Exhaustive Testing of Safety Critical Java

Tomas Kalibera
Pavel Parizek
Michal Malohlava

Charles University

Martin Schoeberl

Technical University of Denmark

Exhaustive Testing with Java PathFinder (JPF)

- JPF is a specialized Java Virtual Machine (JVM)
  - Runs Java programs
  - Saves program state and backtracks over different scheduling sequences
  - Looks for error states (exceptions, races, …)
- Optimizations
  - Re-scheduling only at operations that are not thread local (partial order reduction)
  - Detection of visited states (state matching)
- Designed for plain Java

(there is much more to it, see http://babelfish.arc.nasa.gov/trac/jpf/)

Our Goal: Tool for Exhaustive Testing of SCJ Programs

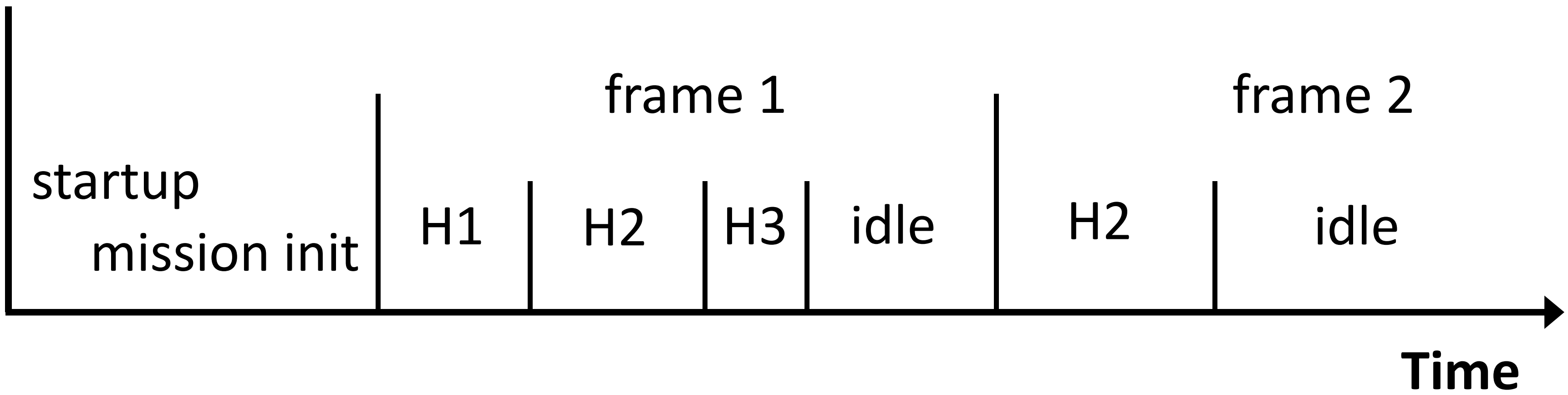
- Features sought
  - Find races (SCJ L1 and higher)
  - Find SCJ specific errors and plain Java errors even if **scheduling sequence dependent**
- Challenges
  - Cover all possible scheduling sequences with a **real-time scheduler**
  - Fight state explosion so that we can check non-toy programs

Our Contribution: Tool for Exhaustive Testing of SCJ

- Prototype implementation R_{SJ} – JPF extension
  - Detects invalid memory assignments, potential races, regular Java errors, failed assertions
  - Supports subset of SCJ L0/L1, only periodic handlers
  - Tested with Collision Detector and **PapaBench**
- SCJ L0,L1 scheduling algorithm for JPF
  - Reduction of the number of states with execution time estimator for target platform
  - Tested with Java Optimized Processor (JOP)

