PHILIPS

sense and simplicity

jointSPACE,
write your first remote application

Jean-Marc Harvengt & Tim Vandecasteele

December 18, 2010

PHILIPS

Developing a remote application in 10 steps

Prepare the development environment

1. Install compiler tool chain for your remote device
2. Download and compile the jointSPACE SDK

Write your first Remote Application

3. Device Discovery
4. DirectFB initialization
5. Create a graphical window on TV
6. Draw using DirectFB primitives
7. Draw without using DirectFB primitives
8. Handle keys and events
9. Handle animations
10. Inject keys

Prepare the development environment

Step1: install the compiler tool chain

1. Select a platform for your Remote Application

2. Download the required compiler tool chain (gnu gcc-g++, make)
   - MacOS => Install Xcode tool chain
   - Windows => install Cygwin tool chain
   - Linux => Install dev-tools from your Linux distribution
   - Android => Install Android SDK+NDK on Mac/Win/Linux
   - iPhone => Install Xcode + iPhone SDK

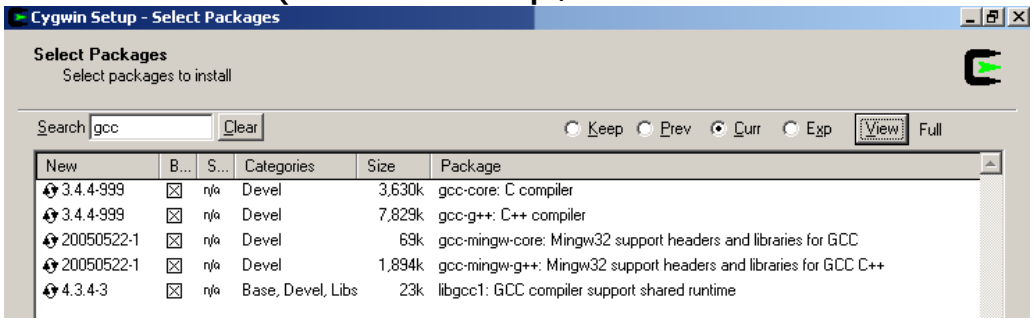
3. Test the tool chain

```
$(CC) $(LD)
CC: no input file
LD: no input file
```

Step1: install Cygwin tool chain on Windows

- Go to http://www.cygwin.com/
- Download and click on  setup.exe file
- Install from Internet (@home!) or from Local Directory (@training!)
  - Choose Next (default install directory is c:\cygwin)
  - Browse to local Package Directory (c:\cygwinlocal)
  - Search consecutively for “gcc”, “make”, “7zip”, press View
    - For each query, select the results (click on Skip, it becomes a checked checkbox)
  - Next, Next…

    until end of installation



- Start Cygwin shell (click on c:\cygwin\cygwin.bat) and type:

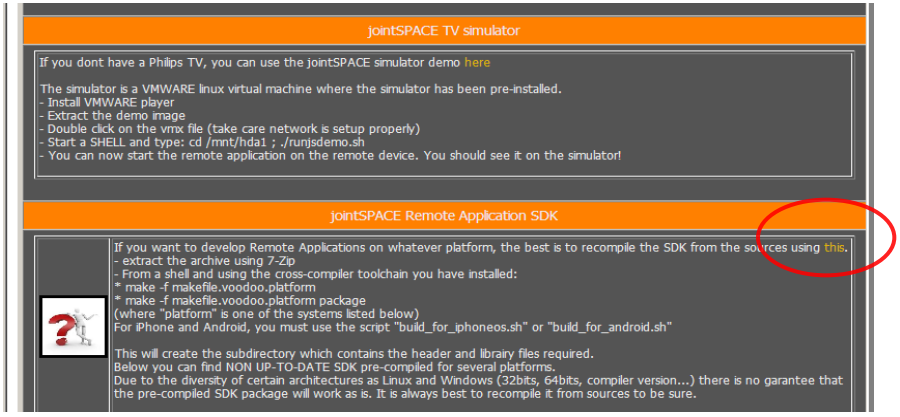
```
$ $(CC) $(LD)
CC: no input file
LD: no input file
```

