HiRTOS: A Multi-Core RTOS written in SPARK Ada

J. Germán Rivera

(jgrivera67@gmail.com)

Topics

- HiRTOS Design Overview
- HiRTOS Thread Scheduler
- HiRTOS Separation Kernel
- Porting HiRTOS to a New Platform
- Future Work

What is HiRTOS?

- HiRTOS: *High-Integrity* RTOS
  - A small real-time operating system kernel and separation kernel written in SPARK Ada
  - HiRTOS targets deeply embedded software applications that run in small single-core and multi-core embedded platforms
  - HiRTOS won the “2023 Alire Embedded Crate of the Year” award
- HiRTOS is available for download as two Alire crates:
  - HiRTOS kernel at https://alire.ada.dev/crates/hirtos
  - HiRTOS separation kernel at https://alire.ada.dev/crates/hirtos_separation_kernel
- HiRTOS code is available in GitHub at https://github.com/jgrivera67/HiRTOS

HiRTOS Design Overview

- In a multi-core platform, there is one HiRTOS instance per CPU.
  - Each HiRTOS instance is independent of each other. No resources are shared between CPUs.
  - No communication/synchronization between CPU cores is supported by HiRTOS.
- Mutexes and condition variables are the only synchronization primitives in HiRTOS
  - Other synchronization primitives such as semaphores, event flags and message queues can be implemented on top of mutexes and condition variables.
- HiRTOS mutexes support both priority inheritance and priority ceiling protocols.

HiRTOS Design Overview (2)

- HiRTOS condition variables can also be waited on while having interrupts disabled, not just while holding a mutex.
  - This prevents missing ”thread wakeups”, when signaling condition variables from interrupt service routines. (Semaphore not needed)
- HiRTOS atomic levels can be used to disable the thread scheduler or to disable interrupts at and below a given priority or to disable all interrupts.
- Threads are bound to the CPU core in which they were created, for the lifetime of the thread.
  - No thread migration between CPU cores is supported

HiRTOS Design Overview (3)

- All RTOS objects such as threads, mutexes and condition variables are allocated internally by HiRTOS from statically allocated internal object arrays
  - Once allocated, RTOS objects cannot be deallocated
  - RTOS object handles provided to application code are just indices into these internal object arrays.
- All application threads run in unprivileged mode by default.
- In unprivileged mode, a thread can only access its own stack.
- To access global variables or MMIO space, application threads must explicitly request permission to HiRTOS.

HiRTOS Design Overview (4)

- HiRTOS pointer-less data structures

```
type HiRTOS_Cpu_Instance_Type is limited record
   Initialized            : Boolean                    := False;
   Last_Chance_Handler_Running : Boolean               := False;
   Tick_Timer_Thread_Work_Requested : Boolean          := False with Atomic;
   Cpu_Id                 : Cpu_Core_Id_Type;
   Thread_Scheduler_State : Thread_Scheduler_State_Type :=
    Thread_Scheduler_Stopped;
   Current_Atomic_Level       : Atomic_Level_Type := Atomic_Level_None;
   Current_Cpu_Execution_Mode : Cpu_Execution_Mode_Type := Cpu_Executing_Reset_Handler;
   Current_Thread_Id             : Thread_Id_Type := Invalid_Thread_Id;
   Timer_Ticks_Since_Boot        : Timer_Ticks_Count_Type := 0;
   Idle_Thread_Id                : Thread_Id_Type := Invalid_Thread_Id;
   Tick_Timer_Thread_Id          : Thread_Id_Type := Invalid_Thread_Id;
   Interrupt_Stack_Base_Address  : System.Address := System.Null_Address;
   Interrupt_Stack_End_Address   : System.Address := System.Null_Address;
   Next_Free_Thread_Id  : Atomic_Counter_Type :=
      Atomic_Counter_Initializer (Cpu_Register_Type (Thread_Id_Type'First));
   Next_Free_Mutex_Id   : Atomic_Counter_Type :=
      Atomic_Counter_Initializer (Cpu_Register_Type (Mutex_Id_Type'First));
   Next_Free_Condvar_Id : Atomic_Counter_Type :=
      Atomic_Counter_Initializer (Cpu_Register_Type (Condvar_Id_Type'First));
   Next_Free_Timer_Id   : Atomic_Counter_Type :=
      Atomic_Counter_Initializer (Cpu_Register_Type (Timer_Id_Type'First));
   Interrupt_Nesting_Level_Stack : Interrupt_Nesting_Level_Stack_Type;
   Runnable_Threads_Queue        : Thread_Priority_Queue_Type;
   Timer_Wheel                   : Timer_Wheel_Type;
   Thread_Instances     : Thread_Array_Type;
   Mutex_Instances      : Mutex_Array_Type;
   Condvar_Instances    : Condvar_Array_Type;
   Timer_Instances      : Timer_Array_Type;
   Thread_Queues_Nodes  : Thread_Queue_Package.List_Nodes_Type;
   Mutex_Lists_Nodes    : Mutex_List_Package.List_Nodes_Type;
   Timer_Lists_Nodes    : Timer_List_Package.List_Nodes_Type;
end record with
   Alignment => HiRTOS_Cpu_Arch_Parameters.Cache_Line_Size_Bytes;
```

Arrays of RTOS objects

Arrays of nodes for linked lists of the corresponding type

```
package Thread_Queue_Package is new Generic_Linked_List
  (List_Id_Type    => Thread_Priority_Type,
   Null_List_Id    => Invalid_Thread_Priority,
   Element_Id_Type => Thread_Id_Type,
   Null_Element_Id => Invalid_Thread_Id);

package Mutex_List_Package is new Generic_Linked_List
  (List_Id_Type    => Thread_Id_Type,
   Null_List_Id    => Invalid_Thread_Id,
   Element_Id_Type => Mutex_Id_Type,
   Null_Element_Id => Invalid_Mutex_Id);

type Per_Priority_Thread_Queues_Array_Type is
  array
    (Valid_Thread_Priority_Type) of Thread_Queue_Package.List_Anchor_Type;

type Boolean_Bit_Map_Type is array (Valid_Thread_Priority_Type) of Boolean
  with Component_Size => 1, Size => HiRTOS_Cpu_Arch_Interface.Cpu_Register_Type'Size;

type Thread_Priority_Queue_Type is record
  Non_Empty_Thread_Queues_Map : Boolean_Bit_Map_Type := [others => False];
  Thread_Queues_Array : Per_Priority_Thread_Queues_Array_Type;
end record;
```

pointer-less linked lists

pointer-less linked lists

```
package Timer_List_Package is new Generic_Linked_List
  (List_Id_Type        => Timer_Wheel_Spoke_Index_Type,
   Null_List_Id        => Invalid_Timer_Wheel_Spoke_Index,
   Element_Id_Type     => Timer_Id_Type,
   Null_Element_Id     => Invalid_Timer_Id);
```

pointer-less linked lists

HiRTOS Design Overview (5)

- HiRTOS pointer-less linked lists

```
generic
   type List_Id_Type is private;
   Null_List_Id : List_Id_Type;
   type Element_Id_Type is range <>;
   Null_Element_Id : Element_Id_Type;
package Generic_Linked_List with
  SPARK_Mode => On
is
   type List_Node_Type is limited private;

   type List_Anchor_Type is private;

   type List_Nodes_Type is limited private;
```

