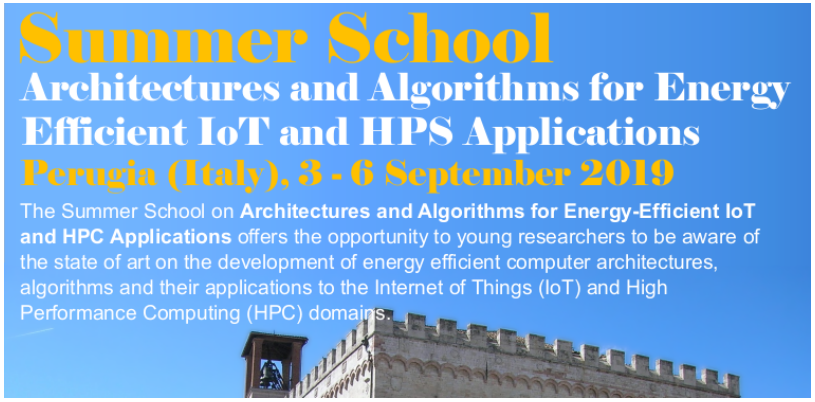
The BondMachine Toolkit

Enabling Machine Learning on FPGA

Mirko Mariotti

Department of Physics and Geology - University of Perugia
INFN Perugia

NiPS Summer School 2019
Architectures and Algorithms for Energy-Efficient IoT and HPC Applications
3-6 September 2019 - Perugia



Introduction

The BondMachine: a comprehensive approach to computing with FPGA.

In this presentation i will talk about:

- Technological background of the project
- The BondMachine Project: the Architecture
- The BondMachine Project: the Tools
- Use cases

Demo sessions

Some topic will have a demo session.

The code will be available at:

`http://bondmachine.fisica.unipg.it`

Requirements:

- Linux Workstation
- Vivado
- Zedboard

Technological Background

Current challenges in computing

- Von Neumann Bottleneck:
  New computational problems show that current architectural models has to be improved or changed to address future payloads.

- Energy Efficient computation:
  Not wasting "resources" (silicon, time, energy, instructions).
  Using the right resource for the specific case

- Edge/Fog/Cloud Computing:
  Making the computation where it make sense
  Avoiding the transfer of unnecessary data
  Creating consistent interfaces for distributed systems

Current challenges in computing

- Von Neumann Bottleneck:
  New computational problems show that current architectural models has to be improved or changed to address future payloads.

- Energy Efficient computation:
  Not wasting "resources" (silicon, time, energy, instructions).
  Using the right resource for the specific case

- Edge/Fog/Cloud Computing:
  Making the computation where it make sense
  Avoiding the transfer of unnecessary data
  Creating consistent interfaces for distributed systems

Current challenges in computing

- Von Neumann Bottleneck:
  New computational problems show that current architectural models has to be improved or changed to address future payloads.

- Energy Efficient computation:
  Not wasting "resources" (silicon, time, energy, instructions).
  Using the right resource for the specific case

- Edge/Fog/Cloud Computing:
  Making the computation where it make sense
  Avoiding the transfer of unnecessary data
  Creating consistent interfaces for distributed systems

FPGA

What is it ?

- A field-programmable gate array (FPGA) is an integrated circuit whose logic is re-programmable. It's used to build reconfigurable digital circuits.
- FPGAs contain an array of programmable logic blocks, and a hierarchy of reconfigurable interconnects that allow the blocks to be "wired together".

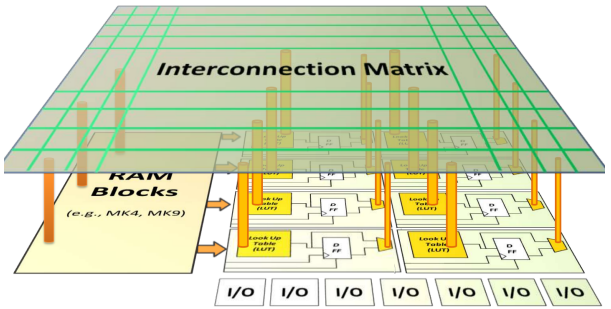
- Logic blocks can be configured to perform complex combinational functions.

- The FPGA configuration is generally specified using a hardware description language (HDL).

FPGA

What is it ?

- A field-programmable gate array (FPGA) is an integrated circuit whose logic is re-programmable. It's used to build reconfigurable digital circuits.
- FPGAs contain an array of programmable logic blocks, and a hierarchy of reconfigurable interconnects that allow the blocks to be "wired together".

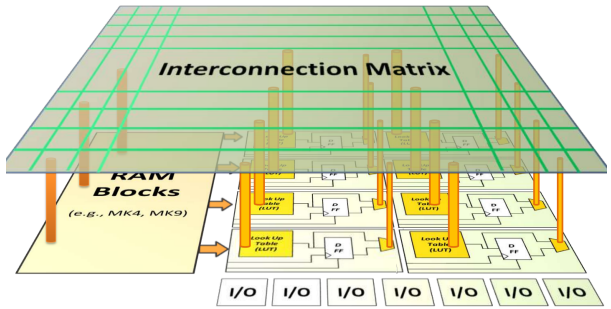


- Logic blocks can be configured to perform complex combinational functions.
- The FPGA configuration is generally specified using a hardware description language (HDL).

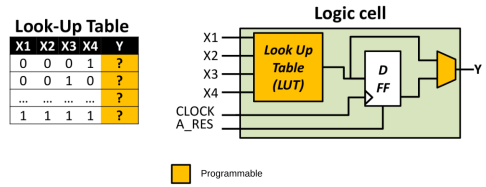
FPGA

What is it ?

- A field-programmable gate array (FPGA) is an integrated circuit whose logic is re-programmable. It's used to build reconfigurable digital circuits.
- FPGAs contain an array of programmable logic blocks, and a hierarchy of reconfigurable interconnects that allow the blocks to be "wired together".



- Logic blocks can be configured to perform complex combinational functions.

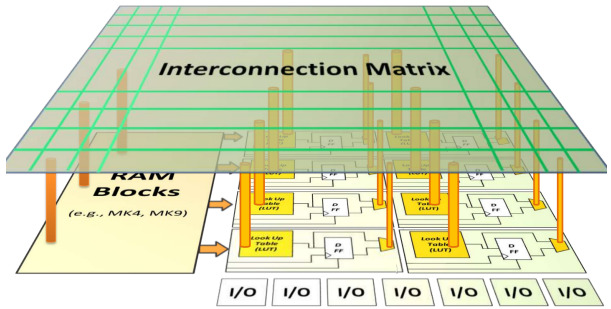


- The FPGA configuration is generally specified using a hardware description language (HDL).

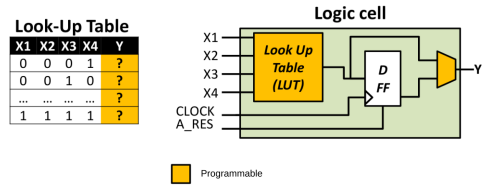
FPGA

What is it ?

- A field-programmable gate array (FPGA) is an integrated circuit whose logic is re-programmable. It's used to build reconfigurable digital circuits.
- FPGAs contain an array of programmable logic blocks, and a hierarchy of reconfigurable interconnects that allow the blocks to be "wired together".



- Logic blocks can be configured to perform complex combinational functions.



- The FPGA configuration is generally specified using a hardware description language (HDL).

FPGA

Use in computing

The use of FPGA in computing is growing due several reasons:

- can potentially deliver great performance via massive parallelism
- can address payloads which are not performing well on uniprocessors (Neural Networks, Deep Learning)
- can handle efficiently non-standard data types

FPGA

Use in computing

The use of FPGA in computing is growing due several reasons:

- can potentially deliver great performance via massive parallelism
- can address payloads which are not performing well on uniprocessors (Neural Networks, Deep Learning)
- can handle efficiently non-standard data types

FPGA

Use in computing

The use of FPGA in computing is growing due several reasons:

- can potentially deliver great performance via massive parallelism
- can address payloads which are not performing well on uniprocessors (Neural Networks, Deep Learning)
- can handle efficiently non-standard data types

FPGA

Challenges in computing

On the other hand the adoption on FPGA poses several challenges:

- Porting of legacy code is usually hard.
- Interoperability with standard applications is problematic.

FPGA

Challenges in computing

On the other hand the adoption on FPGA poses several challenges:

- Porting of legacy code is usually hard.
- Interoperability with standard applications is problematic.

Computer Architectures

Computer Architectures

Multi-core and Heterogeneous

Today's computer architecture are:

- Multi-core,Two or more independent actual processing units execute multiple instructions at the same time.
  - The power is given by the number of cores.
  - Parallelism has to be addressed.
- Heterogeneous, different types of processing units.
  - Cell, GPU, Parallela, TPU.
  - The power is given by the specialization.
  - The units data transfer has to be addressed.
  - The payloads scheduling has to be addressed.

Computer Architectures

Multi-core and Heterogeneous

Today's computer architecture are:

- Multi-core, Two or more independent actual processing units execute multiple instructions at the same time.
  - The power is given by the number of cores.
  - Parallelism has to be addressed.
- Heterogeneous, different types of processing units.
  - Cell, GPU, Parallela, TPU.
  - The power is given by the specialization.
  - The units data transfer has to be addressed.
  - The payloads scheduling has to be addressed.

Computer Architectures

Multi-core and Heterogeneous

Today's computer architecture are:

- Multi-core, Two or more independent actual processing units execute multiple instructions at the same time.
  - The power is given by the number of cores.
  - Parallelism has to be addressed.
- Heterogeneous, different types of processing units.
  - Cell, GPU, Parallela, TPU.
  - The power is given by the specialization.
  - The units data transfer has to be addressed.
  - The payloads scheduling has to be addressed.

Computer Architectures

Multi-core and Heterogeneous

Today's computer architecture are:

- Multi-core, Two or more independent actual processing units execute multiple instructions at the same time.
  - The power is given by the number of cores.
  - Parallelism has to be addressed.
- Heterogeneous, different types of processing units.
  - Cell, GPU, Parallela, TPU.
  - The power is given by the specialization.
  - The units data transfer has to be addressed.
  - The payloads scheduling has to be addressed.

Computer Architectures

Multi-core and Heterogeneous

Today's computer architecture are:

- Multi-core,Two or more independent actual processing units execute multiple instructions at the same time.
  - The power is given by the number of cores.
  - Parallelism has to be addressed.
- Heterogeneous, different types of processing units.
  - Cell, GPU, Parallela, TPU.
  - The power is given by the specialization.
  - The units data transfer has to be addressed.
  - The payloads scheduling has to be addressed.

Computer Architectures

Multi-core and Heterogeneous

Today's computer architecture are:

- Multi-core,Two or more independent actual processing units execute multiple instructions at the same time.
  - The power is given by the number of cores.
  - Parallelism has to be addressed.
- Heterogeneous, different types of processing units.
  - Cell, GPU, Parallela, TPU.
  - The power is given by the specialization.
  - The units data transfer has to be addressed.
  - The payloads scheduling has to be addressed.

Computer Architectures

Multi-core and Heterogeneous

Today's computer architecture are:

- Multi-core,Two or more independent actual processing units execute multiple instructions at the same time.
  - The power is given by the number of cores.
  - Parallelism has to be addressed.
- Heterogeneous, different types of processing units.
  - Cell, GPU, Parallela, TPU.
  - The power is given by the specialization.
  - The units data transfer has to be addressed.
  - The payloads scheduling has to be addressed.

Computer Architectures

Multi-core and Heterogeneous

Today's computer architecture are:

- Multi-core,Two or more independent actual processing units execute multiple instructions at the same time.
  - The power is given by the number of cores.
  - Parallelism has to be addressed.
- Heterogeneous, different types of processing units.
  - Cell, GPU, Parallela, TPU.
  - The power is given by the specialization.
  - The units data transfer has to be addressed.
  - The payloads scheduling has to be addressed.

Computer Architectures

Multi-core and Heterogeneous

Today's computer architecture are:

- Multi-core,Two or more independent actual processing units execute multiple instructions at the same time.
  - The power is given by the number of cores.
  - Parallelism has to be addressed.
- Heterogeneous, different types of processing units.
  - Cell, GPU, Parallela, TPU.
  - The power is given by the specialization.
  - The units data transfer has to be addressed.
  - The payloads scheduling has to be addressed.

Computer Architectures

Multi-core and Heterogeneous

Today's computer architecture are:

- Multi-core,Two or more independent actual processing units execute multiple instructions at the same time.
  - The power is given by the number of cores.
  - Parallelism has to be addressed.
- Heterogeneous, different types of processing units.
  - Cell, GPU, Parallela, TPU.
  - The power is given by the specialization.
  - The units data transfer has to be addressed.
  - The payloads scheduling has to be addressed.

The BondMachine

The first idea

High level sources: Go, TensorFlow, NN, ...

Building a new kind of computer architecture (multi-core and heterogeneous both in cores types and interconnections) which dynamically adapt to the specific computational problem rather than be static.

BM architecture Layer

FPGA

Concurrency and Specialization

The BondMachine

The first idea

High level sources: Go, TensorFlow, NN, ...

Building a new kind of computer architecture (multi-core and heterogeneous both in cores types and interconnections) which dynamically adapt to the specific computational problem rather than be static.

BM architecture Layer

FPGA

Concurrency and Specialization

The BondMachine

The first idea

High level sources: Go, TensorFlow, NN, ...

Building a new kind of computer architecture (multi-core and heterogeneous both in cores types and interconnections) which dynamically adapt to the specific computational problem rather than be static.

BM architecture Layer

FPGA

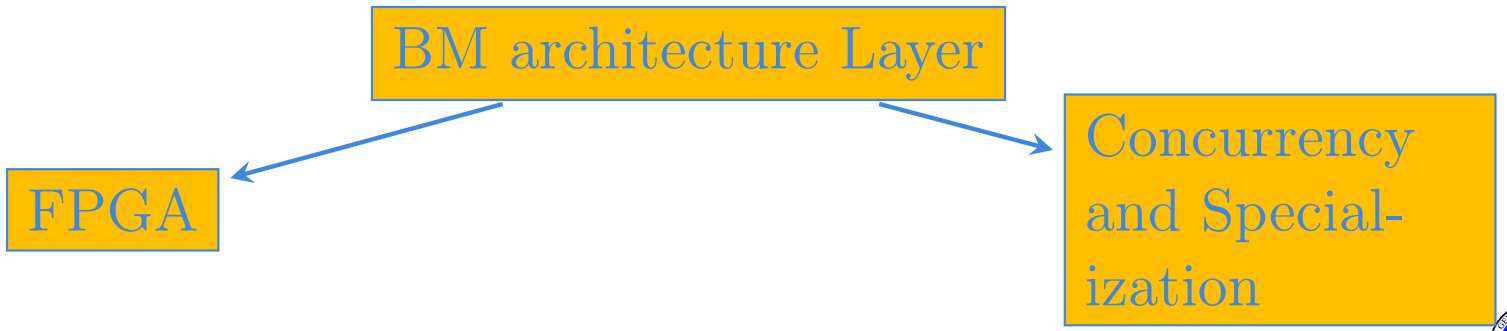
Concurrency and Specialization

The BondMachine

The first idea

High level sources: Go, TensorFlow, NN, ...

Building a new kind of computer architecture (multi-core and heterogeneous both in cores types and interconnections) which dynamically adapt to the specific computational problem rather than be static.



The BondMachine

The first idea

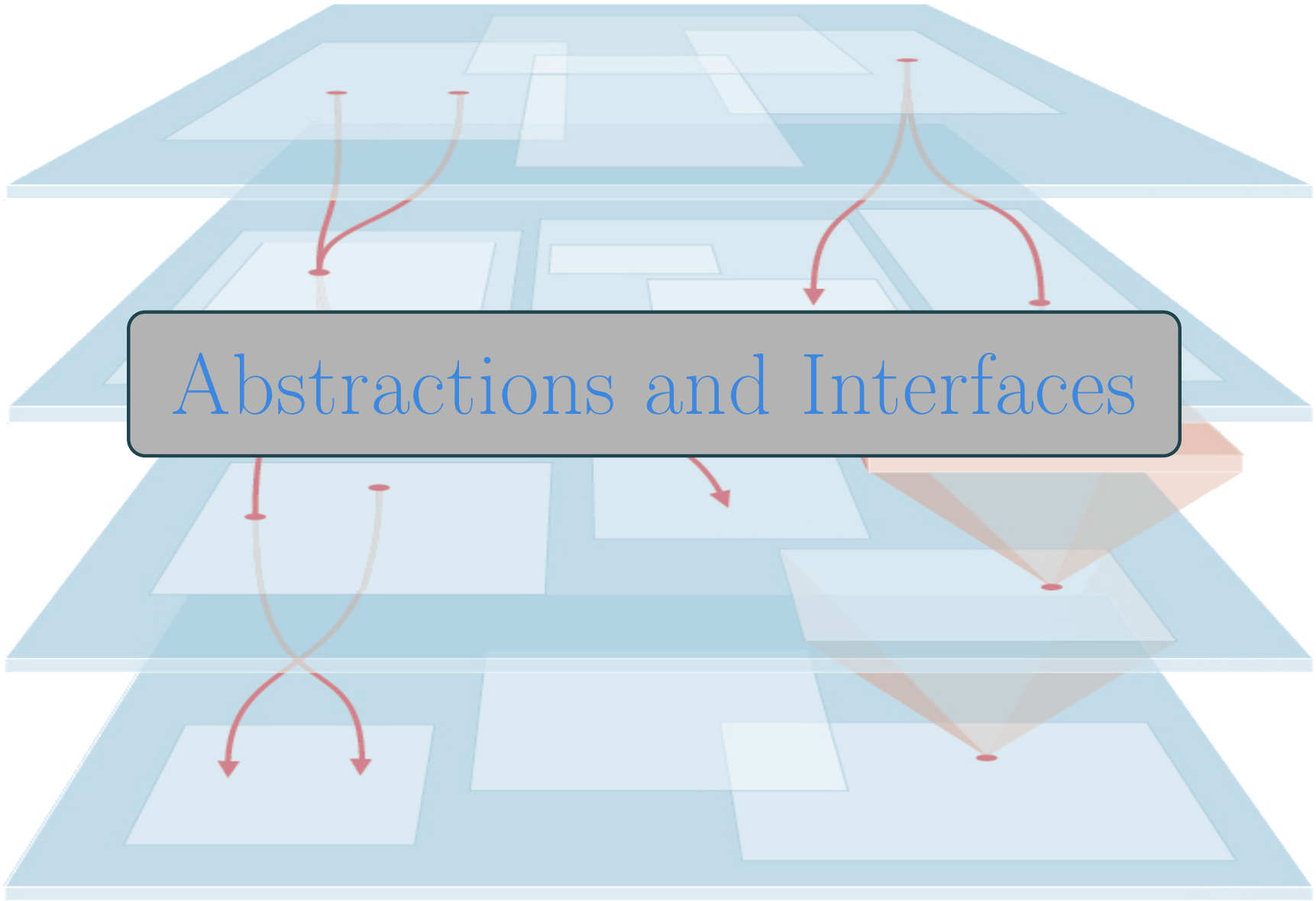
High level sources: Go, TensorFlow, NN, ...

Building a new kind of computer architecture (multi-core and heterogeneous both in cores types and interconnections) which dynamically adapt to the specific computational problem rather than be static.

BM architecture Layer

FPGA

Concurrency and Specialization



Abstractions and Interfaces

Layer, Abstractions and Interfaces

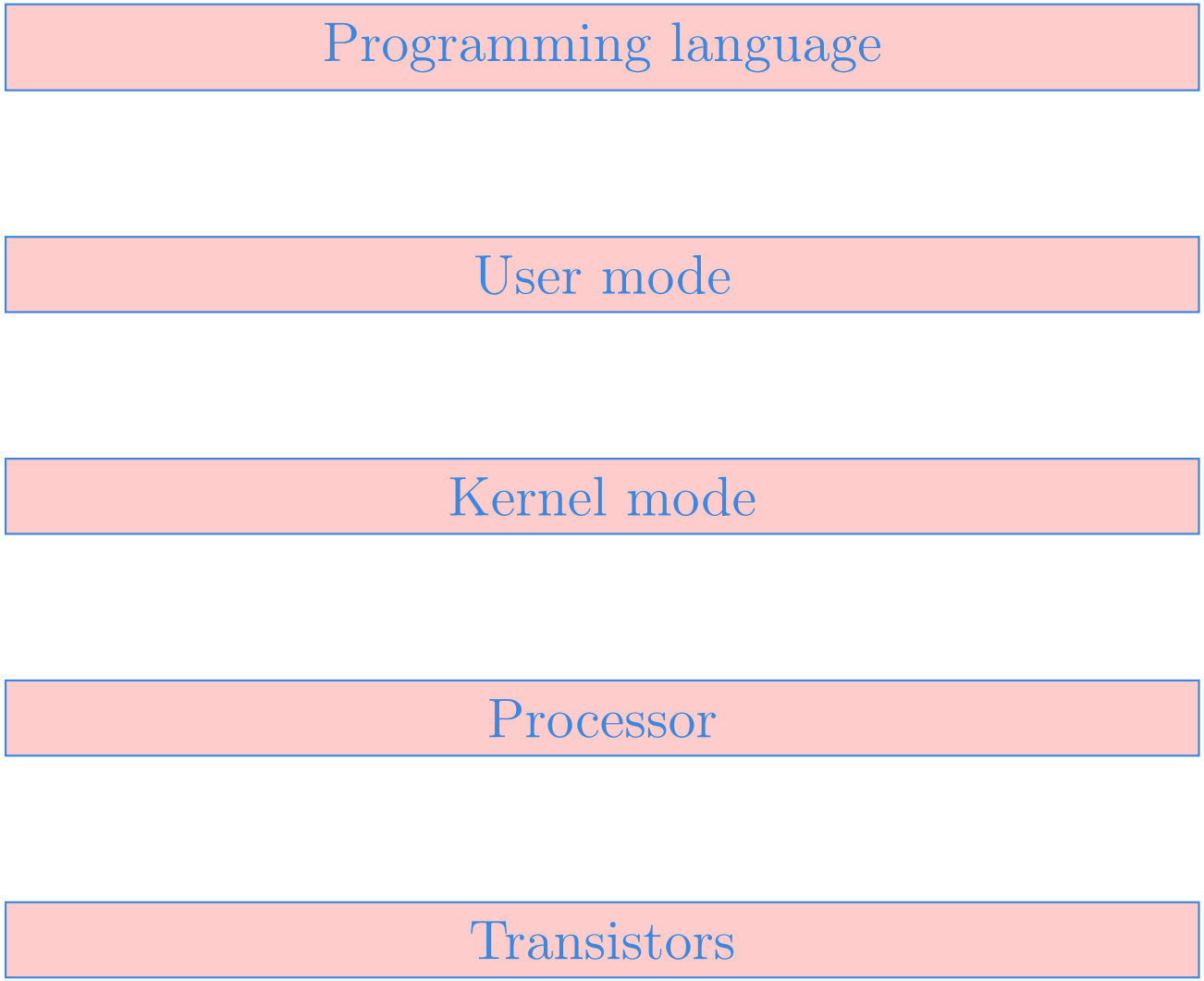
Introduction

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.

