Developing with GTK+

Owen Taylor

Red Hat Software

otaylor@redhat.com

August 22, 1999

Outline

- Introducing GTK+
  An introduction to GTK+ and a look at a simple program.
- Principles of GTK+
  The ideas you need to know to build a GTK+ pprogram.
- Building an Application
  Applying what we've learned to a real program.
- Beyond GTK+
  Related libraries, language bindings, extending GTK+.

Part I: GTK+ Basics

History
Background
Hello World
Basic Concepts
Compiling a GTK+ program

History of GTK+

Fall 1996 GTK+ started as part of the GIMP project by Spencer Kimball and Peter Mattis.

Spring 1998 GTK+ version 1.0 released.
GIMP version 1.0 released

Winter 1999 GTK+ version 1.2 released

Spring 1999 GNOME version 1.0 released

Summer 1999 Development of version 1.4 of GTK+

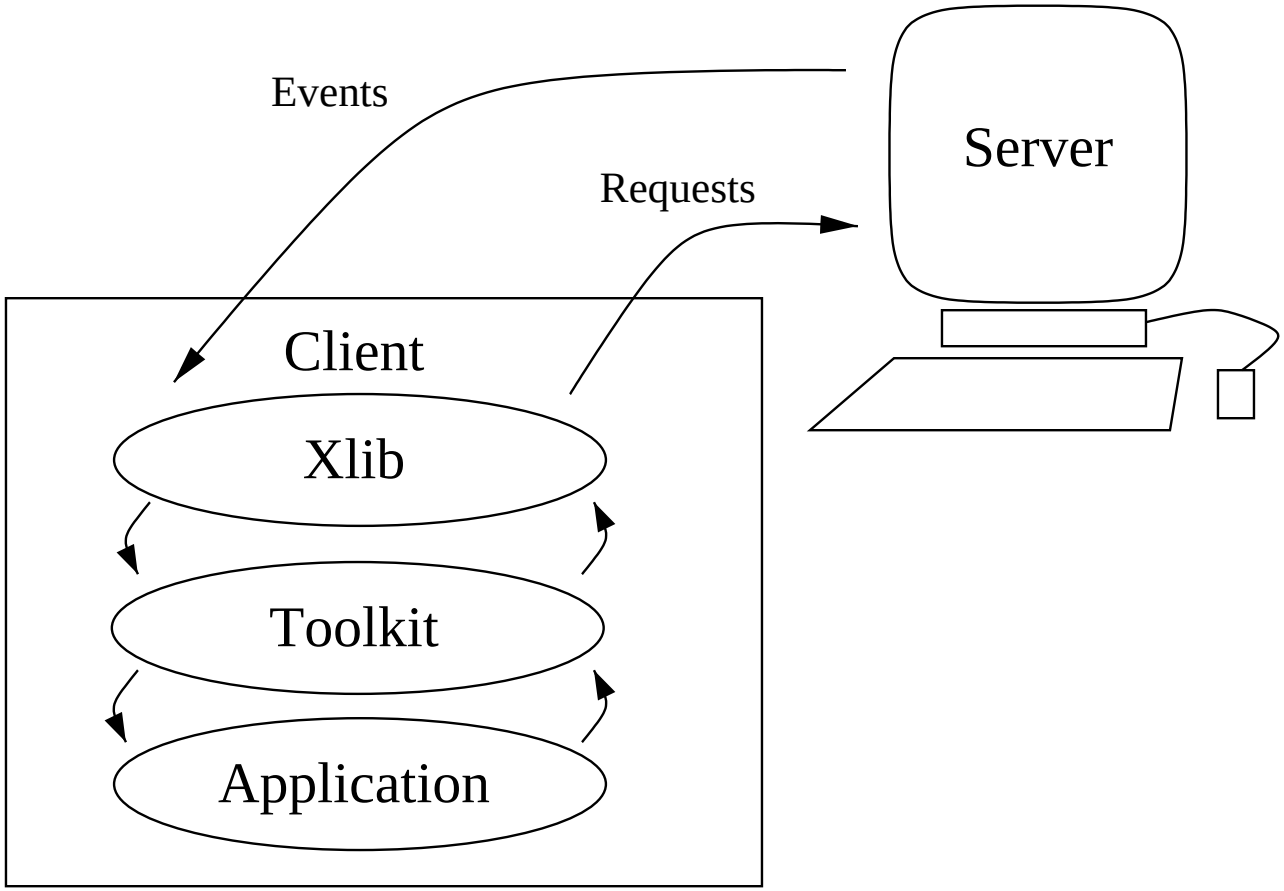
Some projects using GTK+

- GIMP (of course...)
- GNOME
- Mozilla
- AbiWord
- Approx. 500 other free and commercial software projects

Benefits of GTK+

- Convenient but powerful programming interface
- Widespread portability and availability
- Modern appearance, customizable with themes
- Unrestrictive Licensing (LGPL)
- Availability of Language Bindings

Architecture of X

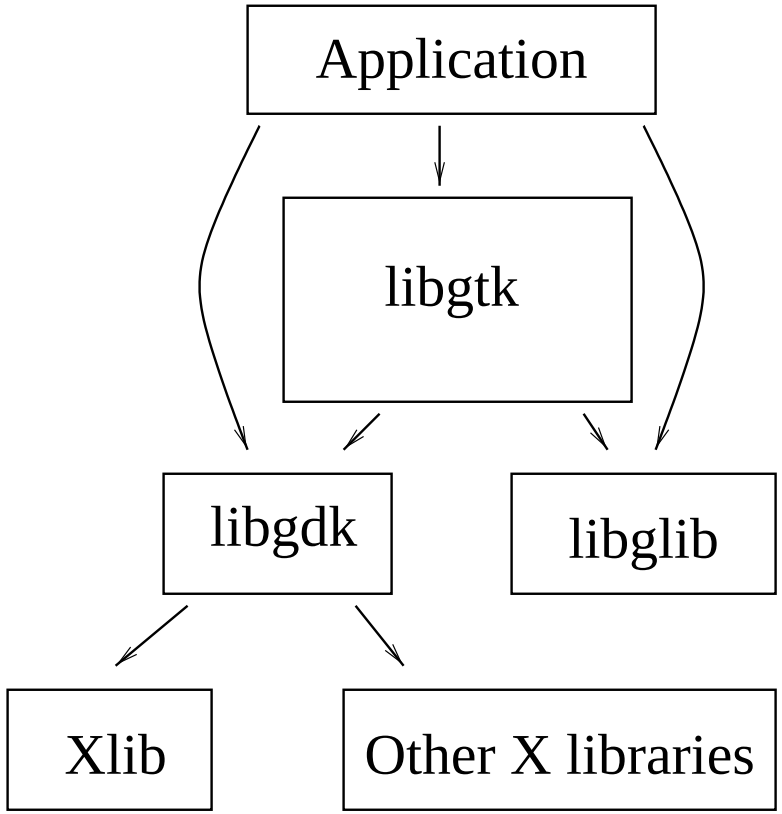


What is a widget?

- User interface component.
- May:
  - Display information.
  - Take input from user.
  - Arrange other widgets.

Button	Entry	FileSelection	Label
List	Menu	MenuItem	Notebook
Scrollbar	SpinButton	Table	Window

The GTK+ Libraries



The GTK+ Libraries

libgtk The widget system and widgets

- GTK+ object model
- Core code for managing, arranging widgets
- 80 types of widgets

libgdk Portability layer for drawing

- Provides basic drawing operations
- A Wrapper around Xlib
- Can be ported to other windowing systems (Win32, BeOS)

libglib Convenient C routines.

- Portable replacements for non-portable C library functions.
- High-level data types.
- Main loop abstraction.

Hello World

```
#include <gtk/gtk.h>

int main (int argc, char **argv)
{
  GtkWidget *window, *button;

  gtk_init (&argc, &argv);

  window = gtk_window_new (GTK_WINDOW_TOPLEVEL);

  button = gtk_button_new_with_label ("Hello World");
  gtk_container_add (GTK_CONTAINER (window), button);

  gtk_signal_connect (GTK_OBJECT (button), "clicked",
                      GTK_SIGNAL_FUNC (clicked), NULL);

  gtk_widget_show_all (window);

  gtk_main();

  return 0;
}
```

Hello World (cont)

Callbacks:

```
void clicked (GtkWidget *widget, gpointer data)
{
  gtk_main_quit();
}
```

Object Orientation

```
GtkWidget *window, *button;

gtk_container_add (GTK_CONTAINER (window), button);
```

- “Objects” represented as structures.
- “Methods” take pointer to object structure as first parameter
- Polymorphism - can call methods for parent classes as well as for object’s own class.

Containers

```
gtk_container_add (GTK_CONTAINER (window), button);
```

- *container* widgets contain other widgets.
- Can have one child (Window widget) or many children (Table widget).
- Even a button is a container that contains the label (or pixmap).
- All layout intelligence lives in container widgets - a container knows how to arrange its children.

Event Driven Programming

```
gtk_main();
```

- All actions done within “main loop”
- Receive events from user, dispatch to program
- Callbacks by *signals*

Signals

```
gtk_signal_connect (GTK_OBJECT (button), "clicked",
                    GTK_SIGNAL_FUNC (clicked), NULL);

void clicked (GtkWidget *widget, gpointer data)
{
  gtk_main_quit();
}
```

- For *notification* and *customization*
- Callback types identified by strings.
- Different prototypes callbacks possible.
- Pass in data to the callback as last argument.

