

**The 2017 International Conference on
Field-Programmable Technology**
Melbourne, Australia, December 11 - 14, 2017



 **POLITECNICO DI MILANO**



FPT Tutorial

**BAMBU: An open-source
framework for research in
high-level synthesis**

Fabrizio Ferrandi - Christian Pilato

- ❑ Presentation of bambu
- ❑ Target Customization and Tool Integration
- ❑ Compiler Based Optimizations, Tuning and Customization of Generated Accelerators
- ❑ Synthesis and optimization of memory accesses
- ❑ Debugging and Automated Bug Detection for Hardware Generated with bambu

□ Christian Pilato

- ▶ Postdoc Researcher– USI Lugano



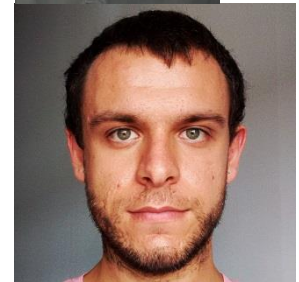
□ Marco Lattuada

- ▶ Postdoc Researcher – Politecnico di Milano



□ Pietro Fezzardi

- ▶ PhD Student – Politecnico di Milano



□ Fabrizio Ferrandi

- ▶ Associate Professor – Politecnico di Milano

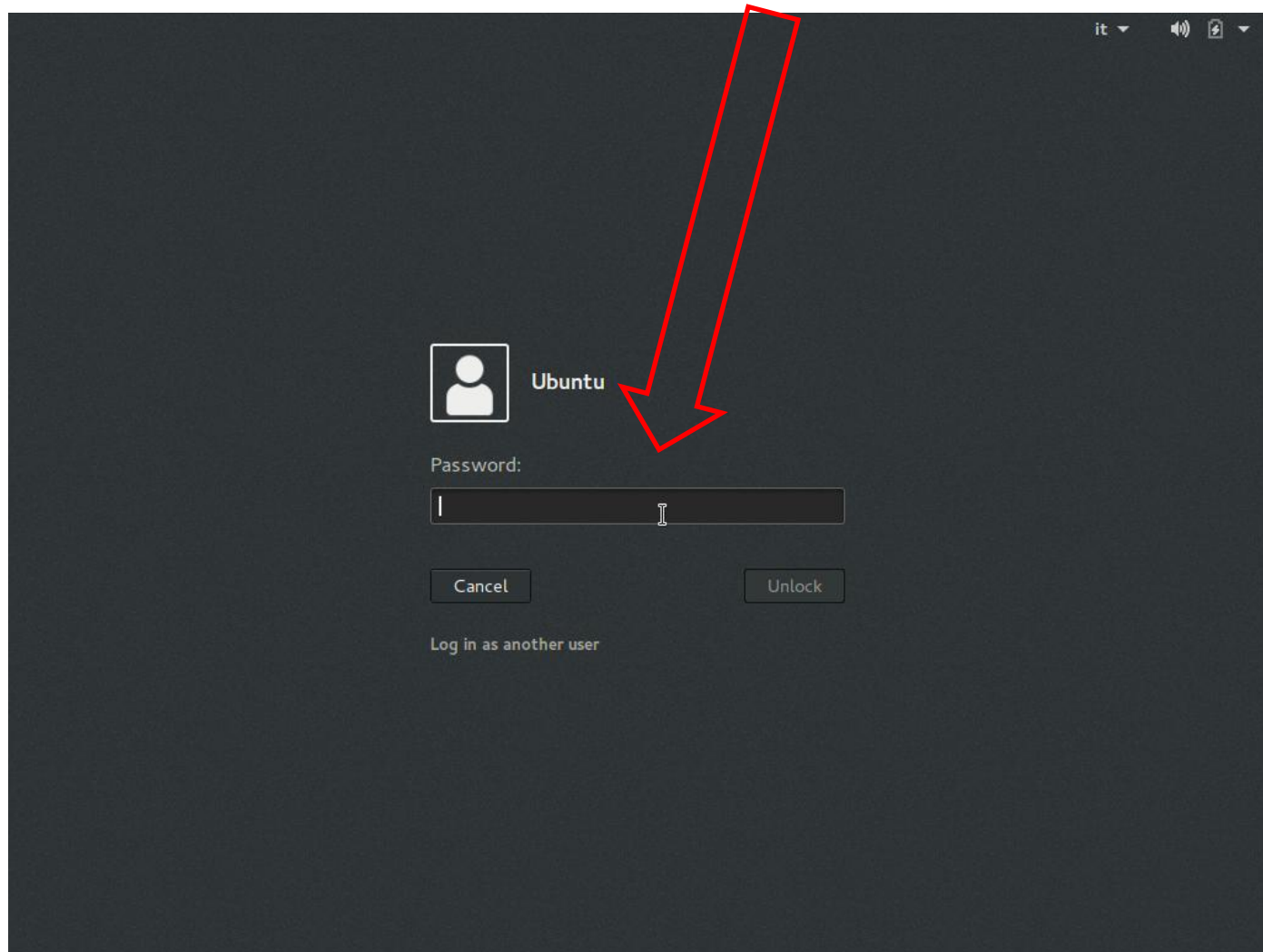


- ❑ Slides and the VM with a pre-configured environment can be downloaded from:

https://panda.dei.polimi.it/?page_id=767

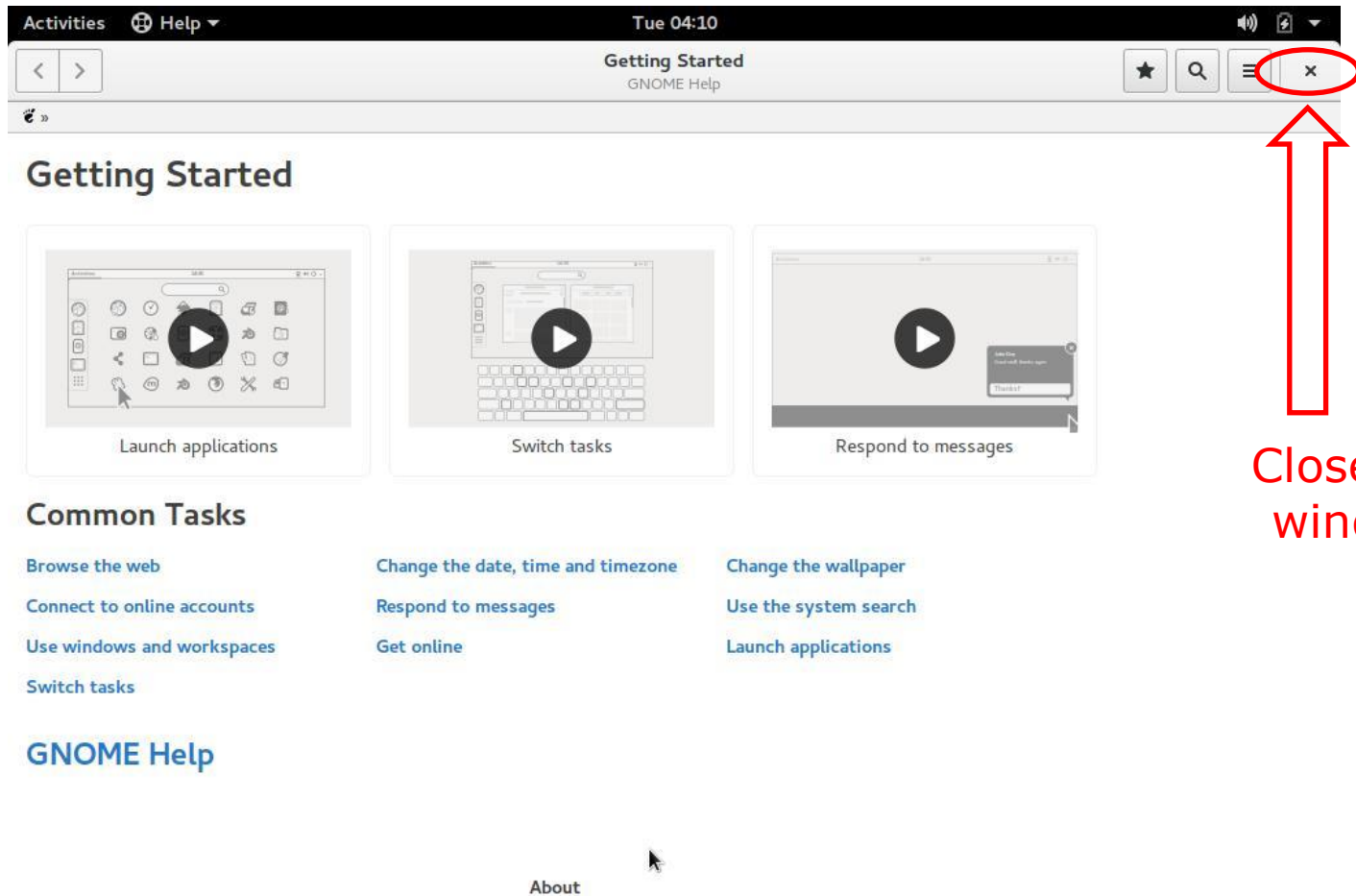
- ❑ There are two .ova files. One for 32bit operating systems and another one for 64bit operating systems.
- ❑ Once downloaded you can import the virtual machine by clicking on the VirtualBox Manager menu:
 - ▶ File
 - Import appliance...

