# Seamless Integration of Hardware and Software in Reconfigurable Computing Systems

**Miljan Vuletić, Laura Pozzi, and Paolo Ienne**
**{Miljan.Vuletic, Laura.Pozzi, Paolo.Ienne}@epfl.ch**
**http://lap.epfl.ch**
**Ecole Polytechnique Fédéral de Lausanne**
**The School of Computer and Communication Sciences**
**Processor Architecture Laboratory**

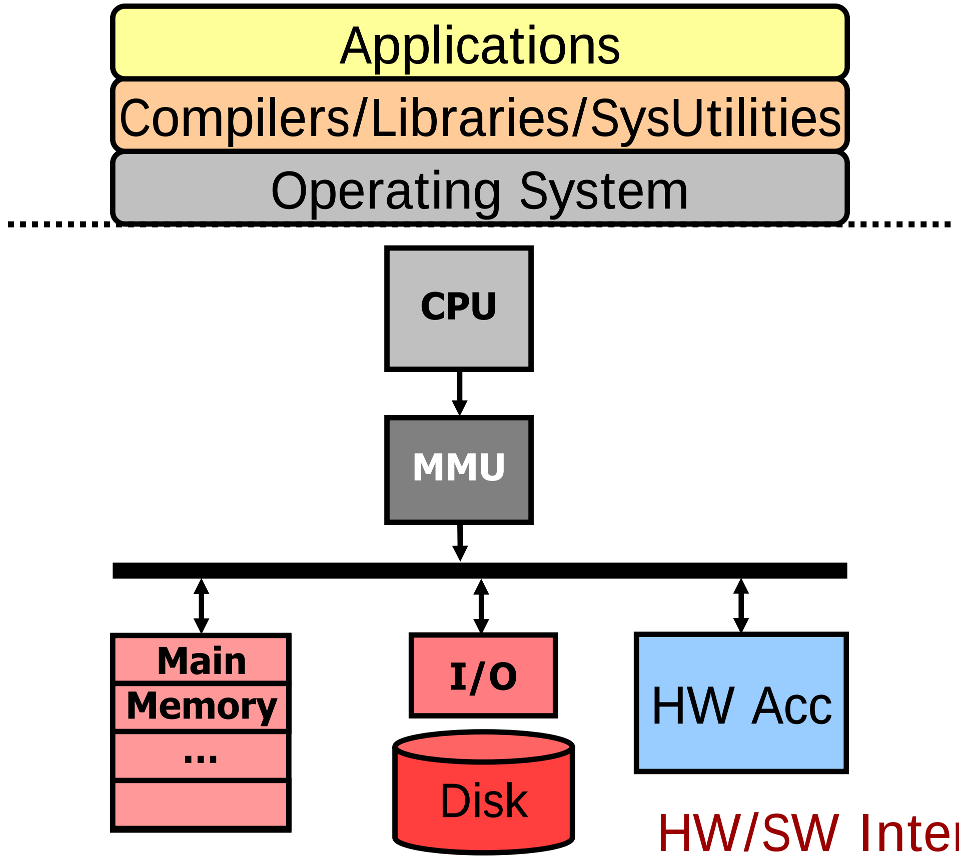
ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

## Potentials of Reconfigurable HW

- Prototyping, glue logic, acceleration
- Mask costs direct users toward FPGAs
- Reconfigurable computing mixes SW and HW:
  Exploit HW parallelism to obtain speedup!
- Major obstacles:
  Lack of portability
  Nontransparent HW/SW interfacing
  Nonstandardised programming paradigms

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

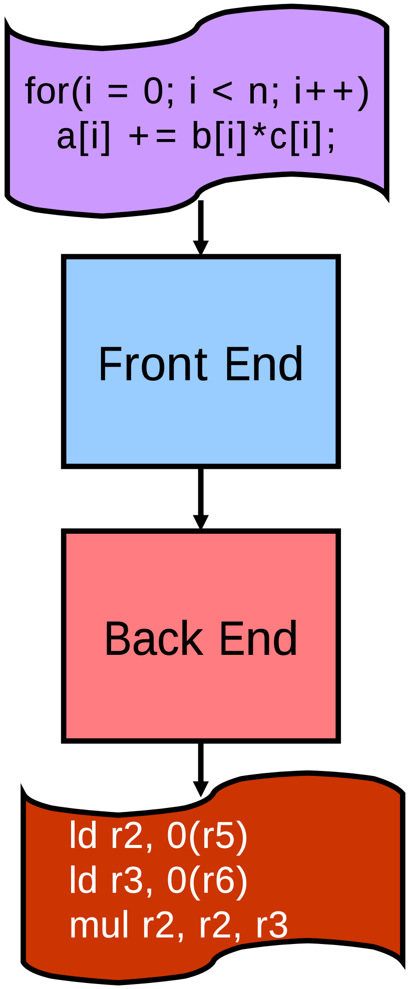
## Microprocessors: Power of programmability



- Temporal computing engines
- Different kinds of problems
- Programming instead of logic design
- Memory abstraction
- Standardised programming paradigms

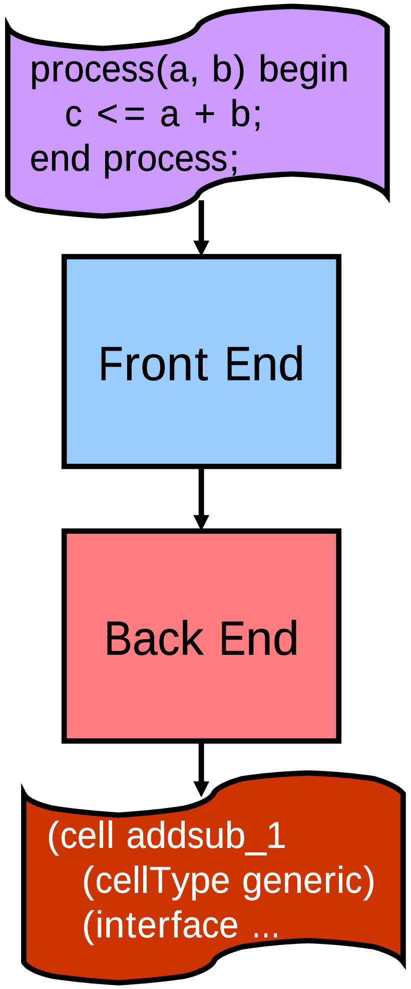
HW/SW Interfacing ?
Portability ?

# Compilers:
# Power of Portability



- Hide machine details
- Improve productivity
- Allow code portability
- Achieve good efficiency

# Compilers (Synthesizers) for Reconfigurable Hardware



- Hide technology details
- Improve productivity
- Achieve good efficiency
- But interfacing/portability?

# Reconfigurable toward General-purpose computing

- Ease of programming:
  transparent software code
- Ease of hardware design:
  platform-agnostic accelerators
- Application portability:
  recompilation and resynthesis suffice
- Abstraction:
  the price to pay should not be too high!

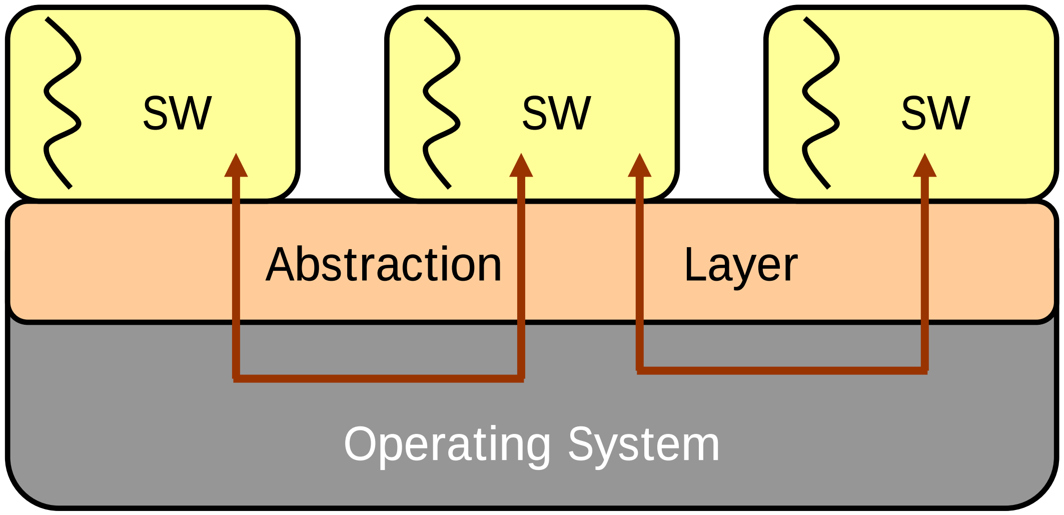
## Related Work

- Component-based design and and IP-reuse: [AMBA, VCI, Gharsalli02]
- Virtualisation of Reconfigurable Resources: [Caspi00, Dales03, Walder03]
- OS support for interfacing reconfigurable HW: [Leong01, Nollet03]
- Programming paradigms for RC: [Hauser97, Mencer01, Vassiliadis04, Brebner04]

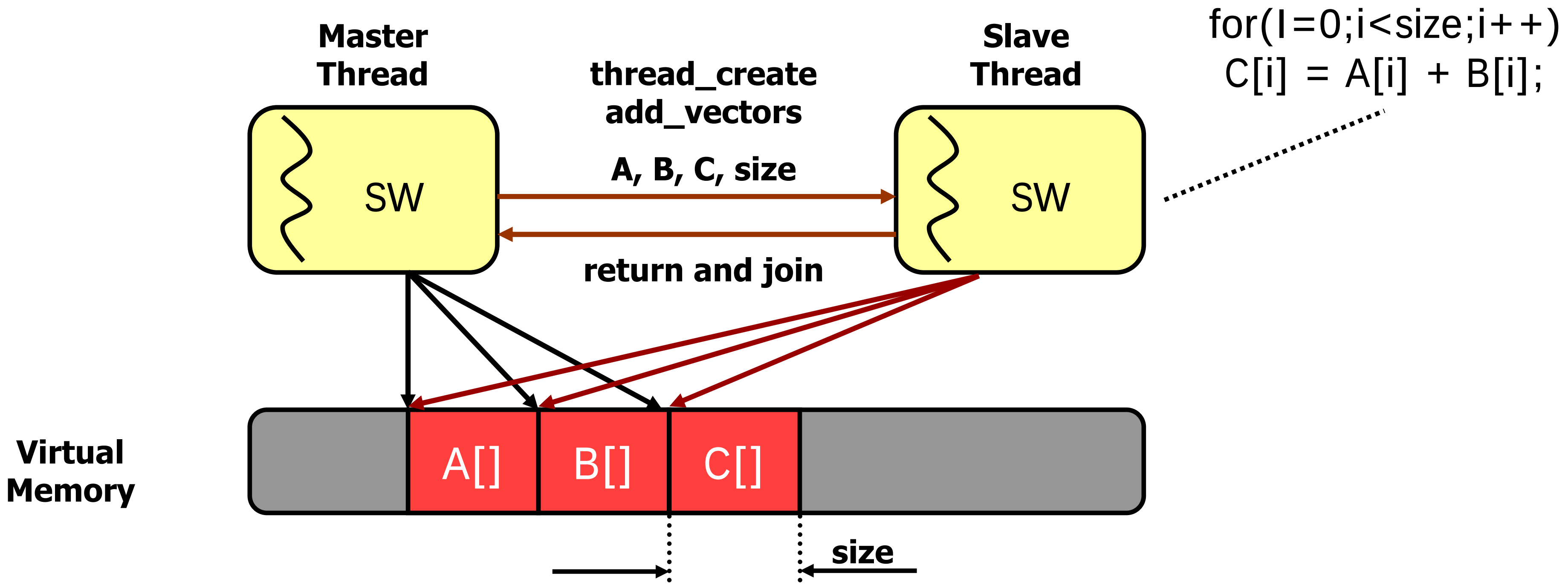
## Outline

- Introduction and Motivation
- Transparent Programming Model
  *Virtual Memory Window*
- Dynamic Prefetching
- Unrestricted Automated Synthesis
- Applications and Future Perspectives

