

# A Concept for a Quantum-Classical Hybrid Computing System: From Low to High Layers

Masao HIROKAWA

Kyushu University

September 28, 2026

# A Concept for a Quantum-Classical Hybrid Computing System: From Low to High Layers

Masao HIROKAWA

**The computing systems humanity has successfully built to date are only for classical computation and operate entirely within the realm of classical computation.**

**While providing an overview of the classical computing system, we identify the challenges involved in constructing a quantum computing system.**

# What Is Computing System?

Computing system of conventional computer

## Inputting Source Program

- # reads the program character by character,
- # extracts and understands the statements written in the program,
- # converts a program into machine language (i.e., binary code),
- ↳ maps the program, converted into binary code, to (vast) bit signals,
- ↳ passes the bit signals to its CPU,
  - ↳ splits the bit signals into several parts and maps them to execution units according to mapping information in the decode circuit, thereby determining their spatial destinations.
  - ↳ feeds the mapped bit signals into the execution units in accordance with the scheduling information in the scheduler circuit, and then, processes them over time.

## Outputting Calculation Results

# What Is Computing System?

## Computing system of conventional computer

### Inputting Source Program

- 
- # reads the program character by character,
  - # extracts and understands the statements written in the program,
  - # converts a program into machine language (i.e., binary code),
  - ↳ maps the program, converted into binary code, to (vast) bit signals,
  - ↳ passes the bit signals to its CPU,
    - ↳ splits the bit signals into several parts and maps them to execution units according to mapping information in the decode circuit, thereby determining their spatial destinations.
    - ↳ feeds the mapped bit signals into the execution units in accordance with the scheduling information in the scheduler circuit, and then, processes them over time.

### Outputting Calculation Results

# What Is Computing System?

Computing system of conventional computer

## Inputting Source Program

- 
- # reads the program character by character,**
  - # extracts and understands the statements written in the program,**
  - # converts a program into machine language (i.e., binary code),**
  - b maps the program, converted into binary code, to (vast) bit signals,**
  - b passes the bit signals to its CPU,**
    - h splits the bit signals into several parts and maps them to execution units according to mapping information in the decode circuit, thereby determining their spatial destinations.**
    - h feeds the mapped bit signals into the execution units in accordance with the scheduling information in the scheduler circuit, and then, processes them over time.**

## Outputting Calculation Results

# What Is Computing System?

Computing system of conventional computer

## Inputting Source Program

- 
- # reads the program character by character,
  - # **extracts and understands the statements written in the program,**
  - # converts a program into machine language (i.e., binary code),
  - ↳ maps the program, converted into binary code, to (vast) bit signals,
  - ↳ passes the bit signals to its CPU,
    - ↳ splits the bit signals into several parts and maps them to execution units according to mapping information in the decode circuit, thereby determining their spatial destinations.
    - ↳ feeds the mapped bit signals into the execution units in accordance with the scheduling information in the scheduler circuit, and then, processes them over time.

## Outputting Calculation Results

# What Is Computing System?

Computing system of conventional computer

## Inputting Source Program

- 
- # reads the program character by character,
  - # extracts and understands the statements written in the program,
  - # **converts a program into machine language (i.e., binary code),**
  - ↳ maps the program, converted into binary code, to (vast) bit signals,
  - ↳ passes the bit signals to its CPU,
    - ↳ splits the bit signals into several parts and maps them to execution units according to mapping information in the decode circuit, thereby determining their spatial destinations.
    - ↳ feeds the mapped bit signals into the execution units in accordance with the scheduling information in the scheduler circuit, and then, processes them over time.

## Outputting Calculation Results

# What Is Computing System?

Computing system of conventional computer

## Inputting Source Program

- # reads the program character by character,
- # extracts and understands the statements written in the program,
- # converts a program into machine language (i.e., binary code),

handled by the language processing system  
(i.e., compiler in a broad sense)

- b maps the program into bit signals,
- b passes the bit signals to the execution units,
  - h splits the bit signals into several parts and maps them to execution units according to mapping information in the decode circuit, thereby determining their spatial destinations.
  - h feeds the mapped bit signals into the execution units in accordance with the scheduling information in the scheduler circuit, and then, processes them over time.

## Outputting Calculation Results

# What Is Computing System?

Computing system of conventional computer

## Inputting Source Program

- 
- # reads the program character by character,
  - # extracts and understands the statements written in the program,
  - # converts a program into machine language (i.e., binary code),
  - ↳ **maps the program, converted into binary code, to (vast) bit signals,**
  - ↳ passes the bit signals to its CPU,
    - ↳ splits the bit signals into several parts and maps them to execution units according to mapping information in the decode circuit, thereby determining their spatial destinations.
    - ↳ feeds the mapped bit signals into the execution units in accordance with the scheduling information in the scheduler circuit, and then, processes them over time.

## Outputting Calculation Results

# What Is Computing System?

Computing system of conventional computer

## Inputting Source Program

# reads the program character by character  
# extracts and understands the statements  
# converts a program into machine language (i.e., binary code)

**implemented as an electrical signal**

- ↳ maps the program, converted into binary code, to (vast) **bit signals**,
- ↳ passes the bit signals to its CPU,
  - ↳ splits the bit signals into several parts and maps them to execution units according to mapping information in the decode circuit, thereby determining their spatial destinations.
  - ↳ feeds the mapped bit signals into the execution units in accordance with the scheduling information in the scheduler circuit, and then, processes them over time.

## Outputting Calculation Results

# What Is Computing System?

Computing system of conventional computer

## Inputting Source Program

- # reads the program character by character,
- # extracts and understands the statements written in the program,
- # converts a program into machine language (i.e., binary code),
- ↳ maps the program, converted into binary code, to (vast) bit signals,
- ↳ **passes the bit signals to its CPU,**
  - ↳ splits the bit signals into several parts and maps them to execution units according to mapping information in the decode circuit, thereby determining their spatial destinations.
  - ↳ feeds the mapped bit signals into the execution units in accordance with the scheduling information in the scheduler circuit, and then, processes them over time.

## Outputting Calculation Results

# What Is Computing System?

Computing system of conventional computer

## Inputting Source Program

- 
- # reads the program character by character,
  - # extracts and understands the statements written in the program,
  - # converts a program into machine language (i.e., binary code),
  - ↳ maps the program, converted into binary code, to (vast) bit signals,
  - ↳ **passes the bit signals to its CPU,**
    - ↳ **splits the bit signals into several parts and maps them to execution units according to mapping information in the decode circuit, thereby determining their spatial destinations.**
    - ↳ feeds the mapped bit signals into the execution units in accordance with the scheduling information in the scheduler circuit, and then, processes them over time.

## Outputting Calculation Results

# What Is Computing System?

Computing system of conventional computer

## Inputting Source Program

- 
- # reads the program character by character,
  - # extracts and understands the statements written in the program,
  - # converts a program into machine language (i.e., binary code),
  - ↳ maps the program, converted into binary code, to (vast) bit signals,
  - ↳ passes the bit signals to its CPU,
    - ↳ splits the bit signals into several parts and maps them to execution units according to **mapping** information in the **decode circuit**, thereby determining their spatial destinations.
    - ↳ feeds the mapped bit signals into the execution units in accordance with the scheduling information in the scheduler circuit, and then, processes them over time.

## Outputting Calculation Results

# What Is Computing System?

Computing system of conventional computer

## Inputting Source Program

- 
- # reads the program character by character,
  - # extracts and understands the statements written in the program,
  - # converts a program into machine language (i.e., binary code),
  - ↳ maps the program, converted into binary code, to (vast) bit signals,
  - ↳ **passes the bit signals to its CPU,**
    - ↳ splits the bit signals into several parts and maps them to execution units according to mapping information in the decode circuit, thereby determining their spatial destinations.
    - ↳ **feeds the mapped bit signals into the execution units in accordance with the scheduling information in the scheduler circuit, and then, processes them over time.**

## Outputting Calculation Results

# What Is Computing System?

Computing system of conventional computer

## Inputting Source Program

- 
- # reads the program character by character,
  - # extracts and understands the statements written in the program,
  - # converts a program into machine language (i.e., binary code),
  - ↳ maps the program, converted into binary code, to (vast) bit signals,
  - ↳ passes the bit signals to its CPU,
    - ↳ splits the bit signals into several parts and maps them to execution units according to mapping information in the decode circuit, thereby determining their spatial destinations.
    - ↳ feeds the mapped bit signals into the execution units in accordance with the **scheduling** information in the **scheduler circuit**, and then, processes them over time.

## Outputting Calculation Results

# What Is Computing System?

Computing system of conventional computer

## Inputting Source Program

# reads the program character by character,  
# extracts and understands the statements written in the program,  
# converts a program into machine code

**designed by the computer architecture**

- ↳ maps the program, converted into binary code, to (vast) bit signals,
- ↳ passes the bit signals to its CPU,
  - ↳ splits the bit signals into several parts and maps them to execution units according to mapping information in the decode circuit, thereby determining their spatial destinations.
  - ↳ feeds the mapped bit signals into the execution units in accordance with the scheduling information in the scheduler circuit, and then, processes them over time.

## Outputting Calculation Results

# What Is Computing System?

Computing system of conventional computer

Inputting Source Program

- # reads the program character by character,
- # extracts and understands the statements written in the program,
- # converts a program into machine language (i.e., binary code),

## Separation of Software & Hardware

- ↳ maps the program, converted into binary code, to (vast) bit signals,
- ↳ passes the bit signals to its CPU,
  - ↳ splits the bit signals into several parts and maps them to execution units according to mapping information in the decode circuit, thereby determining their spatial destinations.
  - ↳ feeds the mapped bit signals into the execution units in accordance with the scheduling information in the scheduler circuit, and then, processes them over time.

Outputting Calculation Results

# What Is Computing System?

## Computing system of conventional computer

### Inputting Source Program

- # reads the program character by character,
- # extracts and understands the statements written in the program,
- # converts a program into machine language (i.e., binary code)

- b maps the program,
- b passes the bit signals

**Only some execution units can benefit from the advantages of quantum computation.**

- ↳ splits the bit signals into several parts and maps them to execution units according to mapping information in the decode circuit, thereby determining their spatial destinations.
- ↳ feeds the mapped bit signals into the **execution units** in accordance with the scheduling information in the scheduler circuit, and then, processes them over time.

### Outputting Calculation Results

# What Is Computing System?

Computing system of conventional computer

## Inputting Source Program

- # reads the program character by character,
- # extracts and understands the statements written in the program,
- # converts a program into machine language (i.e. binary code)
- ↳ maps the program,
- ↳ passes the bit signals,
  - ↳ splits the bit signals into several parts and maps them to execution units according to mapping information in the decode circuit, thereby determining their spatial destinations.
  - ↳ feeds the mapped bit signals into the execution units in accordance with the scheduling information in the scheduler circuit, and then, processes them over time.

**Only some execution units can benefit from the advantages of quantum computation.**

**All other units are implemented using classical computation.**

## Outputting Calculation Results

# What Is Computing System?

Computing system of conventional computer

Inputting Source Program

Quantum computing requires significant assistance from classical computing.

- # reads the program character by character,
- # extracts and understands the statements written in the program,
- # converts a program into machine language (i.e. binary code)
- ↳ maps the program,
- ↳ passes the bit signals,
  - ↳ splits the bit signals into several parts and maps them to execution units according to mapping information in the decode circuit, thereby determining their spatial destinations.
  - ↳ feeds the mapped bit signals into the execution units in accordance with the scheduling information in the scheduler circuit, and then, processes them over time.

Only some execution units can benefit from the advantages of quantum computation.

All other units are implemented using classical computation.

Outputting Calculation Results

# What Is Computing System?

Computing system of conventional computer

Inputting Source Program

Quantum computing requires significant assistance from classical computing.

Quantum-Classical Hybrid Computing System

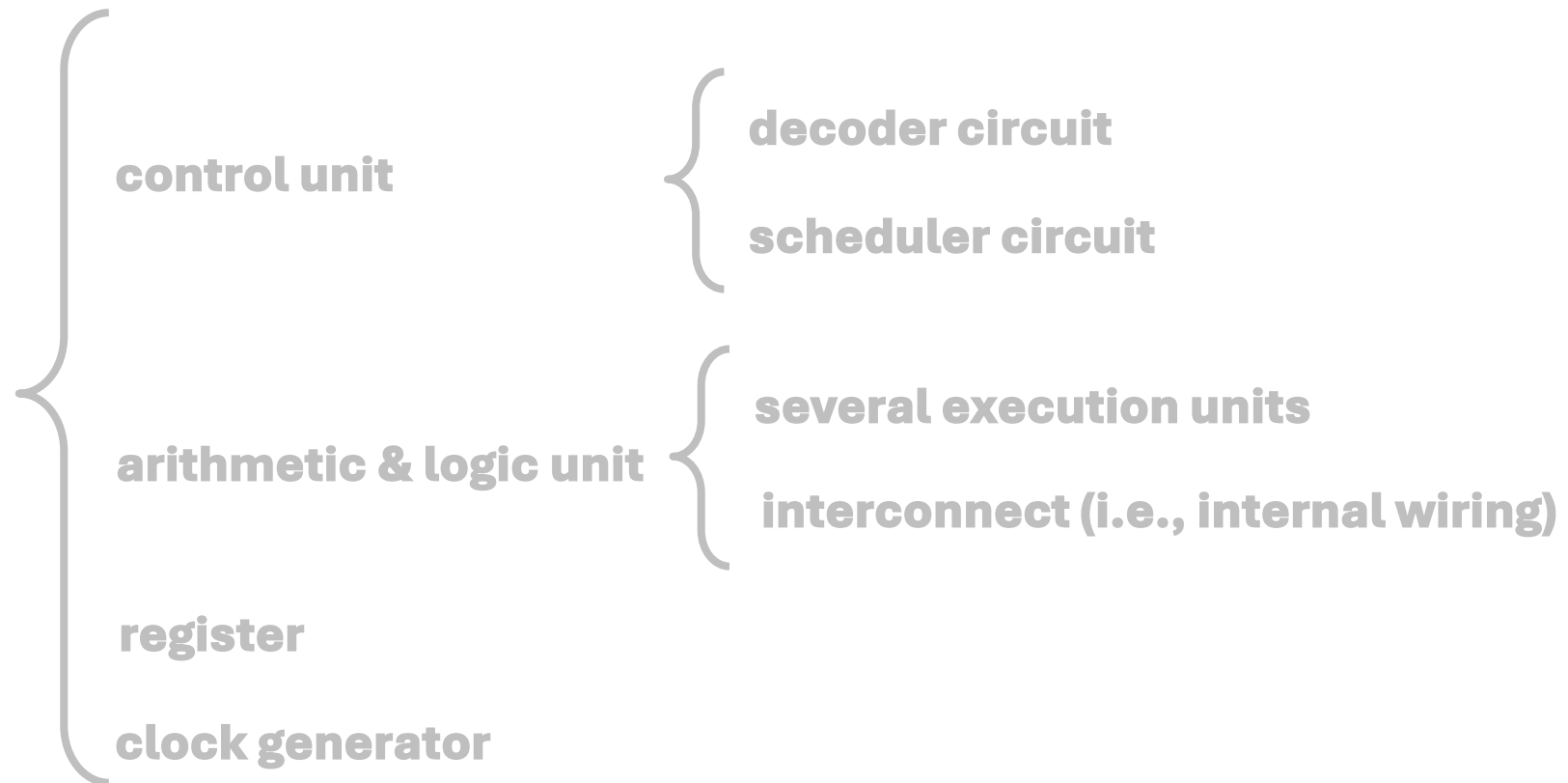
Benefit from the advantages of quantum computation.

All other units are implemented using classical computation.

Outputting Calculation Results

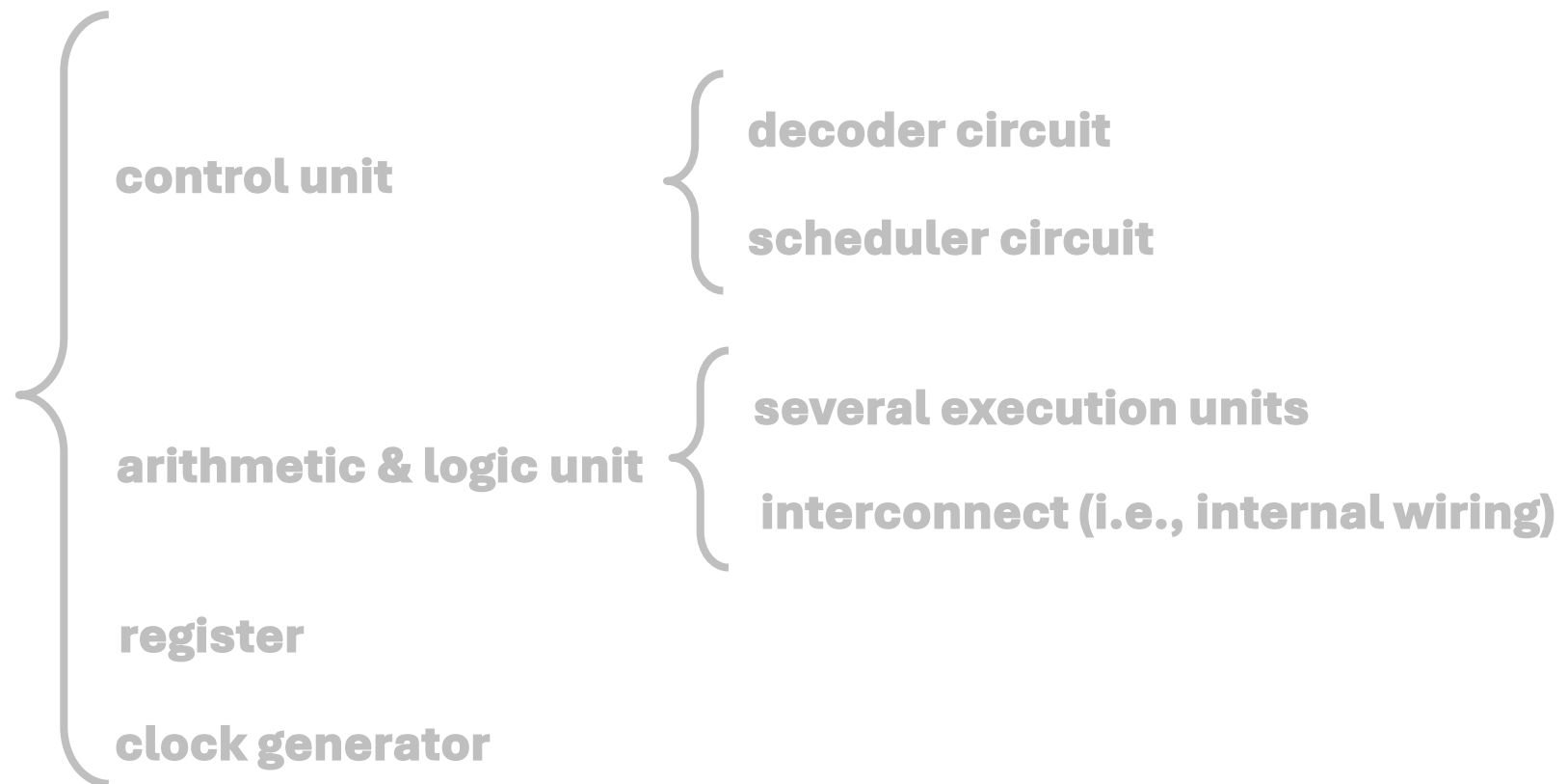
# CPU (Central Processing Unit)

- A CPU is basically a gate-level netlist etched and fixed as a concrete physical implementation.
- An example of the basic configuration of a CPU



# CPU (Central Processing Unit)

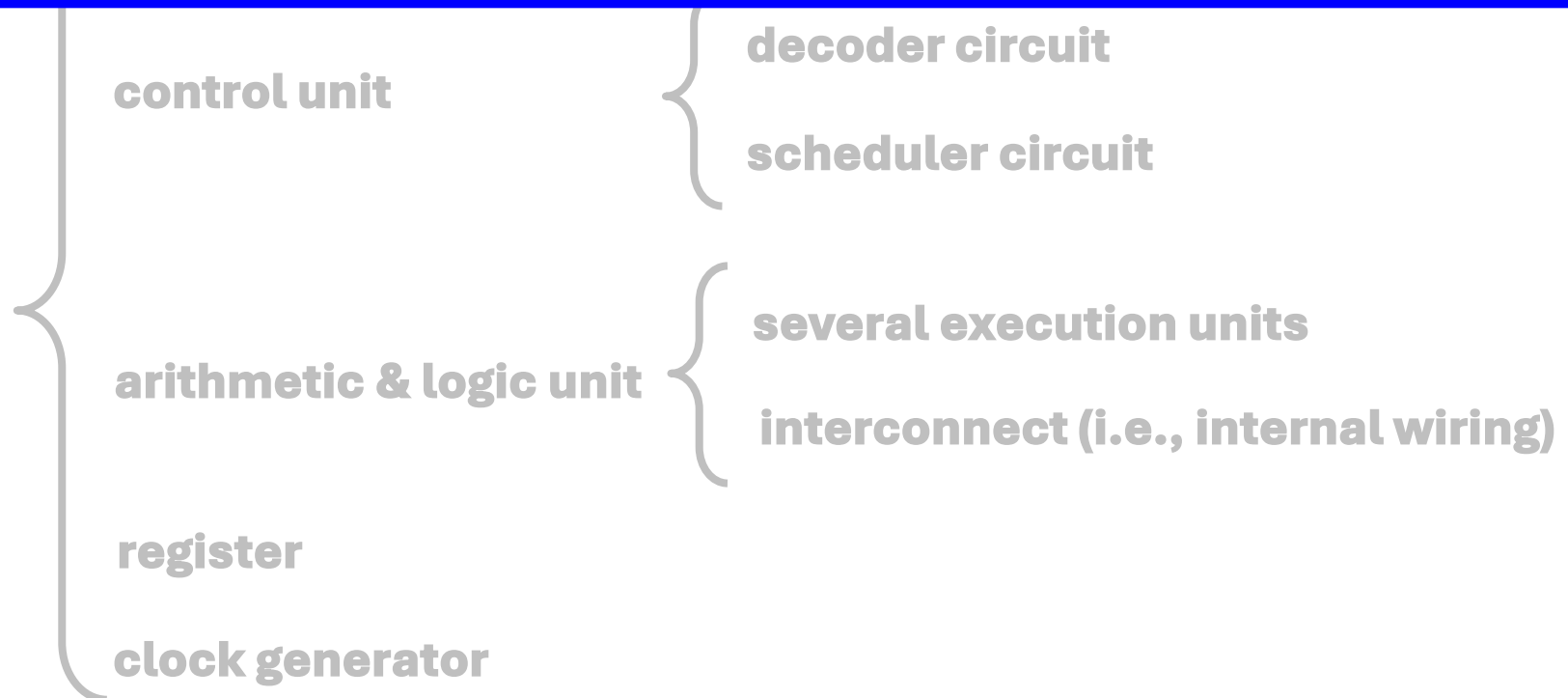
- **A CPU is basically a gate-level netlist etched and fixed as a concrete physical implementation.**
- An example of the basic configuration of a CPU



# CPU (Central Processing Unit)

- A CPU is basically a **gate-level netlist** etched and fixed as a concrete physical implementation.

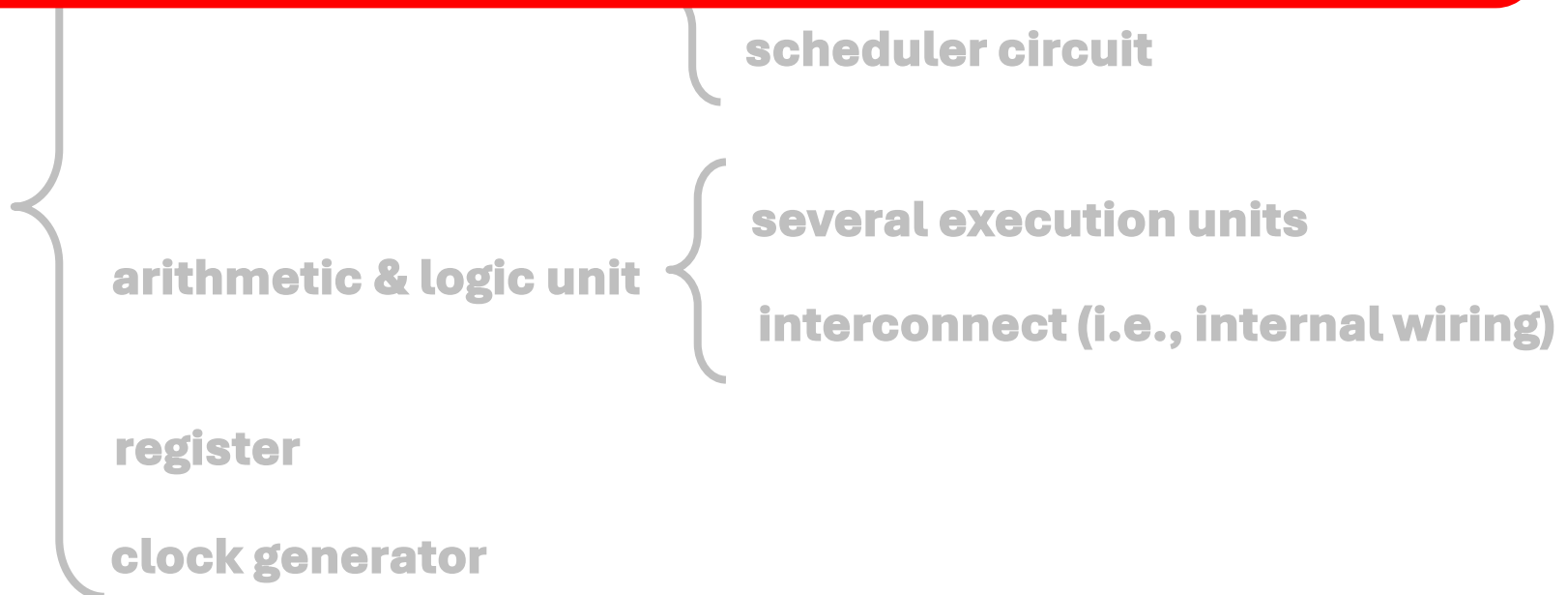
▪ Architecture is represented by **circuit data that represents digital logic gates and their interconnections**



# CPU (Central Processing Unit)

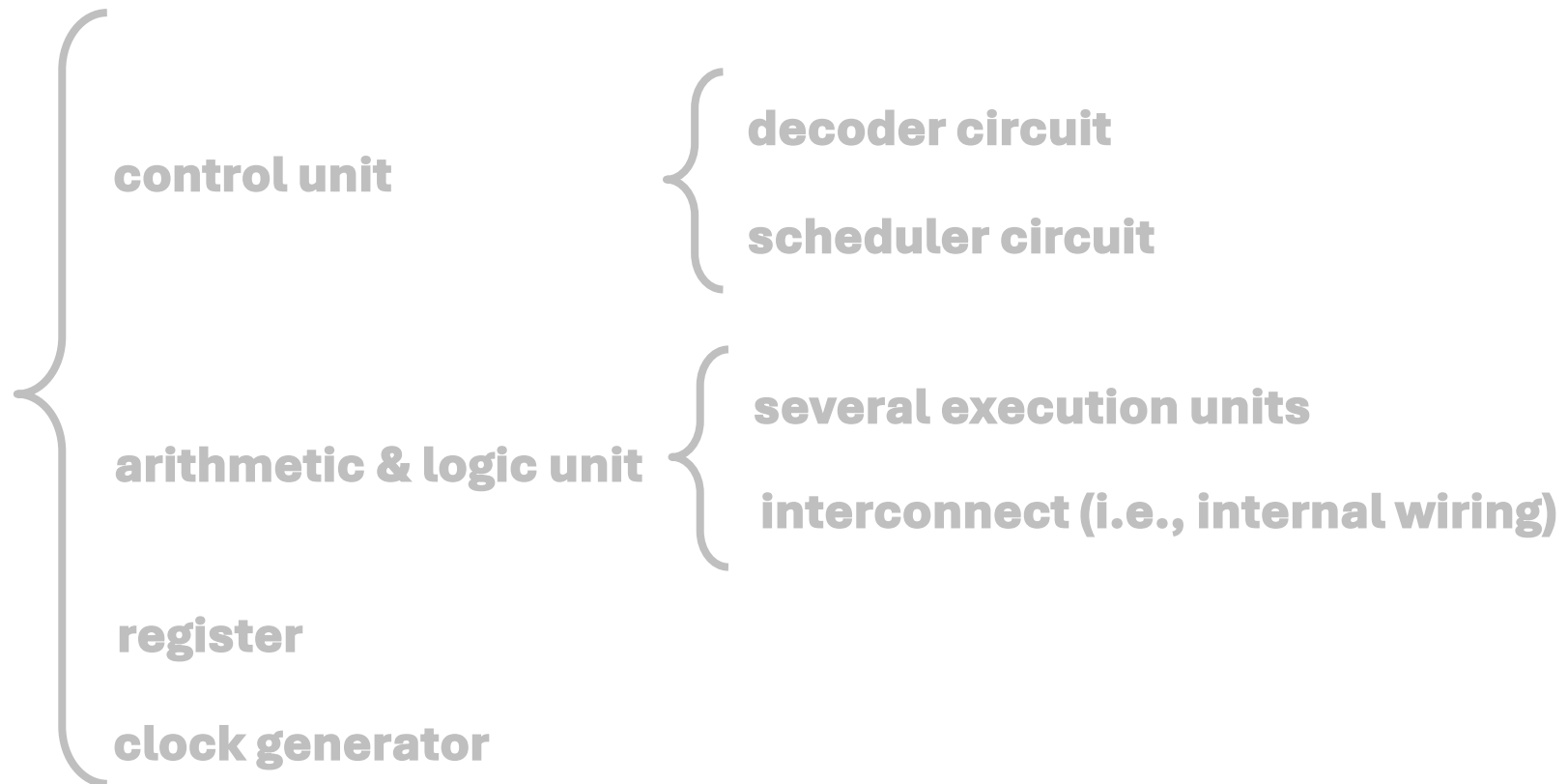
- A **CPU** is basically a gate-level netlist **etched and fixed** as a concrete physical implementation.
- An example of the basic configuration of a CPU

