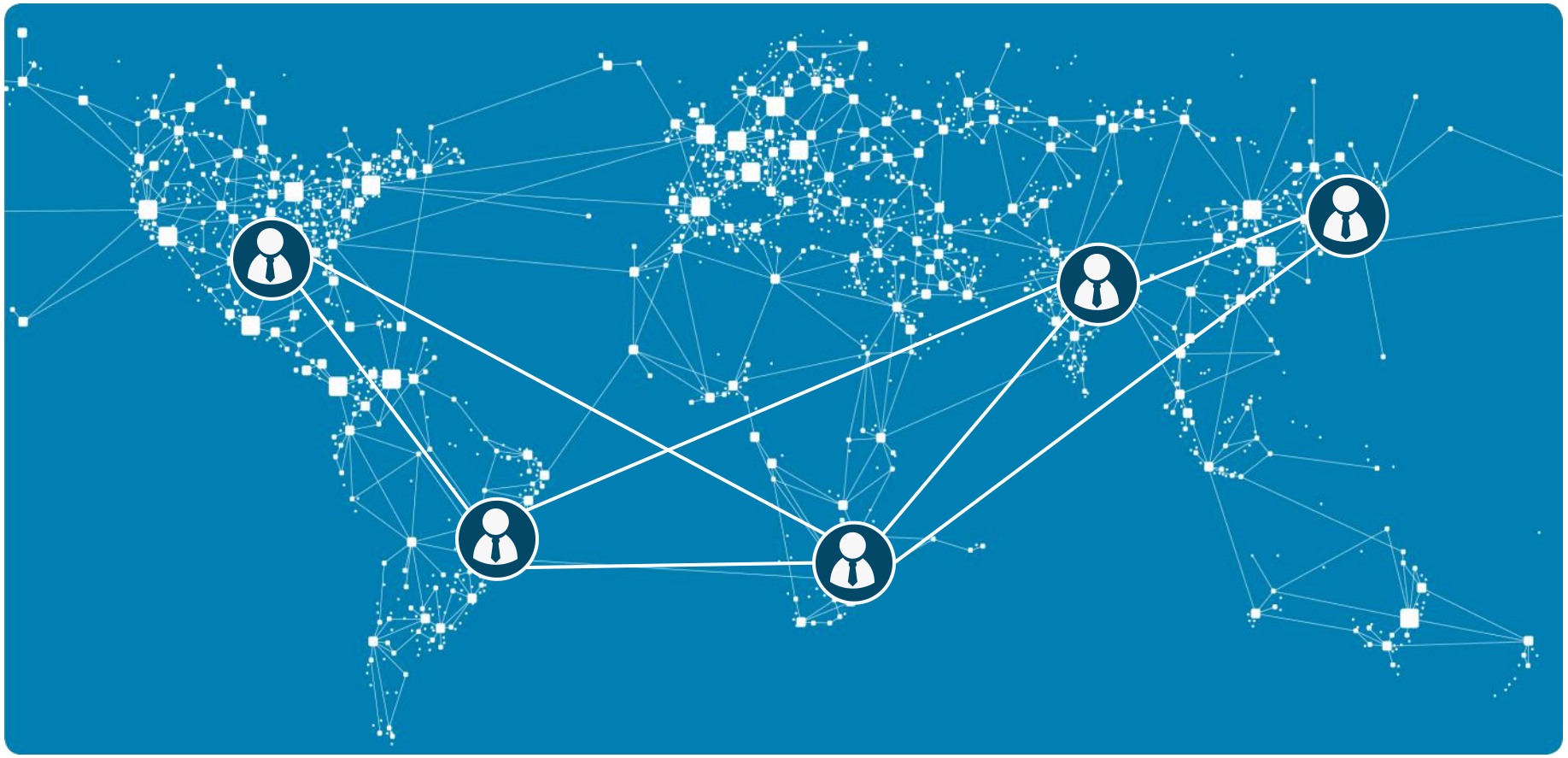


行为建模

行为建模

顺序图建模过程

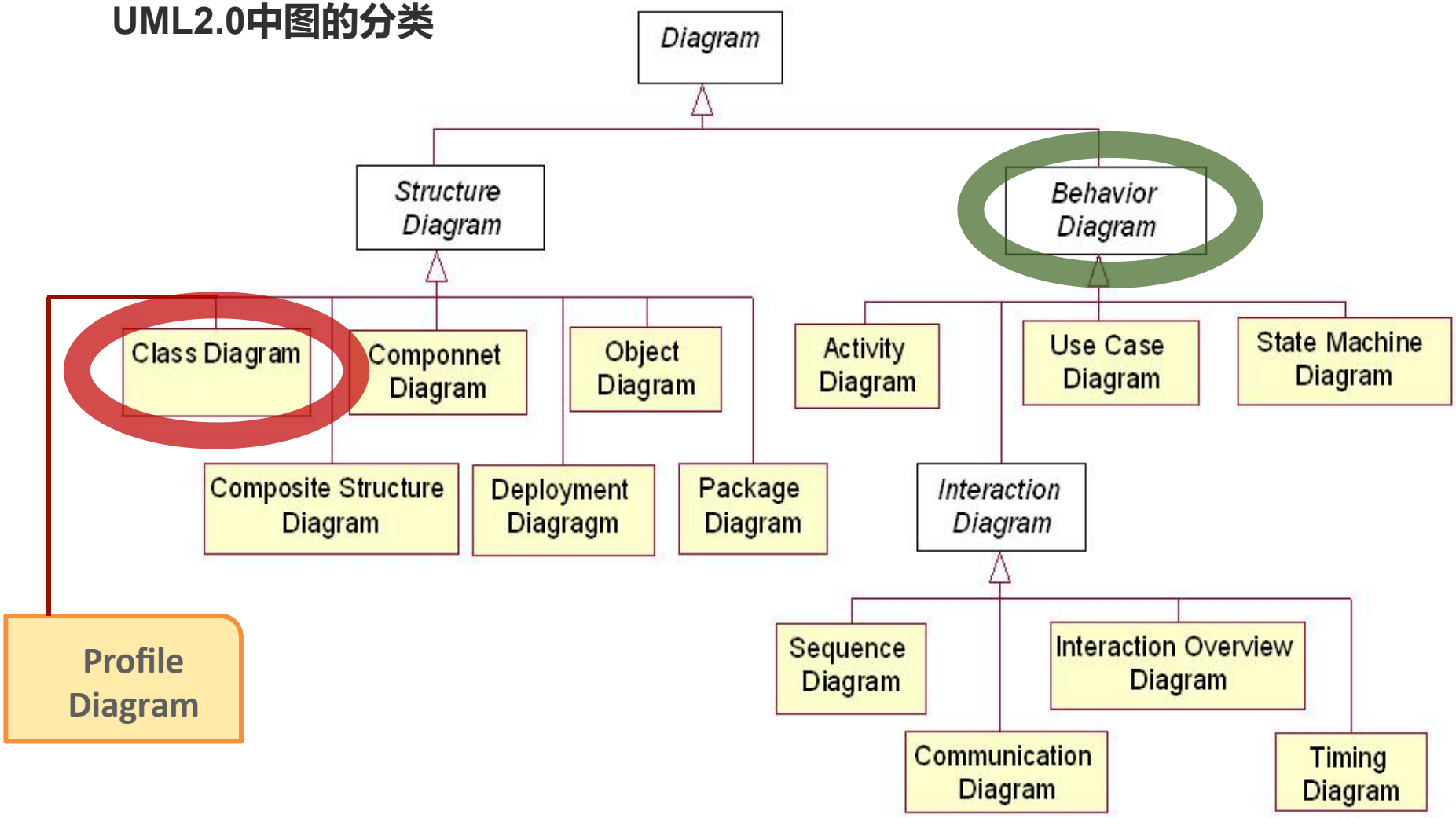
顺序图建模风格

状态建模

状态图绘制

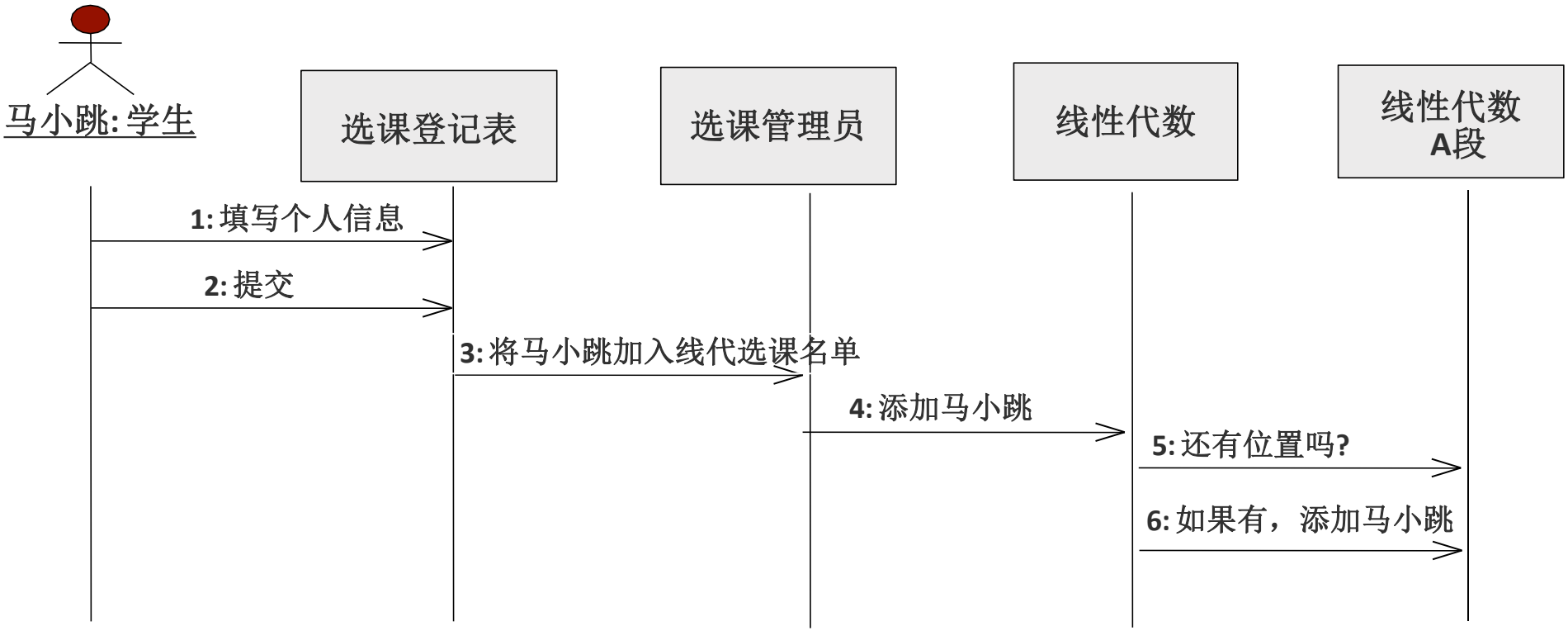
顺序图概念

UML2.0中图的分类



顺序图举例

- 顺序图用来刻画系统实现某个功能的必要步骤

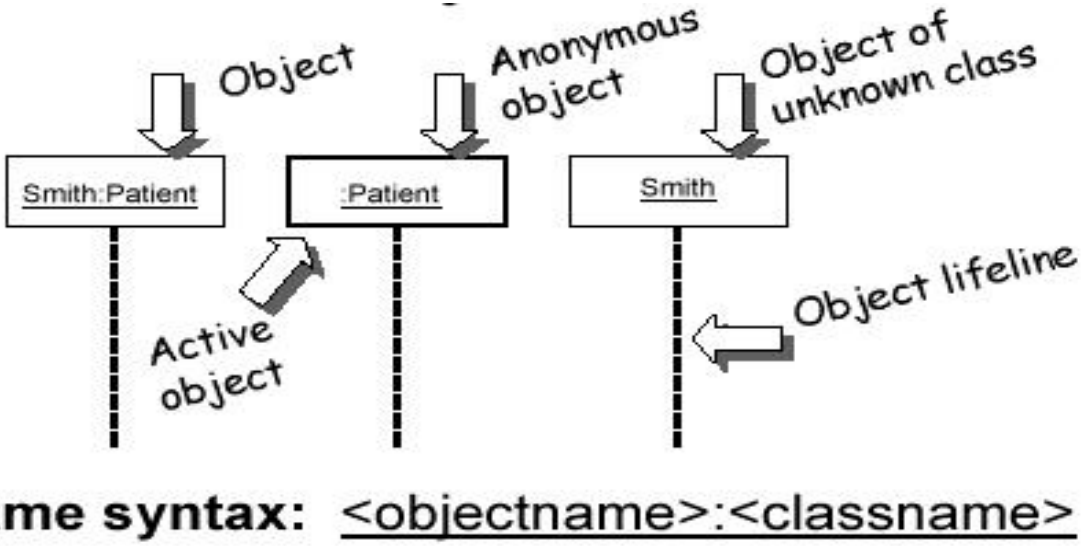


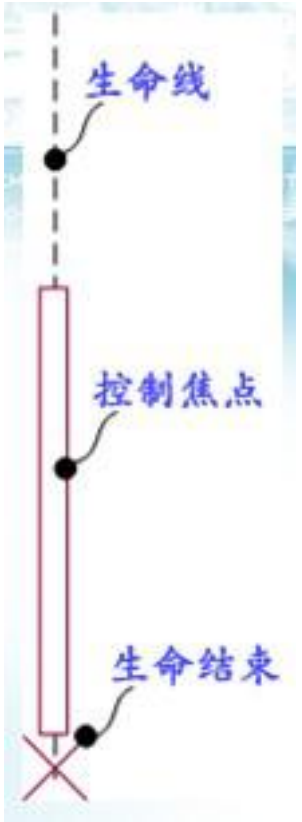
顺序图建模元素--对象（Object）及其生命线（Lifeline）

- 对象以某种角色参与交互

  可以是人、物、其他系统或者子系统

- 生命线：表示对象存在的时间



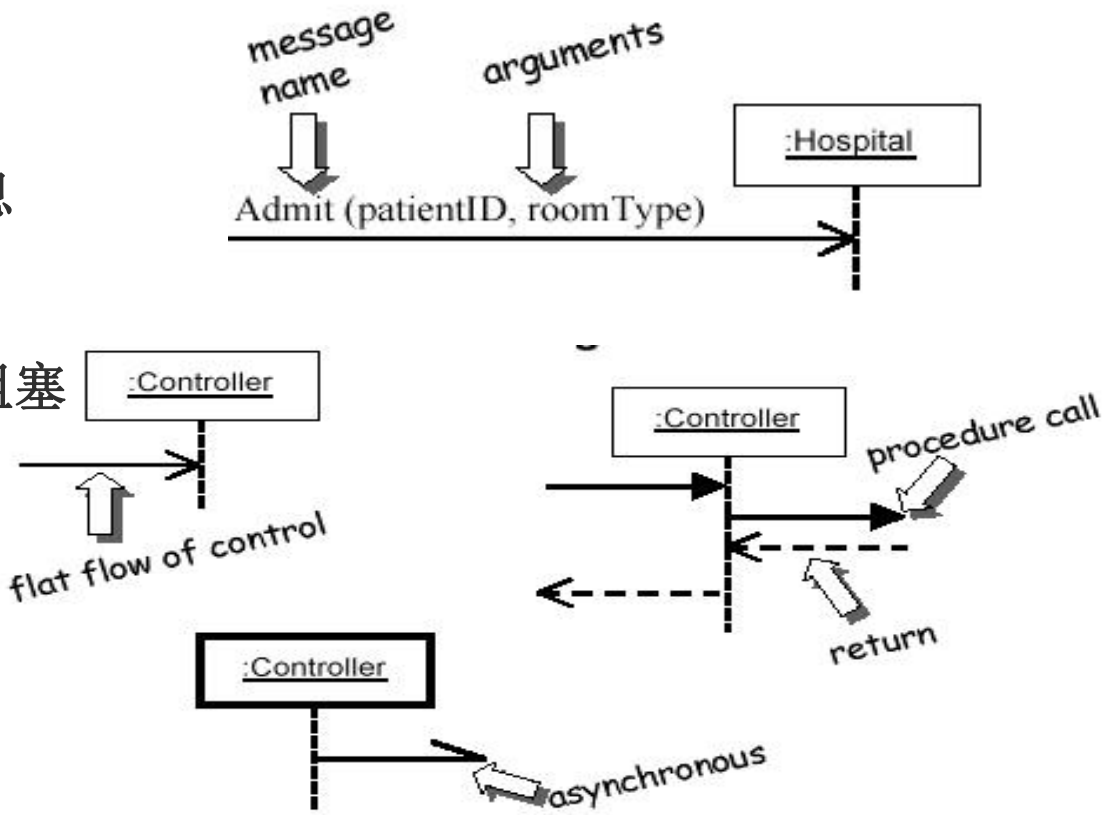


- 控制焦点/激活期(Focus of Control/Activation)：表示对象进行操作的时间片段

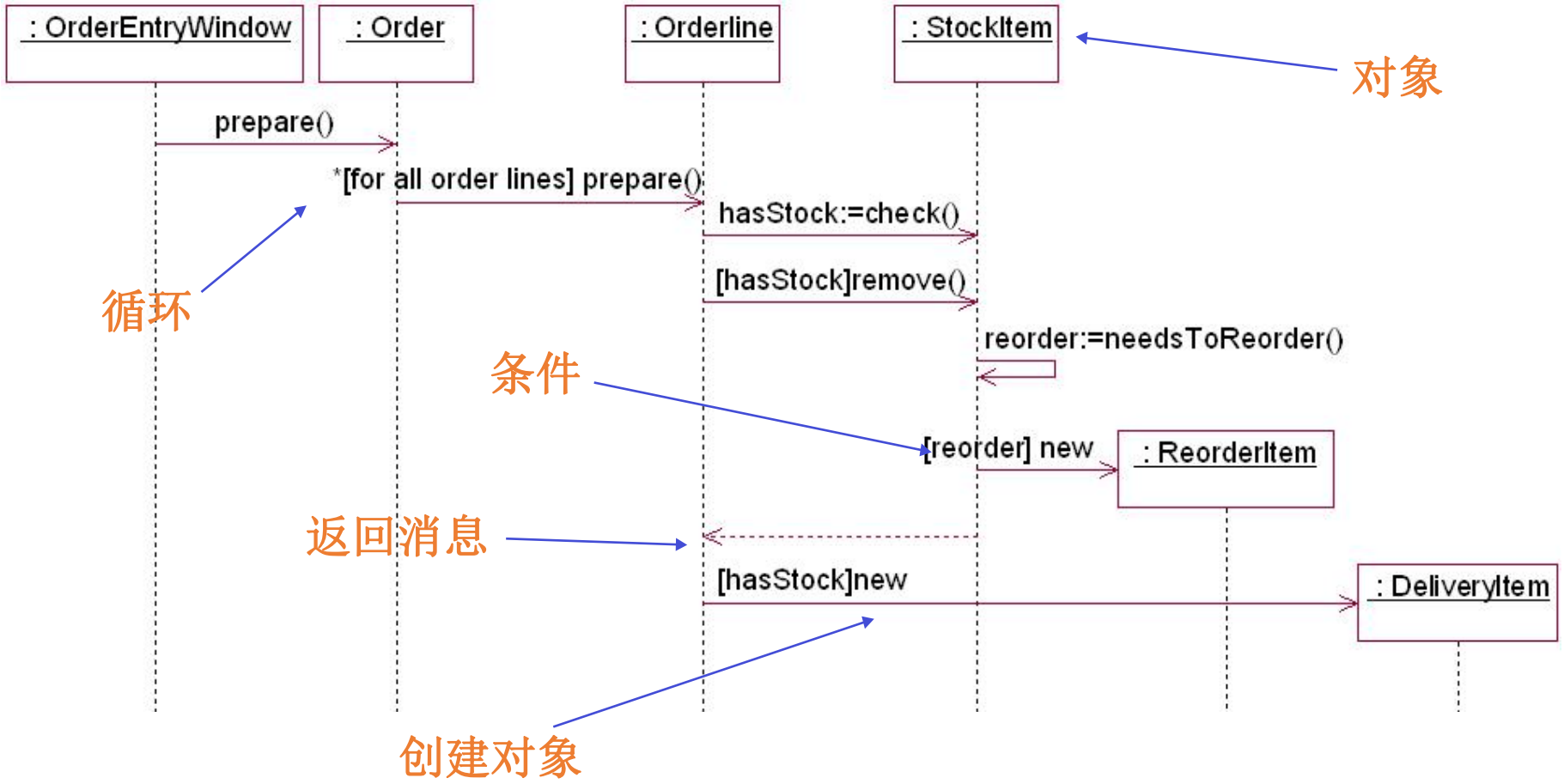
顺序图建模元素--消息（Message）

消息（Message）用于描述对象间的交互操作和值传递过程

- 消息类型:
  - Synchronous 同步消息（调用消息）
  - Asynchronous 异步消息
  - Return 返回消息
  - Self-message 自关联消息
  - Time-out 超时等待
  - Uncommitted / Balking 阻塞



