Liveness in Transactional Memory

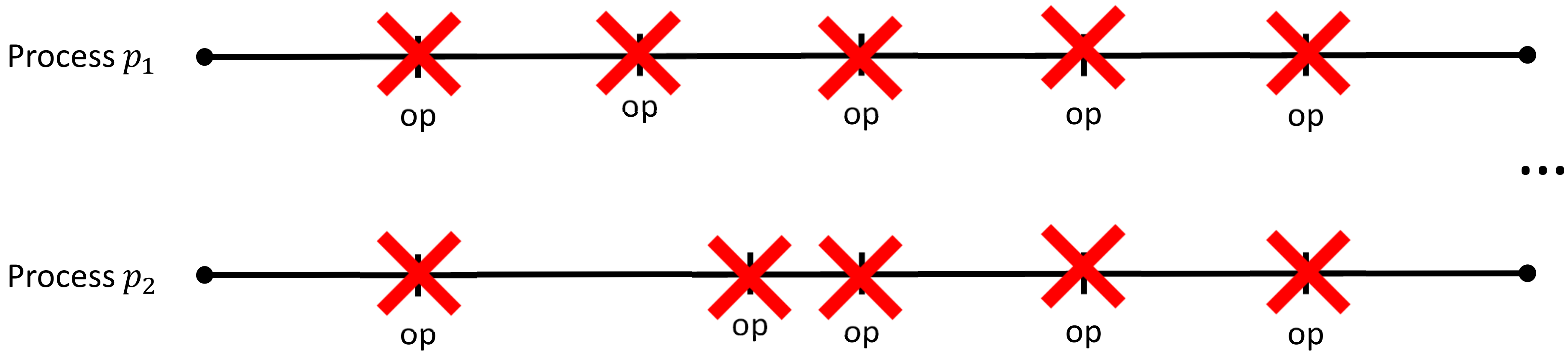
Jan Haltermann

Seminar: Software Transactional Memory

29.11.2017

PADERBORN UNIVERSITY
The University for the Information Society

Motivation - Why do we need Liveness?



Correctness/Safety ✓

Desired behavior ✗

Content

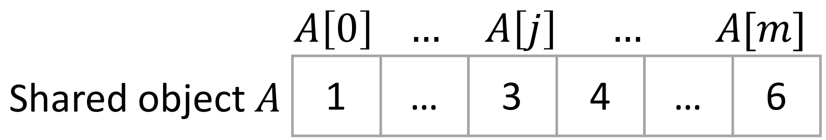
PADERBORN UNIVERSITY
The University for the Information Society

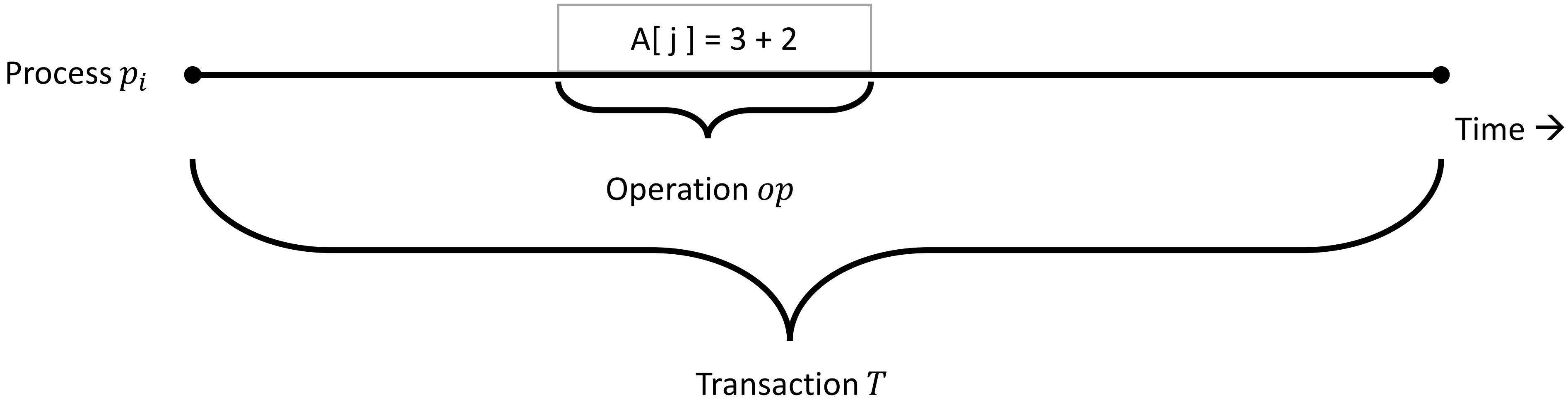
- Preliminaries
- How can liveness be defined?
- Liveness and correctness criteria in a STM
- Summary

Preliminaries

Preliminaries – System Model

- N asynchronous processes $p_1, \ldots, p_n$
- Shared objects/memory for communication
  - using base objects accessible via atomic operations
- Process p_i accesses shared object A
- Processes are sequential (no parallel operations)

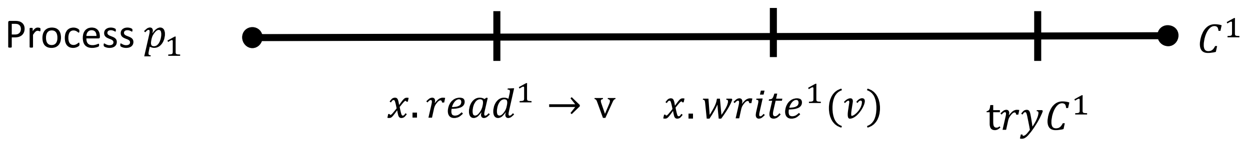




Preliminaries – Transactional Memory

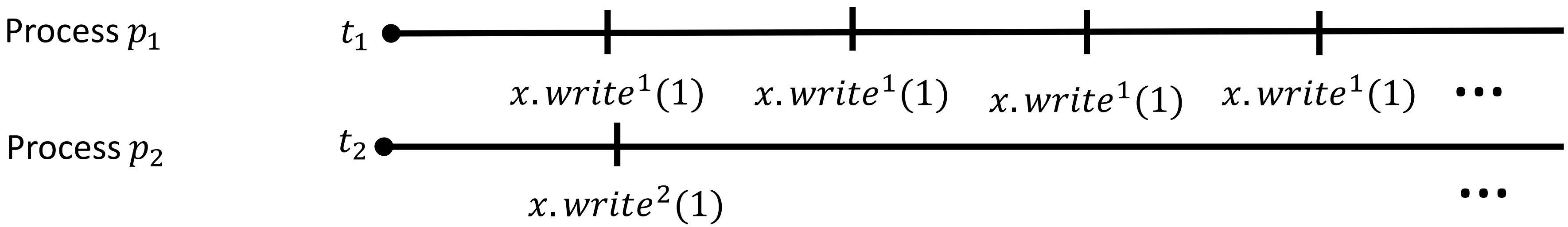
Operation of process p_k on shared variable x

