Good FIFOs Make Good Neighbors

ARM

Jonathan Beard
Staff Research Engineer
ARM Research

Twitter: @jonathan_beard

CPPNow 2017
Aspen, CO

Ground Rules

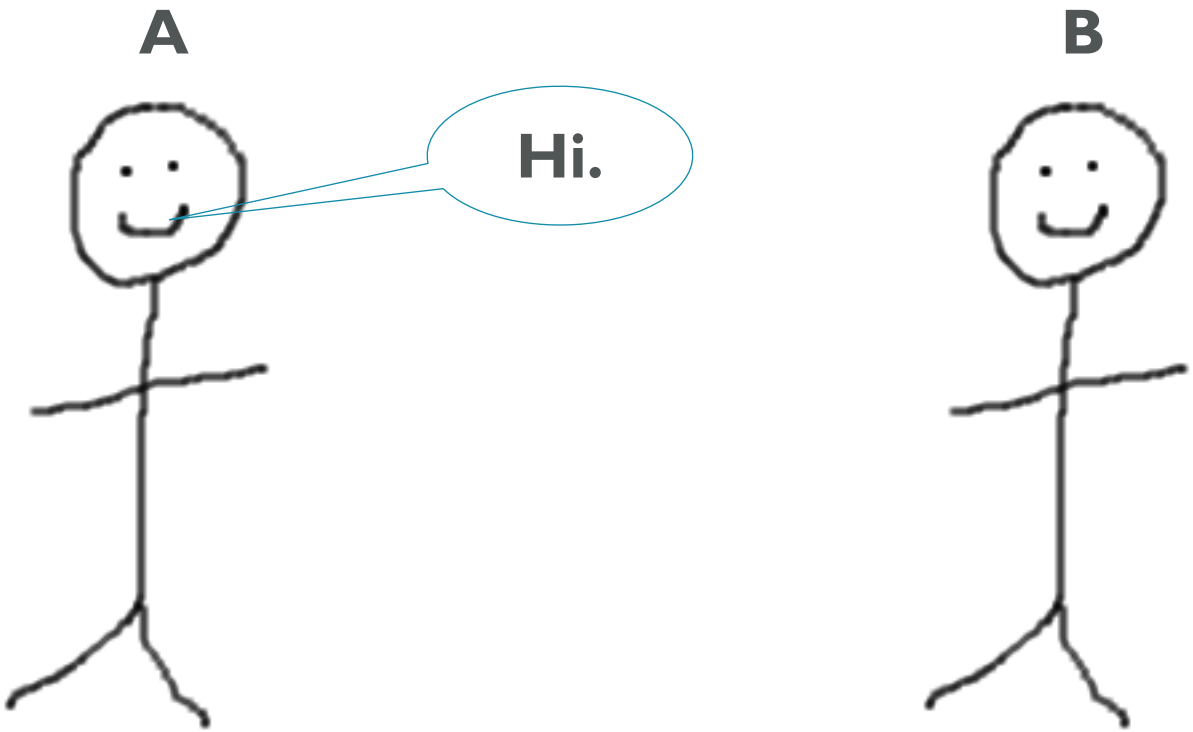
- We'll talk about software profiling on my workstation
- My workstation contains an Intel E52690v3 x2 (most benchmarks are run on this)
- We'll not be discussing ARM hardware…nor current research projects
- Yes, work derives from a research project, which I'm also not going to talk about
- Please ask questions
- Don't worry I'll ask questions too
- Hopefully we'll all have answers

What is a FIFO?

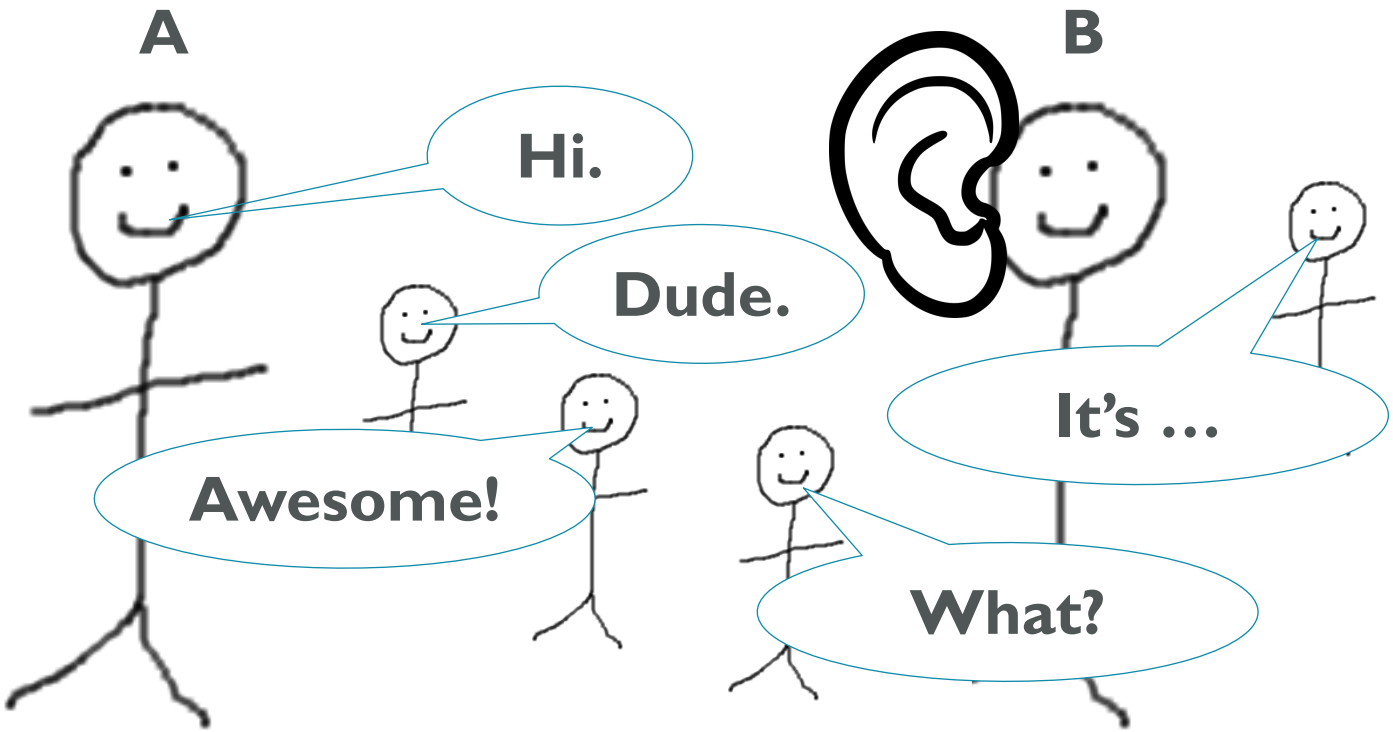
First-In, First-Out Queue,…geez, why do we have to have this slide?

Getting a message across

Getting a message across



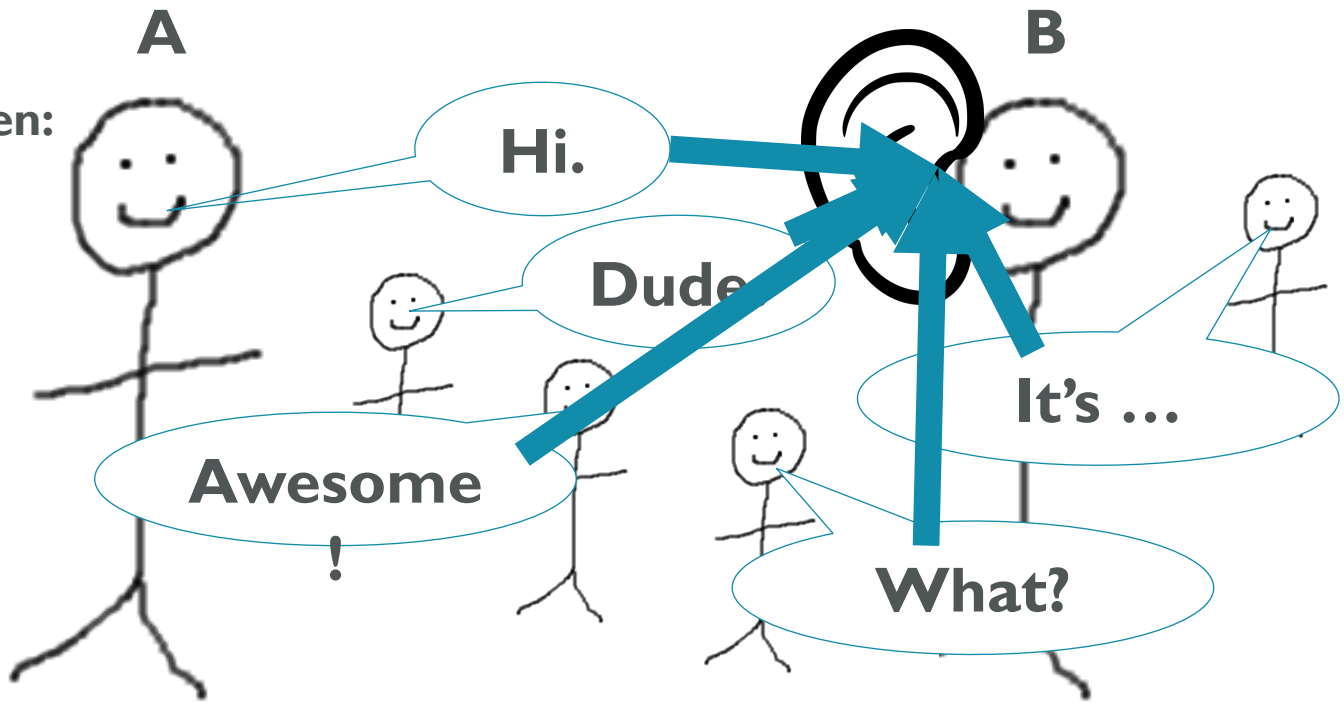
Getting a message across



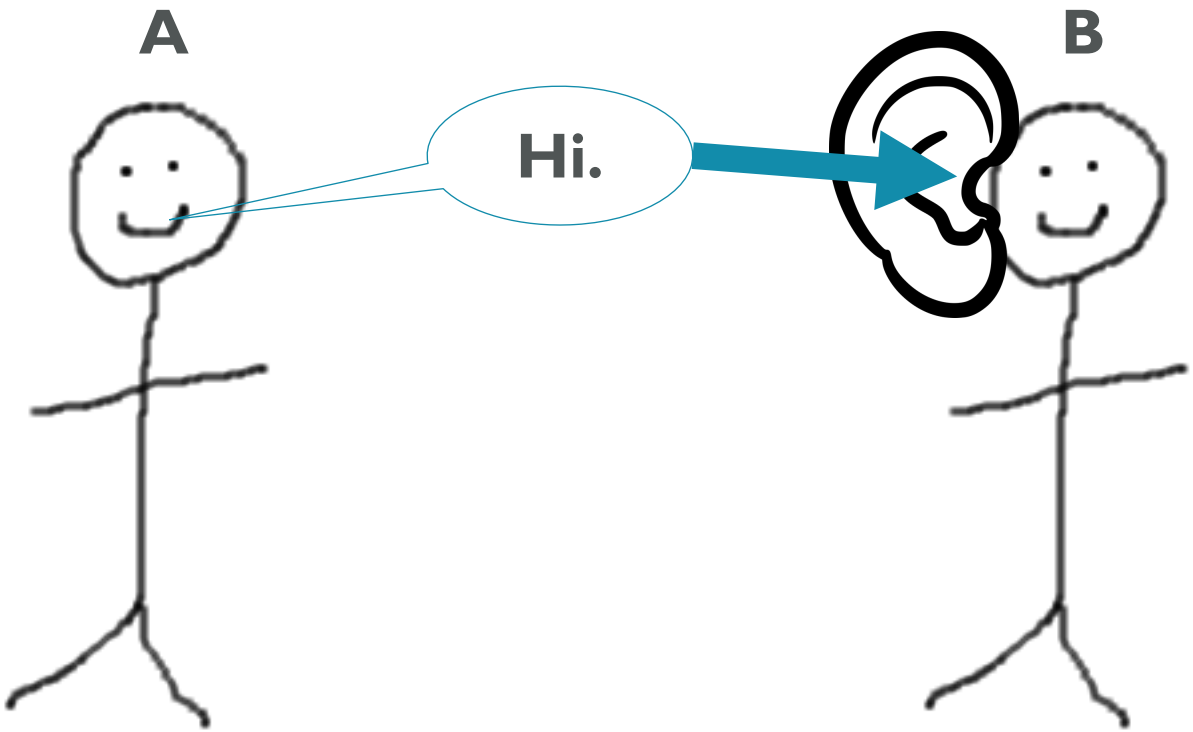
Information – Integrity Matters

Can't discern between:

