A Portable Abstraction Layer for Hardware Threads

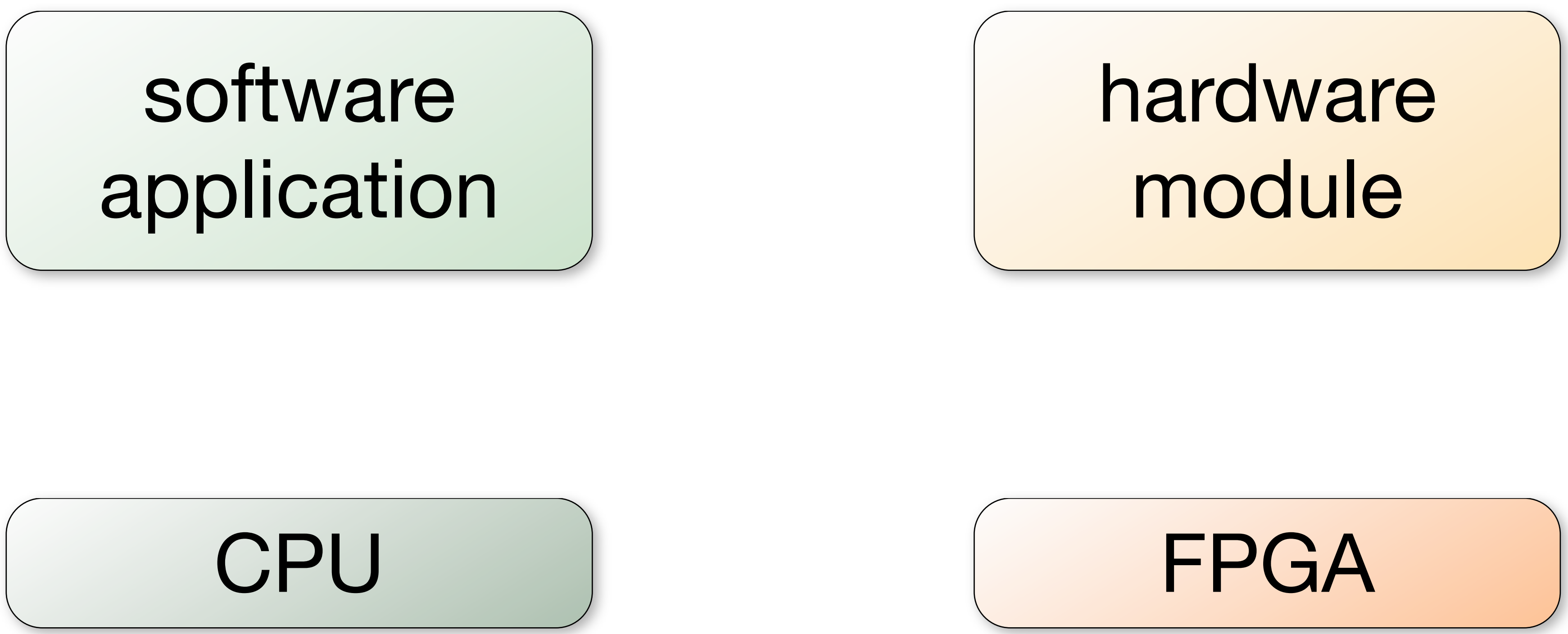
Enno Lübbers and Marco Platzner
Computer Engineering Group
University of Paderborn

{enno.luebbers, platzner}@upb.de

UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft

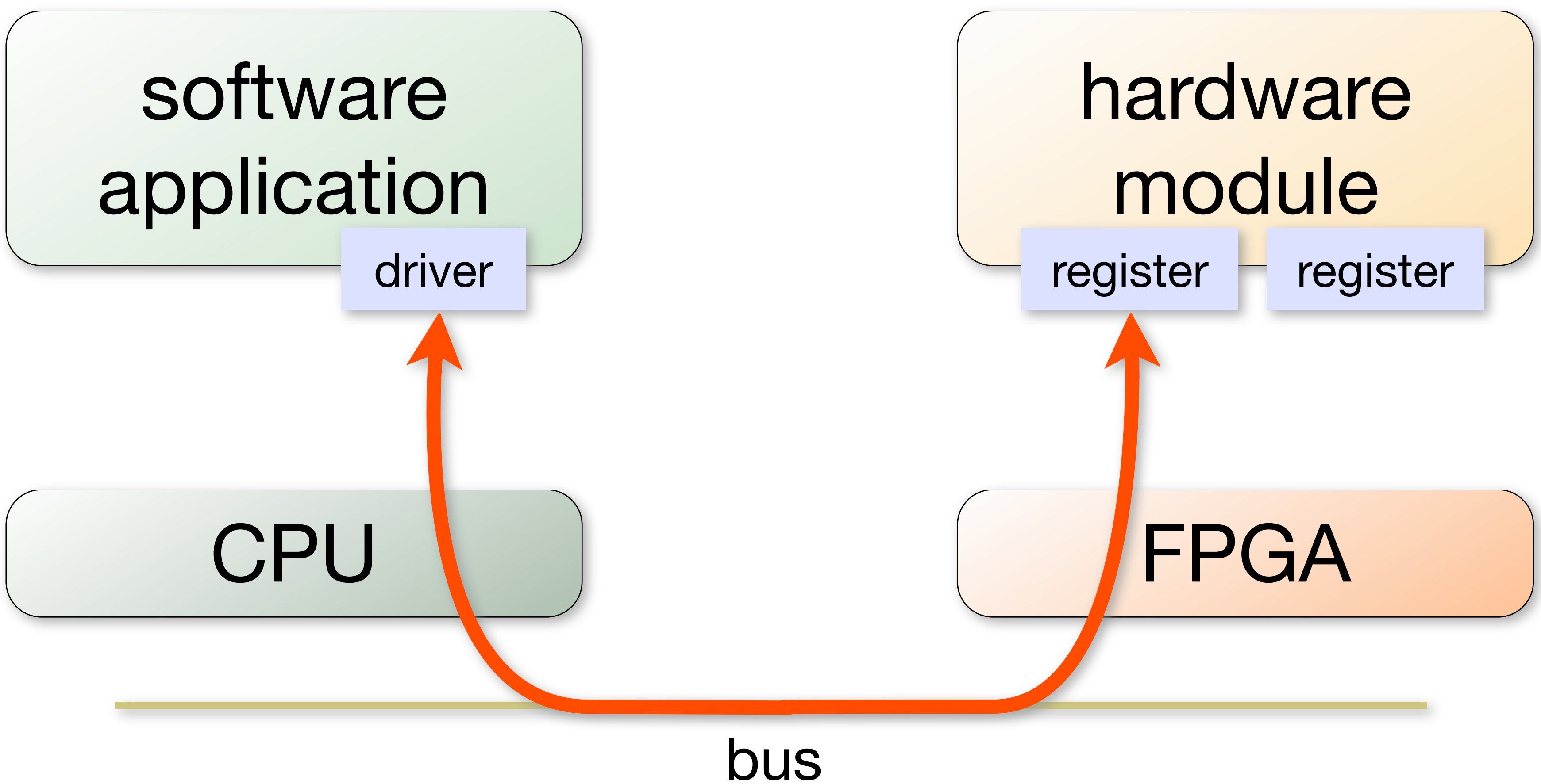
Design of CPU/FPGA Systems

- hardware modules typically integrated as slave coprocessors
- hardware/software boundary explicit
- tedious and error-prone to program
- portability issues



Design of CPU/FPGA Systems

- hardware modules typically integrated as slave coprocessors
- hardware/software boundary explicit
- tedious and error-prone to program
- portability issues

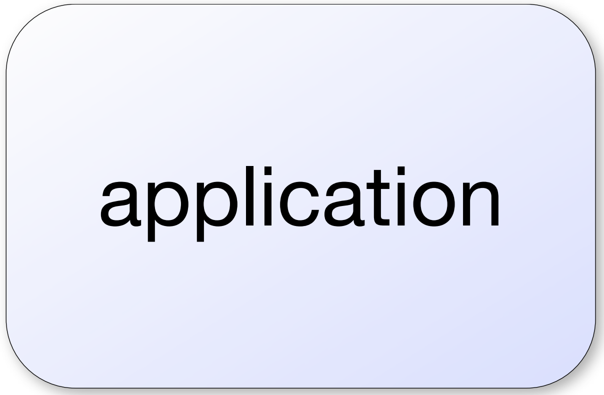


Multithreaded Programming

application

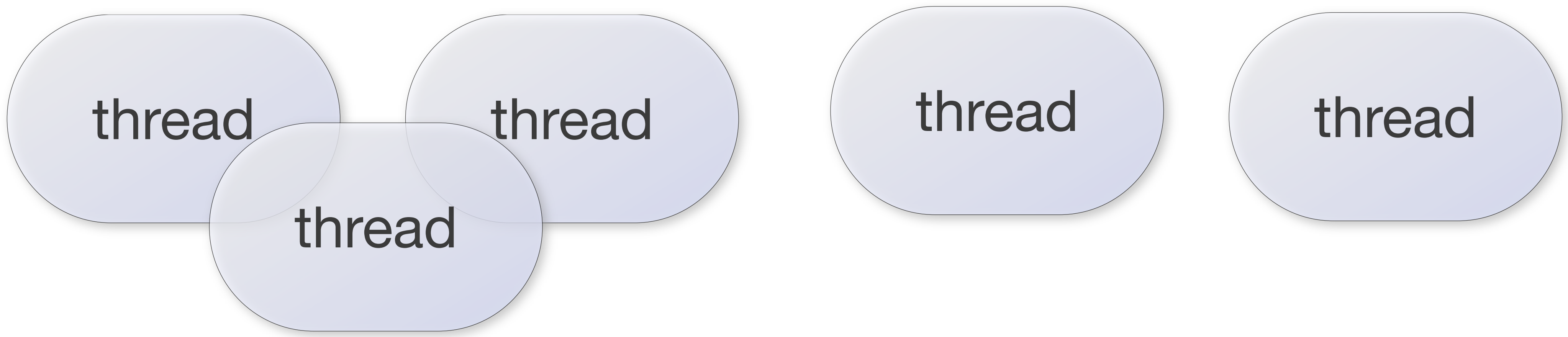
Multithreaded Programming

- multithreaded programming model



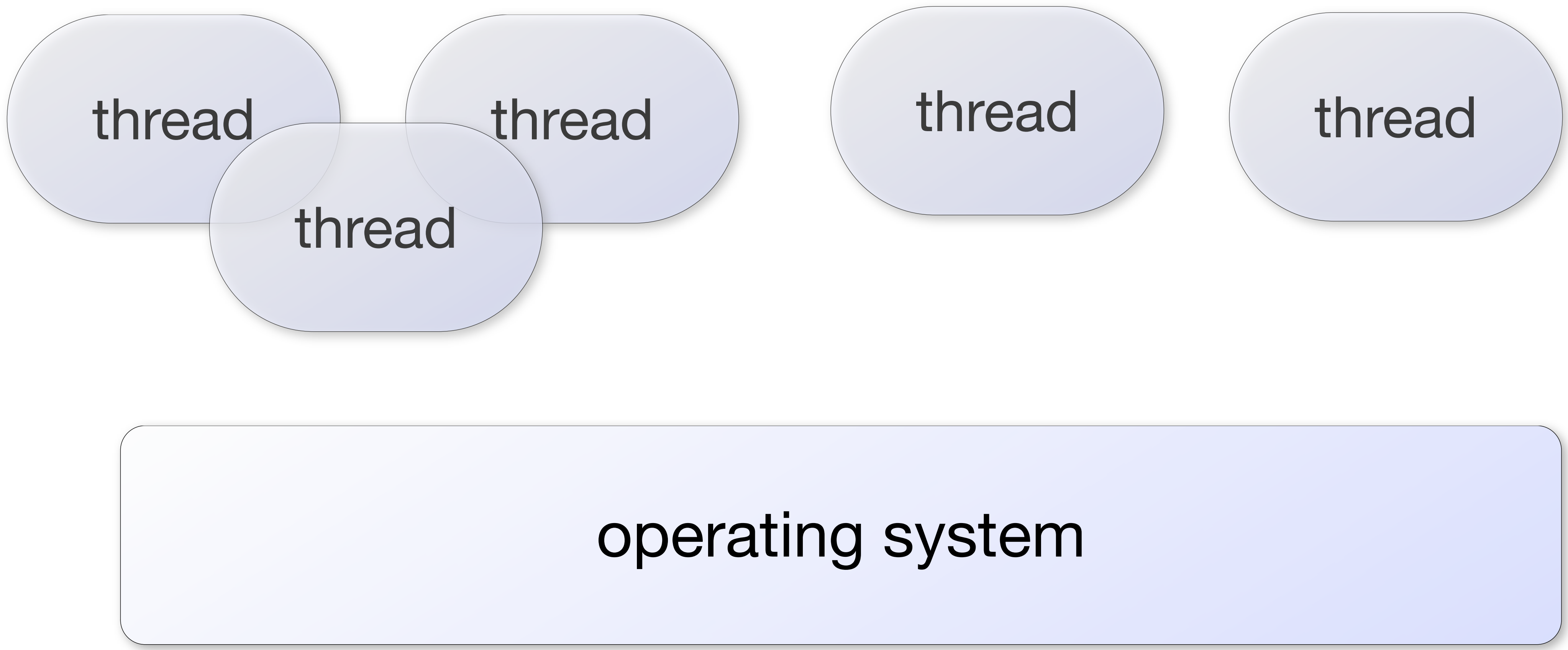
Multithreaded Programming

- multithreaded programming model



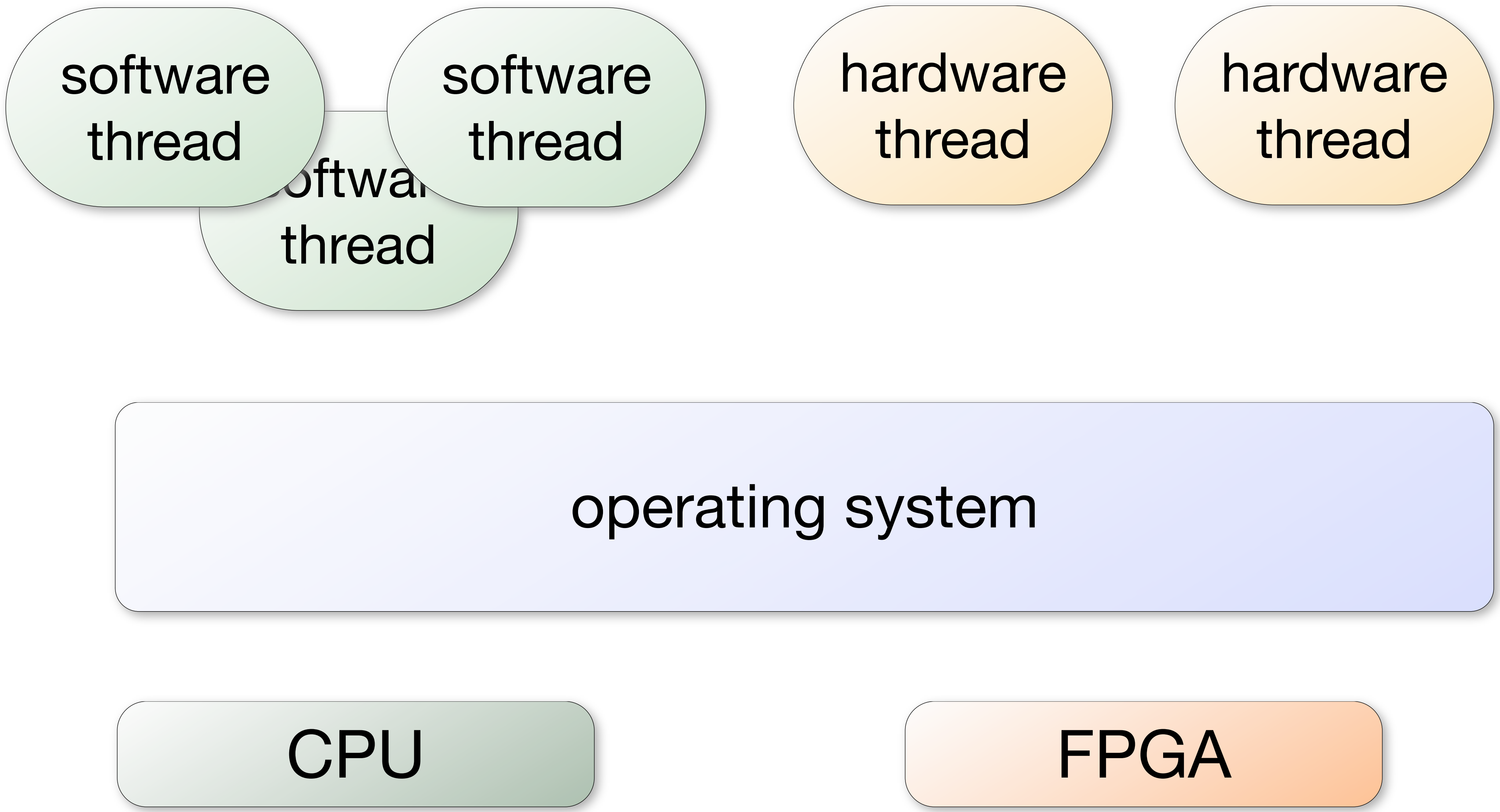
Multithreaded Programming

- multithreaded programming model



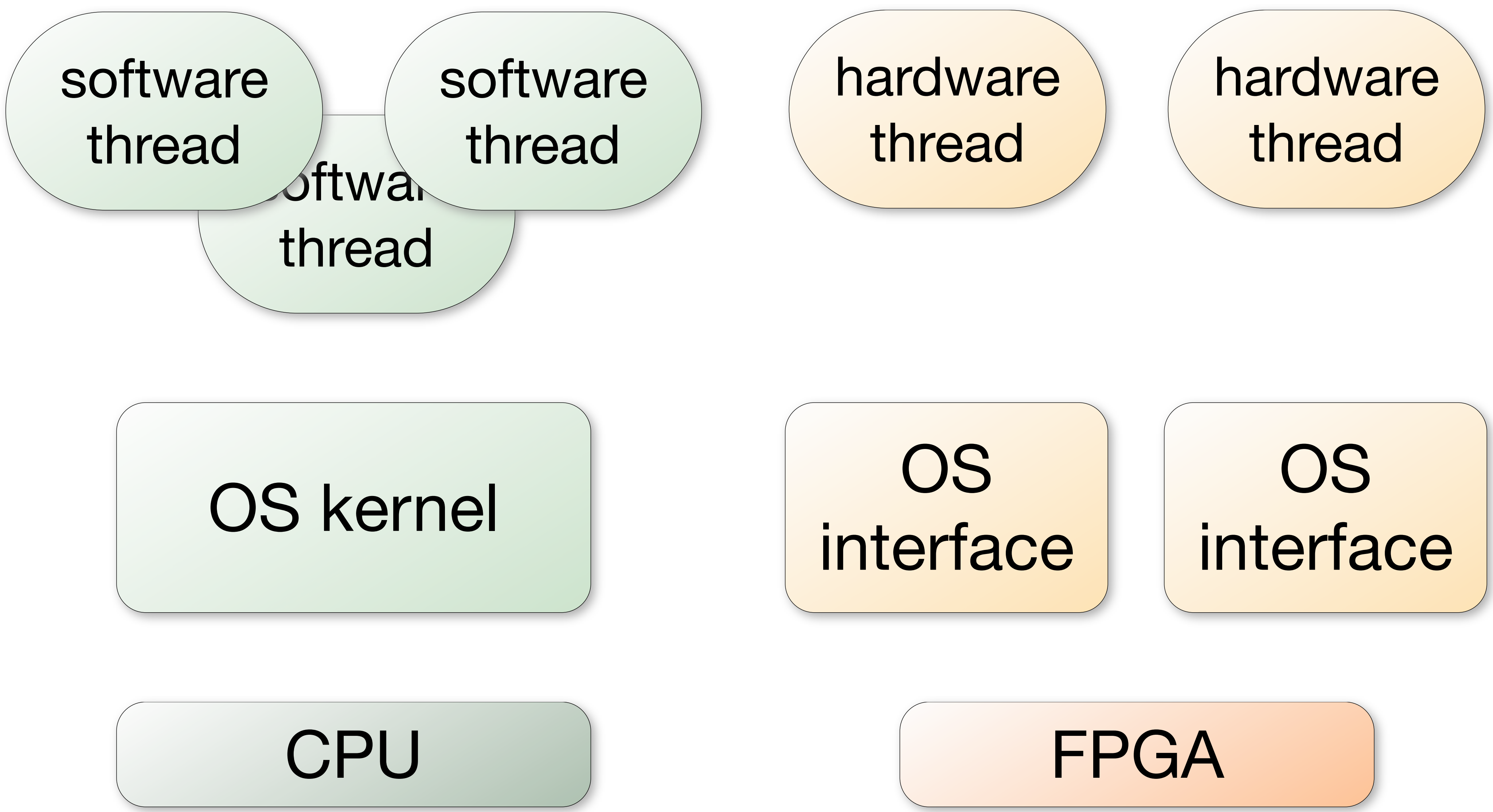
Multithreaded Programming

- multithreaded programming model
  - extended to reconfigurable hardware (ReconOS)



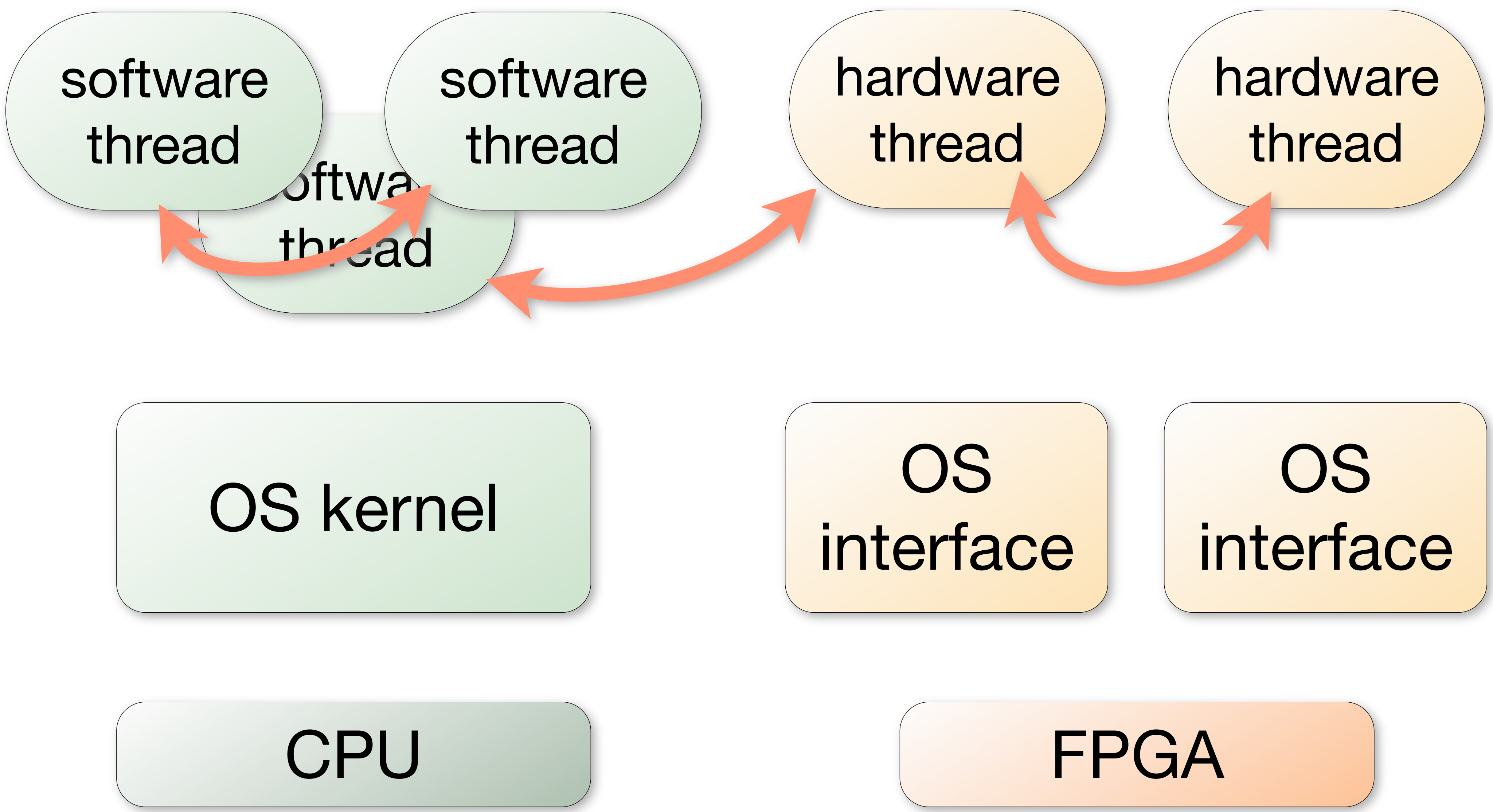
Multithreaded Programming

- multithreaded programming model
  - extended to reconfigurable hardware (ReconOS)



