Aspect Oriented Software Development (ASOD)/
Aspect Oriented Programming (AOP)

Andy Carpenter

School of Computer Science

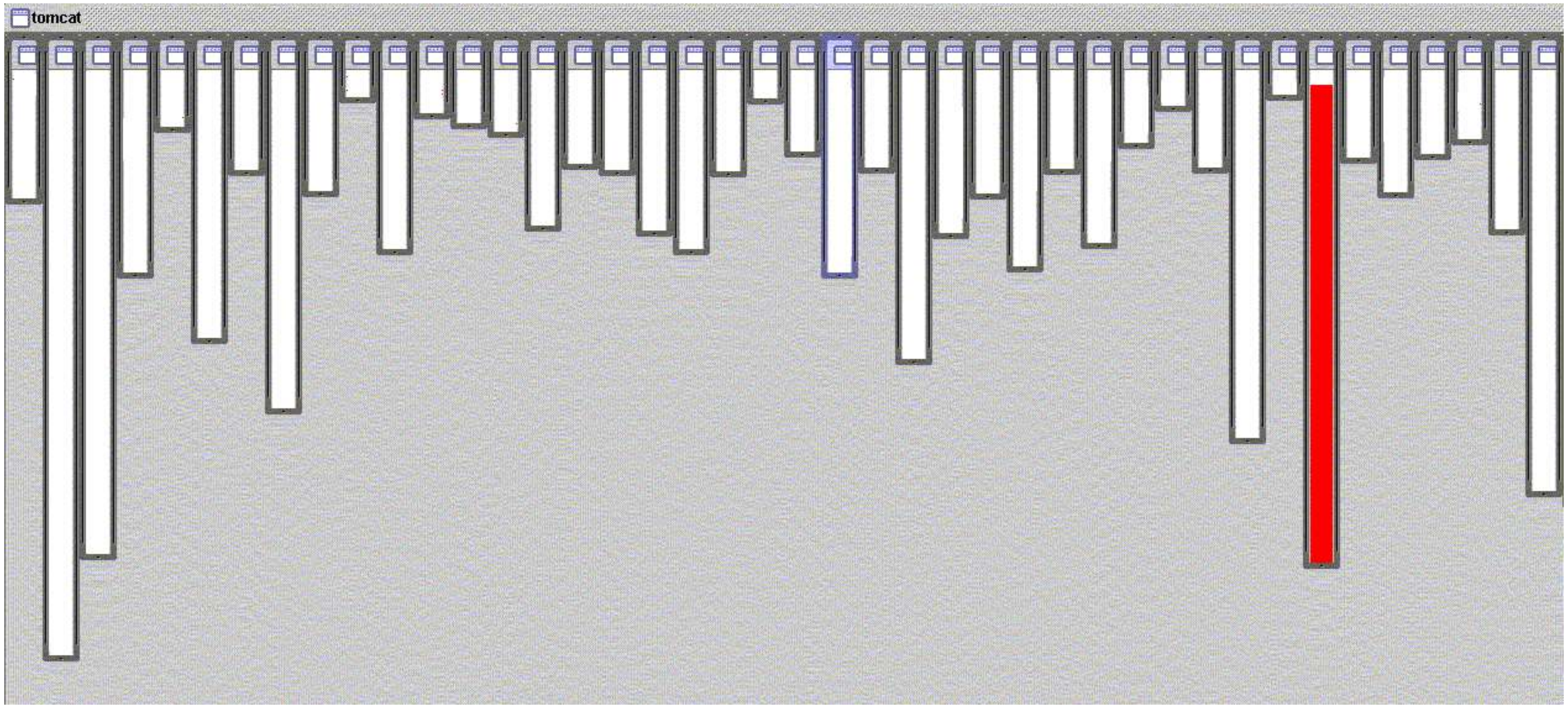
(Andy.Carpenter@manchester.ac.uk)

Based on slides from Viviane Jonckers, Kevin Hoffman

Where do you implement…?

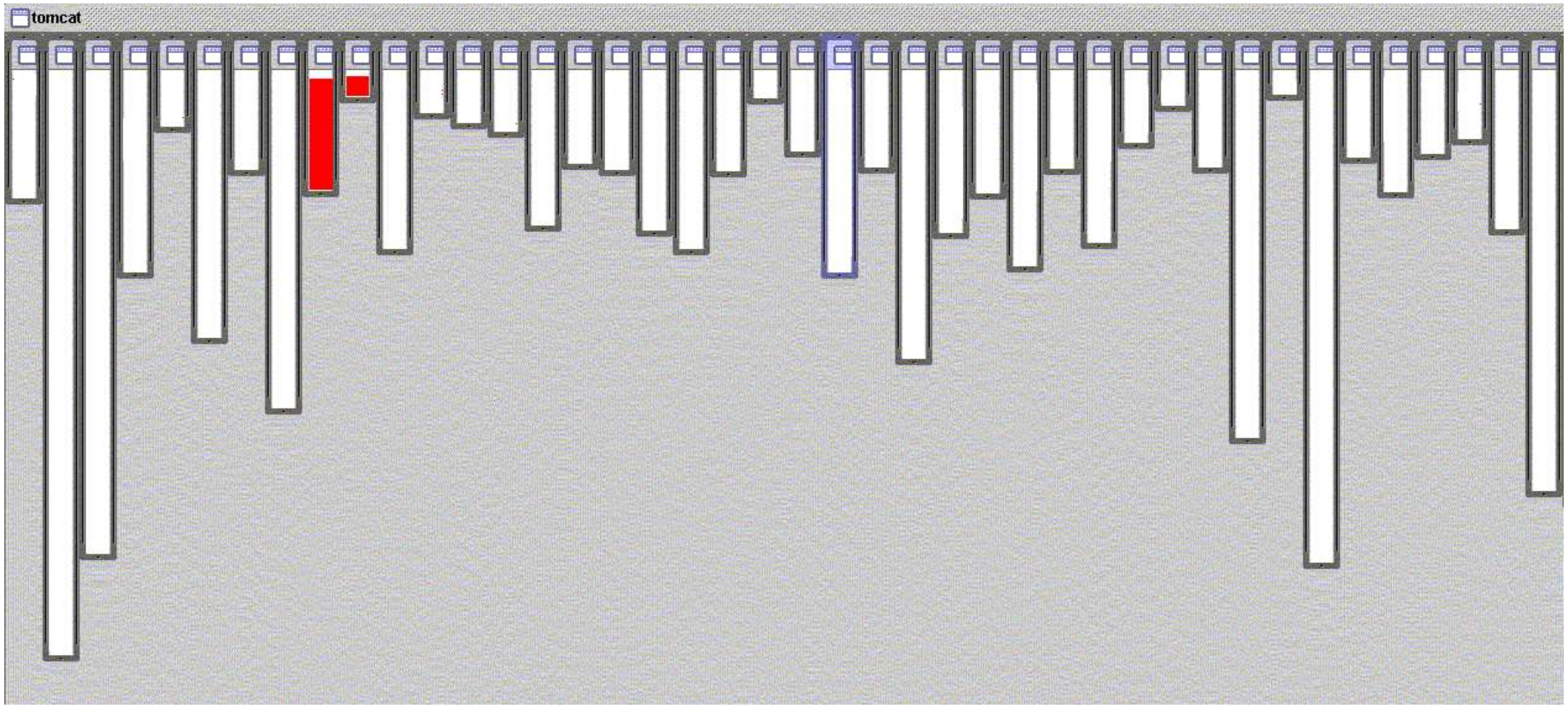
- Logging
- Security
  - access control, confidentiality
- Transaction management
- Error recovery