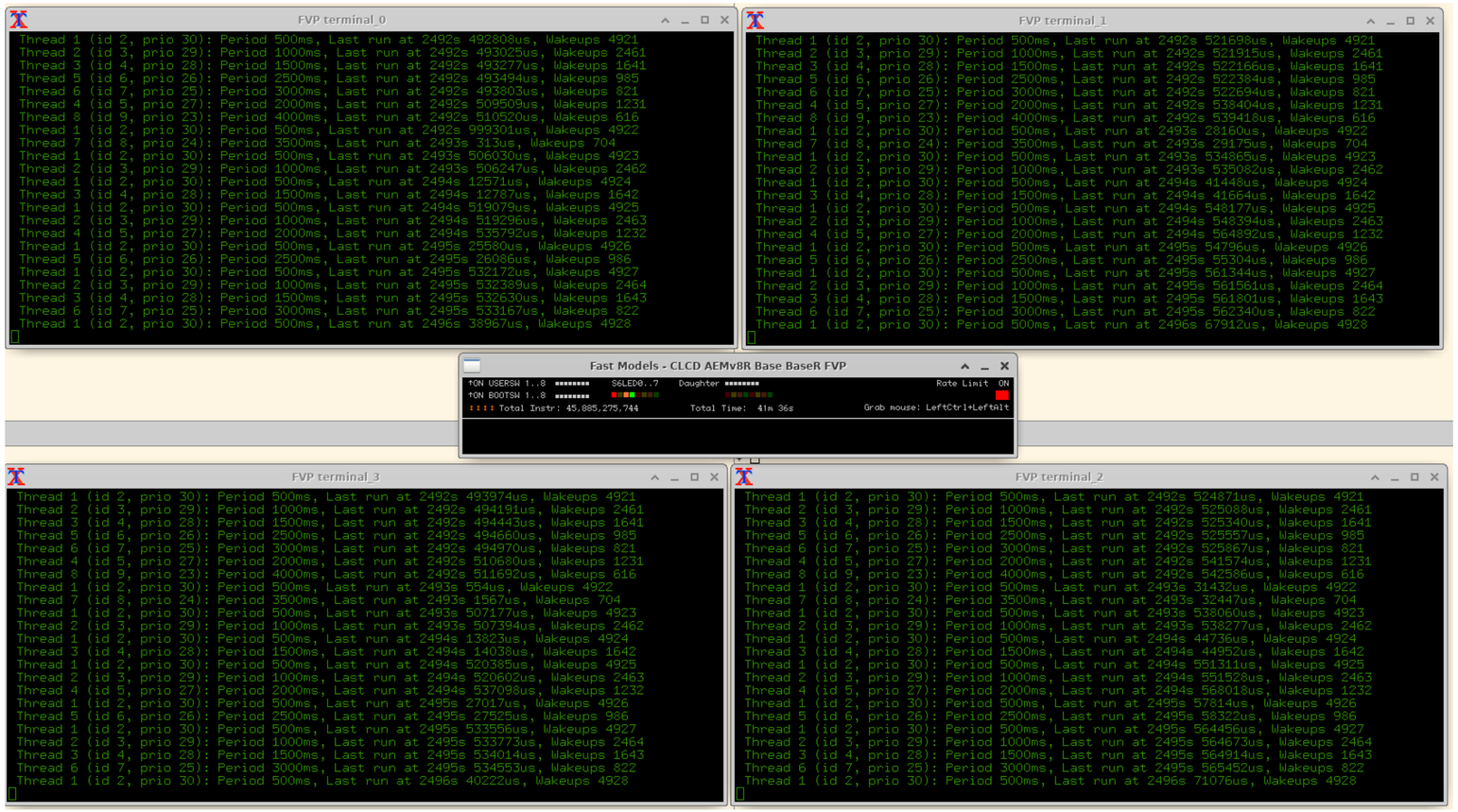
Backing storage for all the linked lists (pairs of "next/prev" injective functions) of a given type

```
private

   type List_Anchor_Type is record
      List_Id : List_Id_Type := Null_List_Id;
      Head    : Element_Id_Type := Null_Element_Id;
      Tail    : Element_Id_Type := Null_Element_Id;
      Length  : Natural := 0;
   end record with
     Type_Invariant =>
      (if List_Anchor_Type.Length = 0 then
         (List_Anchor_Type.Head = Null_Element_Id
          and then List_Anchor_Type.Tail = Null_Element_Id)
       else
         (List_Anchor_Type.Head /= Null_Element_Id
          and then List_Anchor_Type.Tail /= Null_Element_Id
          and then
          (if List_Anchor_Type.Head = List_Anchor_Type.Tail then
             List_Anchor_Type.Length = 1
           else List_Anchor_Type.Length > 1)));
```

```
type List_Node_Type is limited record
   Next_Element_Id    : Element_Id_Type := Null_Element_Id;
   Prev_Element_Id    : Element_Id_Type := Null_Element_Id;
   Containing_List_Id : List_Id_Type := Null_List_Id;
end record;

pragma Compile_Time_Error
  (Null_Element_Id /= Element_Id_Type'Last,
   "Null_element_Id has the wrong value");

subtype Valid_Element_Id_Type is
  Element_Id_Type range Element_Id_Type'First .. Element_Id_Type'Last - 1;


--
--  NOTE: The same element cannot be in more than one list. So,
--  the maximal set of nodes that we need is `Valid_Element_Id_Type`
--
type List_Nodes_Type is array (Valid_Element_Id_Type) of List_Node_Type;
```

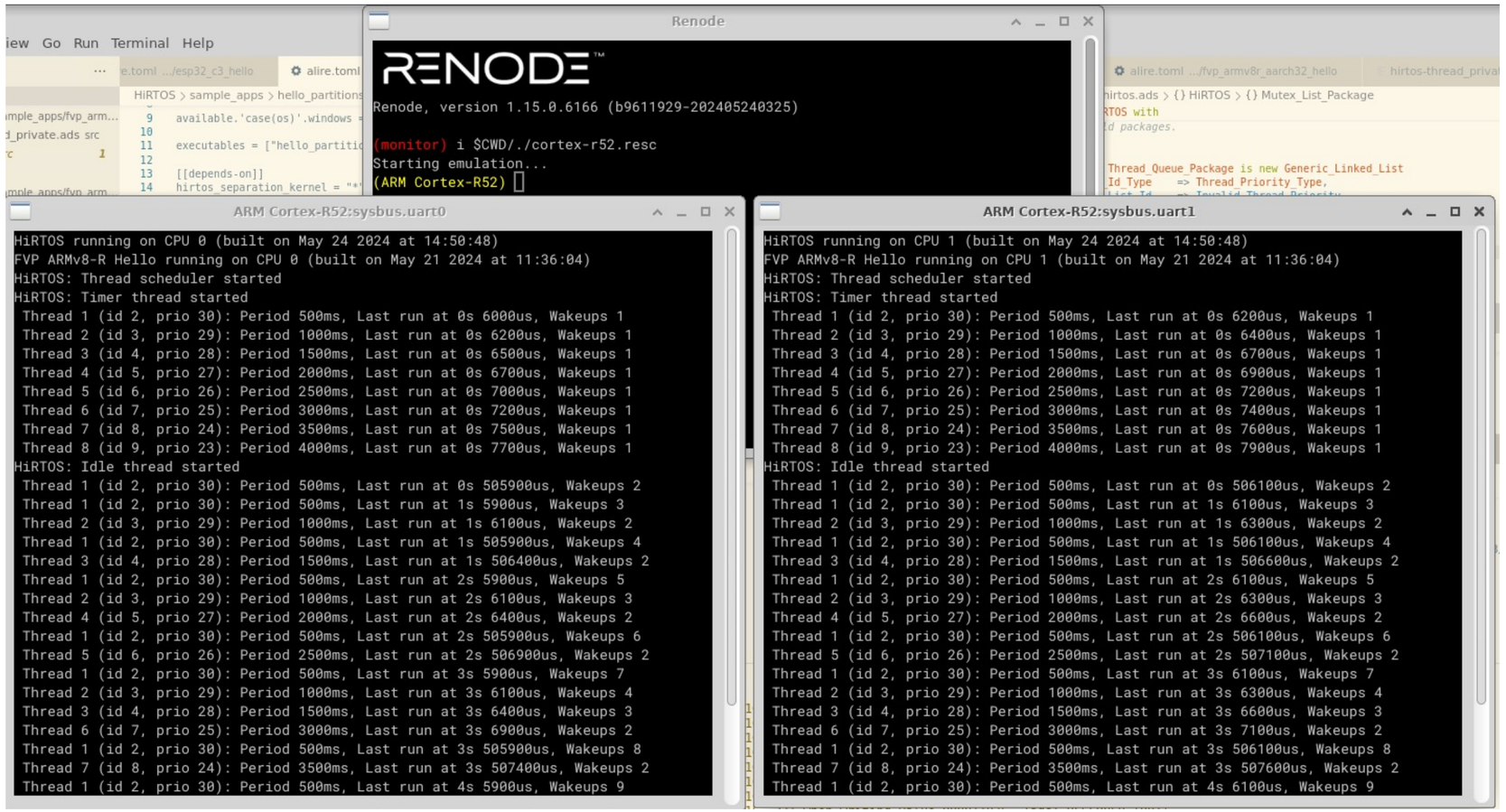
HiRTOS Supported Platforms

- ARM Fixed Virtual Platform (FVP) Simulator for ARMv8-R (ARM Cortex-R52 processor with 4 cores)



