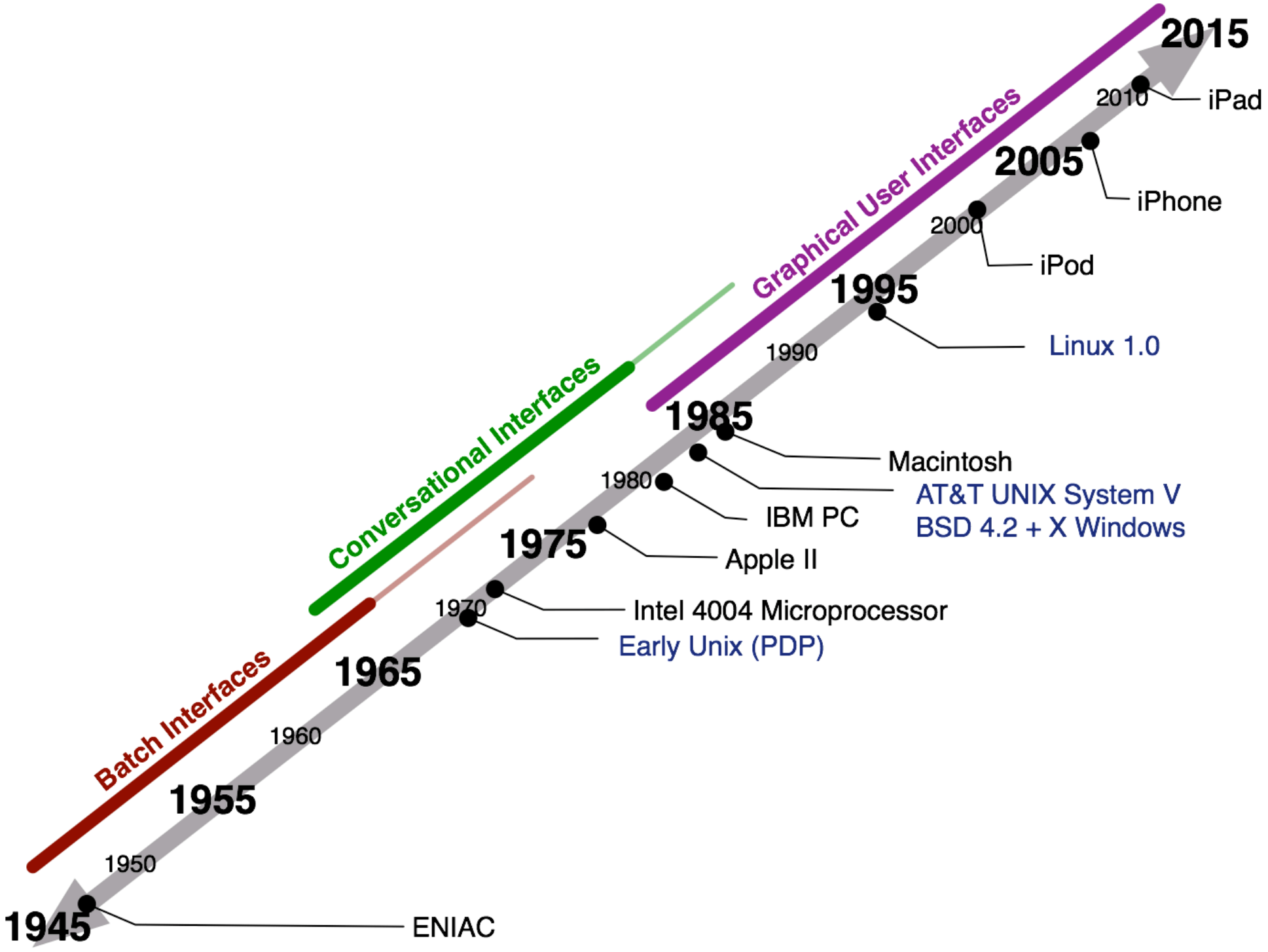
GUI Basics and Windowing Systems

Using X Windows as a case study


Batch Interfaces
Conversational Interfaces
Graphical User Interfaces
1945
ENIAC
1950
1955
1960
1965
1970
Intel 4004 Microprocessor
Early Unix (PDP)
1975
Apple II
1980
IBM PC
1985
Macintosh
AT&T UNIX System V
BSD 4.2 + X Windows
1990
1995
Linux 1.0
2000
iPod
2005
iPhone
2010
iPad
2015

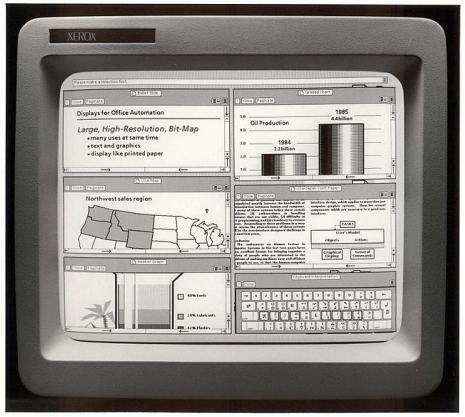
Evolution of GUI Programming

- On early computers, everything was “rolled by hand”
  – “Re-inventing the wheel” with every iteration

- Interaction was very, very limited.
- No graphical input or output
- No standards for interaction
  – Ivan Sutherland Sketchpad system

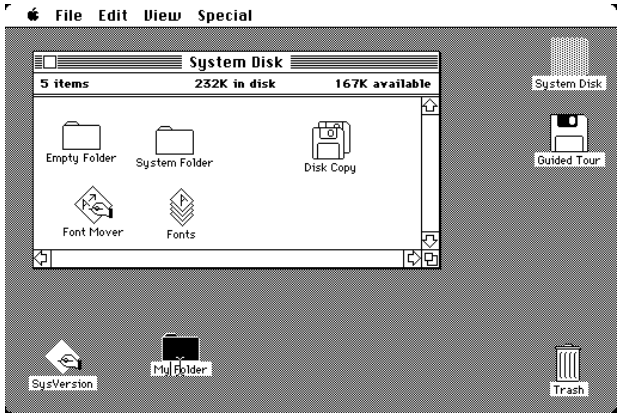
- Leading up to the 1980’s, researchers and industry began to develop GUI architectures and systems to simplify both interaction and programming.

Evolution of GUIs



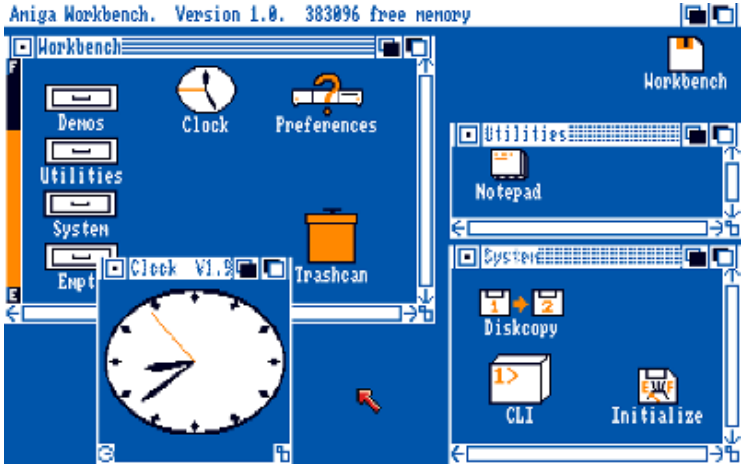
Xero Star (1981)

- Developed at Xerox PARC
- Not commercially successful.