**A general-purpose gate-level netlist is hard-wired into the processor, and its size is limited.**



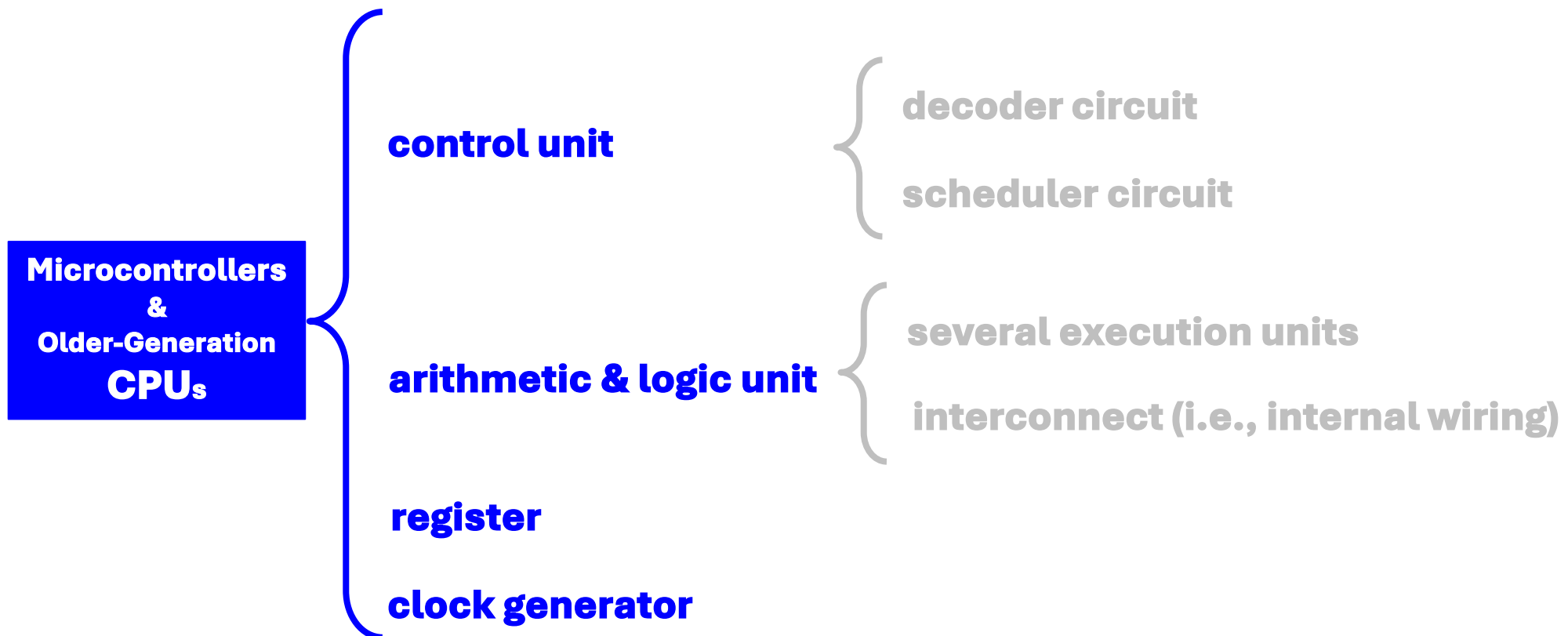
# CPU (Central Processing Unit)

- A CPU is basically a gate-level netlist etched and fixed as a concrete physical implementation.
- **An example of the basic configuration of a CPU**



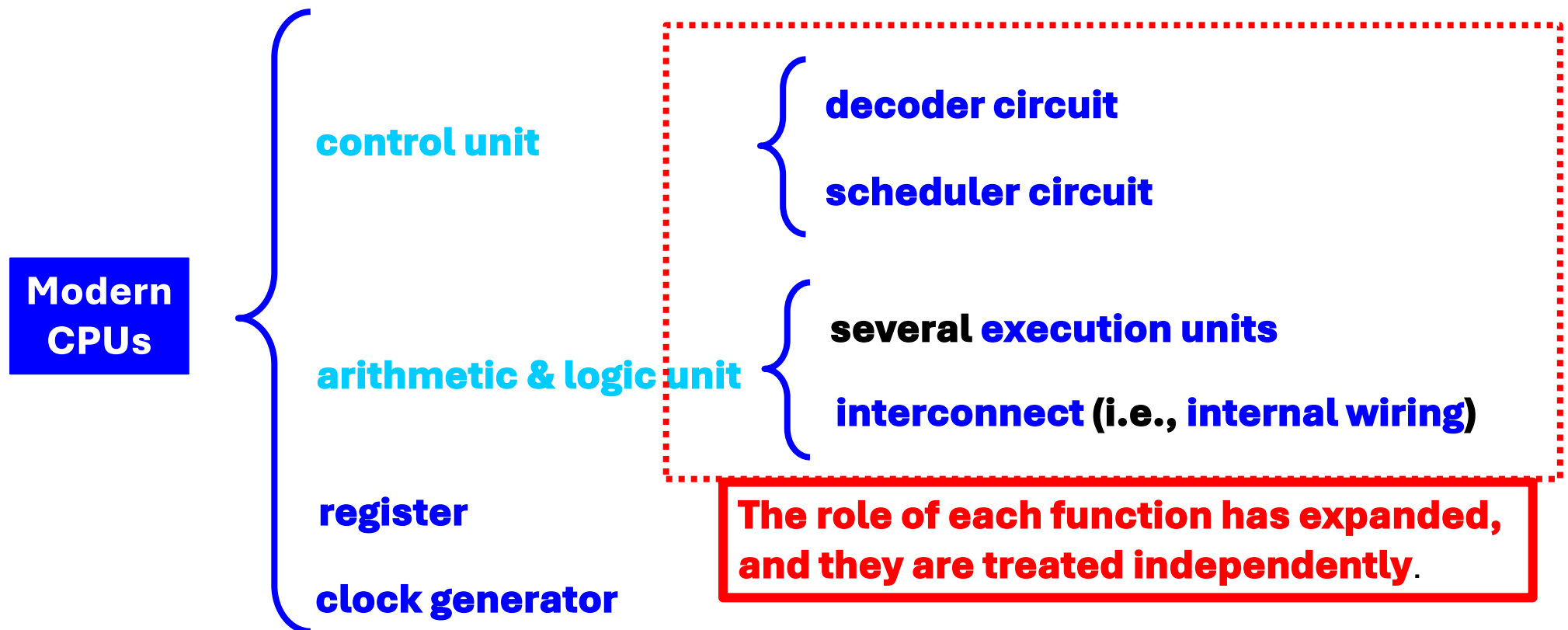
# CPU (Central Processing Unit)

- A CPU is basically a gate-level netlist etched and fixed as a concrete physical implementation.
- An example of the basic configuration of a CPU



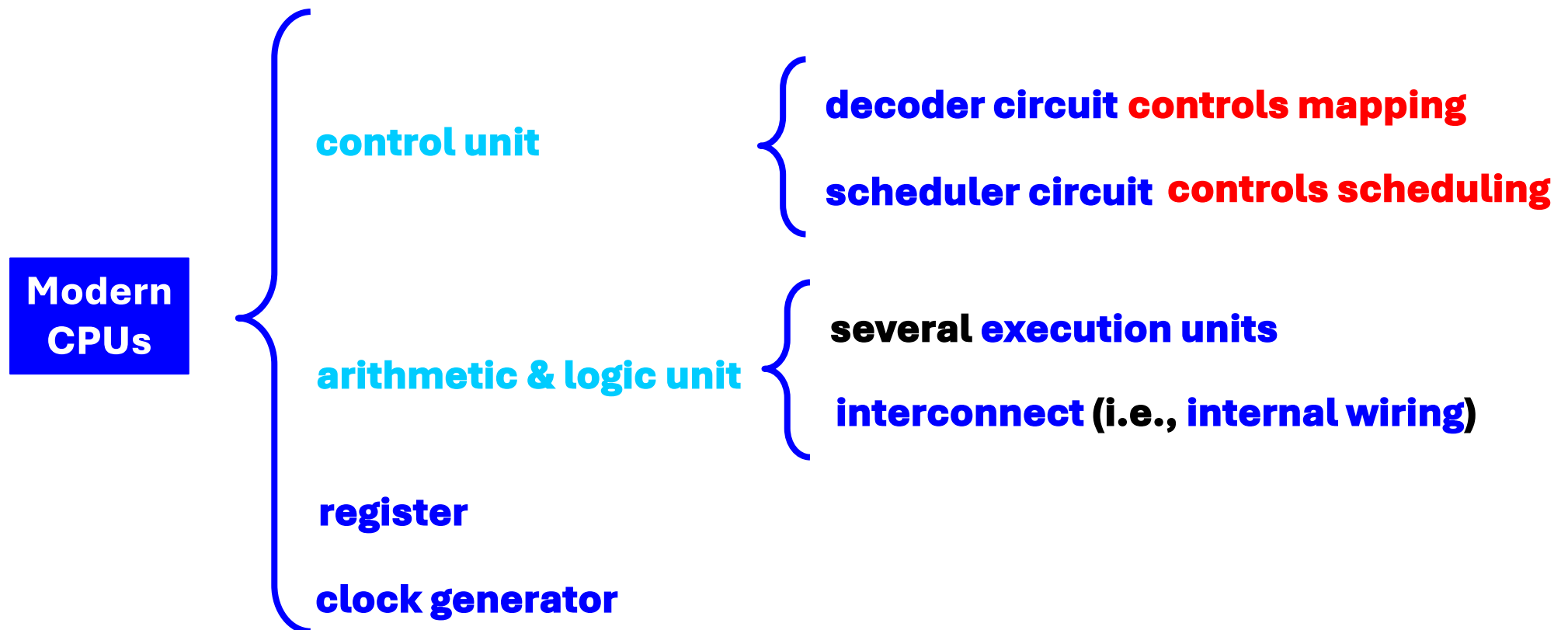
# CPU (Central Processing Unit)

- A CPU is basically a gate-level netlist etched and fixed as a concrete physical implementation.
- An example of the basic configuration of a CPU



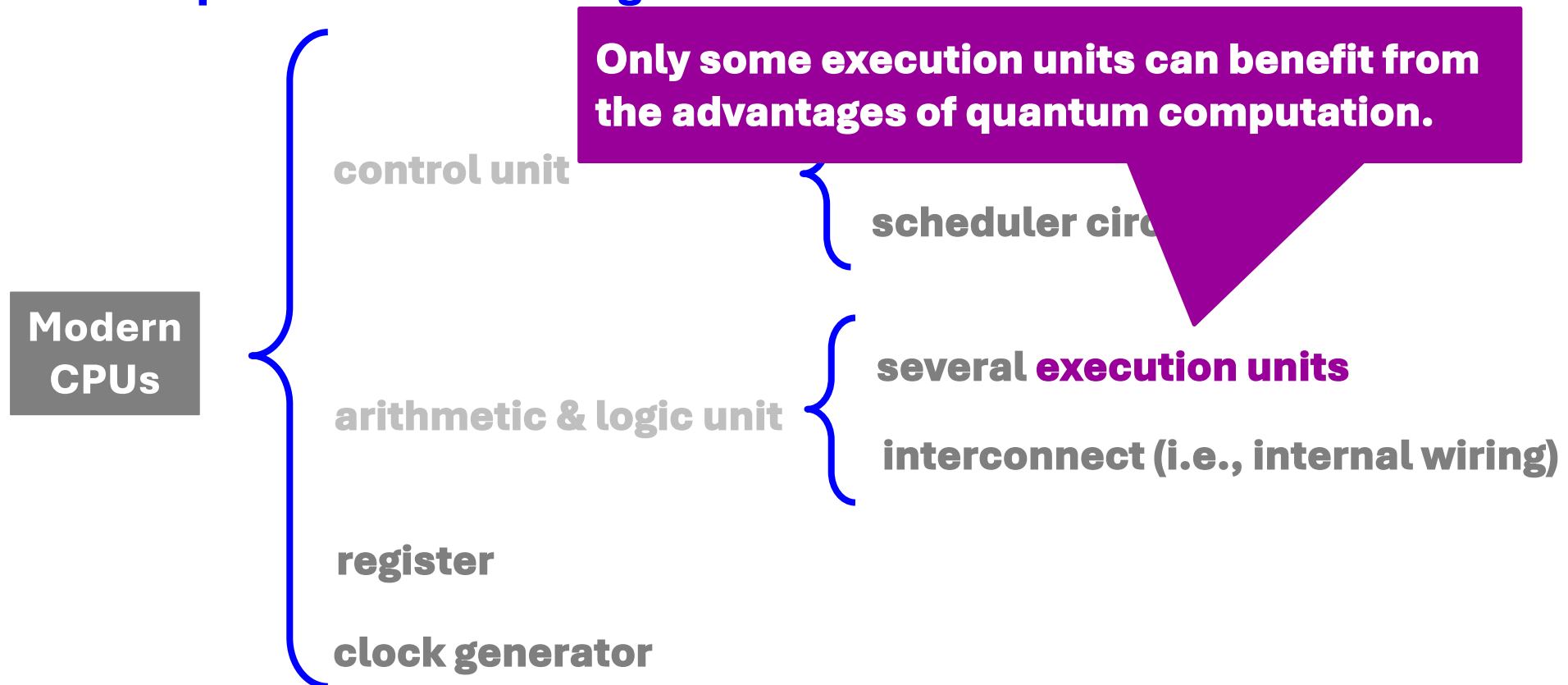
# CPU (Central Processing Unit)

- A CPU is basically a gate-level netlist etched and fixed as a concrete physical implementation.
- An example of the basic configuration of a CPU



# CPU (Central Processing Unit)

- A CPU is basically a gate-level netlist etched and fixed as a concrete physical implementation.
- An example of the basic configuration of a CPU



# What Is Calculation?

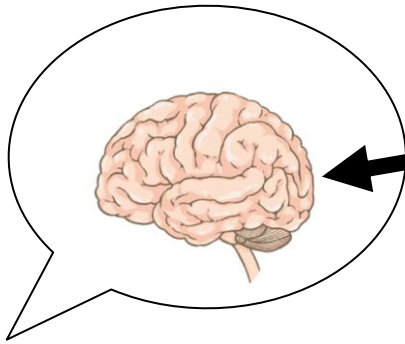
**When we teach a baby how to calculate,  
we are actually teaching the baby's brain.**

**We can rely on the 'computer' that is a baby's brain.**

**When teaching a machine how to compute,  
we cannot depend on its own intelligence  
because it has no brain or inherent mind.**

**We need to teach the machine how to compute from scratch.  
We need to understand what computation actually is.**

# What Is Calculation?



**When we teach a baby how to calculate,  
we are actually teaching the baby's brain.**

**We can rely on the 'computer' that is a baby's brain.**



**When teaching a machine how to compute,  
we cannot depend on its own intelligence  
because it has no brain or inherent mind.**

**We need to teach the machine how to compute from scratch.  
We need to understand “what computation is.”**

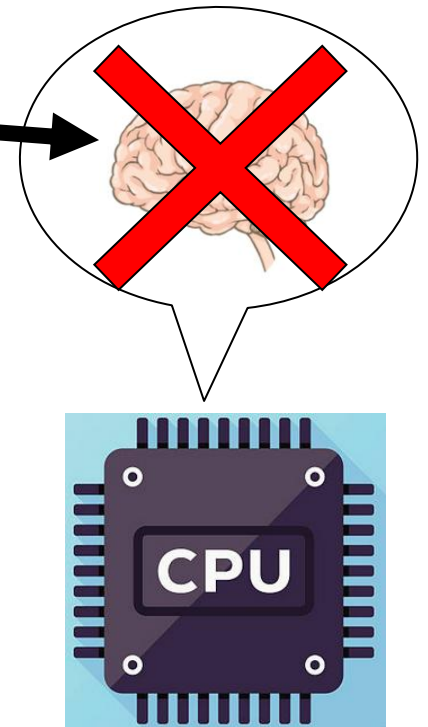
# What Is Calculation?

**When we teach a baby how to calculate,  
we actually teach the baby's brain  
how to do it.**

**We can rely on the 'computer' that is a baby's brain.**

**When teaching a machine how to compute,  
we cannot depend on its own intelligence  
because it has no brain or inherent mind.**

**We need to teach the machine how to compute from scratch.  
We need to understand “what computation is.”**



# What Is Calculation?



- **Classical computation refers to the process of transforming input into output using a Turing machine.**
- A Turing machine consists of an infinitely long tape, a tape head, and a state control unit; it is a mathematical model of computer hardware.
- Individual Turing machines compute specific partial recursive functions and therefore lack universal capability, where 'universal capability' means 'general-purpose capability.'
- However, there exists a universal Turing machine capable of reproducing the computation results of any specific Turing machine.
- The universal Turing machine is the theoretical foundation for general-purpose computers, which later inspired the von Neumann architecture.

**A universal Turing machine simulates any specific Turing machine.**

# What Is Calculation?



- Classical computation refers to the process of transforming input into output using a Turing machine.
- **A Turing machine consists of an infinitely long tape, a tape head, and a state control unit; it is a mathematical model of computer hardware.**
- Individual Turing machines compute specific partial recursive functions and therefore lack universal capability, where 'universal capability' means 'general-purpose capability.'
- However, there exists a universal Turing machine capable of reproducing the computation results of any specific Turing machine.
- The universal Turing machine is the theoretical foundation for general-purpose computers, which later inspired the von Neumann architecture.

**A universal Turing machine simulates any specific Turing machine.**

# What Is Calculation?



- Classical computation refers to the process of transforming input into output using a Turing machine.
- A Turing machine consists of an infinitely long tape, a tape head, and a state control unit; it is a mathematical model of computer hardware.
- **Individual Turing machines** compute **specific** partial recursive functions and therefore **lack universal capability**, where ‘**universal capability**’ means ‘**general-purpose capability**.’
- However, there exists a universal Turing machine capable of reproducing the computation results of any specific Turing machine.
- The universal Turing machine is the theoretical foundation for general-purpose computers, which later inspired the von Neumann architecture.

**A universal Turing machine simulates any specific Turing machine.**

# What Is Calculation?



- Classical computation refers to the process of transforming input into output using a Turing machine.
- A Turing machine consists of an infinitely long tape, a tape head, and a state control unit; it is a mathematical model of computer hardware.
- Individual Turing machines compute specific partial recursive functions and therefore lack universal capability, where 'universal capability' means 'general-purpose capability.'
- **However, there exists a universal Turing machine capable of reproducing the computation results of any specific Turing machine.**
- The universal Turing machine is the theoretical foundation for general-purpose computers, which later inspired the von Neumann architecture.

**A universal Turing machine simulates any specific Turing machine.**

# What Is Calculation?



- Classical computation refers to the process of transforming input into output using a Turing machine.
- A Turing machine consists of an infinitely long tape, a tape head, and a state control unit; it is a mathematical model of computer hardware.
- Individual Turing machines compute specific partial recursive functions and therefore lack universal capability, where 'universal capability' means 'general-purpose capability.'
- However, there exists a universal Turing machine capable of reproducing the computation results of any specific Turing machine.
- **The universal Turing machine is the theoretical foundation for general-purpose computers, which later inspired the von Neumann architecture.**

**A universal Turing machine simulates any specific Turing machine.**

# What Is Calculation?



- Classical computation refers to the process of transforming input into output using a Turing machine.
- A Turing machine consists of an infinitely long tape, a tape head, and a state control unit; it is a mathematical model of computer hardware.
- Individual Turing machines compute specific partial recursive functions and therefore lack universal capability, where 'universal capability' means 'general-purpose capability.'
- However, there exists a universal Turing machine capable of reproducing the computation results of any specific Turing machine.
- The universal Turing machine is the theoretical foundation for general-purpose computers, which later inspired the von Neumann architecture.

**A universal Turing machine simulates any specific Turing machine.**

# What Is Calculation?

Input → **Turing Machine** → Output

- Classical computation refers to the process of transforming input into output using a Turing machine.
- A Turing machine consists of an infinitely long tape, a tape head, and a state control unit; it is a mathematical model of computer hardware.
- Individual Turing machines compute specific partial recursive functions and therefore lack universal capability, where 'universal capability' means 'general-purpose capability.'
- However, there exists a universal Turing machine capable of reproducing the computation of any specific Turing machine.
- The universal Turing machine is a general-purpose computer.

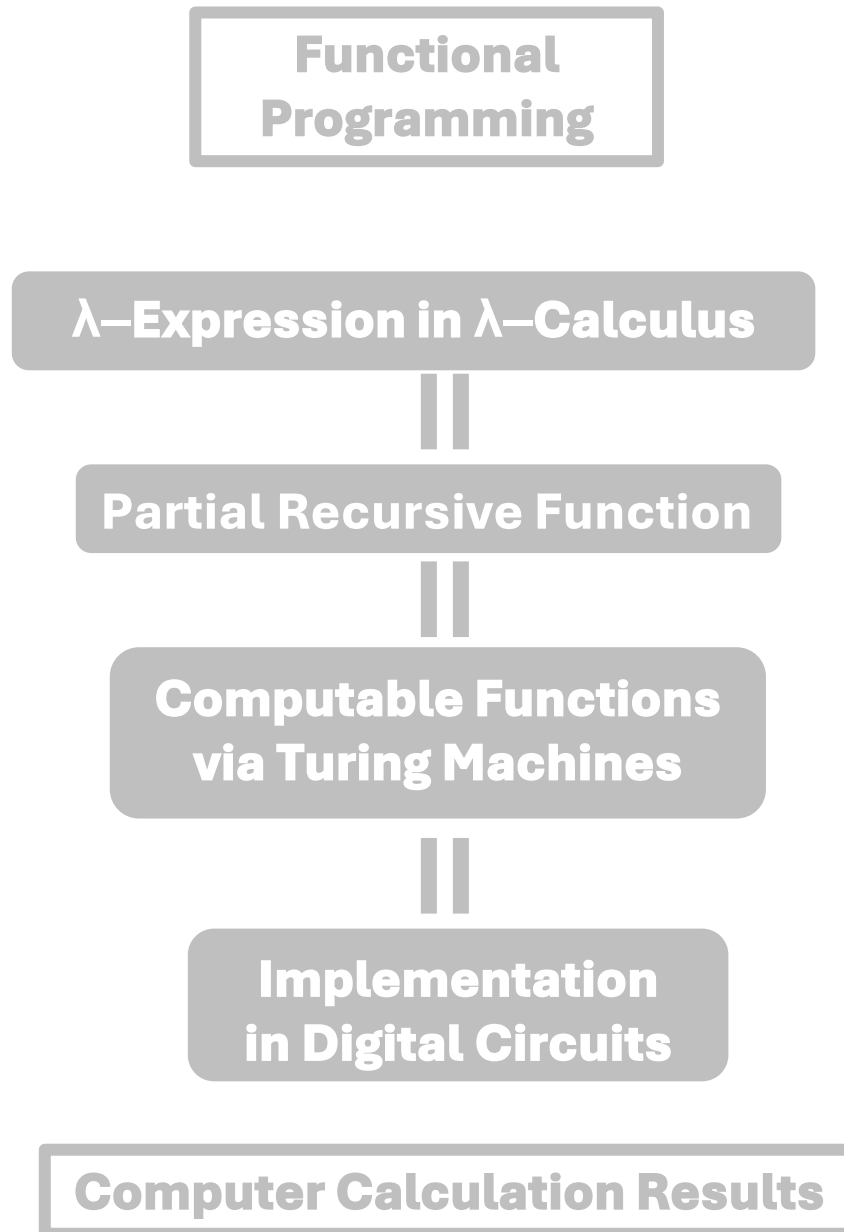
**There exists a universal quantum Turing machine that simulates each quantum Turing machine.**

**A universal Turing machine simulates any specific Turing machine.**

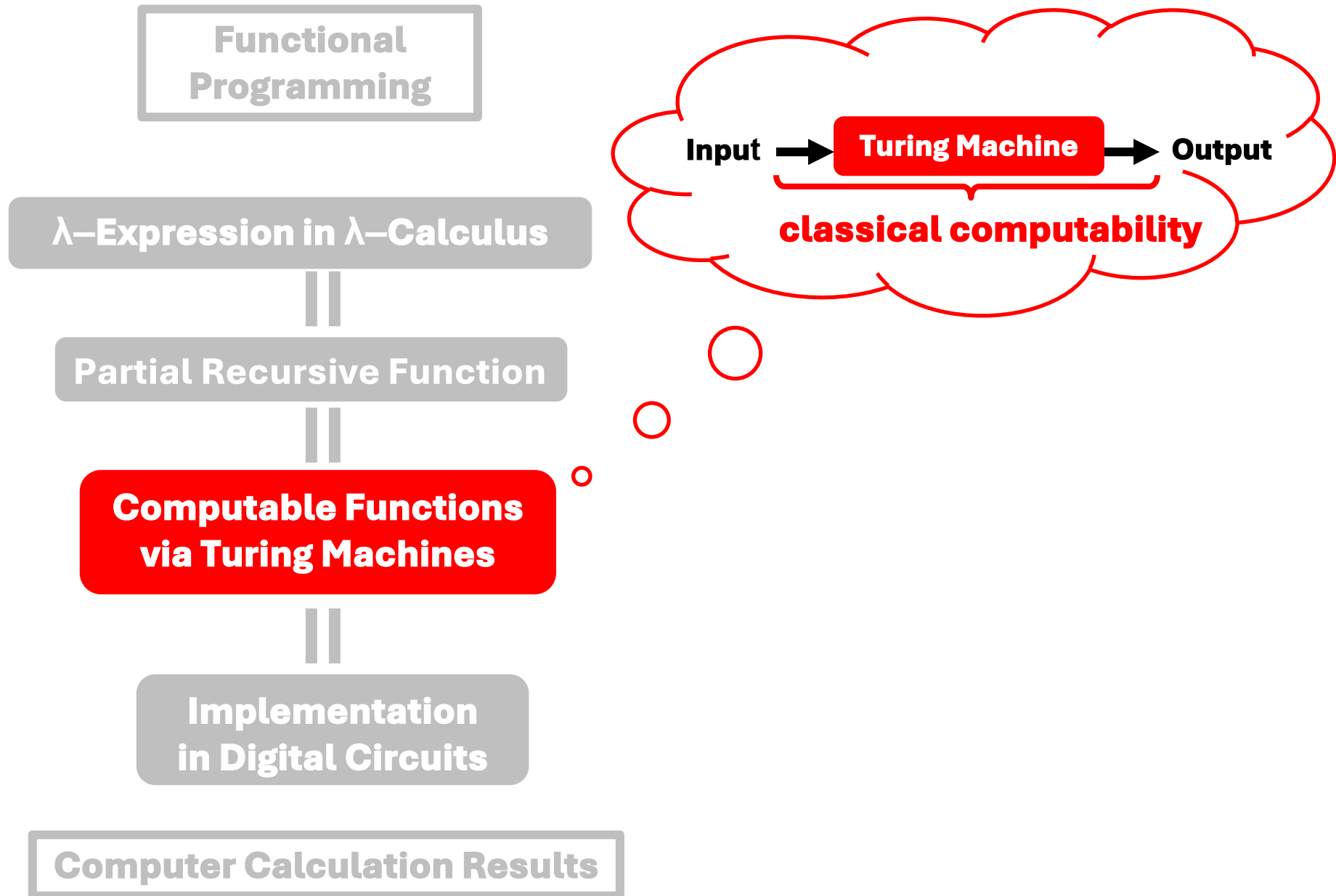
Input → **Quantum Turing Machine** → Output