XML Handing in Apache Tomcat



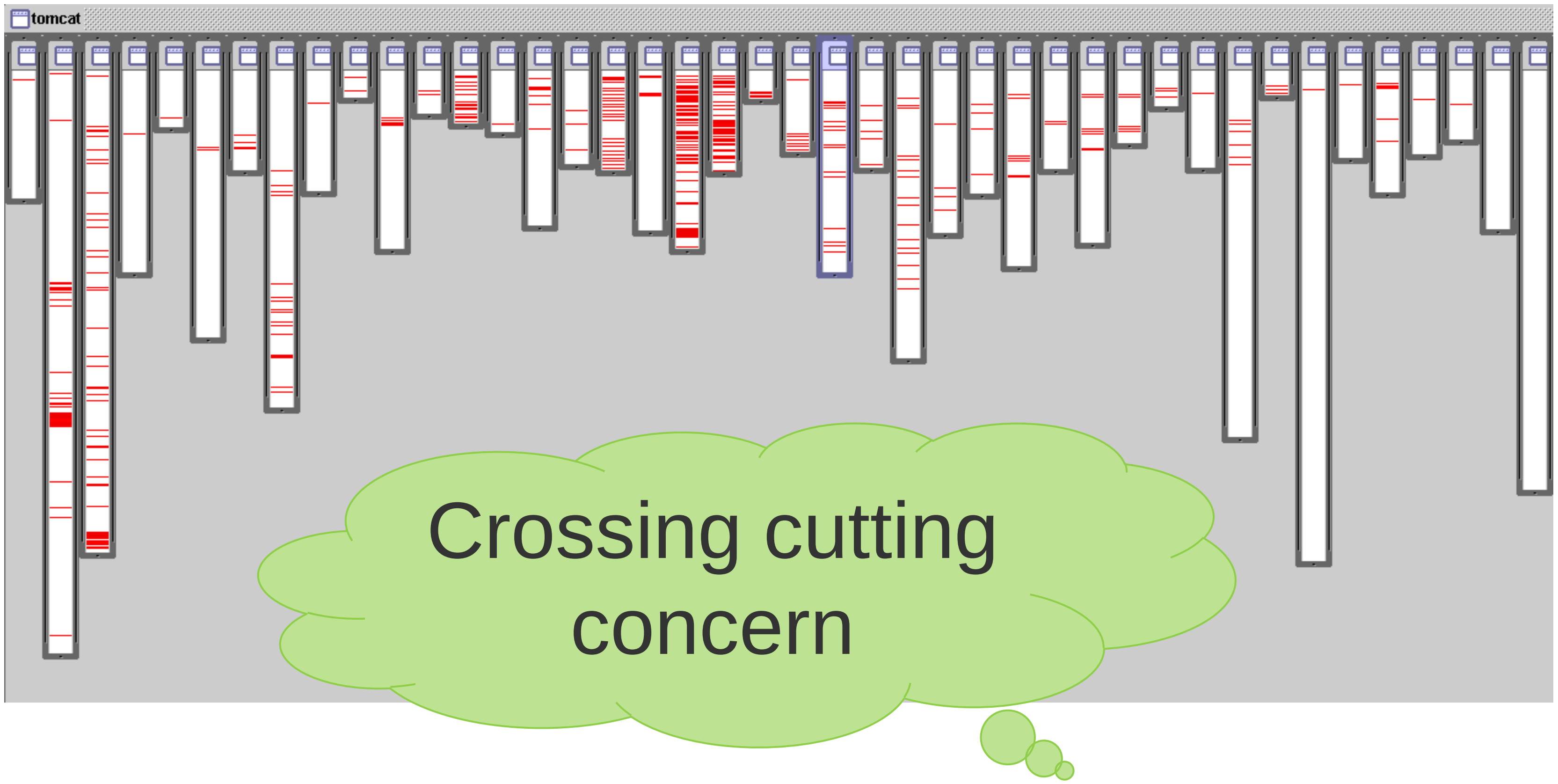
Code handled by one class

URL Handing in Apache Tomcat



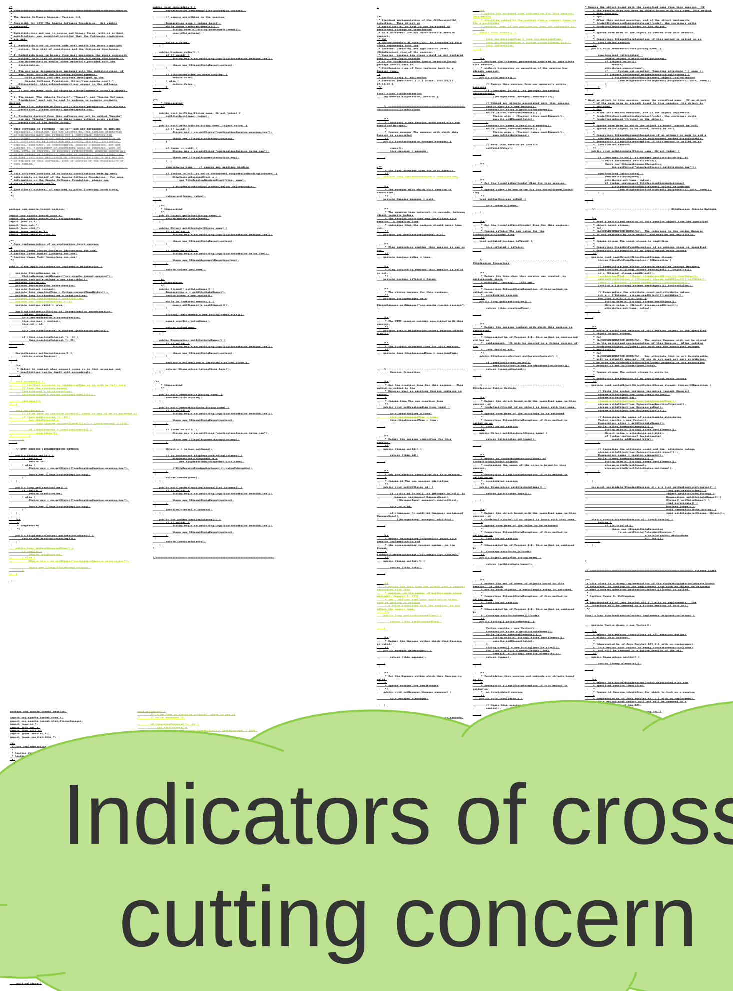
Code handled by two related (by inheritance) classes

