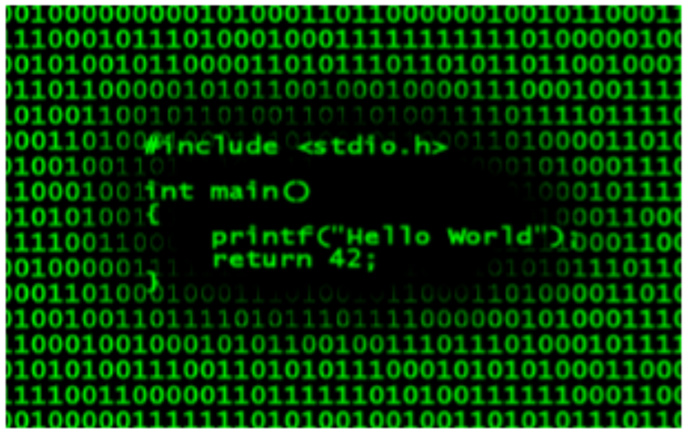
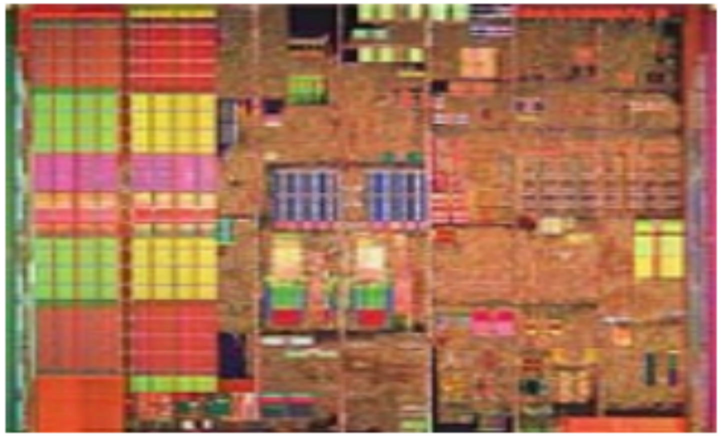
Support of Cross Calls between Microprocessor and FPGA in CPU-FPGA Coupling Architecture



```
#include <stdio.h>
int main()
{
   printf("Hello World");
   return 42;
}
```

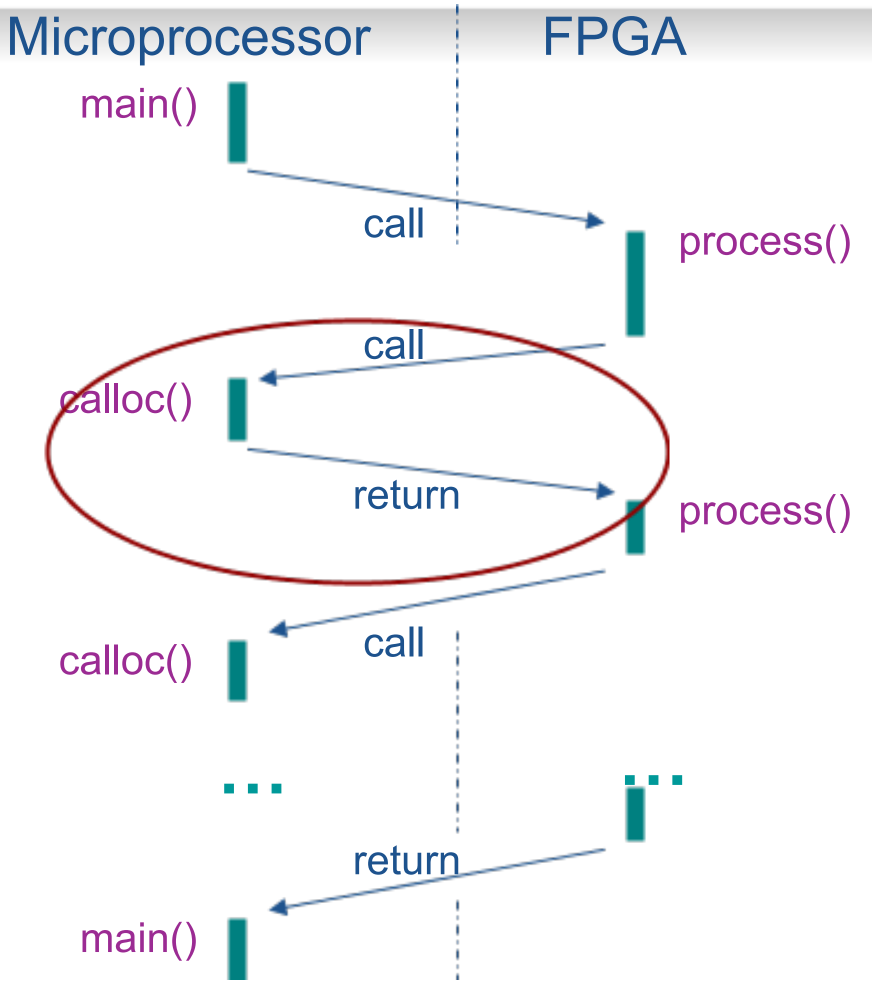


G. NguyenThiHuong and Seon Wook Kim

Microarchitecture and Compiler Laboratory
School of Electrical Engineering
Korea University

Motivation

```
void process (struct data* head)
{
struct data* p;
int ret = 0;
for( p = head; p; p = p->next){
p->content = (struct elem*) calloc (p->size);
if( !p->content ){
ret = 1;
break;
} else{
…..
}
}
return ret;
}

struct data* head;
int main (void) {
…..
error = process (head);
…..
}
```



Many code sections are executed more efficiently in microprocessor: floating intensive codes, system calls, memory management functions, etc.

To support codes containing these functions in FPGA, the FPGA should be able to call back to microprocessor as a master component.

