Verification of weak memory models

Elli Anastasiadi

OPCT 2023, Bertinoro, Italy

26-30 June

UPPSALA
UNIVERSITET

1. Weak memory
   - Why - how - examples

2. Verification
   - Basic problems
   - Basic principles

UPPSALA
UNIVERSITET

Starting point: distributed programs & architectures.

UPPSALA
UNIVERSITET

Starting point: distributed programs & architectures.
$\rightarrow$ Need for verification.

UPPSALA
UNIVERSITET

Starting point: distributed programs & architectures.
$\rightarrow$ Need for verification. ...so far so good.

UPPSALA
UNIVERSITET

Starting point: distributed programs & architectures.
→ Need for verification. ...so far so good.

Necessary: a model for distributed execution.

Starting point: distributed programs & architectures.
→ Need for verification. ...so far so good.

Necessary: a model for distributed execution.
→ Before weak memory: interleaving.

UPPSALA
UNIVERSITET

Starting point: distributed programs & architectures.
→ Need for verification. ...so far so good.

Necessary: a model for distributed execution.
→ Before weak memory: interleaving.

An example (Dekker protocol):

$P_1 :=$	$P_2 :=$
$x := 1$	$y := 1$
$a := y$	$b := x$
if $a = 0$ then	if $b = 0$ then
CS_1	CS_2

Assertion: not(CS_1 and CS_2)

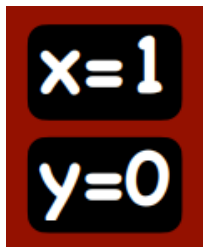
UPPSALA UNIVERSITET

Starting point: distributed programs & architectures.
→ Need for verification. ...so far so good.

Necessary: a model for distributed execution.
→ Before weak memory: interleaving.

An example (Dekker protocol):

$P_1 :=$	$P_2 :=$
$\mathbf{x := 1}$	$y := 1$
$a := y$	$b := x$
if $a = 0$ then	if $b = 0$ then
CS_1	CS_2



Assertion: not(CS_1 and CS_2)

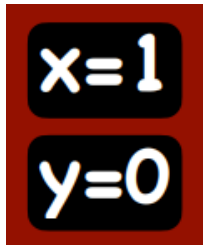
UPPSALA UNIVERSITET

Starting point: distributed programs & architectures.
→ Need for verification. ...so far so good.

Necessary: a model for distributed execution.
→ Before weak memory: interleaving.

An example (Dekker protocol):

$P_1 :=$	$P_2 :=$
$x := 1$	$y := 1$
a:=y	$b := x$
if $a = 0$ then	if $b = 0$ then
CS_1	CS_2



Assertion: not(CS_1 and CS_2)

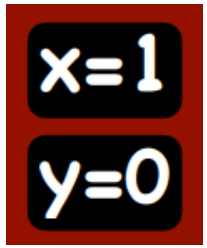
UPPSALA UNIVERSITET

Starting point: distributed programs & architectures.
→ Need for verification. ...so far so good.

Necessary: a model for distributed execution.
→ Before weak memory: interleaving.

An example (Dekker protocol):

$P_1 :=$	$P_2 :=$
$x := 1$	$y := 1$
$a := y$	$b := x$
if $a = 0$ **then**	if $b = 0$ then
CS_1	CS_2



Assertion: not(CS_1 and CS_2)

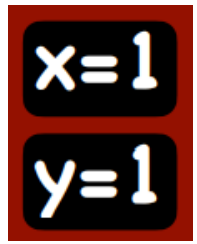
UPPSALA UNIVERSITET

Starting point: distributed programs & architectures.
→ Need for verification. ...so far so good.

Necessary: a model for distributed execution.
→ Before weak memory: interleaving.

An example (Dekker protocol):

$P_1 :=$	$P_2 :=$
$x := 1$	$\mathbf{y := 1}$
$a := y$	$b := x$
if $a = 0$ then	if $b = 0$ then
CS_1	CS_2



Assertion: not(CS_1 and CS_2)

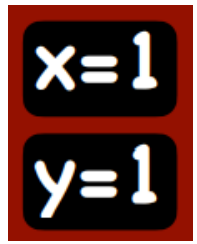
UPPSALA UNIVERSITET

Starting point: distributed programs & architectures.
→ Need for verification. ...so far so good.

Necessary: a model for distributed execution.
→ Before weak memory: interleaving.

An example (Dekker protocol):

$P_1 :=$	$P_2 :=$
$x := 1$	$y := 1$
$a := y$	**b:=x**
if $a = 0$ then	if $b = 0$ then
CS_1	CS_2



Assertion: not(CS_1 and CS_2)

UPPSALA UNIVERSITET

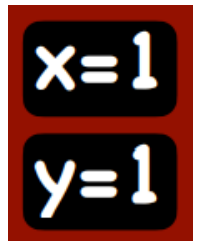
Starting point: distributed programs & architectures.
→ Need for verification. ...so far so good.

Necessary: a model for distributed execution.
→ Before weak memory: interleaving.

An example (Dekker protocol):

$P_1 :=$	$P_2 :=$
$x := 1$	$y := 1$
$a := y$	$b := x$
if $a = 0$ then	if $b = 0$ then
CS_1	CS_2

Assertion: not(CS_1 and CS_2) ✓

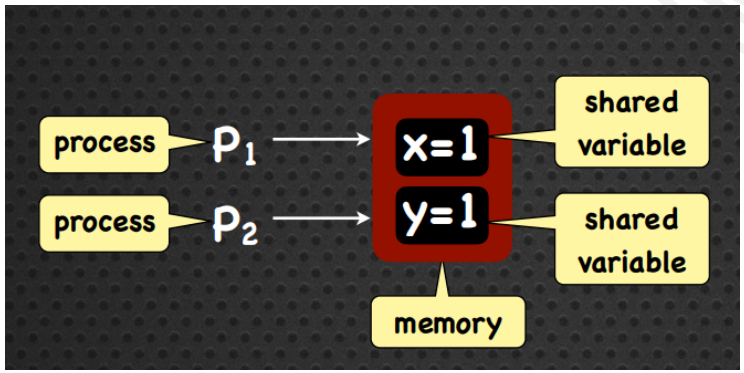


UPPSALA UNIVERSITET

Interleaving: implementation

Interleaving: implementation

- atomic writes,
- read-from-memory



UPPSALA UNIVERSITET

Interleaving: formally

UPPSALA
UNIVERSITET

Interleaving: formally

Sequential consistency - SC

On a global trace, the evens of a process will occur in the order stated locally for the process.

