Generating Worst-case Schedules with Constrained Optimization

An Approach to Support Software Performance Testing

Stefano Di Alesio [1,2]

Shiva Nejati [2]

Lionel Briand [2]

Arnaud Gotlieb [1]

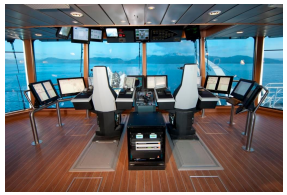
ICS 2015

Richmond, 11/01/2015

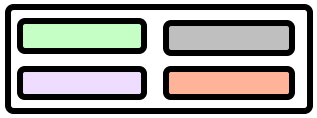
[1] Certus Centre for Software V&V
Simula Research Laboratory
Norway

[2] Interdisciplinary Centre for Reliability, Security and Trust (SnT)
University of Luxembourg
Luxembourg

We present a Constrained Optimization Model to support Performance Testing in RTES



Performance Requirements in Real Time Embedded Systems (RTES)



Generating worst-case schedules: A novel application for COP

Results and Future Directions: Combining CP with GA

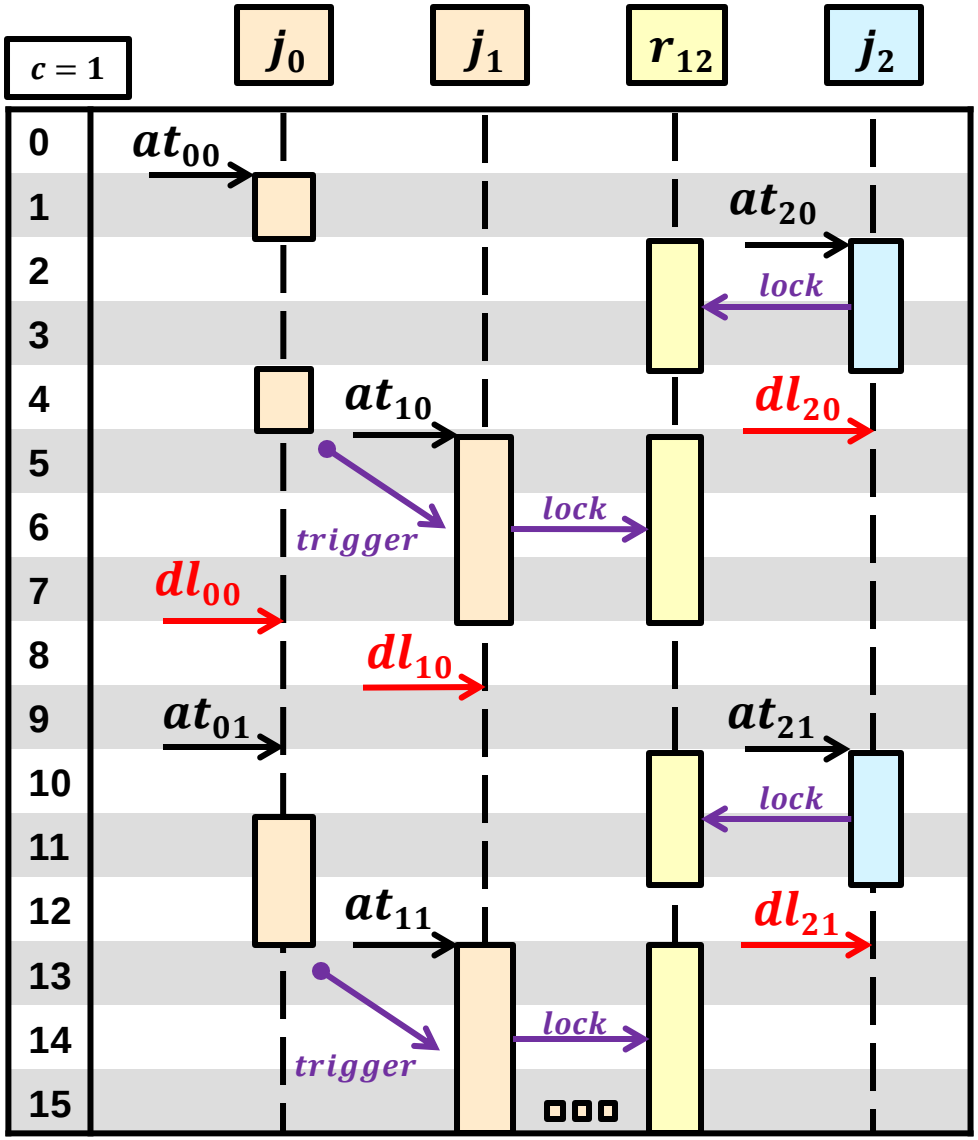
Safety-critical RTES have to meet strict Performance Requirements: they must be thoroughly tested

RTES have concurrent interdependent tasks which have to satisfy Performance Requirements

Each task has a deadline (i.e., latest finishing time) w.r.t. its arrival time

Some task properties depend on the environment, others are design choices

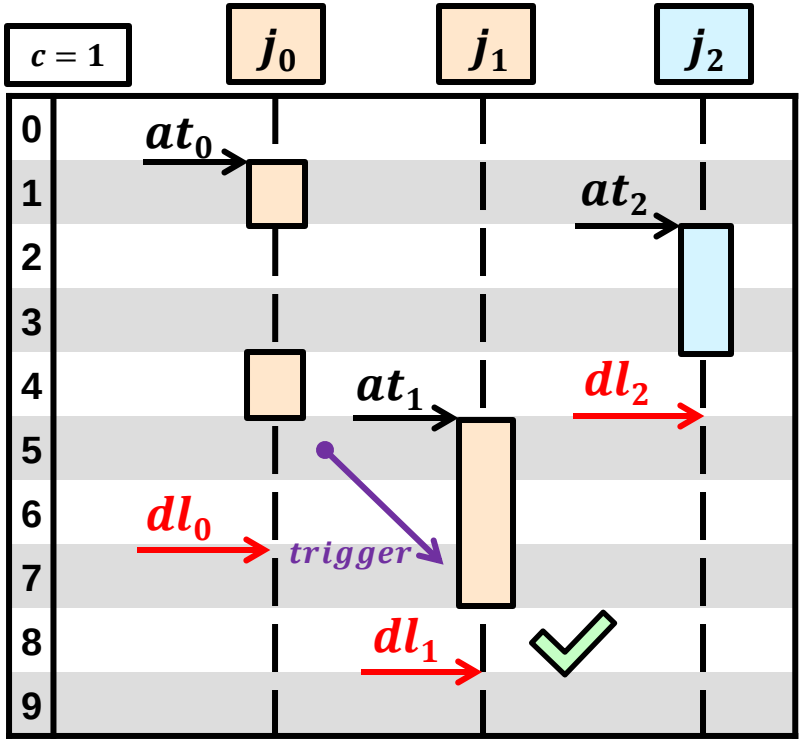
Tasks can trigger other tasks, or share computational resources with them

