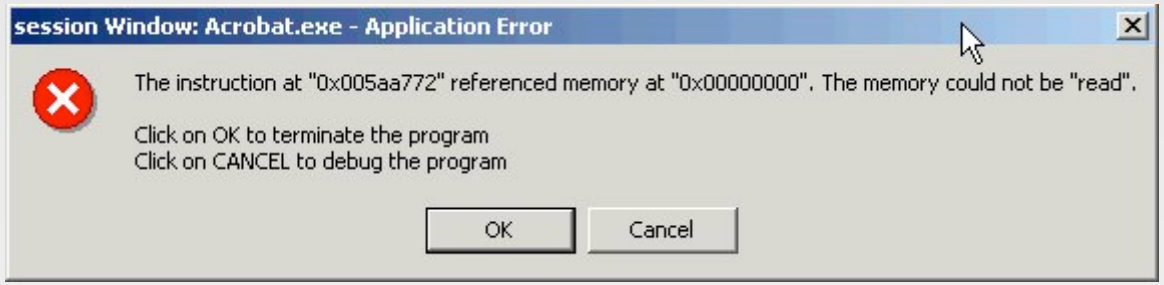
Dependable software deployment

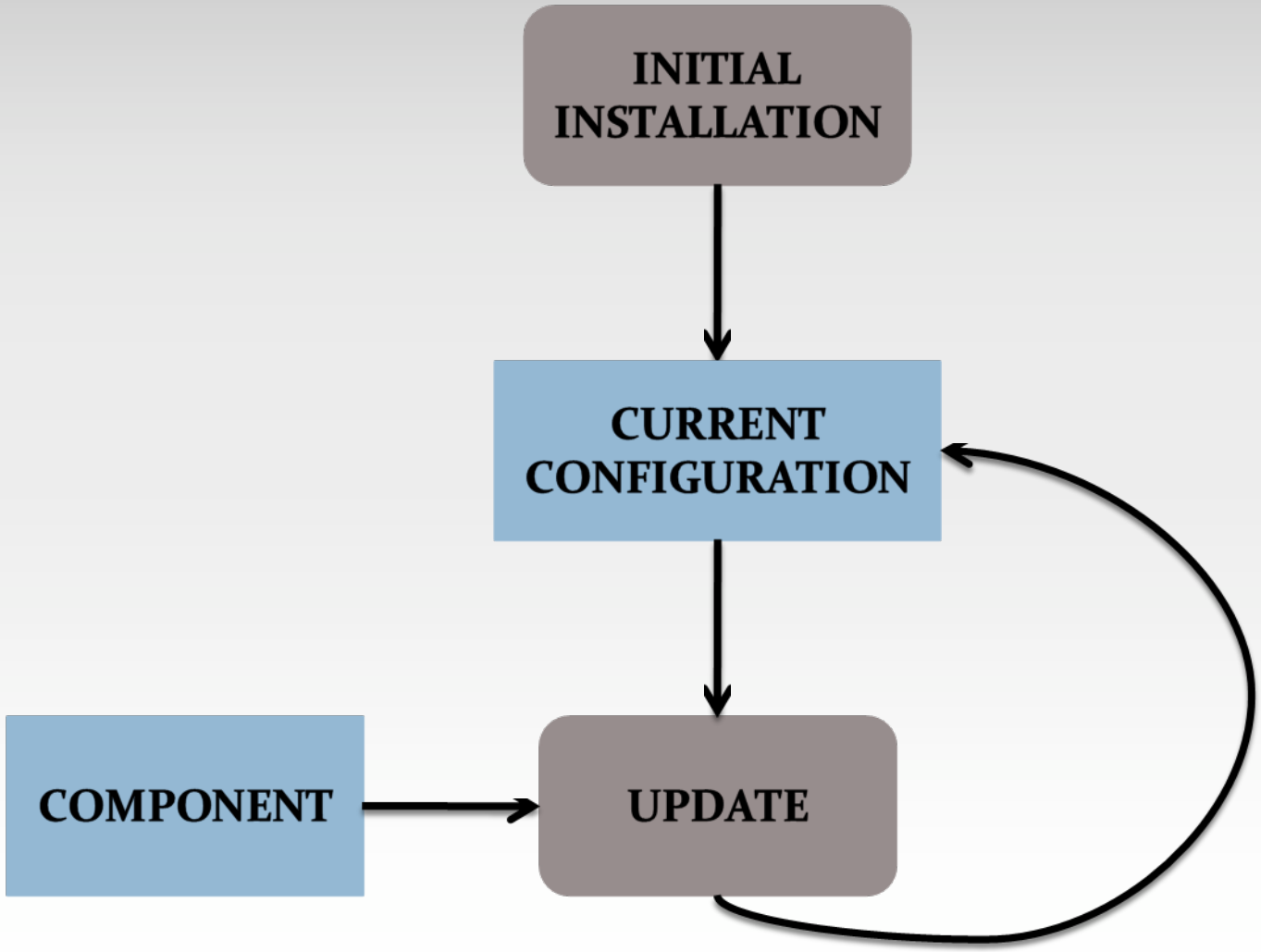
Wouter Swierstra

13 October 2006

