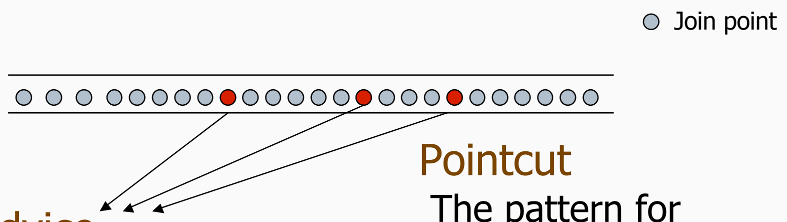
Remote Pointcut
- A Language Construct for Distributed AOP

Muga Nishizawa (Tokyo Tech)
Shigeru Chiba (Tokyo Tech)
Michiaki Tatsubori (IBM)

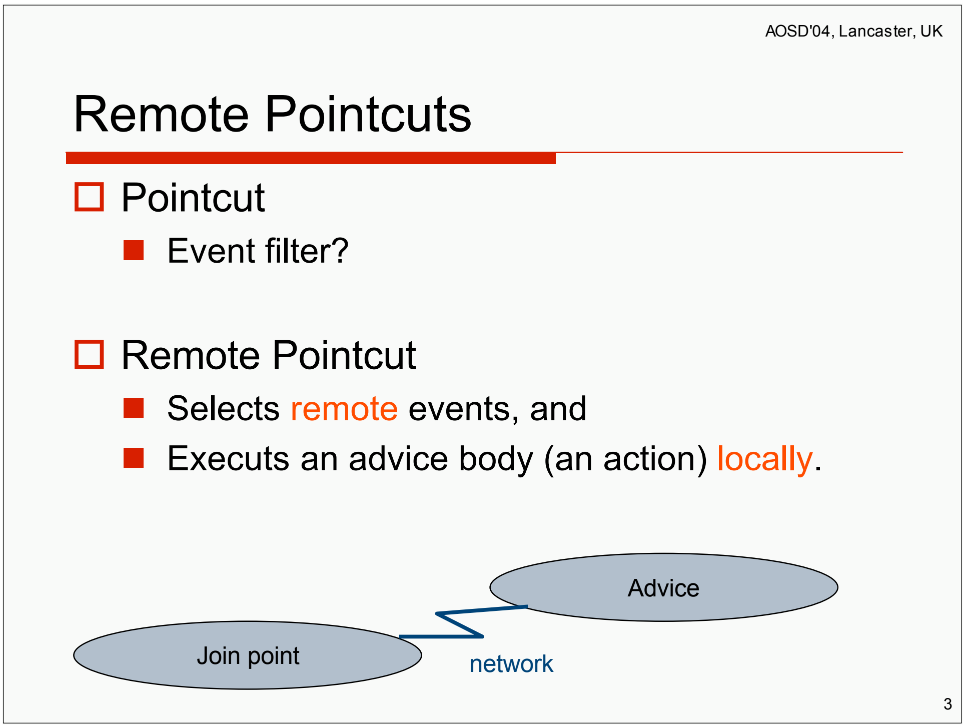
Pointcut-advice model

- Joinpoints
  - Program execution is modeled as a sequence of execution points.



Pointcut
The pattern for selecting joinpoints

Advice
The code invoked at the selected join points



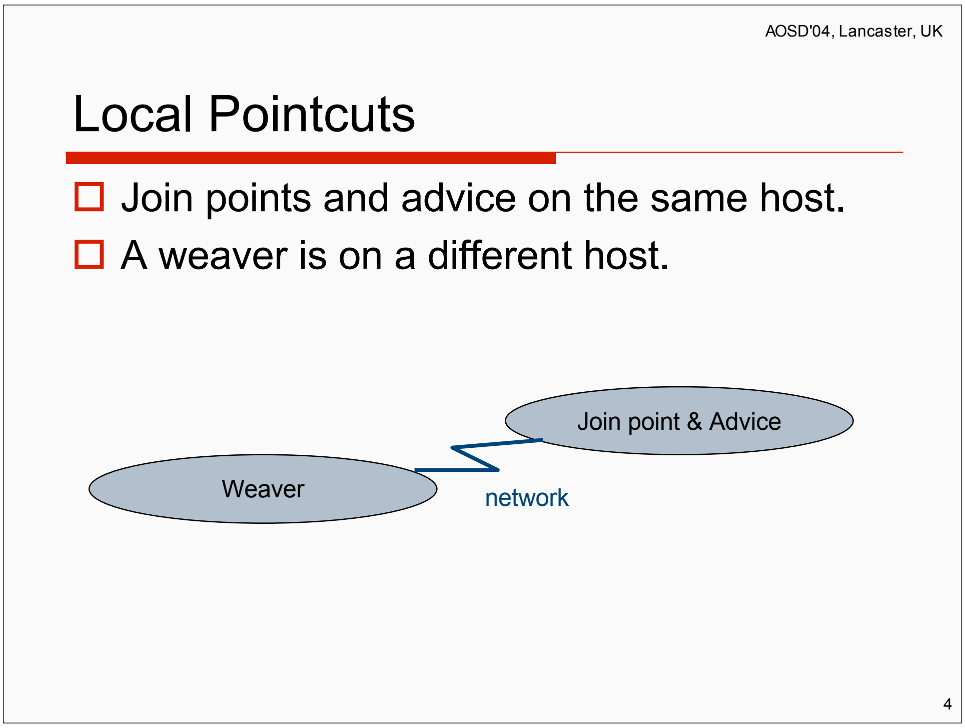
Remote Pointcuts

- Pointcut
  - Event filter?
- Remote Pointcut
  - Selects remote events, and
  - Executs an advice body (an action) locally.

