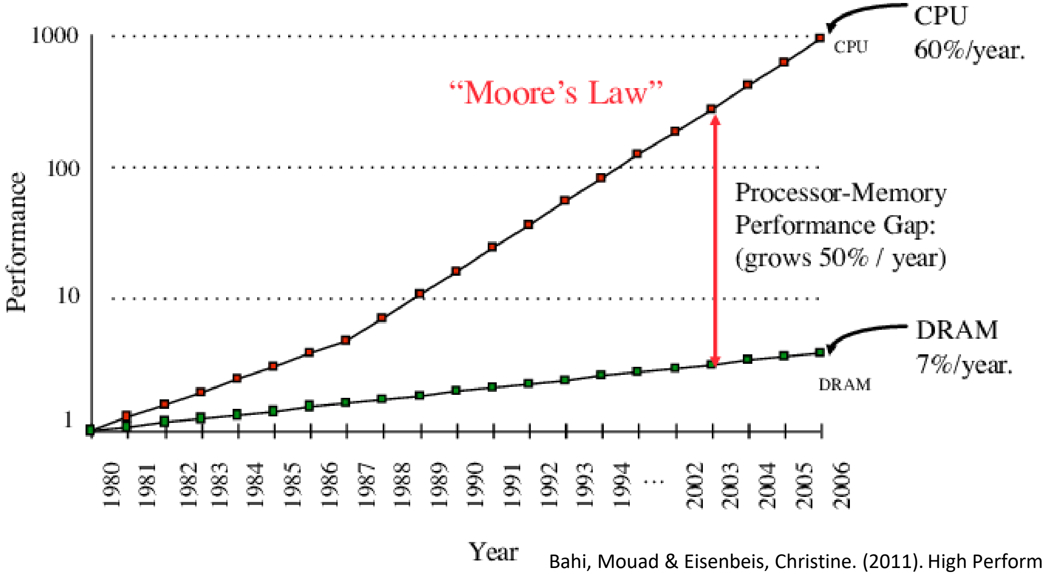
Architecting HBM as a High Bandwidth, High Capacity, Self-Managed Last-Level Cache

Tyler Stocksdale
Advisor: Frank Mueller
Mentor: Mu-Tien Chang
Manager: Hongzhong Zheng
11/13/2017

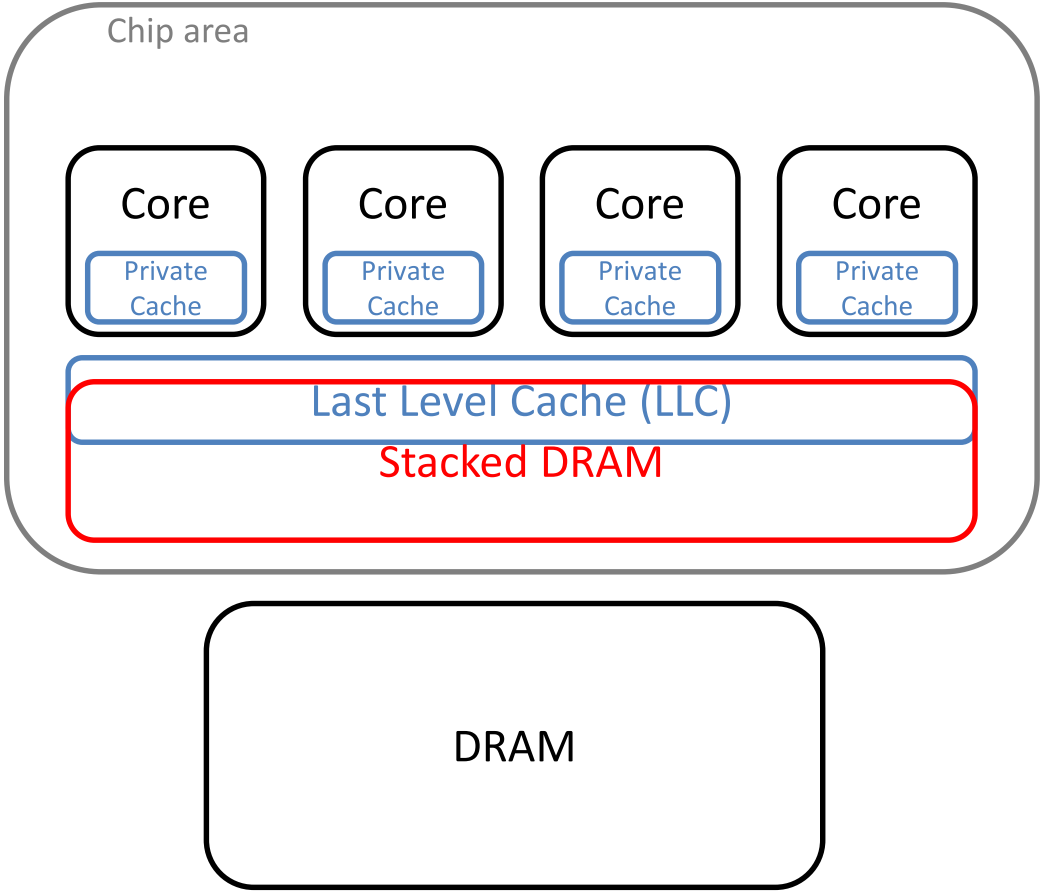
Background

- **Commodity DRAM is hitting the memory/bandwidth wall**
  – Off-chip bandwidth is not growing at the rate necessary for the recent growth in the number of cores
  – Each core has a decreasing amount of off-chip bandwidth



Bahi, Mouad & Eisenbeis, Christine. (2011). High Performance by Exploiting Information Locality through Reverse Computing. 25-32. 10.1109/SBAC-PAD.2011.10.

Motivation

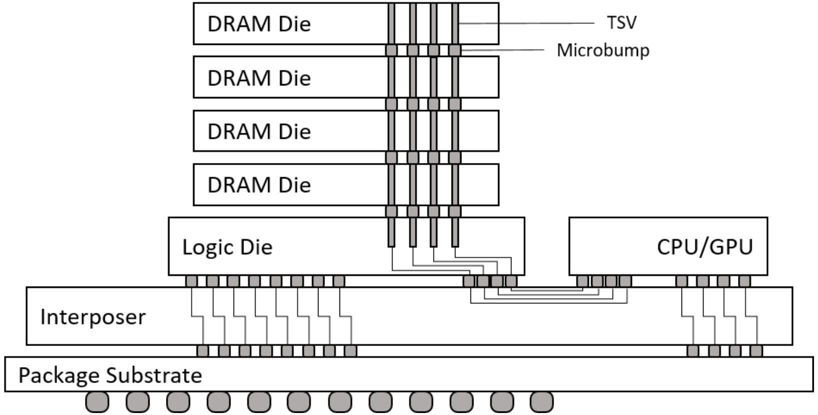


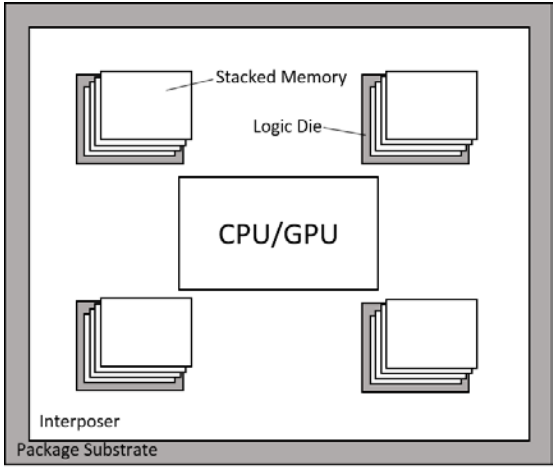
- **Caching avoids memory/bandwidth wall**
- **Large gap between existing LLC's and DRAM**
  - Capacity
  - Bandwidth
  - Latency
- **Stacked DRAM LLC's have shown 21% improvement (Alloy Cache[1])**

What is Stacked DRAM?

- **1-16GB capacity**
- **8-15x the bandwidth of off-chip DRAM [1], [2]**
- **Half or one-third the latency [3], [4], [5]**

- **Variants:**
  - High Bandwidth Memory (HBM)
  - Hybrid Memory Cube (HMC)
  - Wide I/O





Related Work

- **Many proposals for stacked DRAM LLC's [1][2][6][7][11]**

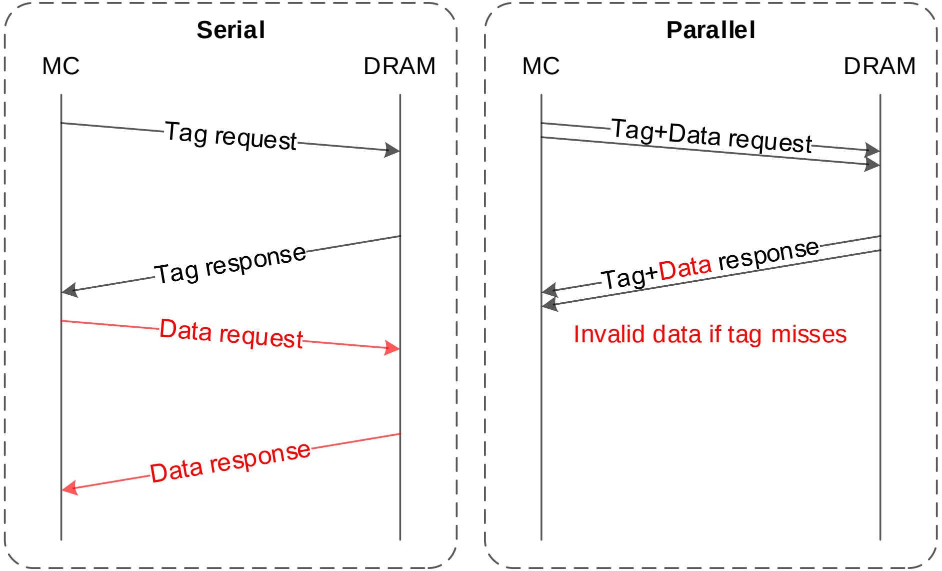
- **They are not practical**
  - Not designed for existing stacked DRAM architecture
  - Major modifications to memory controller/existing hardware

- **They don't take advantage of processing in memory (PIM)**
  - HBM's built-in logic die
  - Tag/data access could be two serial memory accesses

How are tags stored?

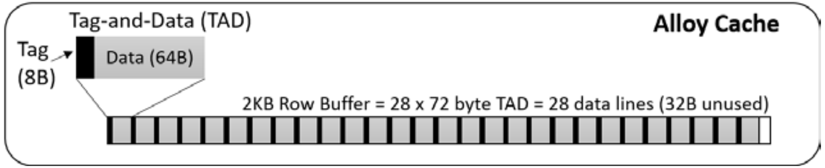
- **Cache address space smaller than memory address space**
  - “Tag” stores extra bits of address
  - Tags are compared to determine cache hit/miss

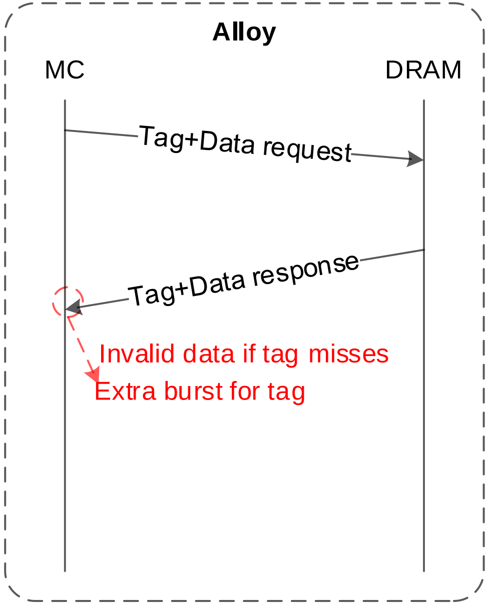
- **Solutions:**
  - Tags in stacked DRAM
  - Memory controller does tag comparisons
  - Two separate memory accesses
  - Serial vs. Parallel access
  - “Alloyed” Tag/Data structure for a single access



Alloy Cache [1]

- **Tag and data fused together as one unit (TAD)**
- **Best performing stacked DRAM cache (21% improvement)**
- **Used as comparison by many papers**
- **Limitations:**
  - Irregular burst size
  - Wastes capacity (32B per row)
  - Direct mapped only
  - Not designed for existing stacked DRAM architecture


Tag-and-Data (TAD)
Alloy Cache
Tag (8B)
Data (64B)
2KB Row Buffer = 28 x 72 byte TAD = 28 data lines (32B unused)



Our Idea

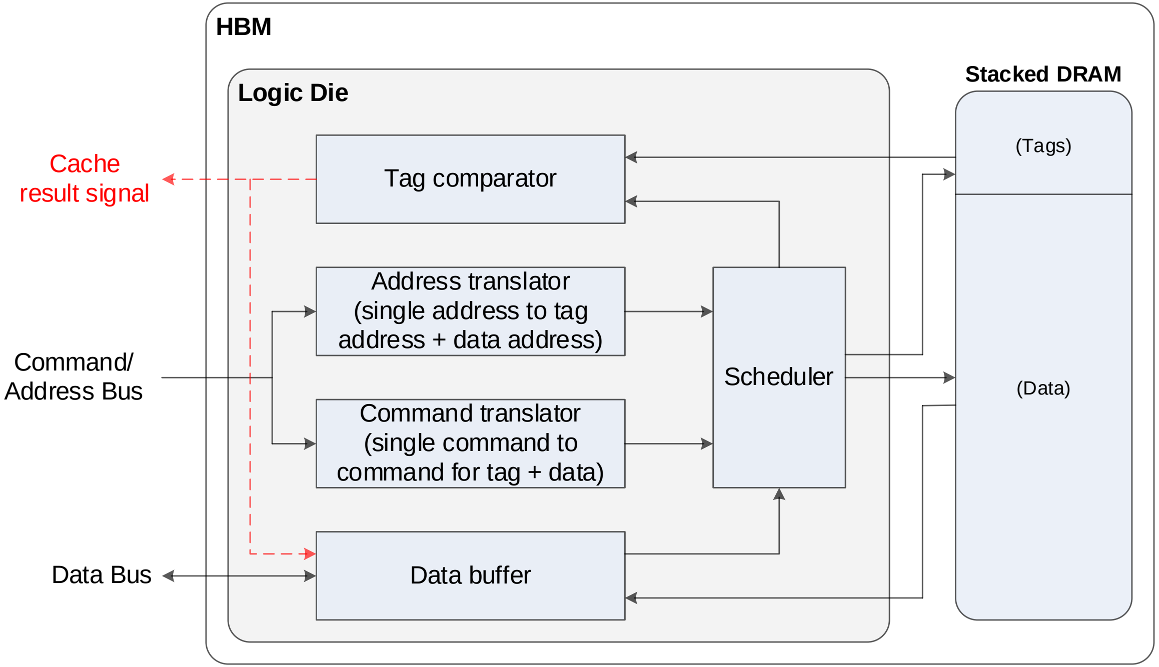
1. **Use HBM for our stacked DRAM LLC**
   - Best balance of price, power consumption, bandwidth
   - Contains logic die

2. **HBM logic die performs cache management**

3. **Store tag and data on different stacked DRAM channels**

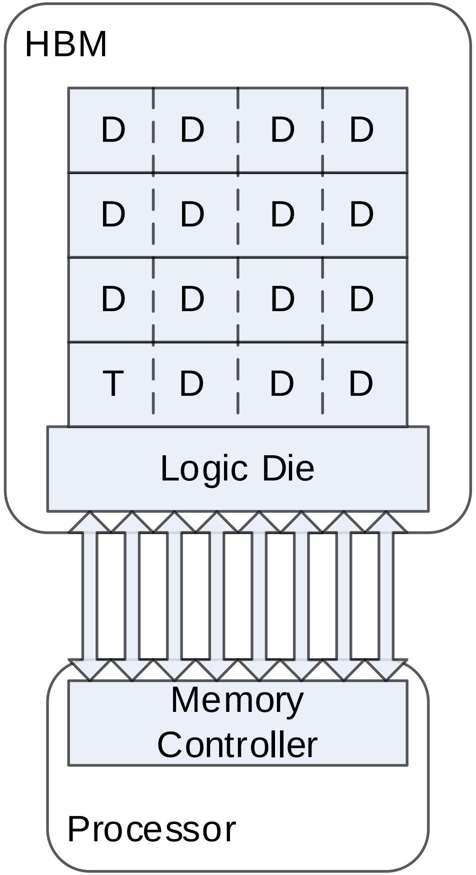
Logic Die Design

- Less bandwidth over data bus
- Memory controller is simple
  - No tag comparisons
  - Sees HBM Cache as ordinary DRAM device
  - Minor modification for Cache Result signal
- Requires new “Cache Result” signal
  - Signals hit, clean miss, dirty miss, invalid, etc.


