Communication and Synchronization in Multithreaded Reconfigurable Computing Systems

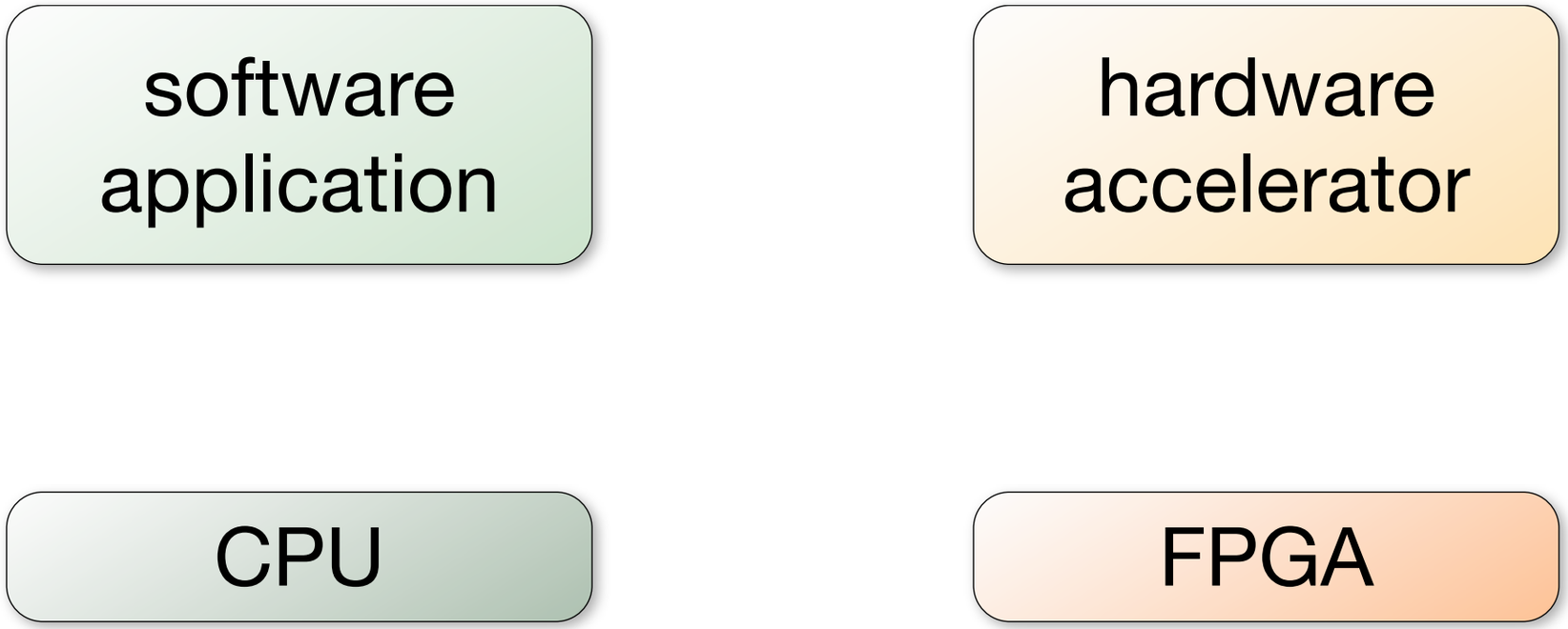
Enno Lübbers and Marco Platzner
Computer Engineering Group
University of Paderborn

{enno.luebbers, platzner}@upb.de

UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft

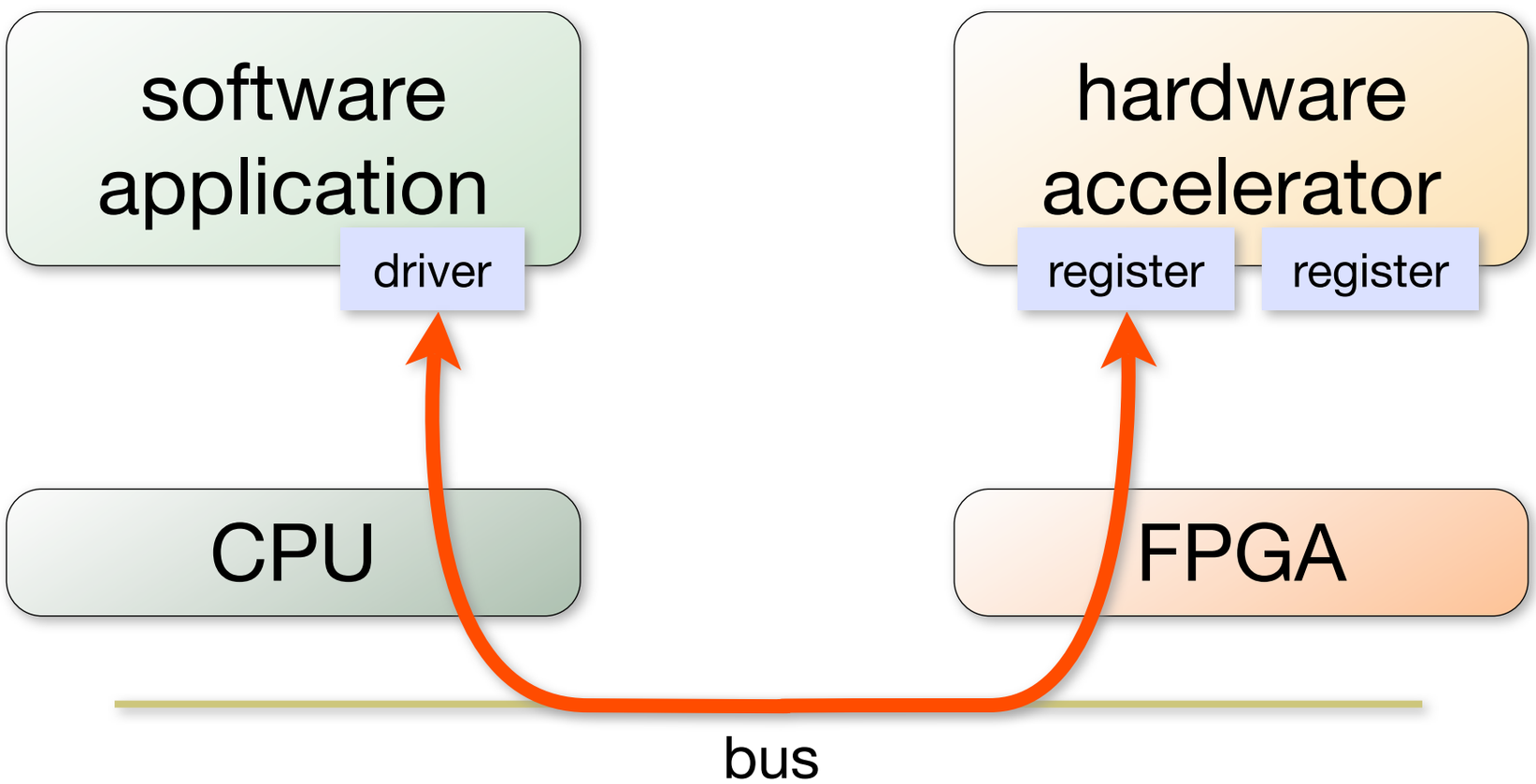
Design of CPU/FPGA Systems

- hardware accelerators typically integrated as slave coprocessors
- hardware/software boundary explicit
- tedious to program
- portability issues

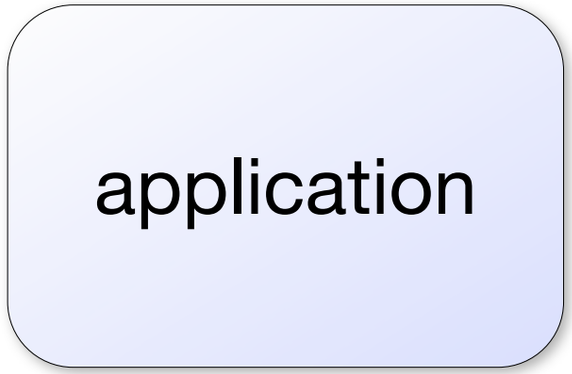


Design of CPU/FPGA Systems

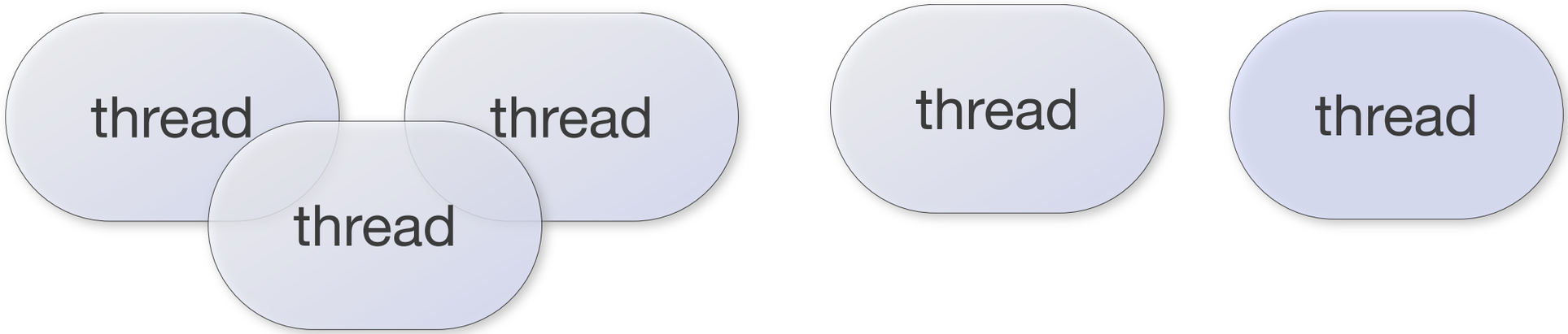
- hardware accelerators typically integrated as slave coprocessors
- hardware/software boundary explicit
- tedious to program
- portability issues



