AOP Framed!

Henrique Rebêlo

Informatics Center

Federal University of Pernambuco

Contacting Me…

Ph.D. student Henrique Rebêlo

Specialist on **AOSD**, **DbC**, Static Metrics, **JML**

Software Architecture, Product Lines, Empirical Studies

email: **hemr@cin.ufpe.br**

Informatics Center

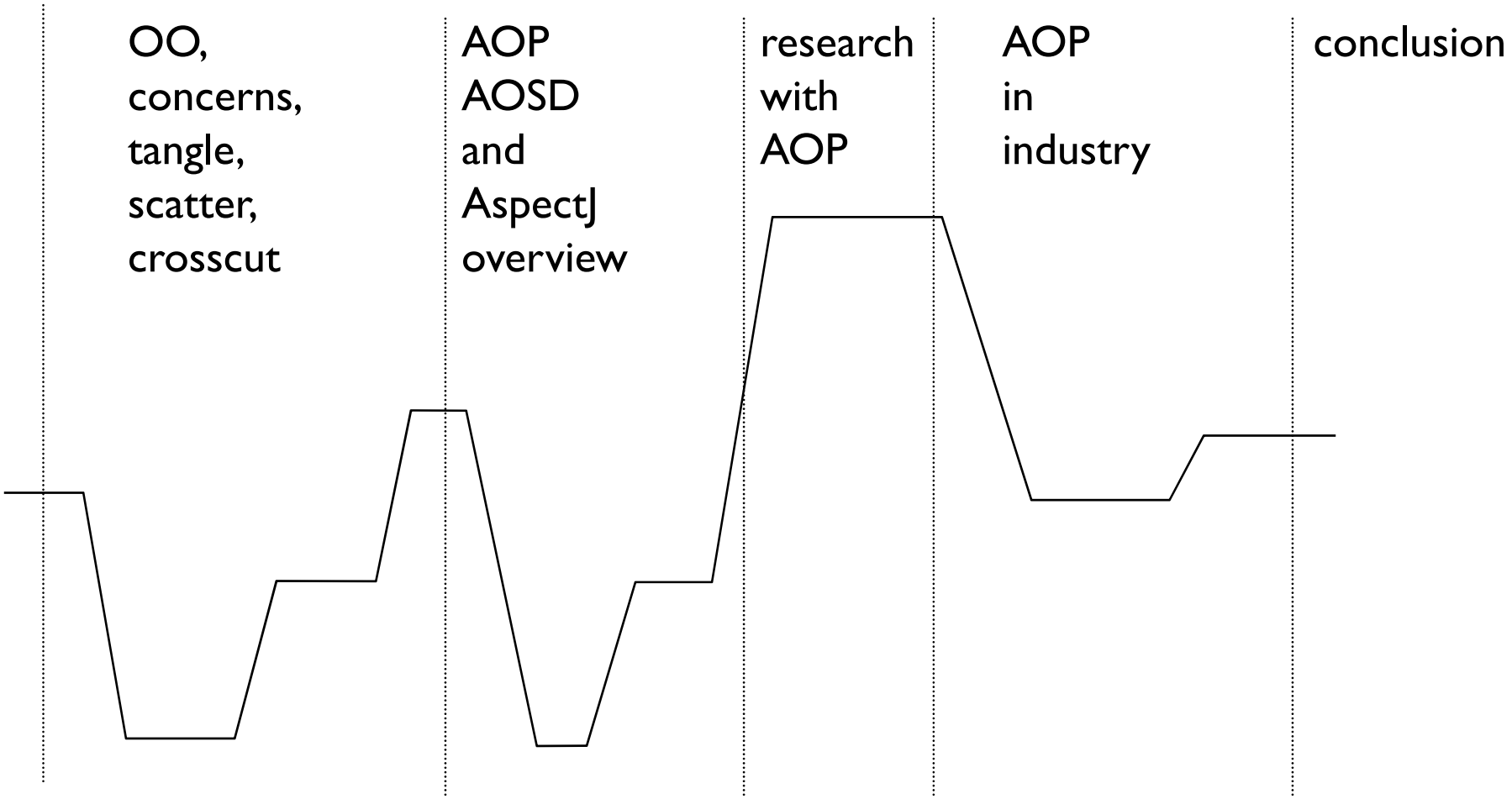
Federal University of Pernambuco, UFPE

Brazil

Talk Material - Contributions

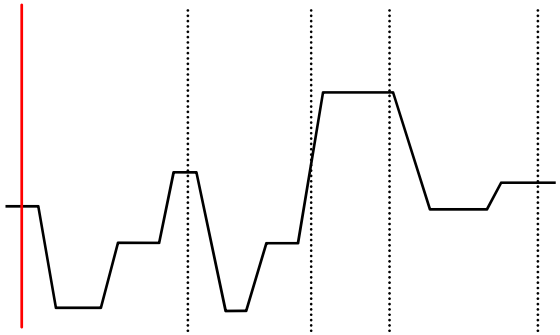
- Prof. Paulo Borba (expert on aspect-oriented languages and refactoring)
- Prof. Sérgio Soares (expert on aspect-oriented software development)
- Prof. Alessandro Garcia (expert on aspect-oriented software assessment metrics)
- Prof. Fernando Castor Filho (expert on exception handling)

talk timeline



OO scatter tangle crosscut and all that

exploring some of the terms we use

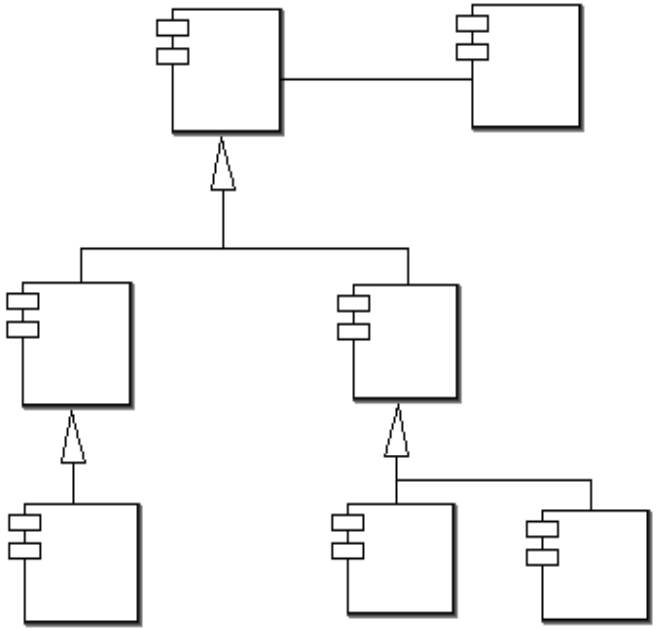


Object-Oriented Paradigm (OOP)

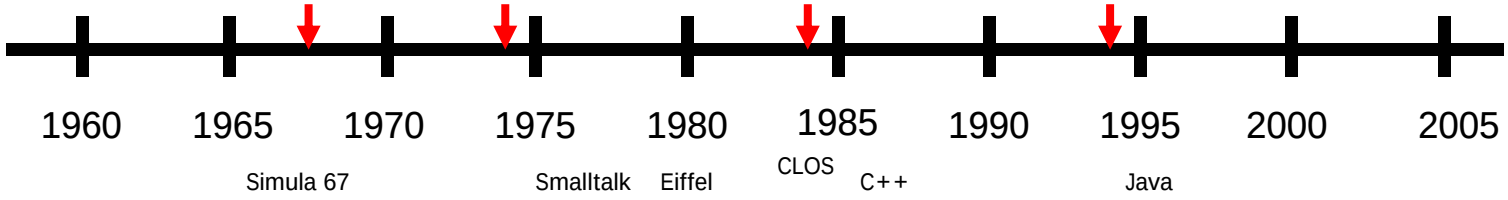
- object –oriented decomposition

- modularization
  - objects
  - classes

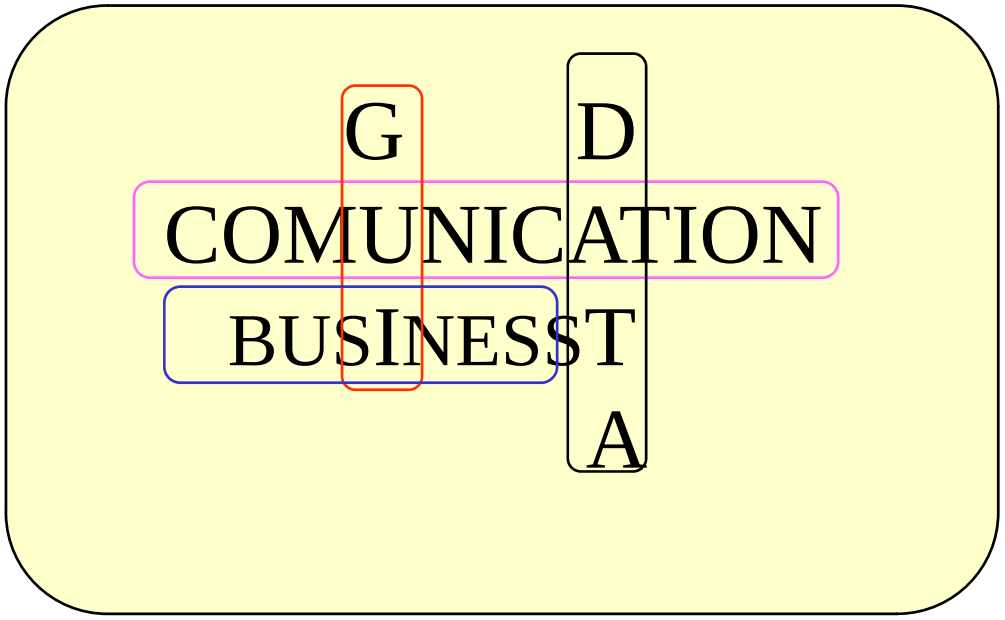
  - encapsulated data and procedures



problem: ?

