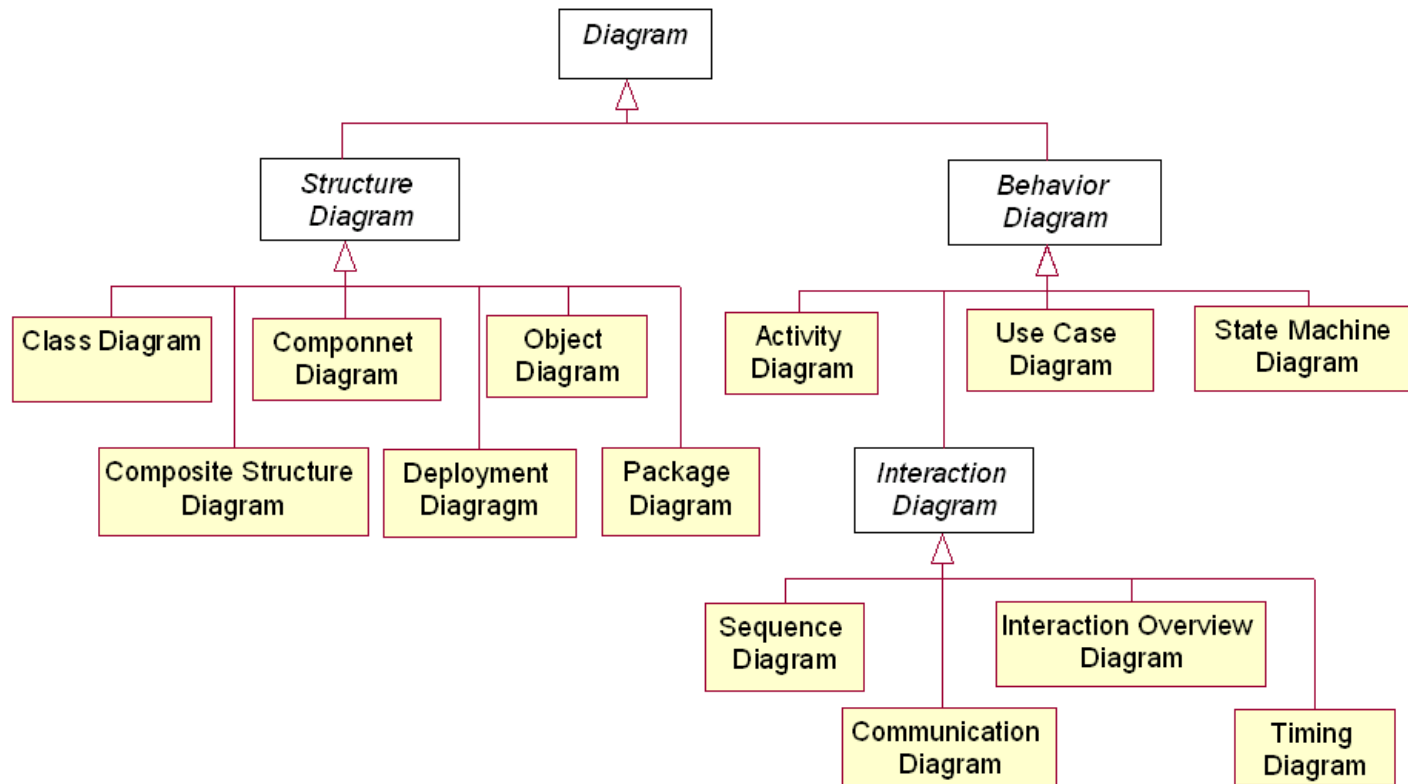# CS10102302
UML行为建模方法

赵君峤 副教授

计算机科学与技术系 电子与信息工程学院

同济大学

Classification of Diagrams in the UML 2.0

Modelling "State"

- **What is State?**
  - State space for an object
  - concrete vs. abstract states

- **Finite State Machines**
  - **states and transitions**
  - events and actions

- **Modularized State machine models: Statecharts**
  - superstates and substates
  - Guidelines for drawing statecharts

Getting objects to behave

- All objects have "state"
  - The object either exists or it doesn't
  - If it exists, then it has a value for each of its attributes
  - Each possible assignment of values to attributes is a "state"
    - (and non-existence of the object is also a state)
- E.g. For a stack object
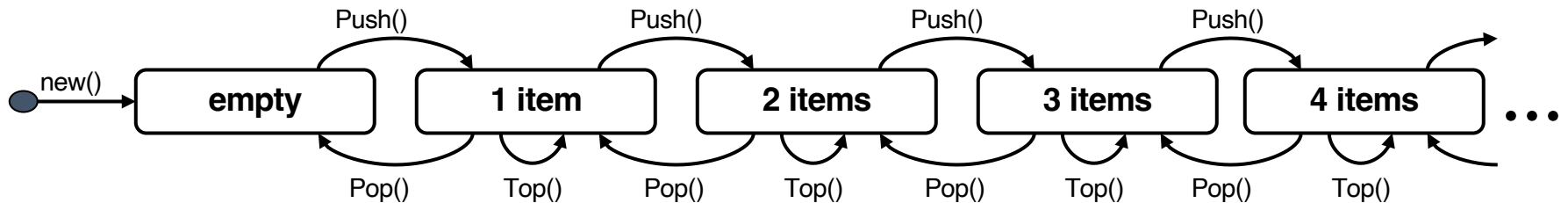
What does the model mean?

- Finite State Machines
  - There are a finite number of states (all attributes have finite ranges)
   - E.g. imagine a stack with max length = 3
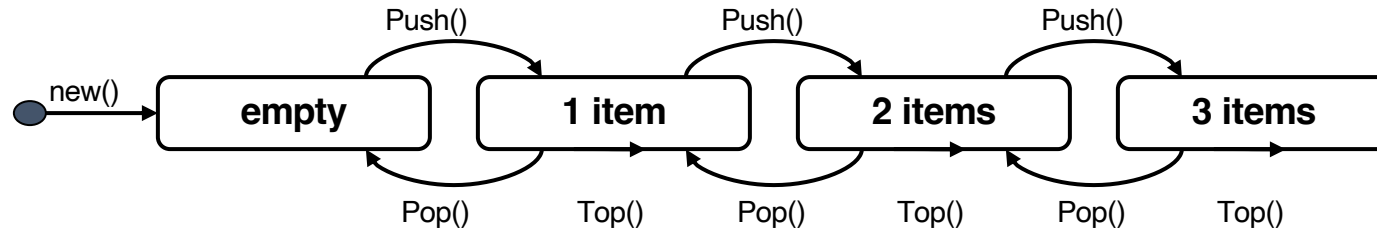
  - The model specifies a set of traces
    - E.g. new();Push();Push();Top();Pop();Push()...
    - E.g. new();Push();Pop();Push();Pop()...
    - There may be an infinite number of traces (and traces may be of infinite length)
  - The model excludes some behaviours
    - E.g. no trace can start with a Pop()
    - E.g. no trace may have more Pops than Pushes
    - E.g. no trace may have more than 3 Pushes without a Pop in between

Abstraction

- The state space of most objects is enormous
  - State space size is the product of the range of each attribute
    - E.g. object with five boolean attributes: 2^5+1 states
    - E.g. object with five integer attributes: $(maxint)^5+1$ states
    - E.g. object with five real-valued attributes: …?
  - If we ignore computer representation limits, the state space is infinite

- Only part of that state space is "interesting"
  - Some states are not reachable
  - Integer and real values usually only vary within some relevant range
  - Often, we're only interested in certain thresholds:
    - E.g. for Age, we may be interested in age<18; 18≤age≤65; and age>65
    - E.g. for Cost, we may want to distinguish cost≤budget, cost=0, cost>budget, and cost>(budget+10%)

Collapsing the state space
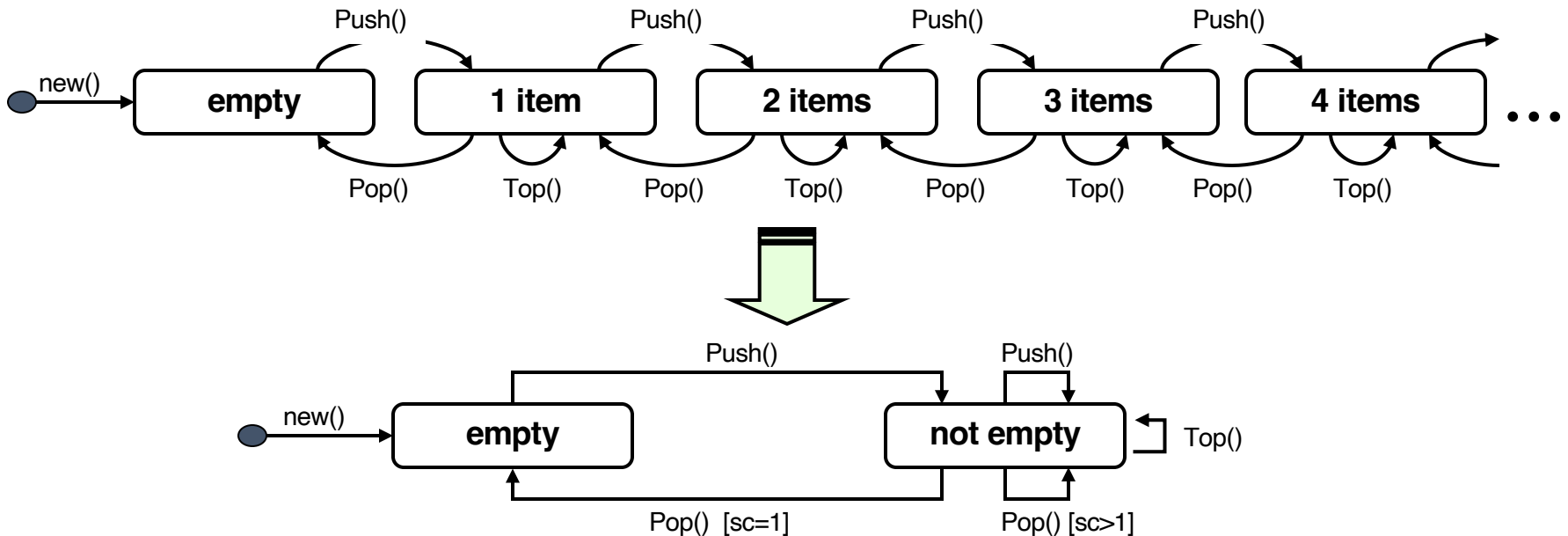
- The abstraction usually permits more traces
  - E.g. this model does not prevent traces with more pops than pushes
  - But it still says something useful

What are we modelling?

Application Domain

Machine Domain

D - domain properties

C - computers

R - requirements

S - specification

P - programs

(D) Observed states of an **application domain** entity?

- E.g. a phone can be idle, ringing, connected, …
- Model shows the states an entity can be in, and how events can change its state
- This is an **indicative** model

(R) Required behaviour of an **application domain** entity?

- E.g. a telephone switch shall connect the phones only when the callee accepts the call
- Model distinguishes between traces that are desired and those that are not
- This is an **optative** model

(S) Specified behaviour of a **machine domain** entity?