KOREA UNIVERSITY

Previous work

- Away from code coordination between CPU and FPGA
  - Handel-C, Impulse C
  - OCPIP, AMBA
- Support nested and recursive only in hardware side
  - ASH (M. Budiu – ASPLOS ‘04), HybridThreads (E. Anderson-ERSA ‘07)
  - Do not allow hardware to call software
- Allows hardware to return back to software for software code execution
  - Comrade (H. Lange-FPL ‘07)
  - Do not support communication among compute units in FPGA

No work to support the cross calls between SW and HW without any limitation!

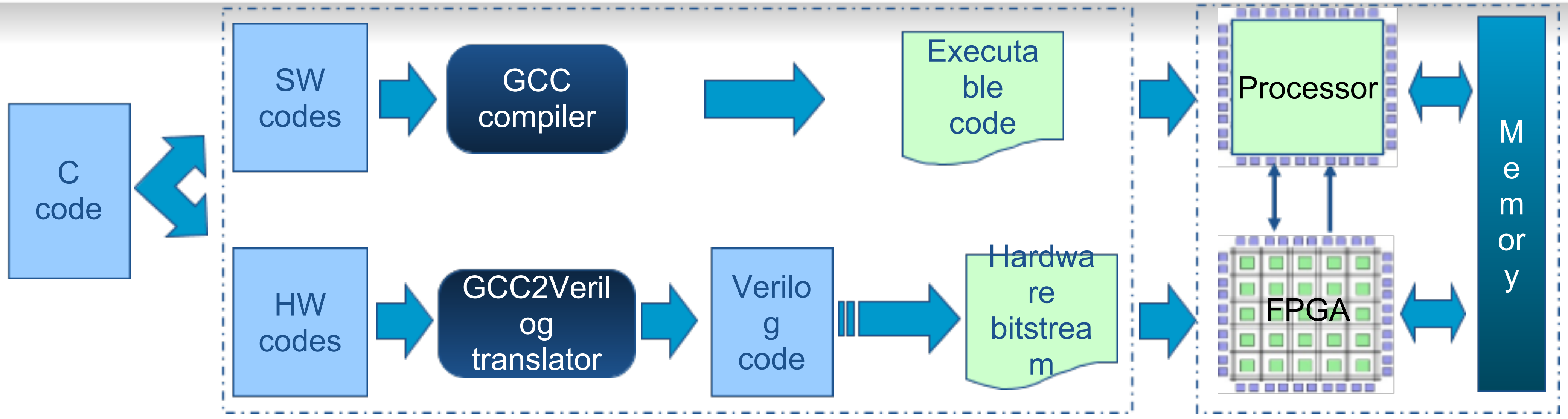
GCC2Verilog approach

- GCC2Verilog: A C-to-Verilog translator based on GCC compiler
  - Including a Verilog backend to generate Verilog code from GCC's RTL
- Making hardware follows software calling convention
  - Software and hardware share one stack space.
    - Arguments passing through argument registers and stack.
  - Preserve software stack layout when performing calls in hardware side.
- Supporting:
  - Unlimited nesting calls in hardware including recursive calls.
  - Unlimited nesting cross calls between software and hardware.

Any hardware function in FPGA can be a master in the system!

Contents

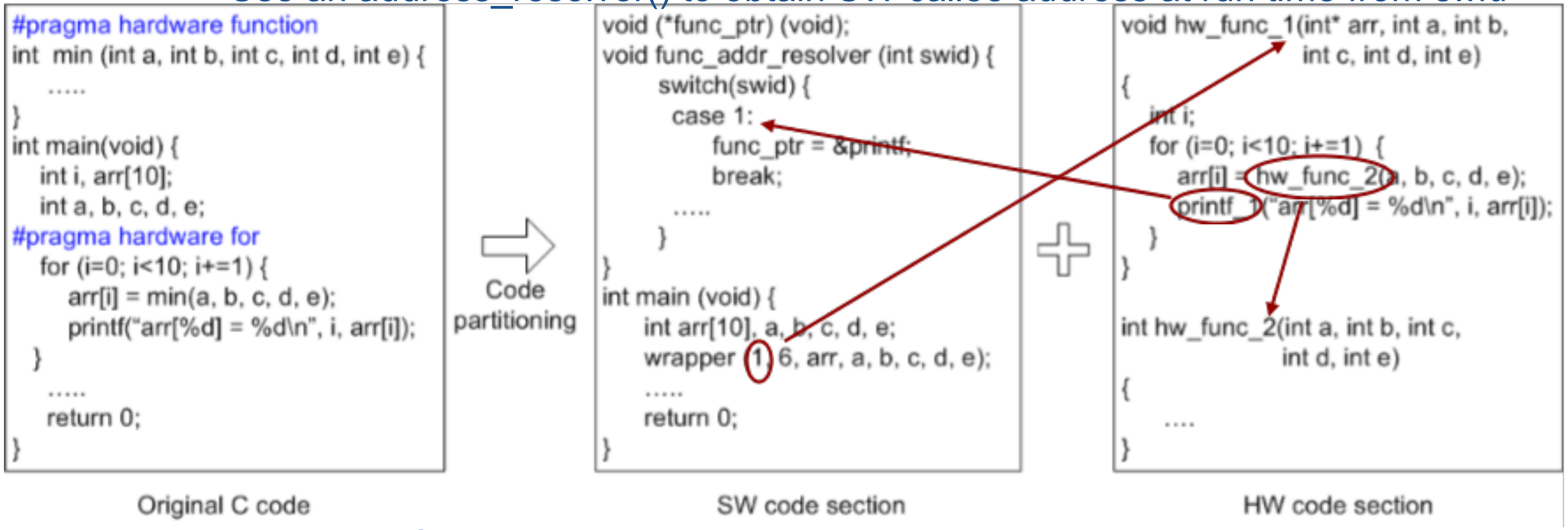
GCC2Verilog: Compilation & Execution Model



- Code partitioning process:
  - Divides codes into hardware and software sections
  - Prepares the address resolution
- Compilation process:
  - Compiles software code section into executable objects
  - Translates hardware code section into Verilog code and synthesizes them to HW bitstreams (HWIPs).
- Execution process:
  - Running SW executable code in a microprocessor & HWIPs in FPGA
  - The FPGA communicates with the host processor through a communication channel and memory.