顺序图中的基本结构


: OrderEntryWindow
: Order
: Orderline
: StockItem
对象
prepare()
*[for all order lines] prepare()
hasStock:=check()
[hasStock]remove()
循环
reorder:=needsToReorder()
条件
[reorder] new
: ReorderItem
返回消息
[hasStock]new
: DeliveryItem
创建对象

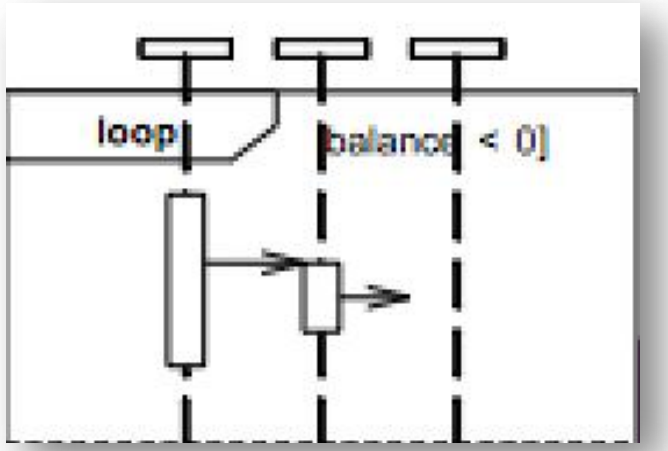
顺序图中消息的循环发送

在消息名字前加循环条件或添加循环控制框

例：

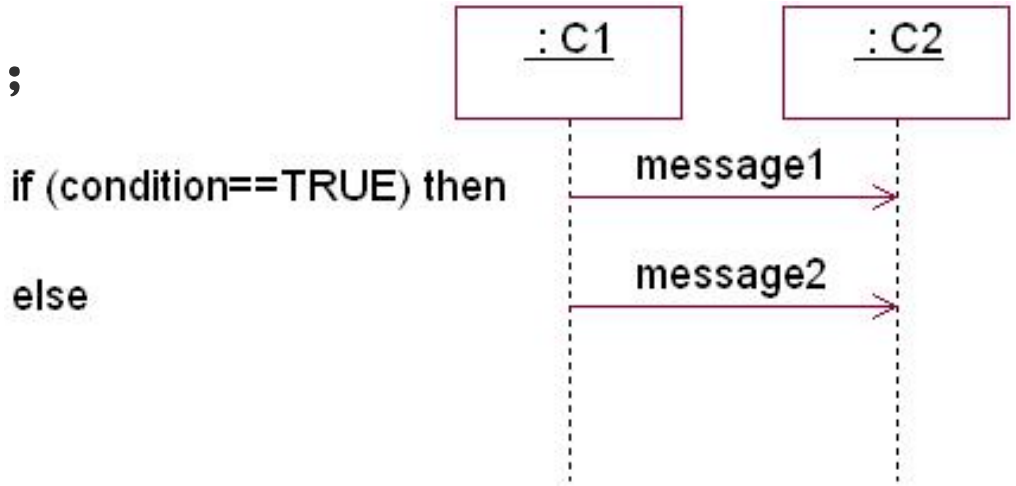
1.1 *[for all order lines]: message1()

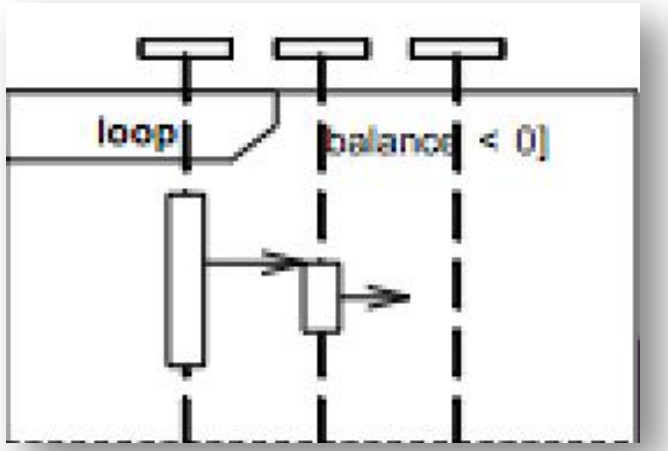
2.1 *[i:=1..n]: message2()



顺序图中带条件消息的发送

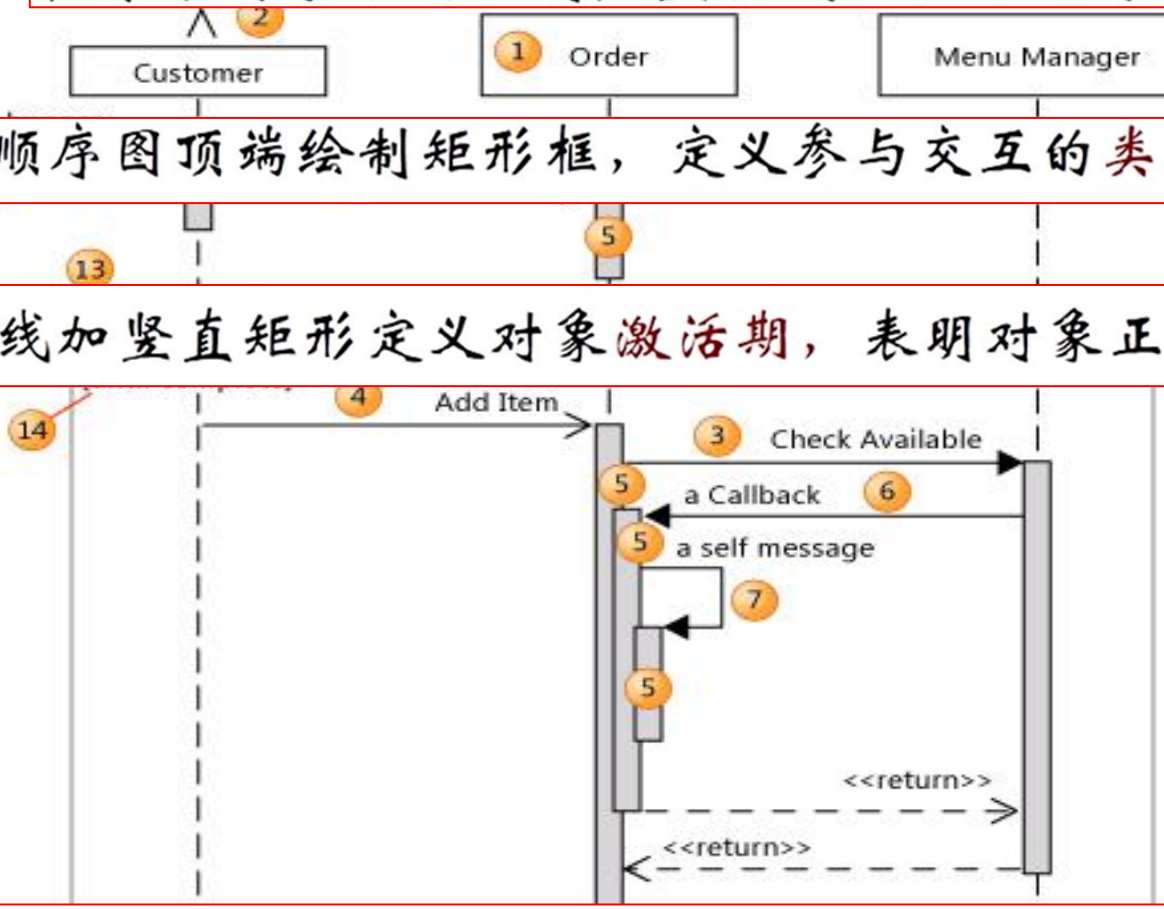
- 在消息名字前加条件子句；
- 使用文字说明；
- 添加条件控制框；
- 分成多个顺序图子图并关联





顺序建模过程

在每个对象下面绘制竖直虚线，表示该对象的生命线；



在顺序图顶端绘制矩形框，定义参与交互的类实例（对象）名；

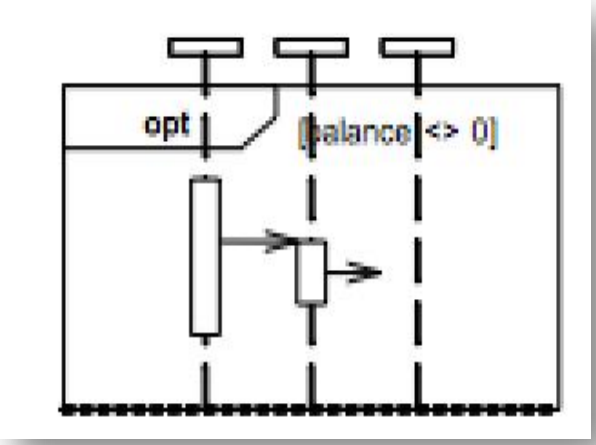
生命线加竖直矩形定义对象激活期，表明对象正在执行某操作；

在对象间添加箭头表示各种类型的消息，跟踪对象间的控制流；

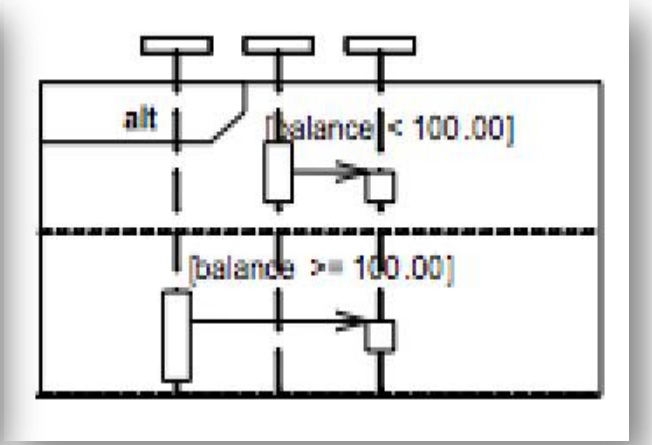
根据需要添加框的组合与关联，表示复杂的控制结构。

组合框：复杂控制结构表示

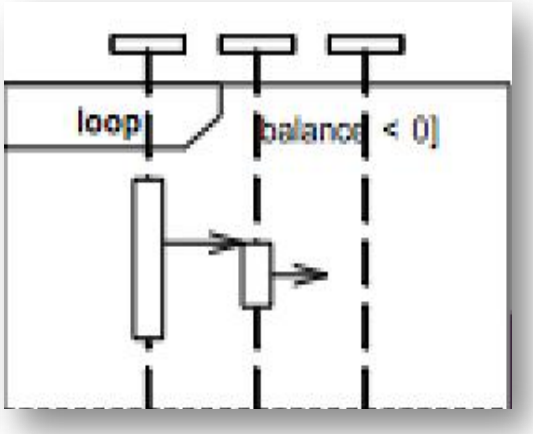
- **框（Frame）**：框中包含顺序图的部分结构，表示选择（selection）或者循环（loop）结构，左上角注明结构类型，[]中注明条件。



If -> (**opt**)[condition]

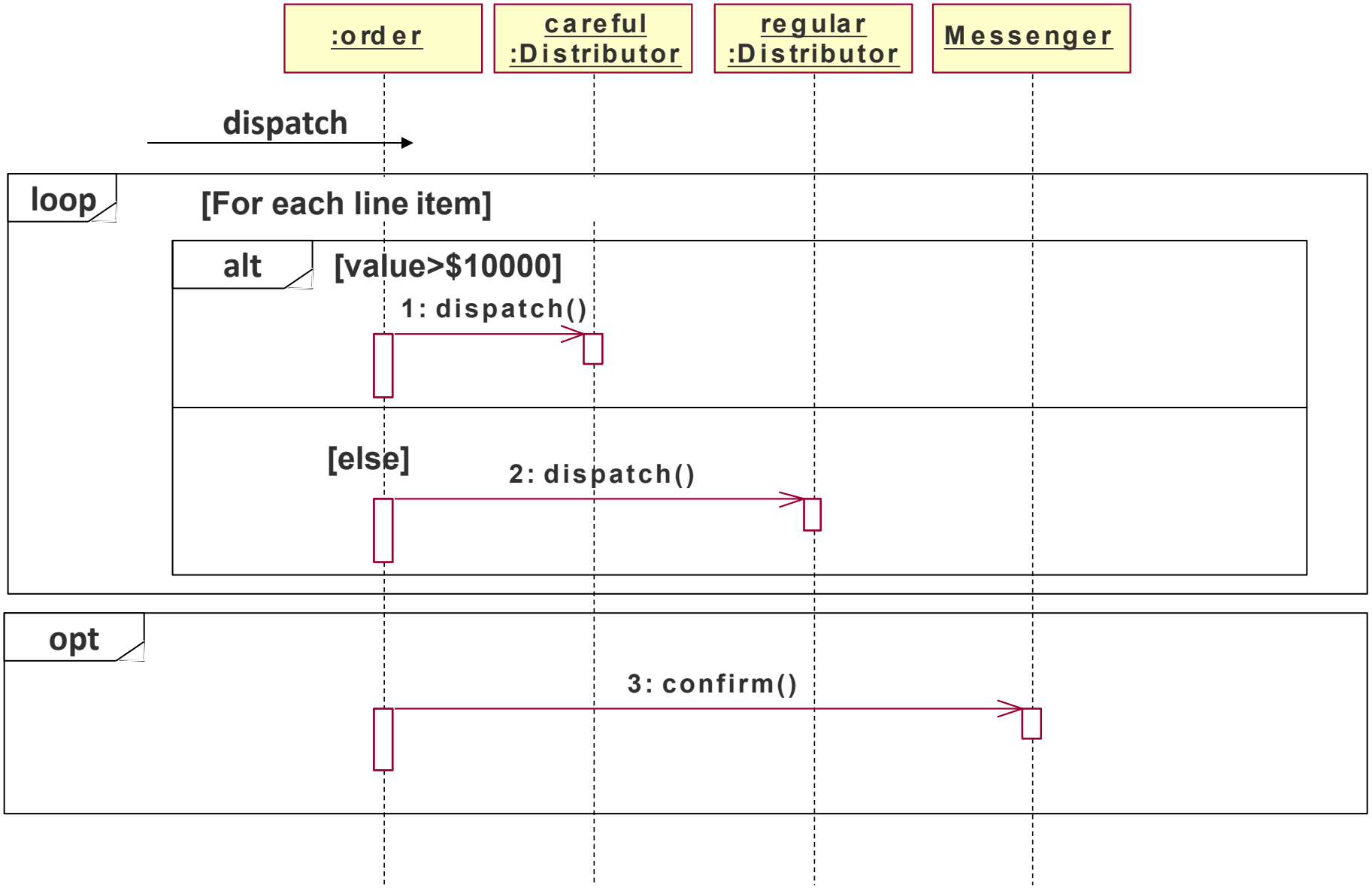


if/else -> (**alt**)[condition]
通过水平虚线分割不同情形
并发结构 -> (**par**)

