Introduction: Formalisms for system design

- Object-oriented design.
  - Design is composed of interacting objects
  - Objects: S/W or H/W
- Unified Modeling Language (UML).
  - Visual language to conceptualize design

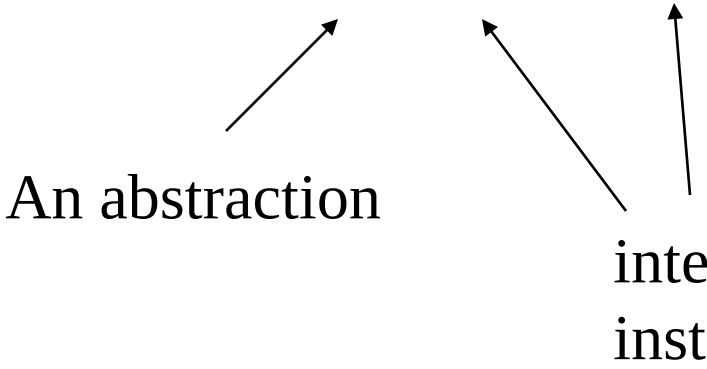
Taken/modified from "Computers as Components, W. Wolf"

System modeling/design

- Need languages to describe systems:
  - useful across several levels of abstraction;
  - understandable within and between organizations.
- Block diagrams are a start, but don’t cover everything.

Object-oriented design

- Object-oriented (OO) design/programming: Design using objects/programming using C++, Java, C, ...
- Class and object: e.g., int a, b, c;



Taken/modified from "Computers as Components, W. Wolf"

Classes and objects

- **Class: Abstraction of the common properties from a set of objects, e.g., AUTOMOBILE, ROBOT, ...**

- **Object = state + methods.**
  - **State provides each object with its own identity.**
  - **Methods provide an abstract interface to the object.**

OO implementation in C++

```
class display {
  pixeltype pixels[IMAX,JMAX];
public:
  display() { }
  pixeltype pixel(int i, int j) {
  return pixels[i,j]; }
  void set_pixel(pixeltype val, int i,
  int j) { pixels[i,j] = val; }
}
display D1; --- D1.pixel(0,0);
```

OO implementation in C

typedef unsigned char pixeltype;

```
typedef struct { pixeltype
  pixels[IMAX,JMAX]; } display;
display d1;
pixeltype pixelval(display *px, int i, int
  j) {
      return px->pixels[i,j];
      }
void set_pixel(display *px, pixeltype val,
  int i, int j){ px->pixels[i,j] = val;}
```

Taken/modified from "Computers as Components, W. Wolf"

