Non-preemptive scheduling of real-time STM

António Barros and Luís Miguel Pinho

CISTER research centre, Porto, Portugal

DMTM — Vienna, 2014-01-22

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

Outline

- Motivation
- Background
- Serialising transactions
- Non-preemptive scheduling of transactions
- Conclusion

MOTIVATION

Motivation

- Multicore architectures for embedded systems:
  - less computing power on each core, but
  - more computing power from parallel concurrent programming.
- Synchronisation mechanisms that are able to cope with parallel programing.

BACKGROUND

Real-time embedded systems

- Computer system.
  - Application functionality is divided into tasks.
  - Tasks are instantiated recurrently throughout time: jobs.
  - A job must provide a correct result within a time window.
  - It does not have to be fast: it has to be ***predictable***!

Synchronisation mechanisms for real-time systems

- Traditionally lock-based, but…
  - Coarse-grained locks impair parallelism.
  - Fine-grained locks impair composability.

SERIALISING TRANSACTIONS

How to serialise transactions in real-time systems?

- Polite, aggressive, karma, exponential back-off
  - Not predictable!
- Priority, deadline
  - Possibly starves transactions with lower priority.
- Slack
  - Possibly starves transactions with more slack.

How to serialise transactions in real-time systems?

- Time of arrival (FIFO)
  - Priority inversion, but…
    - All transactions have a fair chance to commit.
    - Blocking must by limited!
  - Cascading aborts:
    - T1 (earliest) writes objects A and B
    - T2 (in between) writes objects B and C
    - T3 (latest) writes objects C and D
    - T3 may abort because of T2, that is aborting due to T1!

FIFO approach

If already a zombie, repeat.

Get ownership of read set.

For each object in write set…

- Get ownership of object.
- If there is a earlier transaction active conflicting release all objects and repeat.

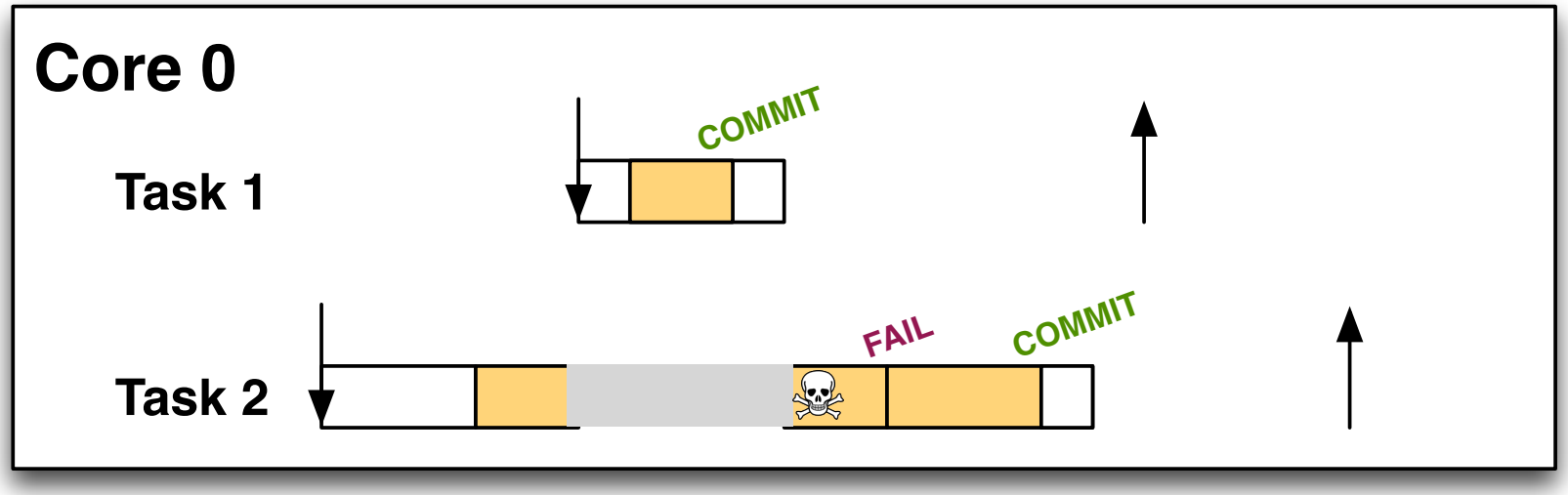
If still active…

- Release read set.
- Set contenders as zombies and release write set.

