About Serial Lines

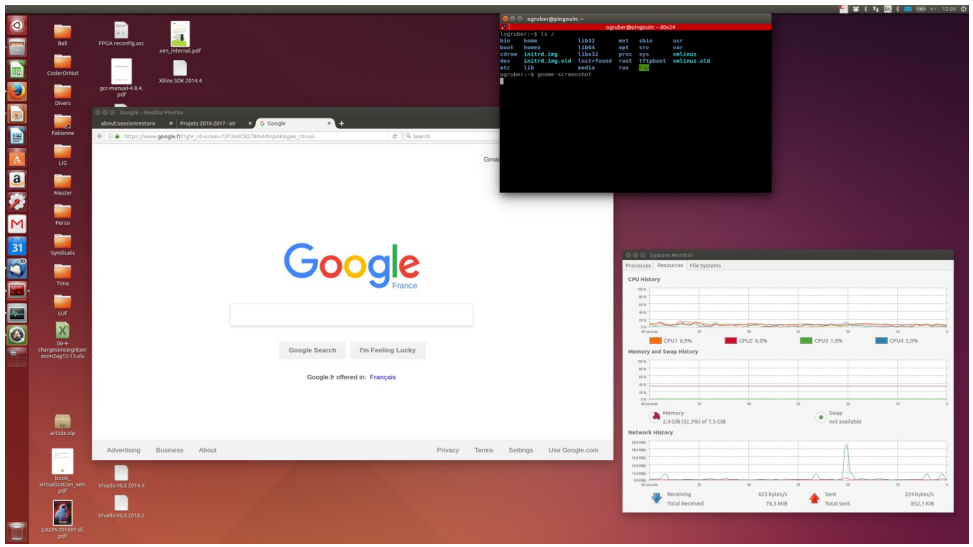
Pr. Olivier Gruber
(olivier.gruber@imag.fr)

Laboratoire d'Informatique de Grenoble

Université de Grenoble-Alpes

Stepping Back...

- *Connecting the pieces together…*
  - *Terminal window*
  - *Shell and forked processes*
  - *QEMU process*
  - *Serial line*
  - Bare-metal software/hardware



Processes

```c
#include <stdio.h>
#include <stdlib.h>

int main(void) {
  printf("Hello World!");
  return EXIT_SUCCESS;
}