HBM
Logic Die
Stacked DRAM
(Tags)
(Data)
Cache result signal
Tag comparator
Address translator (single address to tag address + data address)
Command/ Address Bus
Scheduler
Command translator (single command to command for tag + data)
Data Bus
Data buffer

Tag/Data on Different Channels

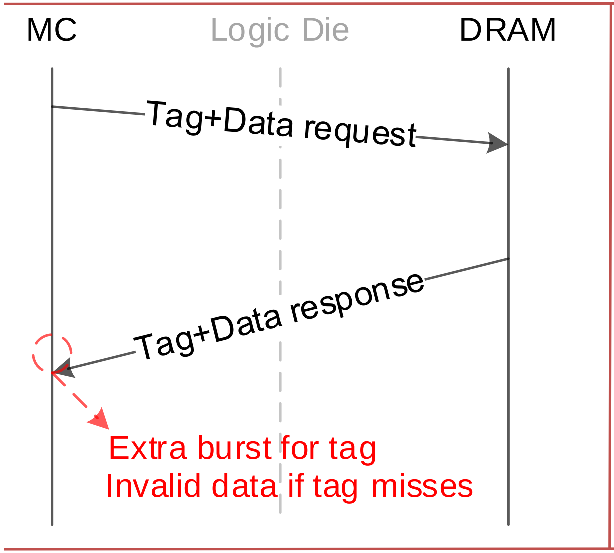
- **16 pseudo-channels**
  - Use 1 pseudo-channel for tags
  - Use 15 pseudo-channels for data
- **Benefits:**
  - Parallel tag/data access
  - Higher capacity than Alloy cache
    - Data channels have zero wasted space
    - Tag channel wastes 16MB total
    - Alloy cache wastes 64MB total



Test Configurations

"Alloy"

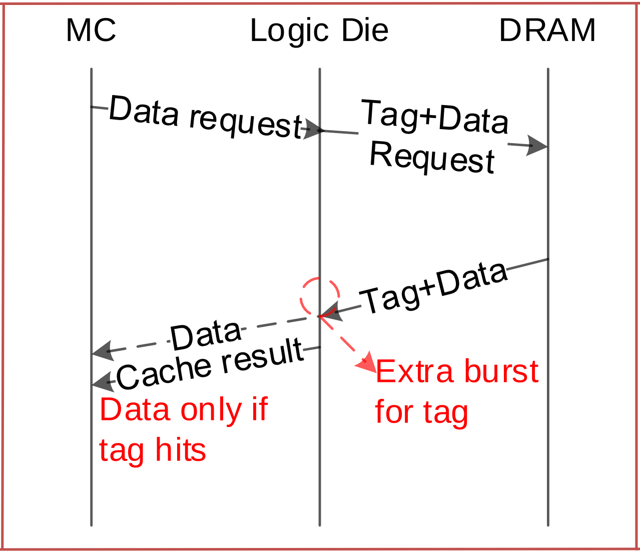
1. Alloy Cache (baseline)



- Implemented on HBM
- Logic die unused

"Alloy-like"

2. Logic Die Cache Management

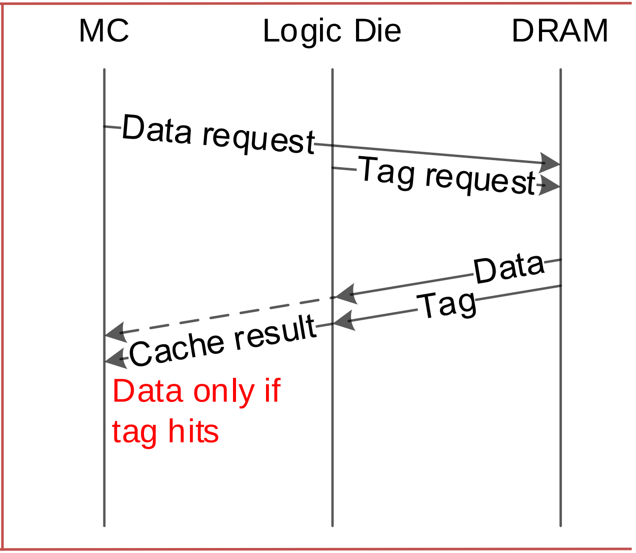


- Cache management moved to logic die
- Still using Alloy TAD's

"SALP"

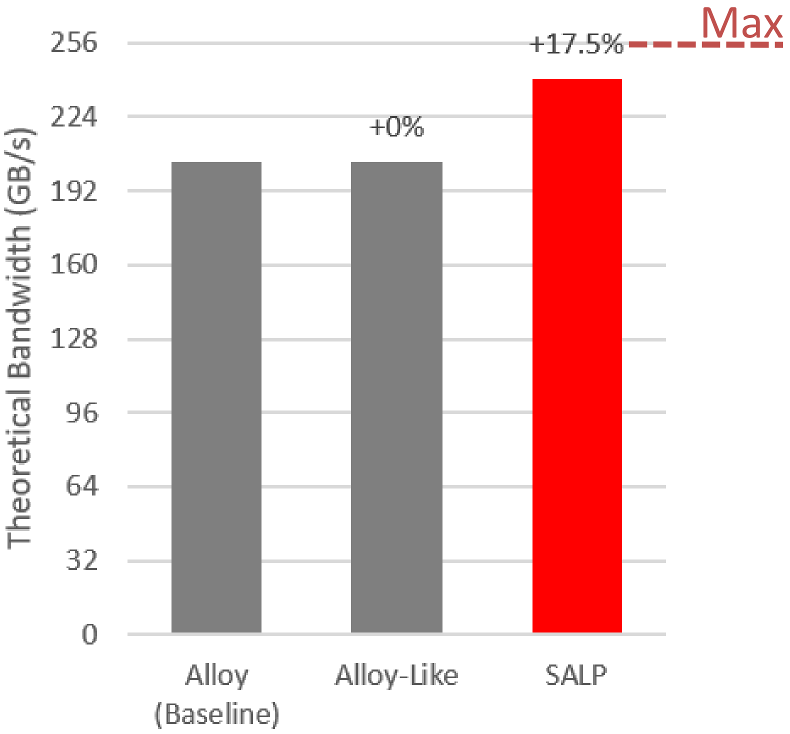
(sub-array level parallelism)

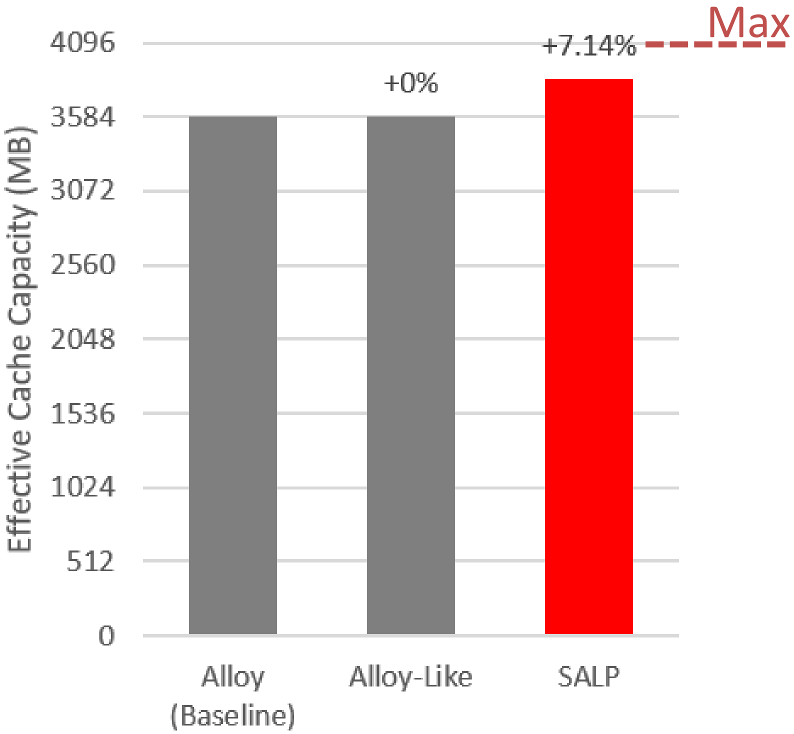
3. Separate Tag/Data Channels



- Cache management still on logic die
- Tag/Data separated

Improved Theoretical Bandwidth and Capacity





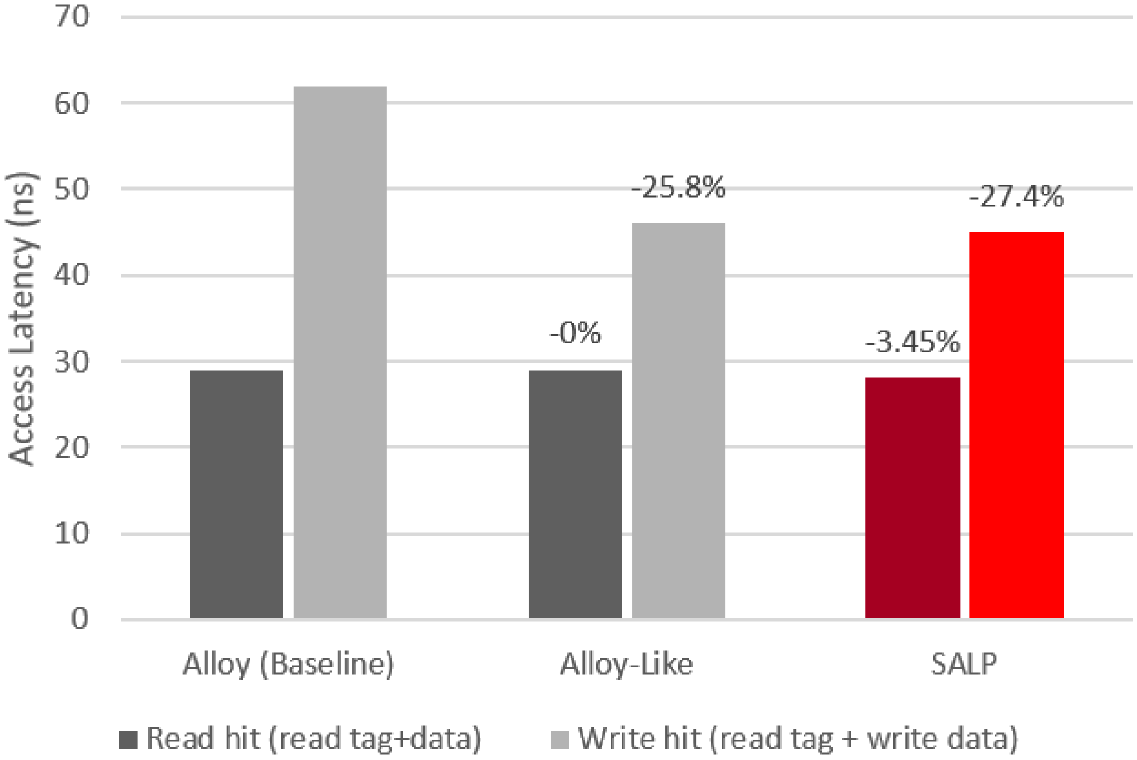
Separate channels for Tag and Data (SALP) result in significant bandwidth and capacity improvements