- $x.read^k$: Read current value of x
  - Returns value v or $abort$
- $x.write^k(v)$: Write value v on x
  - Returns ok or $abort$
- $tryC^k$:Try to commit
  - Returns $commit$ or $abort$



Preliminaries – Transactional Memory

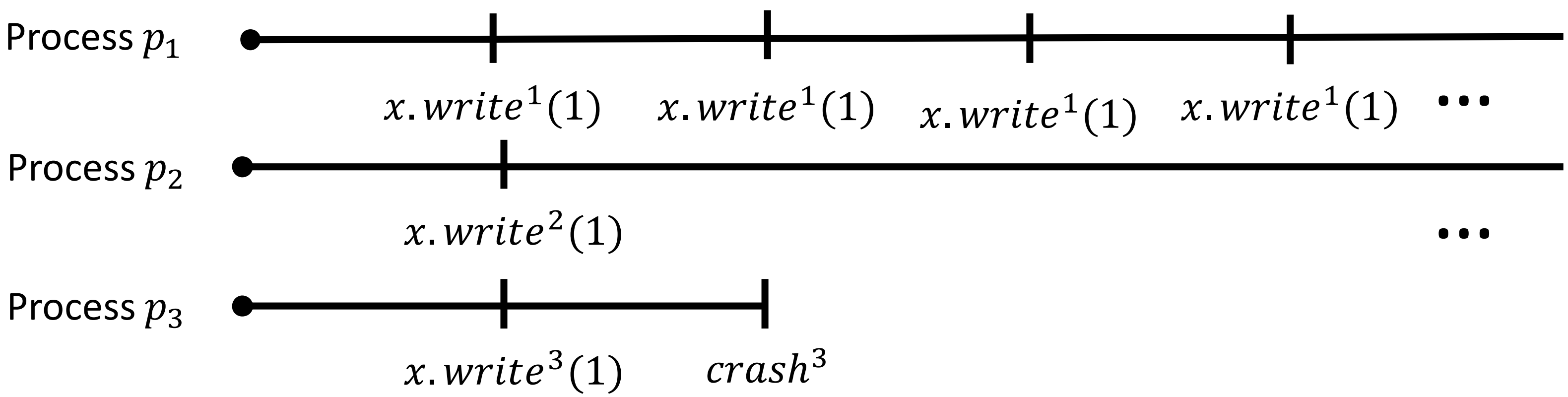
- Transaction: Sequence of reads and writes followed by a commit or abort
- $T_1 <_H \ T_2$: T_1 commits or aborts before first event of T_2 (T_1 preceds T_2)
- Crashed transaction: stops taking operations in infinite history
  - Waiting infinitely long for a response unequal to crashing
- Parasitic transaction: after point t, no $tryC$ request
- Correct transaction: Neither crashed nor parasitic



Preliminaries – Transactional Memory

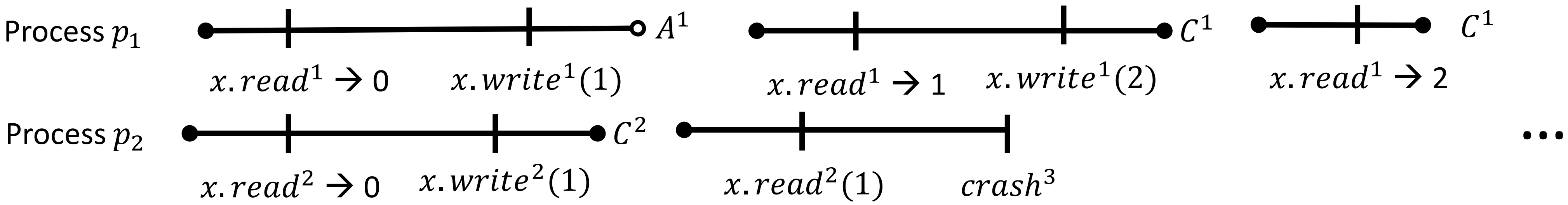
PADERBORN UNIVERSITY
The University for the Information Society

- History: longest subsequence of an execution on a shared object
- Complete history: Every transaction ends with commit or abort
- Sequential history: No two transactions are concurrent
- Fair history: add a $crash^k$ event to all crashing processes



Preliminaries – Transactional Memory

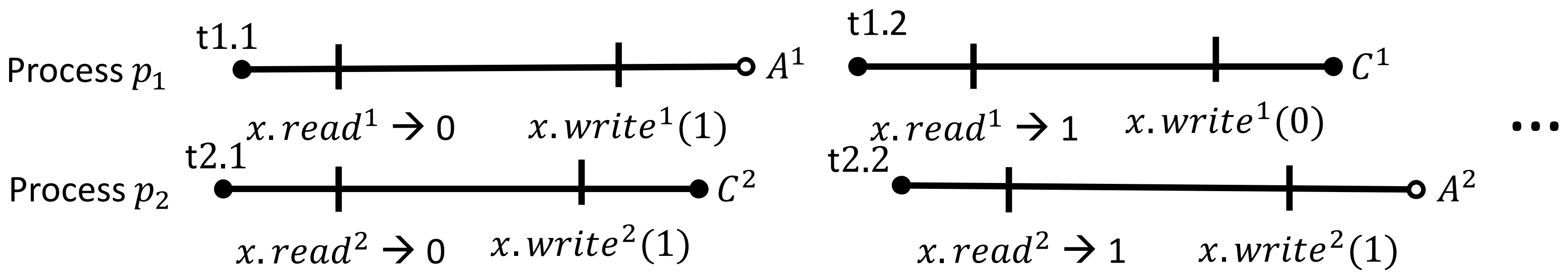
- Legal transaction T: all reads of T and all transactions that precede T return valid values (that are currently stored in shared object
- Process p_k makes progress in fair history H, if H contains infinitely many C^k
  - Implies eventually all operations in transaction are not aborted
- Process p_k runs alone: from point t on, no other process takes steps in execution



