RTOS-Centric Cosimulation for MPSoCs

Hiroyuki Tomiyama

Graduate School of Information Science
Nagoya University
http://www.ertl.jp/~tomiyama/

Team Members

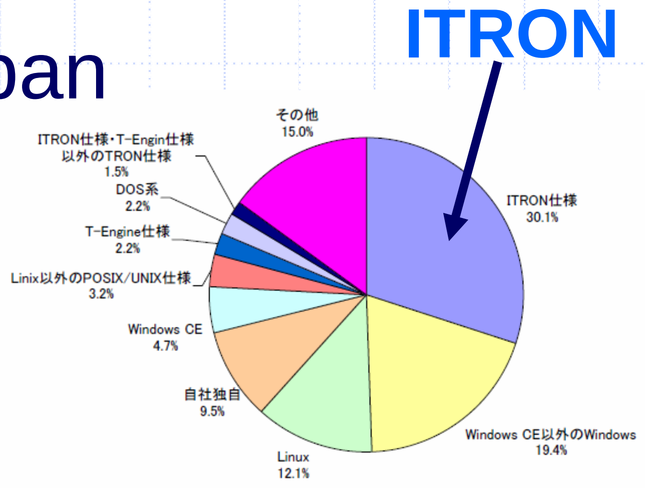
- Hiroaki Takada, *Professor*
  - Keynote speaker yesterday
- Hiroyuki Tomiyama, *Associate Professor*
- Shinya Honda, *Assistant Professor*
- Past and Current Students
  - Takayuki Wakabayashi, *Ph.D.* (currently with Sony)
  - Shin-ichiro Chikada (currently with Sony)
  - Takashi Furukawa
  - Seiya Shibata

Outline

- ITRON RTOS and TOPPERS/JSP Kernel
- RTOS-Centric Embedded System Design Methodology
- RTOS-Centric HW/SW Cosimulator
- Case Studies
  - JPEG Decoder on a Single Processor
  - MPEG Encoder/Decoder on Multiprocessors
- Summary

What’s ITRON?

- A standardized specification of RTOS kernel for small-to mid-scale embedded systems.
- Developed and standardized in Japan for >20 years
  - Prof. Takada has been playing the central role
- ITRON is not a software product but a specification.
  - Defines a set of API functions (service calls)
- Several *profiles* to cover different application domains
  - Standard Profile, Automotive Profile, etc.
- Most popular RTOS specification in Japan
  - 30 - 40% of embedded systems
  - especially in consumer electronics.



TOPPERS/JSP Kernel

- A reference implementation of the Standard Profile of ITRON 4.0
  - Initially, developed by Takada Laboratory
    - TOPPERS/JSP Kernel 1.0 released in November 2000
  - Currently, maintained by NPO TOPPERS Project
    - Incorporated in September 2003. (Chair: Prof. Takada)
    - http://www.toppers.jp/
    - >200 members (universities, companies, and individual volunteers)
    - The latest release is version 1.4.3 (June 2007)
  - Supported processors include

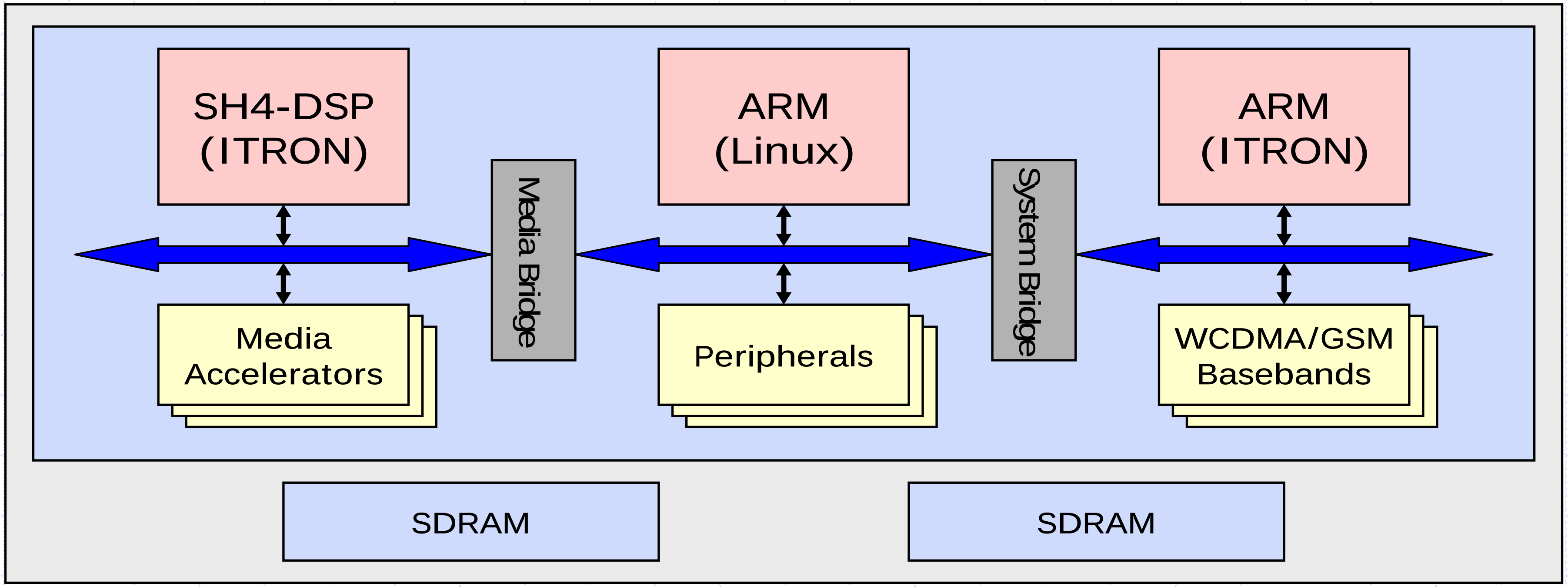
Motorola 68K	Renesas SH1/3/4, H8, M32R
ARM7/9	MIPS3
Xilinx MicroBlaze	TI TMS320C54x
Intel i386	NEC V850
Tensilica Xtensa	and much more!

TOPPERS/JSP Kernel (cont.)

- Compact and highly portable
- Free, open source software
  - Can be used for research, education, and even commercial purposes.
- Production quality
  - Actually, used in a number of commercial products
  - Examples include
    - Karaoke microphone “Do! Karaoke” by Matsushita Electric Industrial Co., Ltd. (Panasonic), February 2003.
    - Ink-jet printers from Epson and Brother
    - Digital pianos from Roland
    - NC machines from Okuma
- Released with a simulation model
  - Executable on Windows and Linux host computers

Motivations

- Embedded systems continuously grow in size and complexity.
- Ex. Cellular phones
  - Phone, e-mail, digital still camera, video camera, TV phone, web browser, TV, e-money, etc.
  - Multi-millions lines of code
- Renesas SH-Mobile G1
  - Heterogeneous multiprocessors

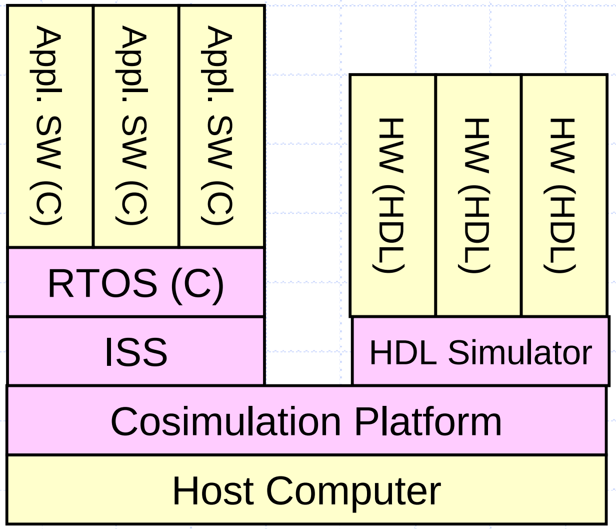


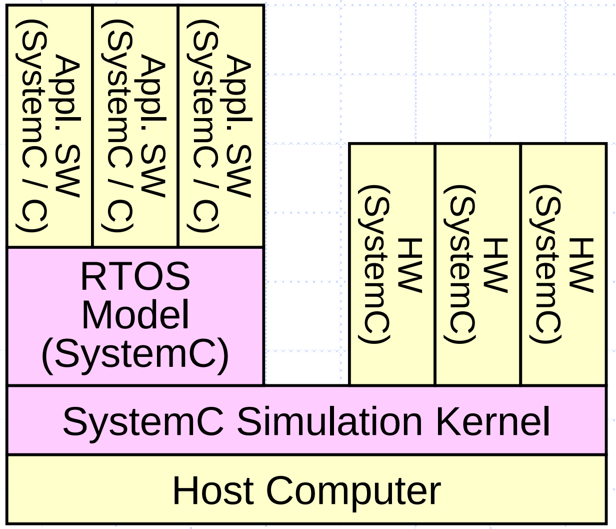
Motivations (cont.)

- RTOS plays an important role in such complex embedded systems.
  - Task scheduling, inter-task communication and synchronization, resource management, etc.
- RTOS should be cosimulated with hardware and application software in order to validate the overall system functionality.
  - Such cosimulation should be done at a very early stage of system design

Traditional Cosimulation with RTOS

- Very traditional cosimulation approach
  - RTOS on ISS + HDL simulator
  - Accurate timing
  - Slow simulation speed
- Generic/simple RTOS model in SystemC/SpecC
  - [Prof. Imai, MPSoC 2007]
  - [Desmet, DAC 2000]
  - [Gerstlauer, DATE 2003]
  - etc.
  - Fast simulation
  - Only for simulation
    - Need rewriting for synthesis/compilation
  - Support a very limited set of RTOS services
    - Actual RTOSs have more than 100 service calls
      - >50 service calls even in small kernels



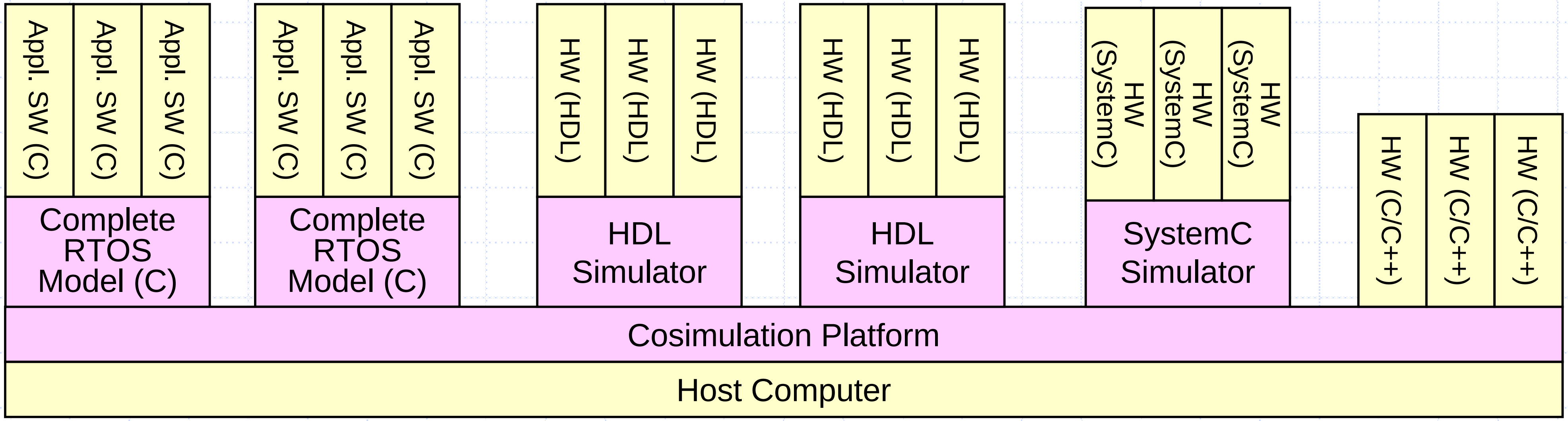


Our Cosimulator

- Complete simulation model of a standard RTOS kernel, i.e., *ITRON*
  - No need to rewrite application software for implementation
- Native, hence fast, execution of software
  - Easily replaceable with ISS
- Cosimulation with hardware models in SystemC/C++/C
  - Fast cosimulation at different abstraction levels
    - Untimed functional
    - Timed functional
    - Bus-transaction level
    - Cycle-accurate
- Cosimulation with hardware designs in HDL
  - Smooth synthesis
- Support for multiprocessor systems