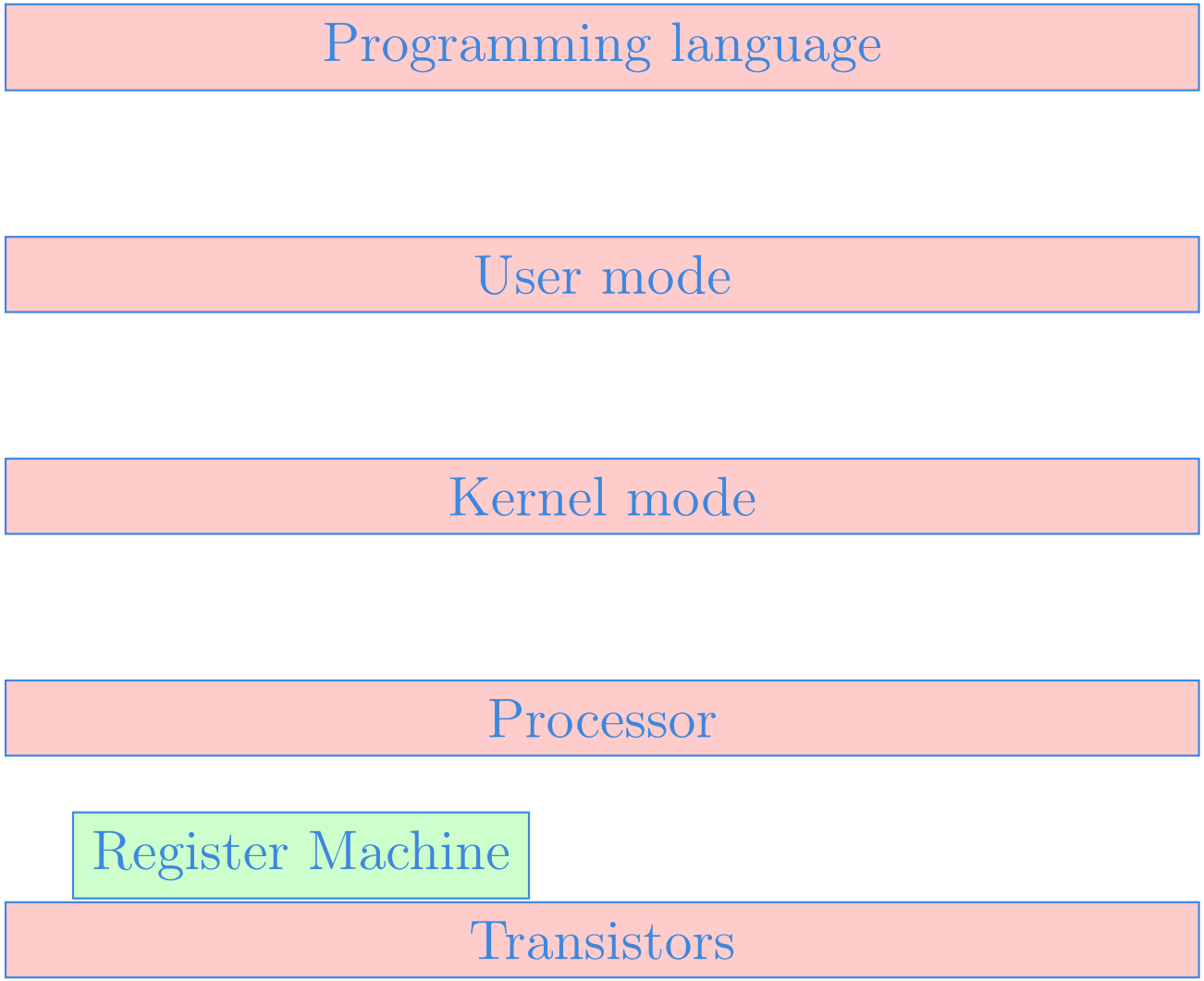
Layers, Abstractions and Interfaces

An example



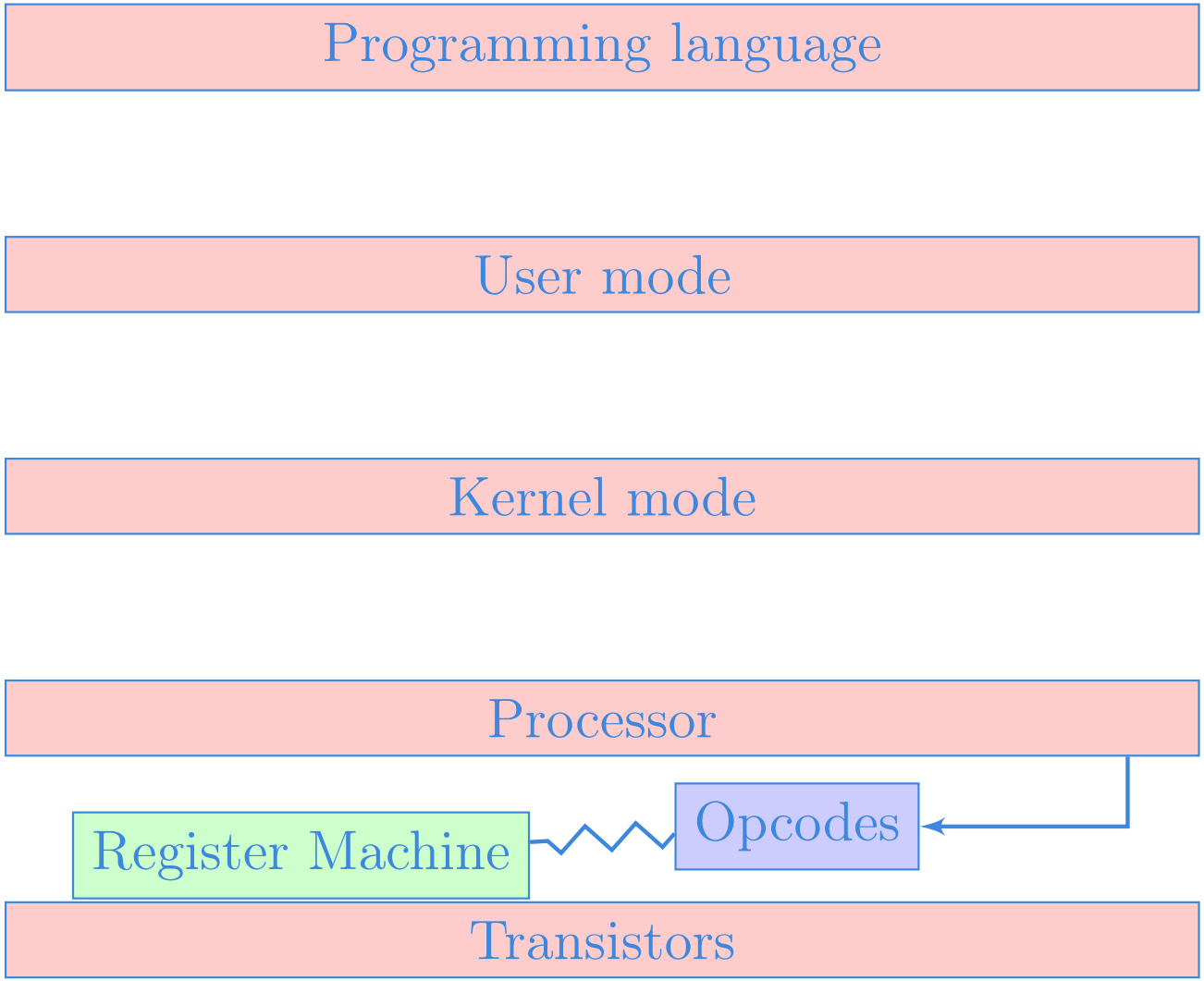
Layers, Abstractions and Interfaces

An example



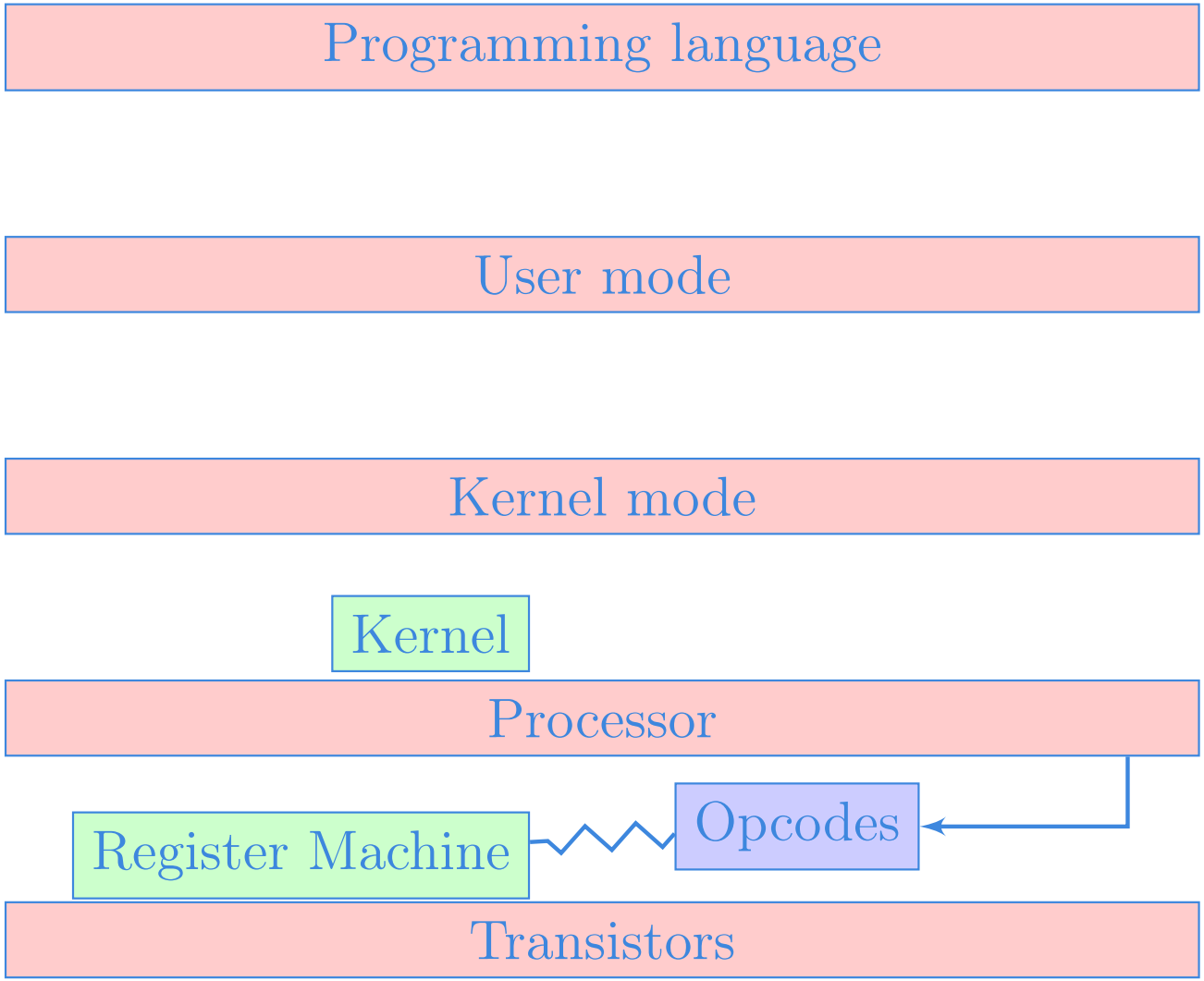
Layers, Abstractions and Interfaces

An example



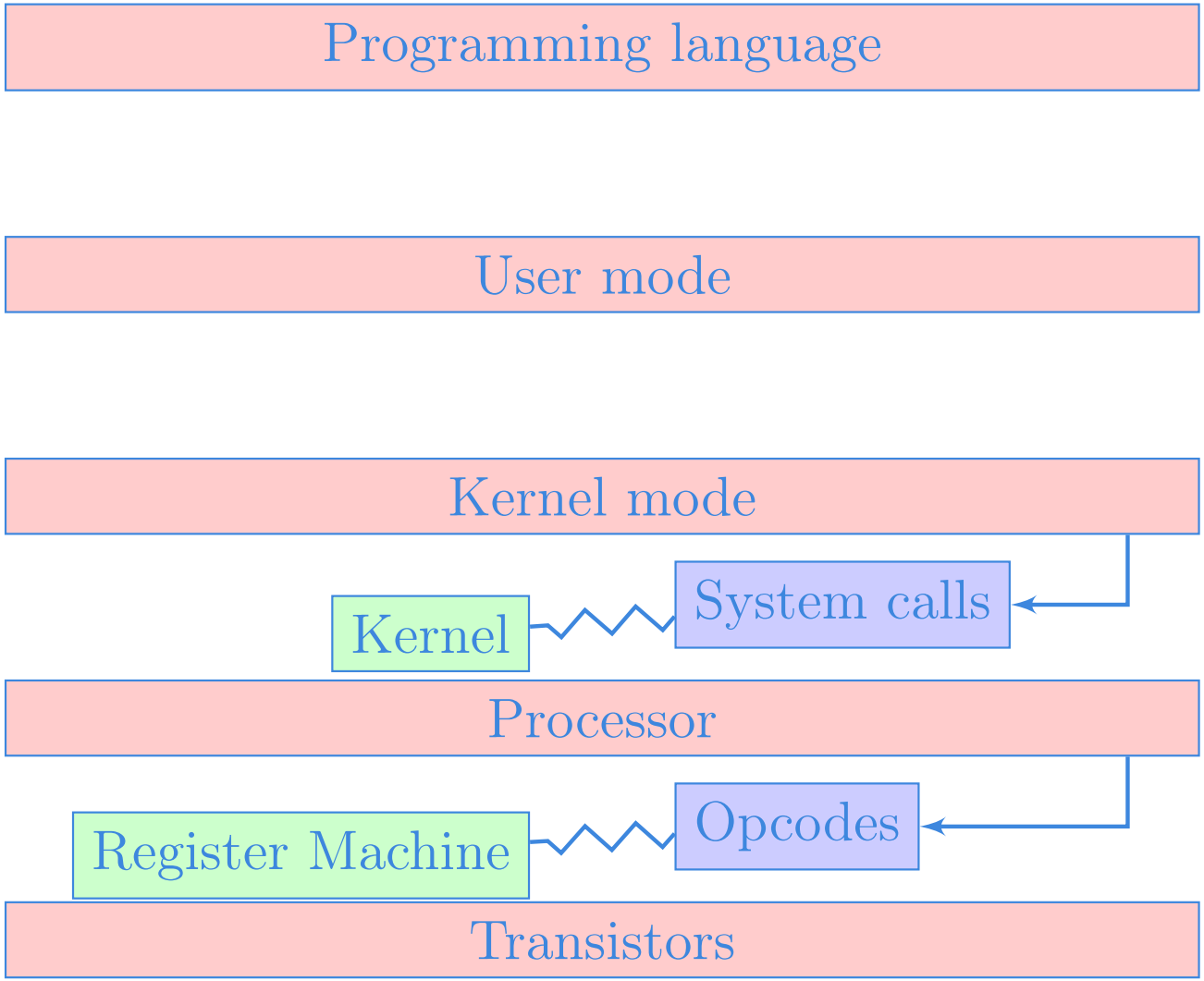
Layers, Abstractions and Interfaces

An example



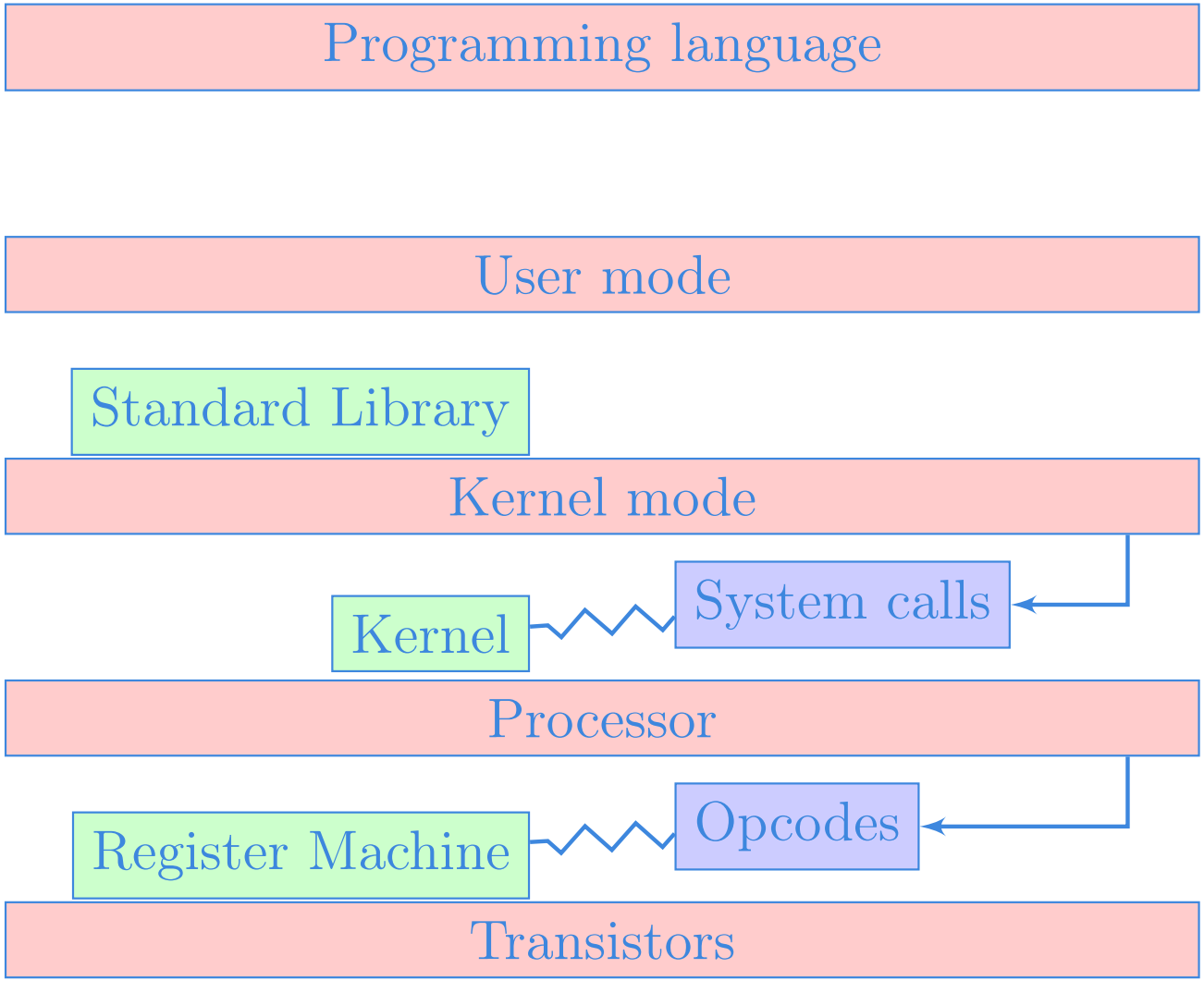
Layers, Abstractions and Interfaces

An example



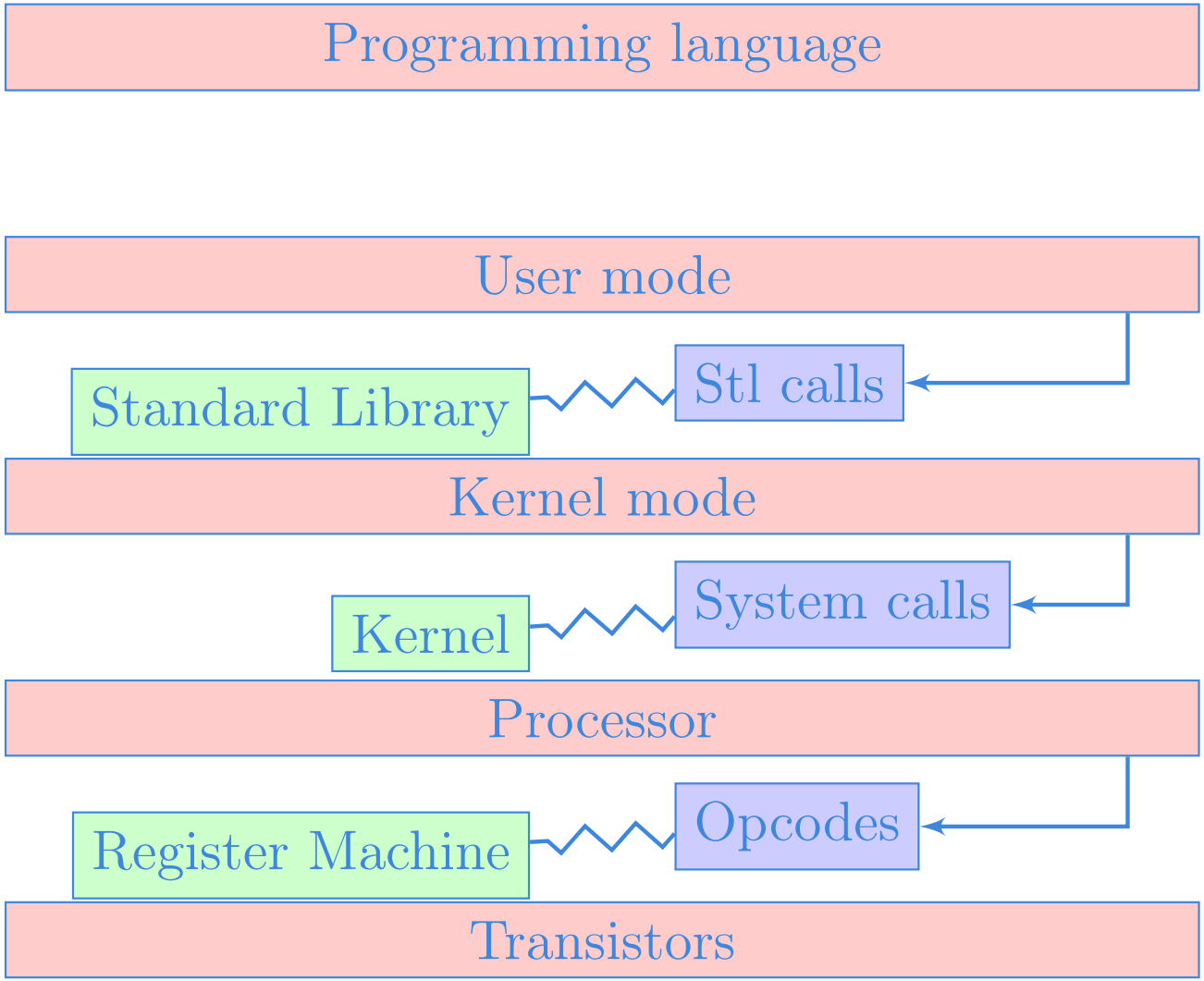
Layers, Abstractions and Interfaces

An example



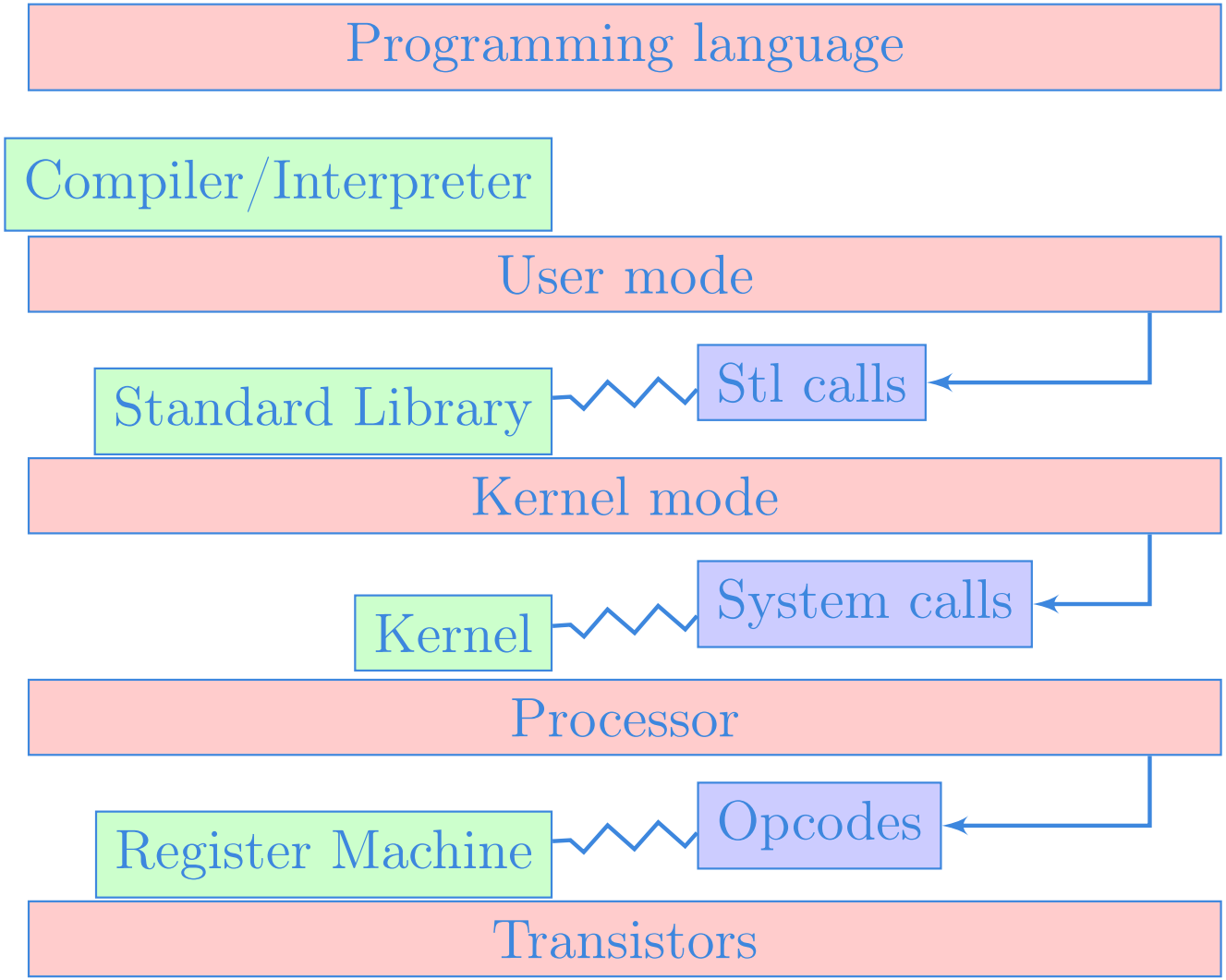
Layers, Abstractions and Interfaces

An example



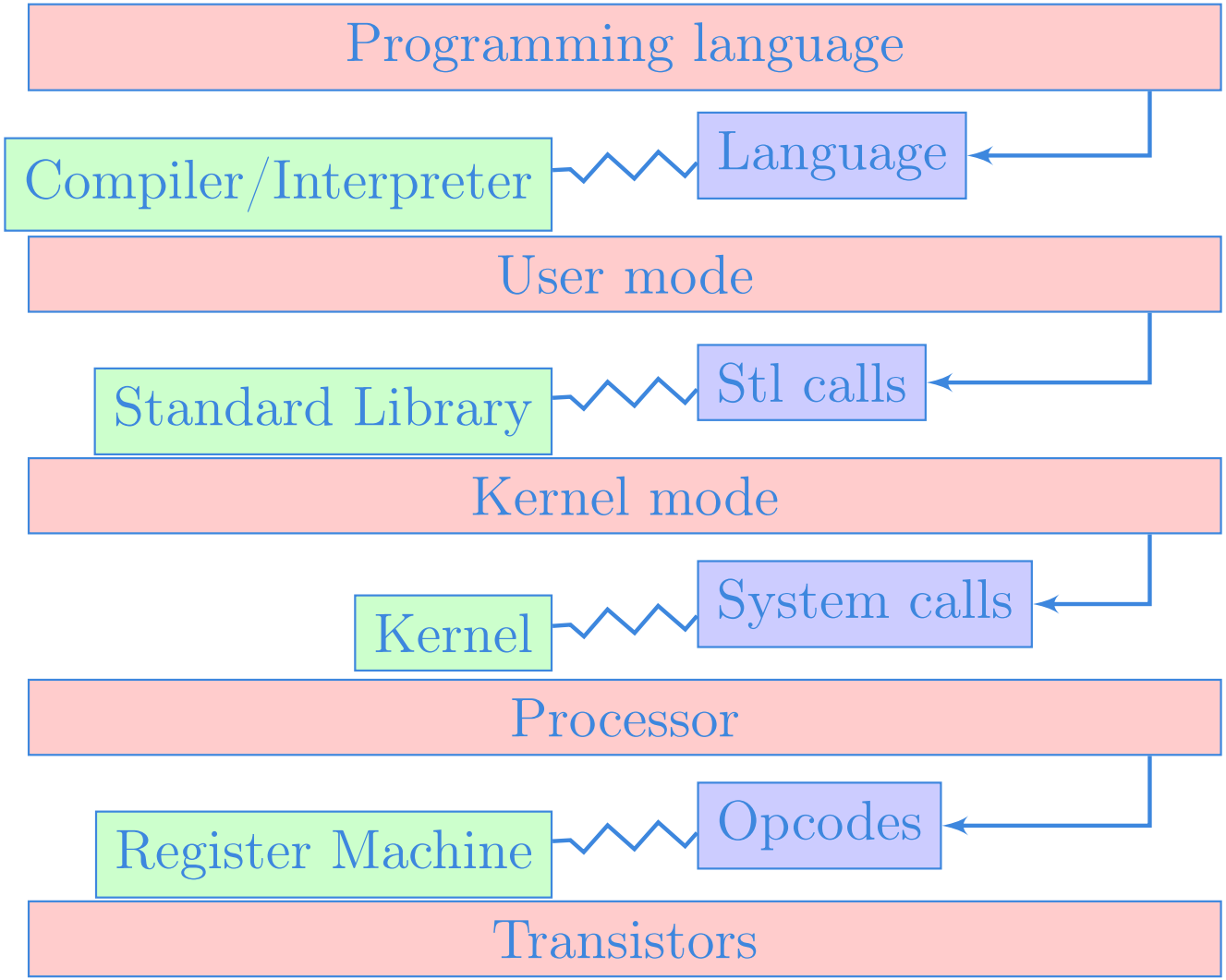
Layers, Abstractions and Interfaces

An example



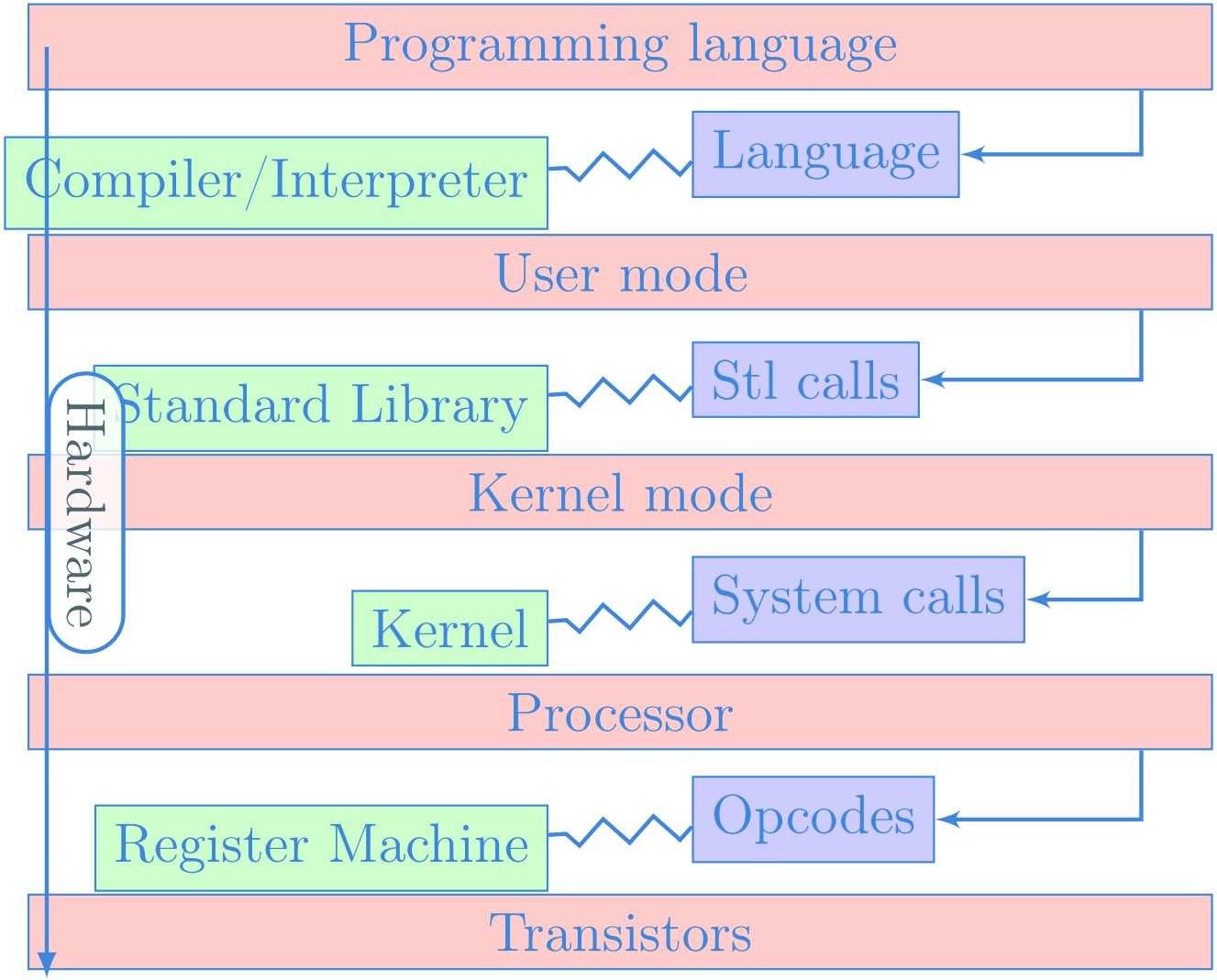
Layers, Abstractions and Interfaces

An example



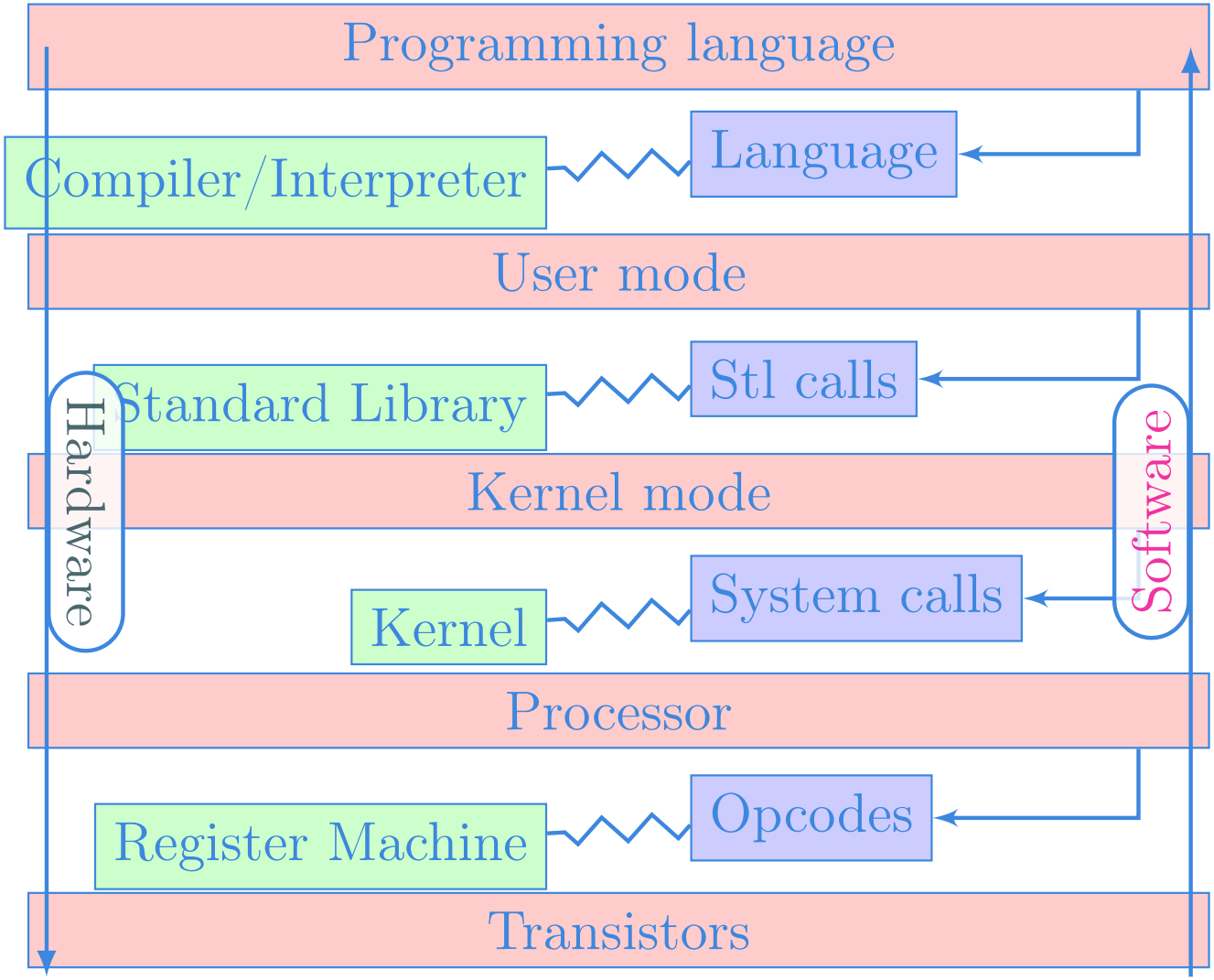
Layers, Abstractions and Interfaces

An example



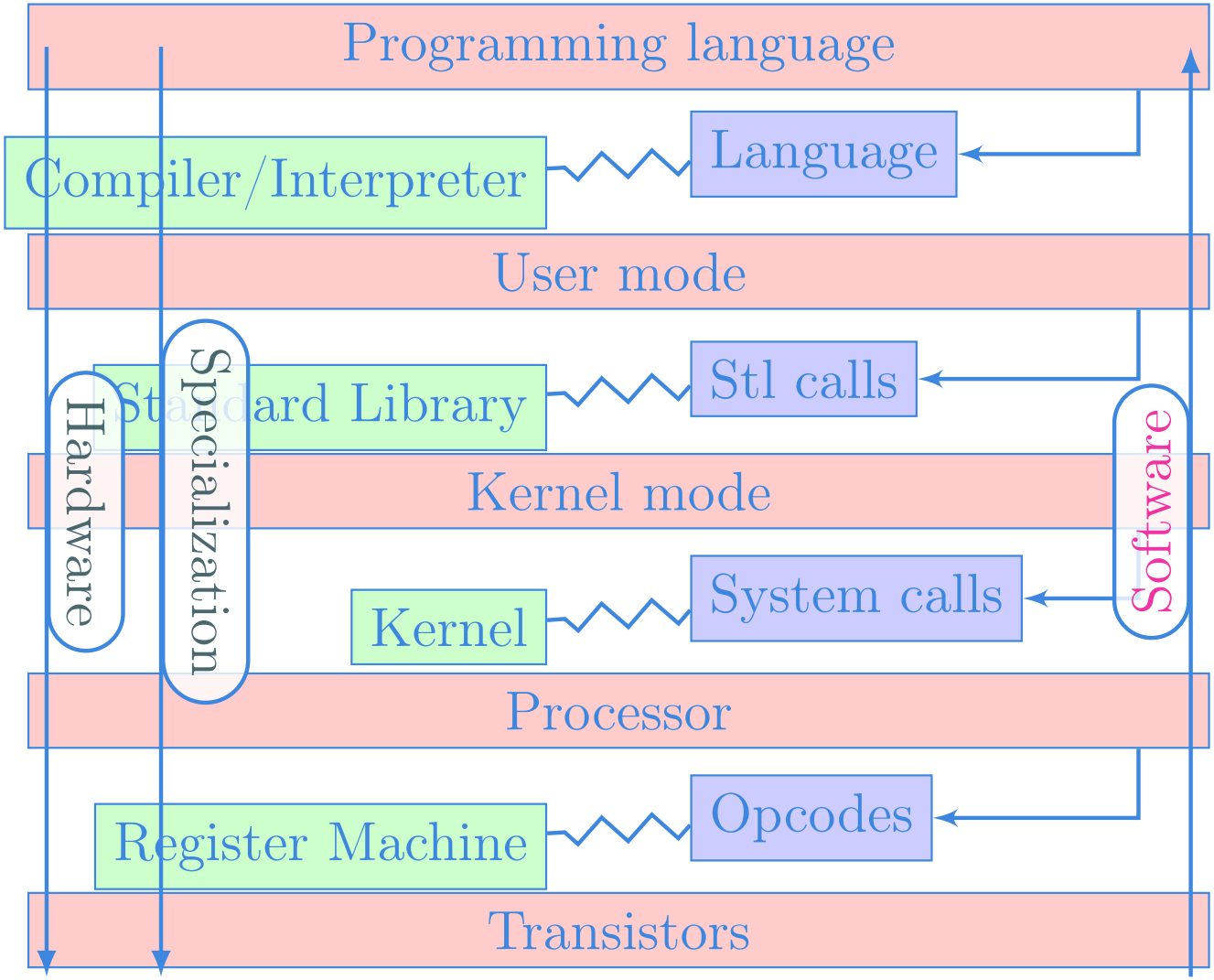
Layers, Abstractions and Interfaces

An example



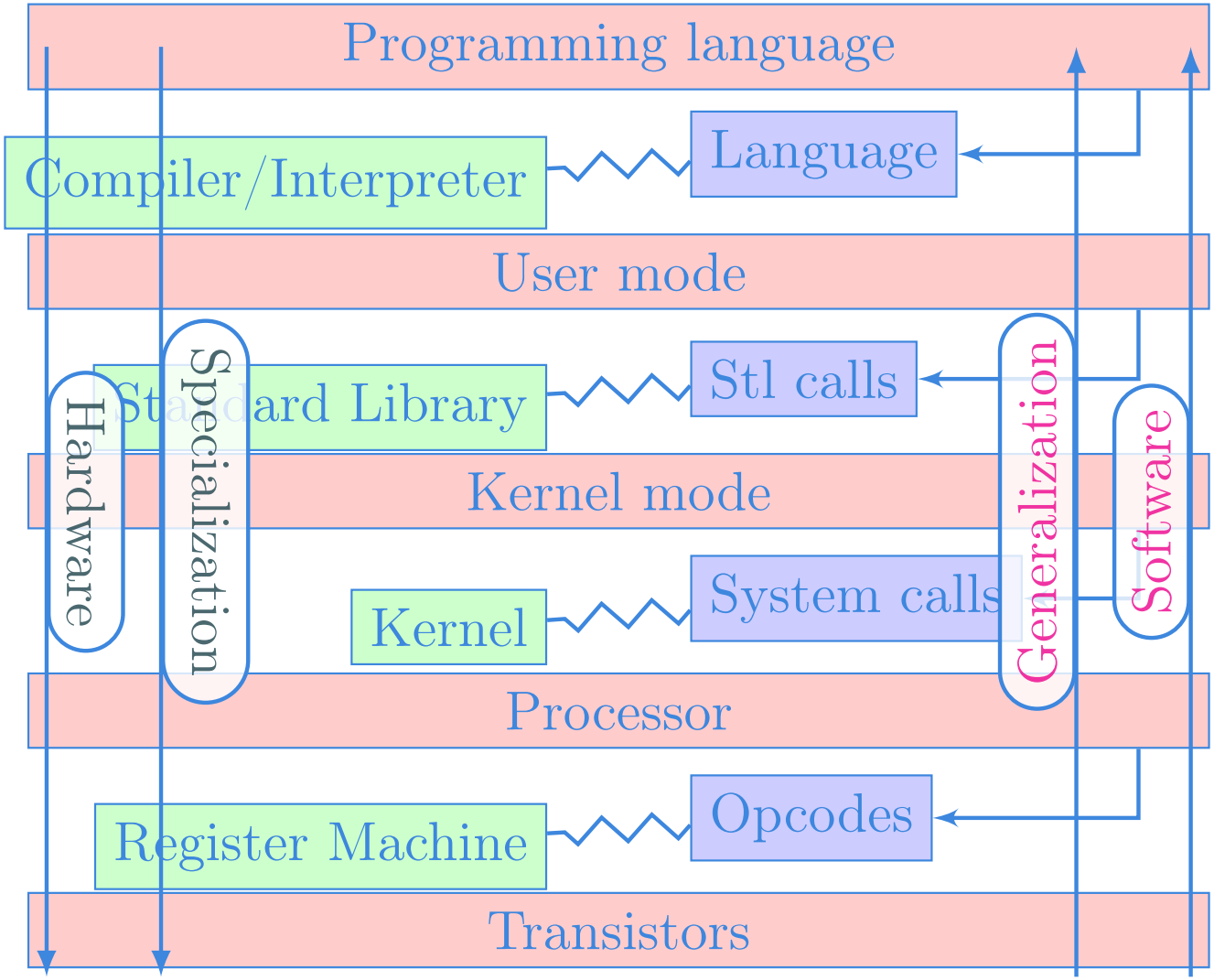
Layers, Abstractions and Interfaces

An example



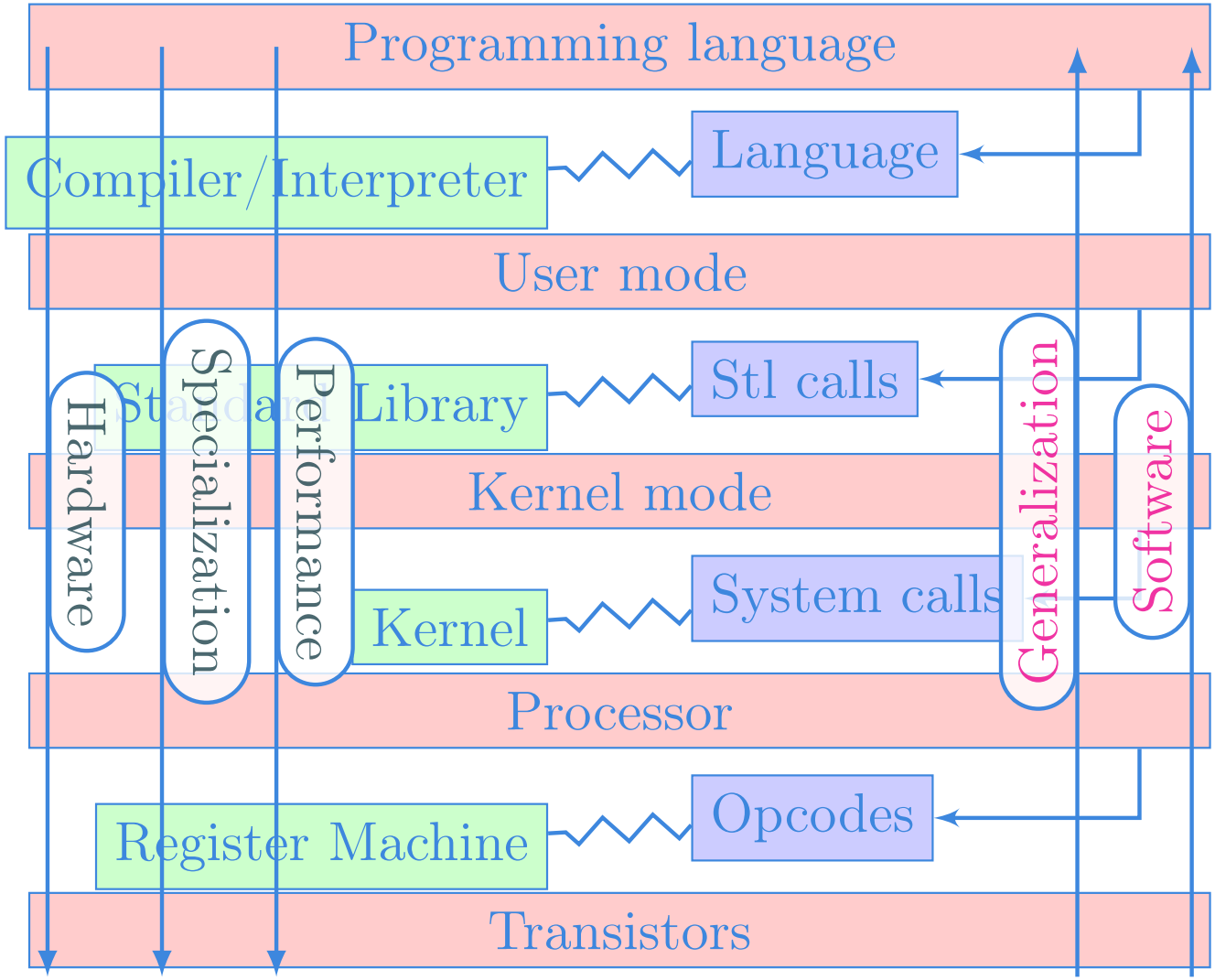
Layers, Abstractions and Interfaces

An example



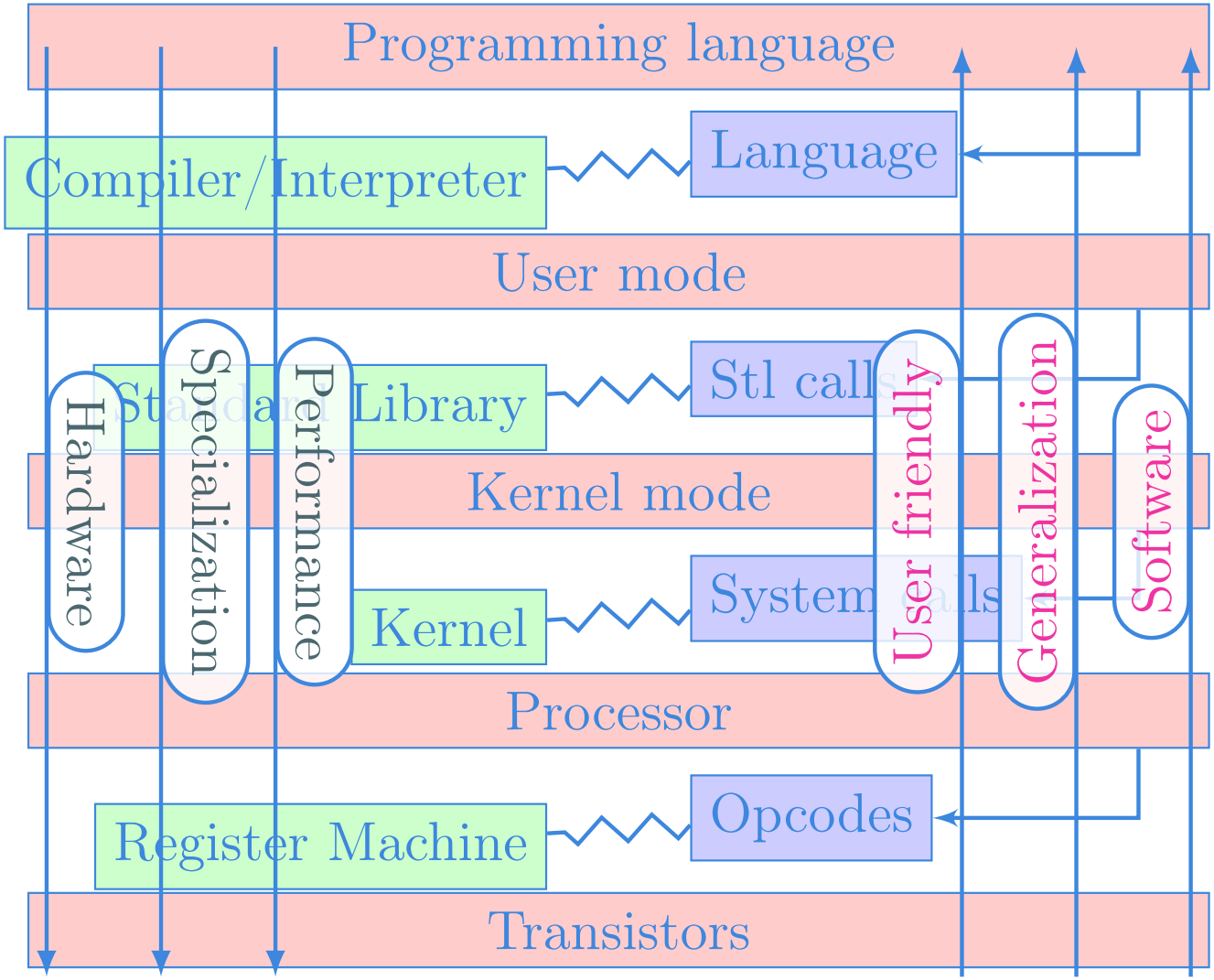
Layers, Abstractions and Interfaces

An example



Layers, Abstractions and Interfaces

An example



Layers, Abstractions and Interfaces

The second idea

Rethinking the stack

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.

