Part V - Features of Composition Languages
50. Configuration with Acyclic Composition Programs

1) Configuration management with acyclic comp. programs
2) Lazy evaluation of composition programs

Obligatory Reading

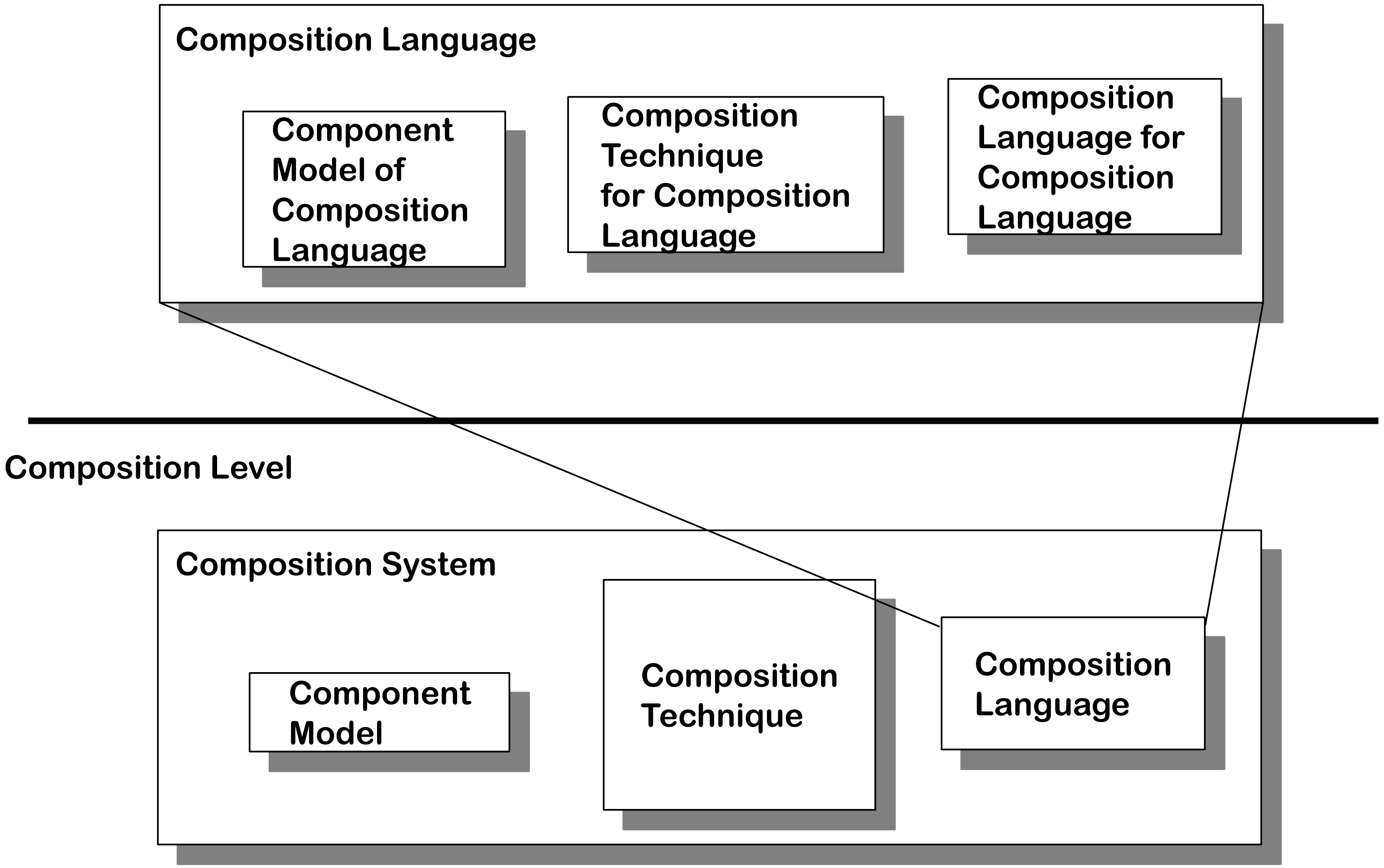
- ISC book Chapter 3, 4

Literature

- Dami, Laurent. Software Composition. PhD University Geneva 1997. The centennial work of the Lambda-N calculus
- Mulet, P., Malenfant, J., Cointe, P. Towards a Methodology for Explicit Composition of MetaObjects. OOPSLA 98.
- Forman, Danforth: MetaClasses in C++. Addison Wesley. 1999. Excellent book on metaclasses and metaclass composition.
- Oscar Nierstrasz and Theo Dirk Meijler. Requirements for a composition language. In Paolo Ciancarini, Oscar Nierstrasz, and Akinori Yonezawa, editors, Object-Based Models and Langages for Concurrent Systems, LNCS 924, pages 147-161. Springer, 1995.

