Software Architecture
软件体系结构

Lecture 6. Event-based Software Architecture
基于事件的软件体系结构

Professor:
Yushan (Michael) Sun
Fall 2020

Lecture 4. Event-based Software Architecture

Content of the lecture

1. 基于事件系统的概念
2. 事件处理策略
3. 使用观察者模式设计的事件系统的例子

Concept of Event-based System

基于事件系统的概念

1. Concept of Event-based System

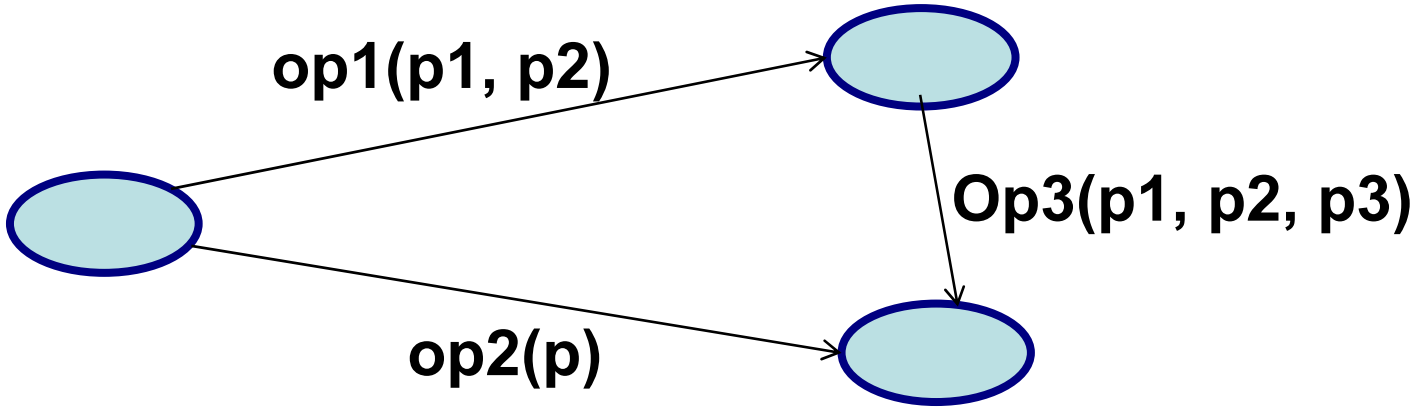
- 显式调用. Explicit Invocation:
- Traditionally, in a system in which the component provides a collection of
  - routines and functions, such as an object-oriented system,

  components typically interact with each other by explicitly invoking those routines.

1. Concept of Event-based System

显式调用的特点：调用者必须知道被调用者的类名、构造方法（包括参数）与要调用的方法（包括参数）

Explicit invocation: the invoking object must know the name of the class being invoked and the method (including parameters) in the class



1. Concept of Event-based System

事件系统使用隐式调用

Event system uses implicit invocation:

- **In an event based system, an event will invoke some procedures and the procedures will run automatically.**
- **Event system uses implicit invocation.**

1. Concept of Event-based System

基于事件系统的定义：
Definition of an event-based system:

- **An event based system is such a system in which procedures are not invoked directly (that means, indirectly, or implicitly).**

What is a procedure （什么是过程）？

- **A procedure is a module in program design language, with or without parameters.**
- **It is executed by procedural calls. The result will be**
  - **assigned to the calling parameter, or to**
  - **modify a gloabal variable within this subroutine.**

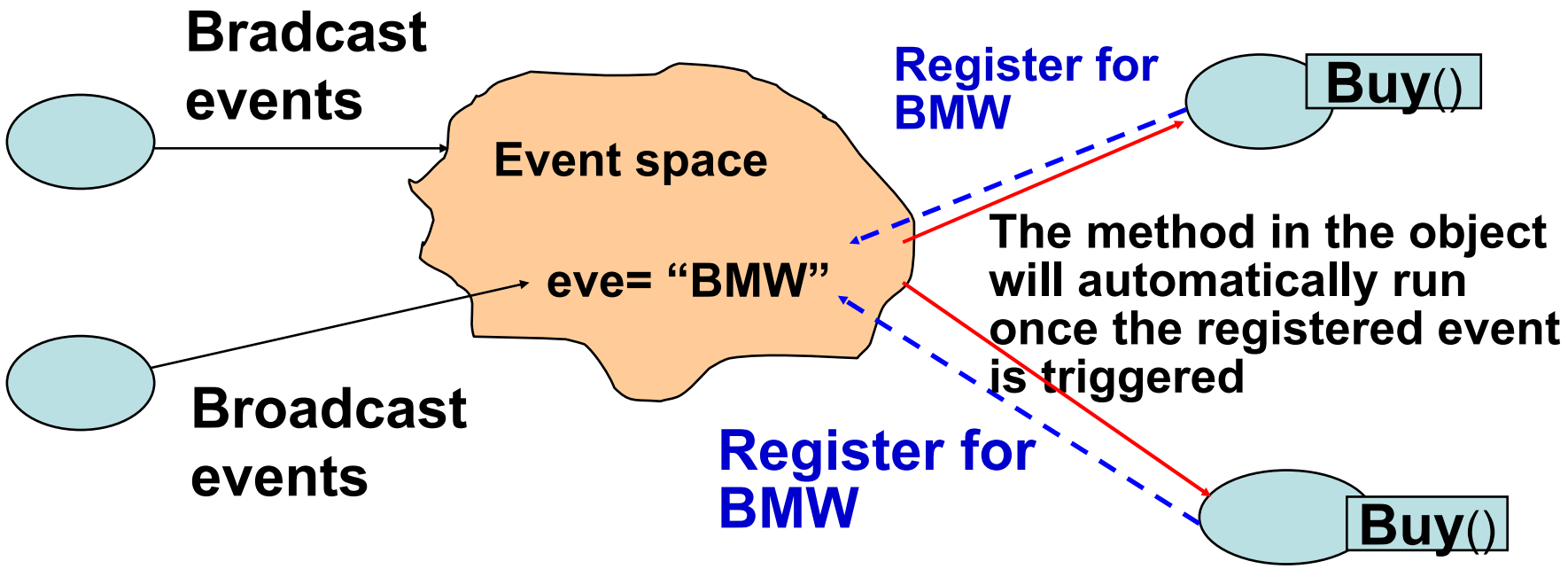
1. Concept of Event-based System

事件广播与处理机制

Event broadcast and processing mechanism:

- **广播机制. A component can *broadcast* one or more events.**
- **注册机制. Other components in the system can *register* an interest in an event**
- **系统调用. Once an event is broadcasted, the system will automaticaly invoke all of the procedures that have registered for the event. ("implicitly “**
- **事件发布者可能不负责调用**

1. Concept of Event-based System



Event-based system: event broadcast, event registration, method being called automatically

