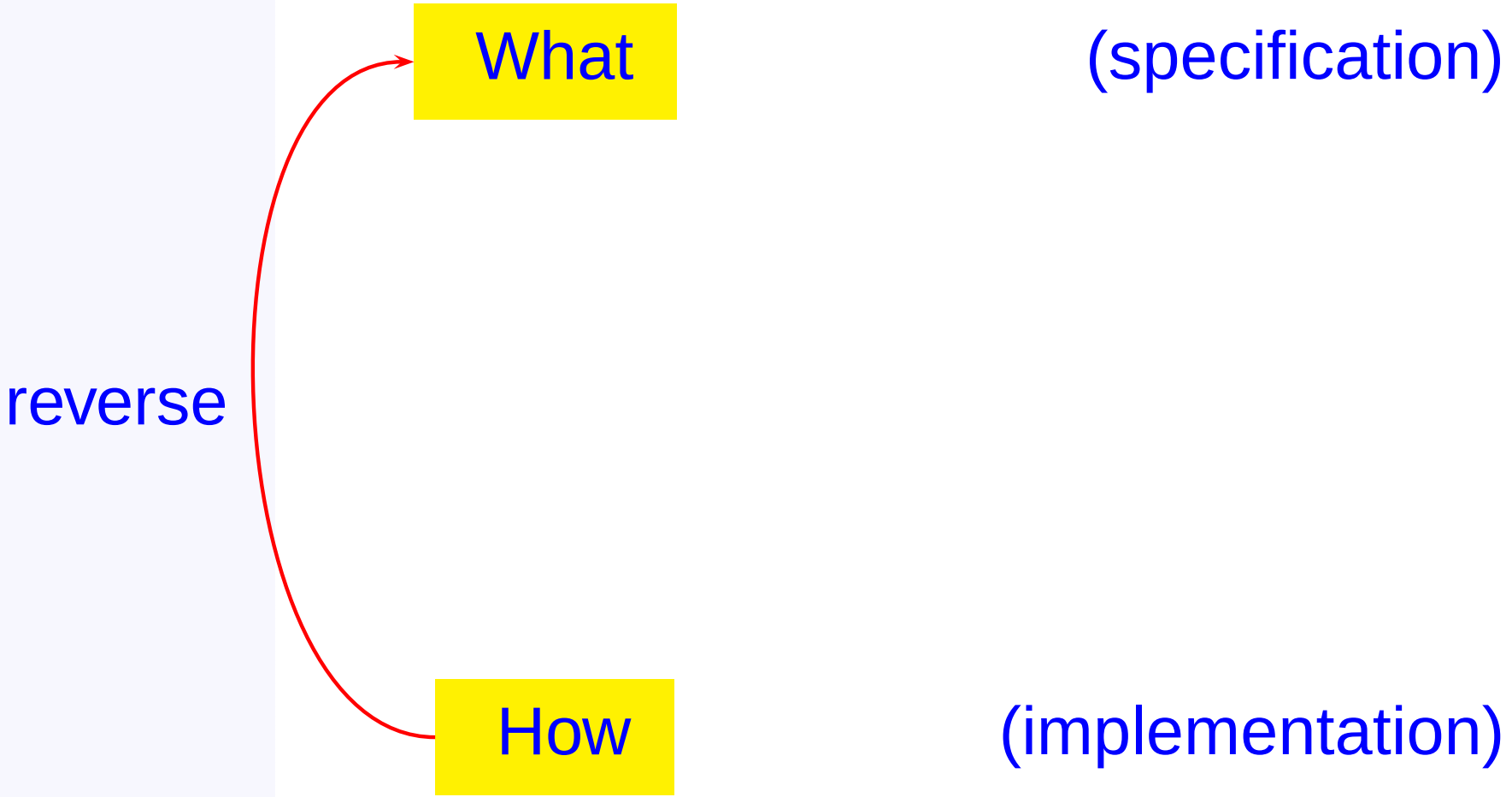
How (implementation)