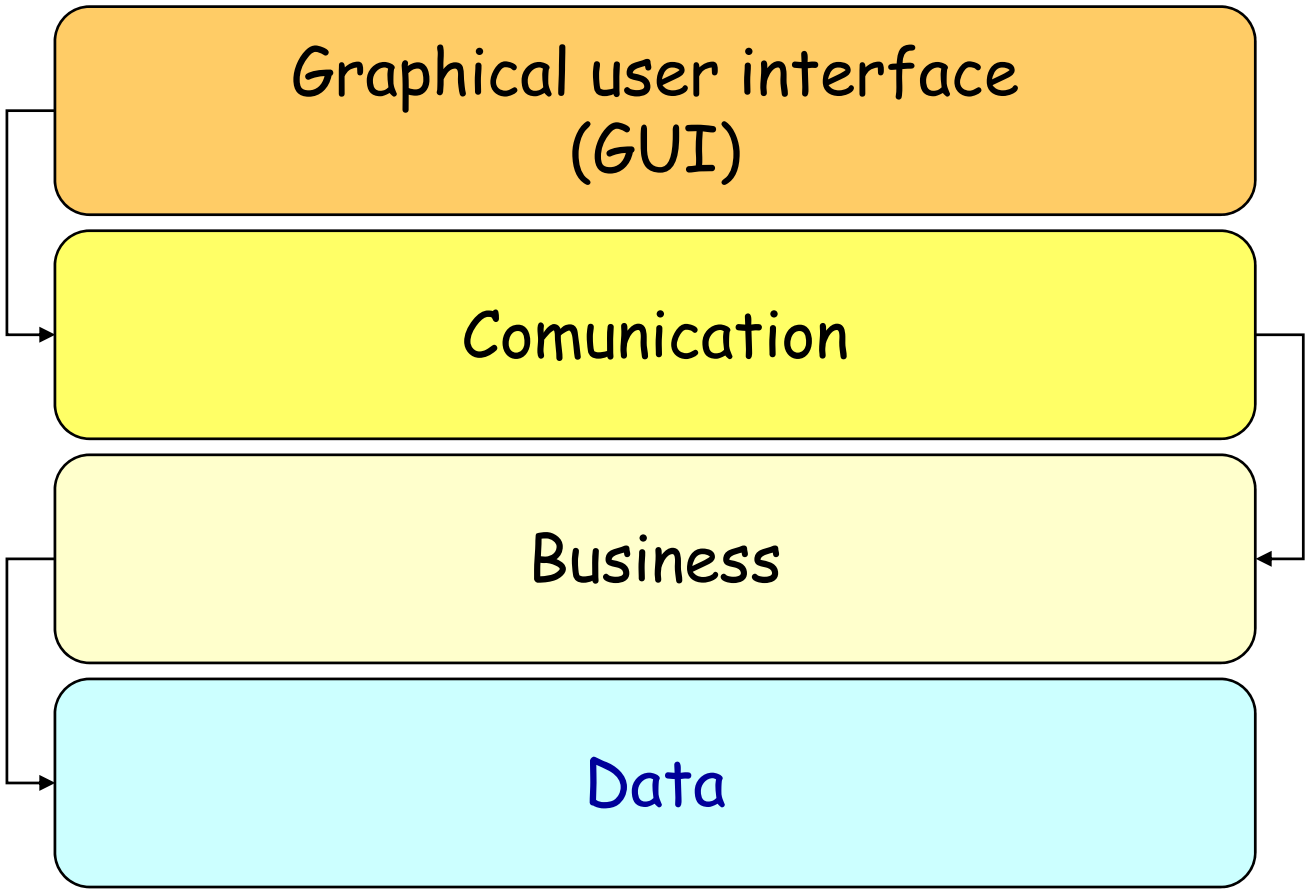


Bad OO Design



Problem: tangling of code with different purposes

Good OO Design

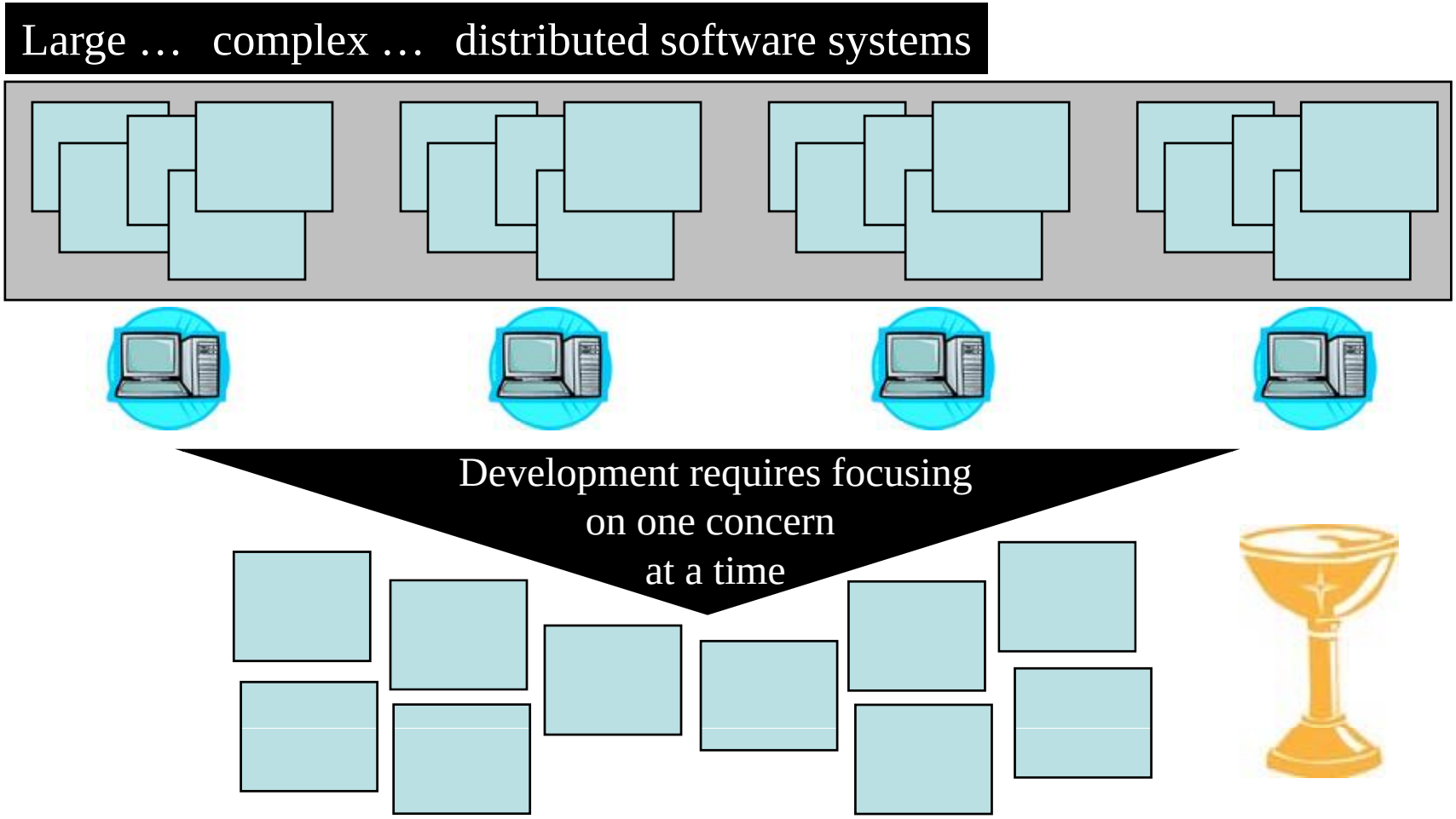


Layered Architecture

- The various kinds of code should be written separately
- Separation of
  - concepts
  - concerns
  - kinds of code

separation of concerns

Need for Separation of Concerns



Separation of Concerns

- A *concern* is a *“specific requirement or consideration that must be addressed in order to satisfy the overall system goal”*

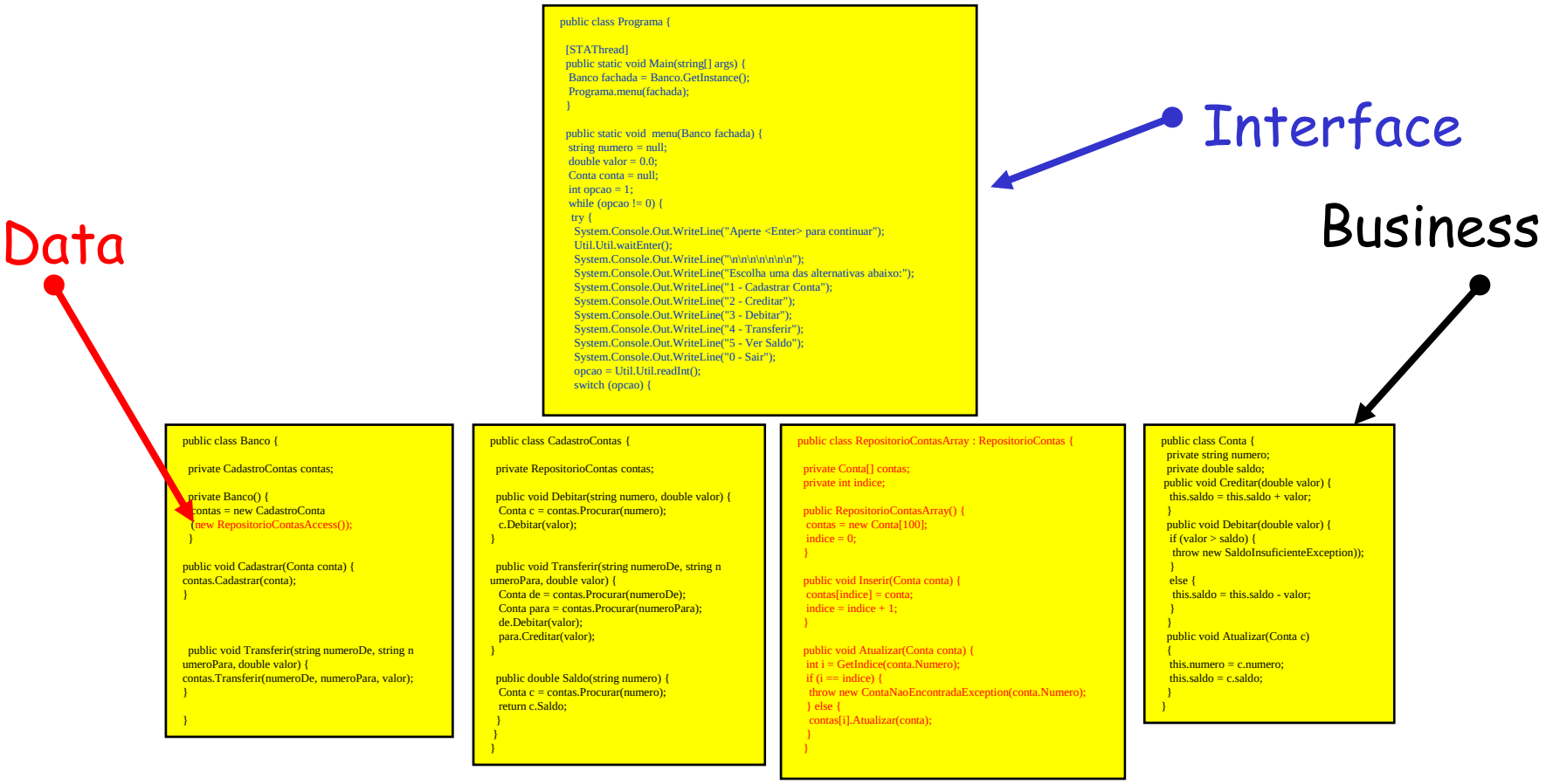
- The principle was coined by **Edsger Dijkstra**
  - 1974 paper "On the Role of Scientific Thought"
  - he received the 1972 *A. M. Turing Award* for fundamental contributions in the area of programming languages

Effective Separation of Concerns improve...