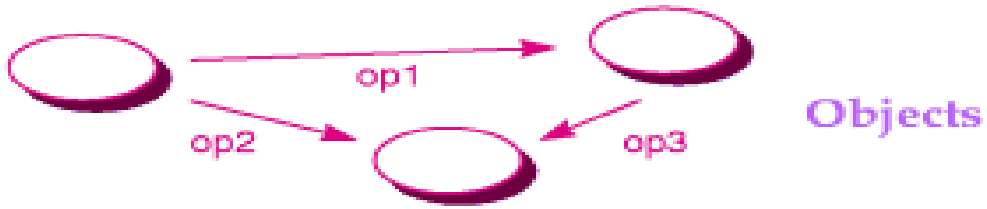
1. Concept of Event-based System

显式调用与隐式调用的区别

Event Systems: Implicit versus Explicit Invocation

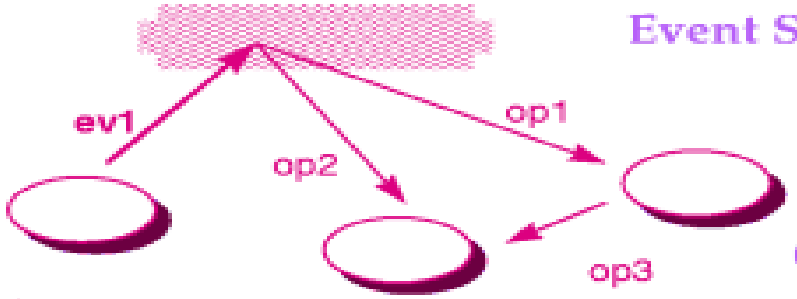
想调用谁，就直接调用

Explicit Invocation



Implicit Invocation

一个对象广播一个事件到事件空间



系统自动调用所有已经注册了该事件的过程

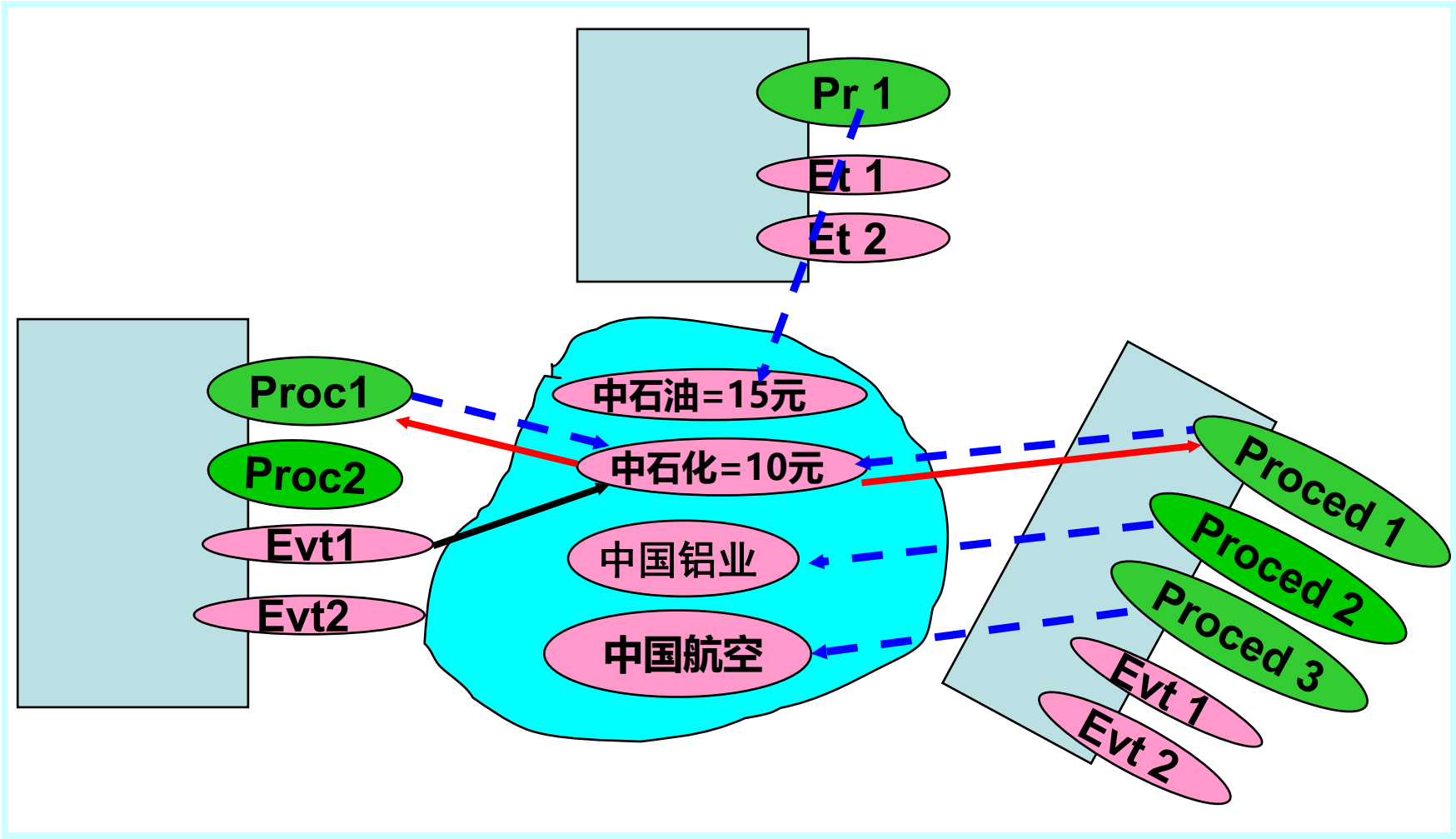
1. Concept of Event-based System

Structure of a component (部件结构)

- **Components:** the components in an implicit invocation style are modules whose interfaces provide both
  - a collection of **procedures** (functions in C++, methods in Java) and
  - a set of **events**