Type as a password: **password**



Starting screen

7



Configure the keyboard

8

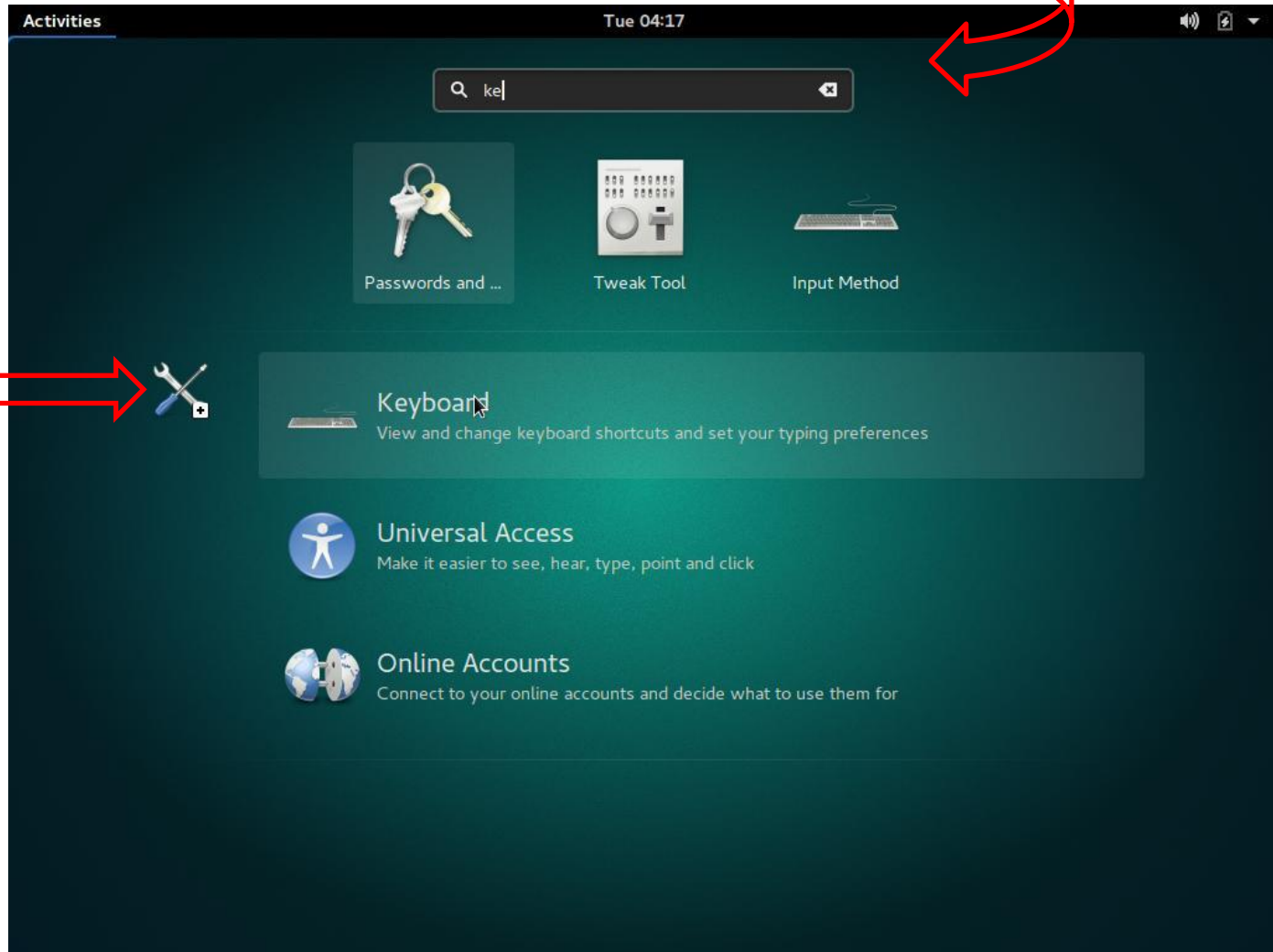
1) Click on **activities**



2) Search for "**keyboard**"