Visibility

```
gtk_widget_show_all (window);
```

- Each widget can be visible or not visible.
- Widgets start off not visible.
- `gtk_widget_show()` shows one widget.
- `gtk_widget_show_all()` shows entire hierarchy.

Compiling a GTK+ program

- Use `gtk-config` to get options

```
$ gtk-config --cflags

-I/usr/X11R6/include -I/opt/gnome/lib/glib/include
   -I/opt/gnome/include

$ gtk-config --libs

-L/opt/gnome/lib -L/usr/X11R6/lib -lgtk -lgdk
   -rdynamic -lgmodule -lglib -ldl -lXi -lXext -lX11 -lm

$ cc -o helloworld ‘gtk-config --cflags‘ helloworld.c \
  ‘gtk-config --libs‘
```

Part II: Principles of GTK+

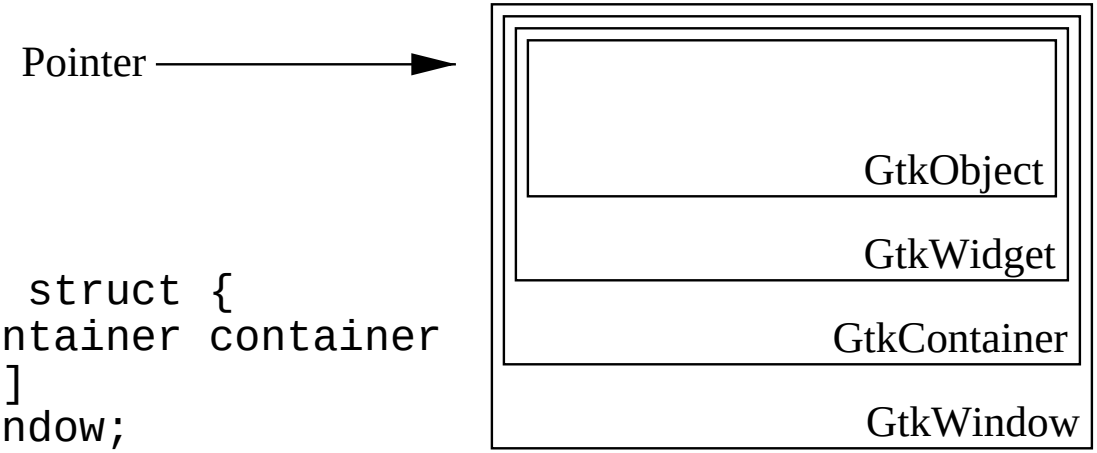
Object System
Geometry Management
Signals and Events
Reference Counting

Object System

- Built in straight C
- Supports conventional object-oriented features
  - Encapsulation
  - Inheritance
  - Polymorphism
- Tuned to needs of GUI programming
  - Introspection
  - Signal system for callbacks
  - Argument system for setting properties via a GUI builder
  - Types can be registered dynamically at run time

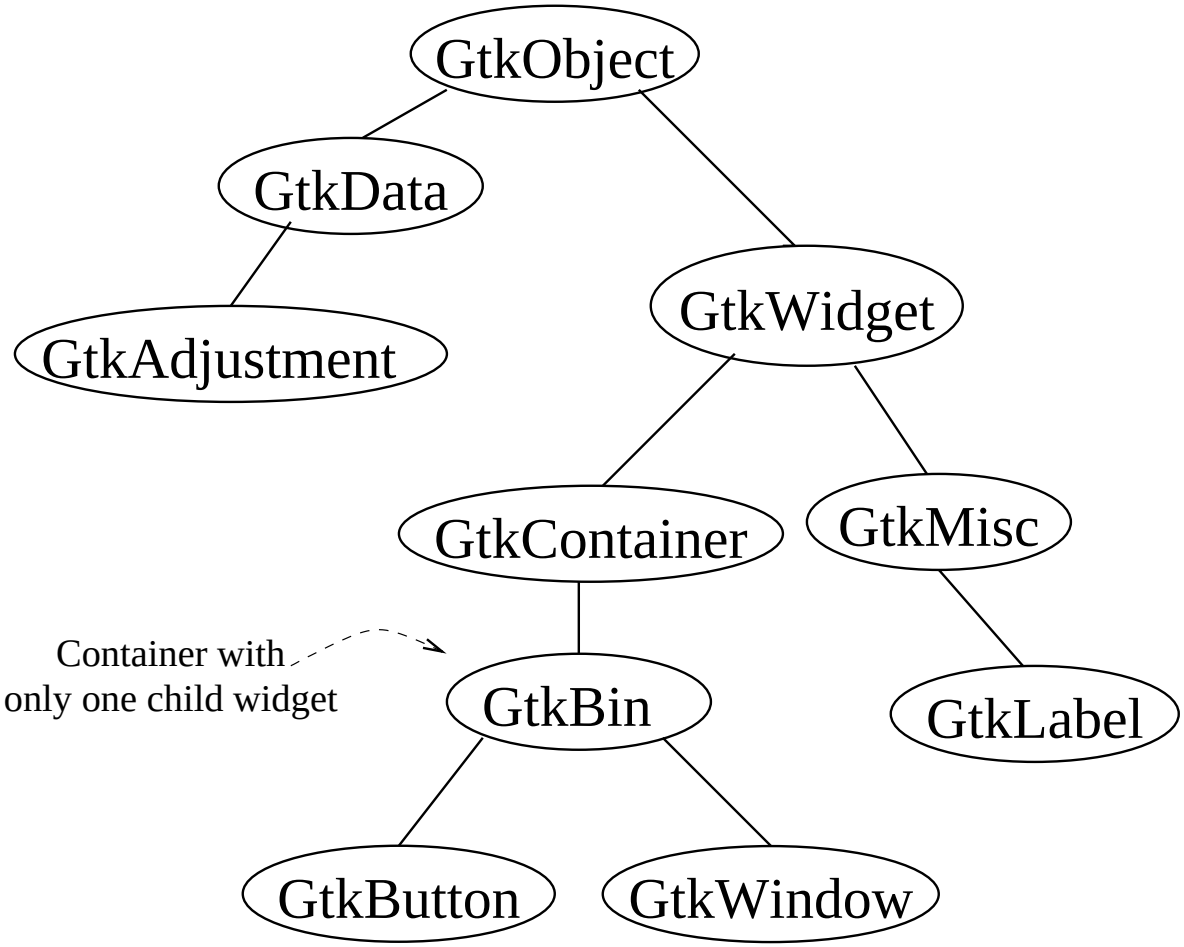
Inheritance

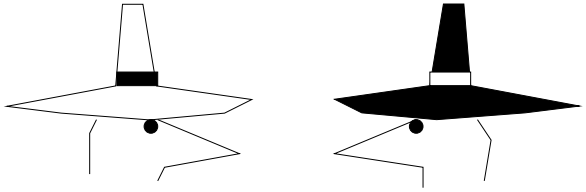
- An object can also be used as any of its parent classes.
- Inheritance done by nesting classes.




```
typedef struct {
  GtkContainer container
   [...]
} GtkWindow;
```

Widget inheritance tree





Hierarchy vs. Hierarchy

- Class Hierarchy
  - parent: base class
  - child: class inheriting from parent
- Widget Hierarchy
  - parent: container
  - child: containee

Casting Macros

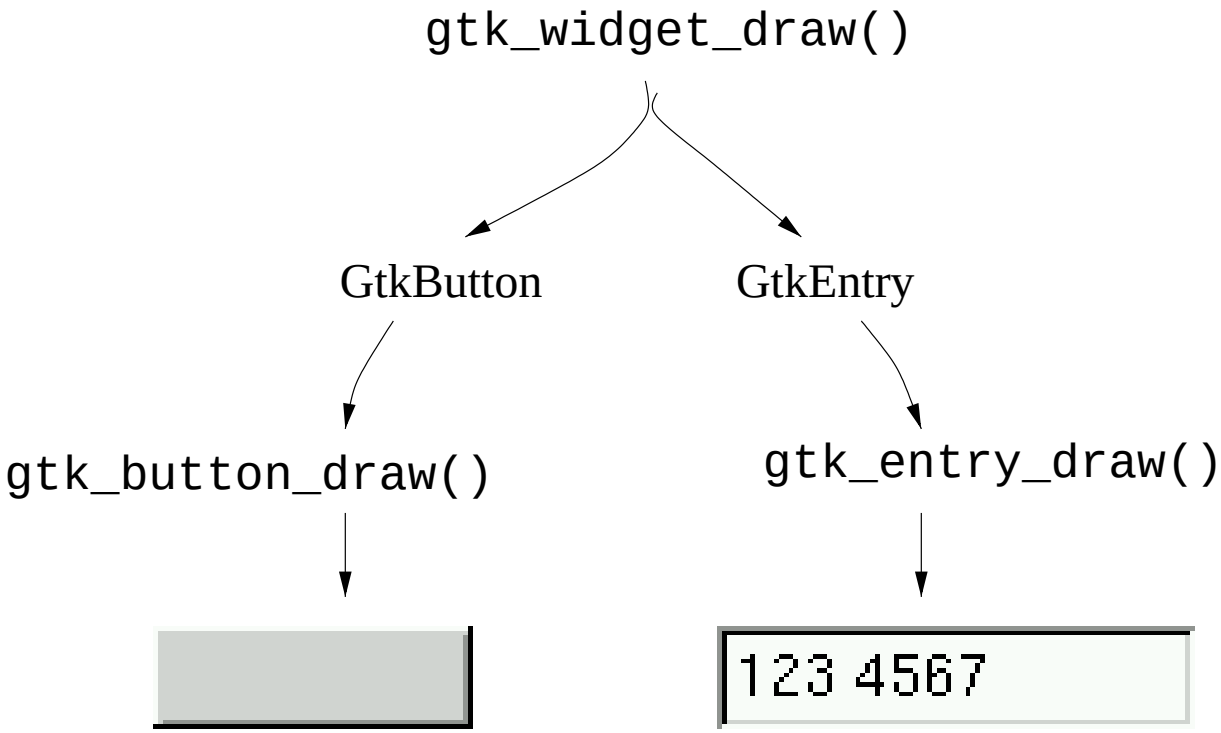
```
GtkWidget *window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
gtk_window_set_title (GTK_WINDOW (window), "My Application");
```