Logging in Apache Tomcat



Handled by code that is scattered over almost all classes

Scattering and Tangling

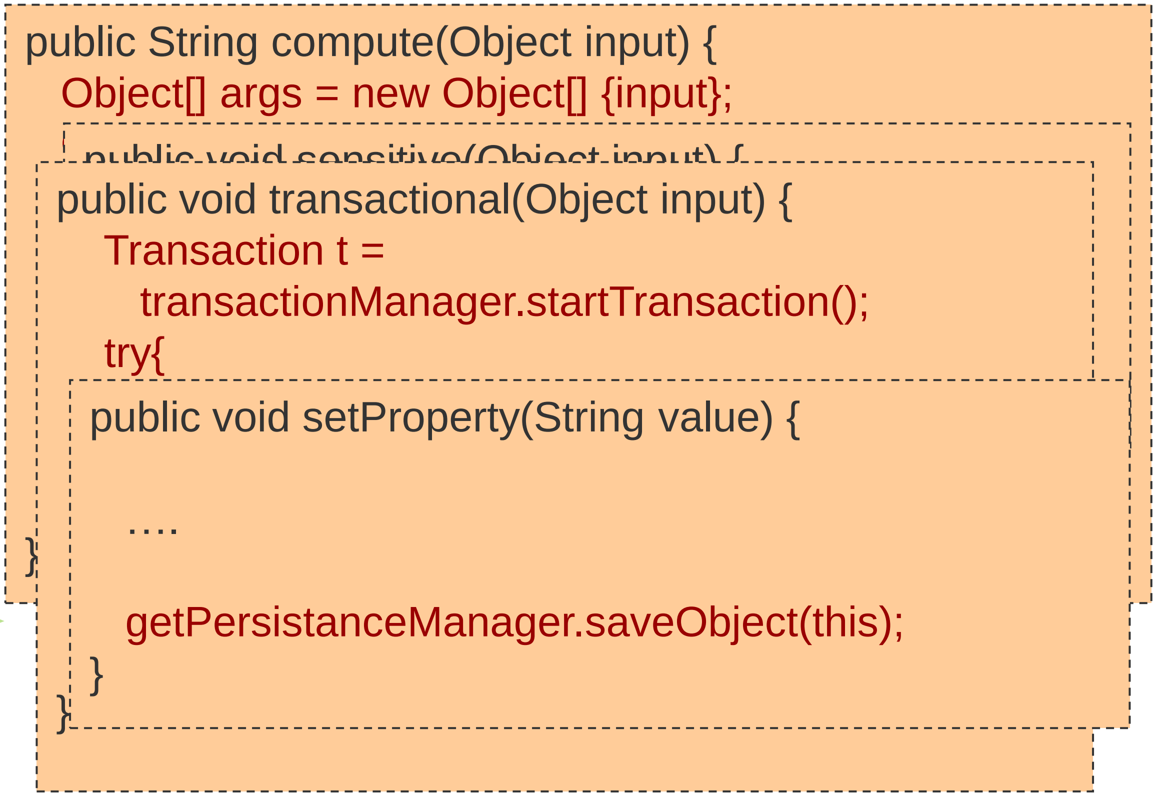


- Scattering
  - code addressing one concern is spread around
- Tangling
  - code in one region addresses multiple concerns
- Tend to occur together

Indicators of cross cutting concern

Examples of CCC

- Logging
- Caching
- Security
  - Access Control
  - Confidentiality
- Transaction Management
- Persistance
- Error recovery
- …



```
public String compute(Object input) {
   Object[] args = new Object[] {input};
```

```
public void sensitive(Object input) {
```

```
public void transactional(Object input) {
    Transaction t =
       transactionManager.startTransaction();
    try{
```

```
public void setProperty(String value) {

    ....

    getPersistanceManager.saveObject(this);
}
```

```
}
}
```

Tyranny of the Dominant Decomposition

Given one out of many possible decompositions of the problem... *(mostly core functionality concerns)*

...then some subproblems cannot be modularized!

(non-functional, functional, added after the facts,...)

- Not only for a given decomposition
  - But for all possible decompositions
- Not only in object-orientation!
  - Also in other paradigms
- Not only in implementation!
  - Also in analysis & design stages

Separation of Concerns

Concern: “*Something the developer needs to care about*” (e.g. functionality, QoS requirement,..)

Separation of concerns: handle each concern separately

- Modular programming
  - Organize code by grouping data/functionality
- Need for language mechanisms
  - Drives evolution of languages & paradigms

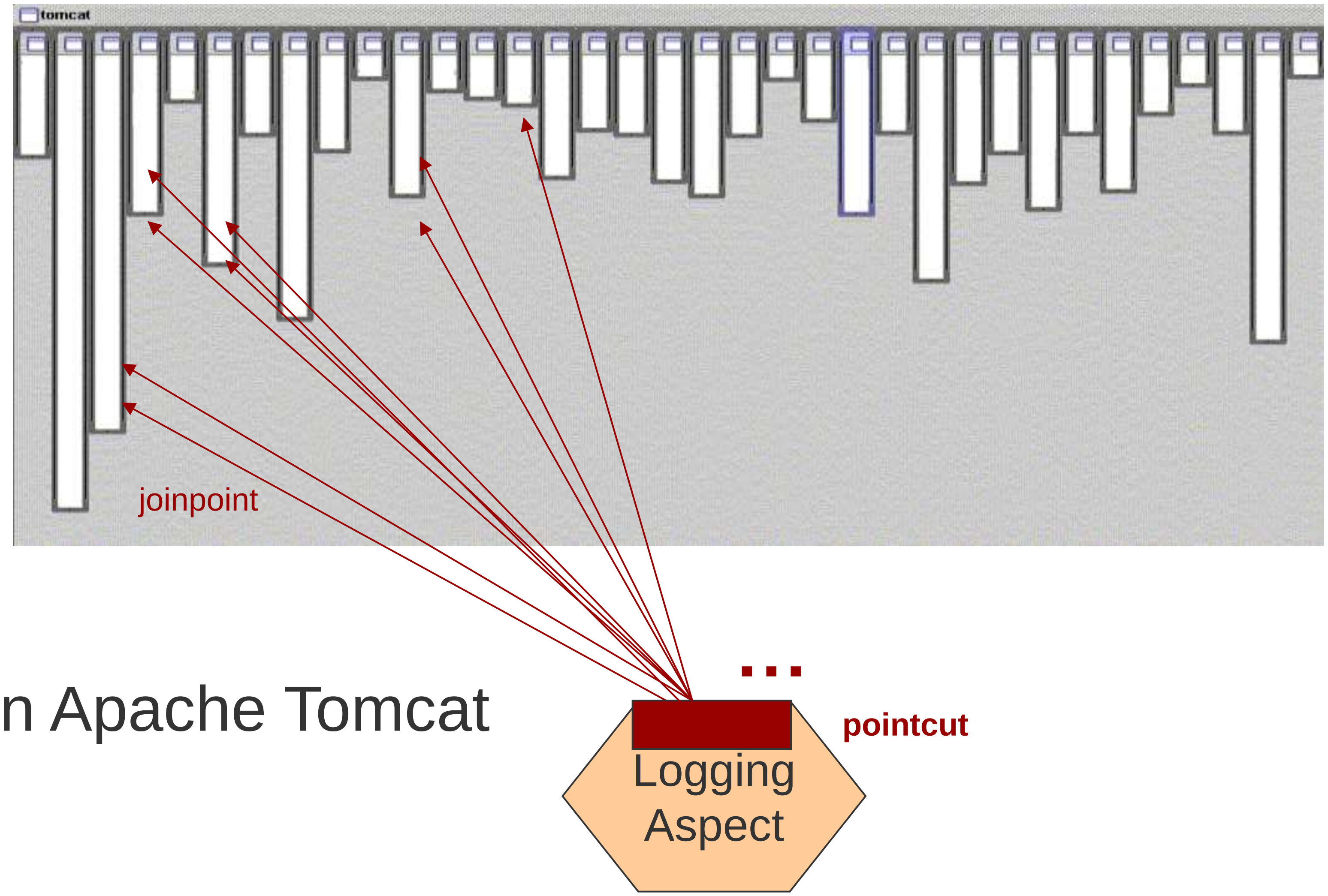
The AOSD idea

- crosscutting is inherent in complex systems

  “tyranny of the dominant decomposition”