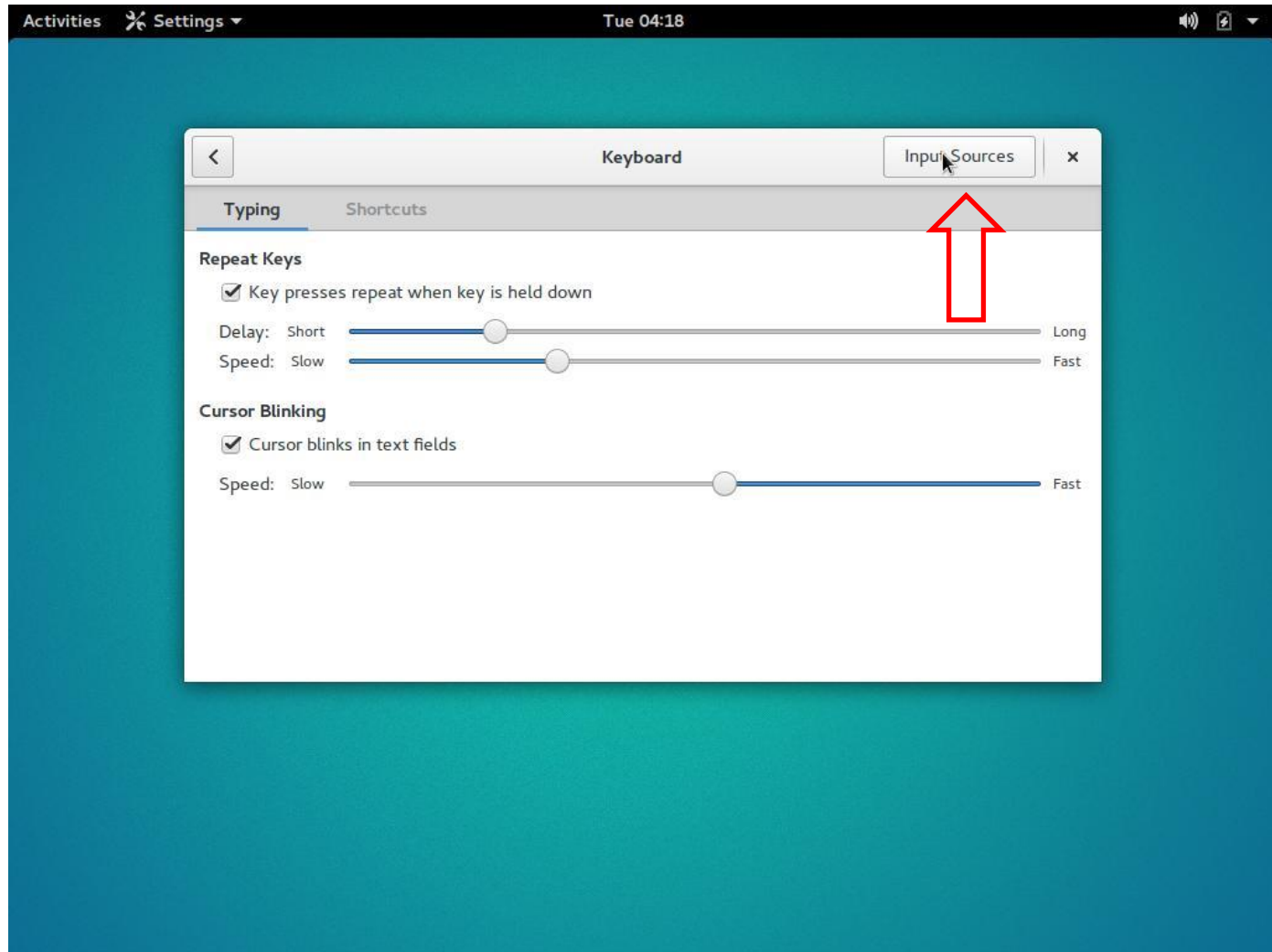


3) Select keyboard



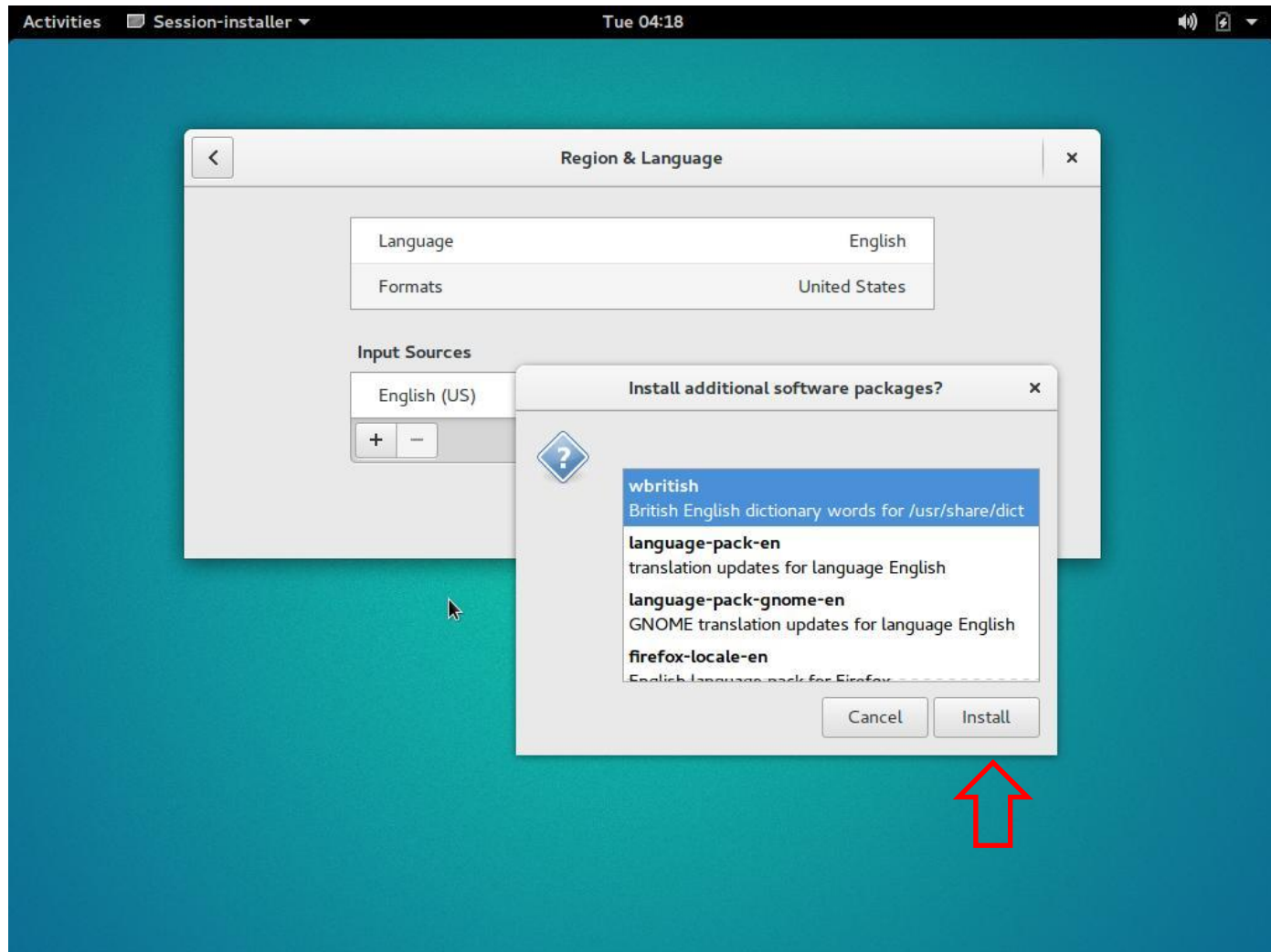
Select the input sources

9



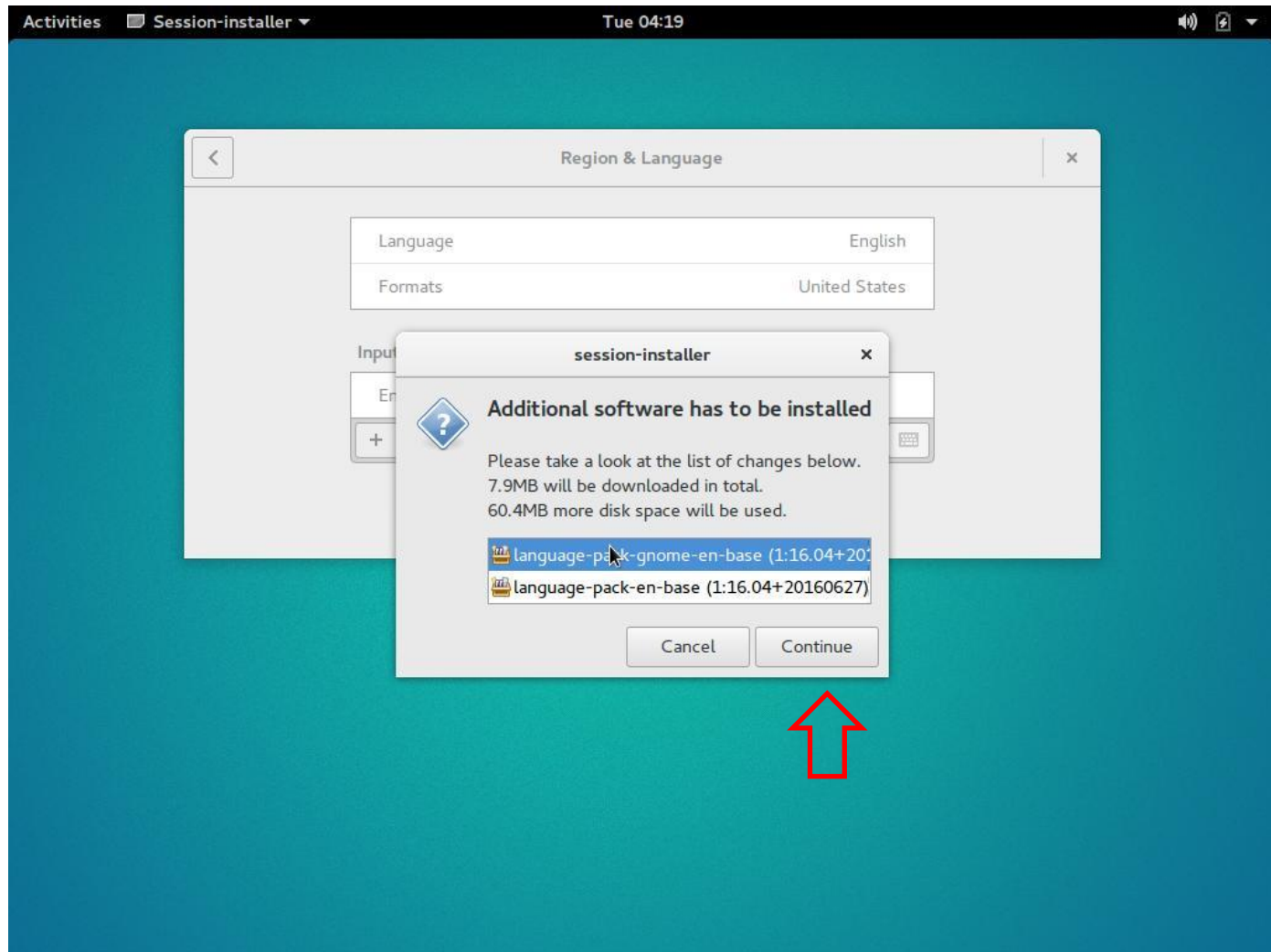
Install additional packages

10



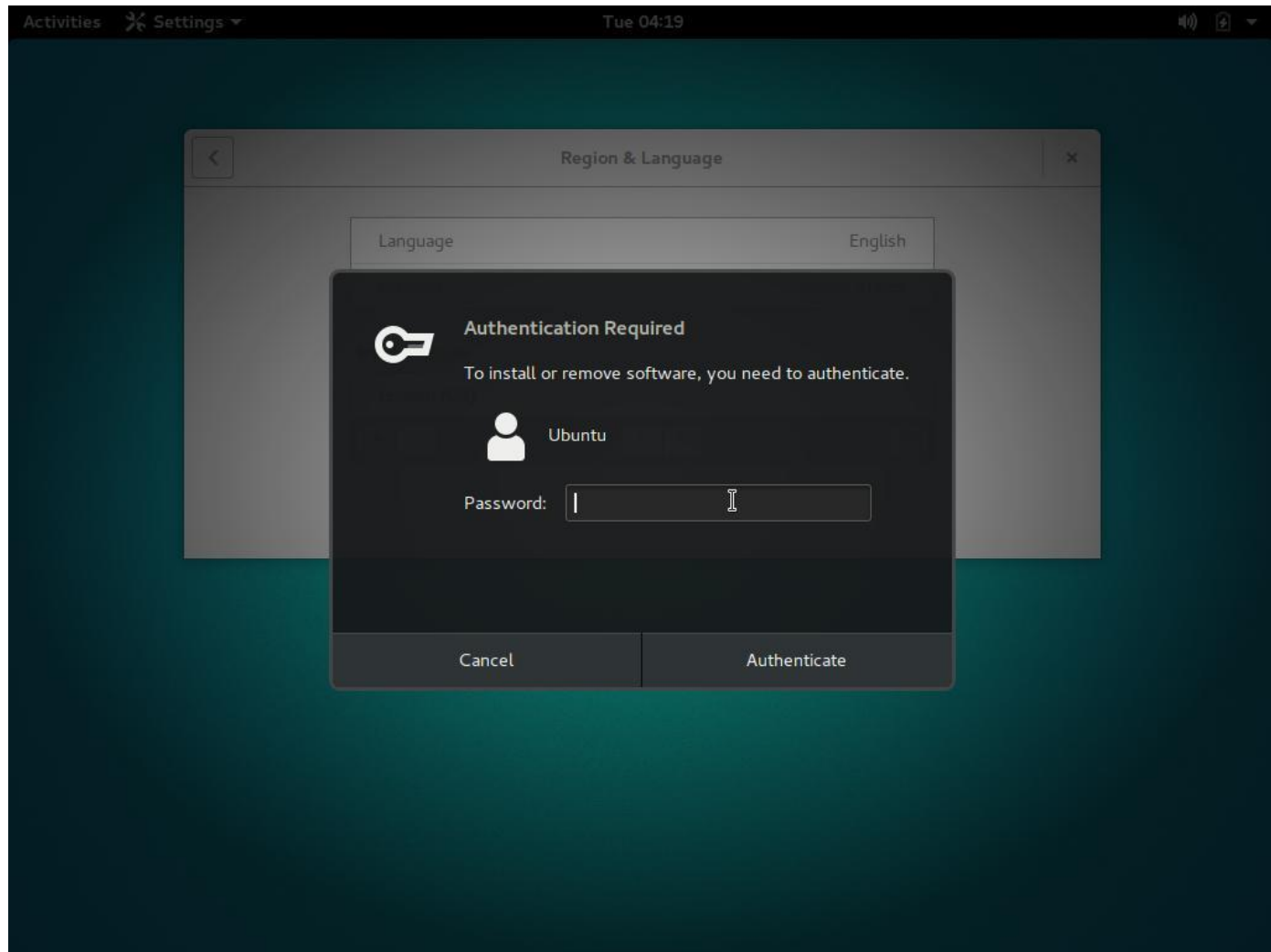
Confirm the installation

11



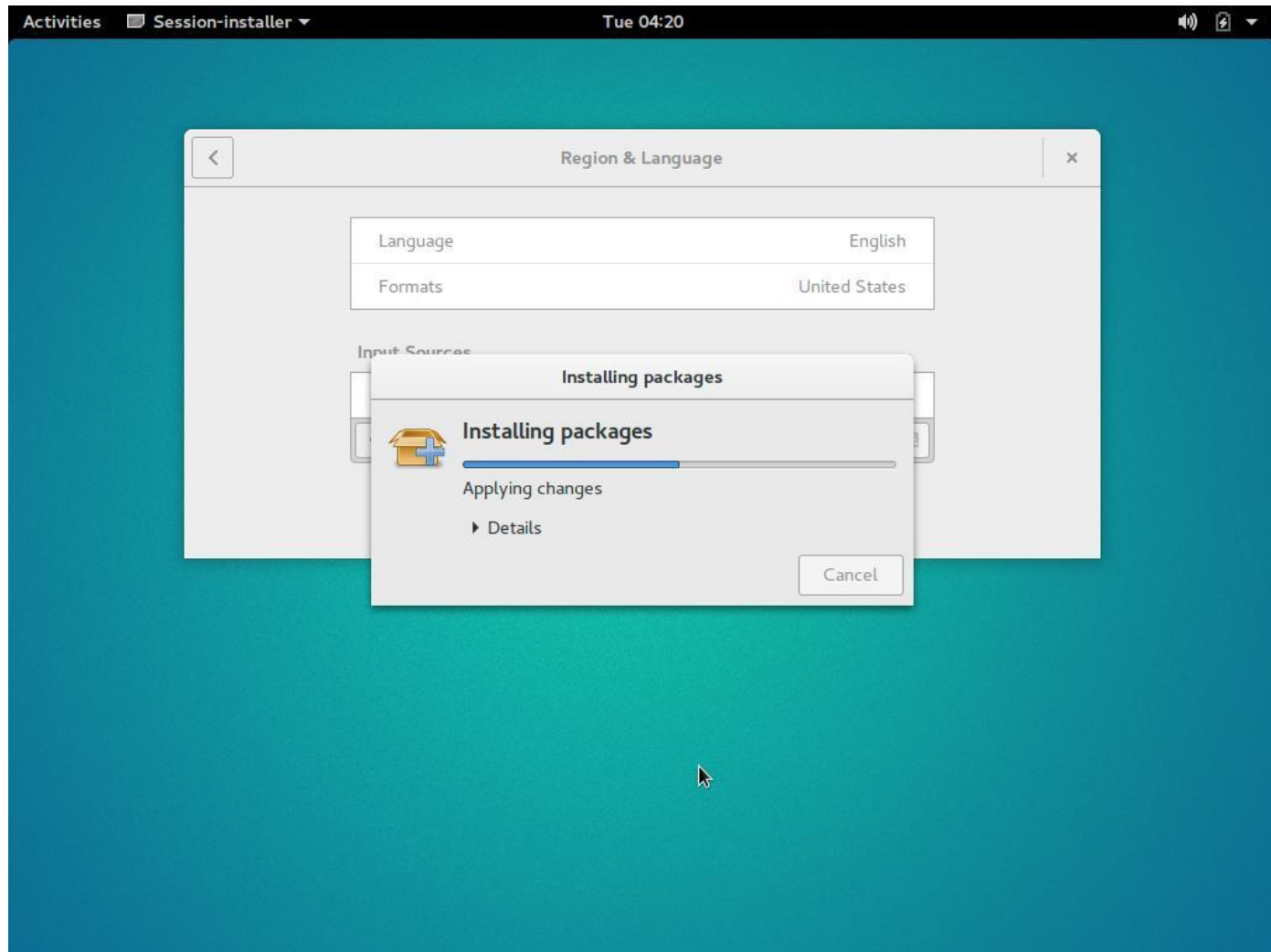
Type as a password: **password**

12



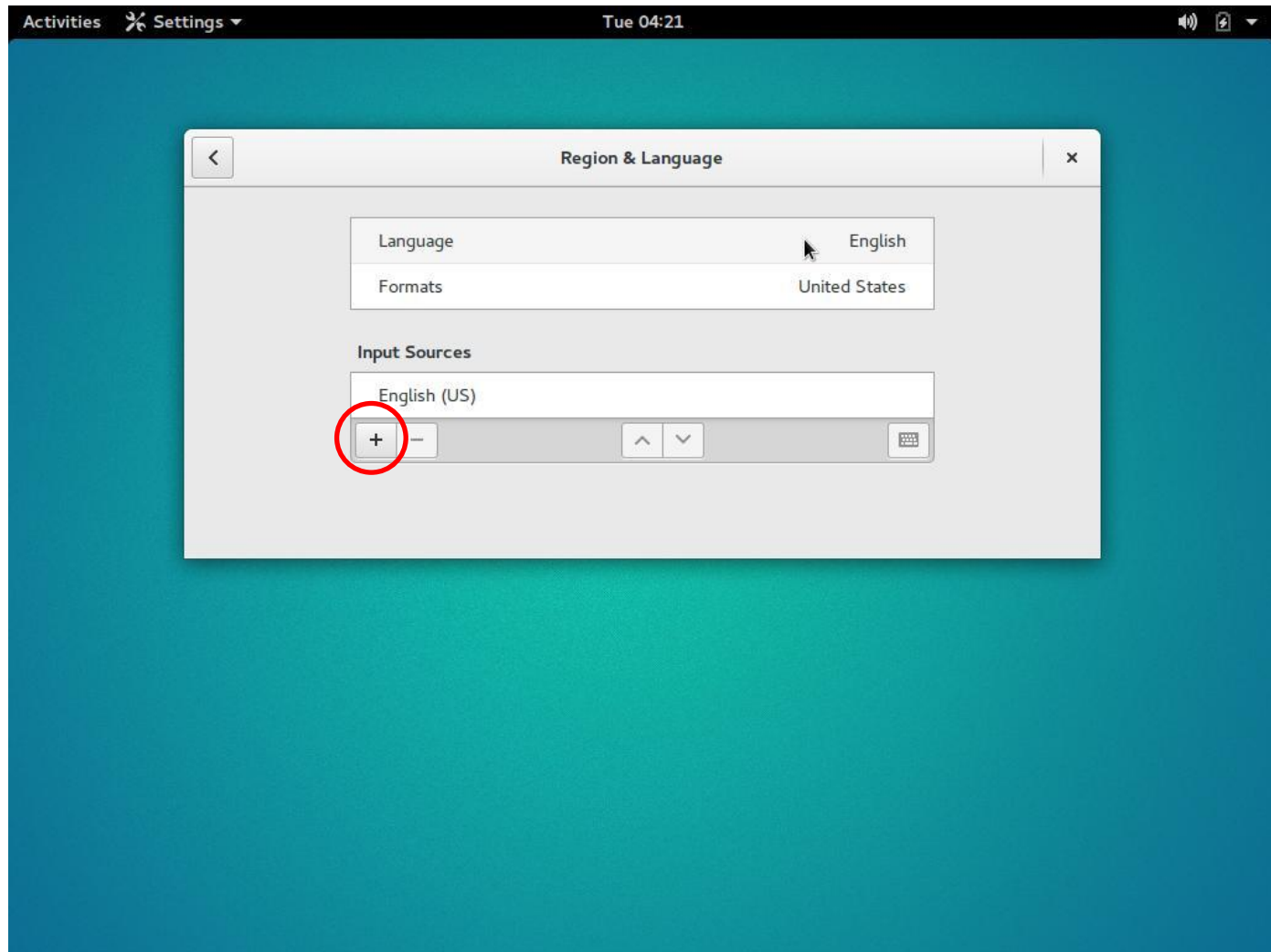
Wait a while

13



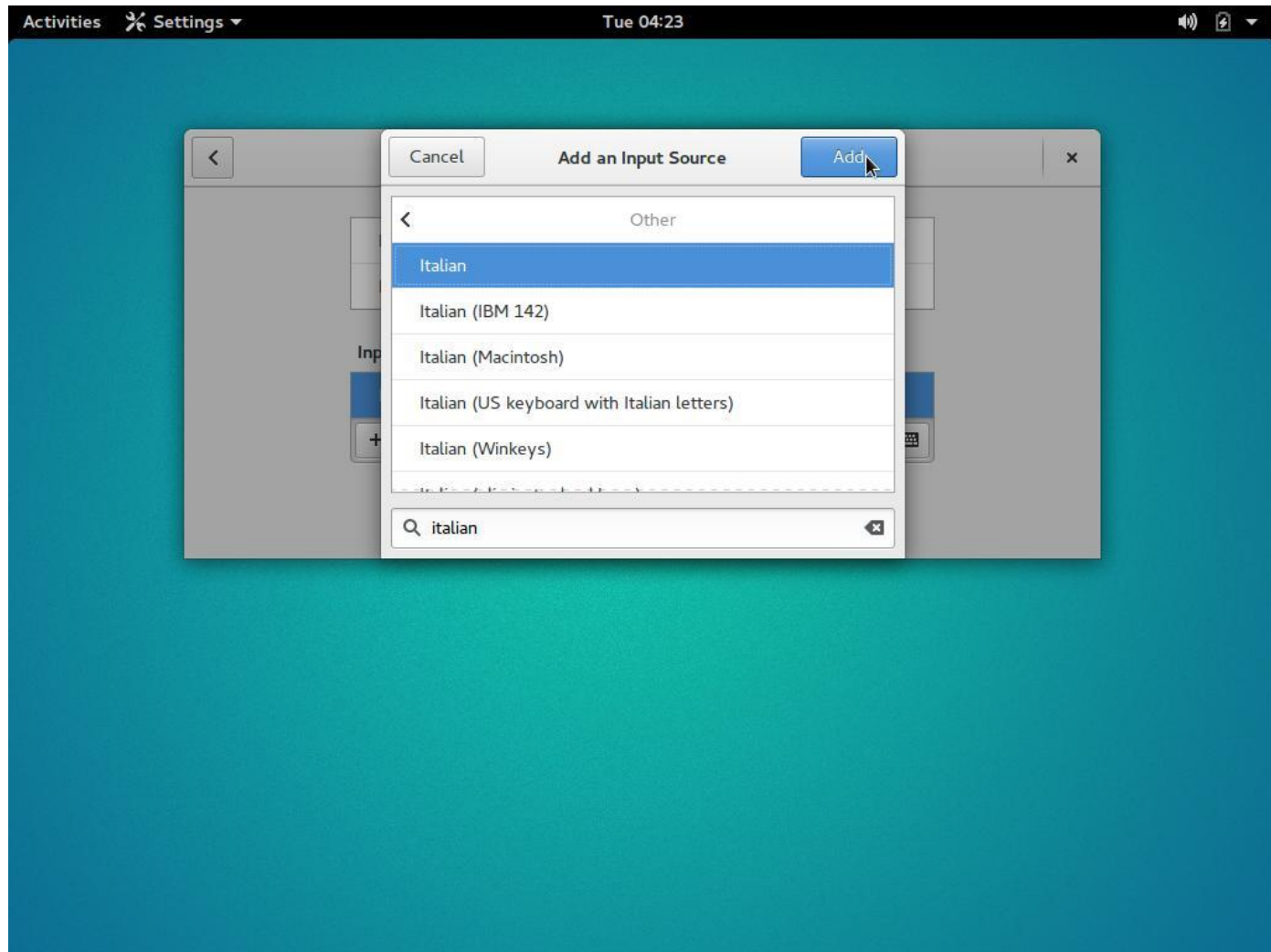
Add a new input source

14



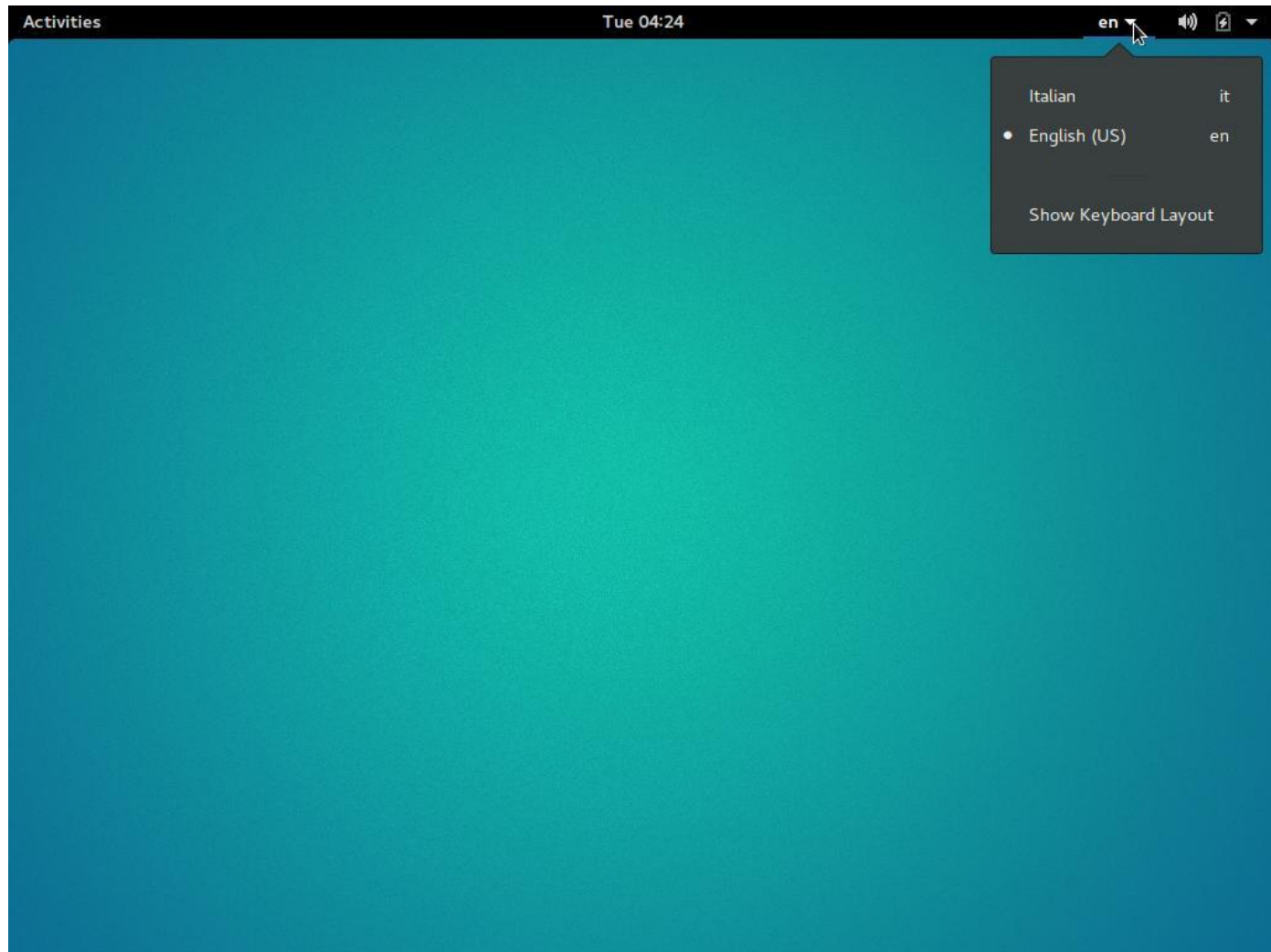
Type the keyboard language

15



Activate the keyboard layout

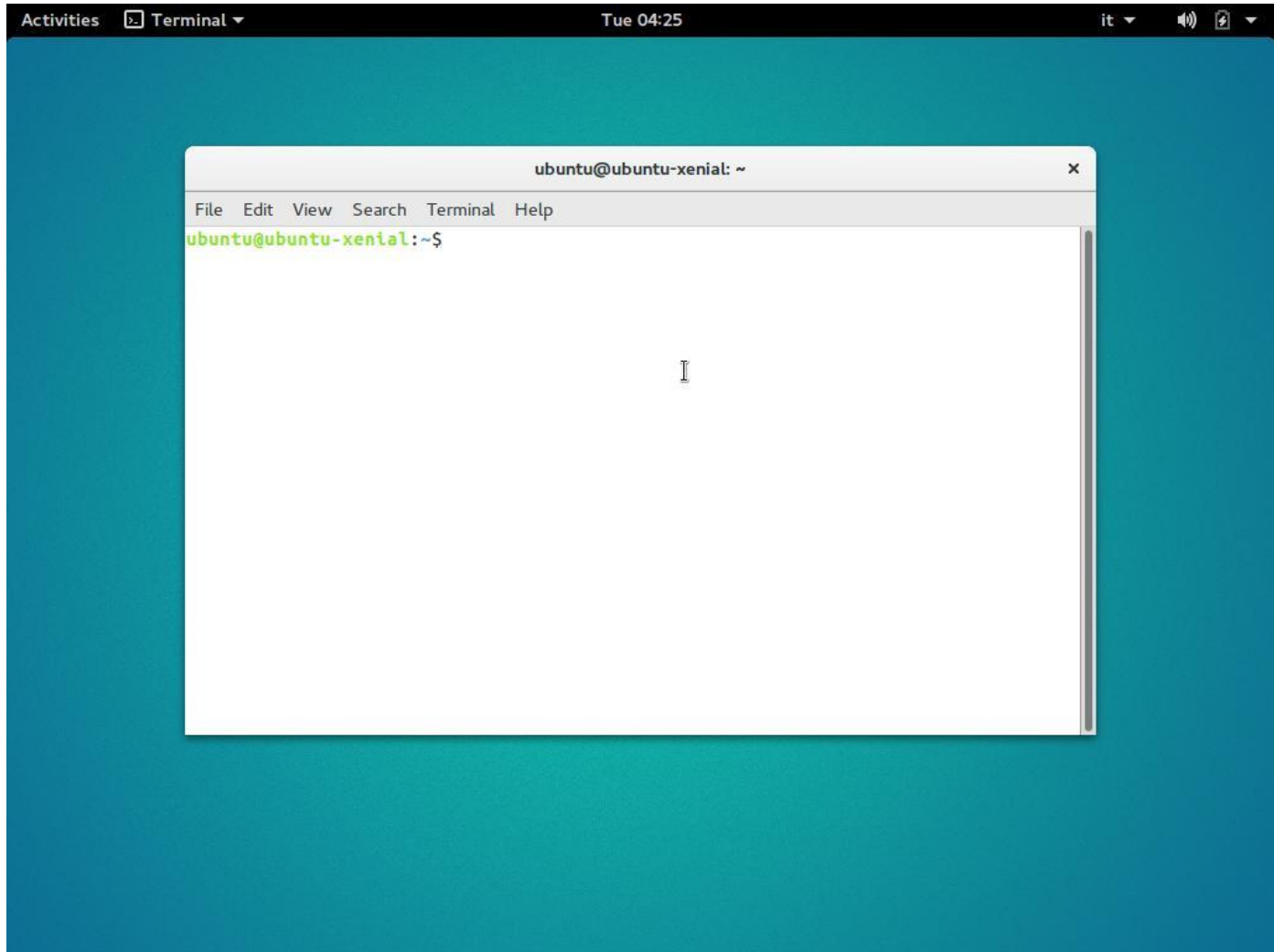
16



Open a terminal

17

Press simultaneously: **ctrl alt t**



- ❑ Change the directory to:

```
cd ~/PandA-bambu
```

- ❑ And then

```
git pull
```

- ❑ Switch to the FPT branch:

```
git checkout FPT17_tutorial
```

❑ Change the directory by typing:

```
cd ~/PandA-bambu/documentation/tutorial_fpl_2017/intro/first
```

❑ Edit the file C by typing:

```
gedit module.c
```

```
unsigned short icrc1(unsigned short crc, unsigned char onech)
{
    int i;
    unsigned short ans=(crc^onech << 8);
    for (i=0;i<8;i++) {
        if (ans & 0x8000)
            ans = (ans <<= 1) ^ 4129;
        else
            ans <<= 1;
    }
    return ans;
}
```

□ Run the script by typing:

```
./bambu.sh
```

```
Summary of resources:
- ASSIGN_SIGNED_FU: 2
- IIconvert_expr_FU: 1
- IUdata_converter_FU: 2
- MUX_GATE: 2
- UIdata_converter_FU: 3
- UUdata_converter_FU: 1
- bit_xor_expr_FU: 1
- constant_value: 5
- ge_expr_FU: 1
- lshift_expr_FU: 1
- ne_expr_FU: 1
- plus_expr_FU: 1
- read_cond_FU: 1
- register_SE: 2
- ui_bit_xor_expr_FU: 1