Advice

Join point

network



Local Pointcuts

- Join points and advice on the same host.
- A weaver is on a different host.

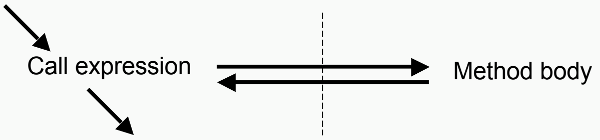
Join point & Advice

Weaver

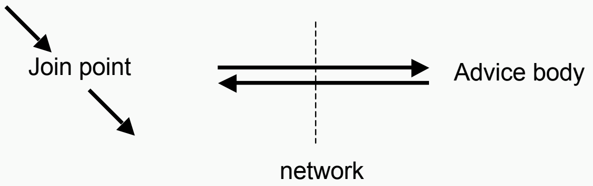
network

RMI and Remote Pointcut

- Remote Method Invocation



- Remote Pointcut



So, what is the research topic?

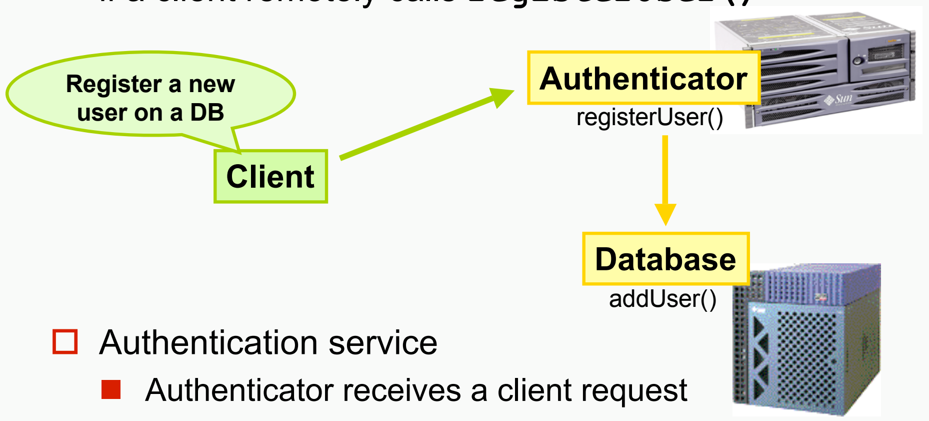
- Examples!
  - Practical languages should provide only useful mechanisms.
  - Not a play ground for academic researchers!

This Talk

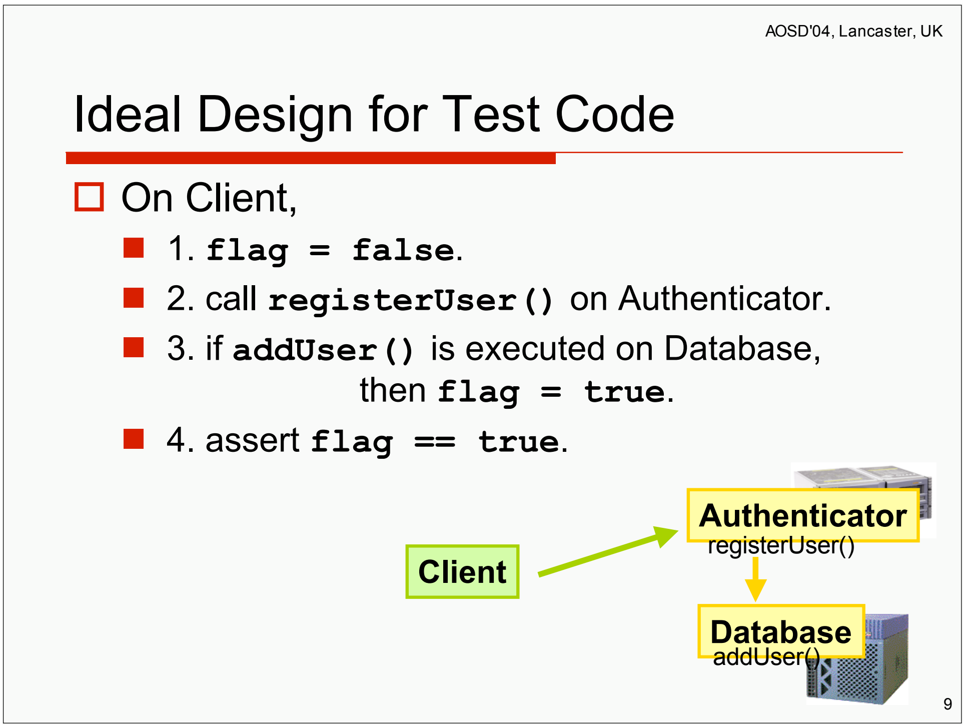
- □ Our goal
  - To modularize crosscutting concerns in distributed software
- □ Motivating problem
  - AspectJ can separate them
  - But, the implementation is NOT simple
  - e.g. a test code for distributed software
- □ Our solution
  - Remote pointcut

Test Code for Distributed Authentication Service

- □ Confirm **`addUser()`** is executed on Database if a client remotely calls **`registerUser()`**



- □ Authentication service
  - Authenticator receives a client request
  - Database adds the new user's information