BondMachine

Introducing the BondMachine (BM)

The BondMachine is a software ecosystem for the dynamic generation of computer architectures that:

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

Introducing the BondMachine (BM)

The BondMachine is a software ecosystem for the dynamic generation of computer architectures that:

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

Introducing the BondMachine (BM)

The BondMachine is a software ecosystem for the dynamic generation of computer architectures that:

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

Introducing the BondMachine (BM)

The BondMachine is a software ecosystem for the dynamic generation of computer architectures that:

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

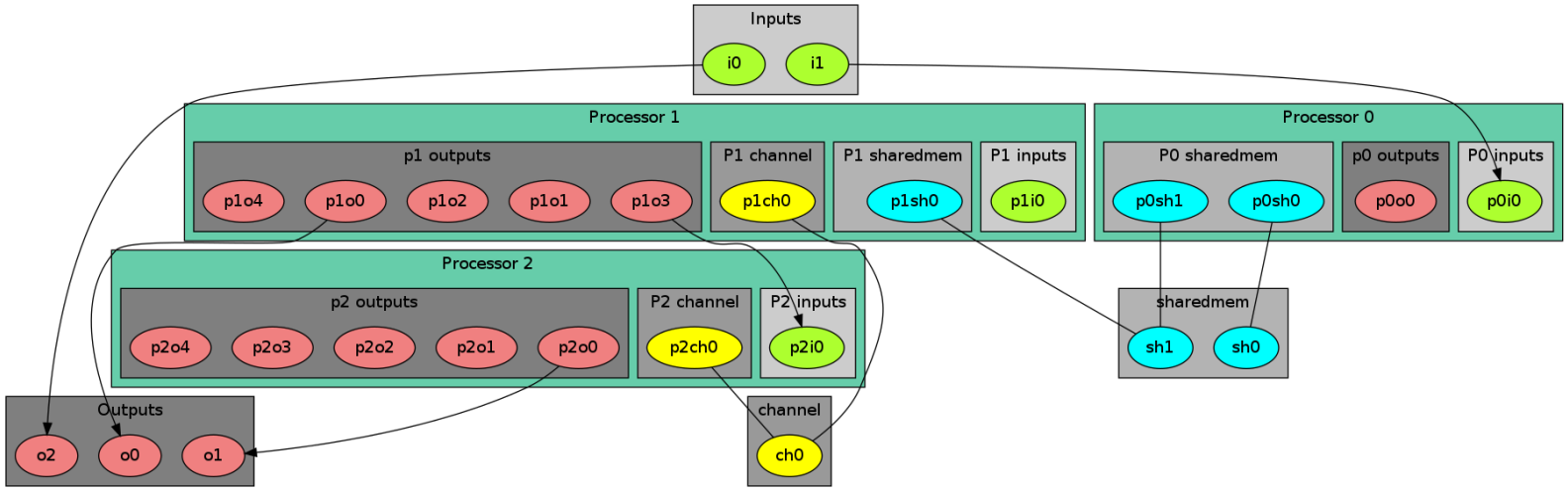
Introducing the BondMachine (BM)

The BondMachine is a software ecosystem for the dynamic generation of computer architectures that:

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

The BondMachine

An example



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)

The non-computational element of the BM

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.

Four kind of SO have been developed so far: the Channel, the Shared Memory, the Barrier and a Pseudo Random Numbers Generator.

Shared Objects (SO)

The non-computational element of the BM

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.

Four kind of SO have been developed so far: the Channel, the Shared Memory, the Barrier and a Pseudo Random Numbers Generator.

Shared Objects (SO)

The non-computational element of the BM

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.

Four kind of SO have been developed so far: the Channel, the Shared Memory, the Barrier and a Pseudo Random Numbers Generator.

Shared Objects (SO)

The non-computational element of the BM

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.

Four kind of SO have been developed so far: the Channel, the Shared Memory, the Barrier and a Pseudo Random Numbers Generator.

Shared Objects (SO)

The non-computational element of the BM

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.

Four kind of SO have been developed so far: the Channel, the Shared Memory, the Barrier and a Pseudo Random Numbers Generator.

Shared Objects (SO)

The non-computational element of the BM

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.

Four kind of SO have been developed so far: the Channel, the Shared Memory, the Barrier and a Pseudo Random Numbers Generator.

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.

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.

Architectures Handling

Handle BM computer architectures

The BM computer architecture is managed by a set of tools to:

- build a specify architecture
- modify a pre-existing architecture
- simulate or emulate the behavior
- Generate the Hardware Description Code (HDL)

Processor Builder

Selects the single processor, assembles and disassembles, saves on disk as JSON, creates the HDL code of a CP

BondMachine Builder

Connects CPs and SOs together in custom topologies, loads and saves on disk as JSON, create BM's HDL code

Simulation Framework

Simulates the behaviour, emulates a BM on a standard Linux workstation

Handle BM computer architectures

The BM computer architecture is managed by a set of tools to:

- build a specify architecture
- modify a pre-existing architecture
- simulate or emulate the behavior
- Generate the Hardware Description Code (HDL)

Processor Builder

Selects the single processor, assembles and disassembles, saves on disk as JSON, creates the HDL code of a CP

BondMachine Builder

Connects CPs and SOs together in custom topologies, loads and saves on disk as JSON, create BM's HDL code

Simulation Framework

Simulates the behaviour, emulates a BM on a standard Linux workstation

Handle BM computer architectures

The BM computer architecture is managed by a set of tools to:

- build a specify architecture
- modify a pre-existing architecture
- simulate or emulate the behavior
- Generate the Hardware Description Code (HDL)

Processor Builder

Selects the single processor, assembles and disassembles, saves on disk as JSON, creates the HDL code of a CP

BondMachine Builder

Connects CPs and SOs together in custom topologies, loads and saves on disk as JSON, create BM's HDL code

Simulation Framework

Simulates the behaviour, emulates a BM on a standard Linux workstation

Handle BM computer architectures

The BM computer architecture is managed by a set of tools to:

- build a specify architecture
- modify a pre-existing architecture
- simulate or emulate the behavior
- Generate the Hardware Description Code (HDL)

Processor Builder

Selects the single processor, assembles and disassembles, saves on disk as JSON, creates the HDL code of a CP

BondMachine Builder

Connects CPs and SOs together in custom topologies, loads and saves on disk as JSON, create BM's HDL code

Simulation Framework

Simulates the behaviour, emulates a BM on a standard Linux workstation

Processor Builder

Procbuilder is the CP manipulation tool.

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

Examples

(32 bit registers counter machine)

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

(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 HDL

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 HDL

Examples

(Assembiling)

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 HDL

Examples

(Create the CP HDL code in Verilog)

procbuilder -create-verilog ...

(Create testbench)

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

Procbuilder demo

Demo N.1

It will be shown how:

- To create a simple processor
- To assemble and disassemble code for it
- To produce its HDL code

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 HDL

Examples

(Add a processor)

bondmachine -add-domains proc.json ... ; ... -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 HDL

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 HDL

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 HDL

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 HDL

Examples

(Visualizing)

bondmachine -emit-dot ...

(Create HDL code)

bondmachine -create-verilog ...

BondMachine demo

Demo N.2

It will be shown how:

- To create a single-core BondMachine
- To attach an external output
- To produce its HDL code

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.

BondMachine demo

Demo N.3

It will be shown how:

- To explore the toolchain
- To flash the board with the code from the previous example

BondMachine demo

Demo N.4

It will be shown how:

- To build a BondMachine with a processor and a shared object
- To flash the board

BondMachine demo

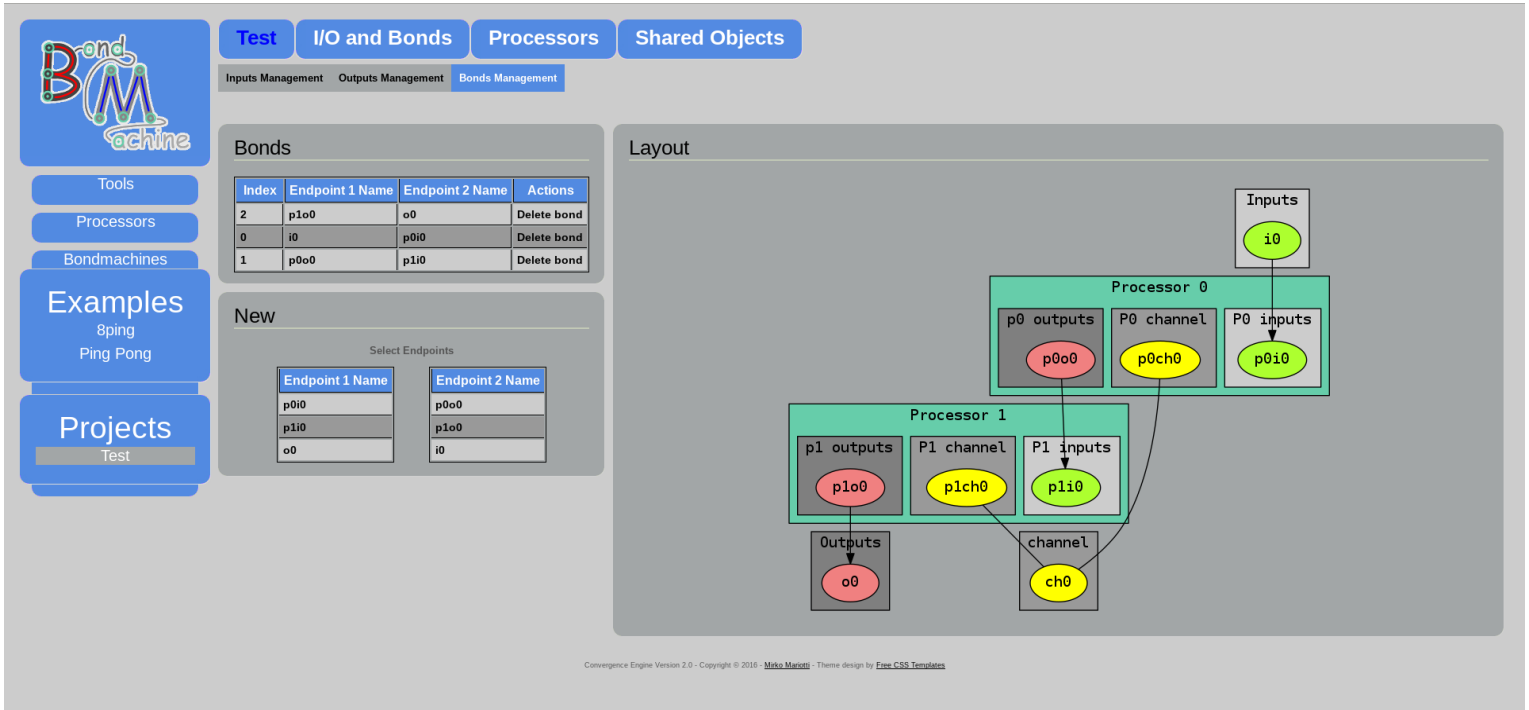
Demo N.5

It will be shown how:

- To build a dual-core BondMachine
- To connect cores
- To flash the board

BondMachine web front-end

Operations on BondMachines can also be performed via an under development 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-i
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
```

Simulation Youtube video

Simulation demo

Demo N.6

It will be shown how:

- To show the simulation capabilities of the framework

Emulation

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

Through the emulator BondMachines can be used on Linux workstations.

Architectures Molding

Molding 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.
- A set of tools to use BondMachine in Machine Learning.

Molding 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.
- A set of tools to use BondMachine in Machine Learning.

Molding 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.
- A set of tools to use BondMachine in Machine Learning.

Molding 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.
- A set of tools to use BondMachine in Machine Learning.

Molding 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.
- A set of tools to use BondMachine in Machine Learning.

Molding 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.
- A set of tools to use BondMachine in Machine Learning.

Use the BM computer architecture

Mapping specific computational problems to BMs

Symbond

Map symbolic mathematical expressions to BM

Boolbond

Map boolean systems to BM

Matrixwork

Basic matrix computation

Evolutive BM

Evolutionary computing to BM

Bondgo

The architecture compiler

ML tools

Map computational graphs to BM

Use the BM computer architecture

Mapping specific computational problems to BMs

Symbond

Map symbolic mathematical expressions to BM

Boolbond

Map boolean systems to BM

Matrixwork

Basic matrix computation

Evolutive BM

Evolutionary computing to BM

Bondgo

The architecture compiler

ML tools

Map computational graphs to BM

Use the BM computer architecture

Mapping specific computational problems to BMs

Symbond

Map symbolic mathematical expressions to BM

Boolbond

Map boolean systems to BM

Matrixwork

Basic matrix computation

Evolutive BM

Evolutionary computing to BM

Bondgo

The architecture compiler

ML tools

Map computational graphs to BM

Use the BM computer architecture

Mapping specific computational problems to BMs

Symbond

Map symbolic mathematical expressions to BM

Boolbond

Map boolean systems to BM

Matrixwork

Basic matrix computation

Evolutive BM

Evolutionary computing to BM

Bondgo

The architecture compiler

ML tools

Map computational graphs to BM

Use the BM computer architecture

Mapping specific computational problems to BMs

Symbond

Map symbolic mathematical expressions to BM

Boolbond

Map boolean systems to BM

Matrixwork

Basic matrix computation

Evolutive BM

Evolutionary computing to BM

Bondgo

The architecture compiler

ML tools

Map computational graphs to BM

Use the BM computer architecture

Mapping specific computational problems to BMs

Symbond

Map symbolic mathematical expressions to BM

Boolbond

Map boolean systems to BM

Matrixwork

Basic matrix computation

Evolutive BM

Evolutionary computing to BM

Bondgo

The architecture compiler

ML tools

Map computational graphs to BM

Use the BM computer architecture

Mapping specific computational problems to BMs

Symbond

Map symbolic mathematical expressions to BM

Boolbond

Map boolean systems to BM

Matrixwork

Basic matrix computation

Evolutive BM

Evolutionary computing to BM

Bondgo

The architecture compiler

ML tools

Map computational graphs to BM

Bondgo

Bondgo

The major innovation of the BondMachine Project is its compiler.

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

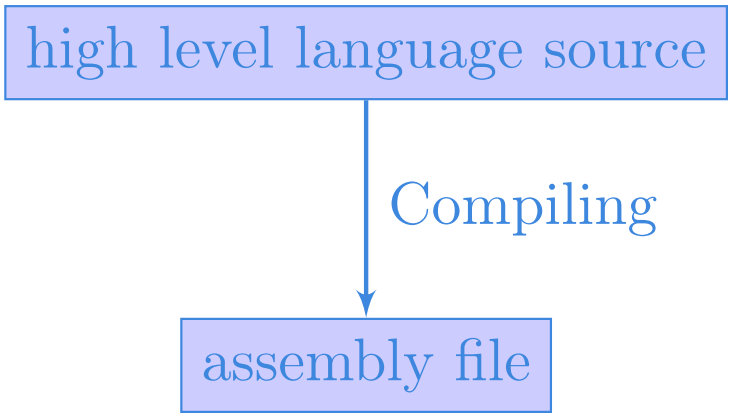
Bondgo

This is the standard flow when building computer programs

high level language source

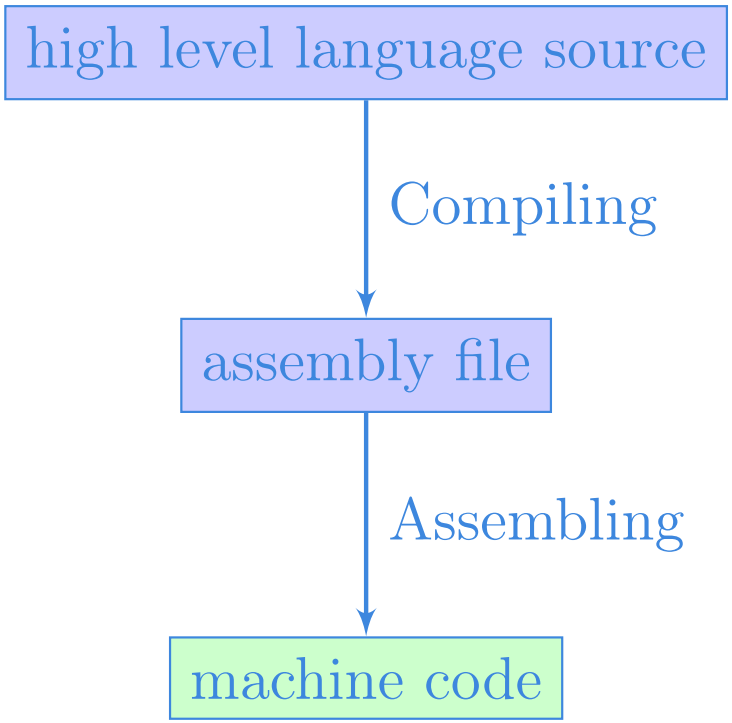
Bondgo

This is the standard flow when building computer programs



Bondgo

This is the standard flow when building computer programs



Bondgo

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 does something different from standard compilers ...

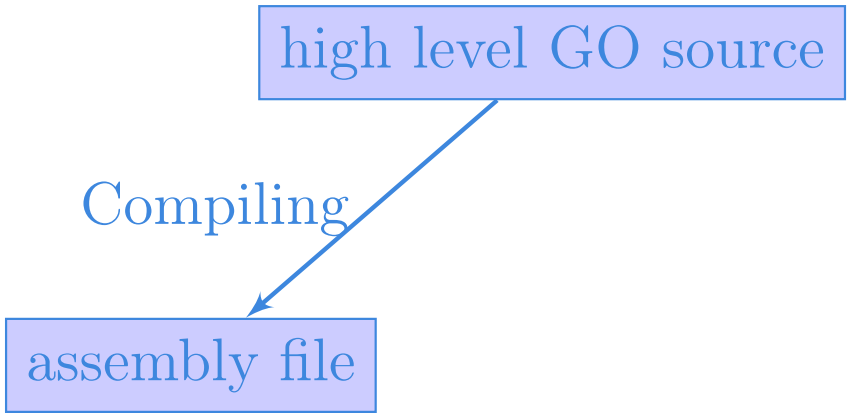
Bondgo

Bondgo does something different from standard compilers ...

high level GO source

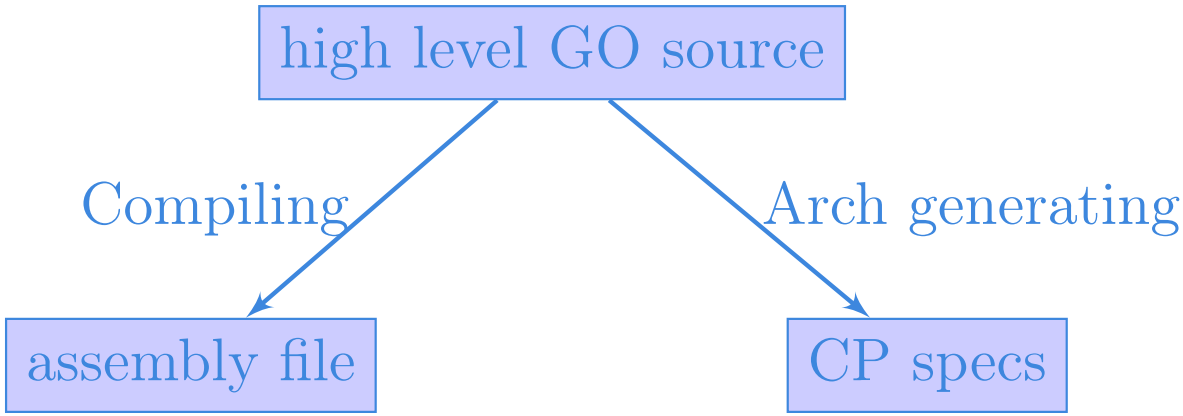
Bondgo

Bondgo does something different from standard compilers ...



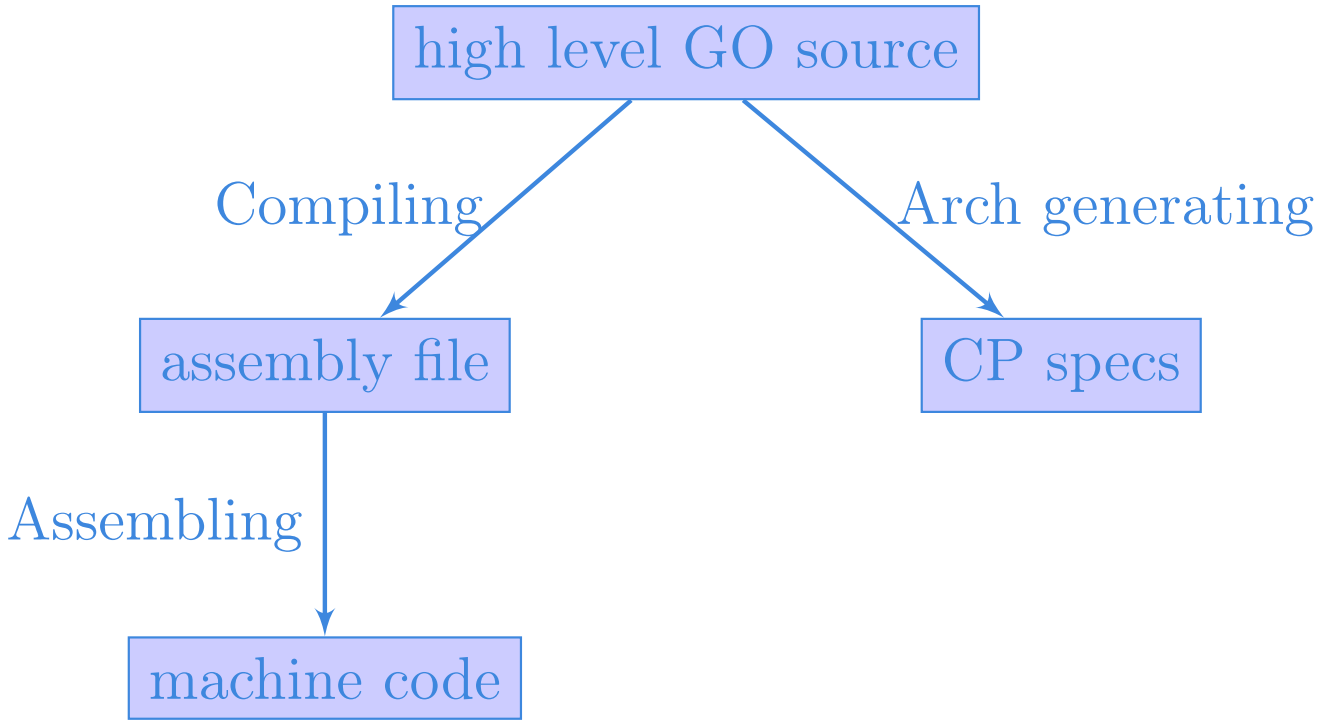
Bondgo

Bondgo does something different from standard compilers ...



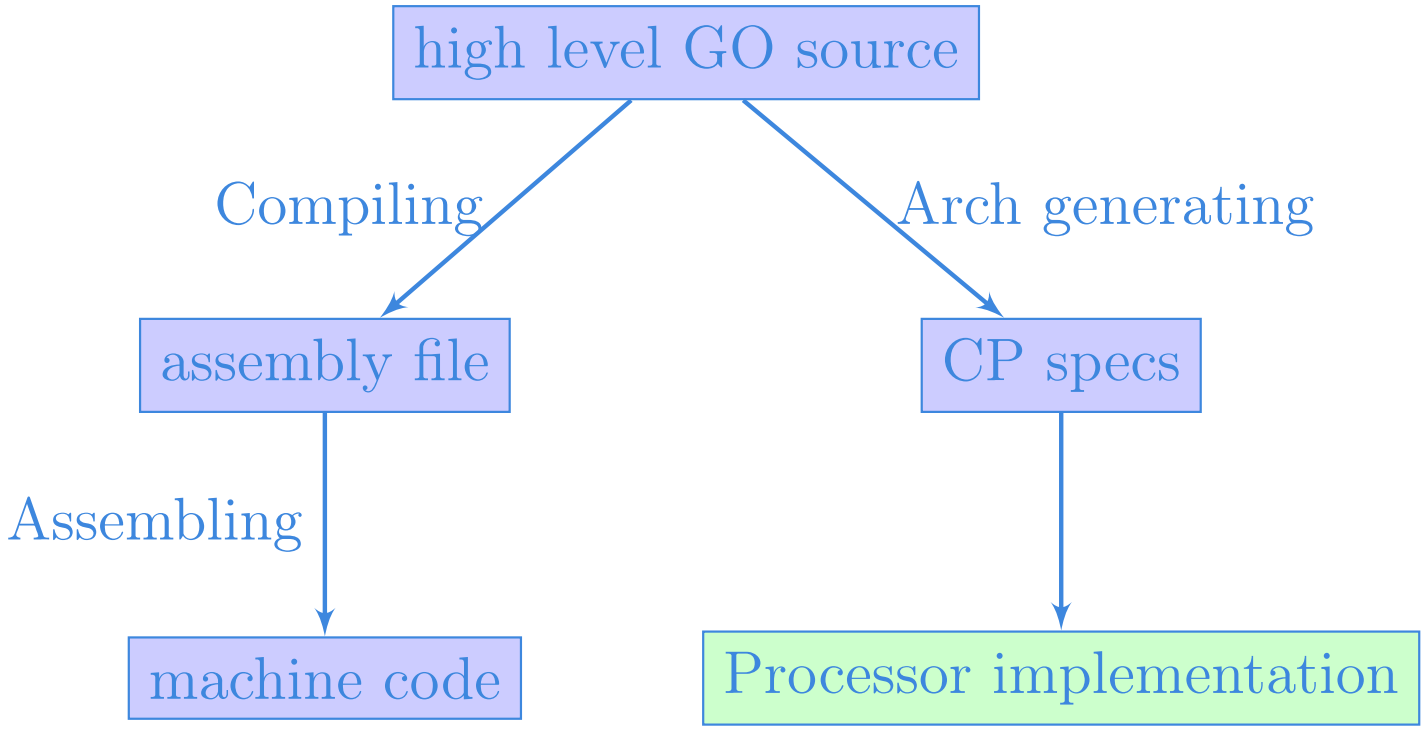
Bondgo

Bondgo does something different from standard compilers ...



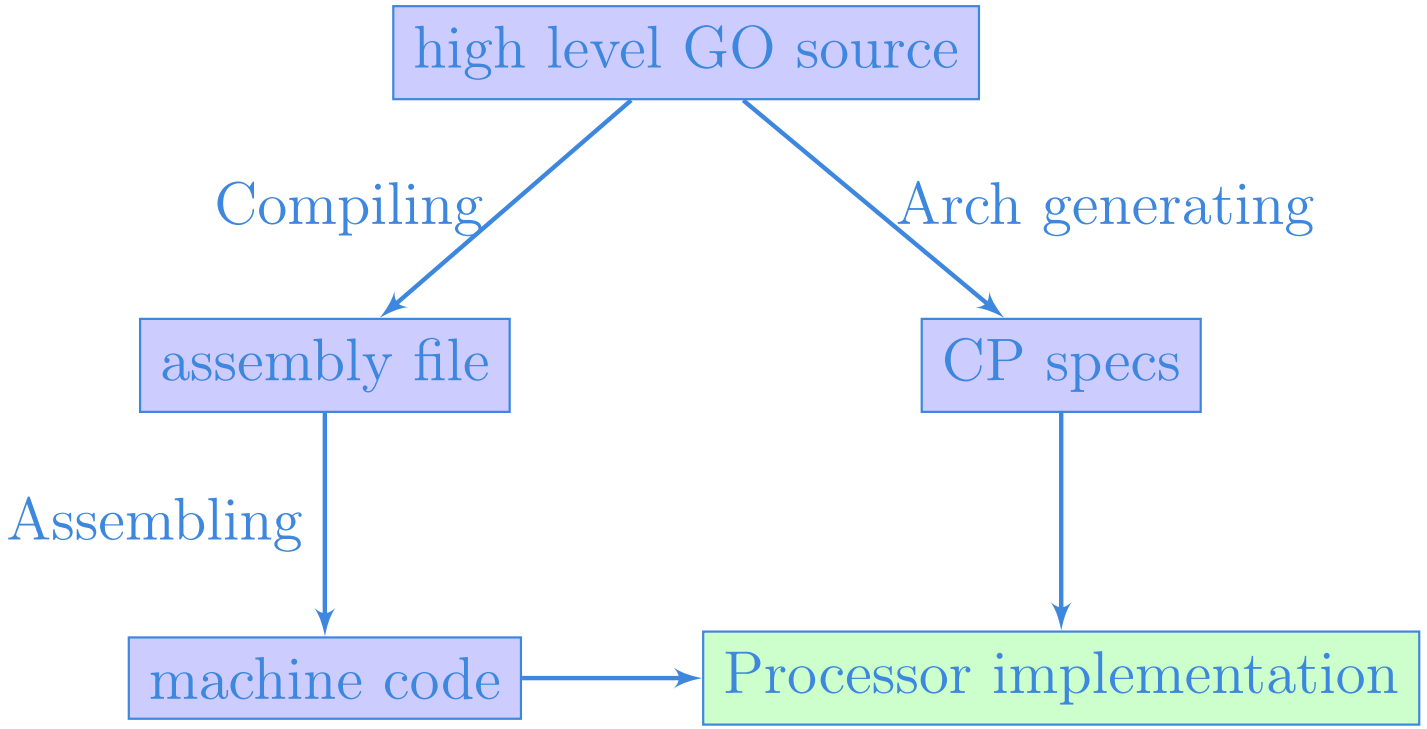
Bondgo

Bondgo does something different from standard compilers ...



Bondgo

Bondgo does something different from standard compilers ...

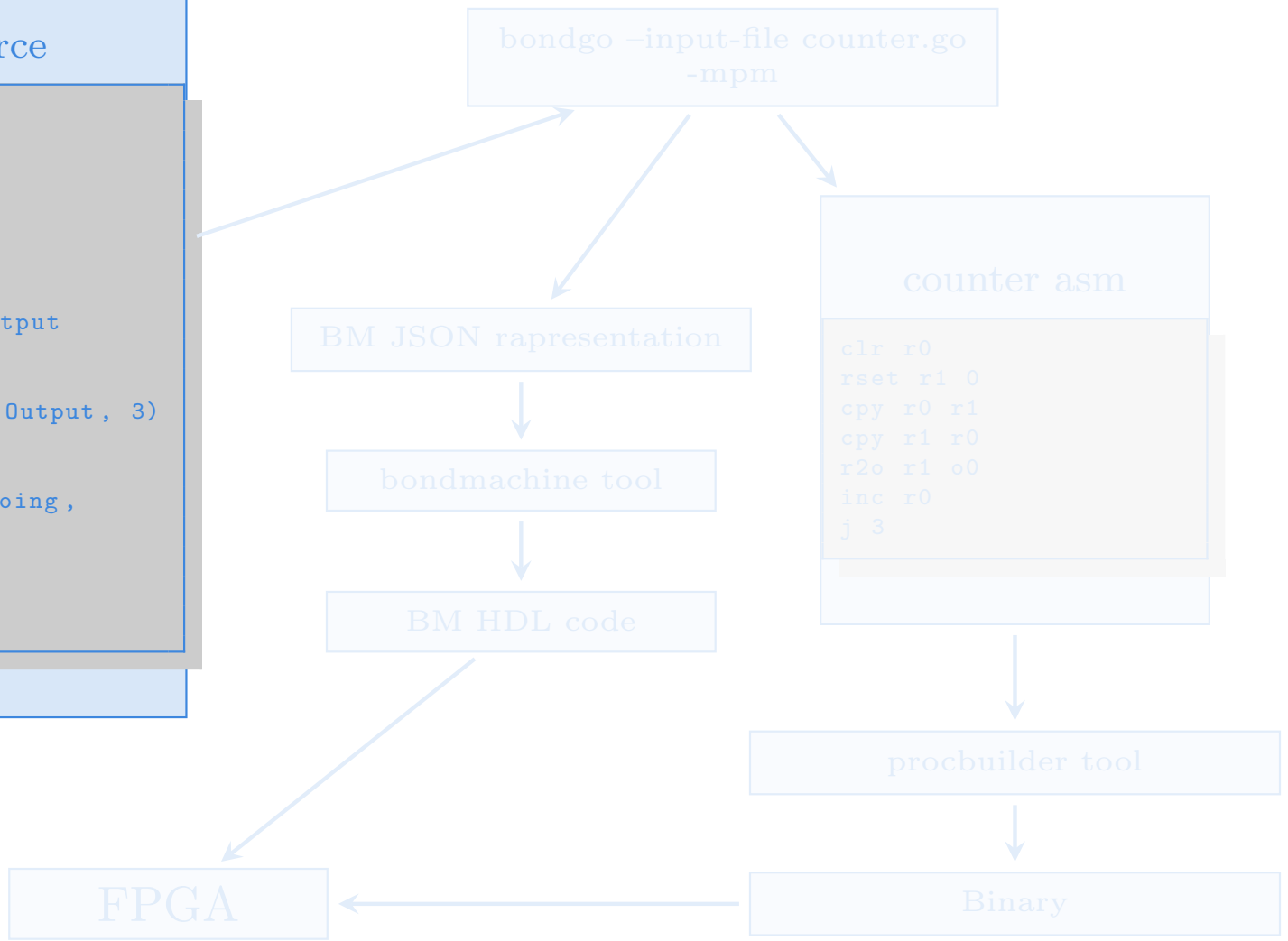


Bondgo

A first example

counter go source

```
package main

import (
    "bondgo"
)

func main() {
    var outgoing bondgo.Output
    var reg_int uint8
    outgoing =
        bondgo.Make(bondgo.Output, 3)
    reg_int = 0
    for {
        bondgo.IOWrite(outgoing,
        reg_int)
        reg_int++
    }
}
```



Bondgo

A first example

counter go source

```
package main

import (
    "bondgo"
)

func main() {
    var outgoing bondgo.Output
    var reg_int uint8
    outgoing =
        bondgo.Make(bondgo.Output, 3)
    reg_int = 0
    for {
        bondgo.IOWrite(outgoing,
        reg_int)
        reg_int++
    }
}
```

bondgo –input-file counter.go -mpm

BM JSON rapresentation

bondmachine tool

BM HDL code

counter asm

```
clr r0
rset r1 0
cpy r0 r1
cpy r1 r0
r2o r1 o0
inc r0
j 3
```

procbuilder tool

Binary

FPGA

Bondgo

A first example

counter go source

```
package main

import (
    "bondgo"
)

func main() {
    var outgoing bondgo.Output
    var reg_int uint8
    outgoing =
        bondgo.Make(bondgo.Output, 3)
    reg_int = 0
    for {
        bondgo.IOWrite(outgoing,
        reg_int)
        reg_int++
    }
}
```

bondgo –input-file counter.go -mpm

BM JSON rapresentation

bondmachine tool

BM HDL code

FPGA

counter asm

```
clr r0
rset r1 0
cpy r0 r1
cpy r1 r0
r2o r1 o0
inc r0
j 3
```

procbuilder tool

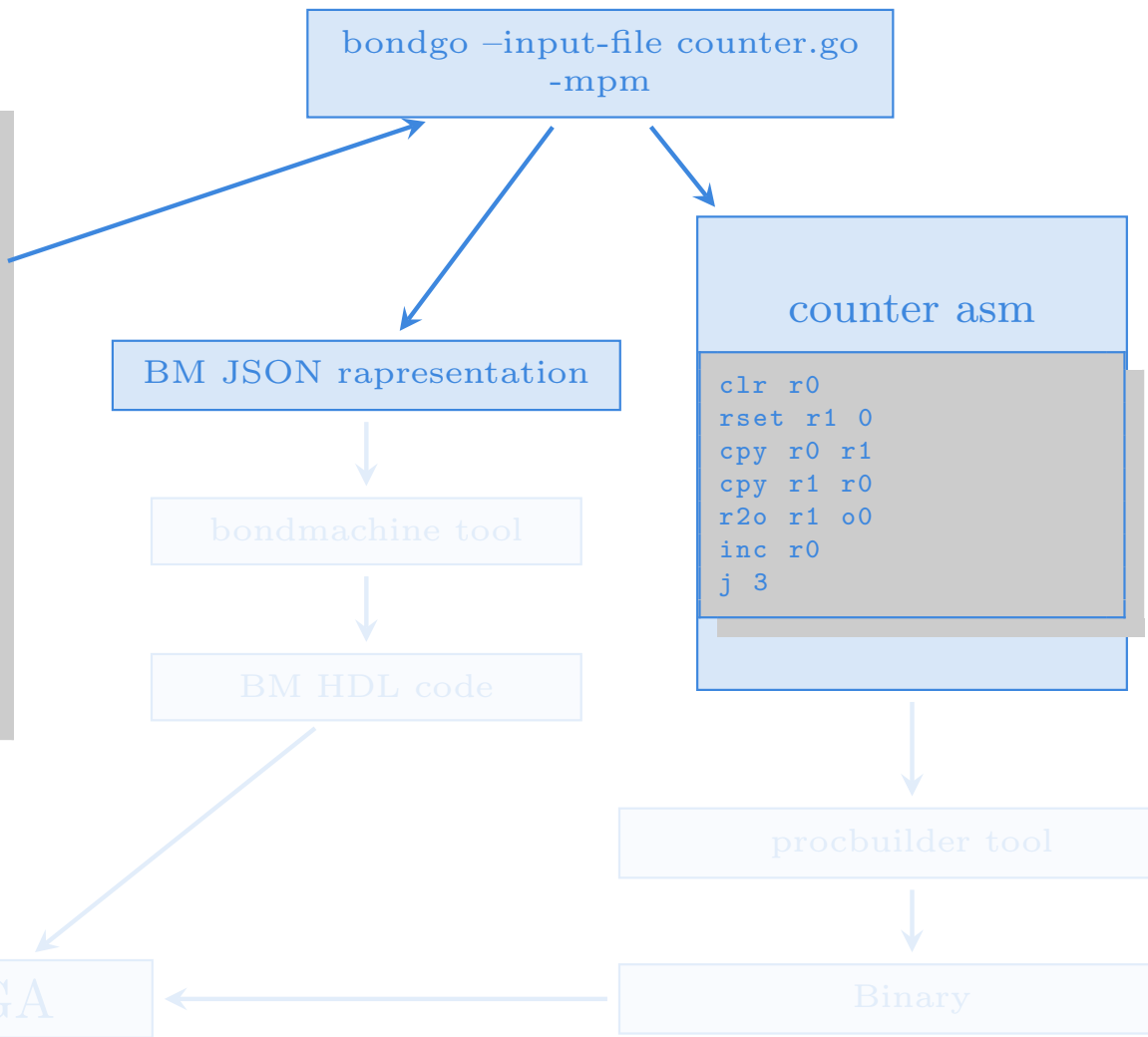
Binary

Bondgo

A first example

counter go source