1. Concept of Event-based System

例：使用事件系统架构设计的股票交易系统部分组件



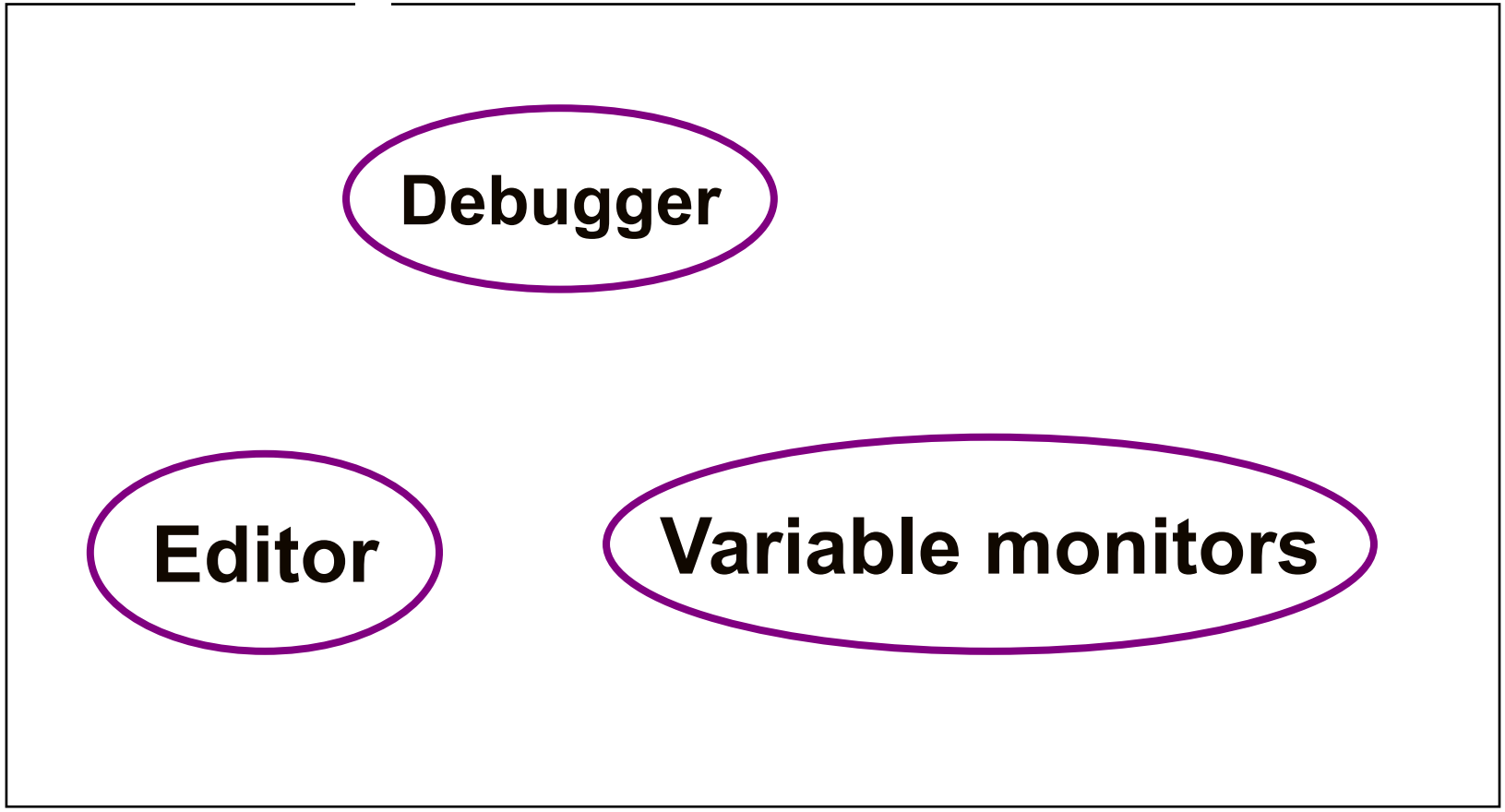
1. Concept of Event-based System

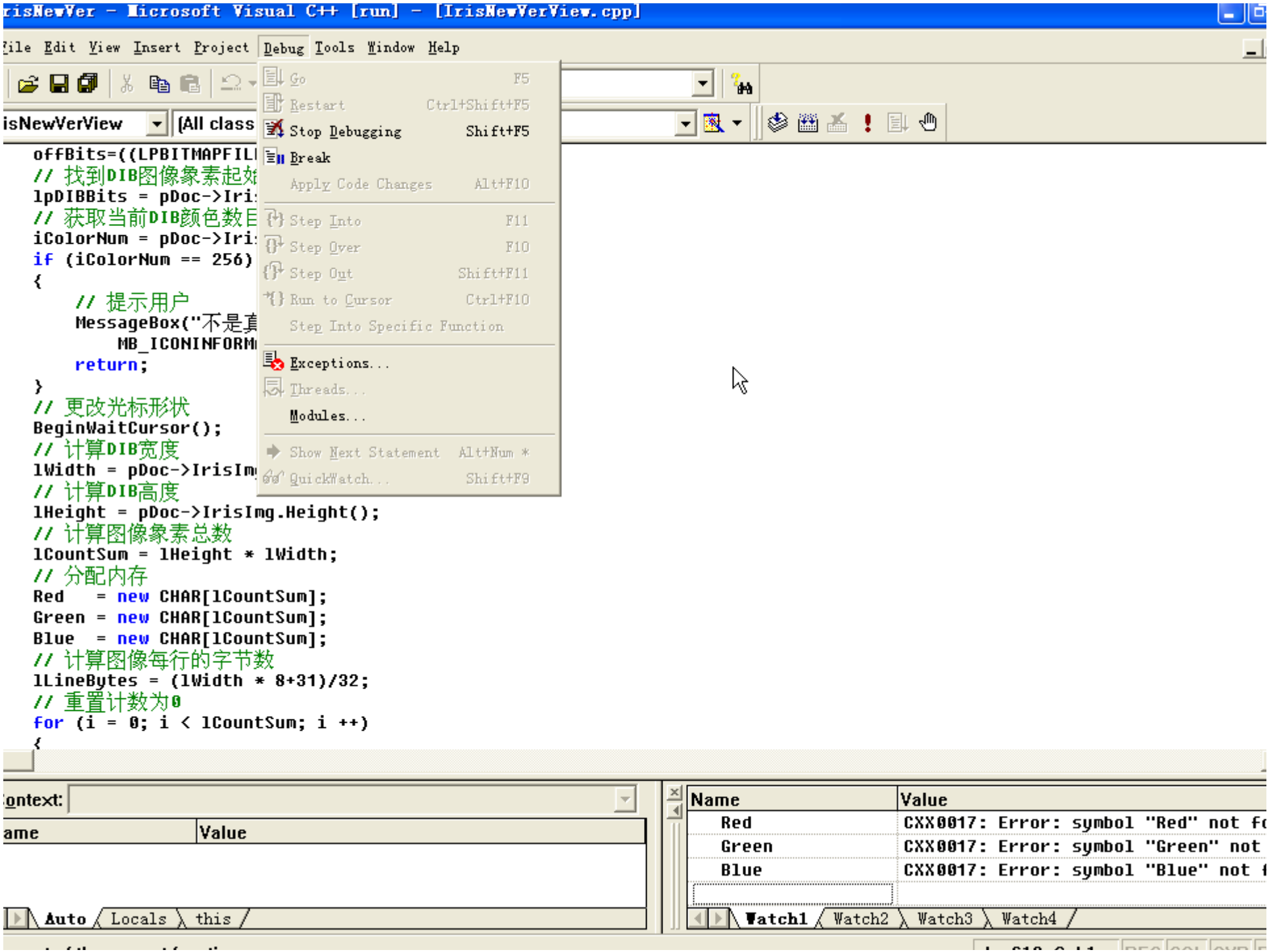
一个事件系统的例子

集成开发环境(IDE)的例子

- Consider an integrated Development Environment for C, C++ or Java.
- Such IDE consists of tools such as
  - 编辑器. Editors for source code,
  - 变量监控器. Variable monitors,
  - 调试器. A debugger, etc.
- Such systems usually utilize an event based Architecture.