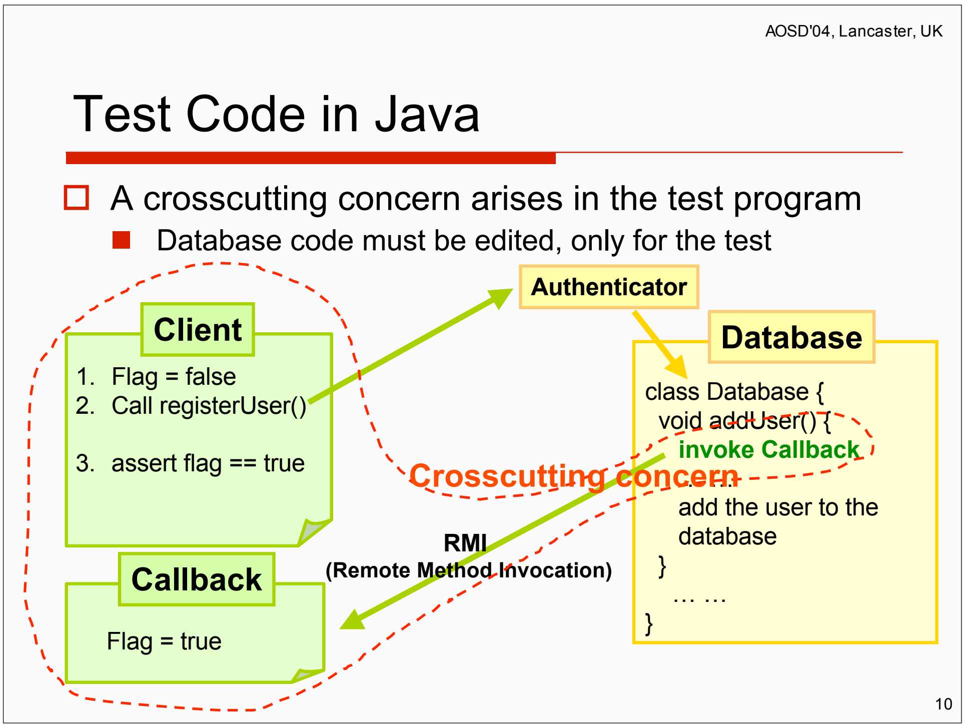


AOSD'04, Lancaster, UK

Ideal Design for Test Code

- On Client,
  - 1. **`flag = false`**.
  - 2. call **`registerUser()`** on Authenticator.
  - 3. if **`addUser()`** is executed on Database, then **`flag = true`**.
  - 4. assert **`flag == true`**.

9



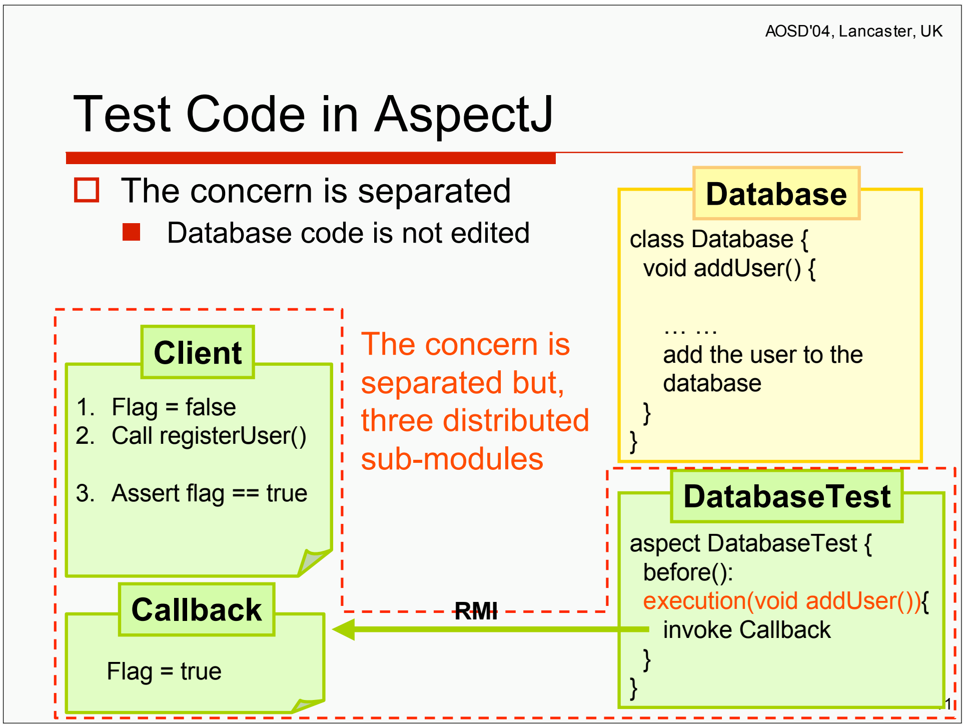
AOSD'04, Lancaster, UK

Test Code in Java

- A crosscutting concern arises in the test program
  - Database code must be edited, only for the test

```
class Database {
  void addUser() {
    invoke Callback
    add the user to the
    database
  }
  … …
}
```