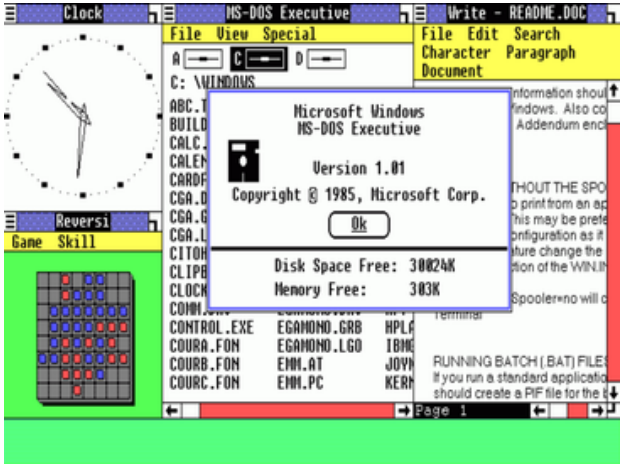
Apple Macintosh (1984)

- *Inspired* by Xerox PARC
- Commercial hit!



Amiga Workbench (1985)

- Limited success
- Never progressed further



Microsoft Windows 1.0 (1985)

- Limited success
- Led to Windows 3.0/3.1

GUI Characteristics

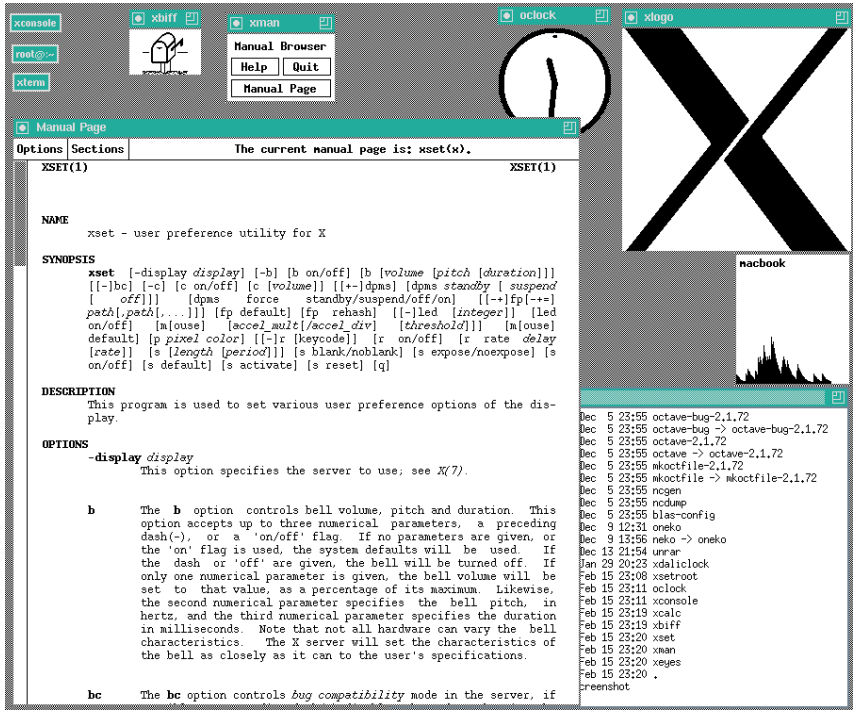
- GUI architectures share common characteristics:
  - Support output to a graphical display
    - Display text and high-resolution graphics
  - Handle user input from multiple devices
    - Minimally, keyboard (text) and a positional input device (mouse)
  - Provide an interface to display / manipulate content.
    - Most modern GUIs use a desktop metaphor
      - Windows contain data and can be manipulated (resized, moved, overlap one another)
    - Common GUI elements. e.g. scrollbars, buttons.

- A windowing system provides input, output and window management capabilities to the operating system.

X Windows (X11) System

X Windows

- Developed in 1984 (based on MIT Athena project) by a consortium of companies.
- Standard windowing system for Unixes.
- Free and cross-platform (OS, processor agnostic)
- One of the most successful free-software projects, ever.



- Base windowing system, separate from operating system.
  - Not a window manager (more on that later)
  - Does not specify the style of user interface
- What does it do?
  - A protocol to create windows, handle input, draw graphics
  - A standard for low-level graphical output and user input

X Windows Design Criteria (~1986)

What should X-windows have?

1. Implementable on a **variety of displays**
2. Applications must be **device independent**
3. Must be **network transparent**
4. Support **multiple, concurrent application displays**
5. Support many different applications
6. Support output to **overlapping windows** (… even when partially obscured)
7. Support a hierarchy of **resizable windows** (… an application can use many windows at once)
8. High-performance, high-quality text, **2-D graphics**, imaging
9. System should be extensible

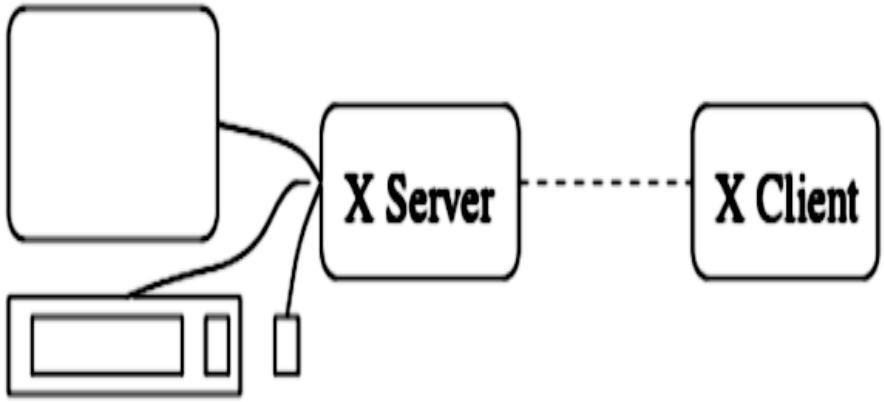
(from Scheifler & Gettys, 1986)

- In X, a display may have multiple screens
- A display may have multiple windows
- A window may cross multiple screens

X Client-Server Architecture

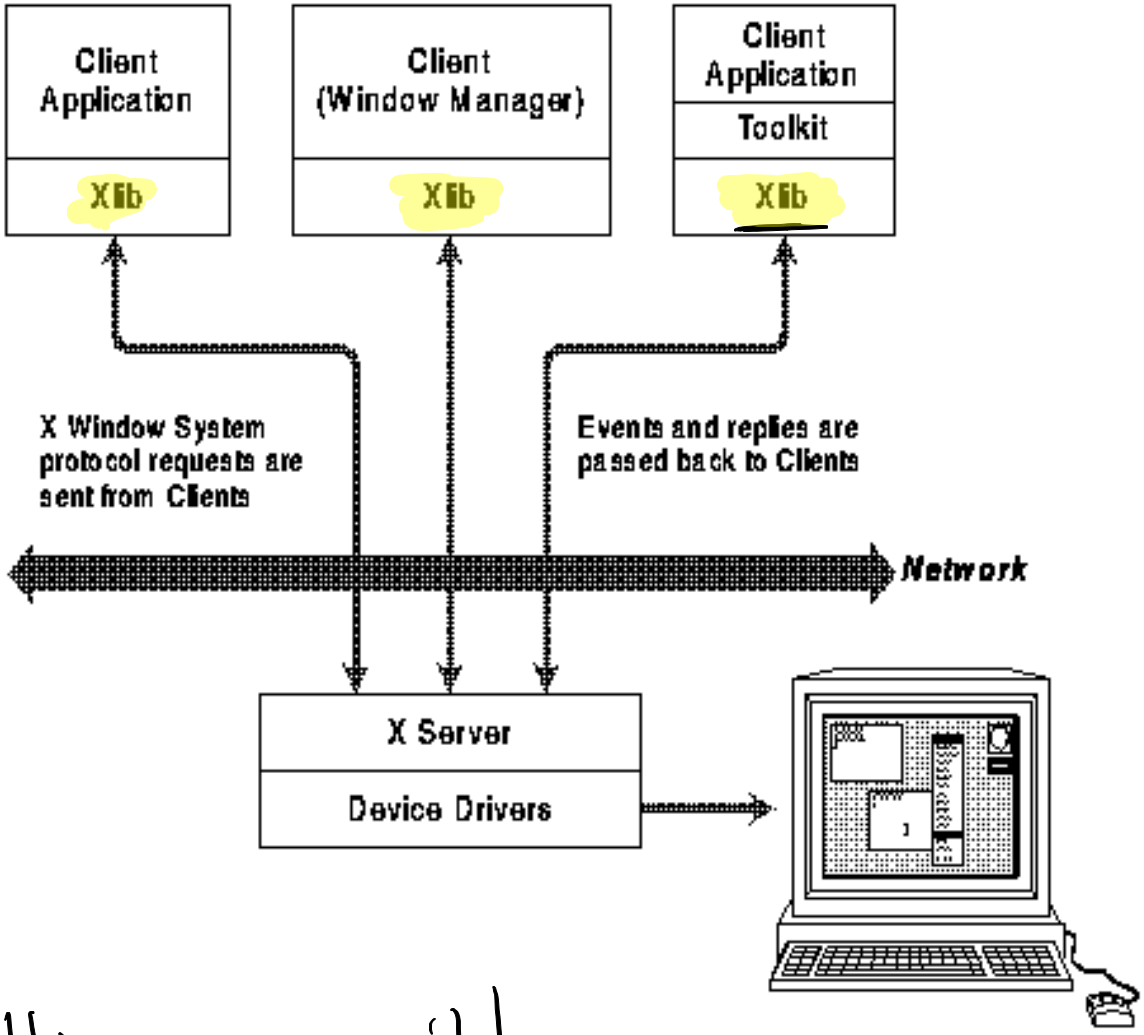
- Separate user interface and application:
  - the **X Client** handles all application logic
  - the **X Server** handles all display output and user input
- A server handles requests from multiple clients, processes data as requested, and returns the results to the clients