Improved Theoretical Hit Latency

- **Timing parameters based on Samsung DDR4 8GB spec.**

- **Write buffering on logic die**

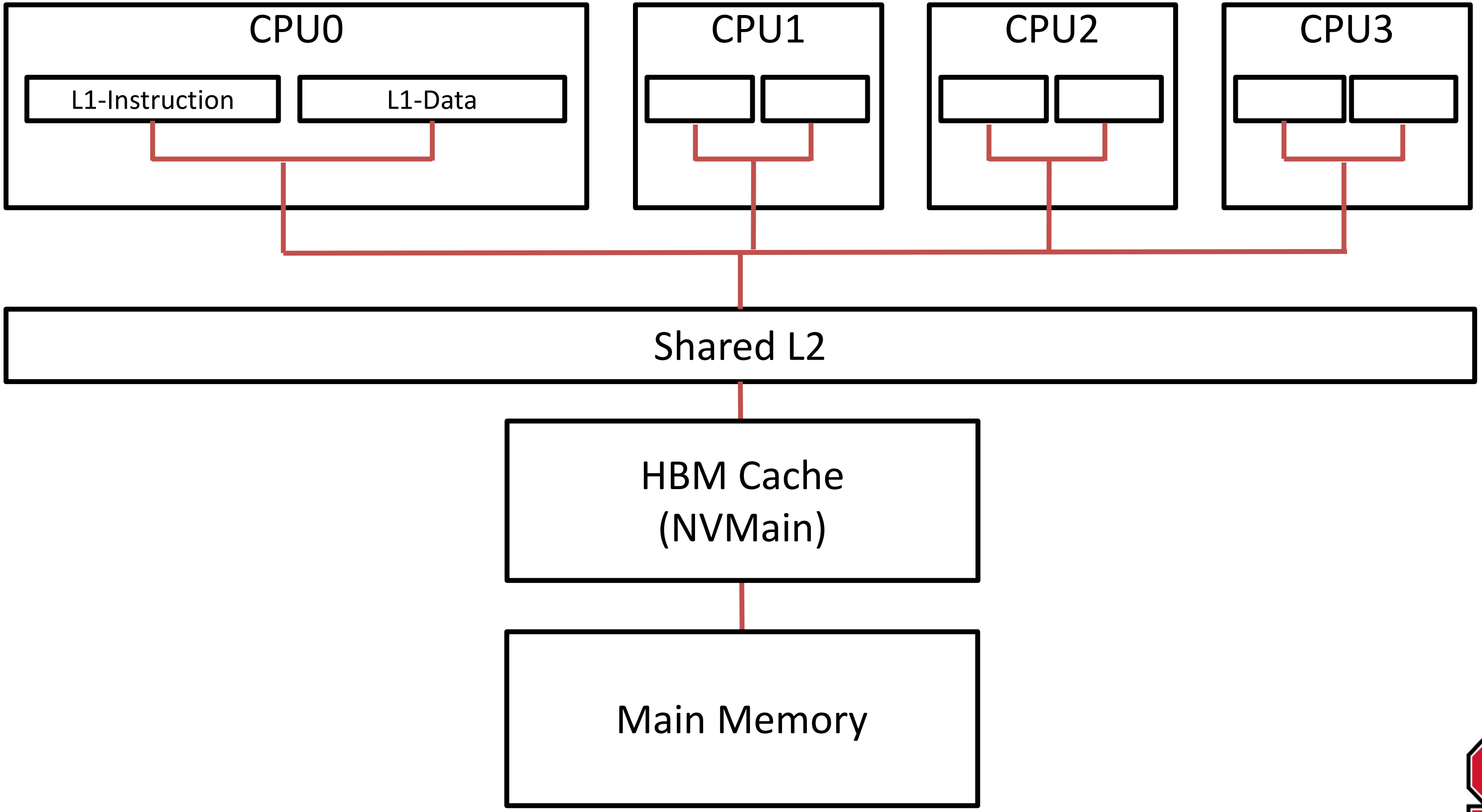
- **SALP adds additional parallelism**



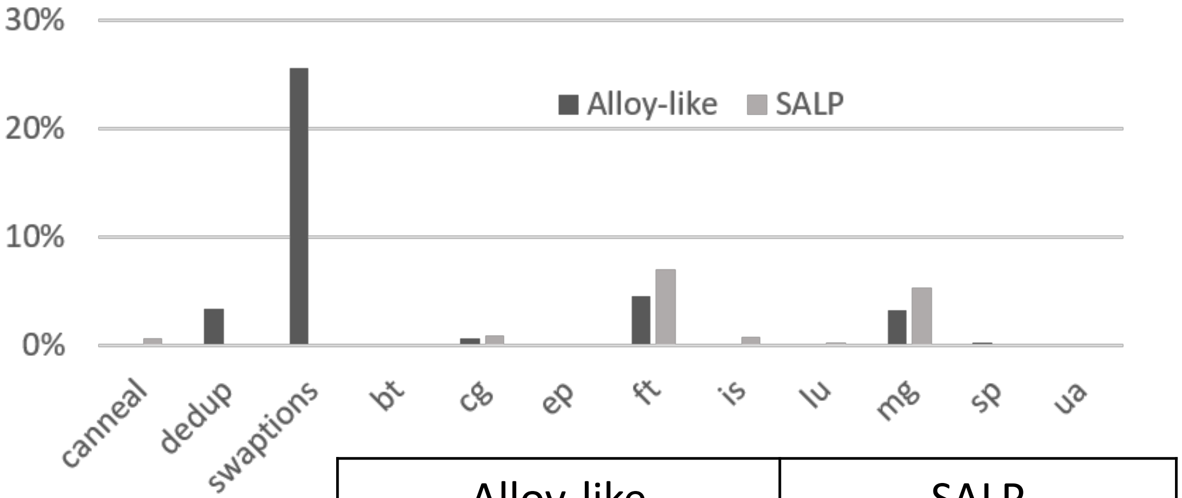
Simulators

- **GEM5** [8]
  - Custom configuration for a multi-core architecture with HBM last-level cache
  - Full system simulation: boots linux kernel and loads a custom disk image
- **NVMain** [9]
  - Contains a model for Alloy Cache
  - Created two additional models for Alloy-like and SALP
- **Configurable parameters:**
  - Number of CPU’s, frequency, bus widths, bus frequencies
  - Cache size, associativity, hit latency, frequency
  - DRAM timing parameters, architecture, energy/power parameters

Simulated System Architecture



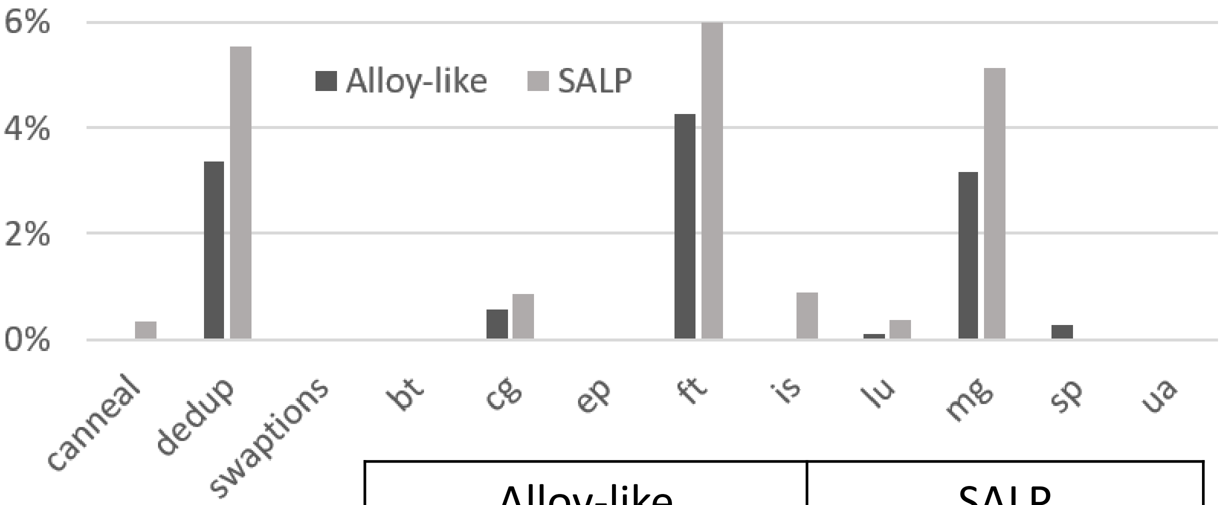
Performance Benefit - Bandwidth



	Alloy-like	SALP
Minimum	-0.30% (UA)	-0.72% (Dedup)
Maximum	25.53% (Swaptions)	7.07% (FT)
Arithmetic Mean	3.10%	1.22%
Geometric Mean	2.89%	1.19%

Alloy-like configuration has higher average bandwidth

Performance Benefit – Execution Time



	Alloy-like	SALP
Minimum	-0.20% (IS)	-0.42% (UA)
Maximum	4.26% (FT)	6.59% (FT)
Arithmetic Mean	0.92%	1.73%
Geometric Mean	0.93%	1.76%

SALP configuration has lower average execution time

Conclusions

- **Beneficial in certain cases**
  - Theoretical results indicate noticeable performance benefit
  - Categorize benchmarks that perform well with HBM cache
  - Benchmark analysis to decide cache configuration
    - Already in progress for Intel Knights Landing

- **Much simpler memory controller**
  - Equal or better performance

References

[1] M. K. Qureshi and G. H. Loh, “Fundamental latency tradeoff in architecting DRAM caches: Outperforming impractical SRAM-tags with a simple and practical design,” in *International Symposium on Microarchitecture*, 2012, pp. 235–246.

[2] “Intel Xeon Phi Knights Landing Processors to Feature Onboard Stacked DRAM Supercharged Hybrid Memory Cube (HMC) upto 16GB,” http://wccftech.com/intel-xeon-phiknights-landing-processors-stacked-dram-hmc-16gb/, 2014.

[3] C. C. Chou, A. Jaleel, and M. K. Qureshi, “CAMEO: A TwoLevel Memory Organization with Capacity of Main Memory and Flexibility of Hardware-Managed Cache,” in *International Symposium on Microarchitecture (MICRO)*, 2014, pp. 1–12.

[4] S. Yin, J. Li, L. Liu, S. Wei, and Y. Guo, “Cooperatively managing dynamic writeback and insertion policies in a lastlevel DRAM cache,” in *Design, Automation & Test in Europe (DATE)*, 2015, pp. 187–192.

[5] X. Jiang, N. Madan, L. Zhao, M. Upton, R. Iyer, S. Makineni, D. Newell, D. Solihin, and R. Balasubramonian, “CHOP: Adaptive filter-based DRAM caching for CMP server platforms,” in *International Symposium on High Performance Computer Architecture (HPCA)*, 2010, pp. 1–12.

[6] B. Pourshirazi and Z. Zhu, "Refree: A Refresh-Free Hybrid DRAM/PCM Main Memory System", International Parallel and Distributed Processing Symposium (IPDPS), 2016, pp. 566-575.

[7] N. Gulur, M. Mehendale, R. Manikantan, and R. Govindarajan, "Bi-Modal DRAM Cache: Improving Hit Rate, Hit Latency and Bandwidth", International Symposium on Microarchitecture (MICRO), 2014, pp. 38-50.

[8] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, “The gem5 simulator,” *SIGARCH Comput. Archit. News*, vol. 39, no. 2, pp. 1–7, 2011.

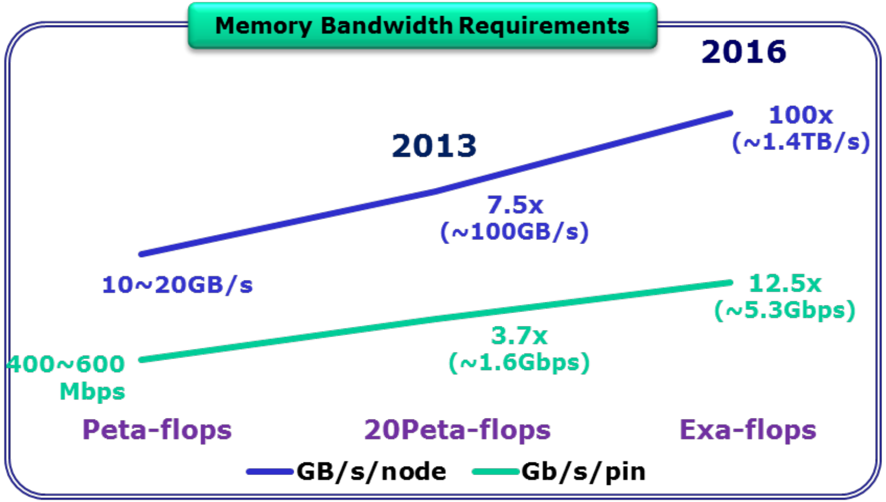
[9] M. Poremba, T. Zhang, and Y. Xie, “NVMain 2.0: Architectural Simulator to Model (Non-)Volatile Memory Systems,” *Computer Architecture Letters (CAL)*, 2015.

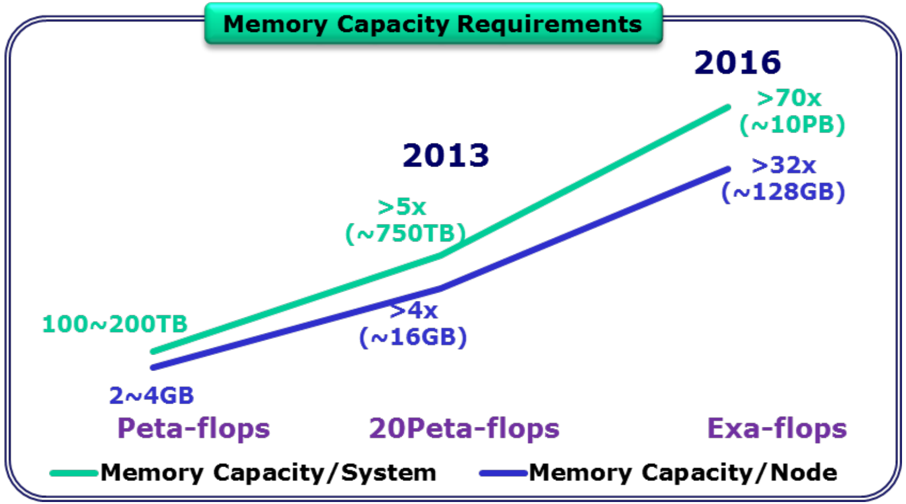
[10]S. Mittal, J.S. Vetter, “A Survey Of Techniques for Architecting DRAM Caches,” *IEEE Transactions on Parallel and Distributed Systems*, 2015.

Outline

- **Background**
- Contribution 1: full-system simulation infrastructure
- Contribution 2: self-managed HBM cache
- Appendix

Background





[Source: “Memory systems for PetaFlop to ExaFlop class machines” by IBM, 2007 & 2010]

Linear to Exponential demand for Memory Bandwidth and Capacity

Overview

- **Background**
  - Stacked DRAM cache as a high bandwidth, high capacity last-level cache potentially improves system performance
  - Prior results [1]: 21% performance improvement
- **Challenges**
  - [Challenge 1] Unclear about the benefit of HBM cache
    - We need a way to study the HBM cache and understand its benefits
  - [Challenge 2] With minimal changes to the current HBM2 spec, how to best architect HBM caches

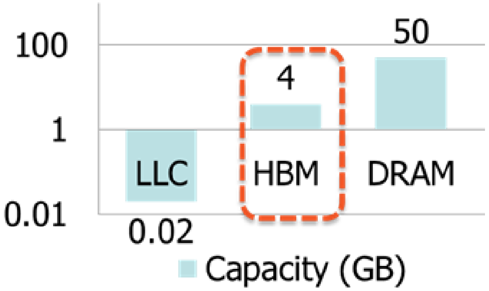
Contributions

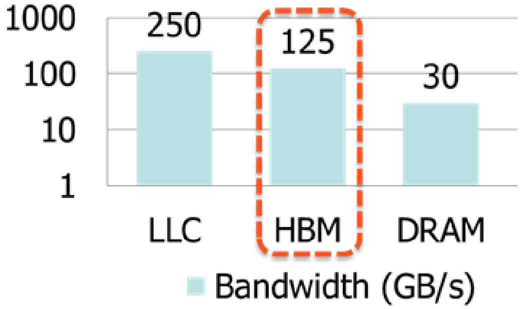
- **Solution to [Challenge 1]: Brought up and augmented the Gem5 and NVMain simulators to study HBM cache in a full-system environment**
  - Simulates a fully bootable linux kernel on top of custom HBM LLC architecture
  - Simulator can be easily modified for system changes
  - Created 3 different cache configurations to test
  - Integrated PARSEC/NAS benchmarks using cross-compiler

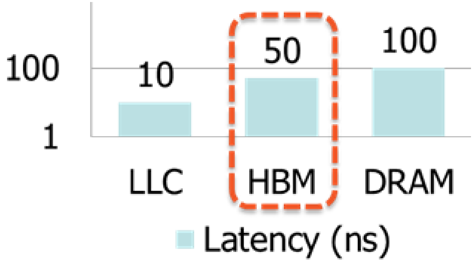
- **Solution to [Challenge 2]: Proposed two HBM cache with in-HBM (logic die) cache manager**
  - Type 1: Alloy-like. Data and tag in the same row. Uses pseudo channel and in-HBM cache manager to reduce tag/data transfers between the host and the HBM.
  - Type 2: SALP. Data and tag on different pseudo channels. We use subarray level parallelism to further improve performance.

