Towards Formal Software Development in VS.NET

MSDN - 14/16th of May, 2002

J.N. Oliveira

SIDEREUS S.A. and DI/UNIV. OF MINHO

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 Peyton 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)

(...)

...ur Anderson accounting firm is
...ored to have made overtures to Peyton
...nes. (...) But Professor Peyton Jones
...lans to remain where he is.

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

(CAMBRIDGE EVENING NEWS,

Cover Story (conclusion)

(...)

rthur Anderson accountin
nored to have made overtures to
ones. (...) But Professor Peyton Jor
lans to remain where he is.

"I'm flattered that my research h
lly 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"

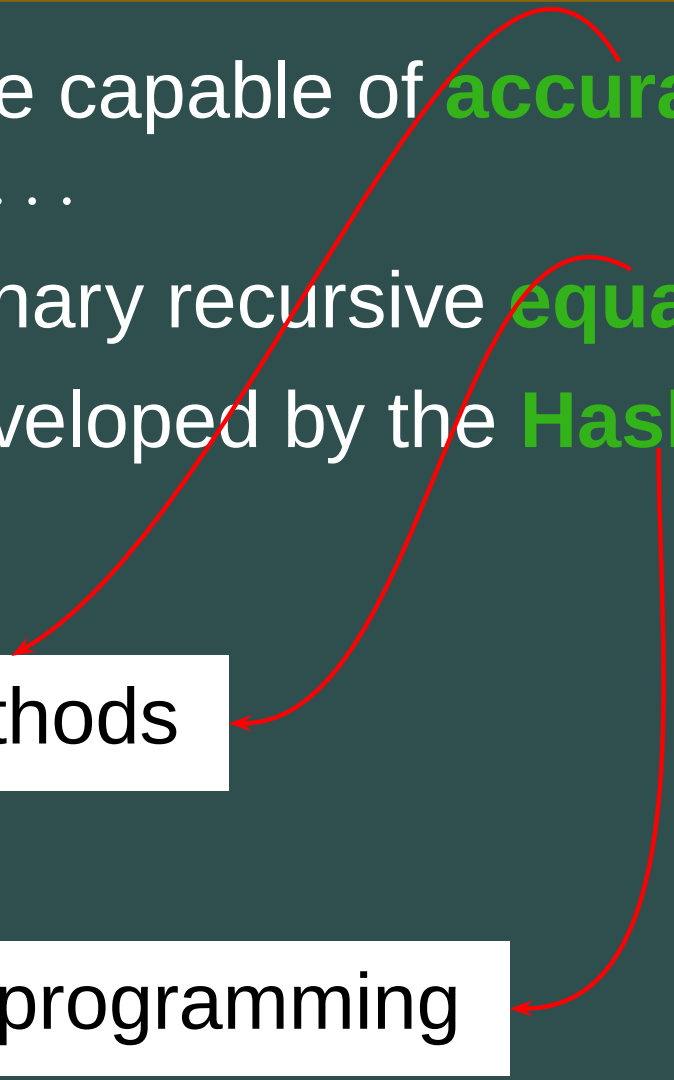
- . . . language capable of **accurately** describing and valuing . . .
- . . . just ordinary recursive **equations** . . .
- . . . tools developed by the **Haskell** community

In other words:

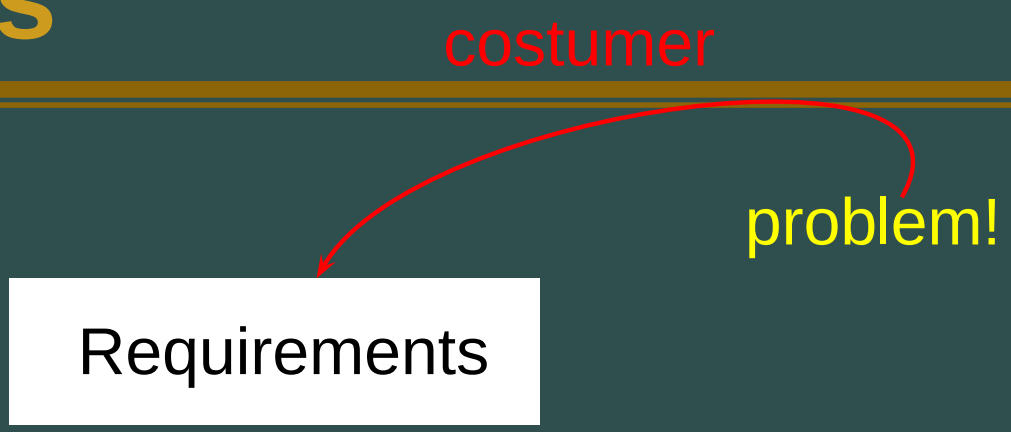
- formal methods

and

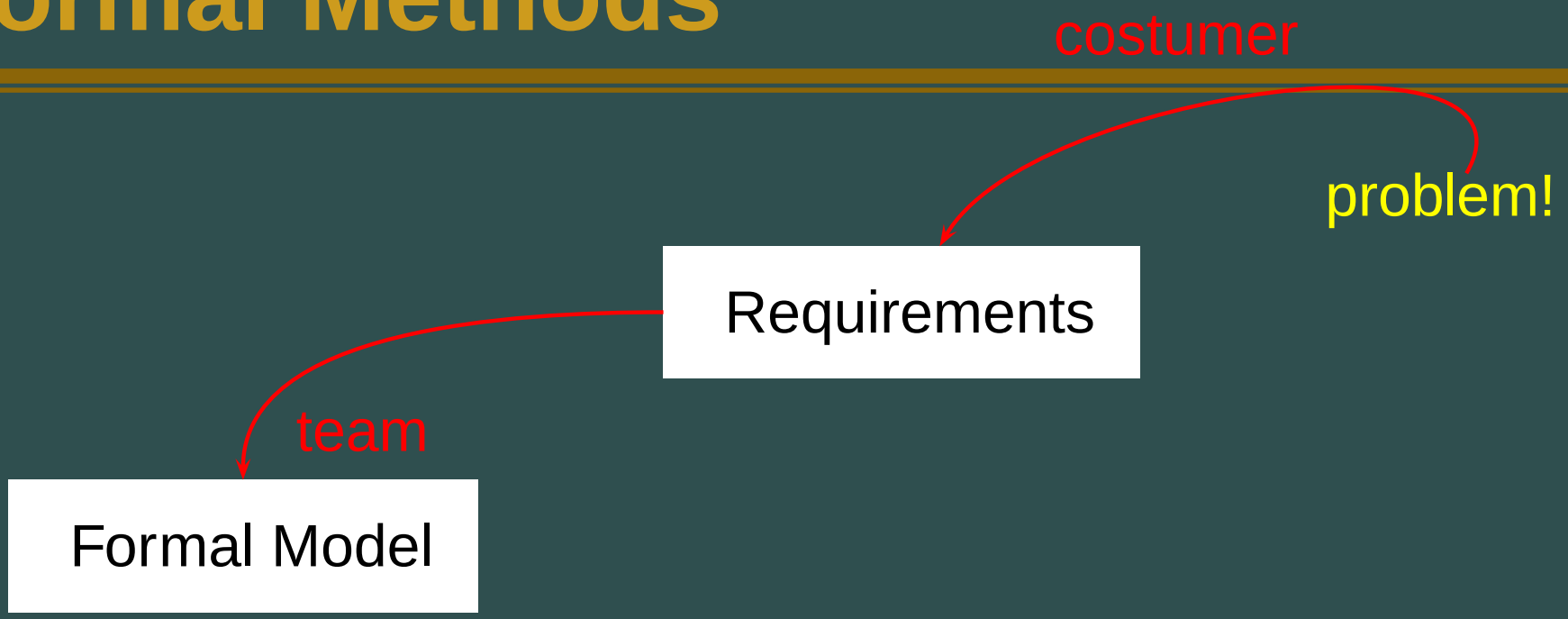
- functional programming



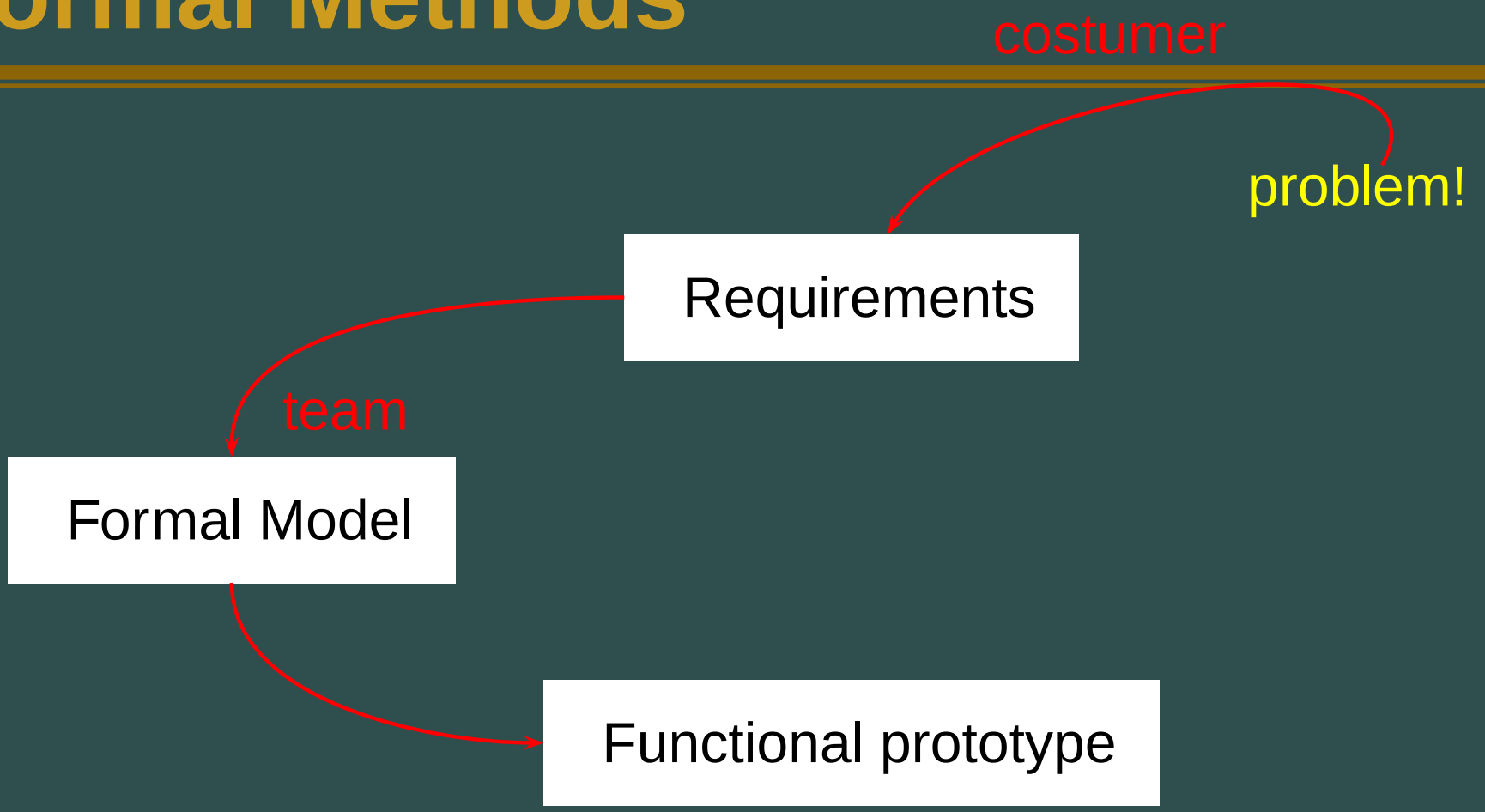
Formal Methods



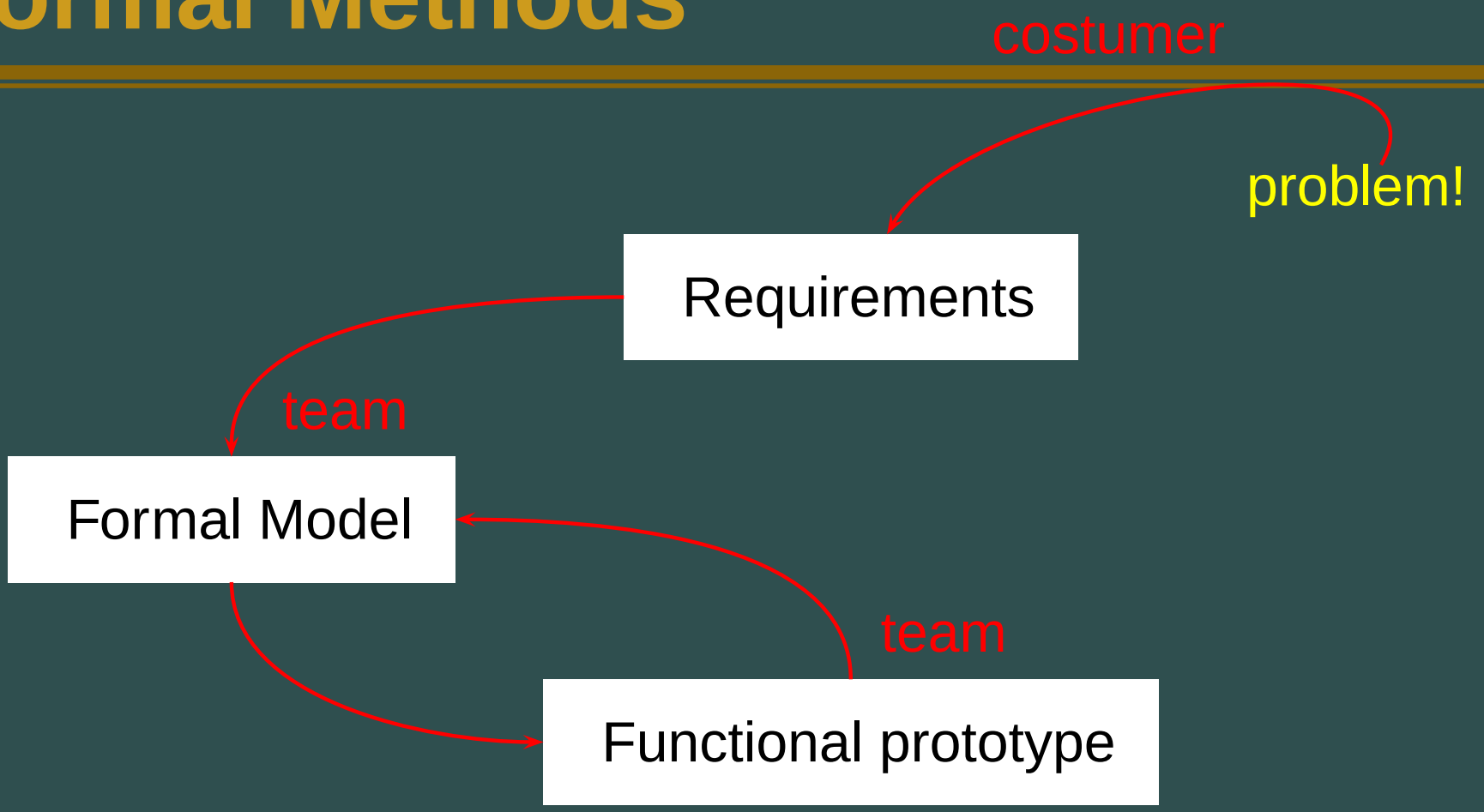
Formal Methods



