Application architecture

System architecture

- System structuring:
  - Repository
  - Client-server
  - Layered
- Control:
  - Centralized
    - Call-return
    - Manager
  - Event-based
    - Broadcast
    - Interrupt-driven

Application perspective

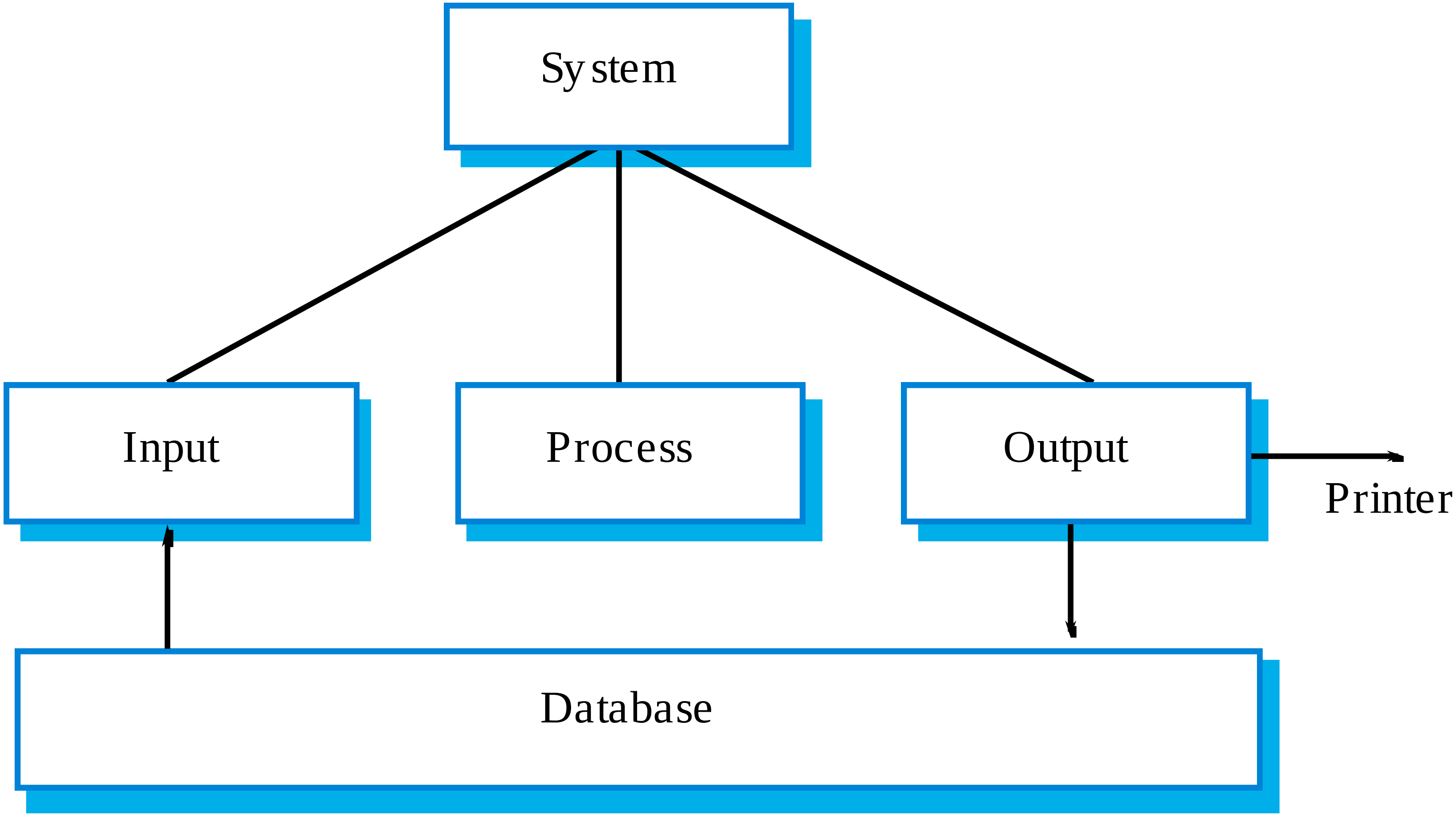
- Generic types of applications
  1. Data-processing
  2. Transaction-processing
  3. Event-processing
  4. Language-processing

Data-processing systems

Data-processing systems

- Systems that are data-centered
- No or reduced user intervention
  - Examples: payroll, billing, accounting
- The databases are usually orders of magnitude larger than the software itself
- Data is input and output in batches
  - Input: A set of customer numbers and associated readings of an electricity meter;
  - Output: A corresponding set of bills, one for each customer number.
- Usually have an *input-process-output* structure.

