17-708 SOFTWARE PRODUCT LINES: CONCEPTS AND IMPLEMENTATION

FEATURE-ORIENTED PROGRAMMING

CHRISTIAN KAESTNER
CARNEGIE MELLON UNIVERSITY
INSTITUTE FOR SOFTWARE RESEARCH

PROJECT UPDATE

READING ASSIGNMENT NOV 4

Apel, S., Batory, D., Kästner, C., & Saake, G. (2013). Feature-Oriented Software Product Lines. Berlin: Springer.

remainder of Chapter 6

Adams, Bram, Wolfgang De Meuter, Herman Tromp, and Ahmed E. Hassan. "Can we refactor conditional compilation into aspects?." In Proceedings of the 8th ACM international conference on Aspect-oriented software development, pp. 243-254. ACM, 2009.

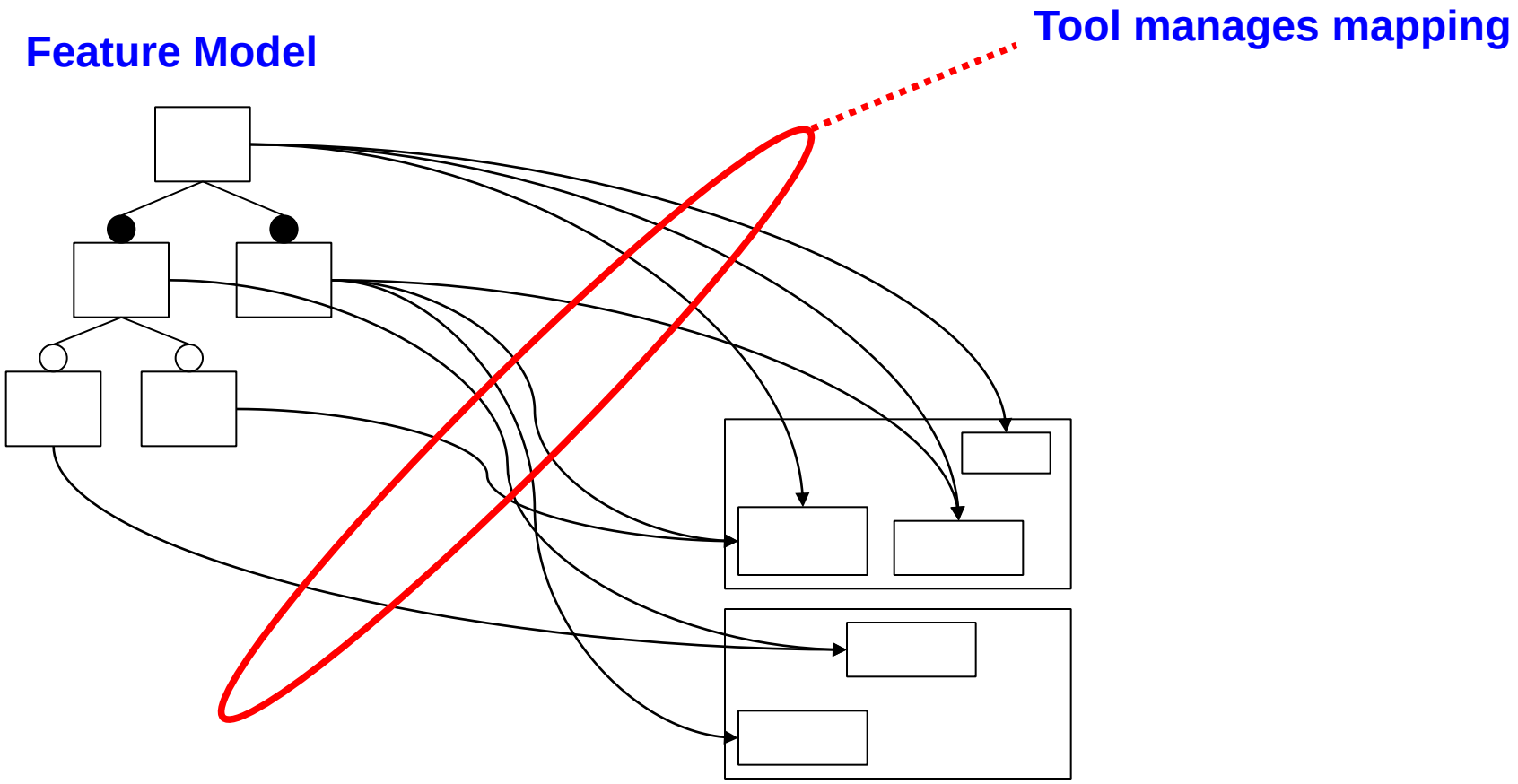
LEARNING GOALS

Understand key concepts of feature-oriented programming

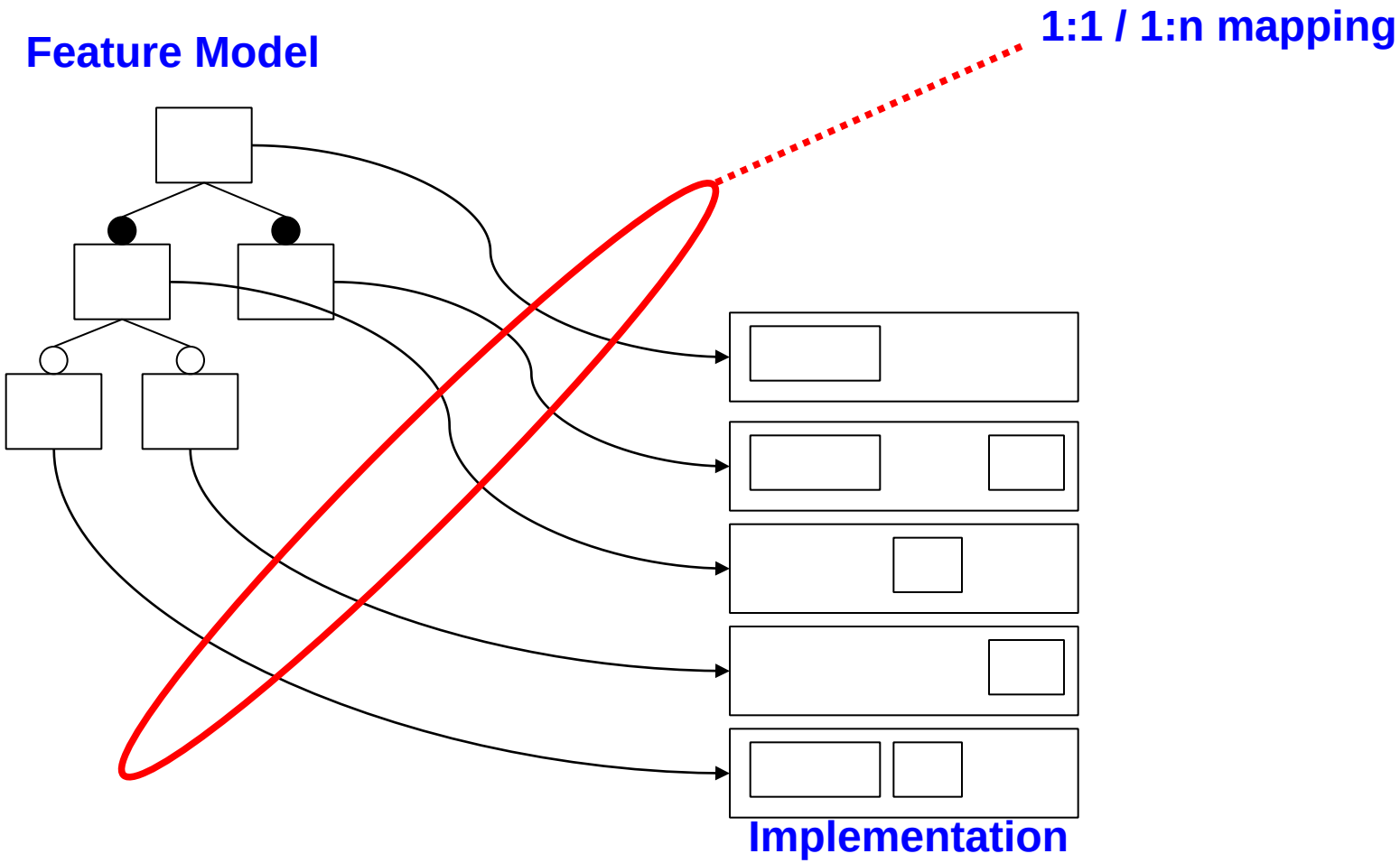
Implement variations with feature-oriented programming

Understand modularity implications of feature-oriented programming and mechanisms to mitigate issues