Or: only the events of different processes can be shuffled.

So what is weak memory?

So what is weak memory?

Rule of thumb: Anything “below” interleaving.

UPPSALA
UNIVERSITET

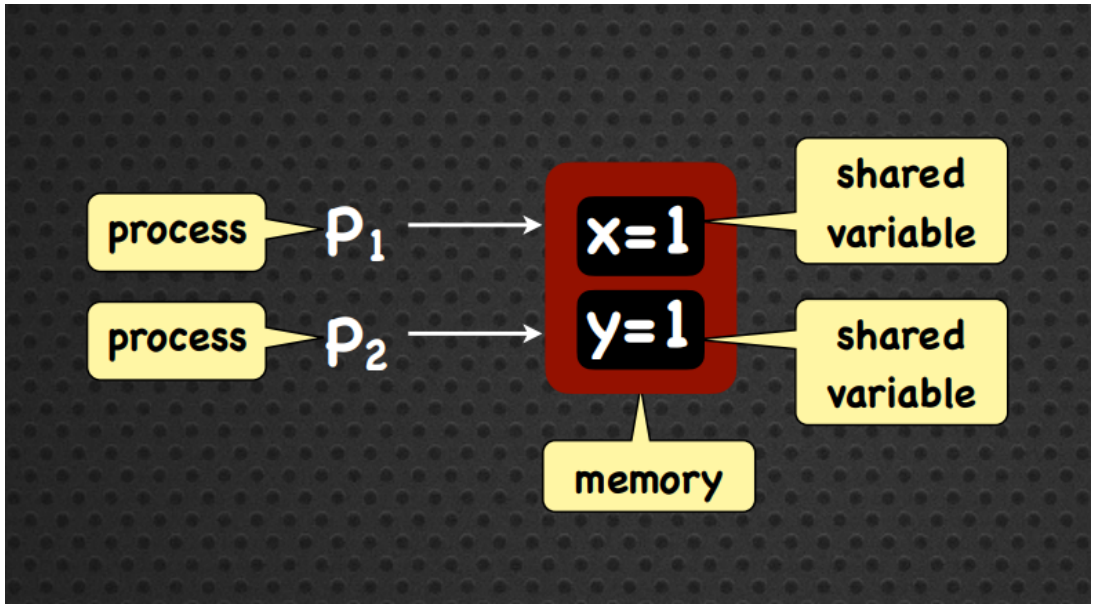
So what is weak memory?

Rule of thumb: Anything "below" interleaving.

...extra reorderings

How?

How?

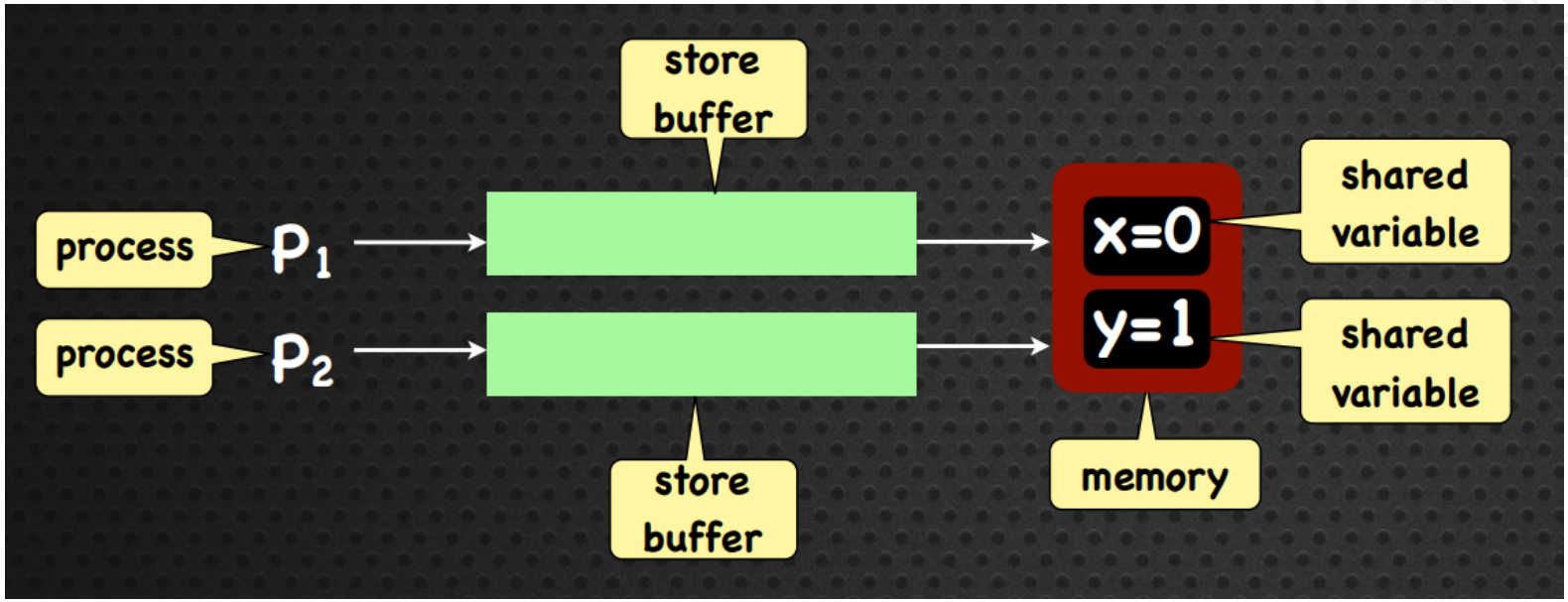


UPPSALA UNIVERSITET

How?


process
P1
process
P2
x=0
y=1
memory
shared variable
shared variable

How?



