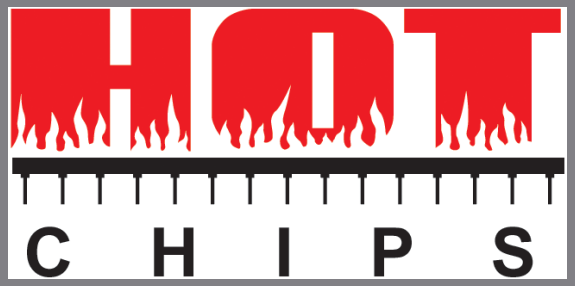


HSA MEMORY MODEL

HOT CHIPS TUTORIAL - AUGUST 2013

BENEDICT R GASTER
WWW.QUALCOMM.COM

OUTLINE

- HSA Memory Model
- OpenCL 2.0
  - Has a memory model too
- Obstruction-free bounded deques
  - An example using the HSA memory model

HSA MEMORY MODEL

TYPES OF MODELS

- Shared memory computers and programming languages, divide complexity into models:
  1. Memory model specifies safety
     - e.g. can a work-item prevent others from progressing?
     - This is what this section of the tutorial will focus on
  2. Execution model specifies liveness
     - Described in Ben Sander's tutorial section on HSAIL
     - e.g. can a work-item prevent others from progressing
  3. Performance model specifies the big picture
     - e.g. caches or branch divergence
     - Specific to particular implementations and outside the scope of today's tutorial

THE PROBLEM

- Assume all locations (a, b, …) are initialized to 0
- What are the values of $s2 and $s4 after execution?

Work-item 0

```
mov_u32 $s1, 1 ;
st_global_u32 $s1, [&a] ;
ld_global_u32 $s2, [&b] ;
```

```
*a = 1;
int x = *b;
```

Work-item 1

```
mov_u32 $s3, 1 ;
st_global_u32 $s3, [&b] ;
ld_global_u32 $s4, [&a] ;
```

```
*b = 1;
int y = *a;
```

initially *a = 0 && *b = 0

THE SOLUTION

- The memory model tells us:
  - Defines the visibility of writes to memory at any given point
  - Provides us with a set of possible executions

WHAT MAKES A GOOD MEMORY MODEL*

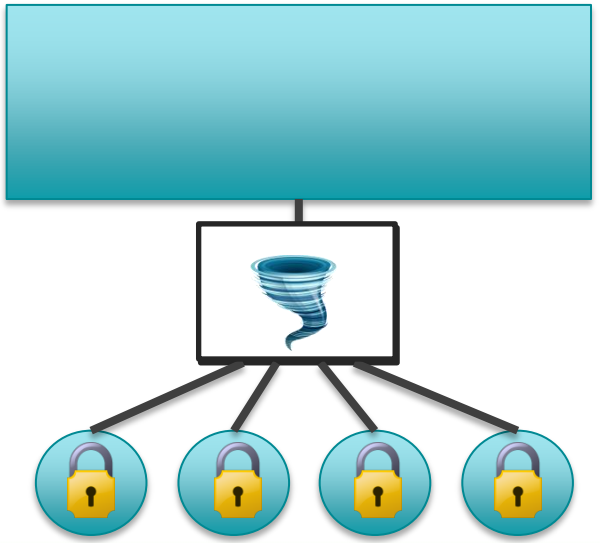
- Programmability ; A good model should make it (relatively) easy to write multi-work-item programs. The model should be intuitive to most users, even to those who have not read the details
- Performance ; A good model should facilitate high-performance implementations at reasonable power, cost, etc. It should give implementers broad latitude in options
- Portability ; A good model would be adopted widely or at least provide backward compatibility or the ability to translate among models

* S. V. Adve. Designing Memory Consistency Models for Shared-Memory Multiprocessors. PhD thesis, Computer Sciences Department, University of Wisconsin–Madison, Nov. 1993.

SEQUENTIAL CONSISTENCY (SC)*

- Axiomatic Definition
  - A single processor (core) sequential if “the result of an execution is the same as if the operations had been executed in the order specified by the program.”
  - A multiprocessor sequentially consistent if “the result of any execution is the same as if the operations of all processors (cores) were executed in some sequential order, and the operations of each individual processor (core) appear in this sequence in the order specified by its program.”

