BondMachine

A Moldable Computer Architecture

Mirko Mariotti

Department of Physics and Geology - University of Perugia

February 23, 2018

Contents

Introduction

The BondMachine: a comprehensive approach to computing.

In this presentation i will talk about:

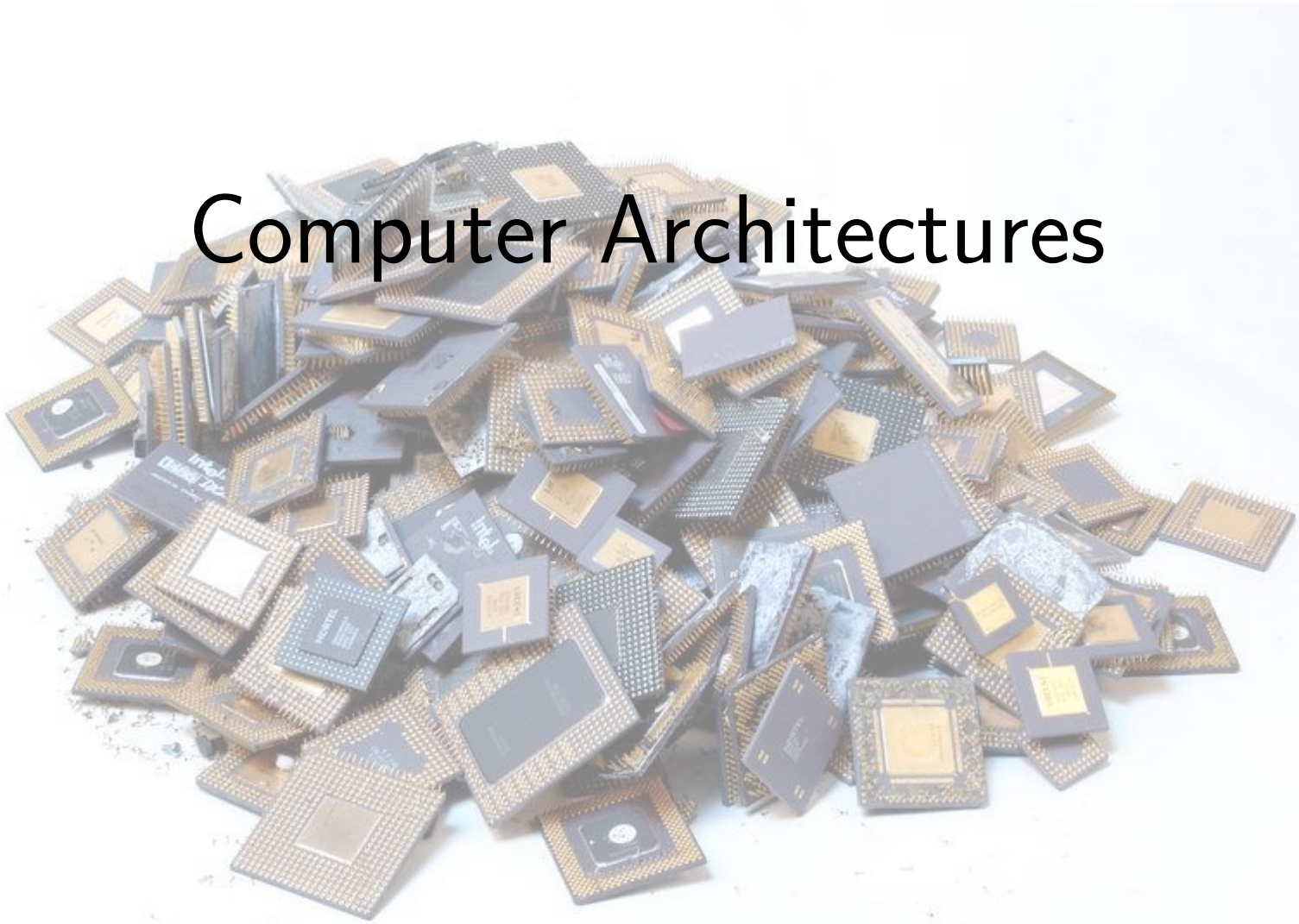
- Ideas that bring to the BondMachine.
- What is it.
- Developed software tools.
- Hardware implementation.
- Use cases.

Quick facts about the project

Started in May 2015 as a Verilog "garage" experiment, with the idea of creating a processor on an FPGA, so completely bottom-up.

A prototype in every aspects.

Computer Architectures



Computer Architectures

Base for the first idea

Today's computer architecture are:

- Multicore,Two or more independent actual processing units execute multiple instructions at the same time.
  - The power is given by the number.
  - Parallel algorithms.
- Heterogeneous, processing units of different type.
  - Cell, GPU, Parallela, TPU.
  - The power is given by the specialization.
  - Hard to make units communicate.
  - Hard to program.
  - Hard to schedule.

Computer Architectures

Base for the first idea

Today's computer architecture are:

- Multicore,Two or more independent actual processing units execute multiple instructions at the same time.
  - The power is given by the number.
  - Parallel algorithms.
- Heterogeneous, processing units of different type.
  - Cell, GPU, Parallela, TPU.
  - The power is given by the specialization.
  - Hard to make units communicate.
  - Hard to program.
  - Hard to schedule.

Computer Architectures

Base for the first idea

Today's computer architecture are:

- Multicore,Two or more independent actual processing units execute multiple instructions at the same time.
  - The power is given by the number.
  - Parallel algorithms.
- Heterogeneous, processing units of different type.
  - Cell, GPU, Parallela, TPU.
  - The power is given by the specialization.
  - Hard to make units communicate.
  - Hard to program.
  - Hard to schedule.

Computer Architectures

Base for the first idea

Today's computer architecture are:

- Multicore,Two or more independent actual processing units execute multiple instructions at the same time.
  - The power is given by the number.
  - Parallel algorithms.
- Heterogeneous, processing units of different type.
  - Cell, GPU, Parallela, TPU.
  - The power is given by the specialization.
  - Hard to make units communicate.
  - Hard to program.
  - Hard to schedule.

Computer Architectures

Base for the first idea

Today's computer architecture are:

- Multicore,Two or more independent actual processing units execute multiple instructions at the same time.
  - The power is given by the number.
  - Parallel algorithms.
- Heterogeneous, processing units of different type.
  - Cell, GPU, Parallela, TPU.
  - The power is given by the specialization.
  - Hard to make units communicate.
  - Hard to program.
  - Hard to schedule.

Computer Architectures

Base for the first idea

Today's computer architecture are:

- Multicore,Two or more independent actual processing units execute multiple instructions at the same time.
  - The power is given by the number.
  - Parallel algorithms.
- Heterogeneous, processing units of different type.
  - Cell, GPU, Parallela, TPU.
  - The power is given by the specialization.
  - Hard to make units communicate.
  - Hard to program.
  - Hard to schedule.

Computer Architectures

Base for the first idea

Today's computer architecture are:

- Multicore,Two or more independent actual processing units execute multiple instructions at the same time.
  - The power is given by the number.
  - Parallel algorithms.
- Heterogeneous, processing units of different type.
  - Cell, GPU, Parallela, TPU.
  - The power is given by the specialization.
  - Hard to make units communicate.
  - Hard to program.
  - Hard to schedule.

Computer Architectures

Base for the first idea

Today's computer architecture are:

- Multicore,Two or more independent actual processing units execute multiple instructions at the same time.
  - The power is given by the number.
  - Parallel algorithms.
- Heterogeneous, processing units of different type.
  - Cell, GPU, Parallela, TPU.
  - The power is given by the specialization.
  - Hard to make units communicate.
  - Hard to program.
  - Hard to schedule.

Computer Architectures

Base for the first idea

Today's computer architecture are:

- Multicore,Two or more independent actual processing units execute multiple instructions at the same time.
  - The power is given by the number.
  - Parallel algorithms.
- Heterogeneous, processing units of different type.
  - Cell, GPU, Parallela, TPU.
  - The power is given by the specialization.
  - Hard to make units communicate.
  - Hard to program.
  - Hard to schedule.

Computer Architectures

Base for the first idea

Today's computer architecture are:

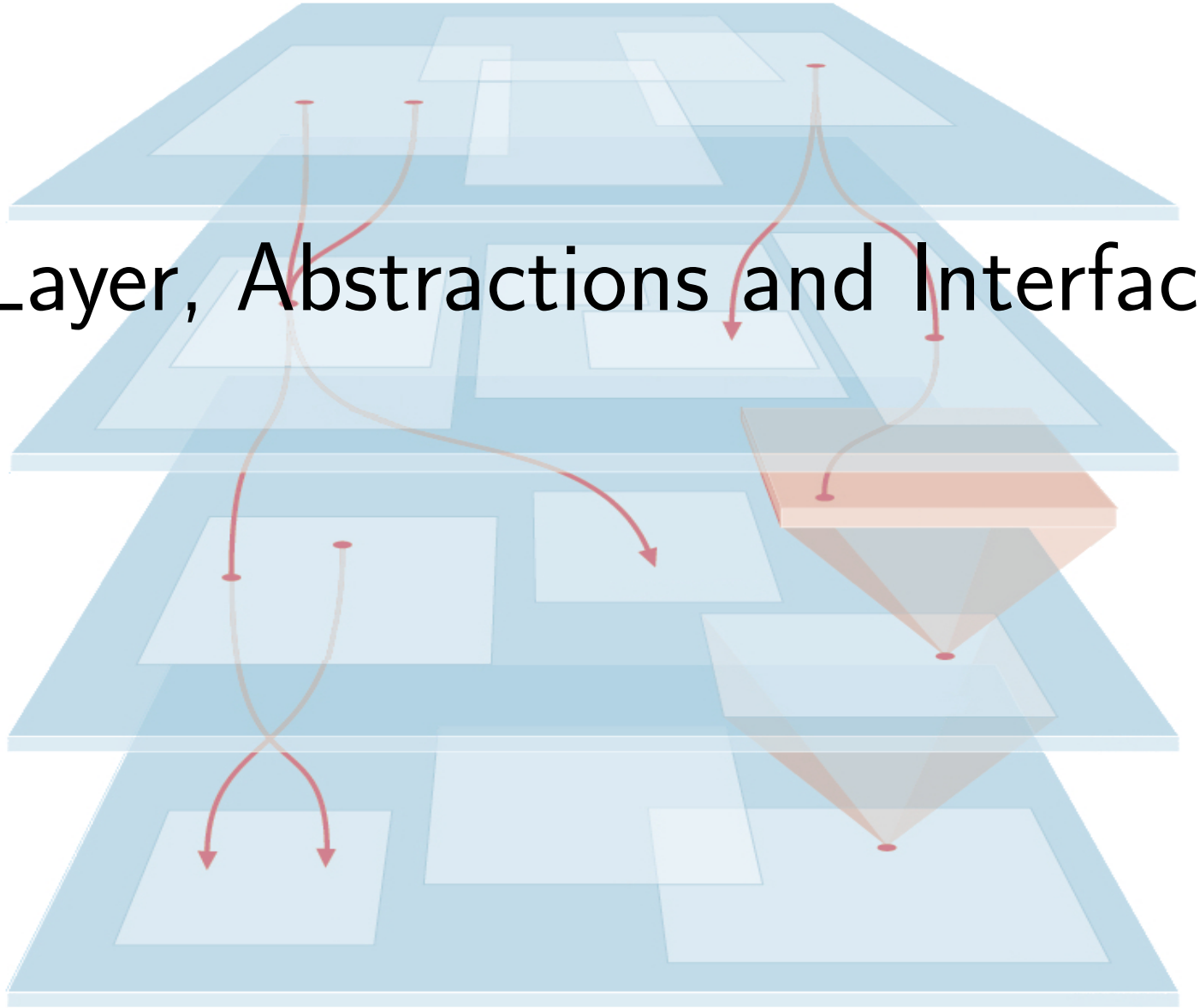
- Multicore,Two or more independent actual processing units execute multiple instructions at the same time.
  - The power is given by the number.
  - Parallel algorithms.
- Heterogeneous, processing units of different type.
  - Cell, GPU, Parallela, TPU.
  - The power is given by the specialization.
  - Hard to make units communicate.
  - Hard to program.
  - Hard to schedule.

Computer Architectures

First idea

Having a multi-core architecture completely heterogeneous both in cores types and interconnections.

Layer, Abstractions and Interfaces



Layer, Abstractions and Interfaces

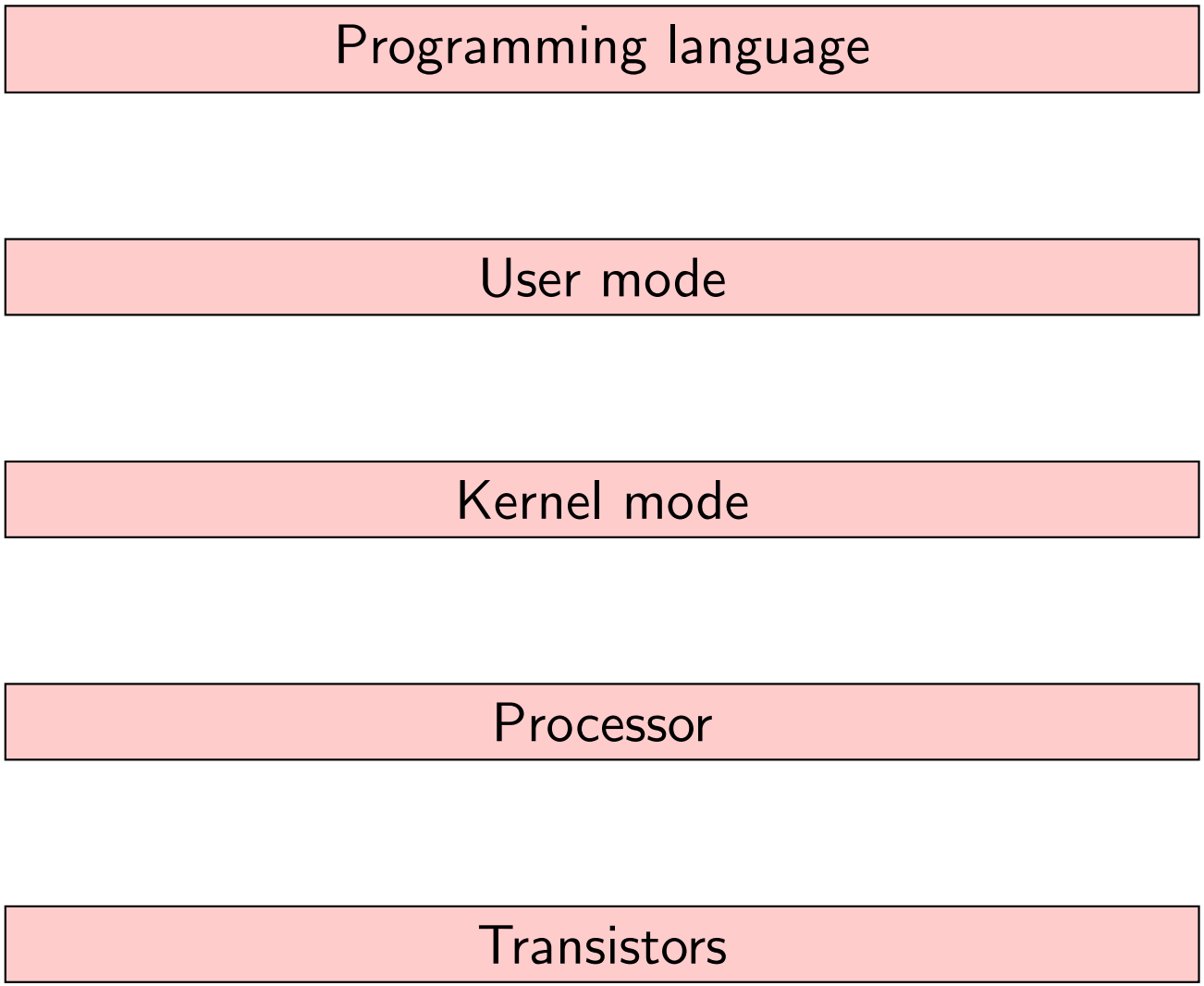
Introduction

A Computing system is a matter of abstraction and interfaces. A lower layer exposes its functionalities (via interfaces) to the above layer hiding (abstraction) its inner details.

The quality of a computing system is determined by how abstractions are simple and how interfaces are clean.