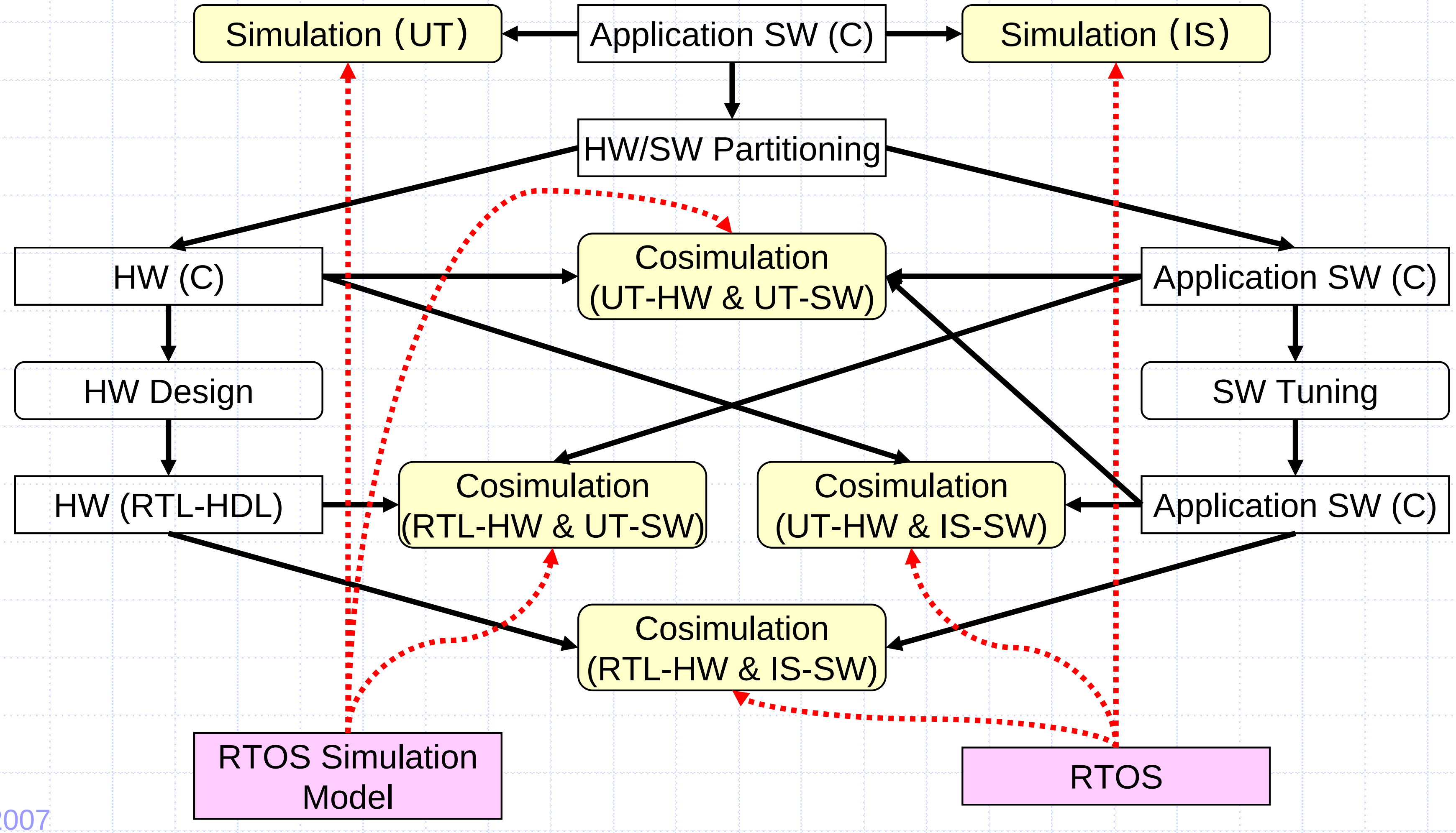
Our Cosimulator (cont.)

- Complete simulation model of ITRON RTOS
- Native, hence fast, execution of software
- Cosimulation with hardware models in SystemC/C++/C
- Cosimulation with hardware designs in HDL
- Support for multiprocessor systems



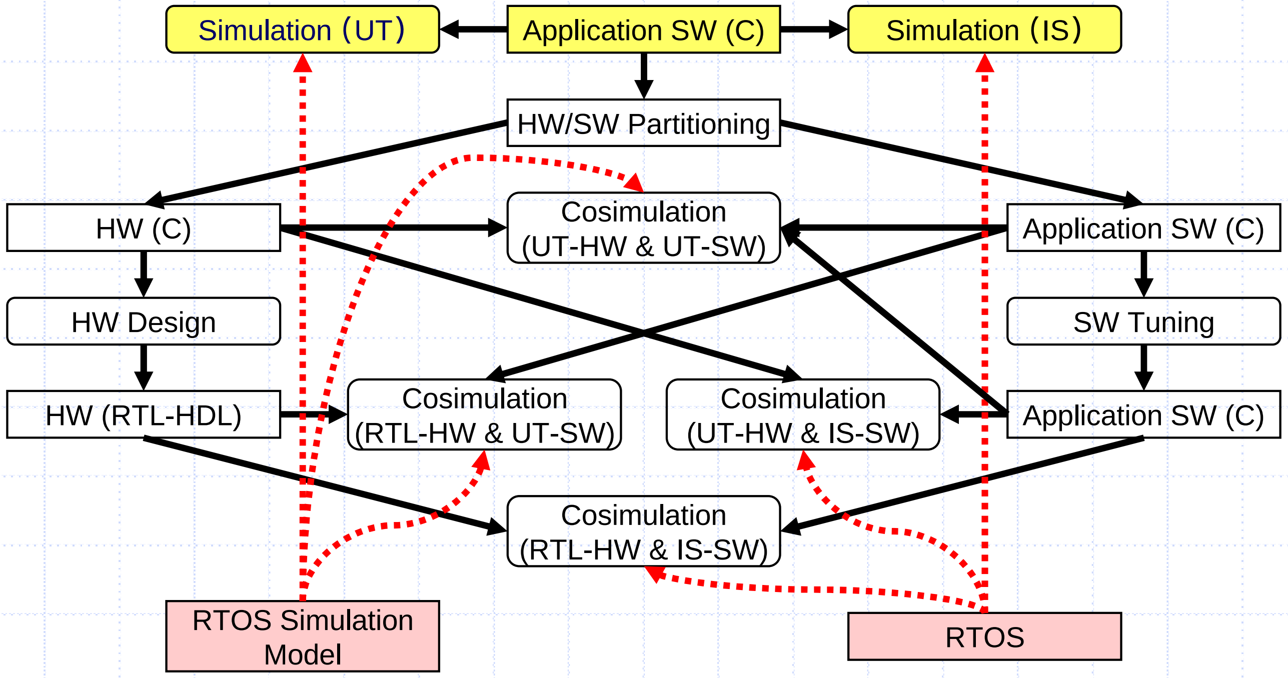
RTOS-Centric Design and Verification Methodology

- Using a complete simulation model of RTOS from a very early stage of the system design



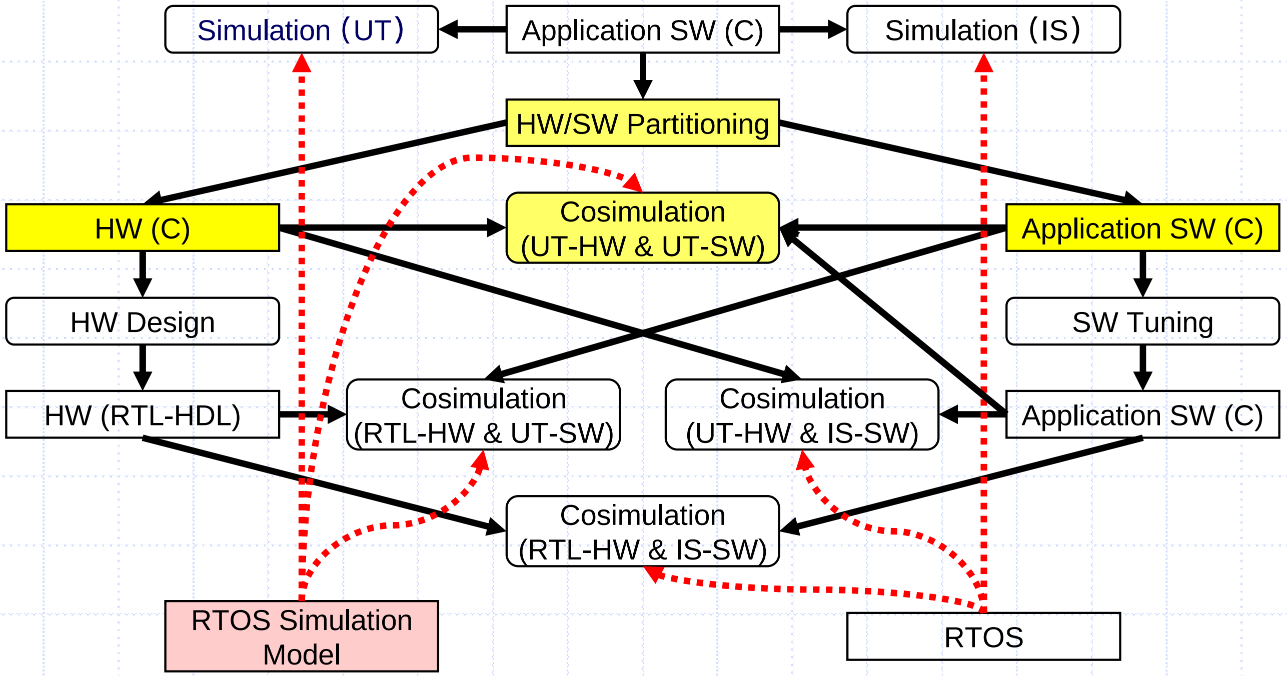
Specification and Simulation

- Describe a system specification as a set of application tasks in C.
  - Use RTOS service calls for communication and synchronization between the tasks.
- Untimed simulation (UT)
  - Compile and link the applications with RTOS simulation model
- Instruction-set level simulation (IS)
  - Find the performance-critical tasks (or parts of tasks) as candidates for hardware implementation



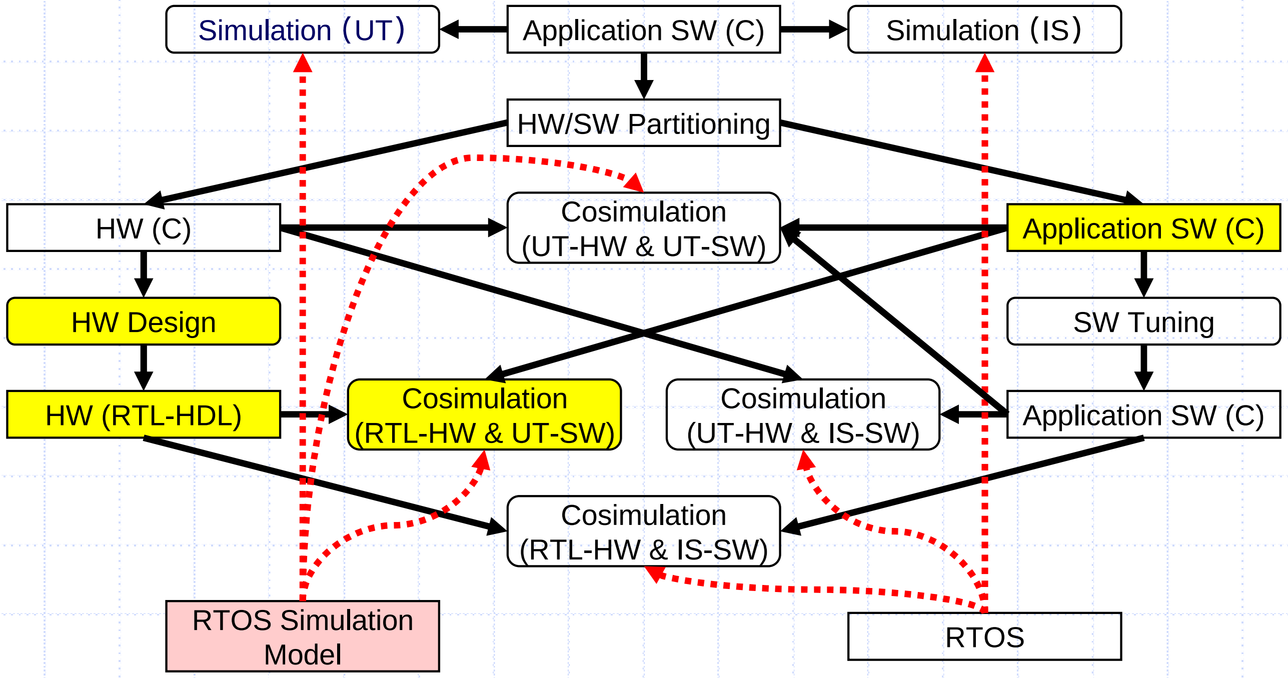
Partitioning and Cosimulation

- HW/SW partitioning and task mapping on multiprocessors
- Untimed cosimulation (UT-HW & UT-SW)
  - Native execution of both SW and HW
    - Functional hardware model in C
  - Validate the functional correctness of the partitioning.