**Quantum Computation**

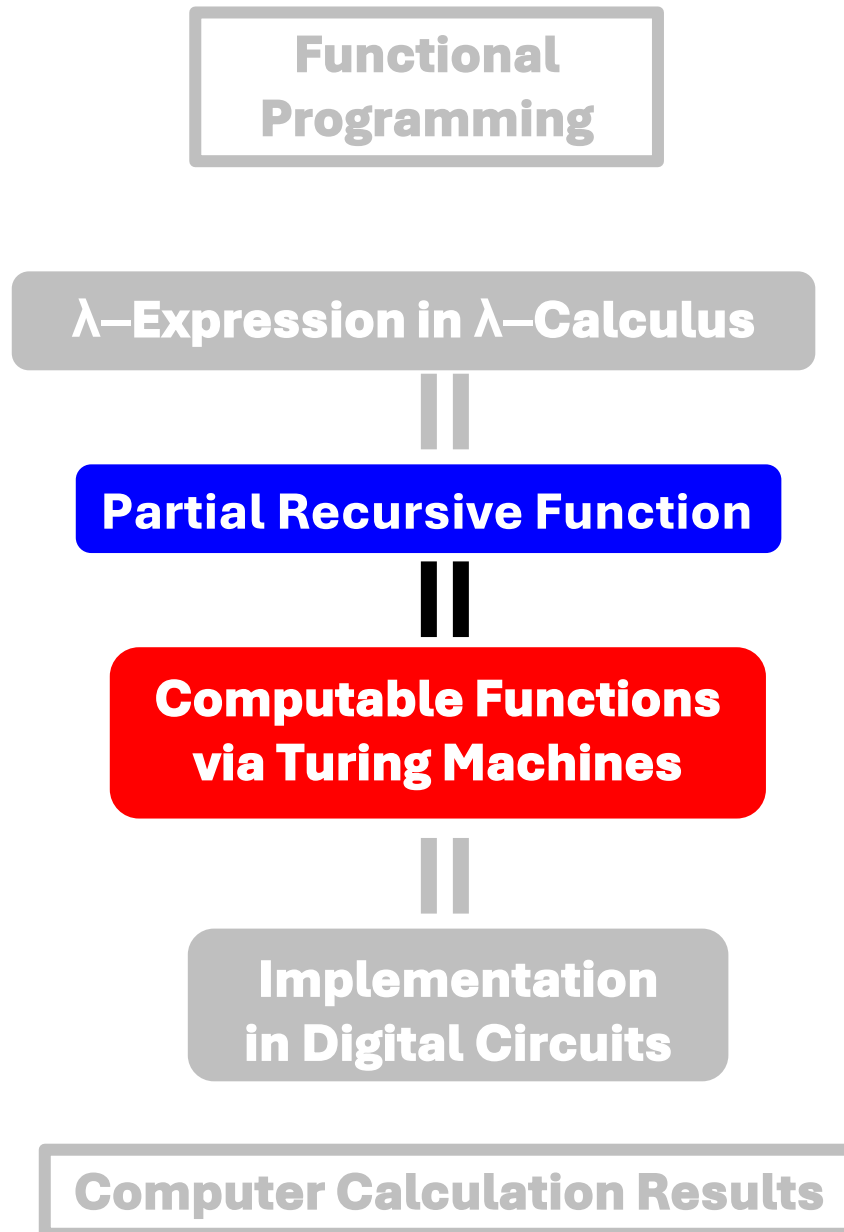
# Separation of Software & Hardware



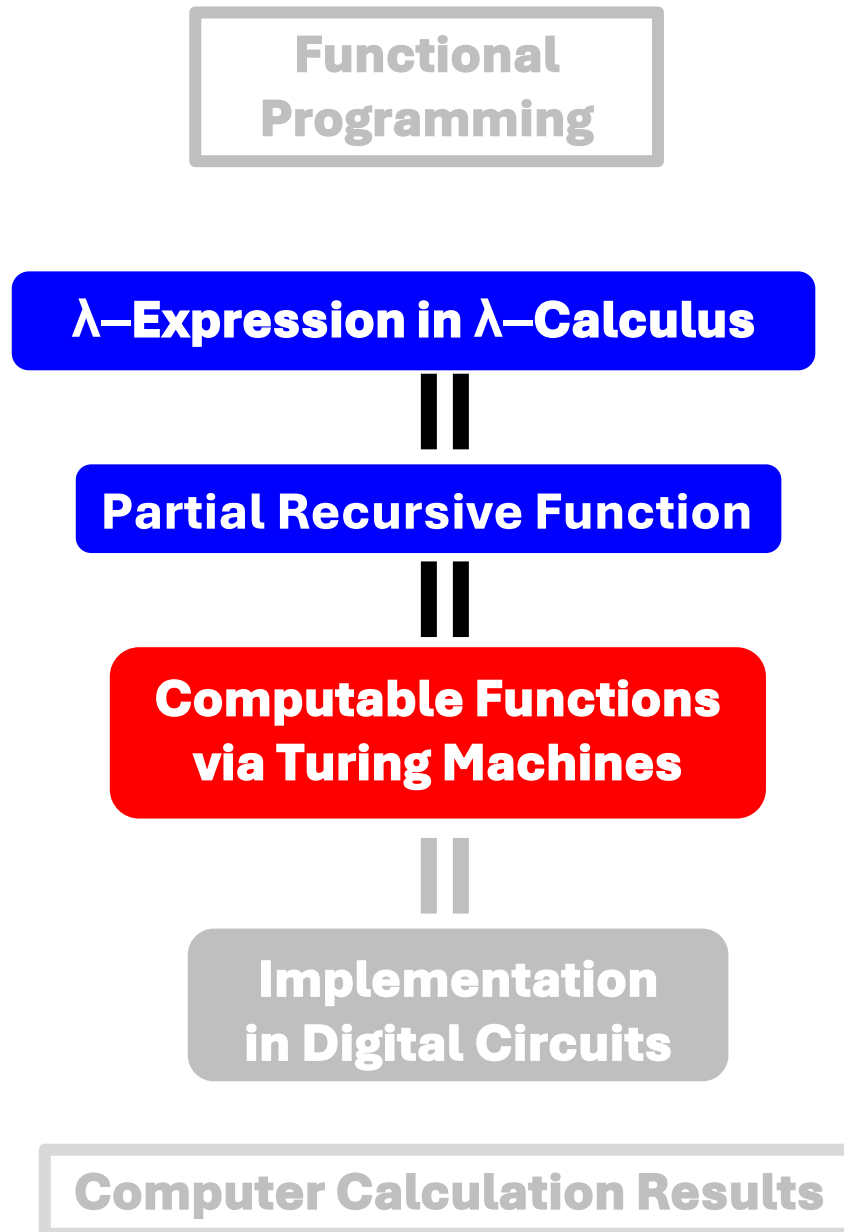
# Separation of Software & Hardware



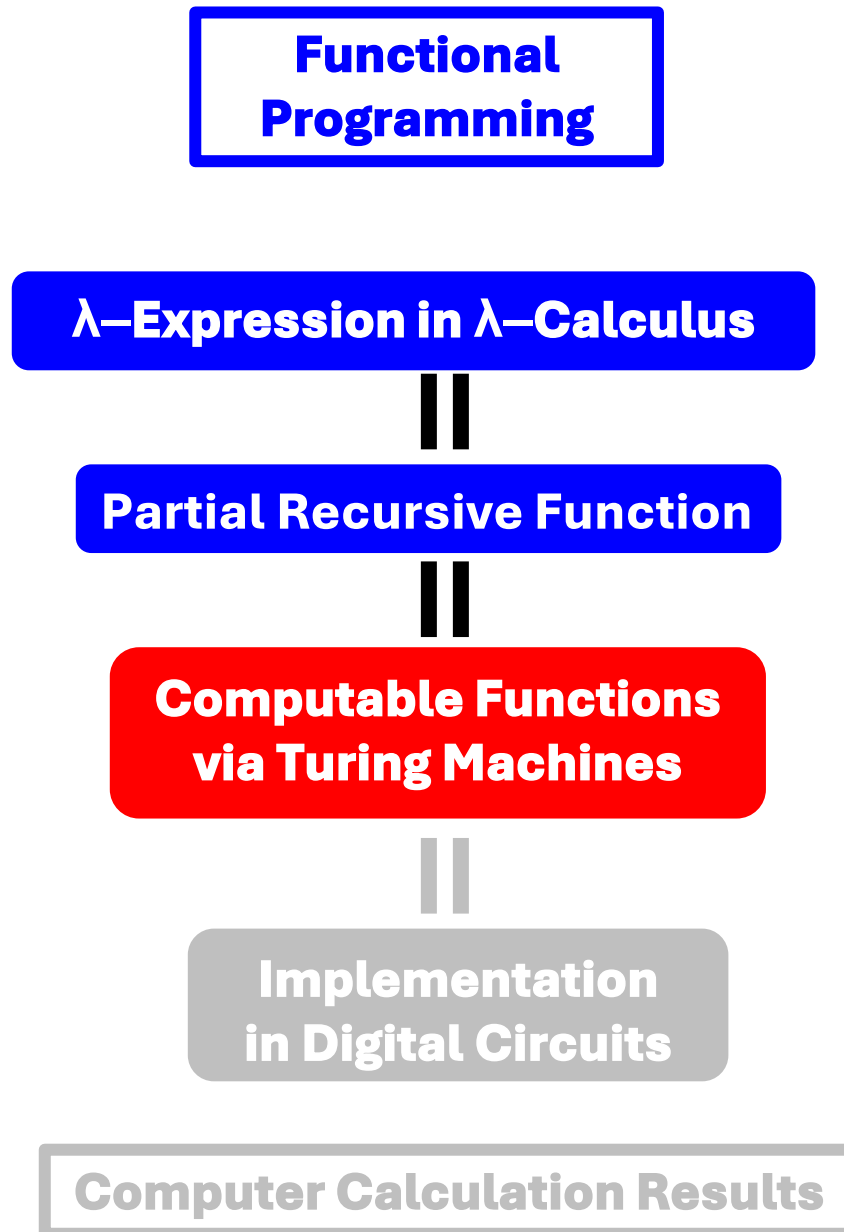
# Separation of Software & Hardware



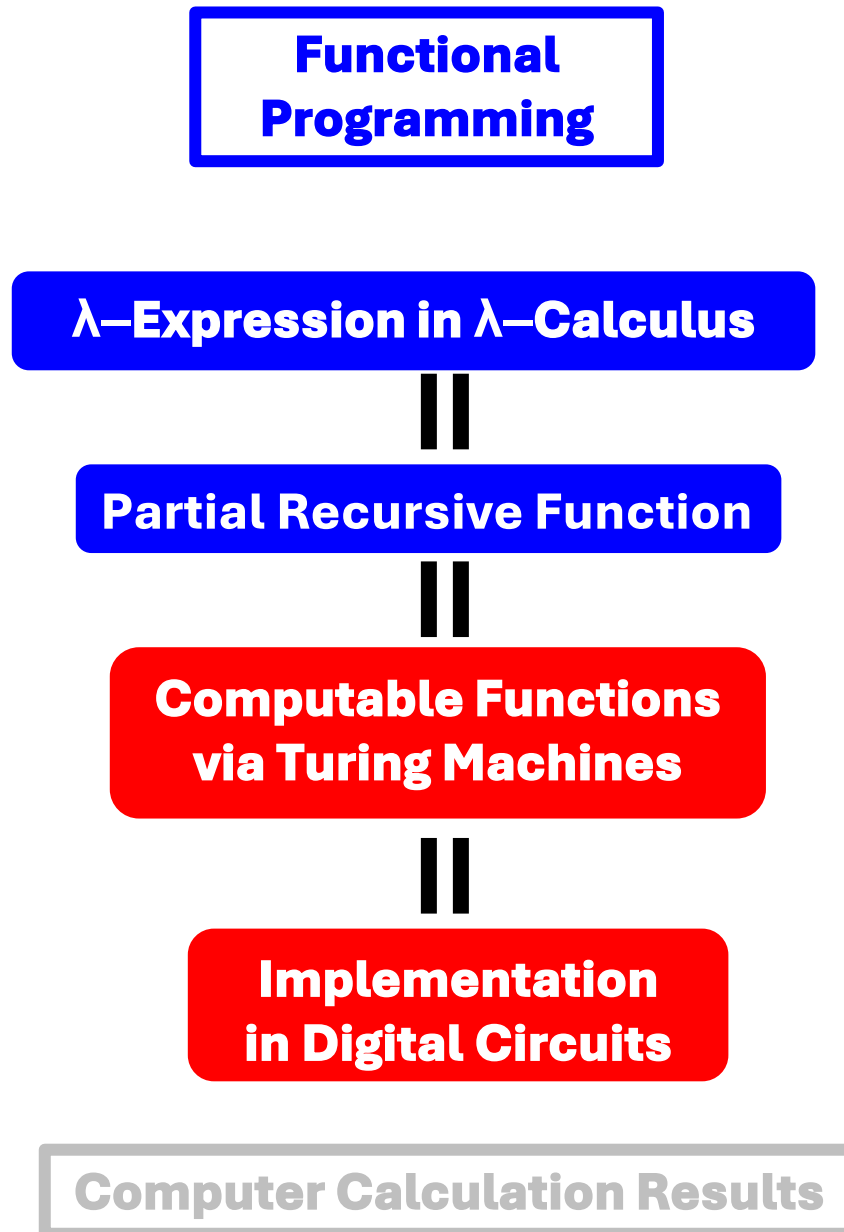
# Separation of Software & Hardware



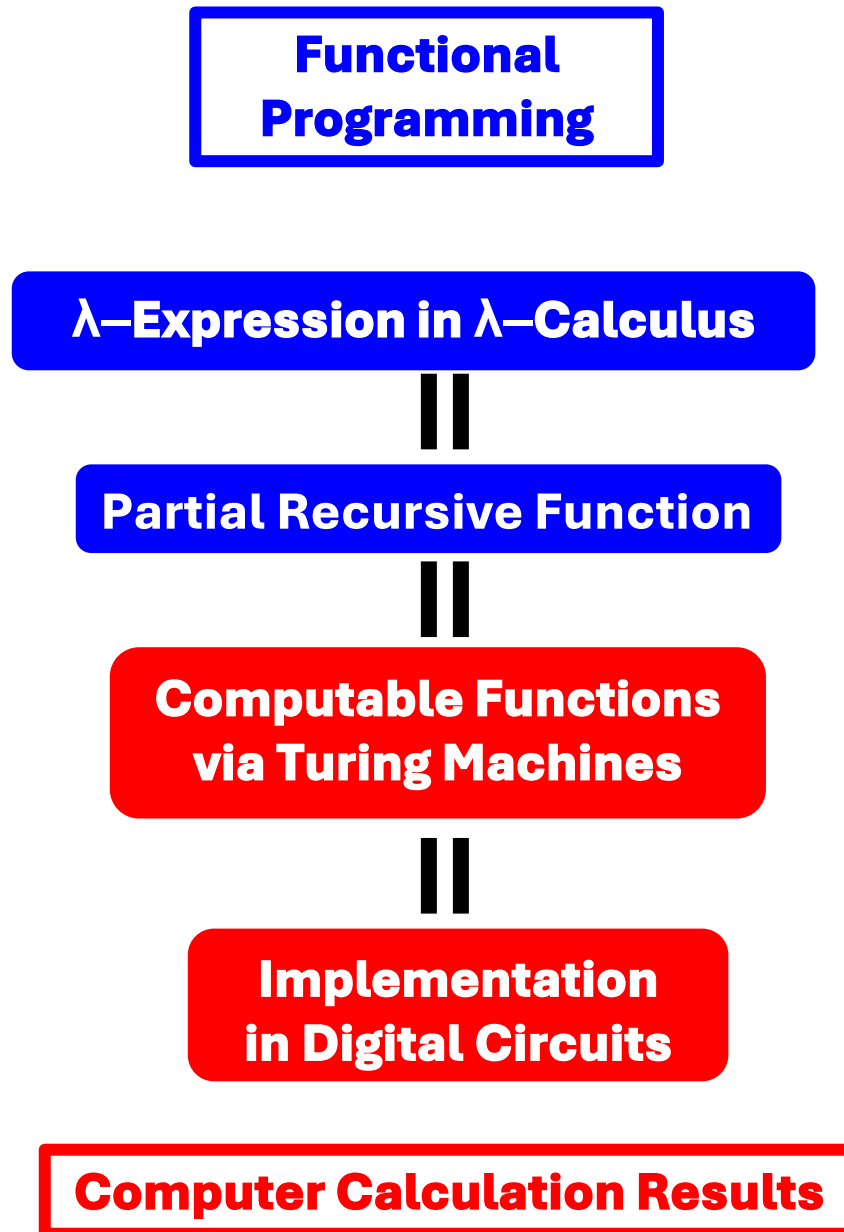
# Separation of Software & Hardware



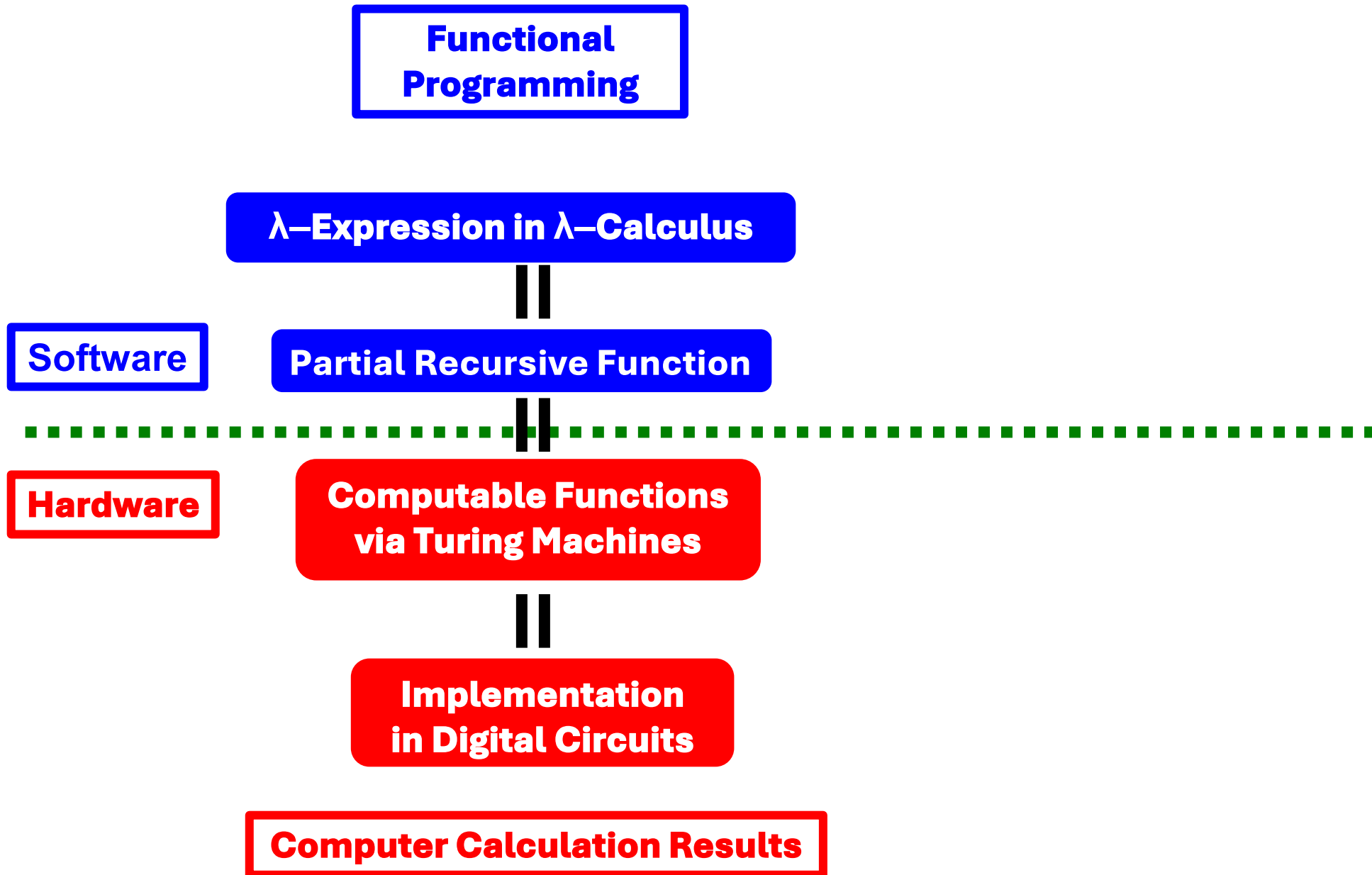
# Separation of Software & Hardware



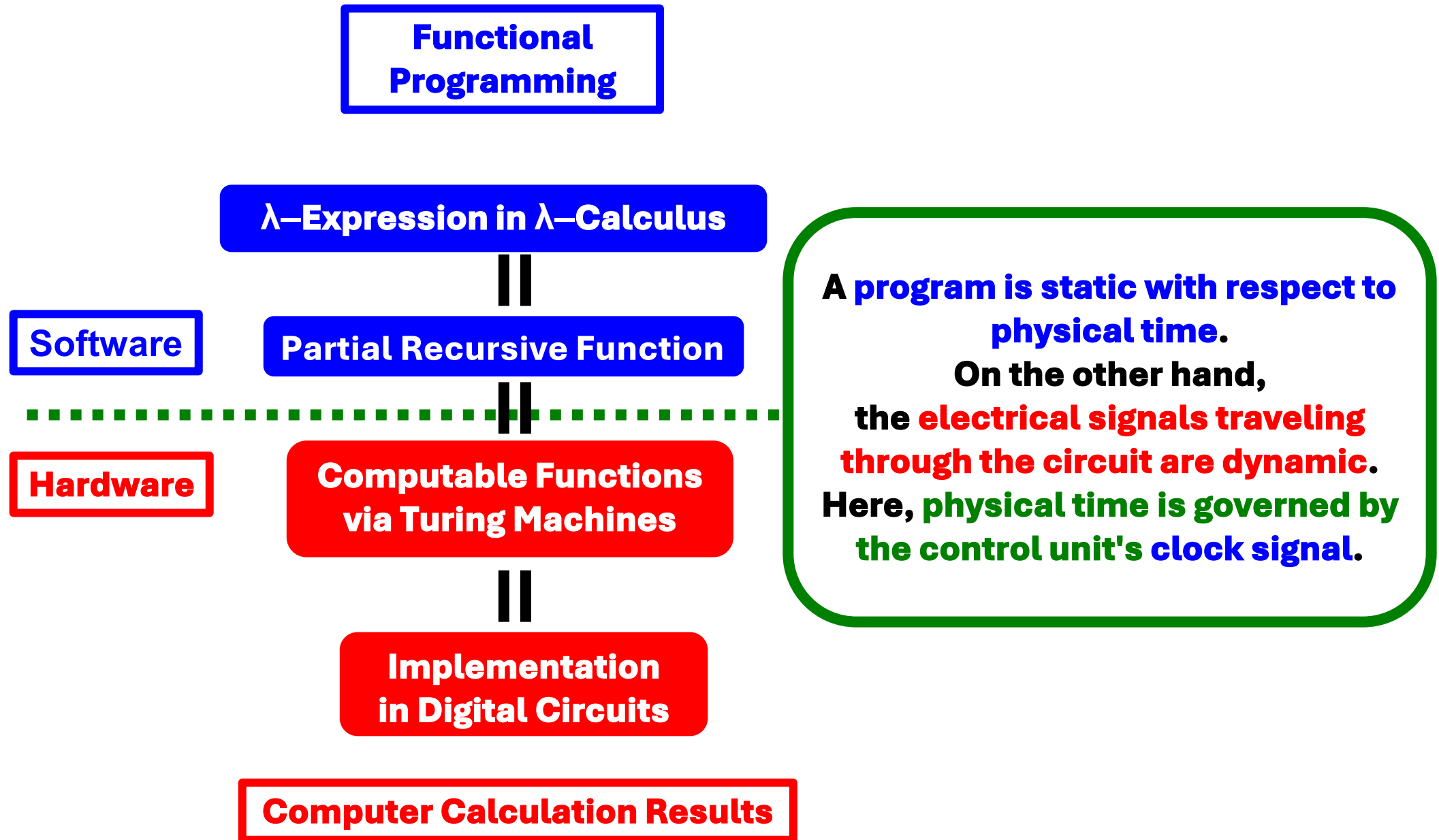
# Separation of Software & Hardware



# Separation of Software & Hardware



# Separation of Software & Hardware



# Separation of Software & Hardware

**Functional  
Programming**

$\lambda$ -Expression in  $\lambda$ -Calculus

**Software**

Partial Recursive Function

**Hardware**

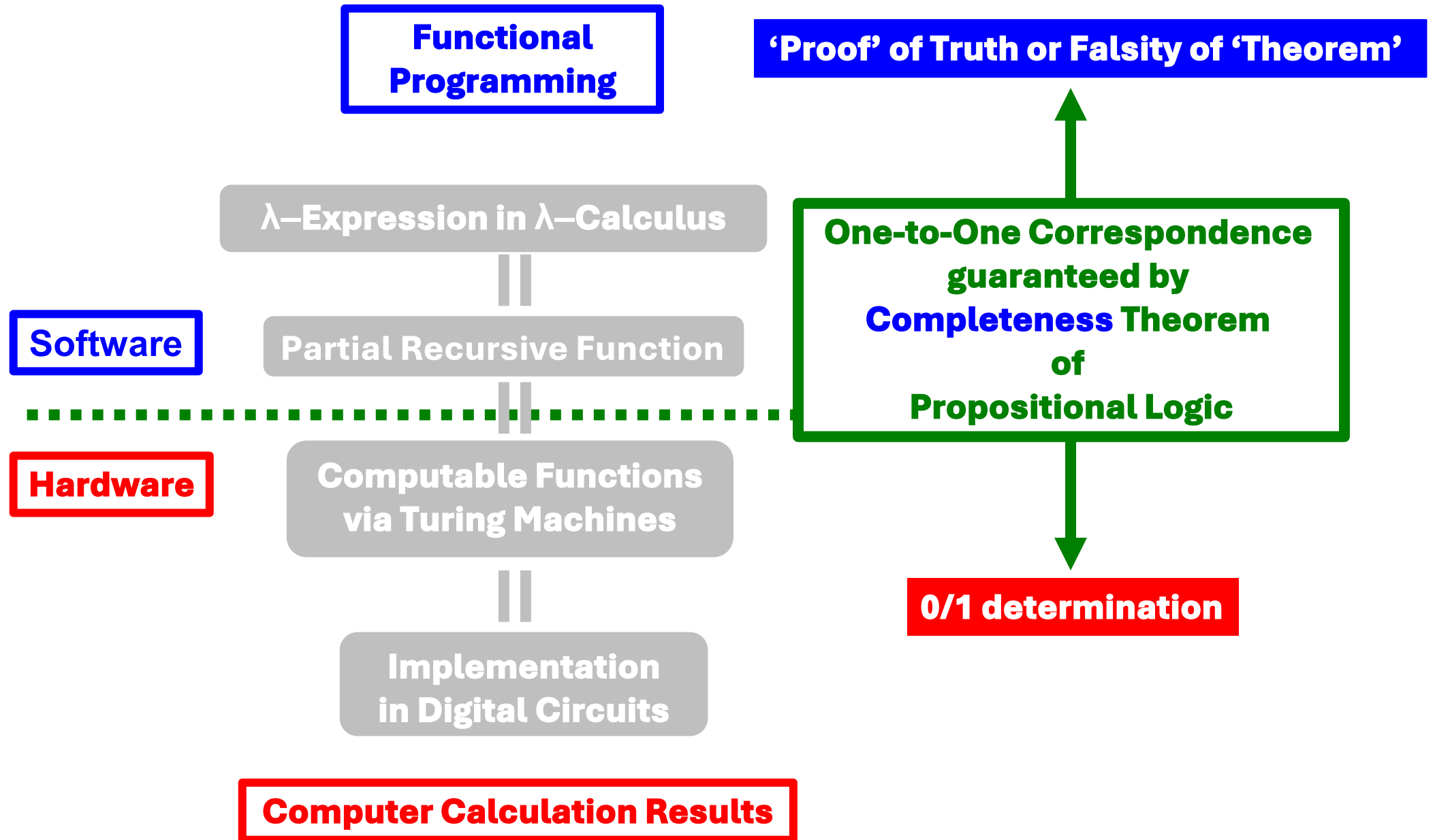
Computable Functions  
via Turing Machines

**Why do you trust the CPU's 0/1 determination?**

Implementation  
in Digital Circuits

**Computer Calculation Results**

# Separation of Software & Hardware



# Separation of Software & Hardware

Functional

What does  
the completeness theorem  
in quantum logic state?

'Proof' of Truth or Falsity of 'Theorem'

One-to-One Correspondence  
guaranteed by  
Completeness Theorem  
of  
Propositional Logic

- Whether the completeness theorem holds in quantum logic depends on the conditions. It is a very complicated and delicate issue.

0/1 determination

Implementation  
in Digital Circuits

Computer Calculation Results

# Separation of Software & Hardware

Functional

What does  
the completeness theorem  
in quantum logic state?

- Regarding quantum Hoare logic, **Relative Completeness** has been established (M. Ying), as in the classical case (S. Cook).
- Relative completeness relates to **partial correctness** and **total correctness** concerning **Halting Problem** (i.e., the loop termination problem).
- The **Loop Invariant** is important for both, while the **Loop Variant** is crucial for total correctness.

'Proof' of Truth or Falsity of 'Theorem'

One-to-One Correspondence  
guaranteed by  
**Completeness Theorem**  
of  
**Propositional Logic**

0/1 determination

# Separation of Software & Hardware

What does  
the completeness theorem  
in quantum logic state?

- Regarding quantum Hoare logic, **Relative Completeness** has been established (M. Ying), as in the classical case (S. Cook).
- Relative completeness relates to **partial correctness** and **total correctness** concerning **Halting Problem** (i.e., the loop termination problem).

