Implementing AUTOSAR Scheduling and Resource Management on an Embedded SMT Processor

Florian Kluge, Chenglong Yu, Jörg Mische, Sascha Uhrig, Theo Ungerer

Chair of Systems and Networking
University of Augsburg

12th International Workshop on
Software and Compilers for Embedded Systems
April 23-24, 2009
Acropolis, Nice, France

Outline

Outline

Motivation (I)

Automotive Software Standards

- AUTOSAR: AUTOmotive Open System ARchitecture
- Provides common platform for automotive applications
- Increases interoperability and interchangeability

Motivation (I)

Automotive Software Standards

- AUTOSAR: AUTOmotive Open System ARchitecture
- Provides common platform for automotive applications
- Increases interoperability and interchangeability

Upcoming Processor Architectures

- Simultaneous MultiThreading (SMT) Cores
- Multi-Core Processors

→ real parallelism!

Motivation (II)

- Is AUTOSAR fit for SMT?

Motivation (II)

- Is AUTOSAR fit for SMT? NO!

Motivation (II)

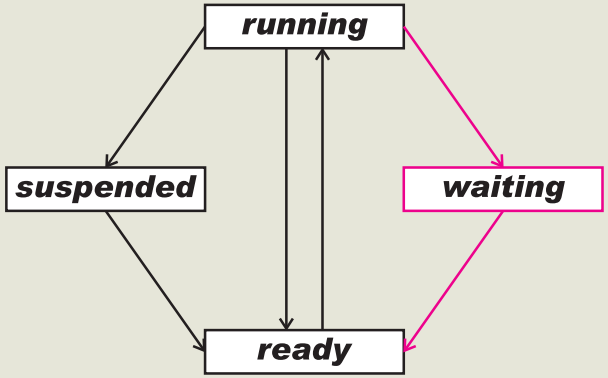
- Is AUTOSAR fit for SMT? NO!
- What can we do?

Motivation (II)

- Is AUTOSAR fit for SMT? NO!
- What can we do?
- How can AUTOSAR benefit from SMT?

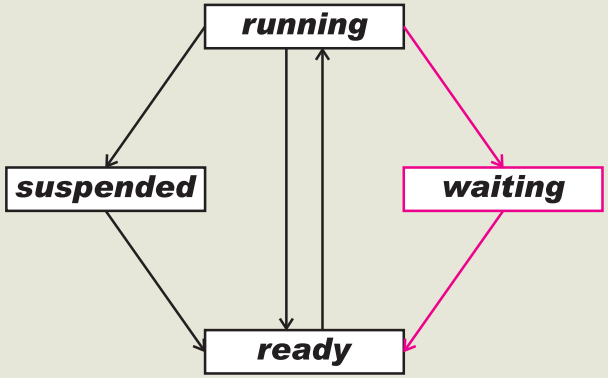
AUTOSAR Scheduling

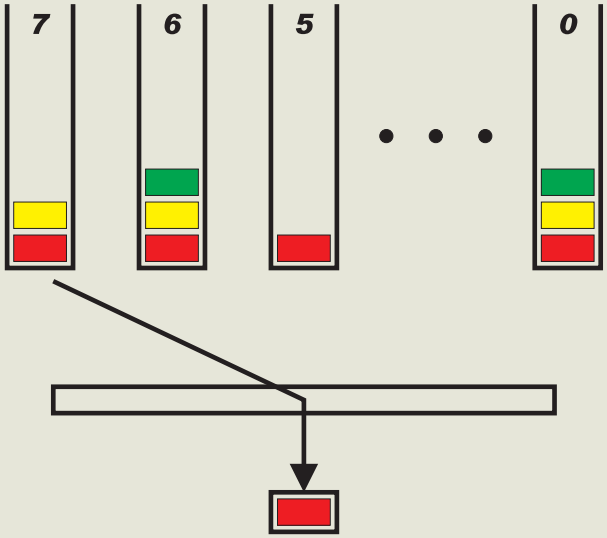
- *Basic* and *Extended* tasks
- Task states: *suspended*, *ready*, *running*, *waiting* (only extended tasks)



AUTOSAR Scheduling

- *Basic* and *Extended* tasks
- Task states: *suspended*, *ready*, *running*, *waiting* (only extended tasks)
- Priority based scheduling: highest priority task is executed
- Designed for singlethreaded processors