- ui_cond_expr_FU: 1
- ui_lshift_expr_FU: 1

Start reading vector          1's values from input file.

Value found for input crc: 0000000000001010
Value found for input onech: 00000010
Reading of vector values from input file completed. Simulation started.
Value found for output ex_return_port: 0010101001000010
  return_port = 10818    expected = 10818

Simulation ended after          10 cycles.

Simulation completed with success

- HLS_output//simulation/testbench_icrcl_minimal_interface_tb.v:441: Verilog $finish
1. Simulation completed with SUCCESS; Execution time 10 cycles;
  Total cycles           : 10 cycles
  Number of executions   : 1
  Average execution      : 10 cycles
```

- ❑ Testbench generated automatically
 - ▶ test_icrc1.xml

- ❑ Simulation and synthesis scripts generated automatically:
 - ▶ icrc1/simulate_icrc1_minimal_interface.sh
 - ▶ icrc1/synthesize_Synthesis_icrc1_minimal_interface.sh

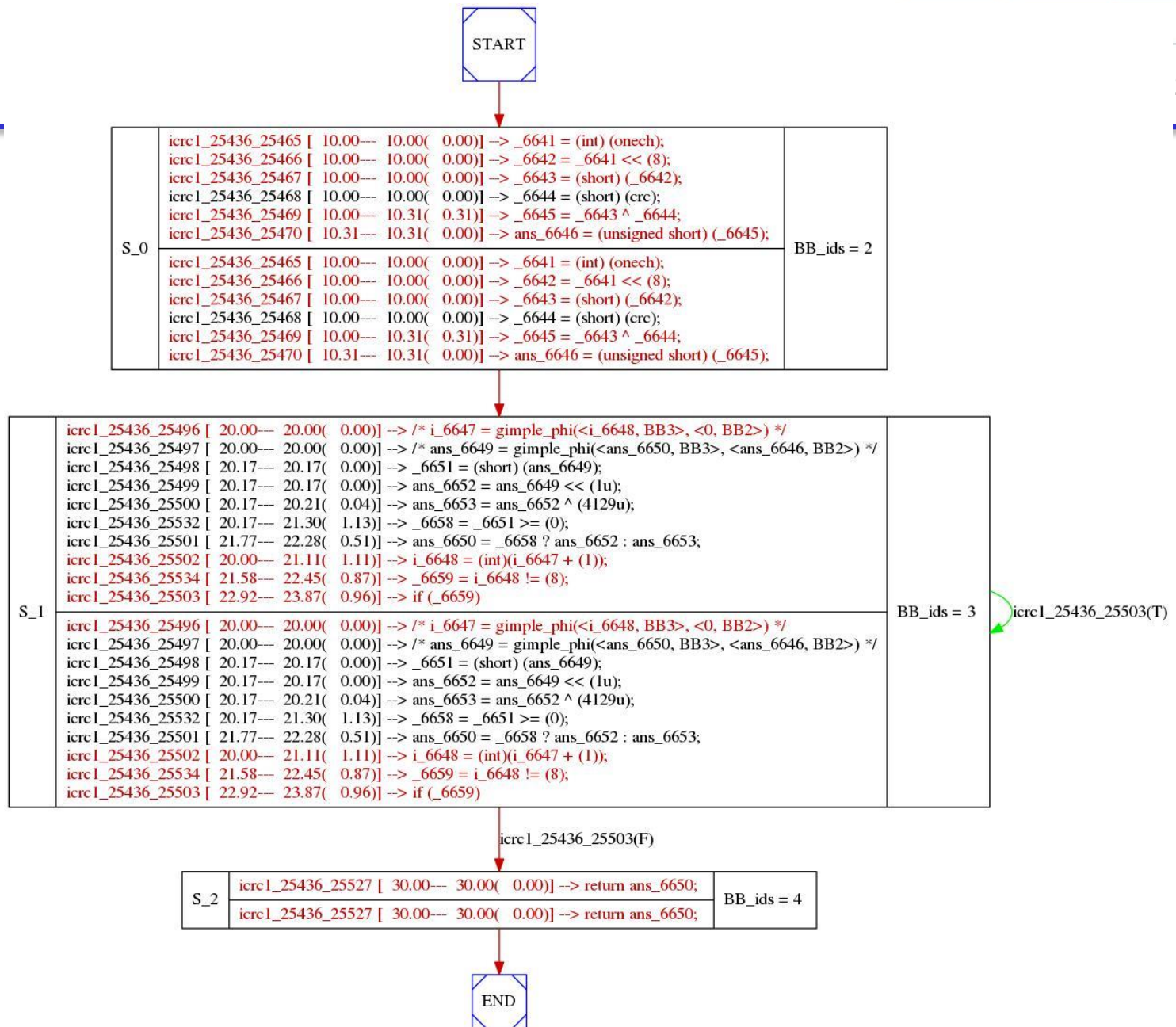
- ❑ Verilog file generated at the end of the HLS step:
 - ▶ icrc1/icrc1.v

- ❑ Change directory to icrc1:

```
cd icrc1
```

- ❑ Display the FSM:

```
xdot HLS_output/dot/icrc1/HLS_STGraph.dot
```



```
module icrc1_minimal_interface(clock, reset, start_port, crc, onech,
done_port, return_port);
    // IN
    input clock;
    input reset;
    input start_port;
    input [15:0] crc;
    input [7:0] onech;
    // OUT
    output done_port;
    output [15:0] return_port;
    // Component and signal declarations

    icrc1 icrc1_i0 (.done_port(done_port), .return_port(return_port),
.clock(clock), .reset(reset), .start_port(start_port), .crc(crc),
.onech(onech));

endmodule
```


- ❑ HLS tool developed at Politecnico di Milano (Italy) within the PandA framework