```
package main

import (
    "bondgo"
)

func main() {
    var outgoing bondgo.Output
    var reg_int uint8
    outgoing =
        bondgo.Make(bondgo.Output, 3)
    reg_int = 0
    for {
        bondgo.IOWrite(outgoing,
        reg_int)
        reg_int++
    }
}
```

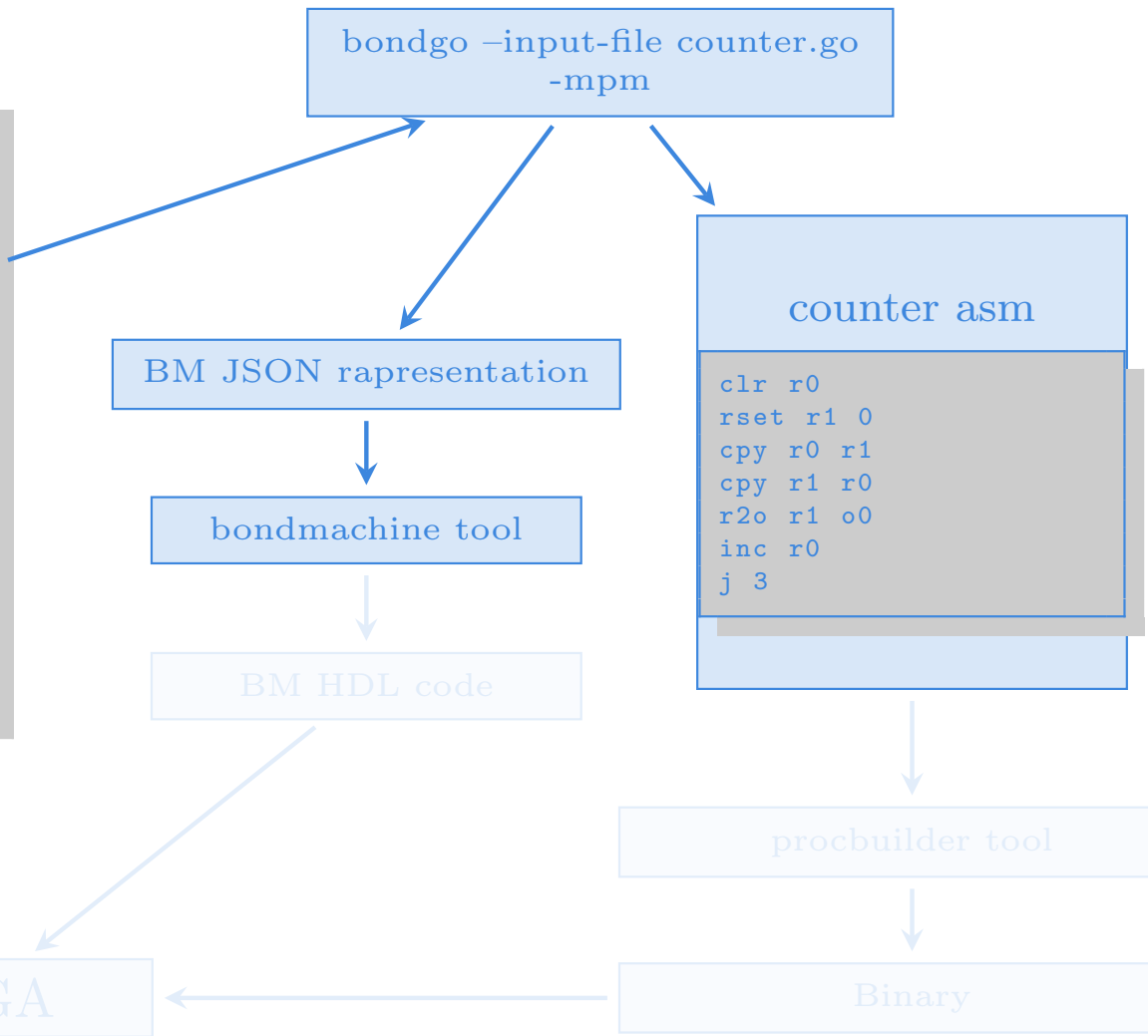


Bondgo

A first example

counter go source

```
package main

import (
    "bondgo"
)

func main() {
    var outgoing bondgo.Output
    var reg_int uint8
    outgoing =
        bondgo.Make(bondgo.Output, 3)
    reg_int = 0
    for {
        bondgo.IOWrite(outgoing,
        reg_int)
        reg_int++
    }
}
```

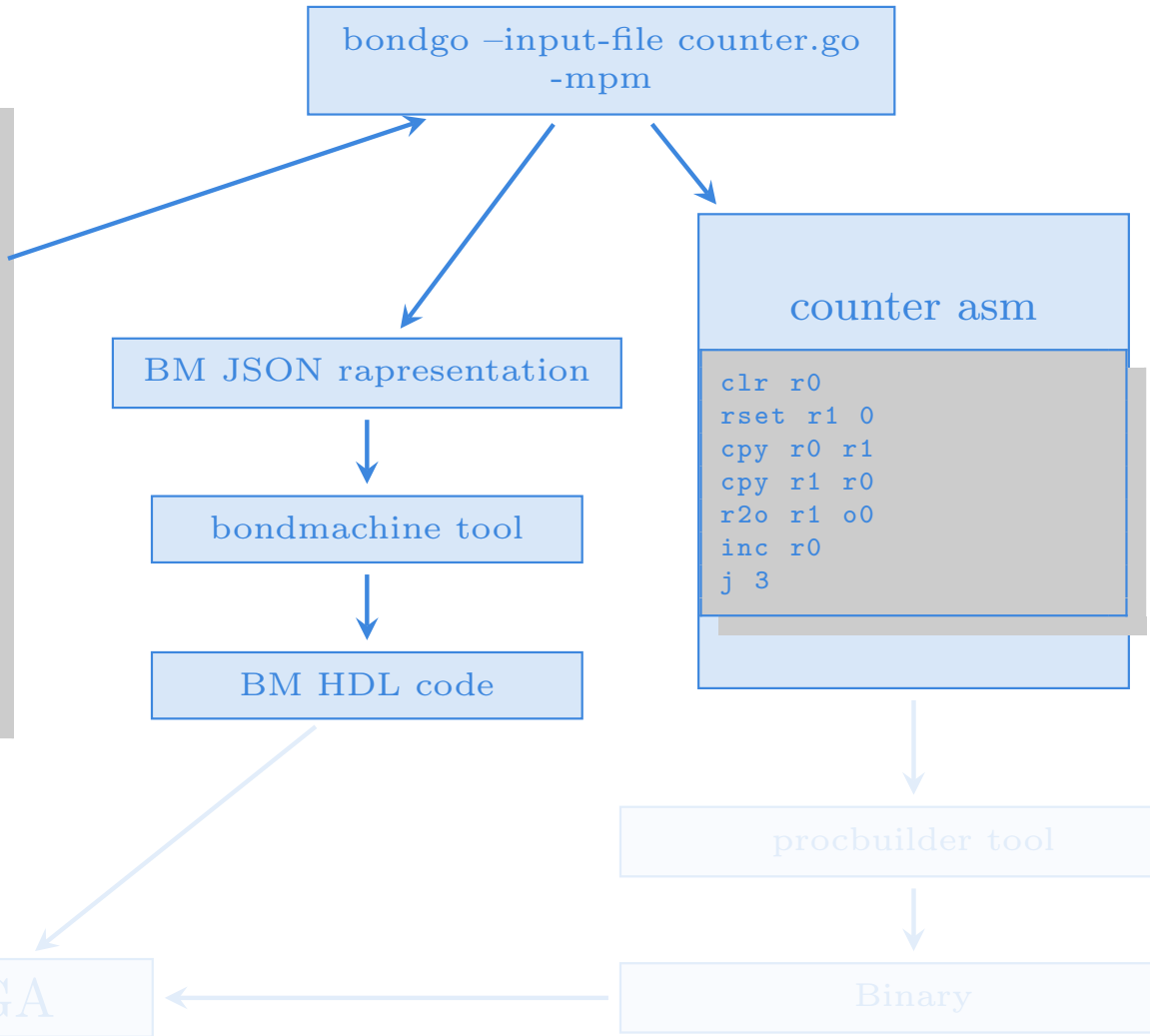


Bondgo

A first example

counter go source

```
package main

import (
    "bondgo"
)

func main() {
    var outgoing bondgo.Output
    var reg_int uint8
    outgoing =
        bondgo.Make(bondgo.Output, 3)
    reg_int = 0
    for {
        bondgo.IOWrite(outgoing,
        reg_int)
        reg_int++
    }
}
```



Bondgo

A first example

counter go source

```
package main

import (
    "bondgo"
)

func main() {
    var outgoing bondgo.Output
    var reg_int uint8
    outgoing =
        bondgo.Make(bondgo.Output, 3)
    reg_int = 0
    for {
        bondgo.IOWrite(outgoing,
        reg_int)
        reg_int++
    }
}
```

bondgo –input-file counter.go -mpm

BM JSON rapresentation

bondmachine tool

BM HDL code

FPGA

counter asm

```
clr r0
rset r1 0
cpy r0 r1
cpy r1 r0
r2o r1 o0
inc r0
j 3
```

procbuilder tool

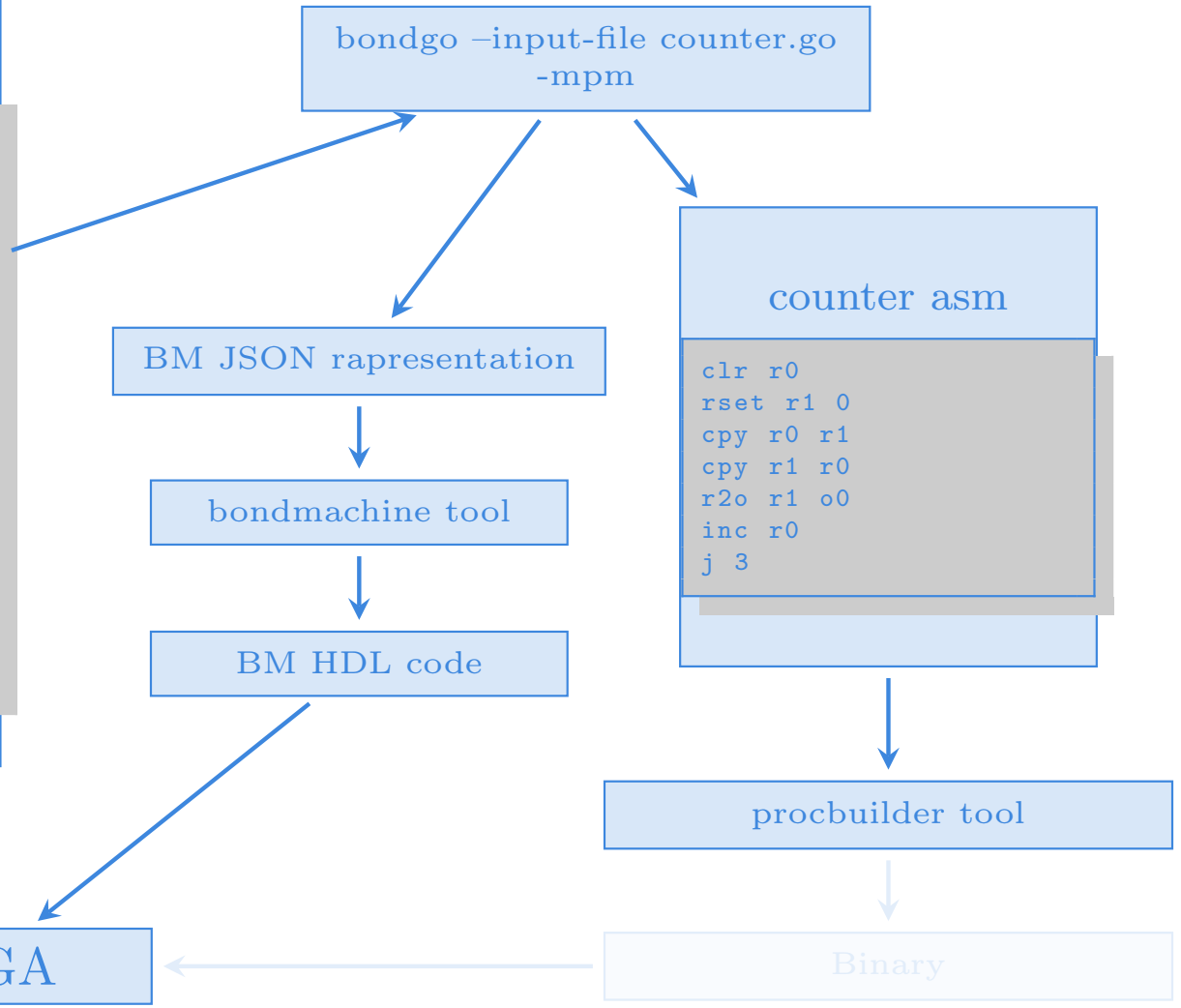
Binary

Bondgo

A first example

counter go source

```
package main

import (
    "bondgo"
)

func main() {
    var outgoing bondgo.Output
    var reg_int uint8
    outgoing =
        bondgo.Make(bondgo.Output, 3)
    reg_int = 0
    for {
        bondgo.IOWrite(outgoing,
        reg_int)
        reg_int++
    }
}
```



Bondgo

A first example

counter go source

```
package main

import (
    "bondgo"
)

func main() {
    var outgoing bondgo.Output
    var reg_int uint8
    outgoing =
        bondgo.Make(bondgo.Output, 3)
    reg_int = 0
    for {
        bondgo.IOWrite(outgoing,
        reg_int)
        reg_int++
    }
}
```

bondgo –input-file counter.go -mpm

BM JSON rapresentation

bondmachine tool

BM HDL code

FPGA

counter asm

```
clr r0
rset r1 0
cpy r0 r1
cpy r1 r0
r2o r1 o0
inc r0
j 3
```

procbuilder tool

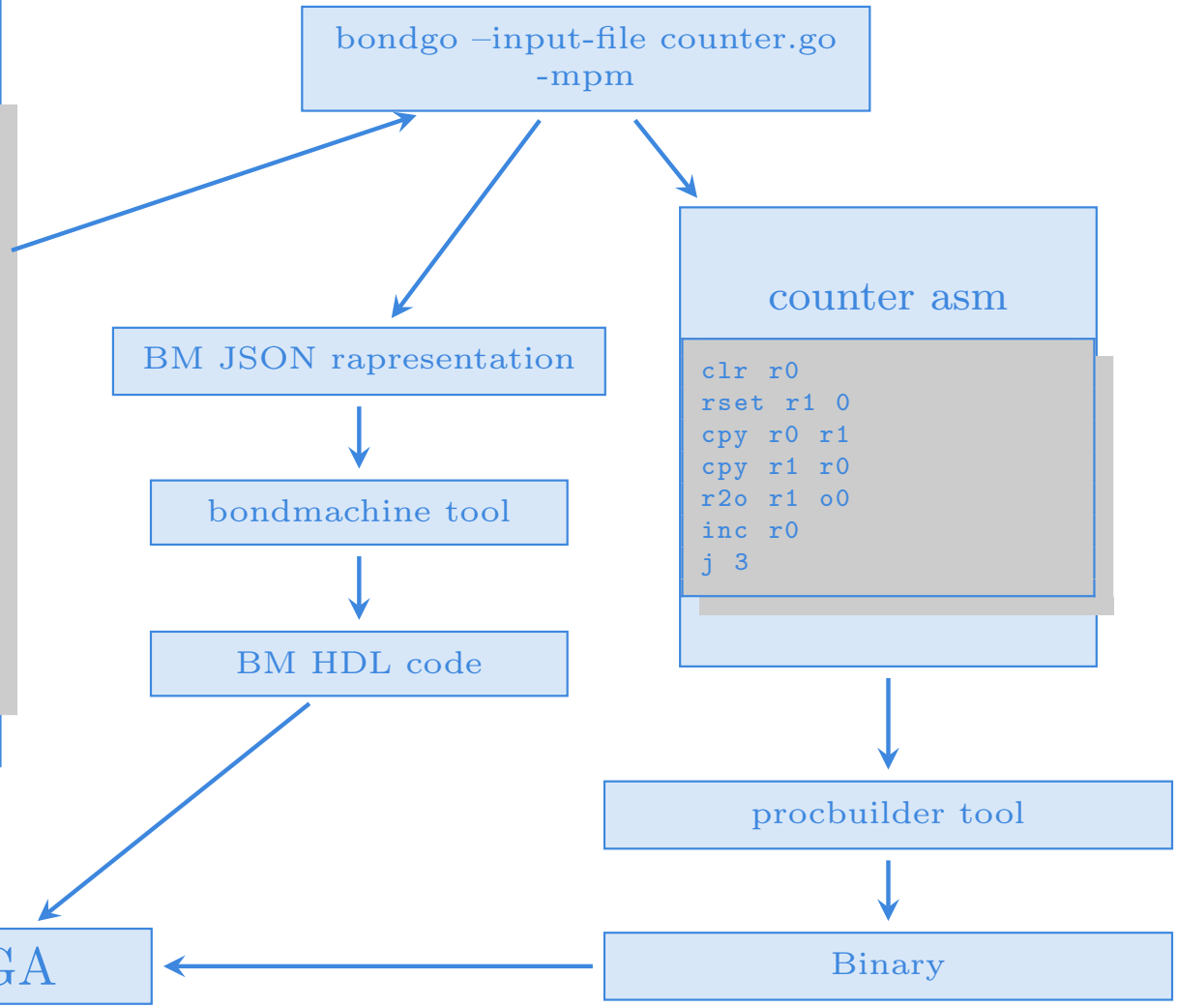
Binary

Bondgo

A first example

counter go source

```
package main

import (
    "bondgo"
)

