# Update on Cheddar : reviewing Multi-Core and ARINC653 scheduling features, software design exploration

**P. Dissaux*, J. Legrand*, A. Schach*, S. Rubini+, J. Boukhobza+, L. Lemarchand+, J.P. Diguet+, N. Tran+, M. Dridi+, R. Bouaziz$, F. Singhoff (speaker)+**

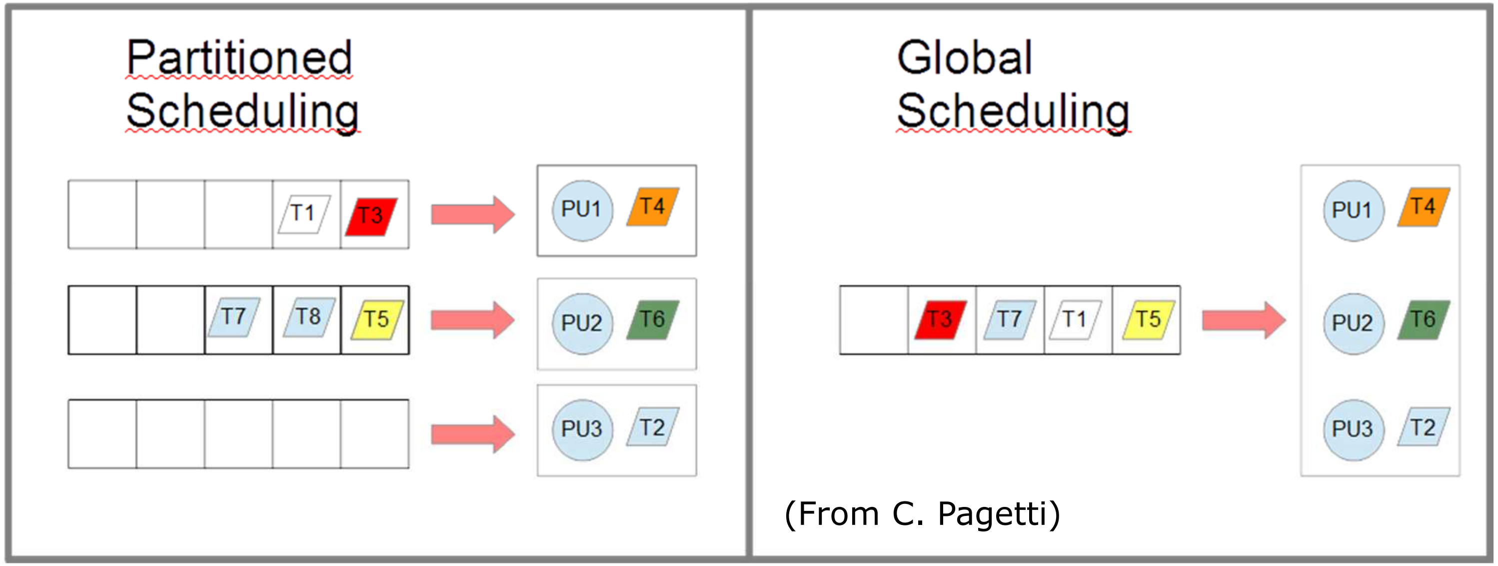
*** Ellidiss Technologies**
**+ Lab-STICC UMR CNRS 6285/UBO**
**$ ReDCAD Laboratory, University of Sfax**

# Multiprocessor sheduling analysis with AADLInspector/Cheddar

- **SMART project (completed in 2014):**
  - Define typical multiprocessor architectures AADLInspector should support (pattern)
  - How to model multiprocessor architectures with AADL
  - Choose or design new scheduling analysis methods for those patterns
  - Prototyping in Cheddar, to be available in AADLInspector

- **Main outcomes:**
  1. Implementation of partitioned and global scheduling methods
  2. Support of shared resources between processing units
  3. Design of partitioning algorithms

