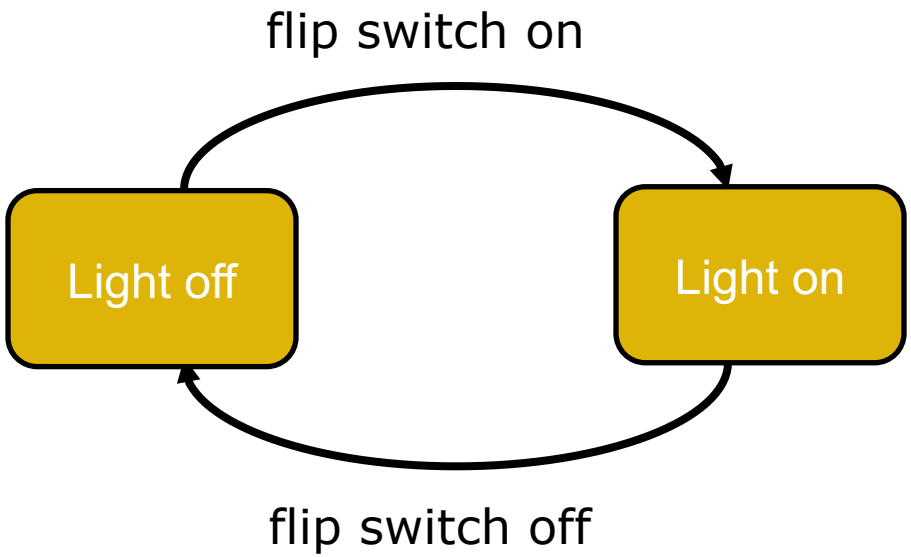
Dynamic Modelling (part 2) UML State Diagrams

