Abort Free SemanticTM by Dependency Aware Scheduling of Transactional Instructions

Shlomi Dolev
Ben-Gurion University of the Negev
Israel

Panagiota Fatourou
University of Crete & FORTH-ICS
Greece

Eleftherios Kosmas
University of Crete & FORTH-ICS
Greece

WTM 2013

Introduction

- When two transactions **conflict**, most TM systems **abort** one of them to ensure consistency

Two transactions **conflict**, if they both access the same t-variable and at least one of these accesses is a write

- **Ideally:**
  - **All** transactions should commit
  - **Parallelism** should not be sacrificed

- TMs that **never abort** transactions are highly desirable
  - support of irrevocable transactions
  - avoid the cost of re-executing aborted transactions

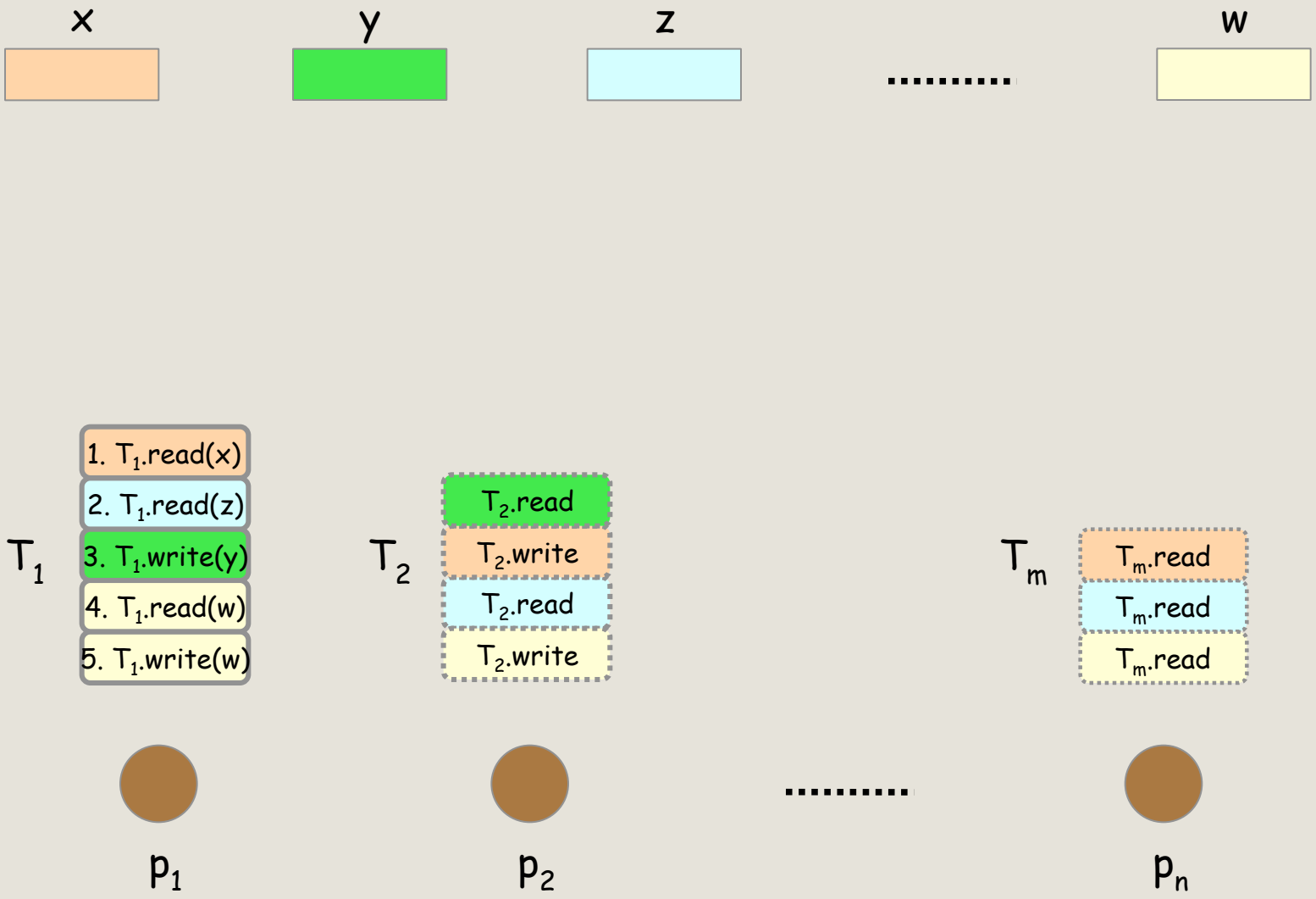
Our Result

SemanticTM

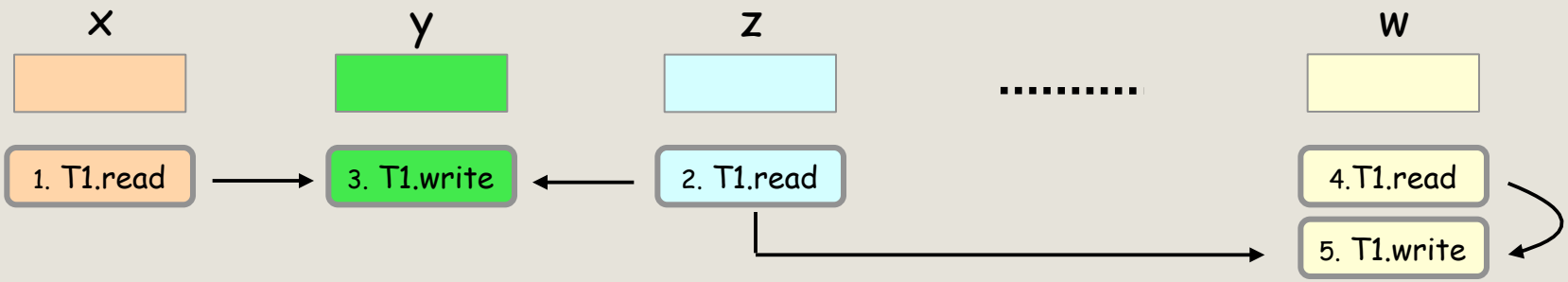
- an opaque TM algorithm
- no transaction ever aborts
  - guarantees wait-freedom/local progress for the execution of transactions
- fine-grain parallelism at the transactional instruction level

An execution of a TM algorithm is **opaque** if it satisfies strict serializability and active transactions read "consistent" values for t-variables