Data processing applications


System

Input

Process

Output

Printer

Database

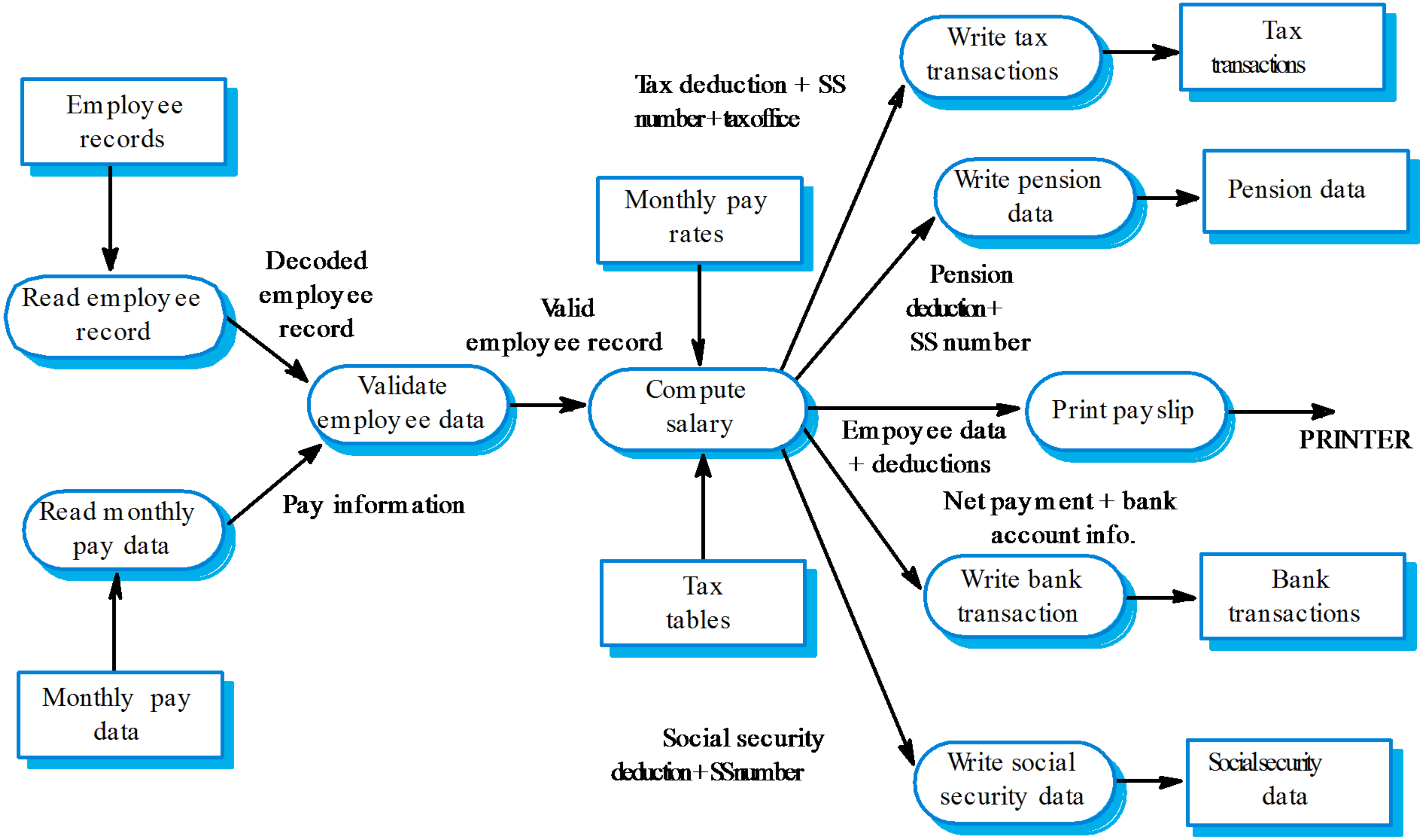
Input-Process-Output

- Input:
  - reads data from a file or database,
  - checks its validity
  - queues the valid data for processing.
- Process
  - takes a transaction from the queue (input),
  - performs computations
  - creates a new record with the results of the computation.
- Output
  - reads these records,
  - formats them accordingly
  - writes them to the database or sends them to a printer

Representation

- Records are processed serially
- No need to store state information
  - ➢Function-oriented systems (rather than object-oriented)
  - ➢Data-flow diagrams are suitable models
- Show data as it moves through the system
- Show end-to-end processing
  - – All functions that act on data are visible

Example: data-flow for payroll



Transaction-processing systems

Transaction-processing systems

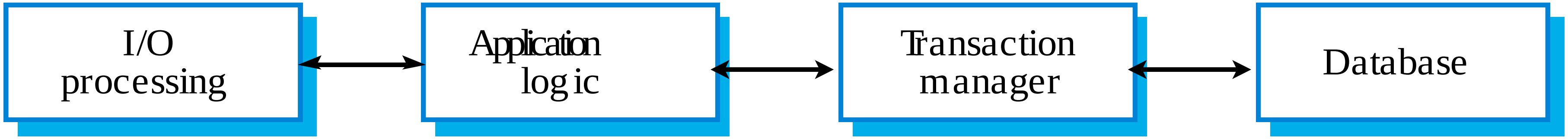
- Database-centered
- Process user requests
- Update information in a system database.
- Examples:
  - interactive banking,
  - e-commerce,
  - booking systems,
  - information systems

Transaction-processing systems

- Process
  - requests for information from a database
  - requests to update a database.
- From a user perspective a transaction is:

  Any coherent sequence of operations that satisfies a goal
- The requests are asynchronous
- They are processed by a transaction manager.