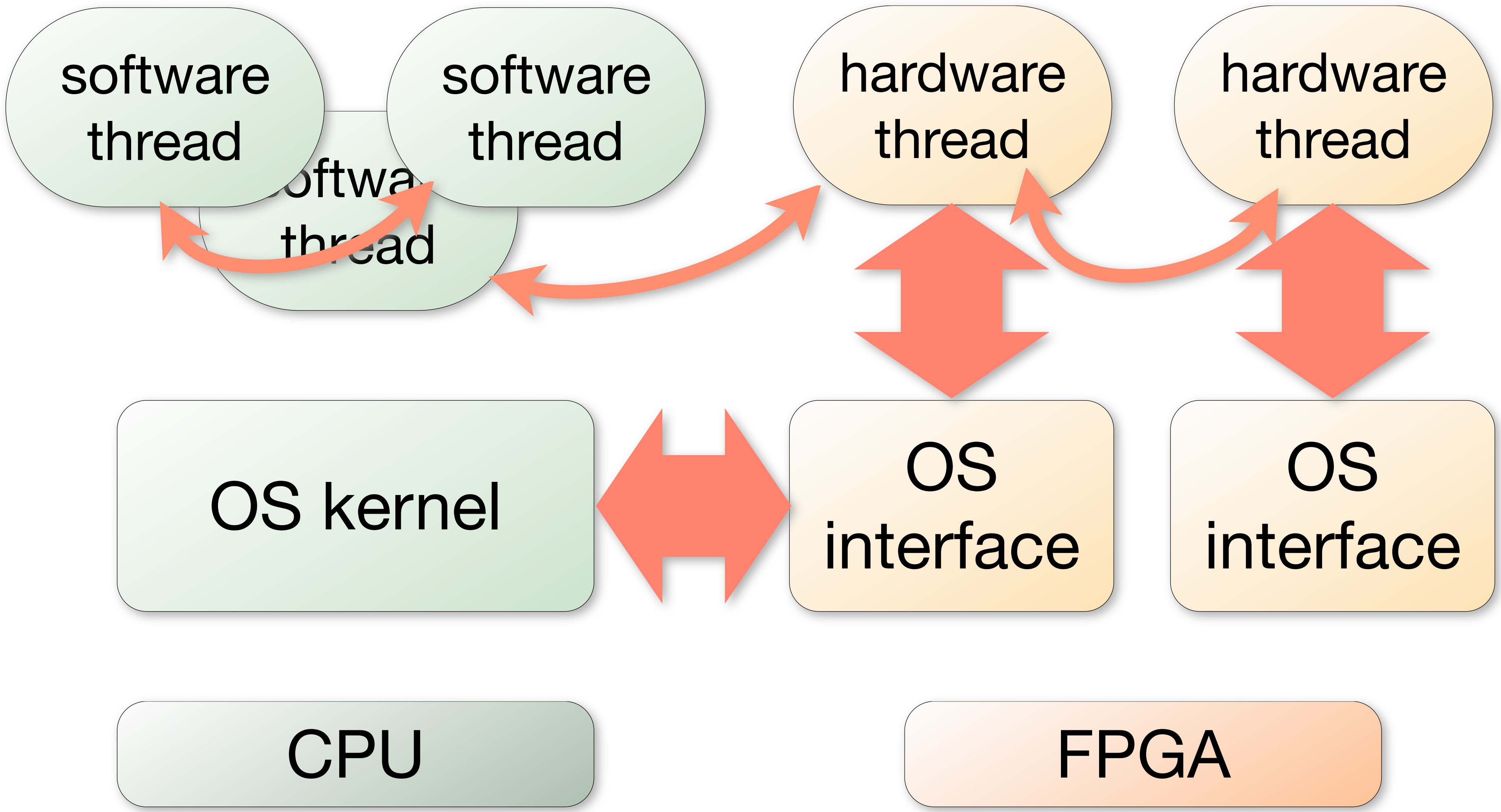
Multithreaded Programming

- multithreaded programming model
  - extended to reconfigurable hardware (ReconOS)
  - provides transparent synchronization and communication b/w threads



Multithreaded Programming

- multithreaded programming model
  - extended to reconfigurable hardware (ReconOS)
  - provides transparent synchronization and communication b/w threads
  - operating system provides low-level synchronization and communication



Application Domains

Application Domains

- multithreaded programming model applicable to several application domains, e.g.

Application Domains

- multithreaded programming model applicable to several application domains, e.g.

- **reconfigurable embedded computing**
  - efficient exploitation of fine-grained parallelism with tight constraints (memory, area, power, processor performance)
  - demand for easy design space exploration regarding HW/SW partitioning
  - short reaction times, possibly real-time requirements

Application Domains

- multithreaded programming model applicable to several application domains, e.g.

- **reconfigurable embedded computing**
  - efficient exploitation of fine-grained parallelism with tight constraints (memory, area, power, processor performance)
  - demand for easy design space exploration regarding HW/SW partitioning
  - short reaction times, possibly real-time requirements
- **reconfigurable high-performance computing**
  - transparent communication and synchronization in heterogeneous execution environments (e.g. CPU nodes + FPGA accelerators)
  - exploitation of both fine-grained and thread-level parallelism, possibly across multiple machines

Application Domains

- multithreaded programming model applicable to several application domains, e.g.
- **reconfigurable embedded computing**
  - efficient exploitation of fine-grained parallelism with tight constraints (memory, area, power, processor performance)
  - demand for easy design space exploration regarding HW/SW partitioning
  - short reaction times, possibly real-time requirements
- **reconfigurable high-performance computing**
  - transparent communication and synchronization in heterogeneous execution environments (e.g. CPU nodes + FPGA accelerators)
  - exploitation of both fine-grained and thread-level parallelism, possibly across multiple machines

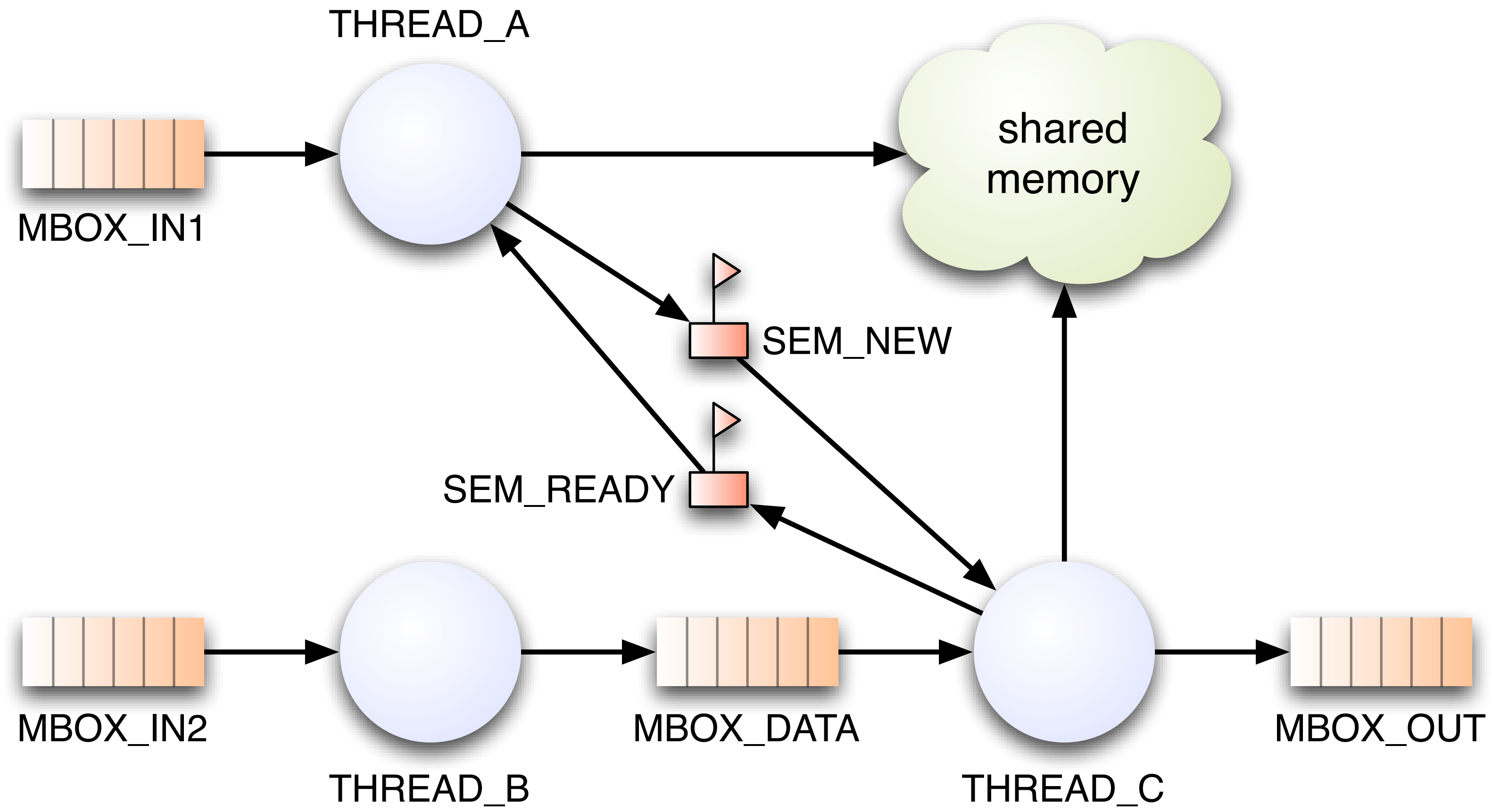
➡ show applicability of ReconOS approach across different host operating systems and CPU/FPGA architectures

Overview

- motivation
- ReconOS abstraction layer
  - programming model
  - hardware architecture
  - hardware threads
  - OS interface & delegate threads
- host OS implementations
  - ReconOS/eCos
  - ReconOS/Linux
- experimental results
- conclusion

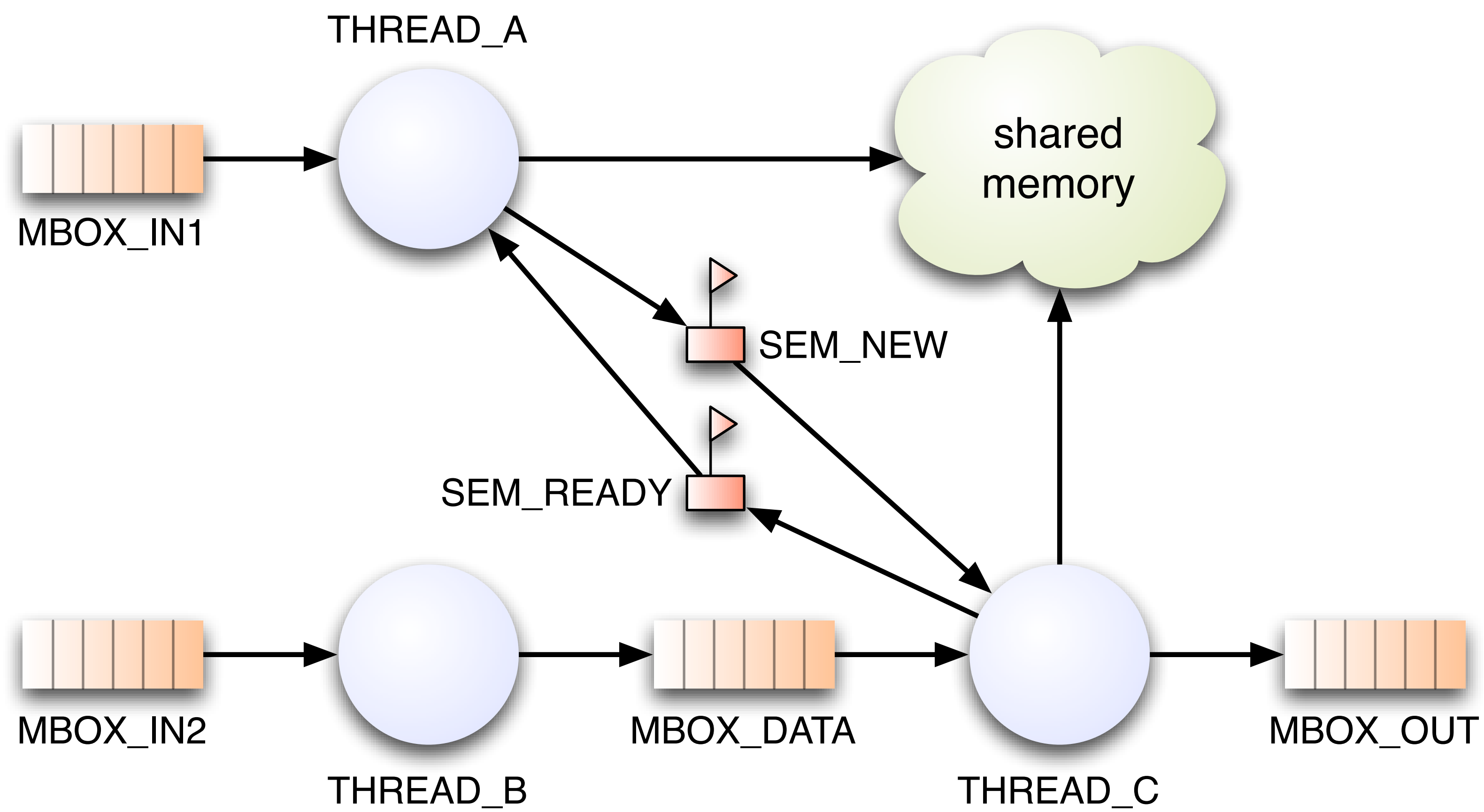
Programming Model

- 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
  - ...

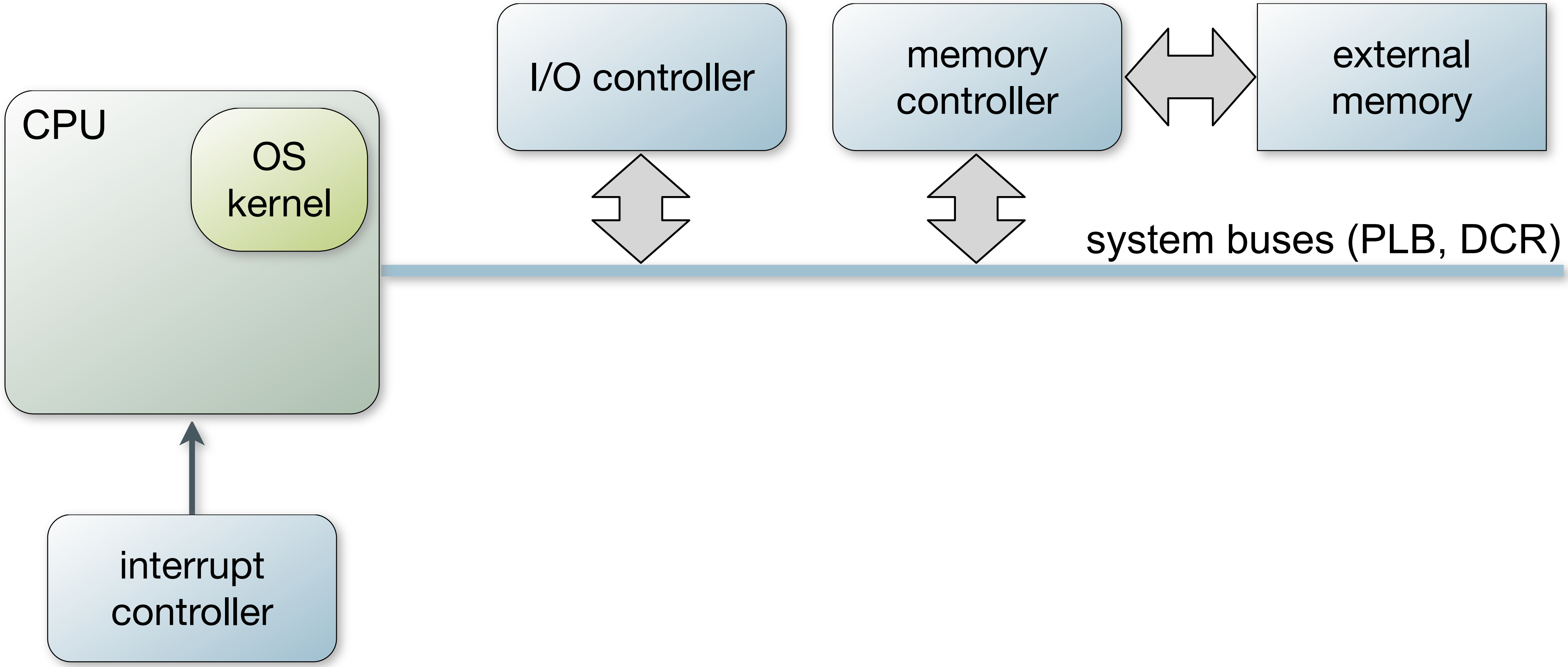


examples for API functions used by threads

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()`

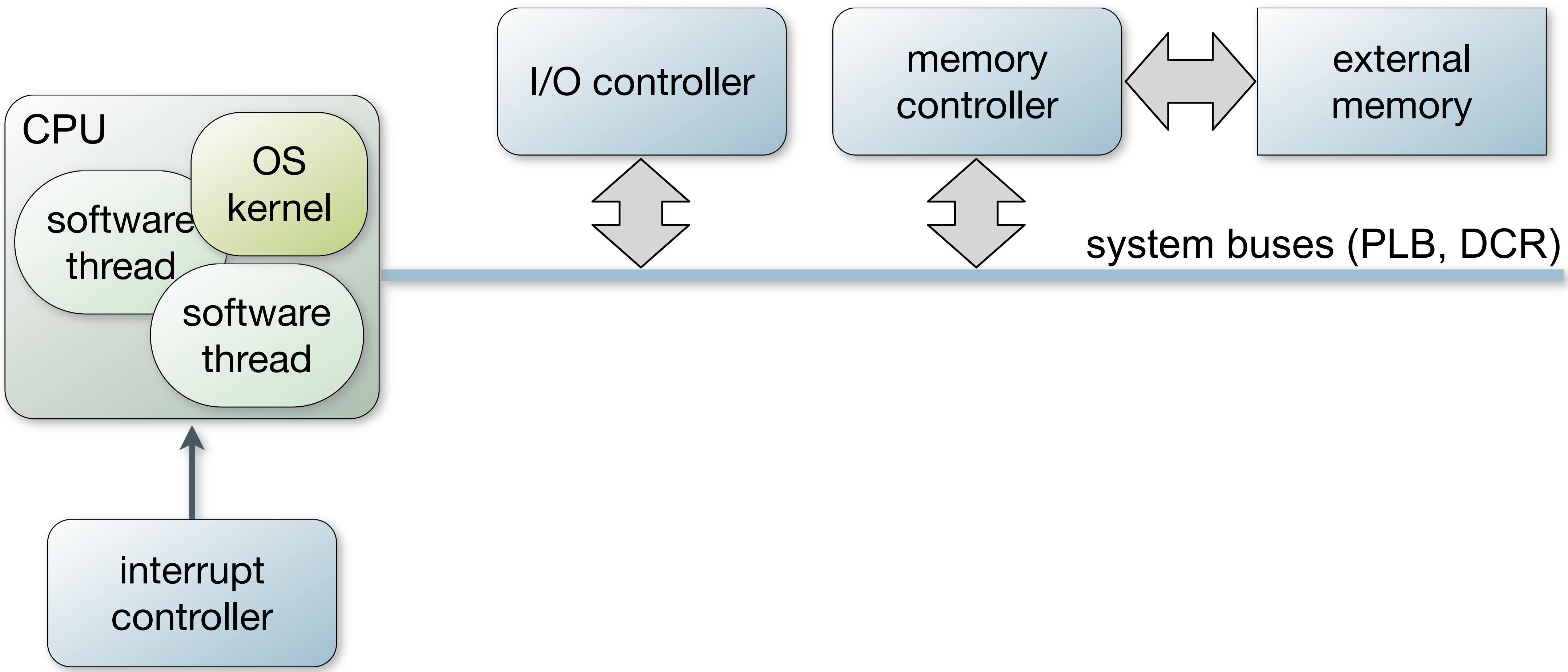
Hardware Architecture

- based on CoreConnect bus topology
- system CPU runs OS kernel and software threads
- hardware threads are synthesized to FPGA fabric
  - connected to OS kernel via OS interface modules and buses



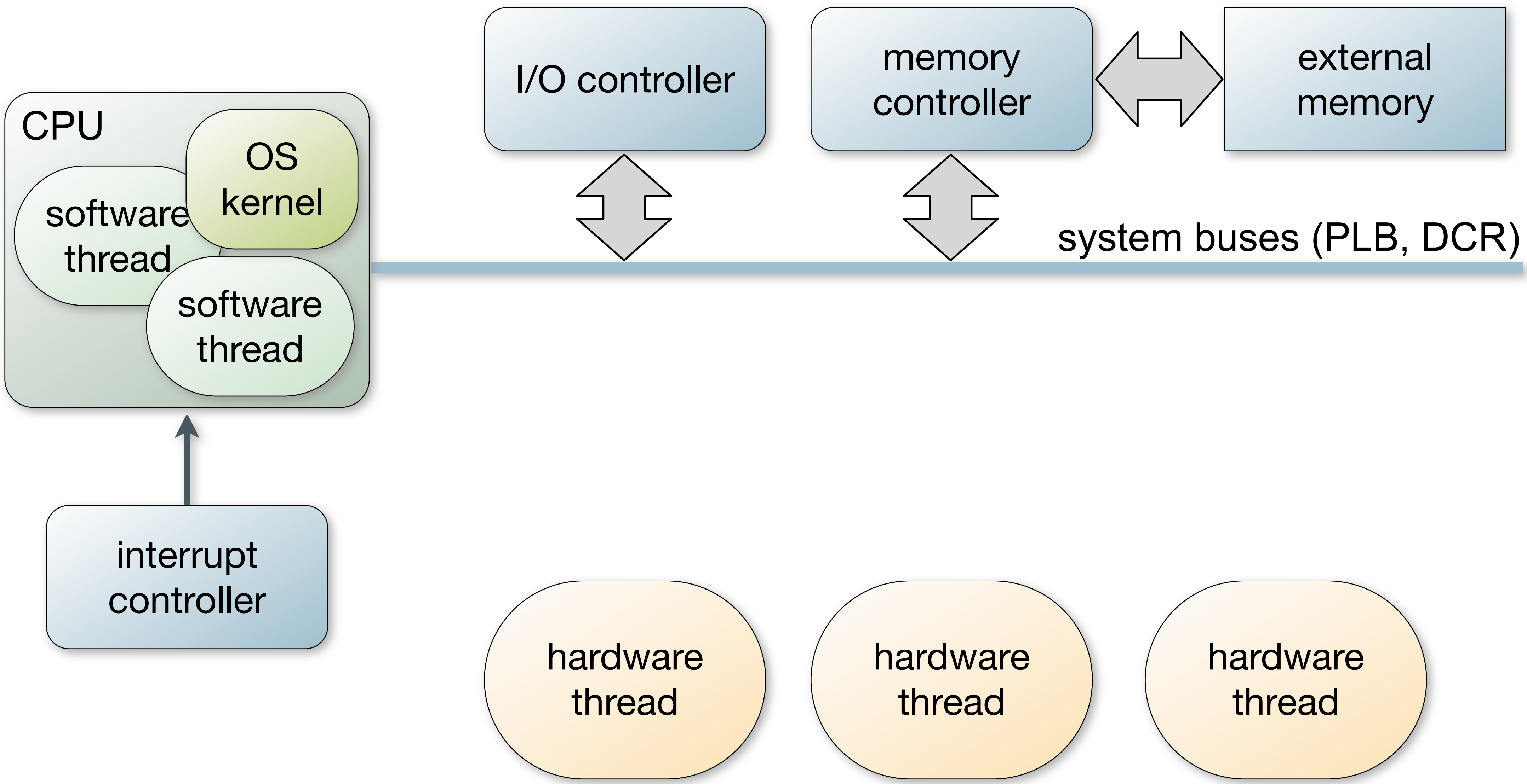
Hardware Architecture

- based on CoreConnect bus topology
- system CPU runs OS kernel and software threads
- hardware threads are synthesized to FPGA fabric
  - connected to OS kernel via OS interface modules and buses