```

QEMU

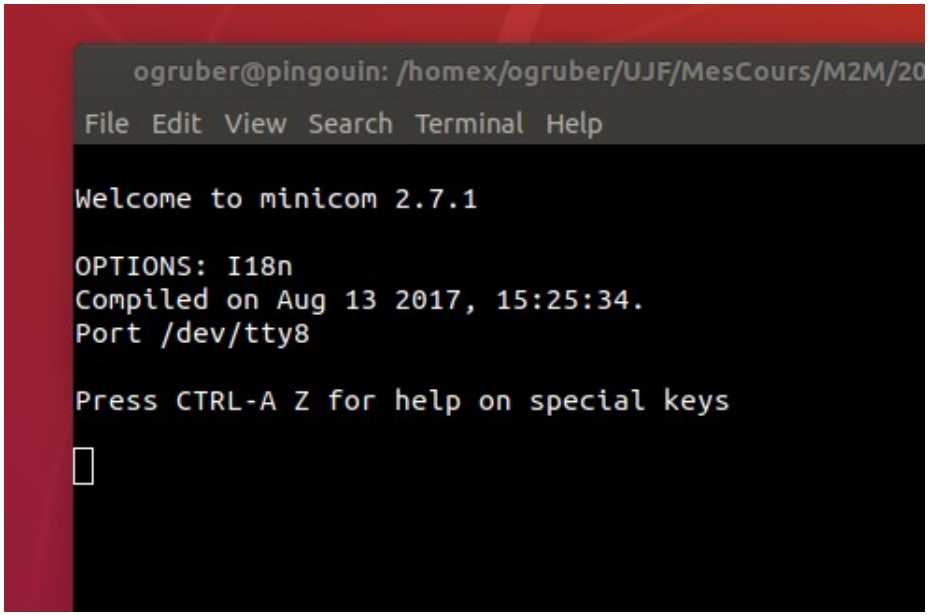
Serial Lines

Yesterday

Today

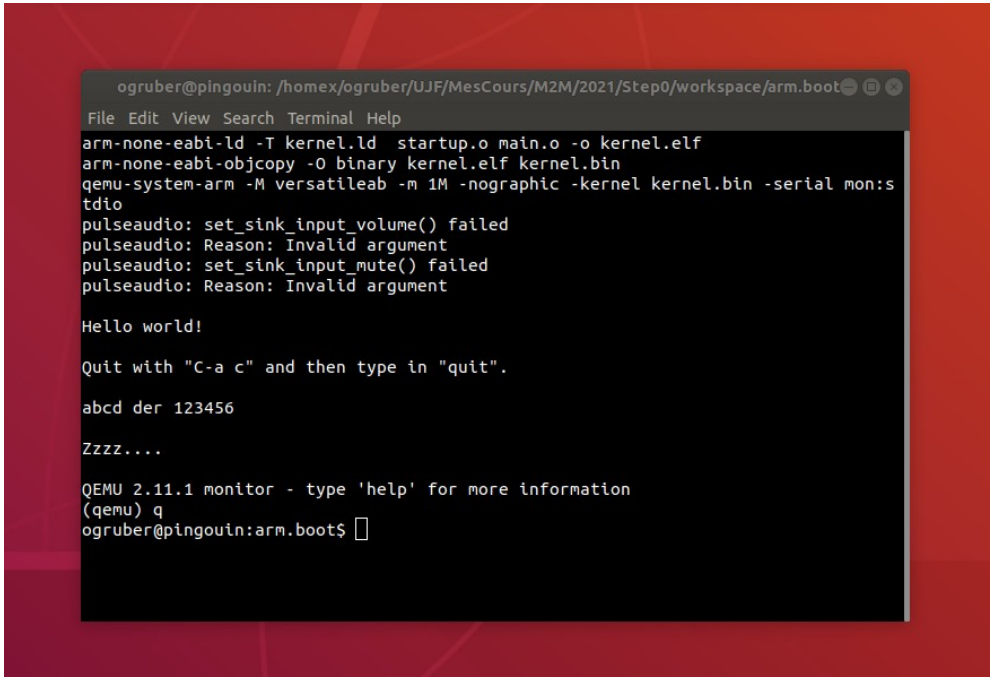
Real Board – Real Serial Line

- Bare-metal Software
  - Runs directly on the "bare metal"
  - Use the serial line as "stdin" and "stdout"
- Developer
  - Runs a shell in a terminal
  - Launches "minicom" on the serial line */dev/tty8*
  - Interacts in the terminal window



QEMU – Emulated Board and Serial Line

- Bare-metal Software
  - Nothing changes…
  - Runs directly on the "bare metal"
  - Use the serial line as "stdin" and "stdout"
- Developer
  - Still runs a shell in a terminal
  - Now launches QEMU
  - Still interacts in the terminal window



Experiment with:
- **try out moving with arrows…**
- **try out typing anywhere in the window…**
- **compare with the behavior of your normal terminal/shell window**

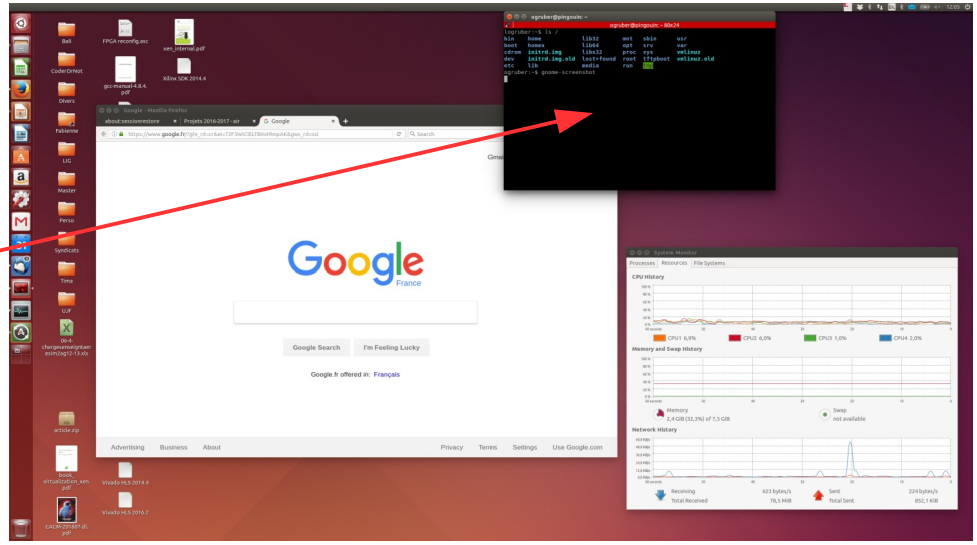
Question: what is going on?