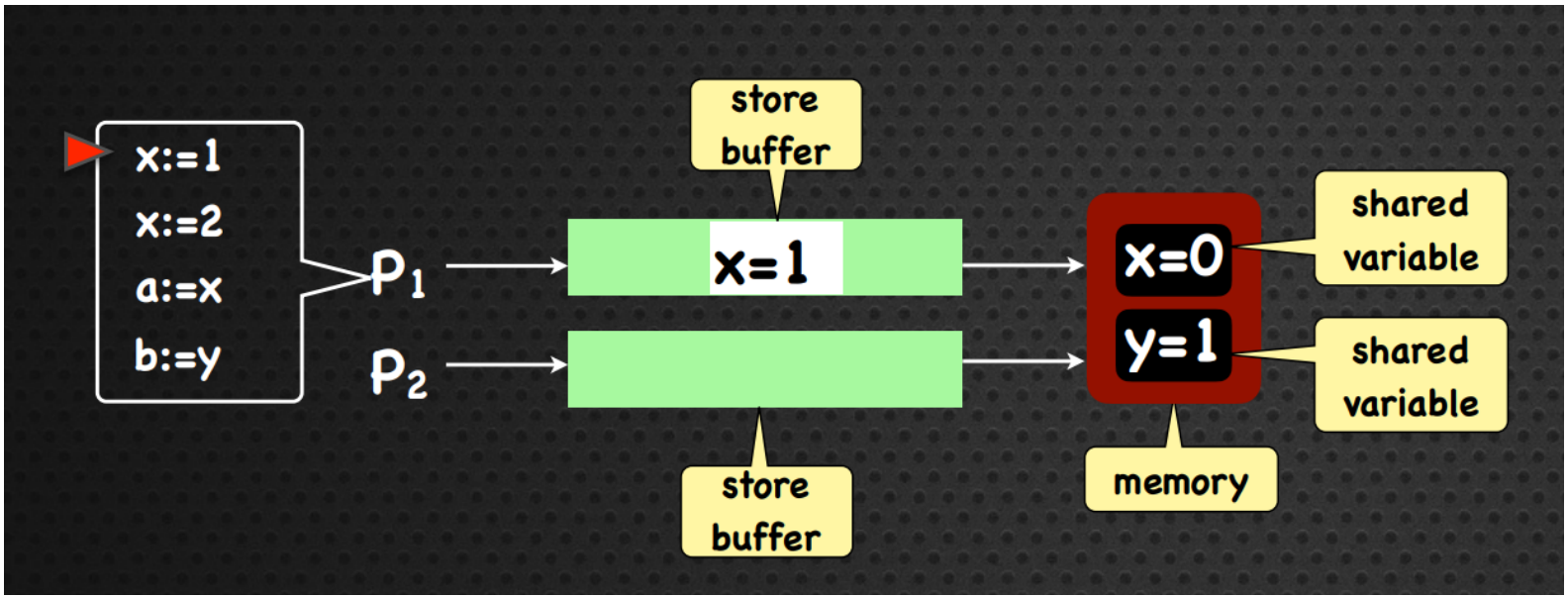
UPPSALA
UNIVERSITET

An example: total store order (TSO)

UPPSALA
UNIVERSITET

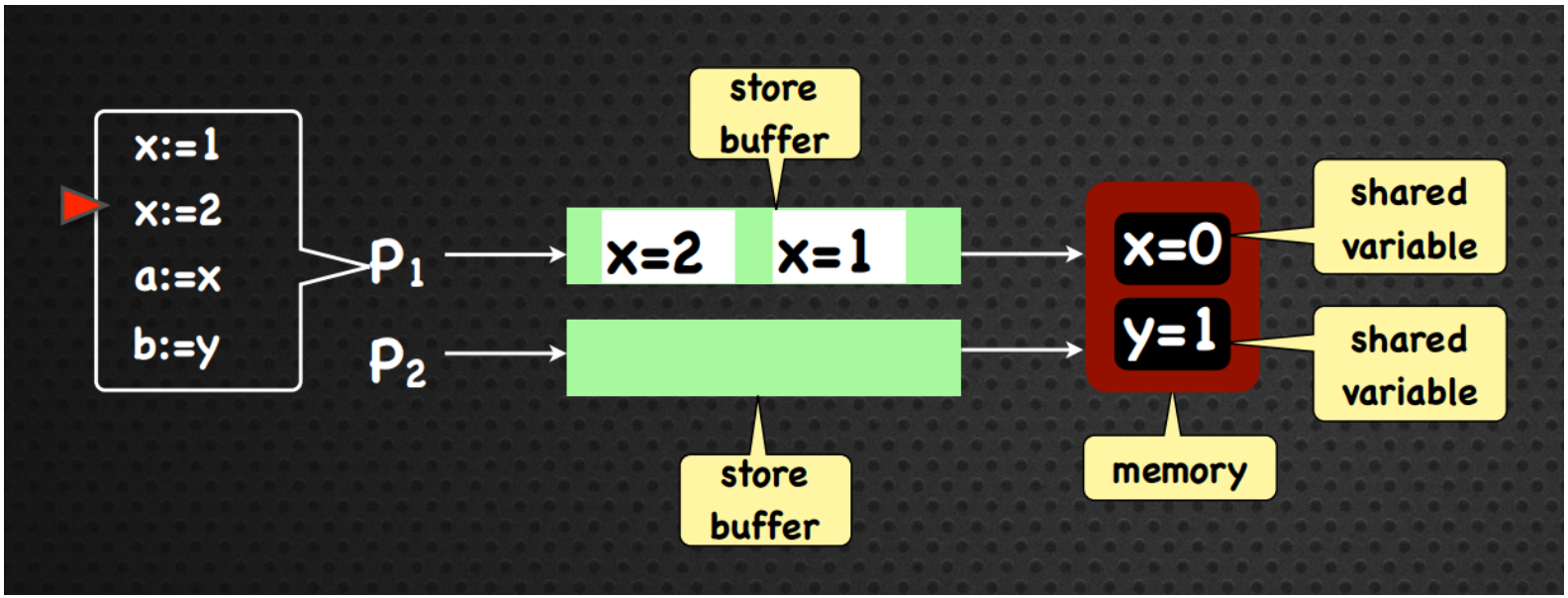
An example: total store order (TSO)