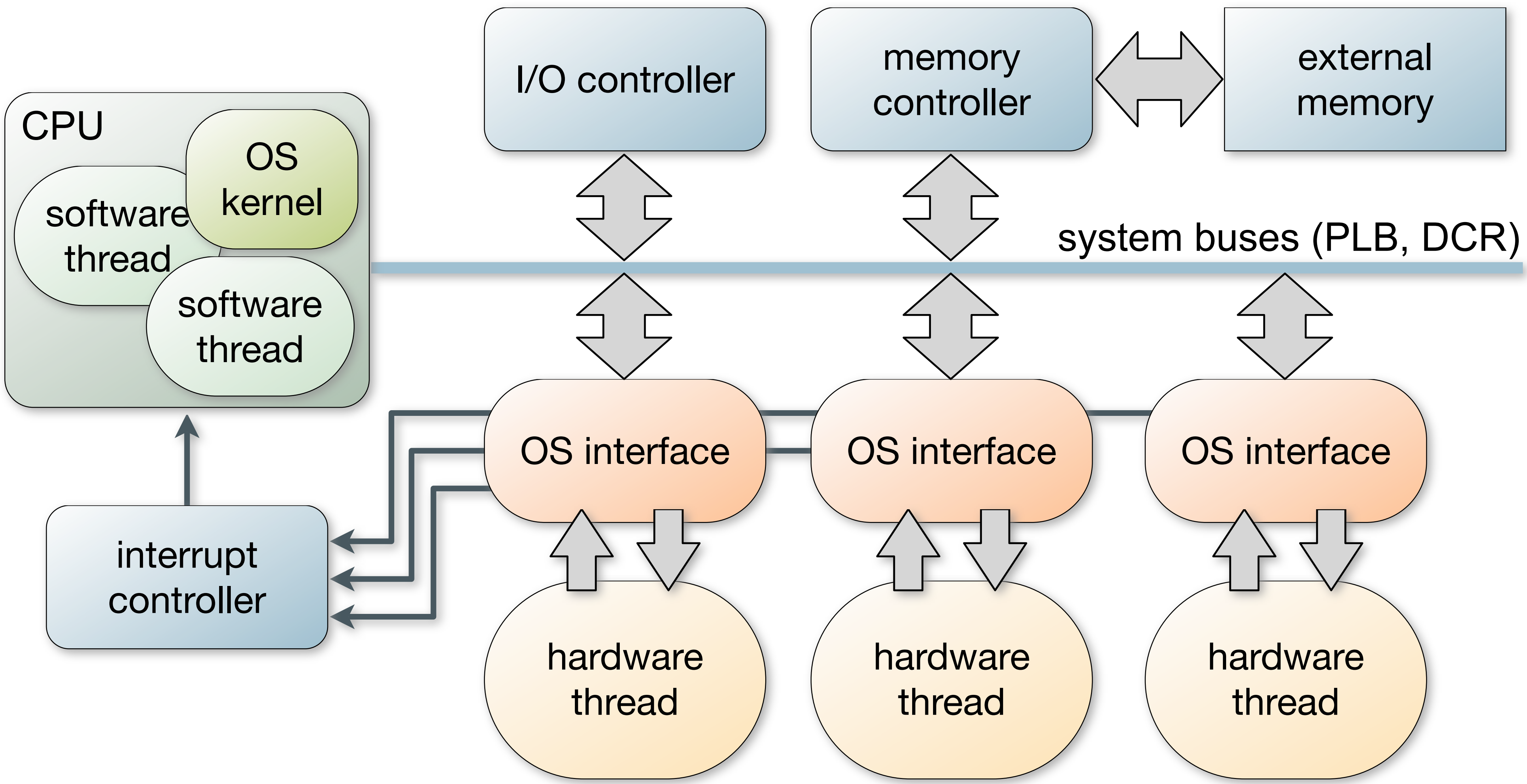
Hardware Architecture

- based on CoreConnect bus topology
- system CPU runs OS kernel and software threads
- hardware threads are synthesized to FPGA fabric
  - connected to OS kernel via OS interface modules and buses



Hardware Architecture

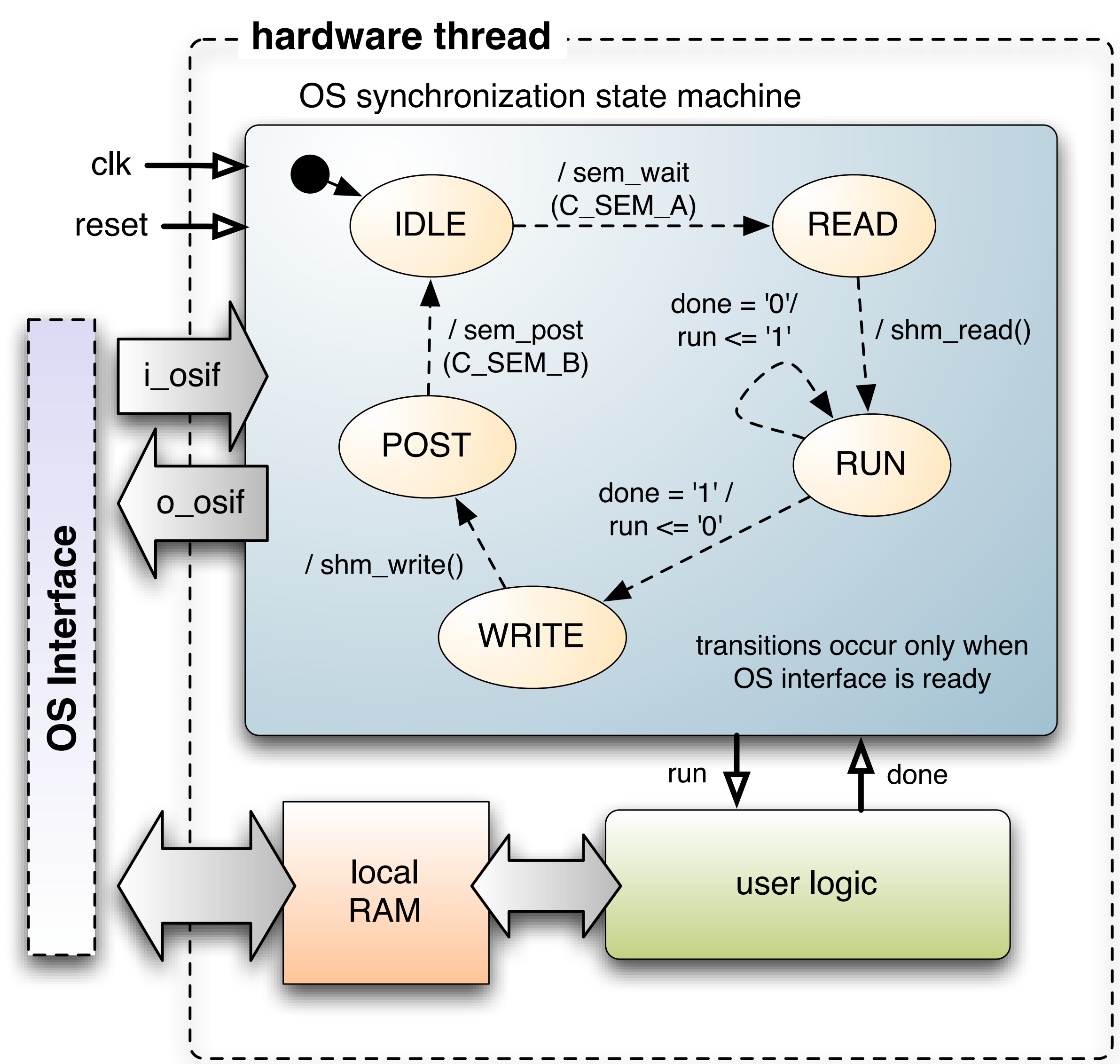
- based on CoreConnect bus topology
- system CPU runs OS kernel and software threads
- hardware threads are synthesized to FPGA fabric
  - connected to OS kernel via OS interface modules and buses



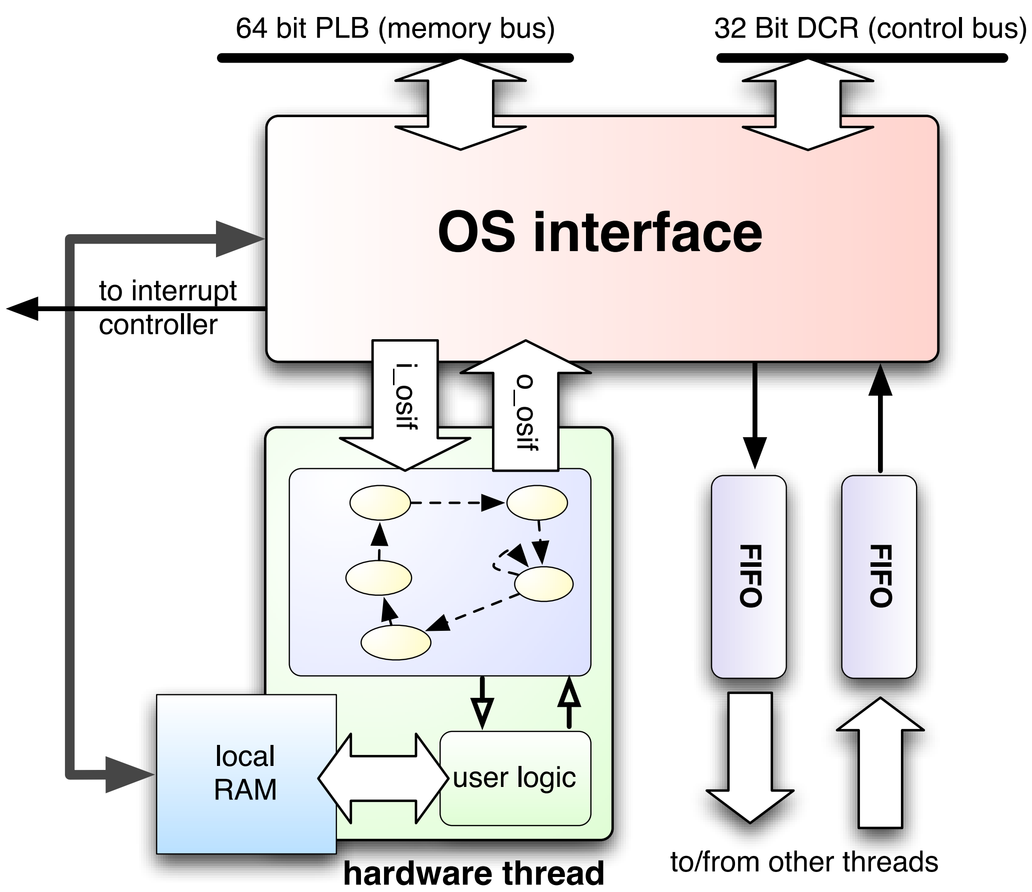
ReconOS API for Hardware Threads

```
osif_fsm: process(clk, reset)
begin
  if (reset = '1') then
    state <= IDLE;
    run <= '0';
    reconos_reset(o_osif, i_osif);
  elsif rising_edge(clk) then
    reconos_begin(o_osif, i_osif);
    if reconos_ready(i_osif) then
      case state is
        when IDLE =>
          reconos_sem_wait(o_osif, i_osif, C_SEM_A);
          state <= READ;

        when READ =>
          reconos_shm_read_burst(o_osif, i_osif,
                                 local_address,
                                 global_address);
          state <= RUN;

         when RUN =>
          run <= '1';
          if done = '1' then
            run <= '0';
            state <= WRITE;
          end if;

        when WRITE =>
          reconos_shm_write_burst(o_osif, i_osif,
                                  local_address,
                                  global_address);
          state <= POST;

        when POST =>
          reconos_sem_post(o_osif, i_osif, C_SEM_B);
          state <= IDLE;

        when others => null;
      end case;
    end if;
  end if;
end process;
```

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

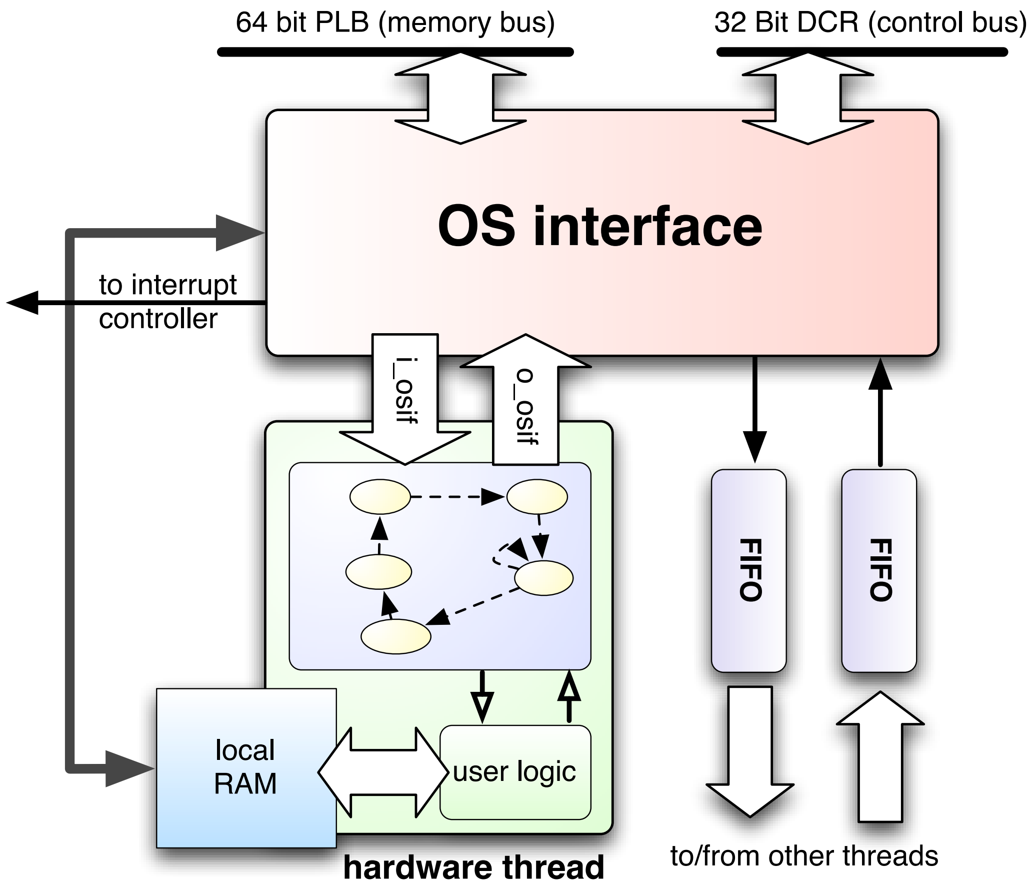


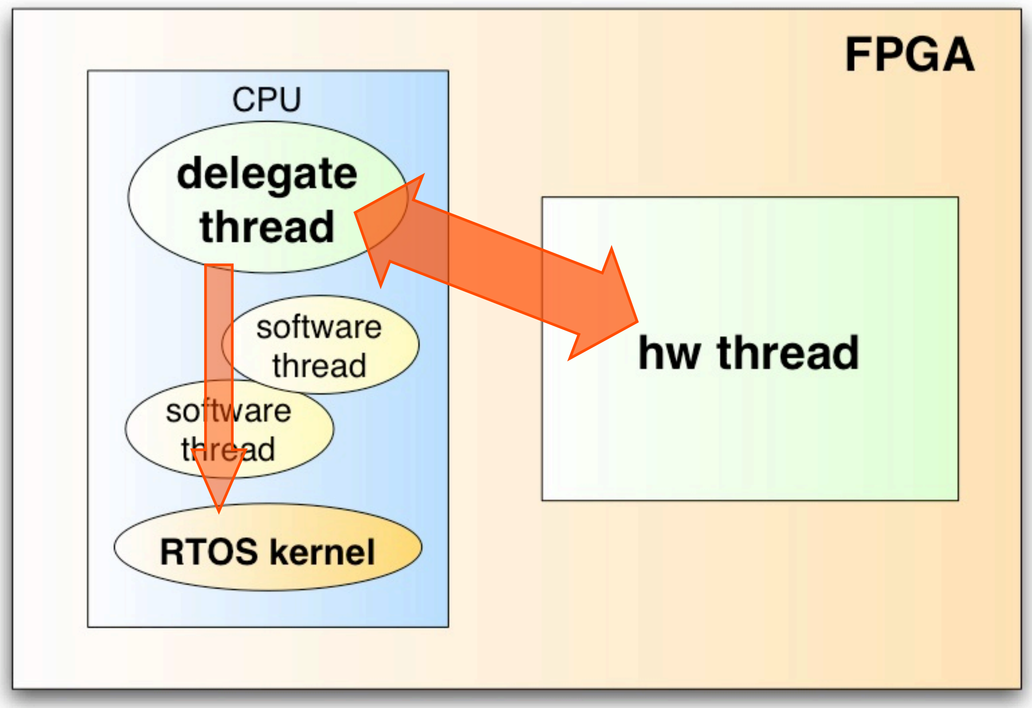
OS Interface and Delegate Threads



- OS interface
  - processes requests from HW thread
  - relays OS object interactions to CPU
  - executes memory accesses
  - provides dedicated FIFO channels

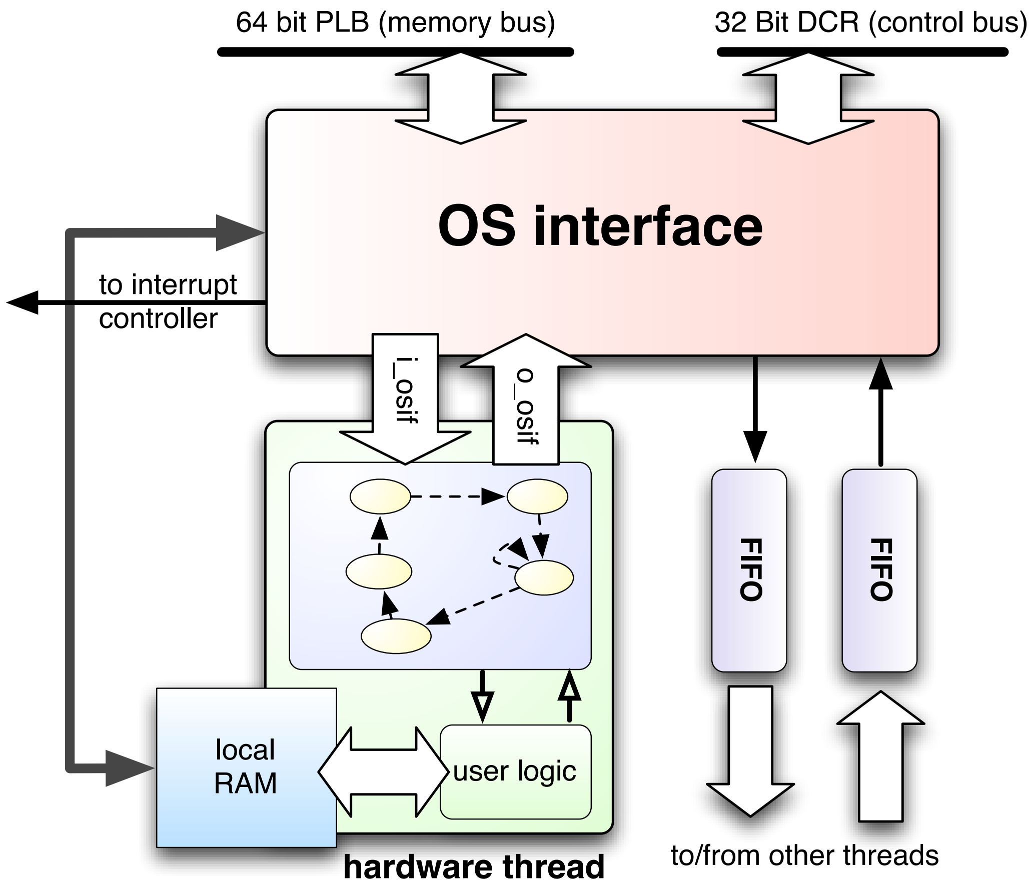
OS Interface and Delegate Threads

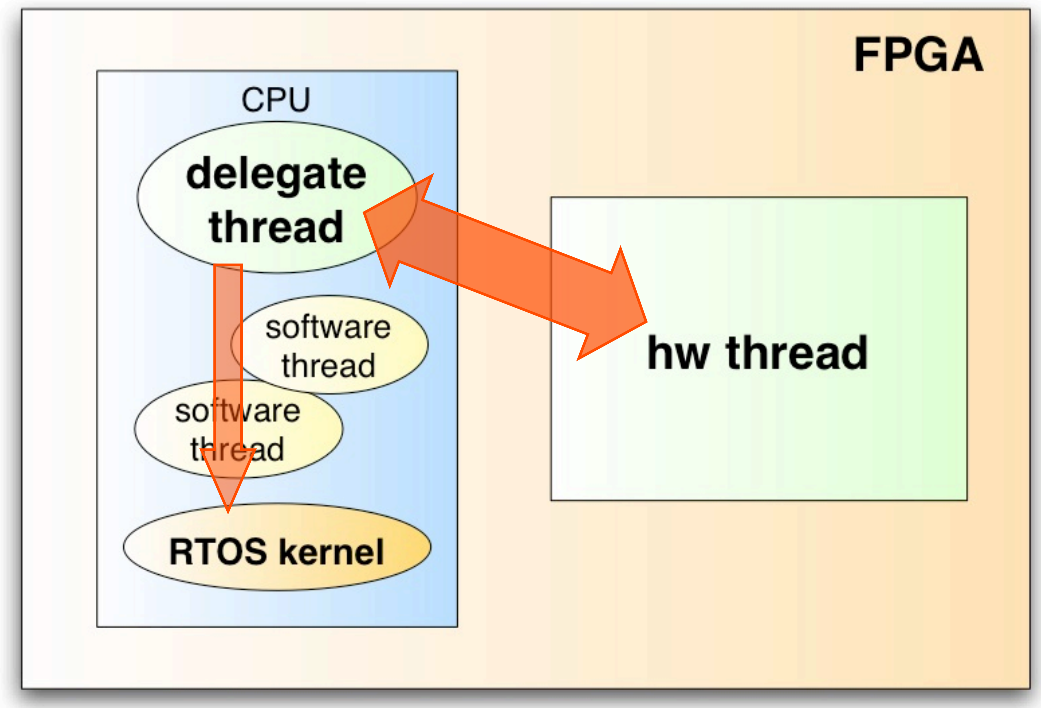




- OS interface
  - processes requests from HW thread
  - relays OS object interactions to CPU
  - executes memory accesses
  - provides dedicated FIFO channels
- delegate thread
  - associated with every hardware thread
  - calls kernel functions on behalf of hardware thread

OS Interface and Delegate Threads



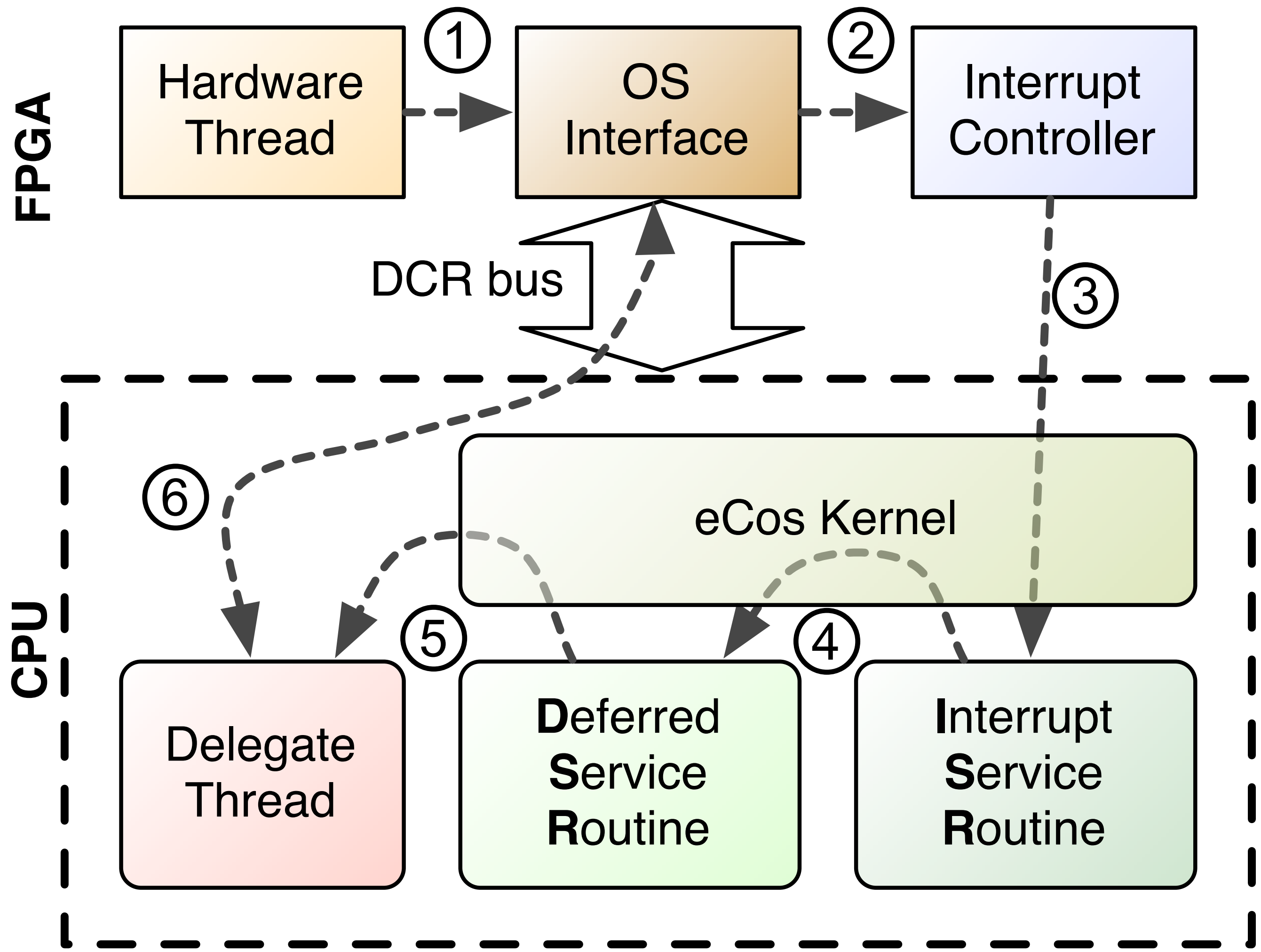


- OS interface
  - processes requests from HW thread
  - relays OS object interactions to CPU
  - executes memory accesses
  - provides dedicated FIFO channels
- delegate thread
  - associated with every hardware thread
  - calls kernel functions on behalf of hardware thread
- provide stable API on different OS‘s and platforms
  - OS interface manages low-level communication to CPU and memory
  - delegate translates HW thread requests to OS kernel API

