Linux User Guide (VI)

GUI in UNIX/Linux

Cristian Vidrașcu

Outline

- Introduction
- The classical model: X Window System
- The window manager
- The future model: Wayland vs Mir

Introduction

- **X Window System** – the module responsible for displaying the graphical interface in most Unix / Linux systems (it is a specification, not a program)

  X – successor to *W Window System*

  Window – the central element in any graphical interface

  System – consisting of several components

- The specification is independent of the hardware platform
- History: project started in 1984 at MIT
- Current Coordinator: Foundation X Org

Introduction

- Reference implementation – Xfree86 / X.Org
- The current version: X11R7.7
- Version history: see on www.x.org
- Other commercial or free implementations exist, including for Microsoft Windows
- X is not the only graphical interface model for Unix systems, but it is the only widely accepted one

  Other models: W Window System (forerunner of X), NeWS (Network extensible Window System), NeXT Display PostScript, Berlin/Fresco, Y Window System, etc.

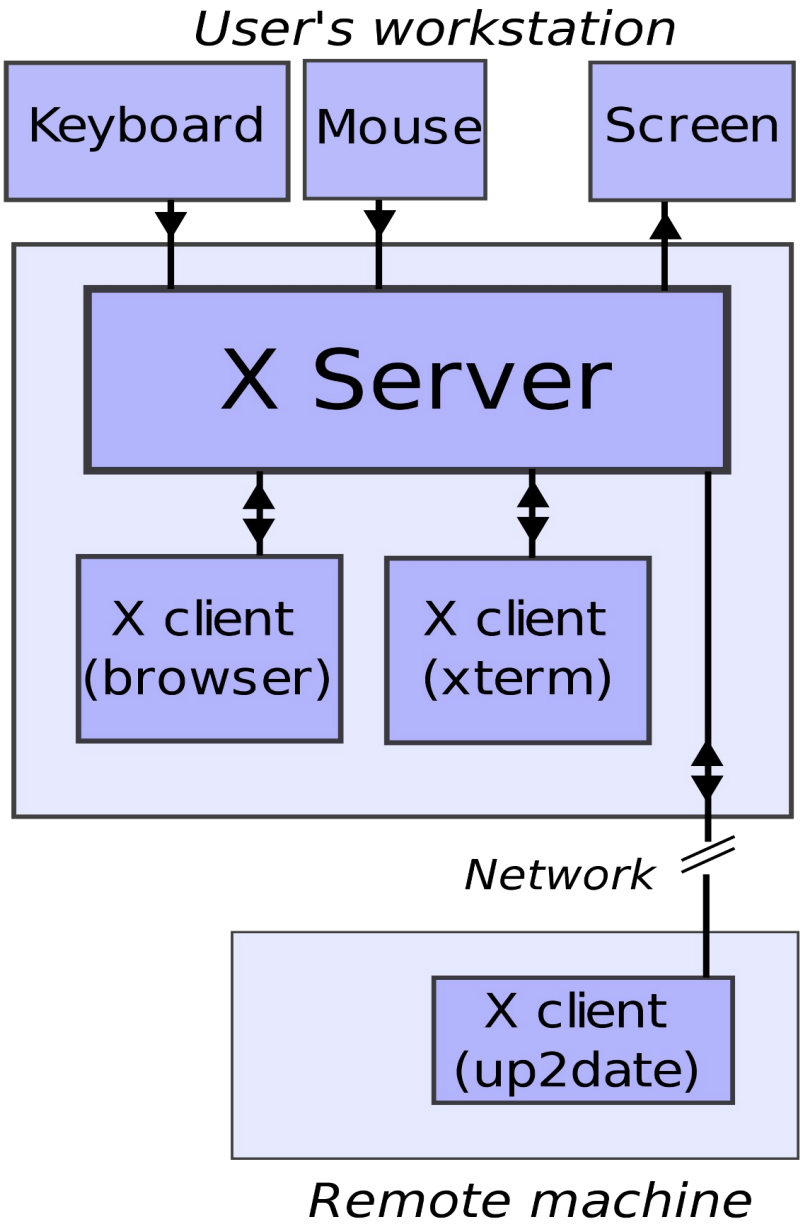
The classical model:

X Window System

Concepts

- *Console* – generally consists of screen, keyboard and mouse
- *Drivers* – for communication with peripherals
- *X Server* – a process responsible for window, font and cursor management
- *X Client* – application that uses windows managed by the X server
- *Protocol* – set of rules for communication between clients and the server
- *Manager* – window manager
- *Toolkit* – graphics component library
- *X Terminal* – a terminal with an X server installed, which communicates with clients possibly on other computer systems

Client – Server



- The client connects to the server, requesting the display of windows (the connection is made through local channels or sockets)
- The server processes client requests and displays the results on the screen
- Keyboard and mouse events are notified to clients

The X Server

- It acts as an intermediary between programs and hardware devices for communication with the user
- Provides support for multiple client applications
- Provides support for multiple consoles
- Manage fonts – XFontServer
- Initialized with the xinit command
- It does not provide advanced interaction components
- Provides support for communication between client applications (common clipboard, drag&drop between apps, etc.)

X applications (X clients)

- X client = any application that uses the X protocol to communicate with an X server
- It does not use hardware resources to create its GUI, but instead calls X graphics server methods
- No programming language, operating system or hardware platform restrictions are imposed
- It may be on a different computer than the X server

Starting the X server

- xinit

  Starts the X server and processes the ~/.xinitrc file

  If it has the path to a program as a parameter, executes that program.

  It is used by other ways to start the X server as a starting point:

- startx

  Starts the X server and a window manager for the current user

- init 5

  Starts the X server and a window manager with a login window

Text and graphics consoles

- Linux is a multi-user OS → allows simultaneous work with:
- 6 text consoles, local (i.e. 6 work sessions in text mode)

  Switch between them with key combinations Ctrl+Alt+F1,...,Ctrl+Alt+F6.

  Terminals are referred to by name tty1,...,tty6.
- up to 6 graphic consoles, local

  Switch between them with key combinations Ctrl+Alt+F7,...,Ctrl+Alt+F12.

  An X server must be started in each of them:

  `startx -- :0` (or, in short, `startx`)

  `startx -- :1`

  ...

  `startx -- :5`

Note: in some distributions, the 6th graphical console is used by the kernel to continuously display status messages, error messages, etc.

Basic graphics elements: Windows

- The only element through which interfaces are created
- Any interface component is created using a number of windows
- Organized hierarchically in a tree structure
- There is only one root window: “the root window”
- There can also be X servers without a root window – “rootless server”, for example if that X is not the main graphics server but is started as an application in another visual environment (for running X applications in MacOS X or Windows environments)

Other basics elements: Events

- Events = messages sent by the server to the client to notify that something of interest to the application has occurred
- Events from input devices – mouse, keyboard
- Visual events – hiding or re-showing a window
- Events sent by one application to another application's window – inter-application communication
- An application specifies the types of events it wants to be notified of
- The *event-driven* programming model

The window manager

- It is a special X client
- Responsible for displaying window "decorations" (title bar, system buttons, resizing borders, etc.)
- Responsible for providing a *desktop* (workspace) and managing active applications
- Only one window manager can be active at a time

The window manager

- Classification by how windows are drawn and updated on the screen:
  - *stacking window manager* – overlapping windows
  - *tiling window manager* – adjacent windows
  - *compositing window manager* – advanced 2D and 3D visual effects (e.g. Compiz/Beryl)

The window manager

- There is a wide range of window managers, from minimal ones to true application suites

- Classification by complexity:
  - *window managers* (simple): controls the placement and appearance of windows within the GUI
  - *desktop environments*: include a window manager plus a file manager, a set of visual themes and some desktop management programs