PHILIPS

Step2: **download** and compile jointSPACE SDK

- Create a sub-directory “js” in c:\cygwin

- Download the SDK source package from jointSPACE web site
  - Go to http://jointspace.sourceforge.net/
  - Select “Download”
  - Go to “jointSPACE Remote Application SDK” section
  - Click on “this”
  - Save “DirectFB141_source_1.2.1beta2.7z” in “c:\cygwin\js”



Step2: download and **compile** jointSPACE SDK

- From the Cygwin shell, compile the jointSPACE SDK

```
$ cd /js/
$ 7z x DirectFB141_source_1.2.1beta2.7z
$ cd /js/DirectFB141_2k11/
$ make -f makefile.voodoo.cygwin
ERROR already ????
Source/DirectFB/lib/direct/clock.c:36:25: sys/syscall.h: No such file or directory
Source/DirectFB/lib/direct/clock.c: In function `direct_monotonic_gettimeofday':
Comment out line 115 in C:\cygwin\usr\include\sys\features.h
//#define _POSIX_MONOTONIC_CLOCK                          200112L

$ make -f makefile.voodoo.cygwin
$ make -f makefile.voodoo.cygwin package
ERROR again ????

$ chmod a+rwx build-package.sh
$ make -f makefile.voodoo.cygwin package
$ mv DirectFB_Voodoo ..
```

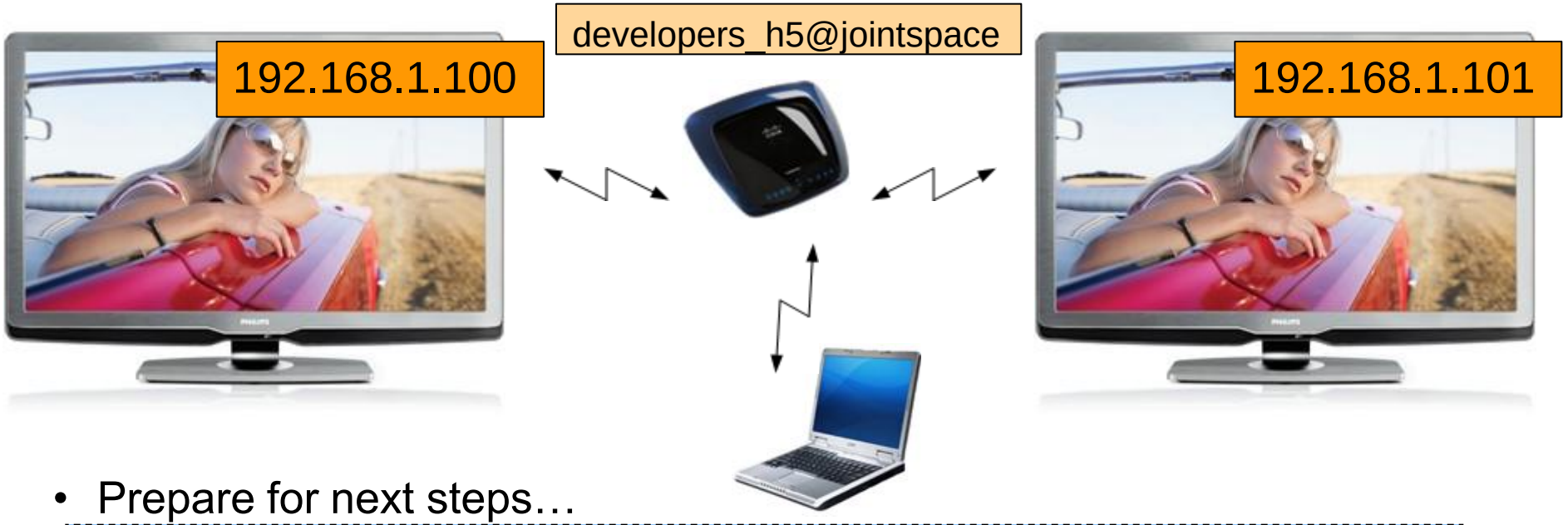
- Download the jointSPACE code samples from jointSPACE web site
  - http://sourceforge.net/projects/jointspace/files/remote_applications_source_samples/sample.zip/download
  - http://sourceforge.net/projects/jointspace/files/remote_applications_source_samples/framework.zip/download