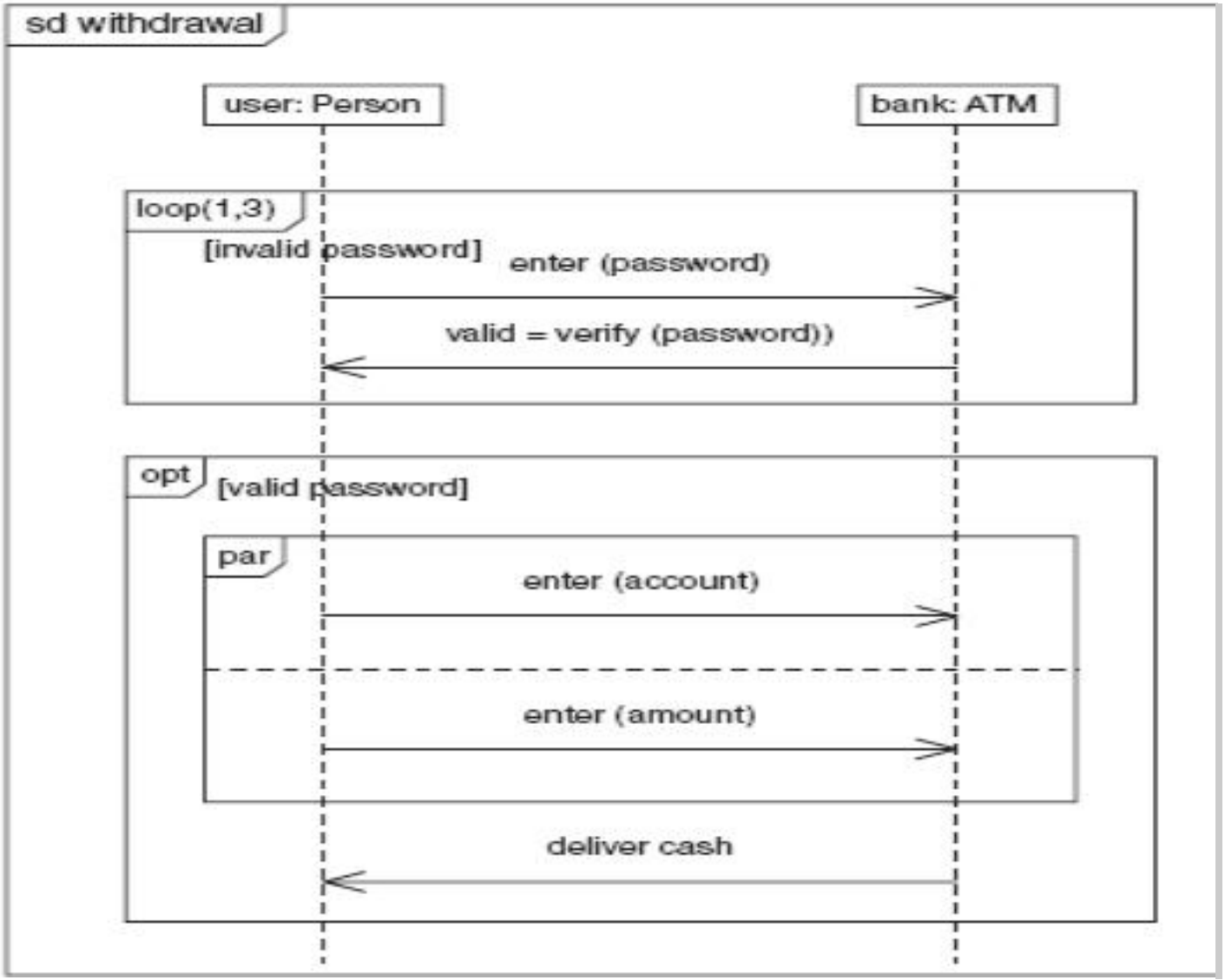


loop -> (**loop**)[condition or items to loop over]

控制框建模例子

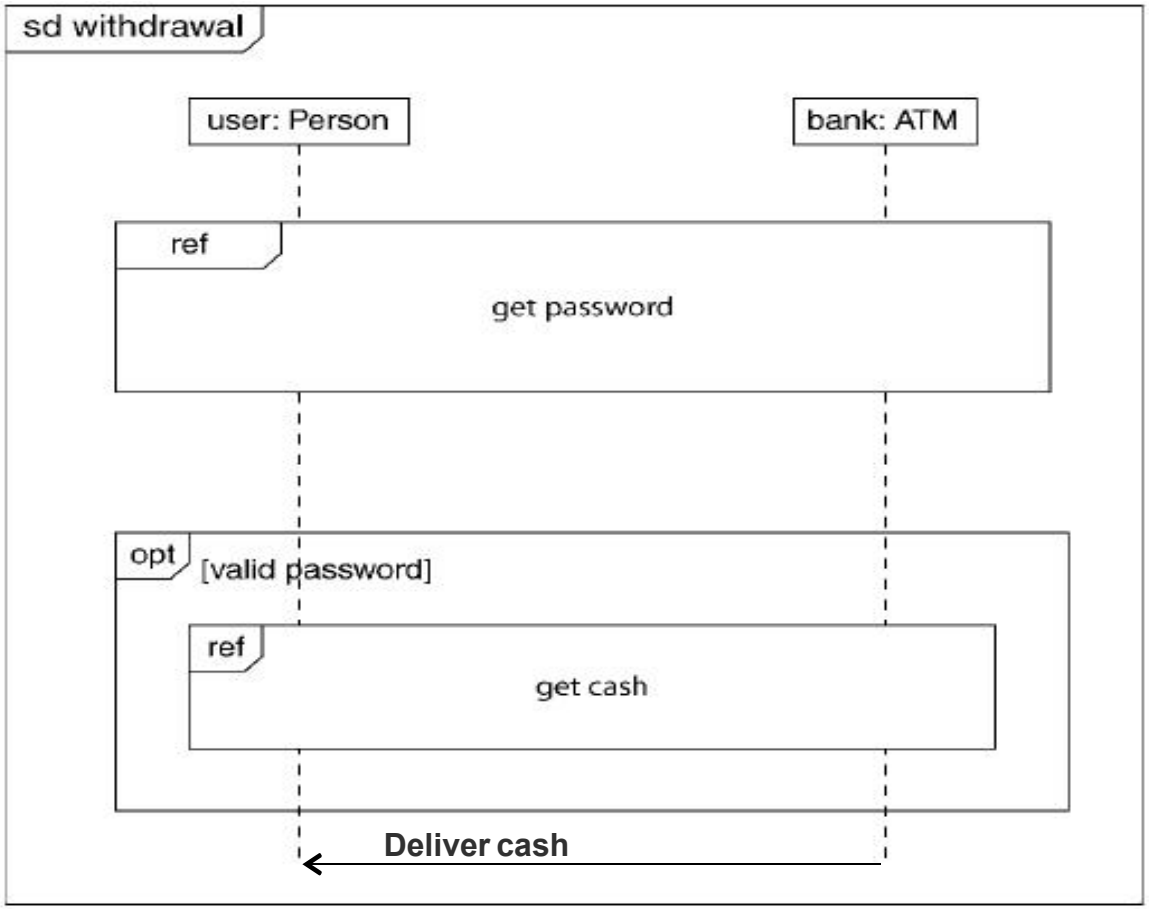

:order
careful :Distributor
regular :Distributor
Messenger
dispatch
loop
[For each line item]
alt
[value>$10000]
1: dispatch()
[else]
2: dispatch()
opt
3: confirm()

控制框建模例子


sd withdrawal
user: Person
bank: ATM
loop(1,3)
[invalid password]
enter (password)
valid = verify (password))
opt
[valid password]
par
enter (account)
enter (amount)
deliver cash

顺序图间的关联

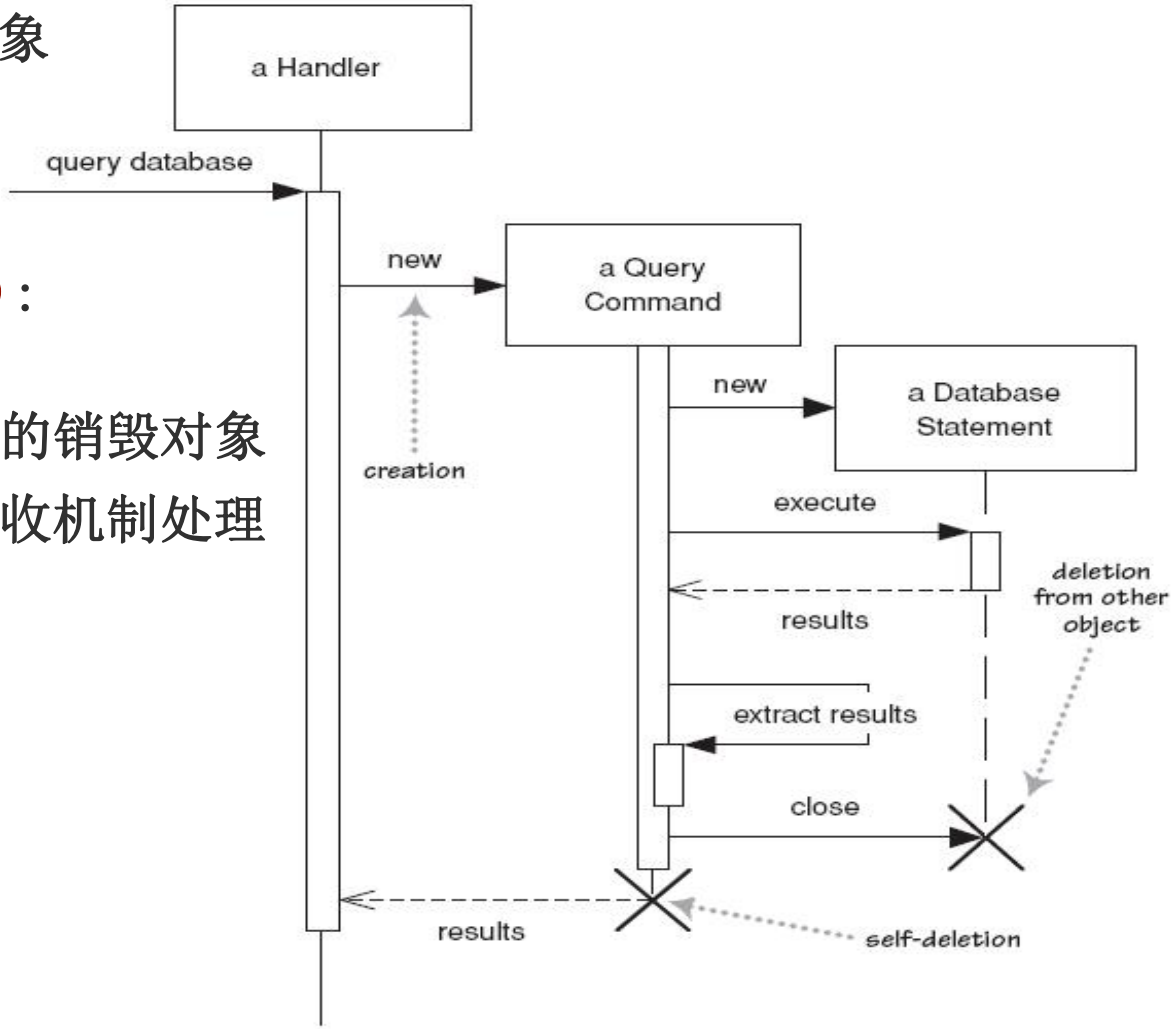
- 当一个顺序图过大
- 需要引用其他图表时，选择下述表示：
  - 不完整的箭头和注释
  - 通过名为”ref”的框图引用相关图表