The window manager

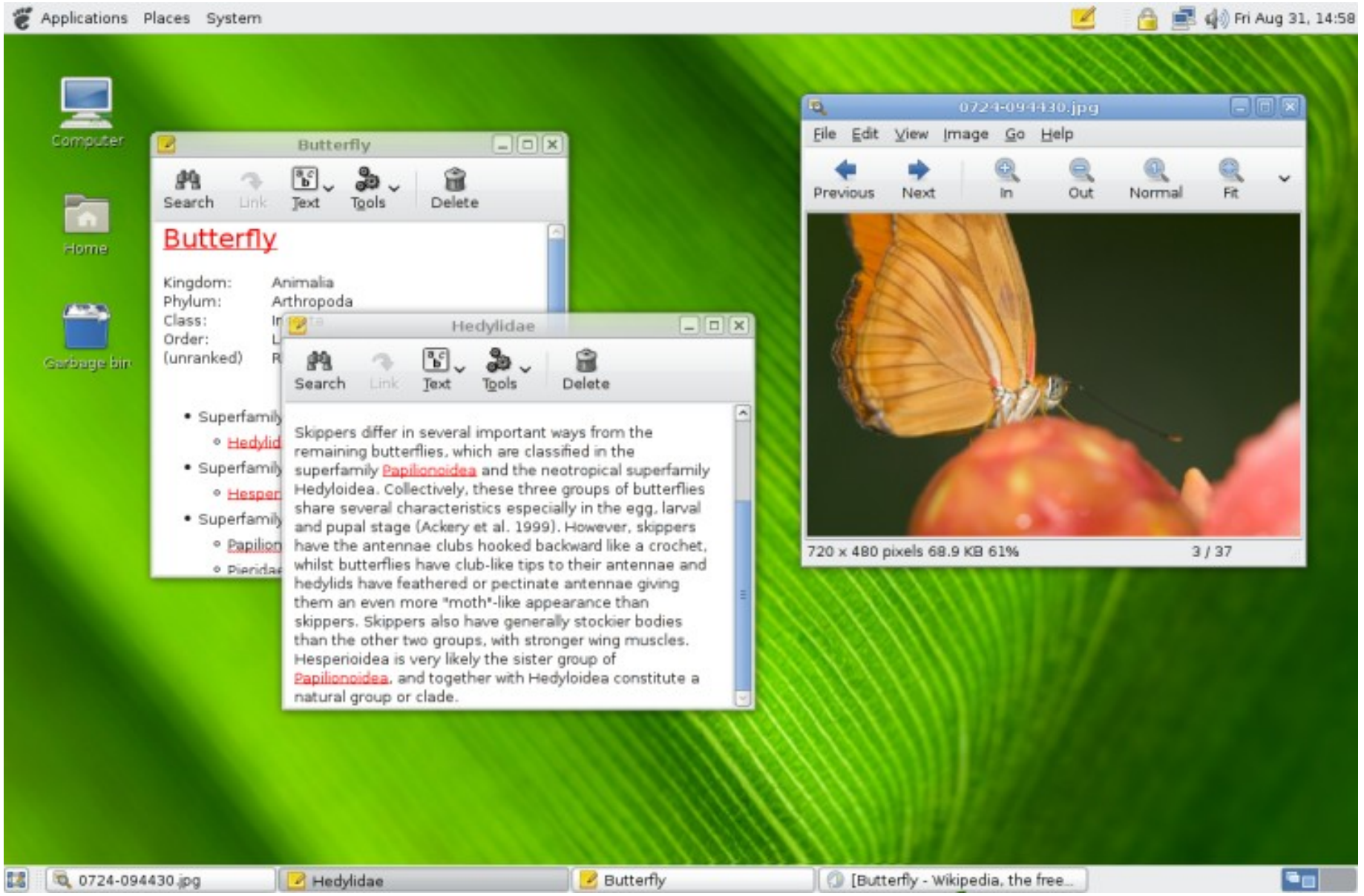
- Most used desktop environments:
  - GNOME
  - KDE
- Other examples: Xfce, Enlightenment, CDE, etc.

- Some desktop environments are configurable in terms of the components they use

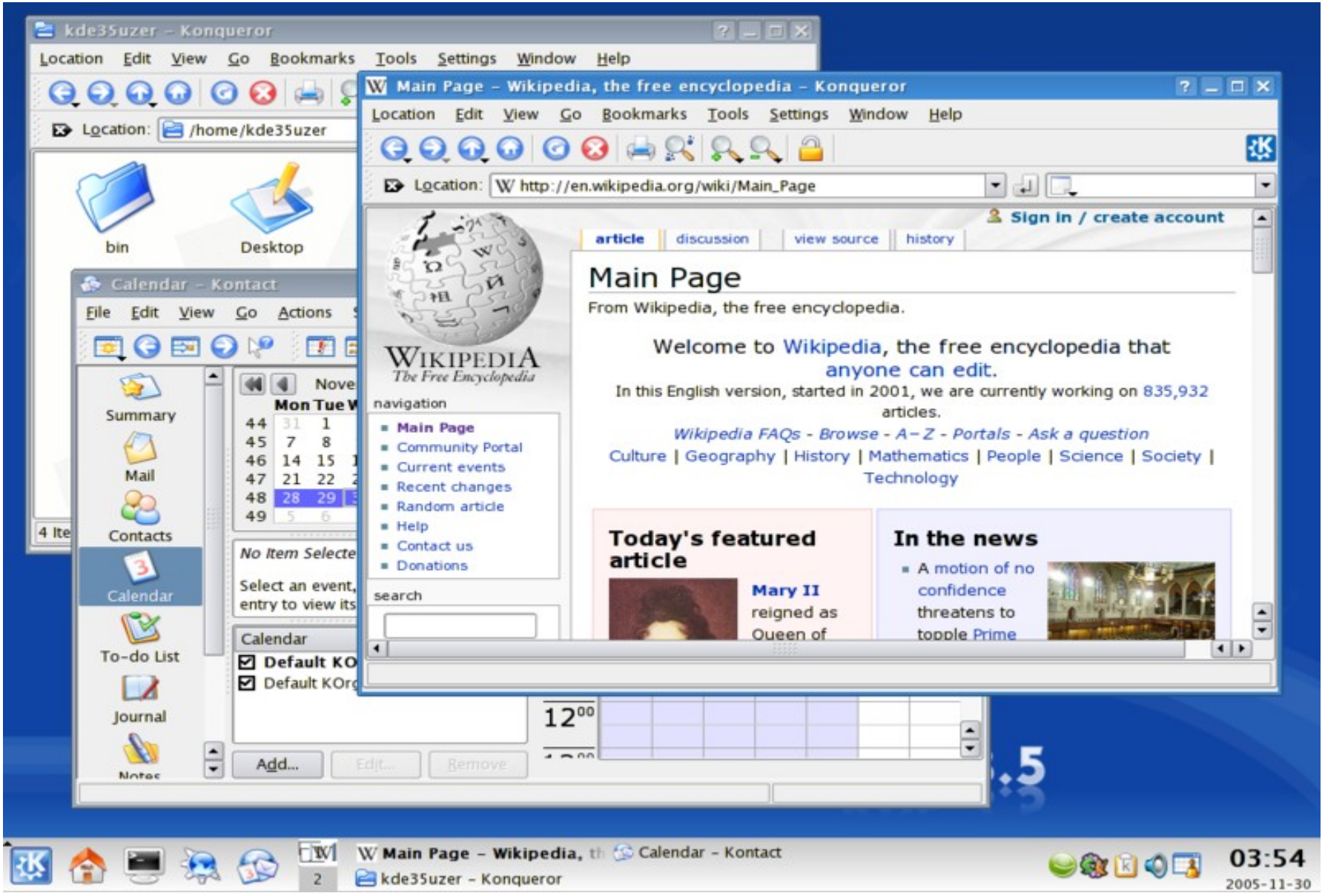
  Example: GNOME can be configured to use Compiz as a window manager instead of Metacity (i.e. the default, stacking window manager type).

  Similarly, in the KDE desktop environment we can use Kwin or Compiz as compositing window manager instead of the classic one.