HiRTOS Supported Platforms (2)

- ARM Cortex-R52-SMP Renode Simulator configuration (2 cores)



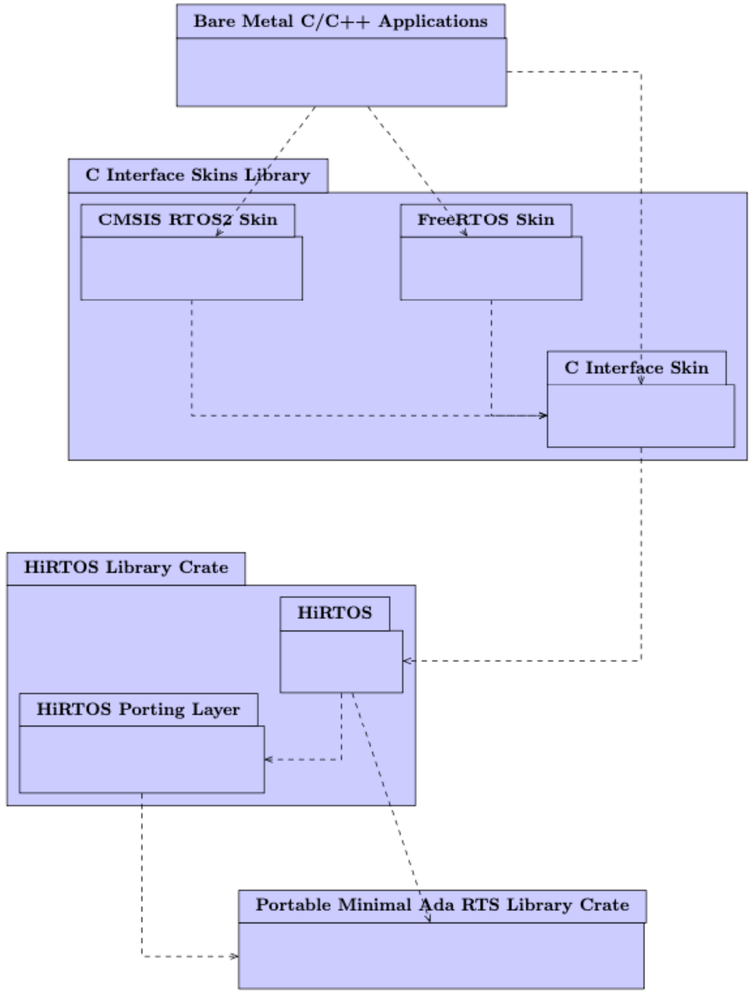
HiRTOS Supported Platforms (3)

- RISC-V-based ESP32-C3 board (single-core)

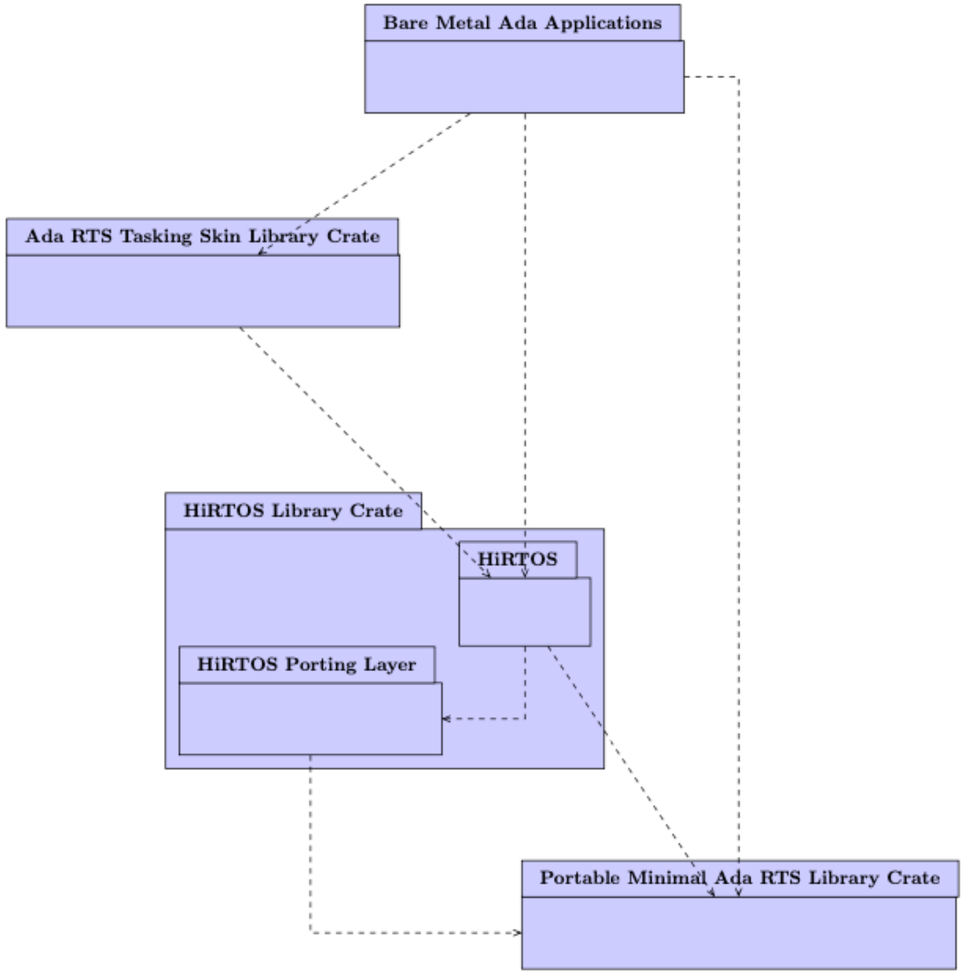
```
0000034ABF5D0E50 Thread 2 (id 3, prio 29): Period 1000ms, Last run at 1005h 1573s 12470us, Wakeups 3619574
0000034ABF649030 Thread 1 (id 2, prio 30): Period 500ms, Last run at 1005h 1573s 504468us, Wakeups 7239148
0000034ABF64CEB0 Thread 3 (id 4, prio 28): Period 1500ms, Last run at 1005h 1573s 520470us, Wakeups 2413050
0000034ABF6C3150 Thread 1 (id 2, prio 30): Period 500ms, Last run at 1005h 1574s 4468us, Wakeups 7239149
0000034ABF6C5090 Thread 2 (id 3, prio 29): Period 1000ms, Last run at 1005h 1574s 12471us, Wakeups 3619575
0000034ABF6C8F10 Thread 4 (id 5, prio 27): Period 2000ms, Last run at 1005h 1574s 28471us, Wakeups 1809788
0000034ABF6CF0B8 Thread 7 (id 8, prio 24): Period 3500ms, Last run at 1005h 1574s 53470us, Wakeups 1034165
0000034ABF73D270 Thread 1 (id 2, prio 30): Period 500ms, Last run at 1005h 1574s 504468us, Wakeups 7239150
0000034ABF7B7390 Thread 1 (id 2, prio 30): Period 500ms, Last run at 1005h 1575s 4471us, Wakeups 7239151
0000034ABF7B92D0 Thread 2 (id 3, prio 29): Period 1000ms, Last run at 1005h 1575s 12471us, Wakeups 3619576
0000034ABF7BB210 Thread 3 (id 4, prio 28): Period 1500ms, Last run at 1005h 1575s 20471us, Wakeups 2413051
0000034ABF7BF090 Thread 5 (id 6, prio 26): Period 2500ms, Last run at 1005h 1575s 36471us, Wakeups 1447831
0000034ABF7C0FD0 Thread 6 (id 7, prio 25): Period 3000ms, Last run at 1005h 1575s 44470us, Wakeups 1206526
0000034ABF8314B0 Thread 1 (id 2, prio 30): Period 500ms, Last run at 1005h 1575s 504468us, Wakeups 7239152
0000034ABF8AB5D0 Thread 1 (id 2, prio 30): Period 500ms, Last run at 1005h 1576s 4468us, Wakeups 7239153
0000034ABF8AD510 Thread 2 (id 3, prio 29): Period 1000ms, Last run at 1005h 1576s 12470us, Wakeups 3619577
0000034ABF8B1390 Thread 4 (id 5, prio 27): Period 2000ms, Last run at 1005h 1576s 28471us, Wakeups 1809789
0000034ABF8C1178 Thread 8 (id 9, prio 23): Period 4000ms, Last run at 1005h 1576s 93470us, Wakeups 904895
0000034ABF9256F0 Thread 1 (id 2, prio 30): Period 500ms, Last run at 1005h 1576s 504468us, Wakeups 7239154
0000034ABF929570 Thread 3 (id 4, prio 28): Period 1500ms, Last run at 1005h 1576s 520470us, Wakeups 2413052
0000034ABF99F810 Thread 1 (id 2, prio 30): Period 500ms, Last run at 1005h 1577s 4469us, Wakeups 7239155
0000034ABF9A1750 Thread 2 (id 3, prio 29): Period 1000ms, Last run at 1005h 1577s 12470us, Wakeups 3619578
0000034ABFA19930 Thread 1 (id 2, prio 30): Period 500ms, Last run at 1005h 1577s 504469us, Wakeups 7239156
0000034ABFA21630 Thread 5 (id 6, prio 26): Period 2500ms, Last run at 1005h 1577s 536470us, Wakeups 1447832
0000034ABFA25898 Thread 7 (id 8, prio 24): Period 3500ms, Last run at 1005h 1577s 553470us, Wakeups 1034166
0000034ABFA93A50 Thread 1 (id 2, prio 30): Period 500ms, Last run at 1005h 1578s 4468us, Wakeups 7239157
0000034ABFA95990 Thread 2 (id 3, prio 29): Period 1000ms, Last run at 1005h 1578s 12471us, Wakeups 3619579
0000034ABFA978D0 Thread 3 (id 4, prio 28): Period 1500ms, Last run at 1005h 1578s 20473us, Wakeups 2413053
0000034ABFA99810 Thread 4 (id 5, prio 27): Period 2000ms, Last run at 1005h 1578s 28470us, Wakeups 1809790
0000034ABFA9D690 Thread 6 (id 7, prio 25): Period 3000ms, Last run at 1005h 1578s 44472us, Wakeups 1206527
0000034ABFB0DB70 Thread 1 (id 2, prio 30): Period 500ms, Last run at 1005h 1578s 504468us, Wakeups 7239158
```

