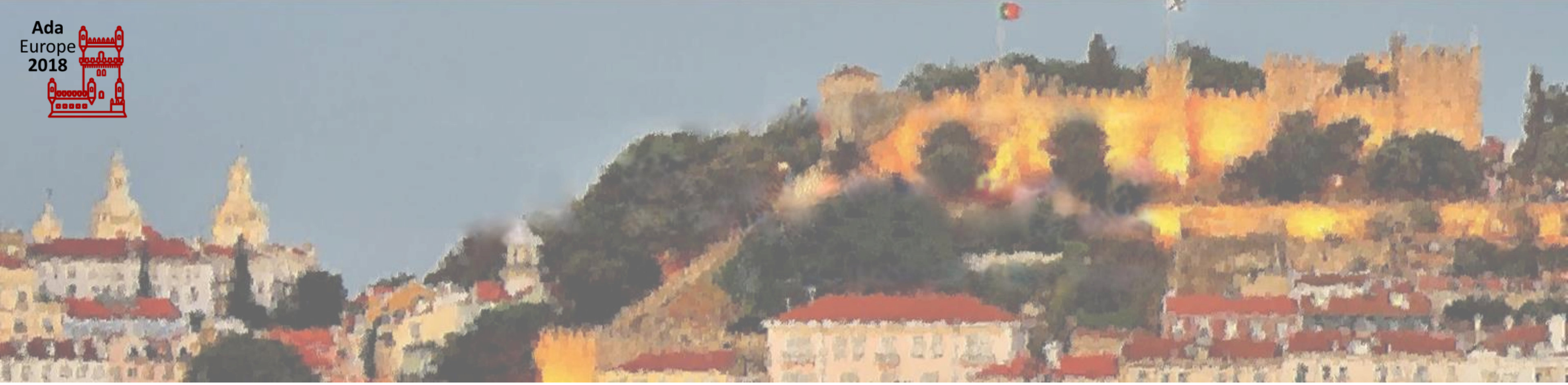
Ada Europe 2018



# Safe Parallelism

## Compiler Analysis Techniques for Ada and OpenMP

**Sara Royuela** [1,3], Xavier Martorell [1,3], Eduardo Quiñones [3], Luis Miguel Pinho [2]

[1] UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

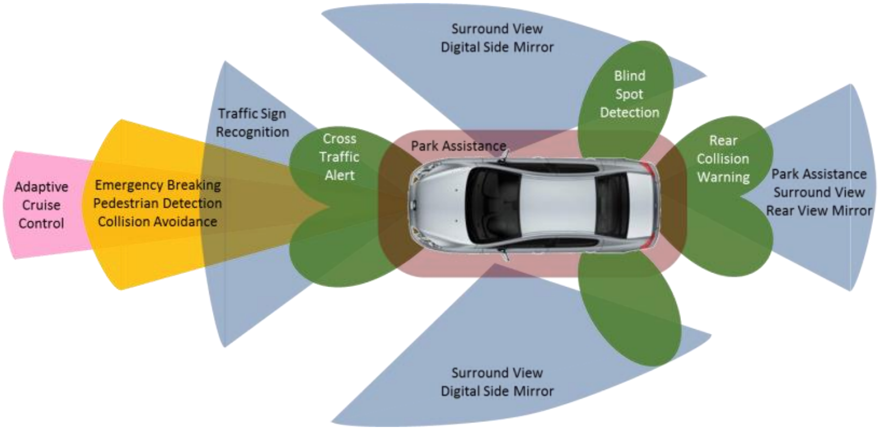
[2] CISTER - Research Centre in Real-Time & Embedded Computing Systems

[3] Barcelona Supercomputing Center Centro Nacional de Supercomputación

# Challenges in safety-critical systems

- Need for **performance**

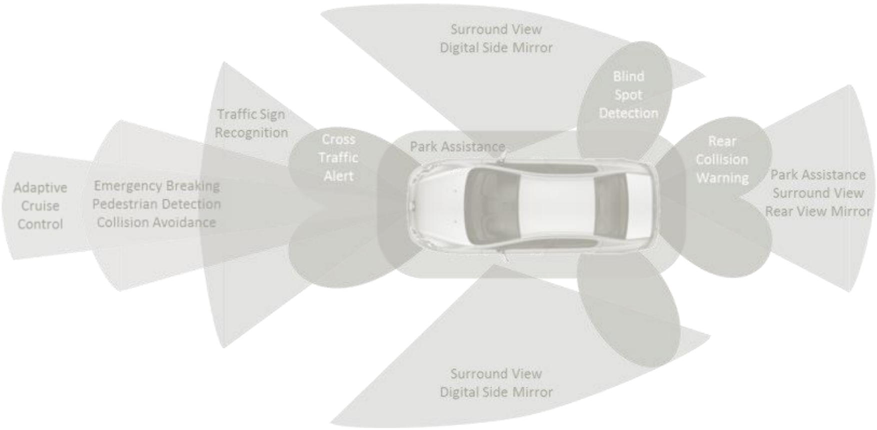
  Current safety-critical real-time systems require computational power beyond simple single-core architectures.



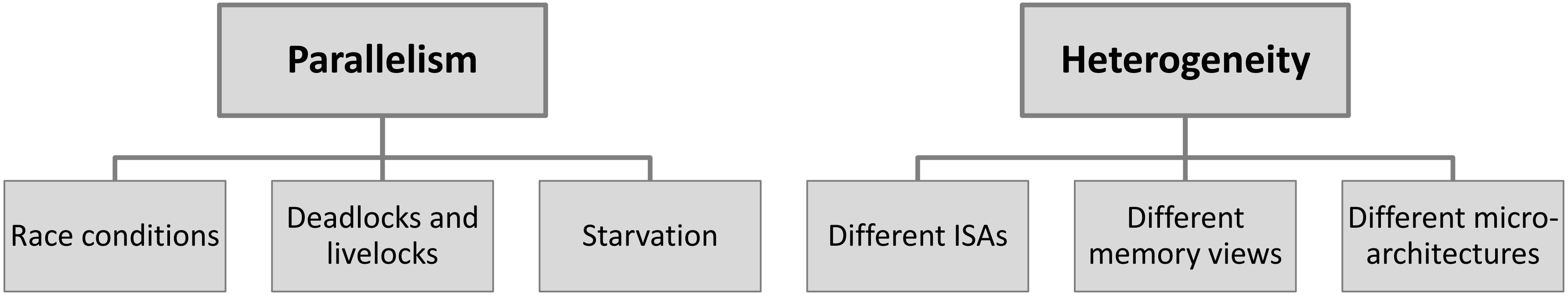
# Challenges in safety-critical systems

– Need for **performance**

Current safety-critical real-time systems require computational power beyond simple single-core architectures.



– **Complexities** of Parallel heterogeneous architectures



**Parallelism**

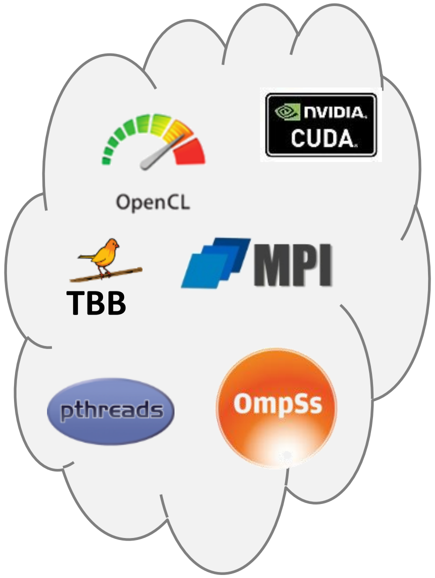
- Race conditions
- Deadlocks and livelocks
- Starvation

**Heterogeneity**

- Different ISAs
- Different memory views
- Different micro-architectures

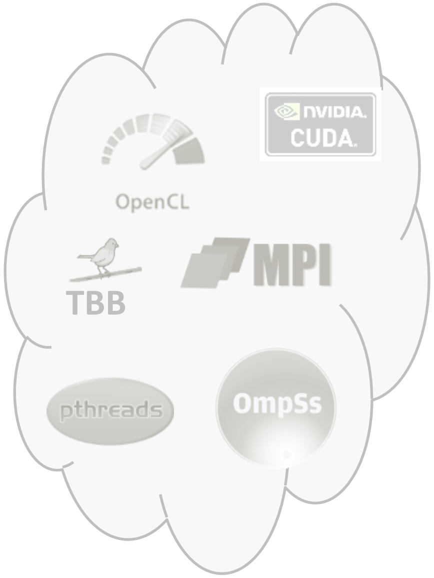
# How to cope with such complexity?