- E.g. when the user presses the 'connect' button the incoming call shall be connected
- Model specifies how the machine should respond to input events
- This is an **optative** model, in which all events are shared phenomena

the world vs. the machine
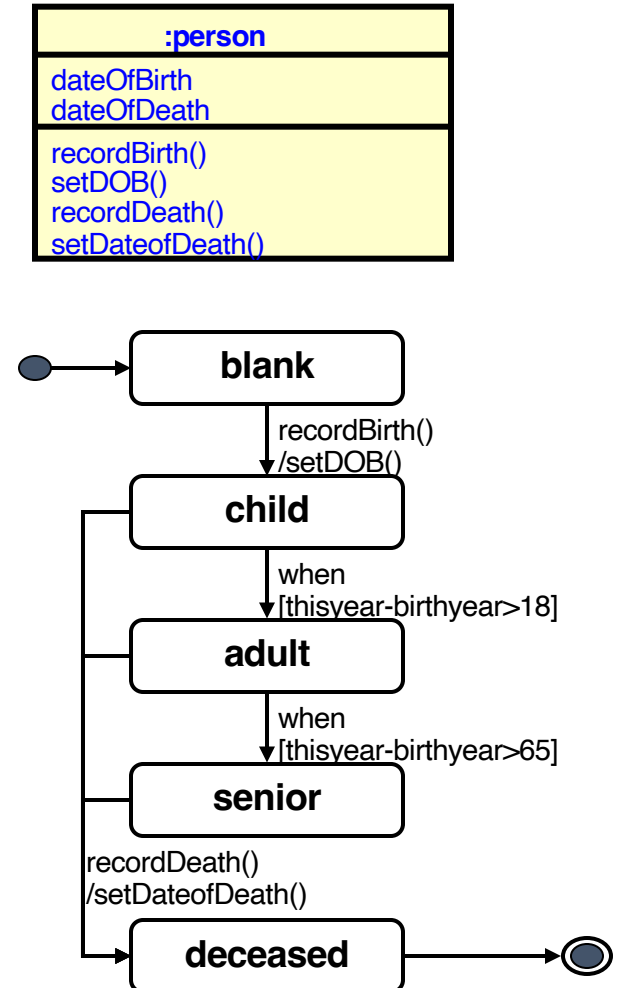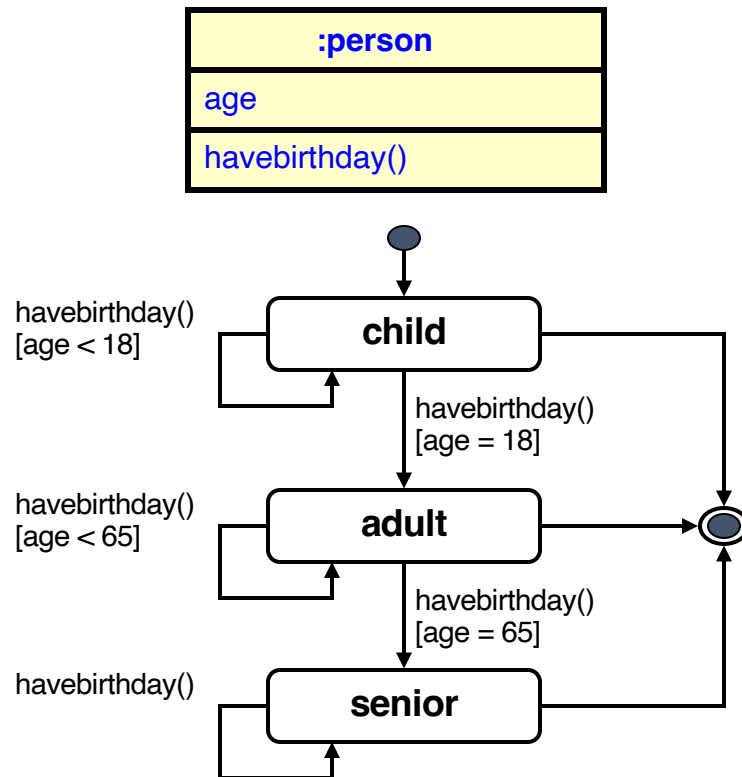
StateCharts

- States
  - "interesting" configurations of the values of an object's attributes
  - may include a specification of action to be taken on entry or exit
  - States may be nested
  - States may be "on" or "off" at any given moment
- Transitions
  - Are enabled when the state is "on"; disabled otherwise
  - Every transition has an event that acts as a trigger
  - A transition may also have a condition (or guard)
  - A transition may also cause some action to be taken
  - When a transition is enabled, it can fire if the trigger event occurs and its guard is true
  - Syntax: **event [guard] / action**
- Events
  - occurrence of stimuli that can trigger an object to change its state
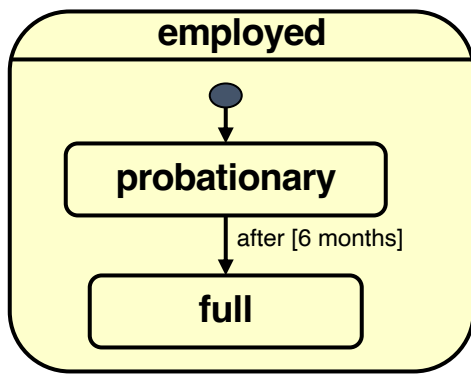 - determine when transitions can fire

Superstates

→ States can be nested, to make diagrams simpler
  ⇨ A superstate consists of one or more states.
  ⇨ Superstates make it possible to view a state diagram at different levels of abstraction.
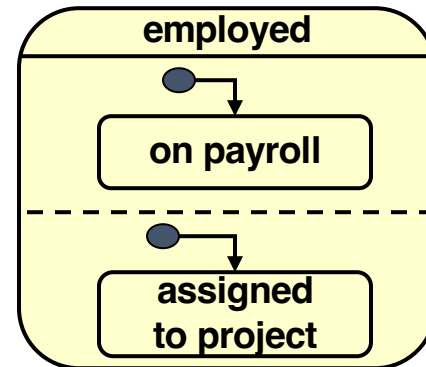
OR superstates

- when the superstate is "on", only one of its substates is "on"
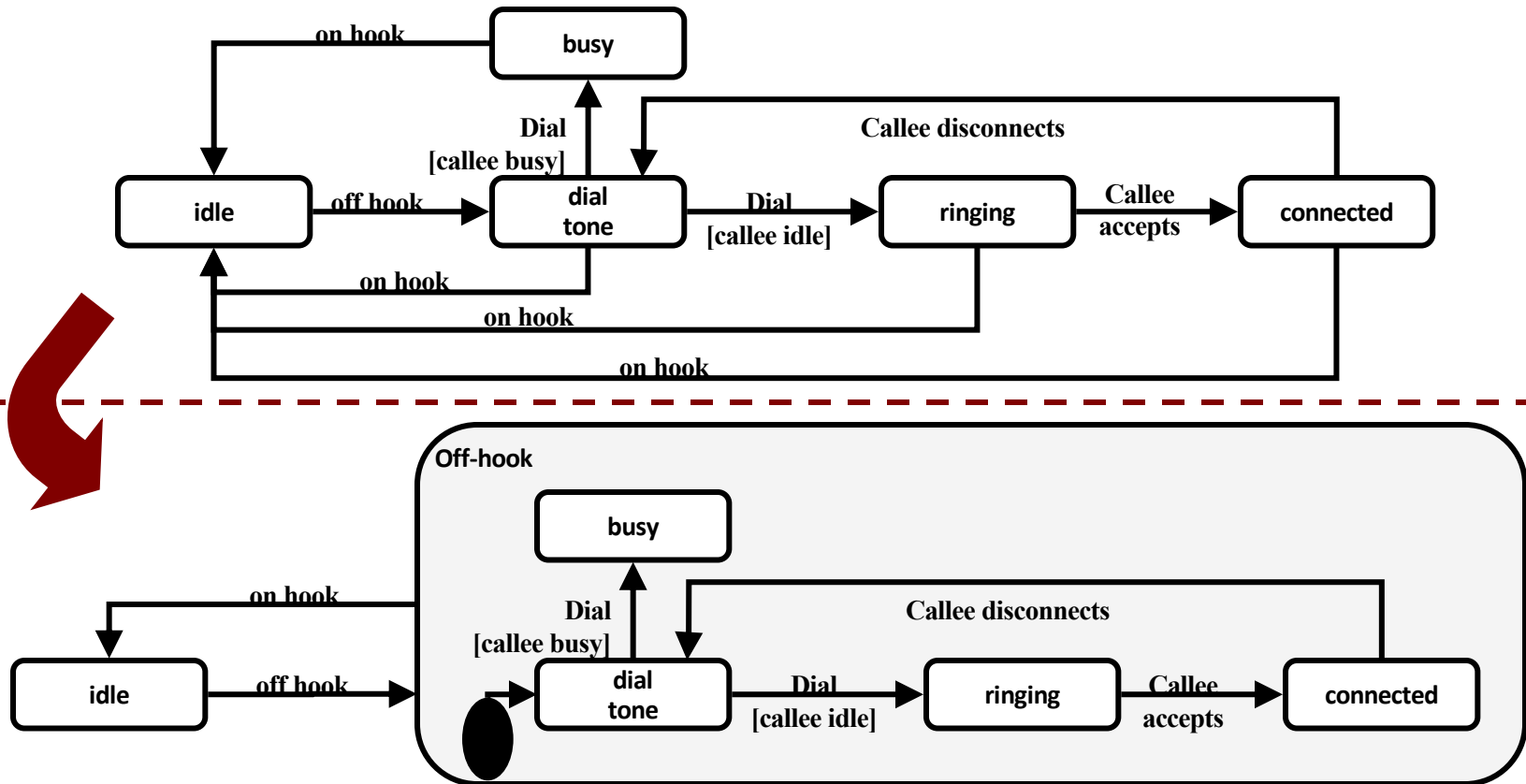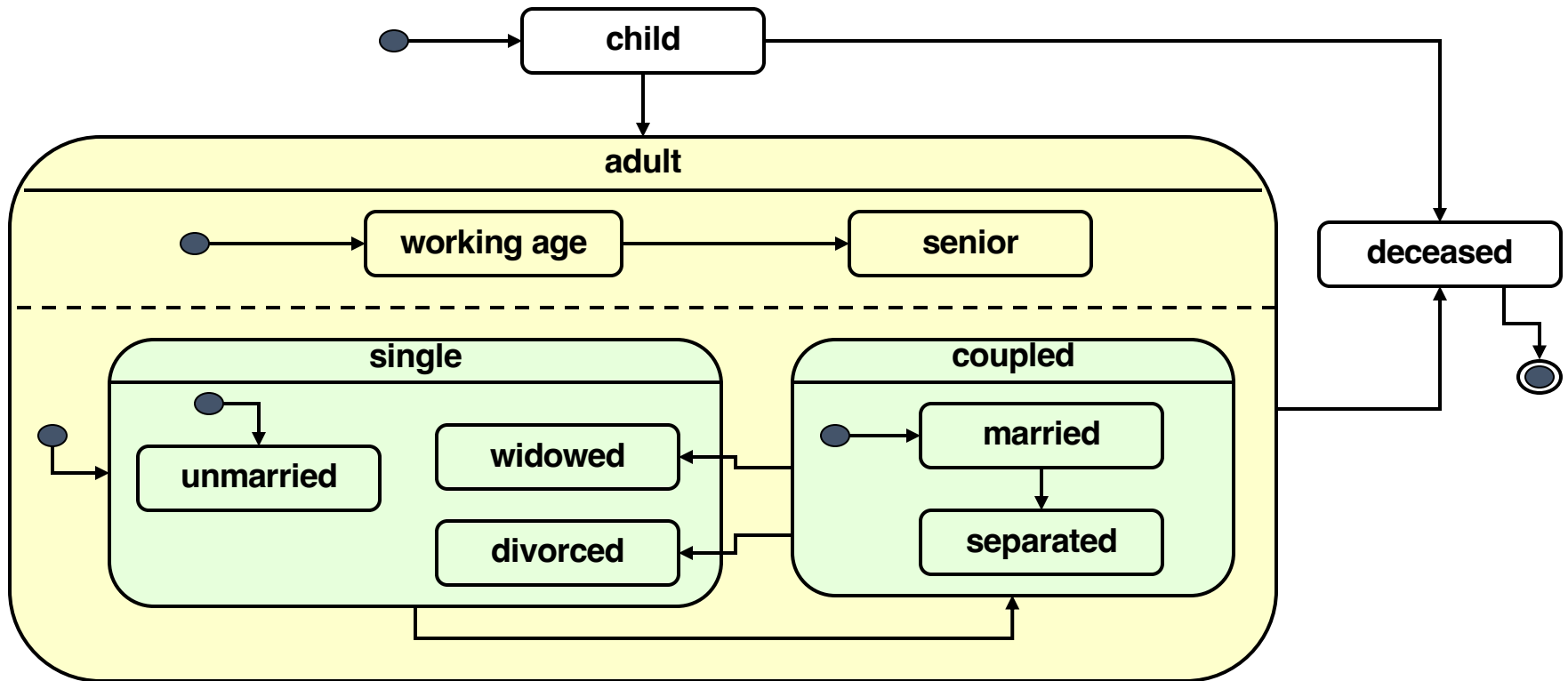
AND superstates
(concurrent substates)

- When the superstate is "on", all of its states are also "on"
- Usually, the AND substates will be nested further as OR superstates

employed

probationary

after [6 months]

full

employed

on payroll

assigned to project

Superstates example

A more detailed example

States in UML

- A state represents a time period during which
  - A predicate is true
    - e.g. (budget - expenses) > 0,
  - An action is being performed, or an event is awaited:
    - e.g. checking inventory for order items
    - e.g. waiting for arrival of a missing order item
- States can have associated activities:
  - do/activity
    - carries out some activity for as long as the state is "on"
  - entry/action   and   exit/action
    - carry out the action whenever the state is entered (exited)
  - include/stateDiagramName
    - "calls" another state diagram, allowing state diagrams to be nested

Events in UML

- Events are happenings the system needs to know about
  - Must be relevant to the system (or object) being modelled
  - Must be modellable as an instantaneous occurance (from the system's point of view)
    - E.g. completing an assignment, failing an exam, a system crash
  - Are implemented by message passing in an OO Design

- In UML, there are four types of events:
  - *Change events* occur when a condition becomes true
    - denoted by the keyword 'when'
    - e.g. when[balance < 0]
  - *Call events* occur when an object receives a call for one of its operations to be perfomed
  - *Signal events* occur when an object receives an explicit (real-time) signal
  - *Elapsed-time events* mark the passage of a designated period of time
    - e.g. after[10 seconds]

(1) **Call event**: The event of receiving a call for an operation that is implemented by actions on state machine transitions.