- X inverts conventional www server/client relationship
  - in www, web browser is the “client”, web site is the “server”



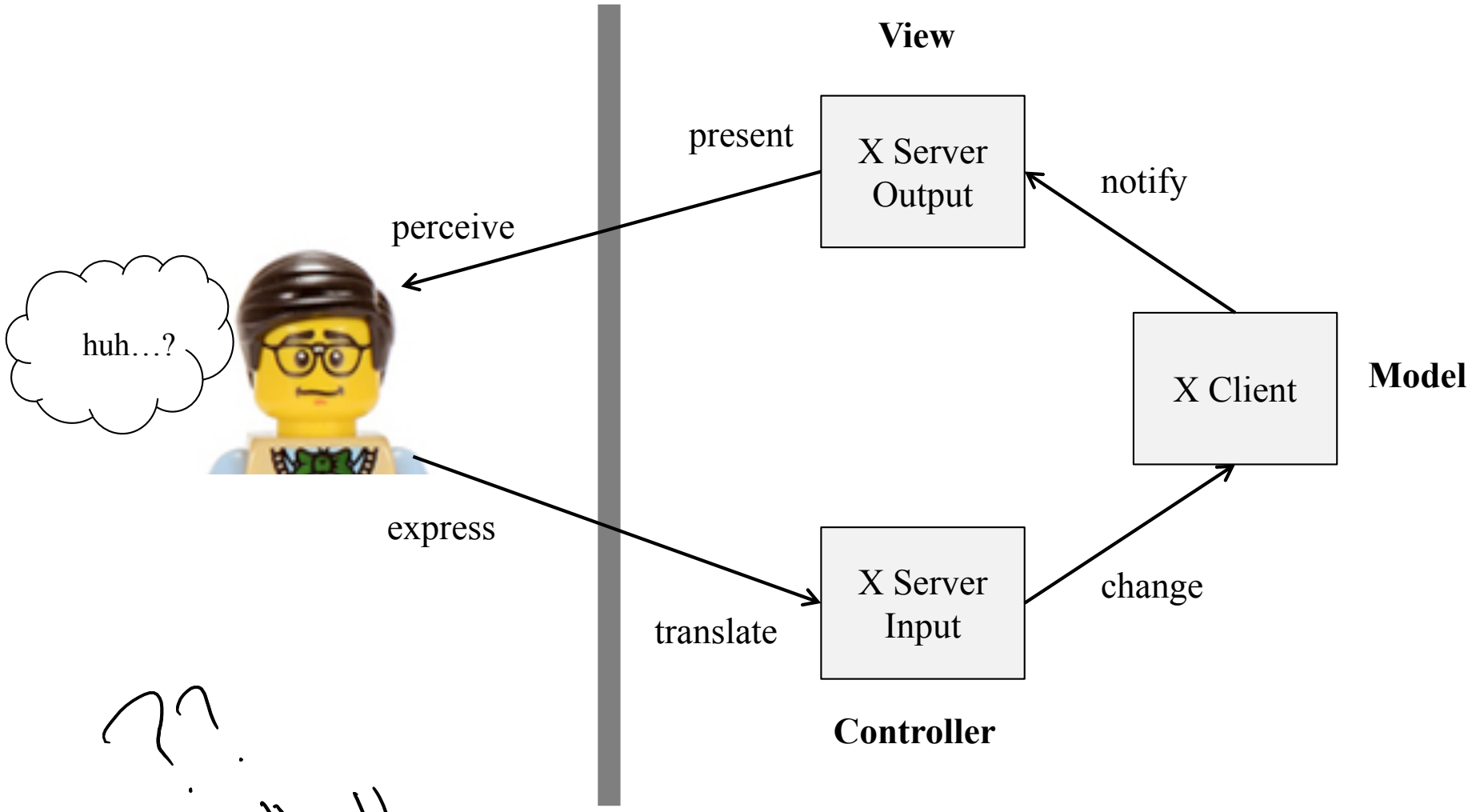
Why Client-Server?

- Goal was flexibility and economy
- Many clients (perhaps multiple machines) running applications
- One display used to monitor the apps.



Xlib makes everything accessible.

X Windows as MVC Architecture



??

Isn't the client the user?

Structure of a Typical X Program

1. Perform client initialization
2. **Connect to the X server**
3. **Perform X related initialization**
4. *Event loop:*
   - **get next event from the X server**
   - handle the event:
     - if the event was a quit message, exit the loop
     - do any client-initiated work
   - **send drawing requests to the X server**
5. **Close down the connection to the X server**
6. Perform client cleanup

Xlib (X Windows Library)

- Library to wrap low level X Window protocol
  - to avoid implementing message passing for every new program
- Xlib is *not* a window manager
- Xlib does not specify style of user interface or provide “widgets”

- Uses buffered input and output queues
  - need to flush them: XSync, XFlush

- Xlib functions:
  - connection operations: e.g. XOpenDisplay, XCloseDisplay, …
  - connection operation requests: e.g. XCreateWindow, XCreateGC,…
  - connection information requests: e.g. XGetWindowProperty, …
  - local event queue operations: e.g. XNextEvent, XPeekEvent, …
  - local data operations: e.g. XLookupKeysym, XParseGeometry, XSetRegion, XCreateImage, XSaveContext, …

- Xlib data types:
  - e.g. Display, Window, GC, XSizeHints, XWhitePixel, etc.

Display a Window (openwindow.min.cpp)

```
Display* display;
Window window;                              // save the window id

int main( int argc, char* argv[] ) {
    display = XOpenDisplay("");             // open display
    if (!display) exit (-1);                // couldn't open, so bail
    int screen = XDefaultScreen(display);// info about the display
    window = XCreateSimpleWindow(display,
    XDefaultRootWindow(display),            // window's parent
         10, 10, 400, 300, 2,               // location: x, y, width, height
         XBlackPixel(display, screen),      // foreground colour
         XWhitePixel(display, screen));     // background colour
    XMapRaised(display, window);            // put window on screen
    XFlush(display);                        // flush the output buffer
    std::cout << "ENTER2exit"; std::cin.get(); // wait for input
    XCloseDisplay(display);
}
```

Can pass in ip-address and control others.