Let‘s track back to a regular "terminal window" and "shell"…

Window Manager – Shell Example

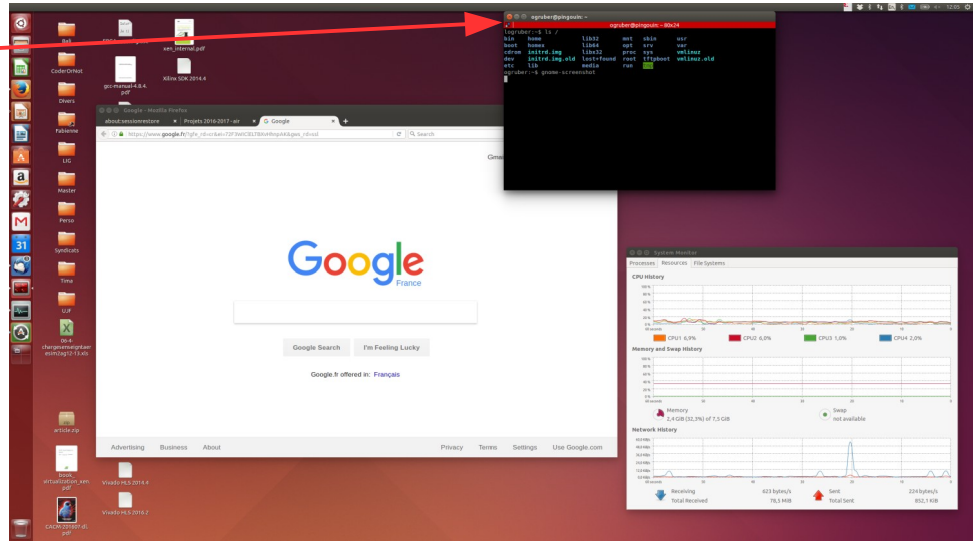
How does a shell work?

What is the relation with the "terminal window"?



Window Manager – Shell Example

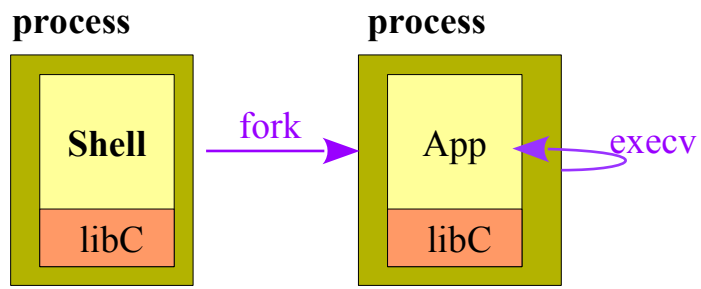
How does a shell work?

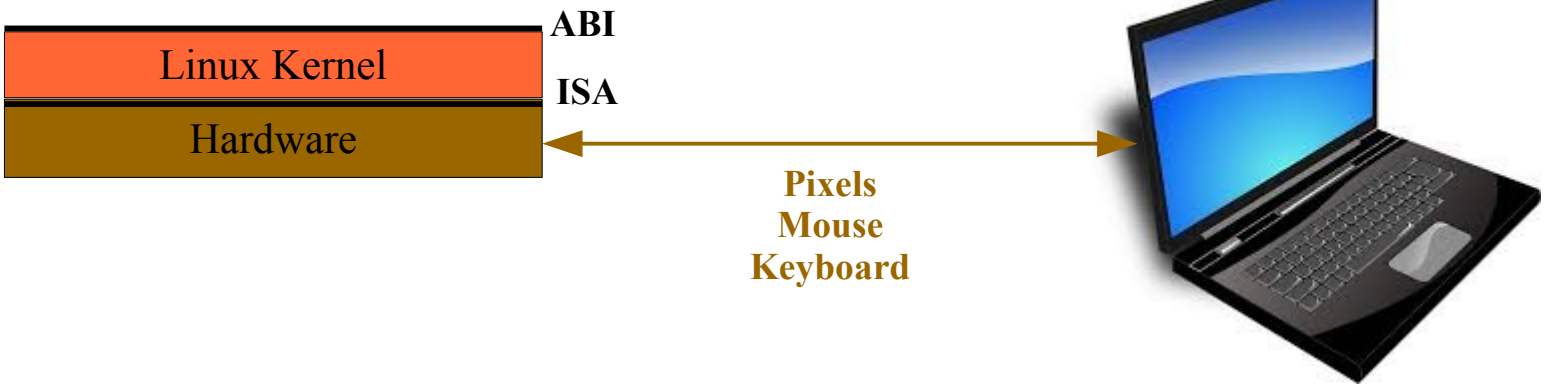


→ **fork** to create a process (copy)