- Call event的语法格式如下：

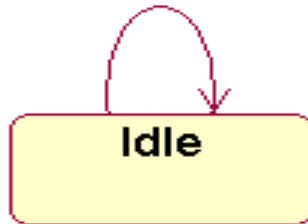**事件名（[逗号分隔的参数列表]）**

其中参数列表中的参数格式为：

例：

(2) **Change event**: The event of a Boolean expression becoming satisfied because of a change to one or more of the values it references.

- Change event用关键字when表示。例

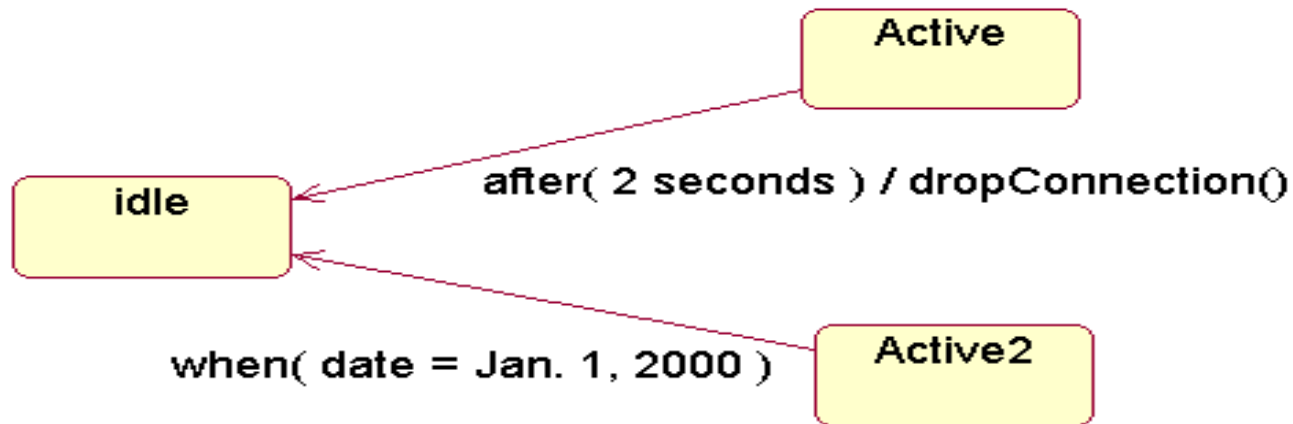when(temperature > 120) / alarm()

Idle

- Change event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。

(3) **Time event**: An event that denotes the satisfaction of a time expression, such as the occurrence of an absolute time or the passage of a given amount of time after an object enters a state.

- Time event用关键字after或when表示。

例:

(4) Signal event: An event that is the receipt by an object of a signal sent to it, which may trigger a transition in its state machine.

- Signal event的语法格式和Call event一样。

- 信号事件是一个异步事件，调用事件一般是一个同步事件。

Checking your Statecharts

- **Consistency Checks**
  - All events in a statechart should appear as:
    - operations of an appropriate class in the class diagram
  - All actions in a statechart should appear as:
    - operations of an appropriate class in the class diagram

- **Style Guidelines**
  - Give each state a unique, meaningful name
  - Only use superstates when the state behaviour is genuinely complex
  - Do not show too much detail on a single statechart
  - Use guard conditions carefully to ensure statechart is unambiguous
    - Statecharts should be deterministic (unless there is a good reason)

- **You probably shouldn't be using statecharts if:**
  - you find that most transitions are fired "when the state completes"
  - many of the trigger events are sent from the object to itself
  - your states do not correspond to the attribute assignments of the class

Statecharts & Relationship to Other Diagrams

- Events on the statechart diagram should appear as an incoming message for the appropriate object on a sequence and collaboration diagram.

  状态图中的事件应为顺序图/交互图中该对象的输入消息

- A statechart diagram is prepared for every object class in the Class Diagram with non-trivial behaviour.
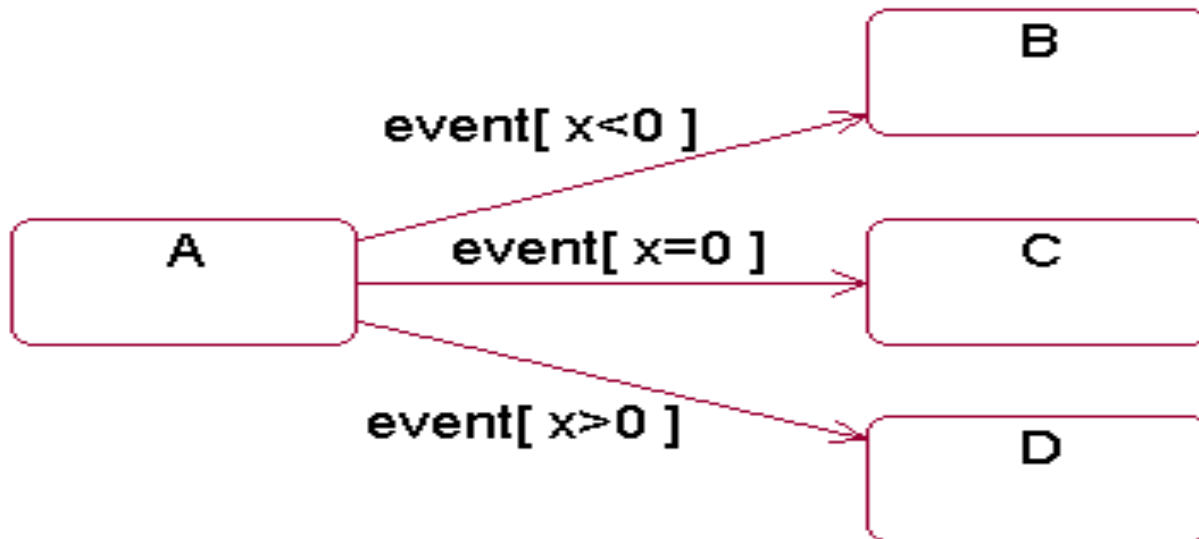
  状态图应针对类图中具有重要行为的类进行建模

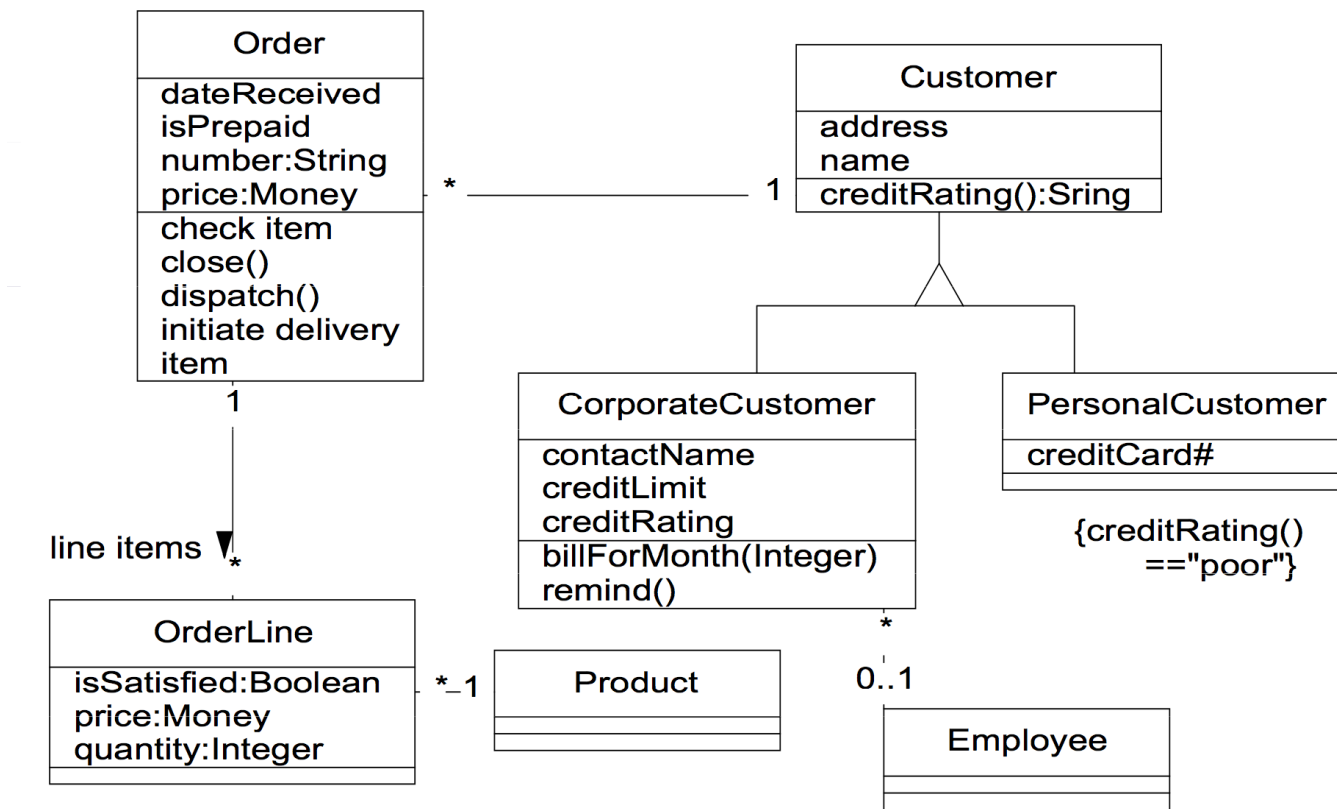- Every event should correspond to an operation on the appropriate class.

  每个事件均映射为相应类的一个动作

- 一个转移包括五部分：source state, event trigger, guard condition, action, target state.