Component and Composition Language Level

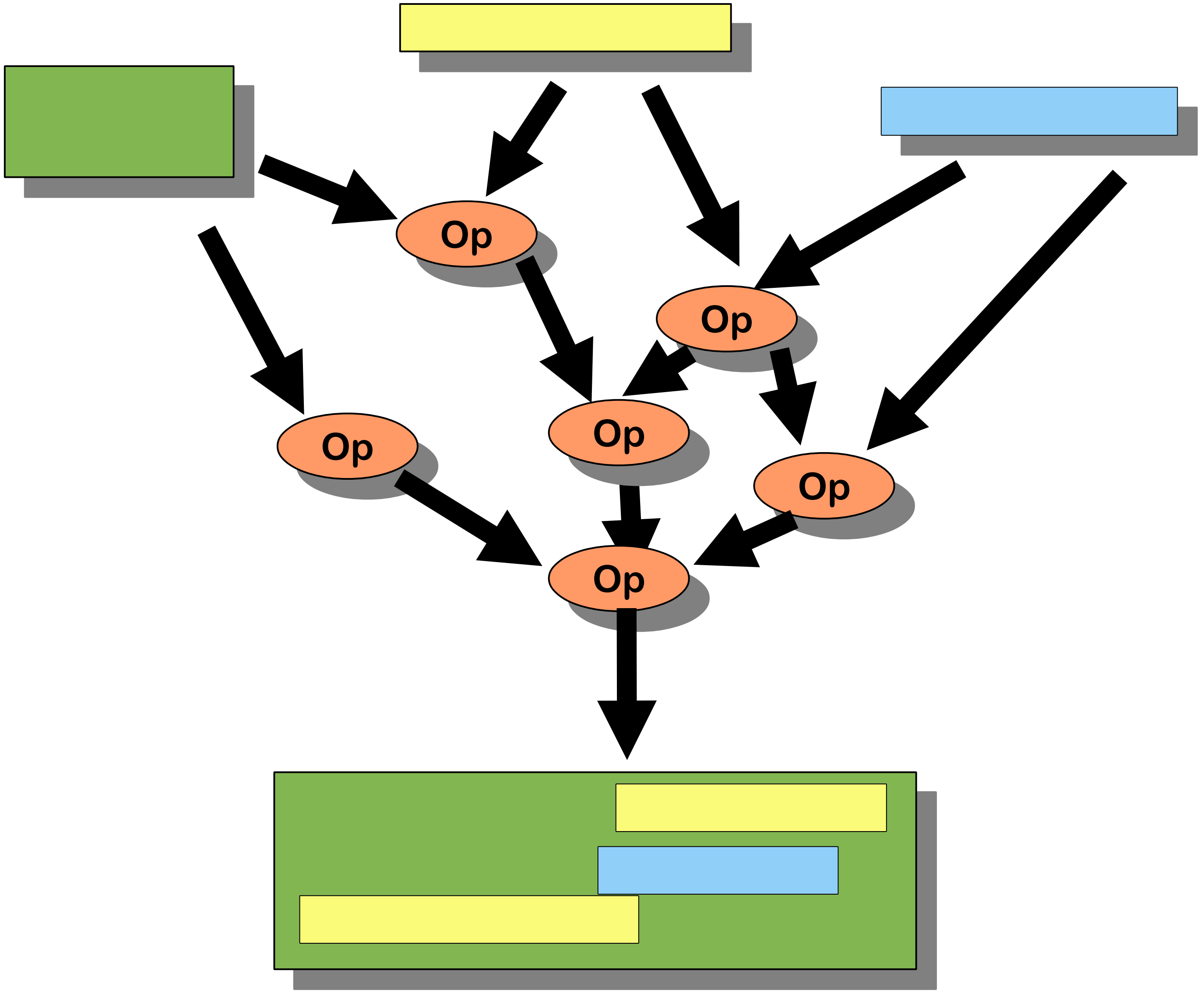
- Holds for black-box and grey-box composition systems