FMs = true software engineering

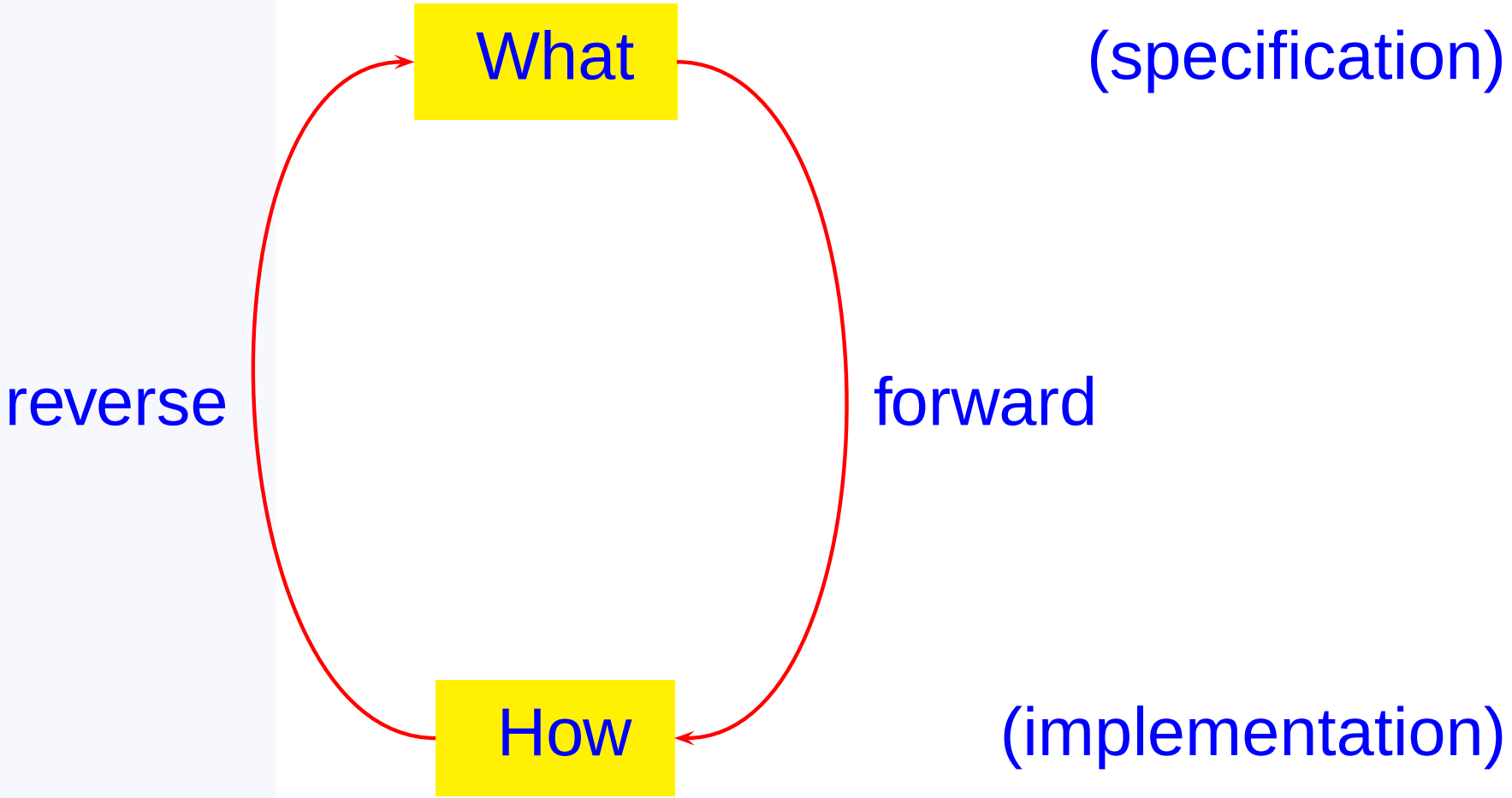
What (specification)

How (implementation)