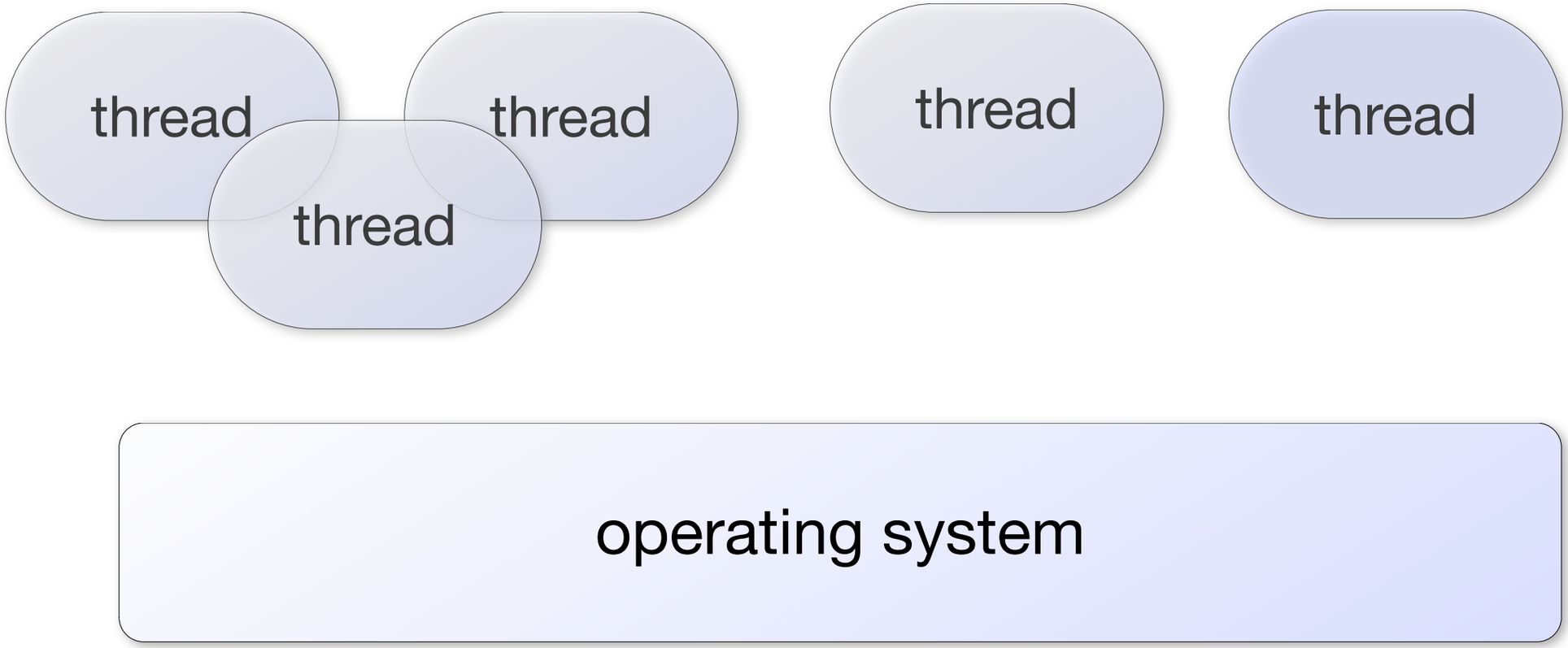
Multithreaded Programming



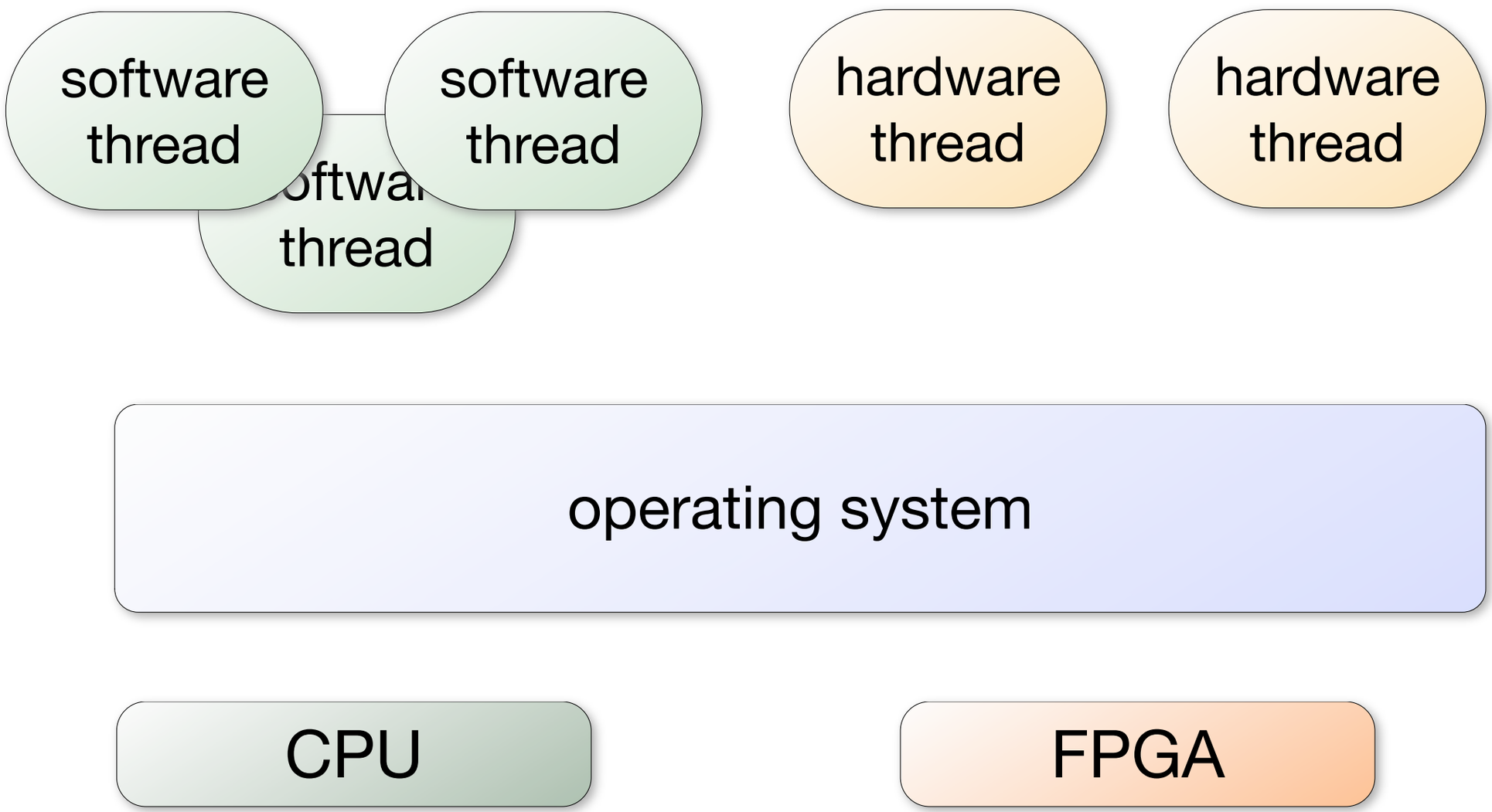
Multithreaded Programming



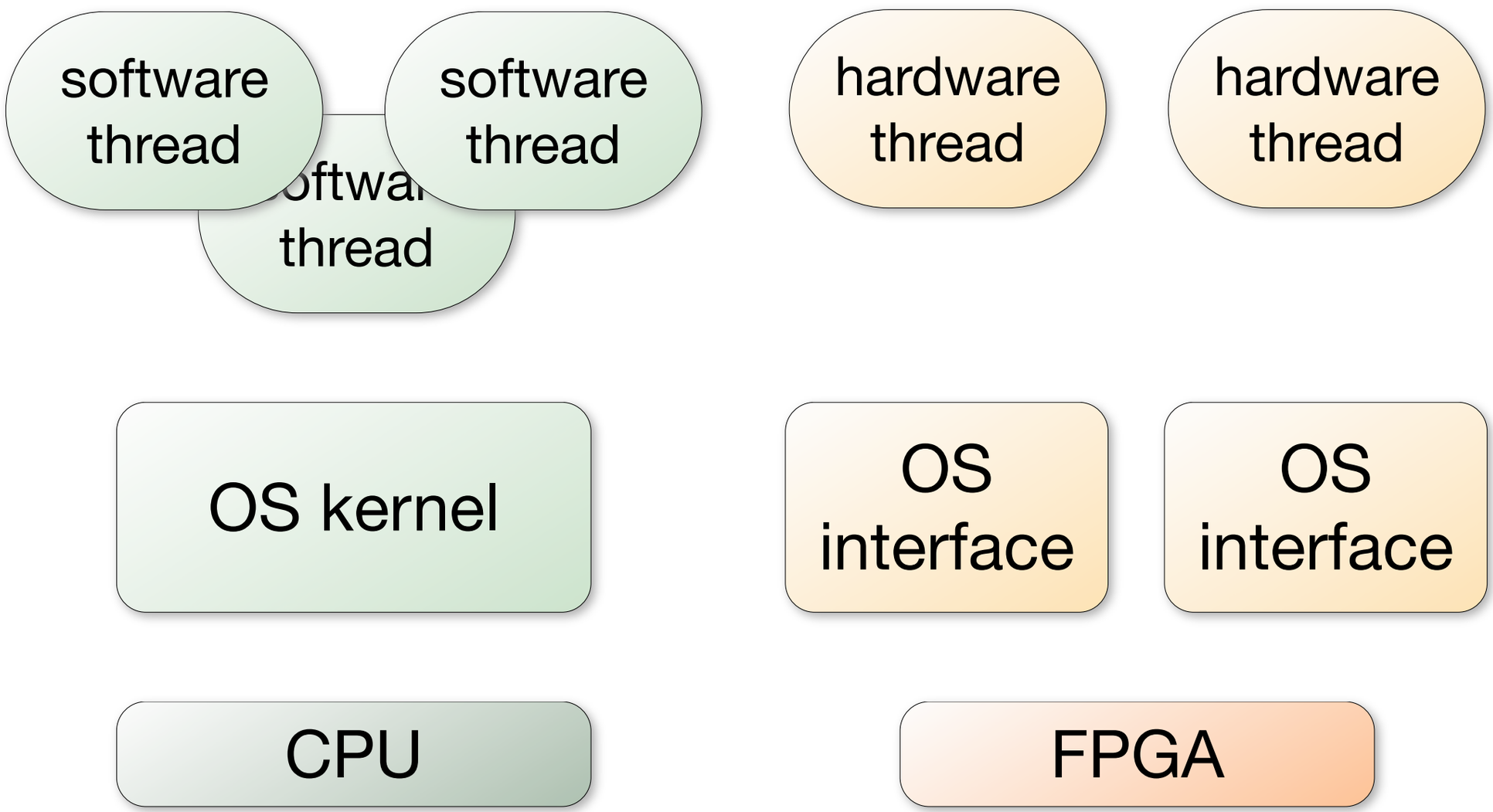
Multithreaded Programming



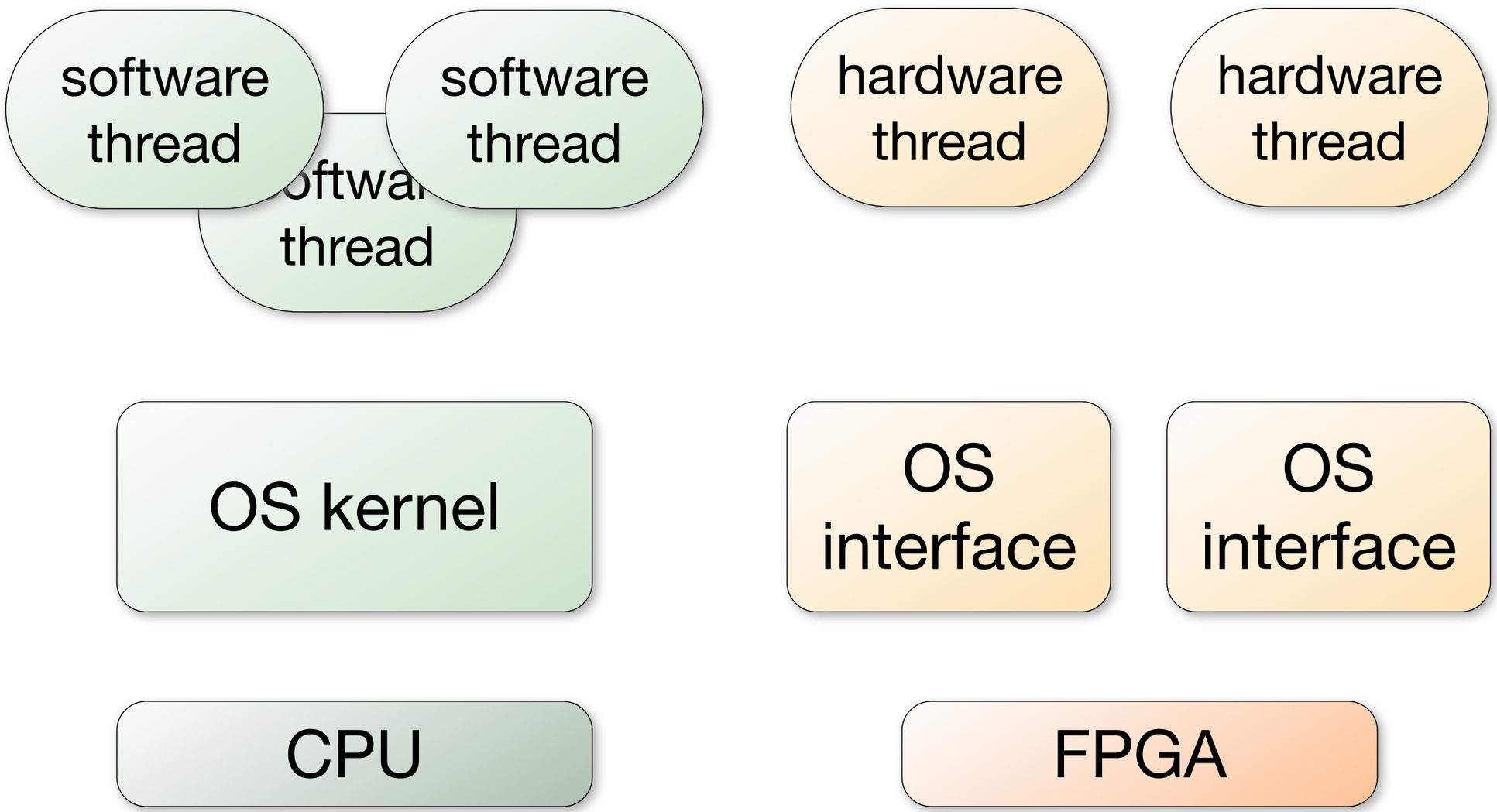
Multithreaded Programming



Multithreaded Programming

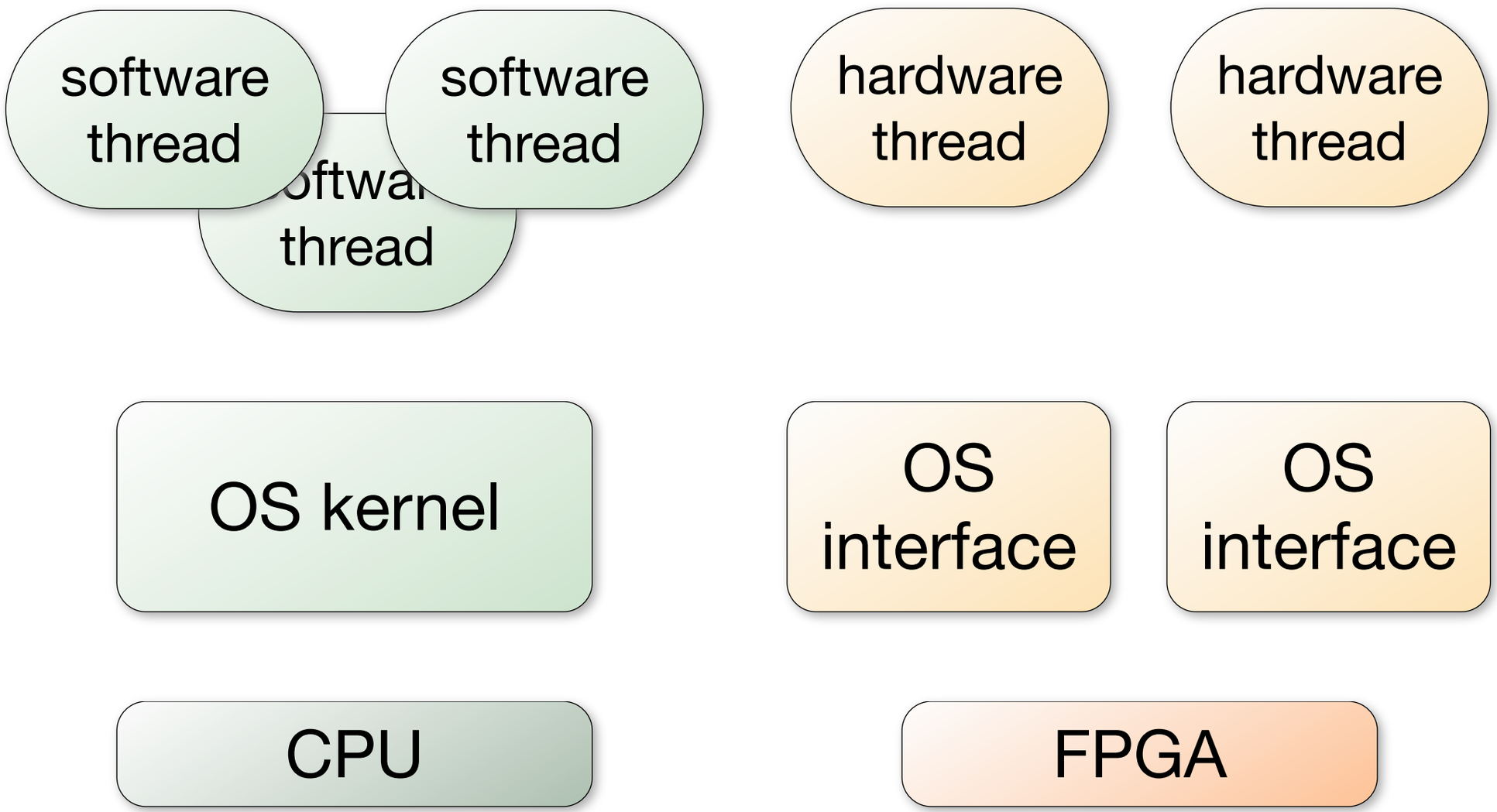


Multithreaded Programming



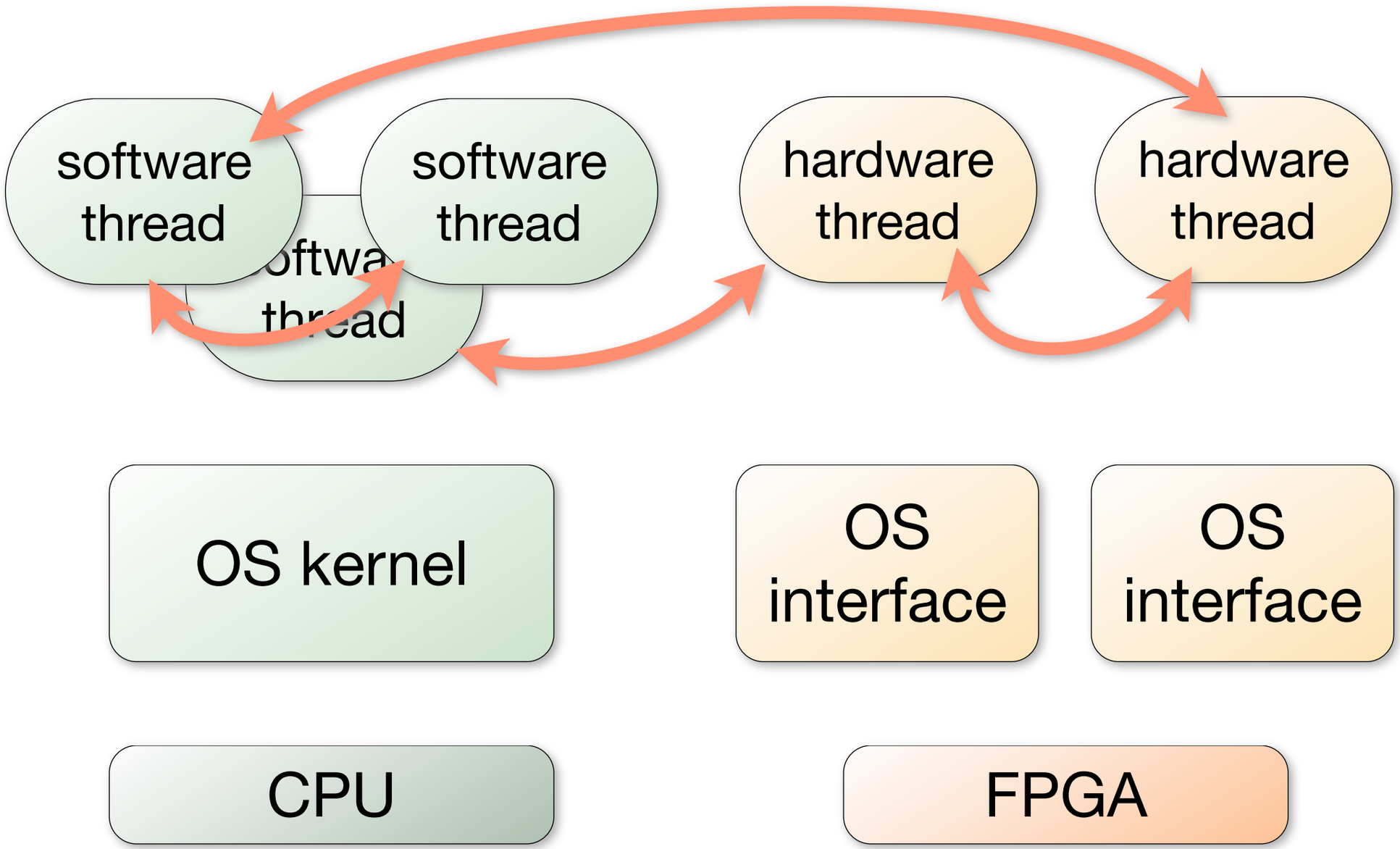
Multithreaded Programming

- communication and synchronization



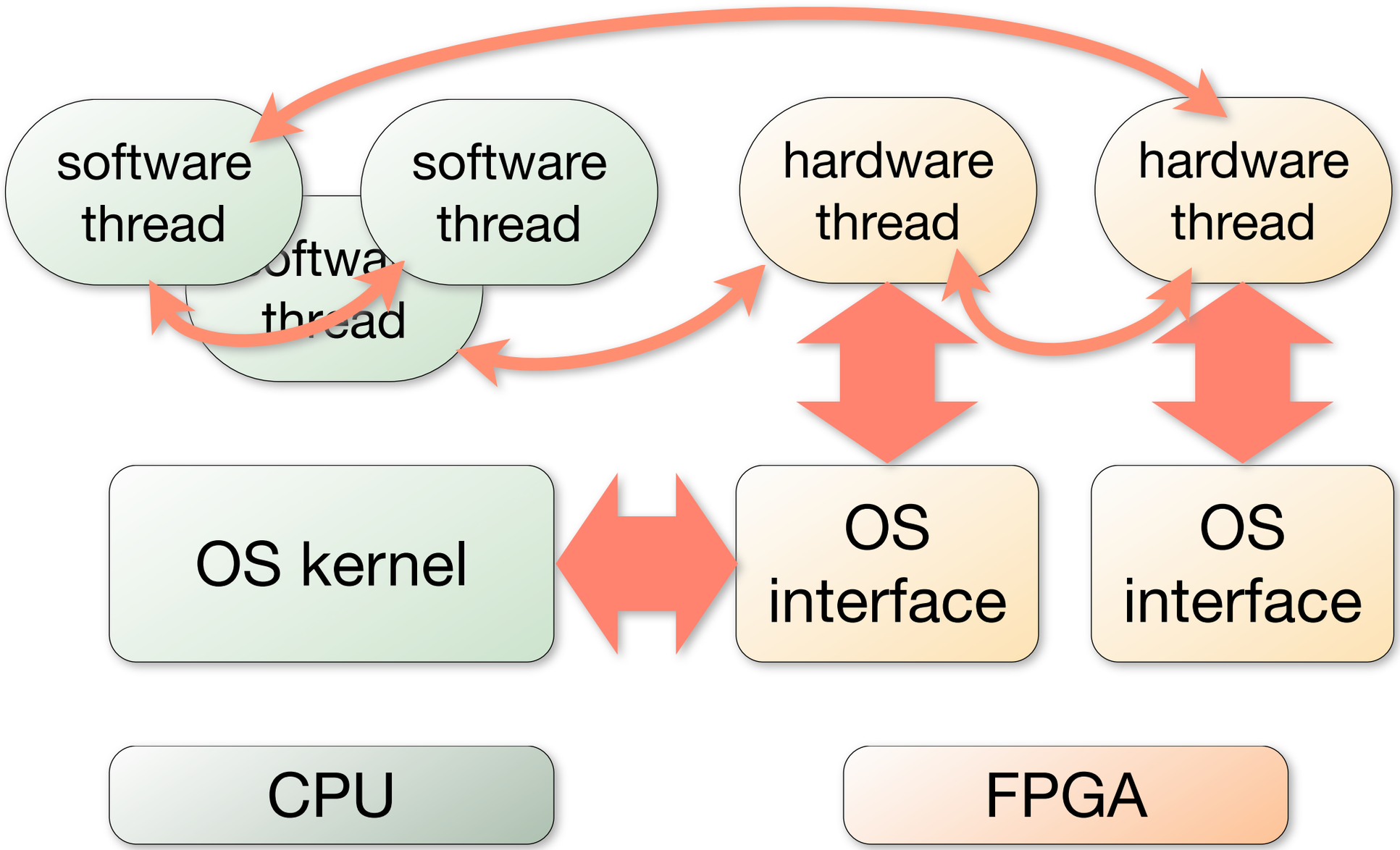
Multithreaded Programming

- communication and synchronization
  - high-level (between the threads themselves)



Multithreaded Programming

- communication and synchronization
  - high-level (between the threads themselves)
  - low-level (between hardware threads and operating system)

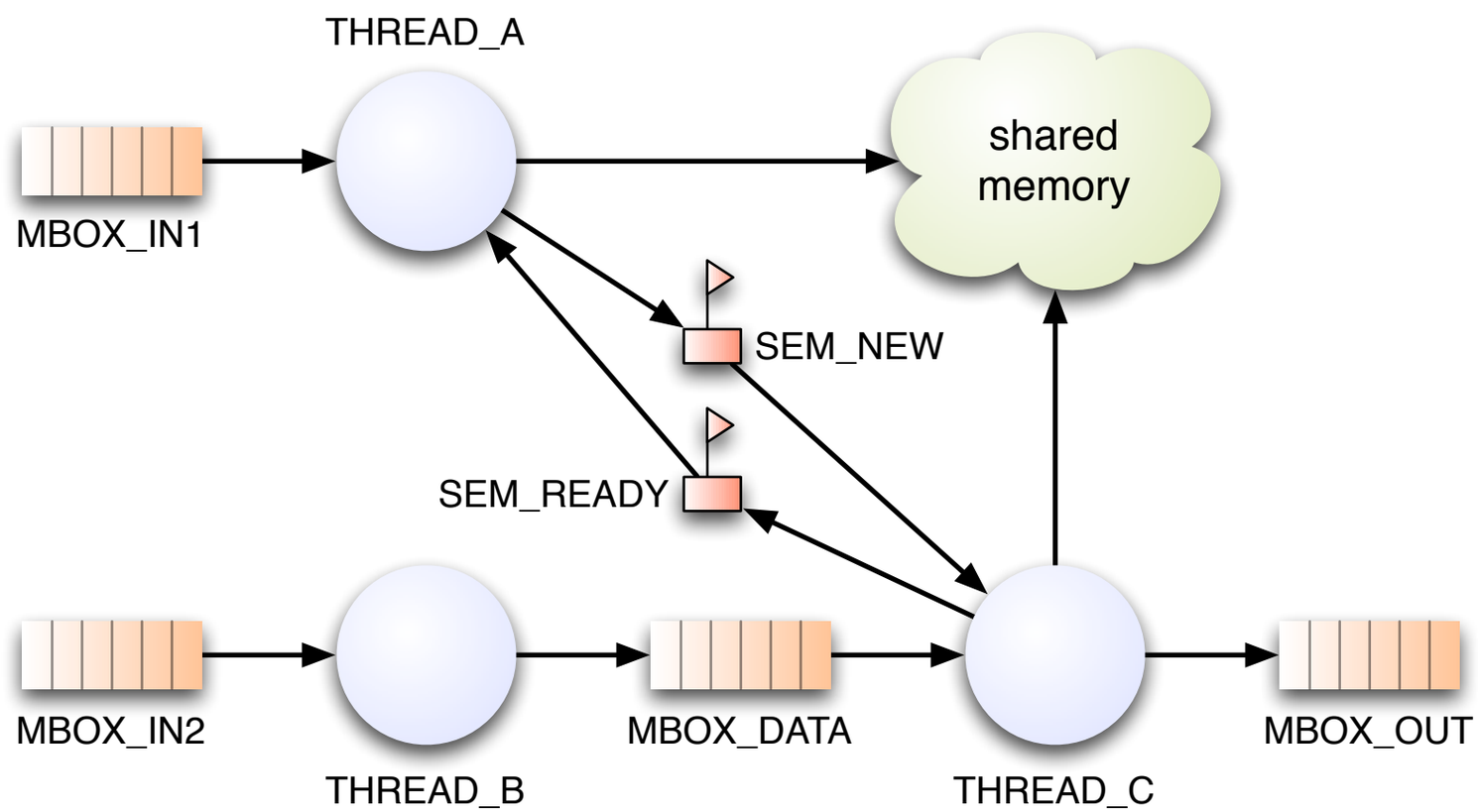


Outline

- motivation
- high-level communication and synchronization
- low-level communication and synchronization
- performance & overheads
- conclusion & outlook

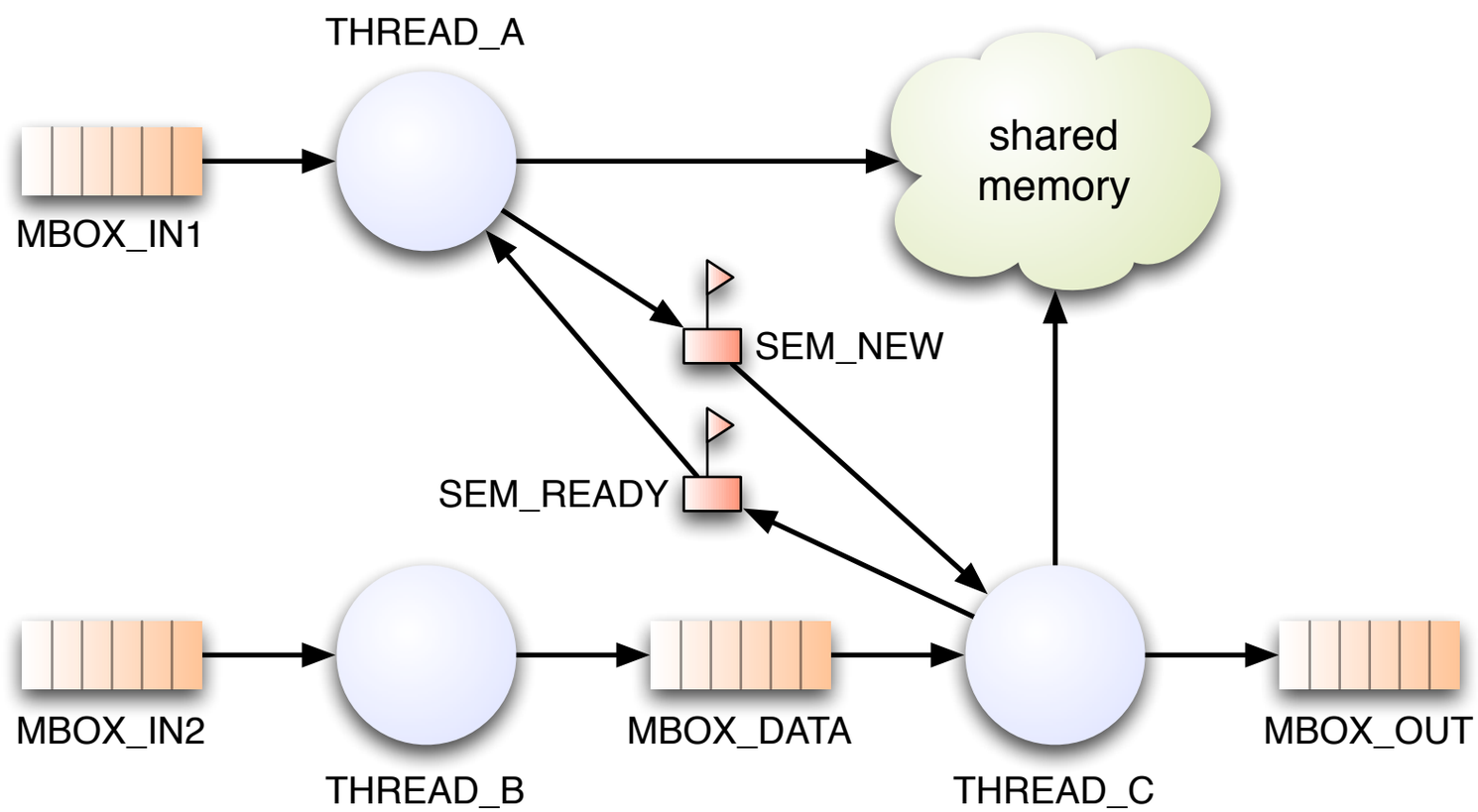
Programming Model

- applications are divided into threads
- threads communicate via operating system objects
  - semaphores
  - mailboxes
  - shared memory
  - …



Programming Model

- applications are divided into threads
- threads communicate via operating system objects
  - semaphores
  - mailboxes
  - shared memory
  - ...

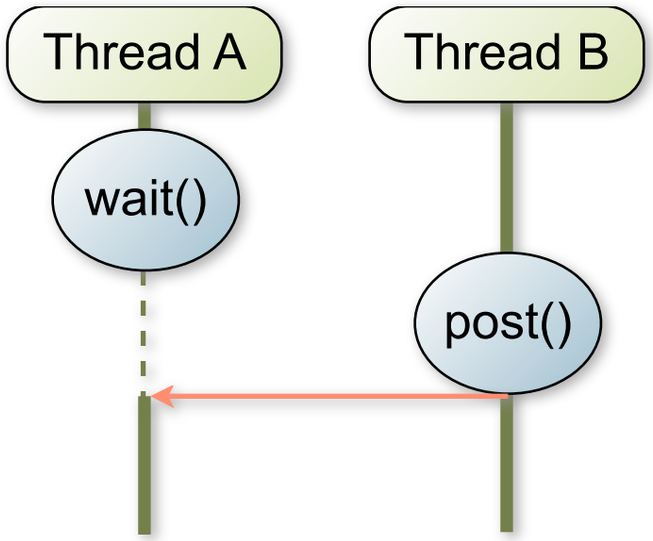


examples for API functions used by threads

software (POSIX, C)	hardware (ReconOS, VHDL)
`sem_post()`	`reconos_sem_post()`
`pthread_mutex_lock()`	`reconos_mutex_lock()`
`mq_send()`	`reconos_mbox_put()`
`value = *ptr`	`reconos_read()`
`pthread_exit()`	`reconos_thread_exit()`

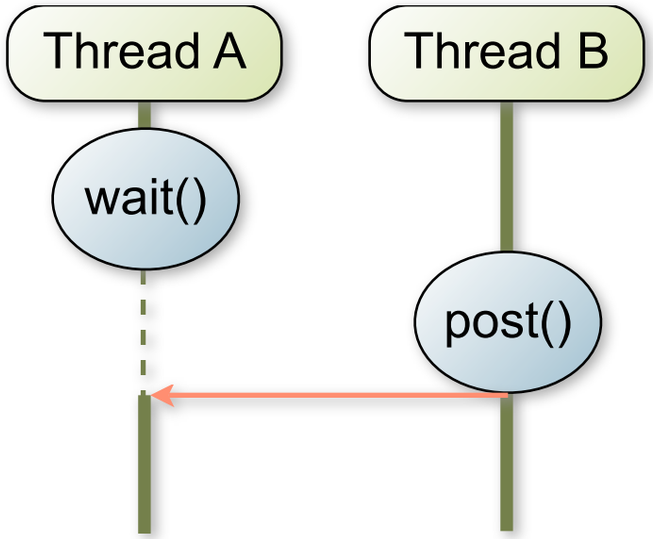
High-Level Synchronization in ReconOS

- semaphores
  - general mechanism to synchronize execution
  - blocking wait() operation, non-blocking post() operation
  - supported by **both** hardware and software threads

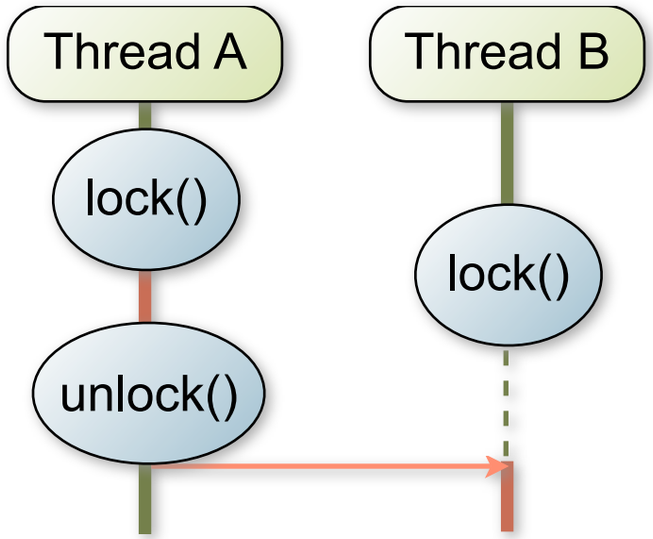


High-Level Synchronization in ReconOS

- semaphores
  - general mechanism to synchronize execution
  - blocking wait() operation, non-blocking post() operation
  - supported by **both** hardware and software threads

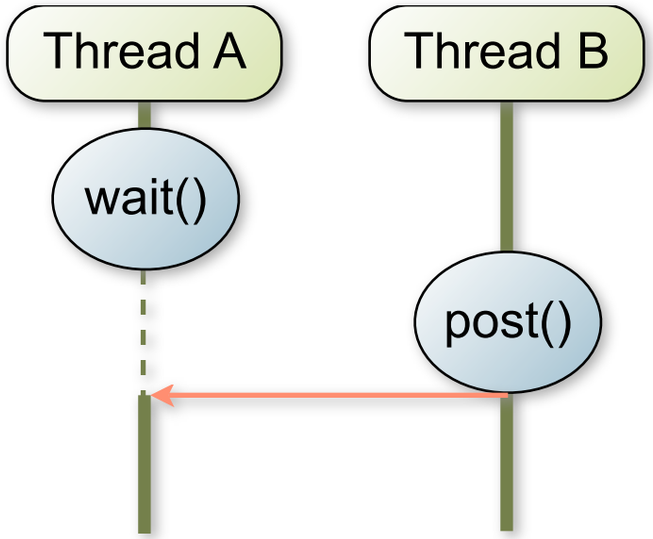


- mutexes
  - specific mechanism to protect critical sections (e.g. read-modify-write to shared memory)
  - a thread can only release a mutex it “owns” (has previously locked)
  - supported by **both** hardware and software threads

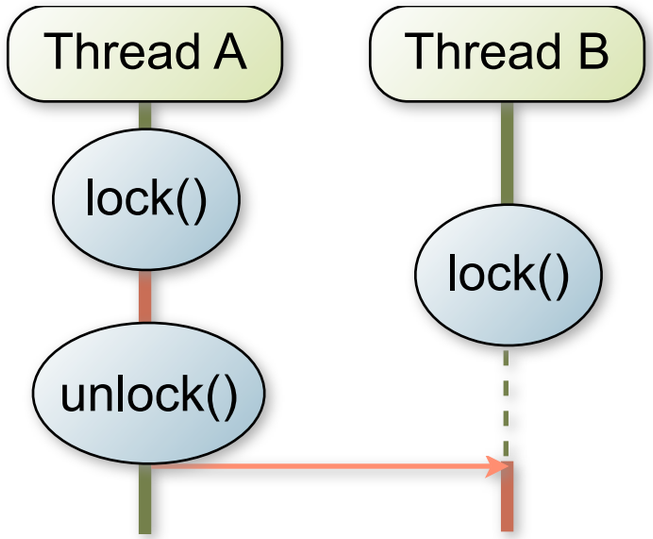


High-Level Synchronization in ReconOS

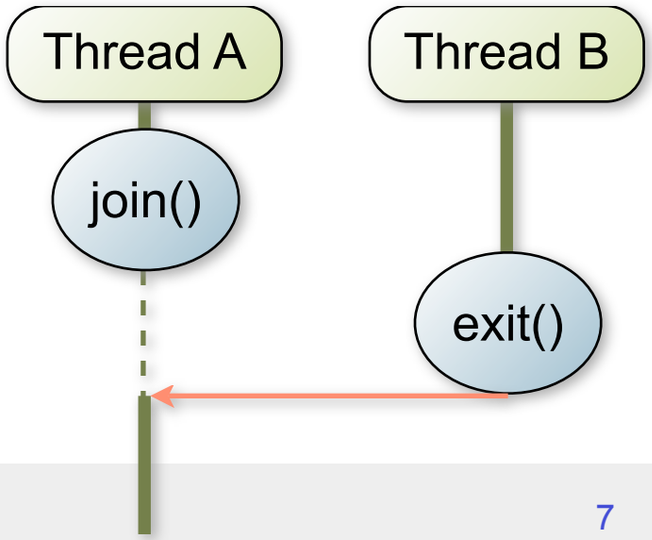
- semaphores
  - general mechanism to synchronize execution
  - blocking wait() operation, non-blocking post() operation
  - supported by **both** hardware and software threads



- mutexes
  - specific mechanism to protect critical sections (e.g. read-modify-write to shared memory)
  - a thread can only release a mutex it “owns” (has previously locked)
  - supported by **both** hardware and software threads

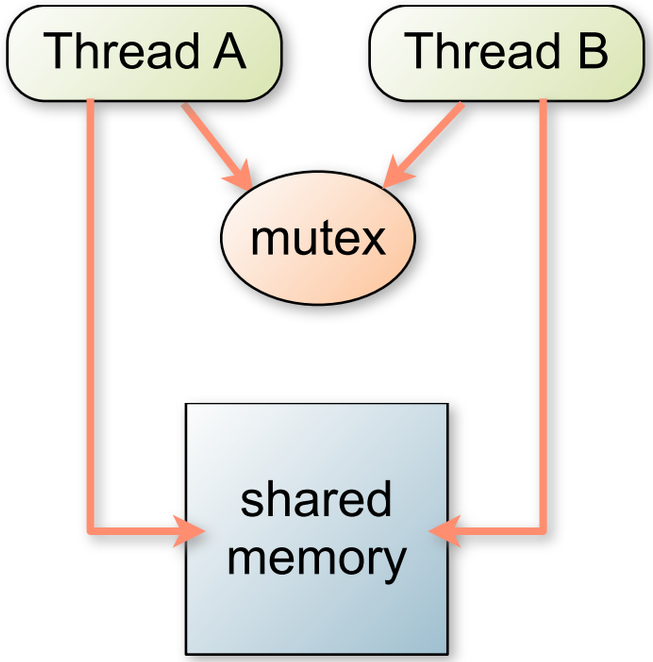


- thread termination
  - a thread can block until another thread exits
  - currently, only software threads can join(), but all threads can exit()



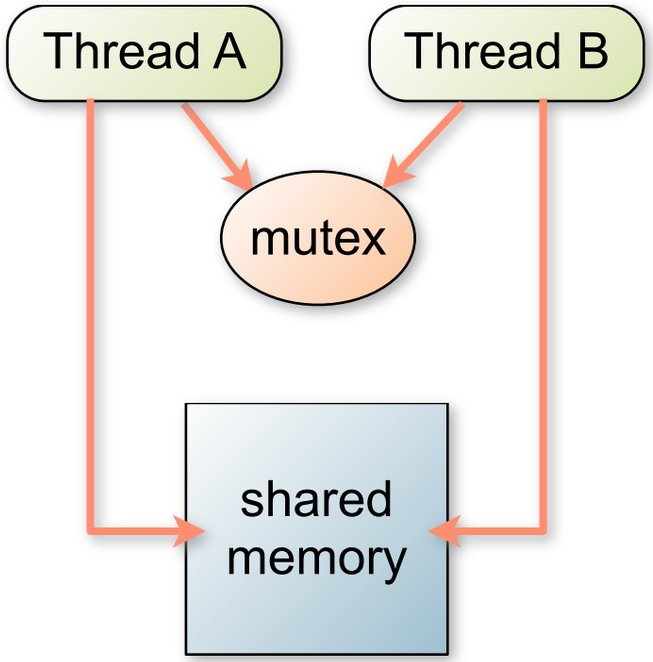
High-Level Communication in ReconOS

- shared memory
  - all threads have direct access to the entire memory space
  - accesses need to be synchronized using semaphores or mutexes
  - dedicated hardware support for burst transfers

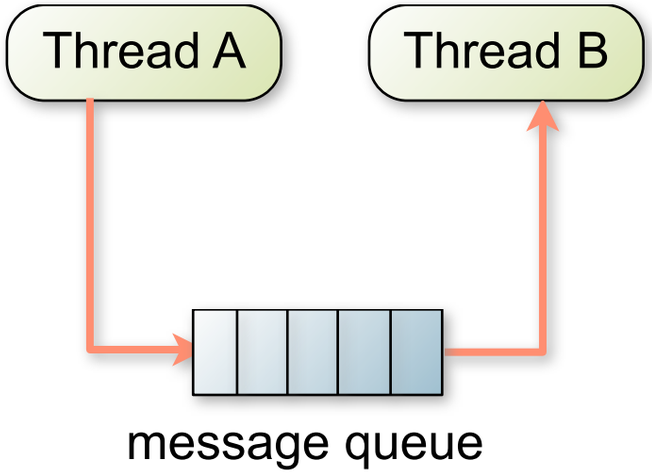


High-Level Communication in ReconOS

- shared memory
  - all threads have direct access to the entire memory space
  - accesses need to be synchronized using semaphores or mutexes
  - dedicated hardware support for burst transfers



- message queues
  - can block if queue is empty / full
  - combined communication and synchronization primitive
  - supported by both hardware and software threads
  - dedicated hardware FIFOs for hardware threads

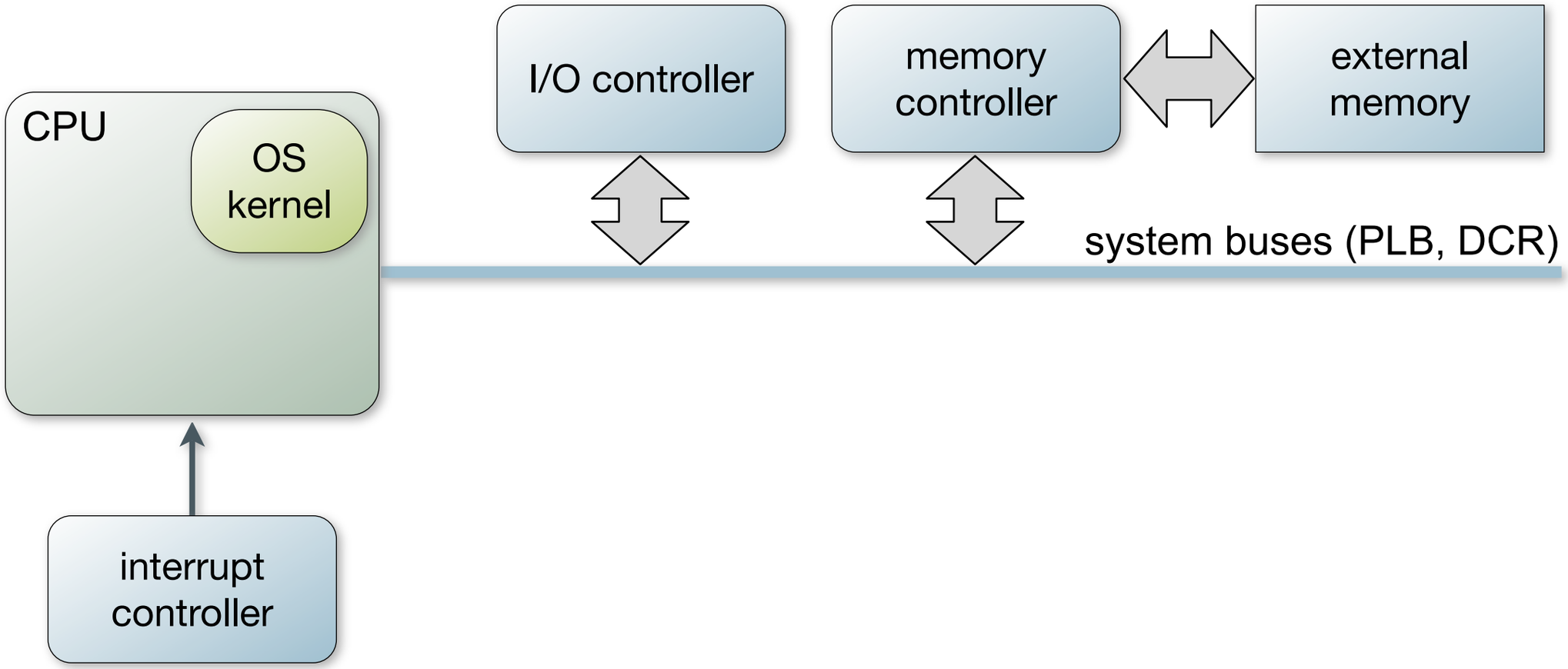


Outline

- motivation
- high-level communication and synchronization
- low-level communication and synchronization
- performance & overheads
- conclusion & outlook

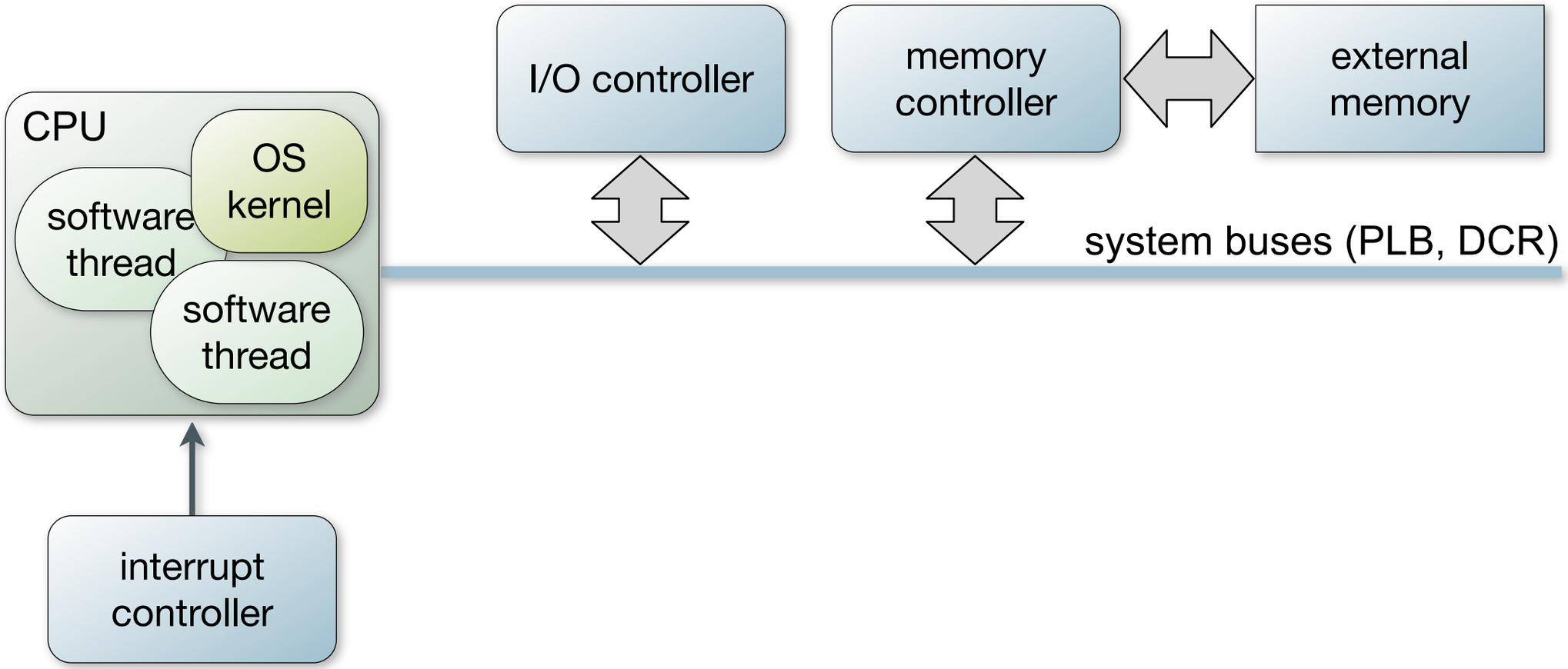
Hardware Architecture

- developed on Xilinx Virtex-II Pro, Virtex-4 FX FPGAs
- based on CoreConnect bus topology
- OS kernel is eCos for PowerPC ported to Virtex