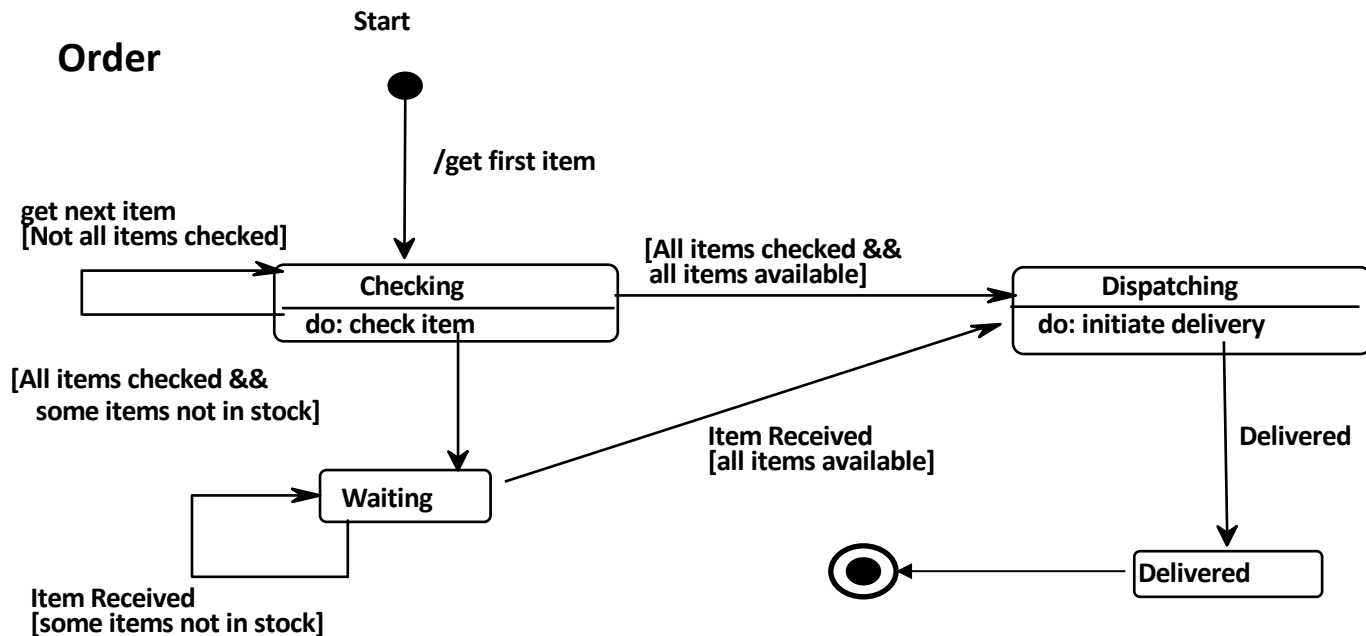- 对于一个给定的状态，最终只能产生一个转移，因此从相同的状态出来的、**事件相同的几个转移之间的条件应该是互斥的**。

例：

Class Diagram: Order Processing

Order Statechart Diagram

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

Add a new state and separate transitions to the diagram to show that an order can be cancelled at any point before it is delivered.
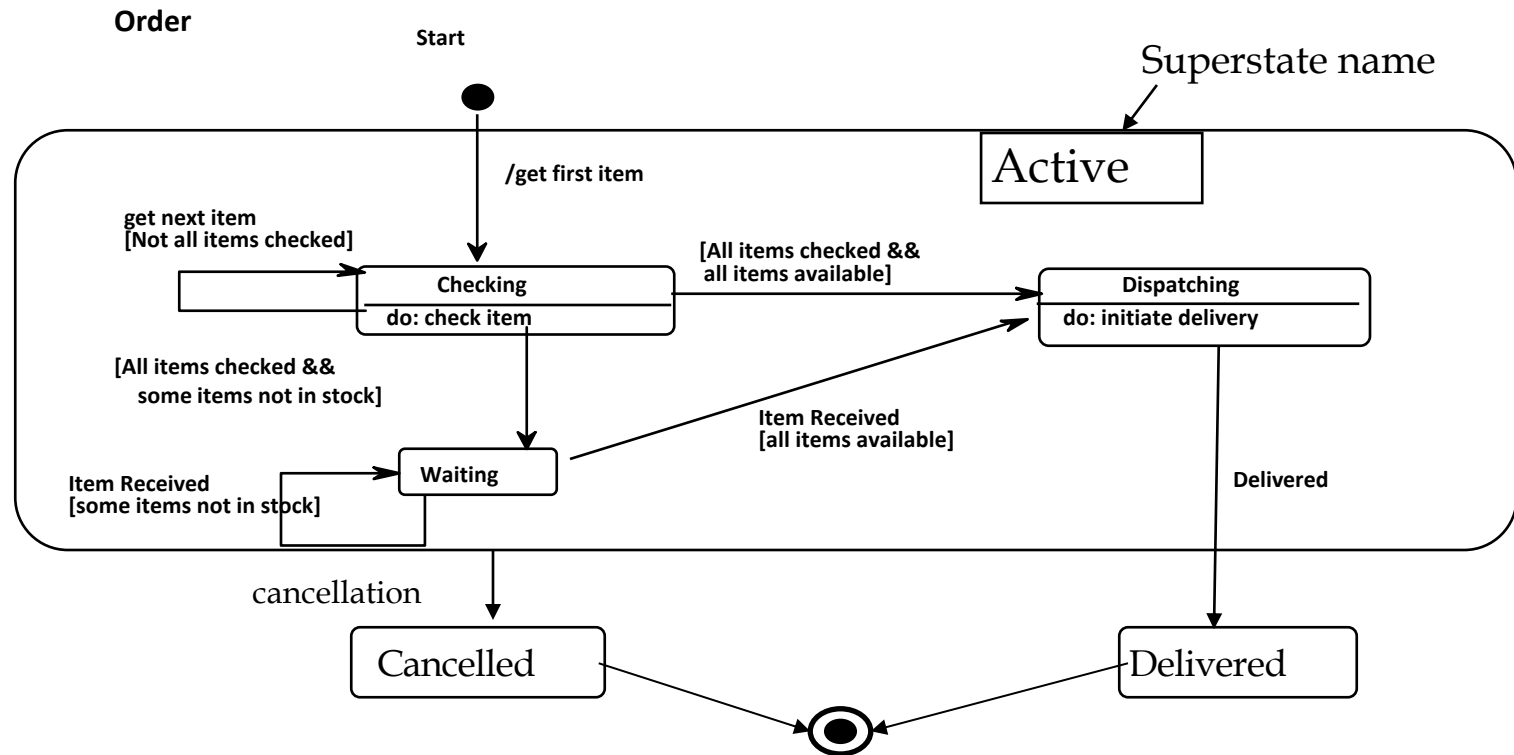
Supertransition 超级迁移

- Where transitions may occur from several states to a single state the Statechart Diagram may show this single state as a *Supertransition*

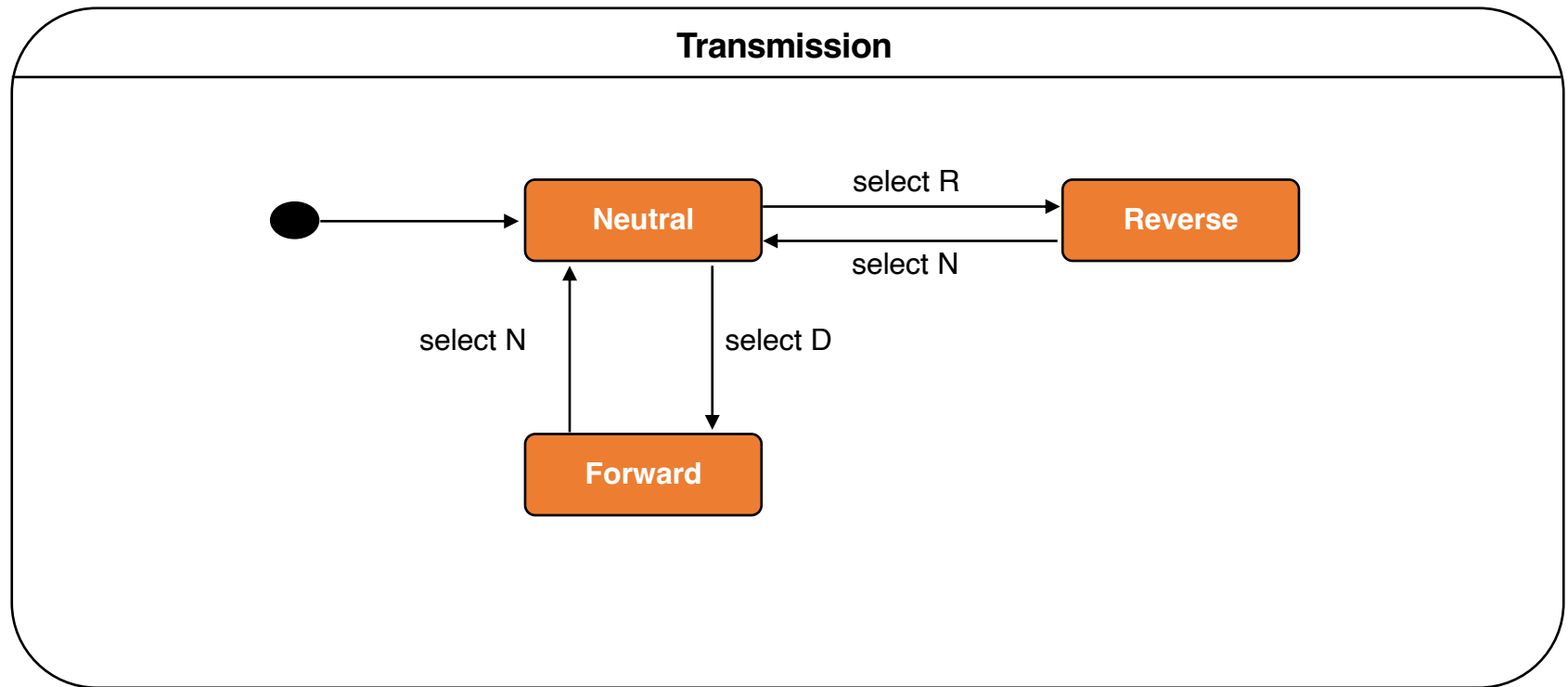 可将由多个状态发出的指向同一个状态的迁移定义为超级迁移

  - e.g. a Purchase Order may be cancelled at any point prior to delivery.

- This may be drawn with several transitions or by creating a Superstate.
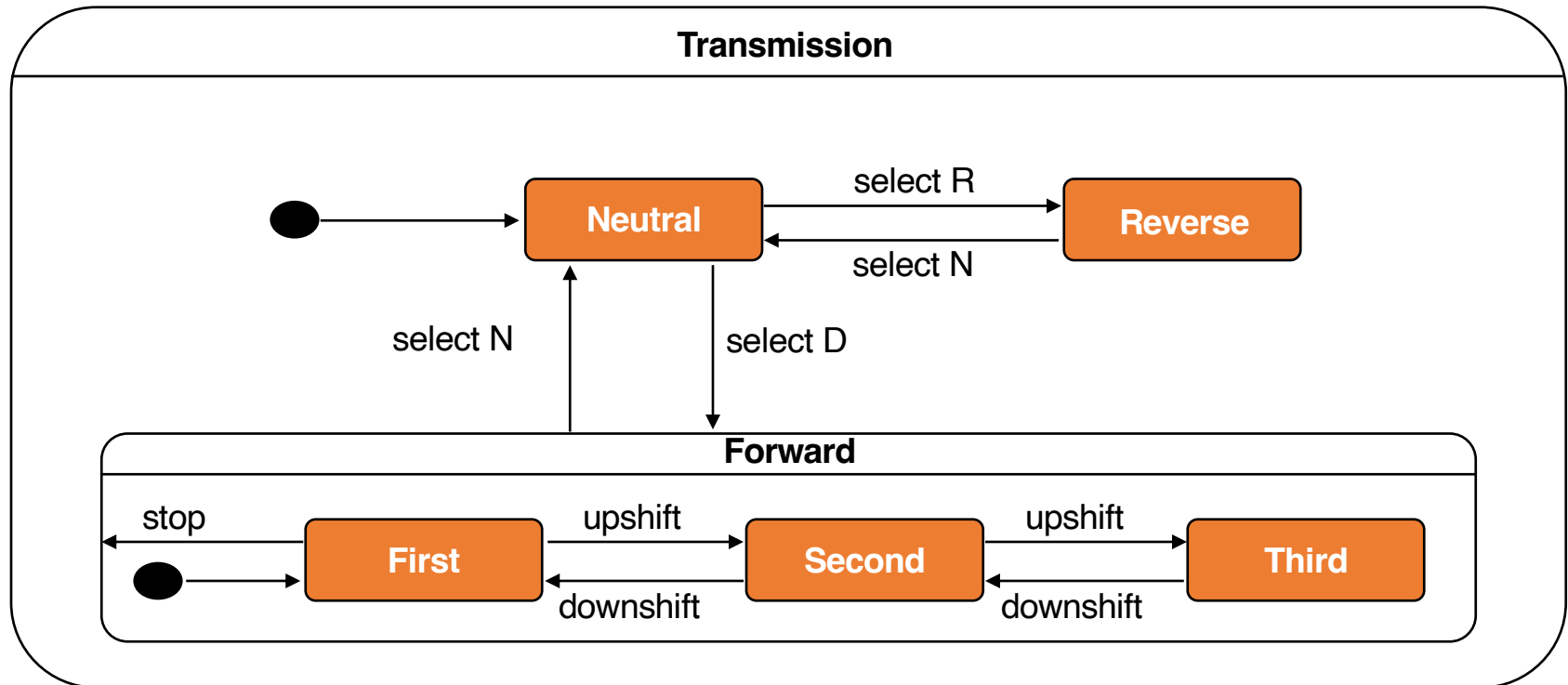
  超级迁移可图形表示为多条迁移，或为之创建一个超级状态