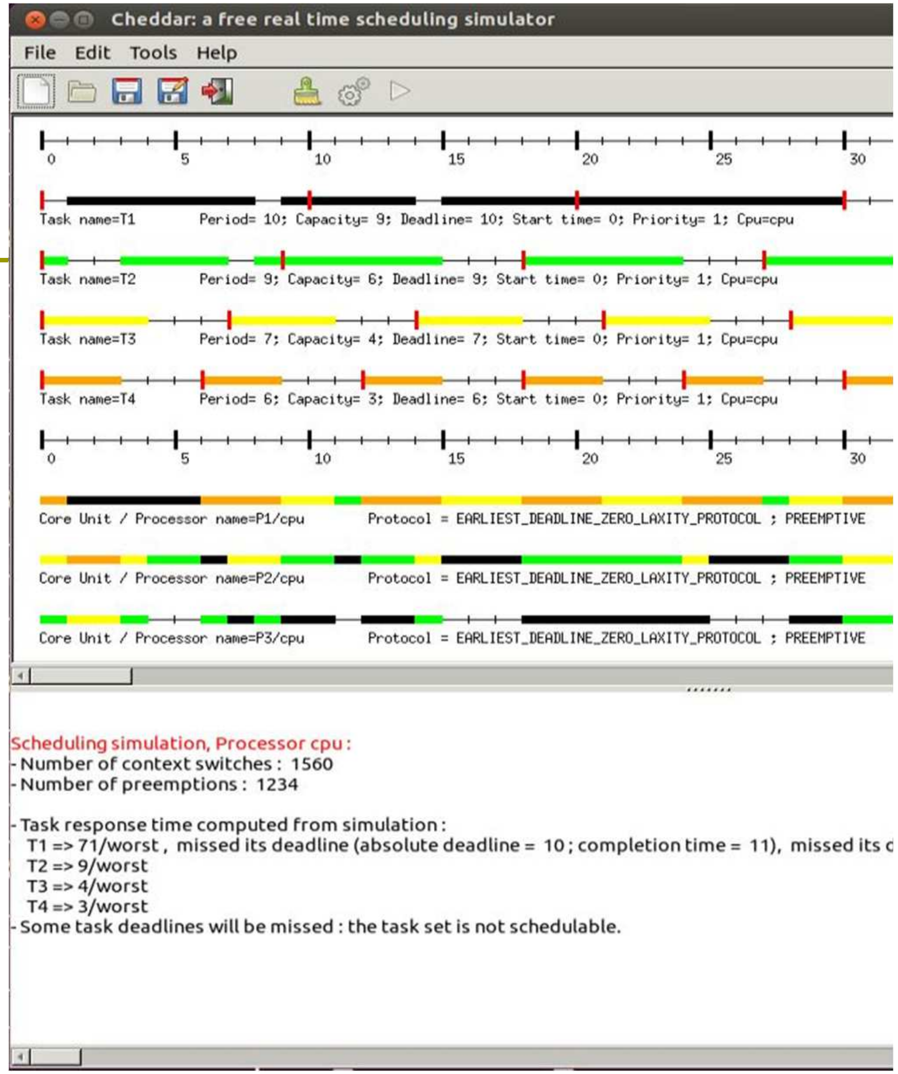
# Typical multiprocessor scheduling analysis: partitioned vs global




- **Partitioned scheduling :** first assign off-line each task on a processing unit ; each processing unit schedules its own task set.
  - No migration. Both on-line and off-line.
- **Global scheduling:** choose the next task to run on any available processing unit (or preempt if all busy).
  - With migration. Fully on-line.