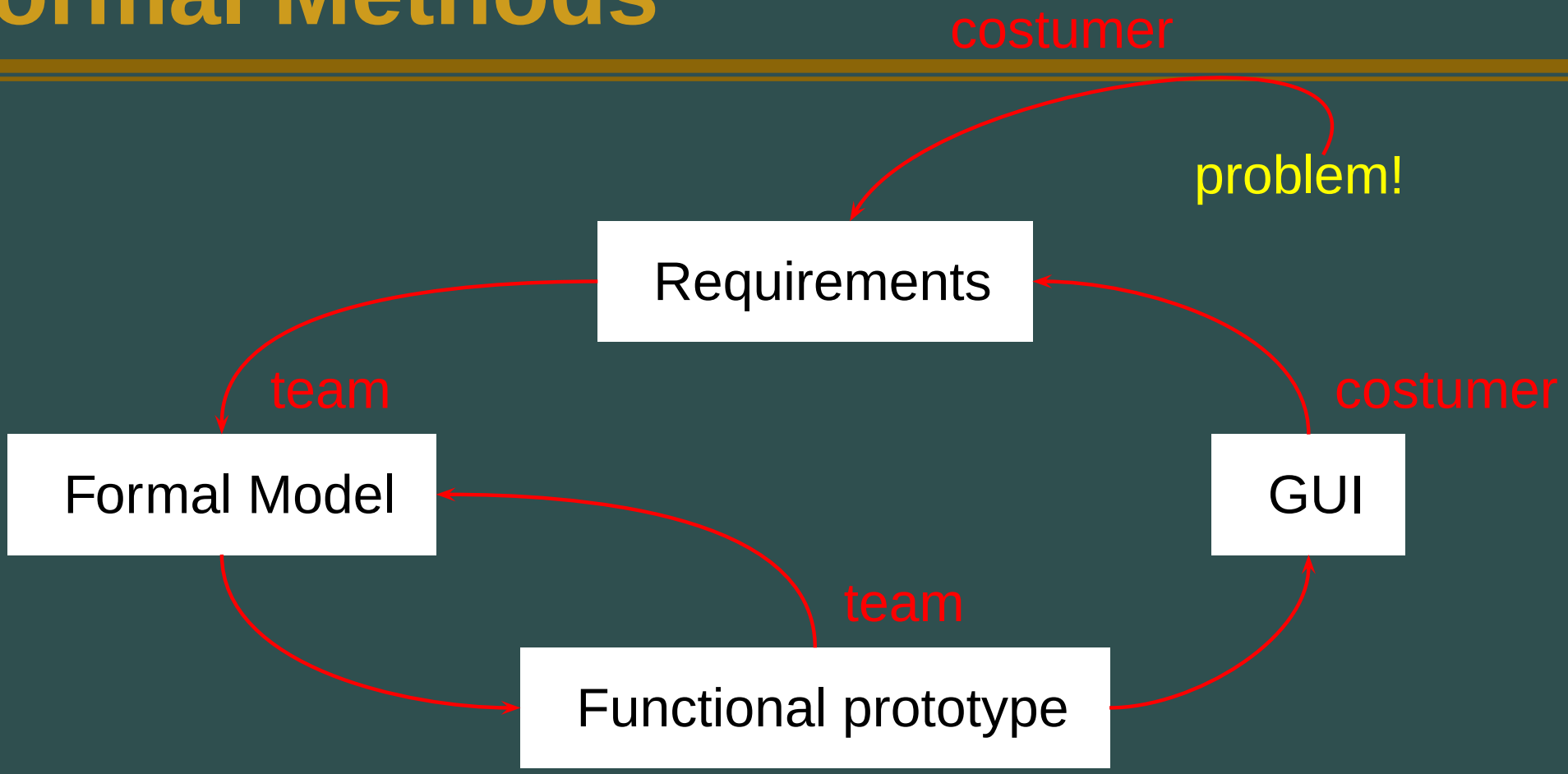
Formal Methods



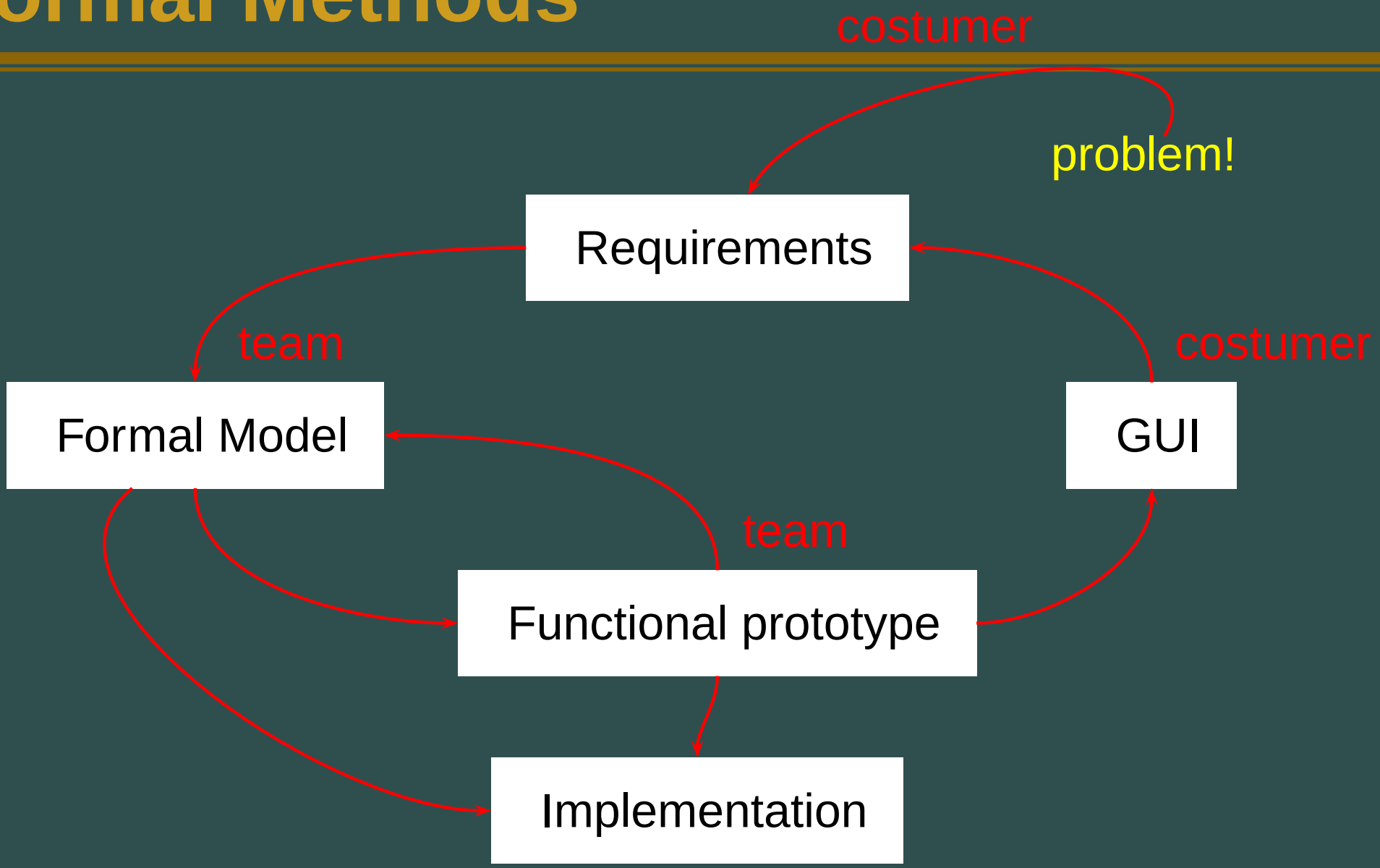
Formal Methods



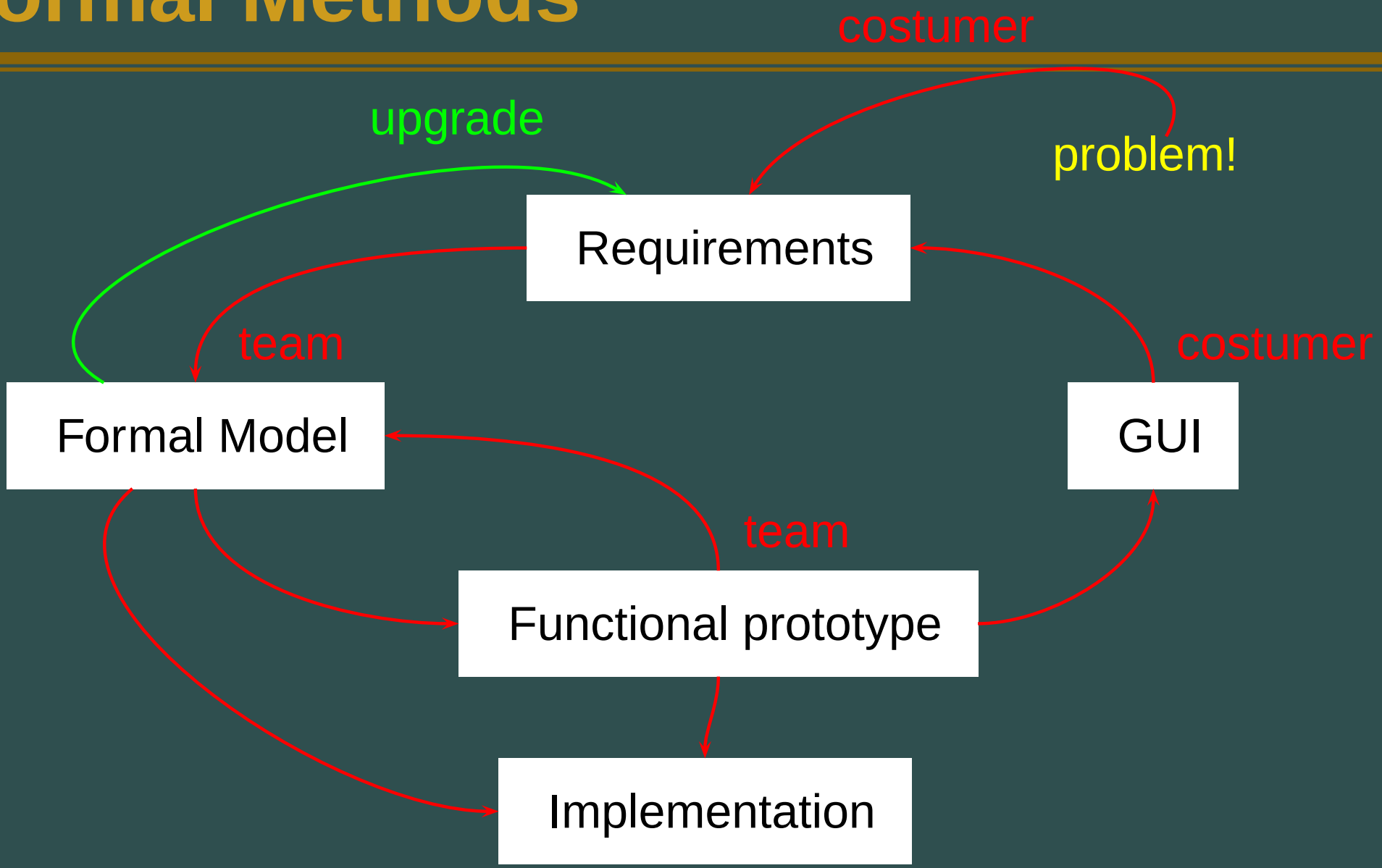
Formal Methods



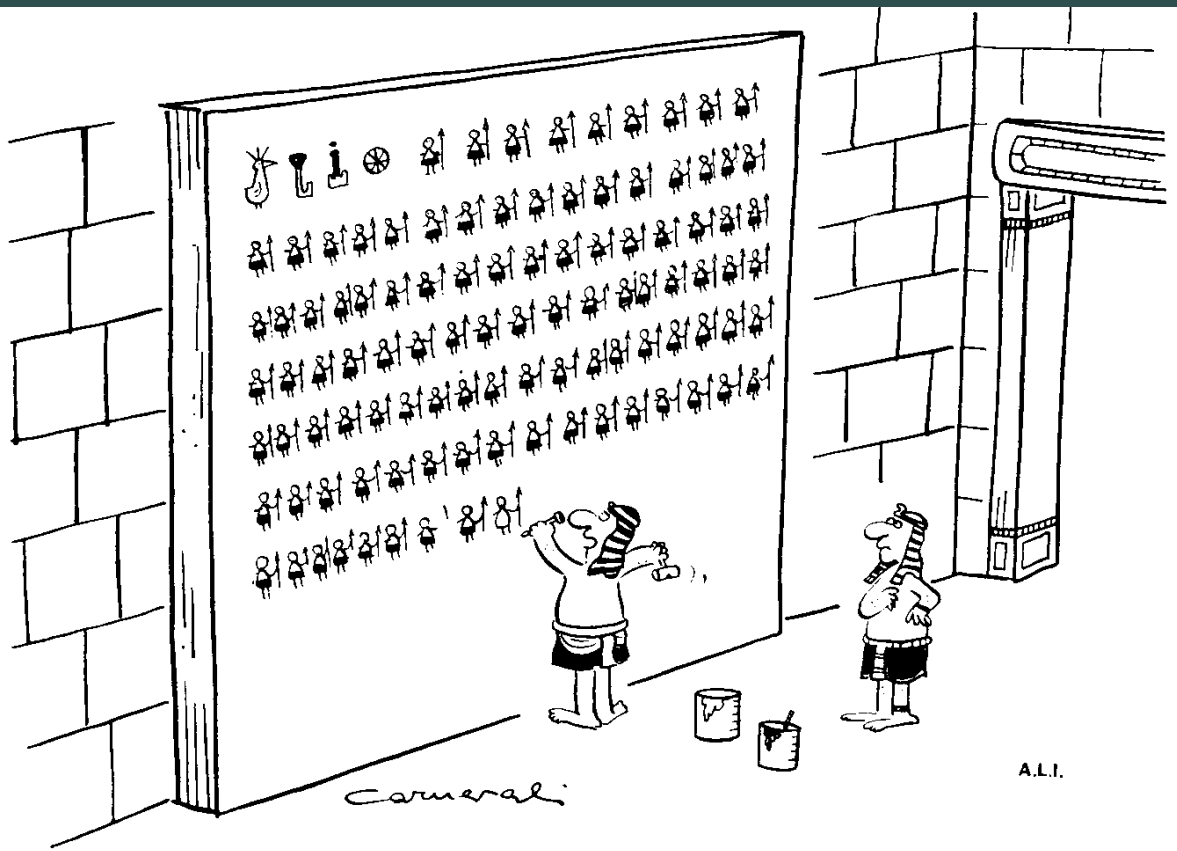
Formal Methods



Formal Methods



Why formal / elegant notations?