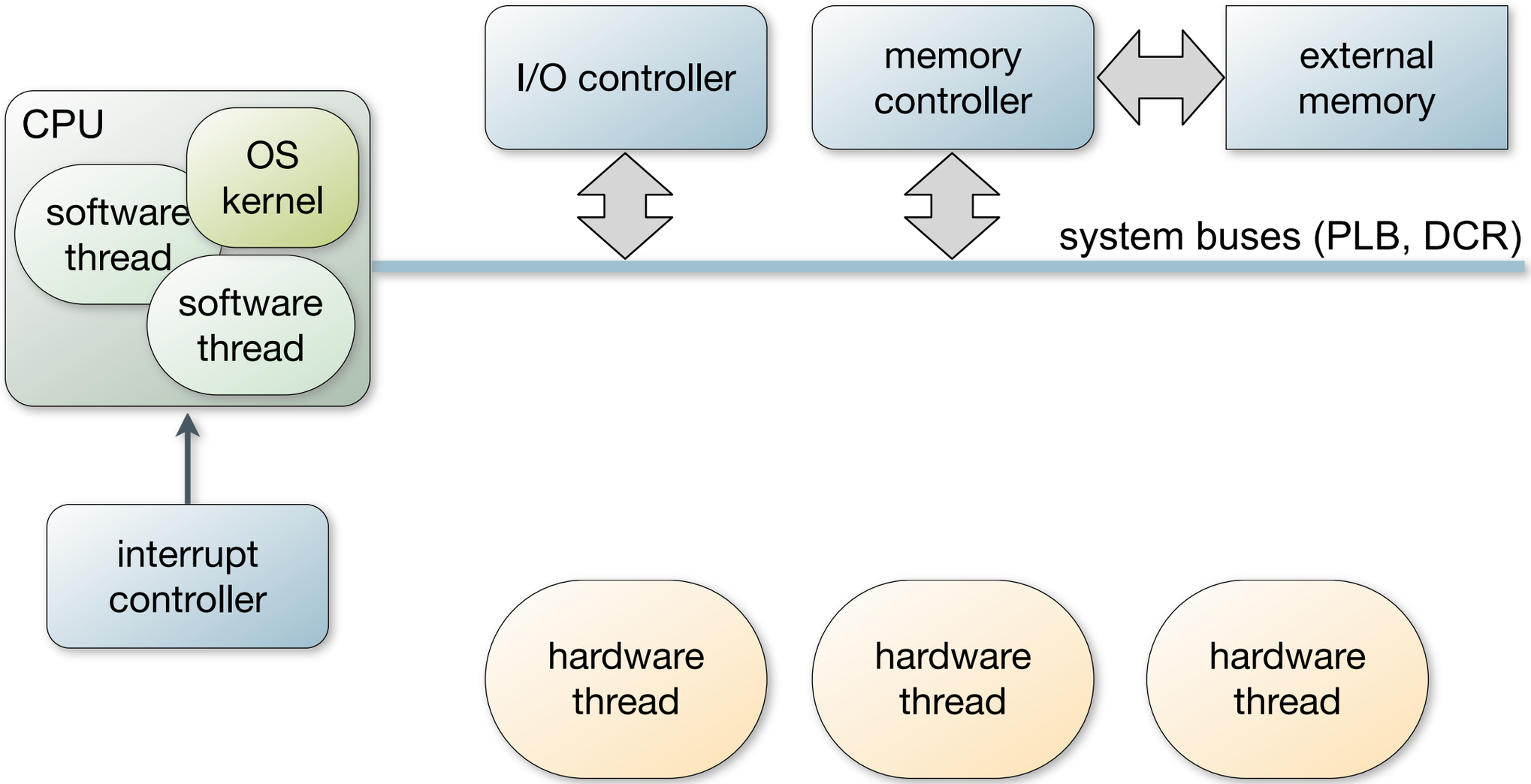
Hardware Architecture

- developed on Xilinx Virtex-II Pro, Virtex-4 FX FPGAs
- based on CoreConnect bus topology
- OS kernel is eCos for PowerPC ported to Virtex



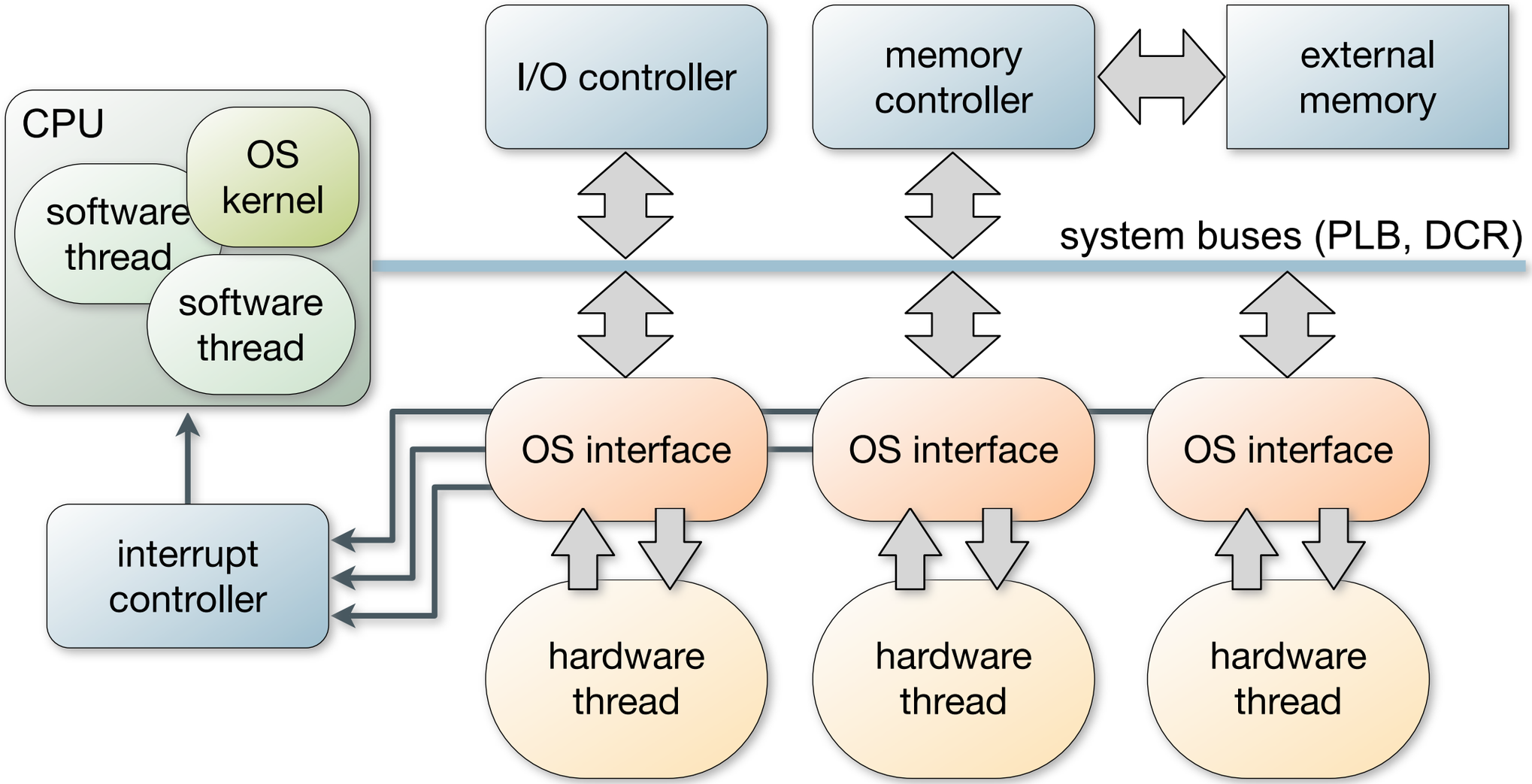
Hardware Architecture

- developed on Xilinx Virtex-II Pro, Virtex-4 FX FPGAs
- based on CoreConnect bus topology
- OS kernel is eCos for PowerPC ported to Virtex

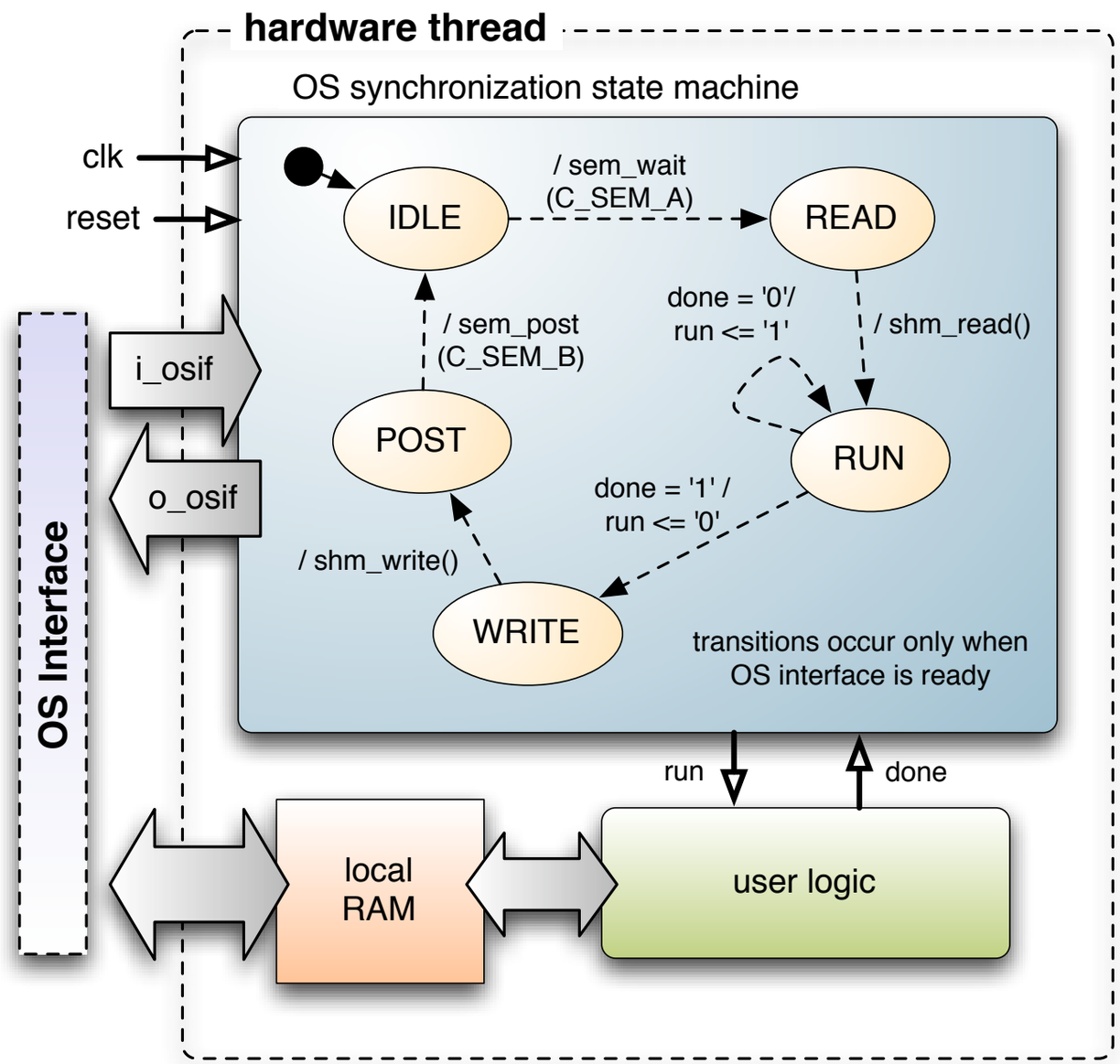


Hardware Architecture

- developed on Xilinx Virtex-II Pro, Virtex-4 FX FPGAs
- based on CoreConnect bus topology
- OS kernel is eCos for PowerPC ported to Virtex

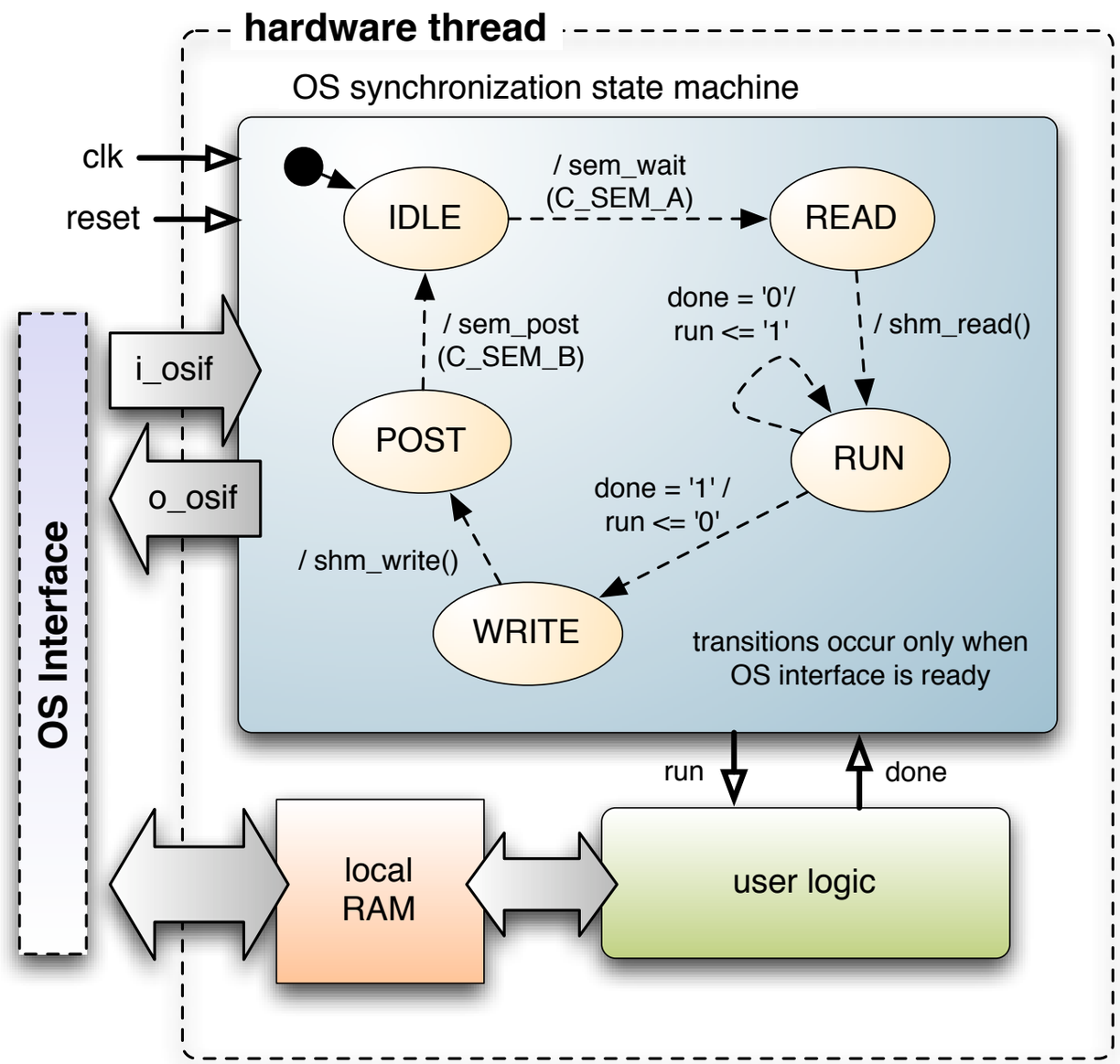


Hardware Thread



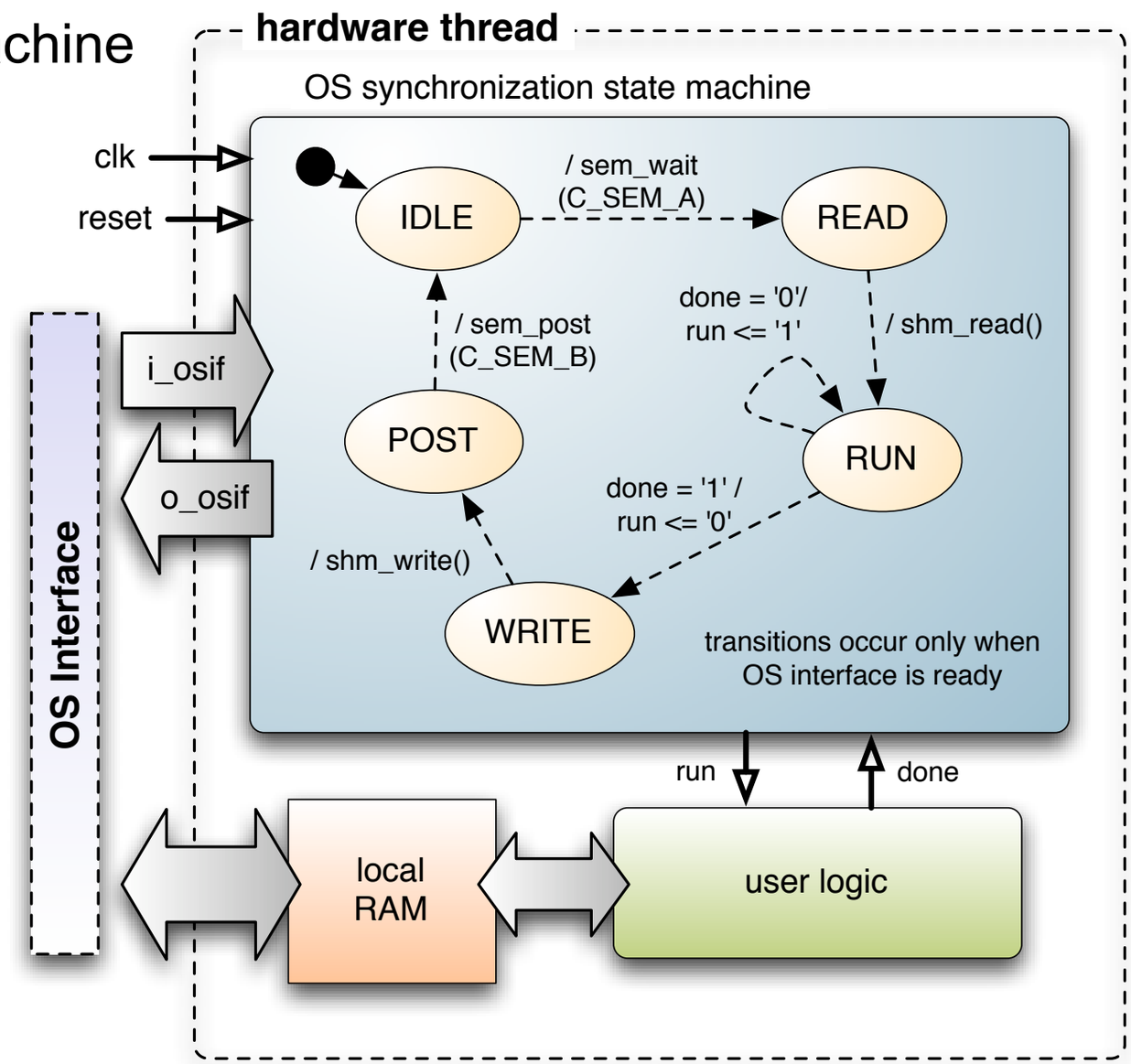
Hardware Thread

- a hardware thread consists of two parts



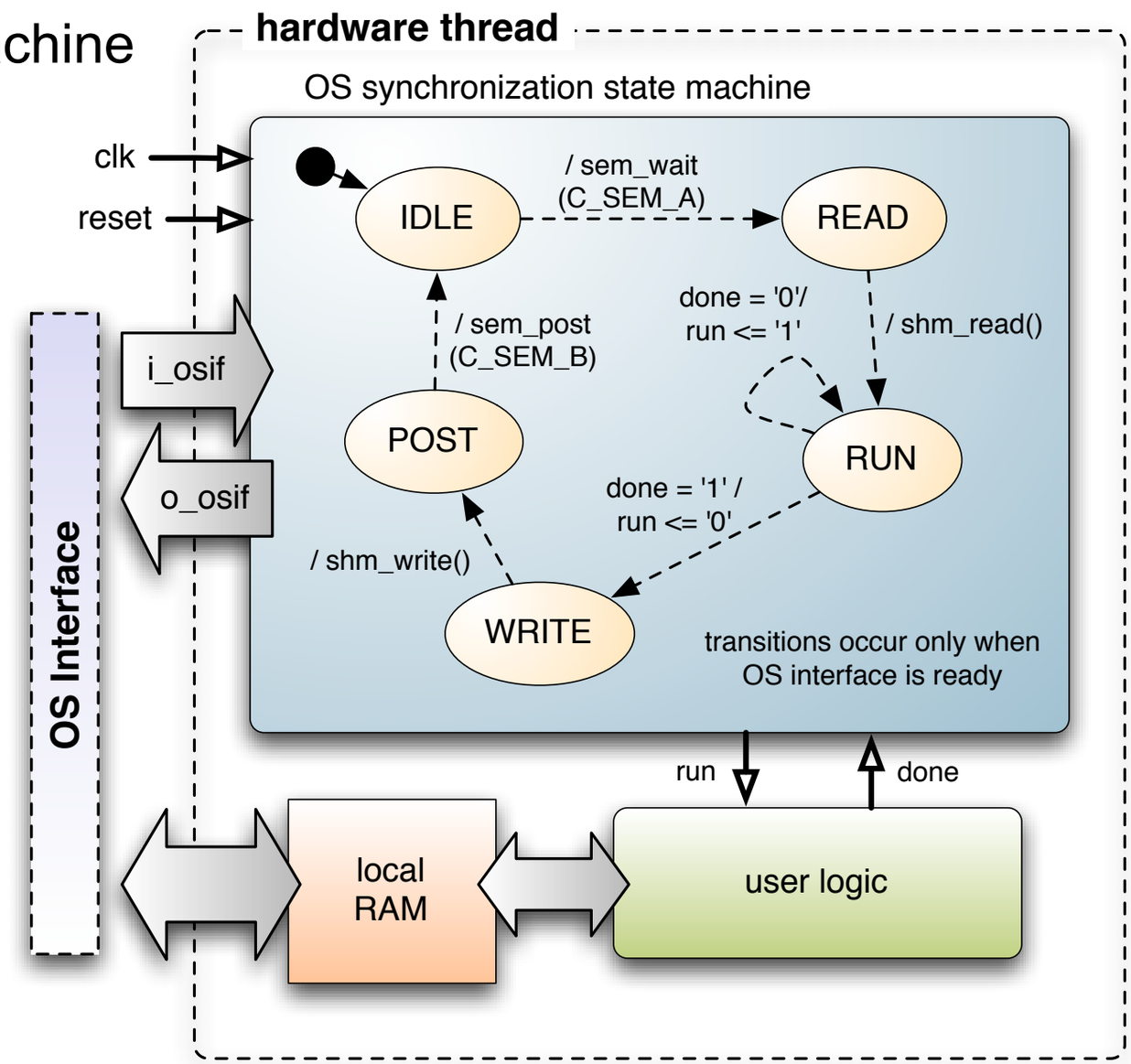
Hardware Thread

- a hardware thread consists of two parts
  - an OS synchronization state machine
    - synchronizes thread with operating system calls
    - serializes access to OS objects via the OS interface
    - can be blocked by the OS interface



Hardware Thread

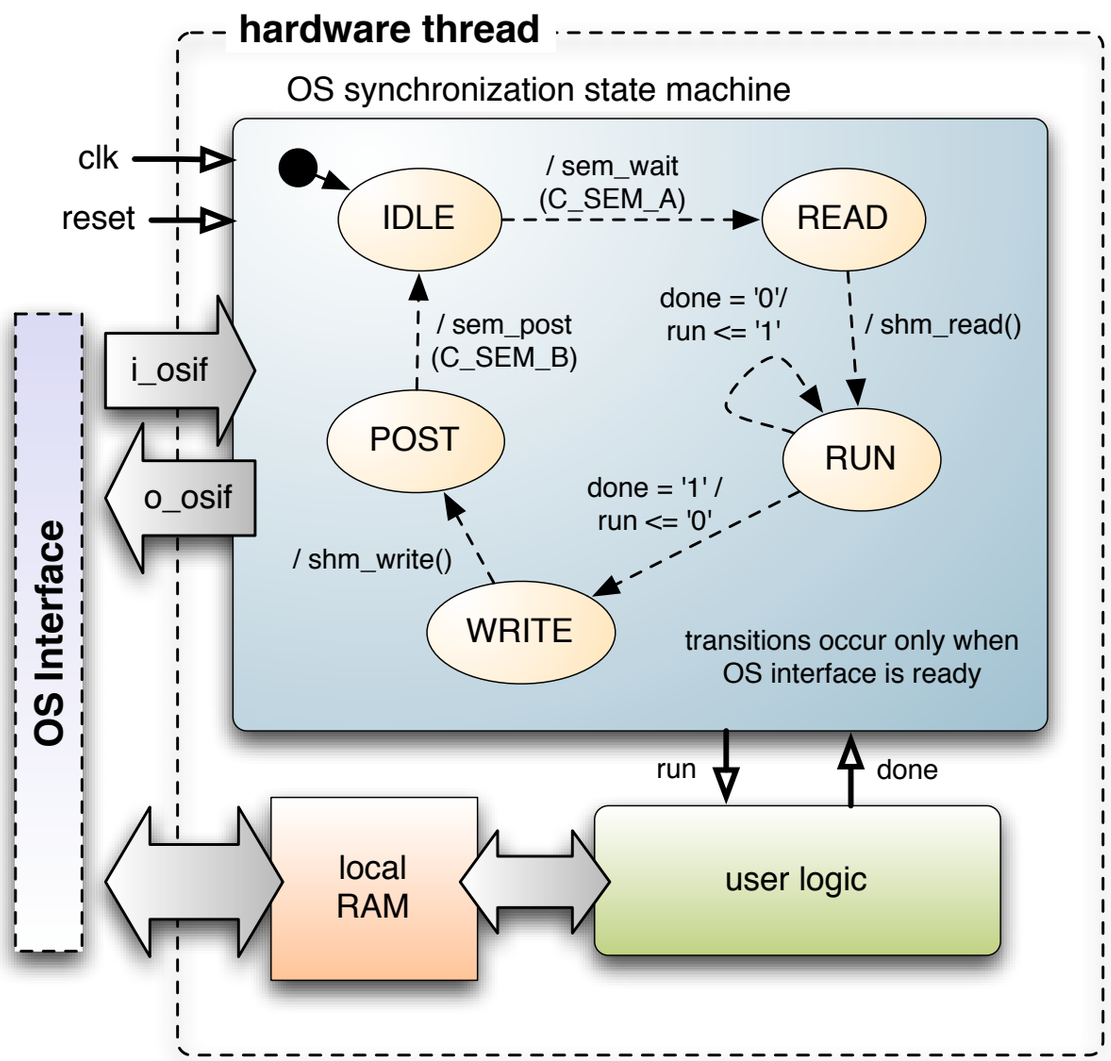
- a hardware thread consists of two parts
  - an OS synchronization state machine
    - synchronizes thread with operating system calls
    - serializes access to OS objects via the OS interface
    - can be blocked by the OS interface
  - parallel “user processes”
    - communicate with OS synchronization state machine
    - can directly access local memory blocks
    - are not necessarily blocked