## Parallel programming models



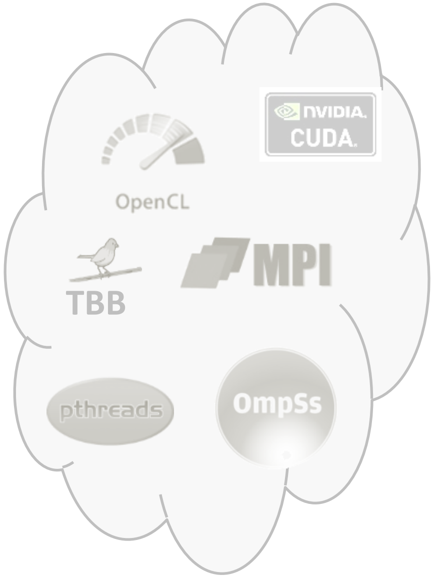
# How to cope with such complexity?

## Parallel programming models



# How to cope with such complexity?

## Parallel programming models



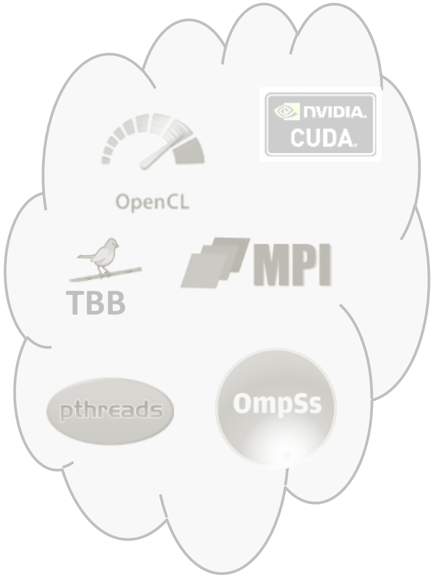
20 years of development gather the benefits of other languages

- Delivers **performance** comparable with Intel TBB, CUDA, OpenCL and MPI
- Offers **robustness** without sacrificing performance compared to Pthreads
- Eases **debugging** by enabling trivial single-threaded compilation

OpenMP®

# How to cope with such complexity?

## Parallel programming models



20 years of development gather the benefits of other languages

- Delivers **performance** comparable with Intel TBB, CUDA, OpenCL and MPI
- Offers **robustness** without sacrificing performance compared to Pthreads
- Eases **debugging** by enabling trivial single-threaded compilation

The latest specification meets the characteristics of heterogeneous architectures

- **Accelerator model** for improved performance/power consumption
- Allows expressing **fine grain**, both **structured** and **unstructured**, **parallelism**
- Implemented by several **chip** (TI Keystone, Kalray MPPA) and **compiler vendors** (GNU, Intel, IBM)

# What is OpenMP and how far is it from the safety-critical domain?

# Introduction to OpenMP

- Forms of parallelism:
  - **Thread model**: direct management of threads (structured)
  - **Tasking model**: tasks as an abstraction of threads (structured and unstructured)

# Introduction to OpenMP

- Forms of parallelism:
  - **Thread model**: direct management of threads (structured)
  - **Tasking model**: tasks as an abstraction of threads (structured and unstructured)
- Based on user directives and clauses for:
  - *Spawning parallelism*: `parallel`
  - *Distributing parallelism*: `task`, `taskloop`
  - *Synchronization*: `barrier`, `taskwait`, `depend`
  - *Driving execution*: `untied`, `priority`, `taskyield`, …

# Introduction to OpenMP

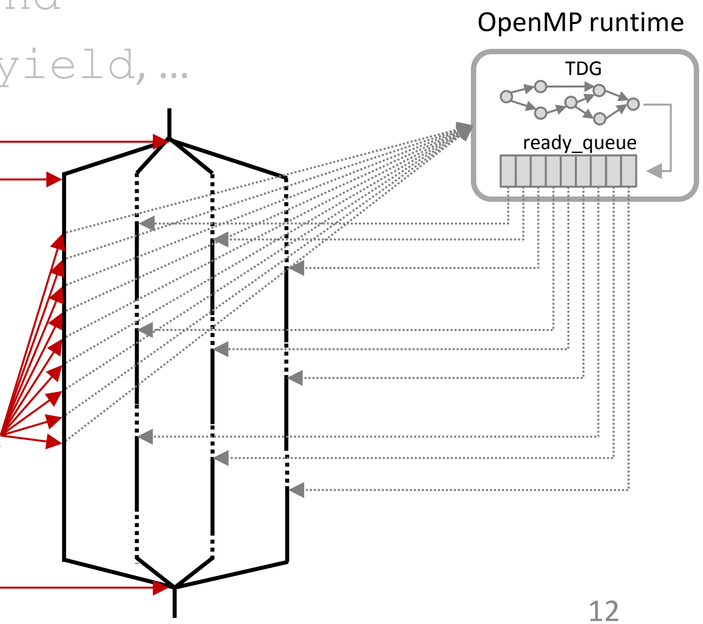
- Forms of parallelism:
  - **Thread model**: direct management of threads (structured)
  - **Tasking model**: tasks as an abstraction of threads (structured and unstructured)
- Based on user directives and clauses for:
  - *Spawning parallelism*: `parallel`
  - *Distributing parallelism*: `task`, `taskloop`
  - *Synchronization*: `barrier`, `taskwait` ,`depend`
  - *Driving execution*: `untied`, `priority`, `taskyield`, …

```
void matmul(int N, float A[N][N], float B[N][N], float C[N][N])
{
  #pragma omp parallel num_threads(4)
  #pragma omp master
  for (int i=0; i<N; i++)
    for (int j=0; j<N; j++)
      for (int k=0; k<N; k++)
        #pragma omp task depend(in:A[i][k]) depend(in: B[k][j])\
                         depend(inout:C[i][j])
        C[i][j] += A[i][k] * B[k][j];
}
```

# Introduction to OpenMP

- Forms of parallelism:
  - **Thread model**: direct management of threads (structured)
  - **Tasking model**: tasks as an abstraction of threads (structured and unstructured)
- Based on user directives and clauses for:
  - *Spawning parallelism*: `parallel`
  - *Distributing parallelism*: `task`, `taskloop`
  - *Synchronization*: `barrier`, `taskwait`, `depend`
  - *Driving execution*: `untied`, `priority`, `taskyield`, ...

```
void matmul(int N, float A[N][N], float B[N][N], float C[N][N])
{
  #pragma omp parallel num_threads(4)
  #pragma omp master
  for (int i=0; i<N; i++)
    for (int j=0; j<N; j++)
      for (int k=0; k<N; k++)
        #pragma omp task depend(in:A[i][k]) depend(in: B[k][j])\
                         depend(inout:C[i][j])
        C[i][j] += A[i][k] * B[k][j];
}
```



# Safety-critical OpenMP: where is the problem?

## *OpenMP* 4.5 (API, page 1)

- *OpenMP-compliant **implementations are not required** to check*
  - for data dependencies, data conflicts, race conditions, or deadlocks, (…)
  - for code sequences that cause a program to be classified as non-conforming
- *Application **developers are responsible** for correctly using the OpenMP API to produce a conforming program*

# Safety-critical OpenMP: requirements

## *OpenMP* 4.5 (API, page 1)

- *OpenMP-compliant **implementations are not required** to check*
  - for data dependencies, data conflicts, race conditions, or deadlocks, (…)
  - for code sequences that cause a program to be classified as non-conforming
- *Application **developers are responsible** for correctly using the OpenMP API to produce a conforming program*

| | Functional Safety | Time Safety |
|---|---|---|
| What to achieve | **Reliability**: do as expected<br>**Resiliency**: recover from errors | **Predictability**: analyzable<br>**Feasibility**: fulfill deadlines |
| How to achieve it | Programming model restrictions<br>Compiler analysis<br>Runtime mechanisms | WCET analysis<br>Schedulability analysis |

# Safety-critical OpenMP: requirements

*OpenMP* 4.5 (API, page 1)

- *OpenMP-compliant **implementations are not required** to check*
  - for data dependencies, data conflicts, race conditions, or deadlocks, (…)
  - for code sequences that cause a program to be classified as non-conforming
- *Application **developers are responsible** for correctly using the OpenMP API to produce a conforming program*

| | Functional Safety | Time Safety |
|---|---|---|
| What to achieve | **Reliability**: do as expected<br>**Resiliency**: recover from errors | **Predictability**: analyzable<br>**Feasibility**: fulfill deadlines |
| How to achieve it | Programming model restrictions<br>Compiler analysis<br>Runtime mechanisms | WCET analysis<br>Schedulability analysis |