HiRTOS Usage Architecture Vision

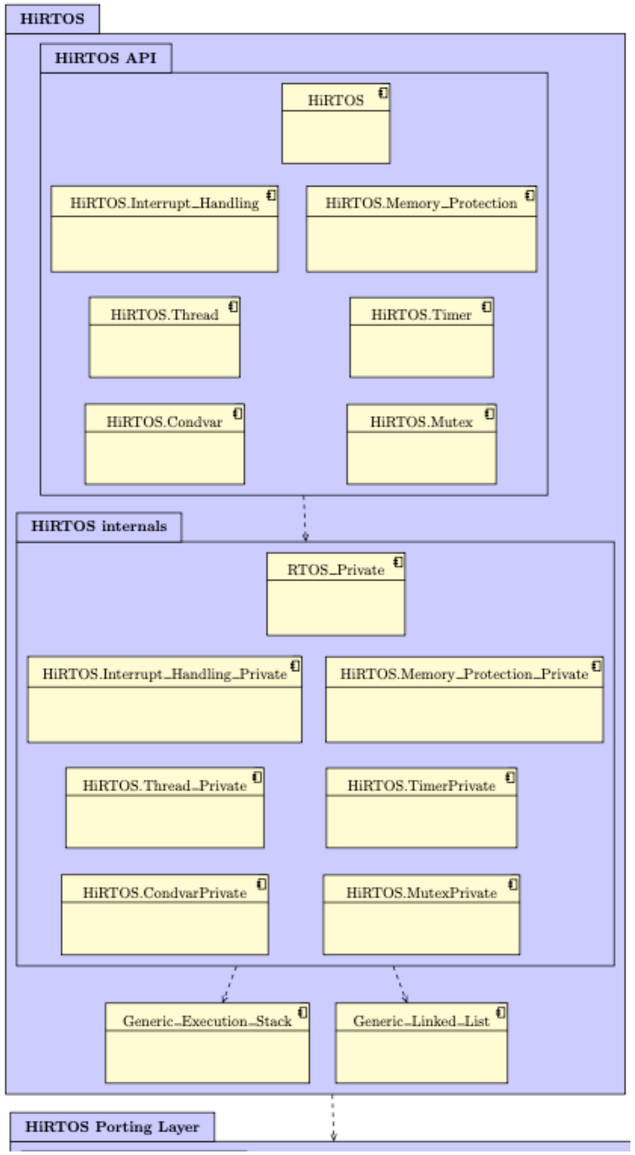
- Using HiRTOS from C/C++

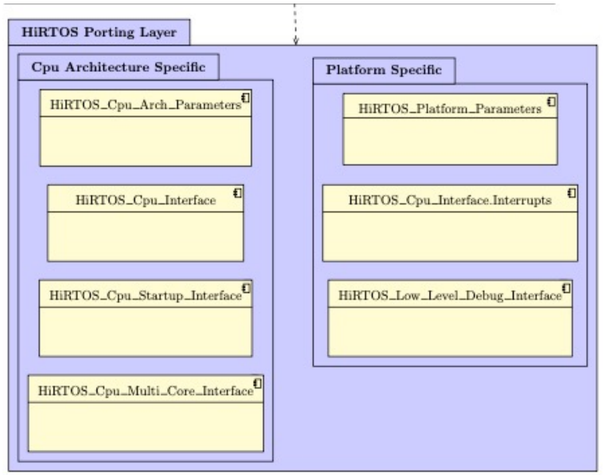


- Using HiRTOS from Ada



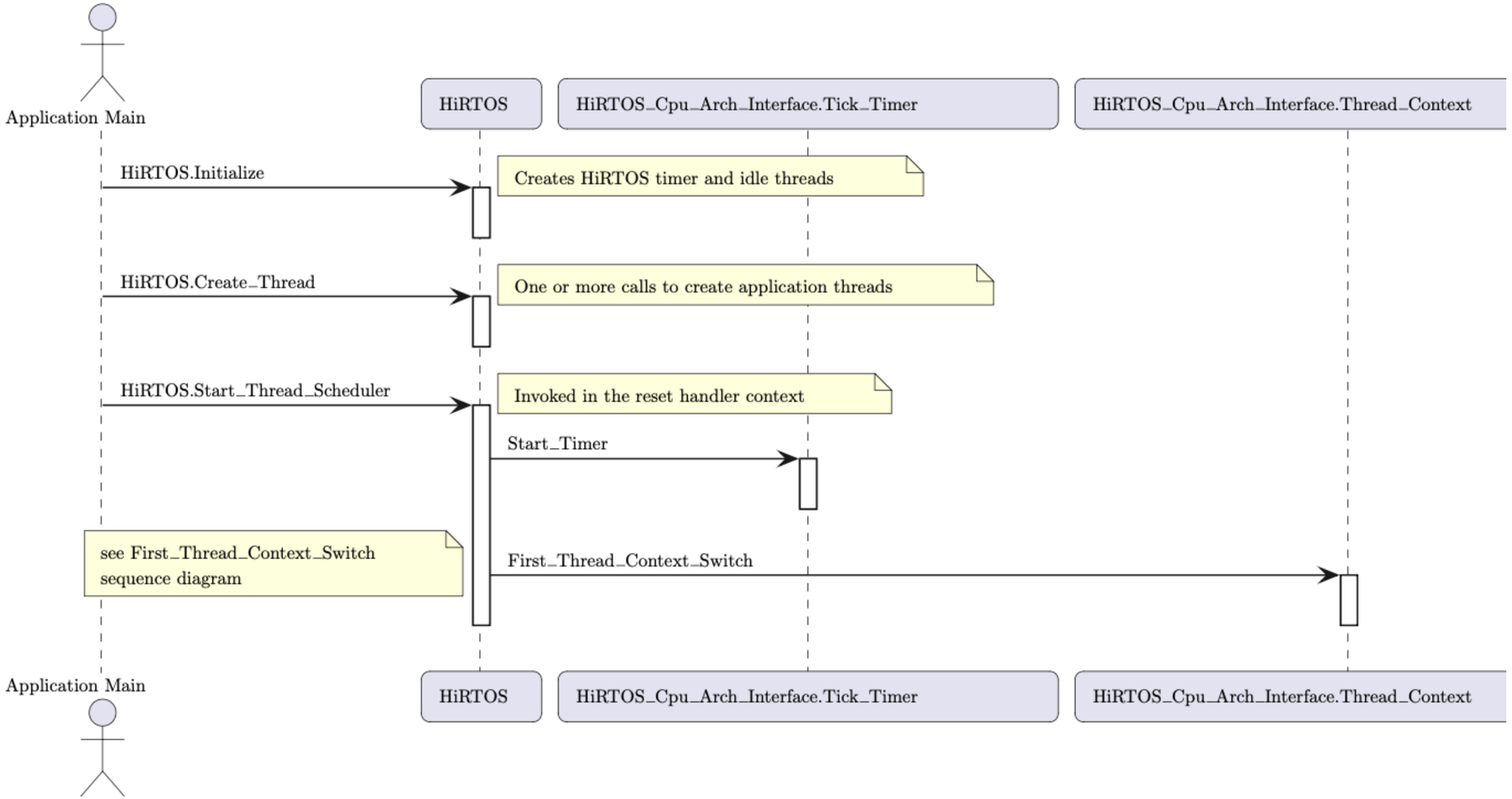
HiRTOS Code Architecture





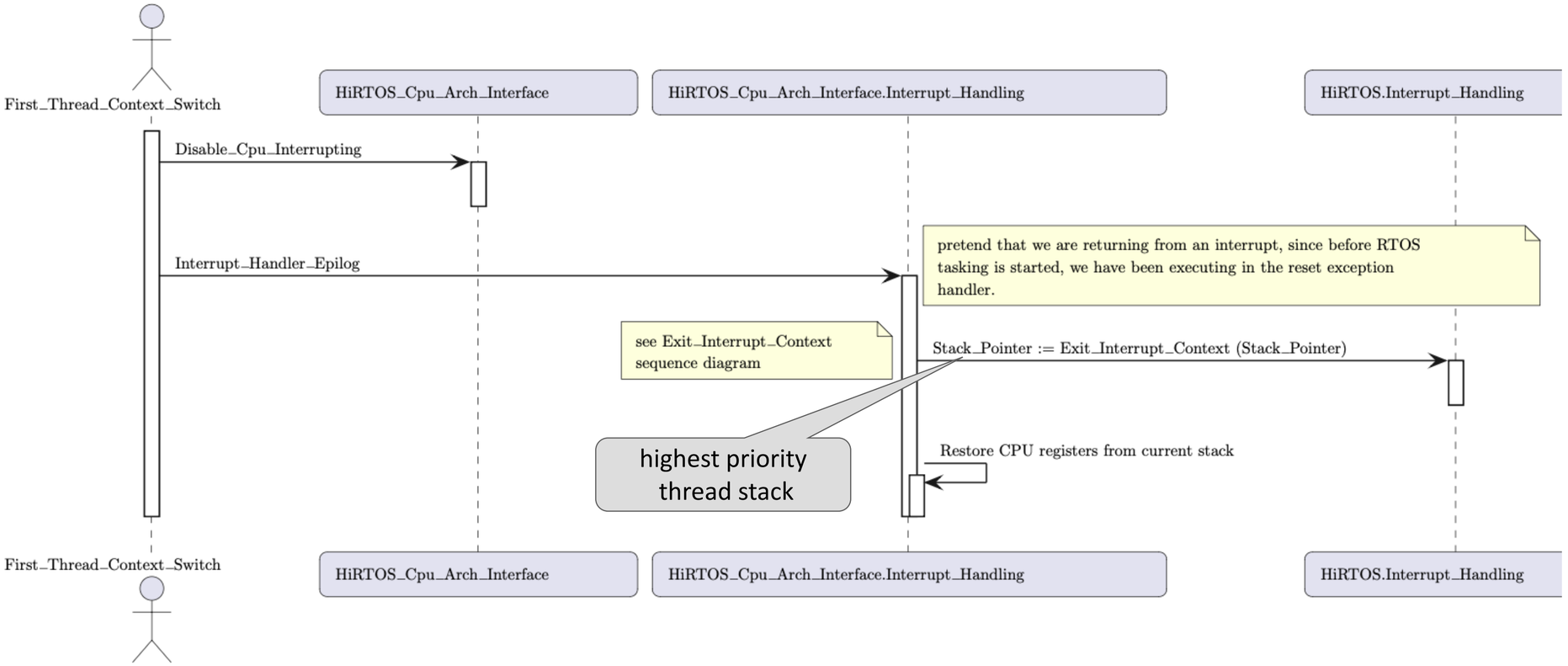
HiRTOS Thread Scheduler

- HIRTOS Thread Scheduler Initialization



HiRTOS Thread Scheduler (2)

- HIRTOS Thread Scheduler Initialization (cont.)