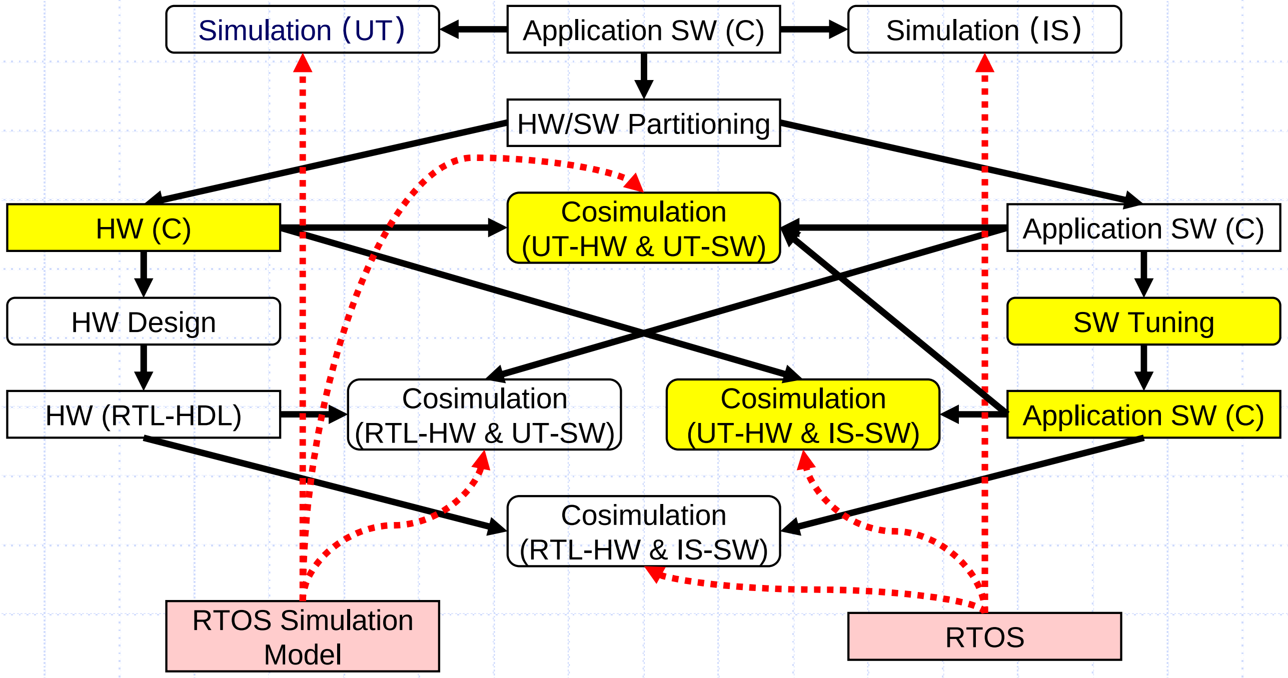
HW Design and Debugging

- Behavioral synthesis or manual design.
- Cosimulation (RTL-HW & UT-SW)
  - RTL debugging on an HDL simulator.
  - Native software execution on the host computer
    - The software serves as an interactive testbench.



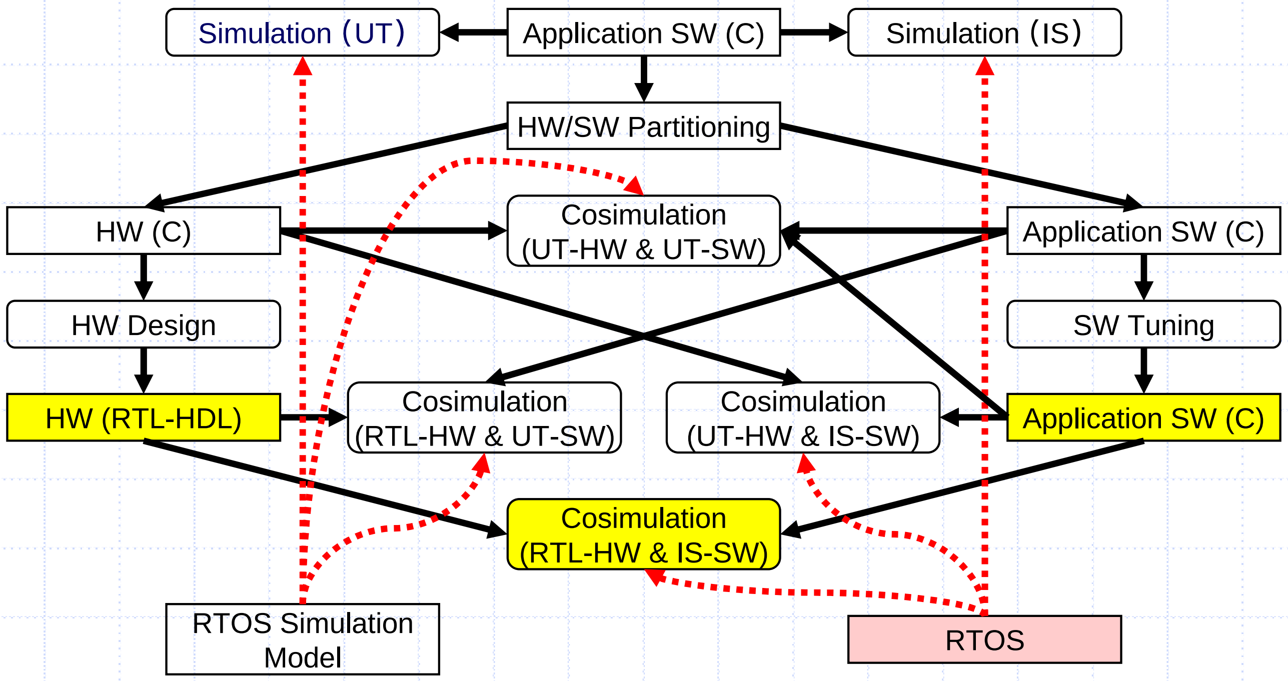
Software Tuning

- Optimize software to fully utilize memory hierarchy and hardware architecture
- Cosimulation (UT-HW & IS-SW)
  - Software execution on instruction-set simulator
  - Untimed hardware model
    - The RTL design may not be completed



Instruction/Cycle-Accurate Cosimulation

- Traditional cosimulation
  - Hardware simulation on HDL simulator
  - Software execution on ISS
- Evaluate the overall performance
  - Go back to HW/SW partitioning, HW design or SW tuning if performance constraint is not satisfied

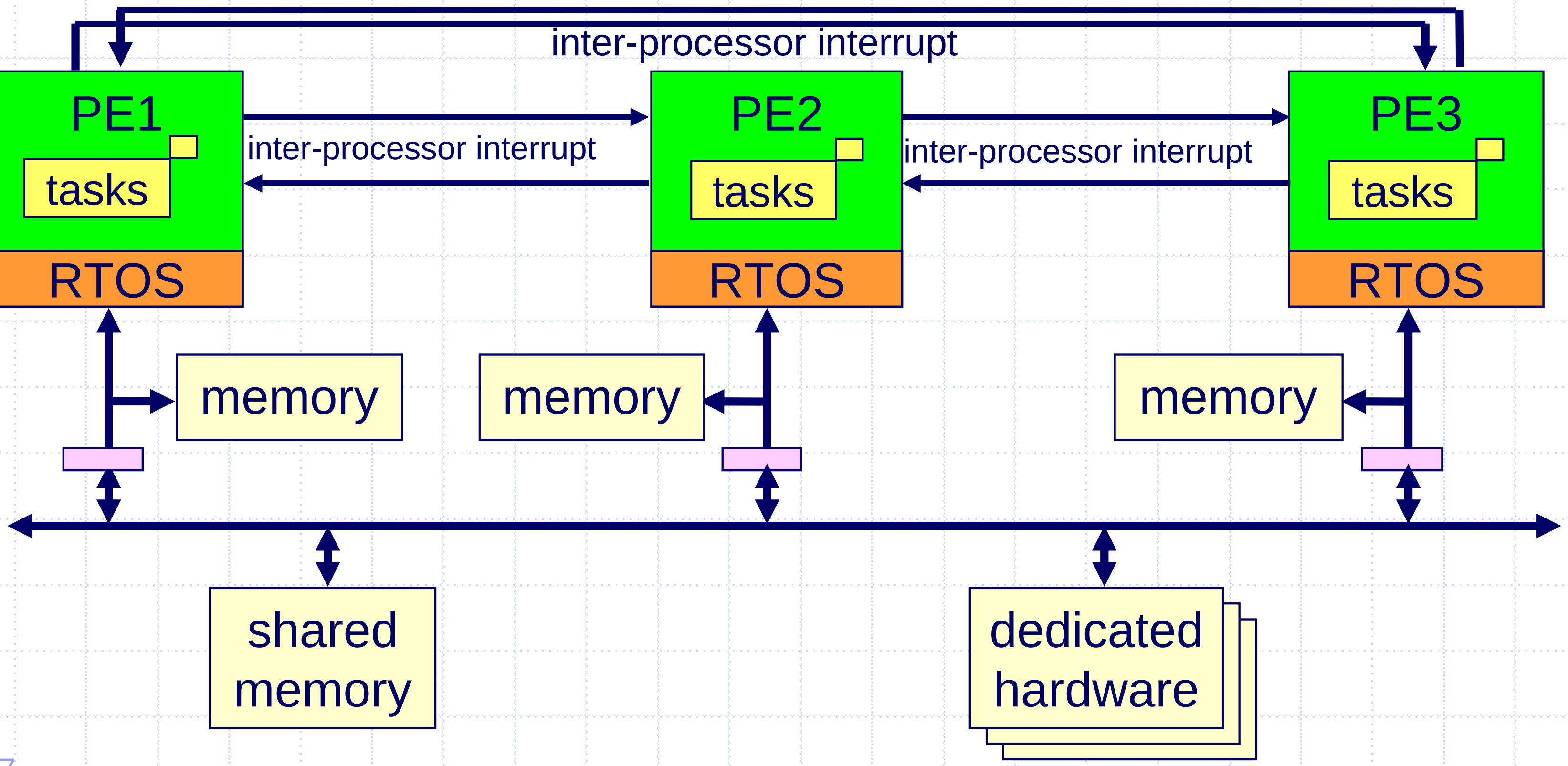


Key Features

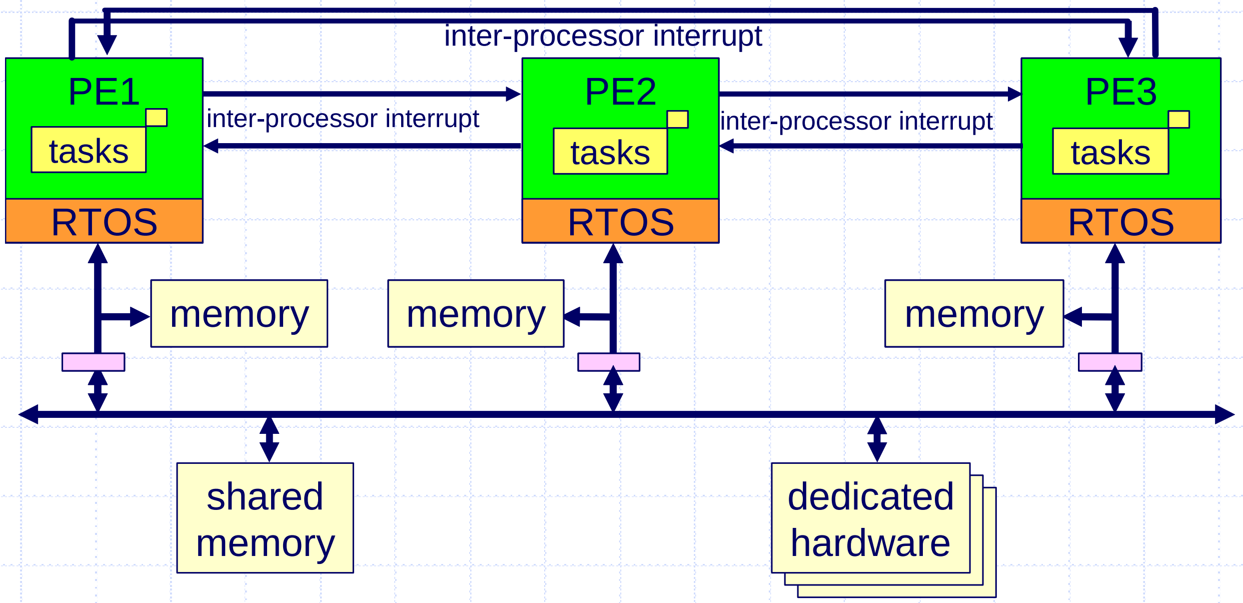
- Use of complete RTOS model from a very early stage of the system design
  - Smooth implementation
    - No need to rewrite application software from specification to implementation
    - Legacy code can be easily reused
- Flexible multilingual cosimulation
  - Efficient validation due to the single cosimulation platform at different levels of abstraction
- Drawbacks? => No!
  - Heavily dependent on a specific RTOS, i.e., ITRON
    - Most design teams in industry want to use the same RTOS as long as possible because RTOS is the most fundamental platform
    - One design team uses only a few RTOSs
  - RTOS-dependent code is too low level
    - Model-driven design can be used on top of our design flow