- non-atomic writes
- read locally or from memory



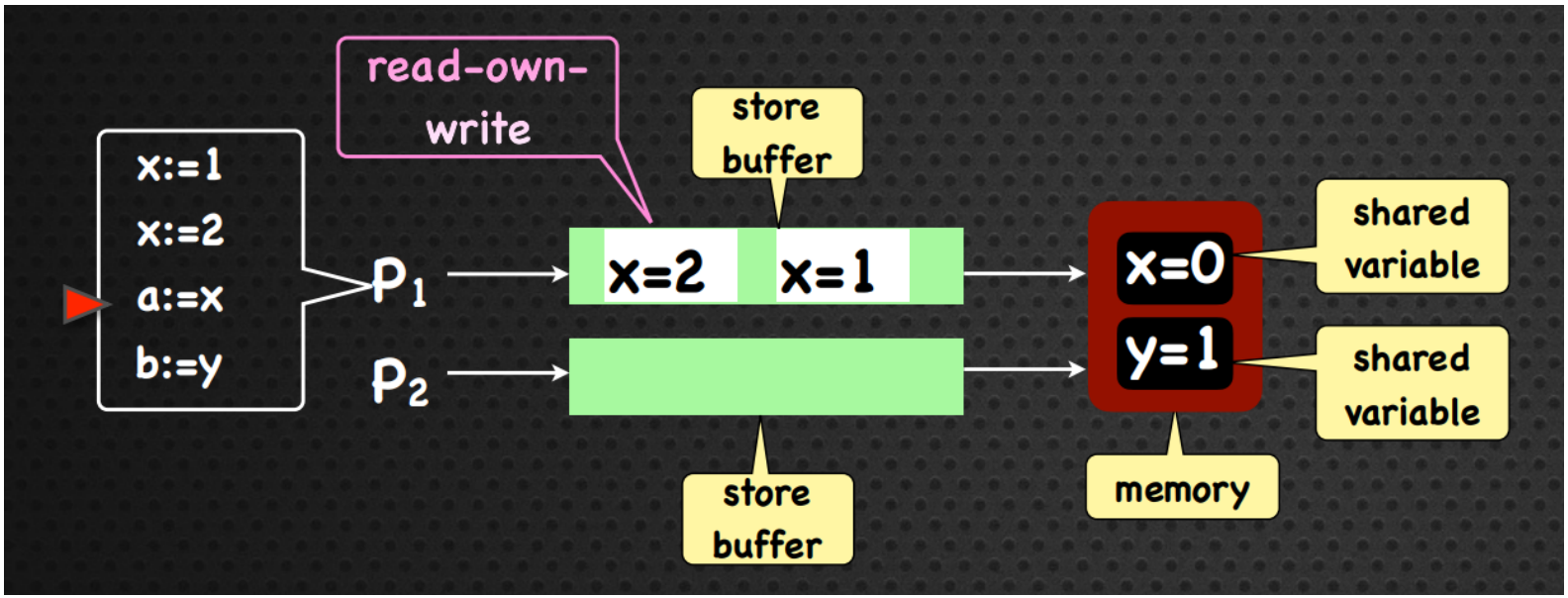
An example: total store order (TSO)

- non-atomic writes
- read locally or from memory



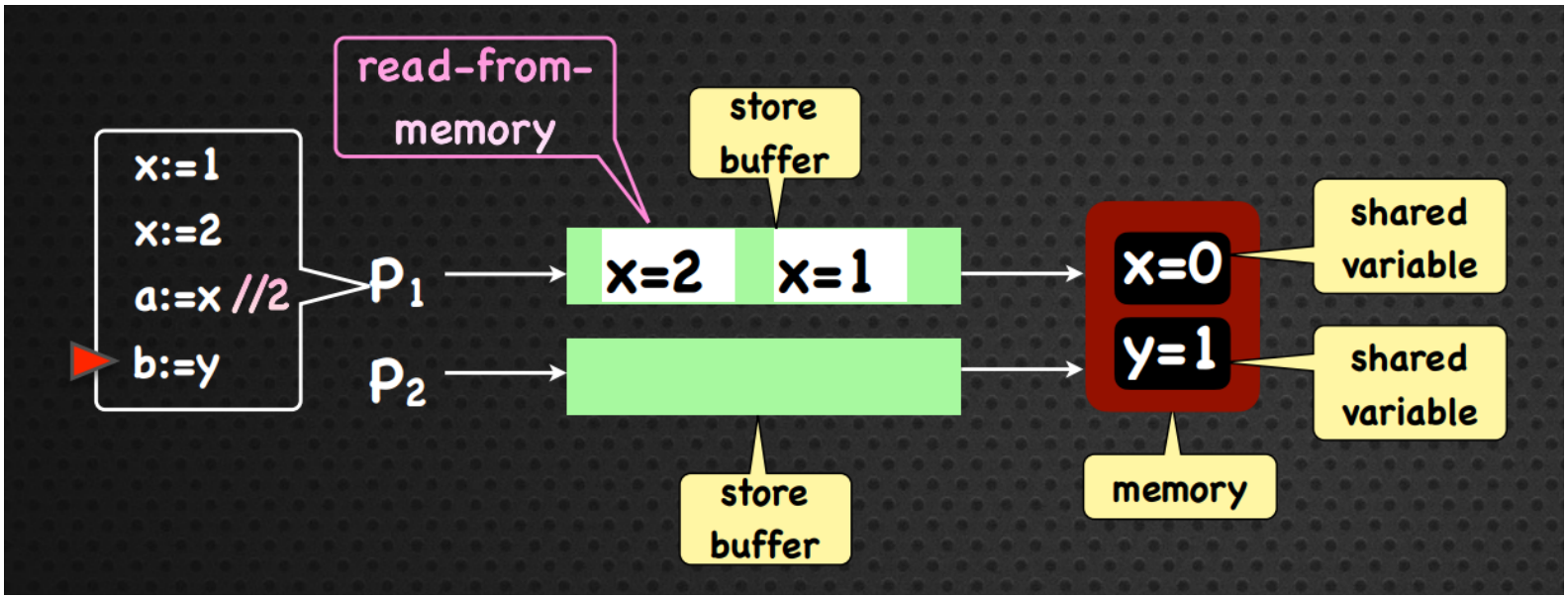
An example: total store order (TSO)