NON-PREEMPTIVE SCHEDULING OF TRANSACTIONS

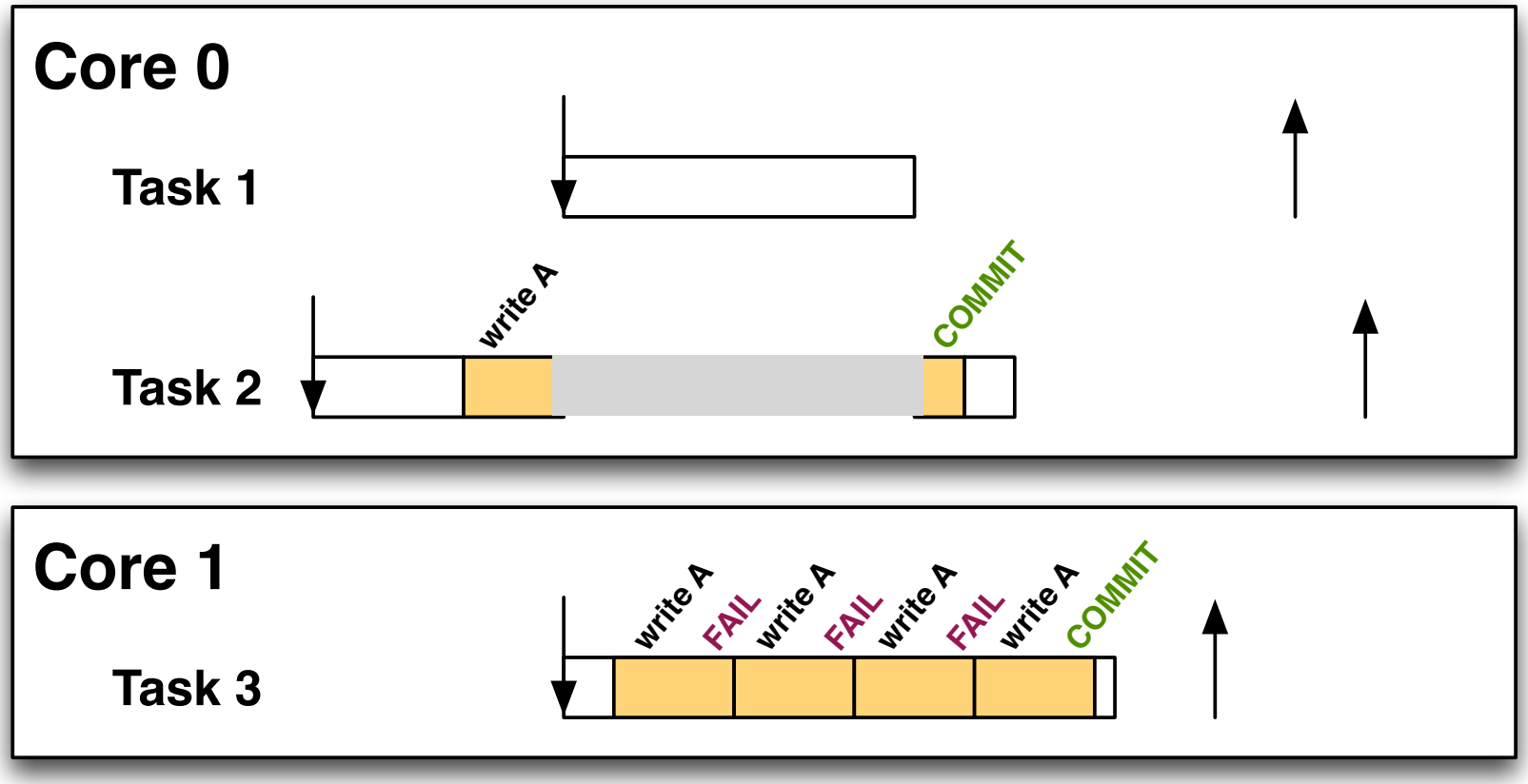
Preemptions can modify the contention management behaviour

- Contention in the same core.