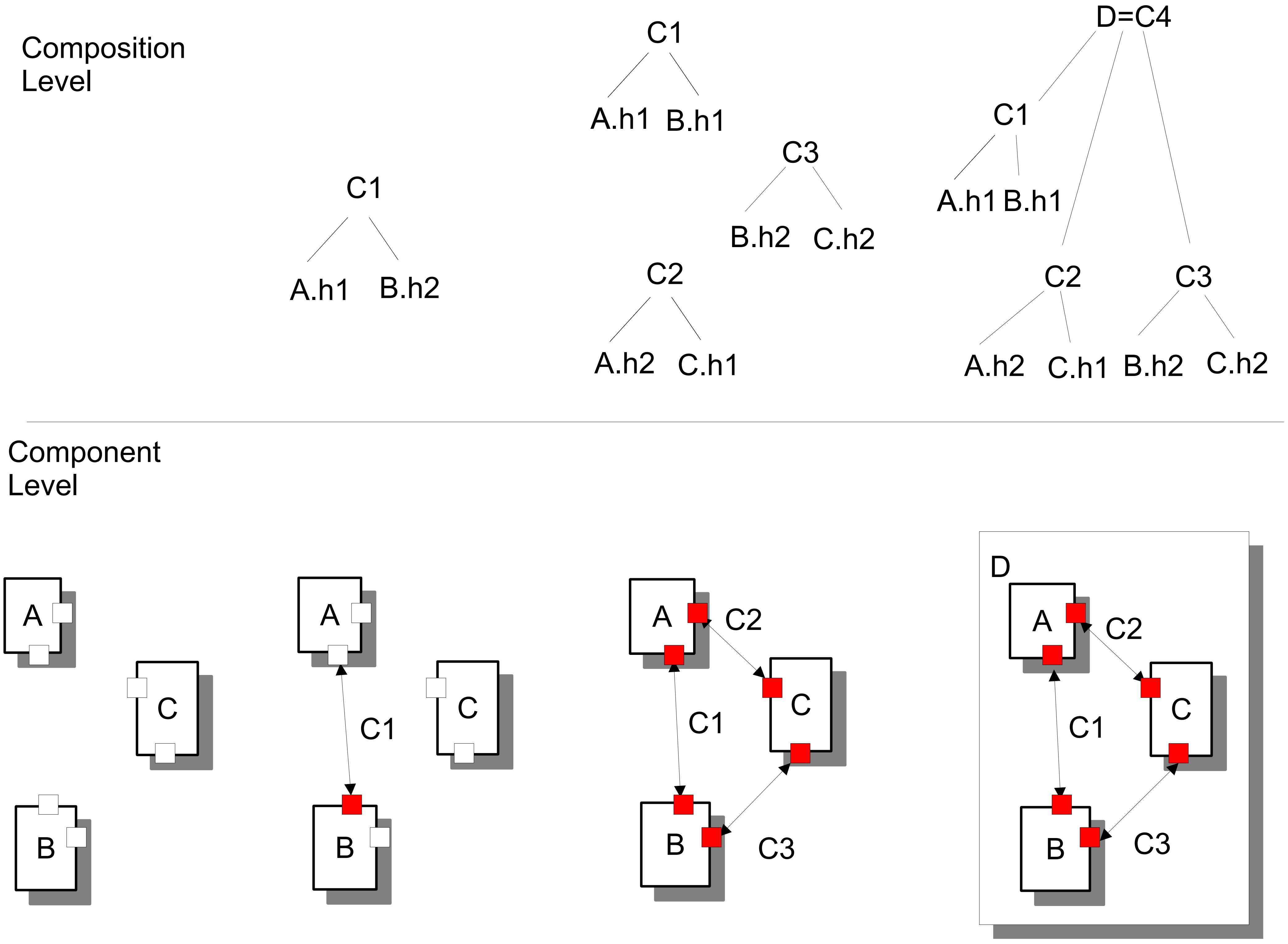

Composition Language
Component Model of Composition Language
Composition Technique for Composition Language
Composition Language for Composition Language
Composition Level
Composition System
Component Model
Composition Technique
Composition Language

System Builds as Composition Expressions and Programs

- A *composition expression* or *composition program* in a *composition language* describes a *system build*