- non-atomic writes
- read locally or from memory

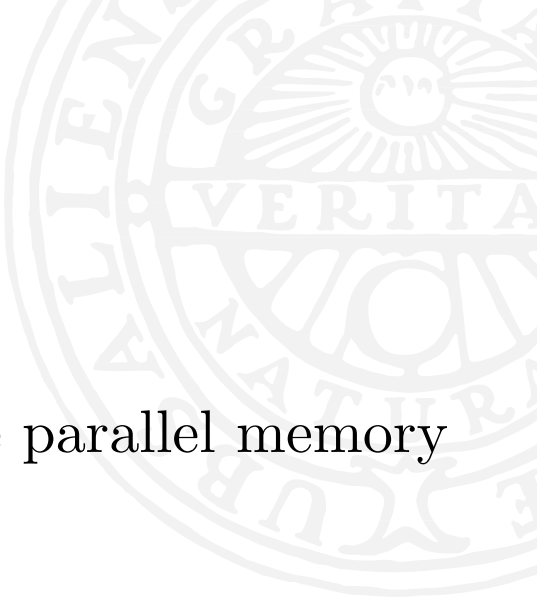


An example: total store order (TSO)

- non-atomic writes
- read locally or from memory



A *specification* of how an implementation will tackle parallel memory access is called a *memory model*.



UPPSALA
UNIVERSITET

A *specification* of how an implementation will tackle parallel memory access is called a *memory model*.

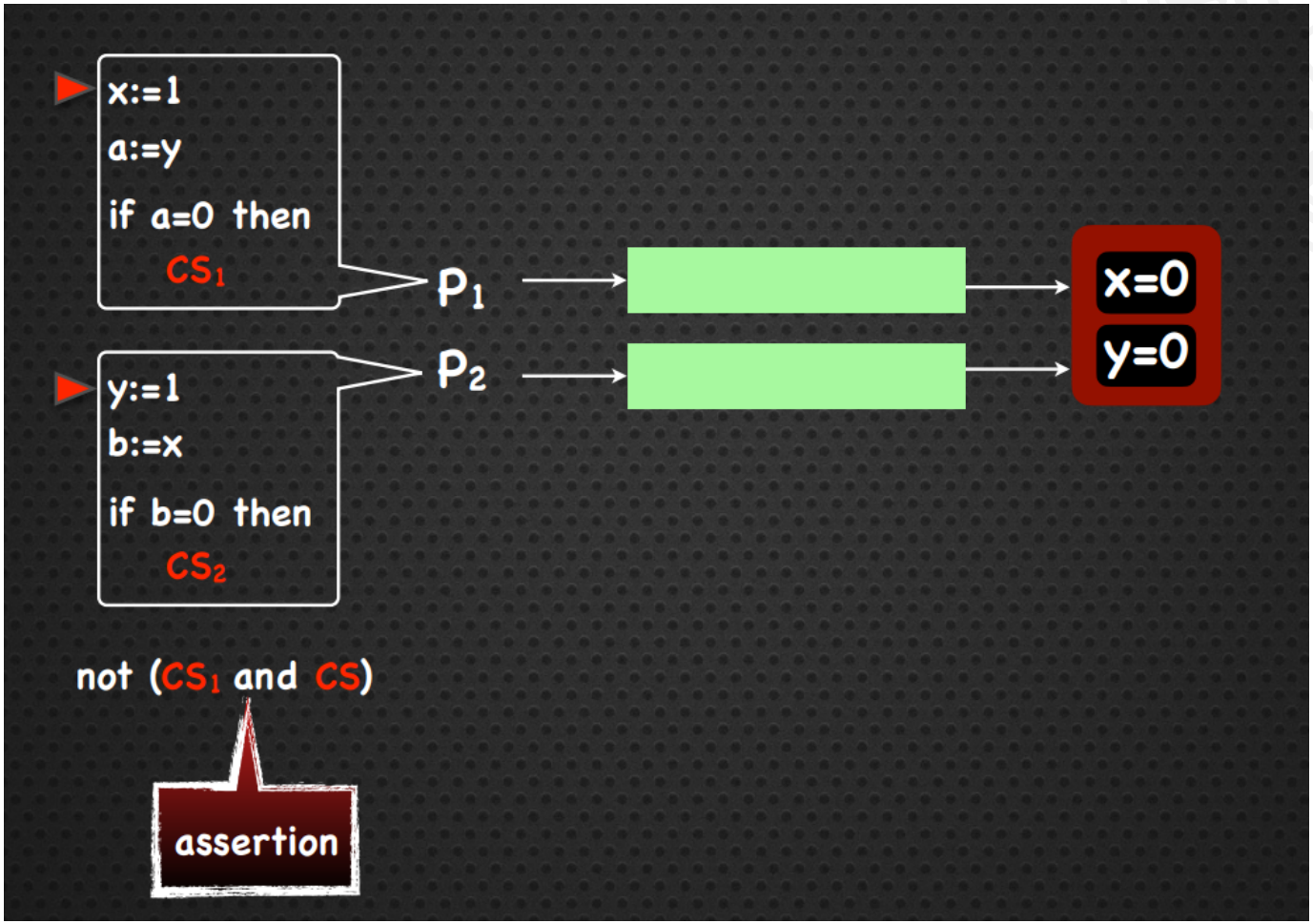
A **weak** memory model is one that allows extra behaviors.

A *specification* of how an implementation will tackle parallel memory access is called a *memory model*.