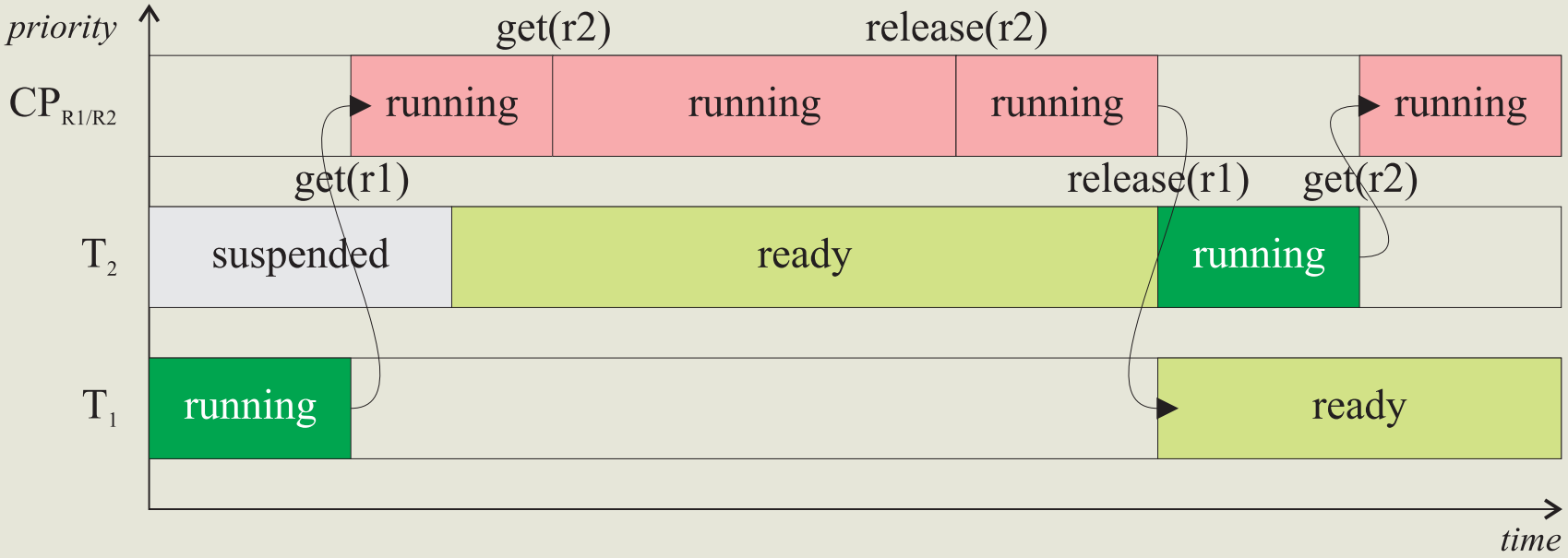




AUTOSAR Resource Management

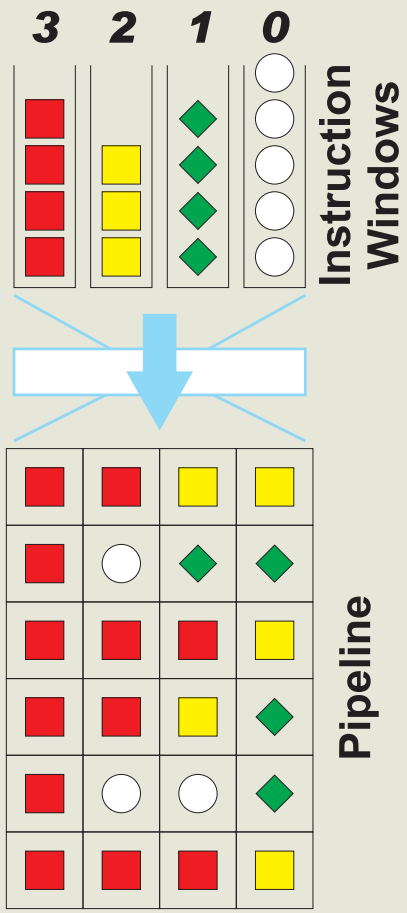
- Mutual exclusion on resource access
- Prevent priority inversion and deadlocks
- Resource access never results in *waiting* state