Overview

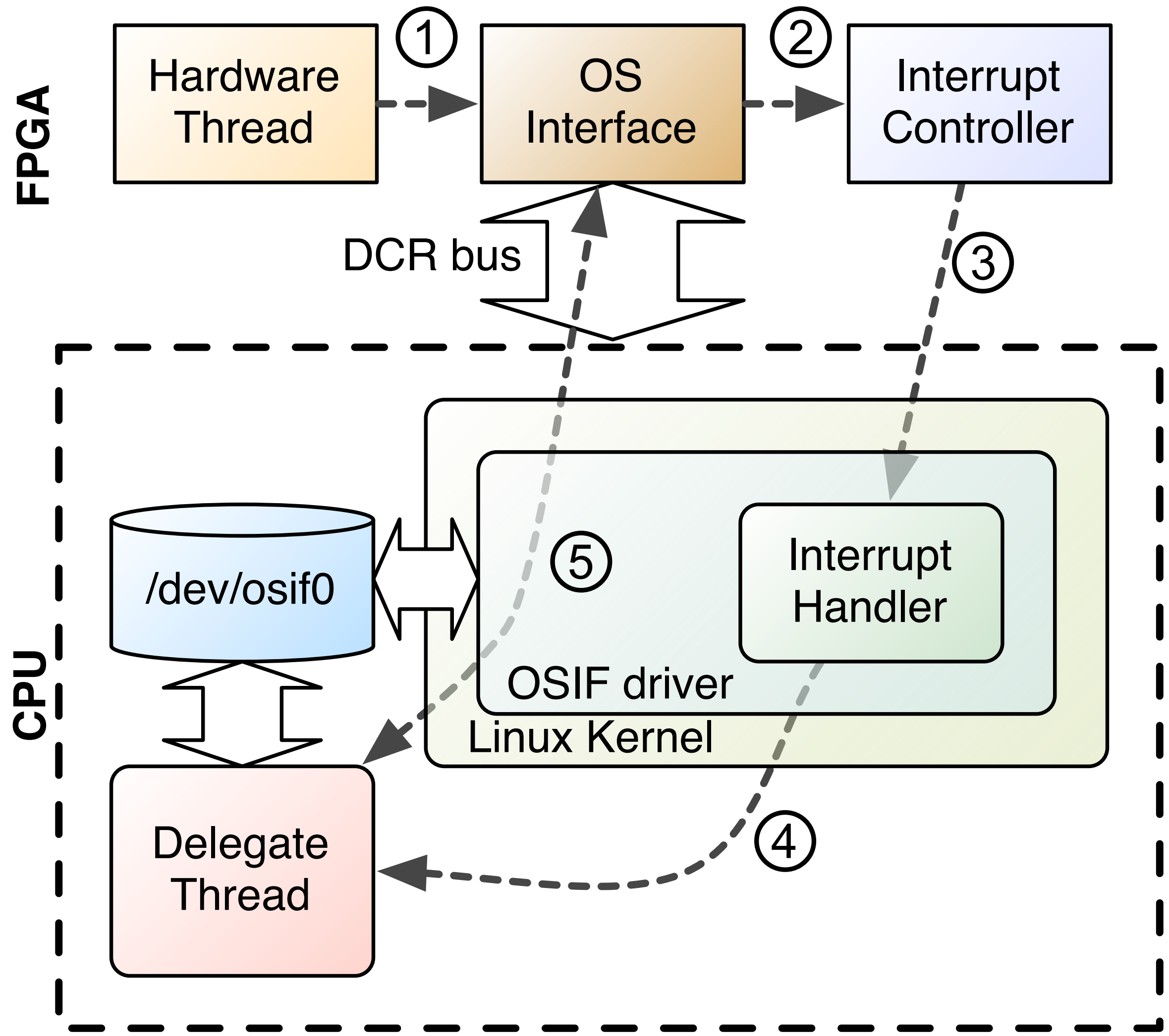
- motivation
- ReconOS abstraction layer
  - programming model
  - hardware architecture
  - hardware threads
  - OS interface & delegate threads
- host OS implementations
  - ReconOS/eCos
  - ReconOS/Linux
- experimental results
- conclusion

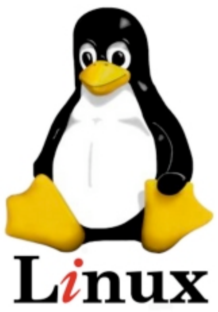
OS Call Sequence (eCos)



- eCos
  - configurable, small-footprint operating system for embedded domain
  - all code executes in kernel mode; simple hardware access possible
- OS call sequence
  - hardware thread initiates request; OS interface raises interrupt
  - delegate is synchronized to interrupts through semaphores
  - delegate thread is woken up and retrieves OS call and parameters

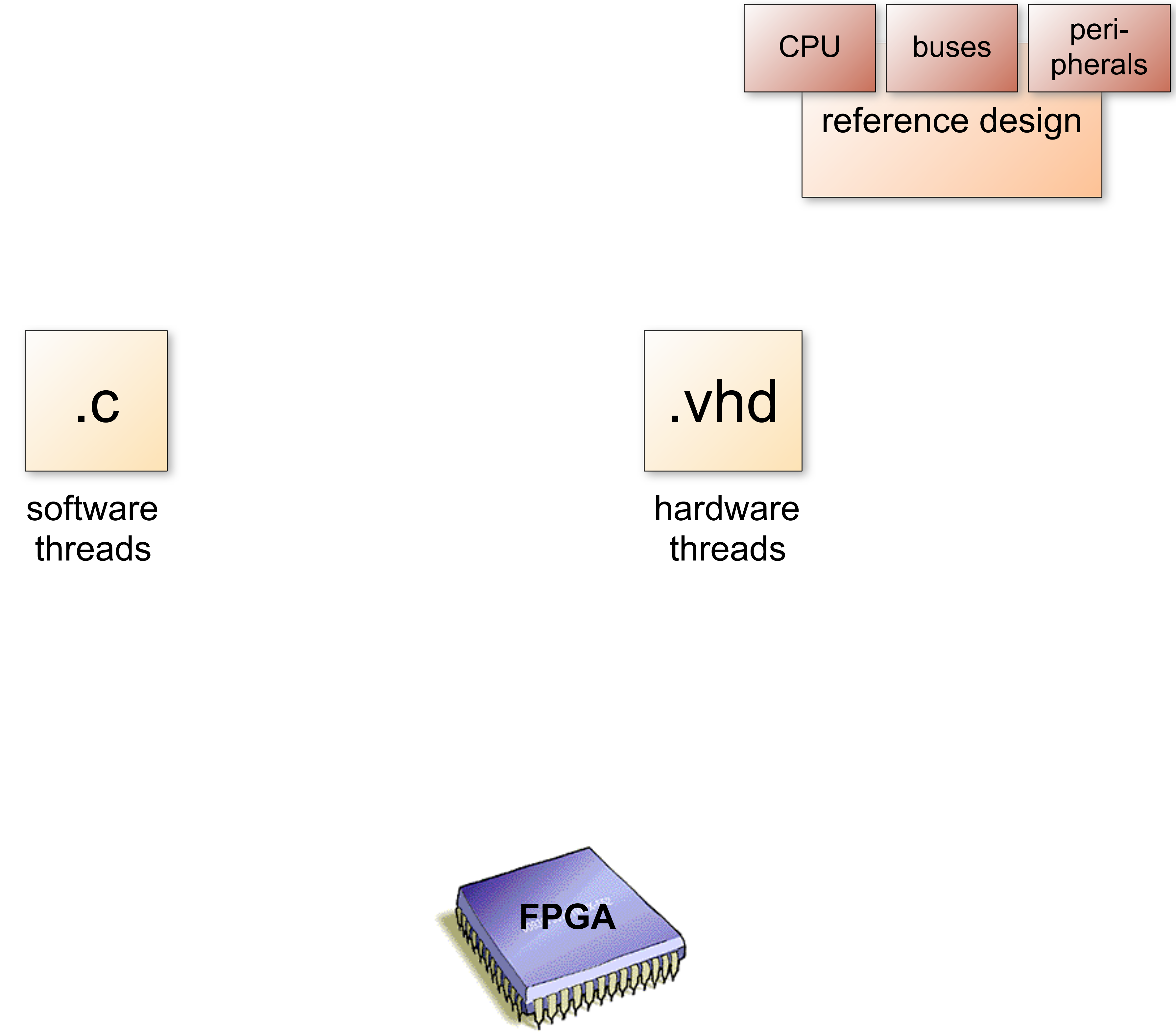
OS Call Sequence (Linux)



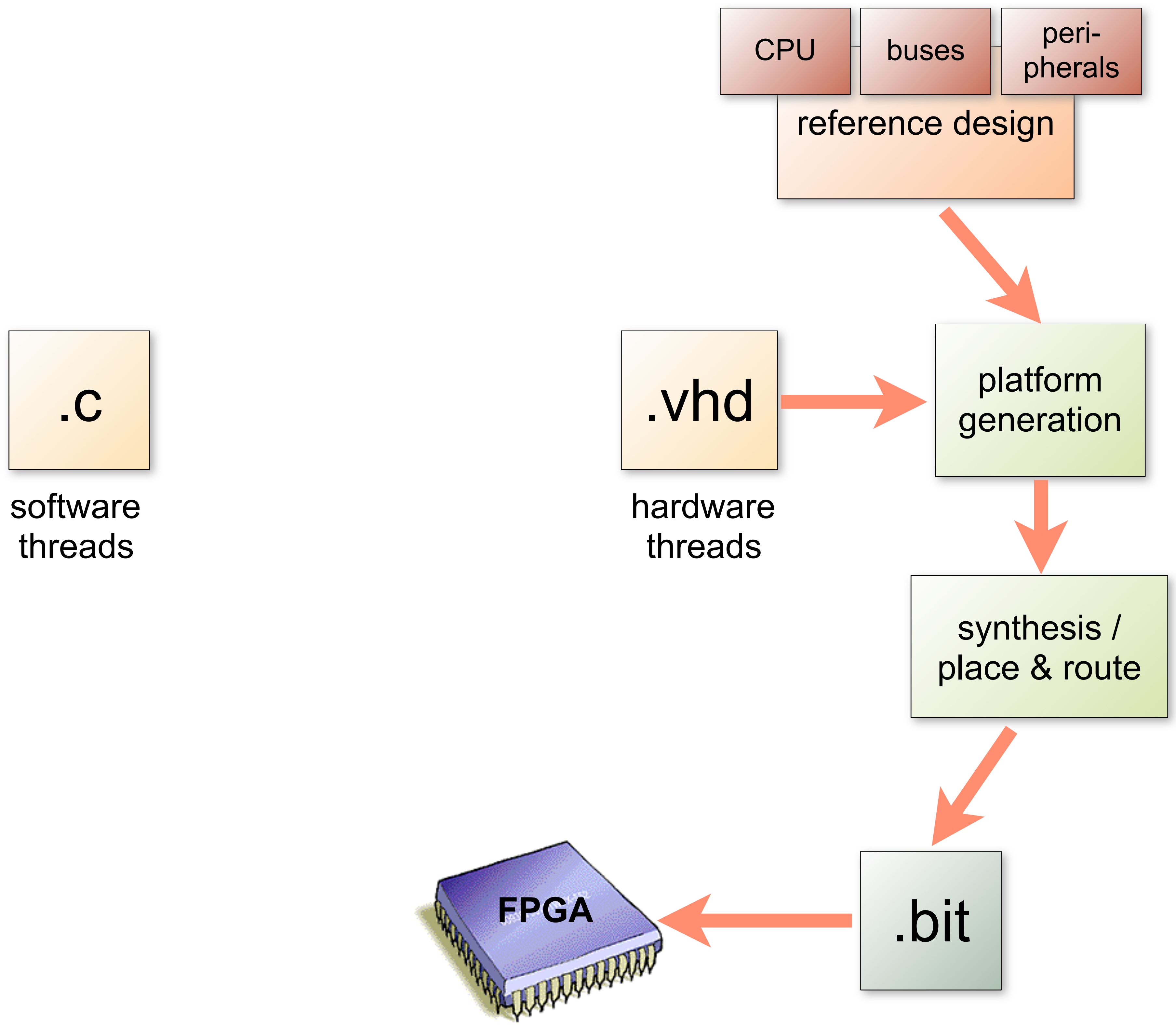


- Linux
  - flexible and widely used OS for embedded and HPC domain
  - no direct hardware access possible from Linux user space; needs driver
- OS call sequence
  - hardware thread initiates request; OS interface raises interrupt
  - delegate is synchronized to interrupts through blocking filesystem accesses
  - delegate thread is woken up and retrieves OS call and parameters

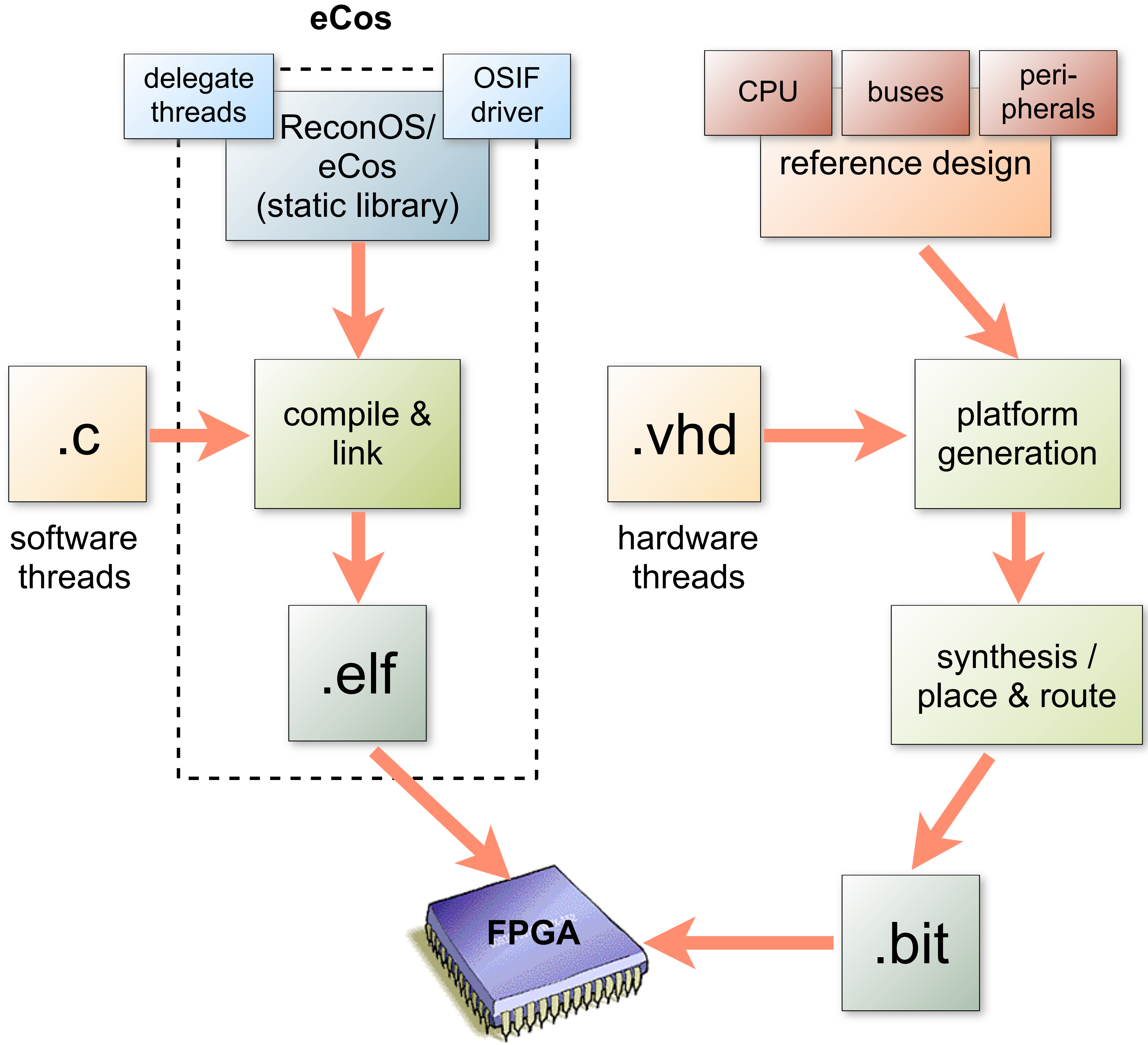
Tool Flow


CPU
buses
peri-
pherals
reference design
.c
software
threads
.vhd
hardware
threads
FPGA

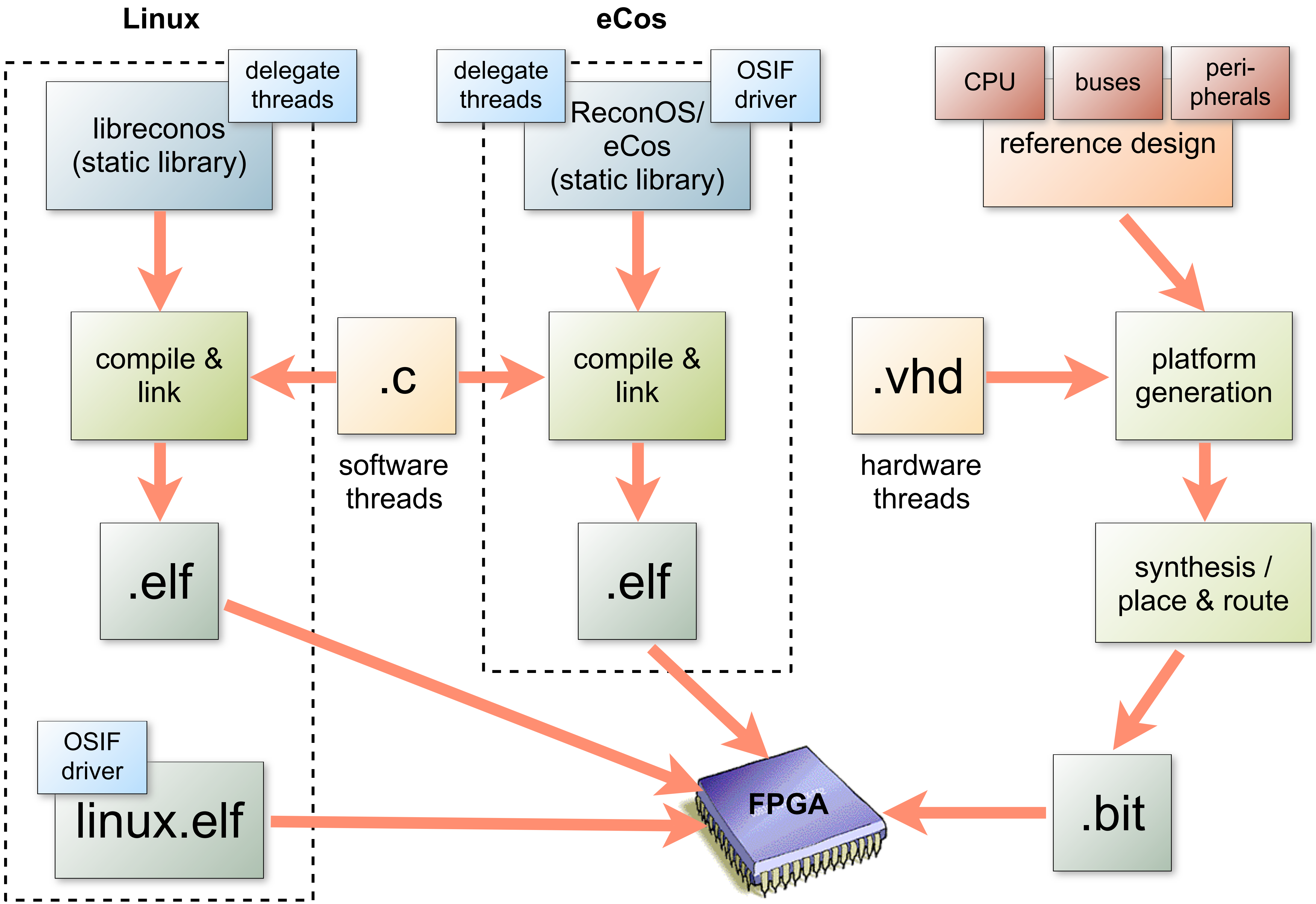
Tool Flow


CPU
buses
peri-
pherals
reference design
.c
software
threads
.vhd
hardware
threads
platform
generation
synthesis /
place & route
.bit
FPGA

Tool Flow


eCos
delegate threads
OSIF driver
ReconOS/ eCos (static library)
CPU
buses
peri- pherals
reference design
.c
software threads
compile & link
.elf
.vhd
hardware threads
platform generation
synthesis / place & route
.bit
FPGA

Tool Flow



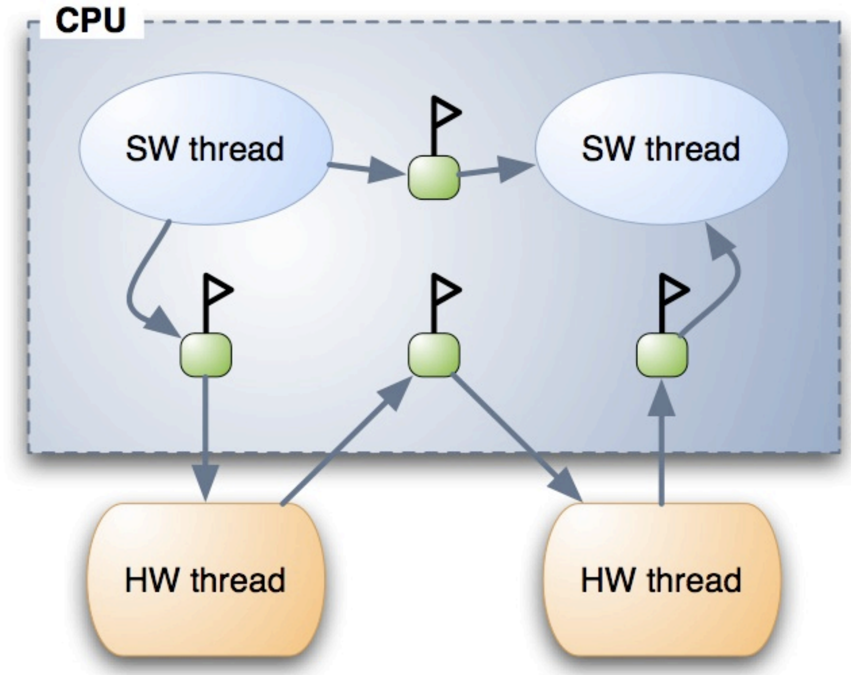
Linux
eCos
delegate threads
libreconos (static library)
delegate threads
ReconOS/ eCos (static library)
OSIF driver
CPU
buses
peri- pherals
reference design
compile & link
.c
software threads
compile & link
.vhd
hardware threads
platform generation
.elf
.elf
synthesis / place & route
OSIF driver
linux.elf
FPGA
.bit