Quality attributtes

- stability
- evolvability
- composability
- reliability
- reusability
- traceability
- maintainability
- testability
- ...

Good modularization without persistence, distribution, ...



We have problems with persistence using JDBC

```
public class Banco {

  private CadastroContas contas;

  private Banco() {
   contas = new CadastroConta
   (new RepositorioContasAccess());
  }

 public void Cadastrar(Conta conta) {
    Persistence.DBHandler.StartTransaction();
    try {
     contas.Cadastrar(conta);
     Persistence.DBHandler.CommitTransaction();
    } catch (System.Exception ex){
     Persistence.DBHandler.RollBackTransaction();
     throw ex;
    }
  }



   public void Transferir(string numeroDe, string n
 umeroPara, double valor) {
    Persistence.DBHandler.StartTransaction();
    try {
     contas.Transferir(numeroDe, numeroPara, valor);
     Persistence.DBHandler.CommitTransaction();
    } catch (System.Exception ex){
     Persistence.DBHandler.RollBackTransaction();
     throw ex;
    }
  }

 }
```

```
public class CadastroContas {

  private RepositorioContas contas;

  public void Debitar(string numero, double valor) {
   Conta c = contas.Procurar(numero);
   c.Debitar(valor);
   contas.Atualizar(c);
  }

  public void Transferir(string numeroDe, string n
umeroPara, double valor) {
   Conta de = contas.Procurar(numeroDe);
   Conta para = contas.Procurar(numeroPara);
   de.Debitar(valor);
   para.Creditar(valor);
   contas.Atualizar(de);
   contas.Atualizar(para);
  }

  public double Saldo(string numero) {
   Conta c = contas.Procurar(numero);
   return c.Saldo;
  }
 }
```

```
public class RepositorioContasAccess : RepositorioContas {

  public void Inserir(Conta conta) {
   string sql = "INSERT INTO Conta (NUMERO,SALDO)
 VALUES ('" + conta.Numero + "'," + conta.Saldo + ")";
   OleDbCommand insertCommand = new OleDbCommand
 (sql,DBHandler.Connection,DBHandler.Transaction);
   insertCommand.ExecuteNonQuery();
  }

  public void Atualizar(Conta conta) {
   string sql = "UPDATE Conta SET SALDO = (" + conta.As
 ldo + ") WHERE NUMERO = '" + conta.Numero + "'";
   OleDbCommand updateCommand = new OleDbCommand(s
 ql,DBHandler.Connection,DBHandler.Transaction);
   int linhasAfetadas;
   linhasAfetadas = updateCommand.ExecuteNonQuery();
   if (linhasAfetadas == 0) {
    throw new ContaNaoEncontradaException(conta.Numero);
   }
  }

 }
```

```
public class Conta {
  private string numero;
  private double saldo;
 public void Creditar(double valor) {
   this.saldo = this.saldo + valor;
  }
  public void Debitar(double valor) {
   if (valor > saldo) {
    throw new SaldoInsuficienteException));
   }
   else {
    this.saldo = this.saldo - valor;
   }
  }
  public void Atualizar(Conta c)
  {
   this.numero = c.numero;
   this.saldo = c.saldo;
  }
 }
```

```
public class DBHandler {
  private static OleDbConnection connection;
  public static OleDbConnection Connection {
   get {
    if (connection == null) {
     string dataSource = "BancoCS.mdb";
     string strConexao =
      "Provider= Microsoft.Jet.OLEDB.4.0; " +
      "Data Source=" + dataSource;
     connection = new OleDbConnection(strConexao);
    }
    return connection;
   }
  }


  public static OleDbConnection GetOpenConnection() {
   Connection.Open();
   return Connection;
  }

  public static void StartTransaction() {
   Connection.Open();
   transaction = Connection.BeginTransaction();
  }

 }
```

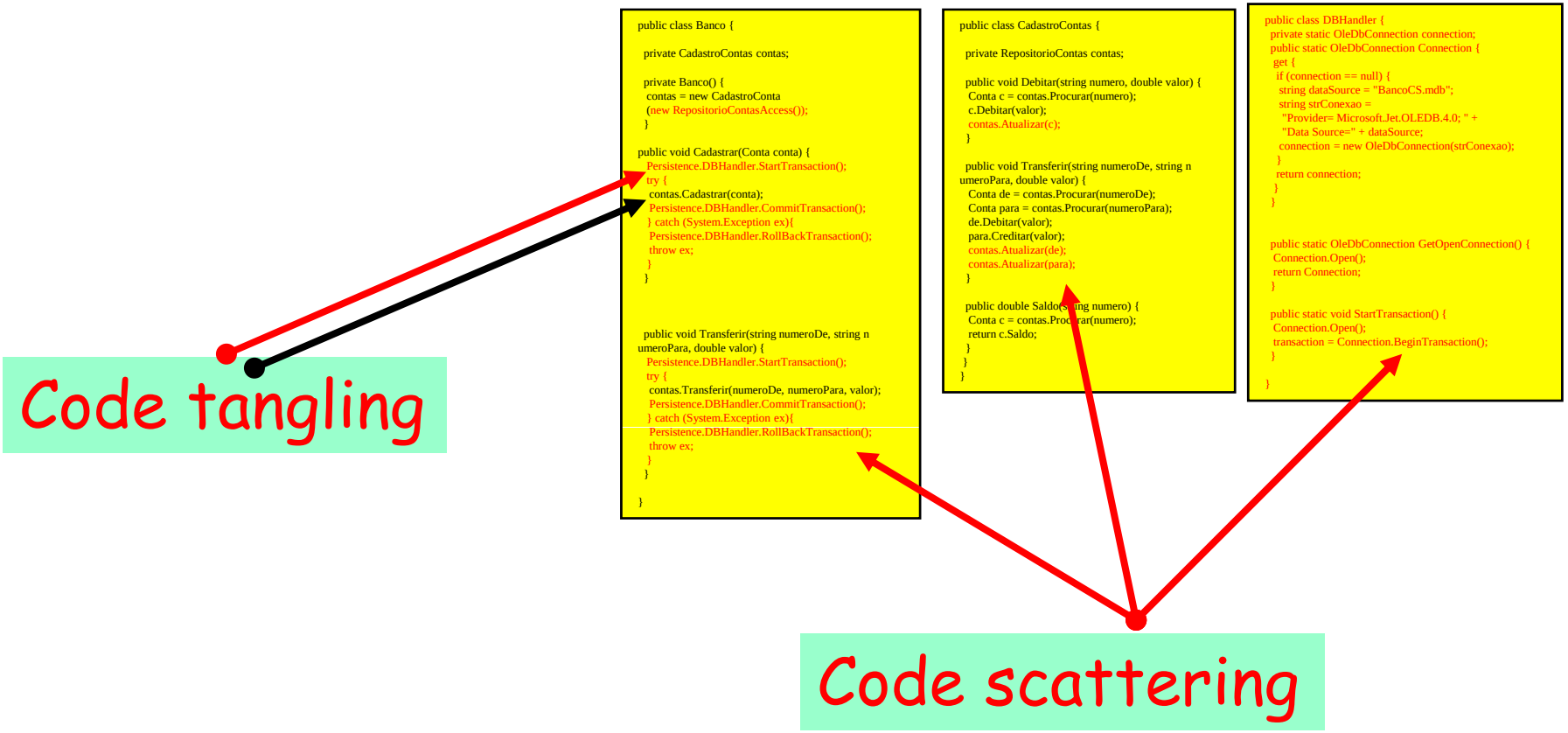
JDBC code in red...

What is the problem?