Liveness for STMs

Local Progress

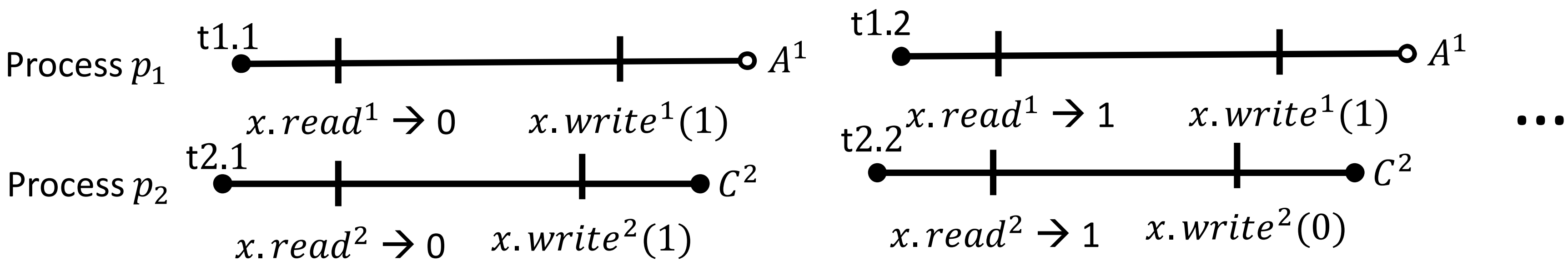
- **Every correct process makes progress** in a fair history
  - Or there are no correct processes



- Stronger property than wait-freedom
  - Wait-freedom: Every operation receives a response
  - Introduced by Herlihy in 1991

Global Progress

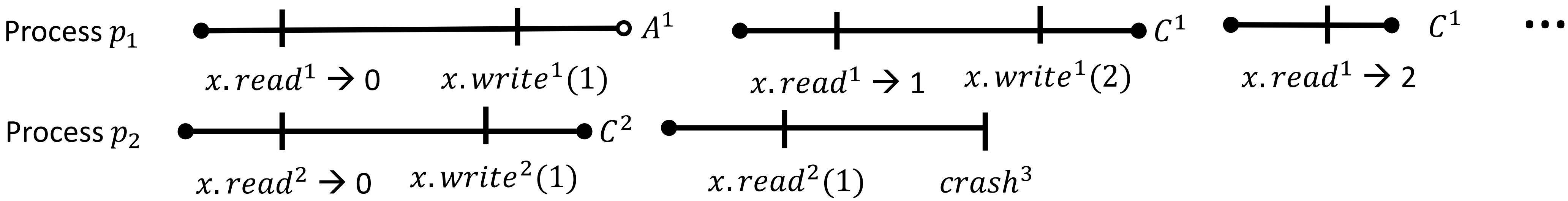
- **At least one correct process makes progress** in a fair history
  - Or there are no correct processes



- Stronger property than non-blocking
  - Blocking-freedom: Some operation receives a response
  - Introduced by Herlihy in 1991
- Weaker than local progress