Overview

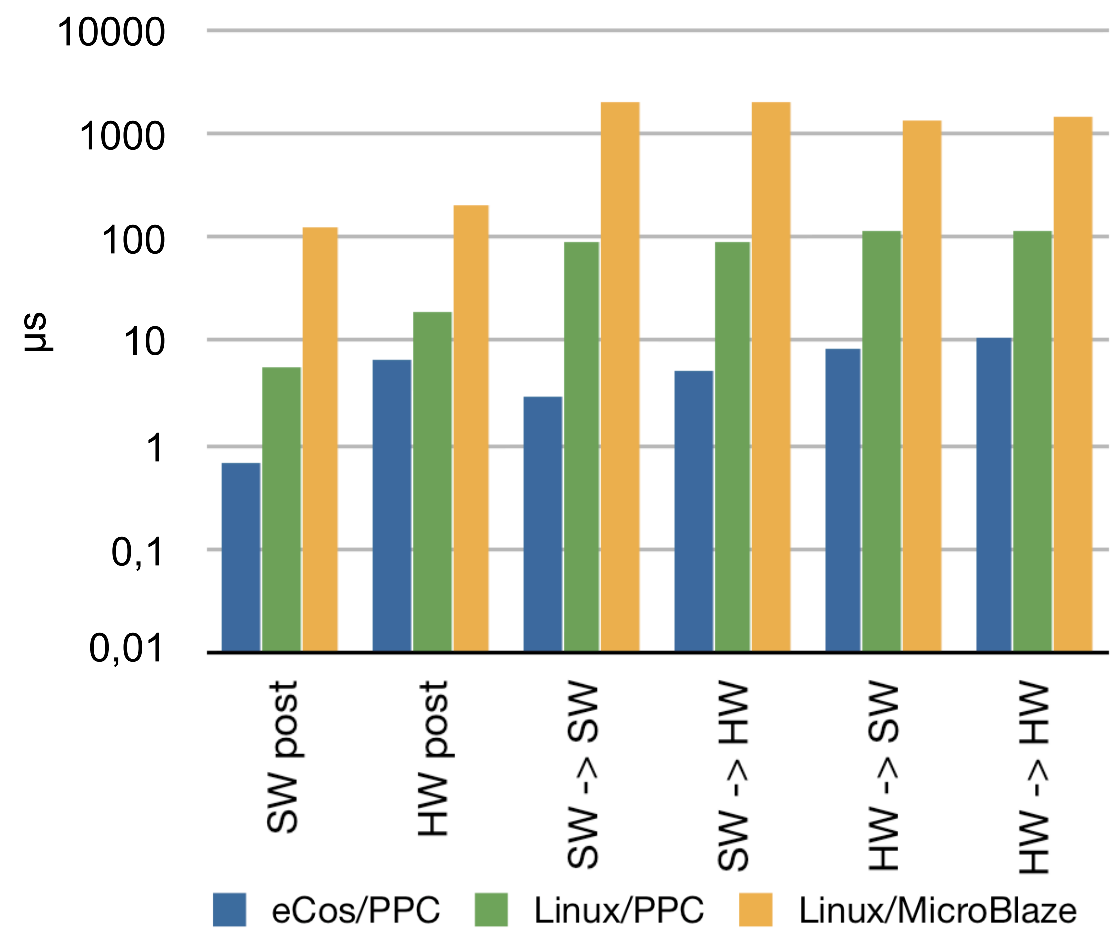
- motivation
- ReconOS abstraction layer
  - programming model
  - hardware architecture
  - hardware threads
  - OS interface & delegate threads
- host OS implementations
  - ReconOS/eCos
  - ReconOS/Linux
- experimental results
- conclusion

OS Call Overheads

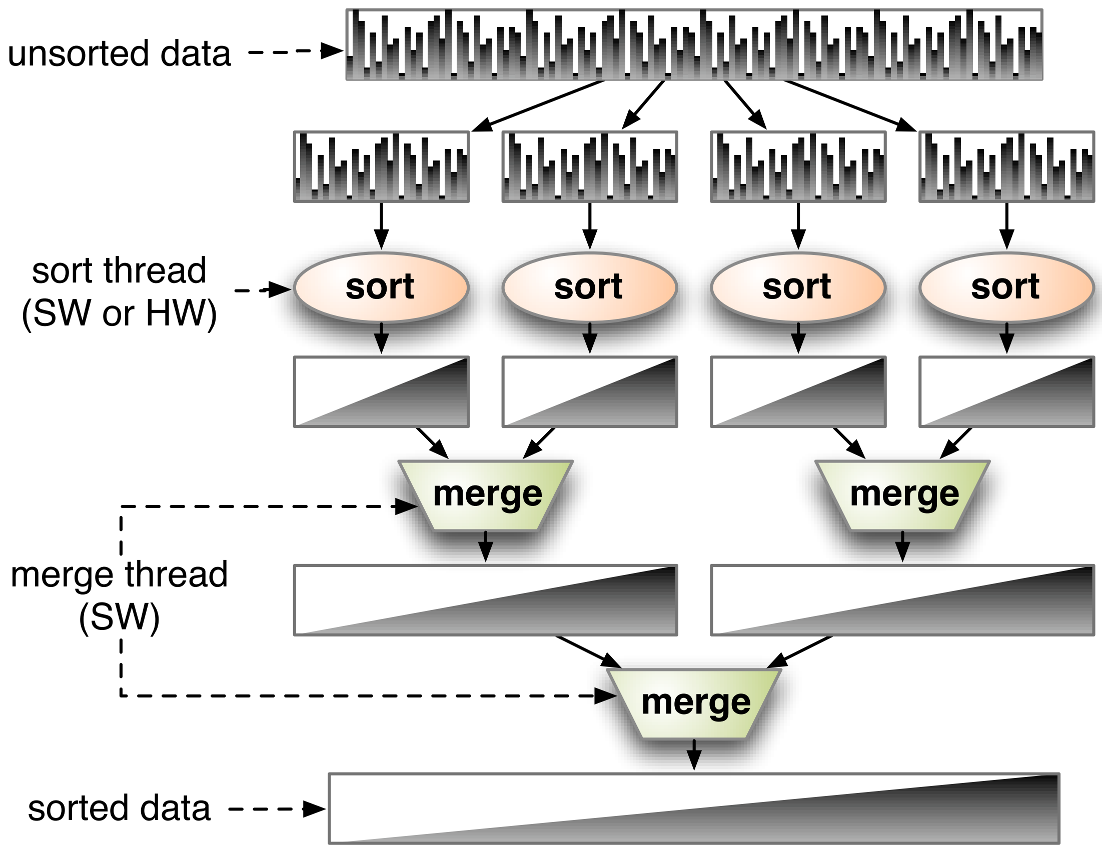
- synthetic hardware and software threads
  - semaphore and mutex processing time (post → wait / unlock → lock)
- executed on three prototypes
  - eCos/PPC
    - XC2VP30
    - PowerPC405 @300MHz
    - HW threads & bus @100MHz
  - Linux/PPC
    - XC2VP30
    - PowerPC405 @300MHz
    - HW threads & bus @100MHz
  - Linux/MicroBlaze
    - XC4VSX35
    - MicroBlaze 4.0 @100Mhz
    - HW threads & bus @100MHz

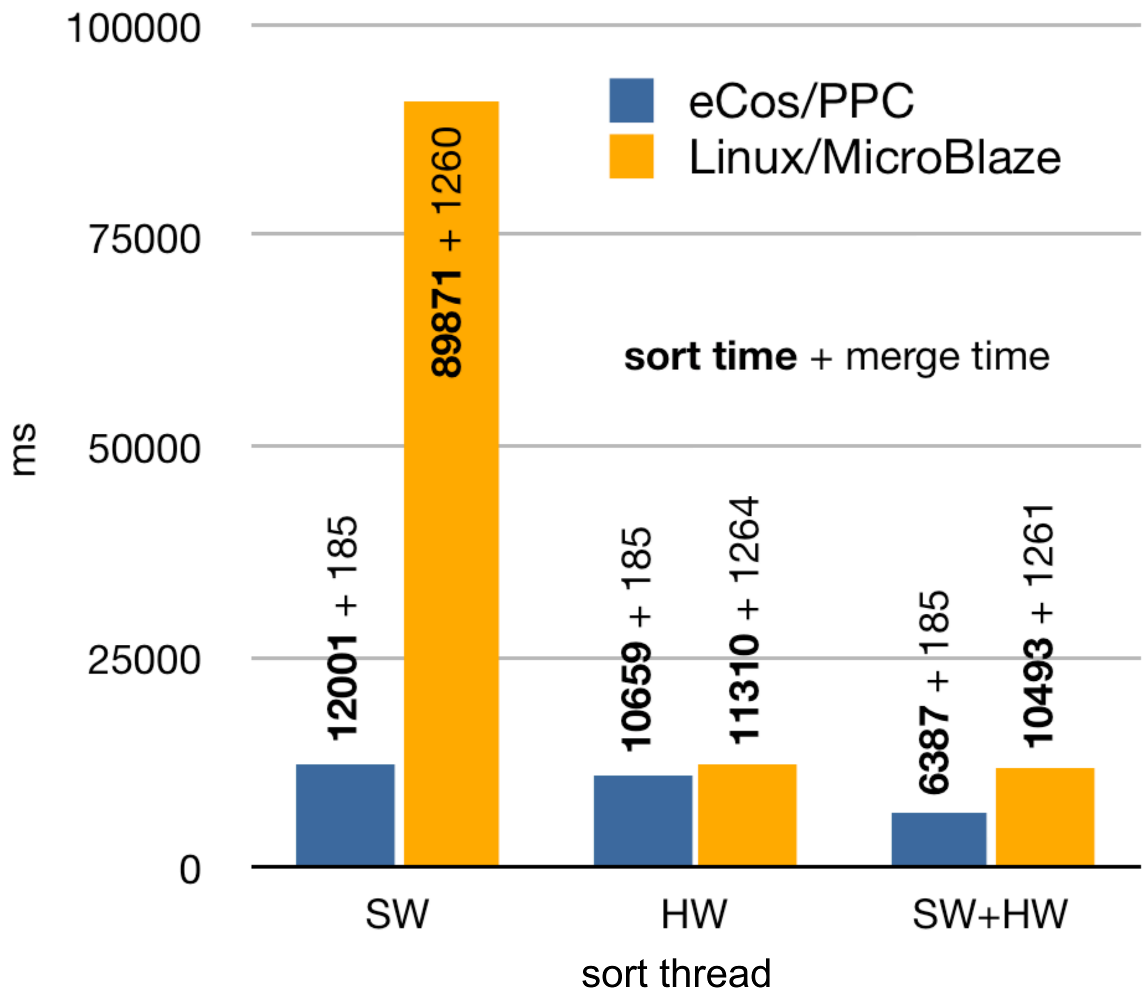


Semaphore Operations



Application Case Study





- sort application
  - sorts an array of integers (1MB) using a combination of bubble sort and merge sort
  - sort thread can be executed either in hardware or software

➡ OS call overhead not a major factor in overall performance

Conclusion & Outlook

- we extended the established multithreaded programming model to reconfigurable hardware
- unified set of abstractions for hard- and software threads provides portability across different host OS‘s and CPU/FPGA architectures
- the additional abstraction layer shows acceptable performance in benchmarks and larger case studies

- future work
  - implementation on FPGA accelerators for high-performance computing
  - extension of OS scheduler to allow hardware thread scheduling using partial reconfiguration

Overview

- motivation
- ReconOS abstraction layer
  - programming model
  - hardware architecture
  - hardware threads
  - OS interface & delegate threads
- host OS implementations
  - ReconOS/eCos
  - ReconOS/Linux
- experimental results
- conclusion

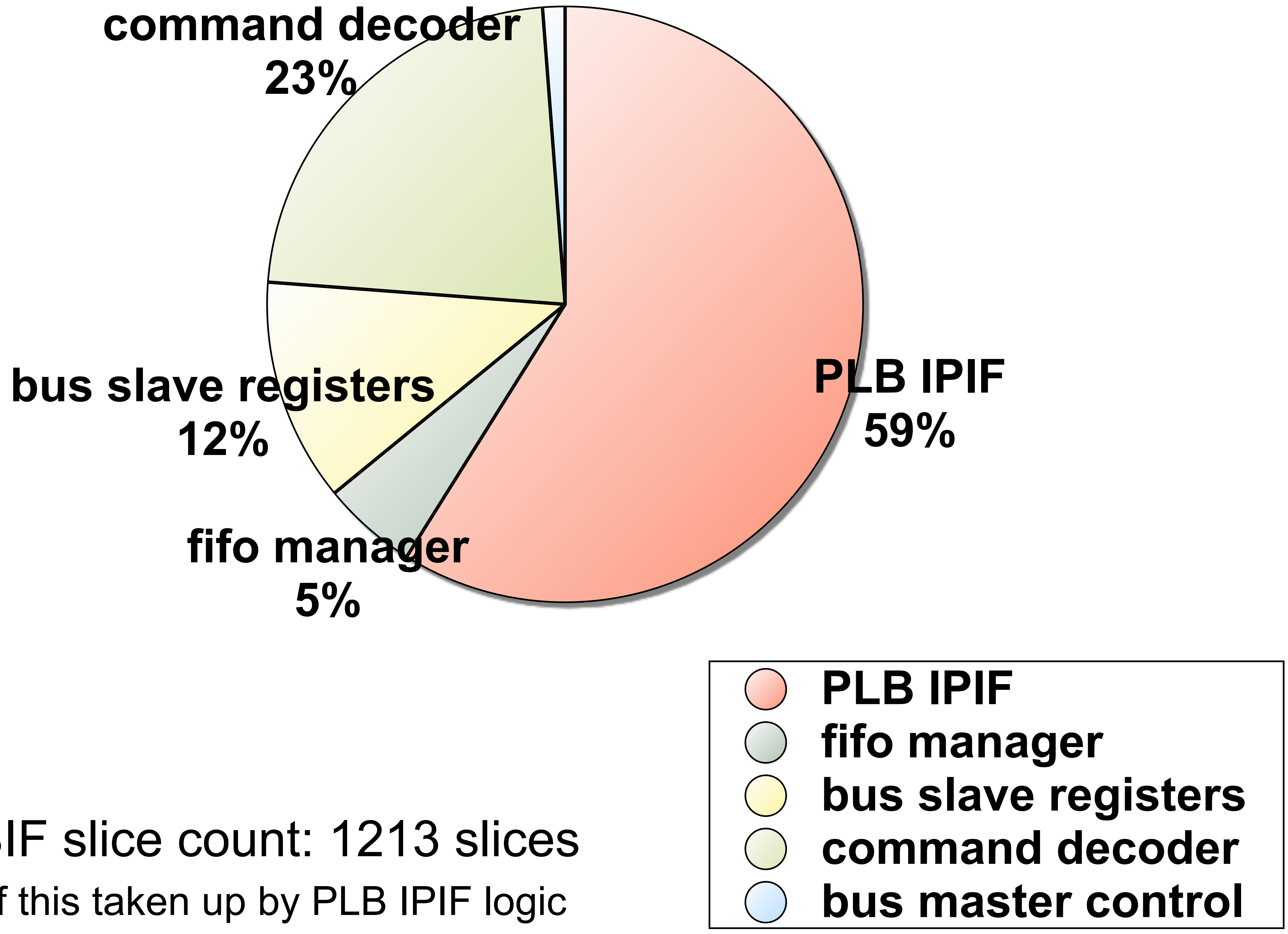
Thank you

www.reconos.de

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

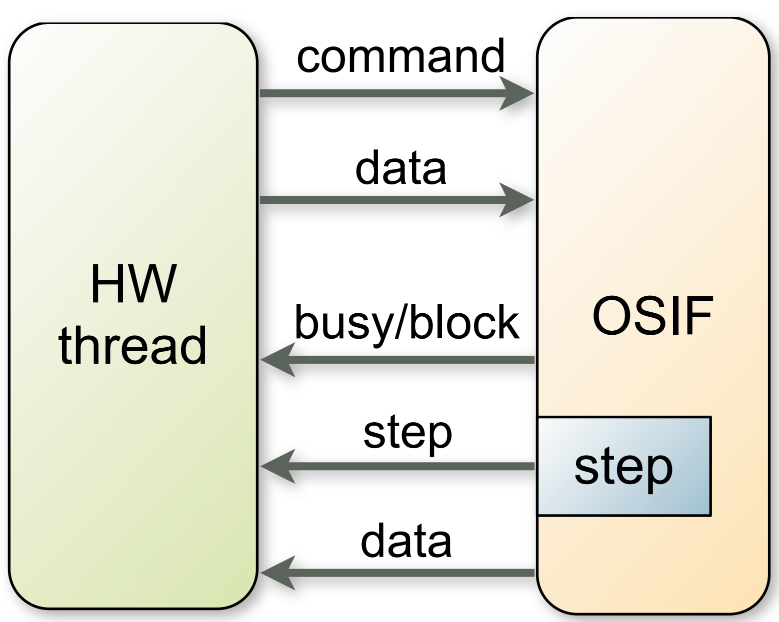
```
mqd_t my_mbox;
sem_t my_sem;

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
);
```

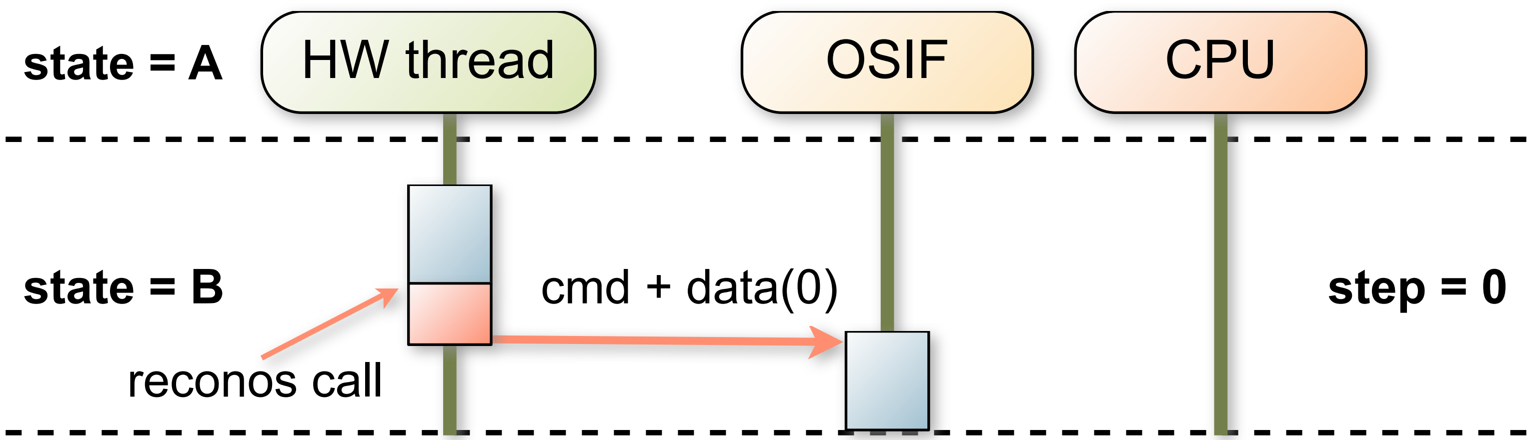
Multi-Cycle Commands

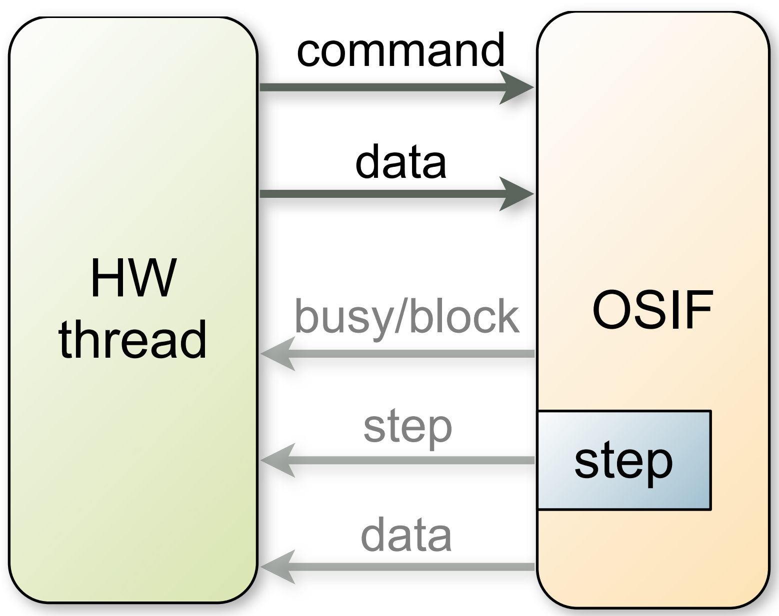
- transfer of multiple parameters and return values with a single VHDL call
- distributes execution of an FSM state across multiple clock cycles



Multi-Cycle Commands

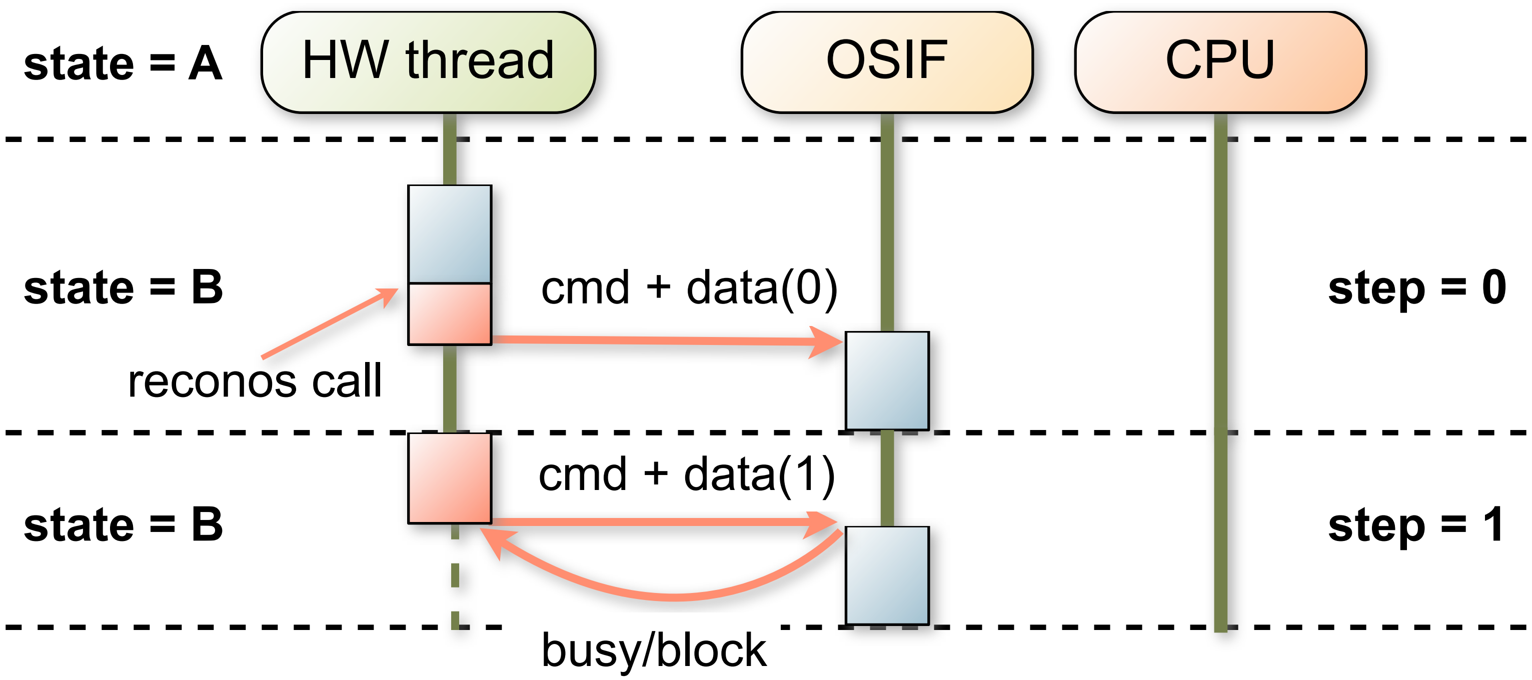
- transfer of multiple parameters and return values with a single VHDL call
- distributes execution of an FSM state across multiple clock cycles