Contrast: Opening a Window in Java

```
import javax.swing.*;

public class TestWindow extends JFrame{

    public TestWindow(){
        this.setTitle("My Window");
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }

    public static void main (String args[]){
        TestWindow myWindow = new TestWindow();
        myWindow.setSize(400, 300);
        myWindow.setVisible(true);
    }
}
```

Recap: X Windows Design

- Much of the XWindows architecture was influenced by its time period
  - Larger server machines and low-cost client displays because computation was expensive
- As computation got cheaper, certain aspects of program behavior could be assumed:
  - Software runs on the client computer
  - OS handles display rendering and has device drivers to coordinate with specific hardware.
- Over time, in a multi-OS world, even this restriction became a burden:
  - Services could be abstracted even farther from the underlying architectures, and programs could run on virtual machines (e.g. JVM, .NET VM, etc.)
  - Write once for a generic ‘virtual’ architecture, run the program on the virtual machine.
  - Any platform for which the ‘VM’ is implemented could execute the same program.

Intermission

How to run X11 sample code.

Setting up XWindows

You need to:

1. Launch an Xserver to manage I/O for your application.
2. Compile your code on a machine that has X11 lib available, then run the executable.

These can be done on

- the same machine (e.g. Linux), or
- two different machines (e.g. Linux executable, redirecting to an Xserver running on Windows or Mac).

Installing an XServer

- Install an Xserver to display output and forward input to the remote Xclient application.

- Steps vary based on your operating system:
  - Windows: Install Xming (https://sourceforge.net/projects/xming/)
  - MacOS: Install XQuartz (http://www.xquartz.org).
  - Linux: You're already running Xwindows and have an Xserver running.

- If running Windows, make sure to launch your Xserver (Xming) before proceeding.
- Mac/Linux users don't need to do this; your Xserver will launch as required.

Compiling C++ Examples

- You need to compile on a system that has Xwindows installed (since you need libraries, headers etc.)
- Steps vary based on your operating system:
  - Windows: You can't compile here. Sorry.
  - MacOS: Install Xquartz from http://www.xquartz.org to get the required libraries.
  - Linux: X is already installed; just compile your code.
- Compile with appropriate flags, then run executable.

```
g++ -o main main.cpp -L/usr/X11R6/lib -lX11 -lstdc++
./main
```

- Output will be sent to an Xserver if configured.

Practical example

- Using the student environment from a Windows notebook:

1. Install Xming and launch it. This is our Xserver that will display the output on our Windows notebook.

2. Connect to the remote Linux machine to compile and run our code (-Y is needed for ssh forwarding).

```
ssh -Y jdoe@linux.student.cs.uwaterloo.ca
```

3. Compile and run the executable.

```
g++ -o main main.cpp -L/usr/X11R6/lib -lX11 -lstdc++
./main
```

Windows: C++ Examples

- On Windows, you can use Virtual Box and Linux for Xlib examples
  - Install Virtual Box, download Ubuntu (or another Linux distro) as an ISO
  - Open Virtual Box and create an Ubuntu Linux VM
  - Select, but do not start, and update IDE secondary master to point to Linux ISO
  - Start VM and install should proceed
  - Then
    - Install g++ with
      - sudo apt-get install g++
    - Install X11 dev options with
      - sudo apt-get install libx11-dev

Windows: VM Setup

- You may wish to map a local directory onto a subdirectory in my home directory as follows:
  - Install Guest Additions from the Devices menu of the VM
  - On your local machine create a folder
  - From Machine > Settings > Shared Folders click the add icon and create a new directory pointing to that folder.
  - Check the auto-mount and make permanent options
  - On your host OS, make sure that the folder permissions make it accessible
  - Create a symbolic link to /media/<shared folder> to a subdirectory of your home directory.

Windowing Systems

Windowing functions

Base window system vs. window manager

Before Windowing Systems

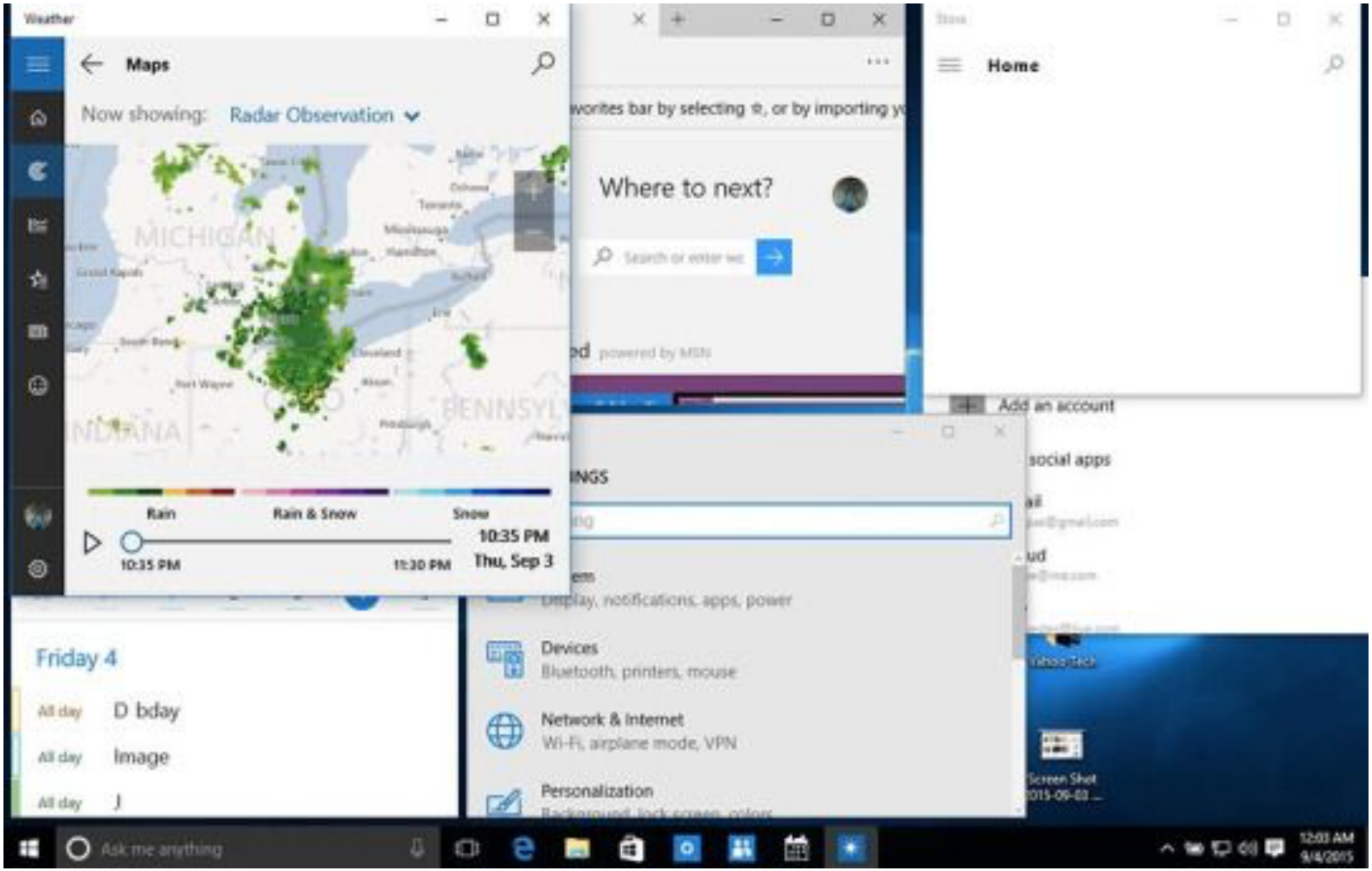
Systems typically ran a single application, which dominated the screen.