Software configuration mismanagement

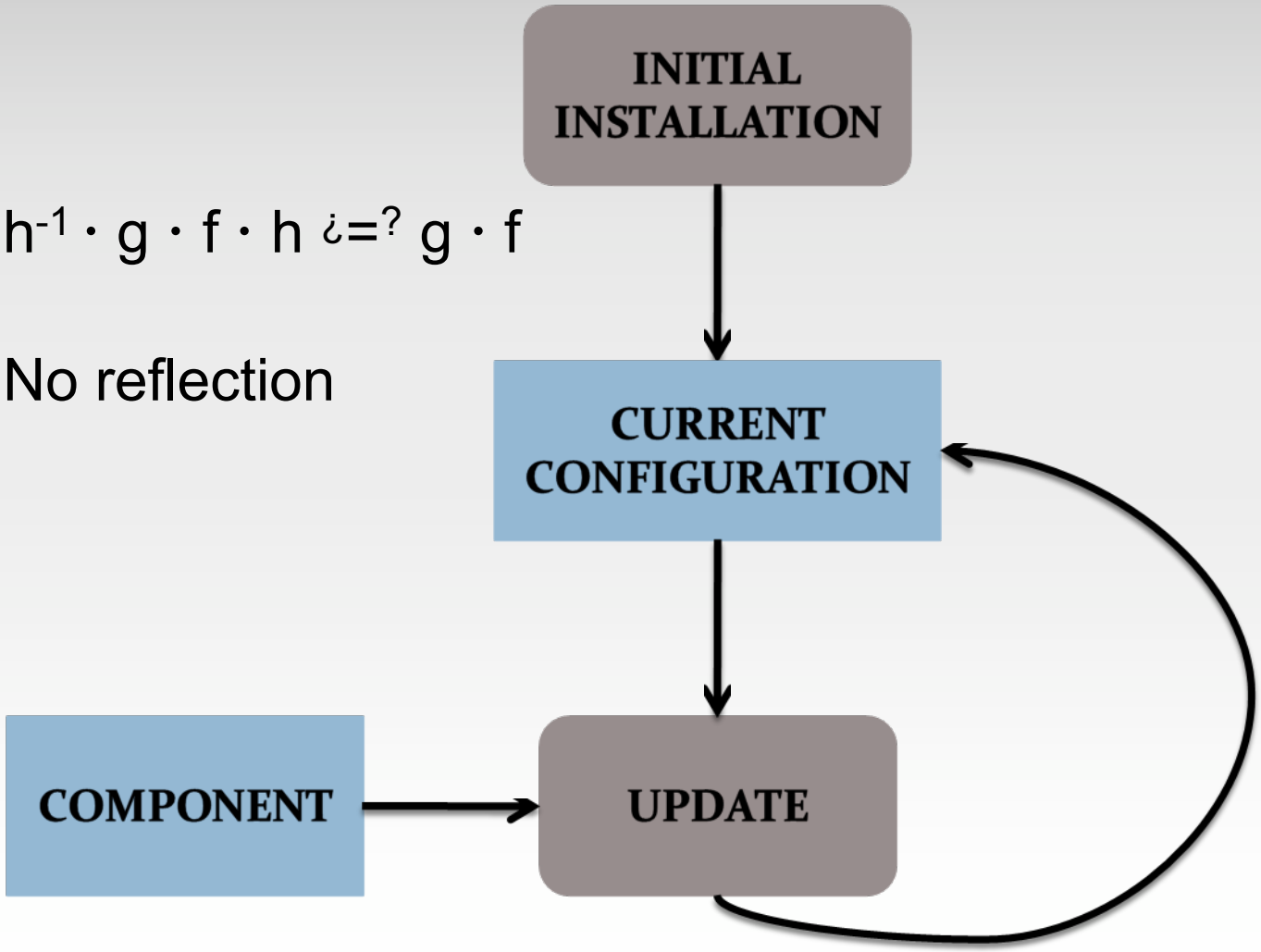


Imperative software deployment

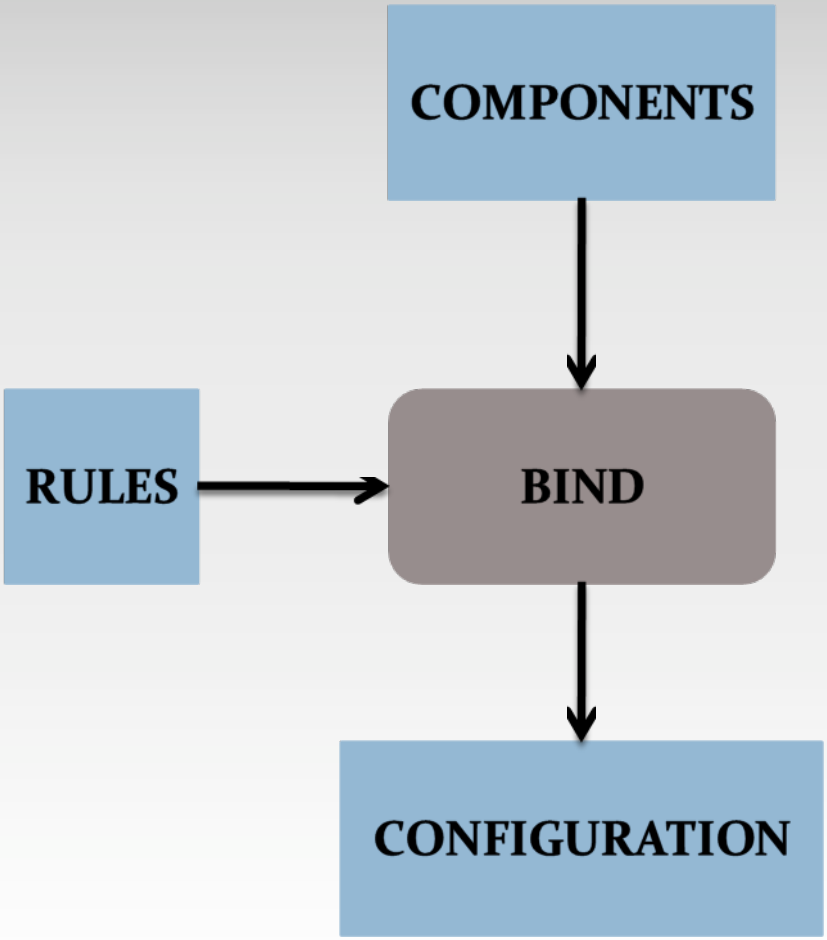