Address Resolution

- Hardware address resolution:
  - Assigning an hardware identification number *hwid* to each HWIP
- Software address resolution:
  - Static link: use the symbol table obtained an executable file to resolve software addresses at HLL-to-HDL translation.
  - Dynamic link:
    - Assign an identification number *swid* to each SW callee called from HW
    - Use an address_resolver() to obtain SW callee address at run time from *swid*



```
#pragma hardware function
int  min (int a, int b, int c, int d, int e) {
    .....
}
int main(void) {
   int i, arr[10];
   int a, b, c, d, e;
#pragma hardware for
   for (i=0; i<10; i+=1) {
      arr[i] = min(a, b, c, d, e);
      printf("arr[%d] = %d\n", i, arr[i]);
   }
   .....
   return 0;
}
```

```
void (*func_ptr) (void);
void func_addr_resolver (int swid) {
      switch(swid) {
        case 1:
            func_ptr = &printf;
            break;
        .....
      }
}
int main (void) {
    int arr[10], a, b, c, d, e;
    wrapper (1, 6, arr, a, b, c, d, e);
    .....
    return 0;
}
```

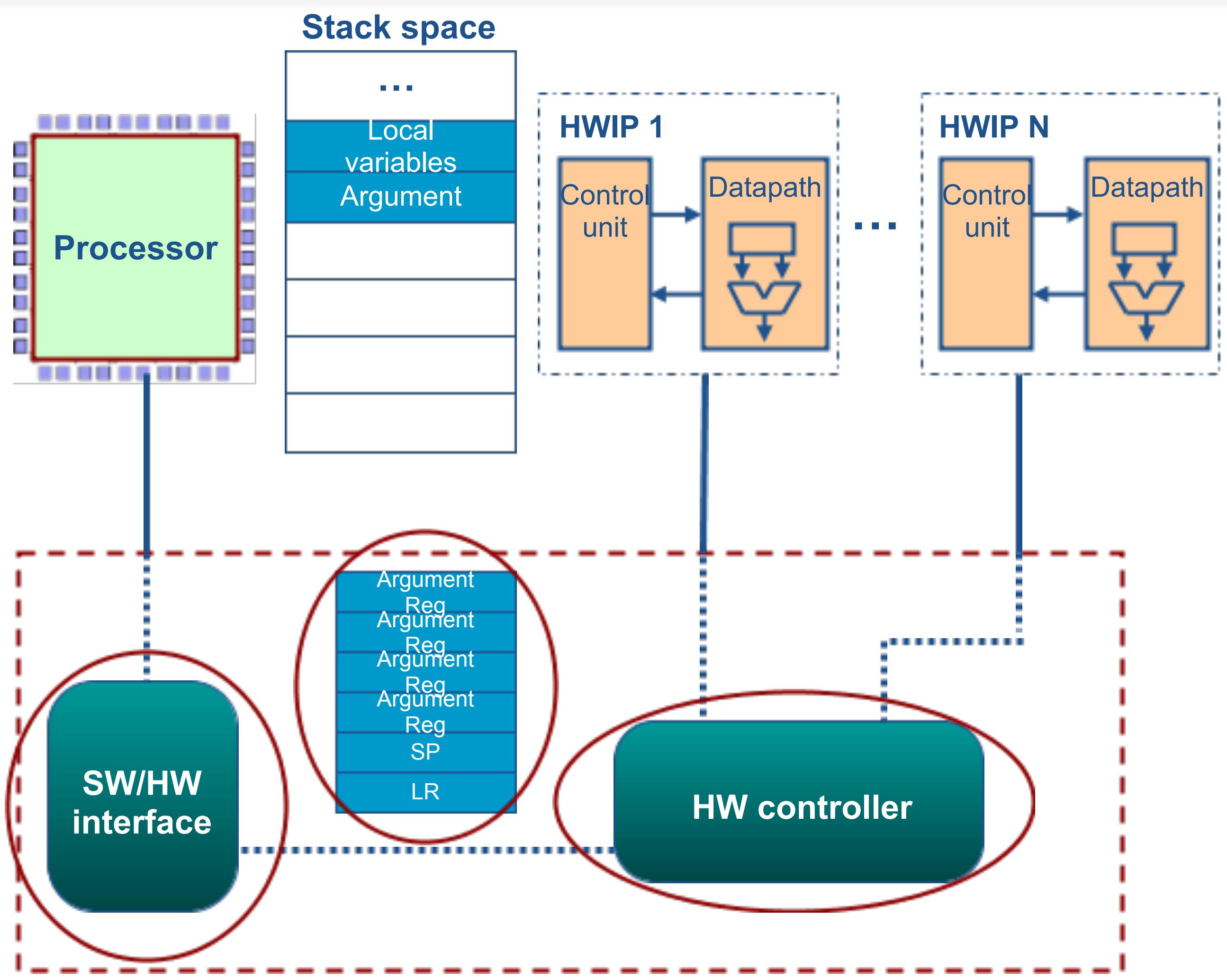
```
void hw_func_1(int* arr, int a, int b,
                int c, int d, int e)
{
   int i;
   for (i=0; i<10; i+=1) {
      arr[i] = hw_func_2(a, b, c, d, e);
      printf_1("arr[%d] = %d\n", i, arr[i]);
   }
}

int hw_func_2(int a, int b, int c,
              int d, int e)
{
   ....
}
```