- Bad modularization of system-wide, peripheral concerns at the implementation level

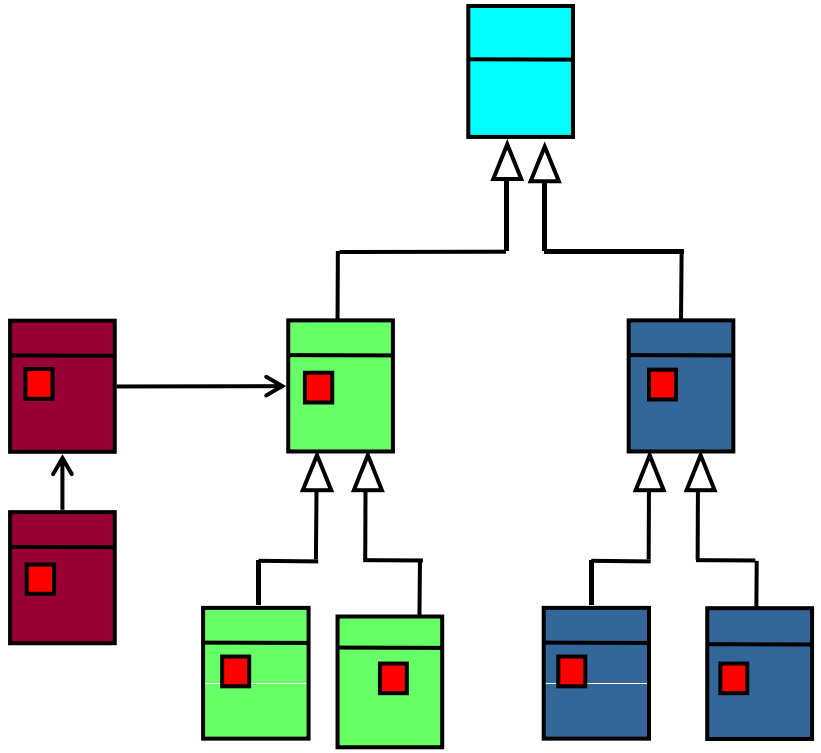
- Symptoms
  - Code tangling
  - Code scattering

Problems with OO implementation



Problem with OO

- Code related to **such** concerns are crosscutting
- Scattered across several modules of implementation (classes)



Each color is one different concern

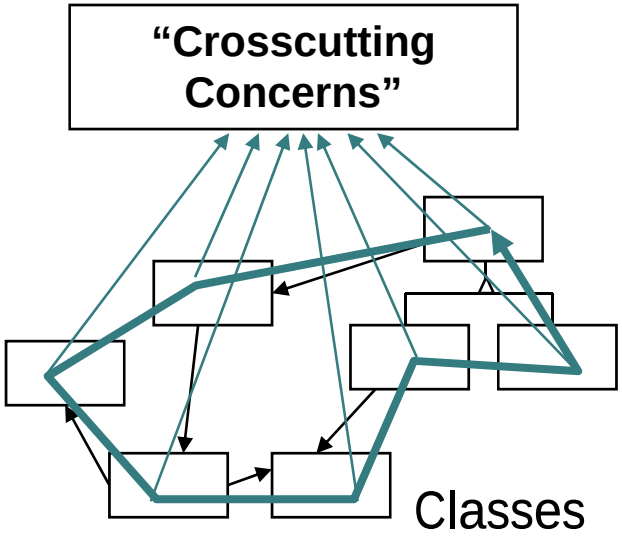
More OO Problems

- Redundance
  - several similar code fragments in several places of the source code
- Weak cohesion
  - methods of the affected classes contain instructions that are not directed related to the core concern in which they implement
- Strong coupling
  - methods of the affected classes need to be aware of methods from classes which implement the scattered funcionality

The Problem of Crosscutting Concerns

- Certain properties naturally have global, pervasive effects over the system decompositions
  - examples: exception handling, distribution, **persistence**…

OO Decomposition



- Hamper software reusability, maintainability, and the like
- Domain-specific programming languages started to emerge...

Persistence is a crosscutting concern

```
public class Banco {

  private CadastroContas contas;

  private Banco() {
   contas = new CadastroConta
   (new RepositorioContasAccess());
  }

 public void Cadastrar(Conta conta) {
   Persistence.DBHandler.StartTransaction();
   try {
    contas.Cadastrar(conta);
    Persistence.DBHandler.CommitTransaction();
   } catch (System.Exception ex){
    Persistence.DBHandler.RollBackTransaction();
    throw ex;
   }
  }

  public void Transferir(string numeroDe, string n
umeroPara, double valor) {
   Persistence.DBHandler.StartTransaction();
   try {
    contas.Transferir(numeroDe, numeroPara, valor);
    Persistence.DBHandler.CommitTransaction();
   } catch (System.Exception ex){
    Persistence.DBHandler.RollBackTransaction();
    throw ex;
   }
  }

 }
```

```
public class CadastroContas {

  private RepositorioContas contas;

  public void Debitar(string numero, double valor) {
   Conta c = contas.Procurar(numero);
   c.Debitar(valor);
   contas.Atualizar(c);
  }

  public void Transferir(string numeroDe, string n
umeroPara, double valor) {
   Conta de = contas.Procurar(numeroDe);
   Conta para = contas.Procurar(numeroPara);
   de.Debitar(valor);
   para.Creditar(valor);
   contas.Atualizar(de);
   contas.Atualizar(para);
  }

  public double Saldo(string numero) {
   Conta c = contas.Procurar(numero);
   return c.Saldo;
  }
 }
}
```

```
public class RepositorioContasAccess : RepositorioContas {

  public void Inserir(Conta conta) {
   string sql = "INSERT INTO Conta (NUMERO,SALDO)
VALUES ('" + conta.Numero + "'," + conta.Saldo + ")";
   OleDbCommand insertCommand = new OleDbCommand
(sql,DBHandler.Connection,DBHandler.Transaction);
   insertCommand.ExecuteNonQuery();
  }

  public void Atualizar(Conta conta) {
   string sql = "UPDATE Conta SET SALDO = (" + conta.As
ldo + ") WHERE NUMERO = '" + conta.Numero + "'";
   OleDbCommand updateCommand = new OleDbCommand(s
ql,DBHandler.Connection,DBHandler.Transaction);
   int linhasAfetadas;
   linhasAfetadas = updateCommand.ExecuteNonQuery();
   if (linhasAfetadas == 0) {
    throw new ContaNaoEncontradaException(conta.Numero);
   }
  }

}
```

```
public class Conta {
  private string numero;
  private double saldo;
 public void Creditar(double valor) {
   this.saldo = this.saldo + valor;
  }
  public void Debitar(double valor) {
   if (valor > saldo) {
    throw new SaldoInsuficienteException));
   }
   else {
    this.saldo = this.saldo - valor;
   }
  }
  public void Atualizar(Conta c)
  {
   this.numero = c.numero;
   this.saldo = c.saldo;
  }
}
```

```
public class DBHandler {
  private static OleDbConnection connection;
  public static OleDbConnection Connection {
   get {
    if (connection == null) {
     string dataSource = "BancoCS.mdb";
     string strConexao =
      "Provider= Microsoft.Jet.OLEDB.4.0; " +
      "Data Source=" + dataSource;
     connection = new OleDbConnection(strConexao);
    }
    return connection;
   }
  }
  public static OleDbConnection GetOpenConnection() {
   Connection.Open();
   return Connection;
  }

  public static void StartTransaction() {
   Connection.Open();
   transaction = Connection.BeginTransaction();
  }

 }
```

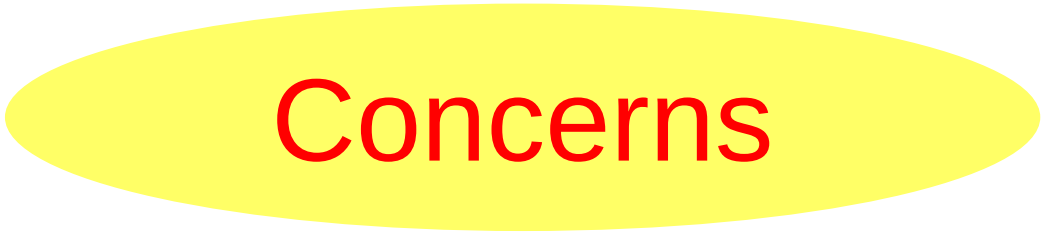
Persistence code in red…

crosscuts business

Crosscutting concerns...

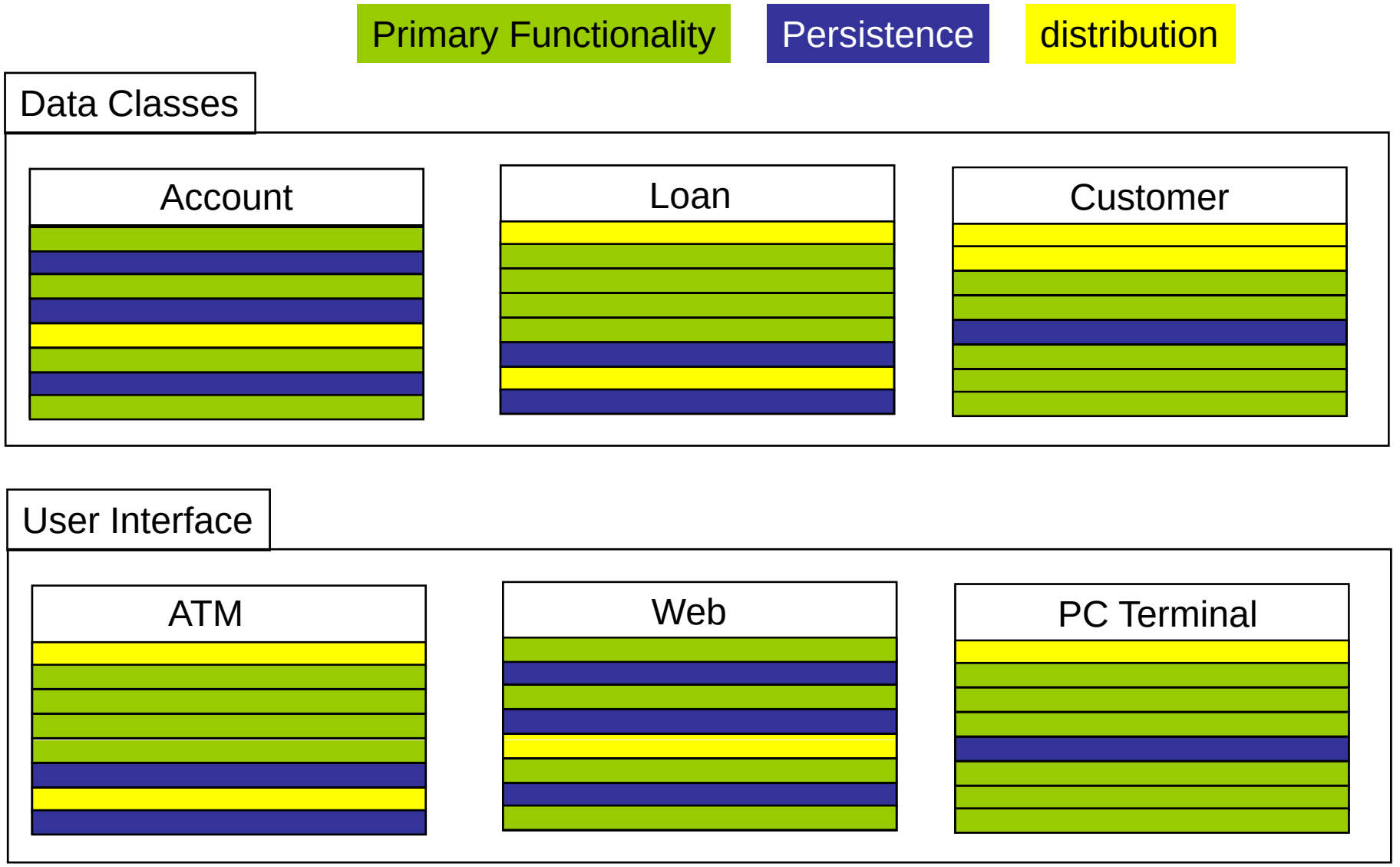
cannot be properly modularized by OO constructs because their realization affects the behavior of several methods, possibly in several classes, cutting across class structures

Tangling and scattering of...