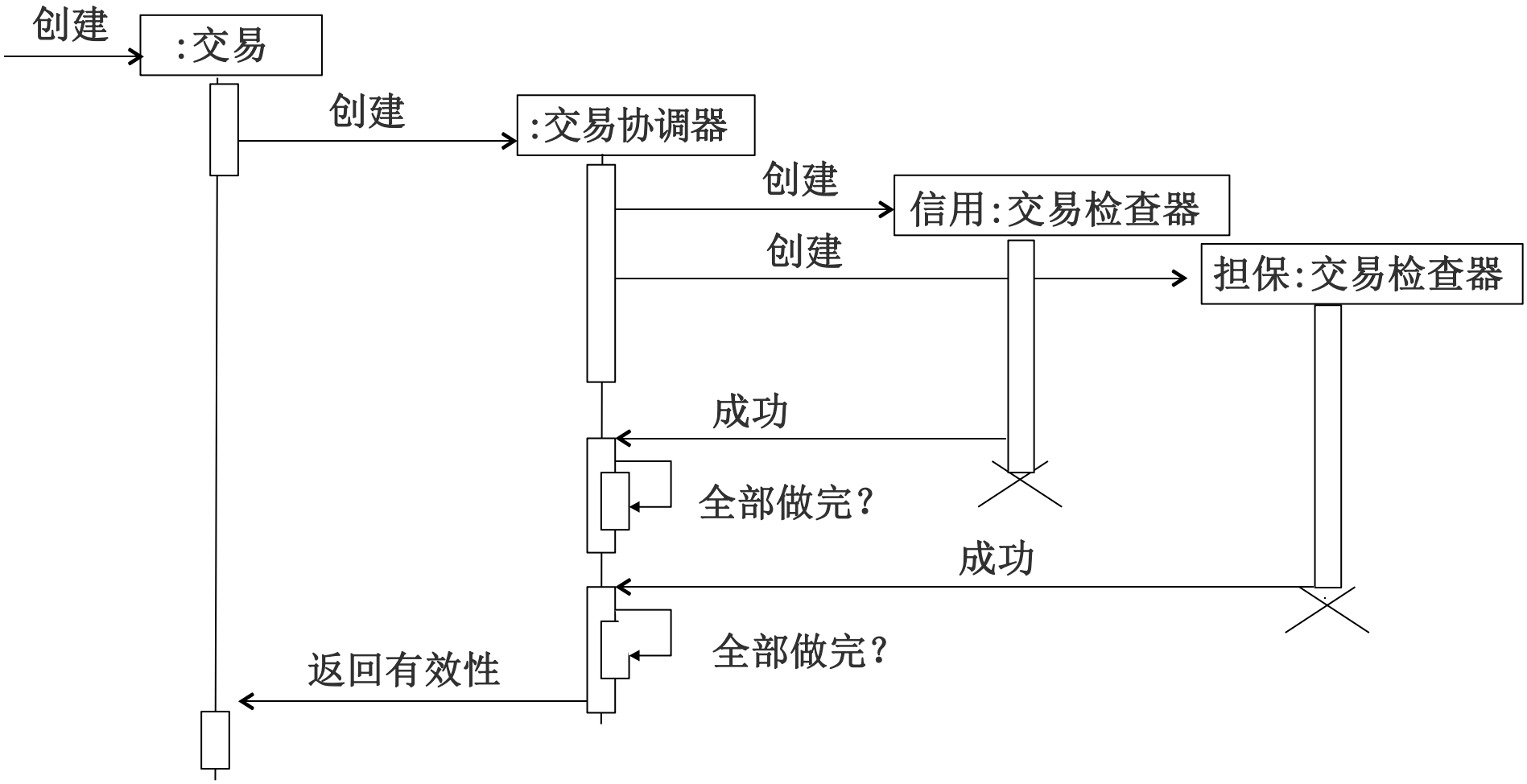
对象的创建与撤销

- 创建(creation)：
  “new”标明的箭头
  - 用例场景中新建的对象在图中的位置较低

- 撤销(deletion, destroy)：
  生命线底部的“X”
  - 注：在Java没有明确的销毁对象的操作，通过垃圾回收机制处理



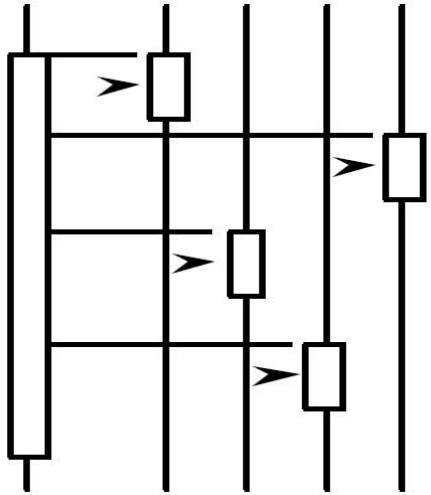
银行系统的交易验证

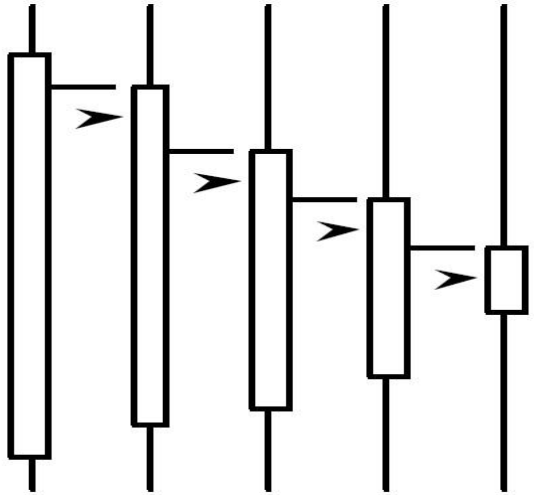


集中控制 or 分布控制

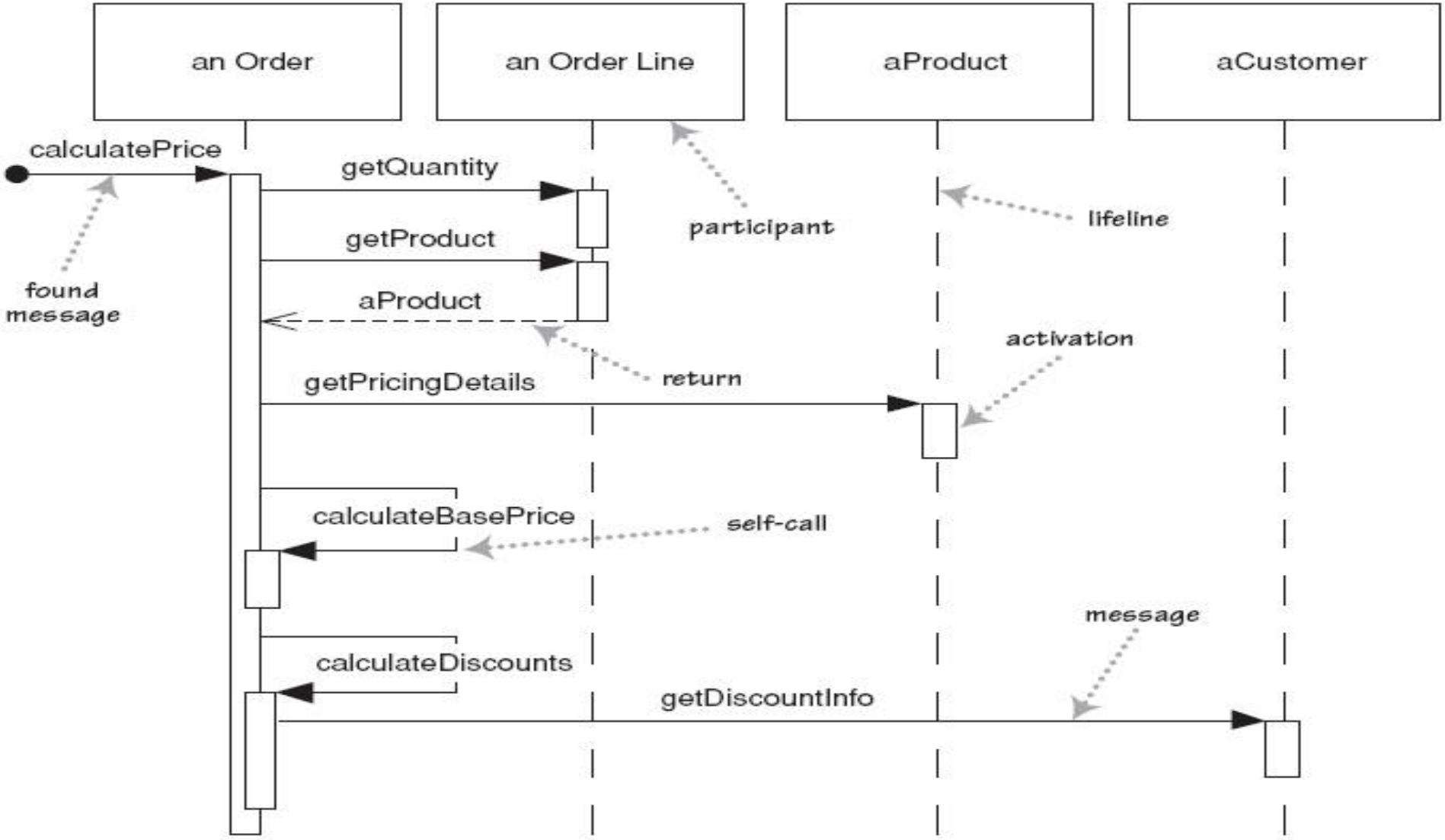
下述两种系统的控制流有什么特点？

- 集中式的？
- 分布式的？

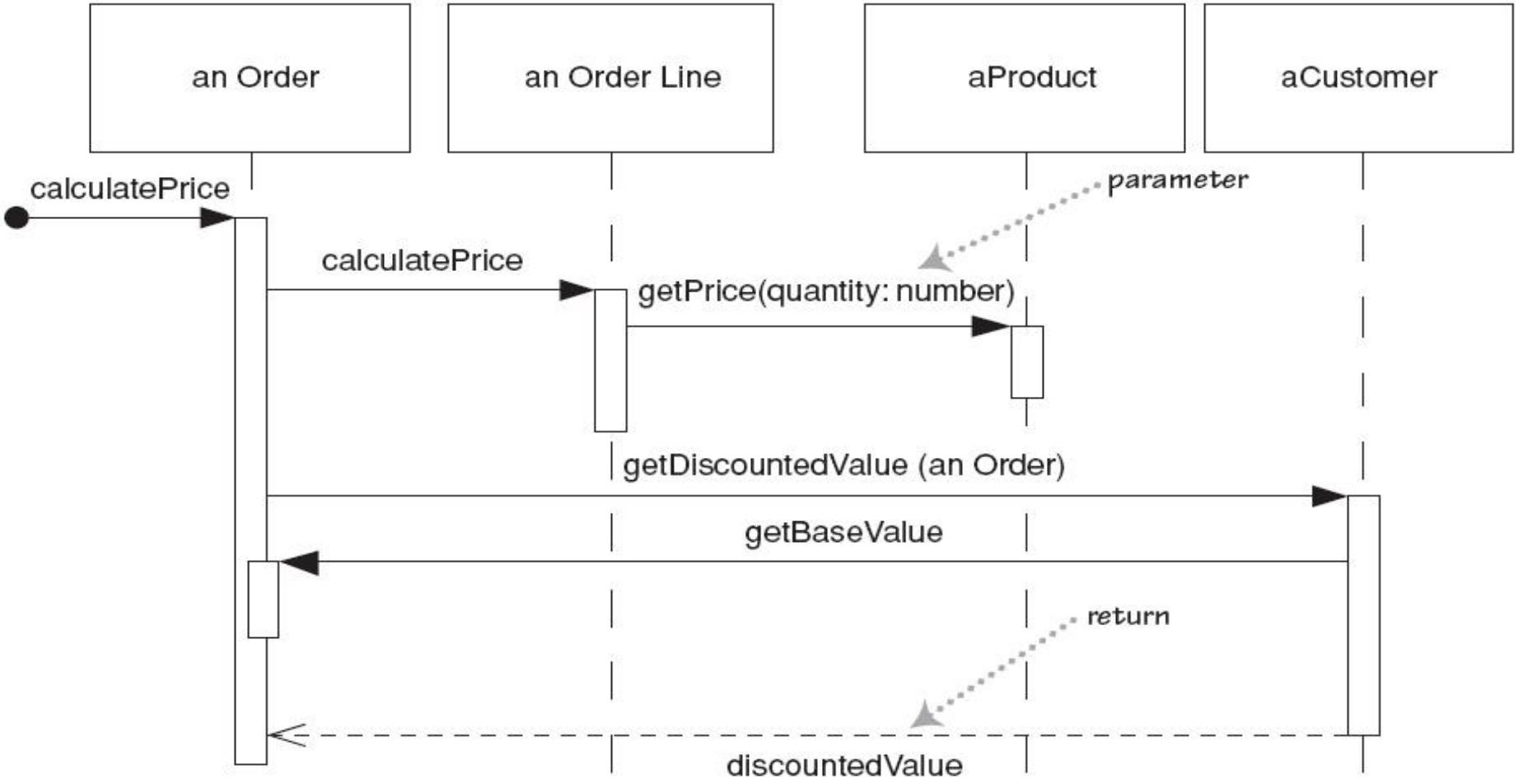




例1：集中控制的计价系统顺序图



例2：分布控制的计价系统顺序图



顺序图与用例的关系(1)

顺序图与用例的关系

- 顺序图表达单个情景实例的行为。
- 每个用例对应一个顺序图。
- 顺序图表达对象间如何协作完成用例所描述的功能。
- 顺序图用于表示为完成用例而在系统边界输入输出的数据以及消息
- 顺序图也用于表示系统内部对象间的消息传递。

顺序图与用例对应

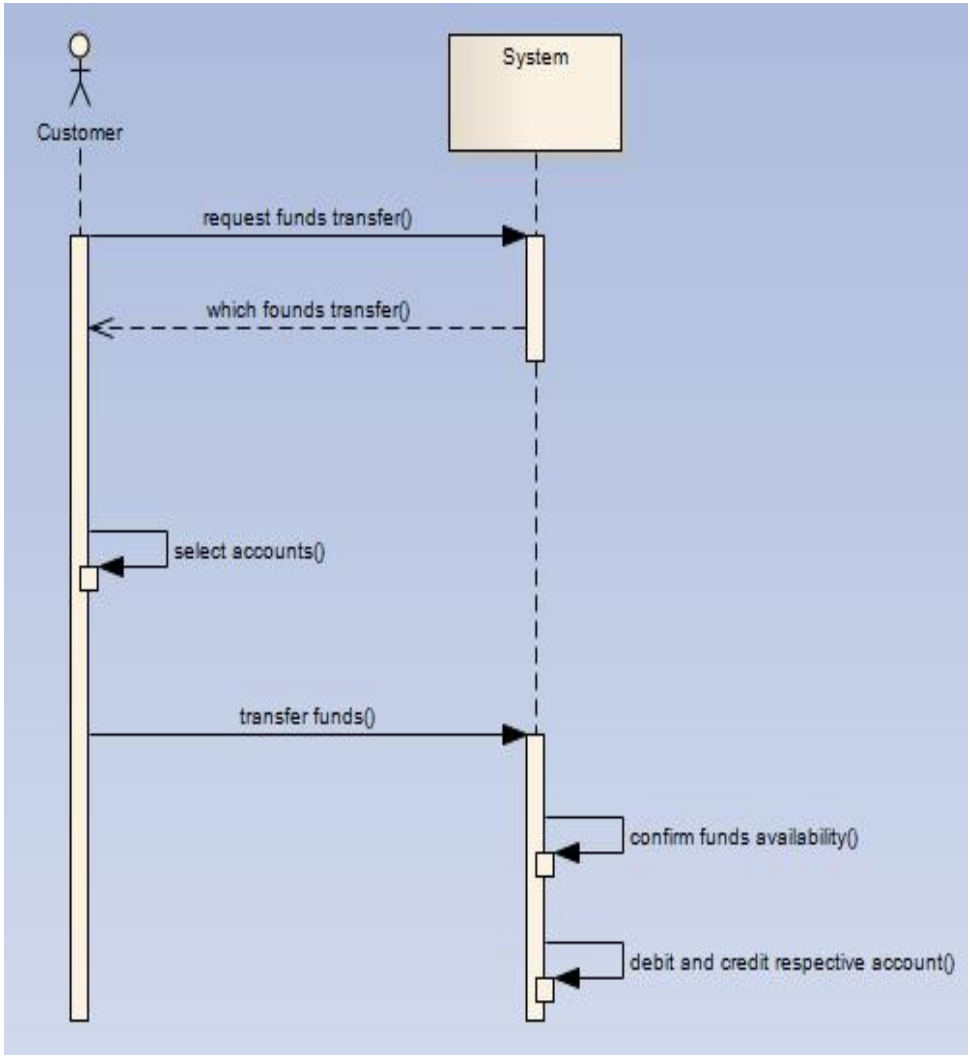
1.The customer requests a funds transfer.

2.The system asks the user to identify the accounts between which funds are to be transferred and the transfer amount.

3.Customer selects the account to transfer funds from, the account to transfer to, and then indicates the amount of funds to transfer.

4.The system checks the account from which funds are to be transferred and confirms that sufficient funds are available.

5.The amount is debited to the account from which funds are to be transferred and credited to the account previously selected by the customer by the system.



顺序图与用例的关系(2)

- 顺序图可帮助分析人员对用例图进行扩展、细化和补遗
- 顺序图可用于开发周期的不同阶段，服务于不同目的，描述不同粒度的行为
- 分析阶段的顺序图不要
  - 包含设计对象
  - 关注消息参数

从用例中抽取顺序图

Basic Course

The Customer specifies an author on the Search Page and then presses the Search button.

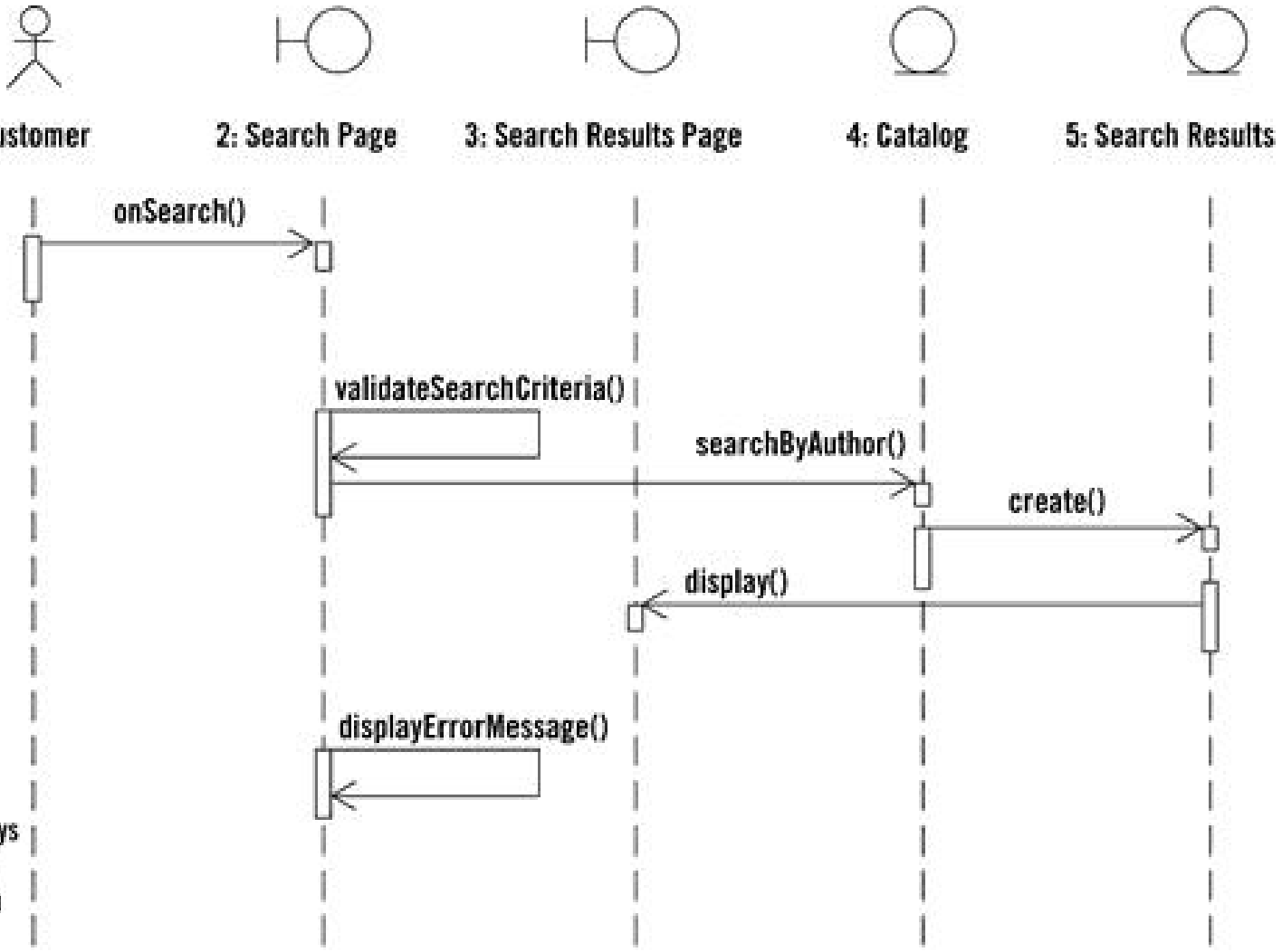
The system validates the Customer's search criteria.

The system searches the Catalog for books associated with the specified author.

When the search is complete, the system displays the search results on the Search Results Page.

Alternate Course

If the Customer did not enter the name of an author before pressing the Search button, the system displays an error message to that effect and prompts the Customer to re-enter an author name.



顺序图建模风格

- 建模风格1：把注意力集中于关键的交互。
  - 创建模型时要把注意力集中于系统的关键方面，而不要包括无关的细节。

    例如：

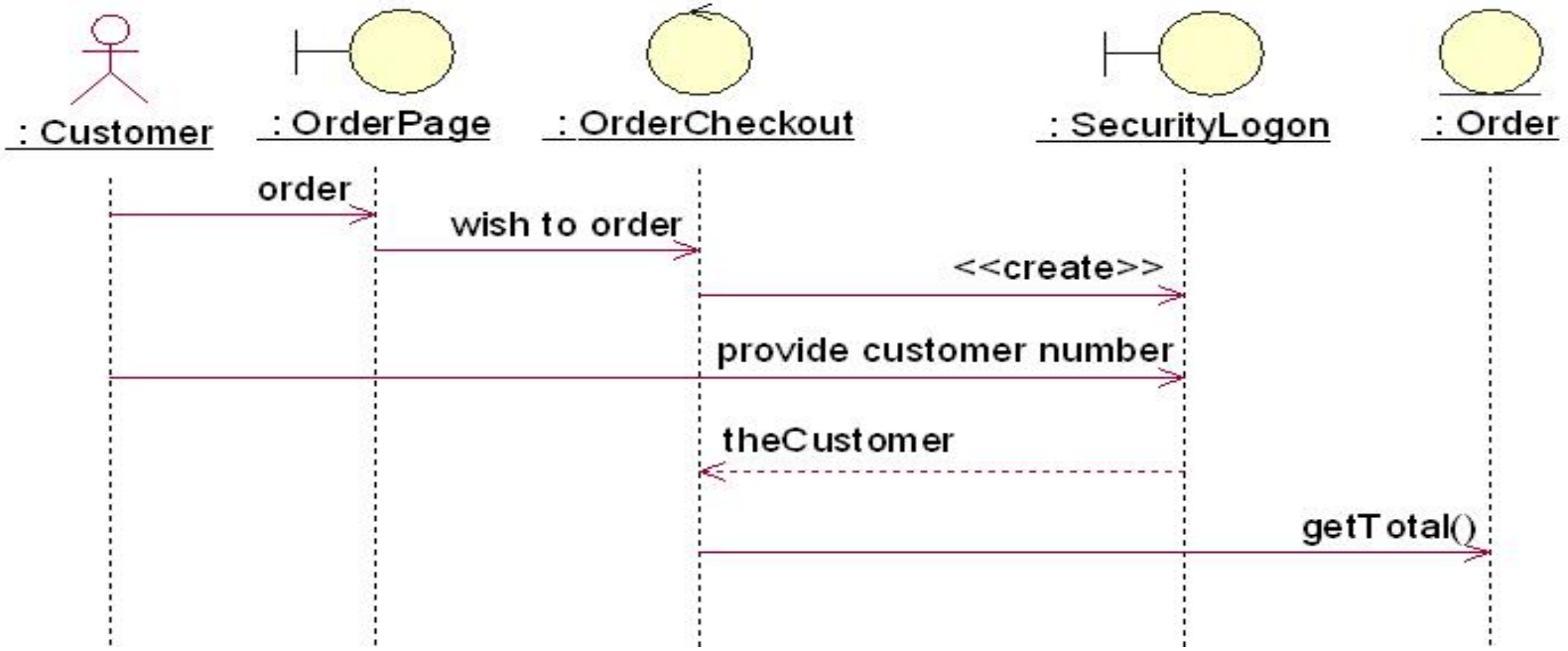
    如果顺序图是用于描述业务逻辑的，就没必要包括对象和数据库之间的详细交互。

顺序图建模风格

- 建模风格2：对于参数，优先考虑使用参数名而不是参数类型。
  - 例如:消息 **addDeposit(amount, target)** 比 **addDeposit(Currency, Account)** 传递了更多的信息
  - 在消息中只使用类型信息不能传递足够的信息
  - 参数的类型信息用UML类图表示更好