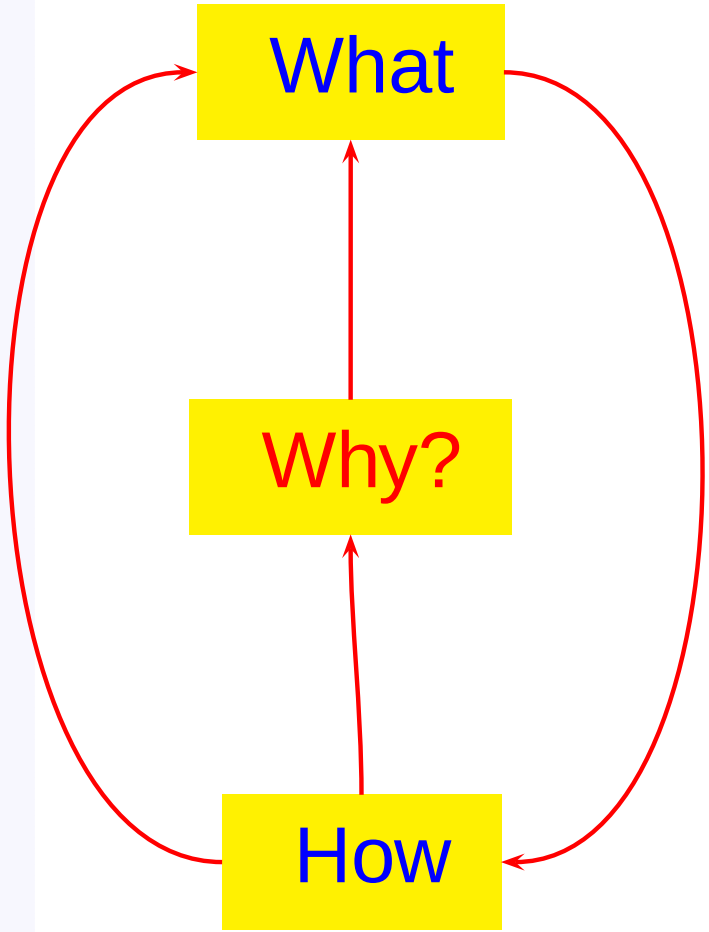
FMs = true software engineering



FMs = true software engineering



FMs = true software engineering



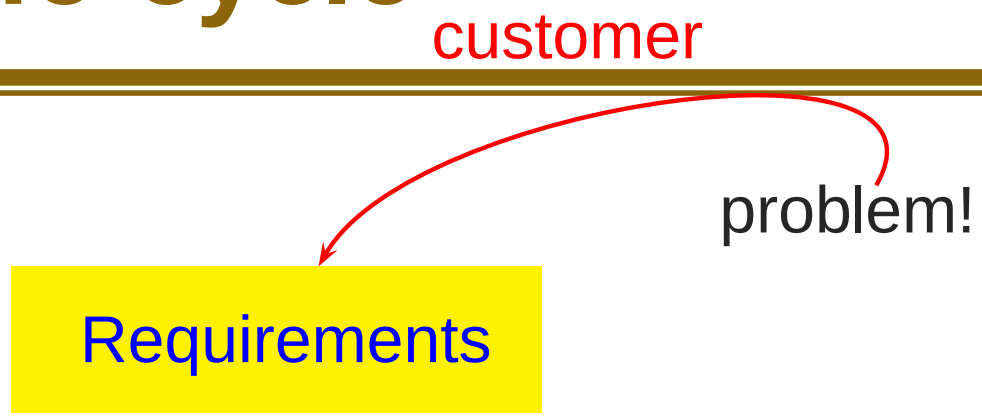
(specification)