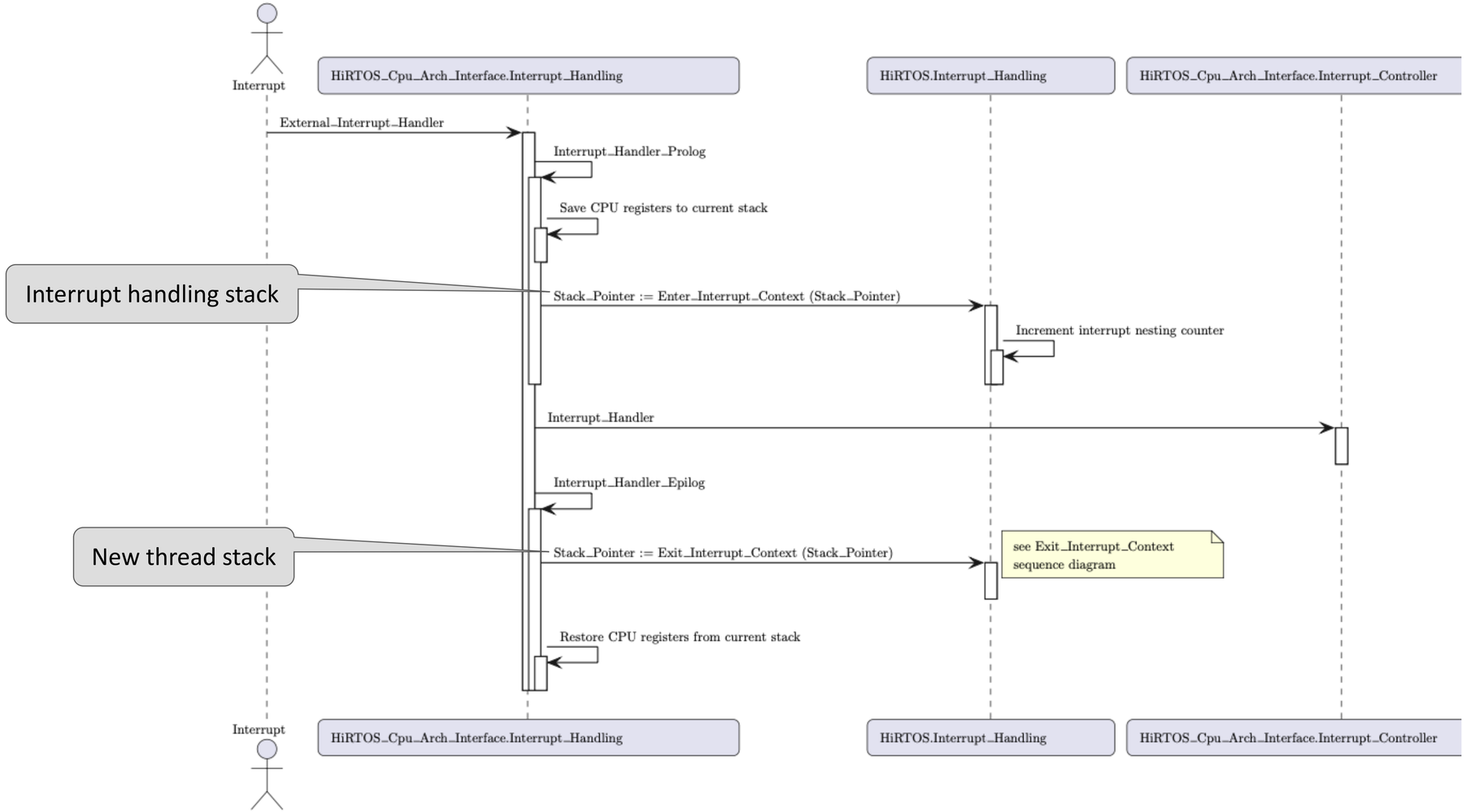


HiRTOS Thread Scheduler (3)

- Asynchronous Thread Context Switch
  - In HiRTOS, thread preemption is implemented by invoking the thread scheduler on the exit path of an interrupt handler.
  - When an interrupt fires while a thread is running, the executing thread's CPU context is saved on thread's stack by the interrupt handler prolog.
  - Then, before calling the actual interrupt handler, the stack is switched to the interrupt handling stack. After the interrupt handler returns, the interrupt handler epilog