X Windows Design Criteria (~1986)

- Implementable on a variety of displays
- Applications must be device independent
- Must be network transparent
- Support multiple, concurrent application displays
- Support many different application and management interfaces
- Support output to overlapping windows
  (… even when partially obscured)
- Support a hierarchy of resizable windows
  (… an application can use many windows at once)
- High-performance, high-quality text, 2-D graphics, imaging
- System should be extensible

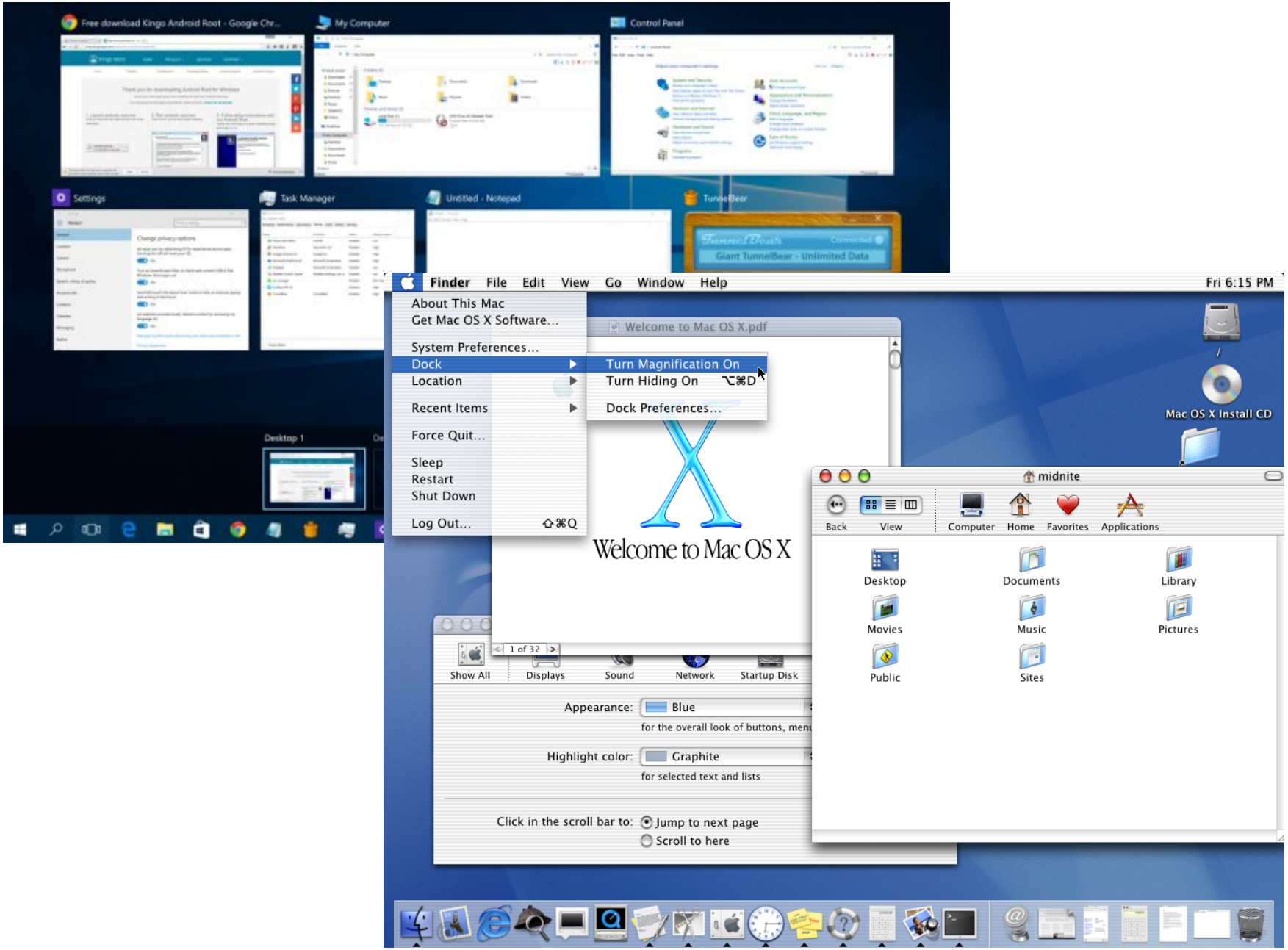
(from Scheifler & Gettys, 1986)

After Windowing Systems

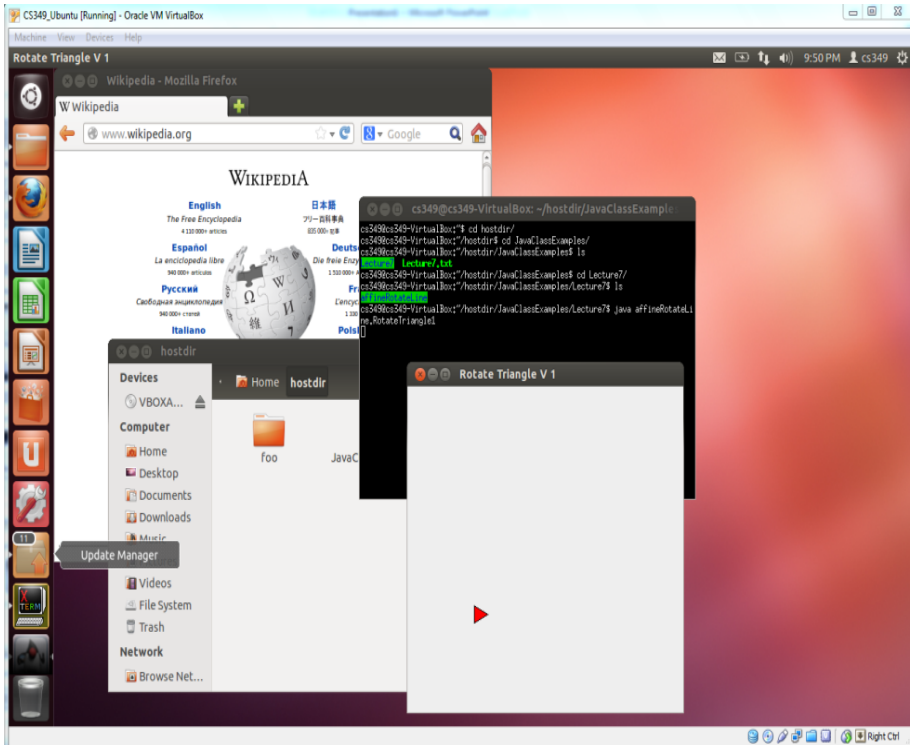


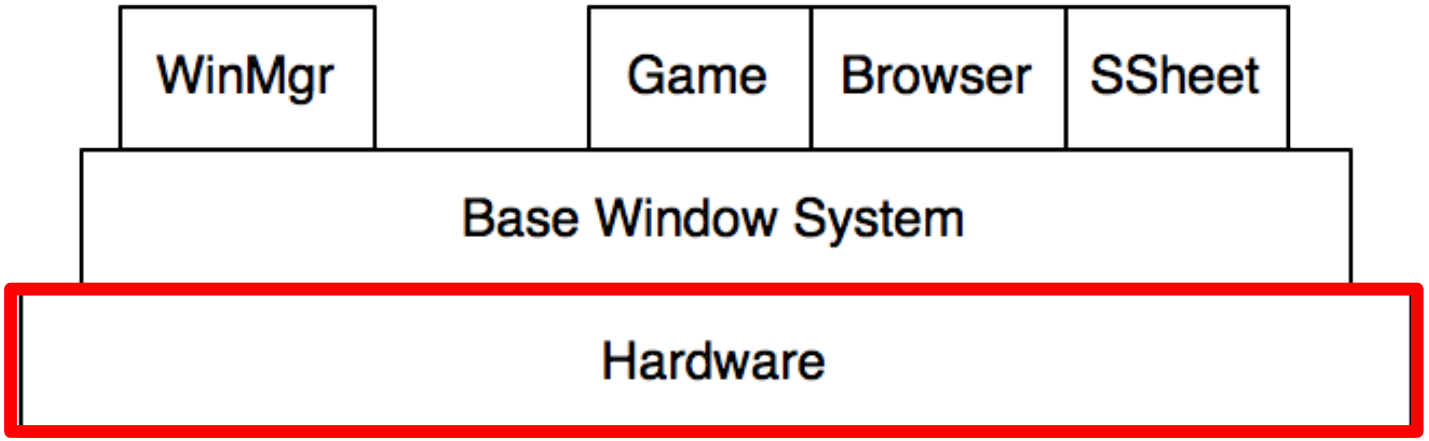
Programs can run simultaneously, each with their own separate window. Windows can exist side-by-side or overlap one another. Input is directed to the correct window by the windowing system.