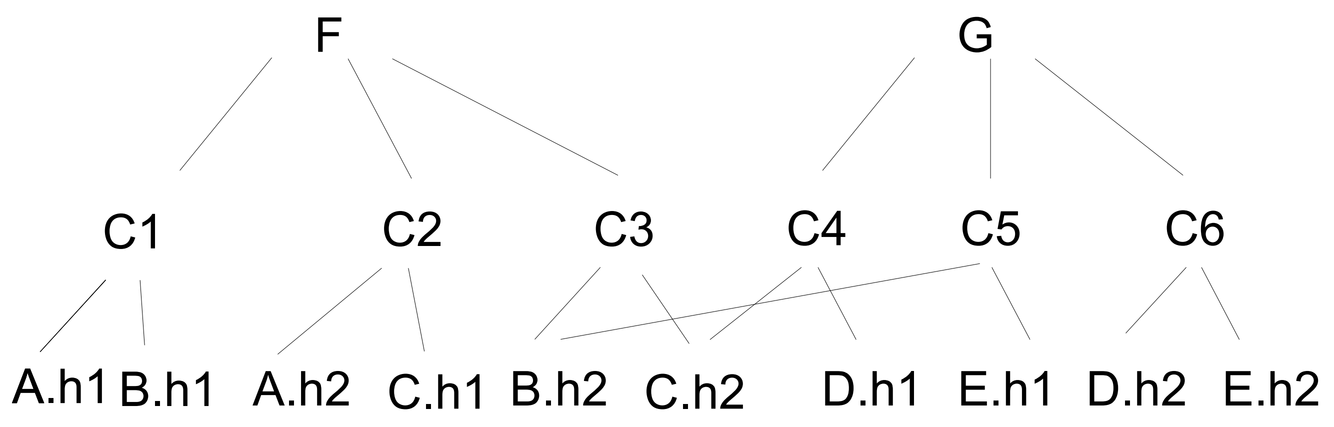
A System Builder Executes a Composition



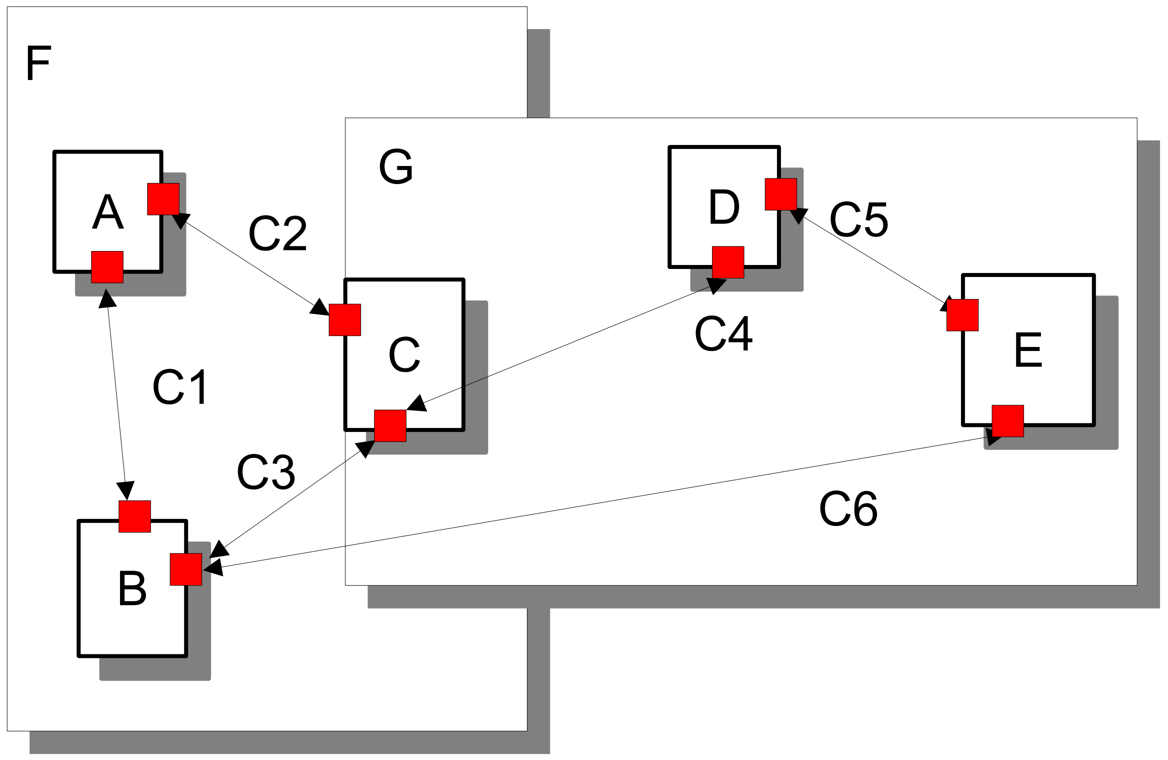


Composition Level
C1
A.h1
B.h2
C1
A.h1
B.h1
C3
B.h2
C.h2
C2
A.h2
C.h1
D=C4
C1
A.h1
B.h1
C2
A.h2
C.h1
C3
B.h2
C.h2
Component Level
A
C
B
A
C1
C
B
A
C2
C1
C
B
C3
D
A
C2
C1
C
B
C3

Composition Level



Component Level



50.1 Configuration Management With Acyclic Composition Programs

Turing-Completeness of Composition Languages

- If a composition language is *not* turing-complete
  - The architecture of the system is simple
  - Can be analyzed much better:
    - Termination can be proved
- If a composition language *is* turing-complete
  - The system is more complex
  - Complex architectures, also recursive ones, can be described

Configuration as Control-Flow of Composition Programs

- Composition programs may contain control-flow statements
- They are executed *before* the components run
  - They *configure* the components, because they depend on static control-flow conditions
    - Global configuration variables

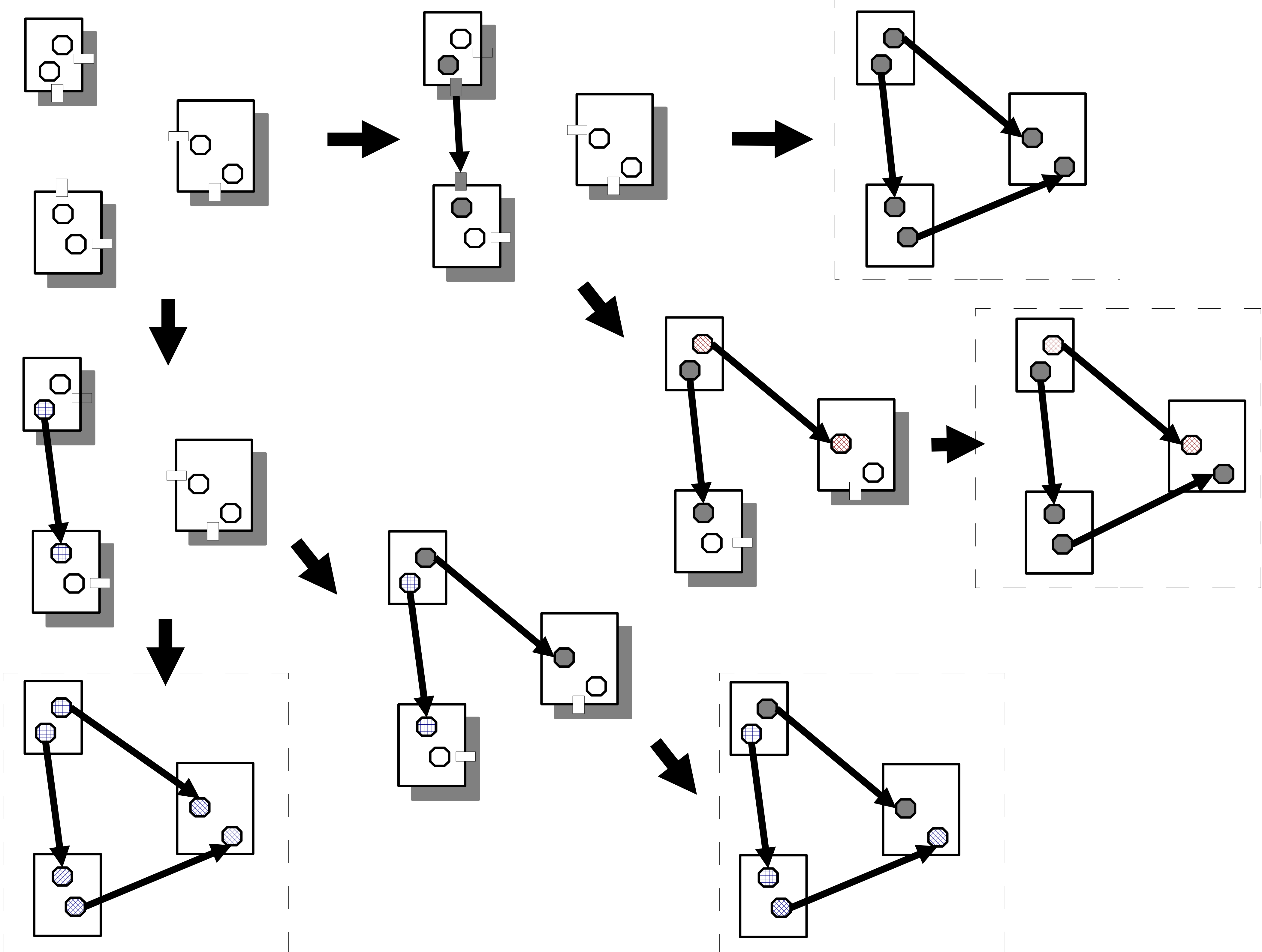
A configuration of a system relies on an acyclic composition program.