PHILIPS

Step2: compile and **test the sample**

- From the Cygwin shell, compile and test the “sample”

```
$ cd /js
$ unzip sample.zip
$ cd sample
$ DIRECTFB_VOODOO=../DirectFB_Voodoo make
$ DFBARGS=remote=192.168.1.100 ./sample.exe
```



- Prepare for next steps…

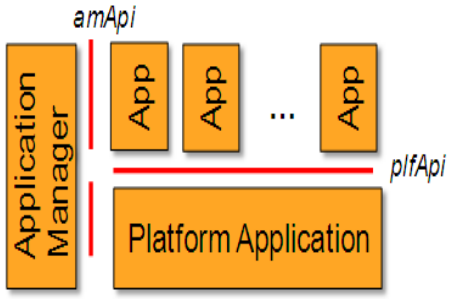
```
$ cd /js
$ unzip framework.zip
$ cd framework
$ DIRECTFB_VOODOO=../DirectFB_Voodoo make
$ DFBARGS=remote=192.168.1.100 ./main.exe
```

PHILIPS

Writing your first Remote Application

Reminder

- jointSPACE uses DirectFB technologies (www.directfb.org)
  - DirectFB
    - Share and provide access to GFX resources for all applications
    - Provides an API to draw graphics (lines, rectangles, bitmaps…)
    - Includes a **Window Manager** to abstract from HW
    - Supports image/font decoding/rendering through specific modules
    - DirectFB does **NOT** provide User Interface Widgets (menu, slider…)
  - Divine
    - Allows events injection to the TV system (keys…)
  - FusionDale
    - Inter Process Communication framework
  - SaWMan
    - Application Manager framework



Step3: Device Discovery

- If a single TV is on the network, nothing special must be done
  - By default, the TV will be detected at initialization

- If multiple TVs are on the network, the IP address of the TV must be specified via an environment variable

```
DFBARGS=remote=<ip_address>
```

  - Environment variable is passed when launching a program

- Discovery can also be done dynamically by creating a discovery loop
  - A tool called "voodooplay" can be used to enumerate the devices found and derive the IP address from the FriendlyName.

```
$ cd /js/DirectFB141_2k11/Source/DirectFB/tools/
$ ./voodooplay

(*) Voodoo/Player:   MS TCP Loopback interface (127.255.255.255)
(*) Voodoo/Player: Received message from ourself (127.0.0.1)
(*) Voodoo/Play: < 235ms> [ PhilipsTV                    ]   192.168.1.2
```

Step4: DirectFB initialization

- A Remote Application always starts calling DirectFBInit()
- Use DirectFBSetOption() to address a specific TV (IP address)

```
#include <directfb.h>

#define DFBCHECK(x...)                                  \
do {                                                    \
  DFBResult err;                                        \
  err = x;                                              \
  if (err != DFB_OK) {                                  \
    printf ("Fail!! err!=DFB_OK");                      \
    DirectFBError (#x, err);                            \
  }                                                     \
} while(0);


/*************************************
* Main entry point
*************************************/
int main (int argc, char *argv[])
{
    /* DirectFB init */
    DFBCHECK(DirectFBInit( (void*)&argc, (void*)&argv ));
//     DFBCHECK(DirectFBSetOption( "remote", "192.168.1.100"));
//...
    printf("Waiting few seconds\n");
    sleep(5);
    return 1;
}
```