Structure of TP Apps



I/O processing

Application logic

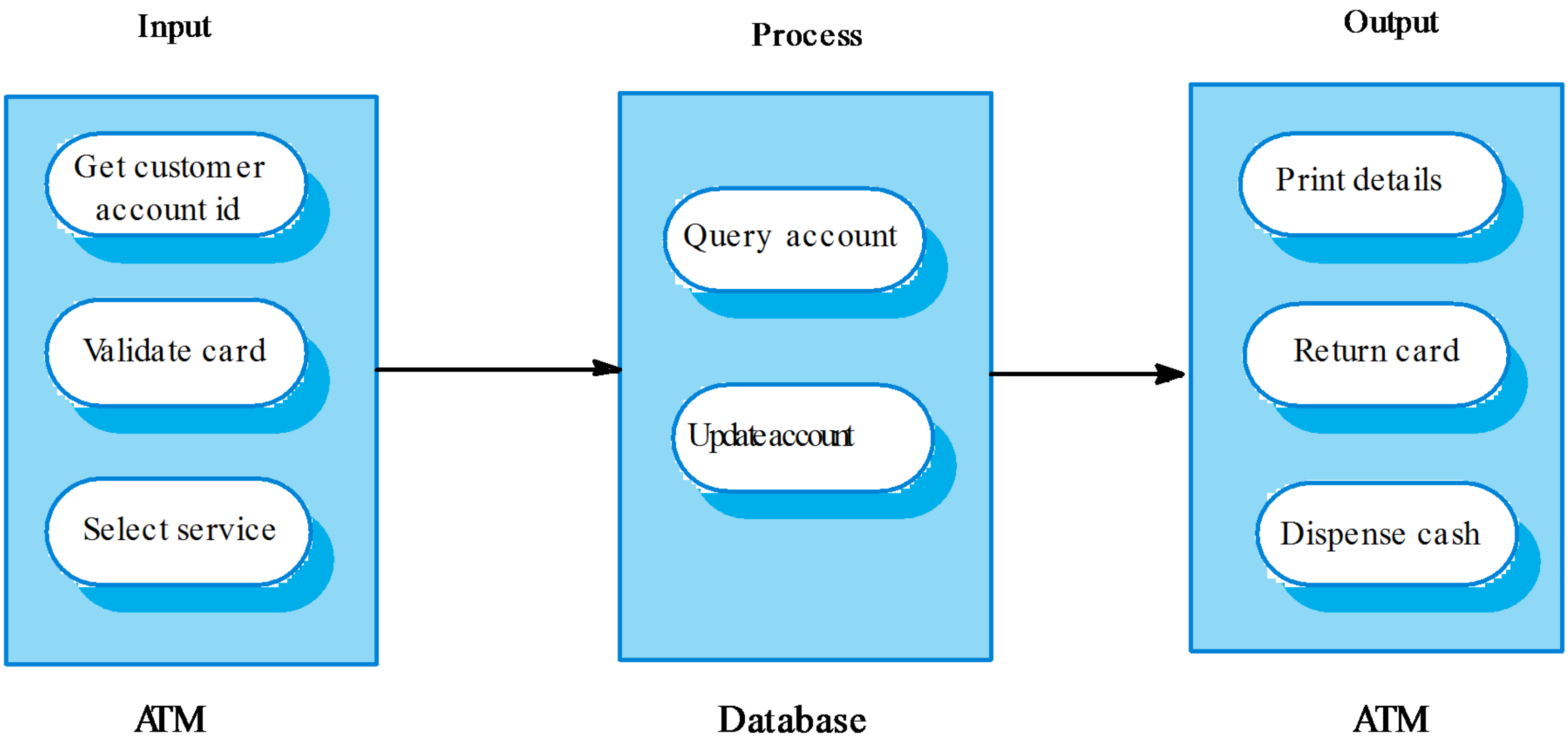
Transaction manager

Database

Transactions

- are defined from the database point of view
  - a transaction is a set of operations treated as a single unit (atomic)
  - all operations in a transactions must be completed before changes in the database are made permanent
  - *failure of operations within a transaction should not lead to database inconsistencies*

Example: cash dispenser



Specifics of TP applications

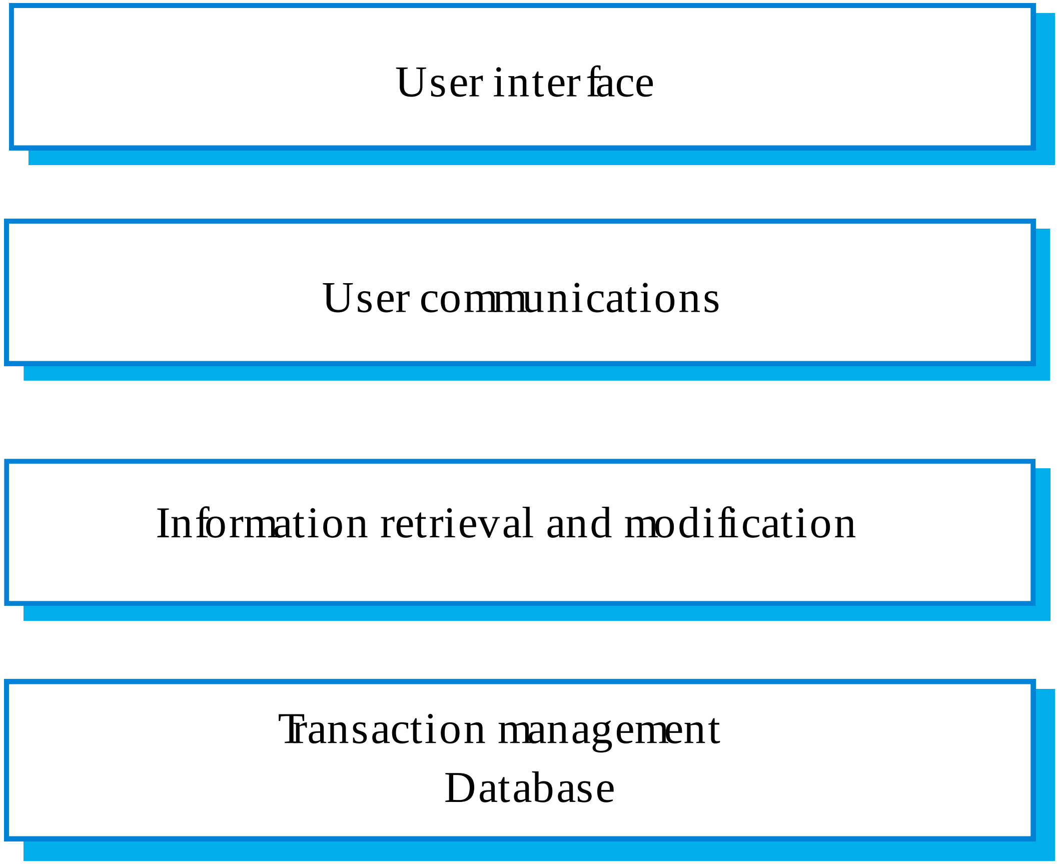
- Highly distributed
- Many types of terminals that interact with users
  - ➢ May include middleware:
    - ➢ infrastructure software that help manage interactions between distributed entities and system database
  - ➢ Transaction management middleware :
    - handle communications with different terminal types
    - serializes data
    - sends data for processing