func main() {
    var outgoing bondgo.Output
    var reg_int uint8
    outgoing =
        bondgo.Make(bondgo.Output, 3)
    reg_int = 0
    for {
        bondgo.IOWrite(outgoing,
        reg_int)
        reg_int++
    }
}
```



Bondgo demo

Demo N.7

It will be shown how:

- To create a BondMachine from a Go source file
- To build the architecture
- To build the program
- To create the firmware and flash it to the board

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.

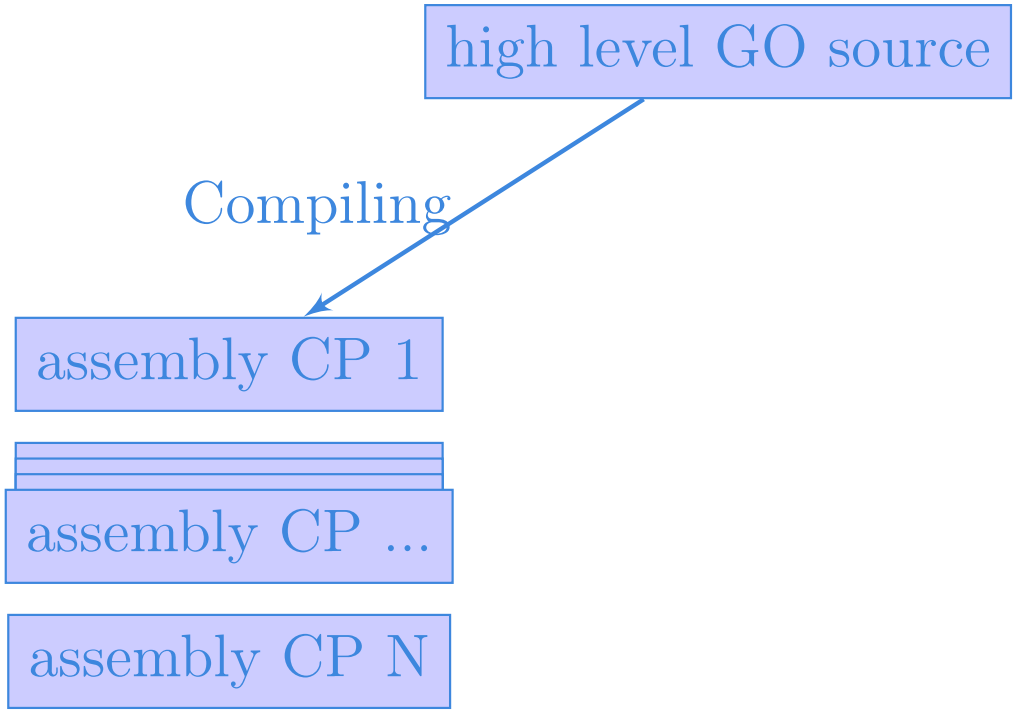
Bondgo

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

high level GO source

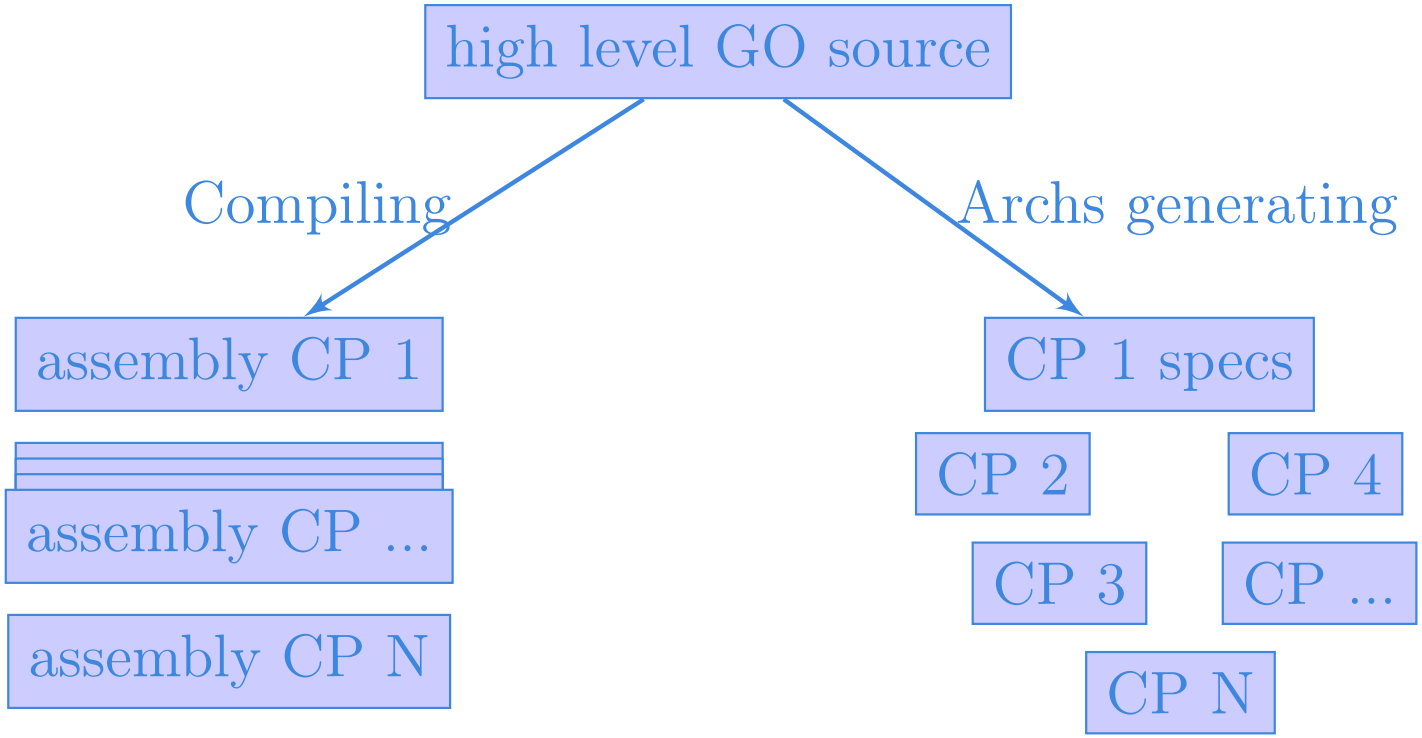
Bondgo

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



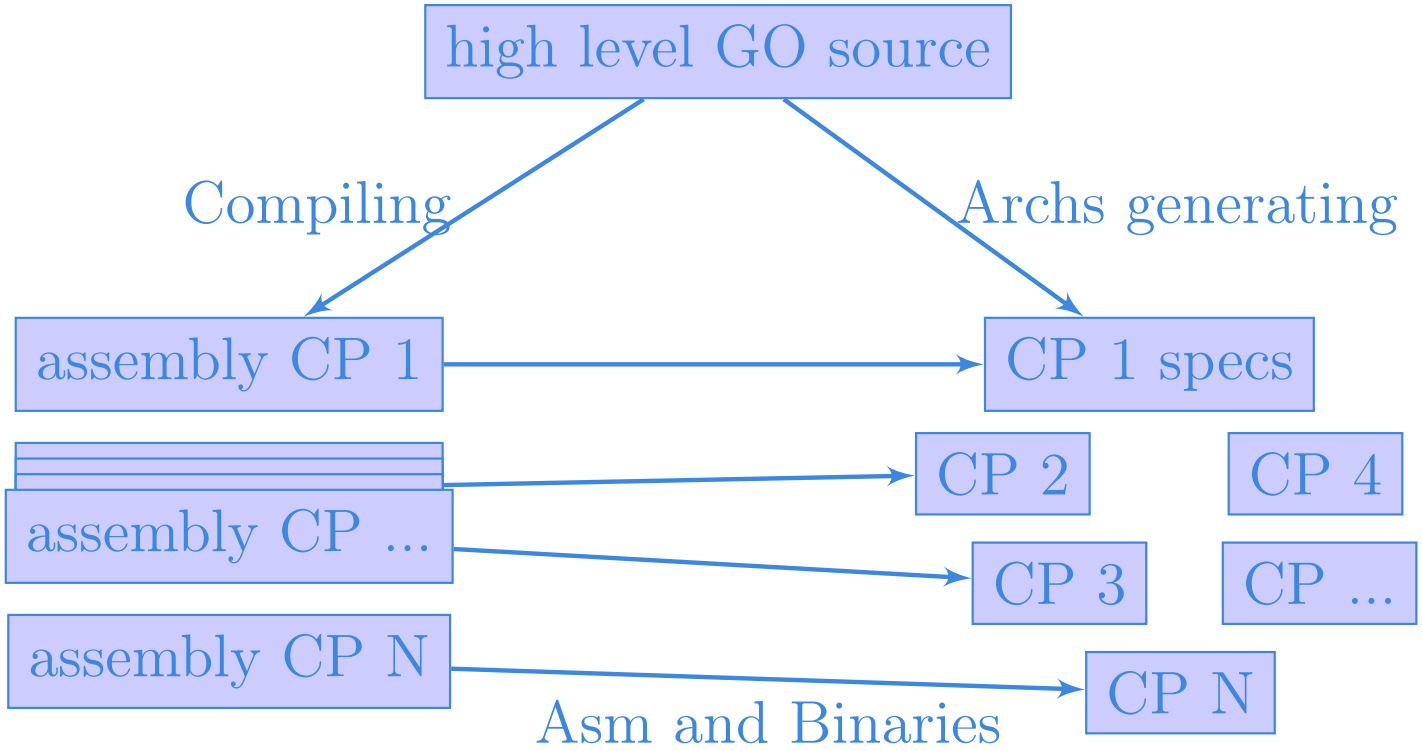
Bondgo

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



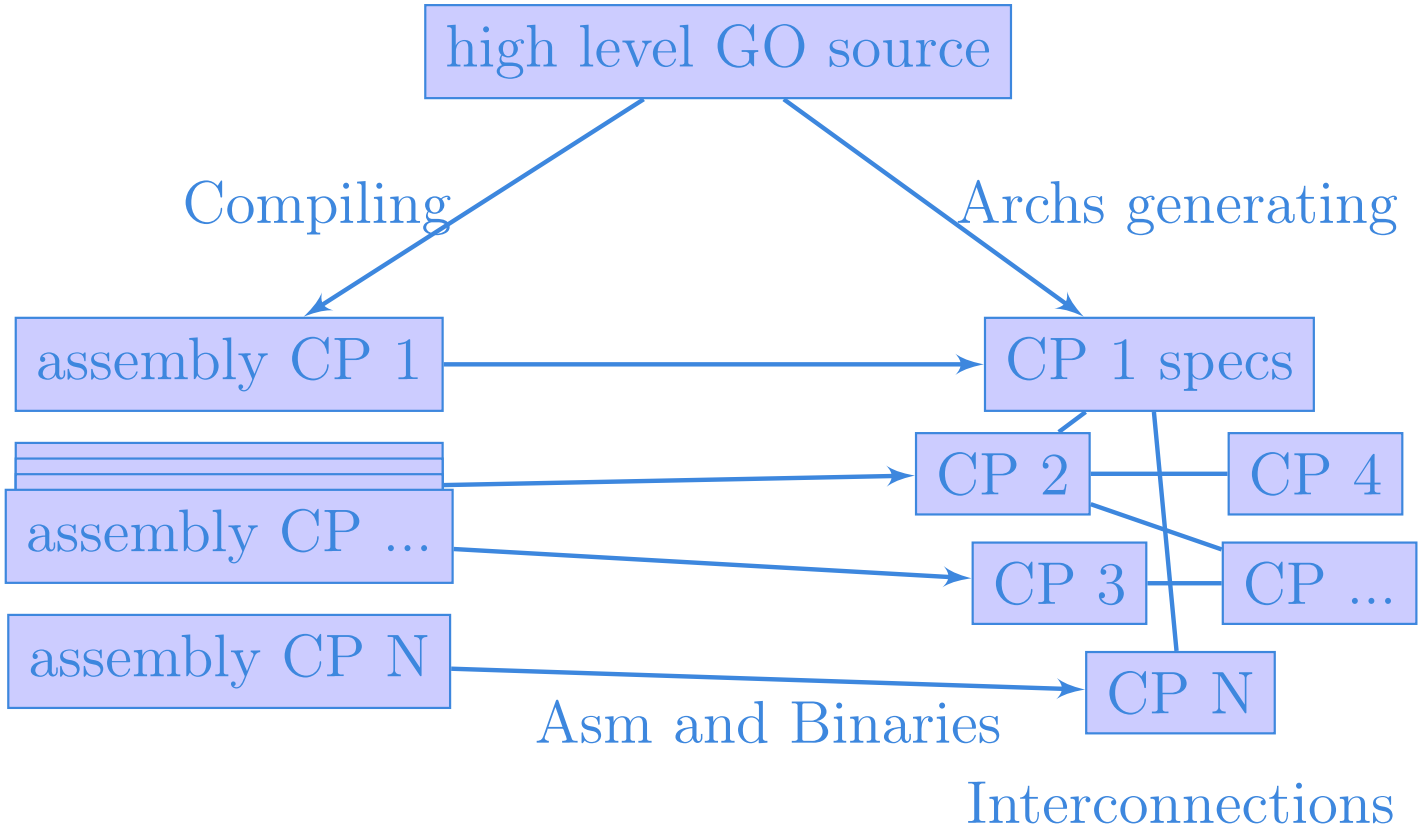
Bondgo

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



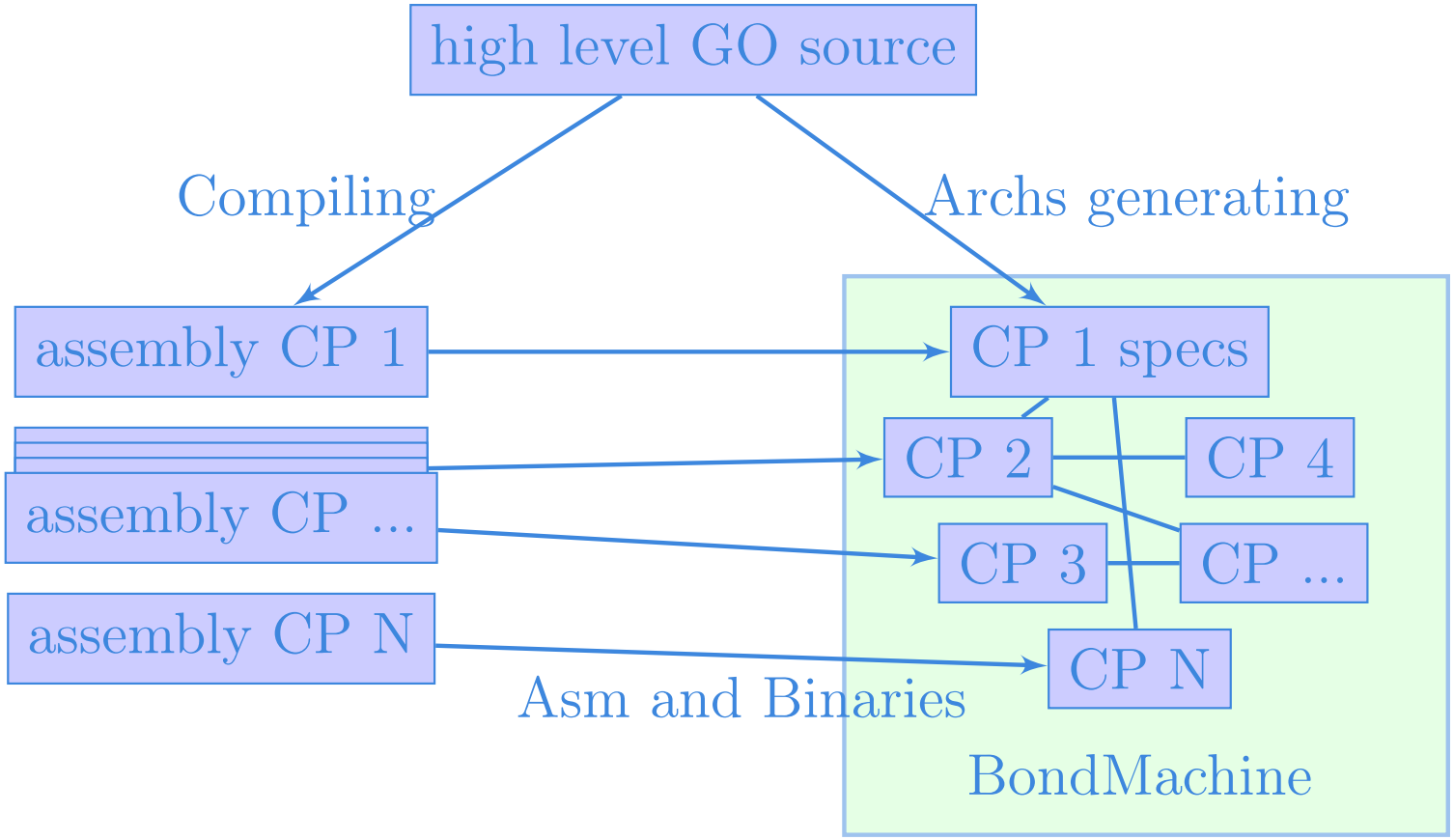
Bondgo

... 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

A multi-core example

multi-core 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)
   }
}

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

A multi-core example

Compiling the code with the bondgo compiler:

bondgo -input-file ds.go -mpm

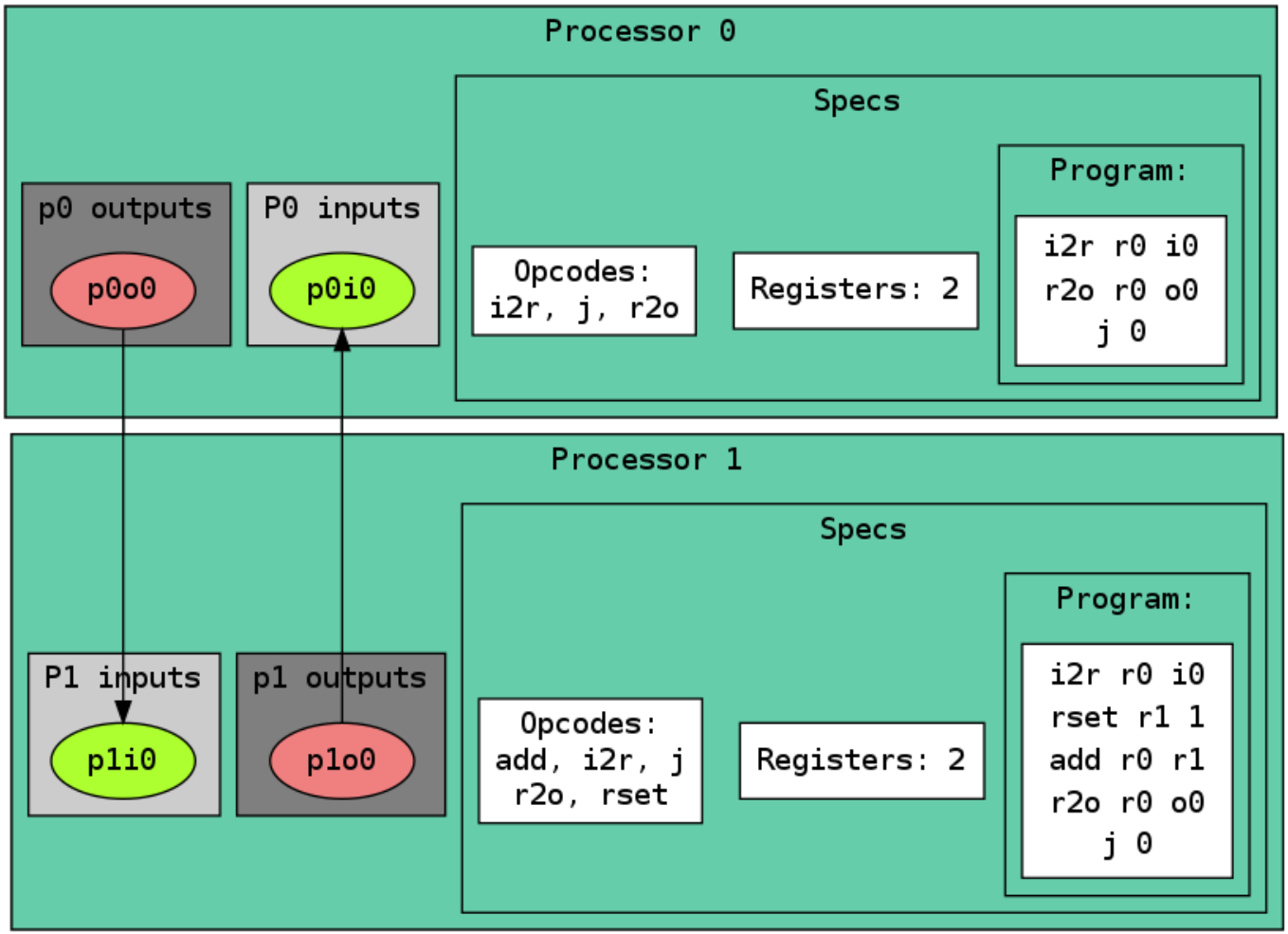
The toolchain perform the following steps:

- 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:

Bondgo

A multi-core example



Simulation demo

Demo N.8

It will be shown how:

- To use bondgo to create a chain of interconnected processors
- To flash the firmware to the board

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".

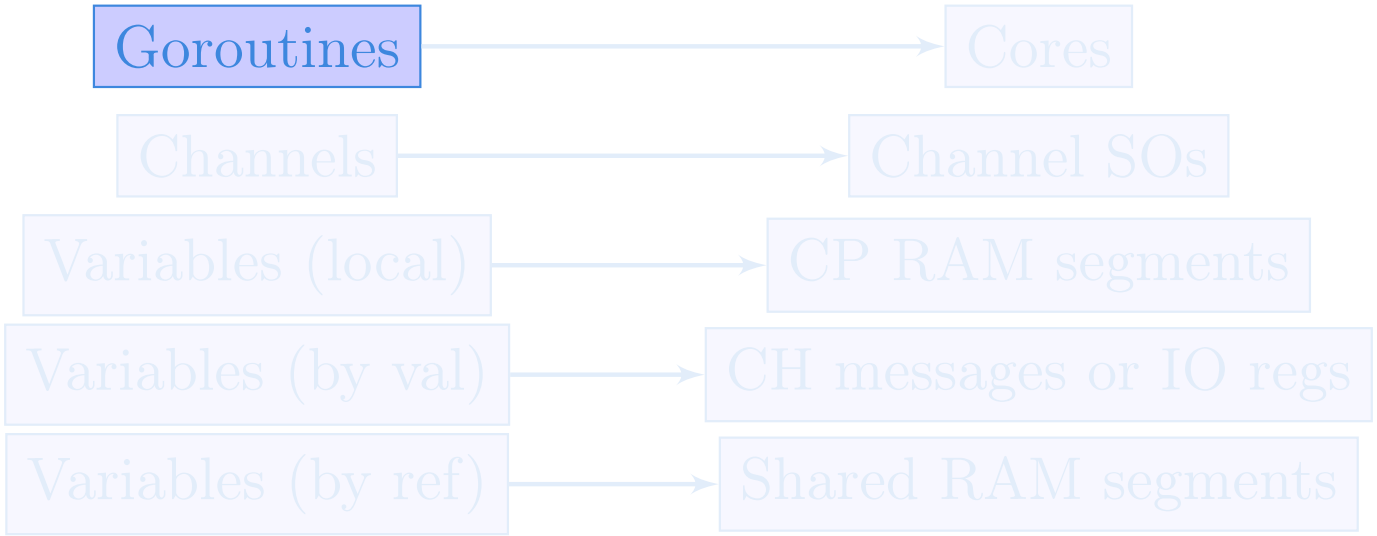
High level Go source code is directly mapped to interconnected processors without Operating Systems or runtimes.

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.

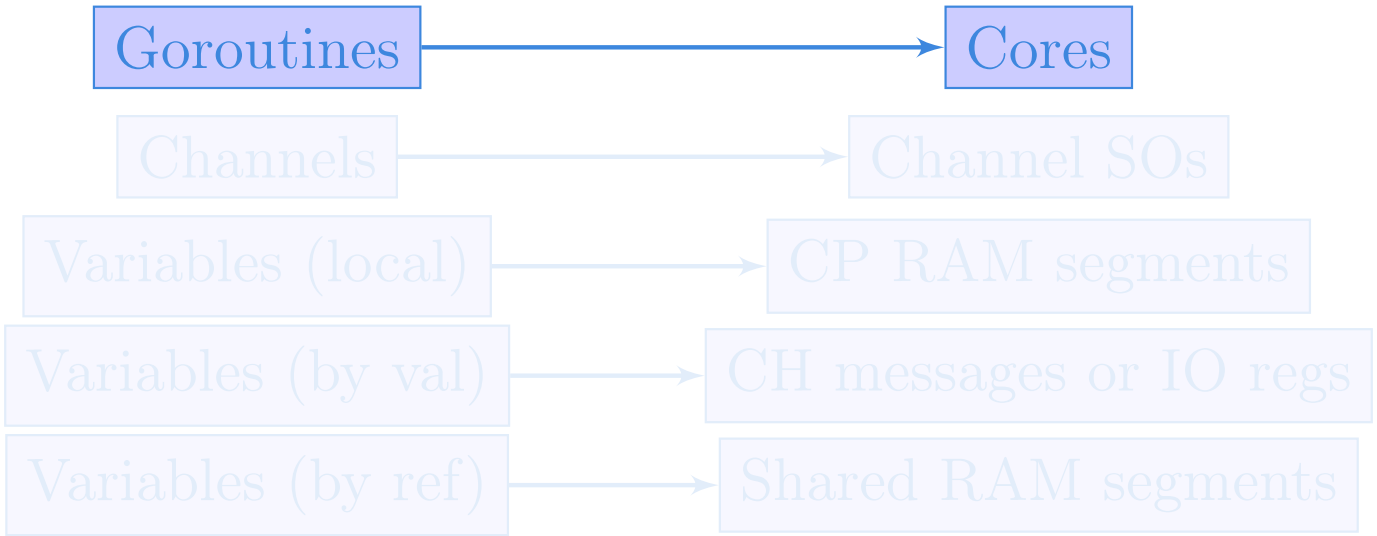


Bondgo

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.

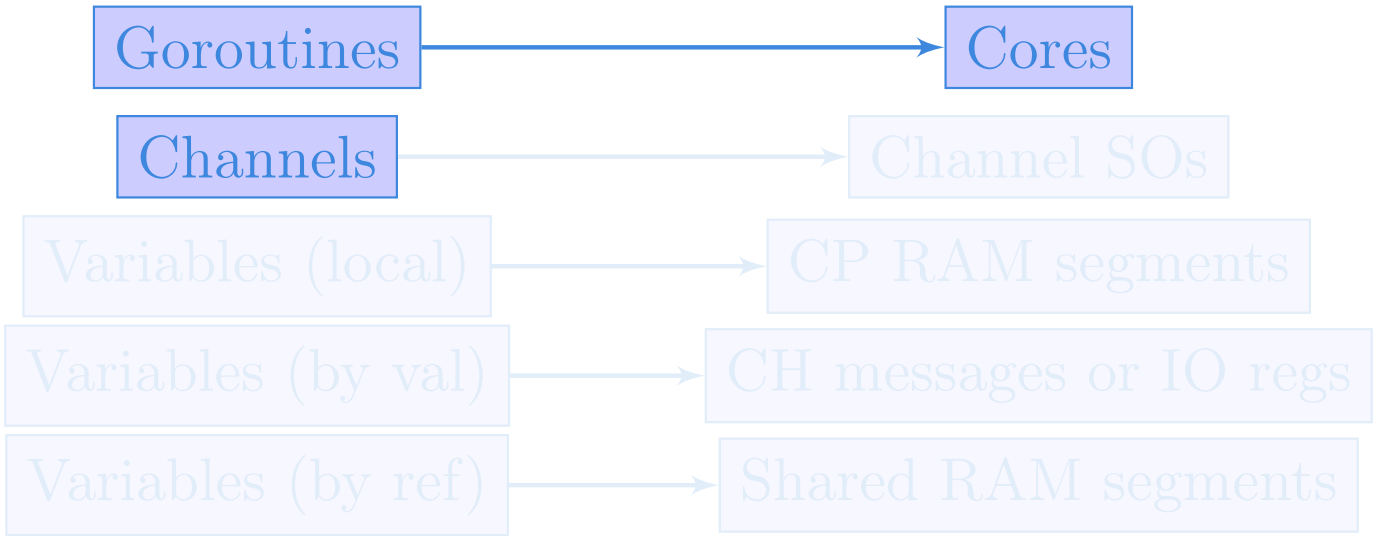


Bondgo

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.

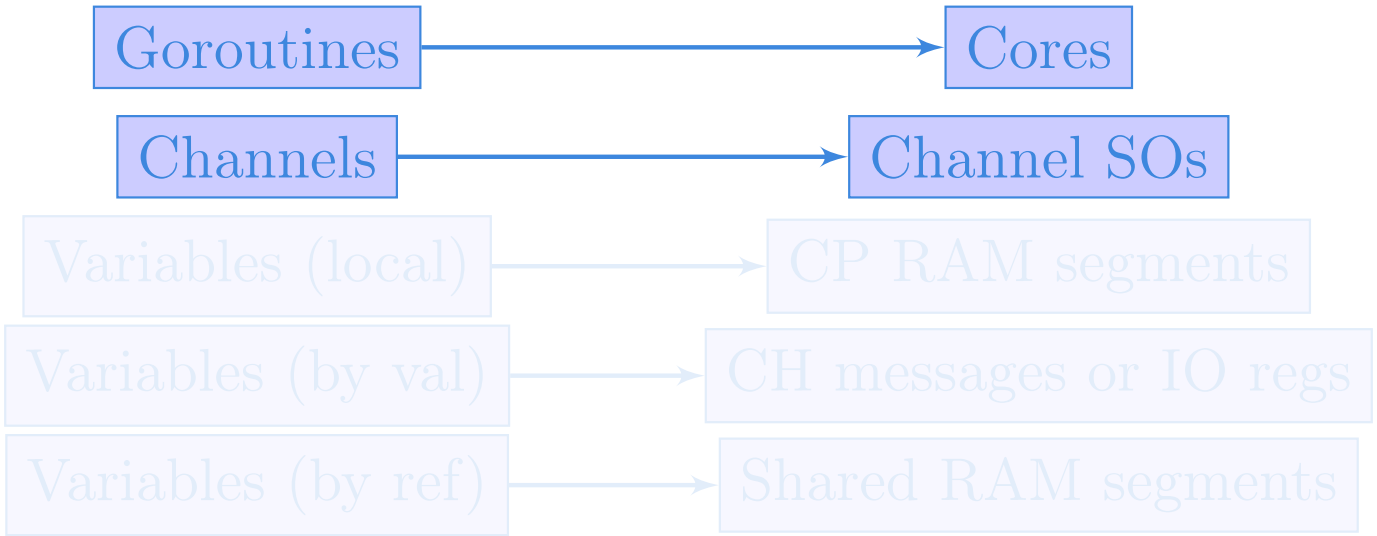


Bondgo

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.

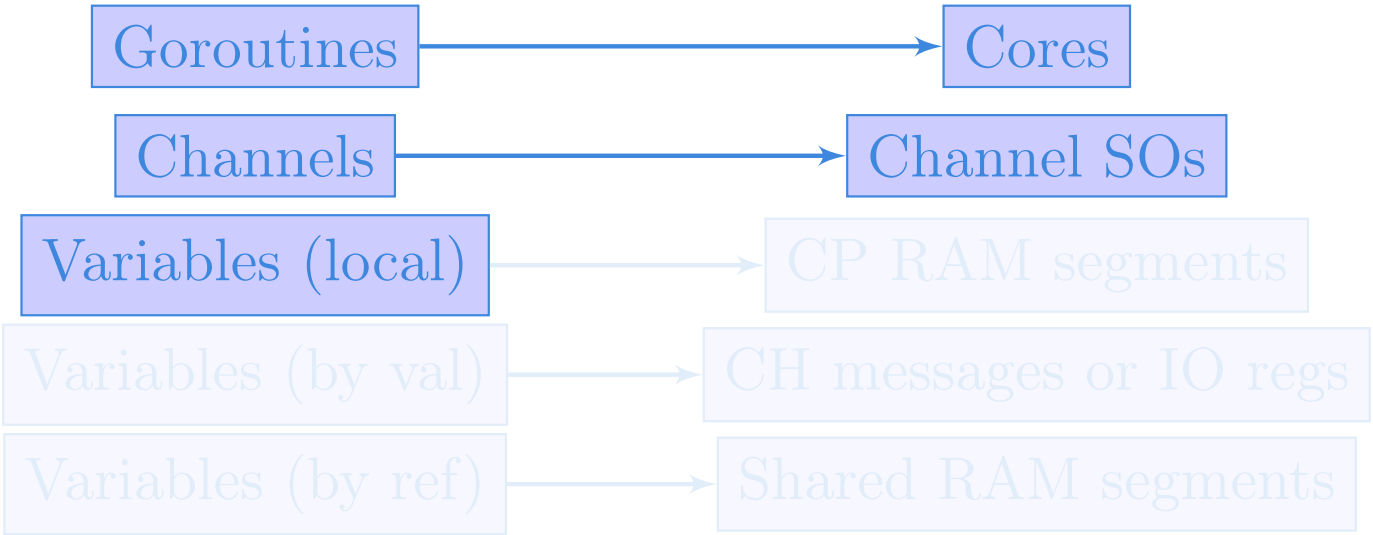


Bondgo

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.

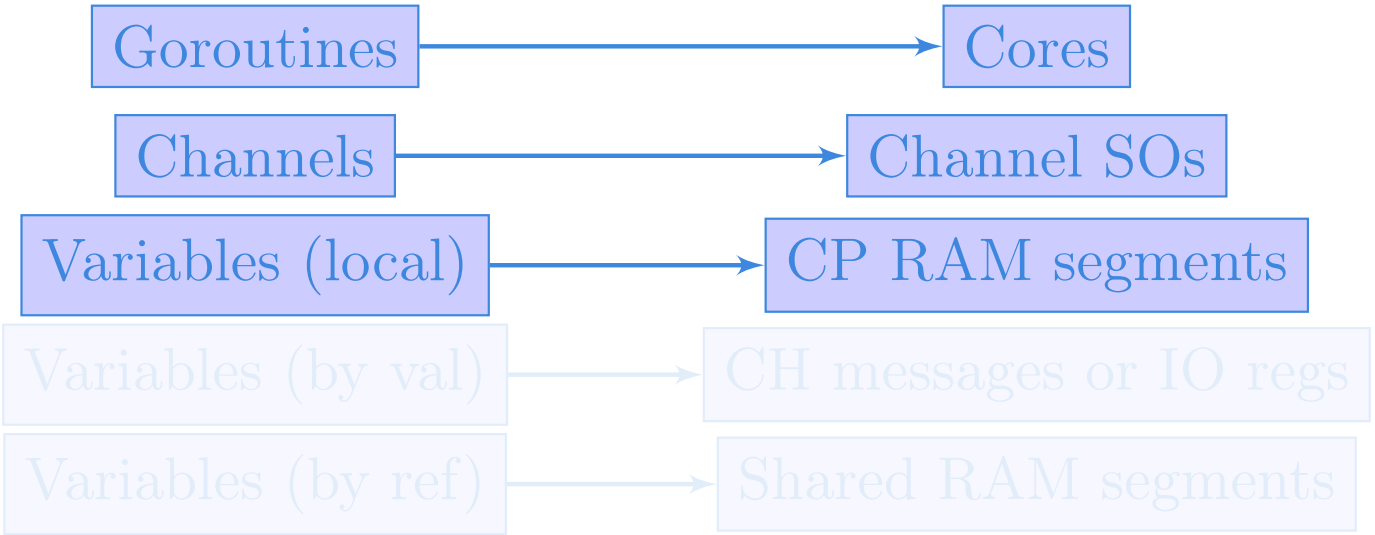


Bondgo

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.

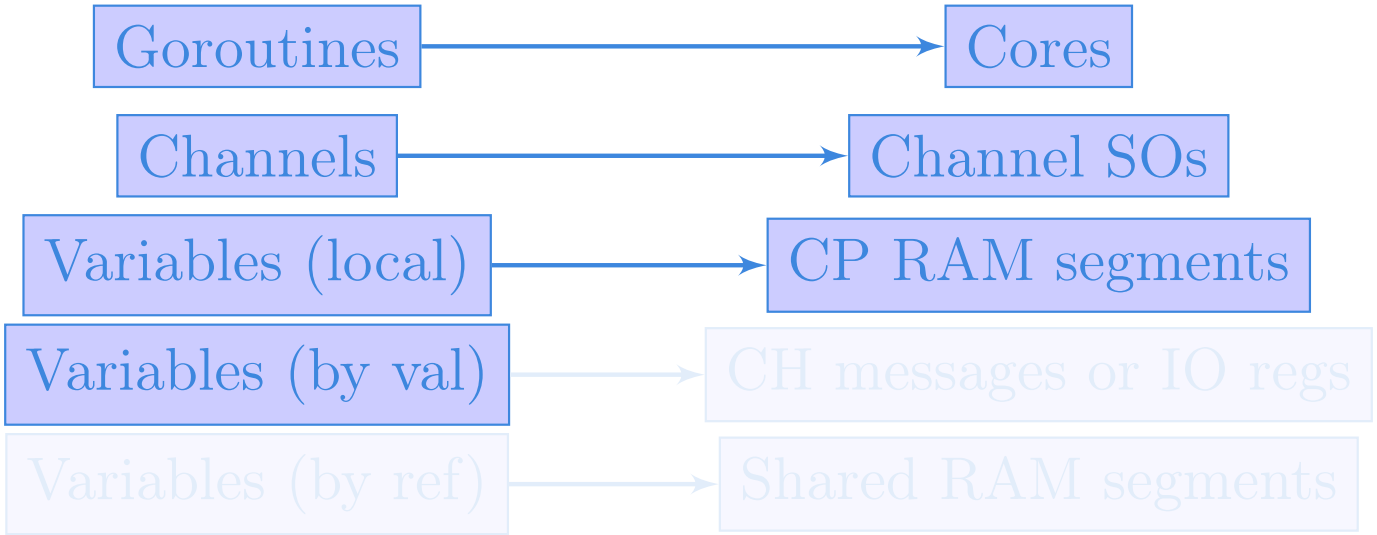


Bondgo

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.

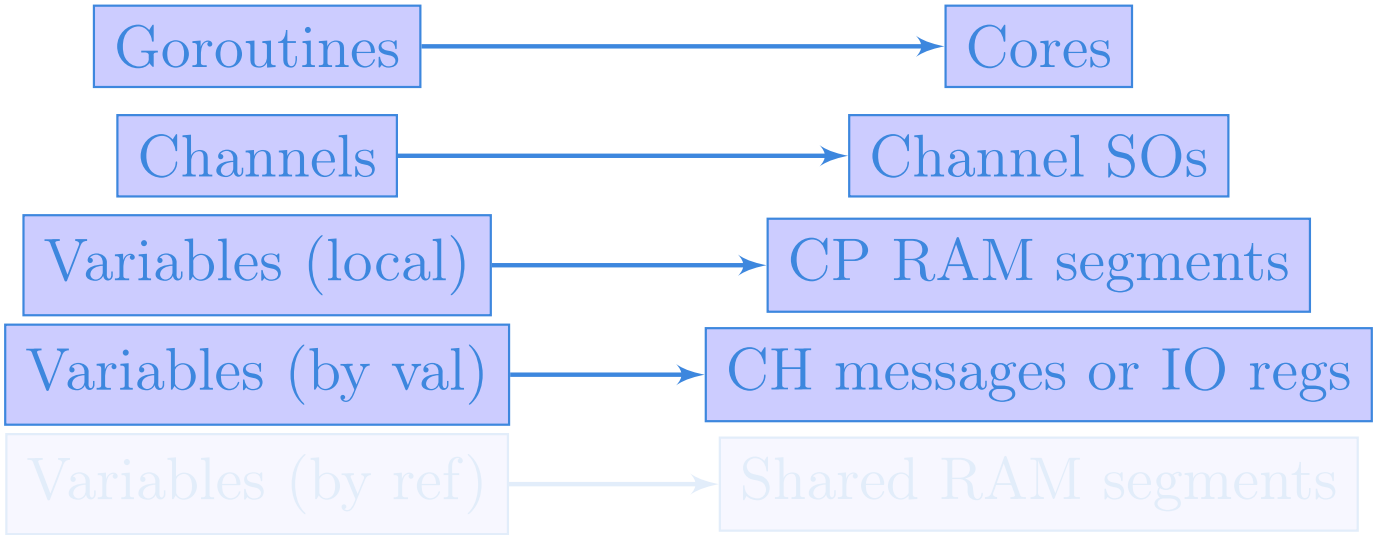


Bondgo

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.

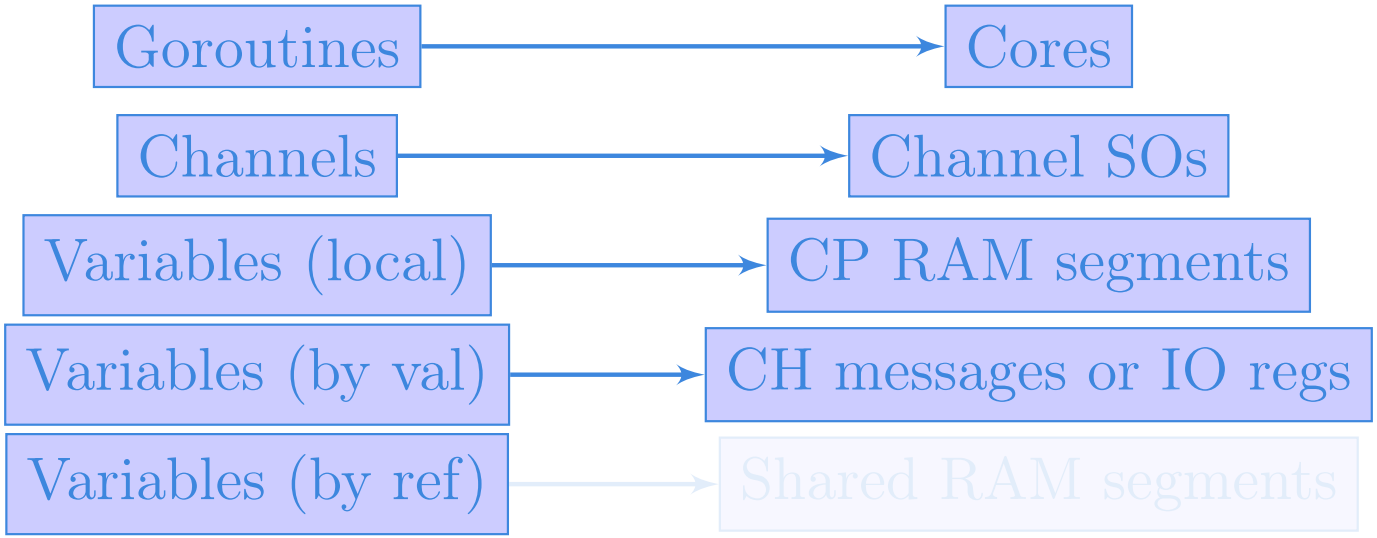


Bondgo

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.

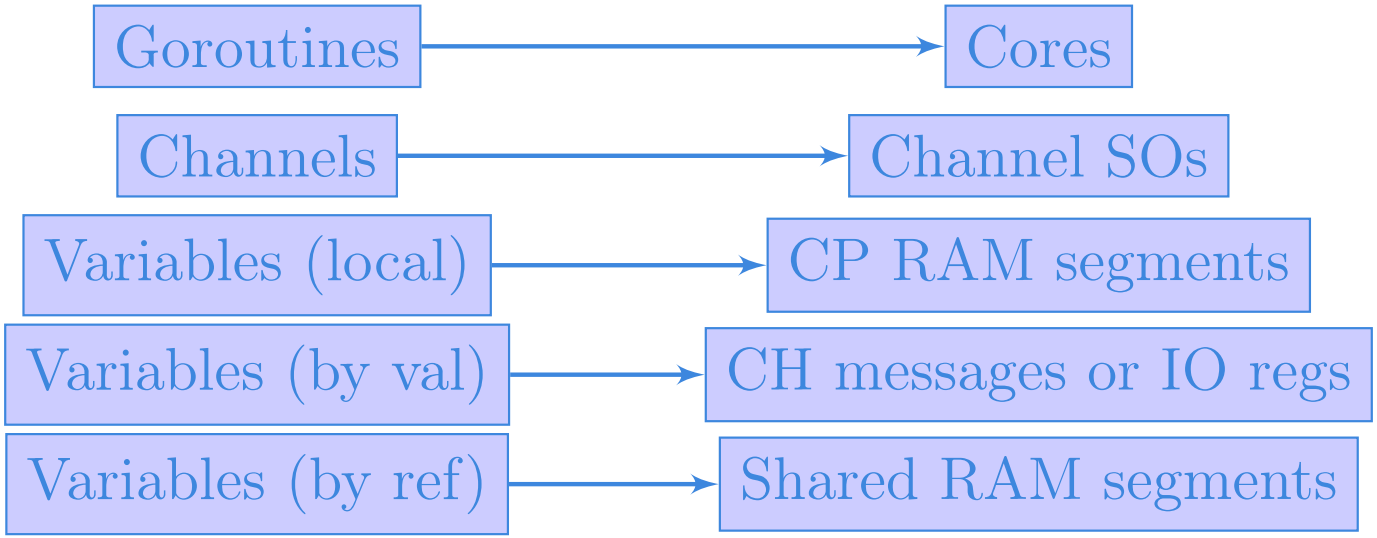


Bondgo

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

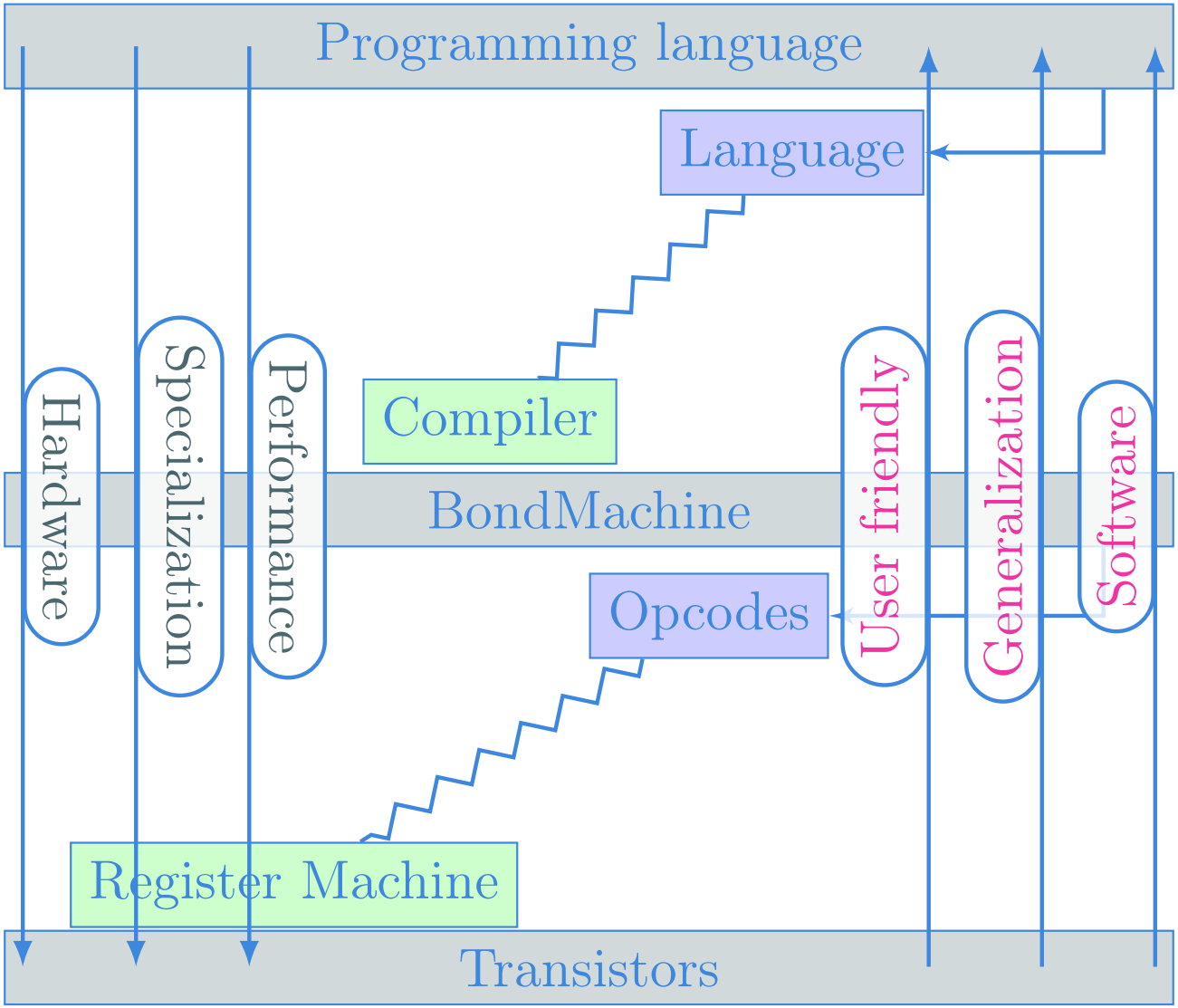
The idea was:
Build a computing system with a decreased number of layers resulting in a lower HW/SW gap.

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

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.

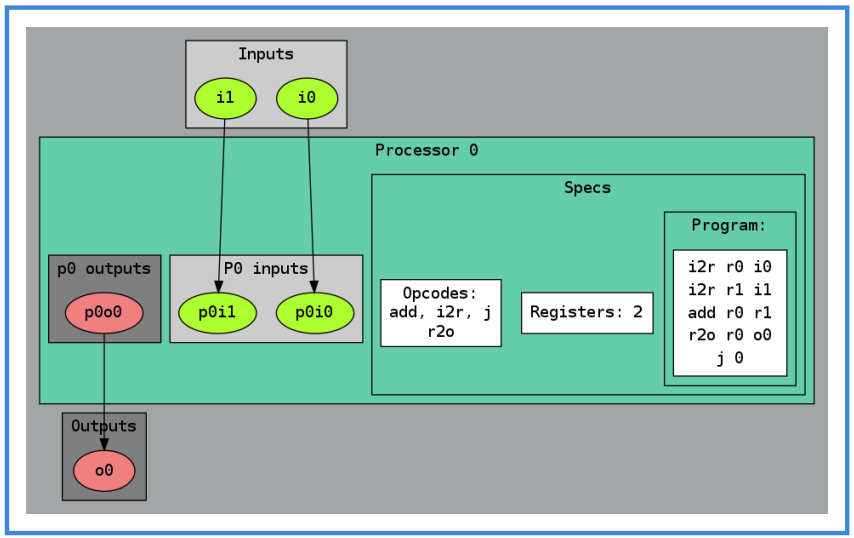
Bondgo

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

Builders API

With these Building Blocks

Several libraries have been 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 been 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 been 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 been 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 been 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))

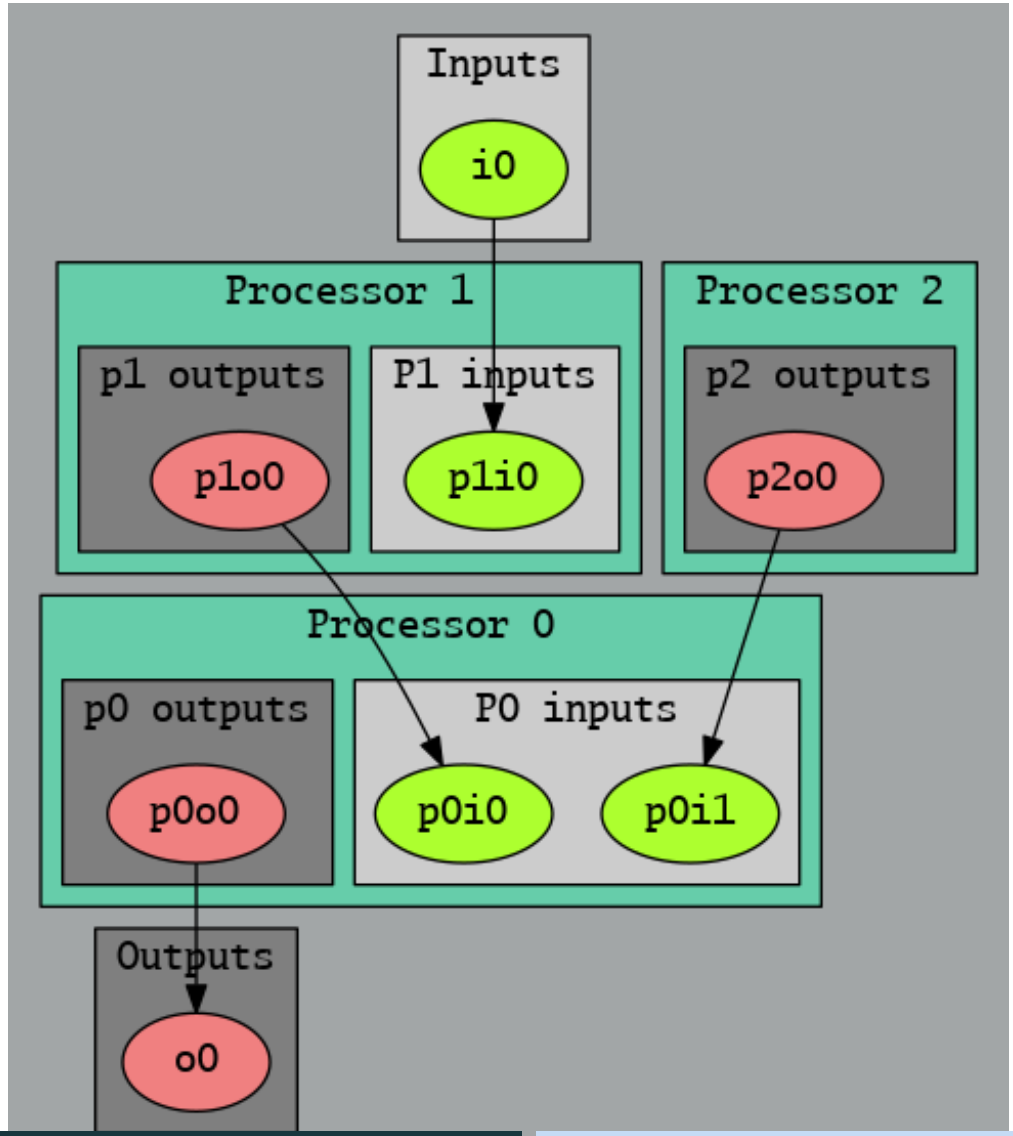
Boolbond

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)
```

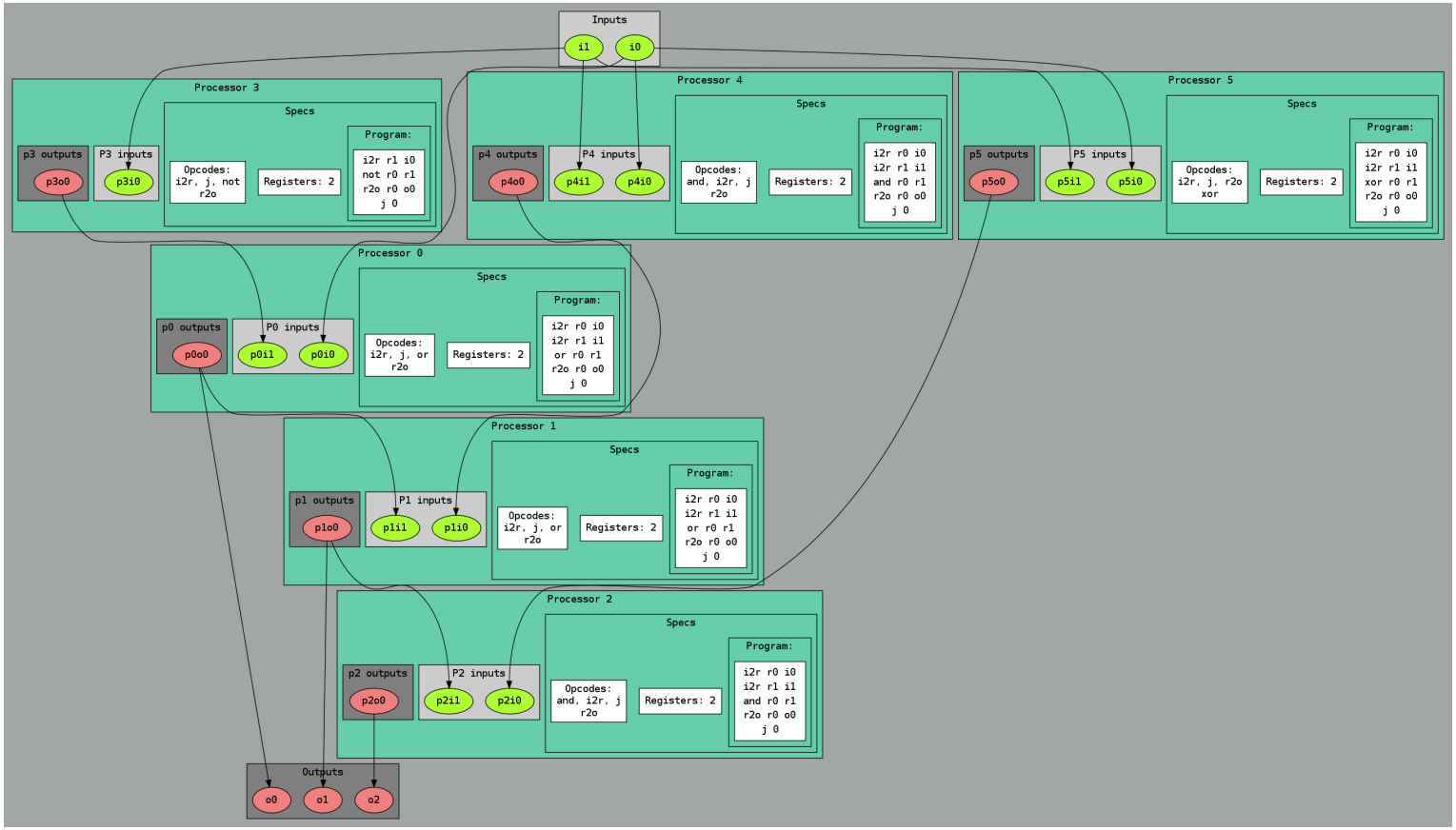
Boolbond

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

Resulting in:

Builders API

Boolbond



Boolbond demo

Demo N.9

It will be shown how:

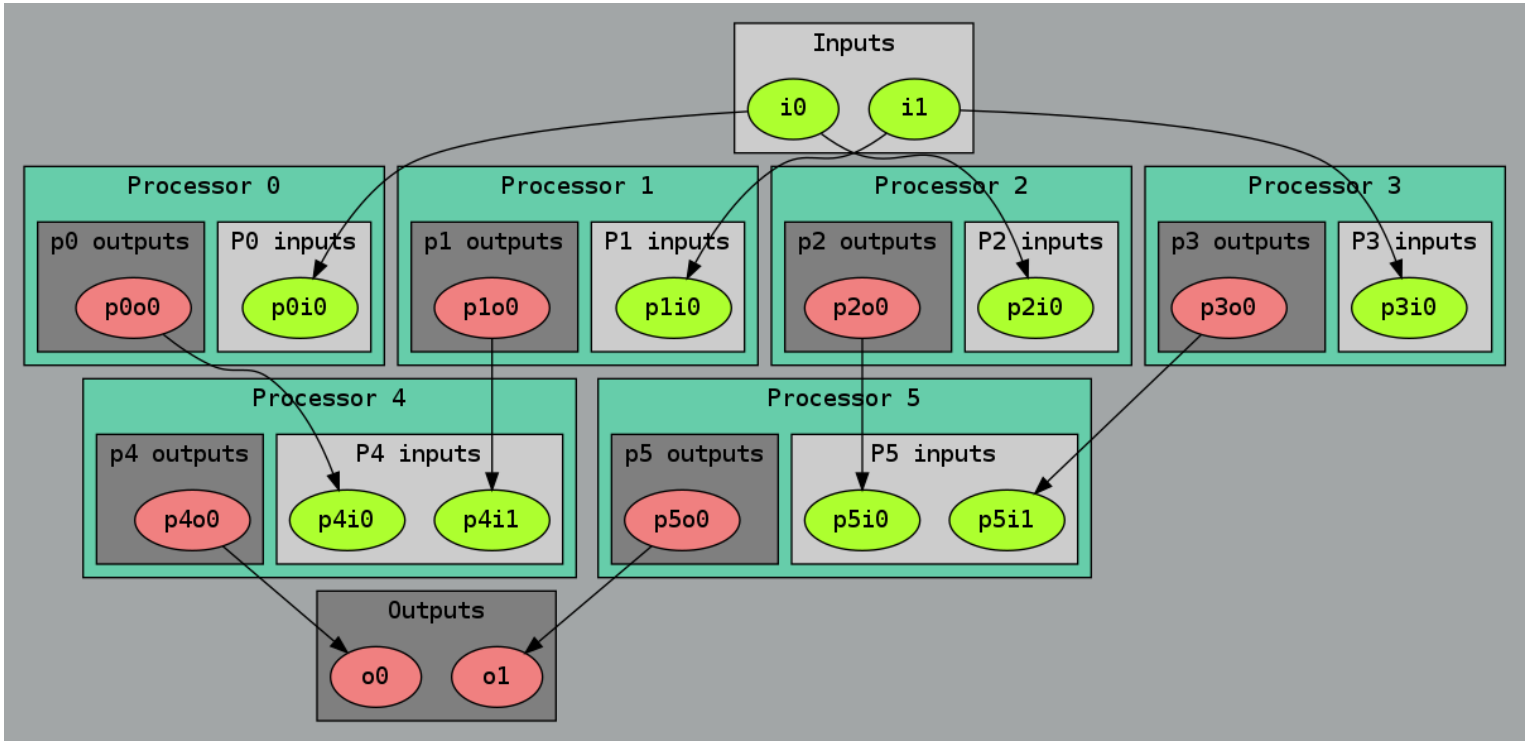
- To create complex multi-cores from boolean expressions

Builders API

Matrixwork

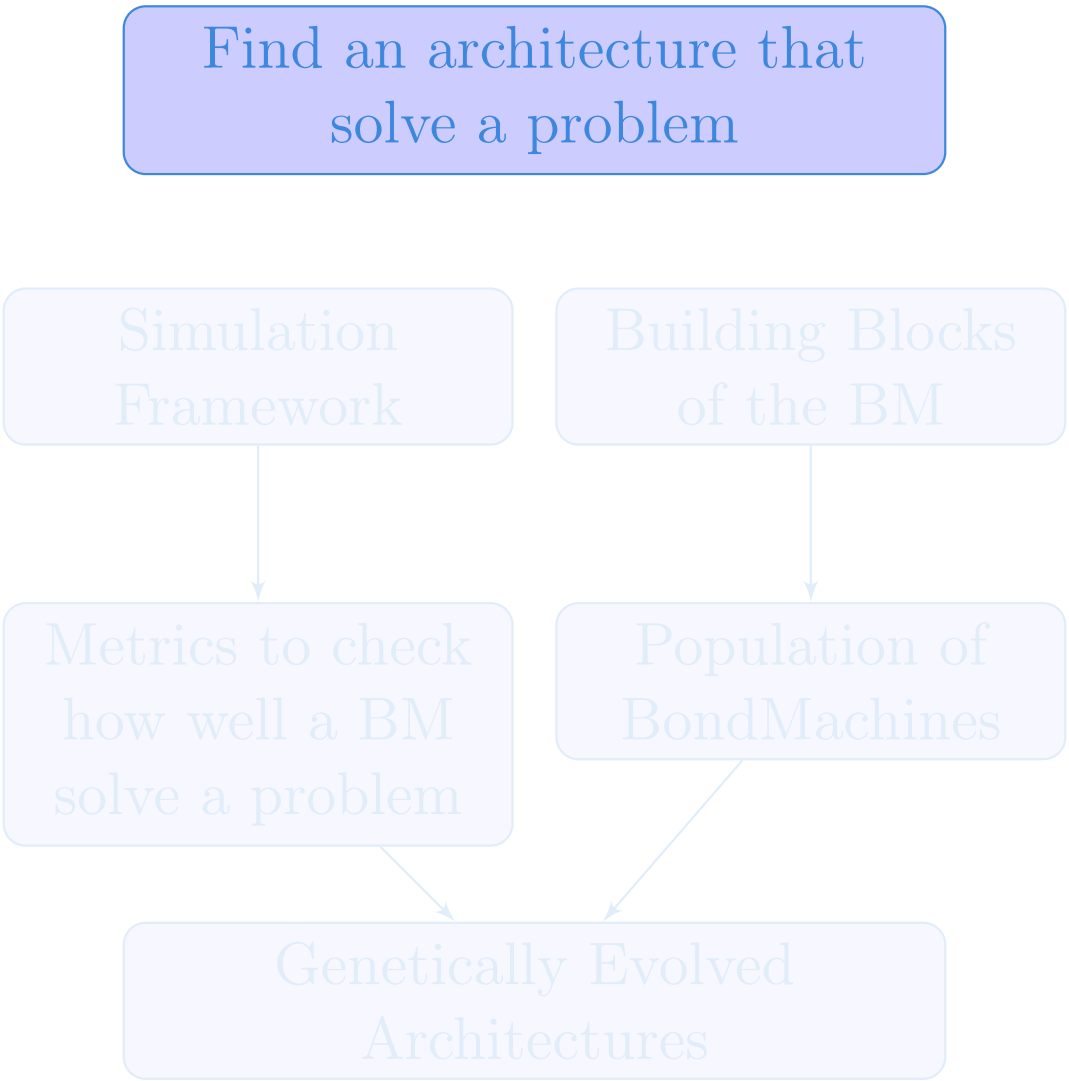
Matrix multiplication

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

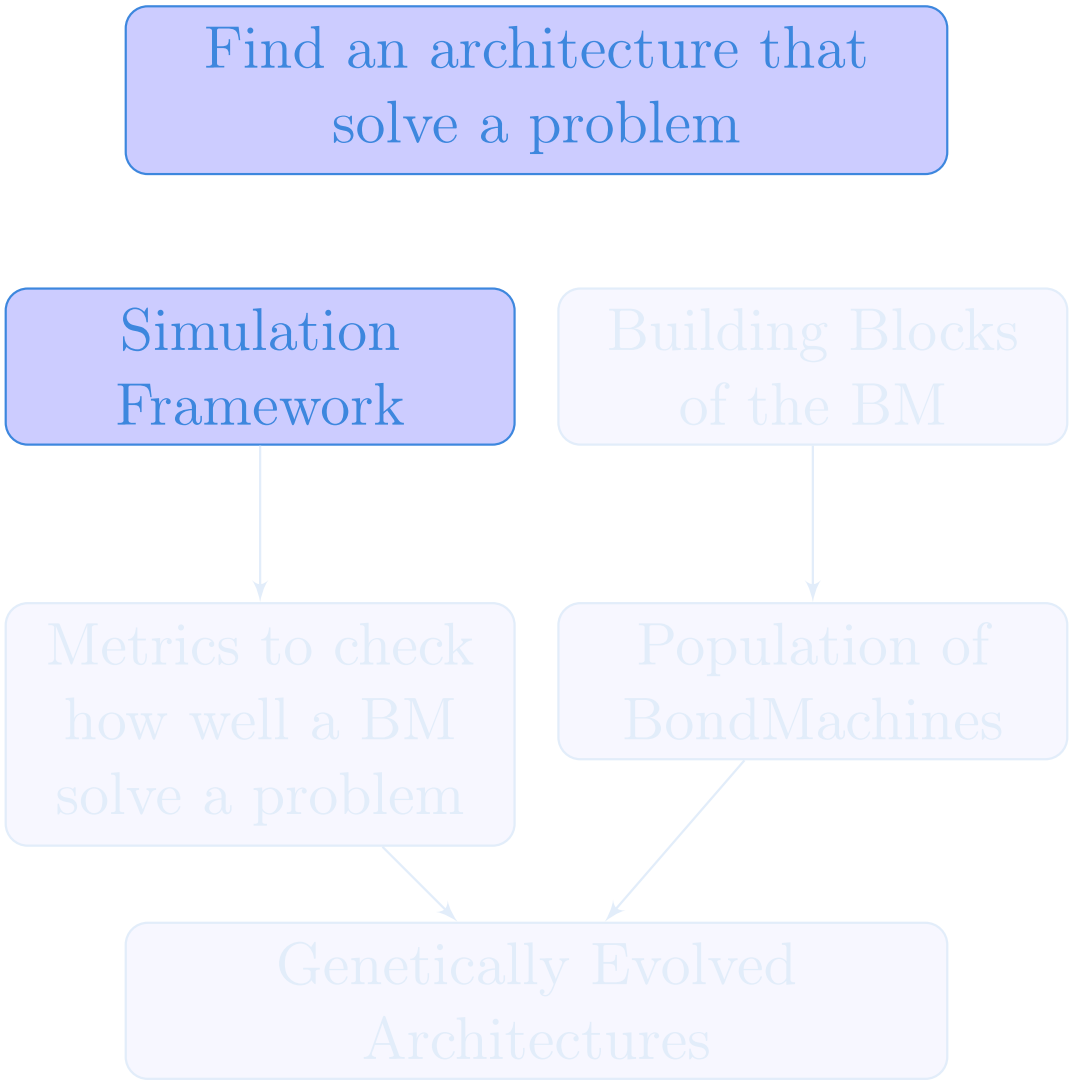


Evolutionary BondMachine

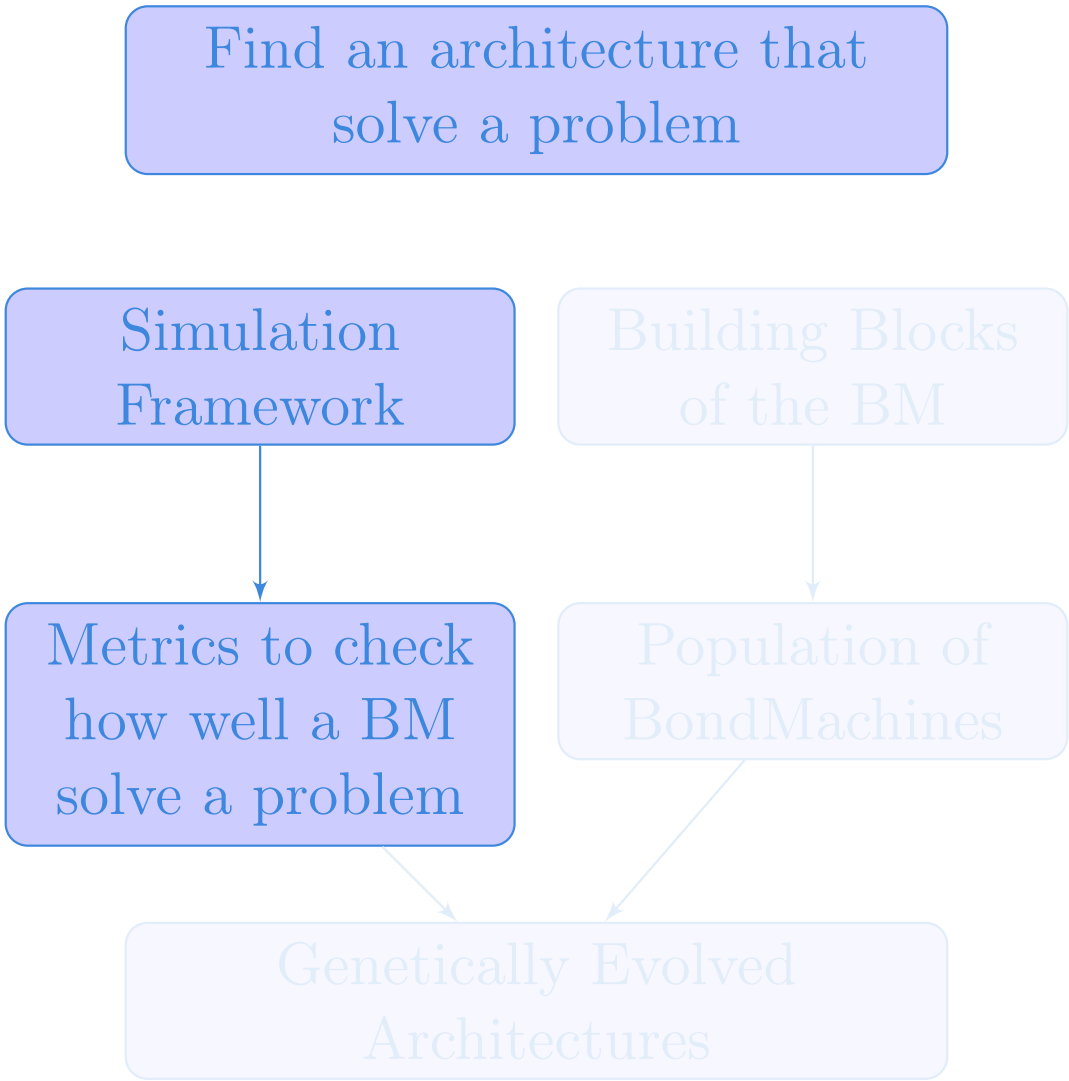
Evolutionary BondMachine



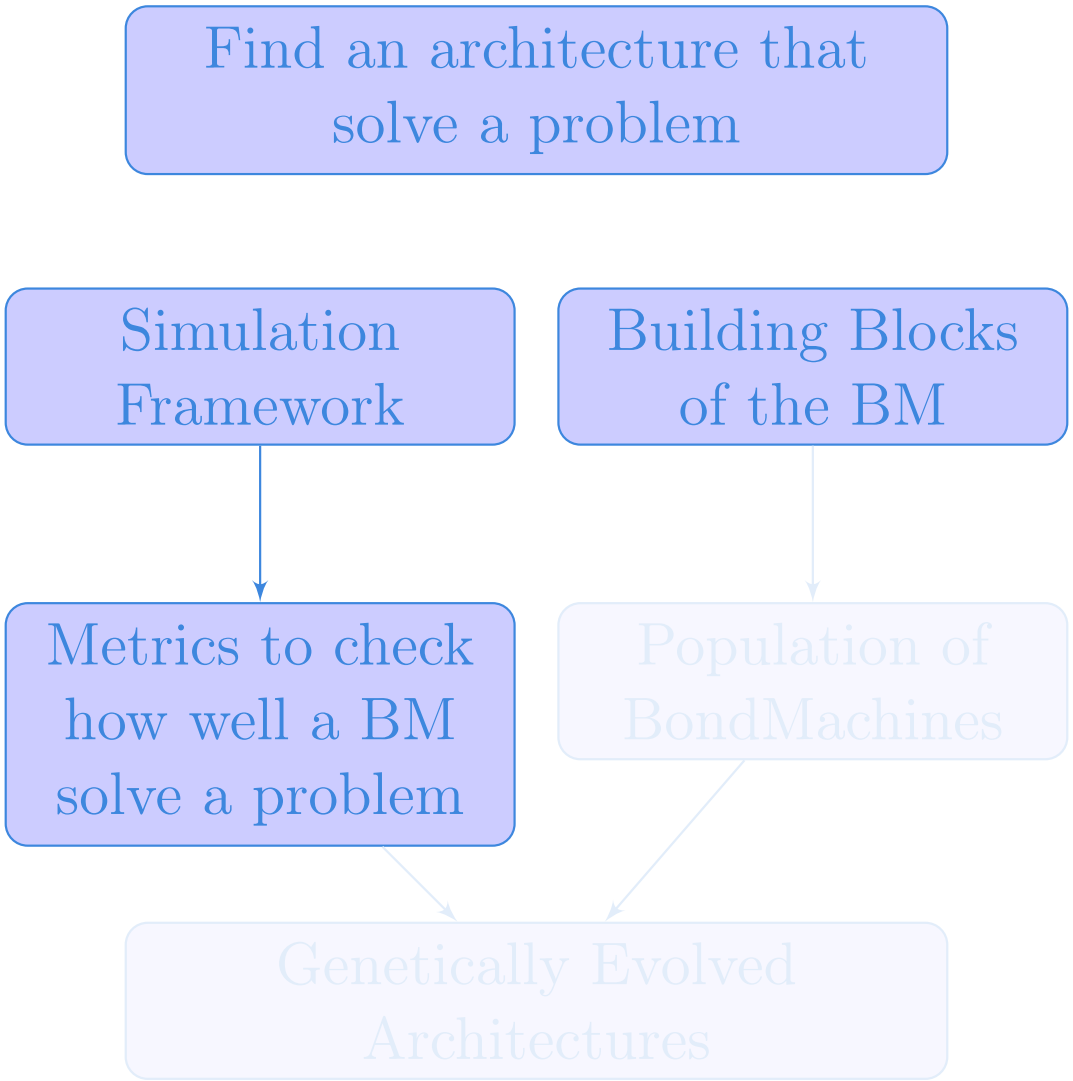
Evolutionary BondMachine



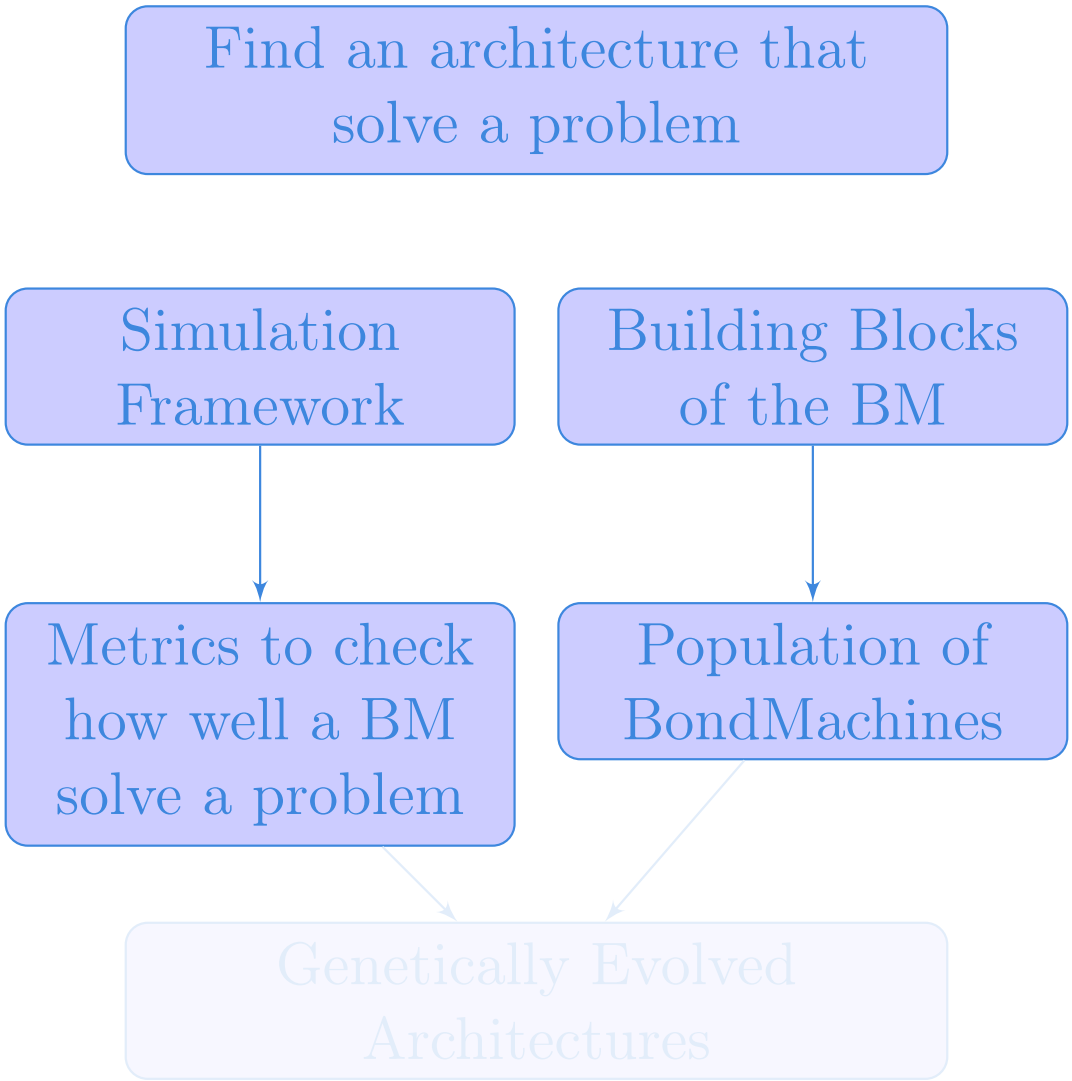
Evolutionary BondMachine



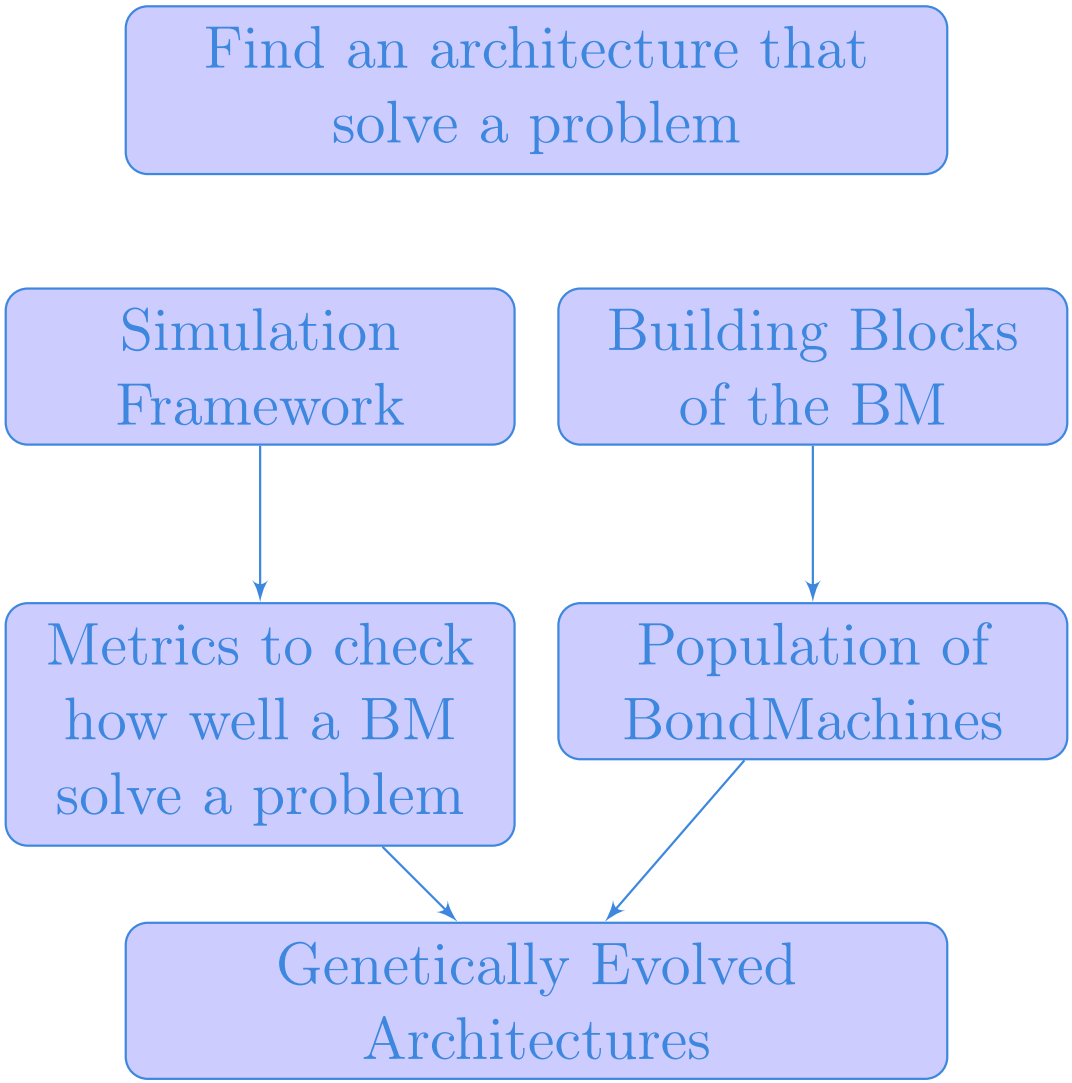
Evolutionary BondMachine



Evolutionary BondMachine



Evolutionary BondMachine



Machine Learning

Machine Learning with BondMachine

Architectures with multiple interconnected processors like the ones produced by the BondMachine Toolkit are a perfect fit for Neural Networks and Computational Graphs.

Several ways to map this structures to BondMachine has been developed:

- A native Neural Network library
- A Tensorflow to BondMachine translator
- An NNEF based BondMachine composer

Machine Learning with BondMachine

Architectures with multiple interconnected processors like the ones produced by the BondMachine Toolkit are a perfect fit for Neural Networks and Computational Graphs.

Several ways to map this structures to BondMachine has been developed:

- A native Neural Network library
- A Tensorflow to BondMachine translator
- An NNEF based BondMachine composer

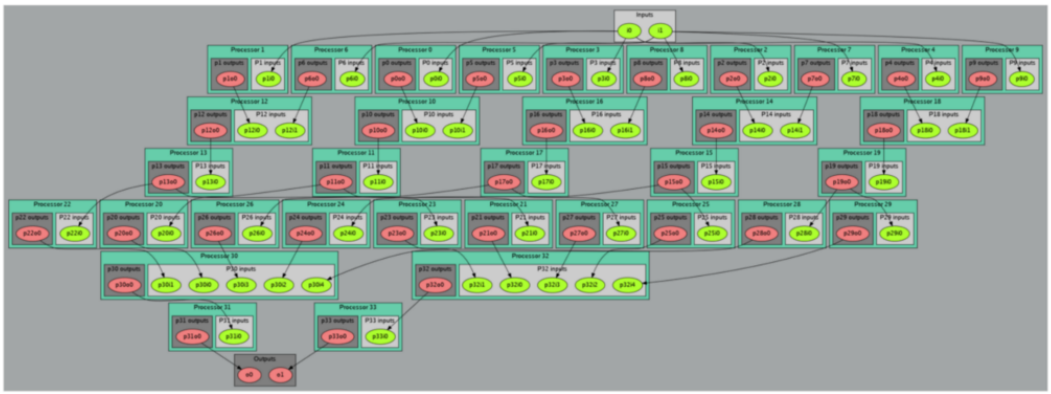
Machine Learning with BondMachine

Native Neural Network library

The tool neuralbond allow the creation of BM-based neural chips from an API go interface.

- Neurons are converted to BondMachine connecting processors.
- Tensors are mapped to CP connections.

```
layers := []int{2, 5, 2}
weights := make([]neuralbond.Weight, 0)

if *save_bondmachine != "" {
   if mymachine, ok :=
      neuralbond.Build_MLP(layers, weights); ok
      == nil {
      if _, err := os.Stat(*save_bondmachine);
      os.IsNotExist(err) {
         f, err := os.Create(*save_bondmachine)
         check(err)
         defer f.Close()
      }
   }
}
```

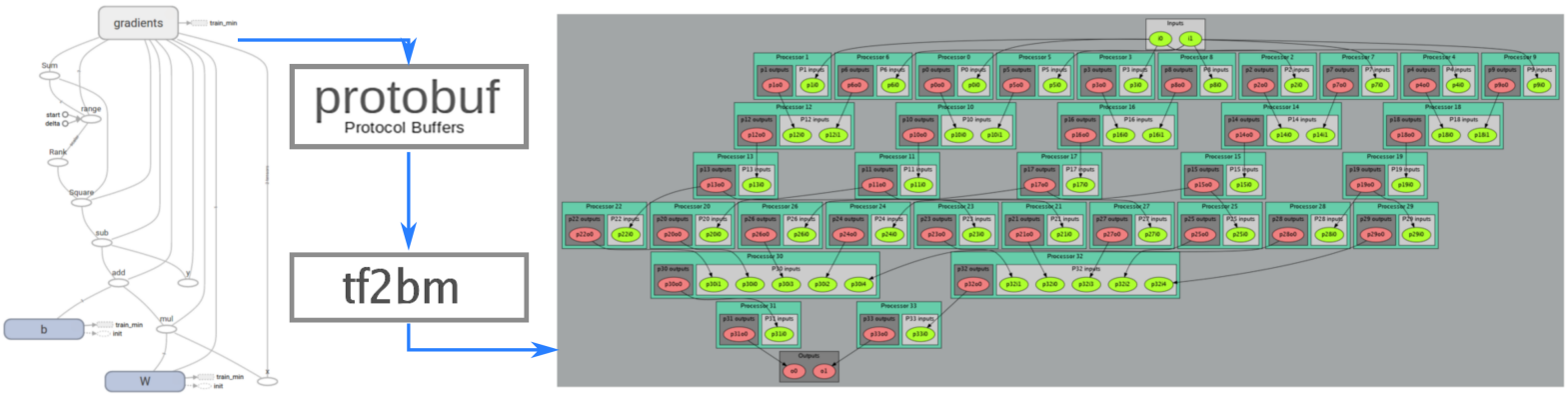


TensorFlow™ to Bondmachine

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.



Machine Learning with BondMachine

NNEF Composer

Neural Network Exchange Format (NNEF) is a standard from Khronos Group to enable the easy transfer of trained networks among frameworks, inference engines and devices

The NNEF BM tool approach is to descent NNEF models and build BondMachine multi-core accordingly

This approch has several advandages over the previous:

- It is not limited to a single framework
- NNEF is a textual file, so no complex operations are needed to read models

Hardware

Hardware implementation

FPGA

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

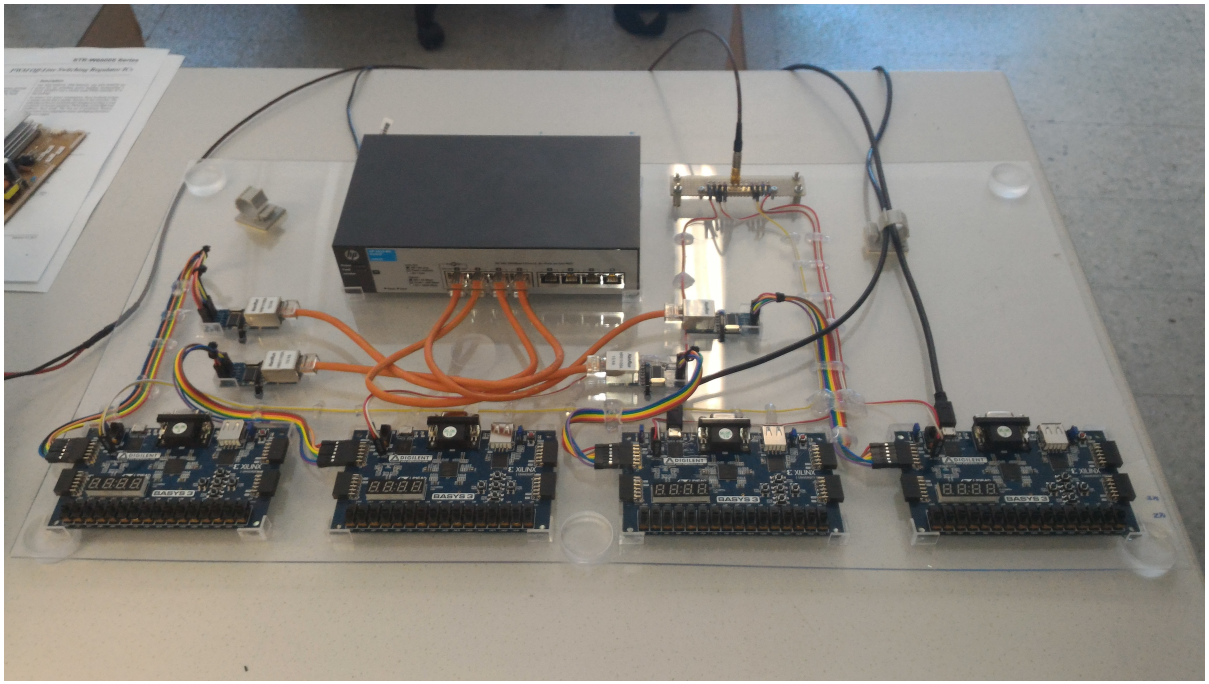
- Digilent Basys3 - Xilinx Artix-7 - Vivado.
- Kintex7 Evaluation Board - Vivado.
- Digilent Zedboard - Xilinx Zynq 7020 - Vivado.
- Linux - Iverilog.
- Terasic De10nano - Intel Cyclone V - Quartus

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.

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 Youtube video

Clustering

BondMachine Clustering

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) over an heterogeneous layer.

Interconnected BondMachines

What if we could extend the this layer to multiple interconnected devices ?

BondMachine Clustering

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) over an heterogeneous layer.

Interconnected BondMachines

What if we could extend the this layer to multiple interconnected devices ?

BondMachine Clustering

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) over an heterogeneous layer.

Interconnected BondMachines

What if we could extend the this layer to multiple interconnected devices ?

BondMachine Clustering

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) over an heterogeneous layer.

Interconnected BondMachines

What if we could extend the this layer to multiple interconnected devices ?

BondMachine Clustering

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 might join a cluster an contribute to a single distributed computational problem.

BondMachine Clustering

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 might join a cluster an contribute to a single distributed computational problem.

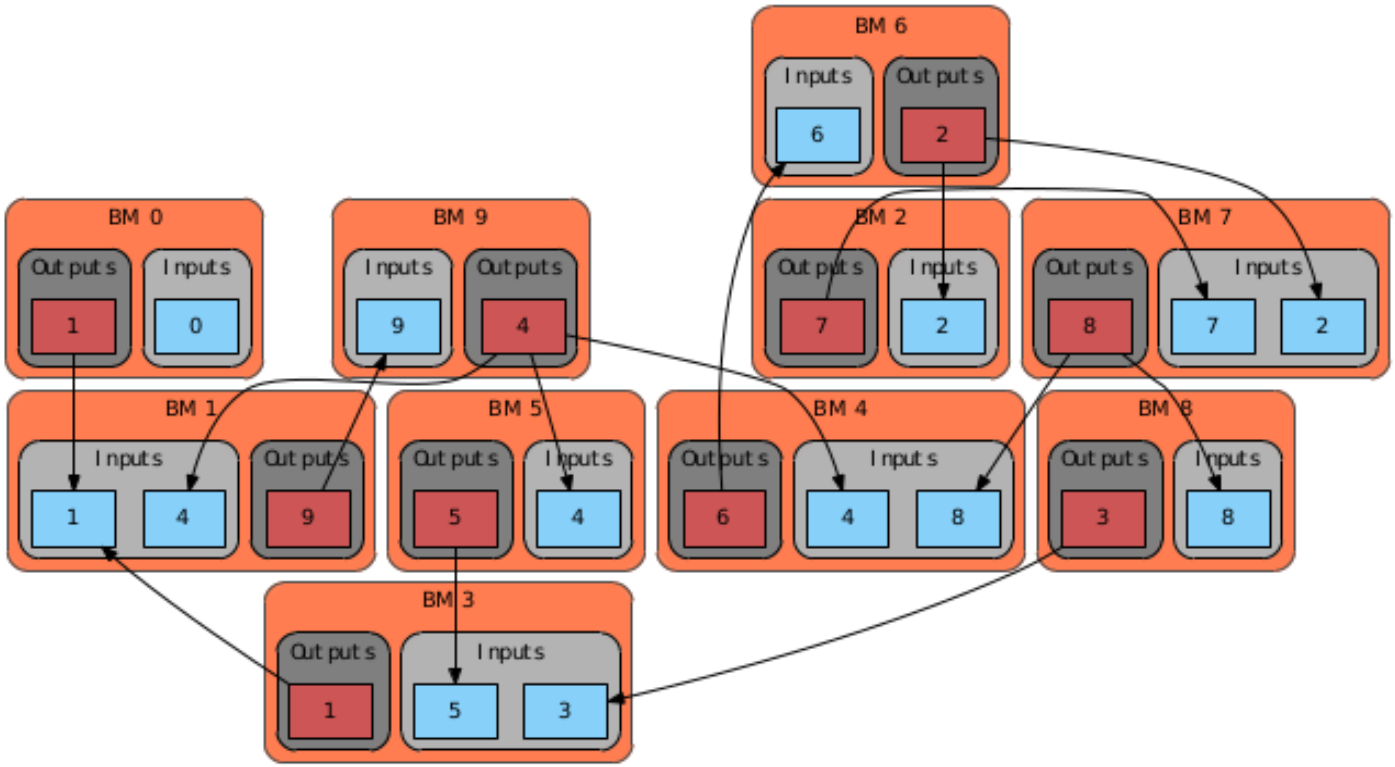
BondMachine Clustering

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 might join a cluster an contribute to a single distributed computational problem.