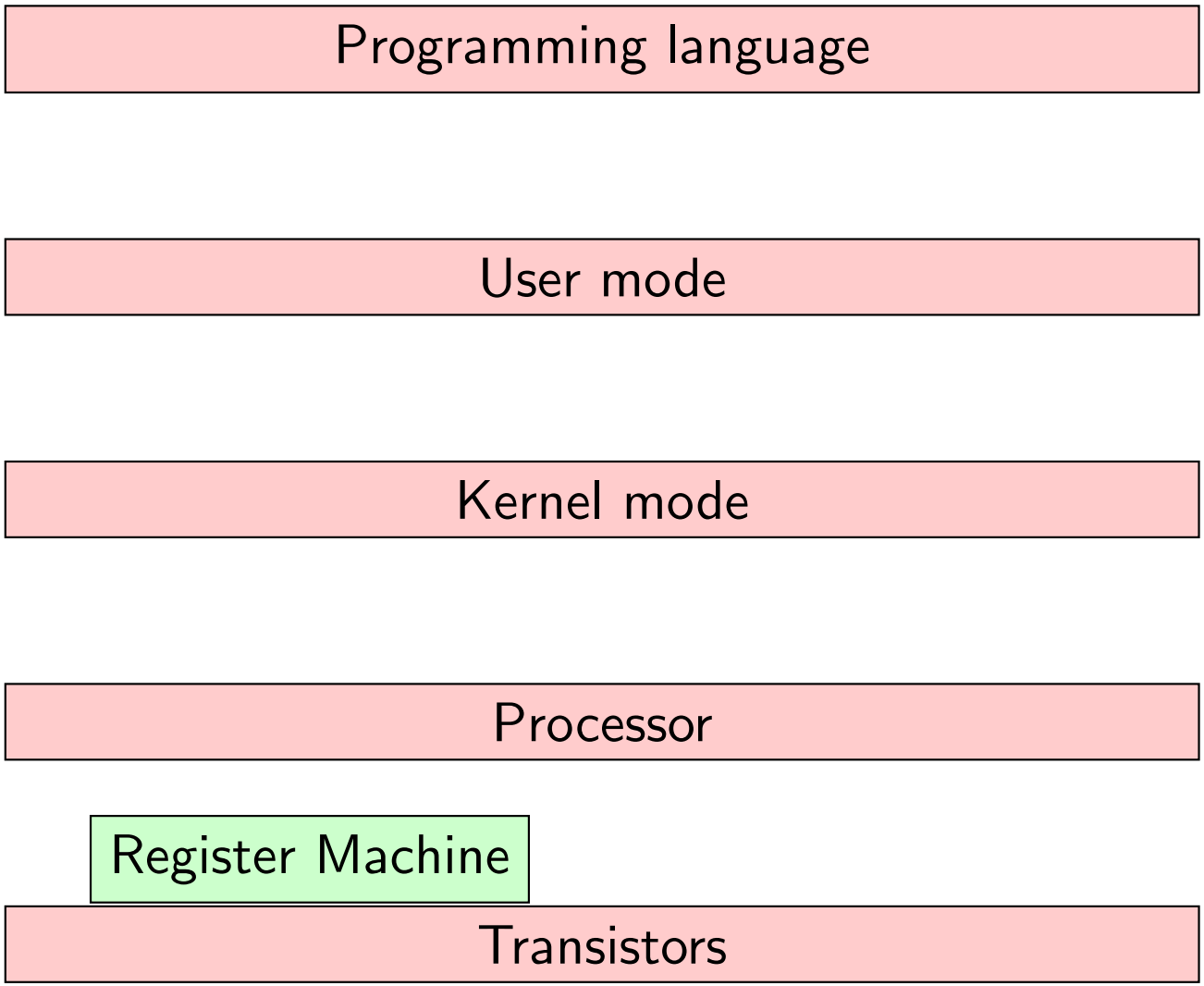
Layers, Abstractions and Interfaces

The base for the second idea



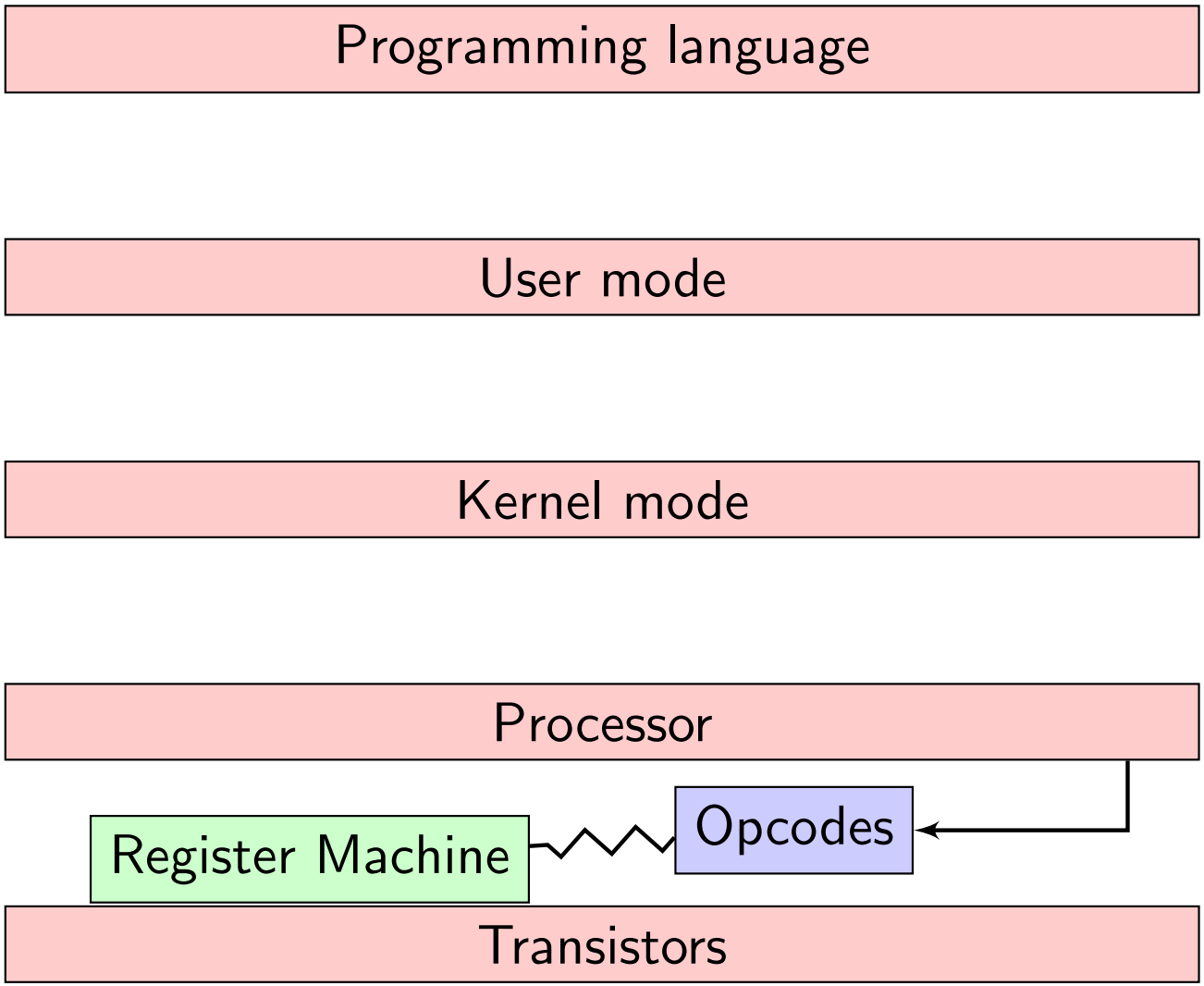
Layers, Abstractions and Interfaces

The base for the second idea



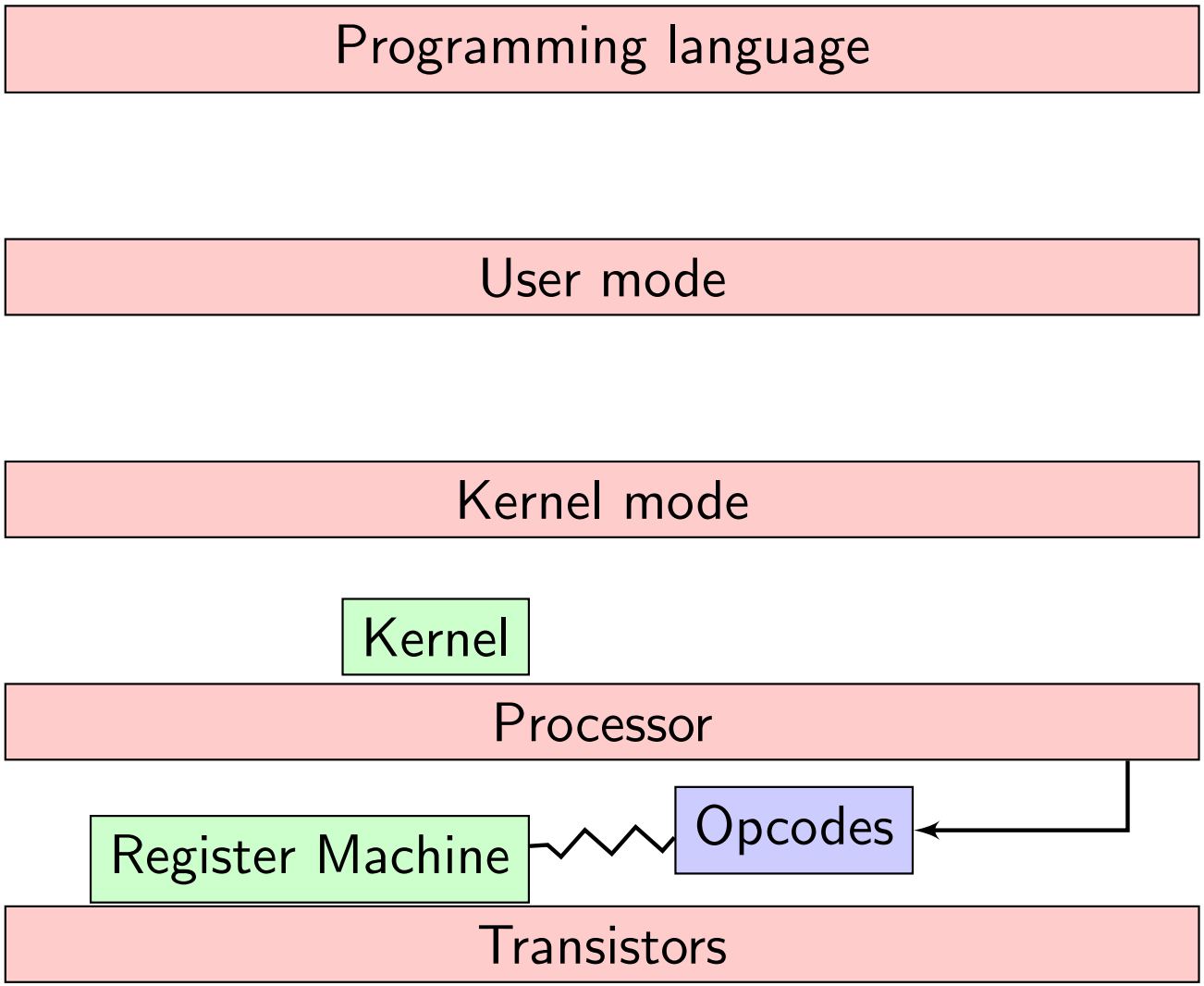
Layers, Abstractions and Interfaces

The base for the second idea



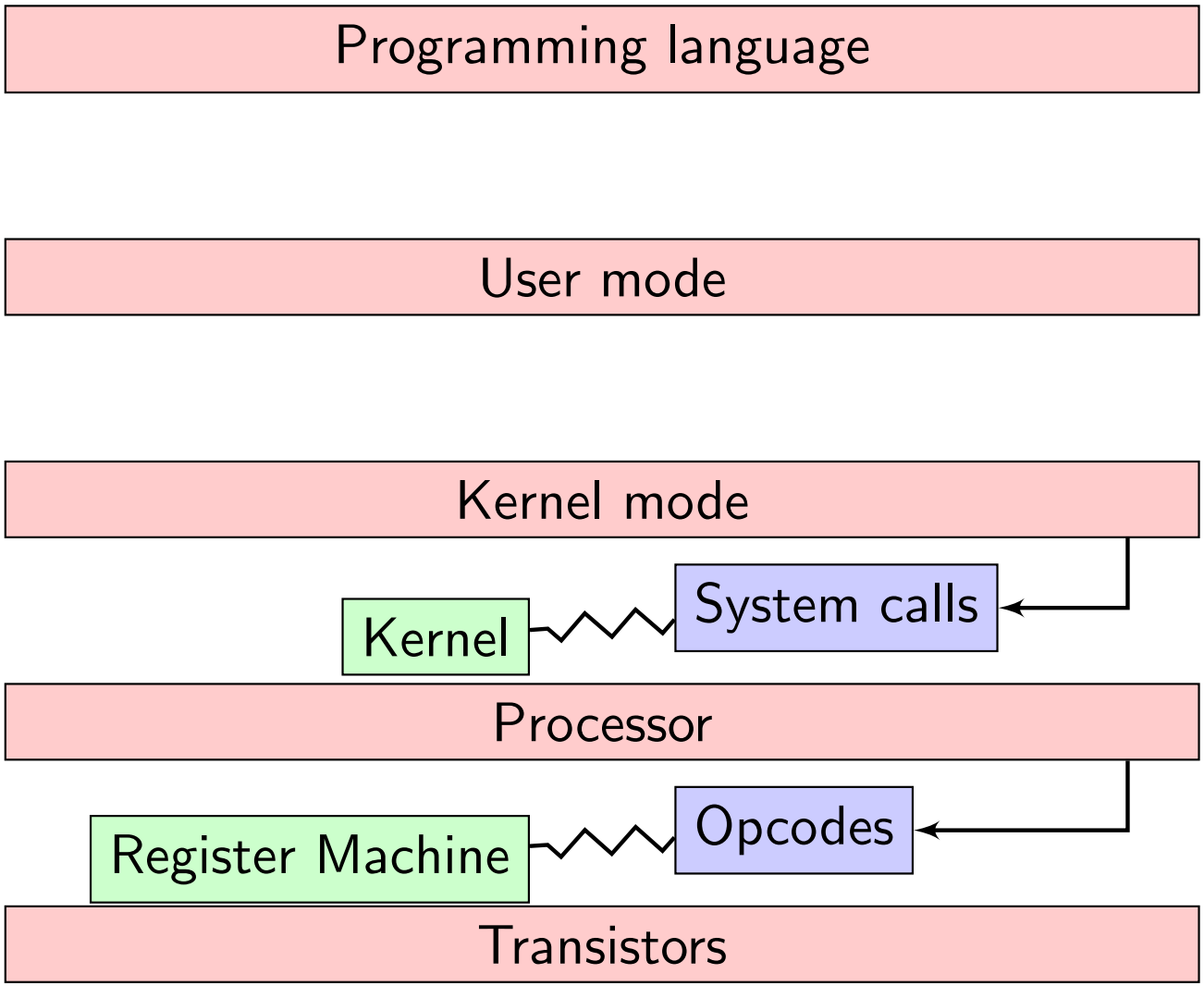
Layers, Abstractions and Interfaces

The base for the second idea



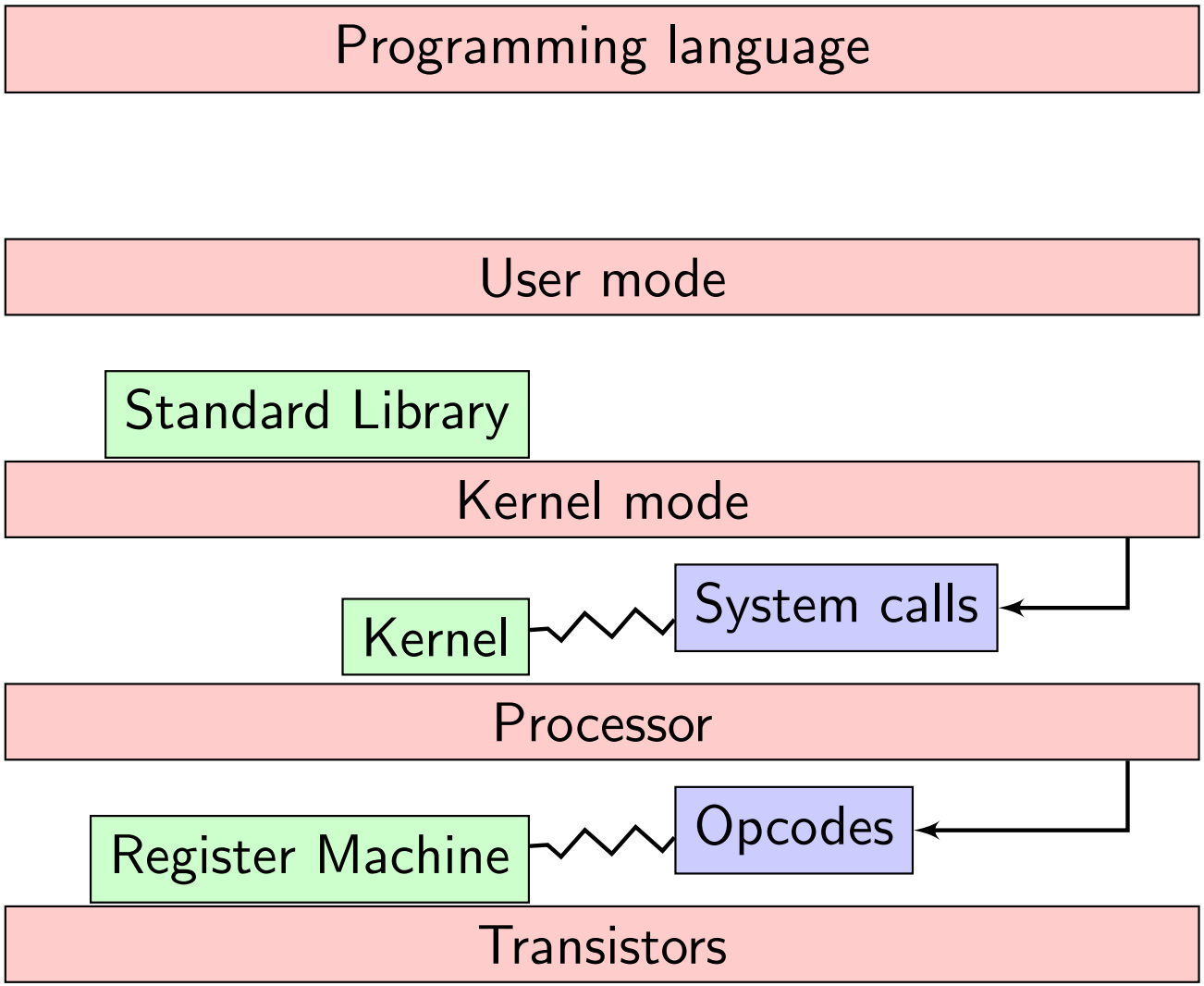
Layers, Abstractions and Interfaces

The base for the second idea



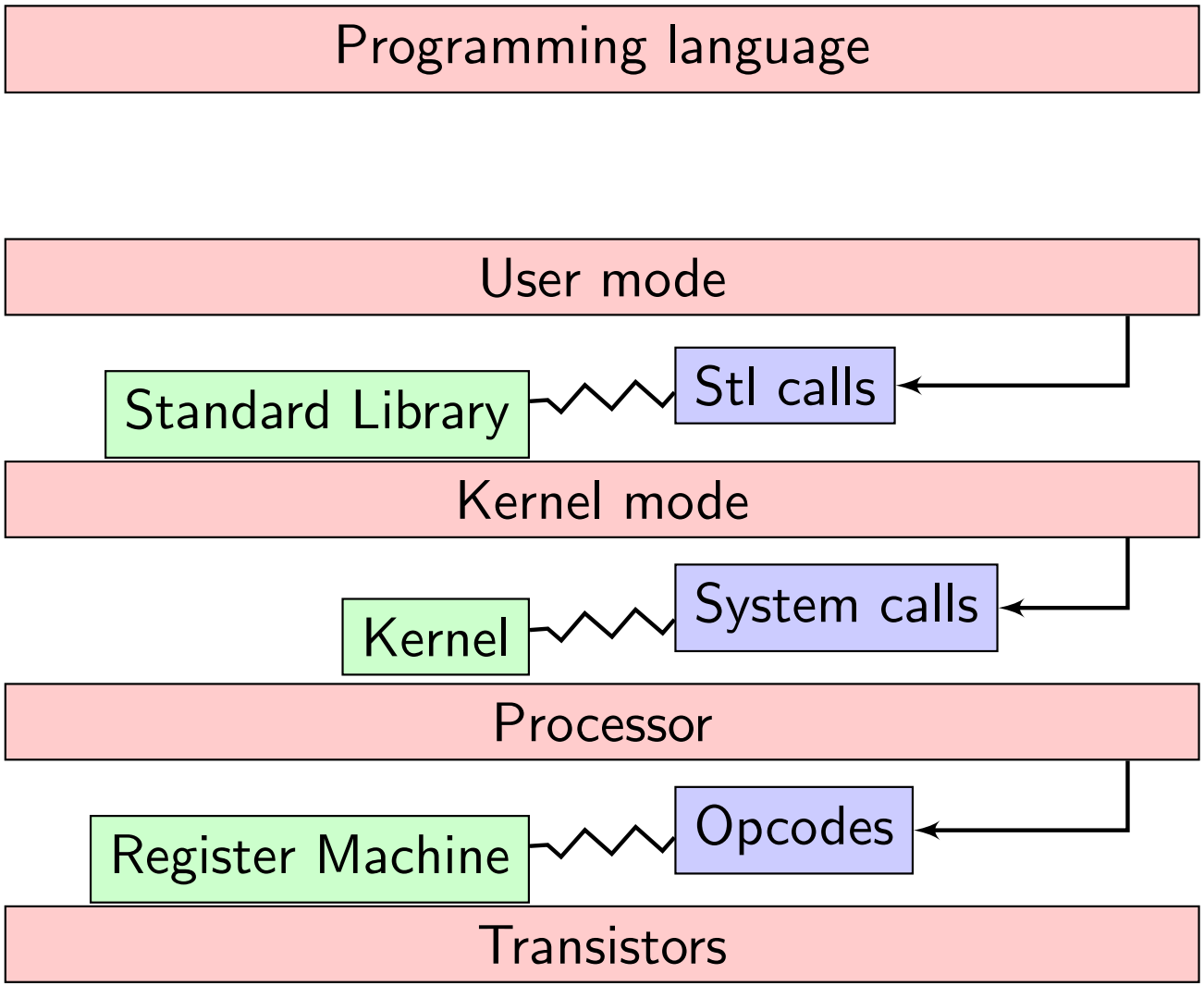
Layers, Abstractions and Interfaces

The base for the second idea



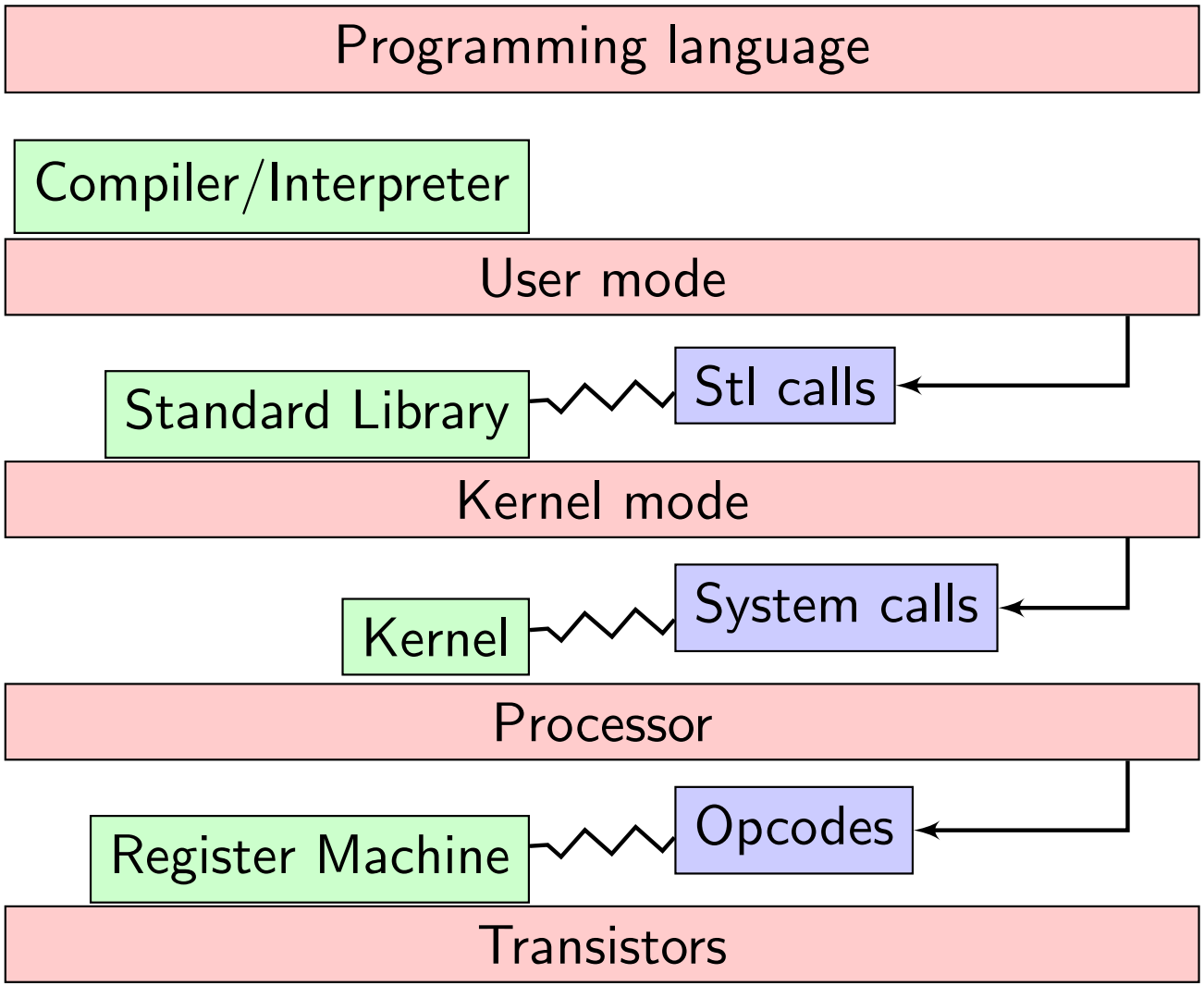
Layers, Abstractions and Interfaces

The base for the second idea



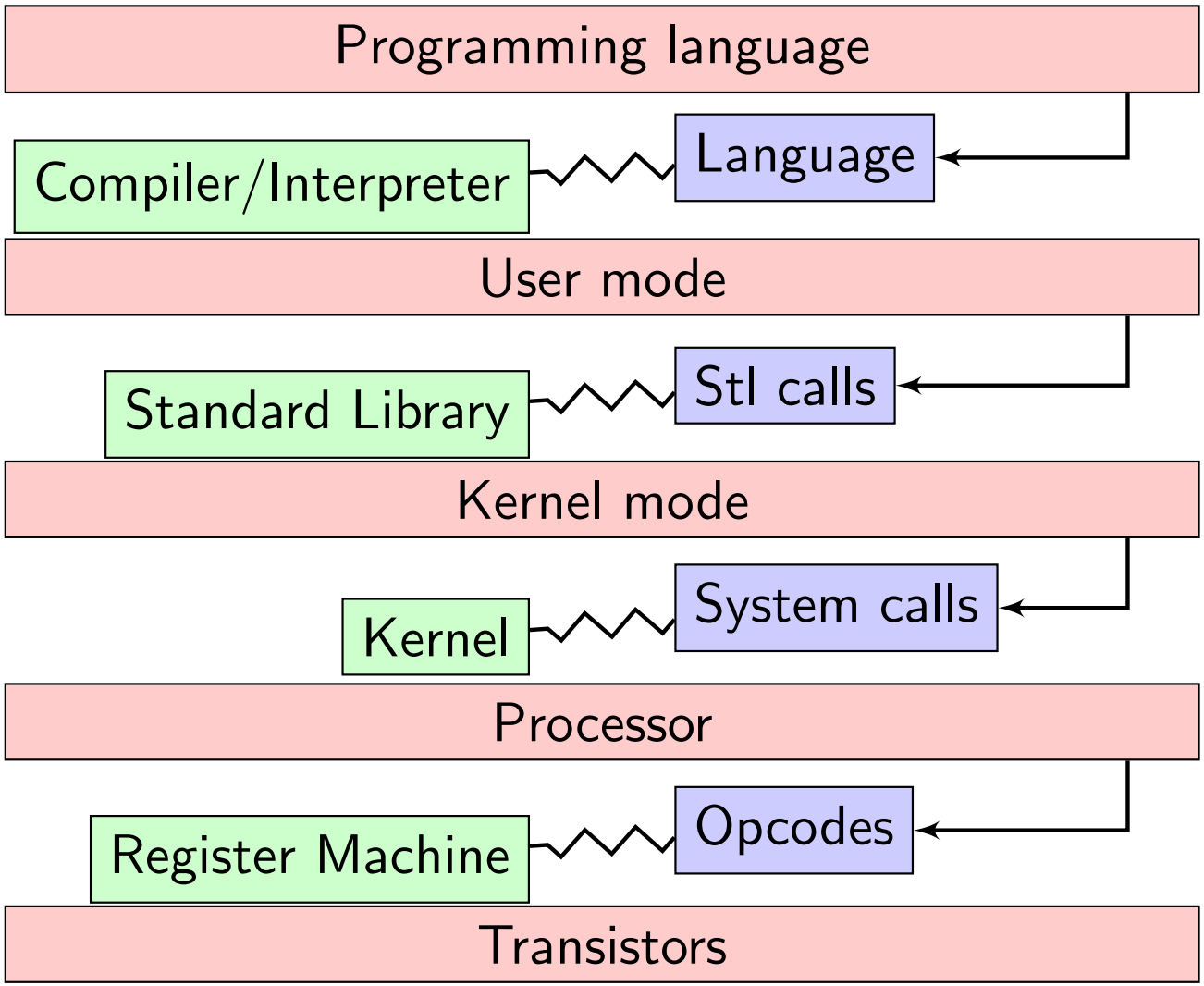
Layers, Abstractions and Interfaces

The base for the second idea



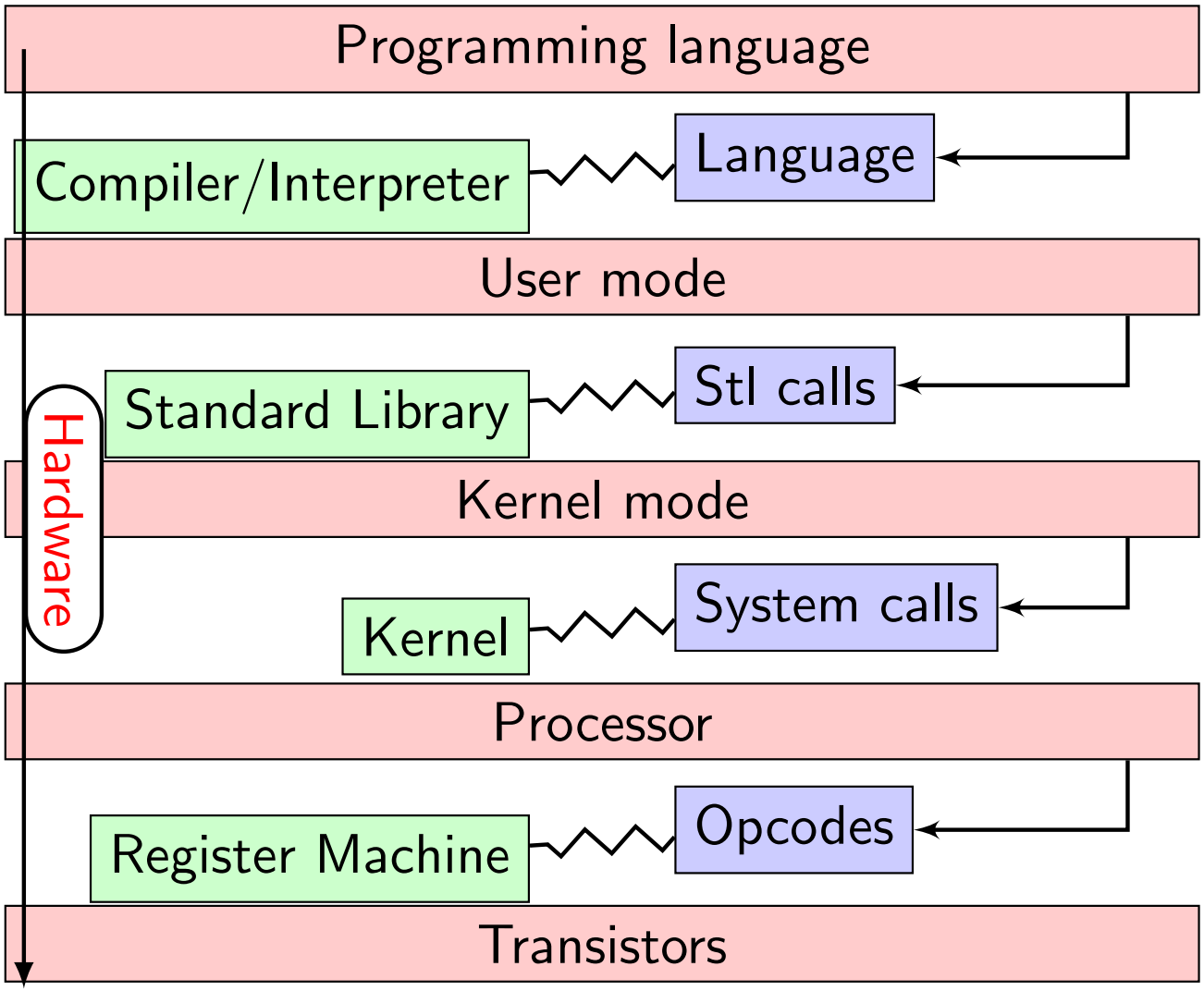
Layers, Abstractions and Interfaces

The base for the second idea



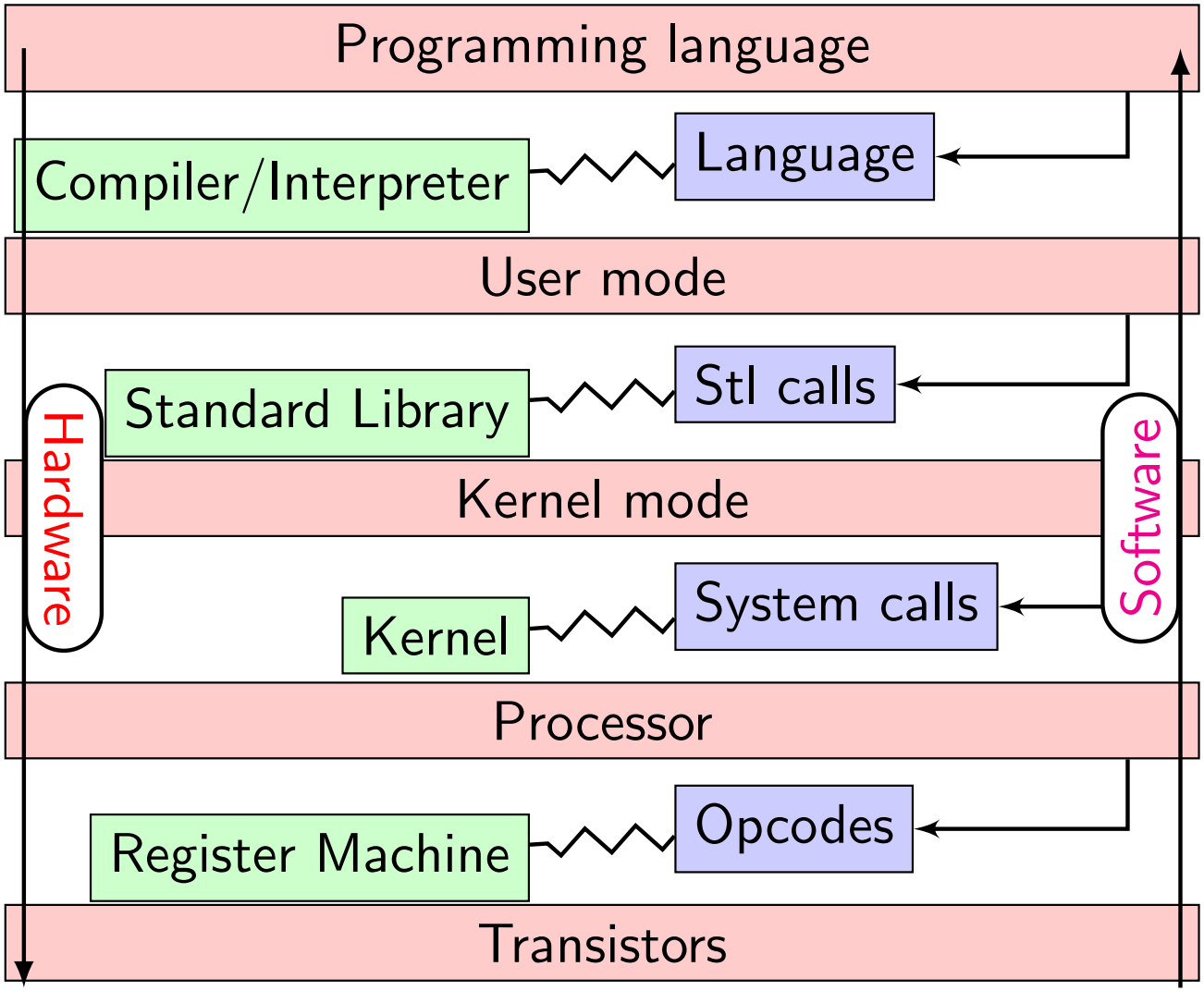
Layers, Abstractions and Interfaces

The base for the second idea



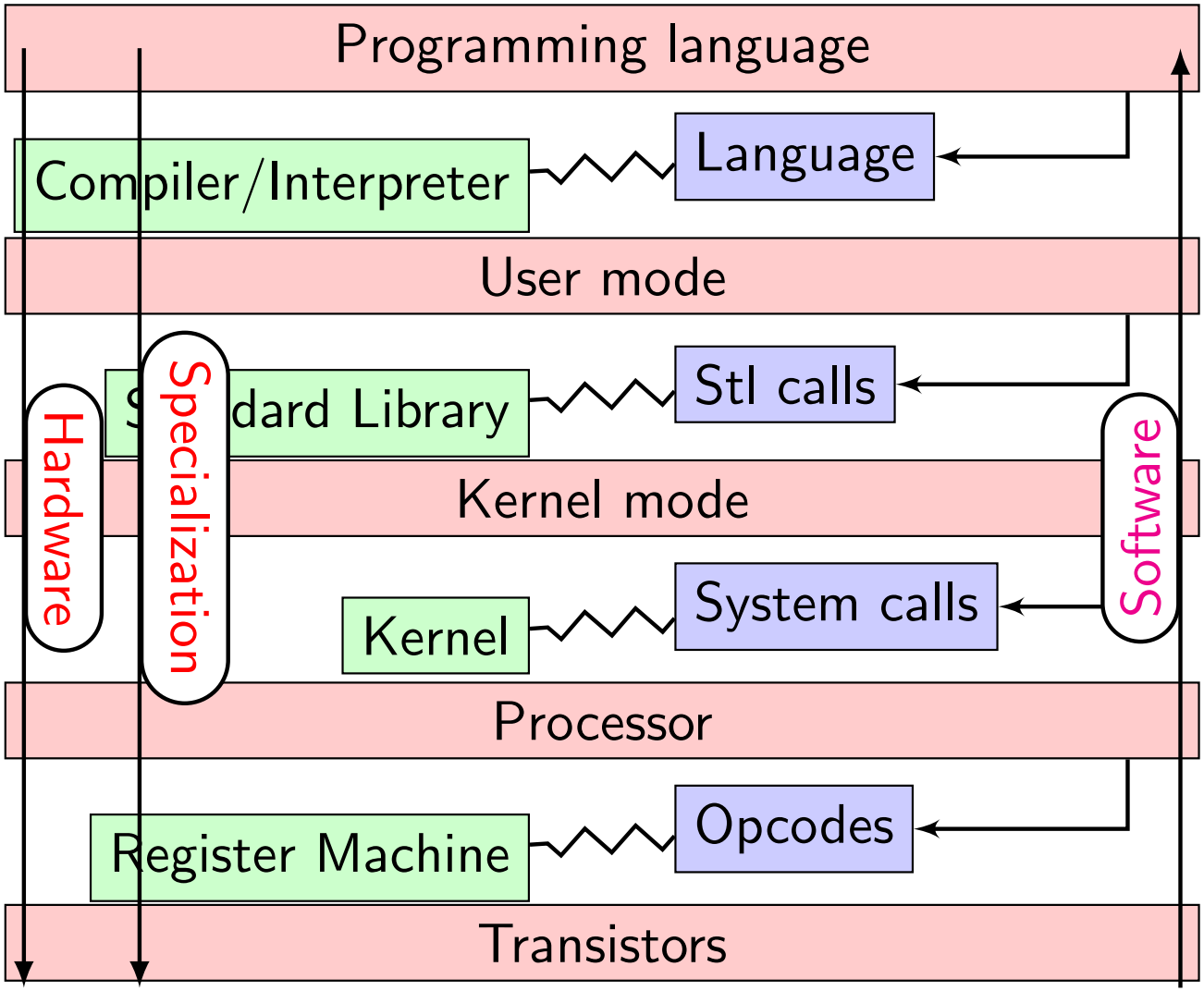
Layers, Abstractions and Interfaces

The base for the second idea



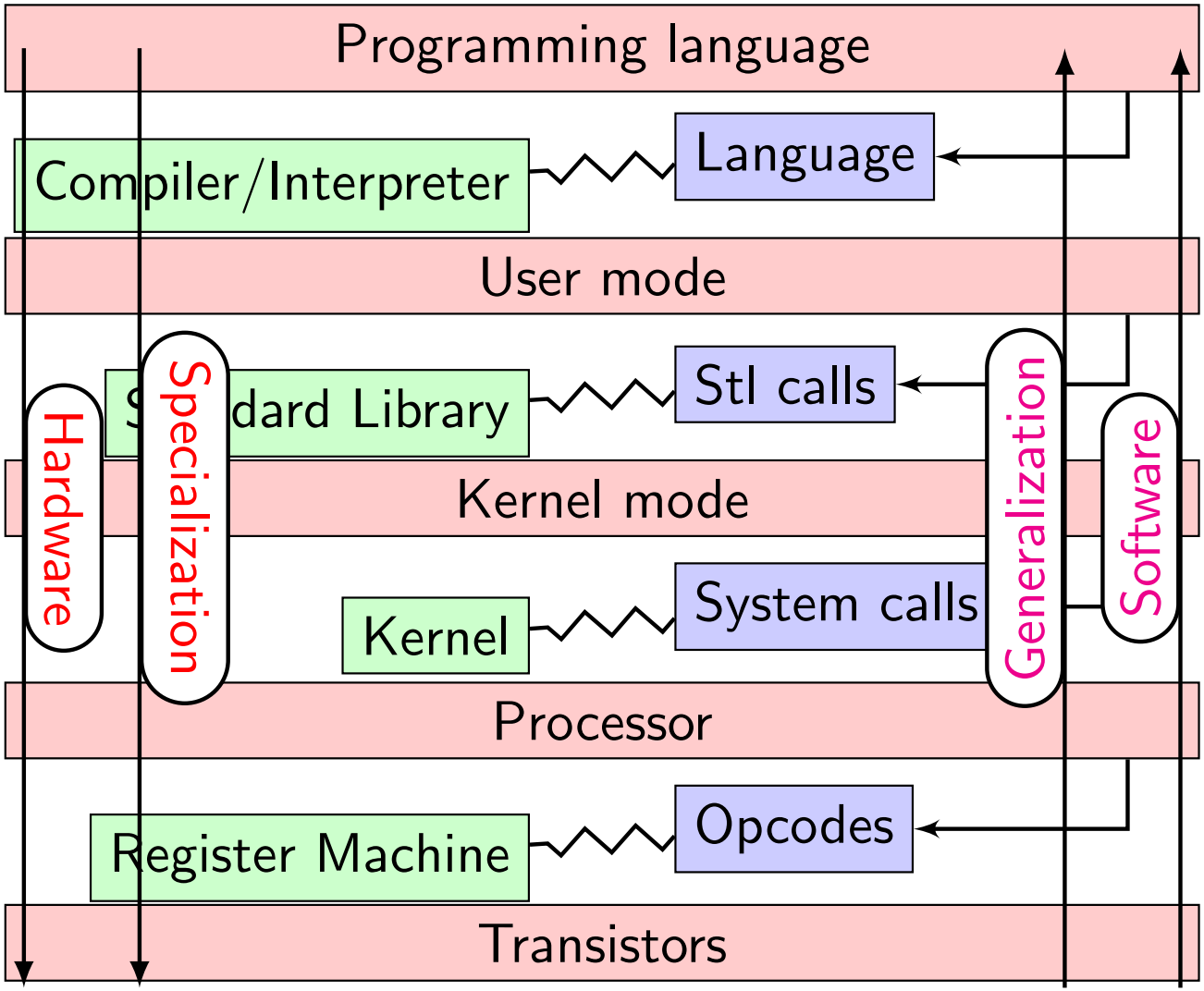
Layers, Abstractions and Interfaces

The base for the second idea



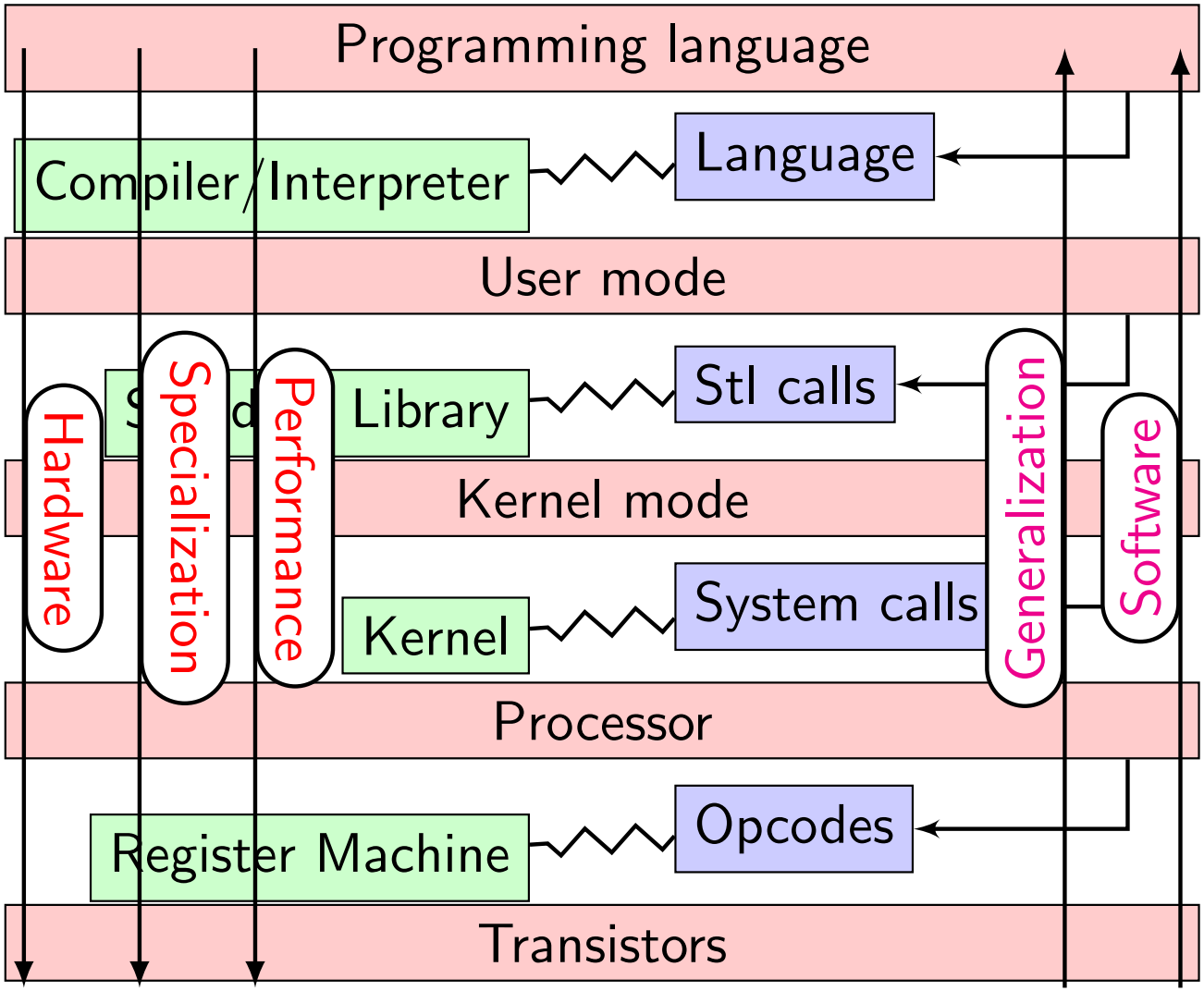
Layers, Abstractions and Interfaces

The base for the second idea



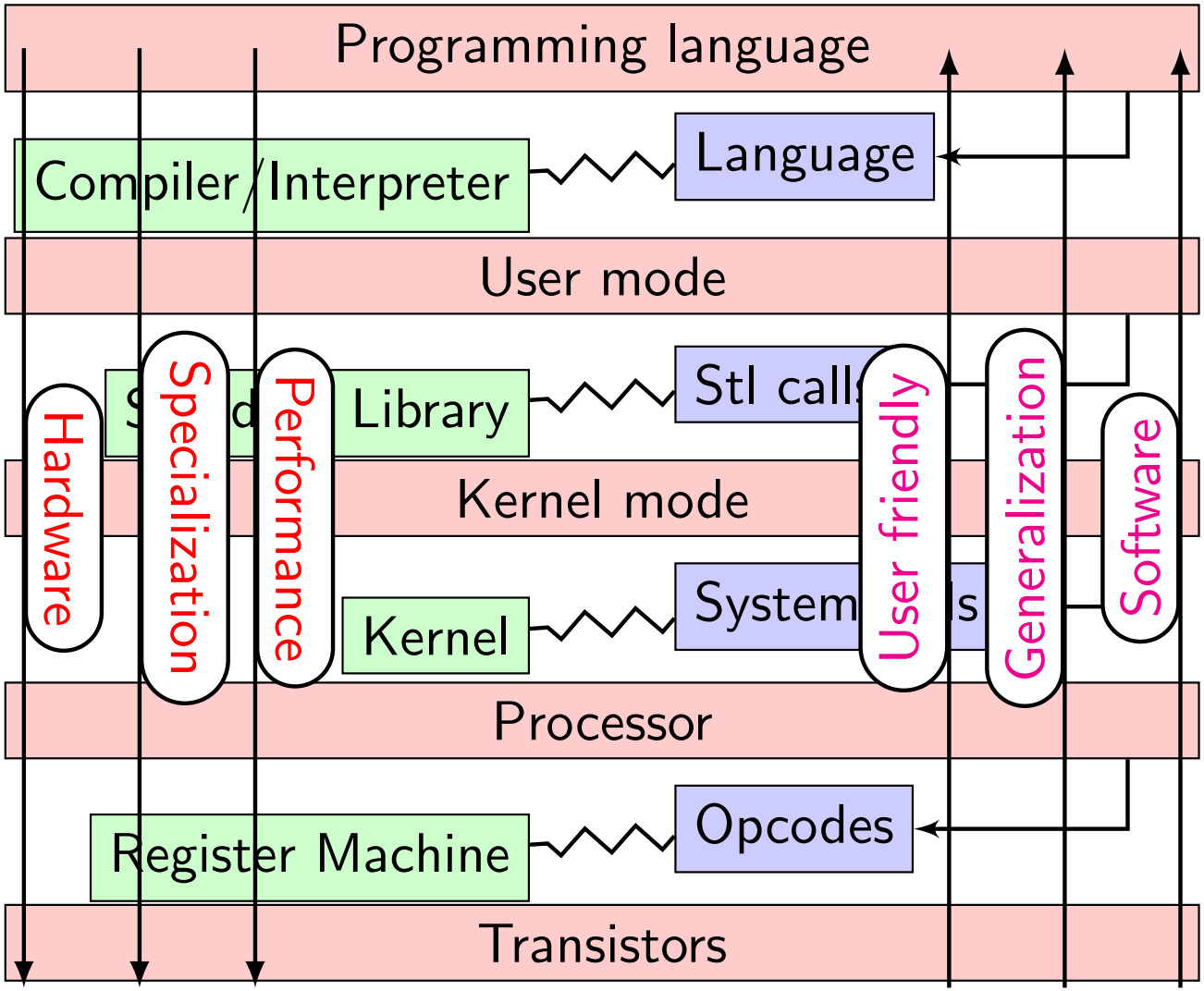
Layers, Abstractions and Interfaces

The base for the second idea



Layers, Abstractions and Interfaces

The base for the second idea



Layers, Abstractions and Interfaces

Second idea

Build a computing system with a decreased number of layers resulting in a minor gap between HW and SW but keeping an user friendly way of programming it.

Introducing the BondMachine (BM)

Inspired from both the ideas we create a new computer architecture that:

- Is composed by many, possibly hundreds, computing cores.
- Has very small cores and not necessarily of the same type (different ISA and ABI).
- Has a not fixed way of interconnecting cores.
- May have some elements shared among cores (for example channels and shared memories).

Introducing the BondMachine (BM)

Inspired from both the ideas we create a new computer architecture that:

- Is composed by many, possibly hundreds, computing cores.
- Has very small cores and not necessarily of the same type (different ISA and ABI).
- Has a not fixed way of interconnecting cores.
- May have some elements shared among cores (for example channels and shared memories).

Introducing the BondMachine (BM)

Inspired from both the ideas we create a new computer architecture that:

- Is composed by many, possibly hundreds, computing cores.
- Has very small cores and not necessarily of the same type (different ISA and ABI).
- Has a not fixed way of interconnecting cores.
- May have some elements shared among cores (for example channels and shared memories).

Introducing the BondMachine (BM)

Inspired from both the ideas we create a new computer architecture that:

- Is composed by many, possibly hundreds, computing cores.
- Has very small cores and not necessarily of the same type (different ISA and ABI).
- Has a not fixed way of interconnecting cores.
- May have some elements shared among cores (for example channels and shared memories).

Introducing the BondMachine (BM)

Inspired from both the ideas we create a new computer architecture that:

- Is composed by many, possibly hundreds, computing cores.
- Has very small cores and not necessarily of the same type (different ISA and ABI).
- Has a not fixed way of interconnecting cores.
- May have some elements shared among cores (for example channels and shared memories).

Connecting Processor (CP)

The computational unit of the BM

The atomic computational unit of a BM is the "connecting processor" (CP) and has:

- Some general purpose registers of size **Rsize**.
- Some I/O dedicated registers of size **Rsize**.
- A set of implemented opcodes chosen among many available.
- Dedicated ROM and RAM.
- Three possible operating modes.

Connecting Processor (CP)

The computational unit of the BM

The atomic computational unit of a BM is the "connecting processor" (CP) and has:

- Some general purpose registers of size **Rsize**.
- Some I/O dedicated registers of size **Rsize**.
- A set of implemented opcodes chosen among many available.
- Dedicated ROM and RAM.
- Three possible operating modes.

General purpose registers

2^R registers: r0,r1,r2,r3 ... $r2^R$

Connecting Processor (CP)

The computational unit of the BM

The atomic computational unit of a BM is the "connecting processor" (CP) and has:

- Some general purpose registers of size **Rsize**.
- Some I/O dedicated registers of size **Rsize**.
- A set of implemented opcodes chosen among many available.
- Dedicated ROM and RAM.
- Three possible operating modes.

I/O specialized registers

N input registers: i0,i1 ... iN
M output registers: o0,o1 ... oM

Connecting Processor (CP)

The computational unit of the BM

The atomic computational unit of a BM is the "connecting processor" (CP) and has:

- Some general purpose registers of size **Rsize**.
- Some I/O dedicated registers of size **Rsize**.
- A set of implemented opcodes chosen among many available.
- Dedicated ROM and RAM.
- Three possible operating modes.

Full set of possible opcodes

add,addf,addi,chc,chw,clr,cpy,dec,divf,dpc,hit,hlt,i2r,inc,j,
je,jz,m2r,mult,multf,nop,r2m,r2o,r2s,rset,sic,s2r,saj,sub,
wrd,wwr

Connecting Processor (CP)

The computational unit of the BM

The atomic computational unit of a BM is the "connecting processor" (CP) and has:

- Some general purpose registers of size **Rsize**.
- Some I/O dedicated registers of size **Rsize**.
- A set of implemented opcodes chosen among many available.
- Dedicated ROM and RAM.
- Three possible operating modes.

RAM and ROM

- 2^L RAM memory cells.
- 2^O ROM memory cells.

Connecting Processor (CP)

The computational unit of the BM

The atomic computational unit of a BM is the "connecting processor" (CP) and has:

- Some general purpose registers of size **Rsize**.
- Some I/O dedicated registers of size **Rsize**.
- A set of implemented opcodes chosen among many available.
- Dedicated ROM and RAM.
- Three possible operating modes.

Operating modes

- Full Harvard mode.
- Full Von Neuman mode.
- Hybrid mode.

Connecting Processor (CP)

Full set of possible opcodes

Opcode	Args	Description
add	reg_dst,reg_add	Add the values in reg_dst and reg_add writing the result in reg_dst
addf	reg_dst,reg_add	Add the values in reg_dst and reg_add writing the result in reg_dst (float32)
addi	reg_dst	Add the values of all the processor inputs in reg_dst
chc	reg_state, reg_op	Check for any channel operation, report the state and eventually which happened
chw	reg_op	Wait for any channel operation and report witch happened on reg_op
clr	reg	Set the register reg to 0
cpy	reg_dst, reg_src	Copy the value of a register to another
dec	reg	Decrement a register by 1
divf	reg_dst,reg_div	Divide the values in reg_dst by reg_div writing the result in reg_dst (float32)
dpc	reg_dest	Decode the program counter into a register
hit	reg_state, barrier_name	Hit a barrier, report the state
hlt	none	Halt the processor
i2r	reg_dst, input_name	Copy the value from an input to a register
inc	reg	Increment a register by 1
j	rom_address	Jump to a given instruction
je	reg1, reg2, rom_address	Jump if the register are equals
jz	reg1, rom_address	Jump if a register is zero
m2r	reg_dest, ram_address	Copy data from the RAM to a register
mult	reg_dst,reg_mult	Multiply the values in reg_mult and reg_dest writing the result in reg_dst
multf	reg_dst,reg_mult	Multiply the values in reg_mult and reg_dest writing the result in reg_dst (float32)
nop	none	No operation
r2m	reg_source, ram_address	Copy data from a register to the RAM
r2o	reg_src, output_name	Copy the value from a register to an output
r2s	reg_source, ram_name, ram_address	Copy data from a register to a shared RAM
rset	reg_dst, numeric_value	Set a value for a register
sic	reg_dst, input_name	Stop until Input Changes accumulating on a register
s2r	reg_dest, ram_name, ram_address	Copy data from a shared RAM to a register
saj	rom or ram_address	Switch operating mode and jump
sub	reg_dst,reg_sub	Subtract the values in reg_sub from reg_dest writing the result in reg_dst