→ Priority Ceiling Protocol (PCP)



SMT Processor

- Simultaneous MultiThreading
  - issue instructions of different threads in one cycle
- Use latencies of one thread to execute another
- Prioritised slots
- Real parallelism



Challenges

Goal: Earlier execution of low-priority tasks

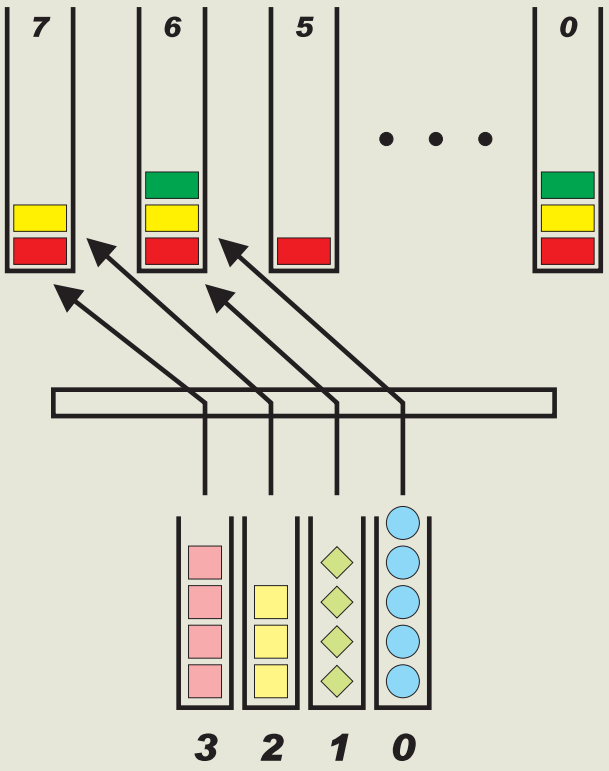
Challenges

- Preserve WCET analysability
- Preserve externally observable behaviour of highest-priority task
  - Lower-priority tasks can be interrupted by higher-priority tasks!
- As few changes as possible to standard definition