# Standard Multithreading



# Multithreading: Memory Point of View

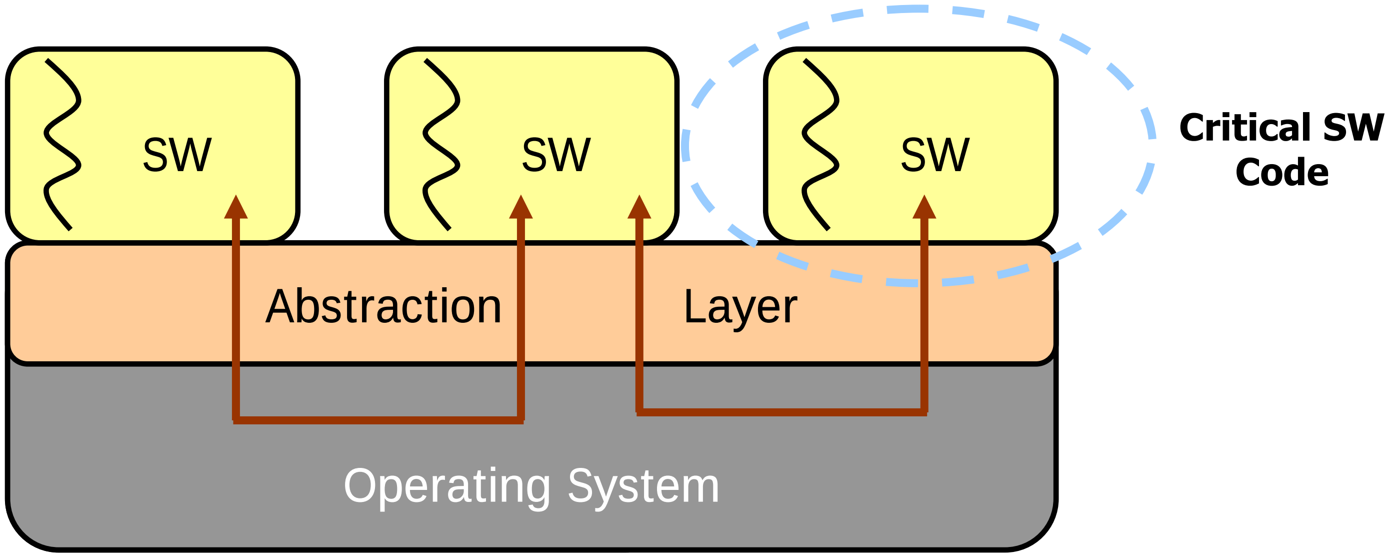


- Same memory address space
- Separate stacks

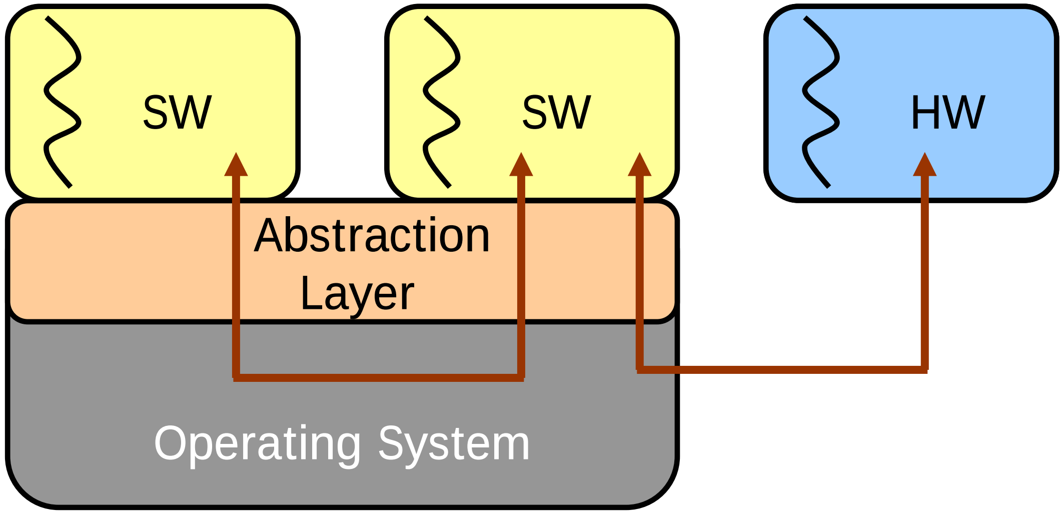
## Adding Vectors in Software

```
/* Typical SW version */

int *A, *B, *C;
int add_vectors(int *, int *, int *, int);
int thrd1;

read(A, SIZE); read(B, SIZE);

thrd1 = thread_create(add_vectors, A, B, C, SIZE);
do_some_work();
thread_join(thrd1);
```

## Extended Multithreading

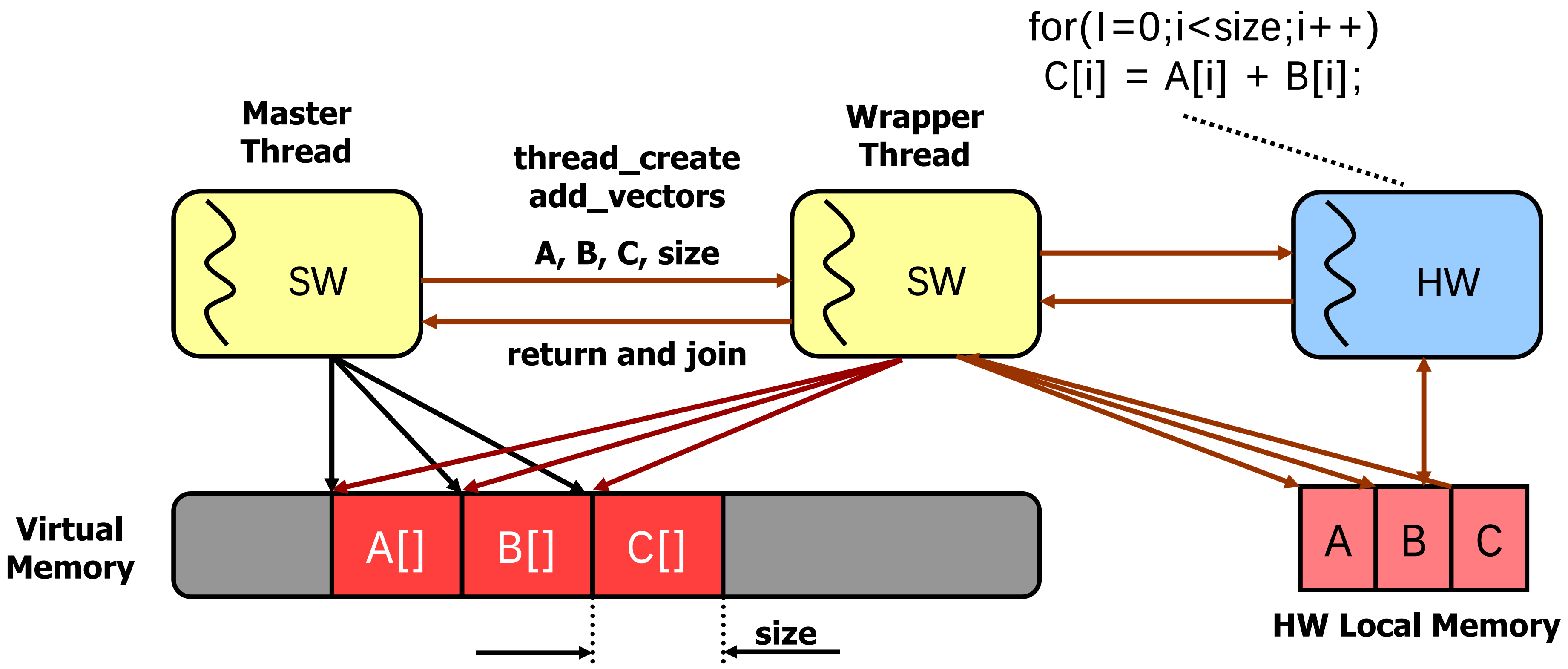


## Extended Multithreading



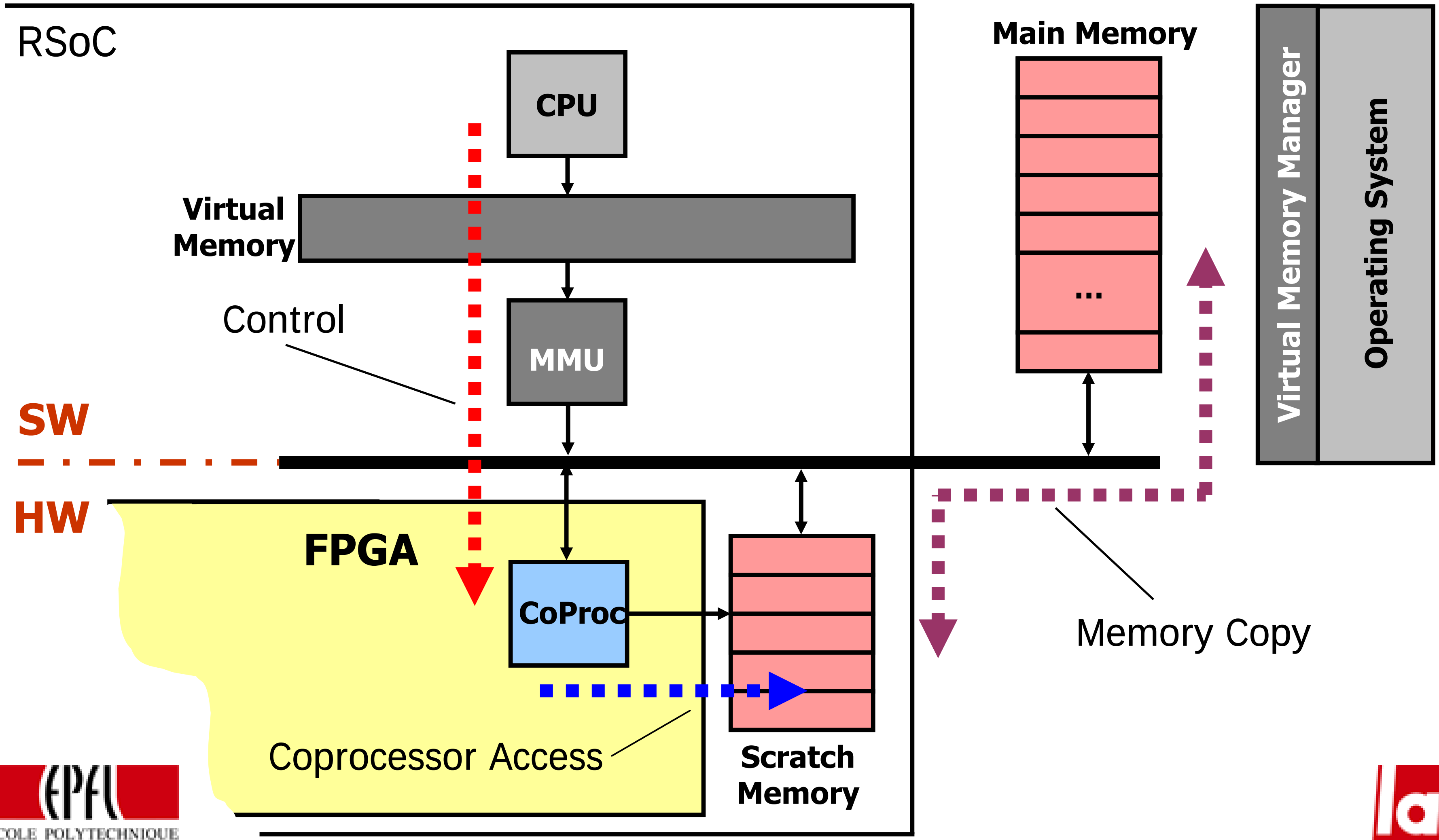
- No standard support when critical threads are to be mapped onto hardware

# Extended Multithreading: Memory Point of View



- Disjoint SW and HW memory address spaces
- Wrapper Thread

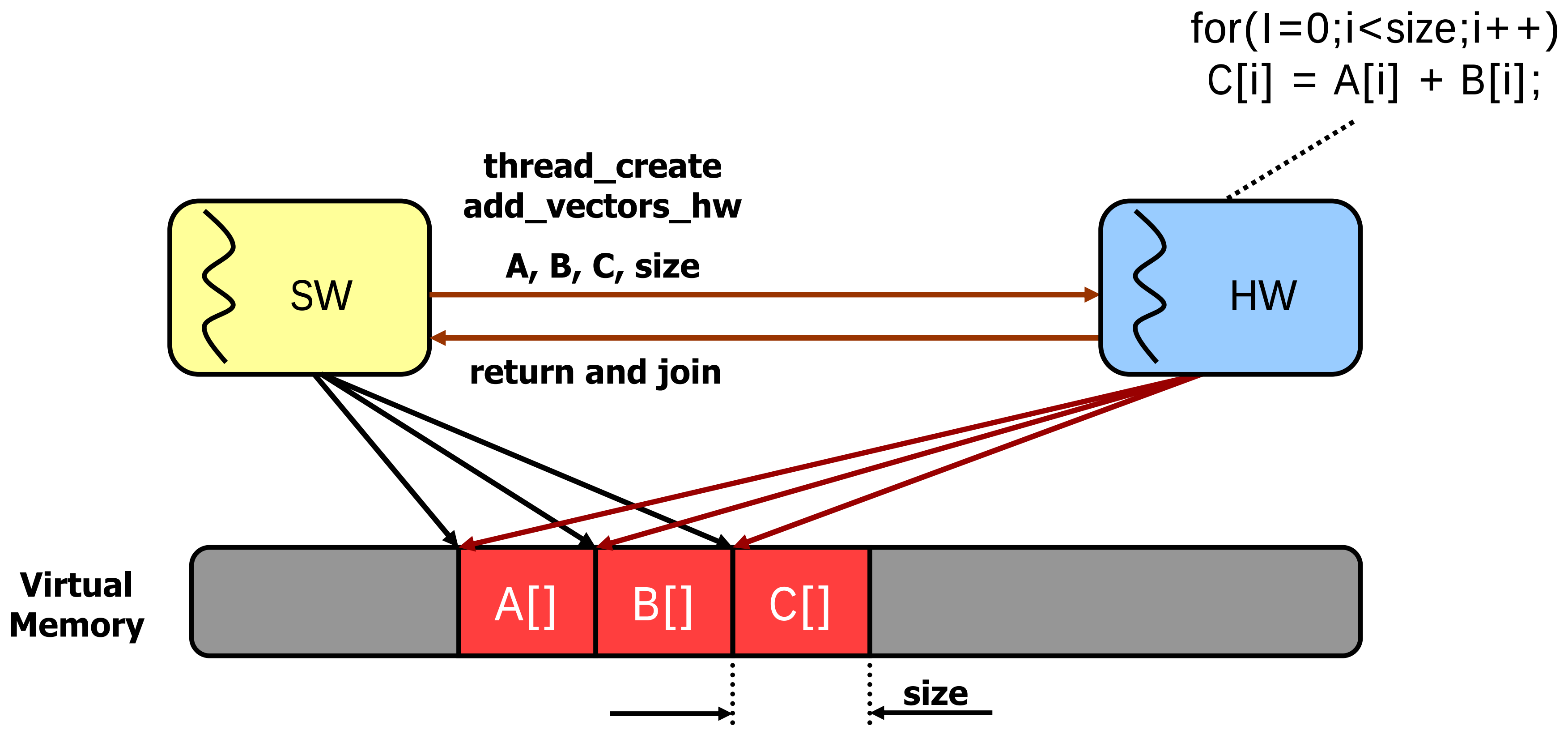
# Typical Coprocessor System



## Burdensome Programming...

```
/* Typical HW accelerator version */
int *A, *B, *C; ... read(A, SIZE); read(B, SIZE);
d_chunk = BUF_SIZE/3; d_ptr = 0;
write(HWACC_CTRL, INIT);
while (d_ptr < SIZE) {
    copy(A + d_ptr, BUF_BASE, d_chunk);
    copy(B + d_ptr, BUF_BASE + d_chunk, d_chunk);
    write(HWACC_CTRL, ADD_VECTORS);
    while() {
        if (read(HWACC_STATUS) == FINISHED) {
            copy(BUF_BASE + 2*d_chunk, C + d_ptr, d_chunk);
            break;
        } else {
            do_some_work();
        }
    }
    d_ptr += d_chunk;
}...
```