A Configuration Variable

- This composition program is a configuration because it is acyclic
- Its variables are **configuration switches**

Configuration switch (configuration variable)

```
// Variant selection for instantiation of generic parameter
public class CompositionProgram {
  public static void main (String[] argv) {
    if (argv[1].equals("-tin")) variant1= true;  else variant1 = false;
    ClassBox SimpleList = compositionSystem.createClassBox("SimpleList");
    if (variant1) {
      ClassBox bagOfPieces =
        SimpleList.bindGenericType("ElementType","Tin");
    } else {
      ClassBox bagOfPieces =
        SimpleList.bindGenericType("ElementType","MetalPlate");
    }
  }
}
```



Traditional Configuration with Cpp

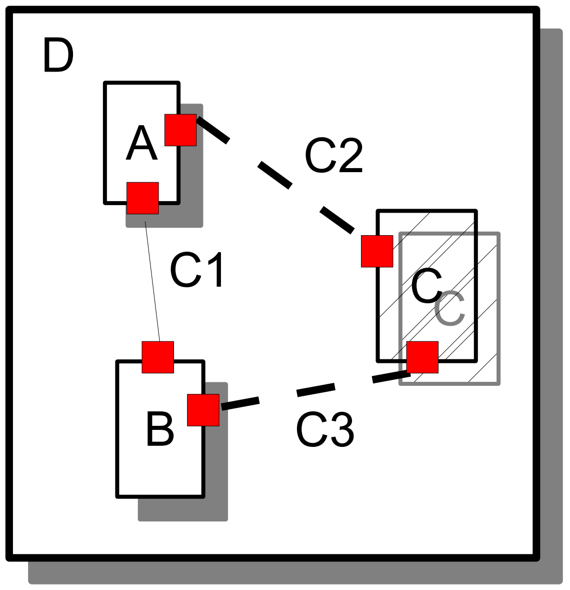
- The C preprocessor is a simple acyclic composition/configuration language
    - with configuration switches for fragment configuration
  - Evaluated statically, before compilation

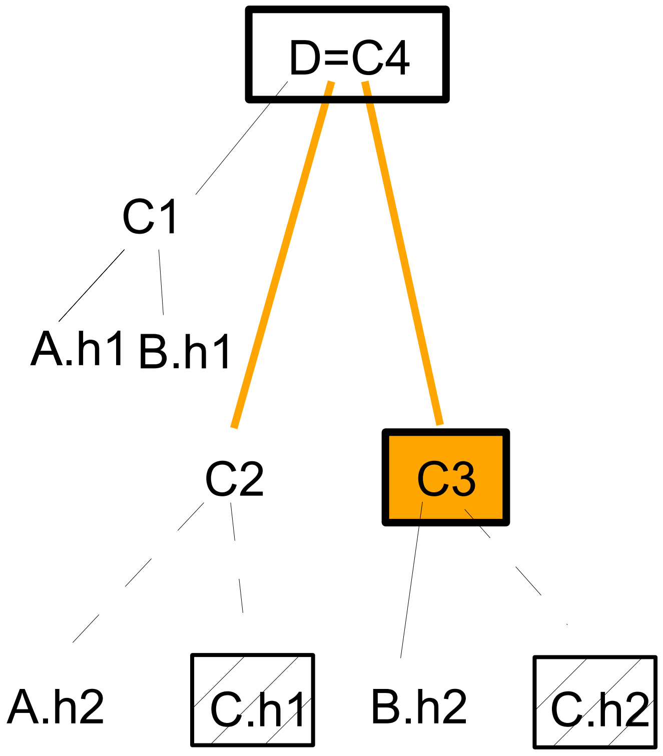
```
#ifdef ConfigurationVariable
  <fragment variant 1>
#else
  <fragment variant 2>
#endif
```