顺序图建模风格

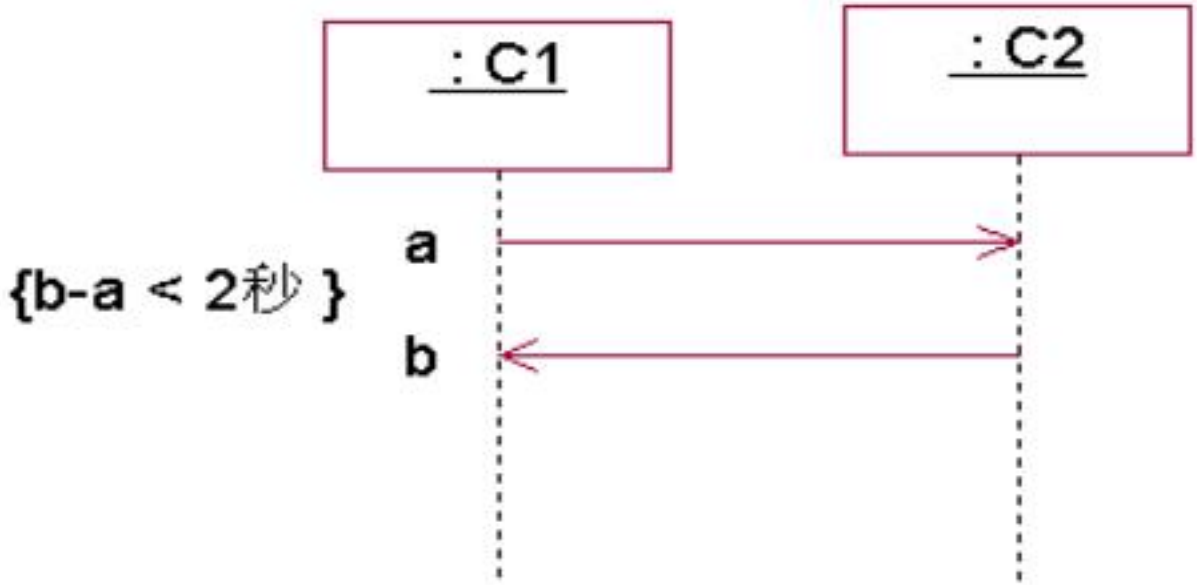
- 建模风格3：不要对明显的返回值建模。



- 建模风格4：可以把返回值建模为方法调用的一部分。

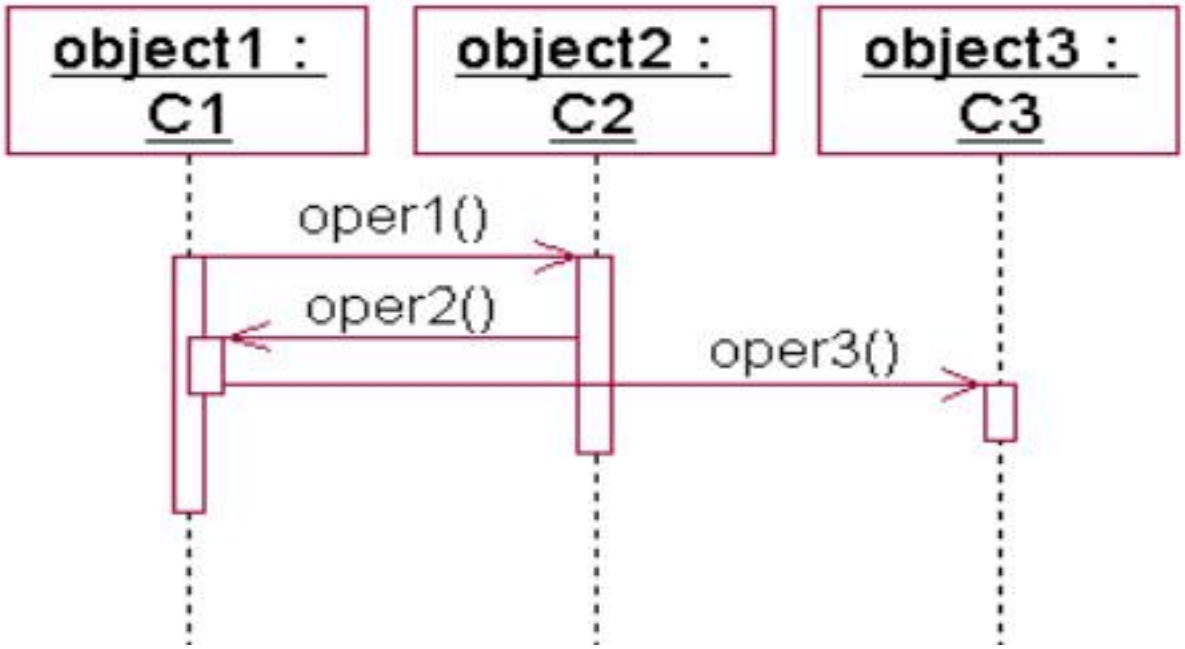
顺序图常见问题分析

- 顺序图中时间约束的表示
  - 用约束（constraint)来表示。



控制焦点（focus of control）的嵌套

- 嵌套的FOC可以更精确地说明消息的开始和结束位置。
- 图例：

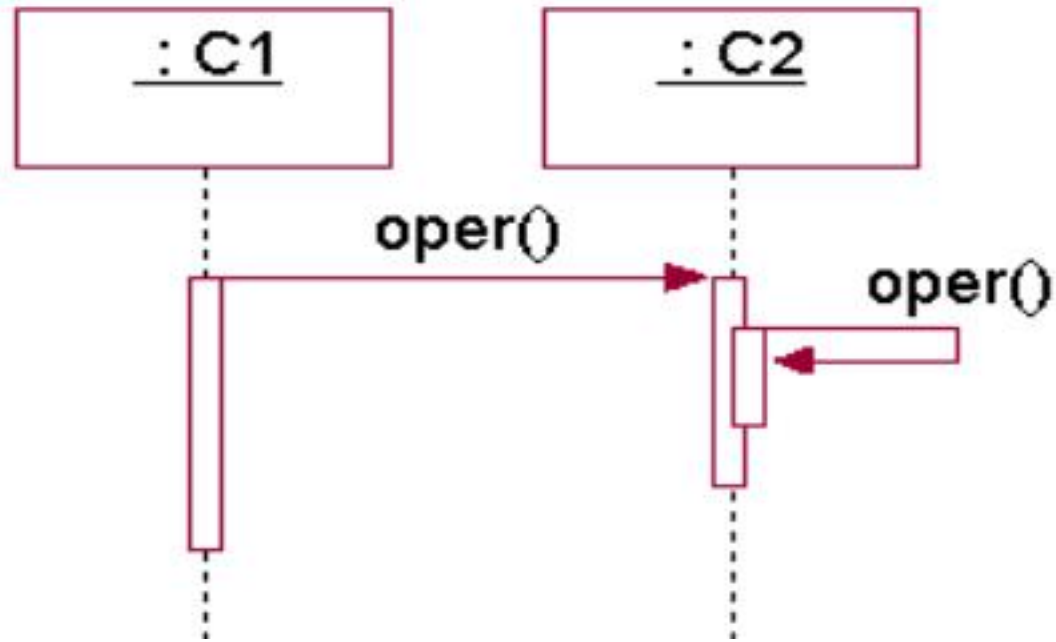


激活期(**activation**)：表示对象执行一个动作的期间（直接操作或者通过下级操作），也即对象激活的时间段。
控制焦点和激活期是同一个概念。

顺序图常见问题分析

- 顺序图中递归的表示
  - 利用嵌套的FOC表示

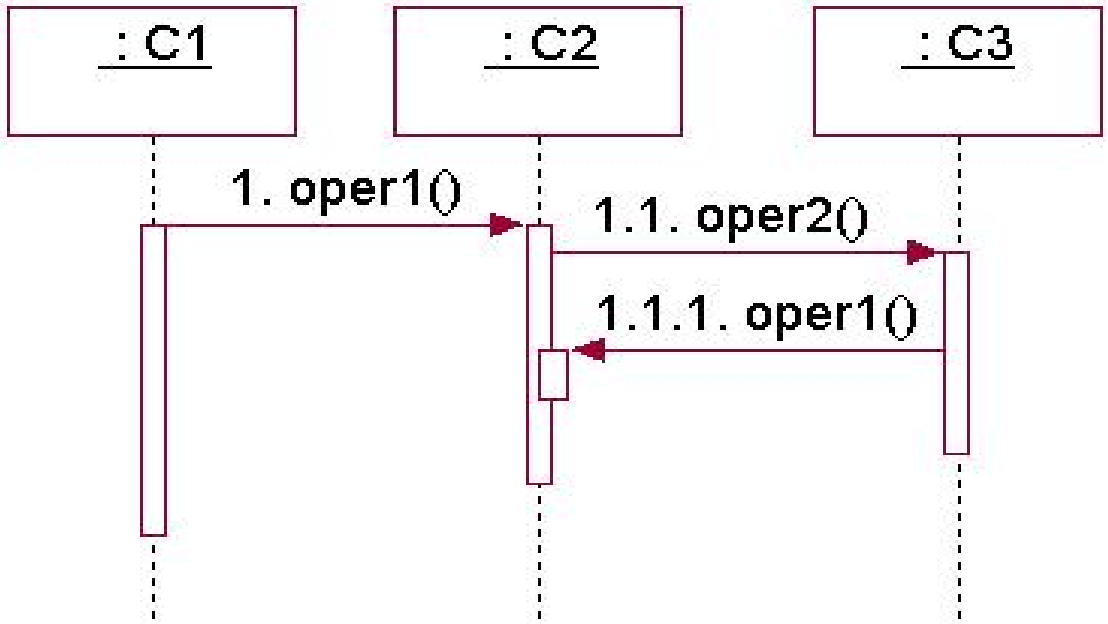
例1. 单个对象自身的递归。



顺序图常见问题分析

- 顺序图中递归的表示
  - 利用嵌套的FOC表示

例2. 多个对象间相互递归调用的表示。

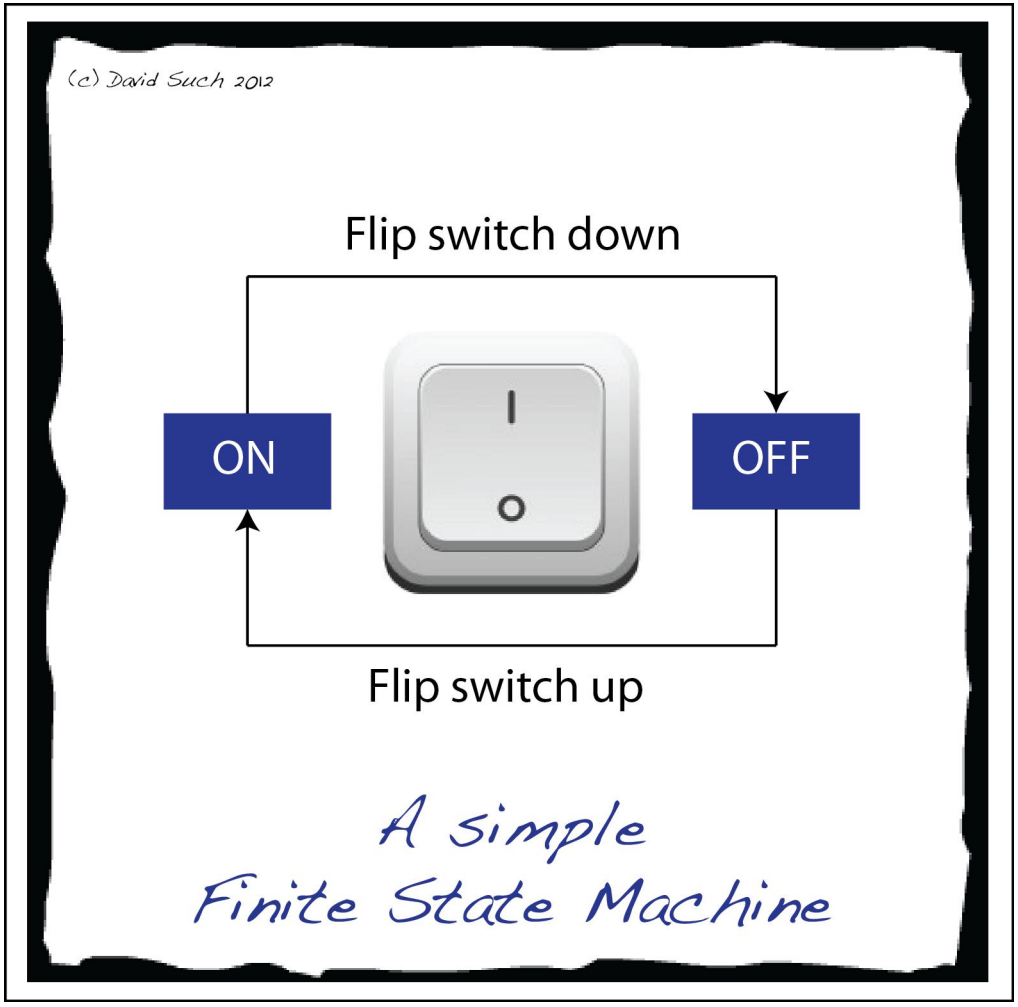


顺序图的作用

- 帮助分析人员对照检查用例中描述的需求是否已经落实给具体对象去实现
- 提醒分析人员去补充遗漏的对象类或操作
- 帮助分析人员识别哪些对象是主动对象
- 通过对一个特定的对象群体的动态行为建模，深入地理解对象之间的交互

状态建模

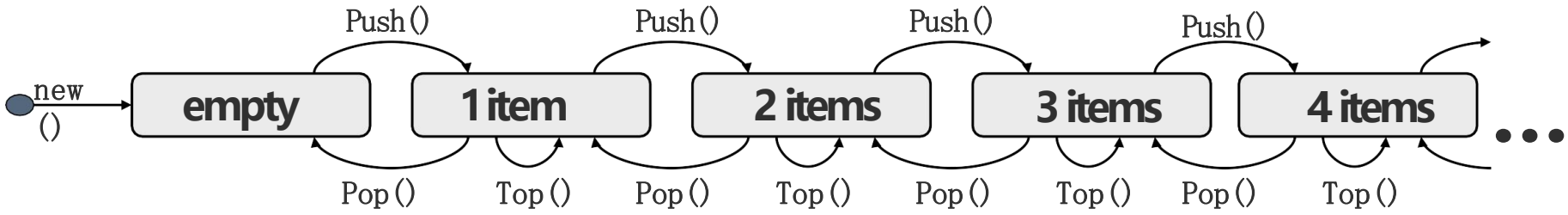
- 什么是状态
  - 一个对象的状态空间
  - 具体状态与抽象状态
- 有限状态机的主要元素
  - 状态和转移
  - 事件和行为
- 模块化的状态机模型: 状态图
  - 组合状态和子状态
  - 绘制状态图的方法



对象及其状态

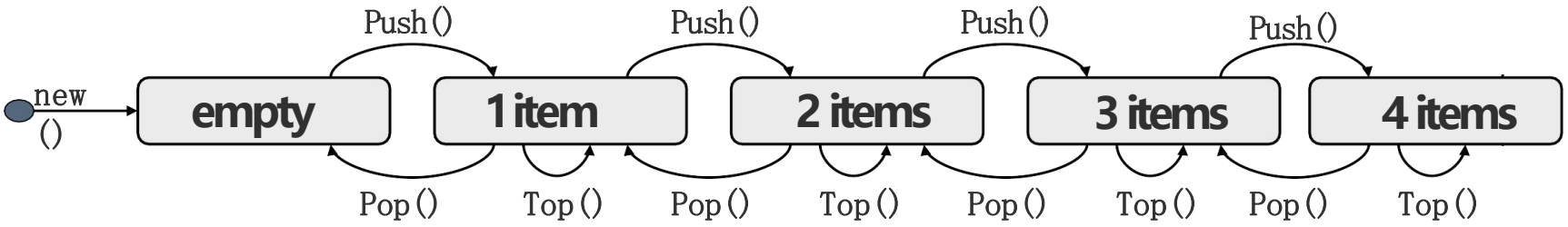
- 所有的对象都有“状态”
  - 对象存在或者不存在
    - 对象不存在也是一种状态
  - 如果对象存在，则具有相应表示其属性的值
  - 每一种状态表示一种可能的状态赋值

- 例如：栈



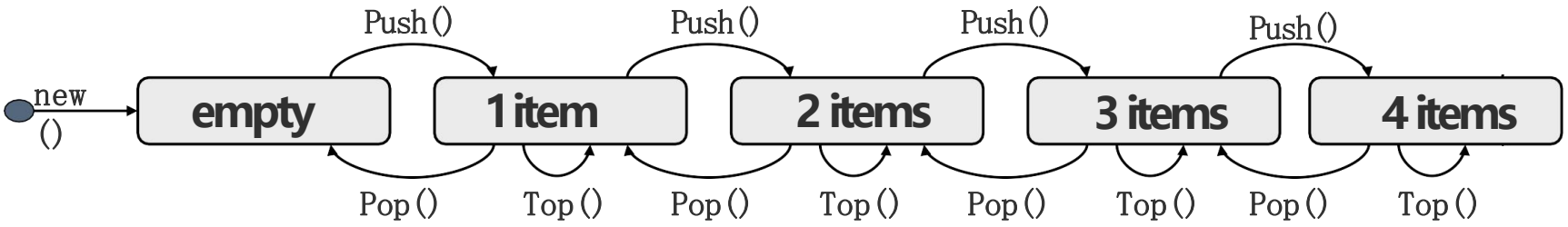
有限状态机

- 有限数量的状态（所有的属性取值为有限的范围）
  - 例如，一个最大容量为4的栈
- 模型可以表示动作序列（状态变化）
  - 例如： new();Push();Push();Top();Pop();Push()…
  - 例如： new();Push();Pop();Push();Pop()…



状态空间

- 对于大部分对象而言，状态空间是非常庞大的
  - 状态空间大小是对象每个属性取值空间的乘积加1
    - 例如. 具有5个布尔值属性的对象有 2^5+1 个状态
    - 例如. 具有5个整数值属性的对象有$(\text{max int})^5+1$个状态
    - 例如. 具有5个实数值属性的对象具有？？个状态
  - 如果忽略计算机表示的局限性，状态空间是无限的



状态的抽象表示

- 但往往状态空间中的局部更有探究的价值
  - 有一些状态是不可能出现的状态
  - 整数或实数值属性往往只在一定范围内取值
  - 通常，我们只关注特定约束下的对象及其行为

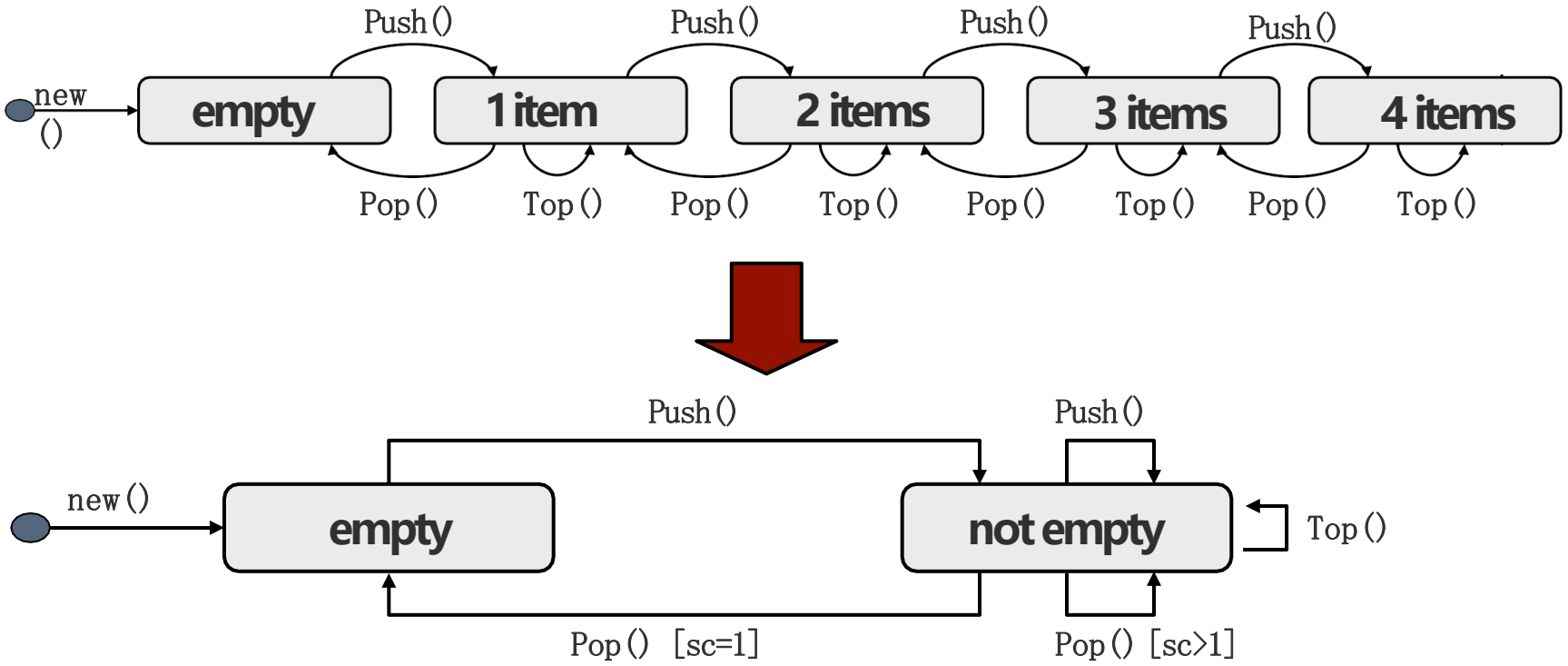
    例如，对于年龄，我们经常选择以下的范围：

    age< 18；18≤age≤65；age> 65

    例如，对于费用信息，我们更关注的约束划分为：

    cost ≤ budget， cost=0， cost >budget， cost >(budget+10%)