# Extended Multithreading: Memory Point of View



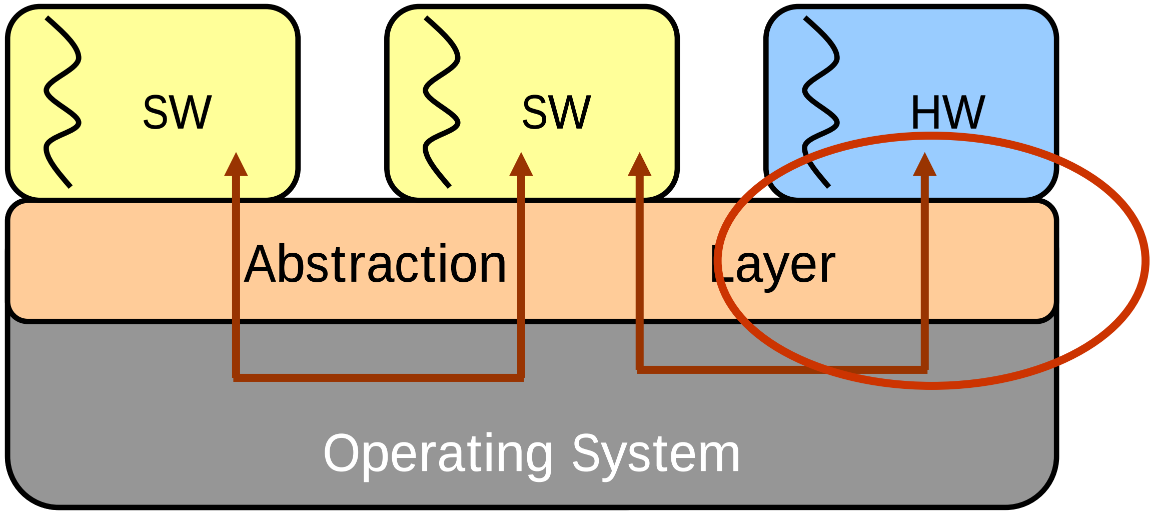
- Same virtual memory address space
- Virtualisation layer

## Transparent Programming…

```
/* Transparent version */

int *A, *B, *C;
int hw_add_vectors(int *, int *, int *, int);
int hwacc_thrd;

read(A, SIZE); read(B, SIZE);

hwacc_thrd = thread_create(hw_add_vectors, A, B, C, SIZE);
do_some_work();
thread_join(hwacc_thrd);
```

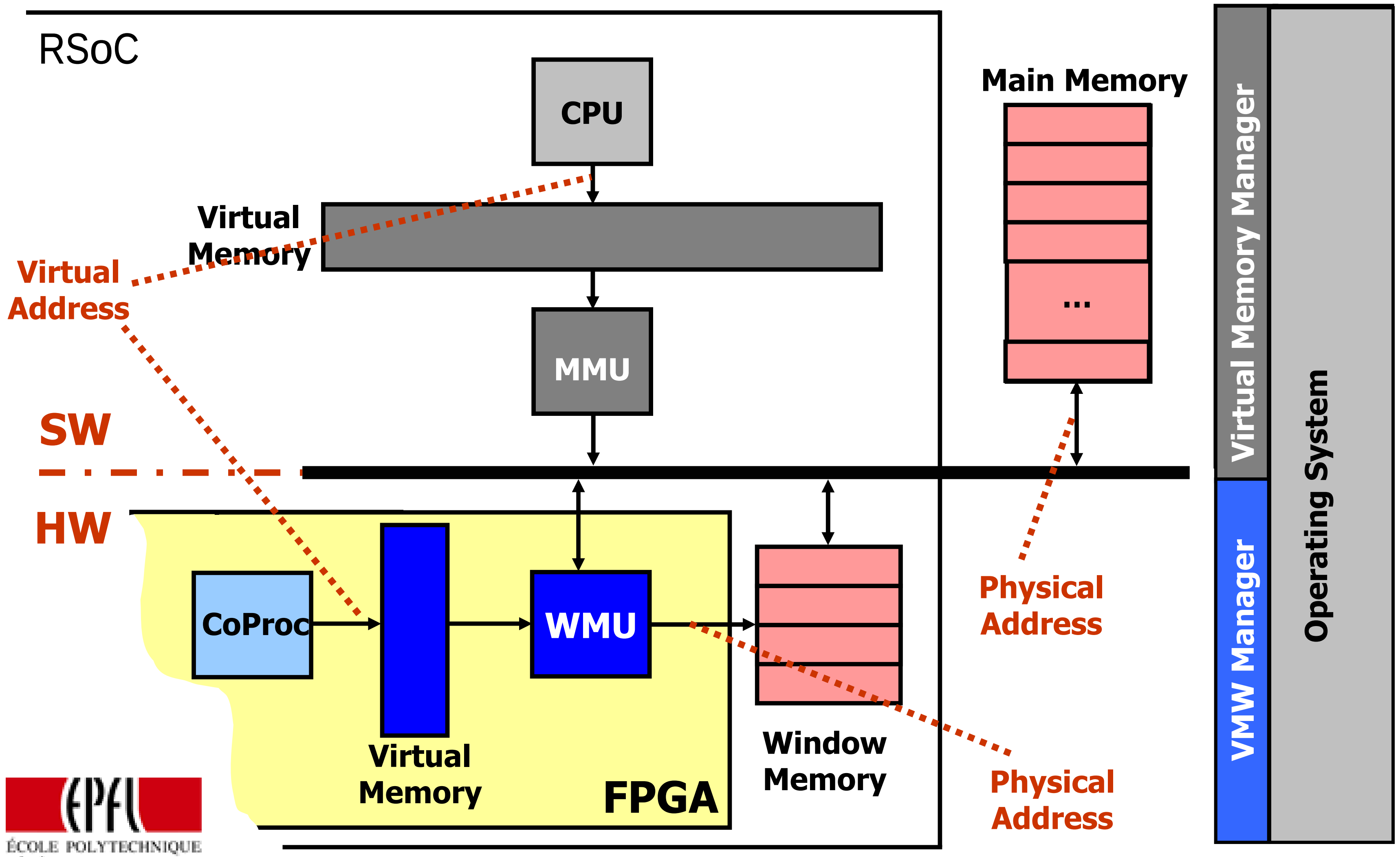
Exactly Same as Software!!!

# Extended Multithreading

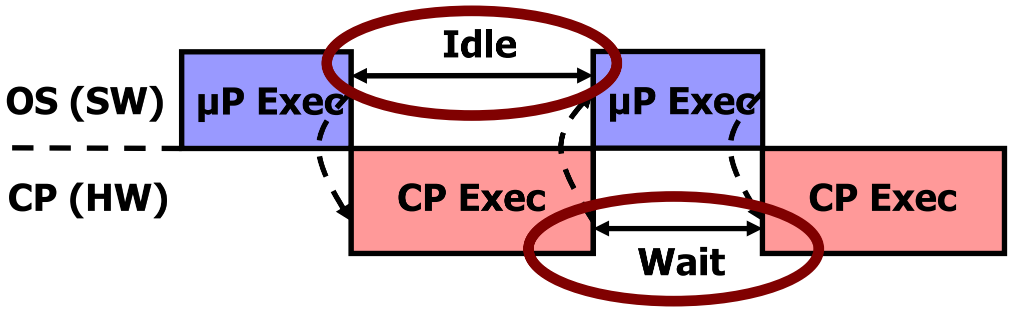


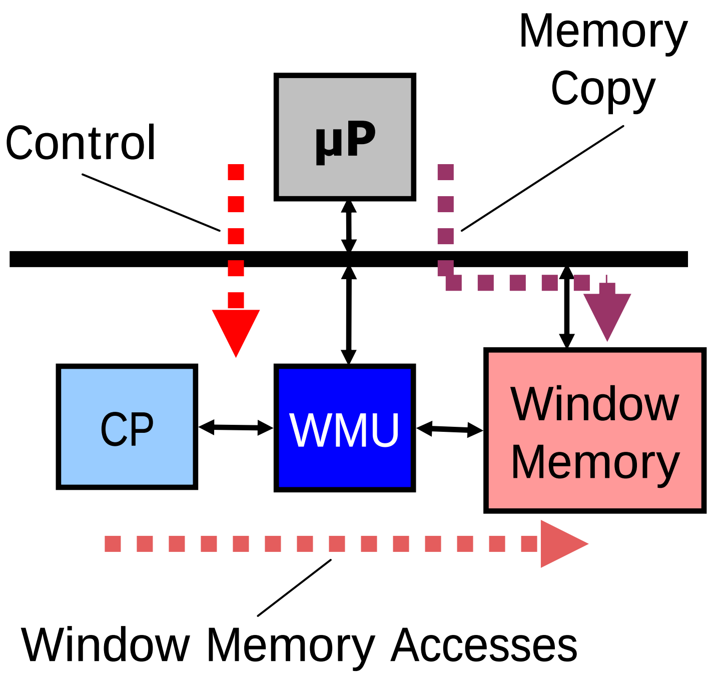
- Abstraction layer extended to support hardware accelerators

# Virtual Memory Window System



## Accelerator Initiated Data Transfers





# Platform-dependent and Portable VHDL-like coding

```
-- Platform dependent
-- Initialisation
ptr_a <= BUF_ADDR;
ptr_b <= BUF_ADDR + BUF_SIZE/3;
ptr_c <= BUF_ADDR + 2*BUF_SIZE;

-- Computation
cycle 1:
    -- partition of A[]
    BUF_ADDR <= ptr_a;
    BUF_ACCESS <= ‘1’;
    BUF_WR <= ‘0’;
```

```
-- Portable
-- Initialisation
ptr_a <= A;
ptr_b <= B;
ptr_c <= C;

-- Computation
cycle 1:
    -- object A[]
    VIRTMEM_ADDR <= ptr_a;
    VIRTMEM_ACCESS <= ‘1’;
    VIRTMEM_WR <= ‘0’;
```

## Platform-dependent and Portable VHDL-like coding (II)

```
-- Platform dependent
cycle 2:
    reg_a <= BUF_DATAIN;
    -- partition of B[]
    BUF_ADDR <= ptr_b;
    BUF_ACCESS <= ‘1’;
    BUF_WR <= ‘0’;
```

```
-- Portable
cycle 2:
    reg_a <= VIRTMEM_DATAIN;
    -- object B[]
    VIRTMEM_ADDR <= ptr_b;
    VIRTMEM_ACCESS <= ‘1’;
    VIRTMEM_WR <= ‘0’;
```