reverse

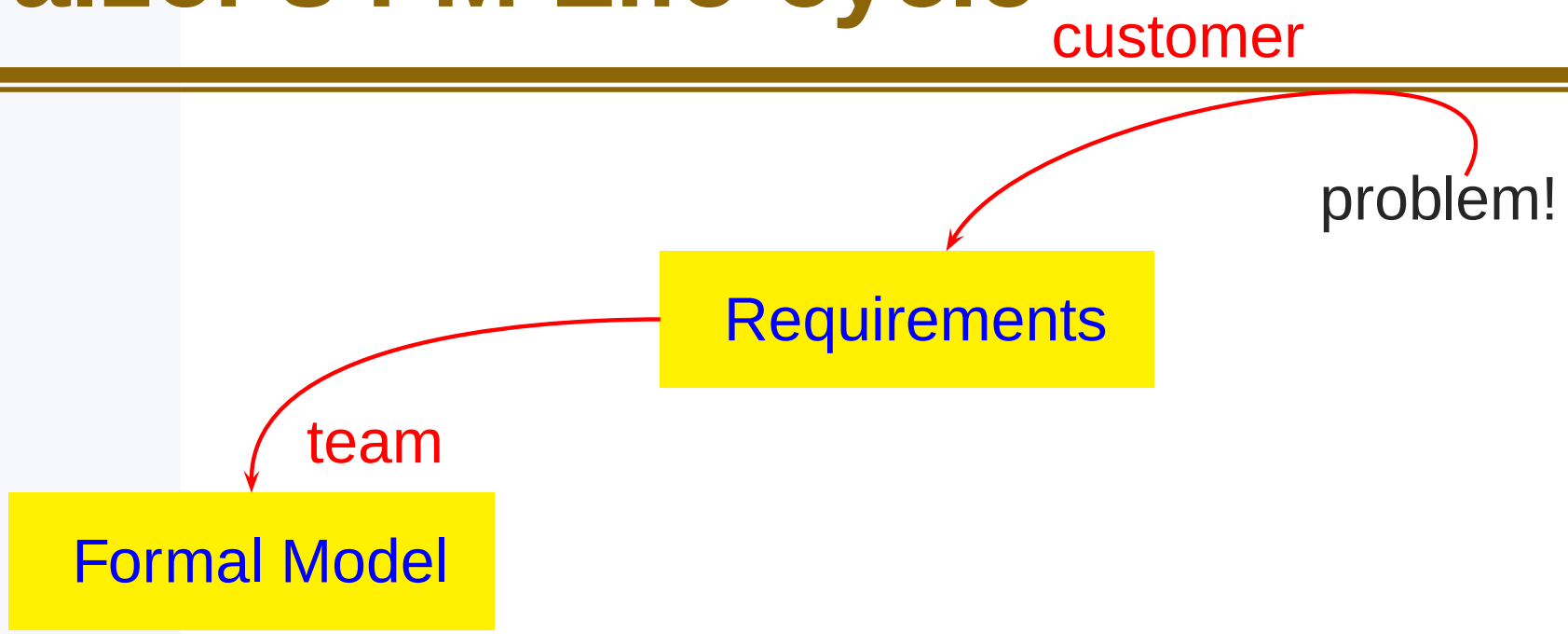
forward *(justification)*

(implementation)