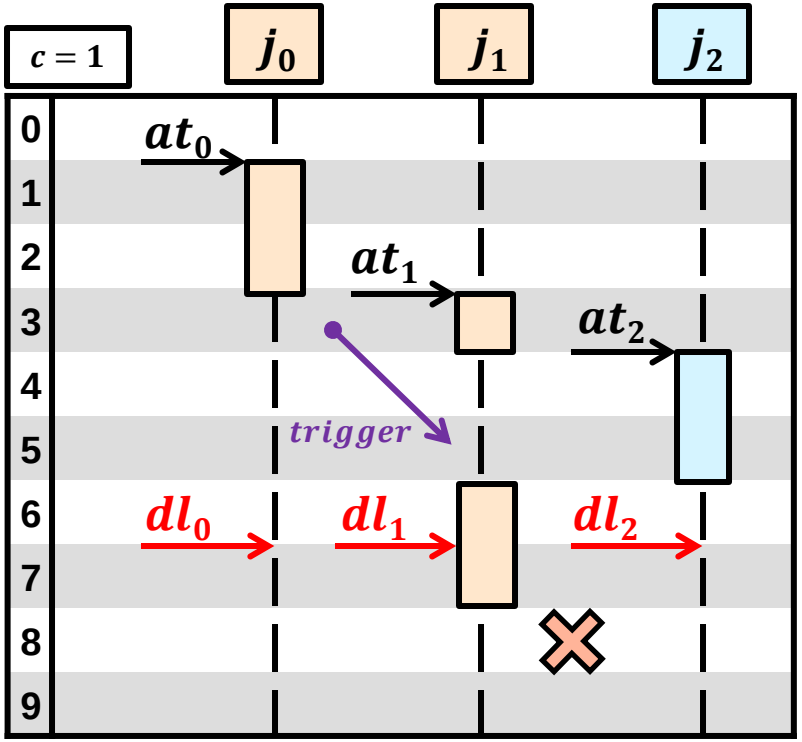


Particular sequences of task arrival times can determine scenarios where Perf. Reqs. are violated

j_0, j_1, j_2 arrive at at_0, at_1, at_2 and must finish before dl_0, dl_1, dl_2

j_1 can miss its deadline dl_1 depending on when at_2 occurs!



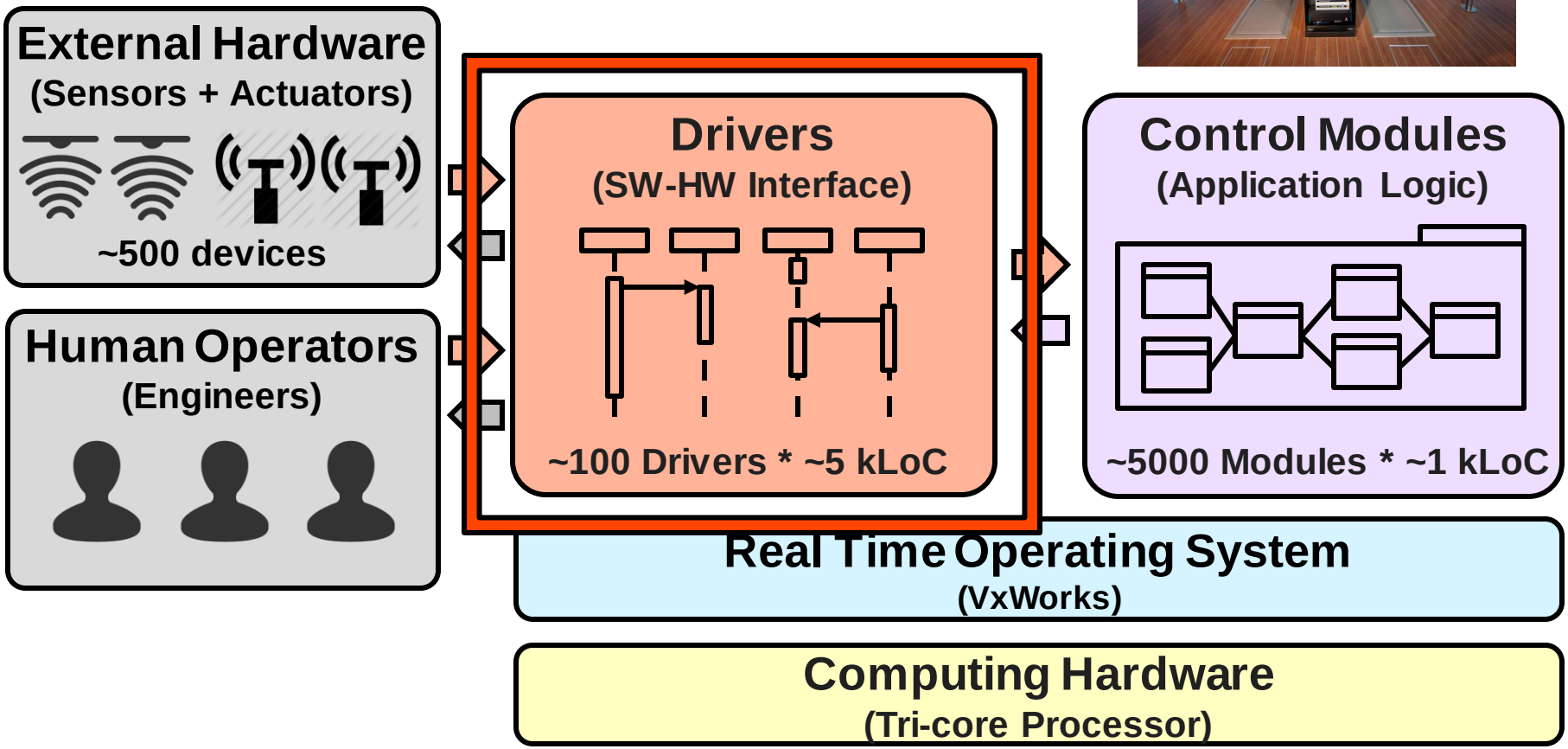


Similar examples can be made for other requirements, e.g., bounds on Response Time and CPU Usage

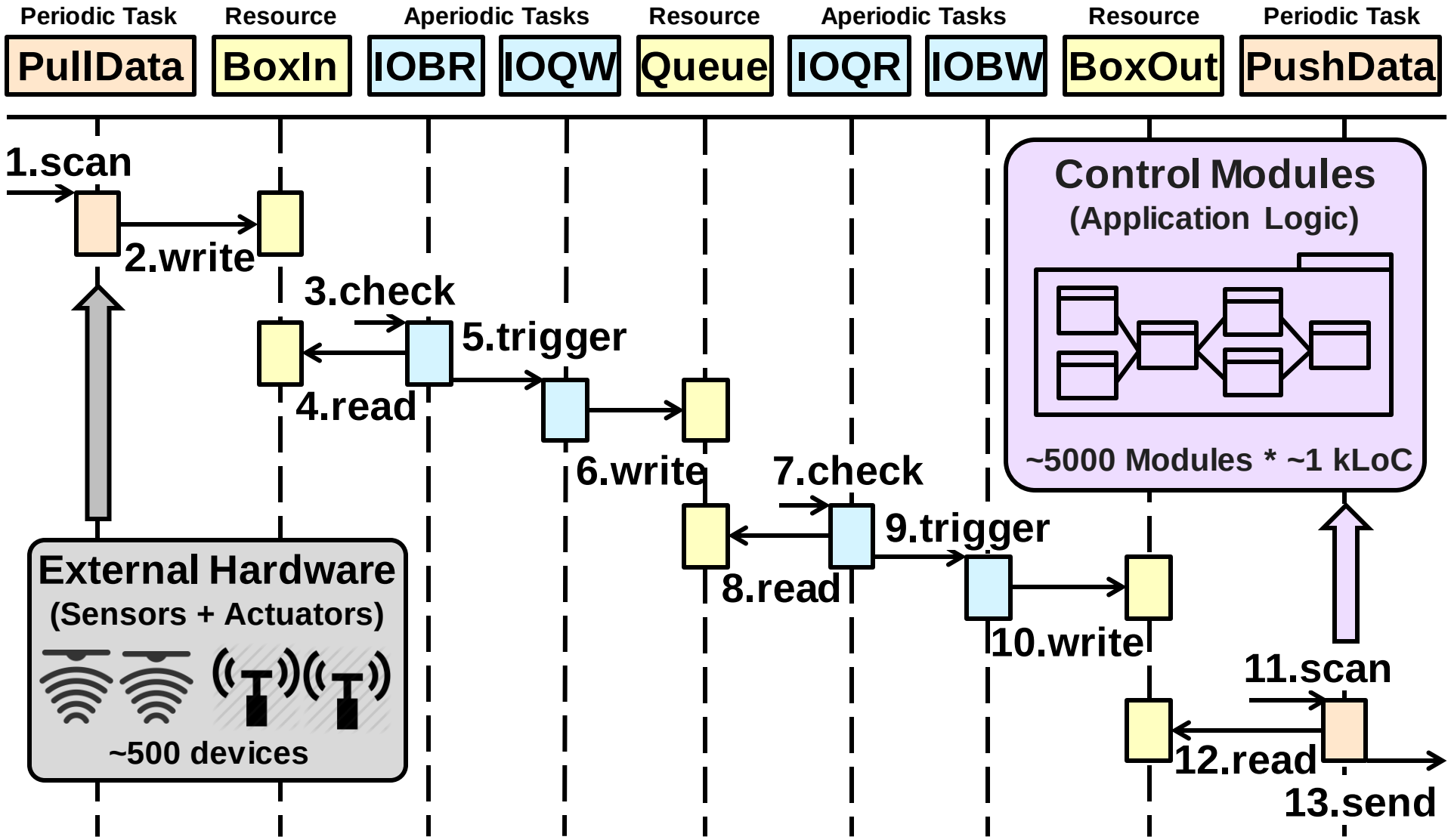
Our case study was a monitoring application for fire/gas leaks detection in offshore platforms