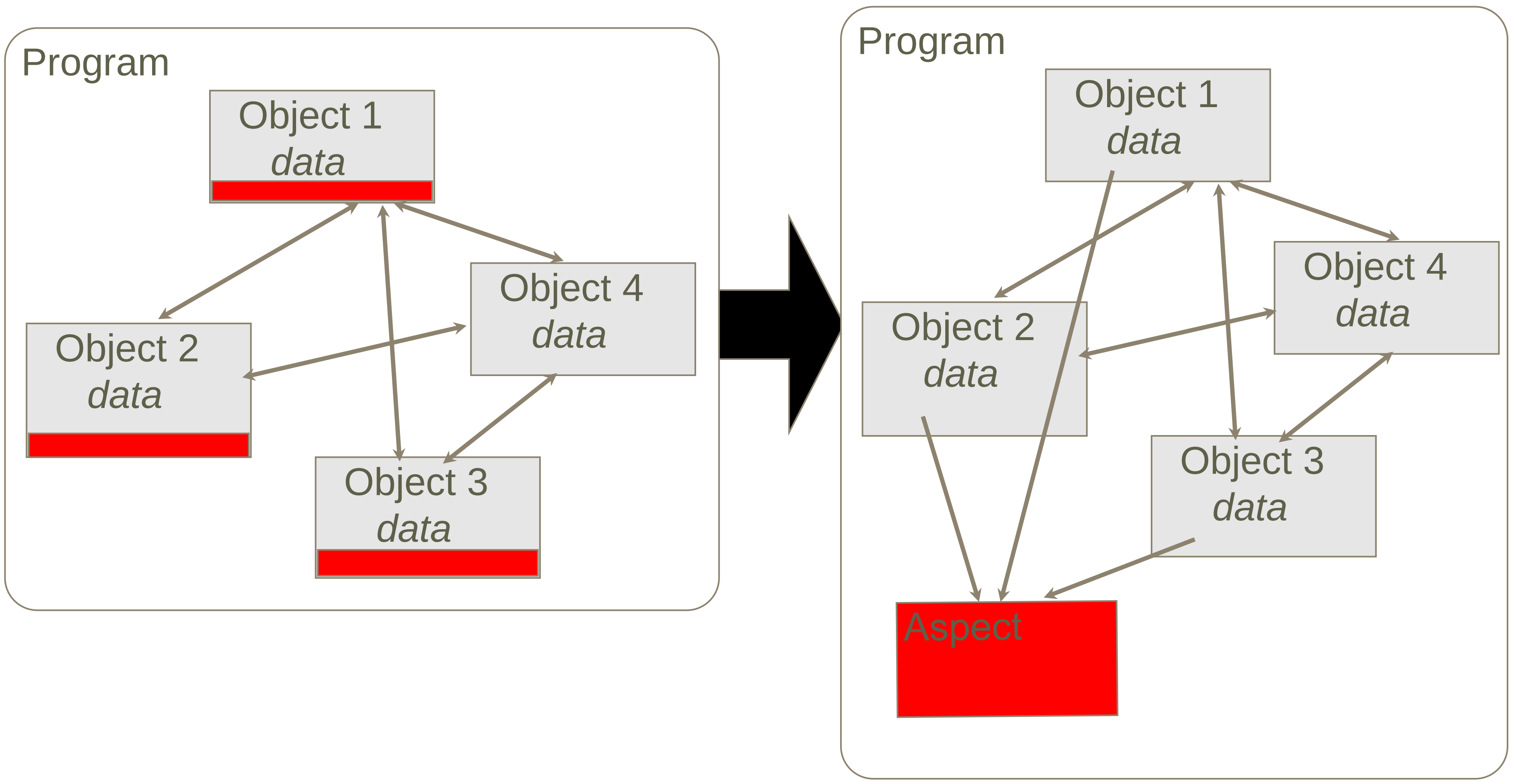
- crosscutting concerns
  - have a clear purpose *What*
  - have some regular interaction points *Where/When*
- AOP proposes to capture crosscutting concerns explicitly...
  - in a modular way
  - with programming language support
  - and with tool support