CITS4401 Software Requirements and Design

Lecture 7a

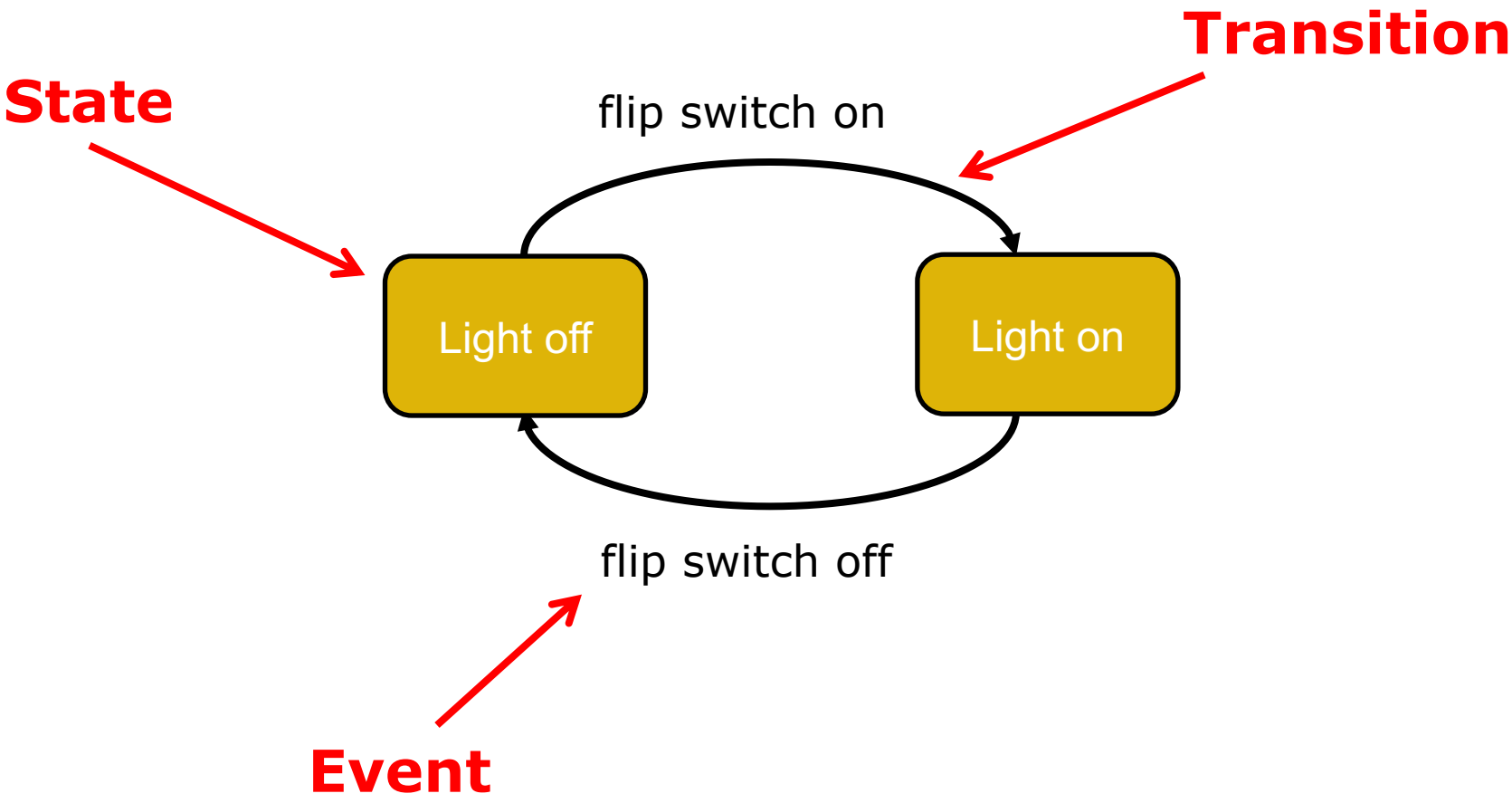
Finite state machine



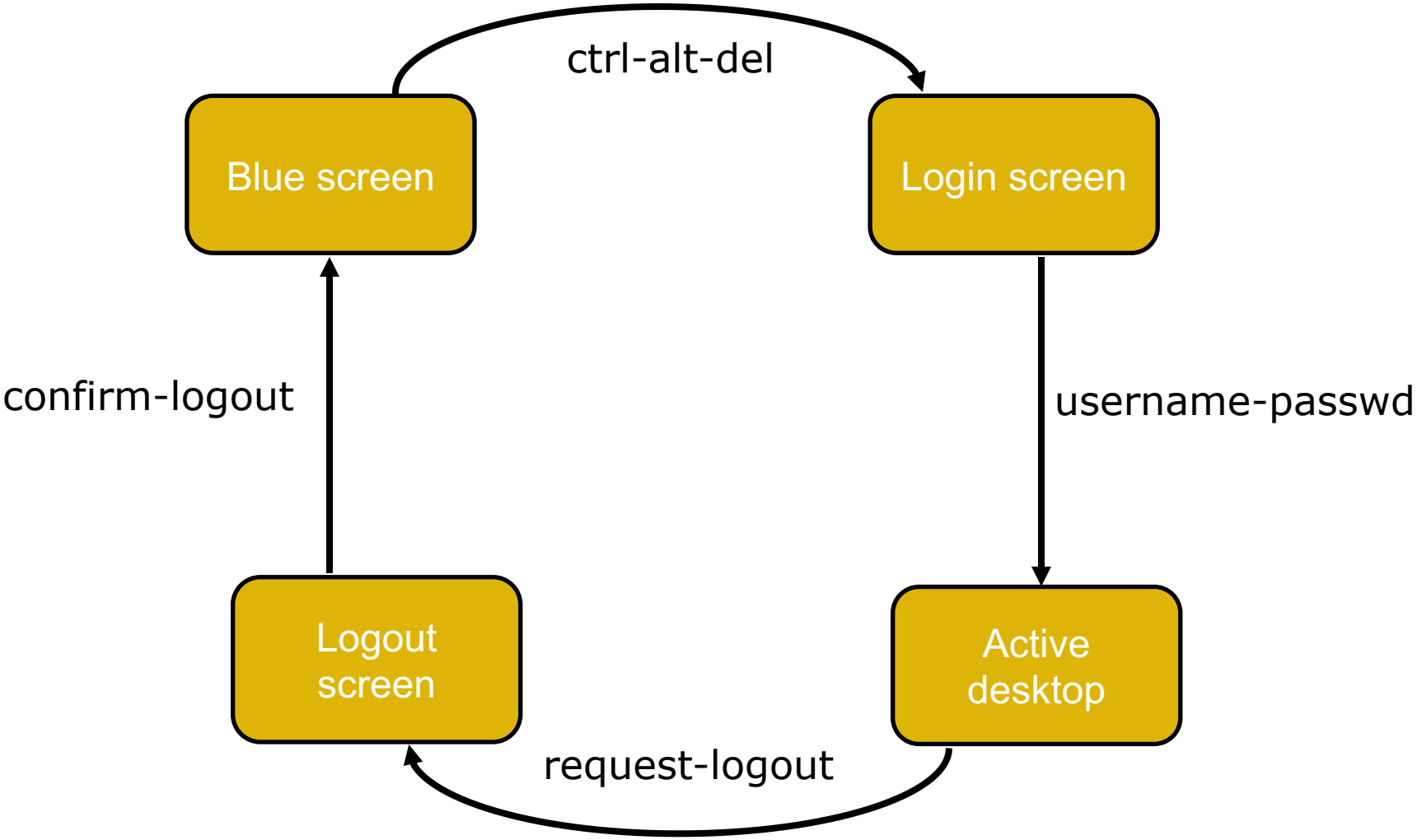
UML Statechart Diagrams

- A UML **state diagram** is a notation for describing the sequence of states an object goes through in response to external events.
- UML state machines are extensions of the finite state machine model.
- A **state** is a condition satisfied by the attributes of an object.
- A **transition** indicates a move from one state to another
- An **event** is a potential trigger for a change of state

State machine



Ex 1. Interaction with PC (V1)