invokes the HiRTOS thread scheduler, to select the highest priority runnable thread.
  - Then, if the newly selected thread is different from the one that was running before the interrupt, the extended context of the old thread is saved and the extended context of the new thread is restored.

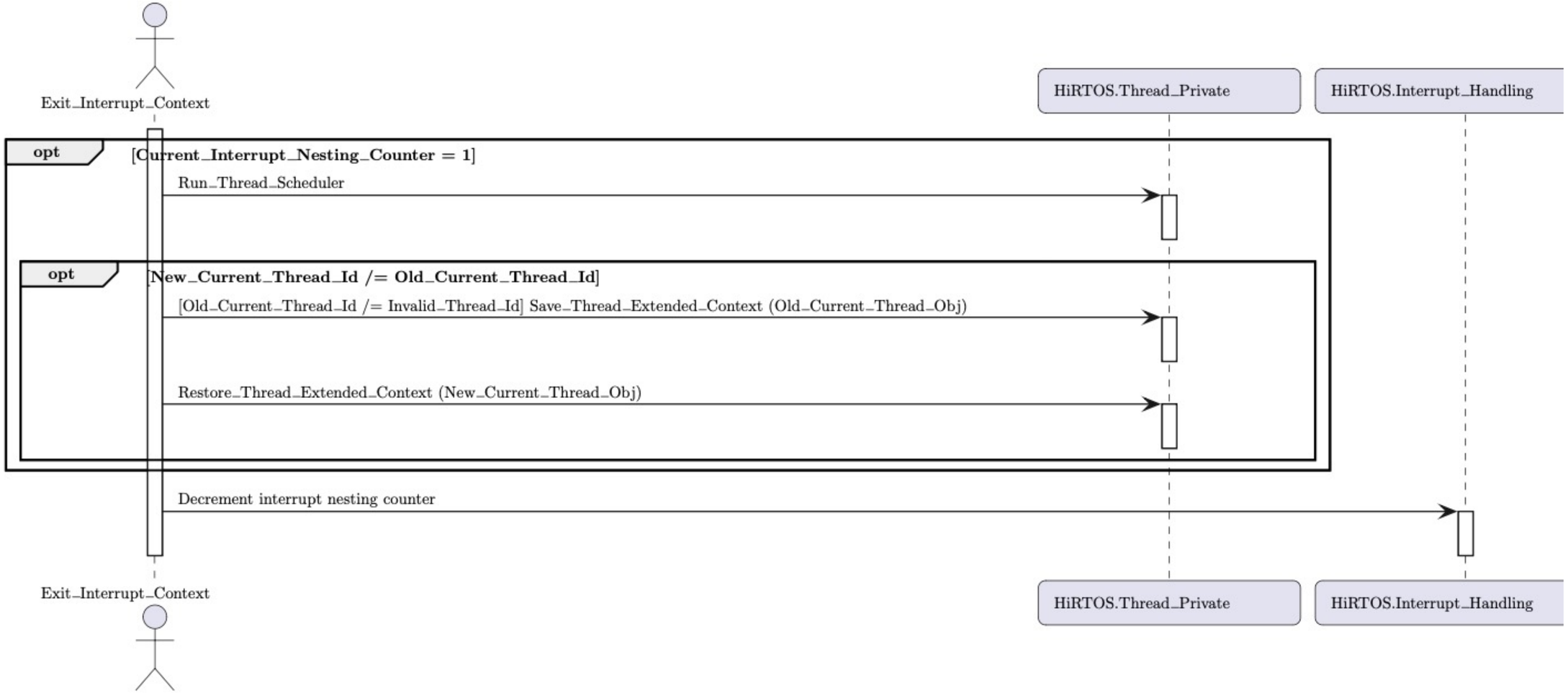
HiRTOS Thread Scheduler (4)

- Asynchronous Thread Context Switch (cont.)



HiRTOS Thread Scheduler (5)

- Asynchronous Thread Context Switch (cont.)

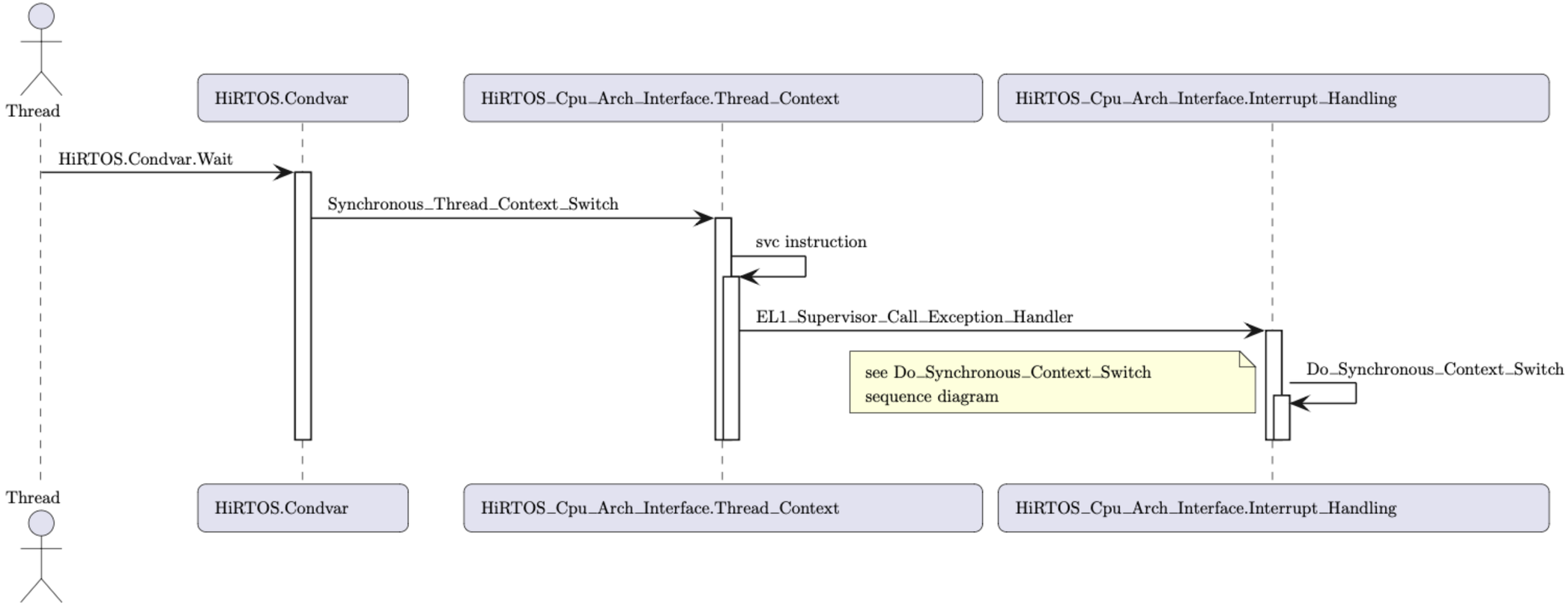


HiRTOS Thread Scheduler (6)