Imperative software deployment

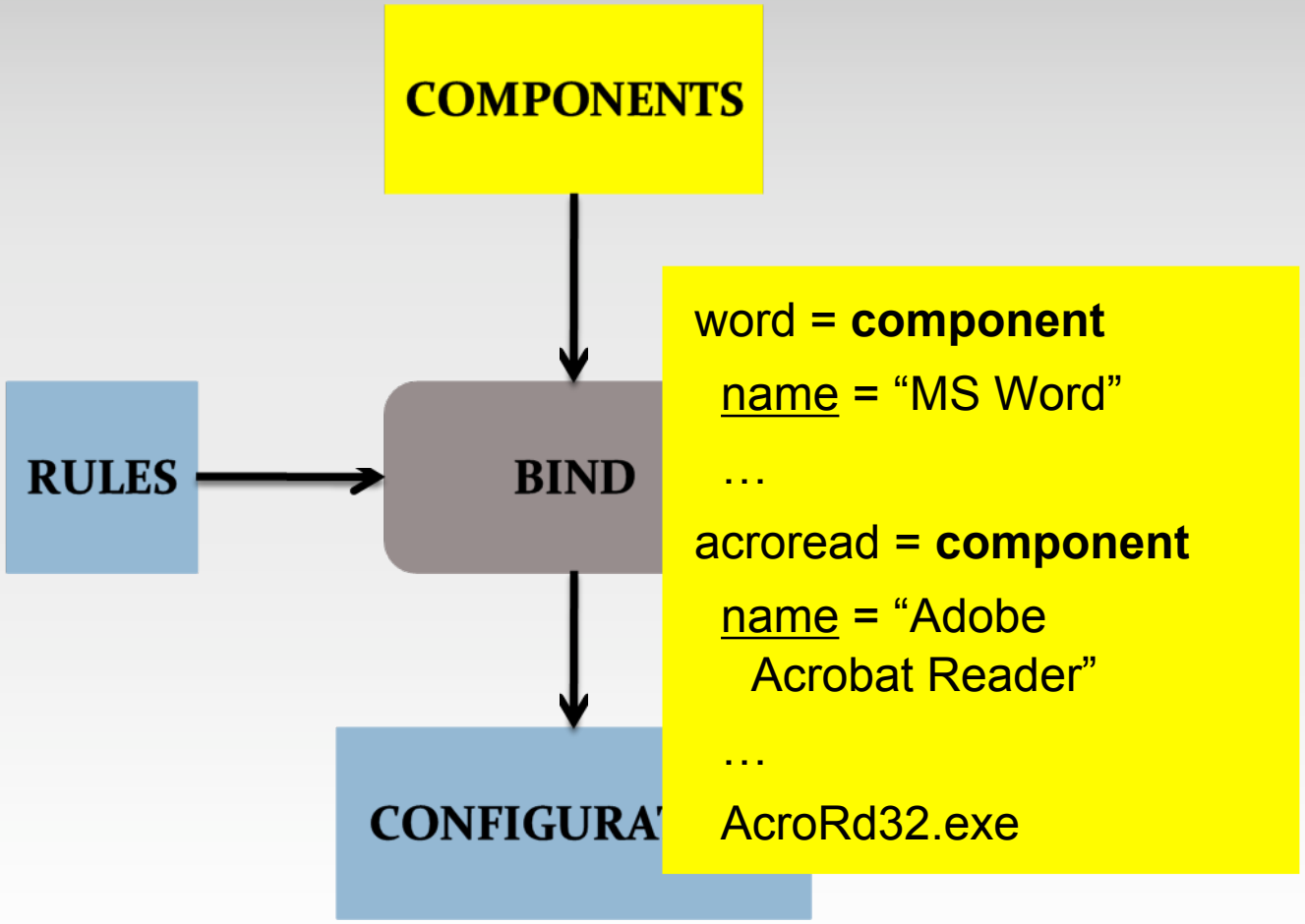
- $h^{-1} \cdot g \cdot f \cdot h \; ¿=^{?} \; g \cdot f$
- No reflection



