CS 580 Client-Server Programming
Fall Semester, 2000
Doc 1 Introduction
Contents

References

Code Complete by Steve McConnell

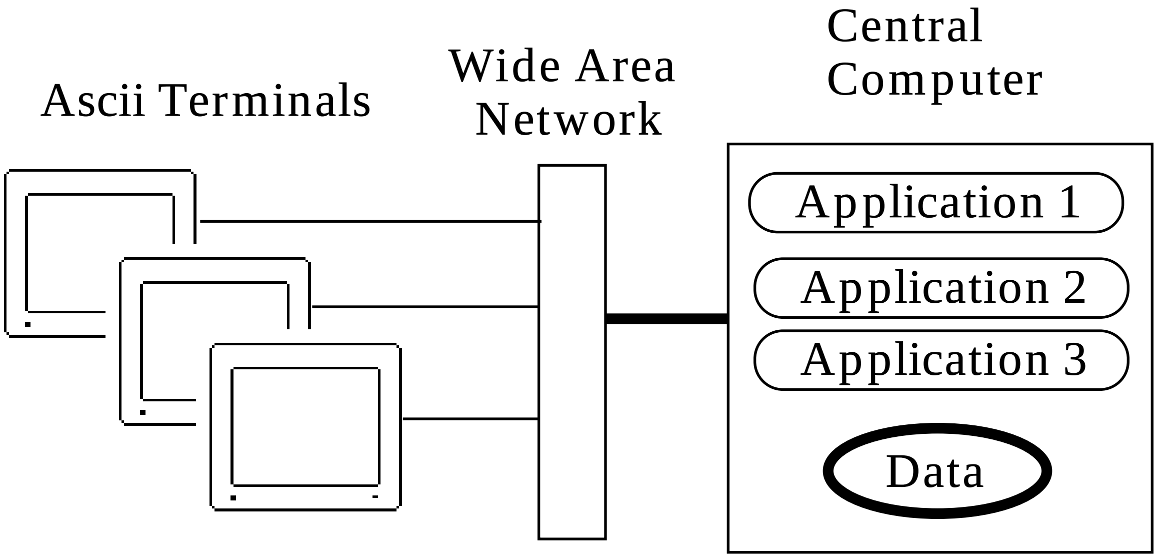
Introduction to Course
Items To Cover

- Prerequisites
- Grades
- Programs
- Homework
- Projects
- Class notes, www
- How lectures will work
- Why this course
- When will it be offered again?
- Crashers
- Machines, accounts, languages

Computing ***"Paradigms"***

- Centralized Multi-user Architecture
- Distributed Single-User Architecture
- Client/Server Architecture

Centralized Multi-user Architecture



Large central computers serving many users

Motivating Factors

Service large number of users (200 to 10,000+)