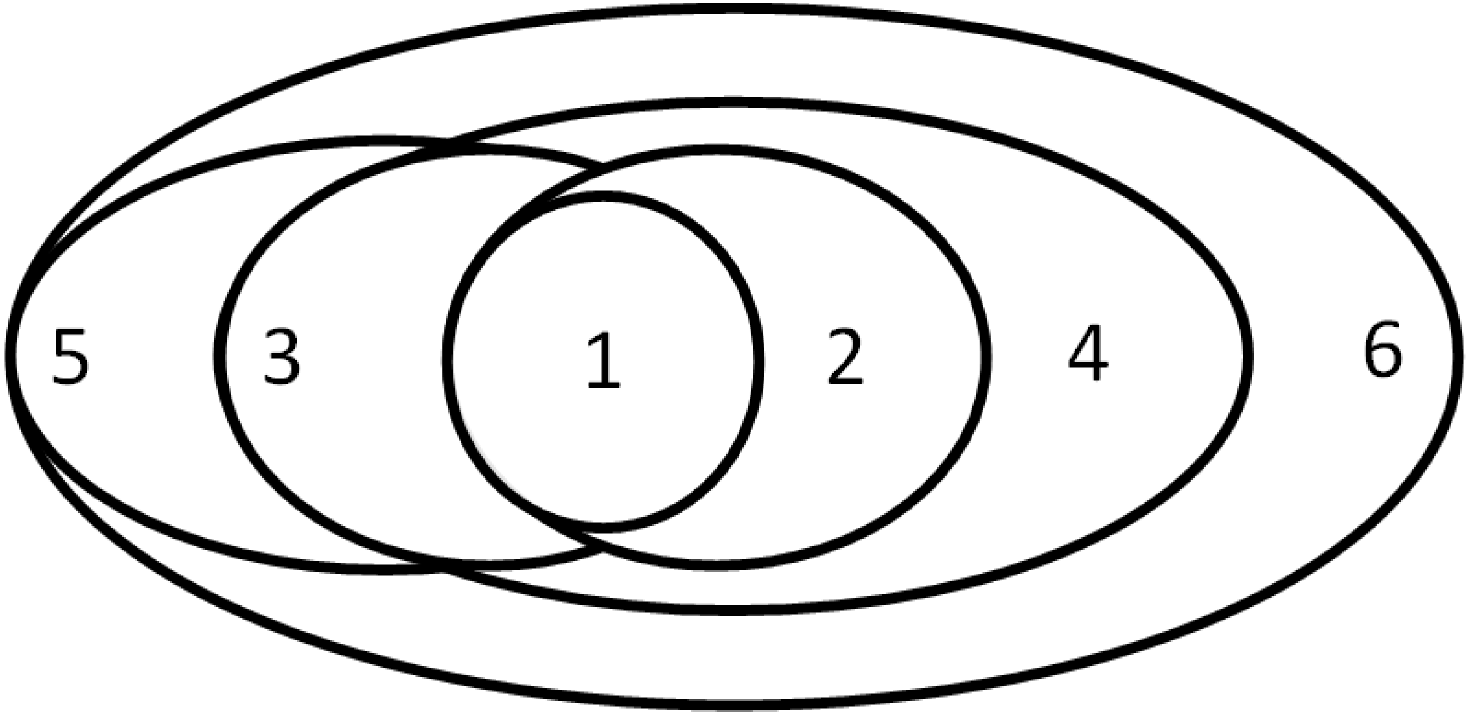
Solo Progress

- Every correct process makes progress in a fair history, **if it runs alone**
  - Or there are no correct processes



- Stronger property than obstruction-freedom
  - Obstruction-freedom: operations executed isolated receive result
  - Introduced by Herlihy et al. in 2003
- Weaker then global progress

Overview of Liveness Properties



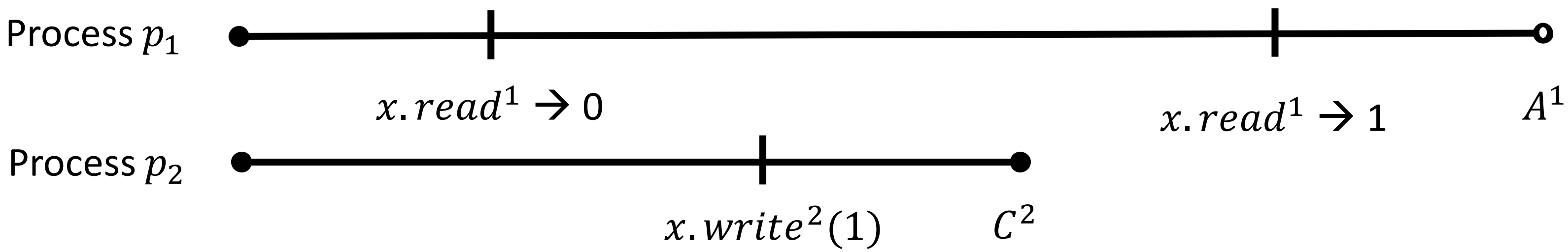
1 = local progress, 2 = wait-freedom, 3 = global progress,
4 = lock-freedom, 5 = solo progress, 6 = obstruction-freedom

Liveness and correctness conditions in a STM

Correctness Conditions

- History H is opaque, iff there is an equivalent sequential history H_s and all transactions in H_s are legal
  - Introduced by Guerrauoi and Kapalka in 2007

- History H is strict serializable, iff there is an equivalent sequential history H_s and all committed transactions in H_s are legal
  - Introduced by Papadimitriou in 1979



Non-existence of opacity and local progress

PADERBORN UNIVERSITY
The University for the Information Society

Theorem 1

For every fault-prone system there does not exists a STM-implementation that ensures local progress and opacity.

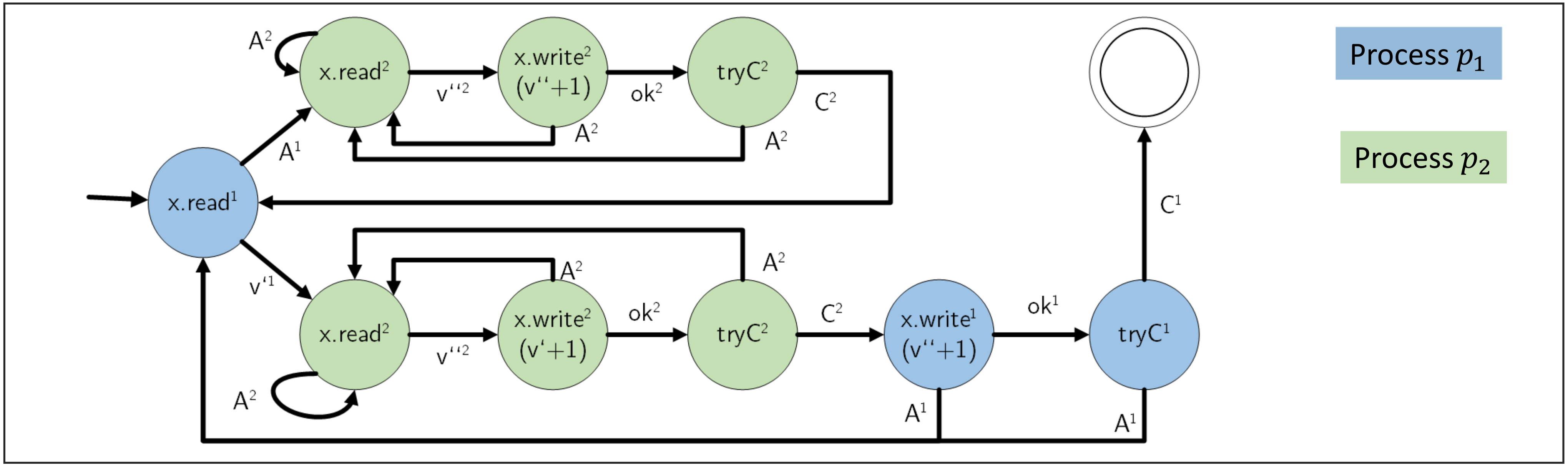
Non-existence of opacity and local progress

- Proof idea:
  - Assume local progress and opacity ensured
  - Construct strategy for two processes resulting in history violating local progress
    - For crash-prone systems
    - (For parasitic-prone systems)