Order Statechart Diagram: Superstates & Supertransition

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]

Waiting

Item Received
[some items not in stock]

Delivered

cancellation

Cancelled

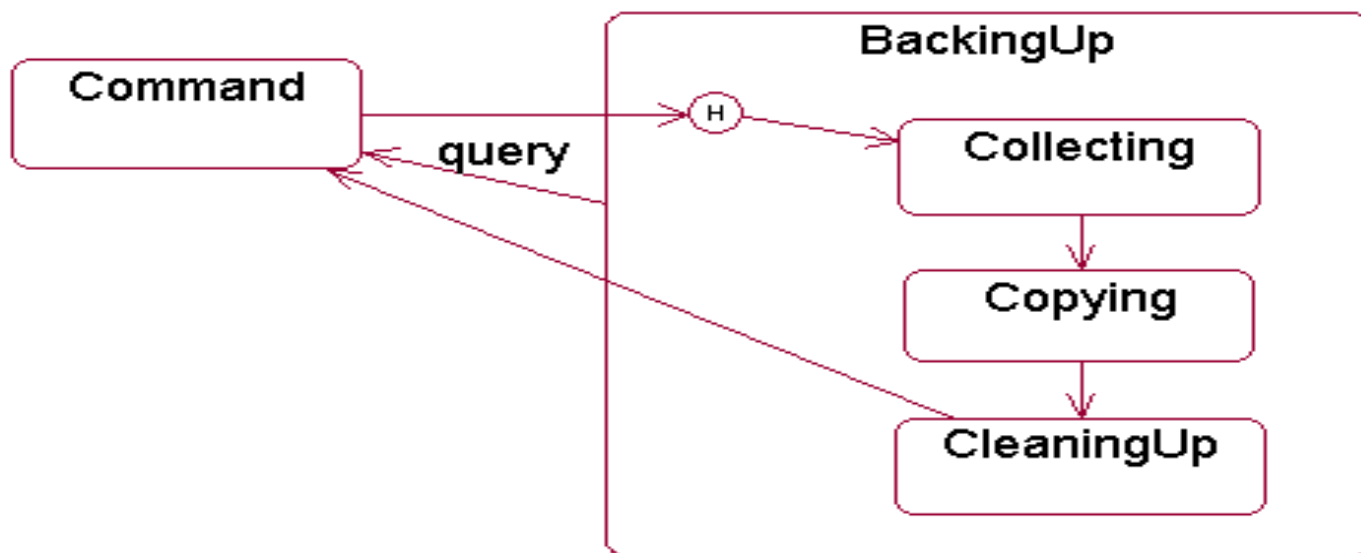| Delivered |
|---|

Nested (Super) States

Nested (Super) States

History State(历史状态)

- History State: A pseudostate whose activation restores the previously active state within a composite state.

- 使用历史状态，可以记住从组合状态中退出时所处的子状态，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: 历史状态的例子。

Action(动作)

- An **action(动作)** is an executable atomic computation.

- 一个**动作**是一个可执行的原子计算。

说明：

- 动作是原子的，不可被中断的，其执行时间可忽略不计的。

- UML并没有规定描述action的语言格式，一般建模时采用实际的程序设计语言来描述。

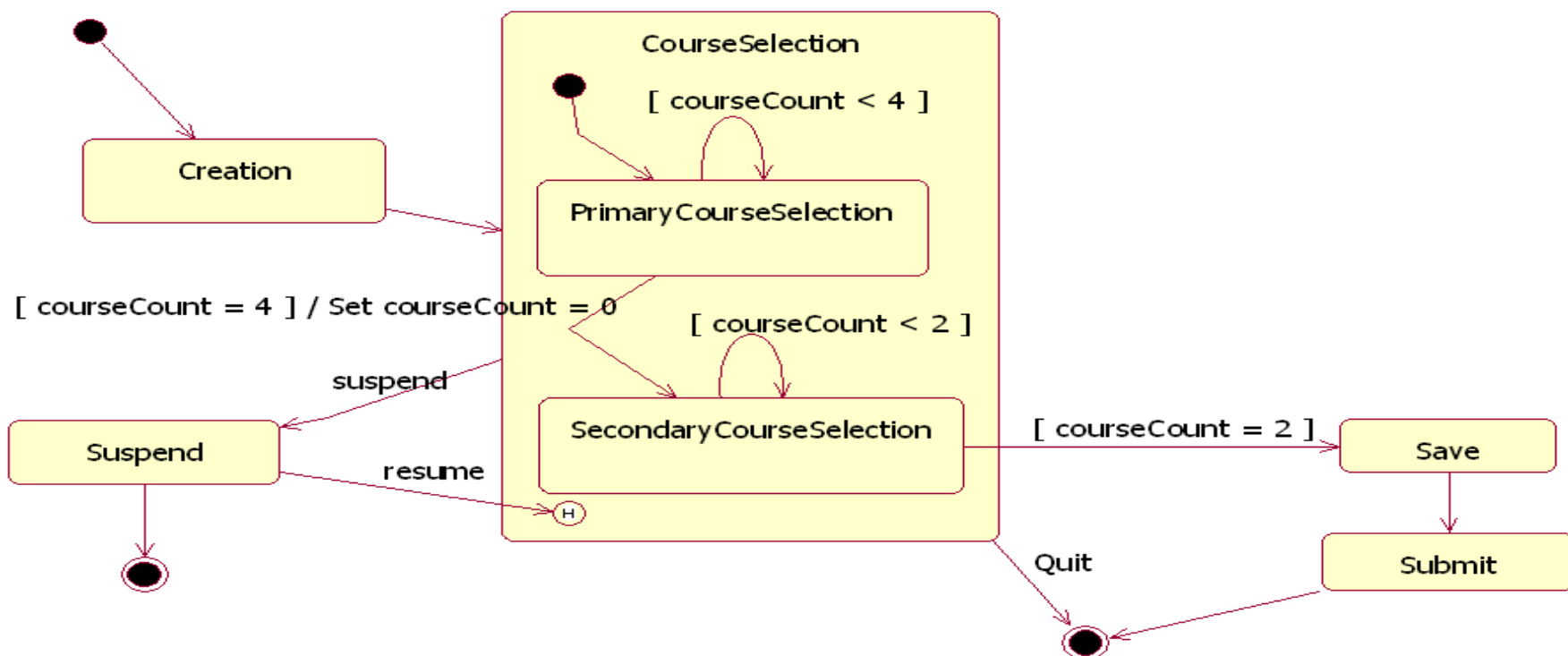- 两种特殊的动作: entry action(进入动作)和exit action(退出动作)。
  - Entry动作：进入状态时执行的活动，格式如下:

    'entry' '/'action-expression

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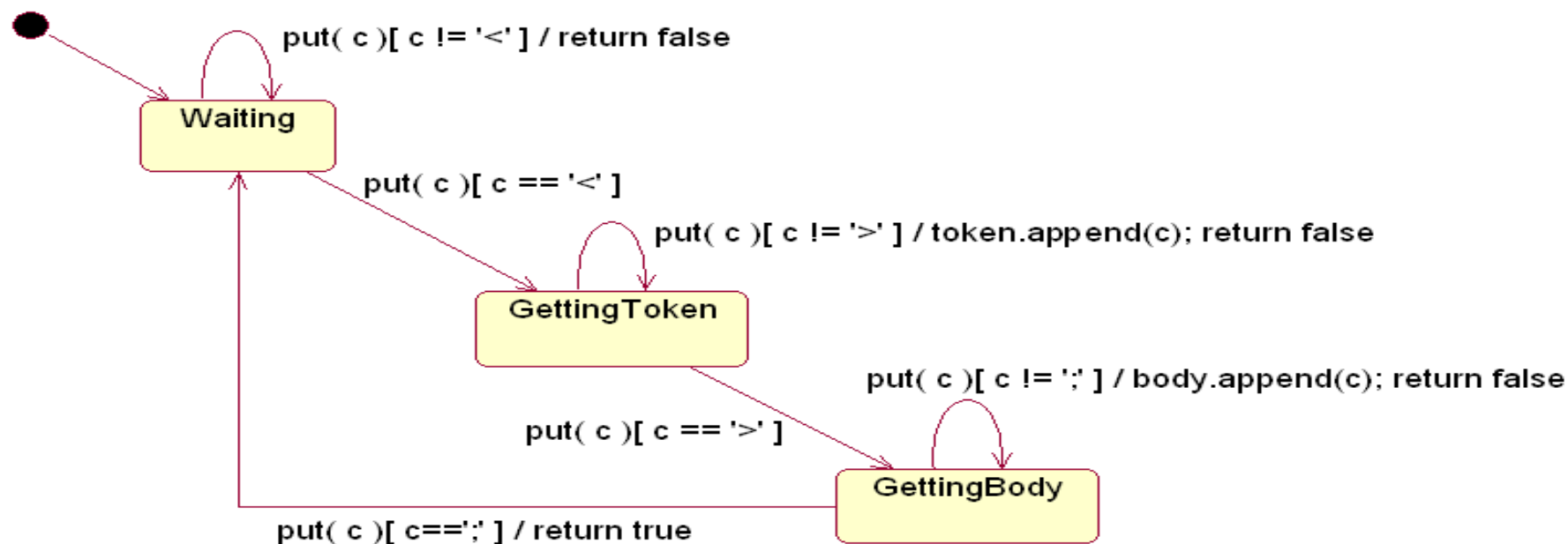
    'exit' '/'action-expression

  (其中 action-expression 可以用到对象本身的属性和输入事件的参数)