Centralized storage for large data bases

Minimize data on slow networks

Strengths

Very stable, very reliable, well supported

Cost-effective why to support thousands of users

Large pool of technical staff

Large number of business applications available

Weakness

Proprietary hardware and software

Very expensive

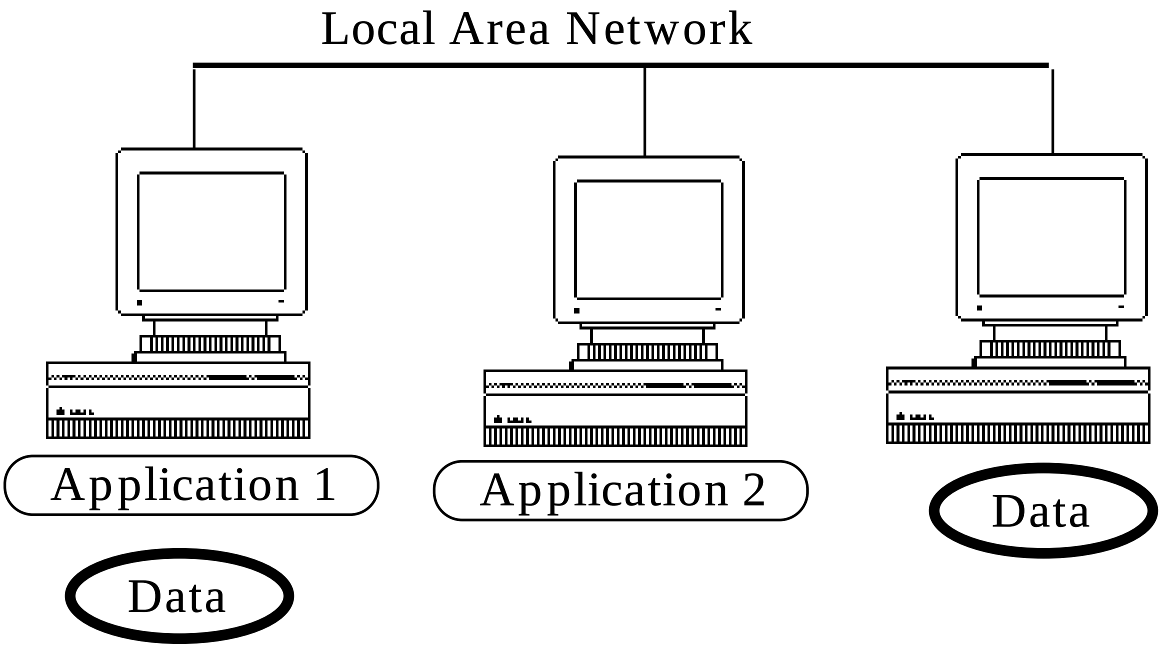
Requires large support staff

Costly to incrementally add more capacity

Mind Set

Hierarchical organization (Bureaucratic heaven)

Distributed Single-User Architecture



Motivating Factors

Low cost fast local area networks

Provide small number of users with compute power

Failure of MIS departments to be responsive and cost-effective

Strengths

Cheap hardware and software

Lots of third-party software

User is in complete control of environment

Low cost to add more users

Weakness

Sharing of resources across many users is difficult

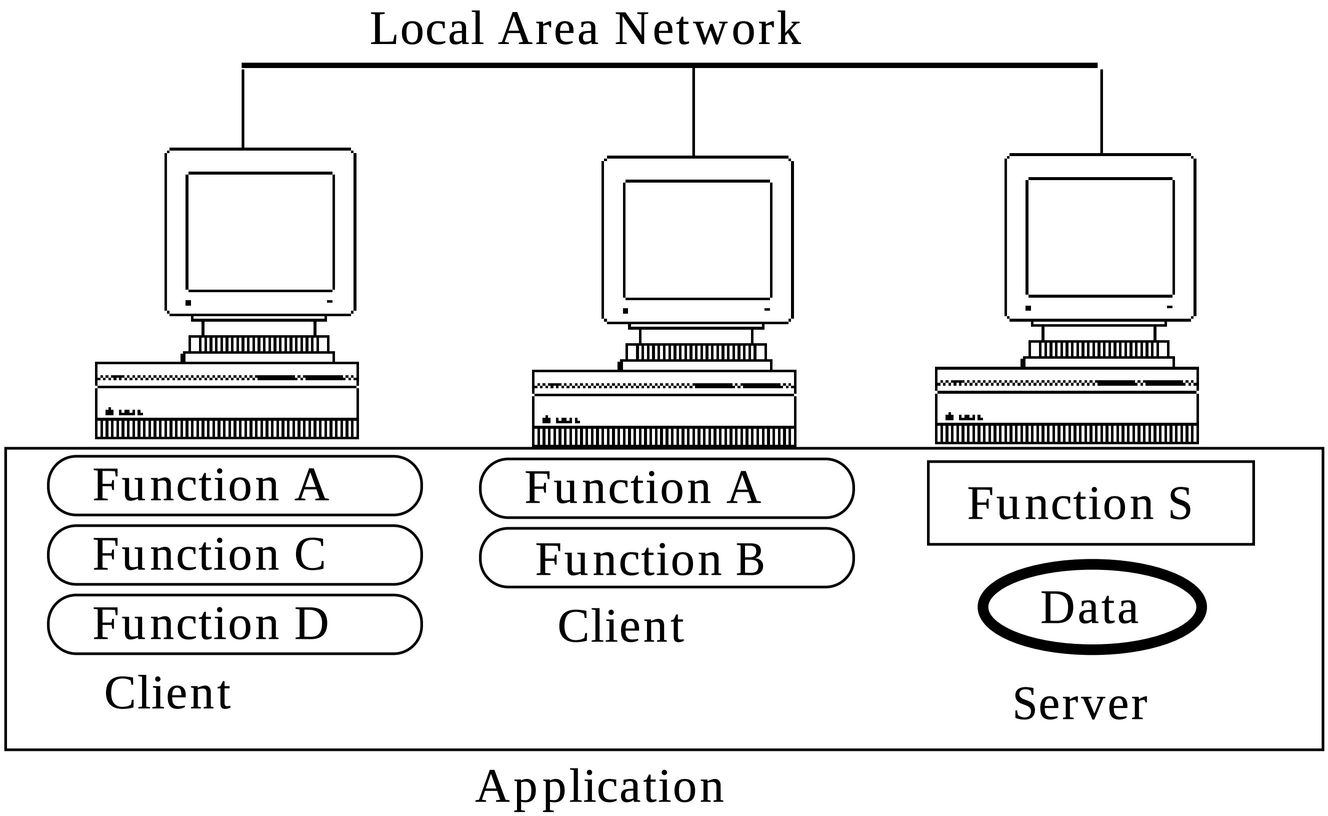
Networks and OS do not provide good control or management over computer resources