SW address resolution in dynamic linking

KOREA UNIVERSITY

Additional Components

- HW controller:
  - Controls and schedules the execution between a processor and HWIPs
- SW/HW interface:
  - Provides a uniform interface to communicate with the host processor
- HW register set: set of registers for calls:
  - Argument registers
  - HW stack pointer
  - Link register

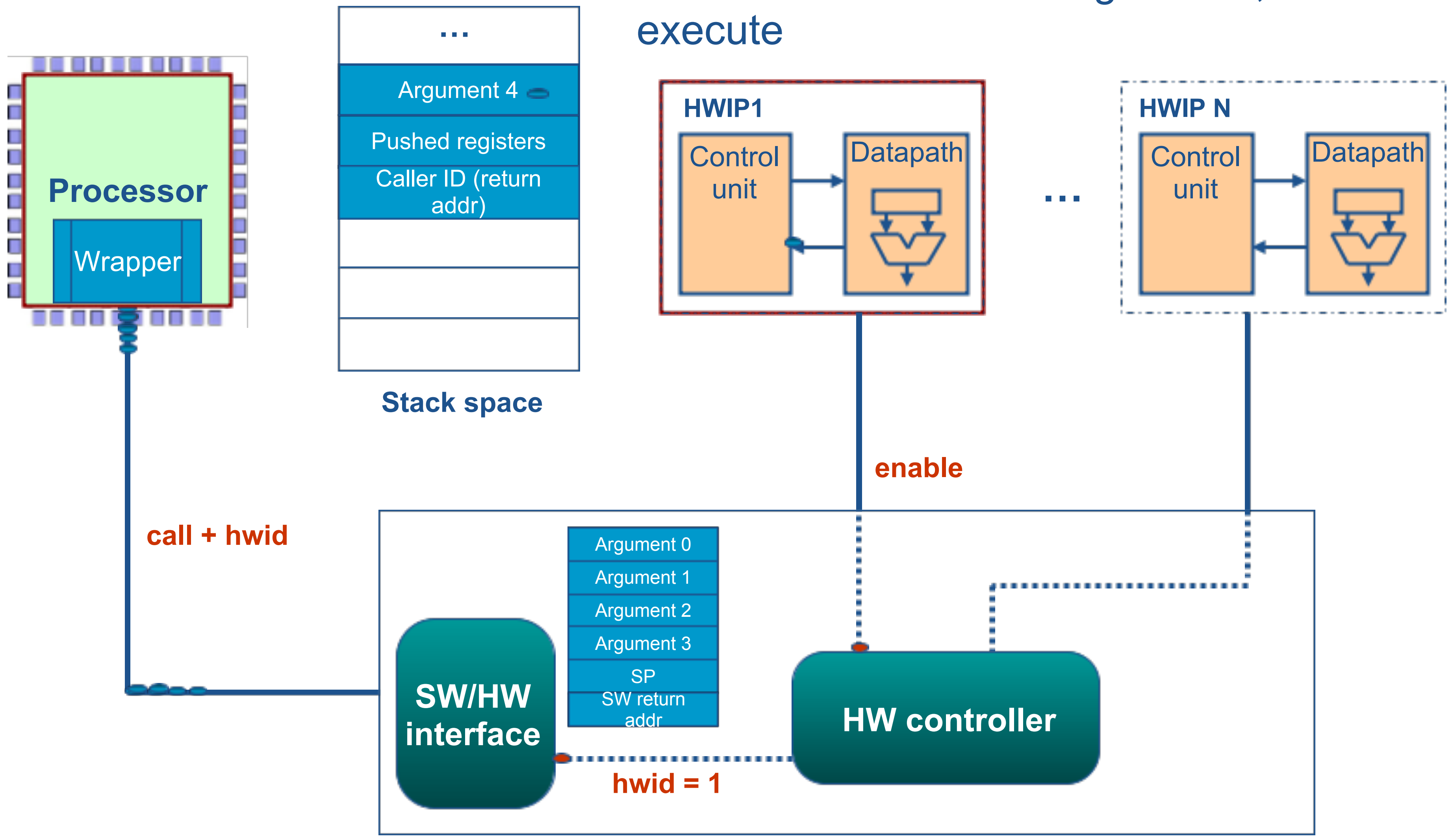


Software Calls Hardware

1. The wrapper function passes arguments, and calls the HW callee

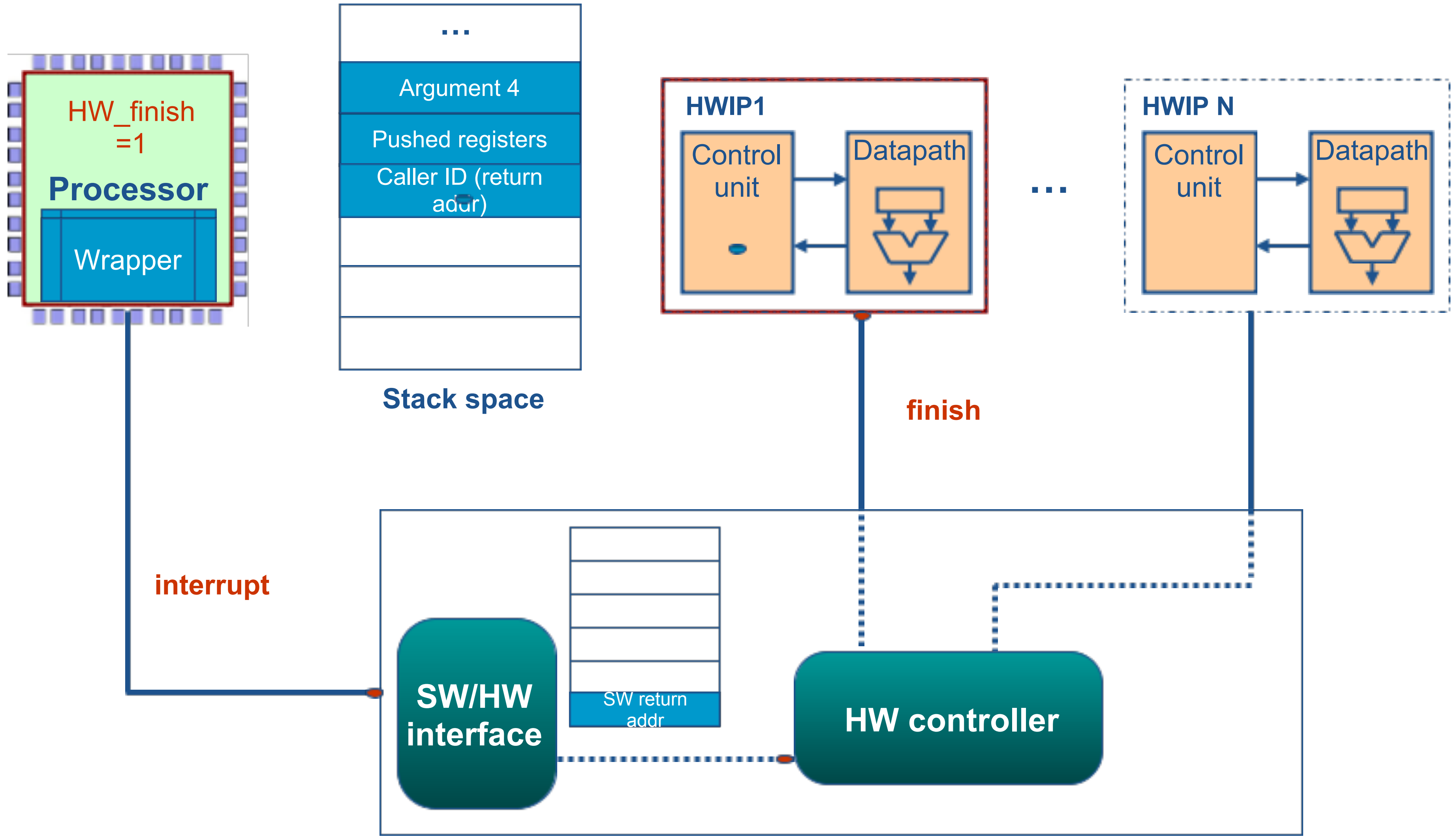
2. HW controller enables the HW callee

3. HW callee reads its arguments, and starts to execute



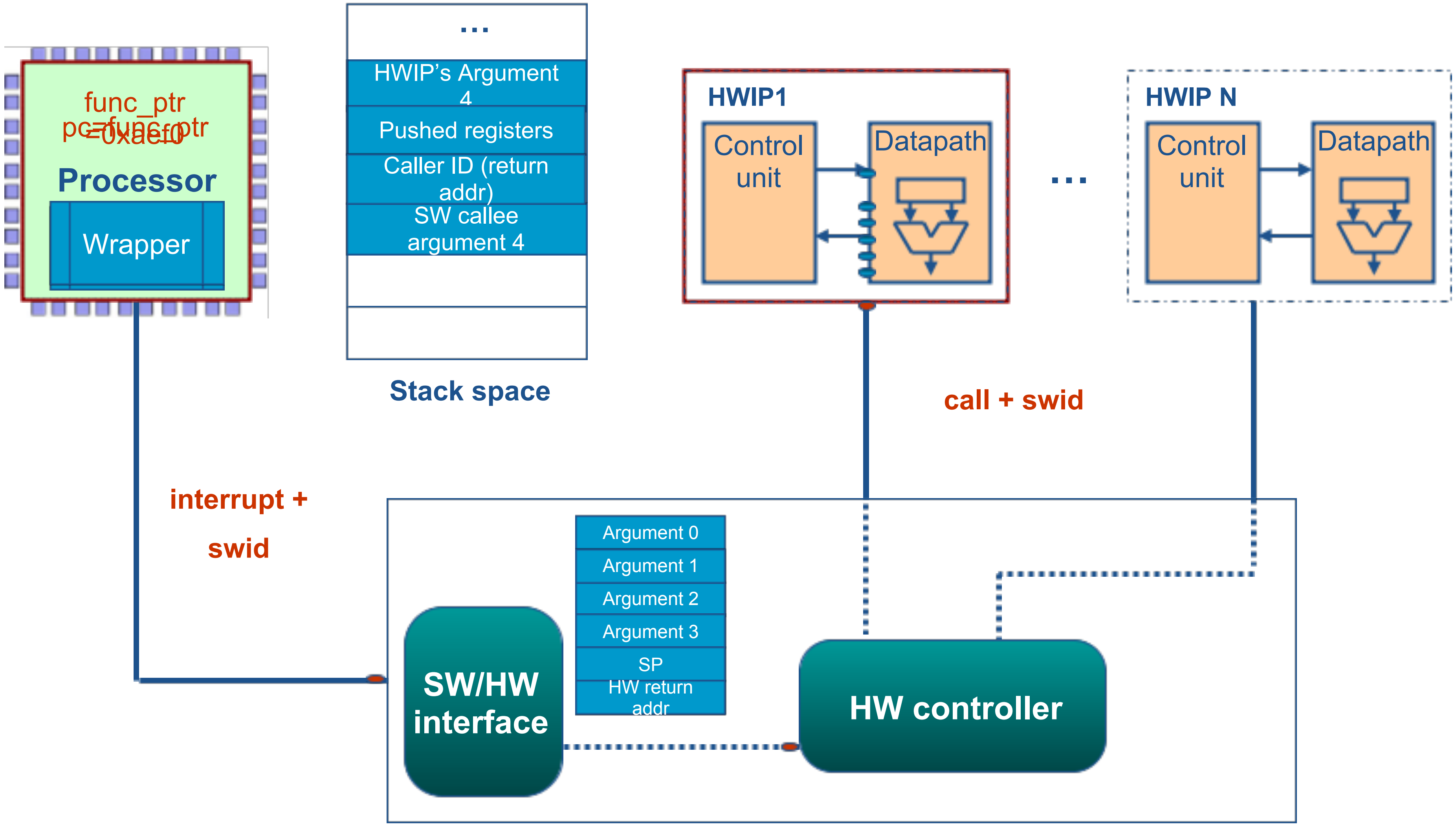
Hardware Callee Returns to Software Caller