Aspect

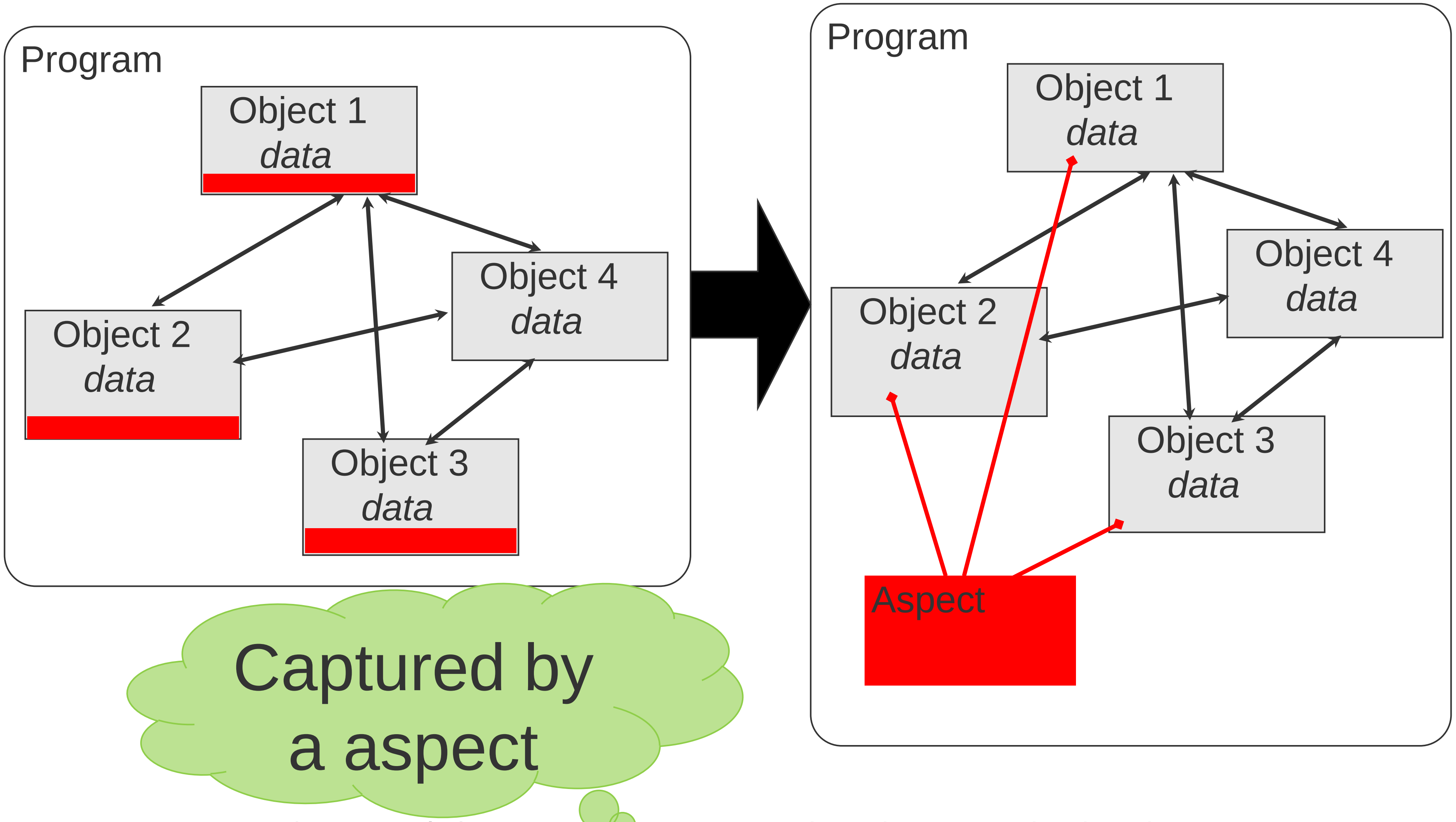


ing in Apache Tomcat

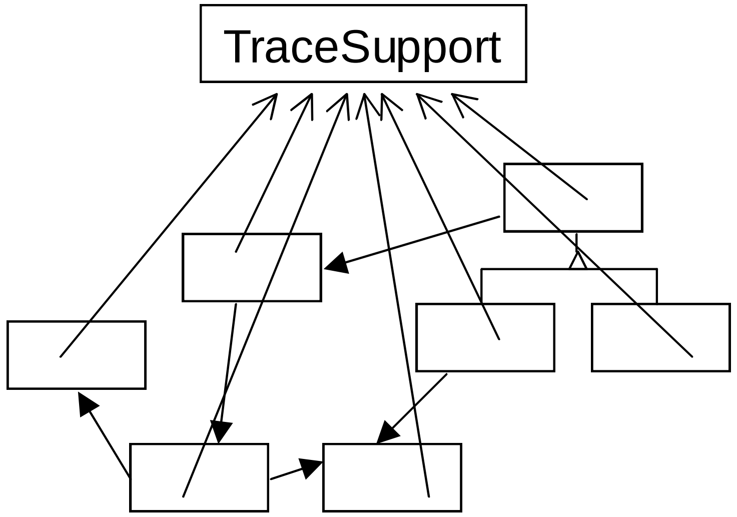
Explicit Invocation

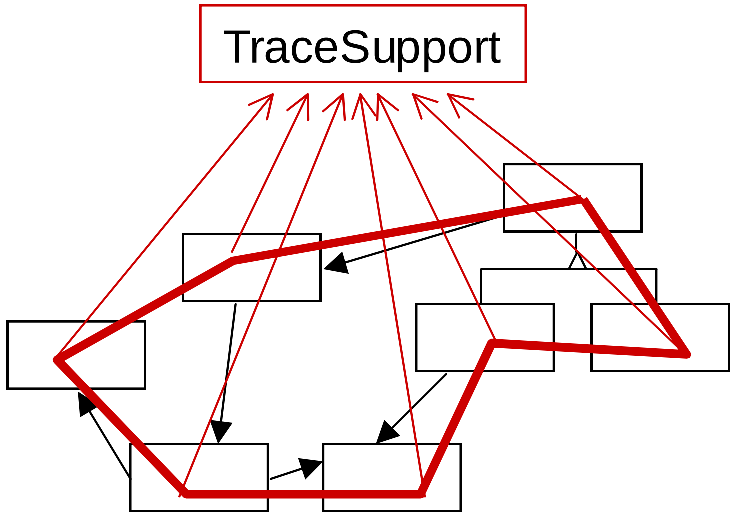


Implicit Invocation



Implicit Invocation is Captured by an aspect





Objects are invoked by other objects through message sends

Aspect captures its own invocation that crosscuts other modules

Aspect

Where/
When

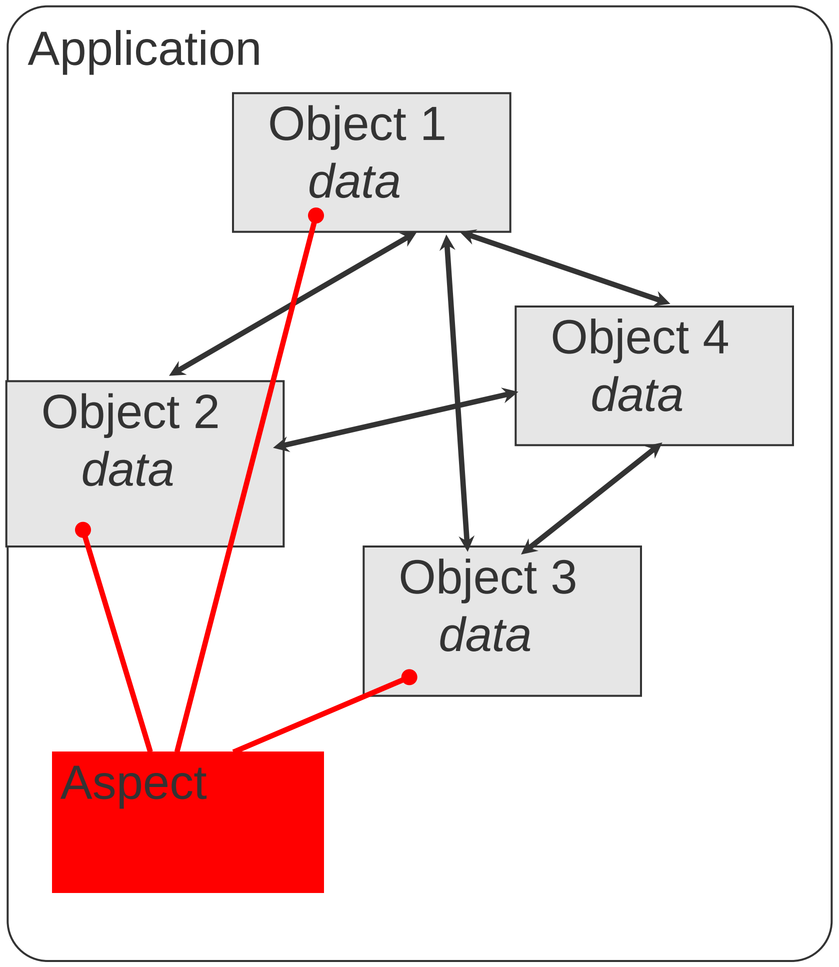
What

Multiple

Terminology

- **joinpoint:** point in the program's execution
  - e.g. Executing method setX on type Y
- **pointcut:** A set of joinpoints — **Where / when**
  - e.g. The execution of all methods which name start with "set" on type Y
- **advice:** What to do at a certain pointcut: — **What**
  - e.g. Print a logging string before the program continues.
- **weaving**
  - applying the advice to all joinpoints defined in the pointcut declaration

Joinpoints



●: joinpoint

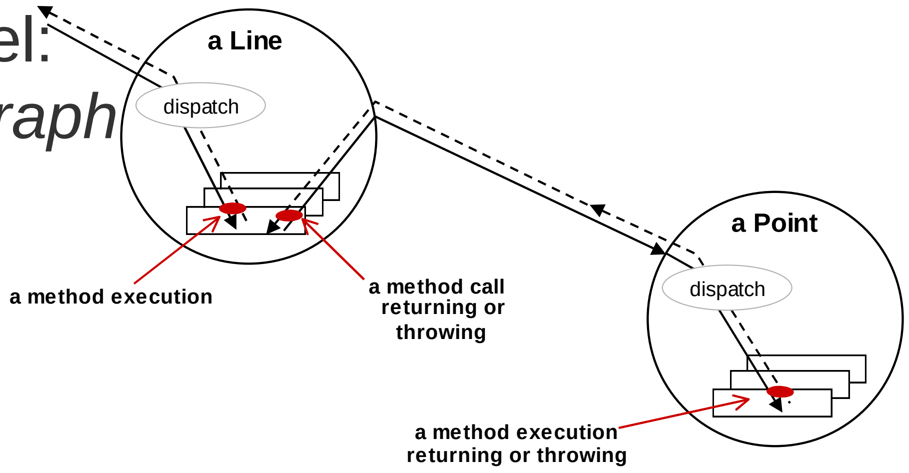
- A join point is a point of interest in some artefact in the software lifecycle through which two or more concerns may be composed.

Examples in implementation artefact:
- message sends
- method executions
- error throwing
- variable assignments
- ...