Typical examples

- Information management systems
- Resource management systems

Information management systems

- An *information system* allows controlled access to a large base of information



User interface

User communications

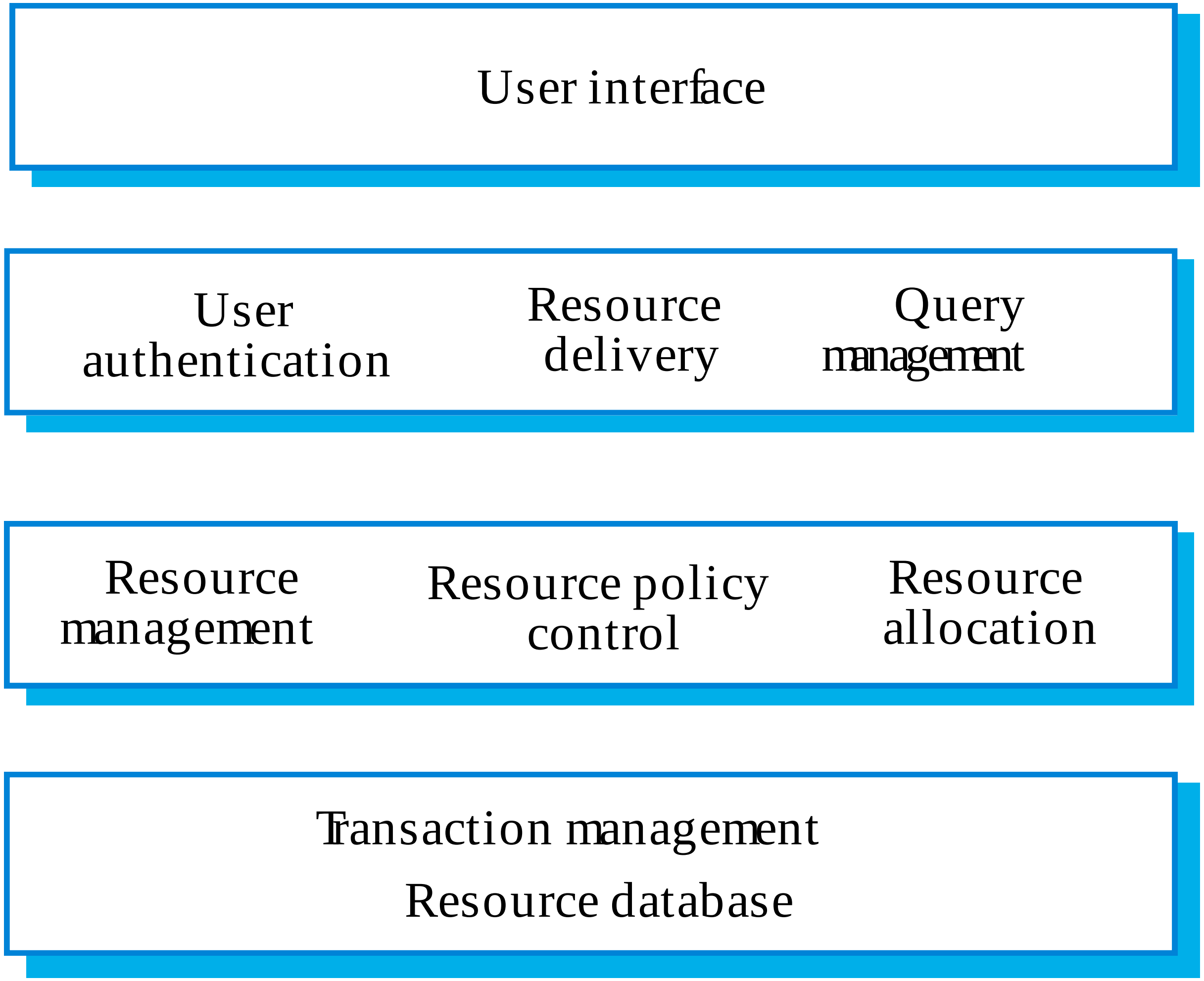
Information retrieval and modification

Transaction management
Database

Resource management systems

- Manage a limited amount of some resources
- The resources are allocated to users who requests them
- Examples:
  - Ticketing systems
  - Timetabling systems (the resource is a time period)
  - Library systems
  - Air traffic management systems (the resource is a segment of airspace)