类型为RegistrationController的对象的状态图：

状态图的工具支持

- 正向工程：根据状态图生成代码。例：

```java
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。
  - 这样做是为了反映转移是事件的结果这个事实。也就是说，事件出现在转移之前，因此引用事件时要用过去式。
  - 例如：用cancelled， closed， scheduled等命名事件。

- 建模风格3：警戒条件不要重叠。
  - 从一个状态出来的相似转移上的警戒条件相互之间必须互斥。
  - 例如，像x < 0，x = 0和x > 0这样的警戒条件是互斥的，而像x <= 0和 x >= 0这样的警戒条件是非互斥的。
- 建模风格4：不要把警戒条件置于初始转移上。
  - 当警戒条件为假的时候，这个对象做什么？

画出描述录像带生命进程的状态图

- 假定录像带首先被工厂生产出来，然后出售，被录像带商店购入后，进行适当的包装和编号，存入录像带商店的数据库，准备出租。顾客选中并准备租用时，在柜台取带，并于三天内归还。如果逾期不还，录像带商店将在次日打电话给顾客催还。两天后将再次打电话给顾客催还。如再过两天后仍未还带，商店将委托代理取带，并从数据库中清除该录像带。如果还回的录像带破损，也将从数据库中清除该带。如果在年度检查中发现该录像带从库存中遗失，也将被清除。

- 注意画出状态迁移的事件和条件。
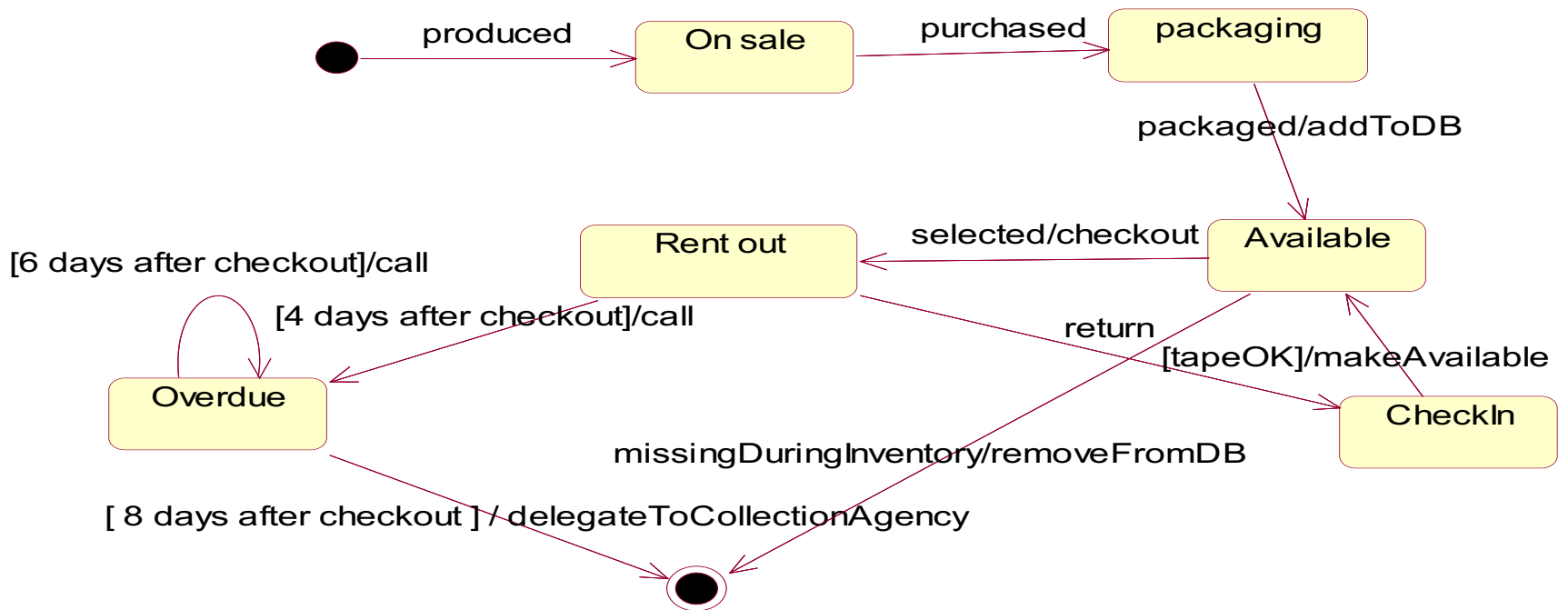
Activity Diagram(活动图)

- An activity diagram is a special case of a state diagram

  活动图是状态图的特例

- in which all (or at least most) of the states are action or sub-activity states

  状态为持续性的活动

- and in which all (or at least most) of the transitions are triggered by completion of the actions or sub-activities in the source states.

  迁移由活动的结束来触发

  (UML specification 1.5, pp3-156)

Activity Diagrams

- Activity diagrams describe activities which involve concurrency and synchronization.

  活动图用于描述并发和同步

- Variation of Statecharts that focuses on a flow of activity driven by internal processing within an object rather than events that are external to it.

  与状态图的不同在于它关注对象内部的事务处理，而非外部的相关事件。

Activity Diagrams

- Can be used
  - To model business workflows (in business modeling for instance).

    对系统的工作流(workflow)建模，即对系统的业务过程建模。

  - To describe a system function that is represented within a use case or between use cases.

    对系统功能建模，尤其是那些表示为一个或一组用例的功能

  - In operation specifications, to describe the logic of an operation.

    对具体的操作建模，描述计算过程的细节。

例1：用活动图对工作流程建模的例子。
(1)：产品制造和发货的use case图。
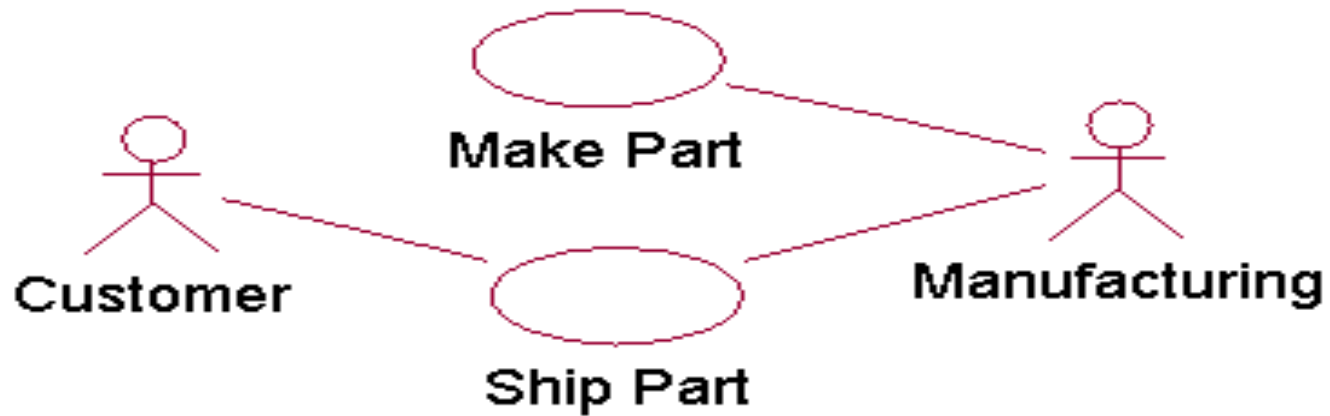
(2). 用活动图来说明具体的工作流程。
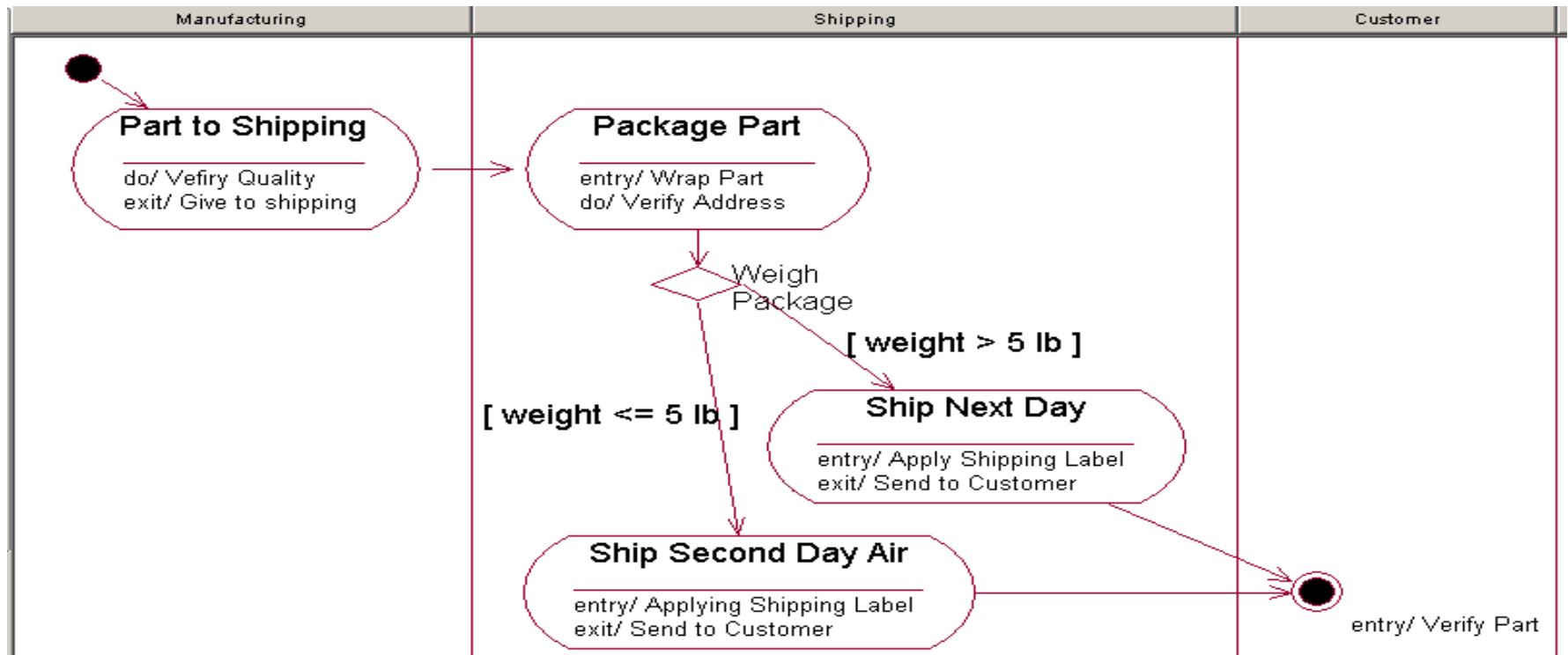
活动图中的基本概念

- Activity (活动)
- transition (转移)
- swimlane (泳道)
- branch (分支)
- fork and join (分叉和汇合)
- object flow (对象流)

activity(活动)
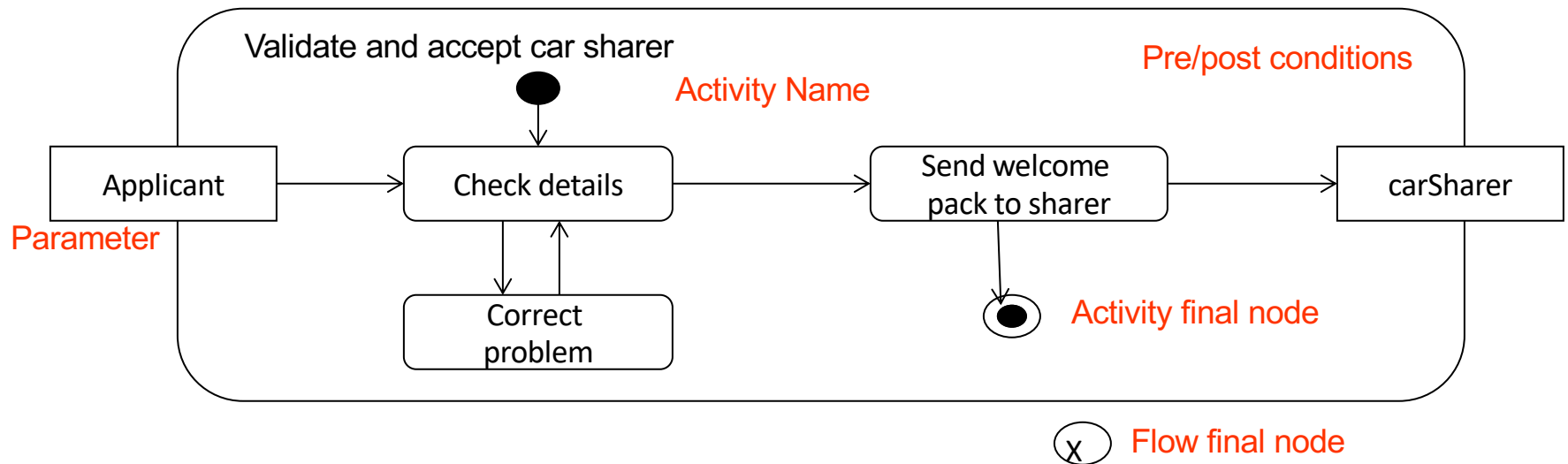
- An activity represents the performance of task or duty in a workflow. It may also represent the execution of a statement in a procedure.

说明:

- Action 和activity的区别:
  - Action是原子的，而activity可进一步分解。