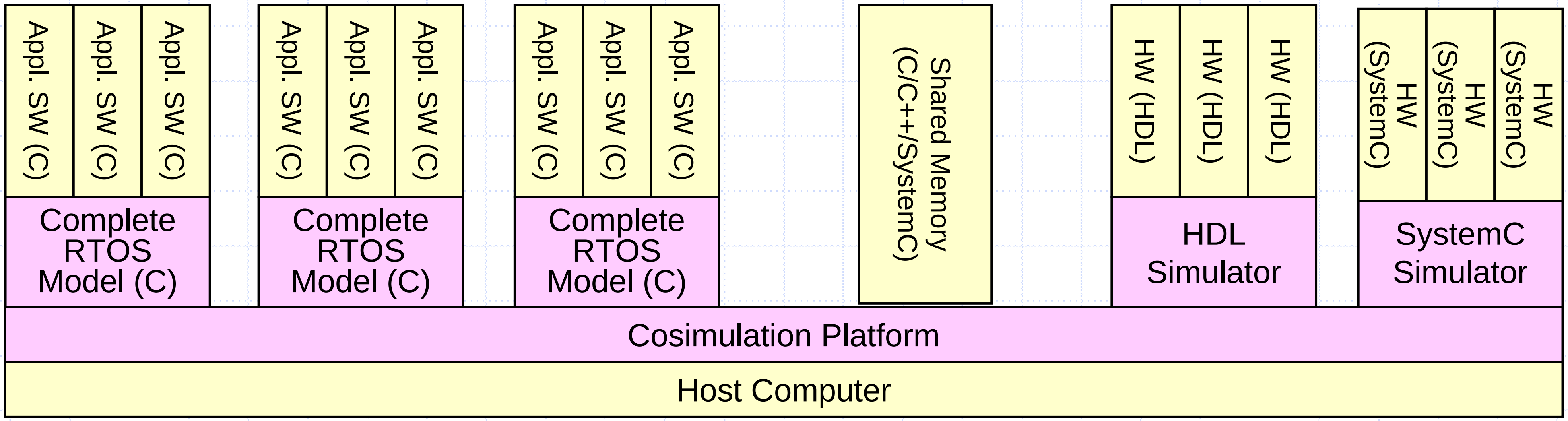
Target Multiprocessor Systems

- RTOS runs on each processor
- Static task allocation
- Shared memory & local memory
- Dedicated hardware



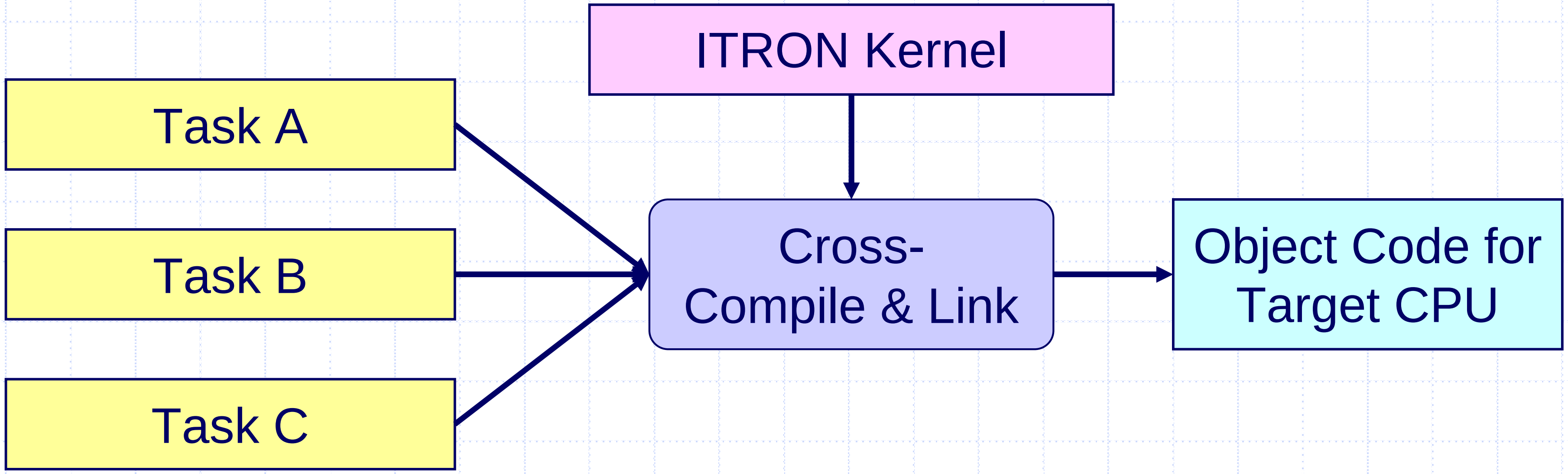
Our Cosimulator



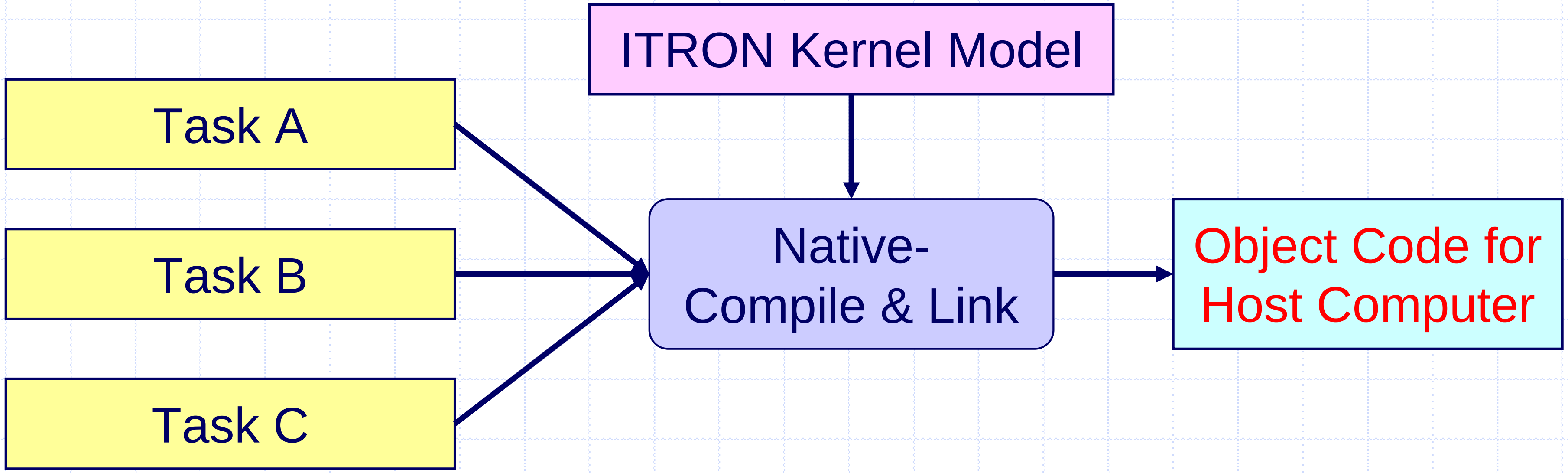


Task and Memory Management in ITRON Standard Profile

- Priority-based preemptive scheduling
- Tasks are statically defined at design time
  - No dynamic loading at run time
- Single memory space shared by all of application tasks and RTOS



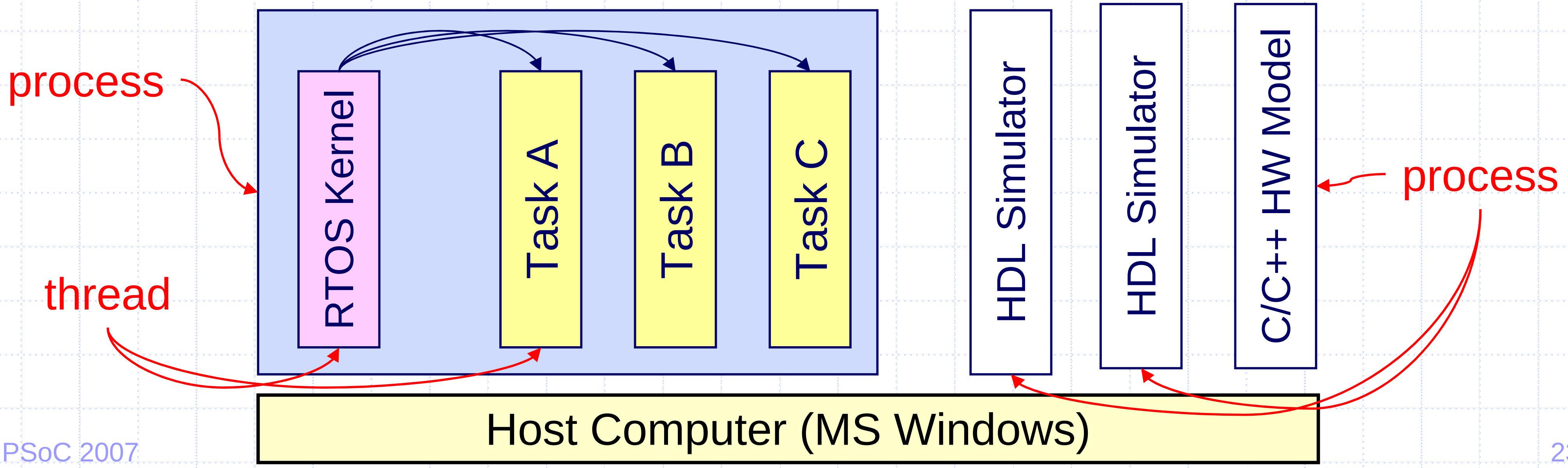
A Straightforward Approach to Native Simulation



- Difficult to debug the software without RTOS-specific supports
  - From the viewpoint of a host computer, the object code is no more than one application process.
  - Hard to observe the context of individual ITRON tasks.

Our Approach: Multi-Threading

- Multi-threaded implementation
  - Each application task is implemented as a thread.
  - The ITRON kernel is also a thread.
    - The kernel thread controls (e.g., start and suspend) the task threads by means of inter-thread communication/synchronization.
- From the host computer's view, both the kernel and the tasks are threads, and scheduled independently.
  - Easy to monitor the individual tasks with a normal C/C++ debugger.