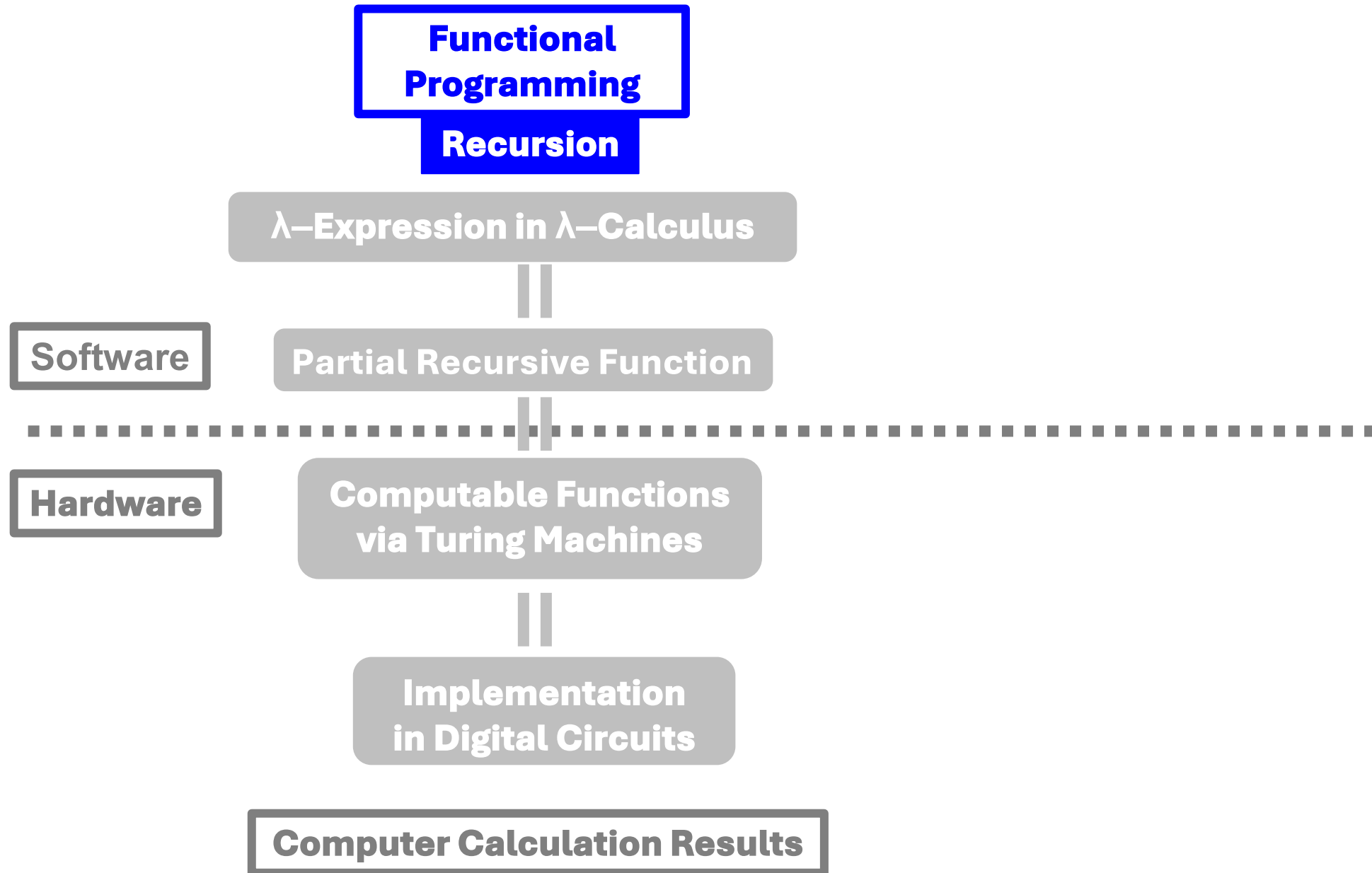
The **Loop Invariant** is important for both, while the **Loop Variant** is crucial for total correctness.

**One-to-One Correspondence**  
guaranteed by  
**Completeness Theorem**  
of  
**Propositional Logic**

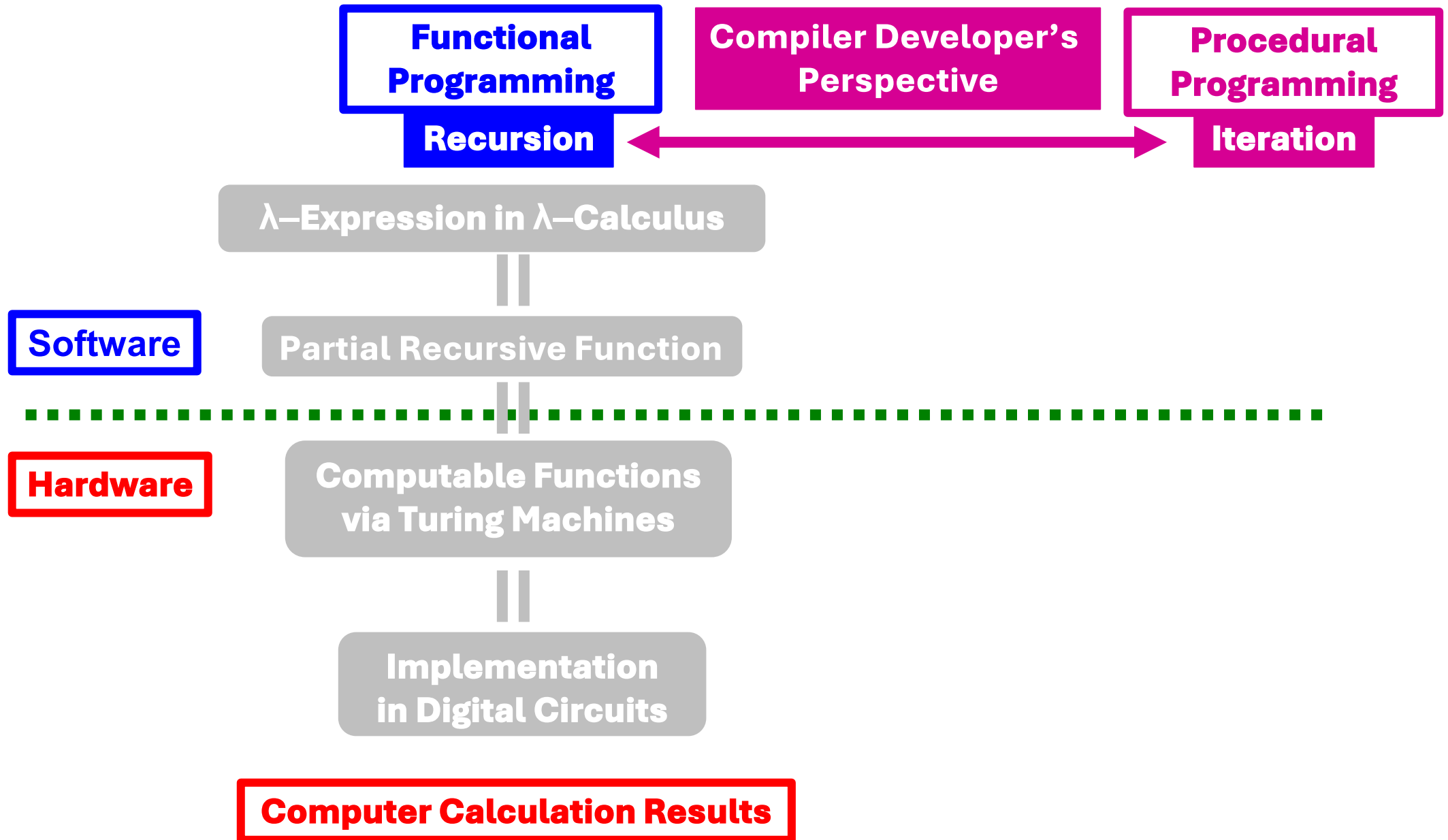
**Quantum Hoare Logic**

**Quantum Super-Invariant**  
+  
**Quantum Martingale**  
(M. Ying)

# Separation of Software & Hardware



# Separation of Software & Hardware



# Separation of Software & Hardware

## Recursion

$$\begin{cases} f(0) = 0, \\ f(n) = f(n-1) + 2, \quad n = 1, 2, \dots, \end{cases}$$

hirokawa@LAPTOP-9VQV46HK: ~/Programming/C/Q-LEAP

File Edit Options Buffers Tools C Help

```
#include <stdio.h>
#include <time.h>

int func1(int X) {
    if(X==0) {
        return 0;
    } else {
        return func1(X-1)+2;
    }
}

int main()
{
    int X;
    printf("n=");
    scanf("%d", &X);
    int Y=func1(X);
    printf("f(%d) = %d\n", X, Y);
    return 0;
}
```

# Separation of Software & Hardware

## Recursion

```
hirokawa@LAPTOP-9VQV46HK: ~/Programming/C/Q-LEAP
hirokawa@LAPTOP-9VQV46HK:~/Programming/C/Q-LEAP$ ./f_rec_time.out
n=100000
f(100000) = 200000
CPU Time = 0.002243 秒
hirokawa@LAPTOP-9VQV46HK:~/Programming/C/Q-LEAP$ ./f_rec_time.out
n=1000000
Segmentation fault
hirokawa@LAPTOP-9VQV46HK:~/Programming/C/Q-LEAP$
```

**We have a trouble at  $n = 10^6$ .**

# Separation of Software & Hardware

## Recursion

```
00000000000011b7 <main>:
11b7: f3 0f 1e fa      endbr64
11bb: 55              push    rbp
11bc: 48 89 e5        mov     rbp, rsp
11bf: 48 83 ec 10     sub     rsp, 0x10
11c3: 64 48 8b 04 25 28 00 mov     rax, QWORD PTR fs:0x28
11ca: 00 00
11cc: 48 89 45 f8     mov     QWORD PTR [rbp-0x8], rax
11d0: 31 c0          xor     eax, eax
11d2: 48 8d 3d 2b 0e 00 00 lea     rdi, [rip+0xe2b] # 2004 <_IO_stdin_used+0x4>
11d9: b8 00 00 00 00 mov     eax, 0x0
11de: e8 9d fe ff ff   call    1080 <printf@plt>
11e3: 48 8d 45 f0     lea     rax, [rbp-0x10]
11e7: 48 89 c6        mov     rsi, rax
11ea: 48 8d 3d 16 0e 00 00 lea     rdi, [rip+0xe16] # 2007 <_IO_stdin_used+0x7>
11f1: b8 00 00 00 00 mov     eax, 0x0
11f6: e8 95 fe ff ff   call    1090 <__isoc99_scanf@plt>
11fb: 8b 45 f0        mov     eax, DWORD PTR [rbp-0x10]
11fe: 89 c7          mov     edi, eax
1200: e8 84 ff ff ff   call    1189 <func1>
1205: 89 45 f4        mov     DWORD PTR [rbp-0xc], eax
1208: 8b 45 f0        mov     eax, DWORD PTR [rbp-0x10]
120b: 8b 55 f4        mov     edx, DWORD PTR [rbp-0xc]
120e: 89 c6          mov     esi, eax
```

```
0000000000001189 <func1>:
1189: f3 0f 1e fa      endbr64
118d: 55              push    rbp
118e: 48 89 e5        mov     rbp, rsp
1191: 48 83 ec 10     sub     rsp, 0x10
1195: 89 7d fc        mov     DWORD PTR [rbp-0x4], edi
1198: 83 7d fc 00     cmp     DWORD PTR [rbp-0x4], 0x0
119c: 75 07          jne     11a5 <func1+0x1c>
119e: b8 00 00 00 00 mov     eax, 0x0
11a3: eb 10          jmp     11b5 <func1+0x2c>
11a5: 8b 45 fc        mov     eax, DWORD PTR [rbp-0x4]
11a8: 83 e8 01        sub     eax, 0x1
11ab: 89 c7          mov     edi, eax
11ad: e8 d7 ff ff ff   call    1189 <func1>
11b2: 83 c0 02        add     eax, 0x2
11b5: c9             leave
11b6: c3             ret
```

# Separation of Software & Hardware

## Recursion

### Call-by-Procedure

$$f(n) = f(n-1) + 2$$



$$f(n-1) = f(n-2) + 2$$



⋮



$$f(2) = f(1) + 2$$



$$f(1) = f(0) + 2$$

**push data onto stack**

### Call-by-Value

$$f(n) = 2(n-1) + 2 = 2n$$



$$f(n-1) = 2(n-2) + 2 = 2(n-1)$$



⋮



$$f(2) = 2 + 2 = 4$$



$$f(1) = 0 + 2 = 2$$



$$f(0) = 0$$

**values**

**pop data from stack**

# Separation of Software & Hardware

## Iteration

Using the optimization option of gcc,

```
hirokawa@LAPTOP-9VQV46HK: ~/Programming/C/Q-LEAP
hirokawa@LAPTOP-9VQV46HK:~/Programming/C/Q-LEAP$ ./f_rec_time_opt.out
n=100000
f(100000) = 200000
CPU Time = 0.000617 秒
hirokawa@LAPTOP-9VQV46HK:~/Programming/C/Q-LEAP$ ./f_rec_time_opt.out
n=1000000
f(1000000) = 2000000
CPU Time = 0.000705 秒
hirokawa@LAPTOP-9VQV46HK:~/Programming/C/Q-LEAP$ ./f_rec_time_opt.out
n=10000000
f(10000000) = 20000000
CPU Time = 0.000655 秒
hirokawa@LAPTOP-9VQV46HK:~/Programming/C/Q-LEAP$ ./f_rec_time_opt.out
n=100000000
f(100000000) = 200000000
CPU Time = 0.000616 秒
hirokawa@LAPTOP-9VQV46HK:~/Programming/C/Q-LEAP$ ./f_rec_time_opt.out
n=1000000000
f(1000000000) = 2000000000
CPU Time = 0.000633 秒
hirokawa@LAPTOP-9VQV46HK:~/Programming/C/Q-LEAP$ _
```

# Separation of Software & Hardware

## Iteration

**In fact, recursion is being replaced by iteration.**

```
00000000000010a0 <main>:  
    return func1(X-1)+2;  
    }  
}  
  
int main()  
{  
    10a0:      f3 0f 1e fa      endbr64  
    10a4:      48 83 ec 18      sub     rsp, 0x18  
}
```

# Separation of Software & Hardware

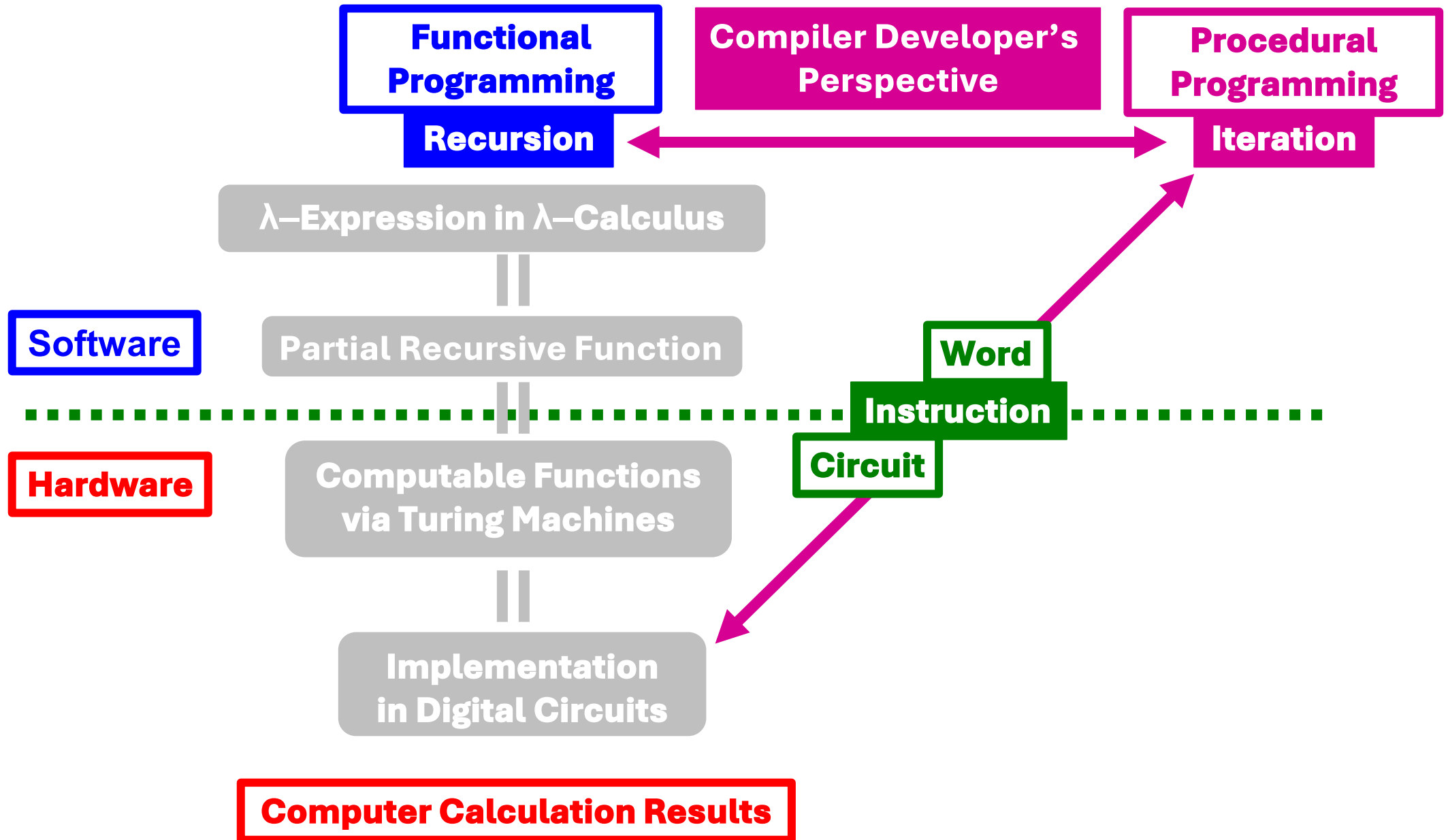
## Iteration

**In fact, recursion is being replaced by iteration.**

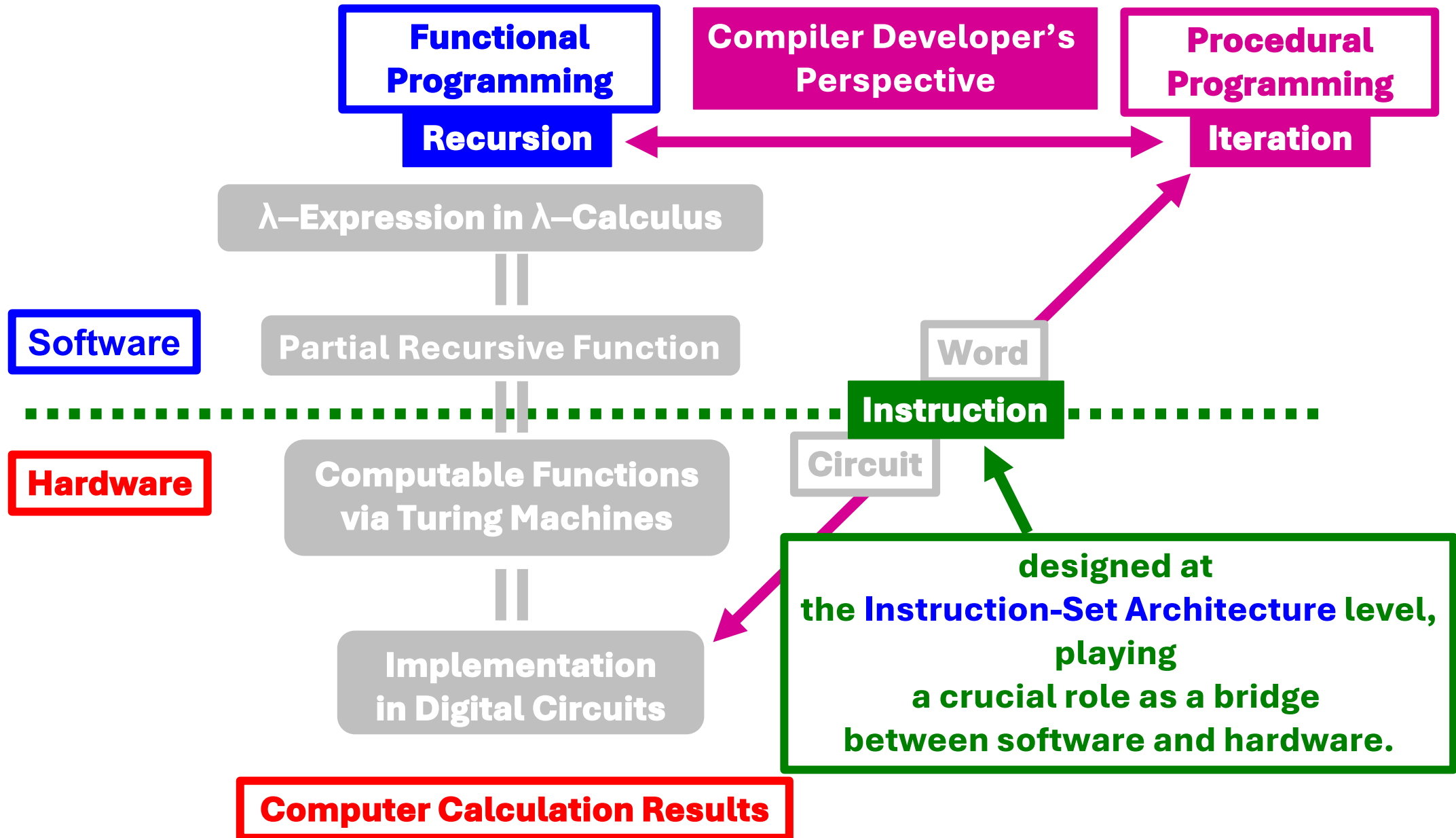
```
00000000000010a0 <main>:  
    return func1(X-1)+2;  
    }  
}  
  
int main()  
{  
    10a0:      f3 0f 1e fa      endbr64  
    10a4:      48 83 ec 18      sub     rsp, 0x18  
}
```

**The compiler can perform this optimization automatically.**

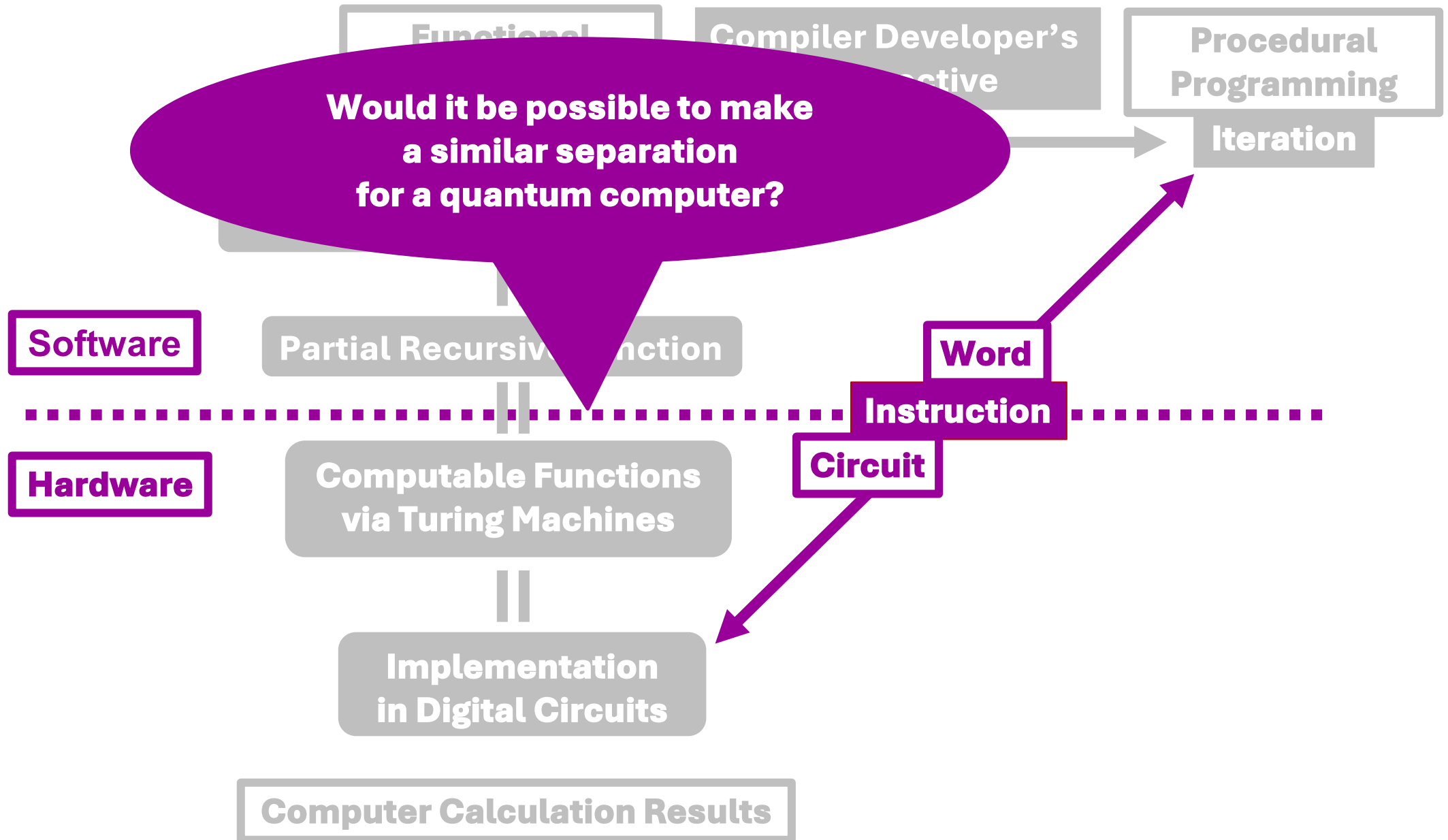
# Separation of Software & Hardware



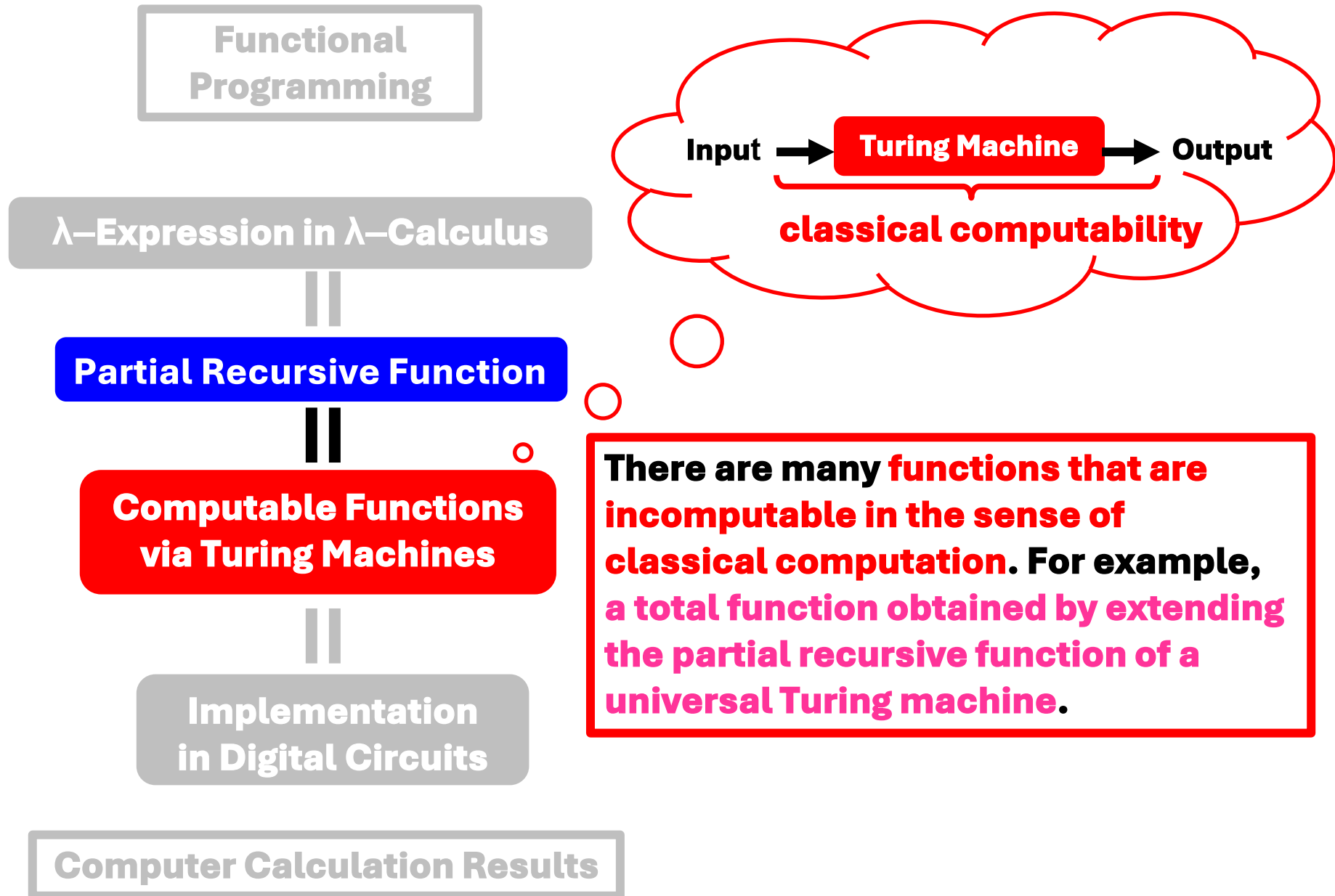
# Separation of Software & Hardware



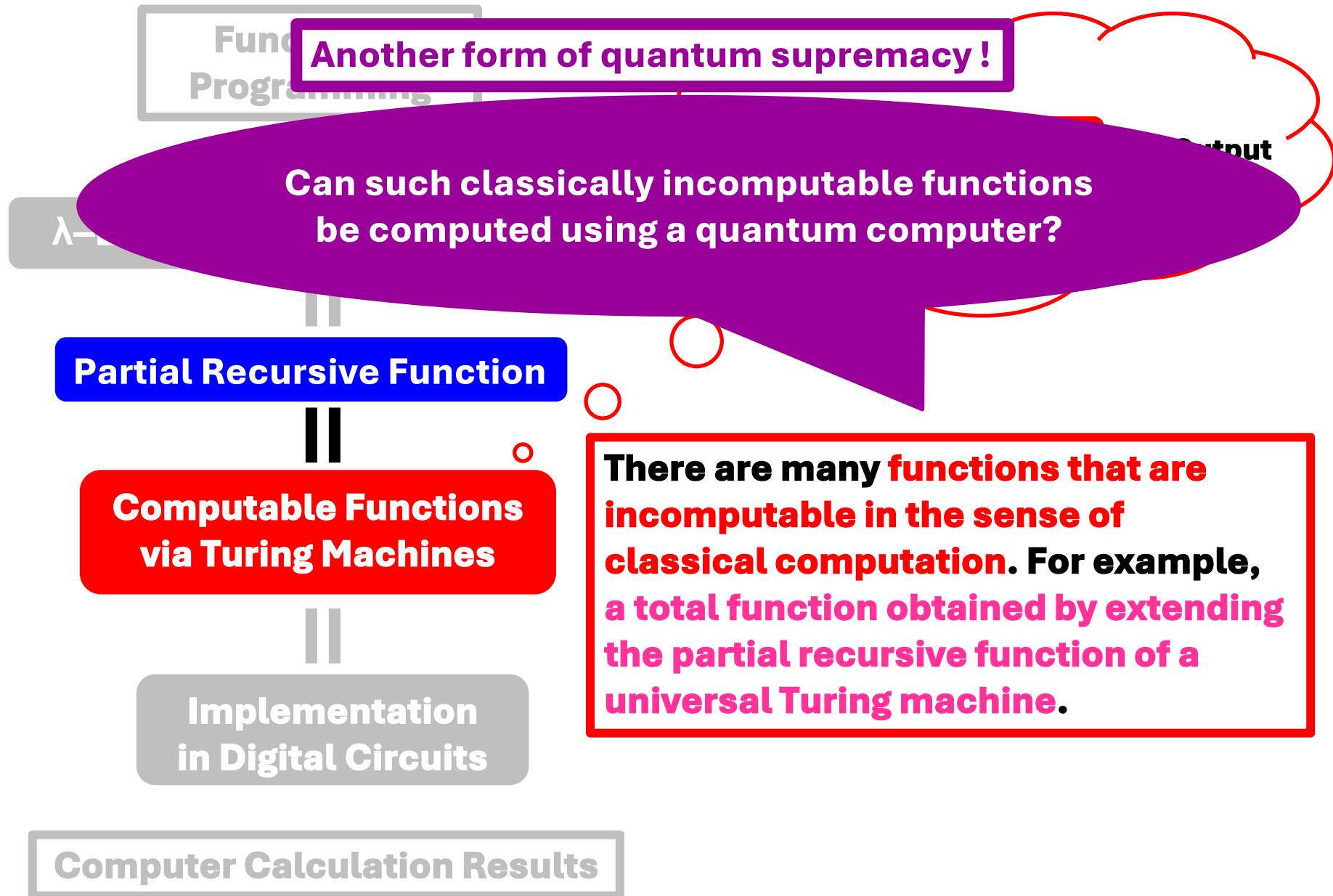
# Separation of Software & Hardware



# Separation of Software & Hardware



# Separation of Software & Hardware



# Separation of Software & Hardware

Another form of quantum supremacy !