- But HW/Compiler actually implements more relaxed models, e.g. ARMv7



* L. Lamport. How to Make a Multiprocessor Computer that Correctly Executes Multiprocessor Programs. *IEEE Transactions on Computers*, C-28(9):690–91, Sept. 1979.

SEQUENTIAL CONSISTENCY (SC)

Work-item 0

```
mov_u32 $s1, 1 ;
st_global_u32 $s1, [&a] ;
ld_global_u32 $s2, [&b] ;
```

Work-item 1

```
mov_u32 $s3, 1 ;
st_global_u32 $s3, [&b] ;
ld_global_u32 $s4, [&a] ;
```

```
mov_u32 $s1, 1 ;
mov_u32 $s3, 1;
st_global_u32 $s1, [&a] ;
ld_global_u32 $s2, [&b] ;
st_global_u32 $s3, [&b] ;
ld_global_u32 $s4, [&a] ;
```

$s2 = 0 && $s4 = 1

BUT WHAT ABOUT ACTUAL HARDWARE

- Sequential consistency is (reasonably) easy to understand, but limits optimizations that the compiler and hardware can perform
- Many modern processors implement many reordering optimizations
  - Store buffers (TSO*), work-items can see their own stores early
  - Reorder buffers (XC*), work-items can see other work-items store early

*TSO – Total Store Order as implemented by Sparc and x86
*XC – Relaxed Consistency model, e.g. ARMv7, Power7, and Adreno

RELAXED CONSISTENCY (XC)

Work-item 0

```
mov_u32 $s1, 1 ;
st_global_u32 $s1, [&a] ;
ld_global_u32 $s2, [&b] ;
```

Work-item 1

```
mov_u32 $s3, 1 ;
st_global_u32 $s3, [&b] ;
ld_global_u32 $s4, [&a] ;
```

```
mov_u32 $s1, 1 ;
mov_u32 $s3, 1;
ld_global_u32 $s2, [&b] ;
ld_global_u32 $s4, [&a] ;
st_global_u32 $s1, [&a] ;
st_global_u32 $s3, [&b] ;
```

$s2 = 0 && $s4 = 0

WHAT ARE OUR 3 Ps?

- Programmability ; XC is really pretty hard for the programmer to reason about what will be visible when
  - many memory model experts have been known to get it wrong!
- Performance ; XC is good for performance, the hardware (compiler) is free to reorder many loads and stores, opening the door for performance and power enhancements
- Portability ; XC is very portable as it places very little constraints

MY CHILDREN AND COMPUTER ARCHITECTS ALL WANT

- **To have their cake and eat it!**

HSA Provides: The ability to enable programmers to reason with (relatively) intuitive model of SC, while still achieving the benefits of XC!

SEQUENTIAL CONSISTENCY FOR DRF*

- HSA, following Java, C++11, and OpenCL 2.0 adopts SC for Data Race Free (DRF)
  - plus some new capabilities !
- (Informally) A data race occurs when two (or more) work-items access the same memory location such that:
  - At least one of the accesses is a WRITE
  - There are no intervening synchronization operations
- SC for DRF asks:
  - Programmers to ensure programs are DRF under SC
  - Implementers to ensure that all executions of DRF programs on the relaxed model are also SC executions

*S. V. Adve and M. D. Hill. Weak Ordering—A New Definition. In Proceedings of the 17th Annual International Symposium on Computer Architecture, pp. 2–14, May 1990

HSA SUPPORTS RELEASE CONSISTENCY

- HSA's memory model is based on RC_{SC}:
  - All ld_acq and st_rel are SC
    - Means coherence on all ld_acq and st_rel to a single address.
    - All ld_acq and st_rel are program ordered per work-item (actually: sequence-order by language constraints)
- Similar model adopted by ARMv8
- HSA extends RC_{SC} to SC for HRF*, to access the full capabilities of modern heterogeneous systems, containing CPUs, GPUs, and DSPs, for example.

*Sequential Consistency for Heterogeneous-Race-Free Programmer-centric Memory Models for Heterogeneous Platforms. D. R. Hower, Beckman, B. R. Gaster, B. Hechtman, M D. Hill, S. K. Reinhart, and D. Wood. MSPC'13.