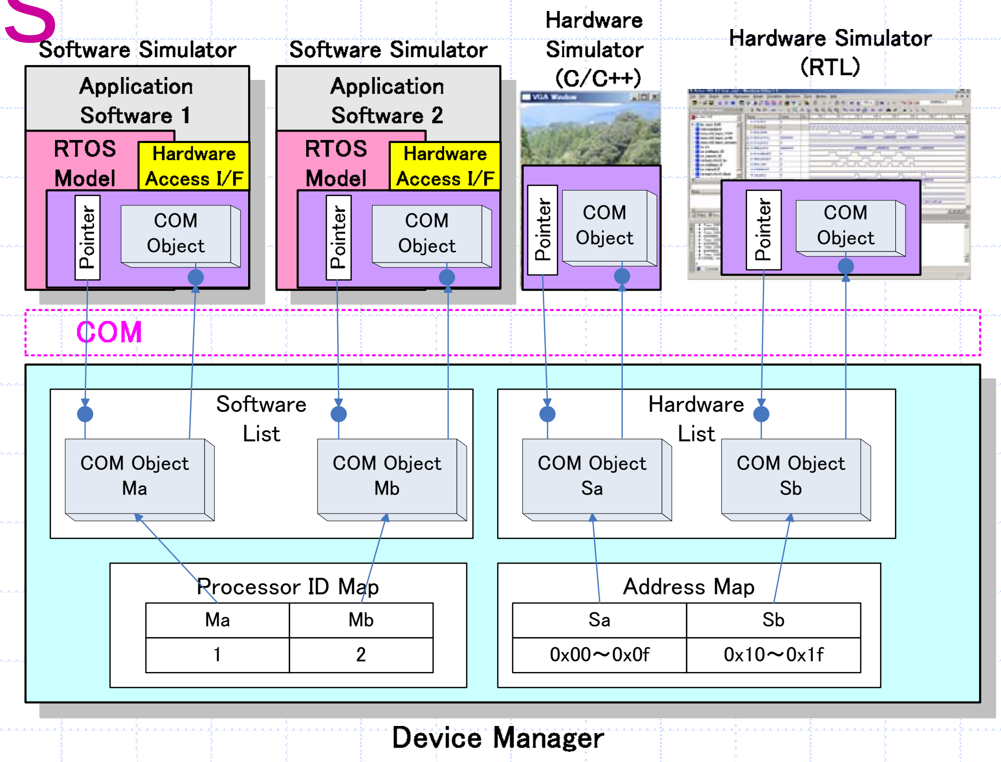
Communication

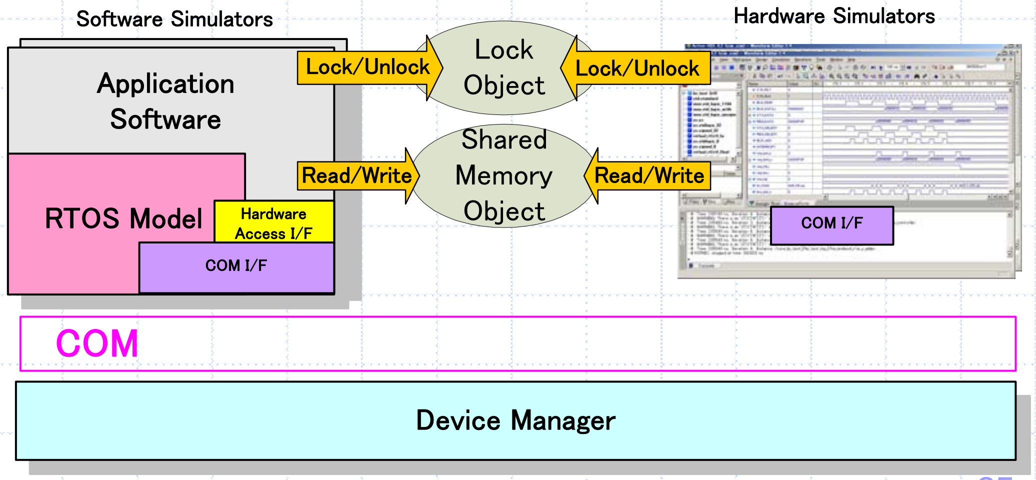
- Type types of communication
  - Read/write accesses based on memory mapped I/O
    - ITRON service calls must be used in application tasks
      - Ex. *sil_reb_mem(addr)*
        - Read 1-byte data from memory pointed by *addr*.
  - Interrupts
    - From hardware to processor
    - From processor to processor
- Implemented with the COM technology
  - COM (Component Object Model)
    - Object-oriented binary-level communication technology for MS-Windows applications

```
char *addr = 0x000f0000;
c = *addr;
```

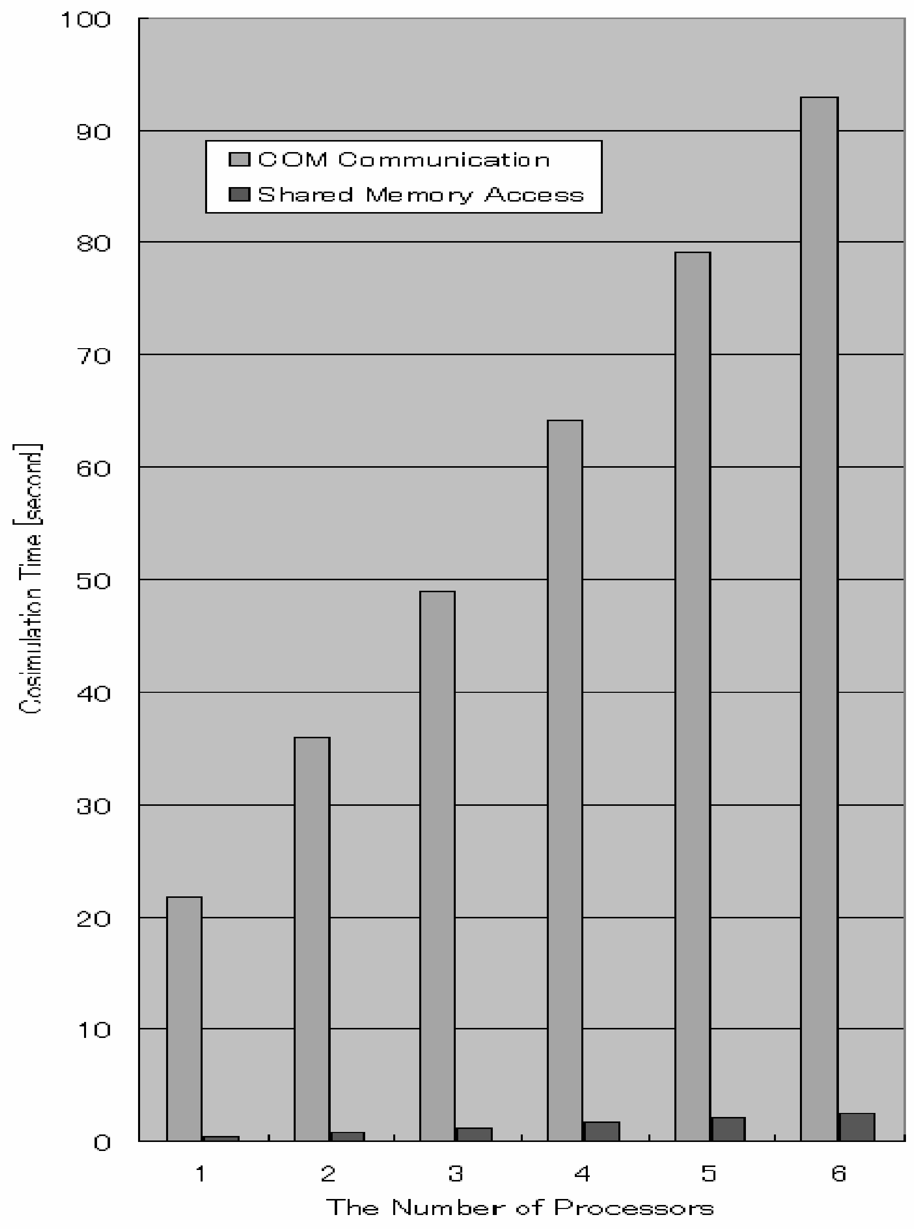
Read/Write Accesses

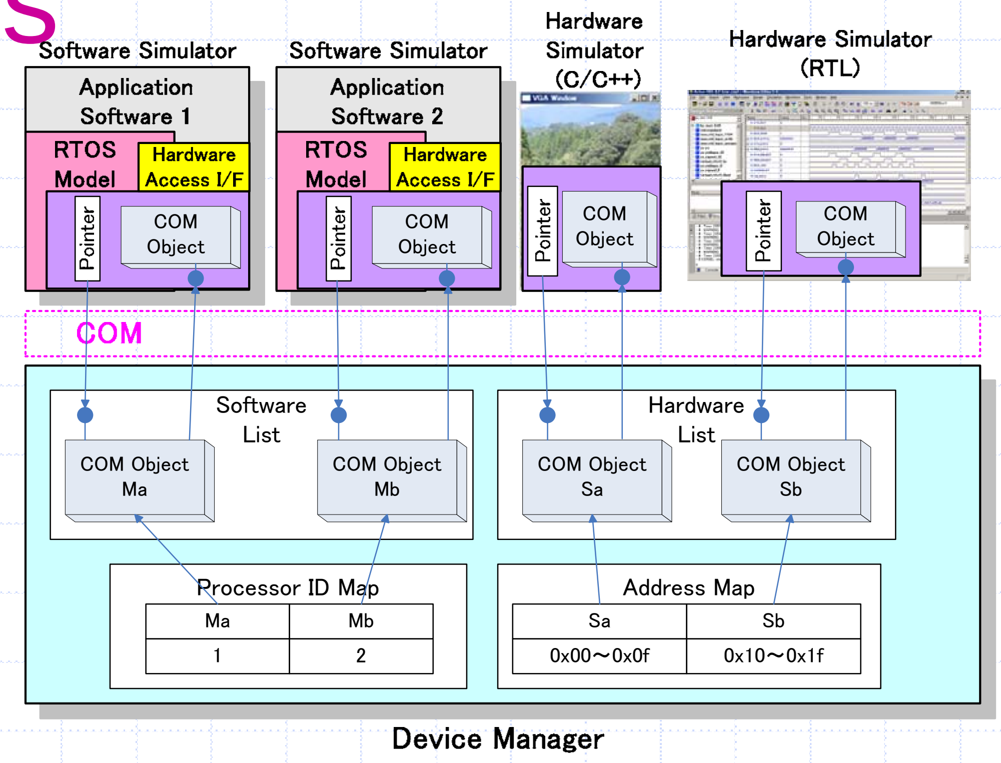
- Two implementation methods for simulation
- RPC-based
  - Describe memory model in hardware simulator
  - Flexible but slow
- Shared memory-based
  - Create shared memory on the host computer
  - Suitable for burst accesses