10



Test Code in AspectJ

- □ The concern is separated
  - ■ Database code is not edited

Client

1. Flag = false
2. Call registerUser()
3. Assert flag == true

Callback

Flag = true

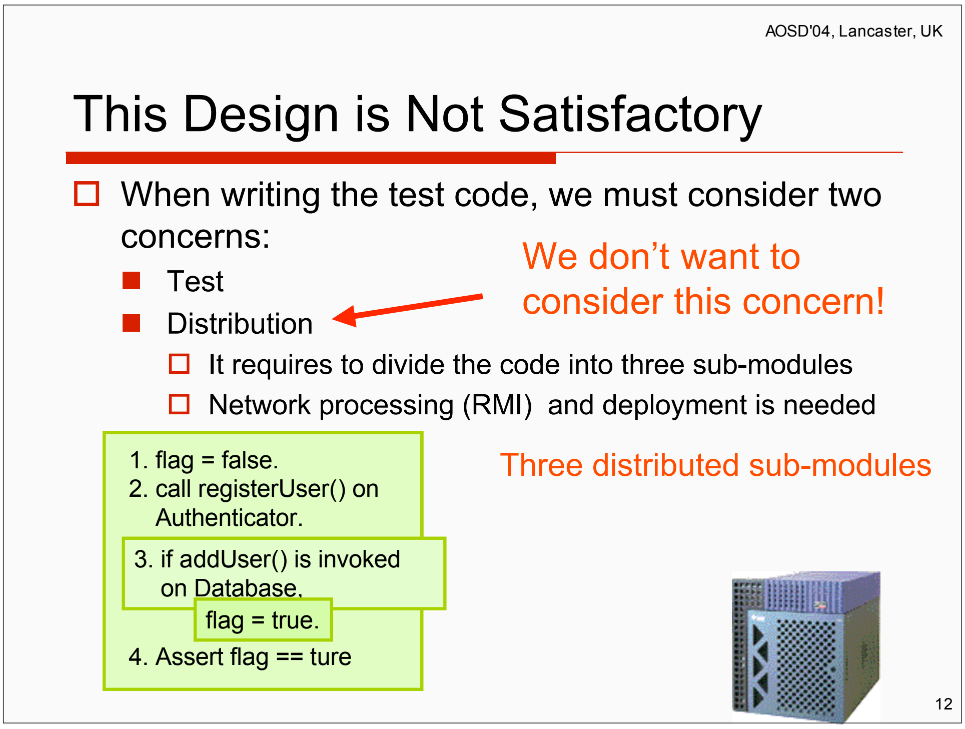
The concern is separated but, three distributed sub-modules

Database

```
class Database {
  void addUser() {

    … …
    add the user to the
    database
  }
}
```

DatabaseTest

```
aspect DatabaseTest {
  before():
  execution(void addUser()){
    invoke Callback
  }
}
```



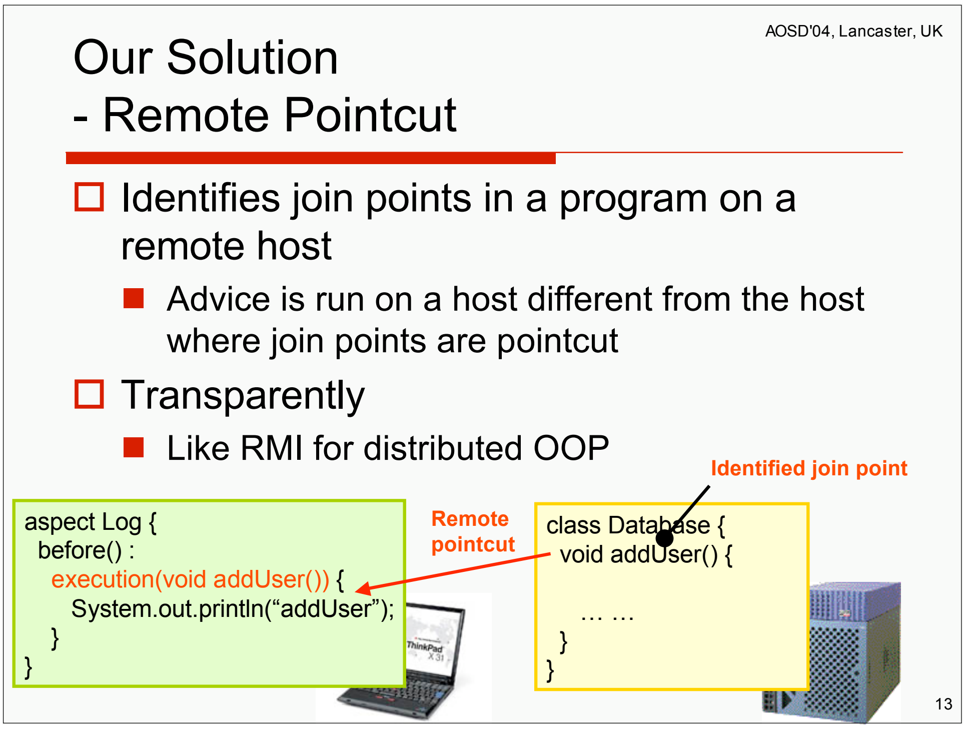
This Design is Not Satisfactory

- □ When writing the test code, we must consider two concerns:
  - ■ Test
  - ■ Distribution
    - □ It requires to divide the code into three sub-modules
    - □ Network processing (RMI) and deployment is needed

We don't want to consider this concern!