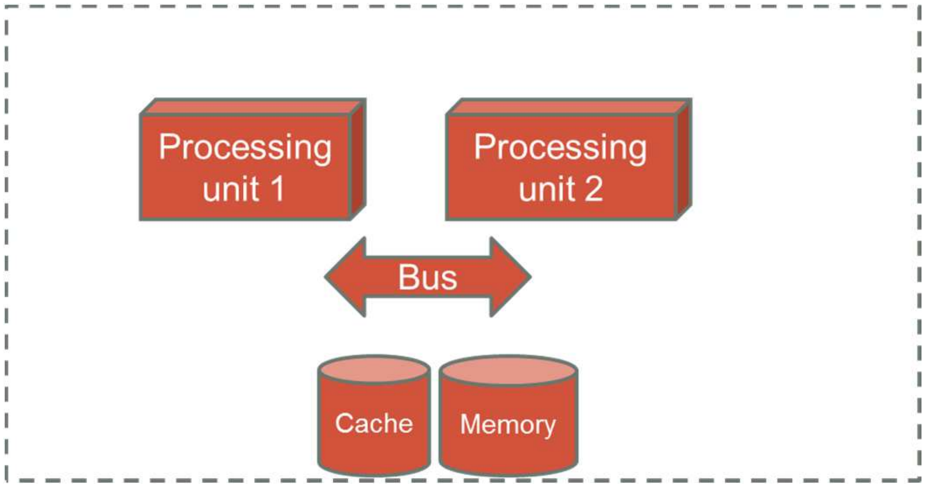
# Typical multiprocessor scheduling analysis: partitioned vs global

- **AADLInspector 1.6 :**
  - Partitioned scheduling only
  - Classical policies (fixed priority, EDF, including ARINC 653, …)
  - Ravenscar data, data port
  - Scheduling simulation & Response time analysis
  - Partitioning policies: Best fit, First Fit, Next Fit, GT, SF




- **Cheddar 3.1 only (not in AI yet):**
  - Global scheduling : any uniprocessor policies + specific policies such as EDZL, LLREF, Pfair,
  - Partitioning policies based on PAES (Pareto Archived Evolution Strategy)
  - Hardware shared resources support

# Shared resources between processing units




- Shared resources: Cache units, bus, NoC, …
- Interferences due to processing units shared resources, make thread WCET (Worst Case Execution Time) difficult to compute
- Specific scheduling methods

# Cache and CRPD

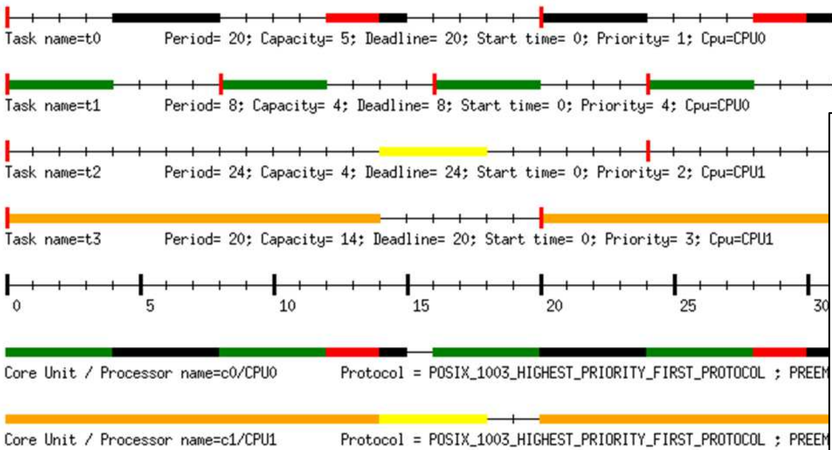
- In fixed priority preemptive scheduling context, tasks can preempt and evict data of other tasks in the cache.
- Cache related preemption delay **(CRPD)**: additional time to refill the cache with the cache blocks evicted by the preemption.

- **Some issues:**
  - CRPD is high, non-negligible preemption cost. It can present up to 44% of the WCET of a task (Pellizzoni et al., 2007)
  - CRPD is difficult to accurately compute off-line (worst case bound, number of preemption)
  - Classical scheduling analysis results cannot be applied with CRPD
    - Applying Rate Monotonic priority assignment algorithm may lead to unschedulable task set
    - Need new priority assignments taking CRPD into account

# Cache/CRPD-Aware Priority Assignment Algorithms

- Extend Audsley's priority assignment algorithm (Audsley, 1995) to take into account CRPD.
- CRPD-aware priority assignment algorithms (**CPA**) that assign priority to tasks and verify theirs schedulability.
- 4 algorithms with different levels of schedulability efficiency and complexity.
- Implemented into Cheddar 3.1, not available with AADLInspector 1.6