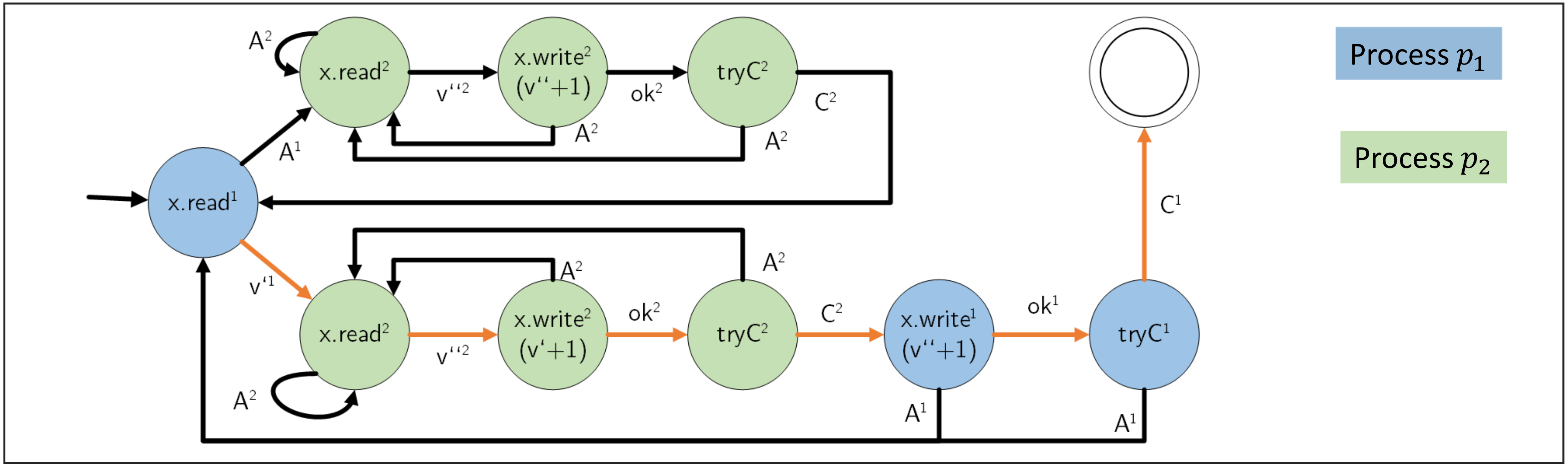
- Assumption:
  - Operations of a transaction not known in advance
  - Fault prone system

- Need to show:
  - Every correct process makes progress
  - Any resulting complete history is opaque

Non-existence of opacity and local progress - Strategy

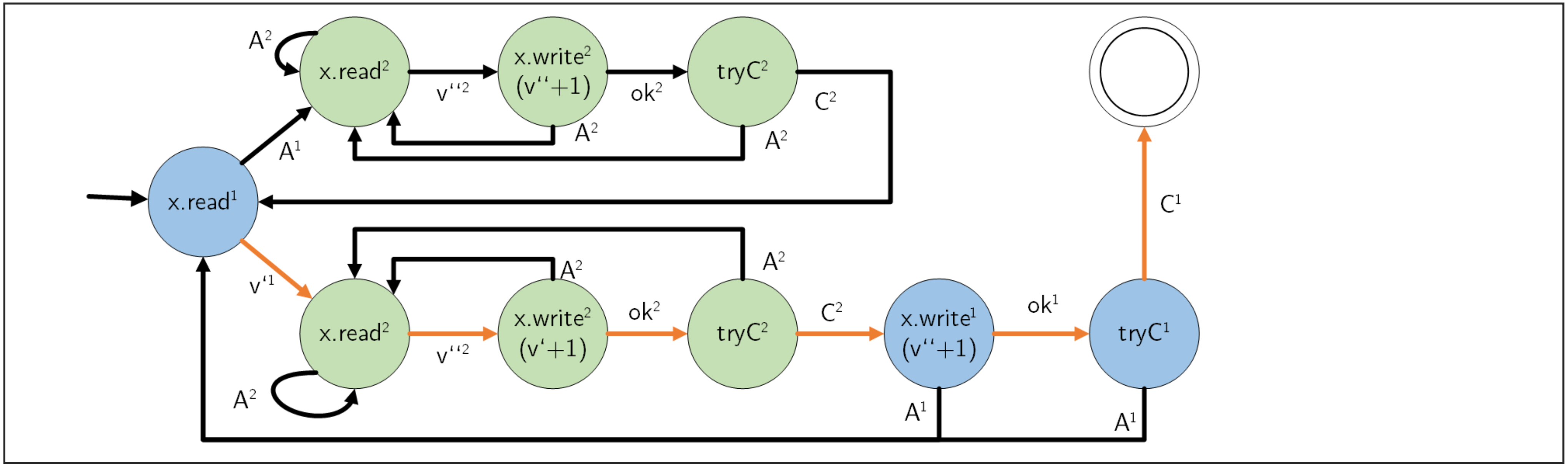


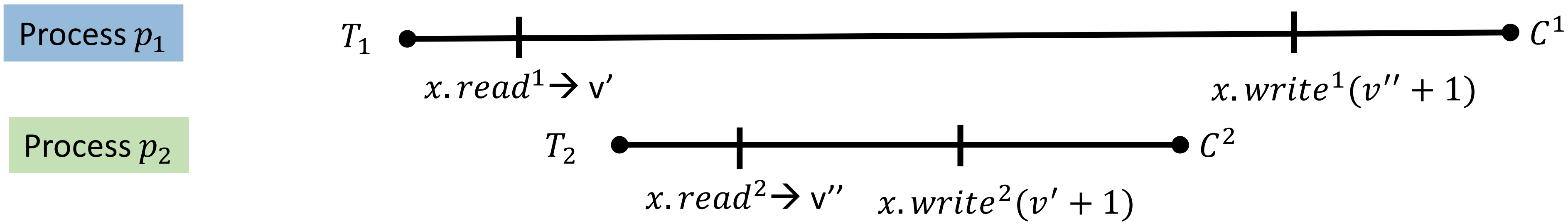
Non-existence of opacity and local progress - Proof



- Local progress ensured ⇔ all correct processes contain infinitely many commits
- Show that strategy produces infinite history
  - By nontermination of strategy
  - Strategy terminates, iff $tryC^1$ returns C^1