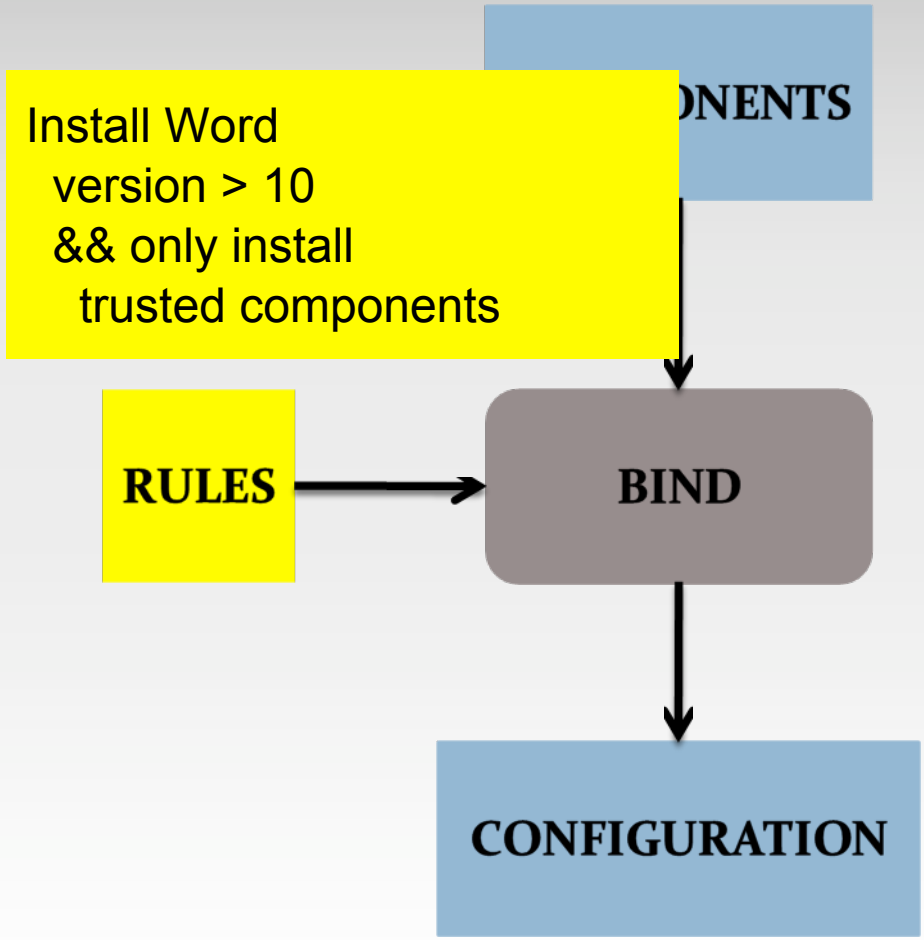
Functional deployment



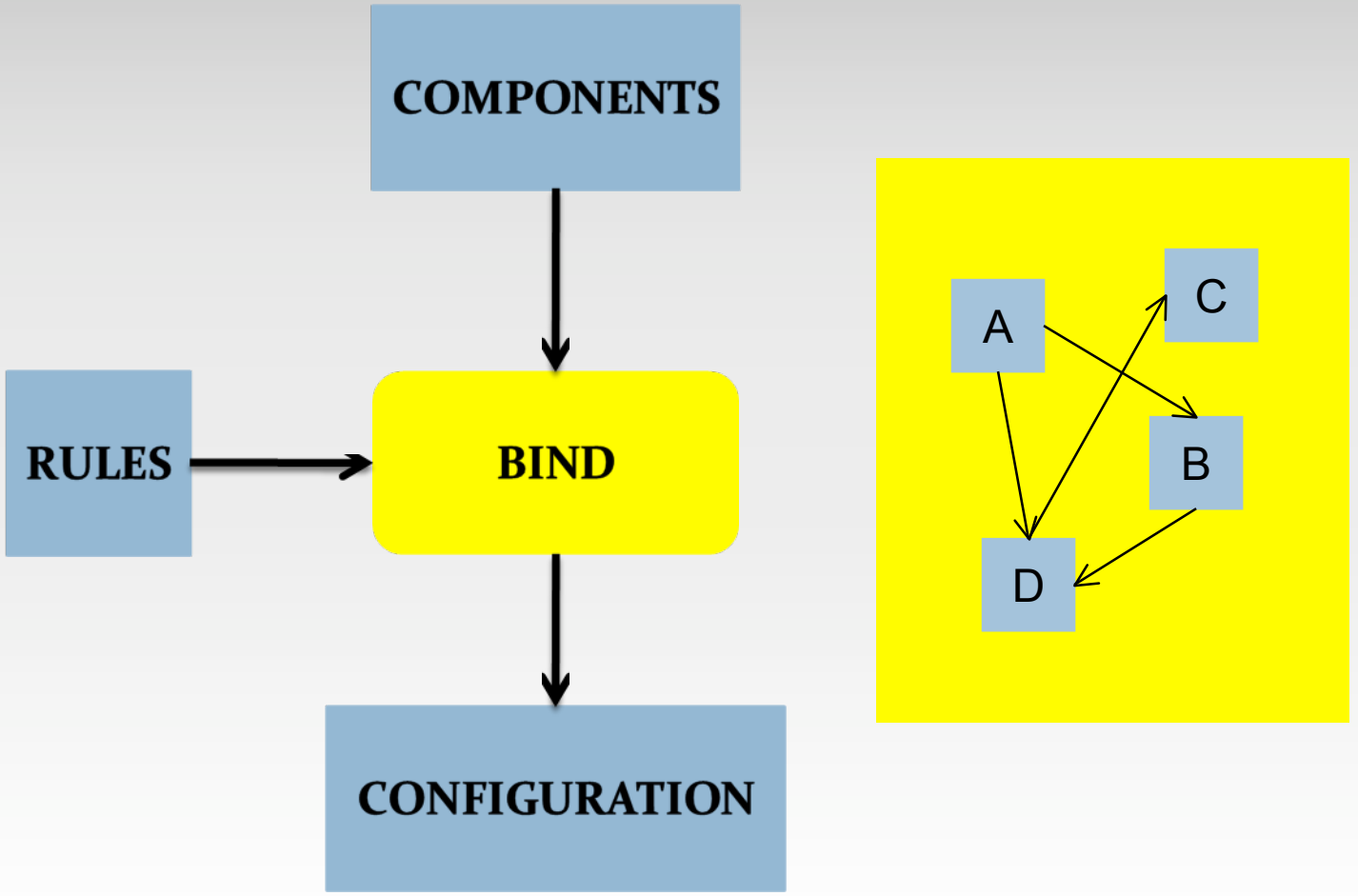
Functional deployment



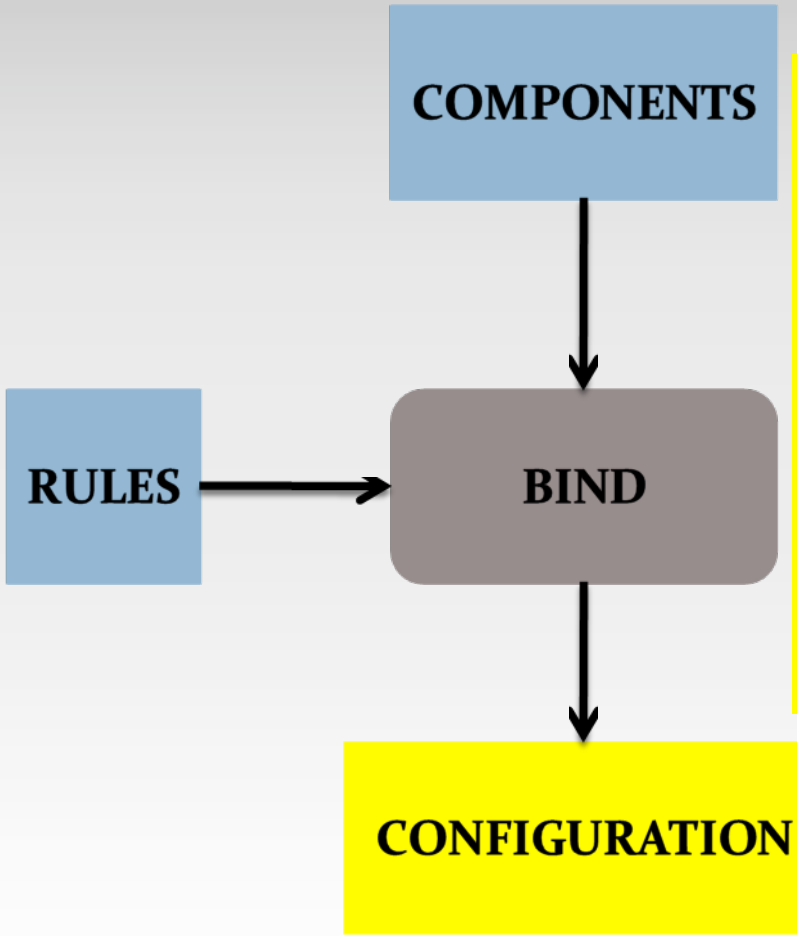
Functional deployment



Functional deployment



Functional deployment



Configurations are:

- Immutable
- Reproducible
- Analyzable

The deployment problem

- Given a set of available components and a set of rules and requirements, construct “the best” configuration.

What is a configuration?

C = **configuration**

notepad = **component**