Multivender environments can cause operation, support and reliability problems

Mind set

Individualism (Lone Ranger syndrome)

Client/Server Architecture



Motivating Factors

Limitations of other modes of computing

Utilize easy to use micro computers as front end to mainframe computers

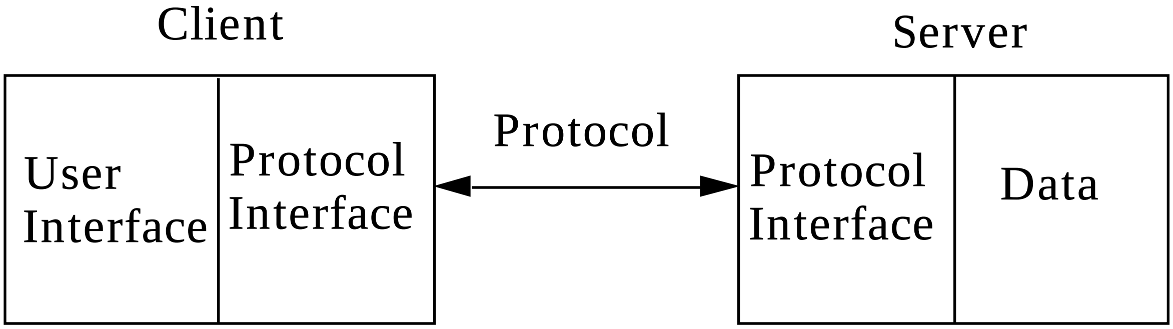
Strengths

- Cost-effective way to support thousands of users
- Low cost to add more users
- Cheap hardware and software
- Provides control over access to data
- User remains in control over local environment
- Flexible access to information

Weaknesses

- Reliability
- Complexity
- Lack of trained developers

Introduction to Client-Server
What is Client-Server?



Client

User Interface

Protocol Interface

Protocol

Server

Protocol Interface

Data

Client

Application that initiates peer-to-peer communication

Translate user requests into requests for data from server via protocol

GUI often used to interact with user

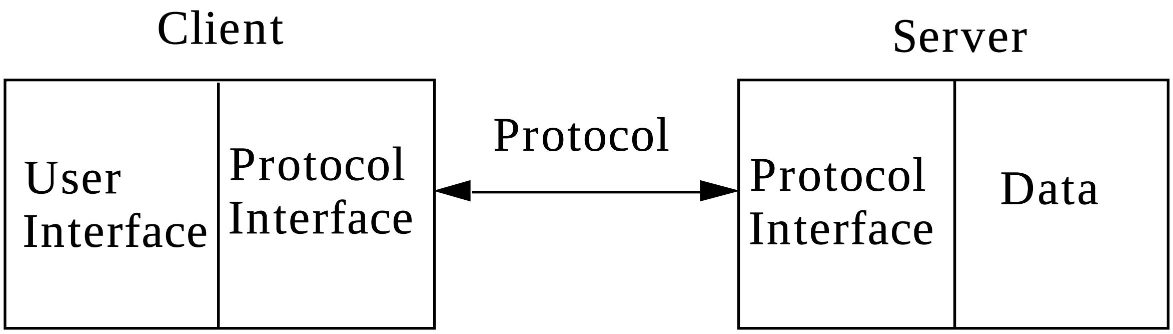
Server

Any program that waits for incoming communication requests from a client

Extracts requested information from data and return to client

Common Issues

- Authentication
- Authorization
- Data Security
- Privacy
- Protection
- Concurrency