Transaction Level Parallelism



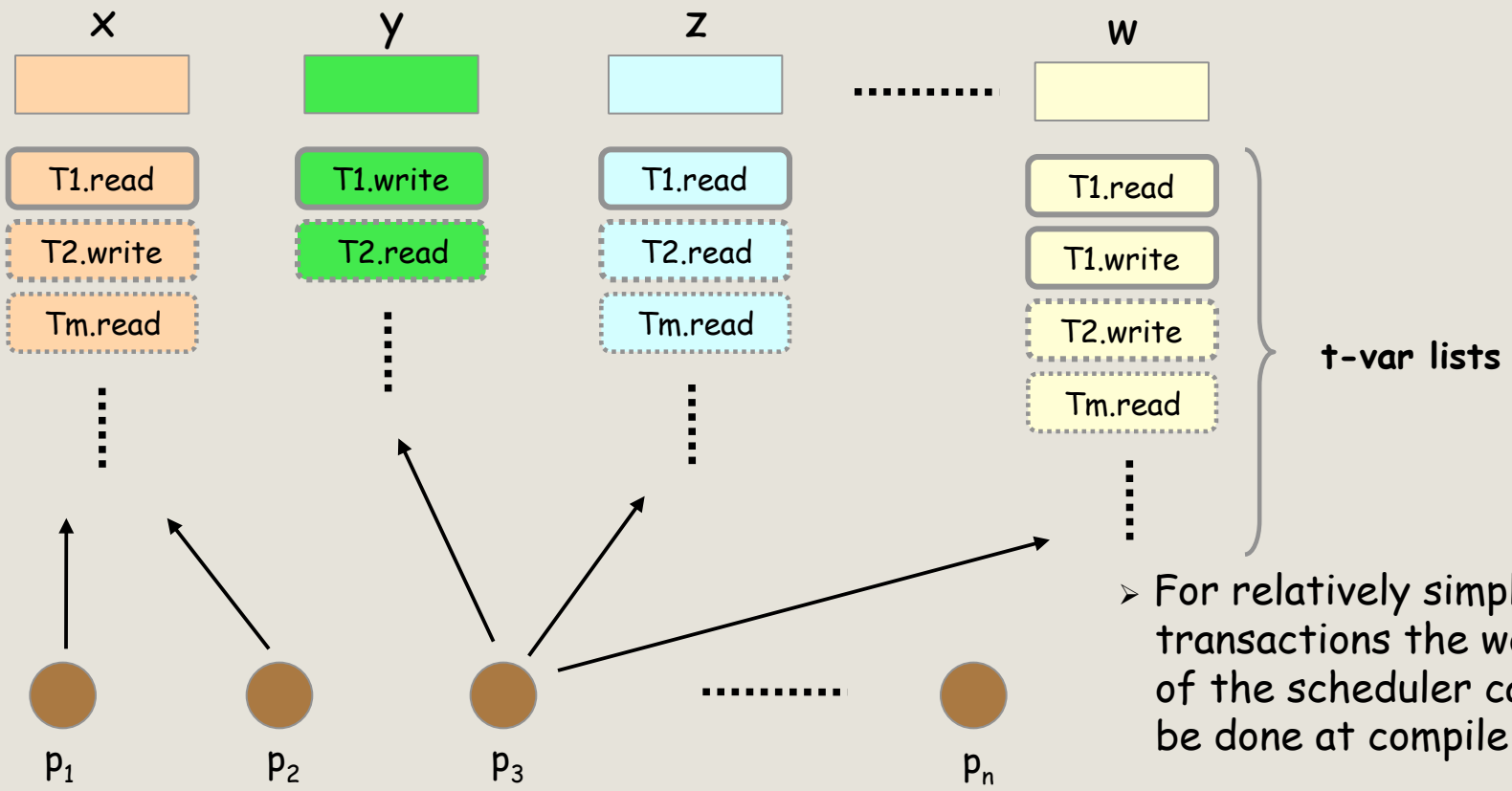
Transactional Instruction Level Parallelism



- **dependencies** may exist between instructions of the same transaction

SemanticTM

- a **scheduler** places transactional intructions into **t-var lists**
  - ➔ together with their dependencies
- transactions are processed by the scheduler the **one after the other**
- each process **randomly chooses** a t-var list and executes its **ready** instructions
- **no conflicts** occur between transactions



- For relatively simple transactions the work of the scheduler can be done at compile time.