→ **execv** to load an executable
and to pass arguments (argc,argv)

→ **share stdin/stdout/stderr**

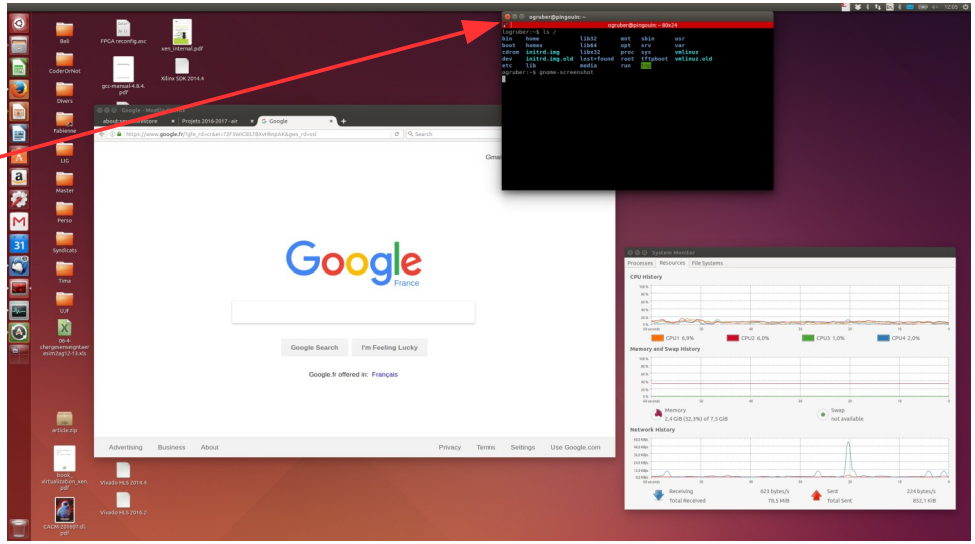


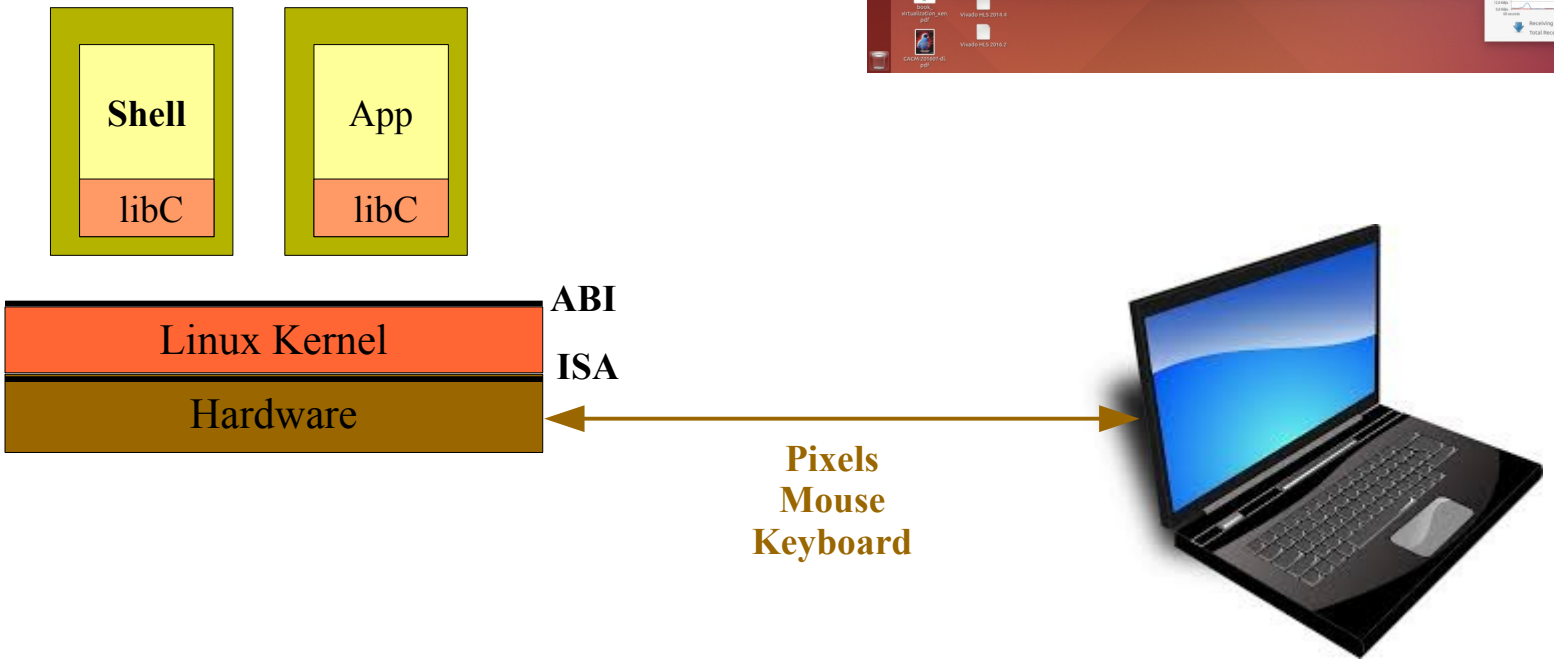