How to support multiple windows?



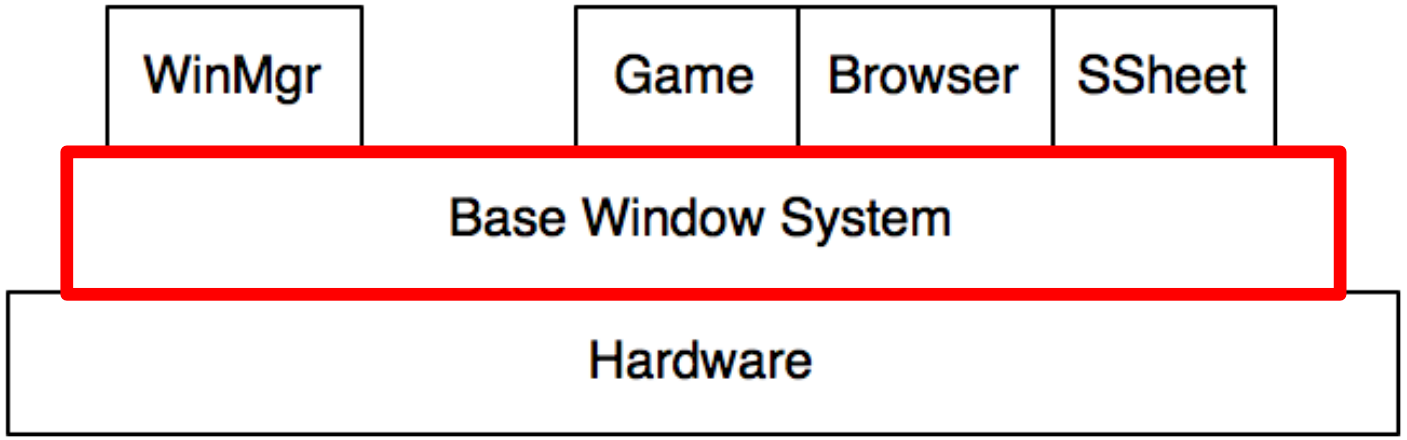
Base Window System (BWS)





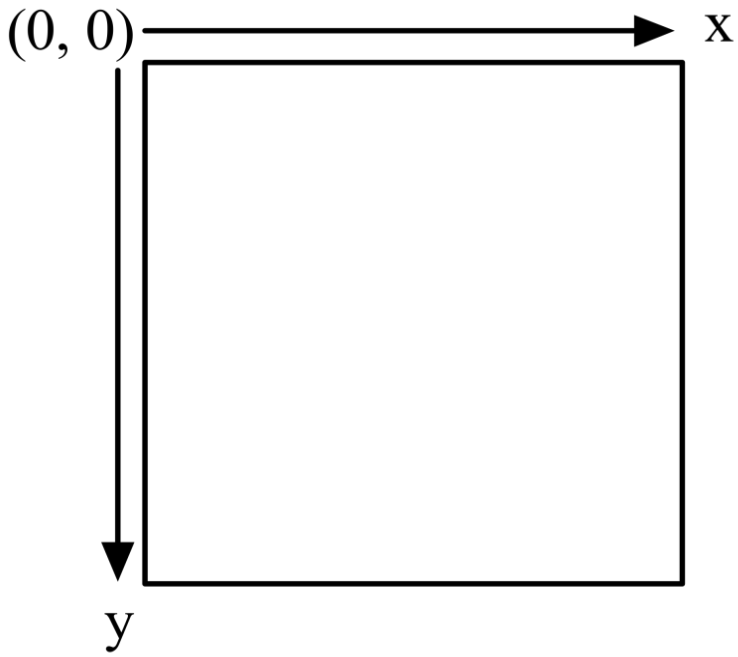
Base Window System

- Lowest level abstraction for windowing system
- Routines for creating, destroying, managing windows
- Routes mouse and keyboard input to correct window
  - only one window “has focus” to receive input
- Ensures only one application changing frame buffer (video memory) at a time
  - one reason why single-threaded / non-thread-safe GUI architectures are popular



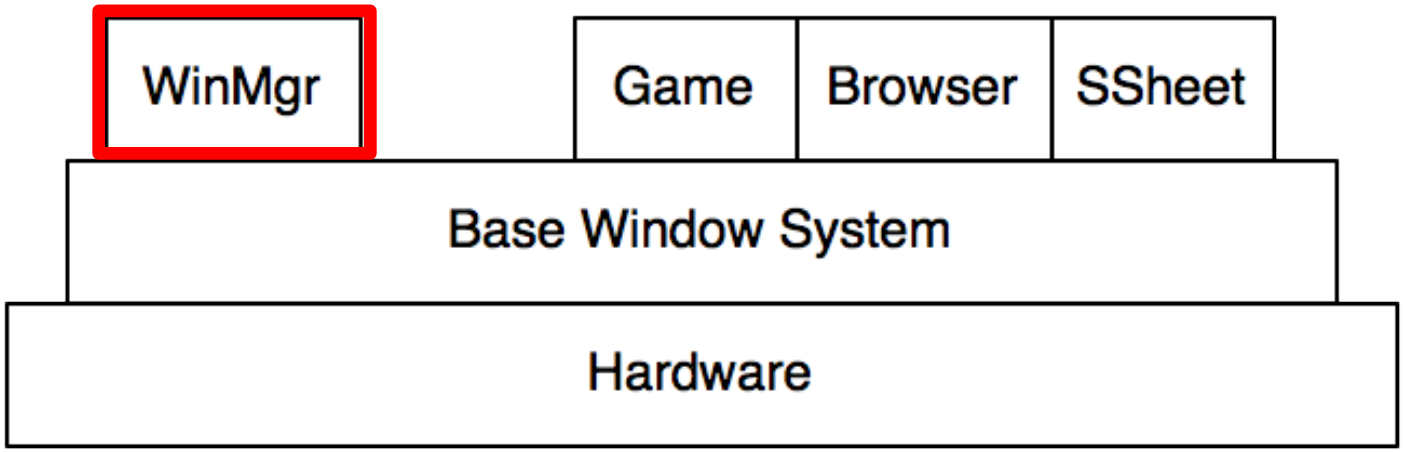
Base Window System

- The BWS provides each program with a window, and manages access to that window.
  - The BWS provides a *drawing canvas* abstraction, where the program can place data.
  - The application is shielded from details of frame buffer, visibility of window, and all other application windows
  - Each window has its own coordinate system
    - BWS transforms between global (screen) and local (window) coordinate systems
    - Each window does not need to worry where it is on screen; program assumes its top-left is (0,0)
  - BWS provides graphics routines to the program for drawing