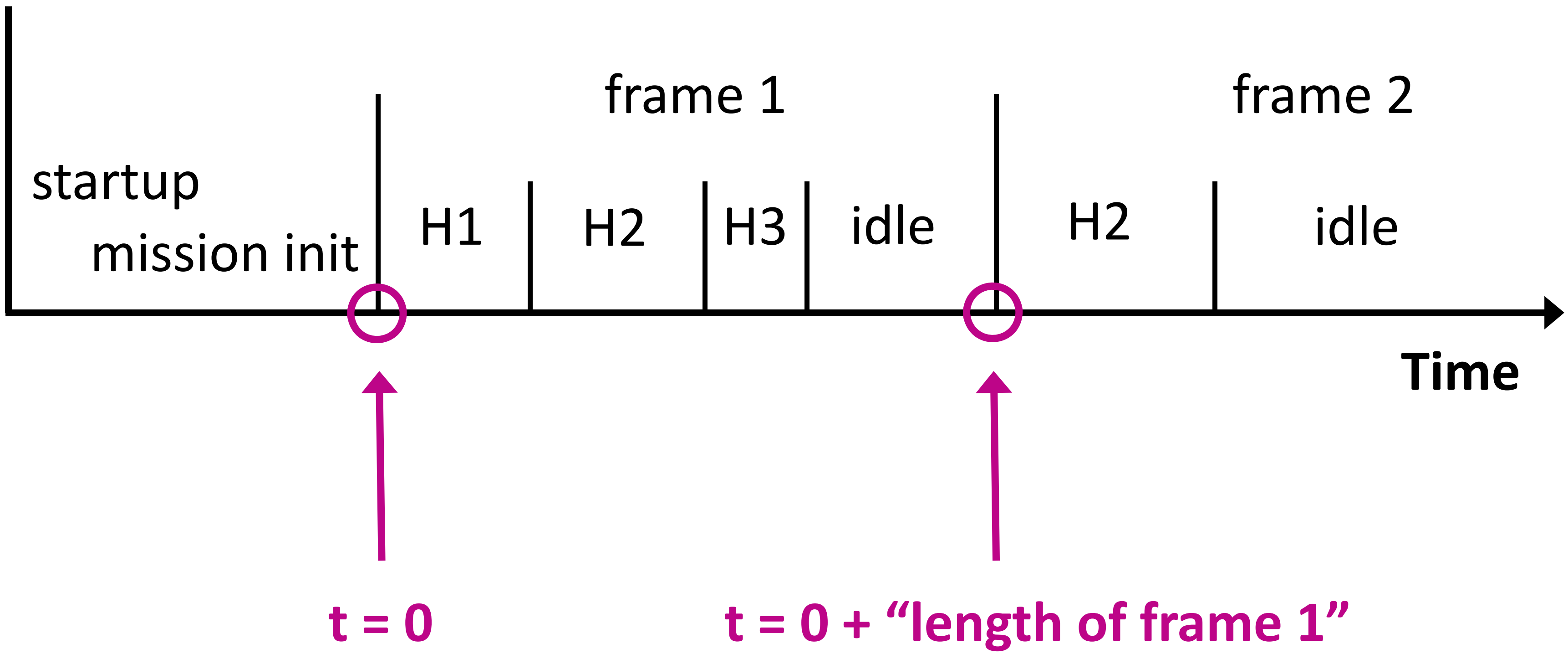
SCJ L0,L1 Scheduling for JPF

The Notion of Time at SCJ Level 0

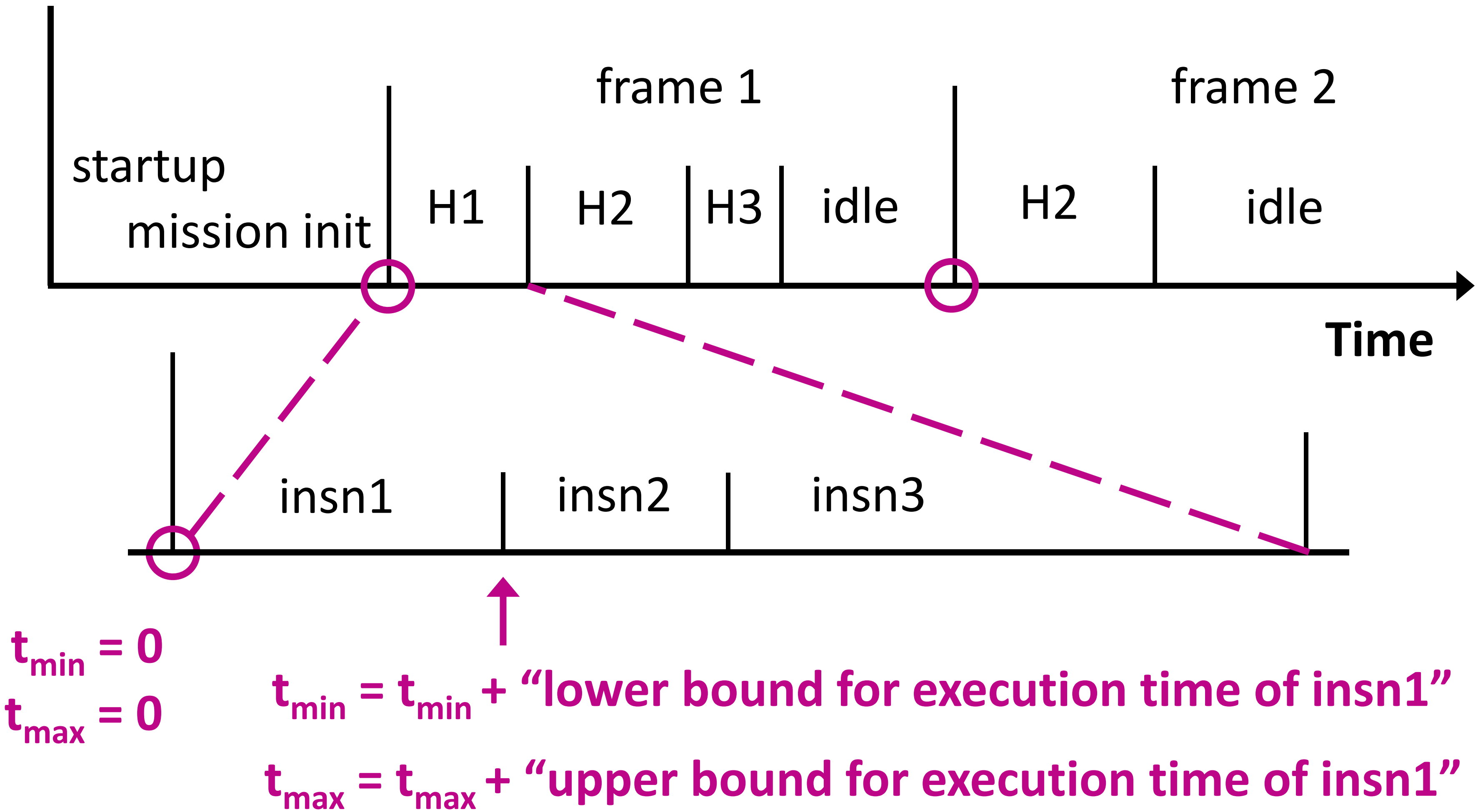


- **Only one valid scheduling sequence**
- **Notion of time is only needed for**
  - **The application – Clock.getTime**
  - **Diagnostics – detect possible frame overruns**