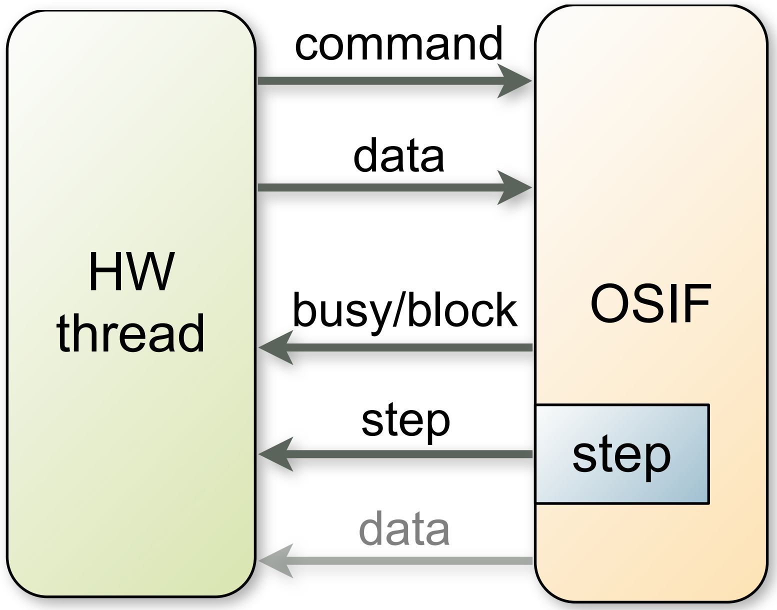




Multi-Cycle Commands

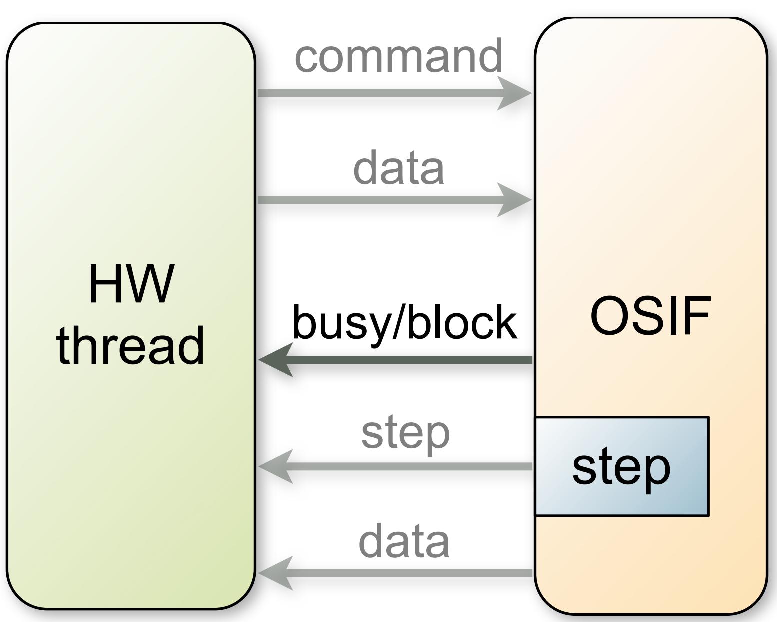
- transfer of multiple parameters and return values with a single VHDL call
- distributes execution of an FSM state across multiple clock cycles

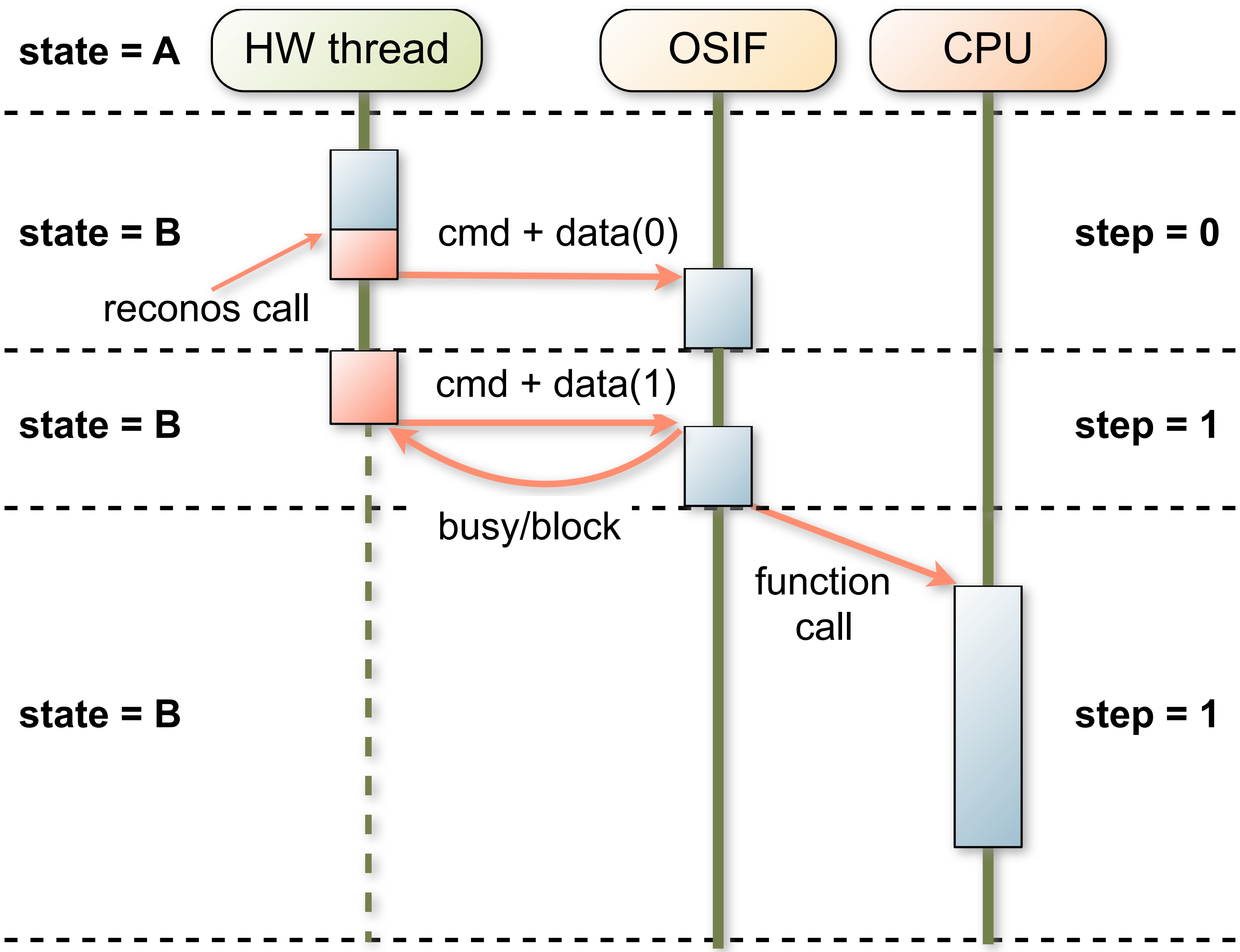




Multi-Cycle Commands

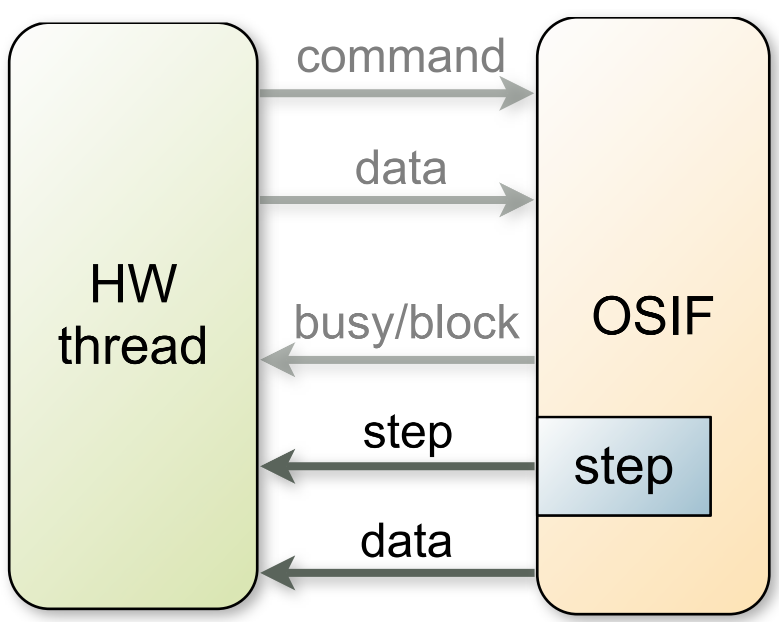
- transfer of multiple parameters and return values with a single VHDL call
- distributes execution of an FSM state across multiple clock cycles

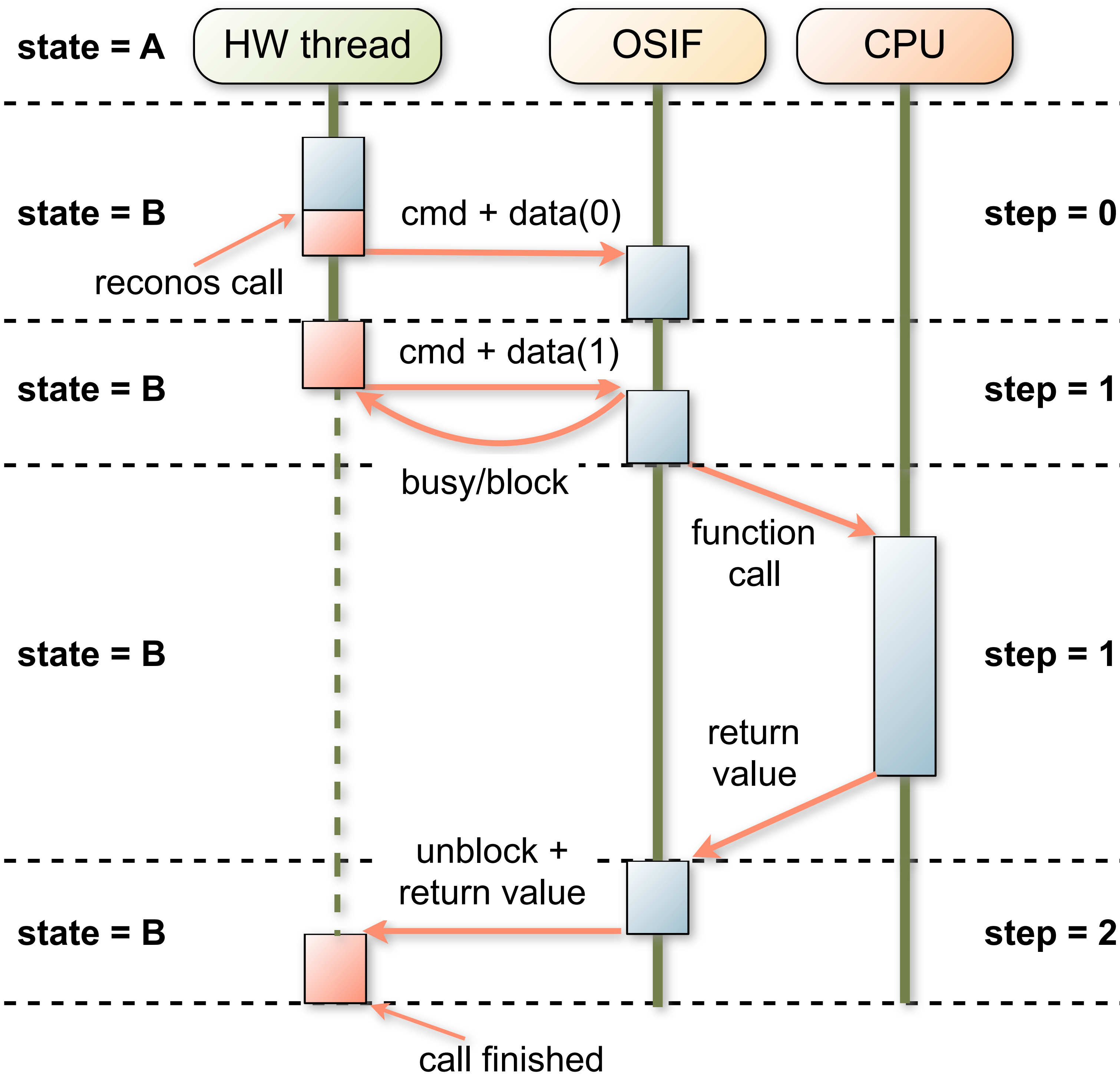




Multi-Cycle Commands

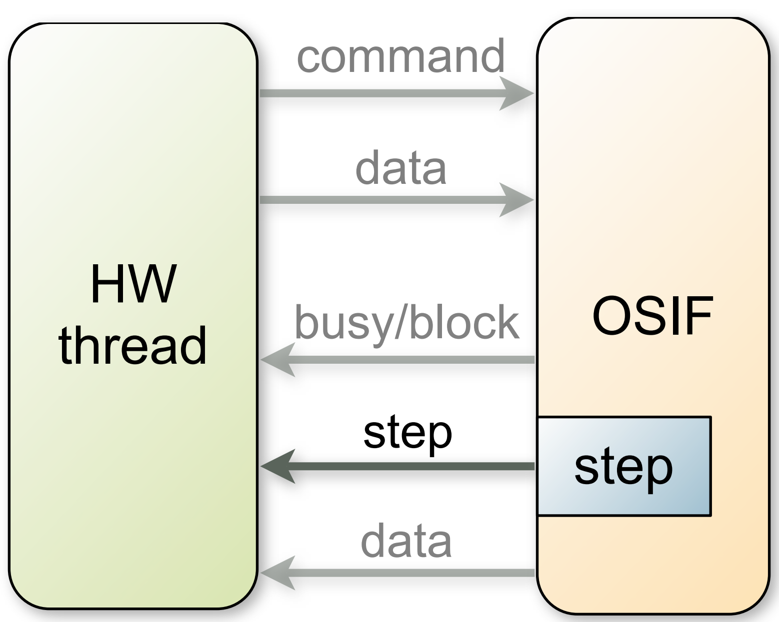
- transfer of multiple parameters and return values with a single VHDL call
- distributes execution of an FSM state across multiple clock cycles