Example: World Wide Web (WWW)

Data

Server normally provides data to clients

Often utilizes some data base

WWW data is HyperText Markup Language (html) files

```
<!DOCTYPE HTML SYSTEM "html.dtd">
<HTML>
<HEAD><TITLE>
Client Server Programming
</TITLE></HEAD>
<BODY>
<H2>Client Server Programming</H2>
<HR>
```

Protocol

How the client and server interact

Glue that makes client-server work

Involves using low level network protocols and application specific protocols

Designing application specific protocols is very important

WWW uses the HyperText Transfer Protocol

```
Request            =    SimpleRequest | FullRequest

SimpleRequest      =    GET <uri> CrLf

FullRequest        =    Method URI ProtocolVersion CrLf
                        [*<HTRQ Header>]
                        [<CrLf> <data>]

<Method>           =    <InitialAlpha>

ProtocolVersion    =    HTTP/1.0

uri                =    <as defined in URL spec>

<HTRQ Header>      =    <Fieldname> : <Value> <CrLf>

<data>             =    MIME-conforming-message
```

Protocol Choices

- Text Based

  Transmit ASCII or Unicode between machines

  HTTP is common transport layer

  XML becoming common

  SOAP new XML standard

- Binary

  Transmit objects between machines

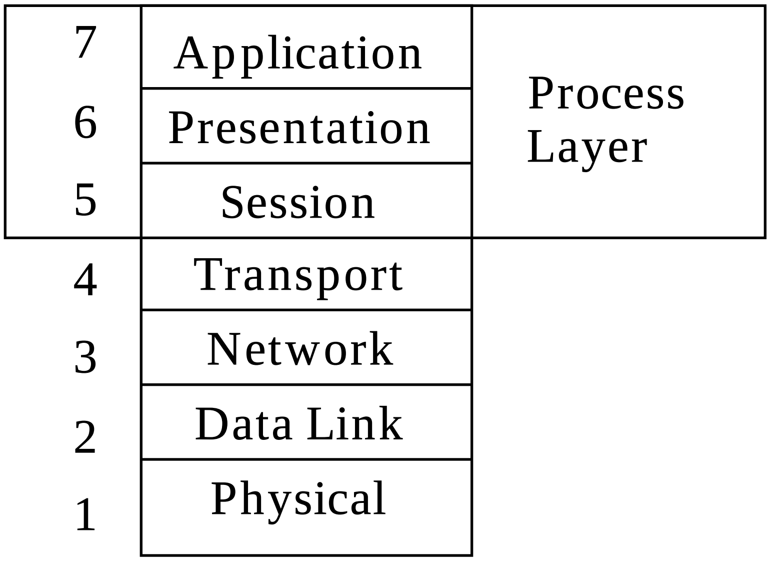
  Faster development time

  RMI, Corba are examples

What this Course is not

An advanced (or beginning) Networking course

OSI Model



7	Application	Process Layer
6	Presentation	
5	Session	
4	Transport	
3	Network	
2	Data Link	
1	Physical	

How to use a client builder application/system

Powerbuilder

What this Course covers

Skills & knowledge required to build client-server applications

What Client-Server Requires of a Programmer

- Designing robust protocols
- Network programming
- Designing usable computer-human interfaces
- Good documentation skills
- Good debugging skills
- Understand the information flow of the company/customer
- Mastery of concurrency
- Multi-platform development
- Database programming
- Security

Programming Issues
Names

"Finding good names is the hardest part of OO Programming"

"Names should fully and accurately describe the entity the variable represents"

What role does the variable play in the program?