- Typically, pointers to widgets stored as type `GtkWidget *`.
- Each class has standard macros for converting to that class
- `GTK_WINDOW`(window) casts to a `GtkWindow *` but *with checking*.
- `GTK_WINDOW`(button) will producing warning

  ```
  Gtk-WARNING **: invalid cast from ‘GtkButton’ to ‘GtkWindow’
  ```

- Checks are efficient but can be disabled at compile time

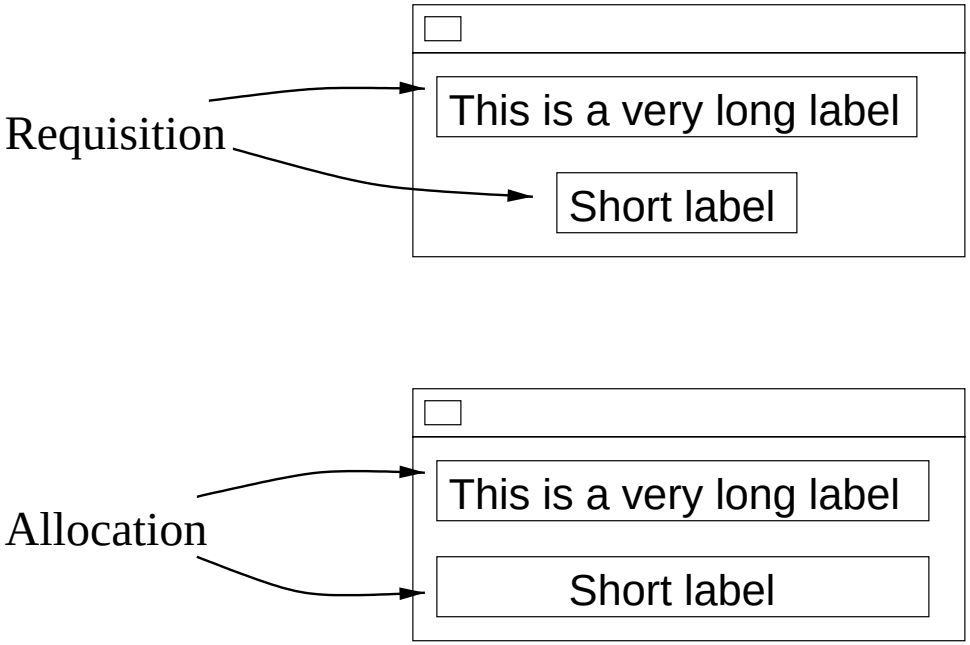
Polymorphism



Geometry Negotiation

- *requisition* is amount of space widget needs based on its contents
- Containers request space based on size of children. (VBox's requested height is sum of height of all children, plus padding.)
- Each child then assigned an *allocation*.
- allocation never smaller than requisition, but may be larger.

Geometry Example



- Allocation will generally be at least as big as requisition, may be bigger.

Packing Boxes

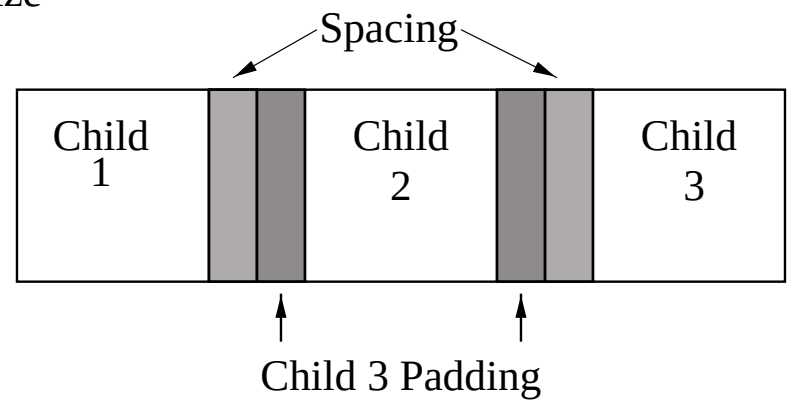
```
void
gtk_hbox_new (gboolean homogeneous, guint spacing);

void
gtk_box_pack_start (GtkBox *box, GtkWidget *child,
                    gboolean expand, gboolean fill, guint padding);
```

- Arrange child horizontally (HBox) or vertically (VBox).
- Per-box `homogeneous` and `spacing` options.
- Each child has `expand`, `fill`, and `paddding` options.

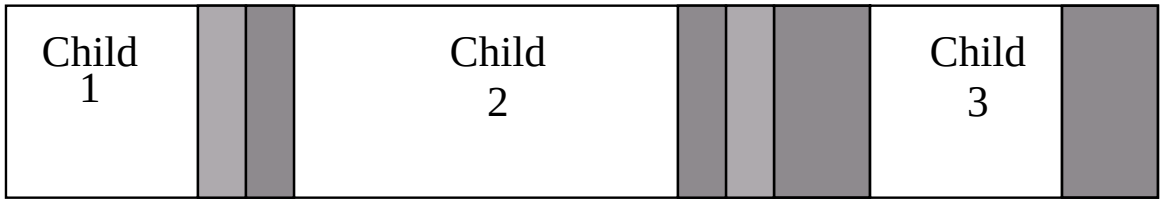
Packing Boxes (cont)