Can such classically incomputable functions  
be computed using a quantum computer?

In the context of Turing machines,  
functions that are not computable  
by total recursive functions arise;  
for quantum Turing machines, the  
issue of observability within the  
framework of quantum measurement  
comes into play.

There are many **functions that are  
incomputable in the sense of  
classical computation**. For example,  
a total function obtained by extending  
the partial recursive function of a  
universal Turing machine.

In Digital Circuits

Computer Calculation Results

# Separation of Software & Hardware

In fact,

- **classical computability and quantum computability are equivalent; therefore, functions that are not classically computable are also not computable via quantum computation.**

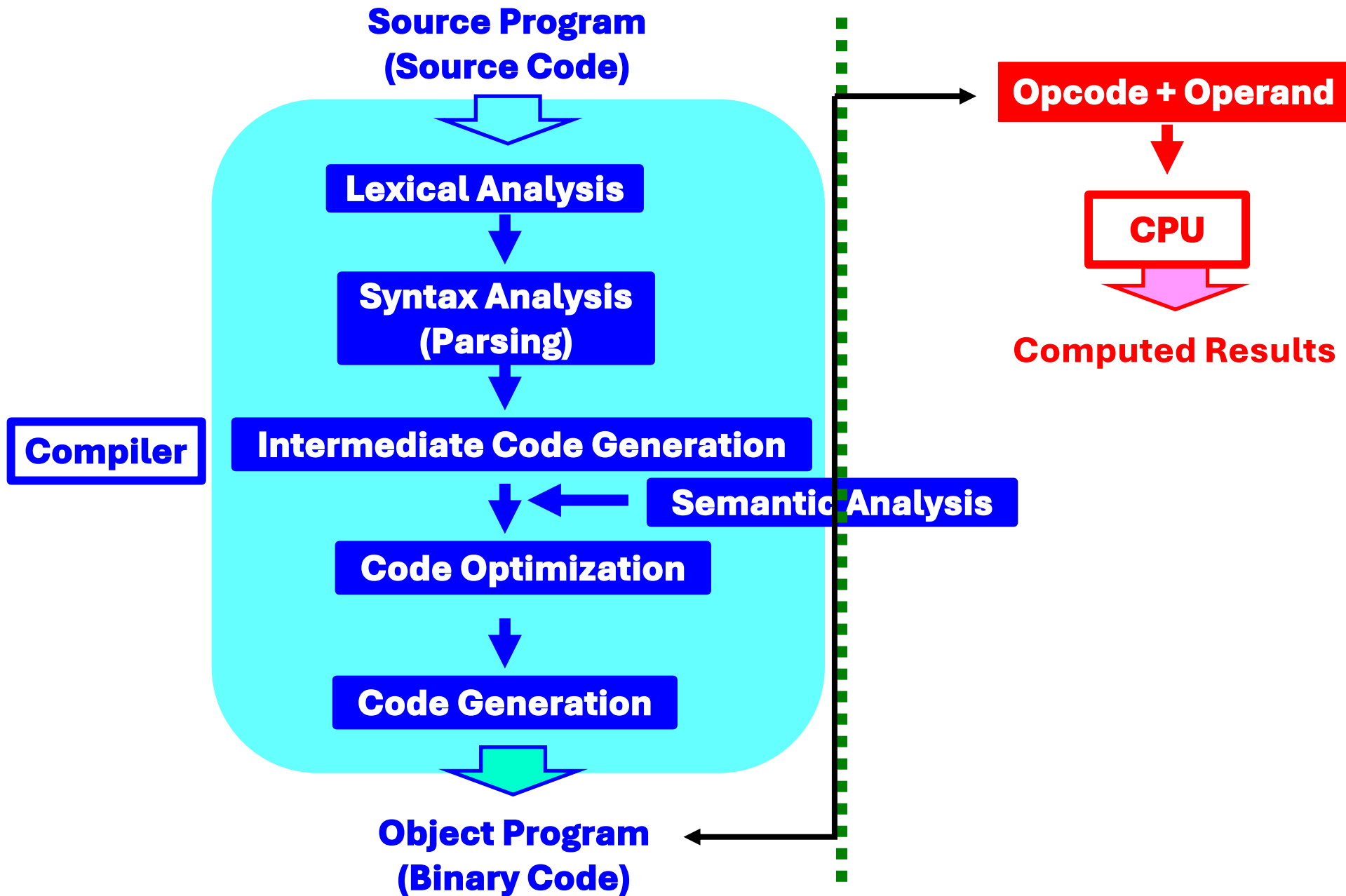
**Here, however, quantum computation is understood to be carried out within the standard quantum computational model with a finite description;**

- **the physical Church–Turing principle asserts that no finitely realizable physical system, including physical systems used to implement quantum computation, can compute a function that is not computable by a classical Turing machine.**

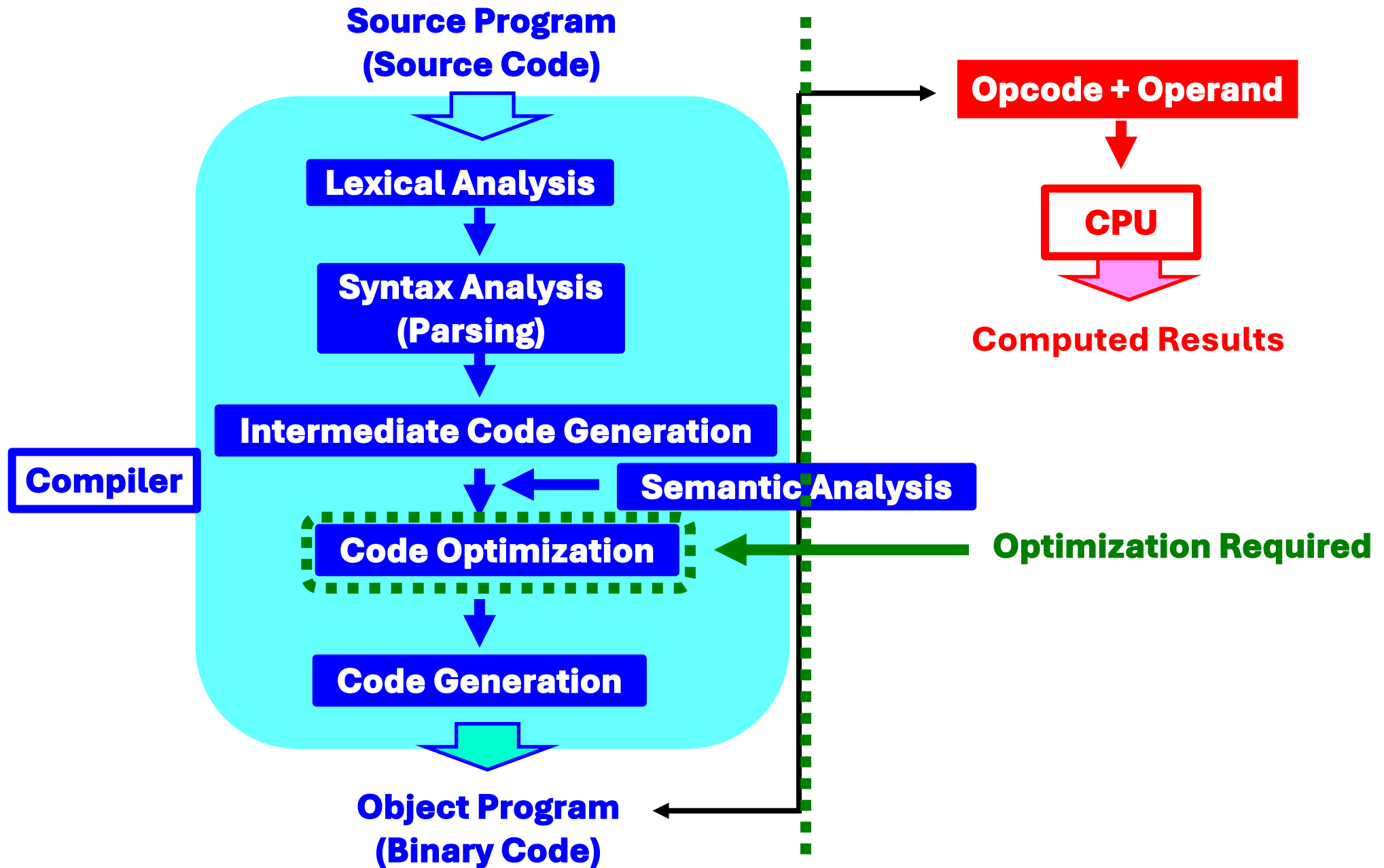
**✂ I am grateful to Prof. Ying for providing the details and references regarding the claims about this fact.**

Computer Calculation Results

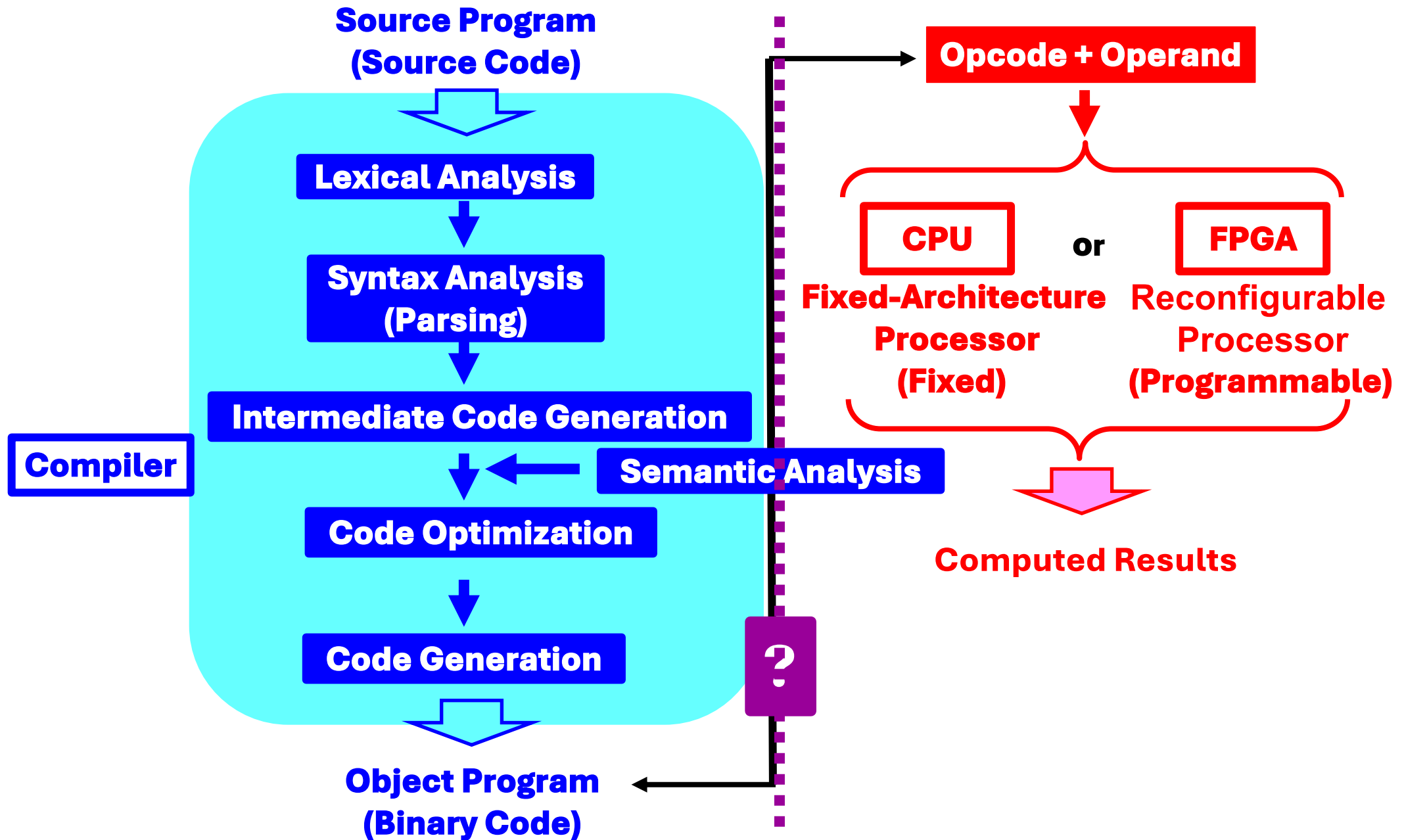
# Process & Role of Compiler



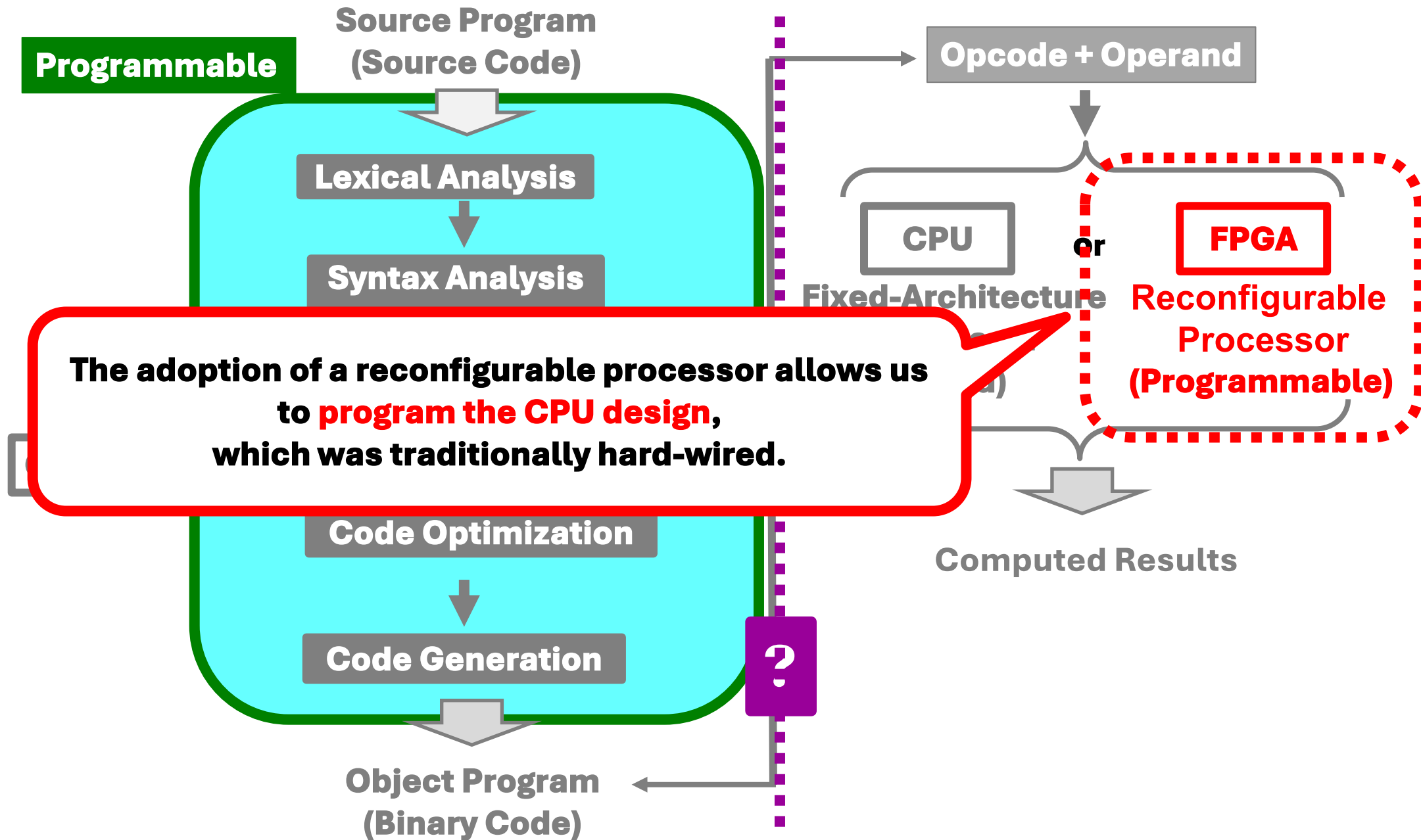
# Process & Role of Compiler



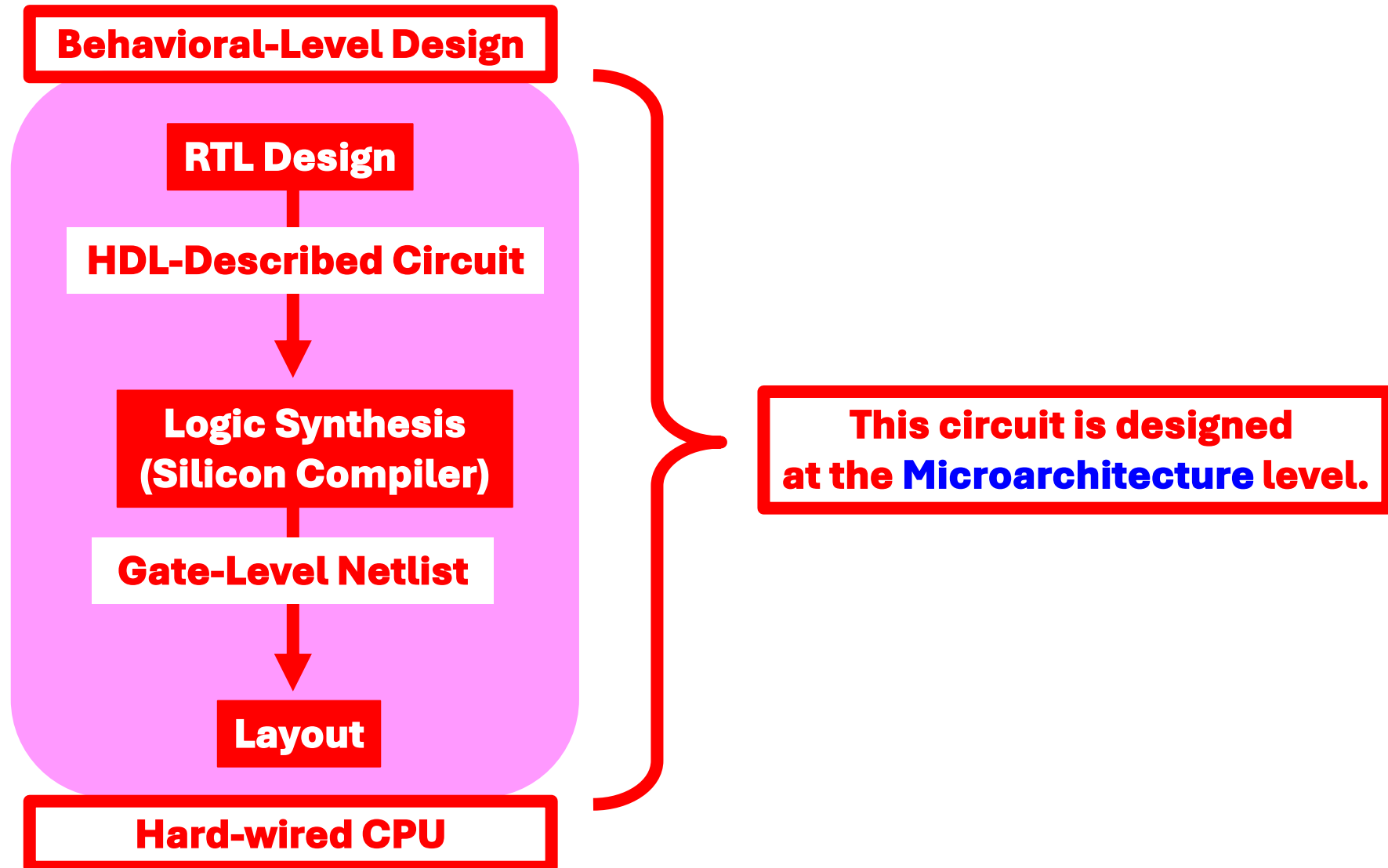
# Process & Role of Compiler



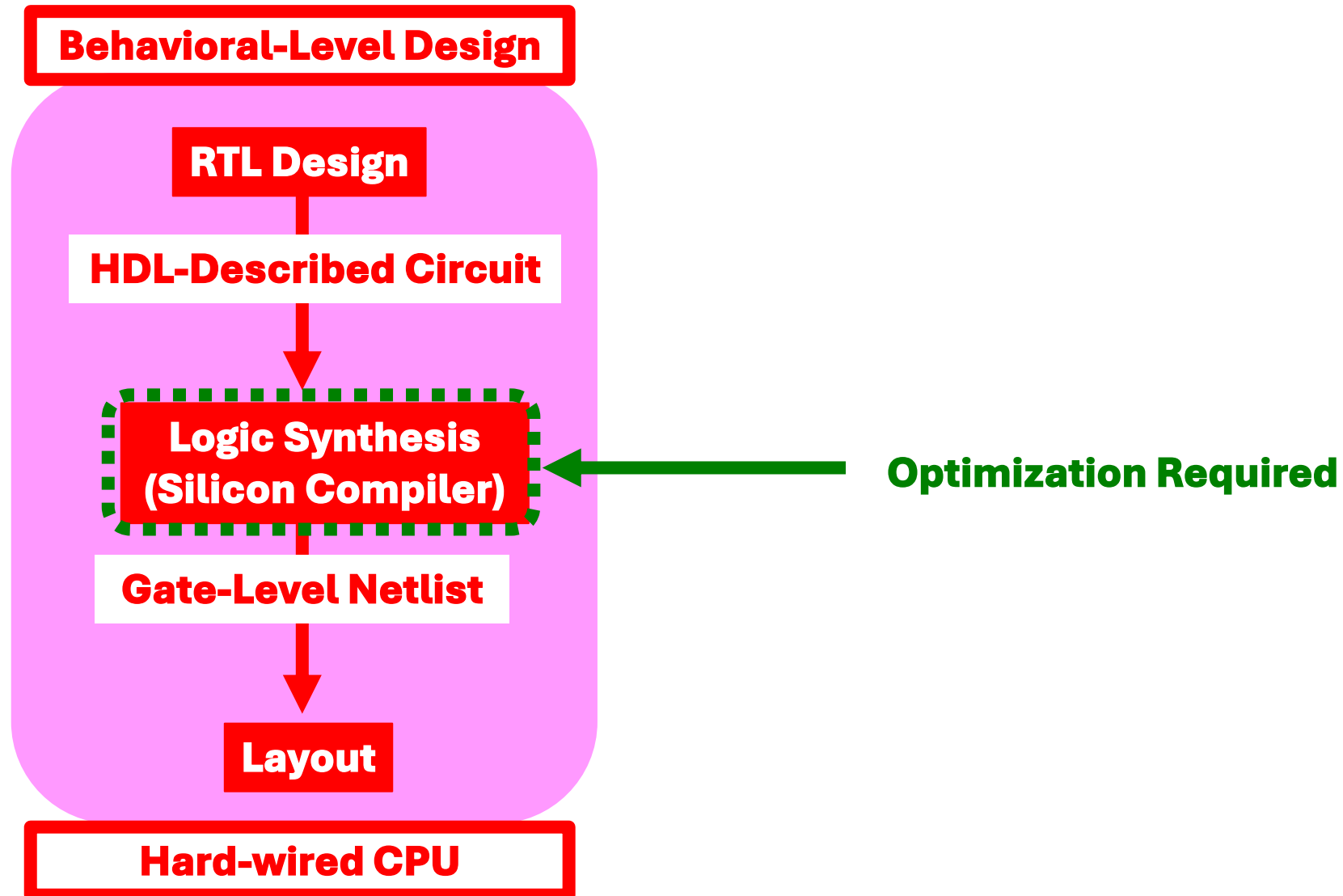
# Process & Role of Compiler



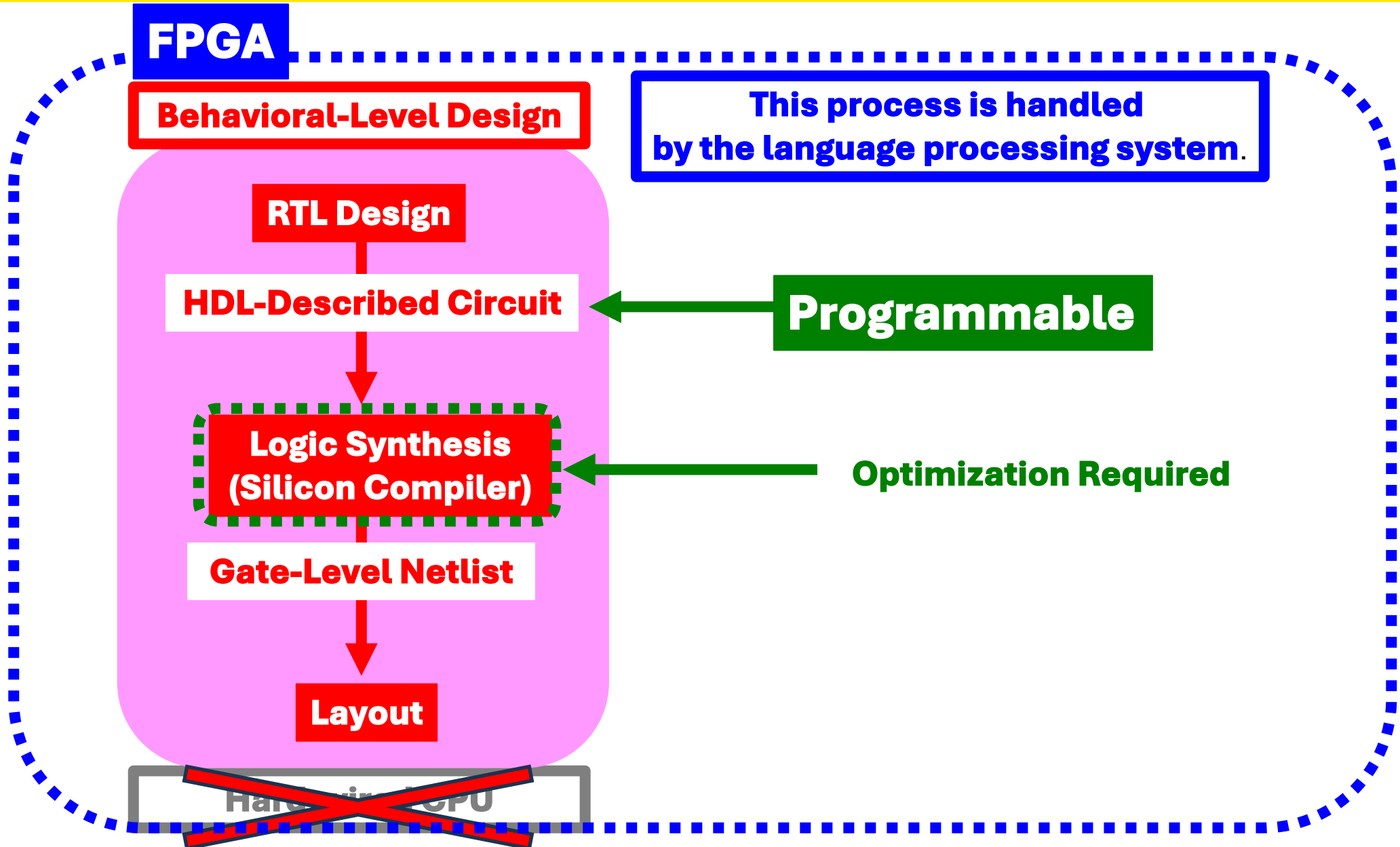
# Process of CPU Design



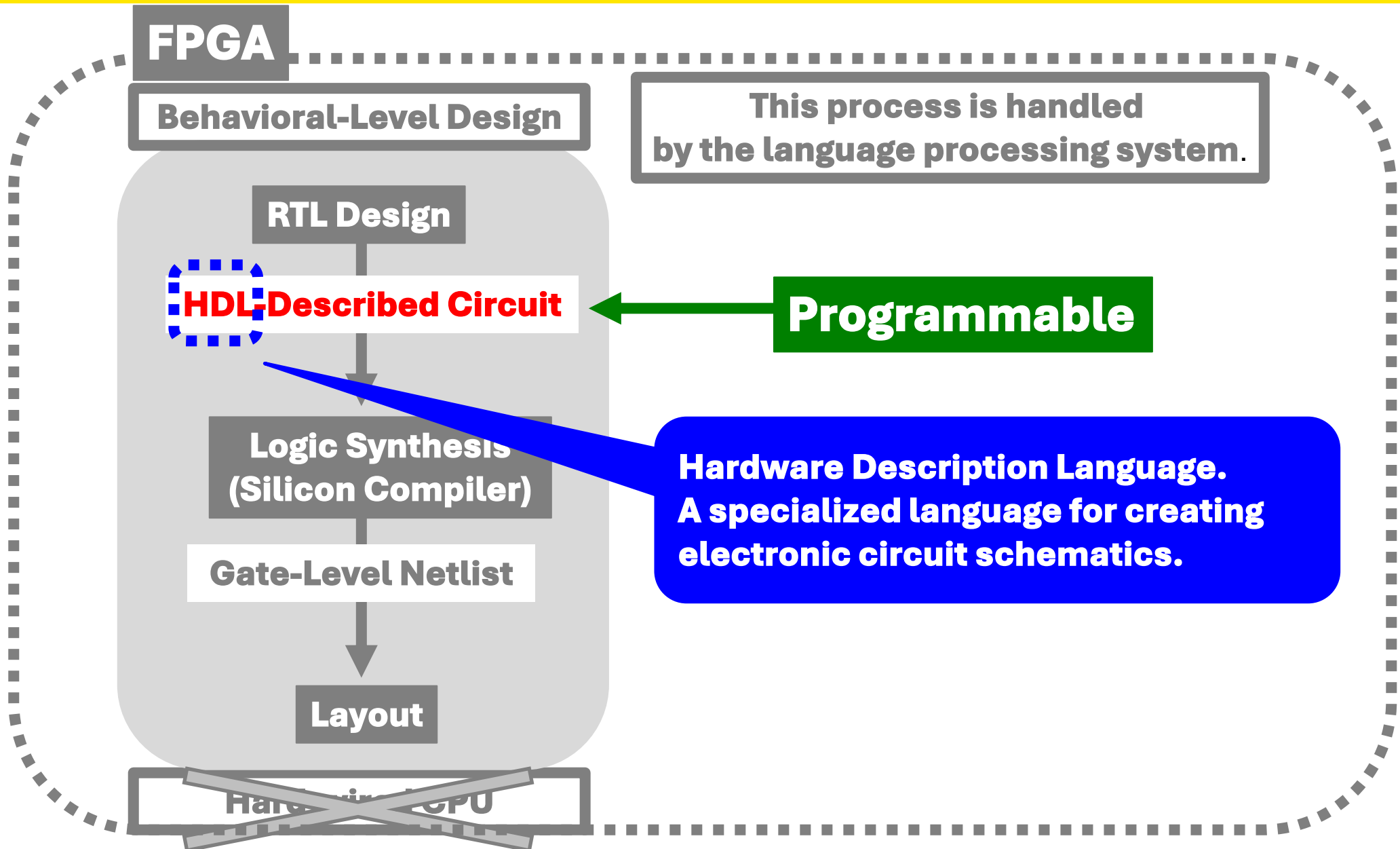
# Process of CPU Design



# Process of ~~CPU Design~~ FPGA Manipulation

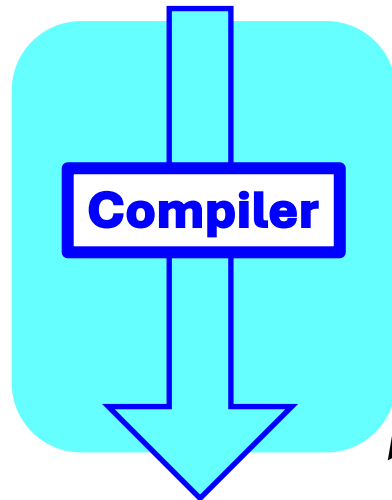


# Process of ~~CPU Design~~ FPGA Manipulation



# Signal Processing in CPU

Source Program  
(Source Code)



