Reducing Dark Bandwidth Through Data Reduction Near Memory

Jonathan Beard

Staff Research Engineer, Arm Inc.

UCAR SEA, Boulder, CO

4 April 2018

Twitter: @jonathan_beard

Disclaimer

This is a research project, not product.

Any changes to architecture must go through the standard architecture review board process

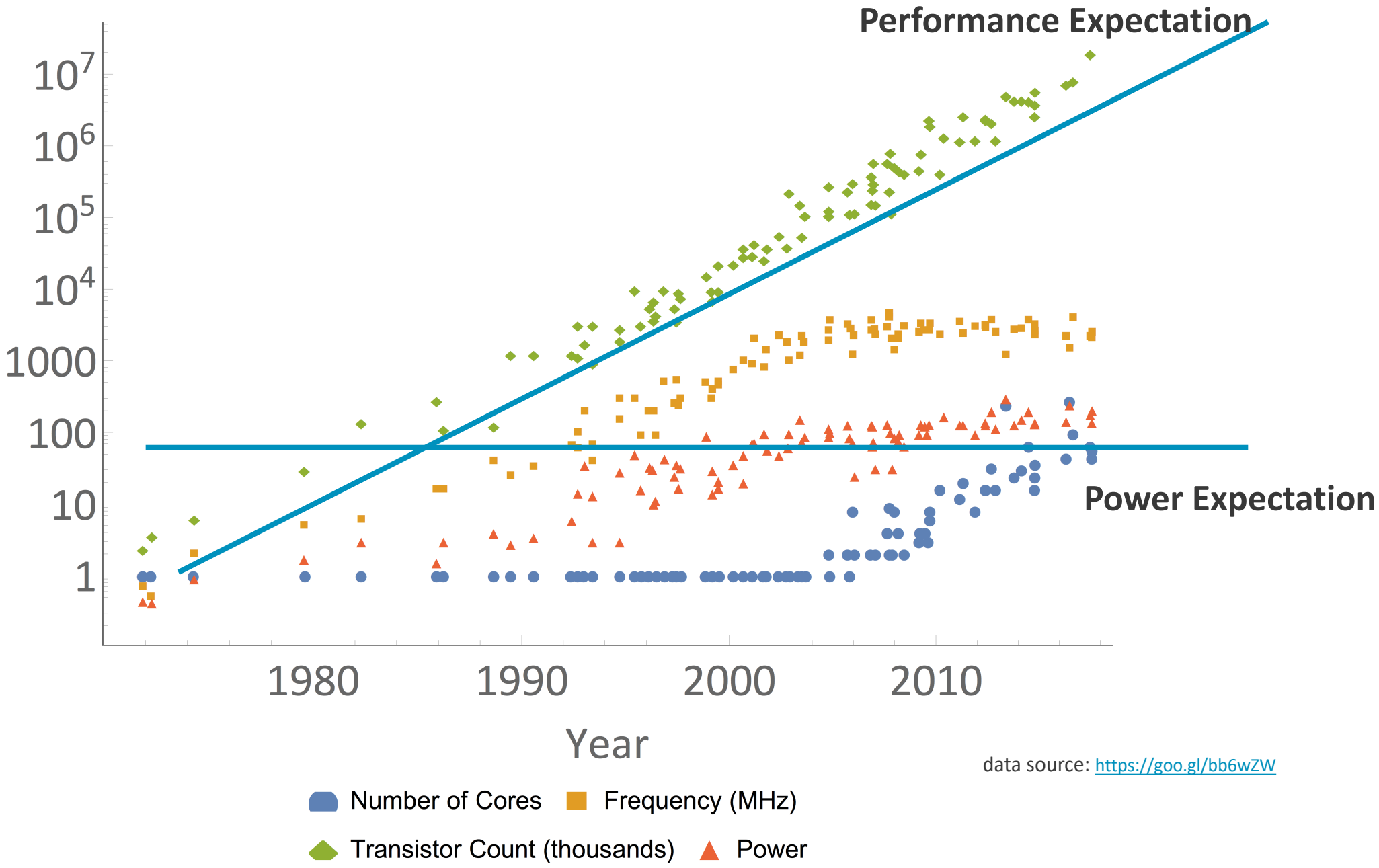
Nothing within this presentation should be taken as a direction of future architecture or product plans for Arm Inc/Ltd.

Please ask questions related to the presentation!

The lunch line is closed

- Put simply, frequency scaling is not a solution
- Performance curve must continue, we need FLOPS/IOPS/etc.
- **As a consequence:**
  - Core counts on chip and accelerators on chip will become more numerous
  - Integration will be closer, move onto system-on-chip (and multi-part chip)
  - What used to move off-chip before now moves on-chip, further stressing on chip interconnects
  - Increased compute density of the on-chip demands even more off-chip data movement bandwidth
  - As a result, less memory bandwidth/cache/interconnect capacity per core on average….
- **Core problem:**
  - We can build machines with tons of FLOPS, but we can’t feed them
  - Data movement is a key problem…guess what, we move a lot of data we don’t need to move

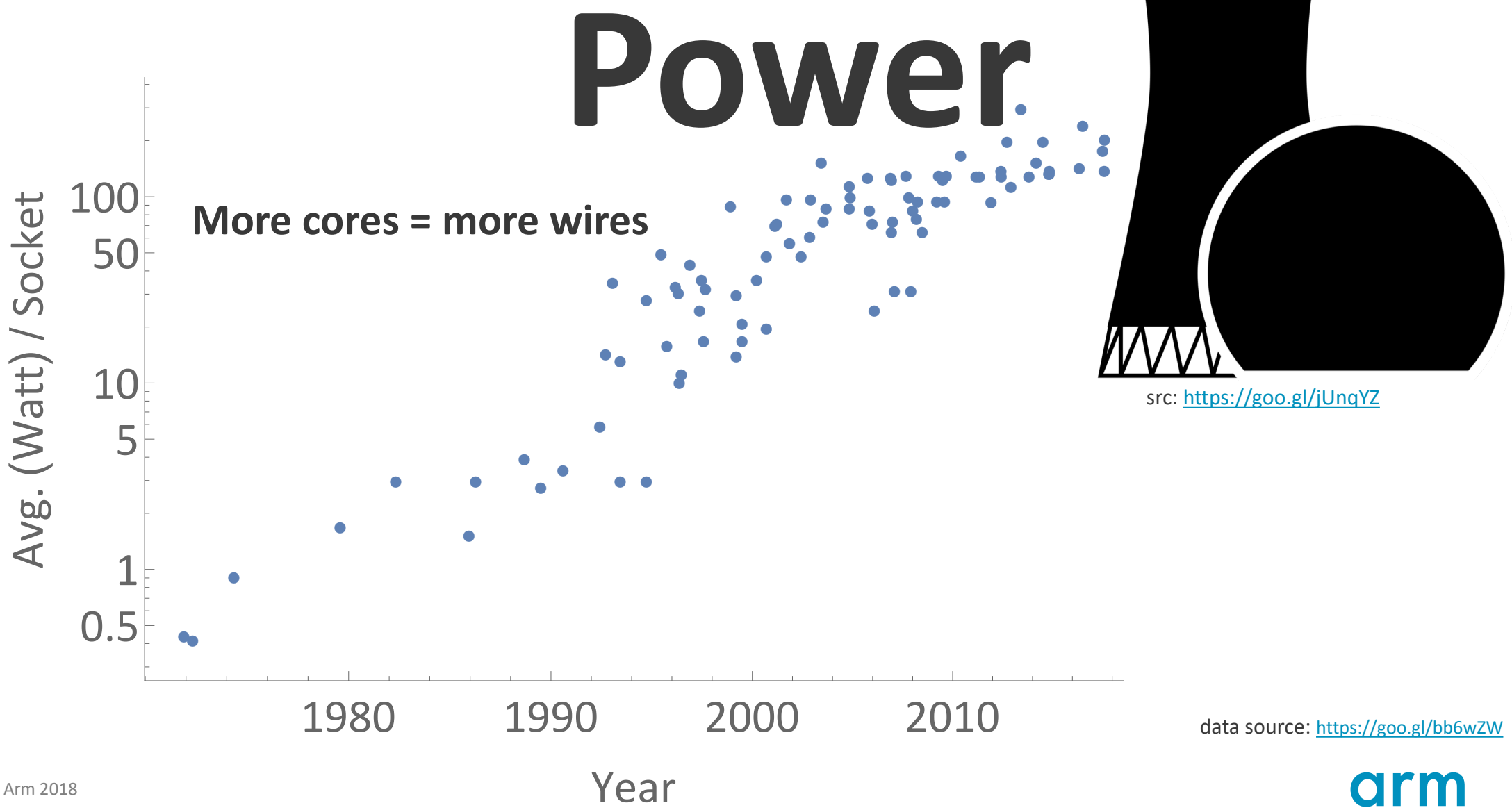
Trends



data source: https://goo.gl/bb6wZW

arm

More cores, more wires, more power

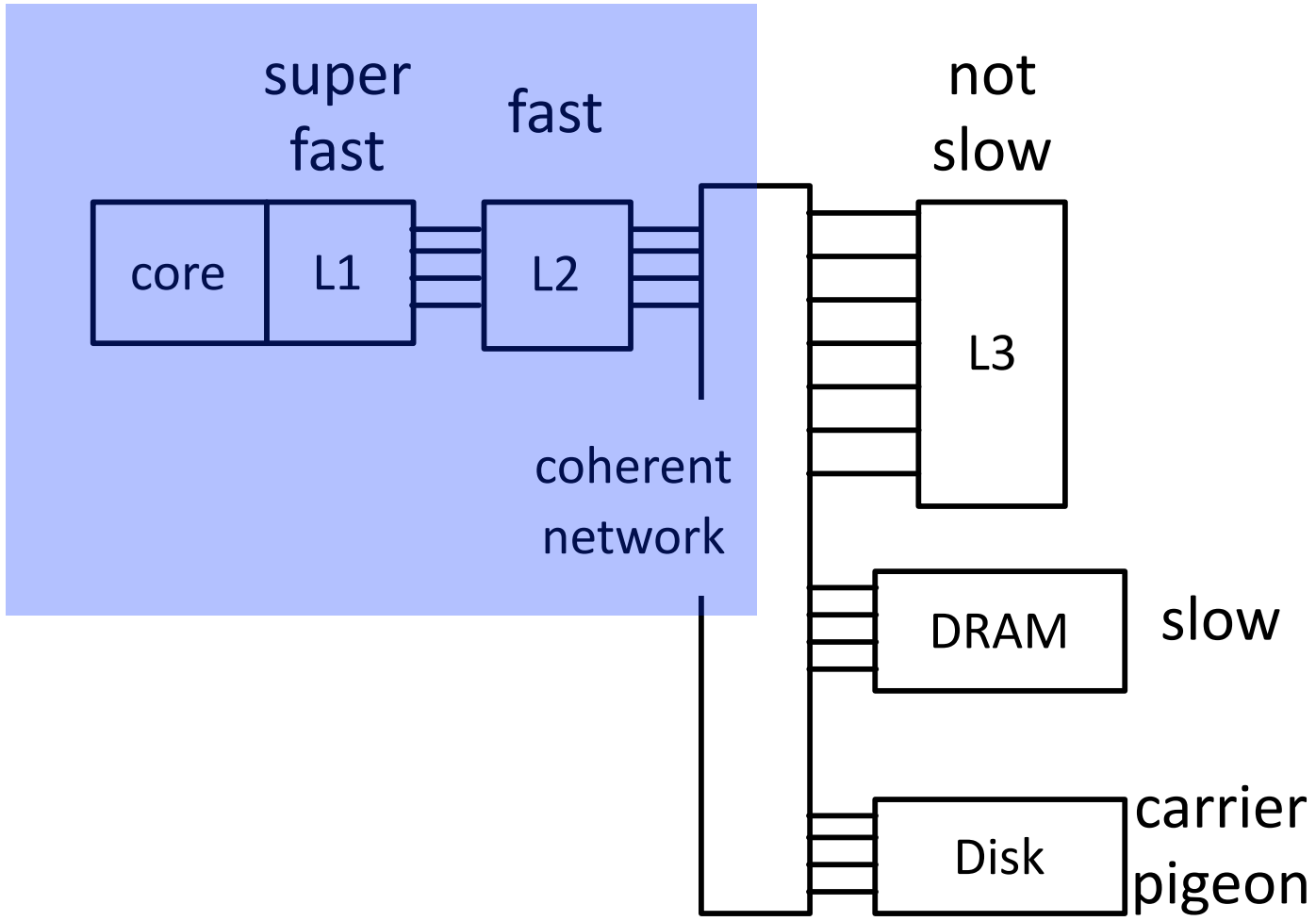


src: https://goo.gl/jUnqYZ

data source: https://goo.gl/bb6wZW

arm

Cores are reuse optimized



arm

Dark Bandwidth:

Data moved throughout the memory hierarchy that is not used after it is moved.

Superfluous Data Movement

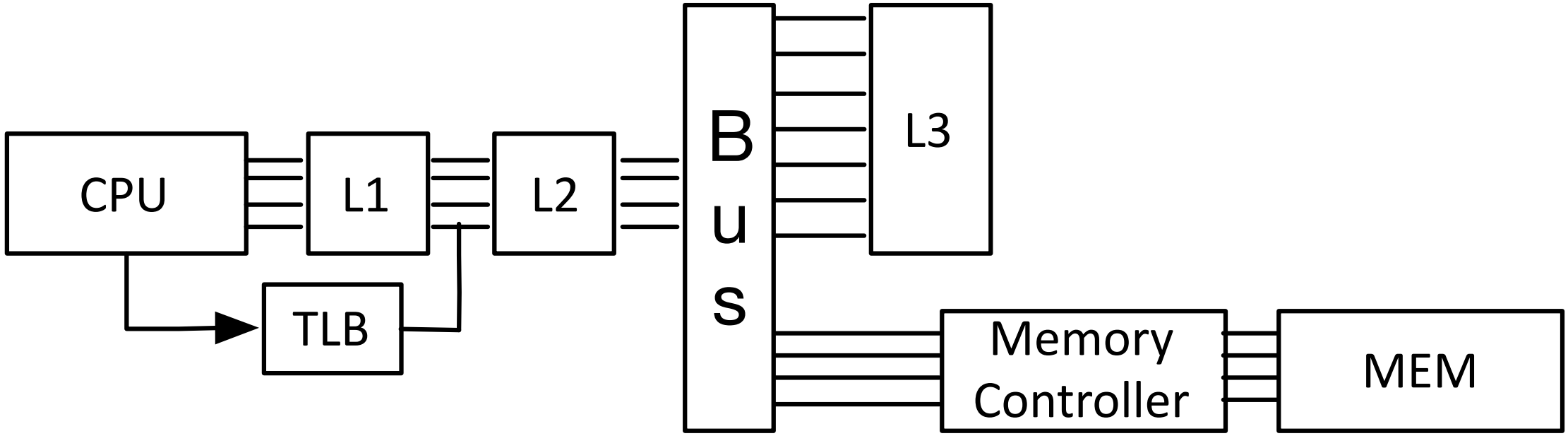
Utilized Bandwidth

Dark Bandwidth

arm

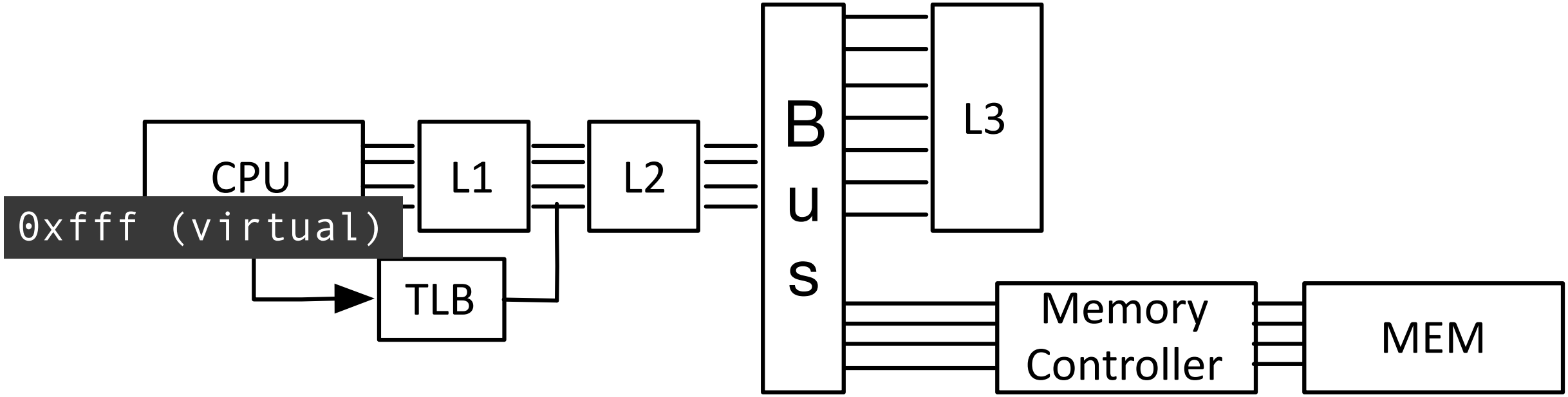
How we get data

```
move    -16(%r1), %r2
```



How we get data

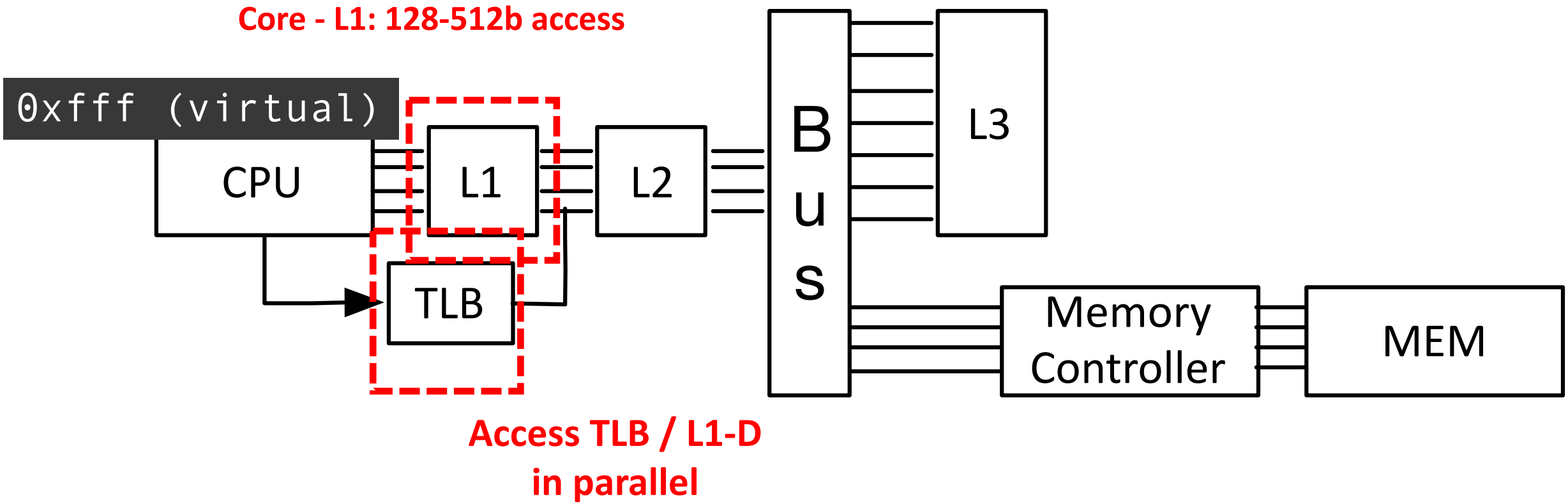
```
move    -16(%r1), %r2
```



arm

How we get data

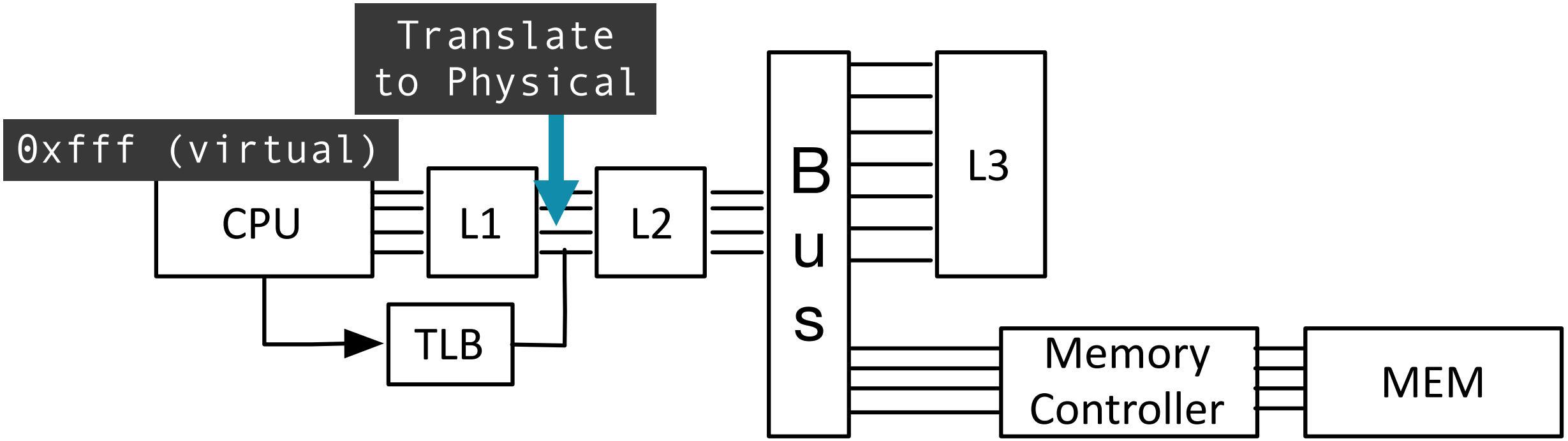
```
move    -16(%r1), %r2
```



arm

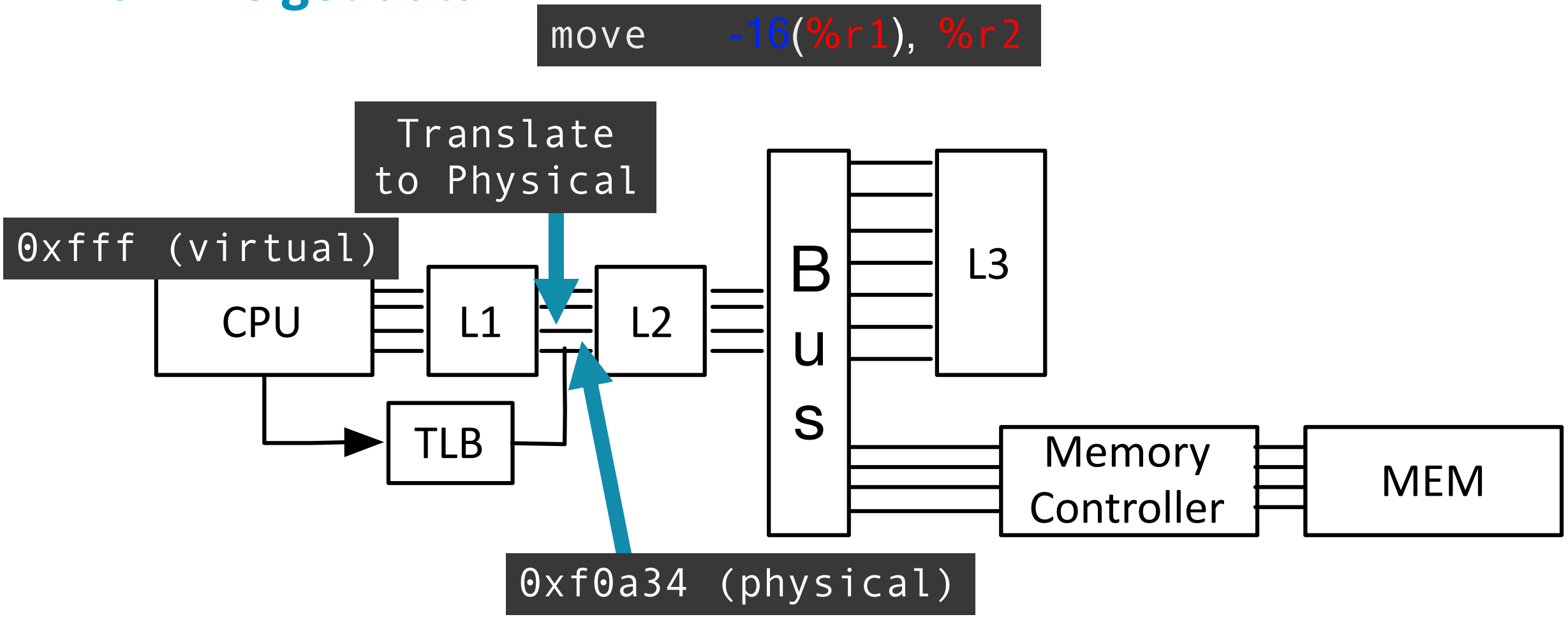
How we get data

```
move    -16(%r1), %r2
```