1. flag = false.
2. call registerUser() on Authenticator.
3. if addUser() is invoked on Database,
   flag = true.
4. Assert flag == ture

Three distributed sub-modules



AOSD'04, Lancaster, UK

Our Solution - Remote Pointcut

- □ Identifies join points in a program on a remote host
  - Advice is run on a host different from the host where join points are pointcut
- □ Transparently
  - Like RMI for distributed OOP

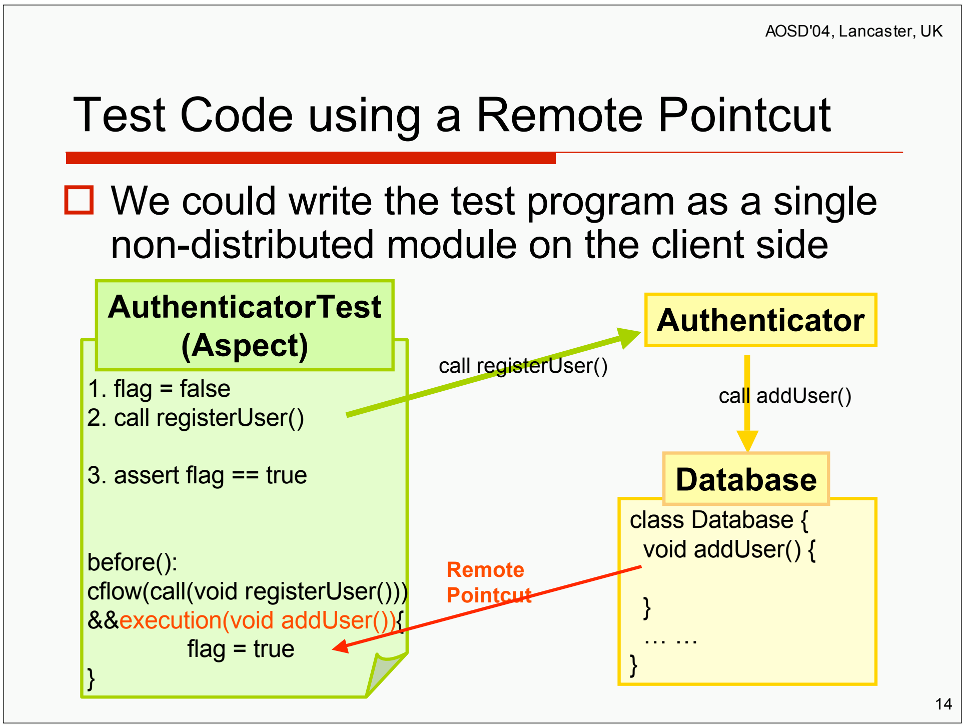
```
aspect Log {
  before() :
    execution(void addUser()) {
      System.out.println("addUser");
    }
}
```

Remote pointcut

Identified join point

```
class Database {
  void addUser() {

     … …
  }
}
```

13



AOSD'04, Lancaster, UK

Test Code using a Remote Pointcut

- □ We could write the test program as a single non-distributed module on the client side

AuthenticatorTest (Aspect)

```
1. flag = false
2. call registerUser()

3. assert flag == true


before():
cflow(call(void registerUser()))
&&execution(void addUser()){
              flag = true
}
```

call registerUser()

Authenticator

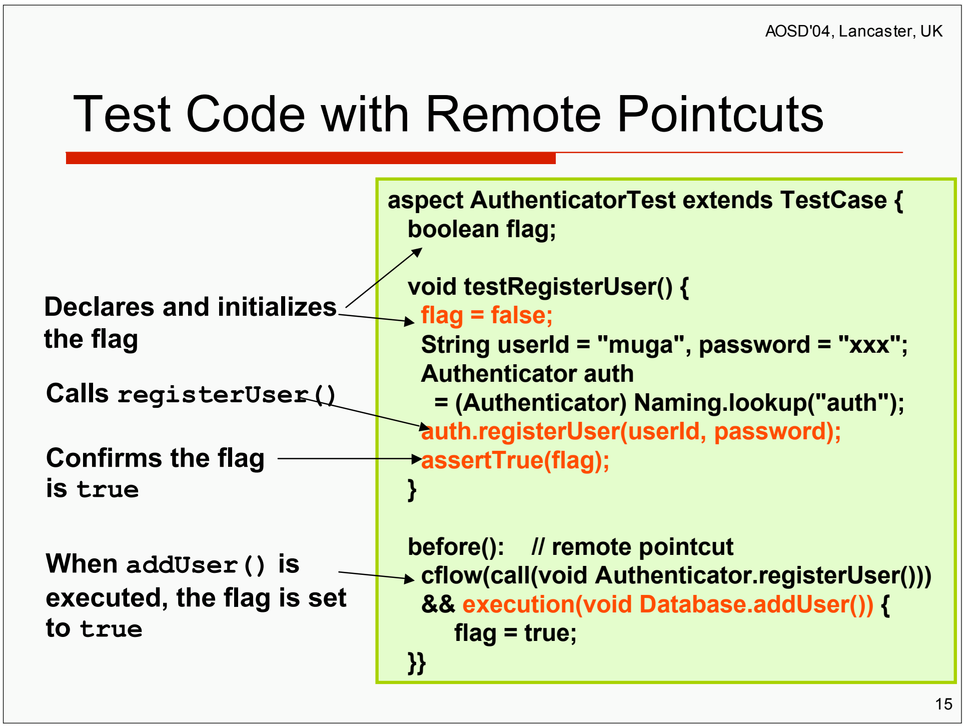
call addUser()