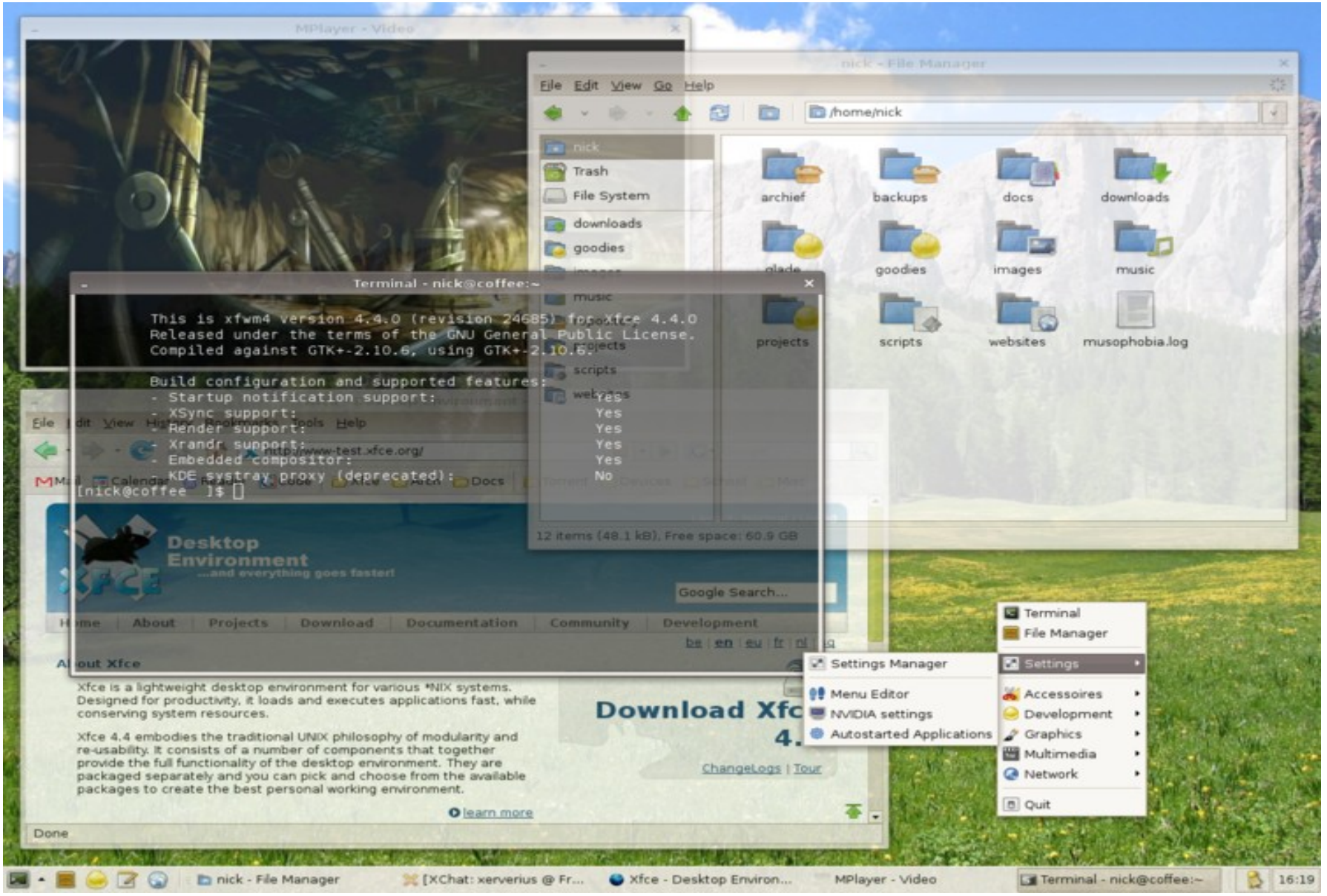
Screenshot – GNOME 2.20 with Metacity



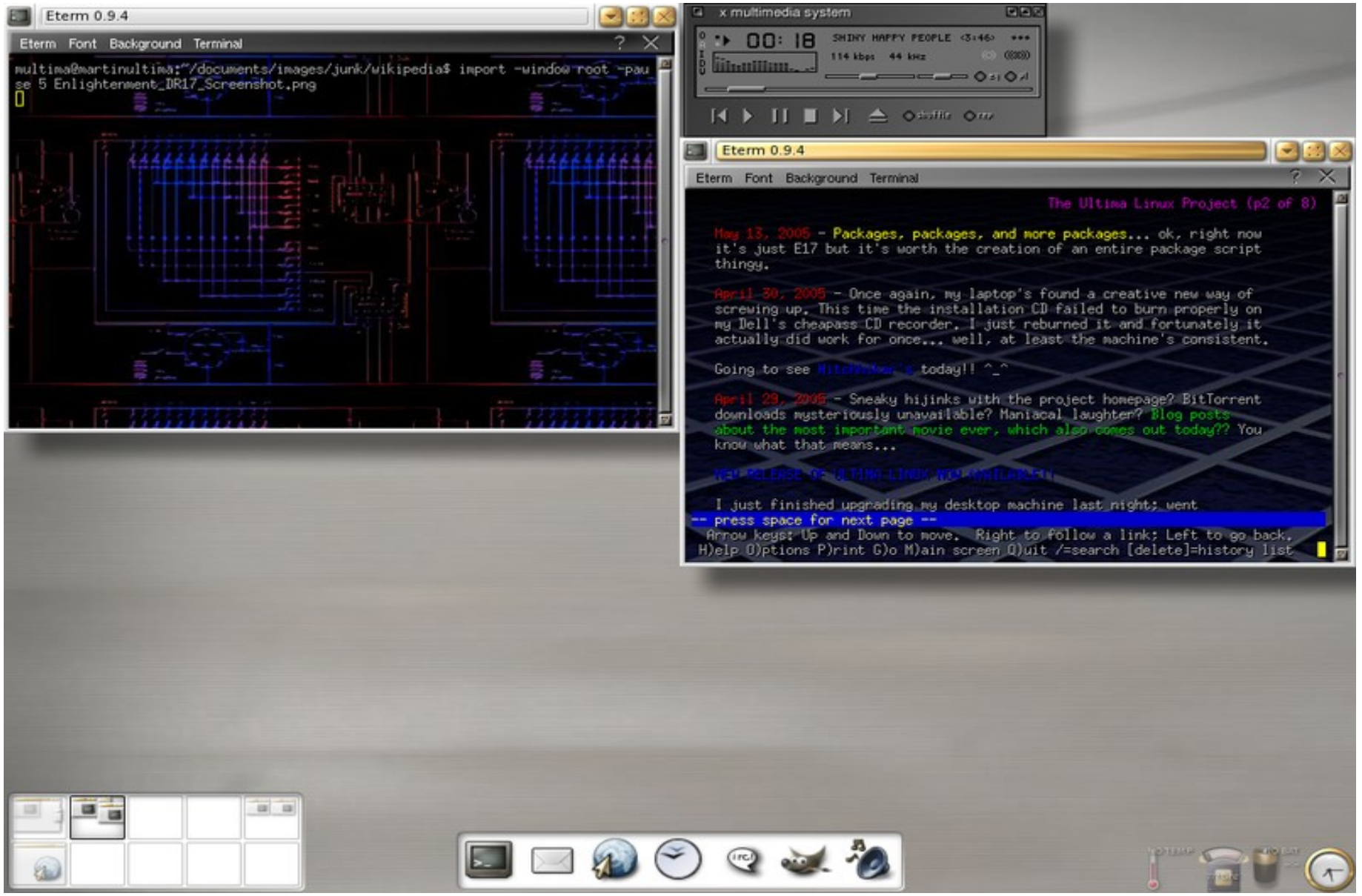
Screenshot – KDE 3.5



Screenshot – Xfce