Motivation

- **Caching avoids the memory/bandwidth wall**
- **Large gap between existing last-level caches (LLC's) and DRAM**
  - Modern workloads demand hundreds of MB's of LLC [2], [3]
  - Existing stacked DRAM LLC's have shown up to 21% system npr



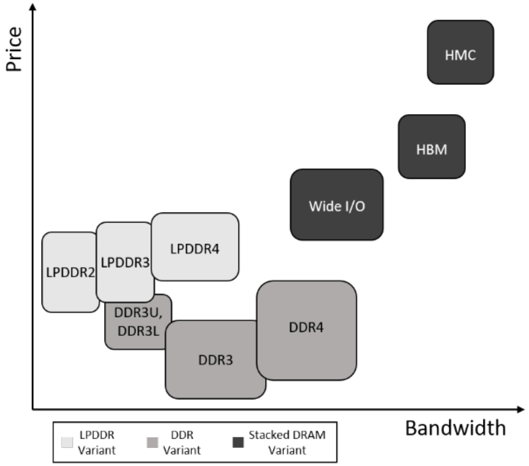


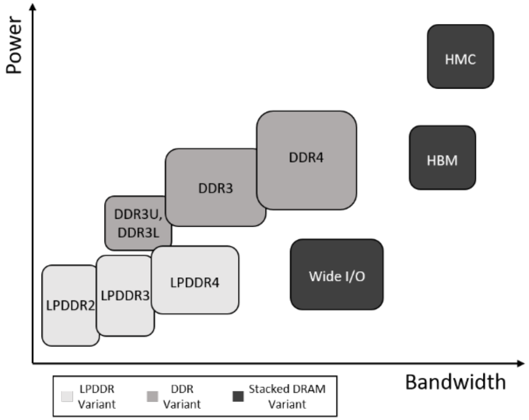


Stacked DRAM Variants

- **Hybrid Memory Cube (HMC)**
  - High end servers/enterprise
  - Highest bandwidth, cost, power
  - Used in Knights Landing Processor
  - Backed by Intel (proprietary)
  - PCB connectivity
- **HBM**
  - Graphics, HPC, networking
  - Slightly less bandwidth, cost, power than HMC
  - Used in Nvidia GPU's
  - JEDEC standard, created by Micron/AMD
  - Logic die
- **Wide I/O**
  - Smartphones, mobile
  - Lowest bandwidth, cost, power
  - JEDEC standard
  - Lots of thermal issues, sits directly on top of processor

Best Choice





Outline

- Background
- **Contribution 1: full-system simulation infrastructure**
- Contribution 2: self-managed HBM cache
- Appendix

Benchmarks

- **PARSEC**
  - Pre-compiled and ready to run
  - Some benchmarks aren’t very stressful for the memory system

- **NAS**
  - Expected to stress the memory system
  - Used cross-compiler and scripts to compile and integrate with GEM5

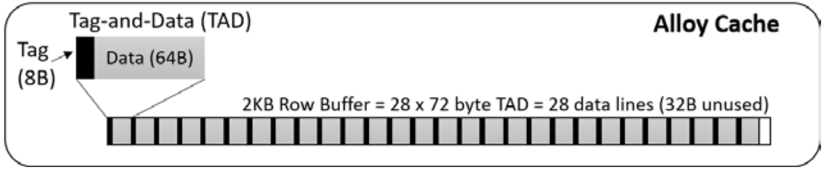
Outline

- Background
- Contribution 1: full-system simulation infrastructure
- **Contribution 2: self-managed HBM cache**
- Appendix

Techniques for self-managed HBM cache

- **Pseudo channel**
  - Benefit: reduce wasted bandwidth to transfer tag
- **Logic die with in-HBM cache manager**
  - Benefit: reduce unnecessary tag/data burst from HBM to Host
- **SALP**
  - Benefit: enable tag/data parallel access

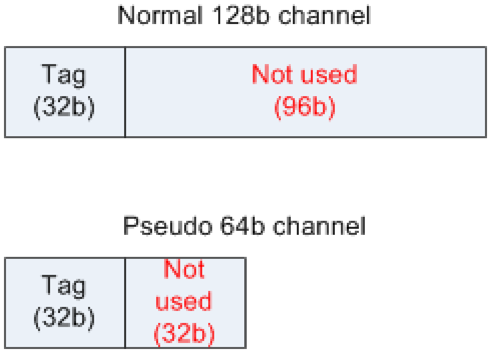
Tag and data organizations


Tag-and-Data (TAD)
Alloy Cache
Tag
(8B)
Data (64B)
2KB Row Buffer = 28 x 72 byte TAD = 28 data lines (32B unused)

- **Host-managed Alloy cache (baseline)**
  - 32B unused per row (wastes 64MB total)
  - 4.2 million less cache lines than our proposal
- **Self-managed Alloy-like HBM Cache**
  - Tag and data arranged exactly like Alloy cache
  - Longer burst length internally, but not externally
- **Self-managed SALP HBM Cache**
  - Reserve 1 pseudo-channel (256MB) for tags and the other 15 for data
  - 60M cache lines require 60M tags
  - 60M, 4B tags requires 240MB of space (wastes 16MB total)
  - 60M, 64B cache lines require 15 tag bits, 2 valid/dirty bits (17 bits total)
  - 4B tags have 15 bits leftover for miscellaneous flags, coherency bits, etc.

Pseudo channel

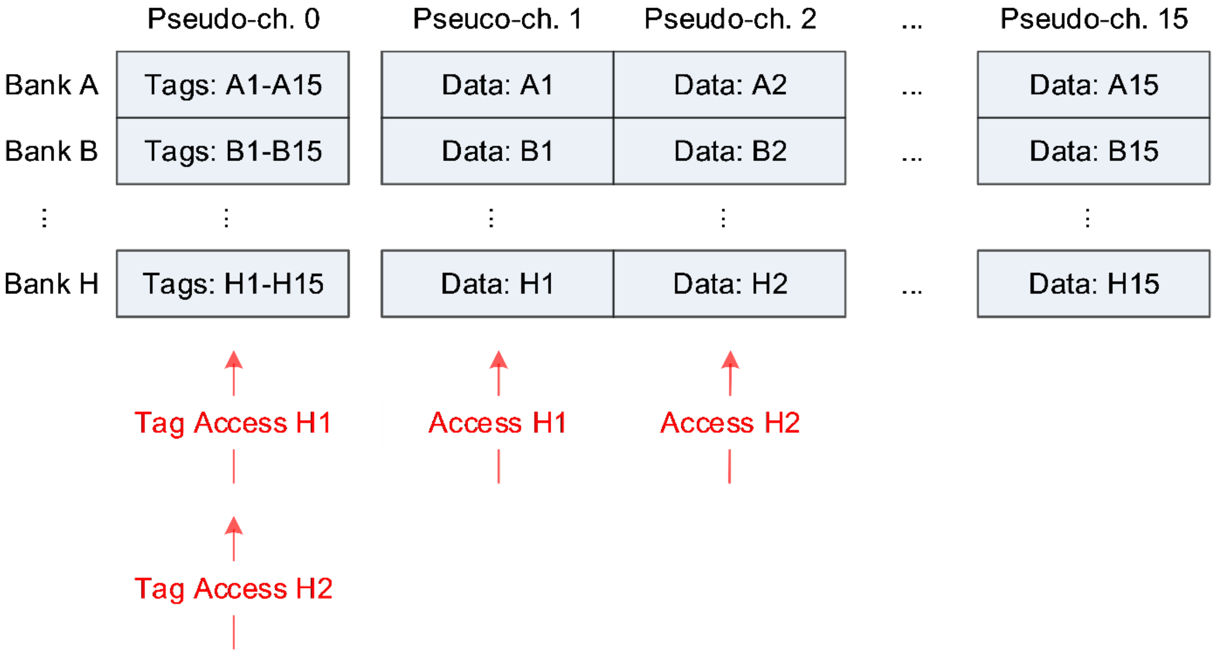
- **HBM2 spec:**
  - Default: 8 channels, 128b-wide
  - Configurable: 16 pseudo channels, 64b-wide
- **Why use pseudo channel?**
  - Normal channel
    - 1 access = 128b
    - But tag is only 4B (32b)
    - Wasting 96b (75%) of channel
  - Pseudo channel
    - 1 access = 64b
    - Wasting 32b (50%) of channel
  - **Pseudo channel organization saves 25% internal data IO bandwidth**



SALP (subarray level parallelism)

Problem:

- Data can be accessed in parallel, but tag accesses may experience a bank conflict

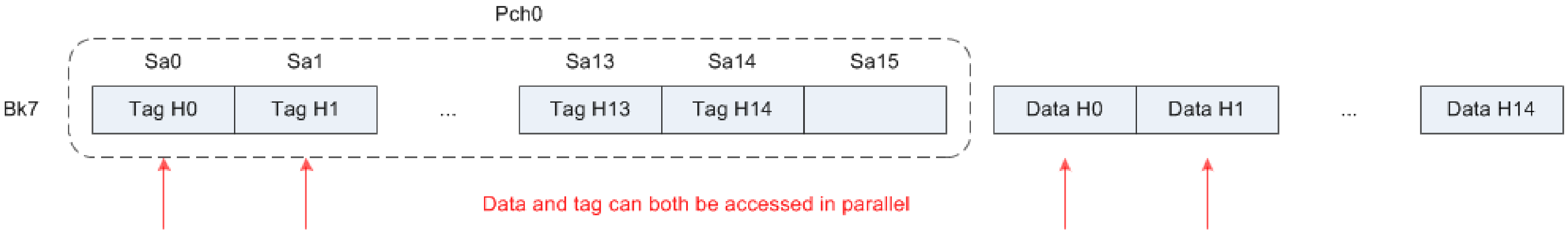


	Pseudo-ch. 0	Pseuco-ch. 1	Pseudo-ch. 2	...	Pseudo-ch. 15
Bank A	Tags: A1-A15	Data: A1	Data: A2	...	Data: A15
Bank B	Tags: B1-B15	Data: B1	Data: B2	...	Data: B15
⋮	⋮	⋮	⋮		⋮
Bank H	Tags: H1-H15	Data: H1	Data: H2	...	Data: H15

SALP (subarray level parallelism)

Solution:

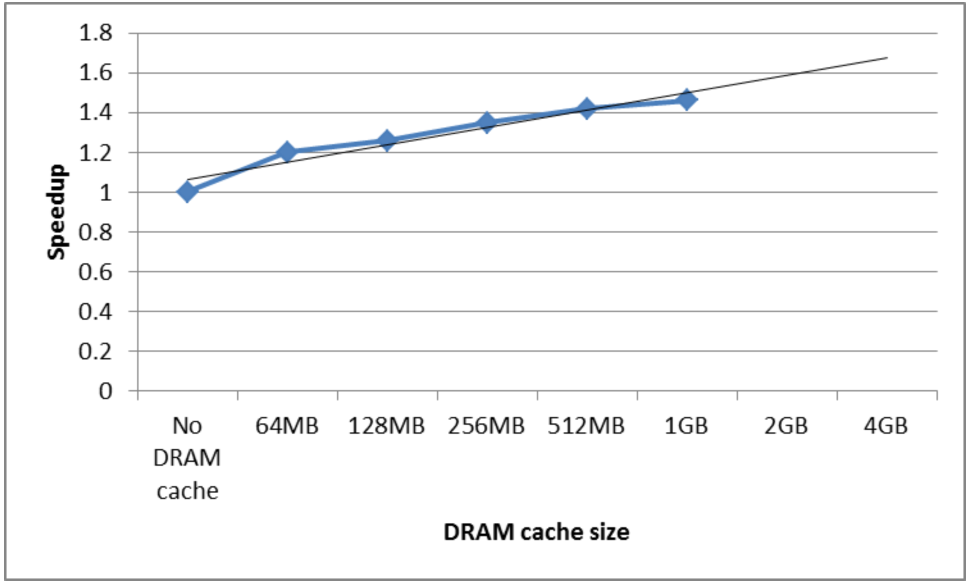
- SALP: Each bank has 16 subarrays, which can be accessed in parallel
- Each subarray stores a different tag
- Accesses can still be processed concurrently even though they are in the same bank



Data and tag can both be accessed in parallel

Future Work

- Study types of applications with workloads that would benefit from HBM
- Study the effect of HBM cache on fused-architecture processors
  - GPU simulation
  - Shared LLC and main memory
  - Private lower level caches
- Add complexity to the logic die to enable cache associativity (replacement policies)
- Add complexity to logic die to support coherency across multiple nodes
- Investigate fault tolerance