 - ▶ Available under GPL v3 at
 - <http://panda.dei.polimi.it/>
 - <https://github.com/ferrandi/PandA-bambu>
- ❑ Example features
 - ▶ Front-end Input: interfacing with GCC for parsing C code
 - Complete support for ANSI C (except for recursion)
 - Support for pointers, user-defined data types, built-in C functions, etc..
 - Source code optimizations
 - may alias analysis, dead-code elimination, hoisting, loop optimizations, etc...
 - ▶ Target-aware synthesis
 - Characterization of the technology library based on target device
 - ▶ Verification
 - Integrated testbench generation and simulation
 - automated interaction with Iverilog, Verilator, Xilinx Isim, Xilinx Xsim, Mentor Modelsim
 - ▶ Back-end: Automated interaction with commercial synthesis tools
 - FPGA: Xilinx ISE, Xilinx Vivado, Altera Quartus, Lattice Diamond
 - ASIC: Synopsis Design Compiler

- ❑ Panda framework development started on 2004 as a support research infrastructure for PoliMi in the context of ICODES – FP6-IST EU-funded project
 - ▶ Parsing and analysis of TLM 2.0 SystemC descriptions (gcc v.3.5)
- ❑ In the hArtes EU-funded project (2006-2010), it was used to
 - ▶ Analyzing generic C-based application annotated with pragmas (OpenMP)
 - ▶ Extracting parallel tasks
 - ▶ Estimating performance of embedded app
 - ▶ C-to-c rewriting
- ❑ Later, in Synaptic (2009-2013) and in Faster (2011-2014) EU-funded projects, logic- and high-level synthesis has been extended
 - ▶ Bambu (HLS tool) was first released in March 2012.
- ❑ ESA funded many research on code predictability analysis, performance analysis, and integration of HLS in model-based design flows.