Box	Expand	Fill
1	NO	NO
2	YES	YES
3	YES	NO

Normal Size



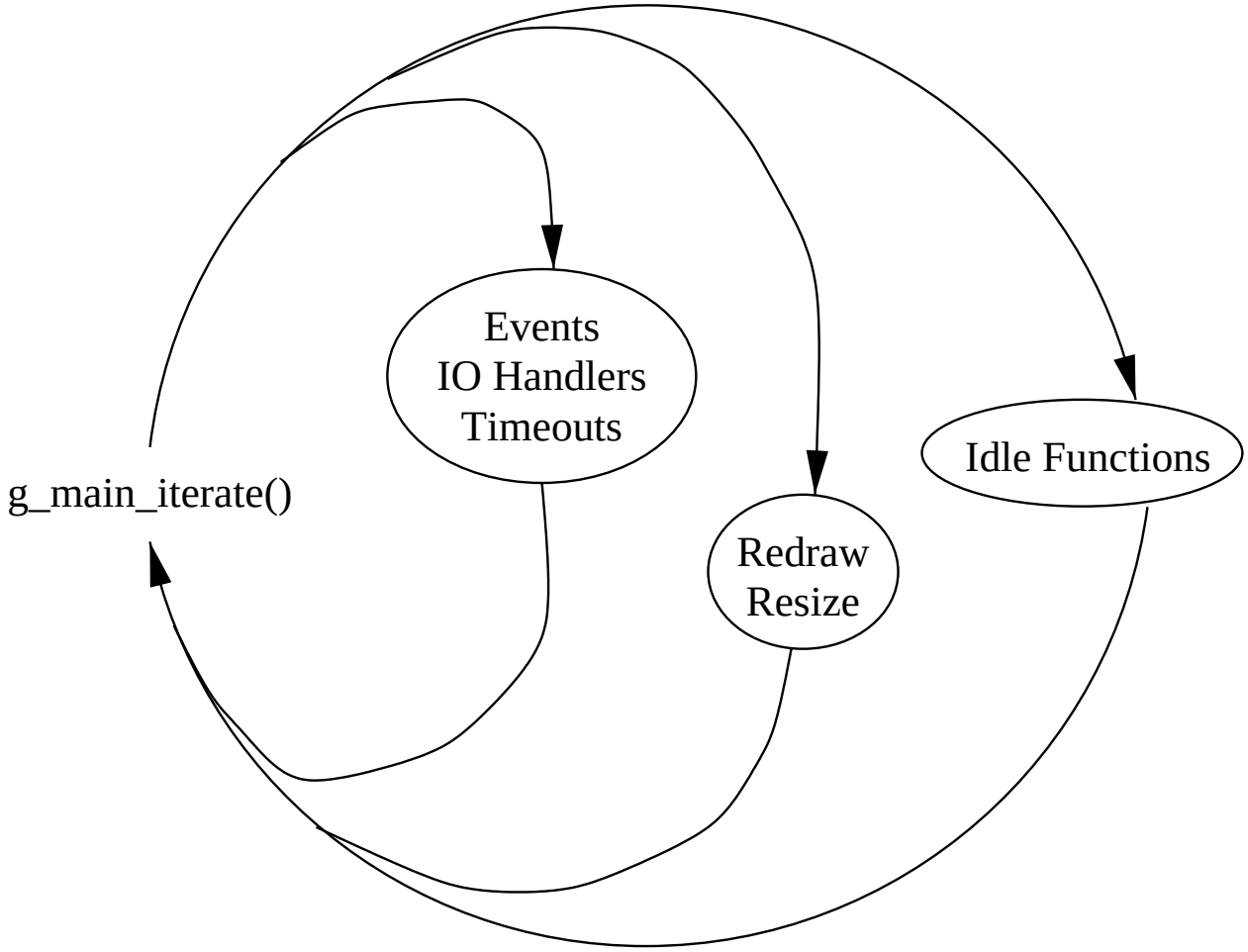
Expanded by User



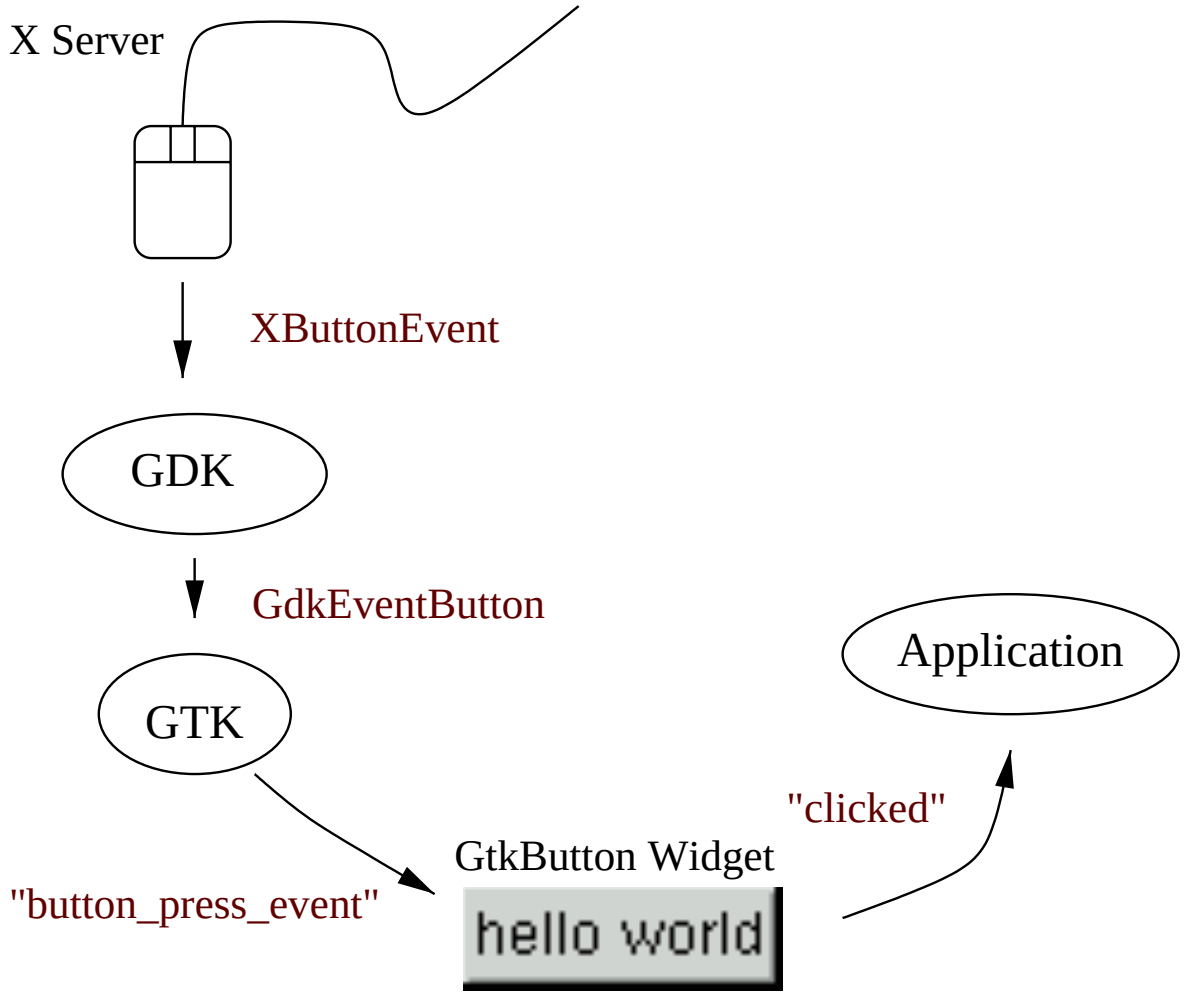
Main Loop

- Events retrieved from event *sources*
- Standard event sources:
  Glib: Timeouts, IO Handlers, Idle Handlers
  GDK: X events
- Additional source types can be created.
- Sources prioritized
  1. Incoming X Events
  2. GTK+'s redraw, resize queues
  3. Application's idle handlers.
- Lower priority sources not serviced until high priority sources finished.

Main Loop (cont)



Events and Signals



Events and Signals

- Lowlevel events sent from X server
- Corresponding signals sent to appropriate widget
- Widgets generate highlevel events
- Event signals have a distinct signature
- Return value determines propagation. `TRUE` => handled.

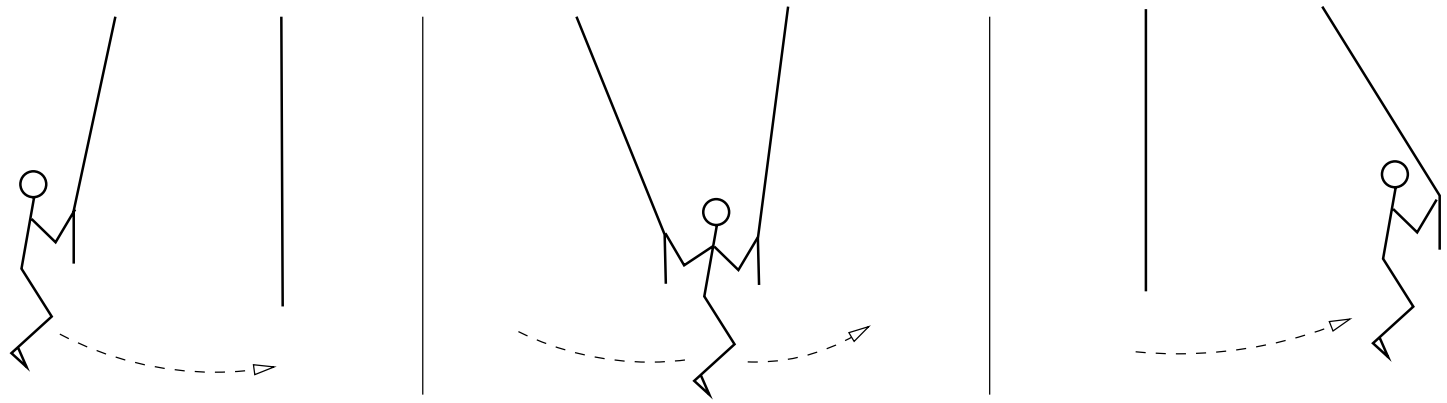
Reference Counting

- Need to know when objects are unused (garbage collection)
- Explicit ownership works badly for GUIs.
- Keep a reference count
  - Create an item, refcount is 1.
  - Begin using item, increase refcount by 1 - `ref()`
  - Finish using item, decrease refcount by 1 - `unref()`
  - When reference count drops to zero, free object

Reference Counting example

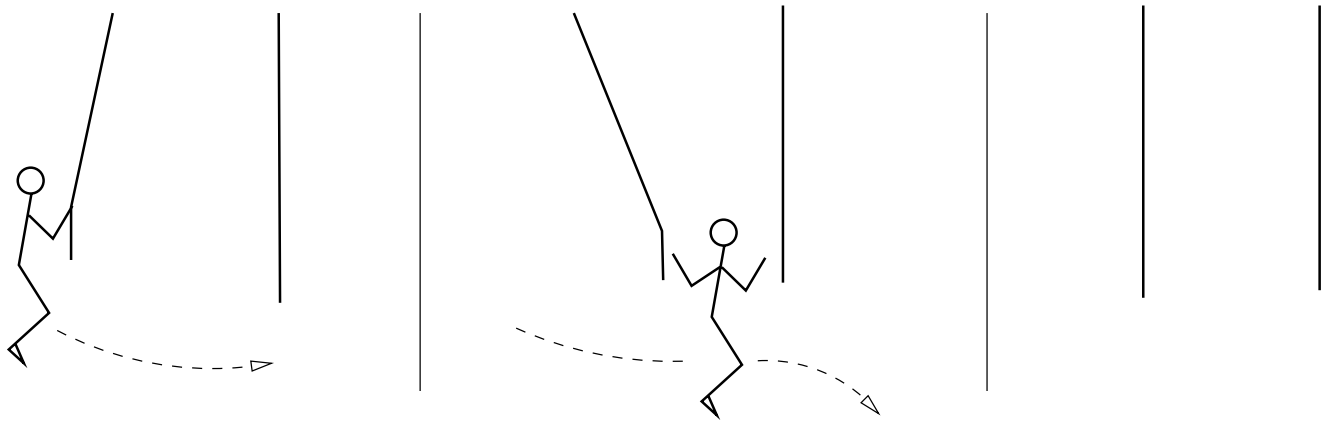
- Parent keeps reference count on children
- Removing child from parent causes it to be freed
- So, to move child from one to other, need to do:

```
gtk_object_ref (GTK_OBJECT (child));
gtk_container_remove (GTK_CONTAINER (old_parent), child);
gtk_container_add (GTK_CONTAINER (new_parent), child);
gtk_object_unref (GTK_OBJECT (child));
```



Reference Counting example (cont)

- If you forget to refcount...



