

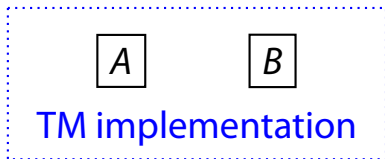
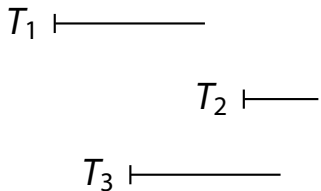
Do We Need a New Theory for TMs?

Michał Kapałka

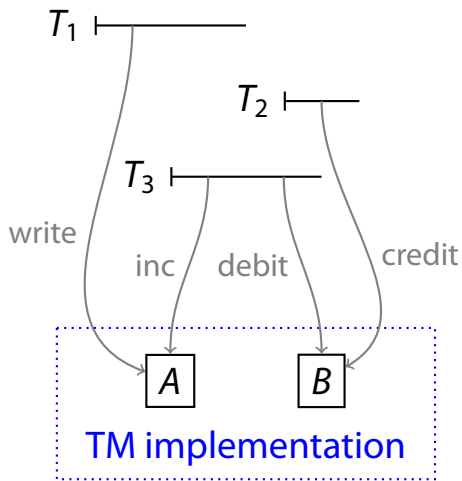
EPFL, Switzerland

Yes, indeed

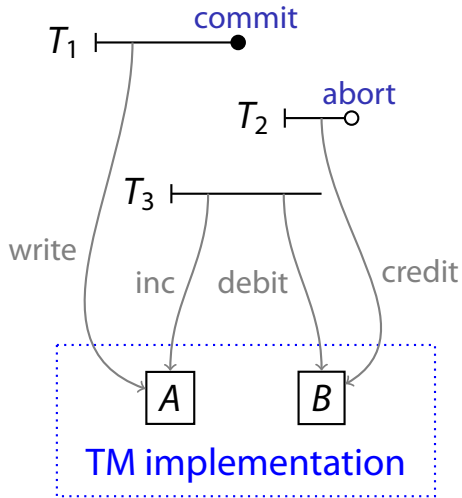
Transactional Memory



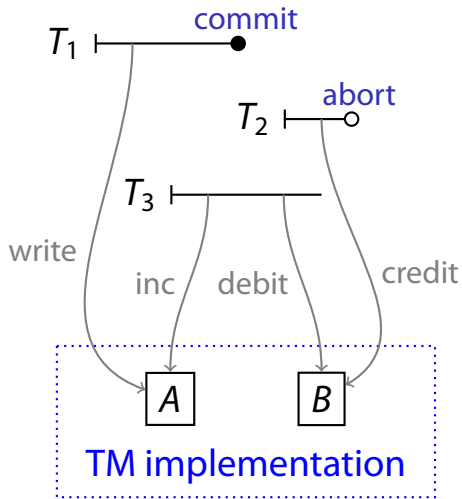
Transactional Memory



Transactional Memory



Transactional Memory



When is it correct?

When is a history correct?



properties of a correct history



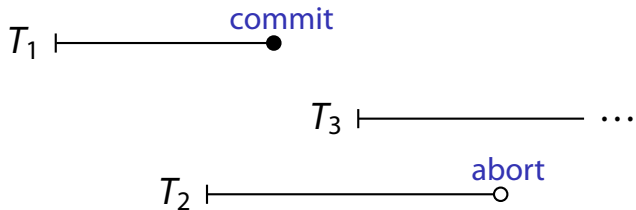
safety



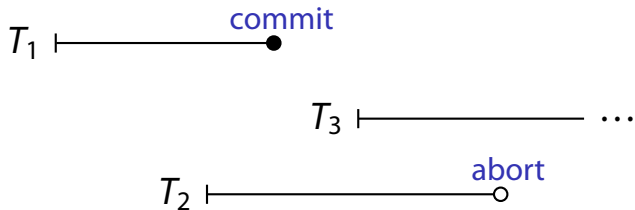
liveness

safety

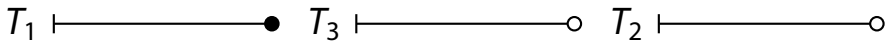
Safety of a TM



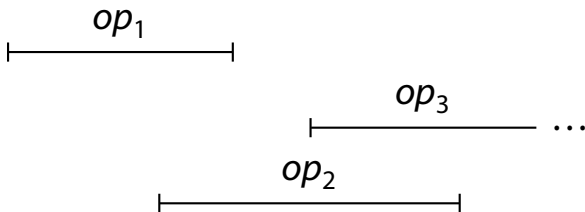
Safety of a TM



"looks like"

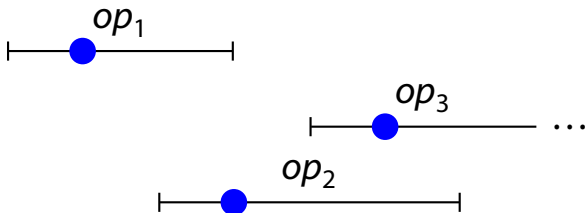


Why Not Linearizability?



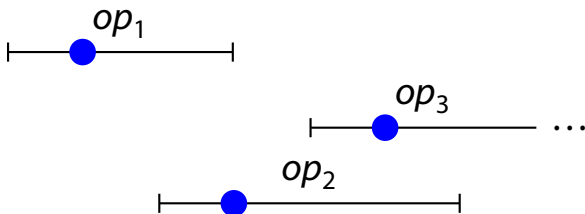
[Herlihy and Wing '90]

Why Not Linearizability?



[Herlihy and Wing '90]

Why Not Linearizability?