KONGSBERG

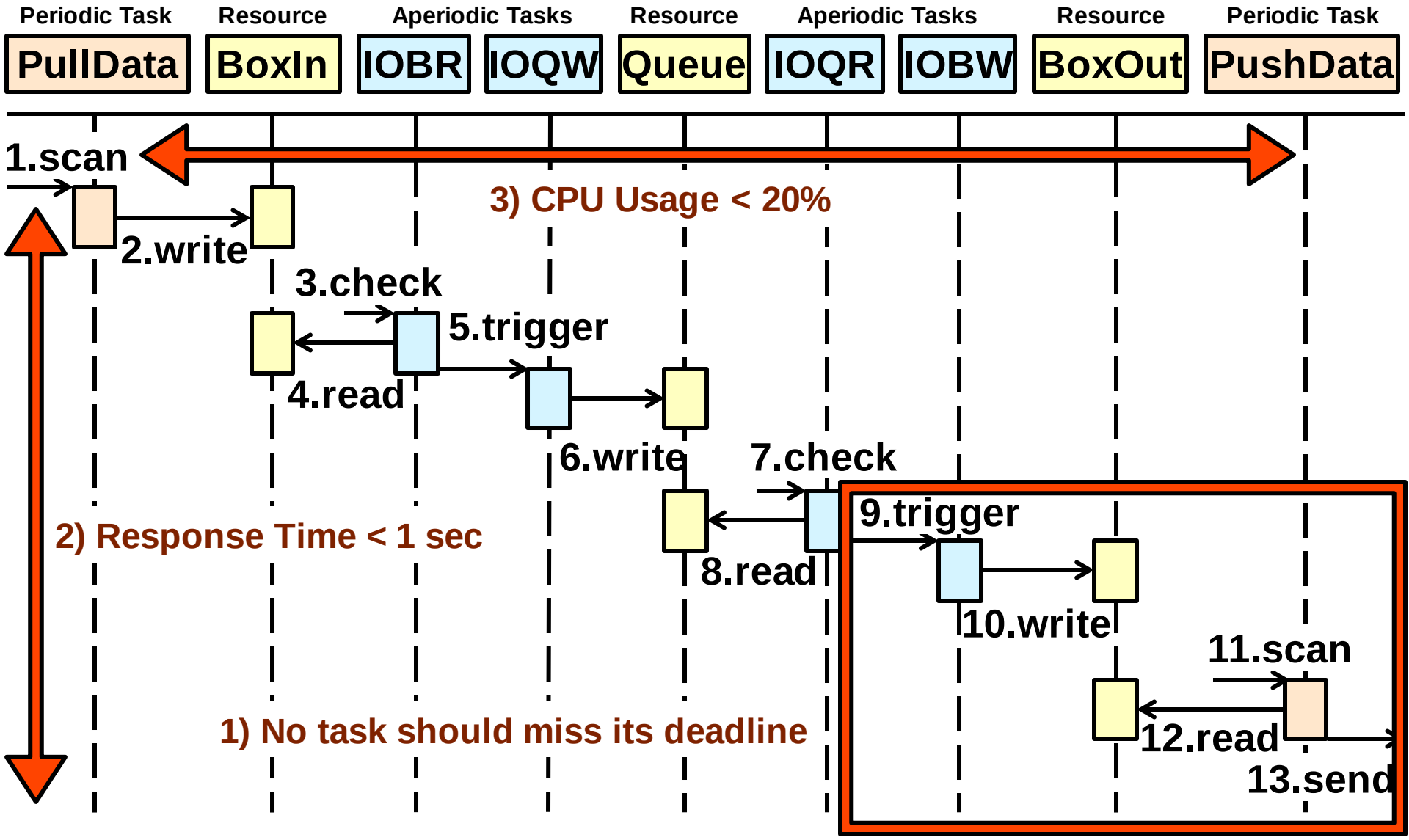
KM: Kongsberg Maritime
FMS: Fire and gas Monitoring System



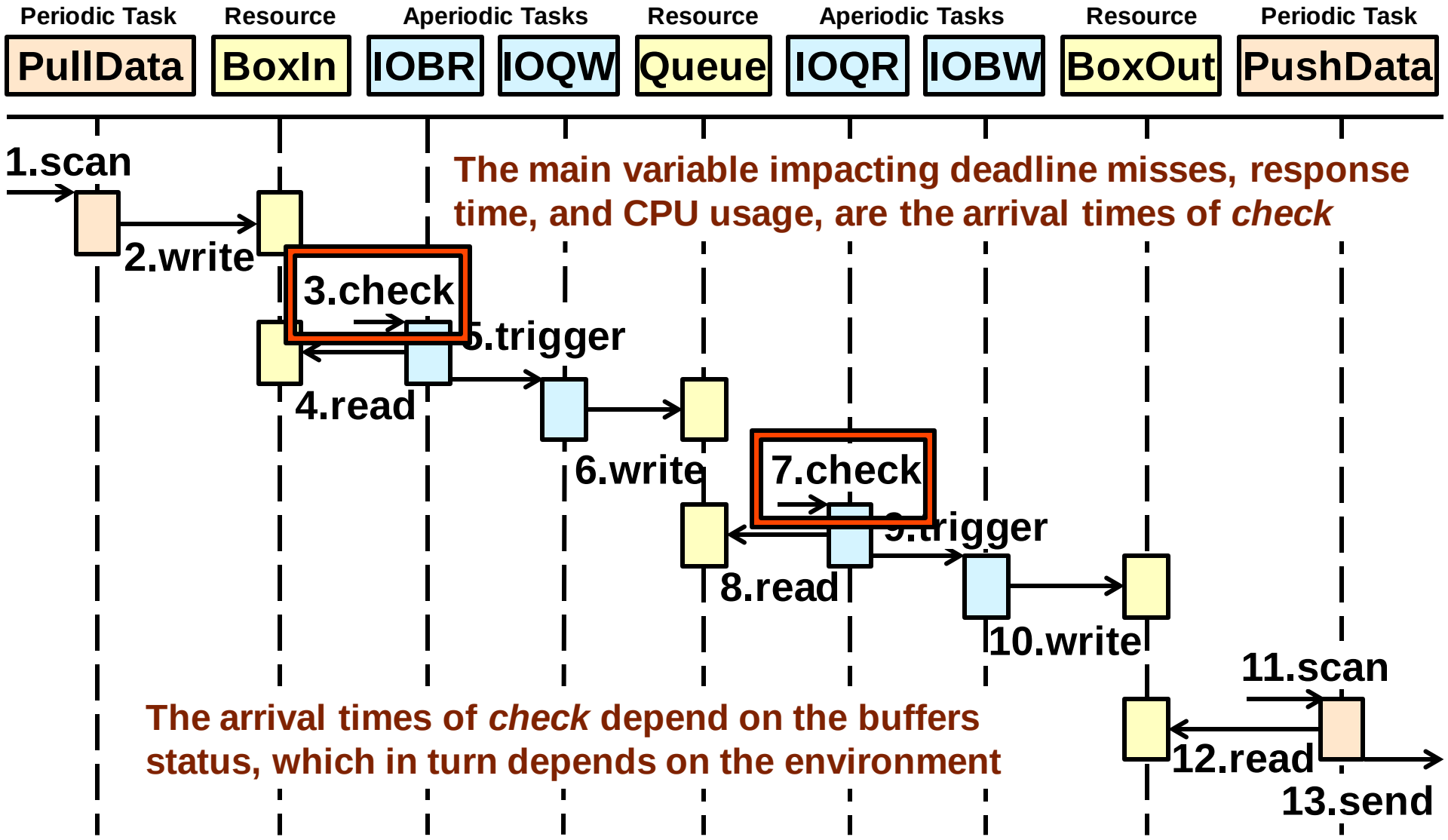
Drivers transfer data between external hardware (sensors and actuators) and control modules