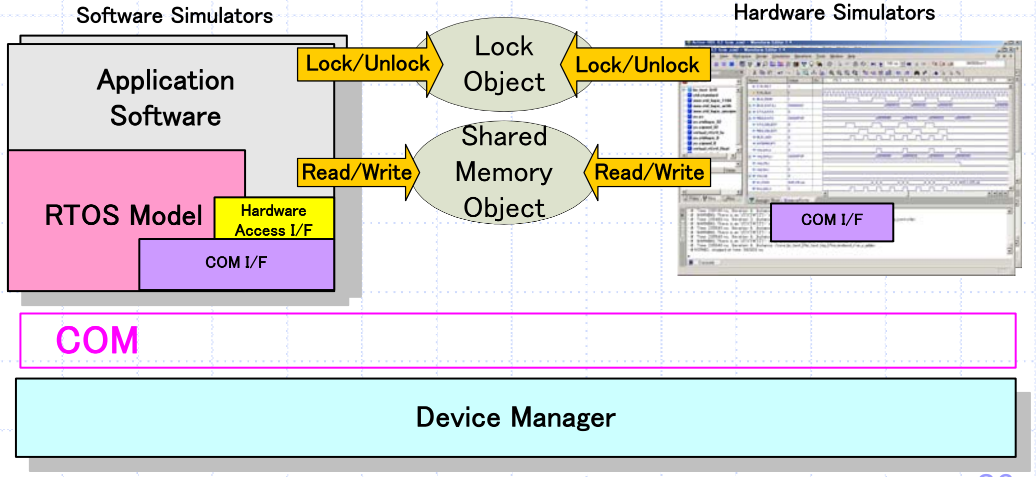




Read/Write Accesses

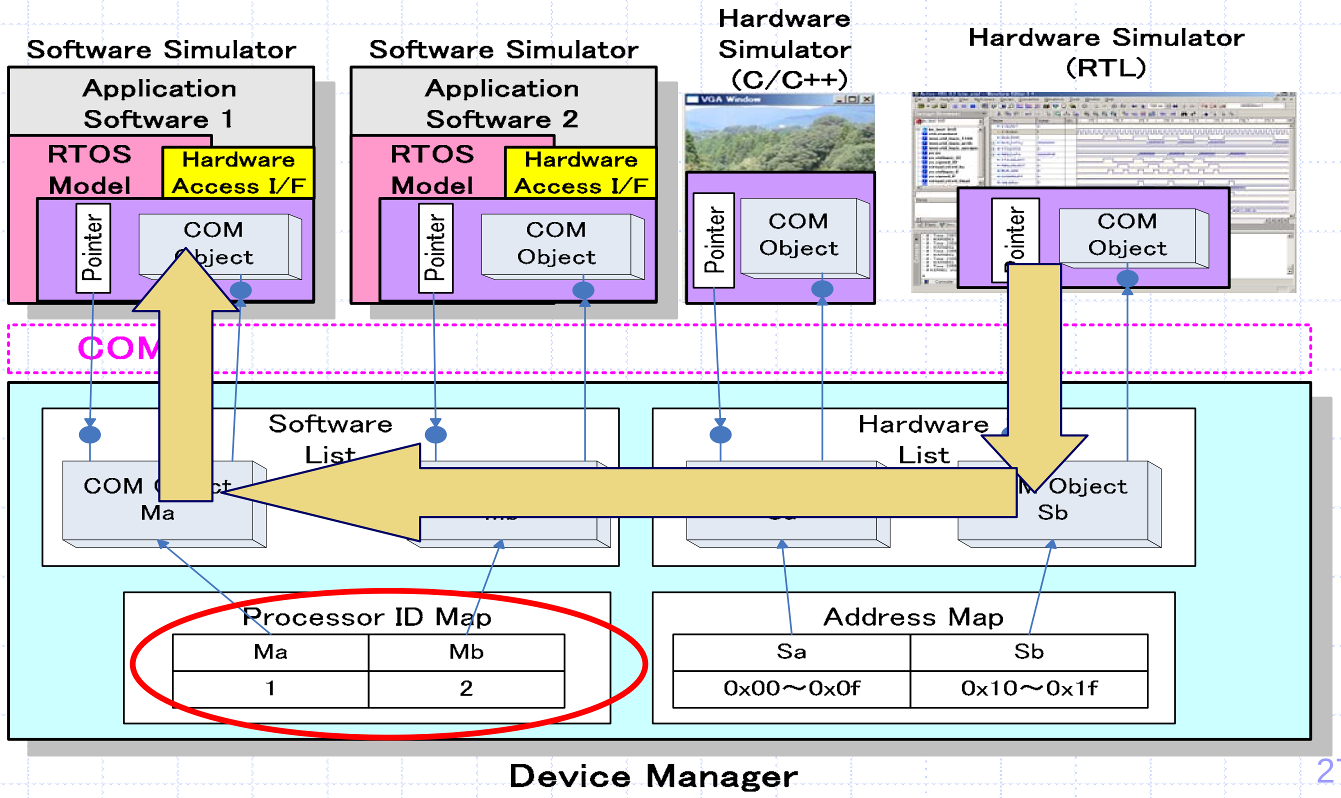






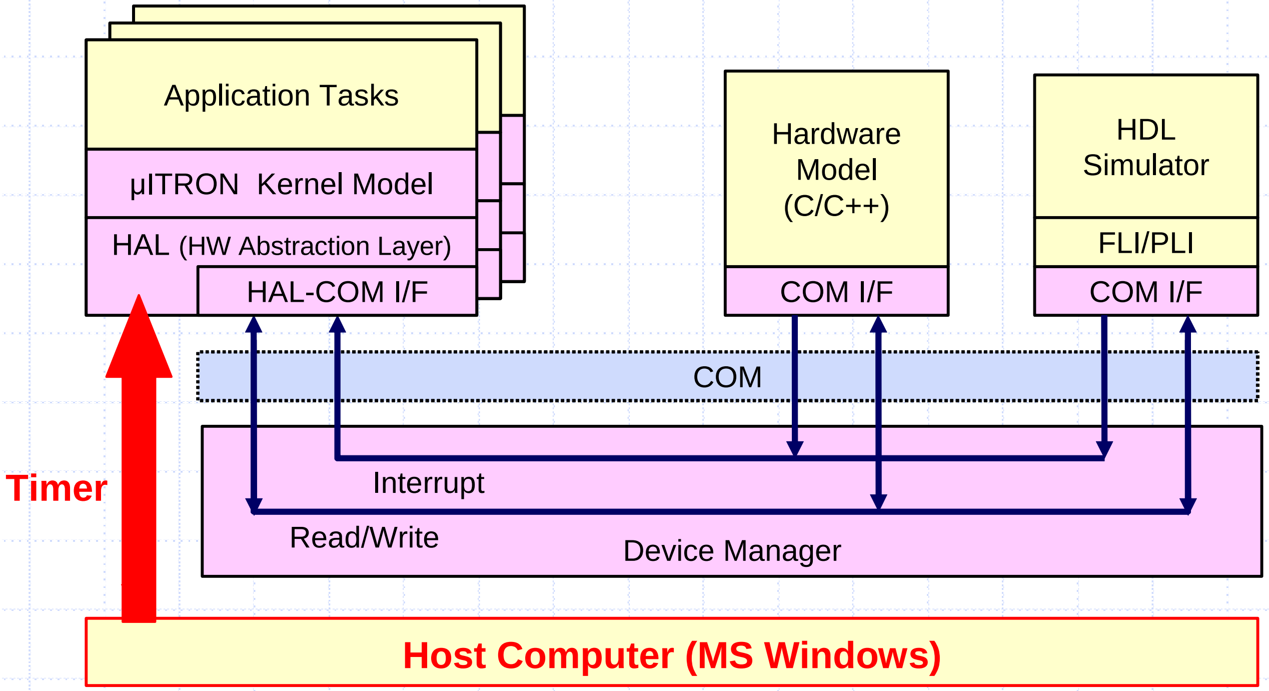
Interrupts

- Two types of interrupts
  - from processor to processor
  - from hardware to processor
- Both implemented with RPC

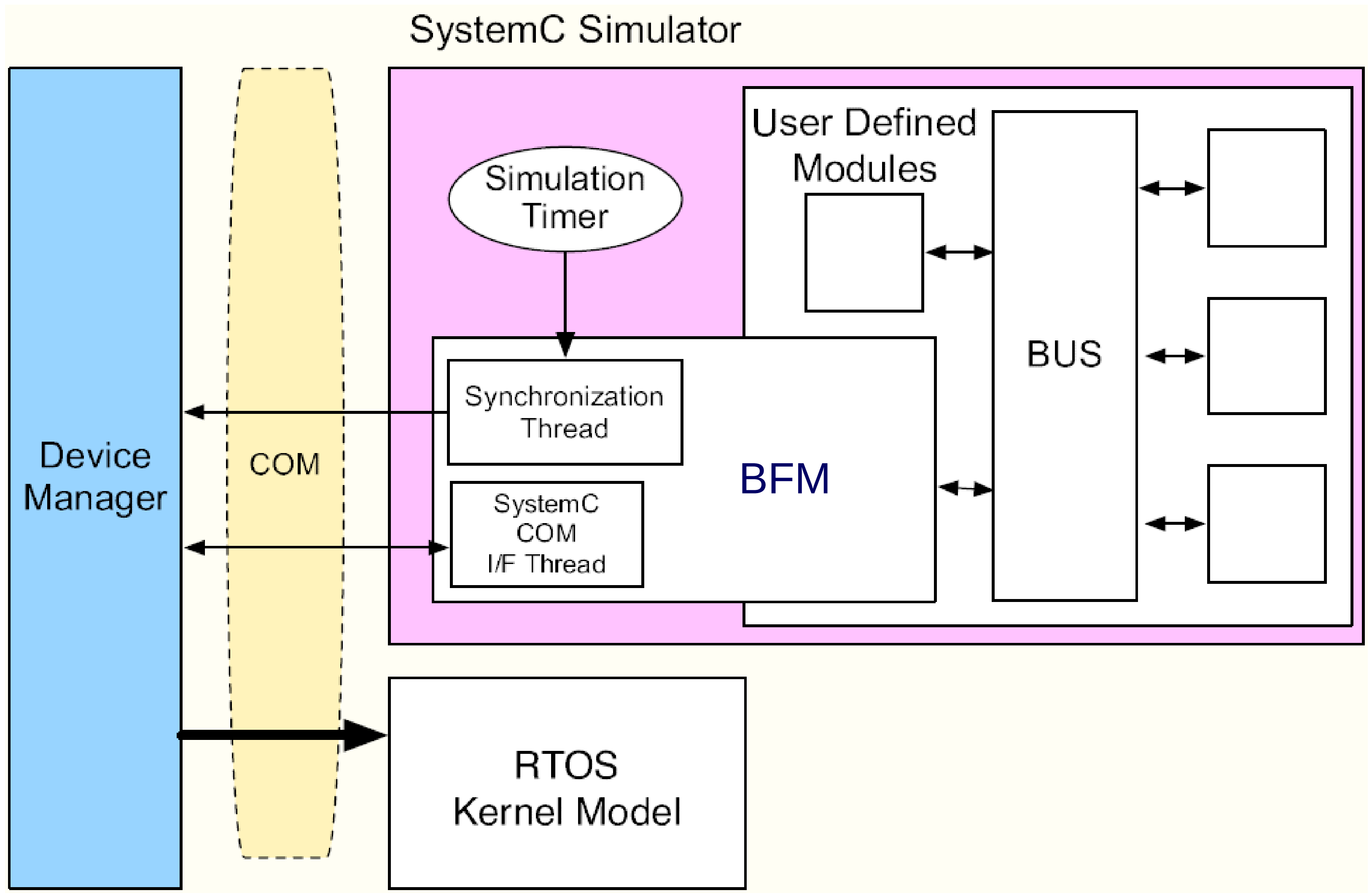


HW/SW Synchronization

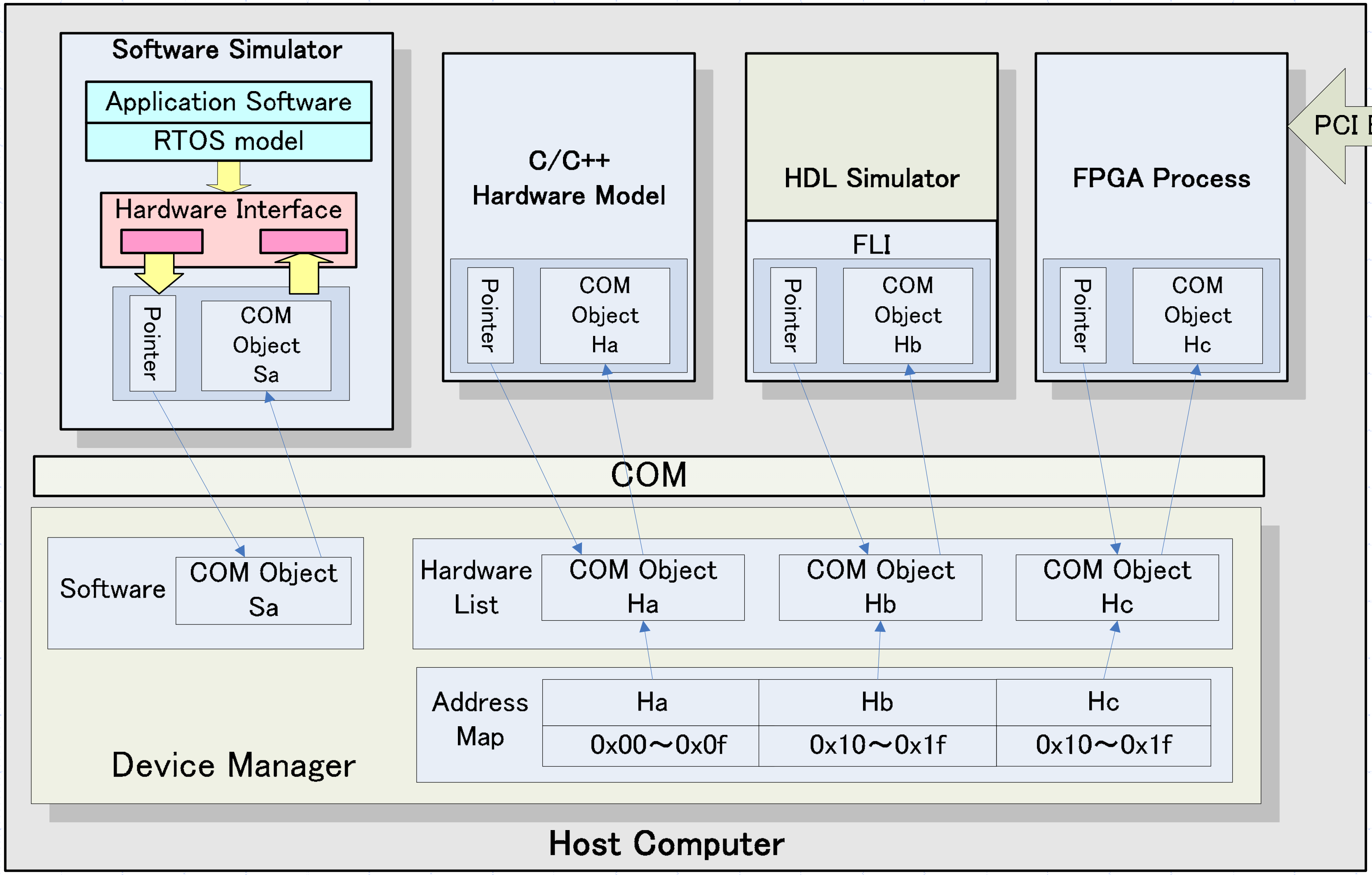
- Actual RTOS needs clock ticks (interrupts) from a hardware timer
  - Clock hander in RTOS manages local clock on each processor
- Similarly, our RTOS simulation model needs clock ticks from somewhere outside
- Two options in our cosimulator
  - Use a timer of MS-Windows
    - Software runs in real time on the host machine, so timing is not accurate at all
  - Describe a simulation model of hardware timer, e.g. in SystemC
    - Timing accuracy is clock tick level (typically, 100us - 1ms)