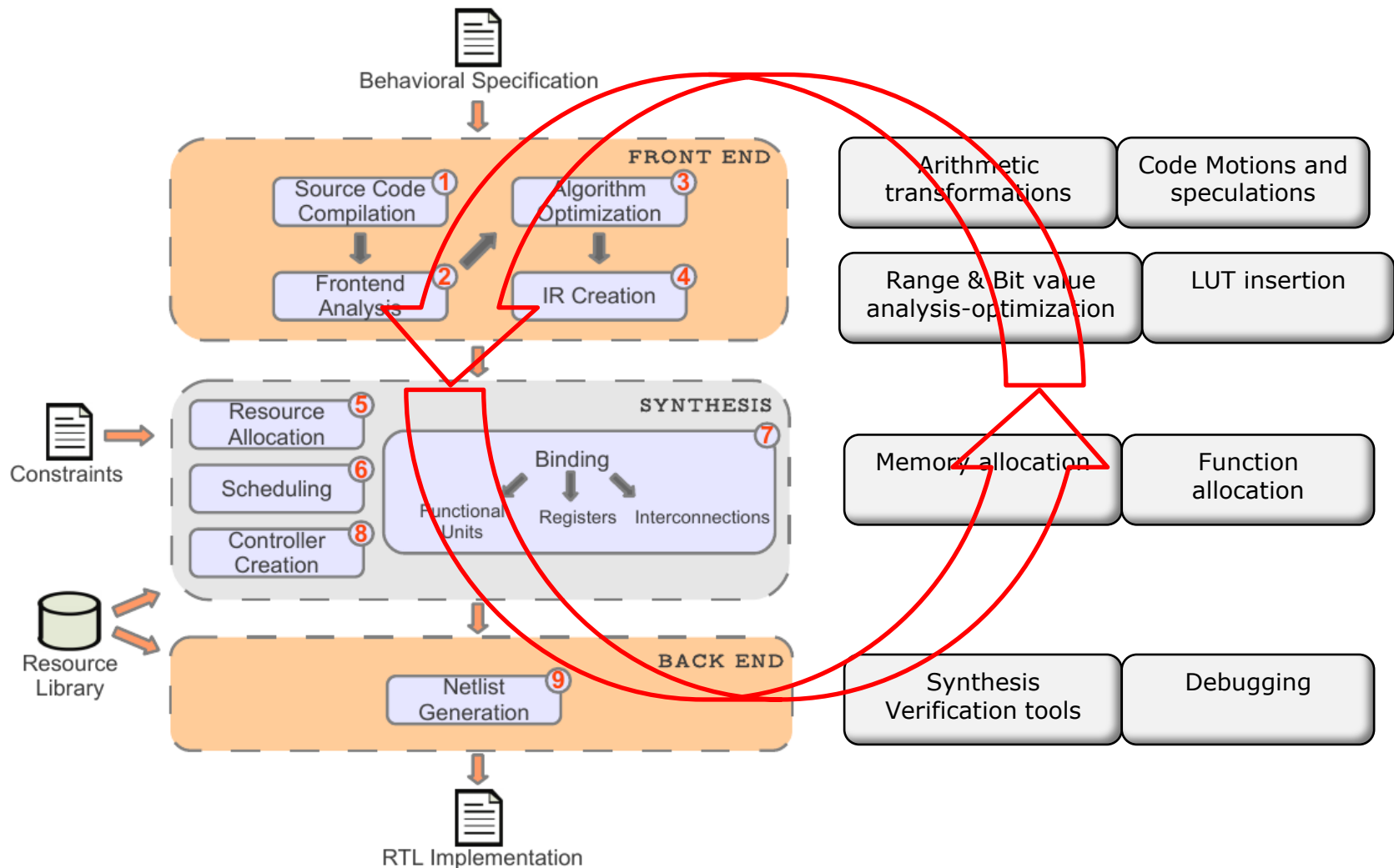
□ From LegUp 1.0:

Well, it turns out that none of the existing high-level synthesis tools have source code available for researchers. **GAUT** claims to be open-source but the code is not available for download. **xPilot** from UCLA is an advanced research tool but only the binary is available and it hasn't been updated since 2007. **ROCCC** provides an open source eclipse plugin based on SUIF and LLVM but only supports small C programs. Standard C code must be rewritten to work with ROCCC because all function parameters must be structs. **Trident** uses a very old version of LLVM to interface with an extensive amount of Java code, but unfortunately no longer compiles with the latest version of LLVM.

1. Are we classical computer hackers, skilled in SW development and with 100% control of our PCs and workstations?
 - ▶ Not too surprisingly, that is the perspective from which RMS wrote the GPL.
 - ▶ RMS defined **freedom** as the ability of people like himself to get read-write access to code.
 - ▶ How does this definition map to today's world?
 - ▶ How much transparent are today's systems?
 - HW vs SW
2. Funding of PandA comes mainly from public institutions
3. ... because it's fun

- ❑ Leon Stok, "Data path synthesis", Integration VLSI Journal, 18(1): 1-71 (1994).
- ❑ Jason Cong, Zhiru Zhang, "An efficient and versatile scheduling algorithm based on SDC formulation", DAC 2006, 433-438.
- ❑ Florent de Dinechin, Bogdan Pasca: "Designing Custom Arithmetic Data Paths with FloPoCo". IEEE Design & Test of Computers 28(4), 18-27 (2011).
- ❑ Ron Cytron, Jeanne Ferrante, Barry K. Rosen, Mark N. Wegman, F. Kenneth Zadeck, "Efficiently Computing Static Single Assignment Form and the Control Dependence Graph", ACM Trans. Program. Lang. Syst. 13(4), 451-490 (1991).
- ❑ Preston Briggs, Keith D. Cooper, Timothy J. Harvey, L. Taylor Simpson, "Practical Improvements to the Construction and Destruction of Static Single Assignment Form", Softw., Pract. Exper. 28(8), 859-881 (1998).
- ❑ Benoit Boissinot, Florian Brandner, Alain Darte, Benoît Dupont de Dinechin, Fabrice Rastello, "A Non-iterative Data-Flow Algorithm for Computing Liveness Sets in Strict SSA Programs". APLAS 2011, 137-154.

- ❑ one component per function
 - ▶ function interface
 - ▶ start and done
 - ▶ parameter passing
 - wires
 - memory interaction
- ❑ hierarchy based on call graph
 - ▶ no-recursion
 - ▶ proxy
- ❑ option
 - ▶ `-fwhole-program`



Modular Framework based on the specialization of HLS_step

❑ Minimal command

- ▶ `$ bambu filename.c`

❑ Controlling the clock period (100Mhz)

- ▶ `$ bambu filename.c --clock-period=10`

❑ Select the device

- ▶ `$ bambu filename.c -device-name=xc7z020,-1,clg484,VVD`

- ❑ We support the standard GCC support
- ❑ Supported features:
 - ▶ Expressions of any kind: arithmetic, logical, bitwise, relational, conditional, comma-based expressions.
 - ▶ Types: integers, single- and double-precision floating point, `_Bool` and `Complex`, struct-or-union, bitfields, enum, typedef, pointers and arrays, type qualifiers.
 - ▶ Variable declarations, initialization, storage-specifiers
 - ▶ Functions definition and declaration, extern or static, pointer to functions, parameters passed by copy or reference, tail recursive functions.
 - ▶ Statements and blocks: labeled (`case`), compound, expression, selection (`if`, `switch`), iteration (`while`, `do`, `for`), jump (`goto`, `continue`, `break`, `return`)
 - ▶ All preprocessor directives
- ▶ Unaligned memory accesses and dynamic pointers resolution
- ▶ GCC vectorization

Subset not supported

34

- ❑ struct returned by copy
- ❑ Non-tailing recursive functions

- ❑ Search and insertion in a binary tree
- ❑ Change the directory by typing:

```
cd ~/PandA-bambu/documentation/tutorial_fpl_2017/intro/second
```

- ❑ Edit the file C by typing:

```
gedit module.c
```

- ▶ Two data structures: stack and binary tree
- ▶ Static memory allocators
- ▶ Tail recursive functions
- ▶ Use of pointer to pointers (some HLSs have problems)

- ❑ `assert, puts, putchar, read, open, close, write, printf, exit, abort`
- ❑ **libc functions:** `bswap32, memcmp, memcpy, memmove, memset, malloc, free, memalign, alloca_with_align, calloc, bcopy, bzero, memchr, memcpy, memchr, rawmemchr, strcpy, strncpy, strcasecmp, strcasestr, strcat, strchr, strchrnul, strcmp, strcpy, strcspn, strdup, strlen, strncasecmp, strncat, strncmp, strncpy, strndup, strnlen, strpbrk, strchr, strsep, strspn, strstr, strtok`
- ❑ **libm functions:** `acos, acosh, asin, asinh, atan, atan2, atanh, cbrt, ceil, cexp, copysign, cos, cosh, drem, erf, exp, exp10, expm1, fabs, fdim, finite, floor, lfloor, fma, fmax, fmin, fmod, fpclassify, frexp, gamma, lgamma, tgamma, hypot, ilogb, infinity, isinf, isnan, j0, j1, jn, ldexp, log, log2, log10, log1p, modf, nan, nearbyint, nextafter, pow, pow10, remainder, remquo, rint, lrint, llrint, round, lround, llround, scalb, scalbln, scalbn, signbit, significand, sin, sincos, sinh, sqrt, tan, tanh, trunc.`

- ❑ Crypto core built composing user defined libraries
- ❑ Change the directory by typing:

```
cd ~/PandA-bambu/documentation/tutorial_fpl_2017/intro/third
```

- ❑ Run the script by typing:

```
./multi.sh
```

- ❑ `tree-panda-gcc` could be used to create a custom library that could be deployed by `bambu` by passing `-l` and `-L` options.

Autotools based project described in directory
[examples/crypto_designs/multi-keccak](#)

- ❑ `bambu` uses the text-based IR and the builtin linker to build a single in-memory representation and perform HLS

▶ `$ bambu file1.c file2.c --top-fname=fname`

- ❑ In case the option `--top-fname` is not used, `bambu` automatically identifies which function can act as the top one.