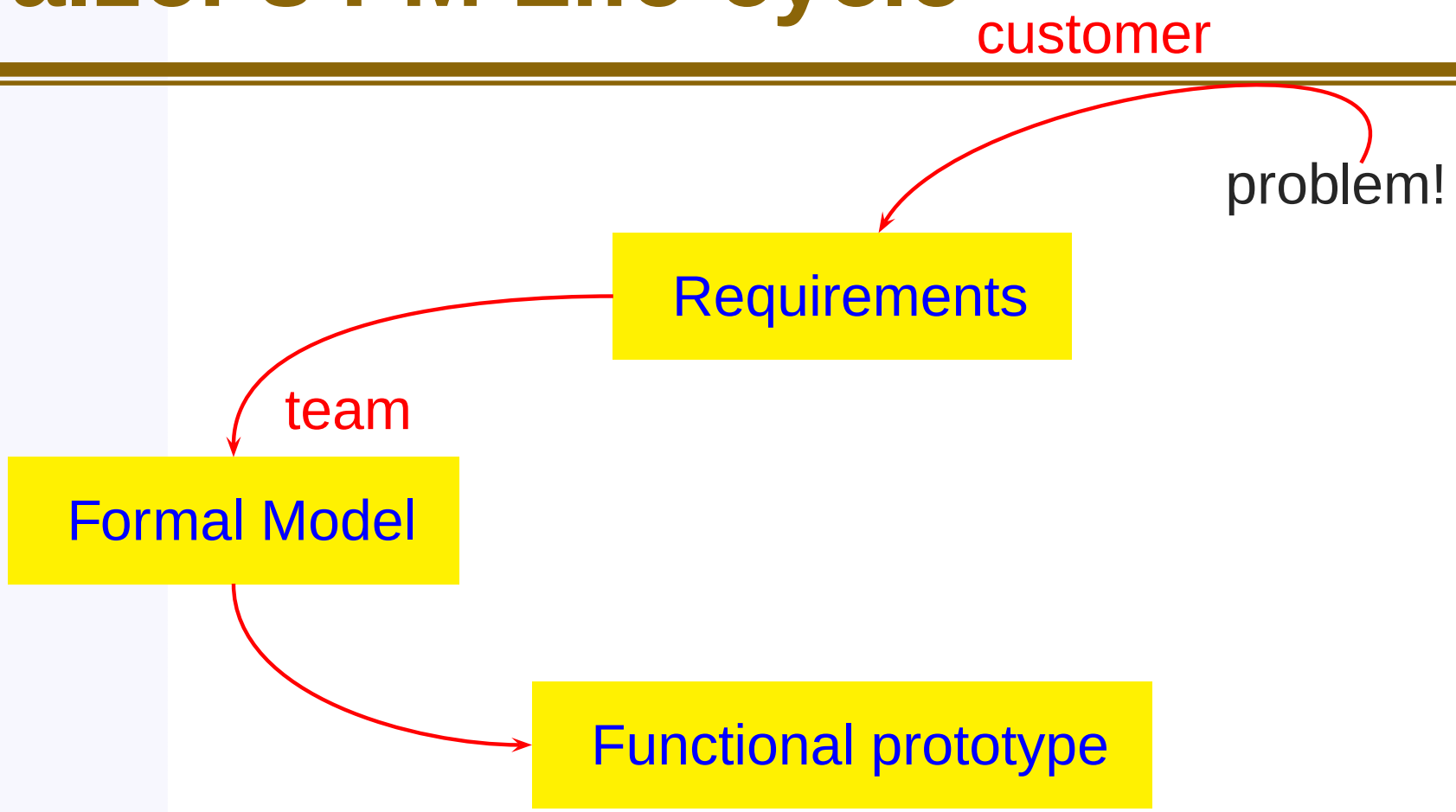
Balzer's FM Life-cycle



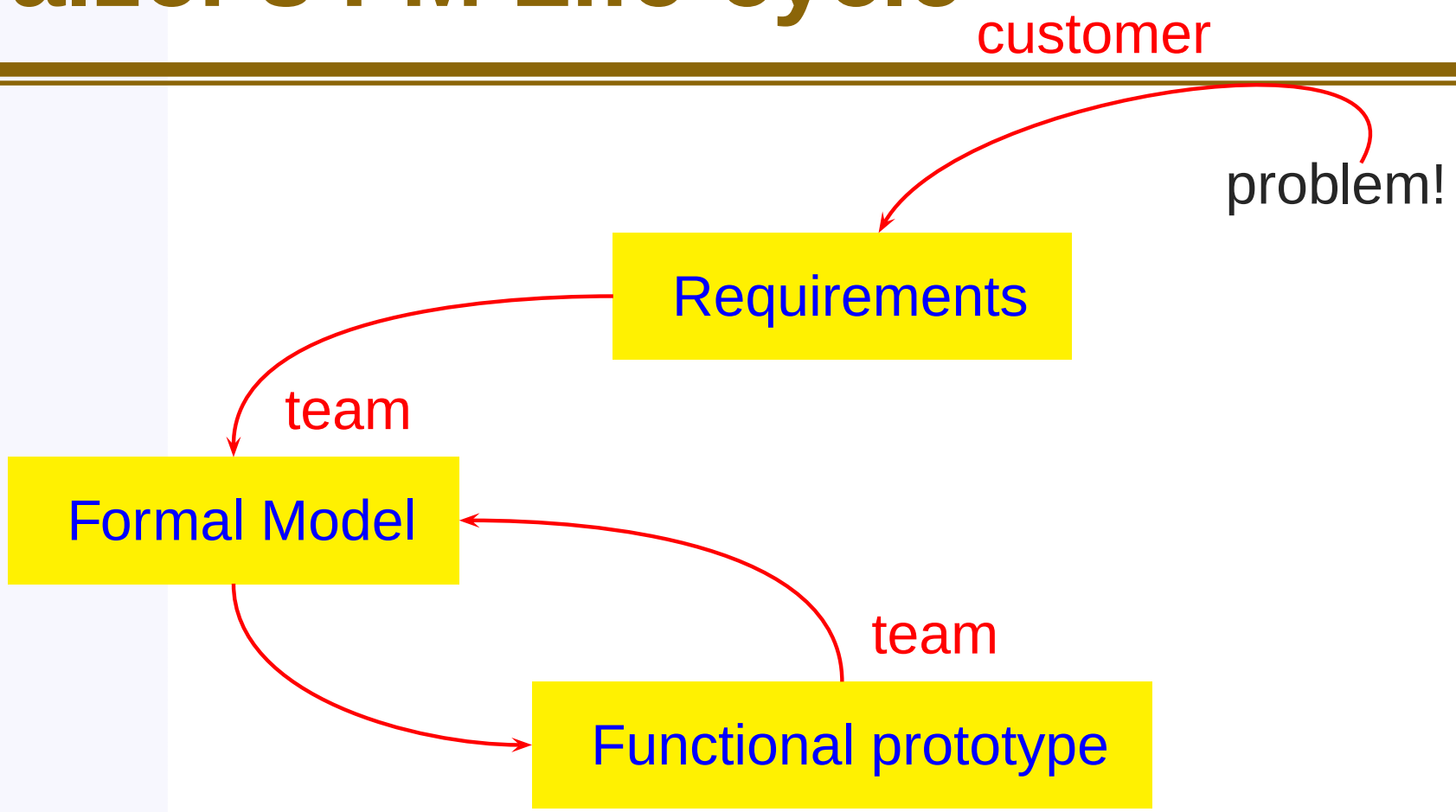
Balzer's FM Life-cycle



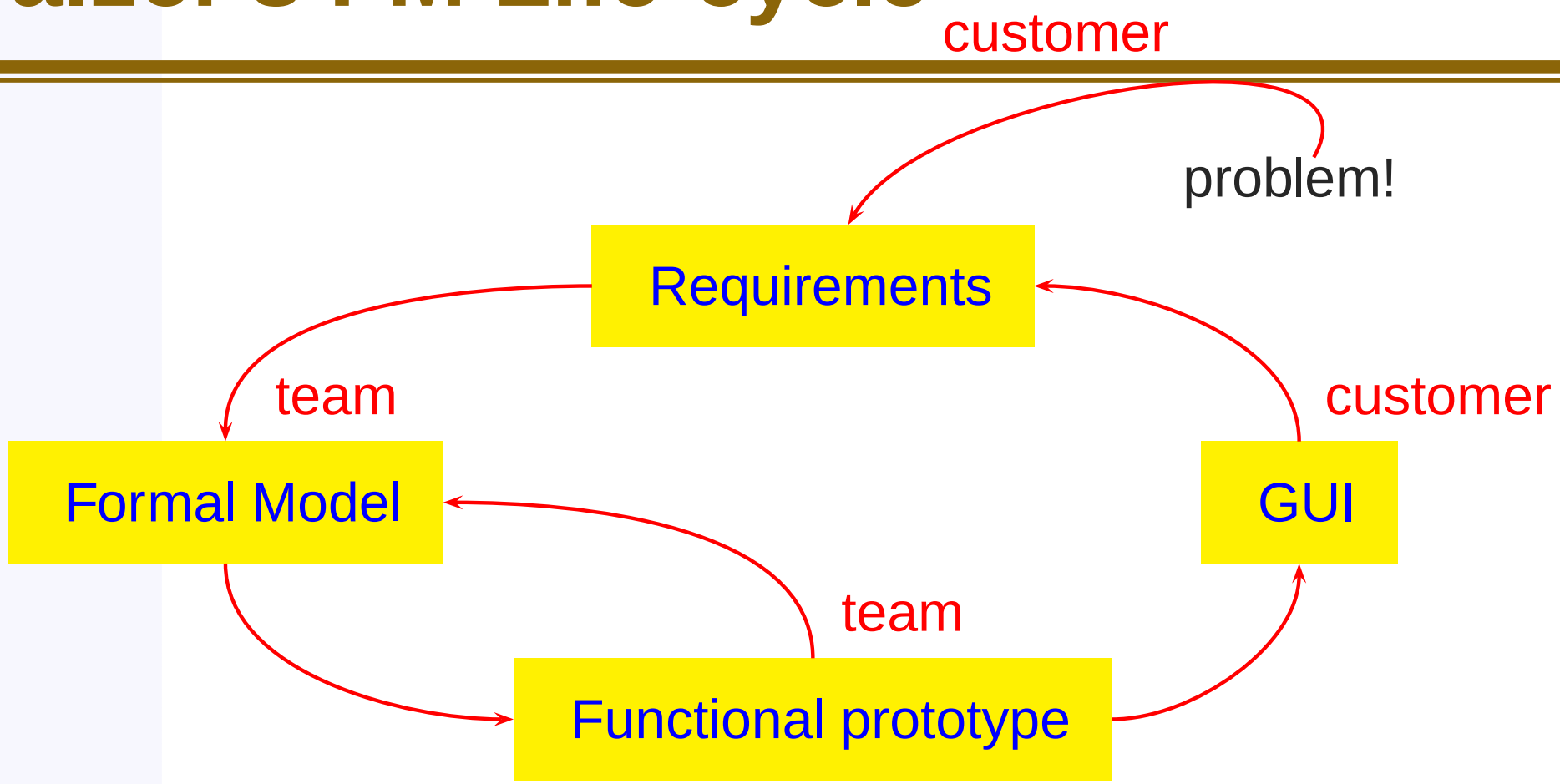
Balzer's FM Life-cycle



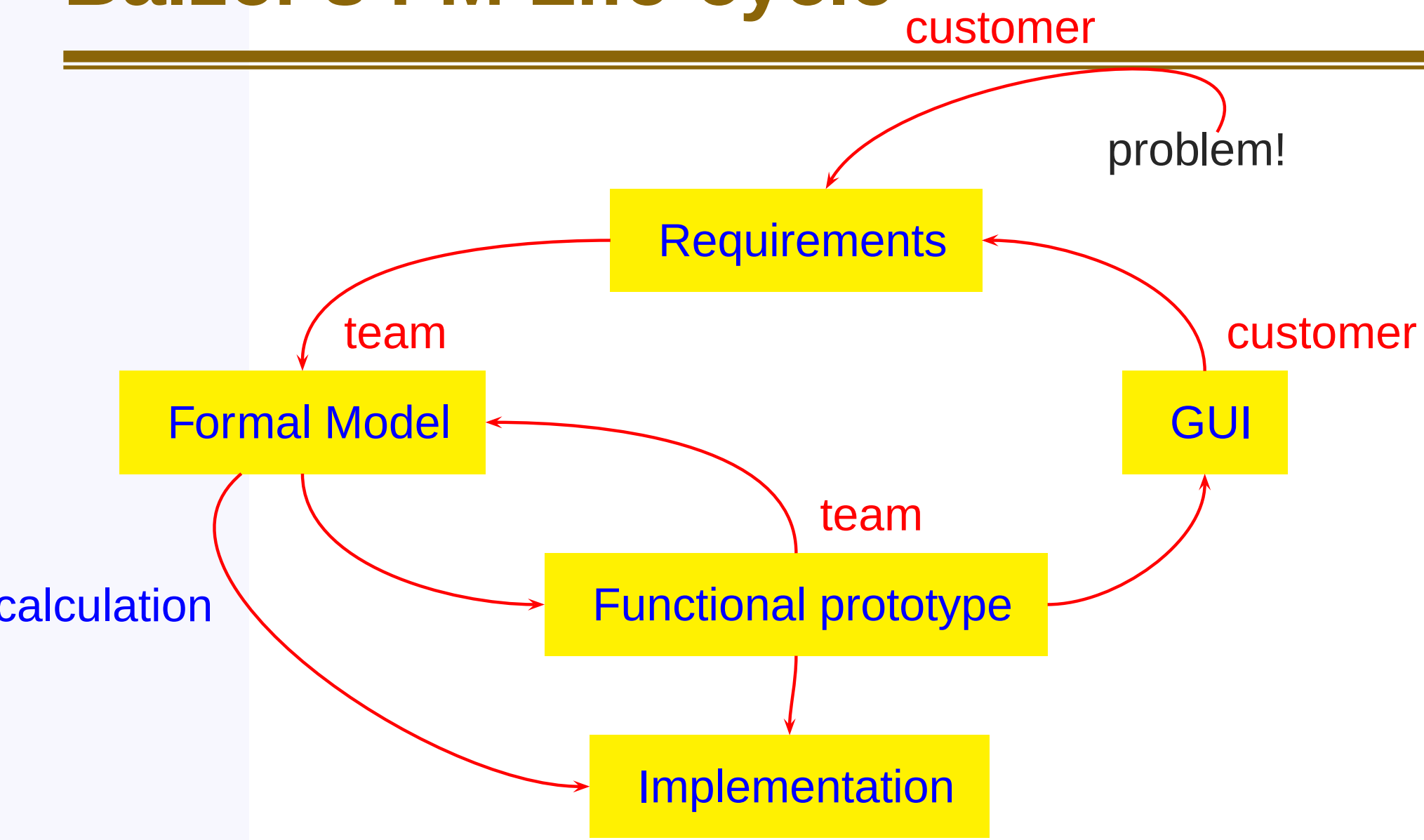
Balzer's FM Life-cycle



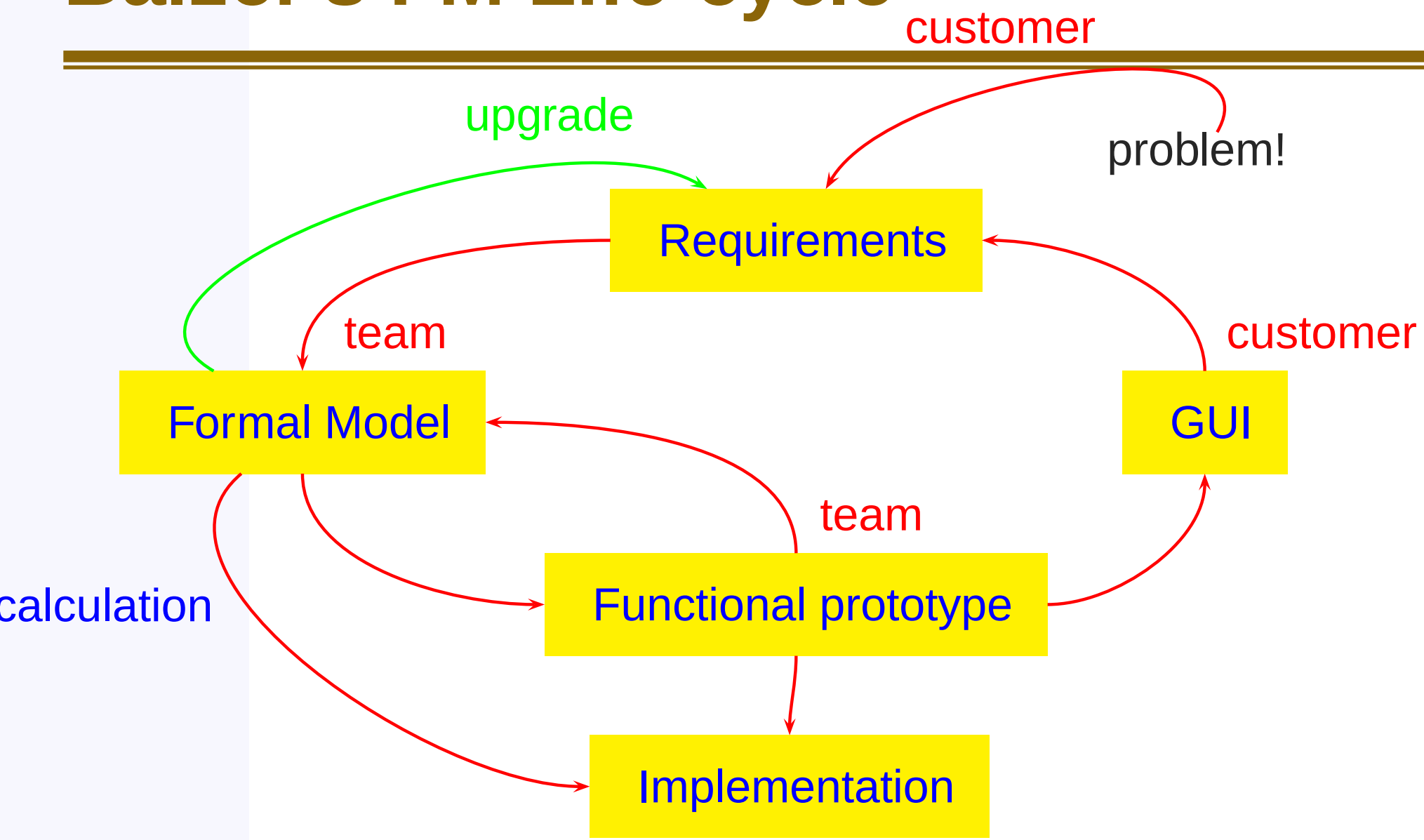
Balzer's FM Life-cycle



Balzer's FM Life-cycle



Balzer's FM Life-cycle



Our background

- By 2004:

  20 years of **FM teaching** at the Univ. of Minho

- $\simeq$ 10 years ago:

  Industrial application of FMs based on FP tested at INESC-BRAGA

- Spin-off of INESC-BRAGA (1996):

  SIDEREUS S.A. - Rigorous Solutions for Software Systems (Porto)

FMs add to competitiveness

Increased productivity:

Code Validation
- Debug
- Verification
- **Calculation** / automation

FMs add to competitiveness

Increased productivity:

Code Validation { Debug; Verification; **Calculation** / automation

Technology-independent documentation:

- the actual enterprise's **wealth**
- **investment** safeguard.

FMs add to competitiveness

Increased productivity:

Code Validation {
- Debug
- Verification
- **Calculation** / automation

Technology-independent documentation:

- the actual enterprise's **wealth**
- **investment** safeguard.

trains **competitive** software designers :-)