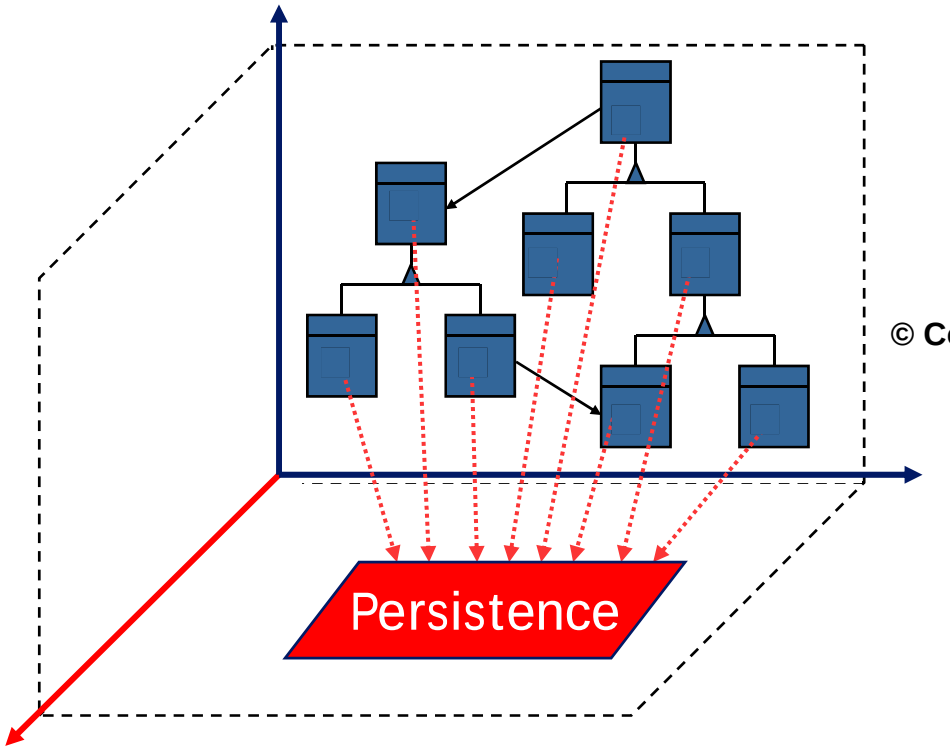


- Persistence
- Concurrency control
- Logging
- Security
- Events
- Use cases
- Performance
- Design by Contract
- ...

Generalizing the Problem…



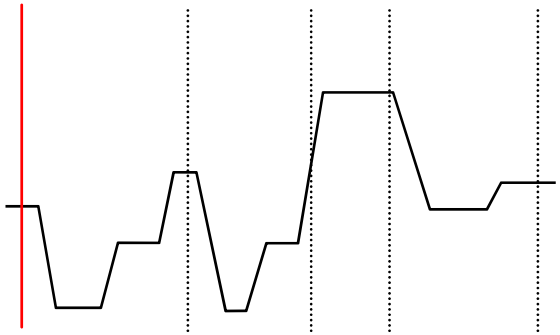
Root of Cause



- The concern is crosscutting to the dimension of the implementation

AOP AOSD and AspectJ overview

exploring another way to promote modularization



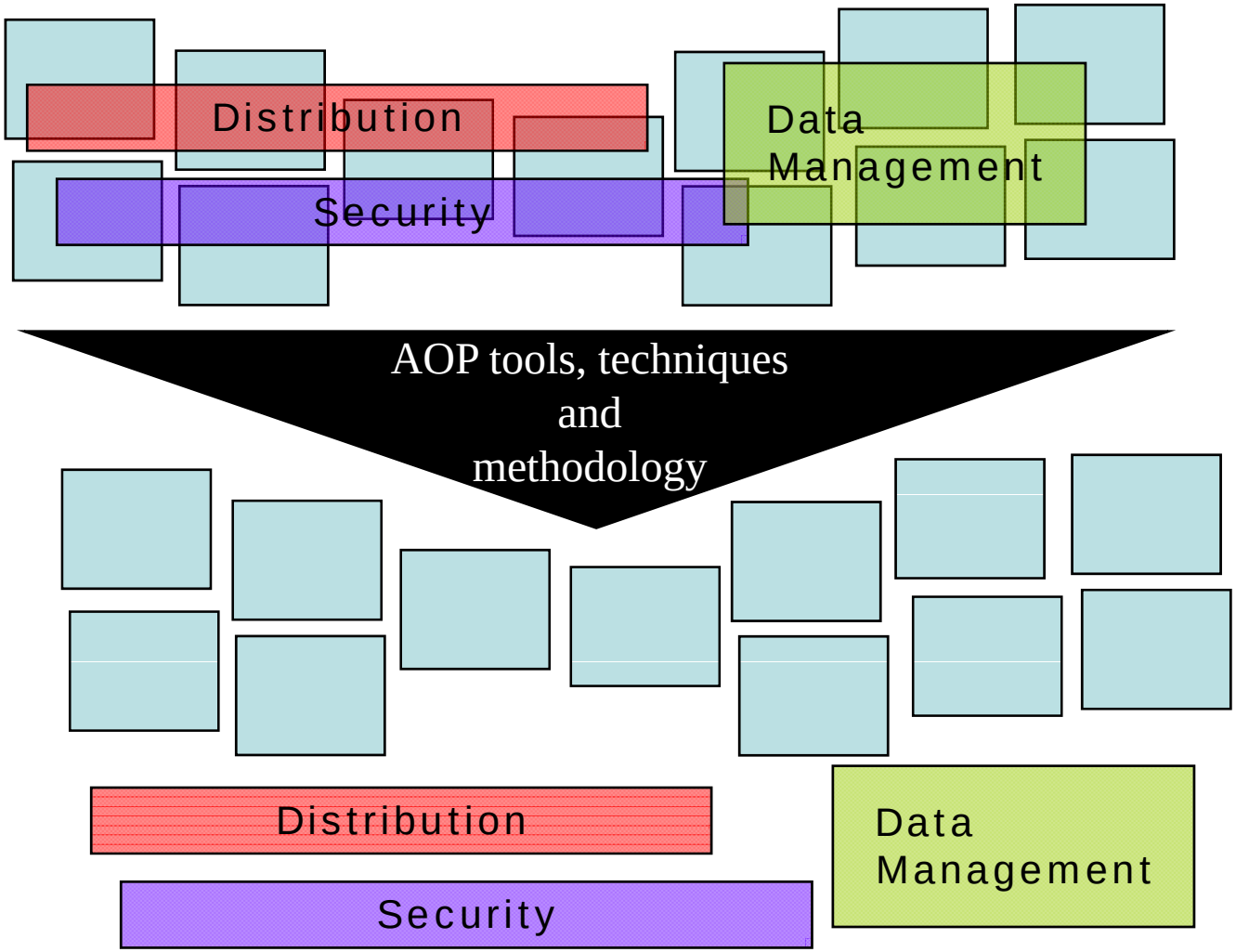
Aspect-oriented languages are quite popular...