# Cache-Aware Scheduling Simulation




**Scheduling analysis for systems with cache**

| | | |
|---|---|---|
| Cache Configuration | Control Flow Graph | Worst-Case Execution Time |
| Memory Layout | Cache Access Profile | Scheduling Policy |

- **Problem Statement:**
  - Theoretical issues with CRPD : feasibility interval, sustainability
  - Various parameters need to be taken into account in scheduling analysis of systems with cache: cache profile, memory layout, CFG
- **Outcomes:**
  - We have designed a new CRPD computation model, sustainable for L1 instruction cache. Feasibility interval proved.
  - Extending Cheddar to model cache/cache access profile

# Summary

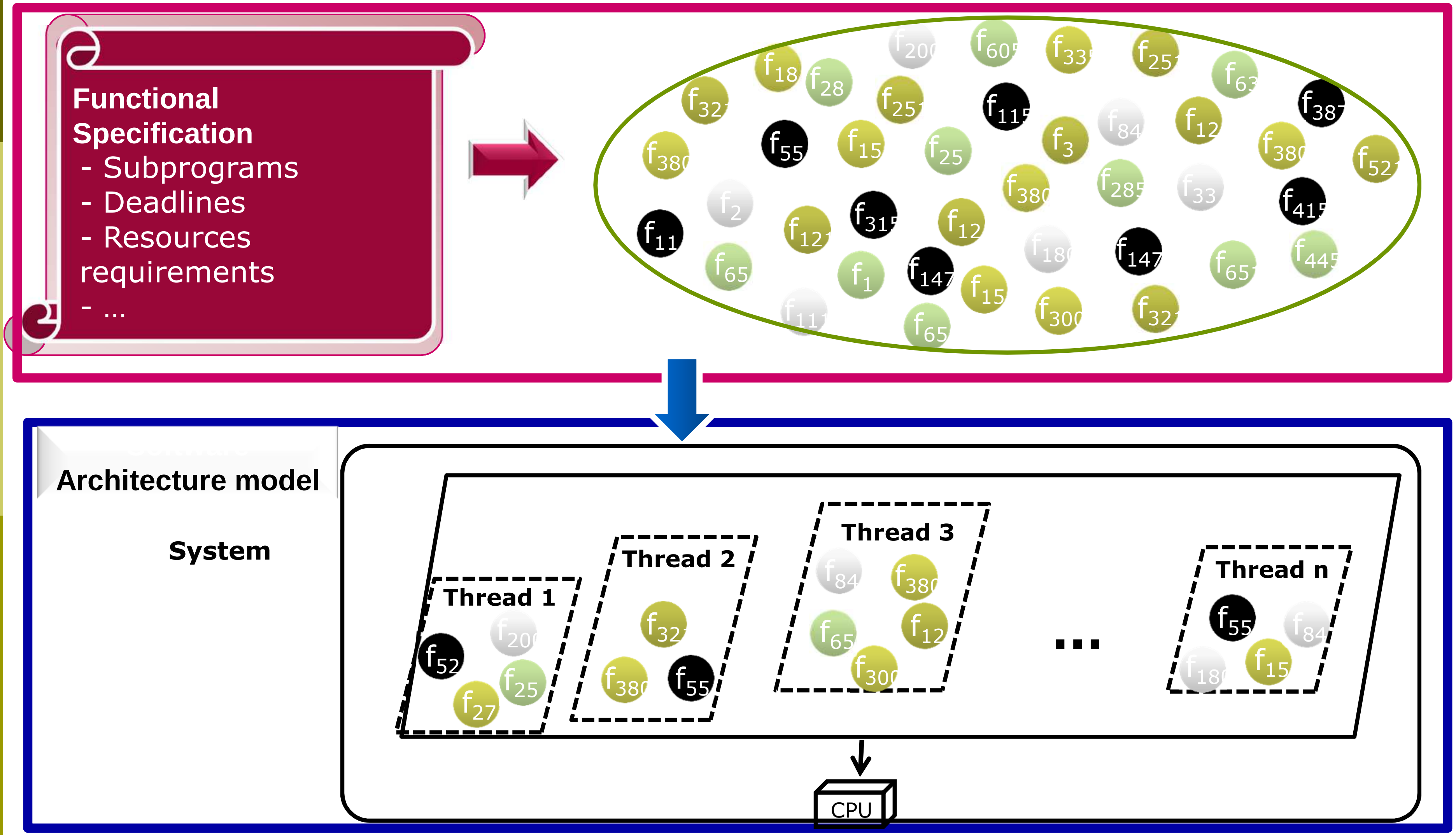
1. Multiprocessor scheduling analysis features
2. Software design space exploration : partitioning with competing objective functions