Resource allocation system model



User interface

User authentication

Resource delivery

Query management

Resource management

Resource policy control

Resource allocation

Transaction management

Resource database

Event-processing systems

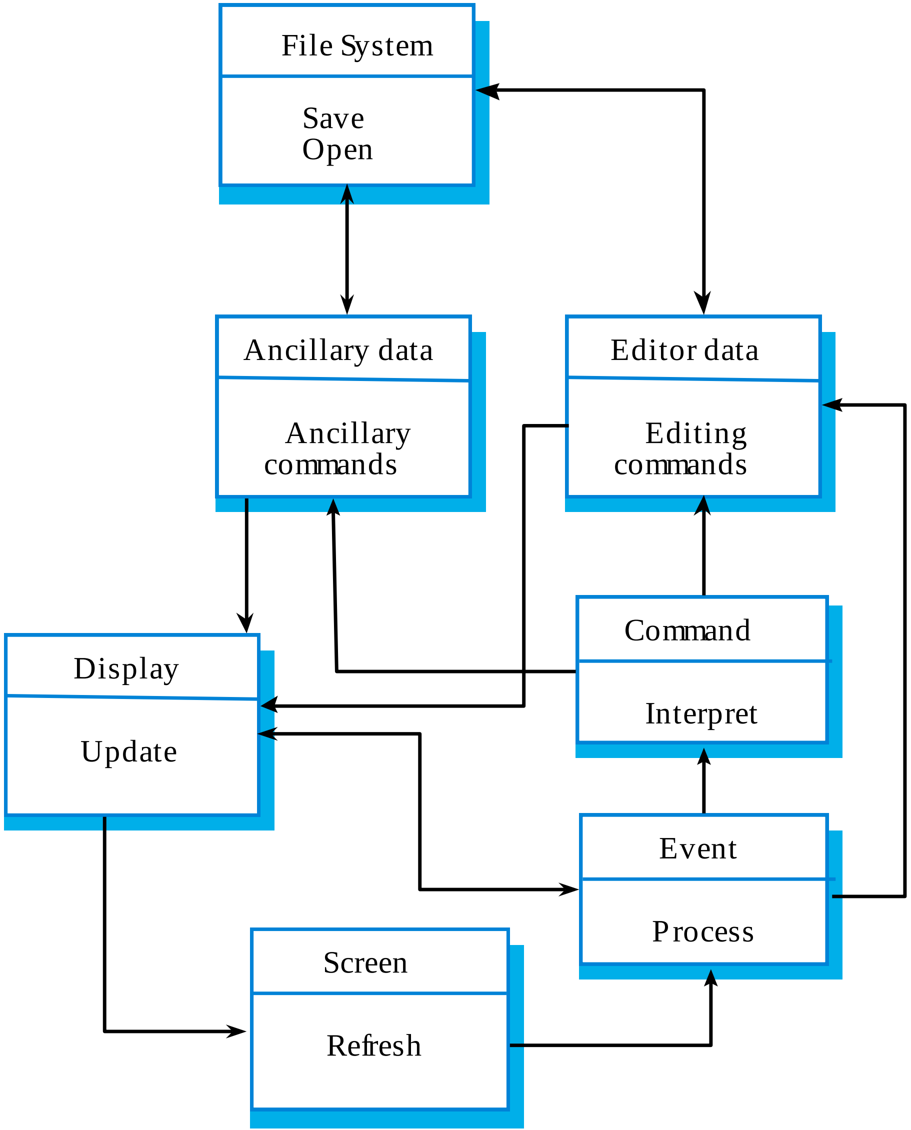
Event-processing systems

- respond to events in the system’s environment
- key characteristic:
  - *event timing is unpredictable,*
  - *the architecture has to be organized to handle this.*
- common systems:
  - word processors,
  - games, etc.

Typical event-processing systems

- Real-time systems
- Editing systems
  - Single user systems;
  - Must provide rapid feedback to user actions;
  - Organized around long transactions so may include recovery facilities