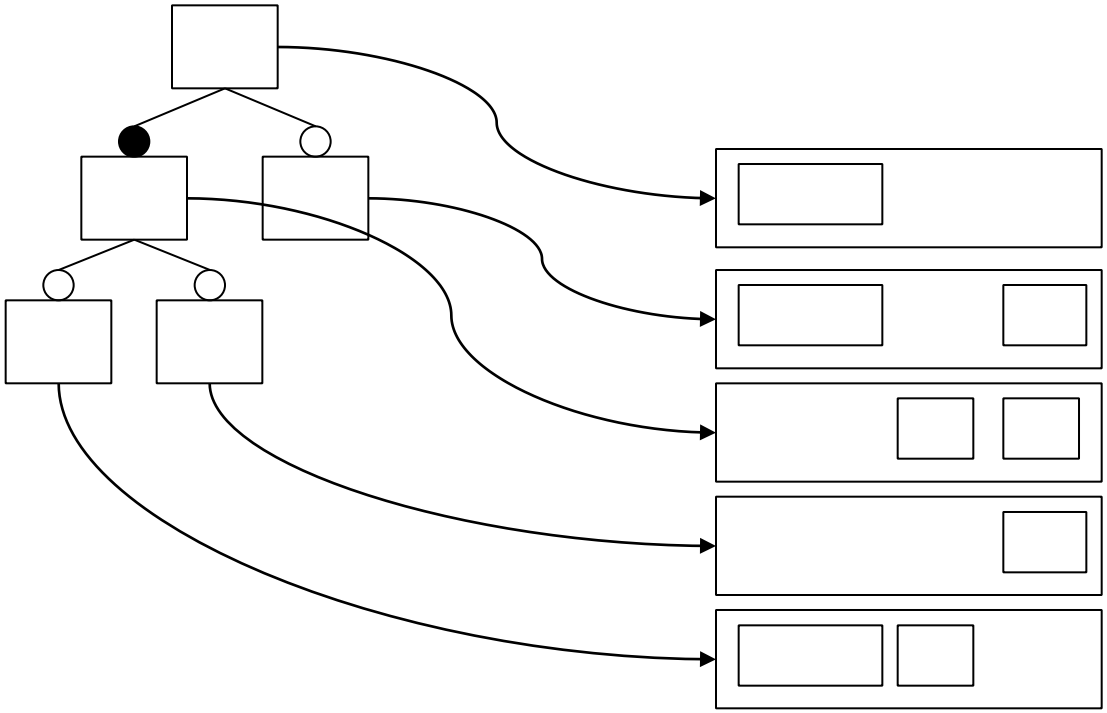
TOOL-BASED TRACEABILITY



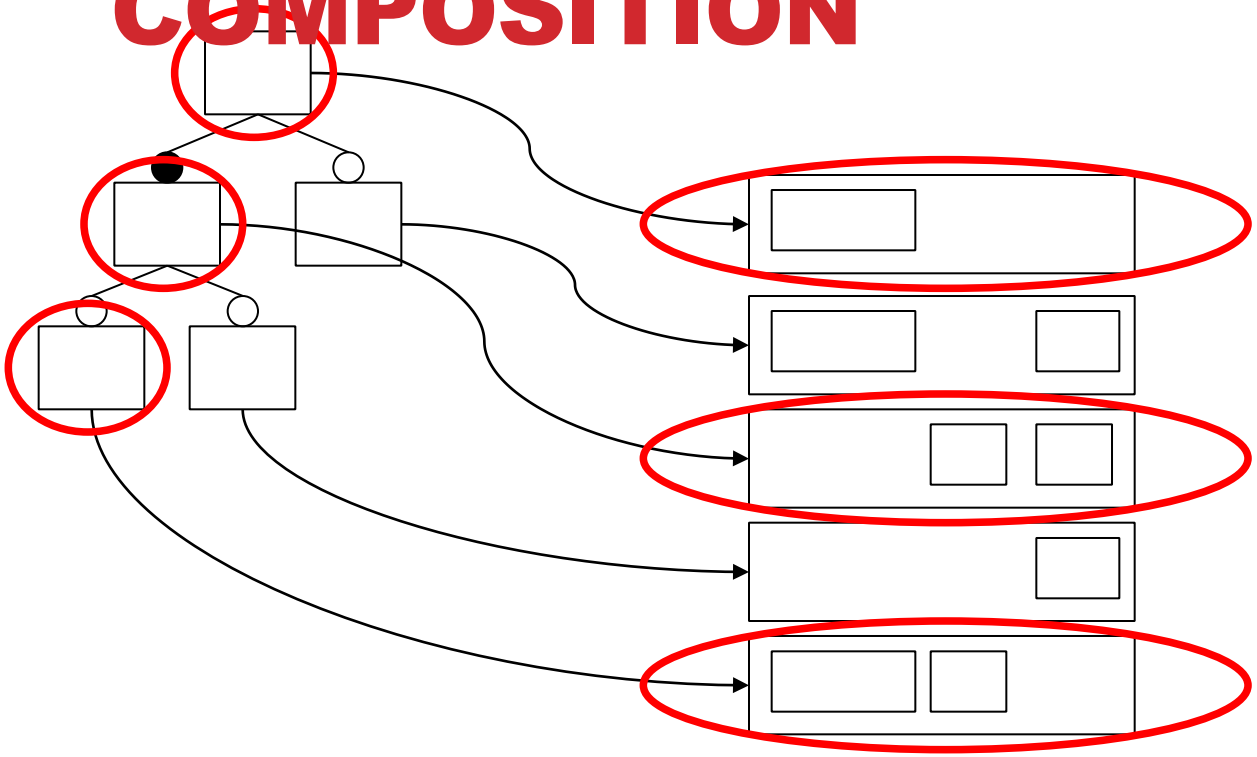
LANGUAGE-BASED TRACEABILITY



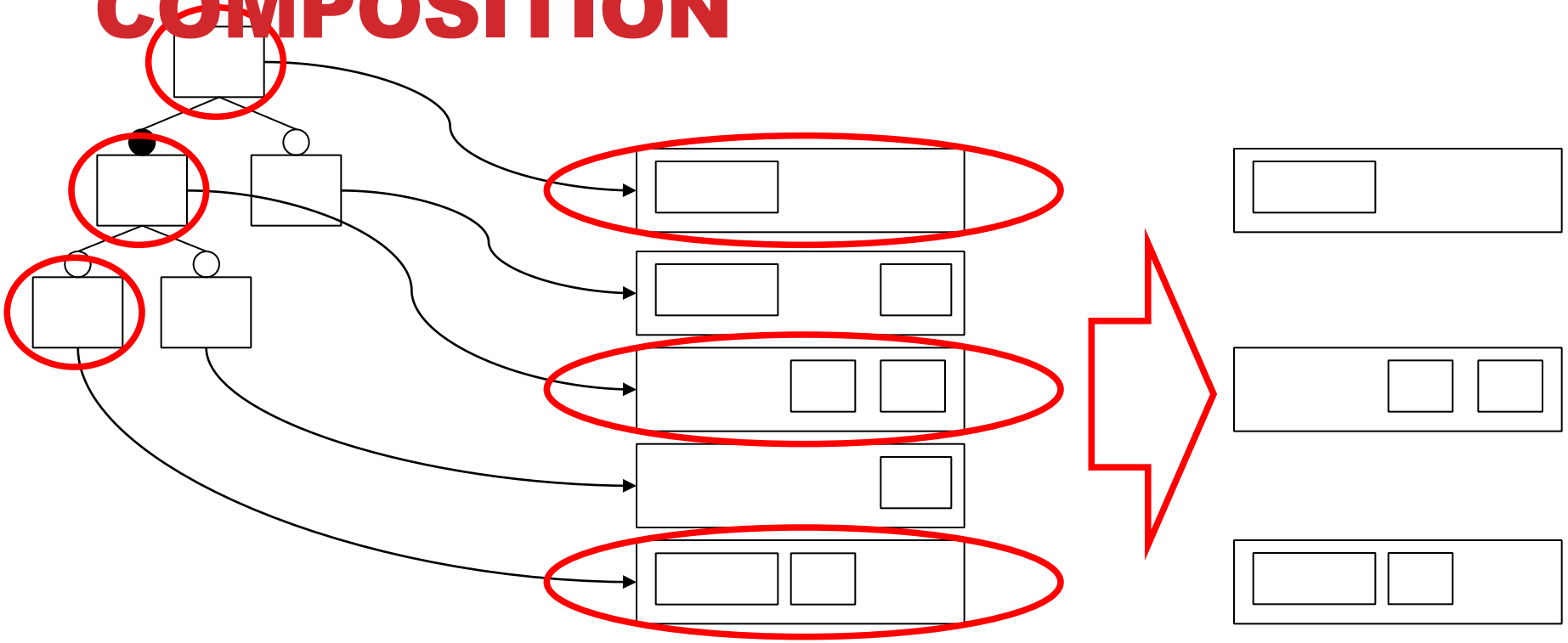
FEATURE COMPOSITION



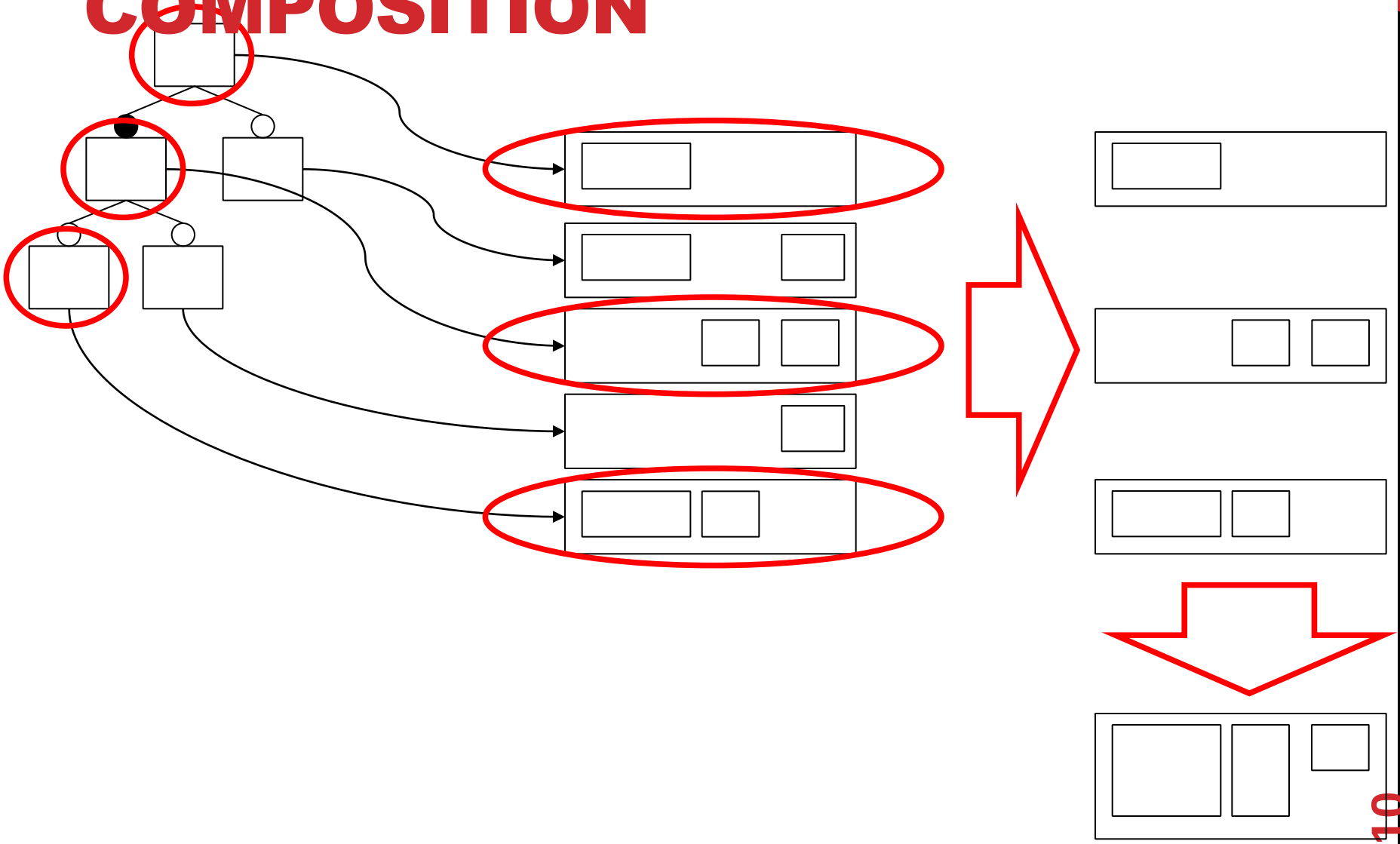
FEATURE COMPOSITION



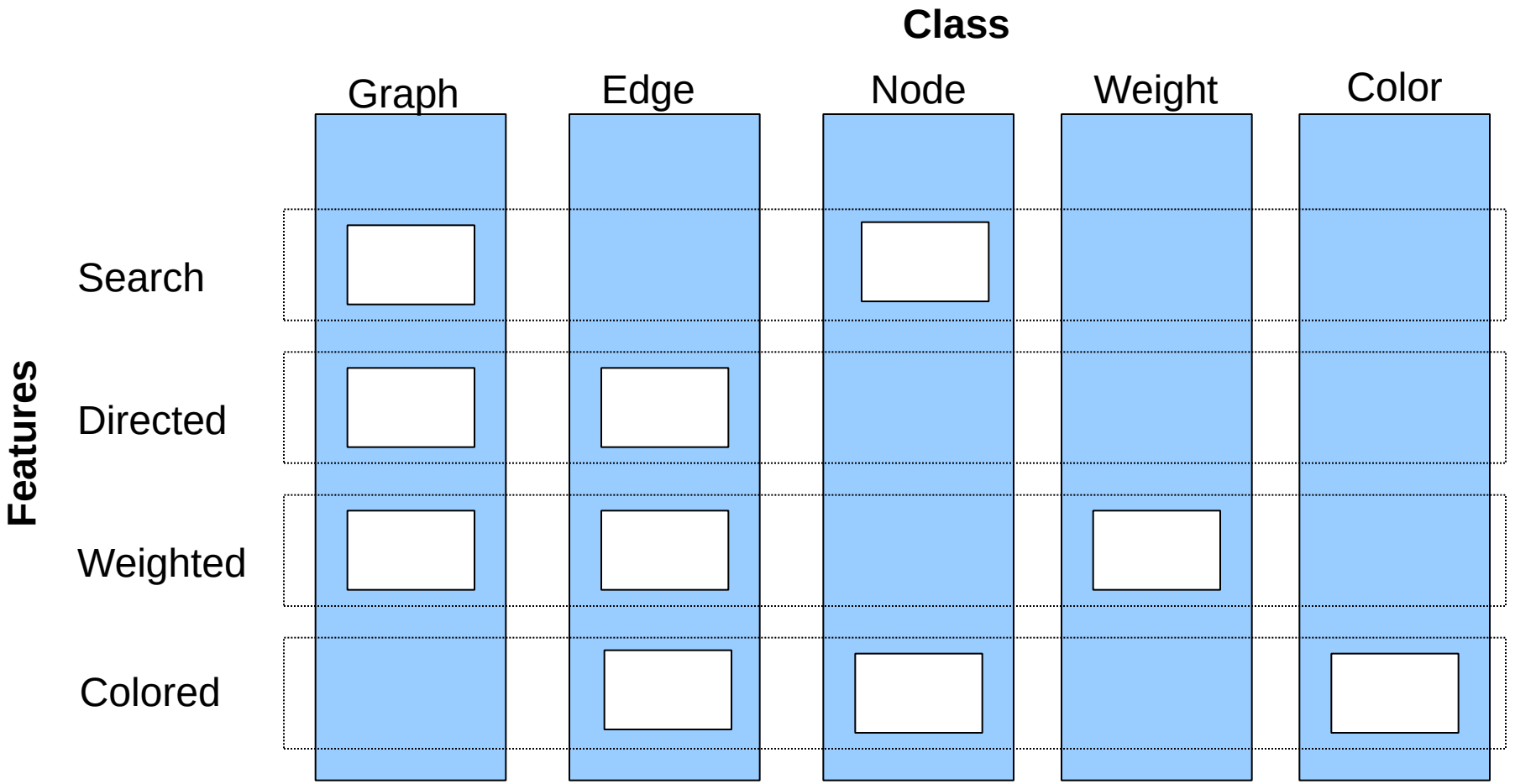
FEATURE COMPOSITION



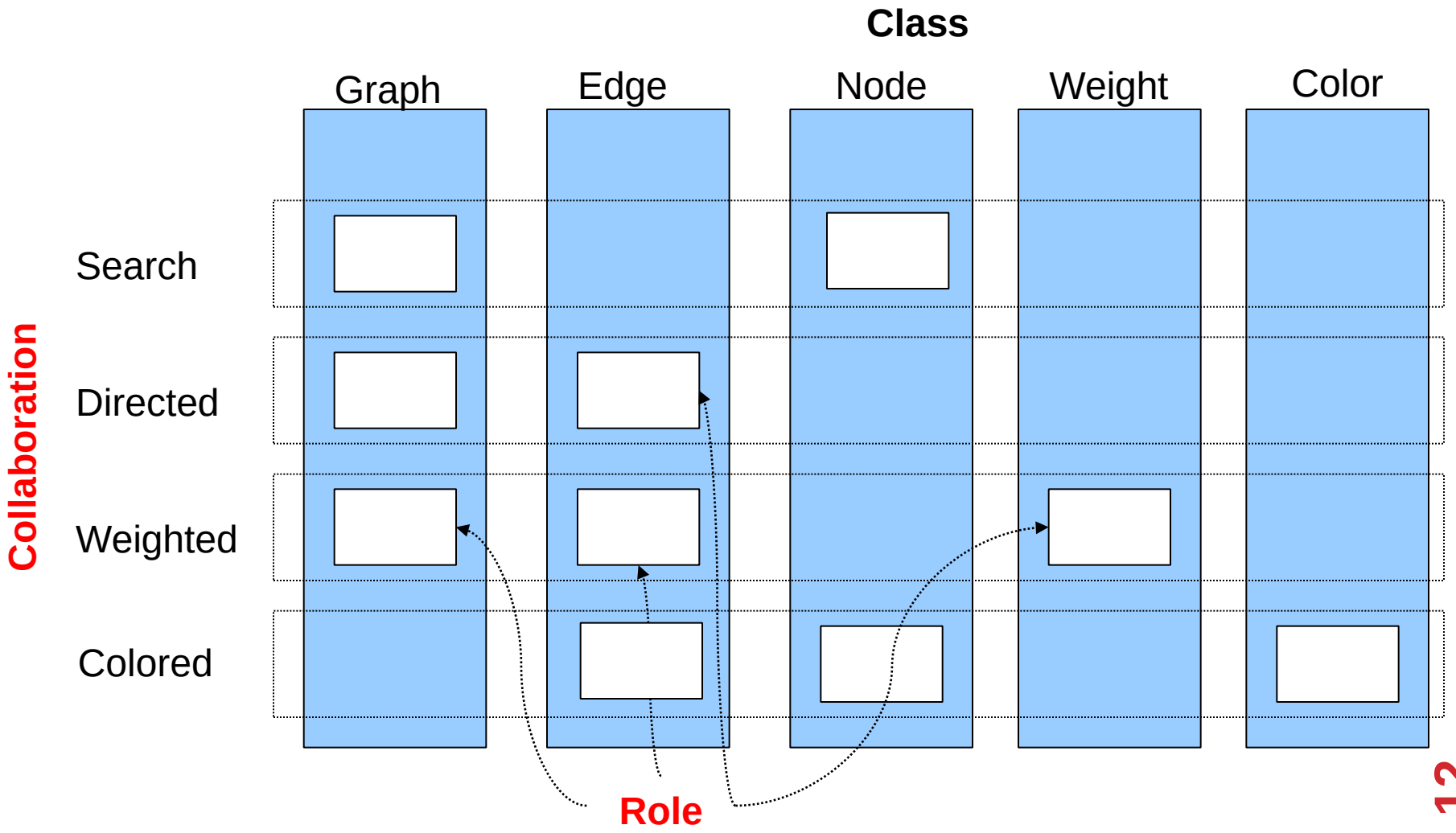
FEATURE COMPOSITION

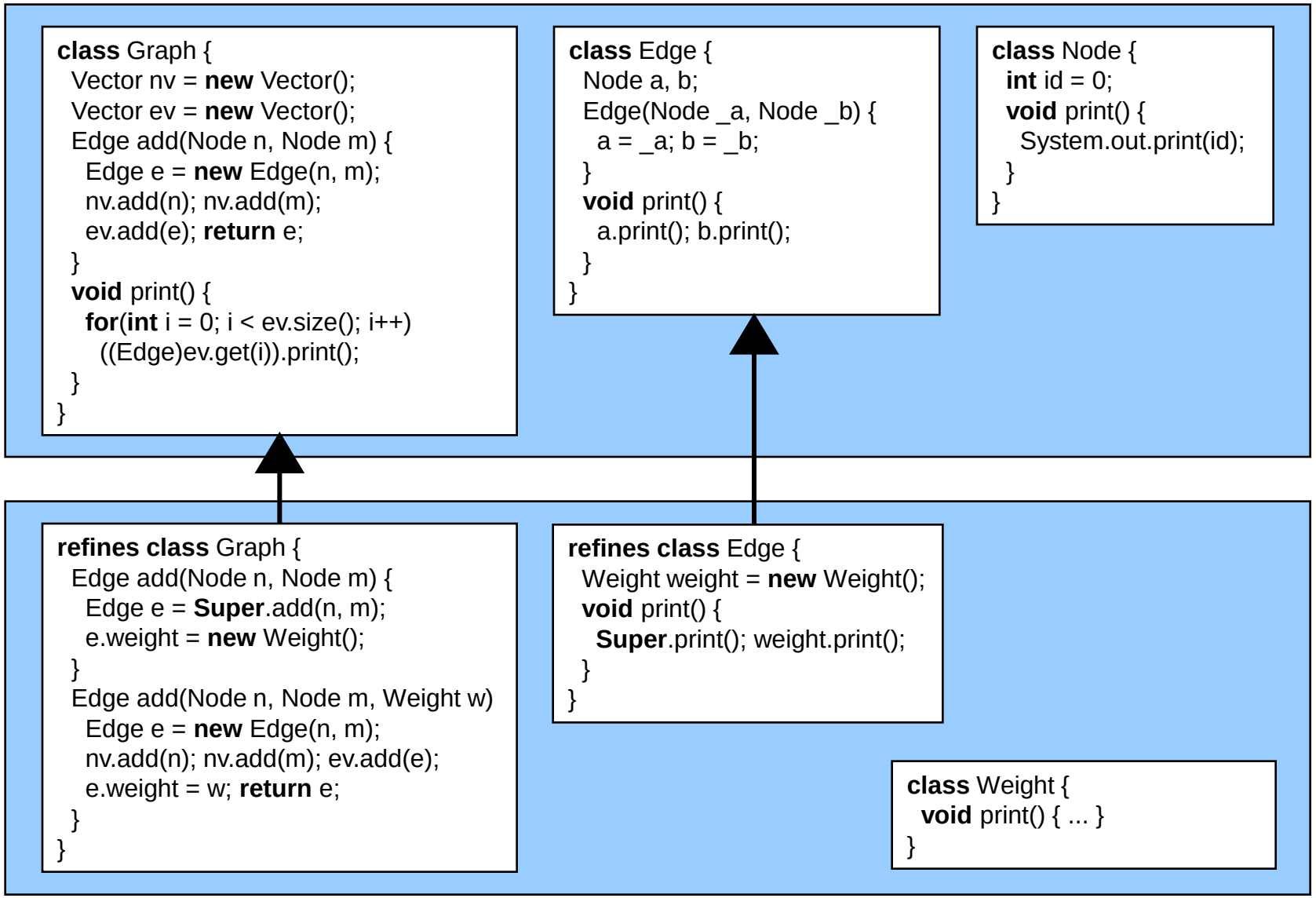


PARTIAL CLASSES

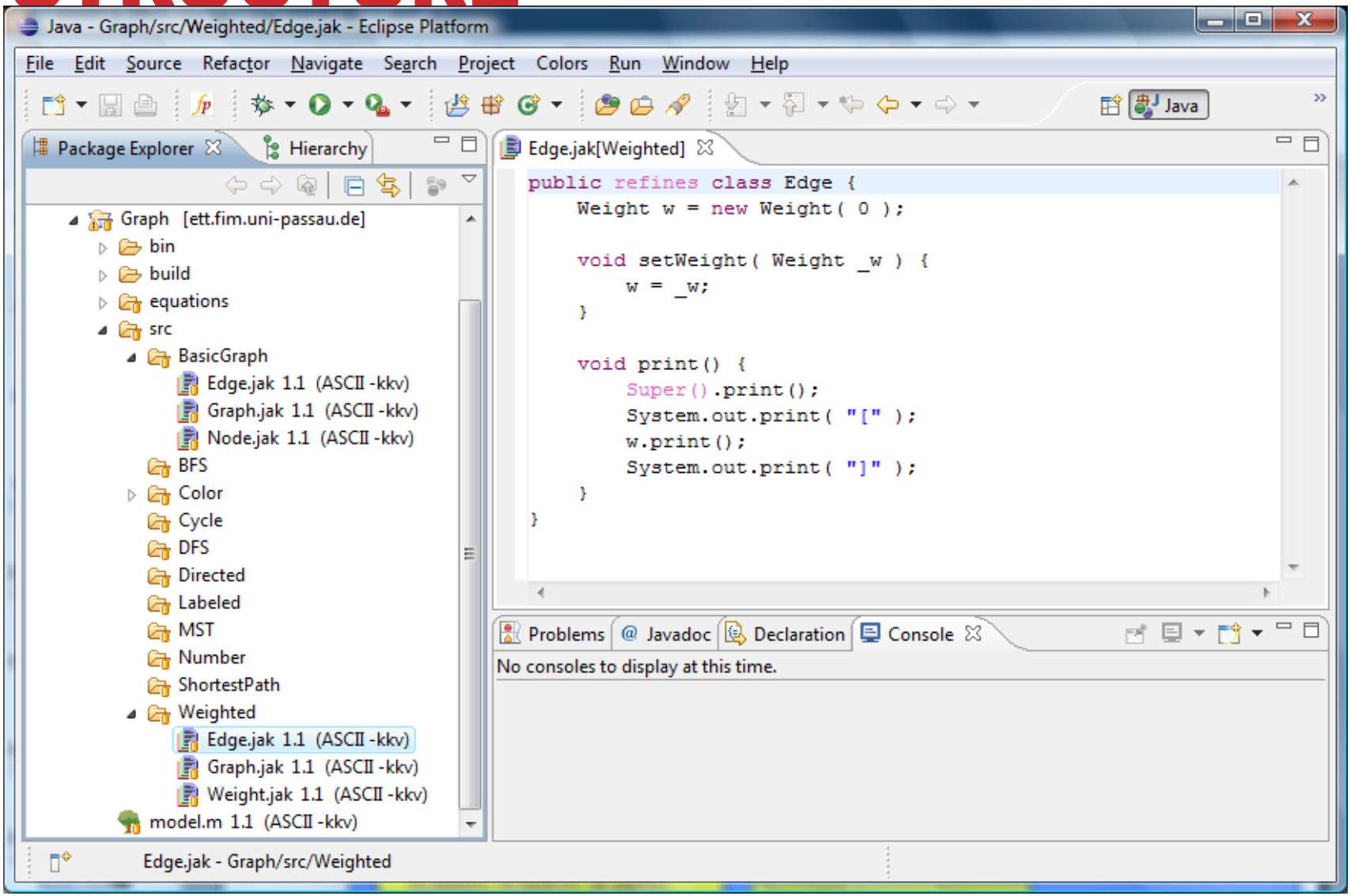


COLLABORATIONS AND ROLES