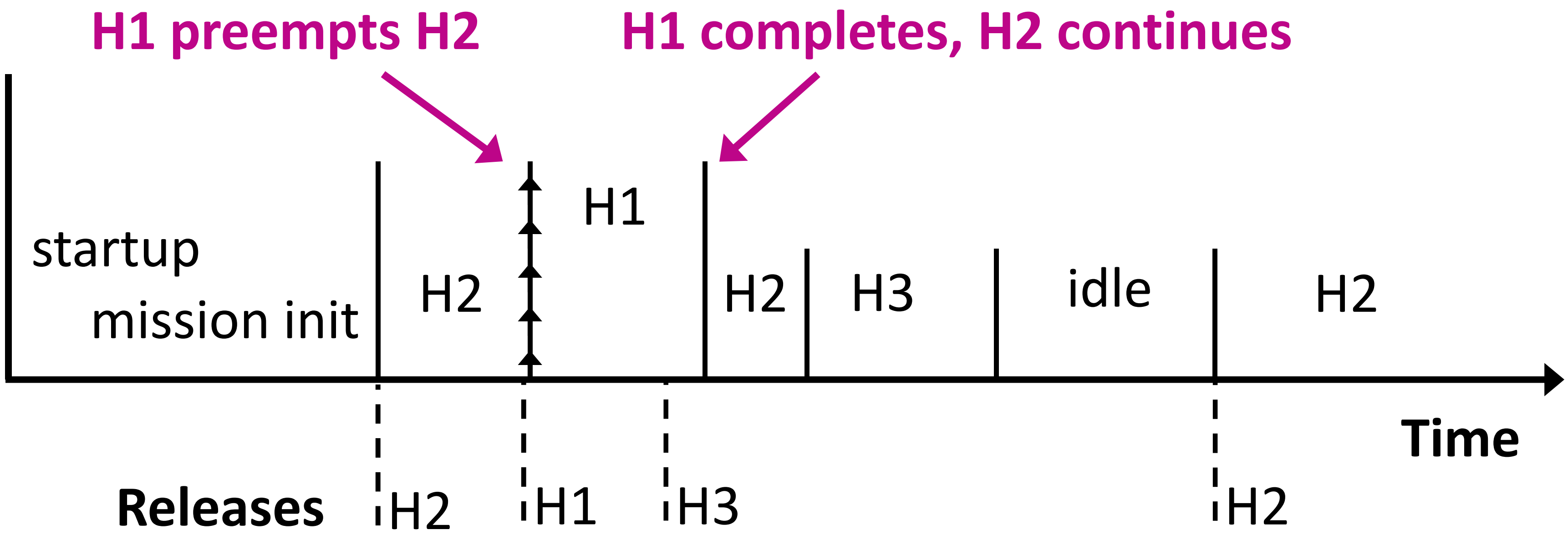
The Notion of Time at SCJ Level 0



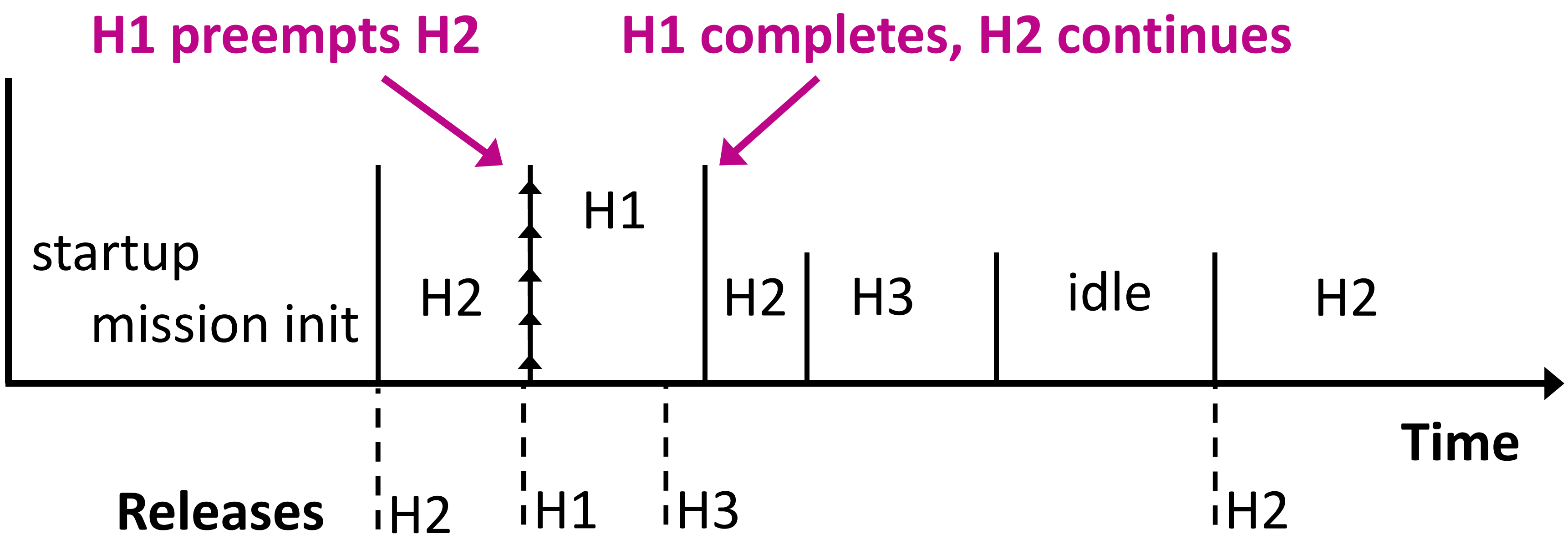
The Notion of Time at SCJ Level 0



