Why the Expressive Power of Languages such as Ada is needed for future Cyber Physical Systems

Alan Burns

Topics of the Talk

- What do Cyber Physical Systems need?
  - Managed resources
- How are resources managed?
  - Scheduling theory
- How can programmers gain access to scheduling theory?
  - Programming abstractions
- Which language provides the most useful set of abstractions?
  - Ada

Cyber Physical Systems

- Complex embedded (software intensive) systems
- Open system boundaries
  - Mixed Criticality subsystems
- Feedback Control
  - discrete and continuous time, deadlines, iteration rates, …
- High reliability requirements
  - Including Safety-Critical
- Mass produced systems need very cost effective hardware solutions
  - Size, weight and power consumption
- High levels of functionality required
  - Many-core, heterogeneous platforms etc

Scheduling

- The branch of Computer Science that deals with resource usage in this context is real-time computation

- Scheduling protocols promote efficient (and at times optimal) resource usage

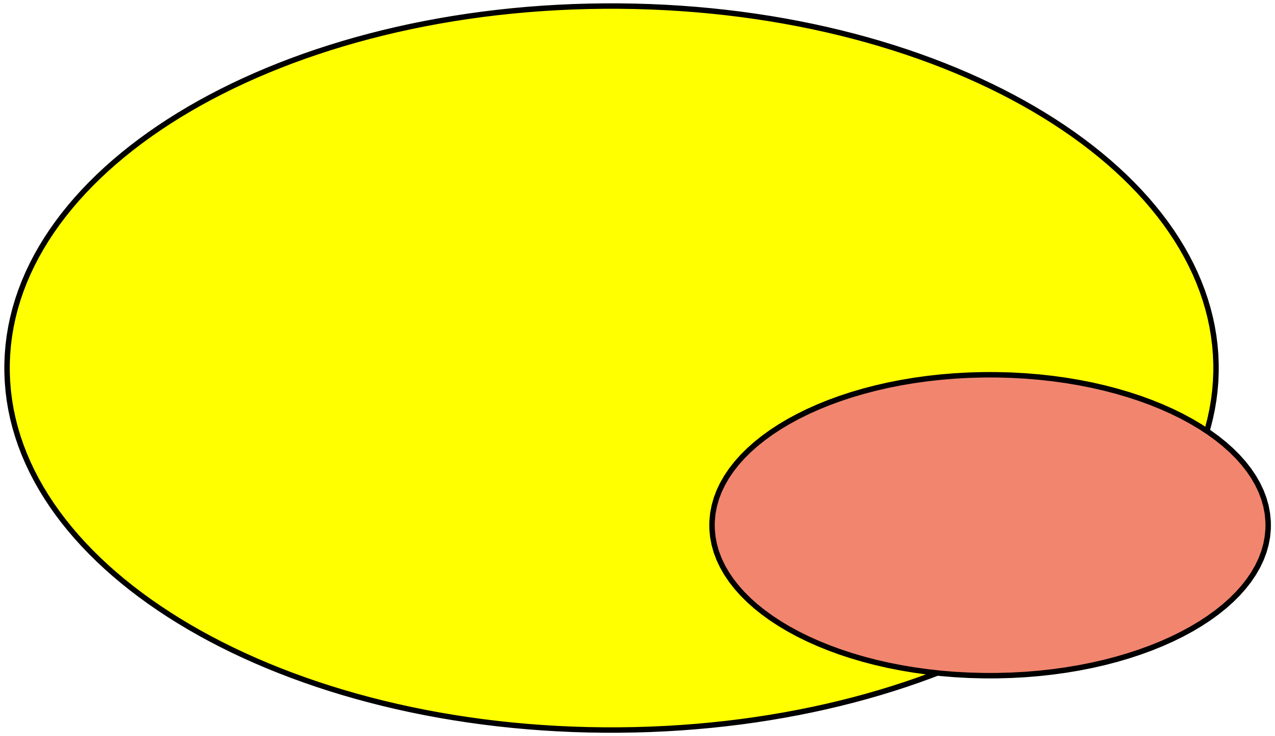
- And scheduling analysis provides the means of verifying that, even in the worst-case, deadlines will be met