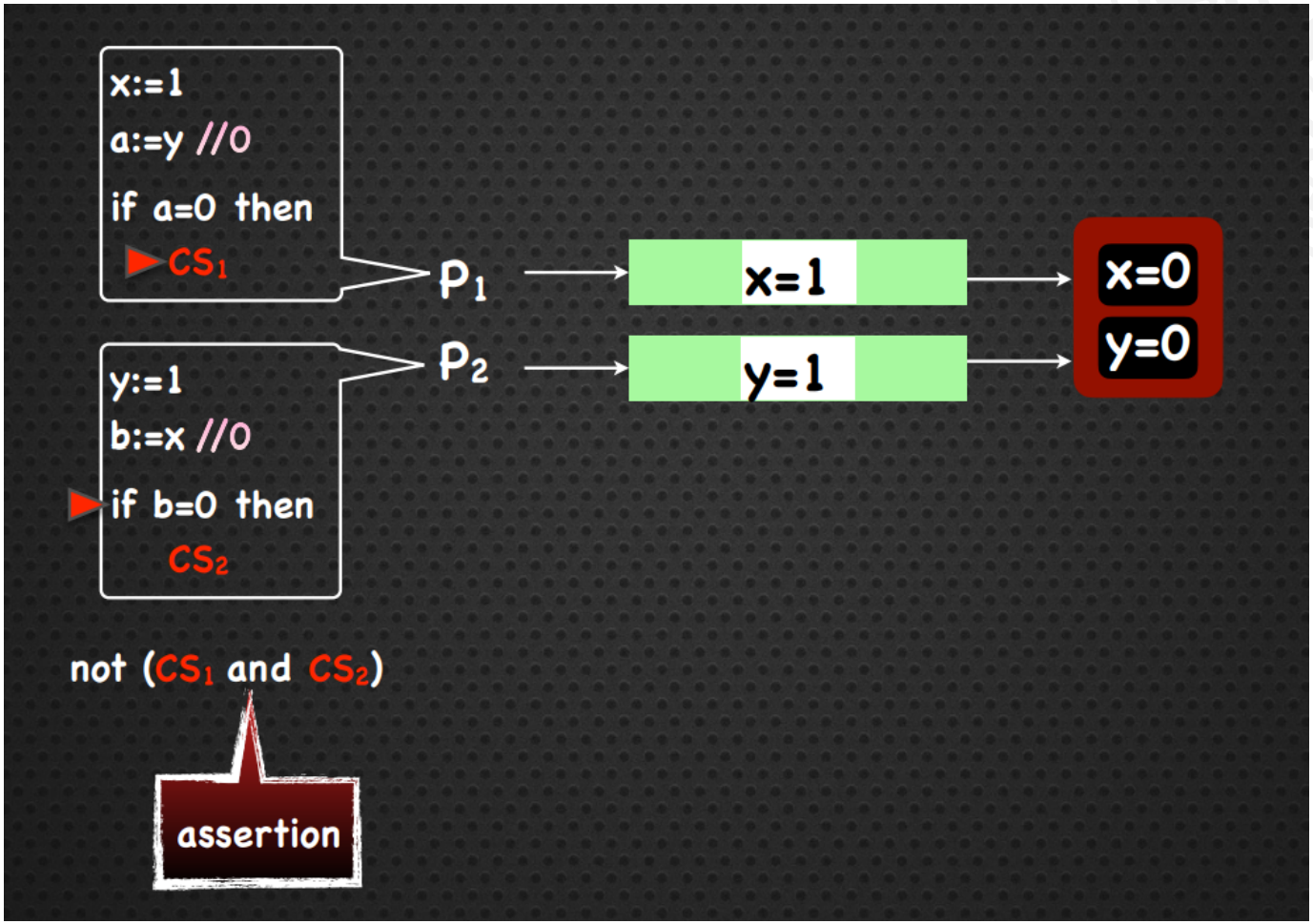
A **weak** memory model is one that allows extra behaviors.