# Platform-dependent and Portable VHDL-like coding (III)

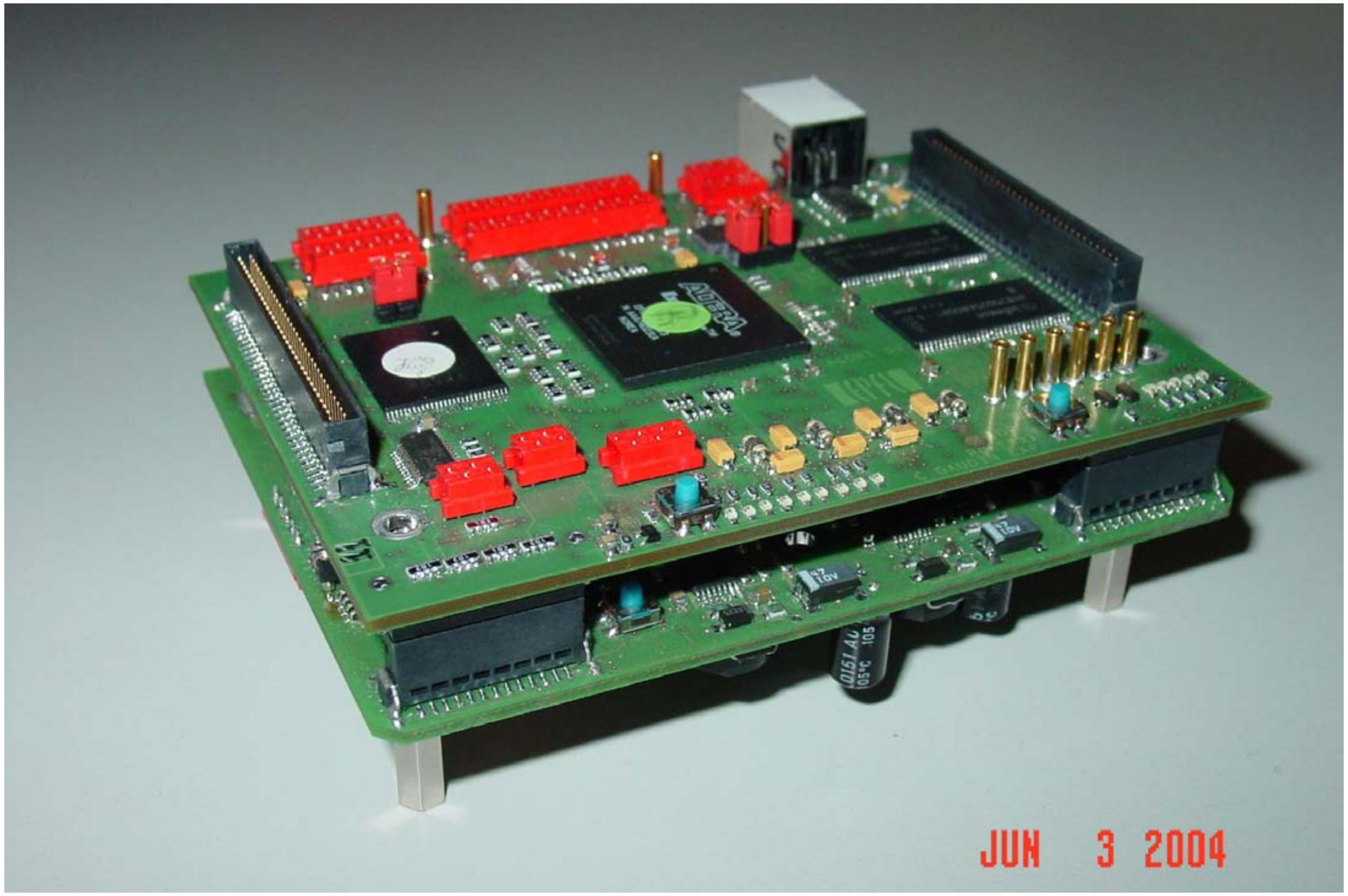
```
cycle 3:
    reg_b <= BUF_DATAIN;
    reg_c <= reg_a + reg_b;
    -- partition of C[]
    BUF_ADDR <= ptr_c;
    BUF_ACCESS <= ‘1’;
    BUF_WR <= ‘1’;
    ptr_{a, b, c} <= ptr_{a, b, c} + 1;
    if (ptr_c = BUF_SIZE) then
    -- finished for a data chunk
            partial_finish();
    else  cycle 1;
    end if;
```

```
cycle 3:
    reg_b <= VIRTMEM_DATAIN;
    reg_c <= reg_a + reg_b;
    -- object C[]
    VIRTMEM_ADDR <= ptr_c;
    VIRTMEM_ACCESS <= ‘1’;
    VIRTMEM_WR <= ‘1’;
    ptr_{a, b, c} <= ptr_{a, b, c} + 1;
    if (ptr_c = SIZE) then
    -- finished for the entire vectors
            finish();
    else  cycle 1;
    end if;
```

## Experimental Setup

- Board based on Altera Excalibur EPXA1:
  ARM (133MHz), FPGA, SDRAM (64MB), Linux
- On-chip DP RAM (16KB) directly accessible by FPGA
- WMU in synthesizable VHDL
- VMW manager as Linux kernel module
- IDEA (encryption/decryption)
- ADPCM Decoder (MediaBench) coprocessor

# Experimental Setup: ROKEPXA Board

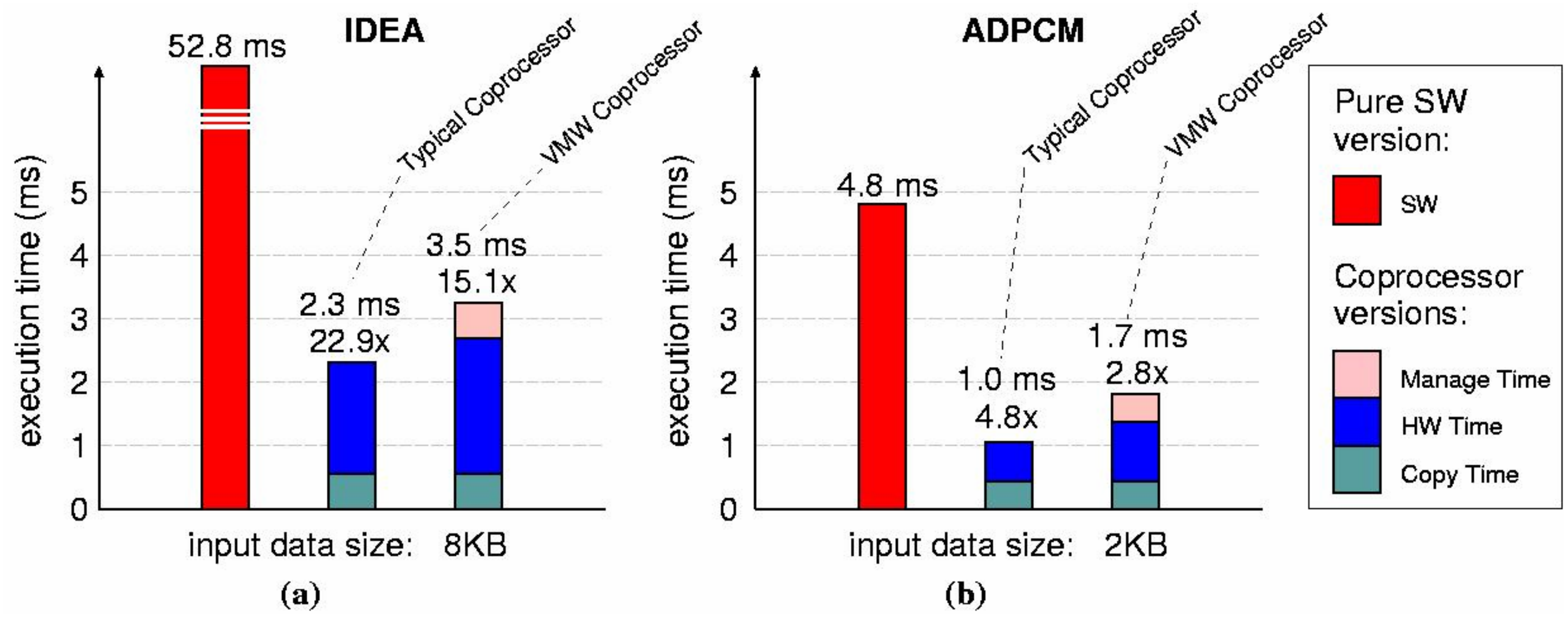


## Programming Example: IDEA Cryptography

```
/* main function */
void main() {
    int *A, *B, n64;
    ...
    read(A, n64);
    idea_encrypt(A, B, n64);
    ...
}
```

```
/* library function */
int idea_encrypt(int *A, int *B, int n64) {
    struct cp_param param;
    ...
    param.u.nparam = 3;
    param.p[0] = A;
    param.p[1] = B;
    param.p[2] = n64;
    FPGA_EXECUTE(HW_IDEA, &param);
    return param.u.retval;
}
```

# Experimental Results



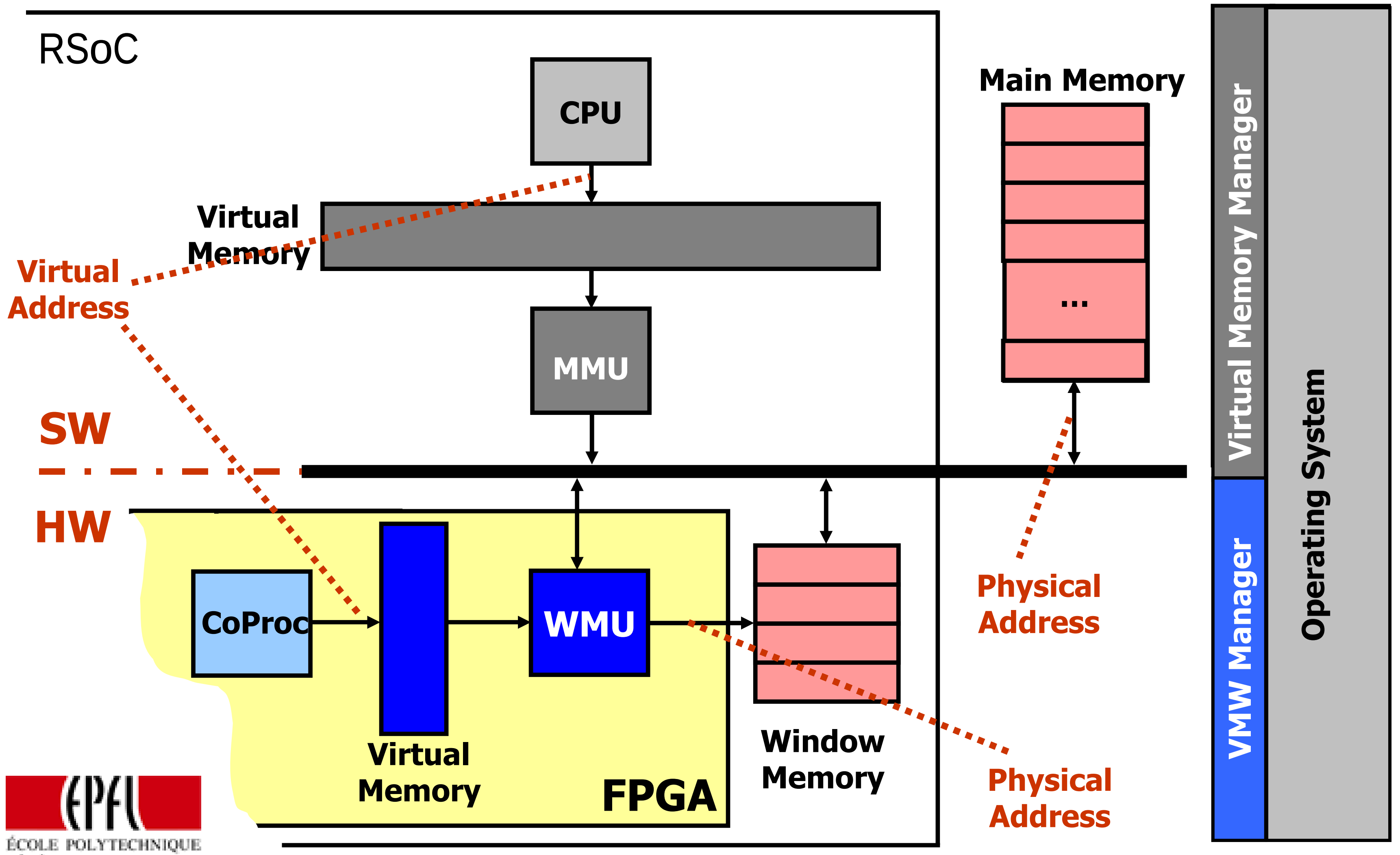
## WMU Area Overhead

| | FPGA (EPXA1) Resources | | Total Accelerator Resources | |
|---|---|---|---|---|
| Block Type | Number of Units | WMU Area (%) | WMU Area, ADPCM (%) | WMU Area, IDEA (%) |
| Logic Cell | 576 | 14 | 48 | 16 |
| Memory | 5 | 19 | 83 | 45 |

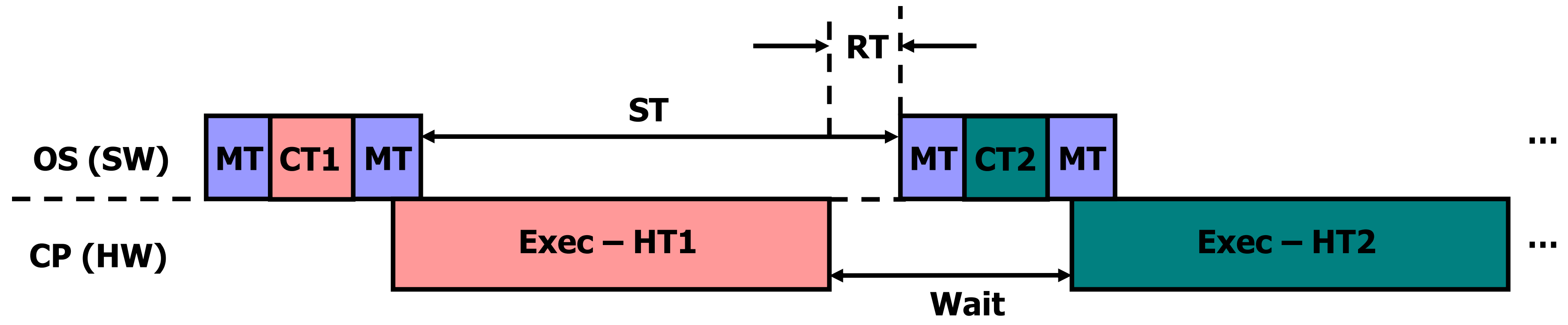
## Outline

- Introduction and Motivation
- Transparent Programming Model
  *Virtual Memory Window*
- Dynamic Prefetching
- Unrestricted Automated Synthesis
- Applications and Future Perspectives

