Remote-Scope Promotion:

clarified rectified verified

John Wickerson
Imperial 🇬🇧

Mark Batty
Kent 🇬🇧

Brad Beckmann
AMD 🇺🇸

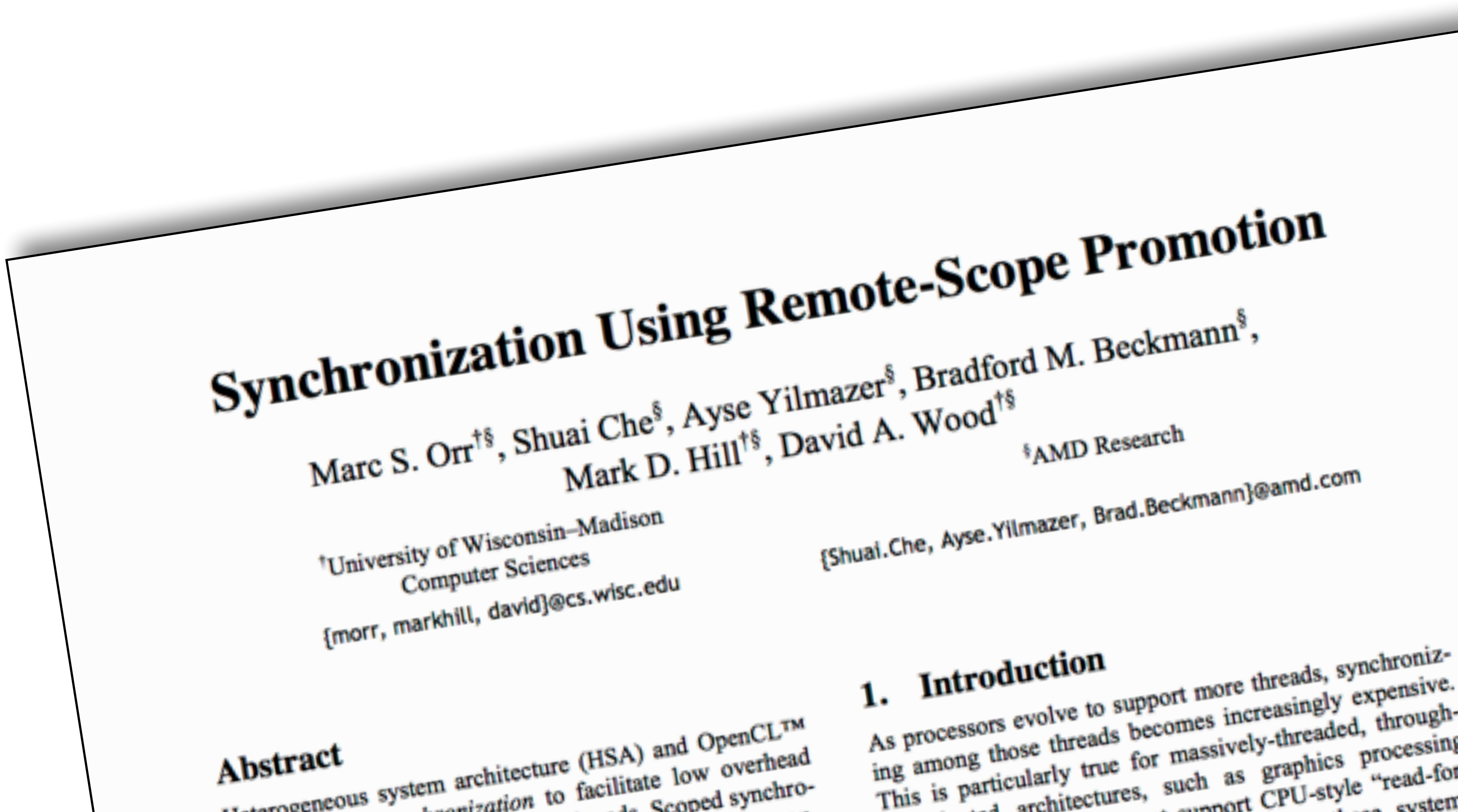
Ally Donaldson
Imperial 🇬🇧

OOPSLA

Pittsburgh, 30 October 2015

In brief

- **Remote-scope promotion** is a GPU programming extension from **AMD** for efficient **work-stealing**



Synchronization Using Remote-Scope Promotion

Marc S. Orr[†§], Shuai Che[§], Ayse Yilmazer[§], Bradford M. Beckmann[§],
Mark D. Hill[†§], David A. Wood[†§]

[†]University of Wisconsin–Madison
Computer Sciences
{morr, markhill, david}@cs.wisc.edu

[§]AMD Research
{Shuai.Che, Ayse.Yilmazer, Brad.Beckmann}@amd.com

Abstract

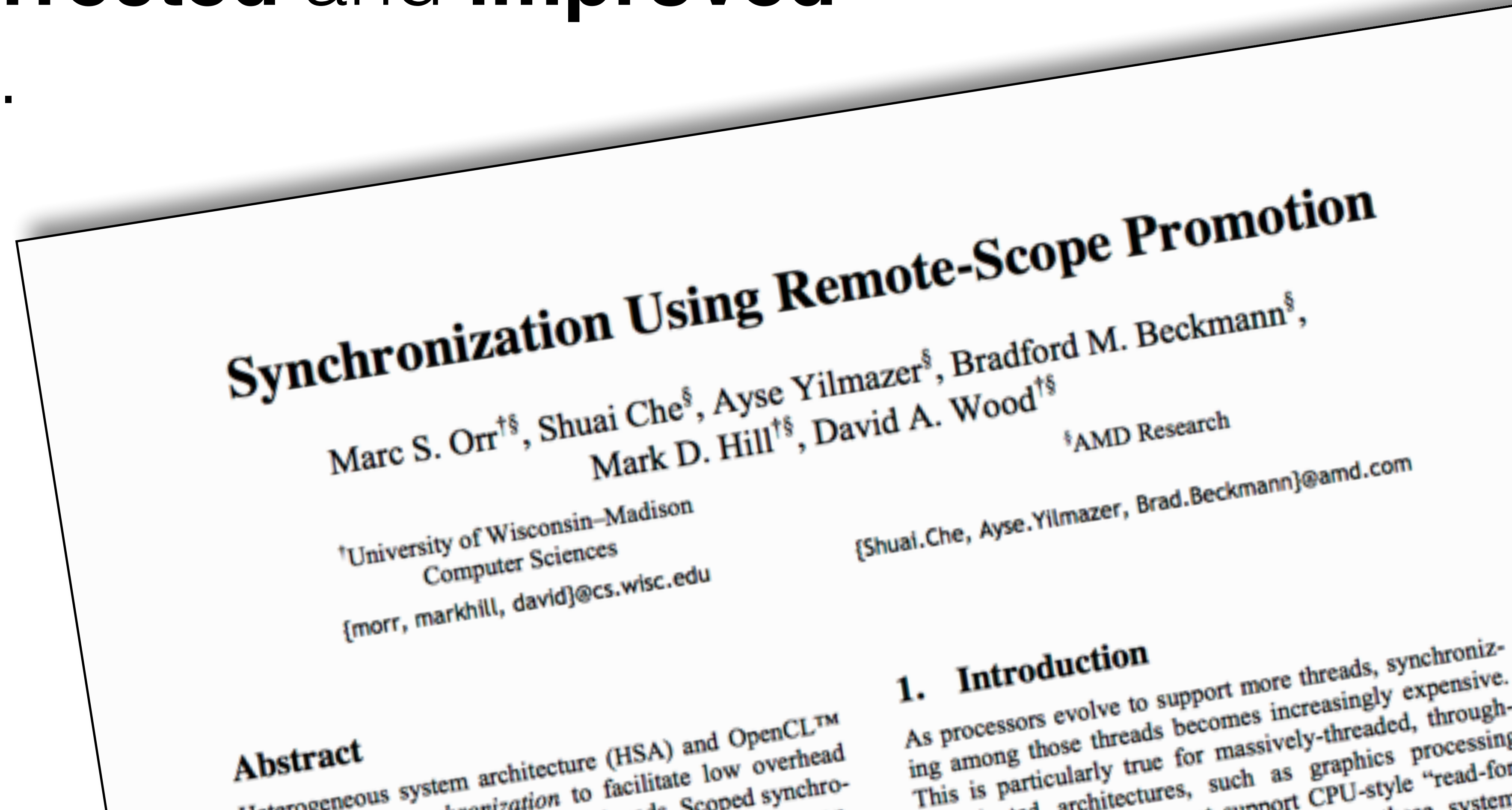
...terogeneous system architecture (HSA) and OpenCL™
...ronization to facilitate low overhead
... Scoped synchro-

1. Introduction

As processors evolve to support more threads, synchroniz-
ing among those threads becomes increasingly expensive.
This is particularly true for massively-threaded, through-
...ted architectures, such as graphics processing
... support CPU-style "read-for
... system

In brief

- **Remote-scope promotion** is a GPU programming extension from **AMD** for efficient **work-stealing**
- We **formalised** the design (at SW and HW level). This led to a **corrected** and **improved** implementation.



Synchronization Using Remote-Scope Promotion

Marc S. Orr[†§], Shuai Che[§], Ayse Yilmazer[§], Bradford M. Beckmann[§], Mark D. Hill[†§], David A. Wood[†§]

[†]University of Wisconsin–Madison
Computer Sciences
{morr, markhill, david}@cs.wisc.edu

[§]AMD Research
{Shuai.Che, Ayse.Yilmazer, Brad.Beckmann}@amd.com

Abstract

Heterogeneous system architecture (HSA) and OpenCL™

synchronization to facilitate low overhead

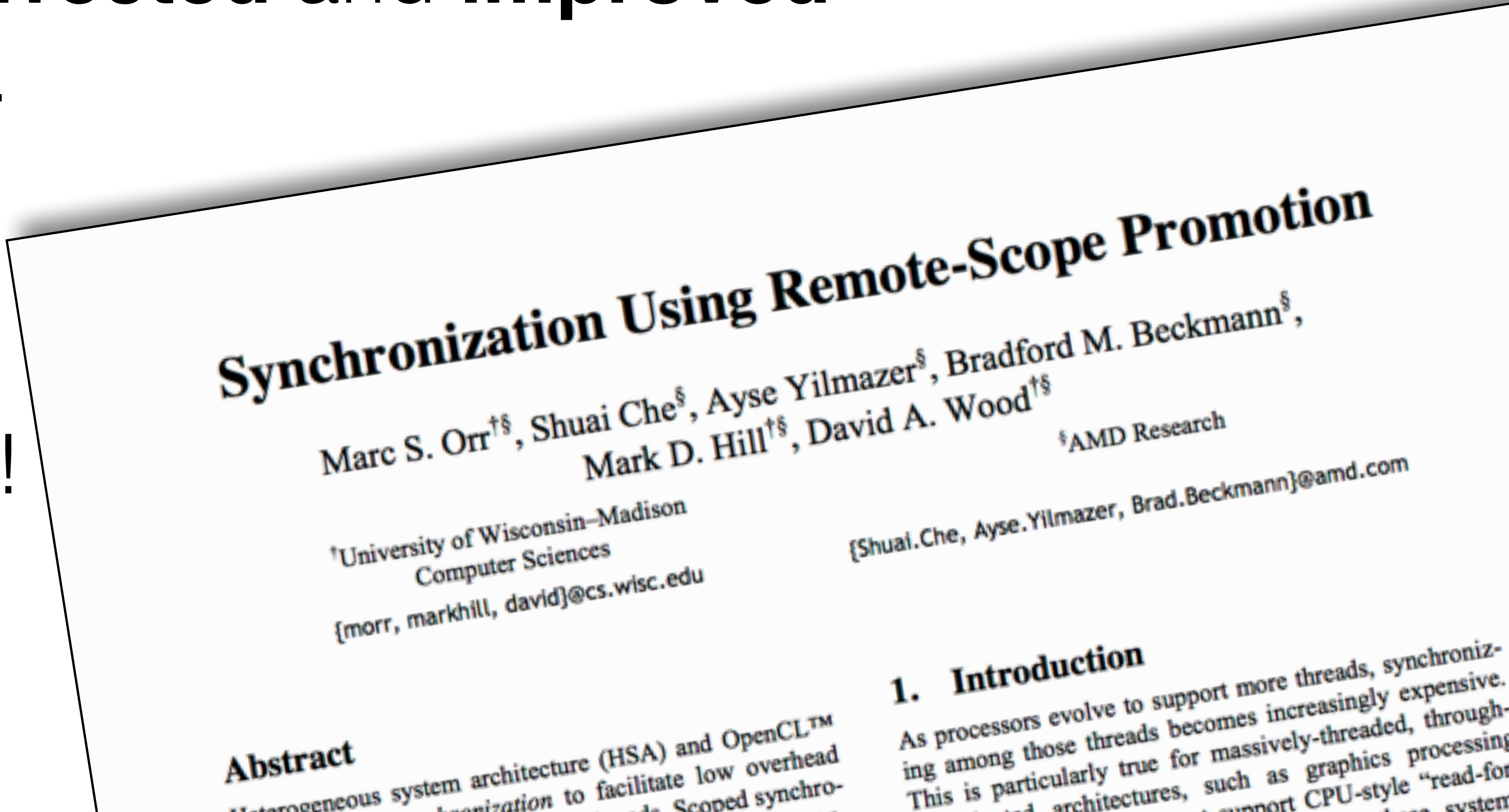
Scoped synchro-

1. Introduction

As processors evolve to support more threads, synchroniz-
ing among those threads becomes increasingly expensive.
This is particularly true for massively-threaded, through-
architectures, such as graphics processing
support CPU-style "read-for

In brief

- **Remote-scope promotion** is a GPU programming extension from **AMD** for efficient **work-stealing**
- We **formalised** the design (at SW and HW level). This led to a **corrected** and **improved** implementation.
- Formalise **early** in the design process!



Synchronization Using Remote-Scope Promotion

Marc S. Orr[†§], Shuai Che[§], Ayse Yilmazer[§], Bradford M. Beckmann[§], Mark D. Hill[†§], David A. Wood[†§]

[†]University of Wisconsin–Madison Computer Sciences
{morr, markhill, david}@cs.wisc.edu

[§]AMD Research
{Shuai.Che, Ayse.Yilmazer, Brad.Beckmann}@amd.com

Abstract

eterogeneous system architecture (HSA) and OpenCL™
ronization to facilitate low overhead
Scoped synchro-

1. Introduction

As processors evolve to support more threads, synchroniz-
ing among those threads becomes increasingly expensive.
This is particularly true for massively-threaded, through-
architectures, such as graphics processing
support CPU-style "read-fo

This talk

1. Background: What is RSP?
2. Adding RSP to the OpenCL memory model
3. A formalised implementation of OpenCL+RSP

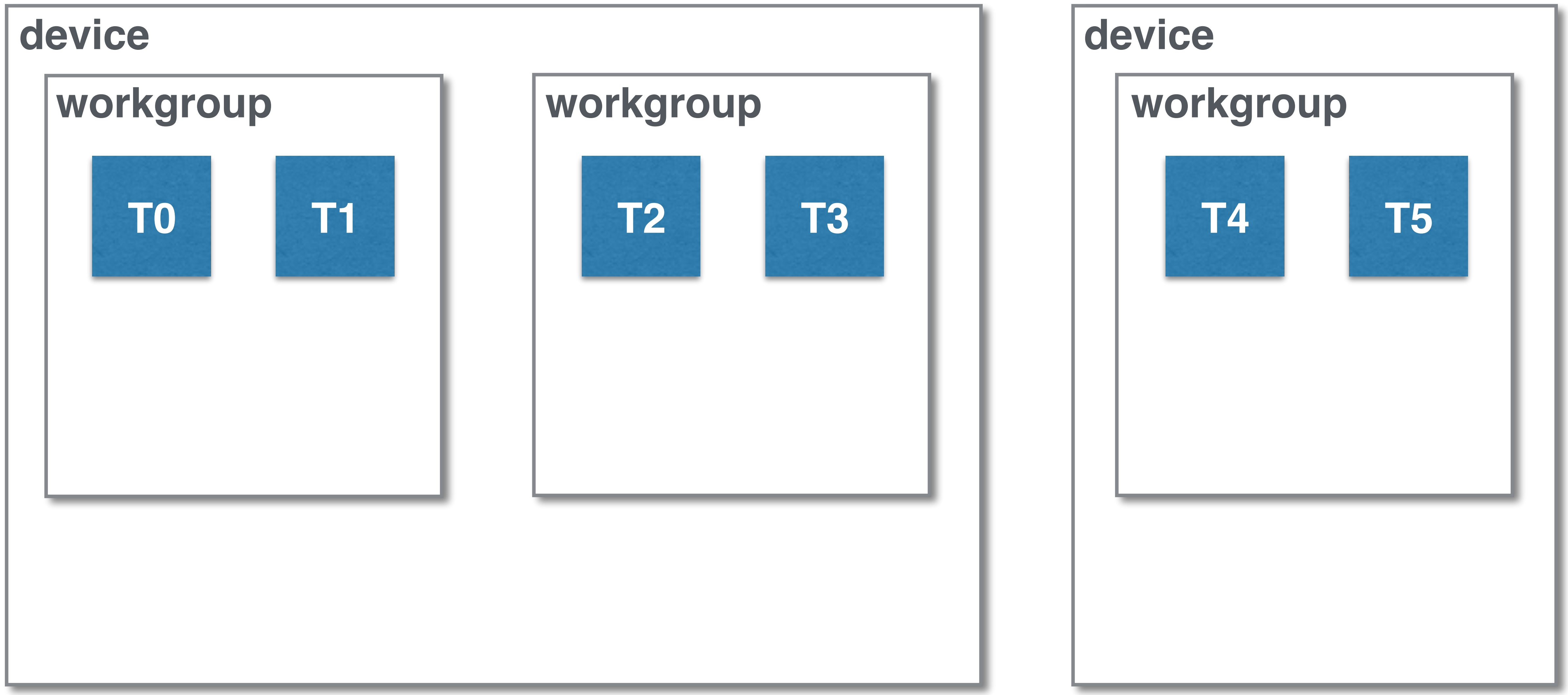
This talk

1. ▶ Background: What is RSP?
2. Adding RSP to the OpenCL memory model
3. A formalised implementation of OpenCL+RSP