MAKING RELAXED CONSISTENCY WORK

Work-item 0

```
mov_u32 $s1, 1 ;
st_global_u32_rel $s1, [&a] ;
ld_global_u32_acq $s2, [&b] ;
```

Work-item 1

```
mov_u32 $s3, 1 ;
st_global_u32_rel $s3, [&b] ;
ld_global_u32_acq $s4, [&a] ;
```

```
mov_u32 $s1, 1 ;
mov_u32 $s3, 1;
st_global_u32_rel $s1, [&a] ;
ld_global_u32_acq $s2, [&b] ;
st_global_u32_rel $s3, [&b] ;
ld_global_u32_acq $s4, [&a] ;
```

$s2 = 0 && $s4 = 1

SEQUENTIAL CONSISTENCY FOR DRF

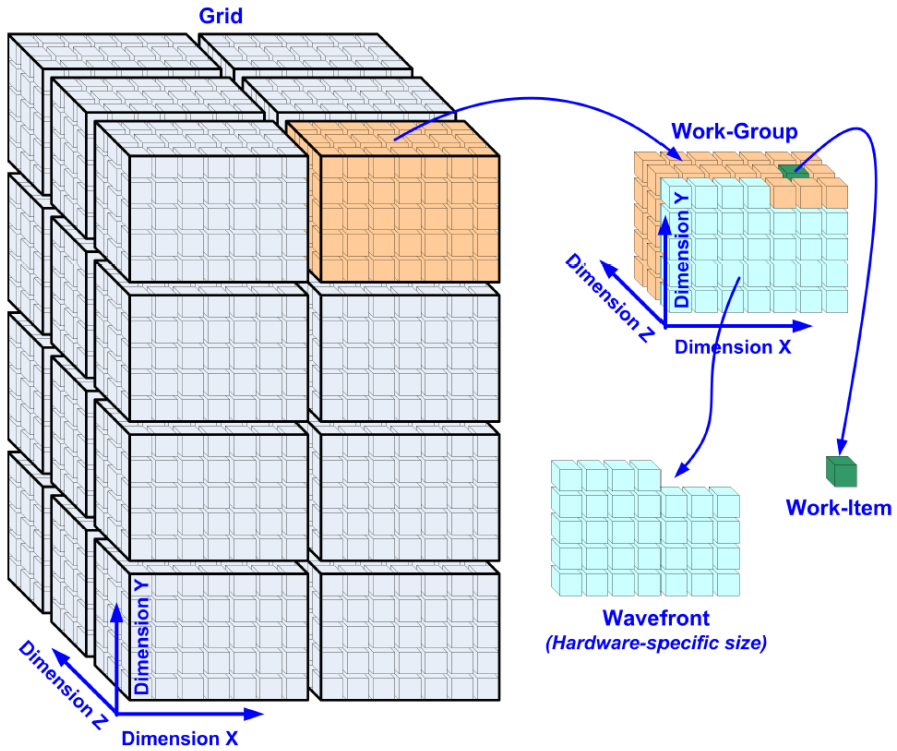
- Two memory accesses participate in a **data race** if they
  - access the same location
  - at least one access is a store
  - can occur simultaneously
    - i.e. appear as adjacent operations in interleaving.

HSA: Not good enough!

- A program is **data-race-free** if no possible execution results in a data race.
- Sequential consistency for data-race-free programs
  - Avoid everything else

ALL ARE NOT EQUAL – OR SOME CAN SEE BETTER THAN OTHERS

- Remember the HSAIL Execution Model



platform scope

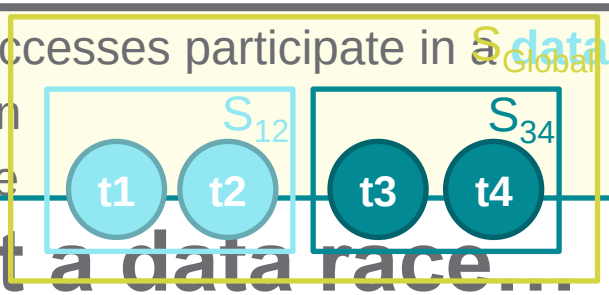
device scope

group scope

wave scope

DATA-RACE-FREE IS NOT ENOUGH

- Two ordinary memory accesses participate in a **data race** if they
  - Access same location
  - At least one is a store
  - Can occur sim



Not a data race...

Is it SC?

Well that depends

group #1-2

oup #3-4

```
   t1                              t2
st_global 1, [&X]
st_global_rel 0, [&flag]
                    atomic_cas_global_ar 1,
                    ...
                    st_global_rel 0, [&flag]
                                    atomic_cas_global_ar ,1 0, [&flag]
                                    ld (??), [&x]
```

visibility implied by causality?