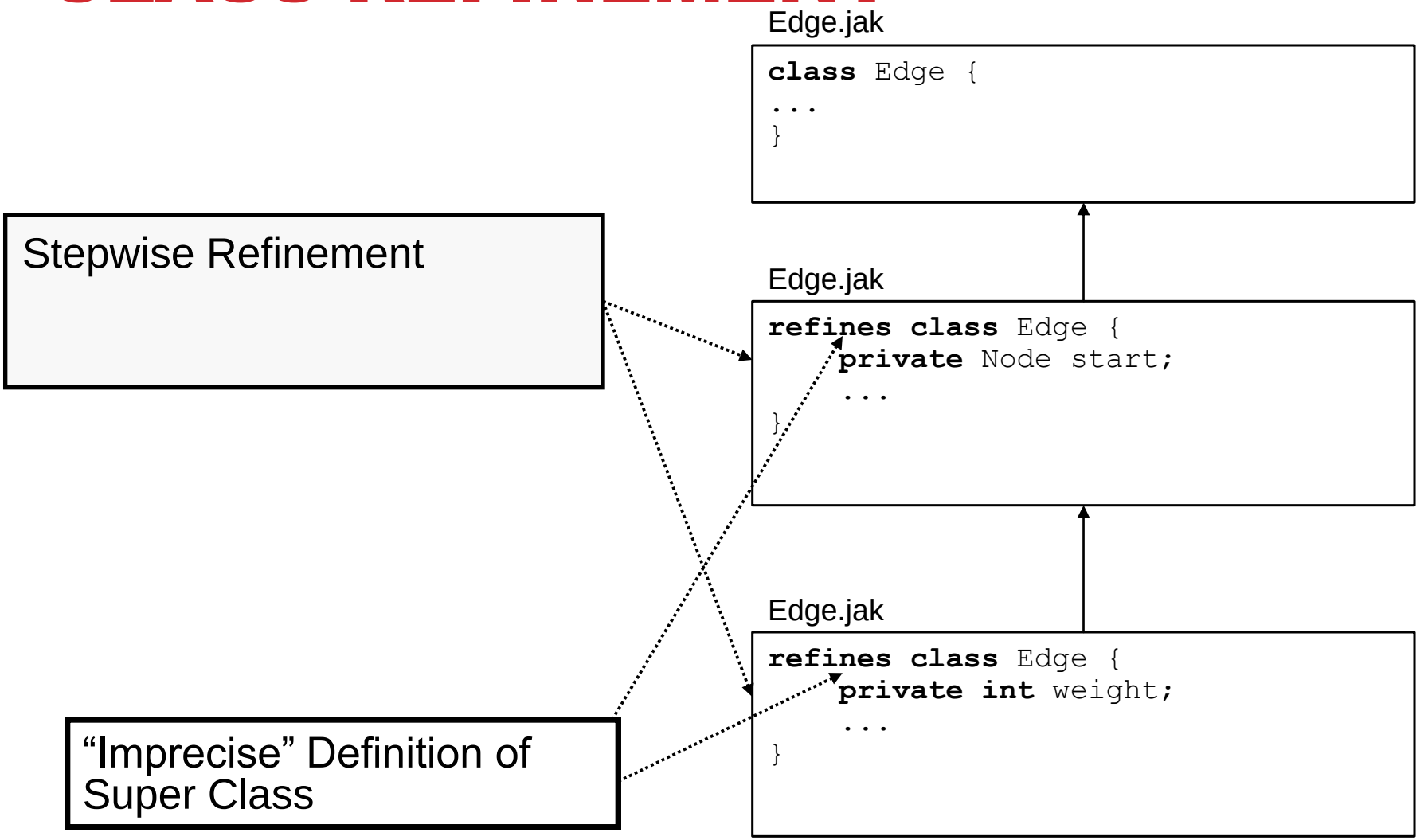



class Graph {
Vector nv = new Vector();
Vector ev = new Vector();
Edge add(Node n, Node m) {
Edge e = new Edge(n, m);
nv.add(n); nv.add(m);
ev.add(e); return e;
}
void print() {
for(int i = 0; i < ev.size(); i++)
((Edge)ev.get(i)).print();
}
}
class Edge {
Node a, b;
Edge(Node _a, Node _b) {
a = _a; b = _b;
}
void print() {
a.print(); b.print();
}
}
class Node {
int id = 0;
void print() {
System.out.print(id);
}
}
refines class Graph {
Edge add(Node n, Node m) {
Edge e = Super.add(n, m);
e.weight = new Weight();