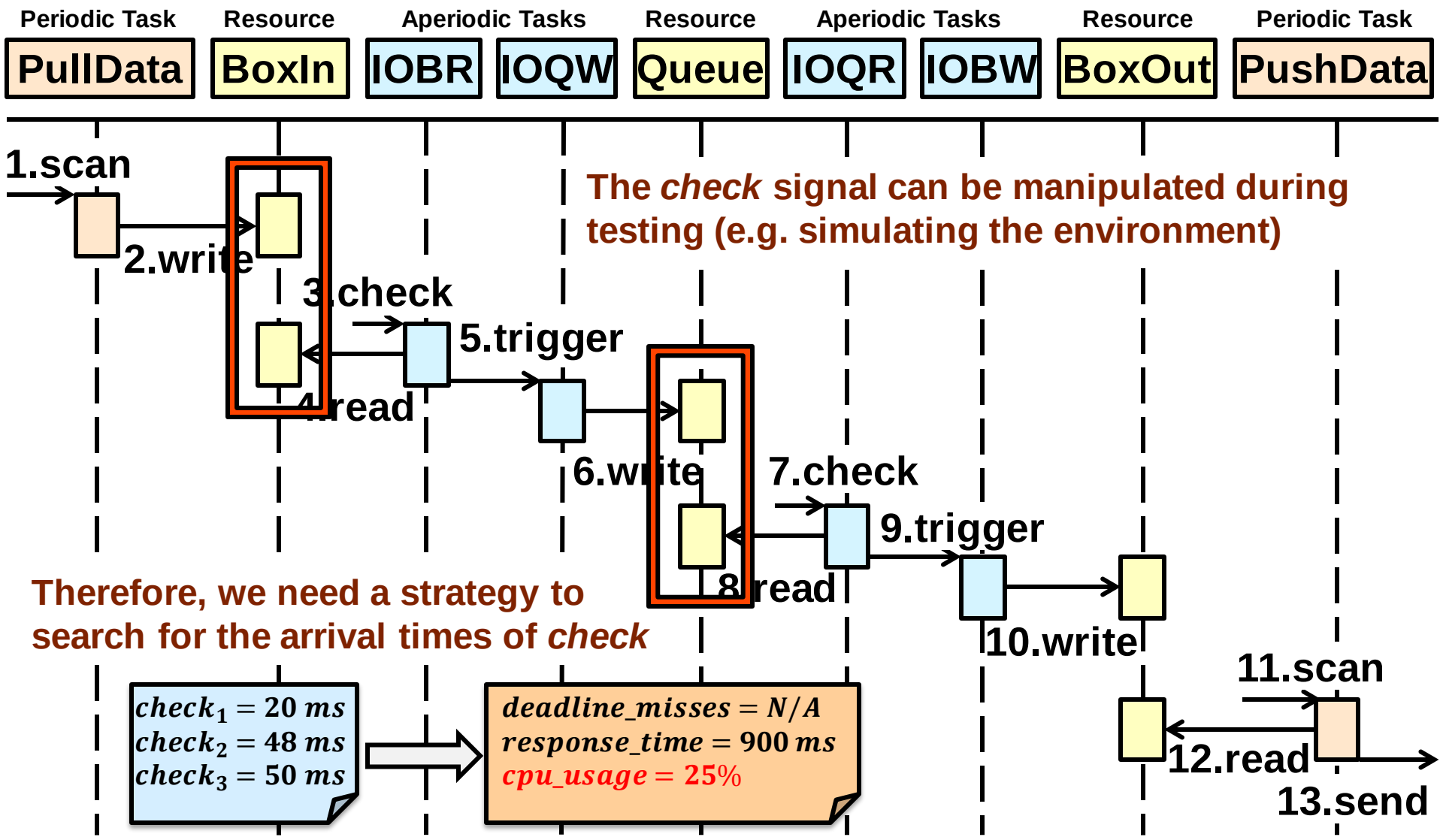
The FMS drivers have performance requirements on task deadlines, response time, and CPU usage



Our goal is to identify worst-case scenarios w.r.t. deadline misses, response time, and CPU usage



Each worst-case scenario can characterize a test case in terms of the arrival times of *check*



Several techniques have been used for solving this problem, but each has its own drawbacks

SE	Formal Verification		Testing	
	Scheduling Theory	**Model Checking**	**Performance Engineering**	**Genetic Algorithms**
Basis	Mathematical Theory	System Modeling	Practice and Tools	System Modeling
Background	Queuing Theory	Fixed-point Computation	Profiling, Benchmarking	Metaheuristics
Key Features	Theorems	Symbolic Execution	Dynamic Analysis	Randomized Search
Drawbacks	Assumptions, Multi-Core	Complex Modeling	Non Systematic	Low Effectiveness