Outline

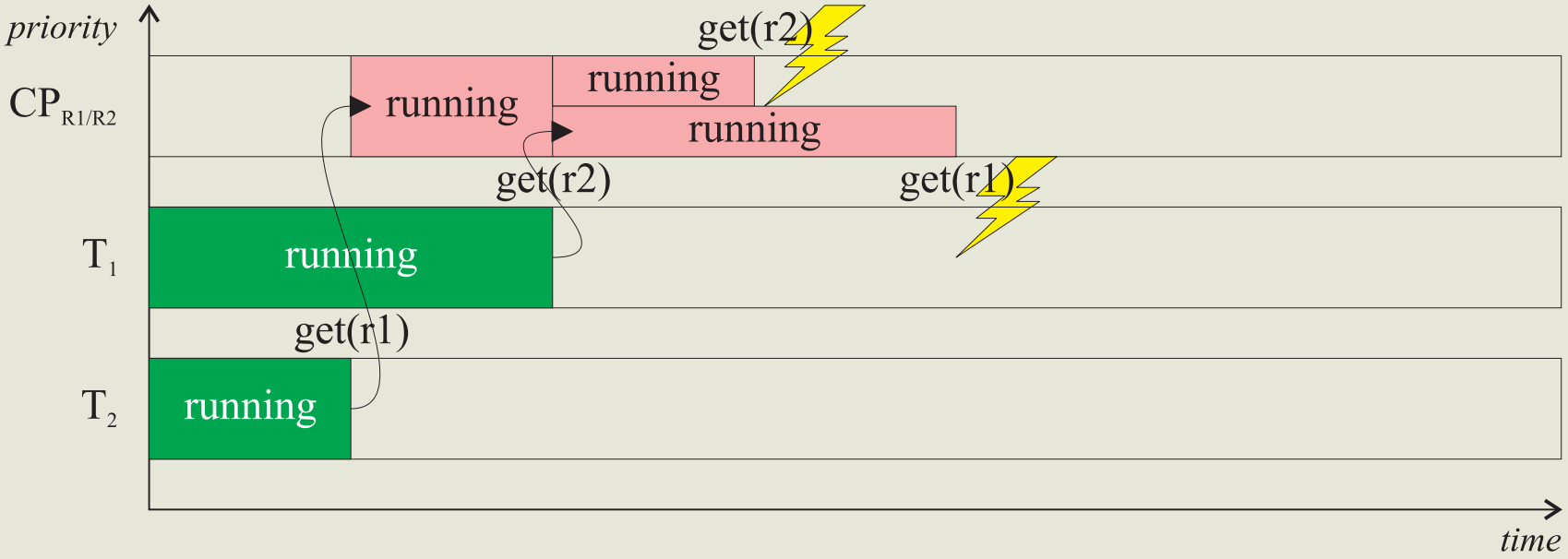
Forward Implementation

- SMT processor with n thread slots
- Issue n highest priority tasks
  - $\longrightarrow$ Keep execution order
- Map AUTOSAR task priorities to thread priorities



Resource Management and Parallel Execution

- PCP relies on singlethreaded program execution
- No prevention from priority inversion and deadlocks in multithreaded program execution



Related Work: Resource Management

Resource Management for Multiprocessor Systems

- Multiprocessor Priority Ceiling Protocol
- Multiprocessor Stack Resource Policy
- LP-Time Inheritance and Returning

Related Work: Resource Management

Resource Management for Multiprocessor Systems

- Multiprocessor Priority Ceiling Protocol
- Multiprocessor Stack Resource Policy
- LP-Time Inheritance and Returning

- ⛔ No transfer of processing time possible on SMT
- ⛔ No nested resource accesses possible

Data Synchronisation

Scheduler

- Shared data structures (FIFO queues)
- NO lock variables available!
- Further synchronisation necessary

Data Synchronisation

Scheduler

- Shared data structures (FIFO queues)
- NO lock variables available!
- Further synchronisation necessary

Related Work

- Lamport (*Bakery Algorithm*), Peterson's Algorithm:
  - Spinning or sleeping/suspending
  - ⛔ Spinning wastes processing time
  - ⛔ Suspending can violate task state model

Data Synchronisation

Scheduler

- Shared data structures (FIFO queues)
- NO lock variables available!
- Further synchronisation necessary

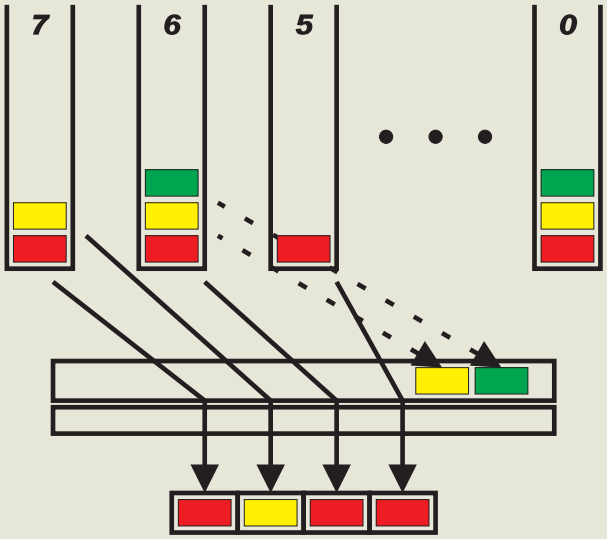
Related Work

- Lamport (*Bakery Algorithm*), Peterson's Algorithm:
  - Spinning or sleeping/suspending
  - ⛔ Spinning wastes processing time
  - ⛔ Suspending can violate task state model
- Lock-free algorithms
  - Special instructions necessary
  - Compare&Swap or Load Link/Store Conditional

Outline

Task Filtering

- Execute only tasks that do not use the same resources
- Deferred tasks are buffered in *task filter*
- Internal task state *resource ready*
  - Task has highest priority of all tasks accessing the same resources