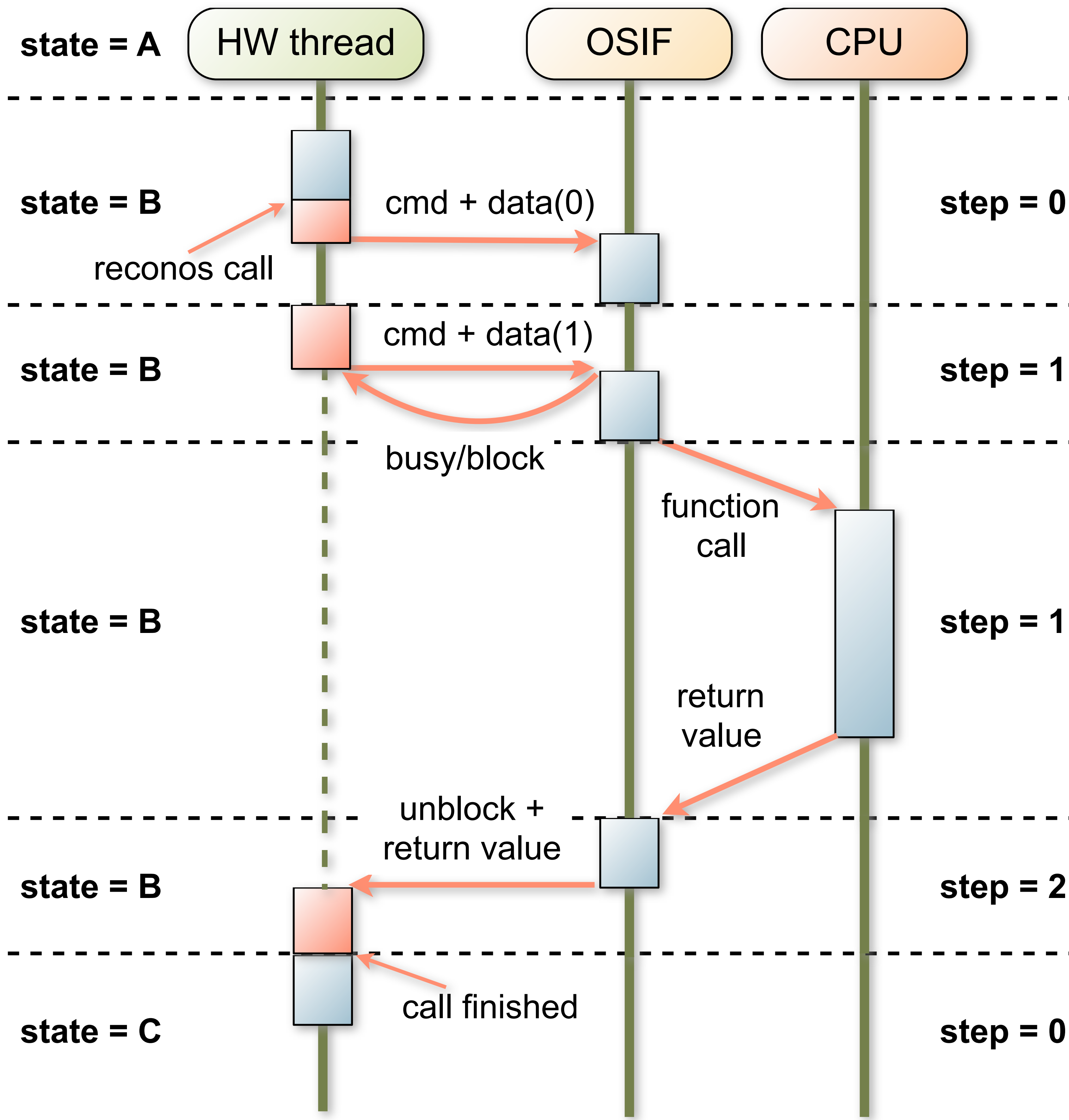




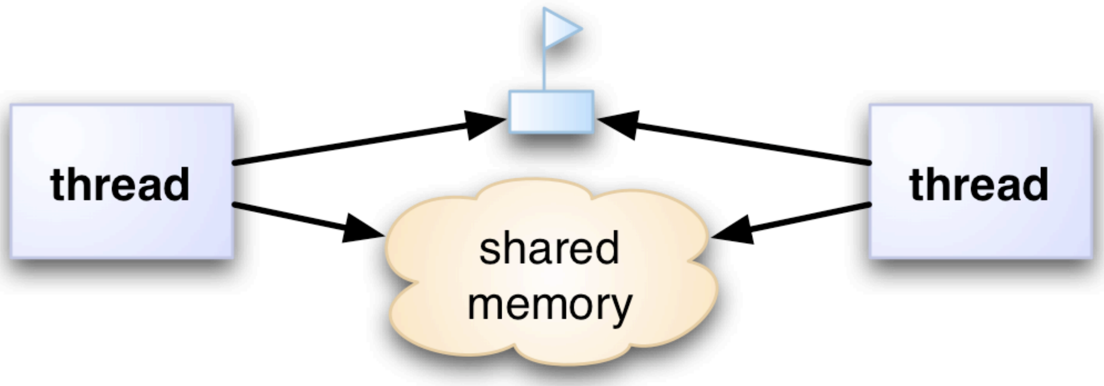
Multi-Cycle Commands

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





Toolchain

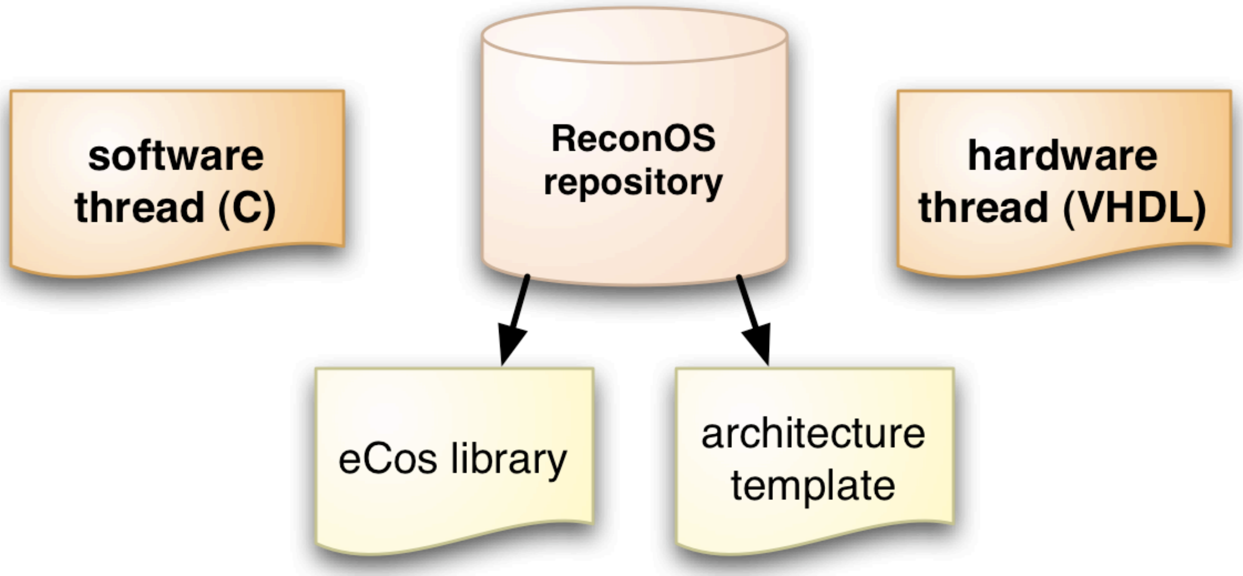


Toolchain


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

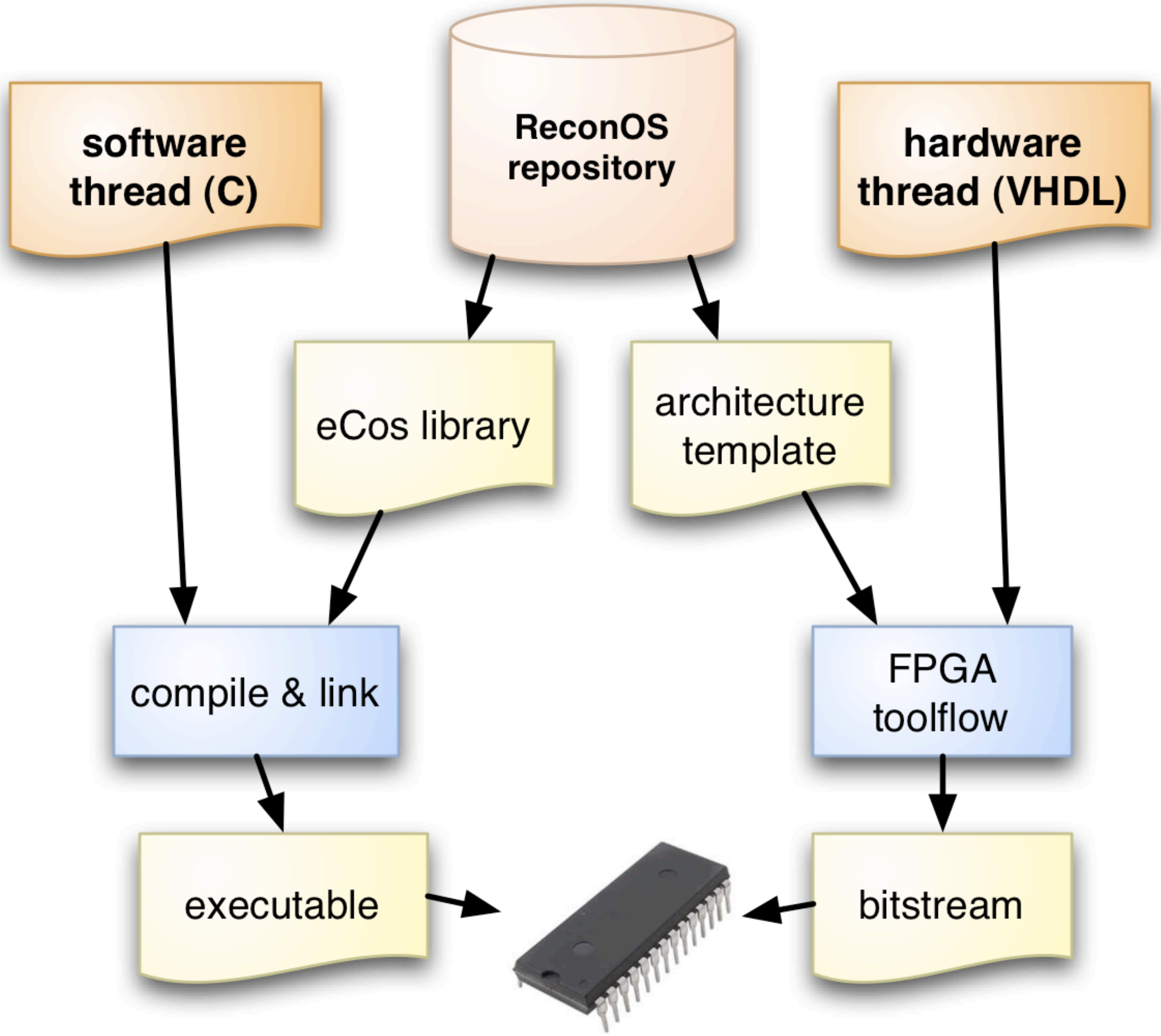
Toolchain

- 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



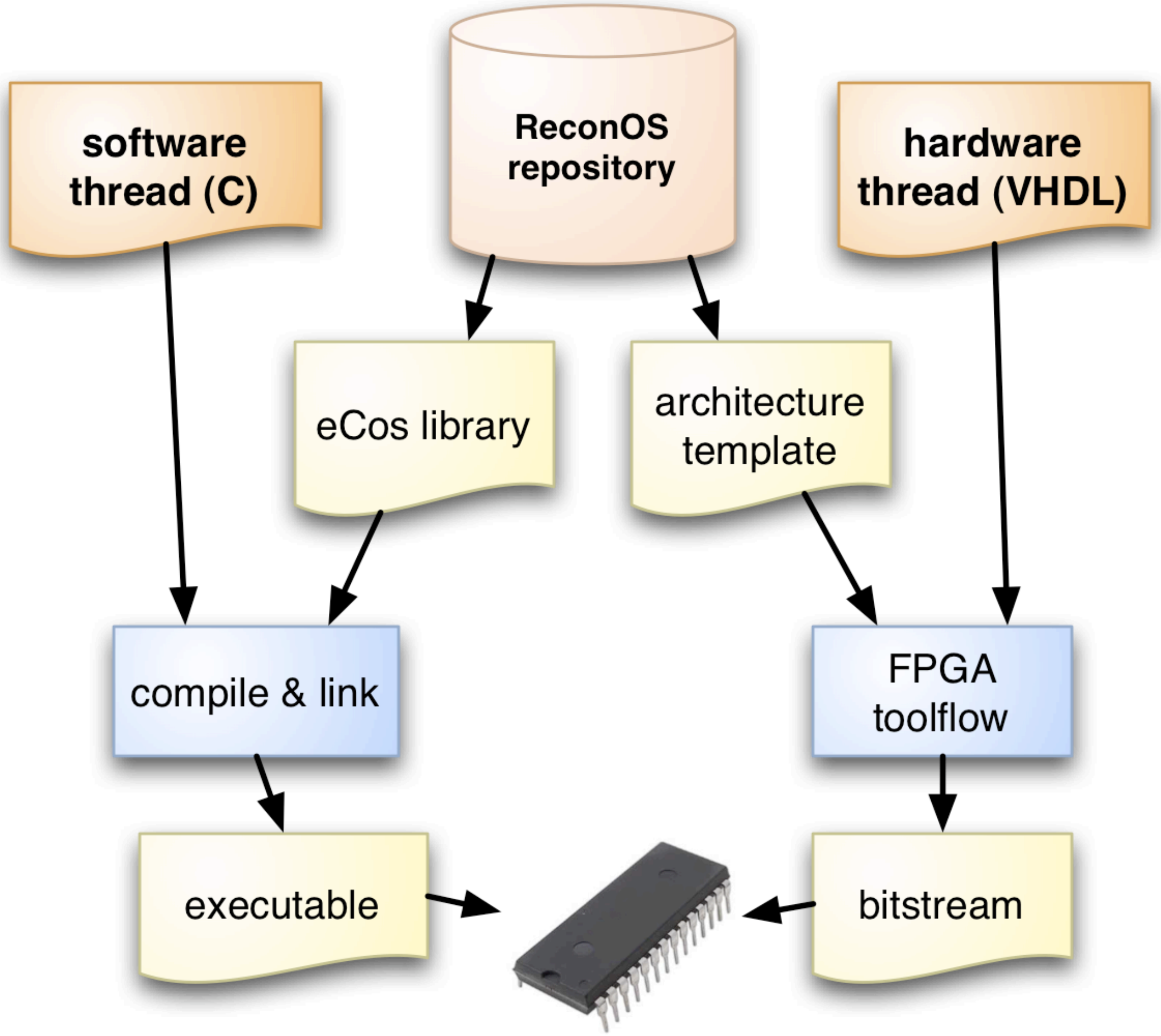
Toolchain

- 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

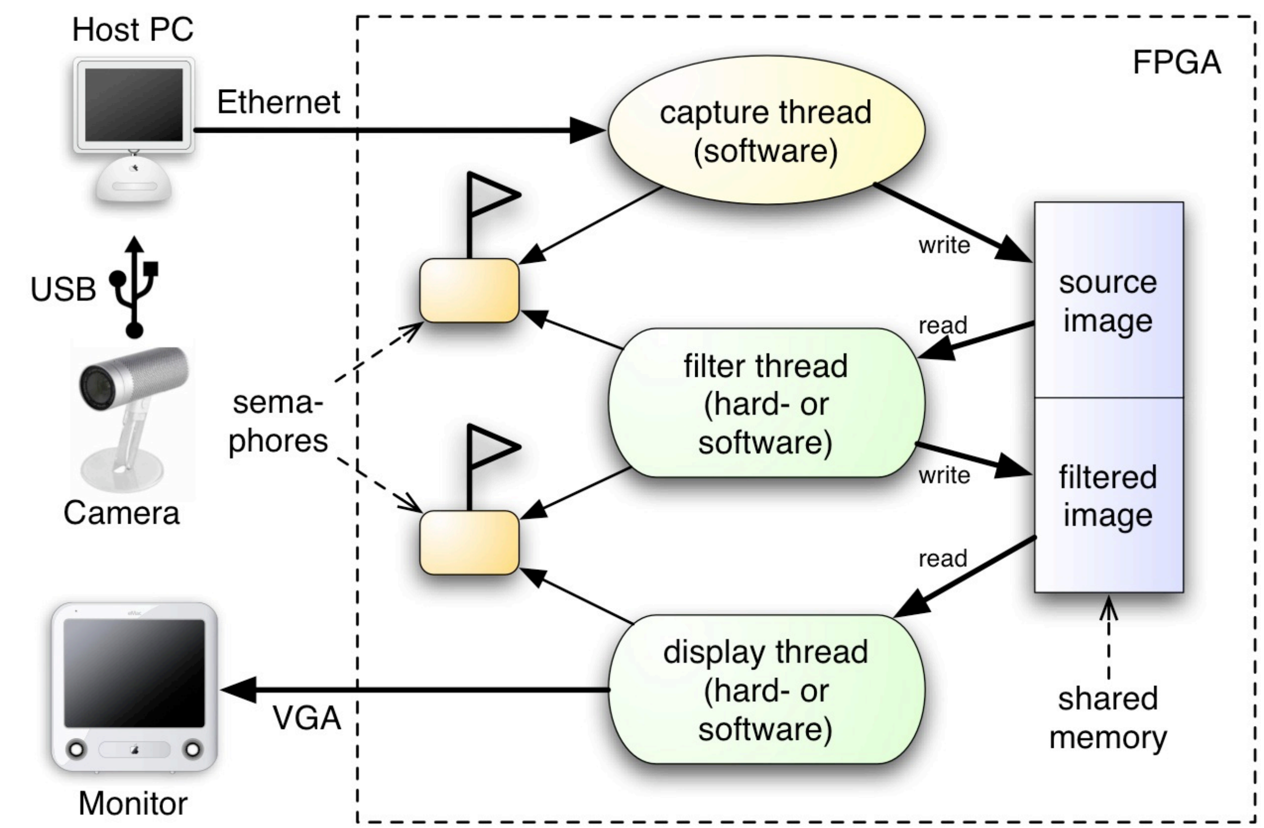
- software threads are written in C
  - using the eCos software API
- hardware threads are written in VHDL
  - using the ReconOS VHDL API
- architecture generation
  - automatically inserts OS interfaces and hardware threads into Xilinx EDK platform templates
  - configures and builds static eCos library

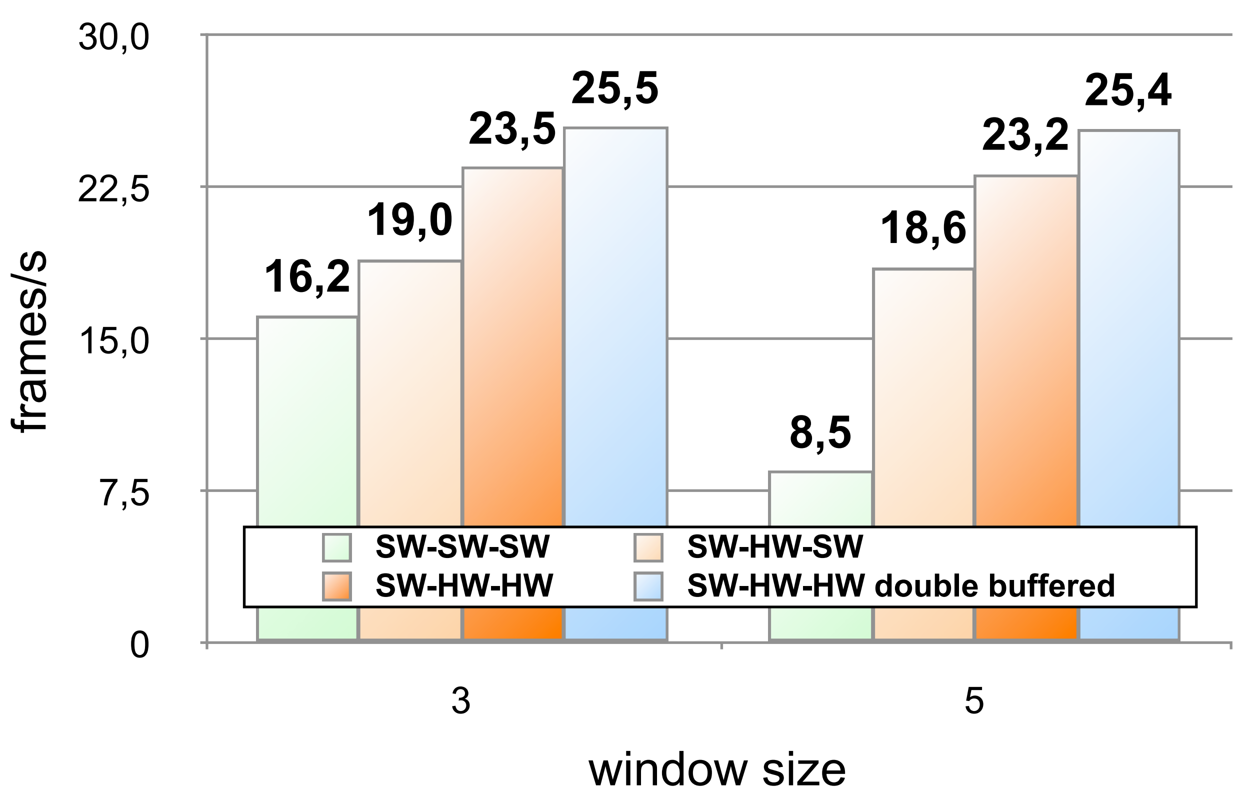


- eCos extensions
  - hardware thread object encapsulating delegate thread and OS interface “driver”
  - profiling support to track the state of the hardware threads' OS synchronization state machines

Case Study - Image Processing Filter

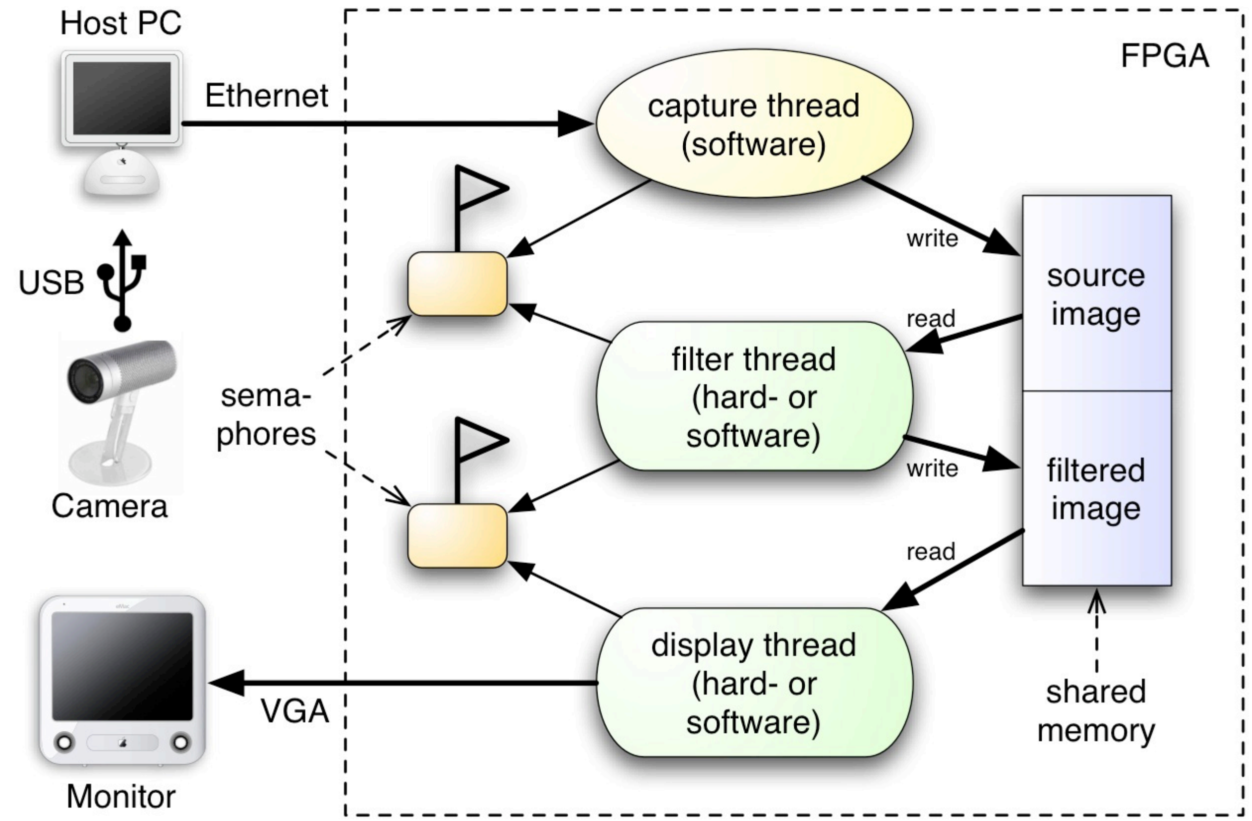
- three threads
  - capture image from Ethernet
  - apply LaPlacian filter
  - display image on VGA monitor
- threads communicate through shared memory
  - image resolution: 320x240 pixels, 8 bit greyscale
  - image data organized into blocks (e.g. 40 lines = 1 block)
  - a block is protected by two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





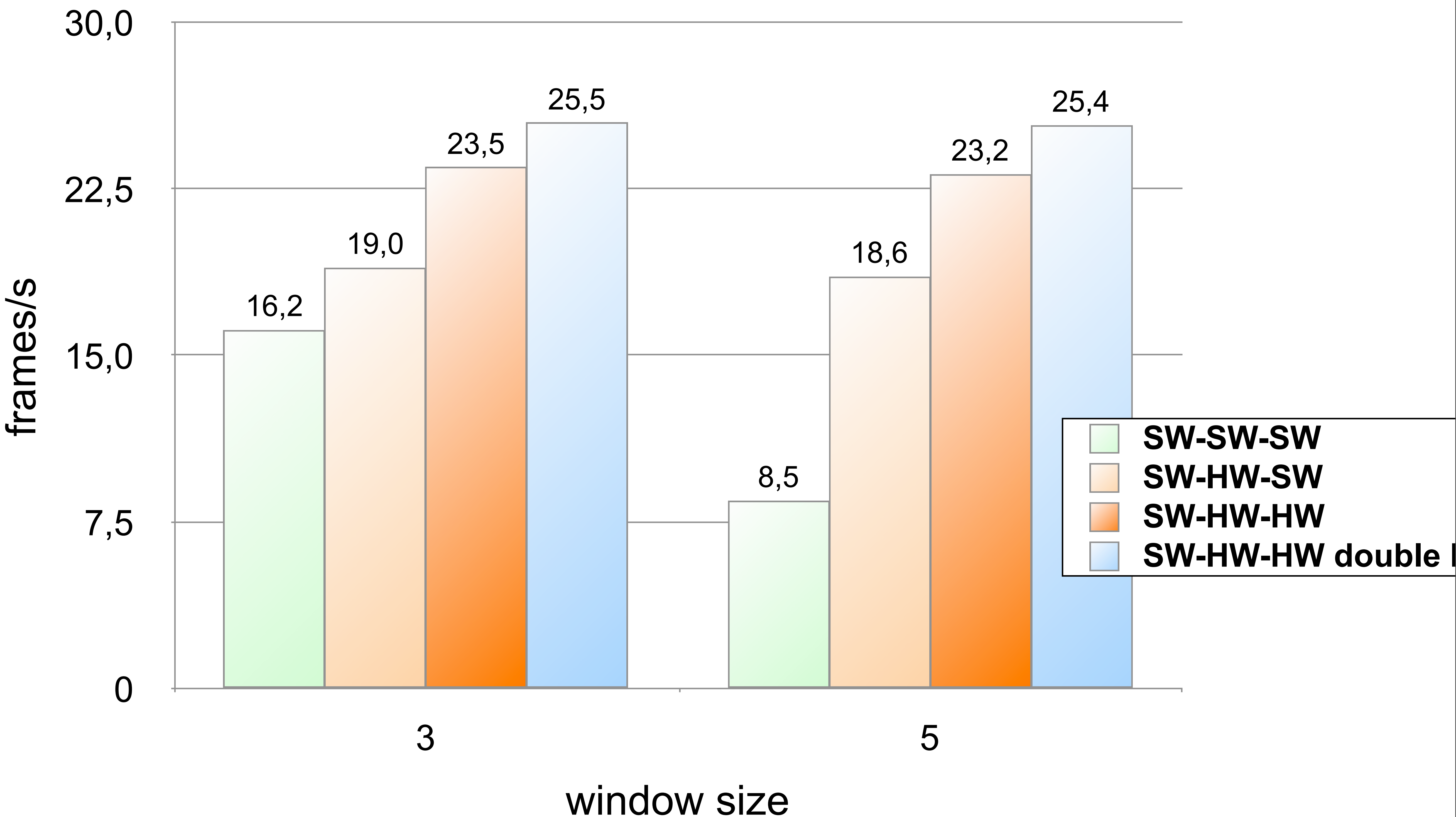
Case Study - Image Processing Filter

- 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



SW-SW-SW | SW-HW-SW
SW-HW-HW | SW-HW-HW double buffered

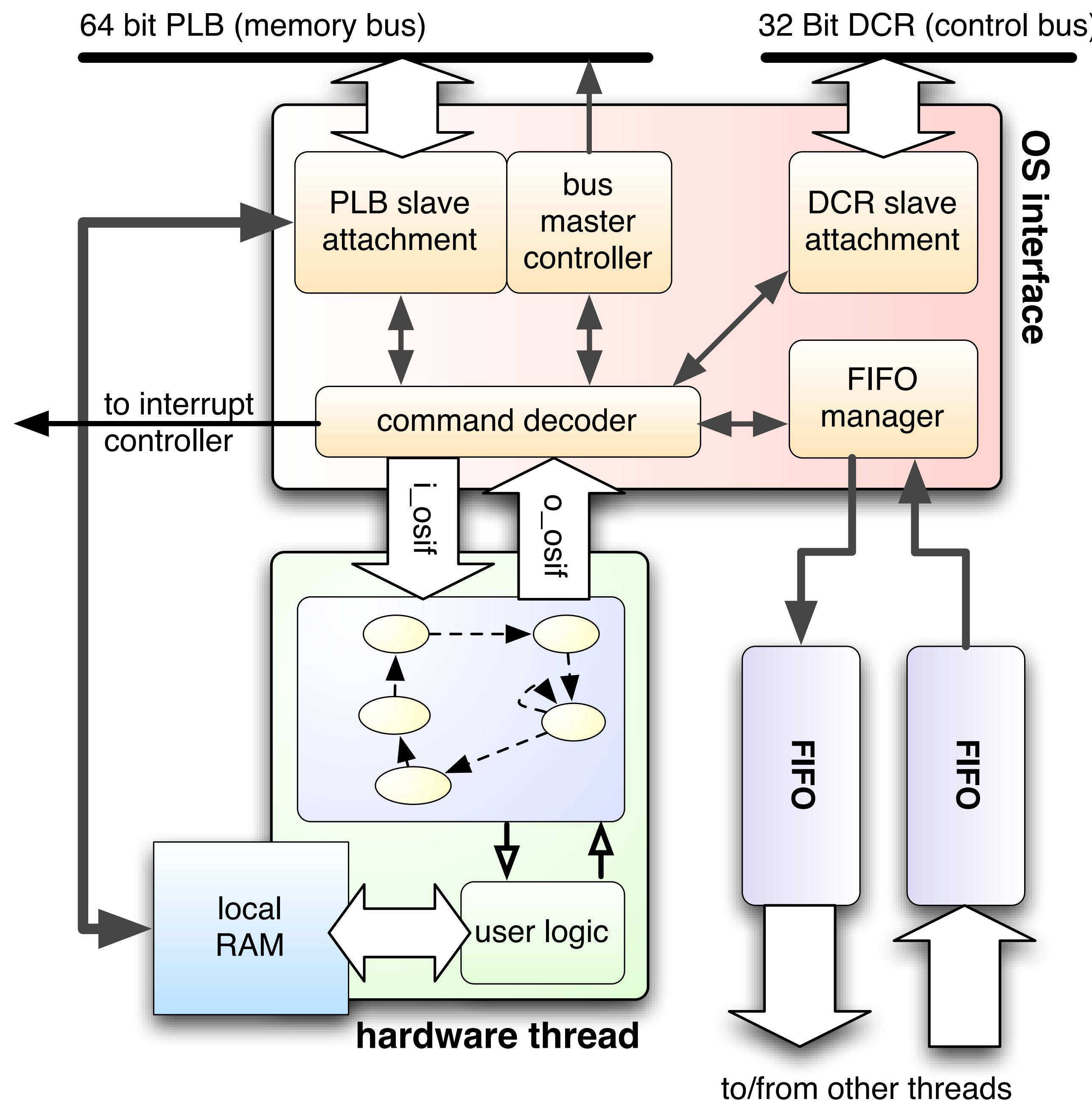
Case Study - Results



Case Study - Results

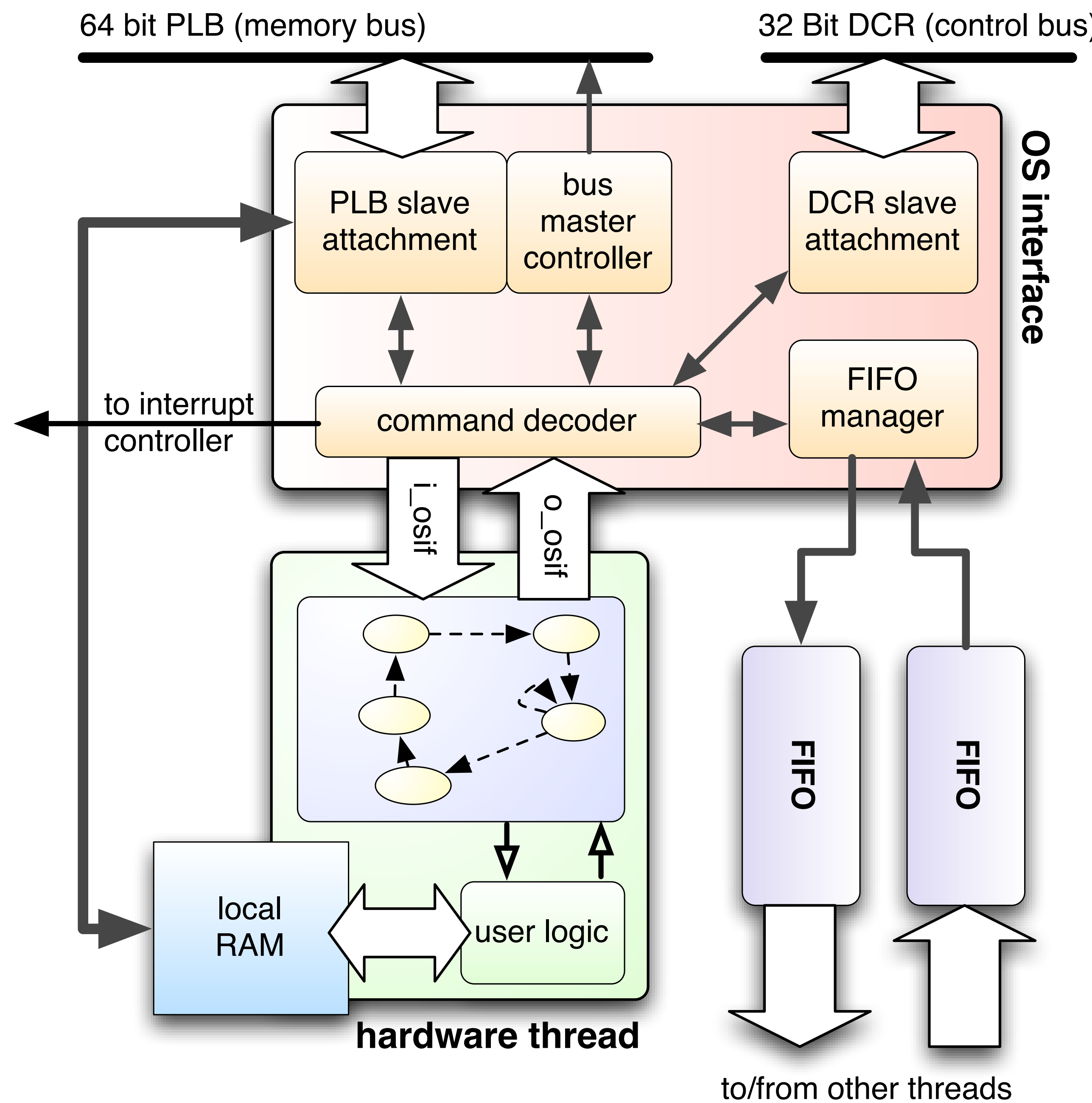
SW-SW-SW
SW-HW-SW
SW-HW-HW
SW-HW-HW double