4. HW controller interrupts the host processor when the HW callee finishes
5. The interrupt handler notifies the HW finishing to the wrapper



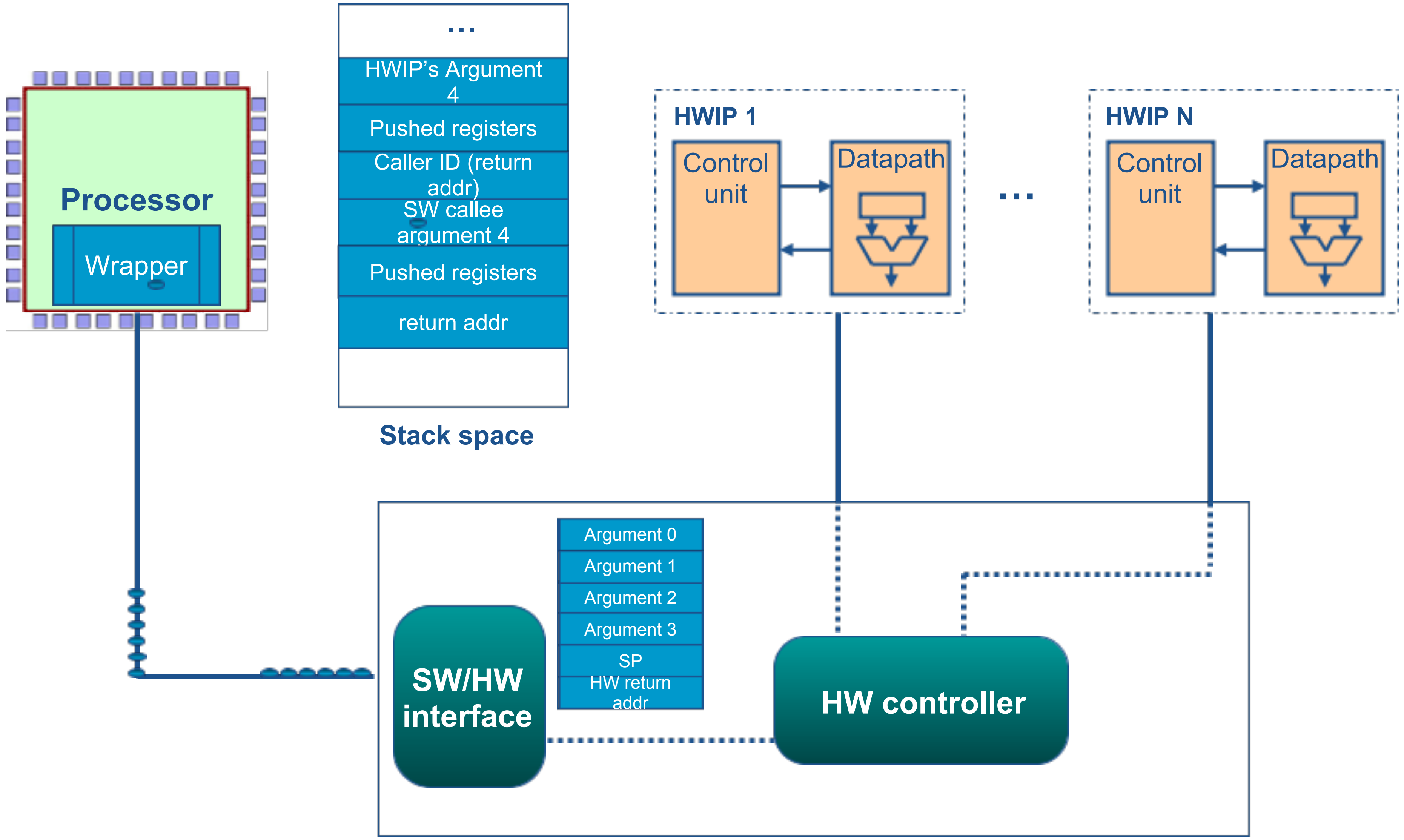
Hardware Calls Software

1. HW caller passes arguments and notifies to the controller about the call
2. HW controller interrupts the processor with SW callee ID
3. The interrupt handler resolves the SW callee's actual address from swid & the wrapper calls the function.