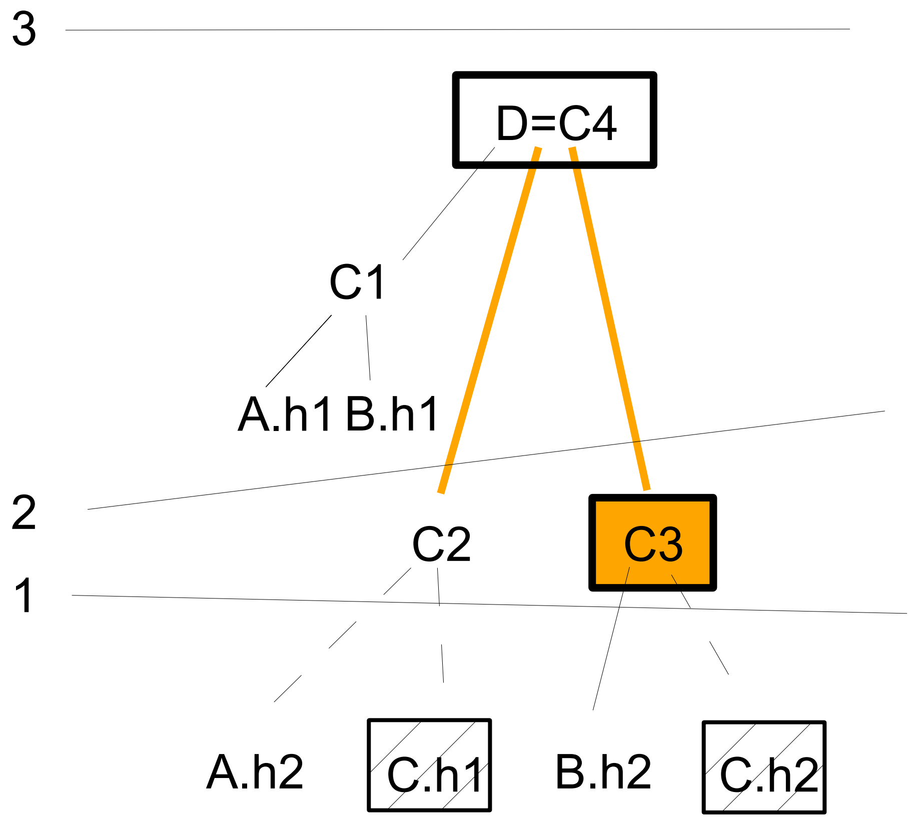
50.2 Lazy Evaluation of Composition Programs

Eager and Lazy Builds of Composition Programs

- As all programs, composition programs can be evaluated with different evaluation strategies
- Eager: direct execution of all composition operations
- Lazy: as needed
- Lazy evaluation is important when
  - Something changes and the system architecture should be recomputed