name = “Notepad”

version = 5.0.3

notepad.exe = **executable**

What is a configuration?

C = **configuration**

winc = **component**

name = “Windows Kernel”

version = 5.0.3

msvcrt.dll = **library**

signal = **proc**

What is a configuration?

C = **configuration**

winc = **component**

name = “Windows Kernel”

version = 5.0.3

msvcrt.dll = **library**

signal = **proc**

ordinal = 759

Imports

C = **configuration**

readline = **component**

readline.dll

…

ghc = **component**

name = “Glasgow Haskell Compiler”

version = 6.4.2

requires = readline • readline.dll

When do configurations make sense?

- **Well-formed:**
  - A configuration should only have components as children:

```
forall c in children C . sort c = component
```

  - Components should not be nested.

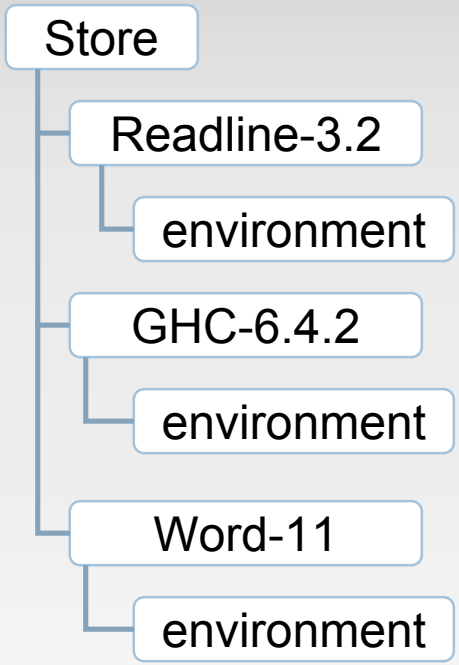
```
sort v = component =>
  forall c in allChildren v . sort c != component
```

When do configurations make sense?

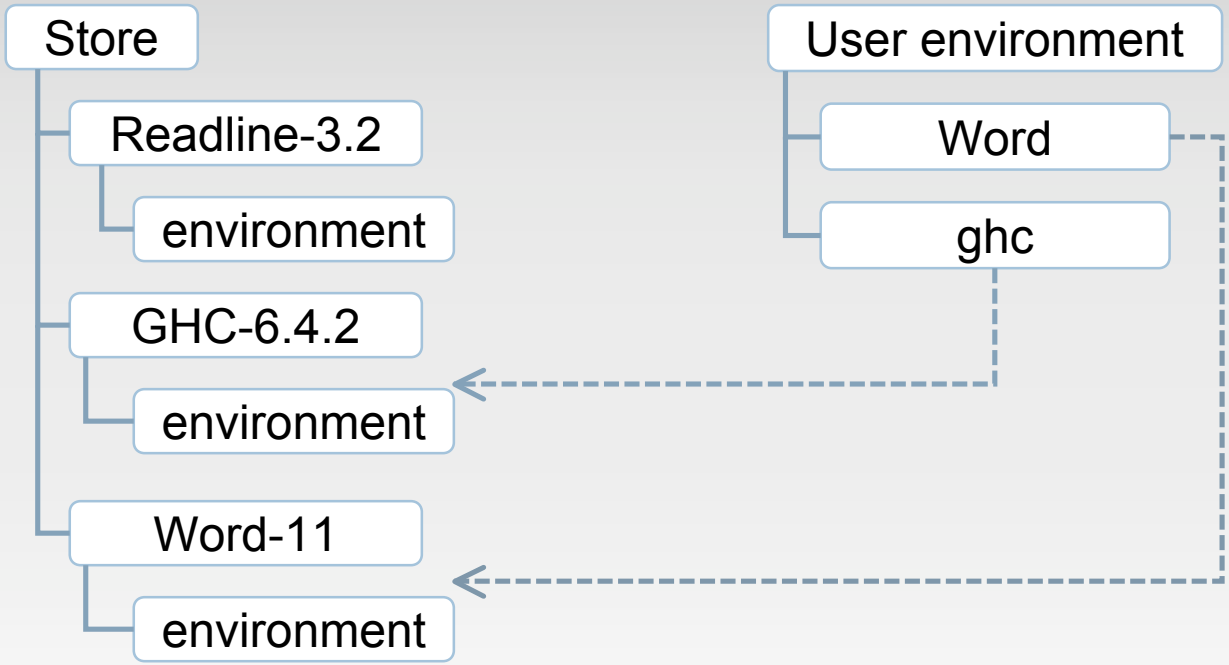
- **Resolved:**
  - Every name can be found:

freeVar(C) = Ø

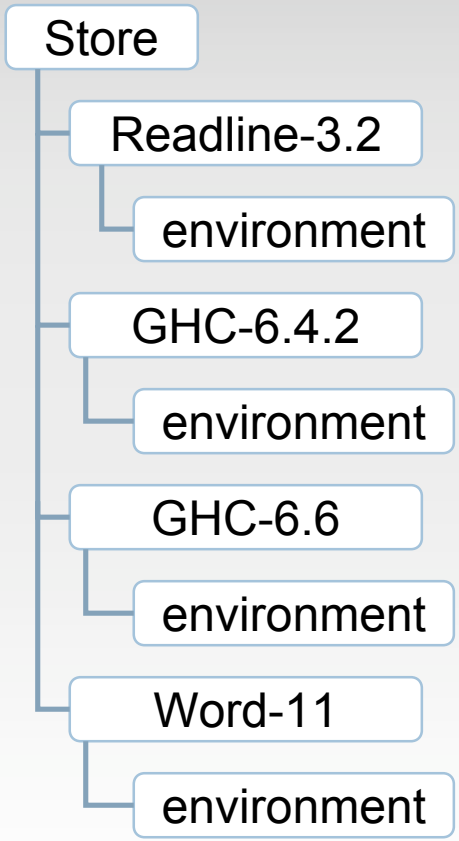
What's on disk?



What's on disk?



What's on disk?

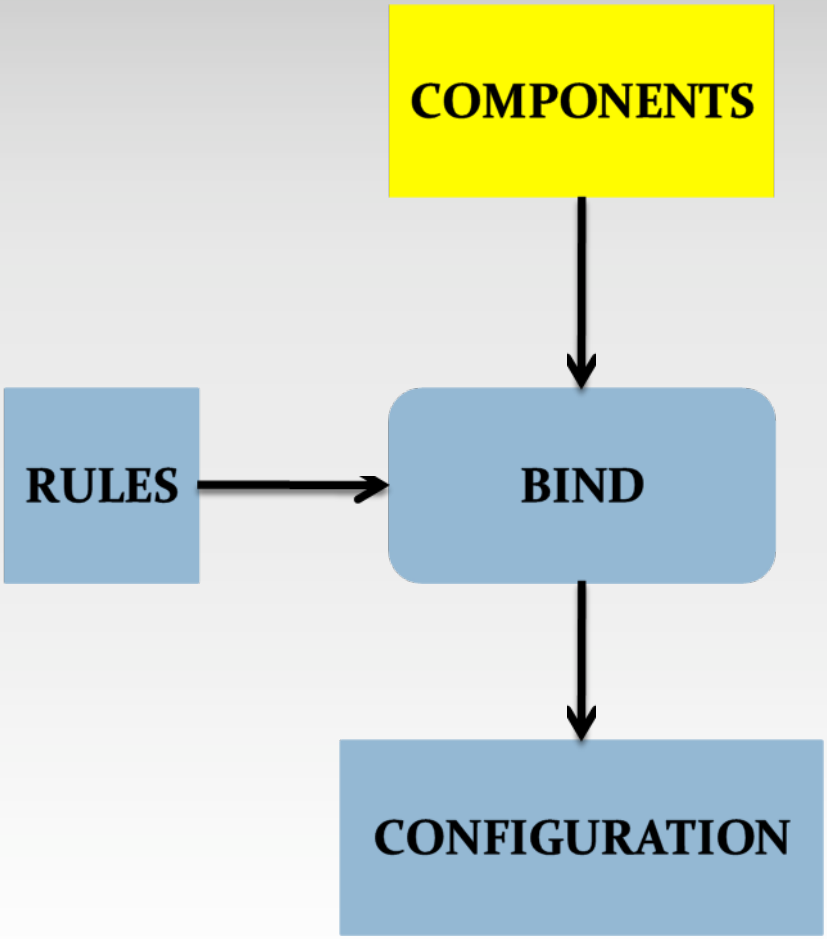


- Multiple versions of components
- Minimize interference
- Hidden from user
- Unambiguous bindings

Memory model

Programming Languages	Example	Software Deployment	Example
Memory		Disk	
Values	5, “Hello”, …	Components	libc, ghc, …
Addresses	0x005aa772	Path names	“/usr/local/”
Pointer arithmetic	*(arr + 5)	String manipulation	“C:\wouter\” + configDir

What are components?