}
Edge add(Node n, Node m, Weight w)
Edge e = new Edge(n, m);
nv.add(n); nv.add(m); ev.add(e);
e.weight = w; return e;
}
}
refines class Edge {
Weight weight = new Weight();
void print() {
Super.print(); weight.print();
}
}
class Weight {
void print() { ... }
}

DIRECTORY STRUCTURE



CLASS REFINEMENT



METHOD REFINEMENT (AHEAD)

```
class Edge {
    void print() {
        System.out.print(
          " Edge between " + node1 +
          " and " + node2);
    }
}
```

```
refines class Edge {
    private Node start;
    void print() {
        Super().print();
        System.out.print(
            " directed from " + start);
    }
}
```

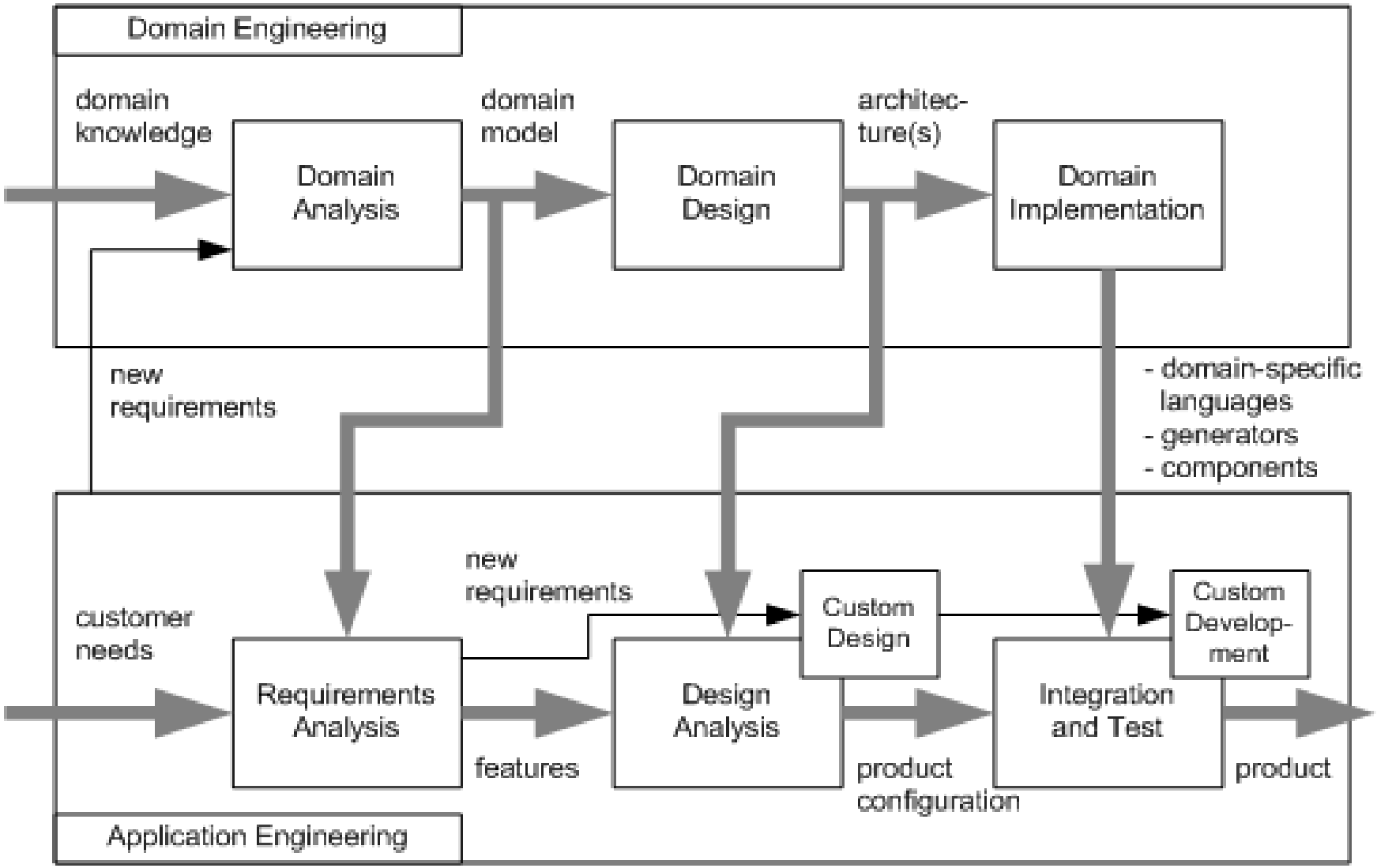
```
refines class Edge {
    private int weight;
    void print() {
        Super().print();
        System.out.print(
            " weighted with " + weigth);
    }
}
```

METHOD REFINEMENT (FEATUREHOUSE)

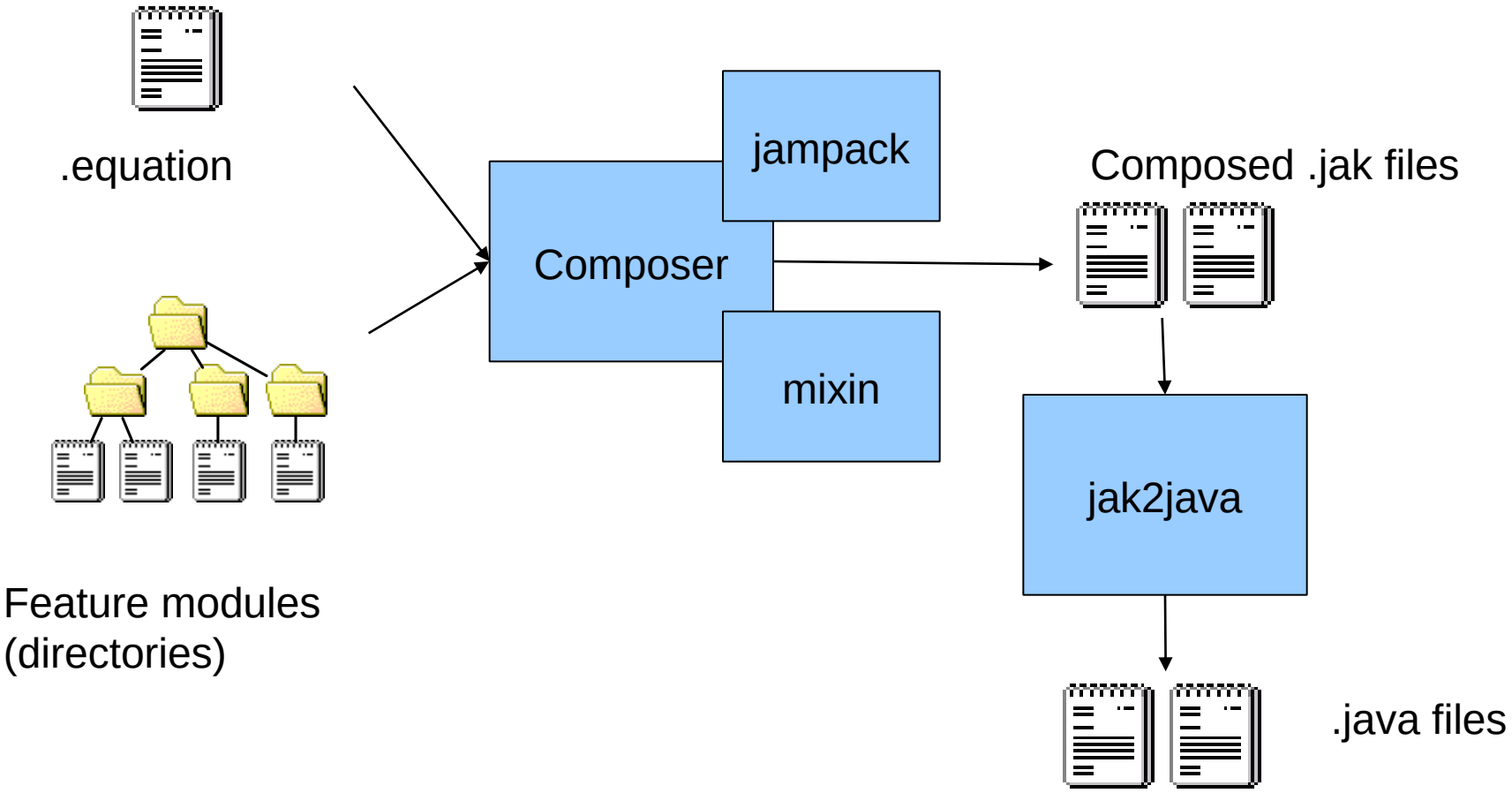
```
class Edge {
    void print() {
        System.out.print(
          " Edge between " + node1 +
           " and " + node2);
    }
}
```