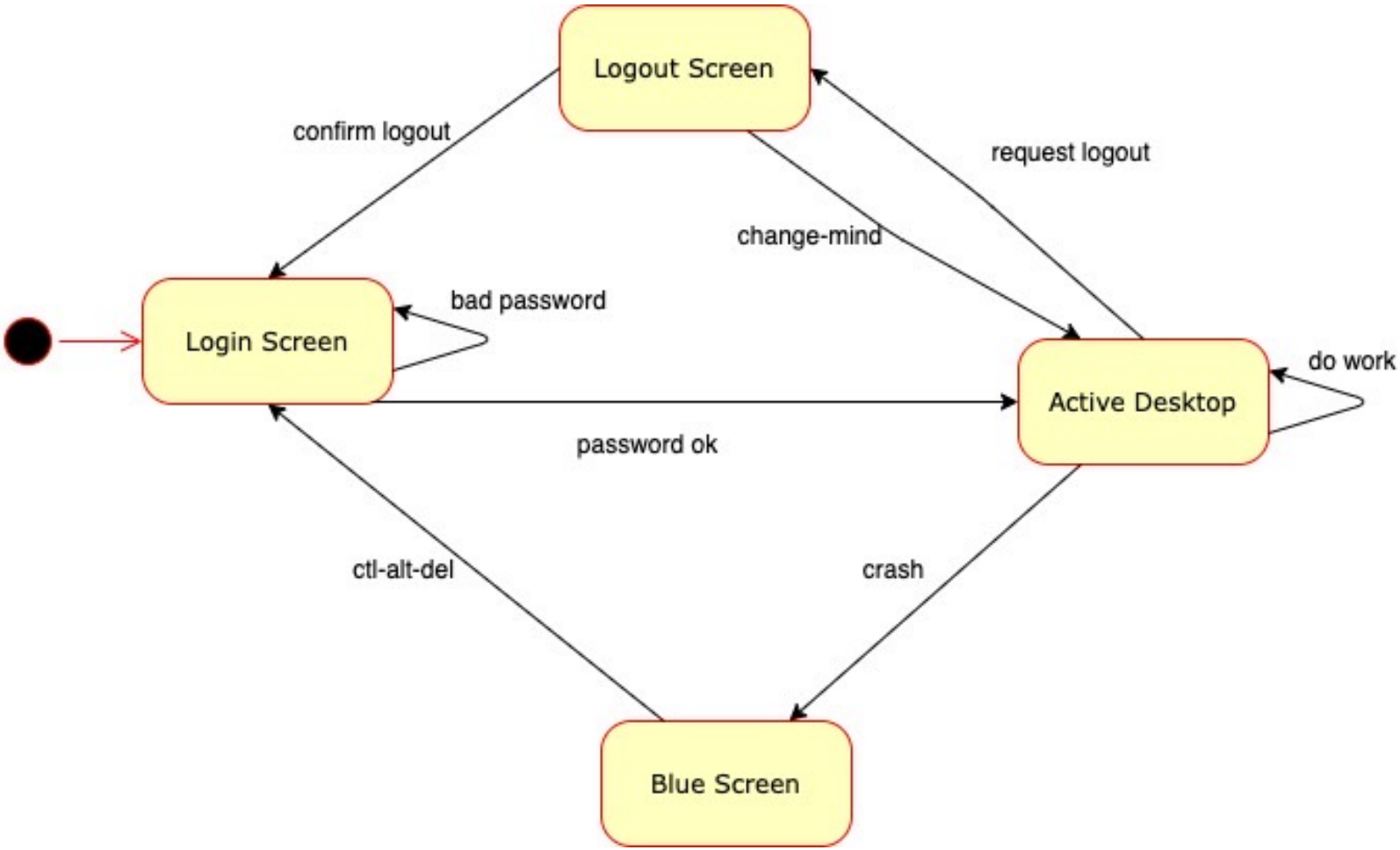
UML Statecharts

- **State machine models** show how individual objects change their state in response to events. They are represented in the UML using **statechart diagrams**

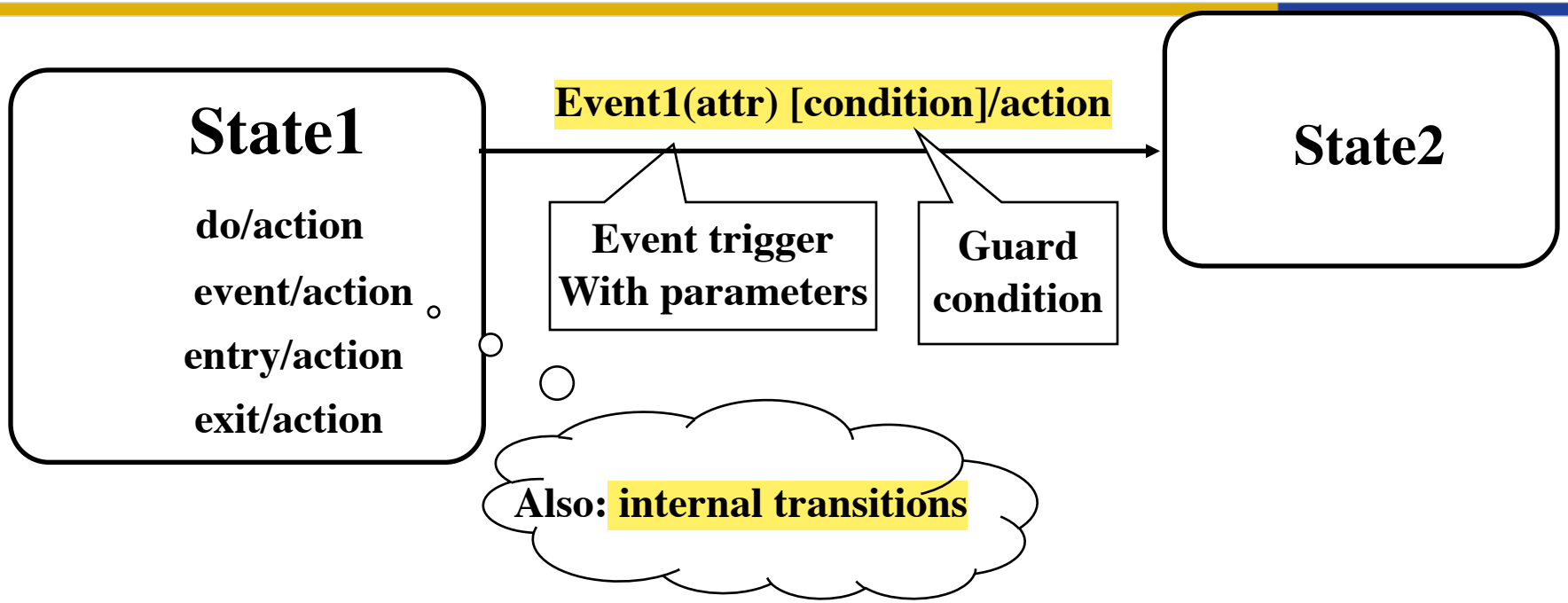
- While **sequence diagrams** are used to model the *combined behaviour* of a group of objects in the system, the **statechart diagrams** are used to model the behaviour of a *single object* in response to external events.