Hardware Calls Software

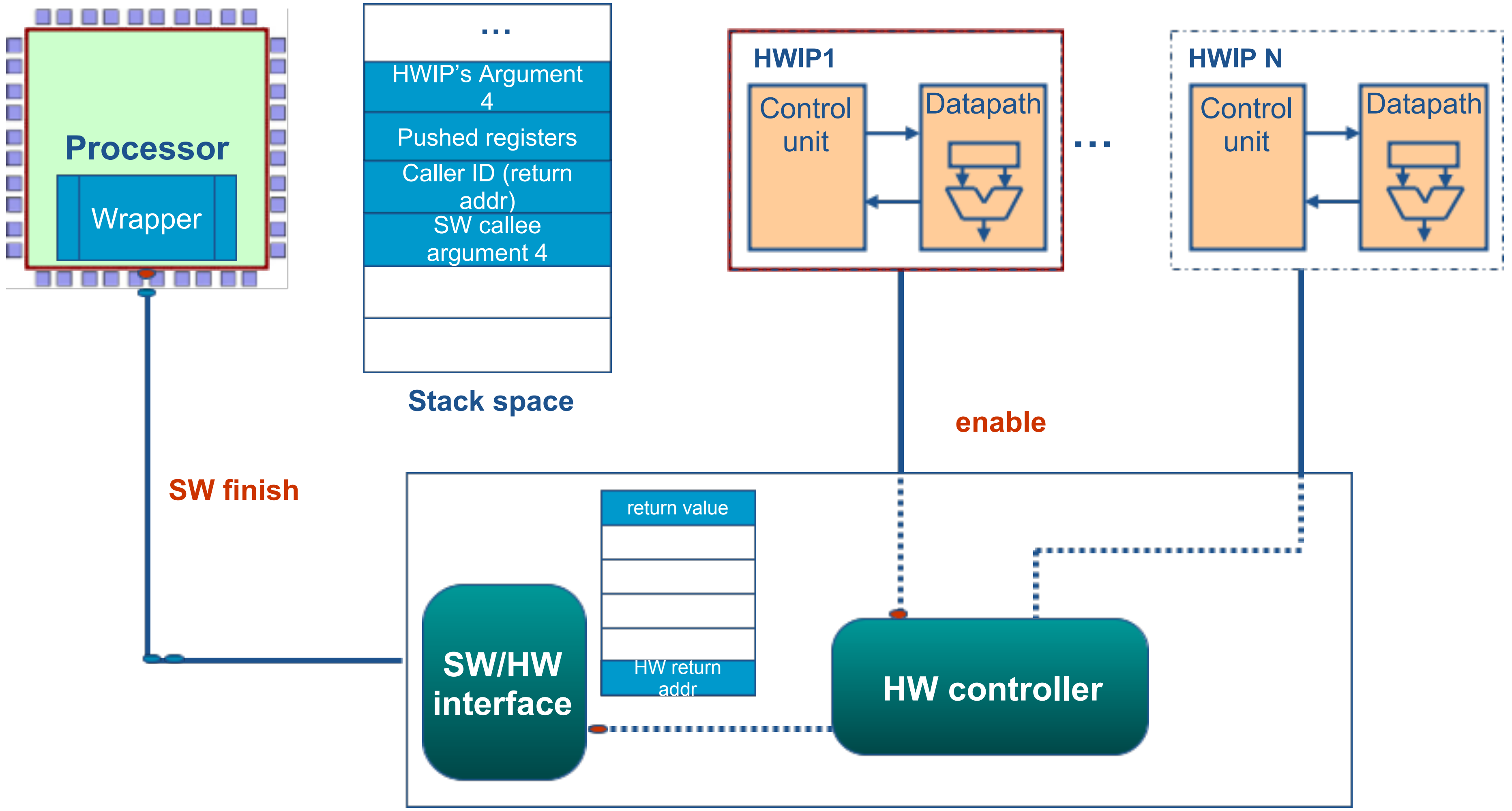
4. SW callee executes its code & returns to the wrapper when finish