Our scope

# Solutions for a safety-critical OpenMP

**Compiler**

- Force implementations to detect:
    - race conditions
    - deadlocks
    - non-conforming sequences

S. Royuela, et al., "**A functional safety OpenMP* for critical real-time embedded systems**", IWOMP 2017

# Solutions for a safety-critical OpenMP

**Compiler**

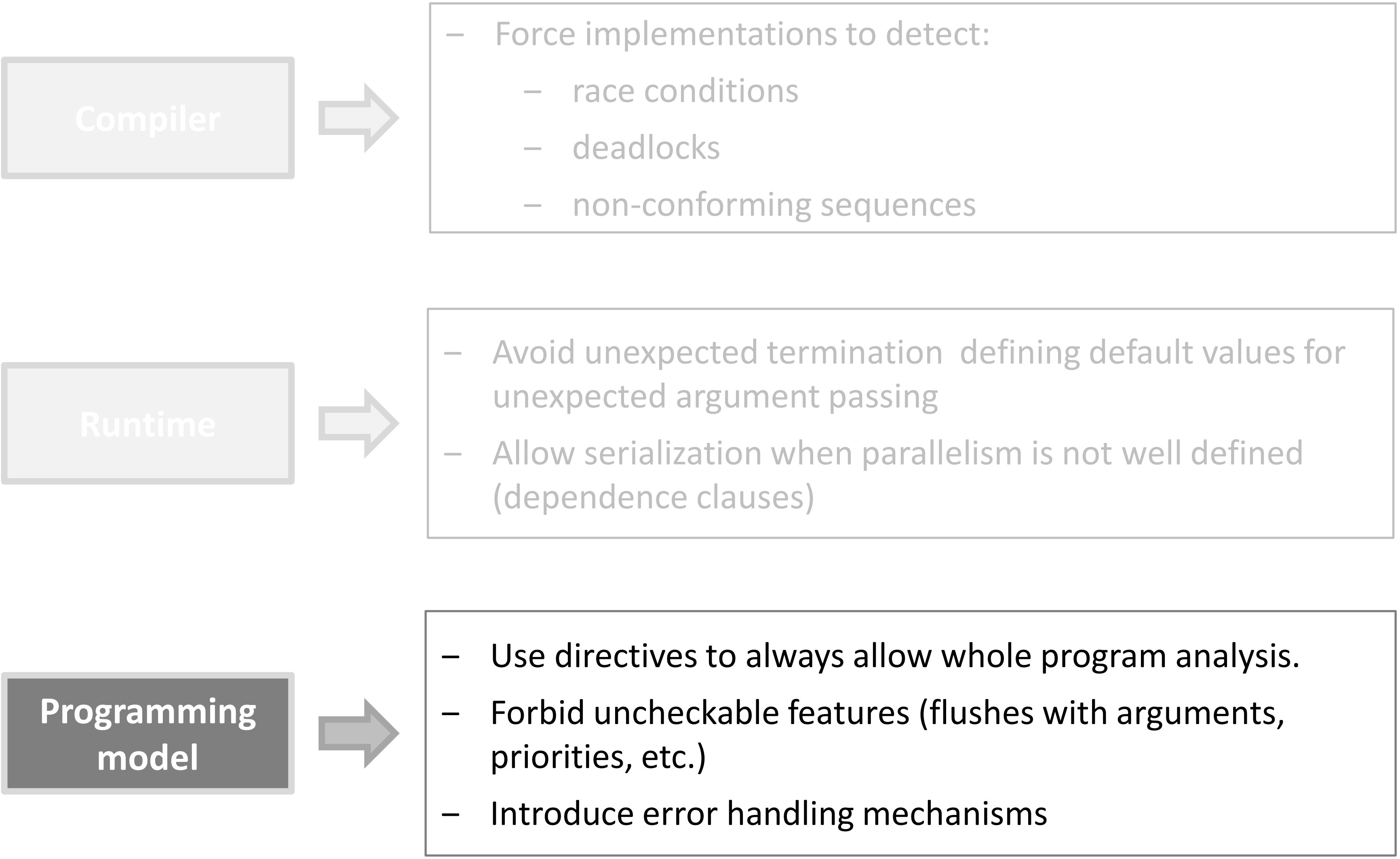
- Force implementations to detect:
  - race conditions
  - deadlocks
  - non-conforming sequences

**Runtime**

- Avoid unexpected termination defining default values for unexpected argument passing
- Allow serialization when parallelism is not well defined (dependence clauses)

S. Royuela, et al., “**A functional safety OpenMP* for critical real-time embedded systems**”, IWOMP 2017

# Solutions for a safety-critical OpenMP



Compiler

- Force implementations to detect:
  - race conditions
  - deadlocks
  - non-conforming sequences

Runtime

- Avoid unexpected termination  defining default values for unexpected argument passing
- Allow serialization when parallelism is not well defined (dependence clauses)

Programming model

- Use directives to always allow whole program analysis.
- Forbid uncheckable features (flushes with arguments, priorities, etc.)
- Introduce error handling mechanisms

S. Royuela, et al., "**A functional safety OpenMP* for critical real-time embedded systems**", IWOMP 2017

# Parallelism in Ada202X

# Ada: concurrency and parallelism now

- Ada **concurrent model** integrated at base language level
  - Tasking facilities for exposing concurrency at **coarse grain**
  - Synchronization mechanisms: **protected objects**, rendezvous

# Ada: concurrency and parallelism now

- Ada **concurrent model** integrated at base language level
  - Tasking facilities for exposing concurrency at **coarse grain**
  - Synchronization mechanisms: **protected objects**, rendezvous

- Ada **parallel model** to be included in Ada202X
  - **Tasklets** for exposing parallelism at fine grain
  - Support for **structured parallelism**

| Parallel blocks | Parallel loops |
|---|---|
| **parallel do**<br>    handled_sequence_of_statements<br>**and**<br>    handled_sequence_of_statements<br>{**and**<br>     handled_sequence_of_statements}<br>**end do;** | **parallel**<br>**for** I **in** LB..UB **loop**<br>    sequence_of_statements<br>**end loop;** |

  - Does not allow **blocking operations** within parallel regions
  - Under implementation (e.g., AdaCore)

# OpenMP to implement the tasklet model

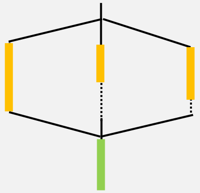
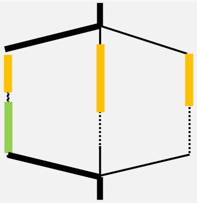
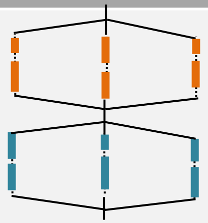
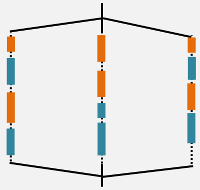
- OpenMP mimics the tasklet model behavior at all levels:
  - *Forms of parallelism*: parallel blocks and parallel loops
  - *Execution model*: run-to-completion
  - *Memory model*:  relaxed consistency memory model
  - *Progression model*: immediate, eventual and limited

S. Royuela, X. Martorell, E. Quiñones, and L.M. Pinho, "**OpenMP tasking model for Ada: safety and correctness**", AE 2017
S. Royuela, L.M. Pinho, and E. Quiñones, "**Converging Safety and High-performance Domains: Integrating OpenMP into Ada**", DATE 2018

# OpenMP to implement the tasklet model

- OpenMP mimics the tasklet model behavior at all levels:
  - *Forms of parallelism*: parallel blocks and parallel loops
  - *Execution model*: run-to-completion
  - *Memory model*:  relaxed consistency memory model
  - *Progression model*: immediate, eventual and limited

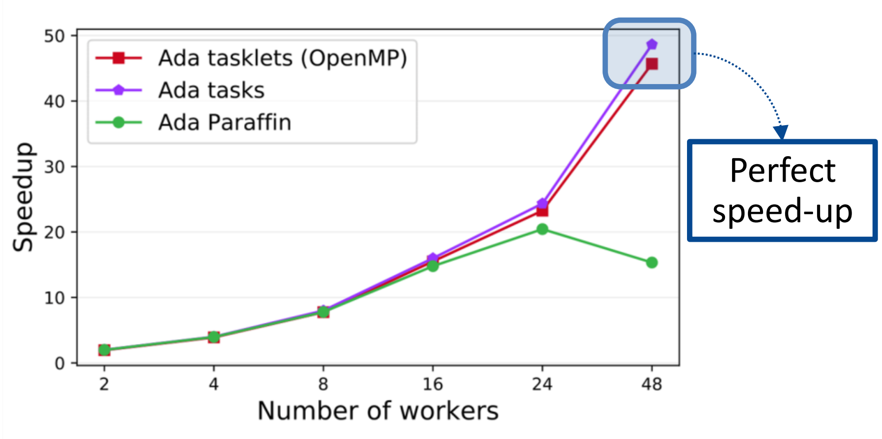
- OpenMP offers more flexibility

| **Parallel blocks** | | **Parallel loops** | |
|---|---|---|---|
| **Tasklet** | **OpenMP** | **Tasklet** | **OpenMP** |
|  |  |  |  |
| 🟨 parallel code<br>🟩 sequential code | | 🟧 parallel loop 1<br>🟦 parallel loop 2 | |

