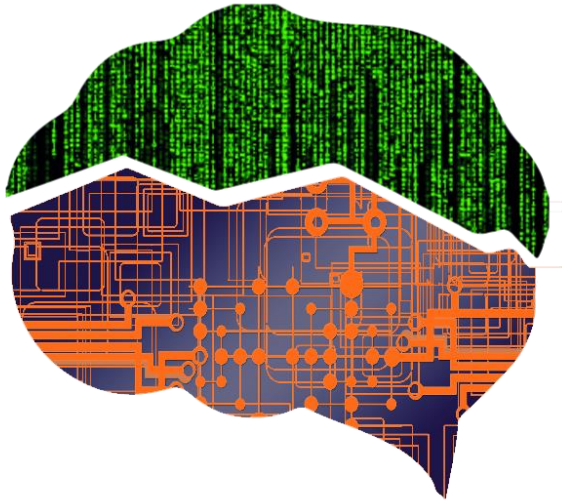
Software Product Line Engineering

Features vs. Aspects

Christian Kästner (Carnegie Mellon University)
Sven Apel (Universität Passau)
Norbert Siegmund (Bauhaus-Universität Weimar)
Gunter Saake (Universität Magdeburg)



Bauhaus-Universität Weimar

AOP vs. FOP

- AOP and FOP do not imply concrete implementation techniques
- Differ in their philosophy
  - AOP focuses on crosscutting concerns
  - FOP focusses on domain abstractions
- However, often we connect theses ideas to concrete implementation techniques
  - AOP ➔ Pointcuts & Advice, Inter-Type-Declarations
  - FOP ➔ Klassen, Refinements, Mixin/Jampack-Composition

Motivation

- AOP à la AspectJ und FOP à la Jak have similar goals
- Both can be used for product line development

- But where are the differences and commonalities?
- When to use which approach?
- Can and should we combine FOP and AOP to implement features?

Feature-Module à la Jak

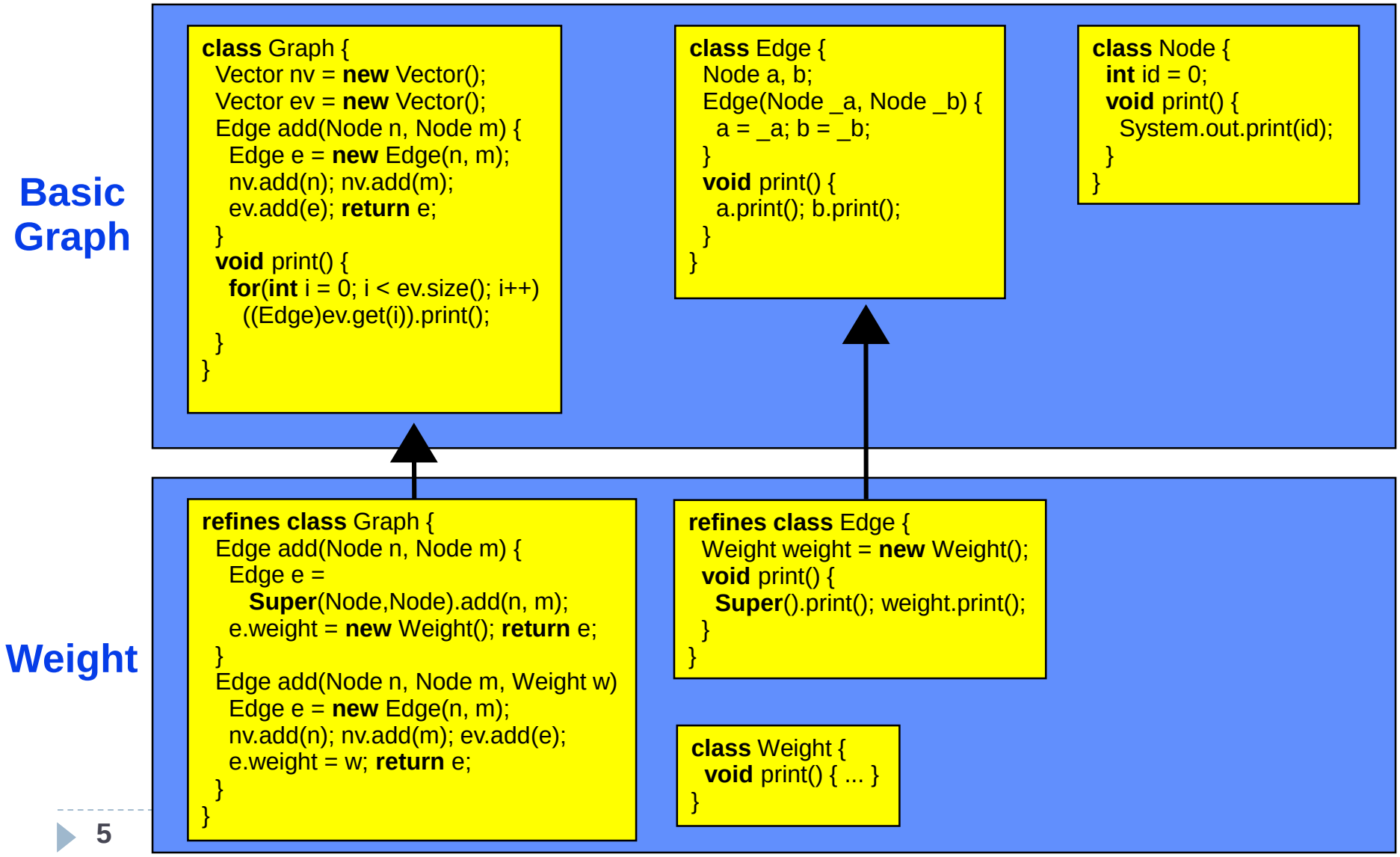
Basic Graph

```
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);
  }
}
```

Feature-Module à la Jak



Aspekte à la AspectJ

Basic Graph

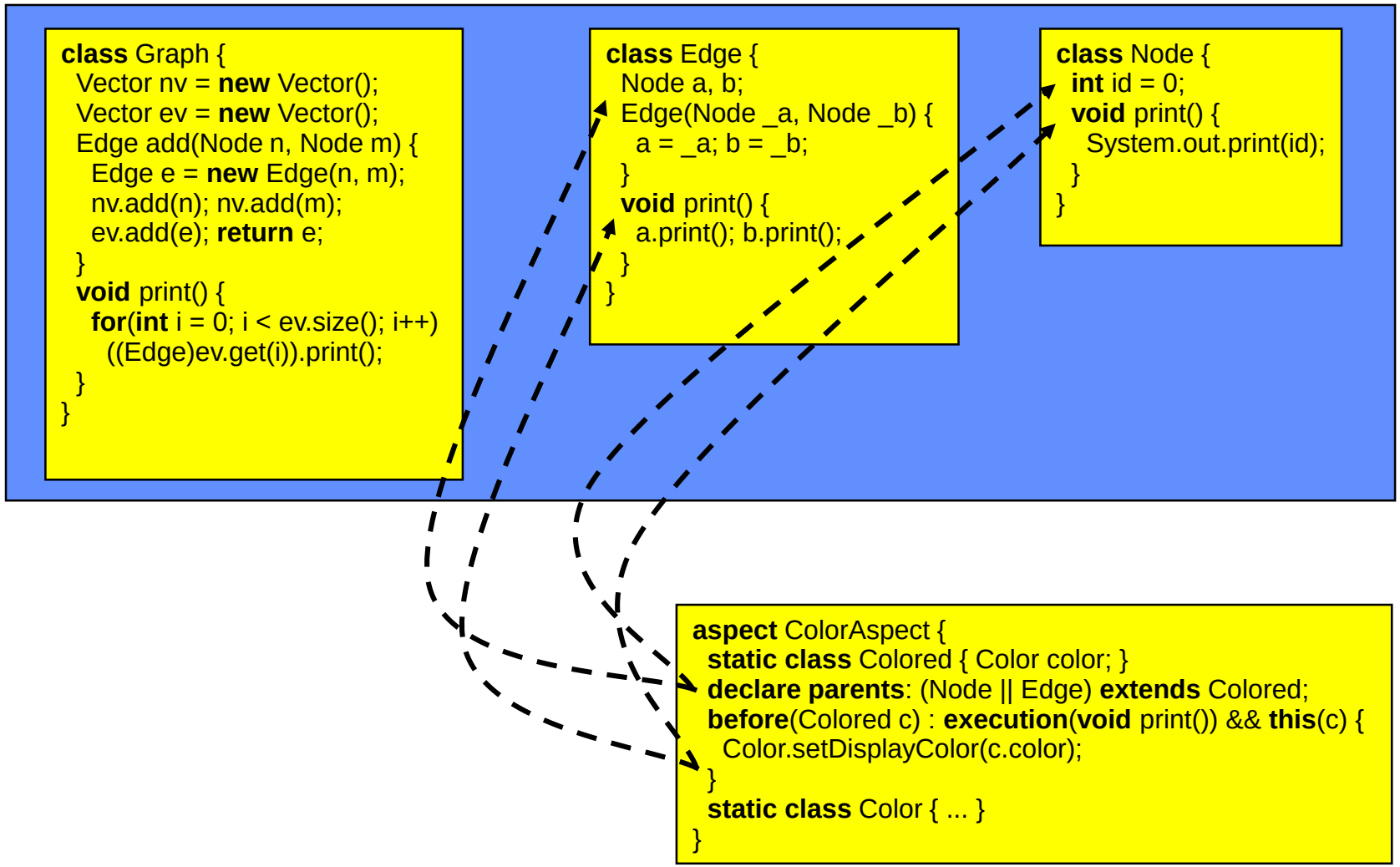
```
class Graph {
  Vector nv = new Vector();
  Vector ev = new Vector();
  Edge add(Node n, Node m) {
    Edge e = new Edge(n, m);
    nv.add(n); nv.add(m);
    ev.add(e); return e;
  }
  void print() {
    for(int i = 0; i < ev.size(); i++)
      ((Edge)ev.get(i)).print();
  }
}
```

```
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);
  }
}
```