potentially bad behaviors

Dekker under TSO

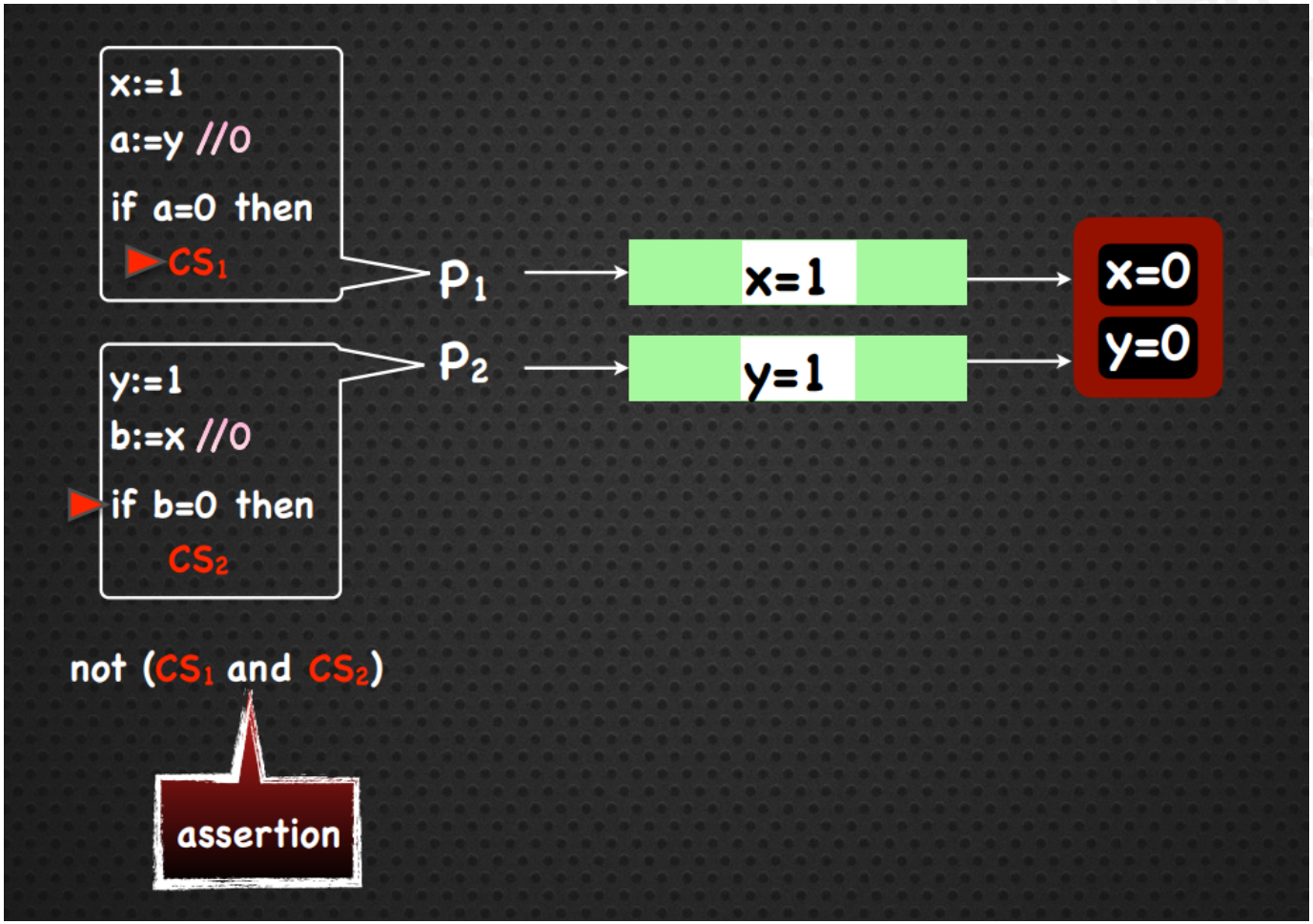


Dekker under TSO



UPPSALA UNIVERSITET

Dekker under TSO



UPPSALA UNIVERSITET

The dichotomy

Sequential consistency

- intuitive, well-researched ☺
- slow, unrealistic ☹

Weak memory

- **not** intuitive ☹
- fast, realistic ☺

UPPSALA UNIVERSITET

The dichotomy

Sequential consistency

- intuitive, well-researched ☺
- slow, unrealistic ☹

Weak memory

- **not** intuitive ☹
- fast, realistic ☺

What about verification?

UPPSALA UNIVERSITET

The dichotomy

Sequential consistency

- intuitive, well-researched ☺
- slow, unrealistic ☹

Weak memory

- **not** intuitive ☹
- fast, realistic ☺

What about verification?

Road-map: Semantics → Complexity → Techniques

UPPSALA UNIVERSITET

Common scenarios

- New software: only works when the architecture below satisfies **at least** a specific weak memory model.

  We need: algorithm for detecting "illegal" memory accesses.

Common scenarios

- New software: only works when the architecture below satisfies **at least** a specific weak memory model.

  We need: algorithm for detecting "illegal" memory accesses.

- New architecture: claims it satisfies some weak memory model.

  We need: guarantee all runs of the new architecture are safe.

Scenario 1

Potentially we don't know the architecture.

UPPSALA
UNIVERSITET

Scenario 1

Potentially we don't know the architecture.

We look at the memory accesses of different processes:

Scenario 1

Potentially we don't know the architecture.

We look at the memory accesses of different processes:

- P_1 : write(x,0), write(y,1), read(x,1)
  P_2 : read(y,1), write(x,1), write(x,0)

UPPSALA UNIVERSITET

Scenario 1

Potentially we don't know the architecture.

We look at the memory accesses of different processes:

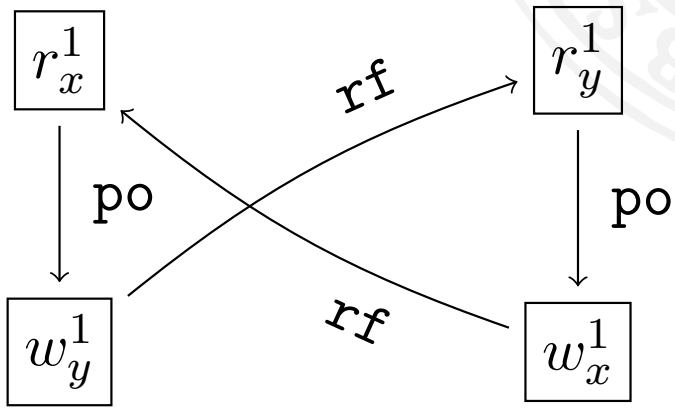
- P_1 : write(x,0), write(y,1), read(x,1)
  P_2 : read(y,1), write(x,1), write(x,0)

- P_1 : write(x,0), write(x,1), write(y,1)
  P_2 : read(y,1), read(x,1), read(x,0)

UPPSALA UNIVERSITET

Some verification primitives: Shasha-Snir traces

P_1 : read(x,1), write(y,1)
P_2 : read(y,1), write(x,1)



Example use: for SC the execution graph must be acyclic.

Sad result: Given the traces, to solve this is NP-complete

Some verification primitives: reachability

Execution graphs are in away the composition/product of a program with the semantics of a memory model.

Some verification primitives: reachability

Execution graphs are in away the composition/product of a program with the semantics of a memory model.

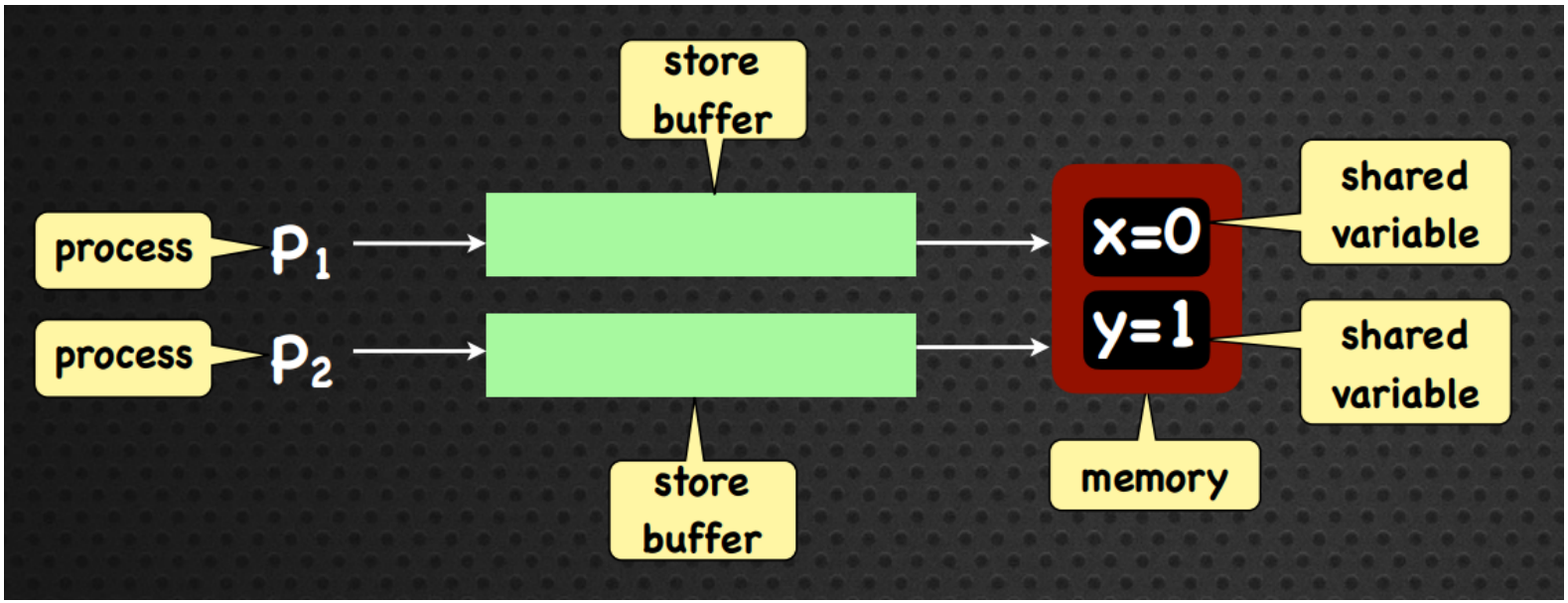
Why would reachability be hard?