wrd	reg_dst, channel_name	Want read from a channel to a register (set flag)
wwr	reg_src, channel_name	Want write to a channel from a register (set flag)

Shared Objects (SO)

Alongside CPs, BondMachines include non-computing units called "Shared Objects" (SO).

Examples of their purposes are:

- Data storage (Memories).
- Message passing.
- CP synchronization.

A single SO can be shared among different CPs. To use it CPs have special instructions (opcodes) oriented to the specific SO.

Three kind of SO have been developed so far: the Channel, the Shared Memory and the Barrier.

Shared Objects (SO)

Alongside CPs, BondMachines include non-computing units called "Shared Objects" (SO).

Examples of their purposes are:

- Data storage (Memories).
- Message passing.
- CP synchronization.

A single SO can be shared among different CPs. To use it CPs have special instructions (opcodes) oriented to the specific SO.

Three kind of SO have been developed so far: the Channel, the Shared Memory and the Barrier.

Shared Objects (SO)

Alongside CPs, BondMachines include non-computing units called "Shared Objects" (SO).

Examples of their purposes are:

- Data storage (Memories).
- Message passing.
- CP synchronization.

A single SO can be shared among different CPs. To use it CPs have special instructions (opcodes) oriented to the specific SO.

Three kind of SO have been developed so far: the Channel, the Shared Memory and the Barrier.

Shared Objects (SO)

Alongside CPs, BondMachines include non-computing units called "Shared Objects" (SO).

Examples of their purposes are:

- Data storage (Memories).
- Message passing.
- CP synchronization.

A single SO can be shared among different CPs. To use it CPs have special instructions (opcodes) oriented to the specific SO.

Three kind of SO have been developed so far: the Channel, the Shared Memory and the Barrier.

Shared Objects (SO)

Alongside CPs, BondMachines include non-computing units called "Shared Objects" (SO).

Examples of their purposes are:

- Data storage (Memories).
- Message passing.
- CP synchronization.

A single SO can be shared among different CPs. To use it CPs have special instructions (opcodes) oriented to the specific SO.

Three kind of SO have been developed so far: the Channel, the Shared Memory and the Barrier.

Shared Objects (SO)

Alongside CPs, BondMachines include non-computing units called "Shared Objects" (SO).

Examples of their purposes are:

- Data storage (Memories).
- Message passing.
- CP synchronization.

A single SO can be shared among different CPs. To use it CPs have special instructions (opcodes) oriented to the specific SO.

Three kind of SO have been developed so far: the Channel, the Shared Memory and the Barrier.

Channel

The Channel SO is an hardware implementation of the CSP (communicating sequential processes) channel.

It is a model for inter-core communication and synchronization via message passing.

CPs use channels via 4 opcodes

- *wrd*: Want Read.
- *wwr*: Want Write.
- *chc*: Channel Check.
- *chw*: Channel Wait.

Channel

The Channel SO is an hardware implementation of the CSP (communicating sequential processes) channel.

It is a model for inter-core communication and synchronization via message passing.

CPs use channels via 4 opcodes

- *wrd*: Want Read.
- *wwr*: Want Write.
- *chc*: Channel Check.
- *chw*: Channel Wait.

Channel

The Channel SO is an hardware implementation of the CSP (communicating sequential processes) channel.

It is a model for inter-core communication and synchronization via message passing.

CPs use channels via 4 opcodes

- *wrd*: Want Read.
- *wwr*: Want Write.
- *chc*: Channel Check.
- *chw*: Channel Wait.

Shared Memory

The Shared Memory SO is a RAM block accessible from more than one CP.

Different Shared Memories can be used by different CP and not necessarily by all of them.

CPs use shared memories via 2 opcodes

- *s2r*: Shared memory read.
- *r2s*: Shared memory write.

Shared Memory

The Shared Memory SO is a RAM block accessible from more than one CP.

Different Shared Memories can be used by different CP and not necessarily by all of them.

CPs use shared memories via 2 opcodes

- *s2r*: Shared memory read.
- *r2s*: Shared memory write.

Shared Memory

The Shared Memory SO is a RAM block accessible from more than one CP.

Different Shared Memories can be used by different CP and not necessarily by all of them.

CPs use shared memories via 2 opcodes

- *s2r*: Shared memory read.
- *r2s*: Shared memory write.

Barrier

The Barrier SO is used to make CPs act synchronously.

When a CP hits a barrier, the execution stop until all the CPs that share the same barrier hit it.

CPs use barriers via 1 opcode

- *hit*: Hit the barrier.

Barrier

The Barrier SO is used to make CPs act synchronously.

When a CP hits a barrier, the execution stop until all the CPs that share the same barrier hit it.

CPs use barriers via 1 opcode

- *hit*: Hit the barrier.

Barrier

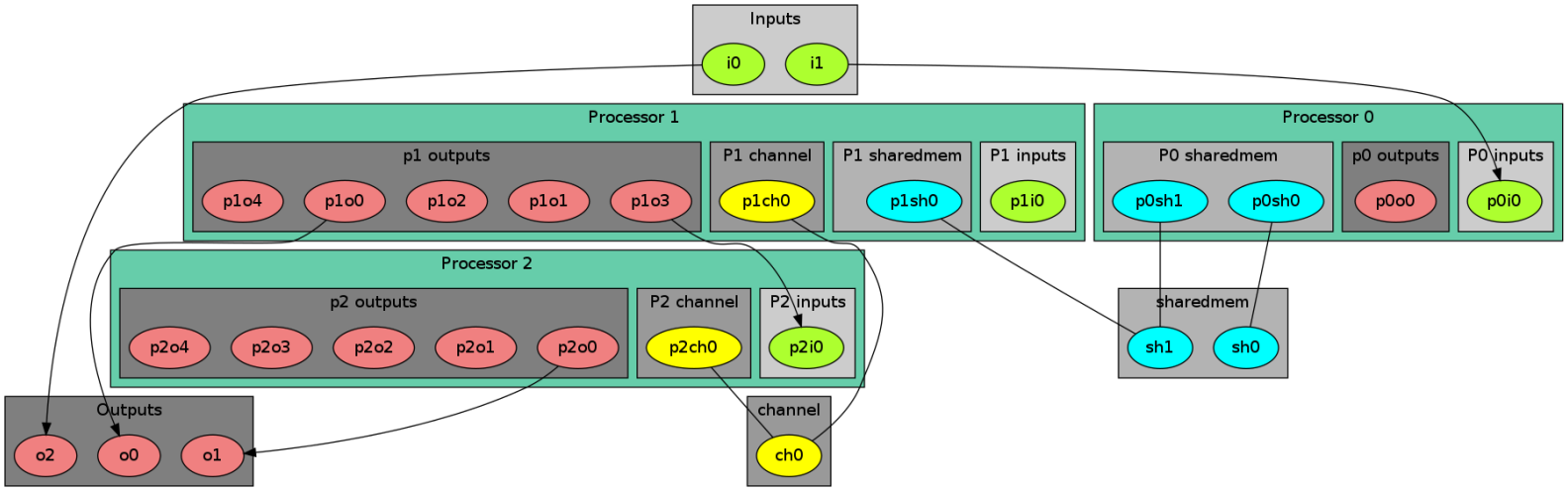
The Barrier SO is used to make CPs act synchronously.

When a CP hits a barrier, the execution stop until all the CPs that share the same barrier hit it.

CPs use barriers via 1 opcode

- *hit*: Hit the barrier.

The BondMachine



Multicore and Heterogeneous

First idea on the BondMachine

The idea was:

Having a multi-core architecture completely heterogeneous both in cores types and interconnections.

The BondMachine may have many cores, eventually all different, arbitrarily interconnected and sharing non computing elements.

Tools

The complexity of programming the BondMachine architecture is managed by using a set of software tools to:

- build a specify architecture as function of the task,
- modify the created architecture,
- simulate the behavior and to check the functionality with the aim to generate the Register Transfer Level (RTL) code.

Processor Builder selects the CP specifics, assembles and disassembles, saves on disk as JSON, emulates and creates the RTL code.

BondMachine Builder connects CPs and SOs together in custom topologies, loads and saves on disk as JSON, emulates and creates the RTL code.

Tools

The complexity of programming the BondMachine architecture is managed by using a set of software tools to:

- build a specify architecture as function of the task,
- modify the created architecture,
- simulate the behavior and to check the functionality with the aim to generate the Register Transfer Level (RTL) code.

Processor Builder selects the CP specifics, assembles and disassembles, saves on disk as JSON, emulates and creates the RTL code.

BondMachine Builder connects CPs and SOs together in custom topologies, loads and saves on disk as JSON, emulates and creates the RTL code.

Tools

The complexity of programming the BondMachine architecture is managed by using a set of software tools to:

- build a specify architecture as function of the task,
- modify the created architecture,
- simulate the behavior and to check the functionality with the aim to generate the Register Transfer Level (RTL) code.

Processor Builder selects the CP specifics, assembles and disassembles, saves on disk as JSON, emulates and creates the RTL code.

BondMachine Builder connects CPs and SOs together in custom topologies, loads and saves on disk as JSON, emulates and creates the RTL code.

Processor Builder

Procbuilder is the CP manipulation tool.

- CP Creation
- CP Load/Save
- CP Assembler/Disassembler
- CP RTL

Examples

(32 bit registers counter machine)

procbuilder -register-size 32 -opcodes clr,cpy,dec,inc,je,jz

(CP dedicated RAM and ROM)

procbuilder -ram 256 ... ; procbuilder -rom 16 ...

(Input and Output registers)

procbuilder -inputs 3 -outputs 2 ...

Processor Builder

Procbuilder is the CP manipulation tool.

- CP Creation
- CP Load/Save
- CP Assembler/Disassembler
- CP RTL

Examples

(Loading a CP)

procbuilder -load-machine conproc.json ...

(Saving a CP)

procbuilder -save-machine conproc.json ...

Processor Builder

Procbuilder is the CP manipulation tool.

CP Creation
CP Load/Save
CP Assembler/Disassembler
CP RTL

Examples

(Assembling)

procbuilder -input-assembly program.asm ...

(Disassembling)

procbuilder -show-program-disassembled ...

Processor Builder

Procbuilder is the CP manipulation tool.

- CP Creation
- CP Load/Save
- CP Assembler/Disassembler
- CP RTL

Examples

(Create the CP RTL code in Verilog)

procbuilder -create-verilog ...