Activities (in UML 2.0)

Activities consists of action nodes, object nodes, and control nodes linked by activity edges.

Validate and accept car sharer

Activity Name

Pre/post conditions

Applicant → Check details → Send welcome pack to sharer → carSharer

Parameter

Correct problem

Activity final node

x Flow final node

Actions

- An action represents an activity state in the execution of the activity. An activity state normally contains an action expression and usually has no associated name.

- Actions may be described by:
  - Natural language
  - Structured English
  - Pseudo-code
  - Programming language
  - Another activity diagram

- An action expression may only use attributes and links of the owning object.

More About Activity Diagrams

- **Decision point/node:**
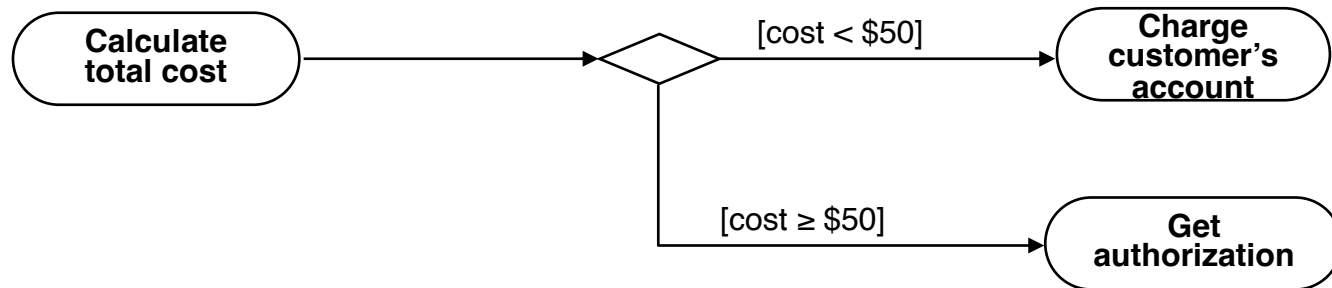
- **Dead end:** there may be transitions in an activity diagram with no destination state; this can mean that:

  - Not all processing has been specified,

  - Or, that another activity diagram will take over.

Swimlanes/ Activity Partition

- An activity diagram can be divided into swimlanes.

- A **swimlane** is a partition on activity graphs for organizing responsibilities for activities.

- Each swimlane:
  - Represents one focus of responsibility(责任区) in the activity.
  - May be handled by a distinct operation in one or more objects.

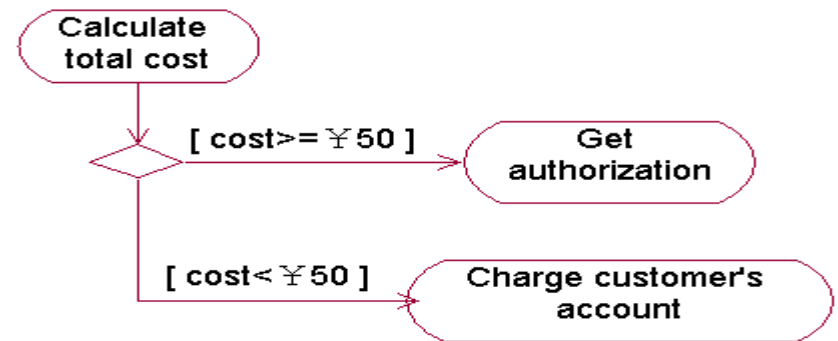- The order of the swimlanes has no significance.

Swimlanes
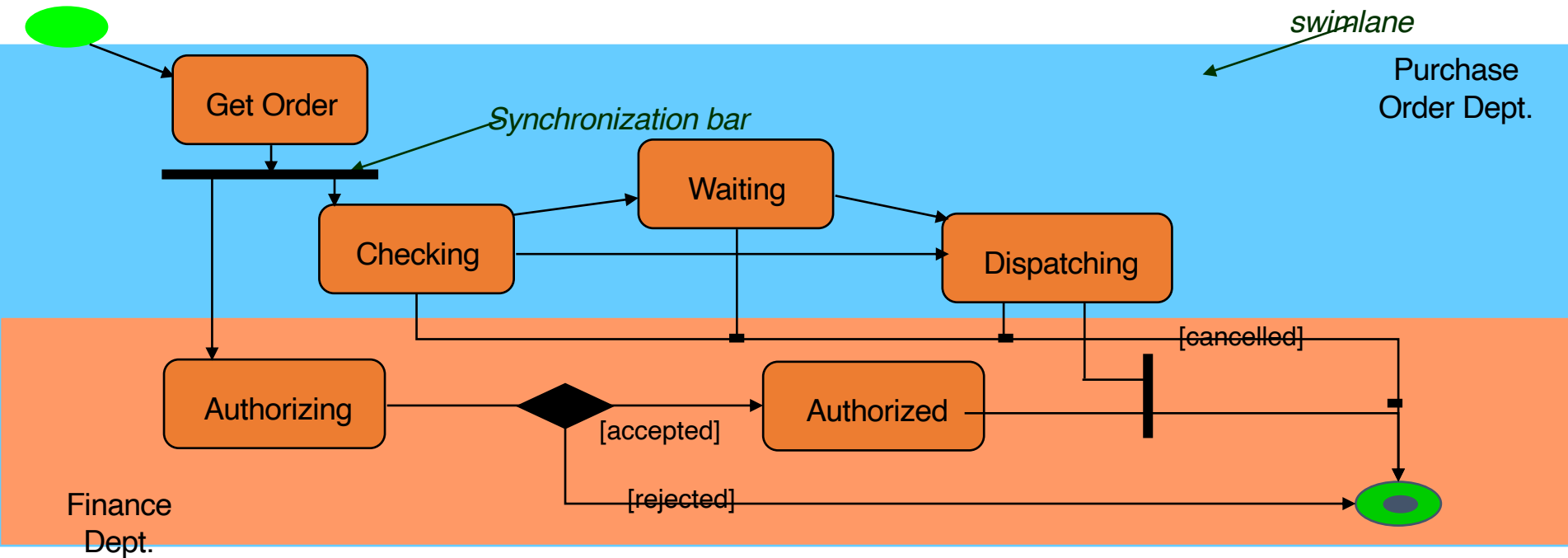
branch(分支)

- A branch is an element in a state machine in which a single trigger leads to more than one possible outcome, each with its own guard condition.
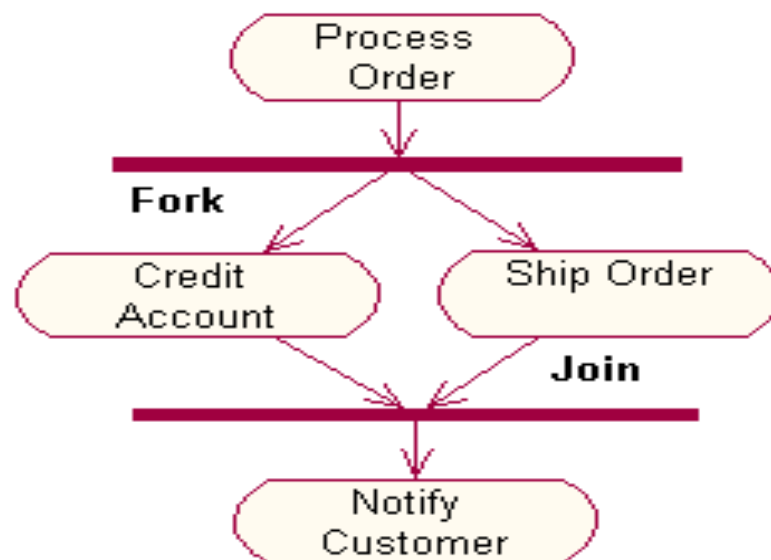
- 表示分支的两种方法:
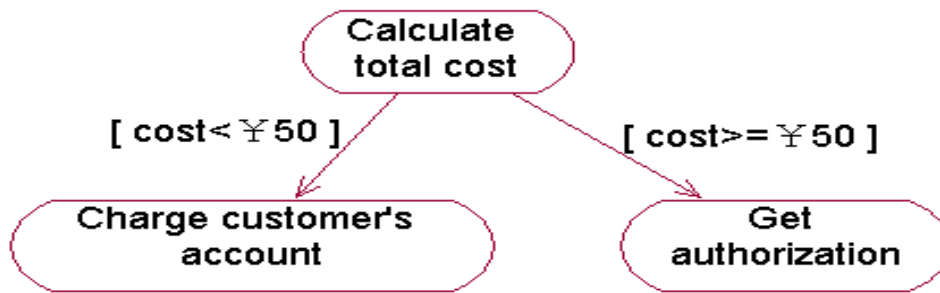
方法1

方法2

fork和join (分叉和汇合)

- A **fork** is a complex transition in which one source state is replaced by two or more target states, resulting in an increase in the number of active states.

- A **join** is a transition with two or more source states and one target state.
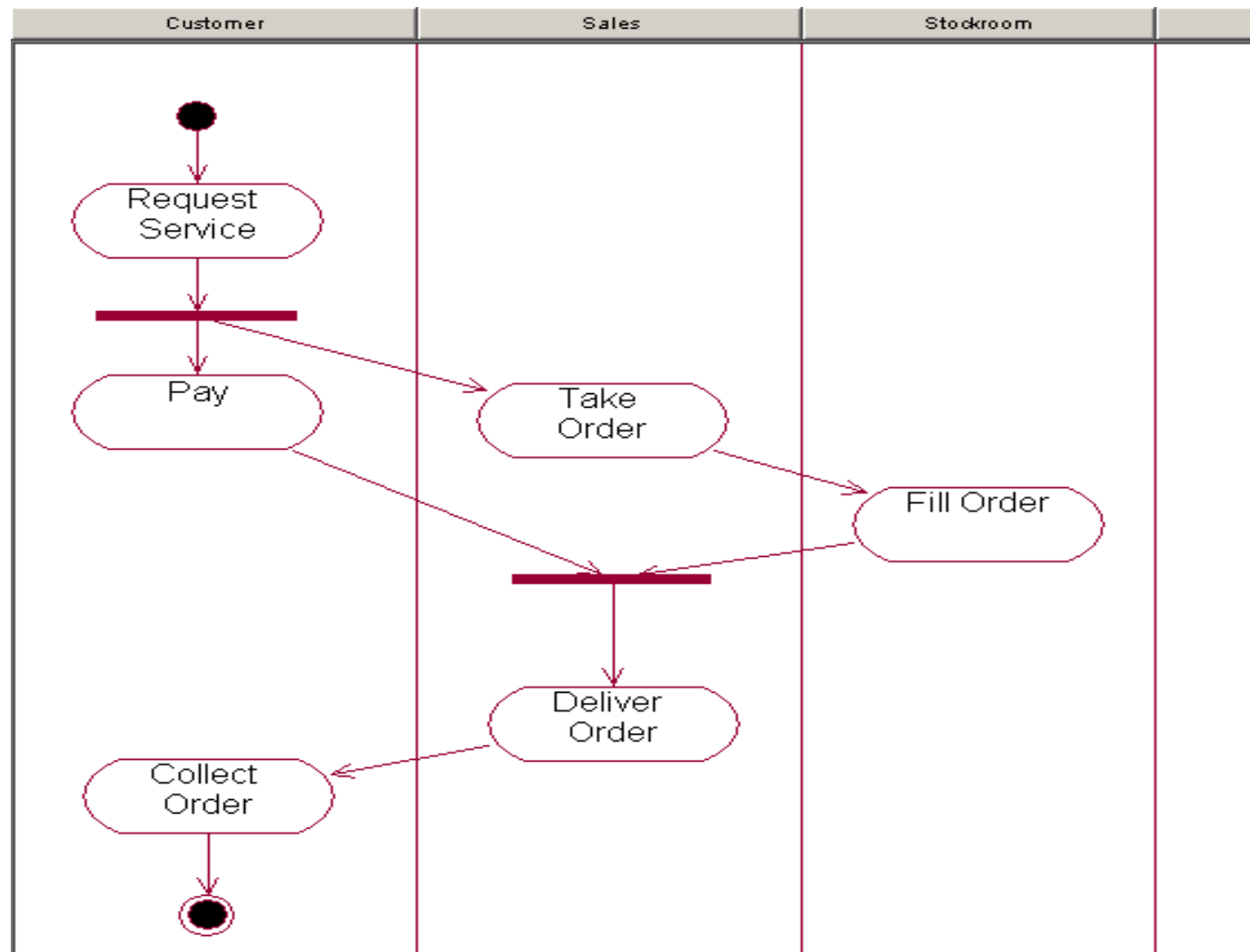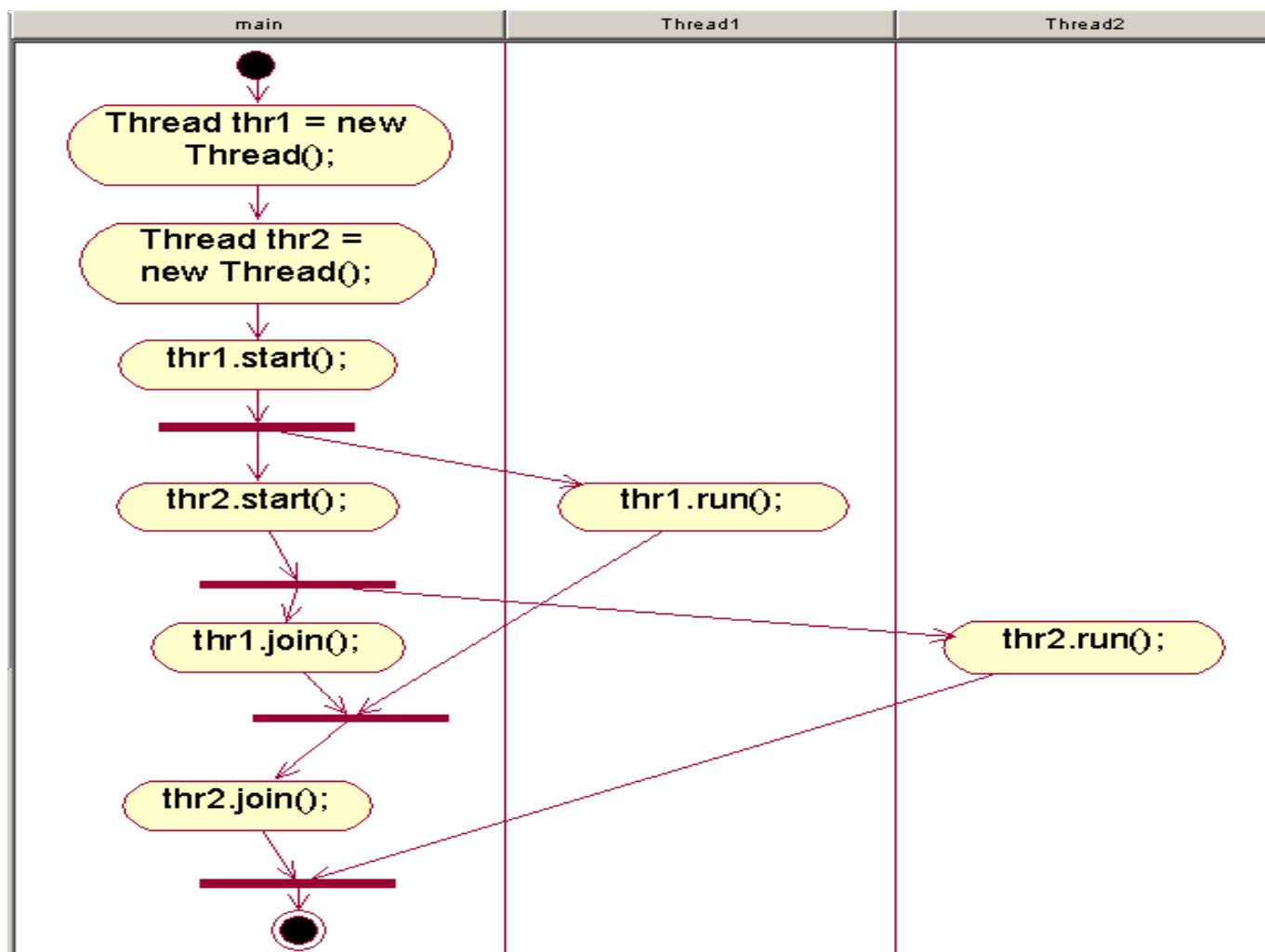
说明:

- 利用fork和join可以表示系统中或对象中的并发行为。
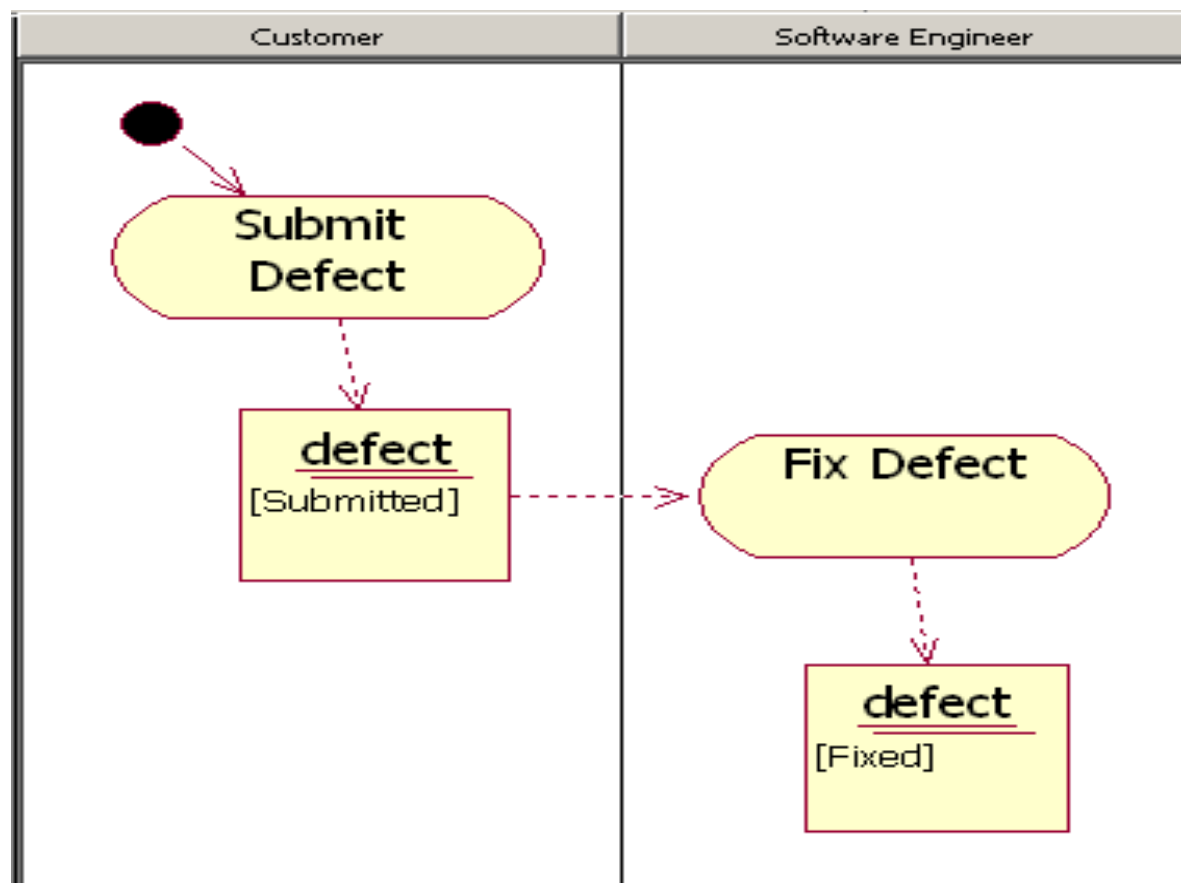
例: fork和join的例子。

例：使用泳道的例子。

例：用活动图描述 Java中使用多线程 的过程。

对象流

- 在活动图中可以出现对象。对象可以作为活动的输入或输出。