Some verification primitives: reachability

Execution graphs are in away the composition/product of a program with the semantics of a memory model.

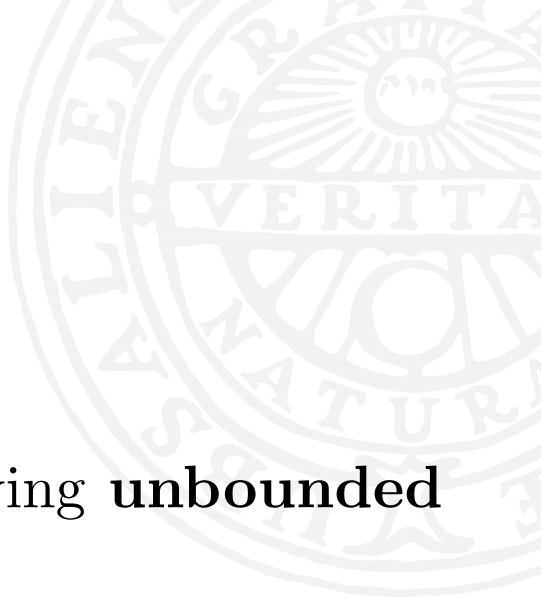
Why would reachability be hard?

Reminder:



UPPSALA UNIVERSITET

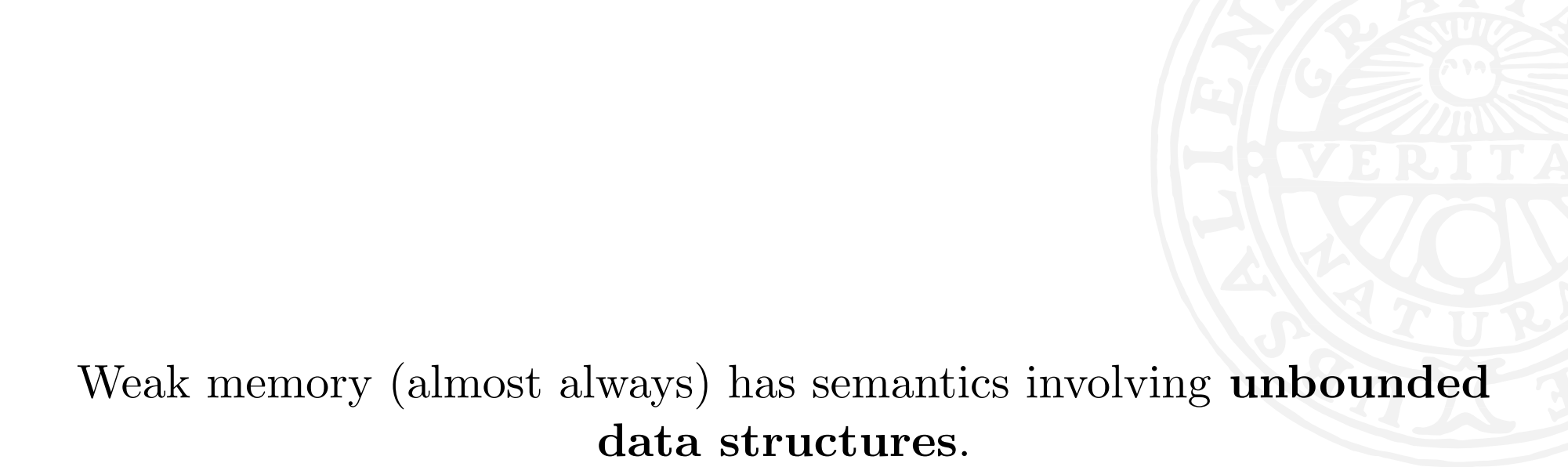
Weak memory (almost always) has semantics involving **unbounded data structures**.



UPPSALA
UNIVERSITET

Weak memory (almost always) has semantics involving **unbounded data structures**.

..sad realization



Some cool things

- Bad patterns
- Well quasi-orderings & monotonicity
- Well-structured systems

Some cool things

Bad patterns

Event sequences that are known to be violating a weak memory model.

When we are lucky: a finite set of bad patterns characterizes a given weak memory model.

Use: Only test these and we get answers for **any** sequence of events.

UPPSALA UNIVERSITET

Some cool things

Well quasi orderings

Orderings between states of the (unavoidable) unbounded data structure associated to the semantics of the model.

When we are lucky: ordering between configurations means monotonicity in satisfaction of memory model.

Use: help us prune the infinitely large tree of configurations that we explore for reachability.

UPPSALA UNIVERSITET

Some cool things

Well-structured systems

Semantic models for weak memory that have:

- Unbounded but FIFO components.
- Monotonicity

When we are lucky: we manage to transform the semantics of a weak memory model to a form that is well-structured.

Use: (Theorem) Reachability is always decidable.

UPPSALA UNIVERSITET

Summary:

- basic idea of how weak memory shows up
- why is it problematic
- what we (usually) do about it.

Future:

- develop good algorithms for specific memory models
- develop hardness results
- (maybe) unification results.

Thank you for your attention!

UPPSALA
UNIVERSITET

Questions?