Problem: different model / level of abstraction

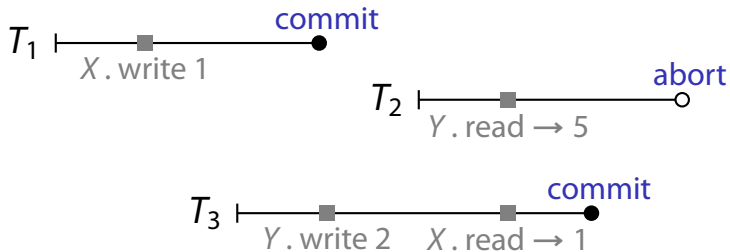
“Serializability is appropriate for systems (...) in which it must be easy for application programmers to preserve complex application-specific invariants spanning multiple objects.”

[Herlihy and Wing '90]

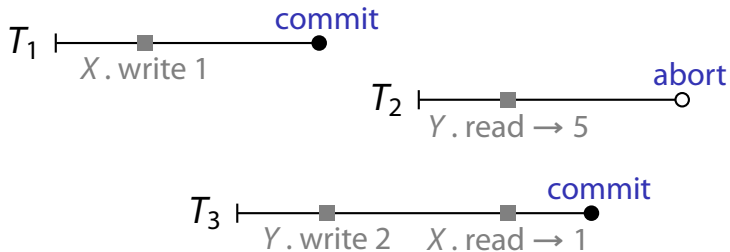
Why Not Serializability?

- (strict) conflict / view / final-state serializability [Papadimitriou '79]
- 1-copy serializability [Bernstein and Goodman '83]
- global atomicity [Weihl '89]

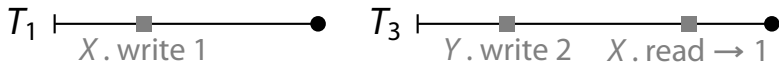
Why Not Serializability?



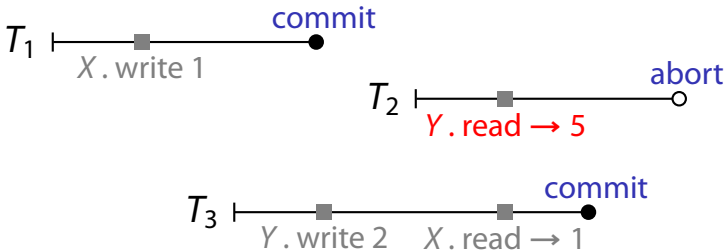
Why Not Serializability?



there exists:



Why Not Serializability?



Problem: no guarantees for aborted / live transactions

All transactions must observe
consistent state.

Observing Inconsistent State

Invariant: $X < Y$

T_1 reads: $X = 10, Y = 2$

$v = 1/(Y - X + 8)$

array[$Y - X$] = v ;

for($k = X, i = Y; k \neq i; k++$) **func**(k);

safeObjs[$Y - X$].**method**();

Observing Inconsistent State

Invariant: $X < Y$

T_1 reads: $X = 10, Y = 2$

$v = 1/0$

`array[-8] = v;`

`for(k = 10, i = 2; k \neq i; k++) func(k);`

`safeObjs[-8].method();`

Other Safety Properties

Recoverability [Hadzilacos '88]

Problems:

- restricted model
- still too weak

Other Safety Properties

Recoverability [Hadzilacos '88]

Problems:

- restricted model
- still too weak

Rigorous scheduling [Breitbart et al. '91]

Problems:

- restricted model
- too strong

liveness

Why Not Wait-Freedom?

Every (correct) process that invokes an operation eventually returns from this operation.

[Herlihy '91]

Problem: different model / level of abstraction

TM “Wait-Freedom”

Every process that keeps executing transactions eventually commits a transaction.

TM “Wait-Freedom”

Every process that keeps executing transactions eventually commits a transaction.

Problem: fundamental difference:

(*async. system with crashes*)

- wait-freedom: every atomic object implementable from CAS [Herlihy '91]
- TM “wait-freedom”: cannot be implemented **at all** [Guerraoui and Kapałka '09]

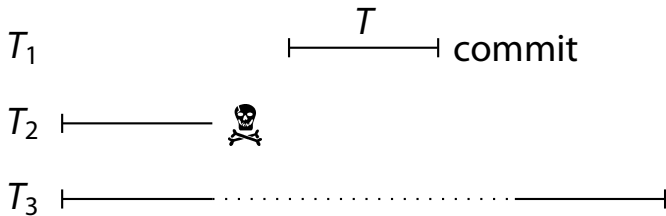
Why Not Obstruction-Freedom?

“A synchronization mechanism is obstruction-free if any thread that runs by itself long enough makes progress (...)”

[Herlihy et al. '03]

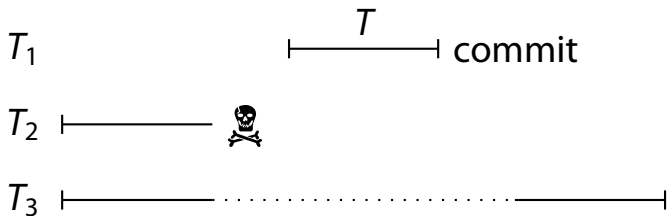
Problem: different model / level of abstraction

Obstruction-Free TMs



+ wait-freedom of individual operations

Obstruction-Free TMs



+ wait-freedom of individual operations

Problem: fundamental difference:
an obstruction-free TM cannot be implemented
using only read-write registers [Guerraoui and Kapalka '08]
([async. system with crashes](#))

Lock-based TMs?

Transactions @ EFPL: lpd.epfl.ch

Safety: opacity (PPoPP'08)

Liveness:

- TM obstruction-freedom (SPAA'08)
- strong/weak progressiveness
(lock-based TMs; POPL'09)
- strongest liveness possible (tech report '09)

joint work with Rachid Guerraoui