Step5: Create a graphical window on TV

- Call DirectFBCreate() to initiate remote rendering connection
- CreateWindow(), SetOpacity(), RequestFocus() and Flip() to make it visible

```
 #define WIN_W                  856
 #define WIN_H                  240

 static IDirectFB               *dfb;
 static IDirectFBDisplayLayer   *layer;
 static DFBWindowDescription    wdesc;
 static IDirectFBWindow         *gwindow;
 static IDirectFBSurface        *gsurface;

//...
     DFBCHECK(DirectFBCreate( &dfb ));
     /* Obtain the layer */
     DFBCHECK(dfb->GetDisplayLayer(dfb, DLID_PRIMARY, &layer));
     /* Setup the Graphical window */
     wdesc.flags  = ( DWDESC_WIDTH | DWDESC_HEIGHT | DWDESC_OPTIONS | DWDESC_PIXELFORMAT | DWDESC_STACKING);
     wdesc.width  = WIN_W;
     wdesc.height = WIN_H;
     wdesc.pixelformat = DSPF_ARGB4444;
     wdesc.stacking  = DWSC_MIDDLE;
     wdesc.options   = DWOP_NONE;
     DFBCHECK(layer->CreateWindow(layer, &wdesc, &gwindow));
     DFBCHECK(gwindow->GetSurface(gwindow, &gsurface));
     DFBCHECK(gsurface->Clear(gsurface, 0xff,0x00,0x00, 0x80)); //R,G,B,A
     DFBCHECK(gwindow->SetOpacity(gwindow, 0xff ));
     DFBCHECK(gwindow->RequestFocus( gwindow ));
     DFBCHECK(gsurface->Flip( gsurface, NULL, DSFLIP_NONE ));
     printf("Waiting few seconds\n");
     sleep(5);
     if (dfb)
         DFBCHECK(dfb->Release( dfb ));
     return 1;
 }
```

PHILIPS

Step6: Draw using DirectFB primitives

- GetSurface() to obtain drawing surface
- SetColor(),FillRectangle(), DrawString() and Flip() to make it visible

```
 #define BAN_W                  WIN_W
 #define BAN_H                  40

 static DFBFontDescription      fontdesc;
 static IDirectFBFont           *font;
 static char                    text[256];
 static int                     tw,th;
 static DFBRectangle            rect;

// ...
     fontdesc.flags       = DFDESC_HEIGHT | DFDESC_ATTRIBUTES;
     fontdesc.attributes  = DFFA_NONE;
     fontdesc.height      = 20;
     DFBCHECK(dfb->CreateFont(dfb, "./decker.ttf", &fontdesc, &font));
      /* Draw an horizontal banner (filled red rectangle) */
     rect.x = 0;
     rect.y = 0;
     rect.w = BAN_W;
     rect.h = BAN_H;
     DFBCHECK(gsurface->SetColor (gsurface, 0xff, 0x00, 0x00,  0xc0)); // R,G,B, A => Red
     DFBCHECK(gsurface->FillRectangle( gsurface, rect.x, rect.y, rect.w, rect.h ));
     /* Render text */
     strncpy (text, "Hello World !", sizeof(text) );
     DFBCHECK(gsurface->SetFont (gsurface, font));
     DFBCHECK(font->GetHeight  (font, &th));
     DFBCHECK(font->GetStringWidth  (font, text, -1, &tw));
     DFBCHECK(gsurface->SetColor (gsurface, 0xff, 0xff, 0xff,  0xff)); // R,G,B, A => White
     DFBCHECK(gsurface->DrawString (gsurface,text, -1, (WIN_W-tw)/2, 0 + (BAN_H-th)/2, DSTF_TOPLEFT ));
     DFBCHECK(gsurface->Flip( gsurface, NULL, DSFLIP_NONE ));
// ...
```

PHILIPS

Step7: Draw without DirectFB primitives

- You can write “pixels“ from a local buffer the window’s surface
- Write() and Flip() to make it visible

```
 #define R4G4B4A4VAL(r,g,b,a)     ((((a>>4)&0x0f)<<12) | (((r>>4)&0x0f)<<8) | (((g>>4)&0x0f)<<4) | (((b>>4)&0x0f)<<0) )
 #define RGBAVAL R4G4B4A4VAL

 static unsigned short bannerPixelBuffer[BAN_H][BAN_W];

 static void MakeGradient( unsigned short cols[], unsigned char r0, unsigned char g0, unsigned char b0, unsigned char a0,
                           unsigned char r1, unsigned char g1, unsigned char b1, unsigned char a1, int n )
 {
   int i;
   int cr,cg,cb,ca;

   for (i=0; i<n; i++)
   {
     cr = r0 + ((r1-r0)*i)/n;
     cg = g0 + ((g1-g0)*i)/n;
     cb = b0 + ((b1-b0)*i)/n;
     ca = a0 + ((a1-a0)*i)/n;
     cols[i]= RGBAVAL(cr,cg,cb,ca);
   }
 }

// ...
//     DFBCHECK(gsurface->FillRectangle( gsurface, rect.x, rect.y, rect.w, rect.h ));
    int i;
    for (i=0; i<BAN_H; i++)
        MakeGradient(&bannerPixelBuffer[i][0], 0x00,0x00,0x00,0xc0  , 0xff,0xff,0xff,0xf0, BAN_W);
    rect.x = 0;
    rect.y = 0;
    rect.w = BAN_W;
    rect.h = BAN_H-1; // This is a bug!
    DFBCHECK(gsurface->Write( gsurface, &rect, (void *)bannerPixelBuffer, BAN_W*2 ));
```