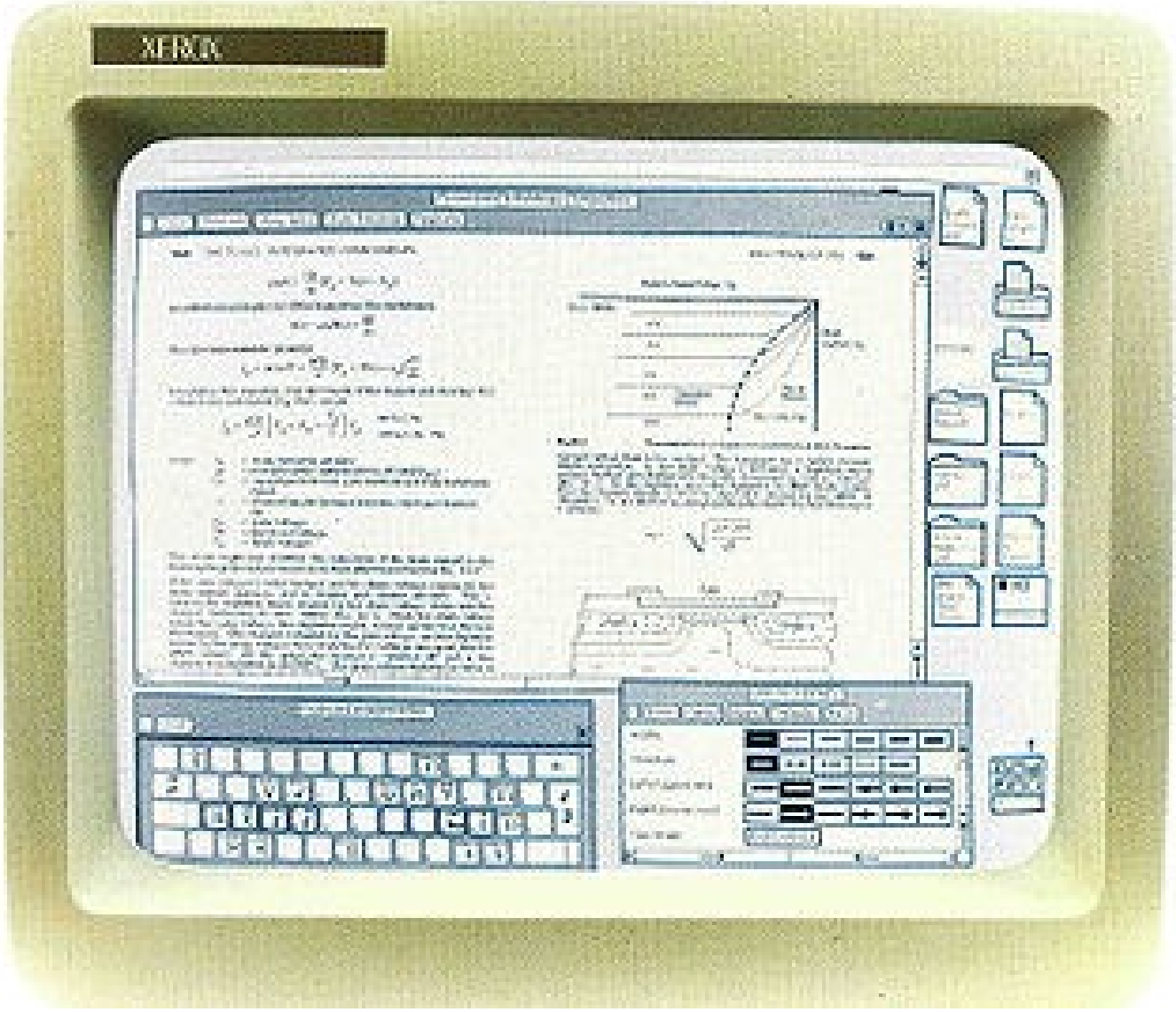
Screenshot – Enlightenment



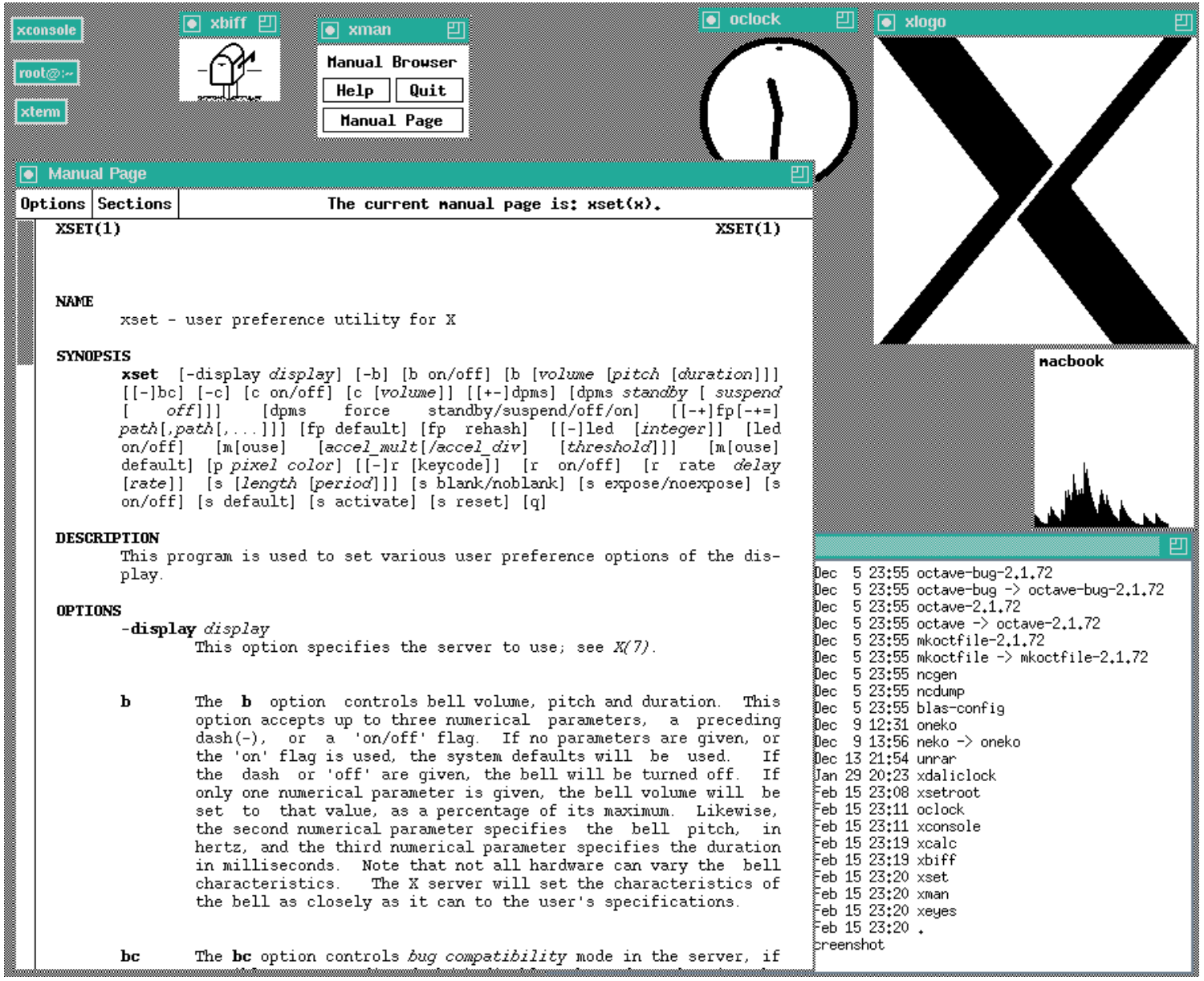
History: The first mouse @SRI

(Stanford Research Institute)