Floating and Sinking

- Reference counting for GtkObject not quite so simple
- Don't want to have to write:

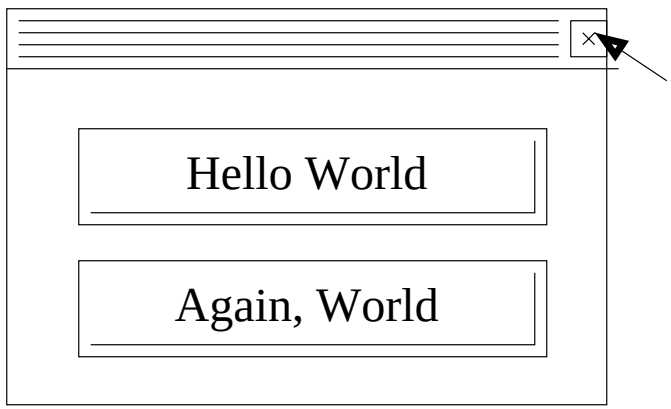
```
button = gtk_button_new_with_label ("Hello World");
gtk_container_add (GTK_CONTAINER (window), button);
gtk_widget_unref (button);
```

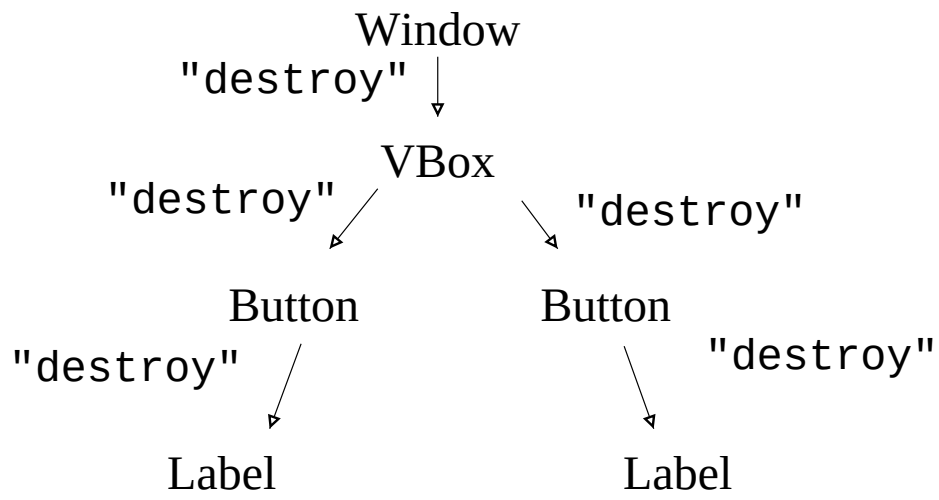
- So, container *assumes* reference count of child.
- GtkObject initially created and marked floating. (Reference count can be assumed)
- Parent calls `gtk_widget_sink()` to remove flag.

The Life Cycle of a Widget

Create	`gtk_label_new()`	
Parenting	`gtk_container_add()`	
Show	`gtk_widget_show()`	
Size Request		`"size_request"`
Size Allocate		`"size_allocate"`
Realize		`"realize"`
Map		`"map"`
Expose		`"expose"`
Destroy	`gtk_widget_destroy()`	`"destroy"`
Unmap		`"unmap"`
Unrealize		`"unrealize"`
Unparent		
Finalize		

Widget Destruction





- Reference counting vulnerable to *cycles*.
- Explicit destruction helps.
- Triggered by user closing window, or app calling `gtk_widget_destroy()`
- Destruction propagates recursively to children.

Life cycle of GdkWindow

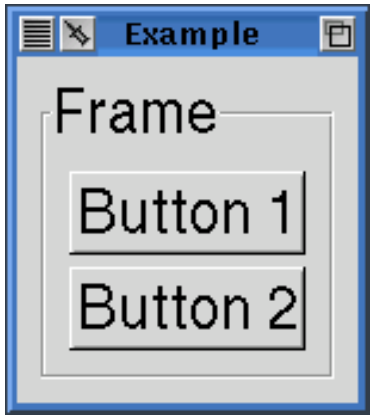
- GdkWindow is server side object. Used to receive events and clip drawing.
- Includes toplevel windows, but also children of those windows.
- **Realization** GDK/X Window for widget is created.

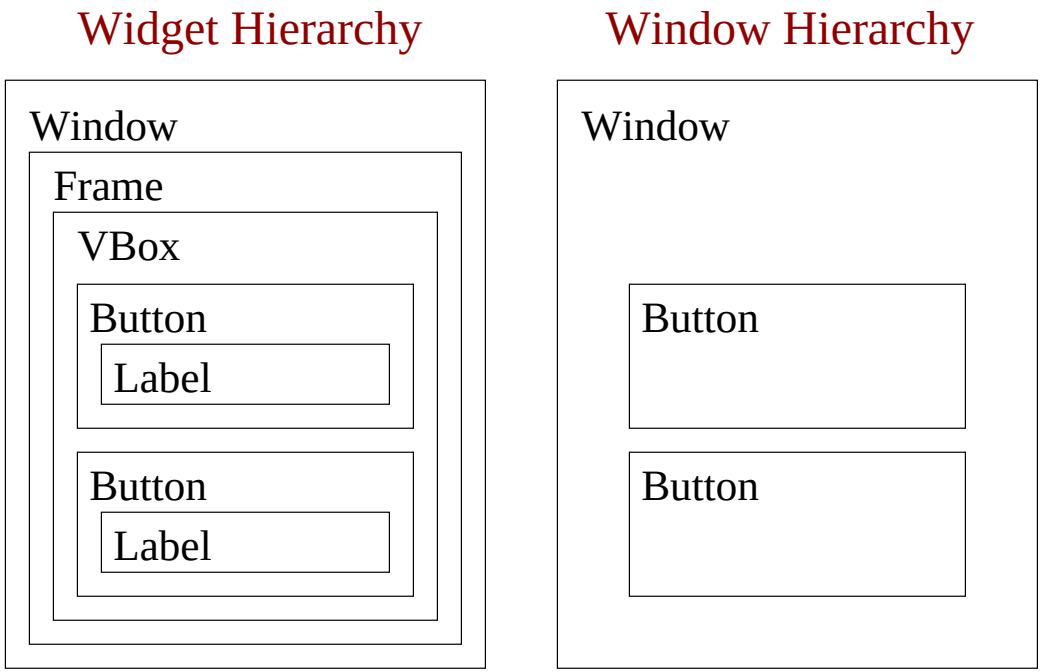
  Map GdkWindow is made visible.

  Expose X asks toolkit to refresh the widget's display.

  Unrealize widget removed from the screen.
- Generally these steps occur automatically. Only time you have to worry about realization is when you want to make GDK calls and need the GdkWindow for a widget.