Step8: Handle keys and events

- CreateEventBuffer(), WaitForEvent() and GetEvent()

```
 static IDirectFBEventBuffer    *events = NULL;
 static int                     quit = 0;

// ...
//     printf("Waiting few seconds\n");
//     sleep(10);
     DFBCHECK(gwindow->CreateEventBuffer( gwindow, &events ));
     /* Event loop */
     while (!quit)
     {
         DFBWindowEvent event;
         events->WaitForEvent( events );
          /* Handle event */
         while (events->GetEvent( events, DFB_EVENT(&event) ) == DFB_OK)
         {
             switch (event.type)
             {
                 case DWET_KEYDOWN:
                     switch (event.key_symbol)
                     {
                         case DIKS_SPACE:
                         case DIKS_OK:
                           quit = 1;
                           break;
                         default:
                           break;
                     }
                     break;
                 default:
                     break;
             }
         }
     }
     if (events)
         DFBCHECK(events->Release( events ));
```

Step9: Handle animations

- The easier is to handle animations within the event loop
  - using timeout on the event
  - e.g. 20ms timeout
- After the new frame generation, do not forget the Flip()

```
//...
     /* Event loop */
     while (!quit)
     {
         DFBWindowEvent event;
//         events->WaitForEvent( events );
         events->WaitForEventWithTimeout( events, 0, 20 );
         // Handle your animation frame here frame
//...
//         DFBCHECK(gsurface->Flip( gsurface, NULL, DSFLIP_NONE ));
```

Step10: Inject keys

- Injecting keys can be done using the jslibrc_client helper library
  - $(DIRECTFB_VOODOO)/lib/libjslibclient.o\
    must be added to LDADD in makefile.voodoo
- Call jslibrc_Init() to initiate remote control connection
- Use jslibrc_KeyUp() and jslibrc_KeyDown() to send RC6 commands

```
#include <jslibrc_client.h>

int main (int argc, char *argv[])
{
    jslibrc_Init( &argc, &argv );

    jslibrc_KeyDown( keySourceRc6, 0, rc6S0Yellow );
    jslibrc_KeyUp( keySourceRc6, 0, rc6S0Yellow );

    jslibrc_Exit();
    return 1;
}
```

- A tool of the SDK called “remco” can be used to test key injection

```
$ cd /js/DirectFB141_2k11/Jslib/tools/
$ ./remco HOME
```

Optimizing GFX rendering

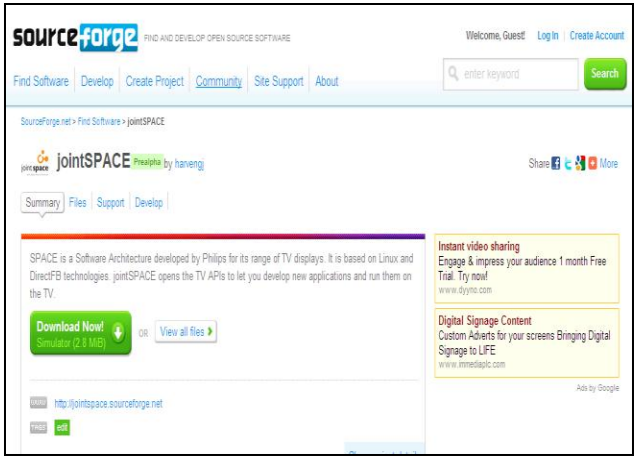
- Be aware that every DirectFB call goes through the network
- Be aware that resources are limited at TV side
  - DirectFB objects are created in TV memory
    - Fonts, Bitmaps, Windows must fit in the remaining 3.5 MB
  - All operations are executed at TV side
    - In a low priority process
    - In the use case currently selected
  - Operation on pixel buffers are RLE compressed (e.g. write())
    - RLE is decompressed at TV side by the CPU
  - Limit the frame rate or limit the window size
    - Max window size is 1280x720x16bits(32bits in 2k10) in MIDDLE layer only