Dependencies

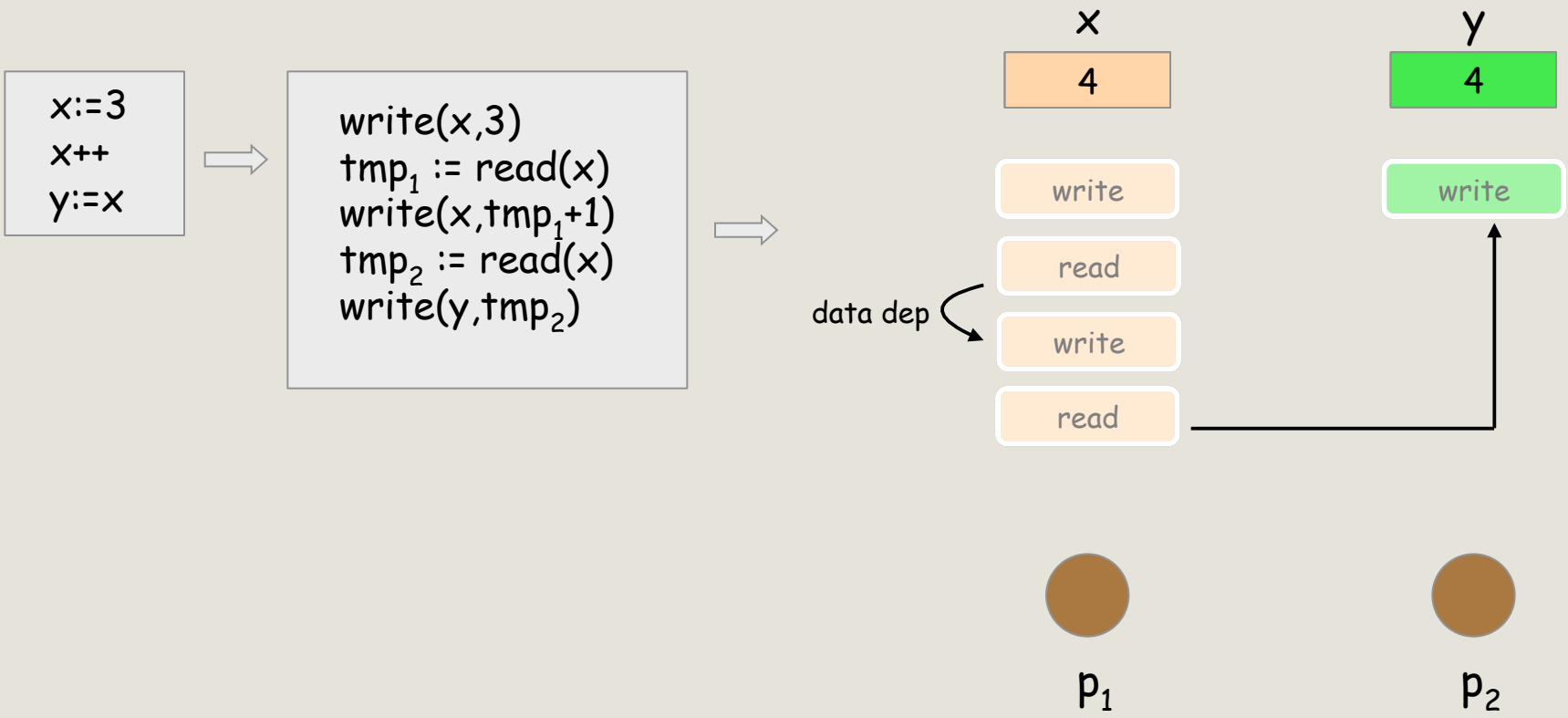
- If the execution of a transactional instruction e_2 **depends** on the execution of a transactional instruction e_1
  - there is a **dependency** between e_1 and e_2
    - **input** dependency for e_2
    - **output** dependency for e_1

```
x:=3
x++
y:=x
```

- A dependency between
  - a read and a write instruction → **data** dependency
  - a cond and a read (or a write) instruction → **control** dependency

```
if ( x+y = 5) then
  x:=1
```