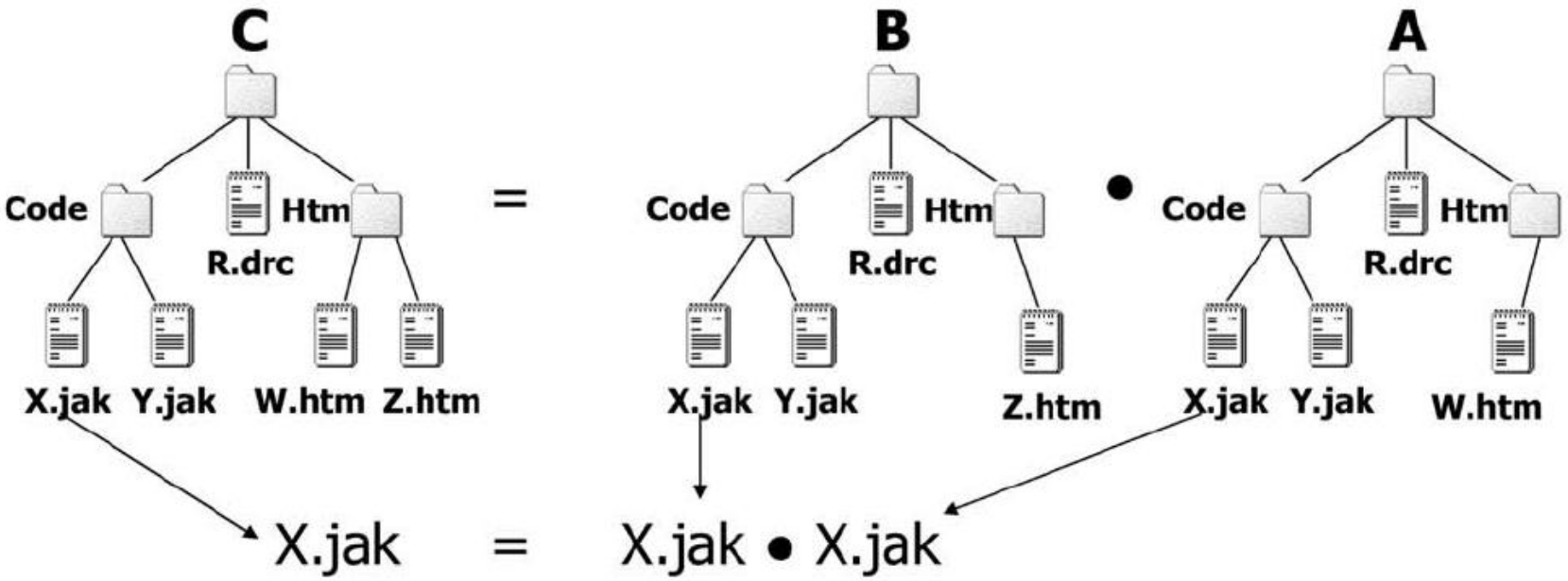
```
class Edge {
    private Node start;
    void print() {
        original();
        System.out.print(
            " directed from " + start);
    }
}
```

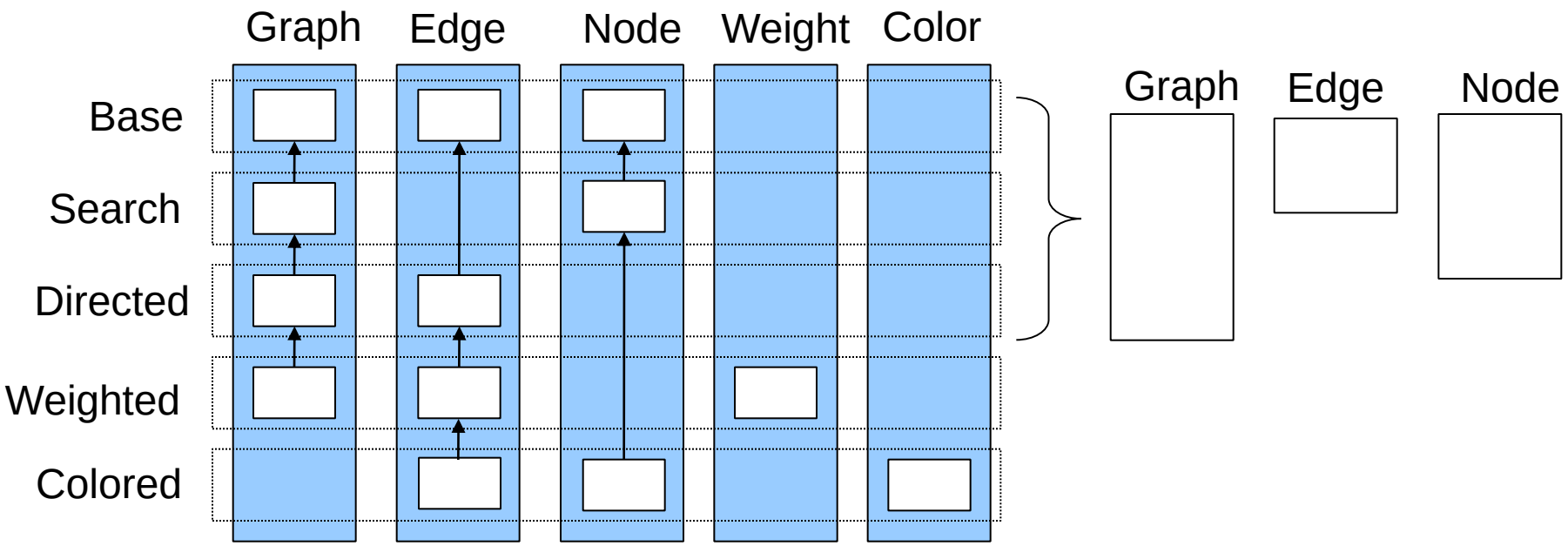
```
class Edge {
    private int weight;
    void print() {
        original();
        System.out.print(
            " weighted with " + weigth);
    }
}
```


Domain Engineering
domain knowledge
Domain Analysis
domain model
Domain Design
architec-ture(s)
Domain Implementation
new requirements
- domain-specific languages
- generators
- components
customer needs
Requirements Analysis
new requirements
Custom Design
Design Analysis
Custom Develop-ment
Integration and Test
features
product configuration
product
Application Engineering

AHEAD


.equation
Feature modules
(directories)
Composer
jampack
mixin
Composed .jak files
jak2java
.java files


C
Code
Htm
R.drc
X.jak
Y.jak
W.htm
Z.htm
=
B
Code
Htm
R.drc
X.jak
Y.jak
Z.htm
•
A
Code
Htm
R.drc
X.jak
Y.jak
W.htm
X.jak = X.jak • X.jak


Graph
Edge
Node
Weight
Color
Base
Search
Directed
Weighted
Colored
Graph
Edge
Node

TOOLS

AHEAD Tool Suite + Documentation

- Command line tools for Jak (Java 1.4 extension)
- http://www.cs.utexas.edu/users/schwartz/ATS.html

FeatureHouse

- Command-line tools for Java, C#, C, Haskell, UML, …
- http://www.fosd.de/fh

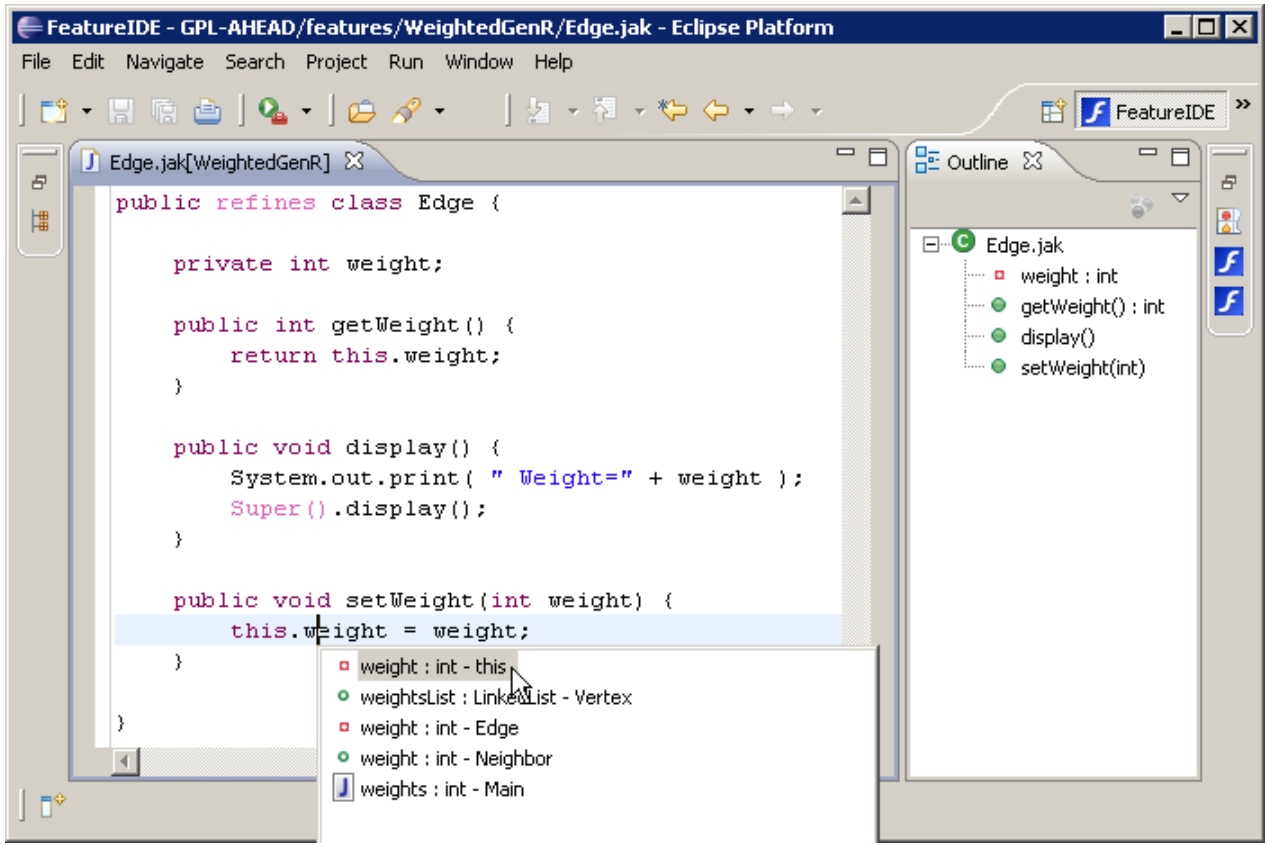
FeatureC++

- Command-line tool for C++
- http://www.fosd.de/fcpp

FeatureIDE

- Eclipse plugin for AHEAD, FeatureHouse and FeatureC++
- Automates compilation; syntax highlighting; etc
- http://www.fosd.de/featureide