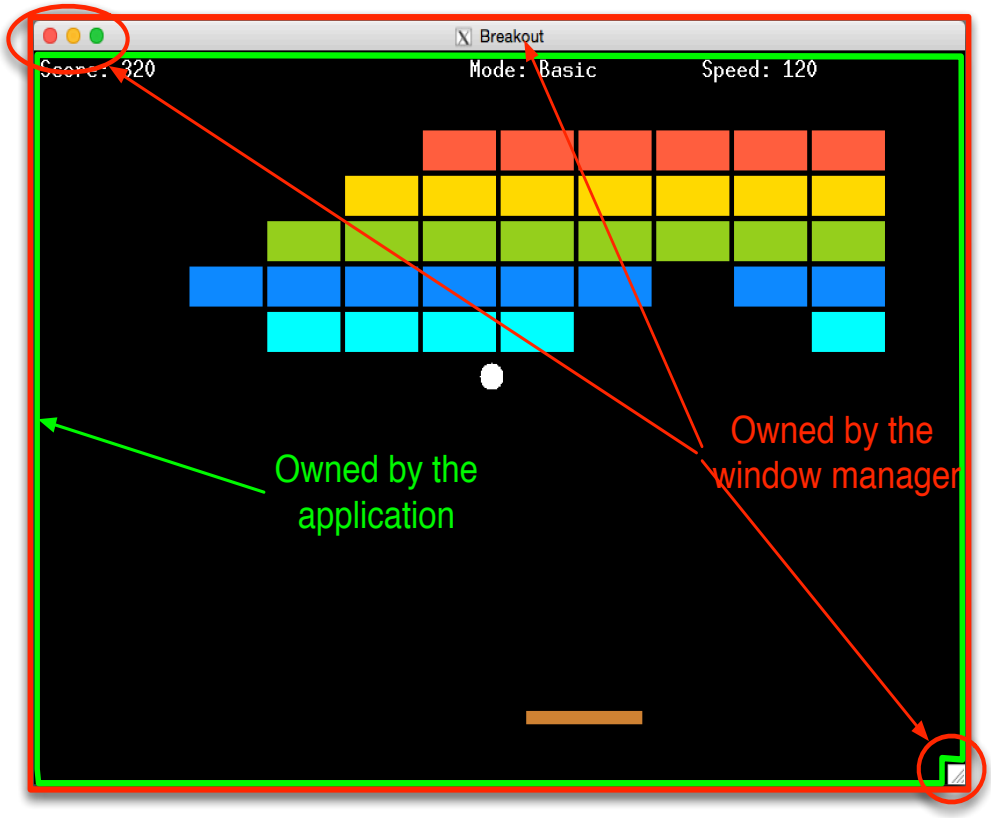


Window Manager

- Provides conceptually different functionality
  - Layered on top of Base Window System
  - Provides interactive components for windows (menus, close box, resize capabilities)
  - Creates the “look and feel” of each window
- Application ”owns” the contents of the window, but the WM “owns” the application window itself!

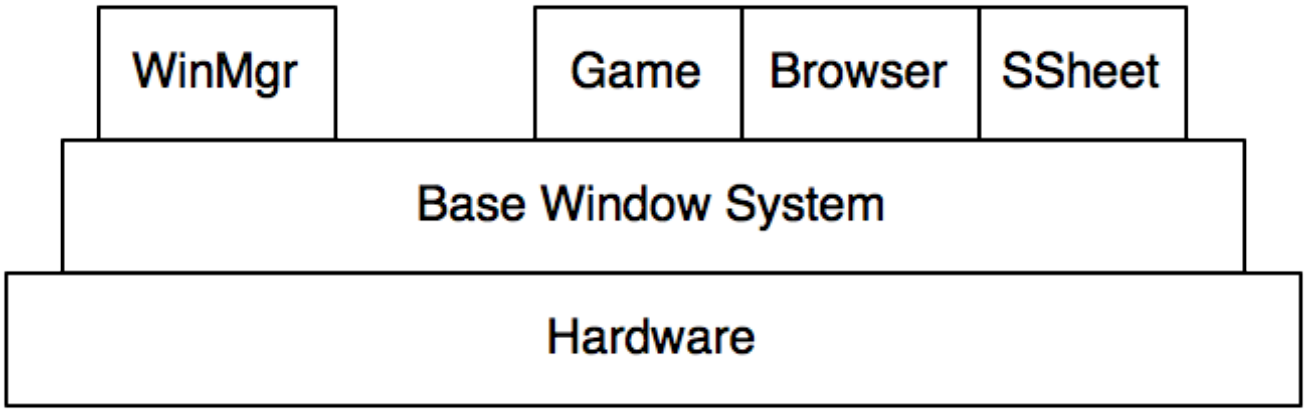


- Application Window vs. Application "Canvas"
  - the window manager owns the window (including its controls)
  - the application owns the canvas