SEQUENTIAL CONSISTENCY FOR HETEROGNEOUS-RACE-FREE

- Two memory accesses participate in a **heterogeneous** race if
  - access the same location
  - at least one access is a store
  - can occur simultaneously
    - i.e. appear as adjacent operations in interleaving.
  - Are not synchronized with “enough” scope
- A program is **heterogeneous**-race-free if no possible execution results in a heterogeneous race.
- Sequential consistency for heterogeneous-race-free programs
  - Avoid everything else

HSA HETEROGENEOUS RACE FREE

- **HRF0: Basic Scope Synchronization**
  - "enough" = both threads synchronize using ***identical*** scope
- Recall example:
  - Contains a heterogeneous race in HSA

HSA Conclusion:
This is bad. Don't do it.

Wo… group #3-4

t1

```
st_global 1, [&X]
st_global_rel_wg 0, [&flag]
...
```

Different Scope

t4

```
atomic_cas_global_ar_wg,1 0, [&flag]
ld (??), [&x]
```

HOW TO USE HSA WITH SCOPES

Want: For performance, use smallest scope possible

What is safe in HSA?

Use smallest scope that includes all producers/consumers of shared data

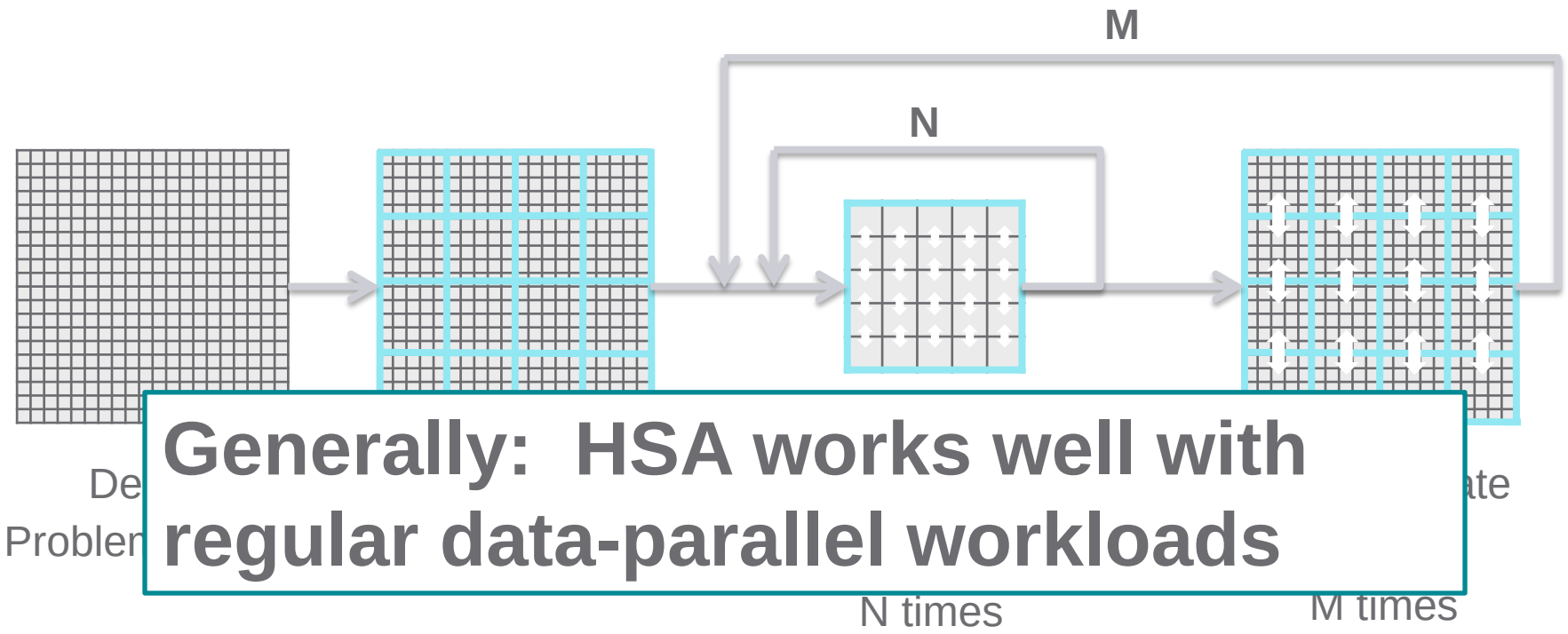
HSA Scope Selection Guideline

Is this a valid assumption?