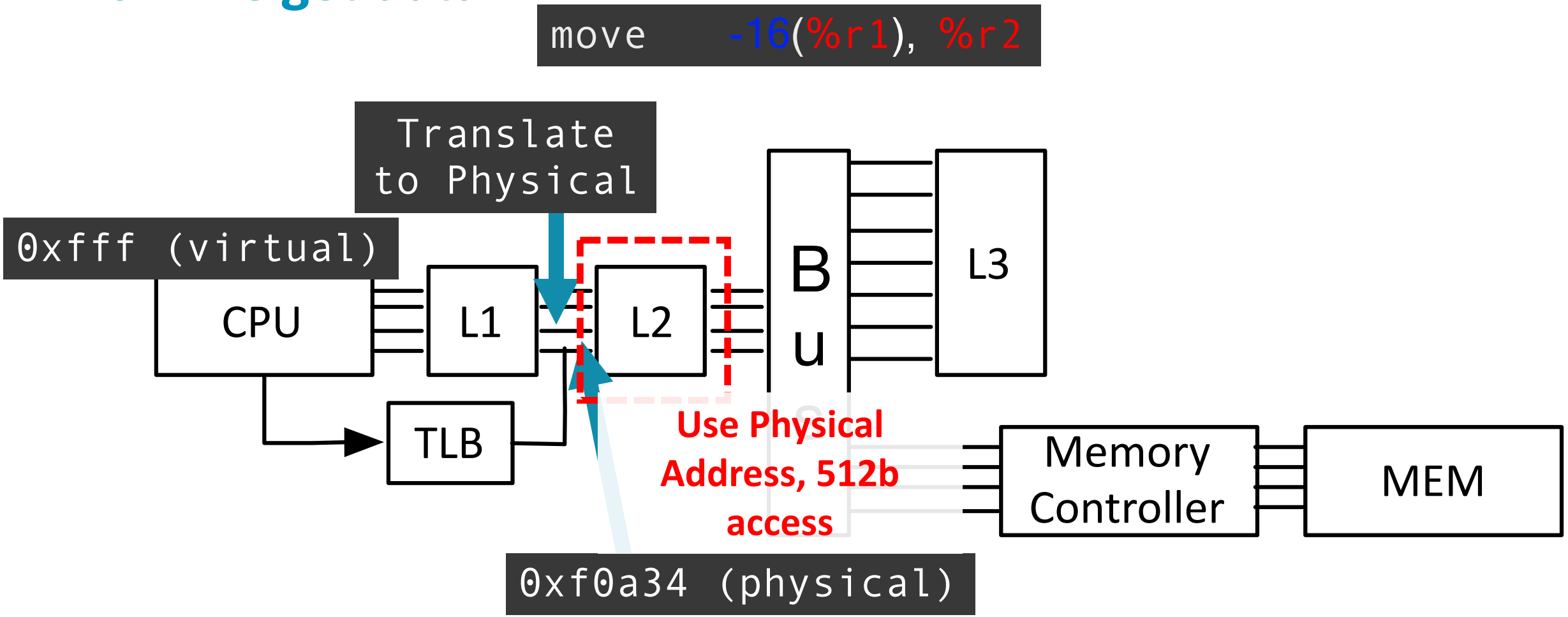
arm

How we get data



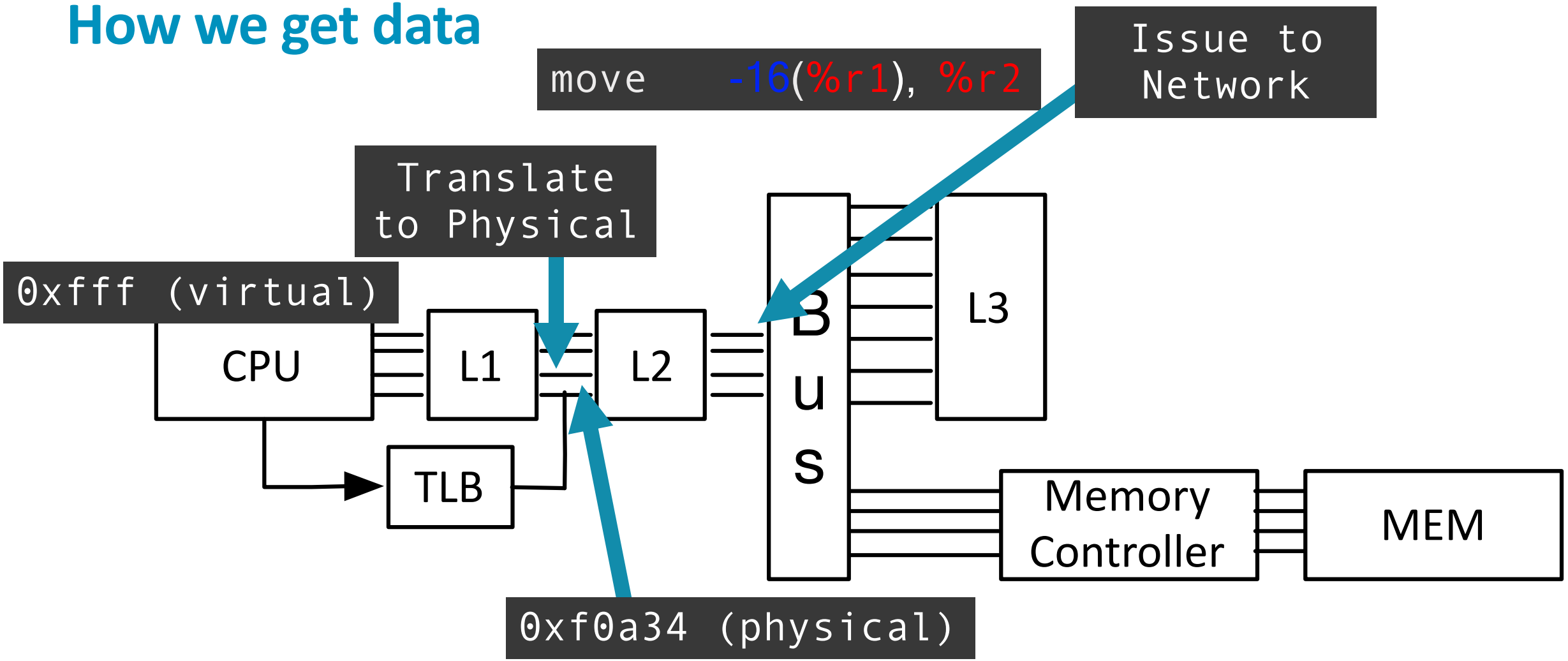
arm

How we get data



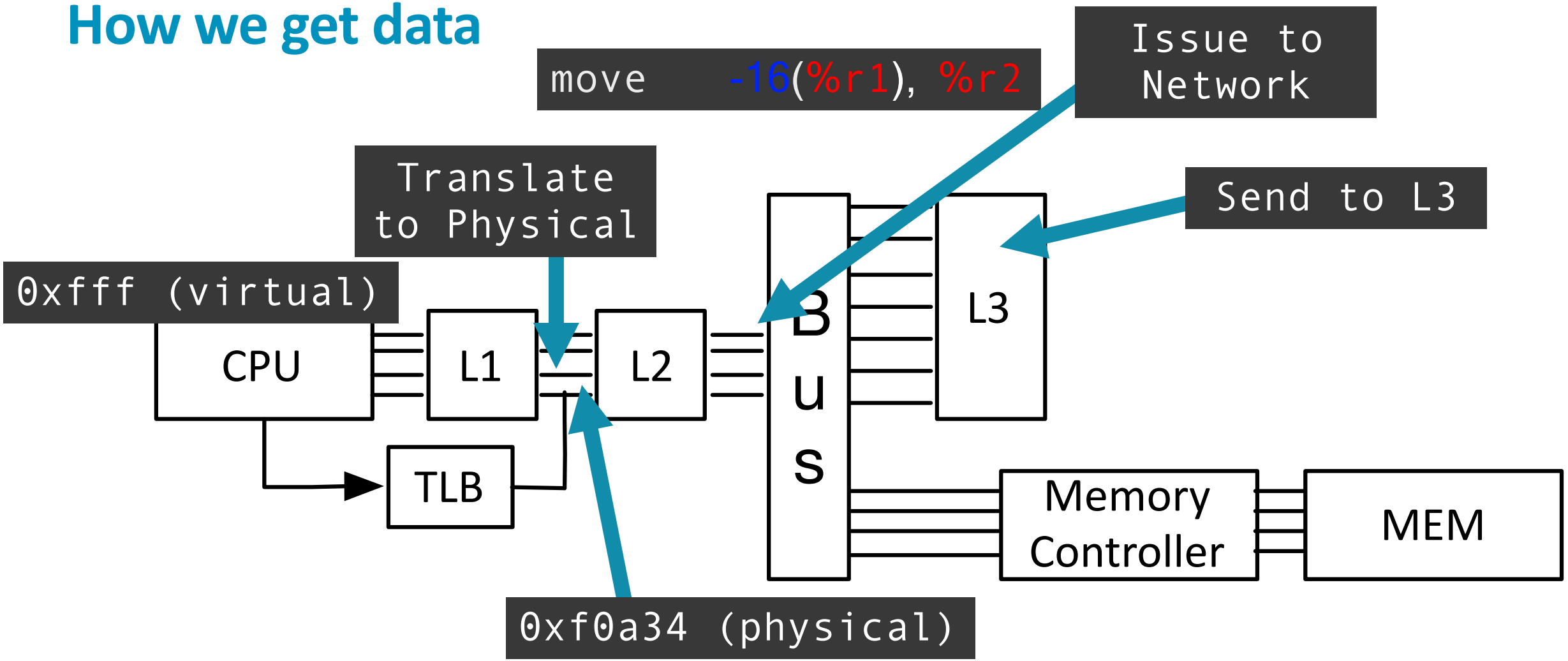
arm

How we get data



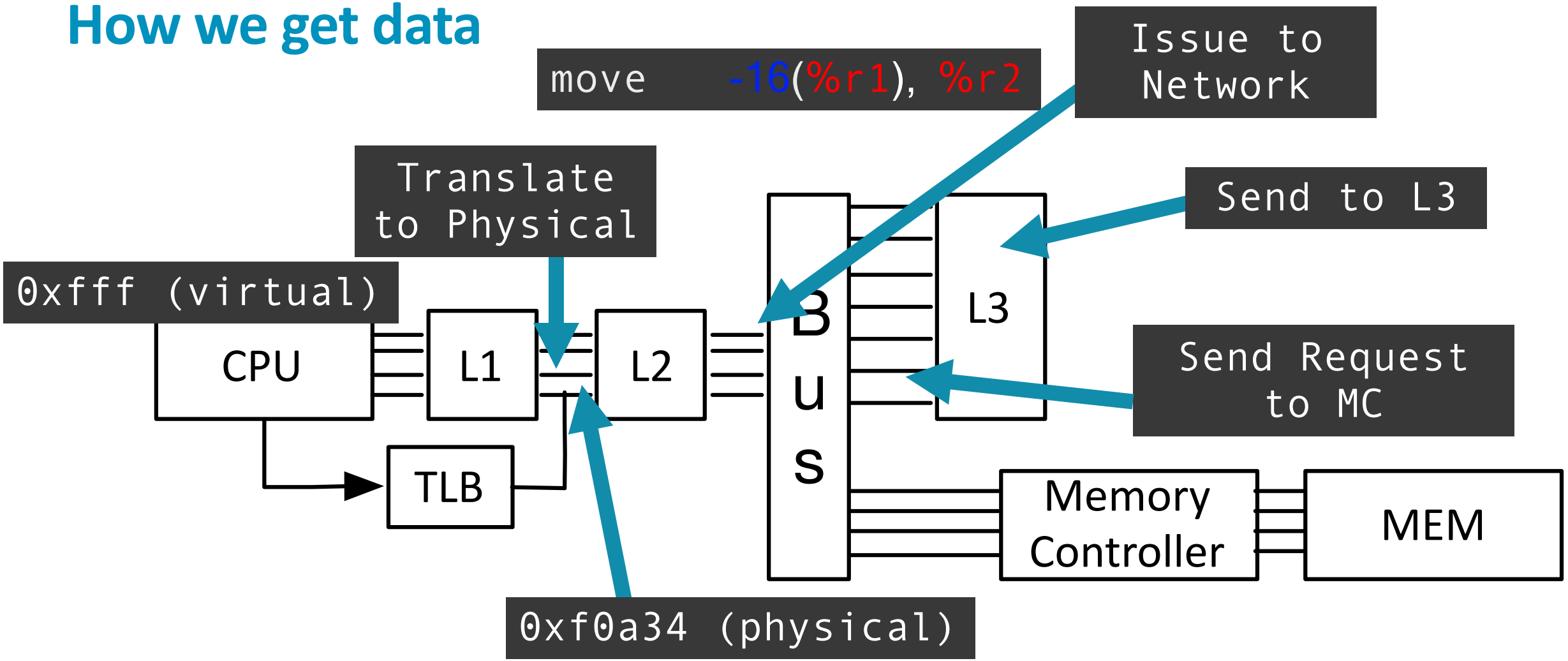
arm

How we get data



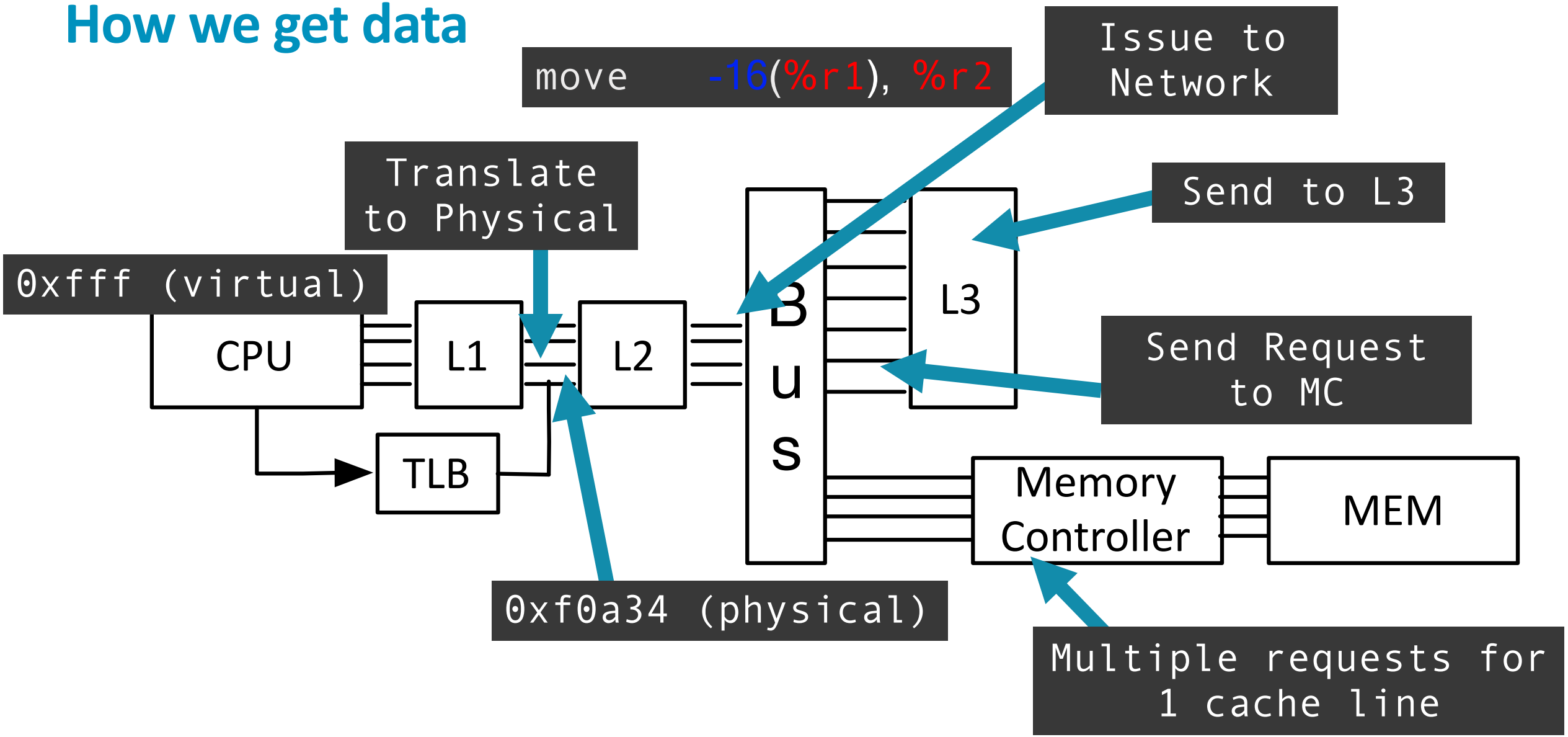
arm

How we get data



arm

How we get data

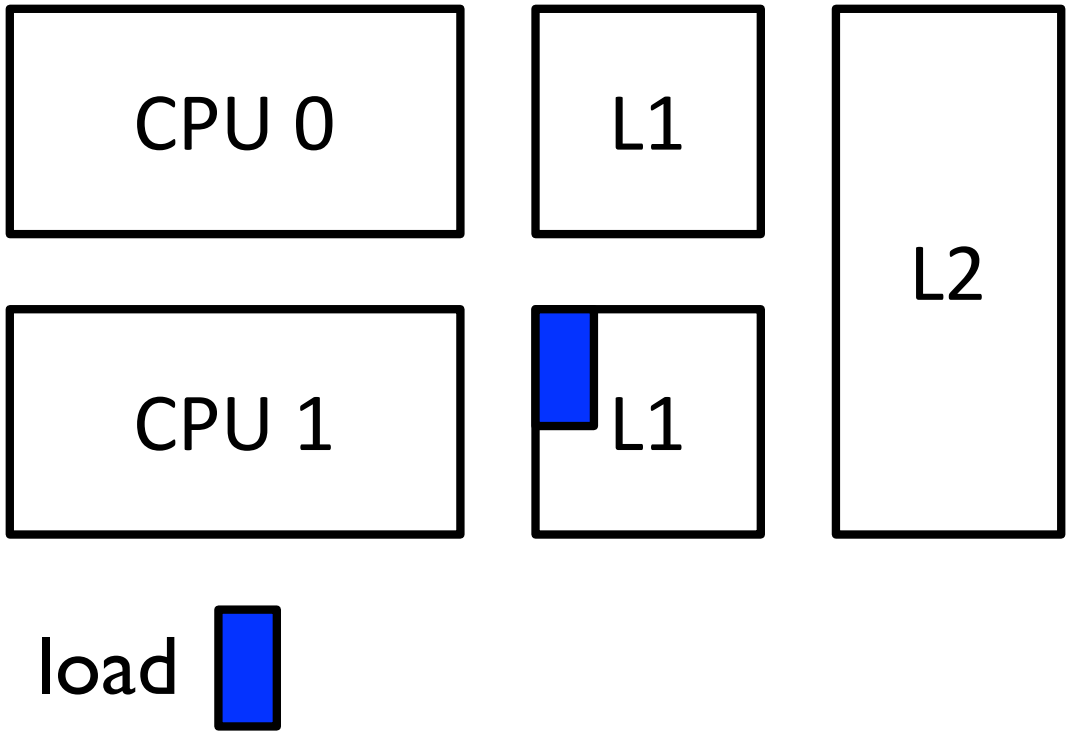


arm

Reuse Distance

If data within a cache line is used more than once, then the cache line is said to be reused

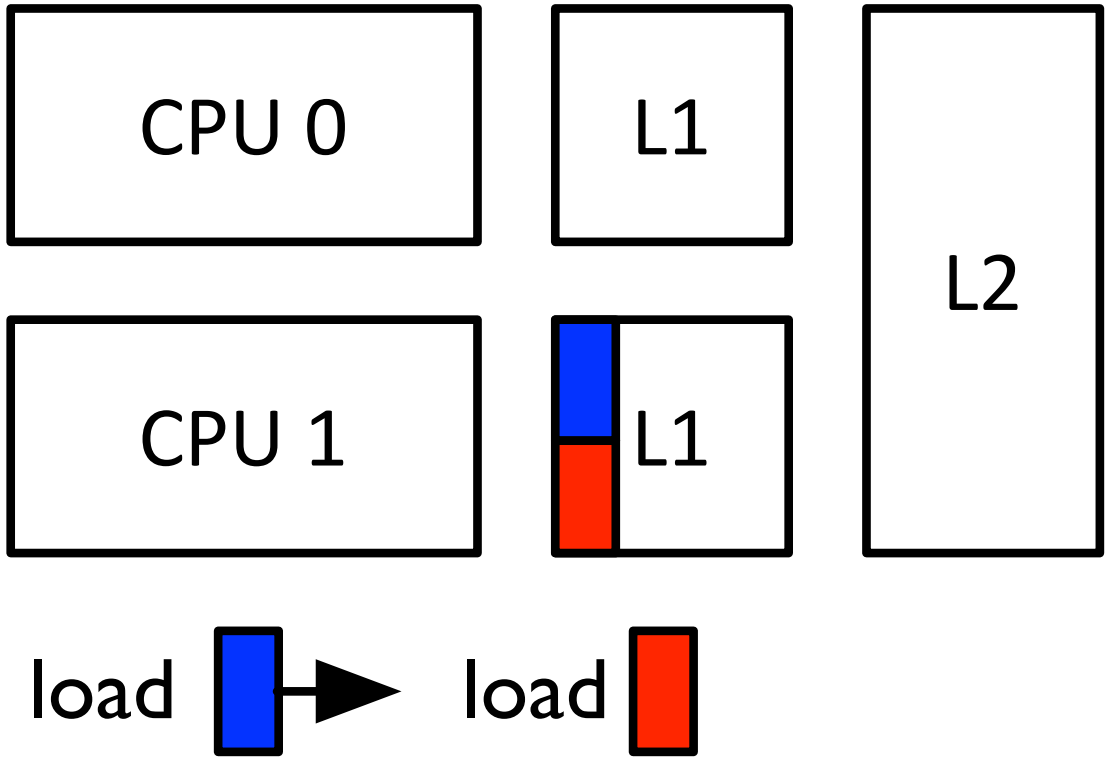
We need to answer: how many other cache lines are used in between accesses?