Database

```
class Database {
  void addUser() {

  }
  … …
}
```

Remote Pointcut

14



Test Code with Remote Pointcuts

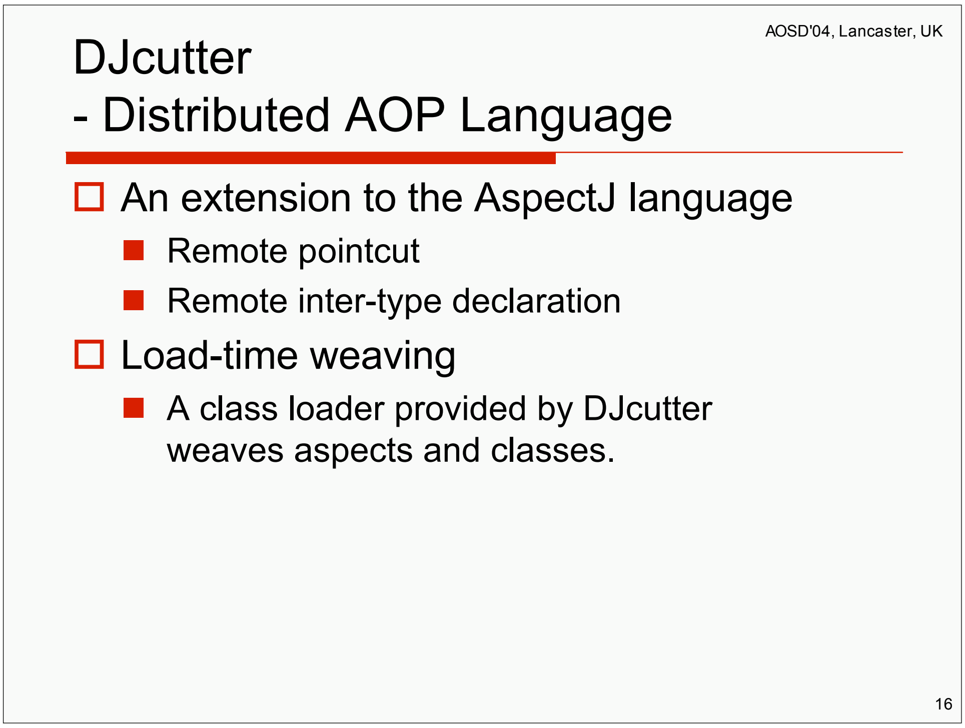
Declares and initializes the flag

Calls `registerUser()`

Confirms the flag is `true`

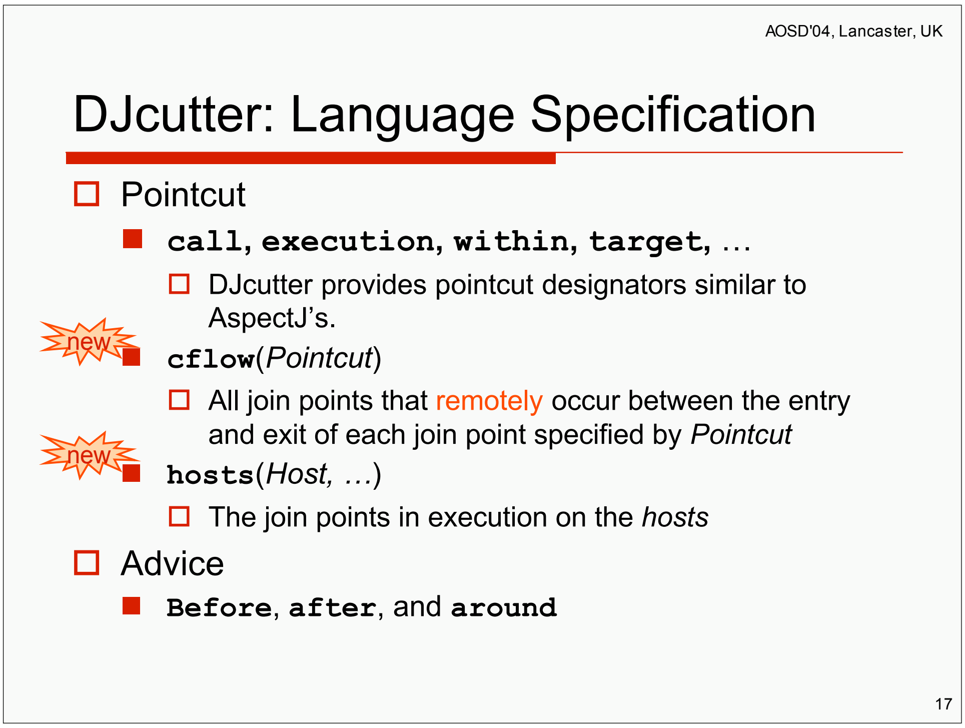
When `addUser()` is executed, the flag is set to `true`

```
aspect AuthenticatorTest extends TestCase {
  boolean flag;

  void testRegisterUser() {
    flag = false;
    String userId = "muga", password = "xxx";
    Authenticator auth
      = (Authenticator) Naming.lookup("auth");
    auth.registerUser(userId, password);
    assertTrue(flag);
  }

  before():    // remote pointcut
    cflow(call(void Authenticator.registerUser()))
    && execution(void Database.addUser()) {
       flag = true;
  }}
```