Aspekte à la AspectJ

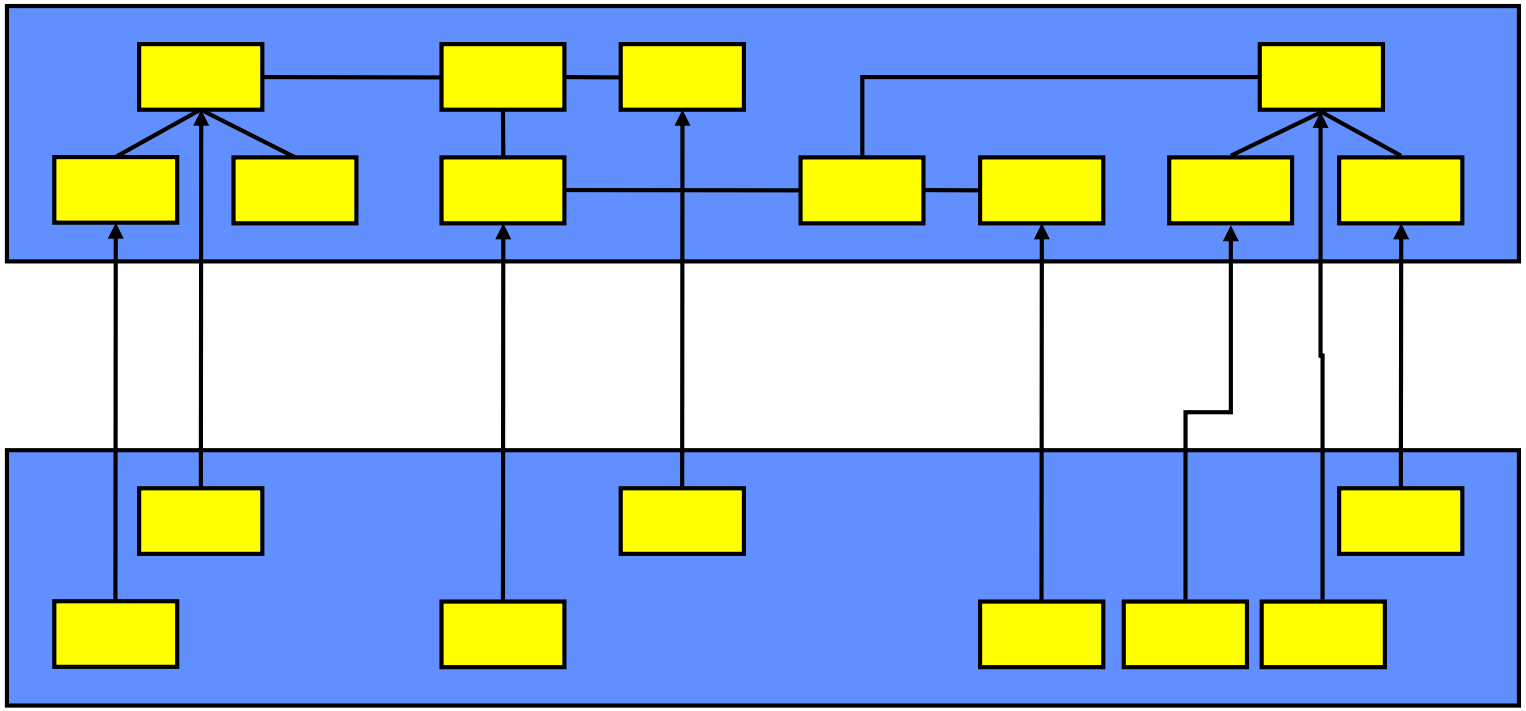
Basic Graph



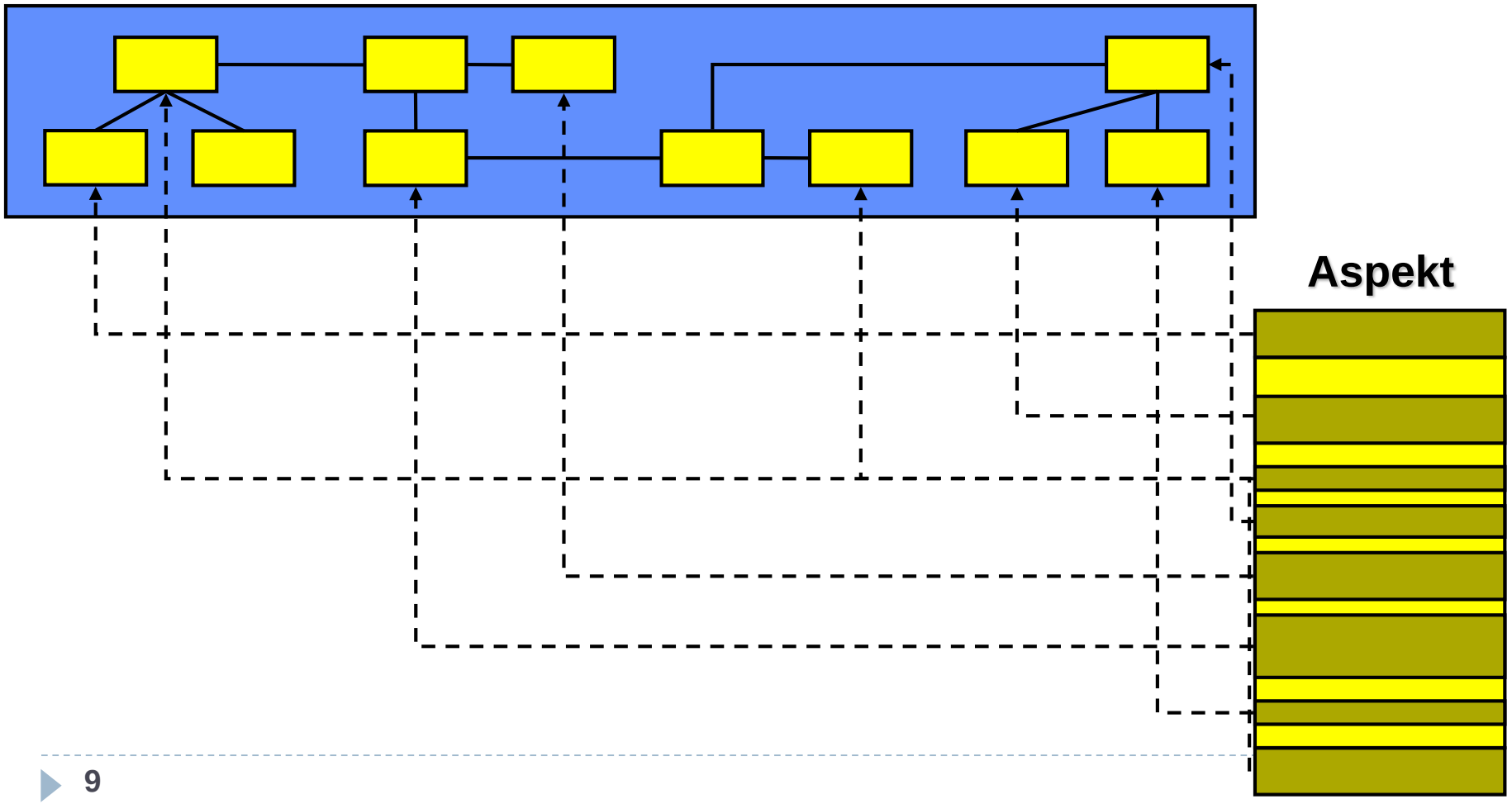
Color

```
aspect ColorAspect {
  static class Colored { Color color; }
  declare parents: (Node || Edge) extends Colored;
  before(Colored c) : execution(void print()) && this(c) {
    Color.setDisplayColor(c.color);
  }
  static class Color { ... }
}
```

AOP vs. FOP



AOP vs. FOP



Terminology

	Analyse, Design	Implementierung
AOP	▯ Aspect-oriented modeling ▯ Early aspects ▯ Aspects as functions	▯ Aspect = Class, Advice, Inter-Type-Declarationen ▯ AspectJ, AspectC++, Eos
FOP	▯ Feature modeling ▯ Feature expressions ▯ domain-specific optimizations	▯ Feature module = Classes, Refinements, Mixin/Jampack-Composition ▯ Jak, FeatureC++, FeatureHouse, Classbox/J, Jiazzi, ObjectTeams/Java

Heterogeneous vs. Homogeneous Extensions

- Heterogeneous: different code at different places
- Homogeneous : same code at different places

```
class Graph { …
   Edge add(Node n, Node m) {
   Edge e = new Edge(n, m);
   nv.add(n); nv.add(m); ev.add(e);
   e.weight = new Weight(); return e;
  }
  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;
  } …
}
```

```
class Edge { …
  Weight weight = new Weight();
  Edge(Node _a, Node _b) { a = _a; b = _b; }
   void print() {
    a.print(); b.print(); weight.print();
  }
}
```

```
class Node {
  int id = 0;
  Color color = new Color();
  void print() {
    Color.setDisplayColor(color);
    System.out.print(id);
  }
}
```

```
class Edge {
  Node a, b;
  Color color = new Color();
  Edge(Node _a, Node _b) { a = _a; b = _b; }
  void print() {
    Color.setDisplayColor(color);
    a.print(); b.print();
  }
}
```

Static vs. Dynamic Extension

- Static:
  change the statistic structure (syntactic structure)
- Dynamic:
  change the behavior (event and action)

```
class Node {
  int id = 0;
  Color color = new Color();
  void print() {
    System.out.print(id);
  }
}
```

```
class Node {
  int id = 0;
  void print() {
    Color.setDisplayColor(color);
    System.out.print(id);
  }
}
```

Simple and Extended Dynamic Extensions

- **Simple** dynamic extensions
  - Extension of method executions
  - Without constraints at runtime
  - No access to the context of an event
    - Except from arguments, return values, and current object
- **Complex** dynamic extensions
  - All kinds of events
  - Constraints at runtime (control flow)
  - Access to the dynamic context

Simple dynamic extensions are method extensions via overriding!