Scheduling Theories

Lots of theoretical material available

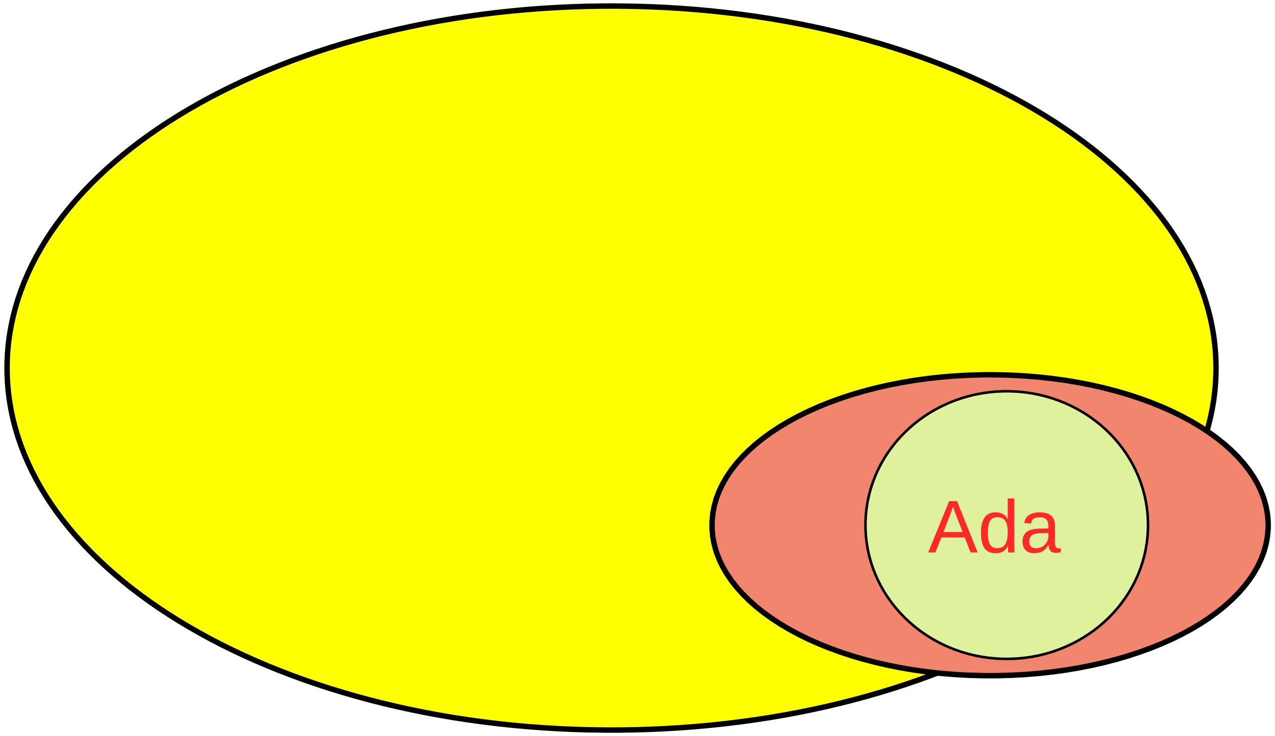
Scheduling Theories

Some of it relevant to CPS



Scheduling Theories

Some of this supported by Ada



Basic Requirements

- Interactions with the parallel world
  - requires concurrency (tasks, threads, processes etc)
- Sharing between distinct software components
  - synchronisation controls (semaphores, mutexes, monitors etc)
- Synchronisation with external real-time
  - clock abstractions, delay primitives and deadlines
- Synchronisation with external events
  - interrupt handling

Basic Scheduling

- Predicable and effective task ordering
  - static priority attributes for tasks, priority ceilings for monitors
- Deadline aware task execution
  - deadline attributes for tasks, protocols for effective sharing
- Deterministic execution order
  - Non-preemptive scheduling (with static priorities)