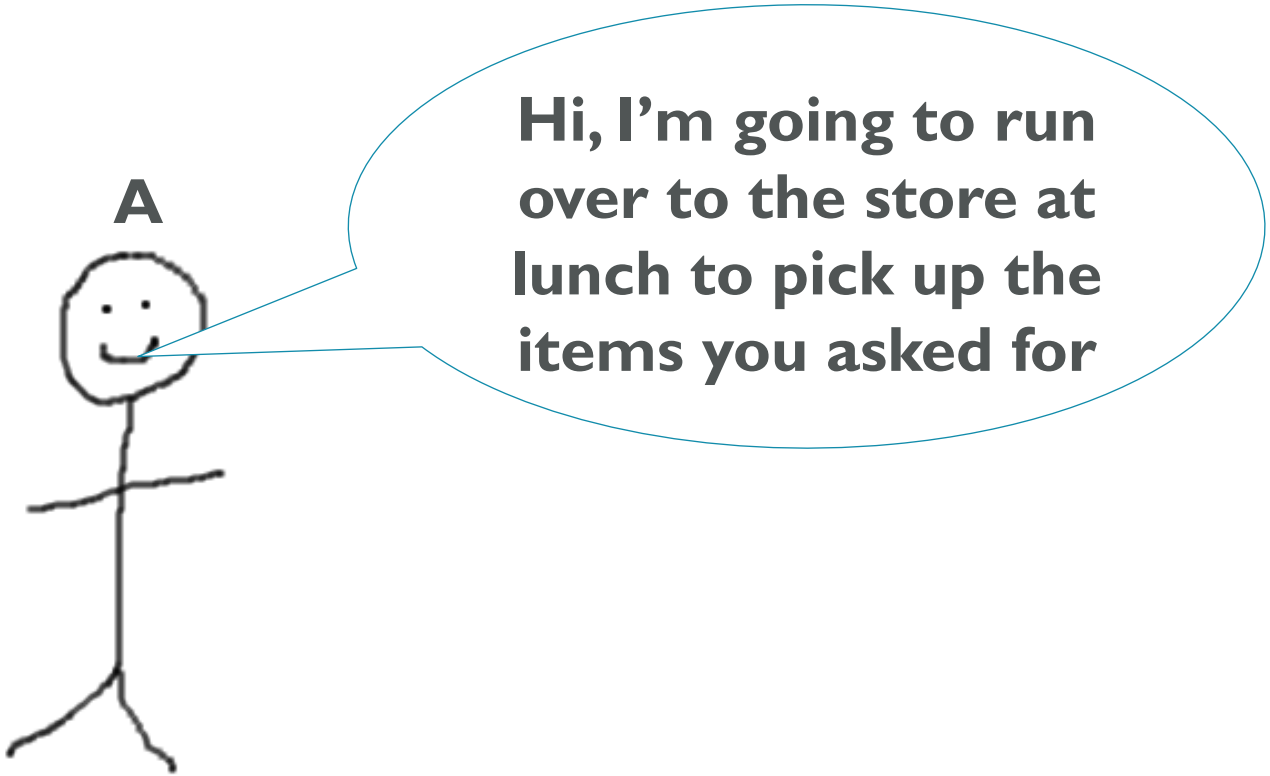
- Hi, Dude, Awesome, What, Its
- Dude, Hi, What, It's, Awesome
- …
- You get the idea



Problem…

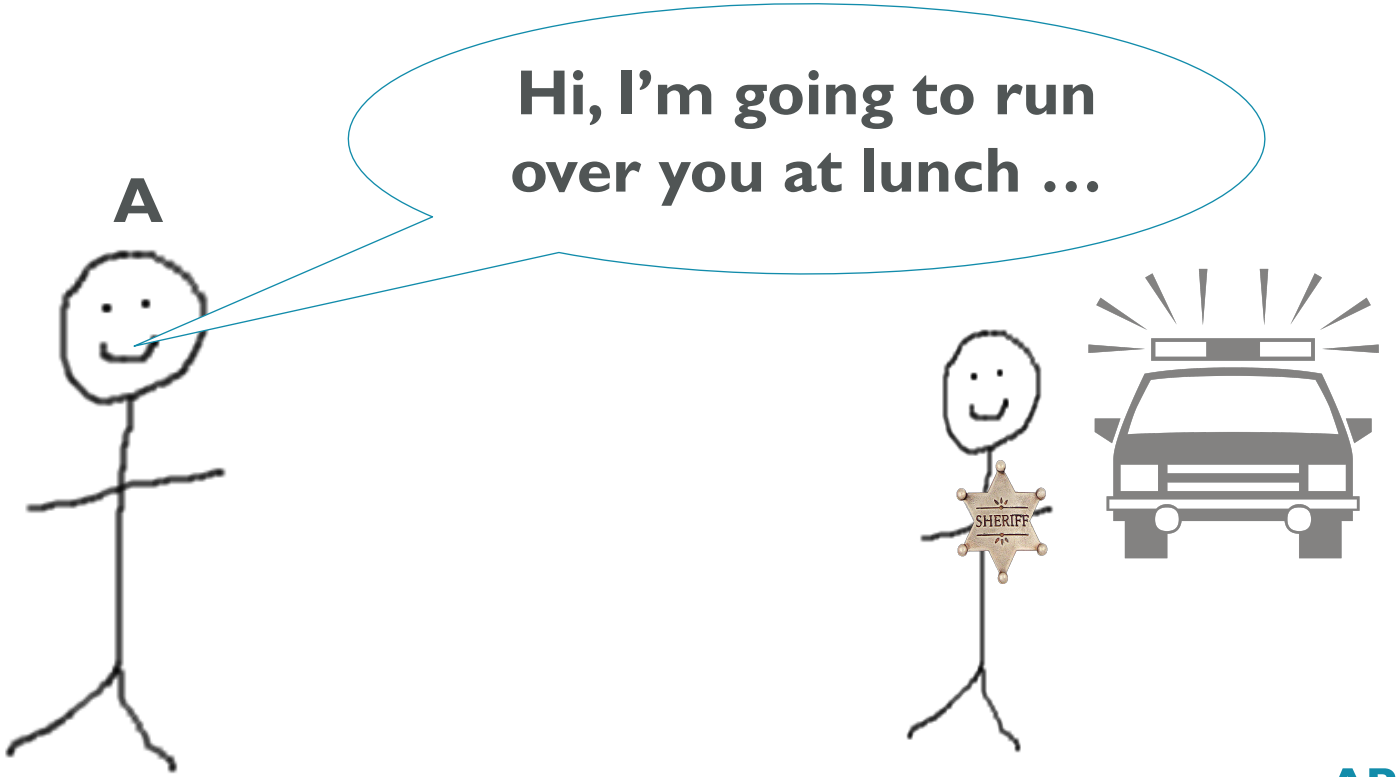


Information – Ordering Matters



Information – Ordering Matters

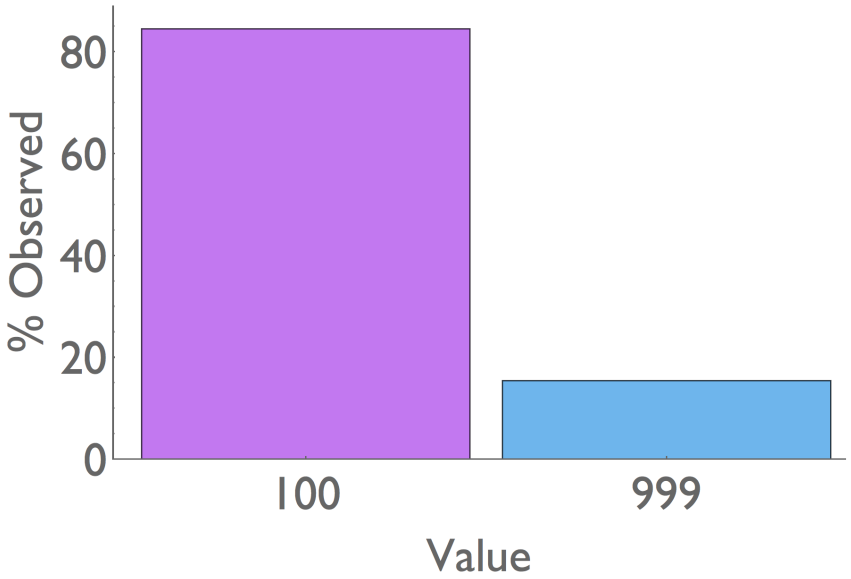
Information – Ordering Matters



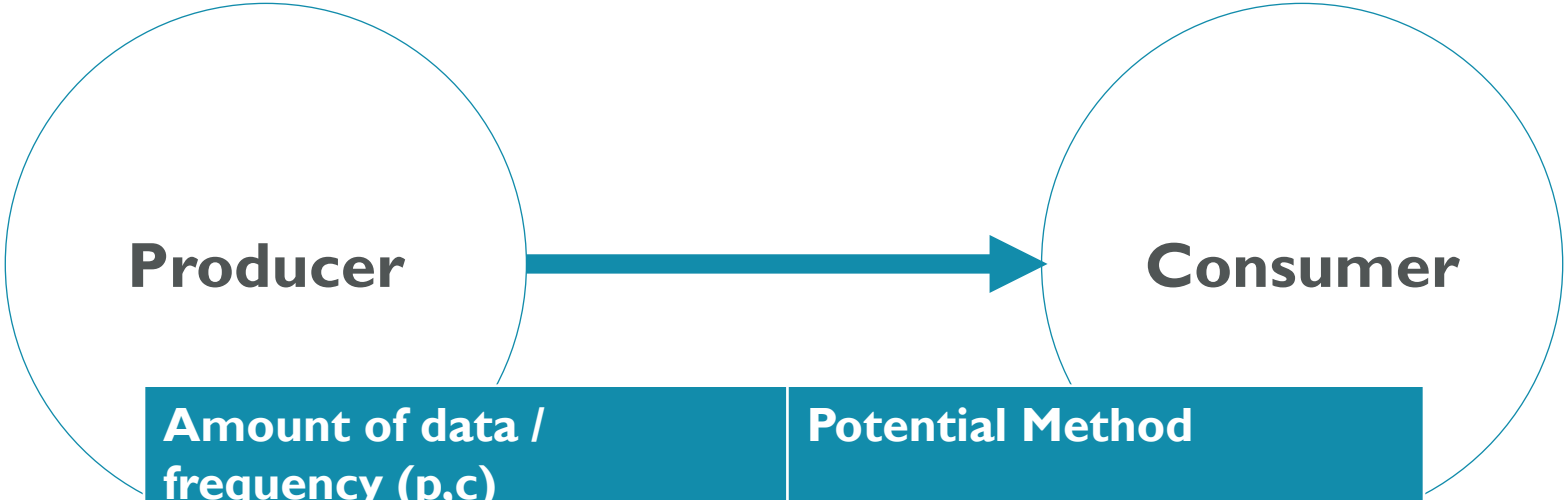
Obligatory Thread Ordering Example

```
static void producer( type_t &ref_test )
{
    std::this_thread::yield();
    ref_test = 5;
}

static void consumer(
     type_t &ref_test,
     type_t &ref_out )
{
    std::this_thread::yield();
    if( ref_test == 0 )
    {
        ref_out = 100;
    }
    else
    {
        ref_out = 999;
    }
}
```