Non-existence of opacity and local progress - Proof

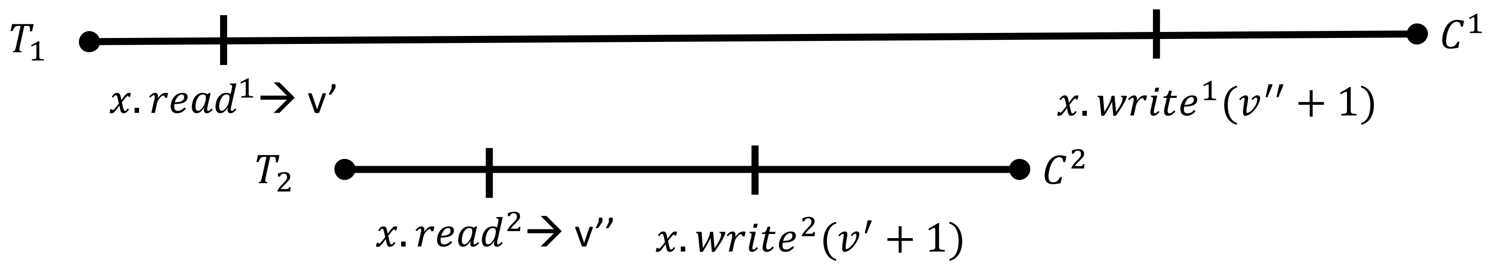




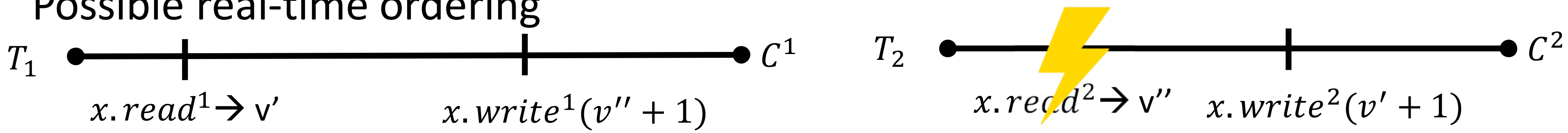
Impossibility of opacity and local progress - Proof