D
A
C2
C1
C
C
B
C3


D=C4
C1
A.h1
B.h1
C2
C3
A.h2
C.h1
B.h2
C.h2


3
D=C4
C1
A.h1
B.h1
2
C2
C3
1
A.h2
C.h1
B.h2
C.h2

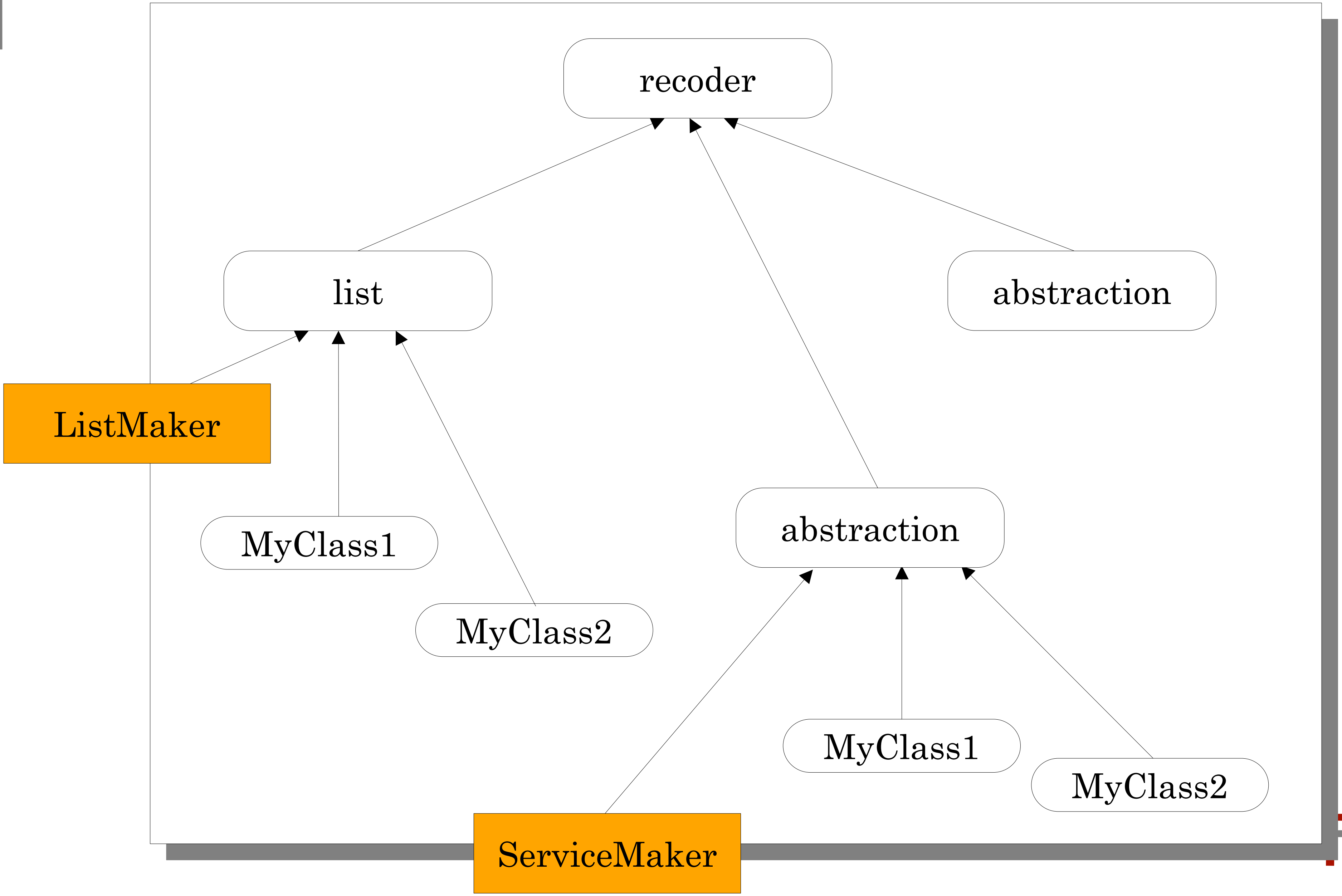
Make as an Example

- Make is a lazy system builder
- Composition language is rule-based
    - Rule dependencies are lazily recomputed
    - Composition expressions are applications of UNIX tools (compiler, linker, generator, preprocessor)

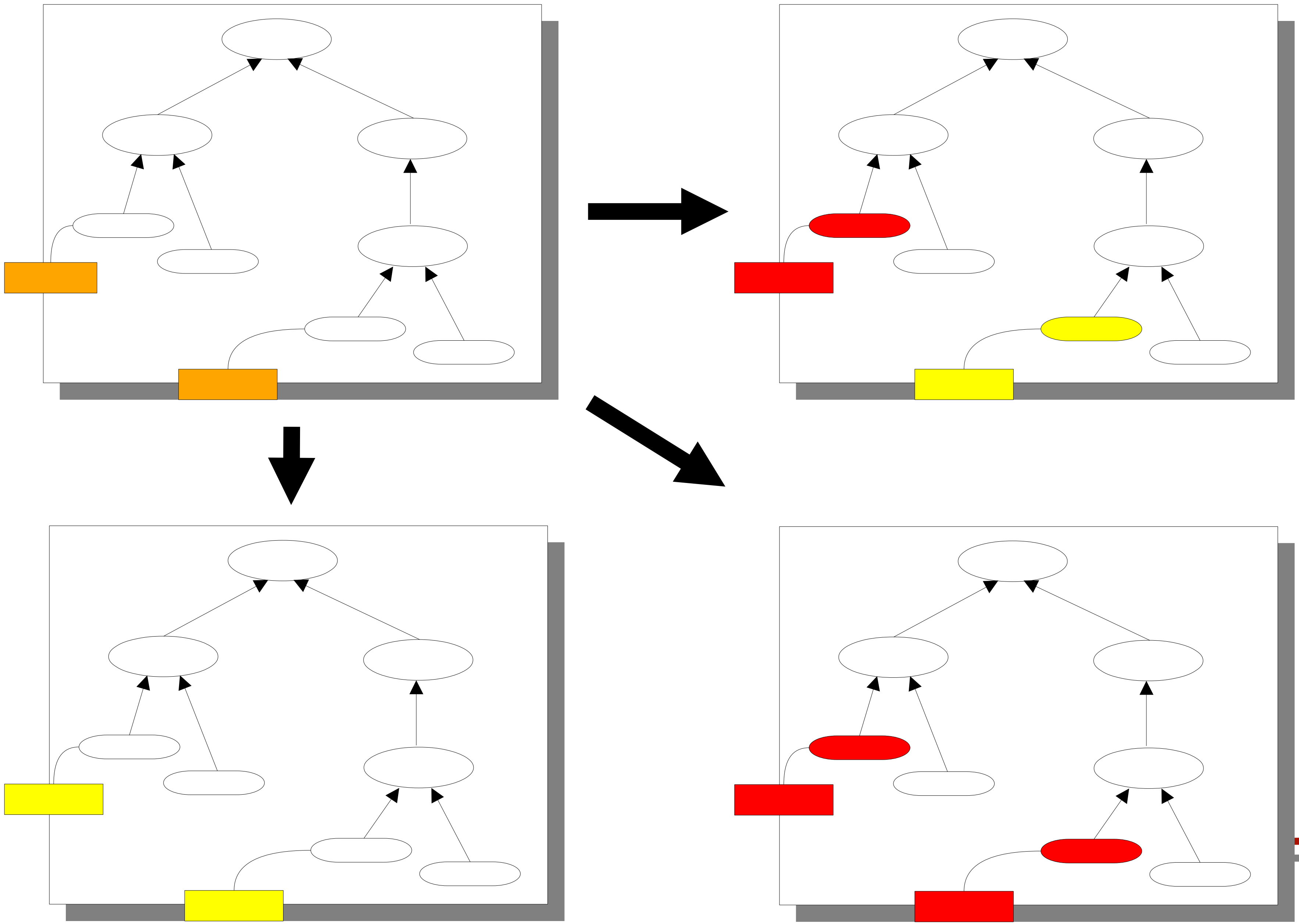
Configuration of Packages with Embedded Composition Programs