BondMachine Clustering



BondMachine Clustering

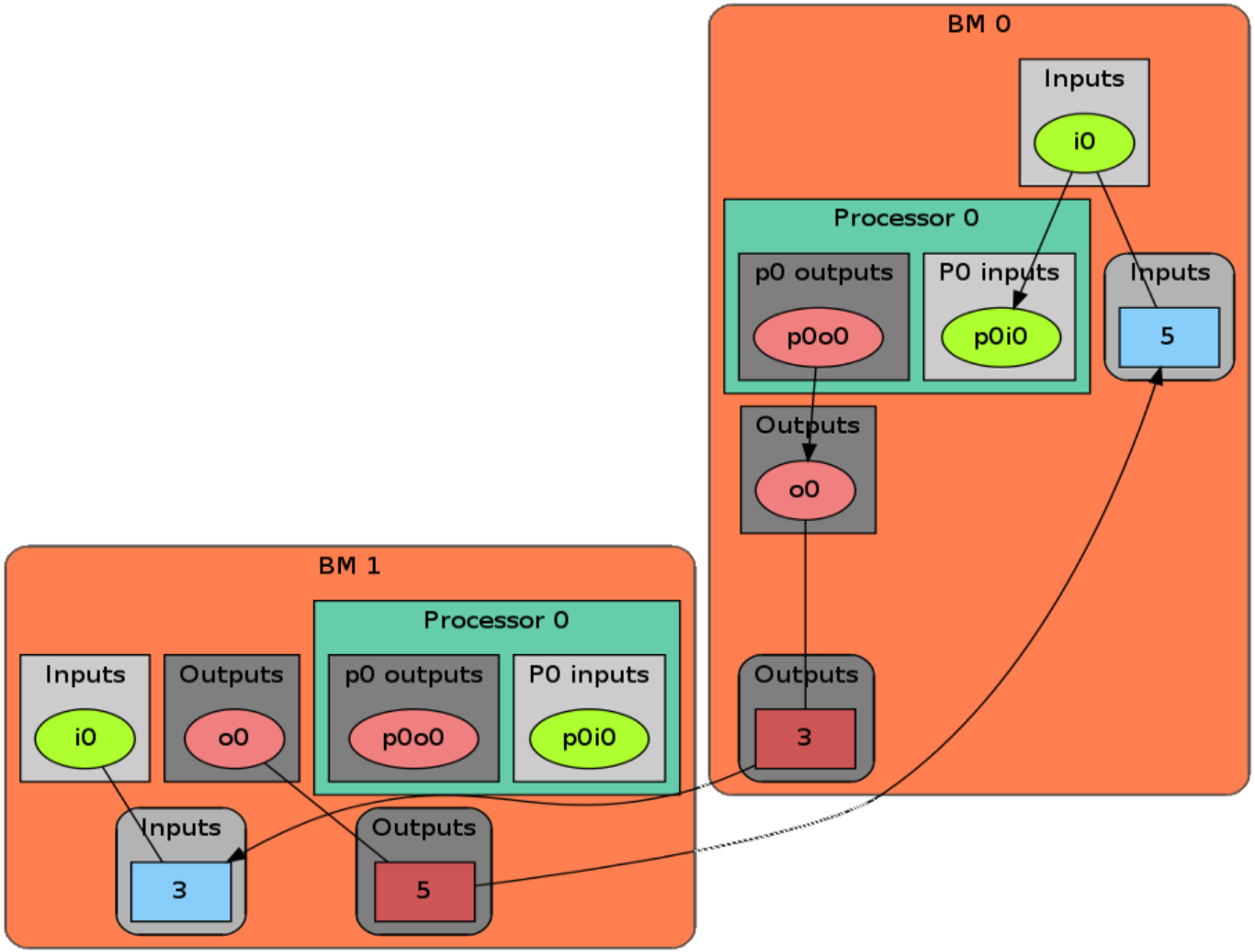
A distributed example

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)
   }
}

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))
   }
}
```

BondMachine Clustering

A distributed example



BondMachine Clustering

A distributed example

The result is:
BondMachine Clustering Youtube video

A general result

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

BondMachine Clustering

Results

Results

- User can deploy an entire HW/SW cluster starting from code written in a high level description (Go, NNEF, etc)
- Workstation with emulated BondMachines, workstation with etherbond drivers, standalone BondMachines (FPGA) may join these clusters.

BondMachine Clustering

Results

Results

- User can deploy an entire HW/SW cluster starting from code written in a high level description (Go, NNEF, etc)
- Workstation with emulated BondMachines, workstation with etherbond drivers, standalone BondMachines (FPGA) may join these clusters.

Use cases

Use cases

Two use cases in Physics experiments are currently being developed:

- Real time pulse shape analysis in neutron detectors
  - bringing the intelligence to the edge
- Test beam for space experiments (DAMPE, HERD)
  - increasing testbed operations efficiency

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:

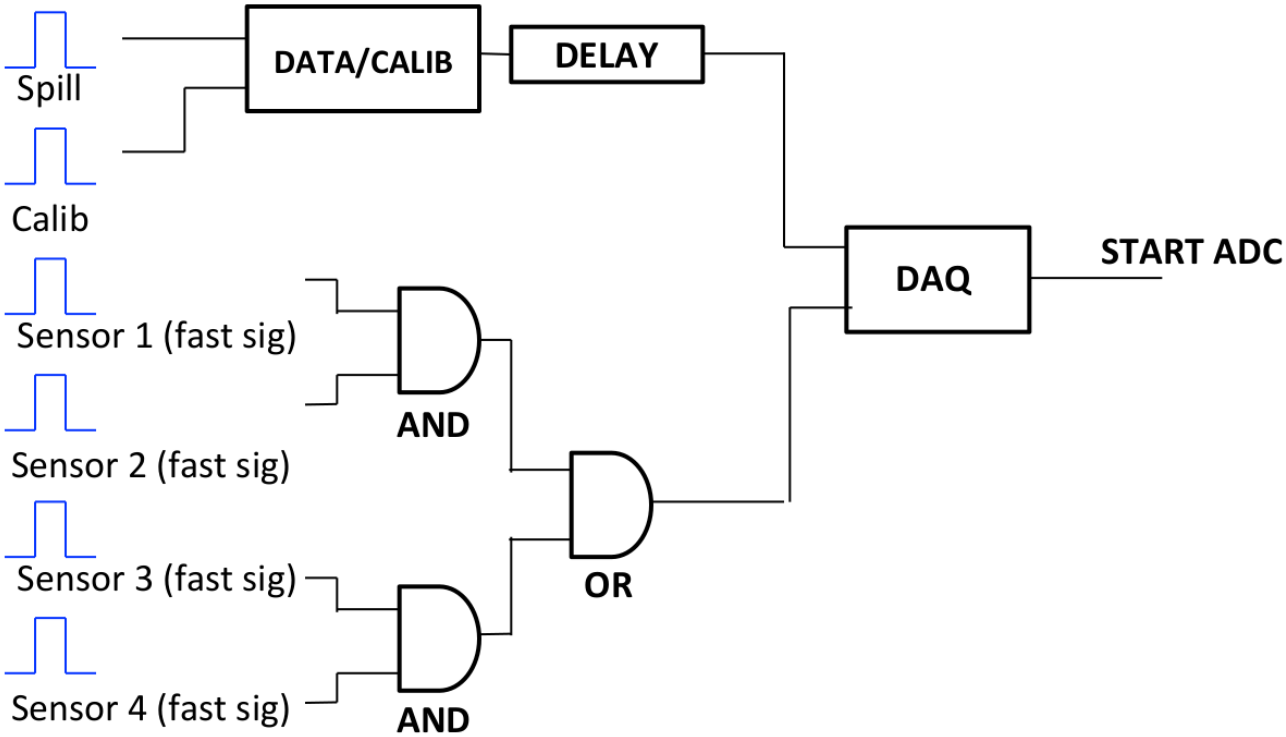
- Clock, pulser
- Logic gates (AND, OR,…)
- Delays

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:

- IoT and CyberPhysical systems.
- Computer Science educational applications.

Computing Accelerator

Our effort is now in enabling the possibility of building computing accelerators to be used from within standard (Linux) applications.

Possible other uses

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

- IoT and CyberPhysical systems.
- Computer Science educational applications.

Computing Accelerator

Our effort is now in enabling the possibility of building computing accelerators to be used from within standard (Linux) applications.

Possible other uses

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

- IoT and CyberPhysical systems.
- Computer Science educational applications.

Computing Accelerator

Our effort is now in enabling the possibility of building computing accelerators to be used from within standard (Linux) applications.

Possible other uses

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

- IoT and CyberPhysical systems.
- Computer Science educational applications.

Computing Accelerator

Our effort is now in enabling the possibility of building computing accelerators to be used from within standard (Linux) applications.

Accelerators

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)

Accelerators

The BondMachine can be used (emulated or on FPGA) as a single stand-alone computing device.

It can be used spawned on multiple devices (emulated, on FPGA or both)

It can be used as computing accelerator

BondMachine can be created and used from within standard (Linux) applications.

Accelerators

The BondMachine can be used (emulated or on FPGA) as a single stand-alone computing device.

It can be used spawned on multiple devices (emulated, on FPGA or both)

It can be used as computing accelerator

BondMachine can be created and used from within standard (Linux) applications.

Accelerators

The BondMachine can be used (emulated or on FPGA) as a single stand-alone computing device.

It can be used spawned on multiple devices (emulated, on FPGA or both)

It can be used as computing accelerator

BondMachine can be created and used from within standard (Linux) applications.

Accelerators

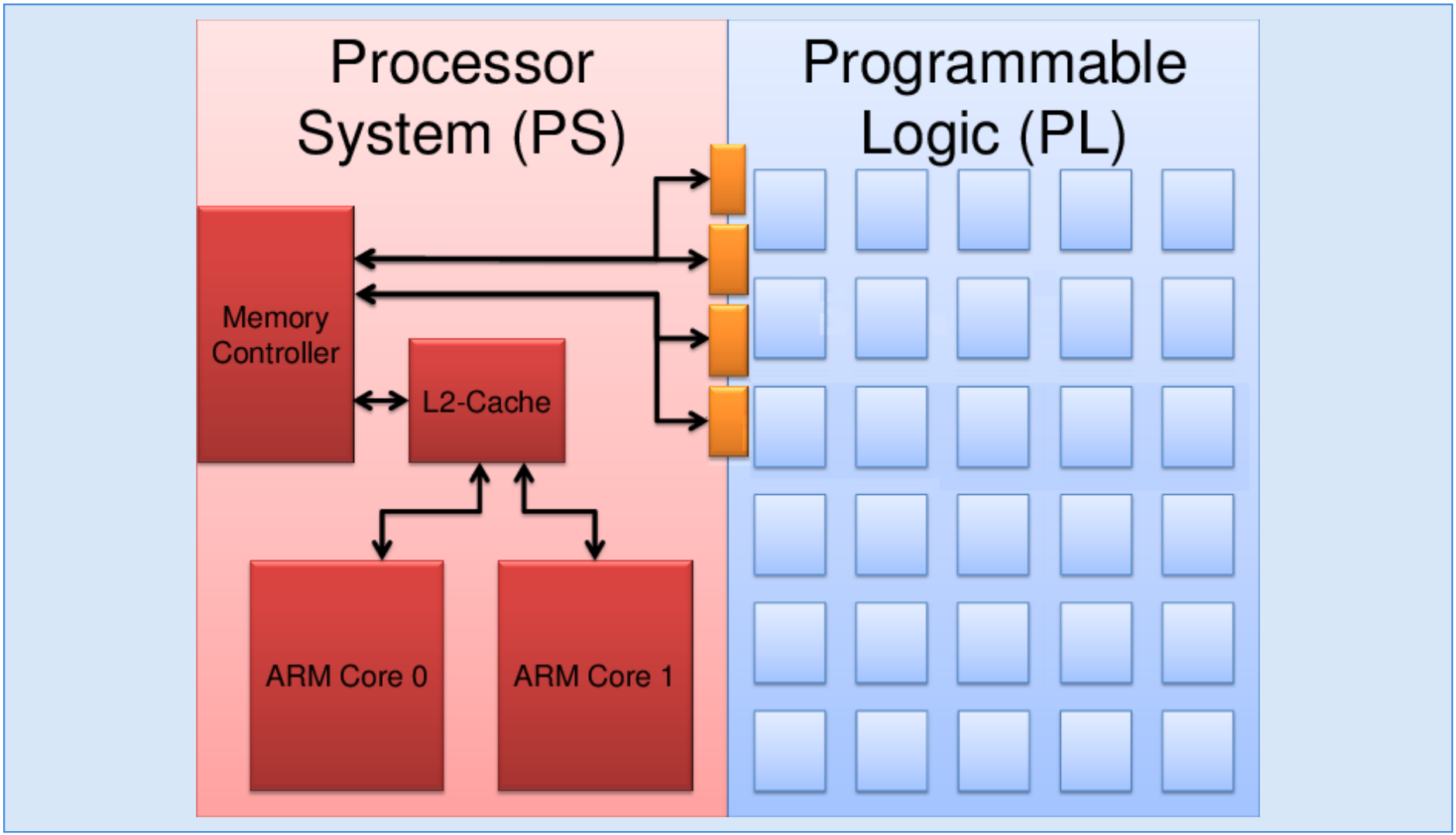
Types

We are currently working to enable the use the BM as accelerator in two directions:

- Using standard processor/FPGA hybrid chips
  - Zynq, Cyclone V
- Using PCI-express FPGA evaluation boards
  - Kintek 7 Evaluation board

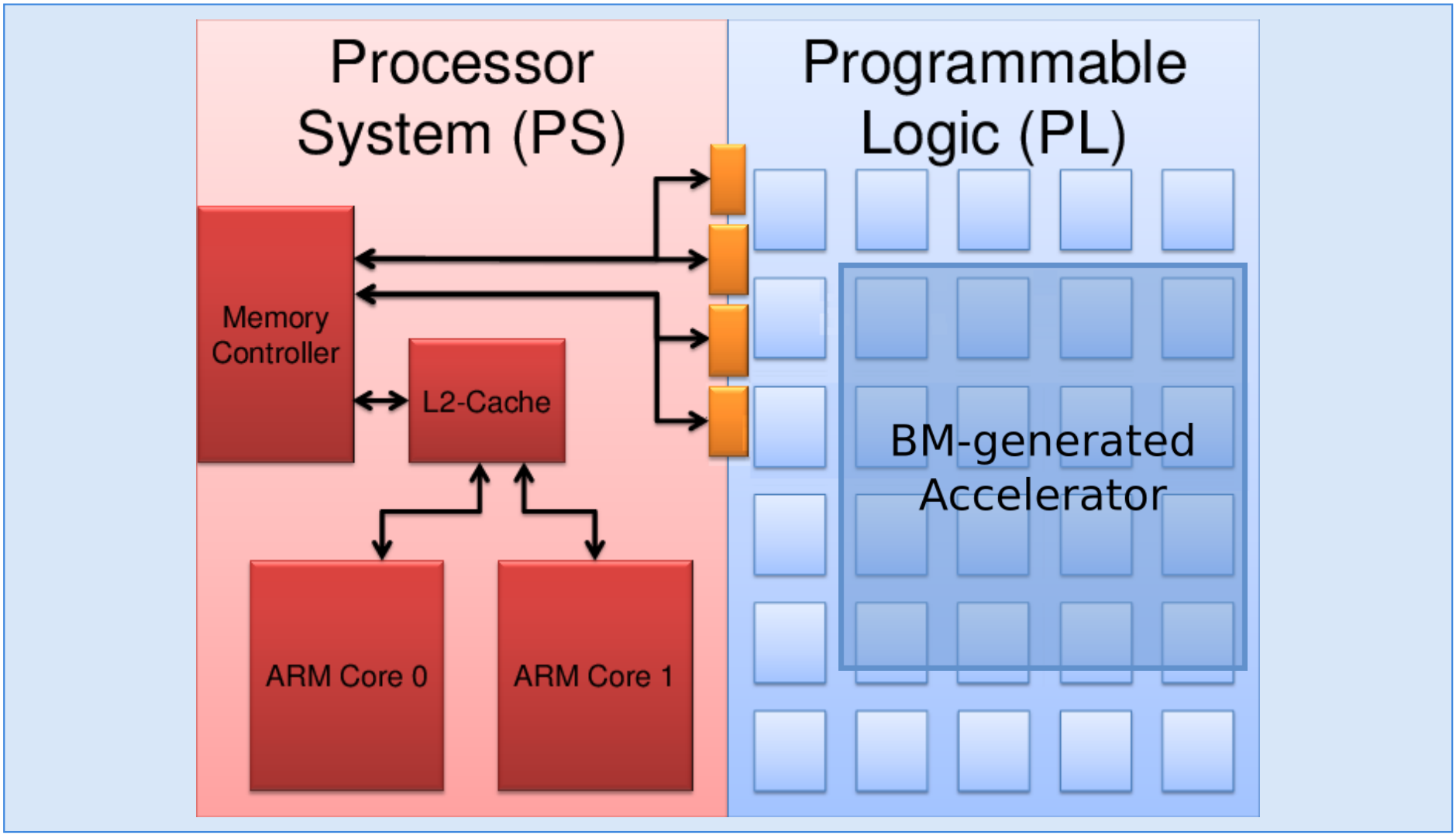
Accelerators

Hybrid chips



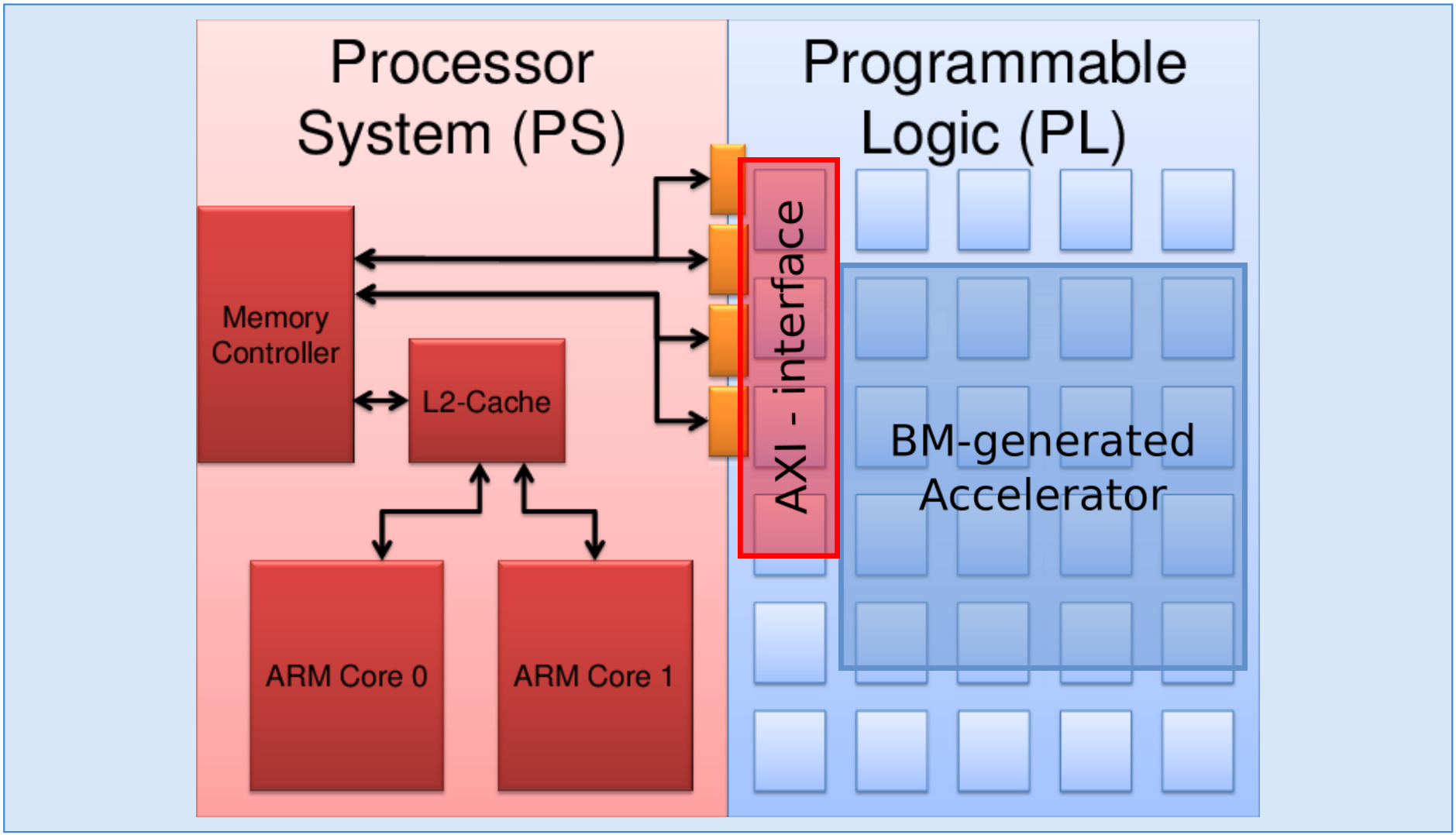
Accelerators

Hybrid chips



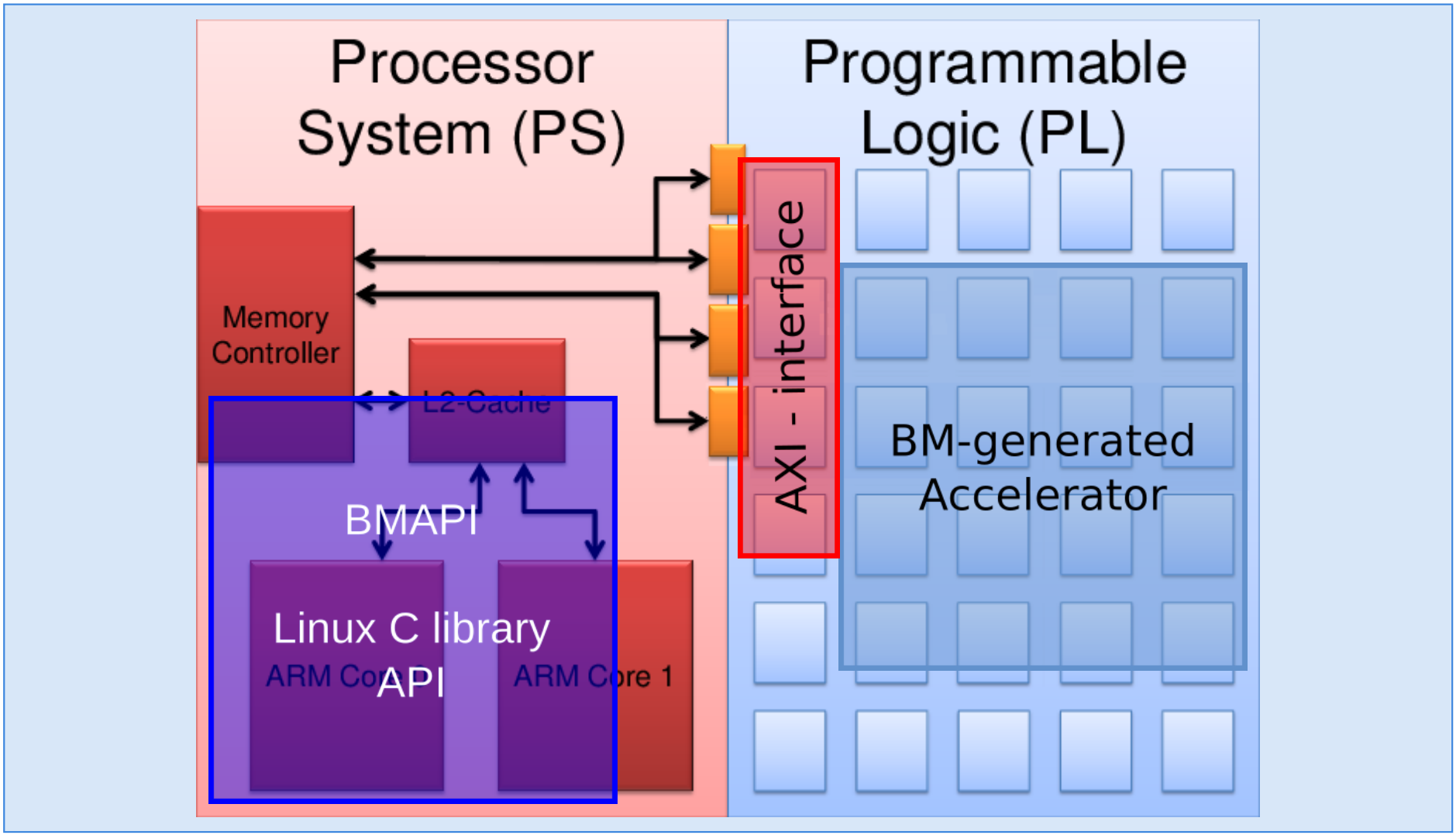
Accelerators

Hybrid chips



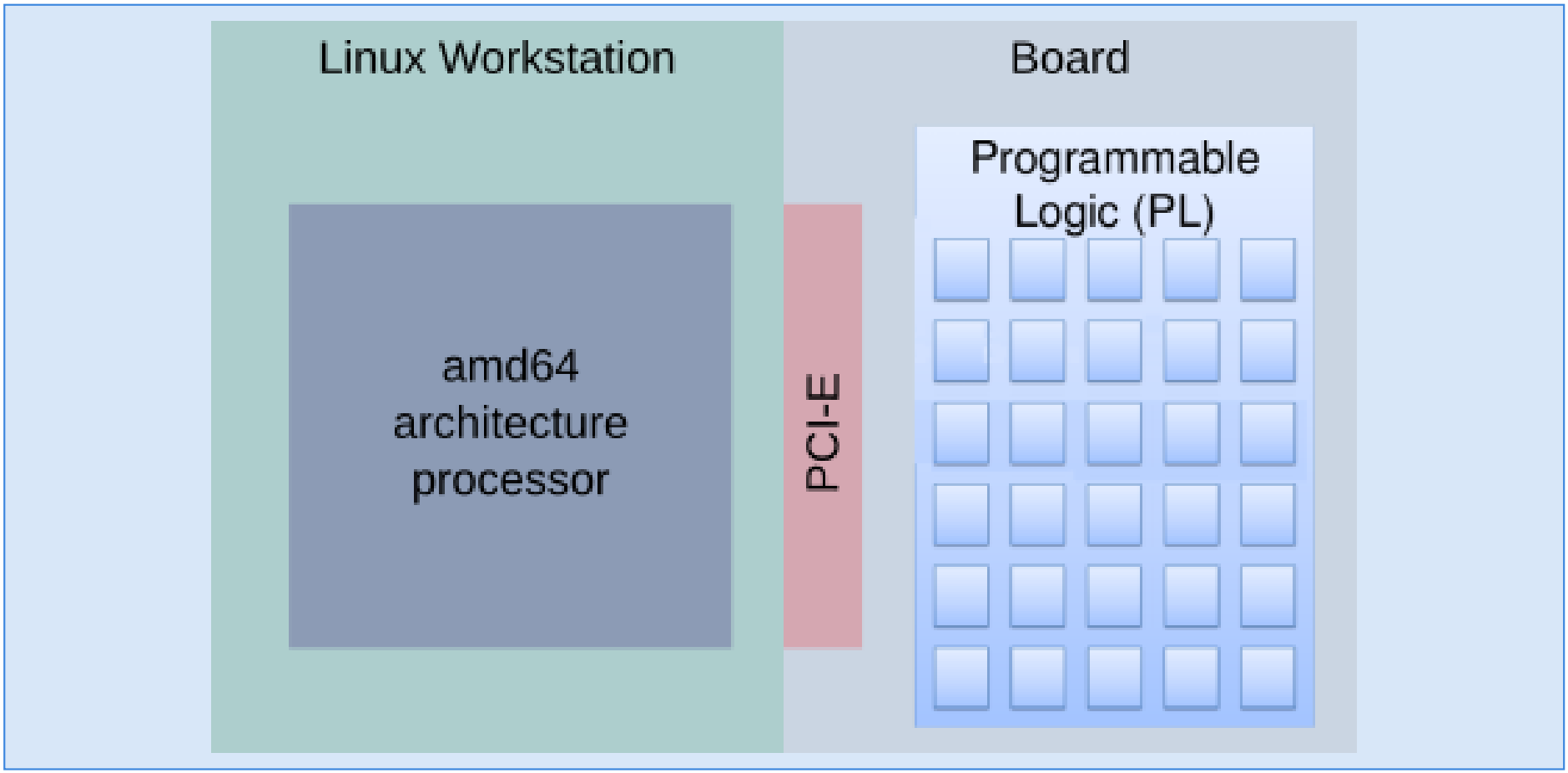
Accelerators

Hybrid chips



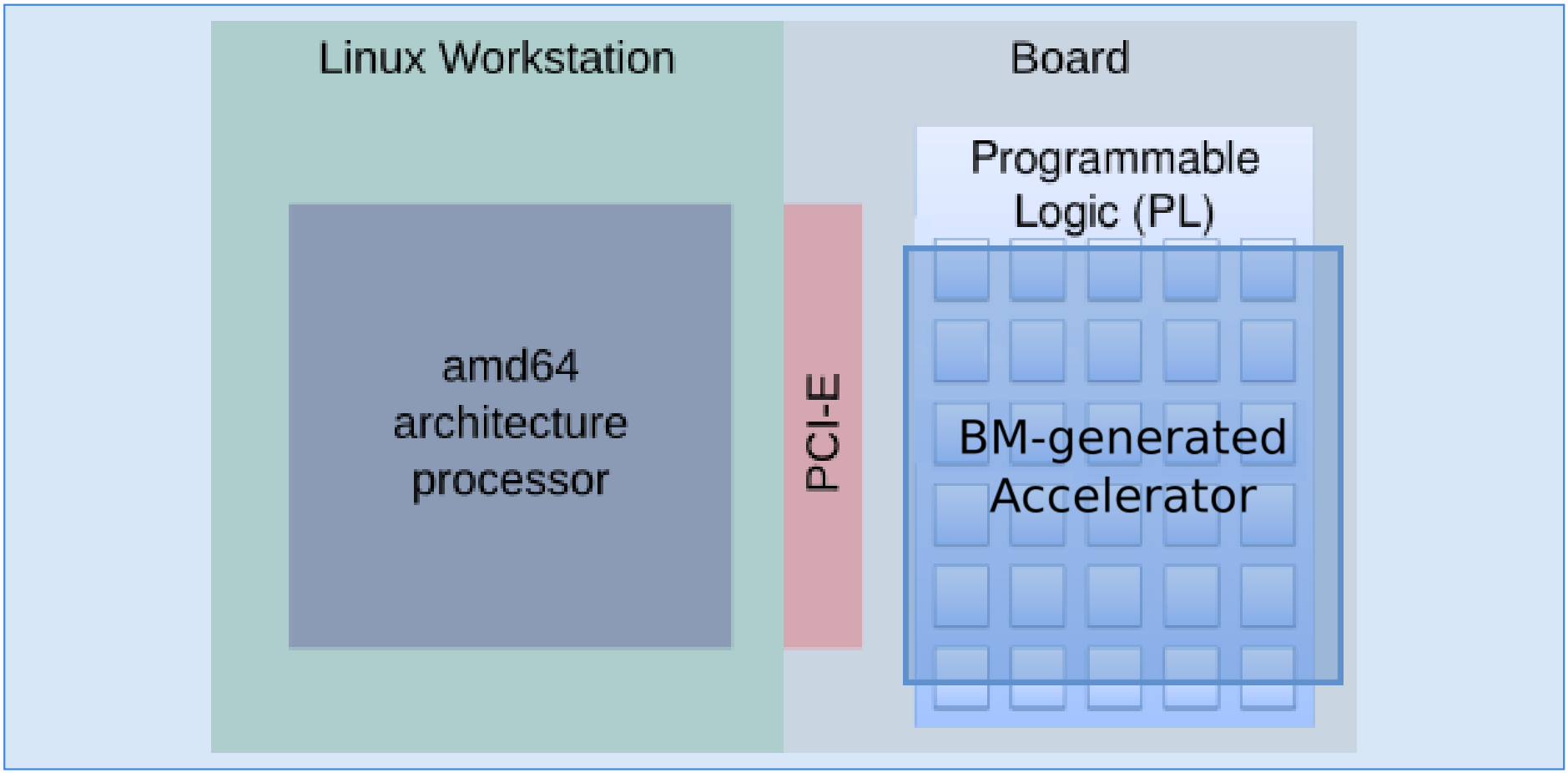
Accelerators

PCI-express boards



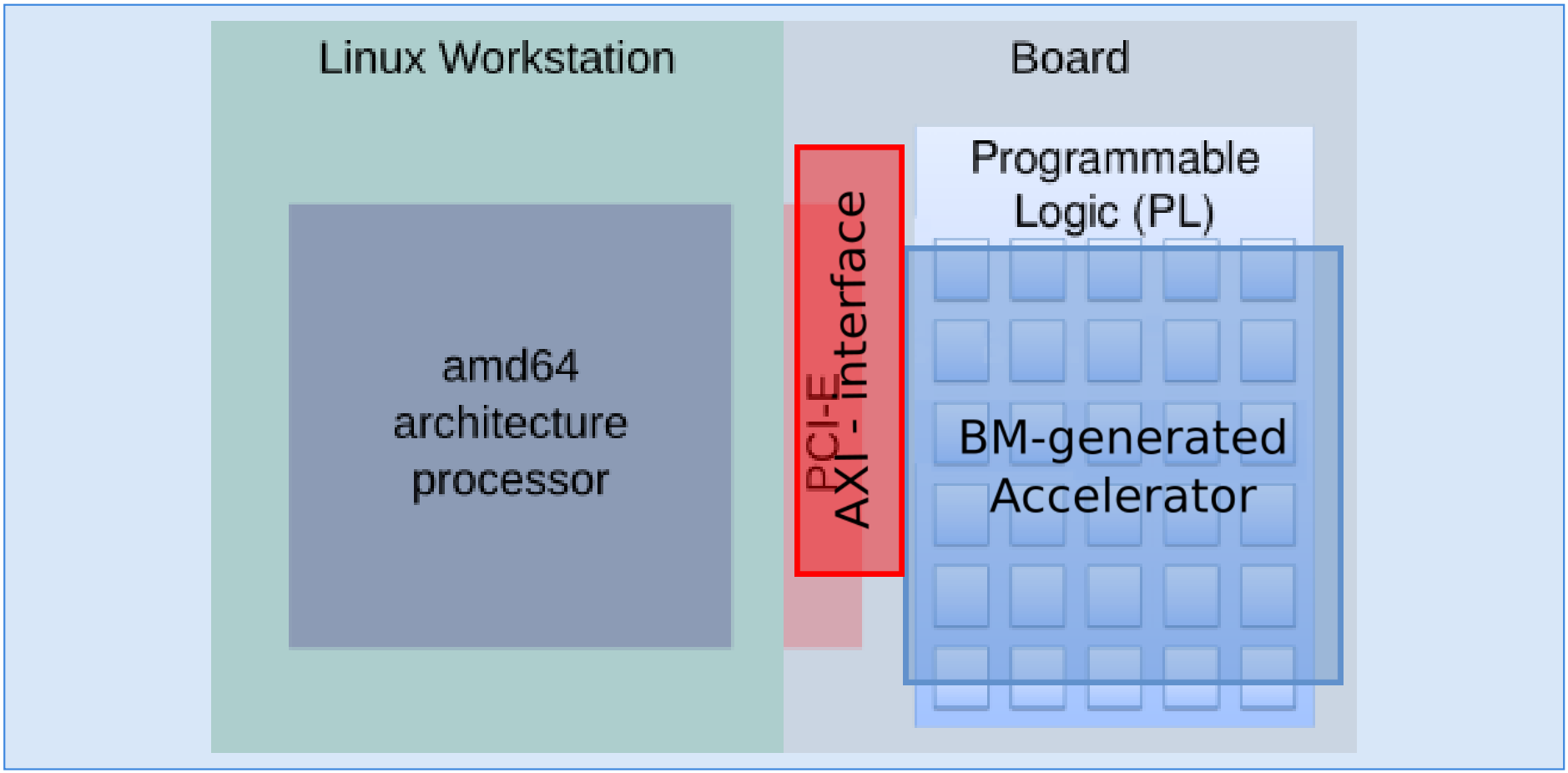
Accelerators

PCI-express boards



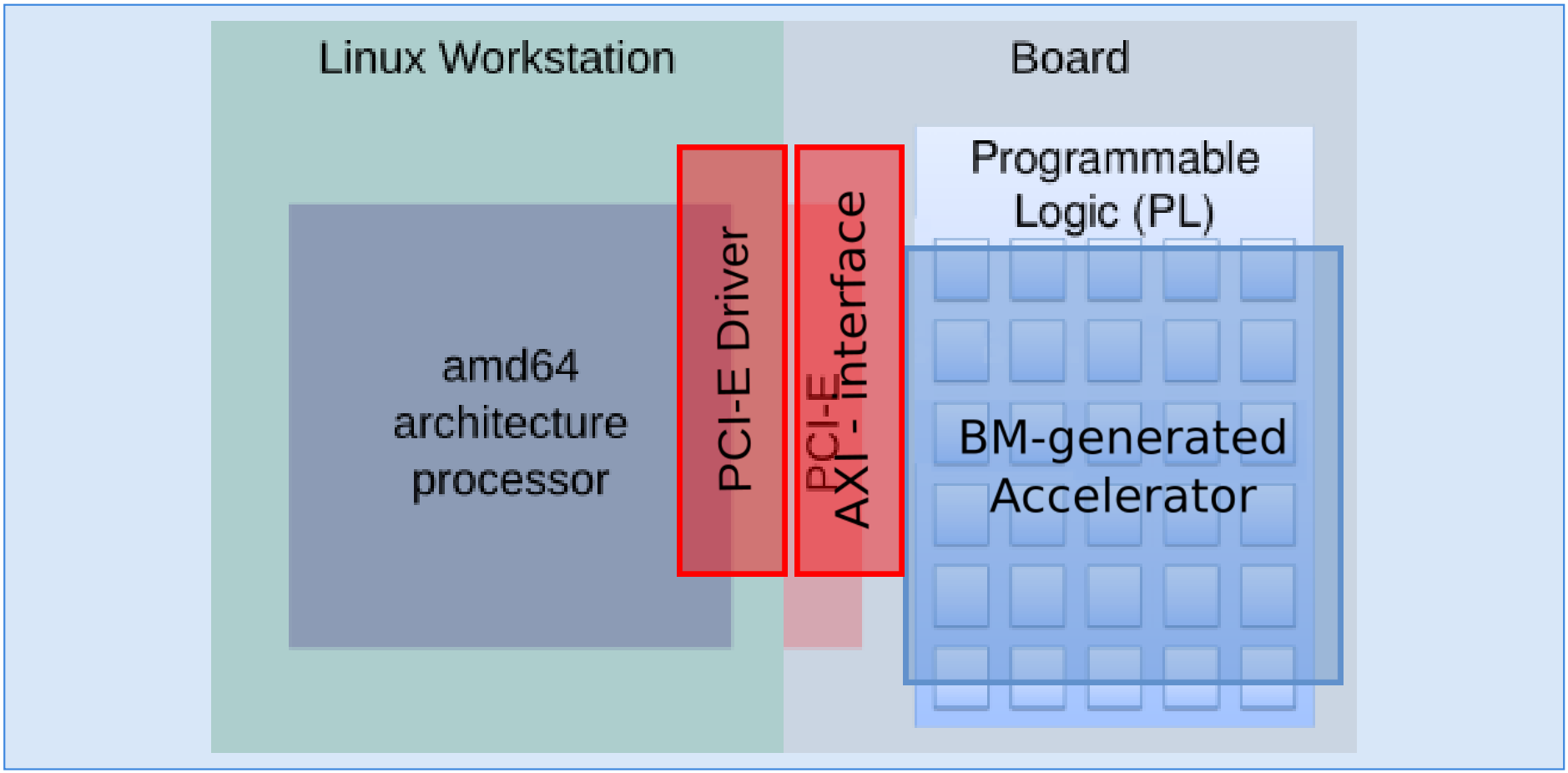
Accelerators

PCI-express boards



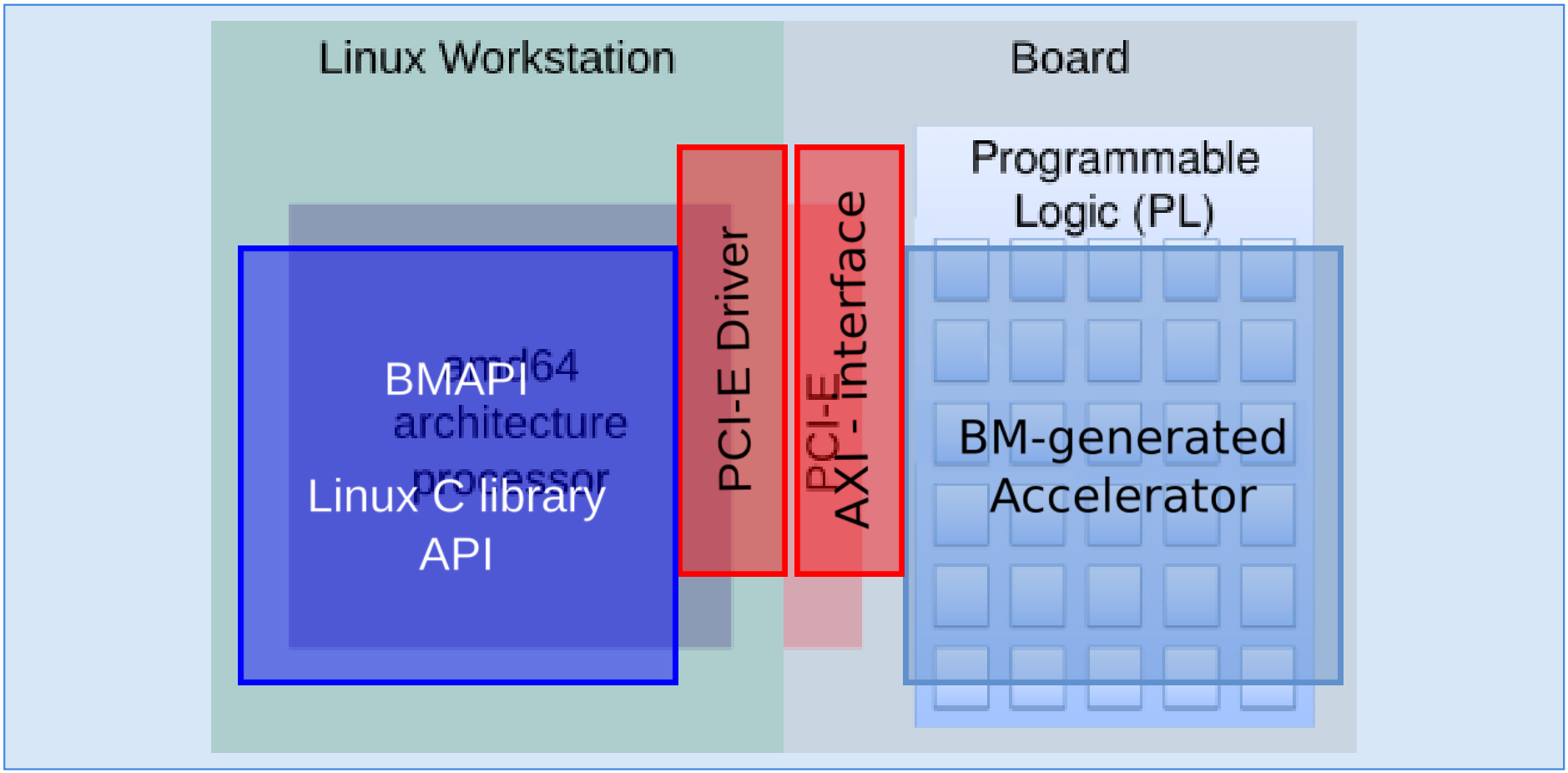
Accelerators

PCI-express boards



Accelerators

PCI-express boards

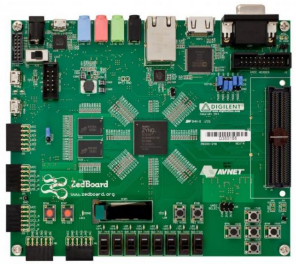


Accelerators

Hardware

Hybrid chips

Digilent Zedboard

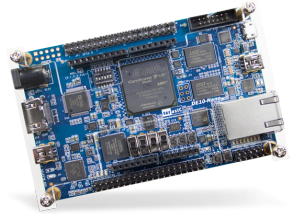


Zynq-7000 SoC XC7Z020
512 MB DDR3
Up to 667 MHz

Xilinx ZC702

Zynq-7000 SoC XC7Z020
1GB DDR3
85k cells - 220 DSP slices

Terasic DE10Nano



Intel Cyclone V
1GB DDR3 SDRAM
110K LEs

PCI-Express board

Xilinx KC705

Kintex-7 FPGAs
1GB DDR3 SODIM
326k cells - 840 DSP slices

Accelerators

Cloud

FPGA accelerators can be used in the cloud:

- Several public cloud providers offers solution of VM connected to FPGAs (Amazon, Nimbix)
- FPGAs can be inserted in private clouds infrastructures

To be used a firmware has to be uploaded to the accelerated VM FPGA

The BondMachine toolkit can be used to build such firmware

Accelerators

Cloud

FPGA accelerators can be used in the cloud:

- Several public cloud providers offers solution of VM connected to FPGAs (Amazon, Nimbix)
- FPGAs can be inserted in private clouds infrastructures

To be used a firmware has to be uploaded to the accelerated VM FPGA

The BondMachine toolkit can be used to build such firmware

Accelerators

Cloud

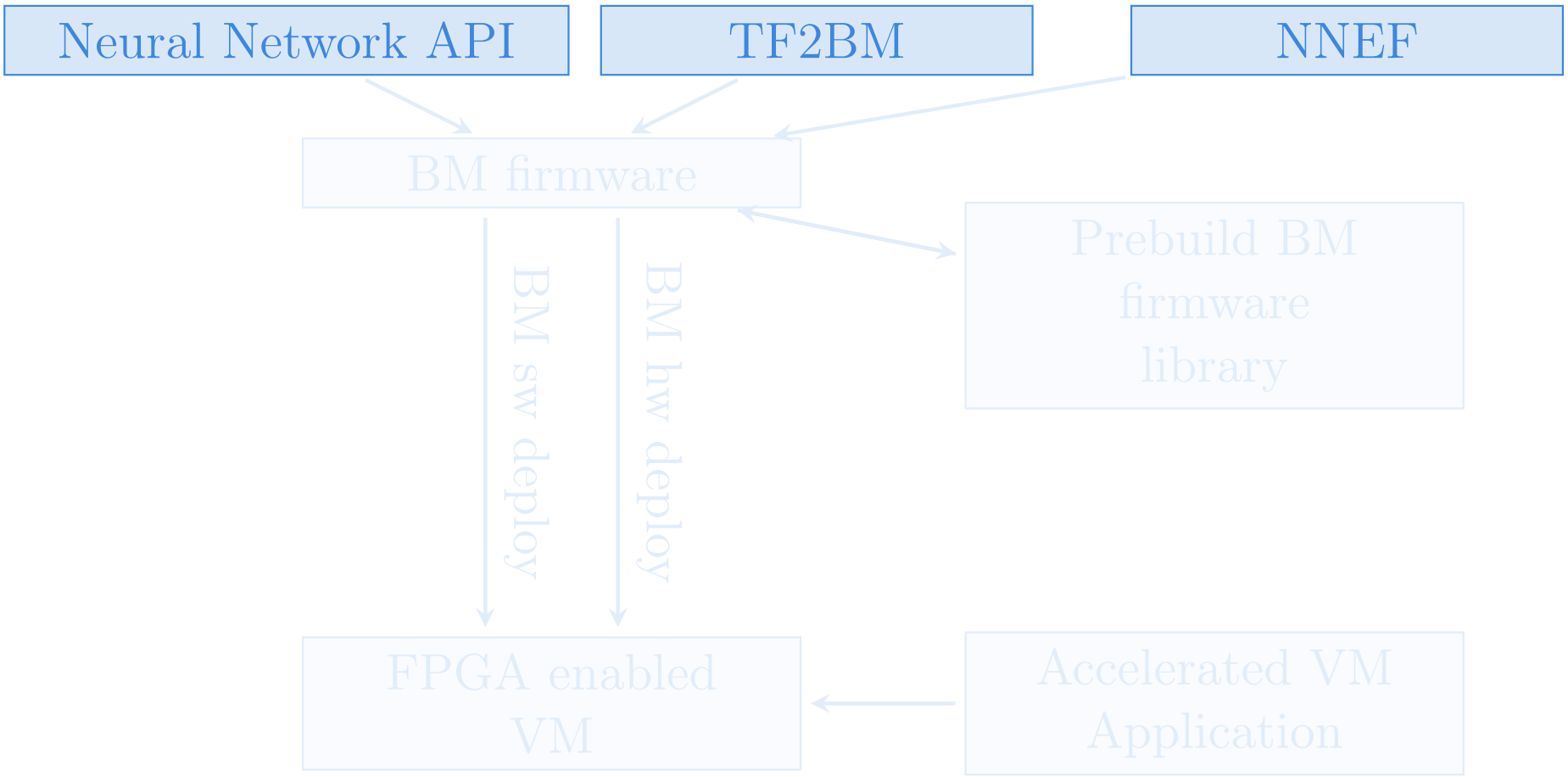
FPGA accelerators can be used in the cloud:

- Several public cloud providers offers solution of VM connected to FPGAs (Amazon, Nimbix)
- FPGAs can be inserted in private clouds infrastructures

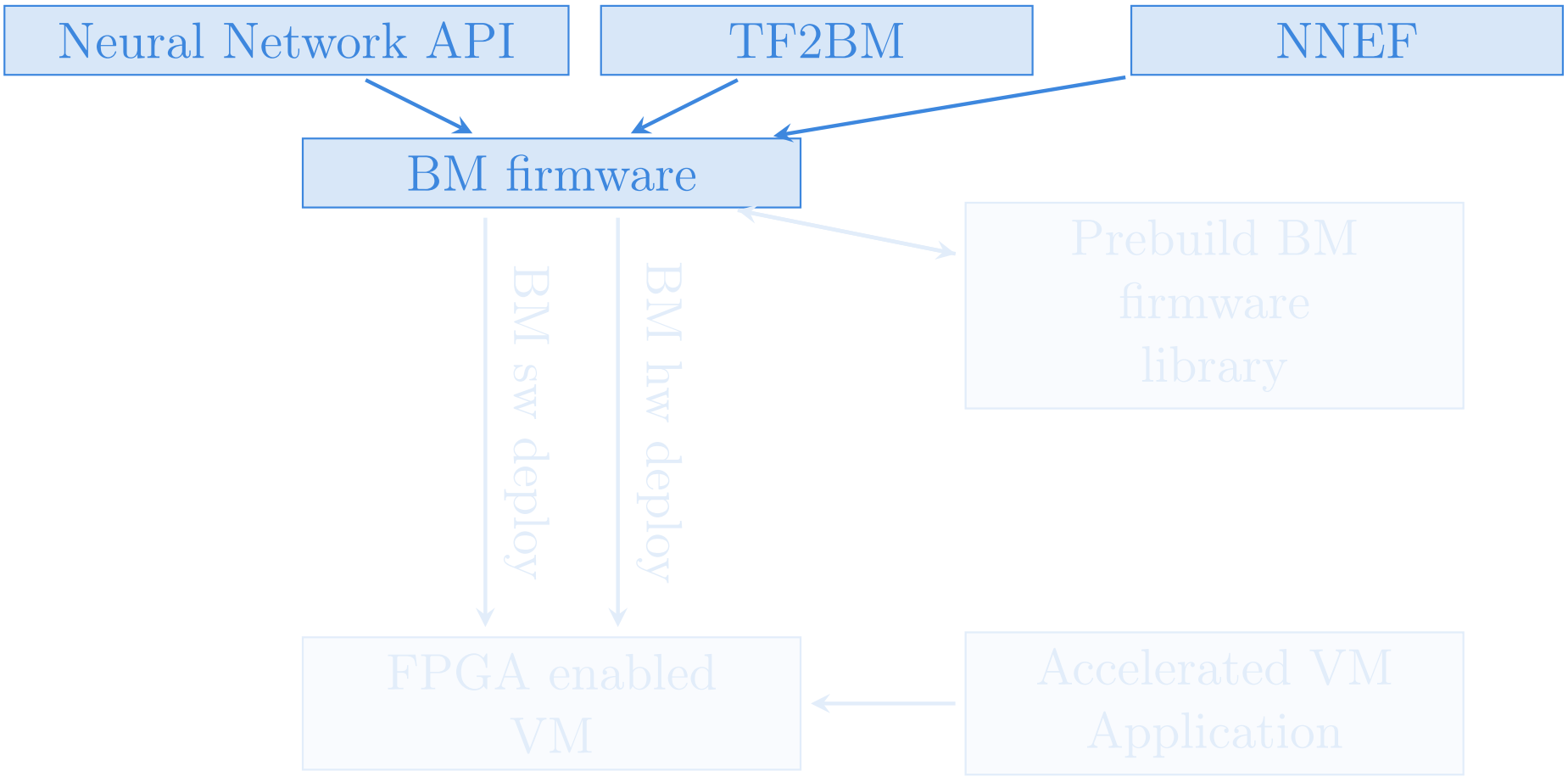
To be used a firmware has to be uploaded to the accelerated VM FPGA