# Virtual Memory Window System



## VMW with No Prefetching



**Legend:**

**MT** – Management Time

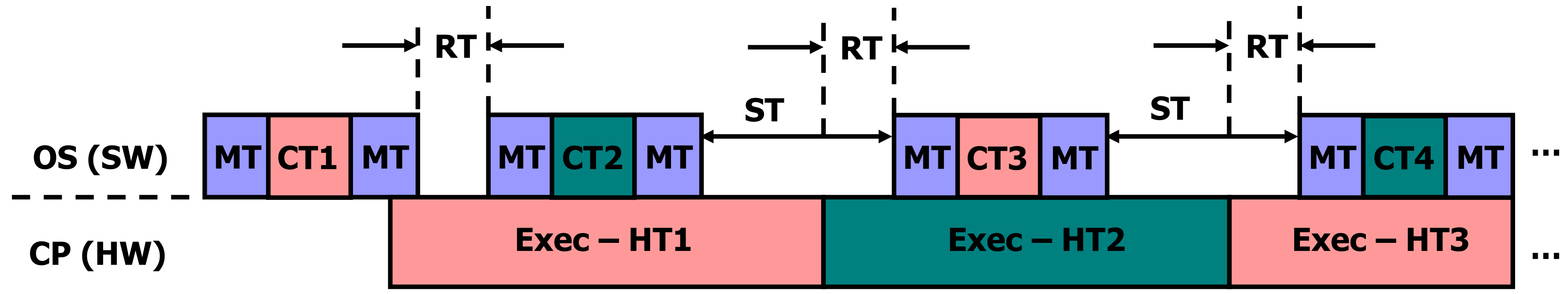
**CT** – Copy Time

**ST** – Sleep Time

**RT** – Response Time

**HT** – Hardware Time

# VMW with Prefetching



**Legend:**

**MT** – Management Time

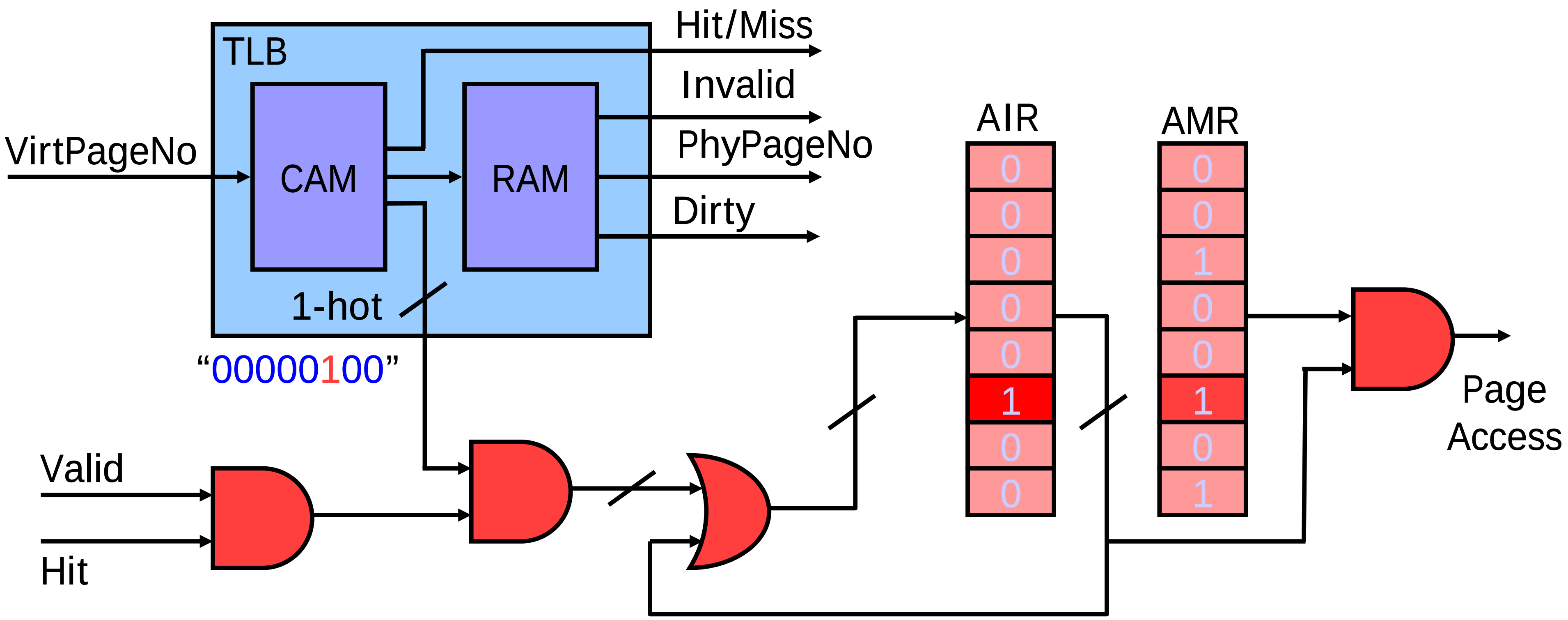
**CT** – Copy Time

**ST** – Sleep Time

**RT** – Response Time

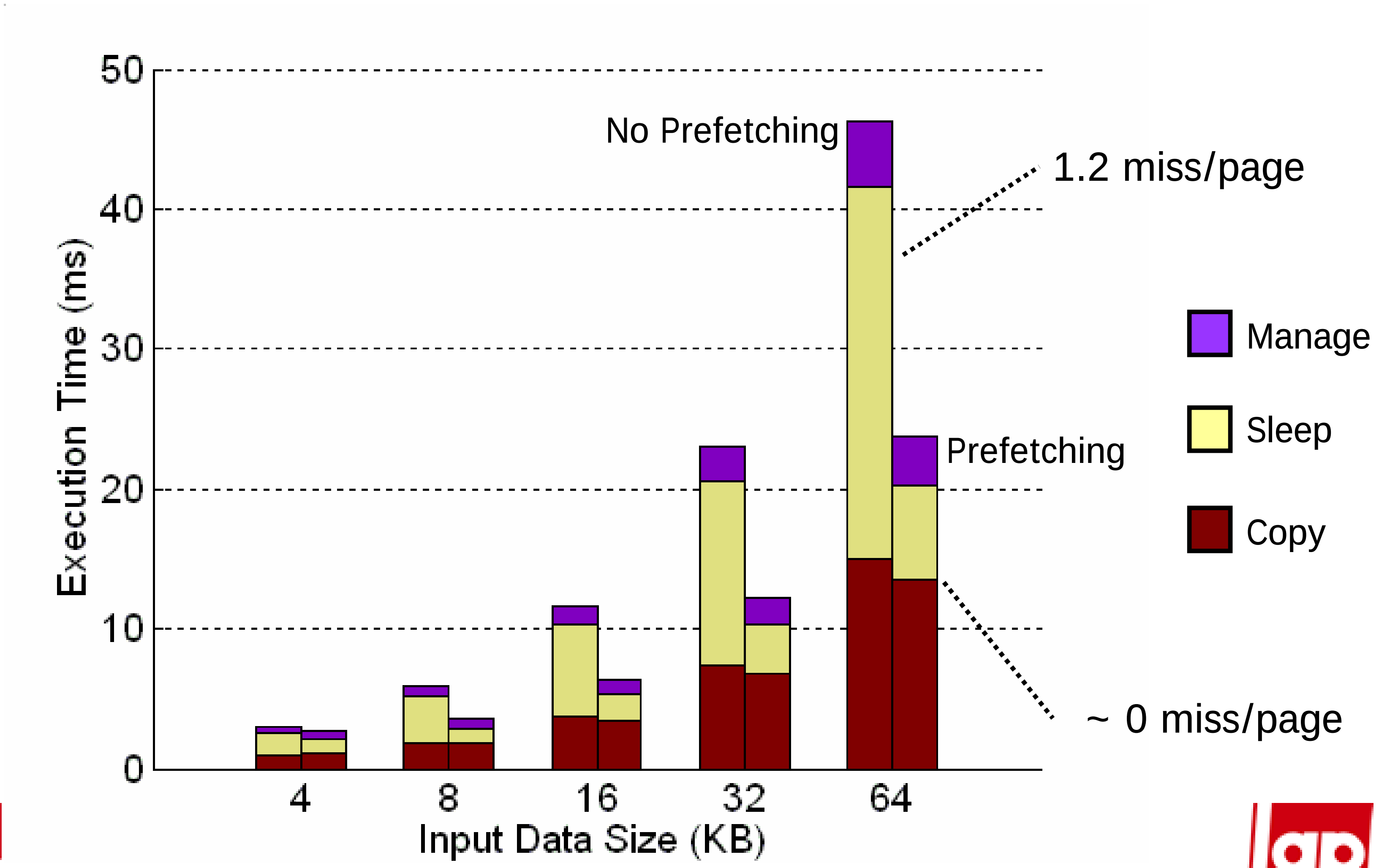
**HT** – Hardware Time

# Window Management Unit (WMU): Page Access Detection

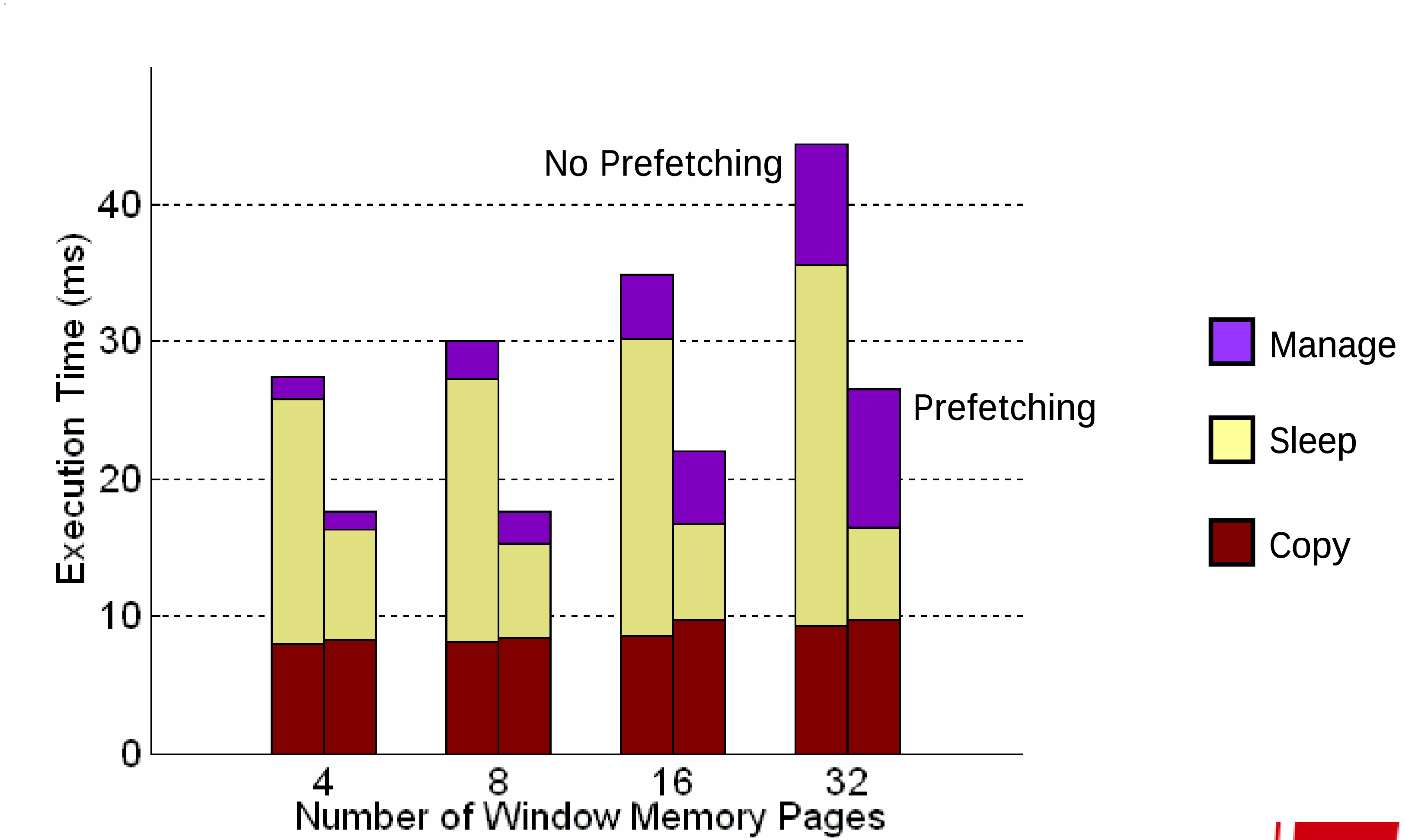


AIR – Access Indicator Register
AMR – Access Mask Register

# ADPCM Decoder Prefetching Improvements



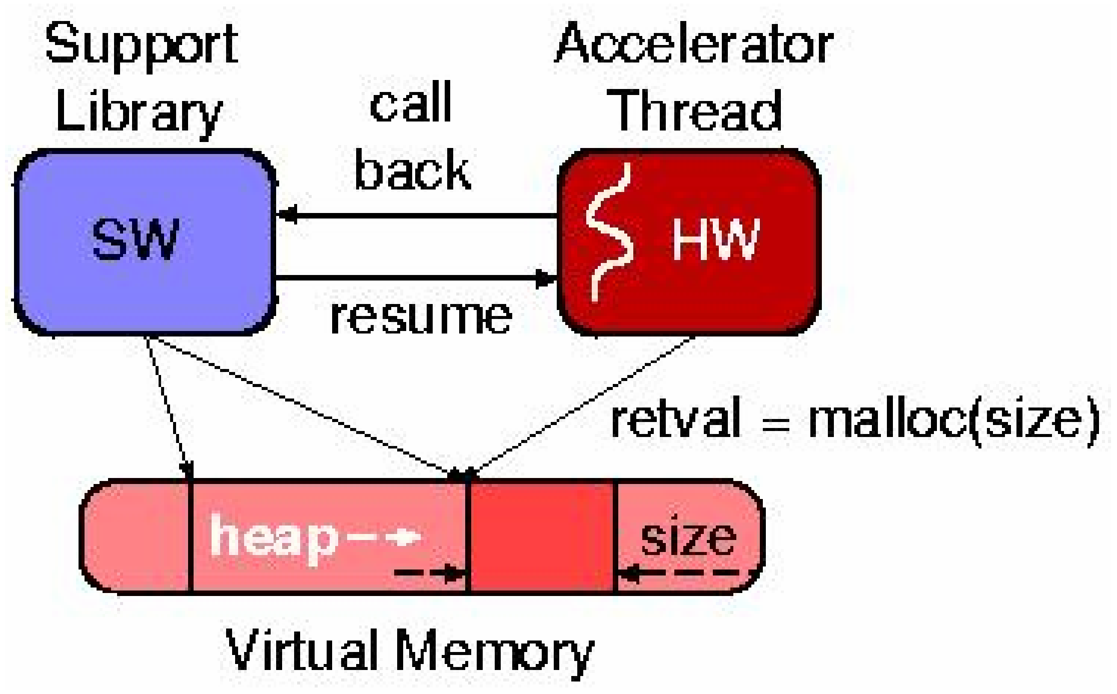
# IDEA Encryption for 64KB of Input Data