Editing systems architecture

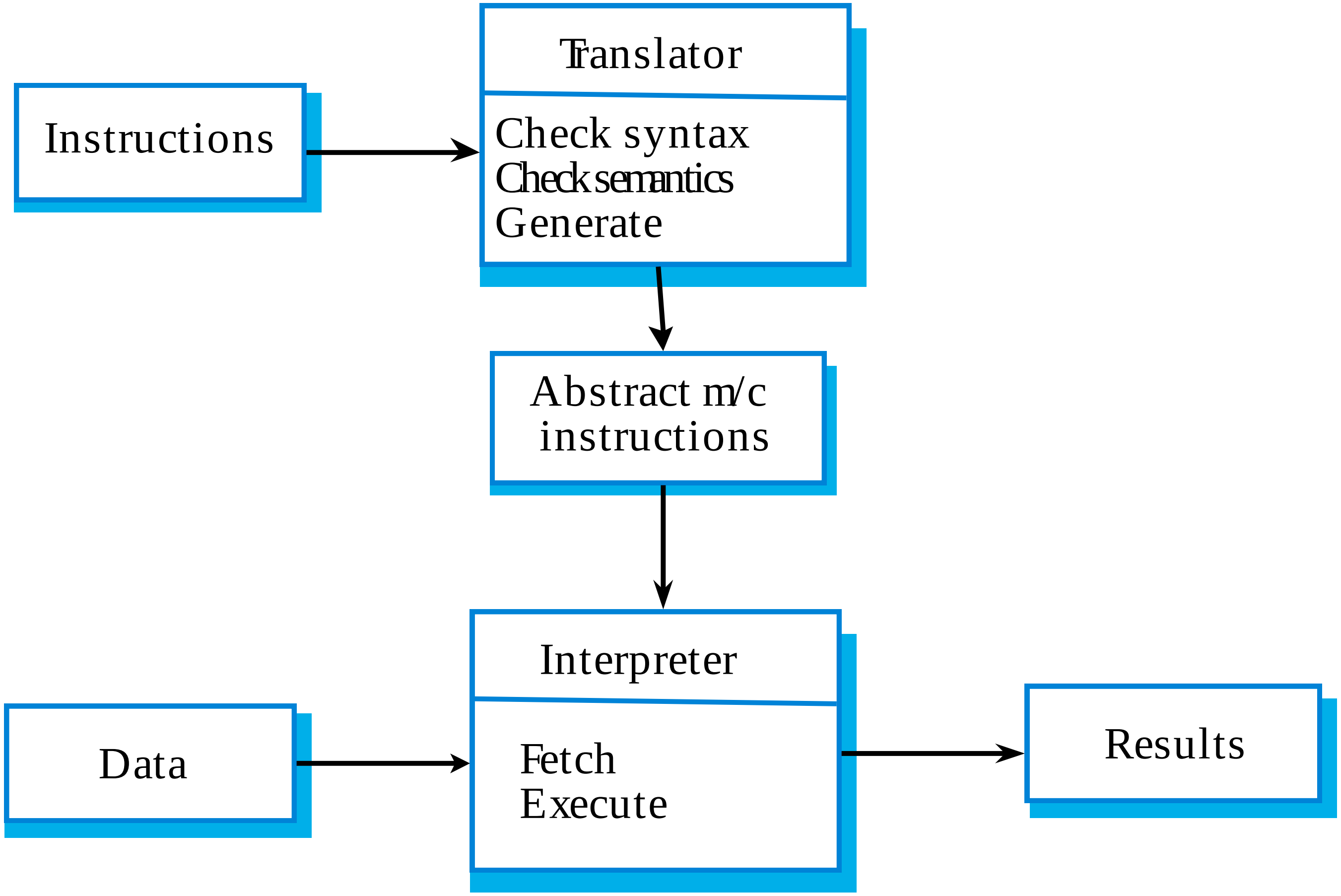


Language processing systems

Language processing systems

- Accept a natural / artificial language as input
- Generate some other representation of that language
- [May include an interpreter to act on the instructions in the language that is being processed]
- Used in situations where the easiest way to solve a problem is to *describe an algorithm or describe the system data*
  - Meta-case tools process tool descriptions, method rules, etc. and generate tools.