OS Interface



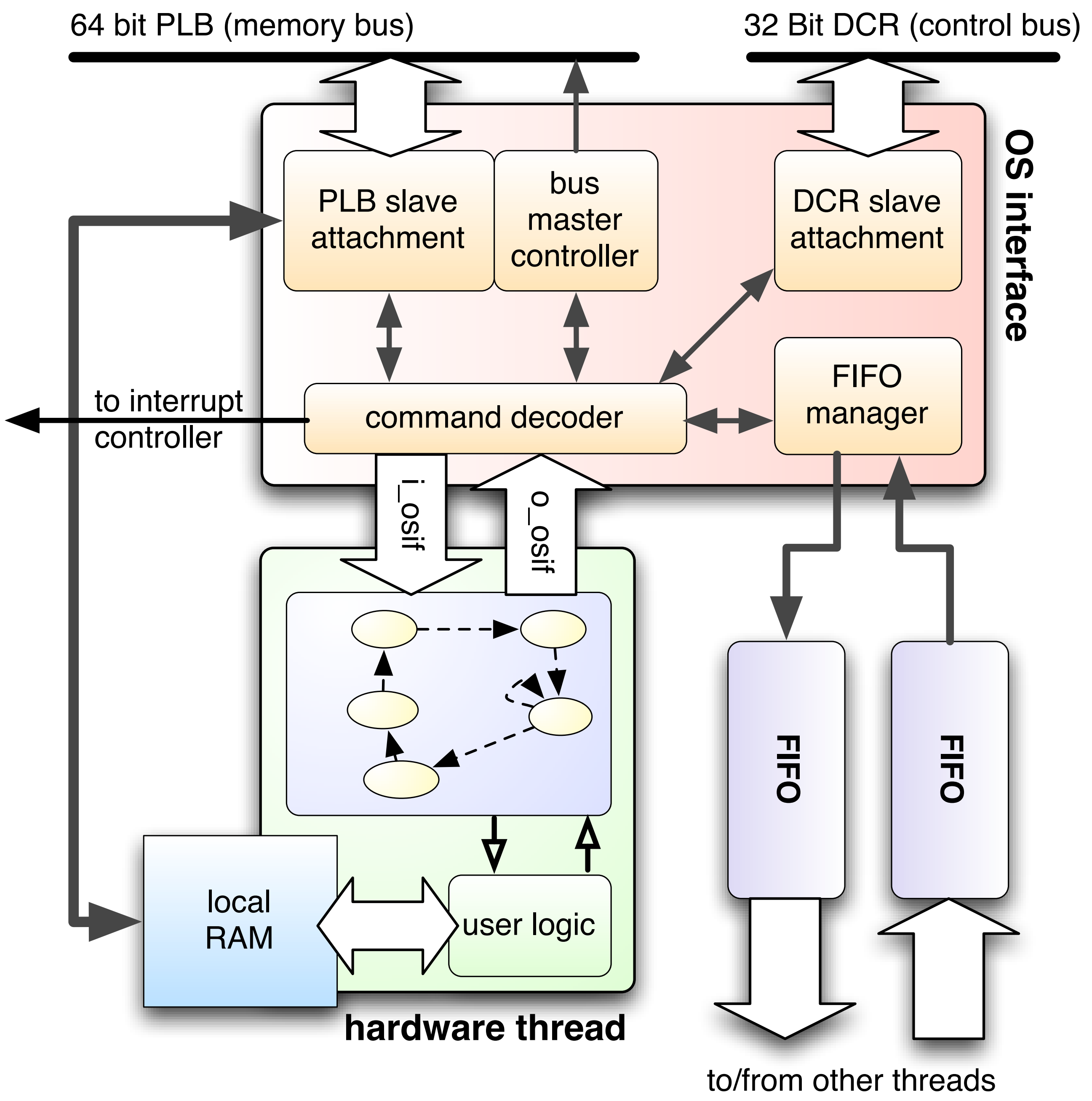
OS Interface

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



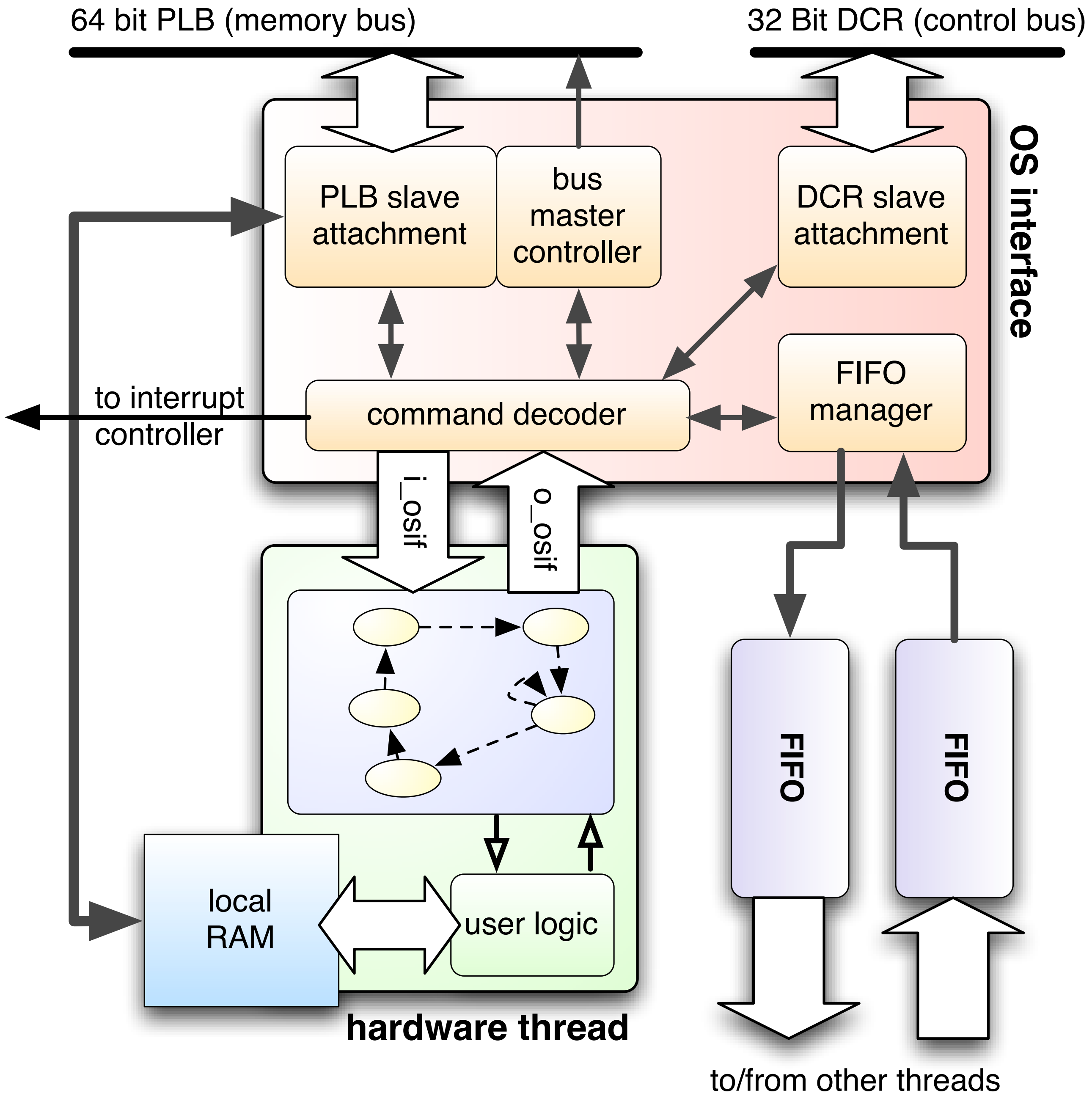
OS Interface

- 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



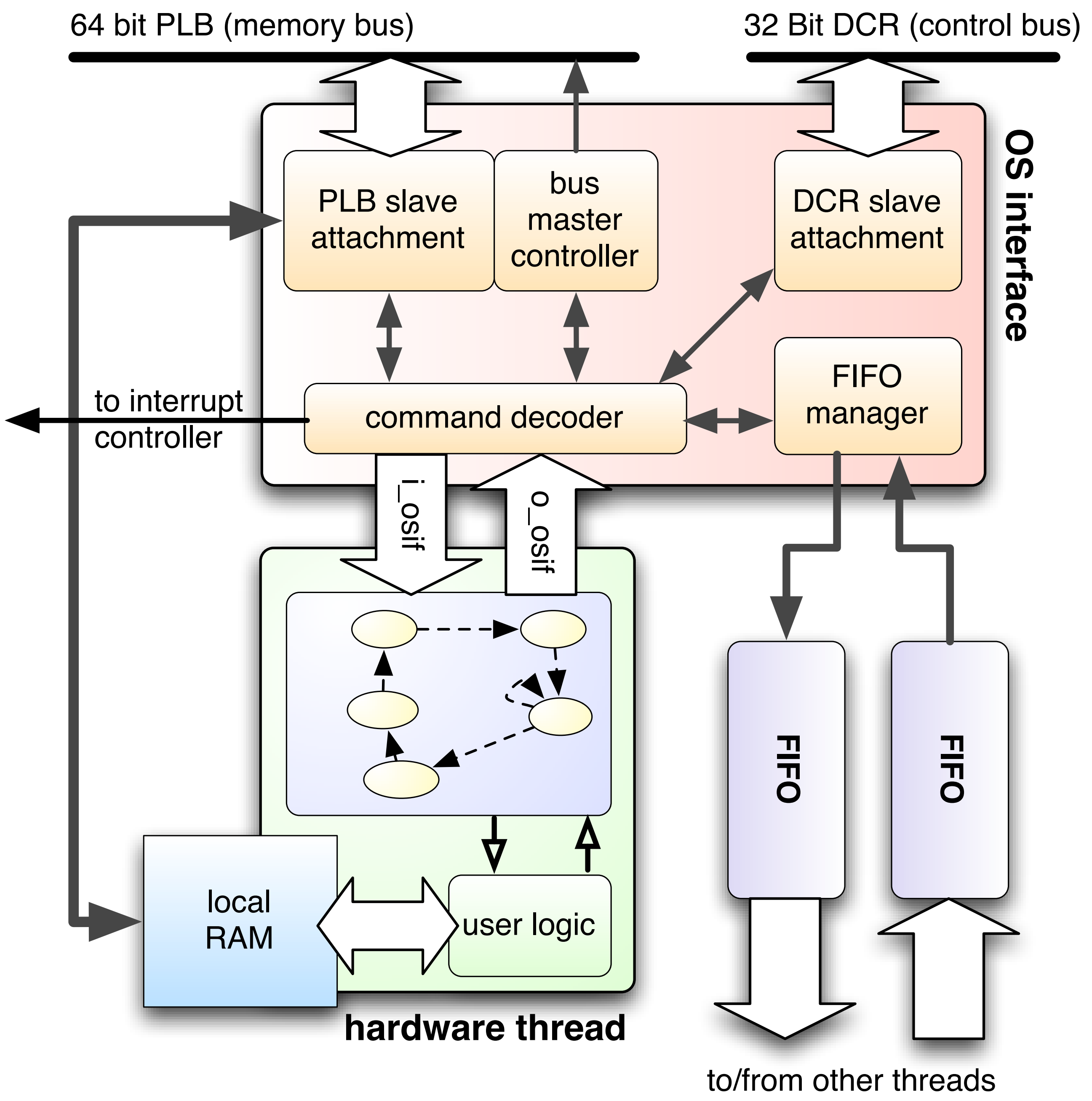
OS Interface

- 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)

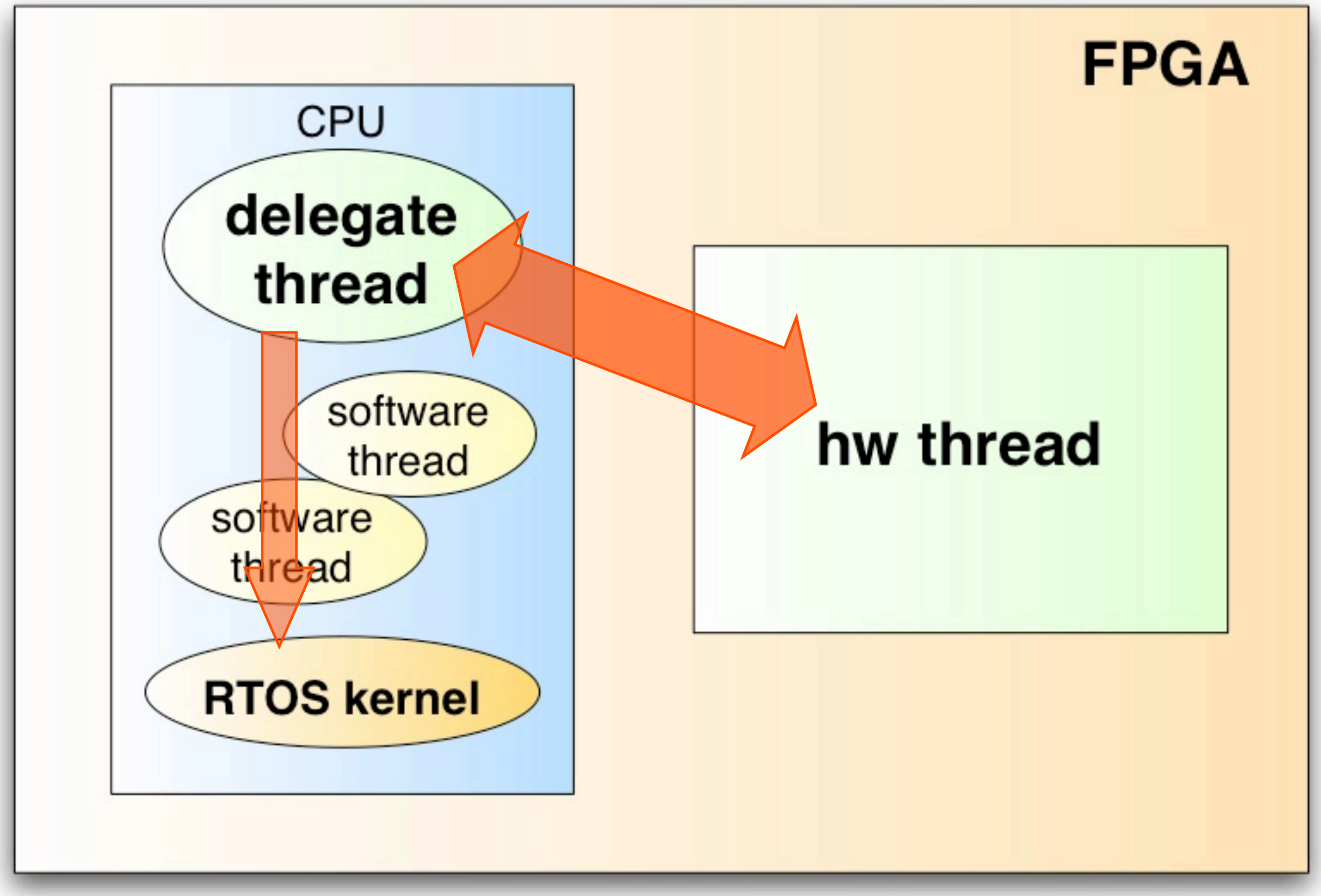


OS Interface

- 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

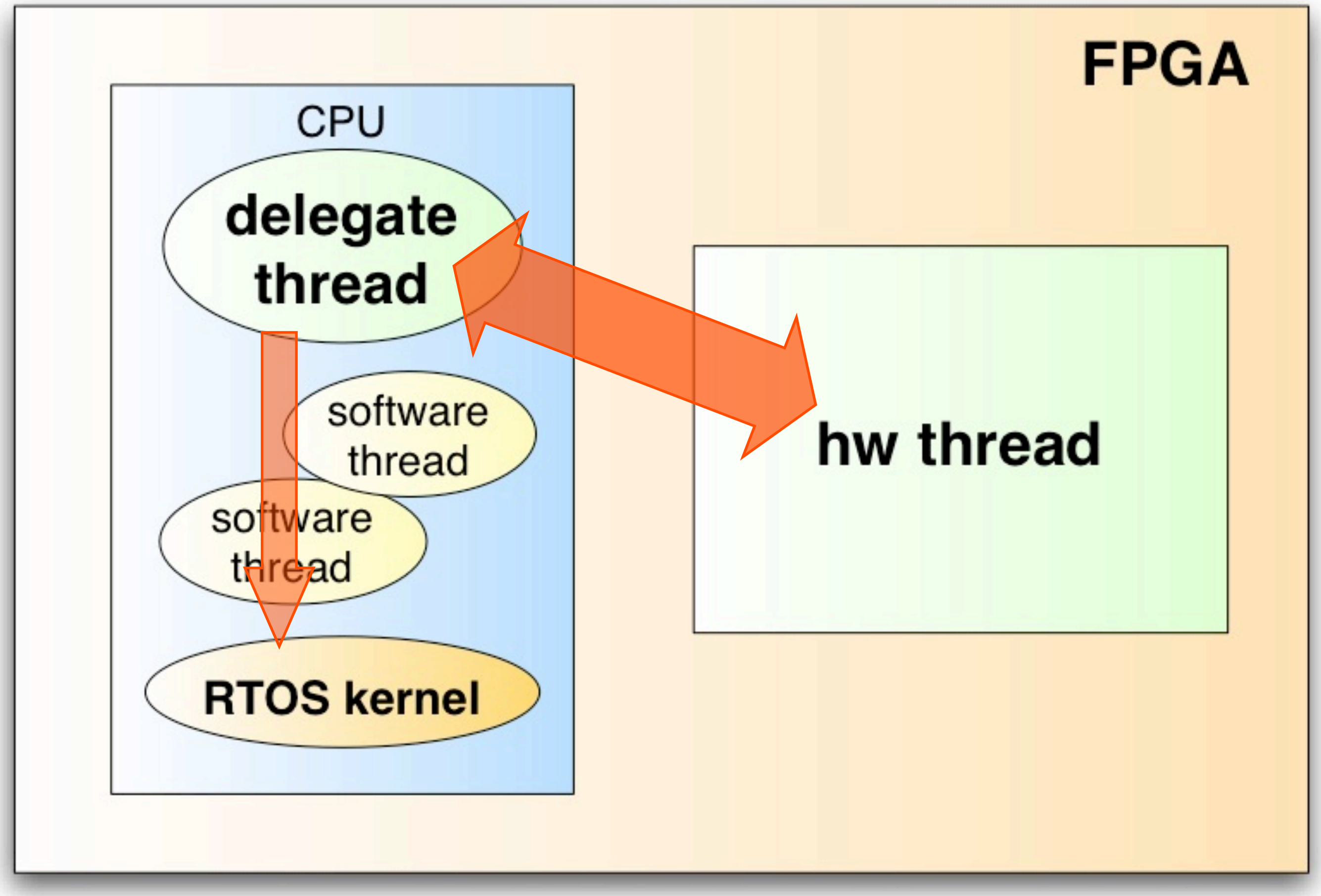


Delegate Threads



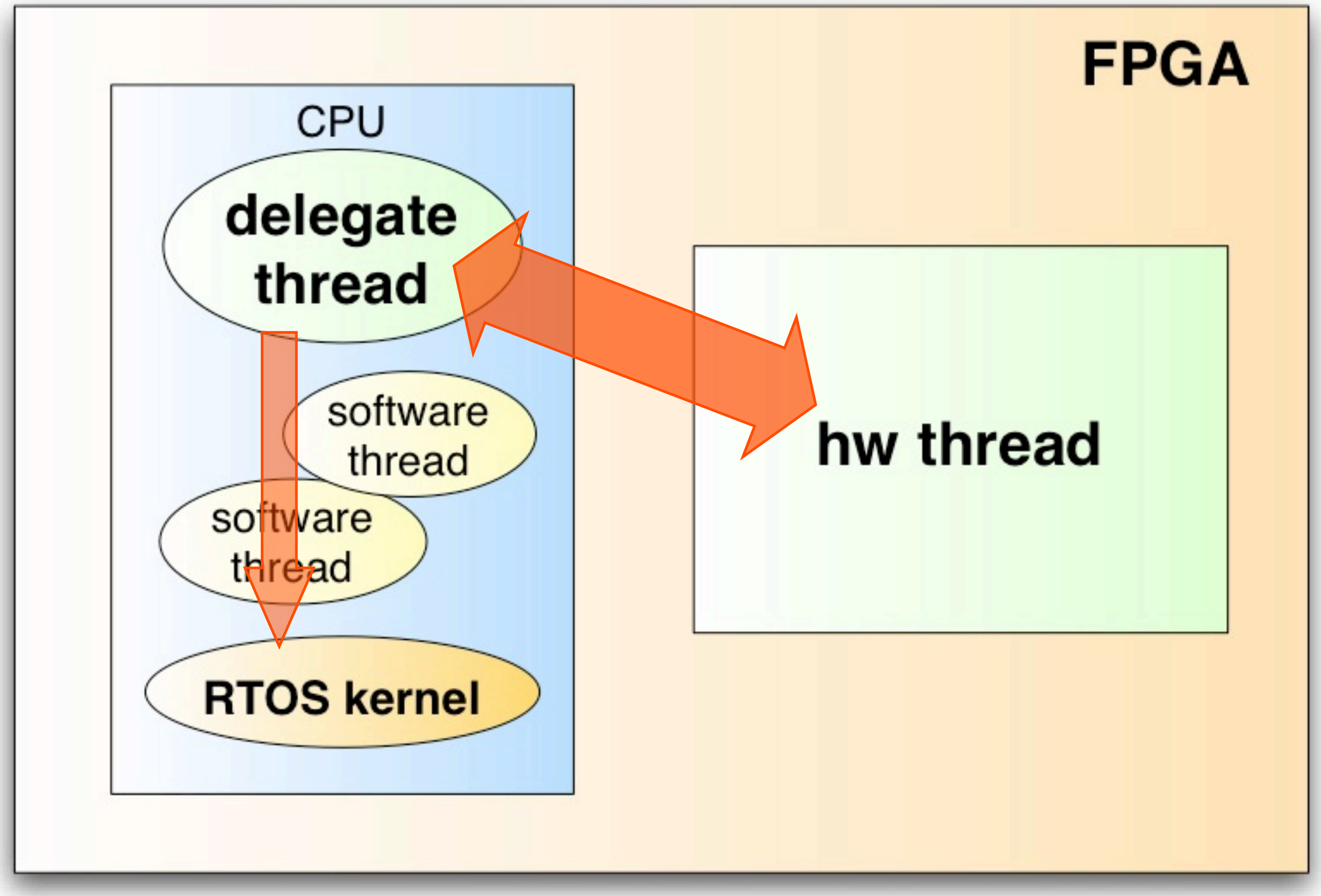
Delegate Threads

- 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
- portability
  - delegate acts as *protocol converter* between HW thread and OS kernel
  - only the delegate thread code needs to be changed to support a new OS API