Process p_1

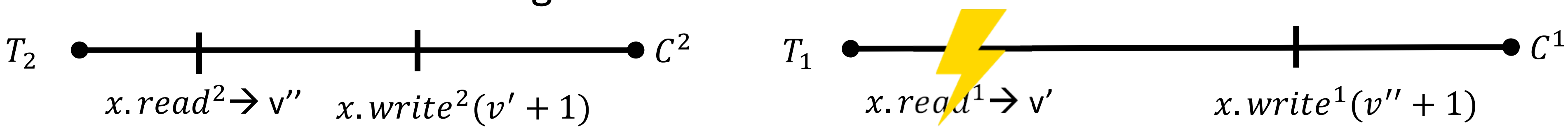
Process p_2



1. Possible real-time ordering

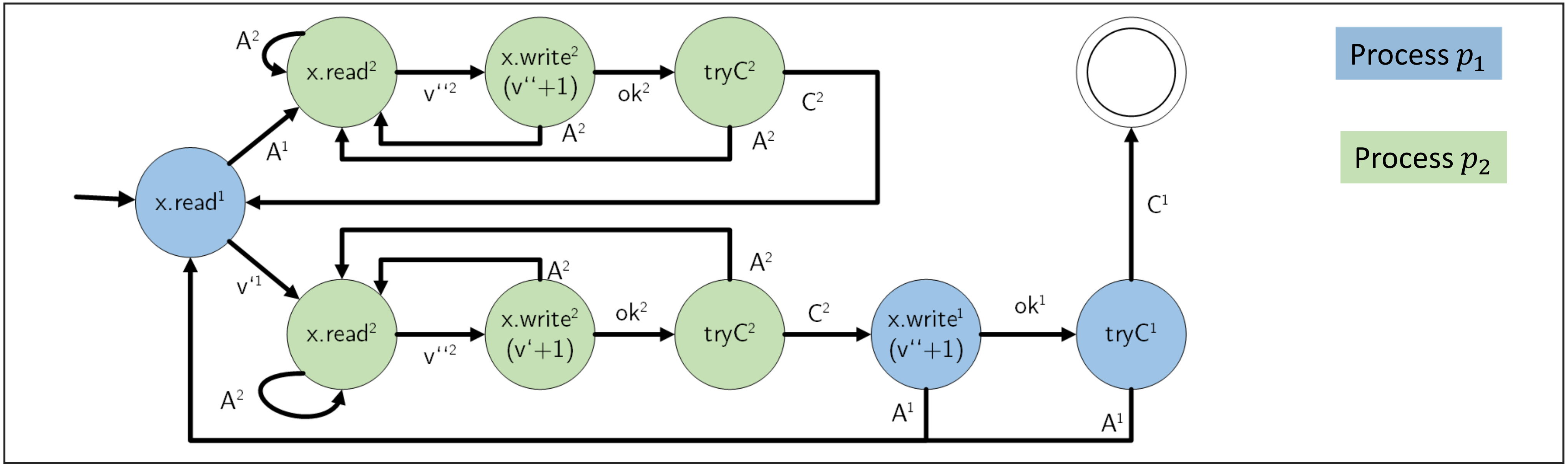


2. Possible real-time ordering



⇒ Strategy will never terminate, since both real time orderings not valid

Impossibility of opacity and local progress - Proof



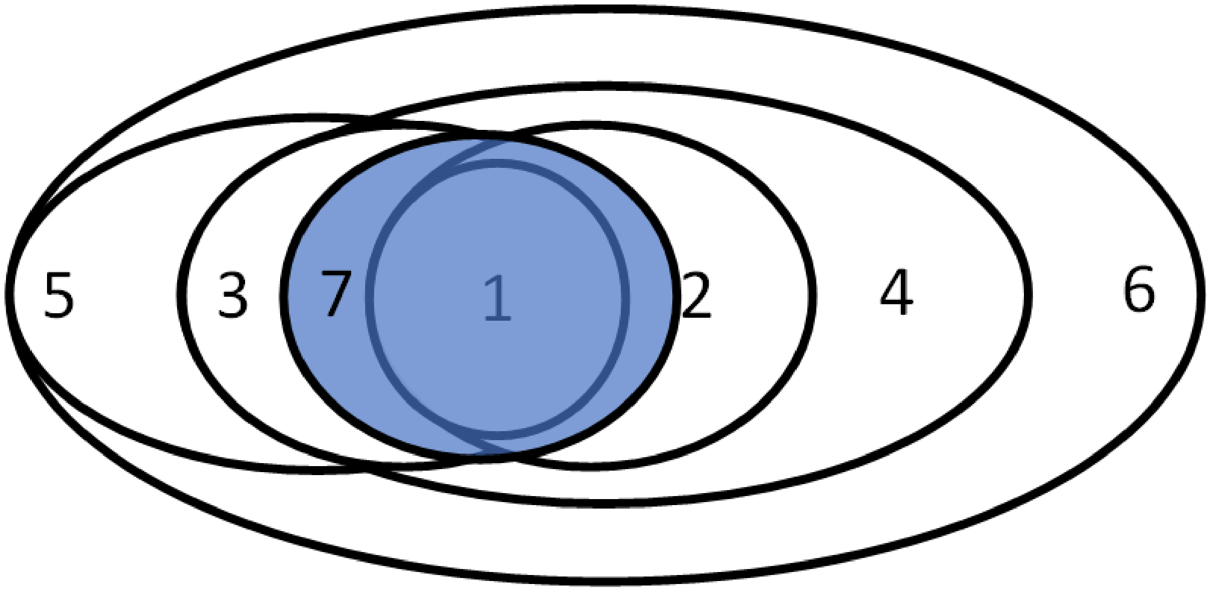
- Observation: In infinite execution, p_1 doesn´t make progress
- If p_1 does not crash $\Rightarrow$ contradiction to local progress (since p_1 makes no progress)
- p_1 crashes, iff $tryC^2$ never returns C^2
- p_2 will eventually make progress $\Rightarrow$ p_1 cannot be crashed

$\square$

Generalized result – further definitions

PADERBORN UNIVERSITY
The University for the Information Society

- Non-blocking liveness: progress for any correct process running alone
  - Local progress, global progress and solo progress are non-blocking
- Biprogressing: at least 2 correct processes makes progress
  - Local progress is biprogressing
  - Global progress and solo progress does not necessary ensure biprogressing



1 = local progress,
2 = wait-freedom,
3 = global progress,
4 = lock-freedom,
5 = solo progress,
6 = obstruction-freedom
5 = non-blocking
7 = biprogressing

Generalized result

PADERBORN UNIVERSITY
The University for the Information Society

Theorem 2

For every fault-prone system and every STM-liveness property L that is non-blocking and biprogressing there is no STM-implementation that ensures L and strict-serializability.

Summary

- Formal definition of liveness-conditions for STM
  - Local progress
  - Global progress
  - Solo progress

- Proven that opacity and local progress cannot be guaranteed in STM

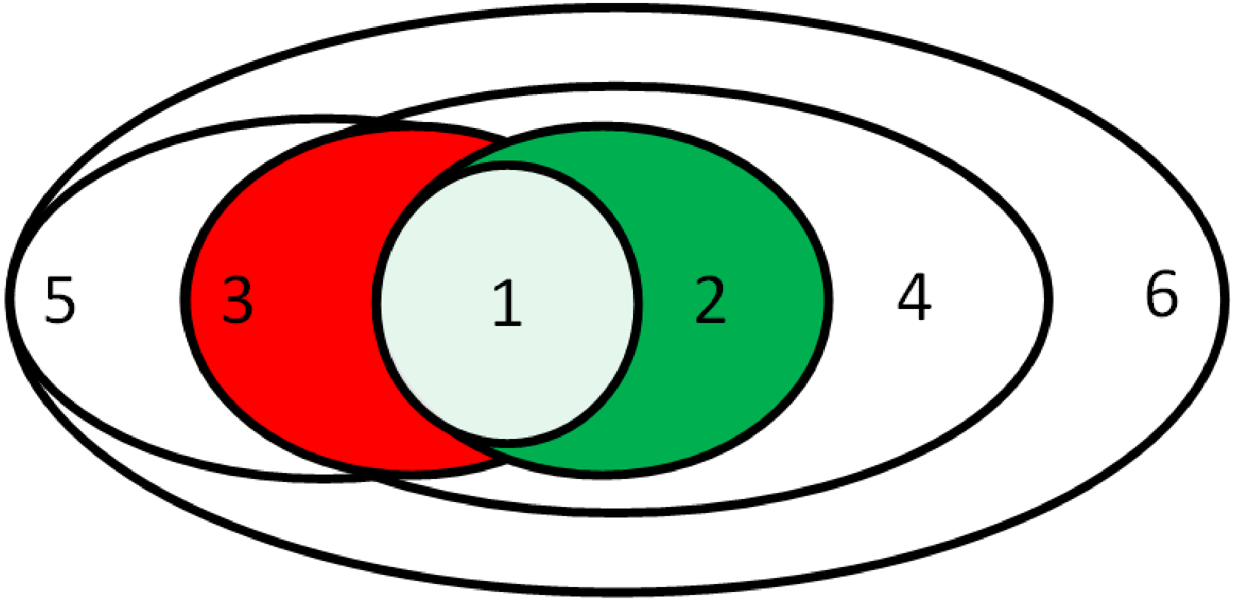
- Options for STMs ensuring liveness and correctness conditions:
  - Weaker liveness criteria (global progress and opacity work together (e.g. in OSTM)
  - Static processes (operations known in advance)
  - Assume fault-free system with deferred-update