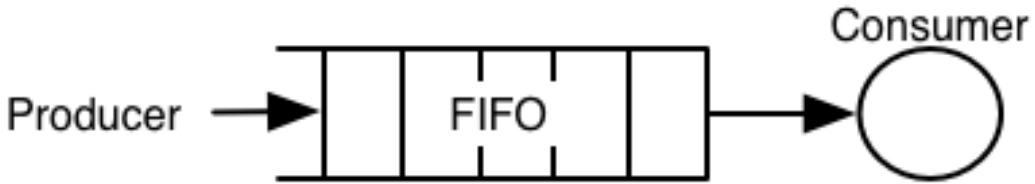
Thread Communication



Amount of data / frequency (p,c)	Potential Method
<64B, once	Atomic
<64B, multiple times	FIFO
>64B, once	Lock or CAS Data Structure
>64B, multiple times	FIFO + pointers + prefetch

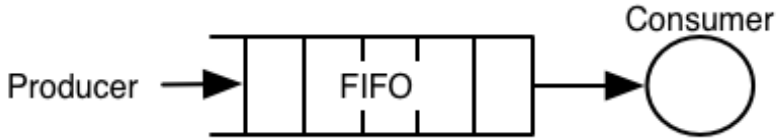
First-On First-Out (FIFO) Queue

- Simple concept, push messages
- Natural, it's how we communicate
- Hard to get right when on hairy edge…
  - Clock differences
  - Coherence
  - Non-uniform Latency
  - Queueing delay (yup, queuing delay for queue, ironic right?)

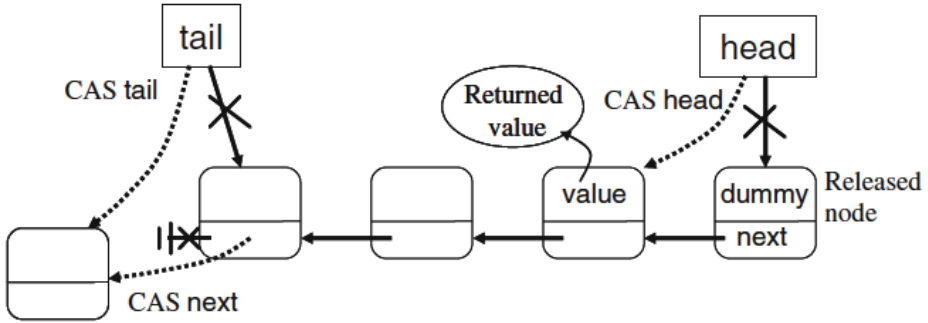


In Software

- FIFO..okay, that's simple right?
- Lockless / Locked (CAS vs. Spinlock generally)
- General Types:
  - Chunk of memory
  - Linked List



- One of the best: Singly linked list (Michael-Scott algo):

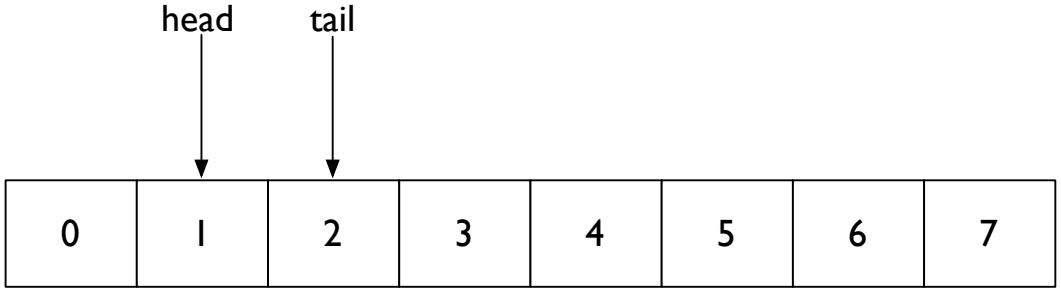


Types of FIFOs

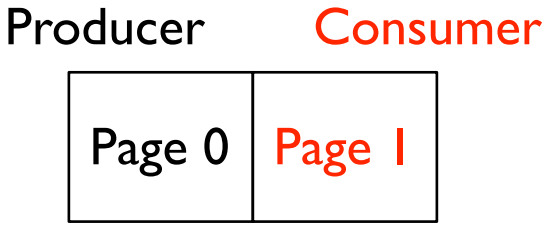
Chunk of Memory Types

In this category:

- Ring Buffer
- Virtual Memory Remap
- Bump Buffer

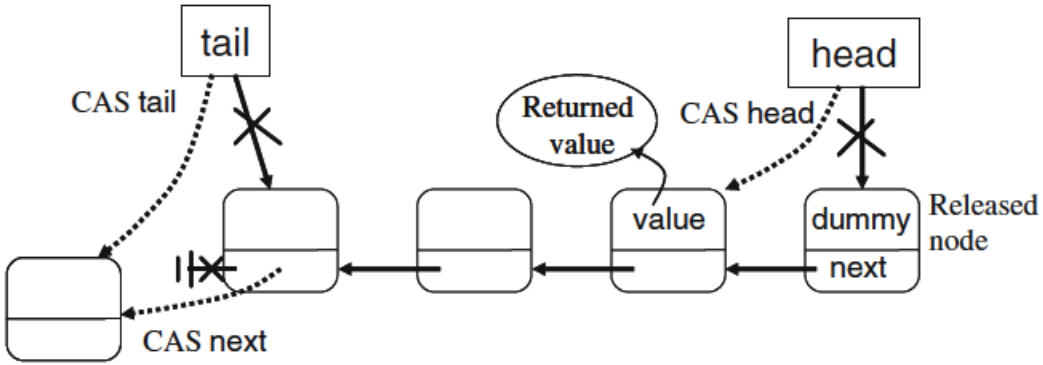


General idea, get data in memory from one thread to another as efficiently as possible



Linked List Types

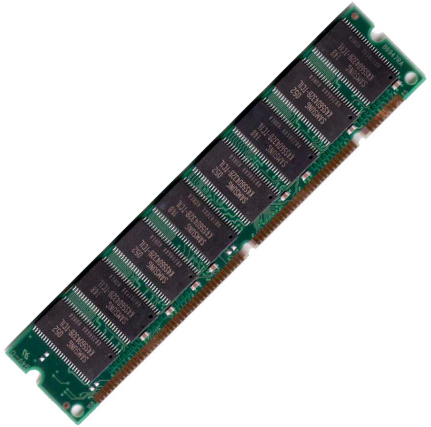
- Michael-Scott
  - Singly linked list
  - Double compare and swap to enqueue
  - Single compare and swap to dequeue
- Ladan-Mozes-Shavit
  - Innovation = doubly linked list
  - Reverse the enqueue and dequeue
  - Single CAS for both



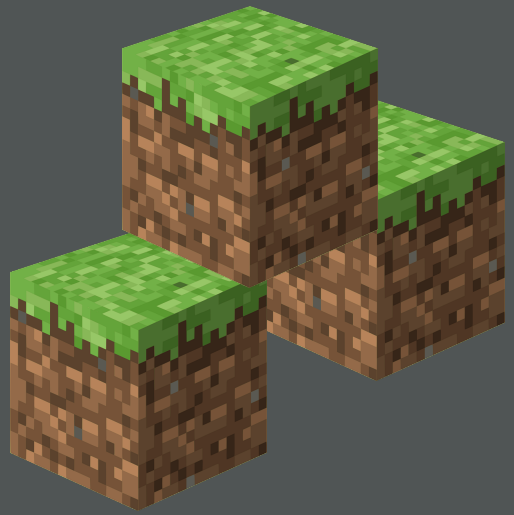
CAS can take upwards of 50 cycles so reducing the number is critical

Commonalities

- Coherence Traffic
  - Ordering
  - Amount
- Memory access
  - Data layouts
  - Virtual memory
  - Prefetch



Before we go further



Hardware Basics

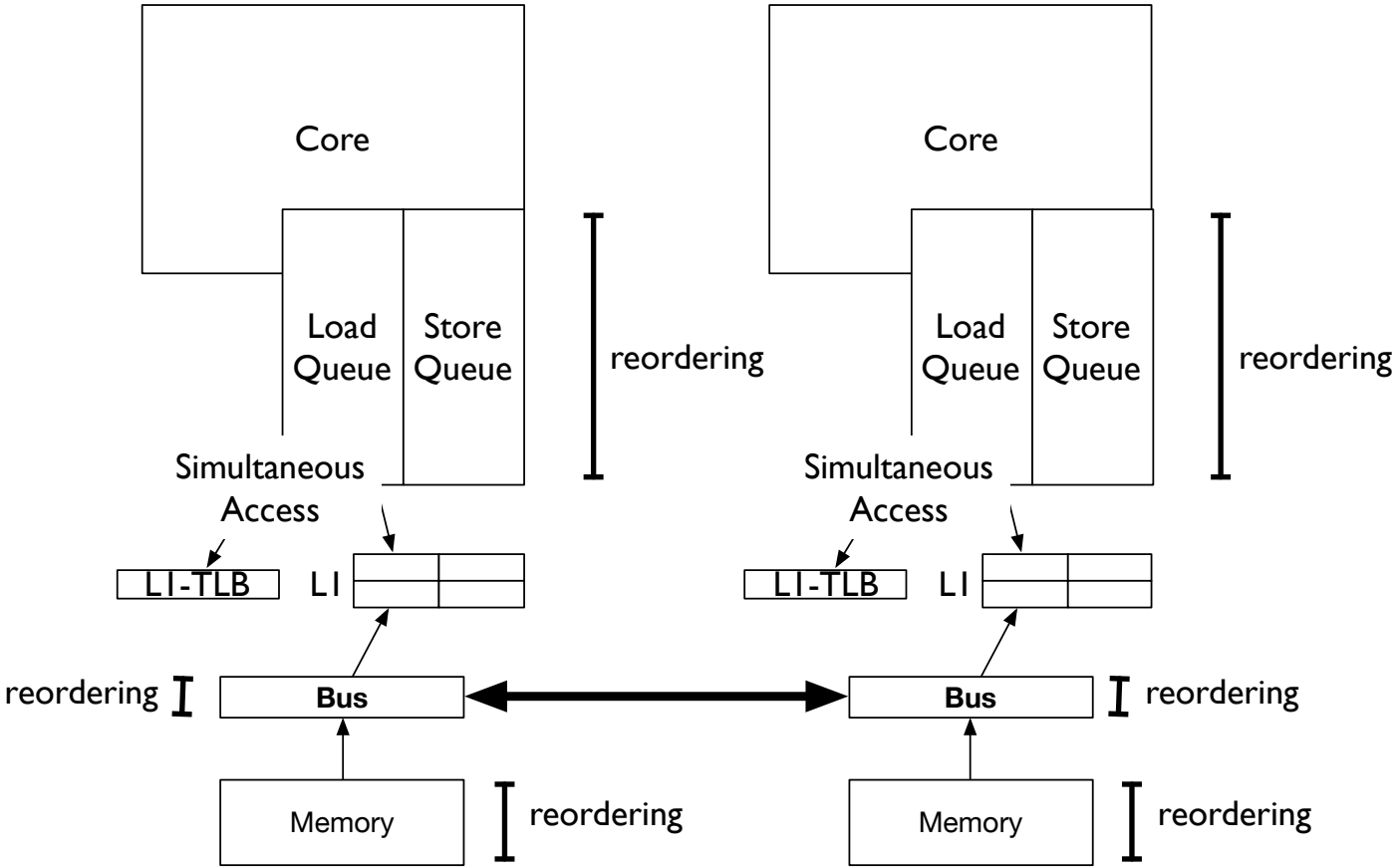
Quiz Time

- Which matters more for FIFO performance / correctness:
  - A. **Core**
  - B. **Interconnect**
  - C. **Memory**