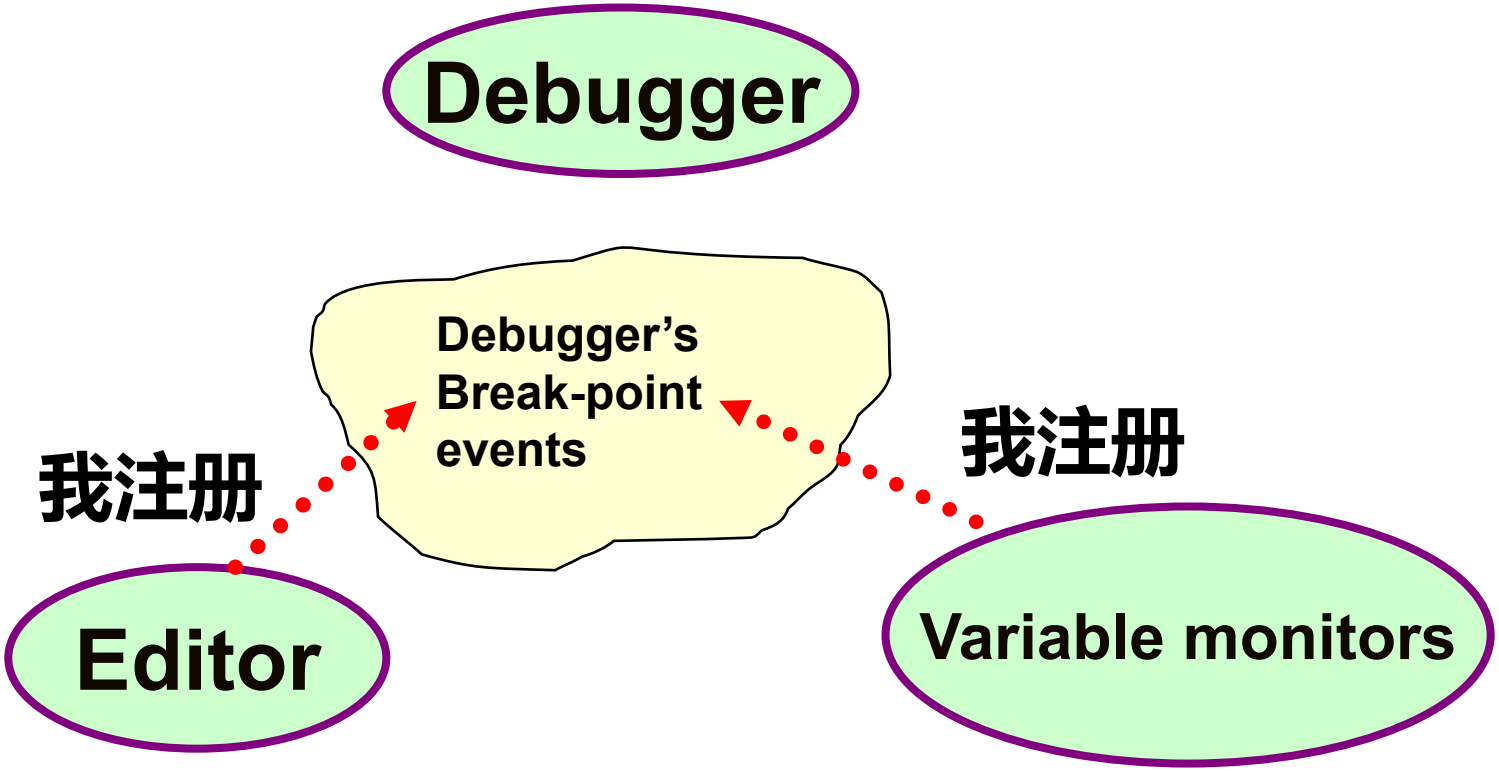
1. Concept of Event-based System




risNewVer - Microsoft Visual C++ [run] - [IrisNewVerView.cpp]
File Edit View Insert Project Debug Tools Window Help
Go F5
Restart Ctrl+Shift+F5
Stop Debugging Shift+F5
Break
Apply Code Changes Alt+F10
Step Into F11
Step Over F10
Step Out Shift+F11
Run to Cursor Ctrl+F10
Step Into Specific Function
Exceptions...
Threads...
Modules...
Show Next Statement Alt+Num *
QuickWatch... Shift+F9
isNewVerView
(All class
offBits=((LPBITMAPFIL
// 找到DIB图像象素起始
lpDIBBits = pDoc->Iri
// 获取当前DIB颜色数目
iColorNum = pDoc->Iri
if (iColorNum == 256)
{
// 提示用户
MessageBox("不是真
MB_ICONINFORM
return;
}
// 更改光标形状
BeginWaitCursor();
// 计算DIB宽度
lWidth = pDoc->IrisIm
// 计算DIB高度
lHeight = pDoc->IrisImg.Height();
// 计算图像象素总数
lCountSum = lHeight * lWidth;
// 分配内存
Red = new CHAR[lCountSum];
Green = new CHAR[lCountSum];
Blue = new CHAR[lCountSum];
// 计算图像每行的字节数
lLineBytes = (lWidth * 8+31)/32;
// 重置计数为0
for (i = 0; i < lCountSum; i ++)
{
ontext:
ame
Value
Auto Locals this
Name
Value
Red
CXX0017: Error: symbol "Red" not fo
Green
CXX0017: Error: symbol "Green" not
Blue
CXX0017: Error: symbol "Blue" not
Watch1 Watch2 Watch3 Watch4

1. Concept of Event-based System

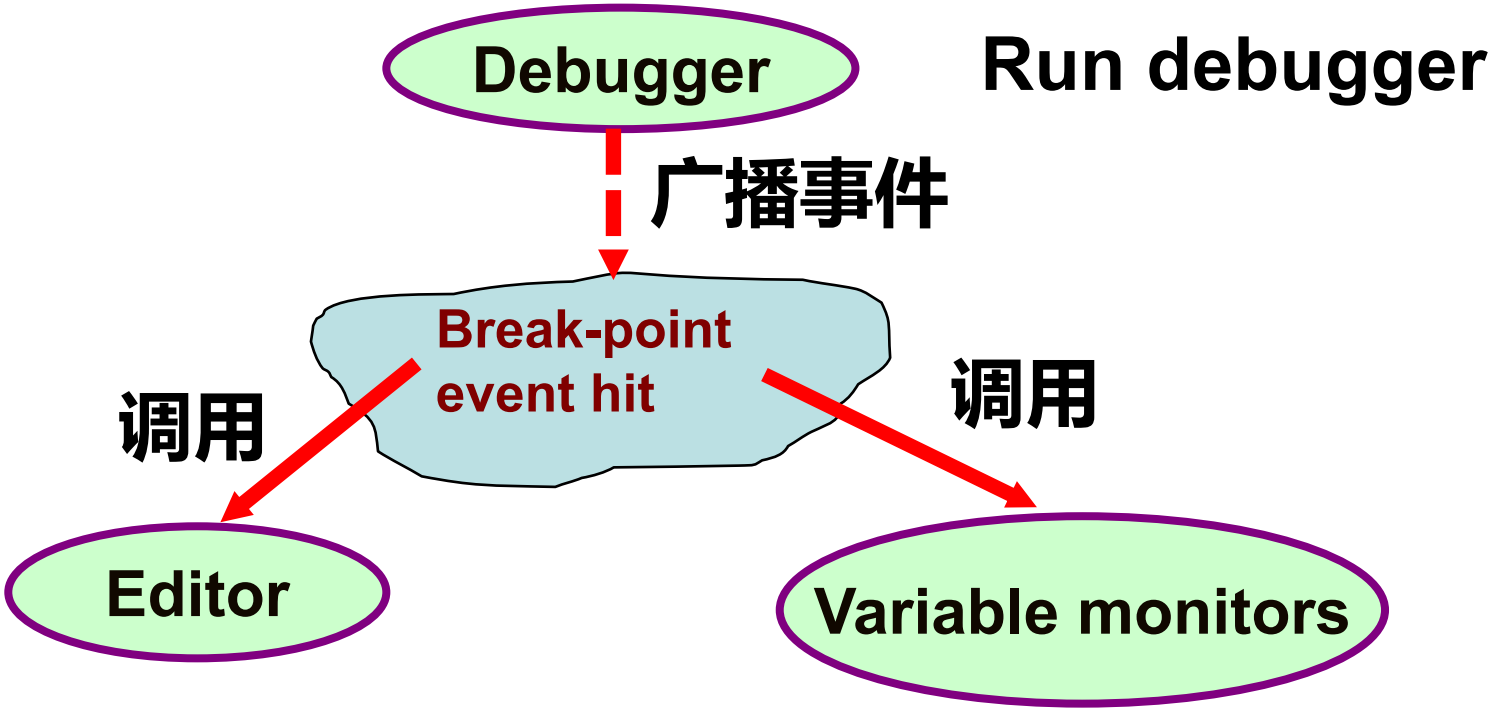
Editors and variable monitors register for the debugger's breakpoint events.