Example for Simple Dynamic Extensions

```
class Edge {
  int weight = 0;
  void setWeight(int w) { weight = w; }
  int getWeight() { return weight; }
}
```

Jak

```
refines class Edge {
  void setWeight(int w) {
    Super(int).setWeight(2*w);
  }

  int getWeight() {
    return Super().getWeight()/2;
  }
}
```

AspectJ

```
aspect DoubleWeight {
  void around(int w) : args(w) &&
    execution(void Edge.setWeight(int)) {
      proceed(w*2);
  }
  int around() :
    execution(void Edge.getWeight()) {
      return proceed()/2;
  }
}
```

Example for Complex Dynamic Extensions

```
class Node {
  void print() {...}
  ...
}
```

Jak

```
refines class Node {
  static int count = 0;
  void print() {
    if(count == 0)
      printHeader();
    count++;
    Super().print();
    count--;
  }
  void printHeader() { /* ... */ }
}
```

AspectJ

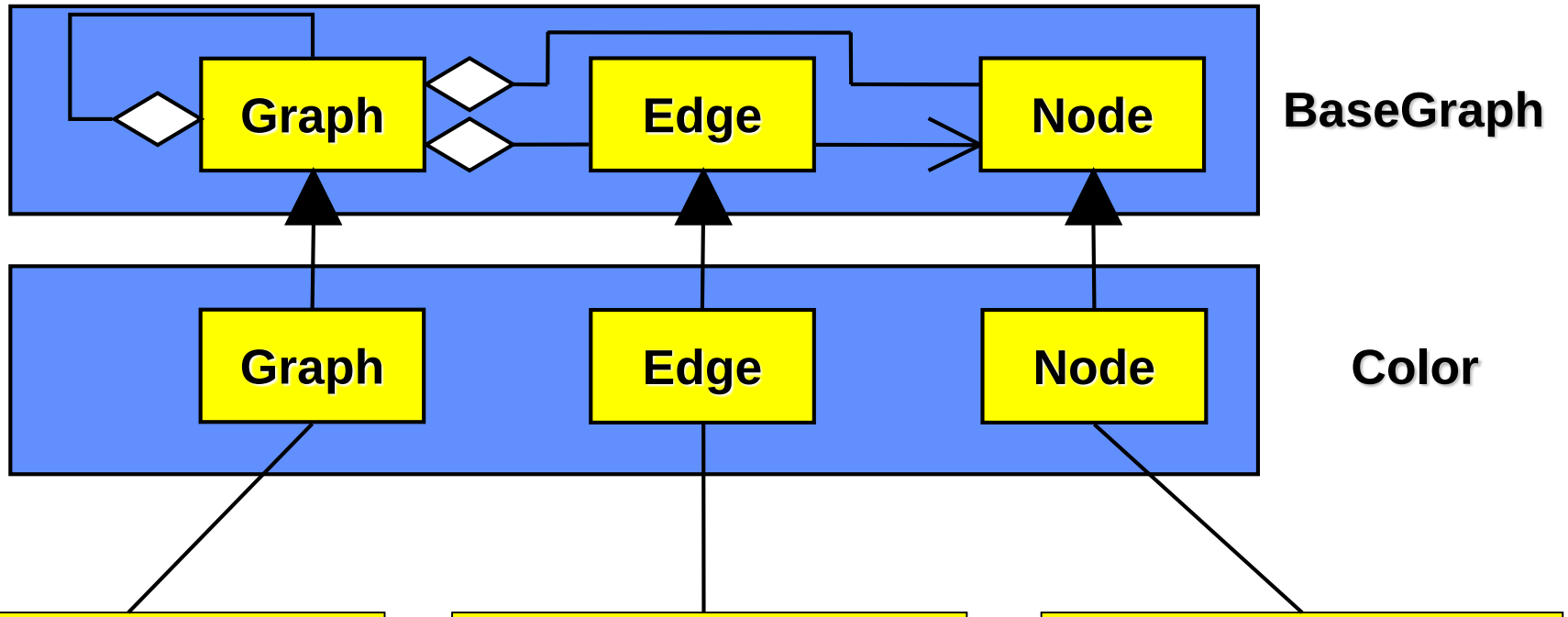
```
aspect PrintHeader {
  before() :
    execution(void print()) &&
    !cflowbelow(execution(void print())) {
      printHeader();
  }
  void printHeader() { /* … */ }
}
```

Comparison FOP and AOP

	FOP	AOP
static	***Good support*** – Attributes, methods, classes	***Limited support*** – Attributes, methods
dynamic	***Bad support*** – Simple dynamic (extensions of methods)	***Good support*** – Extended extensions
heterogeneous	***Good support*** – Refinements and collaborations	***Limitted support*** – No explicit collaborations
homogenenous	***No support*** – A refinement per join point (code replication)	***Good support*** – Wildcards and logical connection of pointcuts

Collaborations Instead of Aspects

- Homogeneous extension lead to replication



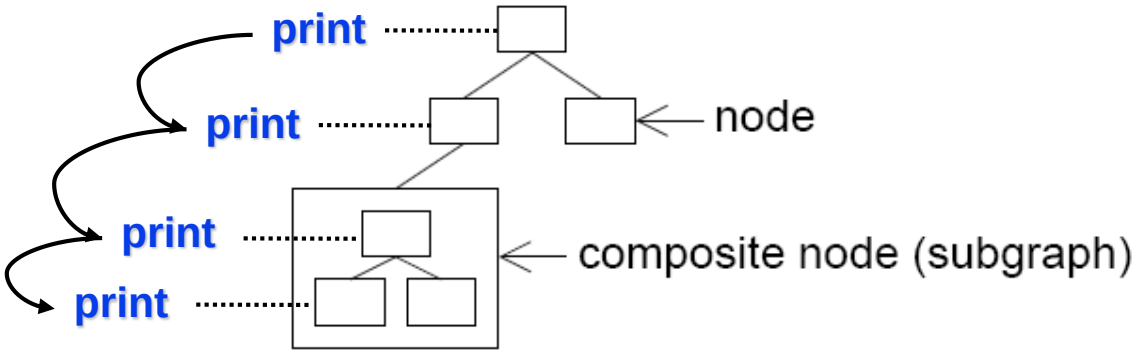

```
refines class Graph {
  Color color ;
  Color getColor () { return color; }
  void setColor (Color c) { color = c; }
}
```

```
refines class Edge {
  Color color ;
  Color getColor () { return col; }
  void setColor (Color c) { color = c; }
}
```

```
refines class Node {
  Color color;
  Color getColor () { return col; }
  void setColor (Color c) { color = c; }
}
```

Collaborations Instead of Aspects

- Dynamic extensions result in spaghetti code



AspectJ

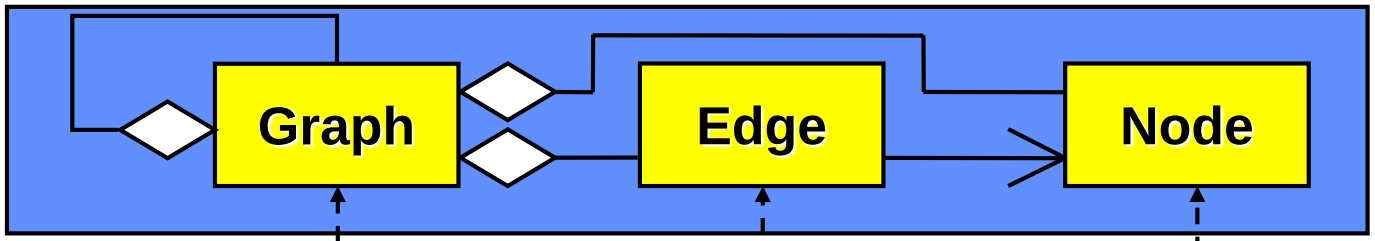
```
aspect PrintHeader {
  before() : execution(void print ()) &&
    !cflowbelow (execution(void print ())) {
      printHeader ();
  }
  void printHeader () { /* … */ }
}
```

Jak

```
refines class Node {
  static int count = 0;
  void print () {
    if(count == 0) printHeader ();
    count++; Super().print (); count--;
  }
  void printHeader () { /* ... */ }
}
```