due to the promise of

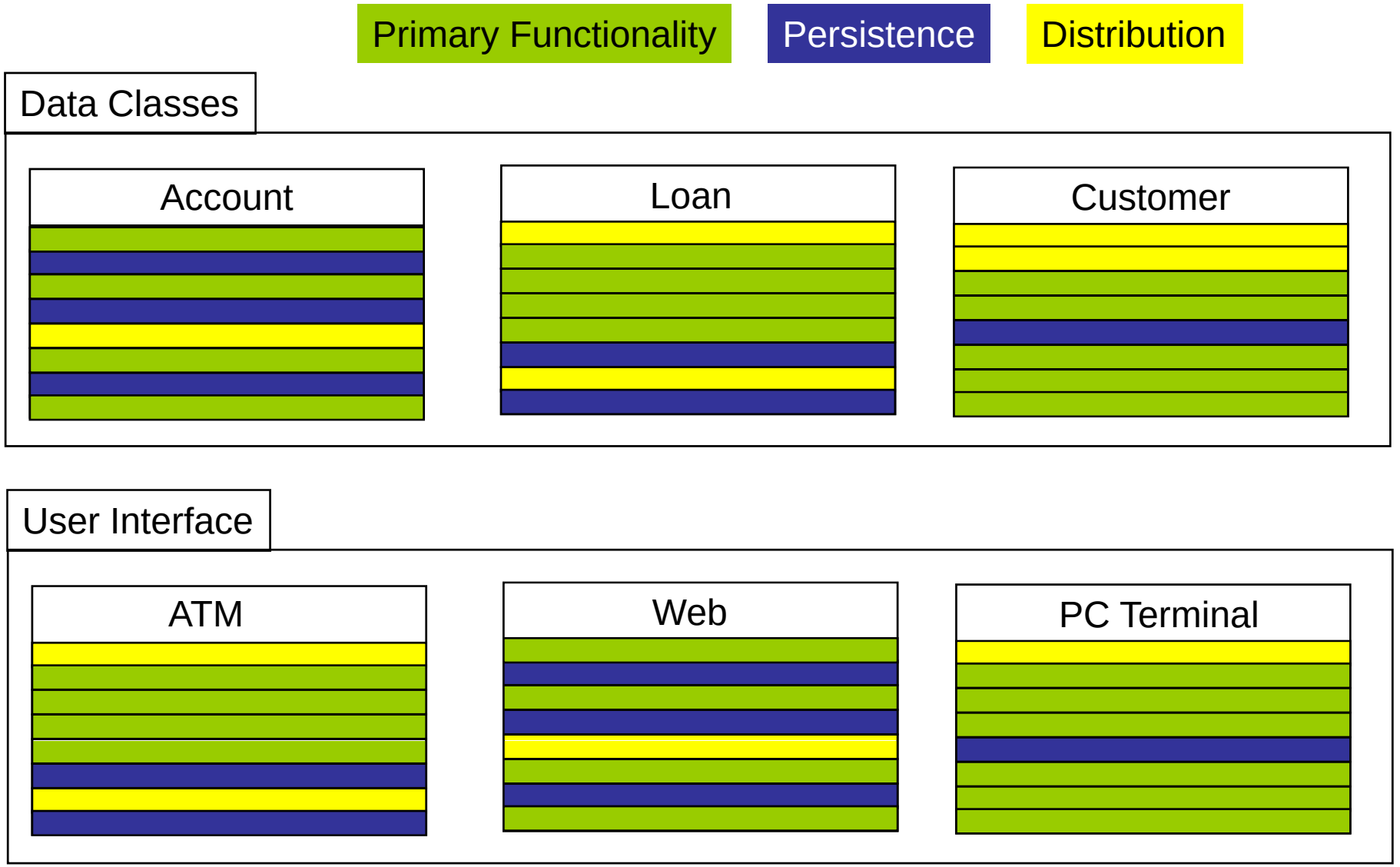
modularizing

crosscutting concerns

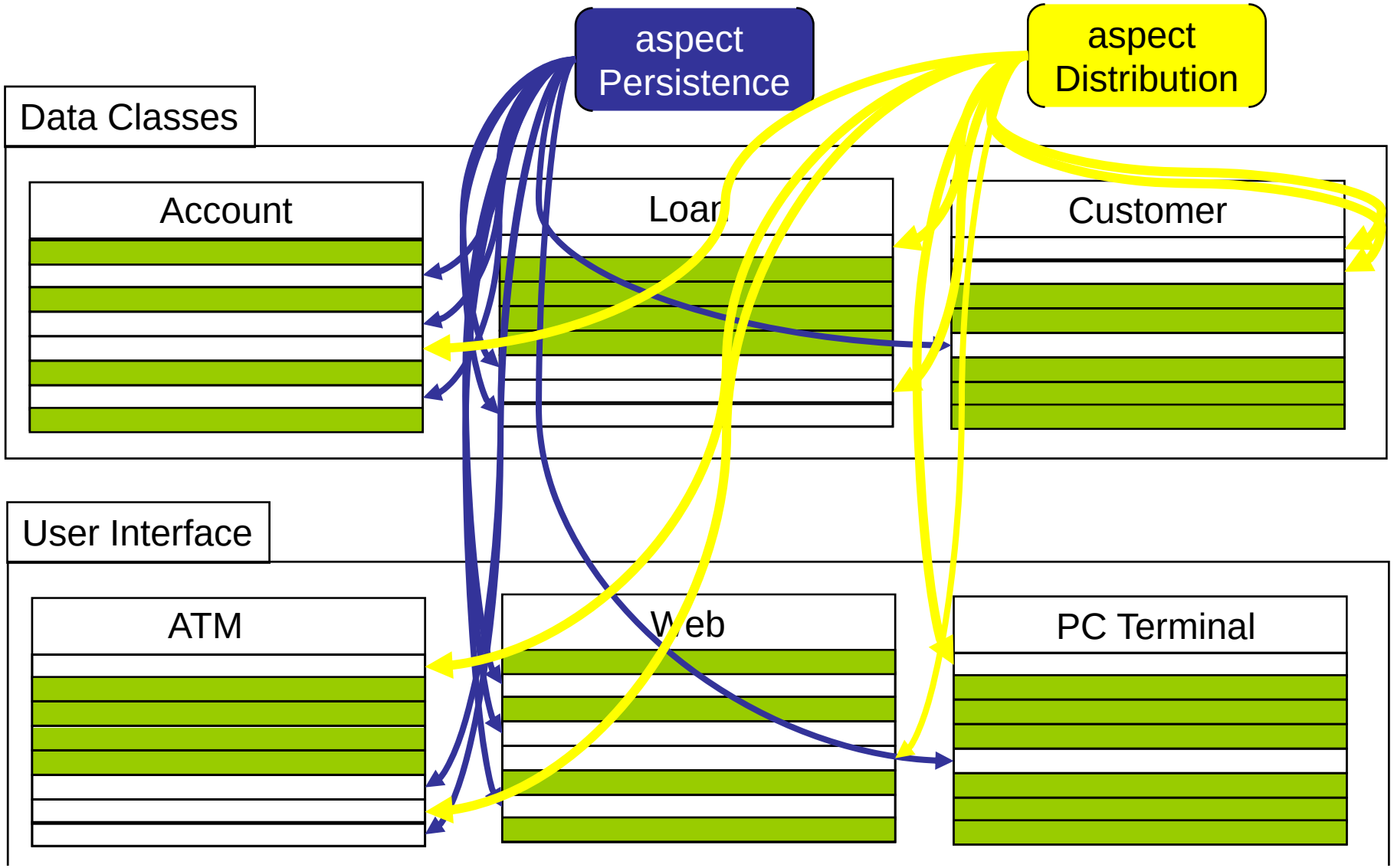
Aspect-Oriented Programming (AOP)



Revisiting the Example



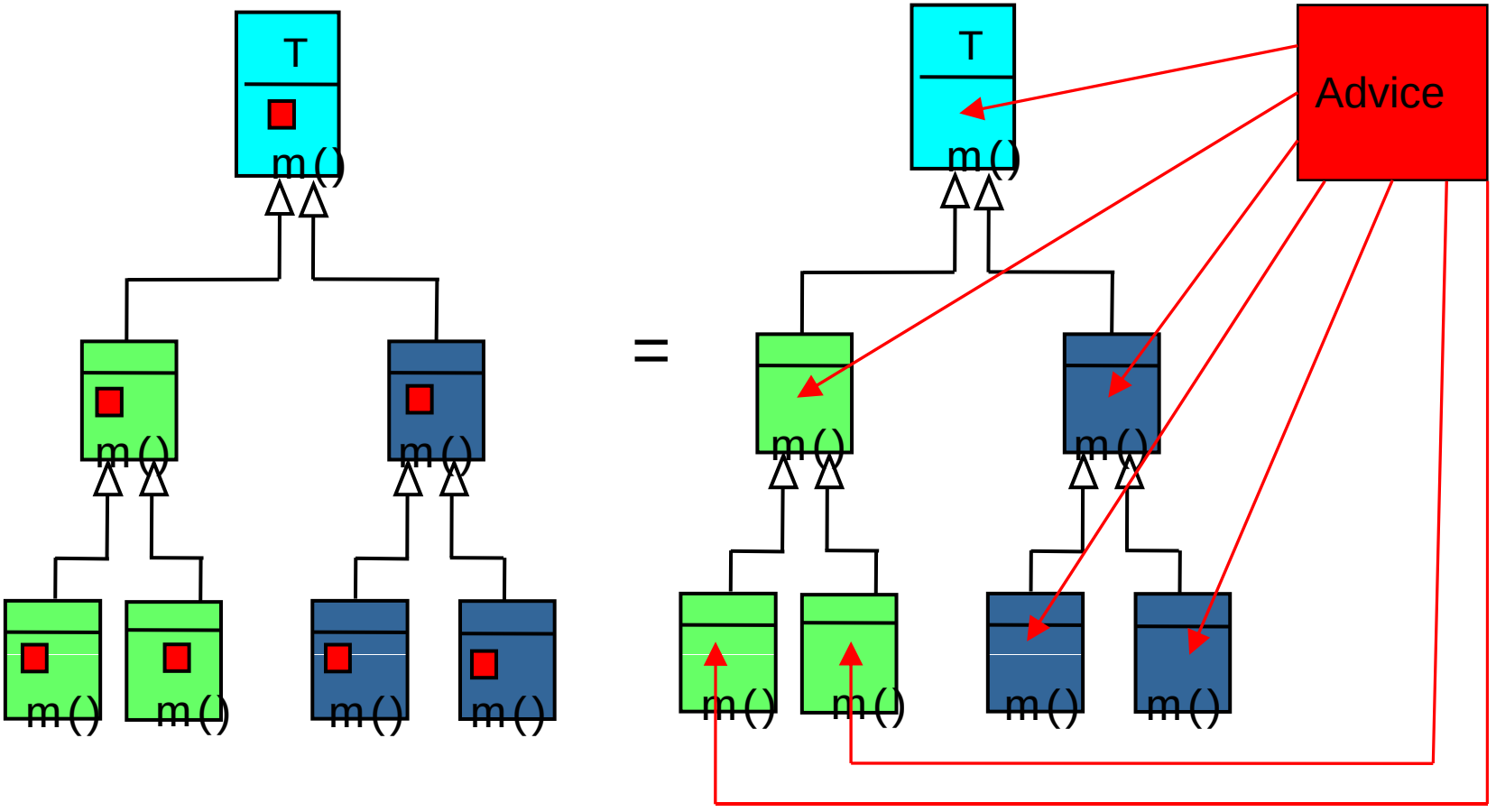
Wouldn’t it be Nice if …



From your experience…

- Which other system concerns are “typically” crosscutting?