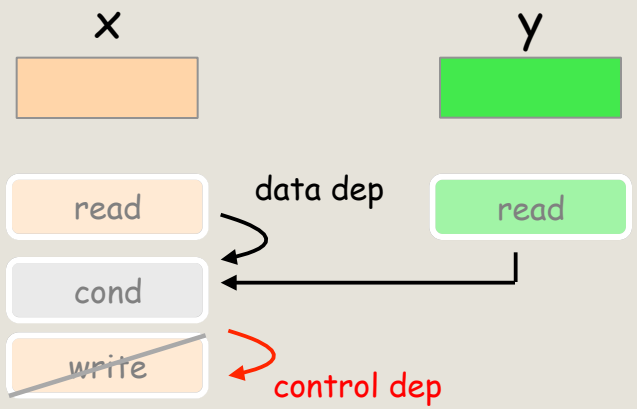
Reads and Writes



Conditionals – if statement

```
if(x+y = 5) then
   x := 1
```

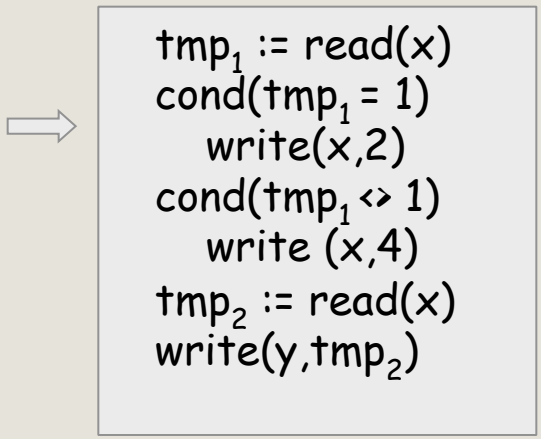
tmp_1 := read(x)
tmp_2 := read(y)
tmp_3 := tmp_1+tmp_2
cond(tmp_3=5)
    write(x,1)

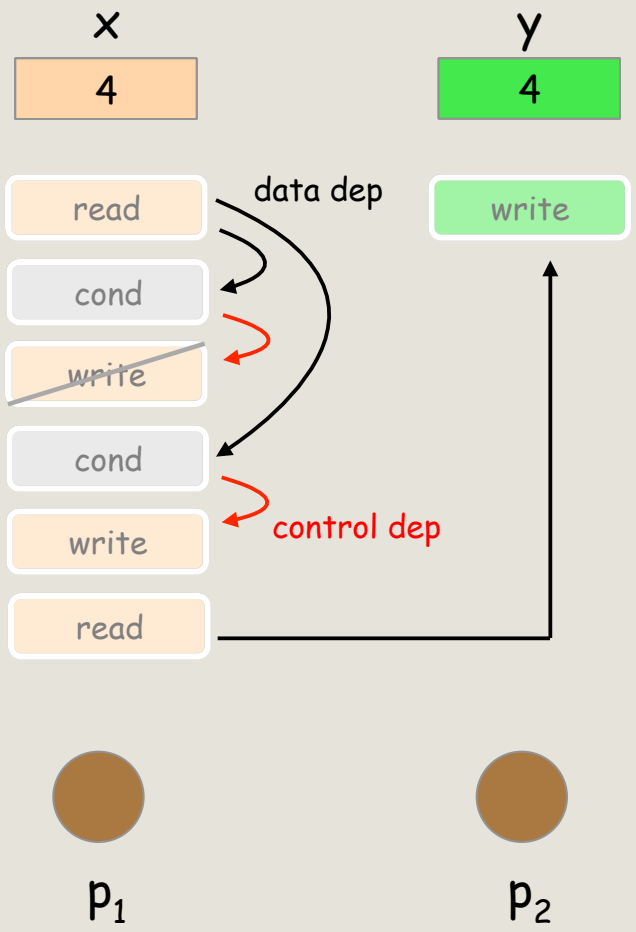


- If cond evaluates to TRUE then the write is executed
- Otherwise it is invalidated

Conditionals – if..else statements

```
if(x = 1) then
    x := 2
else x := 4
y := x
```