Object Program  
(Binary Code)

Opcode + Operand

Digital Electrical Signal

Mapping & Scheduling

The electrical signals converted from the object program become so **vast** that they cannot be processed all at once by a gate-level netlist hard-wired into the CPU.

traverses

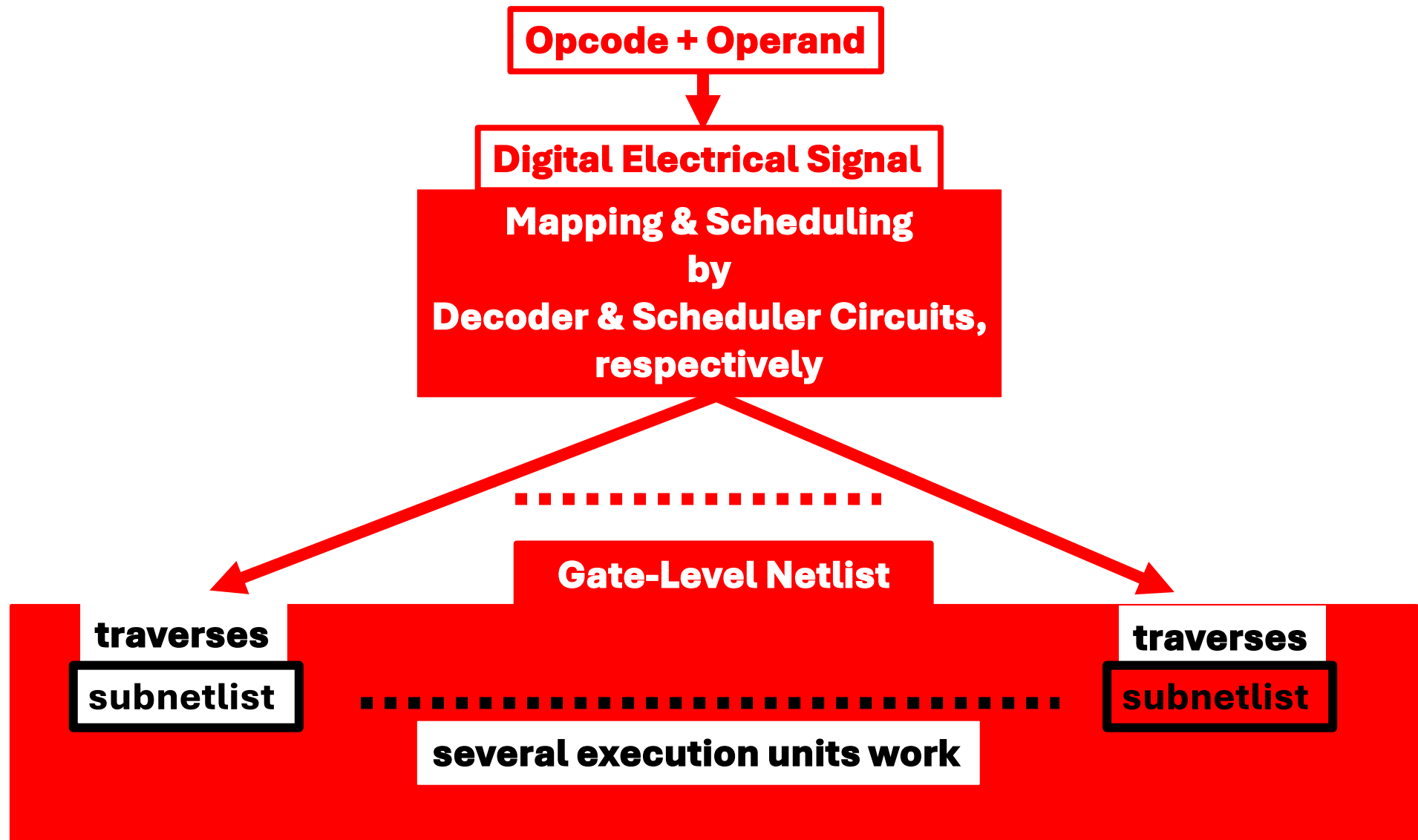
subnetlist

several execution units work

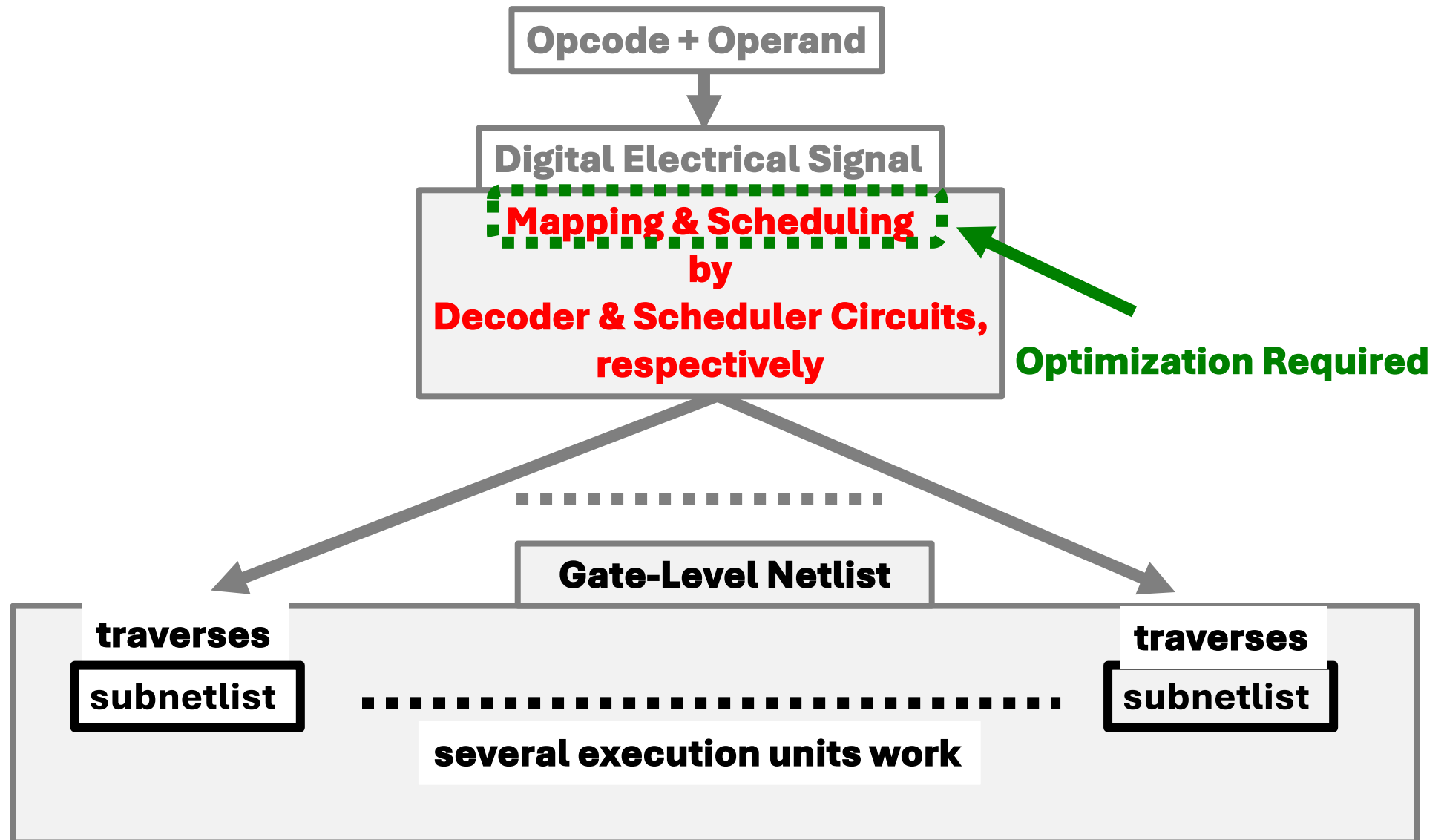
traverses

subnetlist

# Signal Processing in CPU



# Signal Processing in CPU



# Signal Processing in ~~CPU~~ FPGA

**FPGA**

This process is handled  
by the language processing system.

Operand

Digital Electrical Signal

**Mapping & Scheduling**  
by  
**Decoder & Scheduler Circuits,**  
respectively

Optimization Required

affected by this change

Gate-Level Netlist  
**Programmable**

traverses  
subnetlist

traverses  
subnetlist

several execution units work

# Semantics & Optimization

- Each optimization is a major factor determining the capabilities of the computing system.
  - # Optimization in Language Processing System.
  - # Optimization in Logic Synthesis.
  - # Optimization in Mapping.
  - # Optimization in Scheduling.

With quantum computing in mind, we need

- # Hoare Logic, Dijkstra's Predicate Transformer Semantics for Language System.
- # Propositional Logic for Combinational Circuit.
- # Temporal Logic Logic for Sequential Circuit.
- # Quantum Logic Logic for Quantum Logical Circuit.
- # Quantum Hoare Logic Logic for Quantum Logical Circuit.

# Semantics & Optimization

- Each optimization is a major factor determining the capabilities of the computing system.

# Opt

# Opt

# Opt

# Opt

**Each optimization is  
a form of program or circuit transformation  
and  
must not alter the intent of the source program.  
Therefore,  
preserving semantics are crucial.**

With q

# Hoare

# Propositional Logic for Combinational Circuit.

# Temporal Logic Logic for Sequential Circuit.

# Quantum Logic Logic for Quantum Logical Circuit.

# Quantum Hoare Logic Logic for Quantum Logical Circuit.

tem.

# Semantics & Optimization

- **Each optimization is a major factor determining the capabilities of the computing system.**

# Optimization in Language Processing System.

# Optimization in Logic Synthesis.

# Optimization in Mapping.

# Optimization in Scheduling.

With quantum computing in mind, we need

# Hoare Logic, Dijkstra's Predicate Transformer Semantics for Language System.

# Propositional Logic for Combinational Circuit.

# Temporal Logic Logic for Sequential Circuit.

# Quantum Logic Logic for Quantum Logical Circuit.

# Quantum Hoare Logic Logic for Quantum Logical Circuit.

# Semantics & Optimization

- Each optimization is a major factor determining the capabilities of the computing system.

## # Optimization in Language Processing System.

# Optimization in Logic Synthesis.

# Optimization in Mapping.

# Optimization in Scheduling.

With quantum computing in mind, **we need**

## # Hoare Logic, Dijkstra's Predicate Transformer Semantics for Language System.

# Propositional Logic for Combinational Circuit.

# Temporal Logic Logic for Sequential Circuit.

# Quantum Logic Logic for Quantum Logical Circuit.

# Quantum Hoare Logic Logic for Quantum Logical Circuit.

# Semantics & Optimization

$$xy = \underbrace{x + x + \cdots + x}_{(y-1) \text{ additions}}$$

**We have already seen this recursive program for the case where  $x = 2$  and  $y = n$ ,**

$$\begin{cases} f(0) = 0, \\ f(n) = f(n-1) + 2, \quad n = 1, 2, \dots, \end{cases}$$

**but here we adopt an iterative approach.**

**time complexity  $\sim O(y)$**

```
hirokawa@LAPTOP-9VQV46H  ×  +  ▾
File Edit Options Buffers Tools C Help
#include <stdio.h>
#include <time.h>

long double main()
{
    long double X, Y, Z;
    long int I;
    double cpu_time;

    printf("x=");
    scanf("%Lf", &X);
    printf("y=");
    scanf("%Lf", &Y);

    Z=X;
    I=1;
    while(I<Y){
        Z=Z+X;
        I=I+1;
    }

    cpu_time=(double)clock() / CLOCKS_PER_SEC;

    printf("x*y = %Lf\n", Z);
    printf("Number of Loops = %ld\n", I-1);
    printf("CPU Time = %fsec\n", cpu_time);

    return 0;
}
```

# Semantics & Optimization

```
hirokawa@LAPTOP-9VQV46H x + v
hirokawa@LAPTOP-9VQV46HK:~/Programming/C/lectures$ ./times0.out
x=1
y=1000000
x*y = 1000000.000000
Number of Loops = 999999
CPU Time = 0.005361sec
hirokawa@LAPTOP-9VQV46HK:~/Programming/C/lectures$ ./times0.out
x=1000000
y=1000000
x*y = 1000000000000.000000
Number of Loops = 999999
CPU Time = 0.005286sec
hirokawa@LAPTOP-9VQV46HK:~/Programming/C/lectures$ ./times0.out
x=1
y=100000000
x*y = 10000000.000000
Number of Loops = 999999
CPU Time = 0.028560sec
hirokawa@LAPTOP-9VQV46HK:~/Programming/C/lectures$ ./times0.out
x=10000000
y=10000000
x*y = 100000000000000.000000
Number of Loops = 999999
CPU Time = 0.029405sec
hirokawa@LAPTOP-9VQV46HK:~/Programming/C/lectures$ ./times0.out
x=1
y=1000000000
x*y = 1000000000.000000
Number of Loops = 99999999
CPU Time = 0.265413sec
hirokawa@LAPTOP-9VQV46HK:~/Programming/C/lectures$ ./times0.out
x=1000000000
y=1000000000
x*y = 1000000000000000000.000000
Number of Loops = 99999999
CPU Time = 0.270269sec
hirokawa@LAPTOP-9VQV46HK:~/Programming/C/lectures$
```

```
hirokawa@LAPTOP-9VQV46H x + v
hirokawa@LAPTOP-9VQV46HK:~/Programming/C/lectures$ ./times0.out
x=1
y=10000000000
x*y = 10000000000.000000
Number of Loops = 999999999
CPU Time = 2.674888sec
hirokawa@LAPTOP-9VQV46HK:~/Programming/C/lectures$ ./times0.out
x=10000000000
y=10000000000
x*y = 1000000000000000000.000000
Number of Loops = 999999999
CPU Time = 2.645777sec
hirokawa@LAPTOP-9VQV46HK:~/Programming/C/lectures$ ./times0.out
x=1
y=100000000000
x*y = 100000000000.000000
Number of Loops = 9999999999
CPU Time = 27.638808sec
hirokawa@LAPTOP-9VQV46HK:~/Programming/C/lectures$ ./times0.out
x=100000000000
y=100000000000
x*y = 100000000000000000000.000000
Number of Loops = 9999999999
CPU Time = 26.638178sec
hirokawa@LAPTOP-9VQV46HK:~/Programming/C/lectures$ ./times0.out
x=1
y=1000000000000
x*y = 1000000000000.000000
Number of Loops = 99999999999
CPU Time = 274.186348sec
hirokawa@LAPTOP-9VQV46HK:~/Programming/C/lectures$ ./times0.out
x=1000000000000
y=1000000000000
x*y = 1000000000000000000000.000000
Number of Loops = 99999999999
CPU Time = 281.419479sec
hirokawa@LAPTOP-9VQV46HK:~/Programming/C/lectures$
```

# Semantics & Optimization

```
hirokawa@LAPTOP-9VQV46H x + v
hirokawa@LAPTOP-9VQV46HK:~/Programming/C/lectures$ ./times0.out
x=1
y=1000000
x*y = 1000000.000000
Number of Loops = 999999
CPU Time = 0.005361sec
hirokawa@LAPTOP-9VQV46HK:~/Programming/C/lectures$ ./times0.out
x=1000000
y=1000000
x*y = 1000000000000.000000
Number of Loops = 999999
CPU Time = 0.005286sec
hirokawa@LAPTOP-9VQV46HK:~/Programming/C/lectures$ ./times0.out
x=1
y=100000000
x*y = 10000000.000000
Number of Loops = 999999
CPU Time = 0.028560sec
hirokawa@LAPTOP-9VQV46HK:~/Programming/C/lectures$ ./times0.out
x=100000000
y=100000000
x*y = 1000000000000000.000000
Number of Loops = 999999
CPU Time = 0.029405sec
hirokawa@LAPTOP-9VQV46HK:~/Programming/C/lectures$ ./times0.out
x=1
y=1000000000
x*y = 1000000000.000000
Number of Loops = 99999999
CPU Time = 0.265413sec
hirokawa@LAPTOP-9VQV46HK:~/Programming/C/lectures$ ./times0.out
x=1000000000
y=1000000000
x*y = 1000000000000000000.000000
Number of Loops = 99999999
CPU Time = 0.270269sec
hirokawa@LAPTOP-9VQV46HK:~/Programming/C/lectures$
```

```
hirokawa@LAPTOP-9VQV46H x + v
hirokawa@LAPTOP-9VQV46HK:~/Programming/C/lectures$ ./times0.out
x=1
y=100000000000
x*y = 10000000000.000000
Number of Loops = 999999999
CPU Time = 2.674888sec
hirokawa@LAPTOP-9VQV46HK:~/Programming/C/lectures$ ./times0.out
x=100000000000
y=100000000000
x*y = 10000000000000000000.000000
Number of Loops = 999999999
CPU Time = 2.645777sec
hirokawa@LAPTOP-9VQV46HK:~/Programming/C/lectures$ ./times0.out
x=1
y=1000000000000
x*y = 100000000000.000000
Number of Loops = 9999999999
CPU Time = 27.638808sec
hirokawa@LAPTOP-9VQV46HK:~/Programming/C/lectures$ ./times0.out
x=1000000000000
y=1000000000000
x*y = 1000000000000000000000.000000
Number of Loops = 9999999999
CPU Time = 26.638178sec
hirokawa@LAPTOP-9VQV46HK:~/Programming/C/lectures$ ./times0.out
x=1
y=10000000000000
x*y = 1000000000000.000000
Number of Loops = 99999999999
CPU Time = 274.186348sec
hirokawa@LAPTOP-9VQV46HK:~/Programming/C/lectures$ ./times0.out
x=10000000000000
y=10000000000000
x*y = 10000000000000000000000.000000
Number of Loops = 99999999999
CPU Time = 281.419479sec
hirokawa@LAPTOP-9VQV46HK:~/Programming/C/lectures$
```

# Semantics & Optimization

## Slow Multiplication Program

**Let's experience just how slow this program is!**

**The results for  $x \times y = 1 \times 10^7$  &  $x \times y = 10^7 \times 10^7$  are [here](#).**

**The results for  $x \times y = 1 \times 10^8$  &  $x \times y = 10^8 \times 10^8$  are [here](#).**

**The results for  $x \times y = 1 \times 10^9$  &  $x \times y = 10^9 \times 10^9$  are [here](#).**

**The results for  $x \times y = 1 \times 10^{10}$  &  $x \times y = 10^{10} \times 10^{10}$  are [here](#).**

# Semantics & Optimization

$$xy = \underbrace{x + x + \cdots + x}_{(y-1) \text{ additions}}$$

## Program Transformation

**time complexity**  $\sim O(y)$

**Improvement**

**time complexity**  $\sim O(\log_2 y)$

using  
Hoare Logic  
&

**Dijkstra's Predicate Transformer Semantics**

# Semantics & Optimization

## Fast Multiplication Program

**Let's experience firsthand how this program  
has been improved to run faster!**

**The results for  $x \times y = 1 \times 10^7$  &  $x \times y = 10^7 \times 10^7$  are [here](#).**

**The results for  $x \times y = 1 \times 10^8$  &  $x \times y = 10^8 \times 10^8$  are [here](#).**

**The results for  $x \times y = 1 \times 10^9$  &  $x \times y = 10^9 \times 10^9$  are [here](#).**

**The results for  $x \times y = 1 \times 10^{10}$  &  $x \times y = 10^{10} \times 10^{10}$  are [here](#).**

# Semantics & Optimization

$$xy = \underbrace{x + x + \cdots + x}_{(y-1) \text{ additions}}$$

Program Transformation

time complexity  $\sim O(y)$

Improvement

time complexity  $\sim O(\log_2 y)$

using  
Hoare Logic  
&

Dijkstra's Predicate Transformer Semantics

**The standard compiler CANNOT perform this optimization automatically.**

**Computing systems provide built-in functions so that front-end programmers do not have to handle that programming themselves.**

# Semantics & Optimization

begin

$u := x; w := y; s := 0$

```
while  $0 < w$  do
    if even( $w$ ) then  $w := w \text{ div\_2}; u := 2 * u$ 
    else  $w := w - 1; s := s + u$     fi
od
```

end

# Semantics & Optimization

## Pareto Principle (80/20 Rule):

**Approximately 80% of the program's total execution time is spent processing specific iteration statements that account for only about 20% of the total lines of a source program.**

```
while 0 < w do
    if even(w) then w := w div_2; u := 2 * u
    else w := w - 1; s := s + u    fi
od
```

end

# Semantics & Optimization

Developing effective iteration sentences becomes important.

**Pareto Principle (80/20 Rule):**

**Approximately 80% of the program's total execution time is spent processing specific iteration statements that account for only about 20% of the total lines of a source program.**

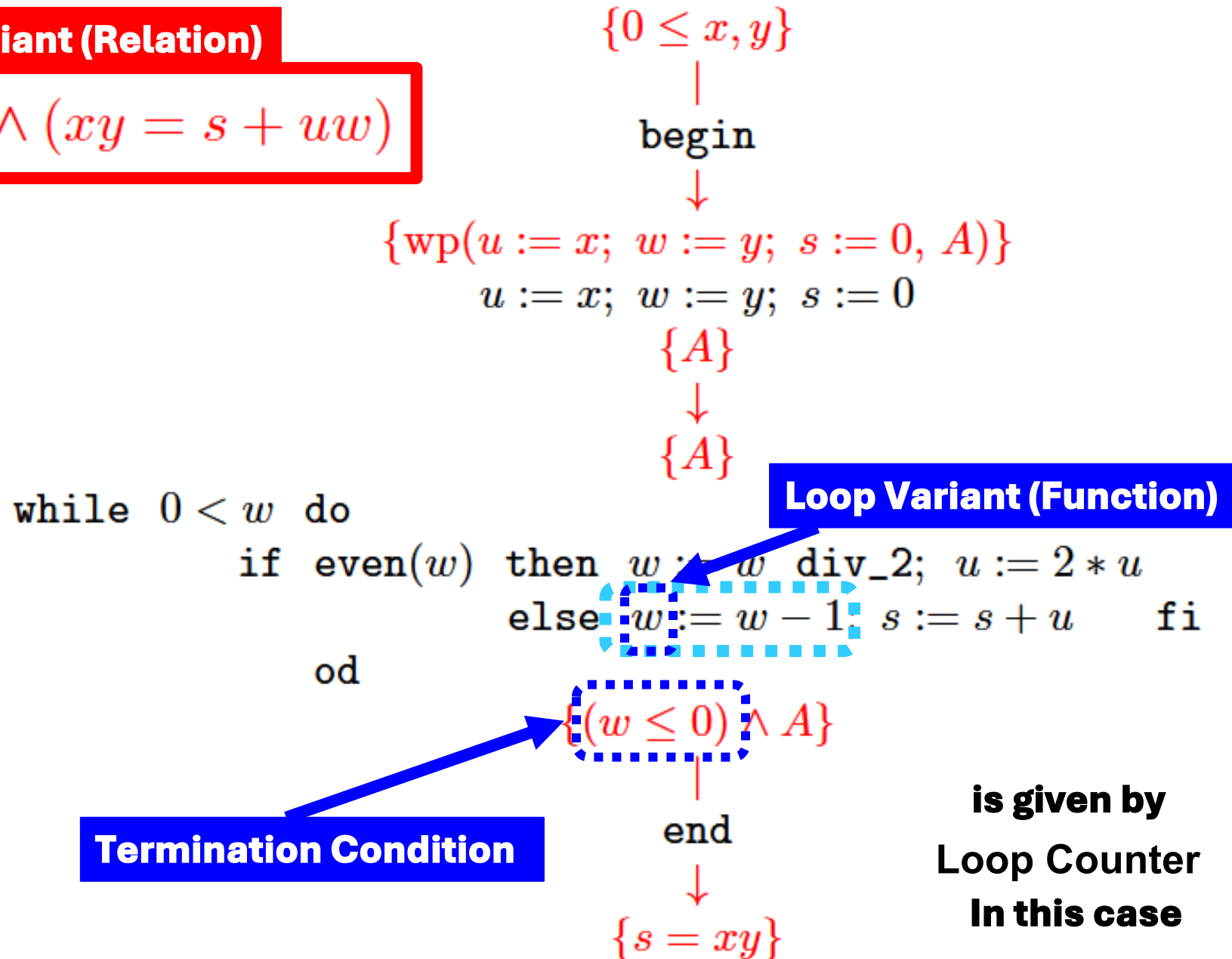
```
while 0 < w do
    if even(w) then w := w div_2; u := 2 * u
    else w := w - 1; s := s + u    fi
od
```

end

# Semantics & Optimization

## Loop Invariant (Relation)

$$A \equiv (0 \leq w) \wedge (xy = s + uw)$$



# Semantics & Optimization

$$A \equiv (0 \leq w) \wedge (xy = s + uw)$$

$$\begin{cases} xy = (2x)(y/2) & \text{if } y \text{ even,} \\ xy = x(y-1) + x & \text{if } y \text{ is odd.} \end{cases}$$

**When multiplying or dividing by 2, we never actually perform multiplication or division; we simply use shift operations.**

```
while 0 < w do
  if even(w) then w := w div_2; u := 2 * u
  else w := w - 1; s := s + u    fi
od
```

$$\{0 \leq x, y\}$$

begin

$$\{\text{wp}(u := x; w := y; s := 0, A)\}$$

$u := x; w := y; s := 0$

$$\{A\}$$

$$\{A\}$$

$$\{(w \leq 0) \wedge A\}$$

end

$$\{s = xy\}$$

# Semantics & Optimization

$$A \equiv (0 \leq w) \wedge (xy = s + uw)$$

$$\begin{cases} xy = (2x)(y/2) & \text{if } y \text{ even,} \\ xy = x(y-1) + x & \text{if } y \text{ is odd.} \end{cases}$$

The number in the register for x shifts left by one digit and the number in the register for y shifts right by one digit.

```
while 0 < w do
  if even(w) then w := w div_2; u := 2 * u
  else w := w - 1; s := s + u    fi
od
```

$$\{0 \leq x, y\}$$

begin

$$\{\text{wp}(u := x; w := y; s := 0, A)\}$$

$u := x; w := y; s := 0$

$$\{A\}$$

$$\{A\}$$

$$\{(w \leq 0) \wedge A\}$$

end

$$\{s = xy\}$$

# Semantics & Optimization

- Each optimization is a major factor determining the capabilities of the computing system.
  - # Optimization in Language Processing System.
  - # **Optimization in Logic Synthesis.**
  - # Optimization in Mapping.
  - # Optimization in Scheduling.