S. Royuela, X. Martorell, E. Quiñones, and L.M. Pinho, "**OpenMP tasking model for Ada: safety and correctness**", AE 2017
S. Royuela, L.M. Pinho, and E. Quiñones, "**Converging Safety and High-performance Domains: Integrating OpenMP into Ada**", DATE 2018

# OpenMP to further exploit parallelism in Ada

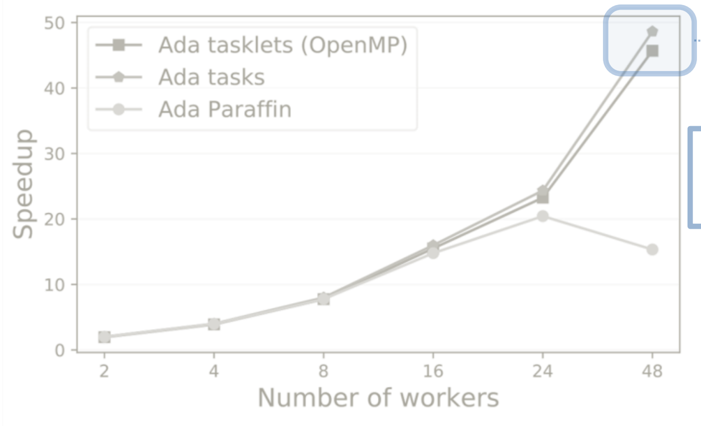
**Matrix** (coarse grain synchronization)



Perfect speed-up

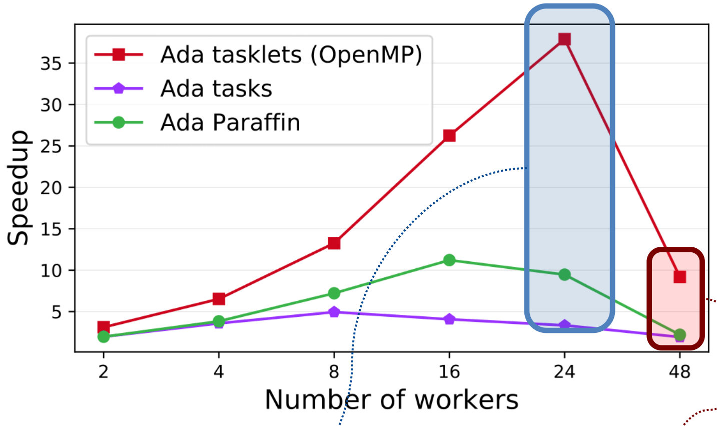
# OpenMP to further exploit parallelism in Ada

**Matrix** (coarse grain synchronization)



Perfect speed-up

**LU** (fine grain synchronization)

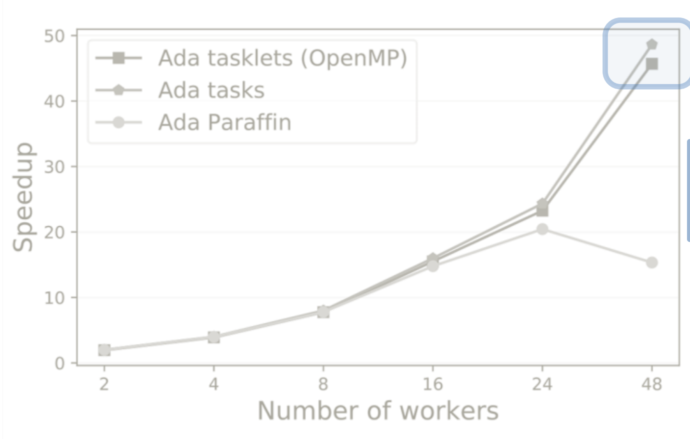


Great speed-up enhancement

Memory boundaries NUMA effect

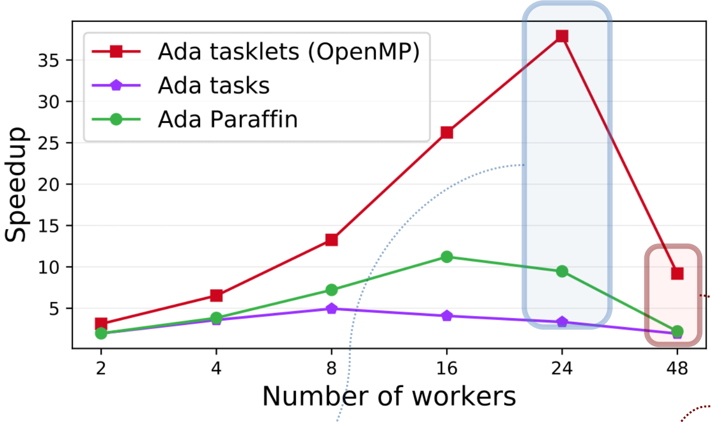
# OpenMP to further exploit parallelism in Ada

## **Matrix** (coarse grain synchronization)



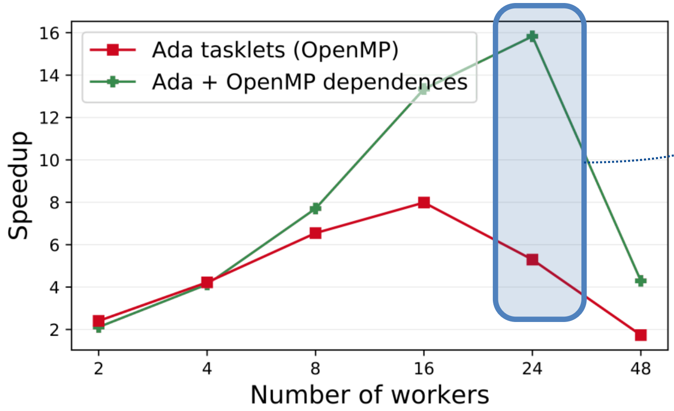
Perfect speed-up

## **LU** (fine grain synchronization)



Memory boundaries NUMA effect

## **Cholesky** (unstructured parallelism)



Great speed-up enhancement

# Analyze Ada/OpenMP programs for data-race detection

# Compiler analysis for Ada/OpenMP programs

Currently:

- Ada **lacks static analyses** for data-race detection
- OpenMP correctness[*] techniques do not **consider concurrency**

[*] *S. Royuela, A. Duran, C. Liao and D.J. Quinlan*, "**Auto-scoping for OpenMP tasks**", IWOMP12.
*S. Royuela, A. Duran and X. Martorell*, "**Compiler automatic discovery of OmpSs tasks dependences**", LCPC12.

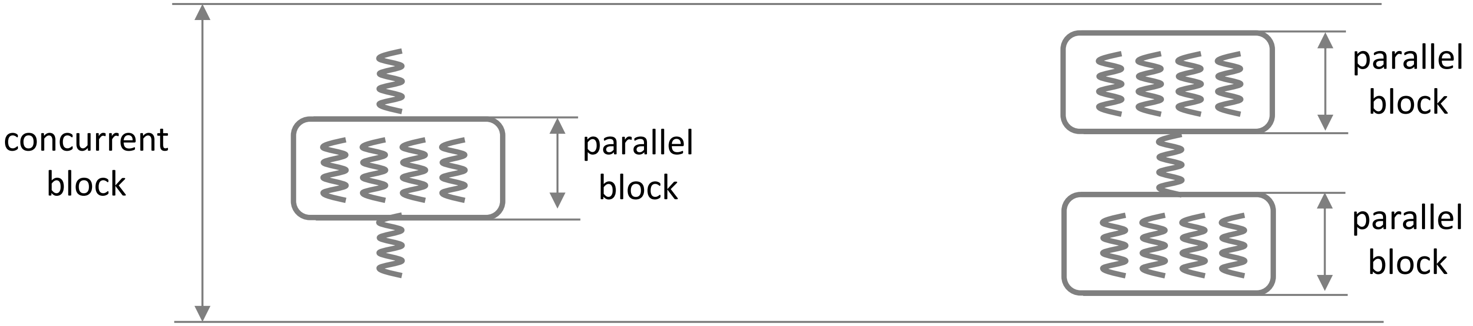
# Compiler analysis for Ada/OpenMP programs

Currently:

- Ada **lacks static analyses** for data-race detection
- OpenMP correctness* techniques do not consider concurrency

Solution:

- Extend current **OpenMP techniques***



* *S. Royuela, A. Duran, C. Liao and D.J. Quinlan*, "**Auto-scoping for OpenMP tasks**", IWOMP12.
*S. Royuela, A. Duran and X. Martorell*, "**Compiler automatic discovery of OmpSs tasks dependences**", LCPC12.