Interpreters: Generic architecture


Translator

Check syntax
Check semantics
Generate

Instructions

Abstract m/c
instructions

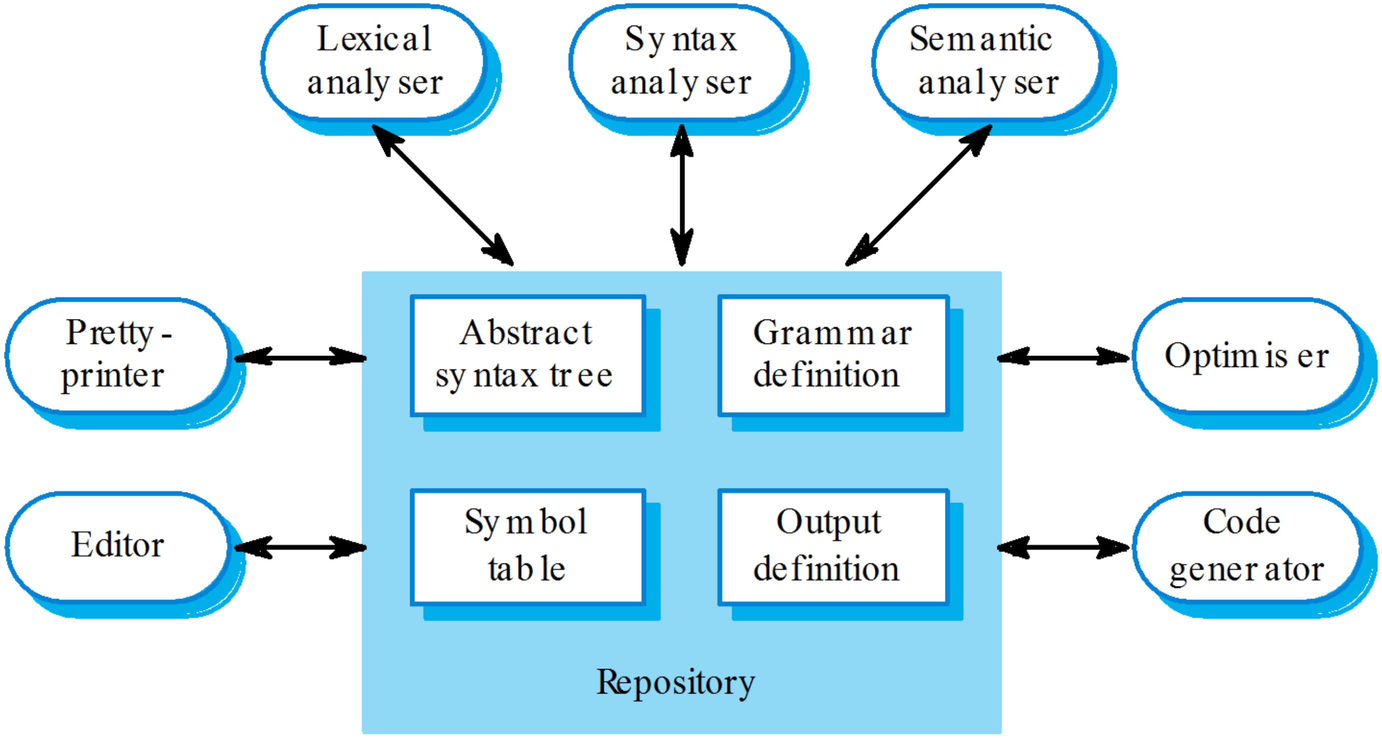
Interpreter

Fetch
Execute

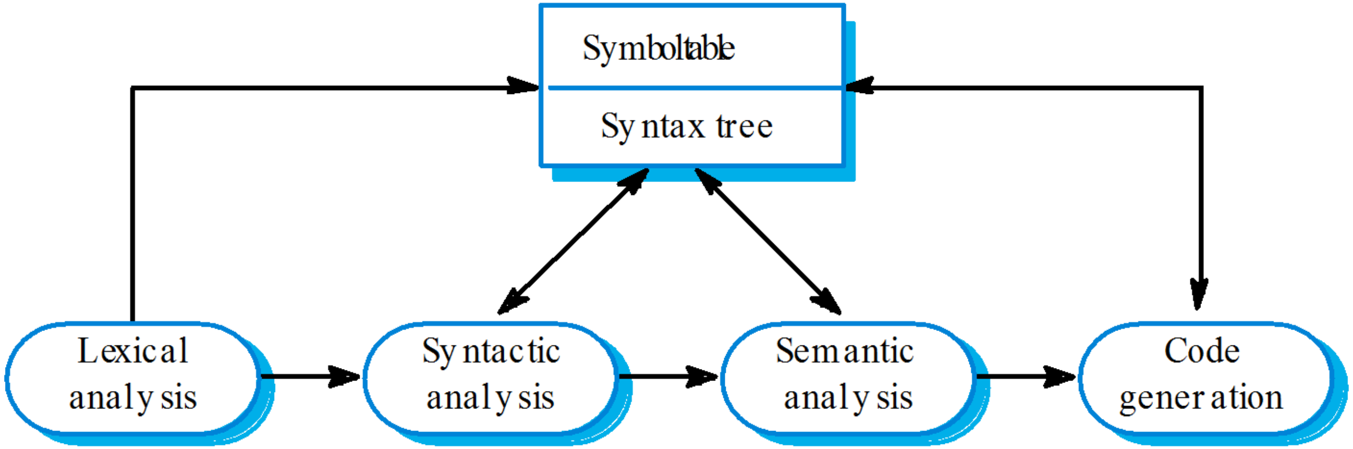
Data

Results

Compilers: repository model



Compilers: data-flow model



C# Lecture

Graphical User Interfaces:

.NET Windows Forms

- **Introductory remark:**
  - There are currently 2 platforms that provide support for creating GUIs with C#:
    1. .NET Windows Forms
    2. .NET Windows Presentation Foundation (WPF).

  - I will only speak about the first platform.

Graphical user interfaces

- C# project type: Windows Forms Application
  - reference to `System.Windows.Forms` automatically added
  - ...and to other packages necessary for, e.g., drawing.
- A class that is supposed to have windowed user interface must inherit from `Form`.

Windows

```
public partial class Form1 : Form
{
        public Form1()
        {
            InitializeComponent();
        }
}
```

partial :

- the code of the class is split into more .cs-files
- each file contains a part of the class
- this is the normal file structure generated automatically by the Visual Studio designer.

Windows

- Showing a window:
  - In program.cs / Main:

```
Application.Run(new Form1());
```

  - ...but it is possible to create and show a window at any time (for instance, dialog-boxes):

```
Form f = new Form();
f.Show();
```