- In HiRTOS, a synchronous thread context switch occurs when:
  - A thread calls HiRTOS.Condvar.Wait
  - A thread calls HiRTOS.Condvar.Signal or HiRTOS.Condvar.Broadcast, and there are threads waiting on the condition variable
  - A thread calls HiRTOS.Mutex.Acquire and the mutex is not available
  - A thread calls HiRTOs.Mutex.Release and there are threads waiting to acquire the mutex
  - A thread calls HiRTOS.Thread.Thread_Delay_Until (which calls HiRTOS.Condvar.Wait)
  - A thread calls HiRTOS.Thread.Suspend_Current_Thread
  - A thread calls HiRTOS.Thread.Resume_Thread
  - A thread calls HiRTOS.Restore_Atomic_Level and the old atomic level is Atomic_Level_None

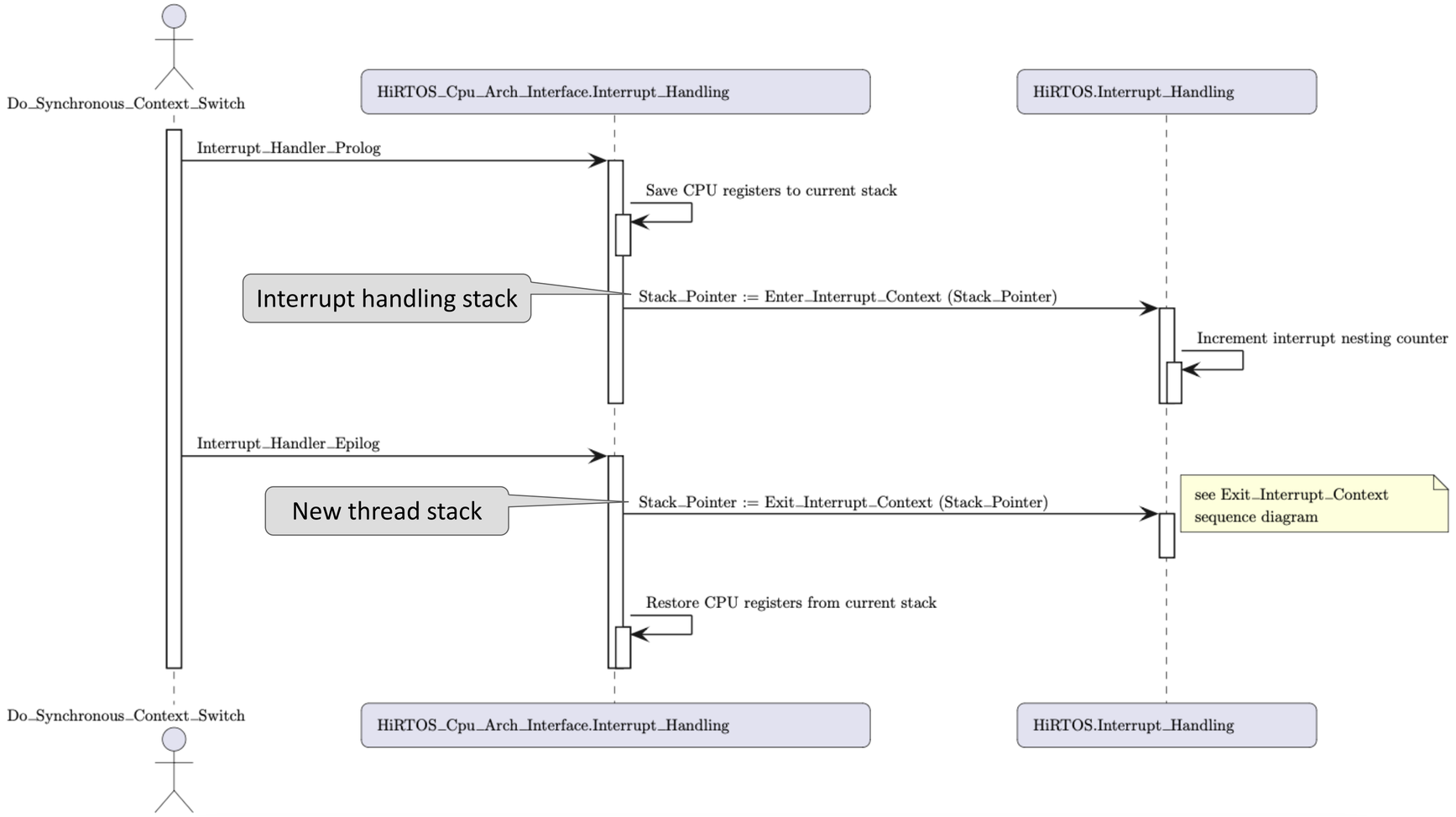
HiRTOS Thread Scheduler (7)

- Synchronous Thread Context Switch



HiRTOS Thread Scheduler (8)

- Synchronous Thread Context Switch (cont.)



HiRTOS Separation Kernel Overview

- A separation kernel can be seen as an RTOS that schedules partitions instead of threads
  - CPU needs to support hypervisor privilege mode and two-stage memory protection
- A partition is a spatial and temporal separation/isolation unit on which a bare-metal or RTOS-based application runs.
- Each partition consists of one or more address ranges covering disjoint portions of RAM and MMIO space that only that partition can access
- HiRTOS uses the hypervisor-controlled MPU to enforce isolation between partitions “address spaces”
- Each partition can only use the supervisor-controlled MPU within its own address space