With quantum computing in mind, **we need**

- # Hoare Logic, Dijkstra's Predicate Transformer Semantics for Language System.
- # **Propositional Logic for Combinational Circuit.**
- # Temporal Logic Logic for Sequential Circuit.
- # Quantum Logic for Quantum Logical Circuit.
- # Quantum Hoare Logic Logic for Quantum Software.

# Semantics & Optimization

- Each optimization is a major factor determining the capabilities of the computing system.
  - # Optimization in Language Processing System.
  - # **Optimization in Logic Synthesis.**
  - # Optimization in Mapping.
  - # Optimization in Scheduling.

With quantum computing in mind, **we need**

- # Hoare Logic, Dijkstra's Predicate Transformer Semantics for Language System.
- # Propositional Logic for Combinational Circuit.
- # **Temporal Logic for Sequential Circuit.**
- # Quantum Logic for Quantum Logical Circuit.
- # Quantum Hoare Logic Logic for Quantum Software.

# Semantics & Optimization

- Each optimization is a major factor determining the capabilities of the computing system.

# Optimization in Language Processing System.

# **Optimization in Logic Synthesis.**

# Optimization in Mapping.

# Optimization in Scheduling.

**The electrical signals  
traveling through  
the circuit are dynamic**

With quantum computing in mind, we need

# Hoare Logic, Dijkstra's Predicate Transformer Semantics for Language System.

# Propositional Logic for Combinational Circuit.

# **Temporal Logic for Sequential Circuit.**

- **Temporal logic is used to describe the dynamic behavior of logical formulas.**
- **While implication always  $(p \longrightarrow p)$  holds in classical logic (propositional logic, predicate logic), the Omnitemporal Implication  $\Box(p \longrightarrow p)$  in temporal logic does not necessarily hold. Here, we note that the time of the " $p$ " on the right differs from that of the " $p$ " on the left.**

# Semantics & Optimization

- Each optimization is a major factor determining the capabilities of the system.

**This logical formula is to be proven.**

$\alpha_1$  as  $(\alpha \rightarrow ((\alpha \rightarrow \alpha) \rightarrow \alpha))$ .

$\alpha_2$  as  $((\alpha \rightarrow ((\alpha \rightarrow \alpha) \rightarrow \alpha)) \rightarrow ((\alpha \rightarrow (\alpha \rightarrow \alpha)) \rightarrow (\alpha \rightarrow \alpha)))$ .

$\alpha_3$  as  $((\alpha \rightarrow (\alpha \rightarrow \alpha)) \rightarrow (\alpha \rightarrow \alpha))$ .

$\alpha_4$  as  $(\alpha \rightarrow (\alpha \rightarrow \alpha))$ .

$\alpha_5$  as  $(\alpha \rightarrow \alpha)$ .

With

# Ho

# Propositional

# Temporal Logic for Sequential Circuit.

Large System.

Sequential Circuit.

- Temporal logic is used to describe the dynamic behavior of logical formulas.
- While implication  $(p \rightarrow p)$  always holds in classical logic (propositional logic, predicate logic), the **Omnitemporal Implication**  $\Box(p \rightarrow p)$  in temporal logic **does not necessarily hold**. Here, we note that the time of the “ $p$ ” on the right differs from that of the “ $p$ ” on the left.

# Semantics & Optimization

- Each optimization is a major factor determining the capabilities of the system.

**This logical formula is to be proven.**

$\alpha_1$  is an axiom.

$\alpha_2$  is an axiom.

$\alpha_3$  is conclusion of MP from the premises  $\alpha_1$  and  $\alpha_2$ .

$\alpha_4$  is an axiom.

$\alpha_5$  is the conclusion of MP from the premises  $\alpha_3$  and  $\alpha_4$ .

With

# Ho

# Propositional

# Temporal Logic for

ational Circuit.

ge System.

- Temporal logic is used to describe the dynamic behavior of logical formulas.
- While implication  $(p \rightarrow p)$  always holds in classical logic (propositional logic, predicate logic), the Omnitemporal Implication  $\Box(p \rightarrow p)$  in temporal logic does not necessarily hold. Here, we note that the time of the “ $p$ ” on the right differs from that of the “ $p$ ” on the left.

# Semantics & Optimization

- **Each optimization is a major factor determining the capabilities of the computing system.**

**# Optimization in Language Processing System.**

**# Optimization in Logic Synthesis.**

**# Optimization in Mapping.**

**# Optimization in Scheduling.**

**These comprehensive optimization problems are NP-hard.**

With quantum computing in mind, we need

**# Hoare Logic, Dijkstra's Predicate Transformer Semantics for Language System.**

**# Propositional Logic for Combinational Circuit.**

**# Temporal Logic for Sequential Circuit.**

**# Quantum Logic for Quantum Logical Circuit.**

**# Quantum Hoare Logic Logic for Quantum Software.**

# Semantics & Optimization

- Each optimization is a major factor determining the capabilities of the computing system.

- # Optimization in Language Processing System.
- # Optimization in Logic Synthesis.
- # Optimization in Mapping.
- # Optimization in Scheduling.

These comprehensive optimization problems are NP-hard.

**With quantum computing in mind, we need**

- # Hoare Logic, Dijkstra's Predicate Transformer Semantics for Language System.
- # Propositional Logic for Combinational Circuit.
- # Temporal Logic for Sequential Circuit.
- # **Quantum Logic for Quantum Logical Circuit.**
- # Quantum Hoare Logic Logic for Quantum Software.

# Semantics & Optimization

- Logical formulas in quantum logic describe **closed** subspaces that serve as state spaces.
- **By the orthogonal projection theorem, the closed subspace corresponds to a projection operator.**
- That projection operator represents the observation of a quantum measurement; therefore, a probabilistic interpretation is introduced into the measurement.
- **In quantum logic, the distributive law that holds in classical logic (propositional logic and predicate logic) does not hold.**

With quantum computing in mind, we need

- # Hoare Logic, Dijkstra's Predicate Transformer Semantics for Language System.
- # Propositional Logic for Combinational Circuit.
- # Temporal Logic for Sequential Circuit.
- # **Quantum Logic for Quantum Logical Circuit.**
- # Quantum Hoare Logic for Quantum Software.

# Semantics & Optimization

- Each optimization is a major factor determining the capabilities of the computing system.

# Optimization in Language Processing System.

# Optimization in Logic Synthesis.

# Optimization in Mapping.

# Optimization in Scheduling.

**These comprehensive optimization problems are NP-hard.**

**With quantum computing in mind, we need**

# Hoare Logic, Dijkstra's Predicate Transformer Semantics for Language System.

# Propositional Logic for Combinational Circuit.

# Temporal Logic for Sequential Circuit.

# Quantum Logic for Quantum Logical Circuit.

# Quantum Hoare Logic Logic for Quantum Software.

# Semantics & Optimization

- Each optimization is a major factor determining the capabilities of the computing system.

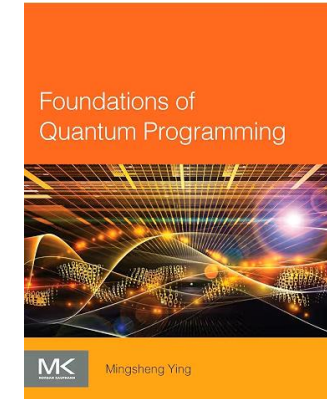
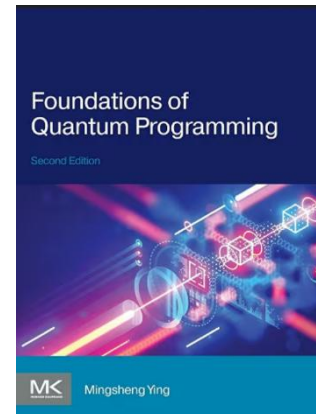
# Optimization in Language Processing.

# Optimization in Logic Synthesis.

# Optimization in Mapping.

# Optimization in Scheduling.

These comprehensive optimizations



With quantum computing in mind, we need

# Hoare Logic, Dijkstra's Predicate Transformer Semantics for Language System.

# Propositional Logic for Combinational Circuit.

# Temporal Logic for Sequential Circuit.

# Quantum Logic for Quantum Logical Circuit.

# Quantum Hoare Logic Logic for Quantum Software.

# Semantics & Optimization

- Each optimization is a major factor determining the capabilities of the computing system.

# Optimization in Language Processing System.

# Optimization in Logic Synthesis.

# Optimization in Mapping.

# Optimization in Scheduling.

**These comprehensive optimization problems are NP-hard.**

**With quantum computing in mind, we need**

# Hoare Logic, Dijkstra's Predicate Transformer Semantics for Language System.

# Propositional Logic for Combinational Circuit.

# Temporal Logic for Sequential Circuit.

# Quantum Logic for Quantum Logical Circuit.

# Quantum Hoare Logic Logic for Quantum Software.

**What kind of approach is required  
to combine these logics?**

# Hoar Logic for Classical Programming

## Program with Assertion

- A program with assertions is written as

$$\{A\} S \{B\}$$

or

$$\begin{array}{c} \{A\} \\ S \\ \{B\} \end{array}$$

where  $S$  is a statement (i.e., instruction, command), and  $A$  and  $B$  are assertions.

- The assertion  $A$  is called the **precondition** of  $S$ , and the assertion  $B$  is called the **postcondition** of  $S$ .

# Hoar Logic for Classical Programming

## Hoare Triple

- Let  $S$  be a statement (i.e., instruction, or command),
- Let  $A$  and  $B$  be the preconditions and postconditions of  $S$ , respectively.
- Then,  $\{A\} S \{B\}$  is called the **Hoare triple**.

## Program with Assertions

Assuming a program is a sequence of sentences, let program  $P$  be defined as follows:

# Hoar Logic for Classical Programming

Program without Assertion:

**begin**

$S_1$

$\vdots$

$S_{i-1}$

$S_i$

$\vdots$

$S_n$

**end**

where  $S_i$ ,  $i = 1, \dots, n$ , are statements.

# Hoar Logic for Classical Programming

Program with Assertions:

```
begin  
  { $A_1$ }  
   $S_1$   
  { $B_1$ }  
  ⋮  
  { $A_{i-1}$ }  
   $S_{i-1}$   
  { $B_{i-1}$ }  
  { $A_i$ }  
   $S_i$   
  { $B_i$ }  
  ⋮  
  { $A_n$ }  
   $S_n$   
  { $B_n$ }  
end
```

where  $S_i$ ,  $i = 1, \dots, n$ , are statements.

# Hoar Logic for Classical Programming

**We adopt structured programming based on the Böhm-Jacopini theorem.**

Axiom  $A_{sk}$  (Empty Statement Axiom Schema): Skip

$$\{A\} \text{ skip } \{A\}$$

Axiom  $A_{as}$  (Assignment Axiom Schema): Assignment

$$\{A_t^x\} x := t \{A\}$$

Since the assignment  $x := t$  in  $\text{Min}(PV)$  has the same form as a substitution in predicate logic, **we denote the predicate logic substitution as  $A_x^t$ .**

# Hoar Logic for Classical Programming

**We adopt structured programming based on the Böhm-Jacopini theorem**

**We use only  
this  
for this talk.**

Axiom  $A_{sk}$  (Empty Statement Axiom Schema)

$$\{A\} \text{ skip } \{A\}$$

Axiom  $A_{as}$  (Assignment Axiom Schema): Assignment

$$\{A_t^x\} x := t \{A\}$$

Since the assignment  $x := t$  in  $\text{Min}(PV)$  has the same form as a substitution in predicate logic, **we denote the predicate logic substitution as  $A_x^t$ .**

# Hoar Logic for Classical Programming

Inference Rule **R<sub>cp</sub>** (Composition Rule): Sequence

For sentences  $S_1, \dots, S_n$ , and assertions  $A, B, C_1, \dots, C_{n-1}$ ,

$$\frac{\{A\} S_1 \{C_1\} \dots S_i \{C_i\} S_i \dots \{C_{n-1}\} S_n \{B\}}{\{A\} S_1; \dots ; S_n \{B\}}$$

**Remark:** For each  $i$ , each assertion  $C_i$  serves as both the postcondition of  $S_i$  and the precondition of  $S_{i+1}$ .

# Hoar Logic for Classical Programming

Inference Rule **R<sub>if</sub>** (Conditional Rule): Selection

For sentences  $S_1$ ,  $S_2$ , assertions  $A$ ,  $B$ , and a Boolean expression  $C$ ,

$$\frac{\{C \wedge A\} S_1 \{B\} \quad \{\neg C \wedge A\} S_2 \{B\}}{\{A\} \text{ if } C \text{ then } S_1 \text{ else } S_2 \text{ fi } \{B\}}$$

# Hoar Logic for Classical Programming

Inference Rule  $\mathbf{R}_{\text{wh}}$  (While Rule): Iteration

For a sentence  $S$ , assertion  $A$ , and Boolean expression  $C$ ,

$$\frac{\{C \wedge A\} S \{A\}}{\{A\} \text{ while } C \text{ do } S \text{ od } \{\neg C \wedge A\}}$$

Here,  $A$  is called the **loop invariant**.

# Hoar Logic for Classical Programming

Inference Rule  $\mathbf{R}_{cs}$  (Consequence Rule)

For a sentence  $S$ , assertions  $A, B, C, D$ ,

$$\frac{G_P \vdash (A \rightarrow B) \quad \{B\} S \{C\} \quad G_P \vdash (C \rightarrow D)}{\{A\} S \{D\}}$$

Here,  $G_P$  is the set of non-logical axioms for a theory used by programmers to write a program.

# Dijkstra's Weakest Precondition

- Dijkstra introduces the notion of the **weakest precondition**, shows its existence for **skip**, **assignment**, **sequence**, **selection**, **iteration**, and **abort** statements, and **expresses these weakest preconditions using logical formulas**.

## Weakest Precondition

- The weakest precondition  $\text{wp}(S, B)$  for a statement  $S$  and a postcondition  $B$  is a precondition such that

$$A \rightarrow \text{wp}(S, B)$$

holds for any valid precondition  $A$  of the Hoare triple  $\{A\} S \{B\}$ .

# Dijkstra's Weakest Precondition

## Weakest Precondition of Skip

For any assertion  $B$ ,  $\text{wp}(\text{skip}, B)$  is given by

$$B.$$

## Weakest Precondition of Assignment

For any assertion  $B$ , every individual variable  $x$  and term  $e$  of  $\text{Fml}(AV)$ ,  $\text{wp}(x := e, B)$  is given by

$$B_e^x.$$

Here,  $e$  is applicable for the substitution.

# Dijkstra's Weakest Precondition

## Weakest Precondition of Sequence

For statements  $S_1, \dots, S_n$  and a postcondition  $B$ ,  
 $\text{wp}(S_1; \dots; S_n, B)$  is given by

$$\text{wp}(S_1, \dots \text{wp}(S_{n-1}, \text{wp}(S_n, B)) \dots).$$

In particular,

- $n = 2$ :  $\text{wp}(S_1; S_2, B)$  is

$$\text{wp}(S_1, \text{wp}(S_2, B)).$$

- $n = 3$ :  $\text{wp}(S_1; S_2; S_3, B)$  is

$$\text{wp}(S_1, \text{wp}(S_2, \text{wp}(S_3, B))).$$

# Dijkstra's Weakest Precondition

## Weakest Precondition of Selection

For statements  $S_1$ ,  $S_2$ , a postcondition  $B$ , and a Boolean expression  $C$ ,  $\text{wp}(\text{if } C \text{ then } S_1 \text{ else } S_2 \text{ fi}, B)$  is given by

$$(C \rightarrow \text{wp}(S_1, B)) \wedge (\neg C \rightarrow \text{wp}(S_2, B)).$$

# Dijkstra's Weakest Precondition

## Weakest Precondition of Iteration

Let  $S$  be a statement,  $B$  a postcondition, and  $C$  a Boolean expression.  $\text{wp}(\text{while } C \text{ do } S \text{ od}, B)$  is characterized by

$$\bigvee_{\ell=0}^L H_{\ell}(C, S, B)$$
$$\left( \equiv H_0(C, S, B) \vee H_1(C, S, B) \vee \cdots \vee H_L(C, S, B) \right),$$

where  $H_{\ell}(C, S, B) \not\equiv \text{False}$  and it is defined by

$$H_0(C, S, B) \equiv \neg C \wedge B,$$

$$H_{i+1}(C, S, B) \equiv C \wedge \text{wp}(S, H_i(C, S, B)).$$

# Low-Level Computation

- There are some differences in meaning between low-level and high-level computations.

## Example

- Math

$$x + y = y + x$$

- Assembly Language for CISC Architectures

$$\text{add } x \ y \neq \text{add } y \ x$$

**This difference stems from how registers are used—specifically, where data is stored.**

# Low-Level Computation

- Let consider unitary operators  $A, B$ . We easily have  $AB = I \implies B = A^*$ .
- By devising ways to utilize the quantum registers, quantum circuit  $B$  can be implemented differently from  $A^*$ .

For example, a SWAP gate can be implemented using three CNOT gates, and we have  $A = \text{SWAP} = A^*$ .

- We can make another SWAP gate,  $\text{SWAP}'$ , so that  $\text{SWAP} \text{SWAP}' = I$ .
- We call  $(\text{SWAP} \text{SWAP}')^n$ ,  $n \in \mathbb{N}$ , a **noise generation quantum circuit (NGQC)**, though some people call this a computer virus :p

**NGQC is an implementation of skip.**

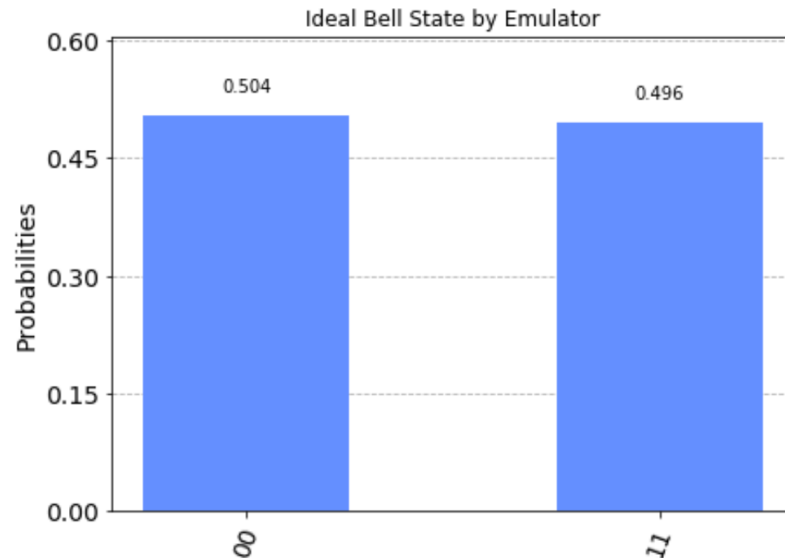
- We expect NGQC to be eliminated during the transpilation optimization phase; however, the IBM Quantum transpiler was incapable of this.
- In other words, **NGQC serves as a tool for evaluating the optimization capabilities of a transpiler.**
- Furthermore, since increasing  $n$  also increases the number of CNOT gates, it enables an **assessment of the quantum processor's resilience against CNOT-induced noise** by examining the gate-level netlist.

# Low-Level Computation

- These are the results of an experiment conducted about 7 years ago in which 'ibmq\_athens' was made to generate a Bell state.

## Example

★QISKitの結果: ショット数 8192  
{ '00': 4130, '11': 4062 }



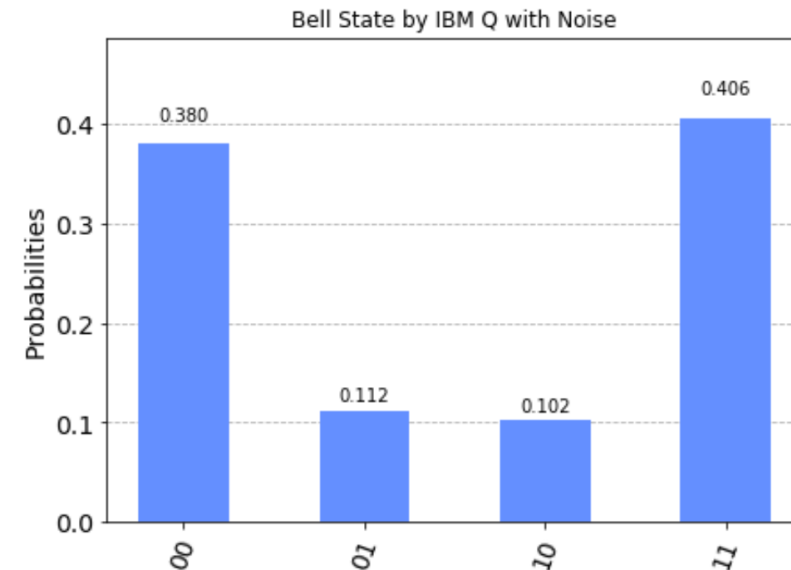
## Emulator Results

We use the backend: ibmq\_athens

★IBM Qの結果: ショット数 8192

ノイズ度 5

{ '01': 919, '10': 839, '11': 3322, '00': 3112 }



$n = 5$

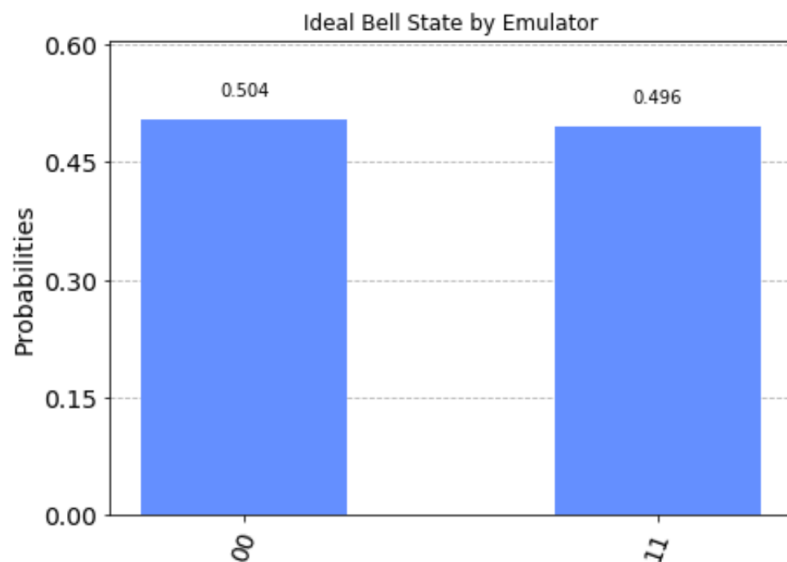
$$\text{Bell State: } \frac{|00\rangle + |11\rangle}{\sqrt{2}}$$

# Low-Level Computation

- These are the results of an experiment conducted about five years ago in which 'ibmq\_athens' was made to generate a Bell state.

## Example

★QISKitの結果: ショット数 8192  
{ '00': 4130, '11': 4062 }



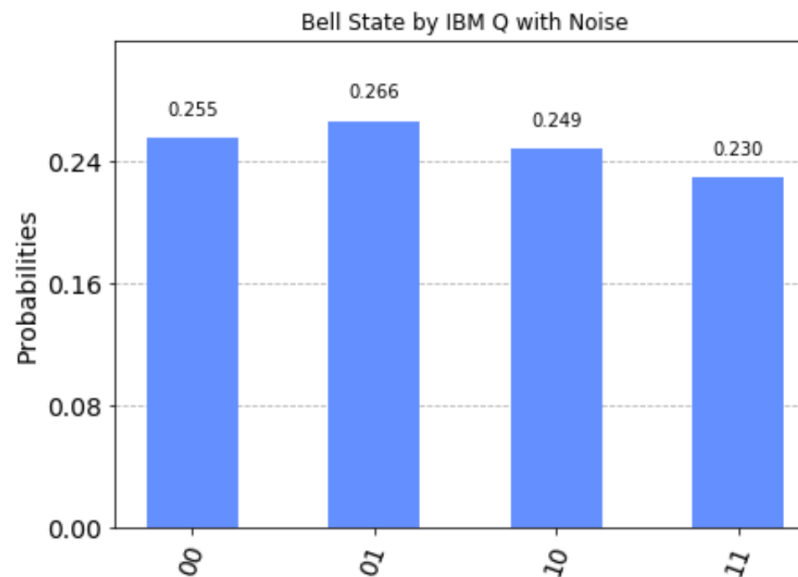
## Emulator Results

We use the backend: ibmq\_athens

★IBM Qの結果: ショット数 8192

ノイズ度 10

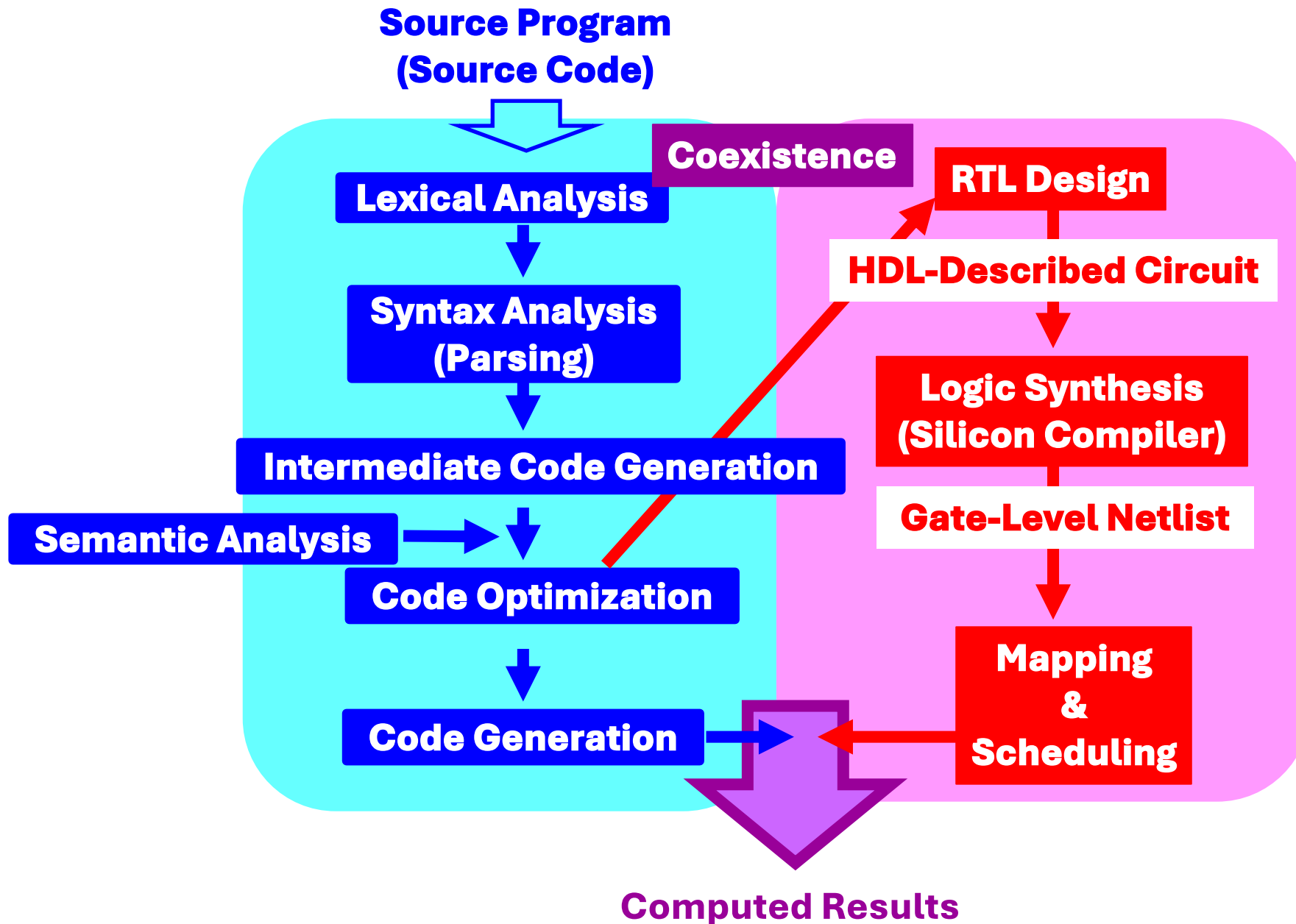
{ '00': 2091, '01': 2180, '10': 2037, '11': 1884 }