Improved Resource Utilisation

- Deferred pre-emption
    - Non-preemptive final section
- Dual priorities
- Dynamic priorities
    - which can be used to program a wide variety of protocols

More General Computational Models

- Logical Execution Time (no internal I/O)

- Open Systems with admission control
- Anytime or imprecise algorithms
- Dynamic periods and deadlines (elastic)

- N in M
- Multiframe
- Generalised Task (DAG model)

Resilience

- Deadline miss detection

- Budget monitoring
- Budget overrun detection
- Budget enforcement – various forms of servers

- Watchdog timers
- Aborting rogue computation
- Budget management per task
- Budget management per group of tasks
- Early task termination identification

Multiprocessor Scheduling

- Partitioned scheduling
  - managing the static assignment of tasks/threads to processors/cores
- Global scheduling
  - managing the run-time migration of tasks/threads to follow the rules of the scheduling protocol
- Semi-partitioned scheduling
  - managing the controlled migration of individual tasks/threads at run-time
- Sharing
  - controlling the sharing of resources between potentially parallel executing tasks/threads (this is a major open problem, in that effective general purpose protocols are not yet available).

Advanced Multiprocessor Facilities

- TkC, and DkC
    - global schemes with priority-based scheduling then non-preemptive
- Tasklets
    - to model parallelism within a task/thread
- Barriers
    - to efficiently synchronise tasks/threads on multiprocessor platforms

Mixed Criticality Systems

- Efficient usage of computing resources
- Budget management
- Mode change control
  - task/thread parameter modification (extend period and deadlines)
  - suspending tasks/threads
  - modifying scheduling attributes: priorities and deadlines
  - resume tasks/threads

Some Other Requirements

- Control of when tasks/threads preform I/O
    - e.g. minimising input and output jitter
- Control of memory used by tasks/threads
- Control of power used by tasks/threads
- Control over the speed of variable rate processors
- Control over placement on FPGA type hardware

Required Abstractions and/or Interfaces

- Many facilities can be obtained via APIs

- But language abstractions are:
    - More flexible (periodic task with changing period)
    - More composible (budget control and N in M deadlines)
    - More understandable (deeper semantic definition)

Ada's Provisions

- Calendar and real-time clocks
- Static and dynamic creation of tasks
- Delay mechanisms
- Priority assignment
- Protected objects
    - with requeue to give controlled sharing
- Dynamic task priorities and dynamic priority ceilings

Ada’s Provisions

- Priority based dispatching with priority ceiling protocol
- EDF scheduling with the Stack Resource Protocol
  - and possibly in the future the Deadline Floor Protocol DFP
- Round Robin and non-preemptive dispatching
- Hierarchical scheduling
  - for example, combined priority-based and EDF
  - Particularly useful for mixed criticality systems

Ada Provisions

- Primitives to allow tasks to suspend themselves and other tasks
- Timing events
  - code that executes at a specified time (can be used to control input and output jitter)
- Group budget monitoring and control
  - allows standard execution time servers such as the Periodic Server, Sporadic Server and Deferrable Server to be programmed

Ada's Provisions for Resilient Code

- Budget clocks that monitor task execution time, and can signal when specified levels of usage have been reached
- Task aborting, and the ability to abandon computation at the sub-task level (ATC -- select then abort))
- Timing events -- that are only execute in error conditions, i.e. programmed watchdog timers
- Signalling when a task terminates (useful when the task should not!)

To support multiprocessor execution:

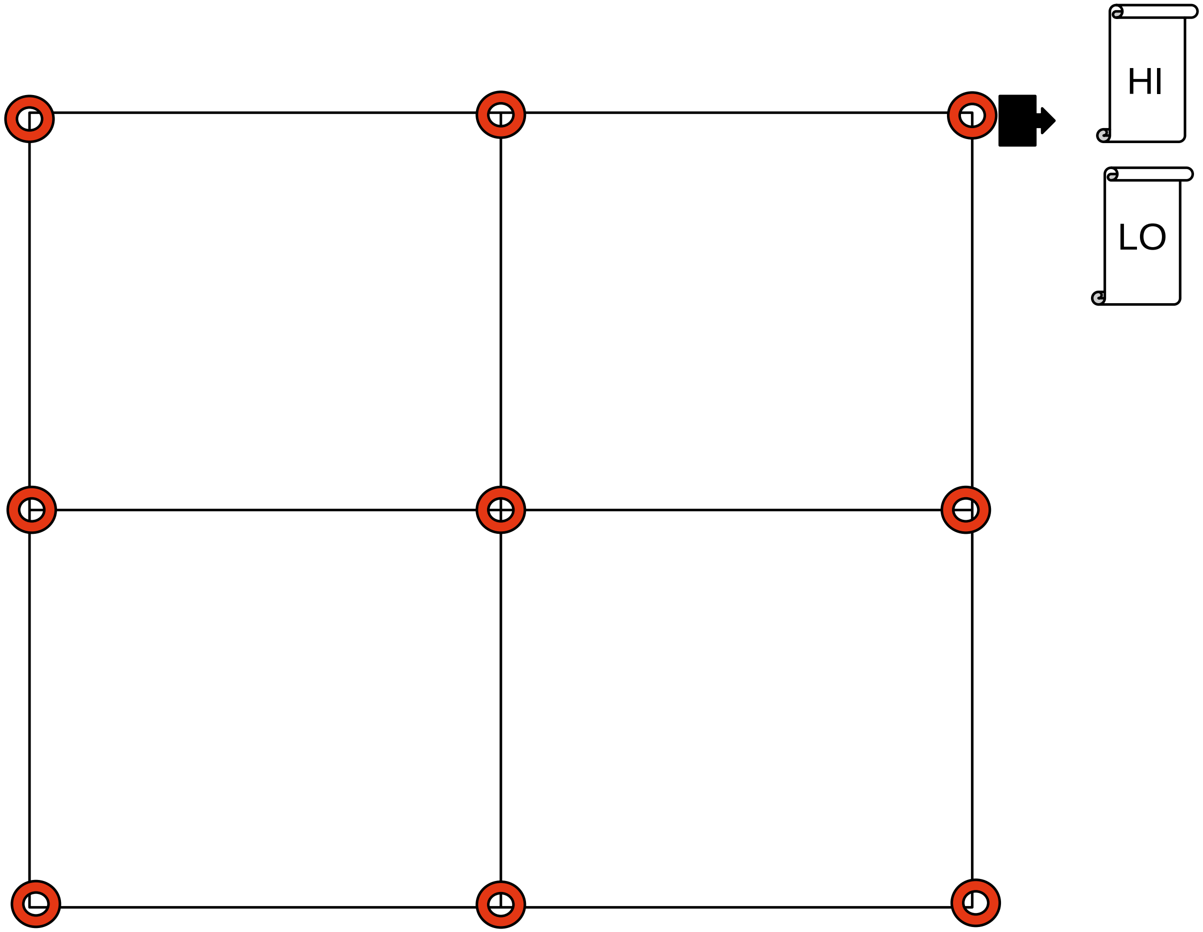
- Use of memory pools to control this important resource
- Affinities that can control where a task executes
  - a task can be restricted to just one CPU, a groups of CPUs or be allowed to execute on any CPU
- Dynamic affinities to allow semi-partitioned schemes to be programmed

Missing Features

- Support for parallel execution within a task
    - a plan for including the notion of tasklet into the language is currently under consideration
- Support for energy aware programming
    - API to whatever is supported by the underlying hardware/run-time is the only current approach available
    - I would like to execute a loop within a bound determined by energy available
- Support for an effective synchronisation scheme for multiprocessor execution
    - many schemes have been proposed in the literature but there is not yet consensus on which Ada can build