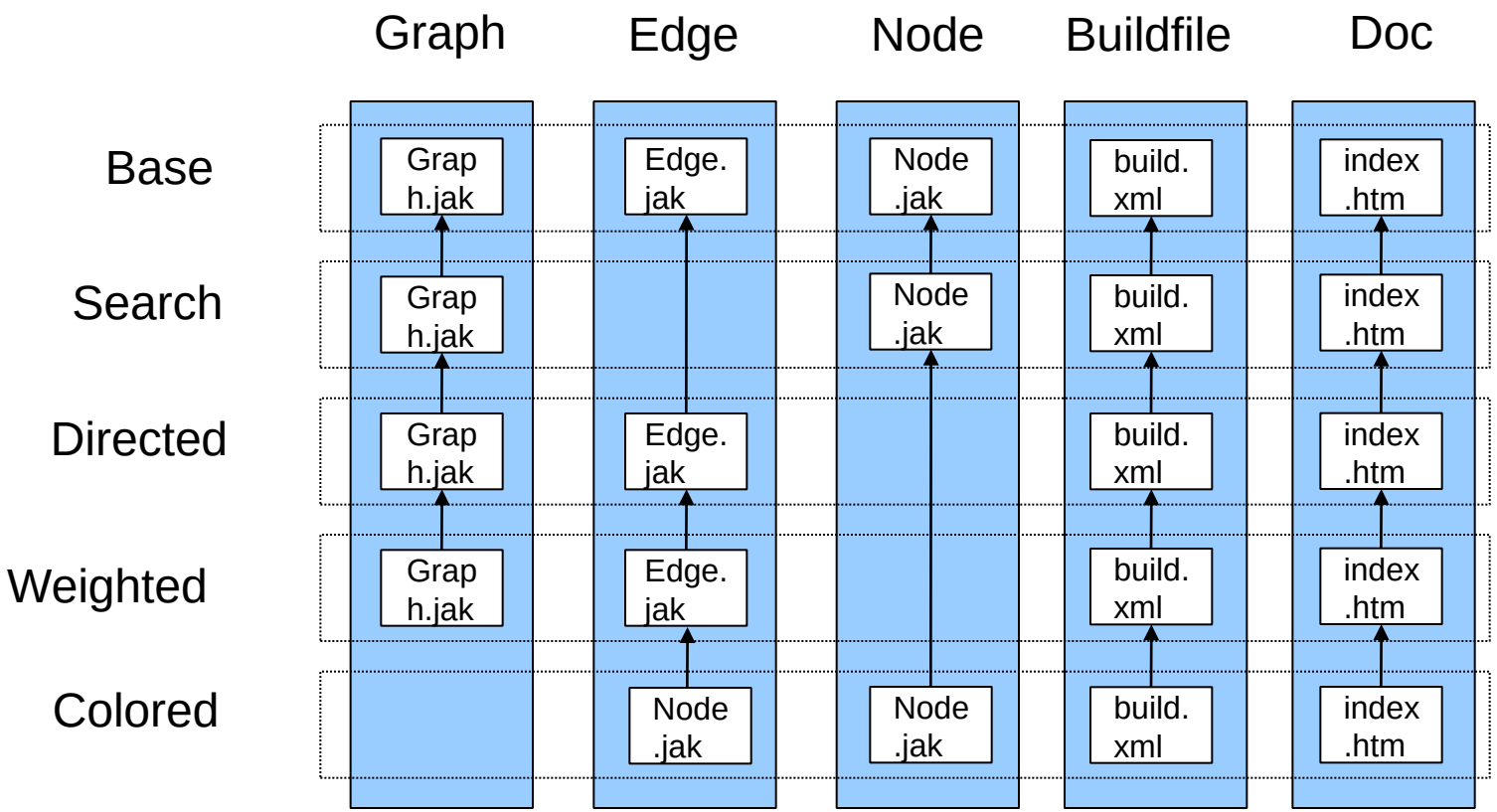
FEATUREIDE – DEMO



UNIFORMITY

Features are implemented by a diverse selection of software artifacts and any kind of software artifact can be subject of subsequent refinement.

– Don Batory


Graph
Edge
Node
Buildfile
Doc
Base
Search
Directed
Weighted
Colored
Graph.jak
Edge.jak
Node.jak
build.xml
index.htm

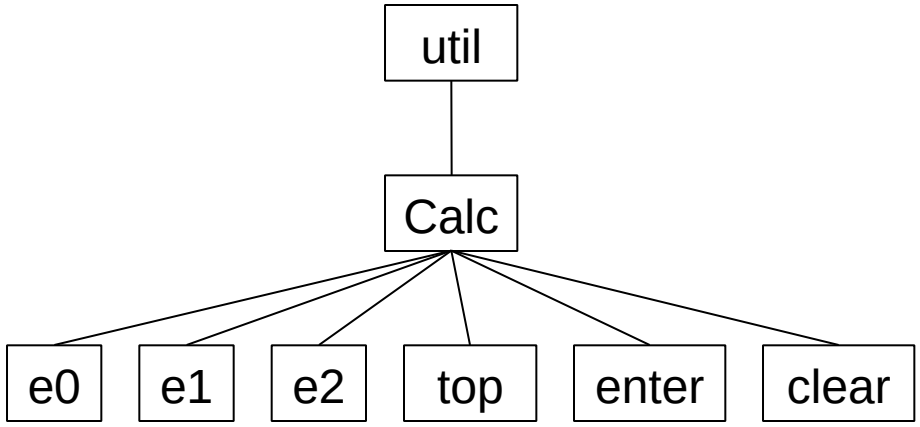
$$\bullet : F \times F \longrightarrow F$$

$$p = f_n \bullet f_{n-1} \bullet \ldots \bullet f_2 \bullet f_1$$

(assoziative, not commutative)

FEATURE-STRUCTURE TREES

```
package util;
class Calc {
  int e0 = 0, e1 = 0, e2 = 0;
  void enter(int val) {
    e2 = e1; e1 = e0; e0 = val;
  }
  void clear() {
    e0 = e1 = e2 = 0;
  }
  String top() {
    return String.valueOf(e0);
  }
}
```



SUPERIMPOSITION

```
package util;
class Calc {
  void add() {
    e0 = e1 + e0;
    e1 = e2;
  }
}
```

feature: Add

•

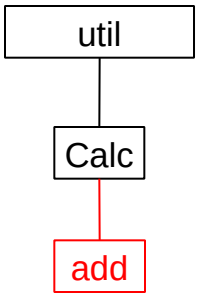
```
package util;
class Calc {
  int e0 = 0, e1 = 0,
      e2 = 0;
  void enter(int val) {
    e2 = e1; e1 = e0;
    e0 = val;
  }
  void clear() {
    e0 = e1 = e2 = 0;
  }
  String top() {
    return String.
      valueOf(e0);
  }
}
```

feature: CalcBase

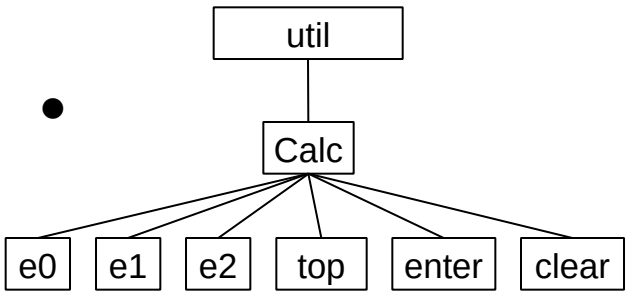
=

```
package util;
class Calc {
  int e0 = 0, e1 = 0,
      e2 = 0;
  void enter(int val) {
    e2 = e1; e1 = e0;
    e0 = val;
  }
  void clear() {
    e0 = e1 = e2 = 0;
  }
  String top() {
    //...
  }
  void add() {
    e0 = e1 + e0;
    e1 = e2;
  }
}
```

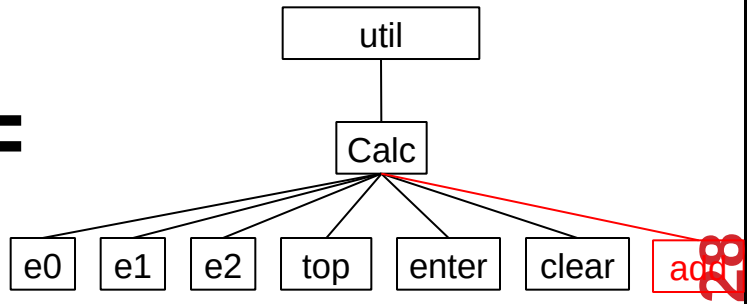
feature: CalcAdd



•



=



COMPOSITION OF TERMINAL NODES

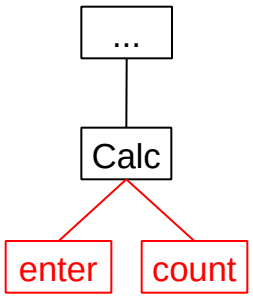
```
class Calc {
  int count = 0;
  void enter(int val) {
    original(val);
    count++;
  }
}
```