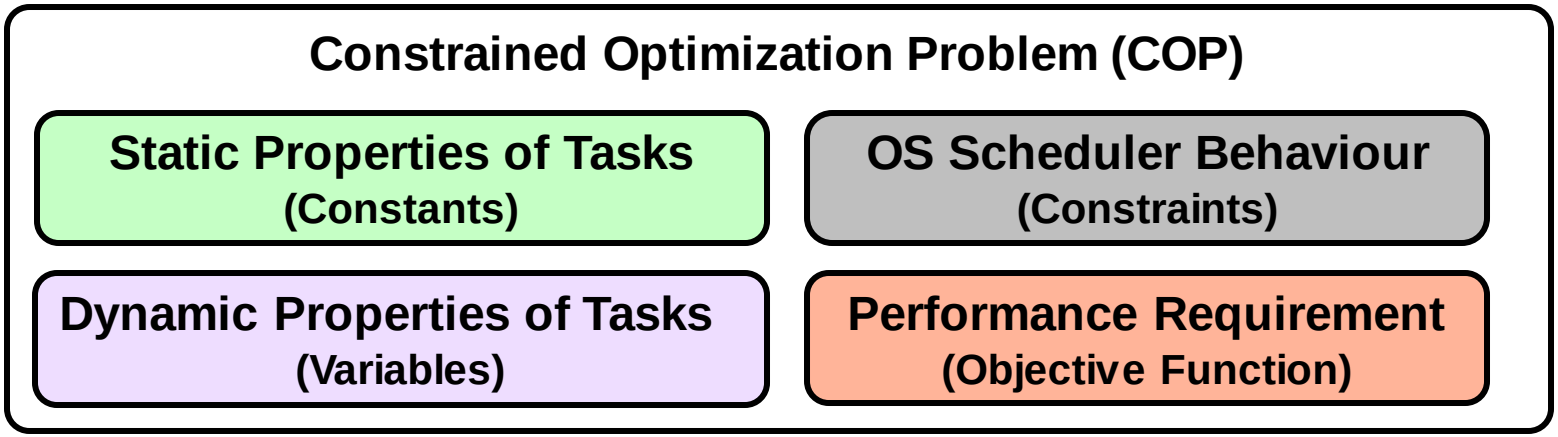
CP

Job-shop scheduling analysis

Multi-core
Priority
Preemption
Dependency
Triggering

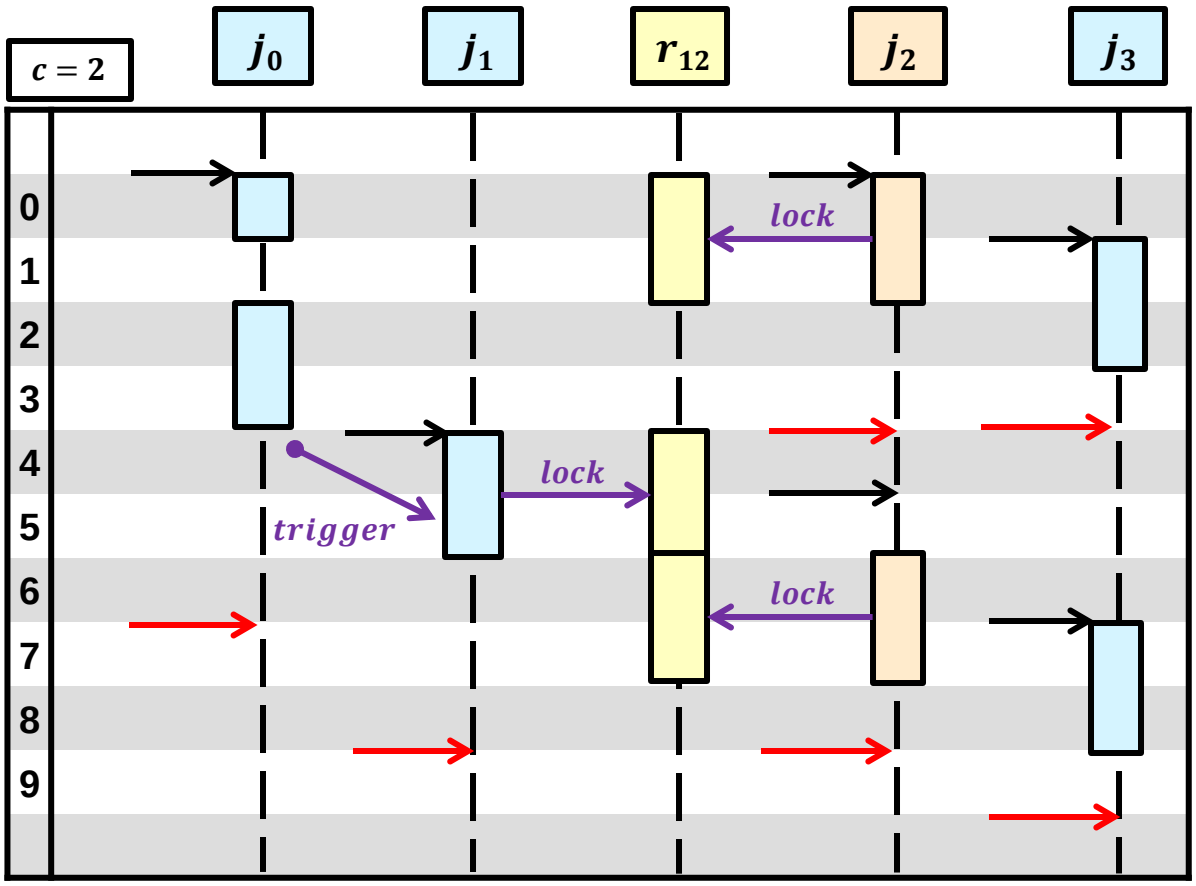
We cast the search for the arrival times of the *check* signal leading to worst-case scenarios as a COP

The COP models a multi-core priority-driven preemptive scheduler with task triggering and dependencies [1,2,3,4]



[1] Di Alesio, S., Gotlieb, A., Nejati, S., and Briand, L. (2012). Testing deadline misses for real-time systems using constraint optimization techniques. In Software Testing, Verification and Validation (ICST), 2012 IEEE Fifth International Conference on, pages 764–769. IEEE.
[2] Nejati, S., Di Alesio, S., Sabetzadeh, M., Briand, L.: Modeling and analysis of CPU usage in safety-critical embedded systems to support stress testing. In: Model Driven Engineering Languages and Systems (MODELS), pp. 759–775. Springer (2012)
[3] Di Alesio, S., Nejati, S., Briand, L., and Gotlieb, A. (2013). Stress testing of task deadlines: A constraint programming approach. In Software Reliability Engineering (ISSRE), 2013 IEEE 24th International Symposium on, pages 158–167. IEEE.
[4] Di Alesio, S., Nejati, S., Briand, L., and Gotlieb, A. (2014). Worst-case scheduling of software tasks – a constraint optimization model to support performance testing. In Principles and Practice of Constraint Programming (CP 2014).