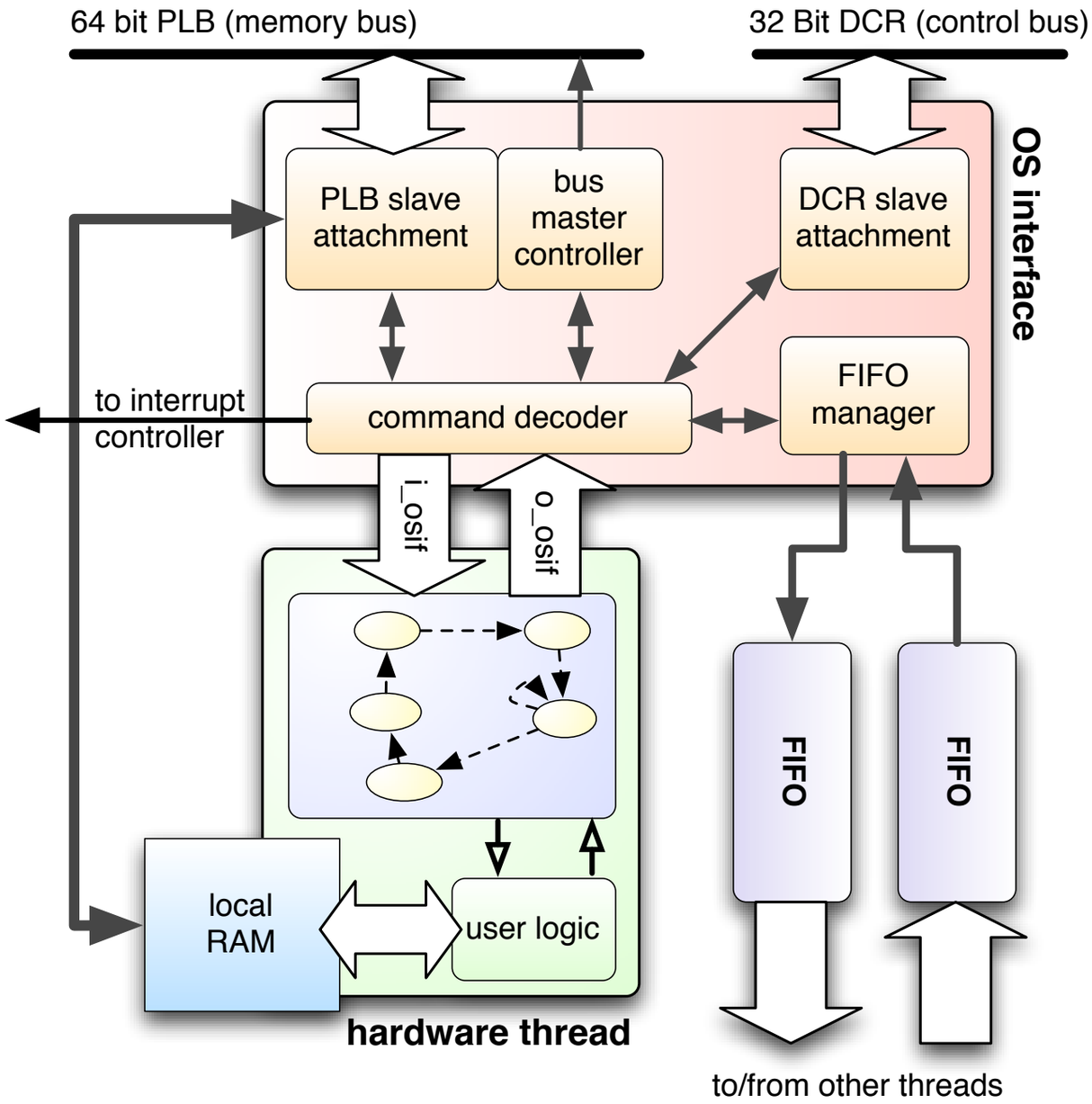
ReconOS API for Hardware Threads

```
1   osif_fsm: process(clk, reset)
2   begin
3     if (reset = '1') then
4       state <= IDLE;
5       run <= '0';
6       reconos_reset(o_osif, i_osif);
7     elsif rising_edge(clk) then
8       reconos_begin(o_osif, i_osif);
9       if reconos_ready(i_osif) then
10        case state is
11          when IDLE =>
12            reconos_sem_wait(o_osif, i_osif, C_SEM_A);
13            state <= READ;
14
15          when READ =>
16            reconos_shm_read_burst(o_osif, i_osif,
17                                   local_address,
18                                   global_address);
19            state <= RUN;
20
21          when RUN =>
22            run <= '1';
23            if done = '1' then
24              run <= '0';
25              state <= WRITE;
26            end if;
27
28          when WRITE =>
29            reconos_shm_write_burst(o_osif, i_osif,
30                                    local_address,
31                                    global_address);
32            state <= POST;
33
34          when POST =>
35            reconos_sem_post(o_osif, i_osif, C_SEM_B);
36            state <= IDLE;
37
38          when others => null;
39        end case;
40      end if;
41    end if;
42  end process;
```

- VHDL function library
- used similar to software API
- may only be used inside OS synchronization state machine

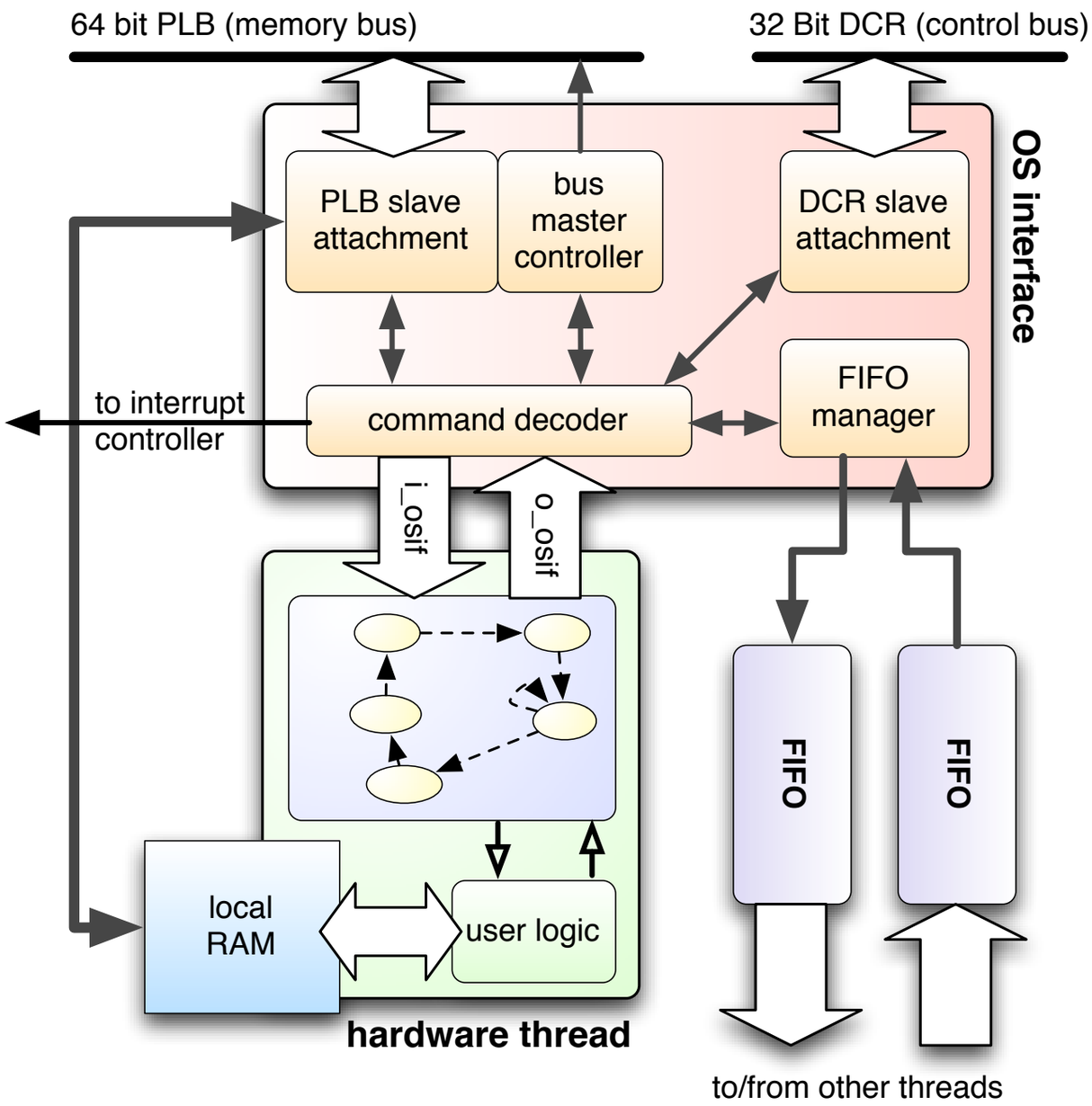


OS Interface



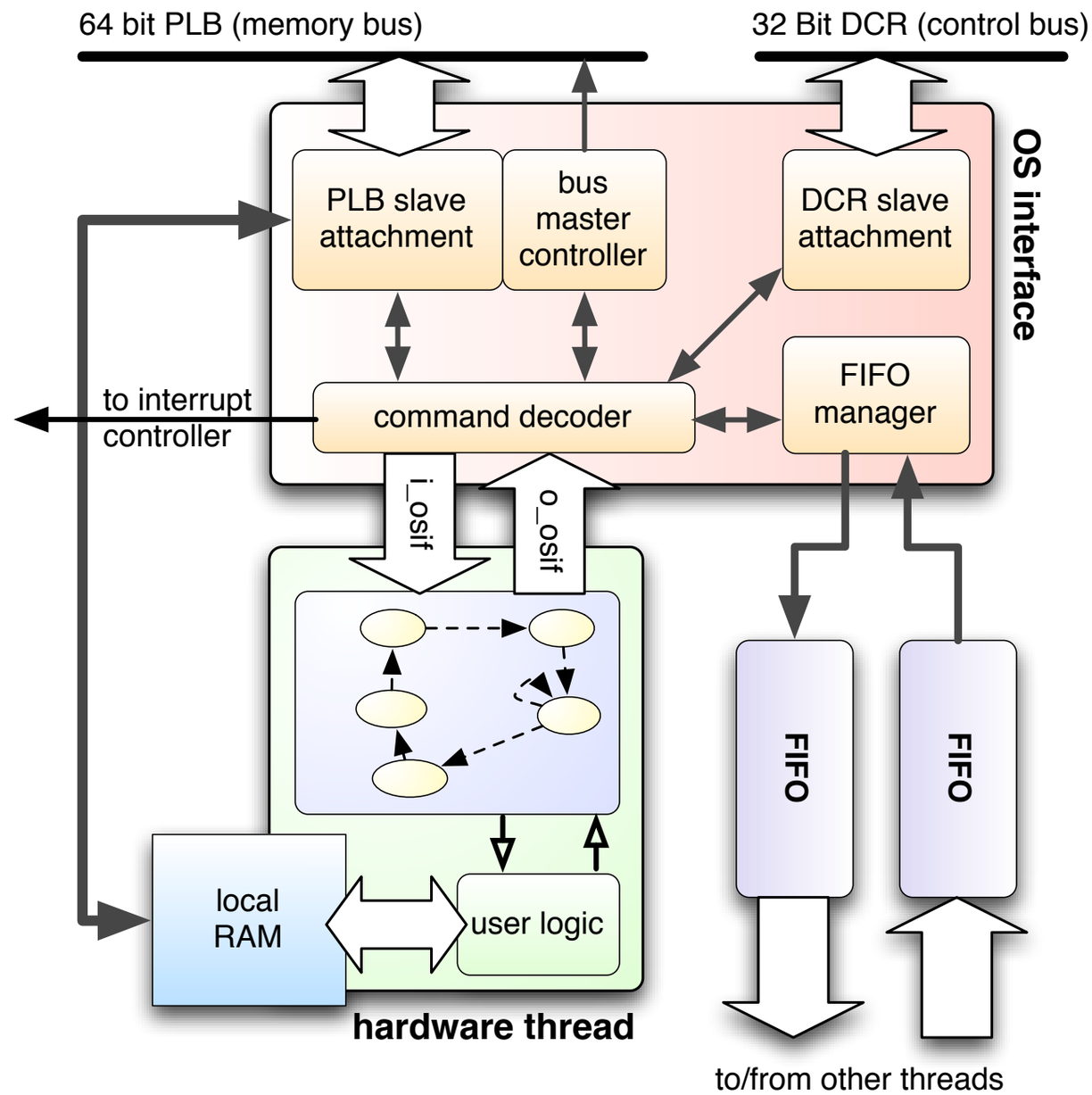
OS Interface

- processes requests from hardware thread
  - handles blocking and resuming of hardware thread



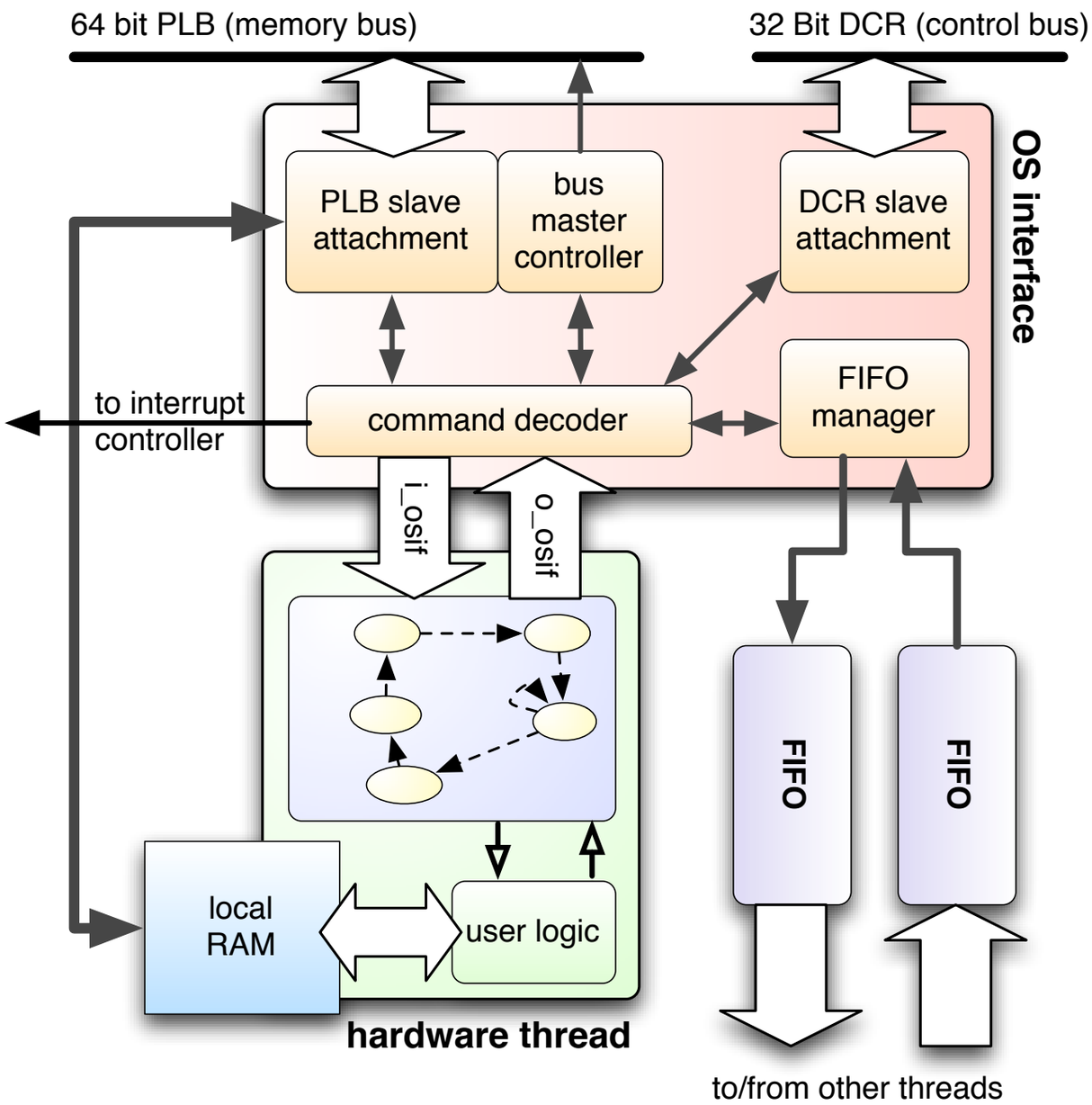
OS Interface

- processes requests from hardware thread
  - handles blocking and resuming of hardware thread
- relays OS object interactions to CPU
  - DCR interface with bus-addressable registers
  - dedicated interrupt



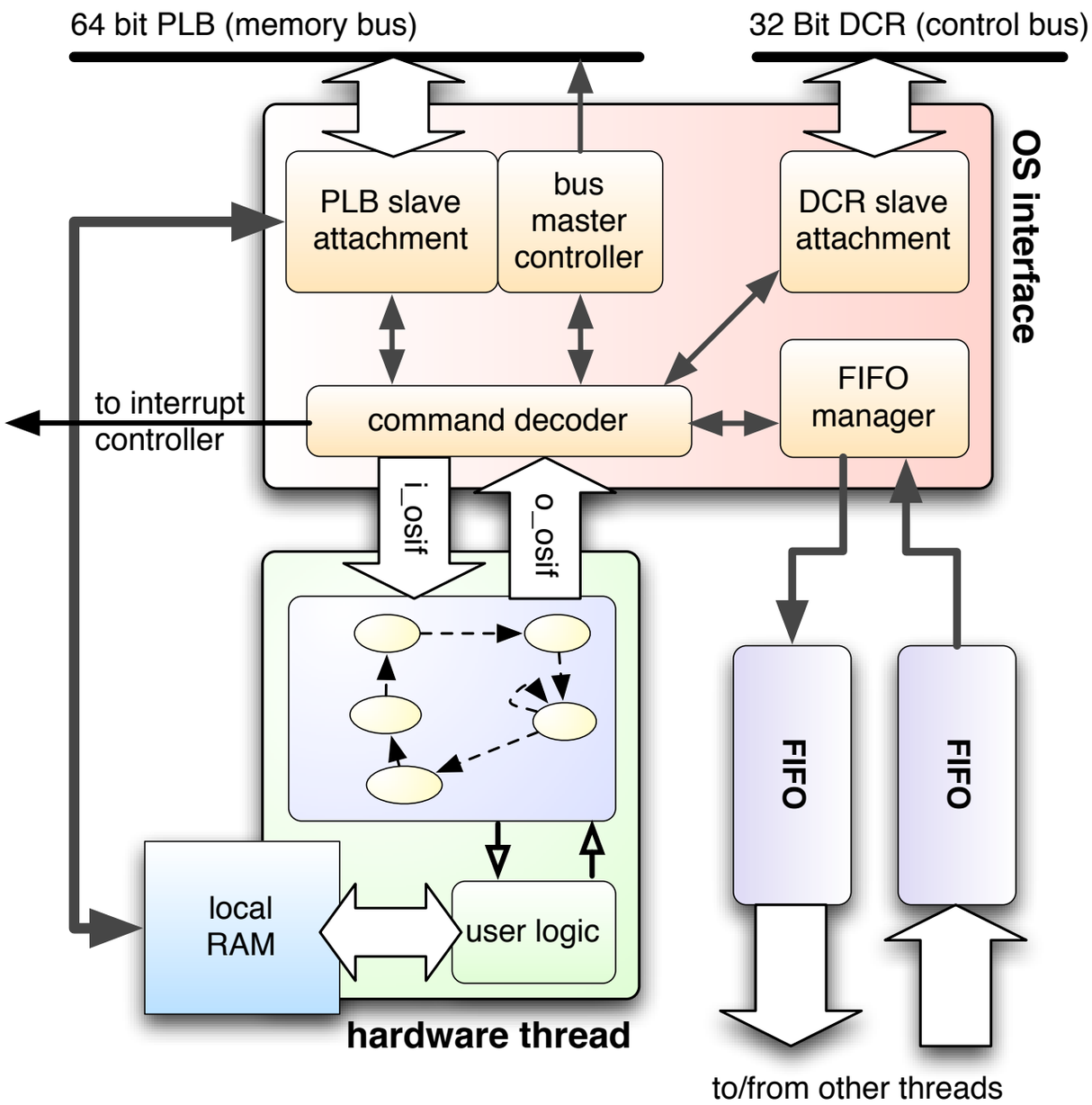
OS Interface

- processes requests from hardware thread
  - handles blocking and resuming of hardware thread
- relays OS object interactions to CPU
  - DCR interface with bus-addressable registers
  - dedicated interrupt
- executes memory accesses
  - PLB master interface
  - direct access to entire system's address space (memory and peripherals)

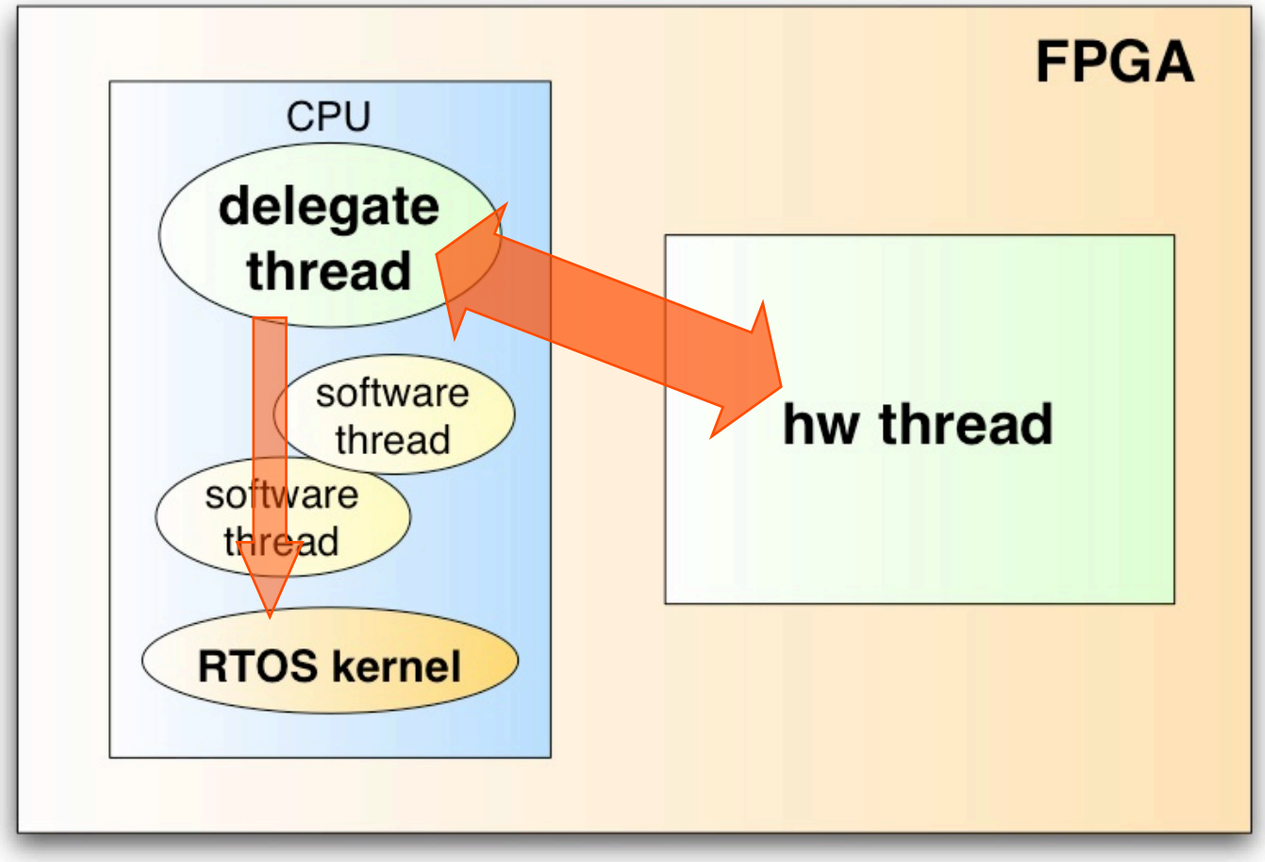


OS Interface

- processes requests from hardware thread
  - handles blocking and resuming of hardware thread
- relays OS object interactions to CPU
  - DCR interface with bus-addressable registers
  - dedicated interrupt
- executes memory accesses
  - PLB master interface
  - direct access to entire system's address space (memory and peripherals)
- dedicated FIFO channels
  - provide high-throughput hardware support for message passing

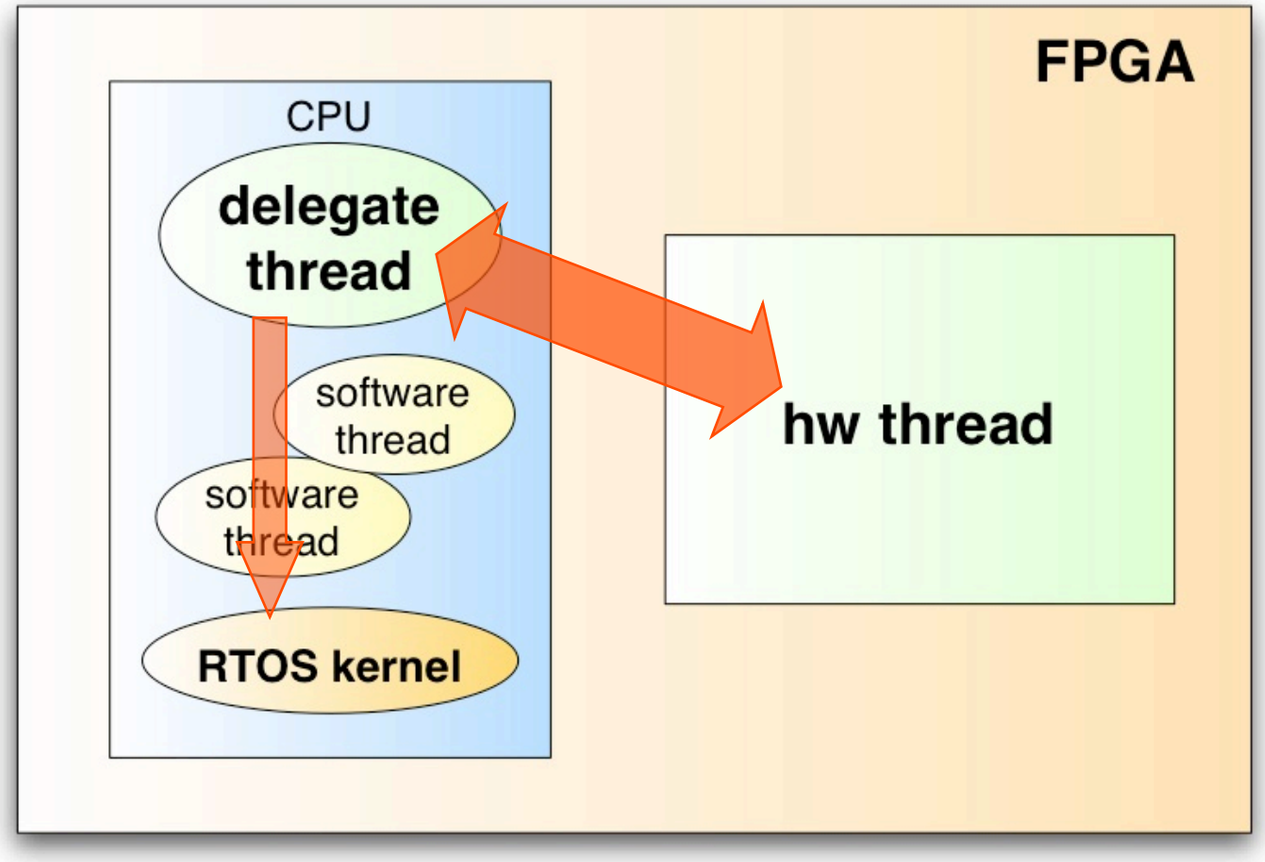


Delegate Threads



Delegate Threads

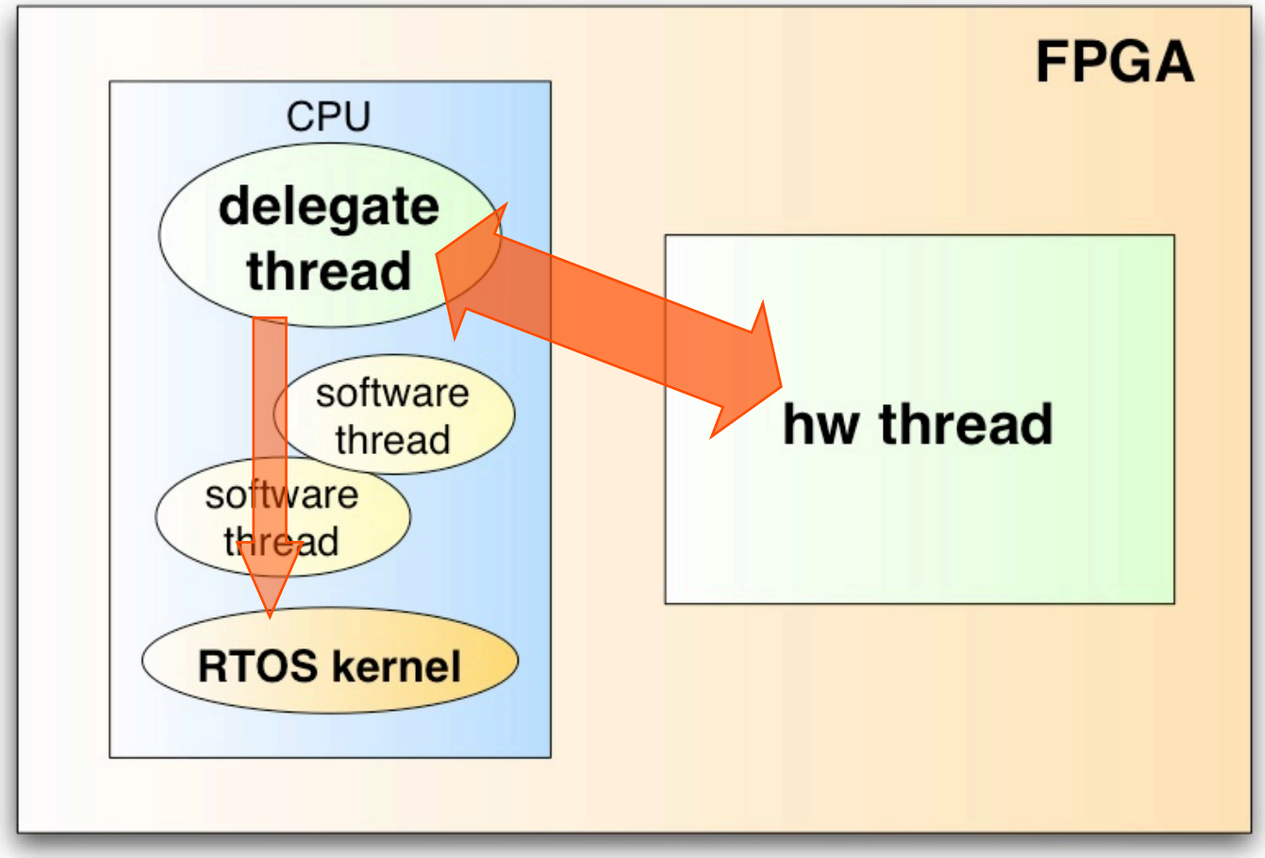
- basic mechanism
  - a delegate thread in software is associated with every hardware thread
  - the delegate thread calls the OS kernel on behalf of the hardware thread
  - all kernel responses are relayed back to the hardware thread



Delegate Threads

- basic mechanism
  - a delegate thread in software is associated with every hardware thread
  - the delegate thread calls the OS kernel on behalf of the hardware thread
  - all kernel responses are relayed back to the hardware thread

- advantages
  - no modification of the kernel required
  - extremely flexible
  - transparent to kernel and other threads