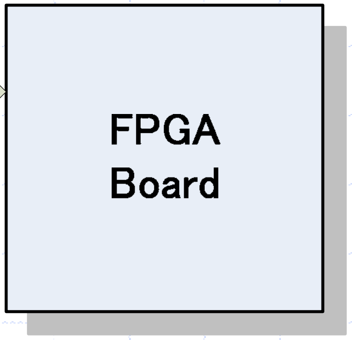


Cosimulation with SystemC



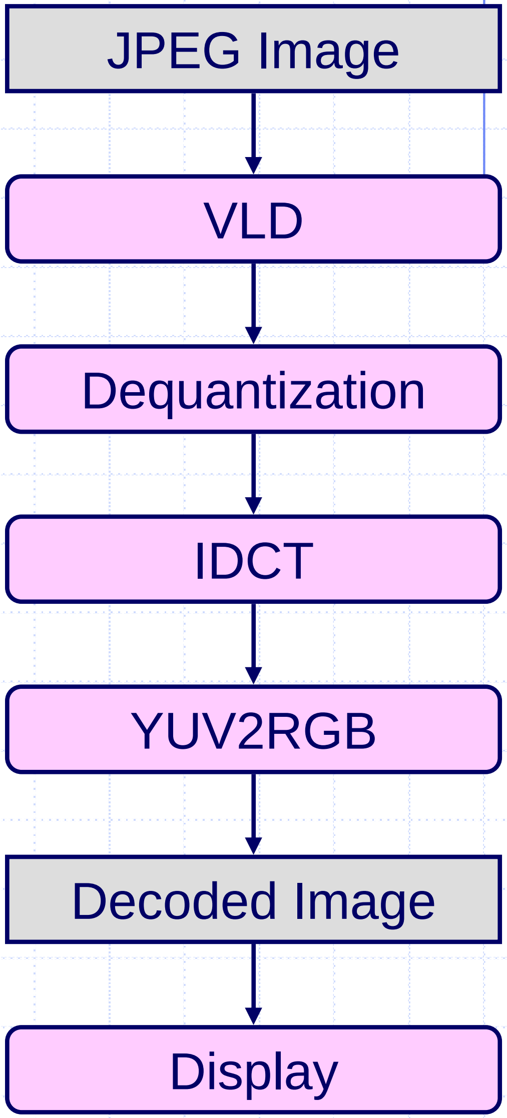
Covalidation with FPGA





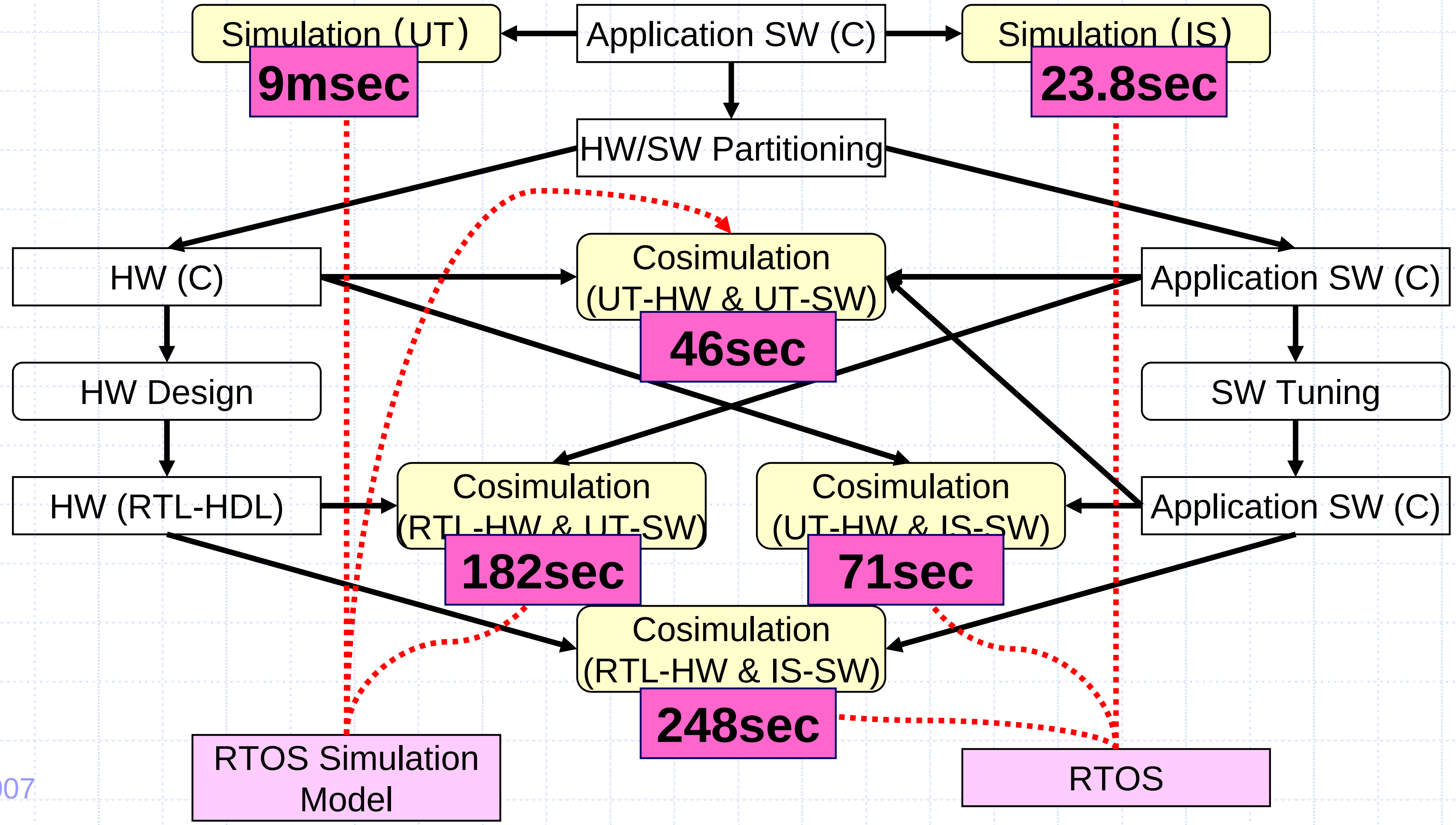
Case Study 1: JPEG Decoder on a Single Processor

- JPEG Decoder
  - Four tasks: VLD, Dequantization, IDCT, and YUV2RGB
  - Display: displays the decoded image on a windows of a host computer during simulation.
  - Decode image size: 240x320
- IDCT implemented in hardware
- Design environment
  - Dual 2.4GHz-Xeon processors with hyper-threading
  - Windows XP
  - HDL simulator: ModelSim
  - ISS: ARMulator

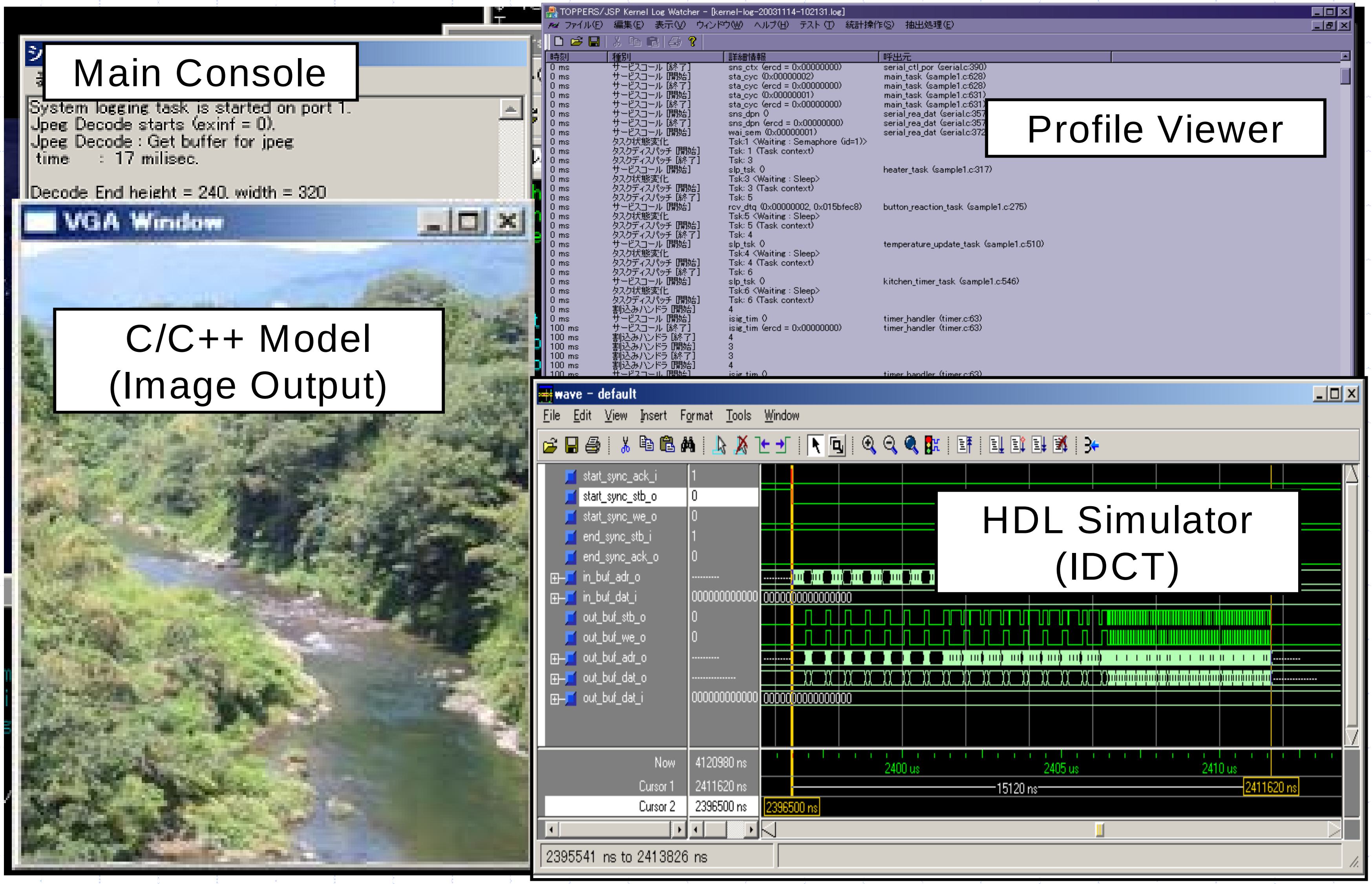


Cosimulation Time

- Cosimulation time was largely different depending on the abstraction levels of both software and hardware.
- In our methodology, comimulation can be performed at just necessary level of abstraction.