Quantification is the basic concept of AOP



The success of AOP

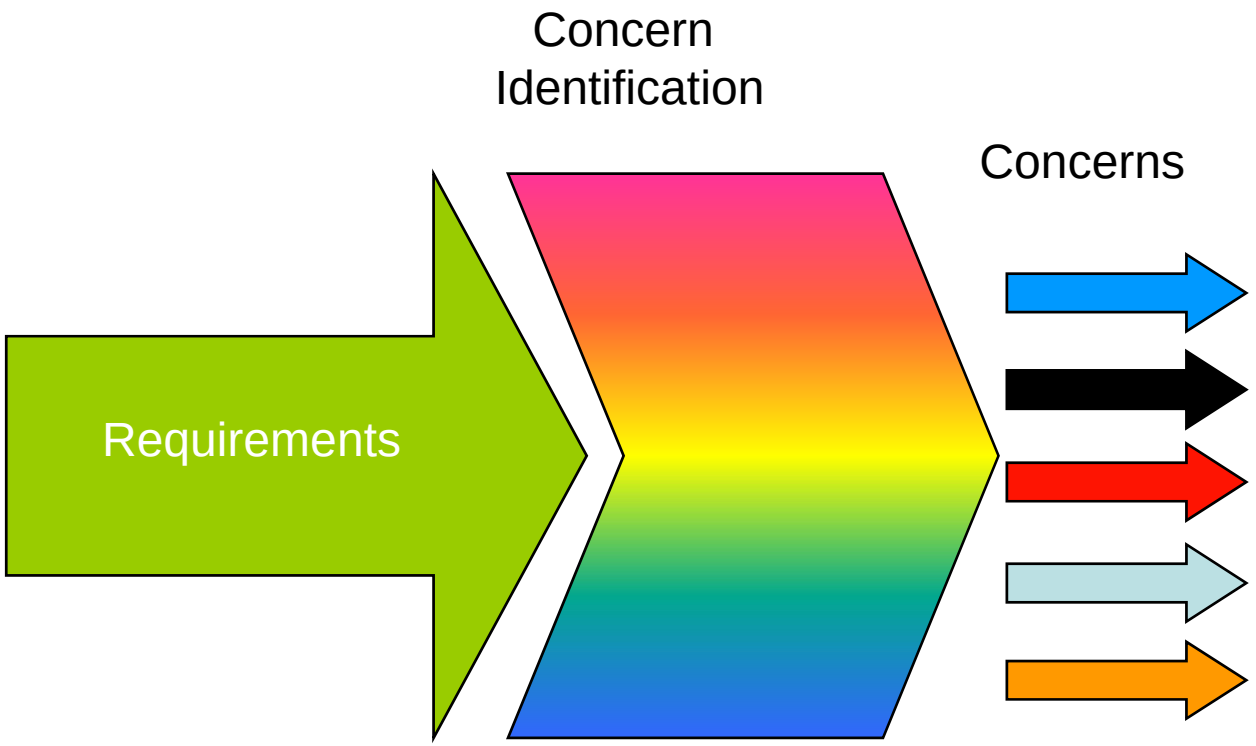
- Persistence code is **localized**, can be understood in isolation, part of it can be reused
- Business code **can** be reused to work with other persistence APIs
- Business code is **not** invaded by changes to persistence APIs
- Less code, more code units

Towards to

Aspect Oriented Software Development

(AOSD)

Step 1: Concern identification (decomposition)



Requirements engineering

Examples of Identified Concerns

Graphical user interface

Data management

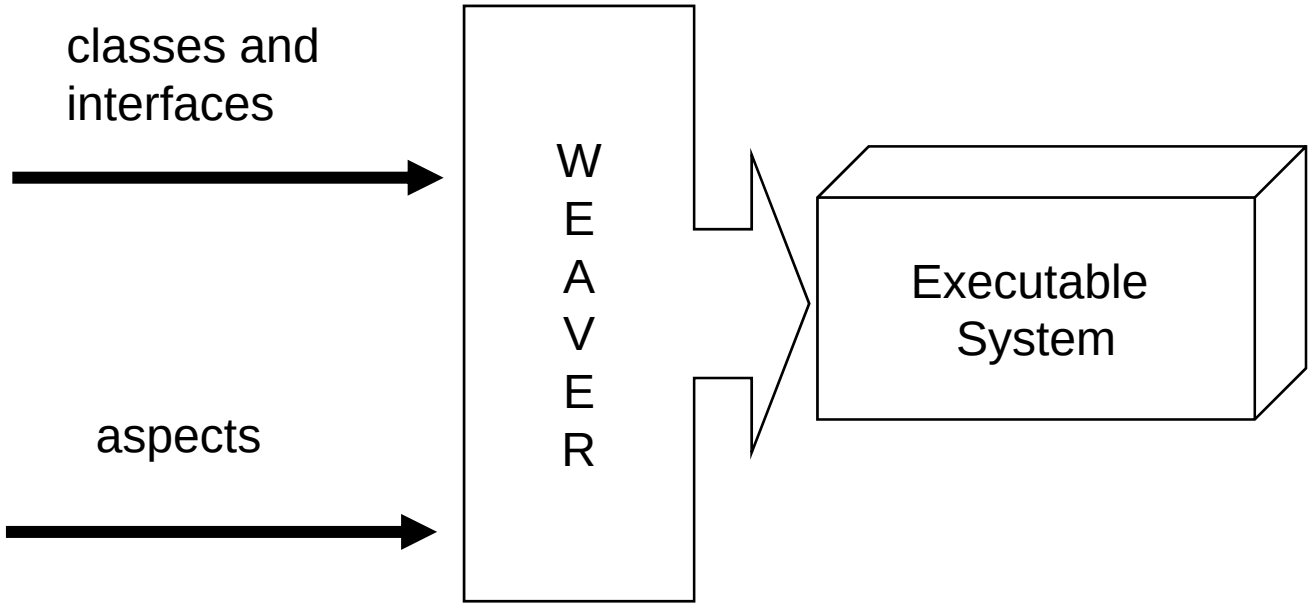
Distribution

Business rules

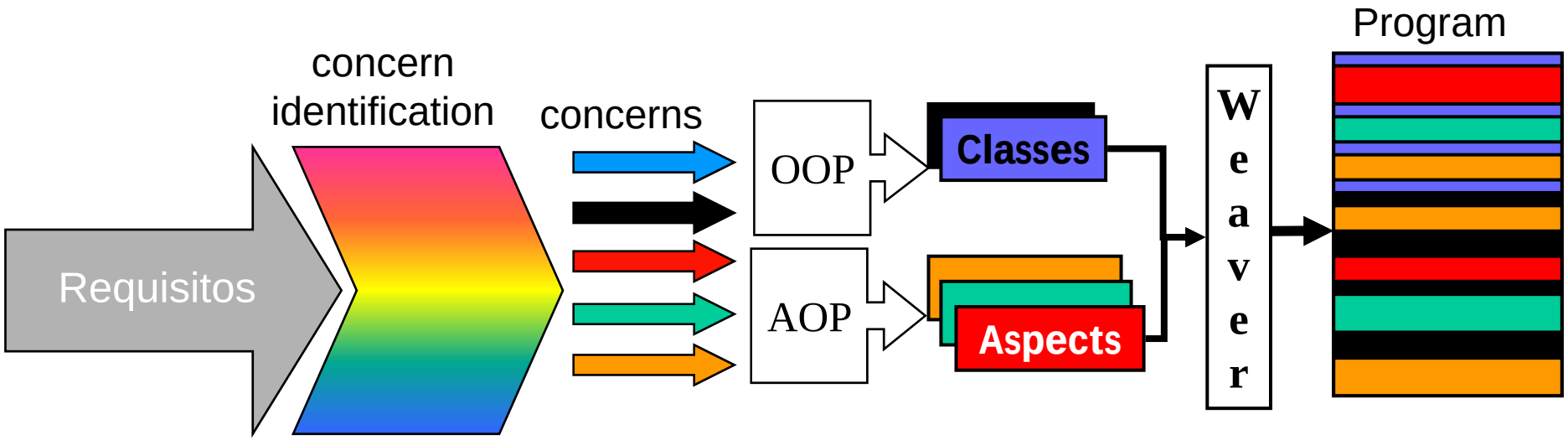
Concurrency control

Step 2: Concern Implementation

Step 3: System Recomposition

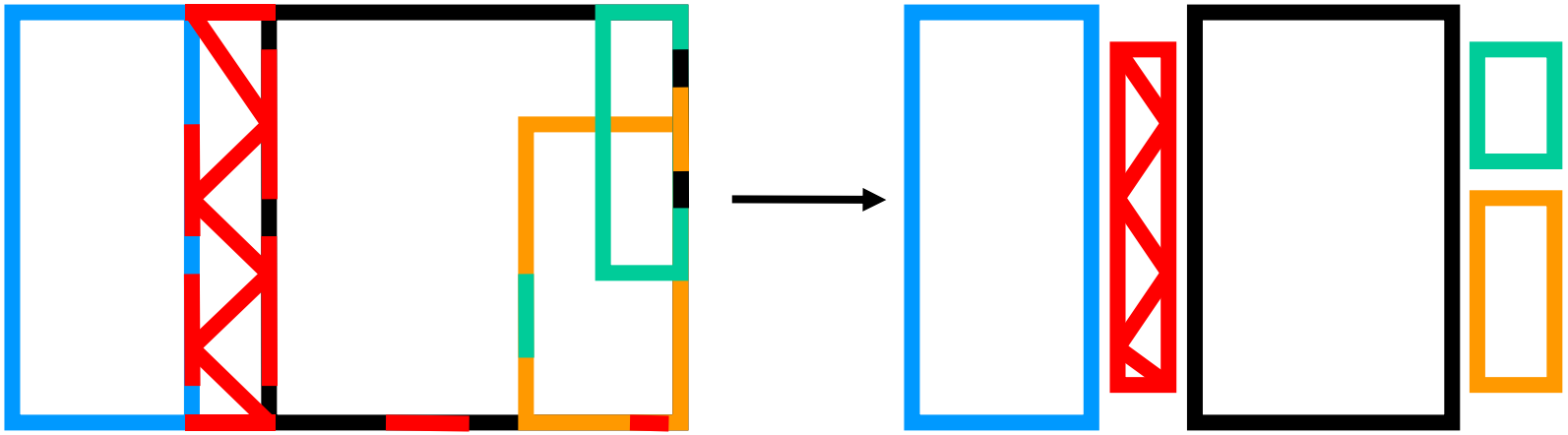


Aspect Oriented Software Development



- Business rules
- Concurrency control
- Data management
- Graphical user interface
- Distribution