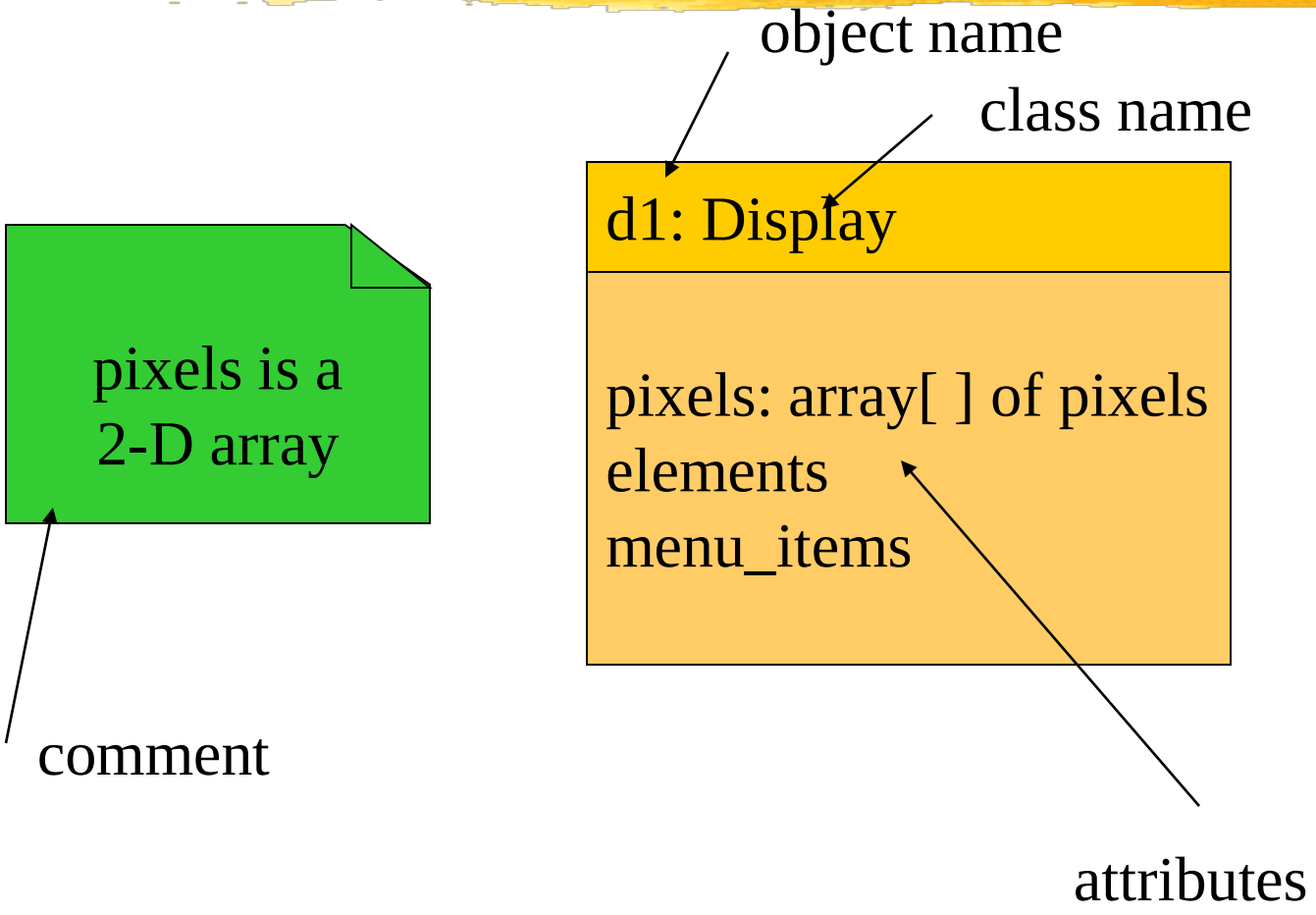
Recap: Objects and classes

- Class: object type → Mechanism to create objects
- Data: Class defines the object's state elements but state values may change over time.
- Function: Class defines the methods used to interact with all objects of that type.
  - Each object has its own state.

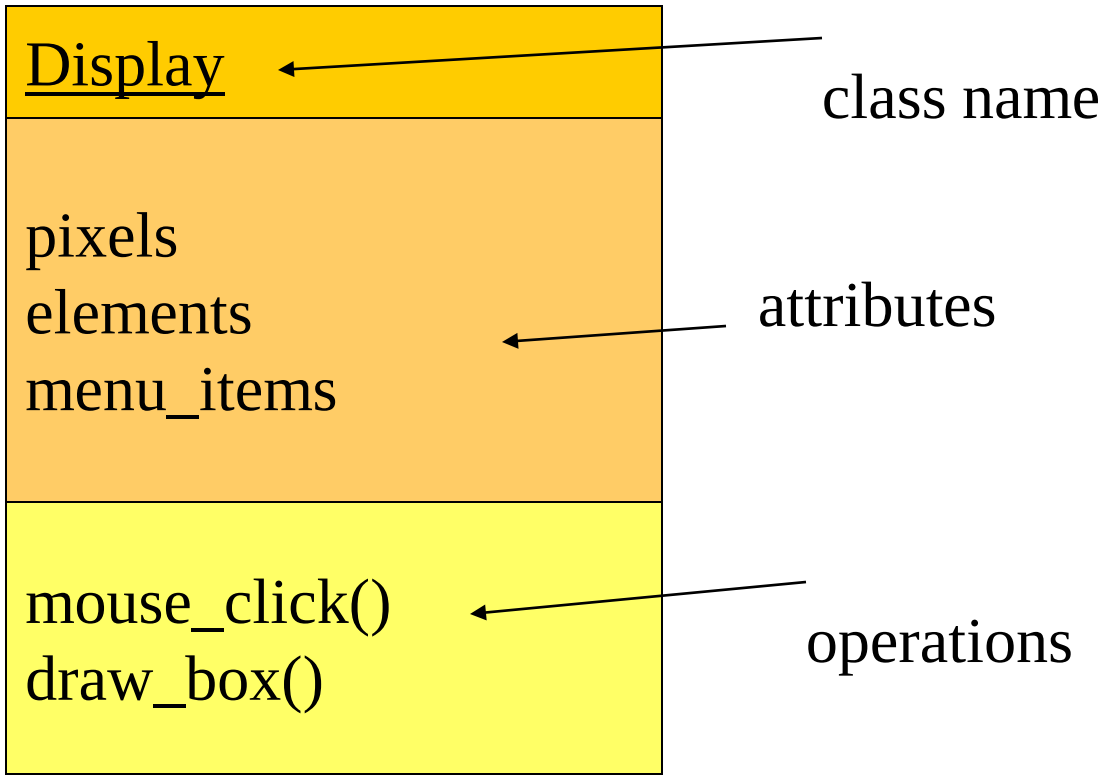
UML: Visual modeling language

- Developed by Grady Booch et al.
- Goals:
  - object-oriented framework
  - visual
  - usable for modeling complex systems
  - Behavioral modeling with statecharts
  - Support for multi-tasking representation