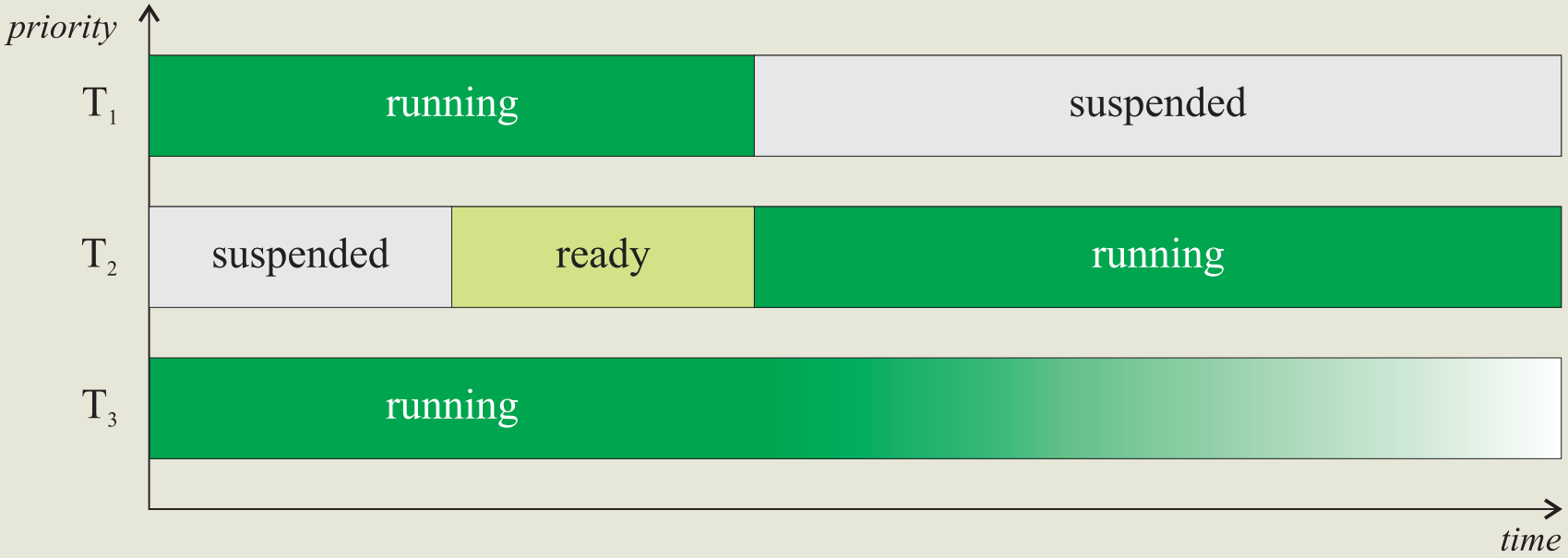


Task Filtering: Example

	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8
R_1	X							
R_2			X			X		
R_3	X							
R_4				X				
R_5	X		X					
R_6					X		X	
R_7			X				X	X
R_8					X			
SC	**o**	**o**		**o**	**o**	**o**		**o**