Implication:

Producers/consumers must be known at synchronization time

REGULAR GPGPU WORKLOADS



Generally: HSA works well with regular data-parallel workloads

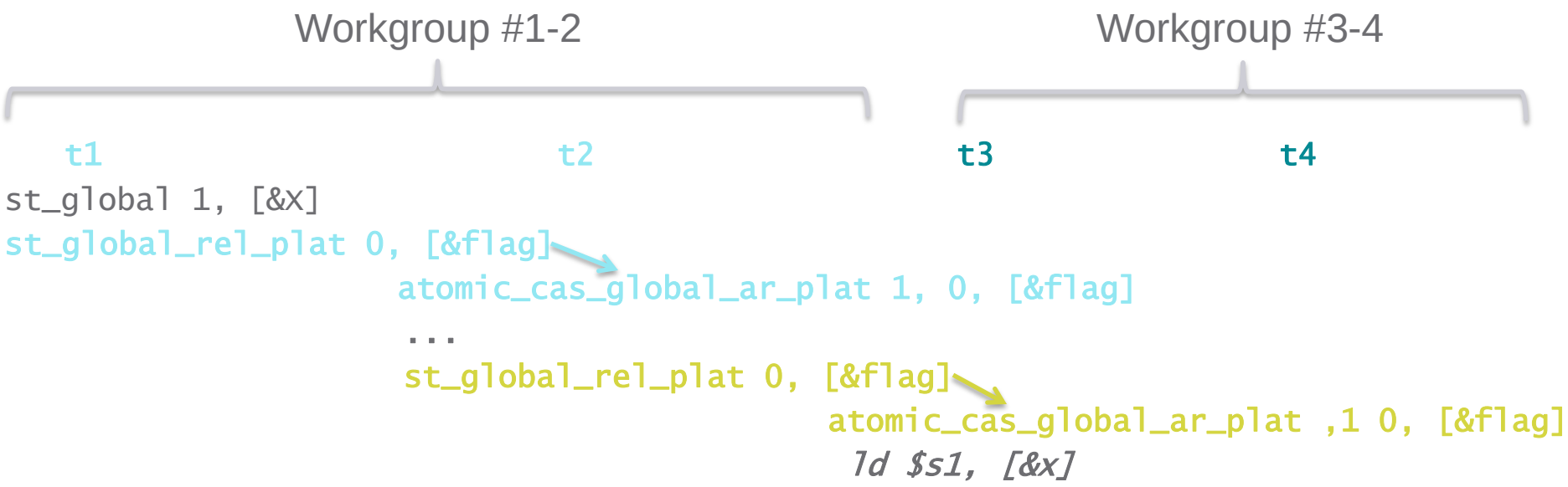
Well defined (regular) data partitioning +

Well defined (regular) synchronization pattern =

- *Producer/consumers are always known*

IRREGULAR WORKLOADS

- HSA: example is race
  - Must upgrade wg (workgroup) -> plat (platform)
- HSA memory model says:
  - ld $s1, [&x], will see value (1)!



OPENCL 2.0 HAS A MEMORY MODEL TOO

MAPPING ONTO HSA'S MEMORY MODEL

OPENCL 2.0 BACKGROUND

- Provisional specification released at SIGGRAPH'13, July 2013.
- Huge update to OpenCL to account for the evolving hardware landscape and emerging use cases (e.g. irregular work loads)
- Key features:
  - Shared virtual memory, including platform atomics
  - Formally defined memory model based on C11 plus support for scopes
    - Includes an extended set of C1X atomic operations
  - Generic address space, that subsumes global, local, and private
  - Device to device enqueue
  - Out-of-order device side queuing model
  - Backwards compatible with OpenCL 1.x

OPENCL 1.X MEMORY MODEL MAPPING

- It is straightforward to provide a mapping from OpenCL 1.x to the proposed model

OpenCL Operation	HSA Memory Model Operation
Load	ld_global_wg ld_group_wg
Store	st_global_wg st_group_wg
atomic_op	atomic_op_global_comp atomic_op_group_wg
barrier(…)	sync_wg