Reuse Distance

If data within a cache line is used more than once, then the cache line is said to be reused

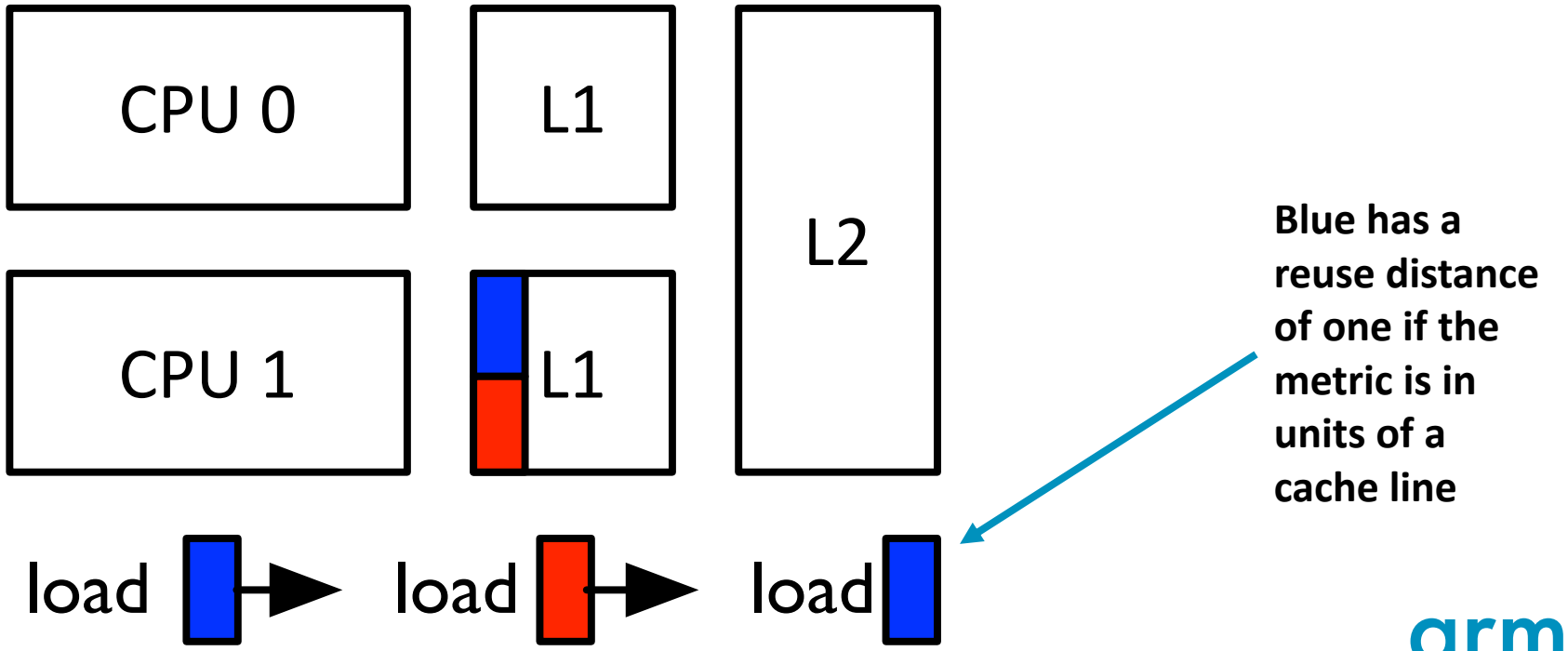
We need to answer: how many other cache lines are used in between accesses?



Reuse Distance

If data within a cache line is used more than once, then the cache line is said to be reused

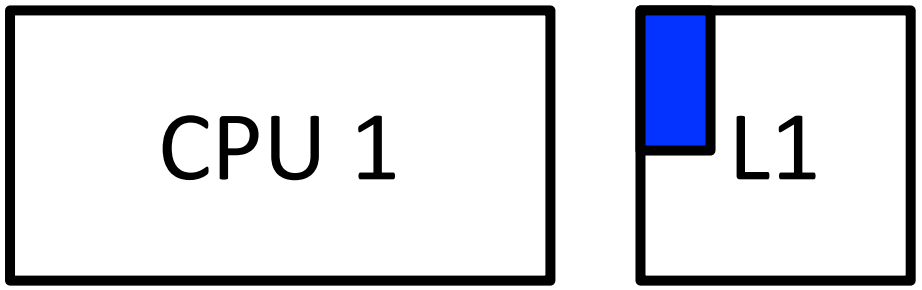
We need to answer: how many other cache lines are used in between accesses?



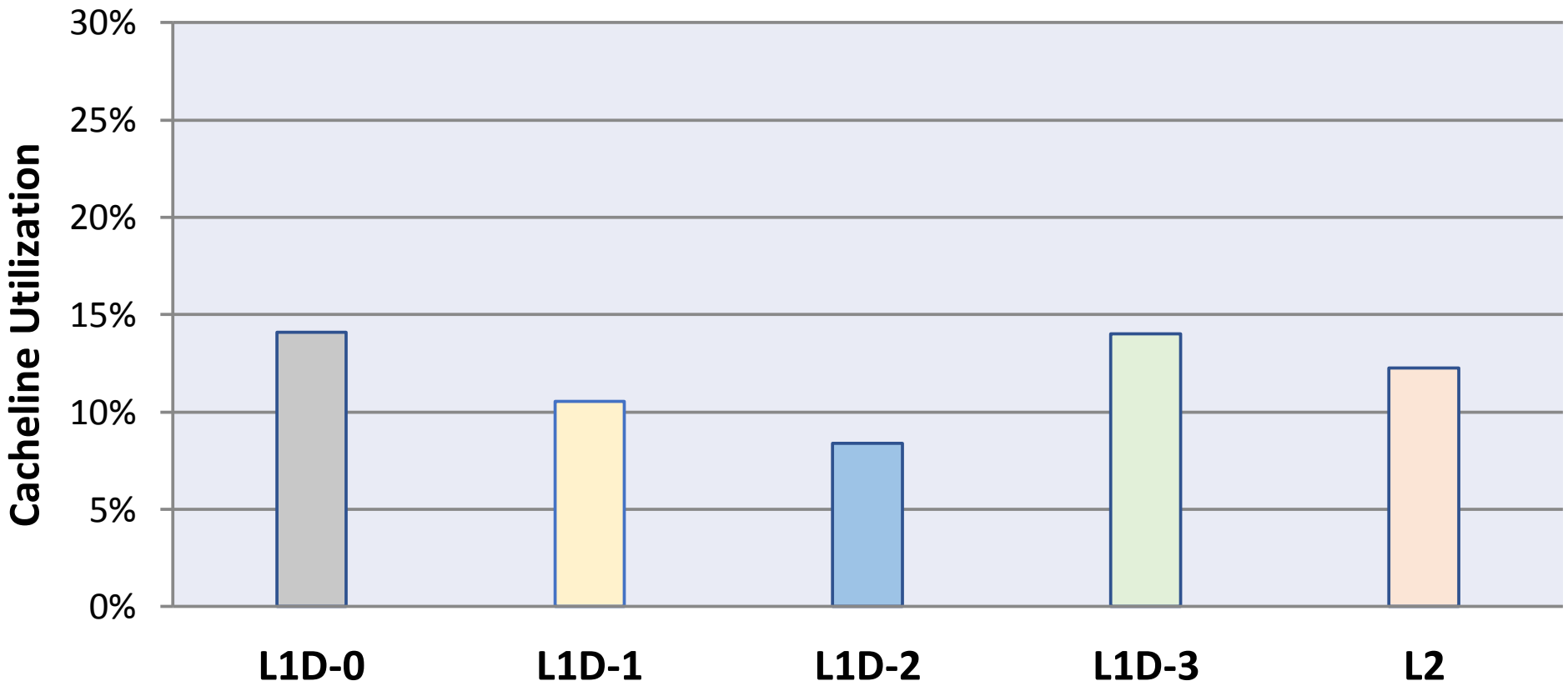
Utilization

How much of a cache line is used before it is evicted?

We moved all the data labeled waste through that entire maze of wires only to evict it again!!

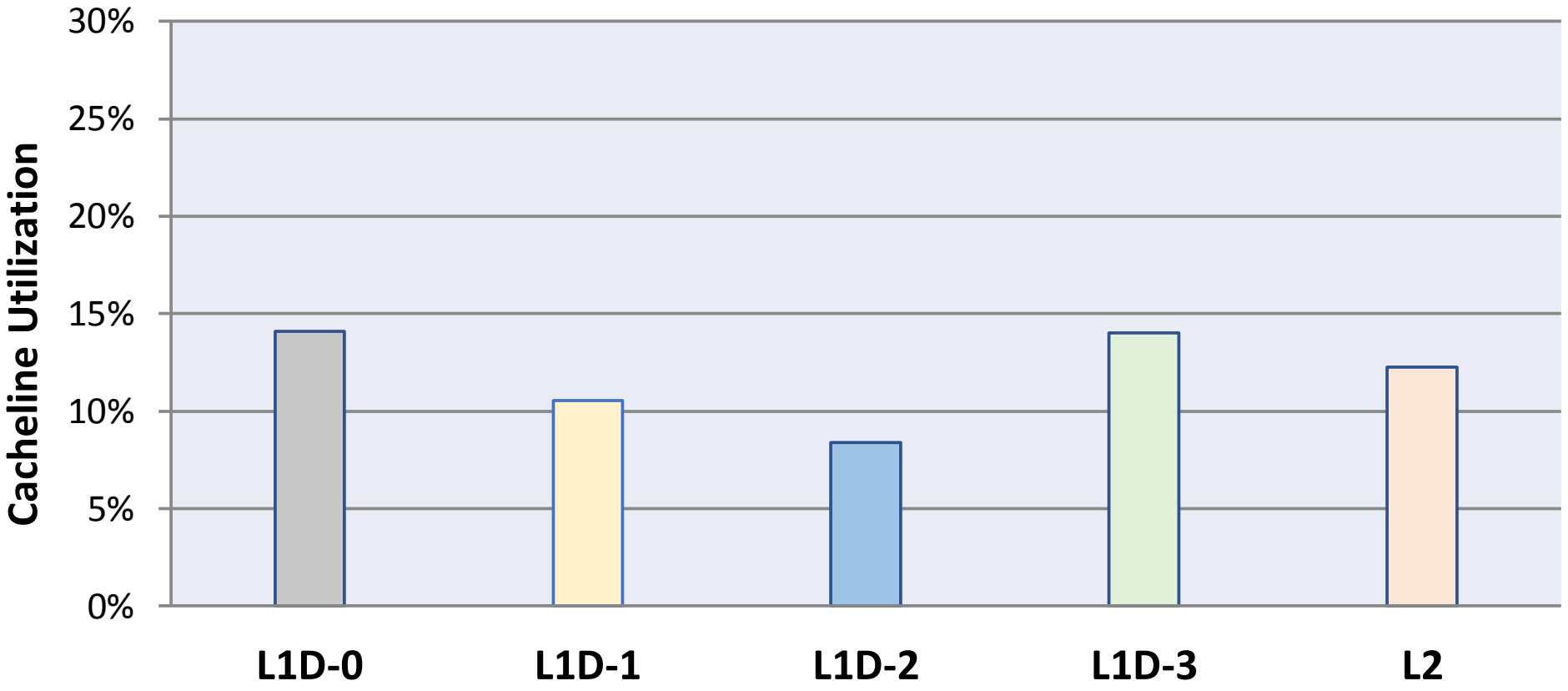


Some Real Applications – Linked List



arm

Some Real Applications – Others

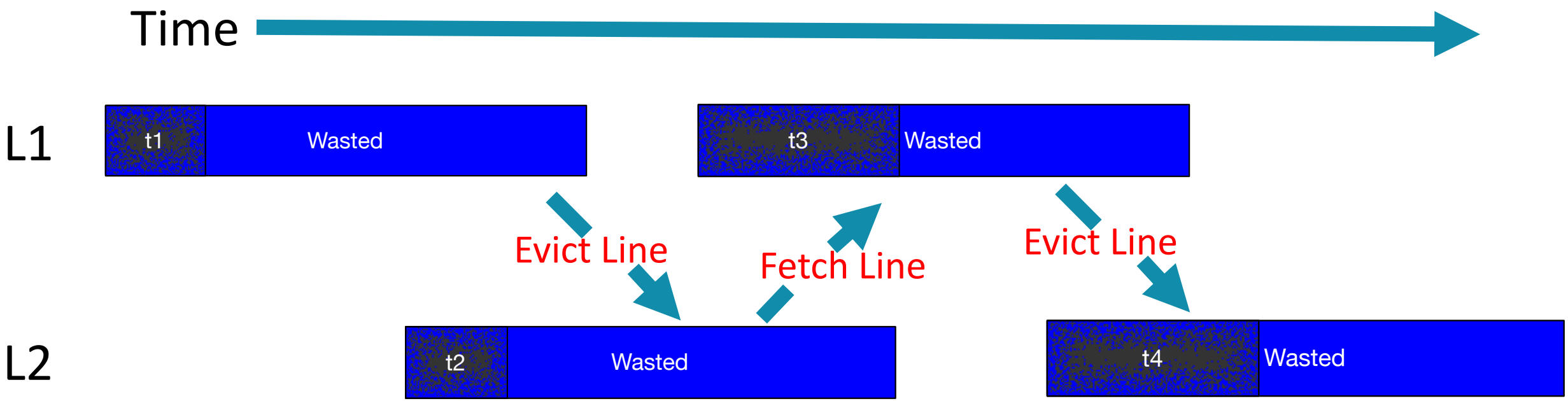


Unused cache data

- *GUPS*: 80%
- *CoMD*: 50%
- *mcb*: 40%
- *LULESH*: 20%
- *DGEMM:* 10%

arm

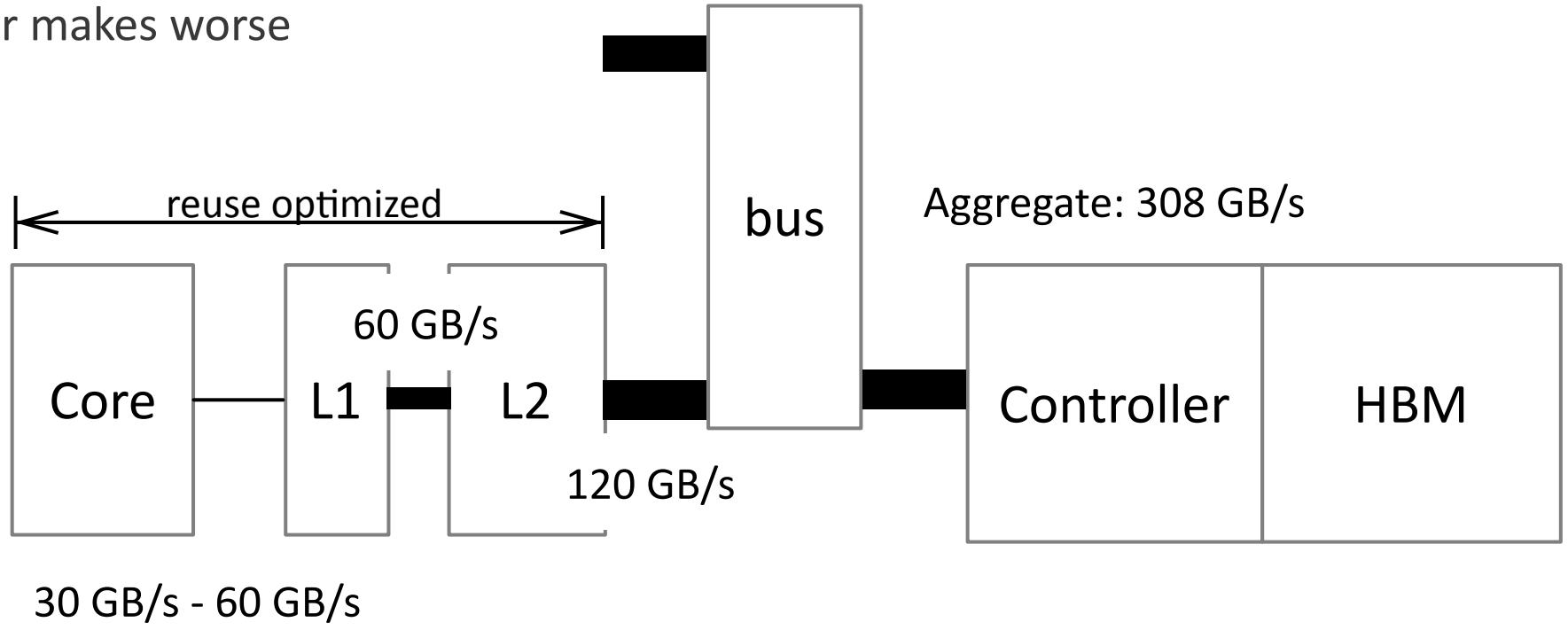
Dance of reuse and utilization



arm

Dark Bandwidth

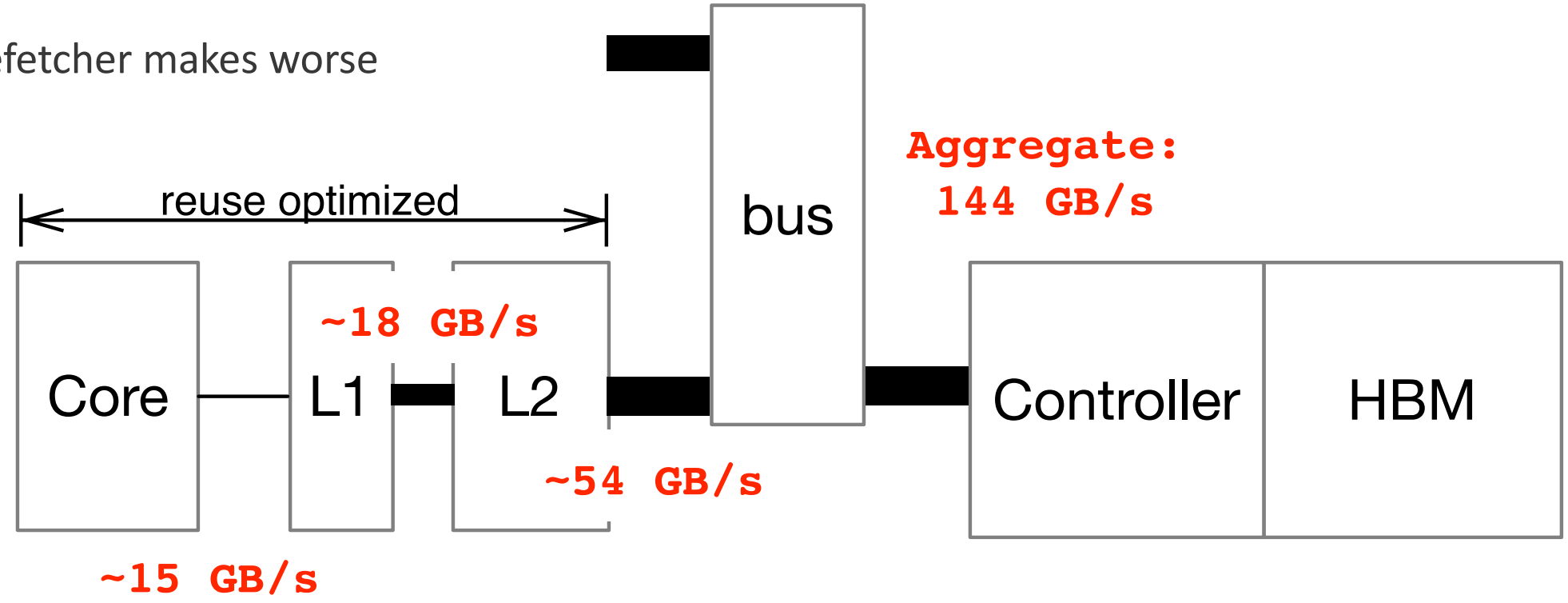
- Here's what we're actually getting (pointer-chasing application)
- Avg. 12.5% line utilization
- Prefetcher makes worse



arm

Dark Bandwidth

- Here's what we're actually getting (pointer-chasing application)
- Avg. 12.5% line utilization
- Prefetcher makes worse