Widgets and GdkWindows



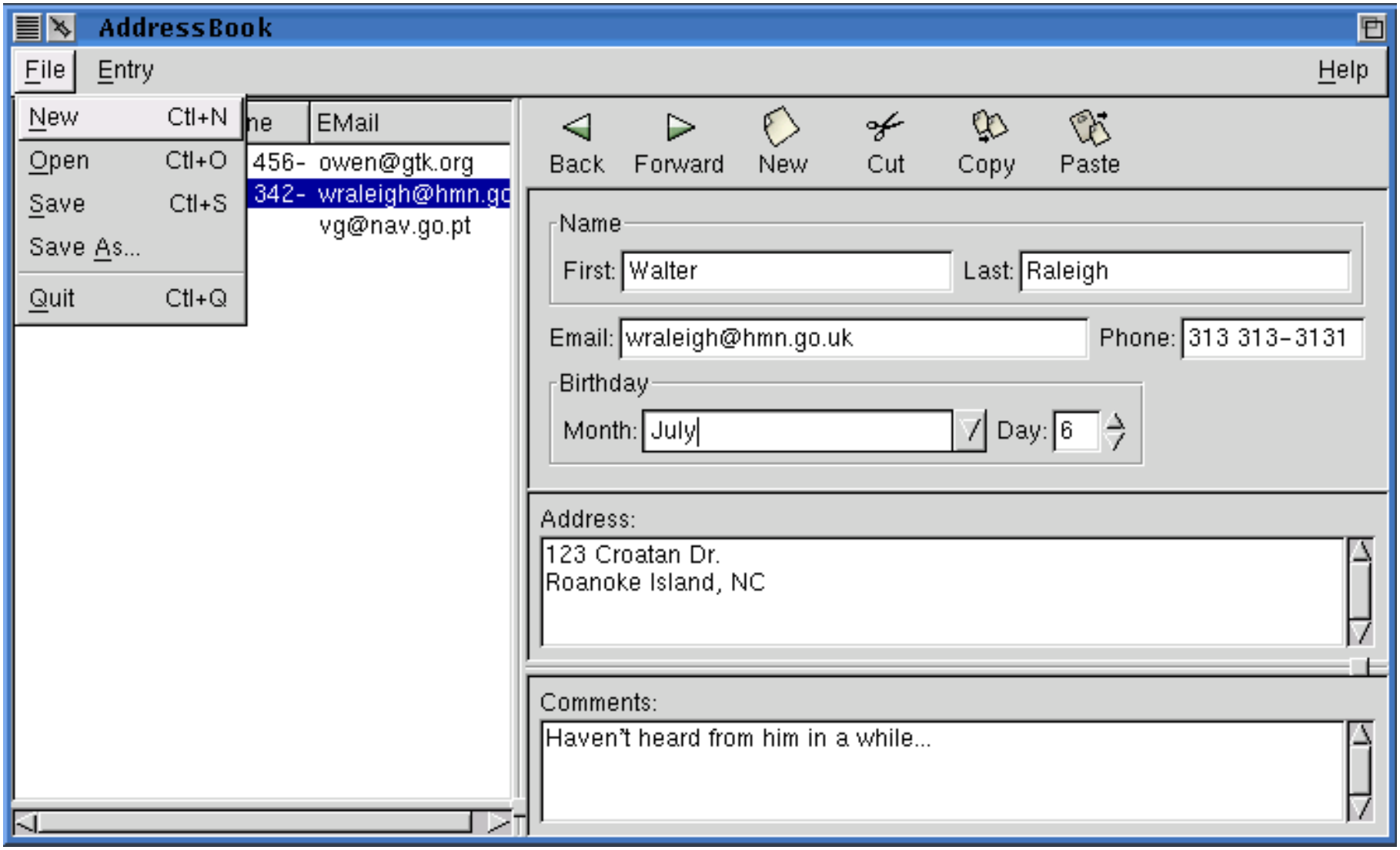


- GdkWindows used for delivering events
- Many widgets do not have corresponding GdkWindow
  Frame, VBox, Label: `NO_WINDOW` widgets.
  These widgets draw on parent widget's window.
- Other widgets have windows: Window, Button, etc.

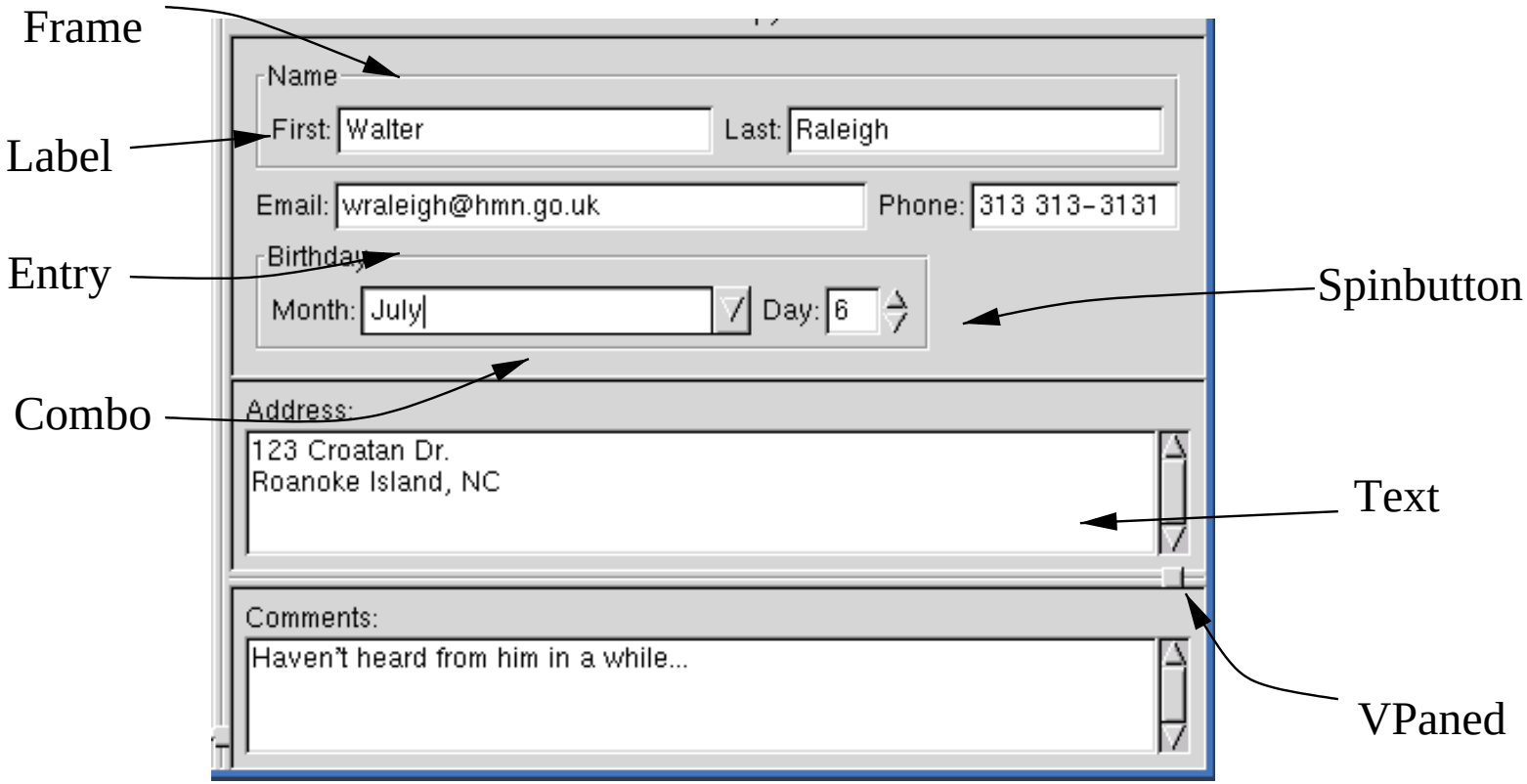
Part III: Building an application

The Application
Widget tour
Geometry management in detail
Using GLib

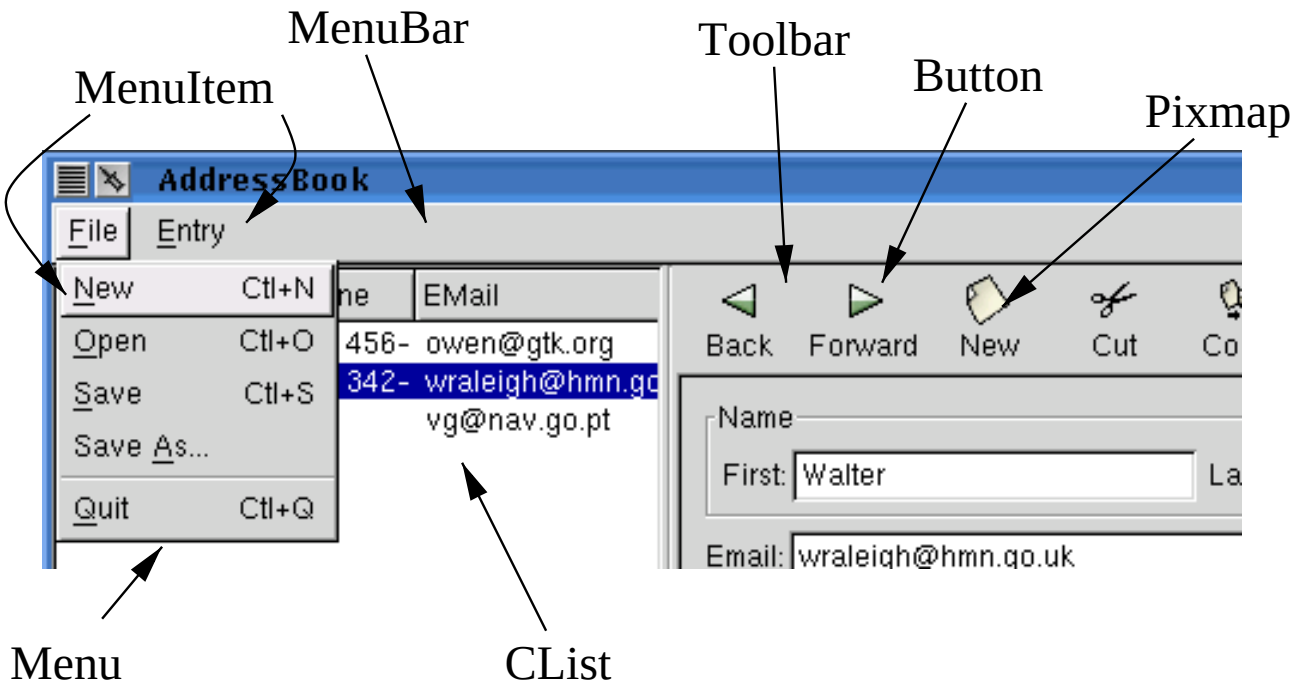
The example


AddressBook
File
Entry
Help
New Ctl+N
Open Ctl+O
Save Ctl+S
Save As...
Quit Ctl+Q
EMail
456- owen@gtk.org
342- wraleigh@hmn.go
vg@nav.go.pt
Back
Forward
New
Cut
Copy
Paste
Name
First: Walter
Last: Raleigh
Email: wraleigh@hmn.go.uk
Phone: 313 313-3131
Birthday
Month: July
Day: 6
Address:
123 Croatan Dr.
Roanoke Island, NC
Comments:
Haven't heard from him in a while...