- OpenCL 1.x atomics are unordered and so map to atomic_op_X
- Mapping for fences not shown but straightforward

OPENCL 2.0 MEMORY MODEL MAPPING

OpenCL Operation	HSA Memory Model Operation
Load memory_order_relaxed	atomic_ld_[global | group]_scope
Store Memory_order_relaxed	atomic_st_[global | group]_scope
Load memory_order_acquire	ld_[global | group]_acq_scope
Load memory_order_seq_cst	fence_rel_scope ; ld_[global | group]_acq_scope
Store memory_order_release	st_[global | group]_rel_scope
Store Memory_order_seq_cst	st_[global | group]_rel_scope ; fence_acq_scope
memory_order_acq_rel	atomic_op_[global | group]_acq_rel_scope
memory_order_seq_cst	atomic_op_[global|group]_acq_rel_scope

OPENCL 2.0 MEMORY SCOPE MAPPING

OpenCL Scope	HSA Scope
memory_scope_work_group	_wg
memory_scope_device	_component
memory_scope_all_svm_devices	_platform

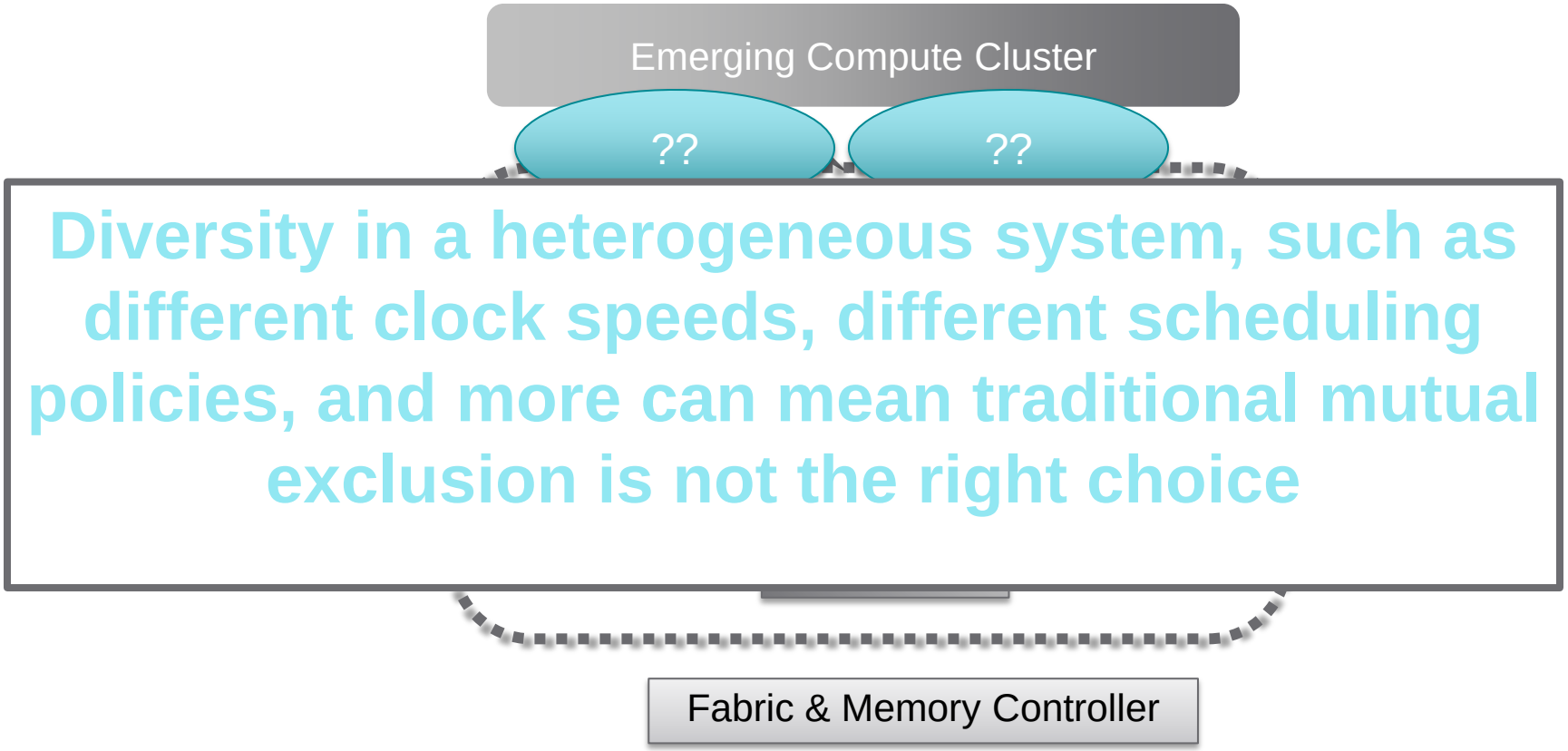
OBSTRUCTION-FREE BOUNDED DEQUES

AN EXAMPLE USING THE HSA MEMORY MODEL

CONCURRENT DATA-STRUCTURES

- Why do we need such a memory model in practice?
- One important application of memory consistency is in the development and use of concurrent data-structures
- In particular, there are a class data-structures implementations that provide non-blocking guarantees:
  - wait-free; An algorithm is wait-free if every operation has a bound on the number of steps the algorithm will take before the operation completes
    - In practice very hard to build efficient data-structures that meet this requirement
  - lock-free; An algorithm is lock-free if every if, given enough time, at least one thread of the work-items (or threads) makes progress