(Create testbench)

procbuilder -create-verilog-testbench test.v ...

BondMachine Builder

Bondmachine is the tool that compose CP and SO to form BondMachines.

BM CP insert and remove
BM SO insert and remove
BM Inputs and Outputs
BM Bonding Processors and/or IO
BM Visualizing or RTL

Examples

(Add a processor)

bondmachine -add-domains processor.json ...
bondmachine -add-processor 0 ...

(Remove a processor)

bondmachine -bondmachine-file bmach.json -del-processor n

BondMachine Builder

Bondmachine is the tool that compose CP and SO to form BondMachines.

BM CP insert and remove

BM SO insert and remove

BM Inputs and Outputs

BM Bonding Processors and/or IO

BM Visualizing or RTL

Examples

(Add a Shared Object)

bondmachine -add-shared-objects specs ...

(Connect an SO to a processor)

bondmachine -connect-processor-shared-object ...

BondMachine Builder

Bondmachine is the tool that compose CP and SO to form BondMachines.

BM CP insert and remove
BM SO insert and remove
BM Inputs and Outputs
BM Bonding Processors and/or IO
BM Visualizing or RTL

Examples

(Adding inputs or outputs)

bondmachine -add-inputs ... ; bondmachine -add-outputs ...

(Removing inputs or outputs)

bondmachine -del-input ... ; bondmachine -del-output ...

BondMachine Builder

Bondmachine is the tool that compose CP and SO to form BondMachines.

BM CP insert and remove
BM SO insert and remove
BM Inputs and Outputs
BM Bonding Processors and/or IO
BM Visualizing or RTL

Examples

(Bonding processor)

bondmachine -add-bond p0i2,p1o4 ...

(Bonding IO)

bondmachine -add-bond i2,p0i6 ...

BondMachine Builder

Bondmachine is the tool that compose CP and SO to form BondMachines.

BM CP insert and remove
BM SO insert and remove
BM Inputs and Outputs
BM Bonding Processors and/or IO
BM Visualizing or RTL

Examples

(Visualizing)

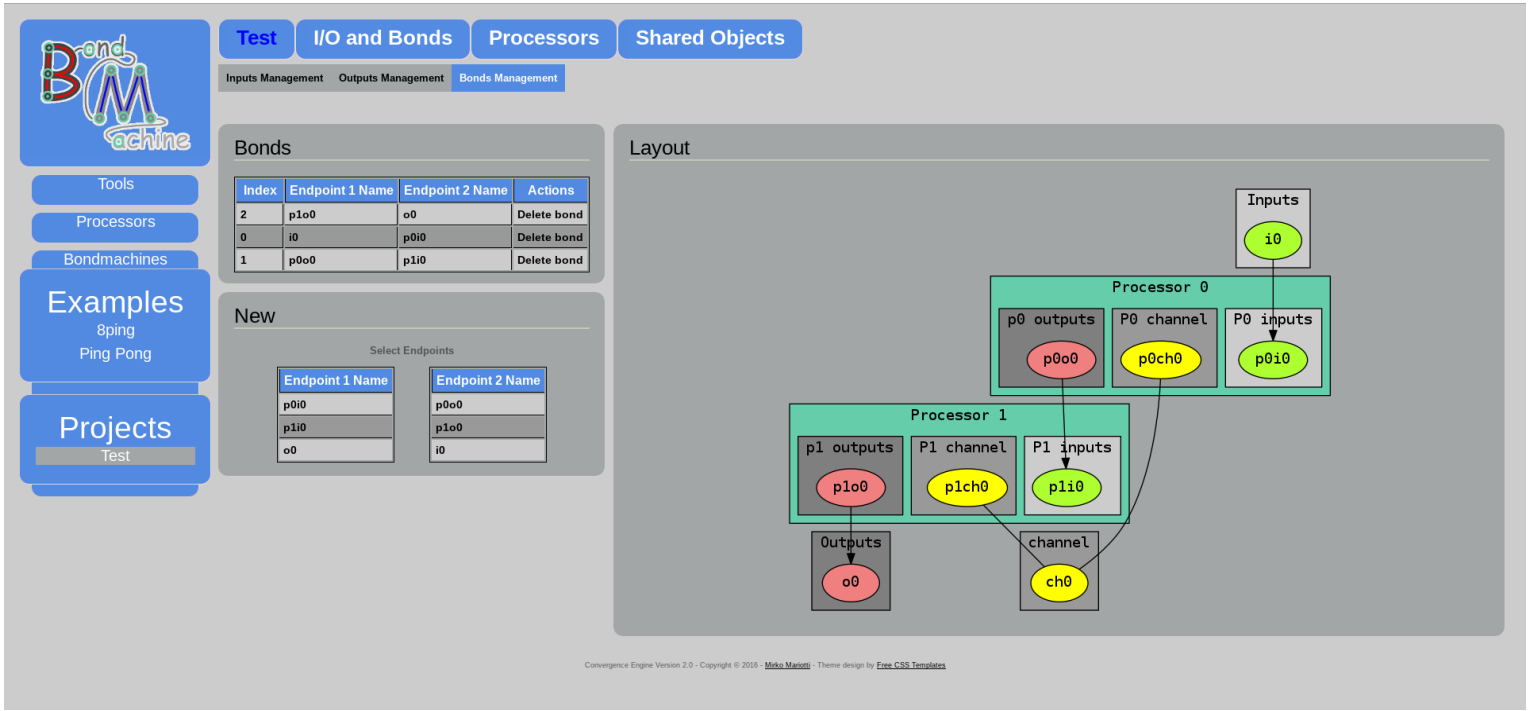
bondmachine -emit-dot ...

(Create RTL code)

bondmachine -create-verilog ...

BondMachine web front-end

Operations on BondMachines can also be performed via a developed web framework



Simulation

An important feature of the tools is the possibility of simulating BondMachine behavior.

An event input file describes how BondMachines elements has to change during the simulation timespan and which one has to be be reported.

The simulator can produce results in the form of:

- Activity log of the BM internal.
- Graphical representation of the simulation.
- Report file with quantitative data. Useful to construct metrics

Simulation

An important feature of the tools is the possibility of simulating BondMachine behavior.

An event input file describes how BondMachines elements has to change during the simulation timespan and which one has to be be reported.

The simulator can produce results in the form of:

- Activity log of the BM internal.
- Graphical representation of the simulation.
- Report file with quantitative data. Useful to construct metrics

Simulation

An important feature of the tools is the possibility of simulating BondMachine behavior.

An event input file describes how BondMachines elements has to change during the simulation timespan and which one has to be be reported.

The simulator can produce results in the form of:

- Activity log of the BM internal.
- Graphical representation of the simulation.
- Report file with quantitative data. Useful to construct metrics

Simulation

examples

Activity log example:

```
[discovery]> /home/mirko/Projects/comproc/tests/asm2sim % bondmachine -register-size 8 -bondmachine-file  asmtest05.json  -sim -sim-in
Loading simbox rule: config:show_pc
Loading simbox rule: config:show_ticks
Loading simbox rule: config:show_instruction
Loading simbox rule: config:show_disasm
Loading simbox rule: config:show_proc_io_pre
Loading simbox rule: config:show_proc_io_post
Loading simbox rule: config:show_proc_regs_pre
Loading simbox rule: config:show_proc_regs_post
Loading simbox rule: config:show_io_post
Loading simbox rule: config:show_io_pre
Loading simbox rule: absolute:1:set:i0:2
Absolute tick:0
        Pre-compute IO: i0: 00000000 o0: 00000000
        Proc: 0
                PC: 0
                Instr: 00000
                Disasm: i2r r0 i0
                Pre-compute IO: i0: 00000000 o0: 00000000
                Pre-compute Regs: r0: 00000000 r1: 00000000
                Post-compute IO: i0: 00000000 o0: 00000000
                Post-compute Regs: r0: 00000000 r1: 00000000
        Post-compute IO: i0: 00000000 o0: 00000000
Absolute tick:1
        Pre-compute IO: i0: 00000010 o0: 00000000
        Proc: 0
                PC: 1
                Instr: 00000
                Disasm: i2r r0 i0
                Pre-compute IO: i0: 00000010 o0: 00000000
                Pre-compute Regs: r0: 00000000 r1: 00000000
                Post-compute IO: i0: 00000010 o0: 00000000
                Post-compute Regs: r0: 00000010 r1: 00000000
        Post-compute IO: i0: 00000010 o0: 00000000
Absolute tick:2
```

A graphical example:

https://youtube.com/embed/Cc1Qzioh2Ng

Emulation

The same engine that simulate BondMachines can be used as emulator.

Through the emulator BondMachines can be used on Linux workstations.

Moulding the BondMachine

As stated before BondMachines are not general purpose architectures, and to be effective have to be shaped according the specific problem.

Several methods (apart from writing in assembly and building a BondMachine from scratch) have been developed to do that:

- *bondgo*: A new type of compiler that create not only the CPs assembly but also the architecture itself.
- A set of API to create BondMachine to fit a specific computational problems.
- An Evolutionary Computation framework to "grow" BondMachines according some fitness function via simulation.
- *tf2bm*: A TensorFlow to BondMachine translator.

Moulding the BondMachine

As stated before BondMachines are not general purpose architectures, and to be effective have to be shaped according the specific problem.

Several methods (apart from writing in assembly and building a BondMachine from scratch) have been developed to do that:

- *bondgo*: A new type of compiler that create not only the CPs assembly but also the architecture itself.
- A set of API to create BondMachine to fit a specific computational problems.
- An Evolutionary Computation framework to "grow" BondMachines according some fitness function via simulation.
- *tf2bm*: A TensorFlow to BondMachine translator.

Moulding the BondMachine

As stated before BondMachines are not general purpose architectures, and to be effective have to be shaped according the specific problem.

Several methods (apart from writing in assembly and building a BondMachine from scratch) have been developed to do that:

- *bondgo*: A new type of compiler that create not only the CPs assembly but also the architecture itself.
- A set of API to create BondMachine to fit a specific computational problems.
- An Evolutionary Computation framework to "grow" BondMachines according some fitness function via simulation.
- *tf2bm*: A TensorFlow to BondMachine translator.

Moulding the BondMachine

As stated before BondMachines are not general purpose architectures, and to be effective have to be shaped according the specific problem.

Several methods (apart from writing in assembly and building a BondMachine from scratch) have been developed to do that:

- *bondgo*: A new type of compiler that create not only the CPs assembly but also the architecture itself.
- A set of API to create BondMachine to fit a specific computational problems.
- An Evolutionary Computation framework to "grow" BondMachines according some fitness function via simulation.
- *tf2bm*: A TensorFlow to BondMachine translator.

Moulding the BondMachine

As stated before BondMachines are not general purpose architectures, and to be effective have to be shaped according the specific problem.

Several methods (apart from writing in assembly and building a BondMachine from scratch) have been developed to do that:

- *bondgo*: A new type of compiler that create not only the CPs assembly but also the architecture itself.
- A set of API to create BondMachine to fit a specific computational problems.
- An Evolutionary Computation framework to "grow" BondMachines according some fitness function via simulation.
- *tf2bm*: A TensorFlow to BondMachine translator.

Moulding the BondMachine

As stated before BondMachines are not general purpose architectures, and to be effective have to be shaped according the specific problem.

Several methods (apart from writing in assembly and building a BondMachine from scratch) have been developed to do that:

- *bondgo*: A new type of compiler that create not only the CPs assembly but also the architecture itself.
- A set of API to create BondMachine to fit a specific computational problems.
- An Evolutionary Computation framework to "grow" BondMachines according some fitness function via simulation.
- *tf2bm*: A TensorFlow to BondMachine translator.

Bondgo

bondgo is the name chosen for the compiler developed for the BondMachine.

The compiler source language is Go as the name suggest.

Bondgo

This is the standard flow when building computer programs

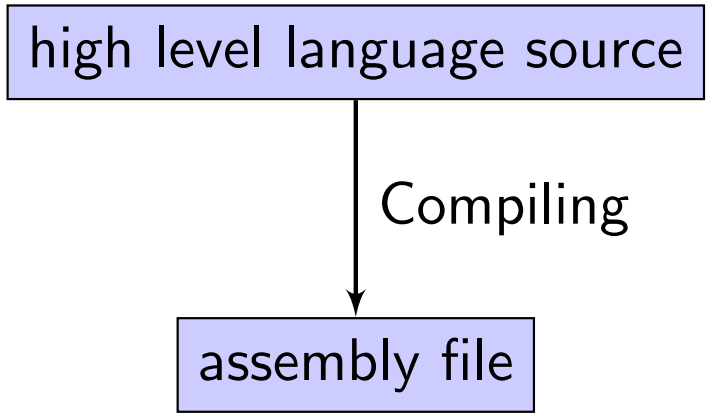
Bondgo

This is the standard flow when building computer programs

high level language source

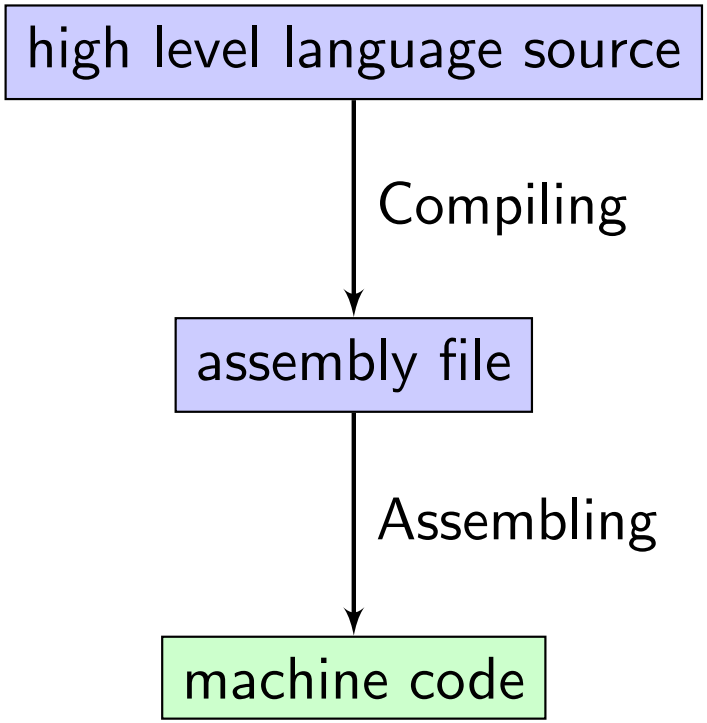
Bondgo

This is the standard flow when building computer programs



Bondgo

This is the standard flow when building computer programs



Bondgo

The standard flow in bondgo

bondgo loop example

```
package main

import ()

func main() {
   var reg_aa uint8
   var reg_ab uint8
   for reg_aa = 10; reg_aa > 0; reg_aa-- {
      reg_ab = reg_aa
      break
   }
}
```

bondgo loop example in asm

```
clr aa
clr ab
rset ac 10
cpy aa ac
cpy ac aa
jz ac 11
cpy ac aa
cpy ab ac
j 11
dec aa
j 4
```

Bondgo

bondgo may also do something different when compiling a single threaded program ...

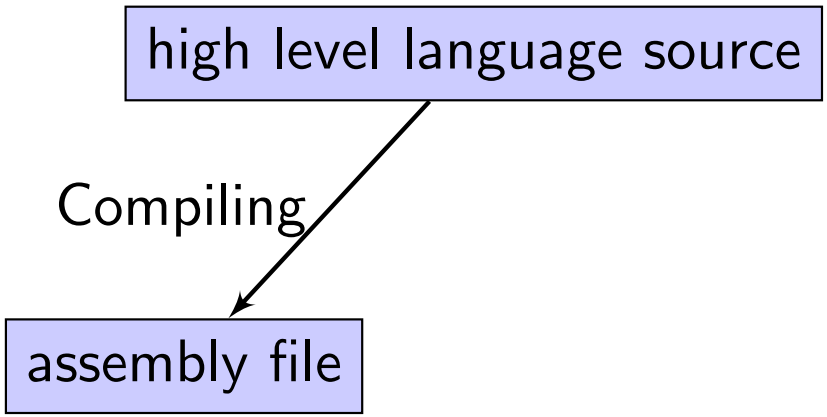
Bondgo

bondgo may also do something different when compiling a single threaded program ...

high level language source

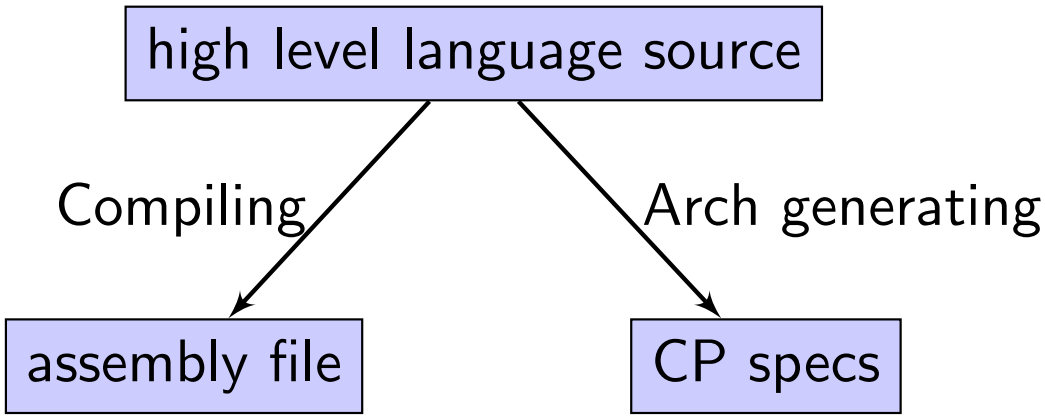
Bondgo

bondgo may also do something different when compiling a single threaded program ...



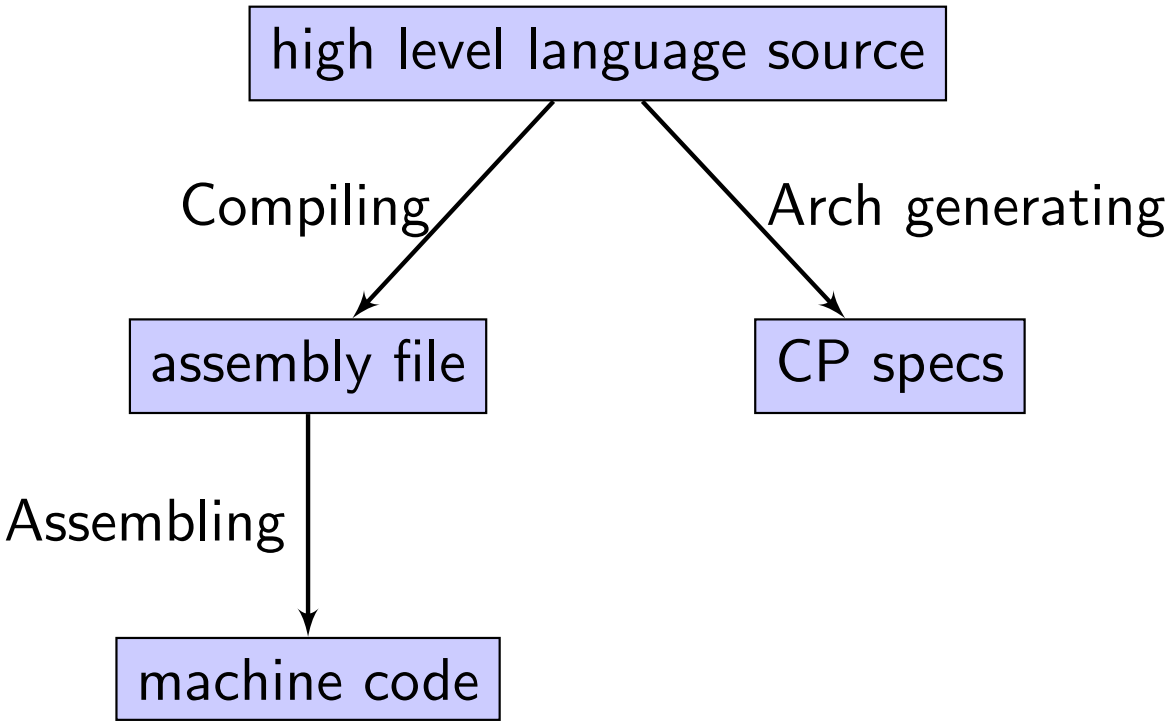
Bondgo

bondgo may also do something different when compiling a single threaded program ...



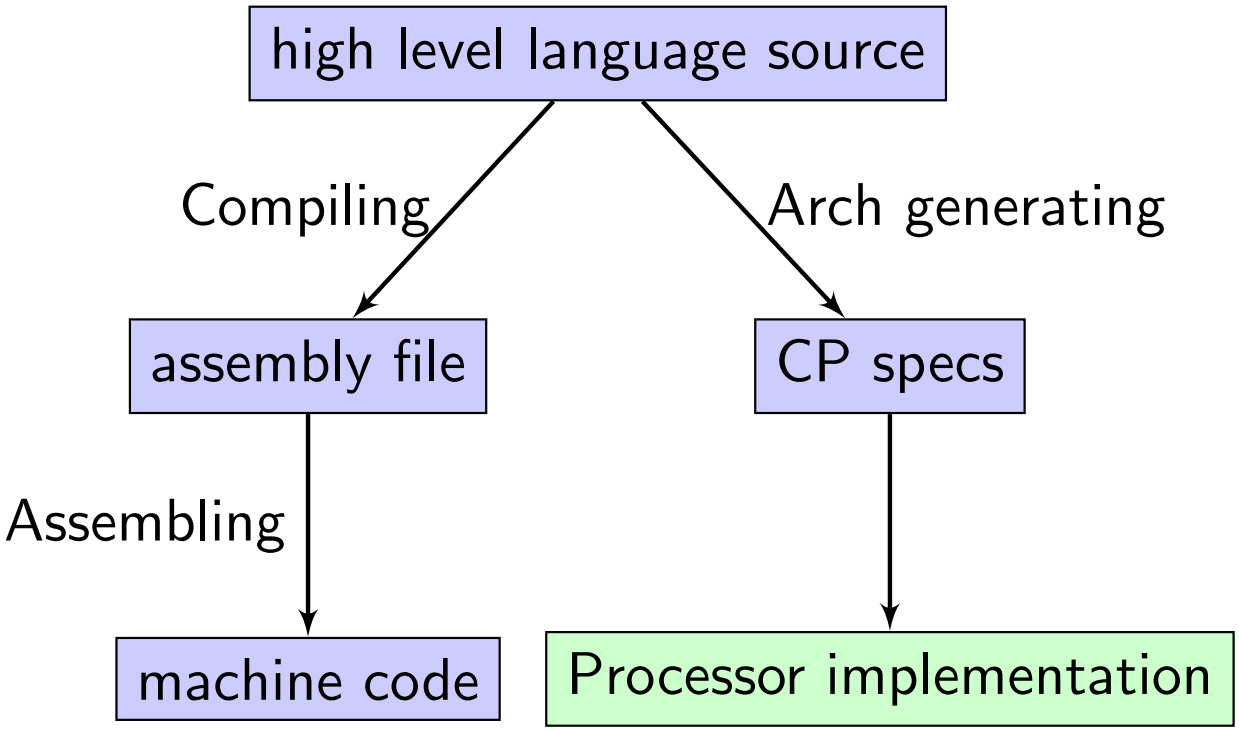
Bondgo

bondgo may also do something different when compiling a single threaded program ...



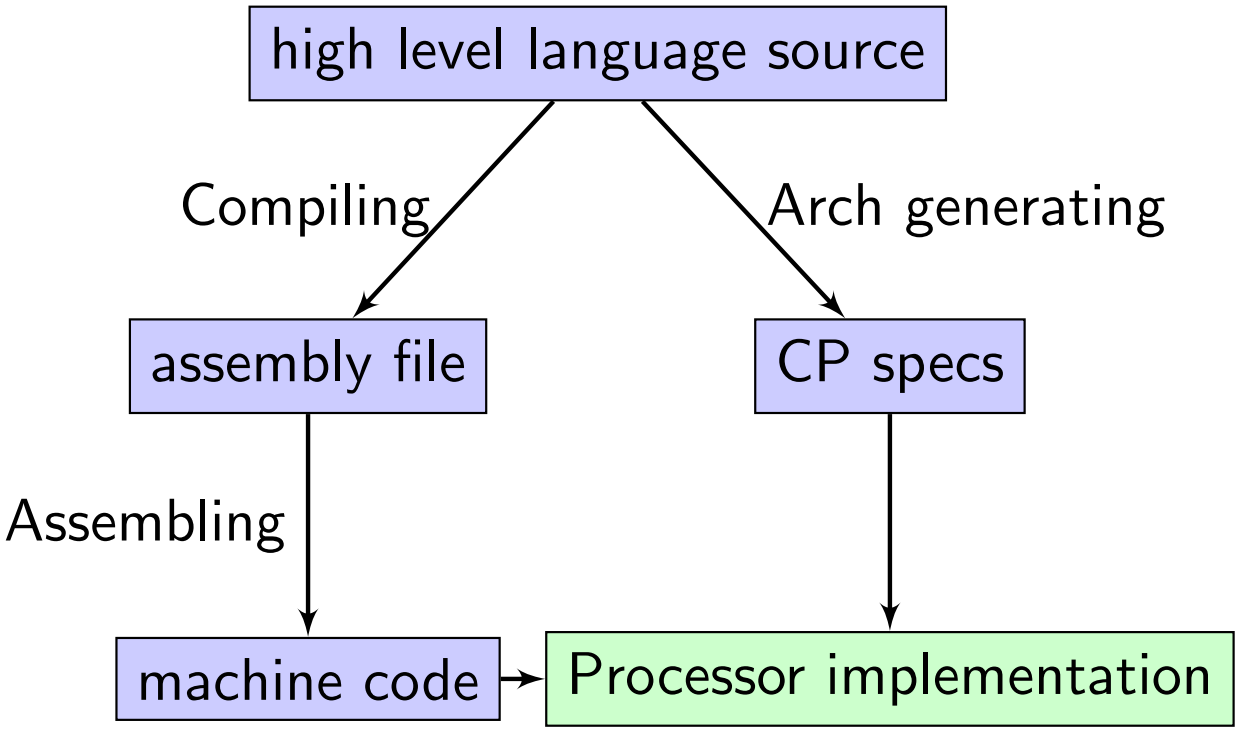
Bondgo

bondgo may also do something different when compiling a single threaded program ...



Bondgo

bondgo may also do something different when compiling a single threaded program ...



Bondgo

... *bondgo* may not only create the binaries, but also the CP architecture, and ...

Bondgo

... it can do even much more interesting things when compiling concurrent programs.