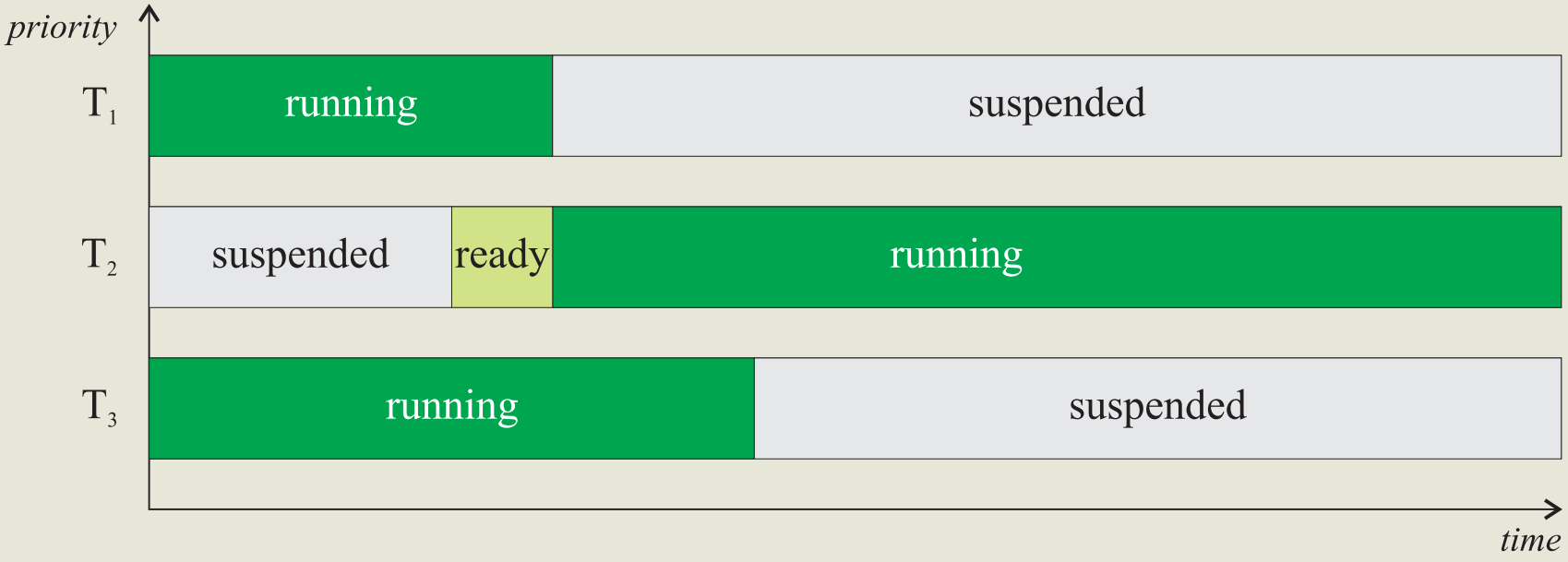
Discussion

- Tasks T_1, T_2, T_3
- Priorities $P_1 > P_2 > P_3$
- 2 thread slots
- T_1, T_3 running
- T_1 and T_2 have resource conflict



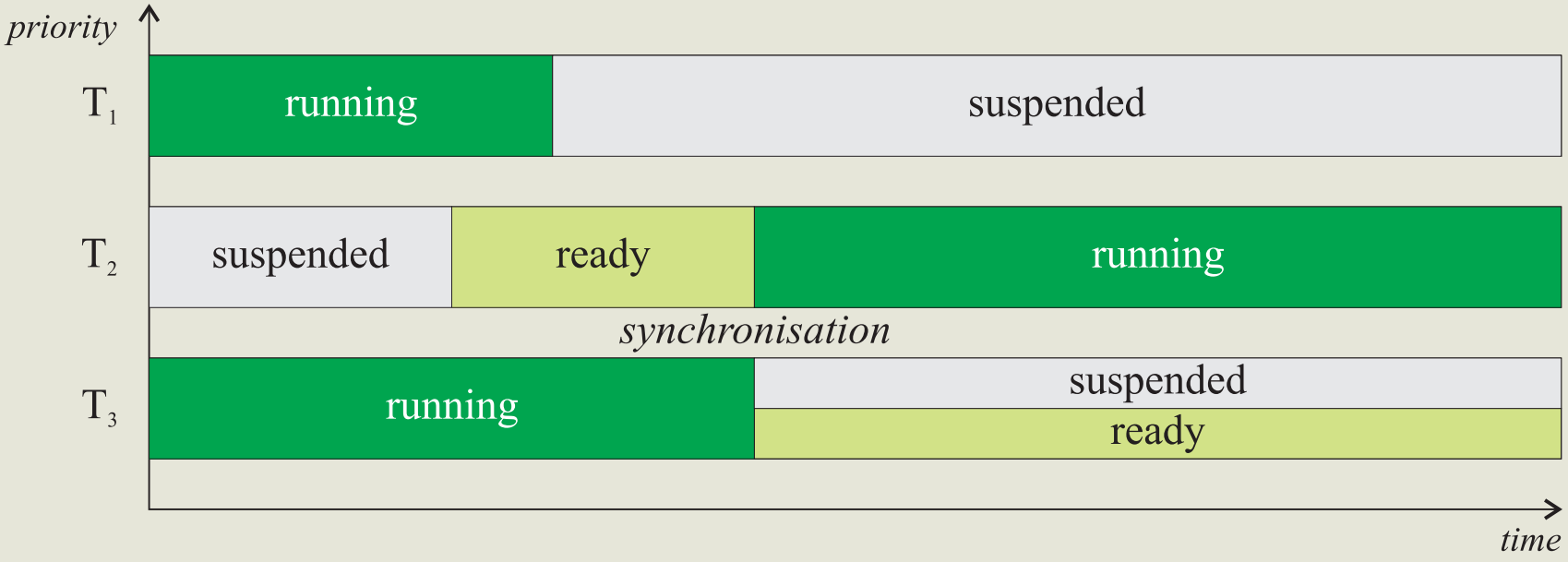
Discussion (I)