    - In practice lock-free algorithms are implemented by work-item cooperating with one enough to allow progress
  - Obstruction-free; An algorithm is obstruction-free if a work-item, running in isolation, can make progress

BUT WAY NOT JUST USE MUTUAL EXCLUSION?



CONCURRENT DATA-STRUCTURES

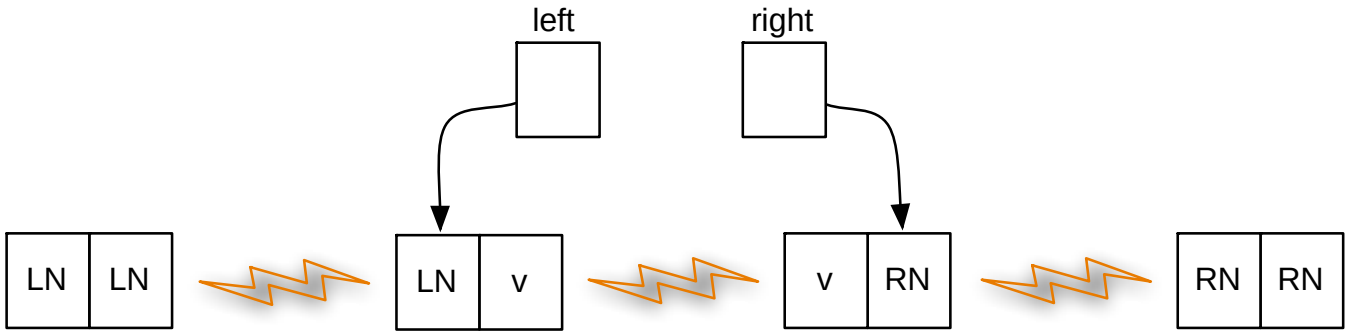
- Emerging heterogeneous compute clusters means we need:
  - To adapt existing concurrent data-structures
  - Developer new concurrent data-structures
- Lock based programming may still be useful but often these algorithms will need to be lock-free
- Of course, this is a key application of the HSA memory model
- To showcase this we highlight the development of a well known (HLM) obstruction-free deque*

*Herlihy, M. et al. 2003. Obstruction-free synchronization: double-ended queues as an example. (2003), 522–529.

HLM - OBSTRUCTION-FREE DEQUE

- Uses a fixed length circular queue
- At any given time, reading from left to right, the array will contain:
  - Zero or more left-null (LN) values
  - Zero or more dummy-null (DN) values
  - Zero or more right-null (RN) values
- At all times there must be:
  - At least two different nulls values
    - At least one LN or DN, and at least one DN or RN
- Memory consistency is required to allow multiple producers and multiple consumers, potentially happening in parallel from the left and right ends, to see changes from other work-items (HSA Components) and threads (HSA Agents)

HLM - OBSTRUCTION-FREE DEQUE



Key:

LN	– left null value
RN	– right null value
v	– value
left	– left hint index
right	– right hint index