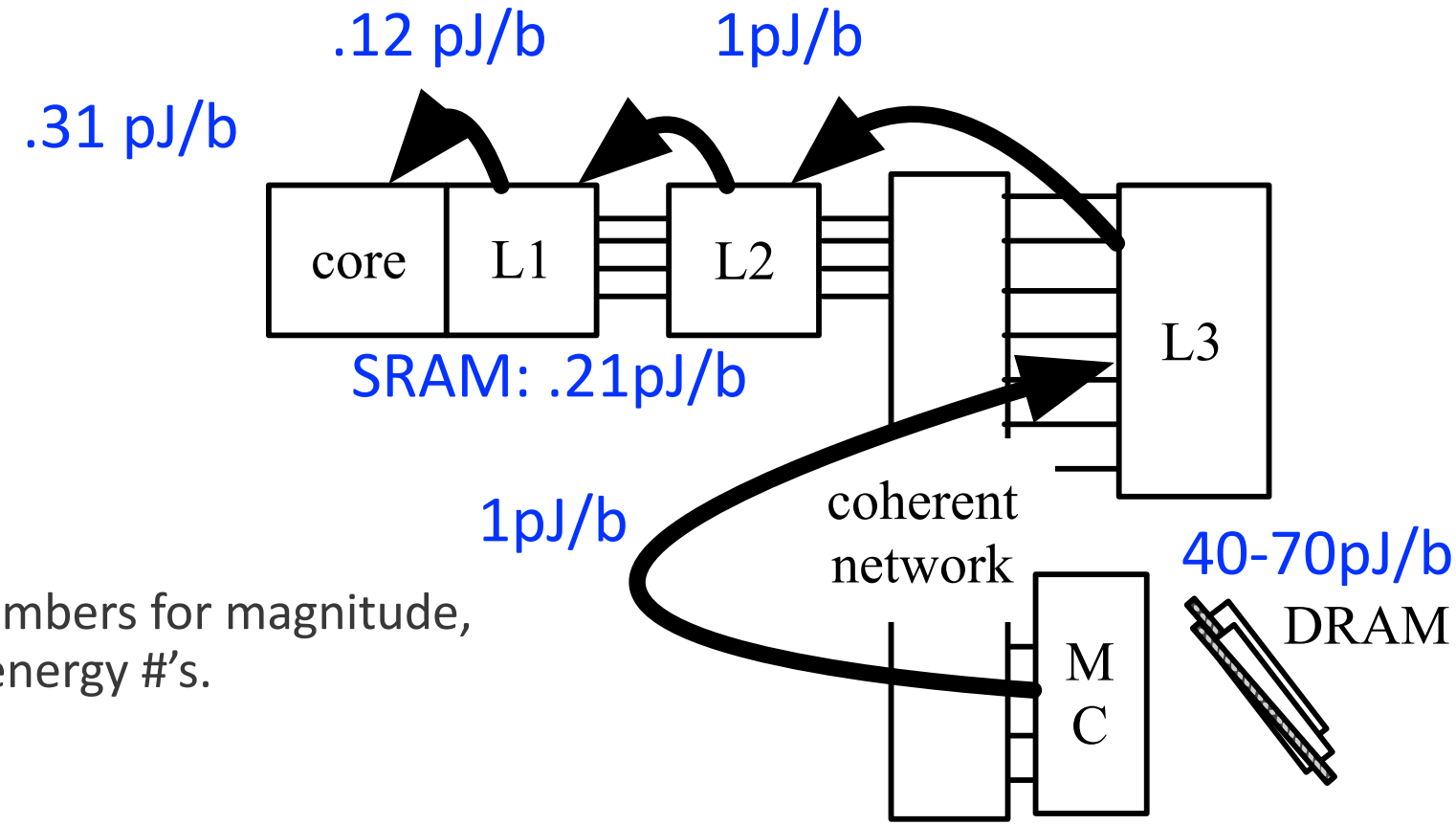


arm

Dark Bandwidth:

Unused data moved throughout the memory hierarchy not at request of programmer but because of hardware design.

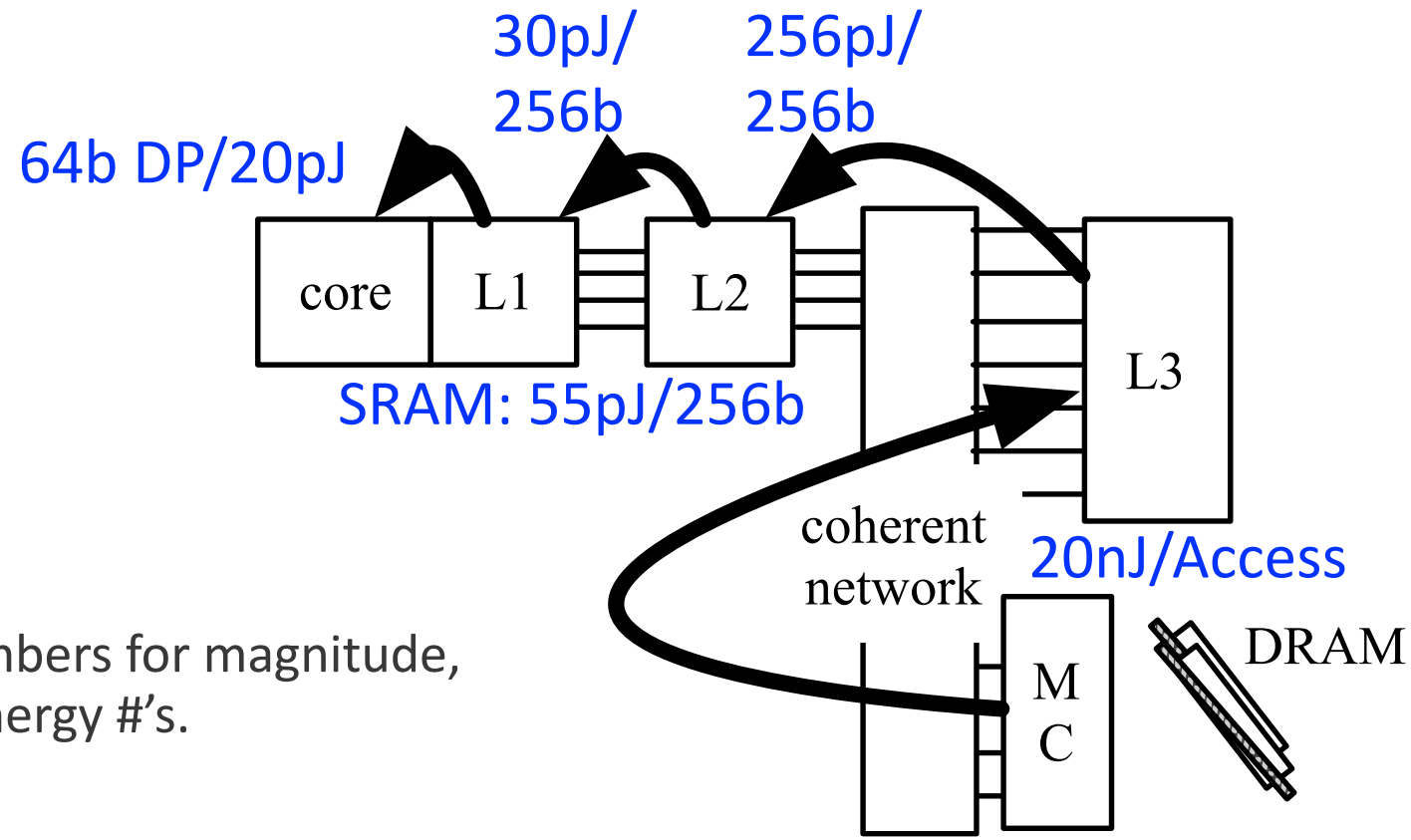
What this means for power



- Example numbers for magnitude, not actual energy #'s.

arm

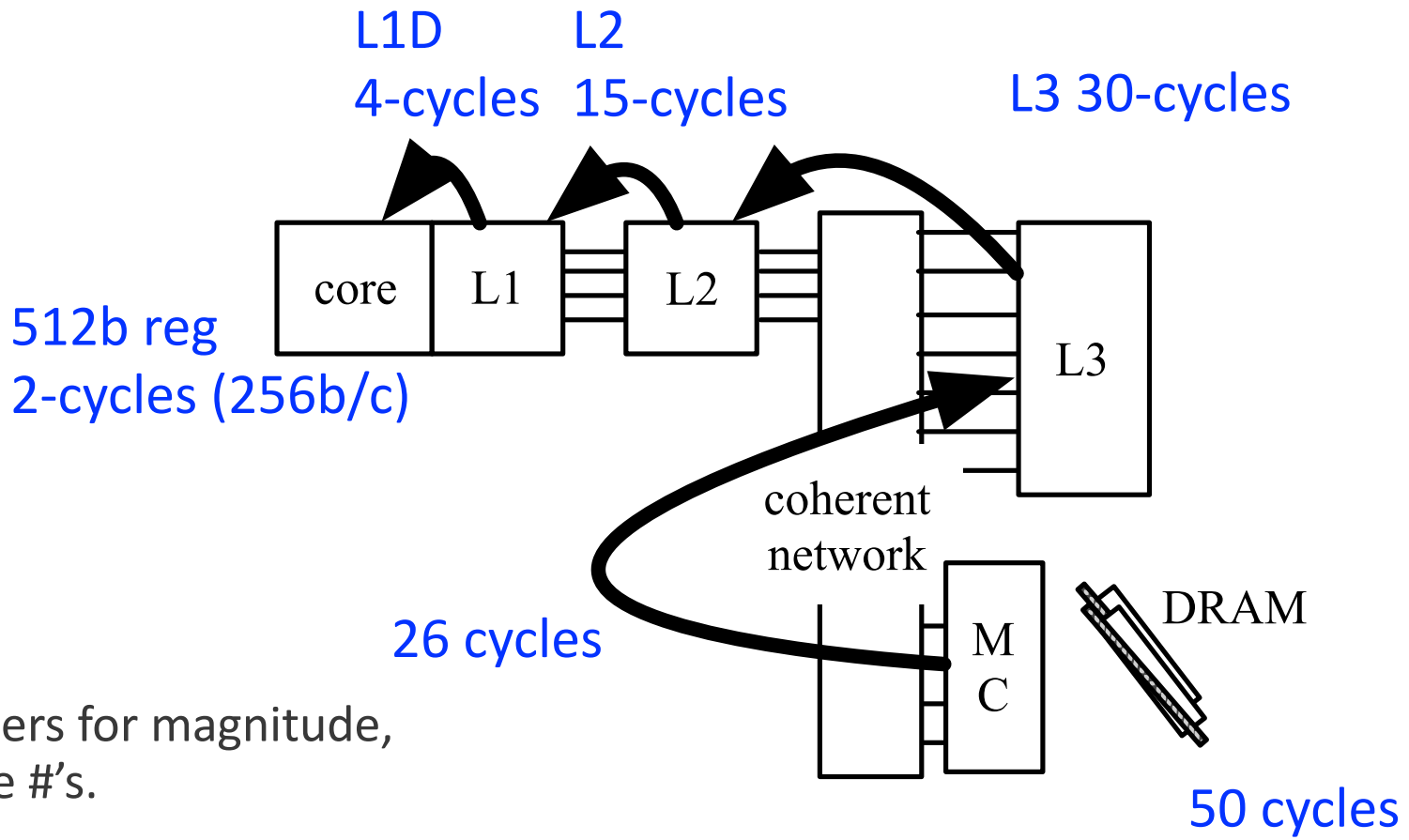
What this means for power



- Example numbers for magnitude, not actual energy #'s.

arm

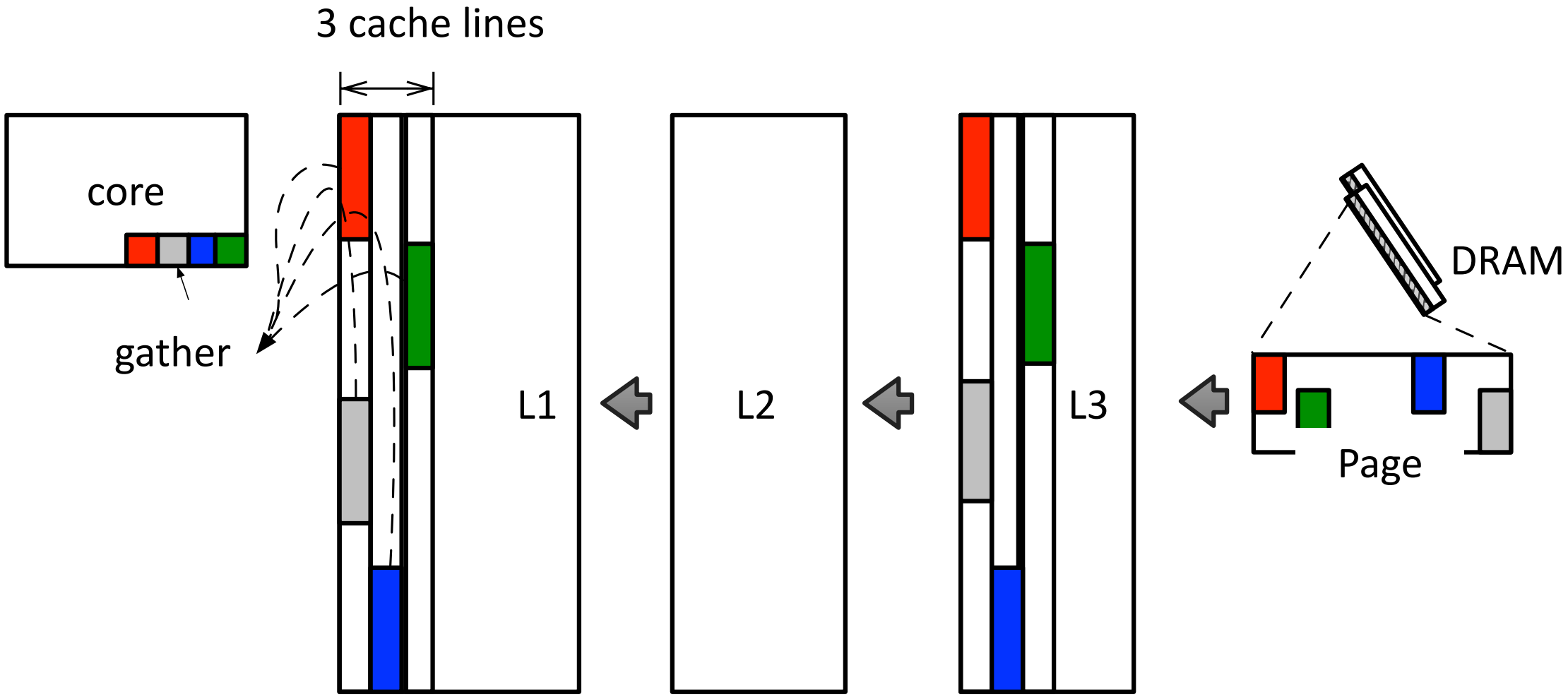
What this means for time to solution



- Example numbers for magnitude, not actual cycle #'s.

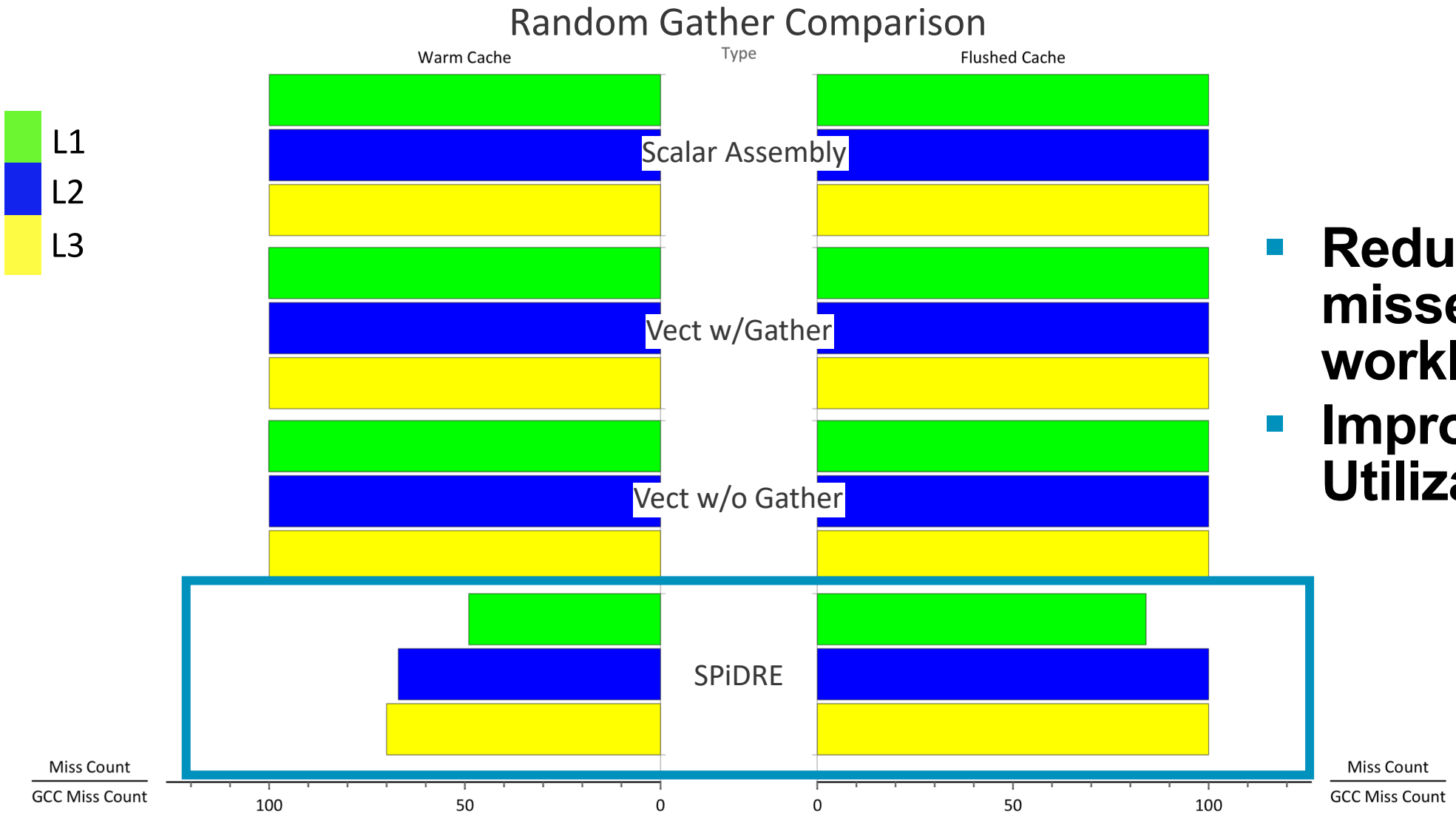
Extant options to solve the problem

Conventional Gather/Scatter



arm

Improving on scatter/gather



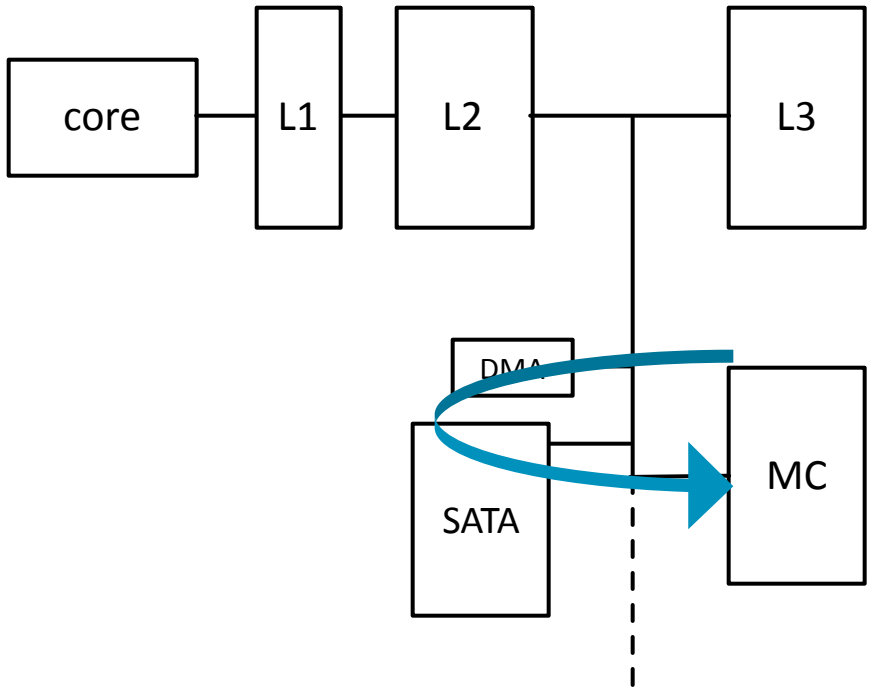
- **Reduce cache misses for sparse workloads**
- **Improve Bandwidth Utilization**

arm

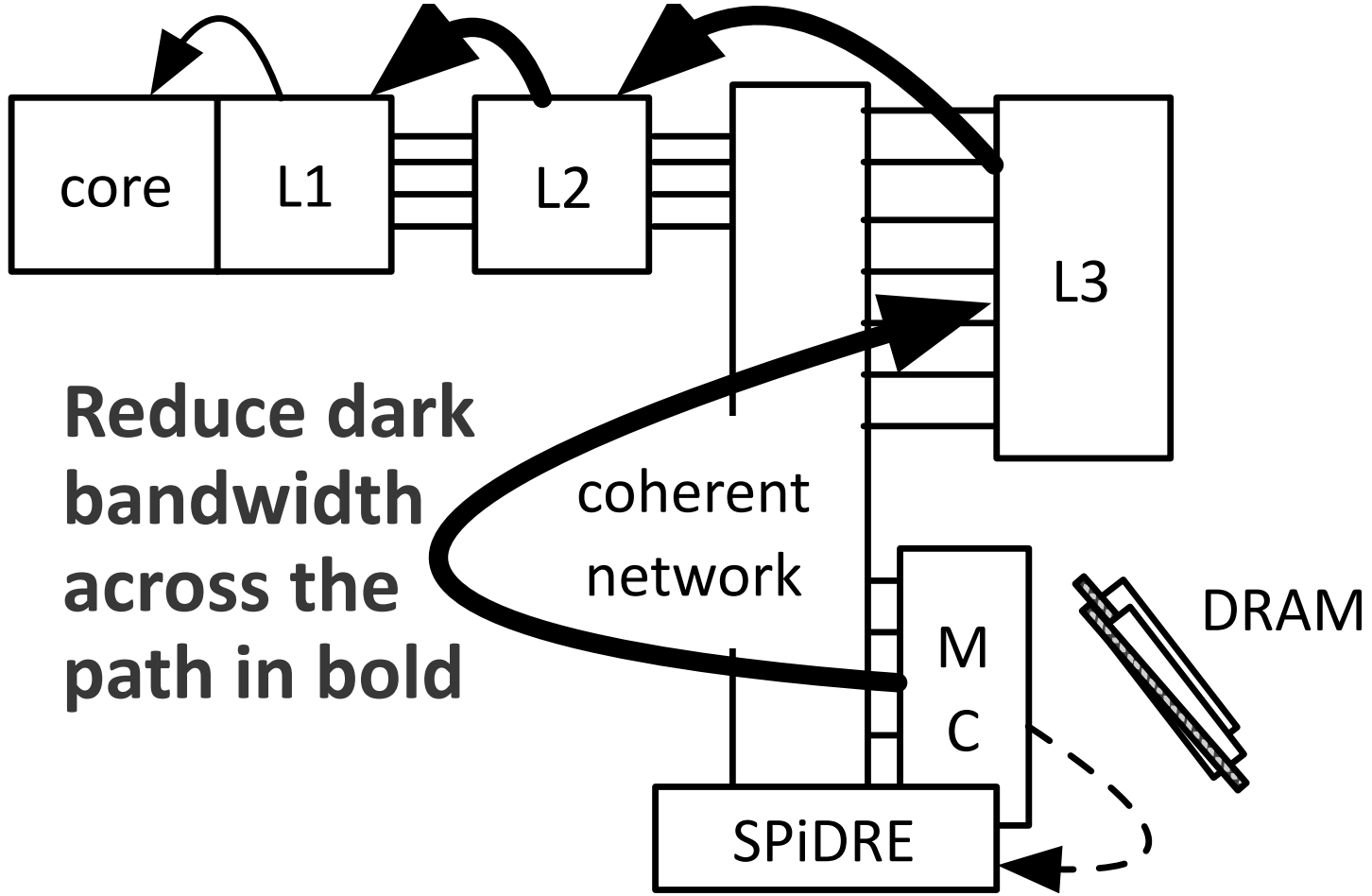
Programmable DMA

- Generally limited to page gather/move/scatter
- Usually coherent (lots of transactions), wastes bus utilization (i.e., all traffic goes through coherent network)
- Often limited when it comes to finer granularity
- Typically no persistent state
- Not solving quite the same problem!