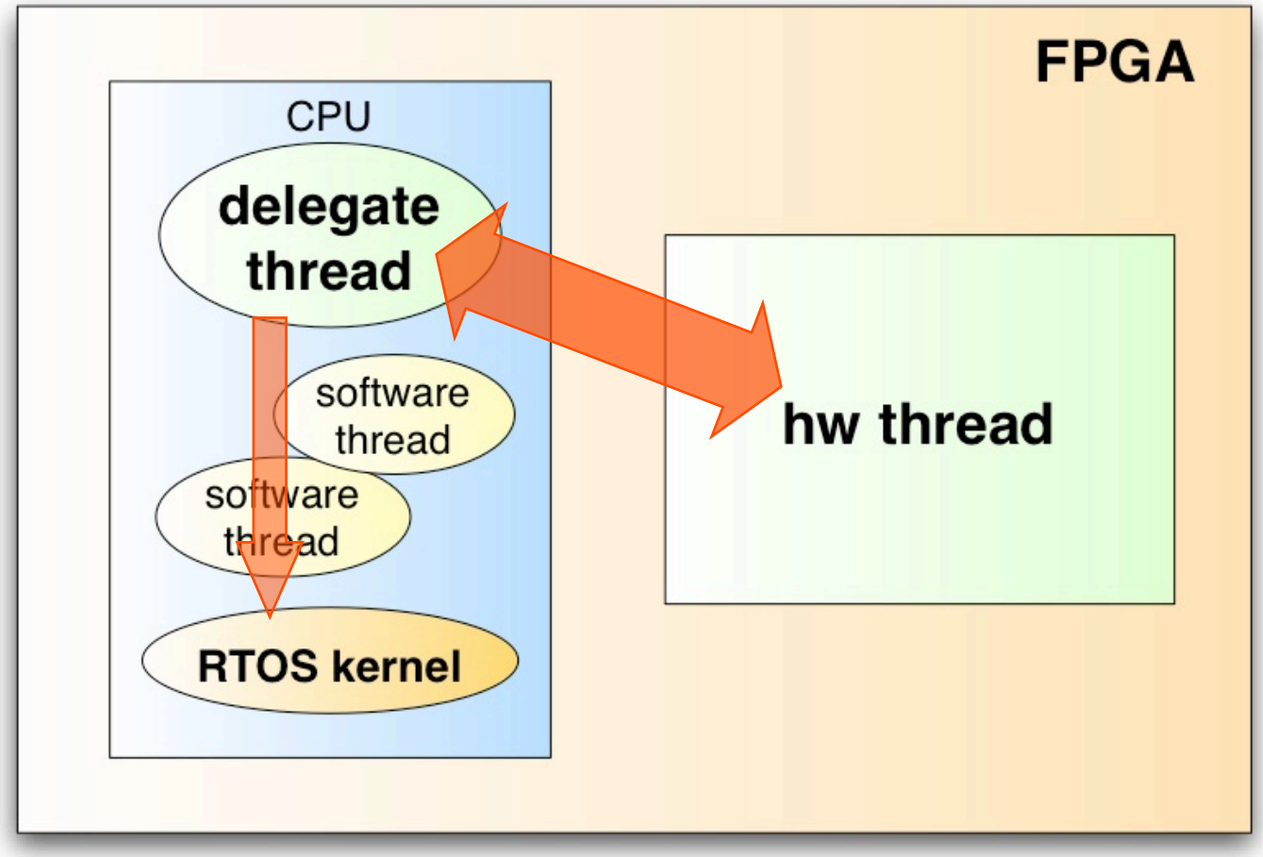


Delegate Threads

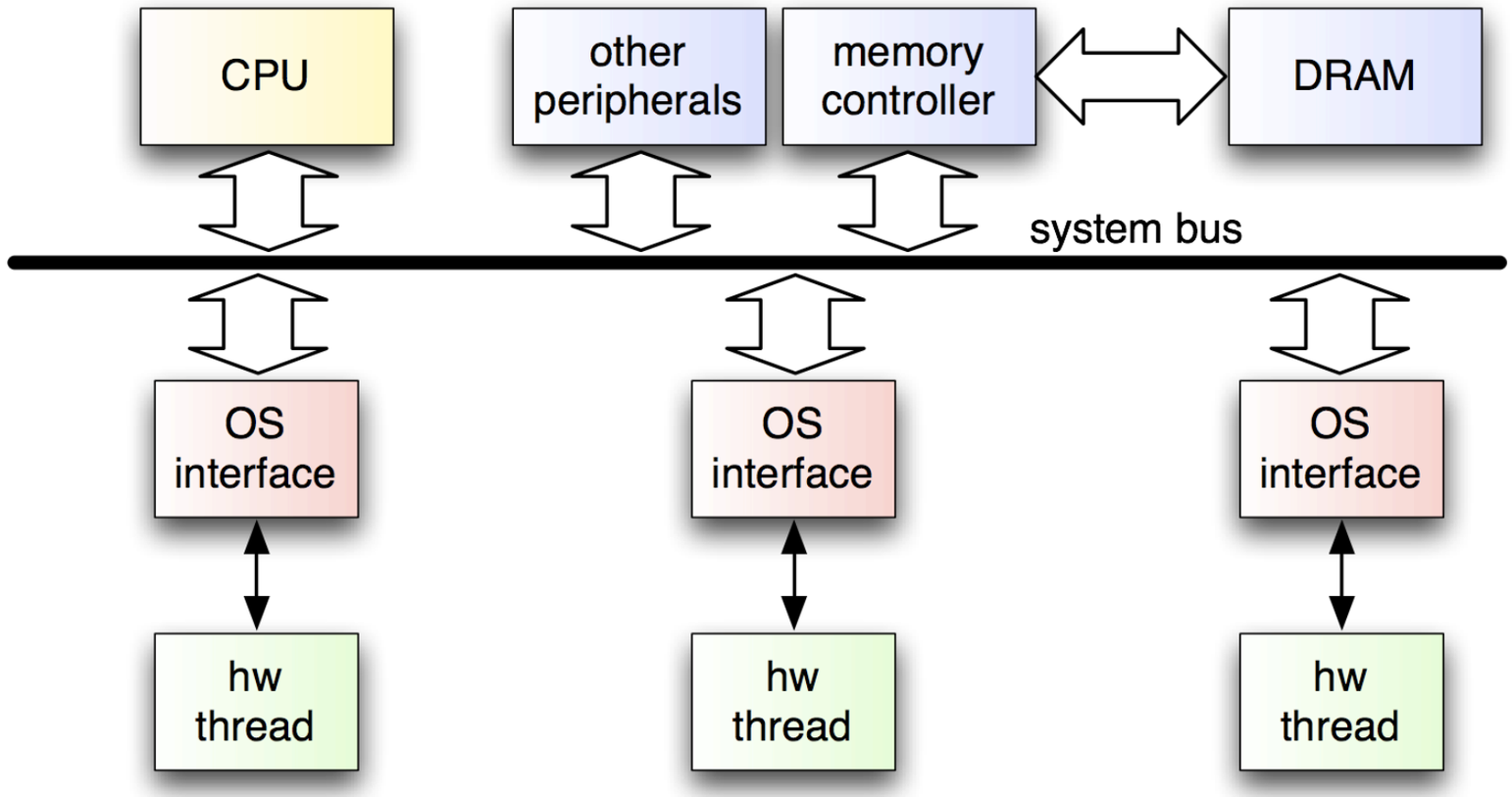
- basic mechanism
  - a delegate thread in software is associated with every hardware thread
  - the delegate thread calls the OS kernel on behalf of the hardware thread
  - all kernel responses are relayed back to the hardware thread

- advantages
  - no modification of the kernel required
  - extremely flexible
  - transparent to kernel and other threads

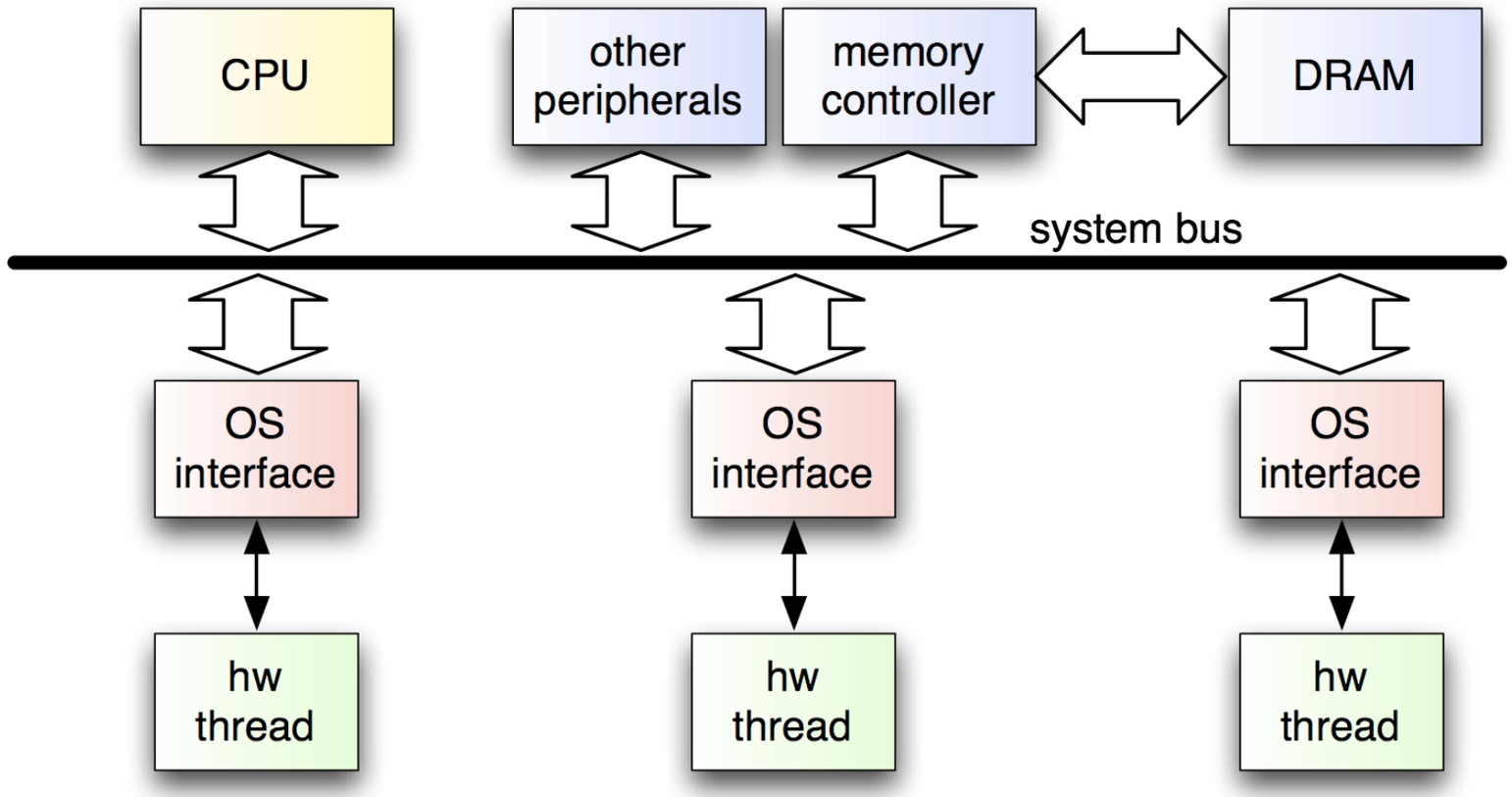
- drawbacks
  - increased overhead due to interrupt processing and context switch



Hardware Support for Message Passing

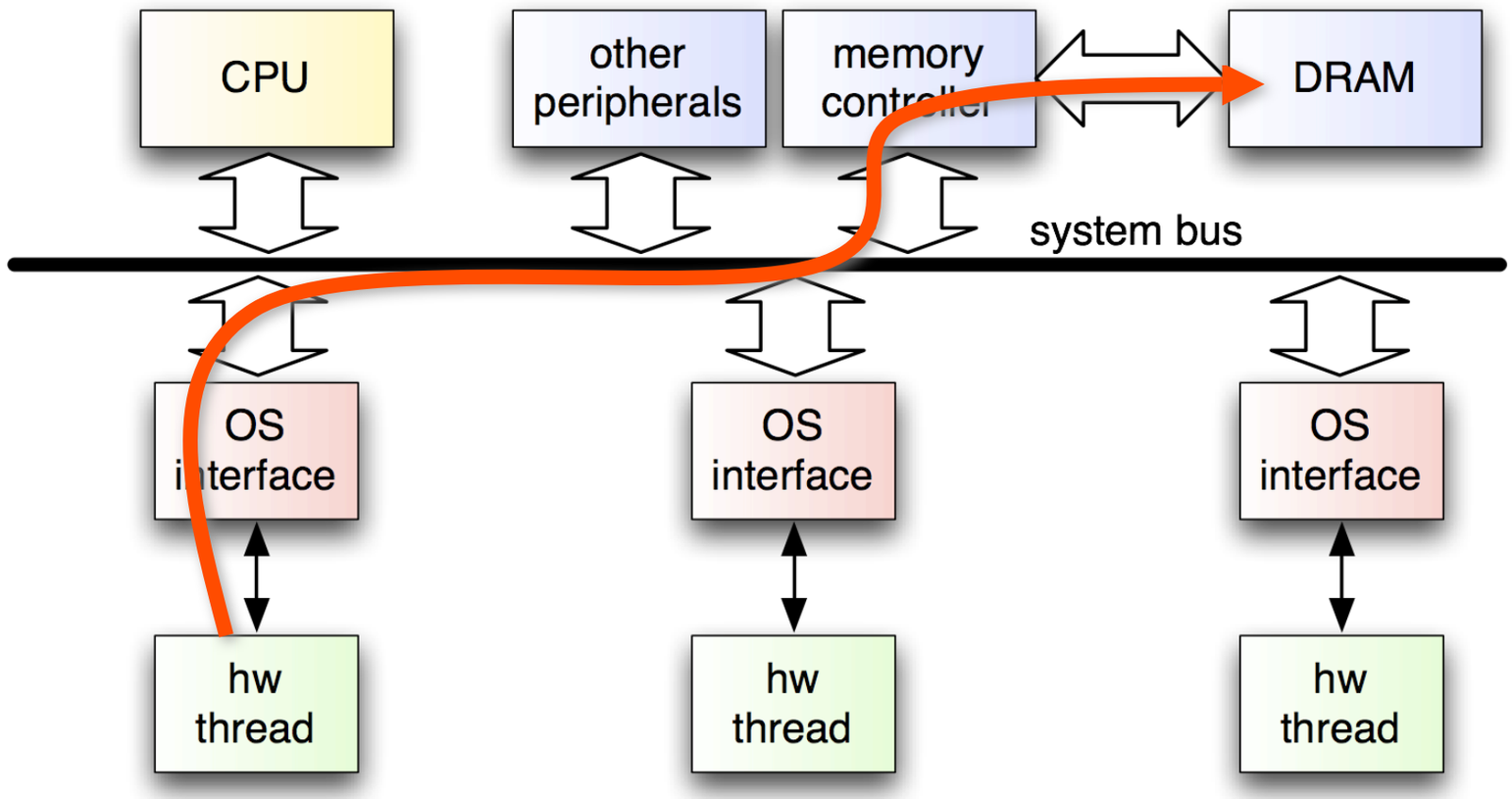


Hardware Support for Message Passing



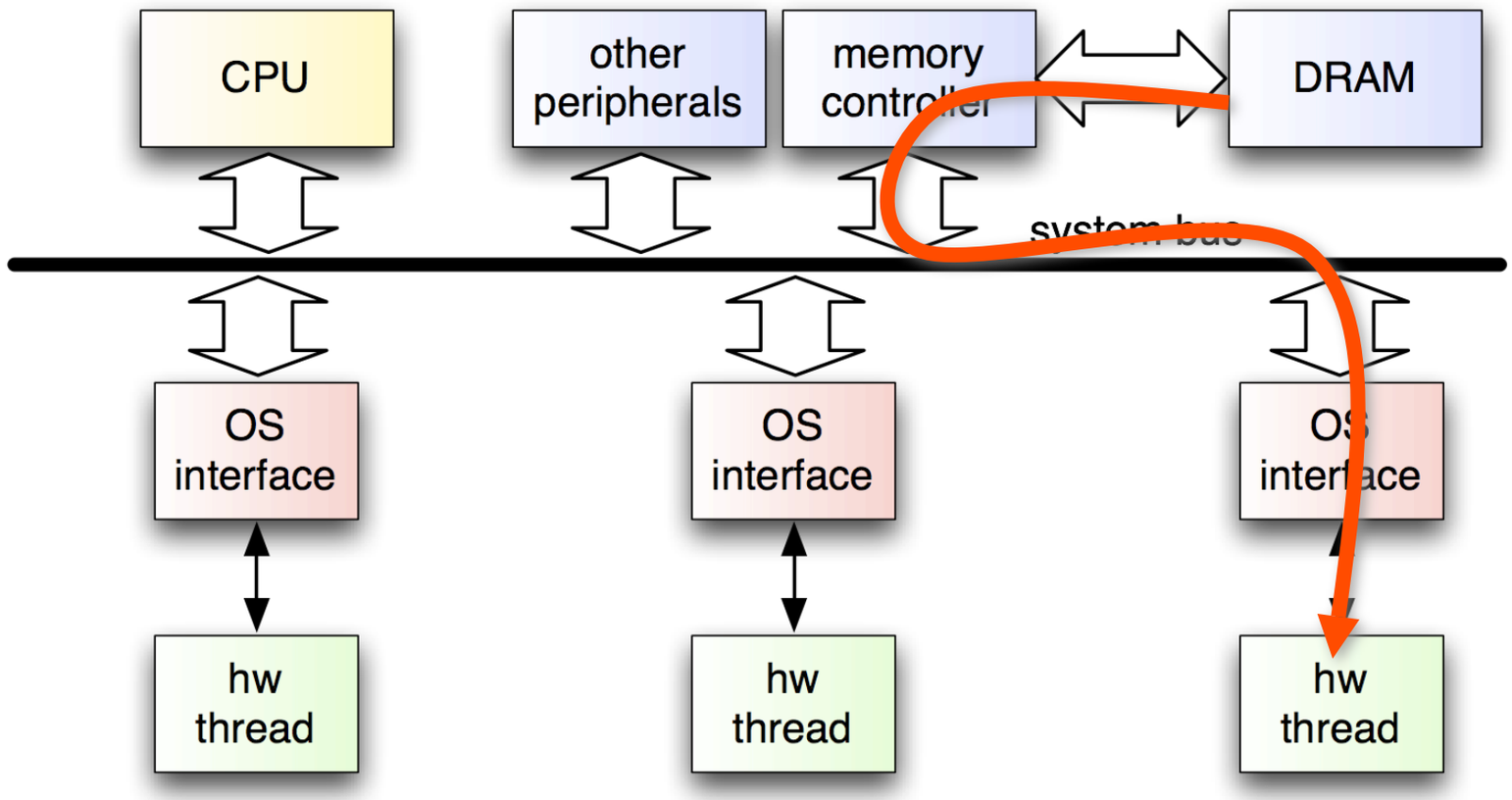
- direct HW thread-to-thread communication can be inefficient

Hardware Support for Message Passing



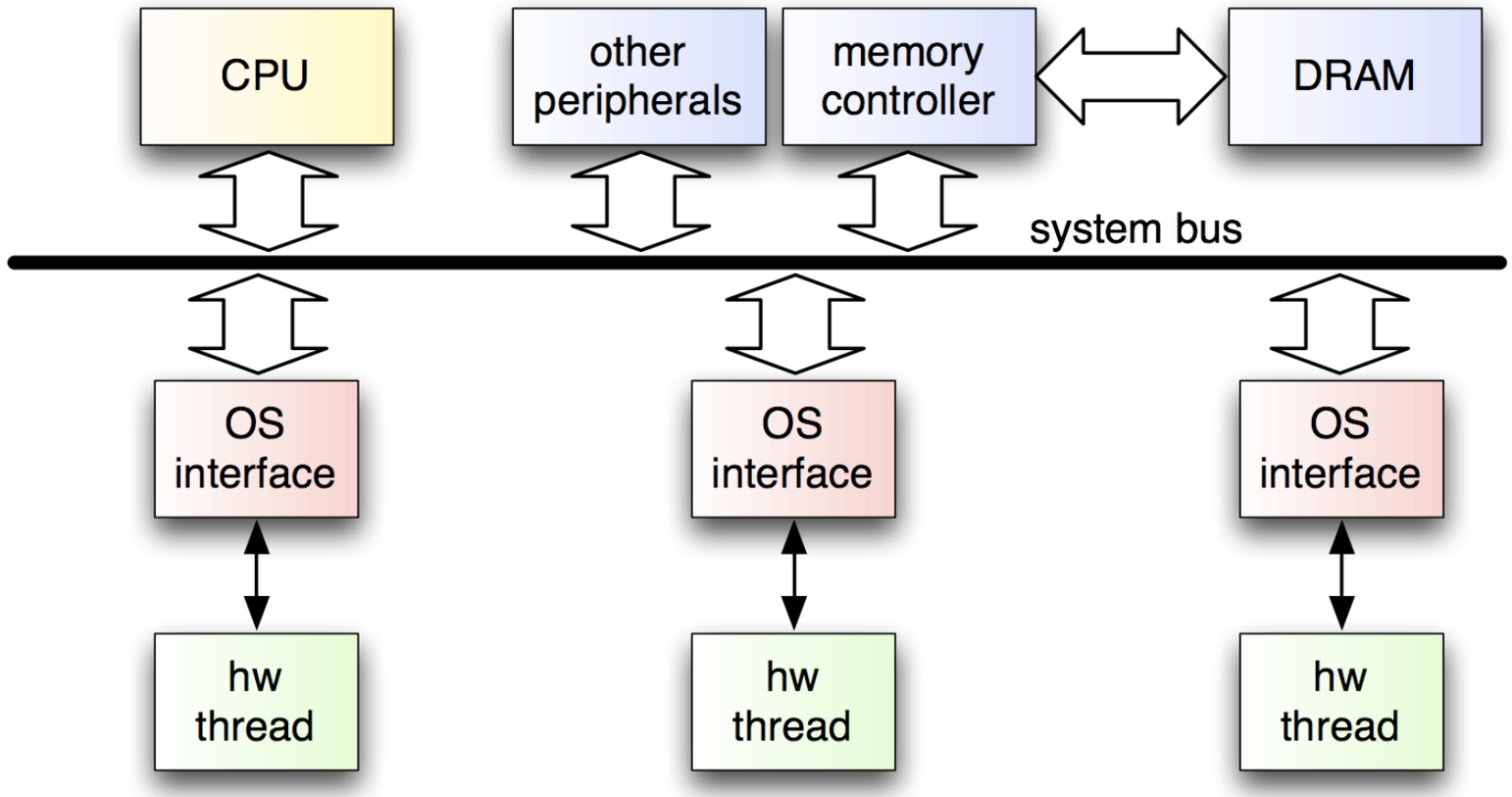
- direct HW thread-to-thread communication can be inefficient
  - shared memory: external memory access + bus arbitration

Hardware Support for Message Passing



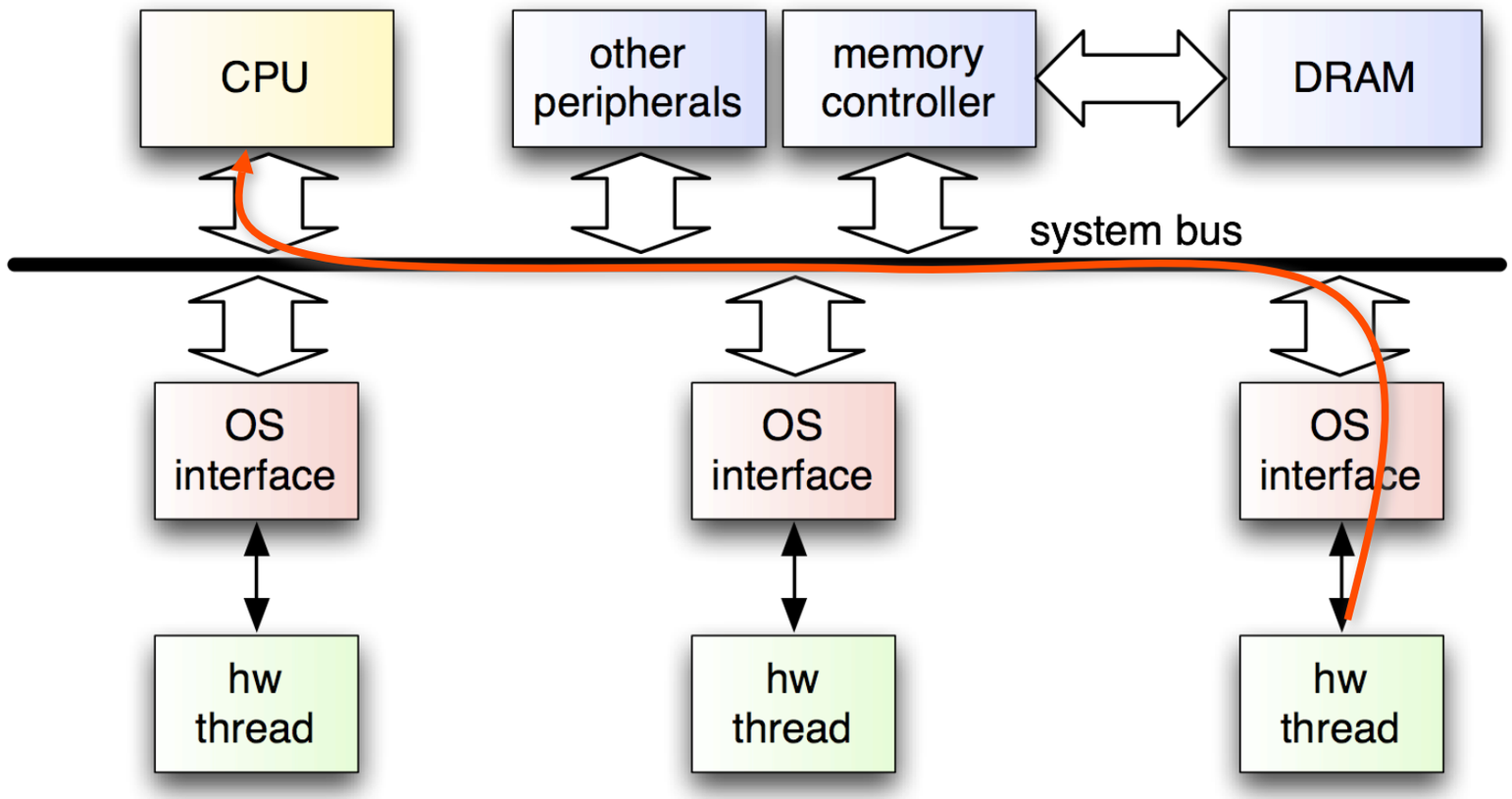
- direct HW thread-to-thread communication can be inefficient
  - shared memory: external memory access + bus arbitration

Hardware Support for Message Passing



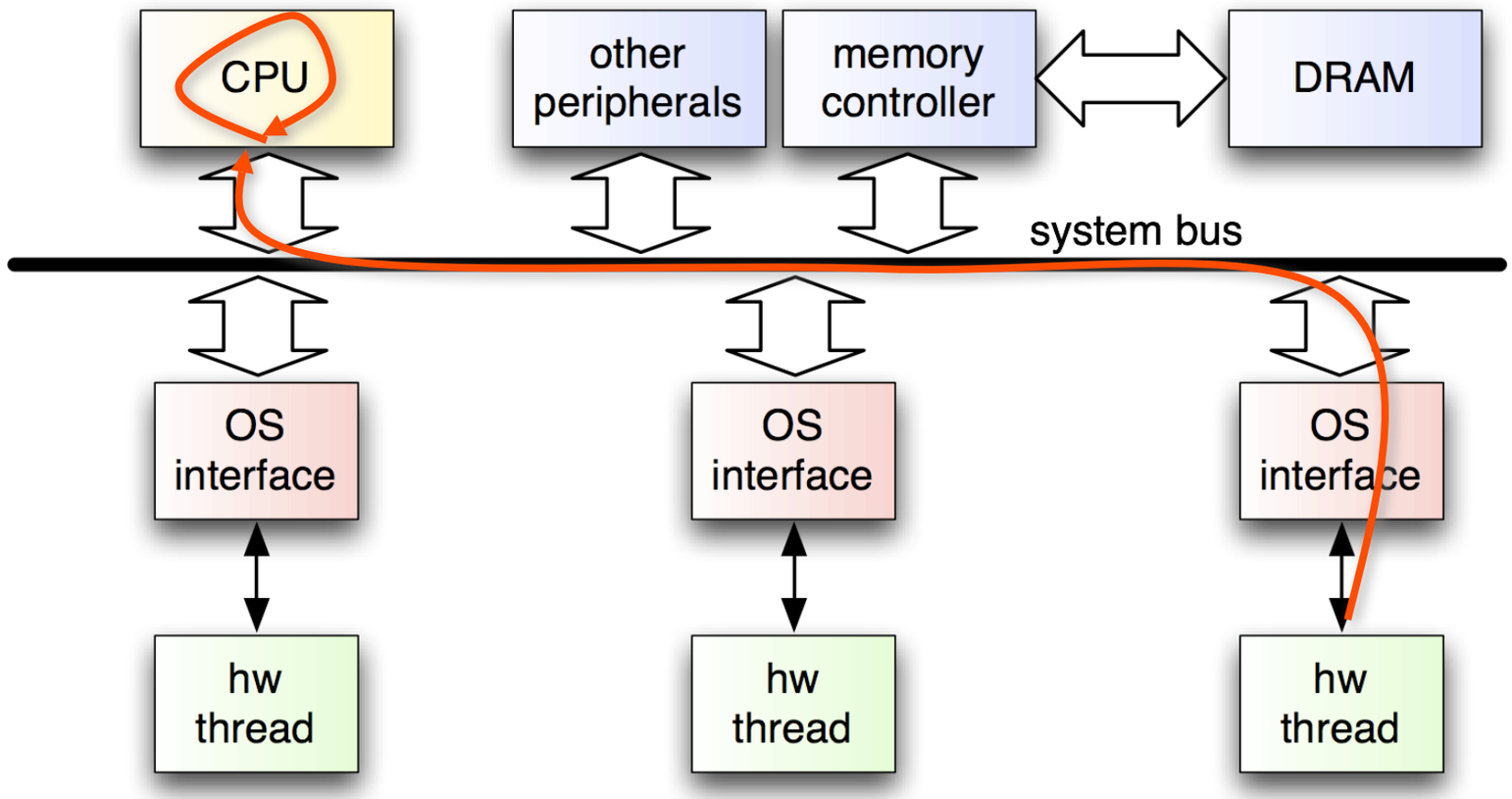
- direct HW thread-to-thread communication can be inefficient
  - shared memory: external memory access + bus arbitration