Static Properties depend on the RTES design, and are modeled as Constants

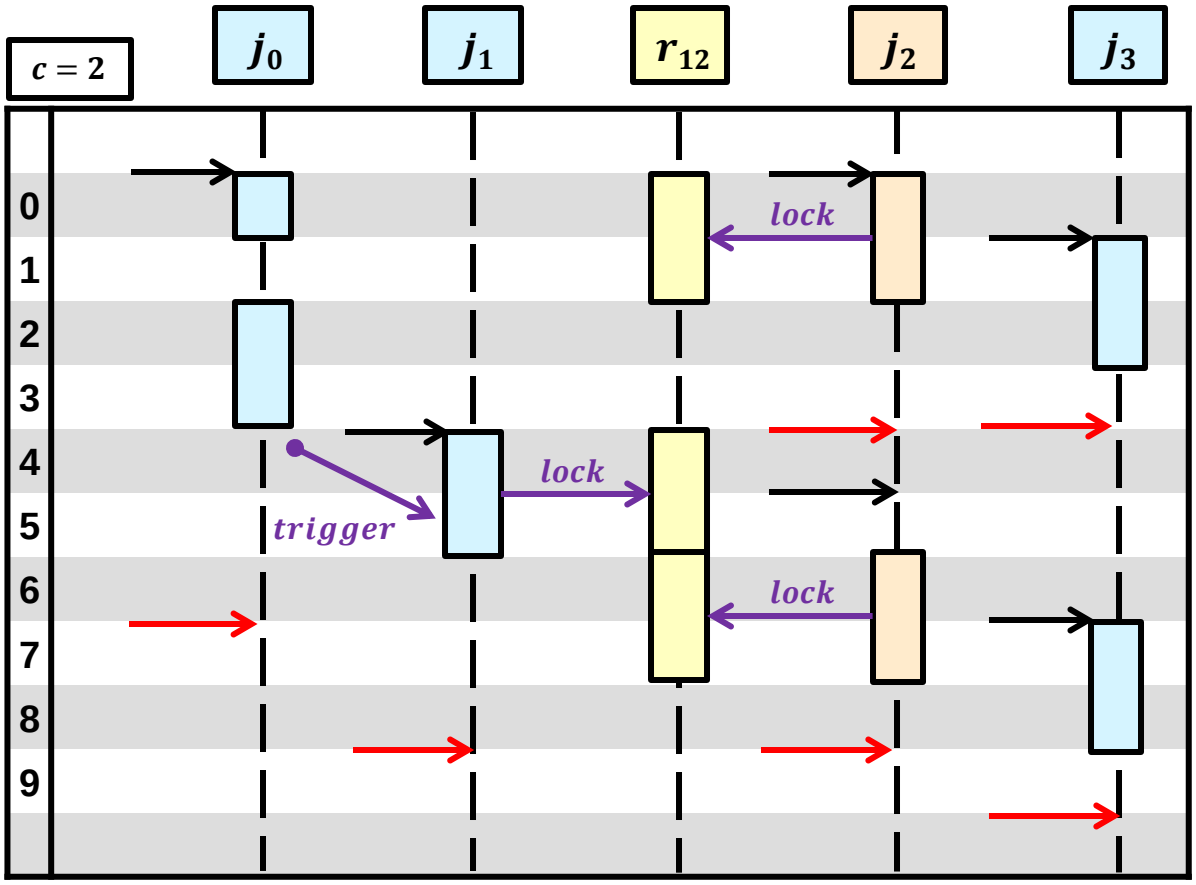


Constants

- Observation Interval: $T = [0, 9]$
- Number of cores: $c = 2$
- Set of Tasks: $J = \{j_0, j_1, j_2, j_3\}$
- Priority of Tasks: $pr(j_i) = i$
- Period of Tasks: $pe(j_2) = 5$
- Min/Max Inter-arrival time of Tasks: $mn(j_0) = 5, mx(j_0) = 10$
- Duration of Tasks: $dr(j_0) = 3$
- Deadline of Tasks: $dl(j_0) = 7$
- Triggering Relation: $tg(j_0, j_1)$
- Dependency Relation: $dp(j_1, j_2)$
- Number of Periodic Task Executions: $te(j) = \left\lfloor \frac{tq}{pe(j)} \right\rfloor$, $te(j_2) = \left\lfloor \frac{10}{5} \right\rfloor = 2$

Time is discretized in our analysis: we solve an IP over finite domains

Dynamic Properties depend on the RTES runtime behavior, and are modeled as Variables



Variables

Independent

- Number of Aperiodic Task Exec.:
  $te(j) \in \left[\frac{tq}{mx(j)}, \frac{tq}{mn(j)}\right]$,
  $te(j_0) \in [1,2], te(j_0) = 1$
- Arrival time of Aperiodic Task Exec.:
  $at(j,k) \in T,\ at(j_0,0) = 0,\ ac(j_3,1) = 7$
- Active time of Task Executions:
  $ac(j,k,p) \in T,\ p \in [0, dr(j) - 1]$,
  $ac(j_0,0,0) = 0,\ ac(j_0,0,1) = 2$

Dependent

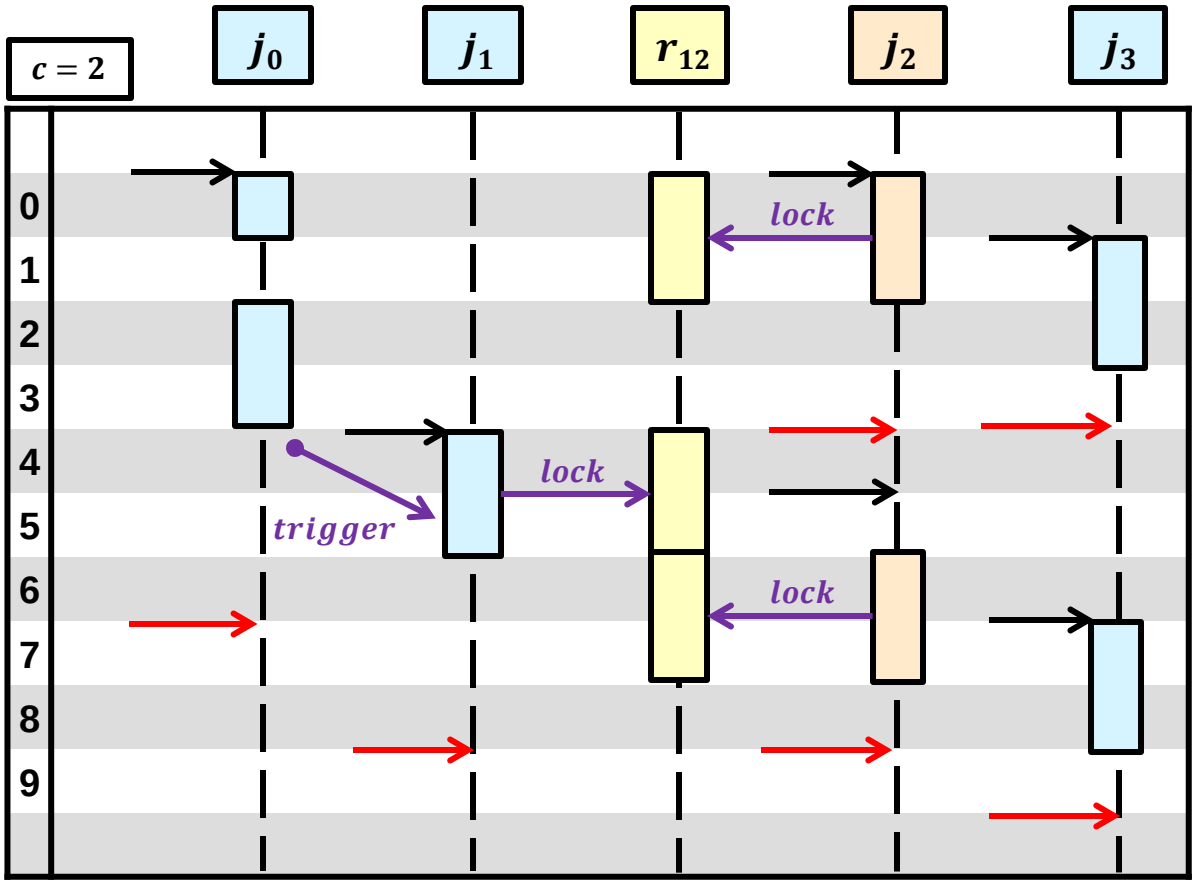
- Set of Aperiodic Task Executions:
  $K_j = [0,\ te(j) - 1], K_{j_0} = [0]$
- Start/End time of Task Executions:
  $st(j,k) = ac(j,k,0),\ st(j_0,0) = 0$,
  $en(j,k) = ac(j,k,dr(j) - 1)$,
  $en(j_0,0) = 3$
- Deadline Miss of Task Executions:
  $dm(j,k) = en(j,k) - at(j,k) - dl(j)$,
  $dm(j_0,0) = 3 - 0 - 6 = -3$
- System Load: $ld(t) = \sum_{j,k,p}(ac(j,k,p) = t)$, $ld(0) = 2$,
  $ld(3) = 1$

te and at of Periodic Tasks Executions are constants:

$$te(j) = \left\lfloor \frac{tq}{pe(j)} \right\rfloor, \qquad te(j_2) = \left\lfloor \frac{10}{5} \right\rfloor = 2$$

$$at(j,k) = k * pe(j), \qquad at(j_2,1) = 1 * 5 = 5$$

The RTES scheduler is modeled through constraints among Static and Dynamic properties (1/2)