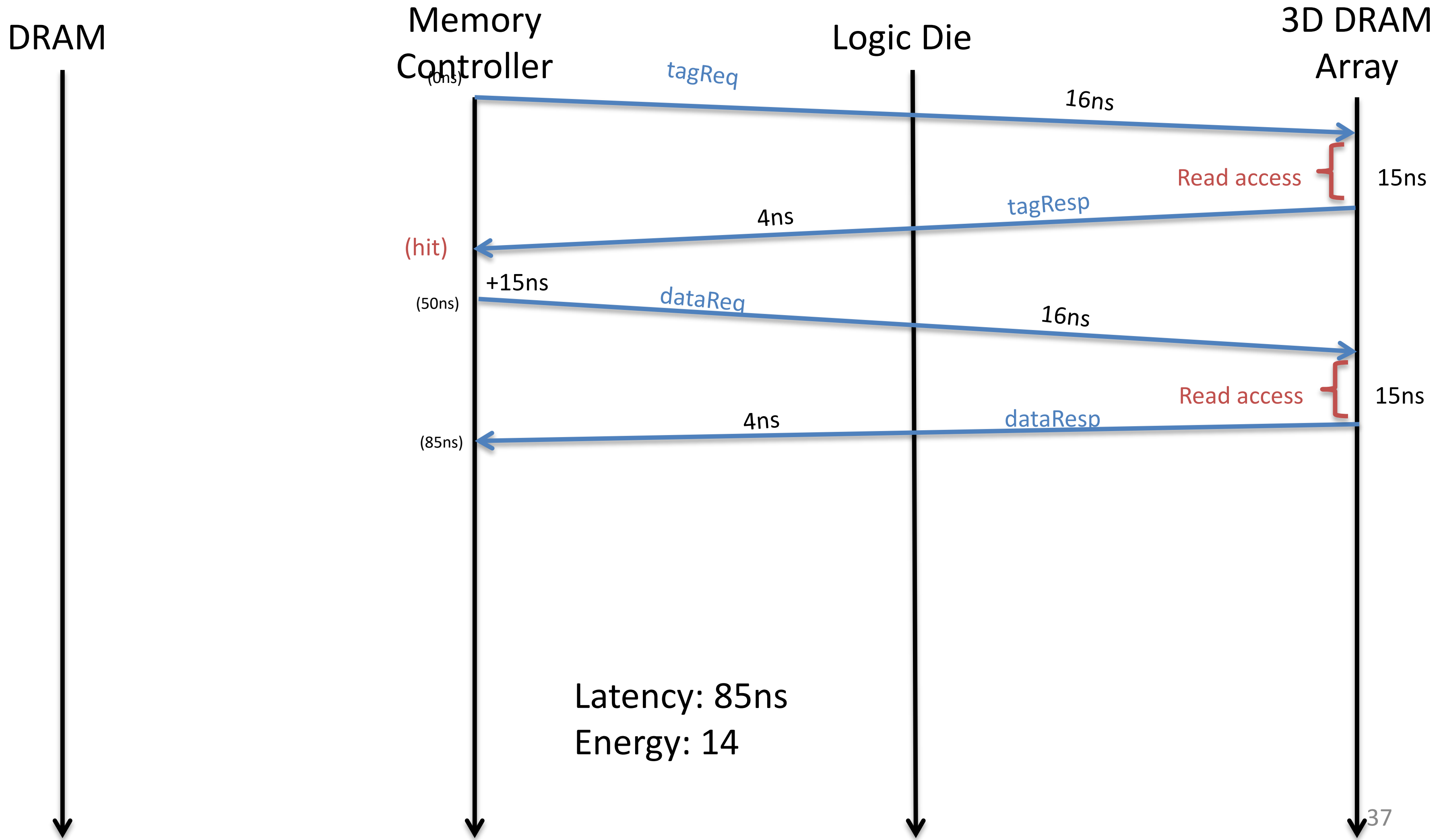
Estimation based on [1]

Outline

- Background
- Contribution 1: full-system simulation infrastructure
- Contribution 2: self-managed HBM cache
- Summary
- **Appendix**

Serial

- Read Hit
- Read Miss – Invalid, Read Miss – Valid Clean
- Read Miss – Valid Dirty
- Write Hit
- Write Miss – Invalid, Write Miss – Valid Clean
- Write Miss – Valid Dirty



DRAM

Memory Controller

Logic Die

3D DRAM Array

(0ns)

tagReq

16ns

Read access

15ns

tagResp

4ns

(hit)

+15ns

(50ns)

dataReq

16ns

Read access

15ns

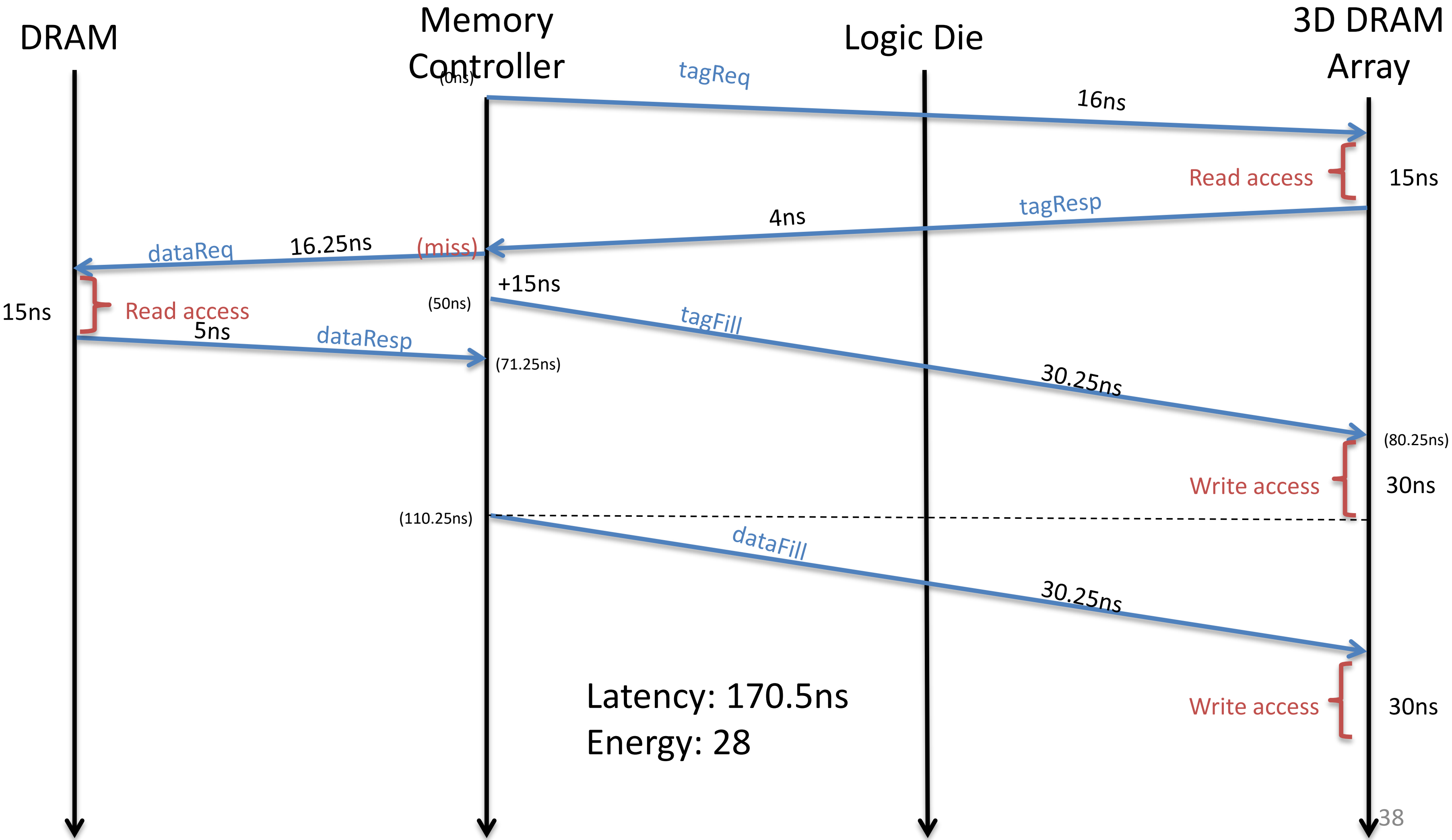
dataResp

4ns

(85ns)