## Outline

- Introduction and Motivation
- Transparent Programming Model
  *Virtual Memory Window*
- Dynamic Prefetching
- Unrestricted Automated Synthesis
- Applications and Future Perspectives

## Malloc Example



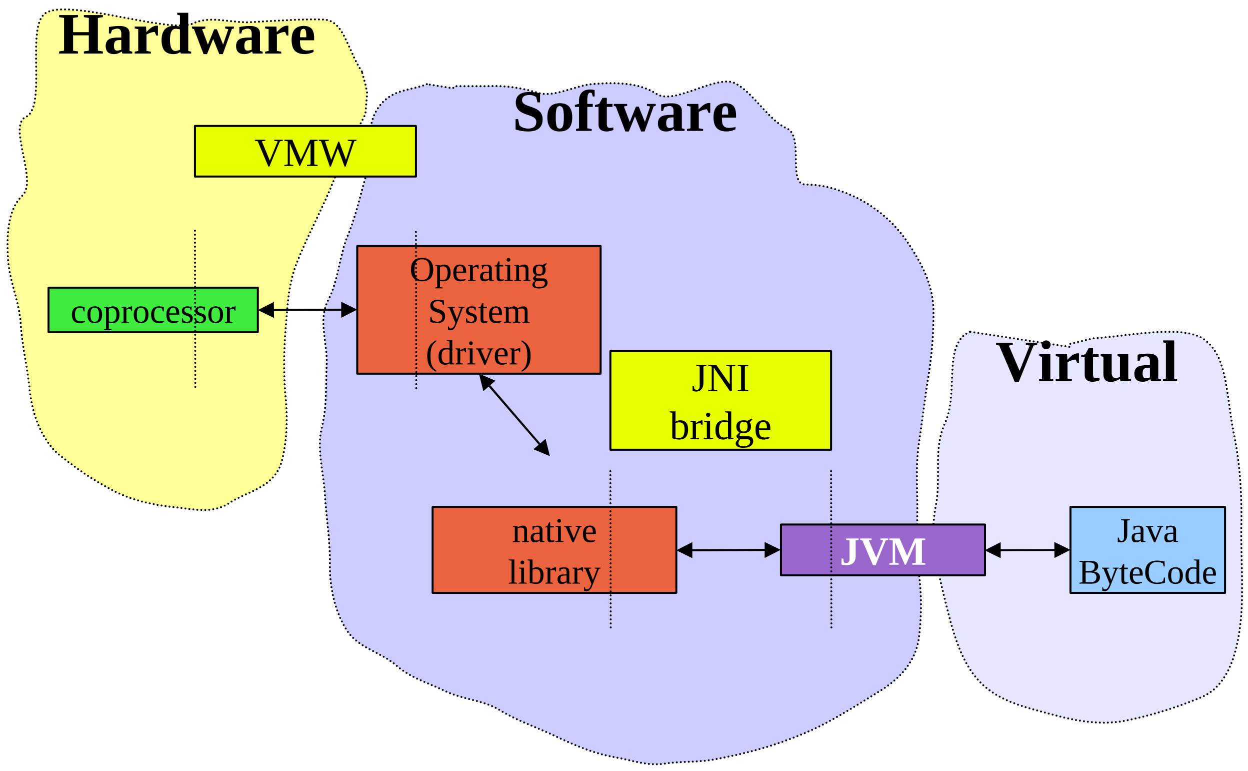
## Unrestricted Automated Synthesis: Function Calls; Malloc Example

```
ioctl(fd, VMW_IOC_START, &params);
...
while(! params.hwret) {
    switch(params.cback) {
        case 1: ... break;
        case 2: ... break;
        case 3: params.retval = malloc(params.p[0]);
                ioctl(fd, VMW_IOC_RESUME, &params); break;
        ...
        case n:
    }
}
```

## Outline

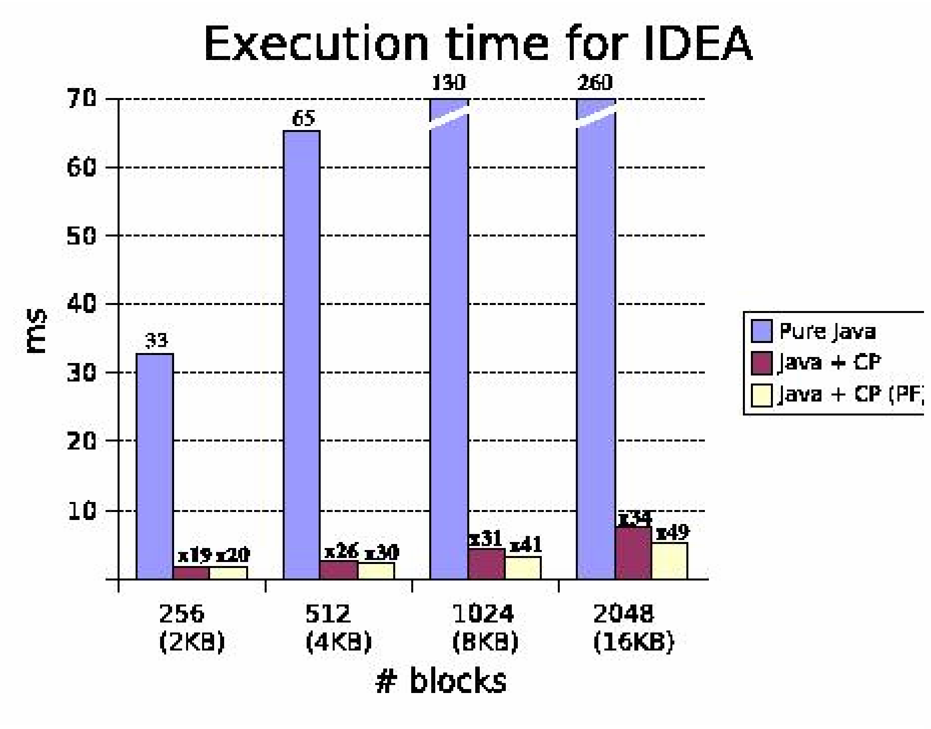
- Introduction and Motivation
- Transparent Programming Model
  *Virtual Memory Window*
- Dynamic Prefetching
- Unrestricted Automated Synthesis
- Applications and Future Perspectives

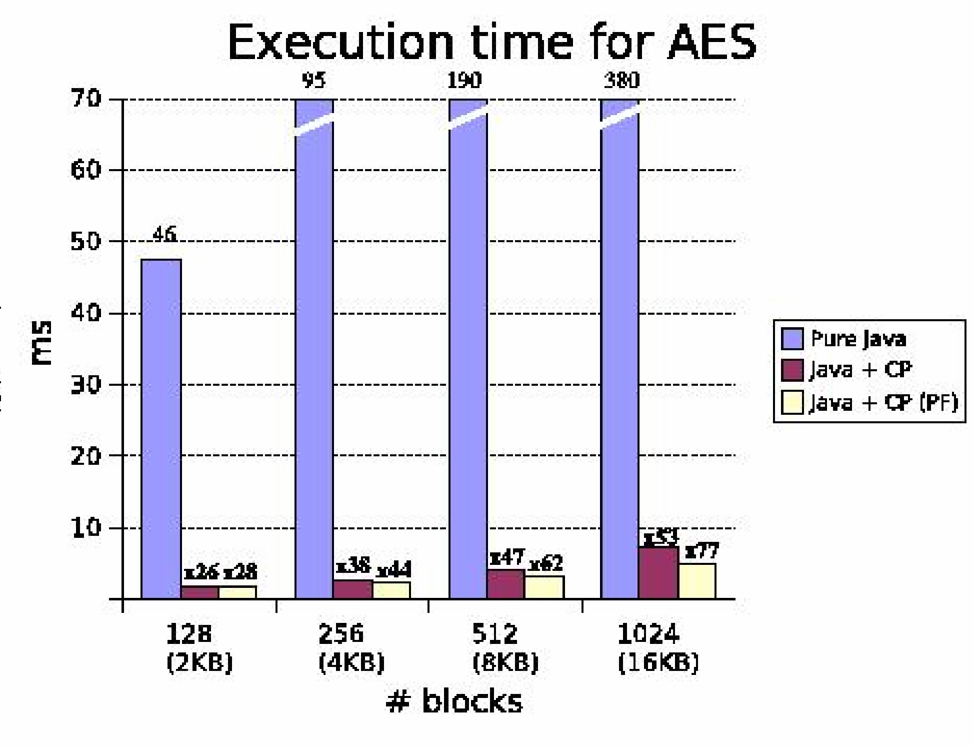
# Portable Hardware Accelerators for Java Virtual Machines