Constraints

Well-formedness

- A task cannot start before it has arrived: $at(j,k) \leq st(j,k)$
- A task cannot finish before it has completed: $st(j,k) + dr(j) \leq en(j,k)$
- Arrival times of aperiodic tasks are separated by min/max interarr. times:
$$at(j,k-1) + mn(j) \leq at(j,k) \leq at(j,k-1) + mx(j)$$

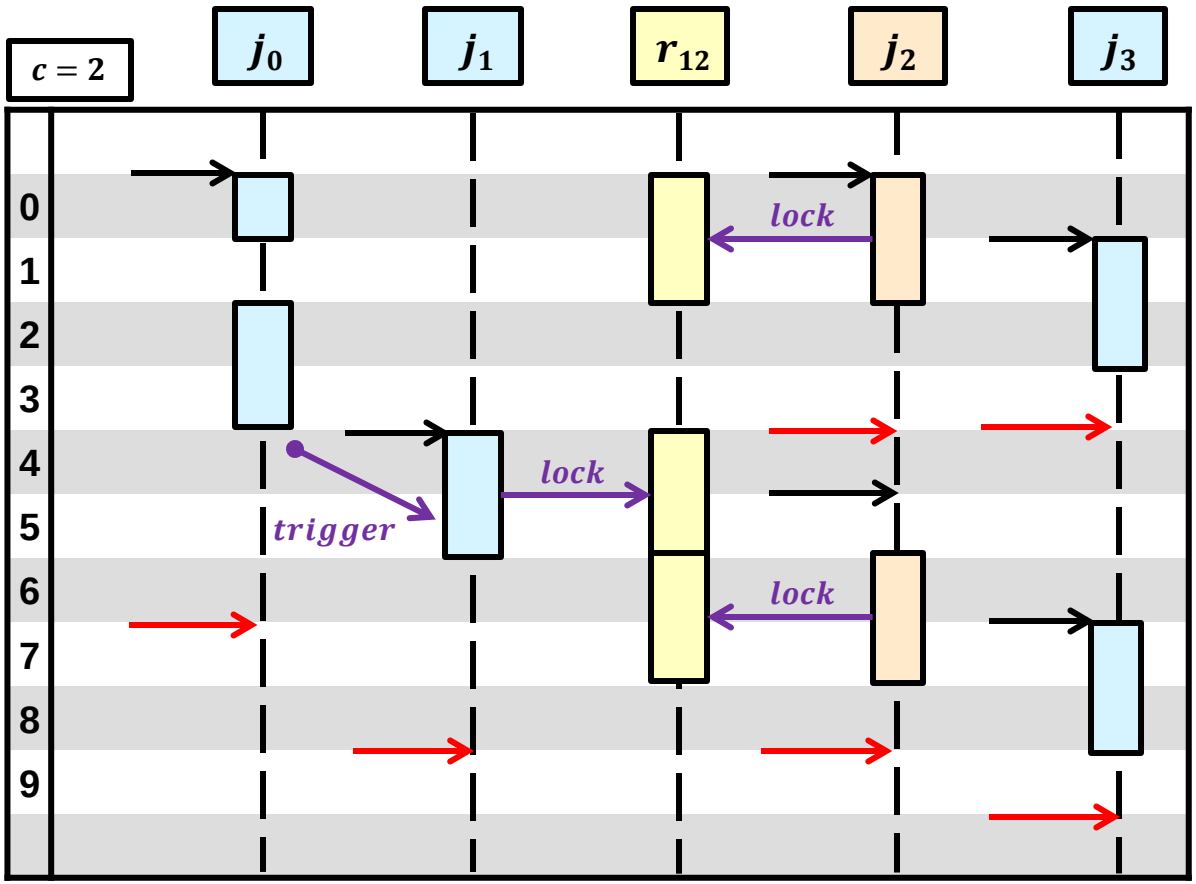
Temporal Ordering

- Triggered tasks arrive when their triggering task ends:
$$tg(j_1,j_2) \rightarrow en(j_1,k) = at(j_2,k)$$
- Dependent tasks cannot overlap:
$$dp(j_1,j_2) \rightarrow en(j_1,k_1) < st(j_2,k_2) \vee en(j_2,k_2) < st(j_1,k_1)$$

Multicore

- The system load is always less than or equal to the number of cores:
$$ld(t) \leq c$$

The RTES scheduler is modeled through constraints among Static and Dynamic properties (2/2)



Constraints

Priority-Driven Preemption

- If a task is preempted, then there are c higher priority tasks running
- Preempted time of Task Executions:
  $pm(j,k,p) = ac(j,k,p) - ac(j,k,p-1)$,
  $pm(j_0,0,1) = 1,\ pm(j_0,0,2) = 0$

Scheduling Efficiency

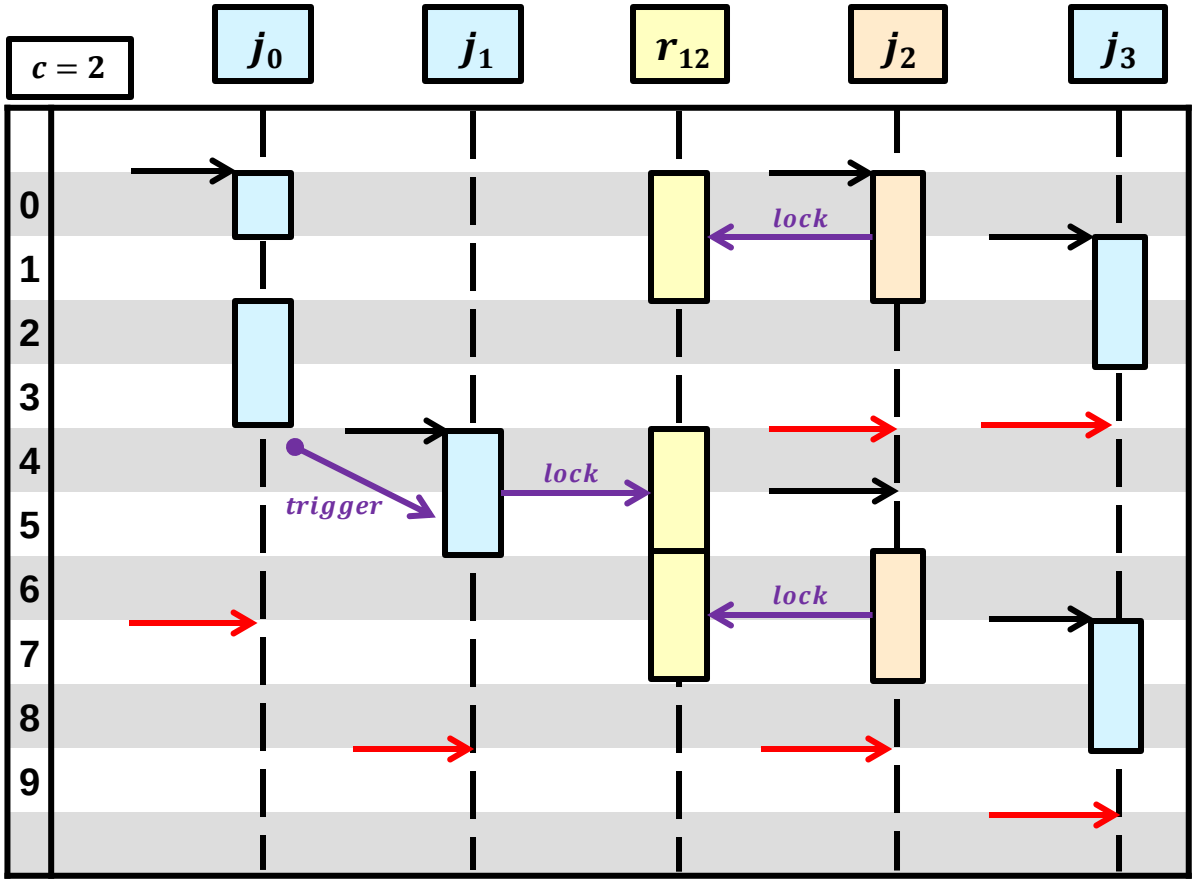
- If a task is waiting, then either
  - There are no free cores, or
  - A dependent task is active, or
  - A dependent task is preempted
- Waiting time of Task Executions:
  $wt(j,k) = st(j,k) - at(j,k)$,
  $wt(j_2,0) = 0,\ wt(j_2,1) = 1$

$$pm(j,k,p) \cdot c = \sum_{j_1 : pr(j_1) > pr(j),\ k_1,\ p_1} ac(j,k,p-1) < ac(j_1,k_1,p_1) < ac(j,k,p)$$

$$wt(j,k) = ha(j,k) + da(j,k) + dp(j,k)$$

- $ha(j,k)$ = Time quanta where c tasks with higher priority are active
- $da(j,k)$ = Time quanta where tasks depending on j are active
- $dp(j,k)$ = Time quanta where tasks depending on j are preempted