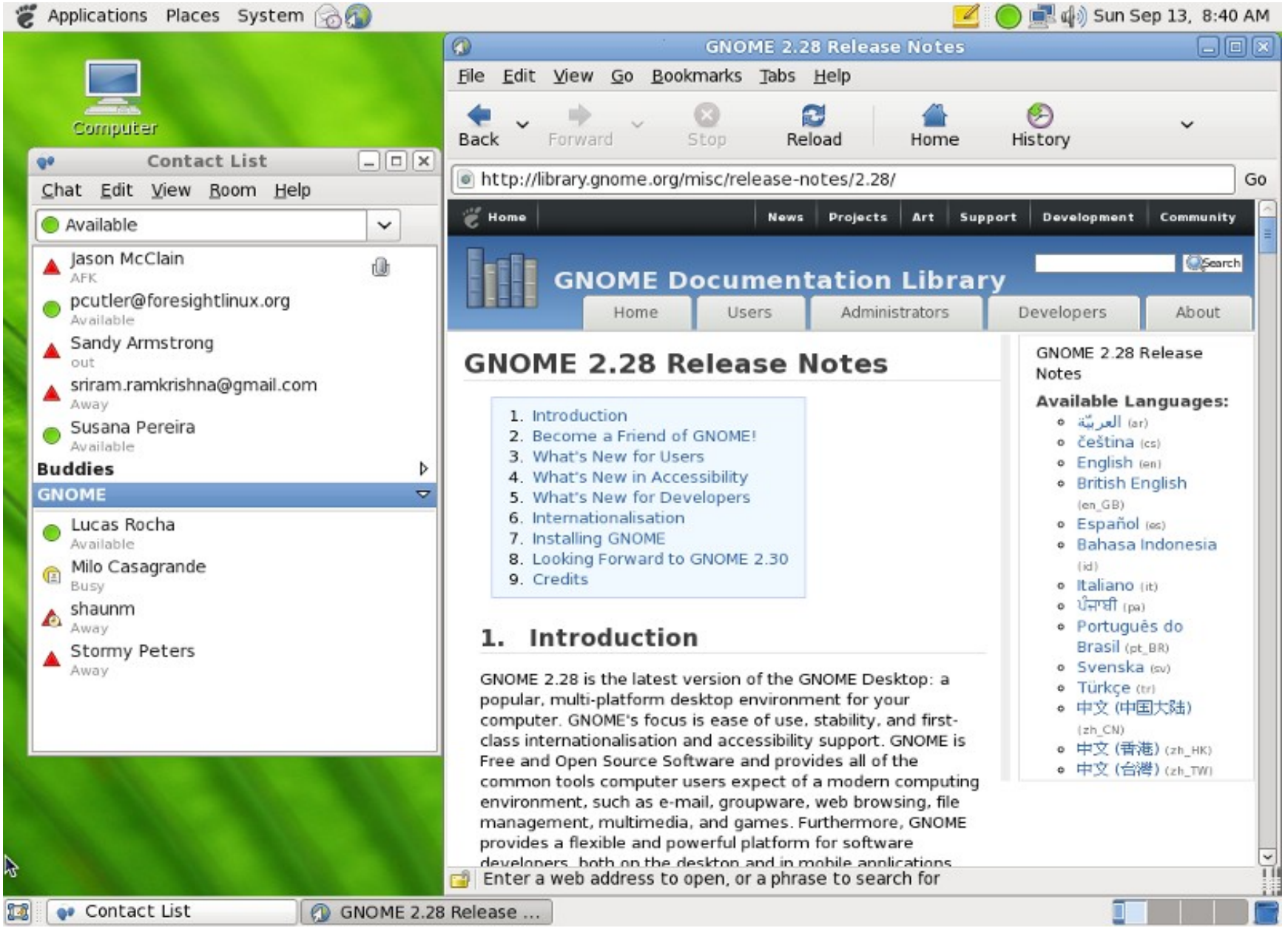
History: The first GUI @ Xerox PARC



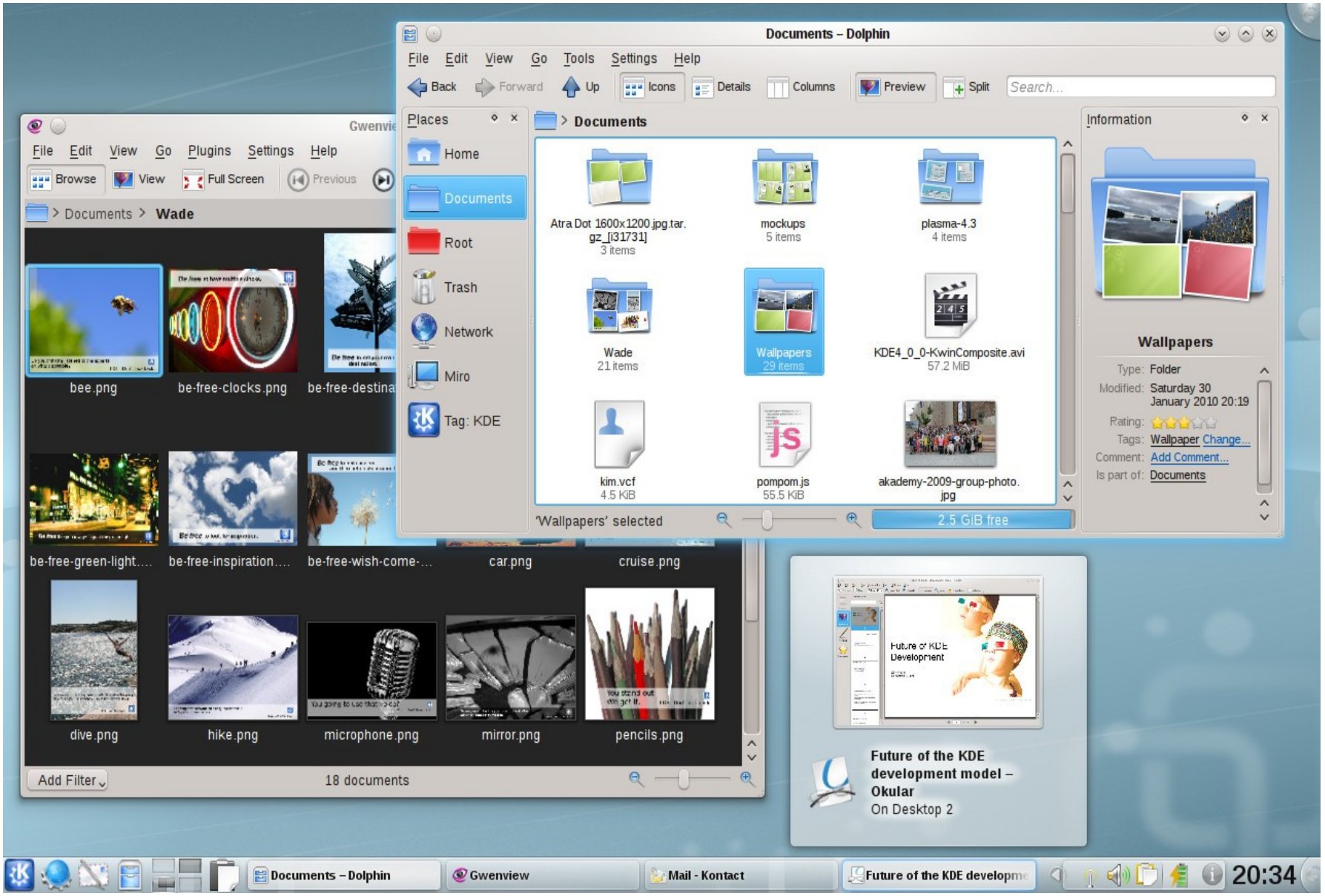
History: A Unix based X Window System desktop (circa 1990)