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

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:

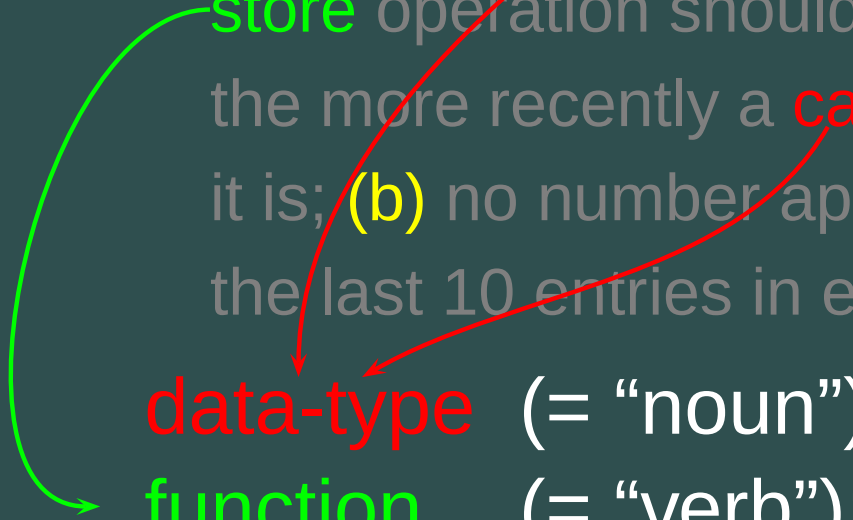
(...) For each list of calls stored in the mobile phone (eg. numbers dialed, SMS messages, lost calls), the store operation should work in a way such that (a) the more recently a call is made the more accessible it is; (b) no number appears twice in a list; (c) only the last 10 entries in each list are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

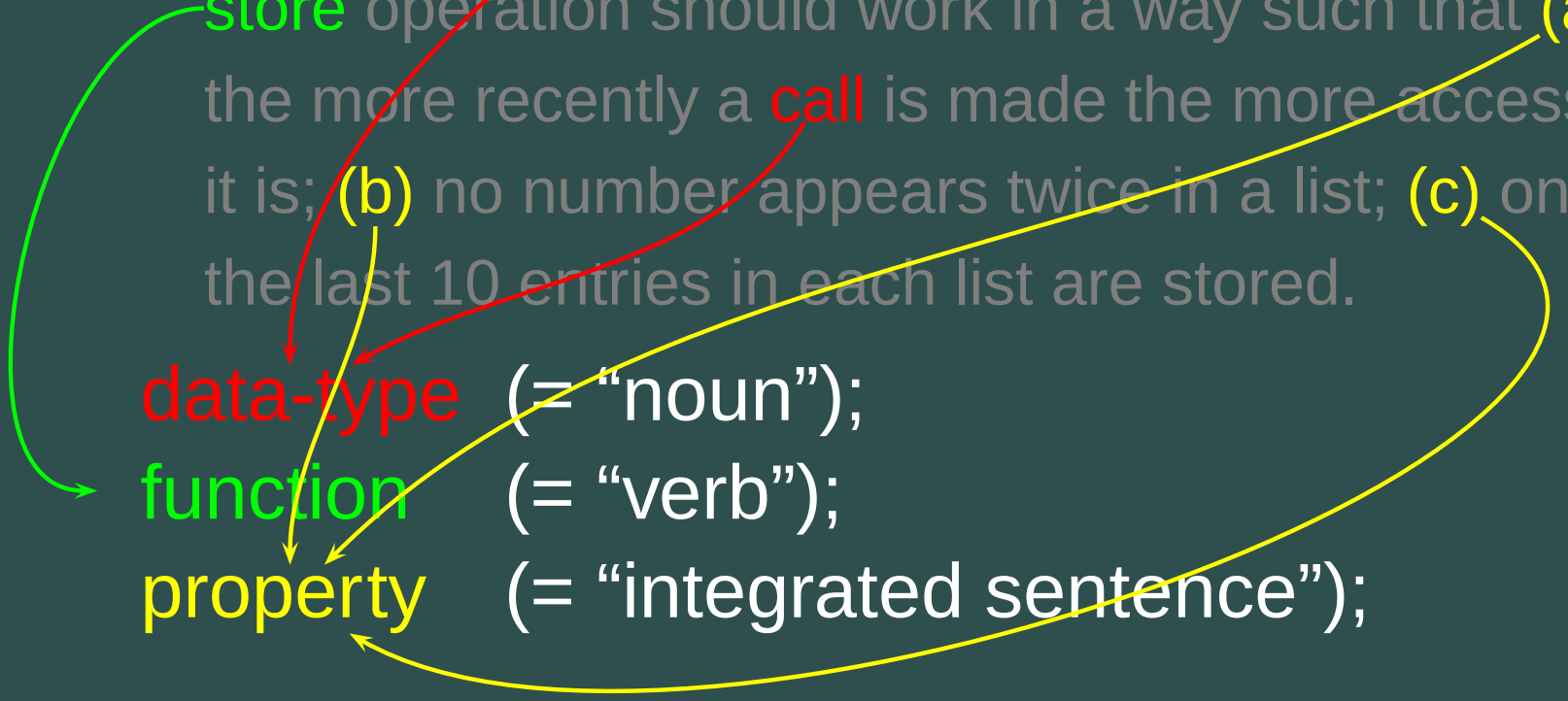


data-type (= “noun”);
function (= “verb”);

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.



Formal model

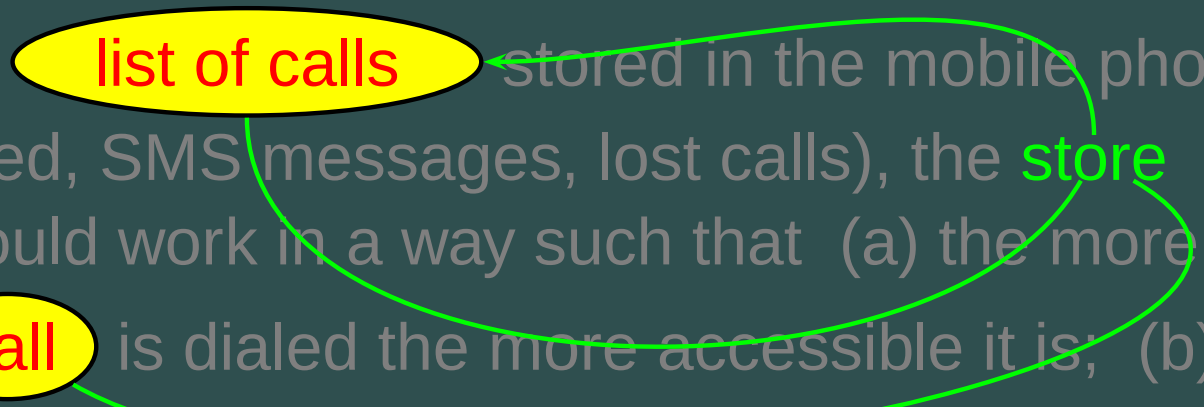
(...) For each list of calls stored in the mobile phone (eg. numbers dialed, SMS messages, lost calls), the store operation should work in a way such that (a) the more recently a call is dialed the more accessible it is; (b) no number appears twice in a list; (c) only the last 10 entries in each list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

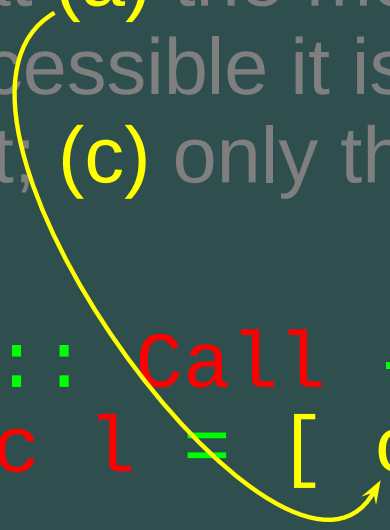
```
store :: Call -> [Call] -> [Call]
store c l = ...
```

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.



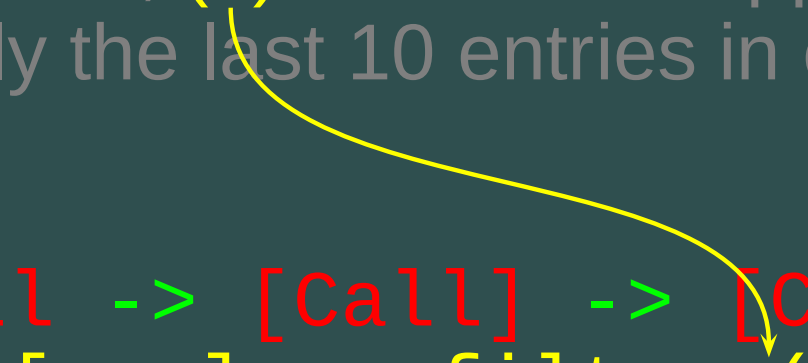
```
store :: Call -> [Call] -> [Call]
store c l = [ c ] ++ l
```

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 = []
```

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 ] ++ filter (/=c) l)
=  [ c ] ++ take 9 filter (/=c) l
=  [ c ] ++ filteratmost 9 (/=c) l
=  ...
```

Notation: calculation stems from formal properties, eg.

```
take m (x ++ y) = (take m x) ++ (take (m-length x) y)
```