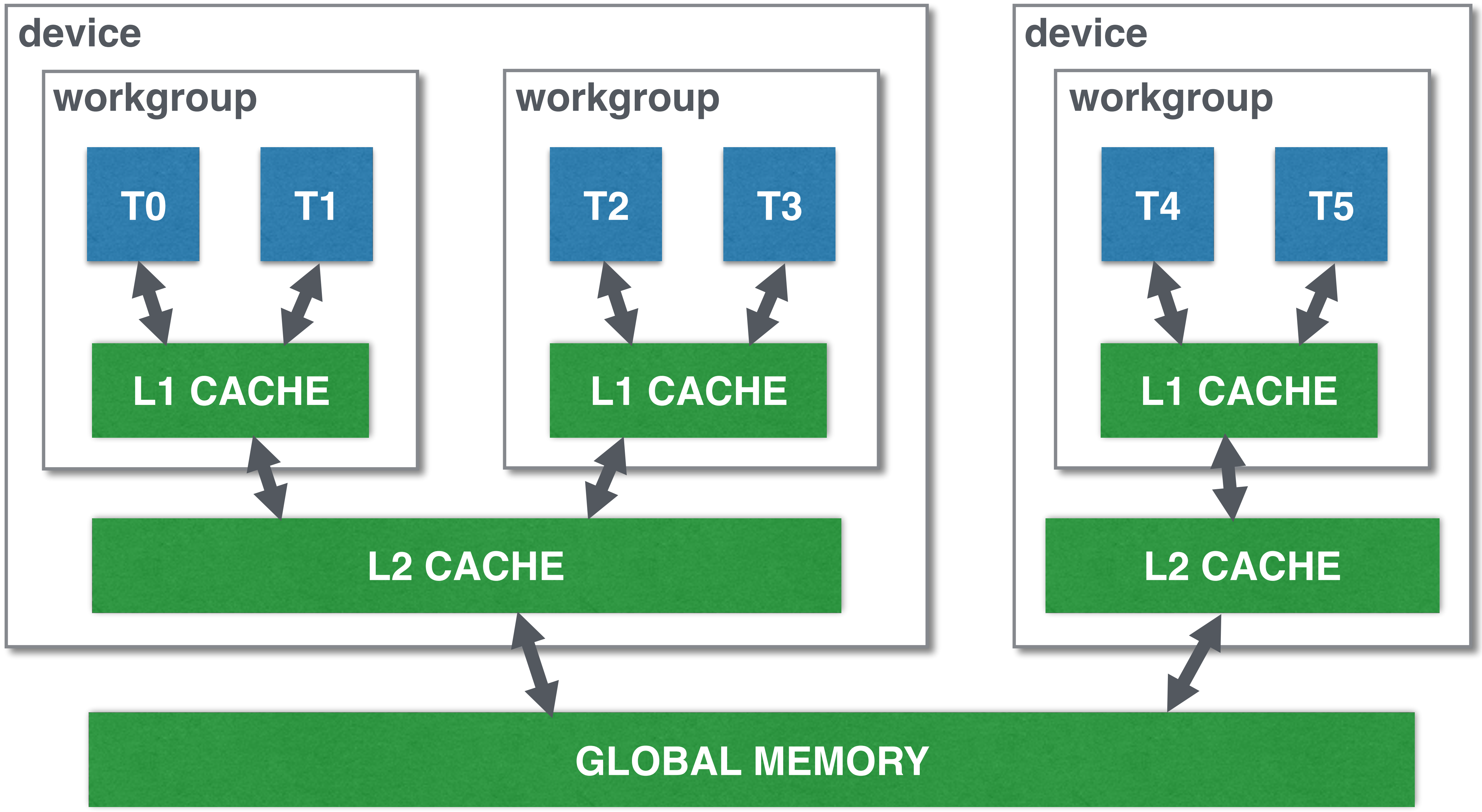
C11: flat thread structure

T0
T1
T2
T3
T4
T5

OpenCL: thread groupings

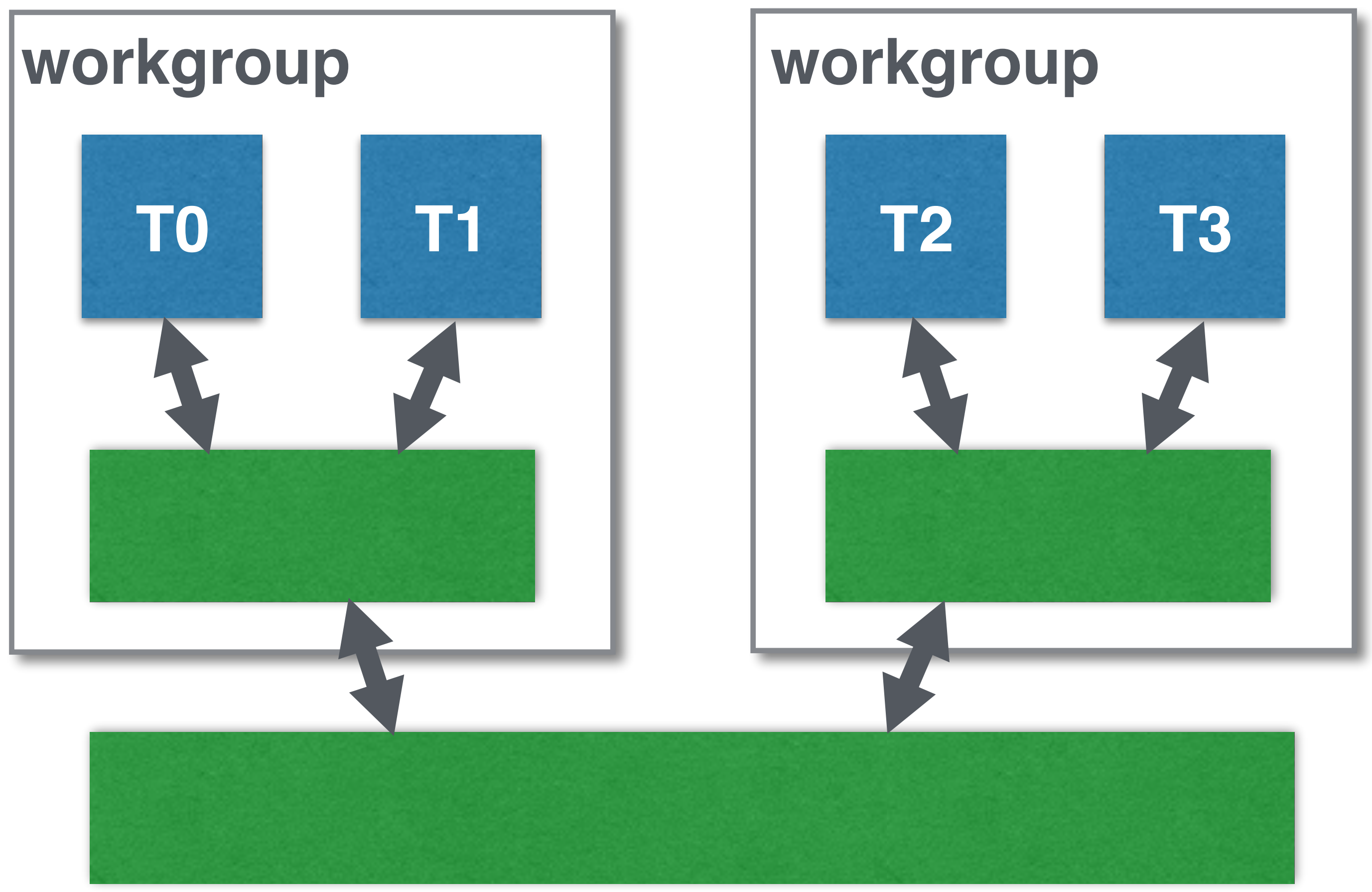


GPUs: hierarchical memory



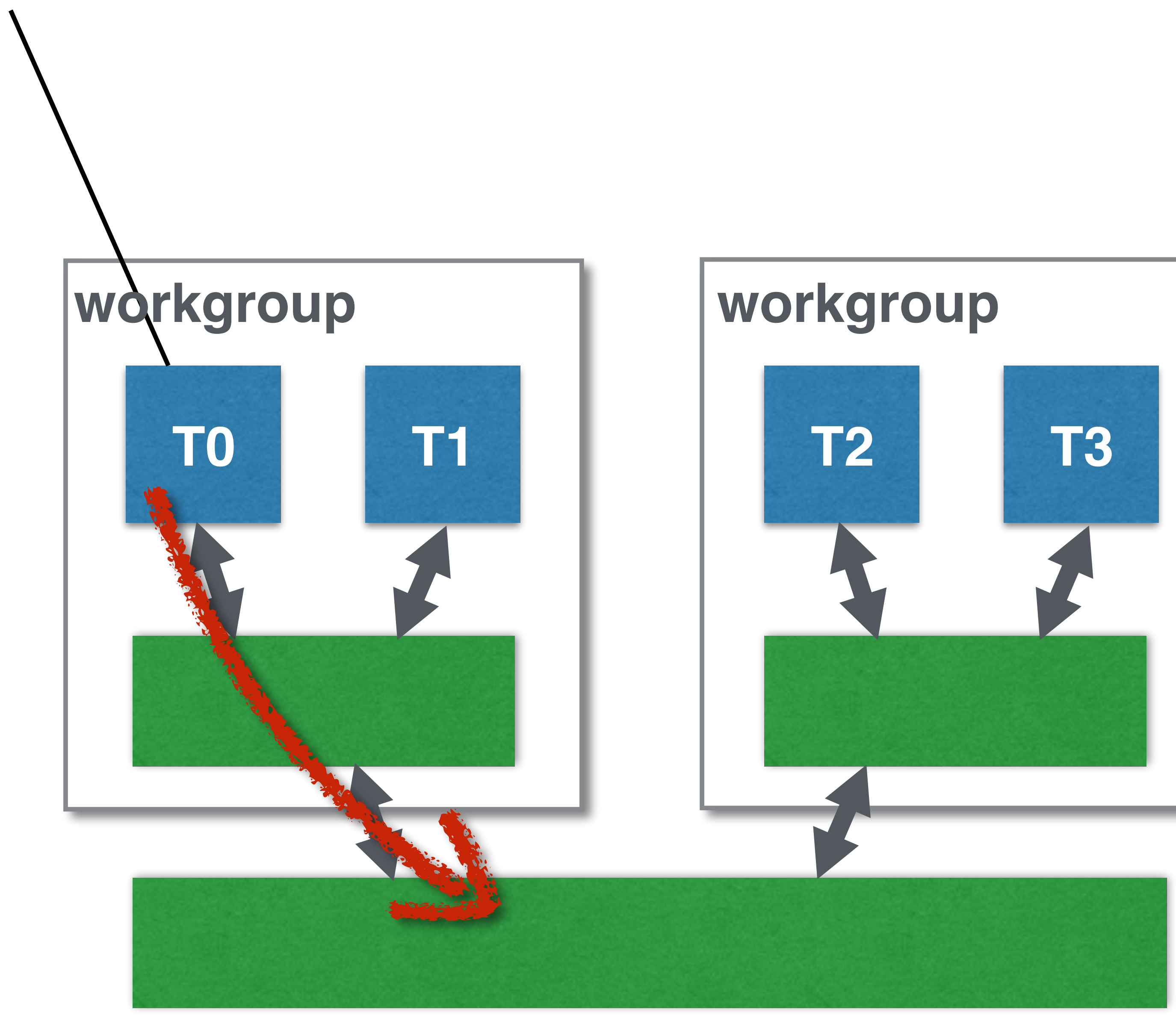
Memory scopes

Memory scopes

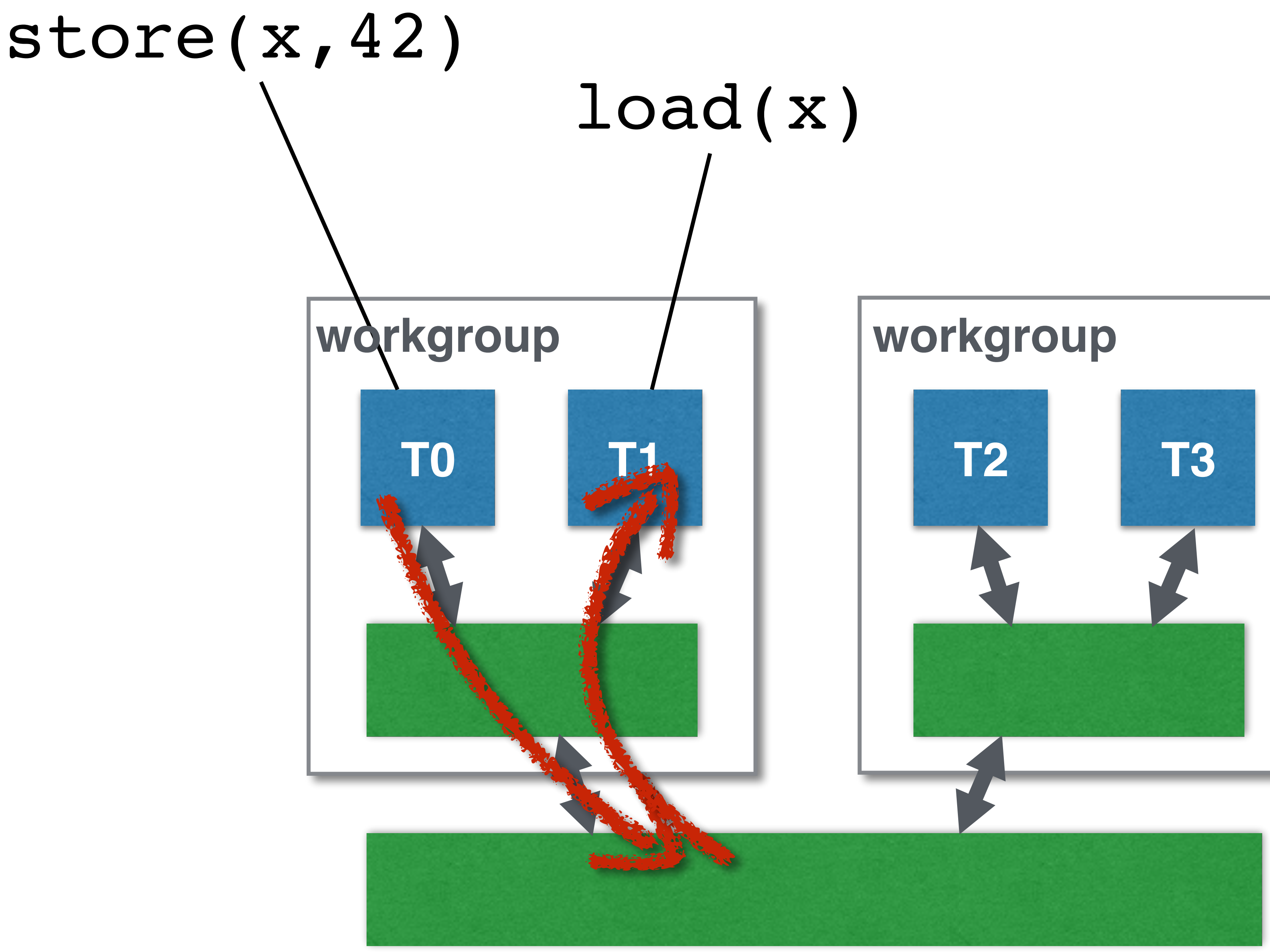


Memory scopes

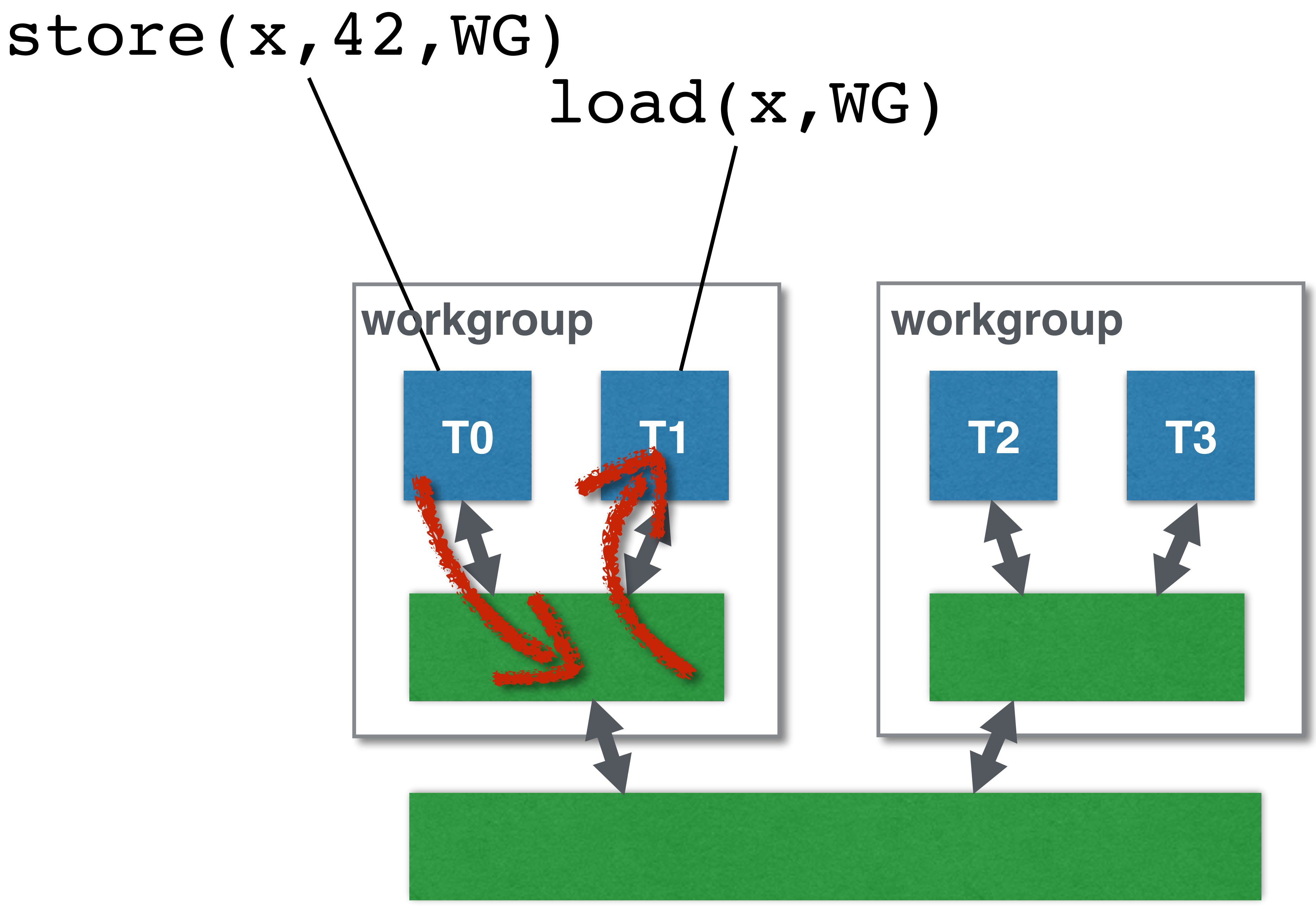
```
store(x,42)
```



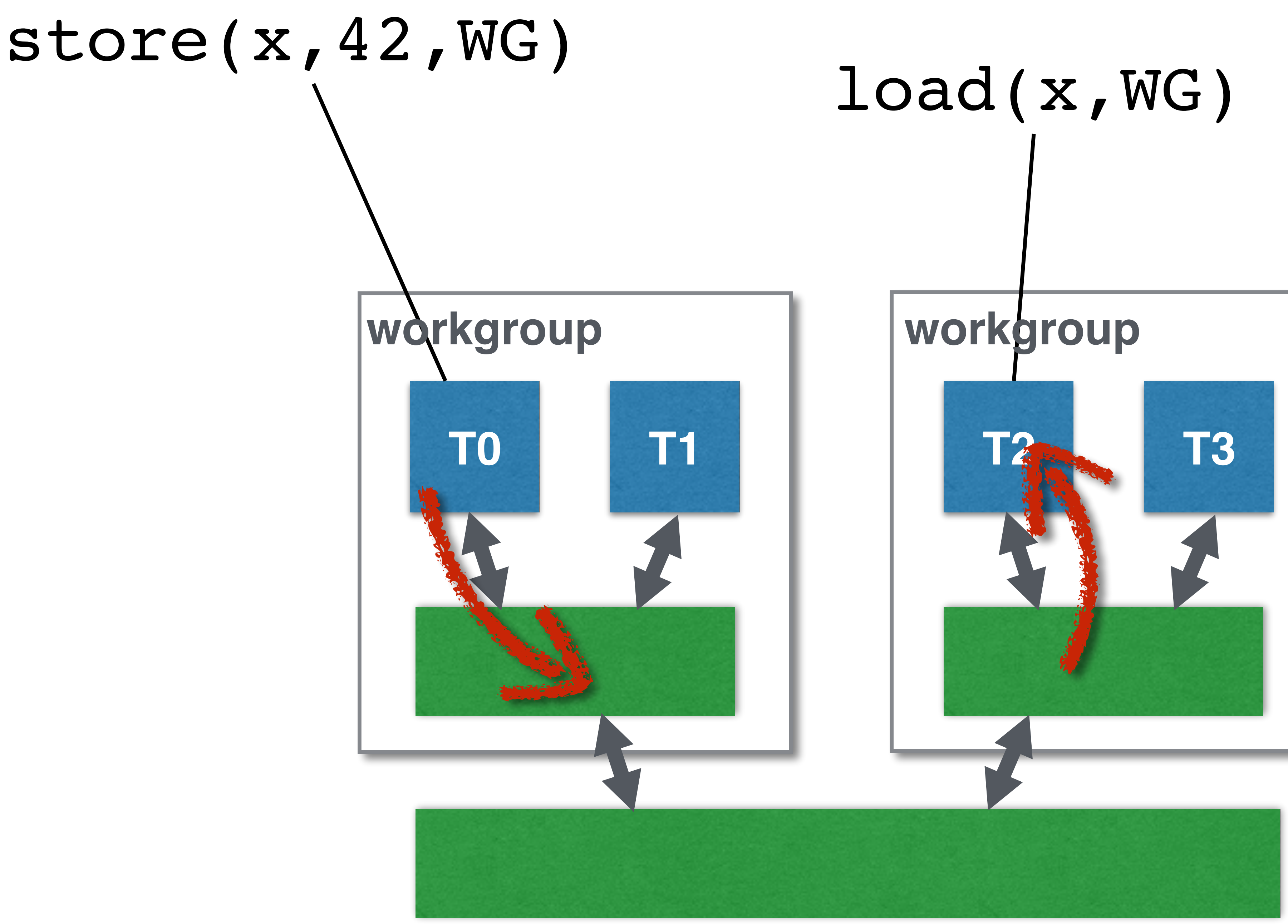
Memory scopes