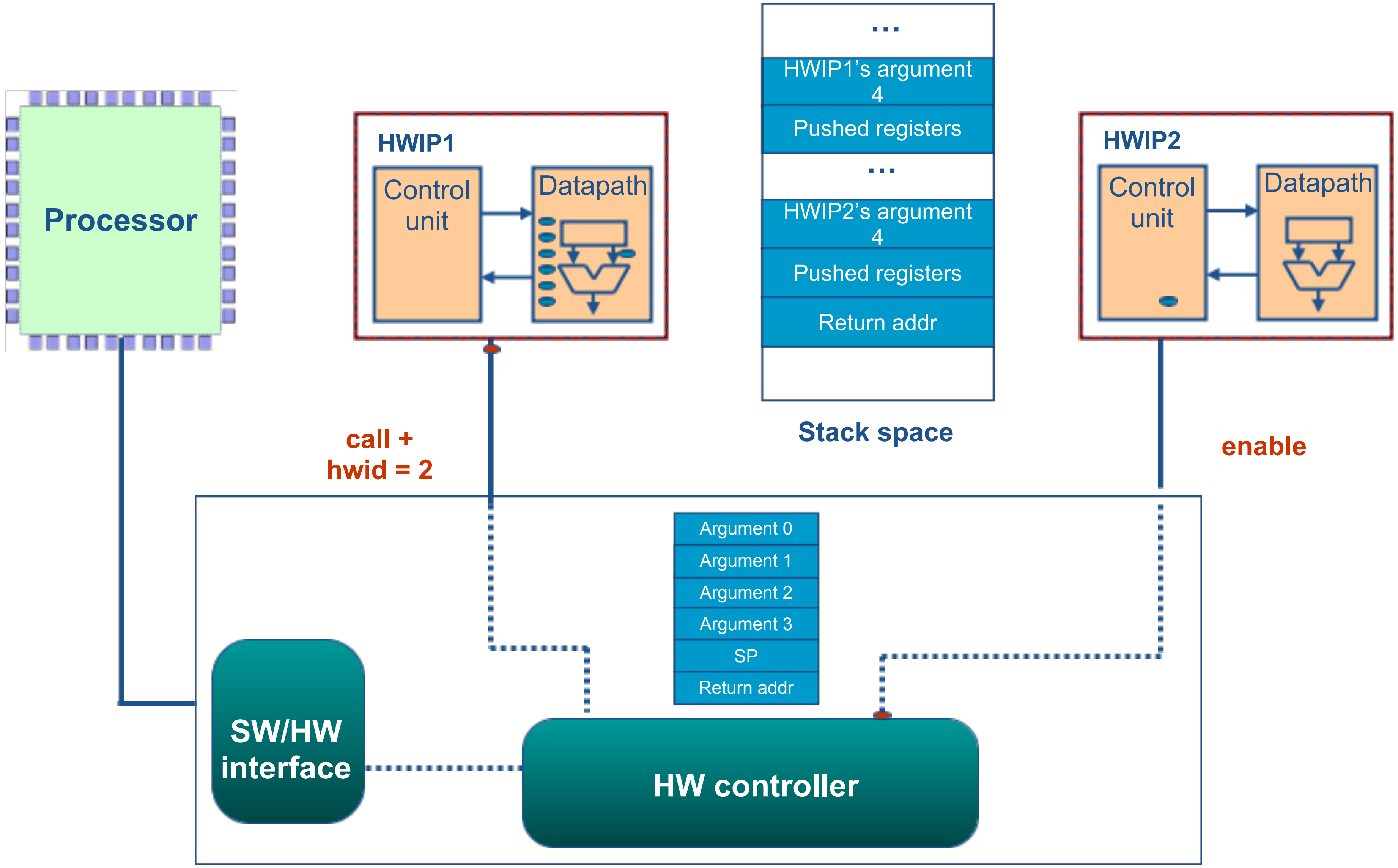
KOREA UNIVERSITY

Software Callee Returns to Hardware caller

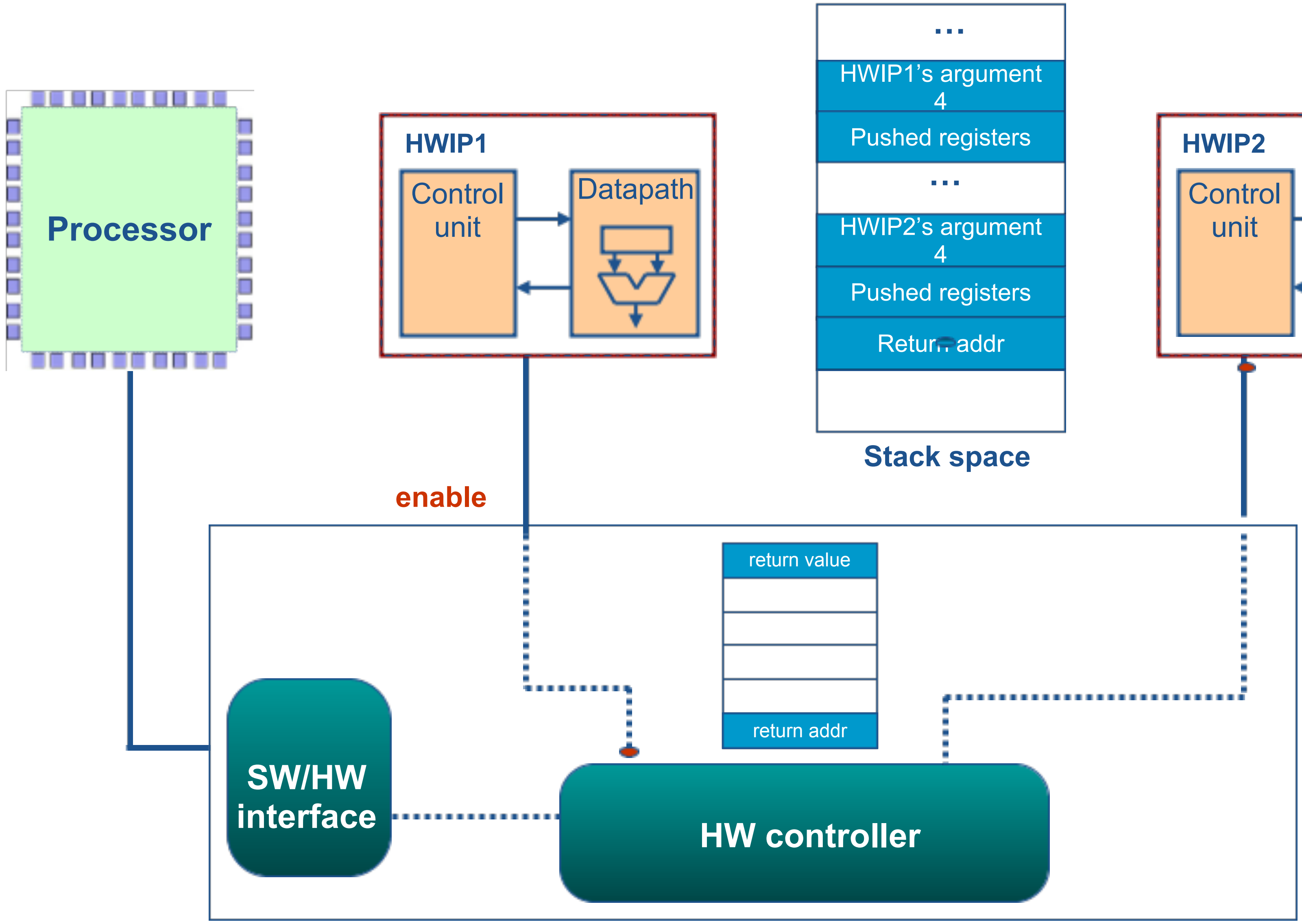
5. The wrapper notifies to HW controller about SW finish
6. The HW caller is enabled again to continue its execution



Hardware Calls Hardware



Hardware Calls Hardware



Experiment Result

- Experiment setup
  - Host processor: ARM922T
  - Benchmarks: EEMBC + factorial (recursion)
- Calling overhead:
  - Cross calls between SW and HW (exclude interrupting time)
    - Static link: 99 cycles
    - Dynamic link: 125 cycles
  - Calls among HWIPs:
    - Less than 5 cycles

Experiment Result

Benchmarks	Number of calls	Call overhead (%)
aifftr	300	3.52
aiifft	300	4.00
fft	100	2.71
bezier	20	0.11
idctrn	600	4.62
rgbyiq	10	0.02
viterb	200	8.37
autcor	100	0.05
factorial	10	19.91

Call overhead including interrupt time

Conclusion

- Novel method to fully support cross calls among microprocessor and FPGA
  - Allowing FPGA to perform calls back to a microprocessor
  - Supporting unlimited nested and recursive calls in FPGA
- Reasonable cross calling overhead
- An importance step toward the full automatic translation of HLL to HDL
- Implemented a C-to-Verilog translator based on GCC compiler

Questions & Answers