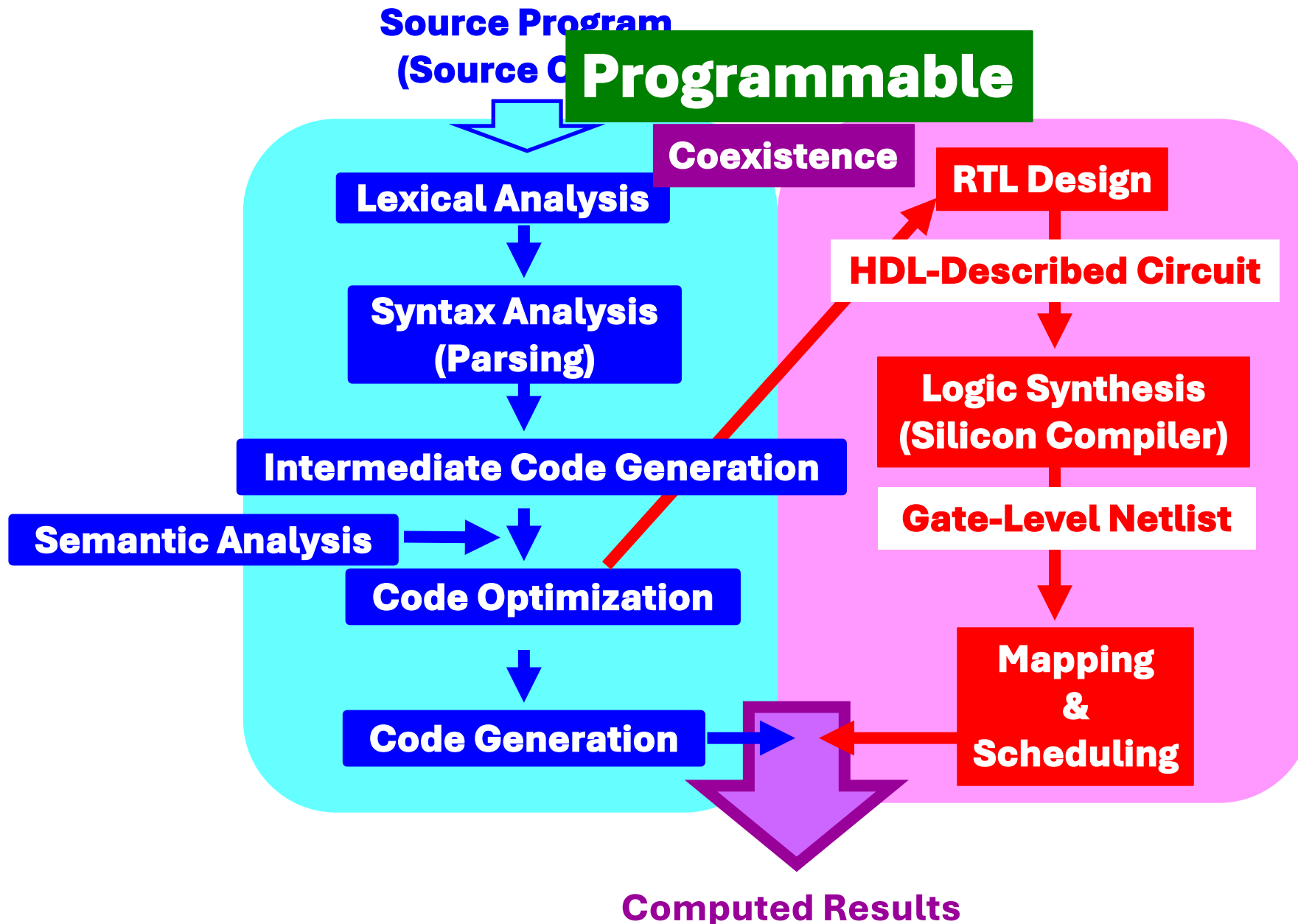
$n = 10$

$$\text{Bell State: } \frac{|00\rangle + |11\rangle}{\sqrt{2}}$$

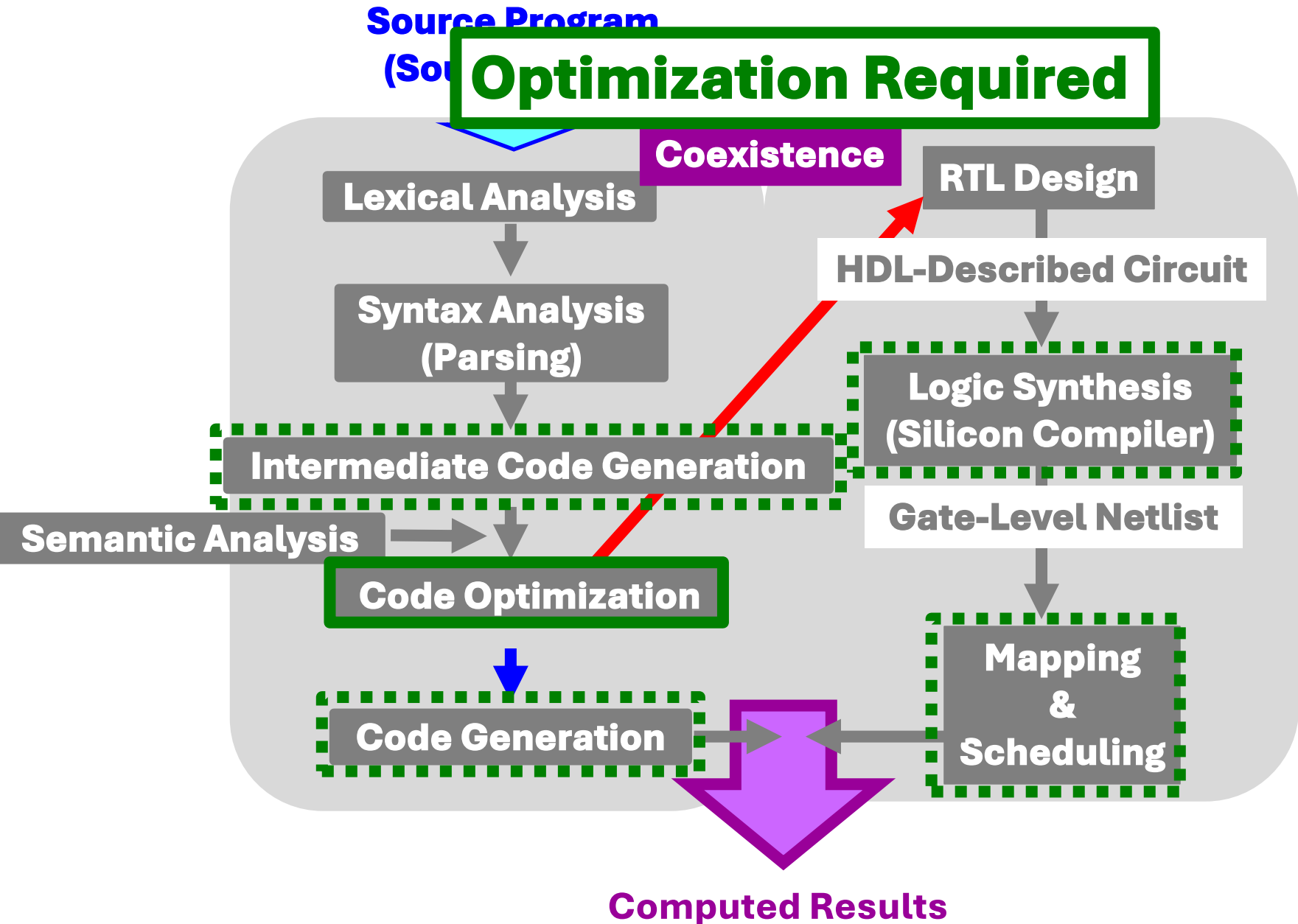
# Our Project



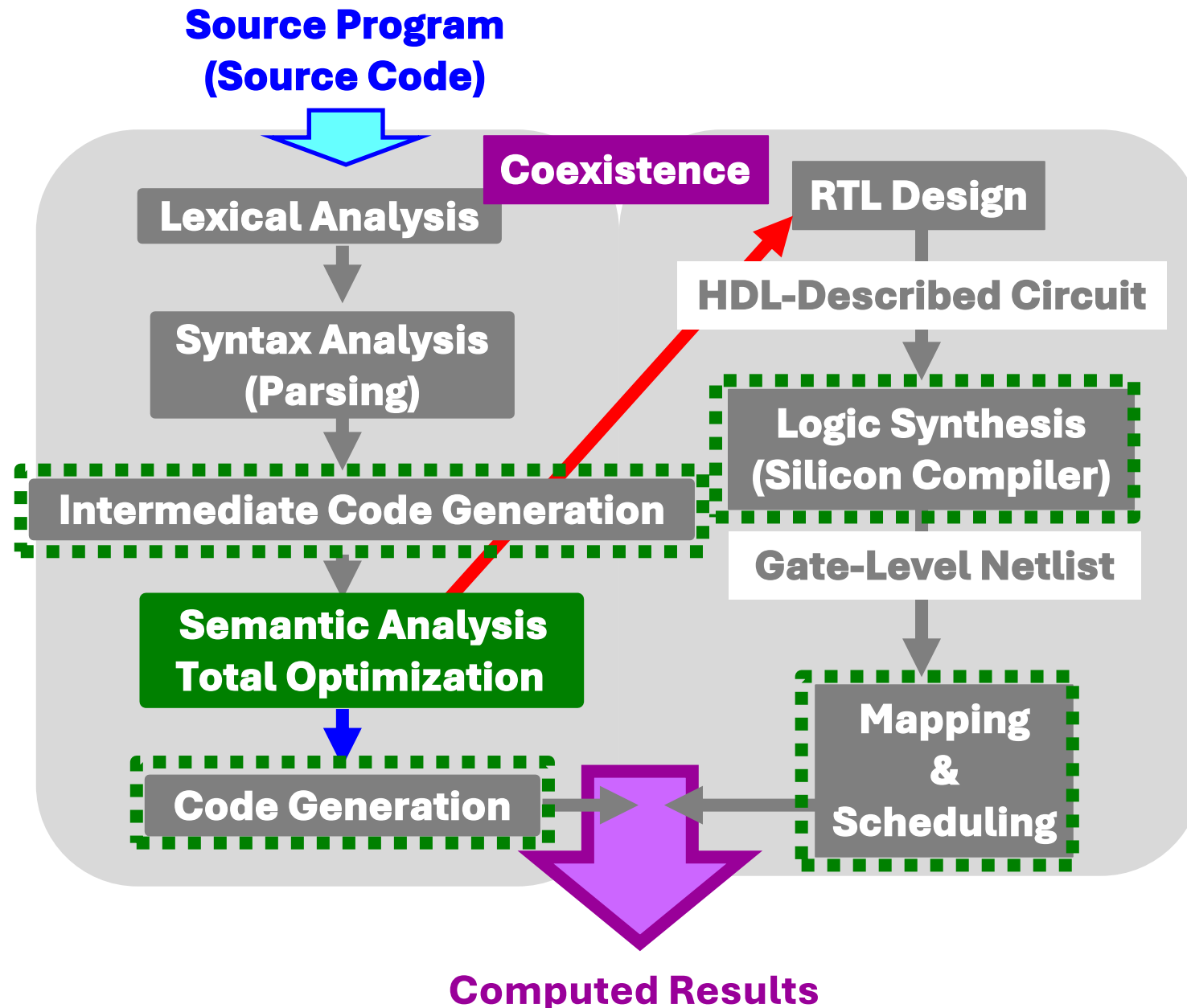
# Our Project



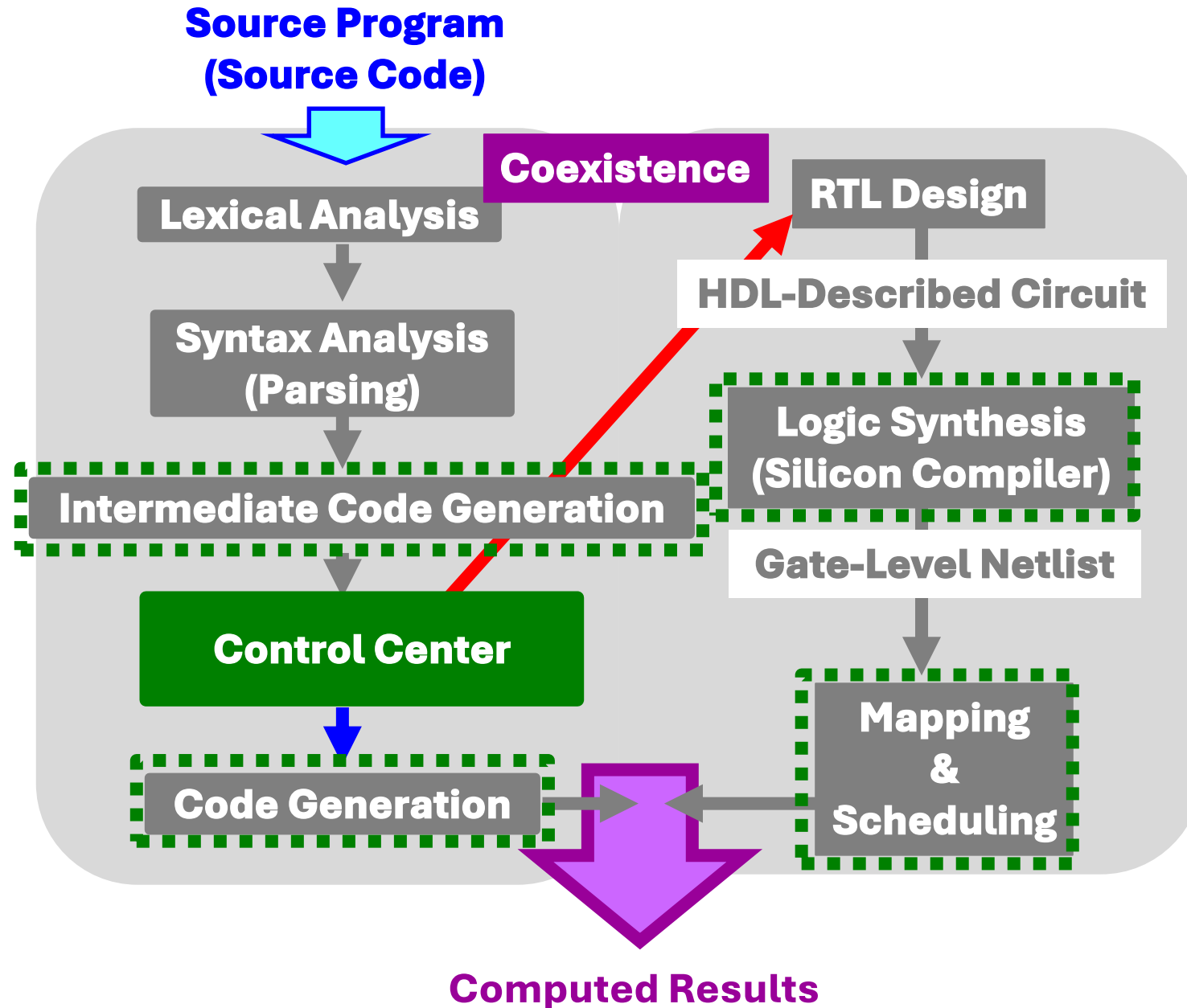
## Our Project



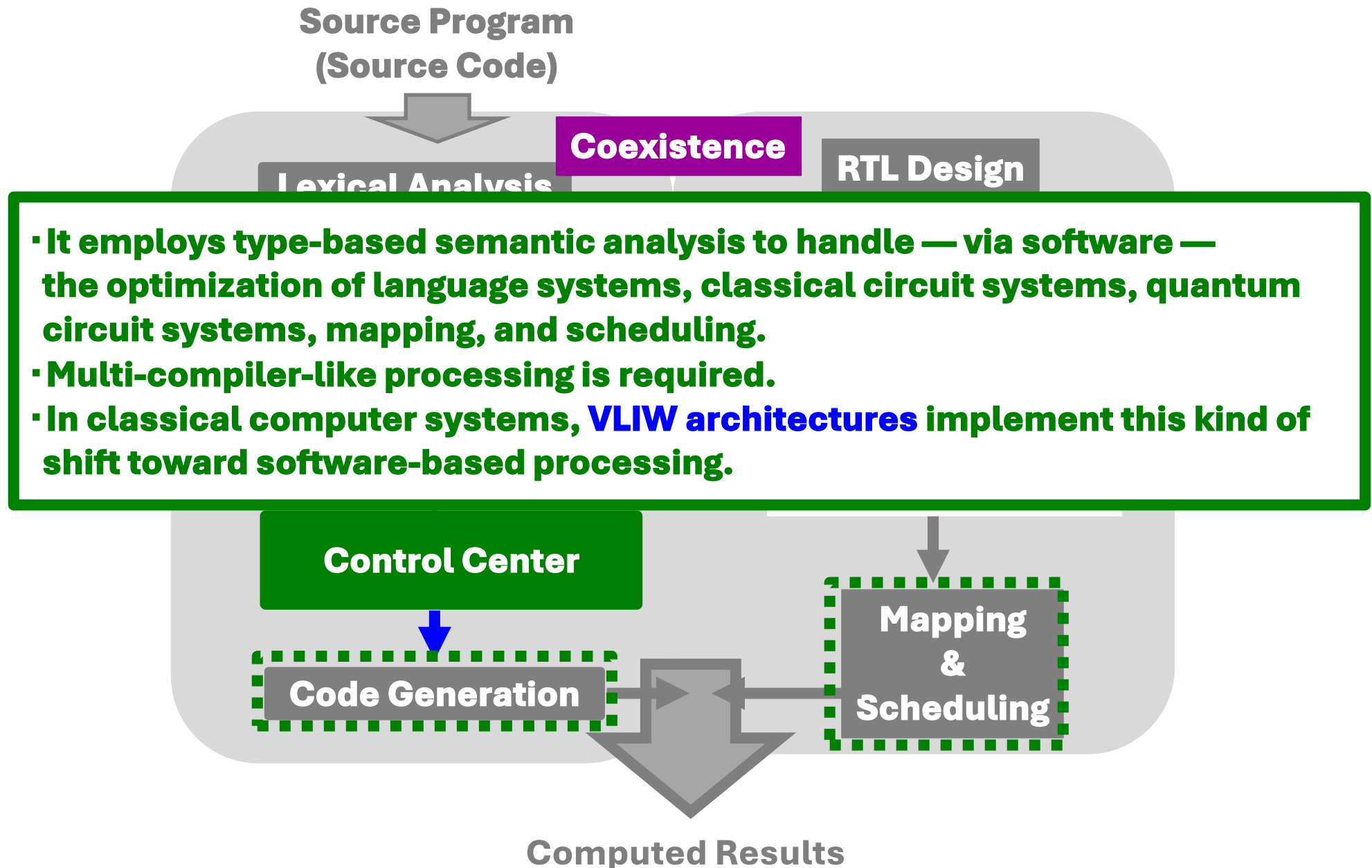
# Our Project



# Our Project



# Our Project



# Our Project

Source Program  
(Source Code)

Lexical Analysis

Coexistence

RTL Design

- It employs type-based semantic analysis to handle — via software — the optimization of language systems, classical circuit systems, quantum circuit systems, mapping, and scheduling.
- Multi-compiler-like processing is required.
- In classical computer systems, **VLIW architectures** implement this kind of shift toward software processing.

## **Very Long Instruction Word (J. A. Fisher)**

- **Digital Signal Processors and AI Chip**  
e.g. TMS320C6000 (Texas Instruments)  
Hexagon (Qualcomm)  
Tensor Processing Unit (Google)  
Ryzen (AMD)  
GroqCard™ Accelerator (Groq)  
Falcon Core (NVIDIA)

# Our Project

## Control Center including Multi-Core Processor-Like Control Device

- **To address differences in individual semantics, we will enhance the semantic analysis phase and handle these differences through typing.**
- **For various optimization tasks, we isolate the portions that can be handled via software and utilize classical computing resources, thereby reducing the need to access quantum devices.**
- **Taking into account the capabilities of quantum processors, we examine the mathematical aspects of quantum-classical hybrid distributed computing.**

some Quantum Processors



# Our Project

## Control Center including Multi-Core Processor-Like Control Device

- To address differences in individual semantics, we will enhance the semantic analysis phase and handle these differences **through typing**.

**It is possible to incorporate type theory into quantum Hoare logic or to consider a quantum version of Martin-Löf type theory:**

- **Quantum Hoare Type Theory,**
- **Linear Dependent Type Theory,**
- **Quantum Linear Homotopy Type Theory,**
- **Quantitative Type Theory.**

1

relay

2

relays

$n-1$

relay

$n$

# Our Project

## Control Center including Multi-Core Processor-Like Control Device

- **To address differences in individual semantics, we will enhance the semantic analysis phase and handle these differences through typing.**
- **For various optimization tasks, we isolate the portions that can be handled via software and utilize classical computing resources, thereby reducing the need to access quantum devices.**
- **Taking into account the capabilities of quantum processors, we examine the mathematical aspects of quantum-classical hybrid distributed computing.**

some Quantum Processors



# Our Project

## Control Center including Multi-Core Processor-Like Control Device

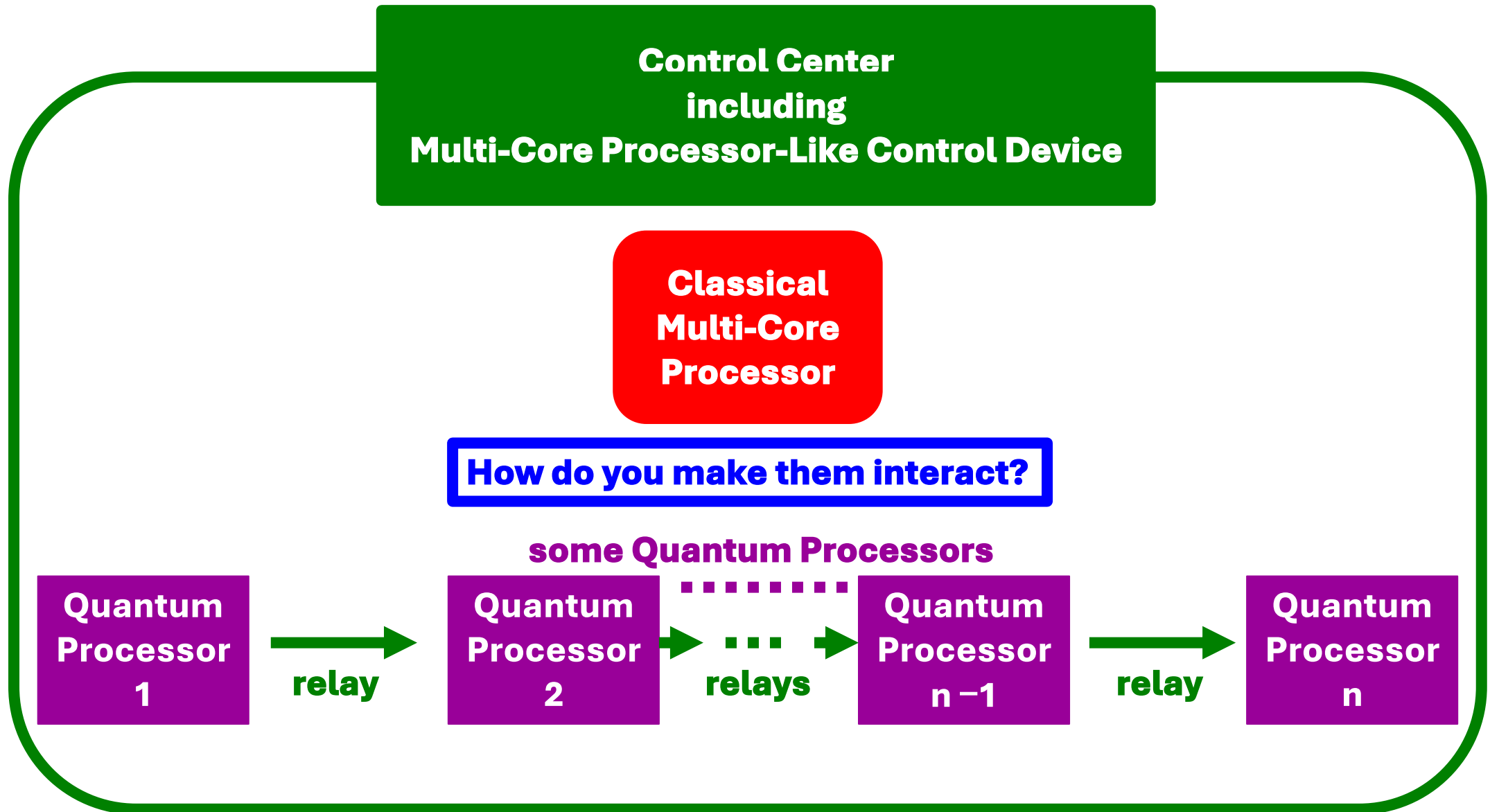
- **To address differences in individual semantics, we will enhance the semantic analysis phase and handle these differences through typing.**
- **For various optimization tasks, we isolate the portions that can be handled via software and utilize classical computing resources, thereby reducing the need to access quantum devices.**
- **Taking into account the capabilities of quantum processors, we examine the mathematical aspects of quantum-classical hybrid distributed computing.**

some Quantum Processors



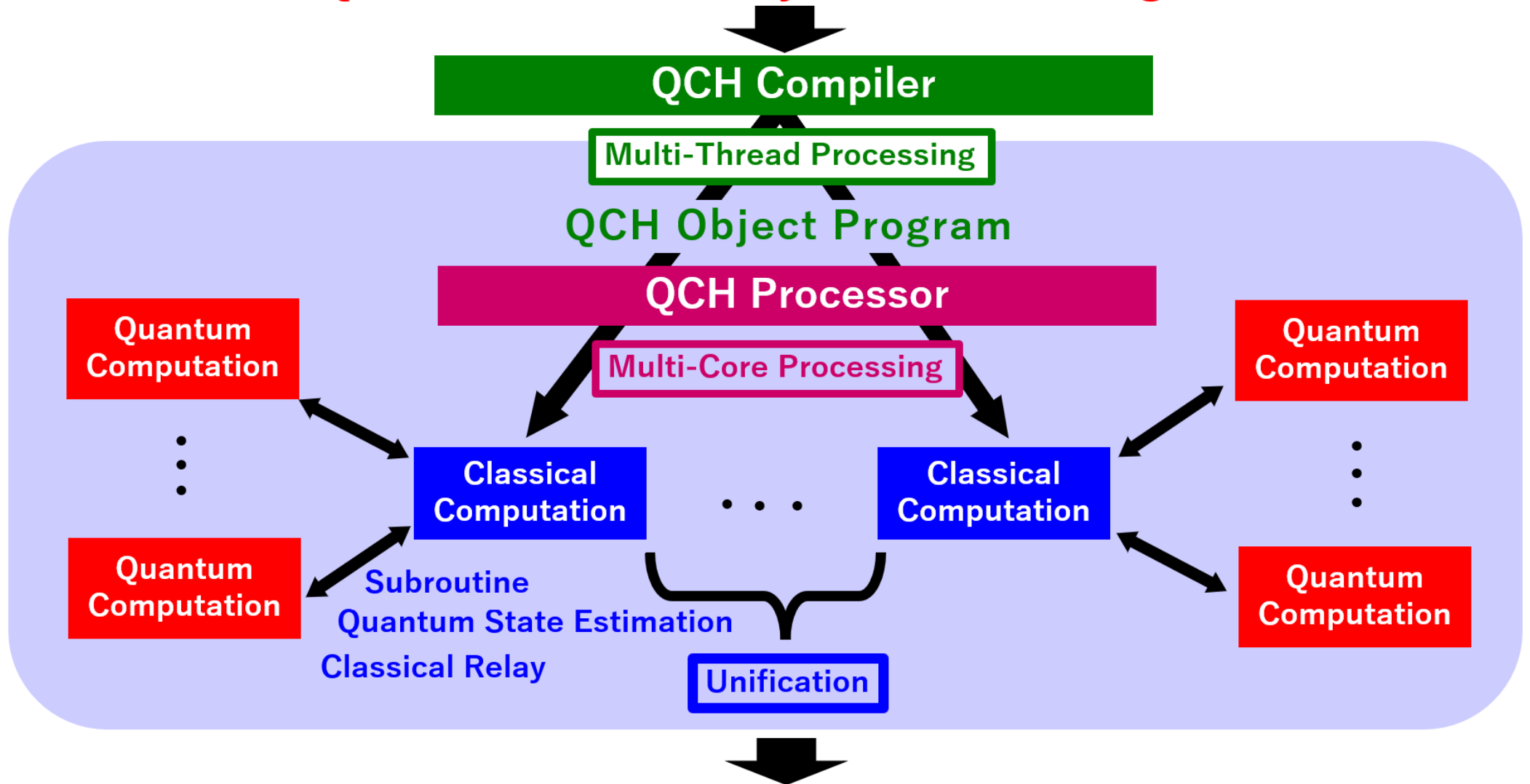
# Our Project

The following function has been added to Mapping & Scheduling.



# Our Project

## Quantum-Classical Hybrid Source Program



# Our Project

## Design and Implementation of a Language and Its Processing System

- S. Arakaki, M. Hirokawa, and H. Watanabe, “**Toward Design and Implementation of a Quantum-Classical Hybrid Computing System,**”  
International Conference on Quantum Software (IQSoft 2025), Bilbao, Spain, 10-12 Jun, 2025.
- S. Arakaki and M. Hirokawa, “**Foundational Design of Multi-Modal Typed Programming Language for Quantum-Classical Hybrid Computing System,**”  
International Conference on Artificial Intelligence, Computer, Data Sciences and Applications (ACDSA 2026), Boracay Island, Philippines, 5-7 Feb, 2026.
- S. Arakaki and M. Hirokawa, “**Interpreter Implementation of a Multi-Modally Typed Staged Language for Quantum-Classical Hybrid Distributed Execution**”  
(submitted to IPSJ-SIGPRO).

## Separation of Quantum Error Detection from Quantum Error Correction

- T. Hinokuma, “**Development of a Surface Code Decoder Robust to Non-Stationary Errors Using Anomaly Detection via Machine Learning**” (**master's thesis**)

## Optimization of Mapping and Scheduling in Quantum Processors

- K. Takayama, A. Soeda, Y. Suzuki, I. Byun, Y. Ueno, M. Hirokawa, T. Tanimoto, “**Error-Capped Gate Scheduling for Parallelism-Induced Errors in Superconducting Quantum Processors**” (submitted)

**Thanks!**

**Thank you for your attention.**