The Notion of Time at SCJ Level 1

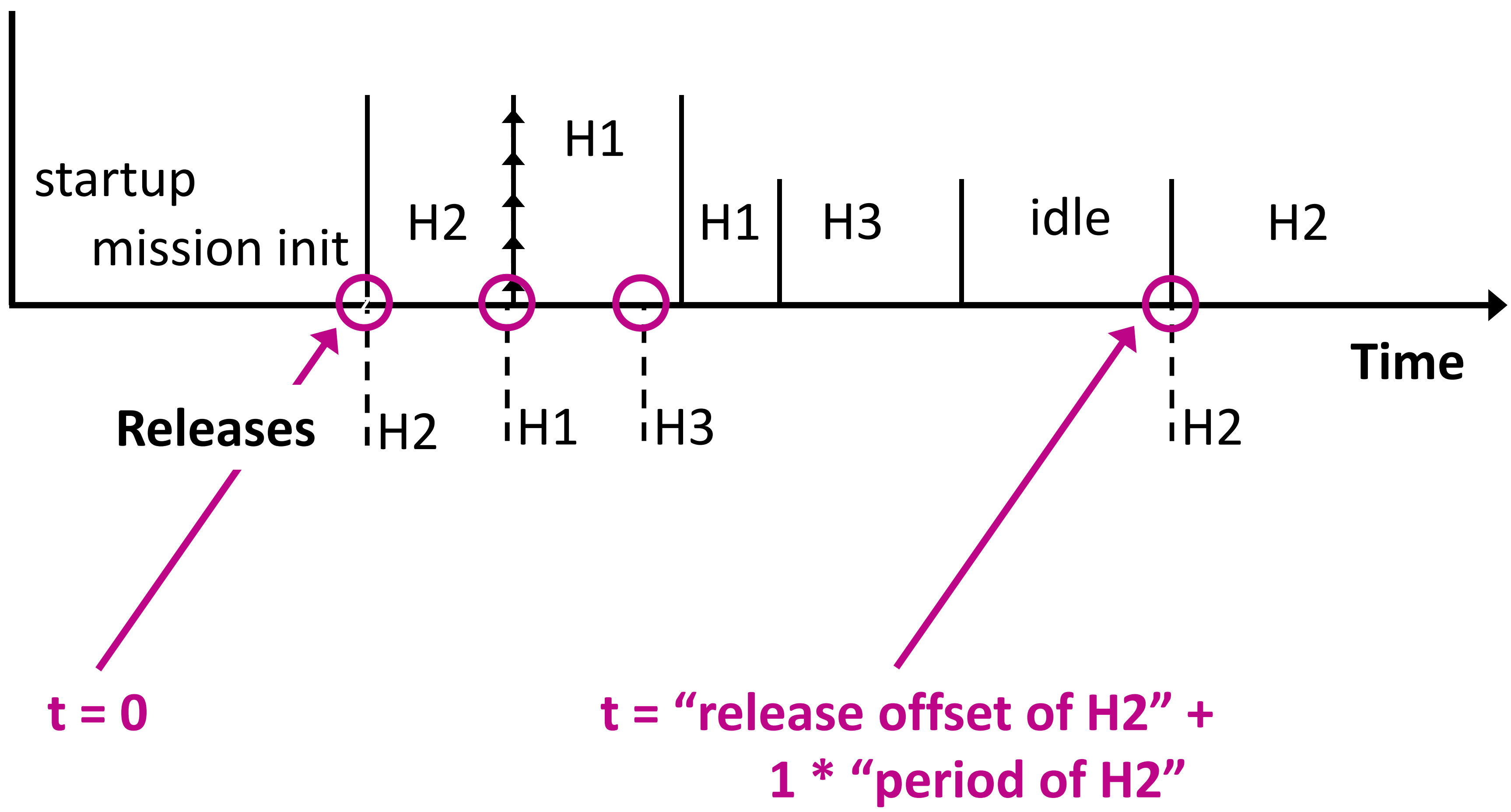


The Notion of Time at SCJ Level 1