UML object


object name
class name
d1: Display
pixels: array[ ] of pixels
elements
menu_items
pixels is a
2-D array
comment
attributes

UML class



Taken/modified from "Computers as Components, W. Wolf"

The class interface:

State + Operation

- Operations (functions): Provide the abstract interface between the class implementation and other classes.
- Operations: May have arguments, return values.
- An operation can examine and/or modify the object's state.

Choose your objects properly: Good cohesion

- **e.g., robot object: joint angles, gripper distance, motion commands**
- **Reusability: robot object, pid object, ...**

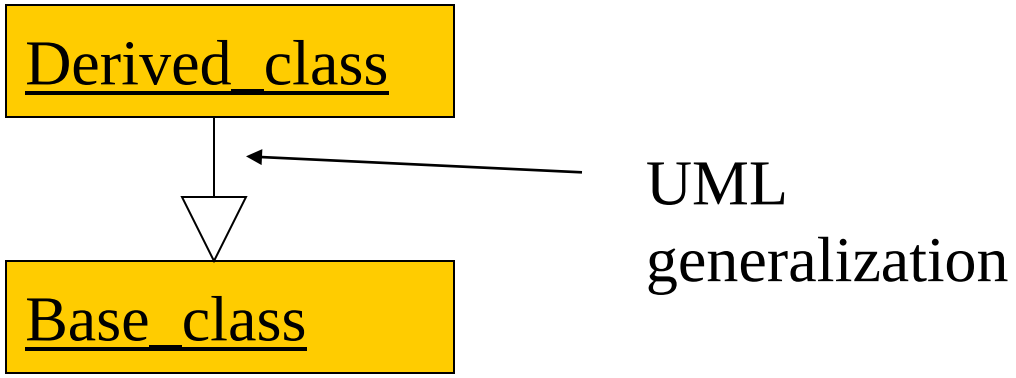
Robot objects

Relationships between objects/classes

- **Association**: objects communicate with each other → flight controller **controls** engine
- **Aggregation:** a complex object is made of several smaller objects→ multimedia: A/V, Robot object **contains** gripper (object)
- **Composition:** Strong aggregation in which owner does not allow access to its components: Engine **has** a piston, sensor **has** A/D
- **Generalization:** define one class in terms of another. → is-a-kind-of relationship, e.g. position sensor **is a kind of** sensor