# Cheddar & partitionning with competing objective functions

- **Problem statement :**
  - Performances (scheduling), is not the unique concern
  - Trade-offs with several competing criteria/objective functions such as performances vs safety vs security
  - How to do partitionning in this context ?
  - PAES helps ? PAES with Cheddar ?

- **Small example to illustrate, assume:**
  - A system running several sub-programs (i.e. functionnal units)
  - Subprograms may shared resources (compliant with Ravenscar)
  - How to assign subprograms to threads

# From the functional specification to a software architecture



# Competing objective functions in software design space exploration

**Subprograms to threads/tasks assignment**

**One subprogram = one thread/task**

- Timing overhead (i.e. preemption cost)
+ Increase task/function laxities

**=> maximize(laxities)**

**Many subprograms = one task/thead**

- Decrease tasks/functions laxities
+ Low timing overhead

**=> minimize(#preemptions)**

Explore several assignment solutions

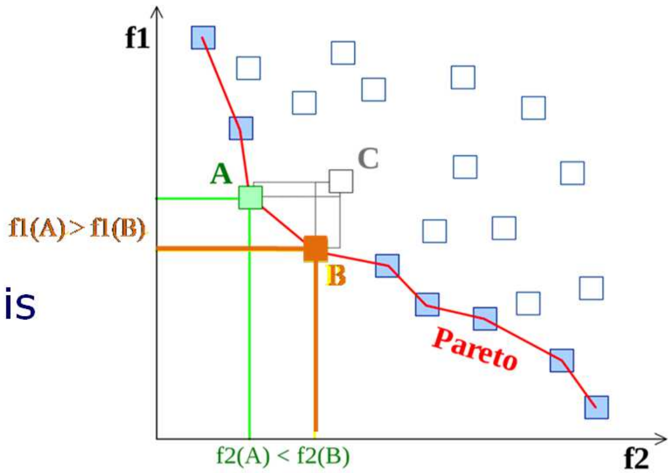
Select assignment solutions that meet at **best the trade-offs between number of preemptions and laxities**