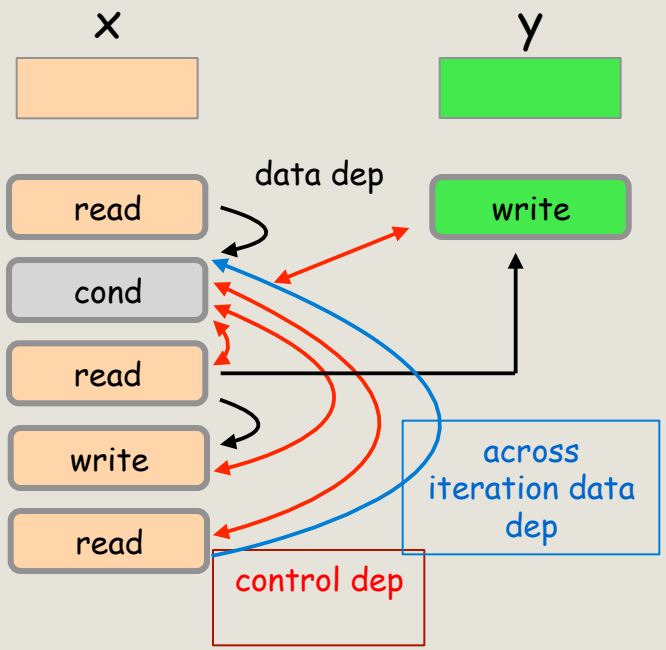




Loops

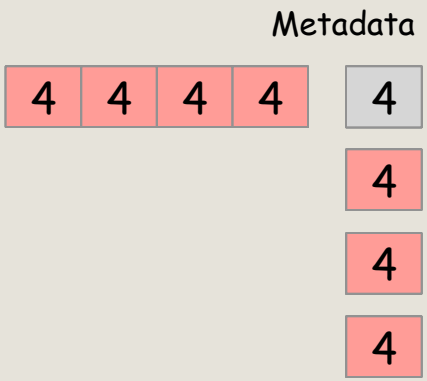
```
while(x < 10) do
    y := x
    x := 2* x
```

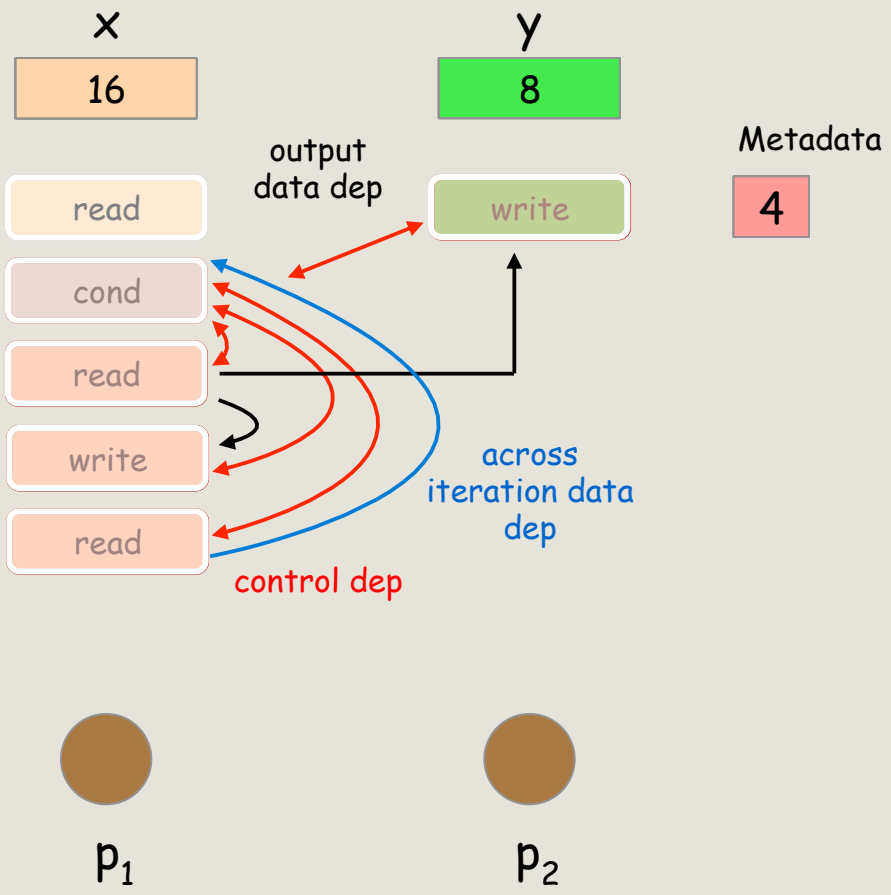
```
tmp1 := read(x)
cond(tmp1 < 10, loop)
    tmp2 := read(x)
    write(y,tmp2)
    tmp3 := 2 * tmp2
    write(x,tmp3)
    tmp1 := read(x)
```