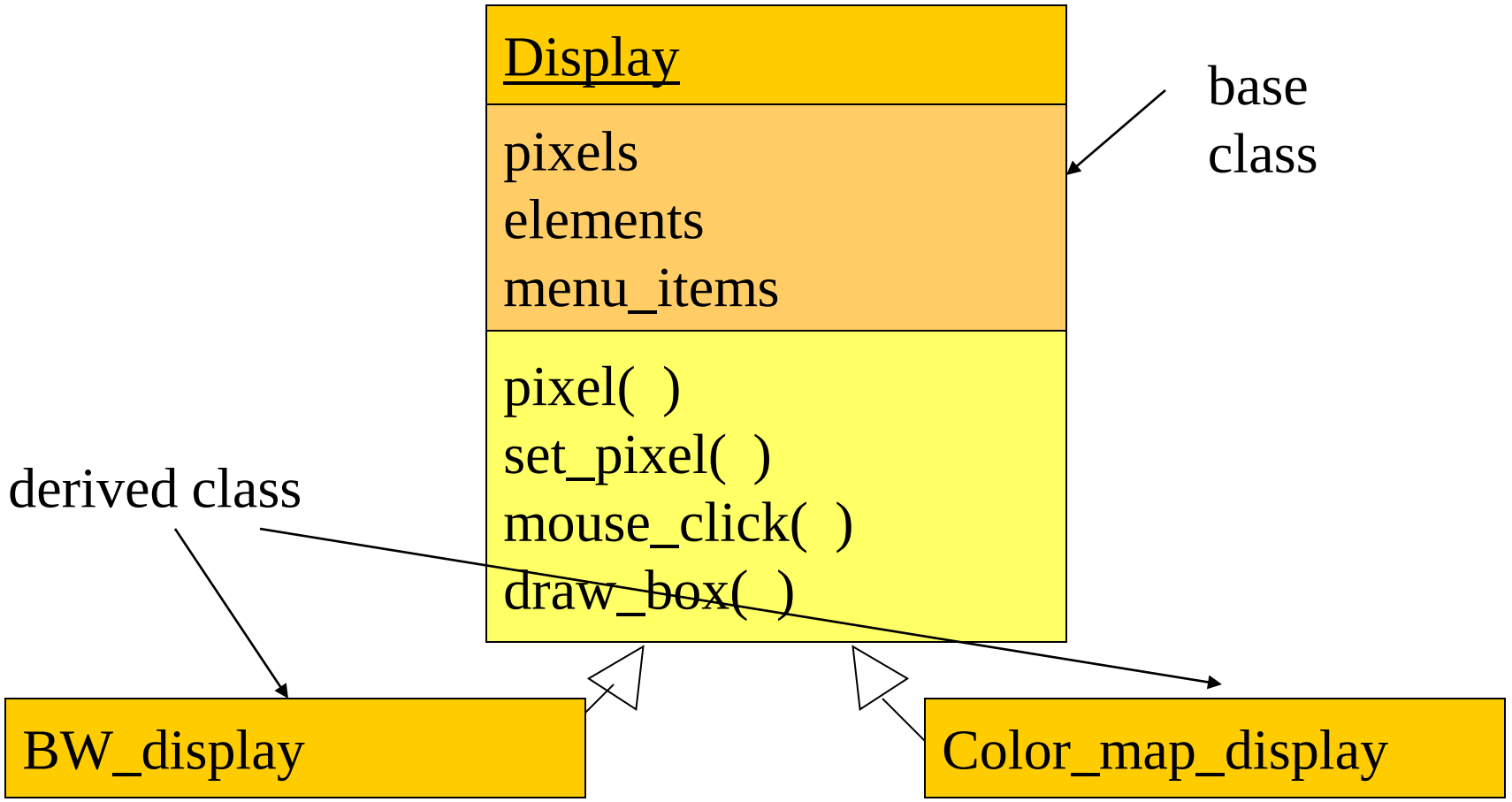
Class derivation

- May want to define one class in terms of another.
  - Derived class inherits attributes, operations of base class, e.g., robot on a track