FMs = true software engineering

How

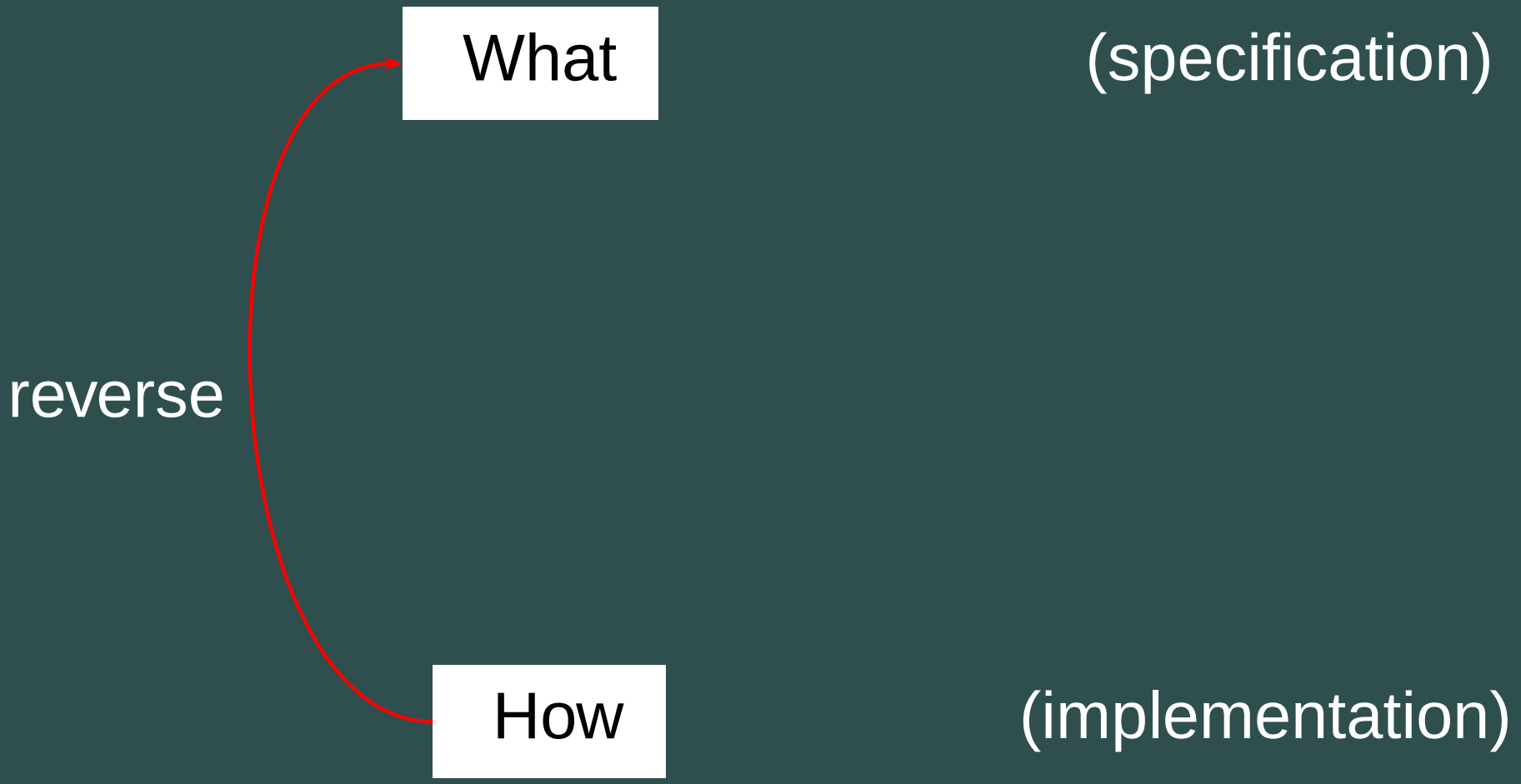
(implementation)

FMs = true software engineering

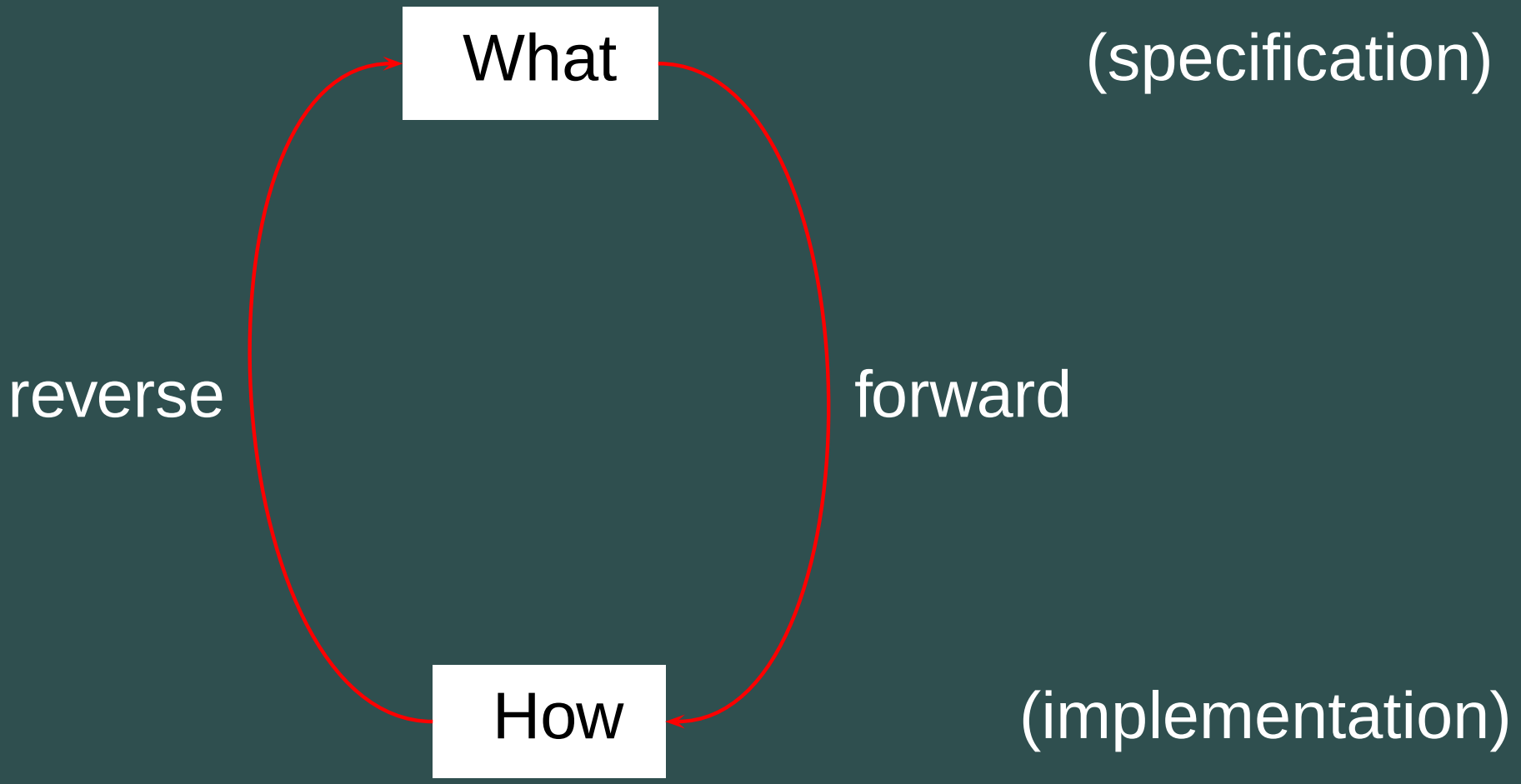
What (specification)

How (implementation)

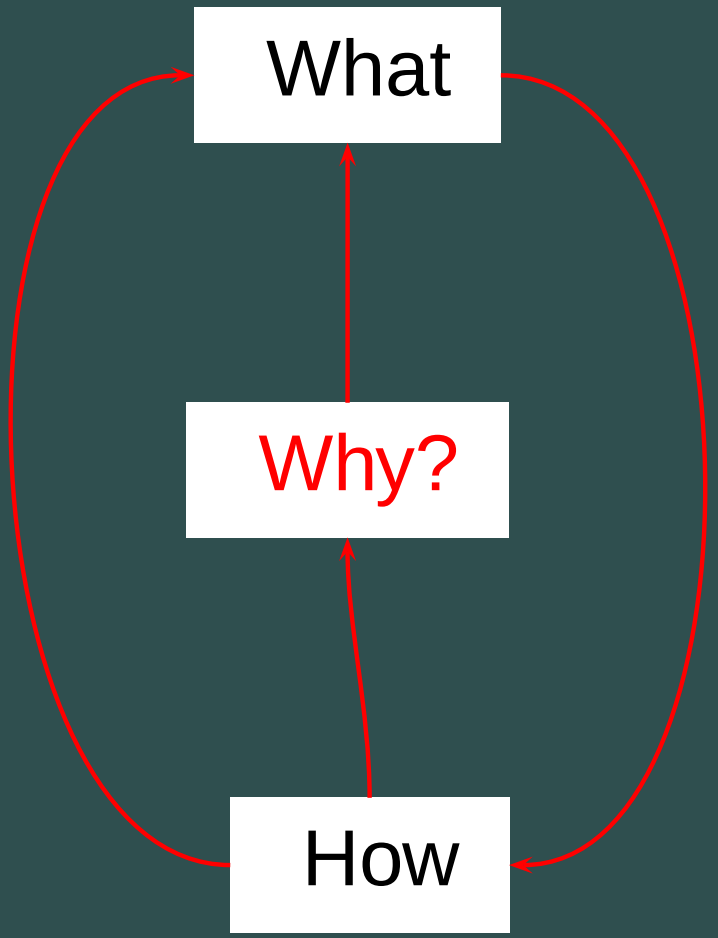
FMs = true software engineering



FMs = true software engineering



FMs = true software engineering



(specification)