- In the context of behaviour modelling, 2 different characterizations of states must be considered:
  1. the state of each class, and
  2. the state of the system as observed from the outside as the system performs its function.

Interaction with PC (V2)



UML Statechart Diagram Notation



- UML notation based on statecharts by Harel in 1987
  - Added are a few object-oriented modifications
- A UML statechart diagram (or UML state machine) can be mapped into a finite state machine (FSM)

Statechart Diagrams

- Graphs whose nodes are **states** and whose directed arcs are transitions labelled by **events**.
- States capture conditions which hold for a period of time
  - e.g. light is on, light is off
- Events *change* the state (except internal)
  - e.g. turning the light on, turning the light off
- Statechart diagrams represent behaviour from the perspective of a single object only
  - An object model with a set of objects must be represented by a set of state diagrams

State

- An abstraction of the attributes of a class
  - State is the aggregation of several attributes of a class

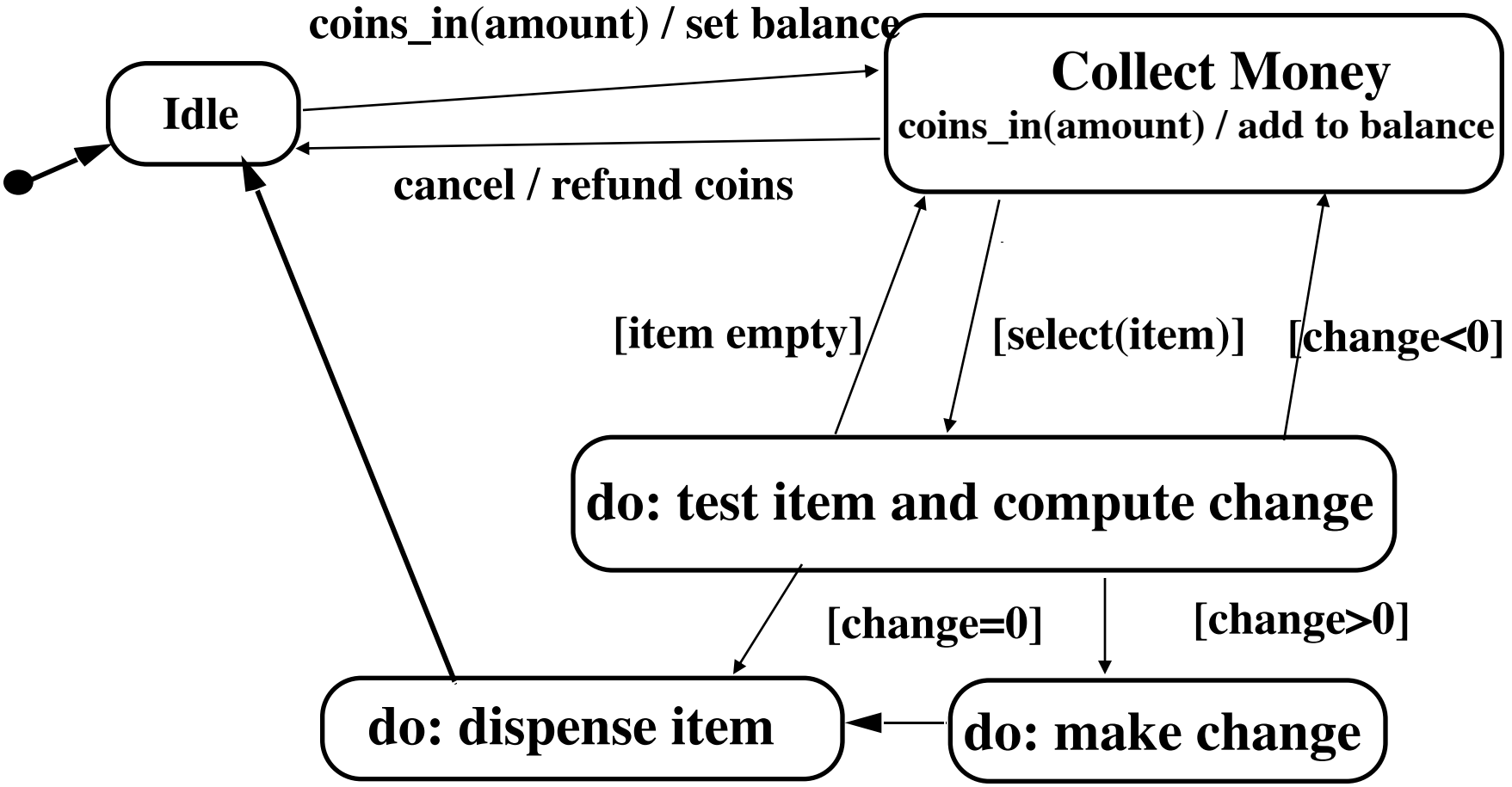
- Basically an equivalence class of all those attribute values and links that do not need to be distinguished as far as the control structure of the system is concerned
  - Example: State of a user interface screen
    - logged in, logged out, active, idle
    - active is an abstraction of all the user's logged in activity

- State has duration

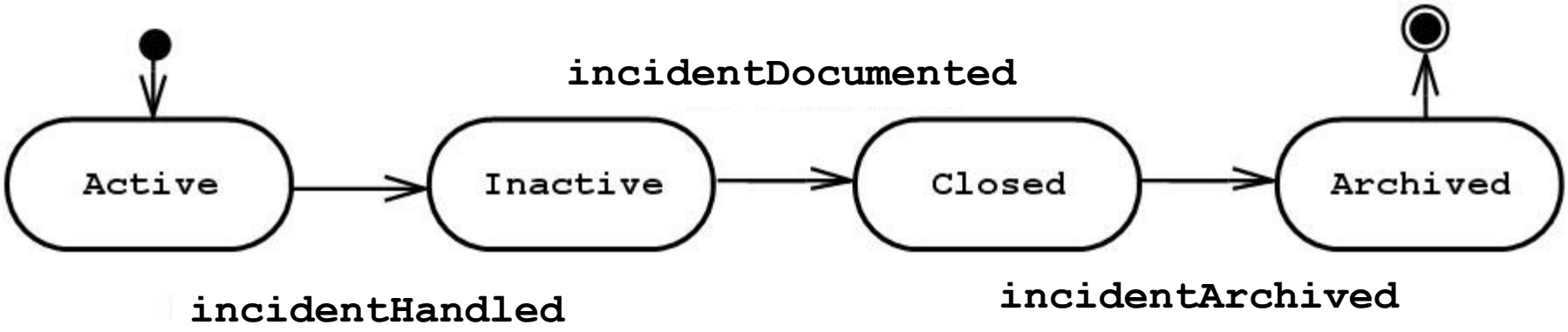
Event

- Something that happens at a point in time (e.g. button press, mouse click)
- Triggers a transition
  - Internal transition (no state change)
  - External transition (change to different state)
- May result in an action being executed