Aspects Instead of Collaborations

- One aspect per collaboration

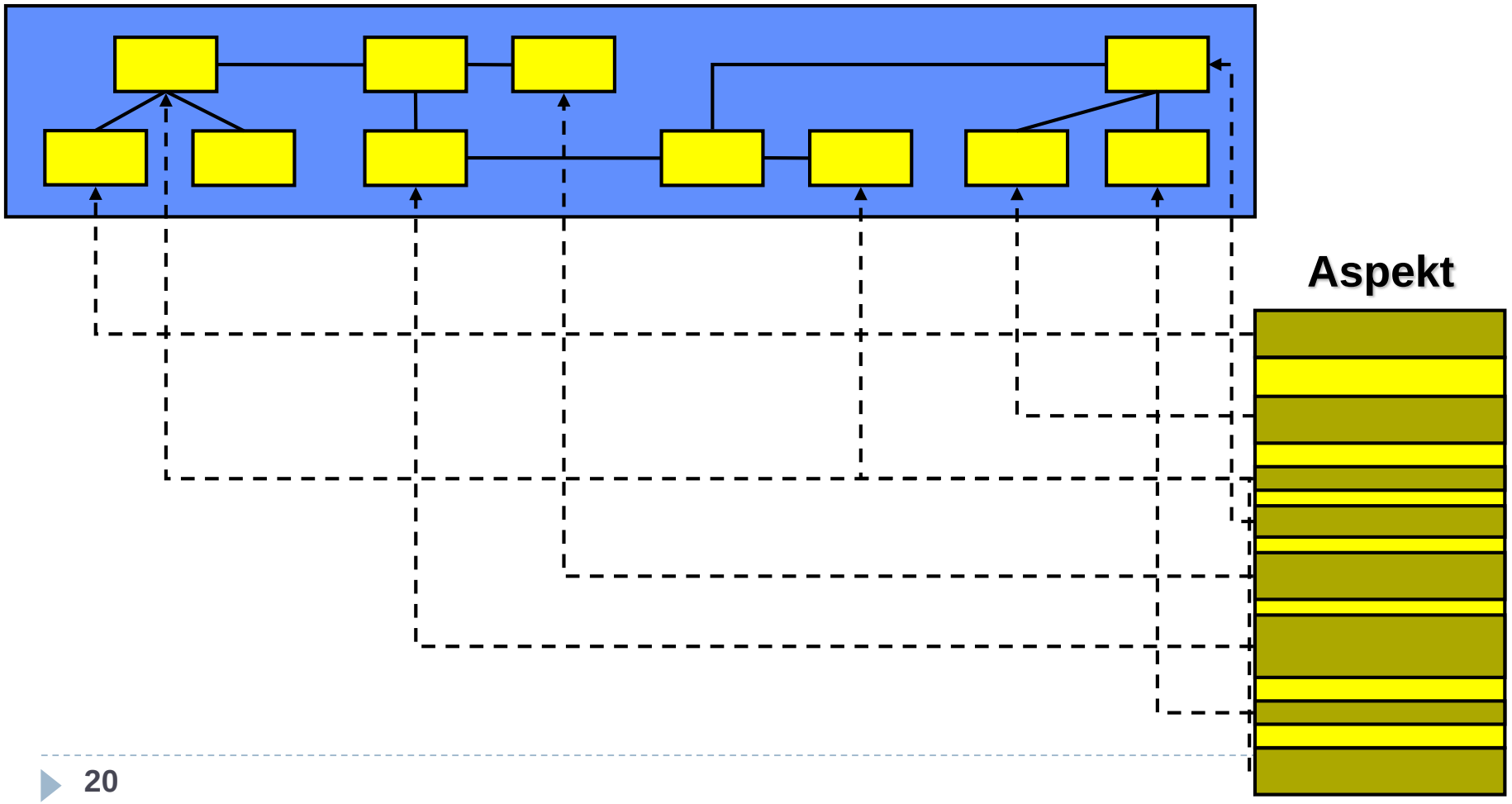


BaseGraph

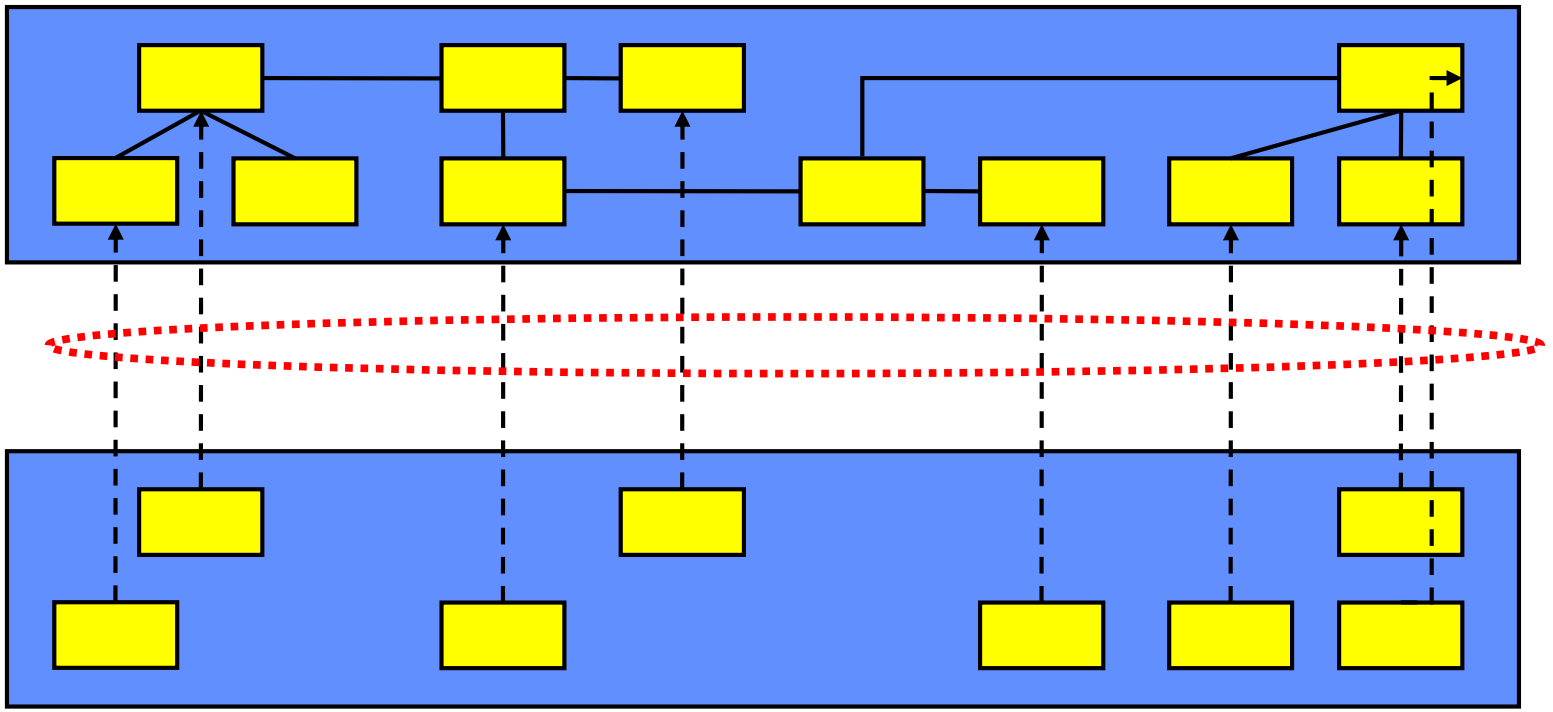
```
aspect AddWeight {
  Edge Graph.add(Node n, Node m, Weight w) {
    Edge res = add(n, m); res.weight = w; return res;
  }
  Weight Edge.weight ;
  after(Edge e) : this(e) &&
    execution(void Edge.print ()) {  /* … */  }
}
```

OO-Structure is not explicit

A Question of Scalability



Alternative: One Aspect per Role



Aspect weaving

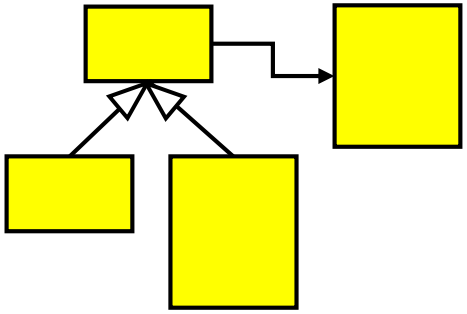
Aspect weaving replaces class refinements and superimposition without any advantage

Summary

- Using aspects or collaborations depends on the problem to be implemented
- Aspects and collaborations have different pros and cons