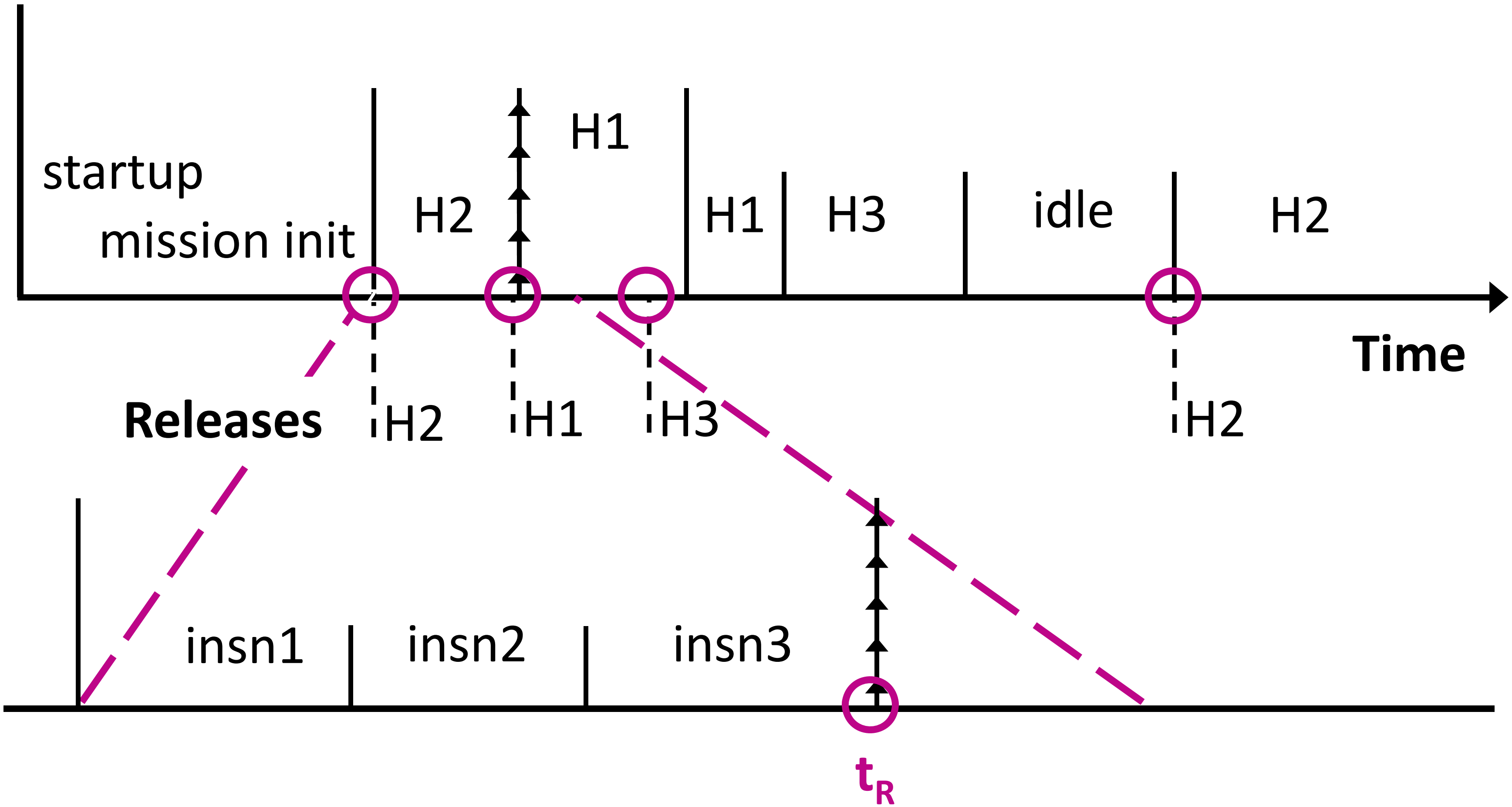


- **Notion of time needed for scheduling**
- **Imprecise notion of time results in multiple valid scheduling sequences**