Preemptions can modify the contention management behaviour

- Contention in different cores.



Non-preemptive approaches

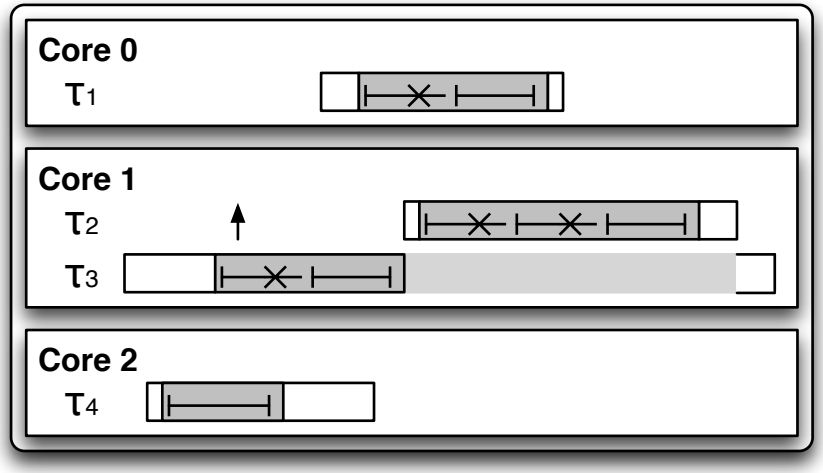
- Non-preemptive until commit (NPUC)
  - Job is not preemptible until transaction successfully commits.
  - At most *one* transaction started on each core.
    - Response time of transaction is totally predictable.
  - Blocking can be long.

Non-preemptive approaches

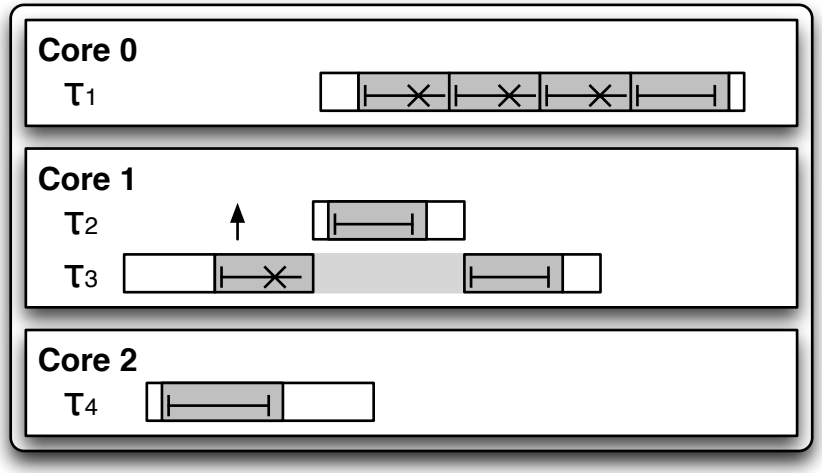
- Non-preemptive during attempt (NPDA)
  - Job is not preemptible during transaction, but has preemption points between attempts.
  - Allows *more than one* transaction started on each core.
    - Response time analysis of transaction is too pessimistic.
  - Blocking is limited to the duration of the longest transaction attempt.

Non-preemptive approaches

- NPUC



- NPDA



CONCLUSIONS

Conclusions

- FIFO serialisation may provide a predictable serialisation method.
- Preemptions can undermine the expected behaviour of a contention manager.
- Two non-preemptive approaches:
  - NPUC, more predictable, less responsiveness.
  - NPDA, more responsiveness, but very hard to analyse.

THANK YOU! QUESTIONS?

This work was partially supported by National Funds through FCT (Portuguese Foundation for Science and Technology) and by ERDF (European Regional Development Fund) through COMPETE (Operational Programme 'Thematic Factors of Competitiveness'), within project FCOMP-01-0124-FEDER-015006 (VIPCORE) and FCOMP-01-0124-FEDER-037281 (CISTER).