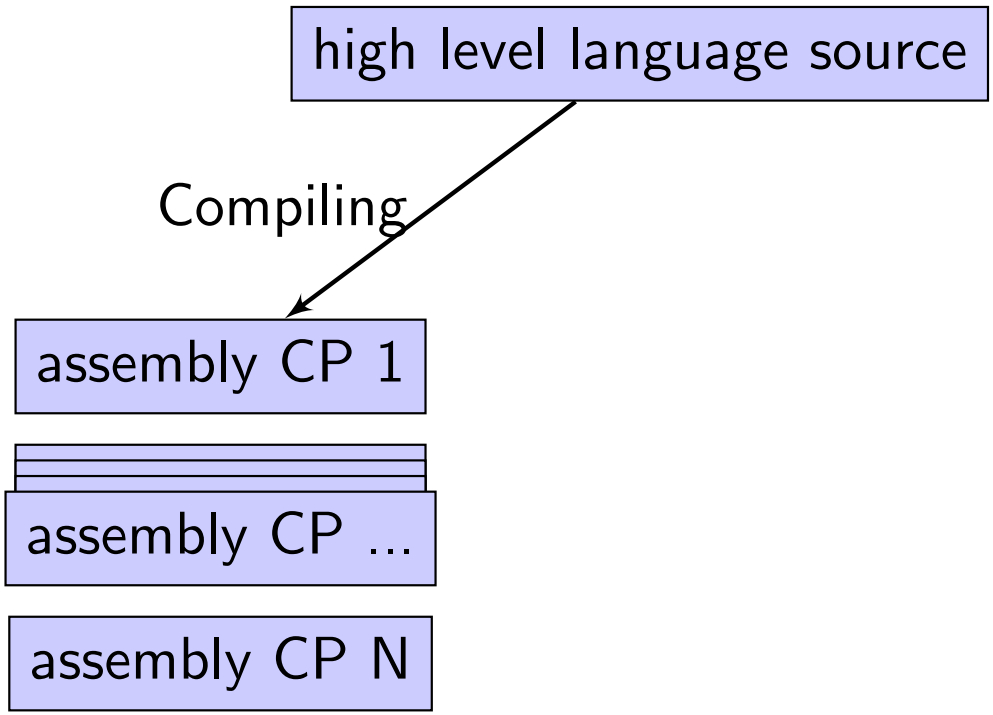
Bondgo

... it can do even much more interesting things when compiling concurrent programs.

high level language source

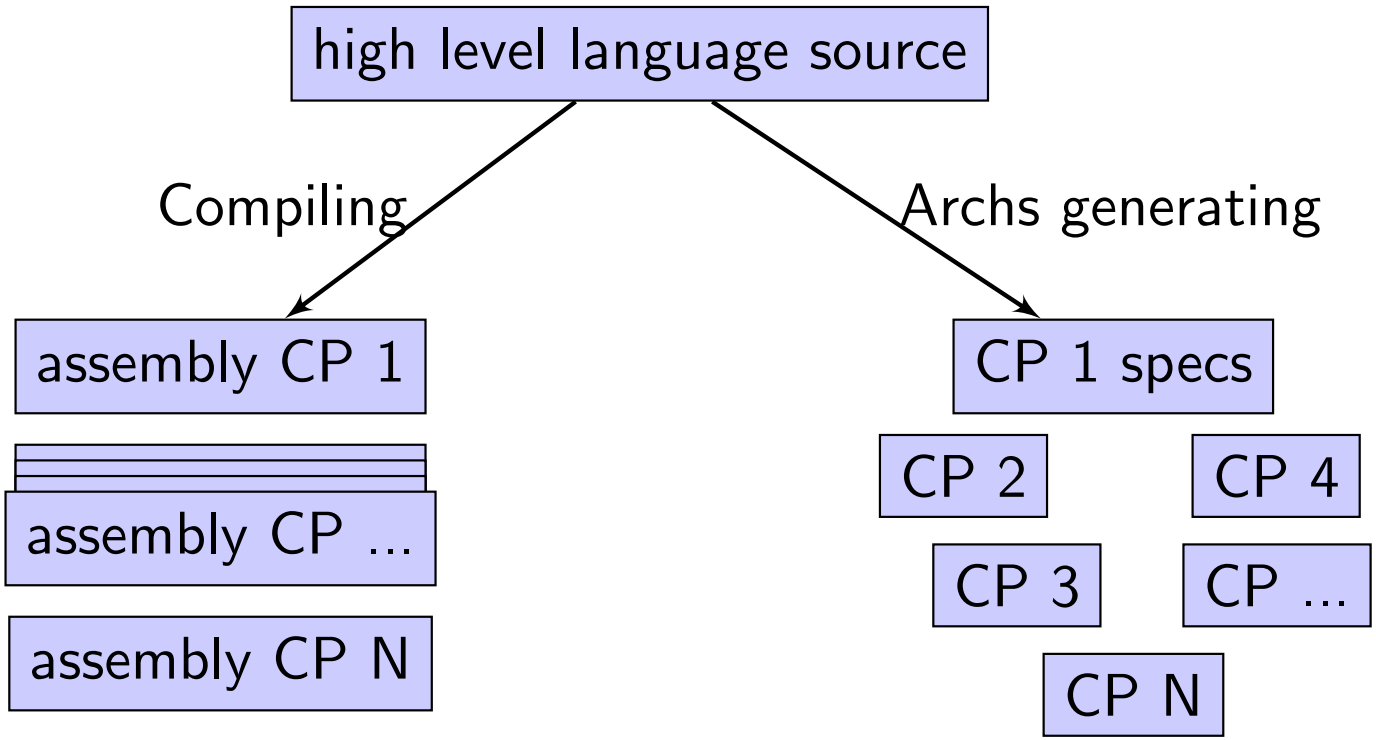
Bondgo

... it can do even much more interesting things when compiling concurrent programs.



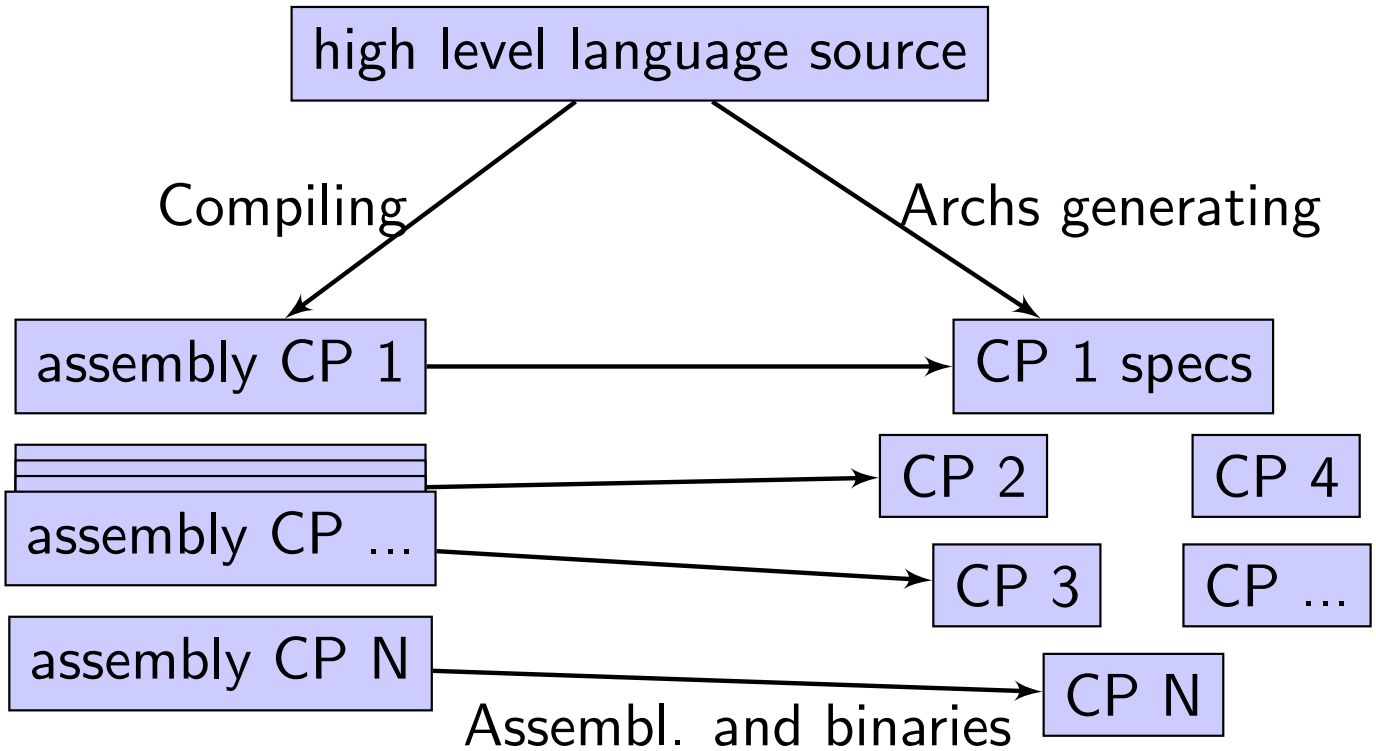
Bondgo

... it can do even much more interesting things when compiling concurrent programs.



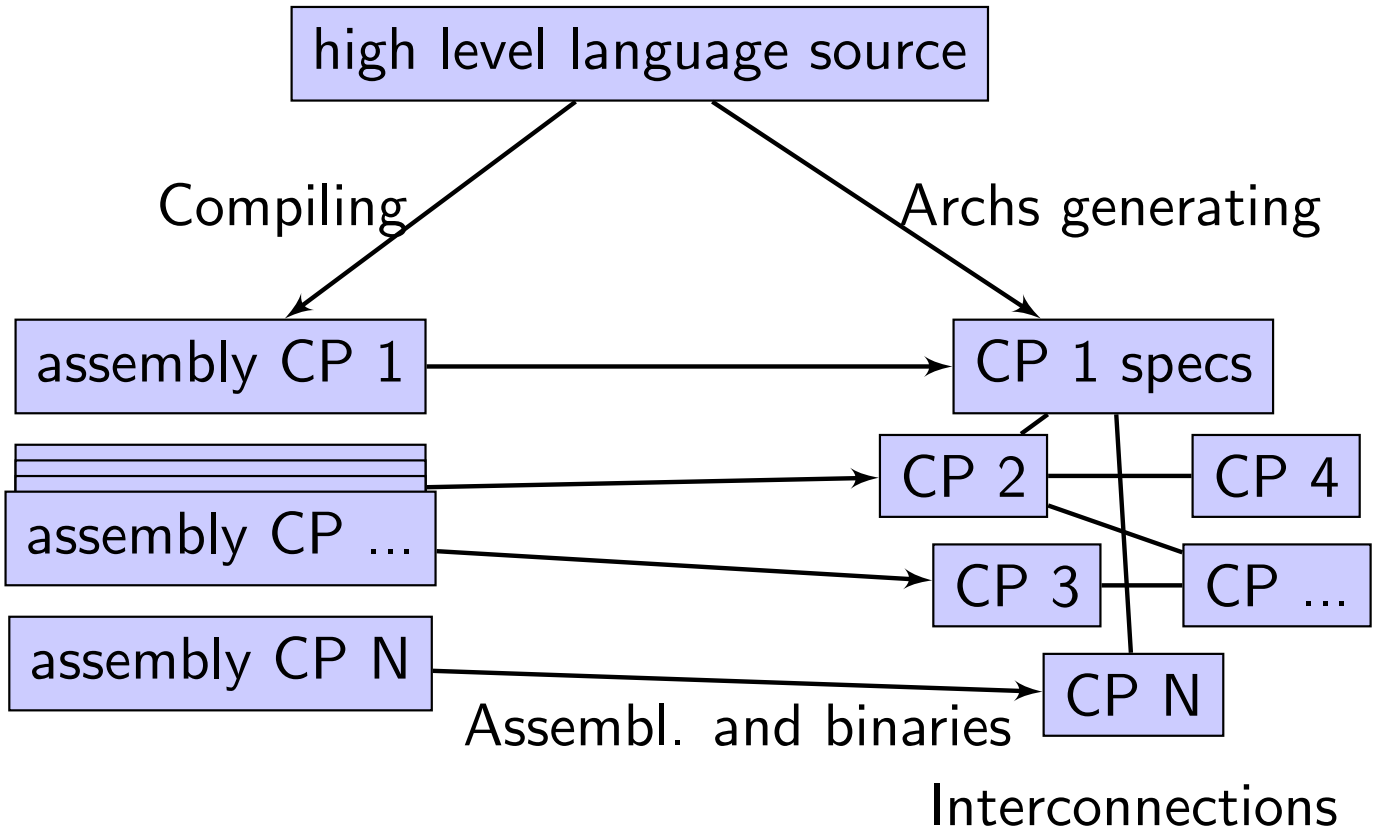
Bondgo

... it can do even much more interesting things when compiling concurrent programs.



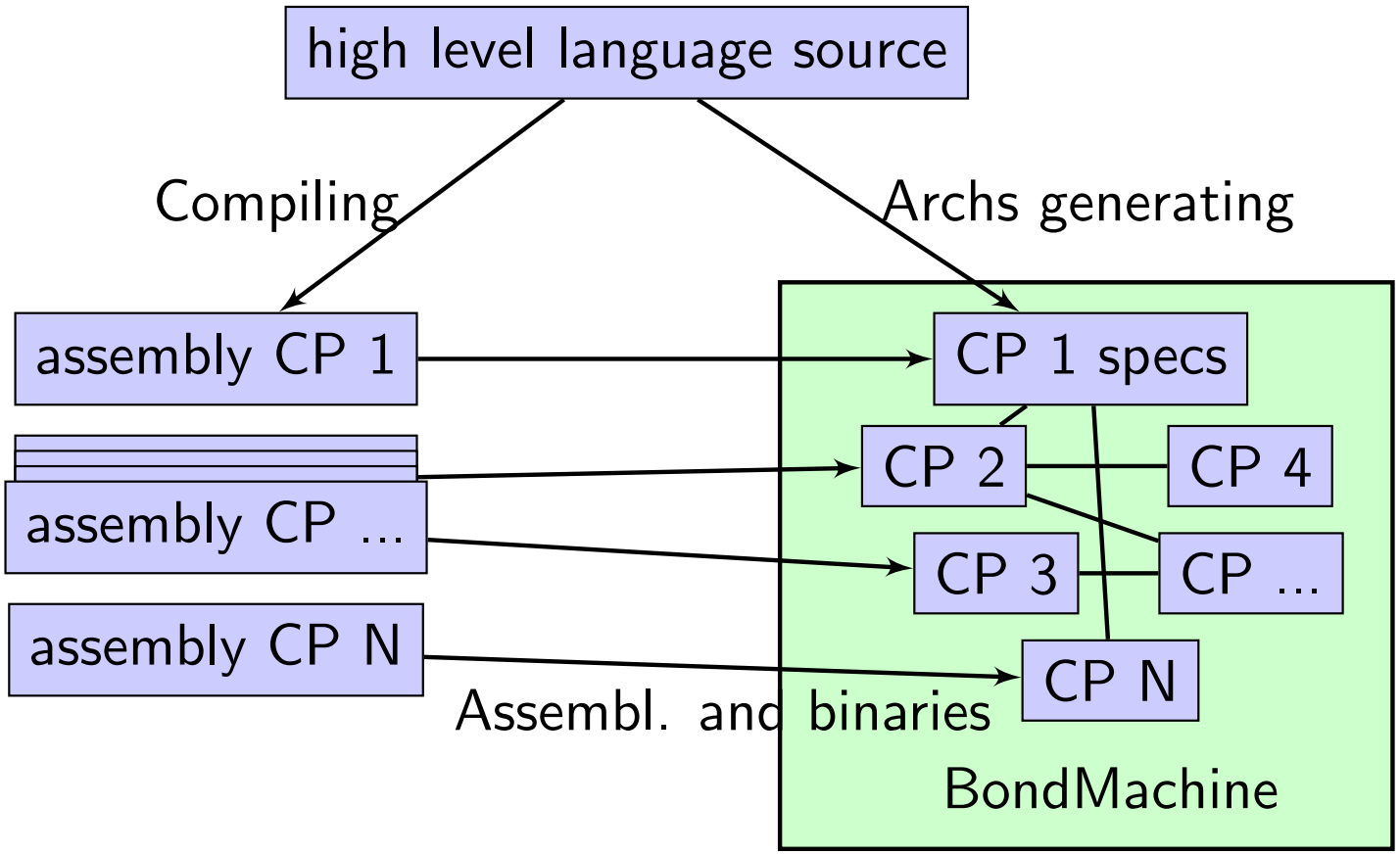
Bondgo

... it can do even much more interesting things when compiling concurrent programs.



Bondgo

... it can do even much more interesting things when compiling concurrent programs.



Bondgo

An example

bondgo stream processing example

```
package main

import (
   "bondgo"
)

func streamprocessor(a *[]uint8, b *[]uint8,
   c *[]uint8, gid uint8) {
   (*c)[gid] = (*a)[gid] + (*b)[gid]
}
func main() {
   a := make([]uint8, 256)
   b := make([]uint8, 256)
   c := make([]uint8, 256)

   // ... some a and b values fill

   for i := 0; i < 256; i++ {
      go streamprocessor(&a, &b, &c, uint8(i))
   }
}
```

The compilation of this example results in the creation of a 257 CPs where 256 are the stream processors executing the code in the function called *streamprocessor*, and one is the coordinating CP. Each stream processor is optimized and capable only to make additions since it is the only operation requested by the source code. The three slices created on the main function are passed by reference to the Goroutines then a shared RAM is created by the *Bondgo* compiler available to the generated CPs.

Compiling Architectures

One of the most important result

The architecture creation is a part of the compilation process.

Bondgo

Go in hardware

Bondgo implements a sort of "Go in hardware".

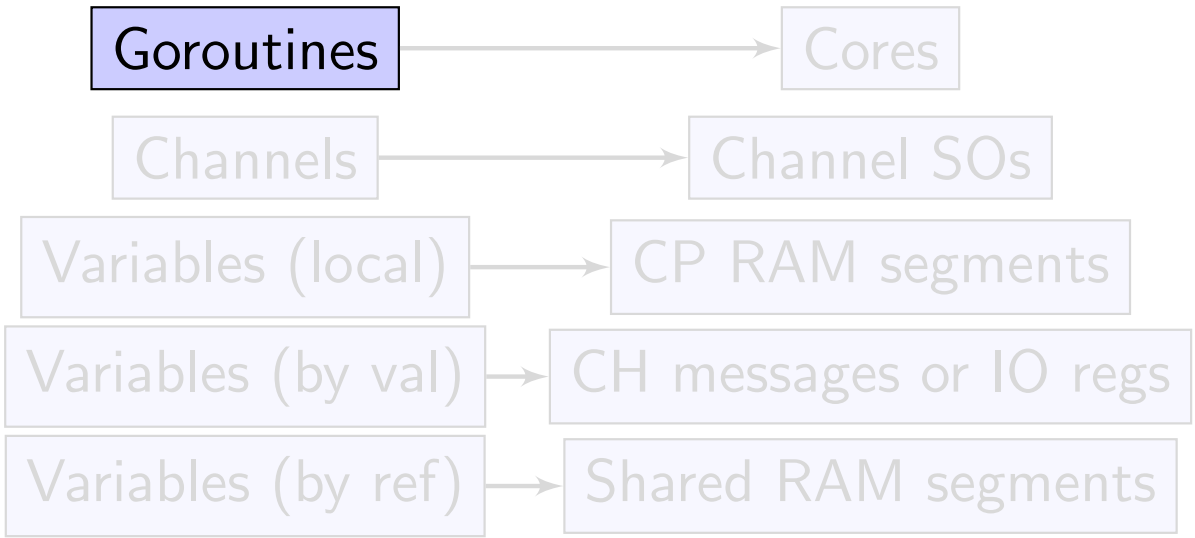
High level Go source code is directly mapped to interconnected processors without Operating Systems or runtimes.

Bondgo

Go in hardware

Bondgo implements a sort of "Go in hardware".

High level Go source code is directly mapped to interconnected processors without Operating Systems or runtimes.

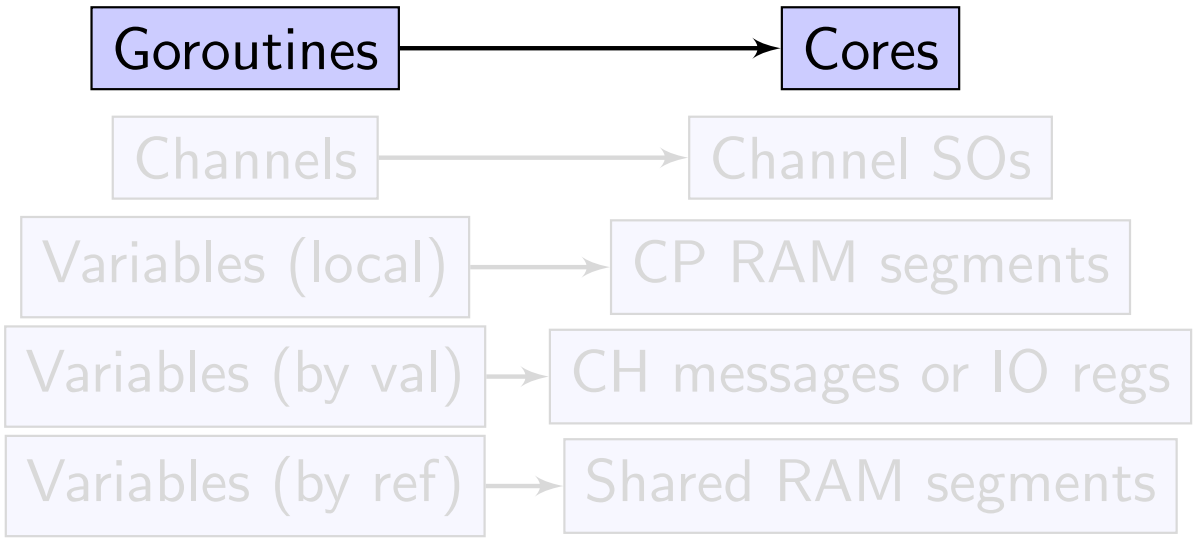


Bondgo

Go in hardware

Bondgo implements a sort of "Go in hardware".

High level Go source code is directly mapped to interconnected processors without Operating Systems or runtimes.

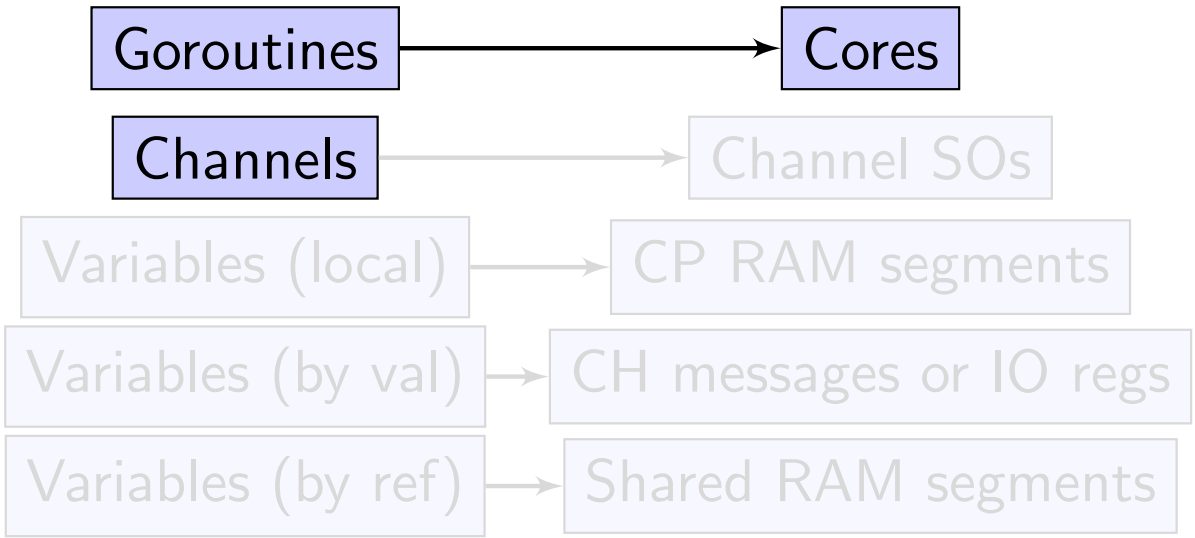


Bondgo

Go in hardware

Bondgo implements a sort of "Go in hardware".

High level Go source code is directly mapped to interconnected processors without Operating Systems or runtimes.

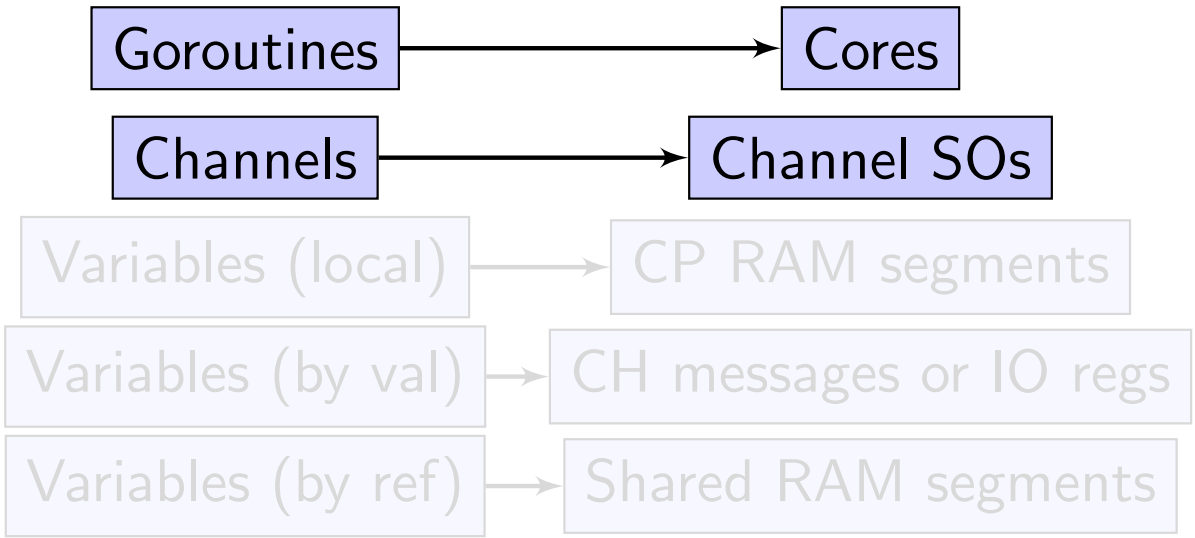


Bondgo

Go in hardware

Bondgo implements a sort of "Go in hardware".

High level Go source code is directly mapped to interconnected processors without Operating Systems or runtimes.

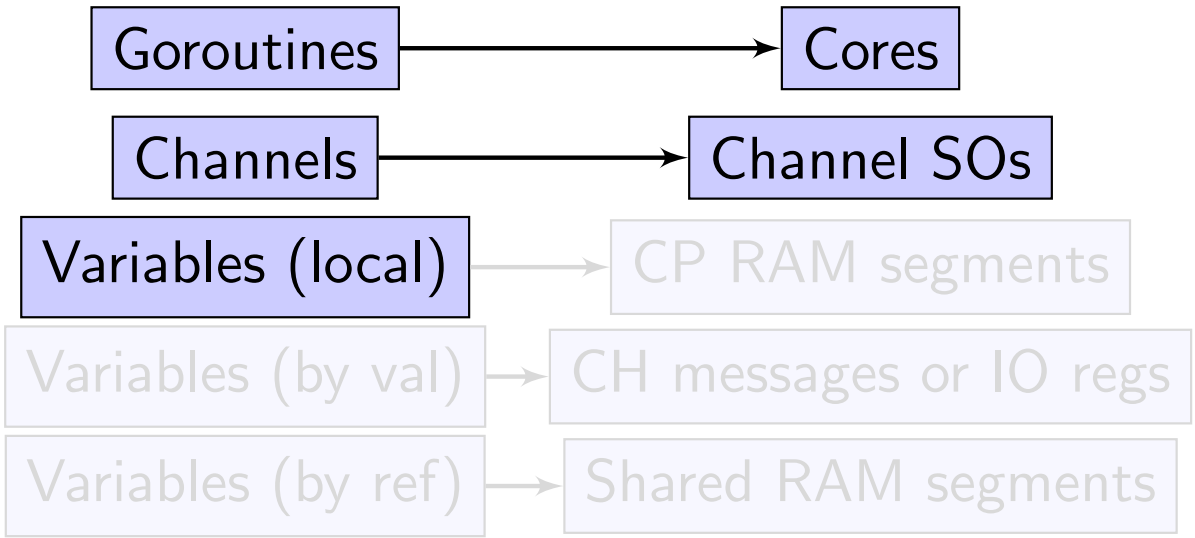


Bondgo

Go in hardware

Bondgo implements a sort of "Go in hardware".

High level Go source code is directly mapped to interconnected processors without Operating Systems or runtimes.

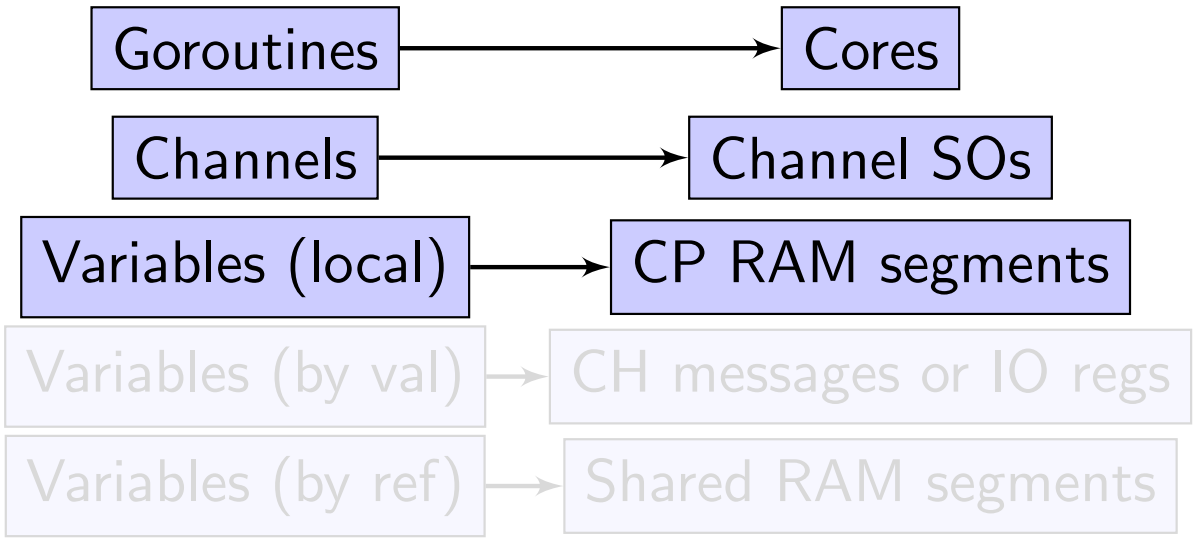


Bondgo

Go in hardware

Bondgo implements a sort of "Go in hardware".