Sources

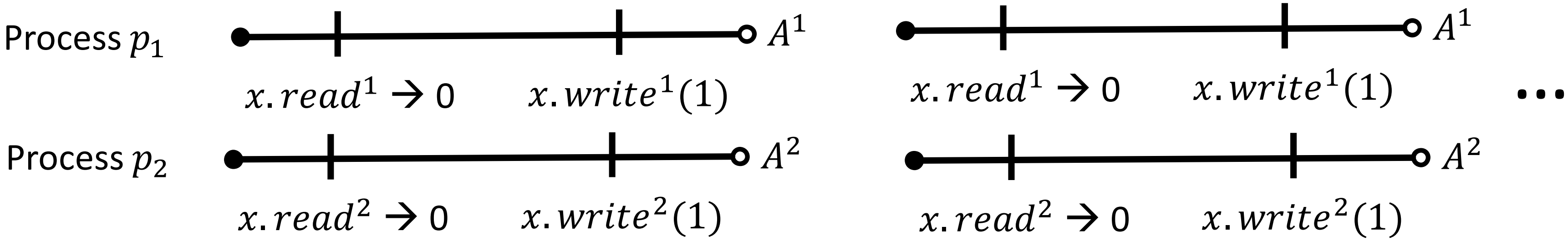
- Bushkov, V., Guerraoui, R.: Liveness in Transactional Memory. In: Transactional Memory. Foundations, Algorithms, Tools, and Applications. pp. 32–49. Springer, Cham (2015).
- Herlihy, M.P.: Wait-free synchronization. Toplas. 13, 124–149 (1991).
- Herlihy, M., Luchangco, V., Moir, M., Scherer, W.N.: Software transactional memory for dynamic-sized data structures. Proc. twenty-second Annu. Symp. Princ. Distrib. Comput. - Pod. ’03. 92–101 (2003).
- Guerraoui, R., Kapałka, M.: Opacity : A Correctness Condition for Transactional Memory. Tech. Rep. LPD-REPORT-2007-004, EPEL. (2007).
- Papadimitriou, C.H.: The serializability of concurrent database updates. J. ACM. 26, 631–653 (1979).
- K. Fraser. Practical Lock-Freedom. PhD thesis, University of Cambridge, 2003.

Backup slides

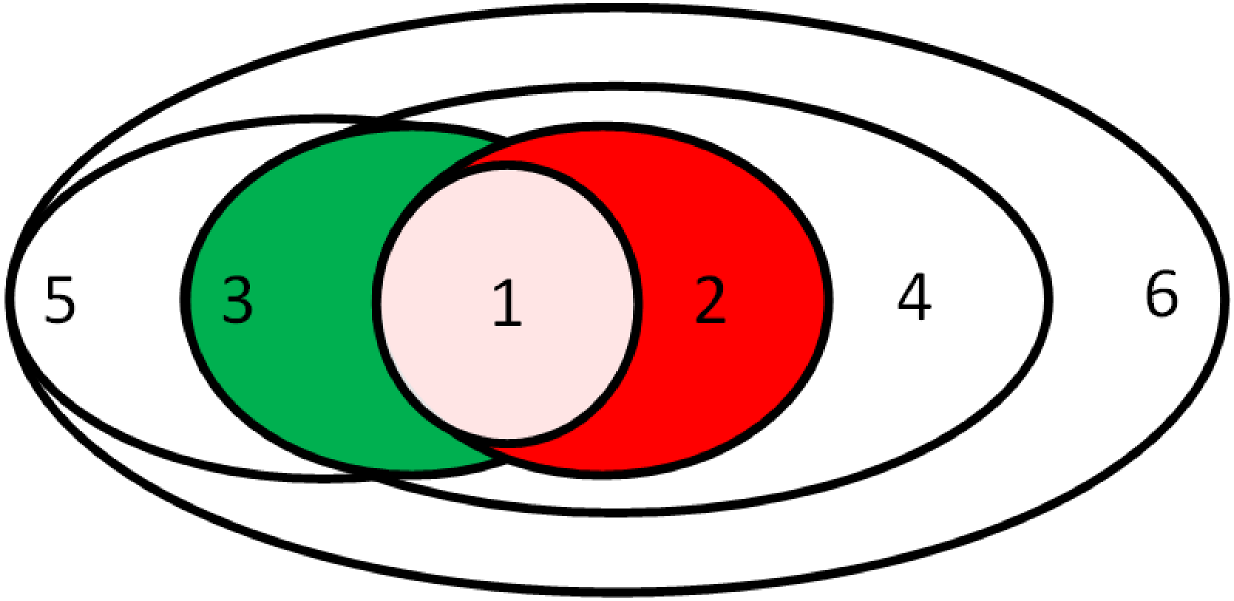
Wait-freedom incomparable to global progress



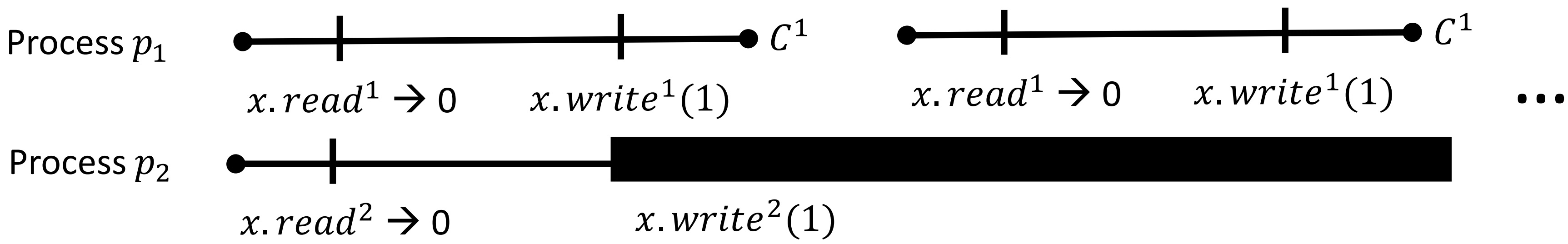
1 = local progress,
2 = wait-freedom,
3 = global progress,
4 = lock-freedom,
5 = solo progress,
6 = obstruction-freedom



Wait-freedom incomparable to global progress



1 = local progress,
2 = wait-freedom,
3 = global progress,
4 = lock-freedom,
5 = solo progress,
6 = obstruction-freedom



Generalized result

- Assume crash-prone system

- Proof structure:
  - Show that presented strategy produces infinite history
    - Same argument as before
    - Only different: Obtained history not strict serializable
  - Show that both processes are correct
    - p_2 cannot crash
    - p_1 crashes iff p_2 receives infinitely many A^2. (impossible due to non-blocking)
  - Show contradiction to biprogressing
    - Both processes correct
    - p_2 makes progress, p_1 does not

$\square$