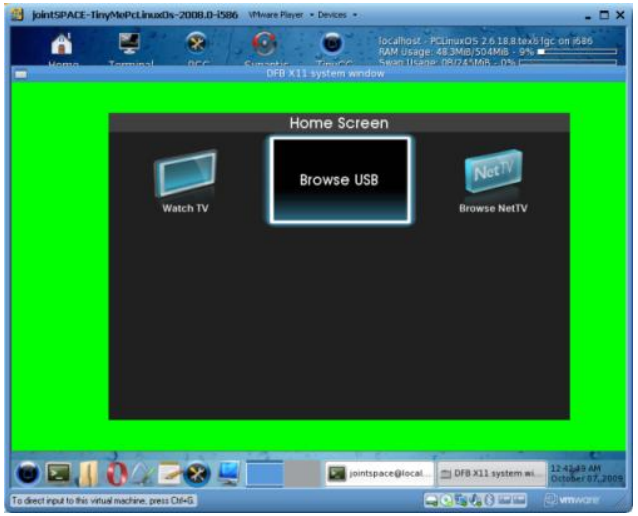
PHILIPS

Questions?

http://jointspace.sourceforge.net

- Infrastructure available:
  - Wiki for documentation
  - Forum for discussions and File repository
  - A Web page to wrapping everything
- Documentation:
  - SPACE porting guides, papi, plfApi, amApi
  - Install procedure and various presentations
  - Tutorial to develop Remote Applications
- Simulator:
  - Basic SPACE packages for Linux PC
  - All in source code and documented in the Wiki

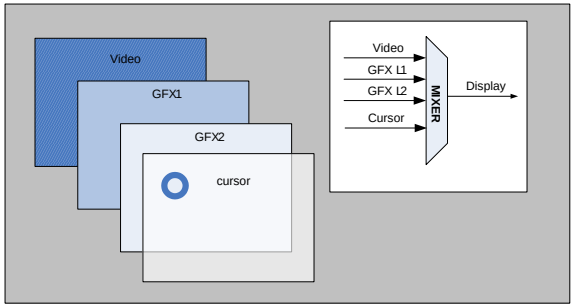




Important links

- Overall jointSPACE web site: http://jointspace.sourceforge.net/
- File repository: http://sourceforge.net/apps/mediawiki/jointspace/index.php?title=Main_Page
  (latest SDK, code sample in source, jointSPACE simulator)
- Wiki Documentation: http://sourceforge.net/apps/mediawiki/jointspace/index.php?title=Main_Page

Windows, client surface and layer (1/3)