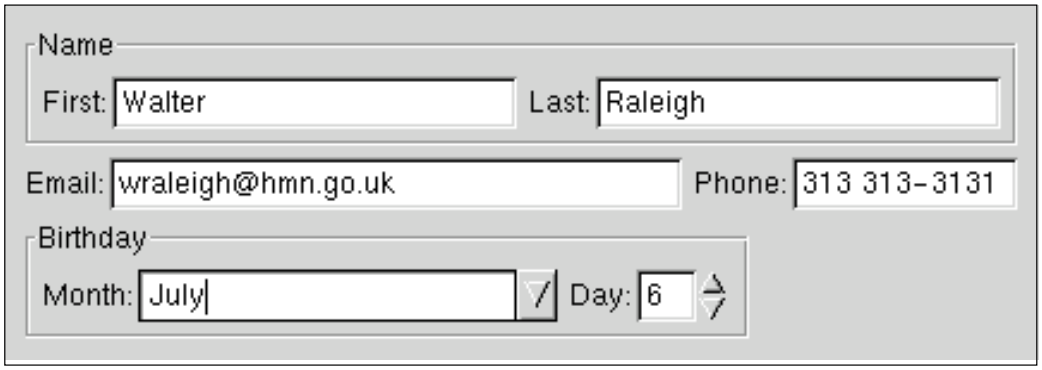
Widgets in an addressbook

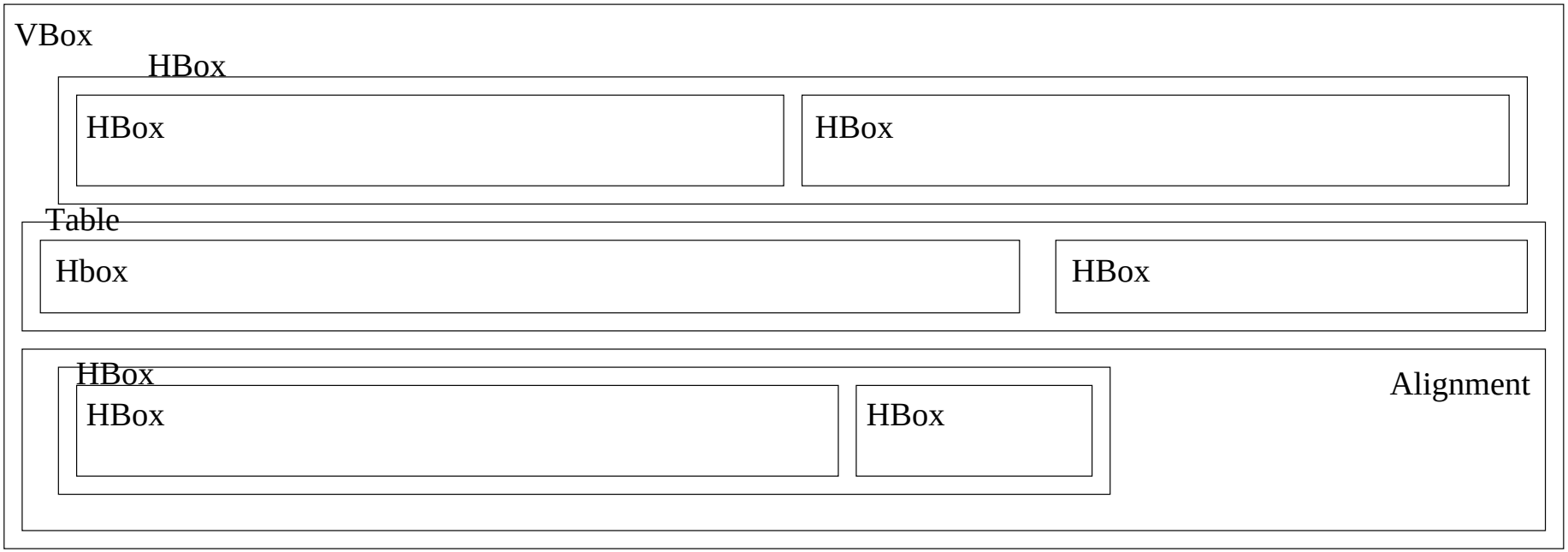


More widgets in an addressbook



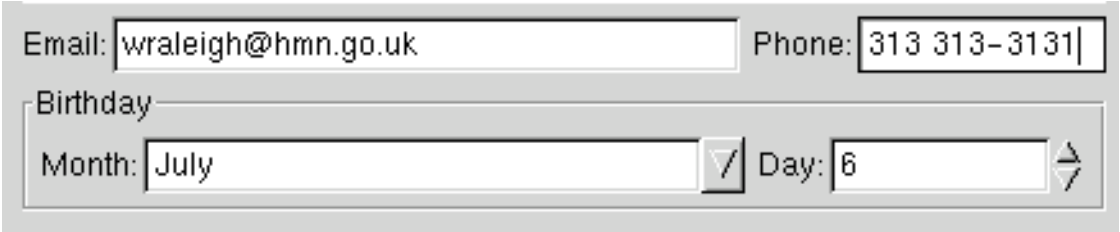
Complex Geometry Management



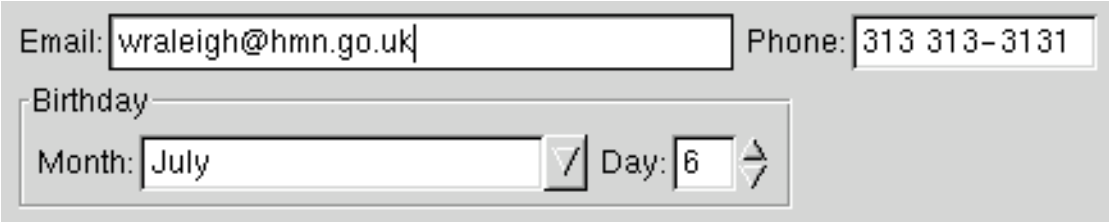


Using a GtkAlignment

- Want widget to take up only part of space

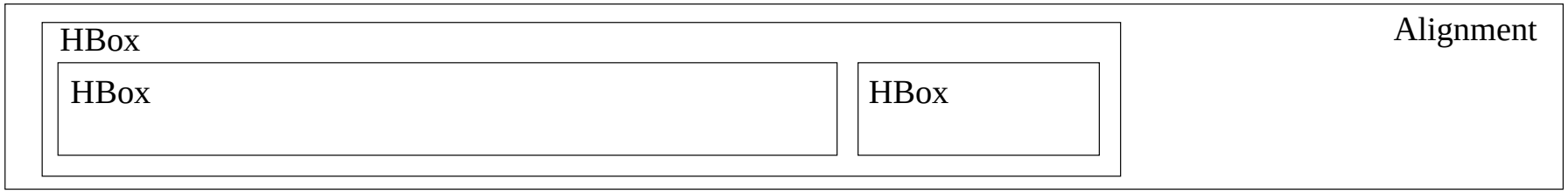


- Solution: use a GtkAlignment



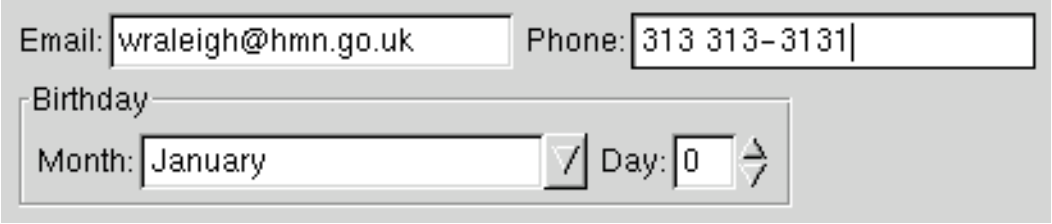

```
GtkWidget *gtk_alignment_new (gfloat xalign, gfloat yalign,
                              gfloat xscale, gfloat yscale);
```

`xscale: 0.0` (no expansion) `xalign: 0.0` (left)

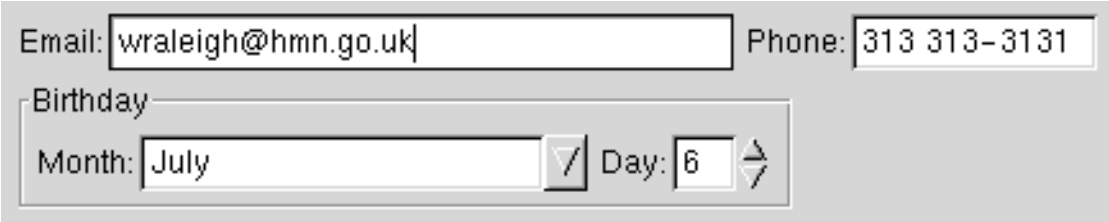


Enforcing a 2:1 ratio between elements

- Want email twice as wide as phone number

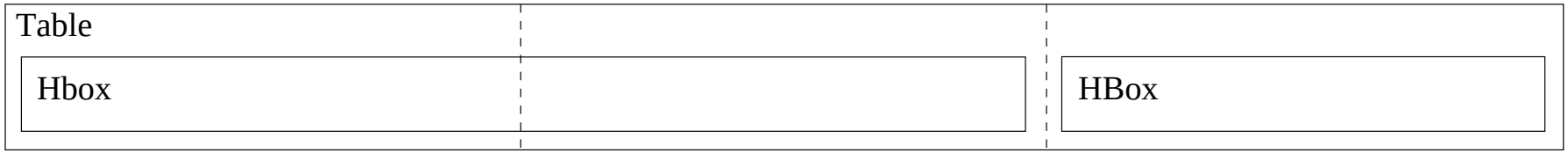


- Solution: use a homogeneous GtkTable




```
GtkWidget *gtk_table_new (guint rows, guint columns,
                          gboolean homogeneous);
```