- No resource conflict between T_2 and T_3



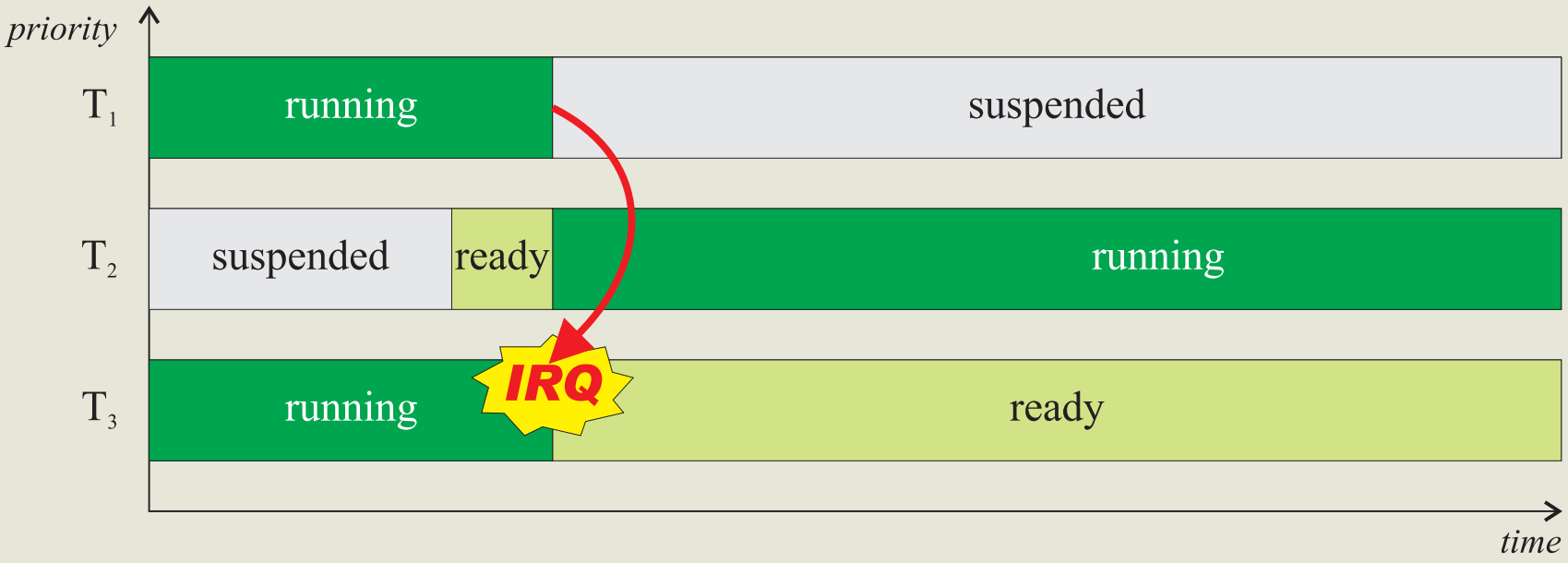
Discussion (II)