Join point Model

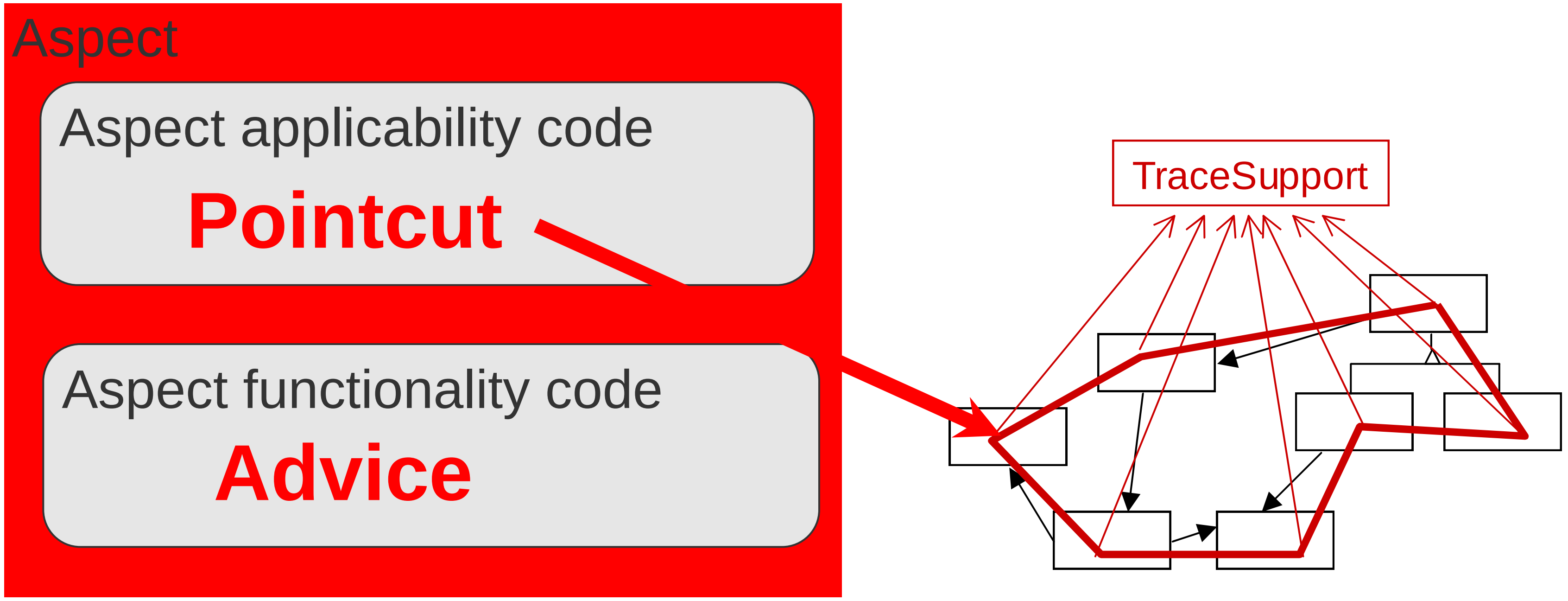
A join point model defines the kinds of join points available and how they are accessed and used.

- Specific to each aspect-oriented programming language
- E.g. AspectJ join point model:
  key points in dynamic call graph

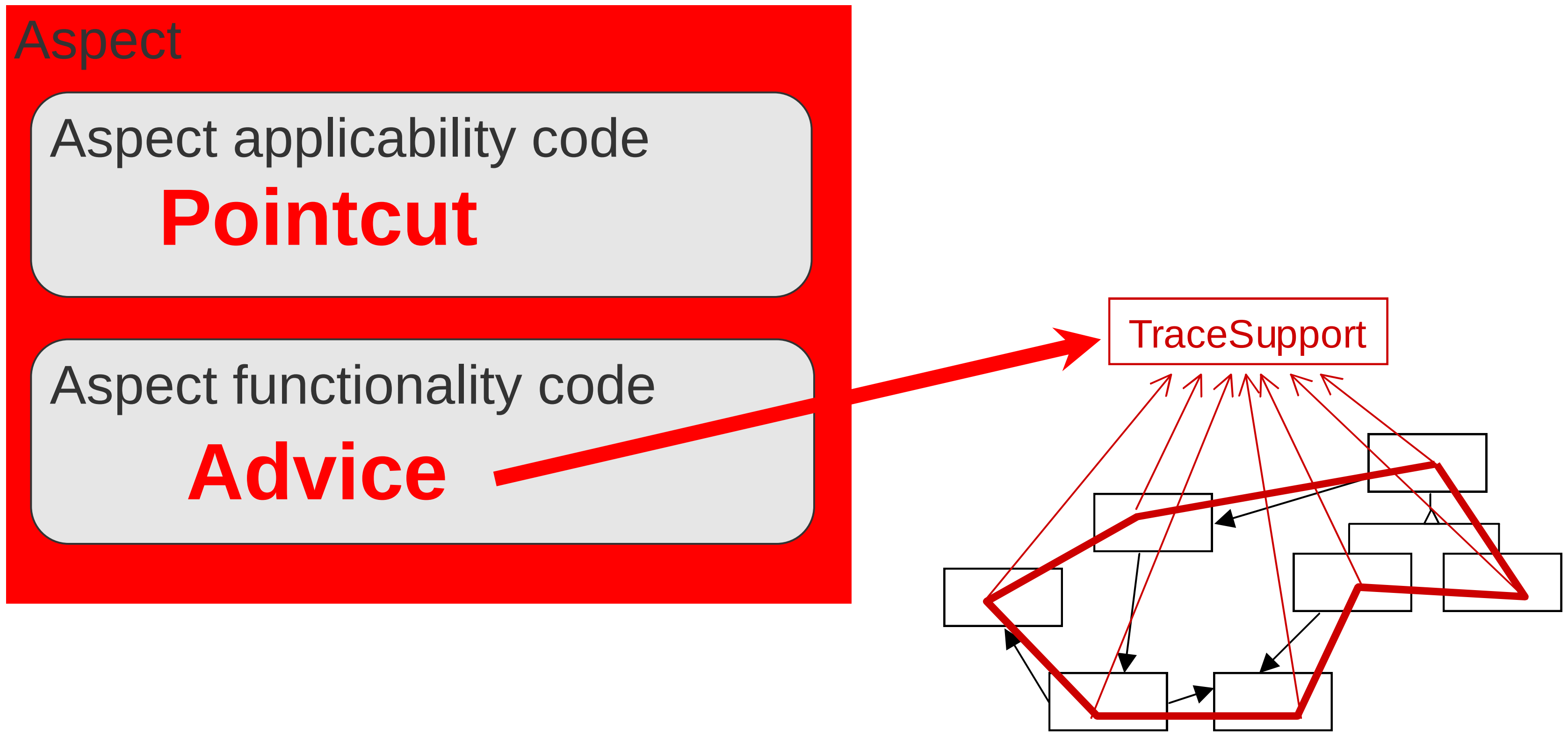

a Line
dispatch
a method execution
a method call returning or throwing
a Point
dispatch
a method execution returning or throwing

Pointcuts

A pointcut is a predicate that matches join points. A pointcut is a relationship 'join point -> boolean', where the domain of the relationship is all possible join points.



Advice


Aspect
Aspect applicability code
Pointcut
Aspect functionality code
Advice
TraceSupport

Example: Synchronised Buffer

```
class Buffer {
  char[] data;
  int nrOfElements;
  Semaphore sema;

  bool isEmpty() {
    bool returnVal;
    sema.writeLock();
    returnVal := nrOfElements == 0;
    sema.unlock();
    return returnVal;
  }
}
```

Buffer functionality concern

Synchronisation concern

Tangling!
Crosscutting concerns!

Synchronisation as an Aspect

When a Buffer object receives the message isEmpty, first make sure the object is not being accessed by another thread through the get or put methods

When a Buffer object receives the message isEmpty, first make sure the object is not being accessed by another thread through the get or put methods

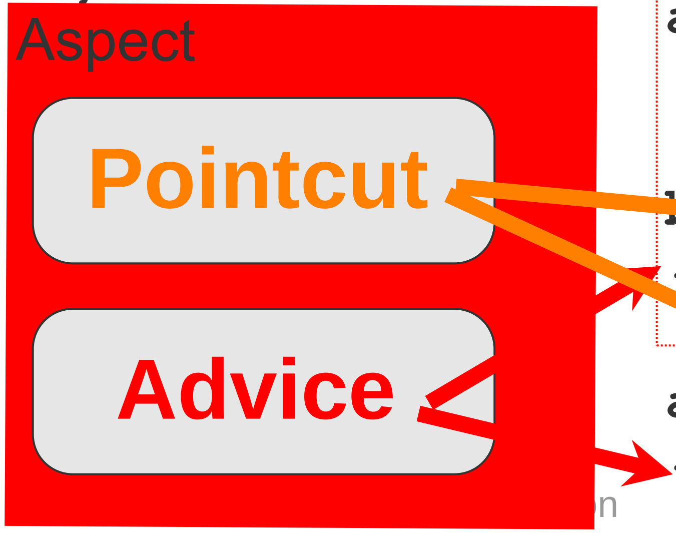
<u>When</u> to execute the aspect (pointcut)