Example 2: vending machine



Another example



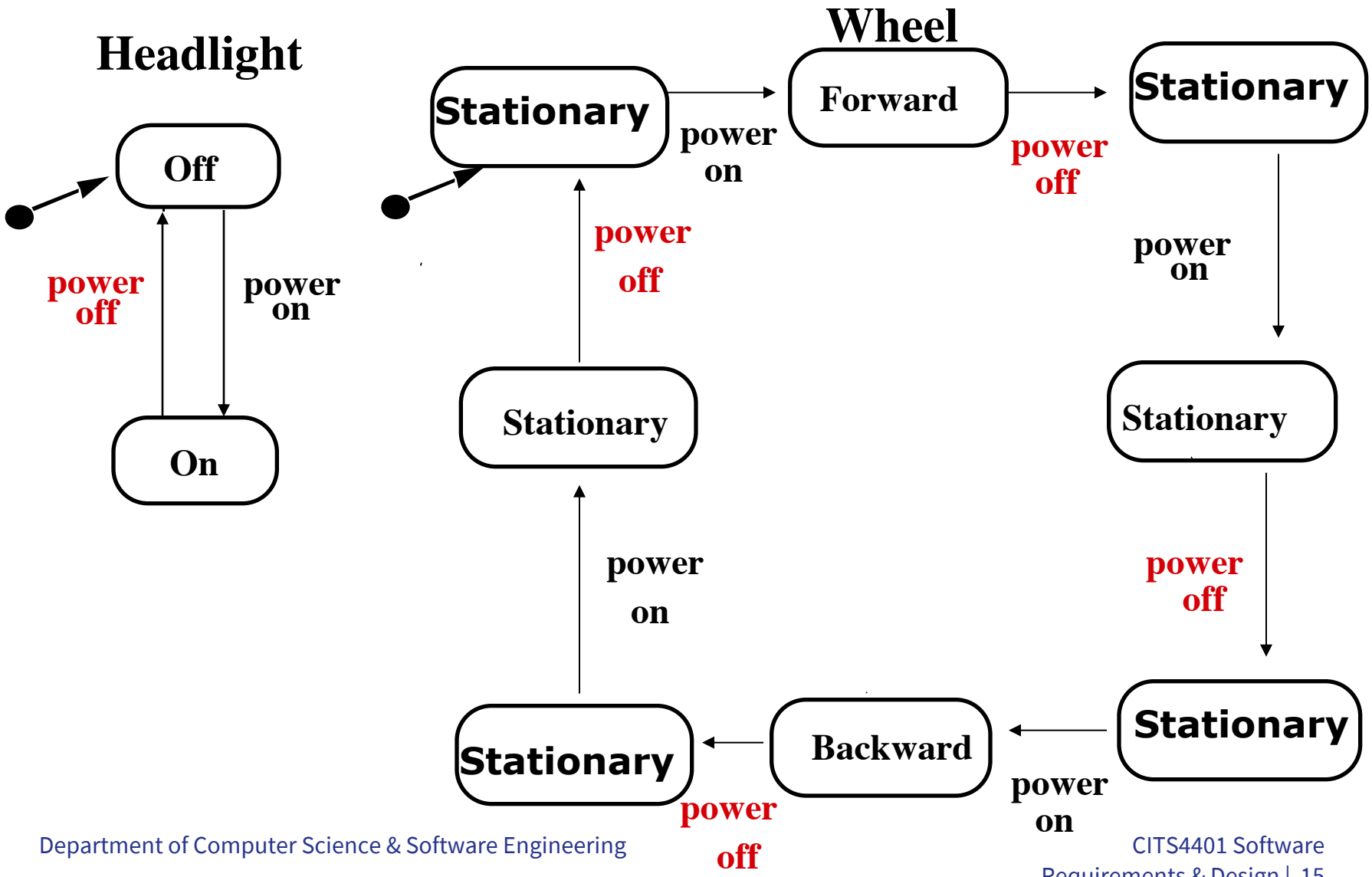
States of the *Incident* object of FRIEND