Snapshot of Cosimulation (RTL-HW & UT-SW)

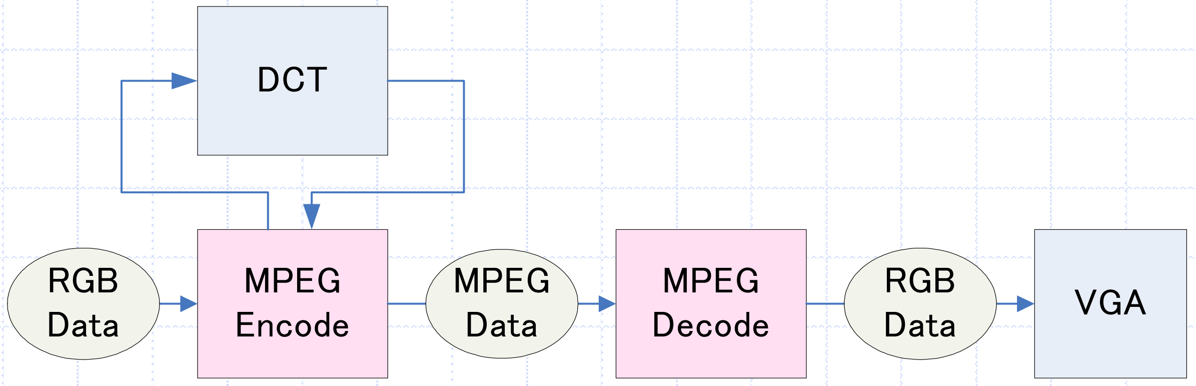


```
System logging task is started on port 1.
Jpeg Decode starts (exinf = 0).
Jpeg Decode : Get buffer for jpeg
 time      :  17 milisec.

Decode End height = 240, width = 320
```

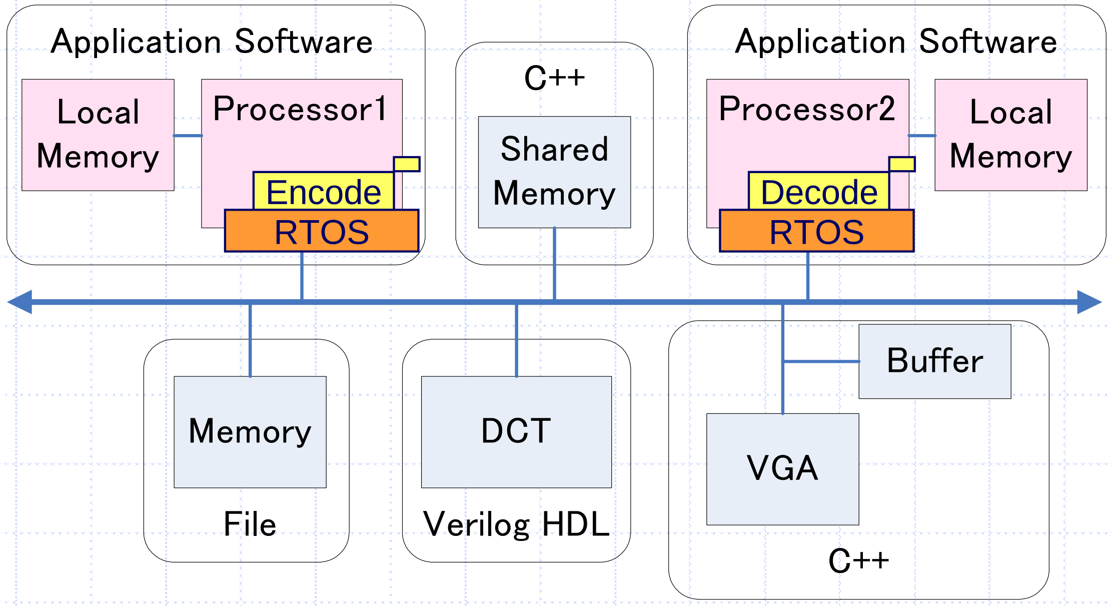
```
TOPPERS/JSP Kernel Log Watcher - [kernel-log-20031114-102131.log]
ファイル(F)  編集(E)  表示(V)  ウィンドウ(W)  ヘルプ(H)  テスト (T)  統計操作(S)  抽出処理(E)

時刻     種別                    詳細情報                               呼出元
0 ms     サービスコール [終了]    sns_ctx (ercd = 0x00000000)            serial_ctl_por (serial.c:390)
0 ms     サービスコール [開始]    sta_cyc (0x00000002)                   main_task (sample1.c:628)
0 ms     サービスコール [終了]    sta_cyc (ercd = 0x00000000)            main_task (sample1.c:628)
0 ms     サービスコール [開始]    sta_cyc (0x00000001)                   main_task (sample1.c:631)
0 ms     サービスコール [終了]    sta_cyc (ercd = 0x00000000)            main_task (sample1.c:631)
0 ms     サービスコール [開始]    sns_dpn 0                              serial_rea_dat (serial.c:357
0 ms     サービスコール [終了]    sns_dpn (ercd = 0x00000000)            serial_rea_dat (serial.c:357
0 ms     サービスコール [開始]    wai_sem (0x00000001)                   serial_rea_dat (serial.c:372
0 ms     タスク状態変化           Tsk:1 <Waiting : Semaphore (id=1)>
0 ms     タスクディスパッチ [開始] Tsk: 1 (Task context)
0 ms     タスクディスパッチ [終了] Tsk: 3
0 ms     サービスコール [開始]    slp_tsk 0                              heater_task (sample1.c:317)
0 ms     タスク状態変化           Tsk:3 <Waiting : Sleep>
0 ms     タスクディスパッチ [開始] Tsk: 3 (Task context)
0 ms     タスクディスパッチ [終了] Tsk: 5
0 ms     サービスコール [開始]    rcv_dtq (0x00000002, 0x015bfec8)       button_reaction_task (sample1.c:275)
0 ms     タスク状態変化           Tsk:5 <Waiting : Sleep>
0 ms     タスクディスパッチ [開始] Tsk: 5 (Task context)
0 ms     タスクディスパッチ [終了] Tsk: 4
0 ms     サービスコール [開始]    slp_tsk 0                              temperature_update_task (sample1.c:510)
0 ms     タスク状態変化           Tsk:4 <Waiting : Sleep>
0 ms     タスクディスパッチ [開始] Tsk: 4 (Task context)
0 ms     タスクディスパッチ [終了] Tsk: 6
0 ms     サービスコール [開始]    slp_tsk 0                              kitchen_timer_task (sample1.c:546)
0 ms     タスク状態変化           Tsk:6 <Waiting : Sleep>
0 ms     タスクディスパッチ [開始] Tsk: 6 (Task context)
0 ms     割込みハンドラ [開始]    4
0 ms     サービスコール [開始]    isig_tim 0                             timer_handler (timer.c:63)
100 ms   サービスコール [終了]    isig_tim (ercd = 0x00000000)           timer_handler (timer.c:63)
100 ms   割込みハンドラ [終了]    4
100 ms   割込みハンドラ [開始]    3
100 ms   割込みハンドラ [終了]    3
100 ms   割込みハンドラ [開始]    4
100 ms   サービスコール [開始]    isig_tim 0                             timer_handler (timer.c:63)
```

```
start_sync_ack_i   1
start_sync_stb_o   0
start_sync_we_o    0
end_sync_stb_i     1
end_sync_ack_o     0
in_buf_adr_o       ---------
in_buf_dat_i       000000000000   0000000000000000
out_buf_stb_o      0
out_buf_we_o       0
out_buf_adr_o      ---------
out_buf_dat_o      ---------------
out_buf_dat_i      000000000000   0000000000000000

Now        4120980 ns
Cursor 1   2411620 ns
Cursor 2   2396500 ns

2400 us   2405 us   2410 us
15120 ns   2411620 ns   2396500 ns

2395541 ns to 2413826 ns
```

Case Study 2: MPEG Encoder/Decoder on Multiprocessors

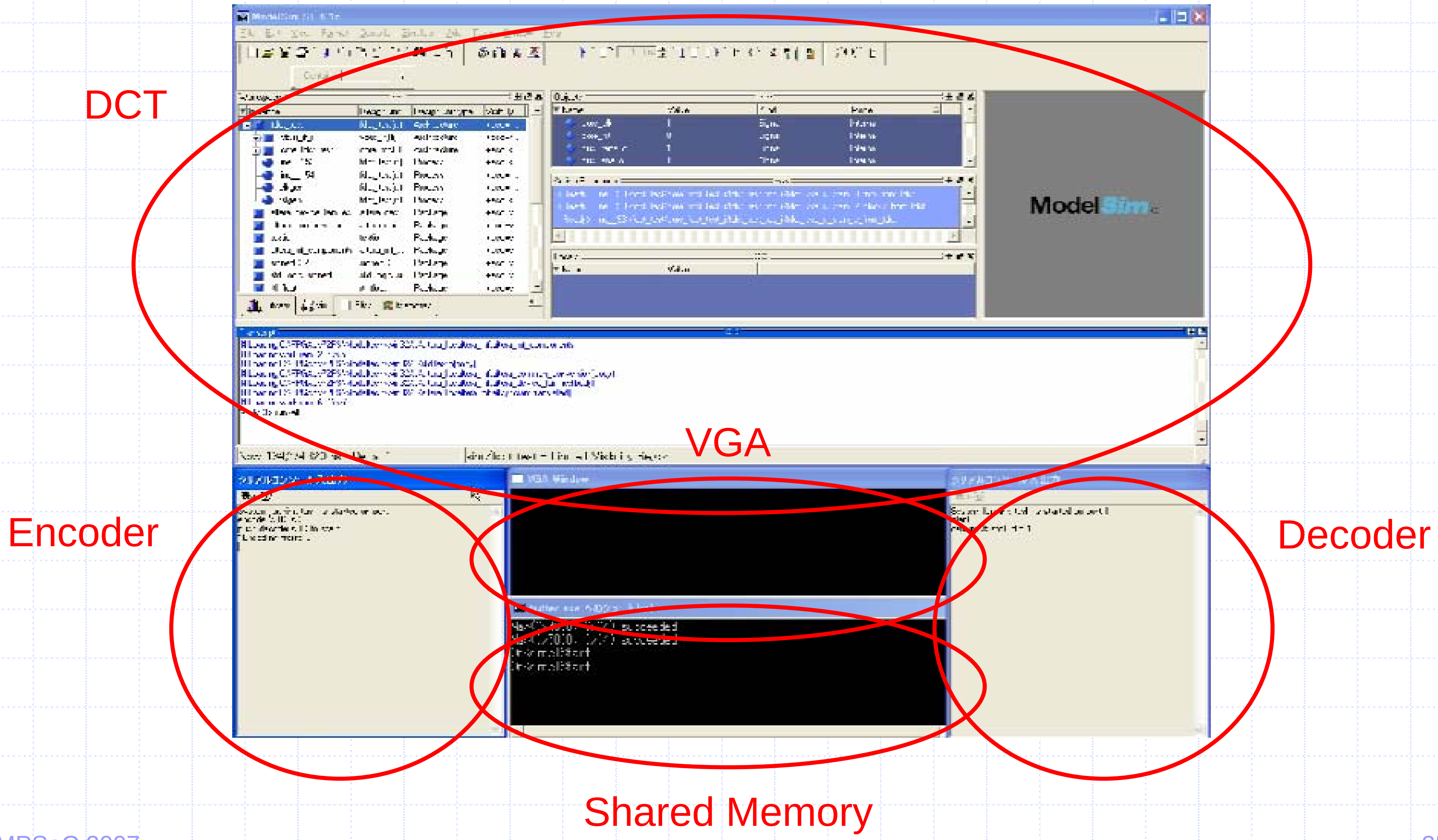


Flow of MPEG encoder/decoder



System organization

Demo Video



Summary

- RTOC-Centric Design and Validation Methodology
- RTOS-Centric Hardware/Software Cosimulator
  - Complete simulation model of ITRON RTOS
  - Native, hence fast, execution of software
  - Cosimulation with hardware designs in HDL
  - Cosimulation with hardware models in SystemC/C++/C
  - Covalidation with FPGA
- Future Work
  - Improvement of timing accuracy
  - Support of other OS (e.g., Linux, VxWorks, OSEK, etc.)
  - Covalidation with software (CPU) on FPGA and HDL simulator on a host computer.
- The RTOS simulation model (with limited functionality for cosimulation) is available as a part of the TOPPERS/JSP kernel package from http://www.toppers.jp/