Hardware Support for Message Passing



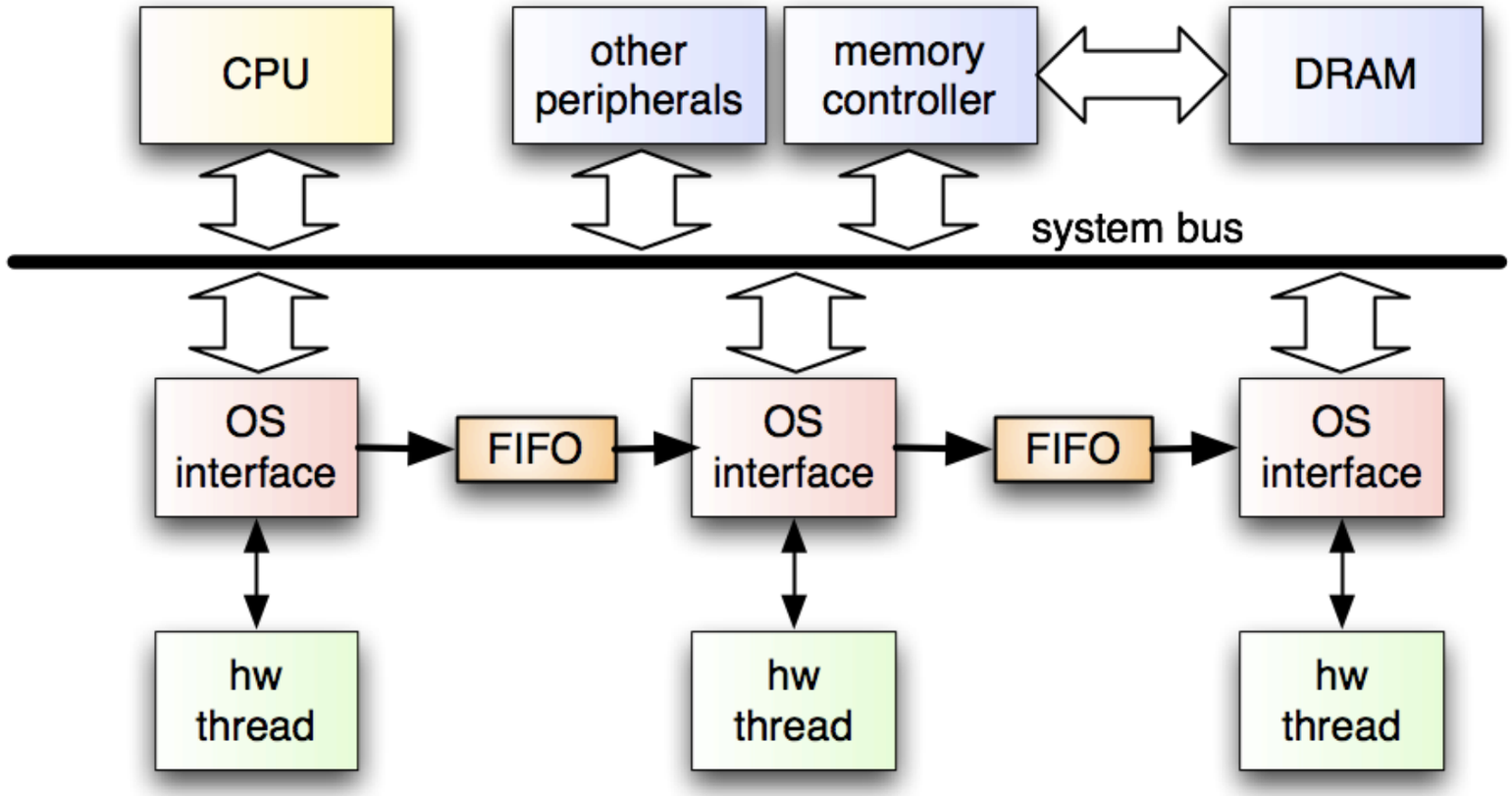
- direct HW thread-to-thread communication can be inefficient
  - shared memory: external memory access + bus arbitration
  - message passing: OS overhead (interrupt processing in CPU)

Hardware Support for Message Passing



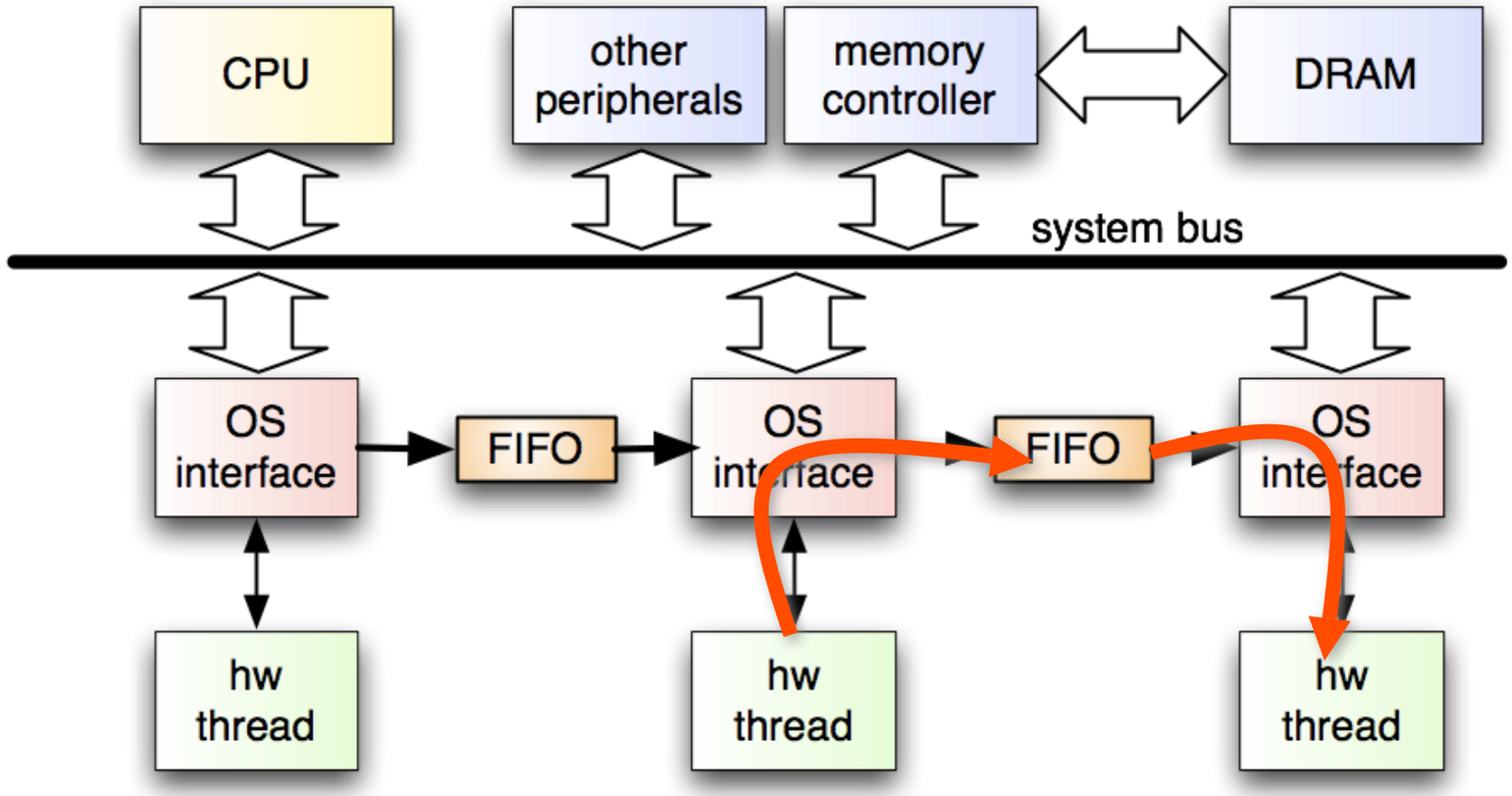
- direct HW thread-to-thread communication can be inefficient
  - shared memory: external memory access + bus arbitration
  - message passing: OS overhead (interrupt processing in CPU)

Hardware Support for Message Passing



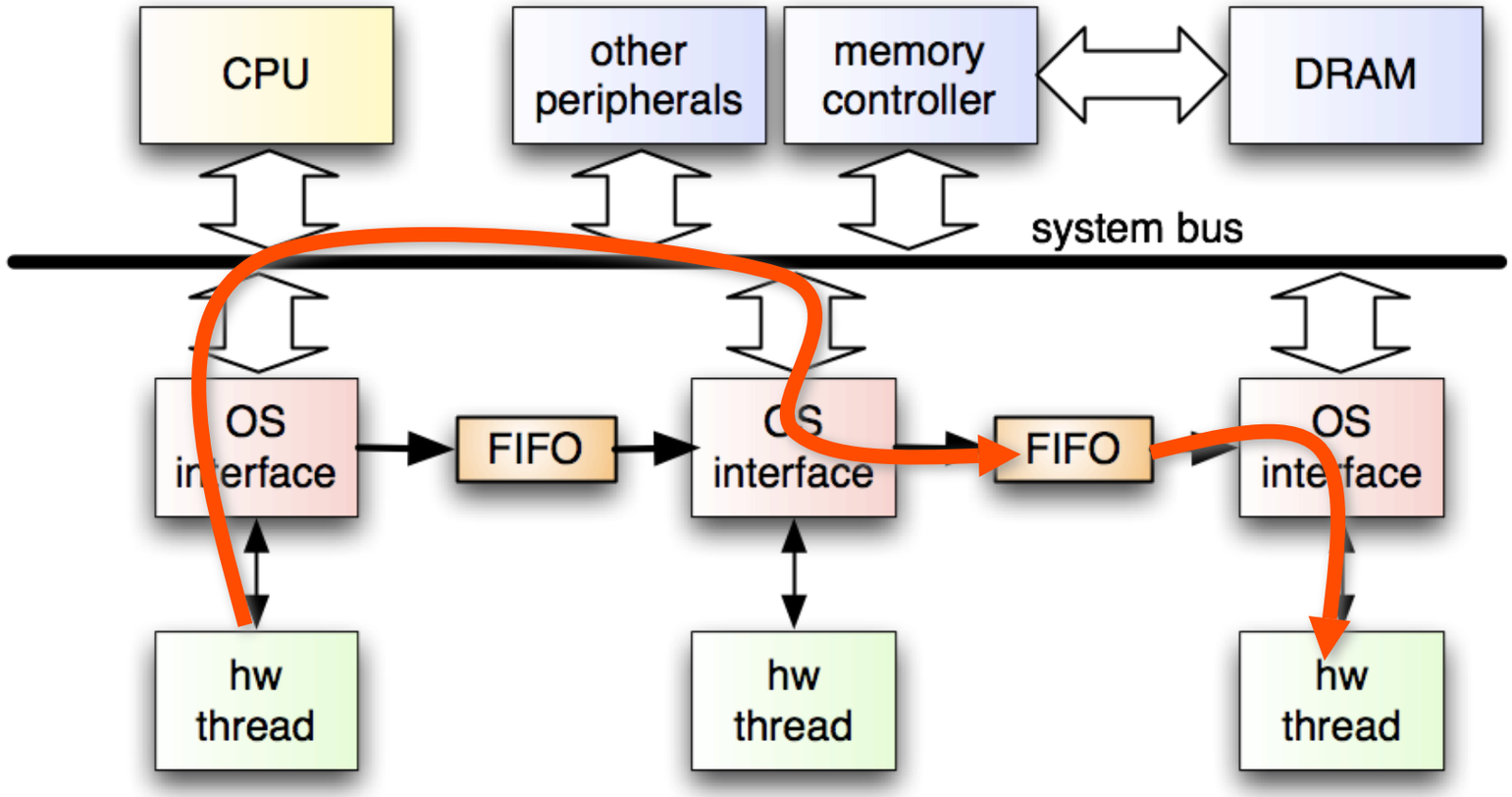
- direct HW thread-to-thread communication can be inefficient
  - shared memory: external memory access + bus arbitration
  - message passing: OS overhead (interrupt processing in CPU)
- direct connection of HW threads via buffered point-to-point links
- transparently supported by existing message queue API
- message routing dependent on current thread location (planned)

Hardware Support for Message Passing



- direct HW thread-to-thread communication can be inefficient
  - shared memory: external memory access + bus arbitration
  - message passing: OS overhead (interrupt processing in CPU)
- direct connection of HW threads via buffered point-to-point links
- transparently supported by existing message queue API
- message routing dependent on current thread location (planned)

Hardware Support for Message Passing

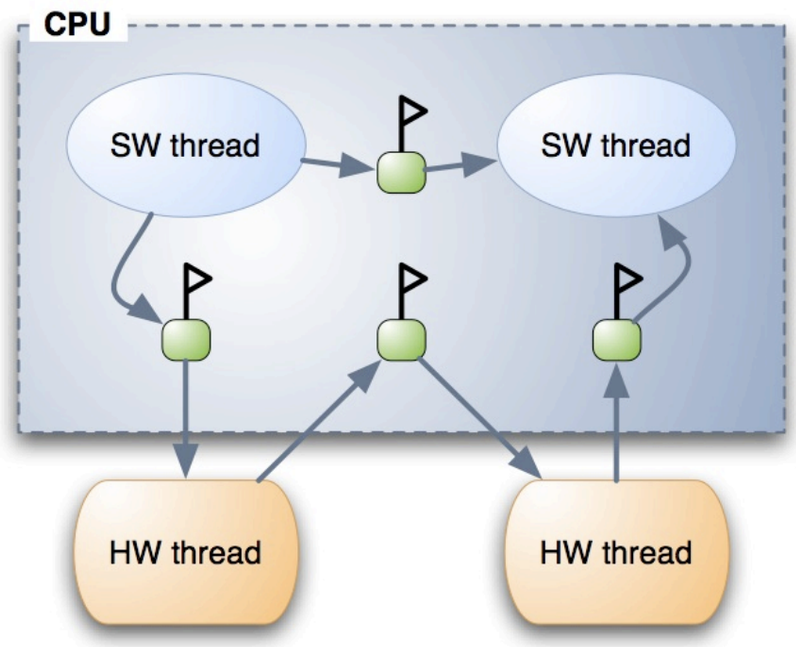


- direct HW thread-to-thread communication can be inefficient
  - shared memory: external memory access + bus arbitration
  - message passing: OS overhead (interrupt processing in CPU)
- direct connection of HW threads via buffered point-to-point links
- transparently supported by existing message queue API
- message routing dependent on current thread location (planned)

Outline

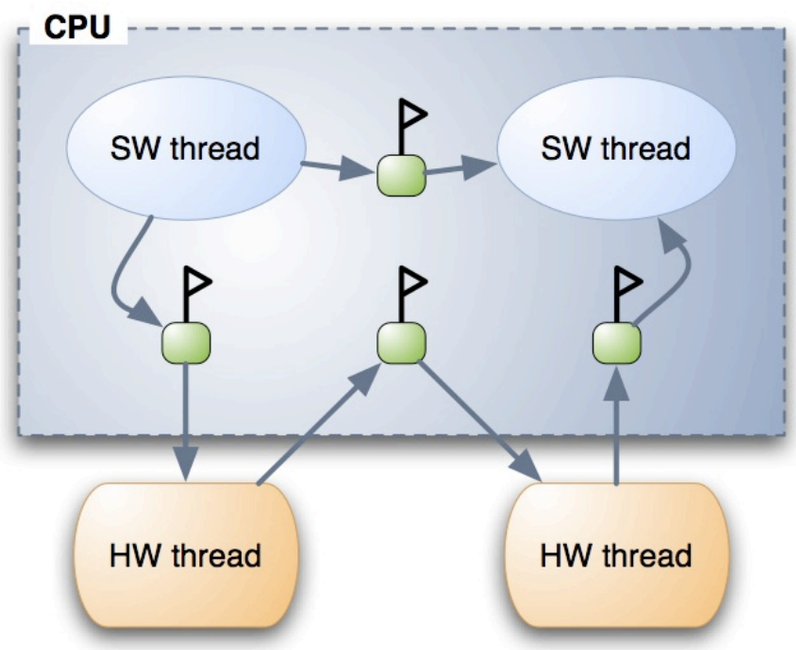
- motivation
- high-level communication and synchronization
- low-level communication and synchronization
- performance & overheads
- conclusion & outlook

Synchronization Overheads



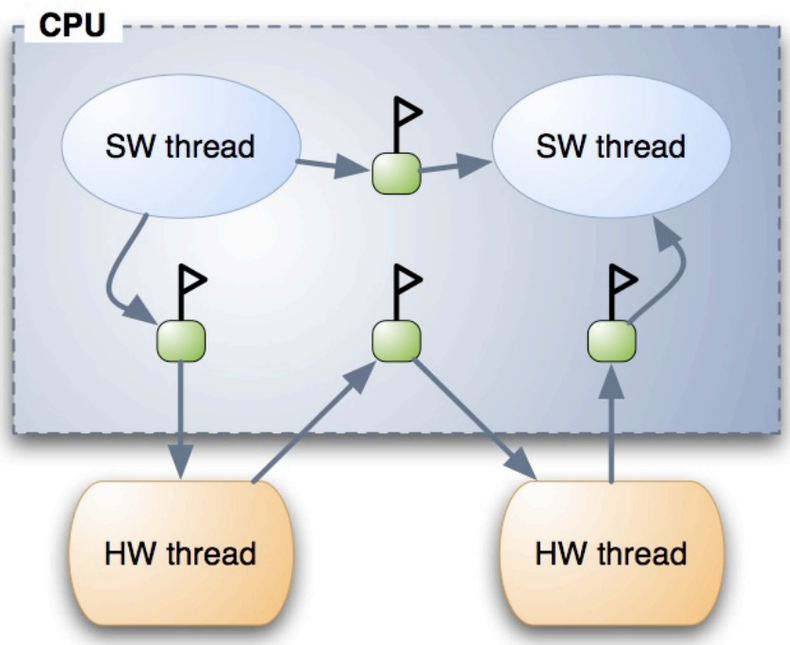
Synchronization Overheads

- synthetic hardware and software threads
  - semaphore and mutex processing time (post → wait / unlock → lock)



Synchronization Overheads

- synthetic hardware and software threads
  - semaphore and mutex processing time (post → wait / unlock → lock)
- OS calls involving hardware exhibit higher latencies
- limited impact on system performance
  - logic resources mainly used for heavy data-parallel processing
  - less synchronization-intensive control dominated code



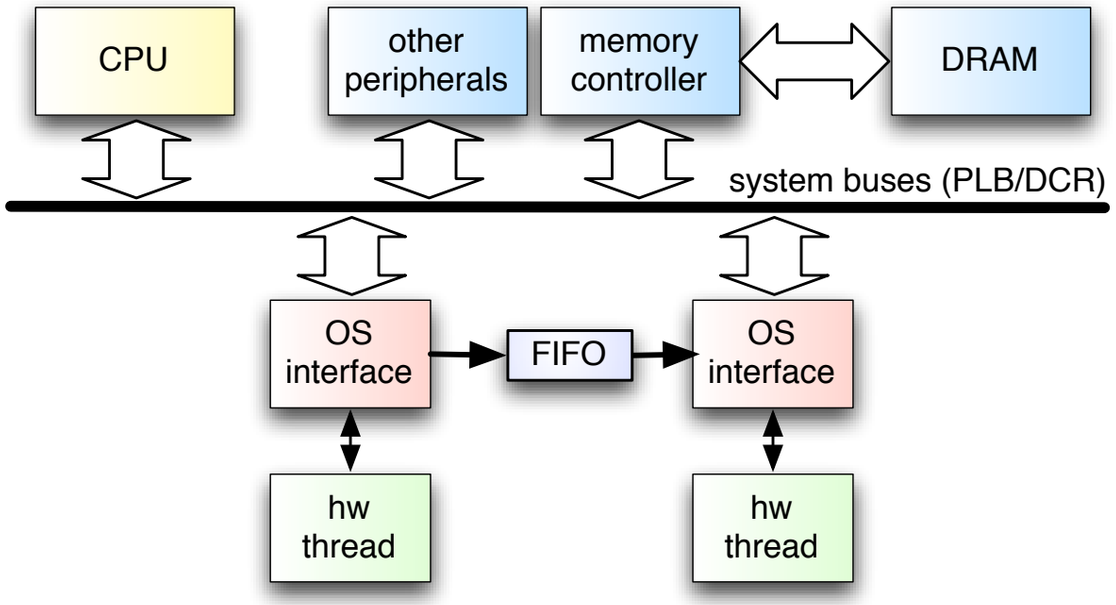
	with data cache		without data cache	
Configuration	semaphore	mutex	semaphore	mutex
SW → SW	3.39	4.53	29.05	43.99
SW → HW	4.71	6.29	30.49	47.43
HW → SW	10.74	14.49	82.90	100.23
HW → HW	11.90	14.60	83.13	101.49

All values are given in μs

Communication Performance

	Operation	with data cache μs	with data cache MB/s	without data cache μs	without data cache MB/s
	MEM→HW (burst read)	45.74	170.80	46.41	168.34
	HW→MEM (burst write)	40.54	192.71	40.55	192.66
	MEM→SW→MEM (memcopy)	132.51	58.96	625.00	12.50
hardware FIFOs	HW→HW (mbox read)	61.42	127.20	61.42	127.20
	HW→HW (mbox write)	61.45	127.14	61.45	127.14
software mailboxes	SW→HW (mbox read)	58500	0.13	374000	0.02
	HW→SW (mbox write)	58510	0.13	374000	0.02

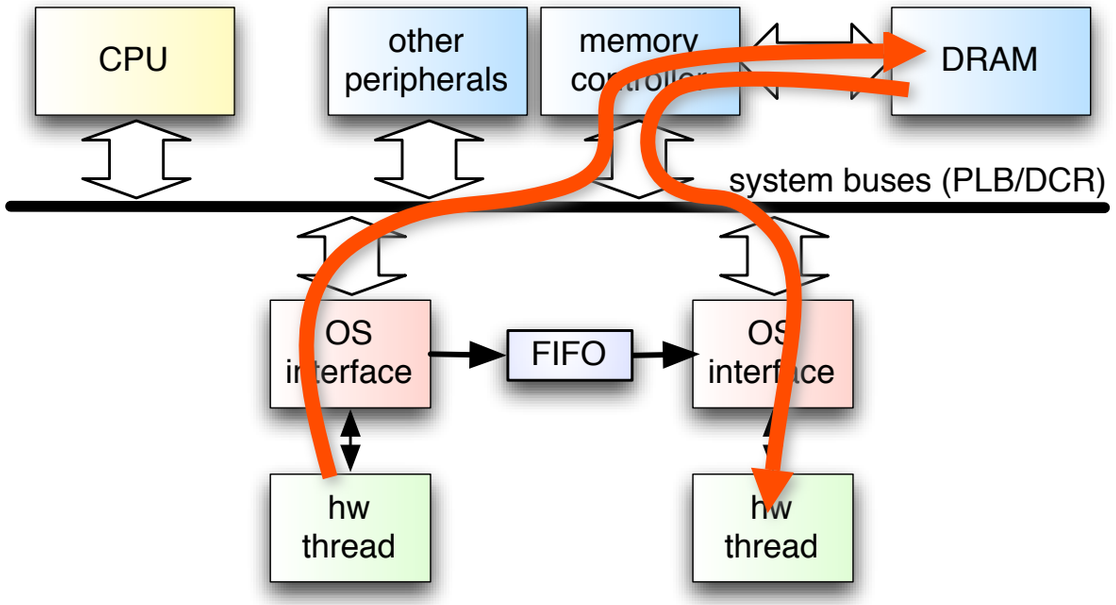
All operations were run for 8 kBytes of data



Communication Performance

	Operation	with data cache μs	with data cache MB/s	without data cache μs	without data cache MB/s
	MEM→HW (burst read)	45.74	170.80	46.41	168.34
	HW→MEM (burst write)	40.54	192.71	40.55	192.66
	MEM→SW→MEM (memcopy)	132.51	58.96	625.00	12.50
hardware FIFOs	HW→HW (mbox read)	61.42	127.20	61.42	127.20
	HW→HW (mbox write)	61.45	127.14	61.45	127.14
software mailboxes	SW→HW (mbox read)	58500	0.13	374000	0.02
	HW→SW (mbox write)	58510	0.13	374000	0.02

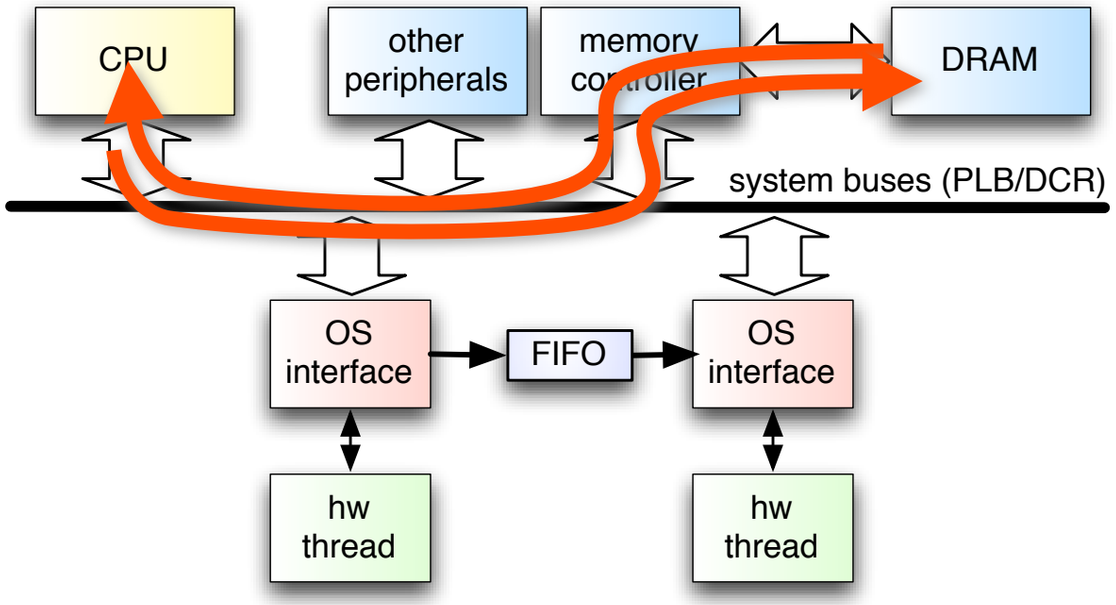
All operations were run for 8 kBytes of data



Communication Performance

	Operation	with data cache μs	with data cache MB/s	without data cache μs	without data cache MB/s
	MEM→HW (burst read)	45.74	170.80	46.41	168.34
	HW→MEM (burst write)	40.54	192.71	40.55	192.66
	MEM→SW→MEM (memcopy)	132.51	58.96	625.00	12.50
hardware FIFOs	HW→HW (mbox read)	61.42	127.20	61.42	127.20
	HW→HW (mbox write)	61.45	127.14	61.45	127.14
software mailboxes	SW→HW (mbox read)	58500	0.13	374000	0.02
	HW→SW (mbox write)	58510	0.13	374000	0.02

All operations were run for 8 kBytes of data



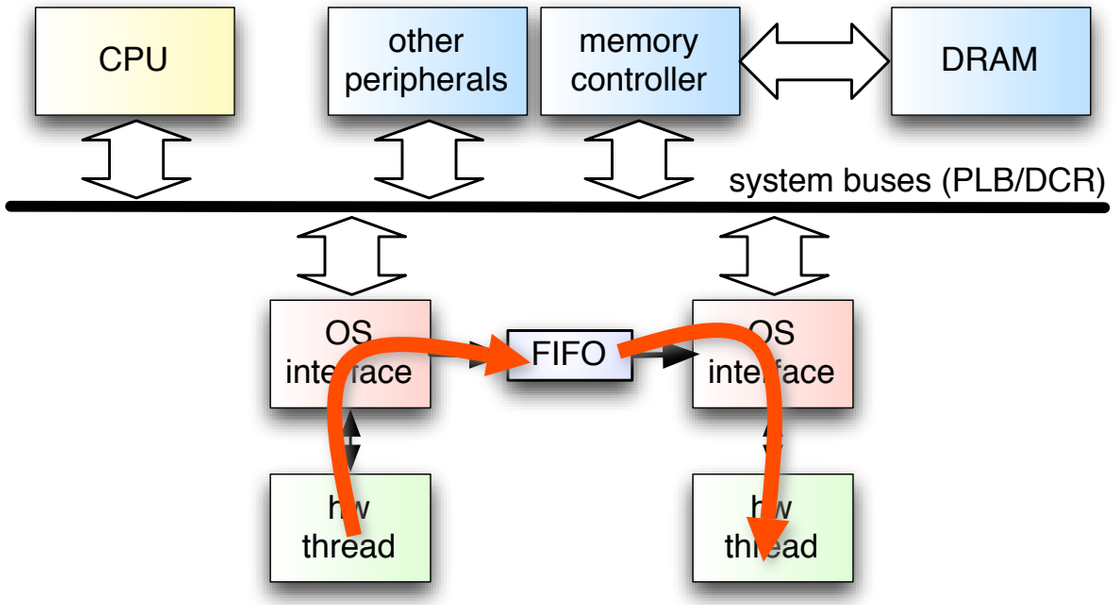
Communication Performance

Operation	with data cache μs	with data cache MB/s	without data cache μs	without data cache MB/s
MEM→HW (burst read)	45.74	170.80	46.41	168.34
HW→MEM (burst write)	40.54	192.71	40.55	192.66
MEM→SW→MEM (memcopy)	132.51	58.96	625.00	12.50
HW→HW (mbox read)	61.42	127.20	61.42	127.20
HW→HW (mbox write)	61.45	127.14	61.45	127.14
SW→HW (mbox read)	58500	0.13	374000	0.02
HW→SW (mbox write)	58510	0.13	374000	0.02

All operations were run for 8 kBytes of data

hardware FIFOs

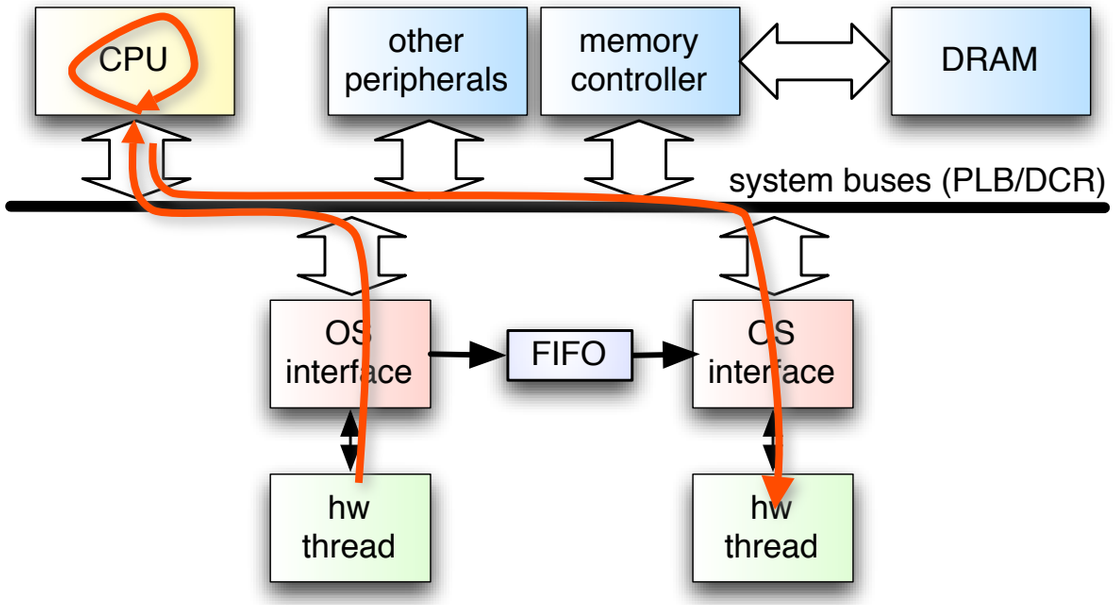
software mailboxes



Communication Performance

	Operation	with data cache μs	with data cache MB/s	without data cache μs	without data cache MB/s
	MEM→HW (burst read)	45.74	170.80	46.41	168.34
	HW→MEM (burst write)	40.54	192.71	40.55	192.66
	MEM→SW→MEM (memcopy)	132.51	58.96	625.00	12.50
hardware FIFOs	HW→HW (mbox read)	61.42	127.20	61.42	127.20
	HW→HW (mbox write)	61.45	127.14	61.45	127.14
software mailboxes	SW→HW (mbox read)	58500	0.13	374000	0.02
	HW→SW (mbox write)	58510	0.13	374000	0.02