`rows: 1  columns: 3  homogeneous: TRUE`



Using signals for *behavior modification*

- Many methods routed through signals; allow modifying behavior without inheriting and overriding the methods.
- Example: create an entry that only accepts `[0-9- ]`.
- GtkEditable (parent class of GtkText and GtkEntry) has `"insert_text"` signal.
- Connect to this signal, and in signal handler
  1. Modify text as desired
  2. Insert this text (being careful not to recurse)
  3. Stop the default handler from running

```
gtk_signal_connect (GTK_OBJECT (phone_entry), "insert_text",
                    GTK_SIGNAL_FUNC (insert_text_handler),
                    NULL);
```

Using signals for *behavior modification* (cont)

```
void
insert_text_handler (GtkEditable *editable, const gchar *text,
                     gint length, gint *position, gpointer data)
{
  int i, j;
  gchar *result = g_new (gchar, length);

  [ copy text into result, stripping out unwanted characters ]

  /* Block ourselves, and insert modified text */
  gtk_signal_handler_block_by_func (GTK_OBJECT (editable),
                                    GTK_SIGNAL_FUNC (insert_text_handler)
                                    data);
  gtk_editable_insert_text (editable, result, length, position);
  gtk_signal_handler_unblock_by_func (GTK_OBJECT (editable),
                                      GTK_SIGNAL_FUNC (insert_text_handle
                                      data);

  /* Keep default handler from being run */
  gtk_signal_emit_stop_by_name (GTK_OBJECT (editable), "insert_text");

  g_free (result);
}
```

Item Factory

- Menus normal containers
- Creating menus and menu items by hand repetition
- Solution: automatic creation from an array

```
static GtkItemFactoryEntry menu_items[] =
{
  /* menu-path      accelerator    callback + action  item-type */
  { "/_File",       NULL,          NULL, 0,           "<Branch>" },
  { "/File/_Open",  "<control>O",  do_open, 0,        NULL },
  { "/File/_Save",  "<control>S",  do_save, 0,        NULL },
  { "/File/sep1",   NULL,          NULL, 0,           "<Separator>" },
  { "/File/_Quit",  "<control>Q",  do_quit, 0,        NULL
};
static int nmenu_items = sizeof (menu_items) / sizeof (menu_items[0]);

gtk_item_factory_create_items (item_factory,
                               nmenu_items, menu_items, NULL);
```

GLib

- Main loop
- Portability functions
  `g_strcasecmp()`, `g_snprintf()`
- Convenience functions
  `g_strsplit()`, `g_get_home_dir()`
- Data types

  GList linked lists

  GHashTable hash tables

  GTree balanced trees

  GString string type
- GScanner - run-time configurable tokenizer

GString

- automatically handles memory allocation, reallocation

```
GString *string = g_string_new (NULL);

g_string_append (string, "Goodbye");
g_string_append_c (string, ',');
g_string_append (string, " Old Paint");

printf(string->str);
g_string_free (string, TRUE);
```

GList

- Doubly-linked list
- Also GSList - singly linked list
- NULL represents empty list

```
GList *word_list, *result = NULL;
for (i=0; i<n_words; i++)
  word_list = g_list_prepend (word_list, g_strdup (word));
[...]
result = g_list_find_custom (word_list, "blue", g_str_equal);
if (result)
{
  word_list = g_list_remove_link (word_list, result);
  g_free (result->data);
  g_list_free_1 (result);
}
```

Part IV: Beyond GTK+

Extending GTK+

Language Bindings

GNOME

Future Directions

Creating custom user interface elements

- If needs are light, can use GtkDrawingArea.
  - Receive low-level events: `"button_press_event"`, `"key_press_event"`.
  - Get notified to size changes by `"size_allocate"`.
  - In response to `"expose"`, draw on widget's window with GDK drawing calls.
- May be better to use GnomeCanvas widget. (see later)
- For heavier applications, can derive own widget types. (code reuse).

Creating widgets

- Define object and *class* structures for new type.
- class structure
  - Pointed to from object structure.
  - Contains function pointers to the *virtual functions* of the widget.
  - Nested like object structures.
  - Usually widgets will override virtual functions from GtkWidget like `size_request()`, `size_alllocate()`, and `expose()`.
- Register type with `init()` and `class_init()` functions.
- `init()` function initializes newly created object structures.
- `class_init()` function initializes class structure, creates signal and argument types for class.

Language Bindings

- GTK+'s object orientation maps well onto languages with native OO features.
- Languages with OO features often make using GTK+ more concise.
- Ability to query types at runtimes simplifies creating language bindings.
- Many language bindings exist:
  C++, Perl, Python, Ada, Dylan, Eiffel, Guile, Haskell, JavaScript, Objective C, Camel, Label, Pascal, Pike, TOM. . .

Perl/GTK

```
use Gtk;
Gtk::init();

$window = new Gtk::Window;
$button = new Gtk::Button "Hello World";
$window->add($button);

$button->signal_connect("clicked",
                sub { Gtk::main_quit() });

$window->show_all;
Gtk::main();
```

PyGtk

```
from _gtk import *
from GTK import *

def clicked(*args):
    mainquit()

window = GtkWindow()
button = GtkButton('Hello World')
window.add(button)

button.connect('clicked', clicked)

window.show_all
mainloop()
```

GLADE

- GUI builder
- Graphically build widget tree
- Write out code (C, C++, Ada), or XML
- Rebuild widget tree in application from XML (libglade)

GNOME

- Layer between GTK+ and application.
- Provides high-level functionality, more widgets.
- Enforces consistency.
- See `http://developer.gnome.org` for more info.

GNOME Widgets

- GnomeMessageBox - easy to display error messages
- GnomeIconList - file-manager-style icon display
- GnomeColorPicker, GnomeFontPicker – standard interfaces for a color or font selection button.
- GnomeDruid – Wizard(tm) style setup dialogs.
- Many more...

GnomeCanvas

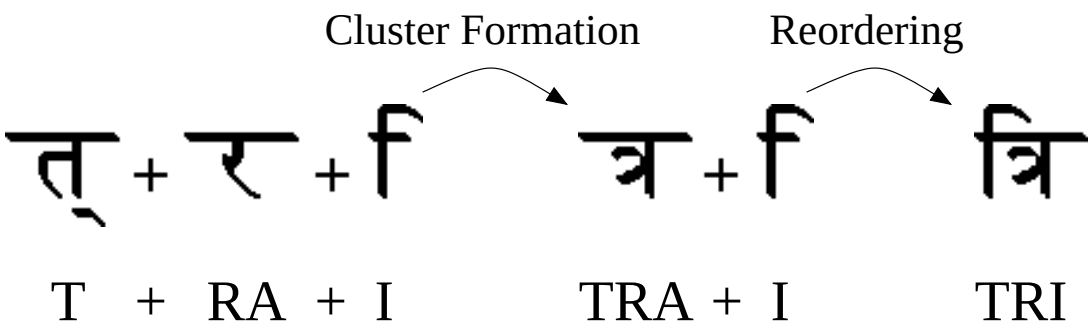
- Structured graphics - hierarchy of graphics objects much like widget hierarchy.
- Easy to make complex, user-manipulatible displays.
- Flicker-free.
- Standard items:
  rectangle, circle, text, image...
- Can implement custom items.
- Can render *anti-aliased*

GNOME High-level Functionality

- Configuration data storage.
- Session-management.
- Application framework.
- Help System.
- Mime-type support.

Unicode and enhanced internationalization support

- GTK+ currently supports Asian double-byte languages and input methods.
- Move to Unicode - 1 encoding for all languages.
- Support right-to-left languages like Hebrew.
- Support *complex-text* languages like Hindi.
- Use “pango” - a toolkit independent framework for rendering with Unicode.



Enhanced Object Model

- Mostly single inheritance works well for GTK+, but sometimes clumsy.
- Java-style multiple interfaces will be added 1.4 - GtkRadioButton and GtkRadioMenuItem would both export a GtkRadio interface.
- Improvements to argument system to better support GUI builders, themes.
  - Notification on changes.
  - Ability to set arguments from an RC file.

Win32 Port

- Only GDK layer needs to be ported.
- Done by Tor Lillqvist.
- Has been functioning for 6 months to the point of running the GIMP.
- Will be integrated in with main line of GTK+ for 1.4.

Acknowledgements

- Original Authors of GTK+

  Peter Mattis Spencer Kimball Josh Macdonald

- The GTK+ Team

Shawn Amundson	Jerome Bolliet	Damon Chaplin
Tony Gale	Jeff Garzik	Lars Hamann
Raja Harinath	Carsten Haitzler	Akira Higuchi
Tim Janik	Stefan Jeske	Elliot Lee
Raph Levien	Ian Main	Takashi Matsuda
Frederico Meña	Paolo Molaro	Jay Painter
Manish Singh		