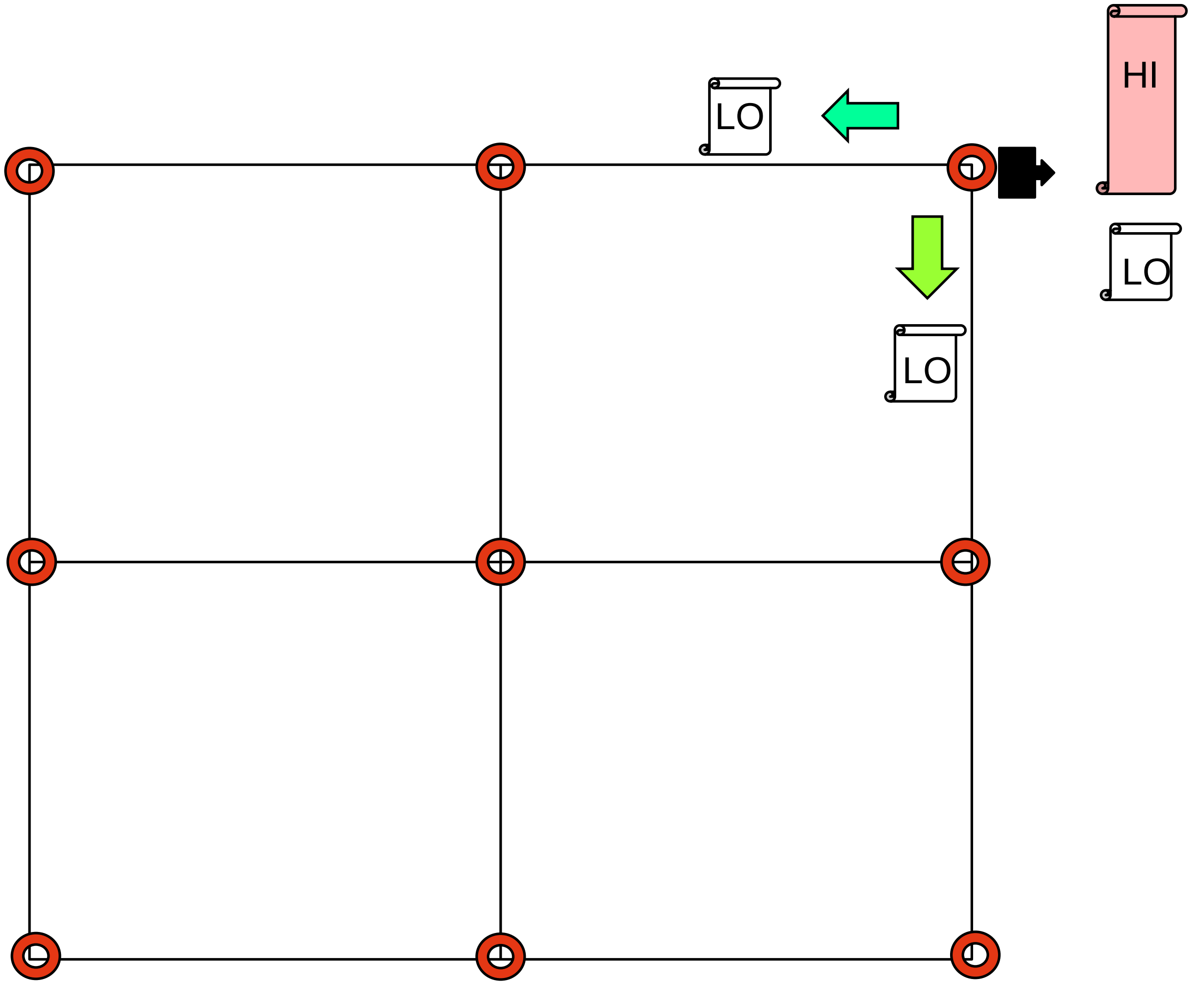
Use Cases (1)

- 9 core platform
- 2 criticality levels (HI and LO)
- Many tasks of either HI or LO criticality
- Static assignment of tasks to cores
- All LO-crit tasks on a core have a policed (shared) budget
  - EDF scheduling
- All HI-crit tasks have an individual budget
  - Fixed Priority scheduling
- If any HI-crit task exceeds its budget then a defined set of LO-crit tasks migrate