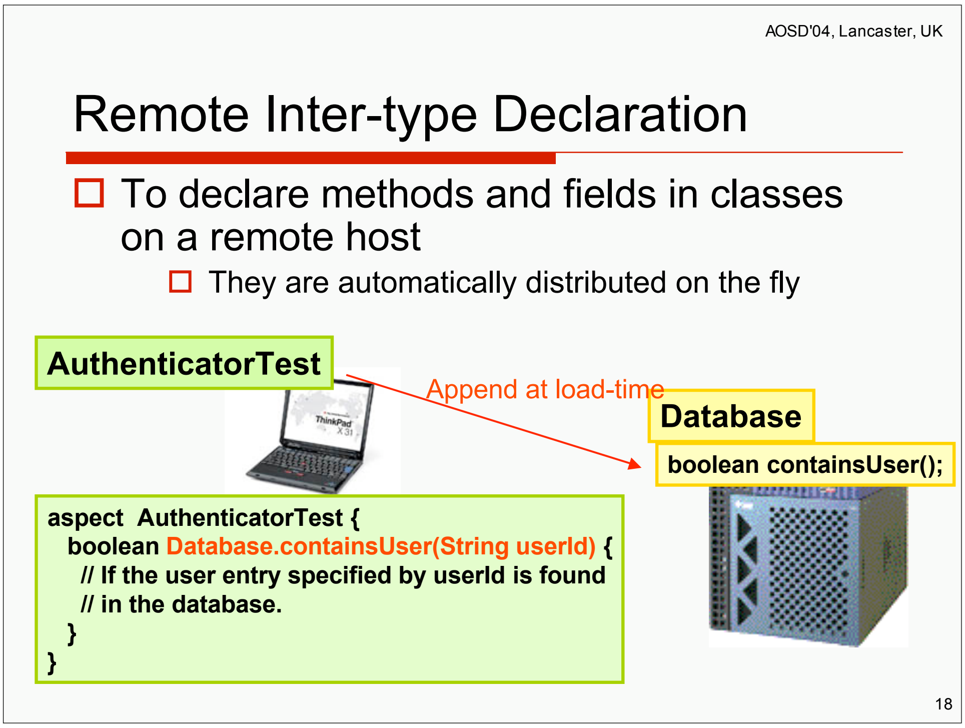
DJcutter
- Distributed AOP Language

- An extension to the AspectJ language
  - Remote pointcut
  - Remote inter-type declaration
- Load-time weaving
  - A class loader provided by DJcutter weaves aspects and classes.



DJcutter: Language Specification

- Pointcut
  - **`call`, `execution`, `within`, `target`**, …
    - DJcutter provides pointcut designators similar to AspectJ's.
  - new **`cflow`**(*Pointcut*)
    - All join points that remotely occur between the entry and exit of each join point specified by *Pointcut*
  - new **`hosts`**(*Host, …*)
    - The join points in execution on the *hosts*
- Advice
  - **`Before`**, **`after`**, and **`around`**



Remote Inter-type Declaration

- To declare methods and fields in classes on a remote host
  - They are automatically distributed on the fly

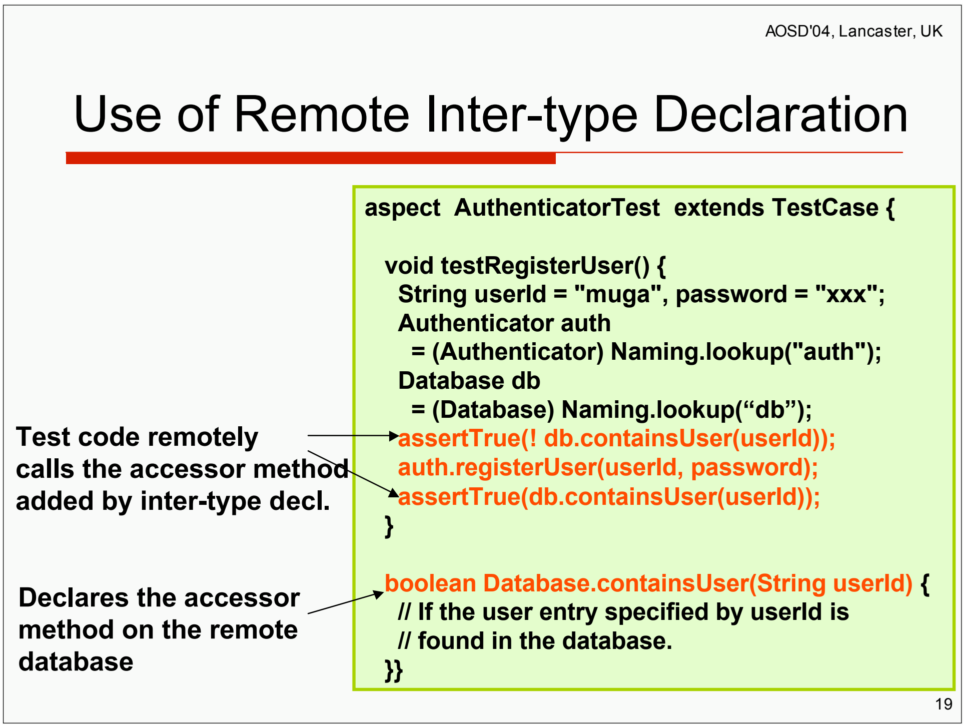
AuthenticatorTest

Append at load-time

Database

boolean containsUser();

```
aspect  AuthenticatorTest {
  boolean Database.containsUser(String userId) {
    // If the user entry specified by userId is found
    // in the database.
  }
}
```