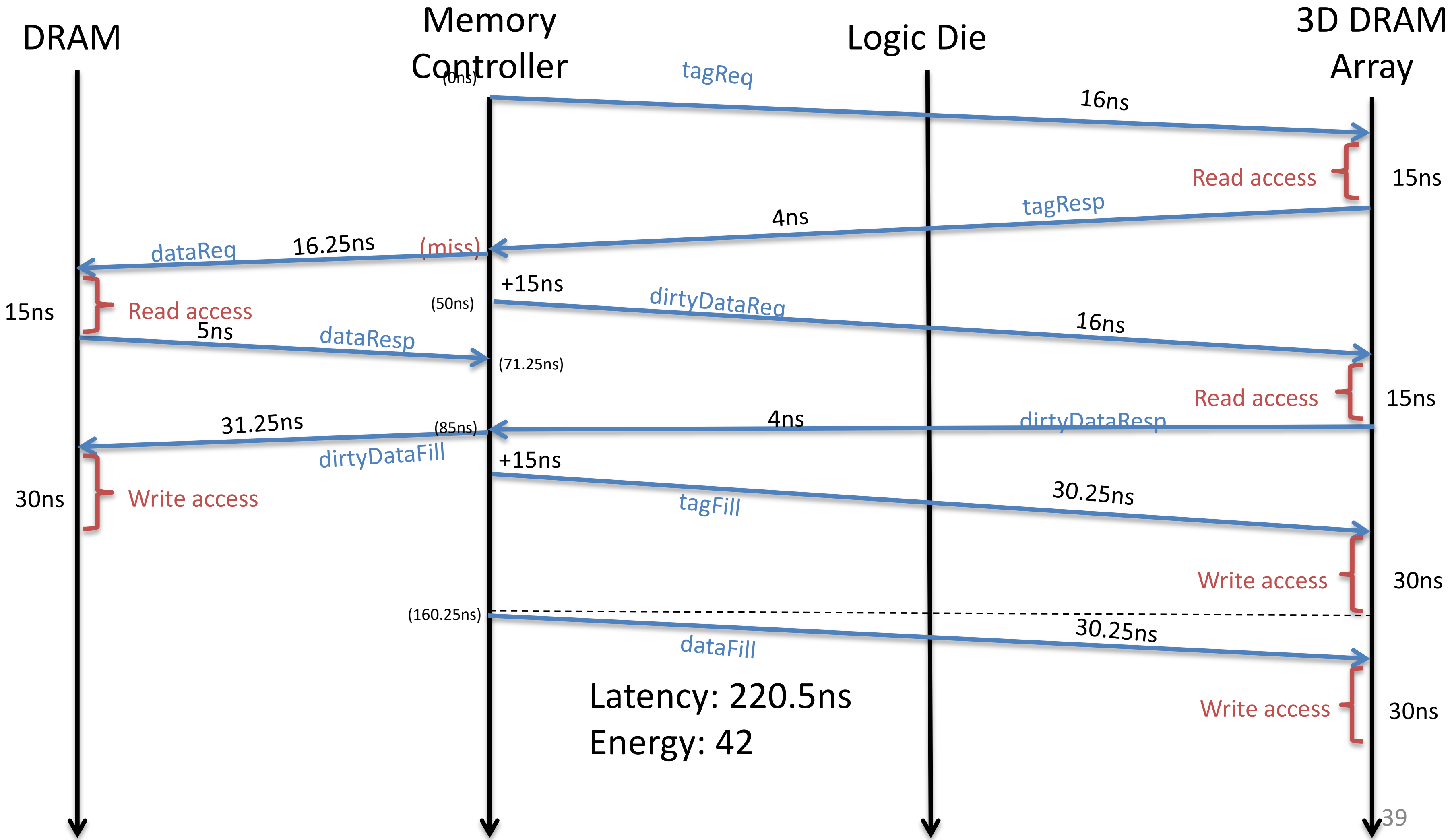
Latency: 85ns

Energy: 14



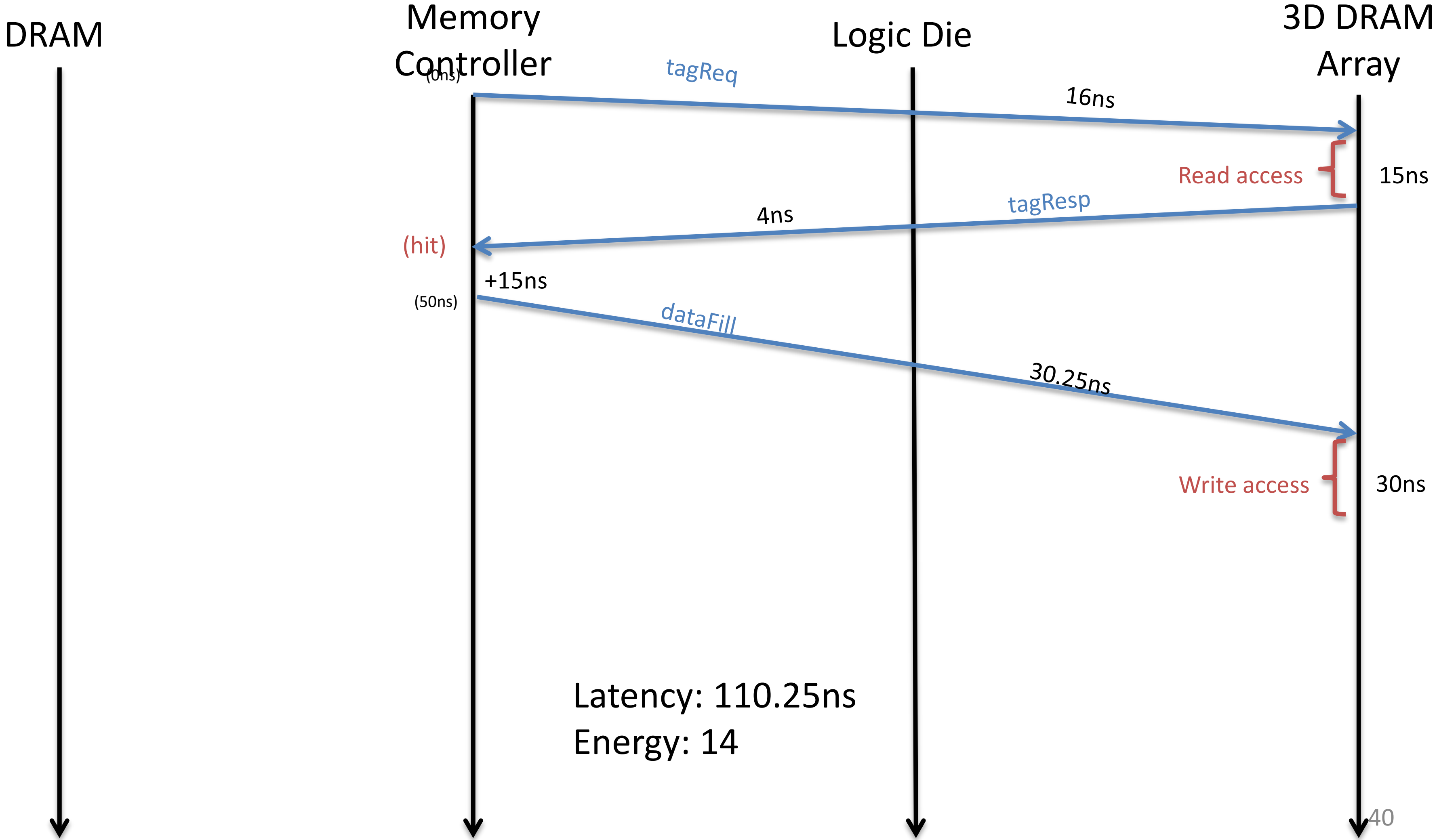
Latency: 170.5ns

Energy: 28



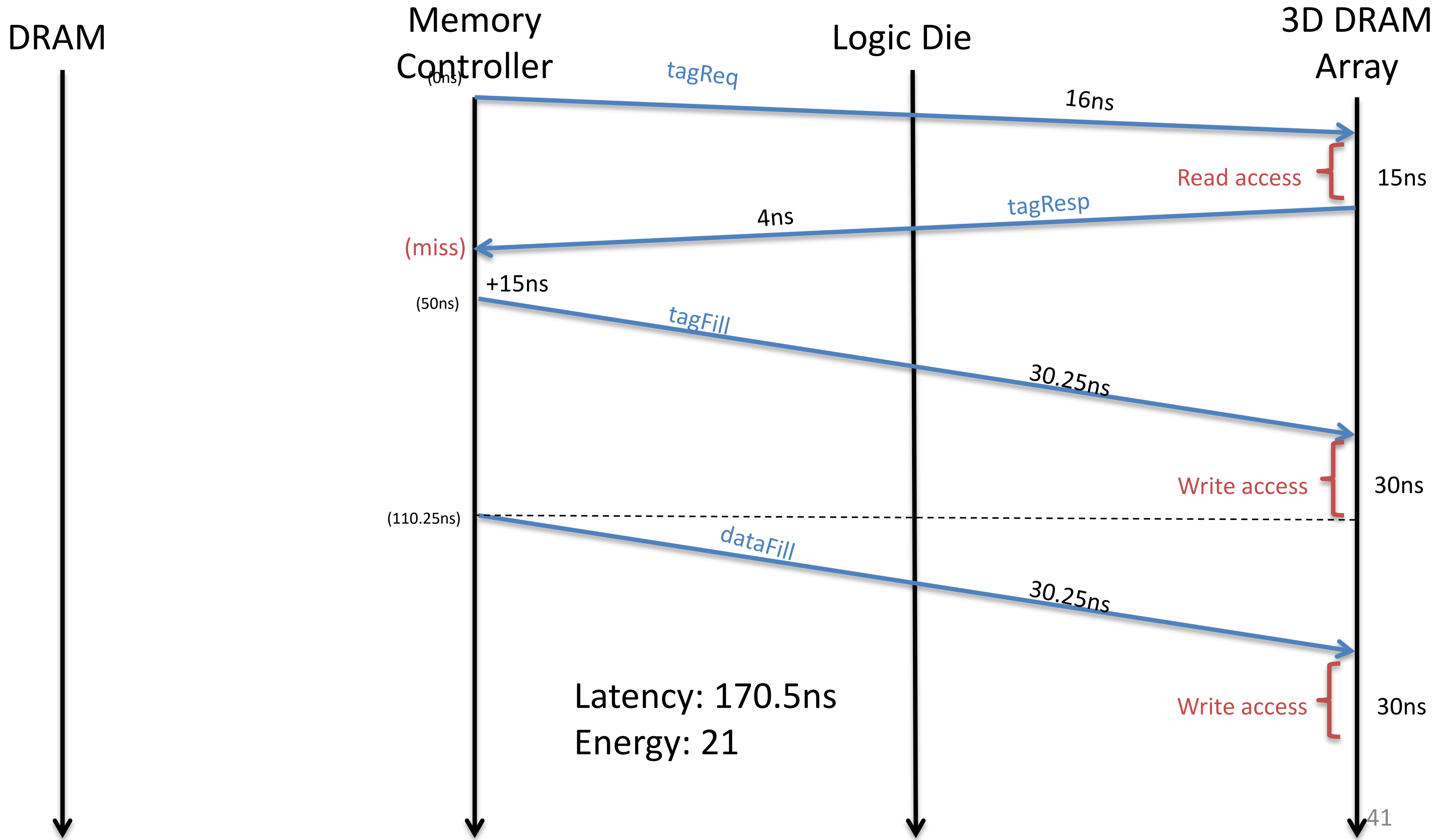
Latency: 220.5ns

Energy: 42



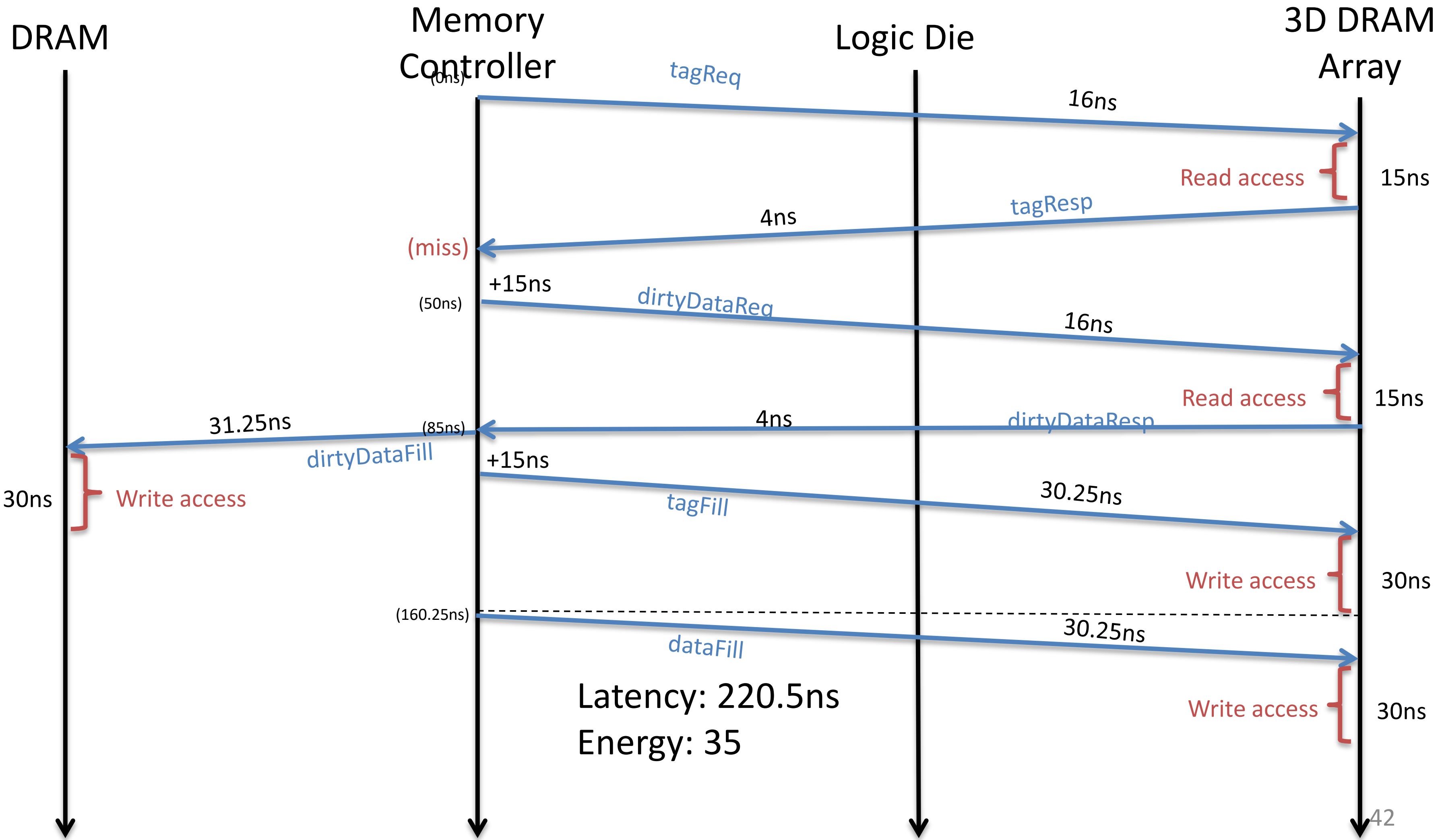
Latency: 110.25ns

Energy: 14



Latency: 170.5ns

Energy: 21

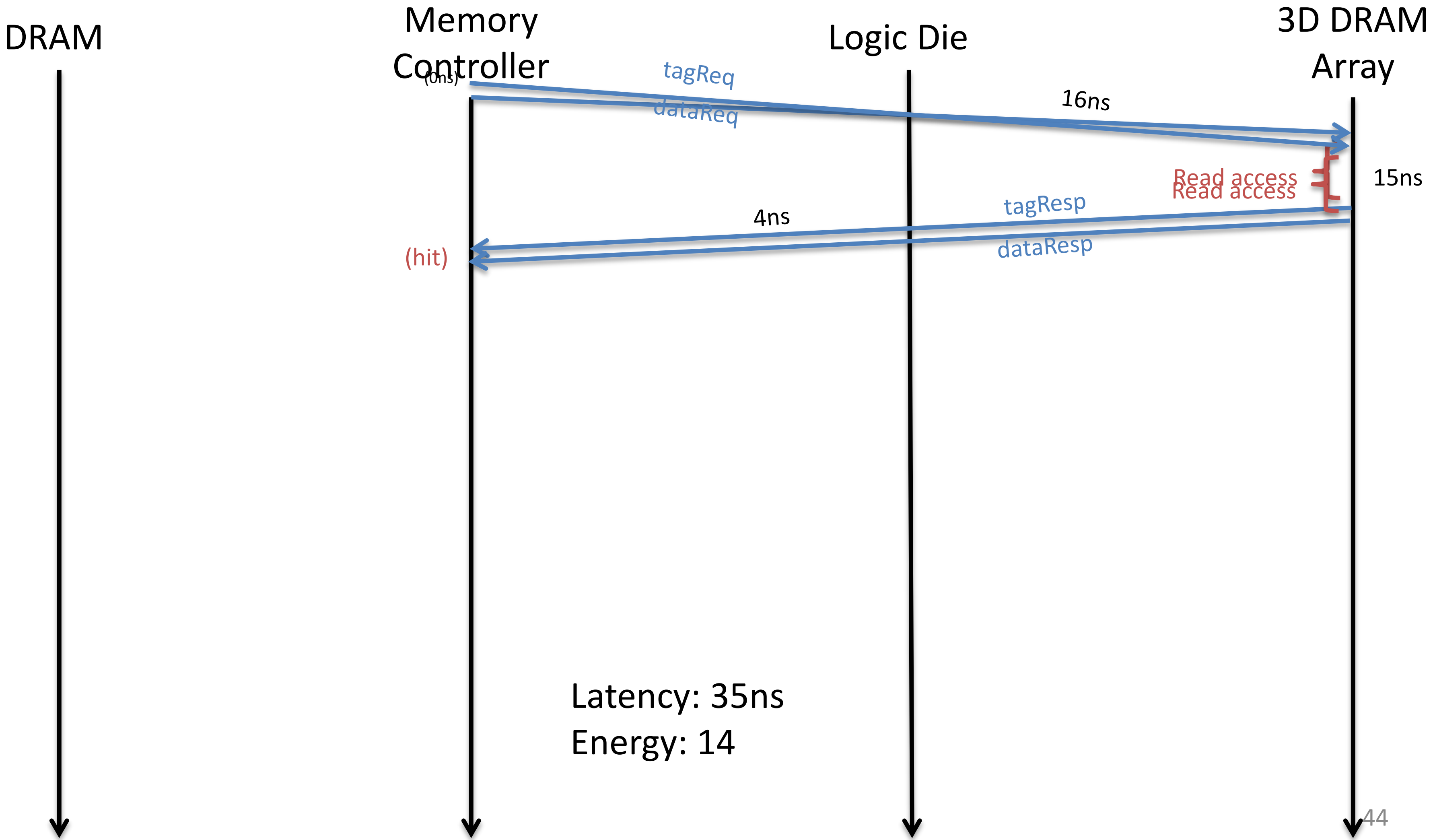


Latency: 220.5ns

Energy: 35

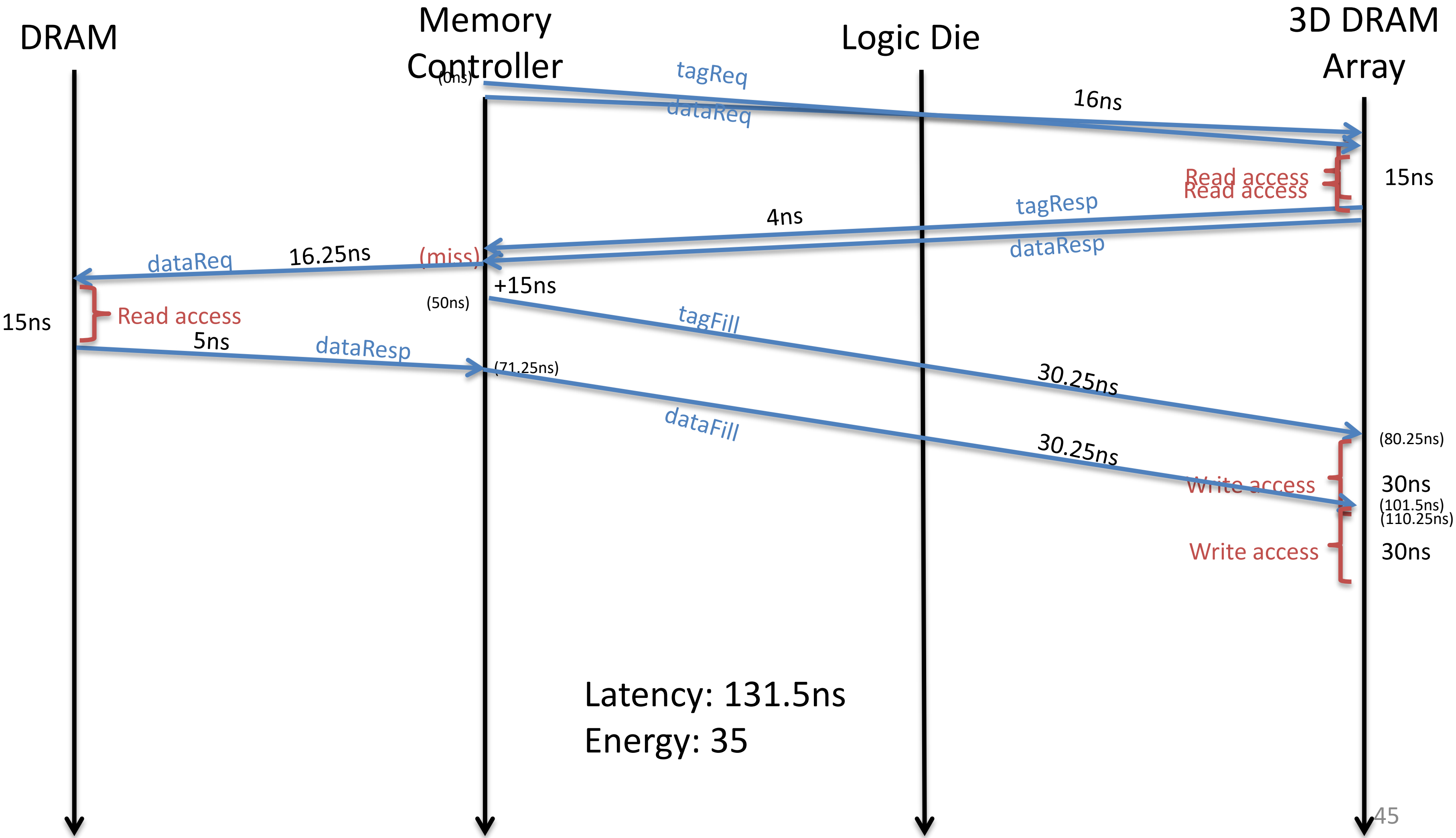
Parallel

- Read Hit
- Read Miss – Invalid, Read Miss – Valid Clean
- Read Miss – Valid Dirty
- Write Hit
- Write Miss – Invalid, Write Miss – Valid Clean
- Write Miss – Valid Dirty