Toy Example:



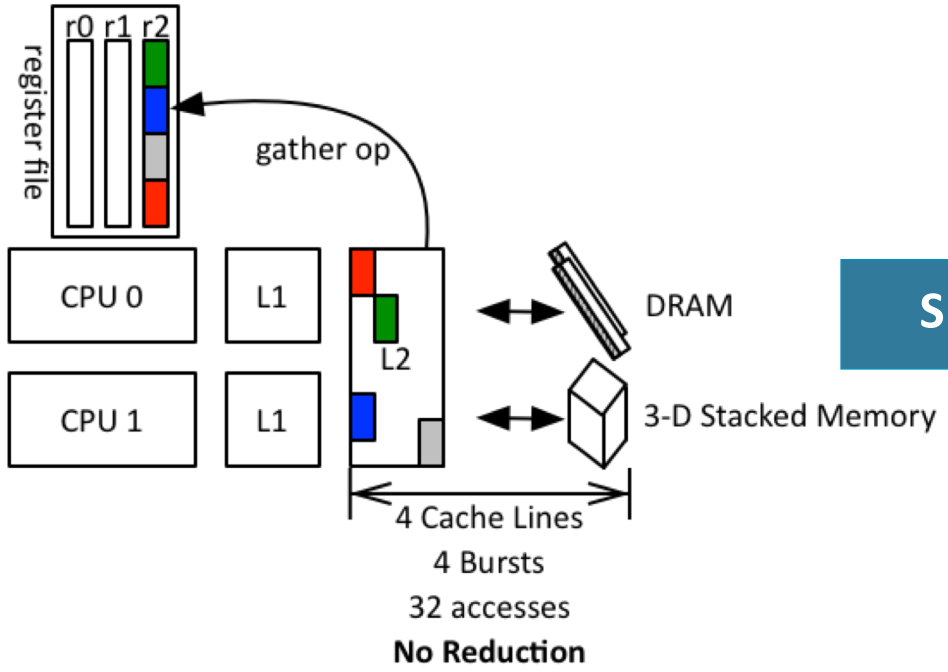
What we need to do



arm

Sparse Data Reduction (mechanisms)

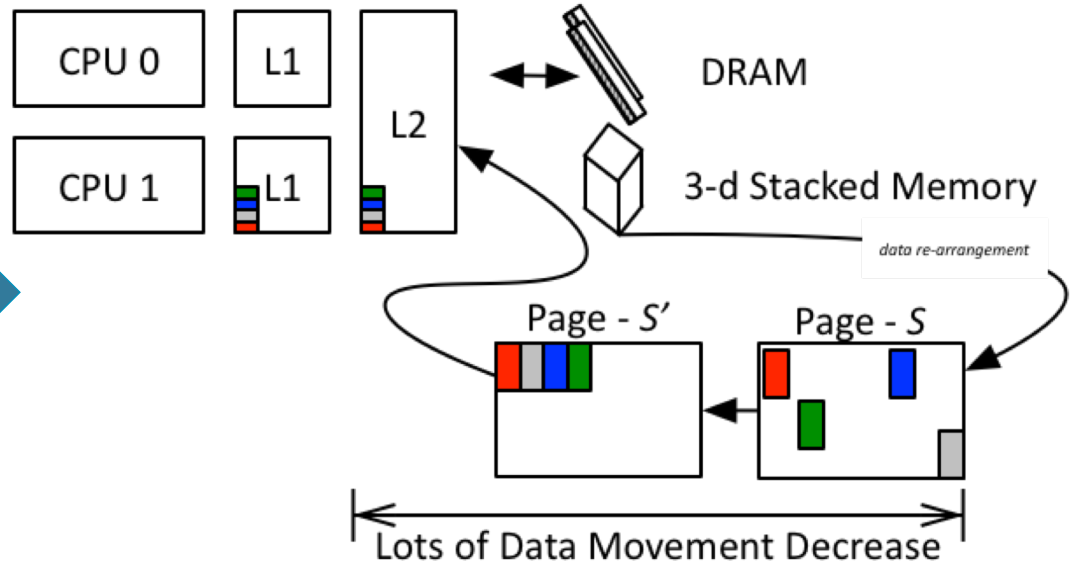
Standard Gather-Scatter (In Core)



- Smaller binary (e.g., single gather vs. multiple loads)

SPDRE

In-Memory Gather-Scatter



- Smaller binary (e.g., single gather vs. multiple loads)
- Improved Cache Line Utilization
- Reduced Overall Data Movement

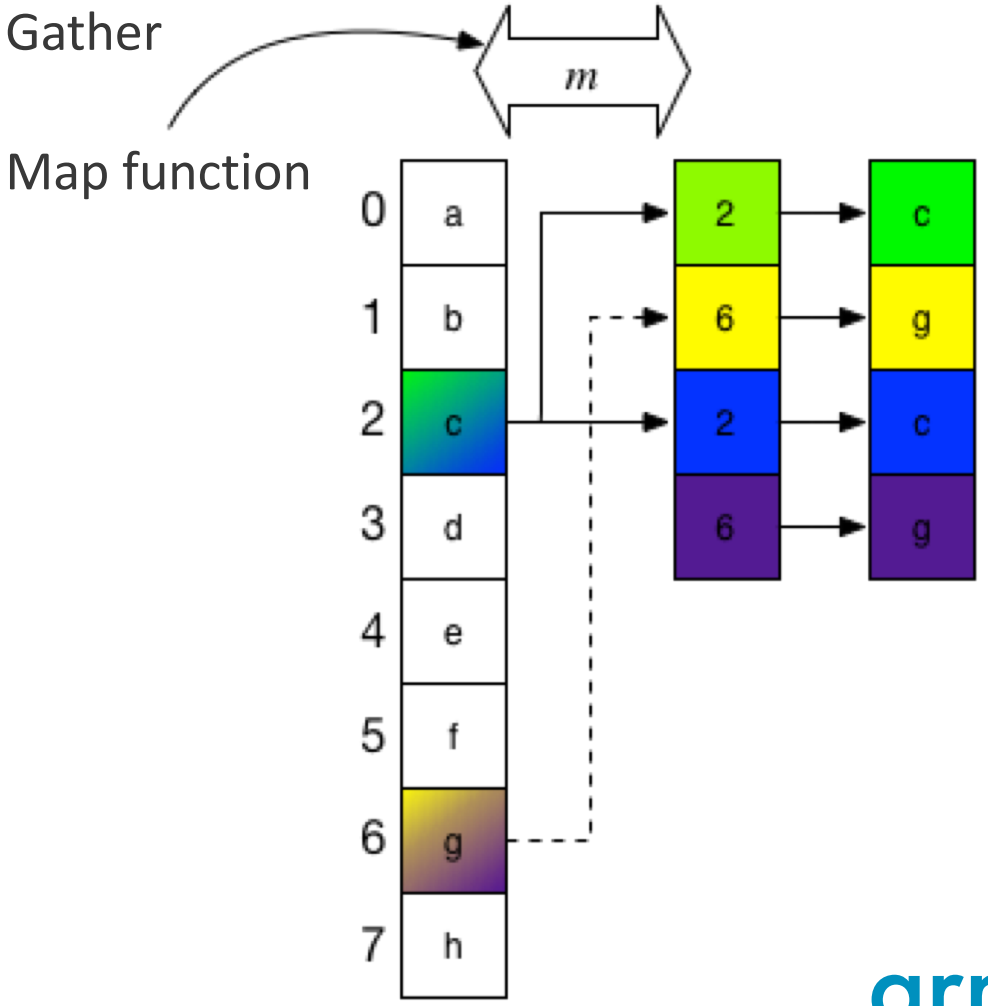
arm

SPiDRE in words

Decoupling access and execute

The idea

- Programmers know what data they want (usually), “SQL for Memory”
- Finding Compressed Sparse Row manipulations for new problems often takes a lot of time (bad for productivity)
- Why not build an interface that allows gathering (potentially) in any device of only the data you (the programmer) needs
- “Bulk” byte-addressed memory
- Doesn’t break coherence…with some caveats
- Can create multiple windows of original segment



arm

The SPiDRE

Programming SPiDRE (one approach)

User inserted allocate (new virtual and physical space)

```
template < typename TYPE >
   static void alloc( TYPE **output,
                      const std::size_t nitems );
```

User inserted rearrange function

```
template < class DST, class SRC >
   static std::size_t rearrange(
        DST * const dst,
        SRC * const src,
        const std::size_t nitems,
        rearrange_func_t< DST, SRC > src_dst );
```

```
template < class S, class S_Prime >
   static void sync( S       * const s,
                     S_Prime * const s_p,
                     const std::size_t nitems );
```