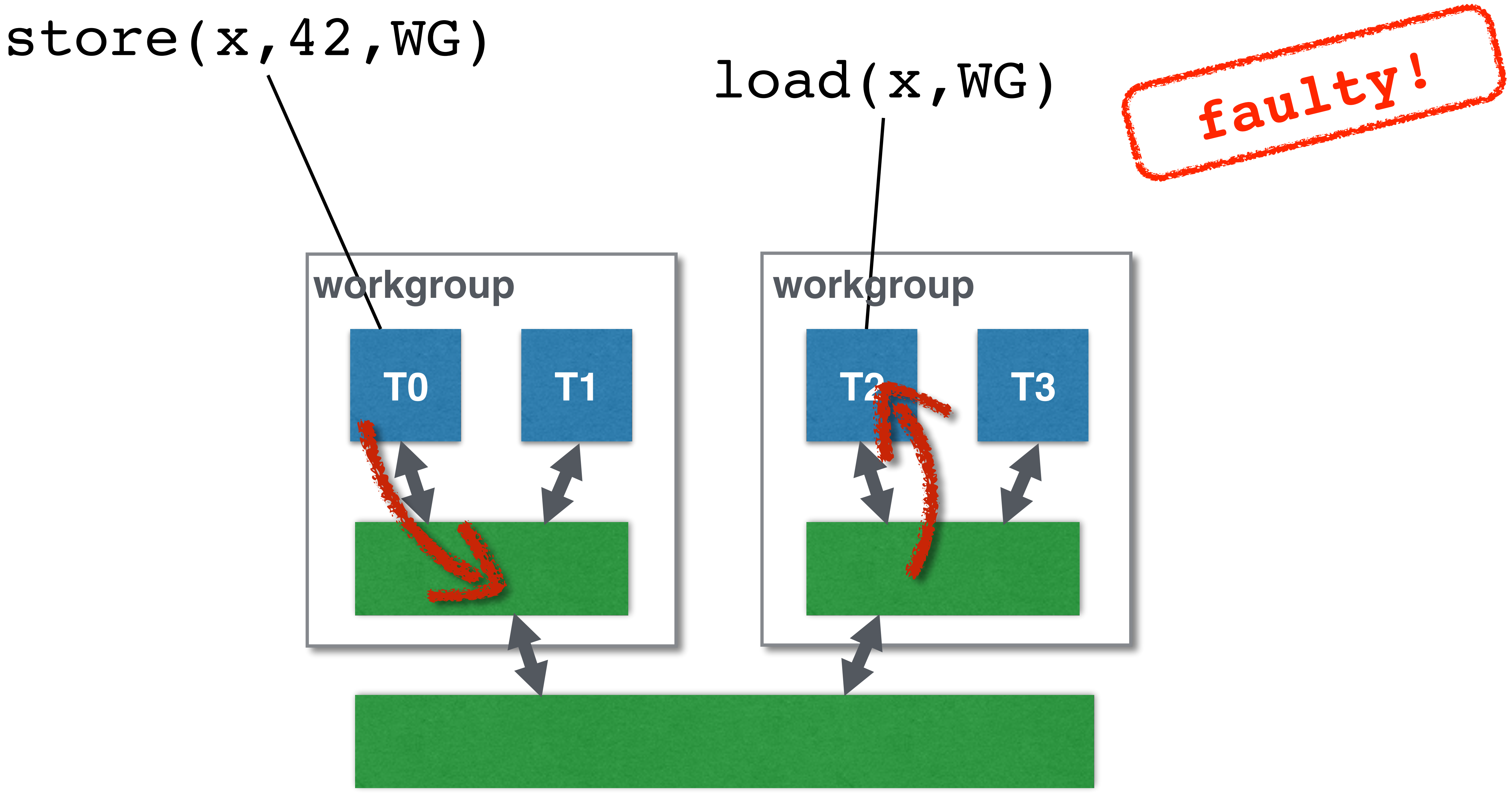
Memory scopes



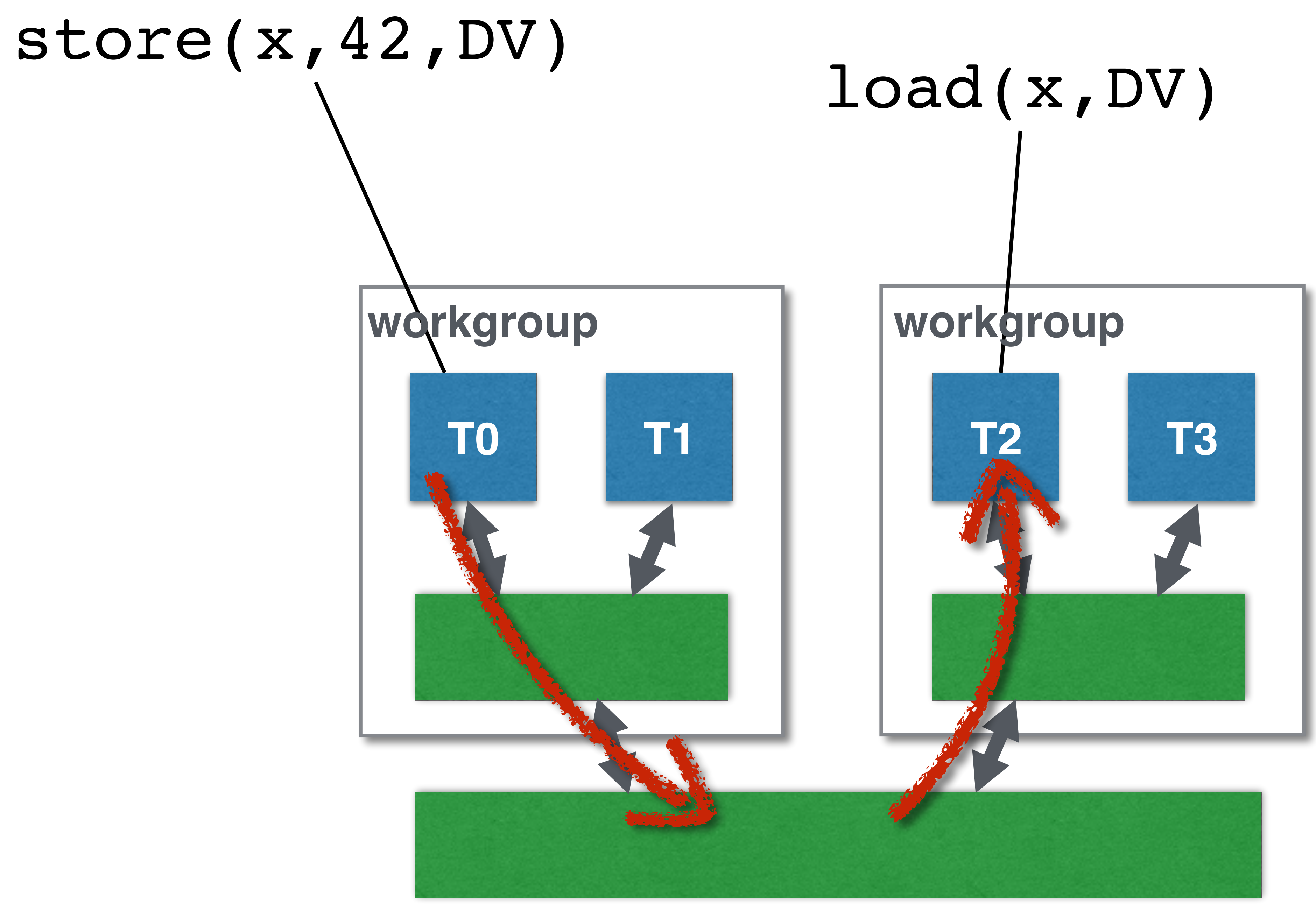
Memory scopes



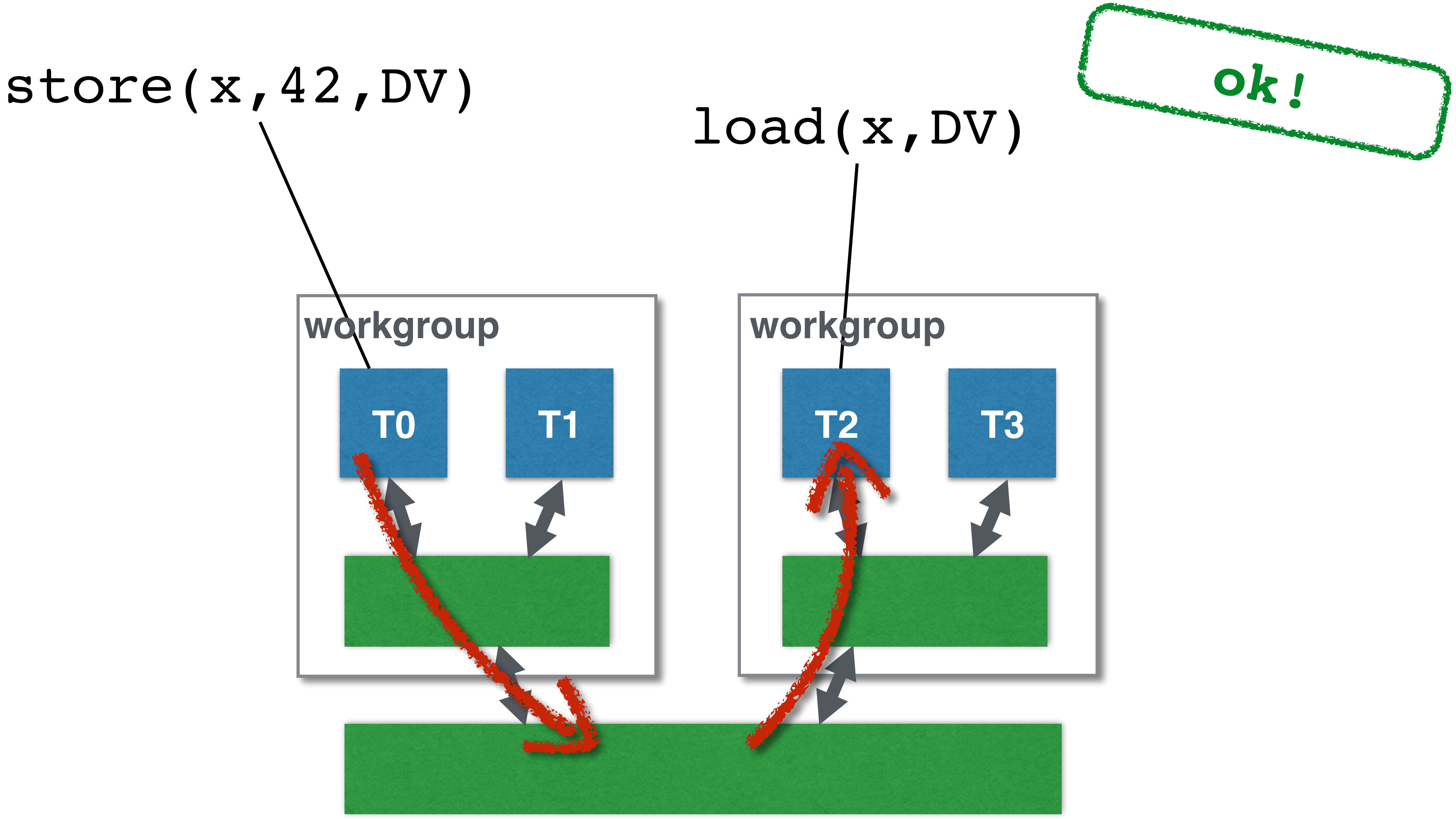
Memory scopes



Memory scopes

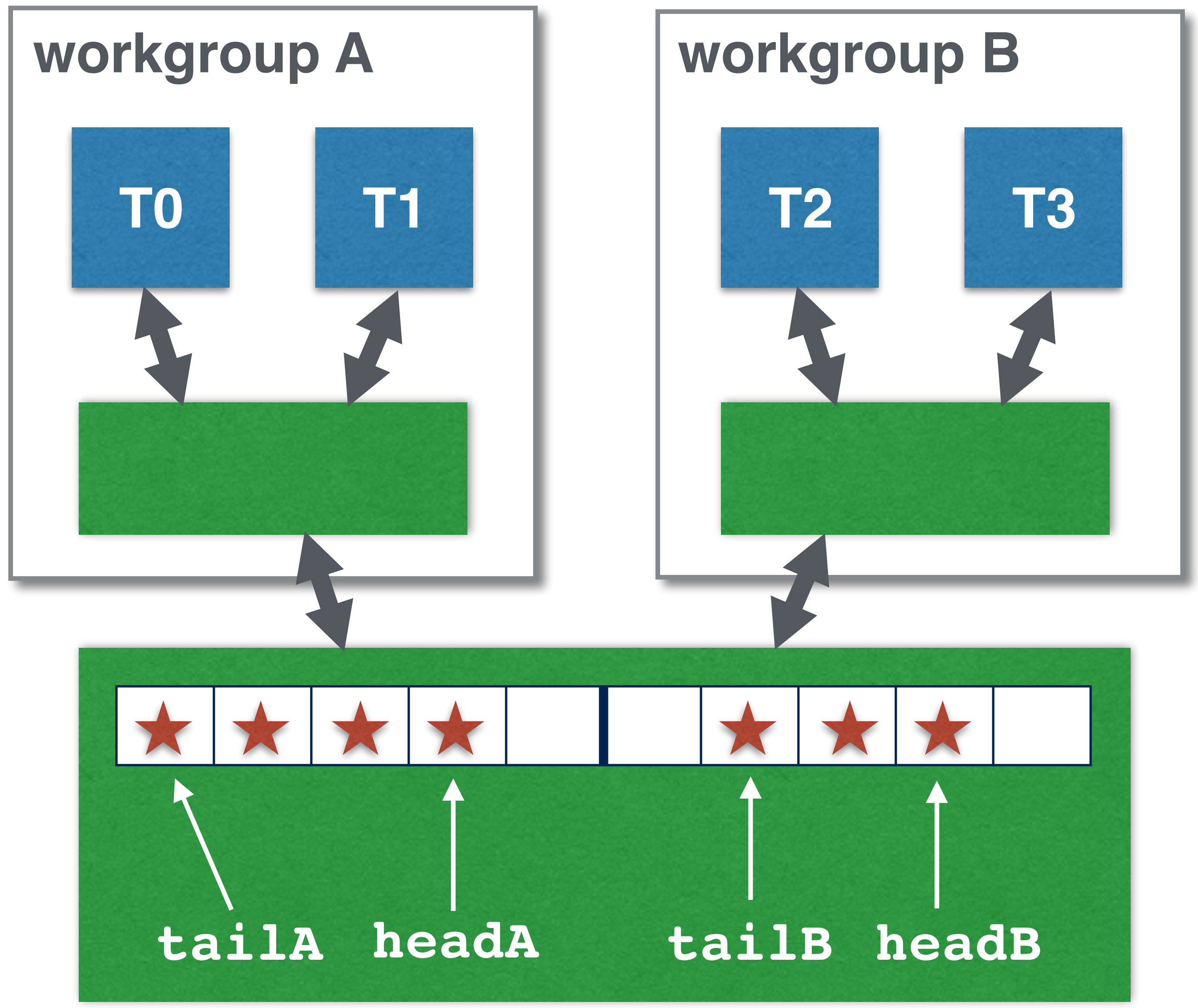


Memory scopes



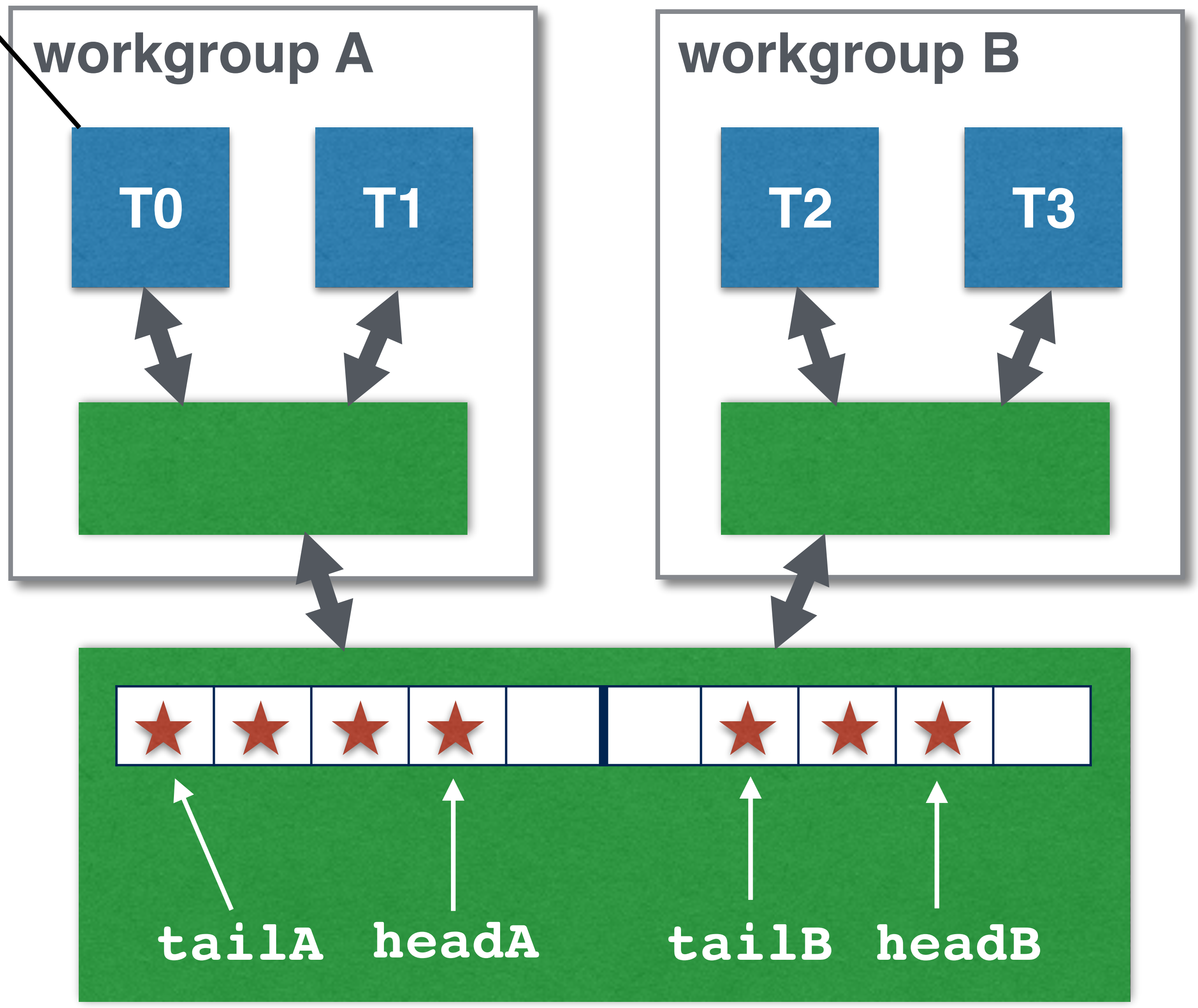
Example: work-stealing

Example: work-stealing

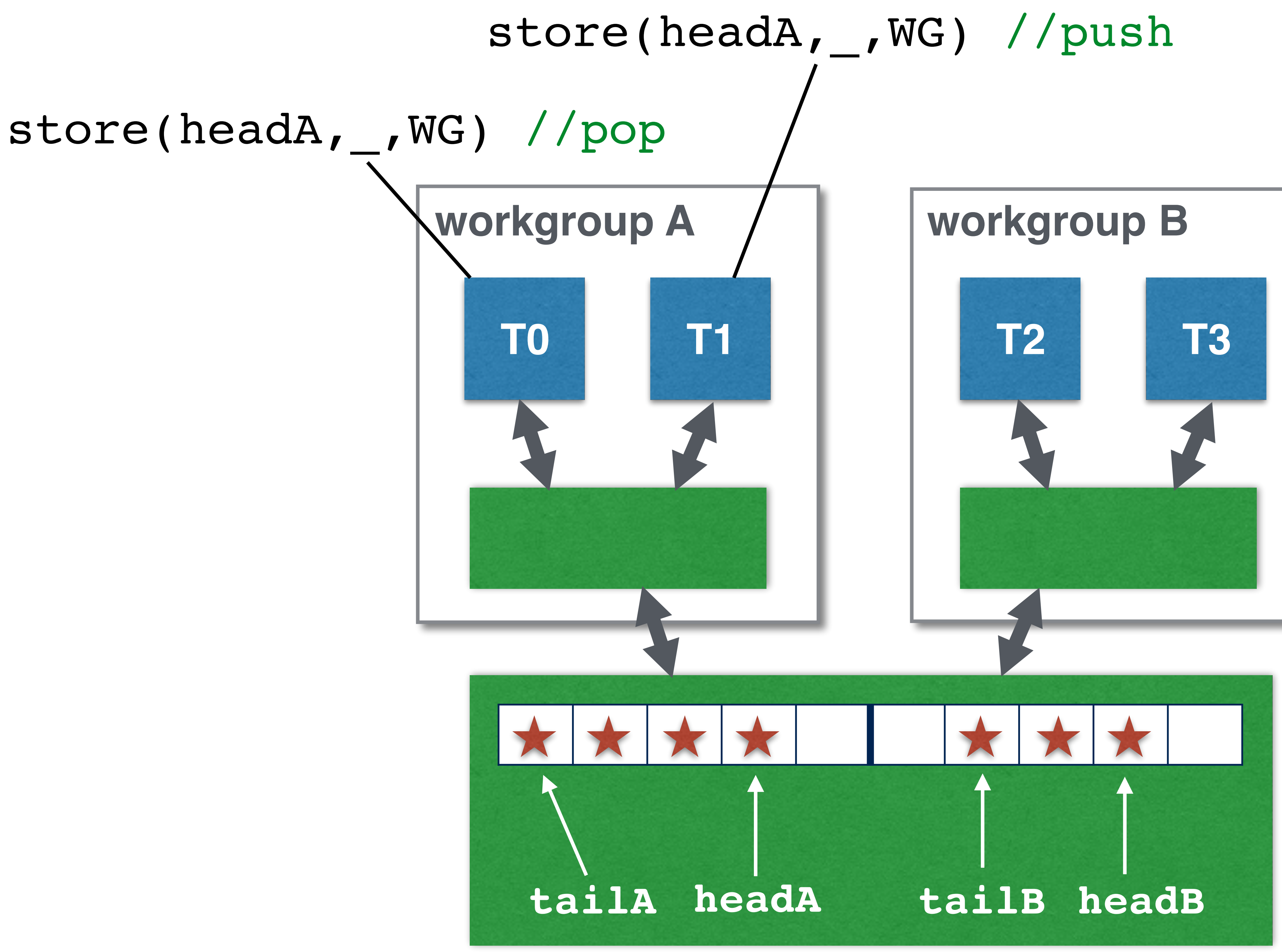


Example: work-stealing

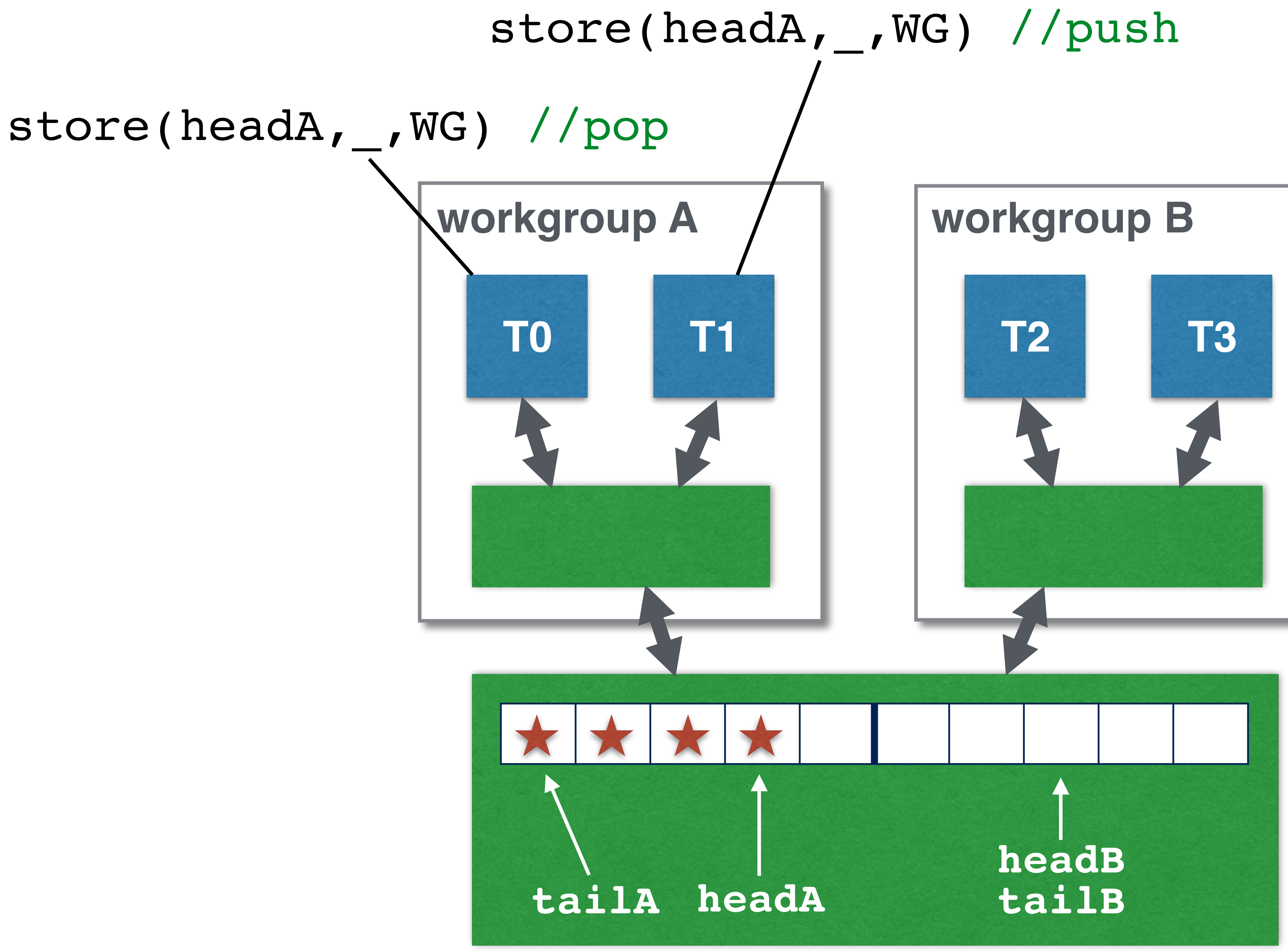
```
store(headA,_,WG) //pop