C REPRESENTATION OF DEQUE

```
struct node {
   uint64_t type :  2;            //  null type (LN, RN, DN)
   uint64_t counter : 8 ;         // version counter to avoid ABA
   uint64_t value : 54 ;          // index value stored in queue
}
struct queue {
    unsigned int size;       // size of bounded buffer
    node * array;           // backing store for deque itself
}
```

HSAIL REPRESENTATION

- Allocate a deque in global memory using HSAIL

```
@deque_instance:
    align 64 global_u32  &size;
    align 8 global_u64 &array;
```

ORACLE

- Assume a function:

```
function &rcheck_oracle (arg_u32 %k, arg_u64 %left, arg_u64 %right) (arg_u64 %queue);
```

- Which given a deque
  - returns (%k) the position of the left most of RN
    - ld_global_acq used to read node from array
    - Makes one if necessary (i.e. if there are only LN or DN)
    - atomic_cas_global_ar, required to make new RN
  - returns (%left) the left node (i.e. the value to the left of the left most RN position)
  - returns (%right) the right node (i.e. the value at position (%k))

RIGHT POP

```
function &right_pop(arg_u32err, arg_u64 %result) (arg_u64 %deque) {
   // load queue address
   ld_arg_u64 $d0, [%deque];
@loop_forever:
   // setup and call right oracle to get next RN
   arg_u32 %k; arg_u64 %current; arg_u64 %next;
   call &rcheck_oracle(%queue) ;
   ld_arg_u32 $s0, [%k]; ld_arg_u64 $d1, [%current]; ld_arg_u64 $d2, [%next];
   // current.value($d5)
   shr_u64 $d5, $d1, 62;
   // current.counter($d6)
   and_u64 $d6, $d1,
   0x3FC0000000000000;
   shr_u64 $d6, $d6, 54;
// current.value($d7)
   and_u64 $d7, $d1, 0x3FFFFFFFFFFFFF;
// next.counter($d8)
   and_u64 $d8, $d2, 0x3FC0000000000000; shr_u64 $d8, $d8, 54;       brn @loop_forever ;
}
```

RIGHT POP – TEST FOR EMPTY

```
// current.type($d5) == LN || current.type($d5) == DN
 cmp_neq_b1_u64 $c0, $d5, LN; cmp_neq_b1_u64 $c1, $d5, DN;
 or_b1 $c0, $c0, $c1;
 cbr $c0, @not_empty ;
 // current node index (%deque($d0) + (%k($s1) - 1) * 16)
 add_u32 $s1, $s0, -1;   mul_u32 $s1, $s1,  16;   add_u32 $d3, $d0, $s0;
 ld_global_acq_u64 $d4, [$d3];
 cmp_neq_b1_u64 $c0, $d4, $d1;
 cbr $c0, @not_empty;
  st_arg_u32 EMPTY, [&err];            // deque empty so return EMPTY
  %ret
@not_empty:
```

RIGHT POP – TRY READ/REMOVE NODE

```
// $d9 = (RN, next.cnt+1, 0)
add_u64 $d8, $d8, 1;
shl_u64 $d9, RN, 62;
and_u64 $d8, $d8, $d9;

// cas(deq+k, next, node(RN, next.cnt+1, 0))
atomic_cas_global_ar_u64 $d9, [$s0],  $d2, $d9;
cmp_neq_u64 $c0, $d9, $d2;
cbr $c0, @cas_failed;

// $d9 = (RN, current.cnt+1, 0)
add_u64 $d6, $d6, 1;
shl_u64 $d9, RN, 62;
and_u64 $d9, $d6, $d9;

// cas(deq+(k-1), curr, node(RN, curr.cnt+1,0)
atomic_cas_global_ar_u64 $d9, [$s1], $d1, $d9;
cmp_neq_u64 $c0, $d9, $d1;
cbr $c0, @cas_failed;
st_arg_u32 SUCCESS, [&err];
st_arg_u64 $d7, [&value];
%ret
@cas_failed:
// loop back around and try again
```

TAKE AWAYS

- HSA provides a powerful and modern memory model
  - Based on the well know SC for DRF
  - Defined as Release Consistency
  - Extended with scopes as defined by HRF
- OpenCL 2.0 introduces a new memory model
  - Also based on SC for DRF
  - Also defined in terms of Release Consistency
  - Also Extended with scope as defined in HRF
  - Has a well defined mapping to HSA
- Concurrent algorithm development for emerging heterogeneous computing cluster can benefit from HSA and OpenCL 2.0 memory models