Fixed Pri

EDF


LO
HI
LO
LO

Analysis

- Analysis for this scenario exists
    - H. Xu and A. Burns, Semi-partitioned Model for Dual-core Mixed Criticality System, 23rd RTNS, pp257-266, 2015

- If no more than 3 core experience overload then all deadlines continue to be met

- If more than 3 core experience overload then all HI-crit tasks continue to meet their deadlines

To program in Ada

- Assign tasks to each core
    - One dispatching domain (per 9 core template)
    - `Set_CPU` in `System.Multiprocessors.Dispatching_Domains`
- Hierarchical scheduling
    - `Priority_Specific_Dispatching`
    - Assign HI-crit tasks priorities in top range `(Set_Priority)`
    - Assign LO-crit tasks to EDF range (`EDF_Across_Priorities`)
    - Assign ceiling priorities to all Protected Objects

To program in Ada

- Allocate all LO-crit tasks in a core a single budget
    - `Add_Task` in `Ada.Execution_Time.Group_Budgets`
    - Assign budget (from analysis) – `Replenish`
- Assign a budget clock to each HI-crit task
    - `Timer`
- Allocate appropriate periods or event triggers for each task
    - `delay until, POs, Attach_Handler`

At run-time for LO-crit tasks

- If group budget exhausted before replenishment
  - `Set_Handler` (from group budgets) to
  - Suspend all LO-crit tasks `(Hold` in `Ada.Asynchronous_Task_Control)`

- Replenish group budget periodically
  - Using Timing event (`Set_Handler`)
  - To `Replenish,` and
  - Release any suspended tasks `(Continue)`

At run-time for HI-crit tasks

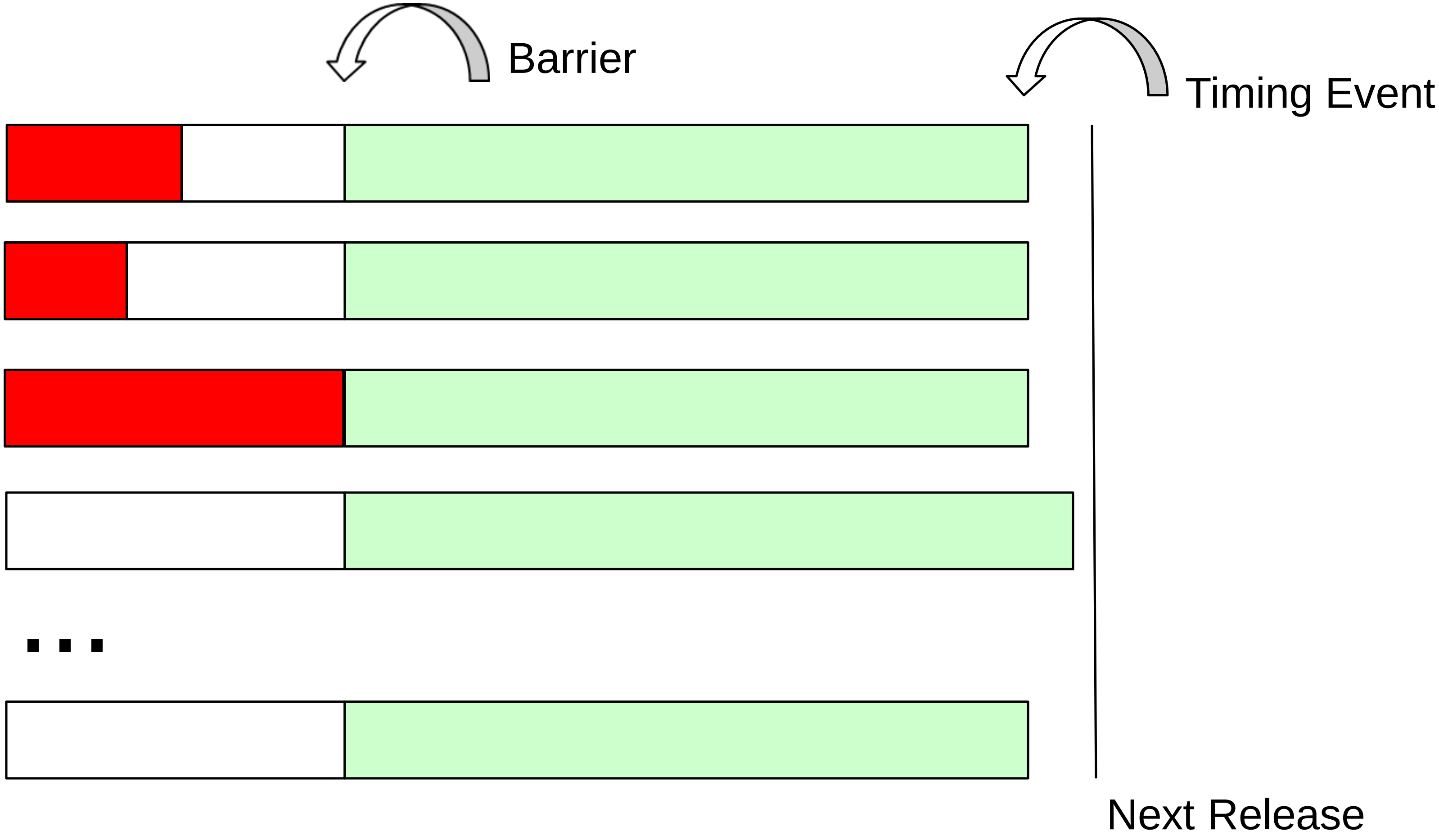
- If any HI-crit task goes above budget
  - `Set_Handler` used to fix the protected procedure that:
  - For each moving LO-crit task
    - Remove from group budget (`Remove_Task`)
    - Migrate to new core (`Set_CPU`)
    - Add to group budget on new core (`Add_Task`)
    - Release if suspended (`Is_Held` and `Continue`)
  - When LO-crit task next released return to original core