High level Go source code is directly mapped to interconnected processors without Operating Systems or runtimes.

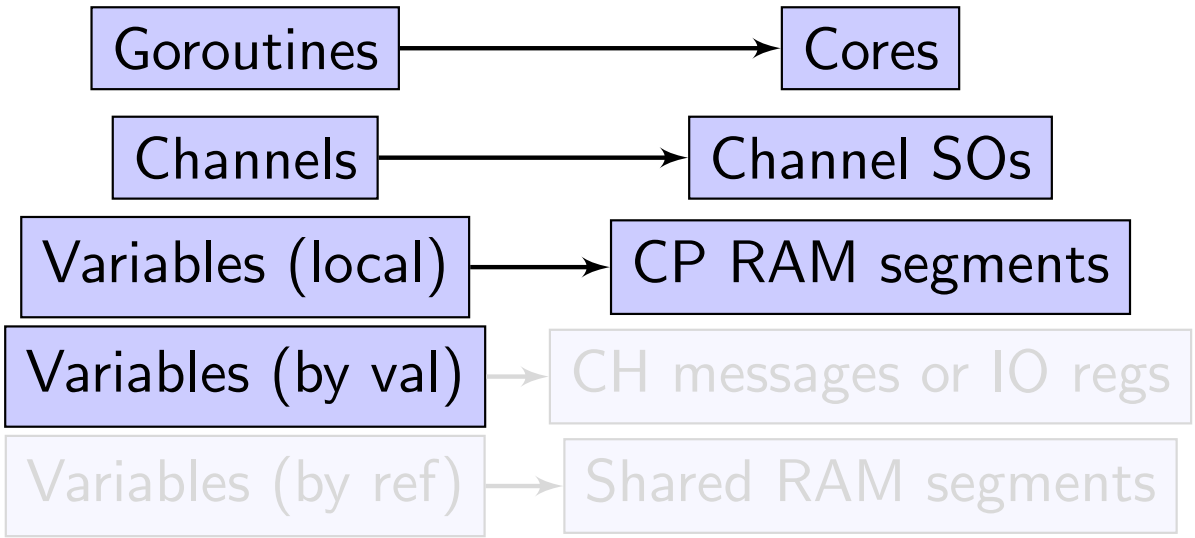


Bondgo

Go in hardware

Bondgo implements a sort of "Go in hardware".

High level Go source code is directly mapped to interconnected processors without Operating Systems or runtimes.

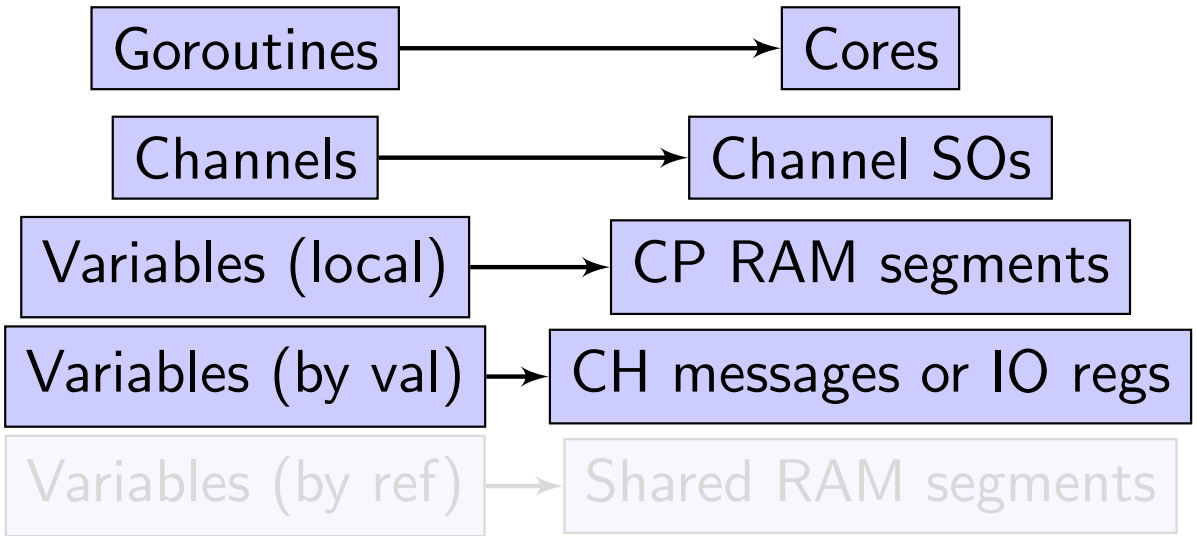


Bondgo

Go in hardware

Bondgo implements a sort of "Go in hardware".

High level Go source code is directly mapped to interconnected processors without Operating Systems or runtimes.

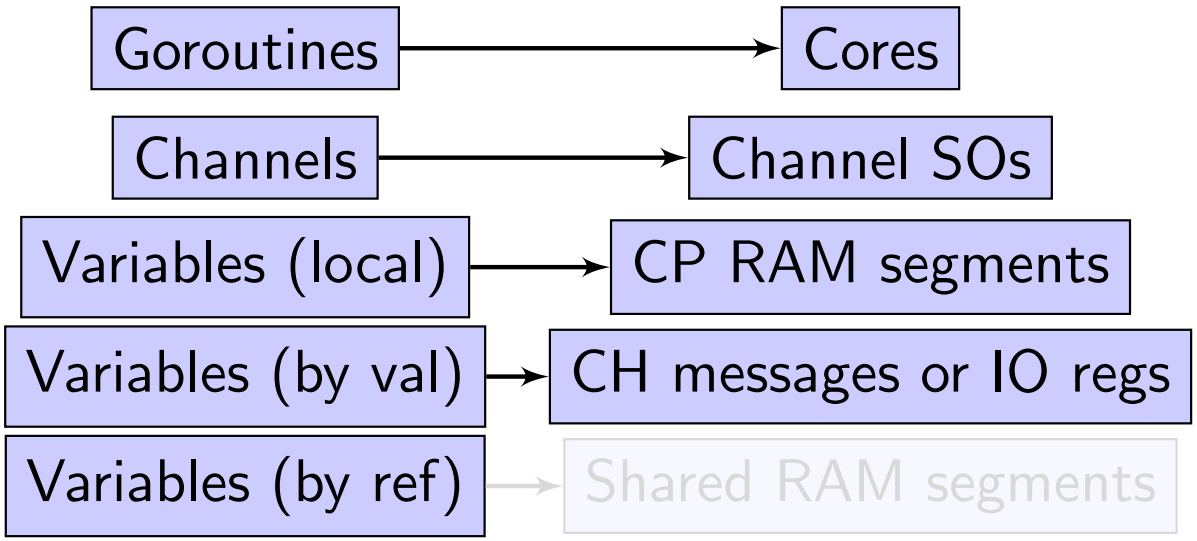


Bondgo

Go in hardware

Bondgo implements a sort of "Go in hardware".

High level Go source code is directly mapped to interconnected processors without Operating Systems or runtimes.

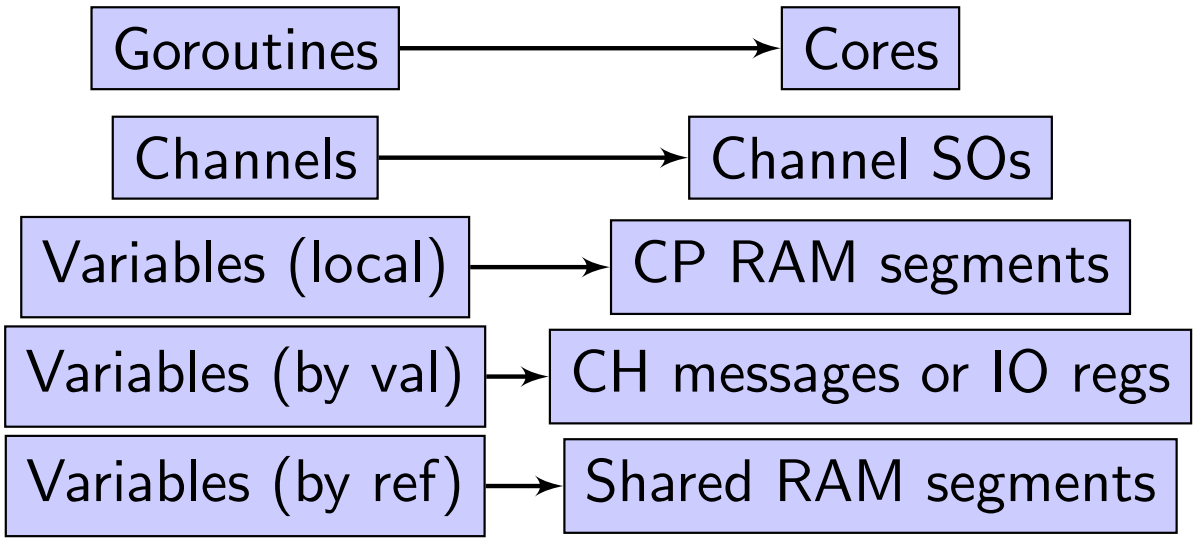


Bondgo

Go in hardware

Bondgo implements a sort of "Go in hardware".

High level Go source code is directly mapped to interconnected processors without Operating Systems or runtimes.



Go in hardware

Second idea on the BondMachine

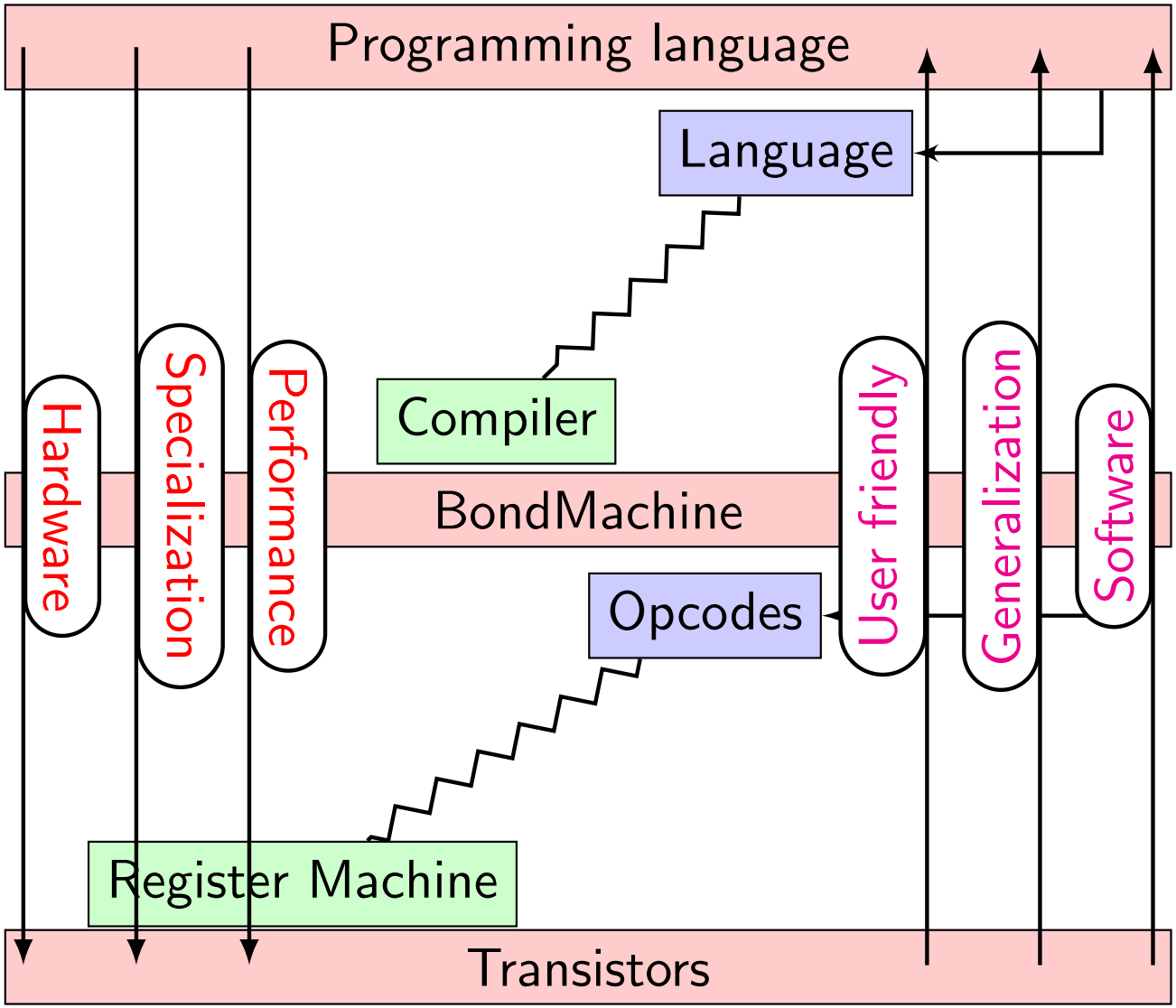
The idea was:
Build a computing system with a decreased number of layers resulting in a lower HW/SW gap.

This would raise the overall performances yet keeping an user friendly way of programming.

Between HW and SW there is only the processor abstraction, no Operating System nor runtimes. Despite that programming is done at high level.

Layers, Abstractions and Interfaces

and BondMachines



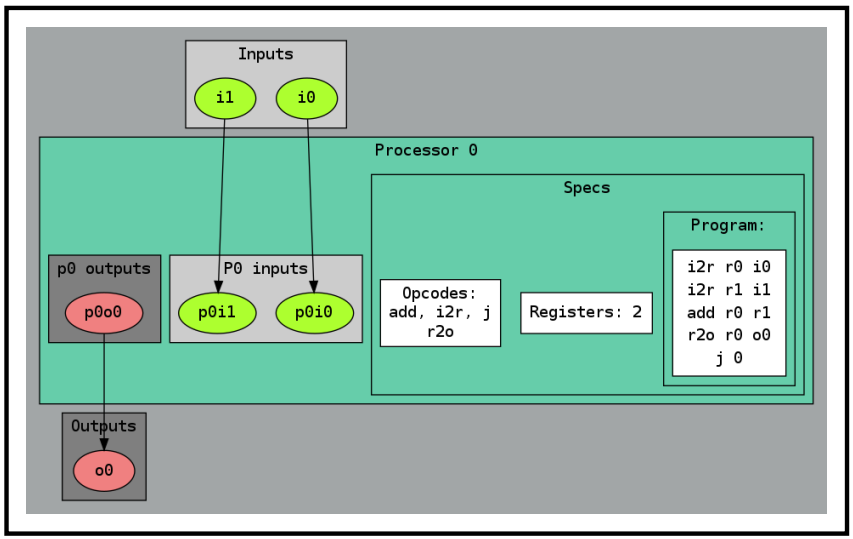
Bondgo

Abstract Assembly

The Assembly language for the BM has been kept as independent as possible from the particular CP.

Given a specific piece of assembly code Bondgo has the ability to compute the "minimum CP" that can execute that code.

```
i2r r0 i0
i2r r1 i1
add r0 r1
r2o r0 o0
  j 0
```



These are Building Blocks for complex BondMachines.

Builders API

With these Building Blocks

Several libraries have to developed to map specific problems on BondMachines:

- Symbond, to handle mathematical expression.
- Boolbond, to map boolean expression.
- Matrixwork, to perform matrices operations.
- Neuralbond, to use neural networks.

Builders API

With these Building Blocks

Several libraries have to developed to map specific problems on BondMachines:

- Symbond, to handle mathematical expression.
- Boolbond, to map boolean expression.
- Matrixwork, to perform matrices operations.
- Neuralbond, to use neural networks.

Builders API

With these Building Blocks

Several libraries have to developed to map specific problems on BondMachines:

- Symbond, to handle mathematical expression.
- Boolbond, to map boolean expression.
- Matrixwork, to perform matrices operations.
- Neuralbond, to use neural networks.

Builders API

With these Building Blocks

Several libraries have to developed to map specific problems on BondMachines:

- Symbond, to handle mathematical expression.
- Boolbond, to map boolean expression.
- Matrixwork, to perform matrices operations.
- Neuralbond, to use neural networks.

Builders API

With these Building Blocks

Several libraries have to developed to map specific problems on BondMachines:

- Symbond, to handle mathematical expression.
- Boolbond, to map boolean expression.
- Matrixwork, to perform matrices operations.
- Neuralbond, to use neural networks.

Builders API

Symbond

A mathematical expression, or a system can be converted to a BondMachine:

sum(var(x),const(2))

Boolbond

```
symbond -expression "sum(var(x),const(2))"
-save-bondmachine bondmachine.json
```

Resulting in:

Builders API

Symbond

A mathematical expression, or a system can be converted to a BondMachine:

sum(var(x),const(2))

Boolbond

```
symbond -expression "sum(var(x),const(2))"
-save-bondmachine bondmachine.json
```

Resulting in:

Builders API

Symbond

A mathematical expression, or a system can be converted to a BondMachine:

sum(var(x),const(2))

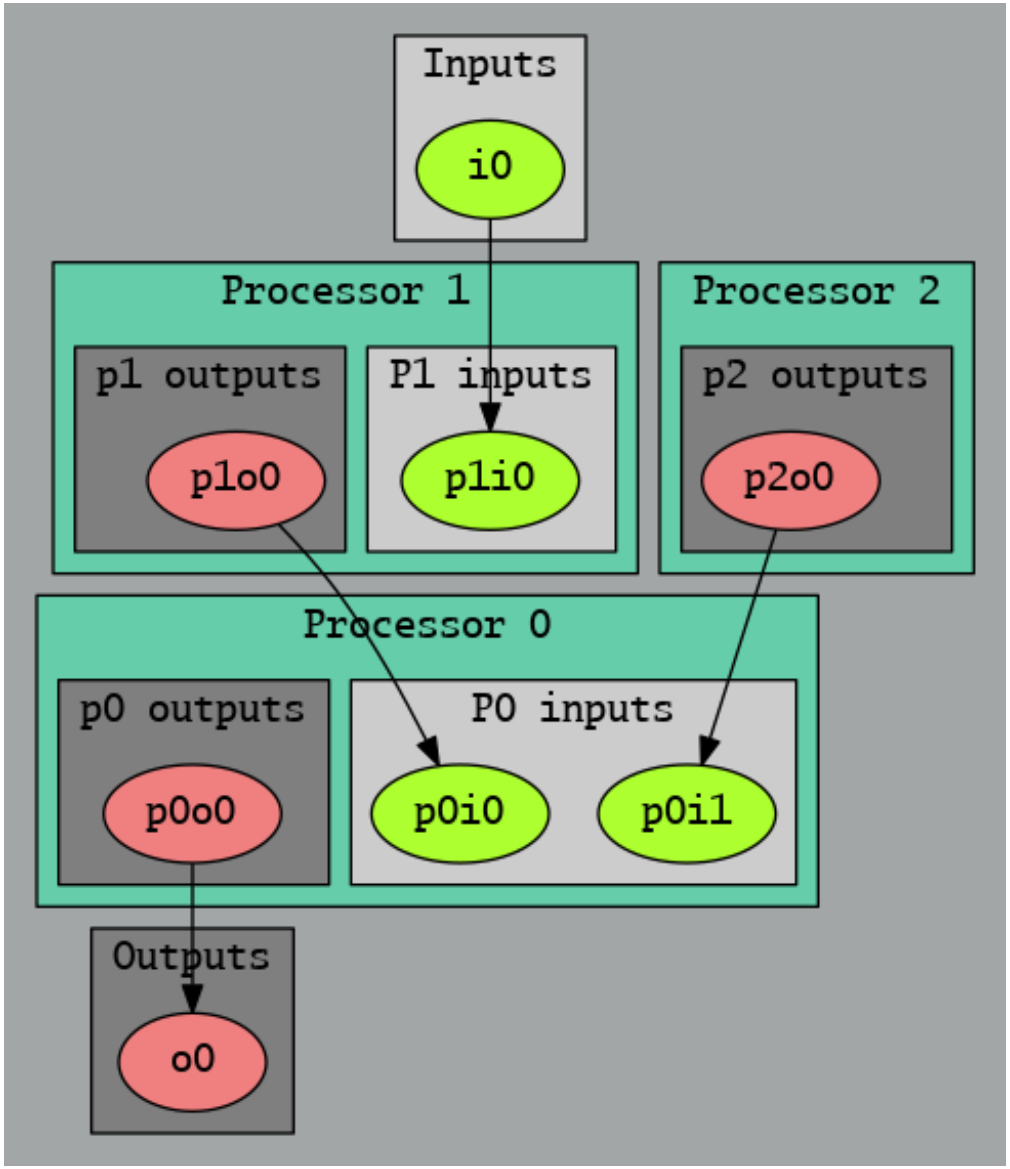
Boolbond

symbond -expression "sum(var(x),const(2))"
-save-bondmachine bondmachine.json

Resulting in:

Builders API

Symbond



Builders API

Boolbond

A system of boolean equations, input and output variables are expressed as in the example file:

```
var(z)=or(var(x),not(var(y)))
var(t)=or(and(var(x),var(y)),var(z))
var(l)=and(xor(var(x),var(y)),var(t))
i:var(x)
i:var(y)
o:var(z)
o:var(t)
o:var(l)
```

Boolbond

```
boolbond -system-file expression.txt -save-bondmachine
bondmachine.json
```

Resulting in:

Builders API

Boolbond

A system of boolean equations, input and output variables are expressed as in the example file:

```
var(z)=or(var(x),not(var(y)))
var(t)=or(and(var(x),var(y)),var(z))
var(l)=and(xor(var(x),var(y)),var(t))
i:var(x)
i:var(y)
o:var(z)
o:var(t)
o:var(l)
```

Boolbond

```
boolbond -system-file expression.txt -save-bondmachine
bondmachine.json
```

Resulting in:

Builders API

Boolbond

A system of boolean equations, input and output variables are expressed as in the example file:

```
var(z)=or(var(x),not(var(y)))
var(t)=or(and(var(x),var(y)),var(z))
var(l)=and(xor(var(x),var(y)),var(t))
i:var(x)
i:var(y)
o:var(z)
o:var(t)
o:var(l)
```

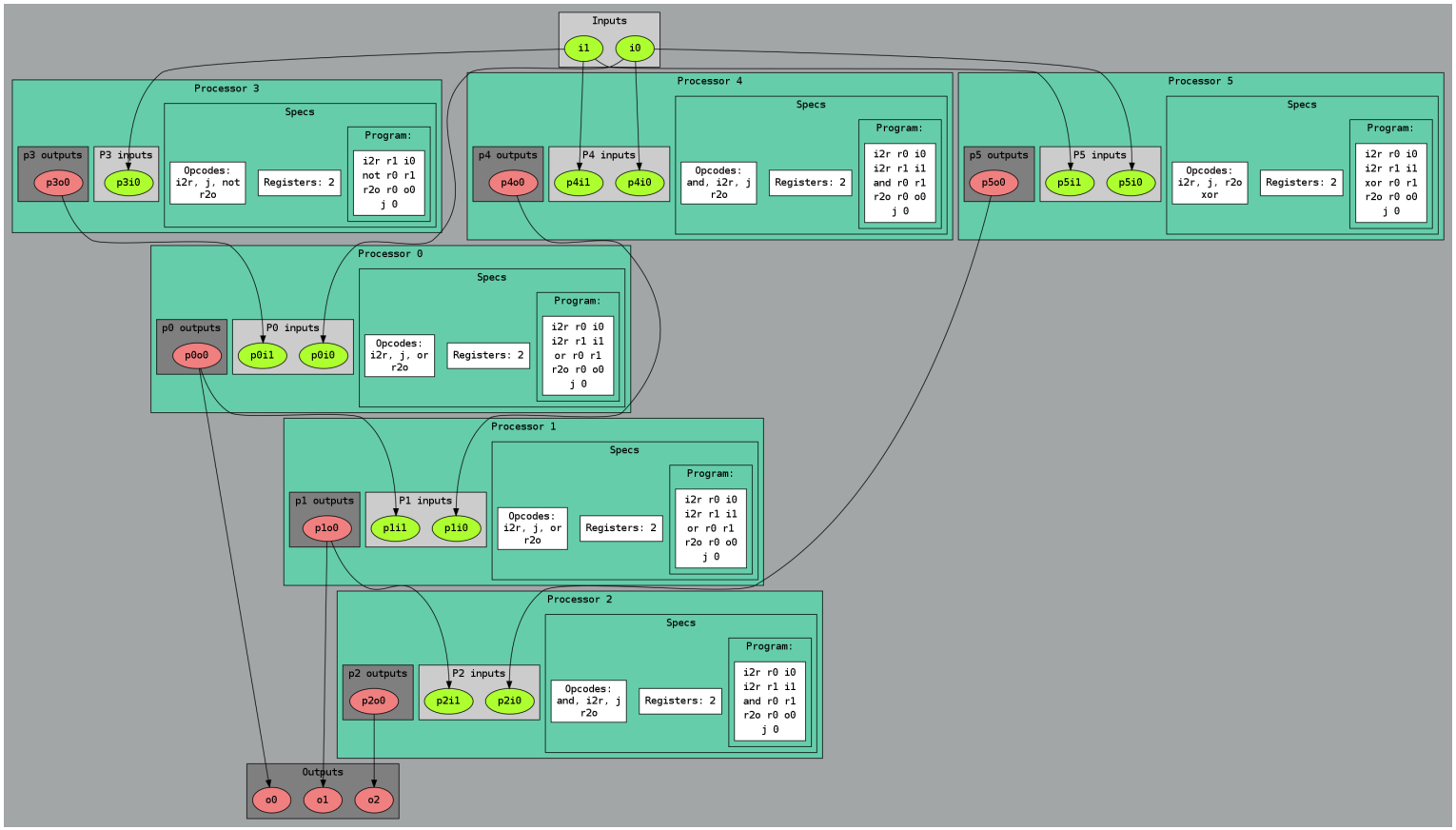
Boolbond

```
boolbond -system-file expression.txt -save-bondmachine bondmachine.json
```

Resulting in:

Builders API

Boolbond

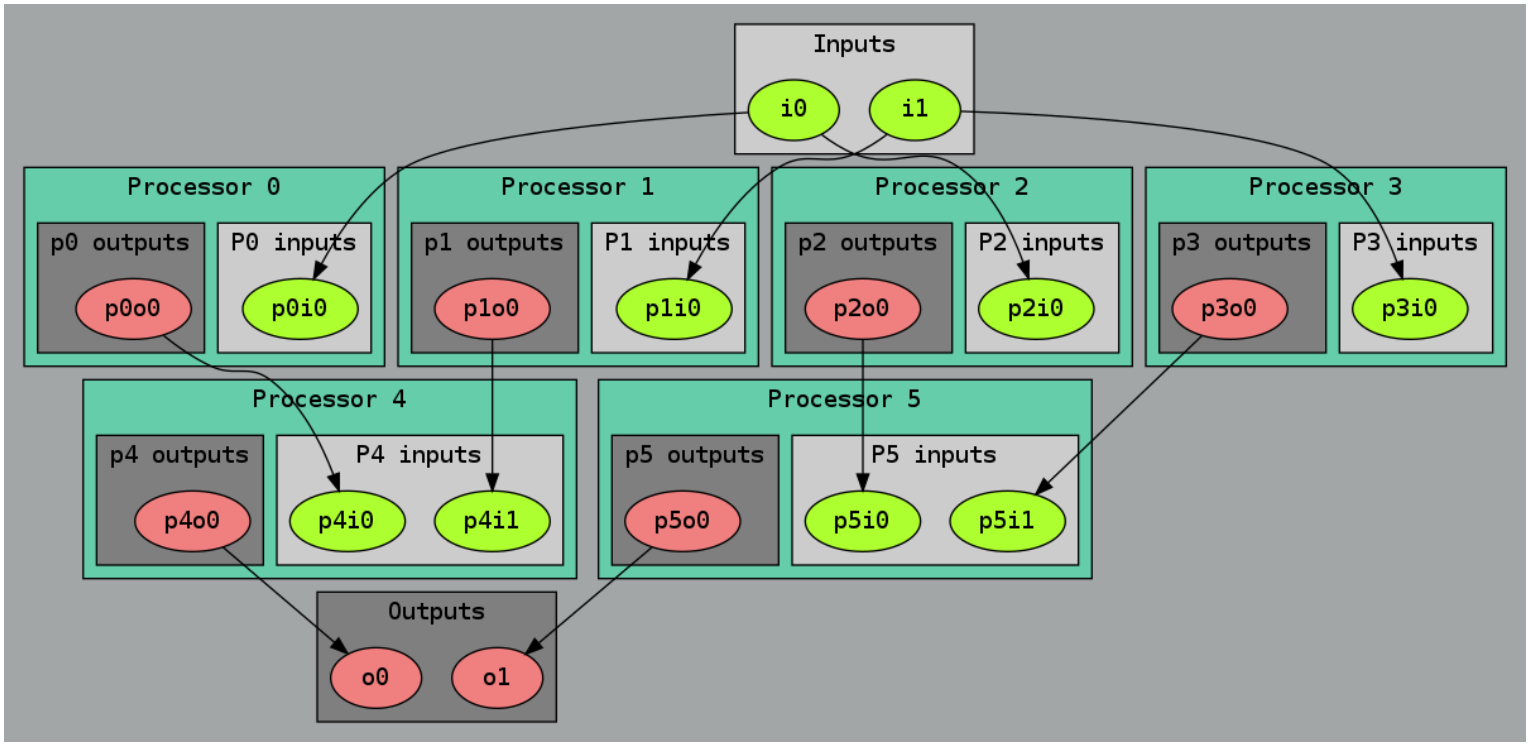


Builders API

Matrixwork

Matrix multiplication

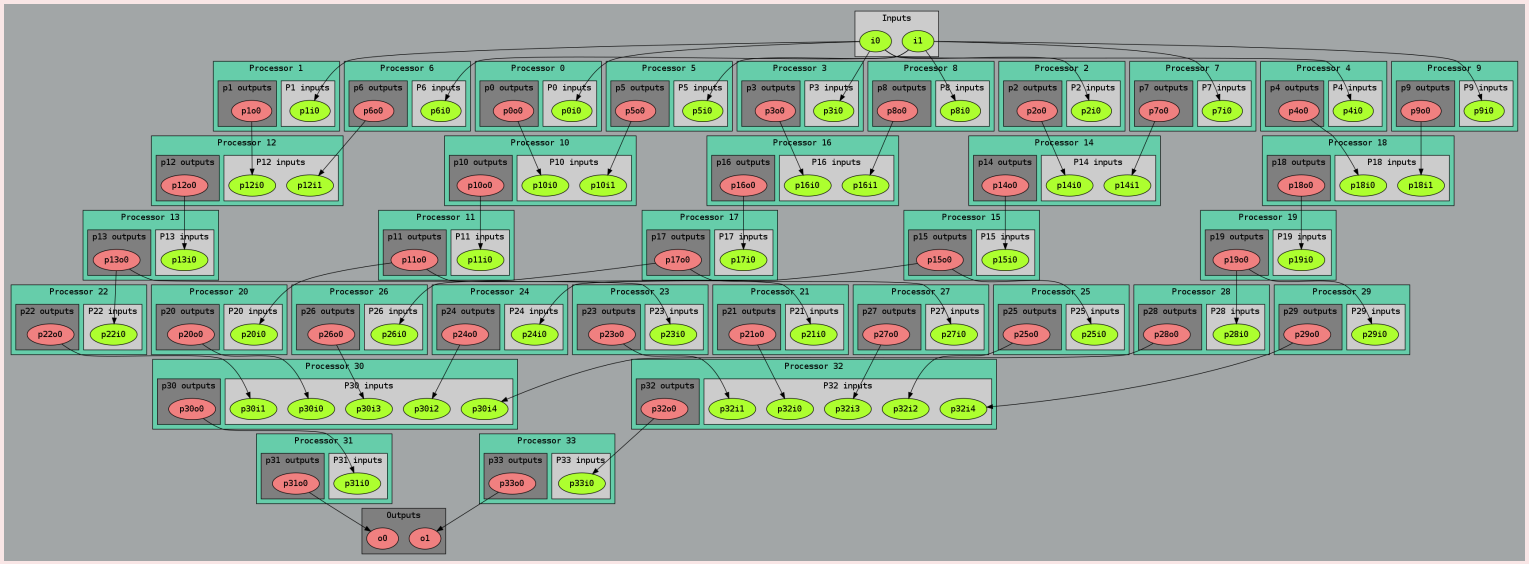
if mymachine, ok := matrixwork.Build_M(n, t); ok == nil ...