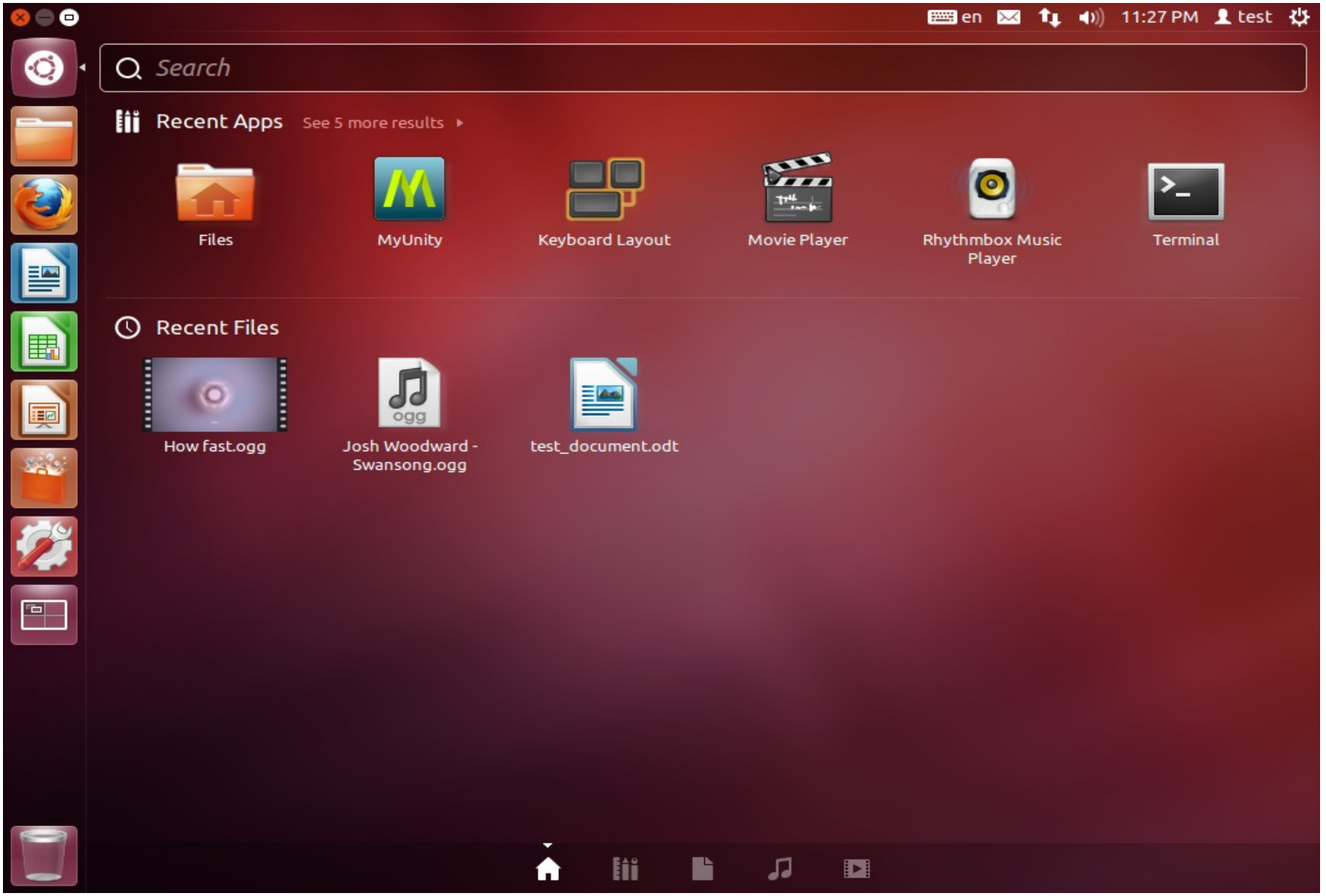
2010: GNOME 2.28 desktop



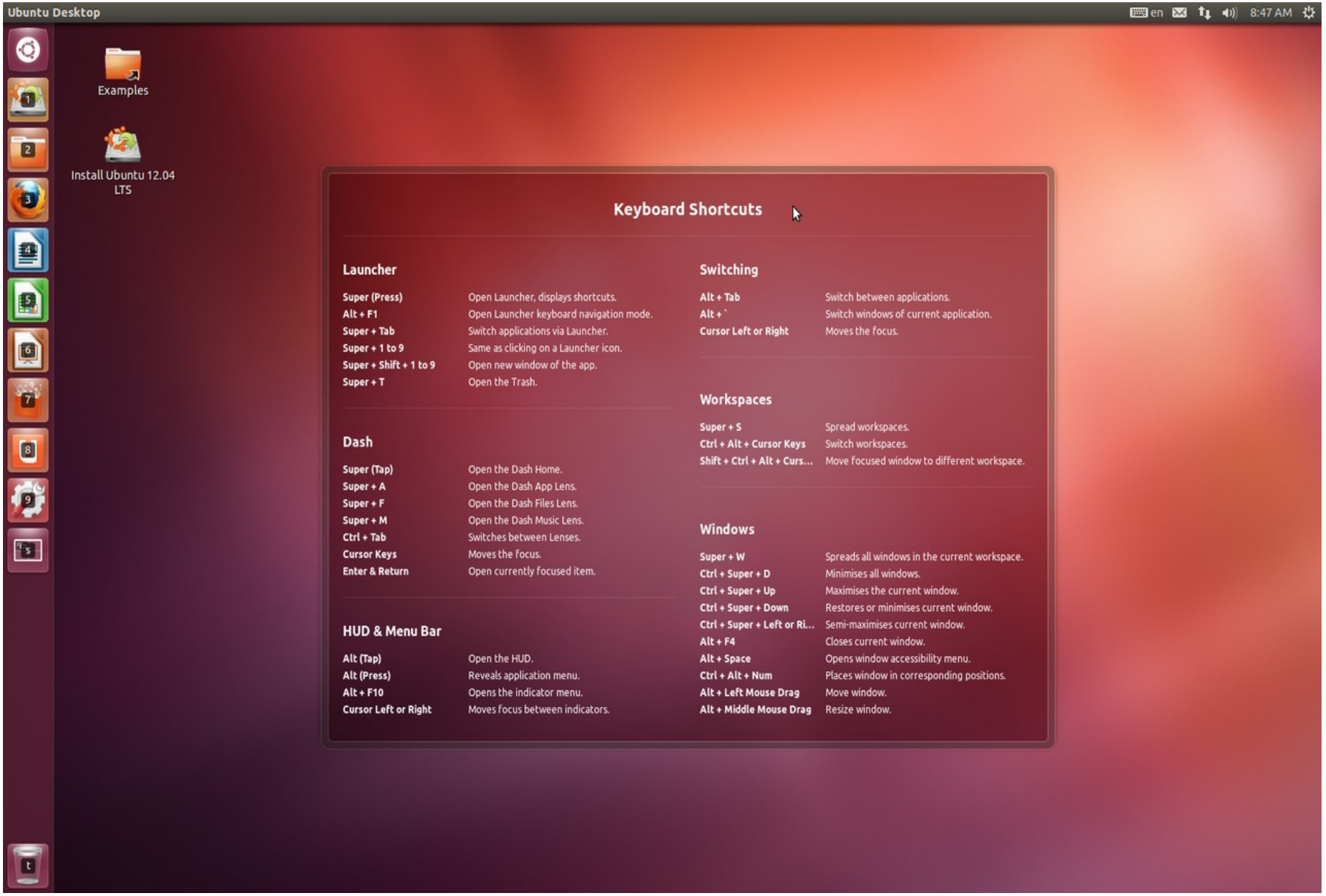
2010: KDE Plasma 4.4 desktop



2012: Unity – shell interface in Ubuntu 12.04



2012: Unity – shell interface in Ubuntu 12.04



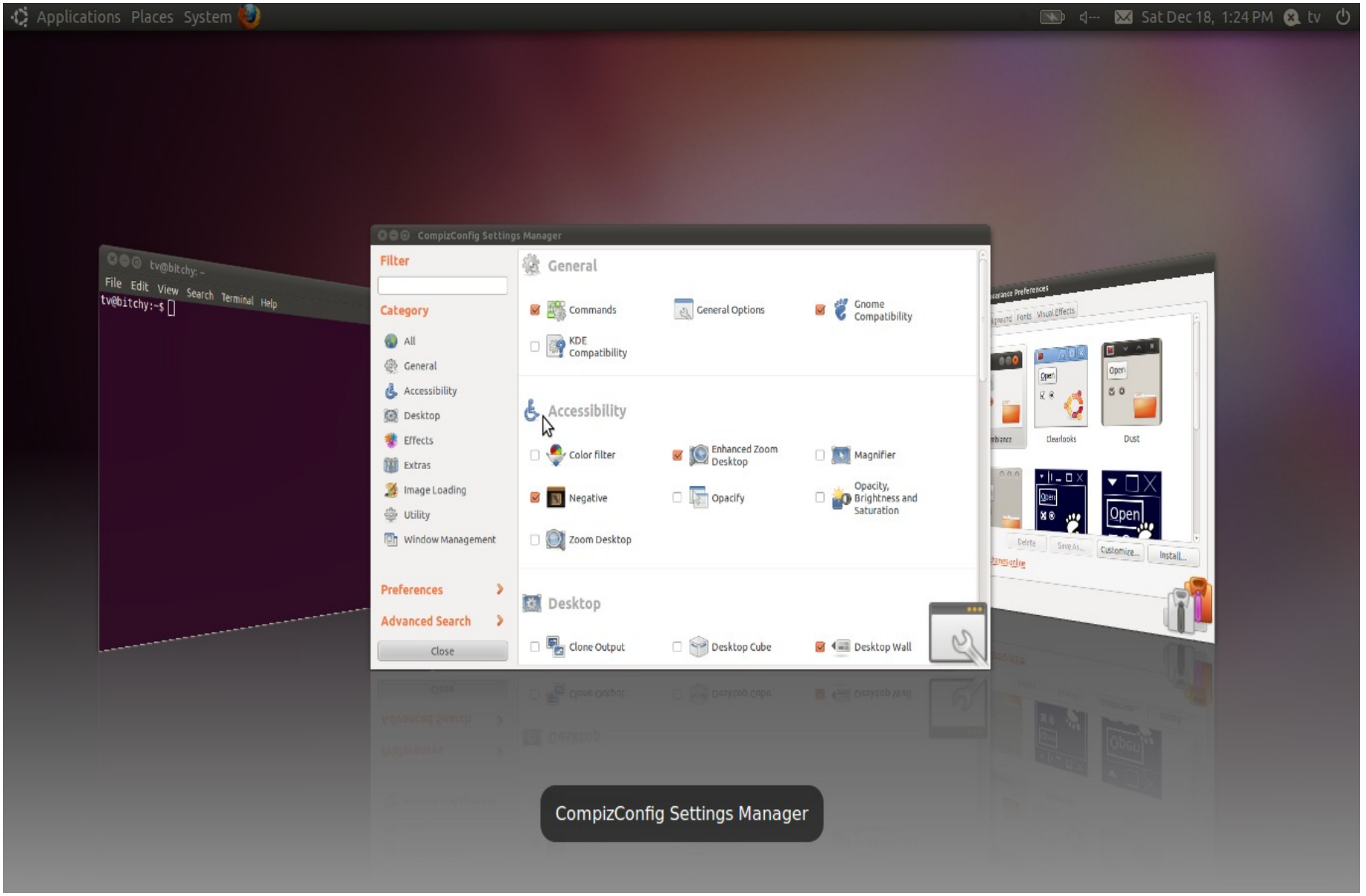
3D GUI

- Compositing window manager
  - A window manager that provides applications with an off-screen buffer to store application windows, then composes all applications' buffers into a single image and copies it to graphics card memory
  - In addition, it additionally processes application window buffers, applying various 2D and 3D visual effects / animations such as: blending, fading, scaling, rotation, duplication, bending and contortion, shuffling, blurring, etc.