Problem Statement: Direction Control for a Toy Car

- Power is turned on
  - Car moves forward and car headlight shines
- Power is turned off
  - Car stops and headlight goes out.
- Power is turned on
  - Headlight shines
- Power is turned off
  - Headlight goes out.
- Power is turned on
  - Car runs backward with its headlight shining.

- Power is turned off
  - Car stops and headlight goes out.
- Power is turned on
  - Headlight shines
- Power is turned off
  - Headlight goes out.
- Power is turned on
  - Car runs forward with its headlight shining.

Toy Car: Dynamic Model



Practical Tips for Dynamic Modeling

- Construct dynamic models only for classes with significant dynamic behavior
  - Avoid “analysis paralysis”
- Consider only relevant attributes
  - Use abstraction if necessary
- Look at the granularity of the application when deciding on actions and activities
- Reduce notational clutter

State Chart Diagram vs Sequence Diagram

- State diagrams help to identify:
  - **Changes to objects** over time

- Sequence diagrams help to identify
  - The **temporal relationships between objects** over time
  - **Sequence of operations** as a response to one or more events

When to use state diagrams

[Fowler]

- State diagrams are good at describing the **behavior** of an object across several use cases.
- State diagrams are not very good at describing behavior that involves a number of objects collaborating.
- Therefore it is useful to combine state diagrams with other techniques.
- If you do use state diagrams, don't try to draw them for every class in the system. … it is almost always a waste of effort.
- Use state diagrams only for those classes that exhibit interesting behaviour, where building the state diagram helps you understand what is going on.
- Many people find that **User Interface and Control objects** have the kind of behaviour that is useful to depict with a state diagram.

Review (1): Requirements Analysis

Functional Modelling

- 1. What are the transformations?
  - Create *scenarios and use case diagrams*
    - Talk to client, observe, get historical records, do thought experiments

- 2. What is the structure of the system?
  - Create *object* and *class diagrams*
    - Identify objects. What are the associations between them? What is their multiplicity?
    - What are the attributes of the objects?
    - What operations are defined on the objects?

Object Modelling

Review (2): Requirements Analysis

- 3. What is its control structure?
  - Create *state diagrams*
    - Only for the dynamically interesting objects.

Dynamic Modelling

  - Create *sequence diagrams*
    - Identify senders and receivers of events
    - Show sequence of events exchanged between objects. Identify event dependencies and event concurrency.

When is a model dominant?

- ***Functional model***
  The model performs complicated transformations such as difficult computations consisting of many steps.

- ***Object model***
  The system has non-trivial data structures.

- ***Dynamic model***
  The model has many different types of events: Input, output, exceptions, errors, etc.

When is a model dominant? Examples

- *Compiler:* Functional model most important. Dynamic model is trivial because there is only one type input and only a few outputs.

- *Database systems:* Object model most important. Functional model is trivial, because their purpose is usually only to store, organize and retrieve data.

- *Spreadsheet program:* Functional model most important. Object model is trivial, because the spreadsheet values are trivial and cannot be structured further. The only interesting object is the cell. Dynamic model is also relatively important.

Recommended reading

UML Distilled by Martin Fowler

Chapter 10 State Machine Diagrams

Object oriented software engineering by Bruegge & Dutoit

Section 5.4 Analysis Activities from Use Cases to Objects

Software Engineering by Pressman (different editions)

Chapter: Requirements Modelling

Section: Creating a Behavioural Model