Data Structure	**Role, function**
InputRec	EmployeeData
BitFlag	PrinterReady

Some Examples of Names, Good and Bad

TrainVelocity	Velt, V, X, Train
CurrentDate	CD, Current, C, X, Date
LinesPerPage	LPP, Lines, L, X

OOP Names - Common Problems

```
class Stack
   {
   Vector theStack = new Vector();

   public void push( object x )
      {
      theStack.add( x );
      }

   // code deleted
   }

class DriverProgram
   {
   public void static main( String[] args )
      {
      // blah blah blah

      Stack stack;

      aFooFunction( stack );

      // more blah
      }

   void aFooFunction( Stack aStack )
      {
      }
   }
```

Comments

"Comments are easier to write poorly than well, and comments can be more damaging than helpful"

What does this do?

```
for i := 1 to Num do
 MeetsCriteria[ i ] := True;
for  i := 1 to Num / 2  do begin
 j := i + i;
 while ( j <= Num ) do begin
  MeetsCriteria[ j ] := False;
  j := j + i;
 end;
for i := 1 to Mun do
 if MeetsCriteria[ i ] then
  writeln( i, ' meets criteria ' );
```

How many comments does this need?

```
for PrimeCandidate:= 1 to Num do
   IsPrime[ PrimeCandidate] := True;

for  Factor:= 1 to Num / 2  do begin
   FactorableNumber := Factor + Factor ;
   while ( FactorableNumber <= Num ) do begin
      IsPrime[ FactorableNumber ] := False;
      FactorableNumber := FactorableNumber + Factor ;
   end;
end;

for PrimeCandidate:= 1 to Num do
   if IsPrime[ PrimeCandidate] then
      writeln( PrimeCandidate, ' is Prime ' );
```

Good Programming Style is the Foundation of Well Commented Program

Kinds of Comments

- Repeat of the code

```
X := X + 1   /* add one to X



/* if allocation flag is zero */

if ( AllocFlag == 0 ) ...
```

- Explanation of code
  Used to explain complicated or tricky code

```
*p++->*c = a

/* first we need to increase p by one, then ..
```

Make code simpler before commenting

```
(*(p++))->*c = a



ObjectPointerPointer++;
ObjectPointer = *ObjectPointerPointer;
ObjectPointer ->*DataMemberPointer = a;
```

- Marker in the code

```
/*  **** Need to add error checking here  **** */
```

- Summary of the code
  Distills a few lines of code into one or two sentences

- Description of the code's intent

  Explains the purpose of a section of code

```
{ get current employee information }   intent

{ update EmpRec structure }     what
```

Commenting Efficiently

- Use styles that are easy to maintain

```
/************************************
 * module: Print                 *
 *                        *
 * author: Roger Whitney          *
 * date:   Sept. 10, 1995        *
 *                        *
 * blah blah blah               *
 *                        *
 ************************************/
```

```
/************************************
  module: Print

  author: Roger Whitney
  date:   Sept. 10, 1995

  blah blah blah

 ************************************/
```

- Comment as you go along

Commenting Techniques
Commenting Individual Lines

Avoid self-indulgent comments

```
MOV AX,  723h        ;    R. I. P. L. V. B.
```

Endline comments have problems

```
MemToInit := MemoryAvailable(); { get memory available }
```

- Not much room for comment
- Must work to format the comment

Use endline comments on

- Data declarations
- Maintenance notes
- Mark ends of blocks

Commenting Paragraphs of Code

Write comments at the level of the code's intent

Comment the why rather than the how

Make every comment count

Document surprises

Avoid abbreviations

How verses Why

How

```
/* if allocation flag is zero */

if ( AllocFlag == 0 ) ...
```

Why

```
/* if allocating a new member */

if ( AllocFlag == 0 ) ...
```

Even Better

```
/* if allocating a new member */

if ( AllocFlag == NEW_MEMBER ) ...
```

Summary comment on How

```
{ check each character in "InputStr" until a
  dollar sign is found or all characters have
  been checked }

Done   := false;
MaxPos := Length( InputStr );
i      := 1;
while ( (not Done) and (i <= MaxLen) ) begin
   if ( InputStr[ i ] = '$' ) then
      Done := True
   else
      i := i + 1
end;
```

Summary comment on Intent

```
{ find the command-word terminator }

Done   := false;
MaxPos := Length( InputStr );
i      := 1;

while ( (not Done) and (i <= MaxPos ) ) begin
   if ( InputStr[ i ] = '$' ) then
      Done := True
   else
      i := i + 1
end;
```

Summary comment on Intent with Better Style

```
{ find the command-word terminator }

FoundTheEnd      := false;
MaxCommandLength := Length( InputStr );
Index            := 1;

while ((not FoundTheEnd) and
      (Index <= MaxCommandLength)) begin

   if ( InputStr[ Index ] = '$' ) then
      FoundTheEnd := True;
   else
      Index := Index + 1;
end;
```