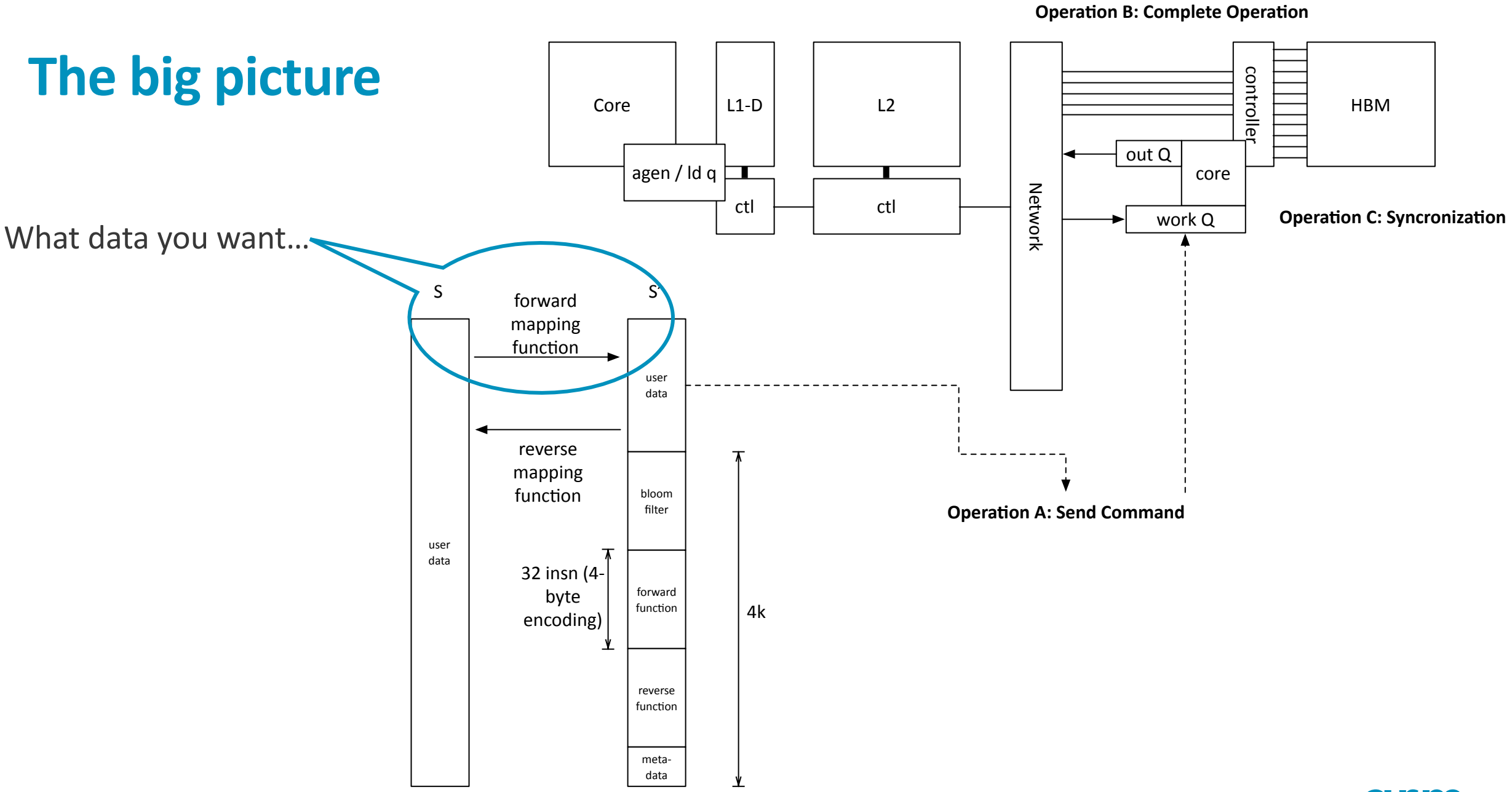
Synchronize S' -> S

free

```
template < typename TYPE >
   static void free( TYPE *ptr );
```

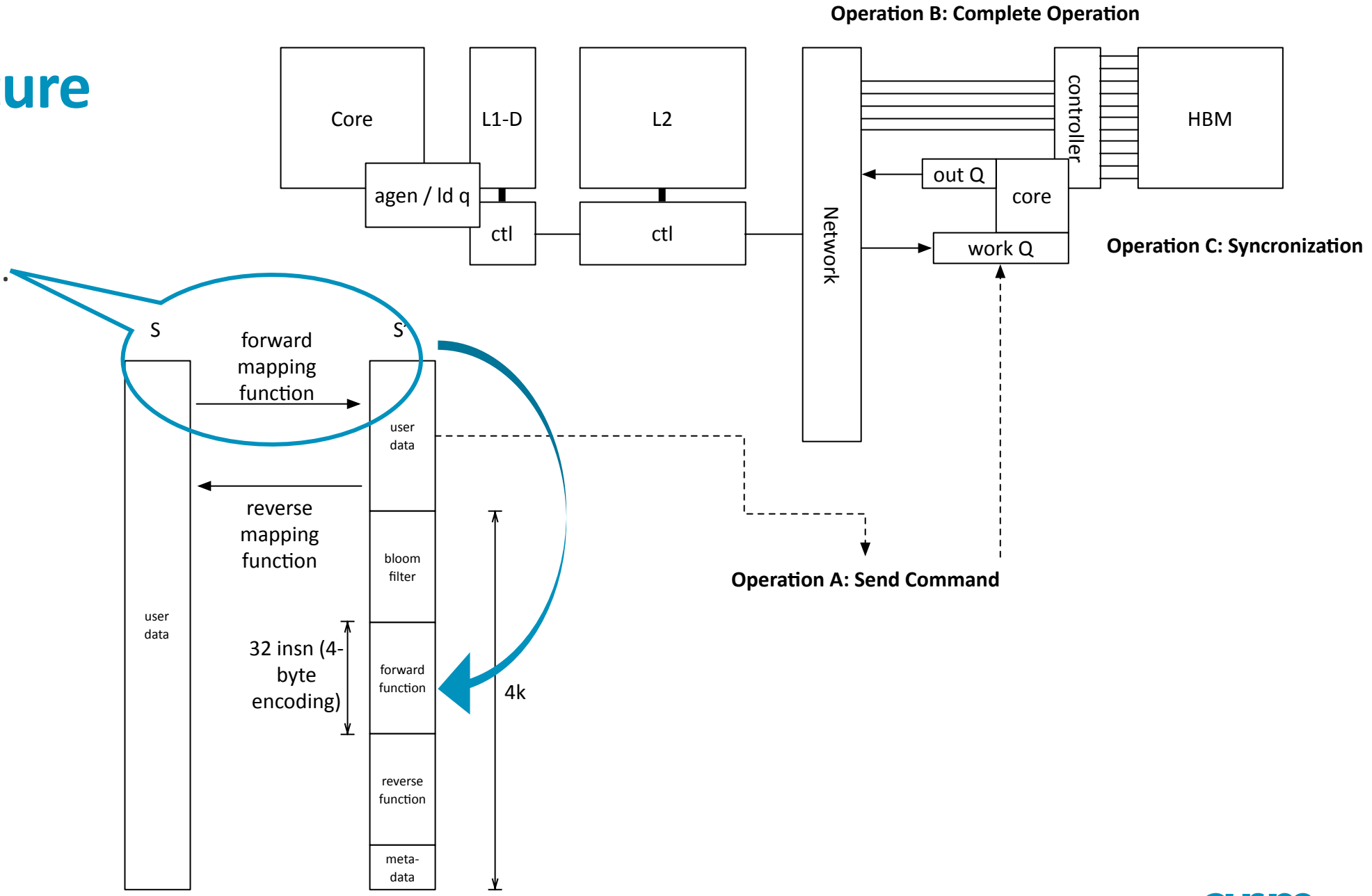

arm

The big picture

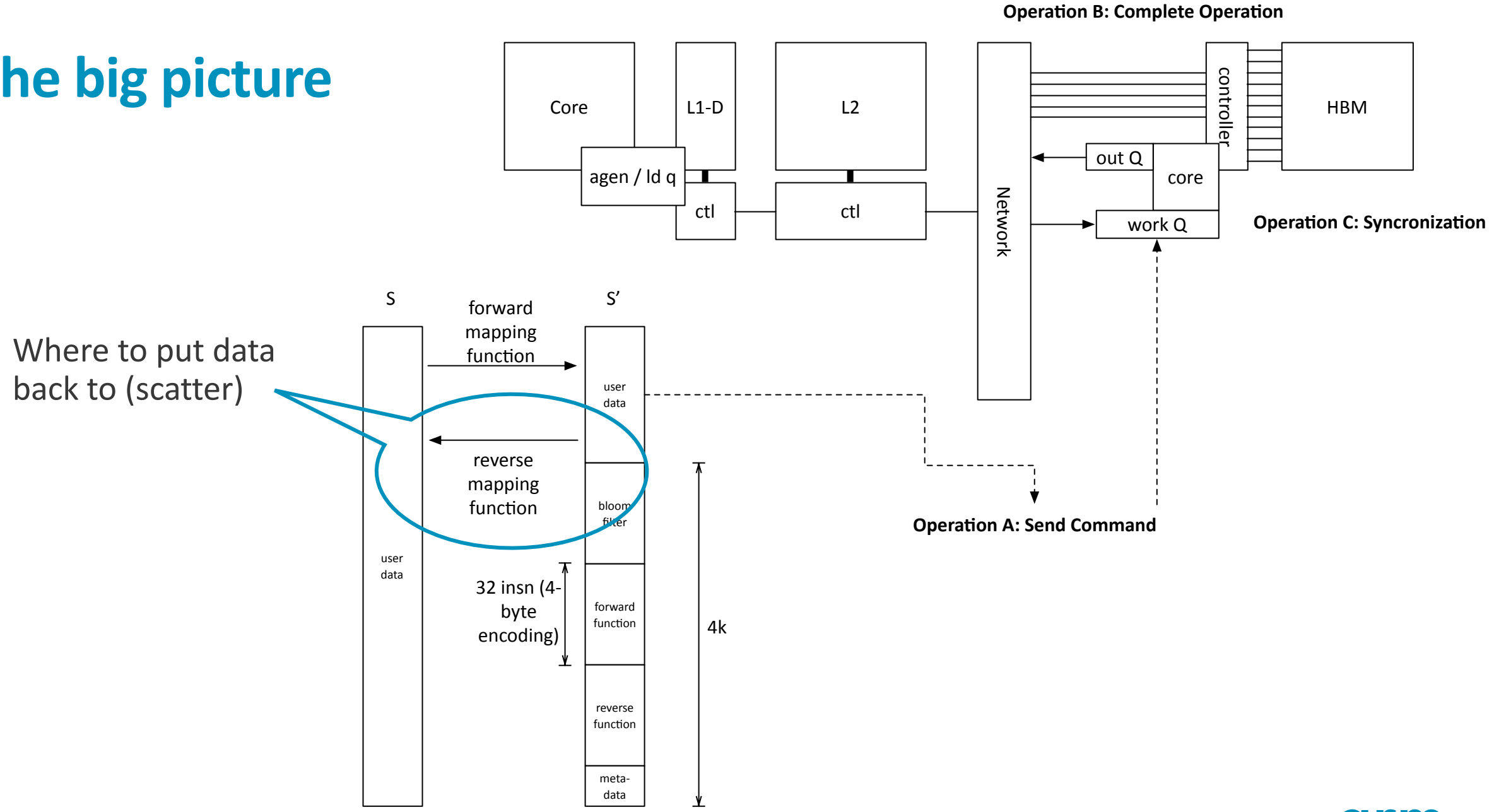


The big picture

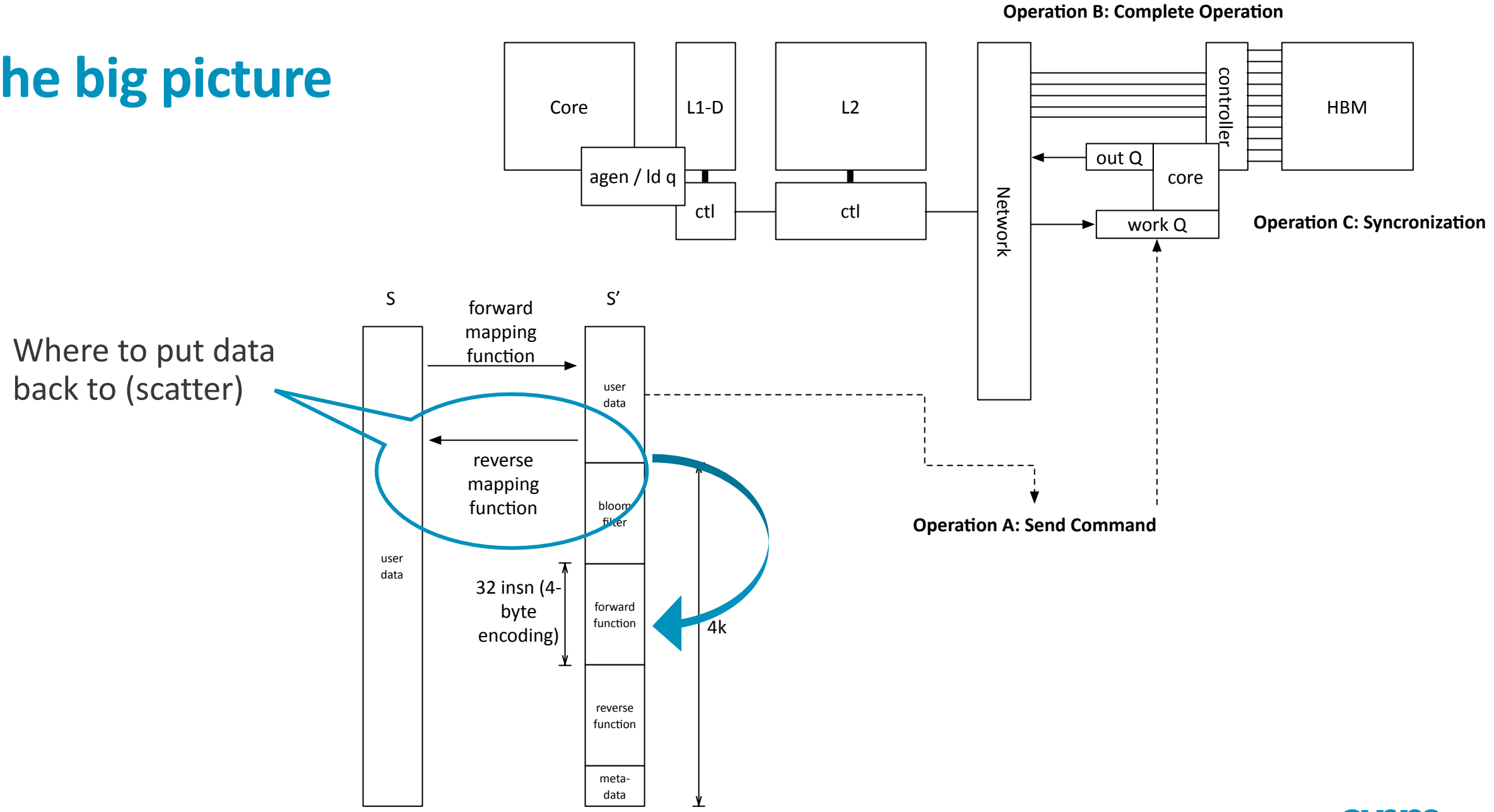
What data you want...



The big picture

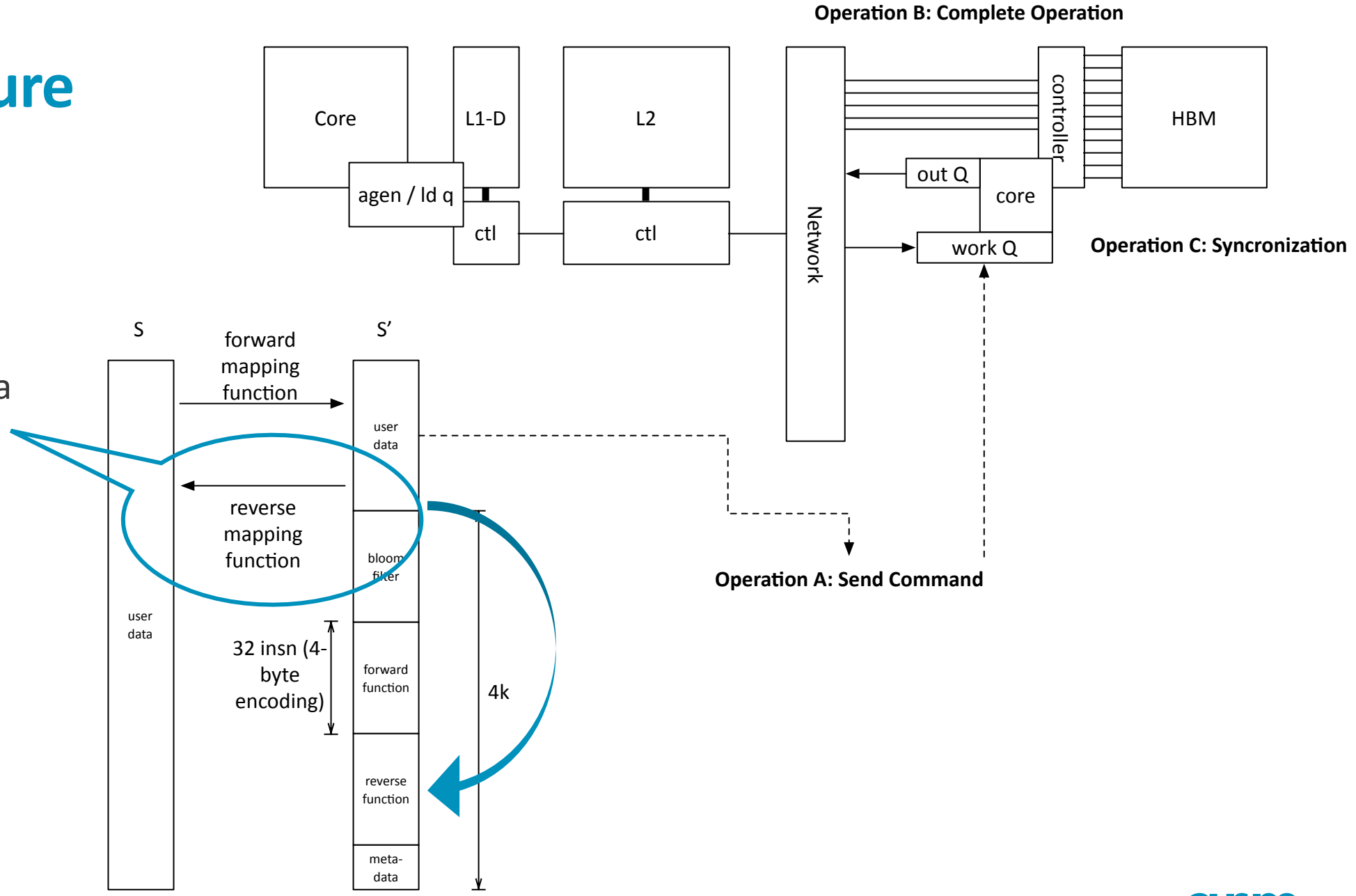


The big picture



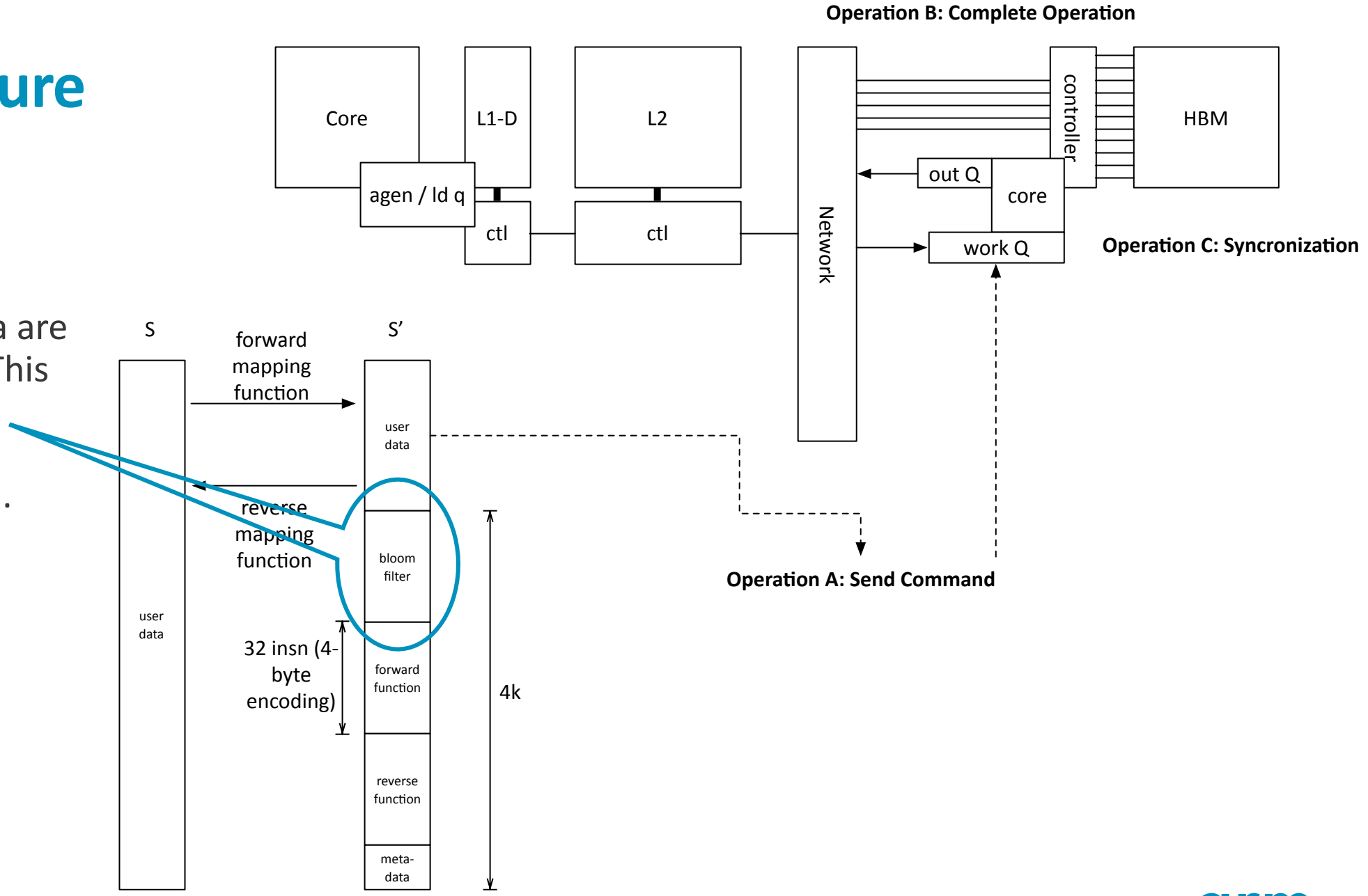
The big picture

Where to put data back to (scatter)

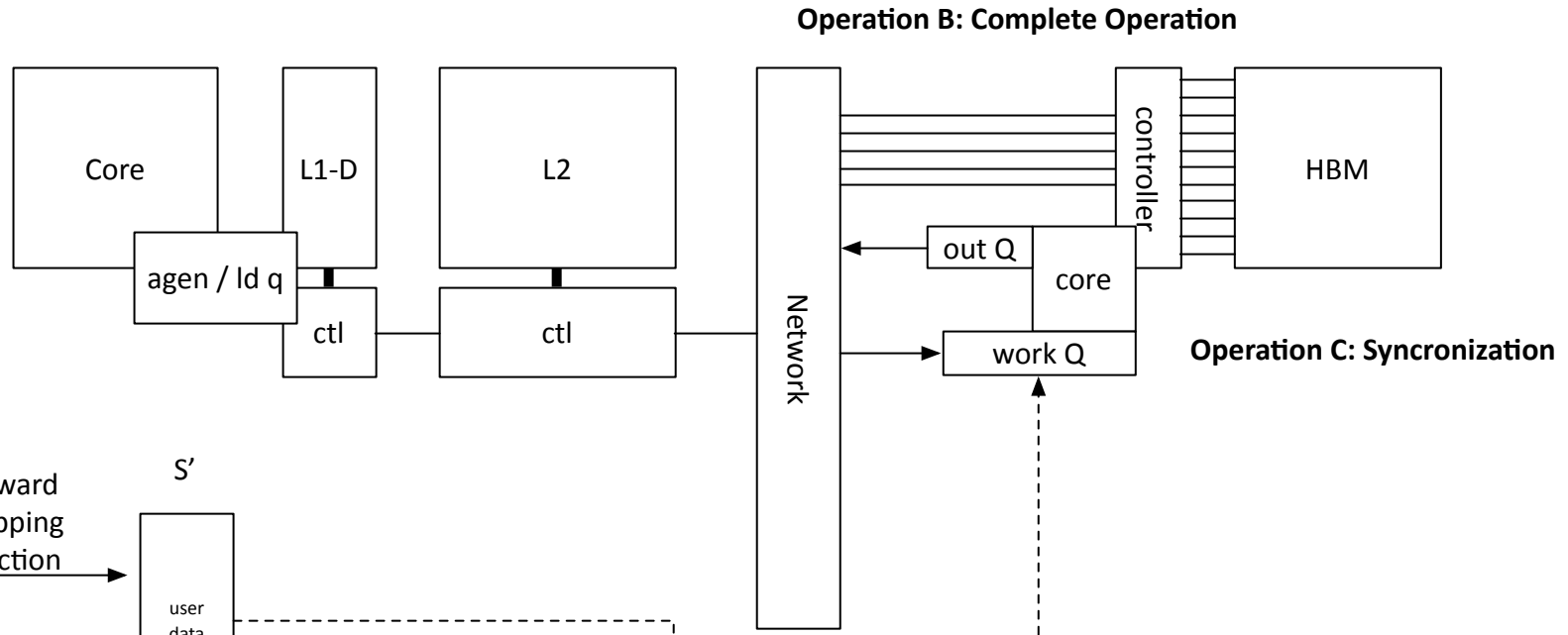


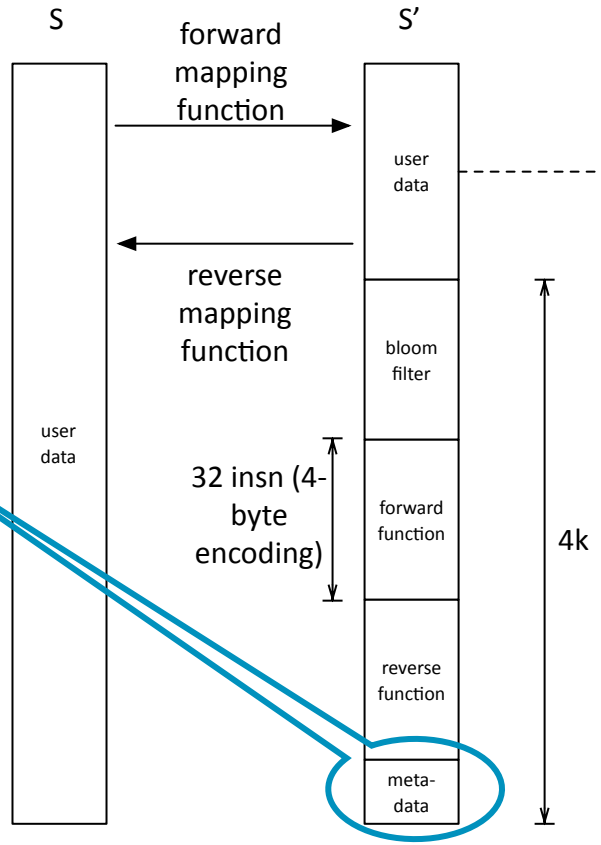
The big picture

Save which data are modified in S'. This enables more efficient synchronization.



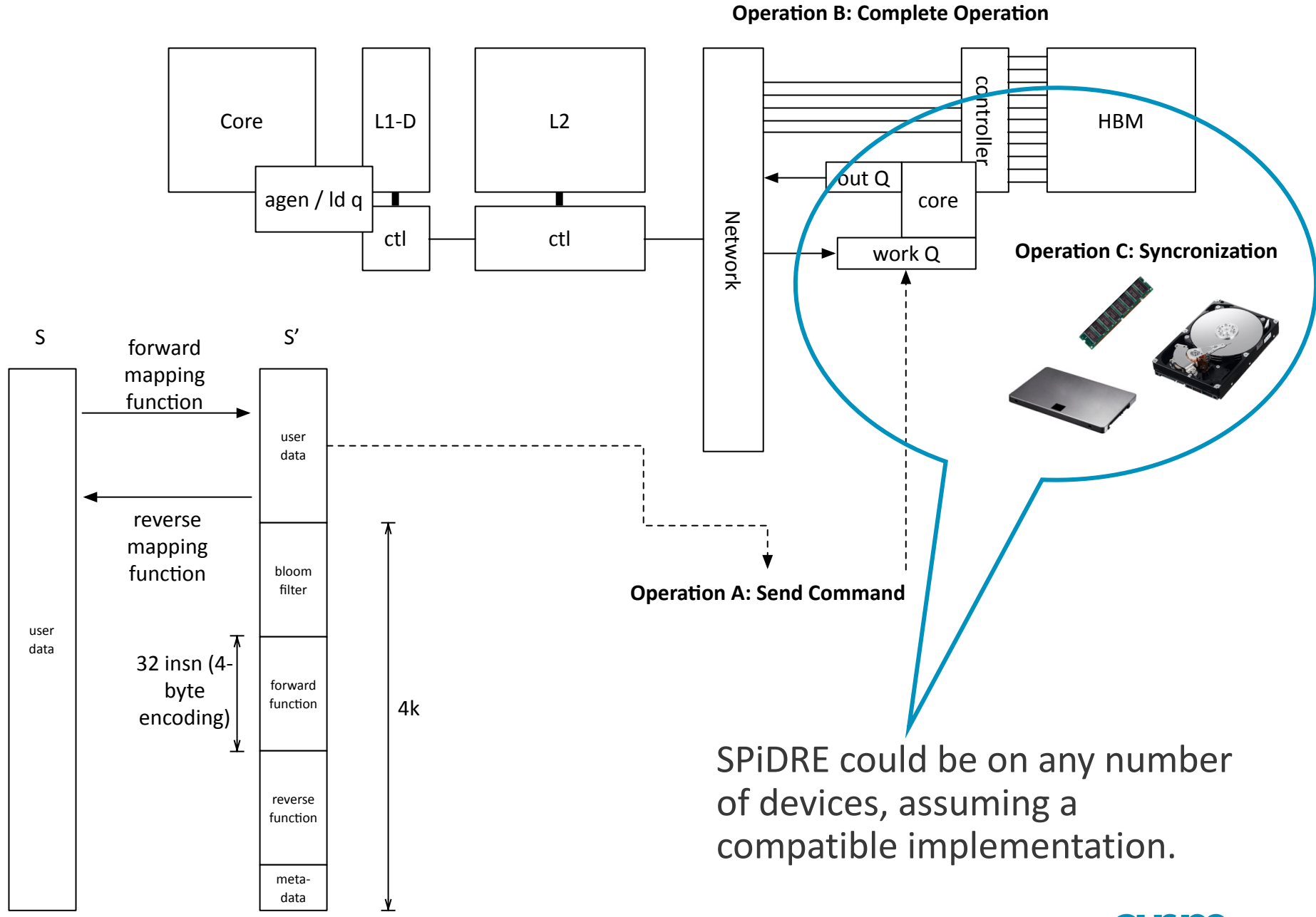
The big picture





A bit of meta data, sizing, etc.

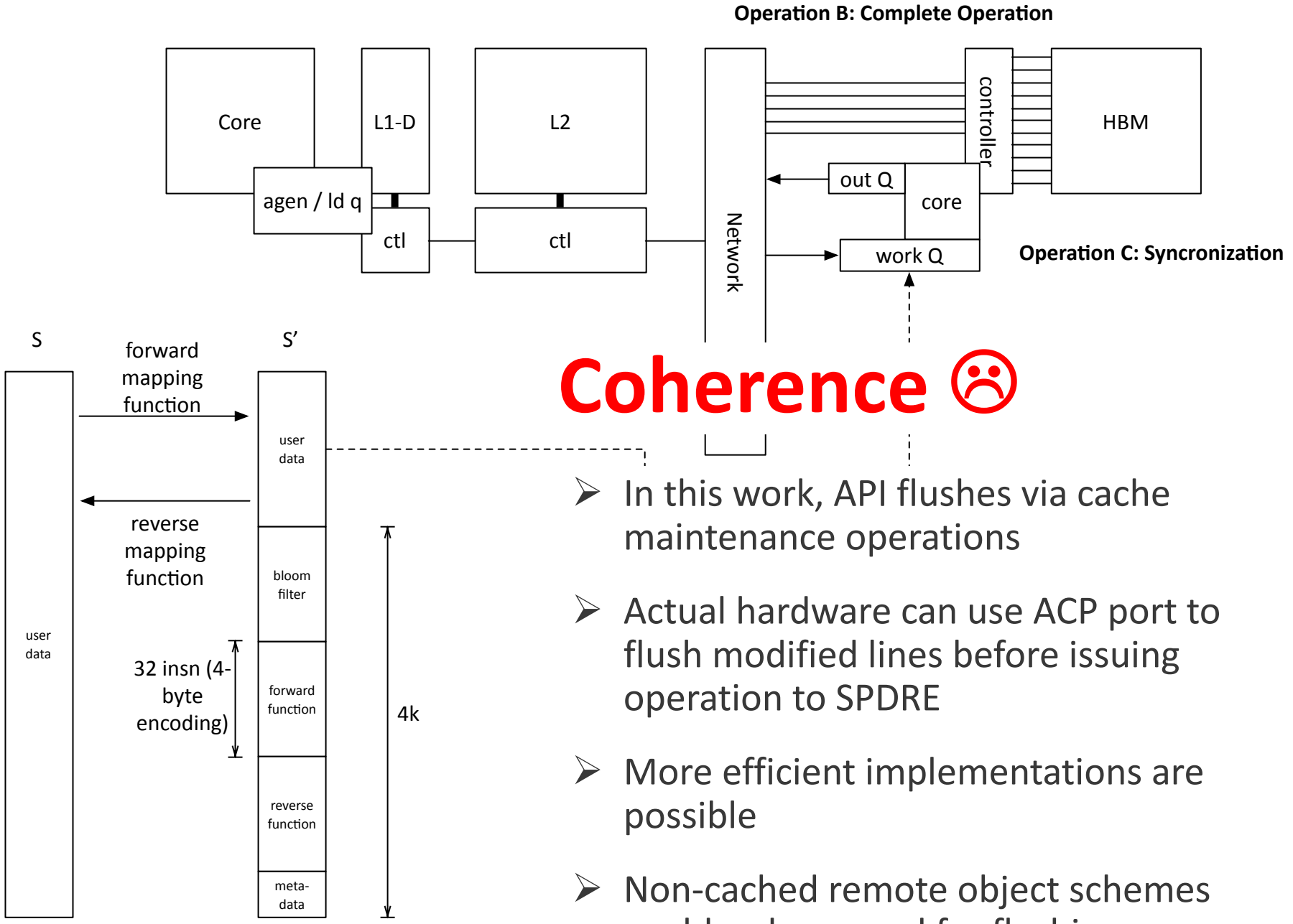
The big picture



SPiDRE could be on any number of devices, assuming a compatible implementation.