Window Manager – Shell Example

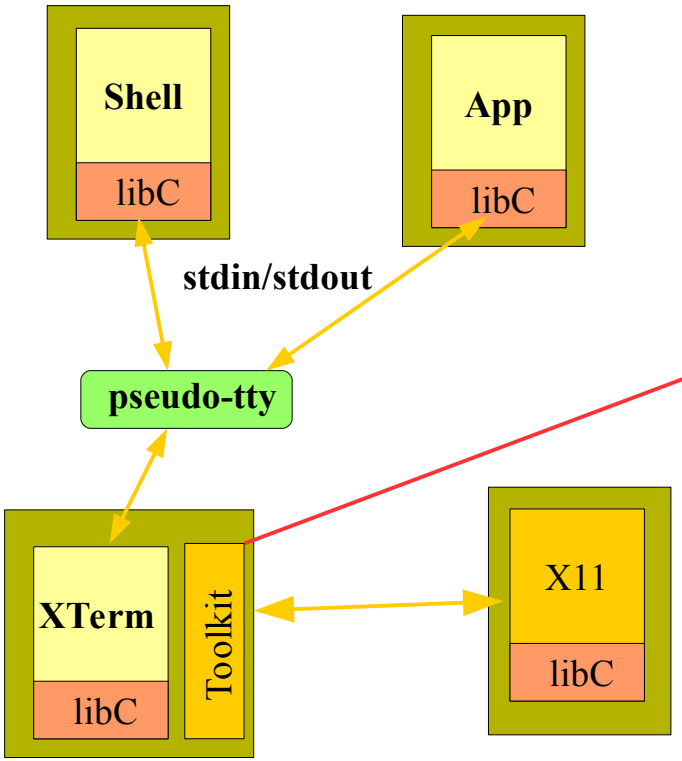
So whose window is that *terminal window*?

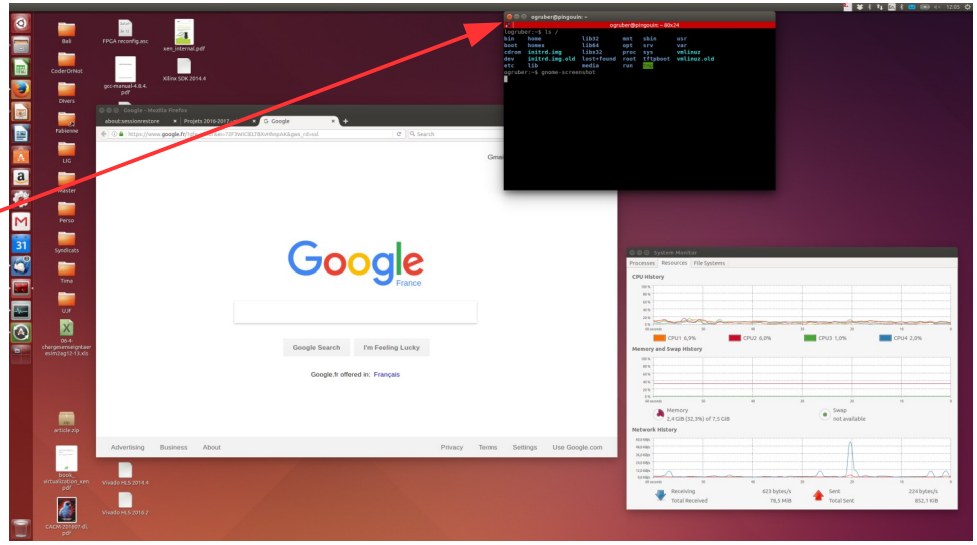
And what is the relationship with stdin/stdout?