Commenting Data Declarations

Comment the units of numeric data

Comment the range of allowable numeric values

Comment coded meanings

```
var
   CursorX:   1..MaxCols;       { horizontal screen position of cursor }
   CursorY:   1..MaxRows;       { vertical position of cursor on screen }

   AntennaLength:   Real;       { length of antenna in meters: >= 2 }
   SignalStrength: Integer;     { strength of signal in kilowatts: >= 1 }

   CharCode:  0..255;      { ASCII character code }
   CharAttib:  Integer;    { 0=Plain; 1=Italic; 2=Bold  }
   CharSize:   4..127;     { size of character in points }
```

Comment limitations on input data

Document flags to the bit level

Commenting Routines

Avoid Kitchen-Sink Routine Prologs

Keep comments close to the code they describe

Describe each routine in one or two sentences at the top of the routine

Document input and output variables where they are declared

Differentiate between input and output data

Document interface assumptions

Keep track of the routine's change history

Comment on the routine's limitation

Document the routine's global effects

Document the source of algorithms that are used

```
procedure InsertionSort
   {
   Var Data:      SortArray;    { sort array elements }
      FirstElement: Integer          {index of first element to sort}
      LastElement: Integer           {index of last element to sort}
   }
```

Object-Oriented Programming
Conceptual Level Definition

Abstraction

"Extracting the essential details about an item or group of items, while ignoring the unessential details."

Edward Berard

"The process of identifying common patterns that have systematic variations; an abstraction represents the common pattern and provides a means for specifying which variation to use."

Richard Gabriel

Example

Pattern: Priority queue

Essential Details: length
items in queue
operations to add/remove/find item

Variation: link list vs. array implementation
stack, queue

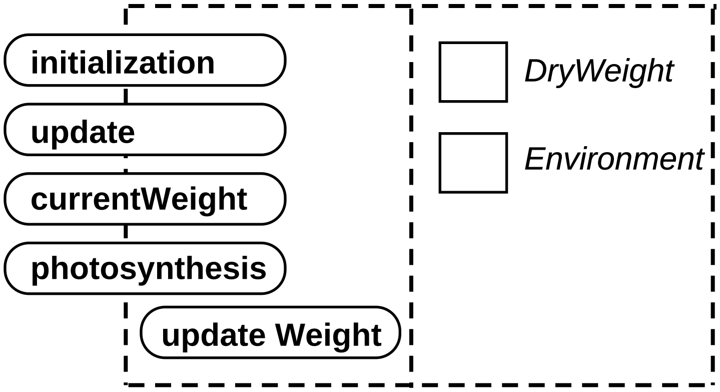
Object-Oriented Programming
Conceptual Level Definition

Encapsulation

Enclosing all parts of an abstraction within a container

Example

Leaf Class



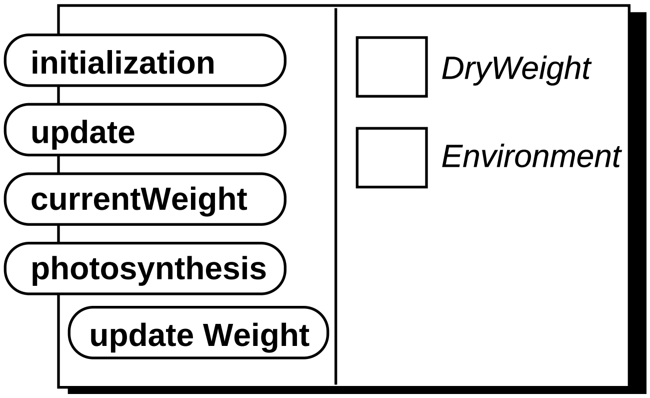
Object-Oriented Programming
Conceptual Level Definition