Composition of <u>when</u> and <u>what</u> (kind of advice)

<u>What</u> to do at the join point (advice)

Synchronisation as an Aspect

```
class Buffer {
   char[] data;
   int nrOfElements;

   bool isEmpty() {
      bool returnVal;
      returnVal := nrOfElements == 0;
      return returnVal;
   }
}
```



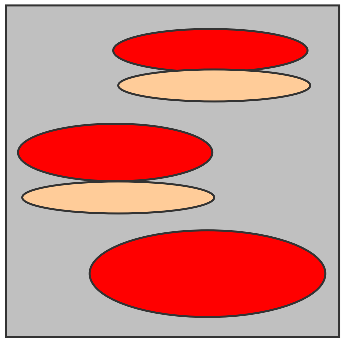
```
aspect Syncronizer
   Semaphore sema;

before: reception(Buffer.isEmpty)
{ sema.writeLock();}

after: reception(Buffer.isEmpty)
{ sema.unlock(); }
```

Types of Decompositions

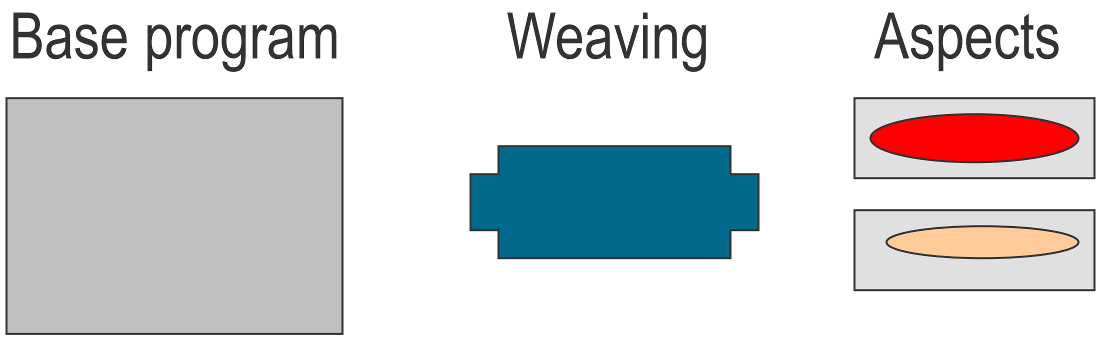
Asymmetric	Symmetric
Crosscutting concerns modularized in special module (aspect).	All concerns modularized in same kind of module

Described until now

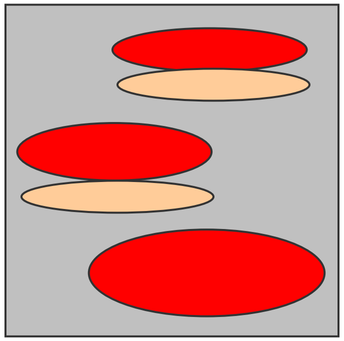
Asymmetric Style (AspectJ)



describe crosscutting concerns
as separate, independent entities
and weave them with the base program

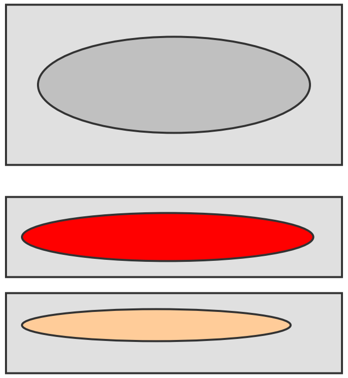


Symmetric Style (HyperJ)