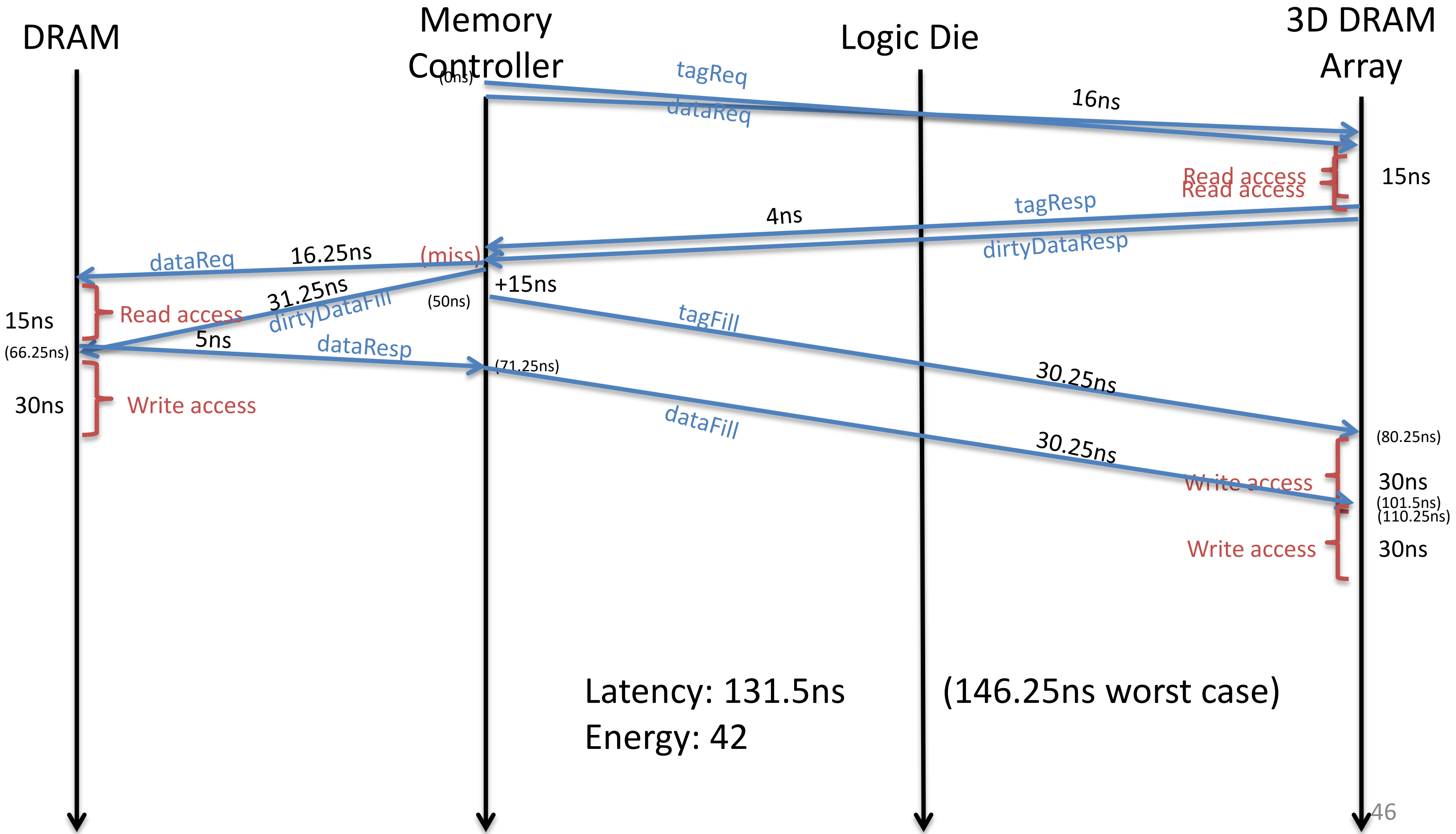
Latency: 35ns

Energy: 14



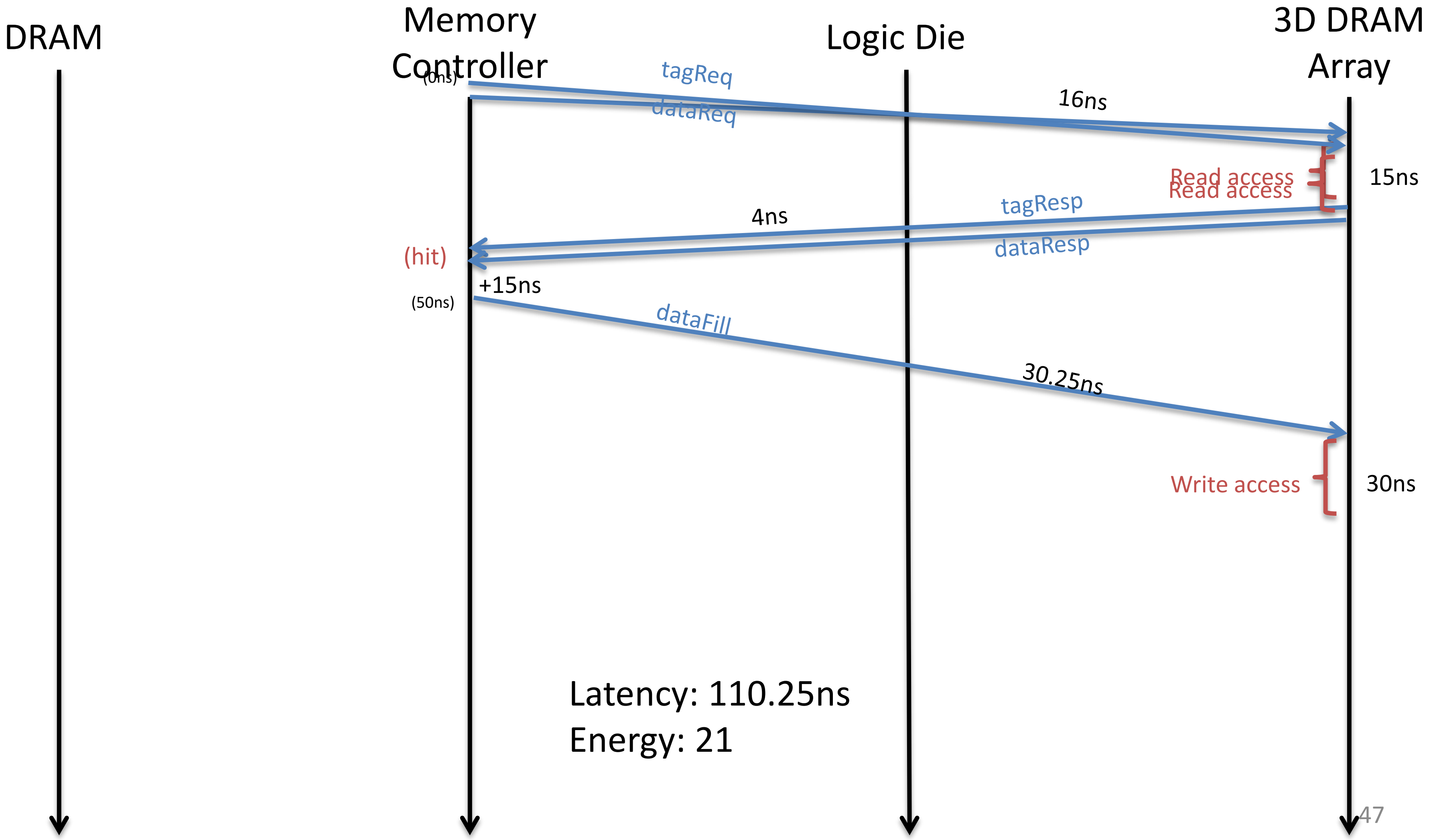
Latency: 131.5ns

Energy: 35



Latency: 131.5ns (146.25ns worst case)

Energy: 42



DRAM

Memory Controller

Logic Die

3D DRAM Array

(0ns)

tagReq

dataReq

16ns

Read access

Read access

15ns

tagResp

4ns

dataResp

(hit)

+15ns

(50ns)

dataFill

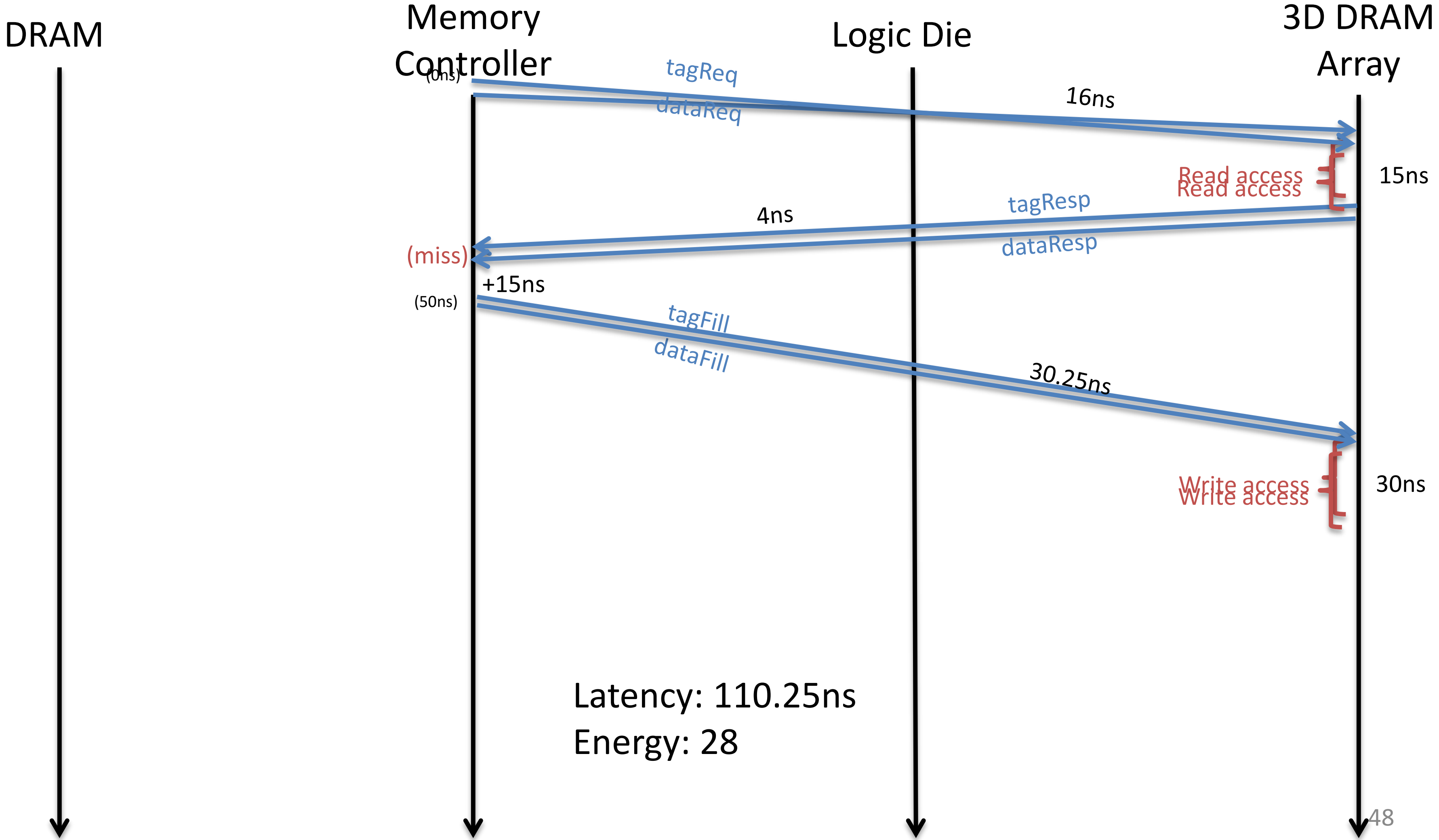
30.25ns

Write access

30ns

Latency: 110.25ns

Energy: 21



DRAM

Memory Controller

Logic Die

3D DRAM Array

(0ns)

tagReq

dataReq

16ns

Read access

Read access

15ns

tagResp

4ns

dataResp

(miss)