- An **object flow** on an activity diagram represents the relationship between an activity and the object that creates it (as an output) or uses it (as an input).
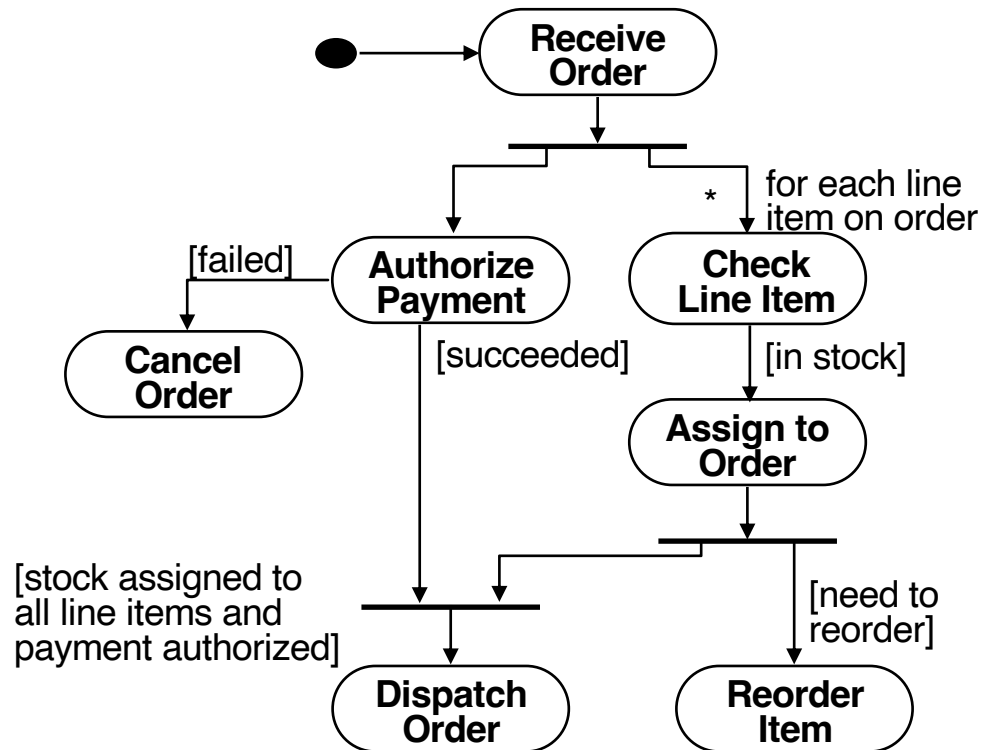
- 对象流的例子。

Example: Order Processing

- The Process Order use case:

  "When we <u>receive an order</u>, we <u>check each line item</u> on the order to see if we have the goods in stock. If we do, we <u>assign the goods</u> to the order. If this assignment sends the quantity of those goods in stock below the reorder level, we <u>reorder the goods</u>. While we are doing this, we <u>check</u> to see if <u>the payment</u> is O.K. If the payment is O.K. and we have the goods in stock, we <u>dispatch the order</u>. If the payment is O.K. but we do not have the goods, we <u>leave the order waiting</u>. If the payment is not O.K., we <u>cancel the order</u>."

Order Processing Activity Diagram

With Swimlanes

Finance

Order Processing

Stock Manager

Receive Order

for each line item on order

*

Authorize Payment

[failed]

[in stock]

Cancel Order

Check Line Item

Choose Outstanding Order Items

Receive Supply

for each chosen order item

*

[succeeded]

Assign to Order

Assign Goods to Order

[stock assigned to all line items and payment authorized]

[need to reorder]

[all outstanding order items filled]

Dispatch Order

Reorder Item

Add Remainder to Stock

活动图与状态图比较

1. 活动图和状态图描述的重点不同：
   - 活动图描述的是从activity到activity的控制流，而状态图描述的是对象的状态及状态之间的转移。

2. 活动图和状态图使用的场合不同：
   - 对于以下几种情况可以使用活动图：
     - 分析用例
     - **理解涉及多个用例的工作流**
     - 处理多线程应用
   - 对于下面的情况要使用状态图：
     - 显示一个对象在其生命周期内的行为。

说明：如果要显示多个对象之间的交互情况，可用顺序图或协作图。

活动图建模风格

- 建模风格1：确保从决策点出来的每个转移都有一个警戒条件。
  - 这可以确保建模人员已经考虑到这个决策点的所有可能情况。

- 建模风格2：确保决策点上的警戒条件形成一个完备集。
  - 不管什么情况，要一定能够可以从决策点离开。
  - 例如，像x < 0和x > 0这样的警戒条件是不完备的，因为当x等于0时会发生什么并不清楚。

- 建模风格3：警戒条件不要重叠。

- 建模风格4：确保每个分叉只有一个进入转移。
  - 当发现需要几个转移到同一个分叉时，表明建模人员或者遗漏了一个活动，或者在该点事实上并不需要并发活动。
- 建模风格5：确保每个汇合只有一个退出转移。
  - 当某个汇合点需要多个退出转移时，表明仍然需要并发的活动，因此，把汇合点沿着总体活动过程向前移动。
- 建模风格6：要小于5条泳道。
  - 泳道的一个缺点是使得建模人员无法自由地在图中排列活动以节省空间，因此会增加图的大小。泳道数目越多，这个问题越严重。

Reference

- 清华大学国家级精品课程《软件工程》主讲人 刘强 副教授 刘璘 副教授

- https://www.icourses.cn/sCourse/course_3016.html

- https://www.xuetangx.com/course/THU08091000367/5883555?channel=learn_title

谢谢大家！

THANKS