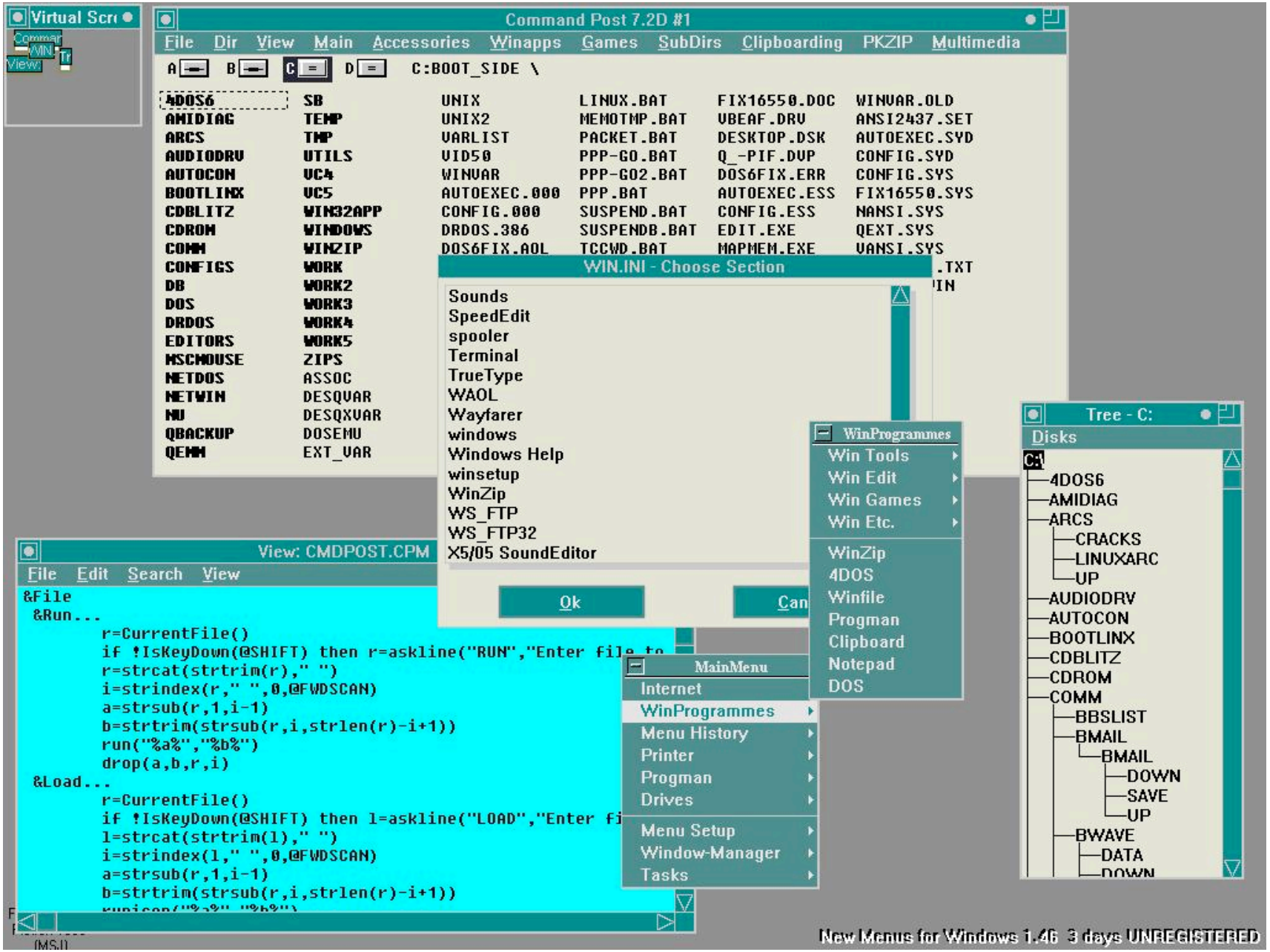


BWS vs. Window Managers

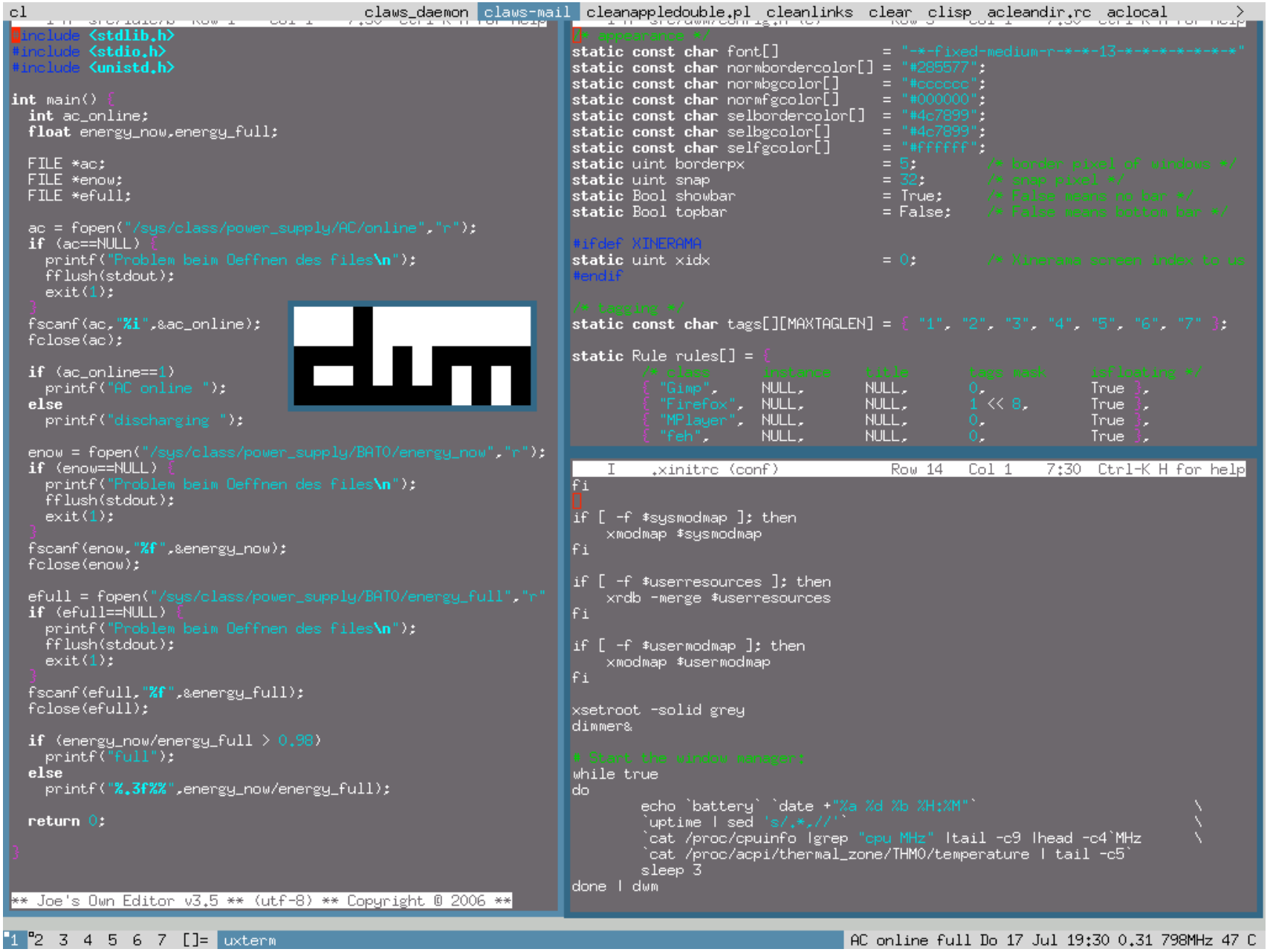
- Separation of Base Window System (BWS) from Window Manager (WM)
  - Enables many alternative “look and feels” for the windowing system (e.g., KDE, GNOME…)
  - One of the keys to its lasting power: can innovate by changing the WM layer
  - Resiliency, since BWS and WM are separate processes



Motif (Stacking)



DWM (tiling)


cl
claws_daemon
claws-mail
cleanappledouble.pl
cleanlinks
clear
clisp
acleandir.rc
aclocal
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
int main() {
int ac_online;
float energy_now,energy_full;
FILE *ac;
FILE *enow;
FILE *efull;
ac = fopen("/sys/class/power_supply/AC/online","r");
if (ac==NULL) {
printf("Problem beim Oeffnen des files\n");
fflush(stdout);
exit(1);
}
fscanf(ac,"%i",&ac_online);
fclose(ac);
if (ac_online==1)
printf("AC online ");
else
printf("discharging ");
enow = fopen("/sys/class/power_supply/BAT0/energy_now","r");
if (enow==NULL) {
printf("Problem beim Oeffnen des files\n");
fflush(stdout);
exit(1);
}
fscanf(enow,"%f",&energy_now);
fclose(enow);
efull = fopen("/sys/class/power_supply/BAT0/energy_full","r"
if (efull==NULL) {
printf("Problem beim Oeffnen des files\n");
fflush(stdout);
exit(1);
}
fscanf(efull,"%f",&energy_full);
fclose(efull);
if (energy_now/energy_full > 0.98)
printf("full");
else
printf("%.3f%%",energy_now/energy_full);
return 0;
}
** Joe's Own Editor v3.5 ** (utf-8) ** Copyright @ 2006 **
/* appearance */
static const char font[] = "-*-fixed-medium-r-*-*-13-*-*-*-*-*-*-*"
static const char normbordercolor[] = "#285577";
static const char normbgcolor[] = "#cccccc";
static const char normfgcolor[] = "#000000";
static const char selbordercolor[] = "#4c7899";
static const char selbgcolor[] = "#4c7899";
static const char selfgcolor[] = "#ffffff";
static uint borderpx = 5; /* border pixel of windows */
static uint snap = 32; /* snap pixel */
static Bool showbar = True; /* False means no bar */
static Bool topbar = False; /* False means bottom bar */
#ifdef XINERAMA
static uint xidx = 0; /* Xinerama screen index to us
#endif
/* tagging */
static const char tags[][MAXTAGLEN] = { "1", "2", "3", "4", "5", "6", "7" };
static Rule rules[] = {
/* class instance title tags mask isfloating */
{ "Gimp", NULL, NULL, 0, True },
{ "Firefox", NULL, NULL, 1 << 8, True },
{ "MPlayer", NULL, NULL, 0, True },
{ "feh", NULL, NULL, 0, True },
I .xinitrc (conf) Row 14 Col 1 7:30 Ctrl-K H for help
fi
if [-f $sysmodmap]; then
xmodmap $sysmodmap
fi
if [-f $userresources]; then
xrdb -merge $userresources
fi
if [-f $usermodmap]; then
xmodmap $usermodmap
fi
xsetroot -solid grey
dimmer&
Start the window manager:
while true
do
echo `battery` `date +"%a %d %b %H:%M"`
`uptime | sed 's/.*,//'`
`cat /proc/cpuinfo |grep "cpu MHz" |tail -c9 |head -c4`MHz
`cat /proc/acpi/thermal_zone/THM0/temperature | tail -c5`
sleep 3
done | dwm
1 2 3 4 5 6 7 []= uxterm
AC online full Do 17 Jul 19:30 0.31 798MHz 47 C

KWin (compositing)

BWS vs. Window Managers

- macOS, Windows combine “BWS” and Window Manager together (or at least, don’t distinguish)
- Trade-offs in approaches?
  - Look and feel…
  - Window management possibilities…
  - Input possibilities…
- Conceptually, on both platforms, there is a separation of canvas (assigned to application) and window decoration/OS overlay handled by window manager
  - Lines do blur when combined, however
  - e.g. Windows fast access menu-bar in the window frame

Summary

- X Windows design goals
- Basic X architecture: client, server, network
- Windows: opening, disposing
- Base Window System vs. Window Manager
- Swapping Window Managers