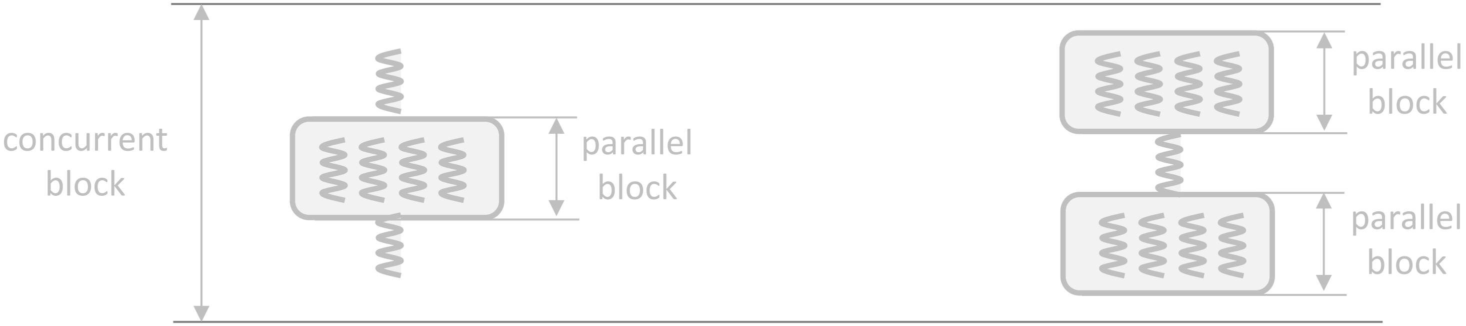
# Compiler analysis for Ada/OpenMP programs

Currently:

- Ada **lacks static analyses** for data-race detection
- OpenMP correctness* techniques do not consider concurrency

Solution:

- Extend current **OpenMP techniques***



The **Ada Ravenscar profile** eases the generation of blocks of concurrency because dynamic task allocation and task termination are forbidden

* *S. Royuela, A. Duran, C. Liao and D.J. Quinlan*, "**Auto-scoping for OpenMP tasks**", IWOMP12.
*S. Royuela, A. Duran and X. Martorell*, "**Compiler automatic discovery of OmpSs tasks dependences**", LCPC12.

# Solve race conditions in Ada/OpenMP

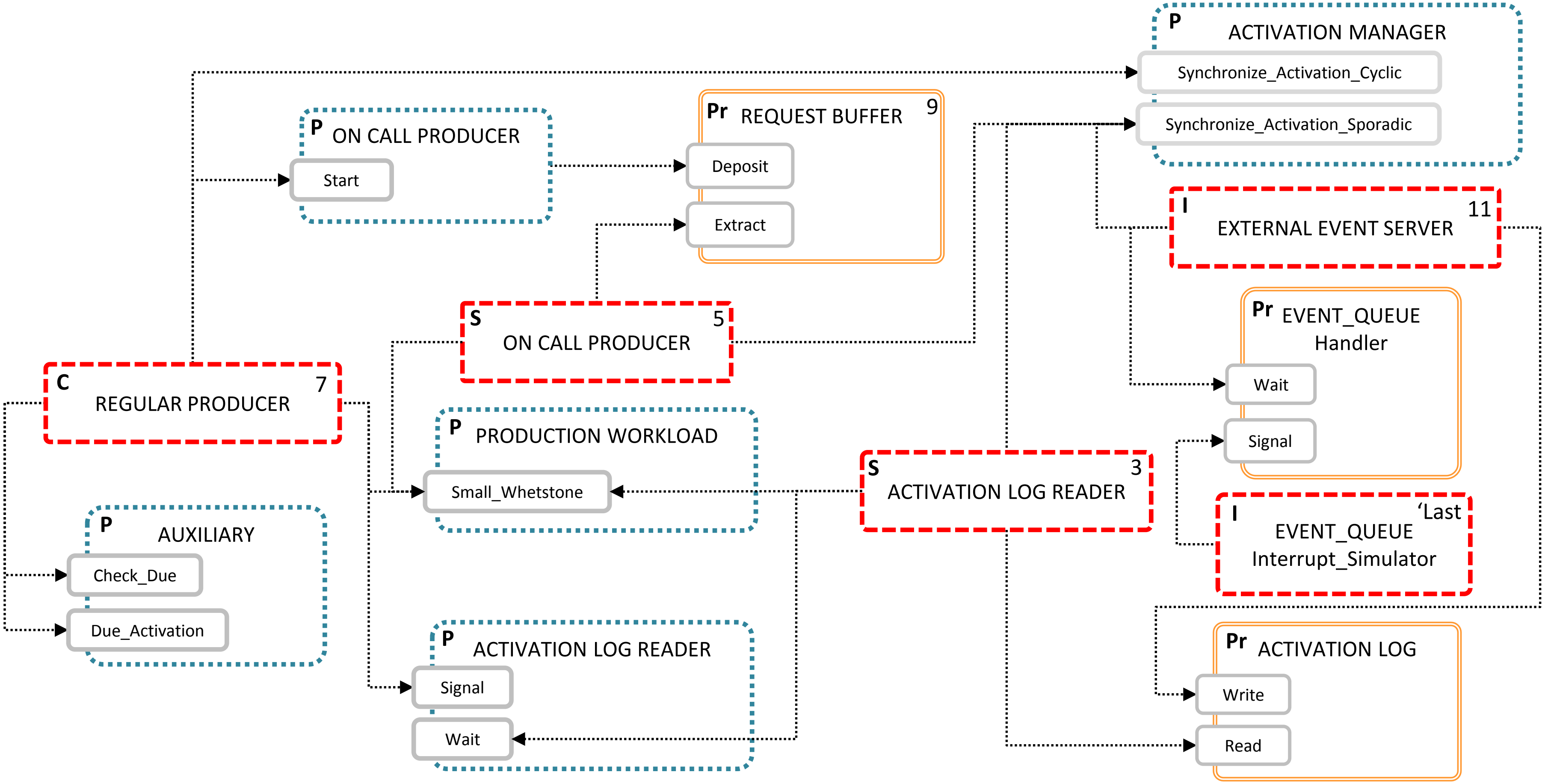
1. Build an interprocedural PCFG
2. Recognize the different blocks of concurrency
3. Apply the following solutions if race conditions may arise:

| Race condition between | | Solution |
|---|---|---|
| Ada tasks | | Ada mechanisms: protected object |
| Ada and OpenMP tasks | | |
| OpenMP tasks | different binding regions[1] | |
| | same binding region[1] | OpenMP mechanisms[2]:<br>– Synchronization constructs and clauses: `taskwait, barrier, depend`<br>– Mutual exclusion constructs: `critical, atomic`<br>– Data-sharing attributes: `private, firstprivate, lastprivate` |

[1] Binding region: the enclosing region that determines the execution context and limits the scope of the effects of the bound region.

[2] *S. Royuela et al.*, "**Compiler Analysis for OpenMP Tasks Correctness**", CF2015.

# *Ravenscar* application (HRT-HOOD)



P ACTIVATION MANAGER
Synchronize_Activation_Cyclic
Synchronize_Activation_Sporadic

P ON CALL PRODUCER
Start

Pr REQUEST BUFFER 9
Deposit
Extract

I EXTERNAL EVENT SERVER 11

S ON CALL PRODUCER 5

Pr EVENT_QUEUE Handler
Wait
Signal

C REGULAR PRODUCER 7

P PRODUCTION WORKLOAD
Small_Whetstone

S ACTIVATION LOG READER 3

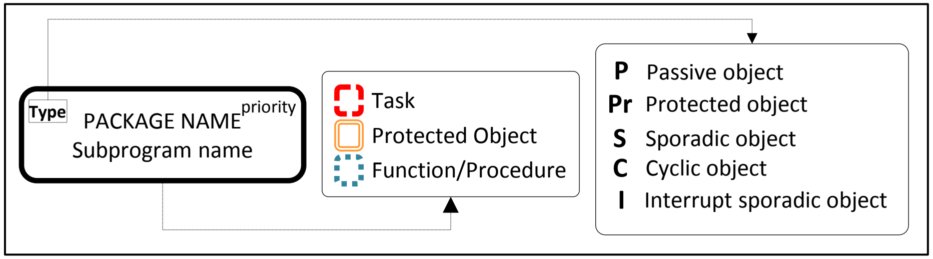
I EVENT_QUEUE Interrupt_Simulator ‘Last

P AUXILIARY
Check_Due
Due_Activation

P ACTIVATION LOG READER
Signal
Wait

Pr ACTIVATION LOG
Write
Read

Extracted from: Burns, B. Dobbing and T. Vardanega, *“Guide for the use of the Ada Ravenscar Profile in high integrity systems”*, 2003.