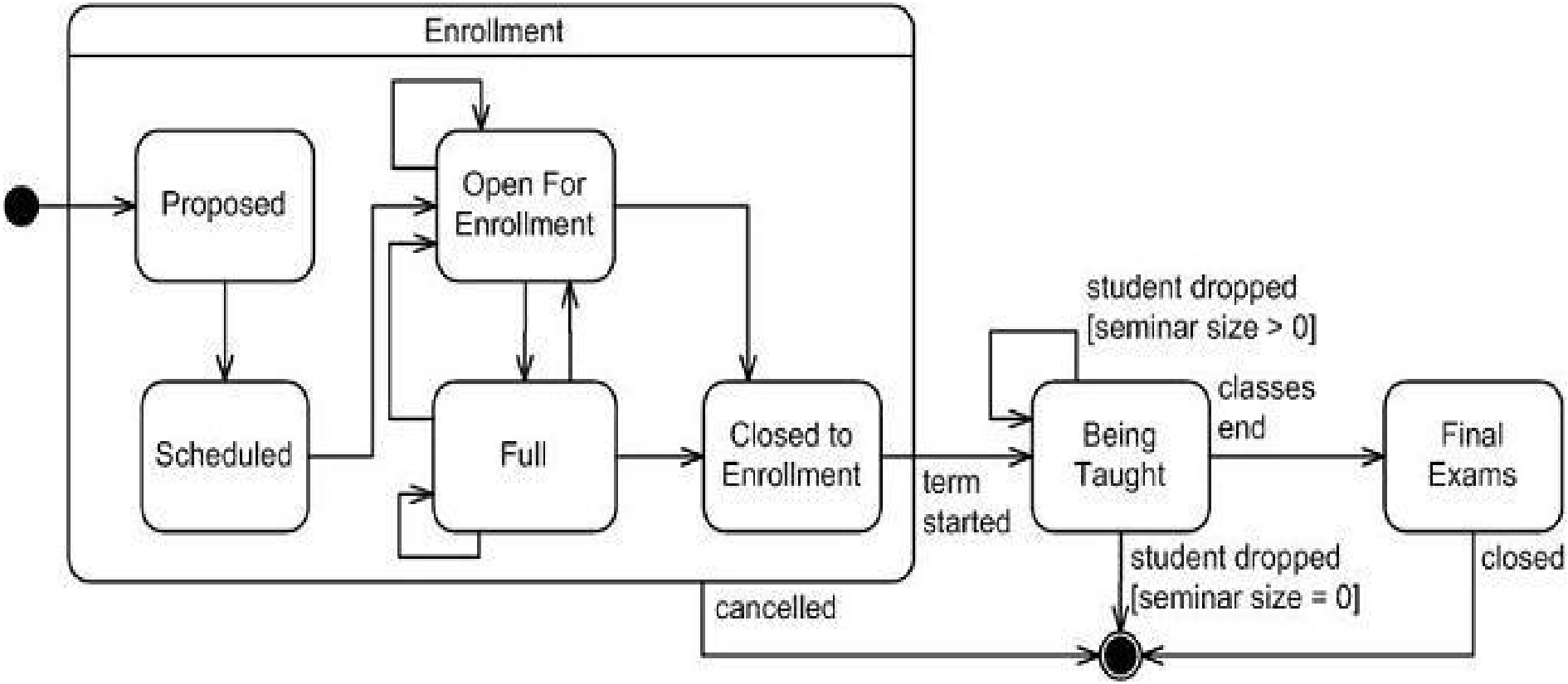
模型建立的过程——状态空间的分解



- 抽象之后的模型可以表达更多的状态序列
  - 例如：上面的模型并不能防止pop()多于push()的序列出现
  - 仍然表达了很多信息

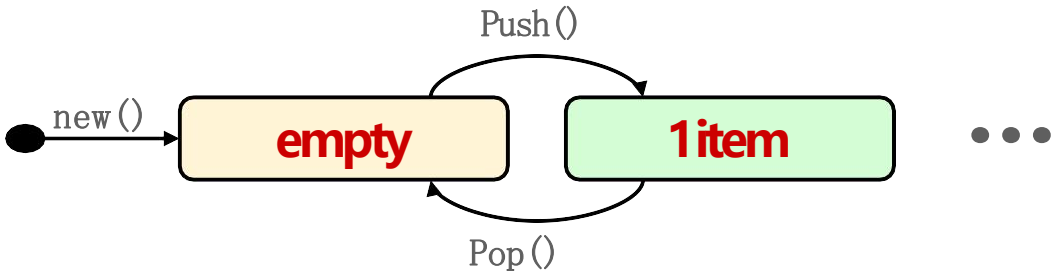
状态图绘制

- 状态图用来表示一个类的全生命周期过程



状态图建模

- 建模元素
  - 状态
  - 事件
  - 状态转移
- 特殊的状态
  - 初始状态、结束状态
  - 组合状态、嵌套状态
  - 历史状态
- 状态图的绘制



状态

定义：

一个对象生命期的一个阶段，该阶段中对象要满足一些特定的条件、执行特定的活动或等待某个（些）事件的发生

- 体现为对象属性的取值
- 包含状态入口或出口、行为描述
- 从不同的抽象层次分析对象，因此其状态是可嵌套（组合）的
- 在给定的场景下，对象状态是确定的，可满足或不满足某个状态

事件

定义：

可以触发对象状态改变的外部刺激，也就是消息的发出与接收

- 决定状态迁移何时发生

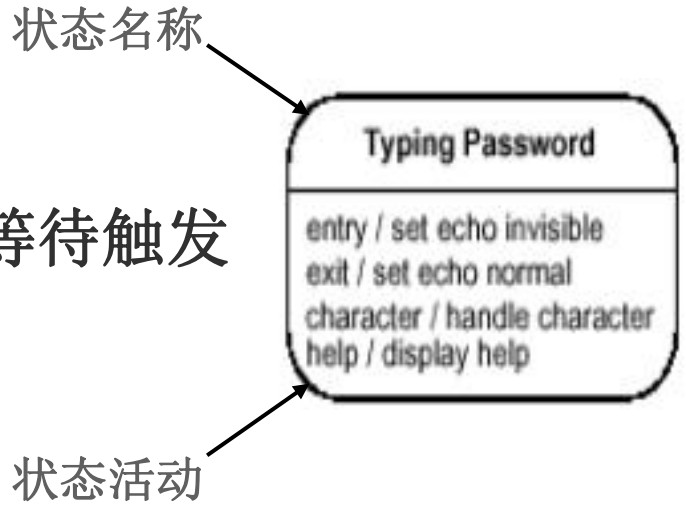
状态迁移

定义：是状态之间的关系，当发生一个事件，条件满足时就会发生从源状态(Source State)到目标状态(Object State)的转变

- 当且仅当迁移条件满足时才能触发状态迁移
- 每个状态迁移都对应一个触发“事件”
- 同时还需要满足一定的“警戒条件(Guard Condition)”
- 当触发事件发生，或相关警戒条件满足时，进行相应的状态迁移
- 状态迁移的过程会伴随相关的对象操作

UML状态图中的状态（State）

- 一个状态表示在某个时间段内
  - 某个陈述是正确的
    - 例如. (budget -expenses) > 0
  - 某个动作正在执行或者在某个时间等待触发
    - 例如. 检查订单商品的存货
    - 例如. 等待缺货产品到货



- 状态相关的活动类型
  - do/activity
    - 只要处于这个状态，某个活动就会一直执行，直到离开这个状态
  - entry/action and exit/action
    - 当进入（/离开）某个状态时执行的动作
  - include/stateDiagramName
    - 调用另一个状态图，形成嵌套的状态图

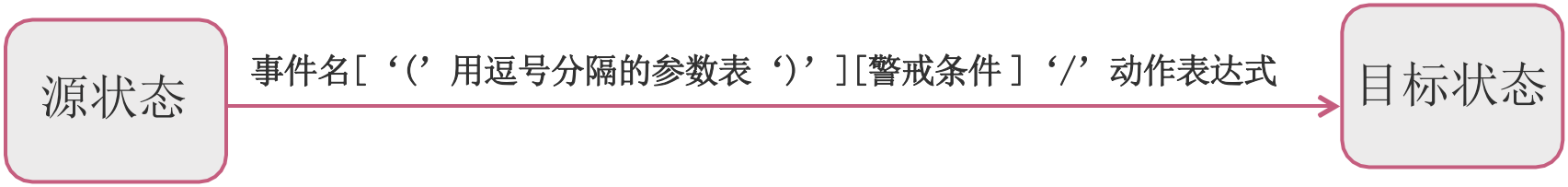
初始状态

结束状态

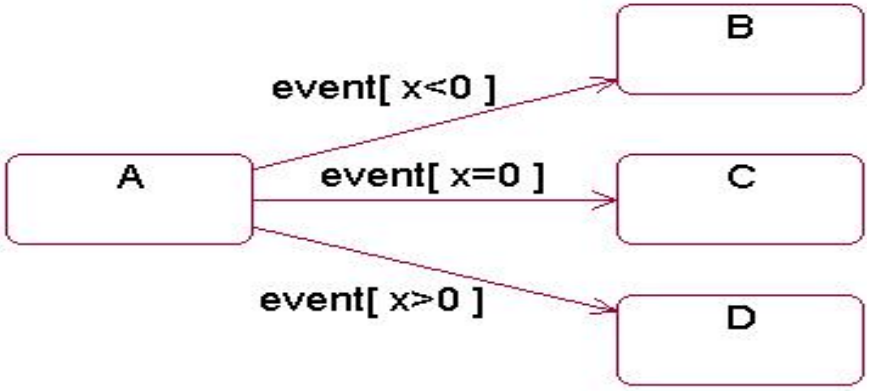
UML状态图中的迁移（Transitions）

迁移包括五部分：

- 源状态(source state)、触发事件 (event trigger)，警戒条件(guard condition)， 动作(action)，目标状态(target state).



- 对于给定的状态，最终只能产生一个迁移，因此从相同的状态出来的、事件相同的几个迁移之间的条件应该是互斥的。



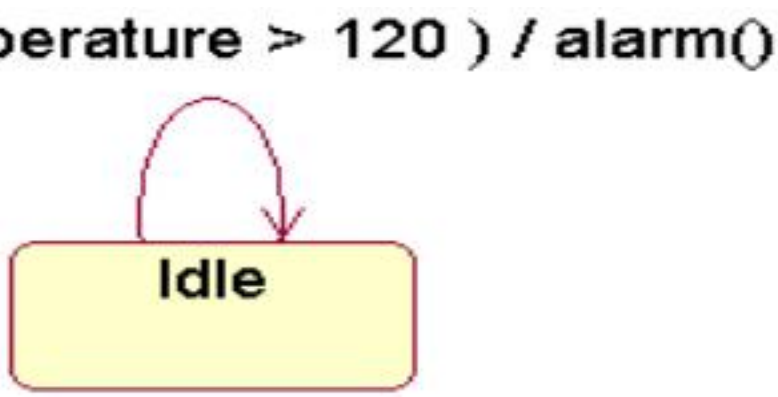
UML状态图中的事件（Events）

- 事件(Events)的意义在于系统需要了解正在发生什么
  - 状态图中，事件仅需和系统或当前建模的对象相关
  - 从系统角度出发，事件必须建模成一个瞬间可完成的动作
    - 例如. 完成工作，考试未通过，系统崩溃
  - 在OOD(面向对象设计)中通过传递消息的方式实现事件
- 在UML中，有四种类型的事件
  - 变更事件(Change events) 当给定条件成立时就会发生变更事件
  - 调用事件(Call events) 当给定对象的操作被调用执行时会发生调用事件
  - 时间事件(Elapsed-time events) 表明时间段过去，或某个特殊时间点的触发
  - 信号事件(Signal events) 当给定对象收到某实时信号

UML状态图中的事件（Event）--变更（Change）事件

- **变更事件(Change event)**：通过布尔表达式中变量的改变，使得表达式成立的事件，通过“when”关键字进行提示。

  例：



- 变更事件和警戒条件(guard condition)的区别：
  - 警戒条件只在所相关的事件出现后计算一次，如果值为false，则不进行状态转移。

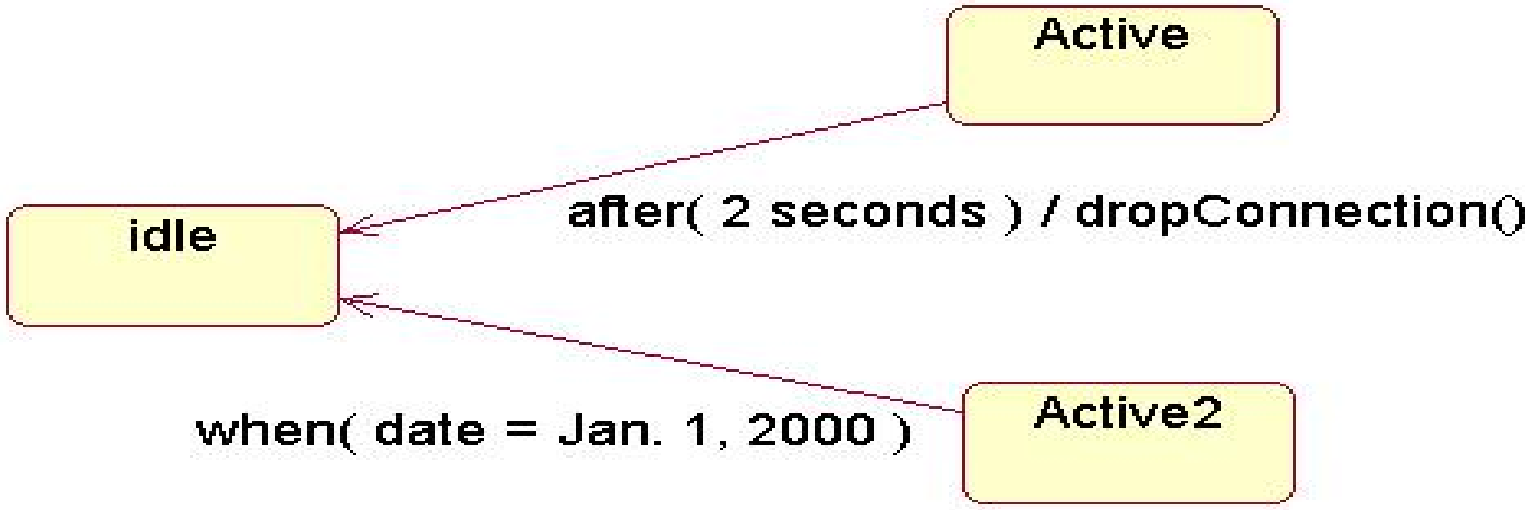
UML状态图中的事件（Event）--调用（Call）事件

- **调用事件(Call event)**：在这一类事件中，状态迁移的动作会调用对象的方法
- 语法格式如下：　**事件名（[逗号分隔的参数列表]）**
  - 其中参数列表中的参数格式为：　**参数名：类型**

例：

UML状态图中的事件（Event）--时间（Time）事件

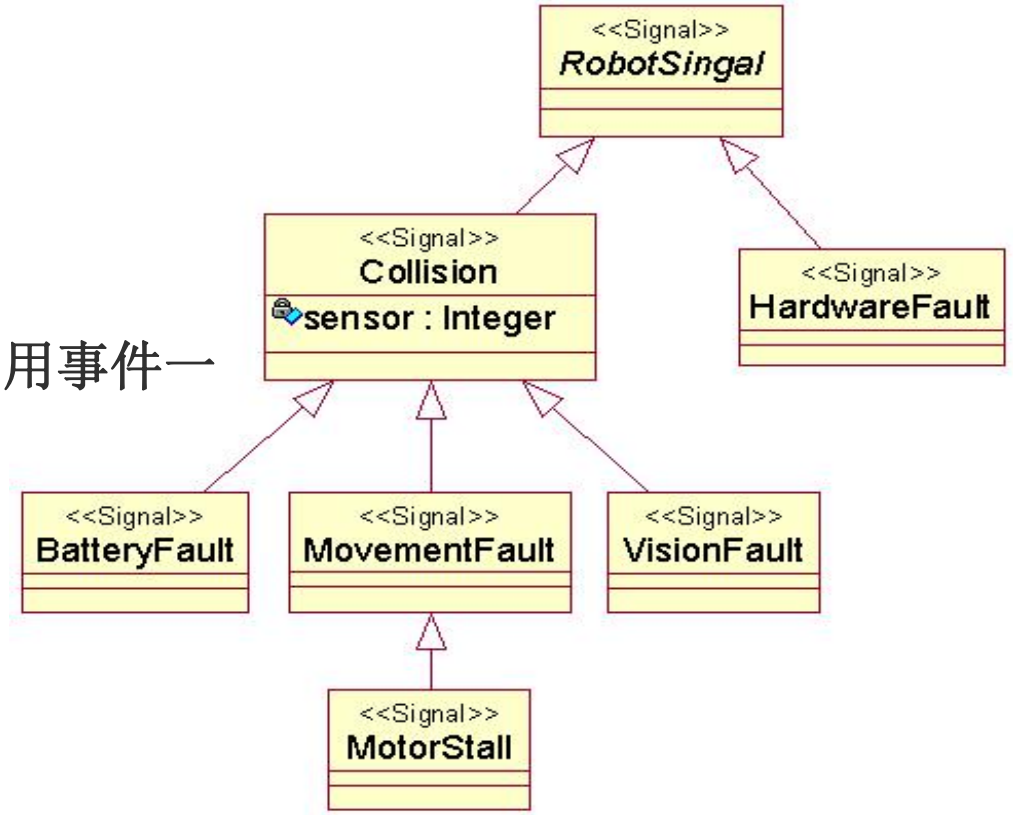
- **时间事件(Time event)**：通过时间表达式是否满足来表示事件，例如一个绝对时间点的到来，或者经过时间段过去后对象进入一个新状态。
- 用关键字after或when表示。例：