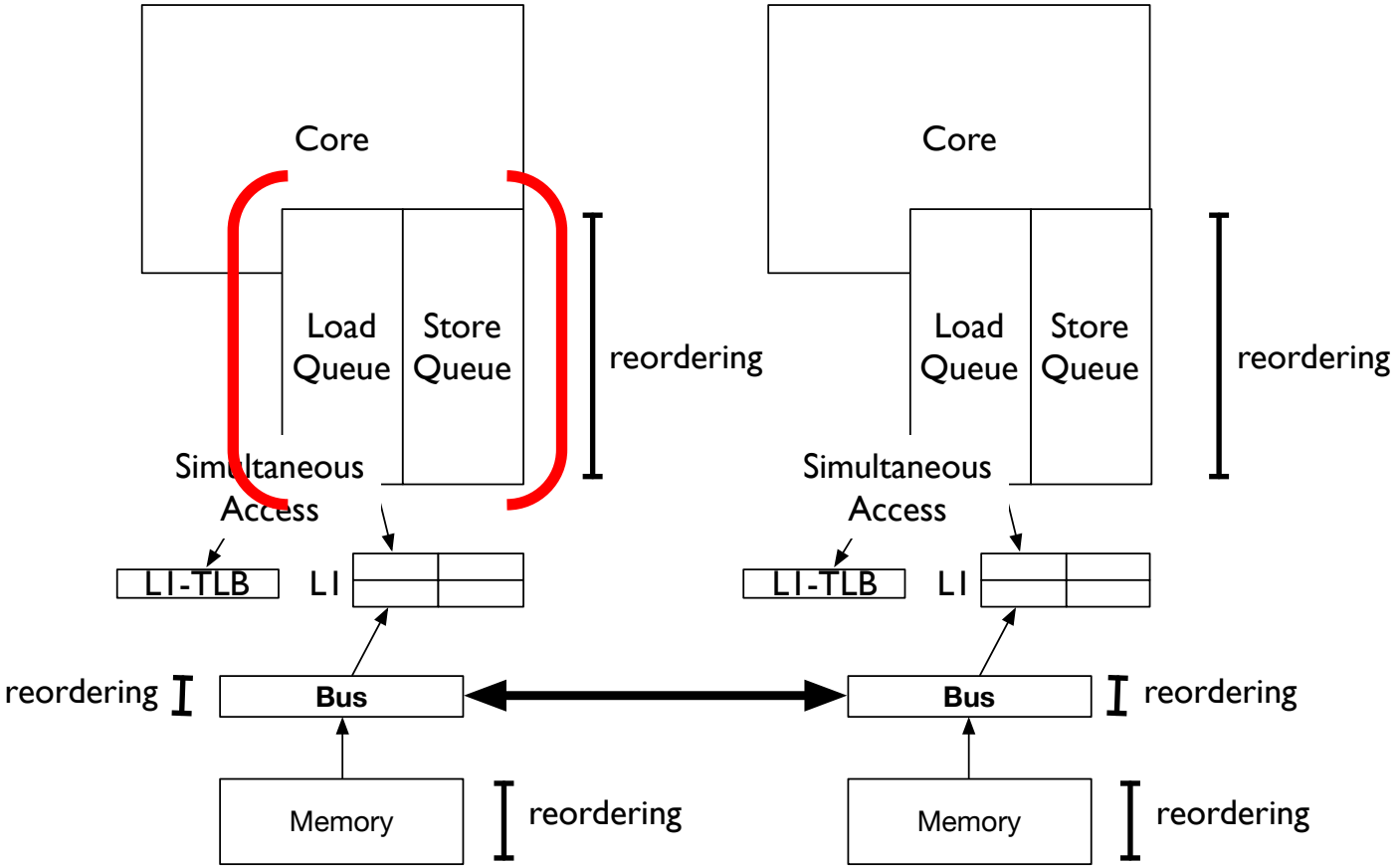
Quiz Time

- Which matters more for FIFO performance / correctness:
  - A. **Core**
  - B. **Interconnect**
  - C. **Memory**