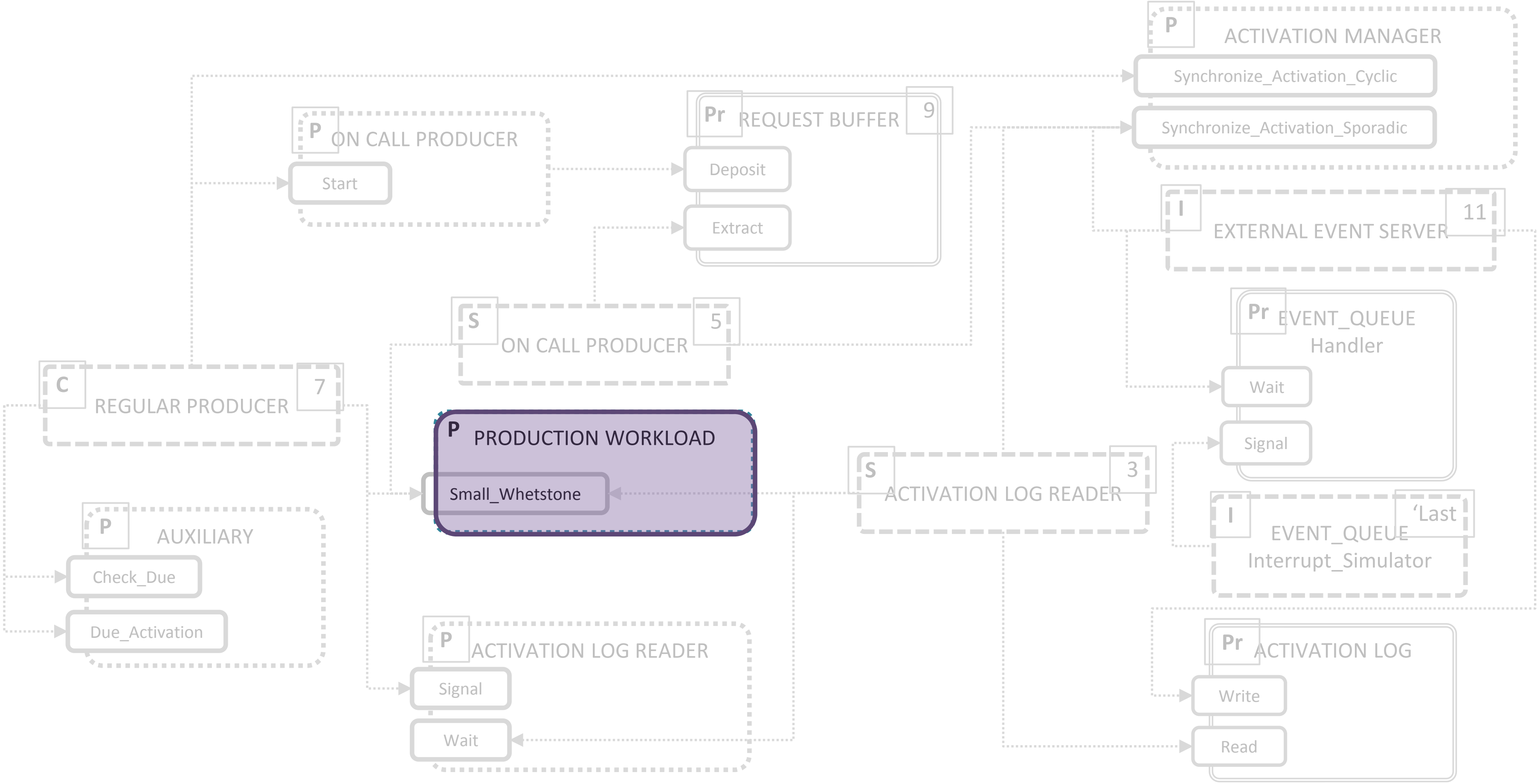


Type PACKAGE NAME priority
Subprogram name

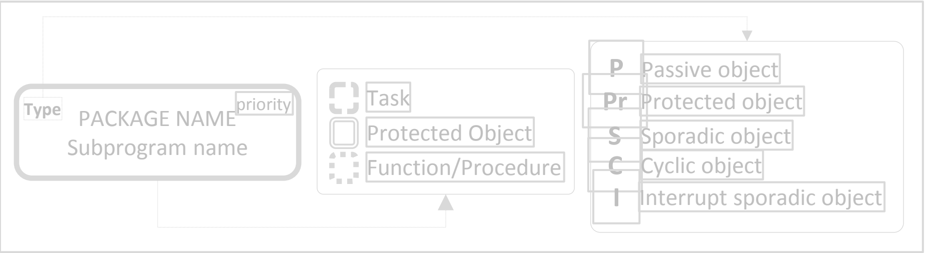
🟥 Task
🟧 Protected Object
🟦 Function/Procedure

P Passive object
Pr Protected object
S Sporadic object
C Cyclic object
I Interrupt sporadic object

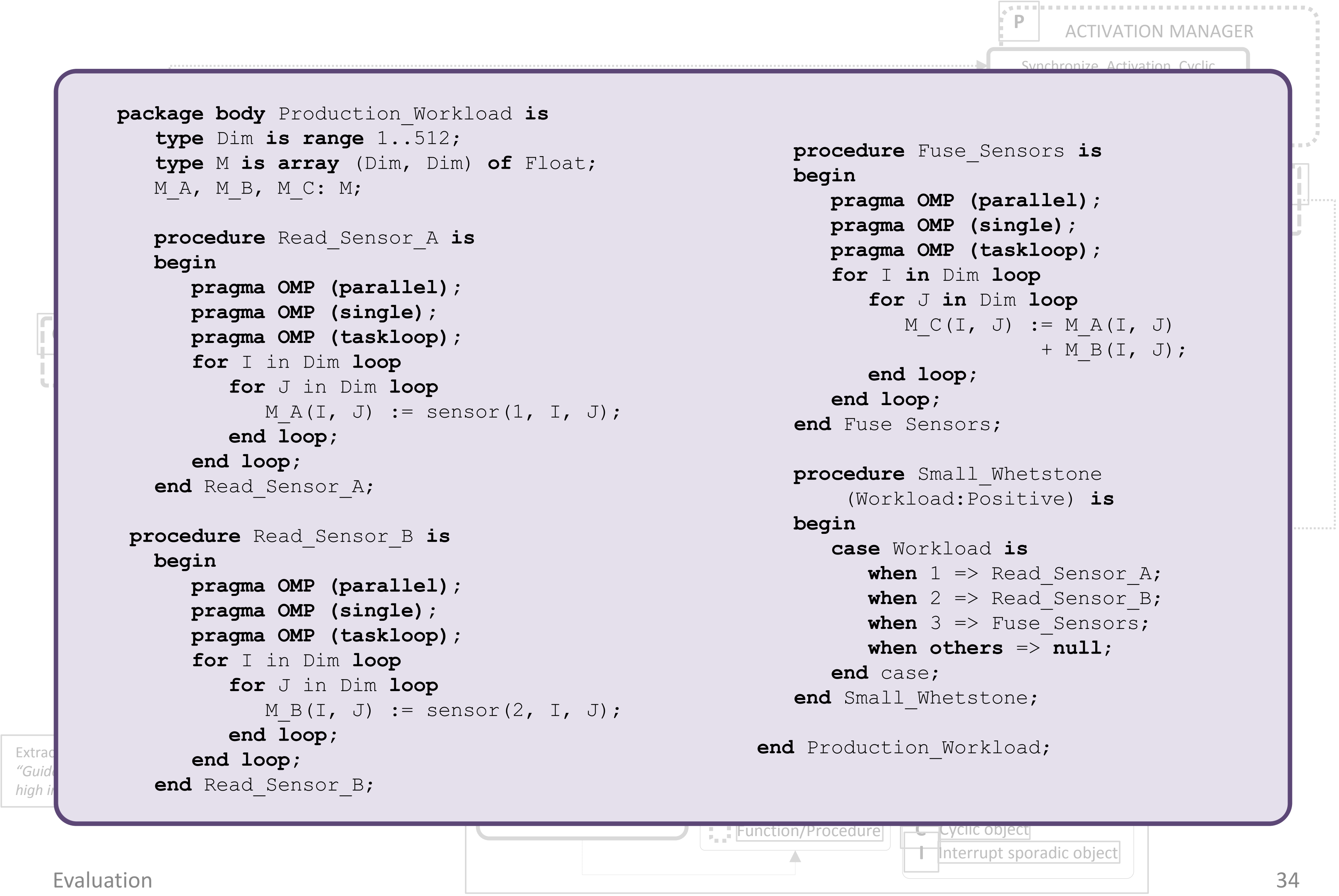
# *Ravenscar* application (HRT-HOOD)



Extracted from: Burns, B. Dobbing and T. Vardanega, *"Guide for the use of the Ada Ravenscar Profile in high integrity systems"*, 2003.