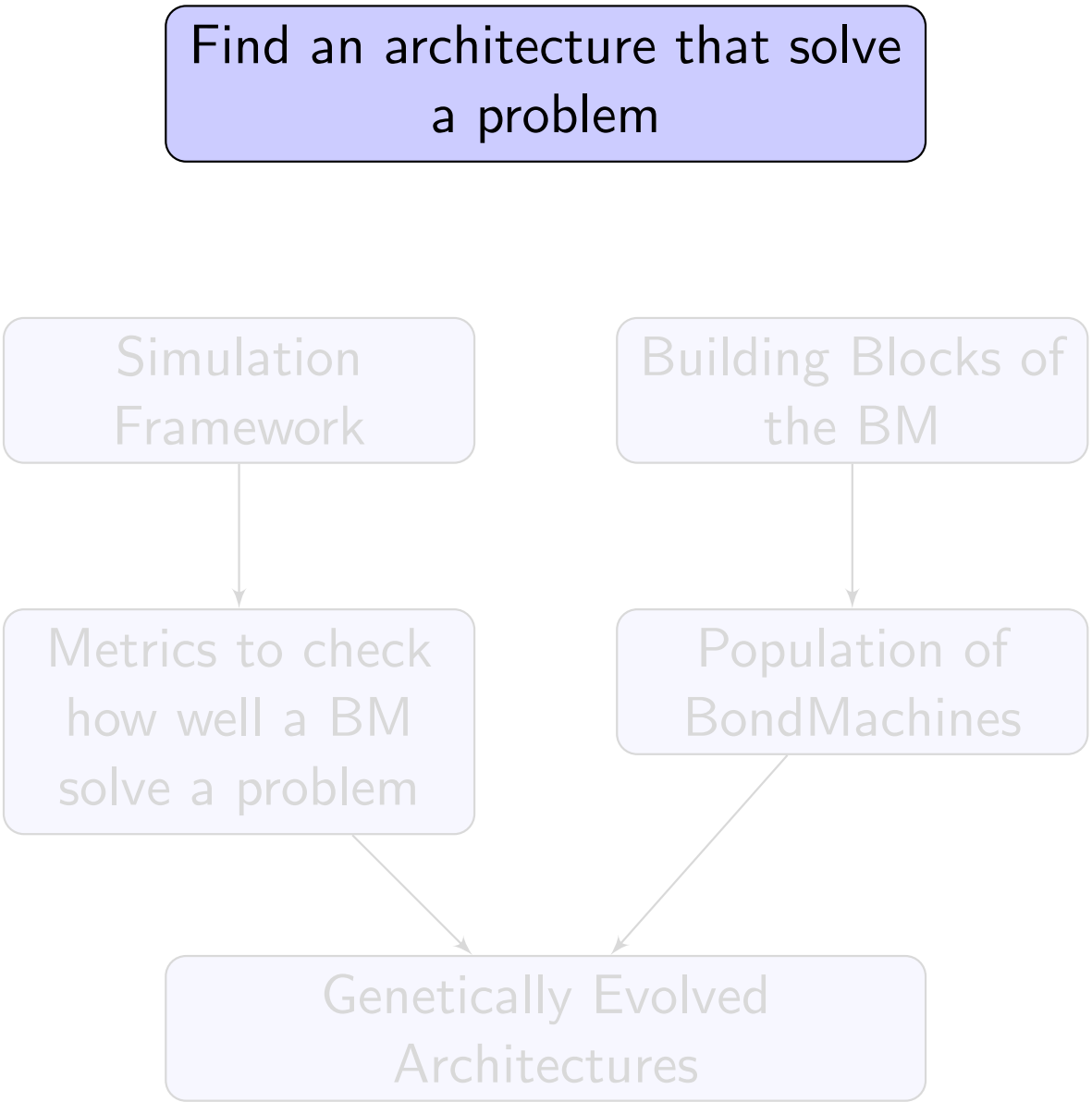
Builders API

Neuralbond

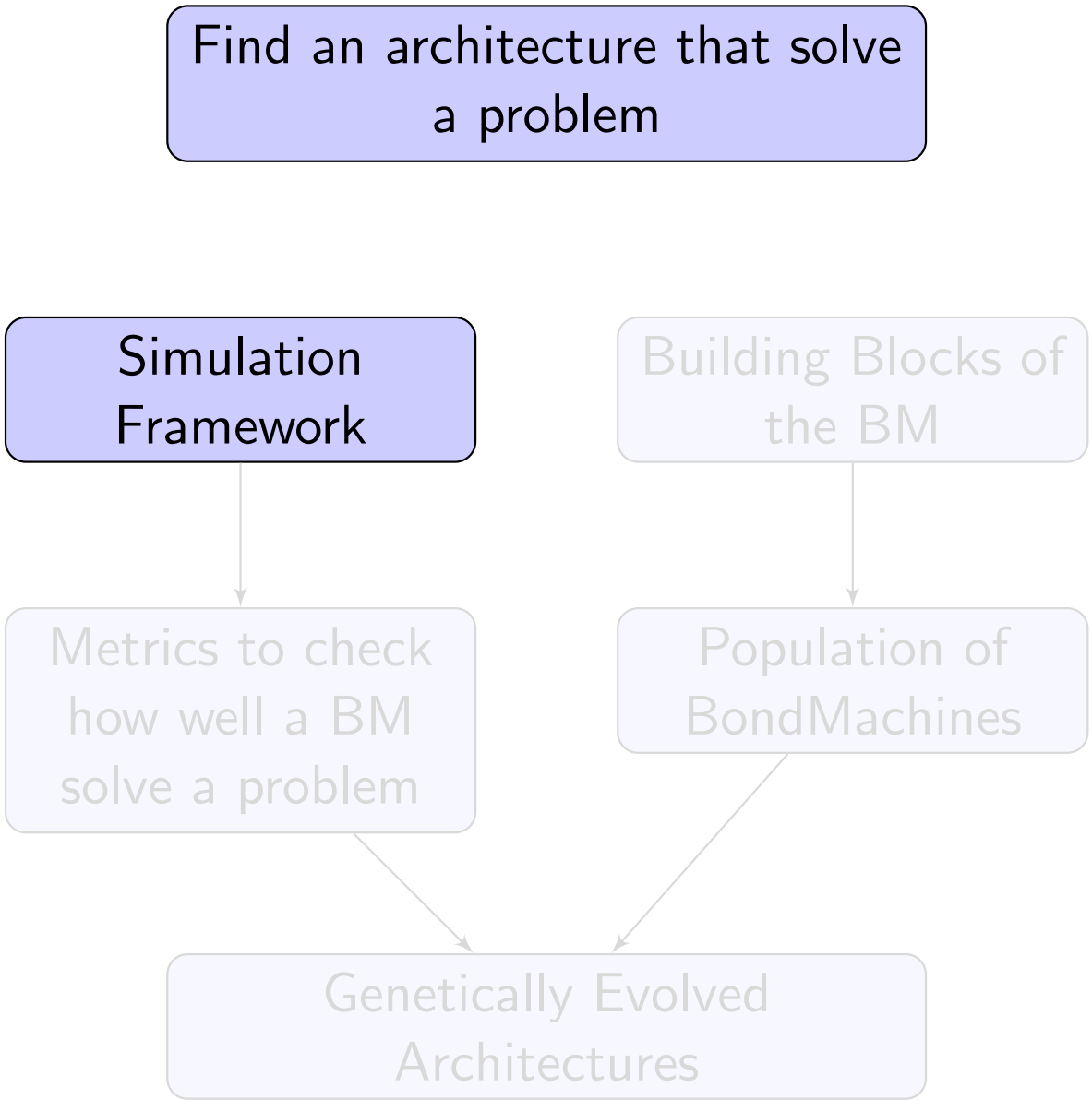
3 Layer neural network



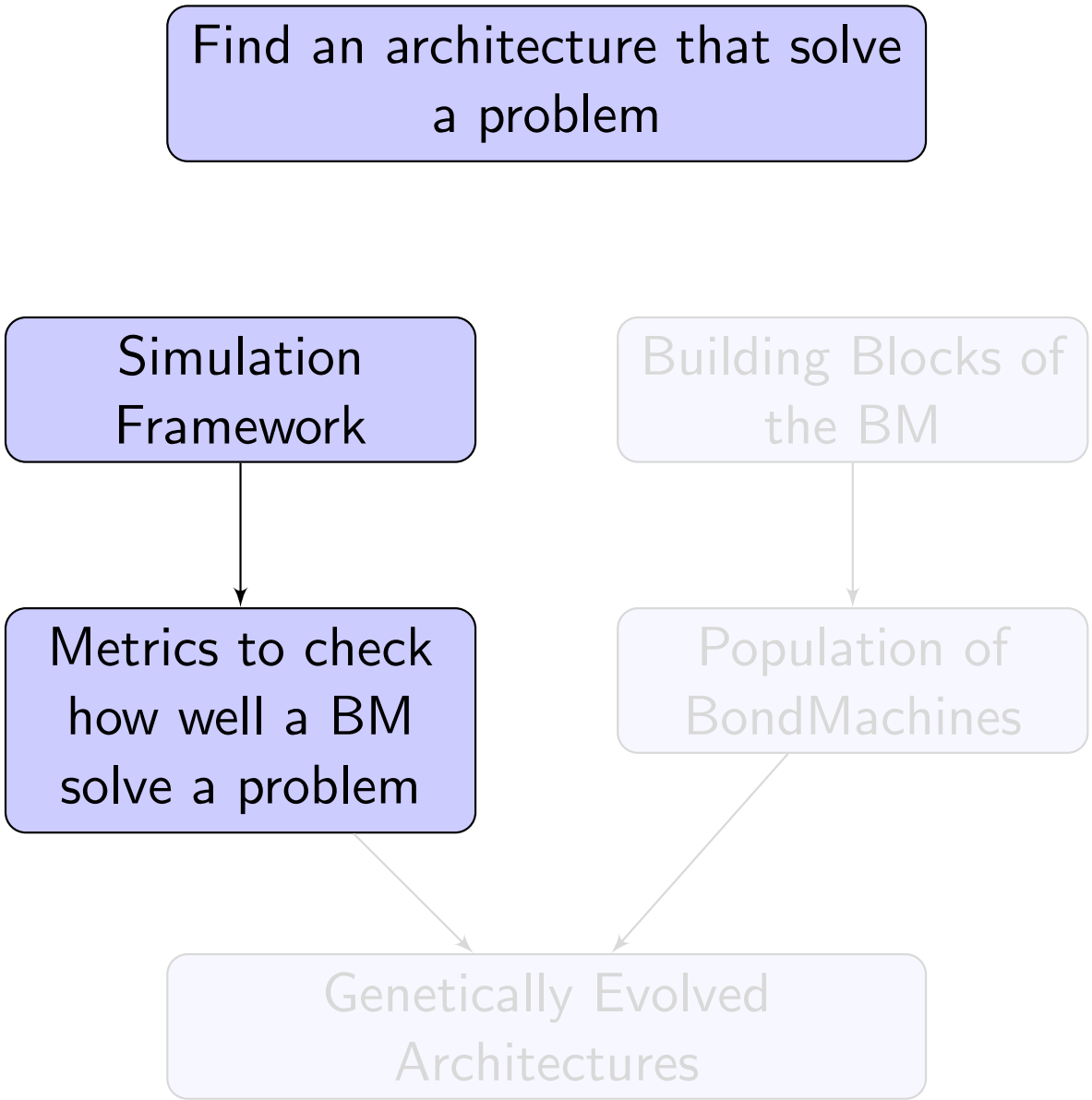
Evolutionary BondMachine



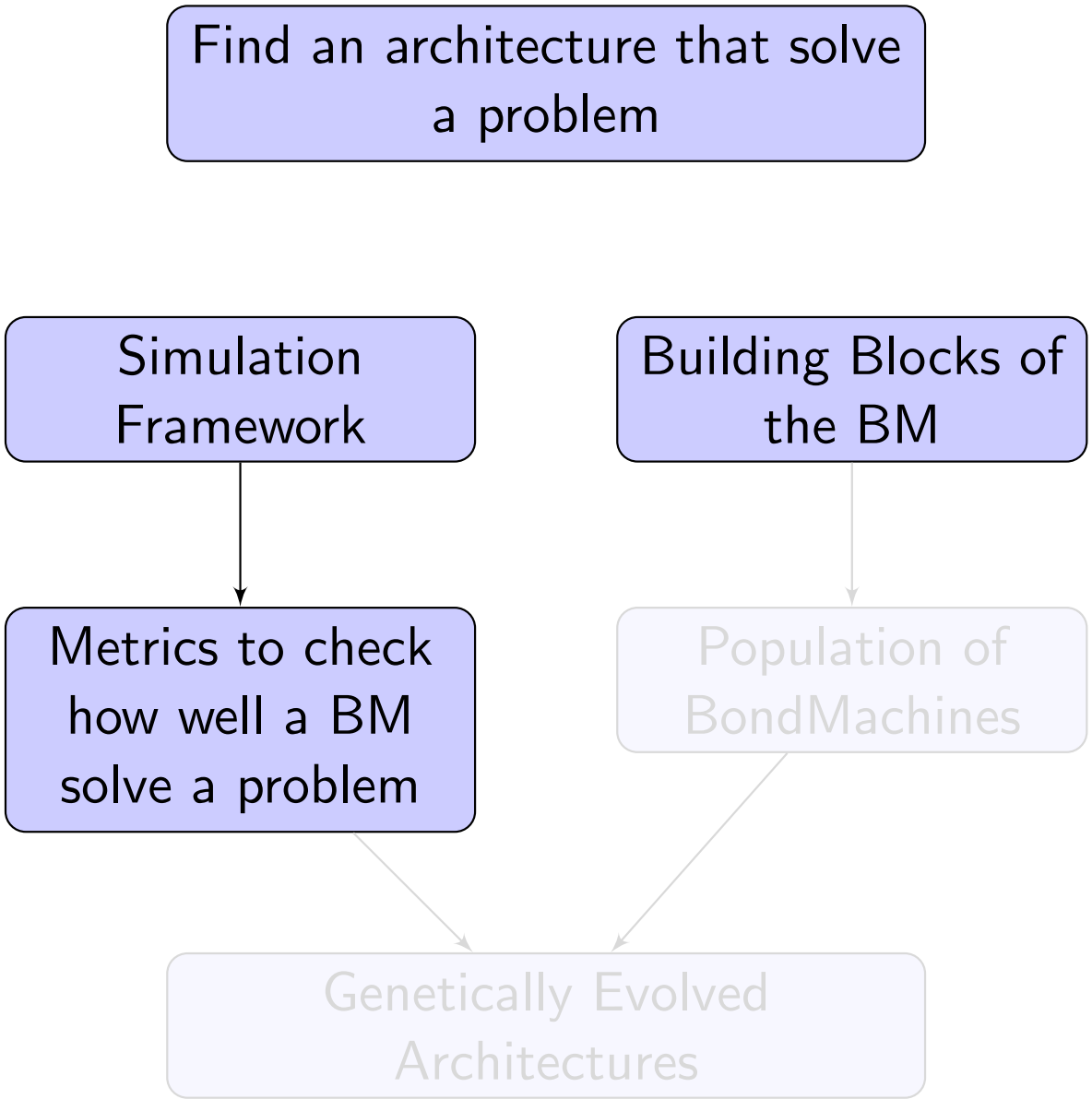
Evolutionary BondMachine



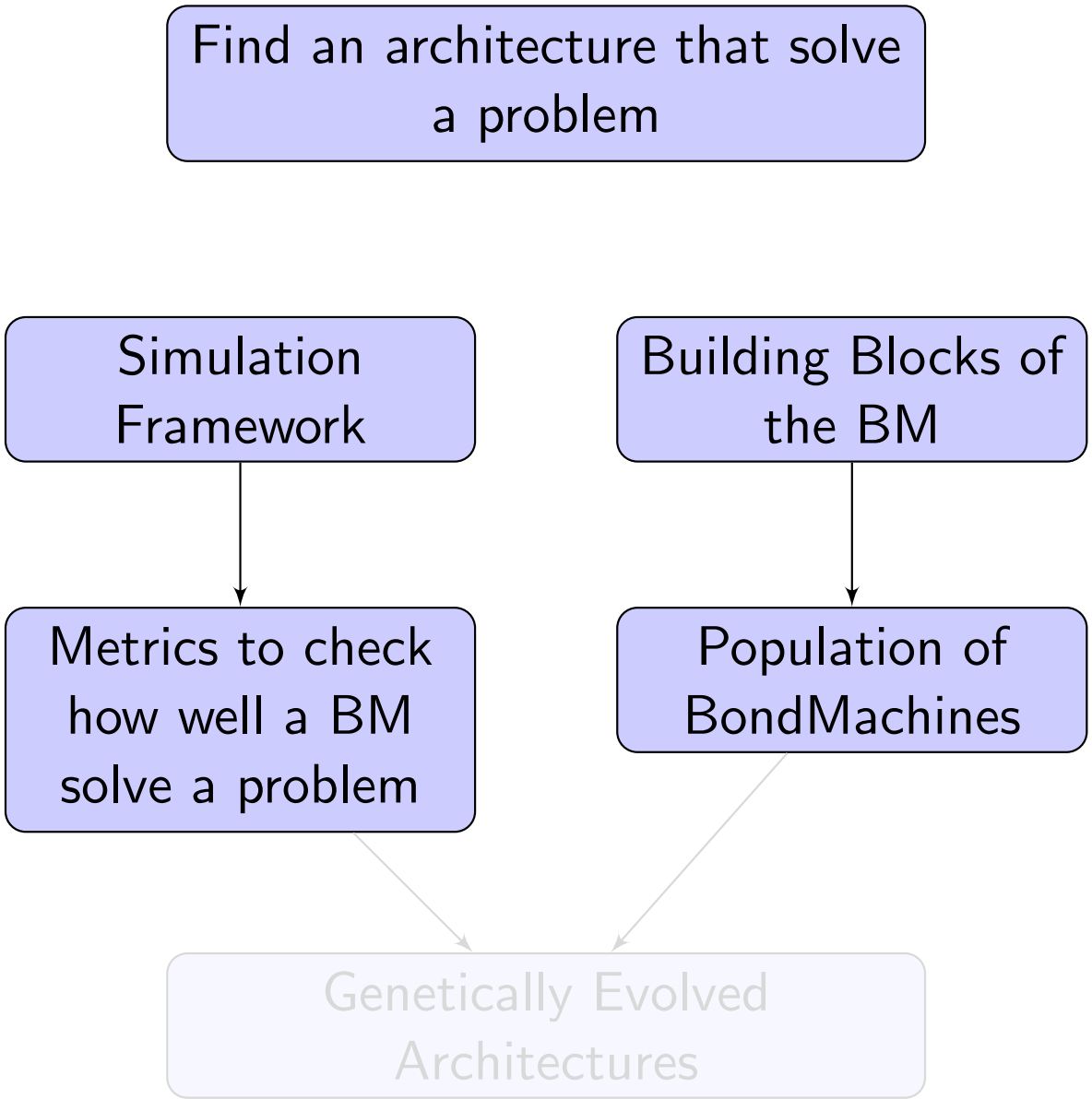
Evolutionary BondMachine



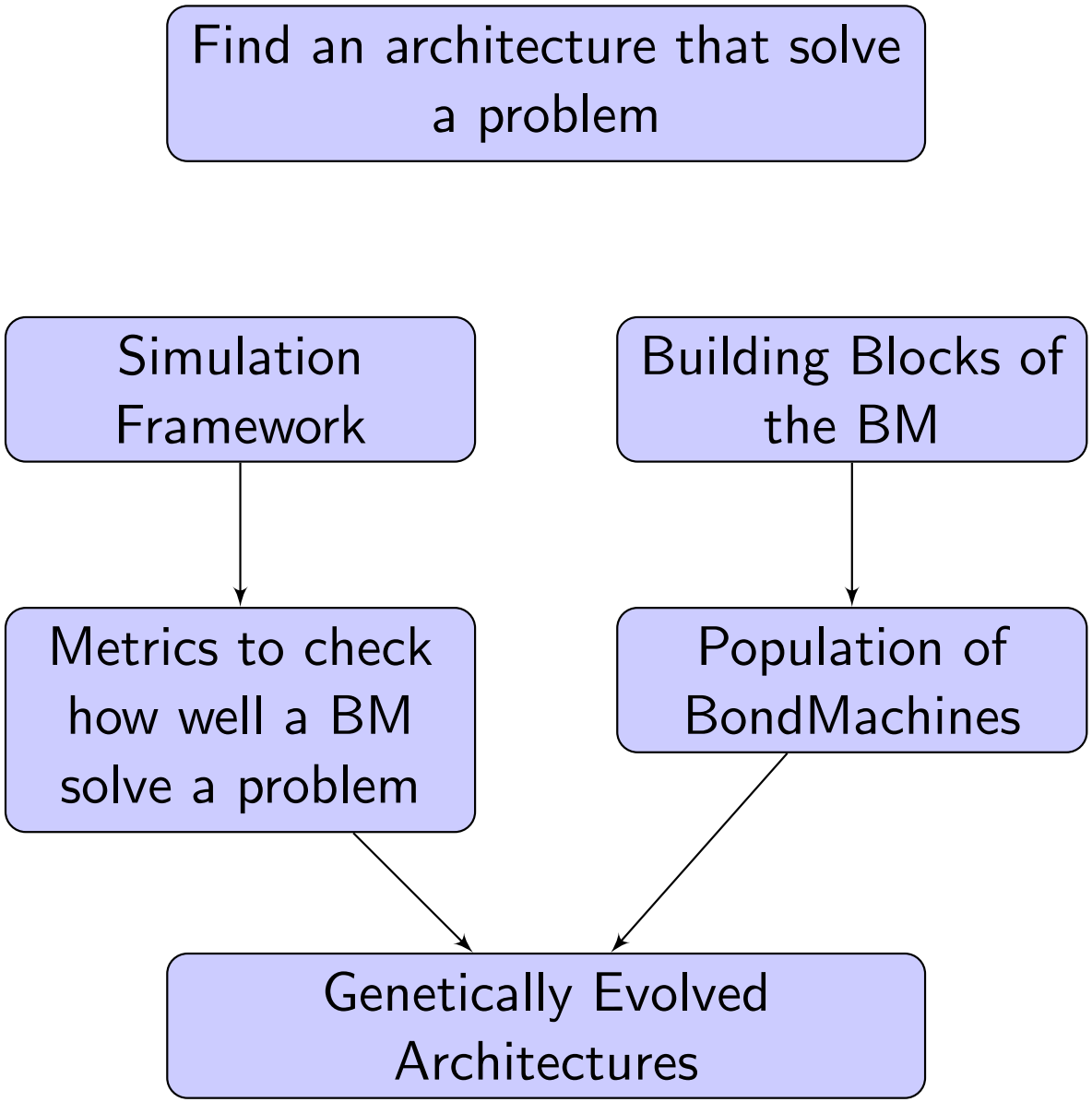
Evolutionary BondMachine



Evolutionary BondMachine



Evolutionary BondMachine

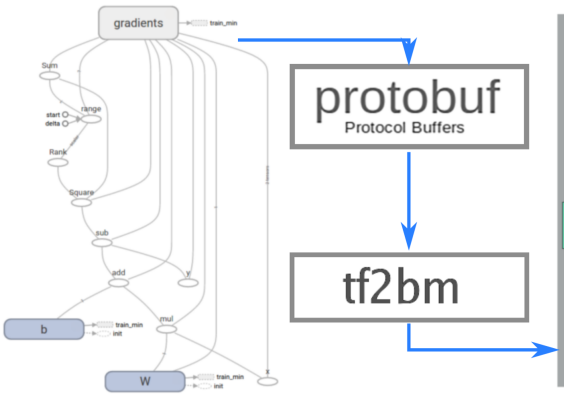


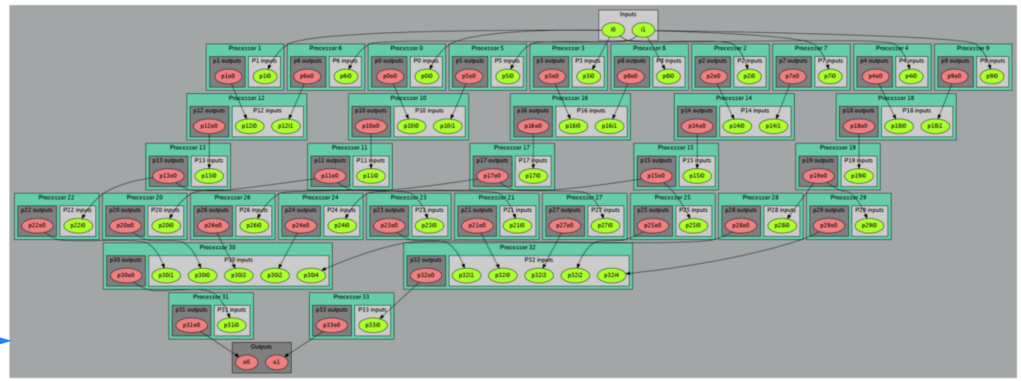
TensorFlow™ to Bondmachine

tf2bm

TensorFlow™ is an open source software library for numerical computation using data flow graphs.