The Notion of Time at SCJ Level 1



Non-deterministic Execution at SCJ Level 1

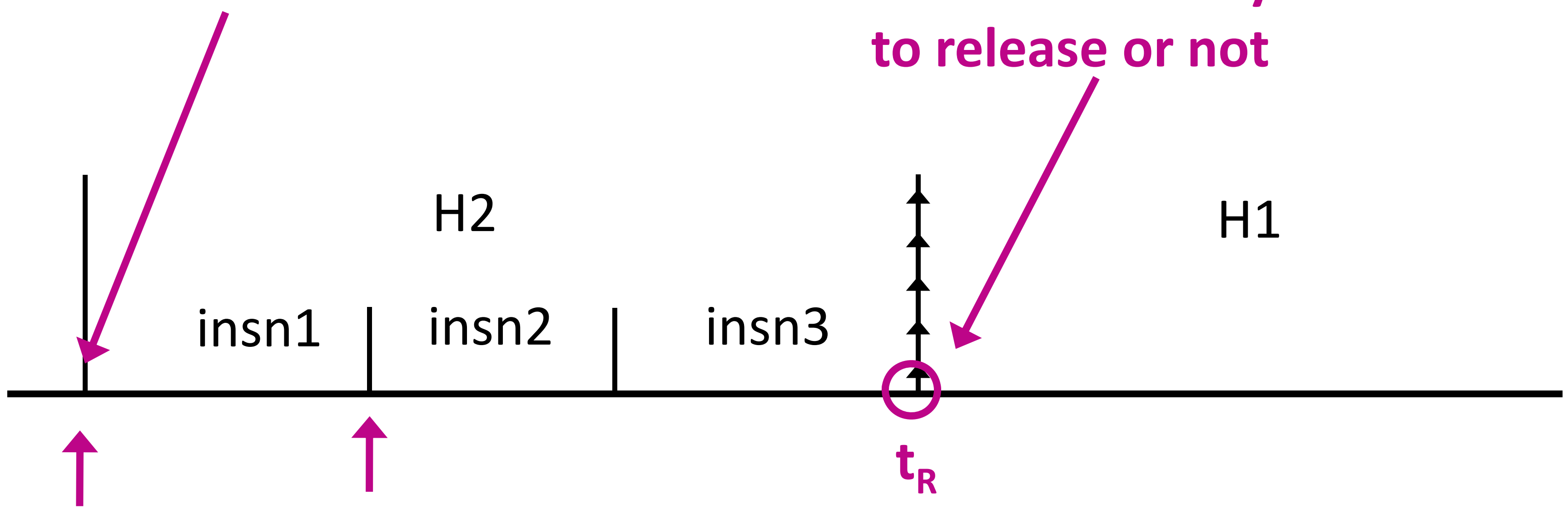


Non-deterministic Execution at SCJ Level 1

Is $t_{min} <= t_R <= t_{max}$?
(Can the release happen now ?)

If NOT, keep executing H2

If YES, choose non-deterministically whether to release or not



$t_{min} = 0$
$t_{max} = 0$

$t_{min} = t_{min}$ + “lower bound for execution time of insn1”

$t_{max} = t_{max}$ + “upper bound for execution time of insn1”

Evaluating R_{SJ}

Does it scale to real programs ?
What are the caveats of our scheduling algorithm ?

Testing with Application Benchmarks

Benchmark	# of Tasks	SCJ	Checking Time	Memory Used
CDx – no simulator	1	L0	8s	490M
		L1	12s	490M
CDx – with simulator	2	L0	34s	580M
		L1	35s	710M
PapaBench	14	L0	15min	14G
		L1	31min	15G

CDx — Collision Detector benchmark (Purdue), aircraft collision detection. We implemented the SCJ port of CDx with simulator and the L1 version

PapaBench — Based on Paparazzi UAV auto-pilot. We translated the C version of PapaBench to Java and extended it to be executable.

Java PapaBench: A Better RT Java Application Benchmark

- Paparazzi Project
  - Free auto-pilot (free sw, open-design hw)
  - ENAC University, France, http://www.enac.fr/
  - Implemented in C, has flown real UAVs
- C PapaBench
  - A subset of an earlier version of Paparazzi, intended for testing WCET analysis tools
  - IRIT, France
- **Java PapaBench**
  - Java/RTSJ/SCJ translation of PapaBench
  - Includes environment simulation to be executable
  - Michal Malohlava, Charles University
  - http://d3s.mff.cuni.cz/~malohlava/projects/jpapabench/

(Java) PapaBench Components

- Autopilot
  - Produces low-level flight commands to FBW
  - Follows a pre-configured high-level flight plane
  - Reacts to input from GPS and IR
- Fly-by-wire (FBW)
  - Low-level access to aircraft hardware
- Simulator
  - GPS, IR interrupt source
  - Physical environment simulation

Checking RT Programs: Lessons Learned

Checking RT Programs: Lessons Learned

- State matching needs revisiting
  - Current time is part of program state – SM has to be disabled, otherwise we fail to fully check a program
- Partial order reduction does not apply
  - Scheduler decisions in a real system are deterministic
  - Potential preemption points have to be fine grained (i.e. a single instruction in RSJ) to bound release jitter
- More work is needed to customize JPF-core
  - By default, states are saved even at deterministic thread switch

See the official RTEmbed extension of JPF at
http://babelfish.arc.nasa.gov/trac/jpf/wiki/projects/*rtembed* for our related efforts.