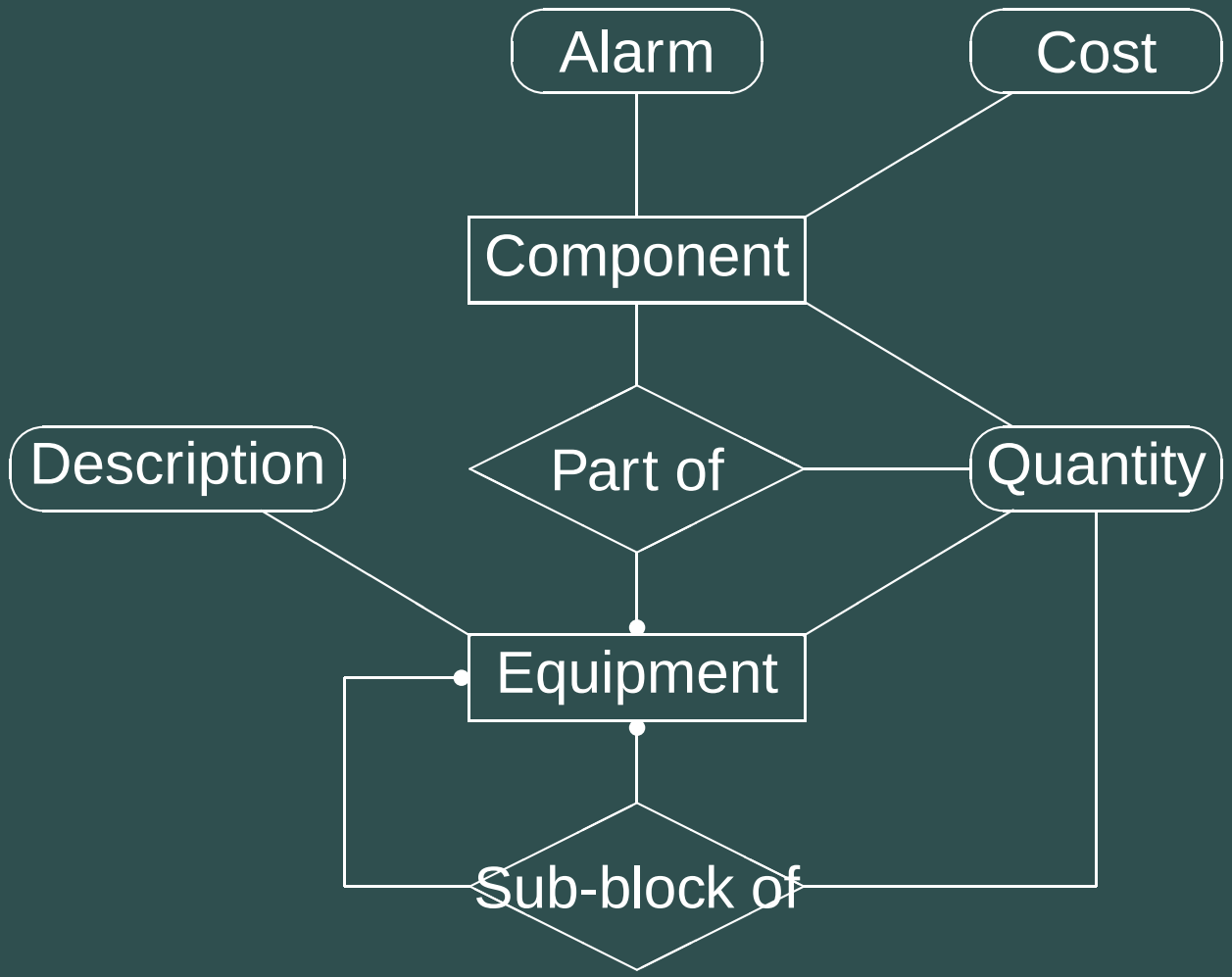
reverse

forward *(justification)*

(implementation)

Scalling up

Bill of materials (ER):



Bill of materials (SQL)

```
CREATE TABLE COMPONENTS (
       CompId CHAR   (8)   NOT NULL,
       CStock NUMBER (10)  NOT NULL,
       Alarm NUMBER  (10)  NOT NULL,
       Cost NUMBER   (3,3) NOT NULL
       CONSTRAINT COMPONENTS_pk
                  PRIMARY KEY(CompId)
        );

CREATE TABLE EQUIPMENTS (
       EqId CHAR (8) NOT NULL,
       Description CHAR (73) NOT NULL,
       EStock NUMBER (10) NOT NULL,
       CONSTRAINT EQUIPMENTS_pk PRIMARY KEY (EqId)
       );
```

(...)

Bill of materials (Haskell)

The entities:

```
data Component = Comp_Record {
        compID :: String,
        alarm  :: Int,
        cost   :: Float,
        cstock  :: Int
        }

data Equipment = Eq_Record {
        eqID          :: String,
        description   :: String,
        estock        :: Int
        }
```

Bill of materials (Haskell)

The relationships:

```
data Part_of    = Part_of_Record {
        comp    :: String,
        equip   :: String,
        howManyC :: Int
        }
data Sub_Block_of = Sub_Block_of_Record {
        equipL   :: String,
        equipS   :: String,
        howManyE :: Int
        }
```

The relational tables themselves:

```
type Components = Set Component
type Equipments = Set Equipment
type Parts_of   = Set Part_of
type Sub_Blocks_of = Set Sub_Block_of
```

Data processing is functional

- Every function

$$f : B \to A$$

  is a kind of "data miner": it extracts the A-view of every piece of B-data.

In fact:

Data processing is functional

- Every function

$$f : B \to A$$

  is a kind of "data miner": it extracts the A-view of every piece of B-data.

In fact:

- **data-mining** is functional

Data processing is functional

- Every function

$$f : B \to A$$

  is a kind of "data miner": it extracts the A-view of every piece of B-data.

In fact:

- **data-mining** is functional
- **document processing** (eg. XML) is functional

Haskell at work

Not only ...

- **HaXML** — library for XML processing
- **HaskellDB** — communication with the data access layer

Haskell at work

Not only ...

- **HaXML** — library for XML processing
- **HaskellDB** — communication with the data access layer

but also ...

- **HaskellScript** — library for the integration of Haskell with COM/ActiveX.
- **HSP** — a compositional alternative to ASP/PHP
- **HDirect** — IDL compiler helping to interact with external code.

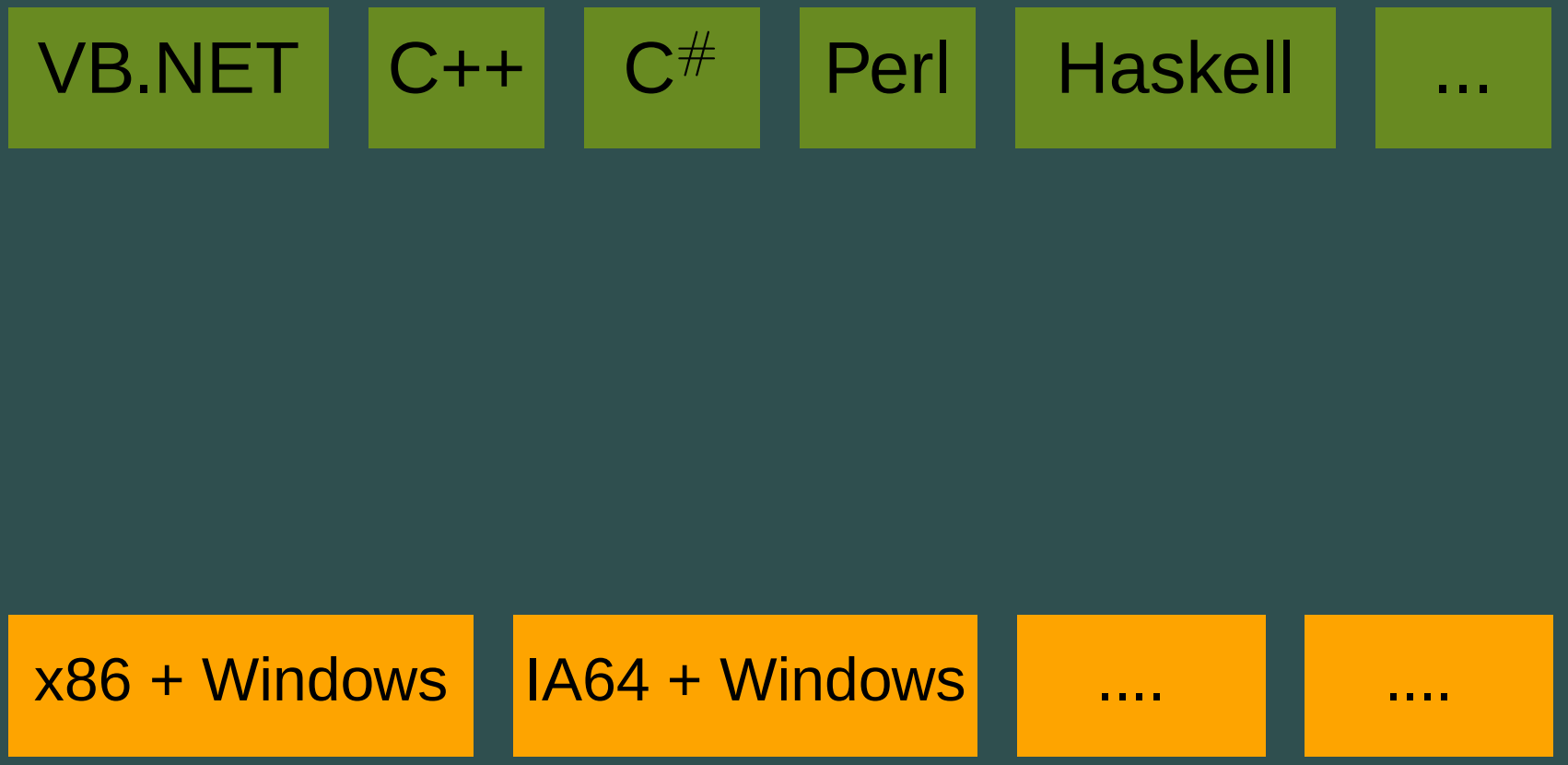
Need for interoperability

- Software **architectures** are getting more and more complex
- Need to reuse / costumize pre-existing software **components**
- Programming = “gluing” software together