The BondMachine toolkit can be used to build such firmware

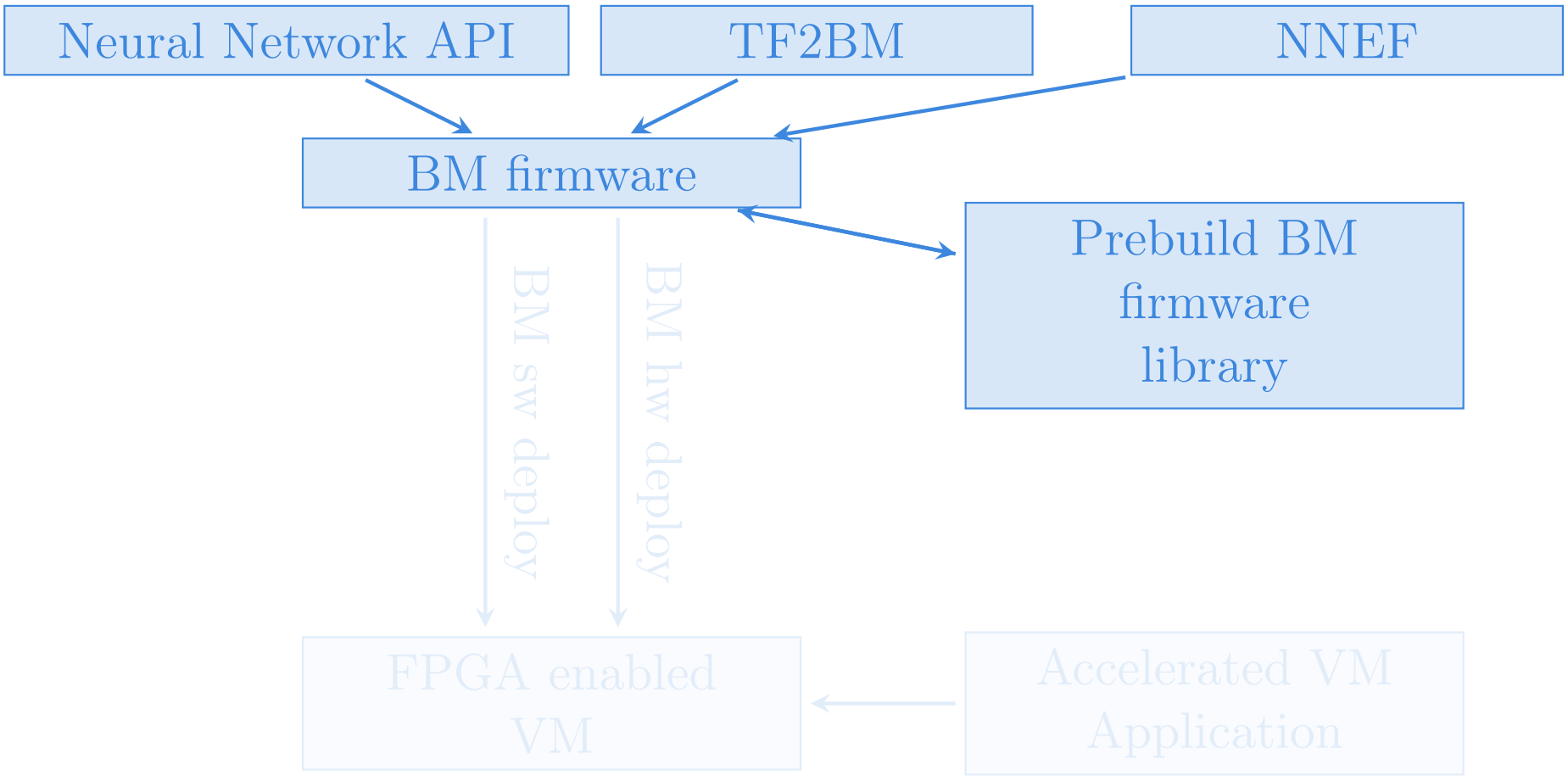
Accelerated ML in the Cloud



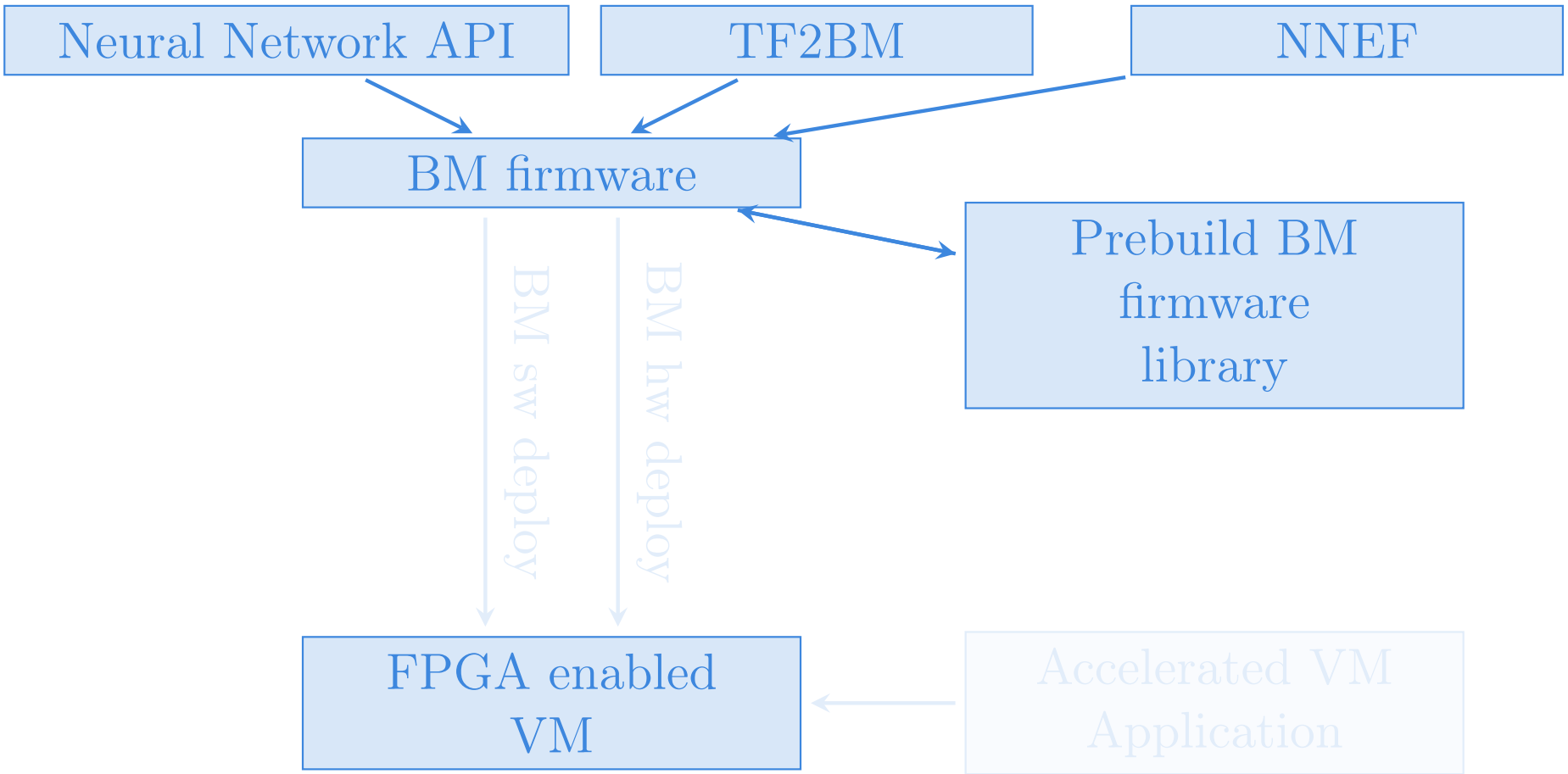
Accelerated ML in the Cloud



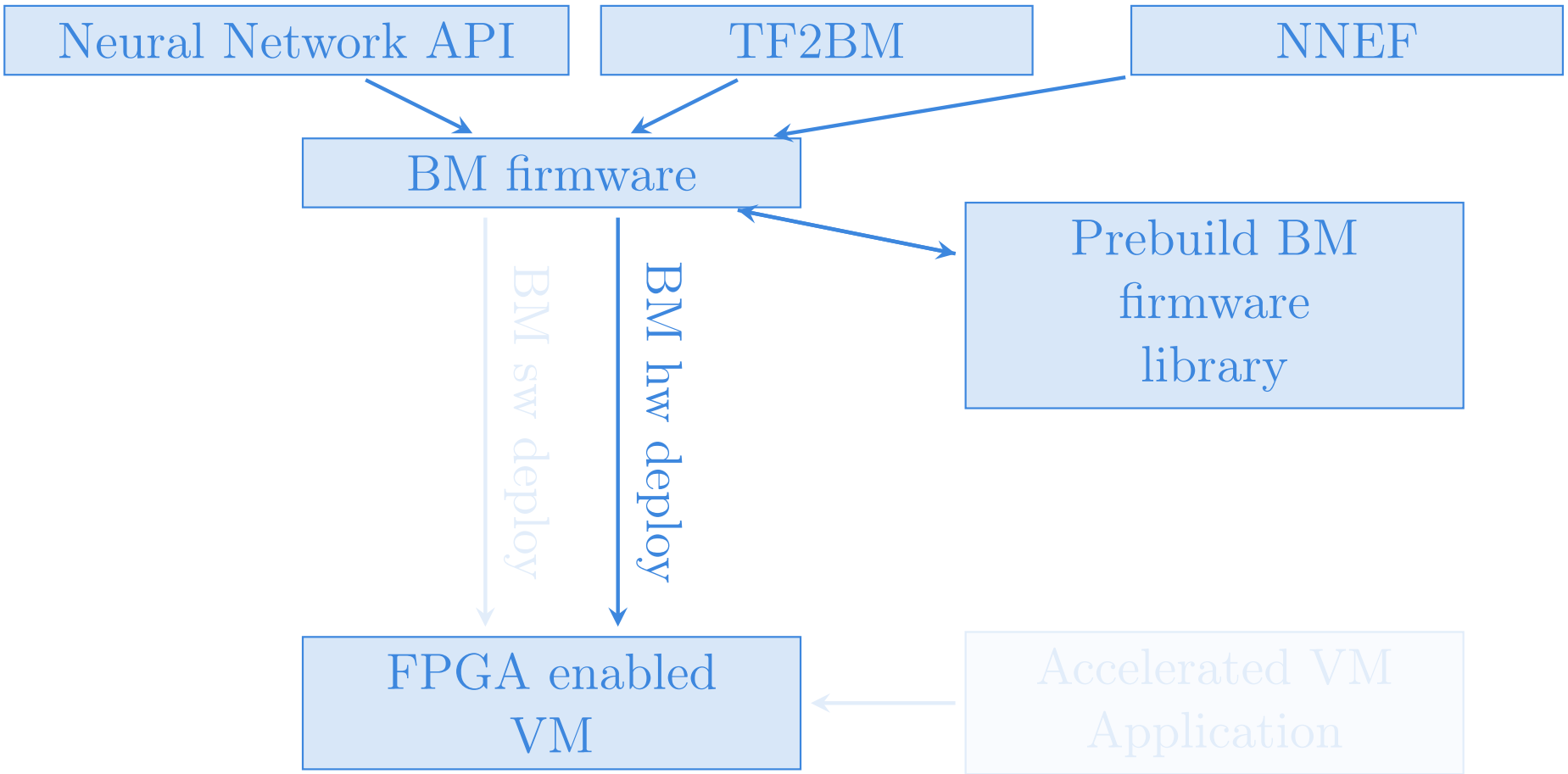
Accelerated ML in the Cloud



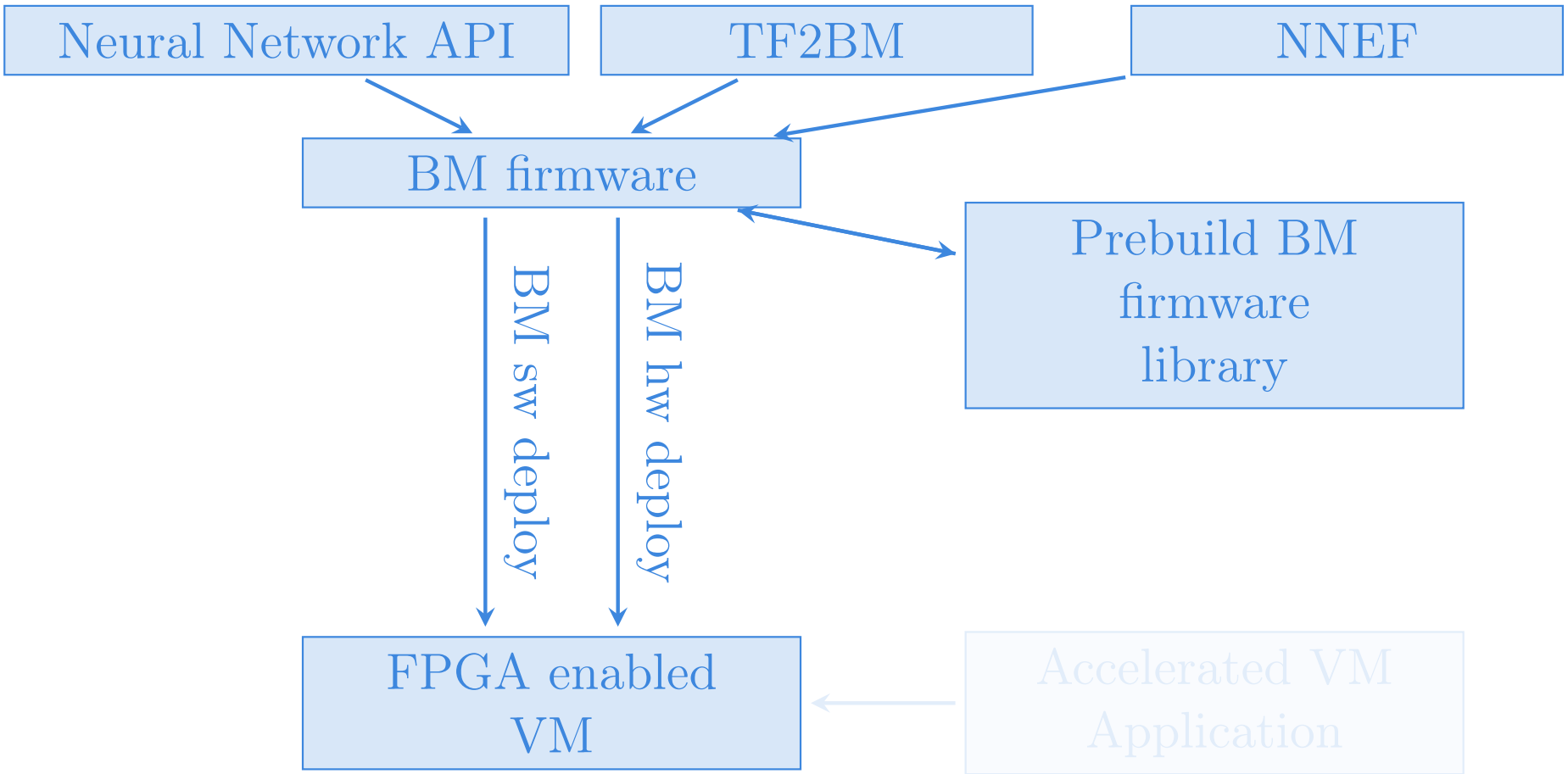
Accelerated ML in the Cloud



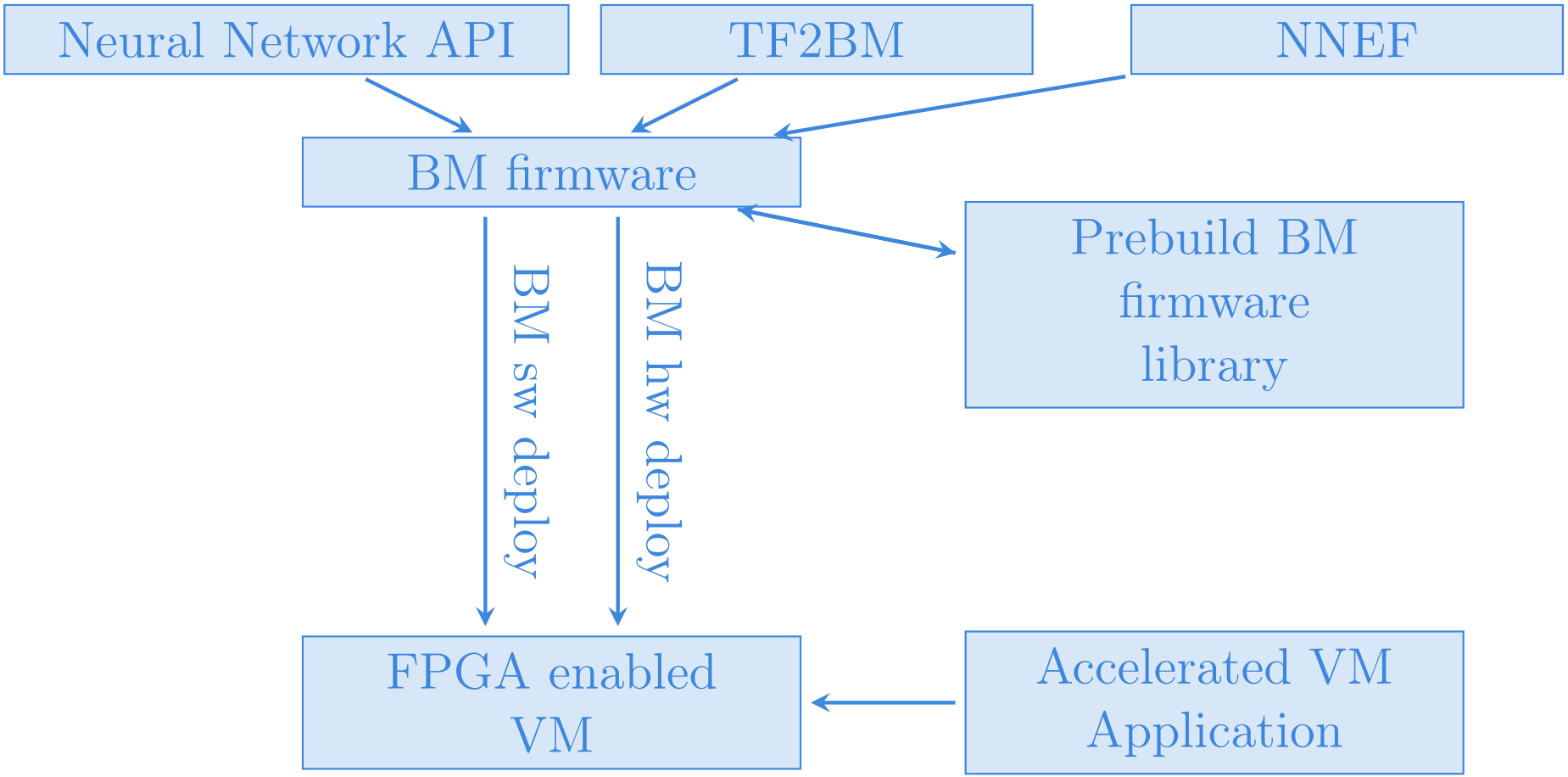
Accelerated ML in the Cloud



Accelerated ML in the Cloud



Accelerated ML in the Cloud



Project History

Project History

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.

Project History

- May 2016 - Idea presented at INFN-computing and Networking Commission Workshop 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 2018

Project History

- May 2016 - Idea presented at INFN-computing and Networking Commission Workshop 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 2018

Project History

- May 2016 - Idea presented at INFN-computing and Networking Commission Workshop 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 2018

Project History

InnovateFPGA 2018

- Feb 2018 - Reached the EMEA Semifinal.
- Jun 2018 - Reached the EMEA Regional final.
- Jul 2018 - EMEA Silver Award, Reached the Grand Final.
- Aug 2018 - Presented at Intel Campus, Santa Jose (CA) .
- Aug 2018 - Won the Iron Award in the Grand Final.

Project History

InnovateFPGA 2018

- Feb 2018 - Reached the EMEA Semifinal.
- Jun 2018 - Reached the EMEA Regional final.
- Jul 2018 - EMEA Silver Award, Reached the Grand Final.
- Aug 2018 - Presented at Intel Campus, Santa Jose (CA) .
- Aug 2018 - Won the Iron Award in the Grand Final.

Project History

InnovateFPGA 2018

- Feb 2018 - Reached the EMEA Semifinal.
- Jun 2018 - Reached the EMEA Regional final.
- Jul 2018 - EMEA Silver Award, Reached the Grand Final.
- Aug 2018 - Presented at Intel Campus, Santa Jose (CA) .
- Aug 2018 - Won the Iron Award in the Grand Final.

Project History

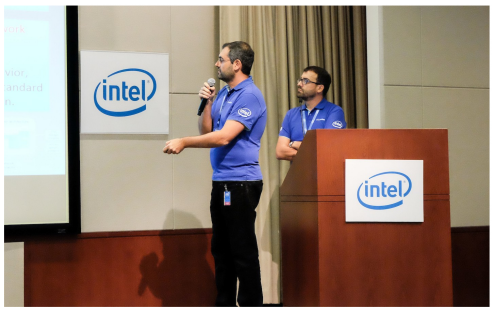
InnovateFPGA 2018

- Feb 2018 - Reached the EMEA Semifinal.
- Jun 2018 - Reached the EMEA Regional final.
- Jul 2018 - EMEA Silver Award, Reached the Grand Final.
- Aug 2018 - Presented at Intel Campus, Santa Jose (CA) .
- Aug 2018 - Won the Iron Award in the Grand Final.

Project History

InnovateFPGA 2018

- Feb 2018 - Reached the EMEA Semifinal.
- Jun 2018 - Reached the EMEA Regional final.
- Jul 2018 - EMEA Silver Award, Reached the Grand Final.
- Aug 2018 - Presented at Intel Campus, Santa Jose (CA) .
- Aug 2018 - Won the Iron Award in the Grand Final.



Conclusions

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.

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, ranging from small interconnected IoT devices to Machine Learning accelerators.

Next few months goals

- Improve the use of BondMachines as accelerators, integrating them into the ecosystem
- Start making benchmarks (the real missing piece)
- Find a way to sustain the project
- Move all the code to github
- Integrate low and trans-precision instructions

Next few months goals

- Improve the use of BondMachines as accelerators, integrating them into the ecosystem
- Start making benchmarks (the real missing piece)
- Find a way to sustain the project
- Move all the code to github
- Integrate low and trans-precision instructions

Next few months goals

- Improve the use of BondMachines as accelerators, integrating them into the ecosystem
- Start making benchmarks (the real missing piece)
- Find a way to sustain the project
- Move all the code to github
- Integrate low and trans-precision instructions

Next few months goals

- Improve the use of BondMachines as accelerators, integrating them into the ecosystem
- Start making benchmarks (the real missing piece)
- Find a way to sustain the project
- Move all the code to github
- Integrate low and trans-precision instructions

Next few months goals

- Improve the use of BondMachines as accelerators, integrating them into the ecosystem
- Start making benchmarks (the real missing piece)
- Find a way to sustain the project
- Move all the code to github
- Integrate low and trans-precision instructions

Future work

The project is at the stage of a working prototype, so work has to be done in several areas:

- Include new processor shared objects and currently unsupported opcodes.
- Extend the compiler to include more data structures.
- Improve the networking including new interconnection firmwares.

What would an OS for BondMachines look like ?

Future work

The project is at the stage of a working prototype, so work has to be done in several areas:

- Include new processor shared objects and currently unsupported opcodes.
- Extend the compiler to include more data structures.
- Improve the networking including new interconnection firmwares.

What would an OS for BondMachines look like ?

Future work

The project is at the stage of a working prototype, so work has to be done in several areas:

- Include new processor shared objects and currently unsupported opcodes.
- Extend the compiler to include more data structures.
- Improve the networking including new interconnection firmwares.

What would an OS for BondMachines look like ?

Future work

The project is at the stage of a working prototype, so work has to be done in several areas:

- Include new processor shared objects and currently unsupported opcodes.
- Extend the compiler to include more data structures.
- Improve the networking including new interconnection firmwares.

What would an OS for BondMachines look like ?

Future work

The project is at the stage of a working prototype, so work has to be done in several areas:

- Include new processor shared objects and currently unsupported opcodes.
- Extend the compiler to include more data structures.
- Improve the networking including new interconnection firmwares.

What would an OS for BondMachines look like ?

If you have question/curiosity on the project:

Mirko Mariotti
mirko.mariotti@unipg.it
http://bondmachine.fisica.unipg.it