The big picture

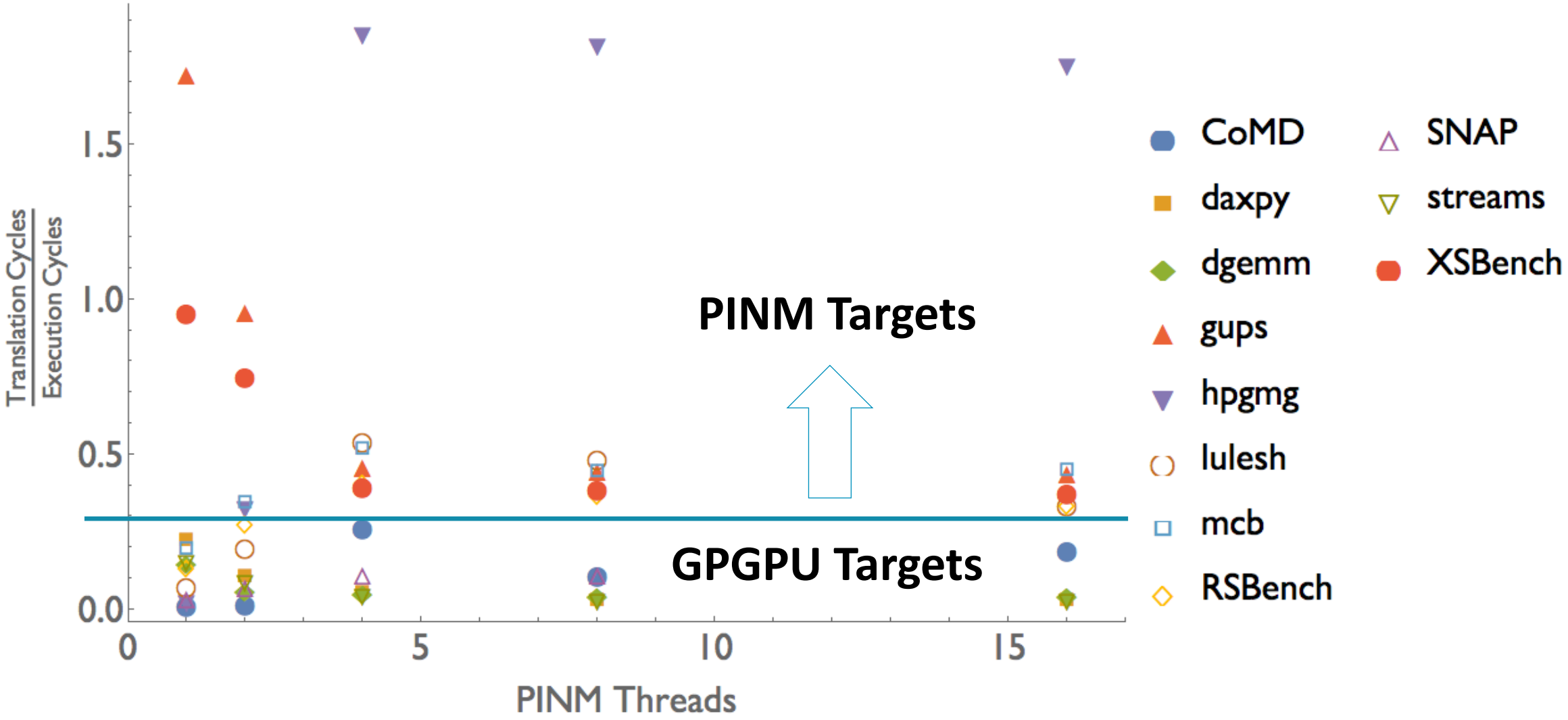
S marked as read only while **S'** exists



Coherence ☹

- In this work, API flushes via cache maintenance operations
- Actual hardware can use ACP port to flush modified lines before issuing operation to SPDRE
- More efficient implementations are possible
- Non-cached remote object schemes could reduce need for flushing

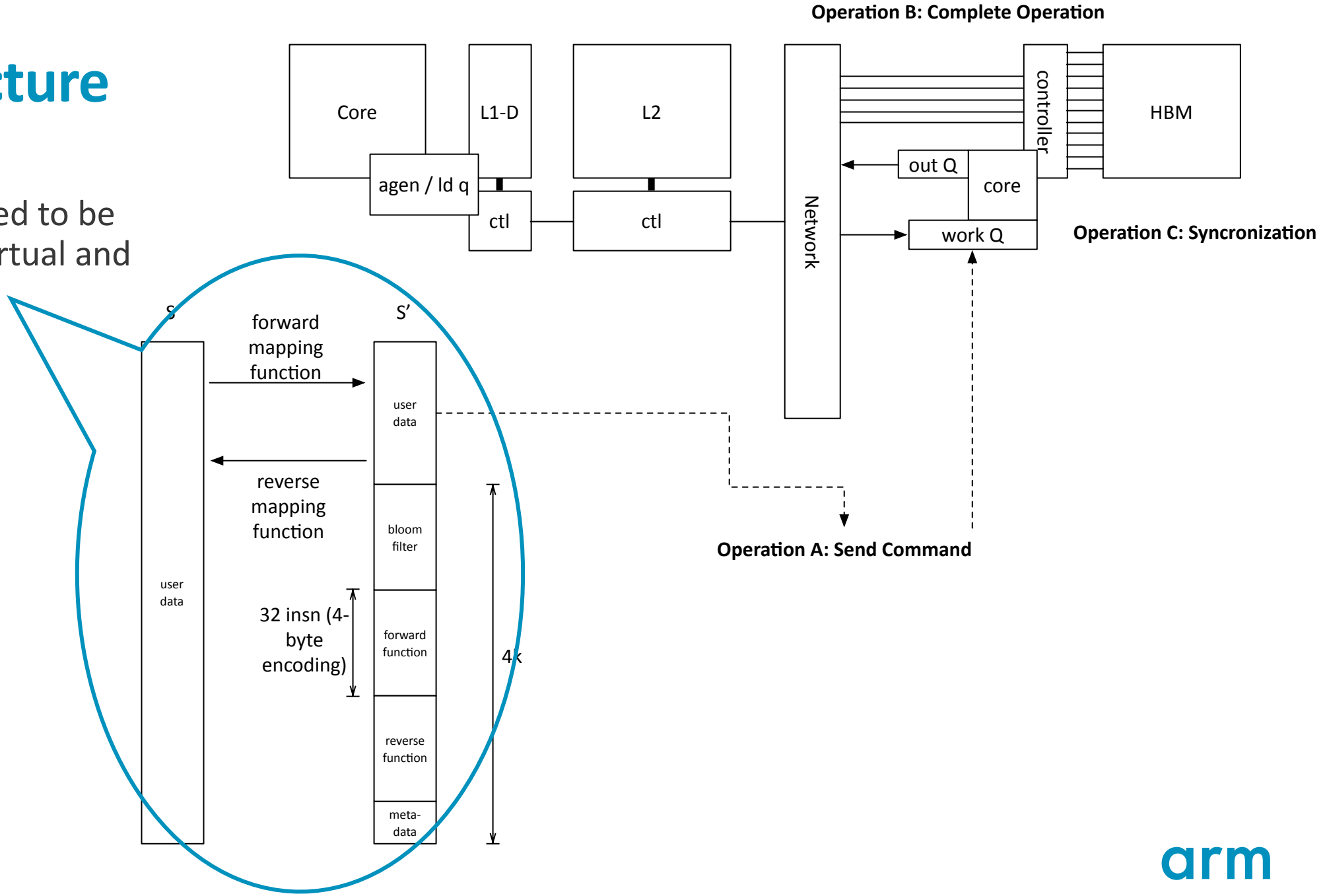
Sparse translation through IOMMU



arm

The big picture

Must be allocated to be contiguous in virtual and physical space…



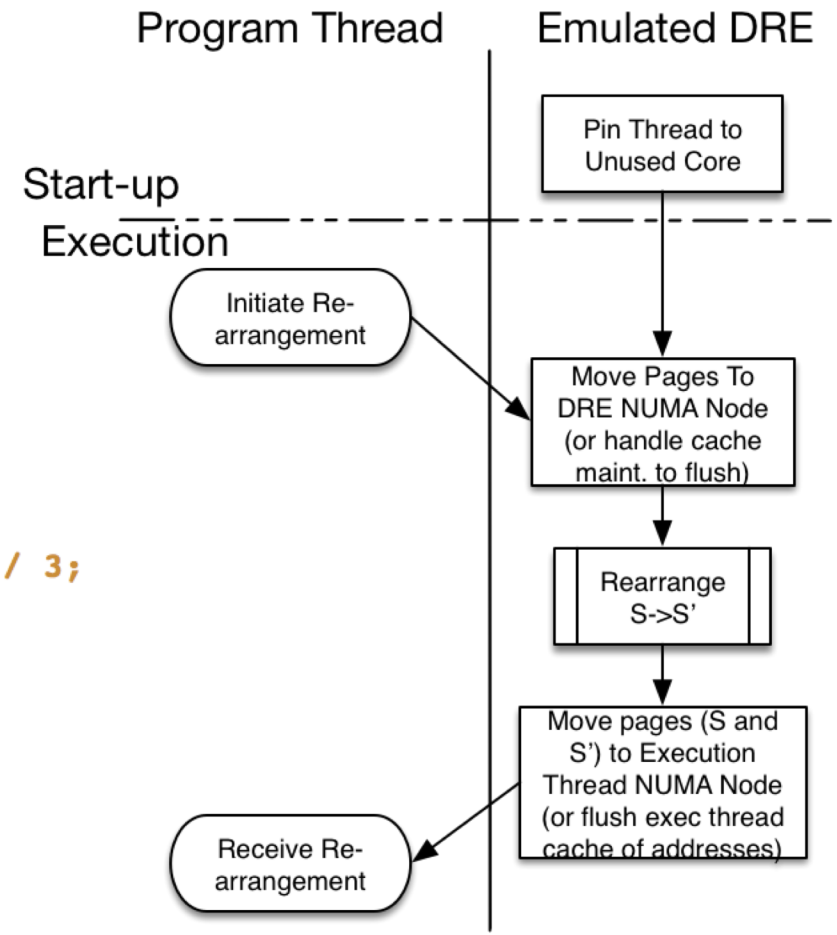
Sim Setup & Results

Emulation Environment (SPiDRE)

```
/** allocate DRE memory **/
dre::alloc( dst, source_length );
const auto num_rearranged(  dre::rearrange( *dst, src, source_length, offset ) );
for( auto dst_index( 0 ); dst_index < ret_val; dst_index++ )
{
  (*dst)[ dst_index ] = workload( (*dst)[dst_index] );
}
/** synchronize  **/
dre::sync( *dst, src, N_SRC );
dre::free( *dst );
dre::release( src, N_SRC );
```

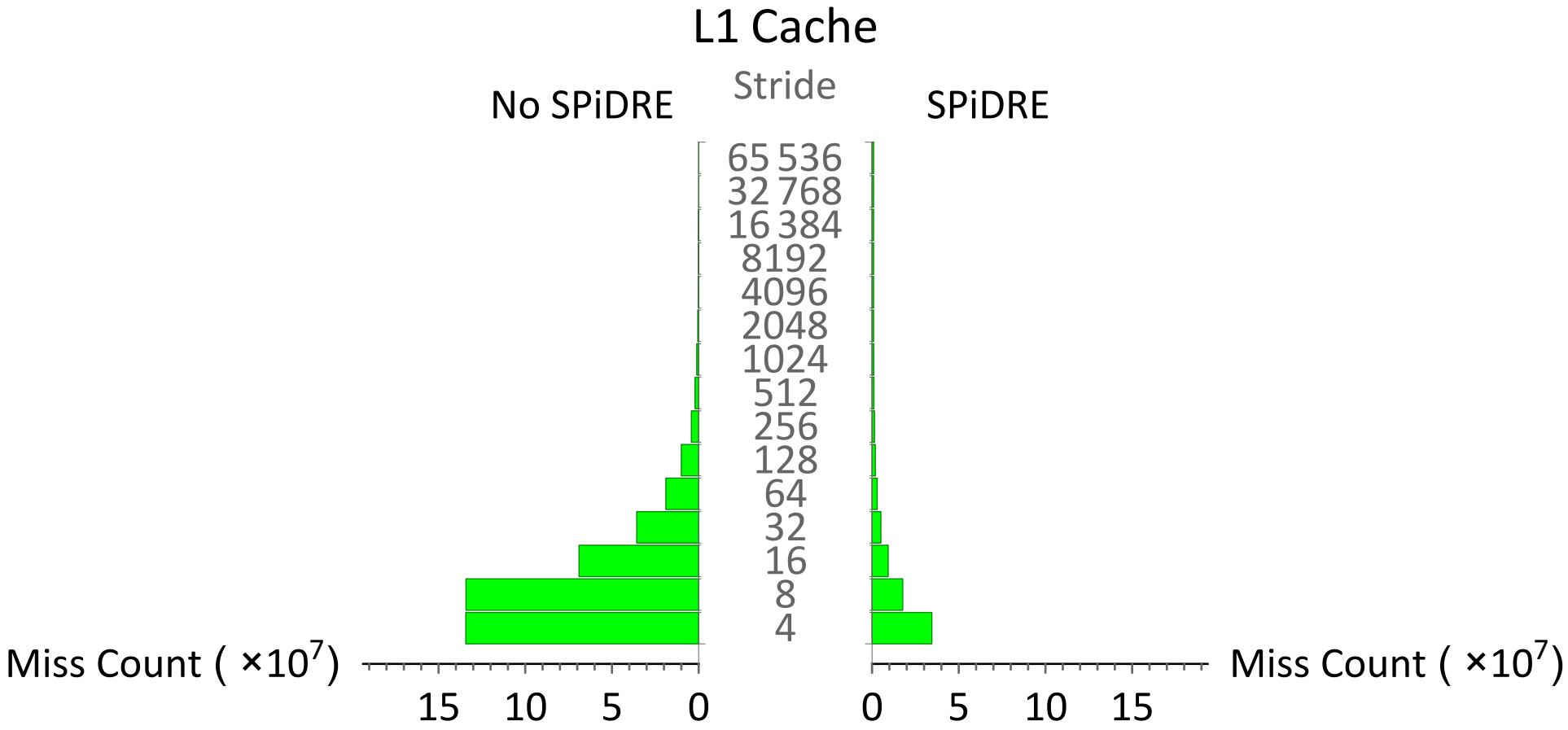
```
#define workload( x )\
   std::exp( static_cast< double>( x )  / 2 ) * 7.0f / 3;
```

Platform	Specifications
Intel	• 2x CPU E5-2690 v3 • 64 GB DDR4 PC4-17000 • Linux Kernel Version 3.13