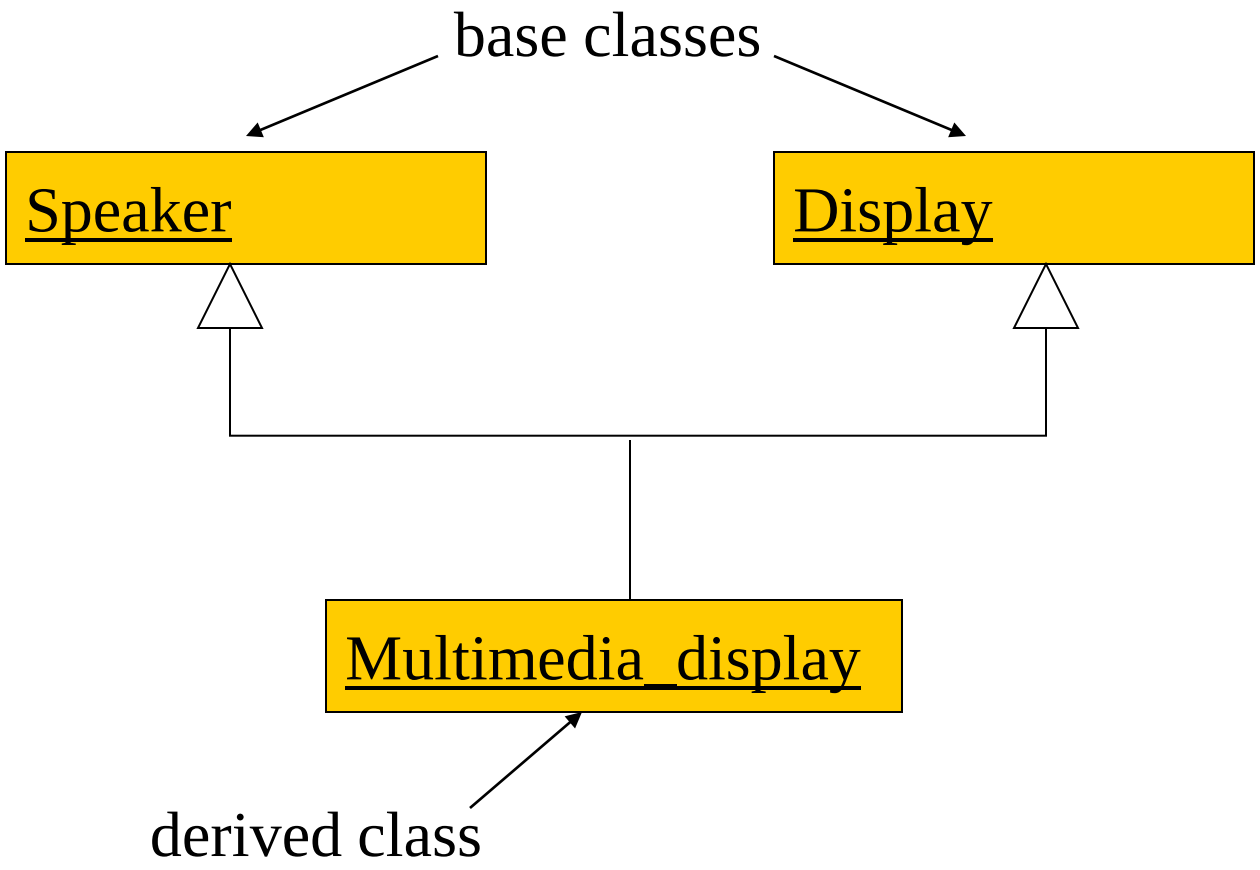


Taken/modified from "Computers as Components, W. Wolf"

Class derivation example



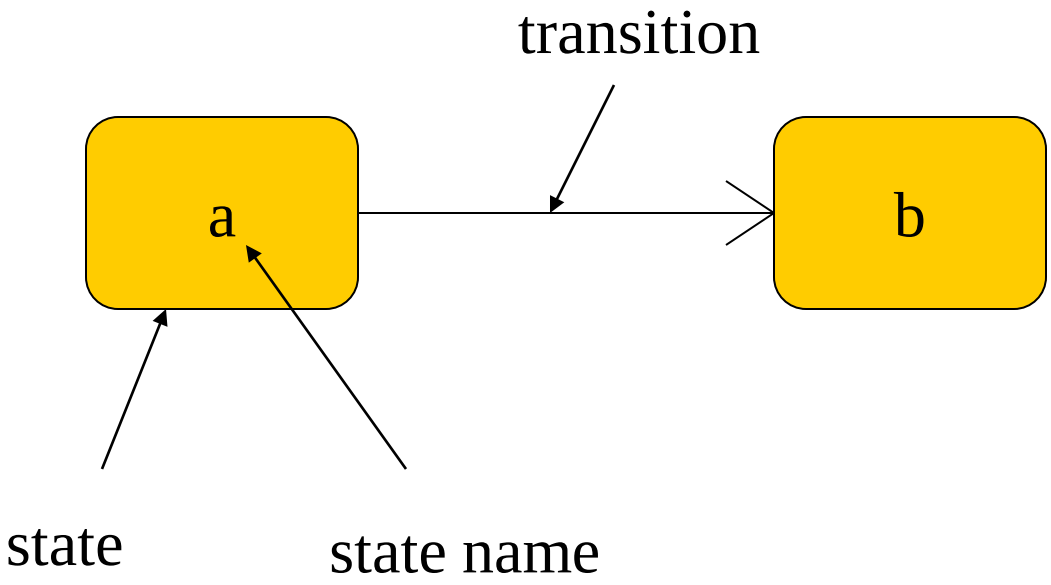
Multiple inheritance



Behavioral description

- State machines: One way to describe behavior
- State machines work based on triggering of events

State machines



Taken/modified from "Computers as Components, W. Wolf"

Event-driven state machines

- Behavioral descriptions are written as event-driven state machines.
  - Machine changes state when receiving an input.

- An event may come from inside or outside of the system.