UML状态图中的事件（Event）--信号（Signal）事件

- **信号事件(Signal event)**：表示接受一个对象发送的信号（信息）的事件，有可能引发状态迁移（状态改变）
- 语法格式如下：

  事件名 ([逗号分隔的参数列表])
- 信号事件与调用事件的区别：
  - 信号事件是一个异步事件，调用事件一般是一个同步事件。



UML状态图中的动作（Action）

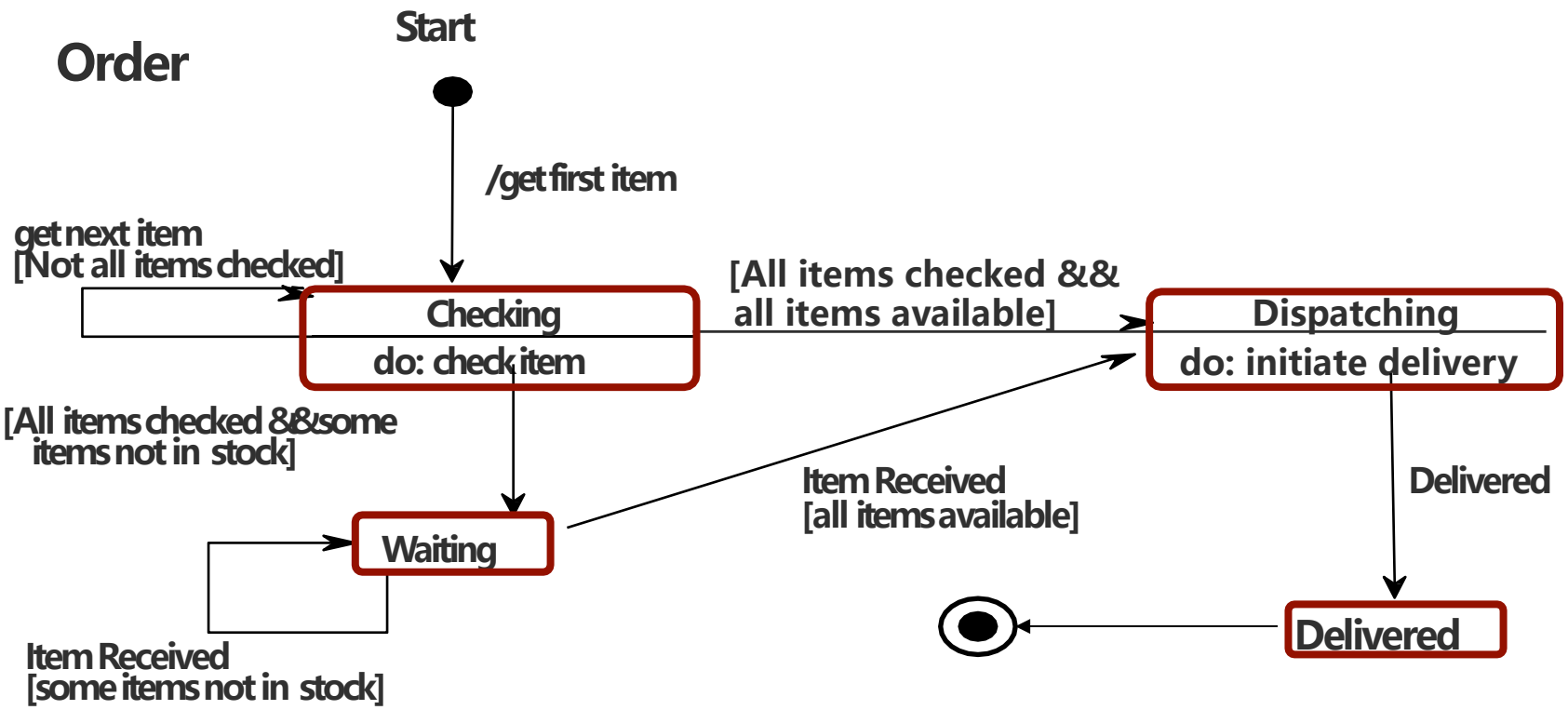
- 动作是在状态内部或者状态间迁移时执行的原子操作
- 两种特殊的动作:入口动作(entry action)和出口动作(exit action)
  - Entry动作：进入状态时执行的活动，格式如下：

    ‘entry’ ‘/’ action-expression
  - Exit动作：退出状态时执行的活动，格式如下：

    ‘exit’ ‘/’ action-expression

  (其中 action-expression 可以引用对象本身的属性和输入事件的参数)

例：订单处理

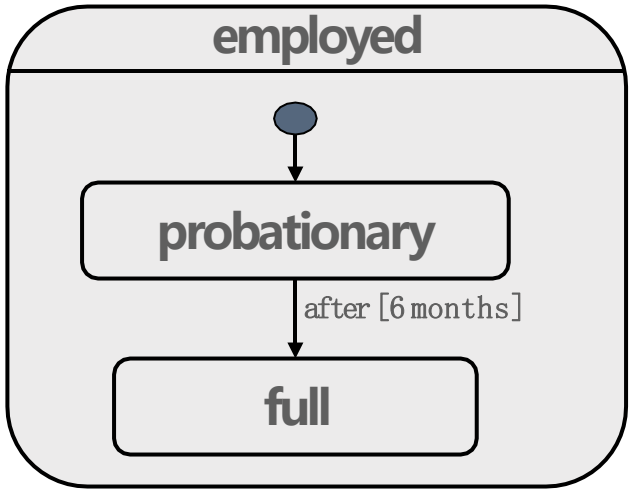

Order
Start
/get first item
get next item
[Not all items checked]
Checking
do: check item
[All items checked &&
all items available]
Dispatching
do: initiate delivery
[All items checked &&some
items not in stock]
Item Received
[all items available]
Delivered
Waiting
Item Received
[some items not in stock]
Delivered

UML状态图中的组合状态（SuperStates）

- 可以通过状态嵌套的方式简化图表
  - 一个组合状态可以包含一个或多个状态
  - 组合状态可以实现从不同抽象层次去体现状态图

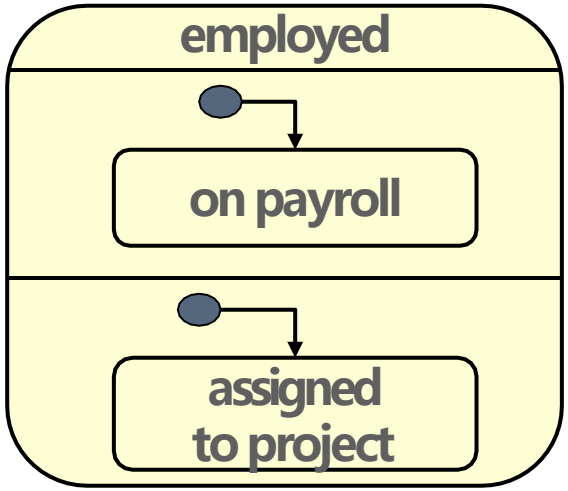
“OR”的组合状态

- 处于组合状态时只能满足其中一个子状态

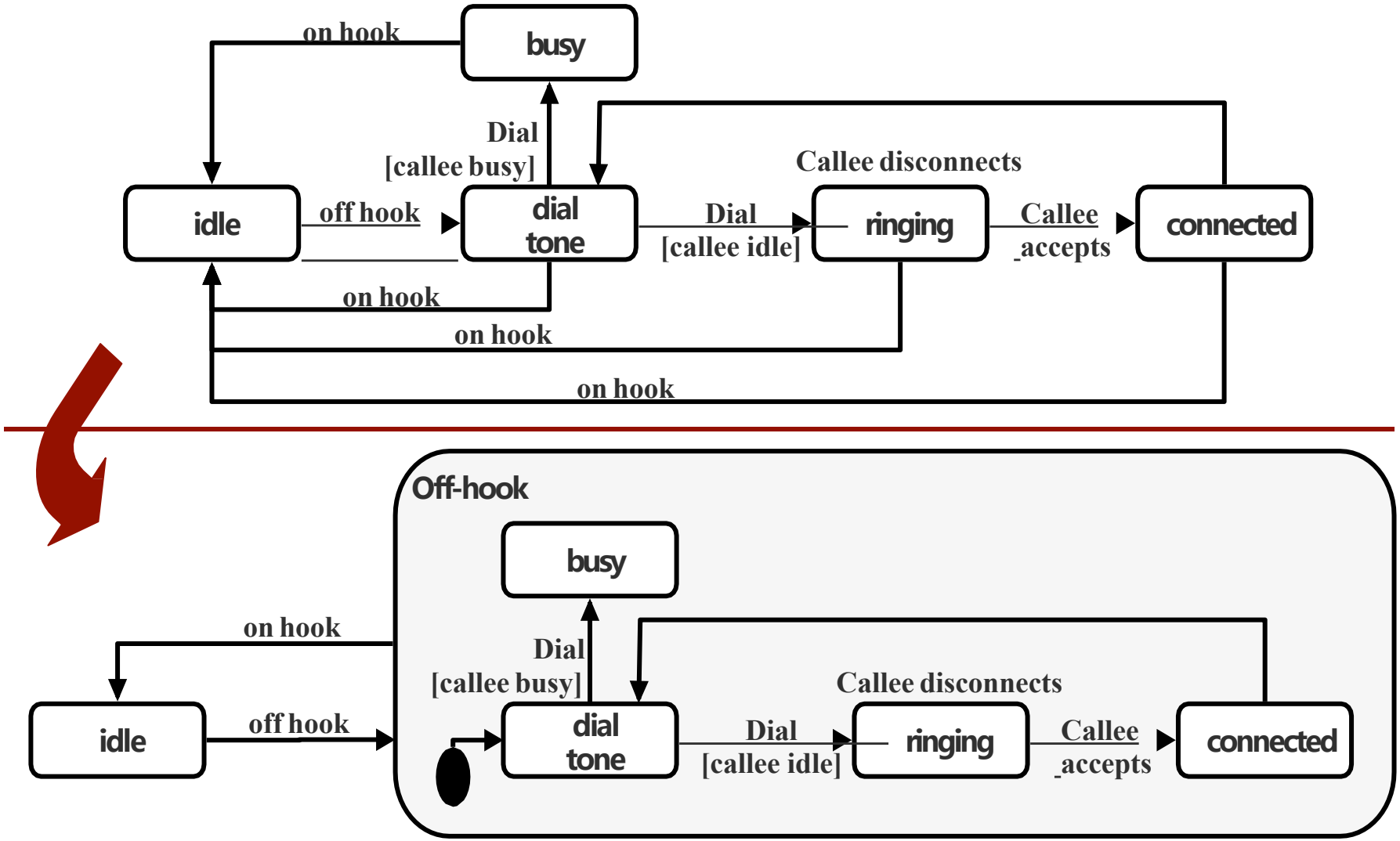


“AND”的组合状态(并发状态)

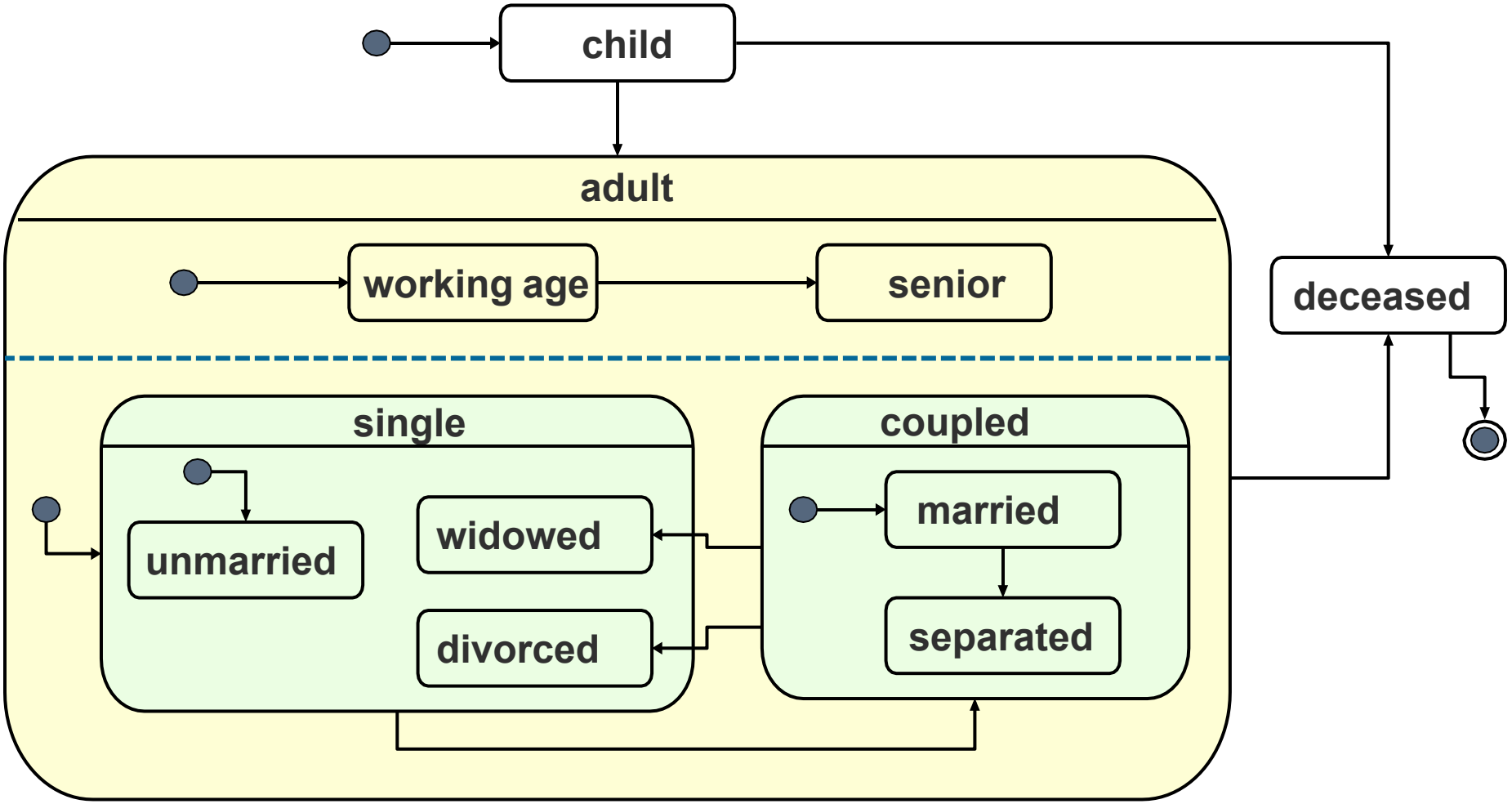
- 处于组合状态时，满足所有的子状态
- 通常，AND的子状态会进一步嵌套为OR的子状态



组合状态的例子

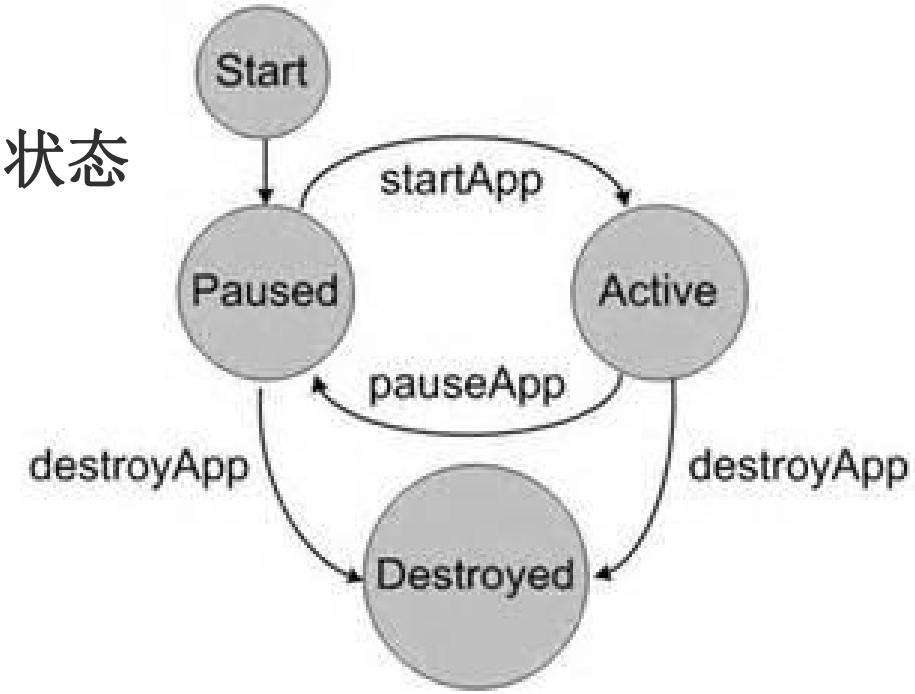

busy
on hook
Dial
[callee busy]
Callee disconnects
idle
off hook
dial
tone
Dial
[callee idle]
ringing
Callee
accepts
connected
on hook
on hook
on hook
Off-hook
busy
on hook
Dial
[callee busy]
Callee disconnects
idle
off hook
dial
tone
Dial
[callee idle]
ringing
Callee
accepts
connected