- ❑ Based on C description
 - ▶ Global variables
 - ▶ Top function parameters
- ❑ No restriction on top function signature
- ❑ C semantic
 - ▶ Scalar parameters: input
 - ▶ struct/union parameters: input
 - ▶ array/pointer: input/output
 - ▶ return value: output
 - ▶ Global variables: input/output

```
int global;  
  
float my_top_function(int * first, struct s second, int  
third[10], float * fourth)  
{...}
```

❑ Input parameters:

- ▶ Second

❑ Input/output parameters:

- ▶ First, third, fourth

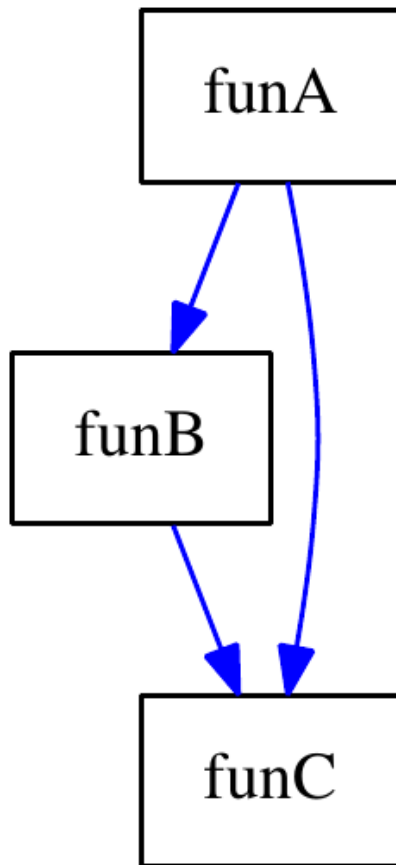
❑ Output parameters:

- ▶ Return value

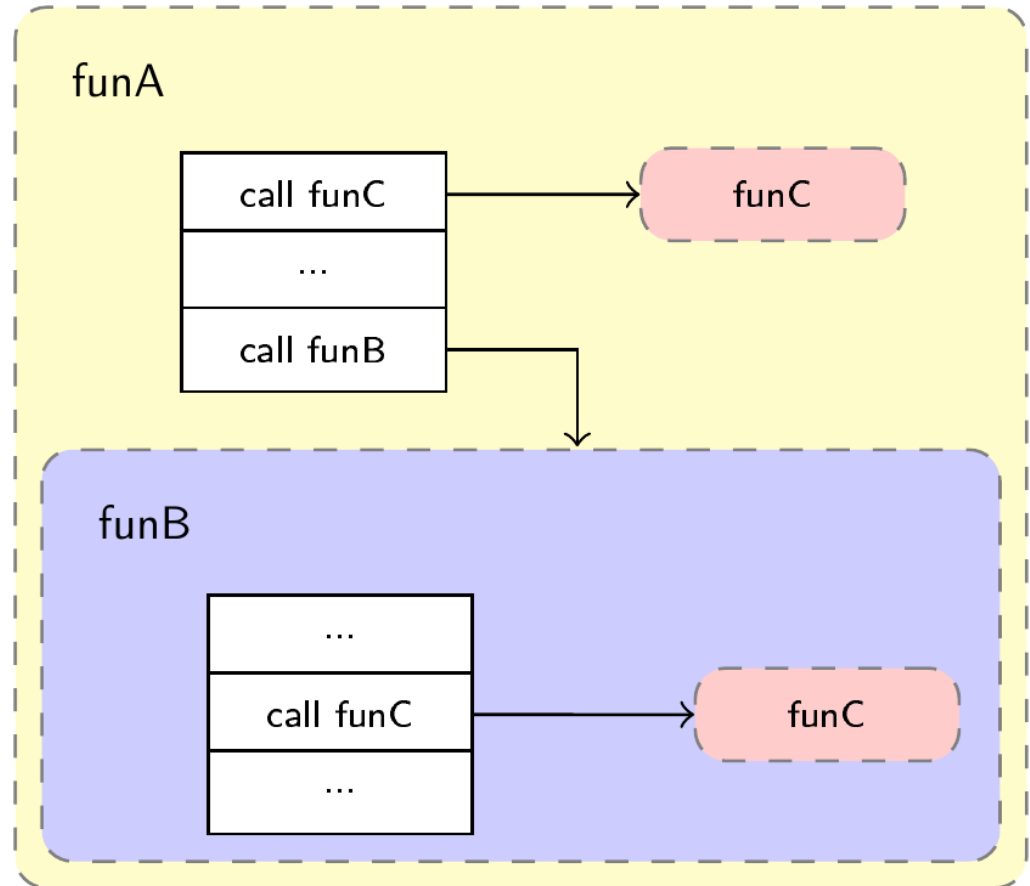
❑ Shared memory:

- ▶ global

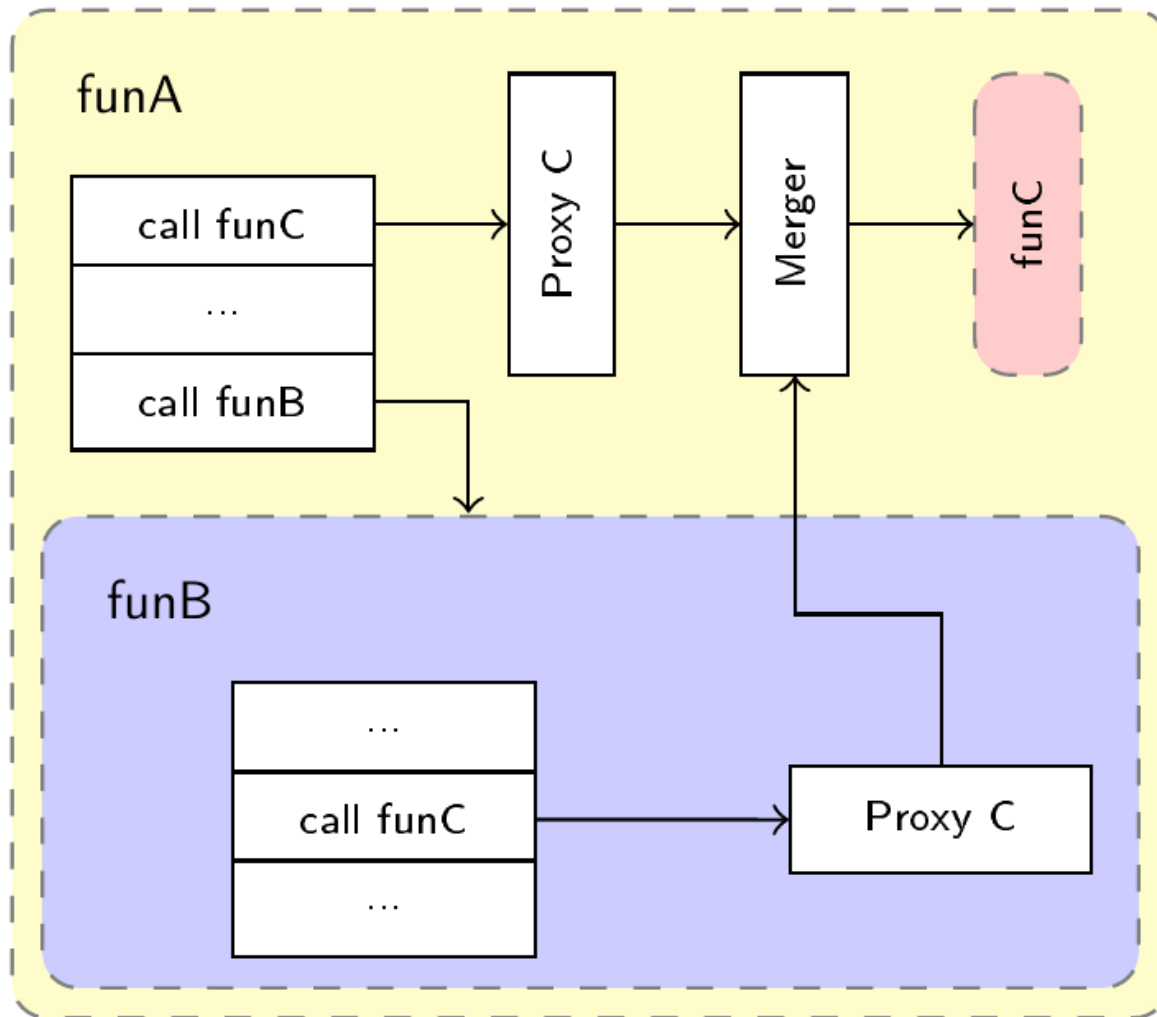
- ❑ One component per function



call graph



RTL hierarchy

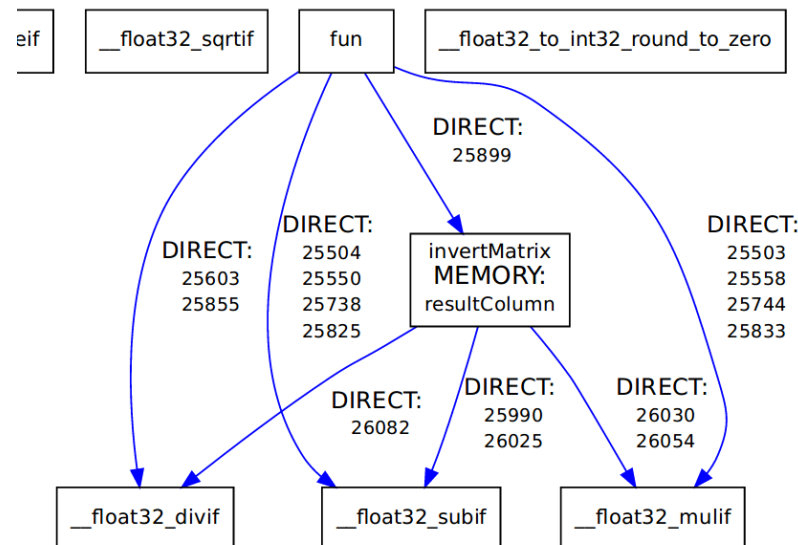


❑ Lu-decomposition with single precision floating point arithmetic

```
cd ~/PandA-bambu/documentation/tutorial_fpl_2017/intro/fourth
```

❑ Run the script by typing:

```
./bambu.sh
```



- ❑ bambu assumes that all global variables could be accessed by external CPU/accelerators
- ❑ It is possible to change this default with option:
 - ▶ `--do-not-expose-globals`
 - ▶ All global variables are considered local to the compilation units as if they are declared `static`.

❑ From GCC man:

- ▶ Assume that the current compilation unit represents the whole program being compiled. All public functions and variables with the exception of "main" and those merged by attribute "externally_visible" become static functions and in effect are optimized more aggressively by interprocedural optimizers.