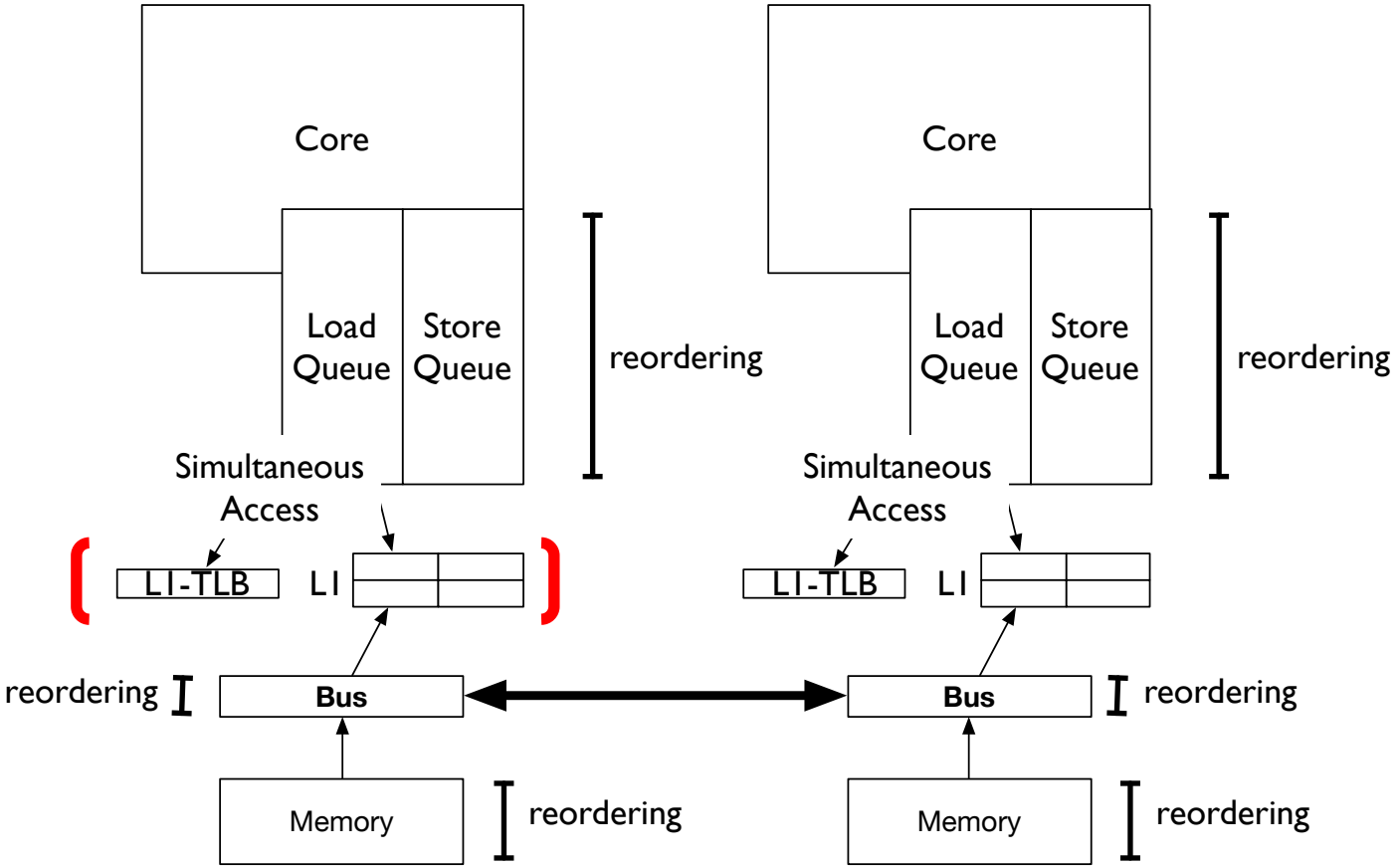
Hardware



Hardware

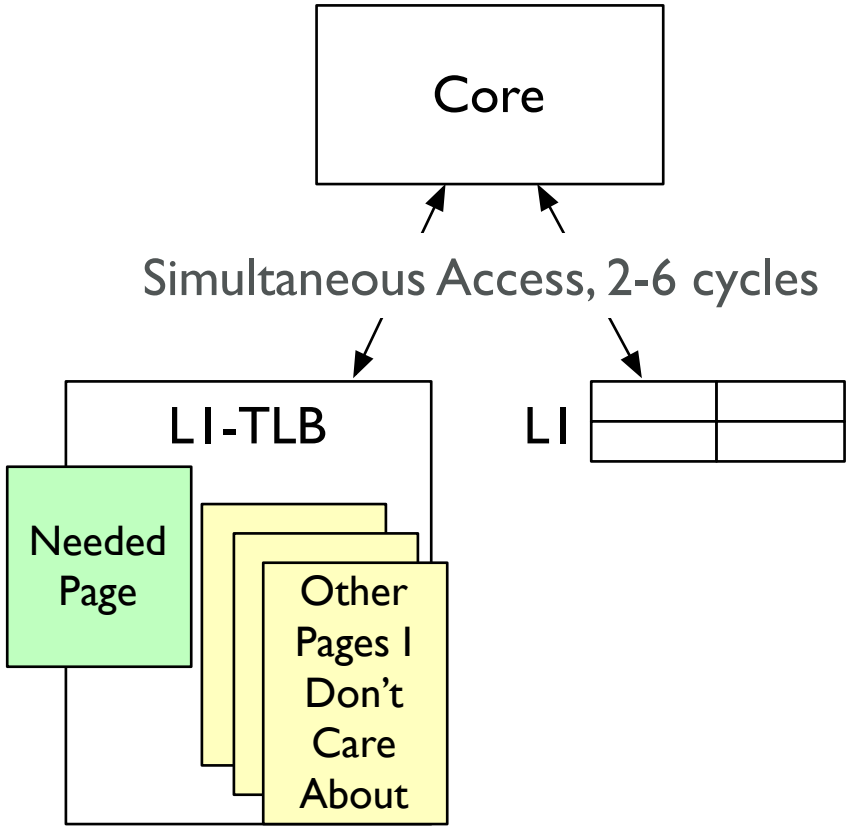


Hardware



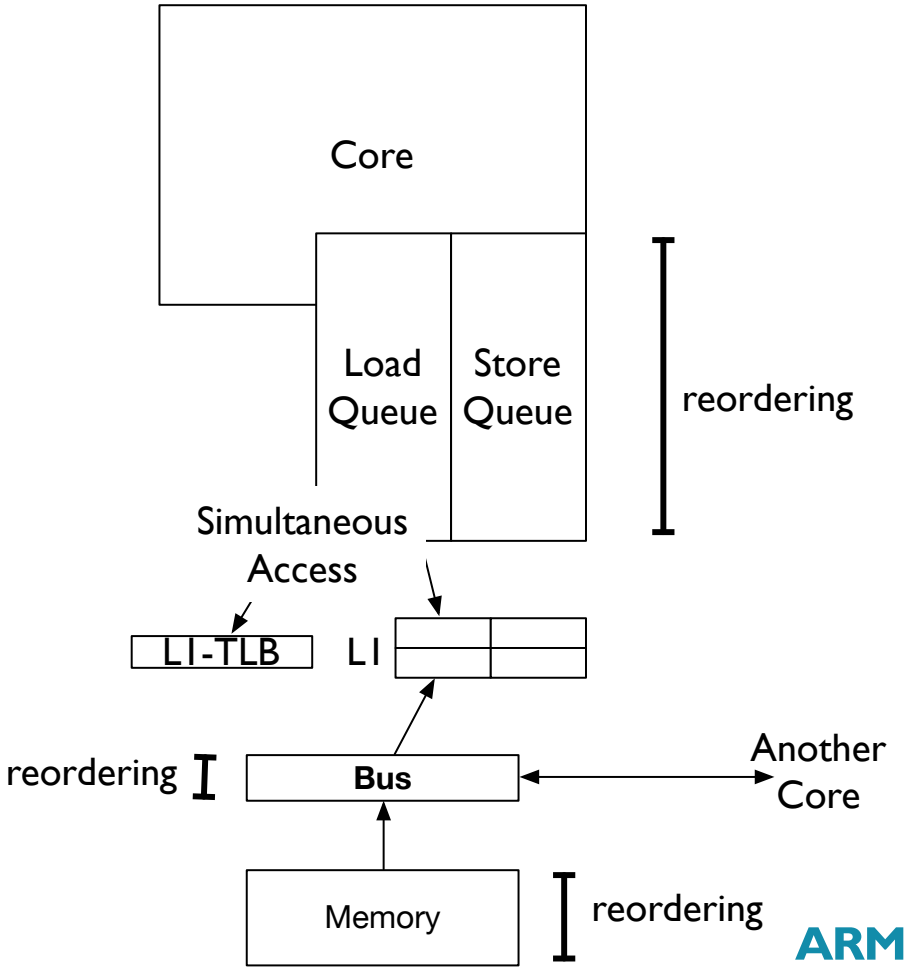
Hardware

- Most systems use page-based virtual memory.
- Pages come in a few select sizes to make it easier to build the hardware (e.g., 4K, 64K, 2M).
- On a translation miss, in L1-D/L2 we have 4 additional memory refs on average
- Each ref could take ~150ns to fill if not cached
- Extremely important on VM's (nested translation)



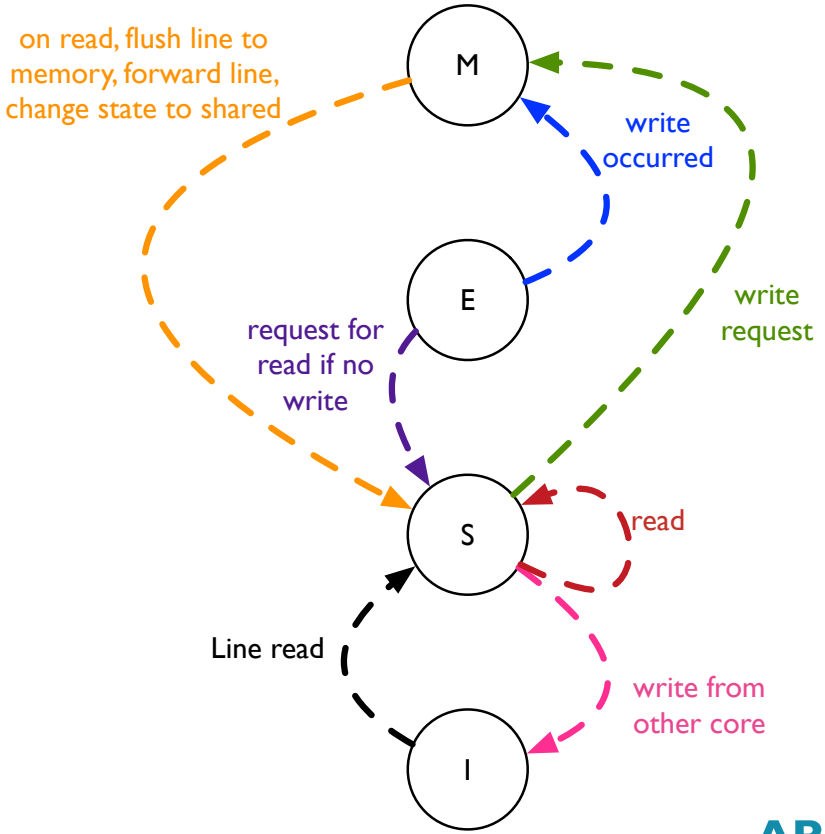
Hardware - Caches

- Lets just talk about L1-D attached to a bus
- Bus between core and CPU is generally 128-256 bits/cycle
- Between L1-D and bus the cache line is the unit of movement
- Before we can read/write a memory location we need to get it

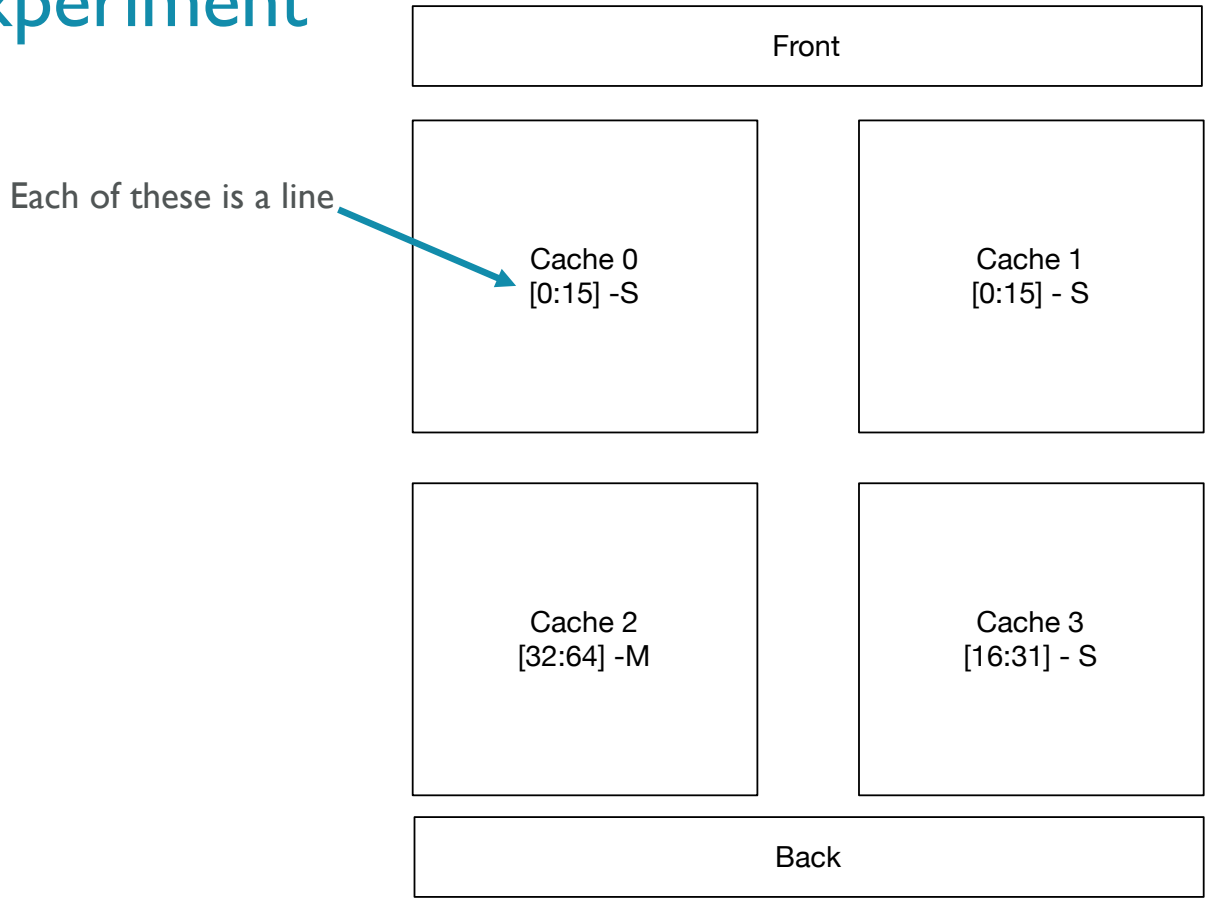


Coherence Network + Coherence

- Bring a line in to read/write in ‘E’
- Bring a line in to read in ‘S’
  - ‘S’ is nice b/c multiple cores caches can have the same line and read it
- If I need to write an ‘S’ line, I have to bring it to ‘M’ in order to write it. This means I have to send a message (to everybody else)
- The efficiency of this depends on the bus topology. Lets do an experiment

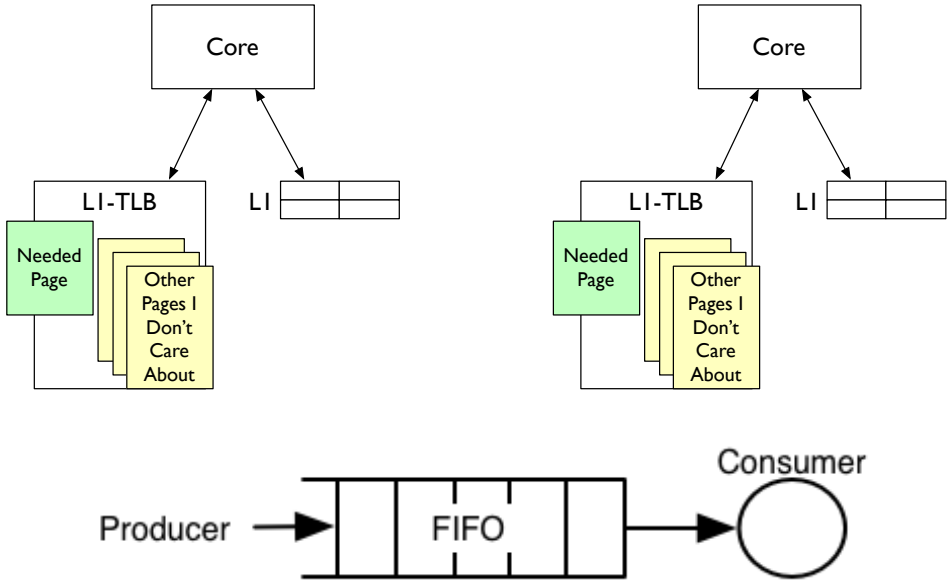


Experiment



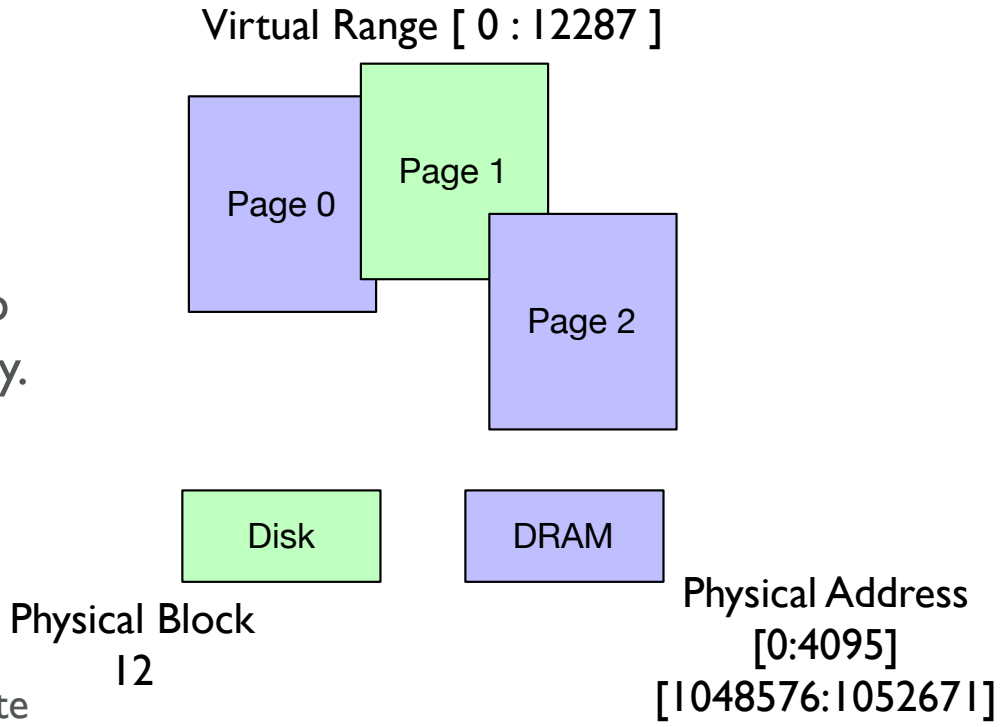
Hardware: Cache + Translation

- For an efficient FIFO, we want to move the cache line on producer and consumer as few times as possible
- We want to request things on the bus as few times as possible
- We also want to keep the translation local so we don't have to request it over and over