组合状态的例子

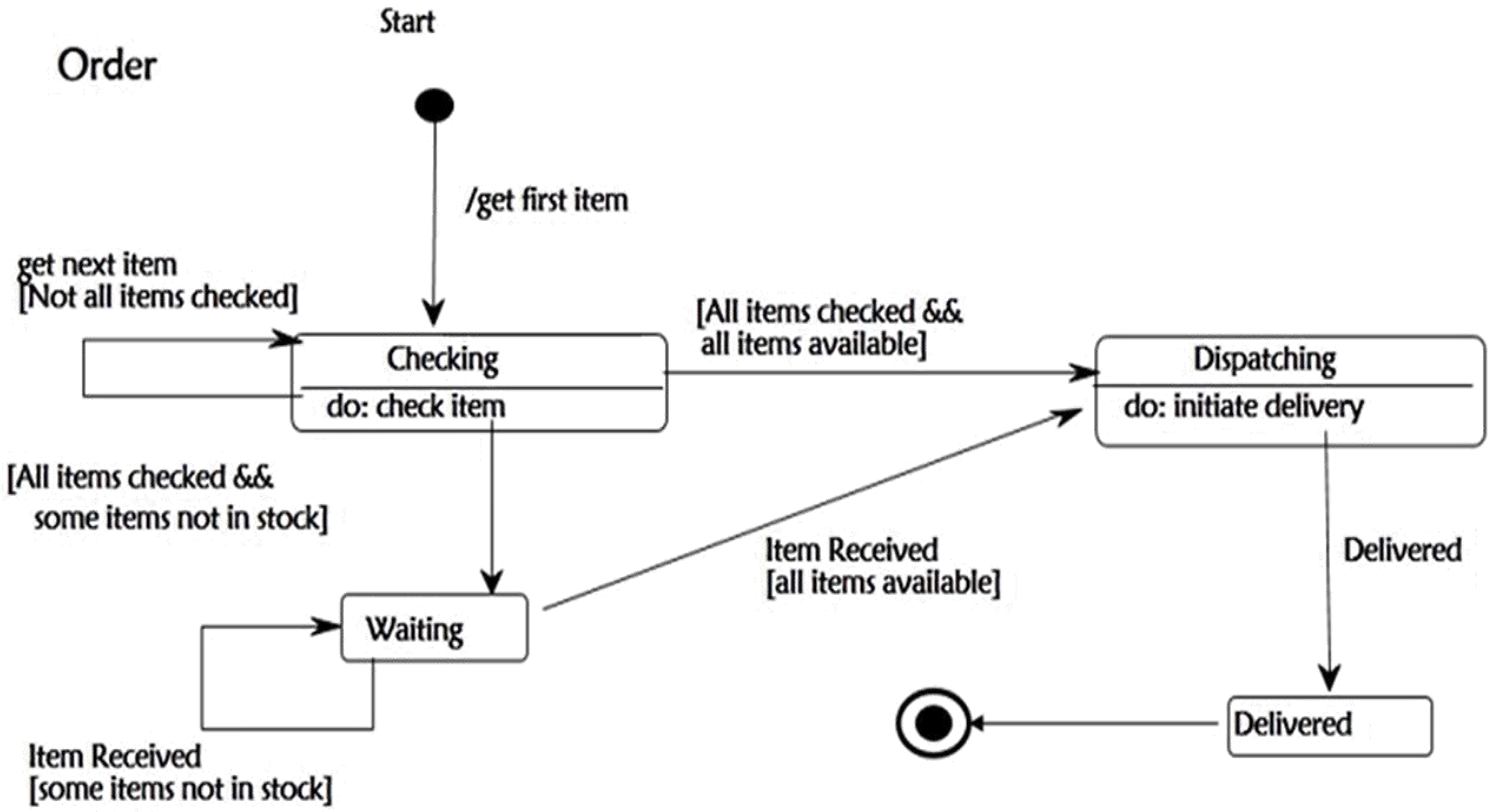

child
adult
working age
senior
deceased
single
unmarried
widowed
divorced
coupled
married
separated

组合状态的状态迁移

- 指向组合状态边界的状态迁移等价于指向该组合状态初态的迁移
  - 所有属于该组合状态的入口条件将被执行
- 从组合状态边界转出的迁移等价于从该组合状态的终态发出迁移
  - 所有出口条件均将被执行
- 迁移可直接指向组合状态的子状态

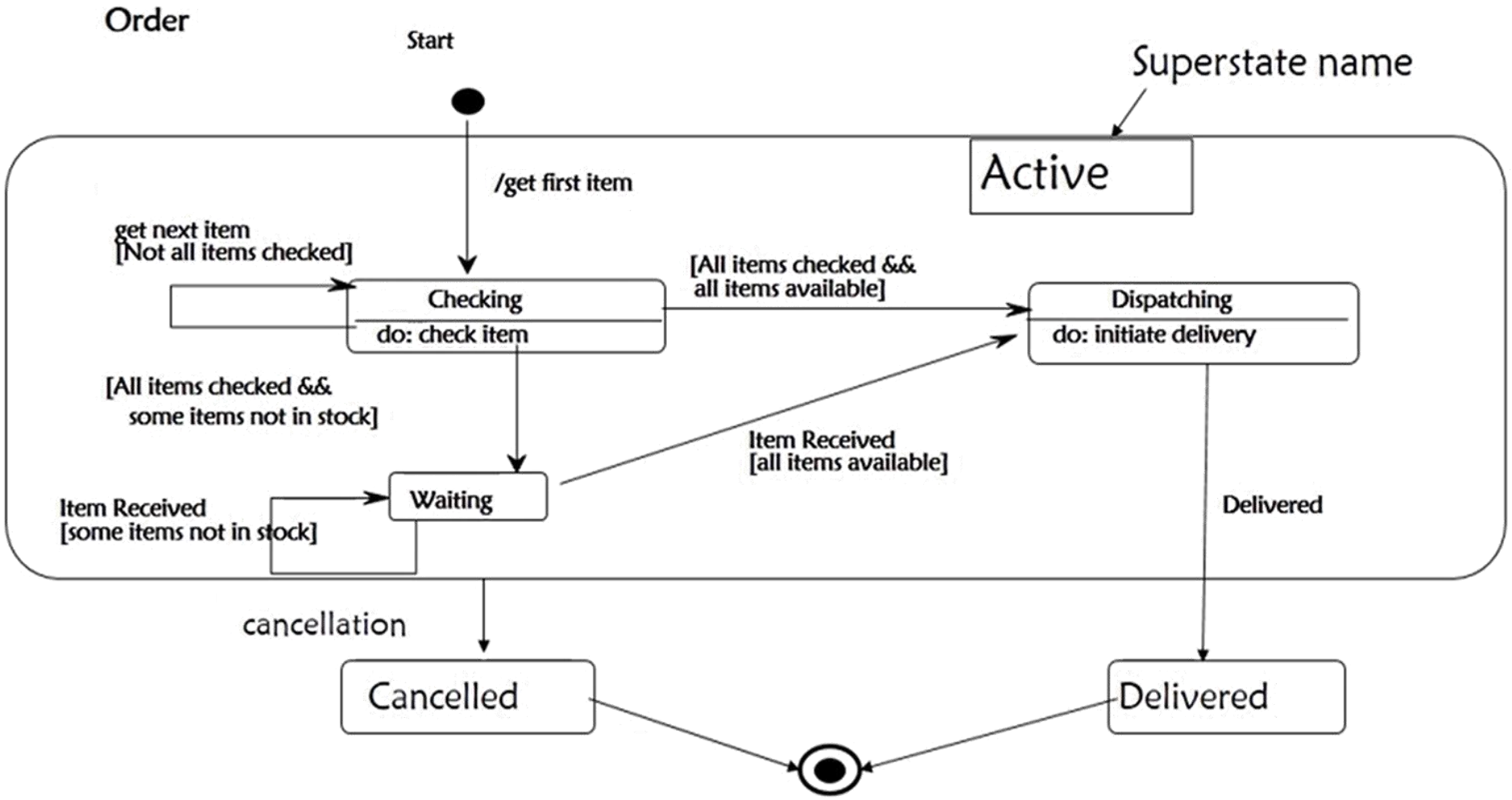


练习：组合状态



练习：在图中增加一个新的状态和相关的状态迁移，表示在物品投递之前的任何环节都可以取消订单

参考答案

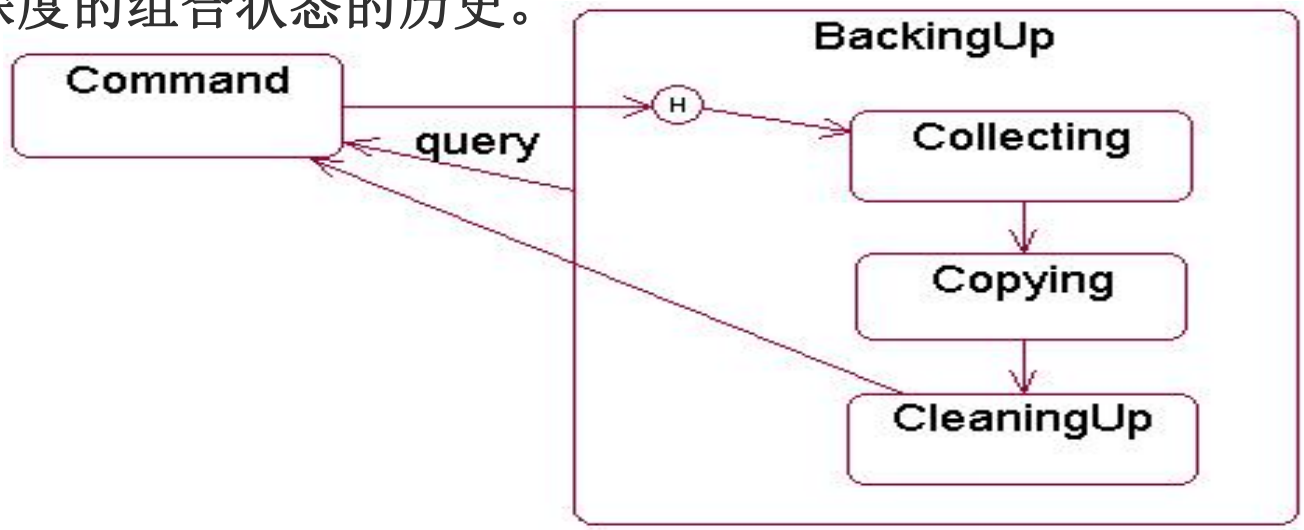

Order
Start
Superstate name
Active
/get first item
get next item
[Not all items checked]
Checking
do: check item
[All items checked &&
all items available]
Dispatching
do: initiate delivery
[All items checked &&
some items not in stock]
Item Received
[all items available]
Item Received
[some items not in stock]
Waiting
Delivered
cancellation
Cancelled
Delivered

UML状态图中的历史状态（History State）

- 历史状态是一种伪状态。当激活这个状态时，会保存从组合状态中退出时所处的子状态，用H表示
- 当再次进入组合状态时，可直接进入到这个子状态，而不是再次从组合状态的初态开始。

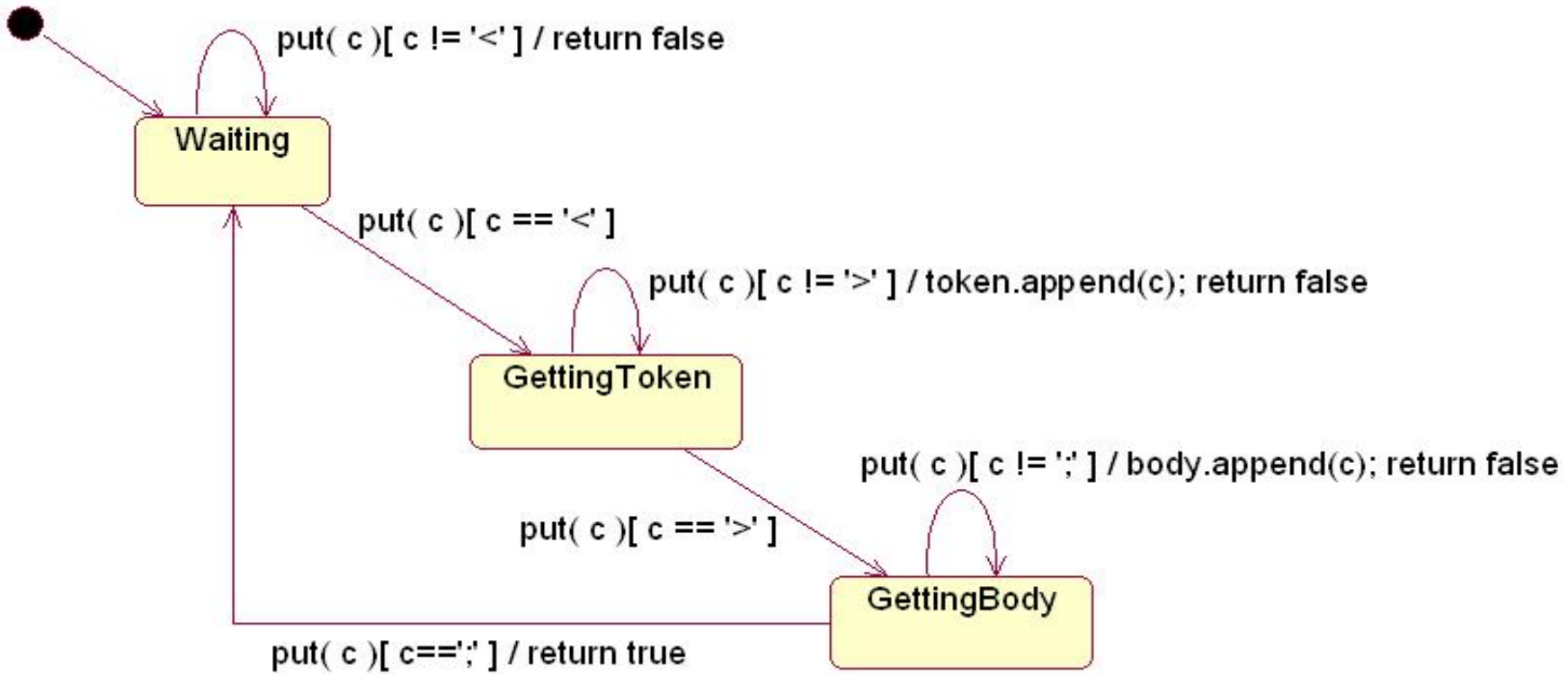
  - H和H*的区别：
    - H只记住最外层的组合状态的历史。
    - H*可记住任何深度的组合状态的历史。

例：



状态图的工具支持

正向工程：根据状态图生成代码。例：

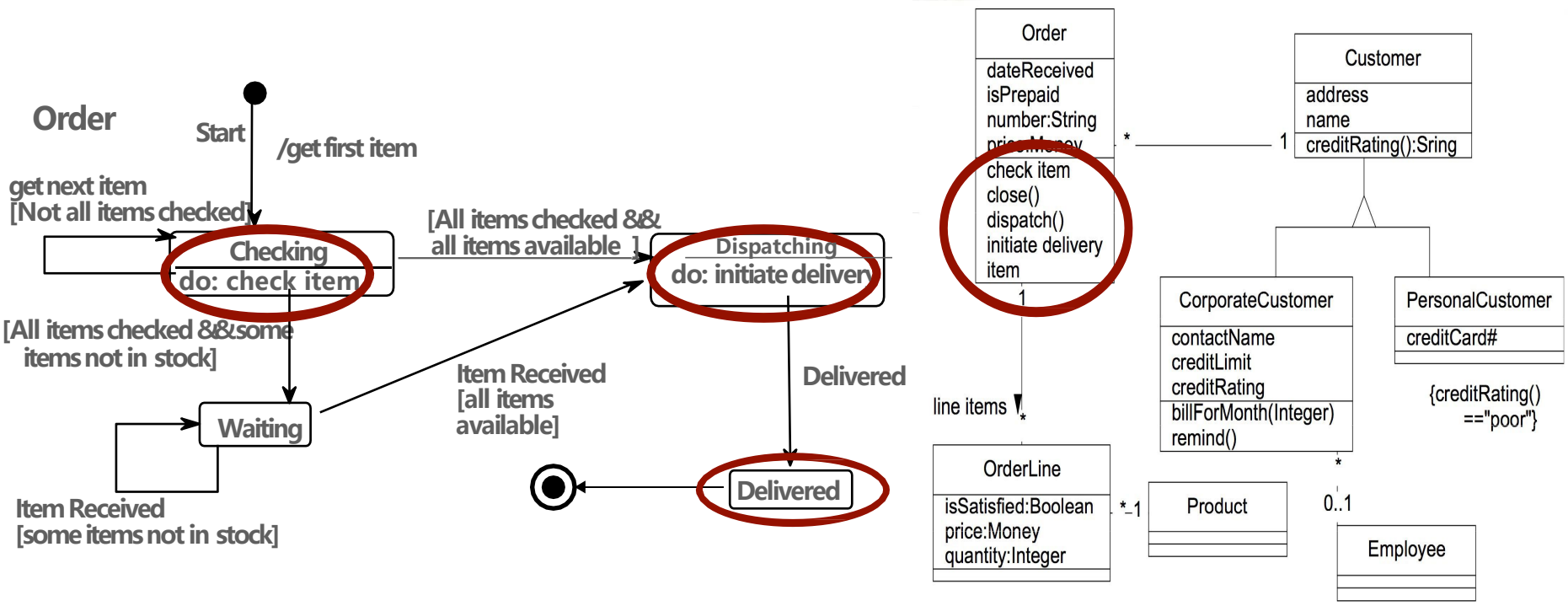


状态图的工具支持

```
class MessageParser {
public boolean put(char c) {
    switch (state) {
        case Waiting:
            if (c == '<') {
                state = GettingToken;
                token = new StringBuffer();
                body = new StringBuffer();
            }
            break;
        case GettingToken :
             if (c == '>') state = GettingBody;
             else token.append(c);
             break;
        case GettingBody :
            if (c == ';') {
                state = Waiting;
                 return true;
            }
            else
                body.append(c);
        }
        return false;
    }
public StringBuffer getToken() {
     return token;
}
public StringBuffer getBody() {
    return body;
}
private final static int Waiting = 0;
private final static int GettingToken = 1;
private final static int GettingBody = 2;
private int state = Waiting;
private StringBuffer token, body;
}
```

状态图与其他UML图的关系

- 状态图中的事件为顺序图/交互图中该对象的输入消息
- 状态图应针对类图中具有重要行为的类进行建模
- 每个事件、动作对应于相应类中的一个具体操作
- 状态图中每个输出消息对应于其他类的一个操作
- 状态图中的操作定义等价于类图中的操作定义



状态图建模风格

- 建模风格1：把初态放置在左上角；把终态放置在右下角
- 建模风格2：用过去式命名转移事件
- 建模风格3：警戒条件不要重叠
- 建模风格4：不要把警戒条件置于初始转移上

状态图的检查表

- 一致性检查
  - 状态图中所有的事件应该是
    - 类图中本对象类的方法
  - 状态图中所有的动作应该是
    - 类图中其他对象类的方法
- 绘图风格
  - 每个状态的命名应该是唯一的，意义明确的
  - 只对行为复杂的状态使用组合状态建模
  - 不要在一个图中包含太多细节
  - 使用警戒条件时要特别注意不要引入二义性
  - 状态图应该具有确定性（除非特殊原因）

下述情况不适宜使用状态图：

- 当大部分的状态转移为“当这个状态完成时”
- 有很多来自对象自身发出的触发事件
- 状态代表的信息与类中的属性赋值并不一致