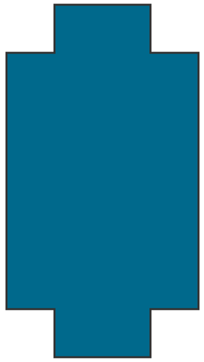


describe different concerns
as separate program fragments
and weave them

Program Fragments

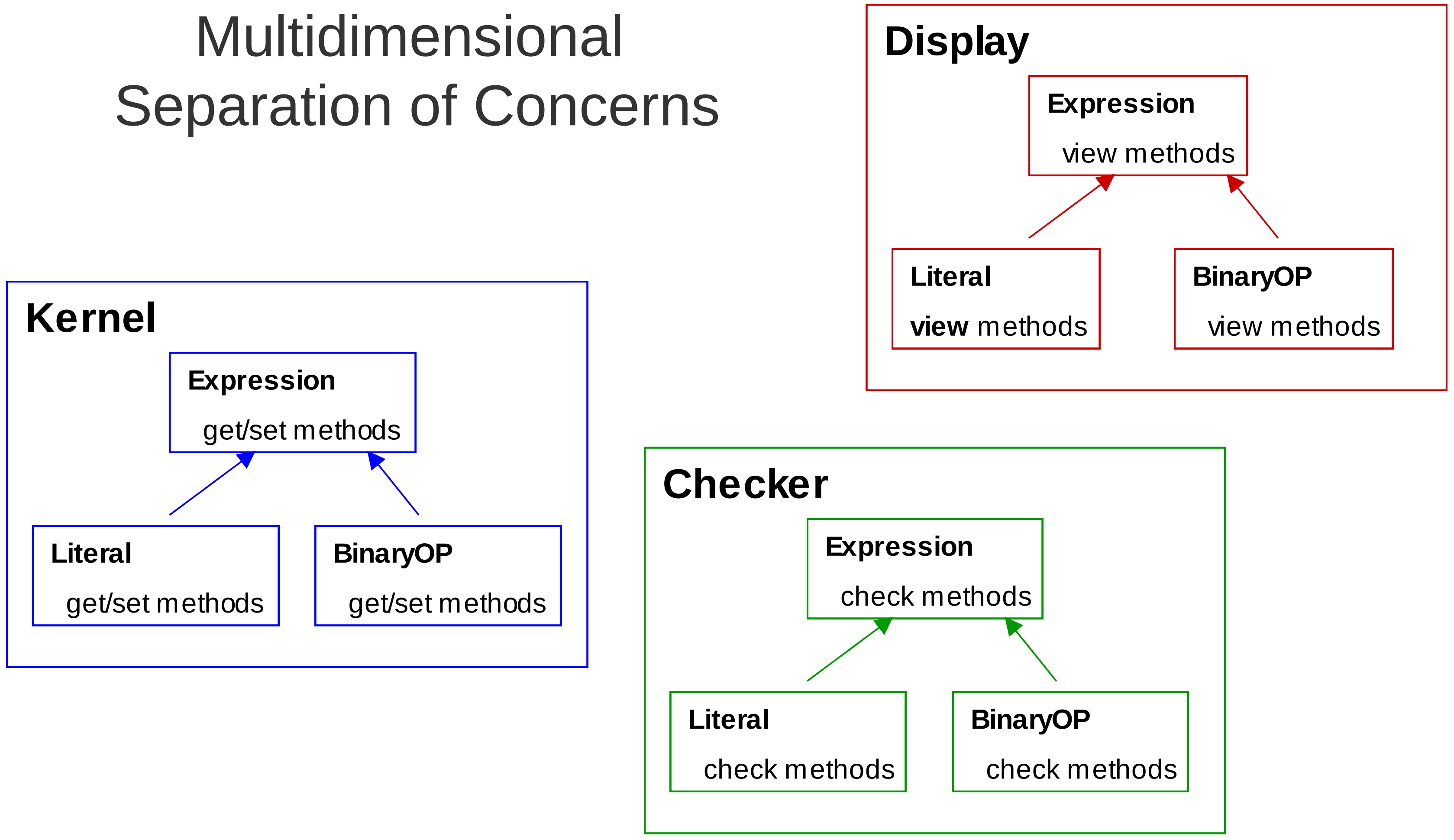


Weaving



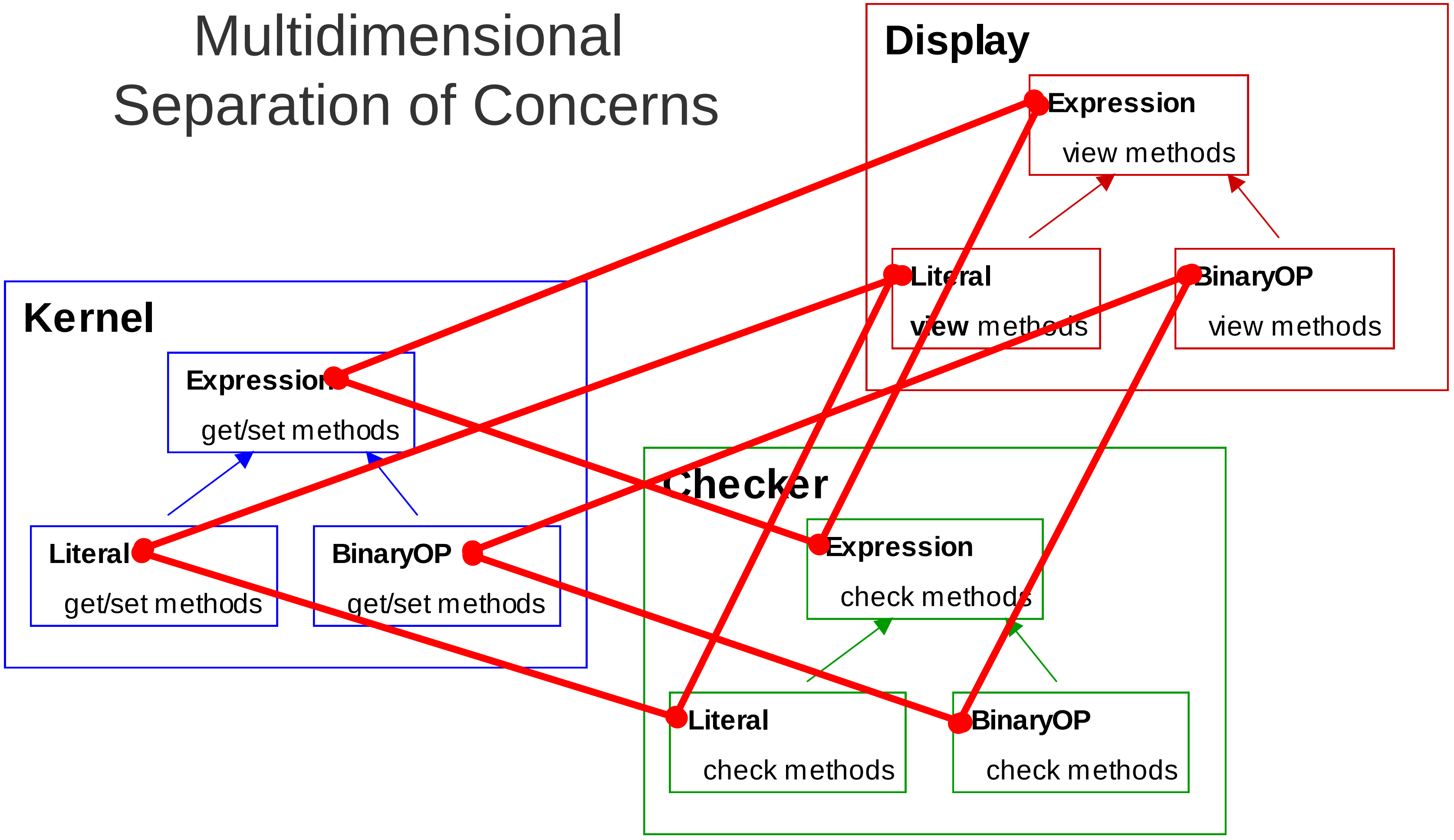
Symmetric Approaches

Multidimensional
Separation of Concerns

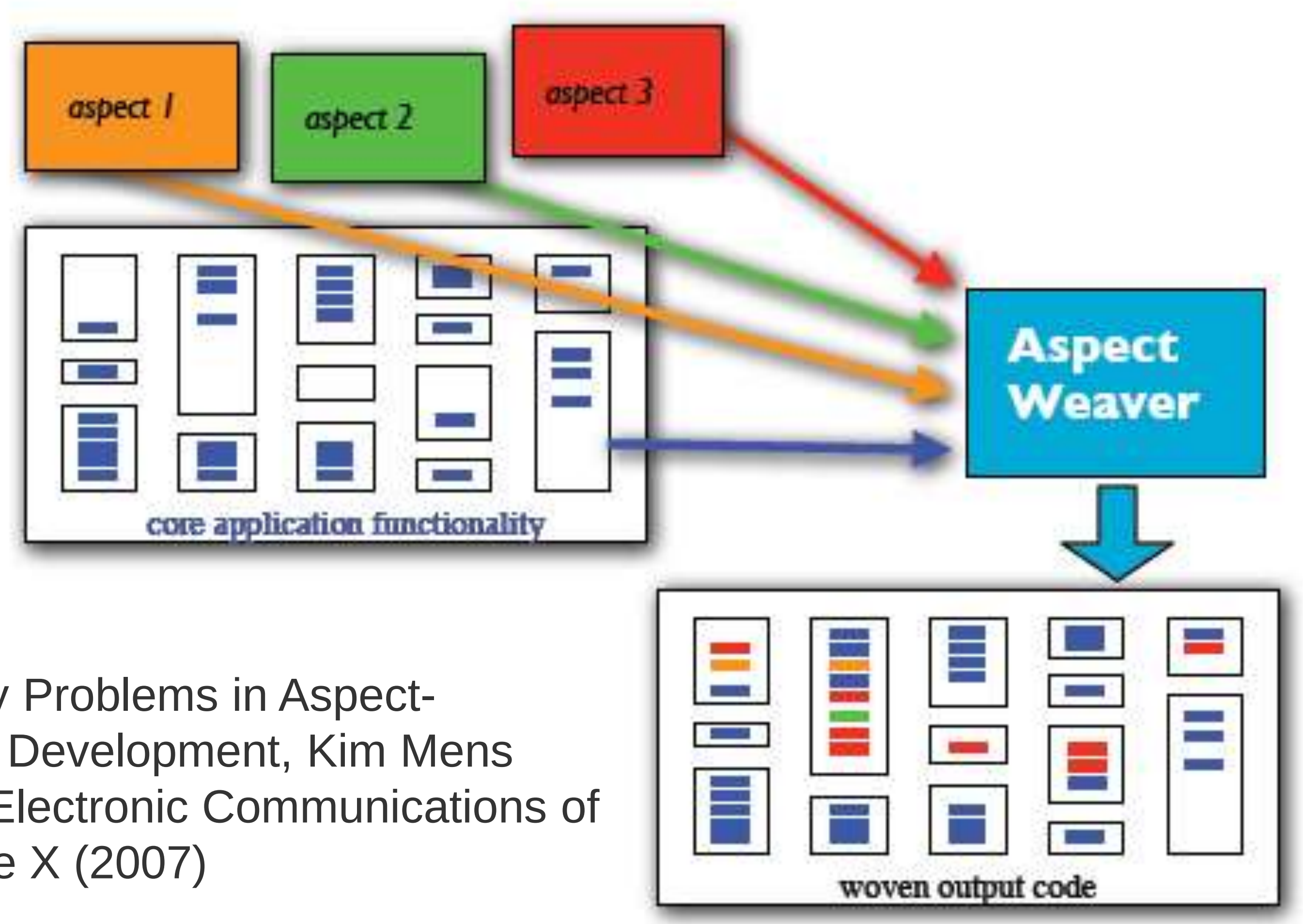


Symmetric Approaches

Multidimensional Separation of Concerns



Execution



From: Evolutionary Problems in Aspect-Oriented Software Development, Kim Mens and Tom Tourwé, Electronic Communications of the EASST Volume X (2007)