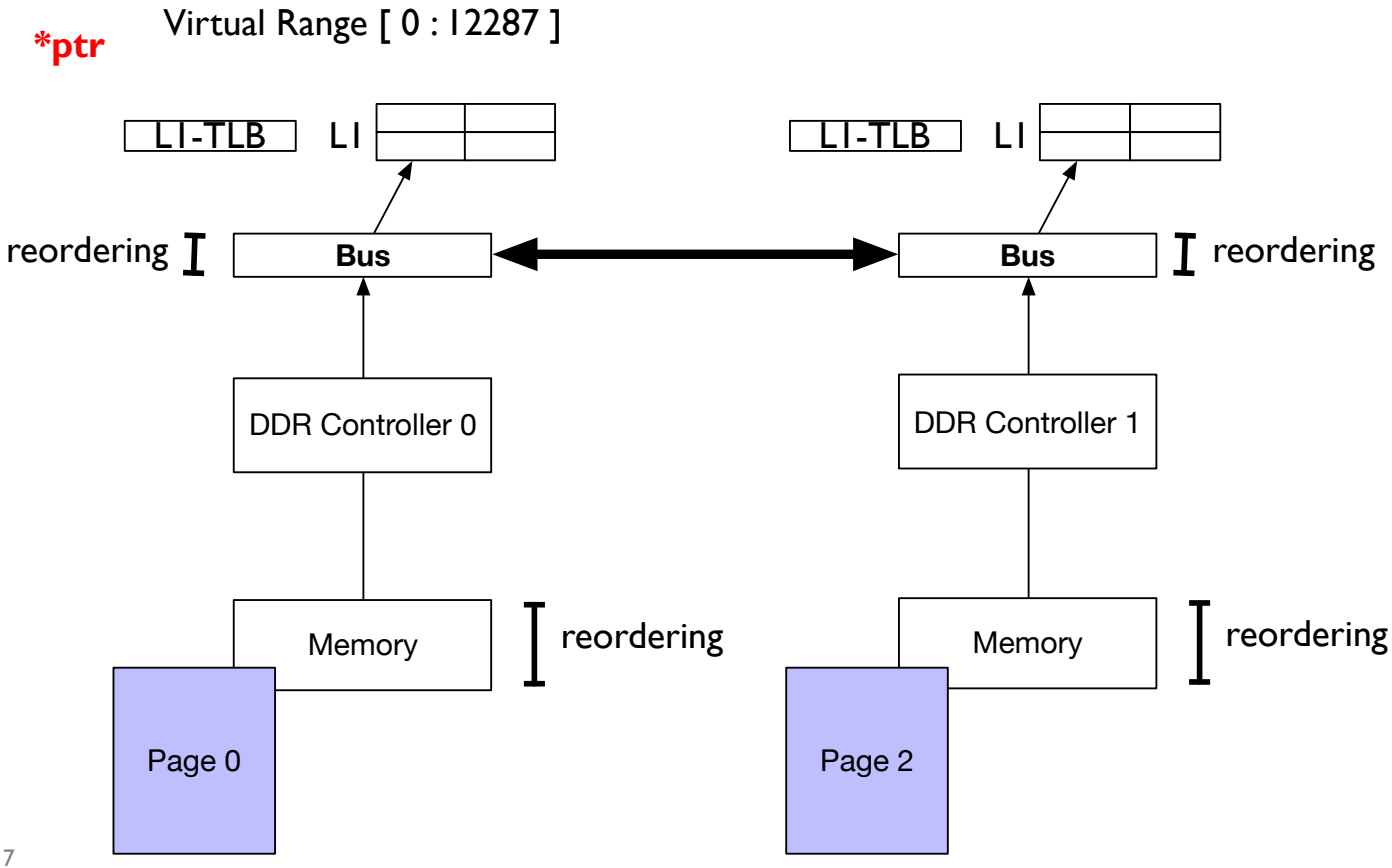


Virtual Memory

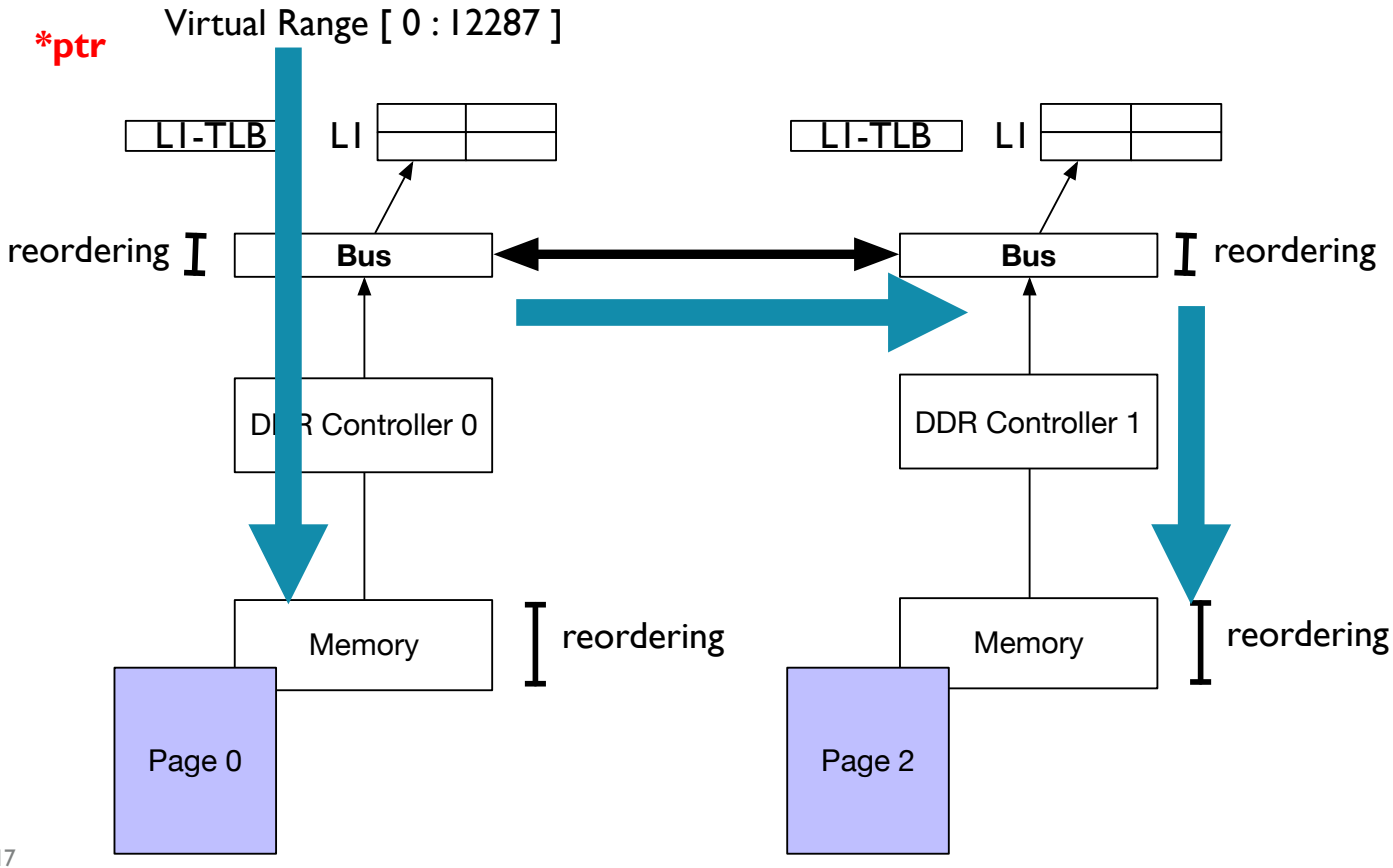
- Long and interesting history, might make interesting pub conversation
- A virtual memory page can map to anywhere in physical memory.
  - In the past this would only be DDR, increasingly there are non-volatile DIMMs as well.
  - Virtual memory is fully associative
  - Virtual pages don't have to be contiguous in physical memory despite being contiguous in virtual space



DDR to Core

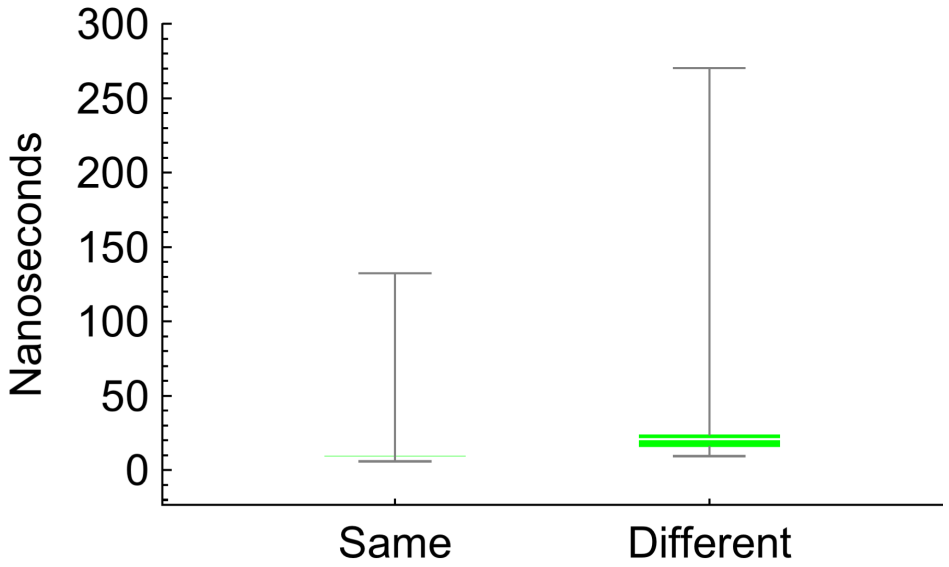


DDR to Core



NUMA influence on Access Latencies

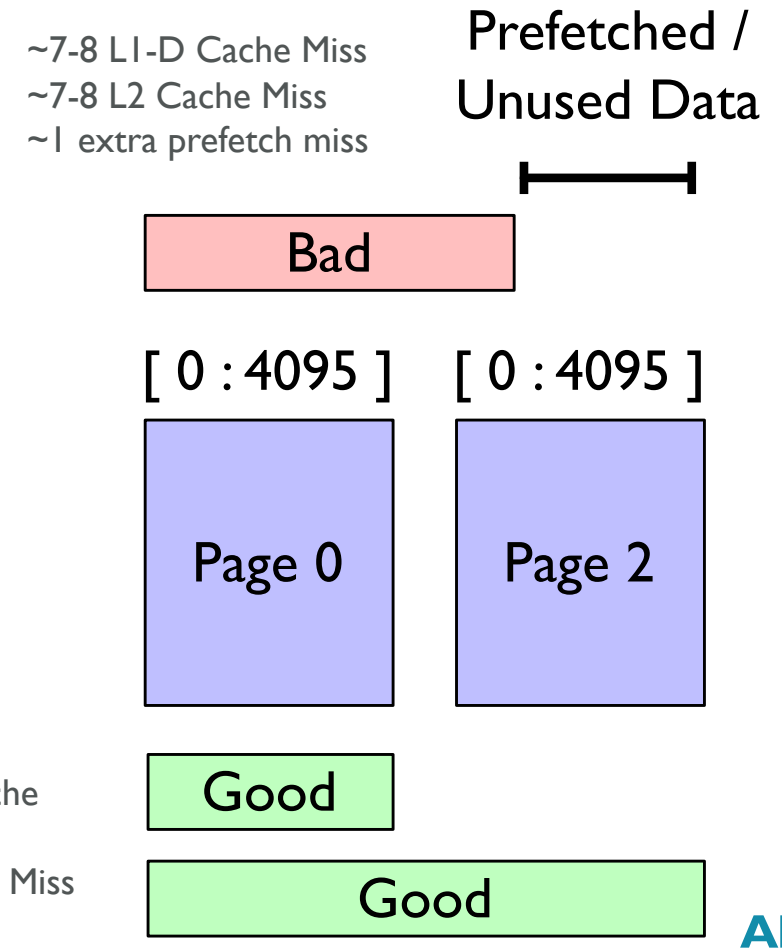
- Depending on the node used to allocate the memory, access times may vary drastically
- Even if *libc* doesn't do the right thing for us, on most operating systems there is a fix: *numa_page_move* which we'll discuss shortly



Prefetch Maybe?