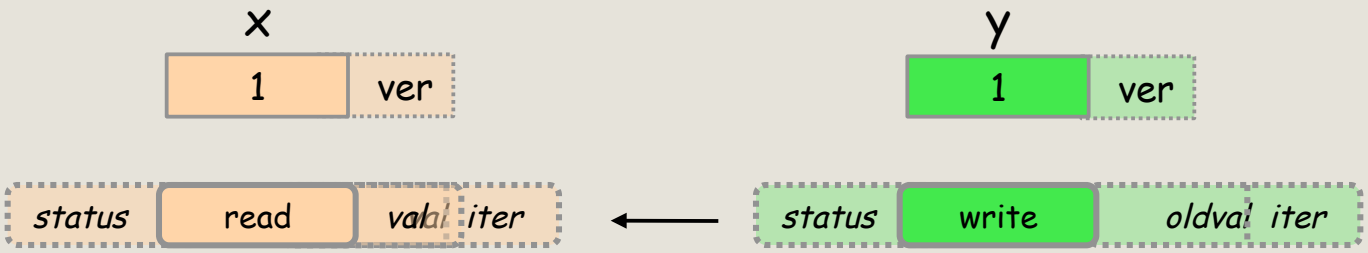
Loops

```
while (x < 10)
  y := x
  x := x*2
```





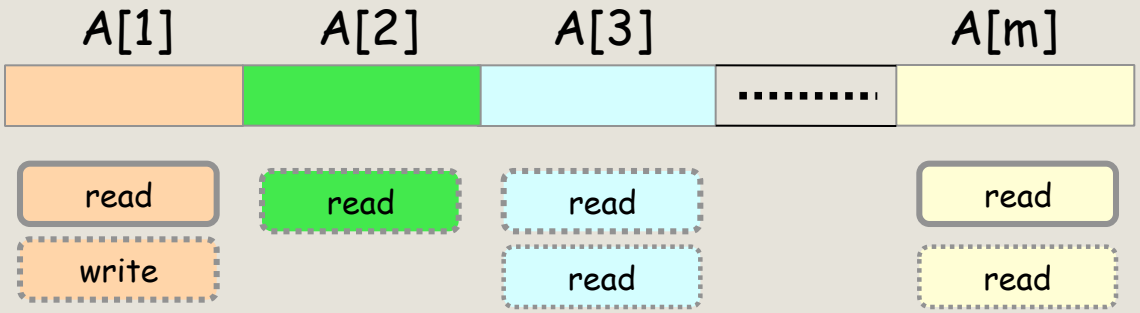
Synchronization

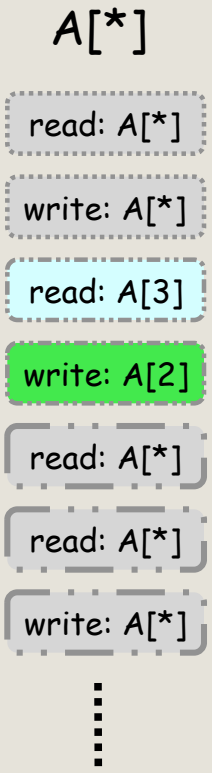


- for each **transactional instruction**
  - status (ACTIVE, DONE)
- together with the **value** of each **input data dependency**
  - iteration number
  - CAS object
- for each **control dependency**
  - CAS object
- for the **value** of each **t-variable**
  - version
  - CAS object
  - update using oldvalue
    - iteration number
    - CAS object

Extensions

- Support of transactions accessing t-variables that are known at run time
  - accessing an element of an array





- Dynamic transactions
  - similar strategy as above, if we consider the memory heap as an array
- Nested conditionals & loops (cond c2 in the block of outer cond c1)
  - add a control dependency from c1 to c2 but not to the instructions of the block of c2.

Conclusion & Future Work

Summary

- We presented SemanticTM:
  - executes transactions without ever causing any aborts
  - parallelism is fine-grained achieved at the level of transactional instructions
- Blocking version of SemanticTM

Future Work

- Implement an optimized version of SemanticTM
- Experimental Study

Thank you!

QUESTIONS?