All operations were run for 8 kBytes of data

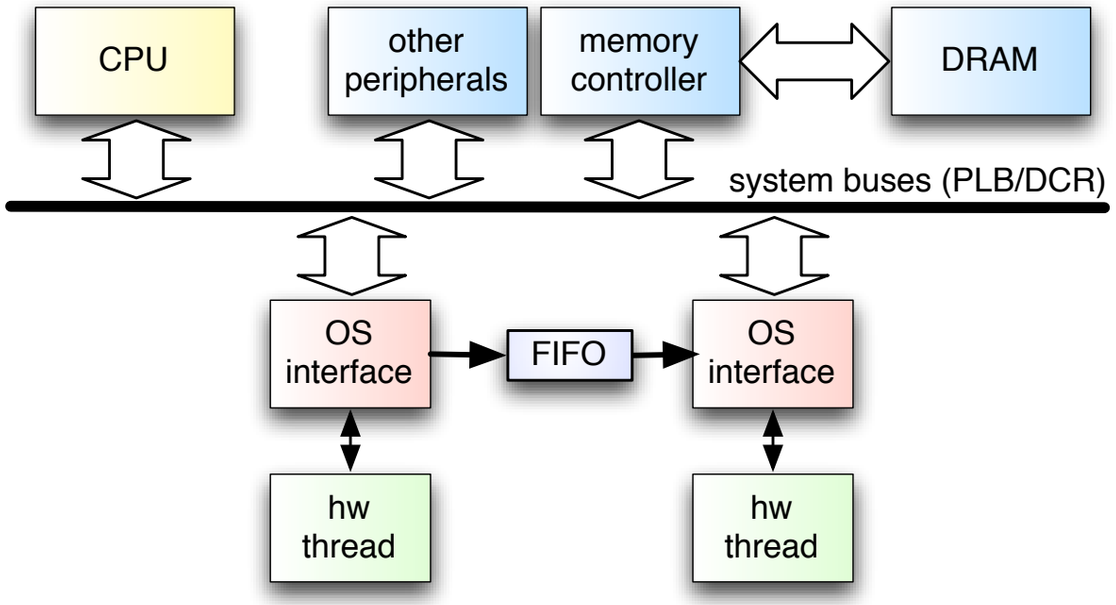


Communication Performance

	Operation	with data cache μs	with data cache MB/s	without data cache μs	without data cache MB/s
	MEM→HW (burst read)	45.74	170.80	46.41	168.34
	HW→MEM (burst write)	40.54	192.71	40.55	192.66
	MEM→SW→MEM (memcopy)	132.51	58.96	625.00	12.50
hardware FIFOs	HW→HW (mbox read)	61.42	127.20	61.42	127.20
	HW→HW (mbox write)	61.45	127.14	61.45	127.14
software mailboxes	SW→HW (mbox read)	58500	0.13	374000	0.02
	HW→SW (mbox write)	58510	0.13	374000	0.02

All operations were run for 8 kBytes of data

- comparison between memory access and FIFO transfers
  - FIFOs are faster for HW thread to HW thread communications (+40%)
  - no additional load on memory system or CPU
  - improves thread-parallelism



Conclusion & Outlook

Conclusion & Outlook

- common set of high-level communication and synchronization objects for hard- and software unifies programming model
- existing operating systems can be extended with mechanisms for low-level communication and synchronization between hardware threads and kernel
- acceptable performance in benchmarks and larger case studies

Conclusion & Outlook

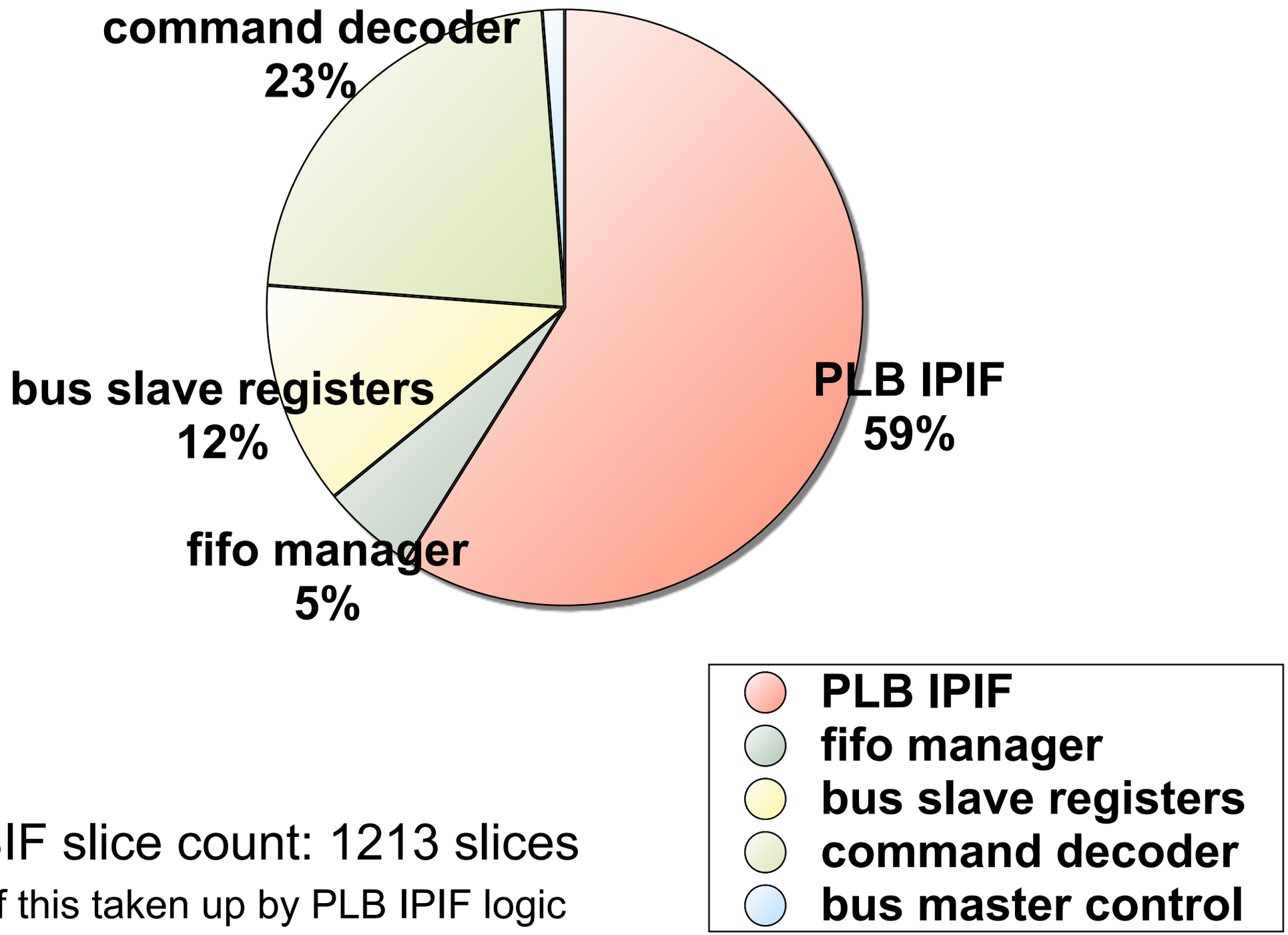
- common set of high-level communication and synchronization objects for hard- and software unifies programming model
- existing operating systems can be extended with mechanisms for low-level communication and synchronization between hardware threads and kernel
- acceptable performance in benchmarks and larger case studies

- future work
  - extension of hardware FIFOs to allow direct access from software threads
  - fusion of shared memory and message-passing interfaces to hardware threads

Thank you

www.reconos.de

OS Overheads (Area)



- total OSIF slice count: 1213 slices
  - most of this taken up by PLB IPIF logic

Supported OS Calls

- Semaphores (counting and binary)
  - reconos_semaphore_post()
  - reconos_semaphore_wait()
- Mutexes
  - reconos_mutex_lock()
  - reconos_mutex_trylock()
  - reconos_mutex_unlock()
  - reconos_mutex_release()
- Condition Variables
  - reconos_cond_wait()
  - reconos_cond_signal()
  - reconos_cond_broadcast()
- Mailboxes
  - reconos_mbox_get()
  - reconos_mbox_tryget()
  - reconos_mbox_put()
  - reconos_mbox_tryput()
- Memory access
  - reconos_read()
  - reconos_write()
  - reconos_read_burst()
  - reconos_write_burst()

basic synchronization primitives

synchronize access to mutual exclusive operations (critical sections)

allow waiting until arbitrary conditions are satisfied

message passing primitives (blocking and not blocking)

CPU-independent access to the entire system address space (memory and peripherals)

handled in software (via delegate thread)

handled in hardware (via system bus / point-to-point links)

ReconOS Software API (POSIX)

- standard POSIX thread creation

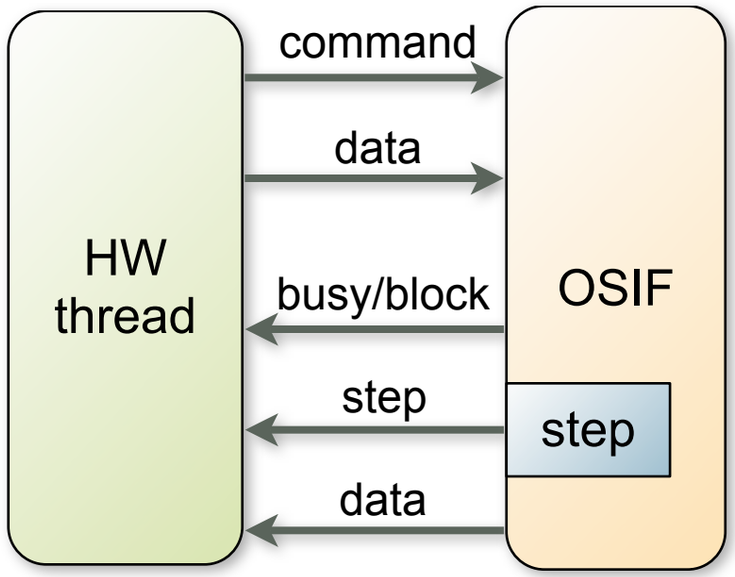
```
mqd_t my_mbox;
sem_t my_sem;

pthread_t      thread;
pthread_attr_t thread_attr;

...

pthread_attr_init(&thread_attr);

pthread_create(
   &thread,                  // thread object
   &thread_attr,             // attributes
   thread_entry,             // entry point
   ( void * ) data           // entry data
);
```

- ReconOS hardware thread creation

```
mqd_t my_mbox;
sem_t my_sem;
reconos_res_t thread_resources[2] = {
            { &my_mbox, POSIX_MQD_T },
            { &my_sem,  POSIX_SEM_T }
};

rthread        thread;
pthread_attr_t thread_swattr;
rthread_attr_t thread_hwattr;
...

pthread_attr_init(&thread_swattr);
rthread_attr_init(&thread_hwattr);
rthread_attr_setslotnum(&thread_hwattr, 0);
rthread_attr_setresources(&thread_hwattr,
                          thread_resources, 2);

rthread_create(
   &thread,                  // thread object
   &thread_swattr,           // software attributes
   &thread_hwattr,           // hardware attributes
   ( void * ) data           // entry data
);
```

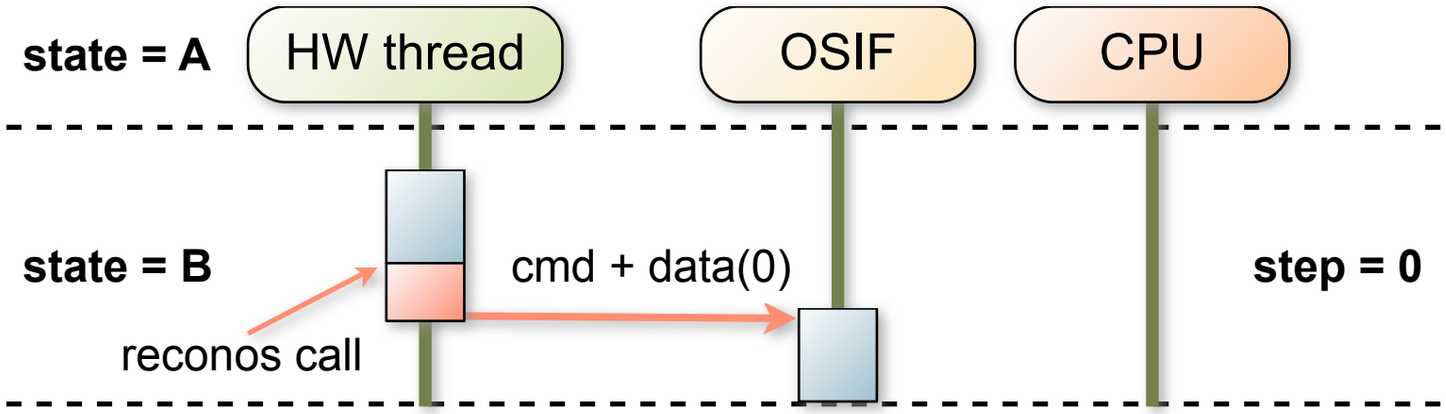
Multi-Cycle Commands

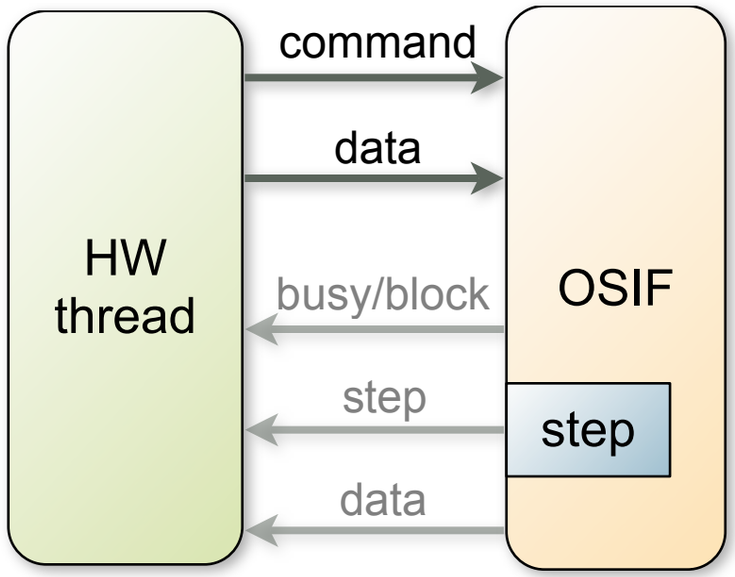
- transfer of multiple parameters and return values with a single VHDL call
- distributes execution of an FSM state across multiple clock cycles



Multi-Cycle Commands

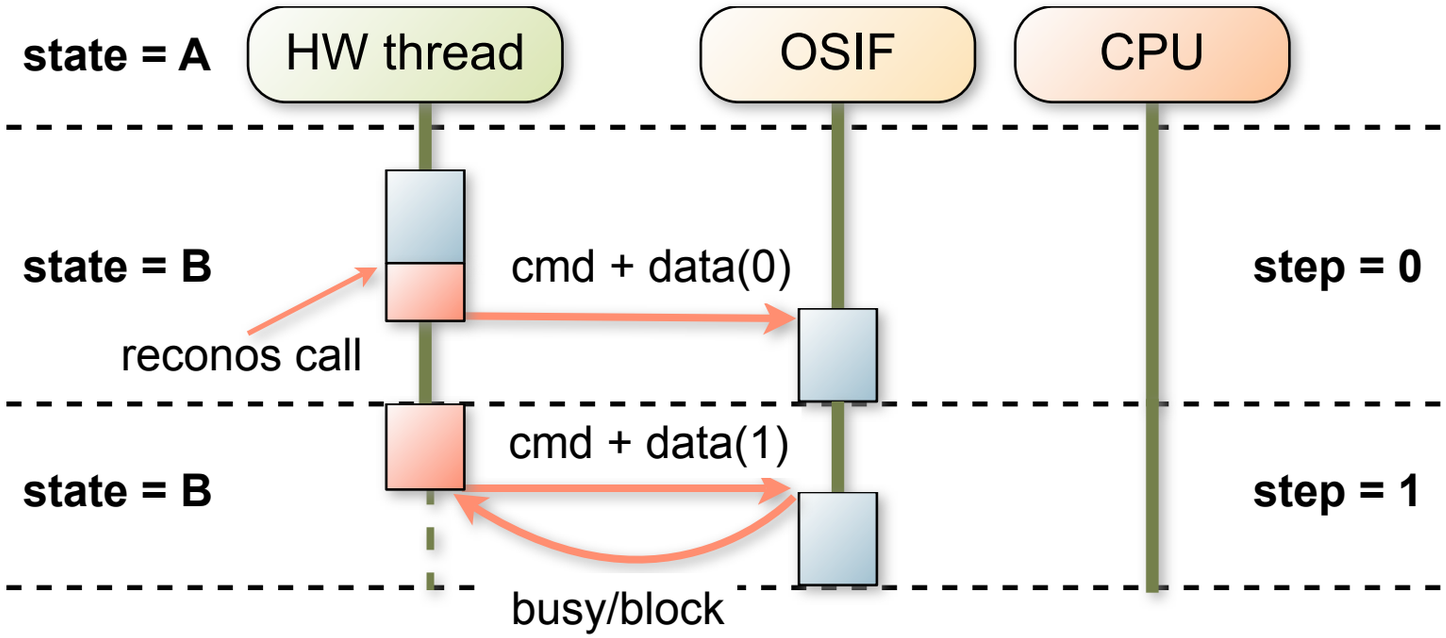
- transfer of multiple parameters and return values with a single VHDL call
- distributes execution of an FSM state across multiple clock cycles

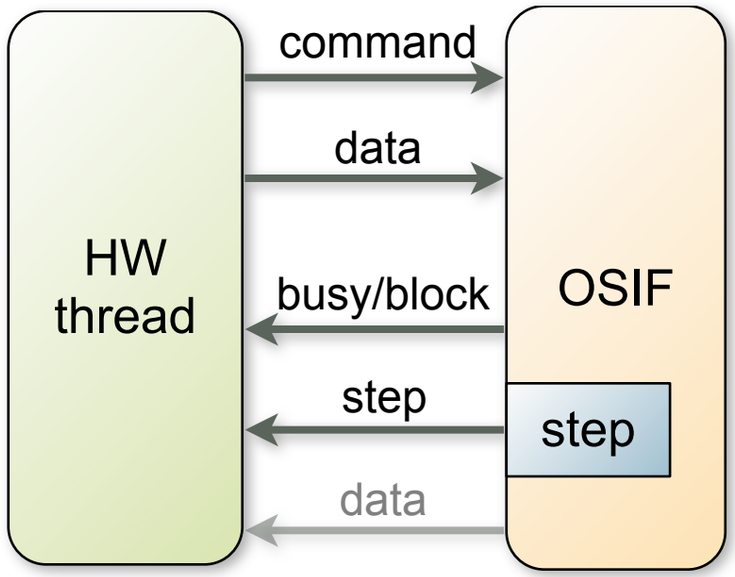




Multi-Cycle Commands

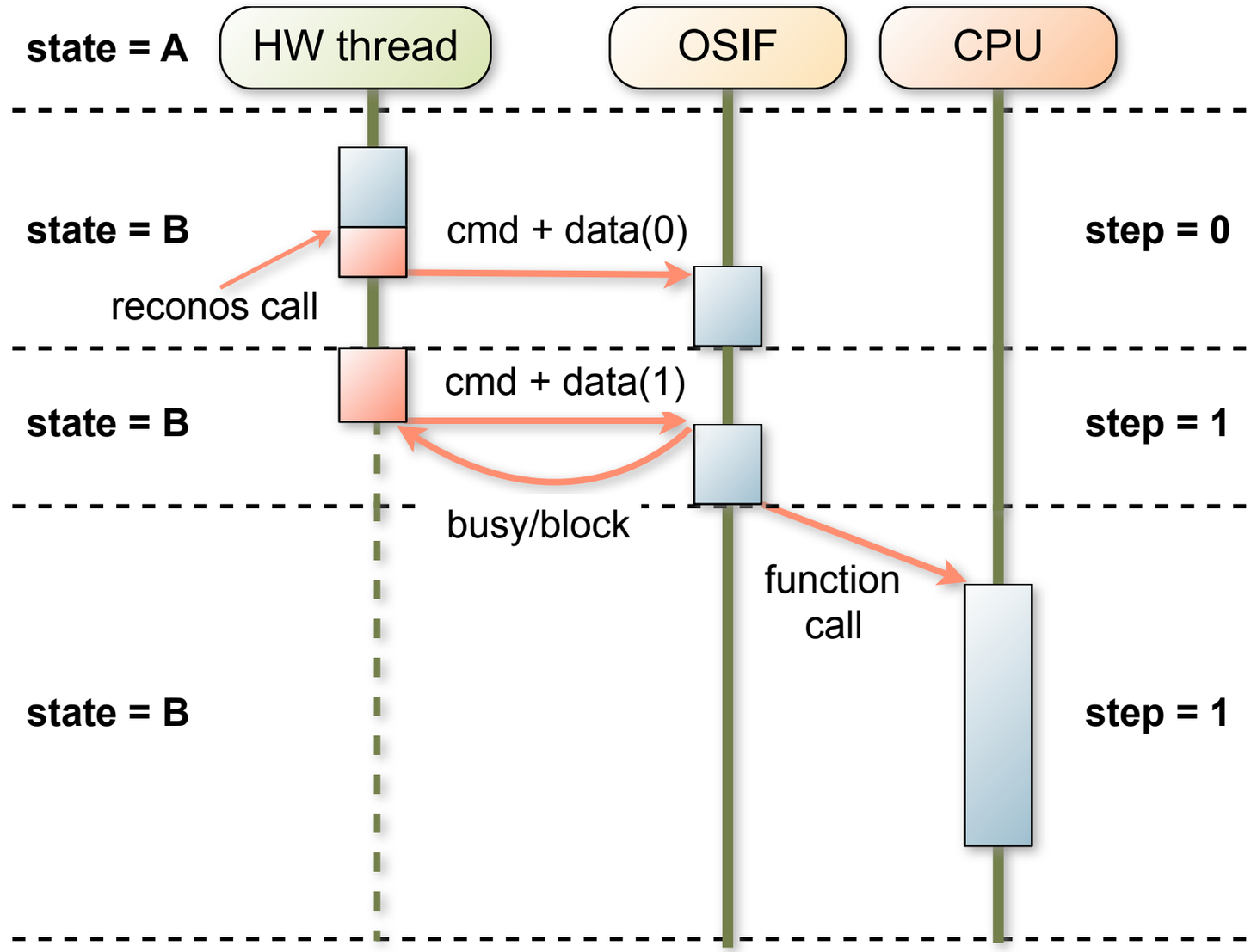
- transfer of multiple parameters and return values with a single VHDL call
- distributes execution of an FSM state across multiple clock cycles

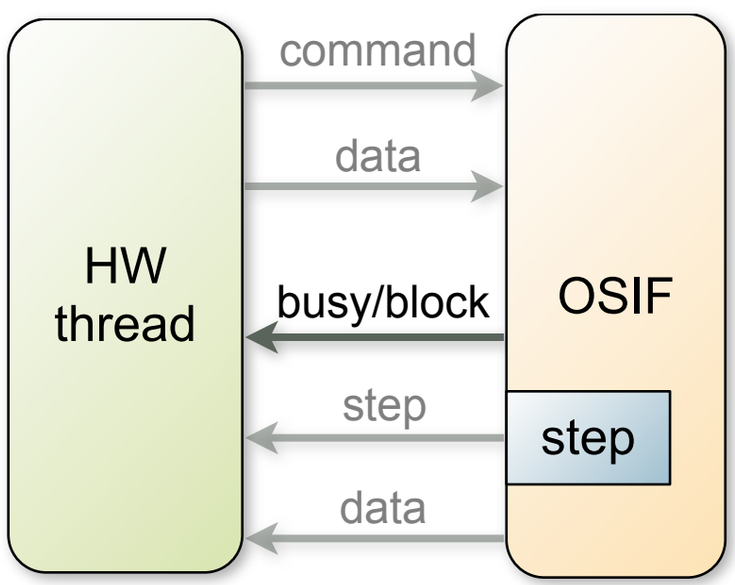




Multi-Cycle Commands

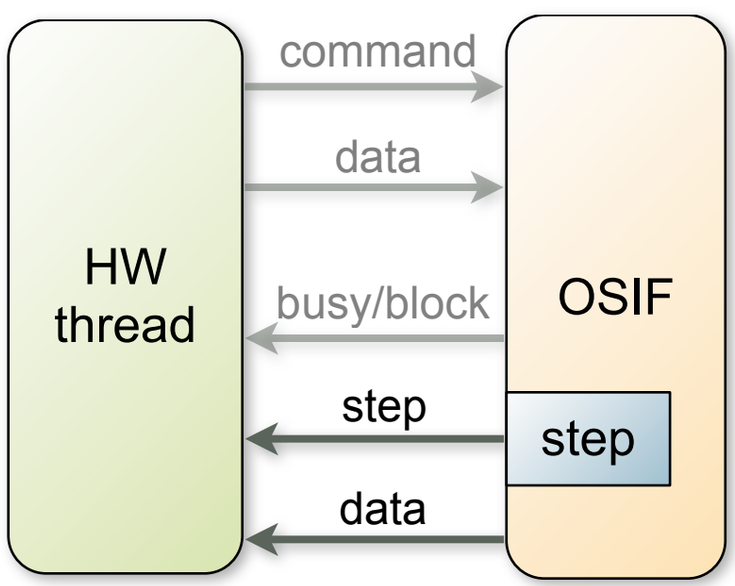
- transfer of multiple parameters and return values with a single VHDL call
- distributes execution of an FSM state across multiple clock cycles

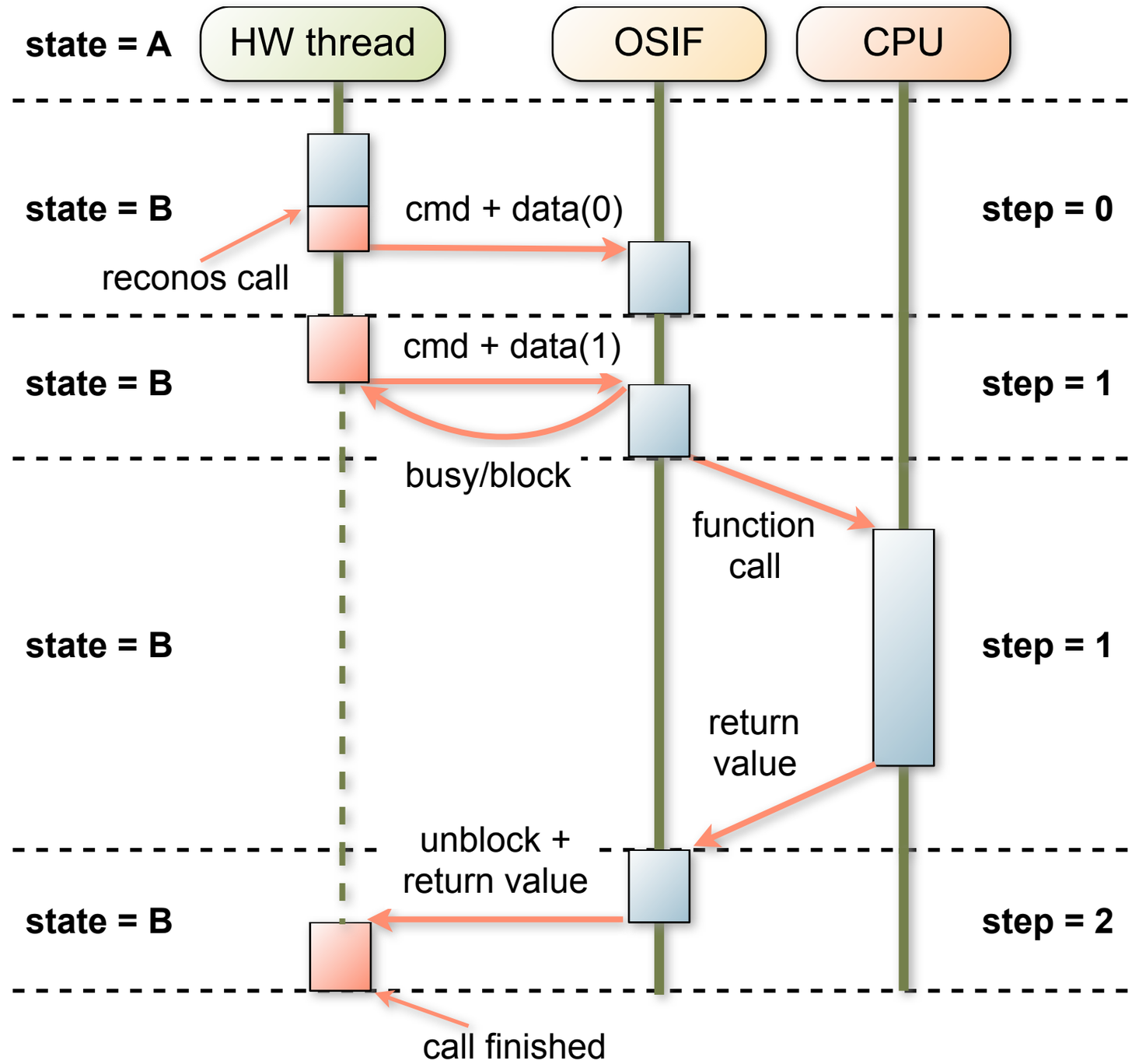




Multi-Cycle Commands

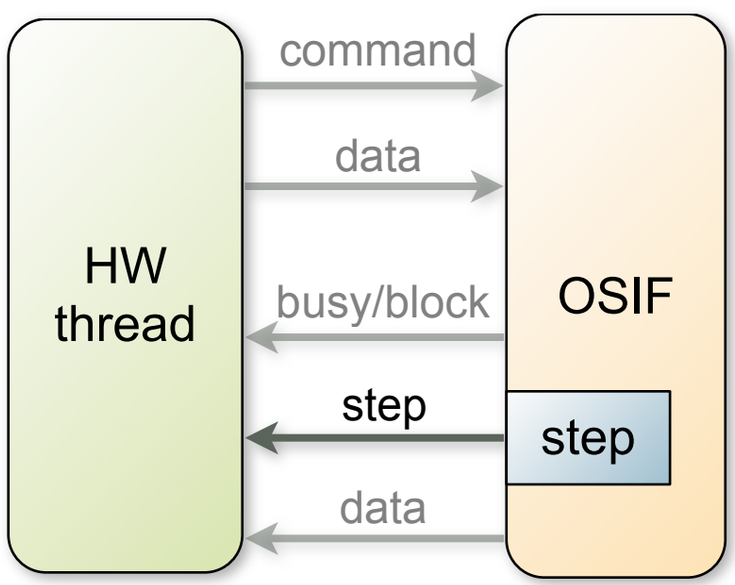
- transfer of multiple parameters and return values with a single VHDL call
- distributes execution of an FSM state across multiple clock cycles