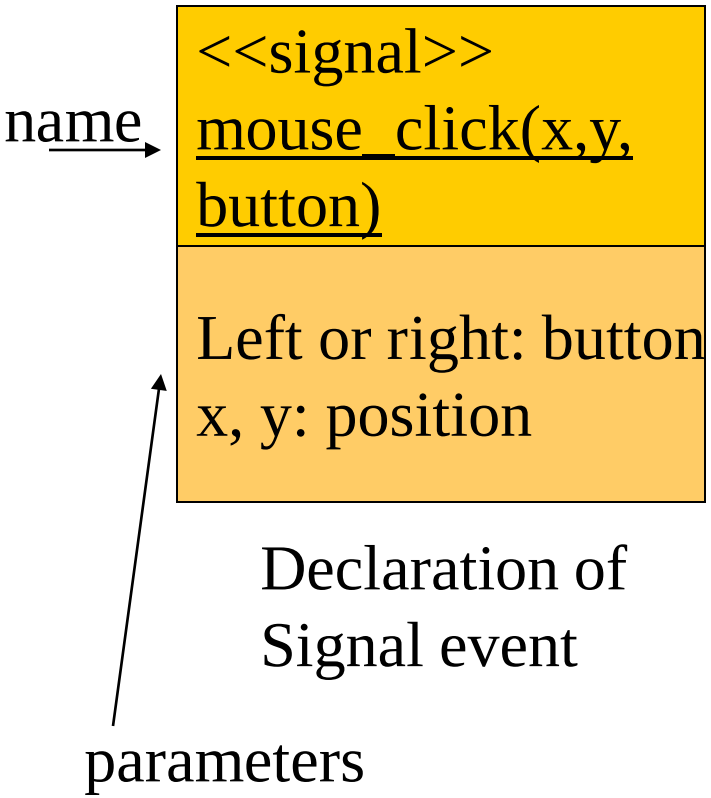
Types of events

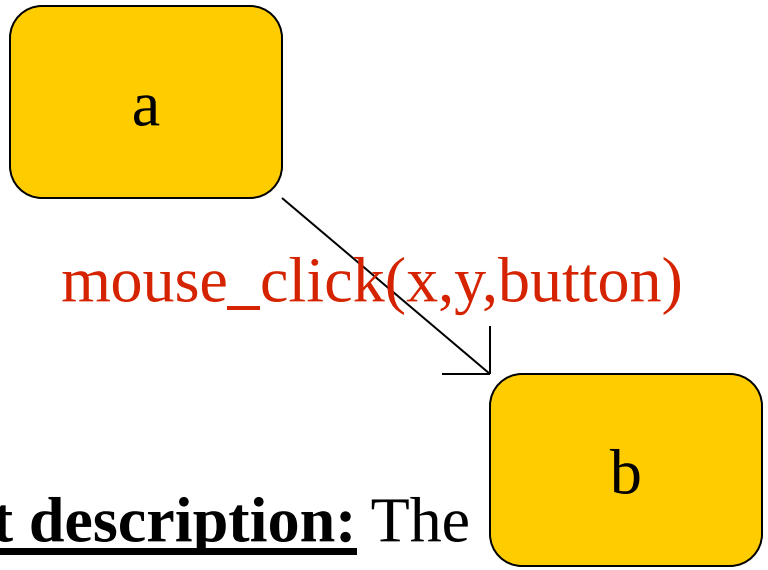
- Signal: asynchronous event.
- Call: synchronized communication.
- Timer: activated by time.

Signal event:

Asynchronous occurrence of an event

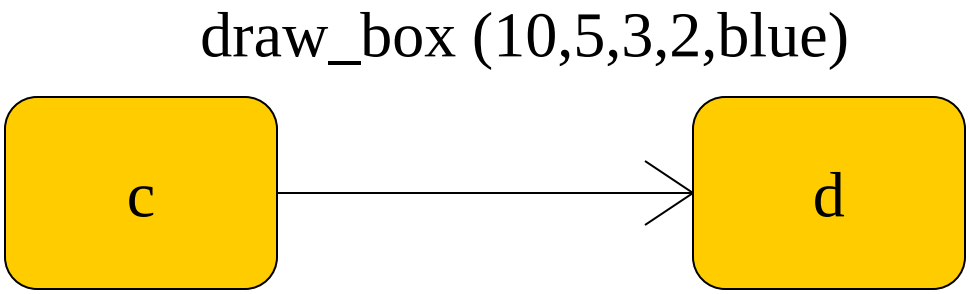
Signal event is defined in UML by an object that is labeled as a signal



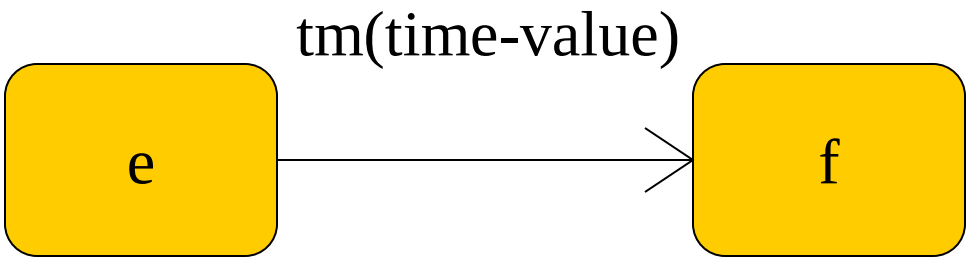


Event description: The occurrence of a signal is shown by mouse_click( )