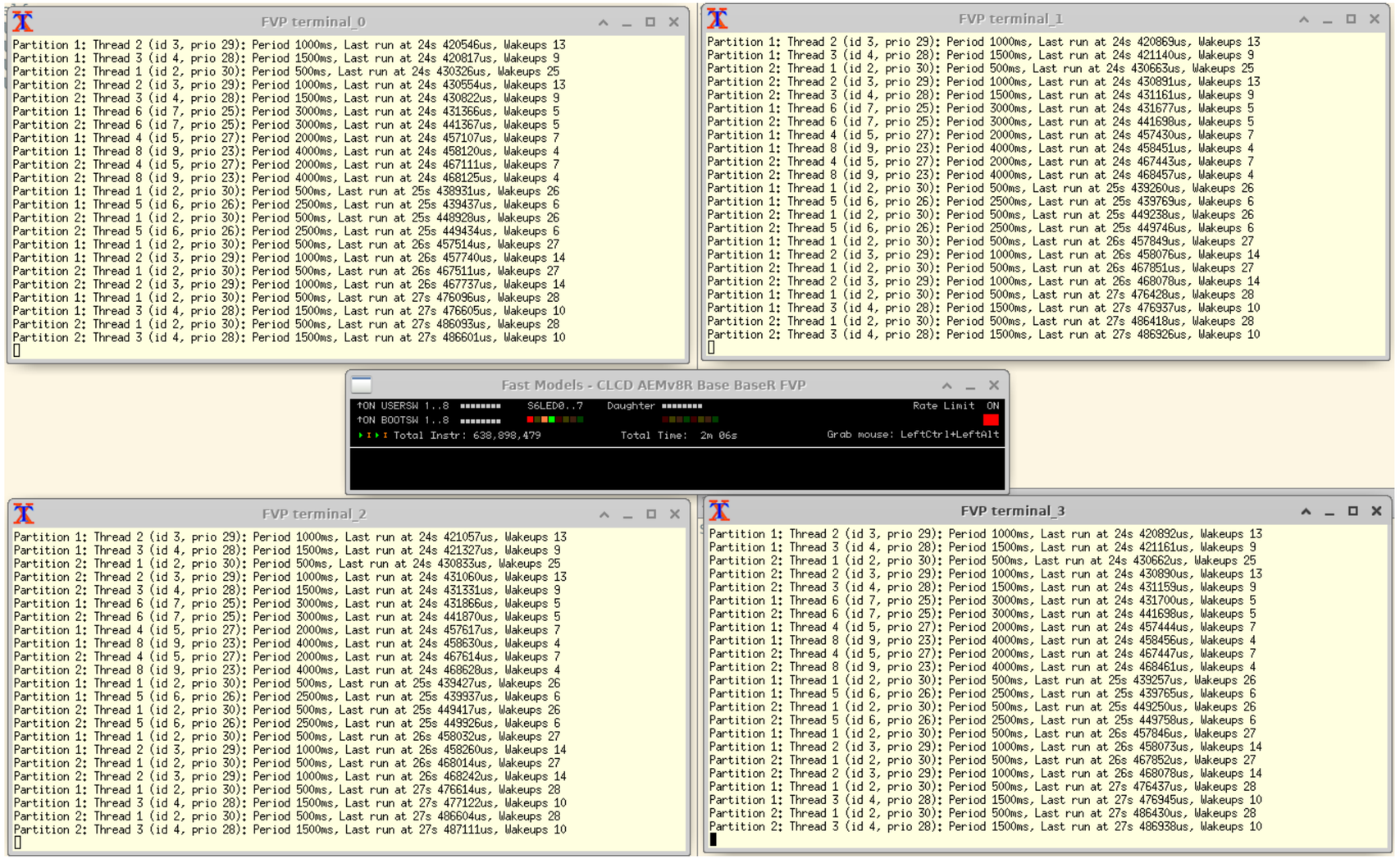
HiRTOS Separation Kernel Overview (2)

- Each partition also has its own interrupt vector table and its own set of physical interrupts.
  - Physical peripherals can be assigned to individual partitions (discrete device assignment).
  - No device virtualization is supported.
- In a multi-core platform, there is one separation kernel instance per CPU Core.
  - Each instance is independent of each other. No resources are shared.
  - The CPU core is time-sliced among the partitions running on the same separation kernel instance.
  - Partitions are bound to the CPU core in which they were created.
  - Partitions are created at boot time before starting the partition scheduler on the corresponding CPU core. Partitions cannot be destroyed or terminated.

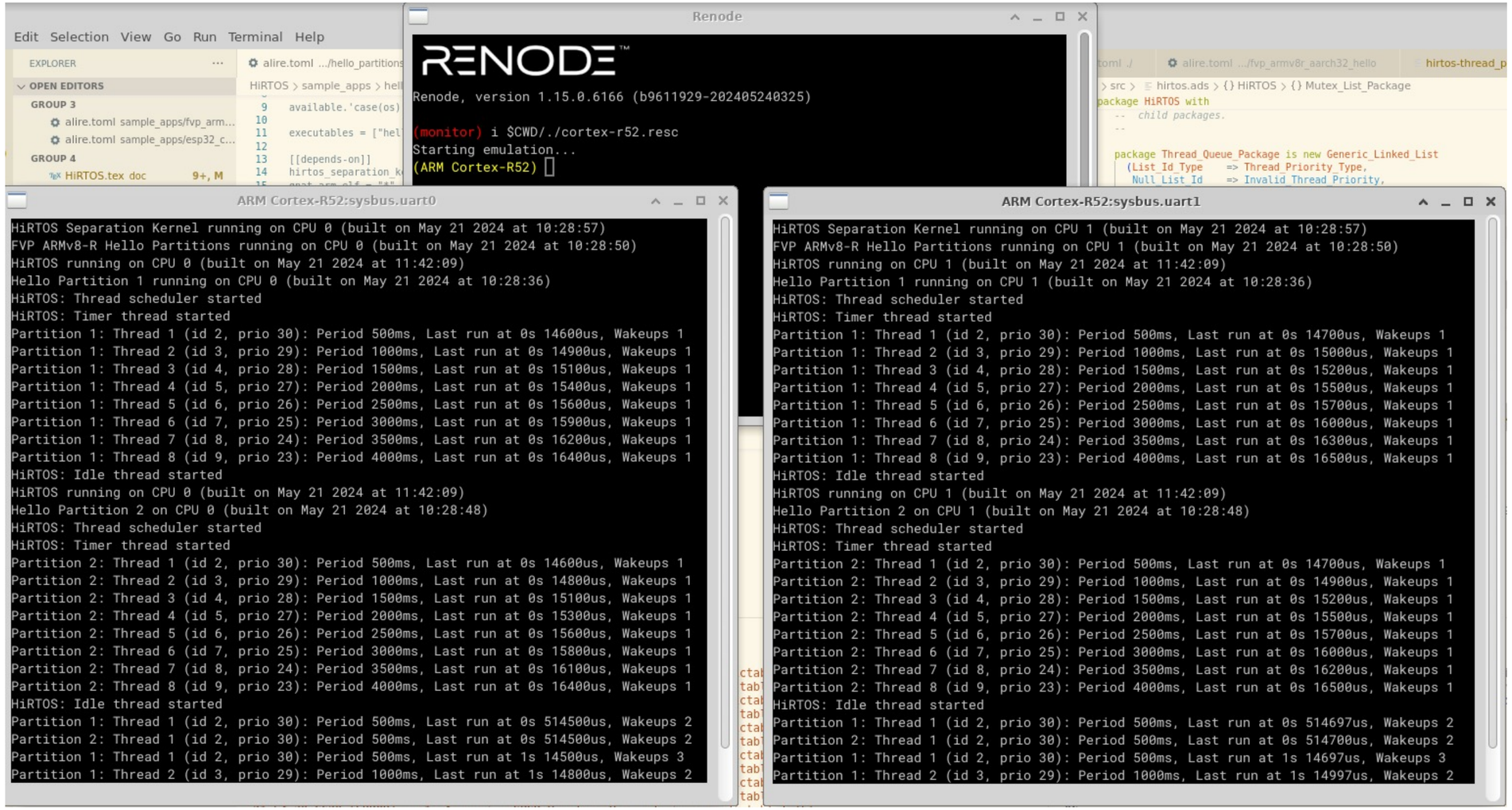
HiRTOS Separation Kernel Overview (3)

- The separation kernel code itself runs in hypervisor privilege mode.
- All partitions run at a privilege lower than hypervisor mode. Partitions can communicate with the separation kernel via hypervisor calls
- Inter-partition communication is not supported yet
  - A shared-memory-based mailbox mechanism could be provided in the future

HiRTOS Separation Kernel Running on ARM FVP Simulator for ARMv8-R



HiRTOS Separation Kernel Running on Renode Simulator Configured for ARMv8-R



Porting HiRTOS to a new platform

- Implement CPU-architecture-specific interfaces for the new CPU architecture, including:
  - Startup code
  - Interrupt handling prolog and epilog code
  - Interrupt controller driver
  - CPU-architecture-specific timer driver (if available)
  - Memory protection unit (MPU) driver
- Implement platform-specific interfaces for the new SoC or board, including:
  - UART driver
  - External timer driver (if needed)
  - Platform-specific hardware initialization

Future Work

- Change the idle thread to be a ”safety patrol” thread to check safety invariants and liveness properties during spare CPU cycles
- Add inter-core communication functionality to the HiRTOS kernel
- Add inter-partition communication functionality to the HiRTOS separation kernel
- Port HiRTOS to other embedded CPU architectures such as ARMv7-M, ARMv8-M, 64-bit ARMv8-R and 64-bit RISC-V
- Port The HiRTOS separation kernel to RISC-V platforms with Hypervisor mode
- Do Formal Verification of HiRTOS code using gnatprove