OOP vs. AOP



Business rules

Data management

Distribution

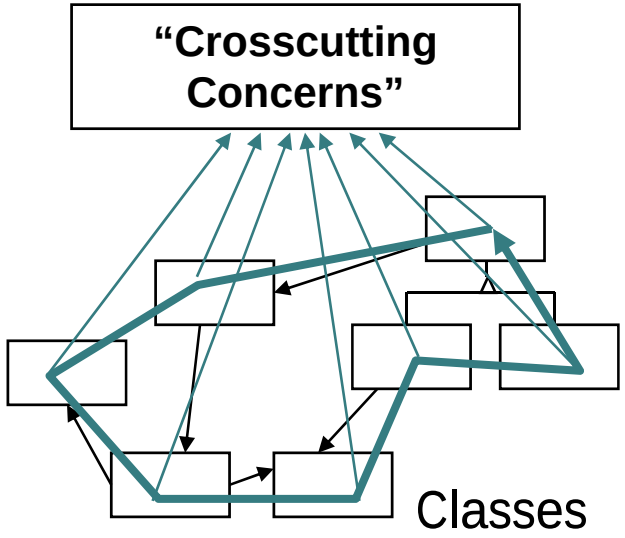
Concurrency control

Graphical user interface

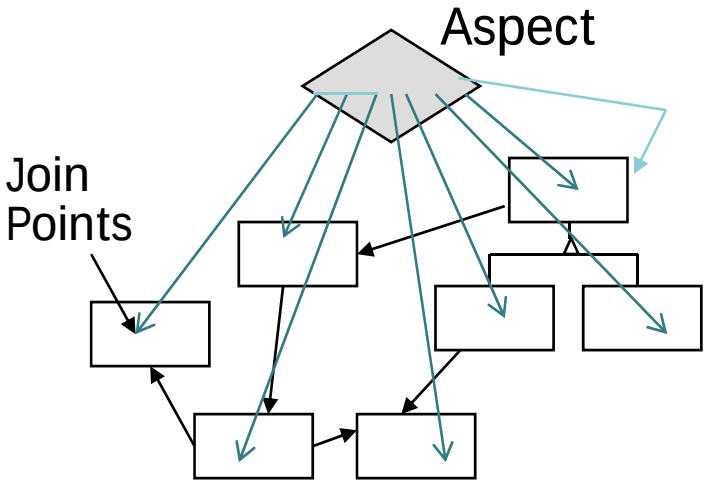
Context: Objects vs. Aspects

- Aspect-oriented programming (AOP)
  - modularizing crosscutting concerns (CCCs)

OO Decomposition

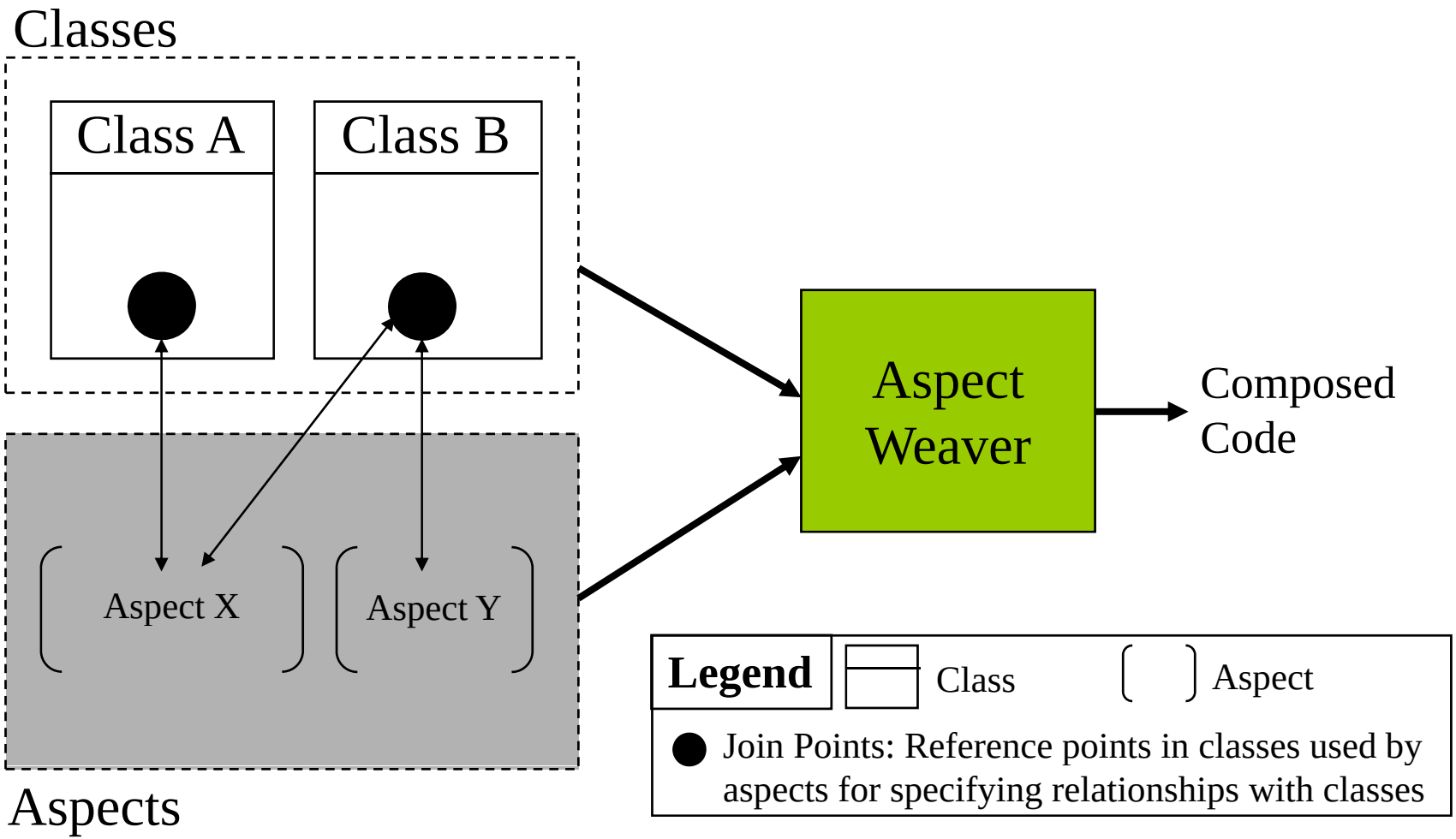


Aspect-Oriented Decomposition



- enhancing reusability, maintainability, and the like
- AspectJ: most popular aspect-oriented language

Aspect-Oriented Programming (AOP): The AspectJ Approach



Pointcuts define a set of **join points**

- join points can be:
  - calls and executions of methods (and also constructors)
  - field access
  - exception handling
  - static initialization
- joint point composition
  - **&&, | |** e **!**

Advice in AspectJ provides extra behavior at join points

- Define additional code that should be executed...
  - **`before`**
  - **`after`**
    - **`after returning`**
    - **`after throwing`**
  - **`or around`**

  join points

Persistence at facade

```
public class Banco {
  private CadastroContas contas;...
  public void cadastrar(Conta conta) {
    try {
      getPM().startTransaction();
      contas.cadastrar(conta); ...
      getPM().commitTransaction();
    } catch (Exception ex){
      getPM().rollBackTransaction();...
    }
  }...
```

Pointcut (with a simple alternative)

```
pointcut transMethods():
  execution (public * Banco.cadastrar(..)) ||
  execution (public * Banco.creditar(..)) ||
  execution (public * Banco.deditar(..)) ||
  ...
```

```
pointcut transMethods():
  execution (public * Banco.*(..));
```

Advice for normal flow

```
public aspect PersistenceAspect {
  before(): transMethods() {
    getPM().StartTransaction();
}

  after() returning(): transMethods() {
    getPM().commitTransaction();
}
```

Advice for exceptional flow

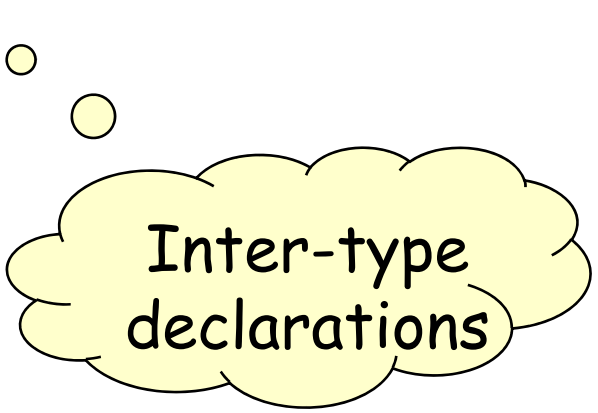
Exception is not captured

```
after() throwing(): transMethods() {
    getPM().rollBackTransaction();
}

...
```

Besides dynamic crosscutting with advice...

- AspectJ also supports static crosscutting
  - change the relation of subtypes
  - add members into classes



Static crosscutting

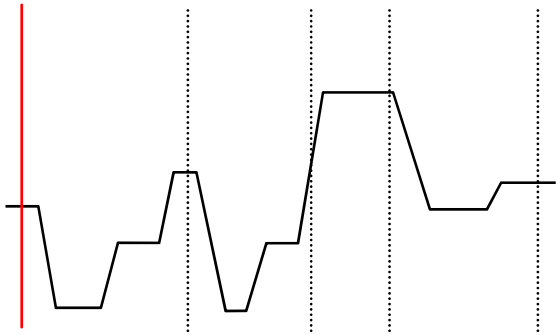
interface local to the aspect

```
pointcut TransMethods():
  execution (public * Trans.*(..));
private interface Trans {
  public void cadastrar(Conta conta);...
}
declare parents: Banco implements Trans;
```

changing the hierarchy

Research with AOP

discussing
about some
researches that
use
AOP



Research with AOP

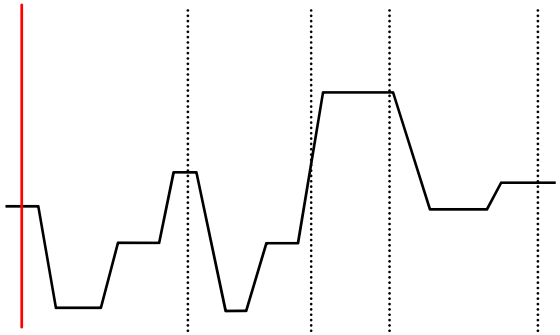
TAO

ample
PROJECT

- There are several other
  - Exception handling improvement
  - Exception handling refactoring
  - AO refactoring
  - ...

AOP in industry

discussing about the adoption of AOP in idustry



AOP in Industry

- Used tools
  - Spring AOP
  - JBoss AOP
  - AspectJ
  - PostSharp (.NET)
  - Compose*
  - EOS (.NET)
  - Glassbox

Aspects in some Brazilian companies

- In most cases this means Spring AOP, but AspectJ is used too
- Popular concerns
  - Transactional control
  - Access control
  - Logging
  - Security
- Business code with reduced maintenance and more reuse

Aspects in some known companies

- IBM
  - supports AspectJ
- Motorola
  - WEAVR
- Microsoft
  - Policy Injection Application Block

Conclusions

- OO appeared with Simula 67
  - became popular in the industry after ...

    25 years?
    - design patterns
- AOP has 15 years
  - some use in big companies

AOP Framed!

Henrique Rebêlo

Informatics Center

Federal University of Pernambuco