Rôle of **interoperability**

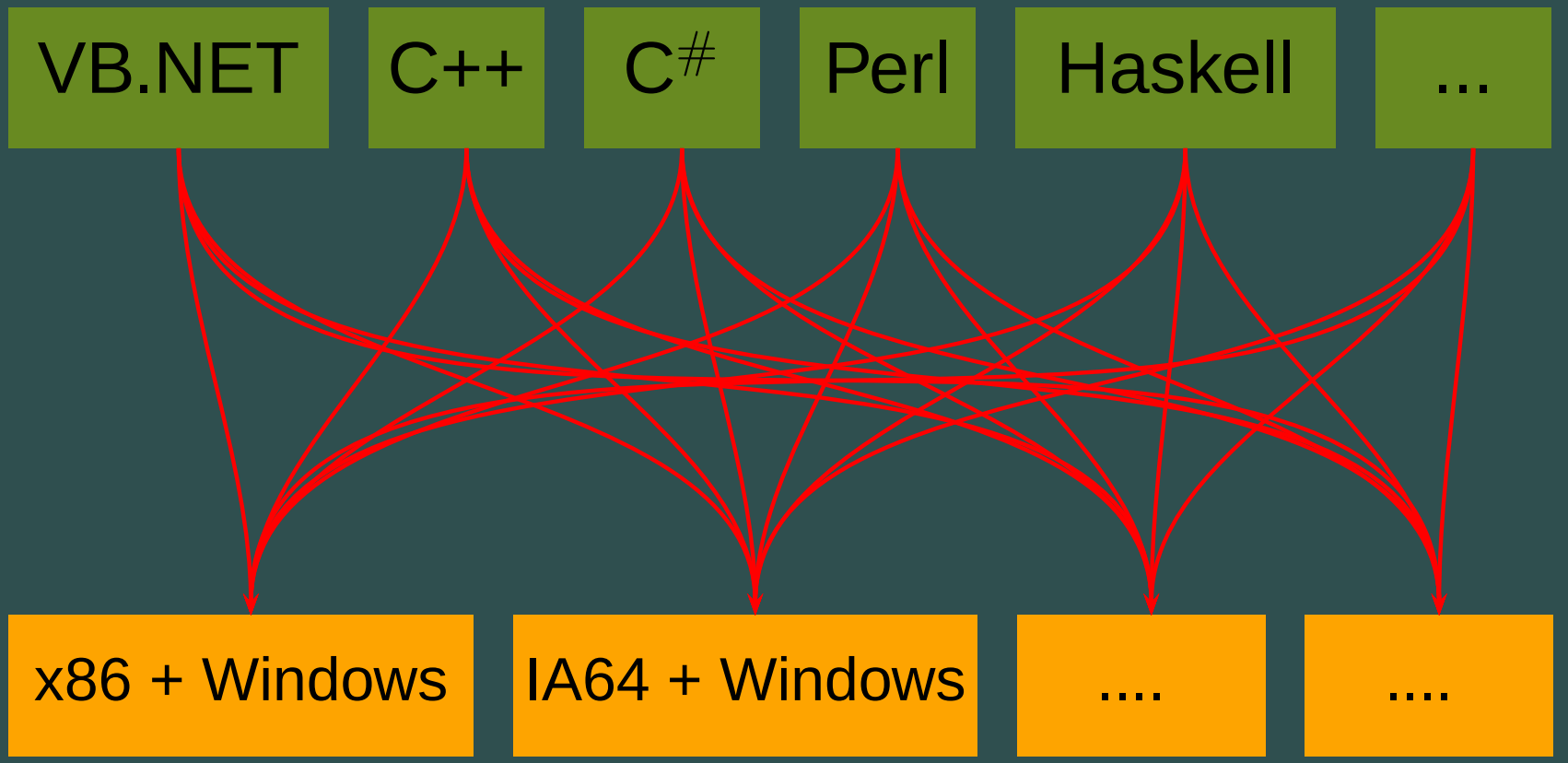
Interoperability in .NET

The (old) idea of UCSD P-code, AS-400, JMV ...



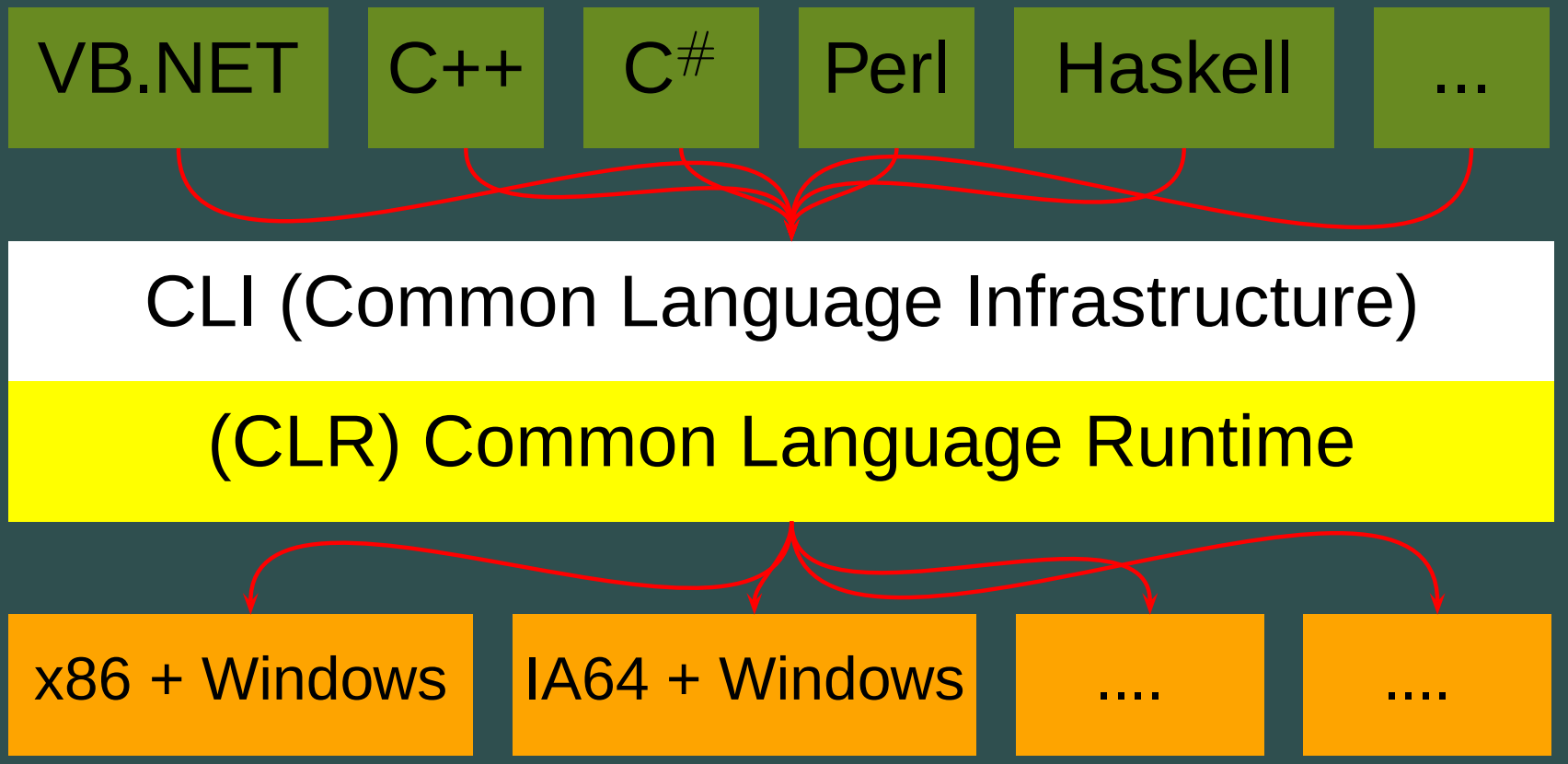
Interoperability in .NET

The (old) idea of UCSD P-code, AS-400, JMV ...



Interoperability in .NET

The (old) idea of UCSD P-code, AS-400, JMV ...



What’s new in CLI.NET

- .NET is thus bound to the capabilities of the CLI (Common Language Infrastructure)

What's new in CLI.NET

- .NET is thus bound to the capabilities of the CLI (Common Language Infrastructure)
- **Microsoft Research** and the .NET product group have attracted a number of computer scientists and language implementors to improve support for a wide range of programming paradigms

What's new in CLI.NET

- .NET is thus bound to the capabilities of the CLI (Common Language Infrastructure)
- **Microsoft Research** and the .NET product group have attracted a number of computer scientists and language implementors to improve support for a wide range of programming paradigms
- Thus novel features in CLI such as "tail-calls" (the `tail.call` instruction) saving stack space in **recursion**-based languages such as **Haskell**.

Haskell for .NET

Haskell.NET = Haskell (GHC) + Mondrian

Glossary:

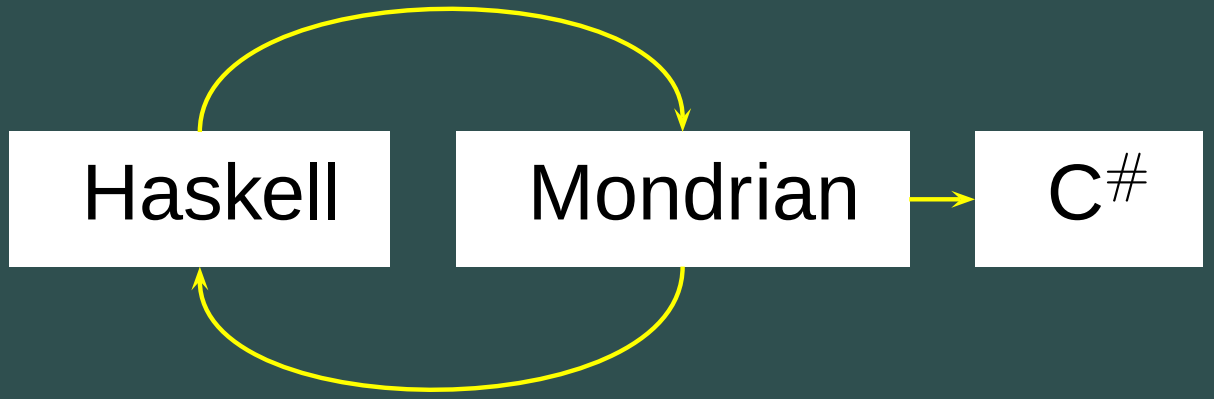
- **GHC**— Glasgow Haskell compiler
- **Mondrian**— OO/functional hybrid used to communicate with .NET
- `hs/ms` — Haskell/Mondrian source code
- `mc` — Mondrian internal code