```



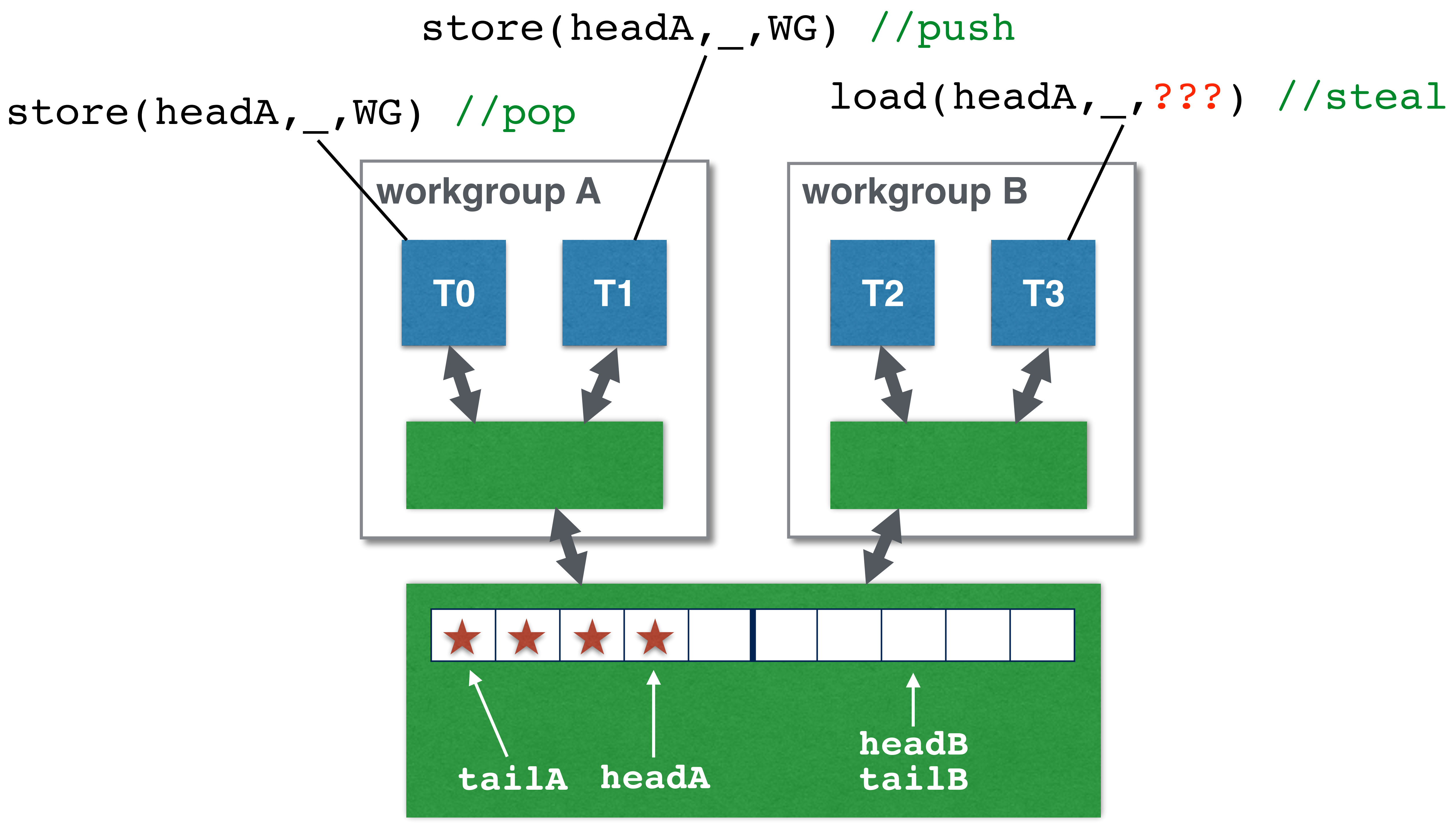
Example: work-stealing



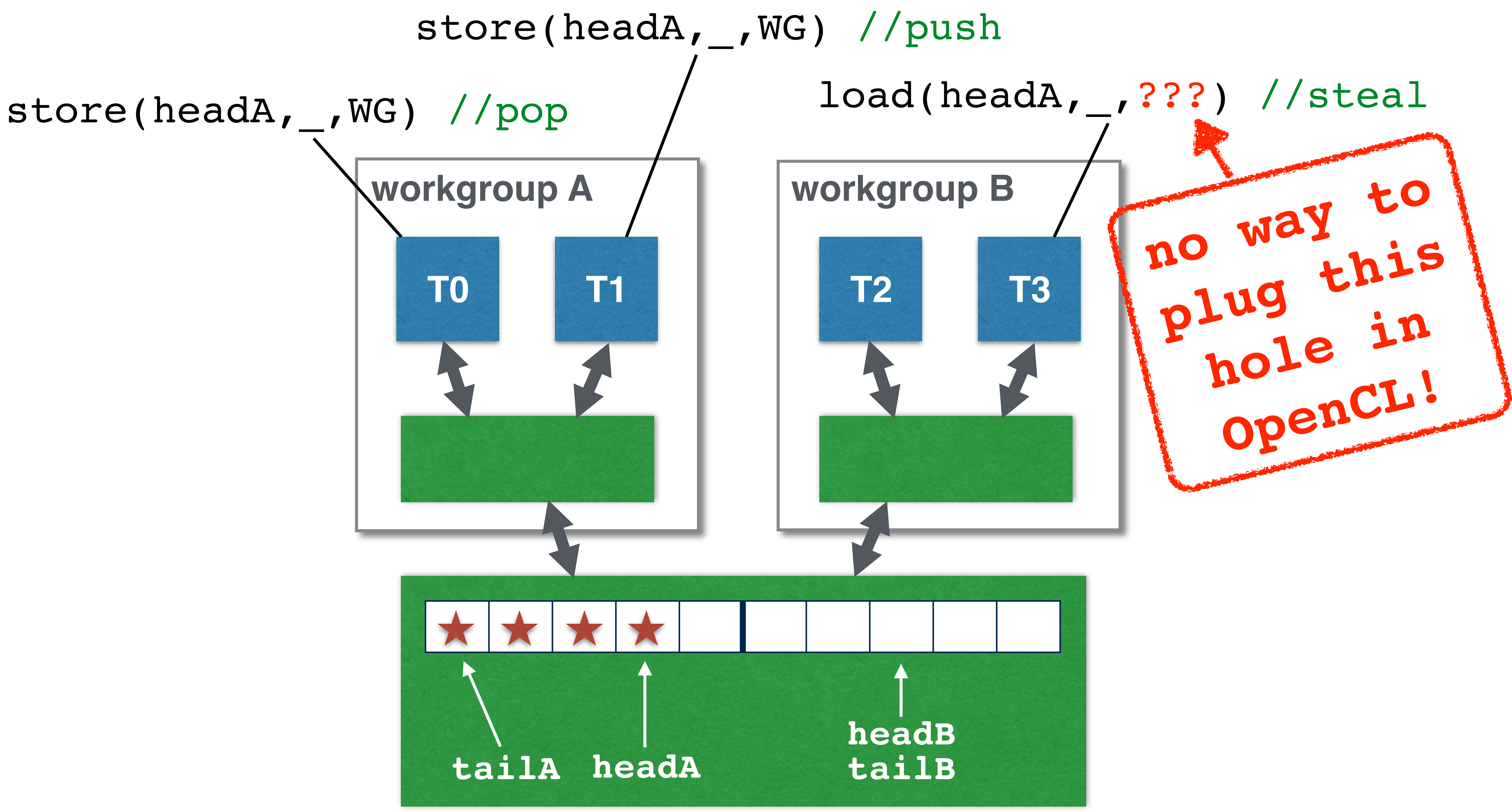
Example: work-stealing



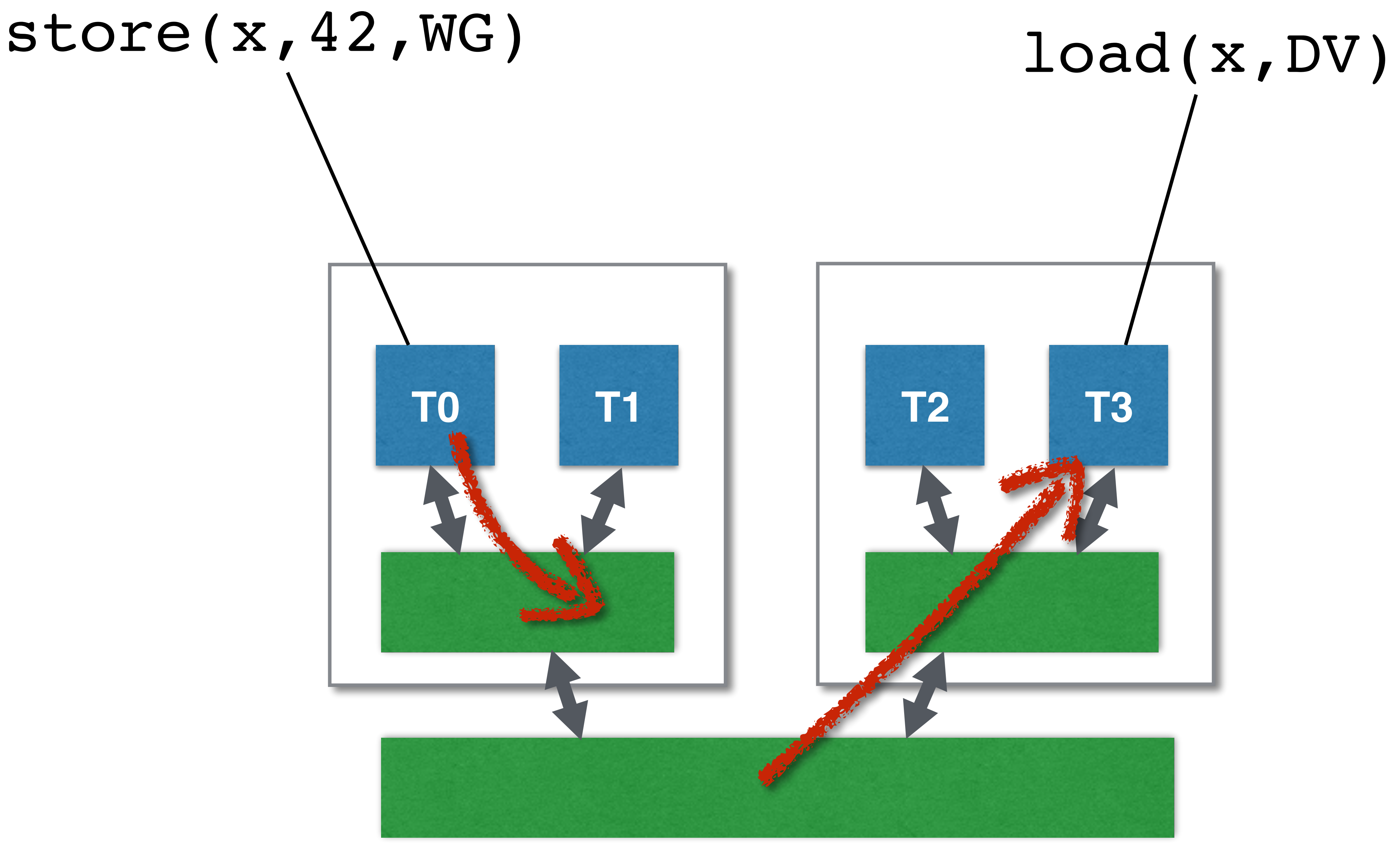
Example: work-stealing



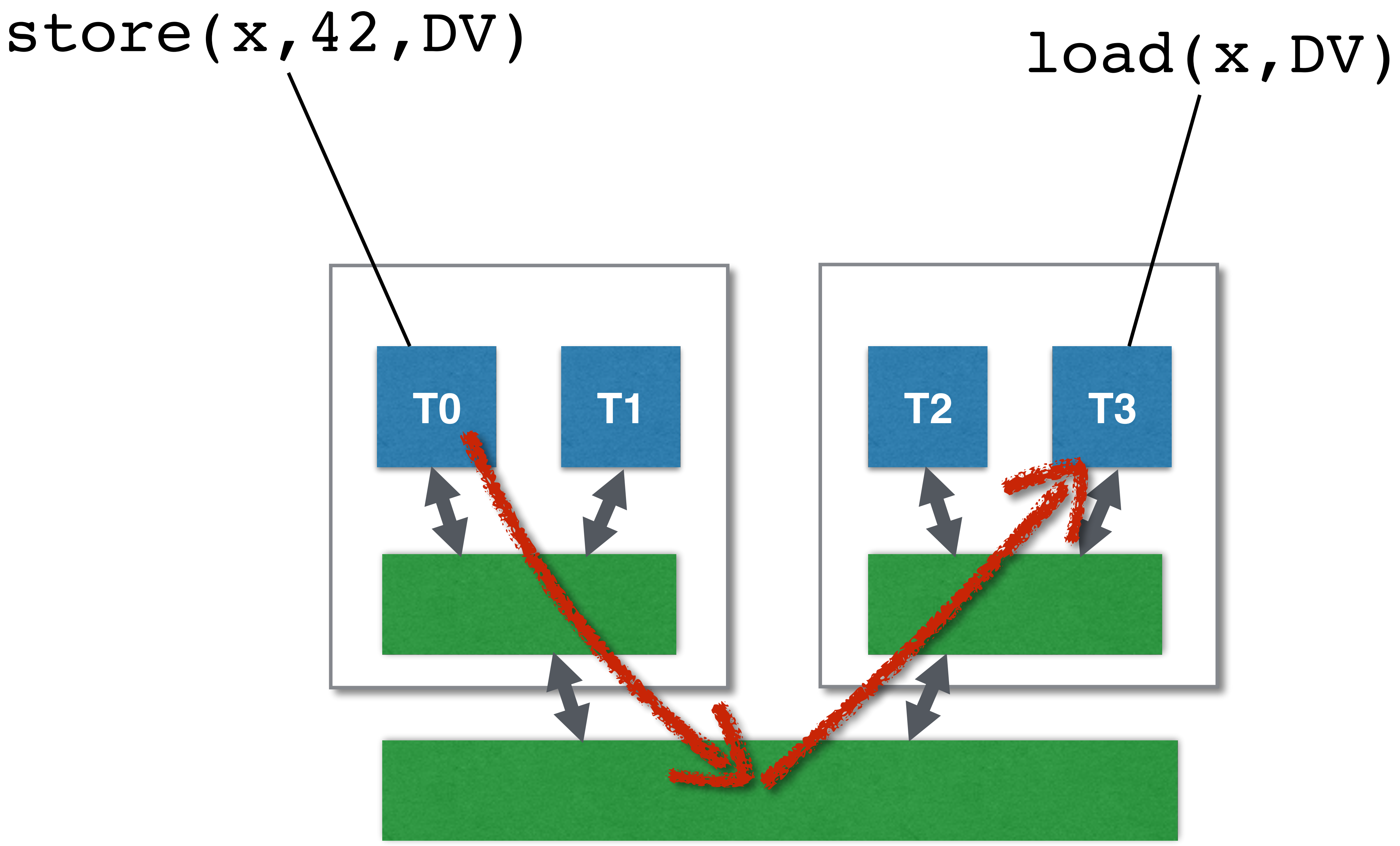
Example: work-stealing



Remote-scope promotion



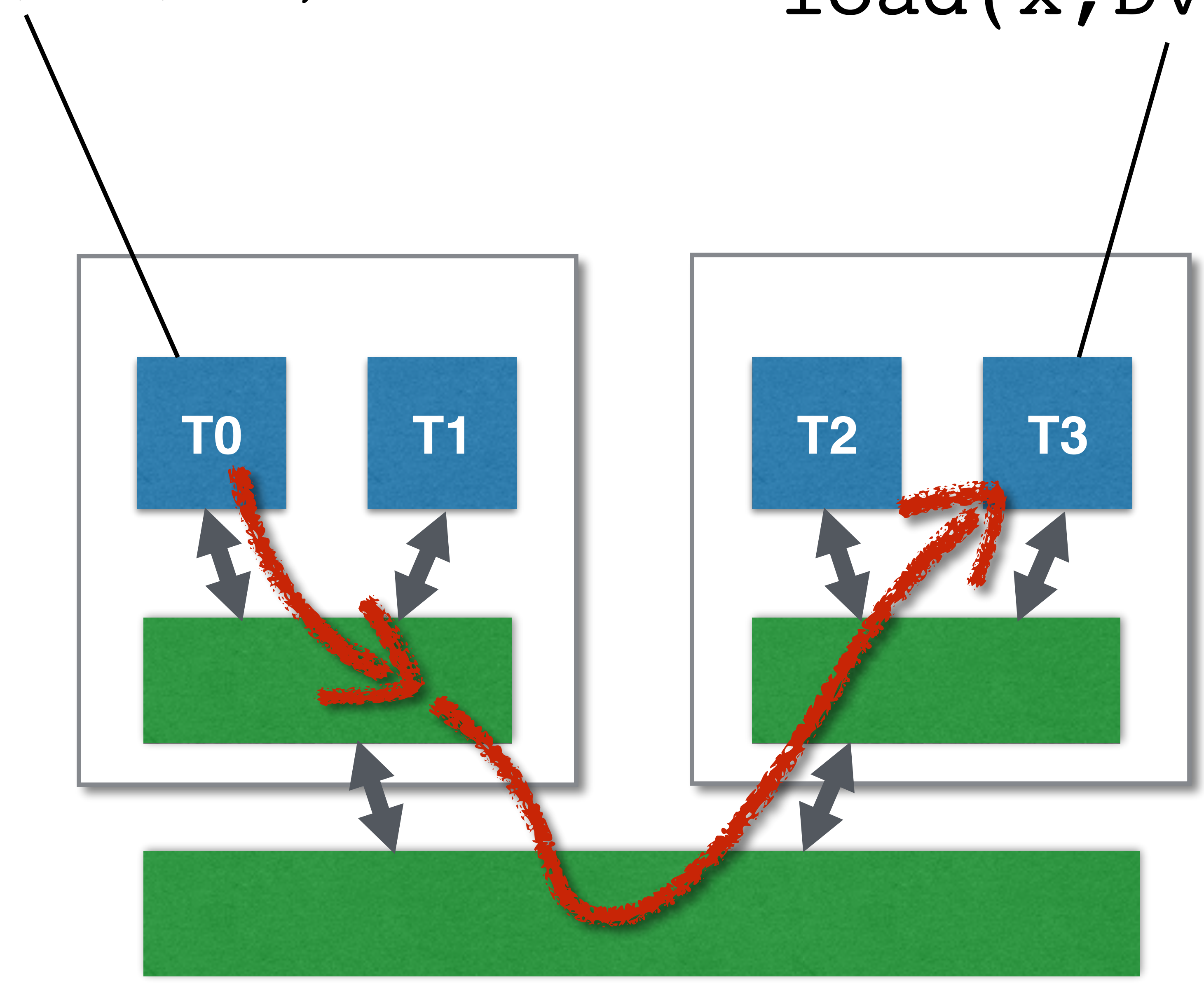
Remote-scope promotion



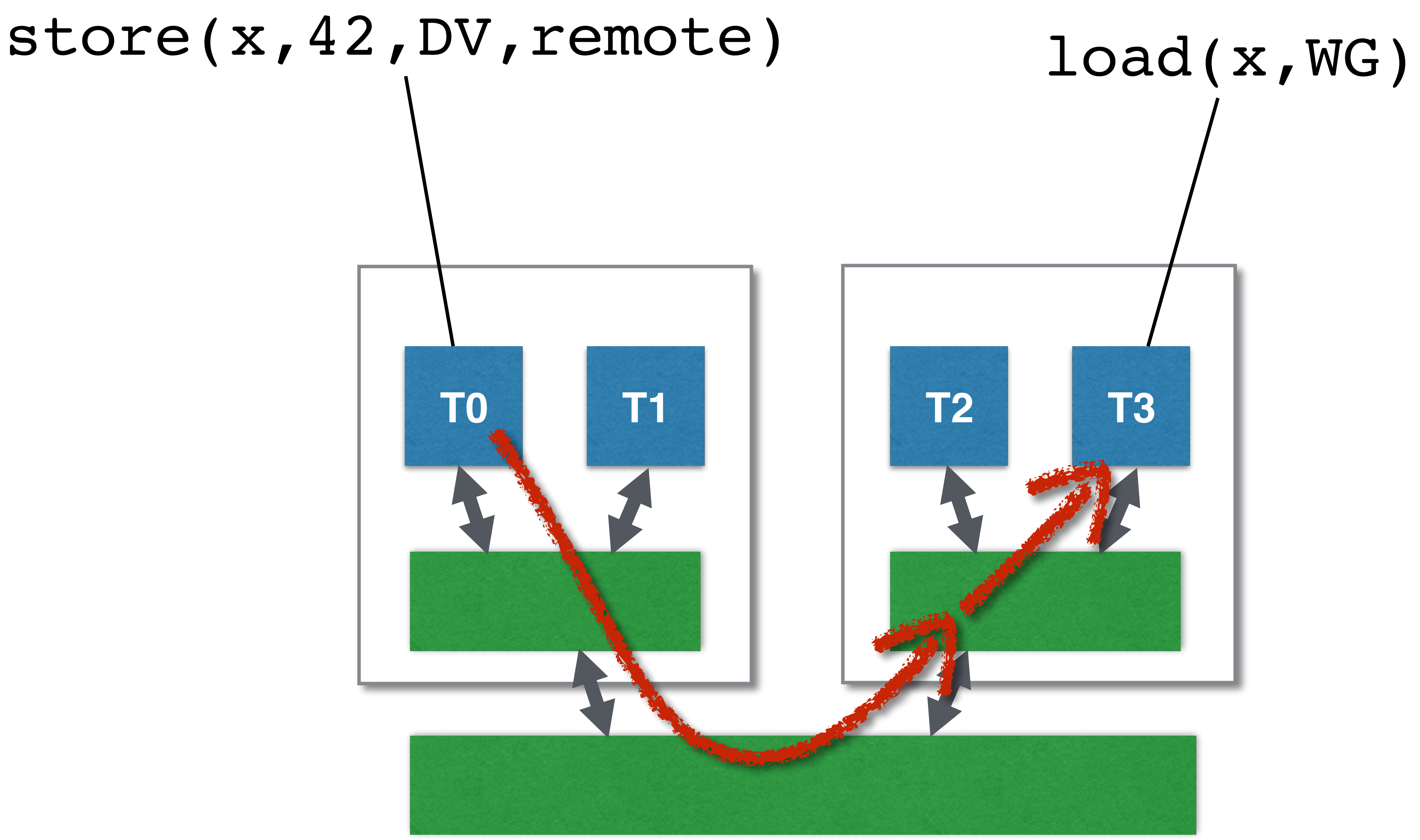
Remote-scope promotion

```
store(x,42,WG)
```