- Prefetch is something that cache controllers typically do in hardware to hide memory latency
- Prefetchers find patterns
- Typically work on a single page
- LLC in some cores can now do next page too
- What does this mean for thread communication?
- Be careful on shared LLC
  - Add 1 prefetch miss if shared

~5-6 L1-D Cache Miss
~5-6 L2 Cache Miss



ARM

Some Systems Programming Stuff You Should Know

Blue Pill of Virtual Memory

- Makes things easy for the programmer
- Hard to know where your memory actually is
- We know that location matters for performance
  - Cache line location / state
  - Page placement
  - Kernel paging policy
  - Sometimes the red pill is better, but harder to swallow

MMAP

- Load a chunk of memory at a page size granularity.
  - In Linux, *sysconf* can get you the page size if you need it
  - Only problem, pages might not physically be allocated, same for regular malloc/new
  - Pages are zero pages until "touched" b/c of Copy-on-Write ()

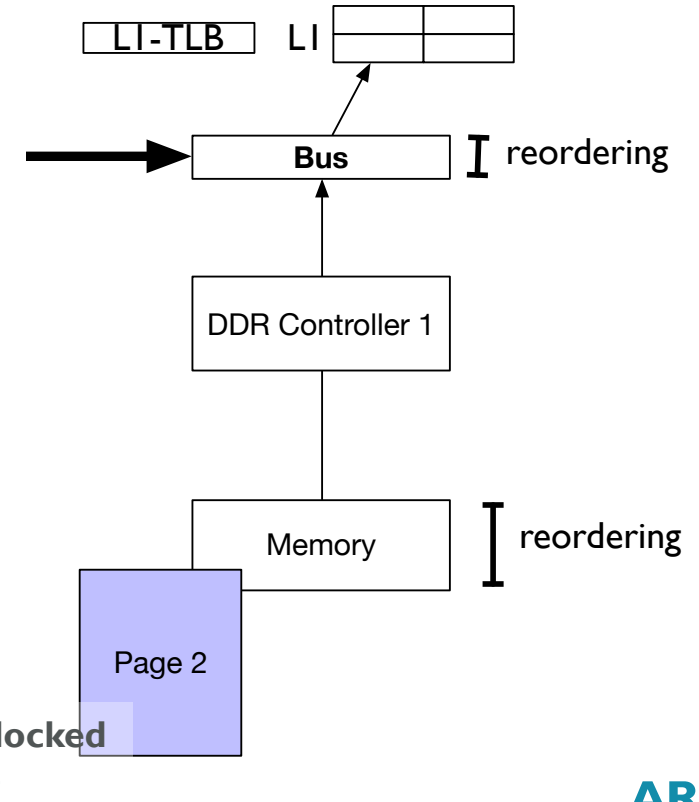
- Why use?
  - Map memory in a specific physical location, or try to at least. Not too helpful
  - Allocate a very large chunk, still divided by pages, but this interface allows multiple types of mapping hints that can be useful
  - Using in conjunction with shared memory (shm_XX) we can use this memory with multiple processes.
  - OTHERWISE NO REAL ADVANTAGE HERE, moving on

madvise

- Tell the kernel and memory system how you'll be using the memory, sounds useful right?
- Why use?
  - Great in theory, in reality, doesn't do much
  - Most modern processors can do next page prefetching at LLC in hardware versus software.
- Profiled with boost lock-free FIFO and several ring buffer implementations greater than 4-pages, no discernable improvement, moving on...

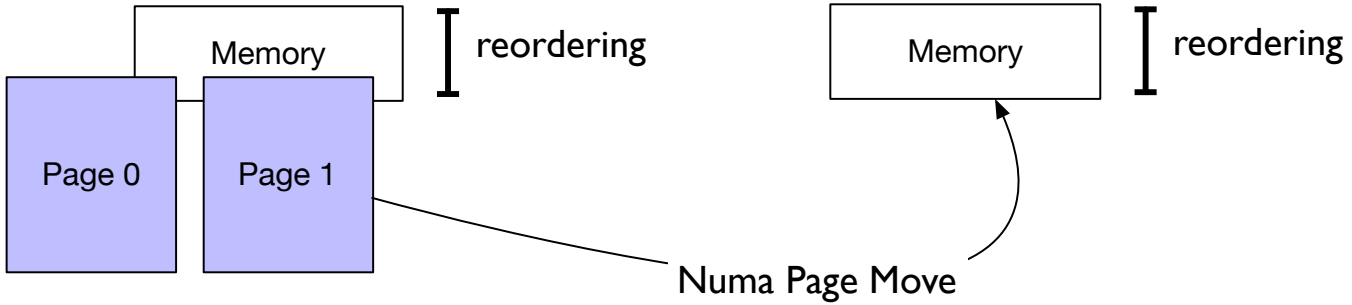
Don’t Pull the Rug – Lock pages

- Lets talk about putting locks on your page.
- Not permissions, but placement.
- Upside:
  - Page stays in physical memory
  - Faster access on average, really just more uniform
  - Uniformity leads to less variation in execution times
  - ~1 fewer cache misses (can’t quite fully explain, likely artifact of something else)
- Downside:
  - Physical pages aren’t available for other processes to use



Move Pages (remember NUMA)

- The beginning and end of our FIFO could be on differing NUMA nodes.
- A source or destination worker threads might move during execution
- Write path is less critical than read path for timing.
- Most operating systems take a 1st Touch Policy
- Moving can net an additional 40ns per read on receiver side
- Virtual address stays the same, physical page moves



Caution

- Be careful with:
  - Locking pages
  - Moving pages
  - madvise-ing pages (especially with the MADV_DONTNEED)

- Many of these don't play kindly with transparent huge pages
  - Odd bugs can result
  - MADV_DONTNEED might not actually release the page you planned on releasing

Now Actual Software

Darth Mutex

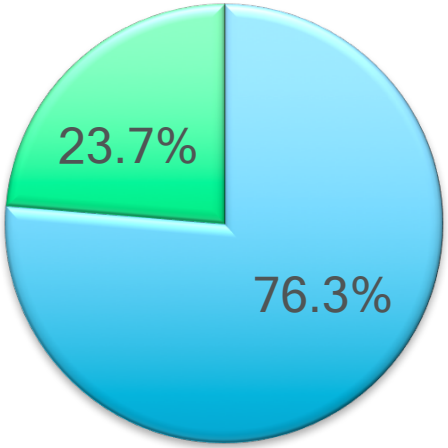
- Many implementations do away with
- Here's what a mutex looks like ⇒
- It's 40-bytes padded to 64
- Will incur cache miss if guarding head/tail pointers on every access
- Getting rid of it can save ~100 cycles for a contended lock
- **Be mindful where you use them**
- **CAS / Spinlock / Atomic likely better choices**

```
/* Data structures for mutex handling.  The structure of the attribute
   type is not exposed on purpose.  */
typedef union
{
  struct __pthread_mutex_s
  {
    int __lock;
    unsigned int __count;
    int __owner;
#ifdef __x86_64__
    unsigned int __nusers;
#endif
    /* KIND must stay at this position in the structure to maintain
       binary compatibility.  */
    int __kind;
#ifdef __x86_64__
    short __spins;
    short __elision;
    __pthread_list_t __list;
# define __PTHREAD_MUTEX_HAVE_PREV  1
/* Mutex __spins initializer used by PTHREAD_MUTEX_INITIALIZER.  */
# define __PTHREAD_SPINS             0, 0
#else
    unsigned int __nusers;
    __extension__ union
    {
      struct
      {
    short __espins;
    short __elision;
# define __spins __elision_data.__espins
# define __elision __elision_data.__elision
# define __PTHREAD_SPINS         { 0, 0 }
      } __elision_data;
      __pthread_slist_t __list;
    };
#endif
  } __data;
  char __size[__SIZEOF_PTHREAD_MUTEX_T];
  long int __align;
} pthread_mutex_t;
```

The Dark Side of Mutex

```
pthread_mutex_lock(&mutex[index][ipar % MUTEXES_PER_CELL]);
cell->a[ipar % PARTICLES_PER_CELL] += acc;
pthread_mutex_unlock(&mutex[index][ipar % MUTEXES_PER_CELL]);
```

Cycles spent



Avg. cycles waiting for lock	Avg. cycles of actual compute
46.32	14.34

Alignment

- Data structures in danger of being shared accidentally should be aligned so that they're solo.
- Known as "false sharing," but also ping-ponging
- Two methods:
  - Padding (e.g., char buffer[56])
  - alignas(64)

- Memory alignment
  - By page (largely covered)
  - posix_memalign / manual mem align