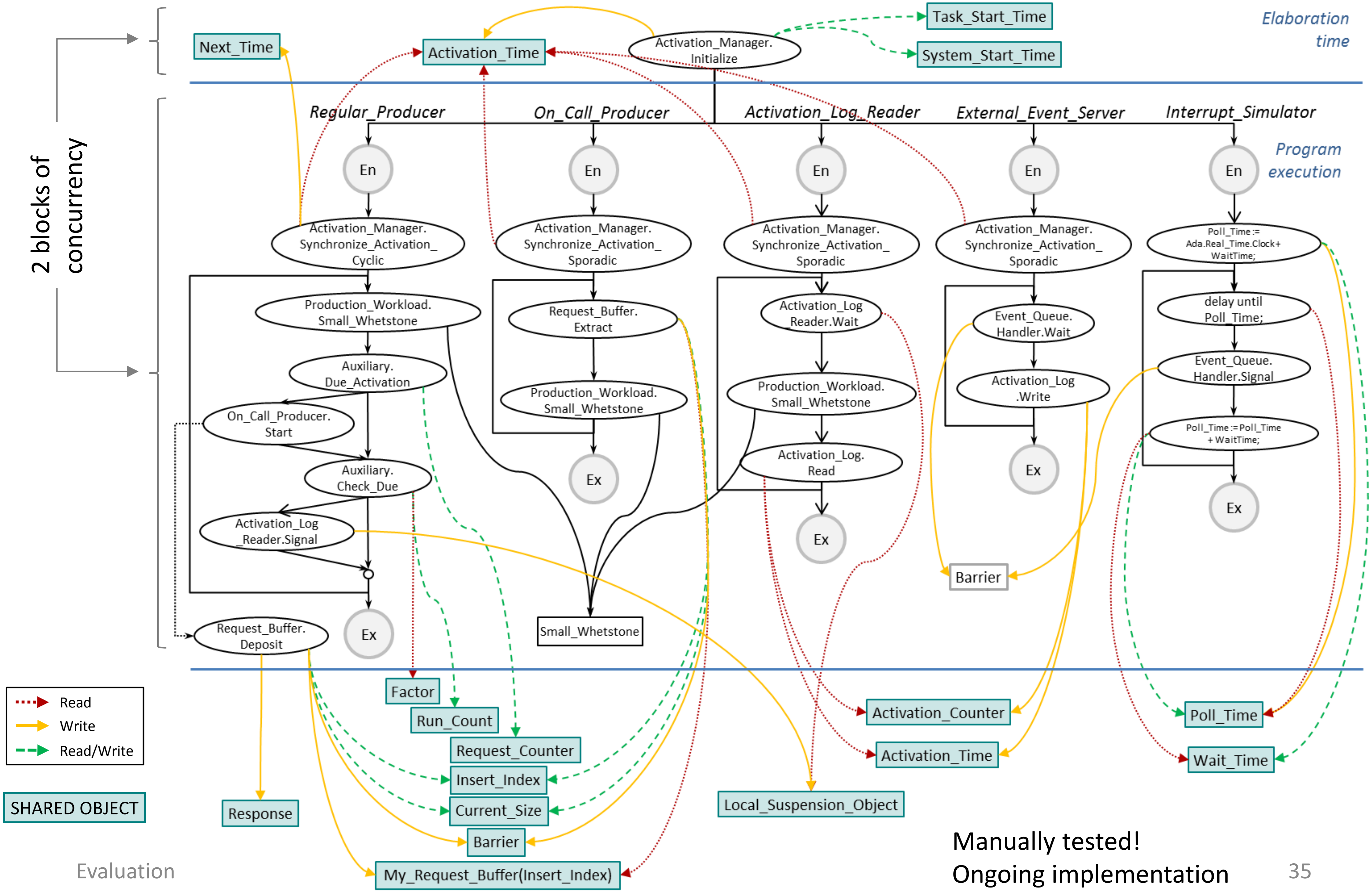
# Evaluation: *Ravenscar* application (HRT-HOOD)



```
package body Production_Workload is
   type Dim is range 1..512;
   type M is array (Dim, Dim) of Float;
   M_A, M_B, M_C: M;

   procedure Read_Sensor_A is
   begin
      pragma OMP (parallel);
      pragma OMP (single);
      pragma OMP (taskloop);
      for I in Dim loop
         for J in Dim loop
            M_A(I, J) := sensor(1, I, J);
         end loop;
      end loop;
   end Read_Sensor_A;

 procedure Read_Sensor_B is
   begin
      pragma OMP (parallel);
      pragma OMP (single);
      pragma OMP (taskloop);
      for I in Dim loop
         for J in Dim loop
            M_B(I, J) := sensor(2, I, J);
         end loop;
      end loop;
   end Read_Sensor_B;
```

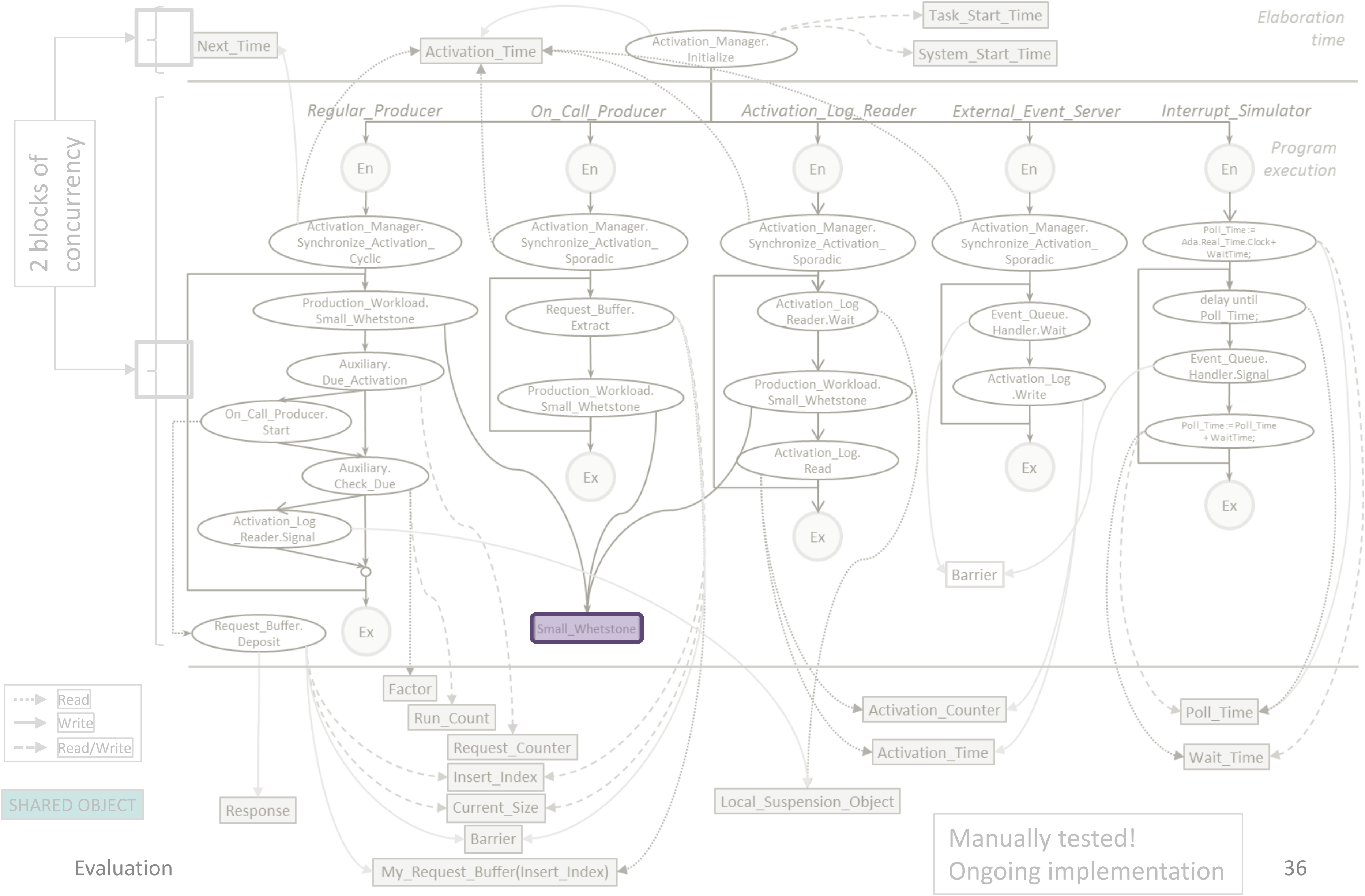
```
   procedure Fuse_Sensors is
   begin
      pragma OMP (parallel);
      pragma OMP (single);
      pragma OMP (taskloop);
      for I in Dim loop
         for J in Dim loop
            M_C(I, J) := M_A(I, J)
                       + M_B(I, J);
         end loop;
      end loop;
   end Fuse Sensors;

   procedure Small_Whetstone
       (Workload:Positive) is
   begin
      case Workload is
         when 1 => Read_Sensor_A;
         when 2 => Read_Sensor_B;
         when 3 => Fuse_Sensors;
         when others => null;
      end case;
   end Small_Whetstone;

end Production_Workload;
```