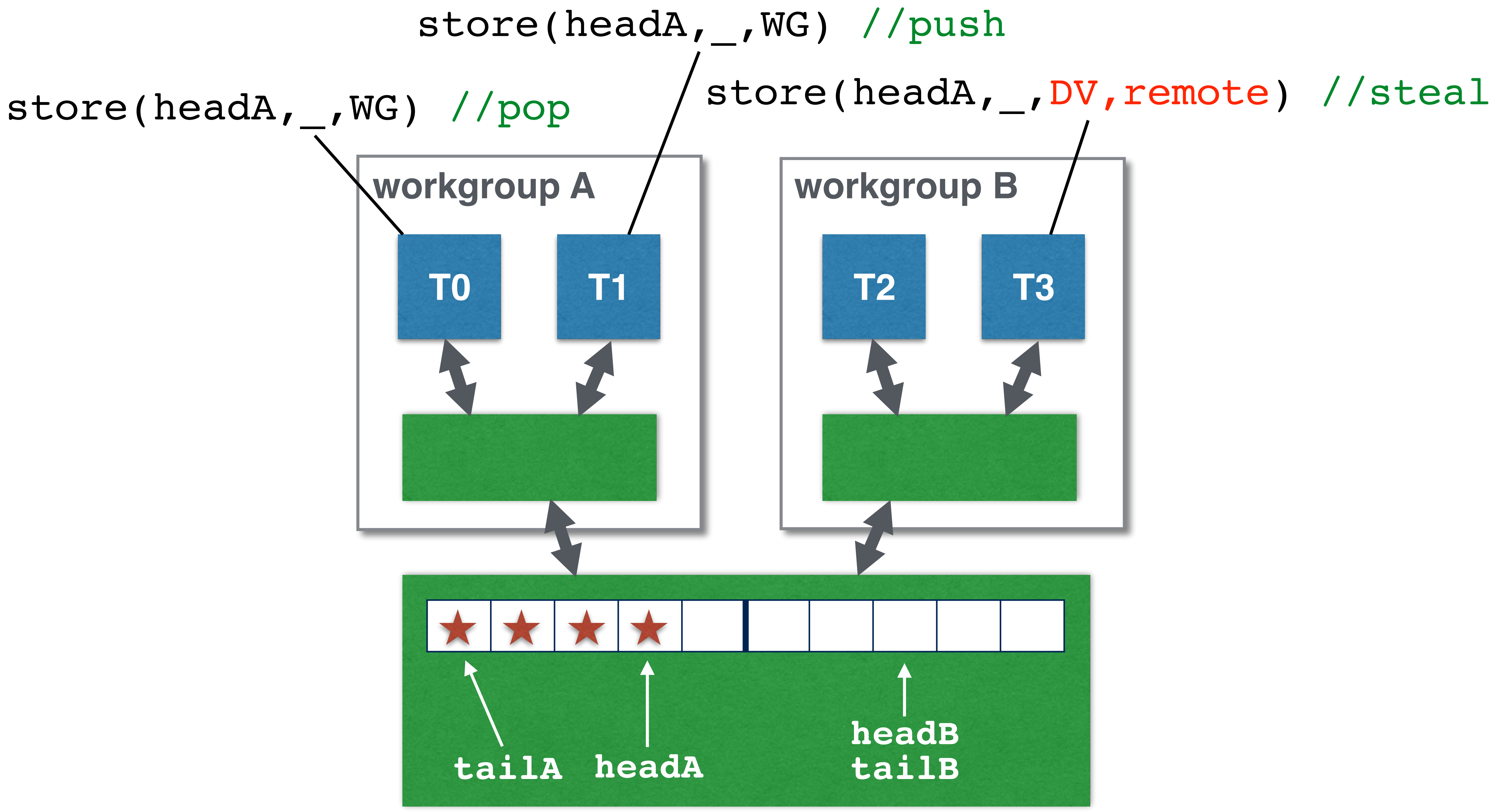
```
load(x,DV,remote)
```



Remote-scope promotion



Work-stealing



This talk

1. Background: What is RSP?
2. ▶ Adding RSP to the OpenCL memory model
3. A formalised implementation of OpenCL+RSP

Scope inclusion in OpenCL

Scope inclusion in OpenCL

- Operations **A** and **B** only synchronise if they have **inclusive scopes**.

Scope inclusion in OpenCL

- Operations **A** and **B** only synchronise if they have **inclusive scopes**.

- **A** and **B** have **inclusive scopes** iff
  A **reaches** **B** and **B** **reaches** **A**.

Scope inclusion in OpenCL

- Operations **A** and **B** only synchronise if they have **inclusive scopes**.

- **A** and **B** have **inclusive scopes** iff
  A **reaches** **B** and **B** **reaches** **A**.

- **A** **reaches** **B** iff
  A has <u>workgroup</u> scope and **B** is in the same workgroup, or
  A has <u>device</u> scope and **B** is in the same device, or
  A has <u>all-devices</u> scope.

Scope inclusion in OpenCL+RSP

- Operations **A** and **B** only synchronise if they have **inclusive scopes**.

- **A** and **B** have **inclusive scopes** iff
  A **reaches** **B** and **B** **reaches** **A**, or
  A **reaches** **B** and **B** **is remote**, or
  A **is remote** and **B** **reaches** **A**.

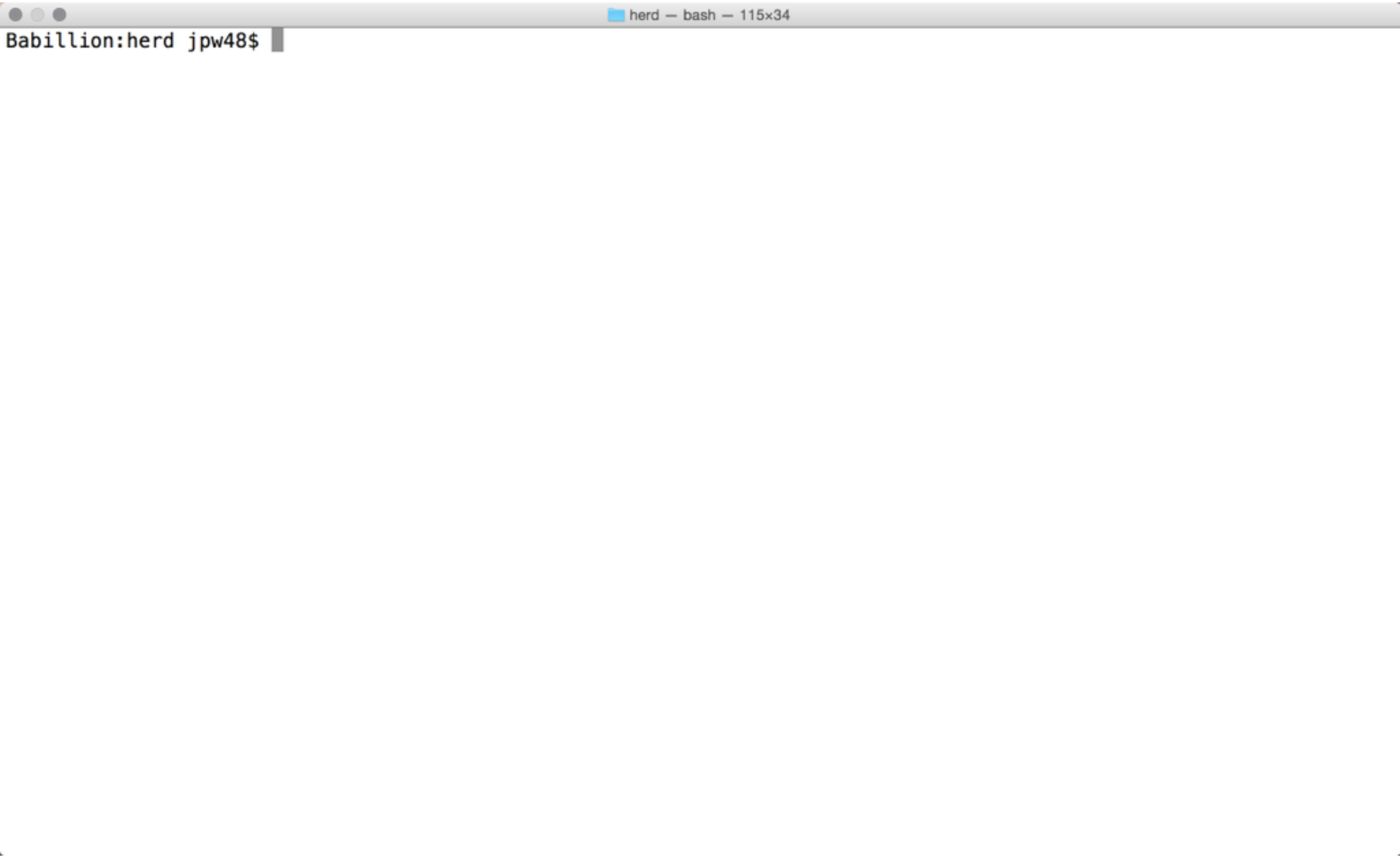
- **A** **reaches** **B** iff
  A has workgroup scope and **B** is in the same workgroup, or
  A has device scope and **B** is in the same device, or
  A has all-devices scope.

```
(* Scope annotations *)
let s_wi = memory_scope_work_item
let s_wg = memory_scope_work_group
let s_dev = memory_scope_device
let s_all = memory_scope_all_svm_devices

(* Inclusive scopes *)

let incl1 = ([s_wi] ; wi)
          | ([s_wg] ; wg)
          | ([s_dev] ; dev)
          | ([s_all] ; unv)

let incl = (incl1 & (incl1^-1))
          | ([remote] ; incl1)
          | ((incl1^-1) ; [remote])

(*******************)
(* Synchronisation *)
(*******************)

let acq = (mo_acq | mo_sc | mo_acq_rel) & (R | F | rmw)
let rel = (mo_rel | mo_sc | mo_acq_rel) & (W | F | rmw)

(* Release sequence *)
```

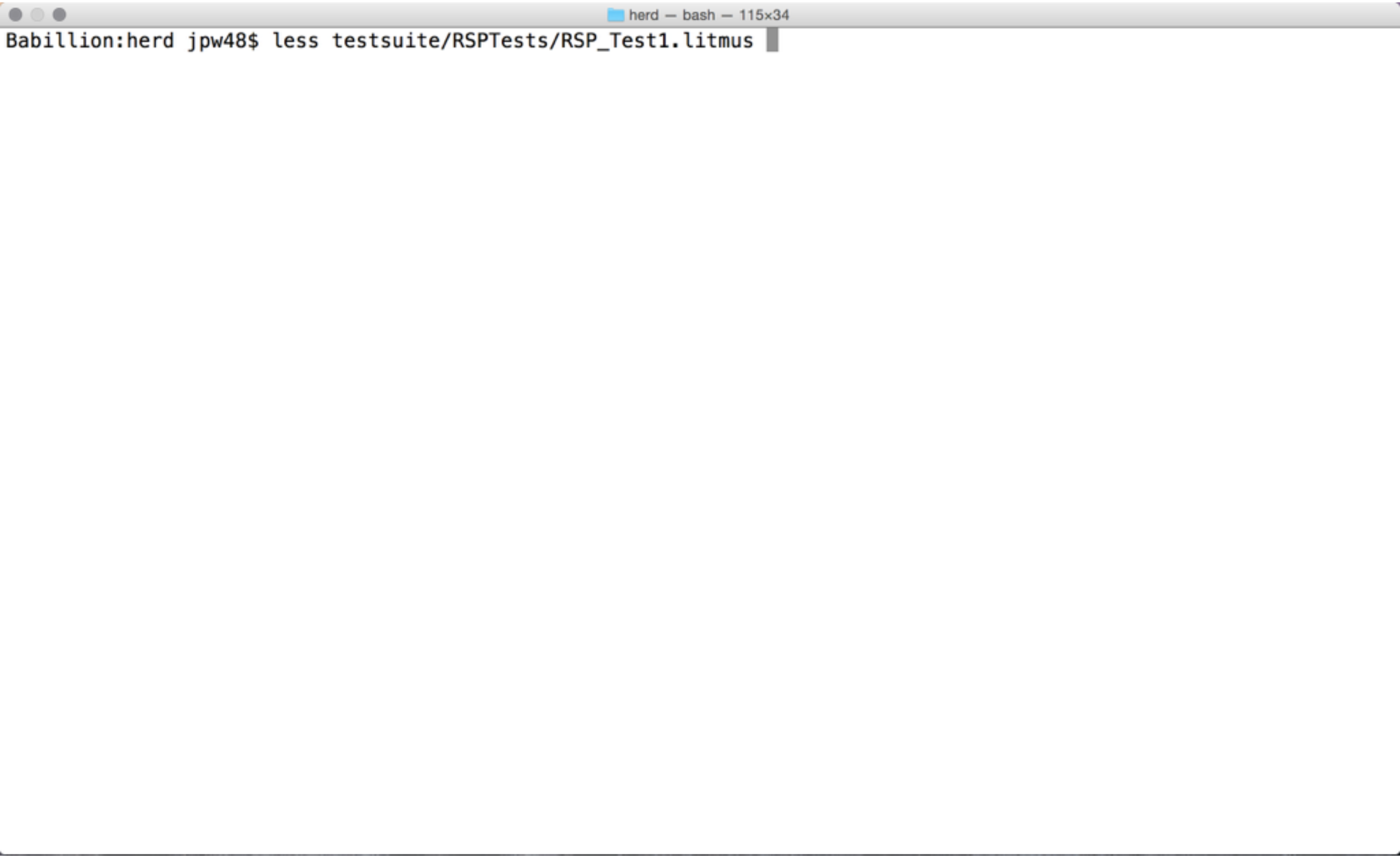
Testing OpenCL+RSP programs

Testing OpenCL+RSP programs

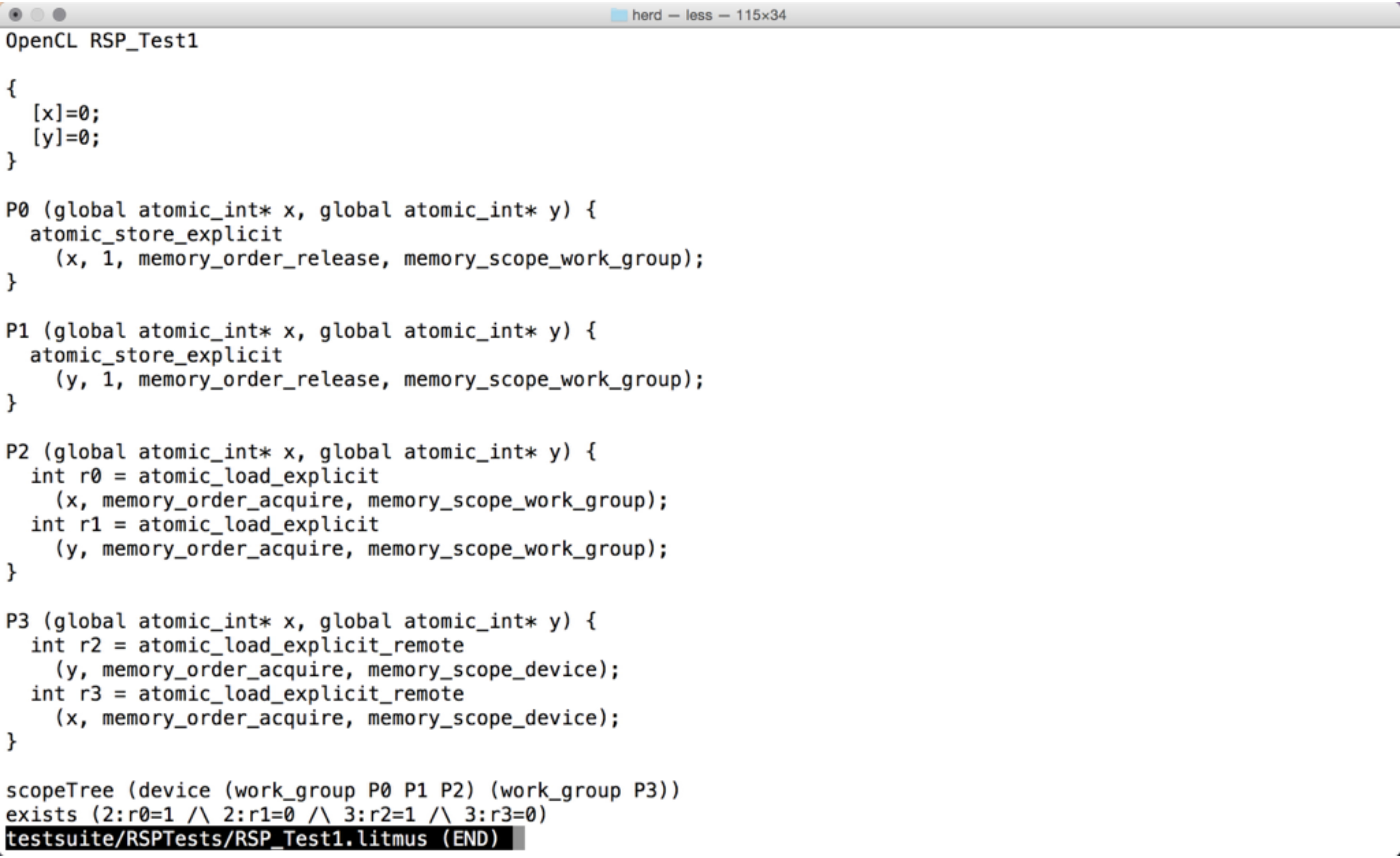
- We simulated the **12** litmus tests designed by the original developers to define their expectations of RSP.