Legend: ·····→ Register event ——→ Send event

1. Concept of Event-based System

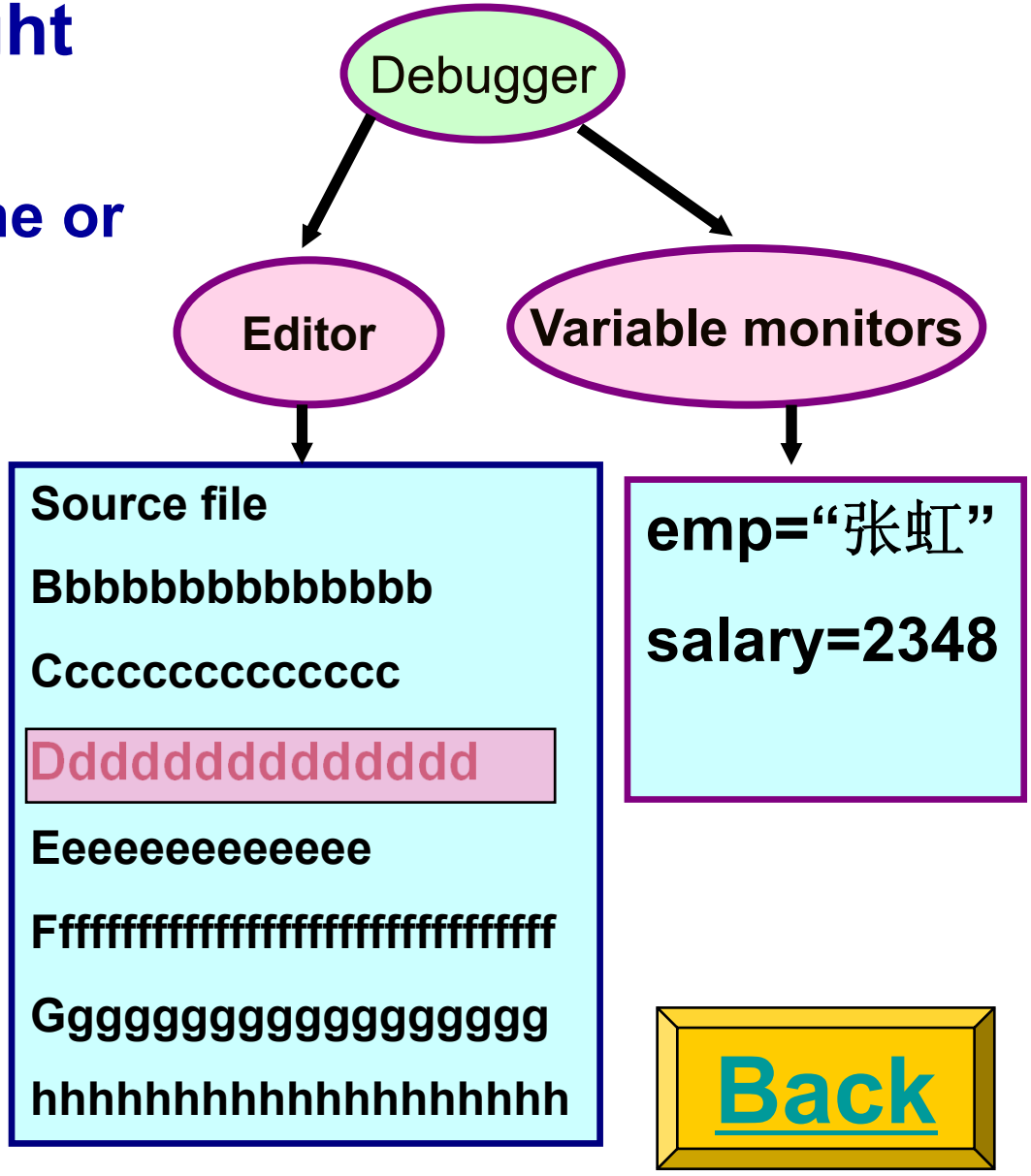
When a debugger stops at a breakpoint, it announces an event that allows the system to automatically invoke procedures of those registered tools.



1. Concept of Event-based System

These procedures might

- **scroll an editor to the appropriate source line or**
- **display the value of monitored variables.**



注：

- 调试器仅仅广播了一个事件，但是不知道其它的组件将要做什么。
- 松耦合

Strategy of Events Handling
事件处理策略

2. Strategy of Events Handling

- 当一个事件被广播了，系统将自动调用那些已经注册了的组件. When an event is announced, the system itself automatically invokes all of the procedures that have been registered for that event.

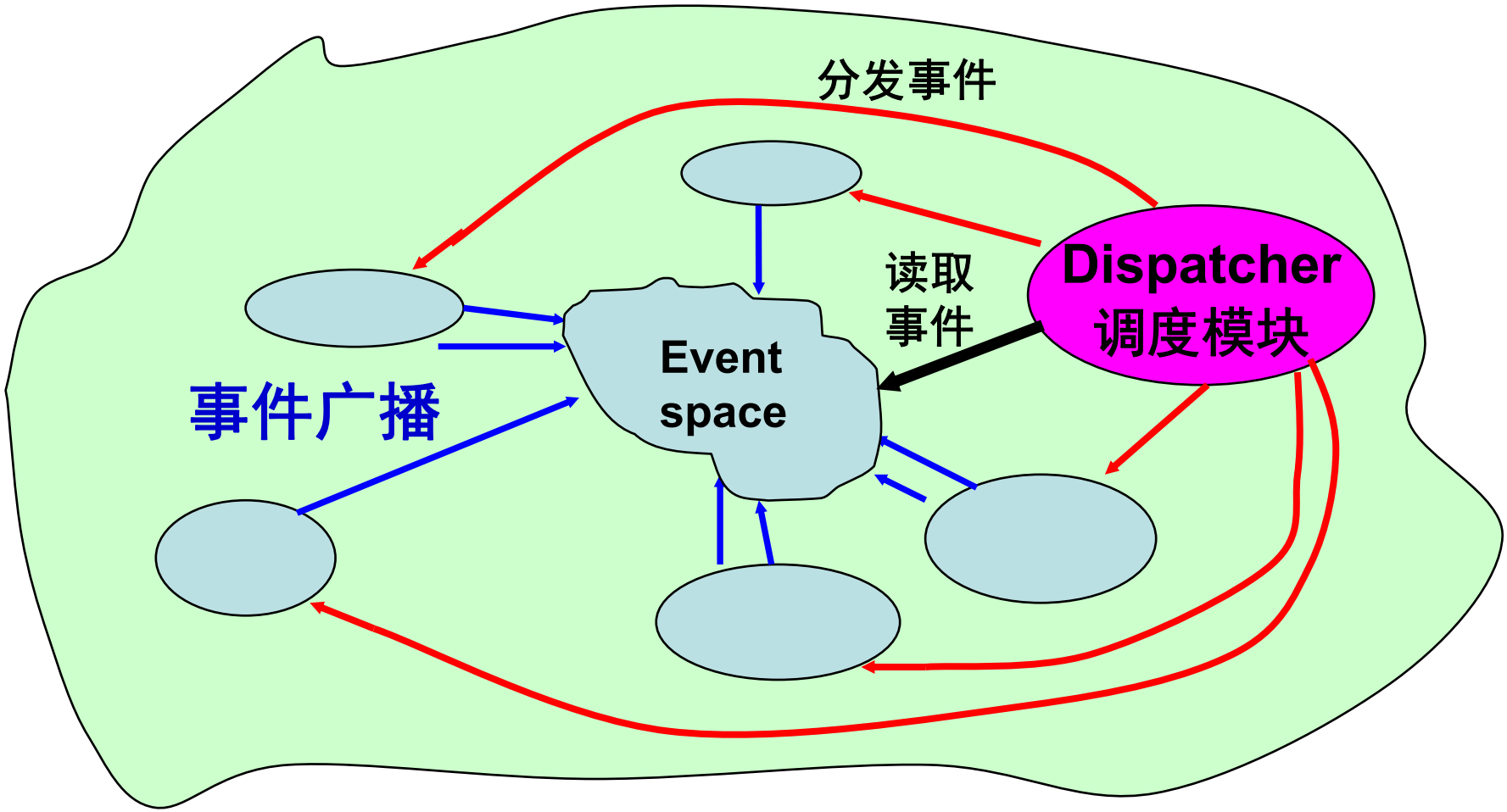
- 问题：怎样将事件发送到已经注册了的组件中呢?
  Strategies:
  - 有独立事件调度模块的系统。Systems with a separate dispatcher module
  - 无中心事件调度模块的系统。Systems without central dispatcher module

2. Strategy of Events Handling

- 事件调度模块的责任
- What the dispatcher module does?
- The dispatcher module is responsible for
  - 接收事件。receiving all incoming events and
  - 分发事件。dispatching them to other modules in the system.