# PAES : a multi-objectives metaheuristic

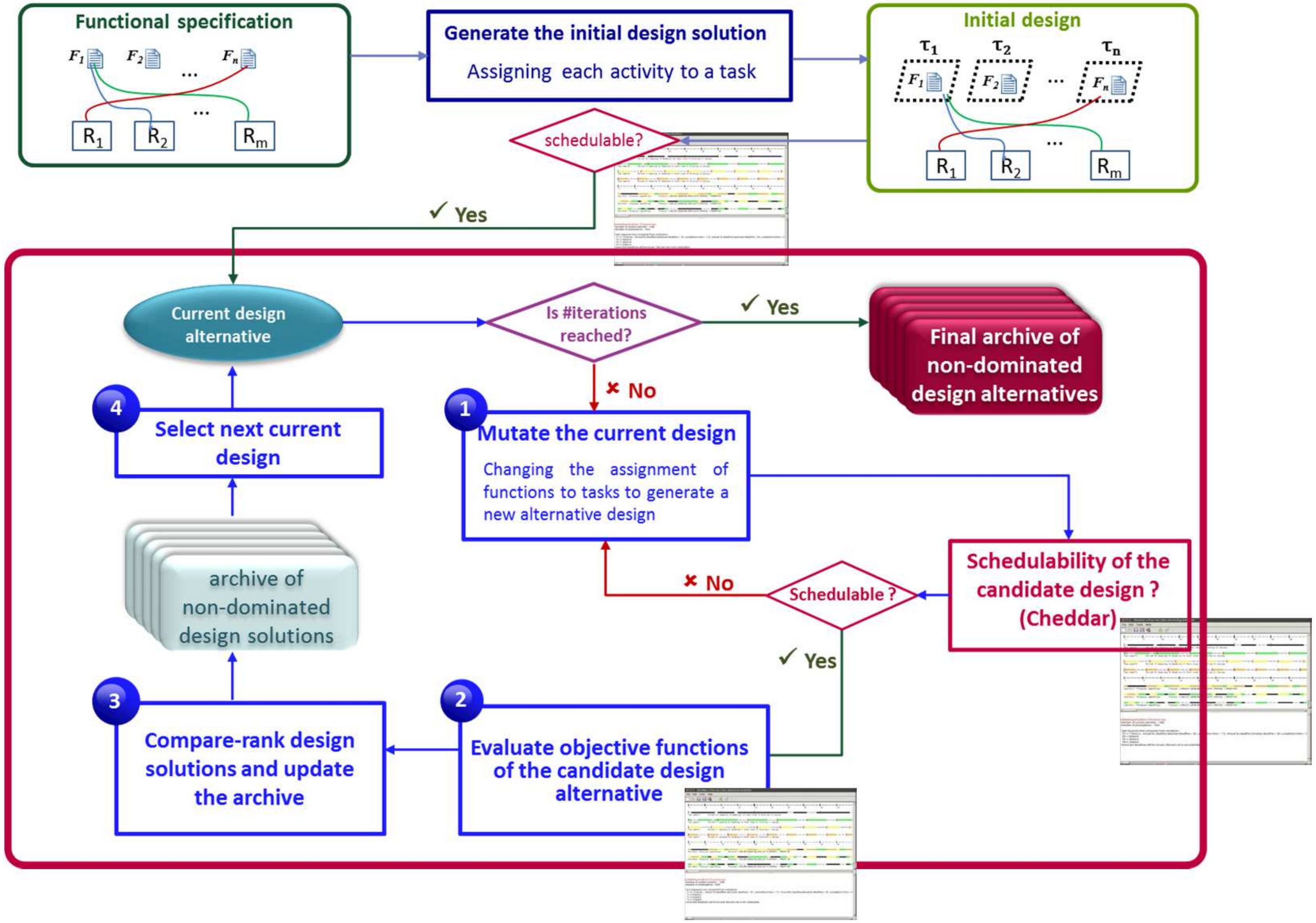
## ❑ Basic steps of PAES algorithms:

1. **Mutate** a solution to generate a new candidate: small change to move from a solution to a nearby neighbour
2. **Evaluate** the mutated solution (conflicting objective functions)
3. **Update** non-dominated solutions set (i.e. archive)
4. **Select** new solution for next iteration : mutated or current solution

- **Pareto Front**: final set of non-dominated solutions
- Solutions A dominates solution C because it is better than C for all objectives



# PAES-based partitioning



# Competing Performance Criteria in the Software Design Space exploration

- **Examples of investigated trade-offs with competing objectives functions such as:**
  - Min (#premptions)
  - Max (laxities)
  - Min (Ravenscar data blocking time)
  - …

  ⇒Performance competing objectives functions only

- **How to be sure that objective functions are competing?**

# Conclusion

- **Multiprocessor scheduling analysis of AADLInspector & Cheddar:**
  - Bunch of classical partitioned vs global scheduling algorithms
  - Shared hardware resources: cache, NoC

- **Multi-objective partitioning**
  - PAES based, for Ravenscar compliant architecture
  - Safety & performance & security objective functions
  - Follow Security annex