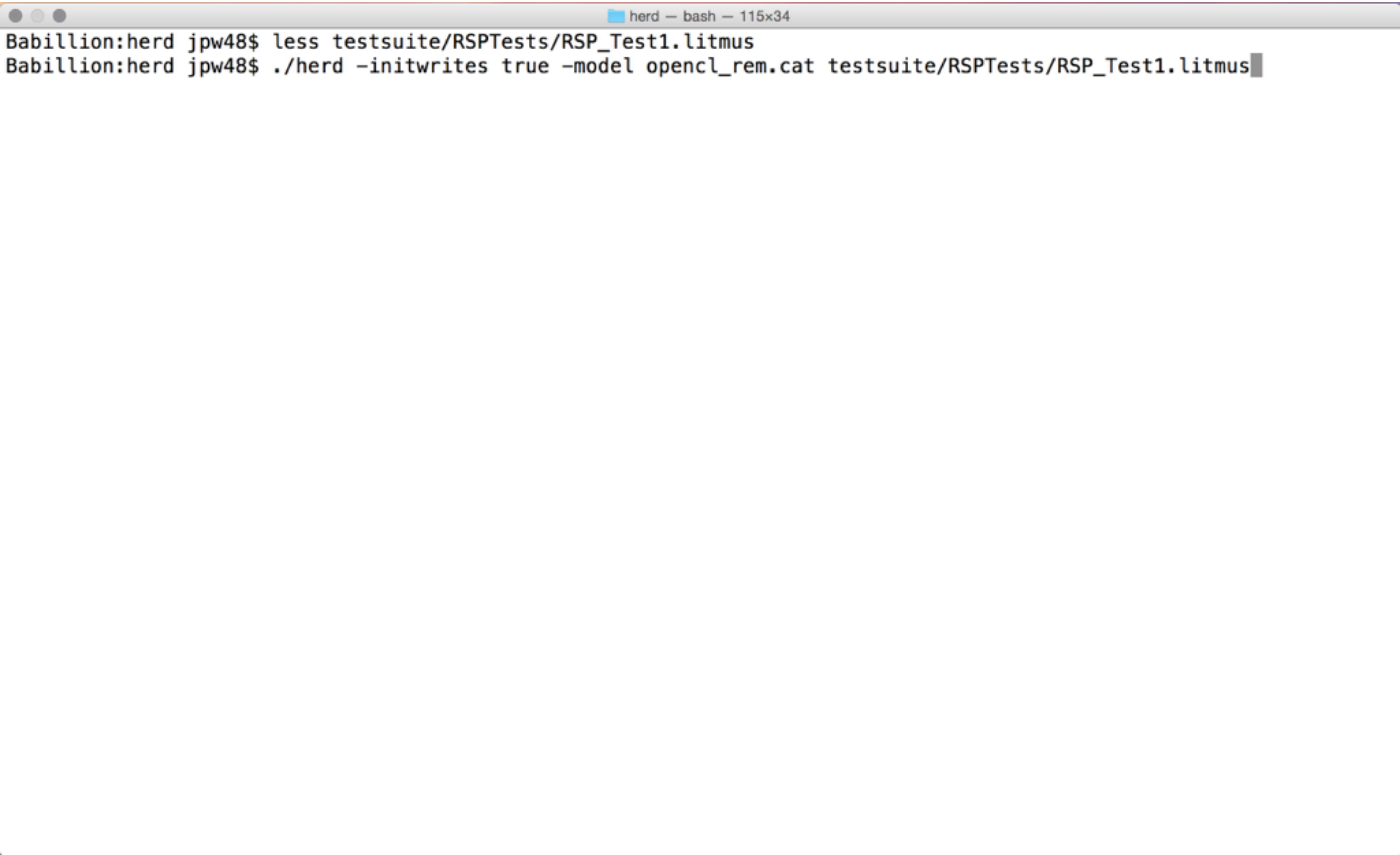
```
Babillion:herd jpw48$ 
```



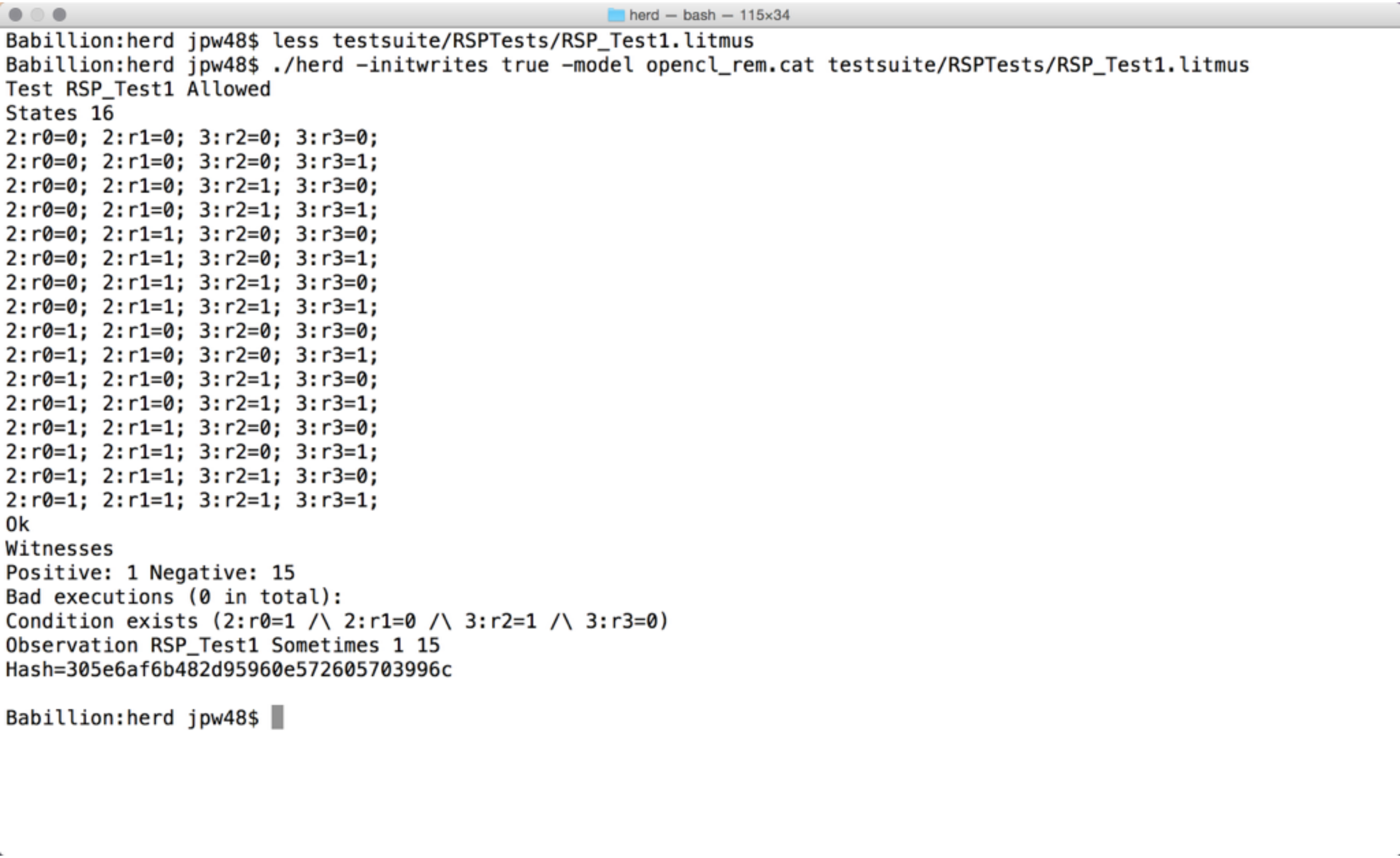
```
Babillion:herd jpw48$ less testsuite/RSPTests/RSP_Test1.litmus
```



```
OpenCL RSP_Test1

{
  [x]=0;
  [y]=0;
}

P0 (global atomic_int* x, global atomic_int* y) {
  atomic_store_explicit
    (x, 1, memory_order_release, memory_scope_work_group);
}

P1 (global atomic_int* x, global atomic_int* y) {
  atomic_store_explicit
    (y, 1, memory_order_release, memory_scope_work_group);
}

P2 (global atomic_int* x, global atomic_int* y) {
  int r0 = atomic_load_explicit
    (x, memory_order_acquire, memory_scope_work_group);
  int r1 = atomic_load_explicit
    (y, memory_order_acquire, memory_scope_work_group);
}

P3 (global atomic_int* x, global atomic_int* y) {
  int r2 = atomic_load_explicit_remote
    (y, memory_order_acquire, memory_scope_device);
  int r3 = atomic_load_explicit_remote
    (x, memory_order_acquire, memory_scope_device);
}

scopeTree (device (work_group P0 P1 P2) (work_group P3))
exists (2:r0=1 /\ 2:r1=0 /\ 3:r2=1 /\ 3:r3=0)
testsuite/RSPTests/RSP_Test1.litmus (END)
```



```
Babillion:herd jpw48$ less testsuite/RSPTests/RSP_Test1.litmus
Babillion:herd jpw48$ ./herd -initwrites true -model opencl_rem.cat testsuite/RSPTests/RSP_Test1.litmus
```



```
Babillion:herd jpw48$ less testsuite/RSPTests/RSP_Test1.litmus
Babillion:herd jpw48$ ./herd -initwrites true -model opencl_rem.cat testsuite/RSPTests/RSP_Test1.litmus
Test RSP_Test1 Allowed
States 16
2:r0=0; 2:r1=0; 3:r2=0; 3:r3=0;
2:r0=0; 2:r1=0; 3:r2=0; 3:r3=1;
2:r0=0; 2:r1=0; 3:r2=1; 3:r3=0;
2:r0=0; 2:r1=0; 3:r2=1; 3:r3=1;
2:r0=0; 2:r1=1; 3:r2=0; 3:r3=0;
2:r0=0; 2:r1=1; 3:r2=0; 3:r3=1;
2:r0=0; 2:r1=1; 3:r2=1; 3:r3=0;
2:r0=0; 2:r1=1; 3:r2=1; 3:r3=1;
2:r0=1; 2:r1=0; 3:r2=0; 3:r3=0;
2:r0=1; 2:r1=0; 3:r2=0; 3:r3=1;
2:r0=1; 2:r1=0; 3:r2=1; 3:r3=0;
2:r0=1; 2:r1=0; 3:r2=1; 3:r3=1;
2:r0=1; 2:r1=1; 3:r2=0; 3:r3=0;
2:r0=1; 2:r1=1; 3:r2=0; 3:r3=1;
2:r0=1; 2:r1=1; 3:r2=1; 3:r3=0;
2:r0=1; 2:r1=1; 3:r2=1; 3:r3=1;
Ok
Witnesses
Positive: 1 Negative: 15
Bad executions (0 in total):
Condition exists (2:r0=1 /\ 2:r1=0 /\ 3:r2=1 /\ 3:r3=0)
Observation RSP_Test1 Sometimes 1 15
Hash=305e6af6b482d95960e572605703996c

Babillion:herd jpw48$
```

Testing OpenCL+RSP programs

- We simulated the **12** litmus tests designed by the original developers to define their expectations of RSP.

- We found **8** were good, but:
  2 had unintentional races,
  1 enforced broken behaviour, and
  1 forbade reasonable behaviour.

Testing OpenCL+RSP programs

- We simulated the **12** litmus tests designed by the original developers to define their expectations of RSP.

- We found **8** were good, but:
  2 had unintentional races,
  1 enforced broken behaviour, and
  1 forbade reasonable behaviour.

- We also found (and fixed) bugs in their work-stealing queue implementation

This talk

1. Background: What is RSP?
2. Adding RSP to the OpenCL memory model
3. ▶ A formalised implementation of OpenCL+RSP

Implementing RSP

Implementing RSP

- Model of GPU hardware

Implementing RSP

- Model of GPU hardware
- Assembly-like language

Implementing RSP

- Model of GPU hardware
- Assembly-like language
- Compiler mapping from OpenCL+RSP operations to sequences of assembly instructions

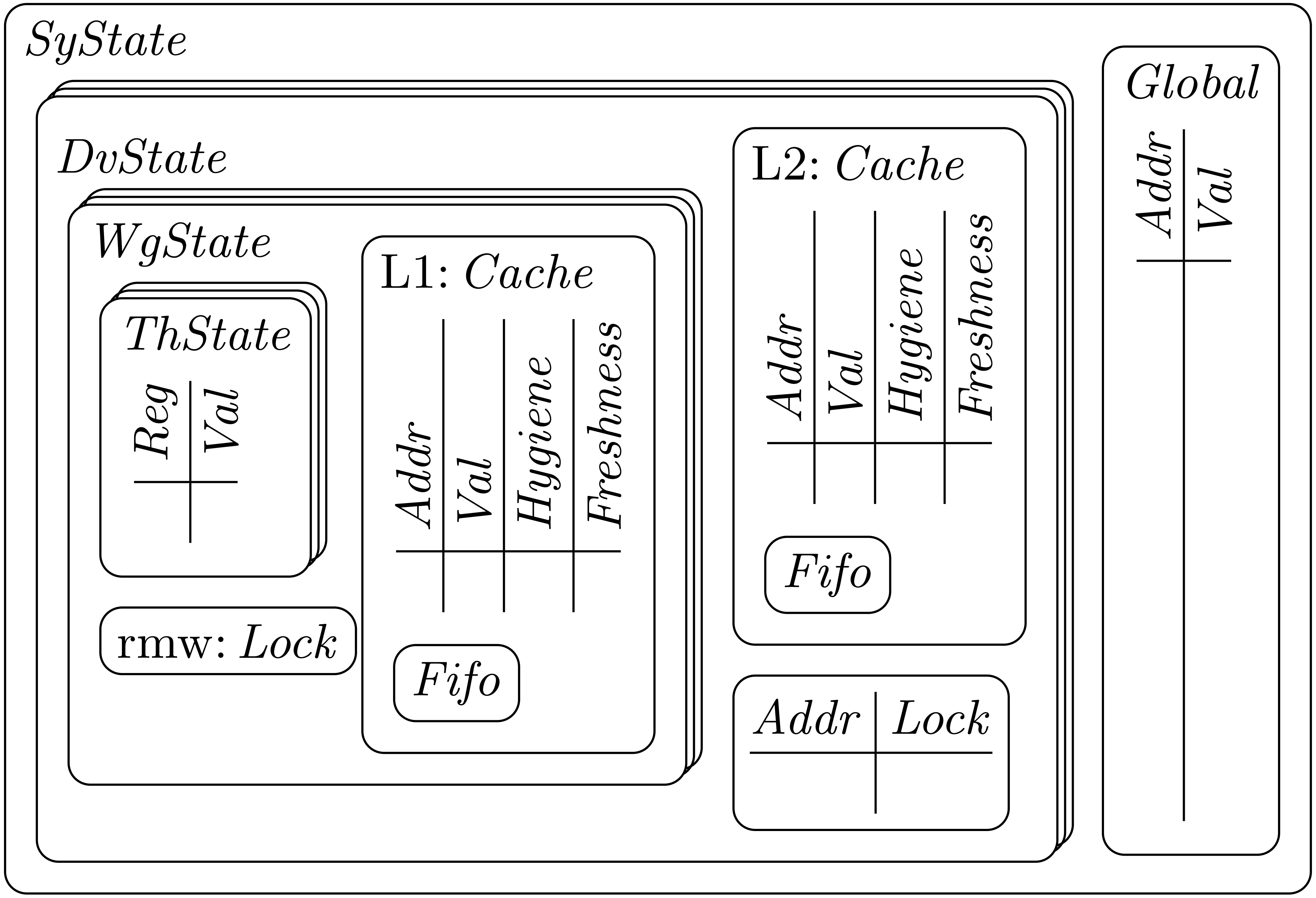
Implementing RSP

- Model of GPU hardware
- Assembly-like language
- Compiler mapping from OpenCL+RSP operations to sequences of assembly instructions
- Can then prove that all behaviours of the compiled program are allowed by the OpenCL+RSP memory model.

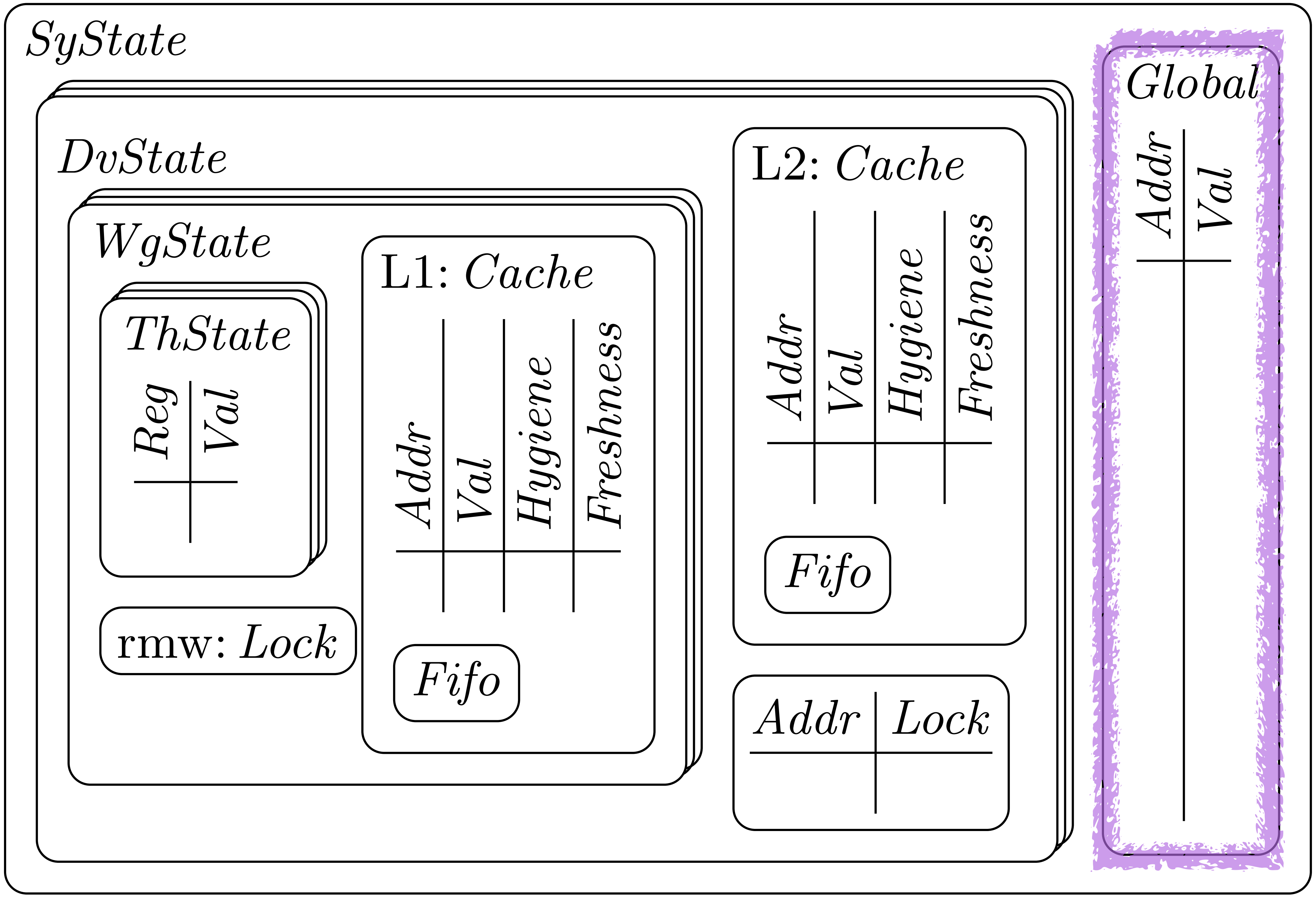
Model of GPU hardware

$$
\begin{array}{rrcl}
x \in & \mathit{Addr} & & \\
r \in & \mathit{Reg} & & \\
v \in & \mathit{Val} & \stackrel{\text{def}}{=} & \mathbb{Z} \\
& \mathit{FifoEl} & \stackrel{\text{def}}{=} & \mathit{Addr} \cup \{\textsc{flush}_{d\,w\,t} \mid d, w, t \in \mathbb{N}\} \\
& \mathit{Fifo} & \stackrel{\text{def}}{=} & \mathit{FifoEl}\ \text{queue} \\
& \mathit{Hygiene} & \stackrel{\text{def}}{=} & \{\textsc{clean}, \textsc{dirty}\} \\
& \mathit{Freshness} & \stackrel{\text{def}}{=} & \{\textsc{valid}, \textsc{inv'd}\} \\
& \mathit{CacheEntry} & \stackrel{\text{def}}{=} & \mathit{Val} \times (\text{hy: } \mathit{Hygiene}) \times (\text{fr: } \mathit{Freshness}) \\
C \in & \mathit{Cache} & \stackrel{\text{def}}{=} & (\mathit{Addr} \rightharpoonup \mathit{CacheEntry}) \times (\text{fifo: } \mathit{Fifo}) \\
& \mathit{Lock} & \stackrel{\text{def}}{=} & \{\text{🔒}, \text{🔓}\} \\
& \mathit{ThState} & \stackrel{\text{def}}{=} & \mathit{Reg} \to \mathit{Val} \\
& \mathit{WgState} & \stackrel{\text{def}}{=} & \mathit{ThState}\ \text{list} \times (\text{L1: } \mathit{Cache}) \times (\text{rmw: } \mathit{Lock}) \\
& \mathit{DvState} & \stackrel{\text{def}}{=} & \mathit{WgState}\ \text{list} \times (\text{L2: } \mathit{Cache}) \times \\
& & & (\text{lockfile: } \mathit{Addr} \to \mathit{Lock}) \\
& \mathit{Global} & \stackrel{\text{def}}{=} & \mathit{Addr} \rightharpoonup \mathit{Val} \\
\Sigma \in & \mathit{SyState} & \stackrel{\text{def}}{=} & \mathit{DvState}\ \text{list} \times (\text{gl: } \mathit{Global})
\end{array}
$$