The Performance Requirements of the RTES are modeled as objective functions to maximize



Objective Function

Deadline Misses

$$F_{DM} = \sum_{j,k} 2^{dm(j,k)},$$

$$F_{DM} = 2^{-3} + 2^{-3} + 2^{-2} + 2^{-1} + 2^{-1} + 2^{-1}$$

Response Time

$$F_{RT} = \max_{j,k}\big(en(j,k)\big) - \min_{j,k}\big(at(j,k)\big),$$

$$F_{RT} = 8 - 0 = 8$$

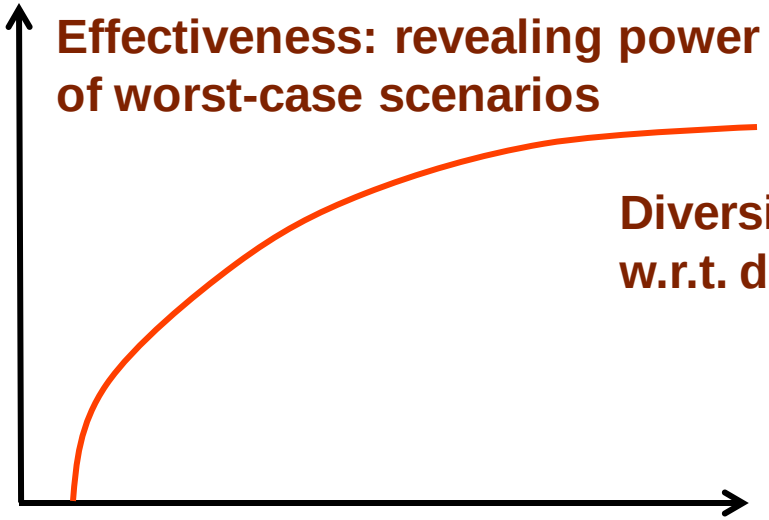
CPU Usage

$$F_{CU} = \frac{\sum_t (ld(t) > 0)}{tq}, F_{CU} = 0.9$$

F_{DM} should properly reward scenarios with deadline misses [5]

[5] L. Briand, Y. Labiche, and M. Shousha, "Using genetic algorithms for early schedulability analysis and stress testing in real-time systems", Genetic Programming and Evolvable Machines, vol. 7 no. 2, pp. 145-170, 2006

We validated our approach on the FMS, and on 5 other case studies in safety-critical domains

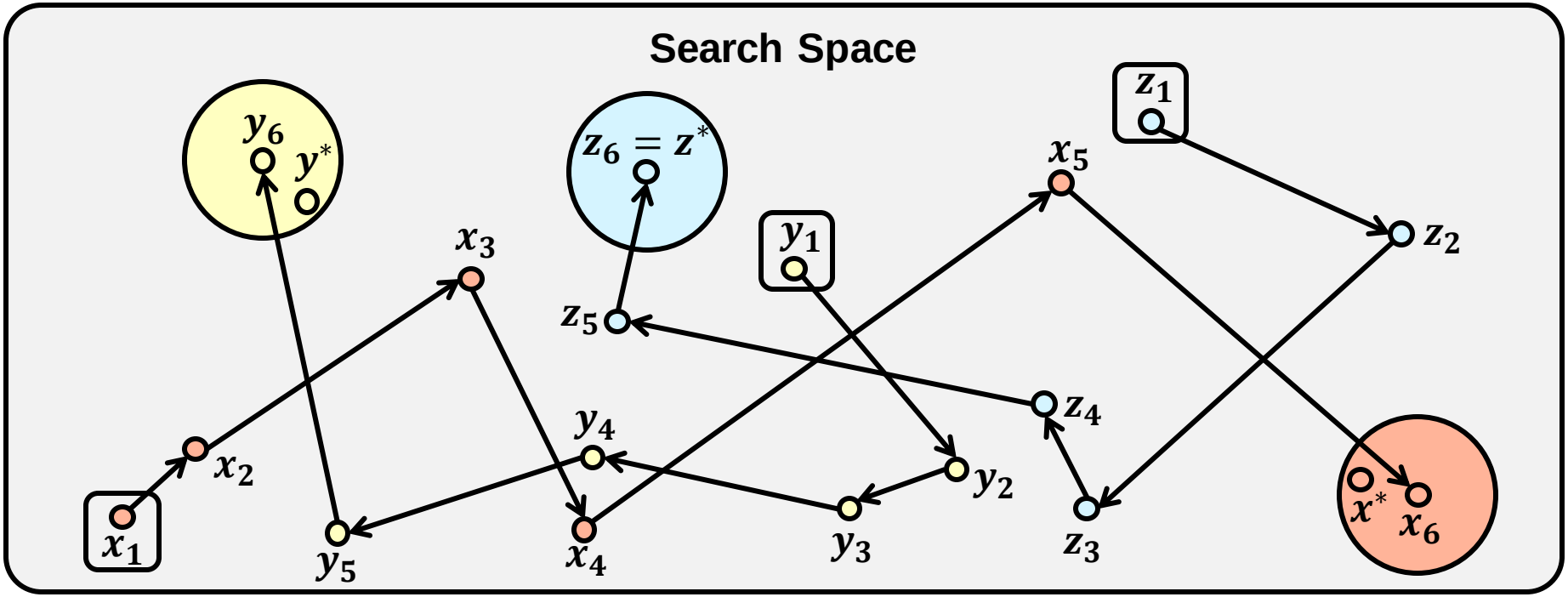


1 solution for the COP = 1 test case

	CP	GA [5]
Effectiveness	✓	
Efficiency		✓
Diversity		✓

[5] L. Briand, Y. Labiche, and M. Shousha, "Using genetic algorithms for early schedulability analysis and stress testing in real-time systems", Genetic Programming and Evolvable Machines, vol. 7 no. 2, pp. 145-170, 2006

The idea of GA+CP is to run complete searches with CP in the neighborhood of solutions found by GA [6]



GA: x_1, y_1, z_1 evolve into x_6, y_6, z_6

CP: x_6, y_6, z_6 are optimized into x^*, y^*, z^*

	CP	GA	GA+CP
Effectiveness	✓		✓
Efficiency		✓	✓
Diversity		✓	✓

[6] Di Alesio, S., Nejati, S., Briand, L., and Gotlieb, A. (2014). Combining genetic algorithms and constraint programming to support stress testing. Technical report (currently under peer-review in TOSEM).

In summary, we showed how Constrained Optimization can support Performance Testing

The COP models the System Tasks, Scheduler, and Performance Requirements

The COP finds arrival times leading to worst-case scenarios → test cases

Combine GA and CP for a good trade-off in efficiency, effectiveness and diversity

Questions?