❑ You can apply this optimization when the top function is `main` or when you specify an alternative top function (`--top-fname`) provided that the option `--do-not-expose-globals` is also given (currently supported only with GCC 4.9)

❑ Integration of existing IPs written in Verilog that receives structs passed by pointers

```
cd ~/PandA-bambu/documentation/tutorial_fpl_2017/intro/fifth
```

```
#include "module_lib.h"

void my_ip(uint8_t command, uint32_t param1, uint32_t param2) {
    static module1_output_t module1_output;
    static module2_output_t module2_output;
    switch(command) {
        case 0:
            module1(param1, param2 >> 16, &module1_output);
            break;
        case 1:
            module2(param1, &module2_output);
            break;
        case 2:
            printer1(module1_output.output1, module1_output.output2, module1_output.output3,
module1_output.output4);
            break;
        case 3:
            printer2(module2_output.output1, module2_output.output2, module2_output.output3);
            break;
        default:
            break;
    }
}
```

- ❑ Function mapped on IPs has to be declared as extern:
 - ▶ `extern void module1(uint32_t input1, uint16_t input2, module1_output_t *outputs);`
- ❑ C code has to be passed with the following option
 - ▶ `--C-no-parse=module1.c,...`
- ❑ Binding between function **module1** and component **module1** has to be specified with a XML file and passed as an option to bambu
 - ▶ `$ bambu ... module_lib.xml`
- ❑ Check these examples:
 - ▶ `examples/IP_integration`
 - ▶ `examples/breakout`
 - ▶ `examples/pong`
 - ▶ `examples/led_example`

```
int laplacian(char *, char *, int, int);
int make_inverse_image(char *, char *, int, int);
int sharpen(char *, char *, int, int);
int sobel(char *, char *, int, int);

int (*pipeline[MAX_DEPTH])(char *, char *, int, int);

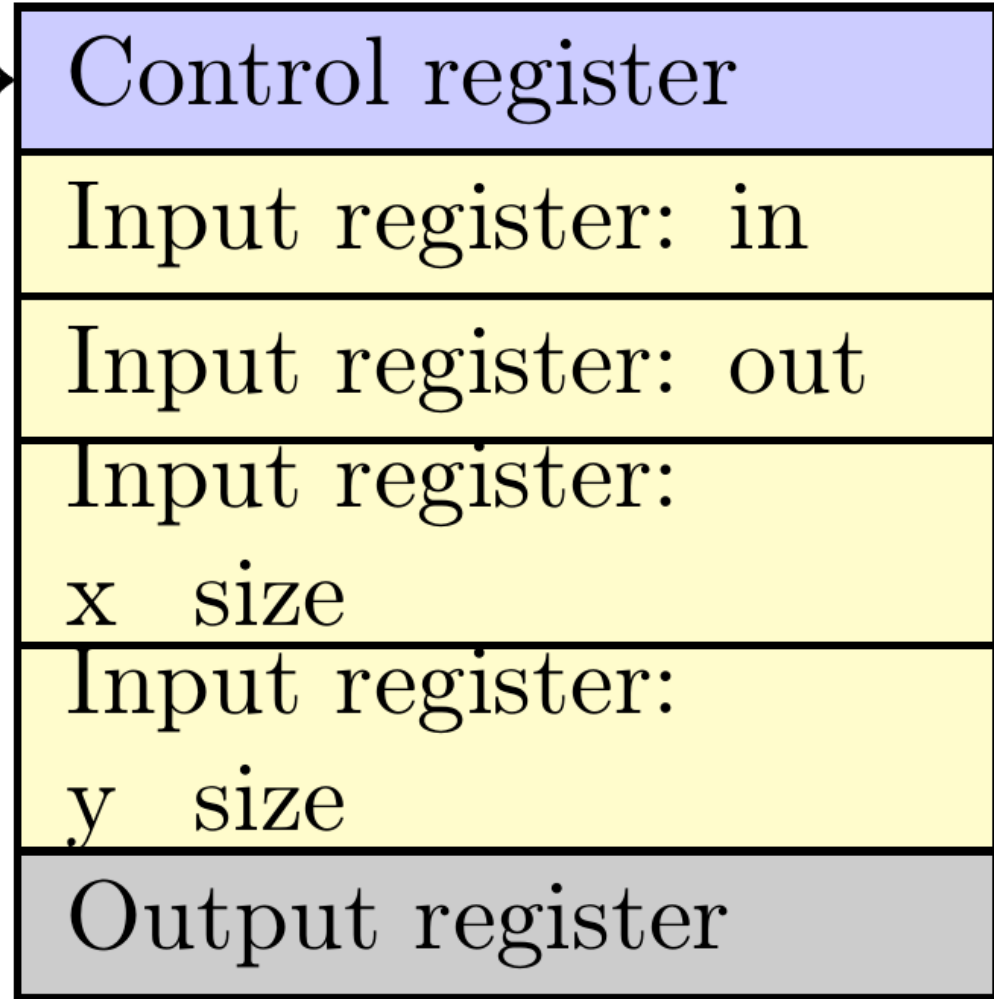
void UserApp(char *in, char *out, int x_size, int y_size) {
    // ...
    // Pipeline configuration using function pointers
    add_filter(0, make_inverse_image);
    add_filter(1, sharpen);
    // ...
    // execute is synthesized in hardware
    execute(in, out, x_size, y_size);
}

void execute(char *in, char *out, int x_size, int y_size) {
    int i = 0;
    for (i = 0; i < MAX_PIPELINE_DEPTH; i++) {
        if (pipeline[i] == 0) break;
        // here other hw accelerator are called
        // using function pointers
        int res = pipeline[i](in, out, x_size, y_size);
        if (res != 0) return;
        swap(in, out);
    }
    move_if_odd(i, in, out);
}
```


Adding a memory mapped interface to filters

49

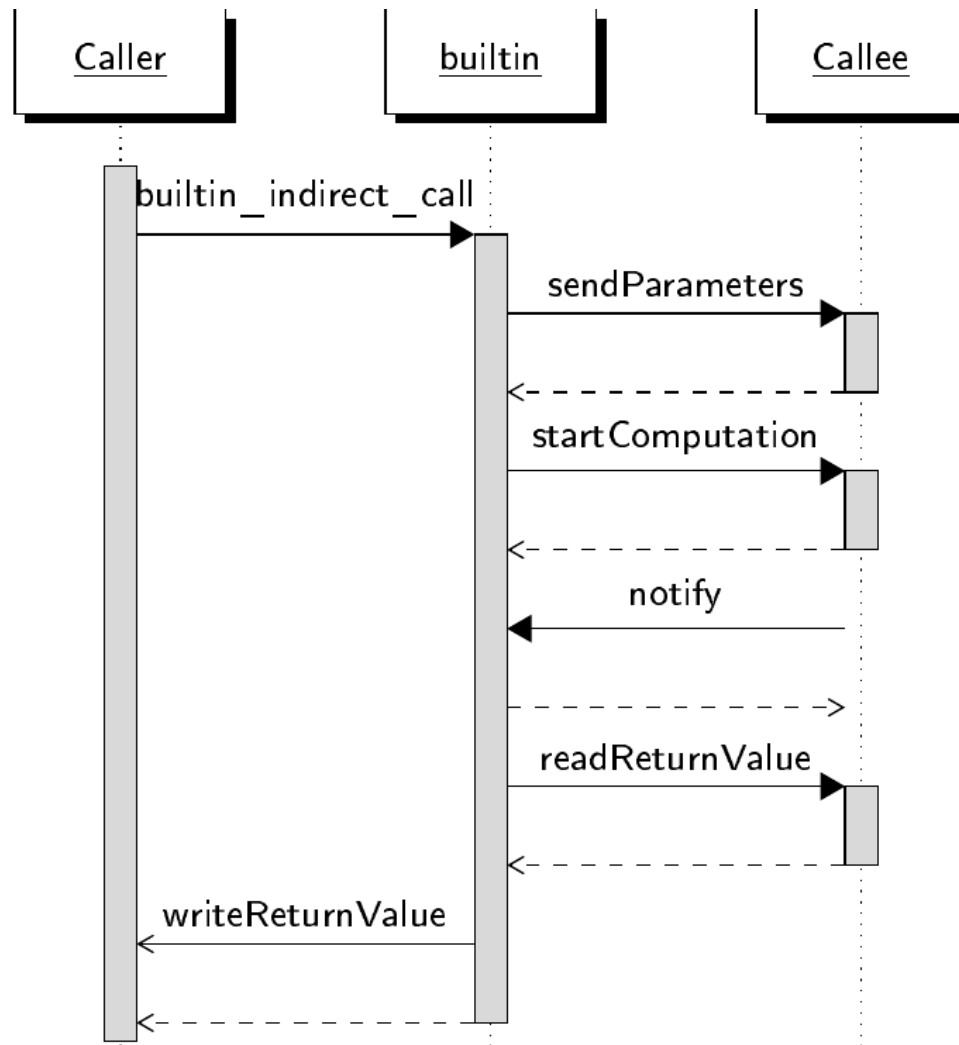
Accelerator
base address



```
void execute(char *in, char *out, int x_size, int y_size) {  
    int i = 0;  
    for (i = 0; i < MAX_PIPELINE_DEPTH; i++) {  
        if (pipeline[i] == 0) break;  
        // here other hw accelerator are called  
        // using function pointers  
        __builtin_indirect_call(  
            pipeline[i], 1, in, out, x_size, y_size, &res);  
        if (res != 0) return;  
        swap(in, out);  
    }  
    move_if_odd(i, in, out);  
}
```

Sequence diagram for function indirect call

51



Call mechanism complexity: $\#cycles = Wl(Np+1) + lhs(Wl+RI)$

- ❑ Parametric quick sort

```
cd ~/PandA-bambu/documentation/tutorial_fpl_2017/intro/sixth
```

- ❑ Quick sort parametric with respect the comparison function
- ❑ Run the script and check the results

- ❑ Support of c++ is ongoing:
 - ▶ templates
 - ▶ C++11 and beyond
 - ▶ ac_types from Mentor Graphics could be used
 - ac_int and ac_fixed currently cannot be used in the function interfaces
 - not c++11 friendly

```
#include <algorithm>
int gcd(int x, int y )
{
    if( x < y )
        std::swap( x, y );

    while( y > 0 )
    {
        int f = x % y;
        x = y;
        y = f;
    }
    return x;
}
```

- * euclid.f (FORTRAN 77)
- * Find greatest common divisor using the Euclidean algorithm

```
FUNCTION NGCD (NA, NB)
  IA = NA
  IB = NB
1  IF (IB.NE.0) THEN
    ITEMP = IA
    IA = IB
    IB = MOD (ITEMP, IB)
    GOTO 1
  END IF
  NGCD = IA
  RETURN
END
```

By default parameters are passed by reference

- * euclid.f (FORTRAN 77)
- * Find greatest common divisor using the Euclidean algorithm

```
FUNCTION NGCD(NA, NB)
  integer, value :: NA ! input
  integer, value :: NB ! input
  IA = NA
  IB = NB
1  IF (IB.NE.0) THEN
    ITEMP = IA
    IA = IB
    IB = MOD(ITEMP, IB)
    GOTO 1
  END IF
  NGCD = IA
  RETURN
END
```

Here parameters are passed
by copy

❑ Fortran support in progress...

❑ #pragma omp simd is supported

```
#pragma omp declare simd
void add (int accelnum, int startidx, int endidx)
{
    int sum = 0;
    int i;
    for (i = 0; i < OPS_PER_ACCEL; i++) {
        sum += array[i+startidx];
    }
    output[accelnum] = sum;
}

main() {
    ...
    #pragma omp simd
    for (i = 0; i < NUM_ACCELS; i++)
    {
        add(i, i * OPS_PER_ACCEL, (i + 1)*OPS_PER_ACCEL);
    }
}
```

❑ OmpMP support in progress...