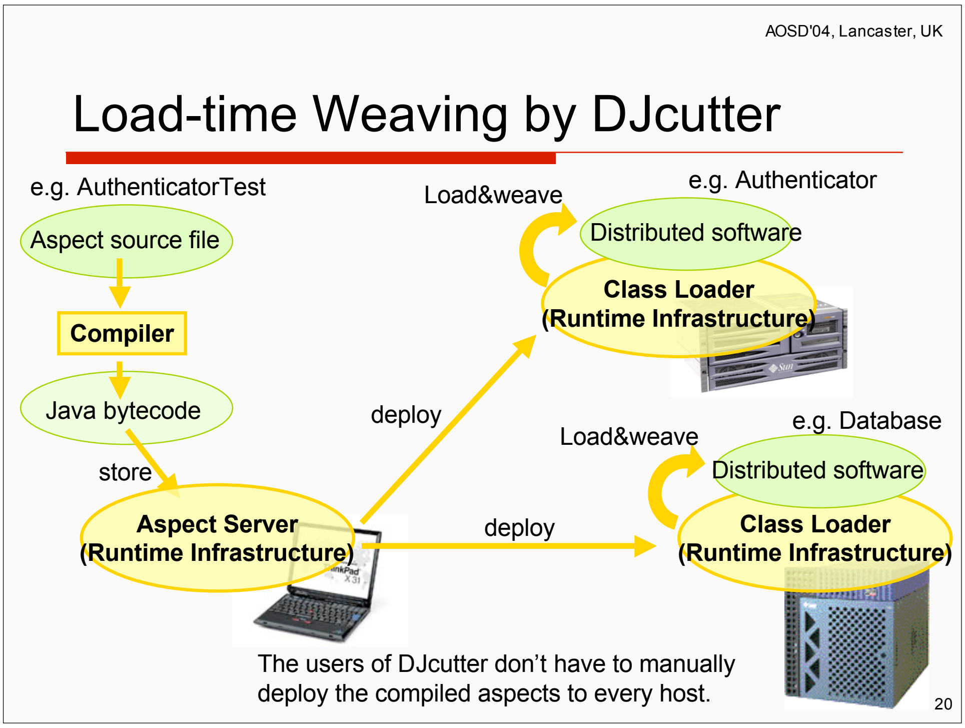


Use of Remote Inter-type Declaration

```
aspect  AuthenticatorTest  extends TestCase {

  void testRegisterUser() {
    String userId = "muga", password = "xxx";
    Authenticator auth
      = (Authenticator) Naming.lookup("auth");
    Database db
      = (Database) Naming.lookup("db");
    assertTrue(! db.containsUser(userId));
    auth.registerUser(userId, password);
    assertTrue(db.containsUser(userId));
  }

  boolean Database.containsUser(String userId) {
    // If the user entry specified by userId is
    // found in the database.
  }}
```

Test code remotely calls the accessor method added by inter-type decl.

Declares the accessor method on the remote database



Load-time Weaving by DJcutter

e.g. AuthenticatorTest

Aspect source file

Compiler

Java bytecode

store

Aspect Server (Runtime Infrastructure)

deploy

deploy

Load&weave

e.g. Authenticator

Distributed software

Class Loader (Runtime Infrastructure)

Load&weave

e.g. Database

Distributed software

Class Loader (Runtime Infrastructure)

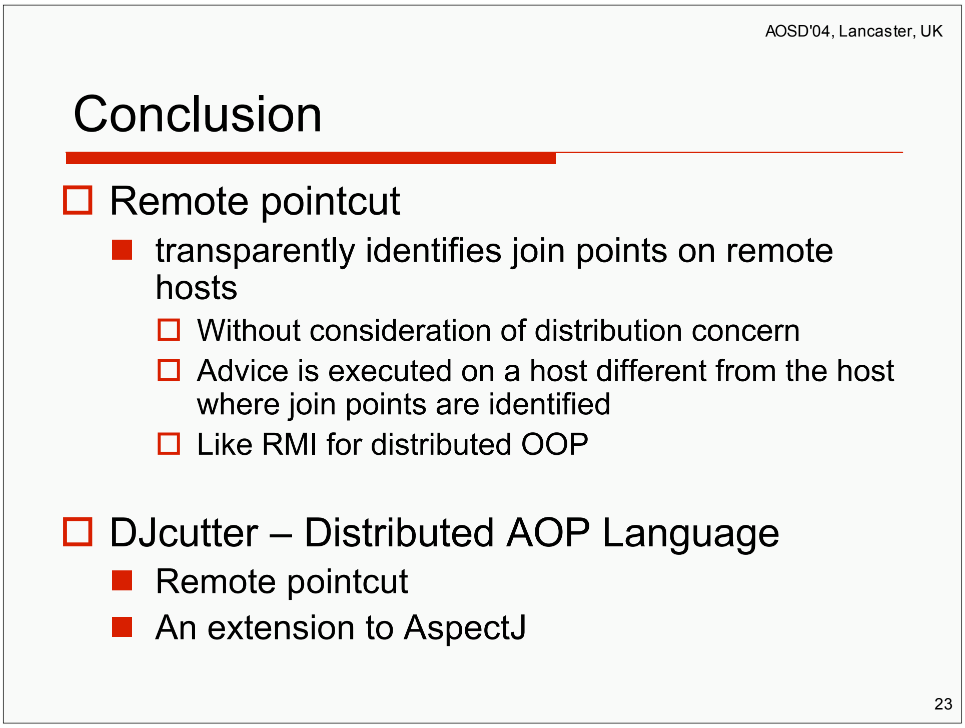
The users of DJcutter don’t have to manually deploy the compiled aspects to every host.

Related Work 1

- Middleware for automatic distribution
  - e.g. Addistant [Ecoop01],
    J-Orchestra [Ecoop02]
  - The distribution concern is completely hidden.
  - DJcutter ≠AspectJ + Addistant
    - DJcutter selectively hides the distribution concern,
      when users don't want to see it.
    - DJcutter works with existing infrastructure such as Tomcat, JBoss, Oracle, …

Related Work 2

- Distributed AOP languages
  - D language framework
    JAC (Java Aspect Componenets)
  - for modularizing non-functional crosscutting concerns
- DJcutter
  - Remote pointcut
  - for modularizing functional crosscutting concerns
  - without consideration of distribution



Conclusion

- Remote pointcut
  - transparently identifies join points on remote hosts
    - Without consideration of distribution concern
    - Advice is executed on a host different from the host where join points are identified
    - Like RMI for distributed OOP

- DJcutter – Distributed AOP Language
  - Remote pointcut
  - An extension to AspectJ