About Mondrian

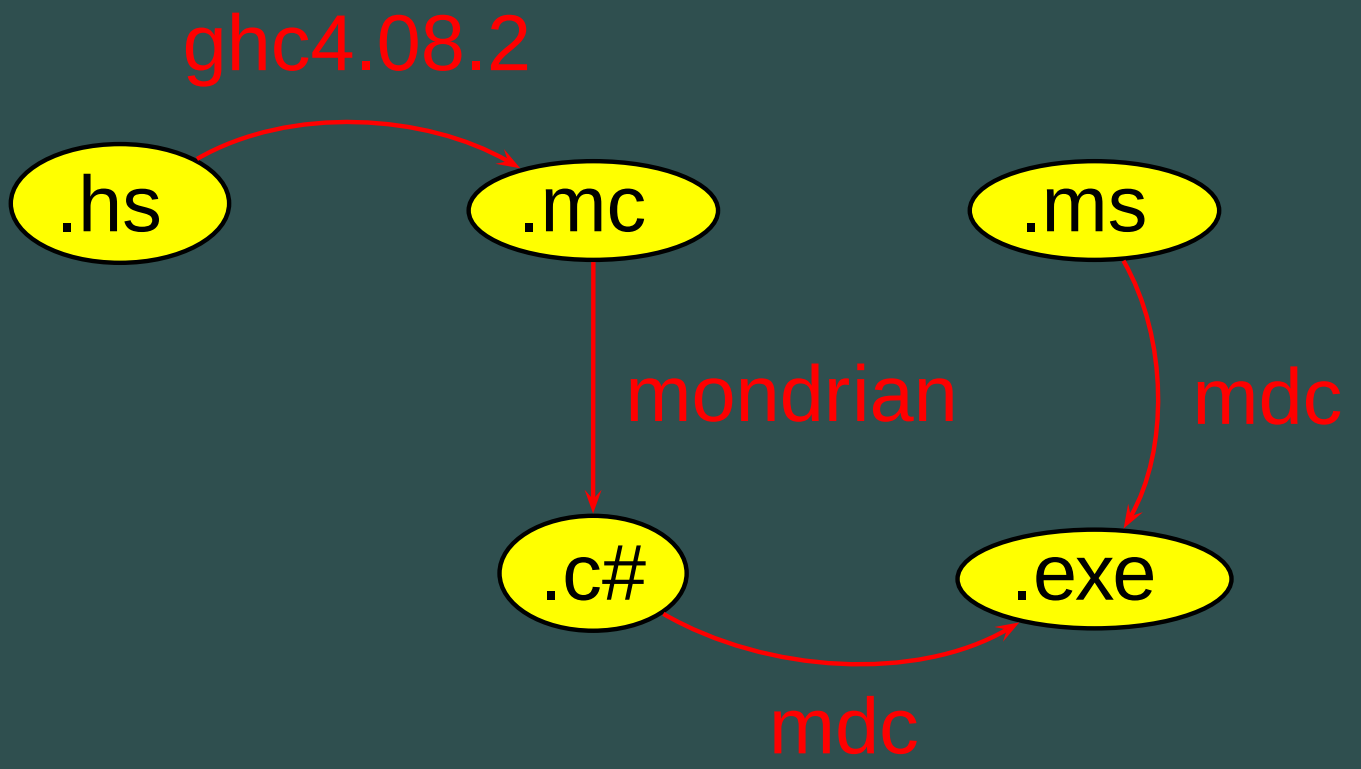
- **Mondrian** — is a non-strict functional language designed for an OO environment
- its syntax is a meld of that of Haskell and Java/C#
- disjoint union types are modelled using subtyping, eg

```
class List {}
class Nil extends List {}
class Cons extends List
{       head : Object;
        tail : List Object
};
```

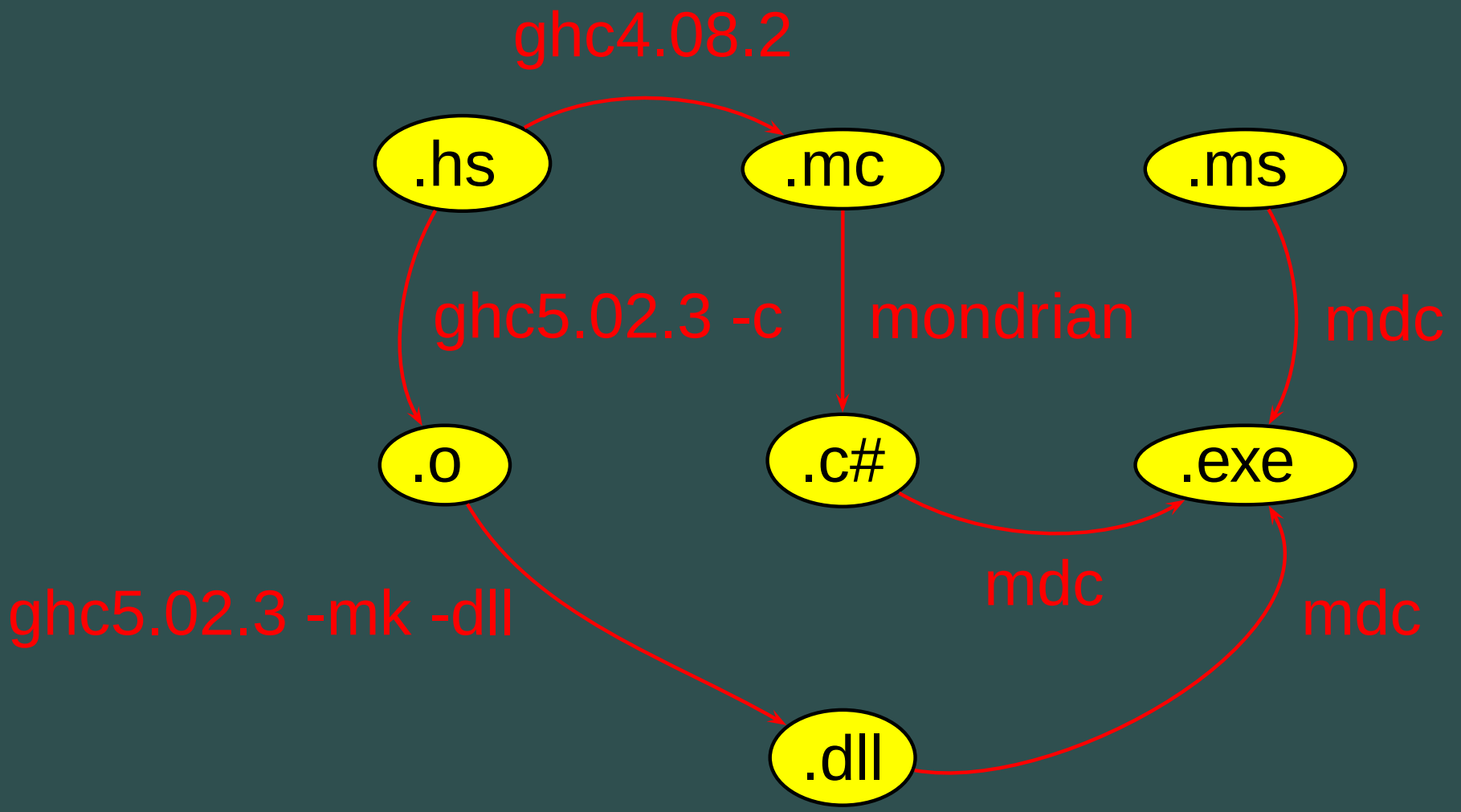
Mondrian (cont.)

- `Object` (the topmost class) plays the rôle of a type variable, cf. parameterization.
- functions are compiled to classes embodying a standard **evaluation** method
- supports concurrent programming using threads and exceptions
- provides access to "foreign" language objects
- 

Haskell.NET in a diagram



Haskell.NET in a diagram



Mobile phone revisited

```
using mondrian.prelude;
using mondrian.runtime;
using store;
...
public class SMain
{
        .....
        public static void store10(string c)
        {
                // Store new call in current list
                callList = store.Apply(c,callList) ;
                // Keep only 10 in current list
                callList = take.Apply(10,callList);
        }
```

Potential of .NET for FMs

- Aim of the **Programming Principles and Tools** (PPT) group at MR:

Potential of .NET for FMs

- Aim of the **Programming Principles and Tools** (PPT) group at MR:

  It *devises formal techniques and models for understanding programs, programming abstractions and languages, and develops related implementation technology.*

Potential of .NET for FMs

- Aim of the **Programming Principles and Tools** (PPT) group at MR:

  It *devises formal techniques and models for understanding programs, programming abstractions and languages, and develops related implementation technology.*

- Preliminary ideas for Haskell/Mondrian scripting were presented by Erik Meijer (Program Manager in the CLR Group) at **6408.70aAFP'98** (Braga) organized by DI/UMinho.

Are FMs cost-effective?

Are FMs cost-effective?

In favour:

- **FM** = discipline, rigour and good documentation (“safety net” for HR mobility)
- **FM**+**FP** = rapid prototyping, early feedback on what one is doing
- **FM**+**FP**+**CLI** = “time to market” integration
- **FM** = the only way to complex problems

Are FMs cost-effective?

In favour:

- **FM** = discipline, rigour and good documentation (“safety net” for HR mobility)
- **FM**+**FP** = rapid prototyping, early feedback on what one is doing
- **FM**+**FP**+**CLI** = “time to market” integration
- **FM** = the only way to complex problems

Against:

- Human factors
- Lack of **FM**-trained people

Are FMs a “may” or a “must”?

Control and automation:

- **Safety-critical** software development requires FMs
- **Dependable** computing

Data processing services are becoming critical:

- Many IS servers required to be available 365 $\times$ 24 hour / year
- Poor-quality data lead to wrong management decisions

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)

Interest in .NET

- Sidereus and DI/UM are experimenting with the .NET as a platform for formal/informal tool integration
- Some experiments/products follow:
  - ADO.NET
  - KMig
  - K-reverse
  - FRMS

Modelling ADO.NET

(...) The DataSet is an in-memory cache of data retrieved from a database (...)

```
data DataSet = DataSet {
        tables :: DataTableCollection,
        relations :: DataRelationCollection,
        caseSensitive :: Bool
        }
```

(...) While DataTable objects contain the data, the DataRelationCollection allows you to navigate through the table hierarchy (...)

```
type DataRelationCollection =
         Map RelationId DataRelation
```

From ADO.NET to KMig

- **KMig** is a data-migration package developed by Sidereus which interoperates with DTS.
- Data-migrations are formally specified in **KMig** internal language (**M2L**) and so can be checked for correctness before they are executed.
- **KMig** and **M2L** have been fully formally specified (most work carried out at specification level)

K-Reverse and FRMS

- Semi-automatic tool for relational database reverse specification.
- Can be used for data-quality checking, re-documentation, re-engineering (in connection with **KMig**).

K-Reverse and FRMS

- Semi-automatic tool for relational database reverse specification.
- Can be used for data-quality checking, re-documentation, re-engineering (in connection with **KMig**).
- **FRMS** is an advanced search engine incorporating fuzzy reciprocal matching.
- **FRMS** requirements are complex: it would have been a risk to approach them informally...

FMs add to competitiveness

Increased productivity:

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.

is competitive :-)

Closing

- 1st of April joke by John Peterson (maintainer of `haskell.org`) inspired in

  Simon L. Peyton Jones, Jean-Marc Eber, and Julian Seward. Composing contracts: an adventure in financial engineering, functional pearl. In *International Conference on Functional Programming*, pages 280–292, 2000.

(thanks to Andrei Serjantov for the broadcasting...)

A Few Links

- More about Microsoft Research **PPT** group:

  `http:/research.microsoft.com/ppt/`

- More about **FM**s

  `http://www.fmeurope.org`

- More about **Haskell**

  `http://www.haskell.org`

- More about **Mondrian**

  `http://www.mondrian-script.org`

- More about us

  `http://www.sidereus.pt`

  `http://www.di.uminho.pt`