# *Ravenscar* application (PCFG)



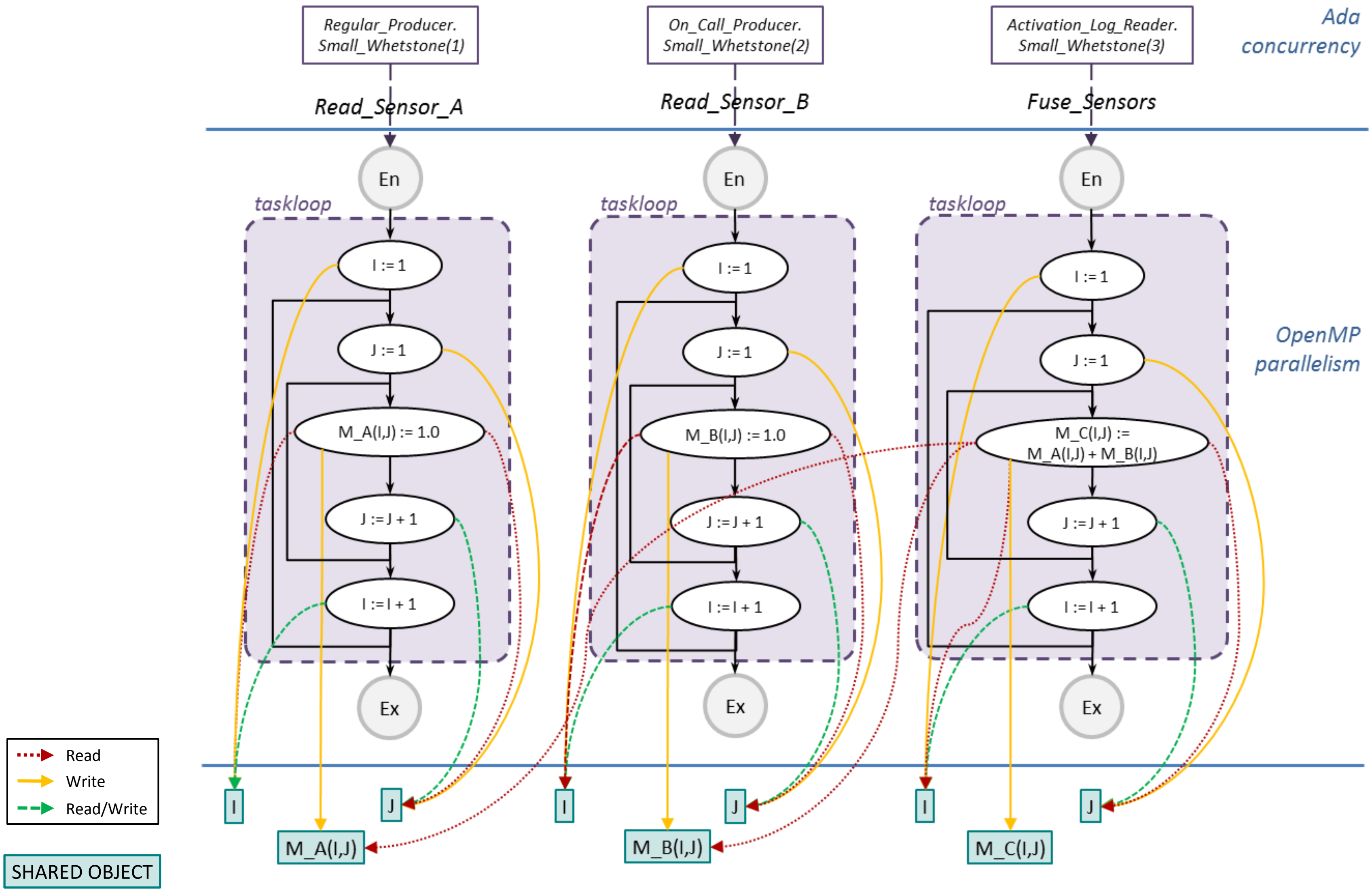
Manually tested!
Ongoing implementation

# *Ravenscar* application (PCFG)



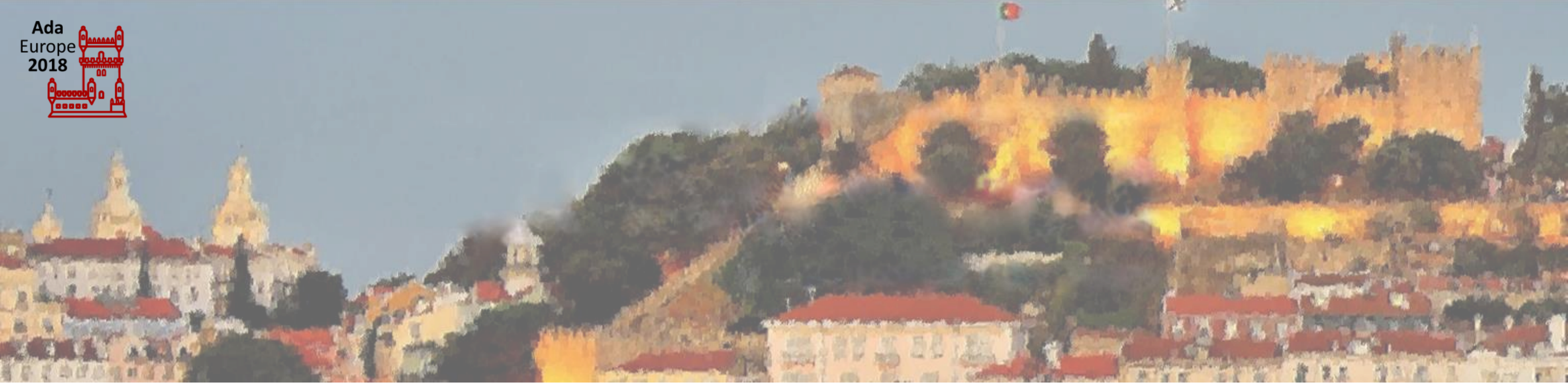
Manually tested!
Ongoing implementation

# *Ravenscar* application (PCFG)



# Conclusions

- Ada moves towards introducing fine-grain mechanisms for parallel execution
- The tasklet model covers some important aspects but has several limitations that may be overcame by OpenMP
- Mixing Ada with OpenMP introduces complexities for static analysis because it mixes concurrency with parallelism
- Ada lacks mechanisms for data-race detection and OpenMP mechanisms only consider parallelism
- OpenMP mechanisms can be used by properly representing concurrency in the PCFG
- Non-Ravenscar applications can be tackled by further enriching the PCFG



Ada Europe 2018

# Safe Parallelism

## Compiler Analysis Techniques for Ada and OpenMP

Reach us at:

| | |
|---|---|
| Sara Royuela | sara.royuela@bsc.es |
| Eduardo Quiñones | eduardo.quinones@bsc.es |
| Luis Miguel Pinho | lmp@isep.ipp.pt |

UPC UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

CISTER - Research Centre in Real-Time & Embedded Computing Systems

BSC Barcelona Supercomputing Center
Centro Nacional de Supercomputación