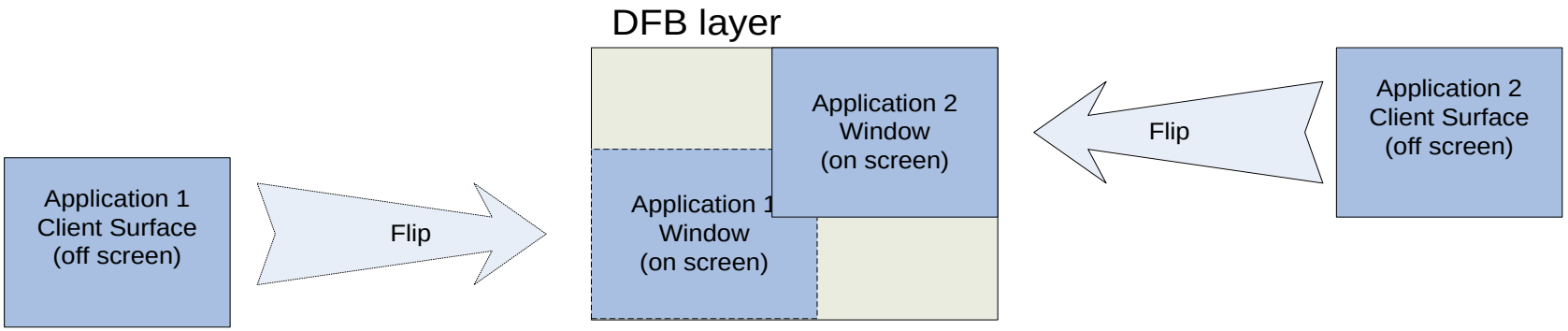
- DirectFB handles HW graphical **layers**
  - Graphical Layers are pointing to memory
    - Memory is interpreted and presented to the screen
    - According to the layers configuration
      - pixel format as for e.g. ARGB32bits,
      - width, height,…
  - GFX layers are mixed with video by the mixer component
    - Layers are read at 50/60/100/120Hz
  - DirectFB share GFX layers between the various applications

- Applications create **windows**
  - From which a **client surface** can be obtained (client buffer)
  - Windows can be configured independently (pixel format, width, height…)

Windows, client surface and layer (2/3)

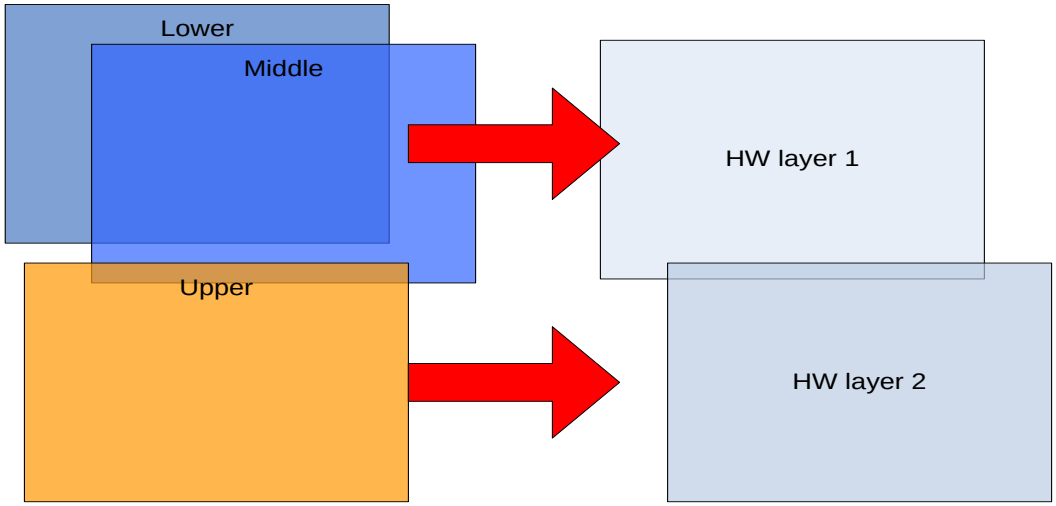
- The application draws into the **client surface**
  - Using DirectFB drawing primitives

- The application signal the update (full screen or partial) by "**flipping**" its surface (or a portion of it) into the layer'surface
  - The data is copied taking into account the Window's position in the layer



Windows, client surface and layer (3/3)

- The window manager of DirectFB supports windows z-ordering
- Next to the z-order, it is possible to group windows in a kind of virtual layers
  - 3 individual **windows stack** are supported
    - LOWER, MIDDLE, UPPER
  - The stack where the window should end up is specified at window creation
  - Each windows stack is statically mapped to a HW Layer



Source and Destination regions of a window (1/2)

- The **position of the windows** on screen (window bounds) is determined by the **Application Manager**
  - controlling the Window Manager (SaWMan)

- It is also possible **for the client**
  - to specify a **source region** of its windows'surface that should be used as window buffer by the Window Manager
  - to define a **destination region**, relative to its window bounds, that the Window Manager should take into account at rendering time:

PHILIPS

Source and Destination regions of a window (2/2)