Information Hiding

Hiding parts of the abstraction

Example

Leaf

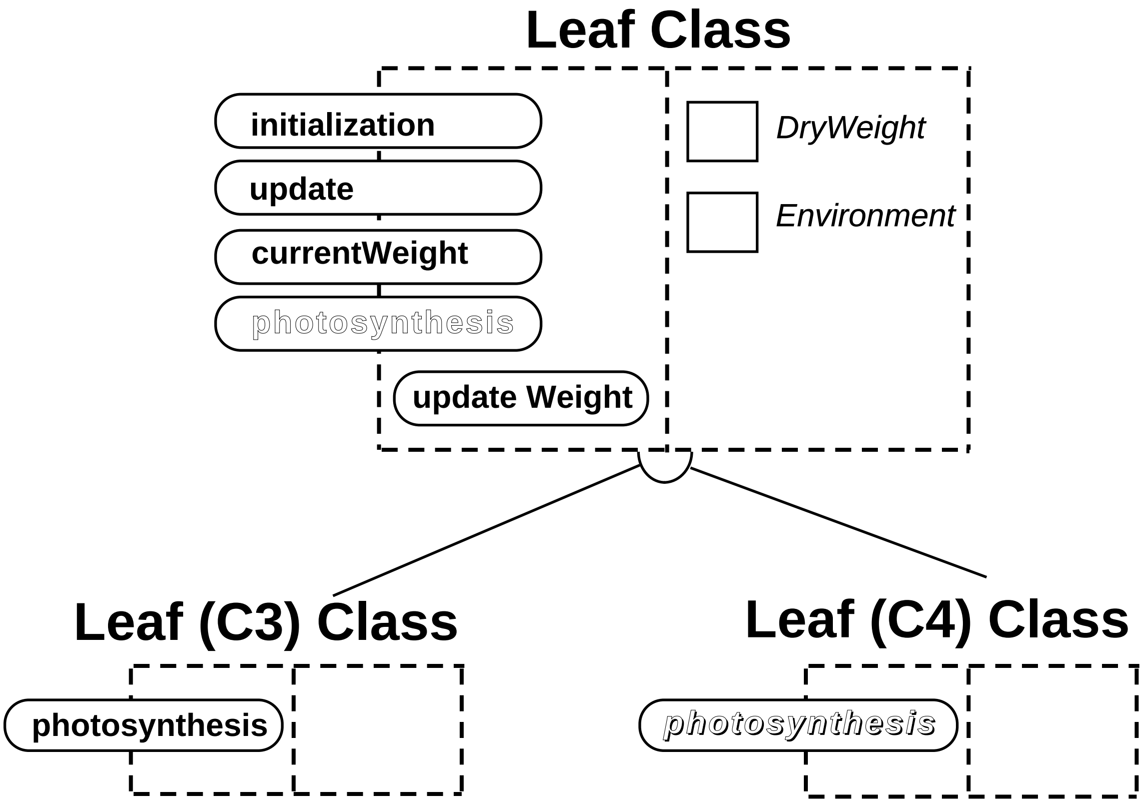


Object-Oriented Programming
Conceptual Level Definition

Hierarchy

Abstractions arranged in order of rank or level

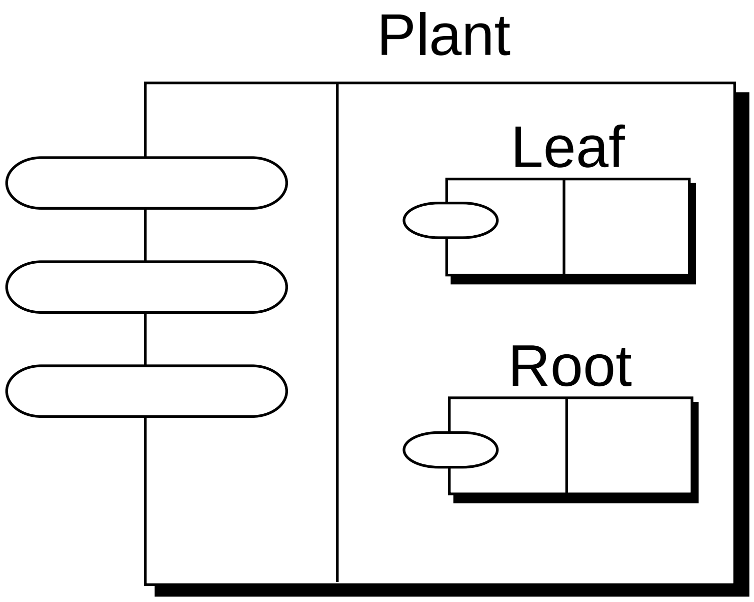
Class Hierarchy



Object-Oriented Programming
Conceptual Level Definition

Hierarchy

Object Hierarchy


Plant
Leaf
Root