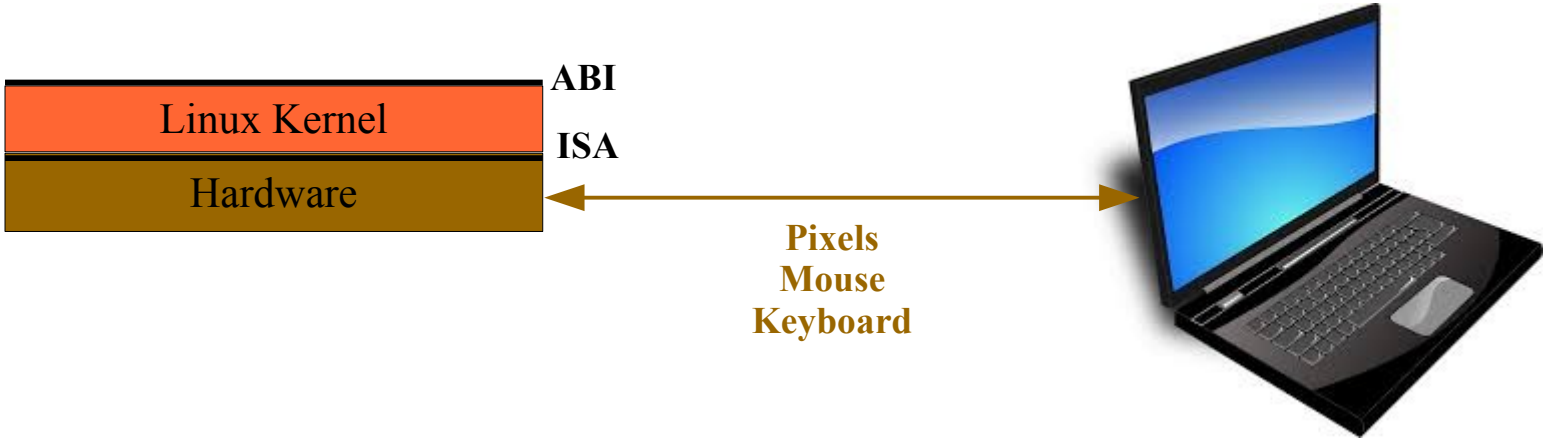




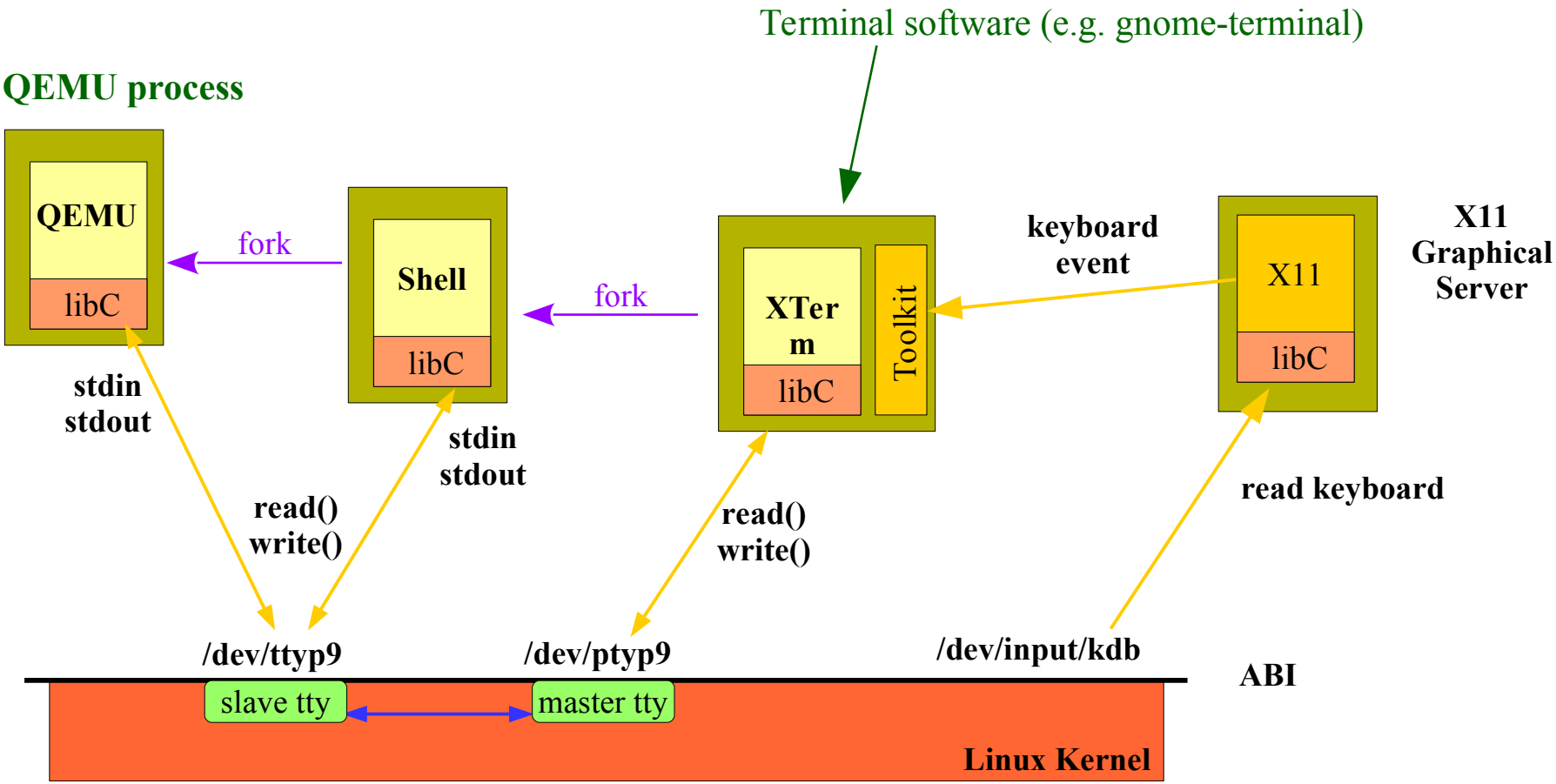
Window Manager – Shell Example







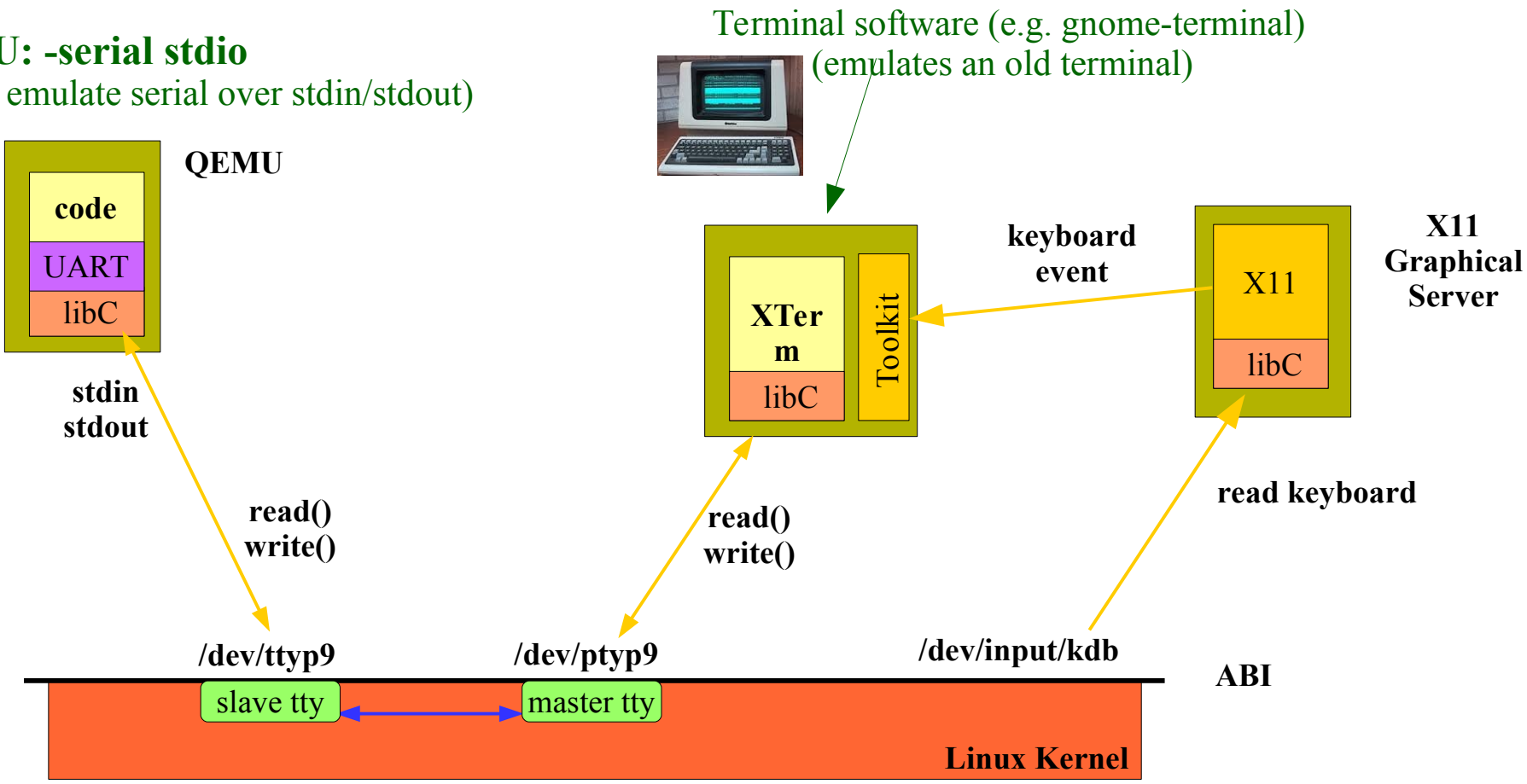
Launching QEMU



For more infos: http://www.linusakesson.net/programming/tty/

QEMU – Serial Line over STDIO

QEMU: -serial stdio
(means emulate serial over stdin/stdout)



For more infos: http://www.linusakesson.net/programming/tty/

How your code sees the world...

Hardware

Software

```
void c_entry() {
  int i = 0;
  int count = 0;
  while (1) {
    unsigned char c;
    if (0==uart_receive(UART0,&c))
      continue;
    if (c == 13) {
      uart_send(UART0, '\r');
      uart_send(UART0, '\n');
    } else {
      uart_send(UART0, c);
    }
  }
}
```

(1) It is your code that does the echo on the screen,
by sending what it receives…

(2) A terminal understands 8-bit encoding of a character set

May be simple **ASCII characters** (*http://ascii-table.com/*)

May be some other **encoding** like *Unicode – UTF-8.*

Nota Bene: *the same is true for stdin/stdout in C programs*

How your code sees the world...

Hardware

Software

```
void c_entry() {
  int i = 0;
  int count = 0;
  while (1) {
    unsigned char c;
    if (0==uart_receive(UART0,&c))