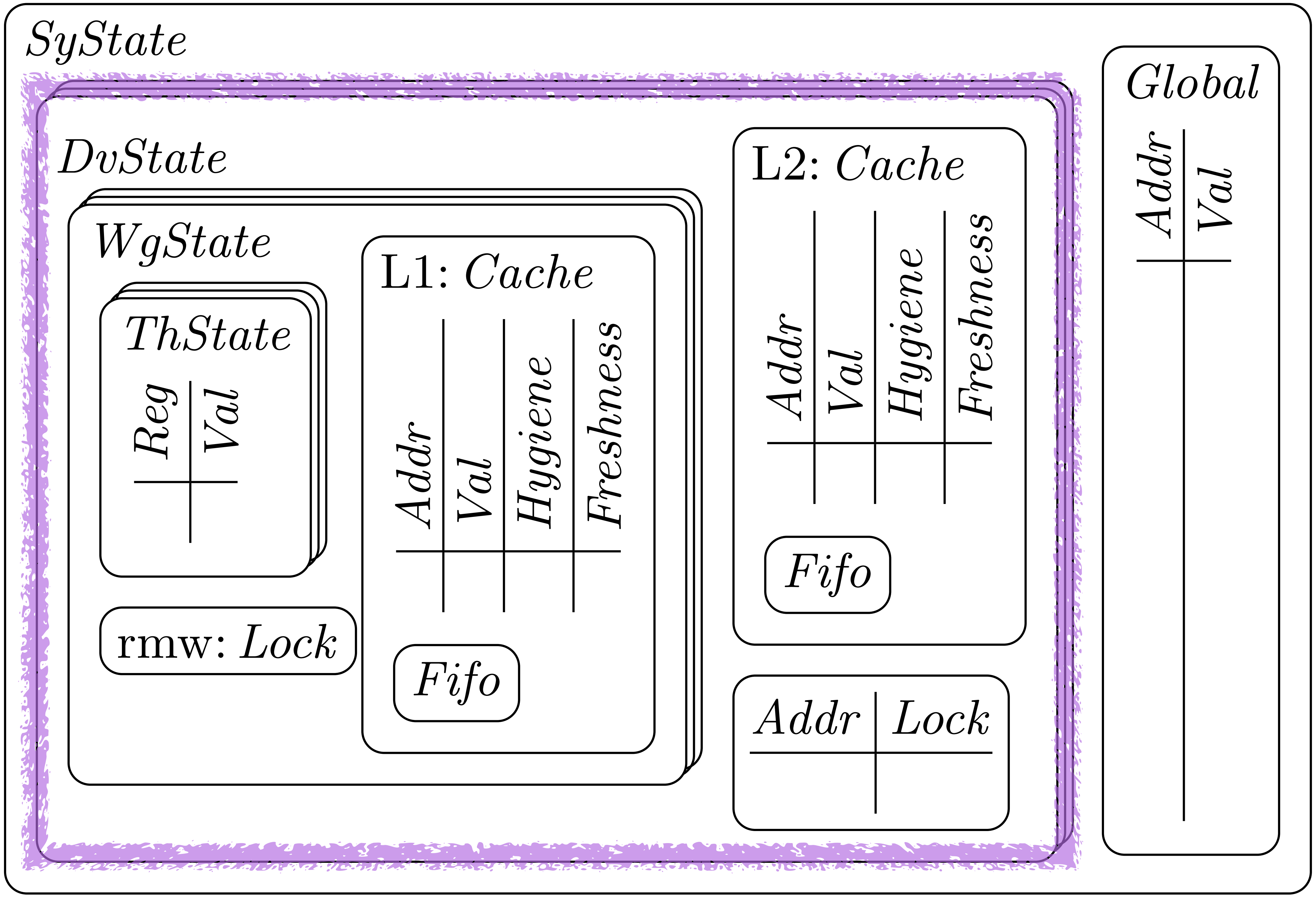
Model of GPU hardware



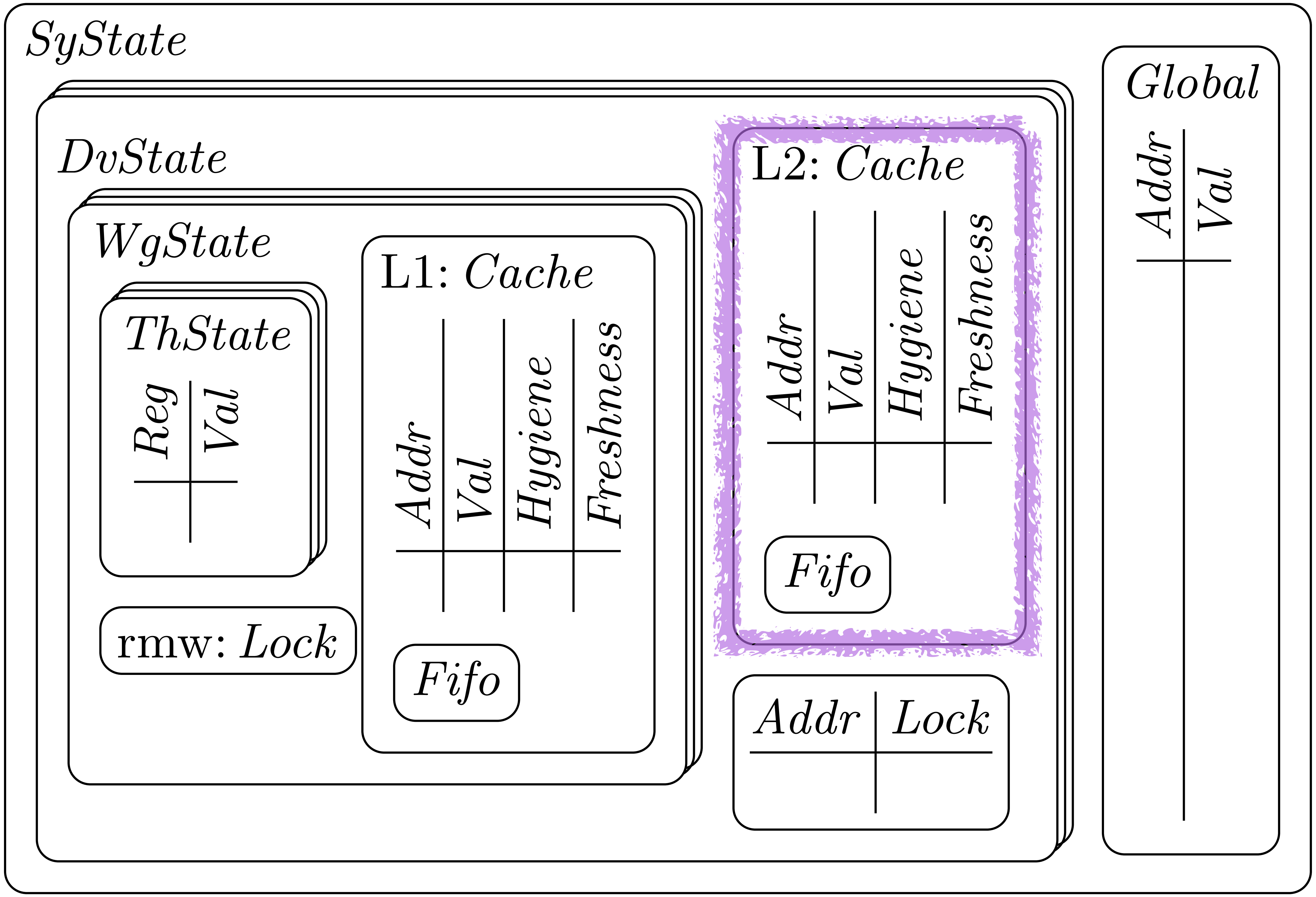
Model of GPU hardware



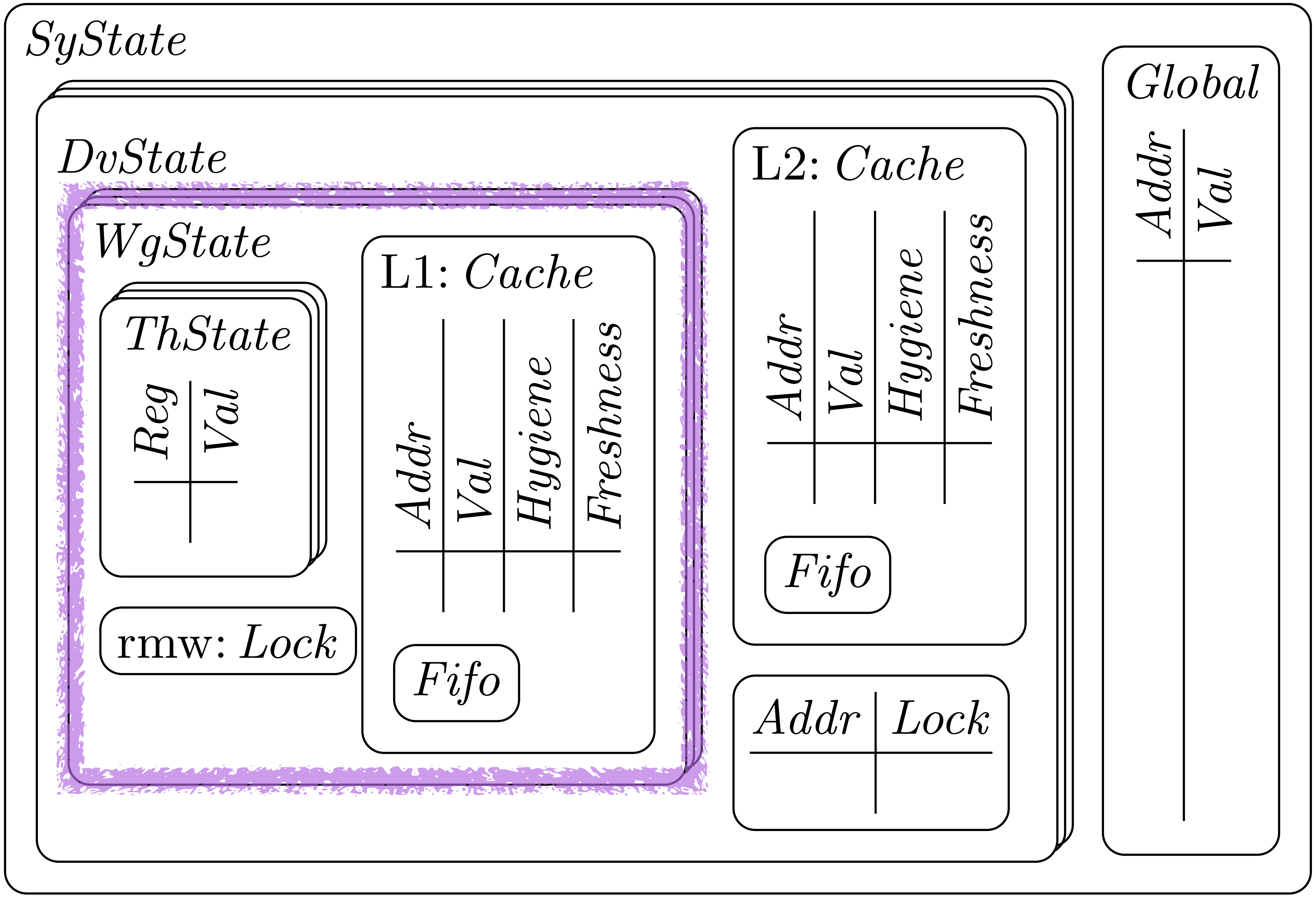
Model of GPU hardware



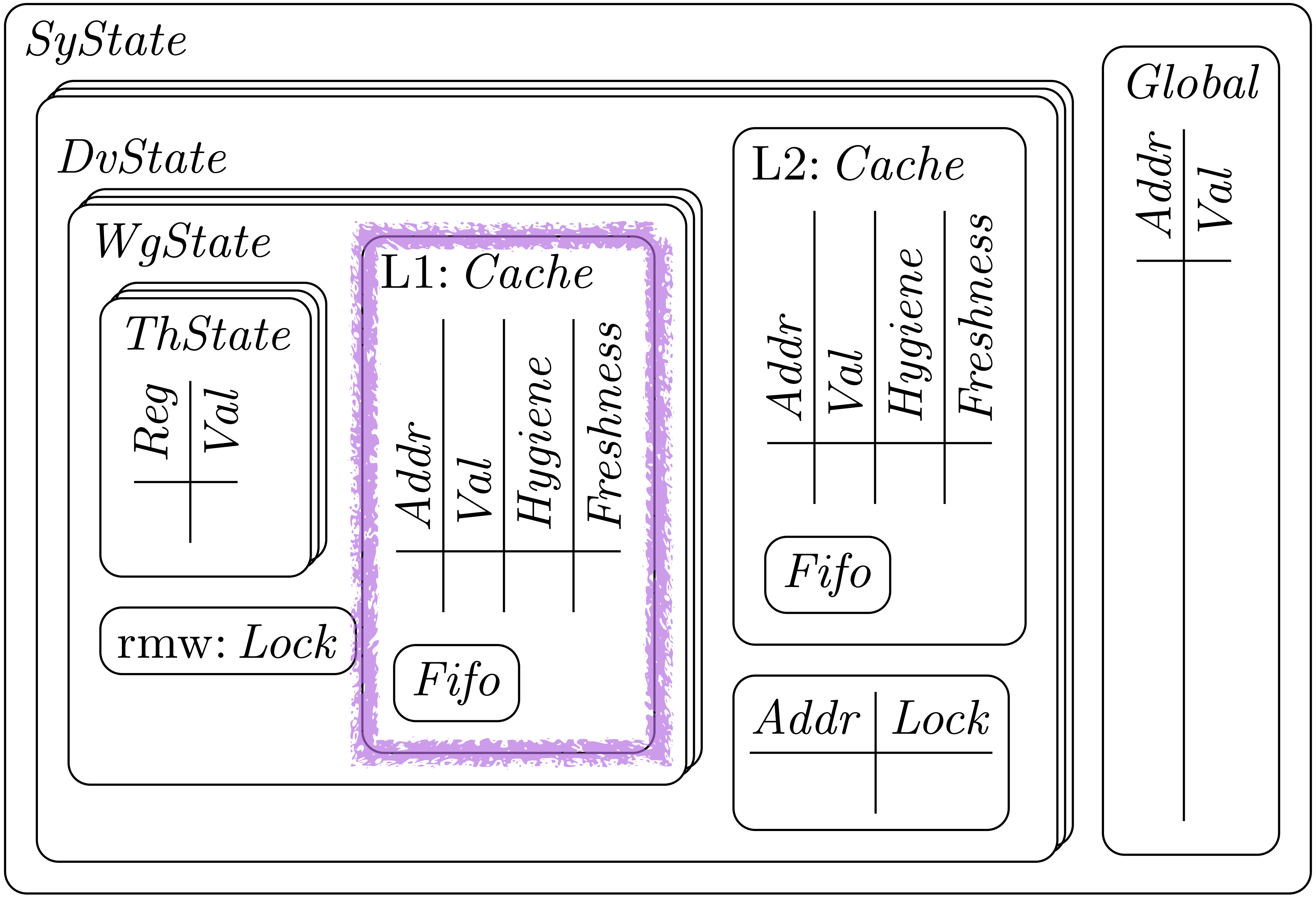
Model of GPU hardware



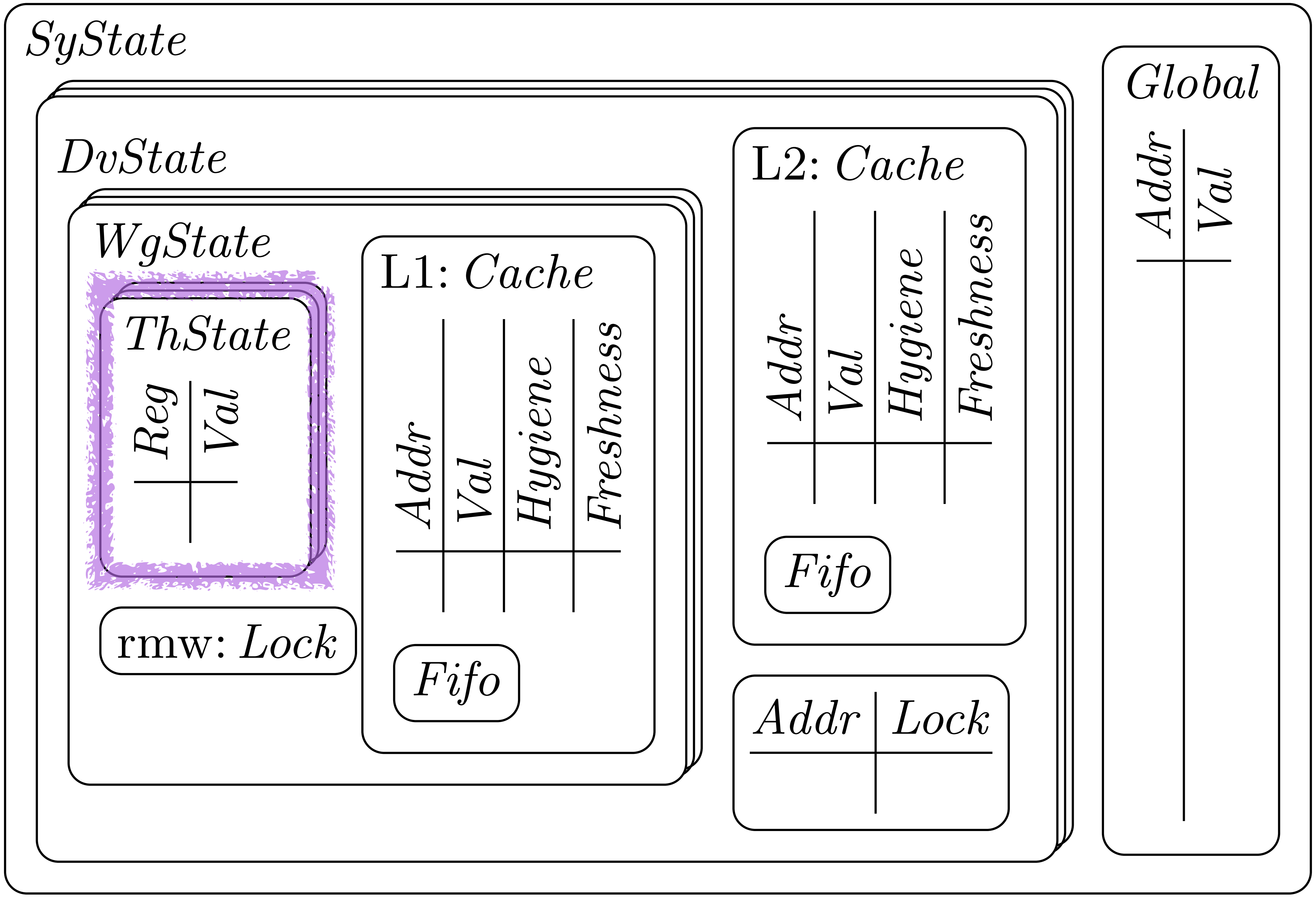
Model of GPU hardware



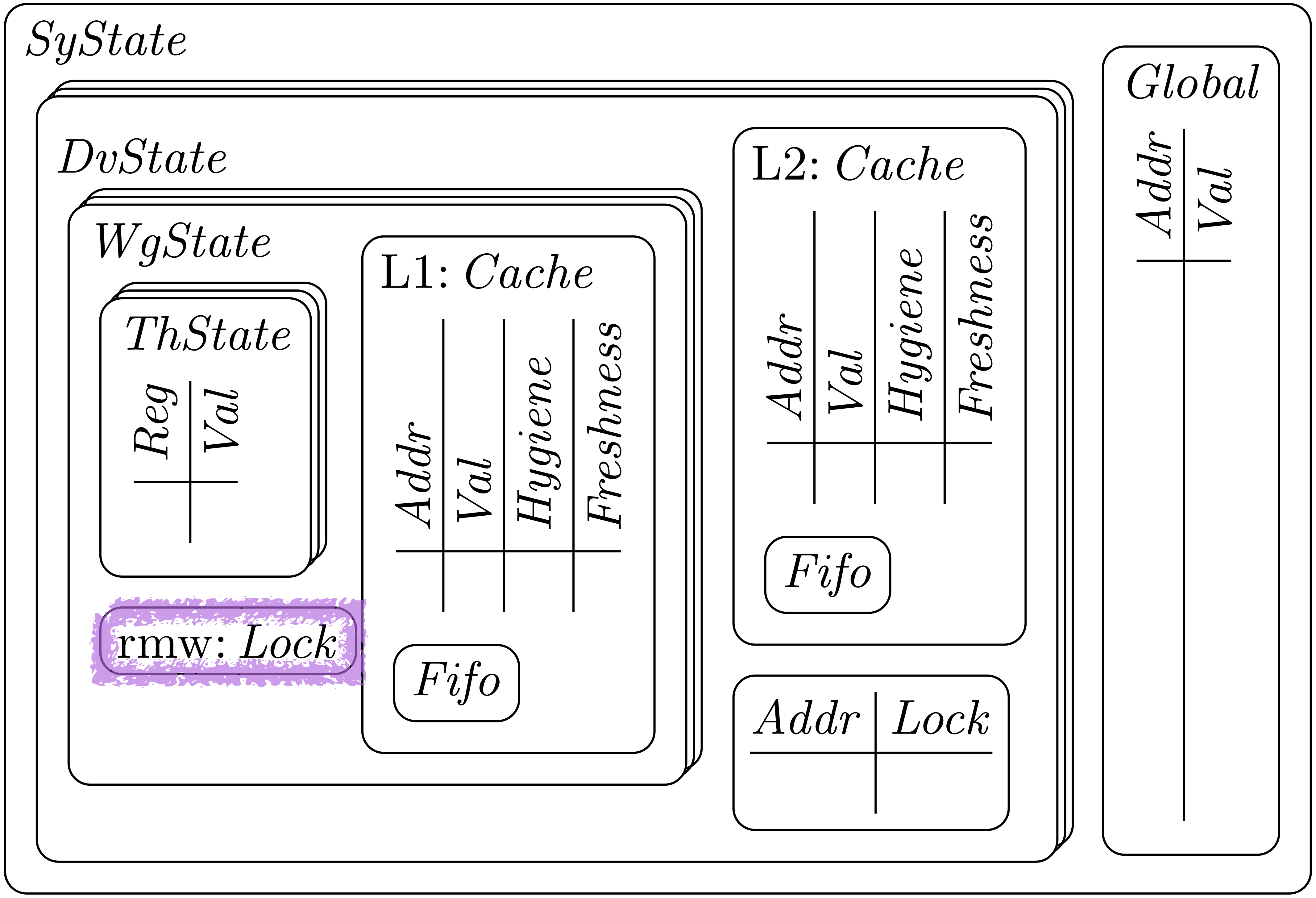
Model of GPU hardware



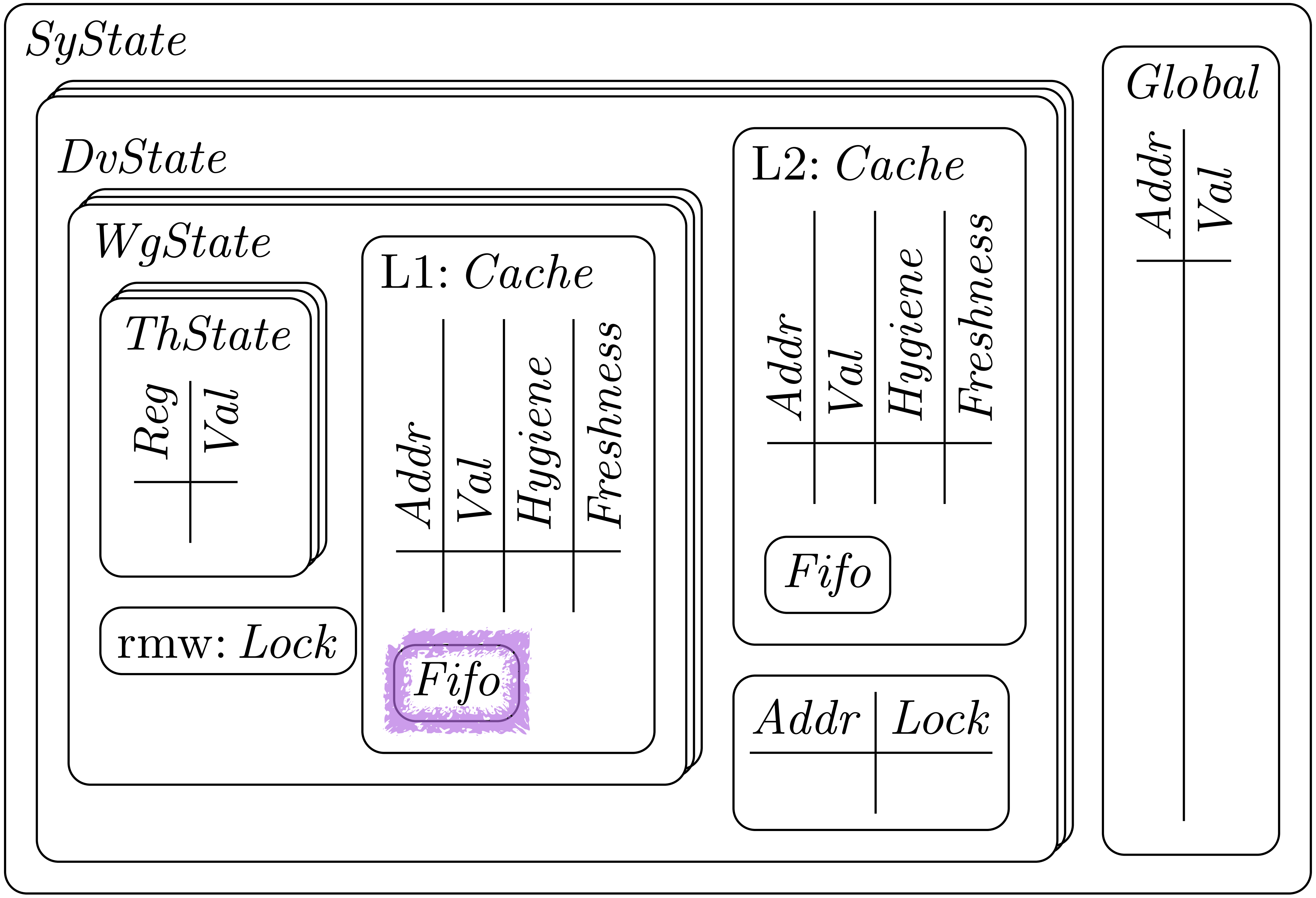
Model of GPU hardware



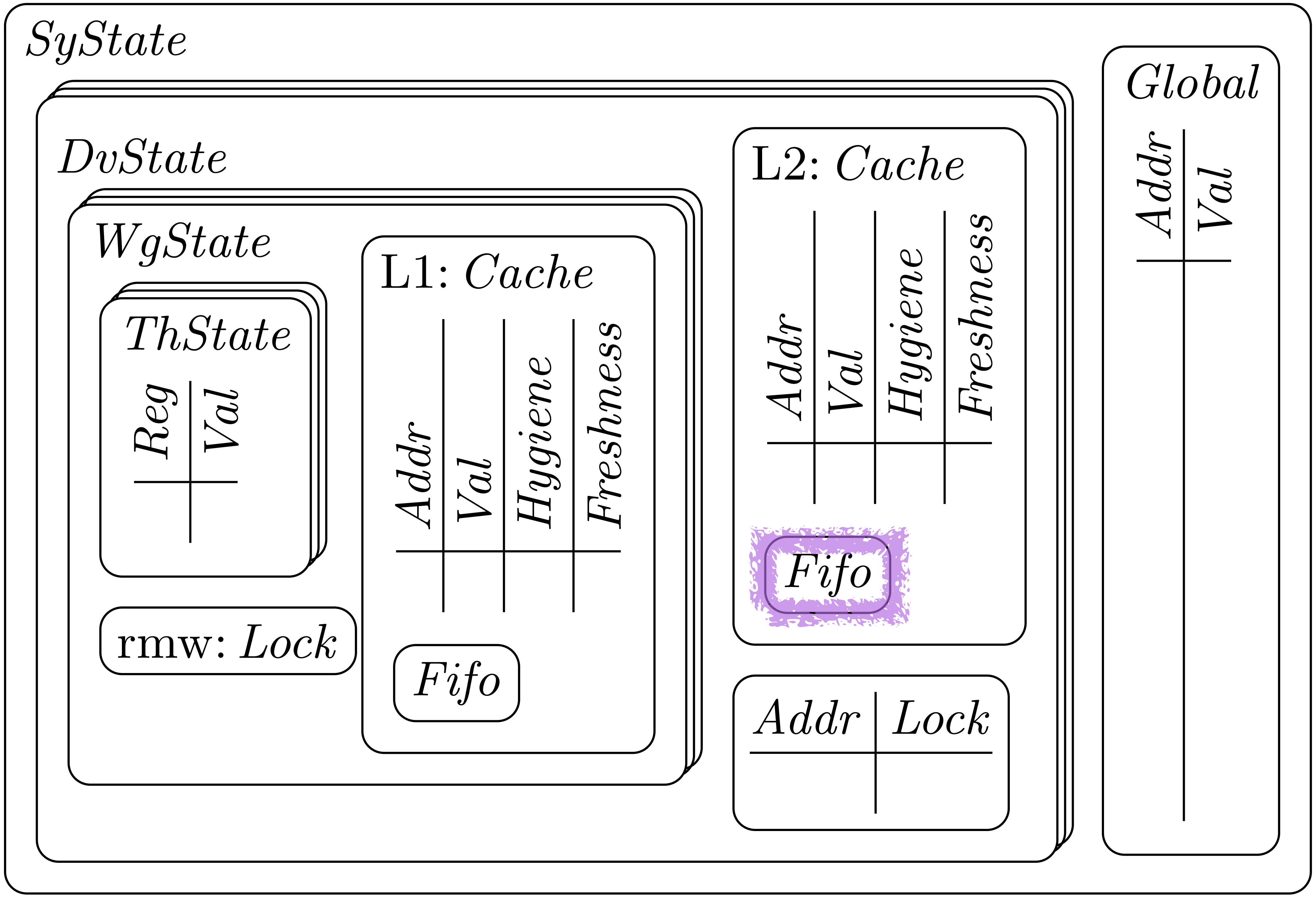
Model of GPU hardware



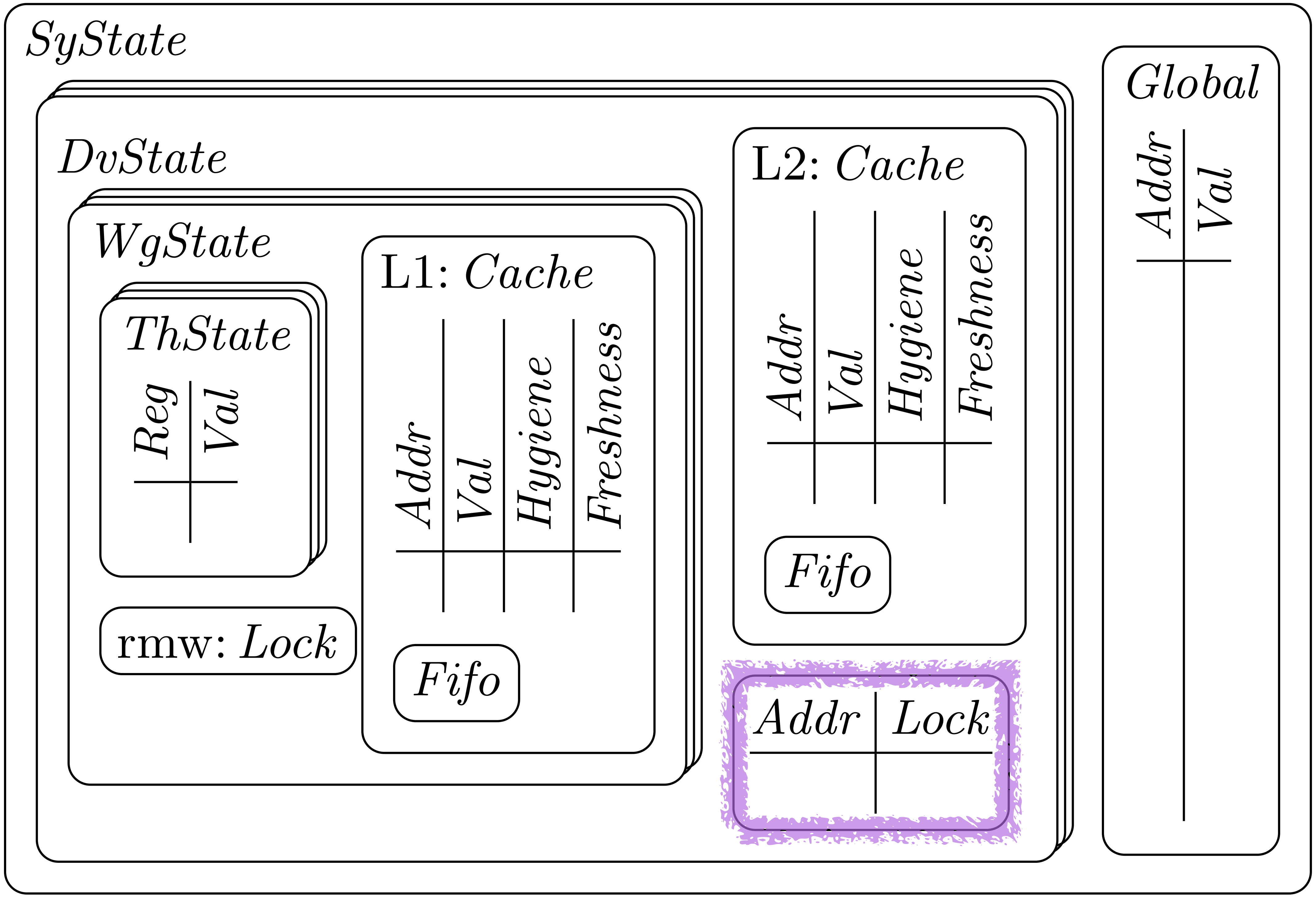
Model of GPU hardware



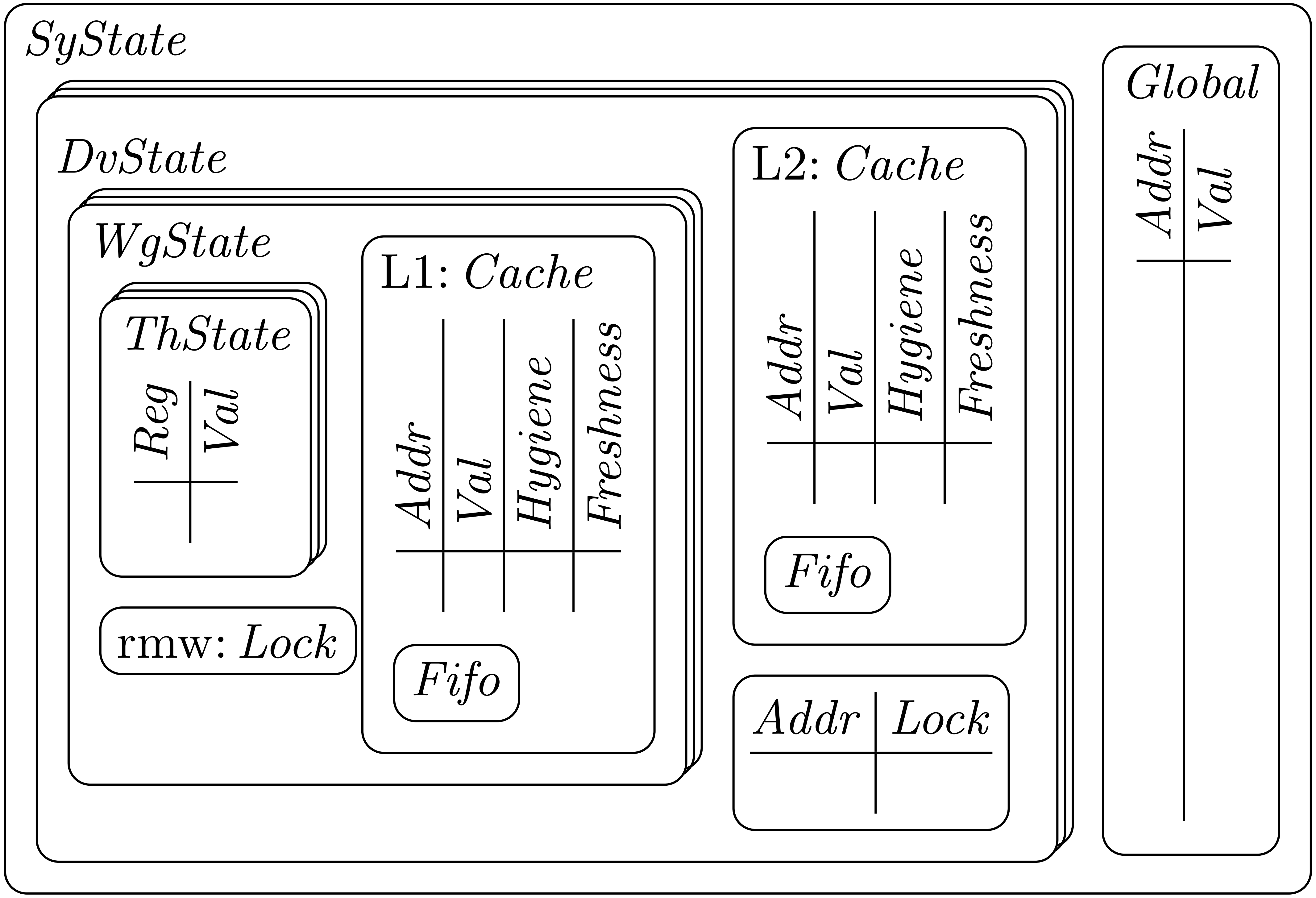
Model of GPU hardware



Model of GPU hardware



Model of GPU hardware



Original scheme

	na or WG	DV (not remote)	DV (remote)
r=load(x)	LD r x	INV$_{L1}$ WG LD r x	$\left.\begin{array}{l}\texttt{FLU}_{L1}\ \texttt{DV}\\ \texttt{INV}_{L1}\ \texttt{WG}\\ \texttt{LD r x}\end{array}\right\}$ LK x
store(x,r)	ST r x	FLU$_{L1}$ WG ST r x	$\left.\begin{array}{l}\texttt{FLU}_{L1}\ \texttt{WG}\\ \texttt{ST r x}\\ \texttt{INV}_{L1}\ \texttt{DV}\end{array}\right\}$ LK x
r=fetch_inc(x)	INC$_{L1}$ r x	FLU$_{L1}$ WG INV$_{L1}$ WG INC$_{L2}$ r x	$\left.\begin{array}{l}\texttt{FLU}_{L1}\ \texttt{DV}\\ \texttt{INV}_{L1}\ \texttt{WG}\\ \texttt{INC}_{L2}\ \texttt{r x}\\ \texttt{INV}_{L1}\ \texttt{DV}\end{array}\right\}$ LK x LK$_{rmw}$

Original scheme

	na or WG	DV (not remote)	DV (remote)
r=load(x)	LD r x	INV$_{L1}$ WG LD r x	$\left.\begin{array}{l}\text{FLU}_{L1}\ \text{DV}\\ \text{INV}_{L1}\ \text{WG}\\ \text{LD r x}\end{array}\right\}\ \text{LK x}$
store(x,r)	ST r x	FLU$_{L1}$ WG ST r x	$\left.\begin{array}{l}\text{FLU}_{L1}\ \text{WG}\\ \text{ST r x}\\ \text{INV}_{L1}\ \text{DV}\end{array}\right\}\ \text{LK x}$
r=fetch_inc(x)	INC$_{L1}$ r x	FLU$_{L1}$ WG INV$_{L1}$ WG INC$_{L2}$ r x	$\left.\begin{array}{l}\text{FLU}_{L1}\ \text{DV}\\ \text{INV}_{L1}\ \text{WG}\\ \text{INC}_{L2}\ \text{r x}\\ \text{INV}_{L1}\ \text{DV}\end{array}\right\}\ \begin{array}{l}\text{LK x}\\ \text{LK}_{rmw}\end{array}$

Original scheme

message-passing fails

		DV (not remote)	DV (remote)
r=load(x)	LD r x	$\texttt{INV}_{L1}$ WG LD r x	$\left.\begin{array}{l}\texttt{FLU}_{L1}\ \texttt{DV}\\ \texttt{INV}_{L1}\ \texttt{WG}\\ \texttt{LD r x}\end{array}\right\}$ LK x
store(x,r)	ST r x	$\texttt{FLU}_{L1}$ WG ST r x	$\left.\begin{array}{l}\texttt{FLU}_{L1}\ \texttt{WG}\\ \texttt{ST r x}\\ \texttt{INV}_{L1}\ \texttt{DV}\end{array}\right\}$ LK x
r=fetch_inc(x)	$\texttt{INC}_{L1}$ r x	$\texttt{FLU}_{L1}$ WG $\texttt{INV}_{L1}$ WG $\texttt{INC}_{L2}$ r x	$\left.\begin{array}{l}\texttt{FLU}_{L1}\ \texttt{DV}\\ \texttt{INV}_{L1}\ \texttt{WG}\\ \texttt{INC}_{L2}\ \texttt{r x}\\ \texttt{INV}_{L1}\ \texttt{DV}\end{array}\right\}$ LK x $\texttt{LK}_{rmw}$

Original scheme

message-passing fails