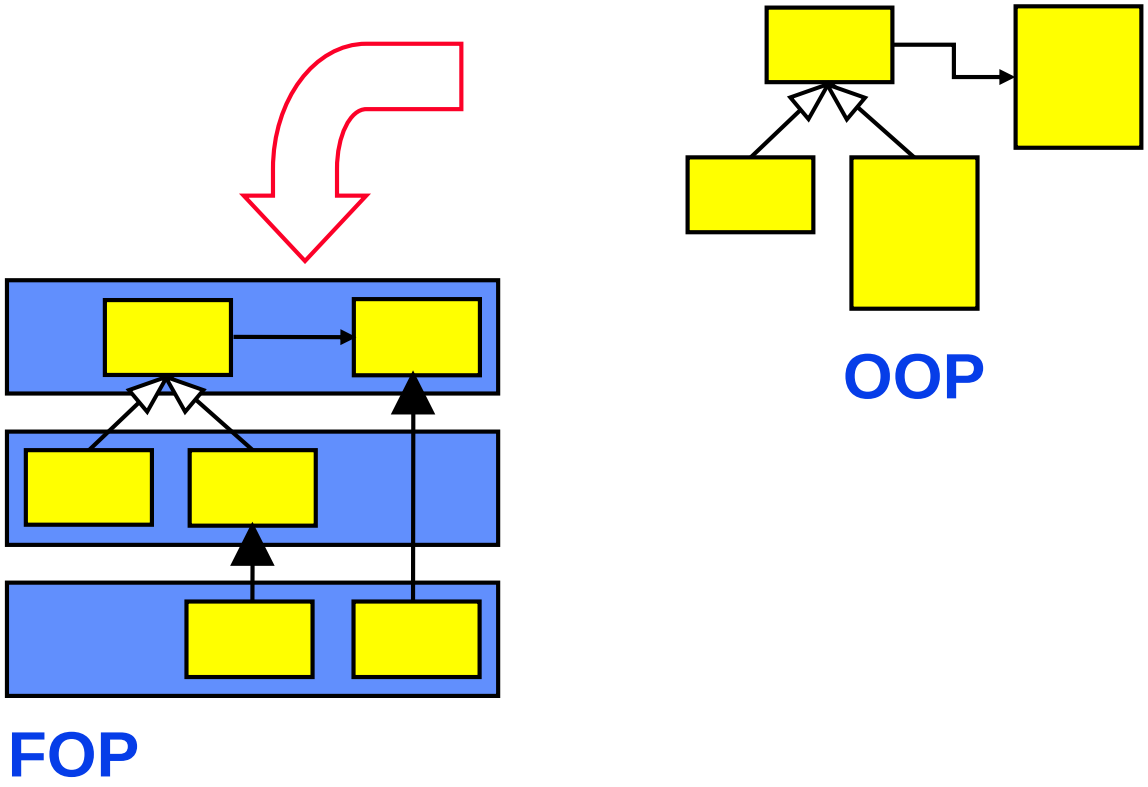
Symbiosis of AOP and FOP

Symbiosis of FOP and AOP

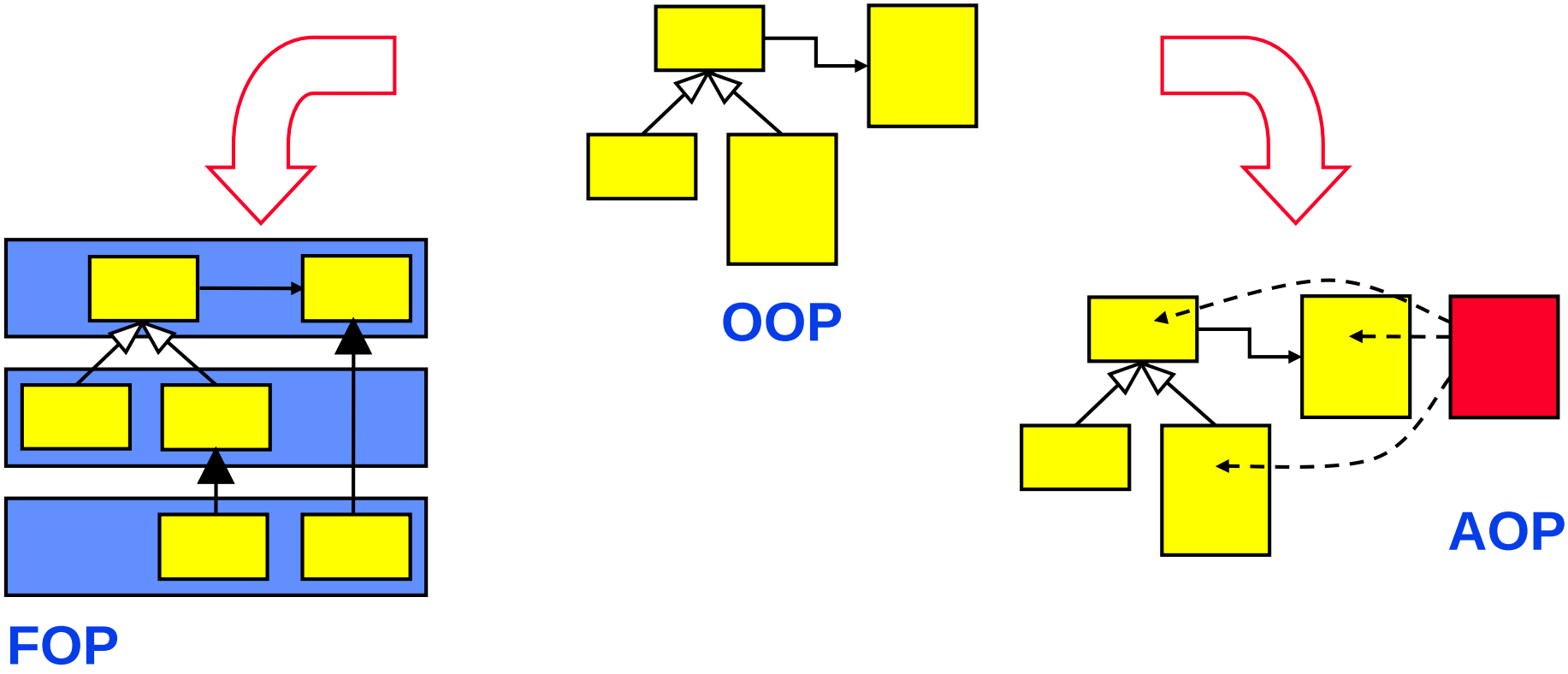


OOP

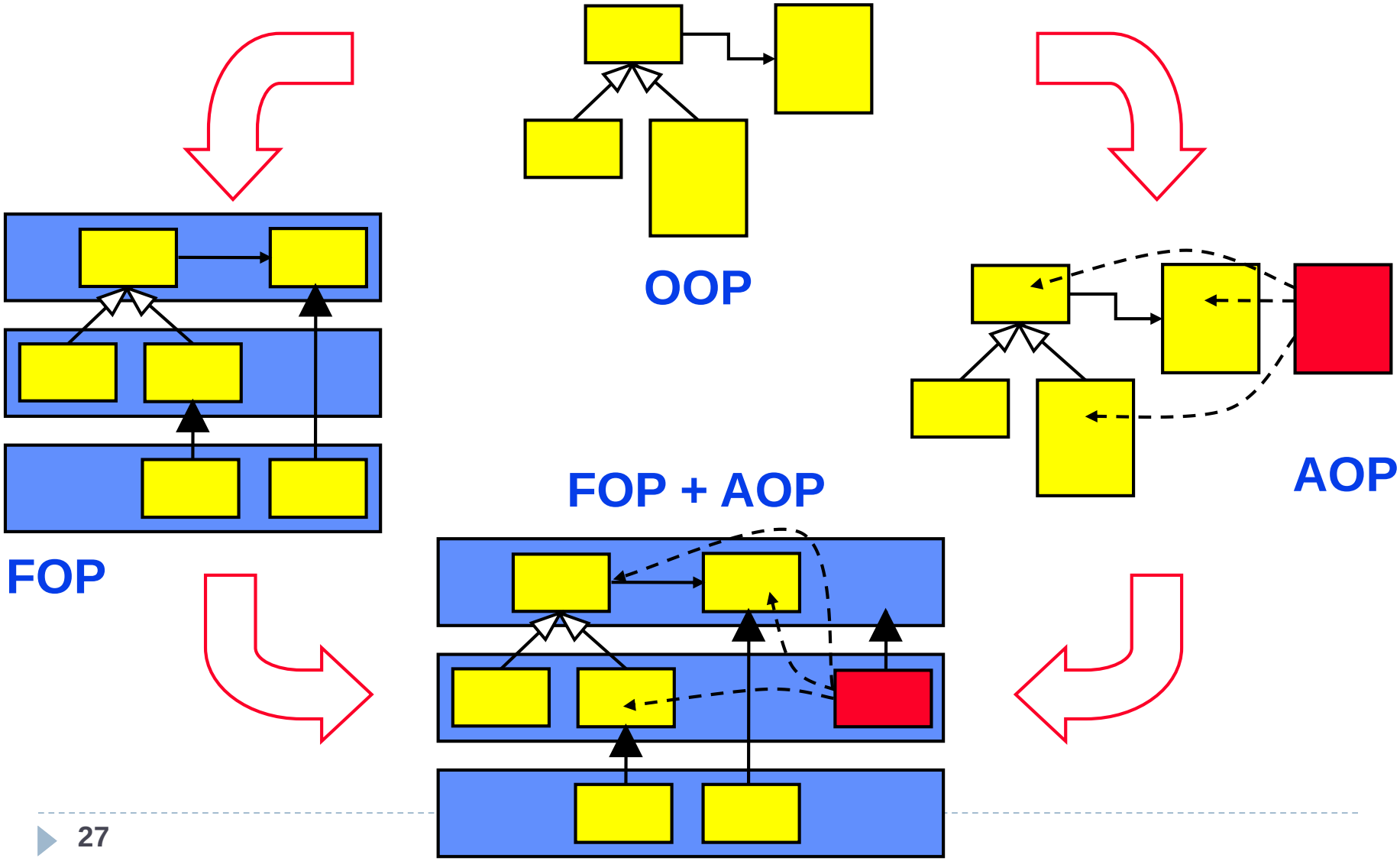
Symbiosis of FOP and AOP



Symbiosis of FOP and AOP

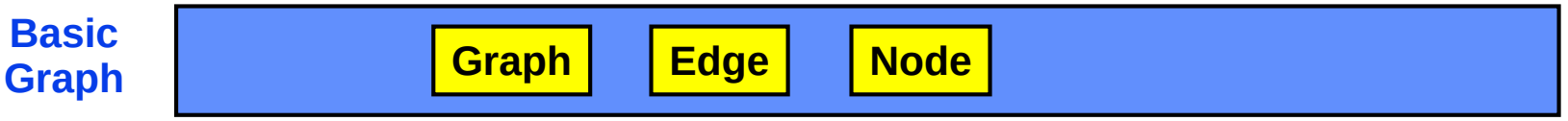


Symbiosis of FOP and AOP



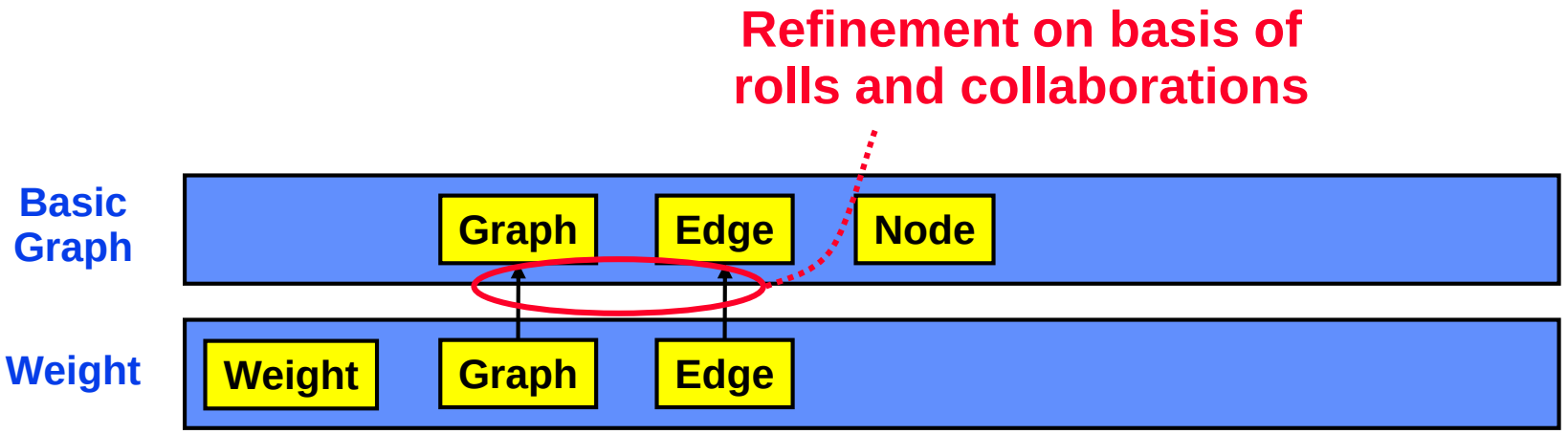
Aspectual Feature Modules

- Integration of aspects, classes, and refinements



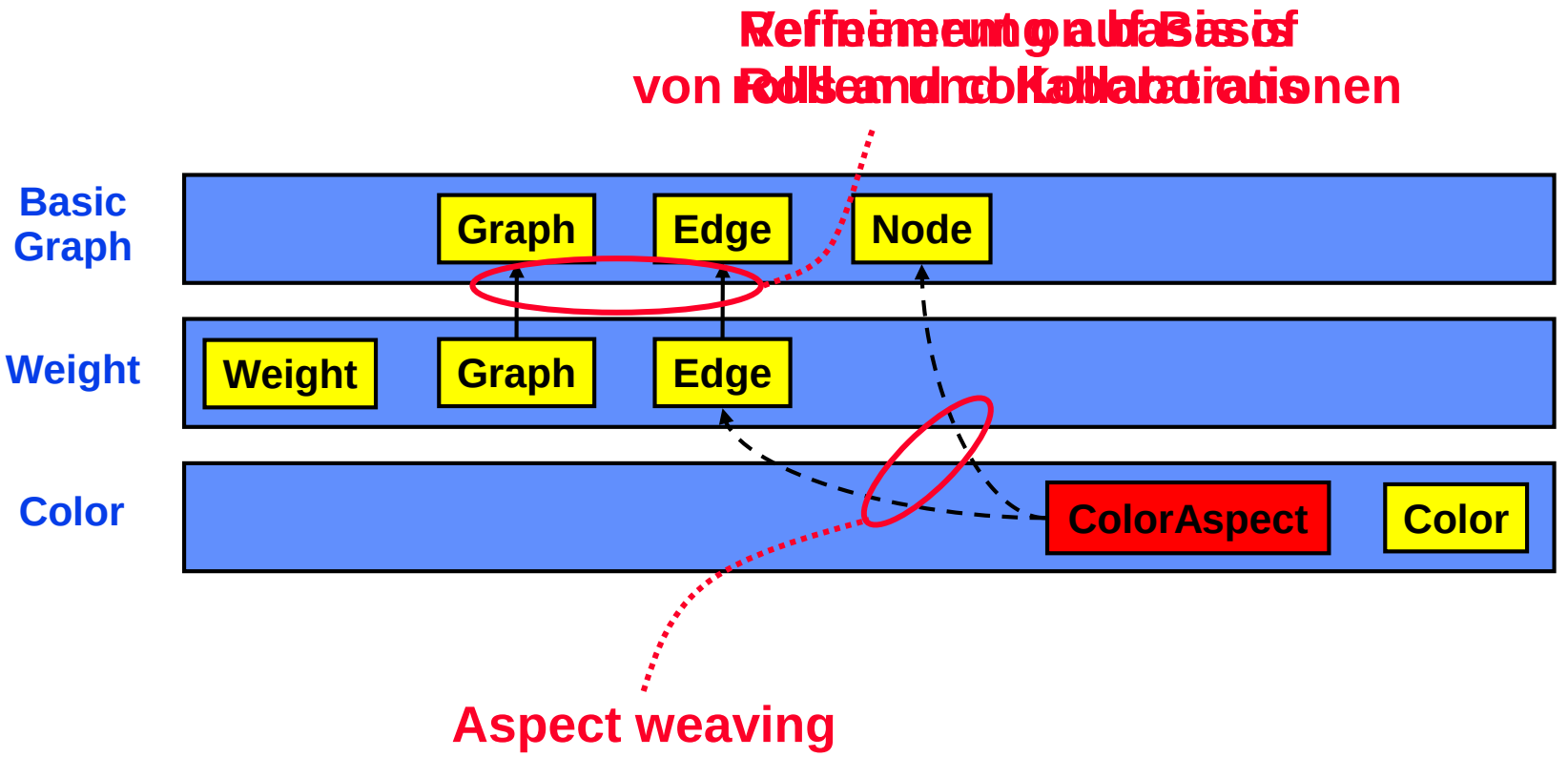
Aspectual Feature Modules

- Integration of aspects, classes, and refinements



Aspectual Feature Modules

- Integration of aspects, classes, and refinements



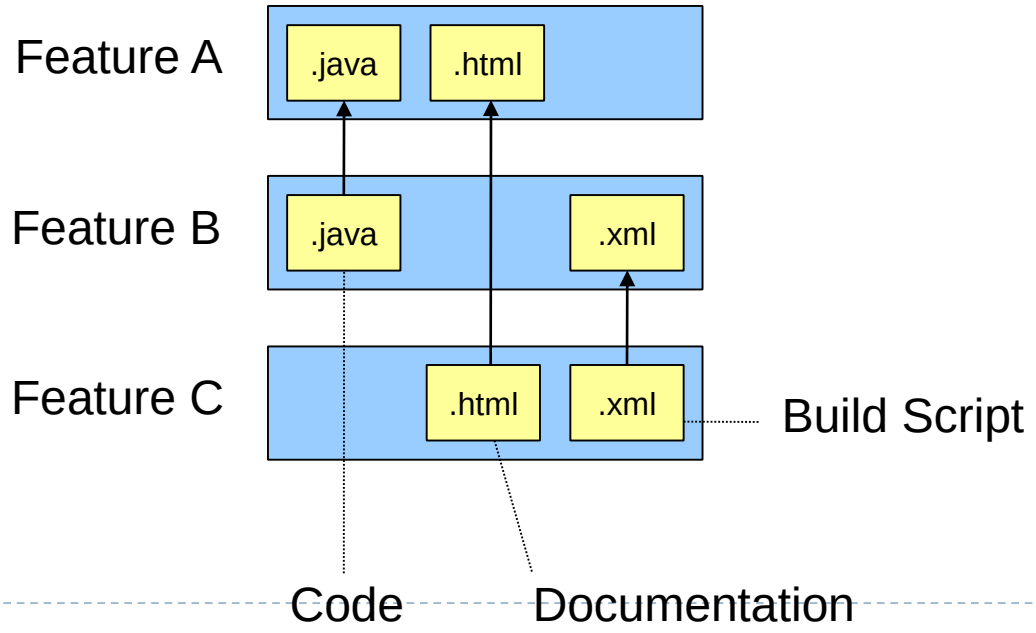
Tool Support

- FeatureC++ & AspectC++