•

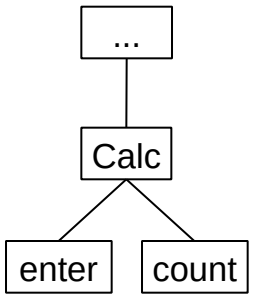
```
class Calc {
  int count;
  void enter(int val){
    e2 = e1;
    e1 = e0;
    e0 = val;
  }
}
```

=

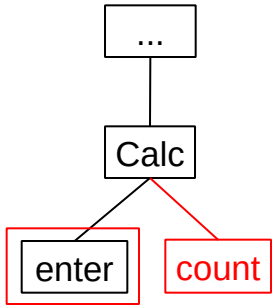
```
class Calc {
  int count = 0;
  void enter(int val) {
    e2 = e1;
    e1 = e0;
    e0 = val;
    count++;
  }
}
```



•



=



RESTRICTIONS

Hierarchical structure

Nodes need name and type

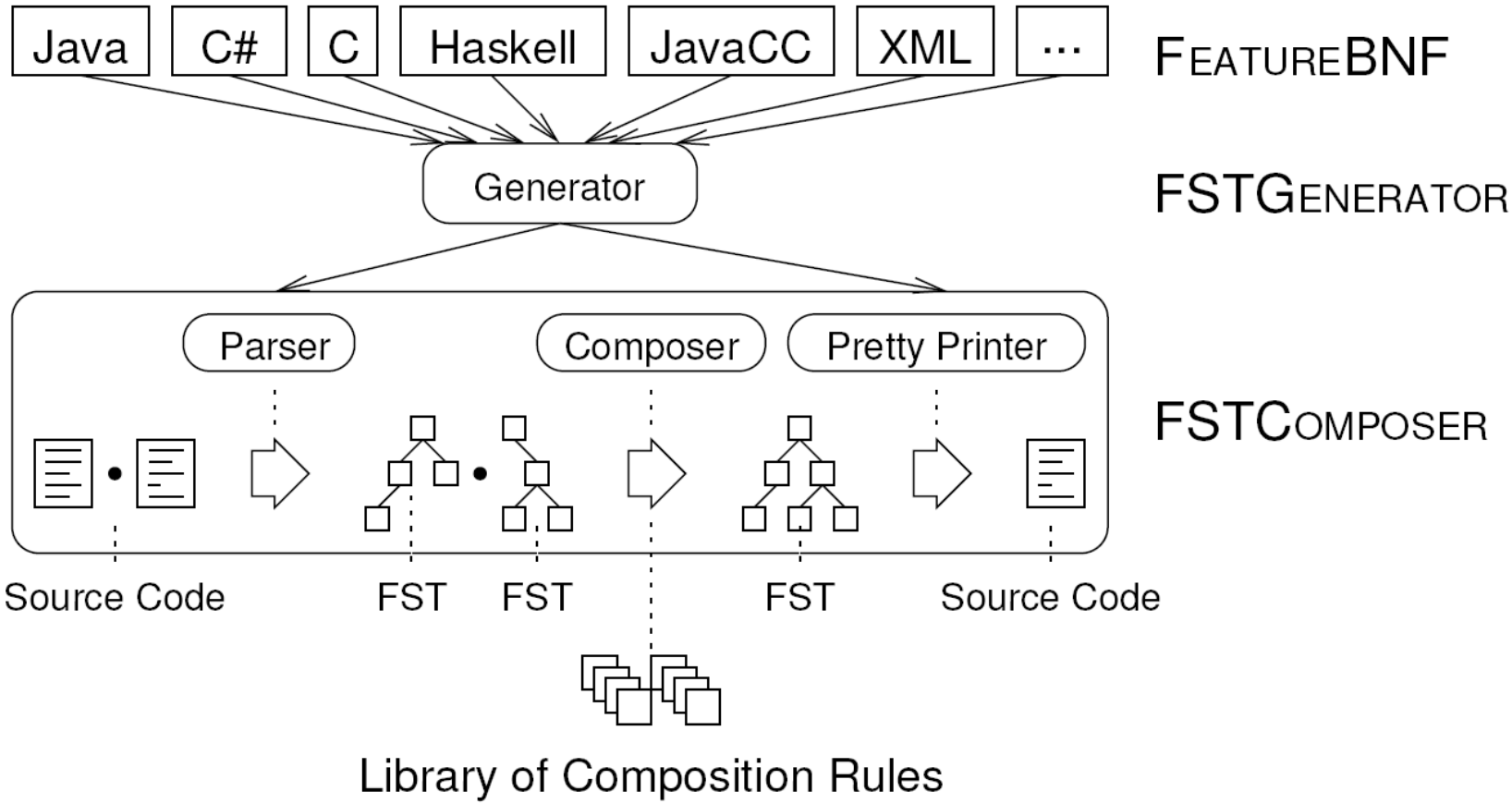
Child elements require unique name/type combinations

Composition rules required for terminal nodes

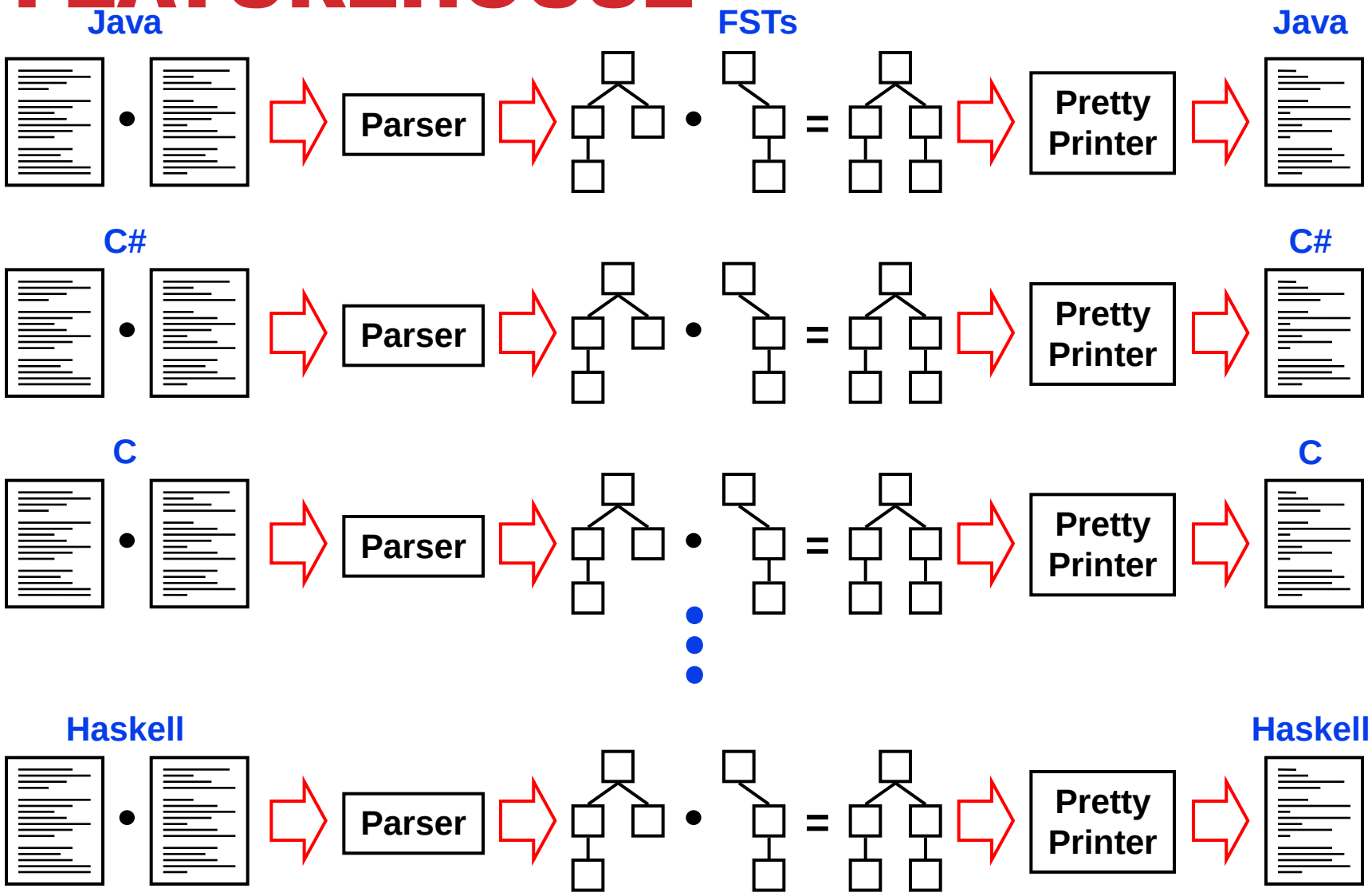
„feature-ready languages“

?

FEATUREHOUSE



FEATUREHOUSE



DISCUSSION

Modularity, Cohesion

Traceability

FURTHER READINGS

D. Batory, J. N. Sarvela, and A. Rauschmayer. Scaling Step-Wise Refinement. IEEE Transactions on Software Engineering, 30(6), 2004.

S. Apel, C. Kästner, and C. Lengauer. Language-Independent and Automated Software Composition: The FeatureHouse Experience. IEEE Transactions on Software Engineering, 39(1), 2013.

S. Apel, C. Lengauer, B. Möller, and C. Kästner. An Algebraic Foundation for Automatic Feature-Based Program Synthesis. Science of Computer Programming, 75(11), 2010.