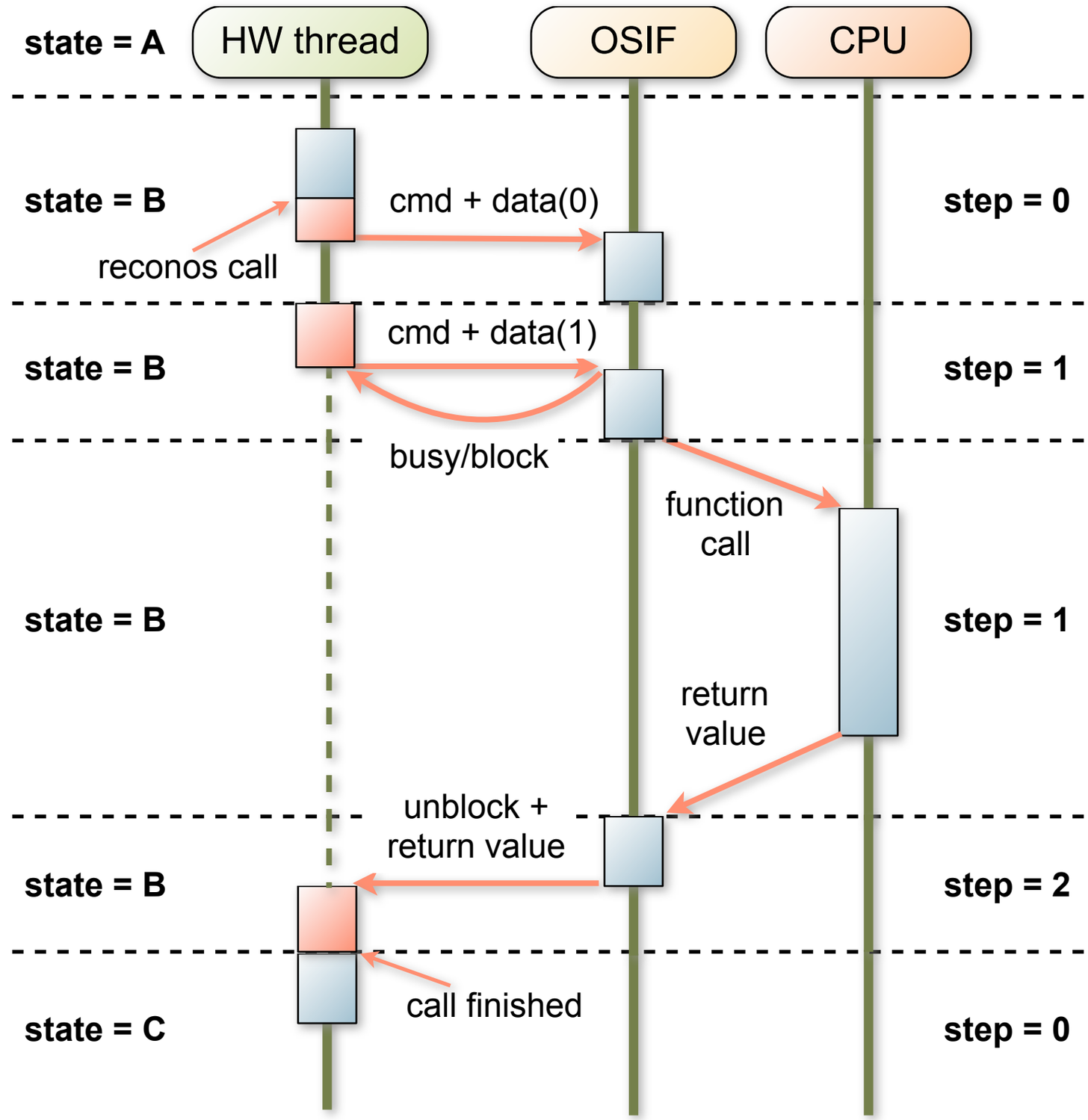




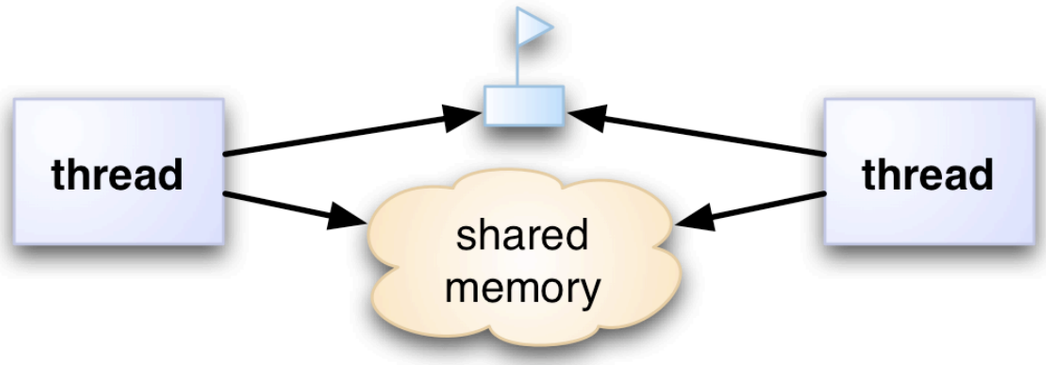
Multi-Cycle Commands

- transfer of multiple parameters and return values with a single VHDL call
- distributes execution of an FSM state across multiple clock cycles





Toolchain

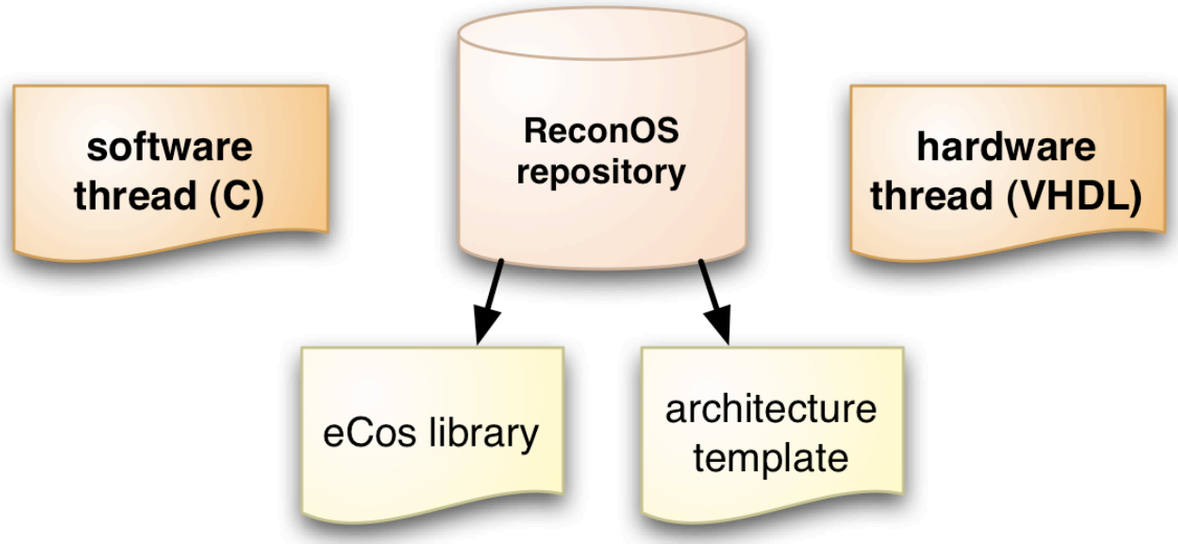


Toolchain


software thread (C)
ReconOS repository
hardware thread (VHDL)

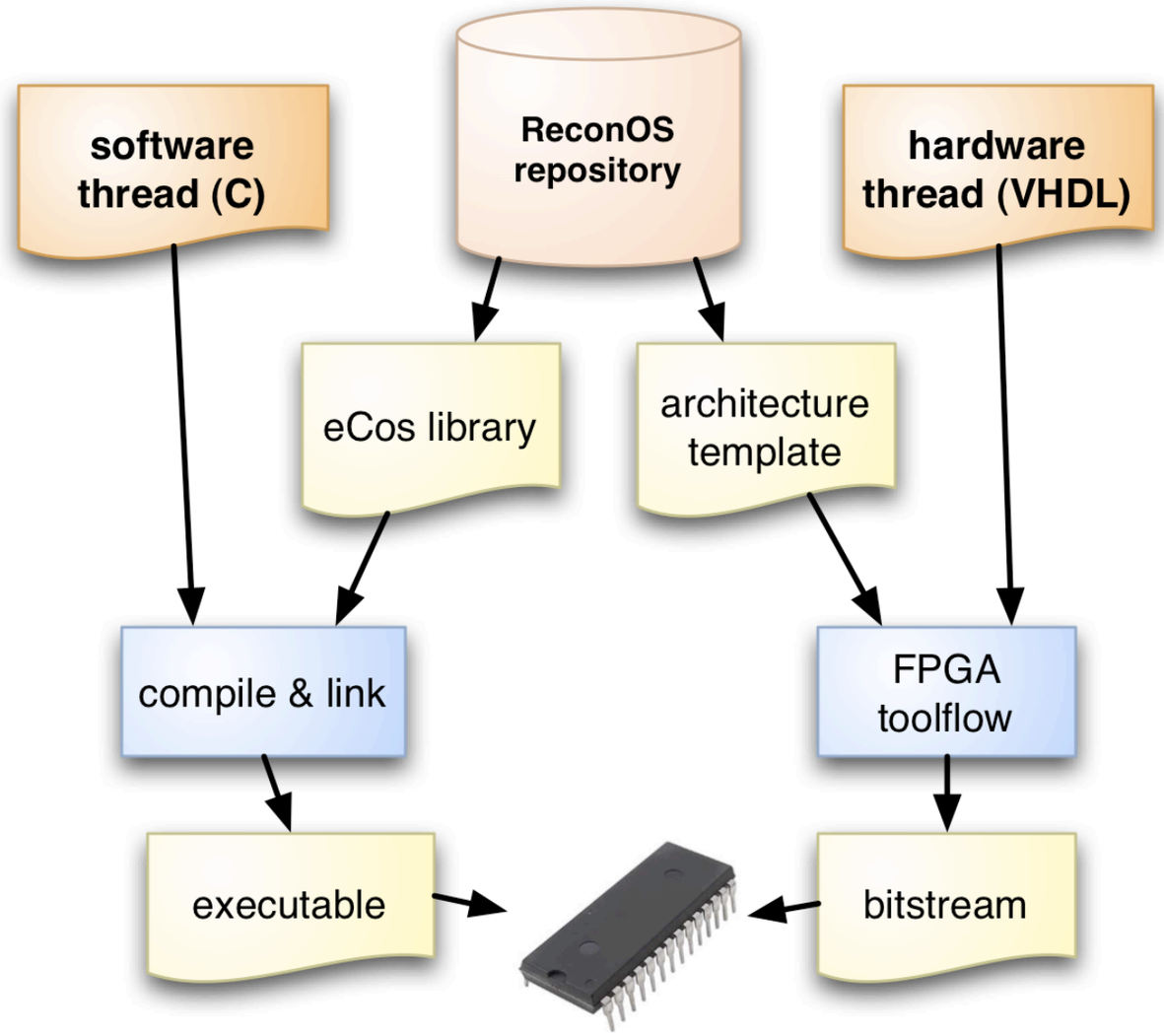
Toolchain

- software threads are written in C
  - using the eCos software API
- hardware threads are written in VHDL
  - using the ReconOS VHDL API
- architecture generation
  - automatically inserts OS interfaces and hardware threads into Xilinx EDK platform templates
  - configures and builds static eCos library



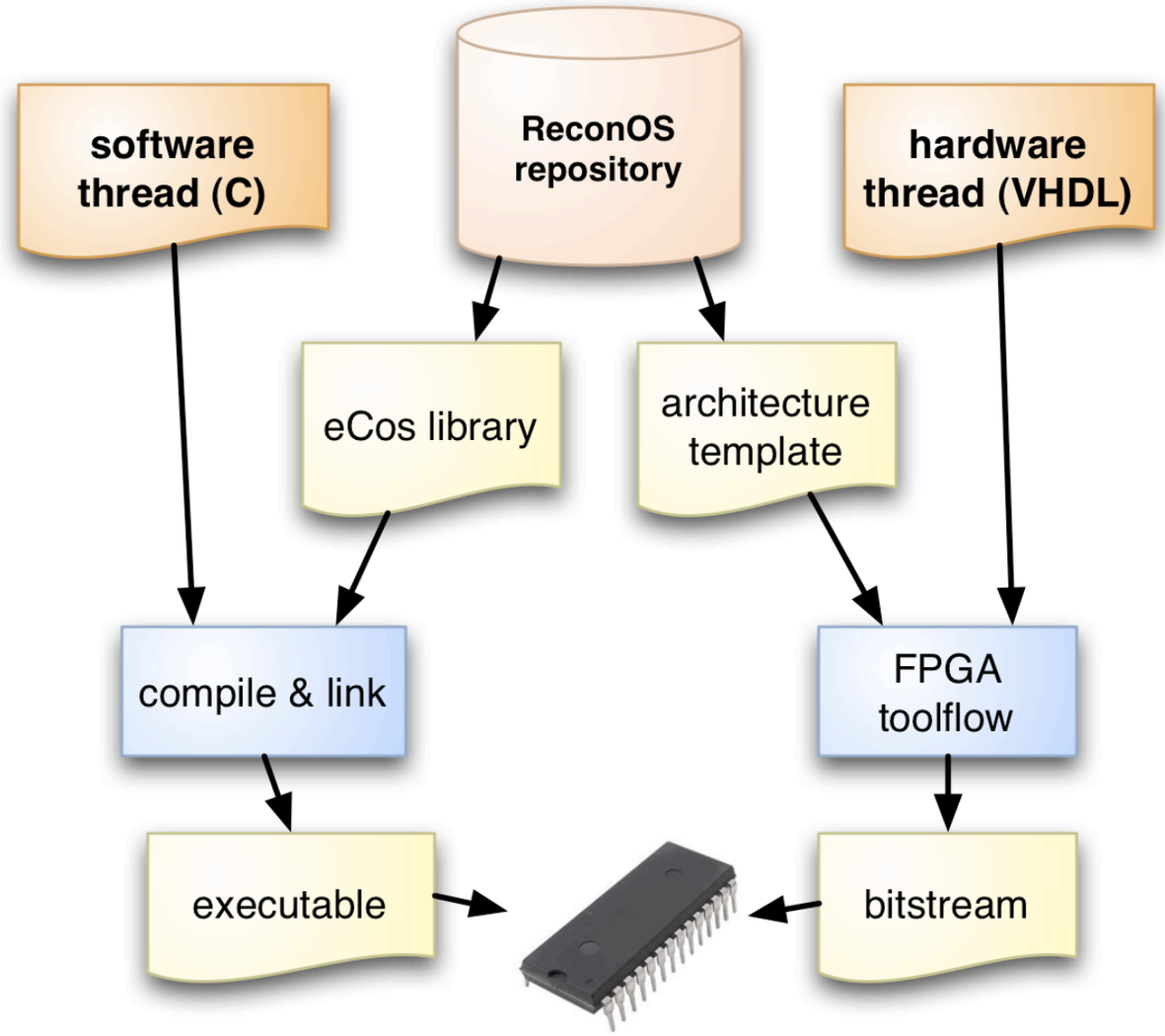
Toolchain

- software threads are written in C
  - using the eCos software API
- hardware threads are written in VHDL
  - using the ReconOS VHDL API
- architecture generation
  - automatically inserts OS interfaces and hardware threads into Xilinx EDK platform templates
  - configures and builds static eCos library



Toolchain

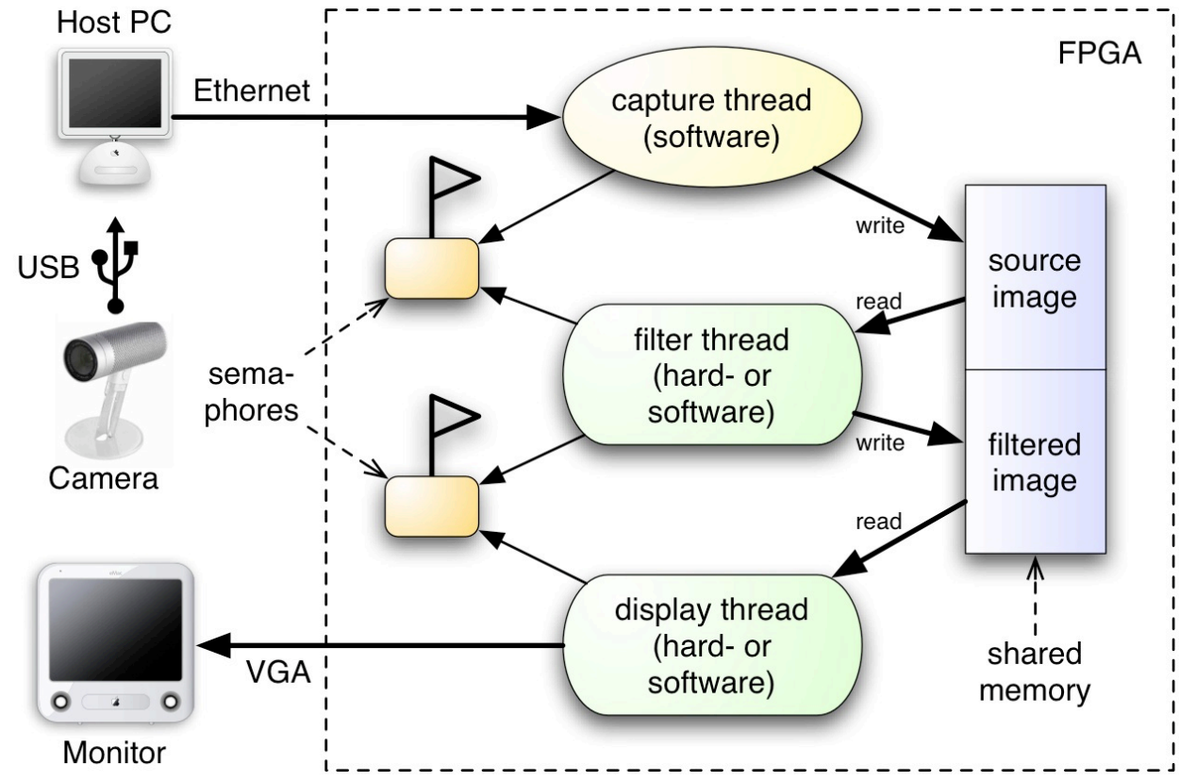
- software threads are written in C
  - using the eCos software API
- hardware threads are written in VHDL
  - using the ReconOS VHDL API
- architecture generation
  - automatically inserts OS interfaces and hardware threads into Xilinx EDK platform templates
  - configures and builds static eCos library

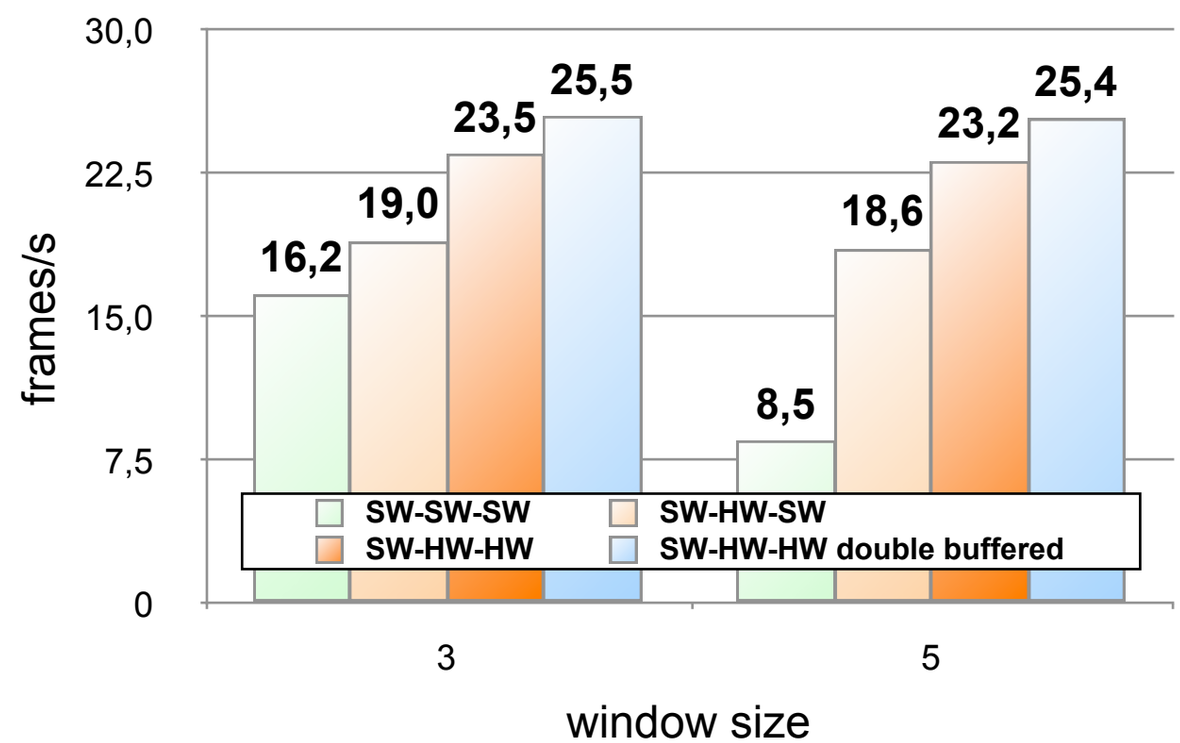


- eCos extensions
  - hardware thread object encapsulating delegate thread and OS interface “driver”
  - profiling support to track the state of the hardware threads' OS synchronization state machines

Case Study - Image Processing Filter

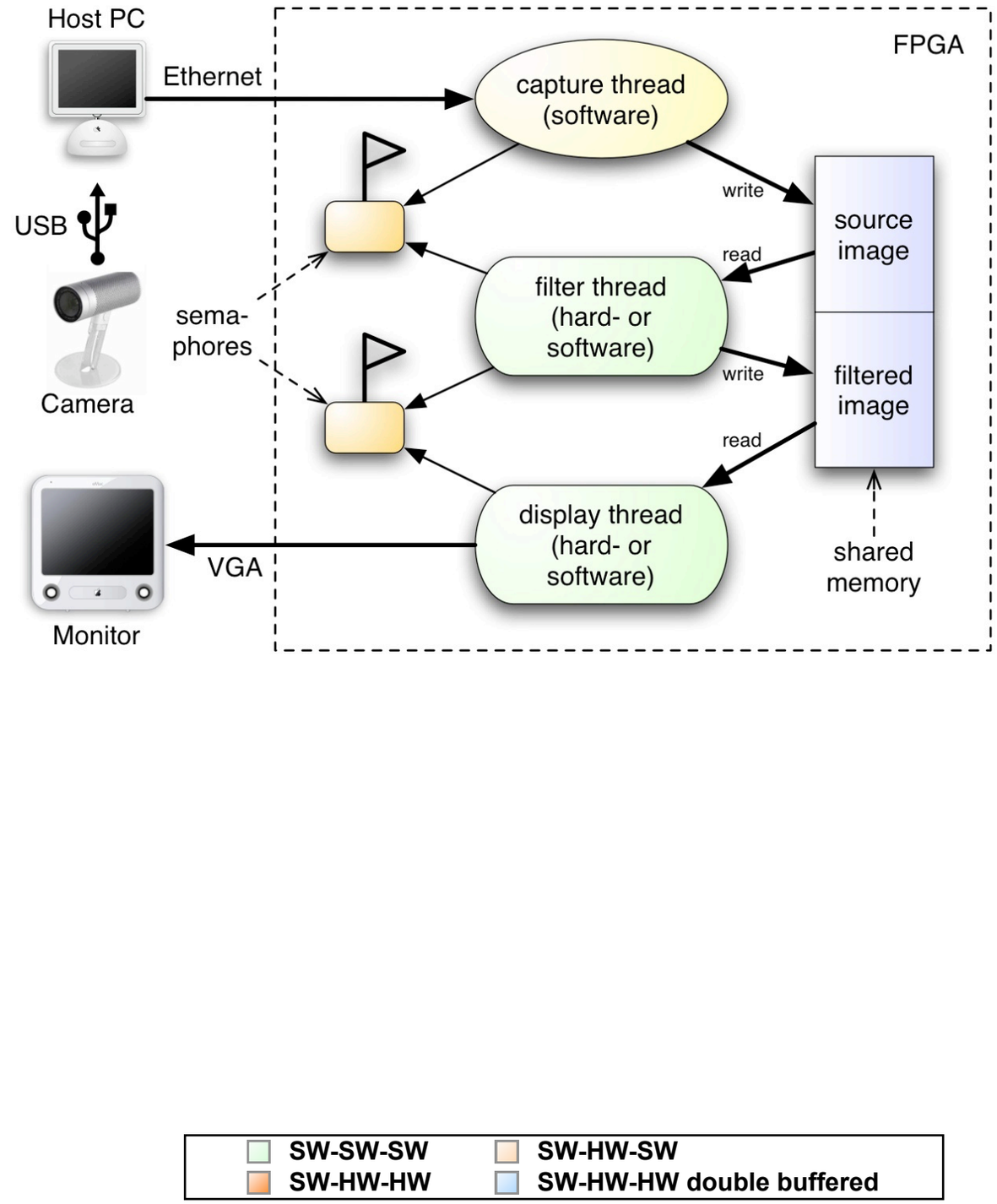
- three threads
  - capture image from Ethernet
  - apply LaPlacian filter
  - display image on VGA monitor
- threads communicate through shared memory
  - image resolution: 320x240 pixels, 8 bit greyscale
  - image data organized into blocks (e.g. 40 lines = 1 block)
  - a block is protected by two semaphores
    - “ready” semaphore: data can be safely written into this block
    - “new” semaphore: new data is available in this block



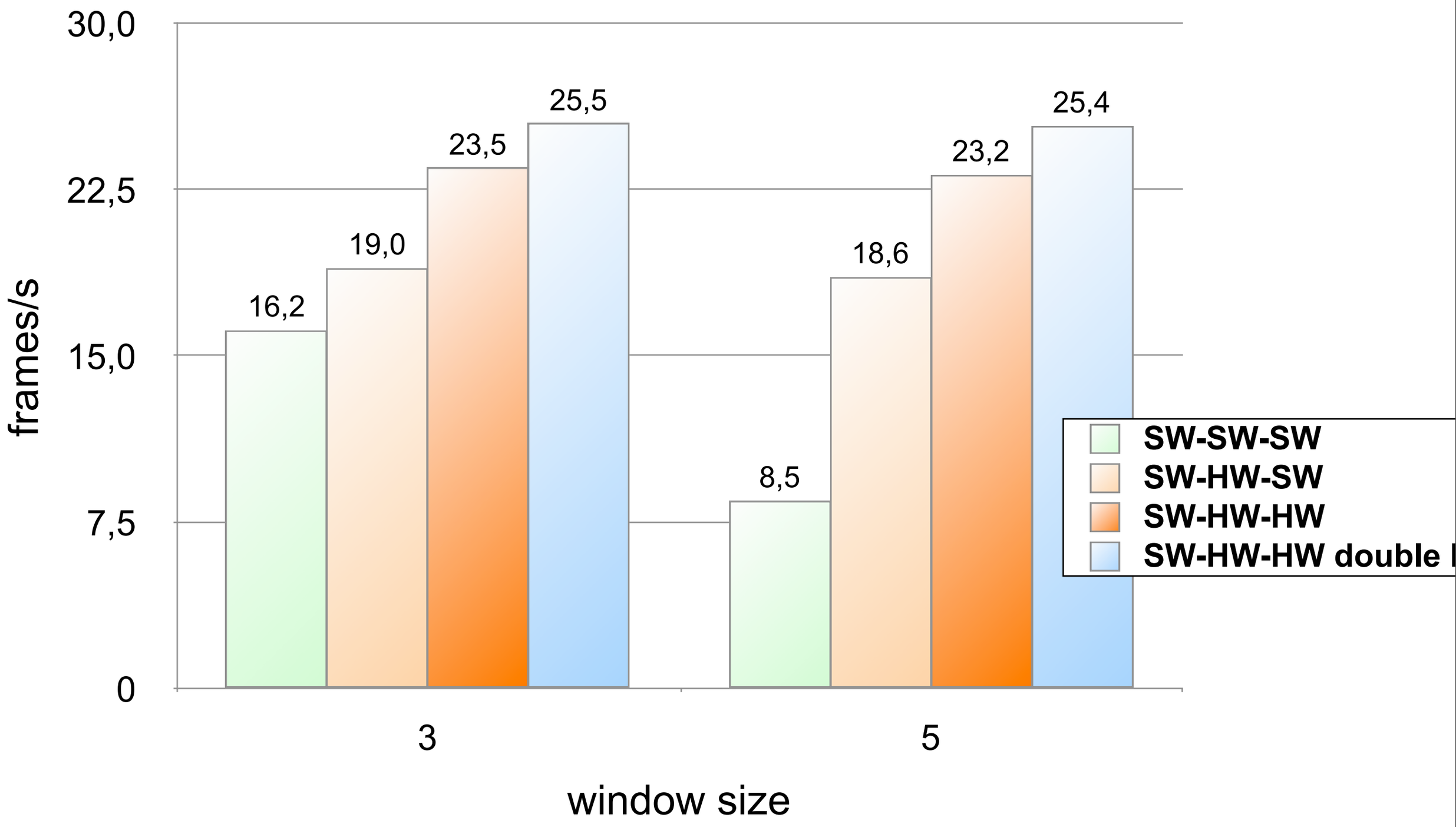


Case Study - Image Processing Filter

- three threads
  - capture image from Ethernet
  - apply LaPlacian filter
  - display image on VGA monitor
- threads communicate through shared memory
  - image resolution: 320x240 pixels, 8 bit greyscale
  - image data organized into blocks (e.g. 40 lines = 1 block)
  - a block is protected by two semaphores
    - “ready” semaphore: data can be safely written into this block
    - “new” semaphore: new data is available in this block



Case Study - Results



Case Study - Results