- Resource conflict between T_2 and T_3 on R
- T_3 non-preemptable



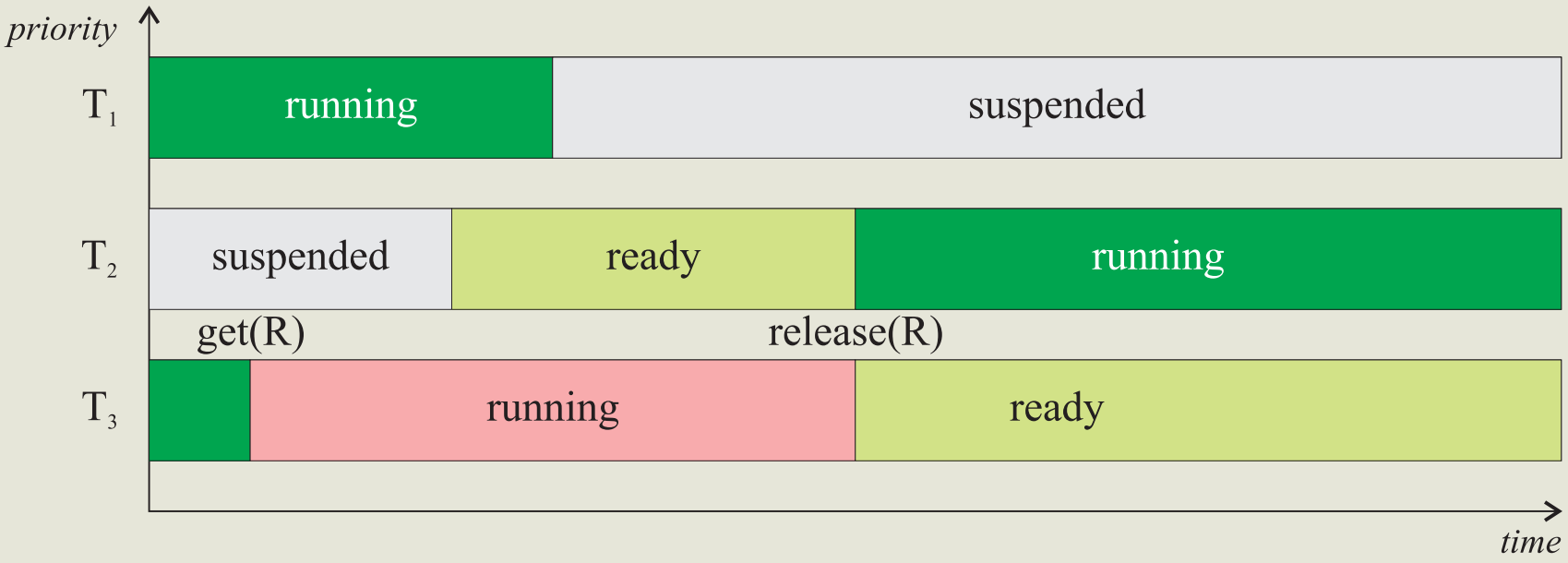
Discussion (III)

- Resource conflict between T_2 and T_3 on R
- T_3 preemptable
- T_3 not holding R or other with $CP > P_2$



Discussion (IV)

- Resource conflict between T_2 and T_3 on R
- T_3 preemptable
- T_3 holding R



Consequences

- Start of high priority tasks may be delayed, but time-predictable
- Once running, behaviour is same as in single-threaded processor for highest priority task
- Lower prioritiy tasks: may start earlier, but with lower performance
- Time for scheduling increases about 100 cycles - total time about 1.600 cycles (on prototype)
- Hardware requirement: slot-to-slot interrupt

Synchronisation of Scheduler Data

- Scheduler's FIFO-Queues accessed from every slot
- Must ensure consistency
- Use lock-free linked lists
- HW Requirement: COMPARE&SWAP or LOAD LINK/STORE CONDITIONAL instructions
- Work in progress

Outline

Summary & Conclusion

Extensions to AUTOSAR

- Task Filtering method prevents resource conflicts
- New state *resource-ready* introduced
- Necessary hardware extensions
  - CAS
  - Slot-to-slot interrupts

⇒ Timing behaviour of highest-priority task is preserved

Summary & Conclusion

Extensions to AUTOSAR

- Task Filtering method prevents resource conflicts
- New state *resource-ready* introduced
- Necessary hardware extensions
  - CAS
  - Slot-to-slot interrupts

⇒ Timing behaviour of highest-priority task is preserved

Conclusions

- Current AUTOSAR not ready for multiprocessing
- More changes on specification necessary
- This work is just one step on the road...

Future Work

- Lock-free Algorithms
  - Implementation and evaluation
- Benchmarking
  - Current benchmarks just look at performance
  - Design benchmark to test resource accesses
- Integration with Multi-Core Research
  - MERASA project (Multi-Core Execution of Hard Real-Time Applications Supporting Analysability)

Thank you!

Any Questions?