3D GUI

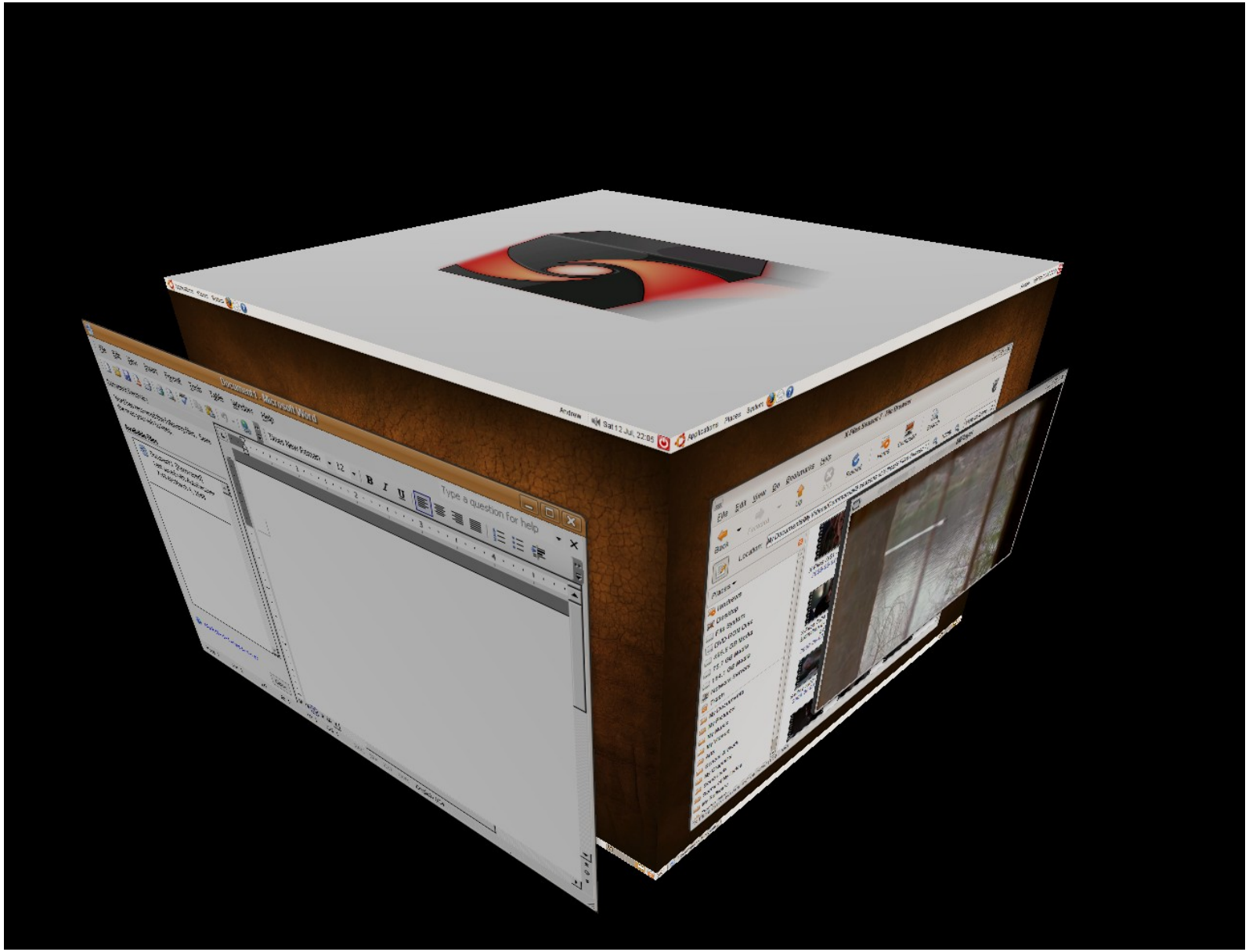
- Compiz = a compositing window manager for Linux
  - Installation in Ubuntu 20.04: compiz, compiz-plugins-extra and compizconfig-settings-manager packages
  - Compiz plugins include special effects such as:
    - the desktop cube effect
    - Alt-Tab application-switching with live previews or icons
    - the Exposé effect (the scale plugin in Compiz)
    - the fire effect, the rain effect, etc.

Compiz: 3D effects configuration panel



Compiz running on Fedora Core 6 with AIGLX
(the desktop cube effect)

Compiz+GNOME running on Ubuntu
(the desktop cube effect)



Kwin 4.4 in KDE
(the desktop cube effect)

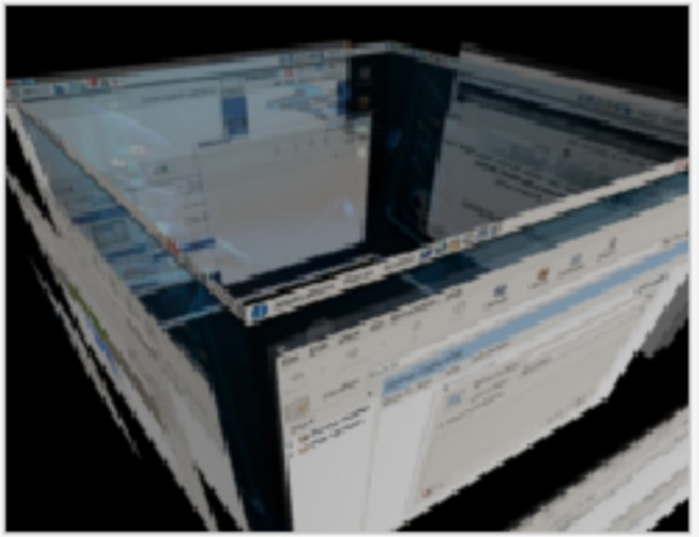
Compiz: the screen magnifier effect

article | discussion | edit this page | history | move | unwatch

Compositing window manager

From Wikipedia, the free encyclopedia

A **compositing window manager** is a component of a computer's graphical user interface that draws windows and/or their borders. It also controls how they are displayed and interact with each other, and the rest of the desktop environment. The main difference between compositing window manager and a standard window manager is that instead of outputting to a common screen, programs each output first to a separate and independent buffer, or temporary location inside the computer, where they can be manipulated before they are shown. [1] [2]

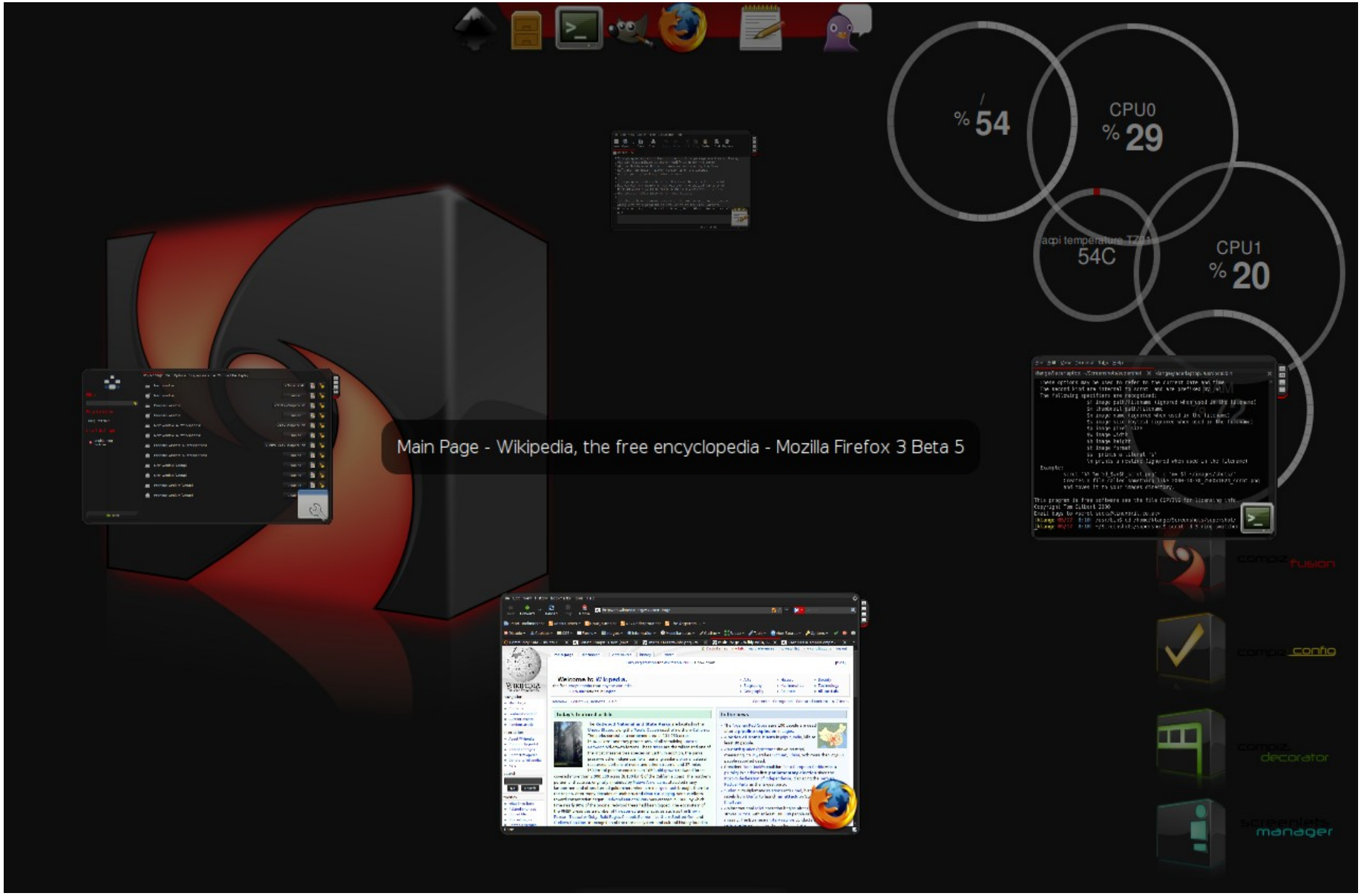


Compiz, a compositing window manager, running on Fedora Core 6

Compiz: the shift switcher in flip mode (Ubuntu 8.04)