Ada Facilities

- The following libraries have been used
  - `Asynchronous_Task_Control`
  - `Task_Identification`
  - `Dispatching.EDF`
  - `Real_Time`
  - `Execution_Time`
  - `Execution_Time.Timers`
  - `Execution_Time.Group_Budgets`
  - `Real_Time.Timing_Events`
  - `System.Multiprocessors.Dispatching_Domains`

Use Case (2)

- Two phases of execution (HI and LO again)
- First is safety-critical and deterministic
- Second is critically but open-ended
    - Involves image processing and data presentation
- First phase runs on only 3 cores
    - To get more predictable memory access times
- Second phase on all 9 cores
- No second phase work can start until all first phase work is completed


Barrier
Timing Event
...
Next Release

To Program in Ada

- Each core has statically allocated a single LO-crit task and a HI-crit task

- Some (3) HI-crit tasks contain application code
  - After completing their work they call the barrier

- The others just contain a call to the barrier

- On release from the barrier they rendezvous with the LO-crit task to release it

To Program in Ada

- LO-crit tasks
  - Wait for rendezvous from HI-crit task
  - When released
    - Iterate through an improvement cycle
    - Abandon when signalled to do so (Timing Event)
    - Use a PO to store safe data (max overrun is ***delta***)
- HI-crit tasks
  - Delay until timing event time + ***delta*** to be released
    - i.e. timing event is at time period - ***delta***
  - When released from barrier rendezvous with LO-crit task

Ada facilities

- Timing Events
- POs (for abort deferred behaviour)
- **`select then abort`**
- Rendezvous
  - Timed entry call, so HI-crit task not blocked
- Barrier protocol

- Allocation of tasks to cores

Conclusions

- I have tried to highlight the significant body of scheduling theory that can be used to build cost-effective and reliable cyber-physical systems
- To use this theory the system developer / programmer must be able to access the protocols and approaches that scheduling theory has defined
- Ada provides an effective means of providing this access

But

- Ada run-times must be available that do faithfully implement language semantics and all defined features in the Real-Time Annex

- There are abstractions that are not as yet available in Ada (or other real-time programming languages)

- And there are still open issues in terms of the required scheduling theory for CPS