  - Aspectual Feature Modules for C++
  - Direct language support
  - Compiles first with FeatureC++ and later with the AspectC++ compiler
- AHEAD Tool Suite & AspectJ
  - Java-based variant of Aspectual Feature Modules
  - Aspect files in feature modules; translation via AspectJ compiler
- (no support in FeatureIDE)

Aspect Refinement

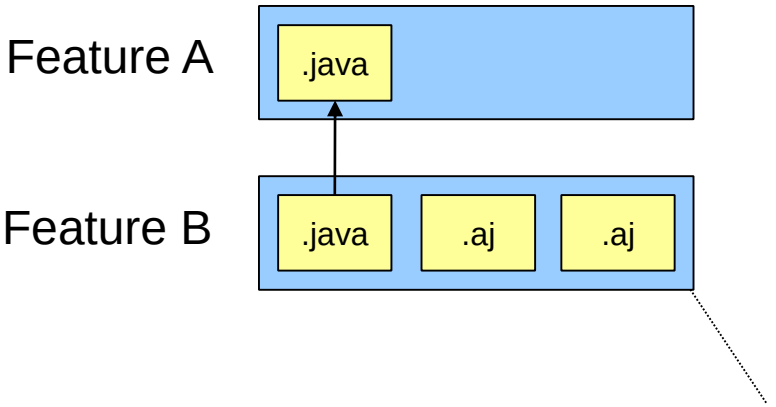
Recap: Principle of Uniformity

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



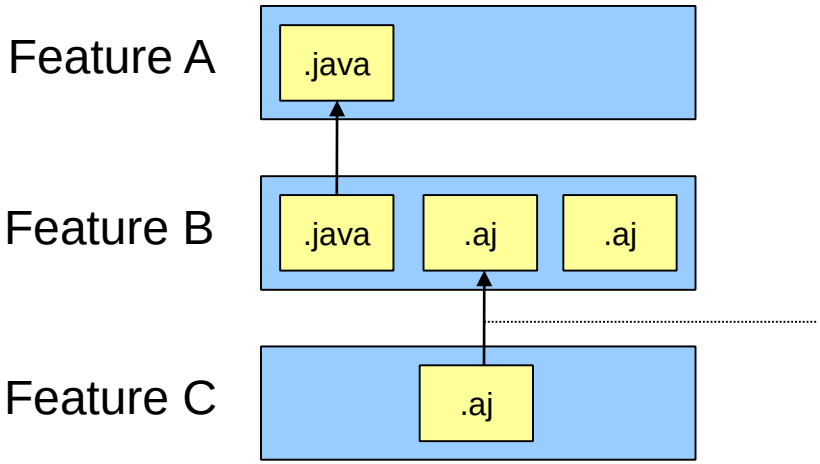
Aspects – An additional Type of Software Artifact

- Aspects collaboration with other artifacts to realize a feature



Aspectual Feature Module

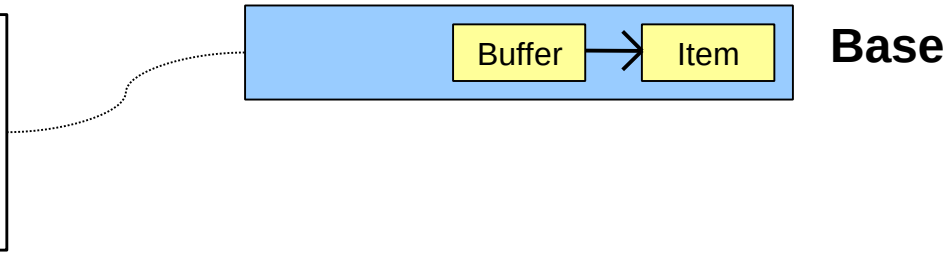
Idea



Why can't we refine aspects?