Windows

- Add *controls*:
    - A few buttons,
    - A Panel

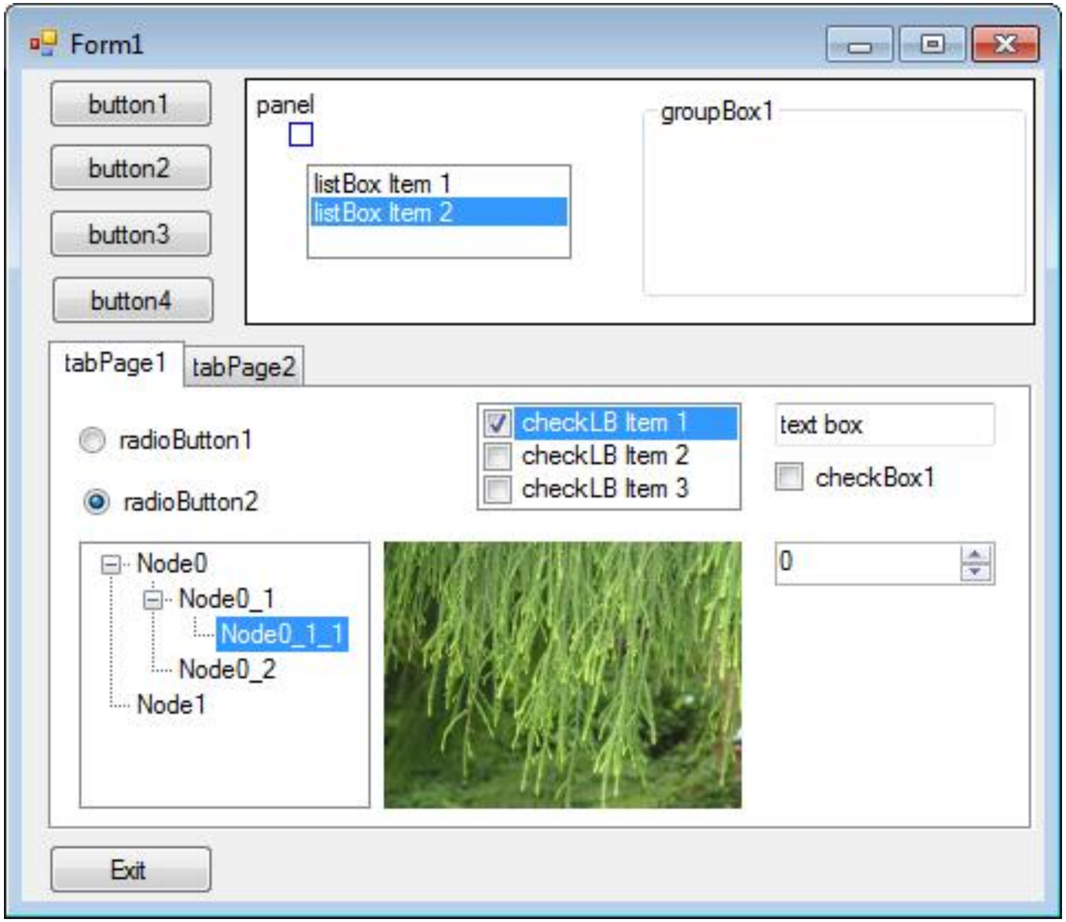
Controls

- Placing controls
  - The components hosting controls are *containers*
  - Examples of containers:
    - Form,
    - Panel,
    - GroupBox,
    - TabControl
  - Other controls can be added to containers

Placing controls

- Laying out controls:
  - In the designer
    - fine-tuning possible, using “Properties” view, in Visual Studio
  - In code
    - *.Designer.cs contains values set in designer (do not modify: it is automatically [re]created by Visual Studio designer!)
    - directly in the form’s .cs file
- Layout concepts:
  - Docking,
  - Anchoring.

Other controls



Delegates

- from MSDN:
  - a delegate is *similar to a function pointer in C or C++*
  - *encapsulates a reference to a method*
  - a *delegate declaration defines a* [reference] ***type*** *that encapsulates a method with a particular set of arguments and return type*

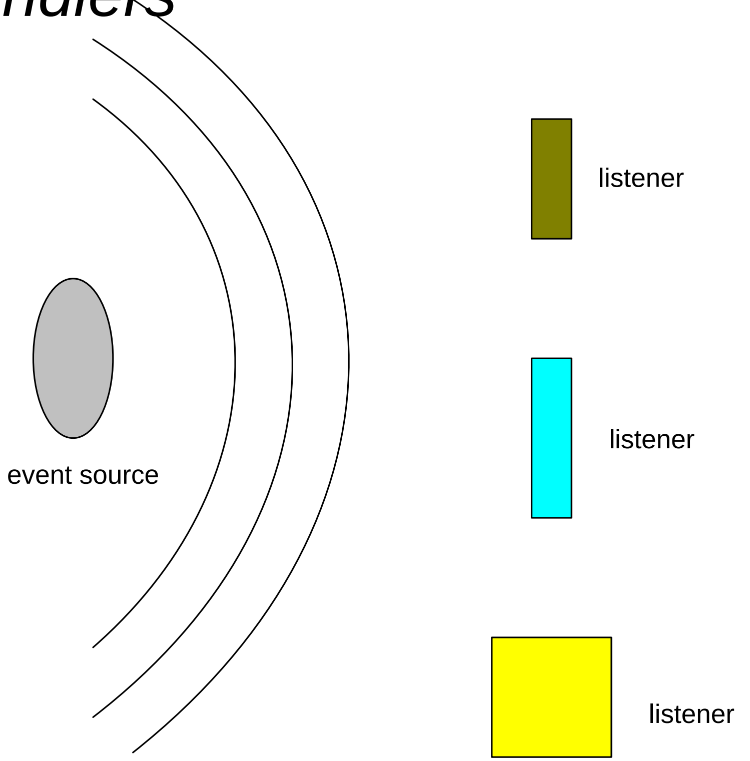
```
[<access>] delegate <return_type> <name>(<param_list>)
```

  - delegates *can be composed using the "**+**" operator*
  - an instance of a delegate is created with *new*

```
public delegate void SendString (string s);
. . .
       SendString mySendStringDelegate = new SendString(DisplayMessage);
. . .
   private void DisplayMessage(string s)
   { . . . }
```

Events

- Sources – event generators
- Listeners – event consumers
  - must provide *event handlers*



Events

- from MSDN:

  An event in C# is a way for a class to provide notifications to clients of that class when some interesting thing happens to an object.

- the object transmits a notification, to whatever is interested, that something has happened / changed
- *events are declared using delegates.*

Events

```
public class MyClassWithEvent {
    public event SendString NewMessage;
    public void MyFunction() {
        bool ok = true; . . .
        if (!ok) OnNewMessage(“Not OK!”);
    }
    private void OnNewMessage(string msg) {
        if (NewMessage != null)
            NewMessage(msg);
    }
}
public class AnotherClass
{
        . . .
        MyClassWithEvent myClass = new MyClassWithEvent ();
        myClass.NewMessage += new SendString(DisplayMessage);
}
```

Event mechanism

- The event consumers must:
  - – ***register*** its event handling function to the event source (also called *wiring*)
- Example - adding a click handler to a button:

```
myBrowseButton.Click +=
       new System.EventHandler(browseButton_Click);

void browseButton_Click(object sender, EventArgs e)
{
   // . . .
}
```

Event handlers

- recommended:

```
public delegate void MyEventHandlingDelegate
        (object sender, TArgs e);
```

- where `TArgs` is a type derived from EventArgs.

Events for UI-components

- mouse events (Click, MouseUp, MouseDown, …)
- key events (KeyPress, KeyDown, …)
- selection events (SelectedIndexChanged, …)
- check events (CheckedChanged)
- form-specific events (Load, Resize, …)

Homework

- Using the four generic application types, can you classify any of the following systems (or parts of them)?
  - ADMSys
  - Leo dictionary
  - The system behind the website geizhals.at