Towards deployment

- How should a developer know how to refer to the component called “readline” on *your* system?

- We need to parameterize components:

ghc rl = **component**

name = “Glasgow Haskell Compiler”

version = 6.4.2

requires = rl • name = “readline”

Deployment

- Finding a component with the right name might not be enough…

ghc rl = **component**

name = “Glasgow Haskell Compiler”

version = 6.4.2

requires = rl • name = “readline” &&

rl • version > 3.0

Predicates - I

- We don’t want to fix our predicate language.
- First-order predicate logic.
- Versions:

  readline • version > 3.0

  libc • version <= 5.0

Predicates – II

- Disjunctive dependencies:

readline cc = **component**

requires = cc • name == “GNU C Compiler”

or cc • name “Visual Studio”

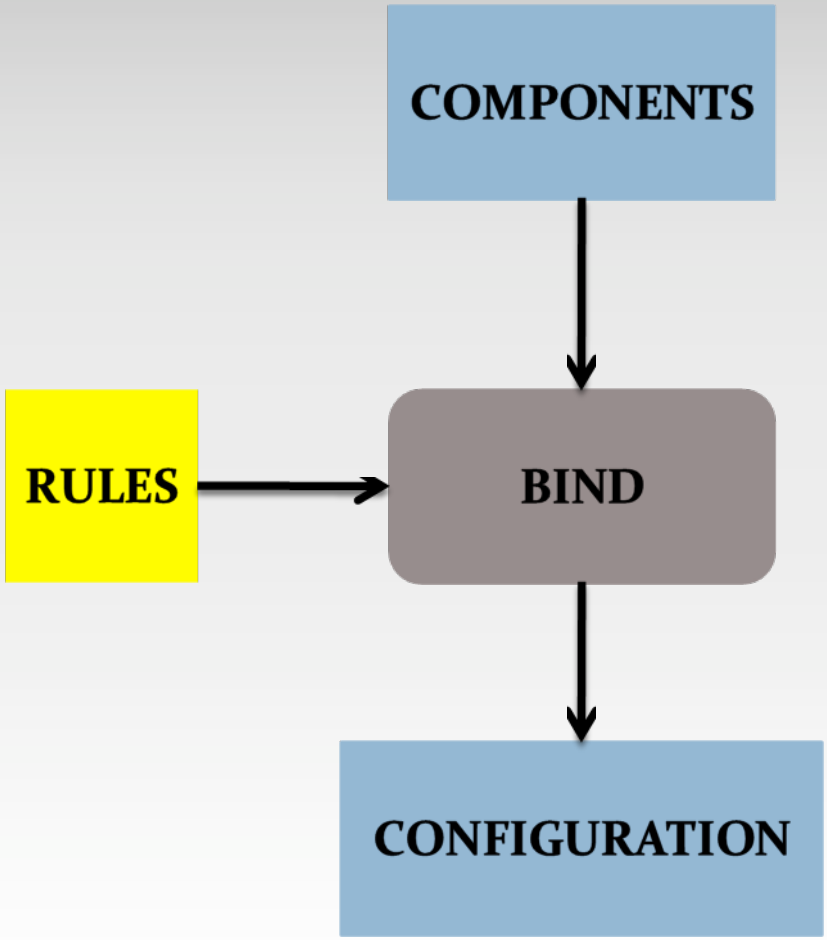
Predicates - III

- Defining recursive configurations allows global constraints:

```
nvidia config = component
  requires =
    forall c in children config .
      c • name == “Monitor driver”
        => c == nvidia
```

- A good predicate language is really, really important.

Guiding the binding



Policies - I

- What if you have more than one choice?
- A policy is a partial order on components.
- State of the art:
  - c.name == d.name => c.version > d.version

Policies - II

- Many websites publish lists that rate software.

- Security, given a rating function:
  - rate(c) >= rate(d)

Policies - III

- Parsimony, given a size measure and installed predicate:
  - if installed(c) then 0 else size(c)
    <= if installed (d) then 0 else size(d)

Windows Installer

- Analyzed lots of msi files
- Declares complete component contents…
- …but deploy files in shared directories
- …and allow custom actions to affect where files are deployed.
- No real predicate language.

Red Hat Package Manager

- Packages specify name, version, dependencies,
- Fixed, simple predicate language.
- No two versions of same component.
- Scripts to build and deploy can execute arbitrary actions.

Conclusions

- A good idea of what the problem is.
- Still open questions:
  - Plug-ins
  - User settings
  - Generating faithful component descriptions
  - …
- Draft paper available.