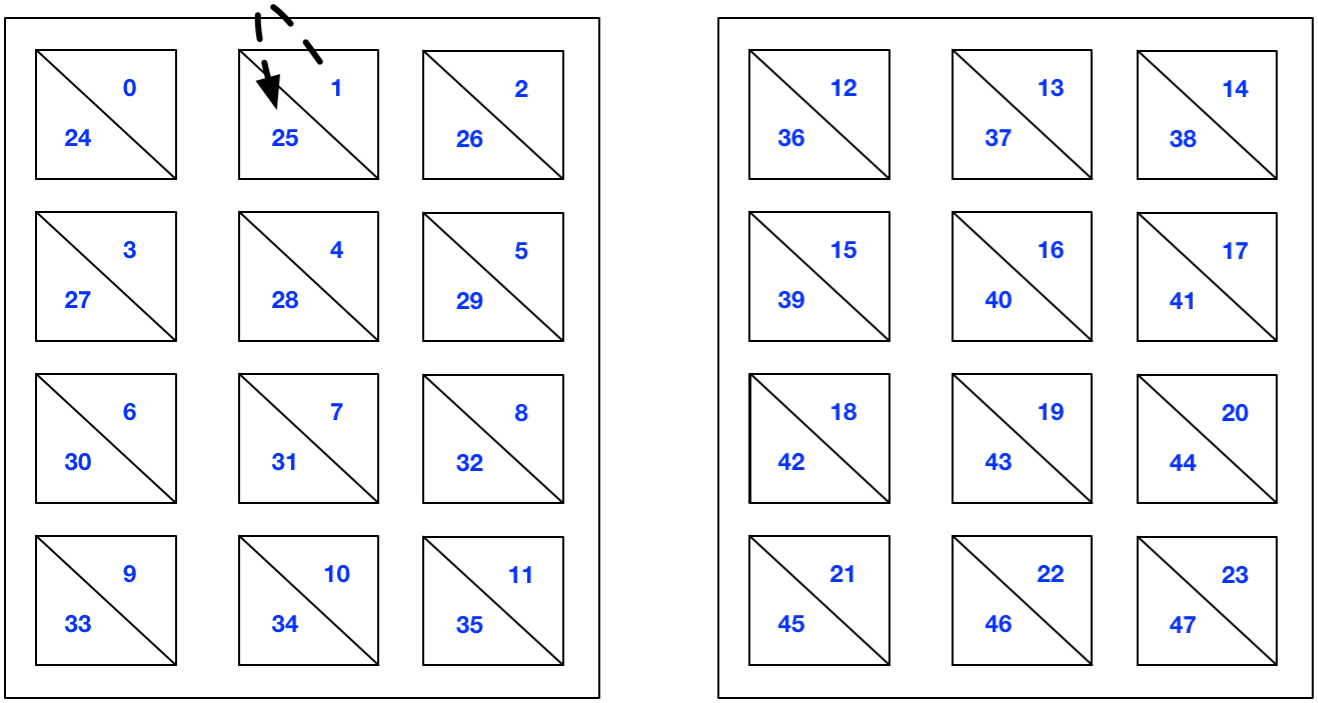
Locality Matters

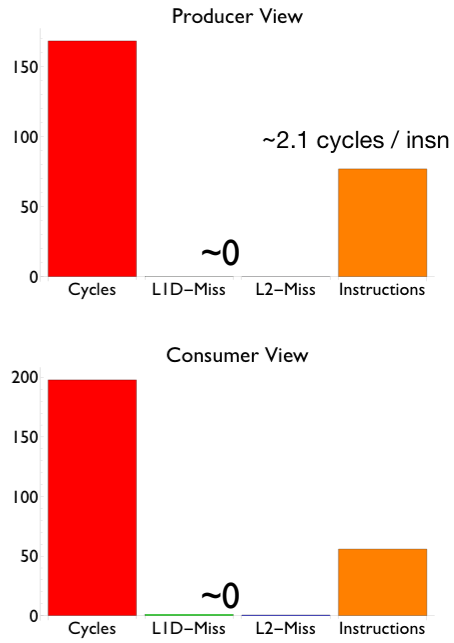
Using a boost lock free FIFO as an example

Locality details

- Given core-core latency numbers, we need a bit more to make sense of them
- Setup
  - Threads are pinned to cores using *sched_setaffinity*
  - Schedule is set to real-time RR
  - The entire binary is page-locked using *mlockall*(MCL_CURRENT | MCL_FUTURE)
  - Touches pages so that they're physically allocated vs. zero
  - Queue set to size=1, inter-push latency measured in cycles using *rdtscp* on a Haswell
  - Duplicated results using a round-trip approach, identical results

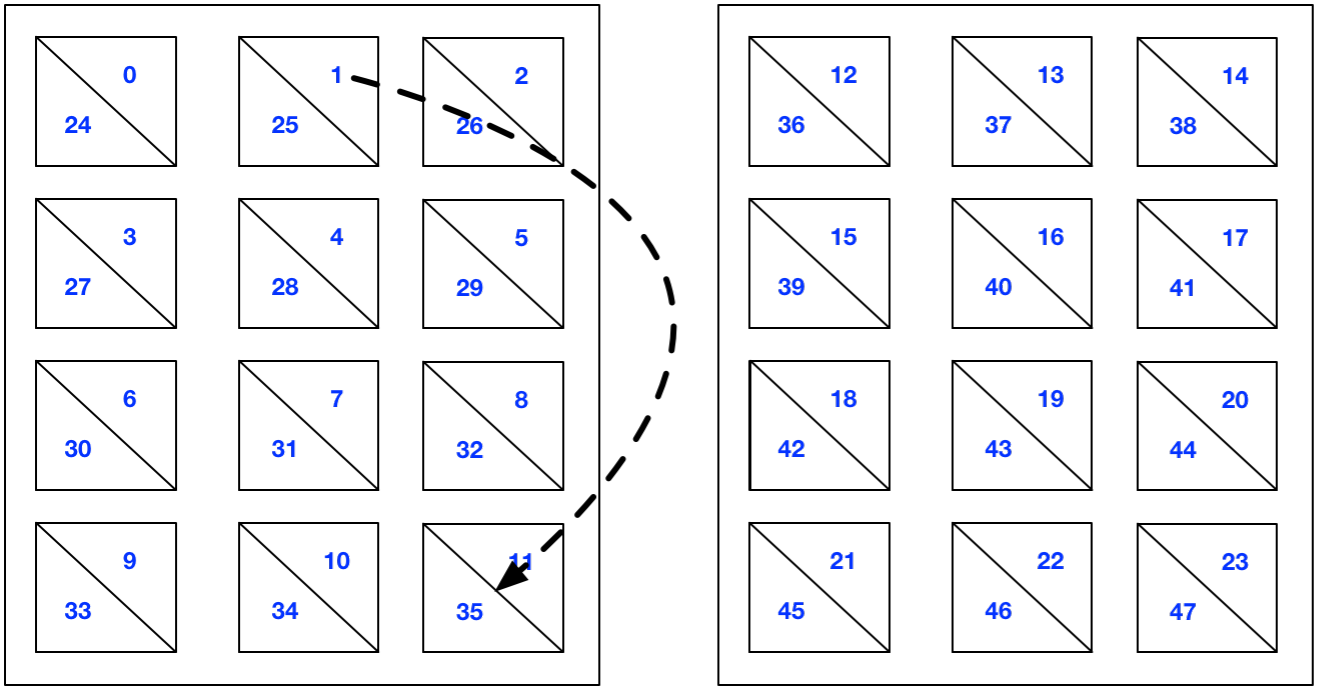
Two socket system, Producer – Consumer Pair

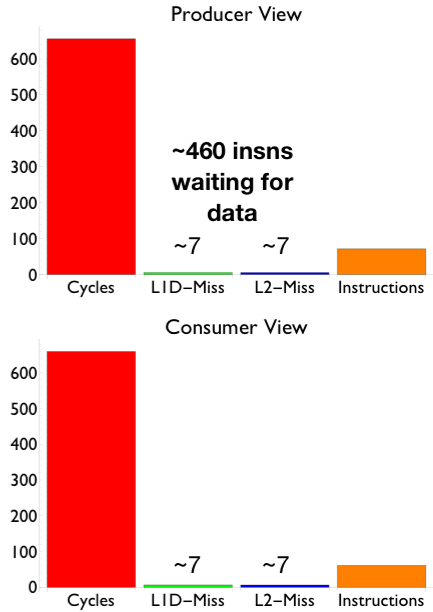




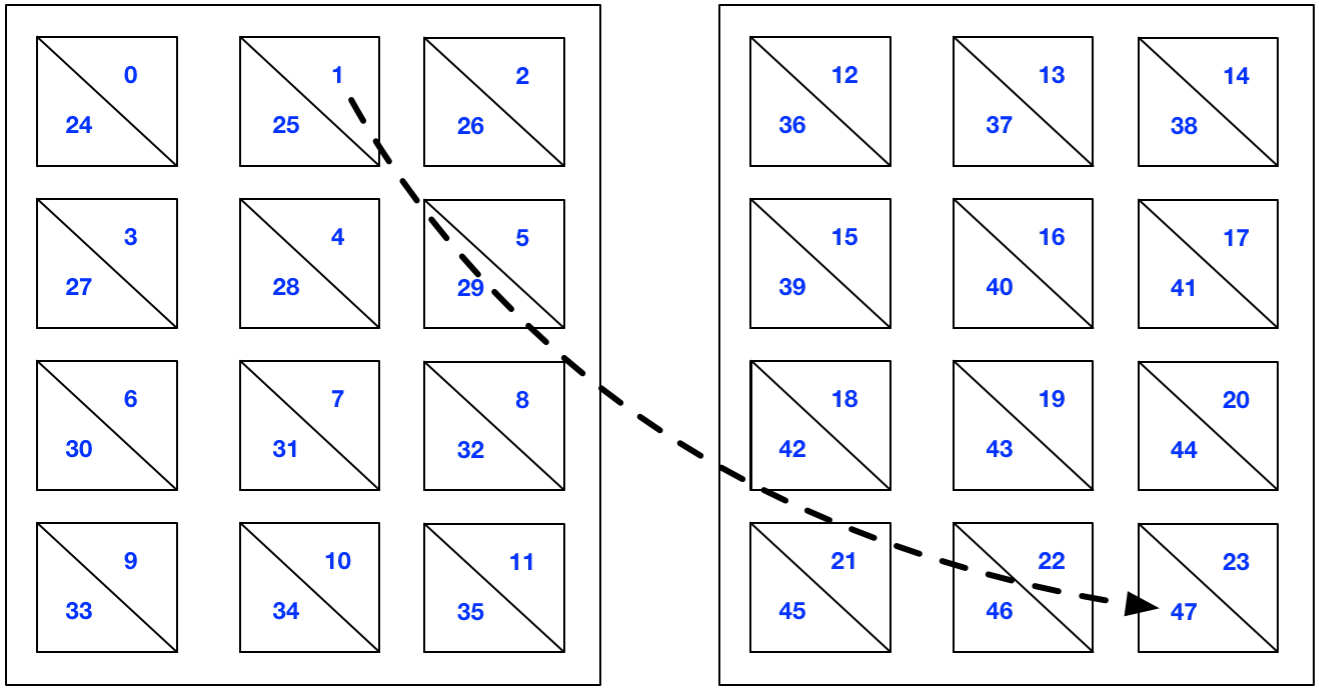
ARM

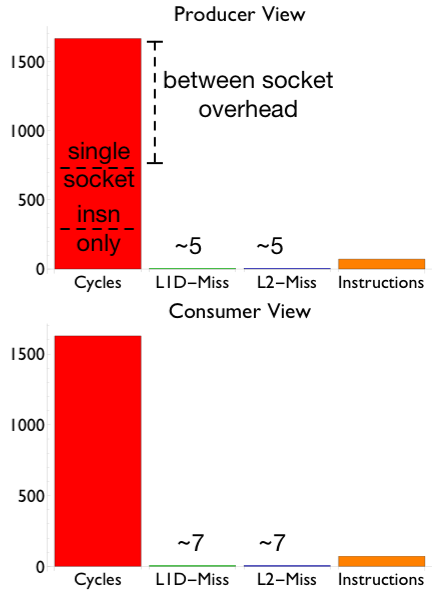
Two socket system, Producer – Consumer Pair





Two socket system, Producer – Consumer Pair





Conclusions

Conclusions

- Moving data from one core to the next is actually pretty complicated

- For maximum performance:
  - **Remember cache line alignment**

Conclusions

- Moving data from one core to the next is actually pretty complicated

- For maximum performance:
  - Remember cache line alignment
  - **Remember page alignment**

Conclusions

- Moving data from one core to the next is actually pretty complicated

- For maximum performance:
  - Remember cache line alignment
  - Remember page alignment
  - **Minimize translation steps**

Conclusions

- Moving data from one core to the next is actually pretty complicated

- For maximum performance:
  - Remember cache line alignment
  - Remember page alignment
  - Minimize translation steps
  - **Remember data placement**

Conclusions

- Moving data from one core to the next is actually pretty complicated

- For maximum performance:
  - Remember cache line alignment
  - Remember page alignment
  - Minimize translation steps
  - Remember data placement
  - **When possible, localize transfers**

Shameless Pitch

Come see me tomorrow for my RaftLib

tutorial presentation @ 11:00 in Bethe