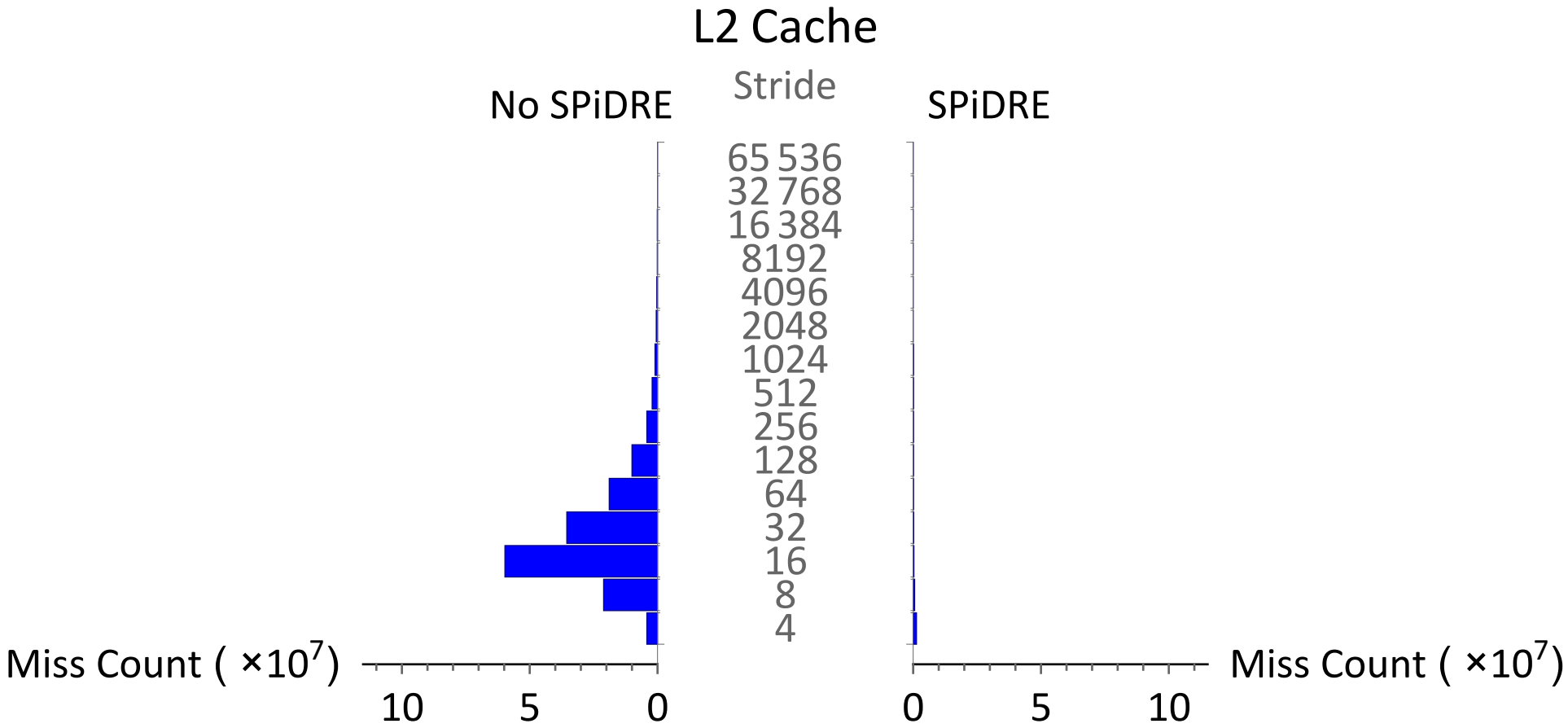
arm

Fixed Stride (1GB data set) - Gather SPiDRE



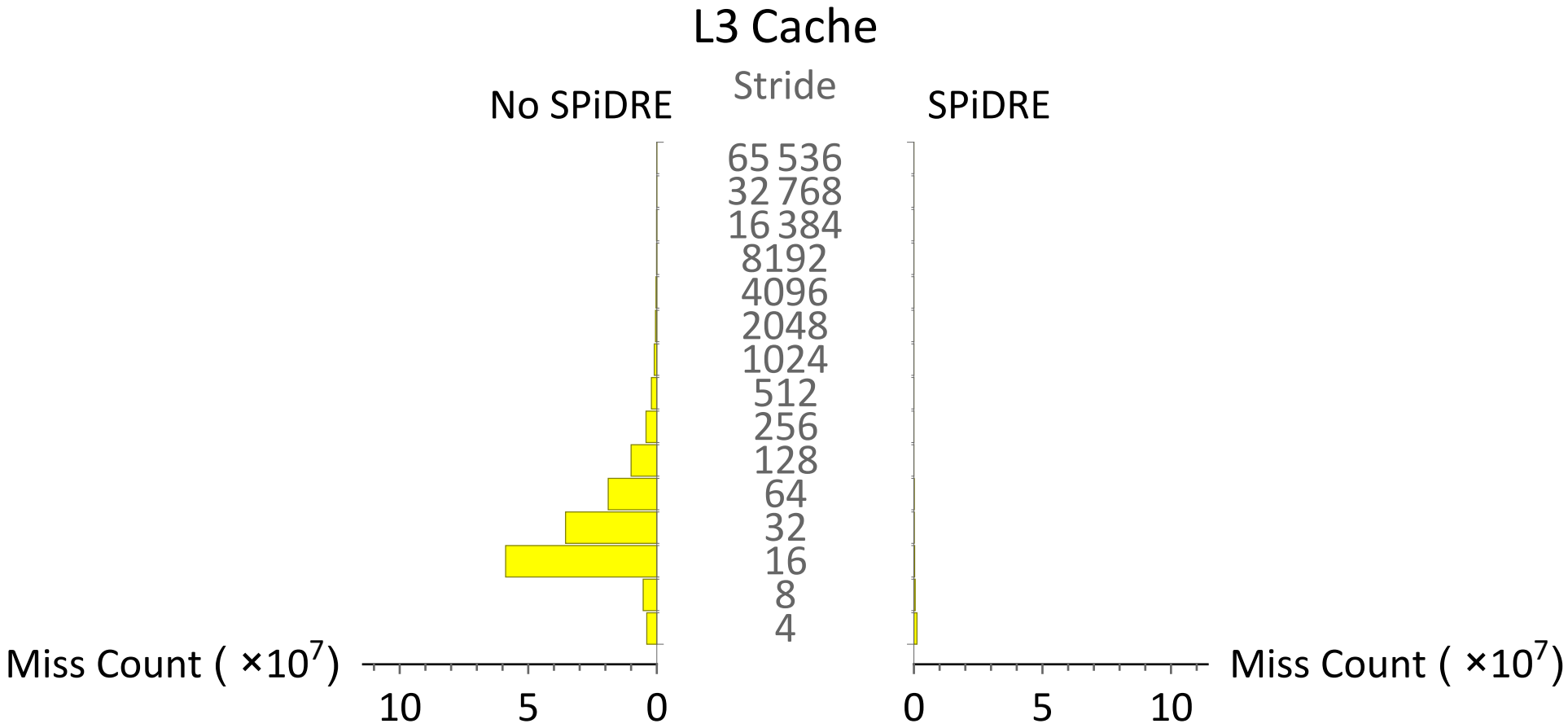
arm

Fixed Stride (1GB data set) - Gather SPiDRE



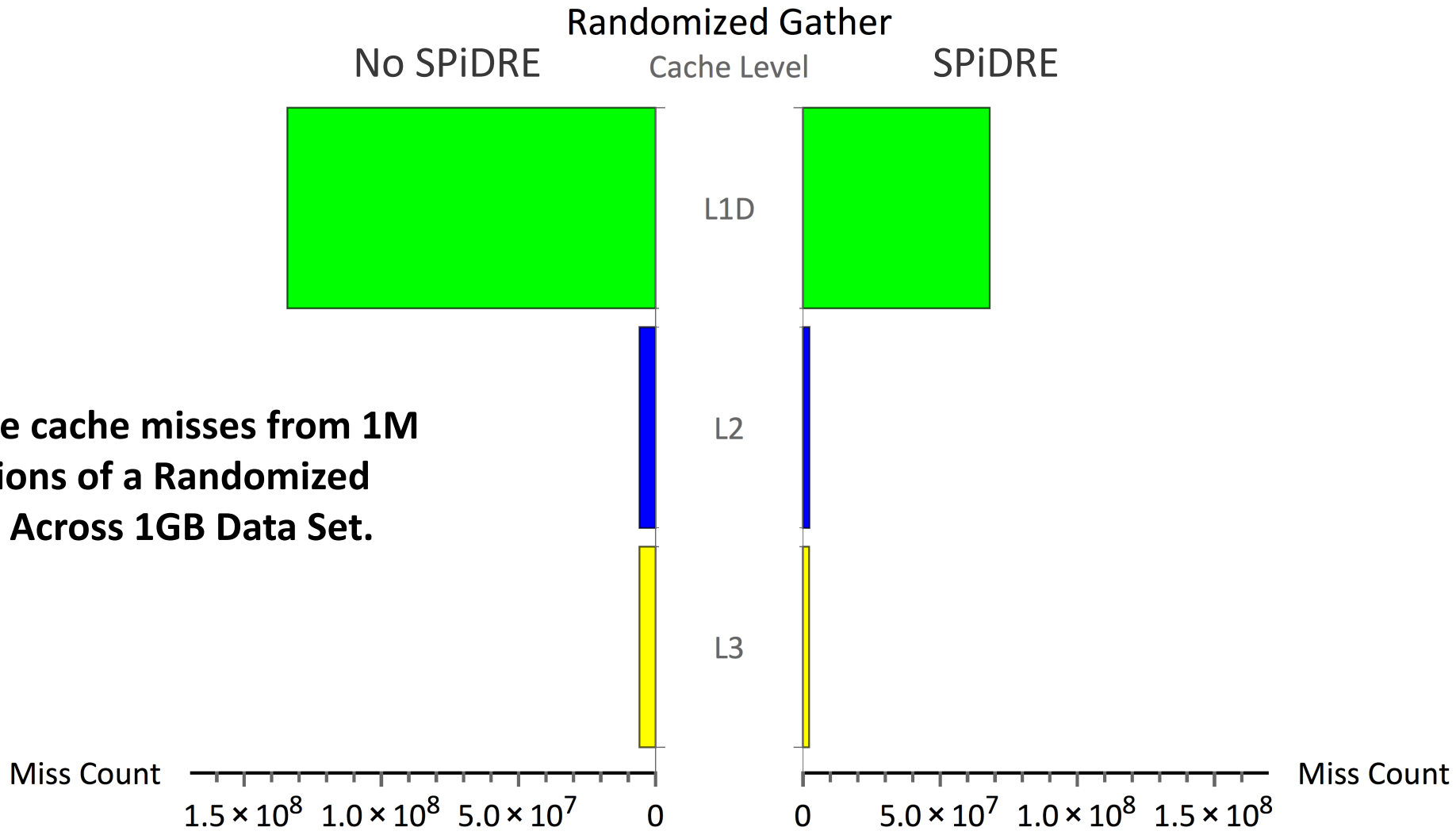
arm

Fixed Stride (1GB data set) - Gather SPiDRE



arm

Random Gather - SPiDRE



Average cache misses from 1M executions of a Randomized Gather Across 1GB Data Set.

arm

HPC Mini-apps

All single threaded executions for initial study

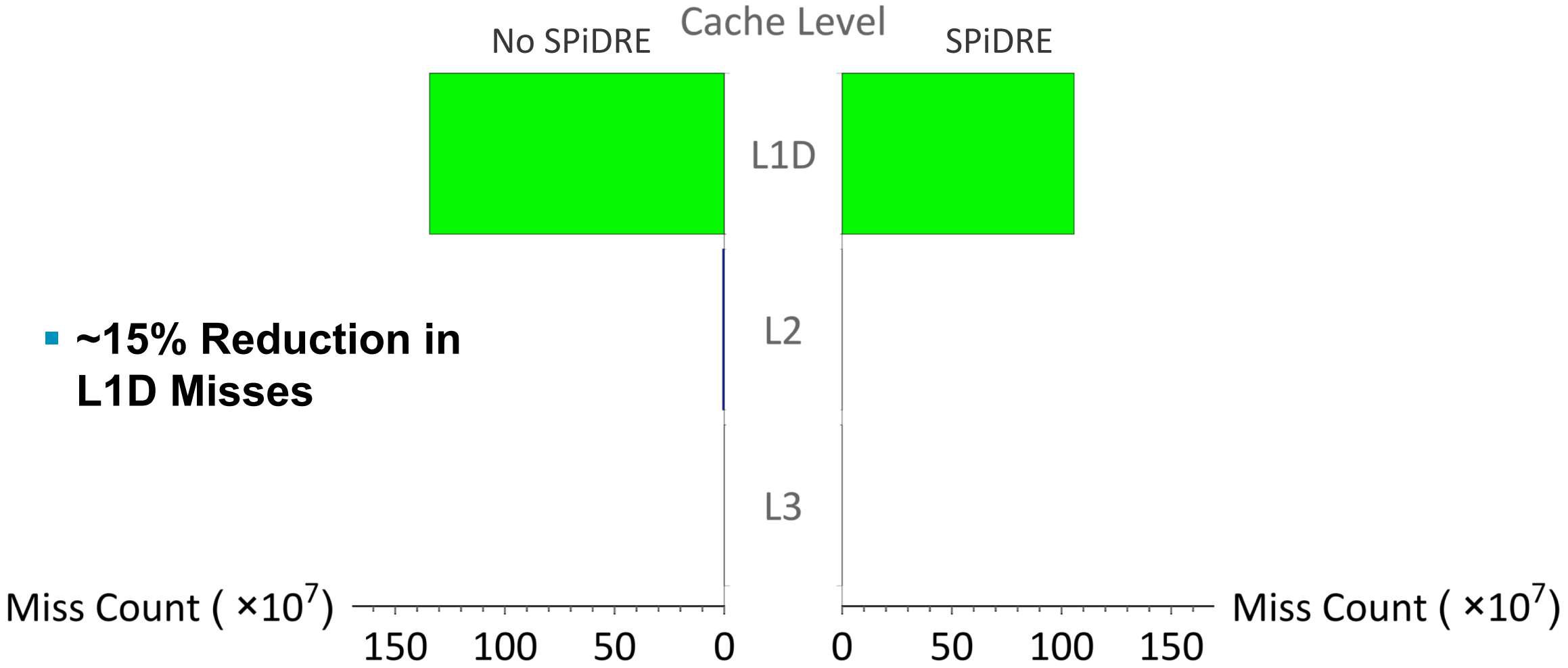
CoMD

- Used Lennard-Jones potentials
- Simple port to gather within the main loop

LULESH

- Multiple ports with varying rearrange to use distances
- Varied size and iteration count
- Demonstrated sparse data reduction combined with programmer placed pre-fetch hints

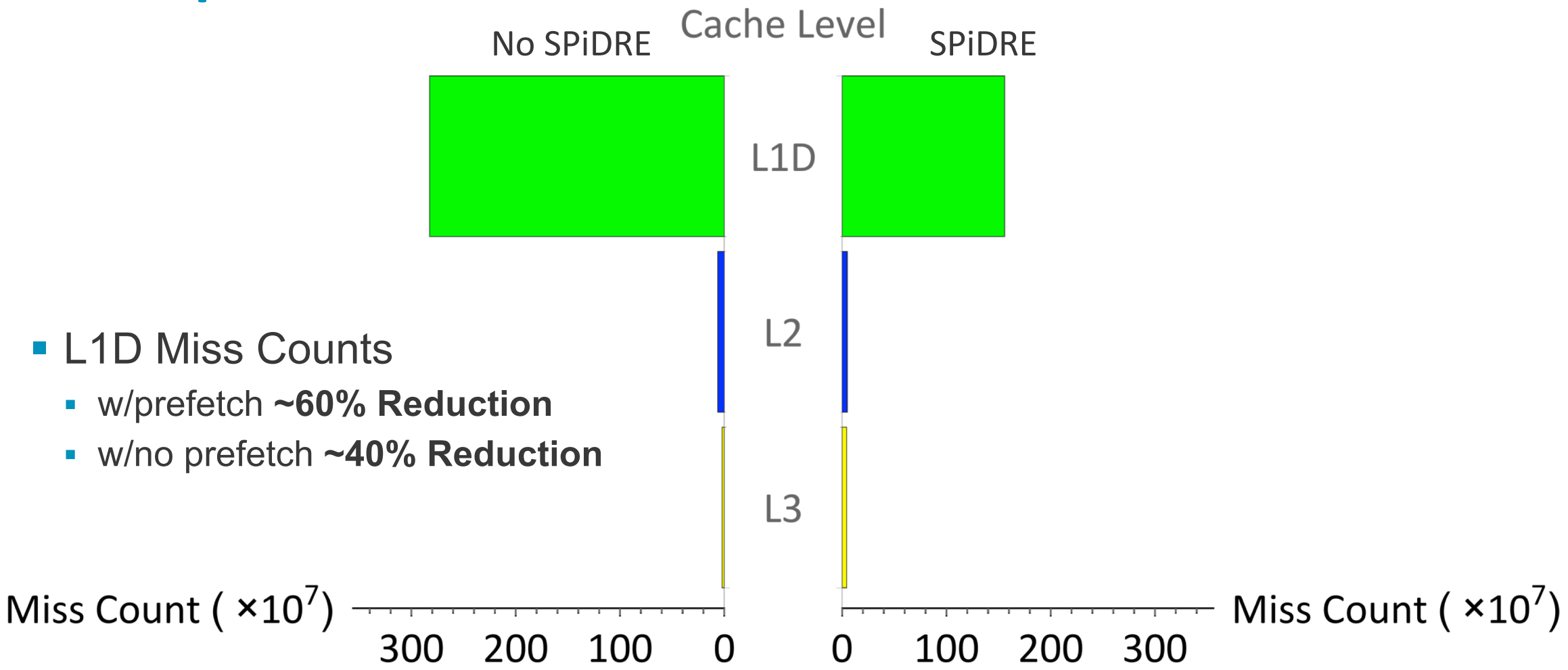
CoMD Naïve Port to SPiDRE



- **~15% Reduction in L1D Misses**

arm

LULESH ported to SPiDRE

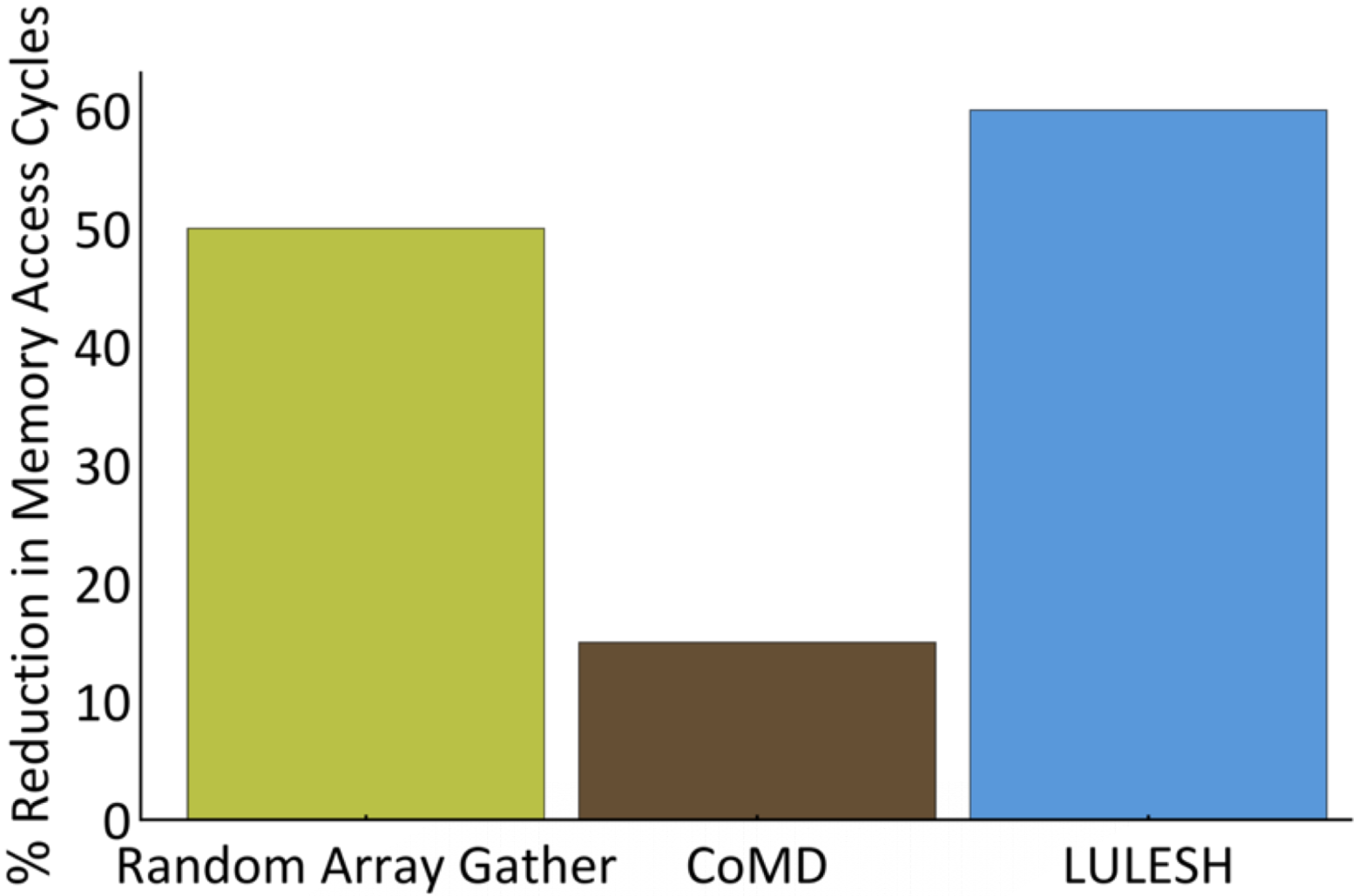


- L1D Miss Counts
  - w/prefetch **~60% Reduction**
  - w/no prefetch **~40% Reduction**

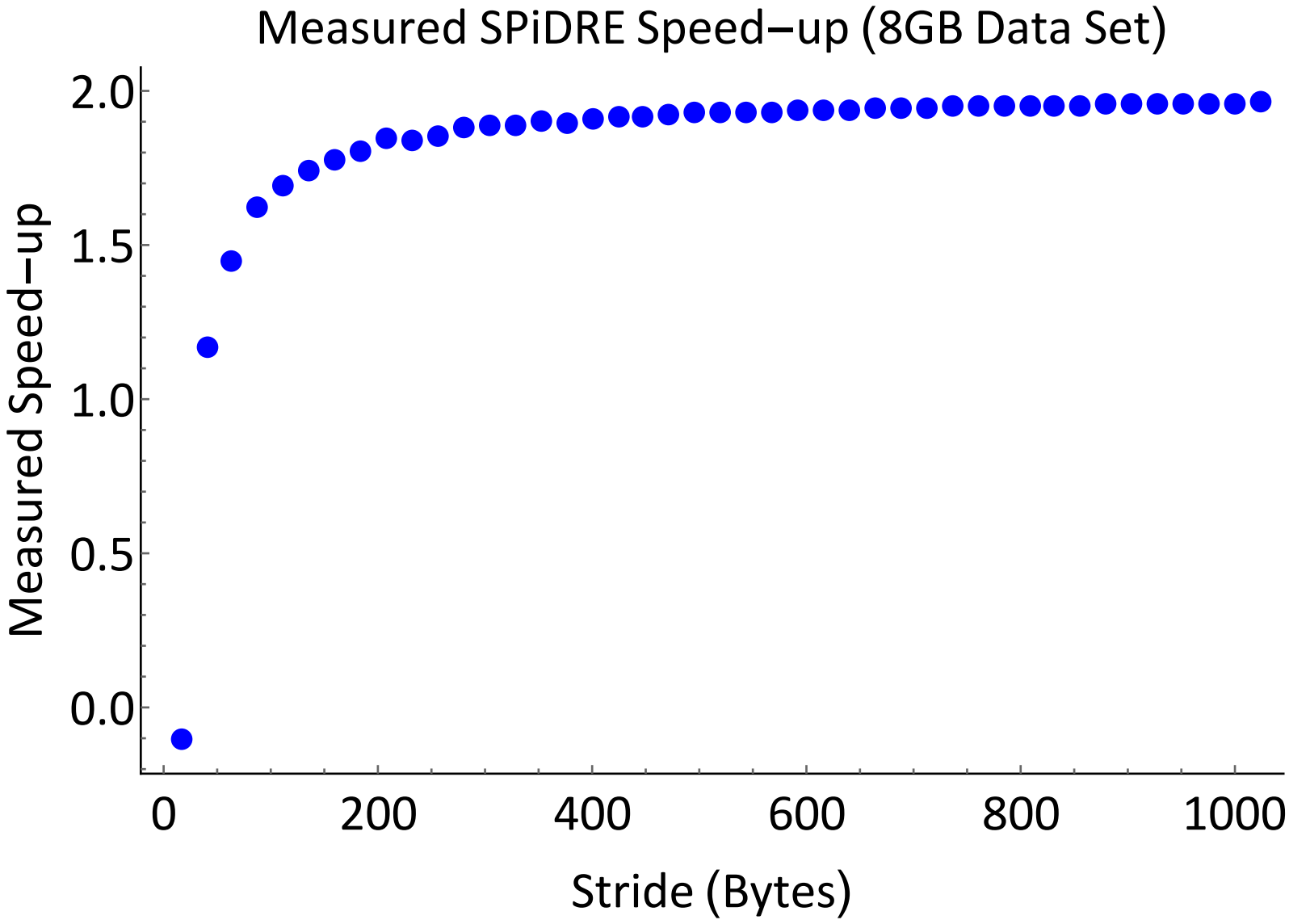
arm

Early Results

Emulator Based - Latency



Speed-up by average stride



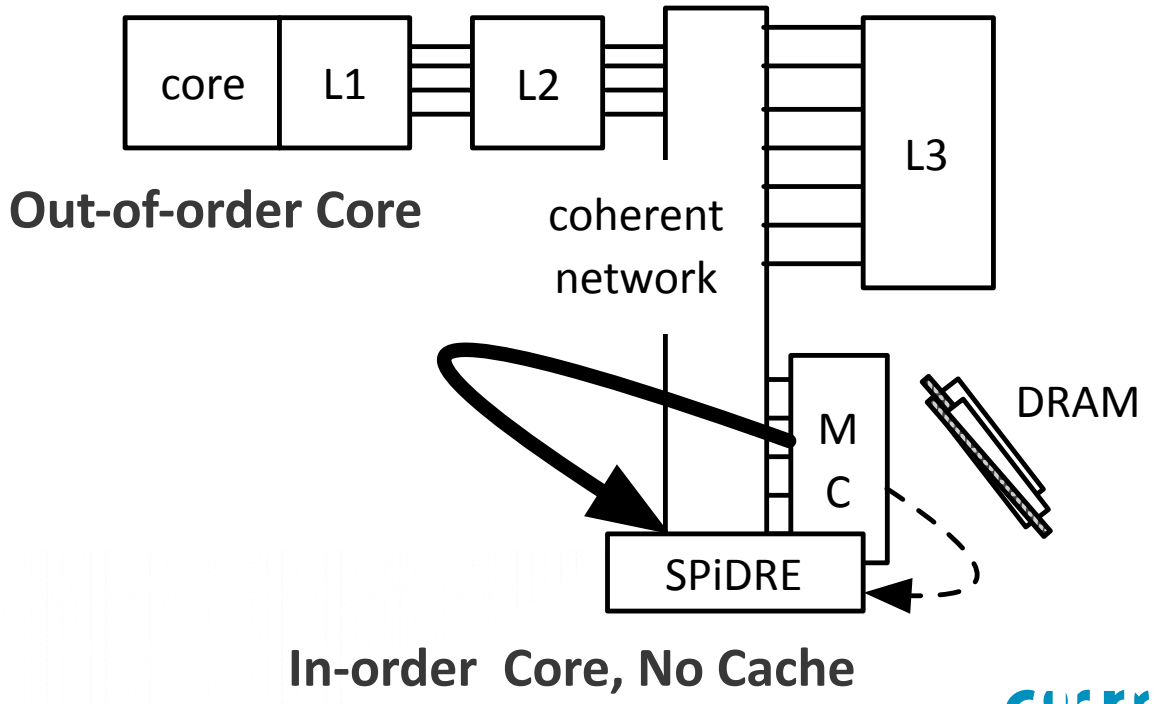
arm

Early Results

Speedup

- **Emulator** on actual hardware:
  - 2x-5x speed-up for random access
  - ~1.4 – 2.6x speed-up for Page Rank
  - ~2x speed-up for LULESH

- Full system **gem5** (early results)
  - ~10x speed-up HPCG



Conclusions

- SPiDRE is an interface and hardware acceleration infrastructure to gather data near memory/storage and make it dense (reducing bandwidth utilization, enabling more vectorization)
- We've shown a 2-10x speedup and significant data movement reduction on several applications, definitely more room (some cases greater than 2x)
- Not shown, increase in vectorization….but pretty obvious.

Future work

- Translation for near-memory compute (inc. gather/scatter)
- Programming models, compilers (incorporate cost models with llvm, etc.)
- More full system simulation

Questions…

Twitter: @jonathan_beard
Email: jonathan.beard@arm.com