Kwin: the shift switcher in cover mode

Compiz: the shift switcher in ring mode



The future model

- Wayland – a project started in 2008 as a replacement for the X Window System, but only for Linux systems
  (it uses certain functionality built into the Linux kernel)

  Wayland : display server protocol and a library that implements it

  Weston : a reference implementation of a composer for Wayland

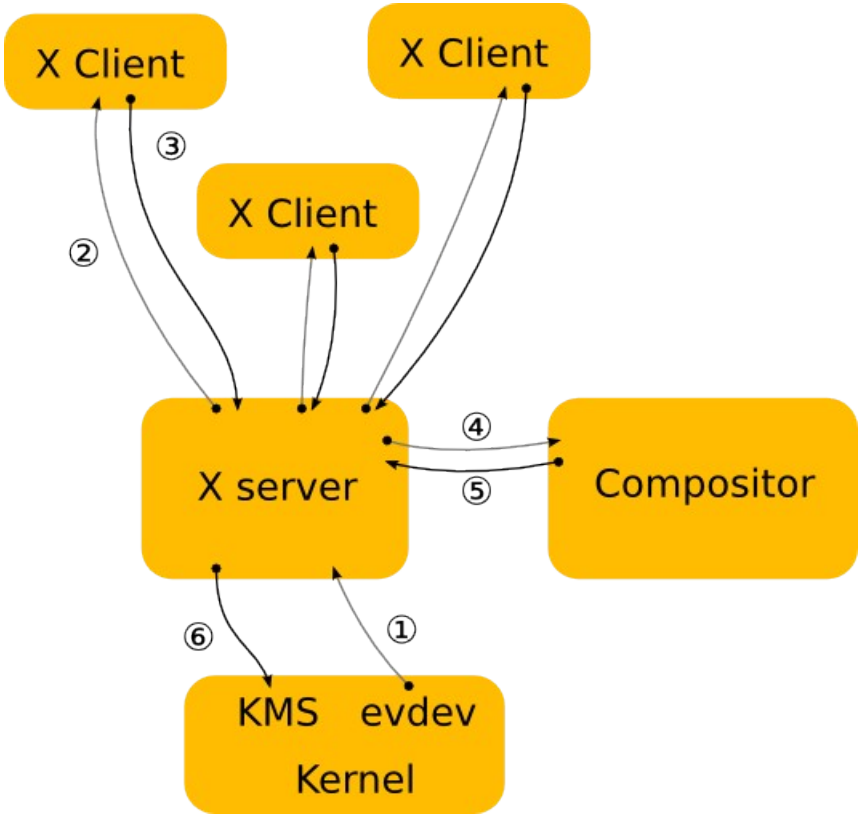
  Status: under development and currently used by some Linux distributions

- Mir – a project announced by Canonical Ltd. in March 2013, as a replacement in future versions of Ubuntu for the X Window System, instead of Wayland

  Status: only used by Unity v8, which Ubuntu dropped in 2017 in favor of GNOME. However, Canonical continues to develop Mir for use in IoT applications.

Wayland vs. X Window System

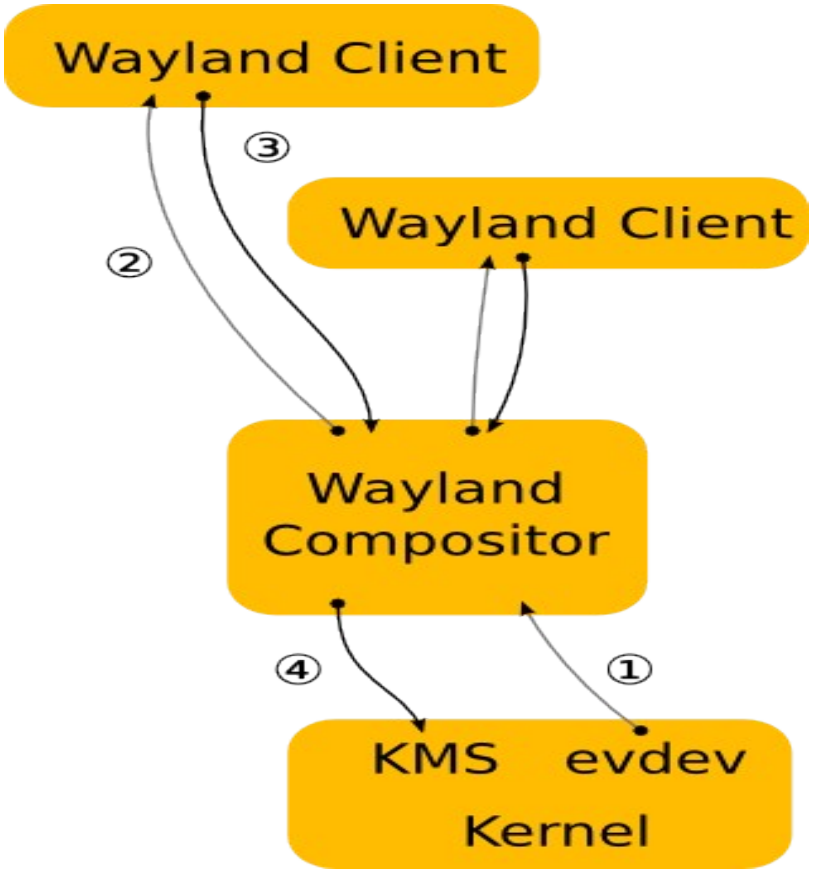
- The X Window System architecture



1. The kernel gets an event from an input device and sends it to X through the evdev input driver. The kernel does all the hard work here by driving the device and translating the different device specific event protocols to the linux evdev input event standard.
2. The X server determines which window the event affects and sends it to the clients that have selected for the event in question on that window. The X server doesn't actually know how to do this right, since the window location on screen is controlled by the compositor and may be transformed in a number of ways that the X server doesn't understand (scaled down, rotated, wobbling, etc).
3. The client looks at the event and decides what to do. Often the UI will have to change in response to the event - perhaps a check box was clicked or the pointer entered a button that must be highlighted. Thus the client sends a rendering request back to the X server.
4. When the X server receives the rendering request, it sends it to the driver to let it program the hardware to do the rendering. The X server also calculates the bounding region of the rendering, and sends that to the compositor as a damage event.
5. The damage event tells the compositor that something changed in the window and that it has to recomposite the part of the screen where that window is visible. The compositor is responsible for rendering the entire screen contents based on its scenegraph and the contents of the X windows. Yet, it has to go through the X server to render this.
6. The X server receives the rendering requests from the compositor and either copies the compositor back buffer to the front buffer or does a pageflip. In the general case, the X server has to do this step so it can account for overlapping windows, which may require clipping and determine whether or not it can page flip. However, for a compositor, which is always fullscreen, this is another unnecessary context switch.

Wayland vs. X Window System

- The Wayland architecture



1. The kernel gets an event and sends it to the compositor. This is similar to the X case, which is great, since we get to reuse all the input drivers in the kernel.
2. The compositor looks through its scenegraph to determine which window should receive the event. The scenegraph corresponds to what's on screen and the compositor understands the transformations that it may have applied to the elements in the scenegraph. Thus, the compositor can pick the right window and transform the screen coordinates to window local coordinates, by applying the inverse transformations. The types of transformation that can be applied to a window is only restricted to what the compositor can do, as long as it can compute the inverse transformation for the input events.
3. As in the X case, when the client receives the event, it updates the UI in response. But in the wayland case, the rendering happens in the client, and the client just sends a request to the compositor to indicate the region that was updated.
4. The compositor collects damage requests from its clients and then recomposites the screen. The compositor can then directly issue an ioctl to schedule a pageflip with KMS.

Bibliography

- X Window System:

  https://www.x.org/

  https://en.wikipedia.org/wiki/X_Window_System

  https://en.wikipedia.org/wiki/List_of_display_servers

- Wayland:

  https://wayland.freedesktop.org/

  https://en.wikipedia.org/wiki/Wayland_(display_server_protocol)

- Mir:

  https://mir-server.io/

  https://en.wikipedia.org/wiki/Mir_(software)