      continue;
    if (c == 13) {
      uart_send(UART0, '\r');
      uart_send(UART0, '\n');
    } else {
      uart_send(UART0, c);
    }
  }
}
```

(1) It is your code that does the echo on the screen,
by sending what it receives…

(2) A terminal understands 8-bit encoding of a character set

(3) A terminal also understands ***escaped sequences***
Like moving the cursor on the screen
(http://ascii-table.com/ansi-escape-sequences.php)

Nota Bene: *this is how certain C programs display an interface in the terminal window, like the program "top" or "htop" or "nano"*

Escape Sequences

Escape Sequences (http://ascii-table.com/ansi-escape-sequences.php)

Cursor Up: **Esc[**Value**A**

Moves the cursor up by the specified number of lines without changing columns.
If the cursor is already on the top line, ANSI.SYS ignores this sequence.

Cursor Down: **Esc[**Value**B**

Moves the cursor down by the specified number of lines without changing columns.
If the cursor is already on the bottom line, ANSI.SYS ignores this sequence.

Cursor Forward: **Esc[**Value**C**

Moves the cursor forward by the specified number of columns without changing lines.
If the cursor is already in the rightmost column, ANSI.SYS ignores this sequence.

Cursor Backward: **Esc[**Value**D**

Moves the cursor back by the specified number of columns without changing lines.
If the cursor is already in the leftmost column, ANSI.SYS ignores this sequence.

Set the cursor position:

Esc[Line**;**Column**H**
Esc[Line**;**Column**f**

Moves the cursor to the specified position (coordinates).
If you do not specify a position, the cursor moves to the home position
at the upper-left corner of the screen (line 0, column 0).

Serial Line – Past and Future

- Serial line
  - Easy setup, allows for early printing and early interaction
- Usage in bare-metal programming
  - Allows to do early debugging by printing…
  - Allows to provide a simple command-line interface to embedded systems
  - Examples: embedded systems in cars, spacecrafts, ...
- Usage with a Linux kernel
  - Kernel bootstrap
    - Supports early printing via kprintf
  - First process
    - At the end of the boot sequence, the kernel creates the first process
    - The serial line becomes the stdin/stdout of that first process
  - Graphics?
    - A terminal is emulated above the graphic card
    - It is the terminal you see when booting Linux before some fancy graphics show up