Call event

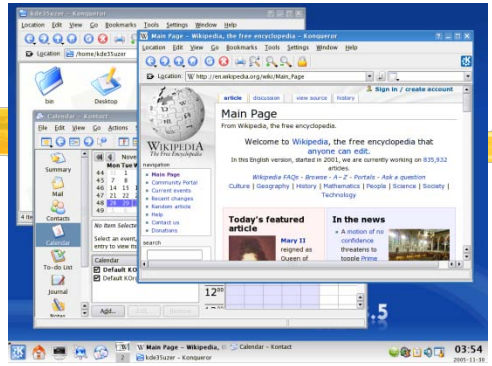


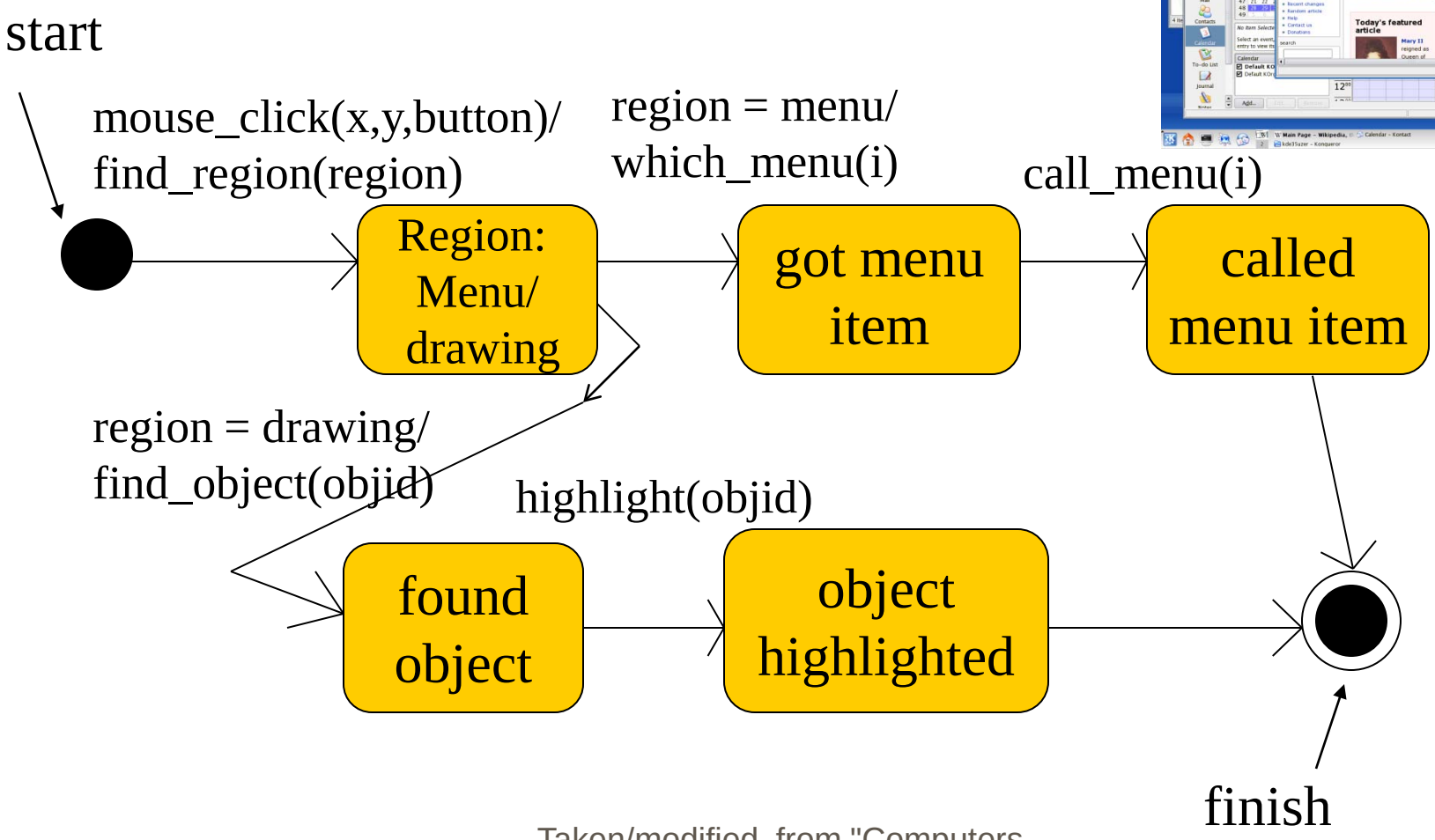
Timer event



Example state machine:

Display with menu bars





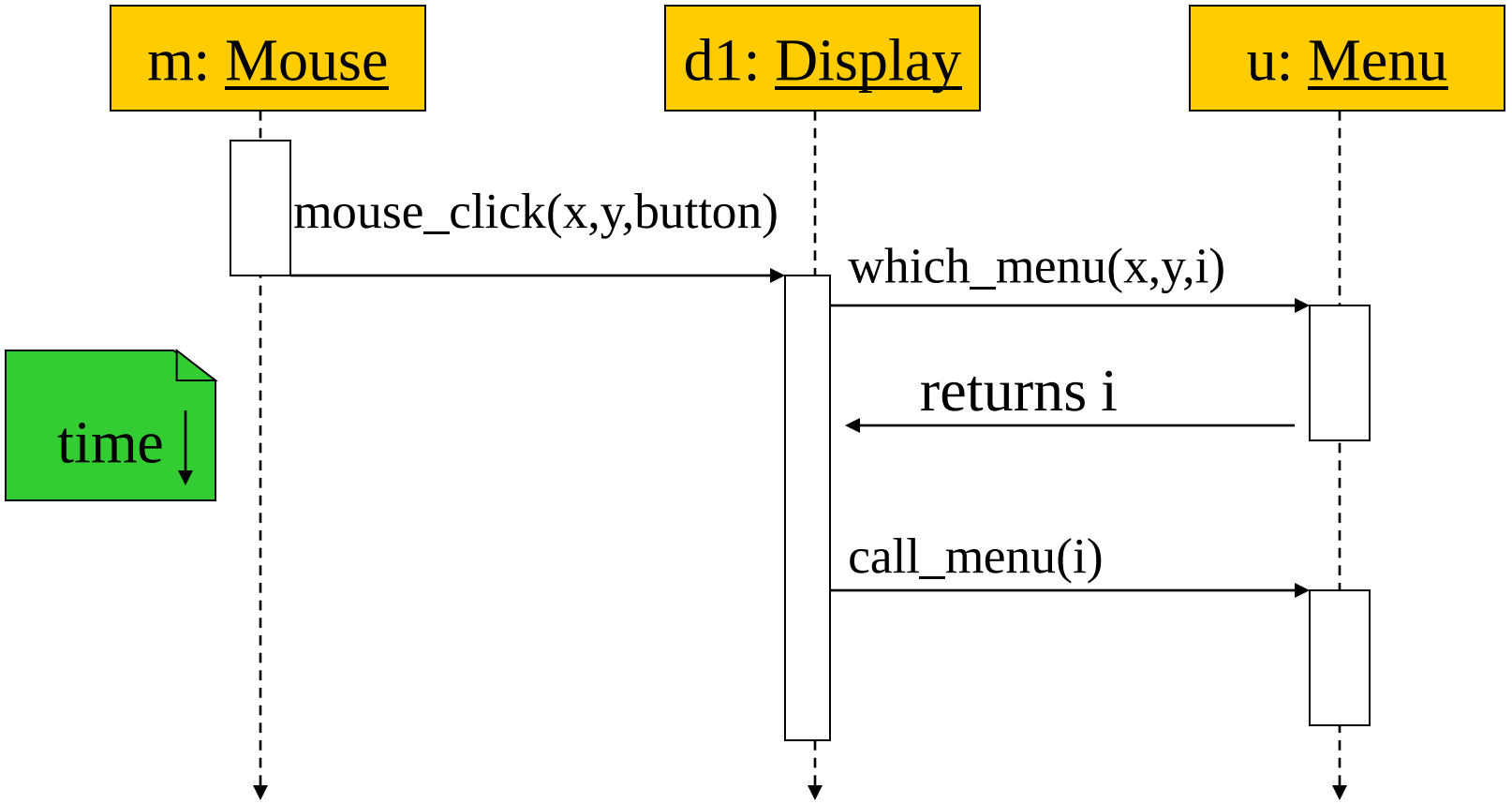
Taken/modified from "Computers as Components, W. Wolf"

Sequence diagram

- Shows sequence of operations over time.
- Relates behaviors of multiple objects.

Sequence diagram example:

Mouse click on the menu region



Mouse, Display, and Menu are objects

Summary

- Object-oriented design (or object-based design) helps us organize a design.
- UML is a transportable system design language.
  - Provides structural and behavioral description primitives.