2. Strategy of Events Handling

Strategy 1. Systems with Separate Dispatcher Module (有独立事件调度模块的系统)



2. Strategy of Events Handling

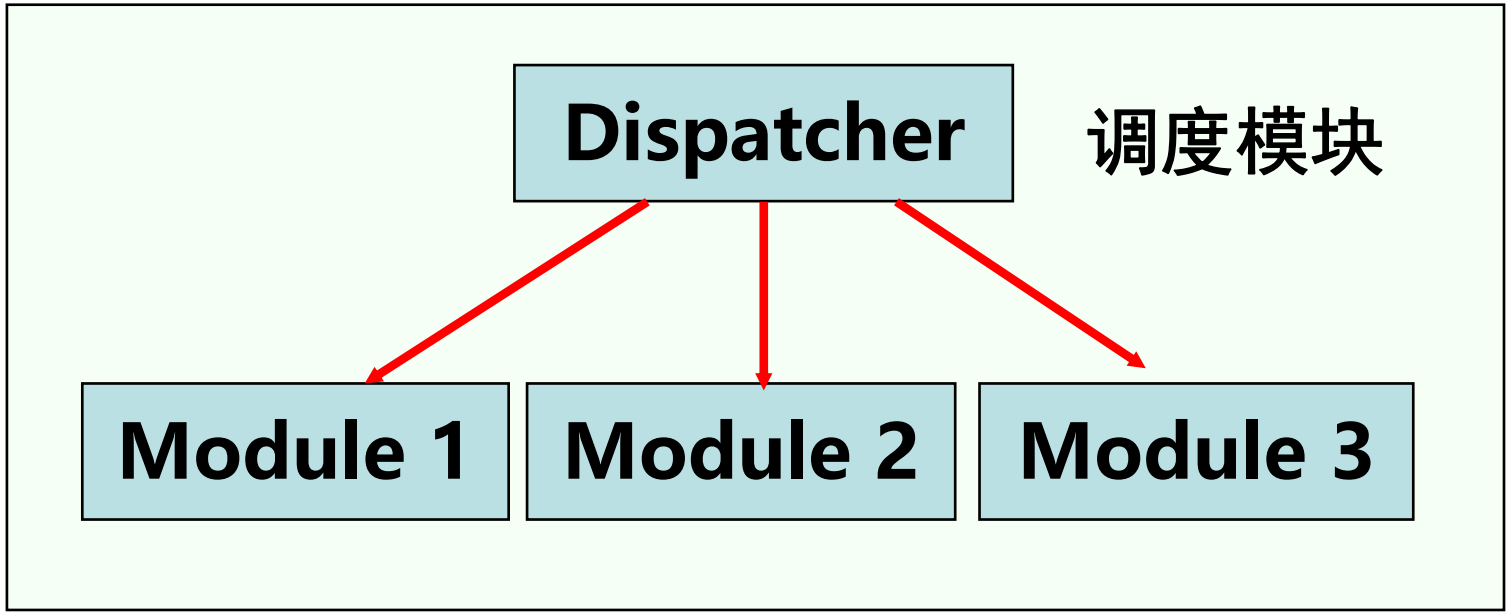
调度模块以两种方式分发事件

Two ways to dispatch events

1. **The dispatcher may broadcast events to all modules in the system**
2. **The dispatcher sends an event just to those modules that registered for that event: Publish/Subscribe strategy (发布/订阅策略)**

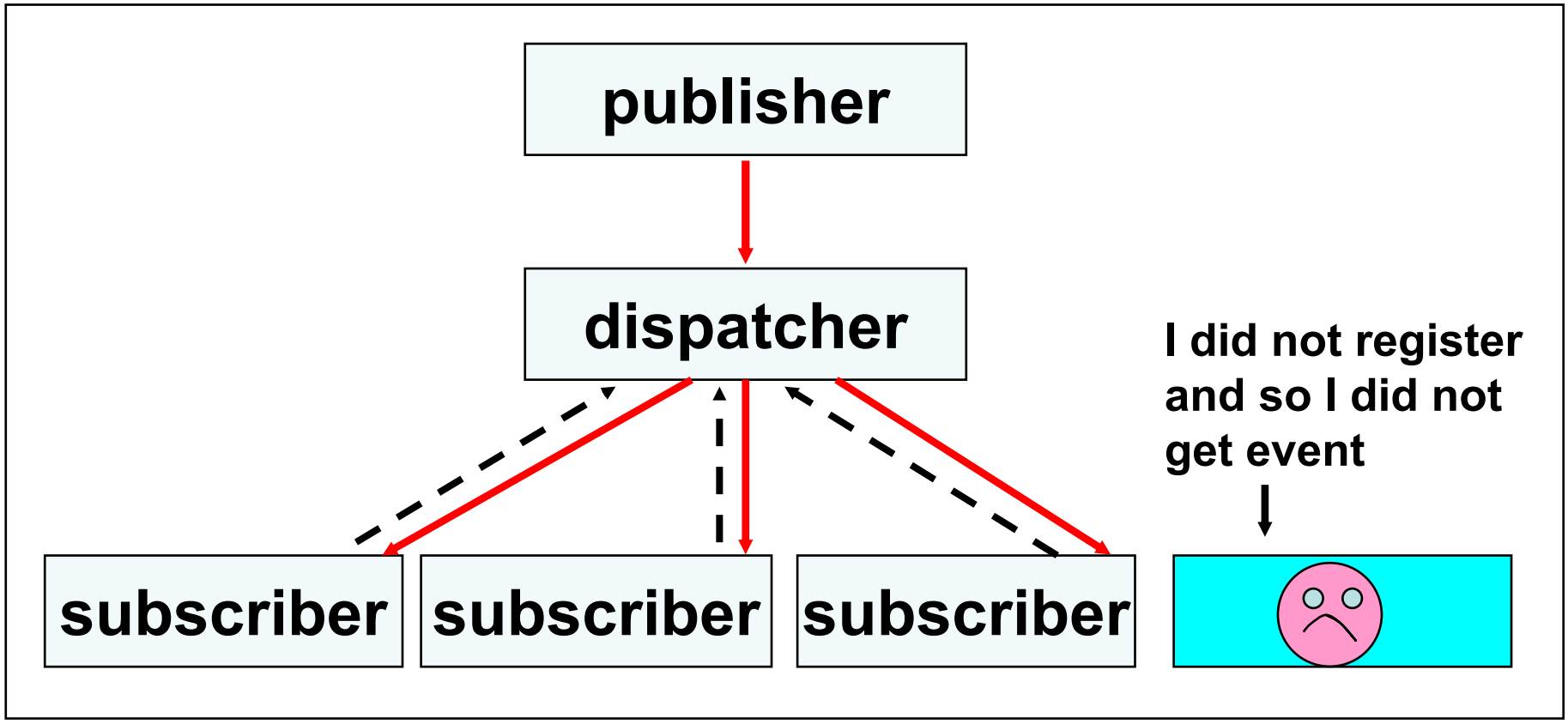
2. Strategy of Events Handling

The dispatcher may broadcast events to all modules in the system

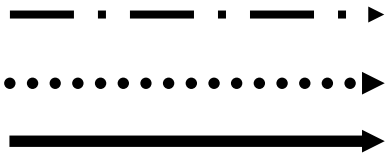


2. Strategy of Events Handling

The dispatcher sends an event just to those modules that registered for that event:



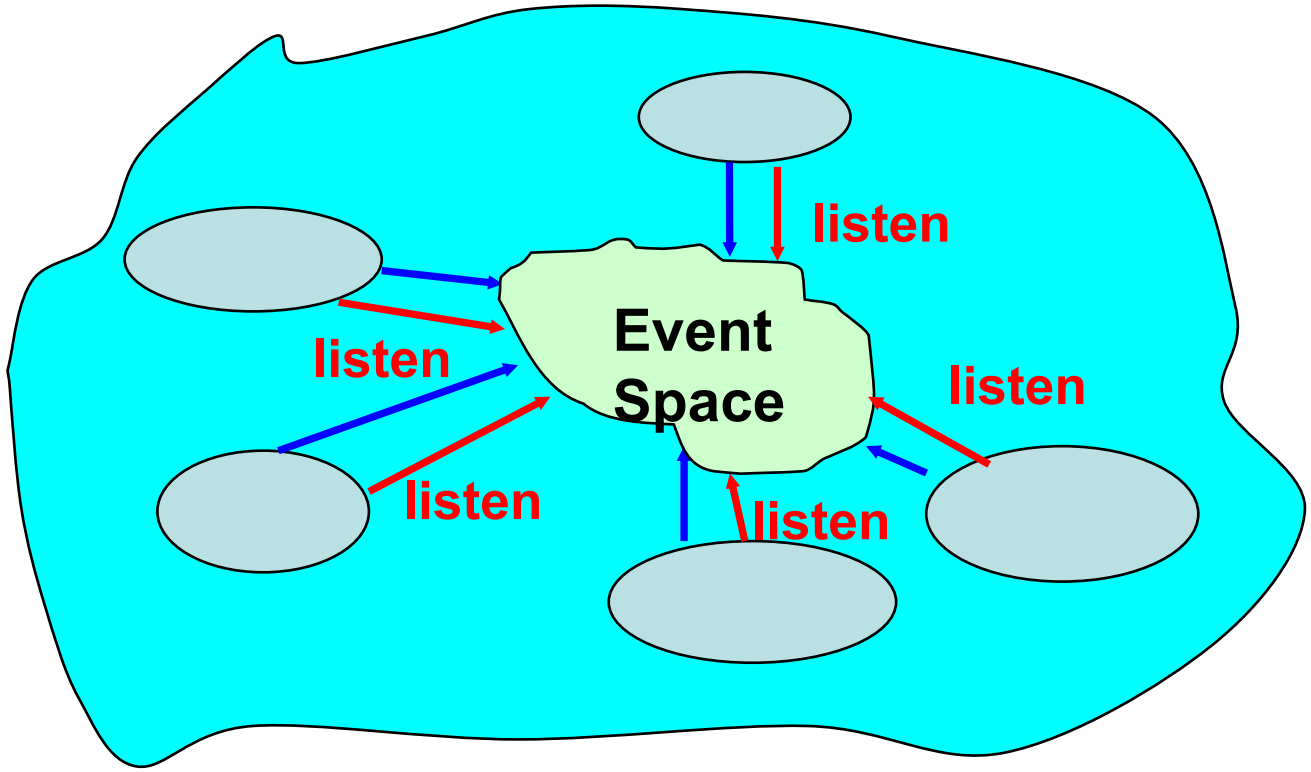
2. Publish/Subscribe strategy



Publish event
Subscribe
Send event

2. Strategy of Events Handling

Strategy 2. Systems without a central dispatcher module (无独立事件调度模块的系统)



Observable/Observer modle

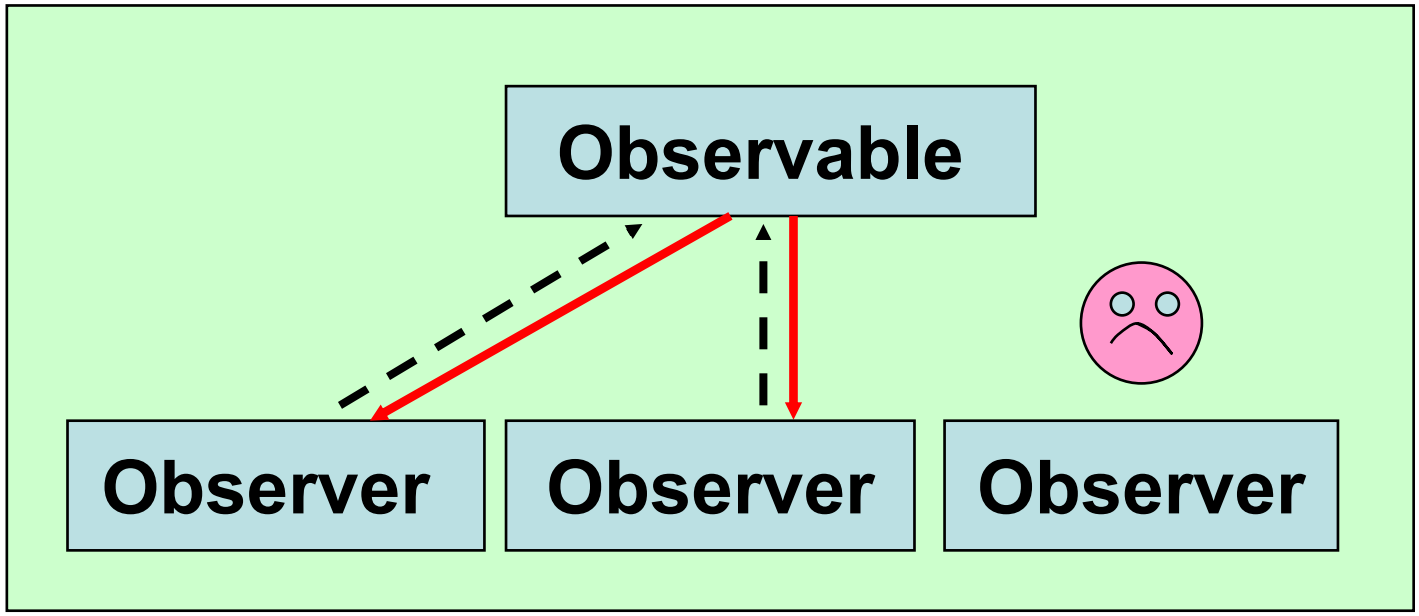
2. Strategy of Events Handling

This modle is usually called observable/Observer

- 每个模块都允许其它模块对其所发送的事件感兴趣
  Each module allows other modules to declare interest in events that it is sending.

- 只将事件发送给注册者
  Whenever a module sends an event it sends that event to exactly those modules that registered interest in that event.

2. Strategy of Events Handling



Observable/Observer modle

(被观察者/观察者模型)