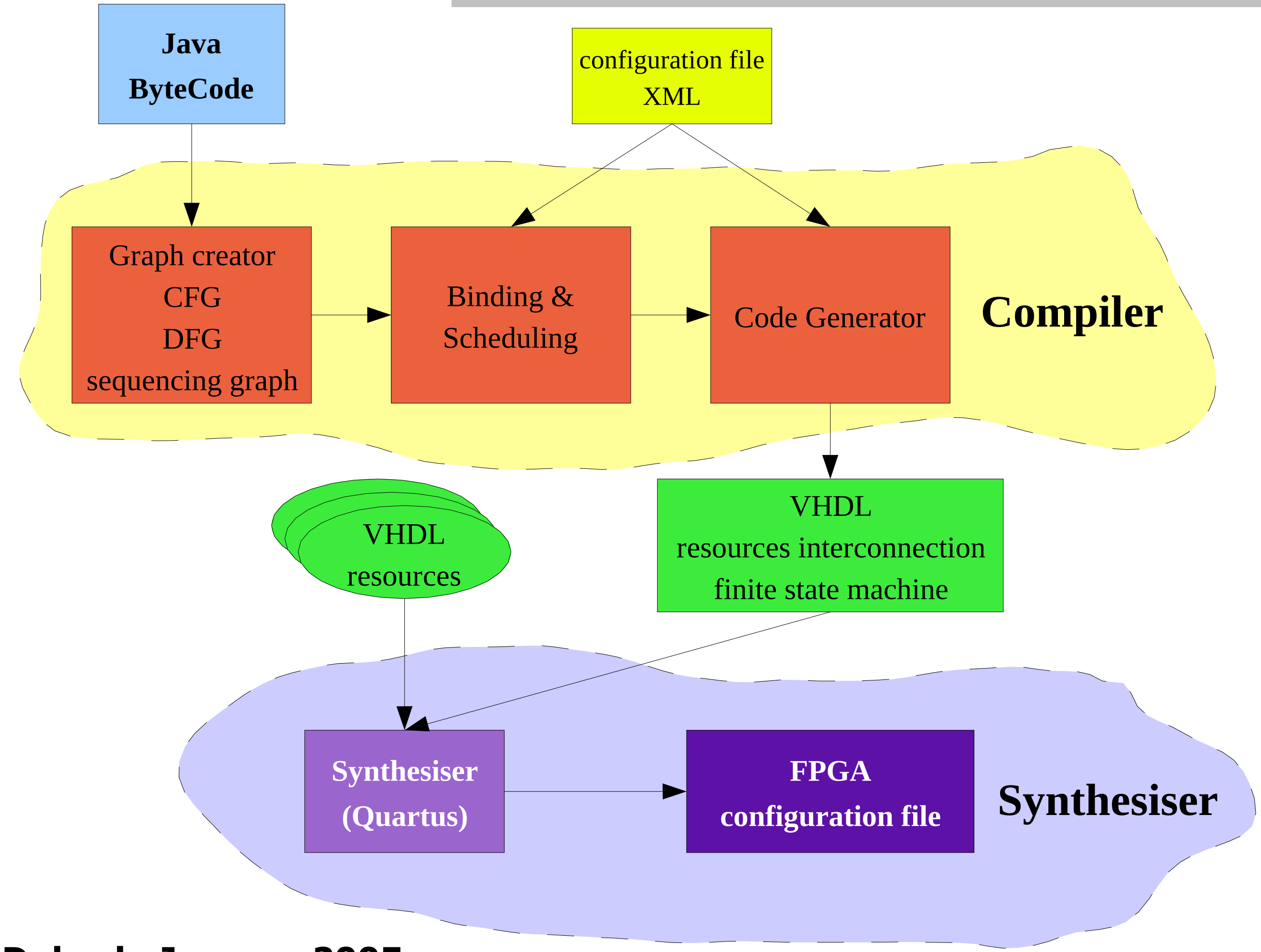
**Dubach et al., February 2004**

## Java Accelerator Results



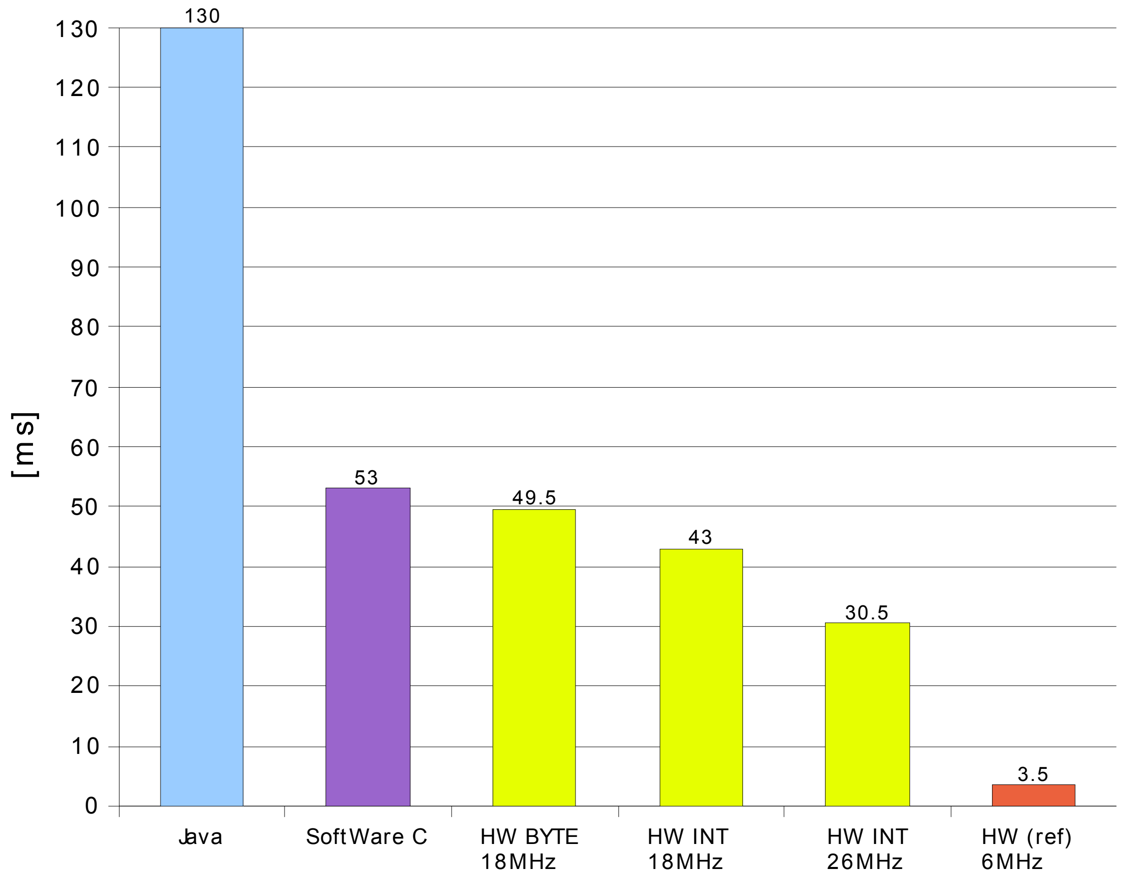


# Unrestricted Java Bytecode Synthesis

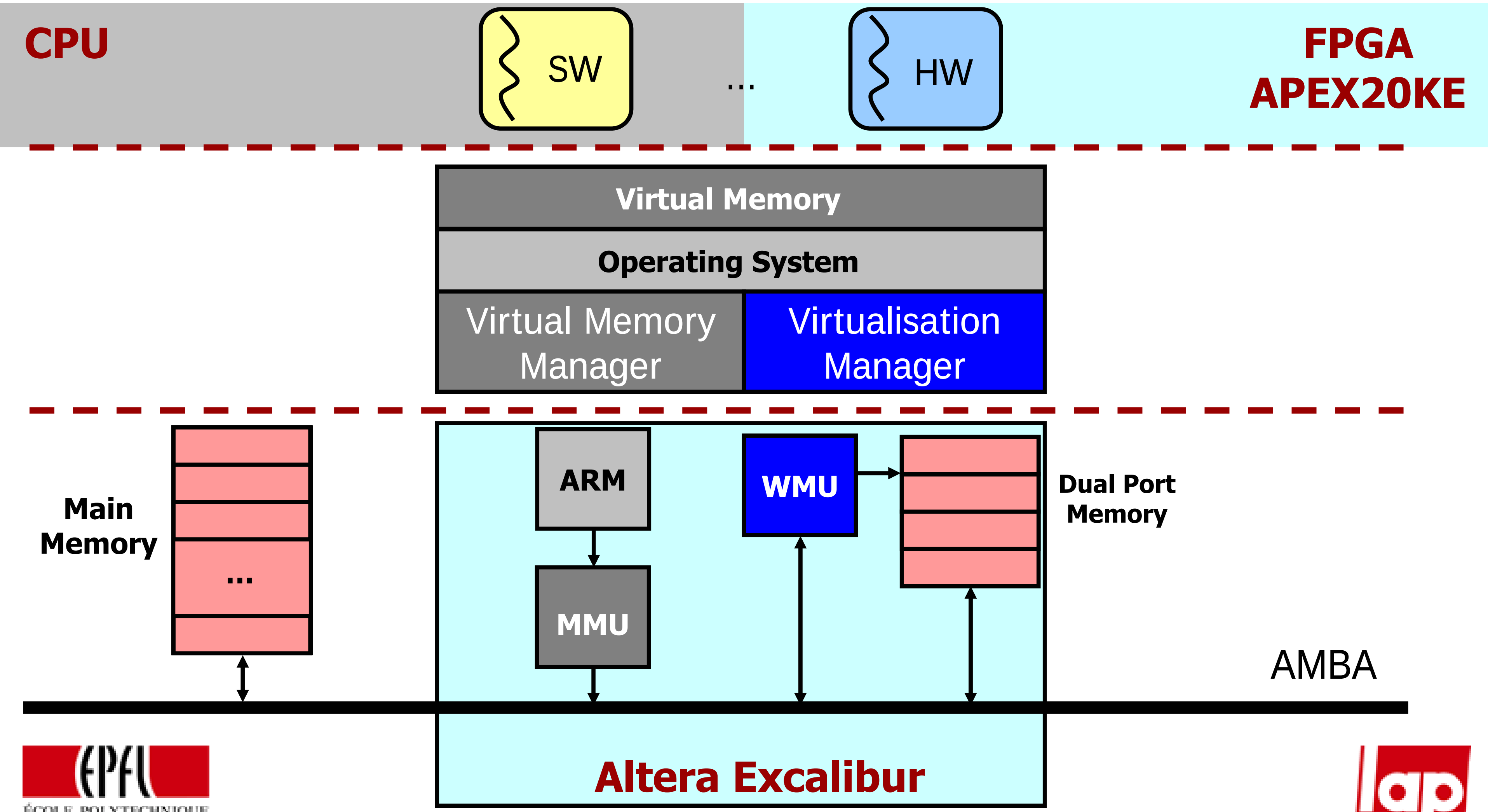


**Dubach, January 2005, Masters thesis**

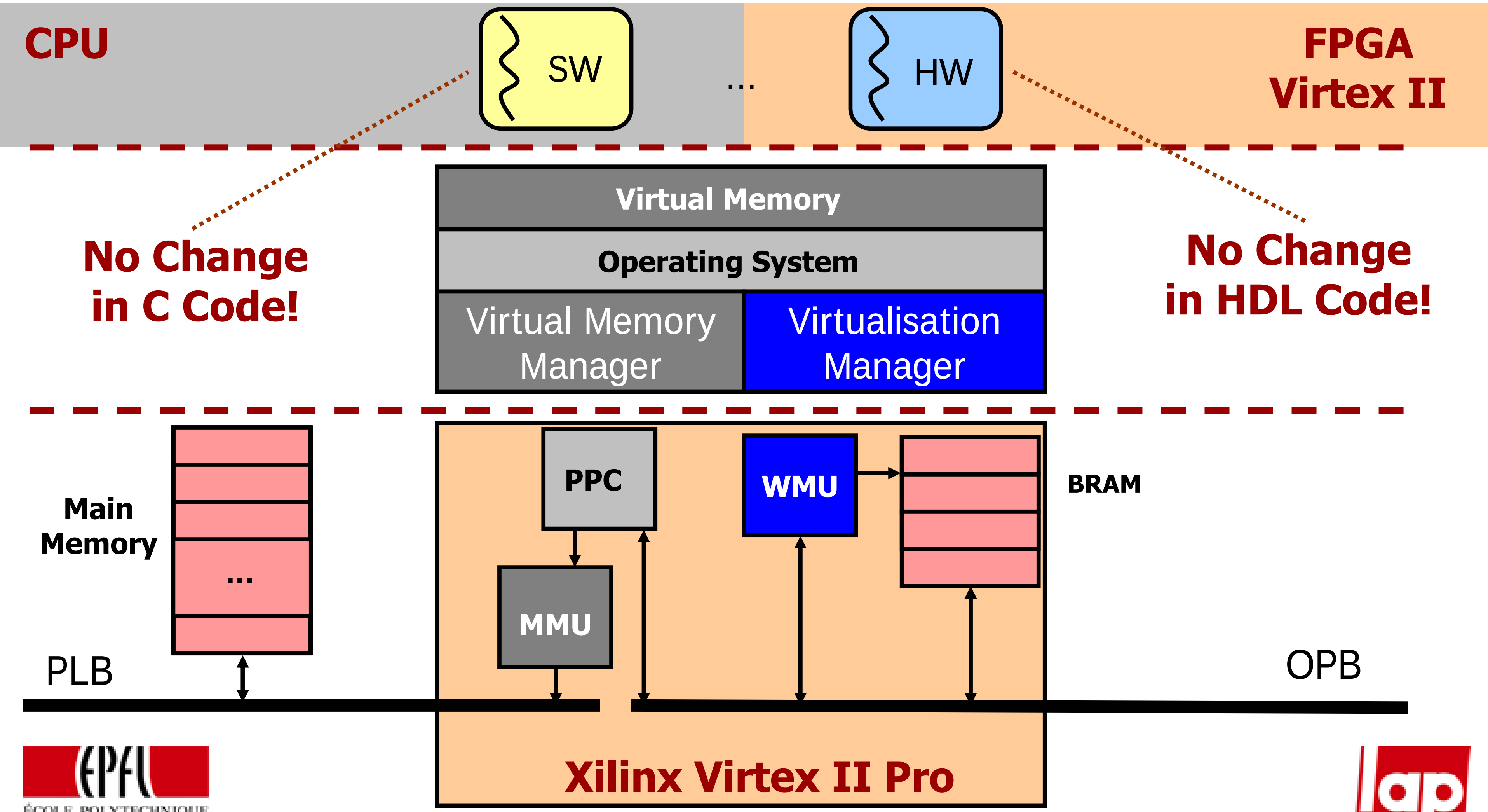
## IDEA Synthesis Results



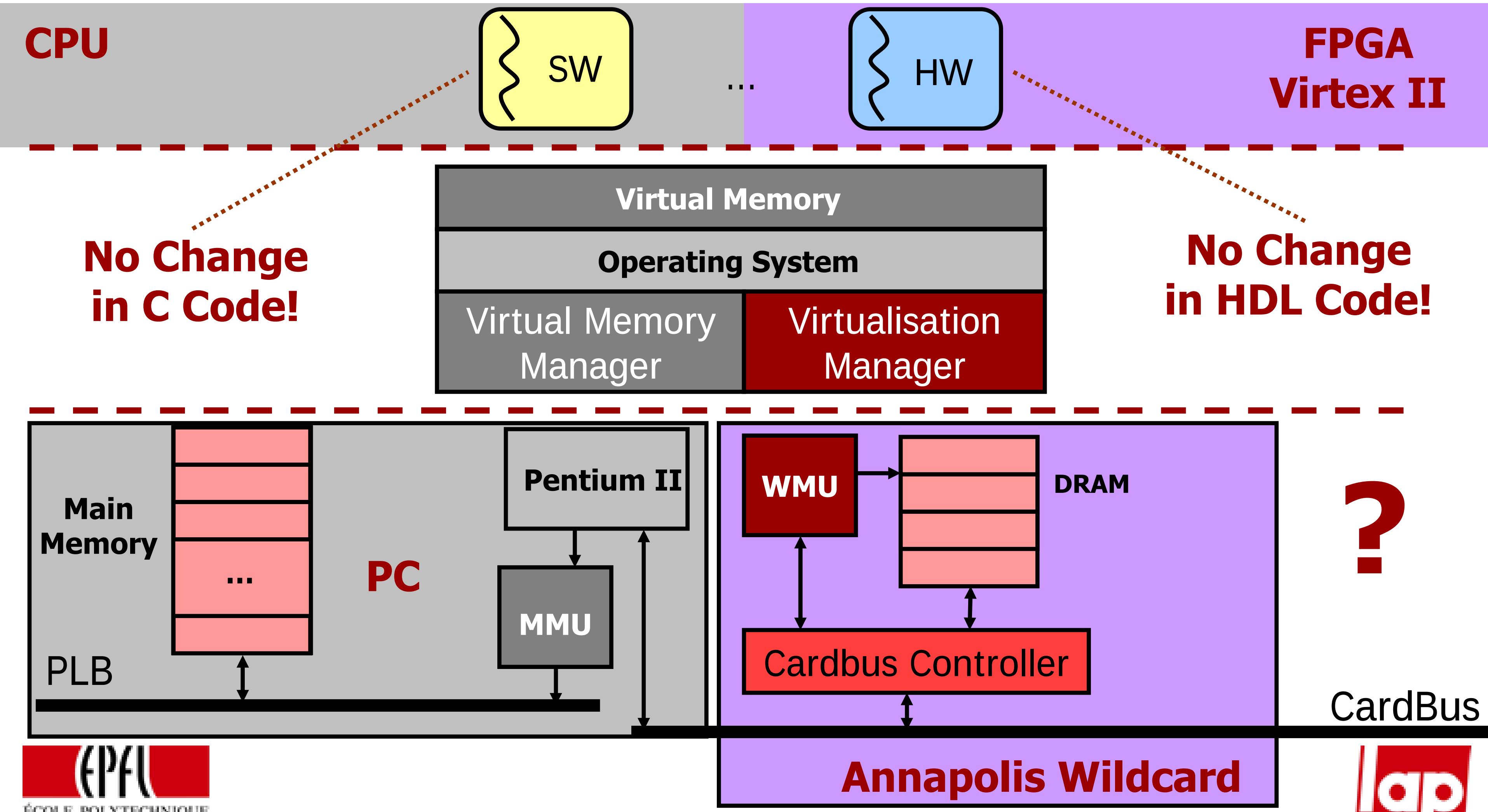
## Virtualisation Layer: Portability of SW/HW Applications