Example – AspectJ

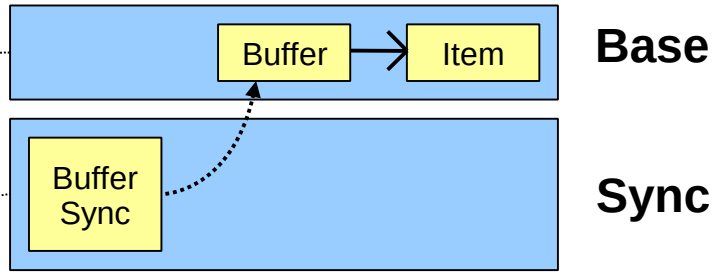
```
class Buffer {
  Vector buf = new Vector();
  void put(Item e) { buf.add(e); }
  Item get(int i) {
return (Item)buf.get(i); }
}
```



Example – AspectJ

```
class Buffer {
  Vector buf = new Vector();
  void put(Item e) { buf.add(e); }
  Item get(int i) {
return (Item)buf.get(i); }
}
```

```
abstract aspect BufferSync {
  pointcut syncPC() :
    execution(Item Buffer.get(int)) ||
    execution(void Buffer.put(Item));
  Object around() : syncPC() {
    lock();
    Object res = proceed();
    unlock();
    return res;
  }
}
```

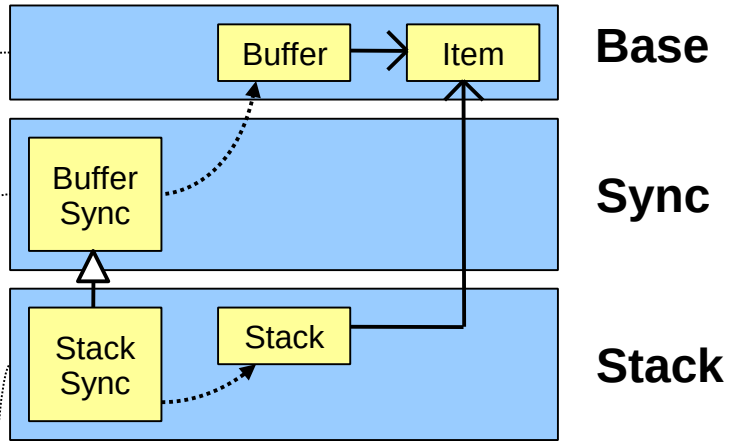


Example – AspectJ

```
class Buffer {
  Vector buf = new Vector();
  void put(Item e) { buf.add(e); }
  Item get(int i) {
return (Item)buf.get(i); }
}
```

```
abstract aspect BufferSync {
  pointcut syncPC() :
    execution(Item Buffer.get(int)) ||
    execution(void Buffer.put(Item));
  Object around() : syncPC() {
    lock();
    Object res = proceed();
    unlock();
    return res;
  }
}
```

```
class Stack {
  LinkedList list = new LinkedList();
  void push(Item i) { list.addFirst(i); }
  Item pop() { return (Item)list.getFirst(); }
}
aspect StackSync extends BufferSync {
  pointcut syncPC() : BufferSync.syncPC() ||
    execution(Item Stack.pop()) ||
    execution(void Stack.push(Item));
}
```

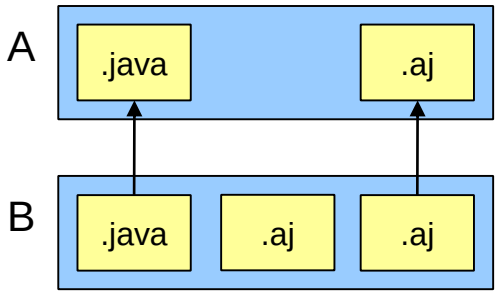


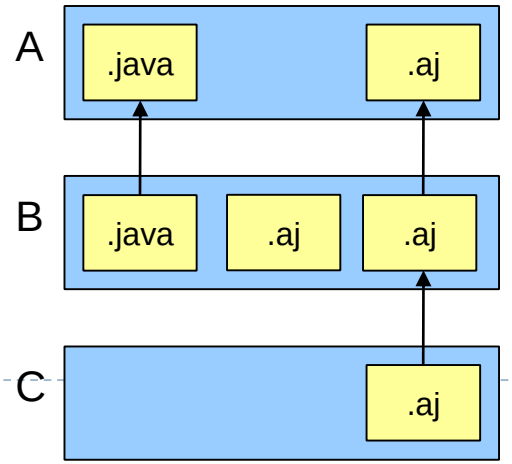
Limits of AspectJ

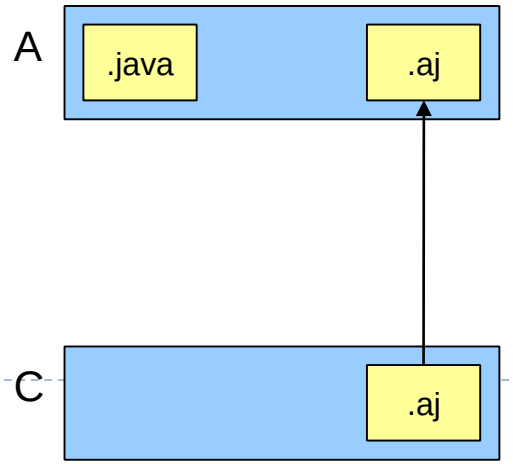
- Aspect inheritance
  - Aspects that are refined, are “fixed”
  - Tight coupling between aspect and refinement
  - Refer to “Inflexible extension mechanisms”
- Abstract aspects
  - Only abstract aspects can be refined
  - Refinements must be preplanned
- Advice constructs have no names
  - Cannot be refined

Collaborations and Rolls for Aspects

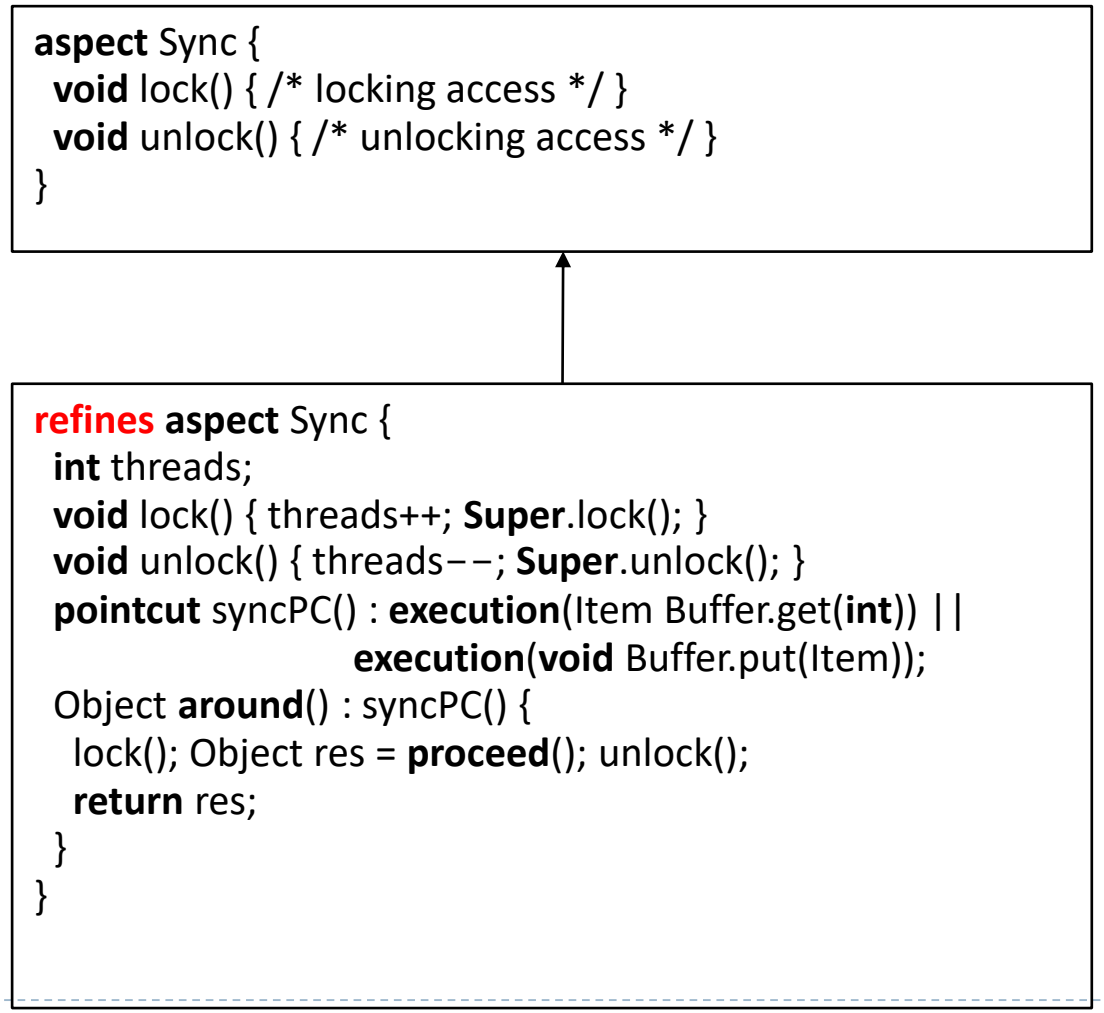
- Analog to class refinements
- Refinements at compile time
  - Each aspect can be refined
- AspectJ-Extension on the basis of Jak (experimental compiler arj), partially in FeatureC++ possible