- Composition classes itself can be hooks of packages
- Then, in system configuration, they can be re-bound (stage 1)
    - This is metacomposition, production of composition programs
- When the configured composition classes are executed (stage 2)
    - They configure the system differently

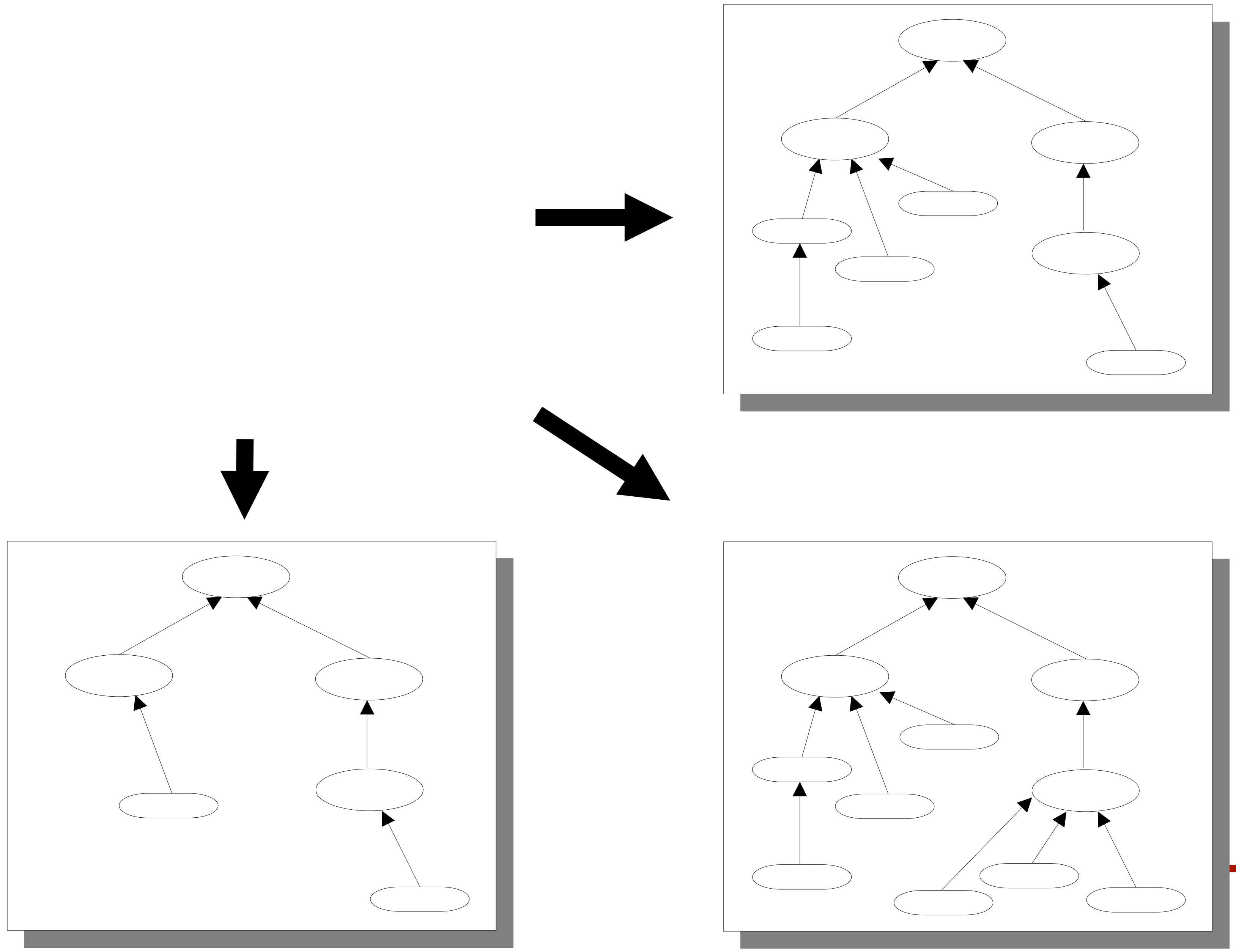
Package with Composition Class Hooks



Stage 1: Metacomposition: Binding Composition Programs in Different Variants



Stage 2: Execution of Composition Programs



The End