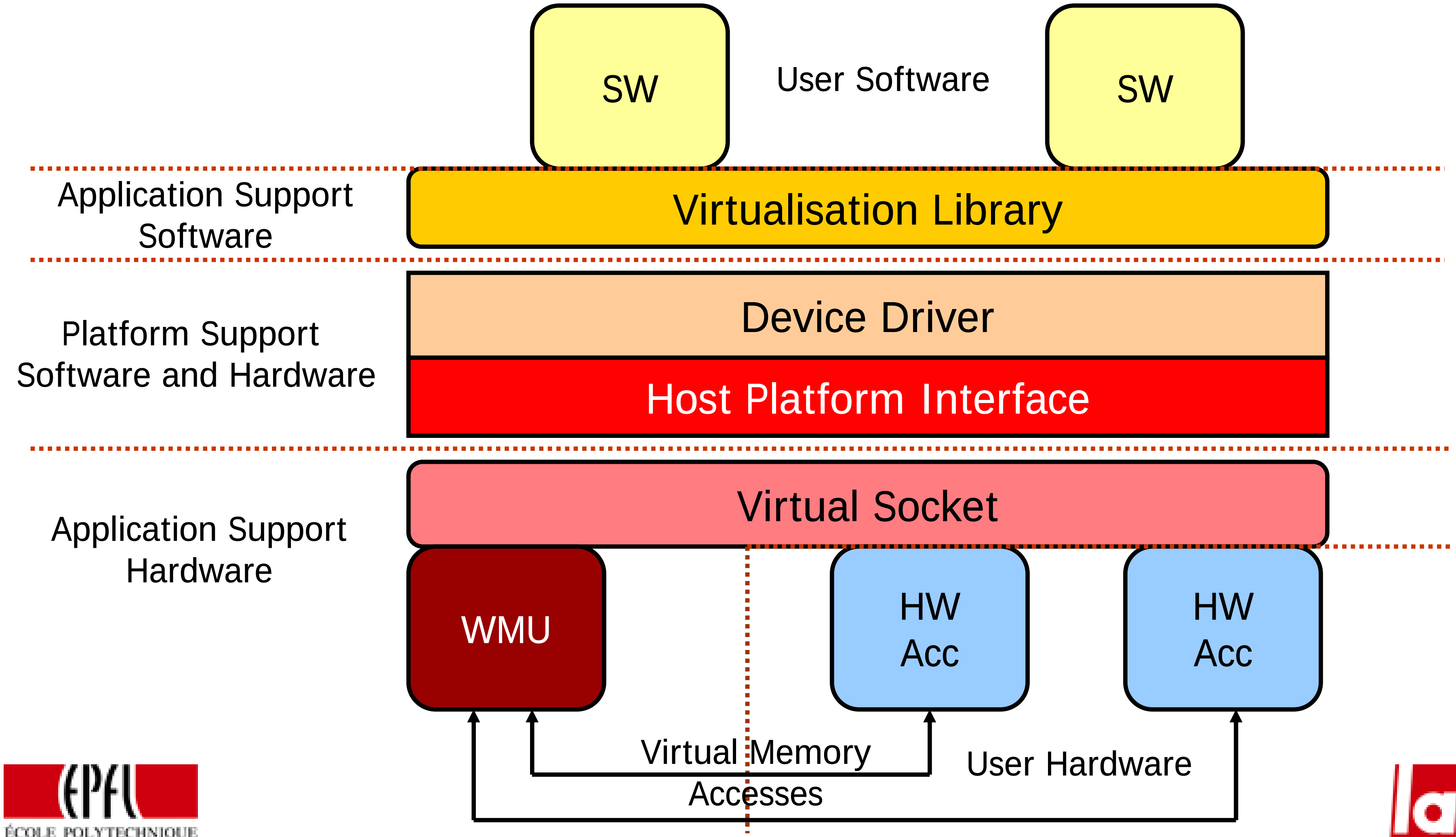
# Virtualisation Layer: Portability of SW/HW Applications (II)



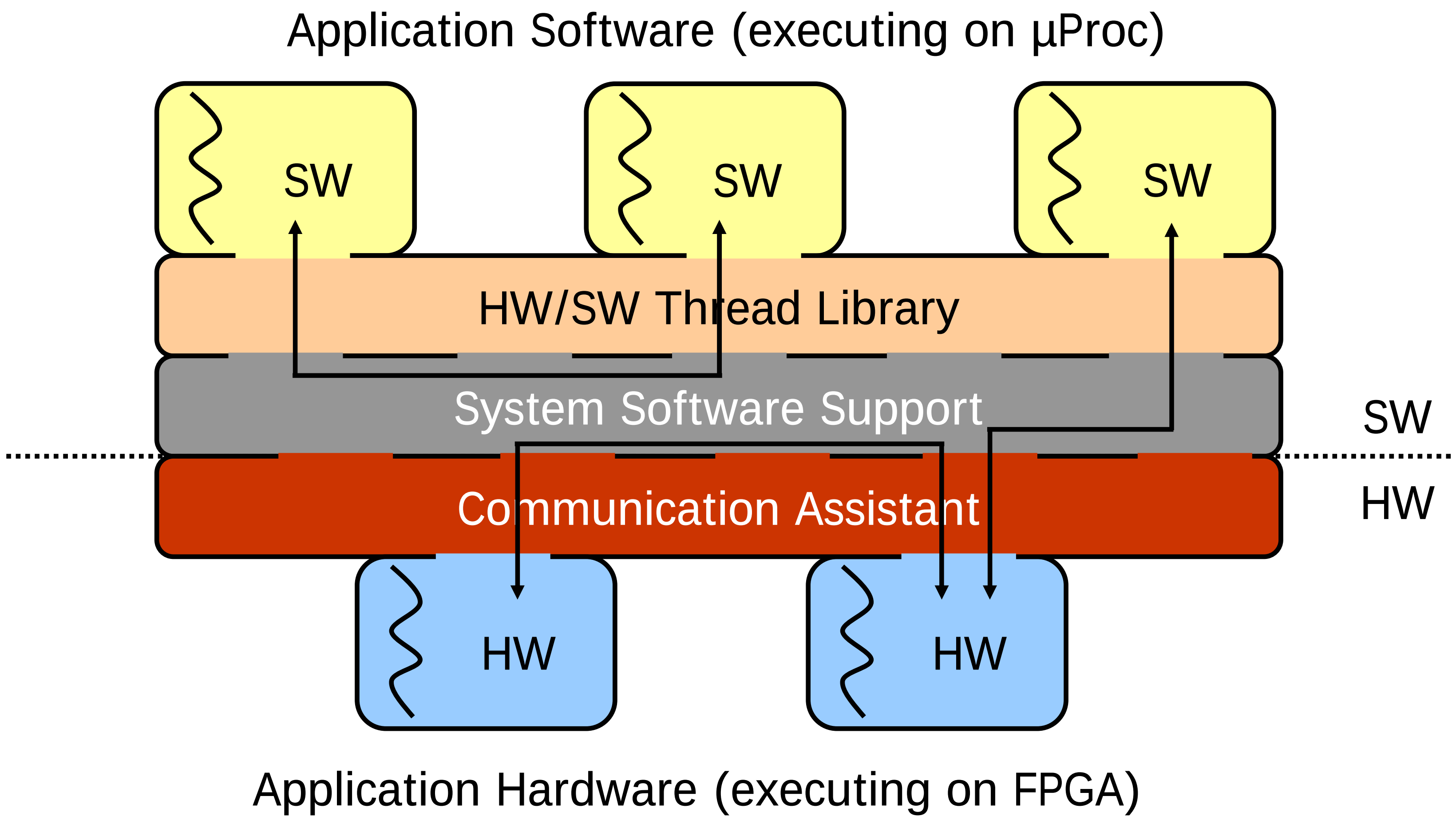
## Portability for MPEG4 HW Reference (Supported by MPEG4 group and Xilinx)



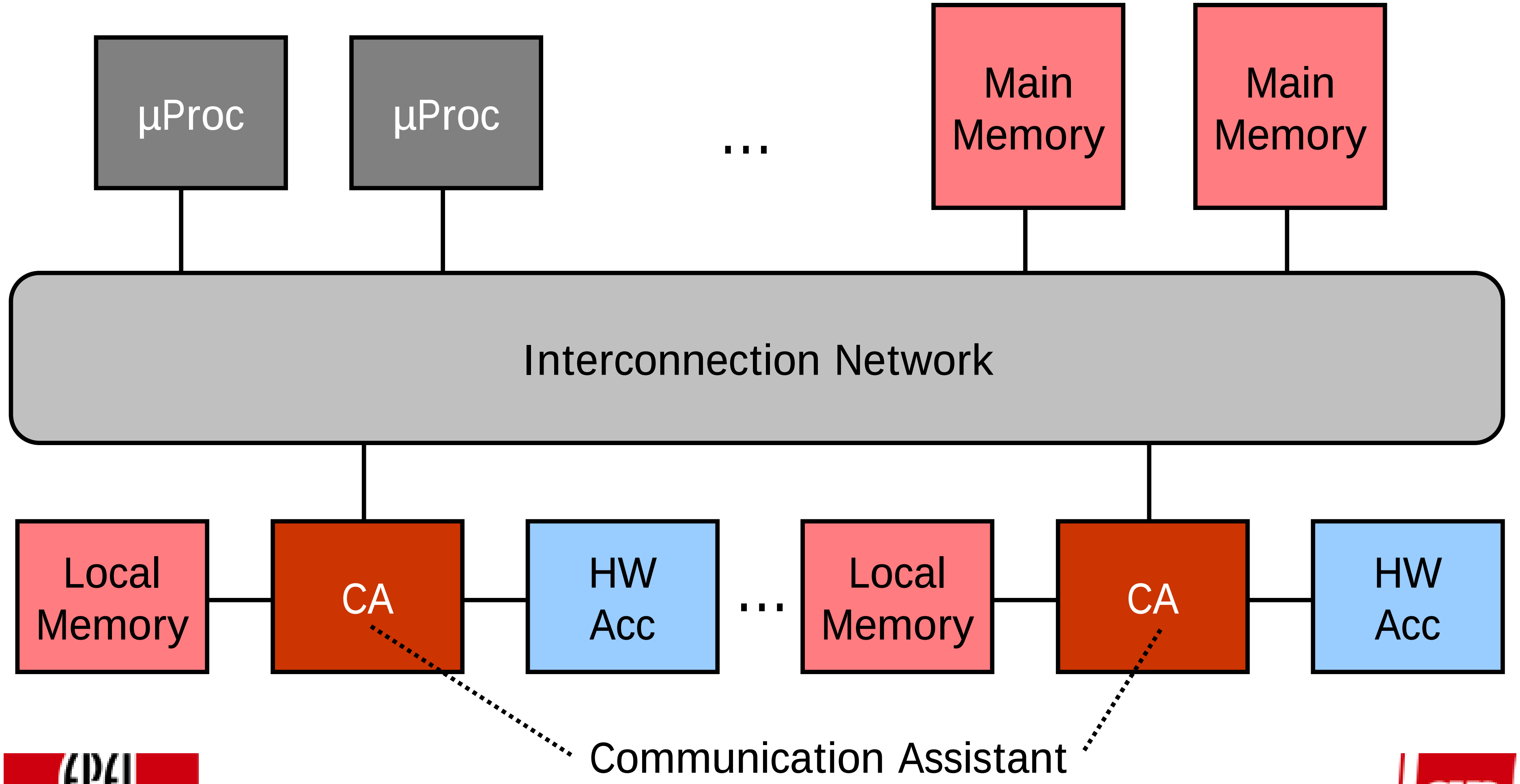
# VMW for Virtual Sockets (MPEG4 Standardisation Group)



# Virtualisation Layer for Transparent HW/SW Multithreading


Application Software (executing on µProc)
SW
SW
SW
HW/SW Thread Library
System Software Support
SW
HW
Communication Assistant
HW
HW
Application Hardware (executing on FPGA)

# Reconfigurable Parallel Architecture



## Conclusions

- Virtualisation layer for seamless hardware and software interfacing
- Portable reconfigurable applications through recompilation and resynthesis
- Compiler, synthesizer and OS suffice!
- Parallel programming paradigm for reconfigurable computing systems

## Conclusions (II)

- Significant speedup on a real platform with limited overhead!
- Translation should go to VLSI!
- Runtime memory optimisation:
  Benefits without any change in user SW/HW code
- Dynamic prefetching almost hides memory latency!
- Support for unrestricted automated synthesis
- Portable prototyping framework
  for SW/HW partitioning

# Seamless Integration of Hardware and Software in Reconfigurable Computing Systems

**Miljan Vuletić, Laura Pozzi, and Paolo Ienne**
**{Miljan.Vuletic, Laura.Pozzi, Paolo.Ienne}@epfl.ch**
**http://lap.epfl.ch**
**Ecole Polytechnique Fédéral de Lausanne**
**The School of Computer and Communication Sciences**
**Processor Architecture Laboratory**