		DV (not remote)	DV (remote)
r=load(x)	LD r x	INV$_{L1}$ WG LD r x	FLU$_{L1}$ DV INV$_{L1}$ WG LD r x } LK x
store(x,r)	ST r x	FLU$_{L1}$ WG T r x	FLU$_{L1}$ WG ST r x INV$_{L1}$ DV } LK x
r=fetch_inc(x)	INC$_{L1}$ r x	FLU$_{L1}$ WG INV$_{L1}$ WG INC$_{L2}$ r x	FLU$_{L1}$ DV INV$_{L1}$ WG INC$_{L2}$ r x INV$_{L1}$ DV } LK x LK$_{rmw}$

RMW atomicity fails

Original scheme

unnecessary cacheline stalling

message-passing fails

RMW atomicity fails

		DV (not remote)	DV (remote)
r=load(x)	LD r x	INV_{L1} WG LD r x	$\left.\begin{array}{l}\text{FLU}_{L1}\ \text{DV}\\ \text{INV}_{L1}\ \text{WG}\\ \text{LD r x}\end{array}\right\}$ LK x
store(x,r)	ST r x	FLU_{L1} WG T r x	$\left.\begin{array}{l}\text{FLU}_{L1}\ \text{WG}\\ \text{ST r x}\\ \text{INV}_{L1}\ \text{DV}\end{array}\right\}$ LK x
r=fetch_inc(x)	INC_{L1} r x	FLU_{L1} WG INV_{L1} WG INC_{L2} r x	$\left.\begin{array}{l}\text{FLU}_{L1}\ \text{DV}\\ \text{INV}_{L1}\ \text{WG}\\ \text{INC}_{L2}\ \text{r x}\\ \text{INV}_{L1}\ \text{DV}\end{array}\right\}$ LK x LK_{rmw}

Revised scheme

	na or WG	DV (not remote)	DV (remote)
r=load(x)	LD r x	$\begin{array}{l}\text{LD r x}\\ \text{INV}_{L1}\ \text{WG}\end{array}$	$\begin{array}{l}\text{LD r x}\\ \text{FLU}_{L1}\ \text{DV}\\ \text{INV}_{L1}\ \text{WG}\end{array}$
store(x,r)	ST r x	$\begin{array}{l}\text{FLU}_{L1}\ \text{WG}\\ \text{ST r x}\end{array}$	$\left.\begin{array}{l}\text{FLU}_{L1}\ \text{WG}\\ \text{INV}_{L1}\ \text{DV}\\ \text{ST r x}\end{array}\right\}\text{LK}_{rmw}$
r=fetch_inc(x)	INC_{L1} r x	$\begin{array}{l}\text{FLU}_{L1}\ \text{WG}\\ \text{INC}_{L2}\ \text{r x}\\ \text{INV}_{L1}\ \text{WG}\end{array}$	$\left.\begin{array}{l}\text{FLU}_{L1}\ \text{WG}\\ \text{INV}_{L1}\ \text{DV}\\ \text{INC}_{L2}\ \text{r x}\\ \text{FLU}_{L1}\ \text{DV}\\ \text{INV}_{L1}\ \text{WG}\end{array}\right\}\text{LK}_{rmw}$

Original scheme

	na or WG	DV (not remote)	DV (remote)
r=load(x)	LD r x	INV$_{L1}$ WG LD r x	$\left.\begin{array}{l}\text{FLU}_{L1}\ \text{DV}\\ \text{INV}_{L1}\ \text{WG}\\ \text{LD r x}\end{array}\right\}$ LK x
store(x,r)	ST r x	FLU$_{L1}$ WG ST r x	$\left.\begin{array}{l}\text{FLU}_{L1}\ \text{WG}\\ \text{ST r x}\\ \text{INV}_{L1}\ \text{DV}\end{array}\right\}$ LK x
r=fetch_inc(x)	INC$_{L1}$ r x	FLU$_{L1}$ WG INV$_{L1}$ WG INC$_{L2}$ r x	$\left.\begin{array}{l}\text{FLU}_{L1}\ \text{DV}\\ \text{INV}_{L1}\ \text{WG}\\ \text{INC}_{L2}\ \text{r x}\\ \text{INV}_{L1}\ \text{DV}\end{array}\right\}$ LK x LK$_{rmw}$

Proof of correctness

- Theorem stated in Isabelle, proved by hand.

Summary

- **Remote-scope promotion** is a GPU programming extension from **AMD** for efficient **work-stealing**
- We **formalised** the design (at SW and HW level). This led to a **corrected** and **improved** implementation.
- Formalise **early** in the design process!