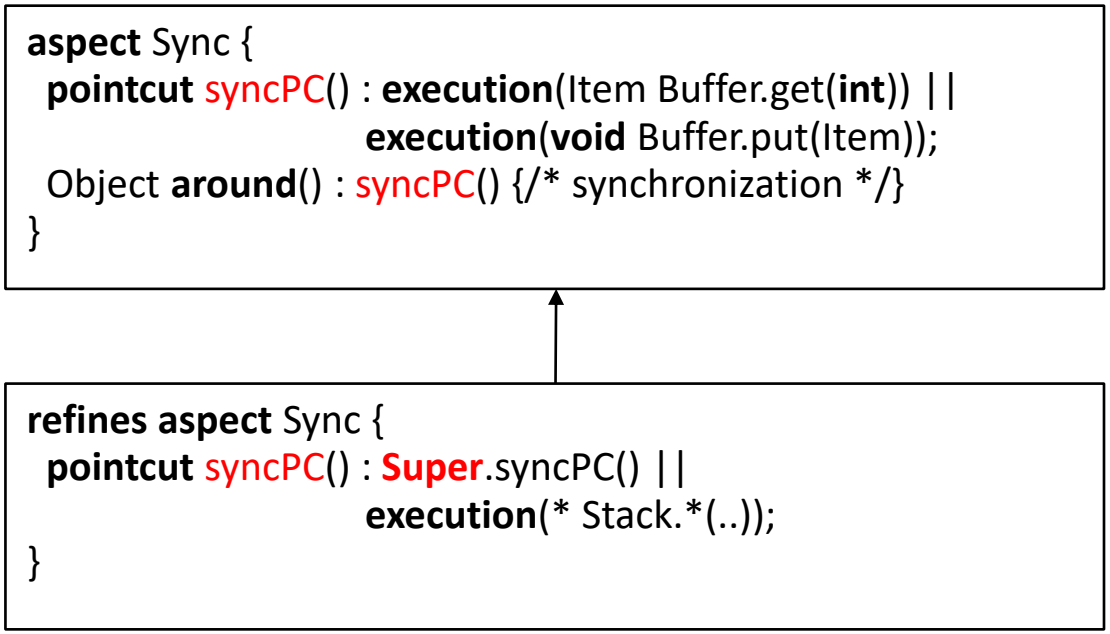




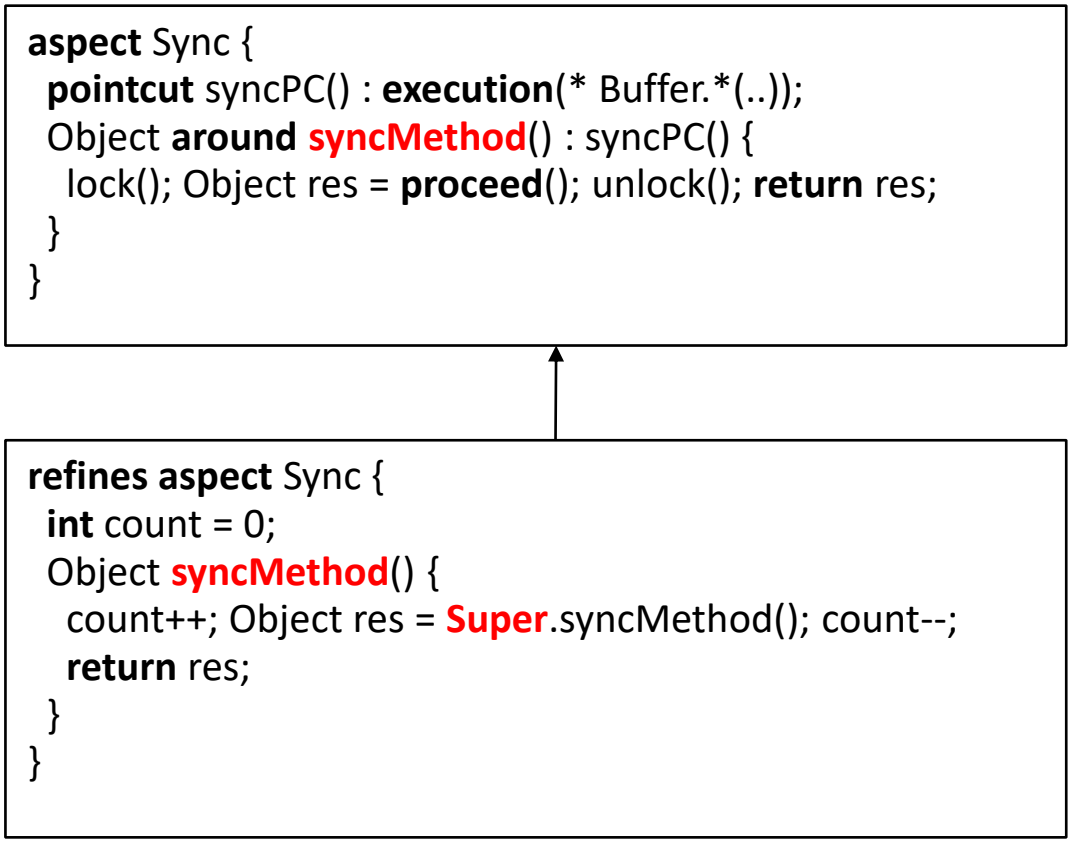
Adding of Elements and Extension of Methods



Pointcut Refinement



(Named) Advice Refinement



Tool Support

- FeatureC++ & AspectC++
  - Without Advice Refinement
- AspectJ
  - Experimental Compiler ARJ
  - Extention of AspectJ
- (no support in FeatureIDE)

Case Studies on using AOP

Why Case Studies?

- So far
  - Aspects and collaborations are complementary
  - Use case of aspects and collaborations depend on the implementation problem
  - Aspects and collaborations have different pros and cons
  - Possibility of Aspect Refinement
- Questions:
  - Why not always using collaborations or aspects?
  - Are their differences actually relevant?
  - What is actually required in practice?

Approach

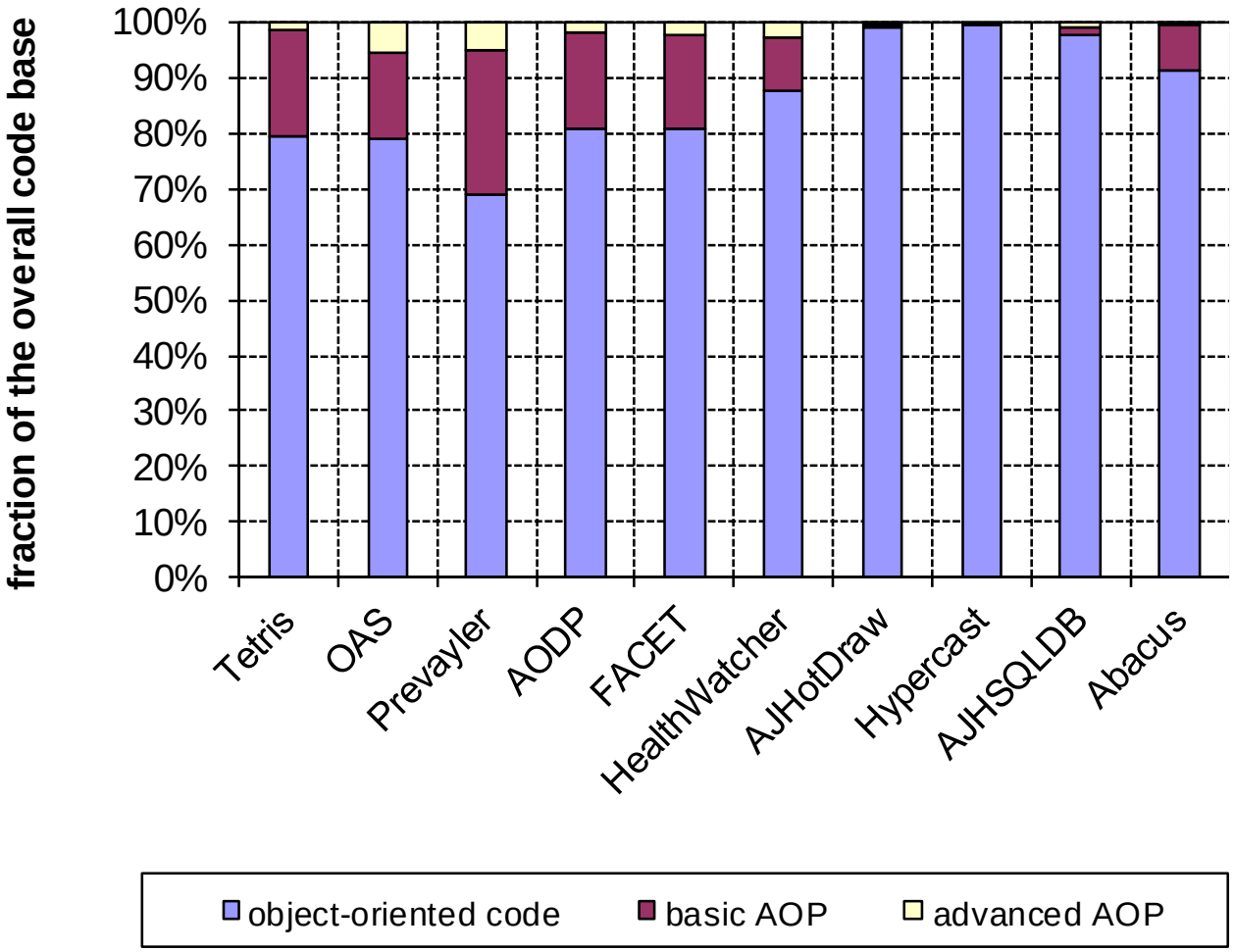
- Empirical analysis of 10 existing AspectJ programs
  - AspectJ-Programs exist
  - AspectJ: Superset of all analyzed functionality
  - FOP for empirical analysis not sufficiently distributed
- Collection of metrics
  - Partially manual
  - Partially automated: AJStats[1], AJDTStats[2]

[1] http://wwwiti.cs.uni-magdeburg.de/iti_db/forschung/ajstats/
[2] http://wwwiti.cs.uni-magdeburg.de/iti_db/forschung/ajdtstats/

Analyzed AspectJ-Programs

Tetris	The popular game	1 KLOC	Blekinge Institute of Technology
OAS	An online auction system	2 KLOC	Lancaster University
Prevayler	Transparent persistence for Java	4 KLOC	University of Toronto
AODP	Aspect-oriented implementation of the Gang-of-Four design patterns	4 KLOC	University of British Columbia
FACET	An aspect-based CORBA event channel	6 KLOC	Washington University
HealthWatcher	Web-based information system for public health systems	7 KLOC	Lancaster University
AJHotDraw	2D graphics framework	22 KLOC	Sourceforge project
Hypercast	Multicast overlay network communication	67 KLOC	University of Virginia, Microsoft
AJHSQLDB	SQL relational database engine	76 KLOC	University of Passau
Abacus	A CORBA middleware framework	130 KLOC	University of Toronto

Application of AOP



Aspect Refinement in P2P-PL

- 7 of 14 aspecs have been refined

Name	No. of Ref.	Description
serialization	11	prepares objects for serialization
responding	4	sends replies automatically
toString	12	introduces toString methods
log/debug	13	mix of logging and debugging
pooling	3	stores and reuses open connections
dissemination	3	piggyback meta-data propagation
feedback	2	generates feedback by observing peers

Literature for Comparison and Symbiosis

- S. Apel, T. Leich, and G. Saake. Aspectual Feature Modules. IEEE Transactions on Software Engineering, 34(2), 2008.
  [Aspectual Feature Modules]
- S. Apel. How AspectJ is Used: An Analysis of Eleven AspectJ Programs. Journal of Object Technology, 9(1), 2010.
  [Analysis of 11 AspectJ-Programs]