Legend: ·····→ Register event ——→ Send event

Back

Example Design of an event system Using the Observer Pattern

一个使用观察者模式进行事件系统设计的例子

4. Design Examples Using the Observer Pattern

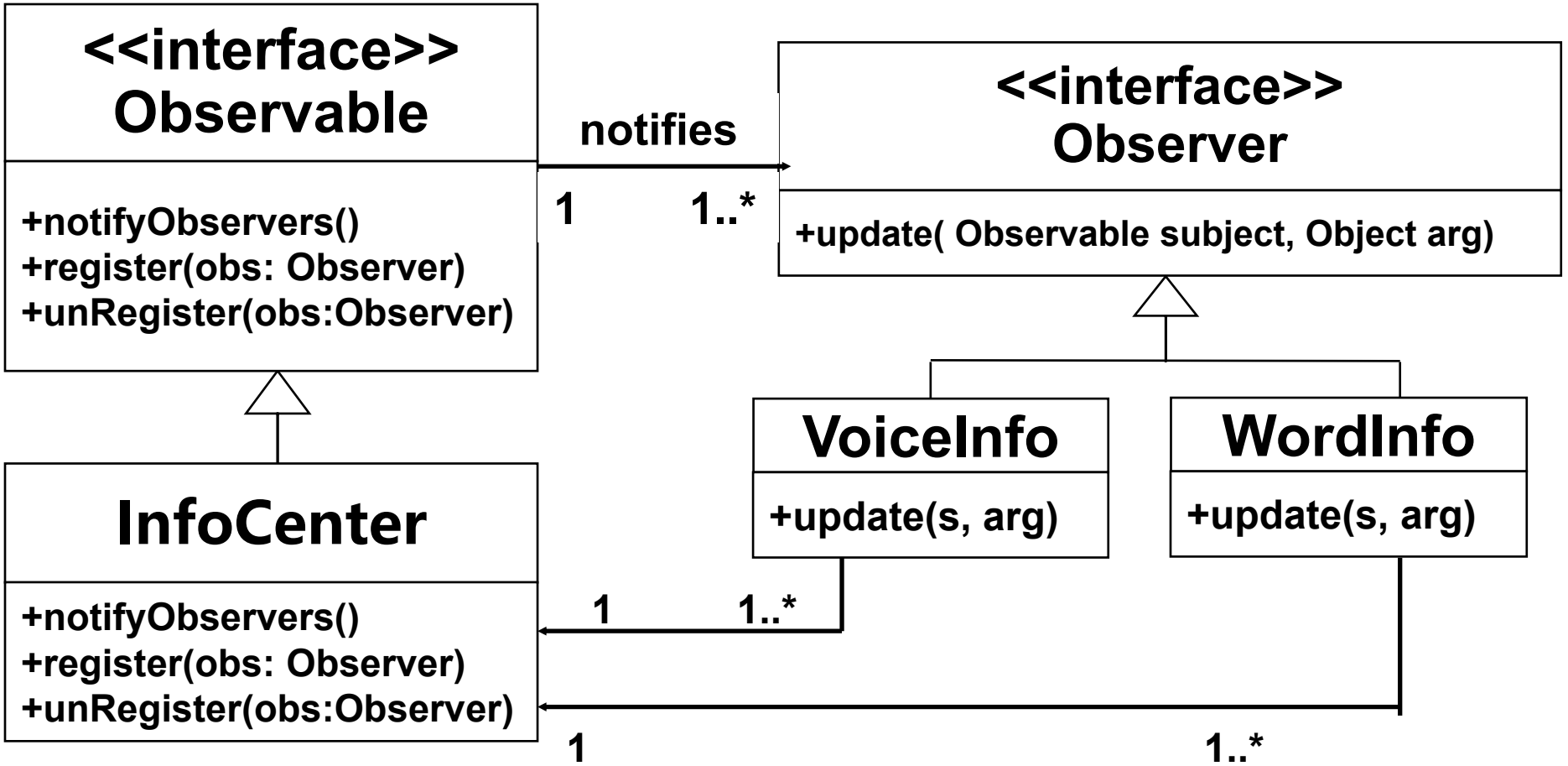
- 观察者模式可被用来设计与实现比较简单的事件系统
- 观察者模式可以被认为是Observable/Observer模型（事件空间包含在Observables里面）的一个实现。

4. Design Examples Using the Observer Pattern

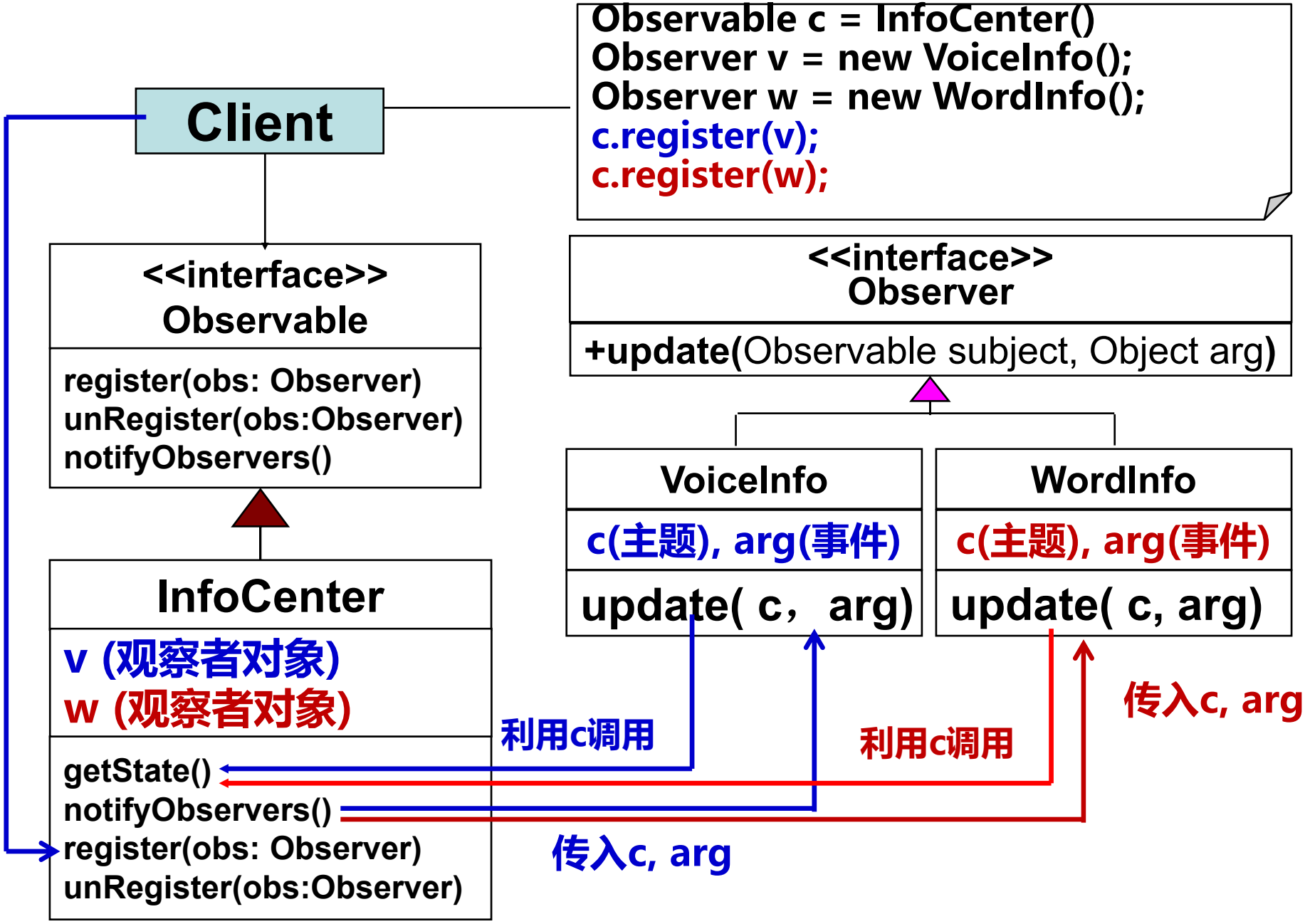
Observerble/Observer example

- **Example**: **机场信息显示的例子。** The airport information system is to receive incoming air flight information and inform the customers information, including on time, delay, and other information about all the arrival airplanes and departure airplanes
- **InforCenter** class: collect all air flight information
- Classes:
- **VoiceInfo**: broadcast air flight info to all travelers
- **WordInfo**: display word info onto LED display

4. Design Examples Using the Observer Pattern



利用观察者模式设计的机场信息发布系统



利用观察者模式设计的机场信息发布系统的典型交互

4. Design Examples Using the Observer Pattern

- **Design highlights: Two interfaces**
  - **Observable**
  - **Observer**

  with exactly all of their methods claimed in the design.

- **In InfoCenter, you need to implement all the methods:**
  - **notifyObservers()**
  - **register(obs: Observer)**
  - **unRegister(obs:Observer)**

4. Design Examples Using the Observer Pattern

- In the **VoiceInfo** and **WordInfo**, you need to implement the method
  - **update(Observable subject, Object arg)**

Back