Graphs can be converted to BondMachines with the tf2bm tool.





Hardware implementation

FPGA

The RTL code for the BondMachine is written in Verilog and System Verilog, and has been tested on these devices/system:

- Digilent Basys3 - Xilinx Artix-7 - Vivado.
- Kintex7 Evaluation Board - Vivado.
- Digilent Zedboard - Xilinx Zynq 7020 - Vivado.
- Linux - Iverilog.

Within the project other firmwares have been written or tested:

- Microchip ENC28J60 Ethernet interface controller.
- Microchip ENC424J600 10/100 Base-T Ethernet interface controller.
- ESP8266 Wi-Fi chip.

Toolchains

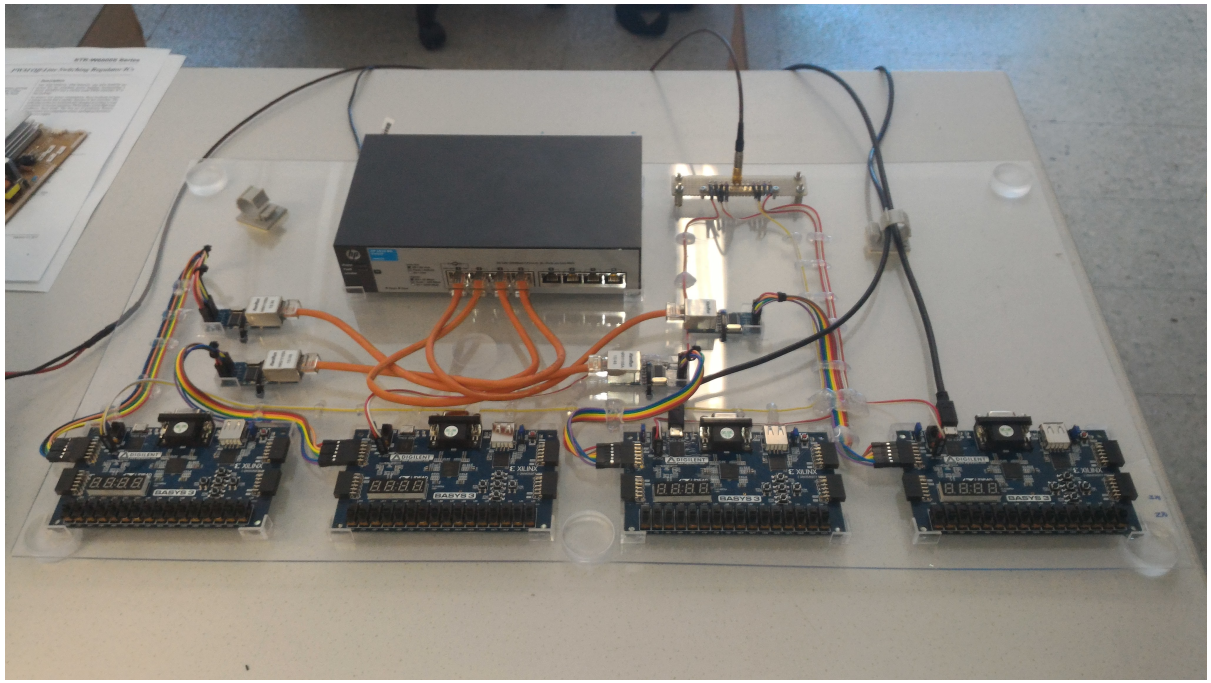
A set of toolchains allow the build and the direct deploy to a target device of BondMachines.

Bondgo Toolchain example

A file local.mk contains references to the source code as well all the build necessities.
make bondmachine creates the JSON representation of the BM and assemble its code.
make show displays a graphical representation of the BM.
make simulate start a simulation.
make videosim create a simulation video.
make flash the device into the destination target.

The Prototype

The project has been selected for the participation at MakerFaire 2016 Rome (The Europen Edition) and a prototype has been assembled and presented.



First run:

`https://youtube.com/embed/hukTrGxTb7A`

The BondMachine computational Ecosystem

So far we saw:

- An user friendly approach to create processors (single core).
- Optimizing a single device to support intricate computational work-flows (multi-cores).

Interconnected BondMachines

What if we could extend the same ideas to multiple interconnected devices ?

The BondMachine computational Ecosystem

So far we saw:

- An user friendly approach to create processors (single core).
- Optimizing a single device to support intricate computational work-flows (multi-cores).

Interconnected BondMachines

What if we could extend the same ideas to multiple interconnected devices ?

The BondMachine computational Ecosystem

So far we saw:

- An user friendly approach to create processors (single core).
- Optimizing a single device to support intricate computational work-flows (multi-cores).

Interconnected BondMachines

What if we could extend the same ideas to multiple interconnected devices ?

The BondMachine computational Ecosystem

So far we saw:

- An user friendly approach to create processors (single core).
- Optimizing a single device to support intricate computational work-flows (multi-cores).

Interconnected BondMachines

What if we could extend the same ideas to multiple interconnected devices ?

The BondMachine computational Ecosystem

The same logic existing among CP have been extended among different BondMachines organized in clusters.

Protocols, one ethernet called *etherbond* and one using UDP called *udpbond* have been created for the purpose.

FPGA based BondMachines, standard Linux Workstations, Emulated BondMachines may join a cluster an contribute to a single distributed computational problem.

The BondMachine computational Ecosystem

The same logic existing among CP have been extended among different BondMachines organized in clusters.

Protocols, one ethernet called *etherbond* and one using UDP called *udpbond* have been created for the purpose.

FPGA based BondMachines, standard Linux Workstations, Emulated BondMachines may join a cluster an contribute to a single distributed computational problem.

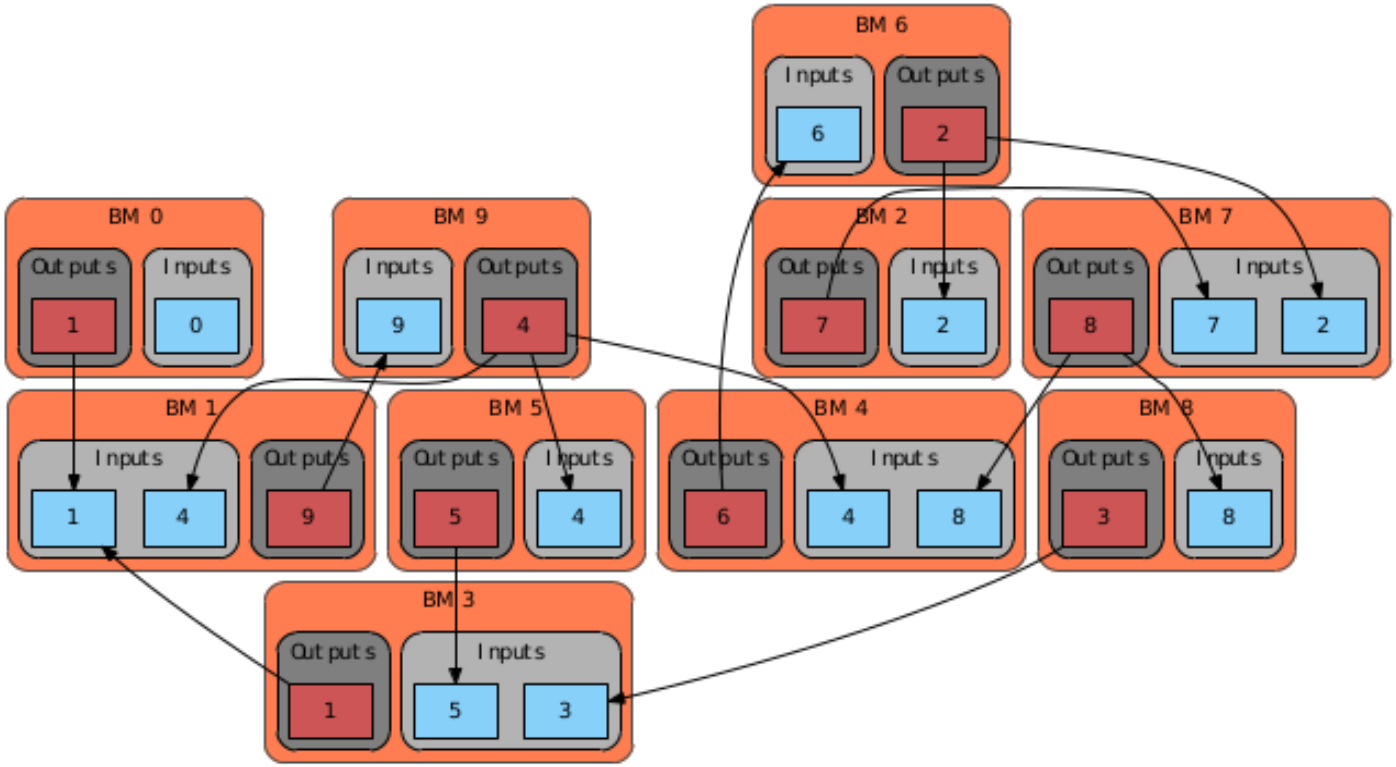
The BondMachine computational Ecosystem

The same logic existing among CP have been extended among different BondMachines organized in clusters.

Protocols, one ethernet called *etherbond* and one using UDP called *udpbond* have been created for the purpose.

FPGA based BondMachines, standard Linux Workstations, Emulated BondMachines may join a cluster an contribute to a single distributed computational problem.

The Ecosystem



The Ecosystem

A complete example

distributed counter

```
package main

import (
   "bondgo"
)

func pong() {
   var in0 bondgo.Input
   var out0 bondgo.Output
   in0 = bondgo.Make(bondgo.Input, 3)
   out0 = bondgo.Make(bondgo.Output, 5)
   for {
      bondgo.IOWrite(out0, bondgo.IORead(in0)+1)
   }
   bondgo.Void(in0)
   bondgo.Void(out0)
}
func main() {
   var in0 bondgo.Input
   var out0 bondgo.Output
   in0 = bondgo.Make(bondgo.Input, 5)
   out0 = bondgo.Make(bondgo.Output, 3)
device_0:
   go pong()
   for {
      bondgo.IOWrite(out0, bondgo.IORead(in0))
   }
   bondgo.Void(in0)
   bondgo.Void(out0)
}
```

The Ecosystem

A complete example

Compiling the code with the bondgo compiler:

bondgo -input-file ds.go -mpm

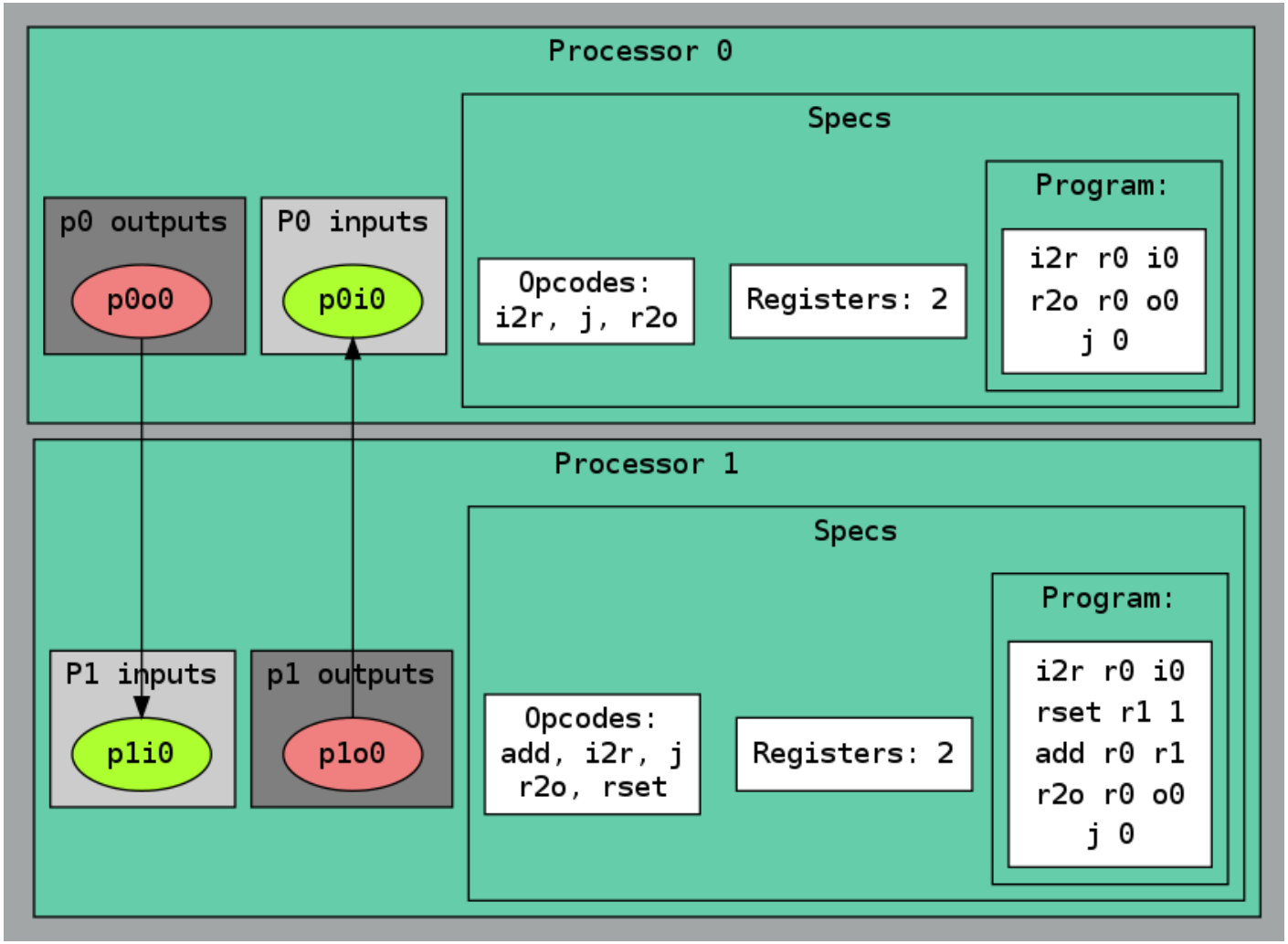
The toolchain perform the following steps:

- Map the two goroutines to two hardware cores.
- Creates two types of core, each one optimized to execute the assigned goroutine.
- Creates the two binaries.
- Connected the two core as inferred from the source code, using special IO registers.

The result is a multicore BondMachine:

The Ecosystem

A complete example



The Ecosystem

A complete example

distributed counter

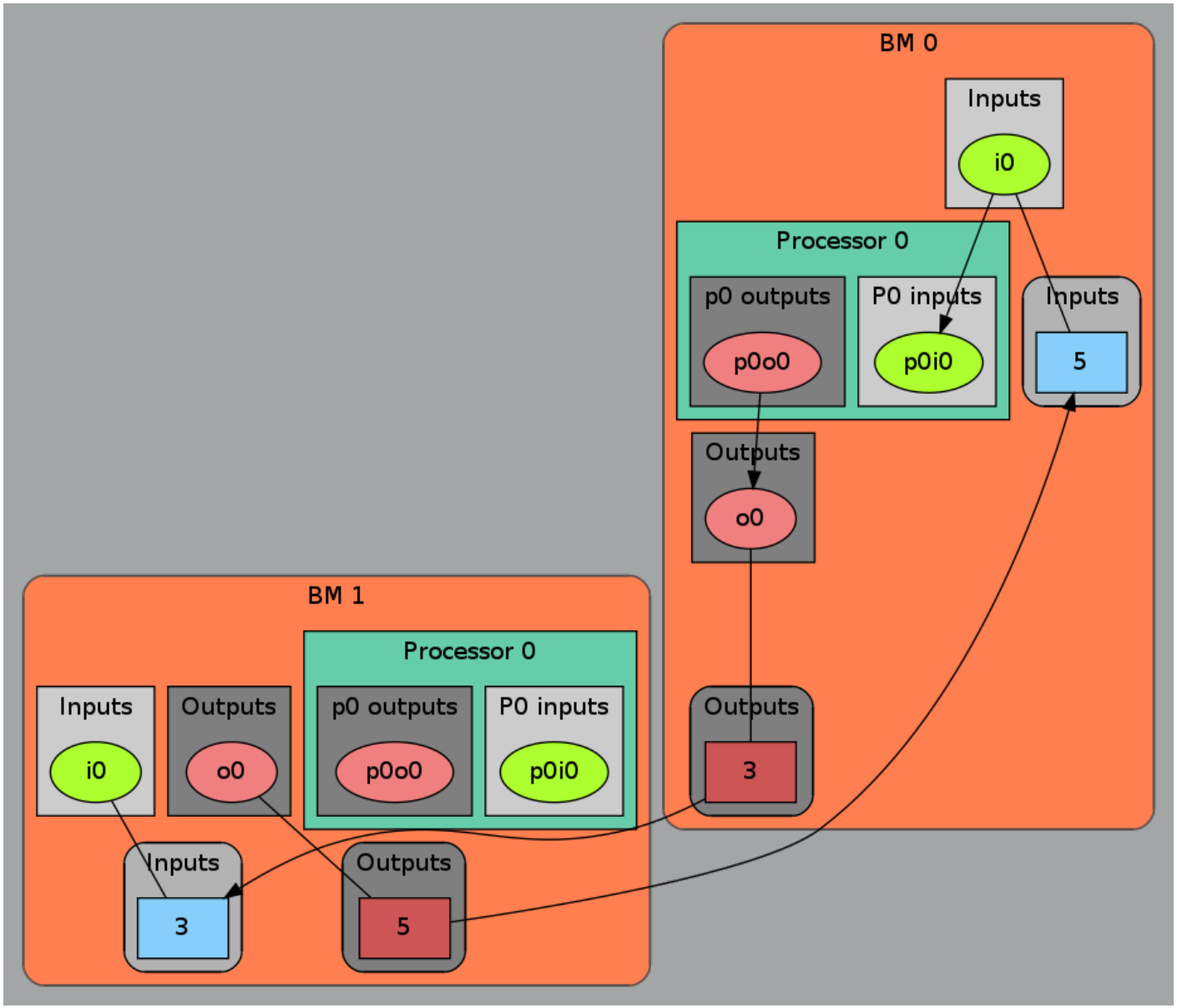
```
package main

import (
   "bondgo"
)

func pong() {
   var in0 bondgo.Input
   var out0 bondgo.Output
   in0 = bondgo.Make(bondgo.Input, 3)
   out0 = bondgo.Make(bondgo.Output, 5)
   for {
      bondgo.IOWrite(out0, bondgo.IORead(in0)+1)
   }
   bondgo.Void(in0)
   bondgo.Void(out0)
}
func main() {
   var in0 bondgo.Input
   var out0 bondgo.Output
   in0 = bondgo.Make(bondgo.Input, 5)
   out0 = bondgo.Make(bondgo.Output, 3)
device_1:
   go pong()
   for {
      bondgo.IOWrite(out0, bondgo.IORead(in0))
   }
   bondgo.Void(in0)
   bondgo.Void(out0)
}
```

The Ecosystem

A complete example



The Ecosystem

A complete example

The result is:

`https://youtube.com/embed/g9xYHK0zca4`

A general result

Parts of the system can be redeployed among different devices without changing the system behavior (only the performances).

Results

- User can deploy an entire HW/SW cluster starting from a code written in a High Level language.
- Workstation with emulated BondMachines, workstation with etherbond drivers, standalone BondMachines (FPGA) may join these clusters.

Results

- User can deploy an entire HW/SW cluster starting from a code written in a High Level language.
- Workstation with emulated BondMachines, workstation with etherbond drivers, standalone BondMachines (FPGA) may join these clusters.

Uses in Physics

Two possible use cases in Physics experiments are currently being explored:

- Real time pulse shape analysis in neutron detectors.
- Test beam for space experiments (DAMPE, HERD)

Real time pulse shape analysis in neutron detectors

The operation of the new generation of high-intensity neutron sources like SNS, JSNS and European Spallation Source (ESS, Lund, Sweden), now under construction, are introducing a new demand for neutron detection capabilities.

These demands yield to the need for new data collection procedures and new technology based on solid state Si devices.

We are trying to use BondMachines to make the real time shape analysis in this kind of detecting devices.

Courtesy of Prof. F.Sacchetti

Test beam for space experiments (DAMPE, HERD)

Trigger logic for test beams

In test beams, the DAQ system relies on the trigger system for data tacking (sensor signal digitization) during

- Calibration (random trigger or “off-spill” trigger)
- On spill data taking

Minimum elements used for trigger system:

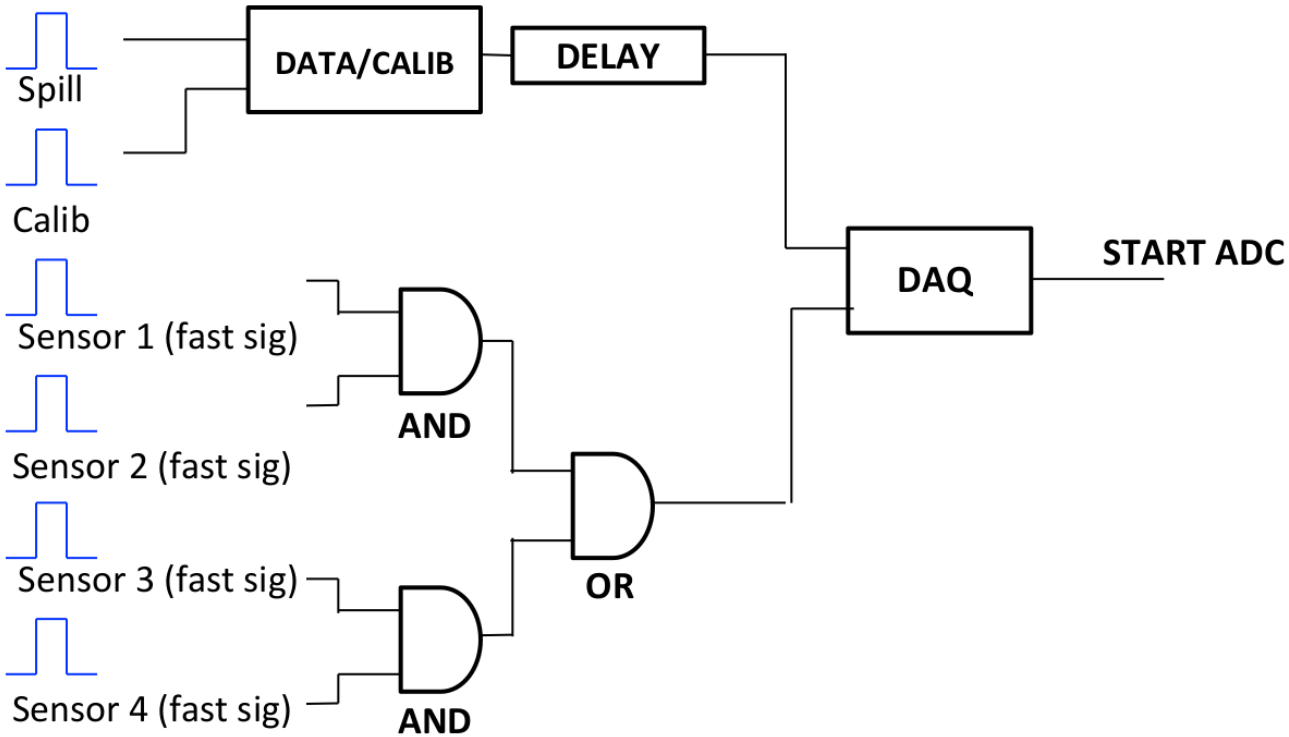
- Clock, pulser
- Logic gates (AND, OR,...)
- Delays

Trigger system implemented using NIM crates and DAQ machines

Courtesy of V.Vagelli and M.Duranti

Test beam for space experiments (DAMPE, HERD)

Trigger logic for test beams



Courtesy of V.Vagelli and M.Duranti

Test beam for space experiments (DAMPE, HERD)

Trigger logic for test beams

We are trying to explore the possibility of using BondMachine to handle efficiently this kind of operations.

Possible other uses

The BondMachine could be used in several types of real world applications, some of them being:

- Workstation FPGA accelerators for any kind of intensive computation.
- IoT and CyberPhysical systems.
- Computer Science educational applications.

Possible other uses

The BondMachine could be used in several types of real world applications, some of them being:

- Workstation FPGA accelerators for any kind of intensive computation.
- IoT and CyberPhysical systems.
- Computer Science educational applications.

Possible other uses

The BondMachine could be used in several types of real world applications, some of them being:

- Workstation FPGA accelerators for any kind of intensive computation.
- IoT and CyberPhysical systems.
- Computer Science educational applications.

Possible other uses

The BondMachine could be used in several types of real world applications, some of them being:

- Workstation FPGA accelerators for any kind of intensive computation.
- IoT and CyberPhysical systems.
- Computer Science educational applications.

Real world applications

Accelerators

A BM may be used as an hardware accelerator so that one can mix all together CPU and BM threads, that is one can off-load a task or a function using the BM (i.e. the FPGA)

The resulting accelerator would the advantage of being better suited to the specific problem than generic accelerators (GPU)

Real world applications

IoT and CyberPhysical systems

Scalability and extensibility of a HW / SW system built as described in the present project is greatly improved.

Indeed a system with interacting agents, of whatever type they are, would be the expression of a single and coherent program written in high level language.

Project History

- May 2016 - Poster presented at INFN CCR 2016.
- September 2016 - The first prototype is built.
- October 2016 - It is Selected and the prototype is presented at "Makerfaire 2016 Rome (The European edition)".
- November 2016 - Presented at "Umbria Business Match 2016".
- March 2017 - First tests for Physics applications.
- November 2017 - Presented at "Umbria Business Match 2017".
- December 2107 - Submitted at InnovateFPGA 2017
- February 2018 - Reached the InnovateFPGA 2017 EMEA Semifinal.

Project History

Next few months goals

- Development of the InnovateFPGA design project.
- Inclusion in some physics experiments.
- Search for people interested in joining the project.

Project History

Next few months goals

- Development of the InnovateFPGA design project.
- Inclusion in some physics experiments.
- Search for people interested in joining the project.

Project History

Next few months goals

- Development of the InnovateFPGA design project.
- Inclusion in some physics experiments.
- Search for people interested in joining the project.

Conclusions

The BondMachine is a new kind of computing device made possible in practice only by the emerging of new re-programmable hardware technologies such as FPGA.

Keeping the register machine abstraction it is possible to borrow well known languages and techniques in programming these devices removing the need of having a general purpose architecture.

The result of this process is the construction of a computer architecture that is not anymore a static constraint where computing occurs but its creation becomes a part of the computing process, gaining computing power and flexibility.

Over this abstraction is it possible to create a full computing Ecosystem.

Future work

- The project is a prototype.
- Include new processor shared objects and currently unsupported opcodes.
- Extend the compiler to include more data structures.
- Improve the networking including new interconnection firmwares.
- Work on BondMachine as accelerators.
- What would an OS for BondMachines look like ?

Future work

- The project is a prototype.
- Include new processor shared objects and currently unsupported opcodes.
- Extend the compiler to include more data structures.
- Improve the networking including new interconnection firmwares.
- Work on BondMachine as accelerators.
- What would an OS for BondMachines look like ?

Future work

- The project is a prototype.
- Include new processor shared objects and currently unsupported opcodes.
- Extend the compiler to include more data structures.
- Improve the networking including new interconnection firmwares.
- Work on BondMachine as accelerators.
- What would an OS for BondMachines look like ?

Future work

- The project is a prototype.
- Include new processor shared objects and currently unsupported opcodes.
- Extend the compiler to include more data structures.
- Improve the networking including new interconnection firmwares.
- Work on BondMachine as accelerators.
- What would an OS for BondMachines look like ?

Future work

- The project is a prototype.
- Include new processor shared objects and currently unsupported opcodes.
- Extend the compiler to include more data structures.
- Improve the networking including new interconnection firmwares.
- Work on BondMachine as accelerators.
- What would an OS for BondMachines look like ?

Future work

- The project is a prototype.
- Include new processor shared objects and currently unsupported opcodes.
- Extend the compiler to include more data structures.
- Improve the networking including new interconnection firmwares.
- Work on BondMachine as accelerators.
- What would an OS for BondMachines look like ?

Mirko Mariotti
Department of Physics and Geology, University of Perugia.
mirko.mariotti@unipg.it
http://bondmachine.fisica.unipg.it