+15ns

(50ns)

tagFill

dataFill

30.25ns

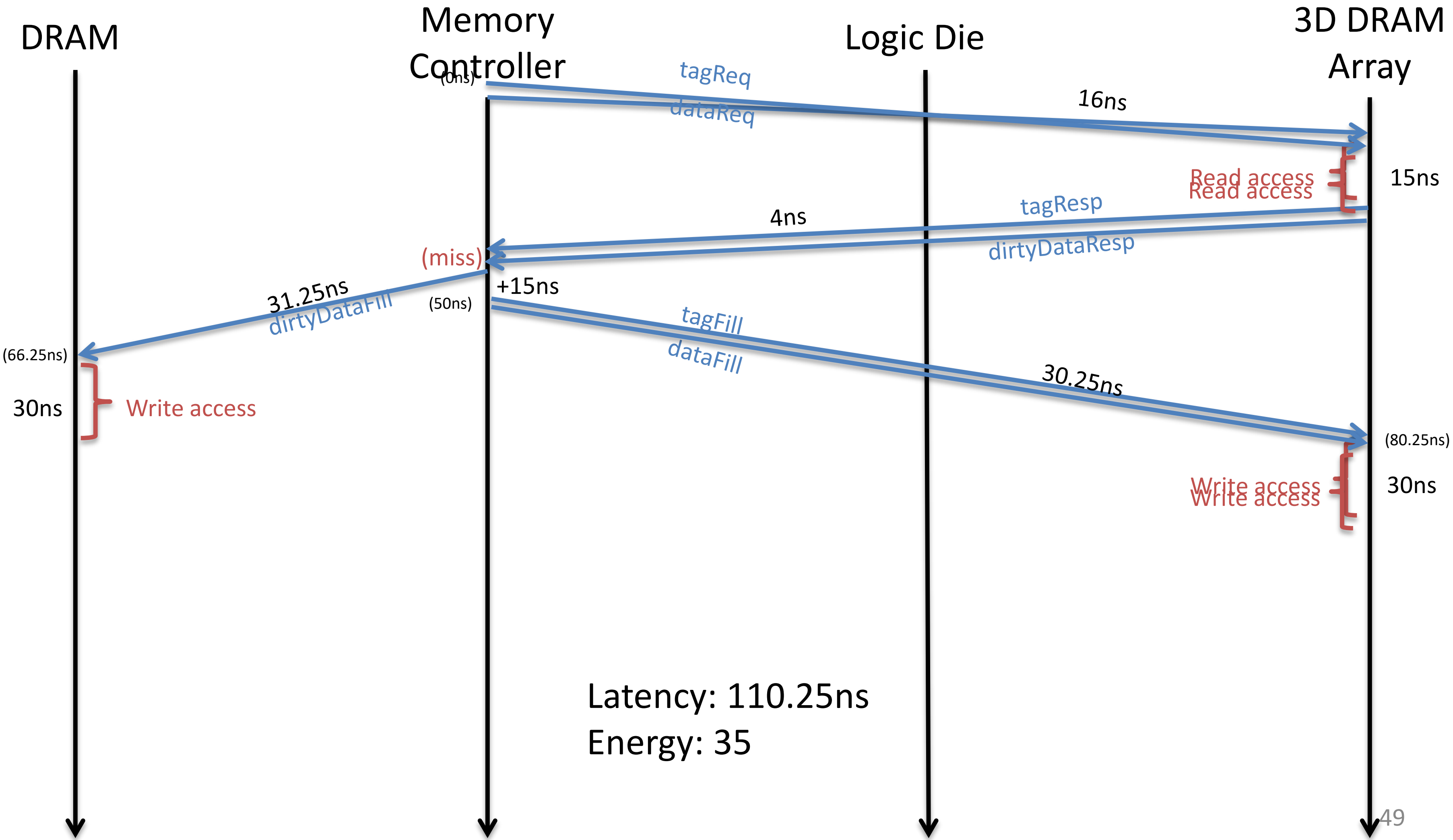
Write access

Write access

30ns

Latency: 110.25ns

Energy: 28



DRAM

Memory Controller

Logic Die

3D DRAM Array

(0ns)

tagReq

dataReq

16ns

Read access

Read access

15ns

tagResp

4ns

dirtyDataResp

(miss)

+15ns

(50ns)

31.25ns

dirtyDataFill

(66.25ns)

30ns

Write access

tagFill

dataFill

30.25ns

(80.25ns)

Write access

Write access

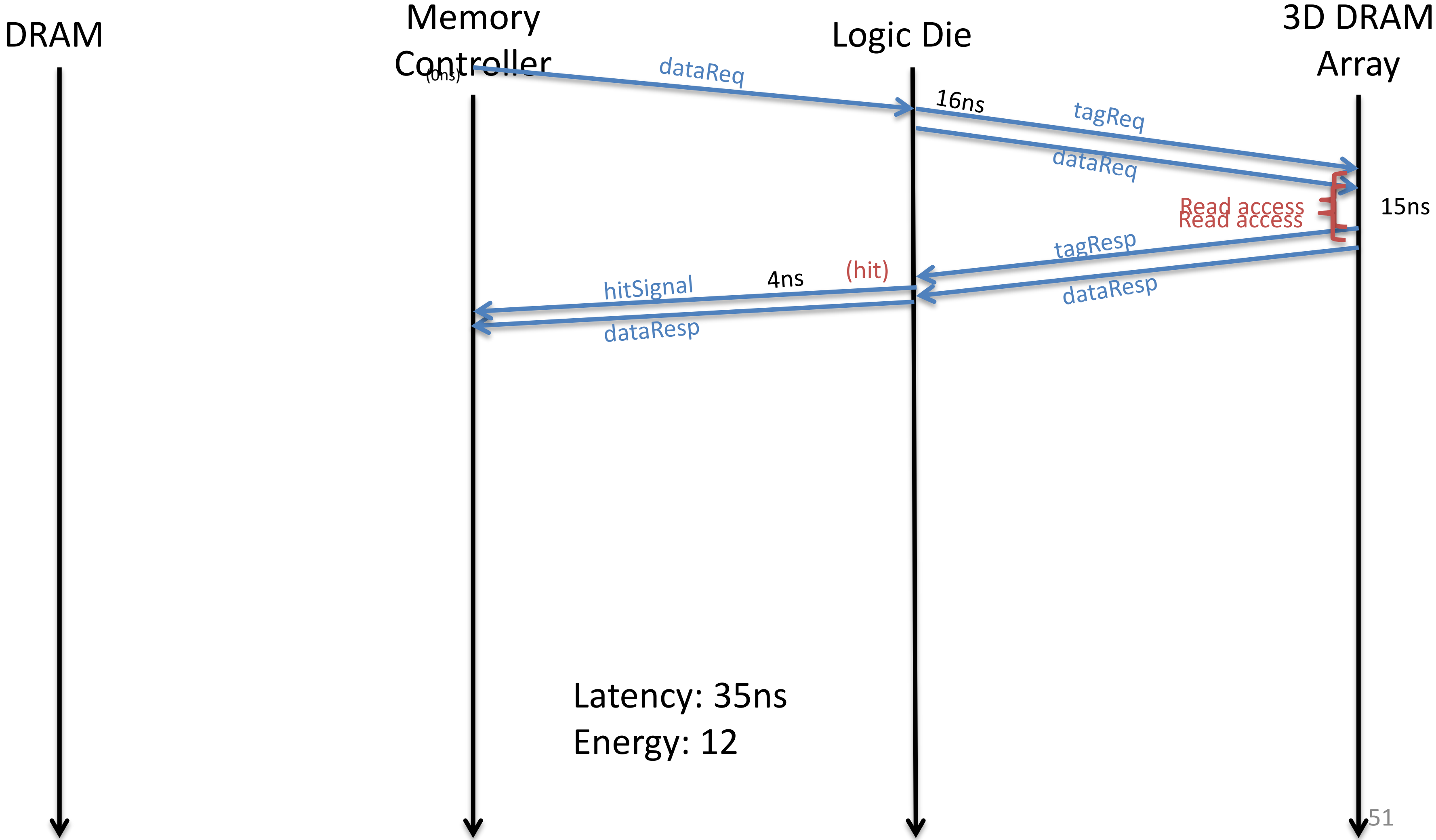
30ns

Latency: 110.25ns

Energy: 35

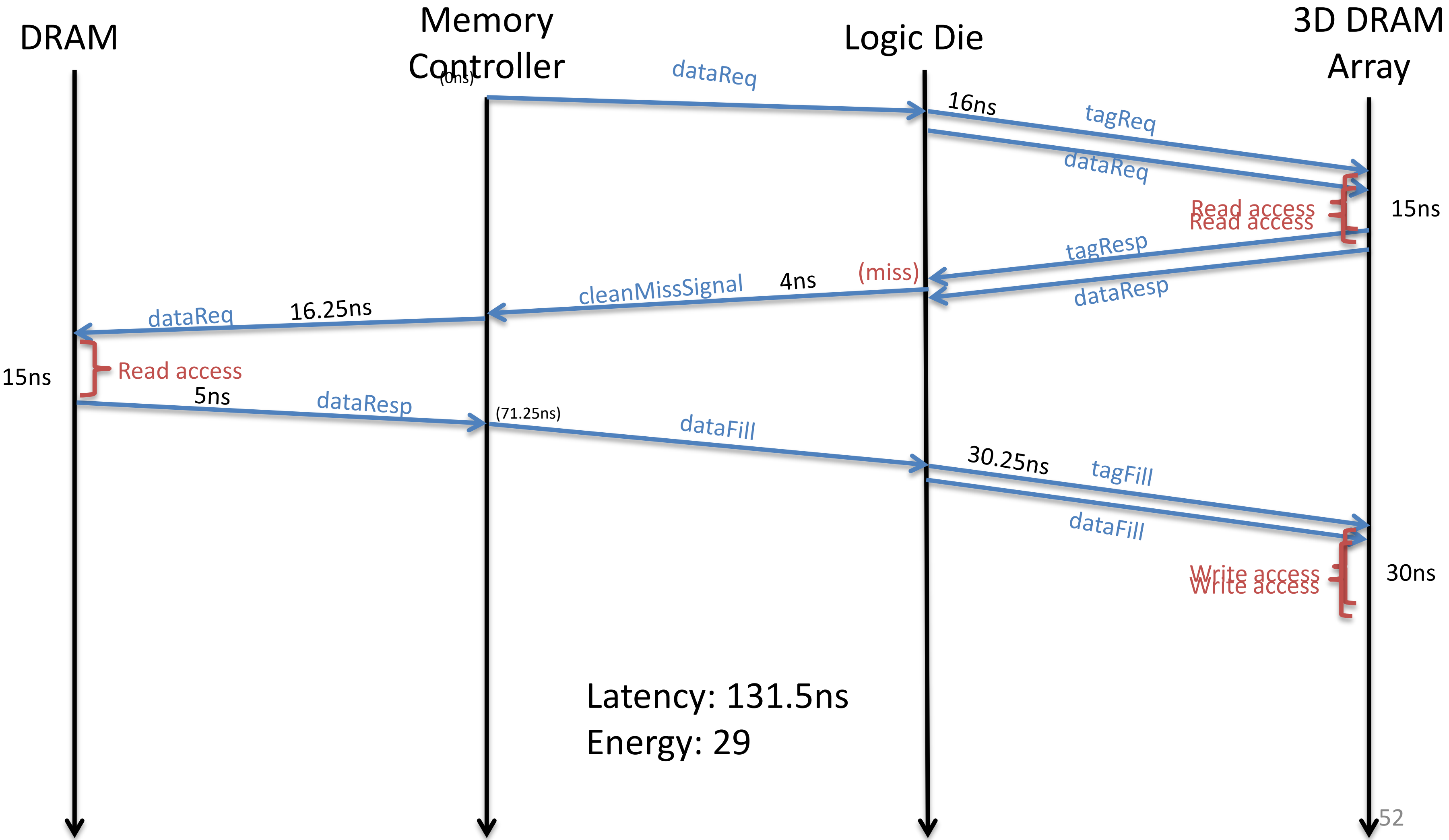
Latency Optimized

- Read Hit
- Read Miss – Invalid, Read Miss – Valid Clean
- Read Miss – Valid Dirty
- Write Hit
- Write Miss – Invalid, Write Miss – Valid Clean
- Write Miss – Valid Dirty



Latency: 35ns

Energy: 12



DRAM

Memory Controller

Logic Die

3D DRAM Array

(0ns)

dataReq

16ns

tagReq

dataReq

Read access

15ns

tagResp

dataResp

(miss)

4ns

cleanMissSignal

16.25ns

dataReq

15ns

Read access

5ns

dataResp

(71.25ns)

dataFill

30.25ns

tagFill

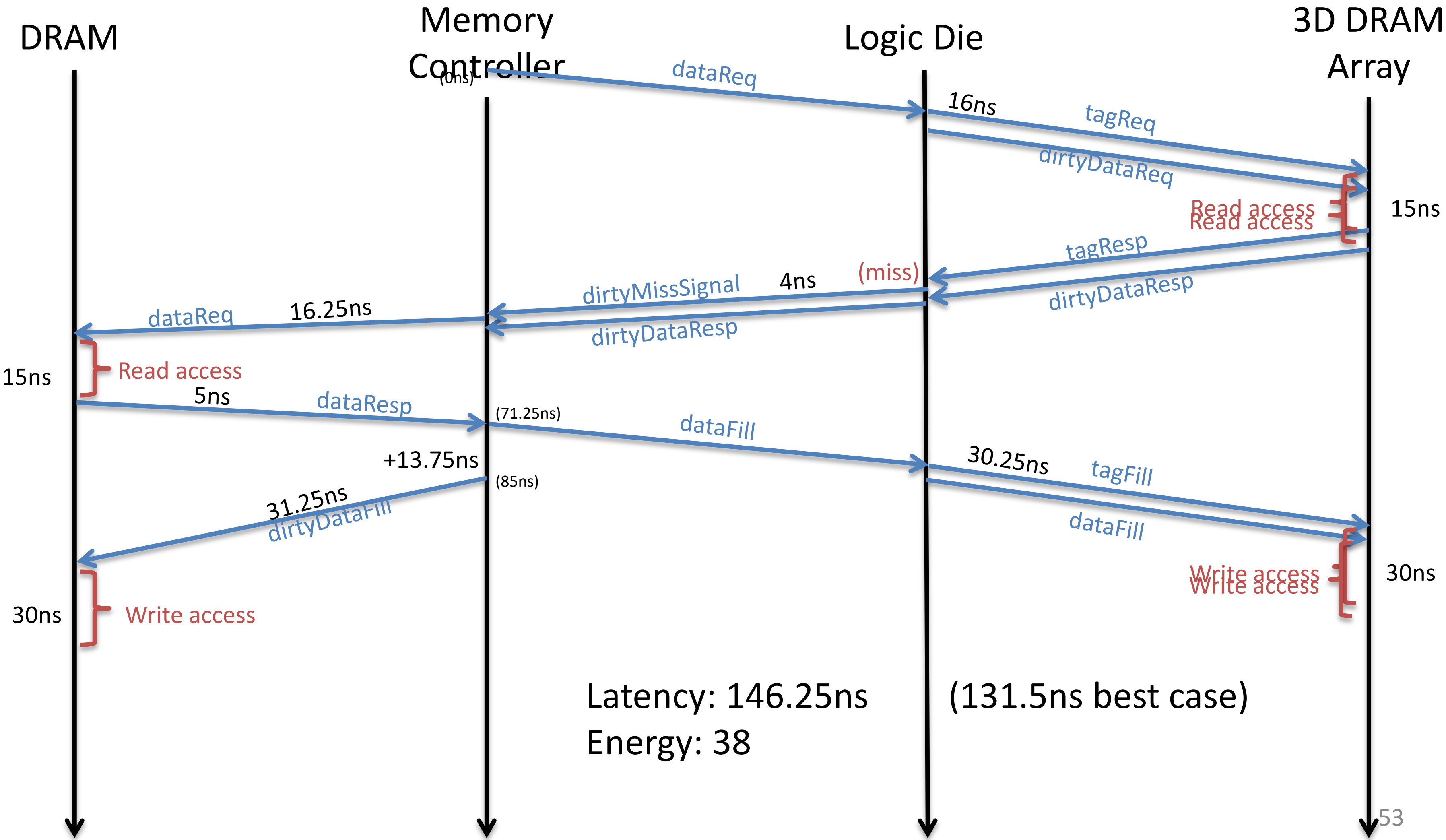
dataFill

Write access

30ns

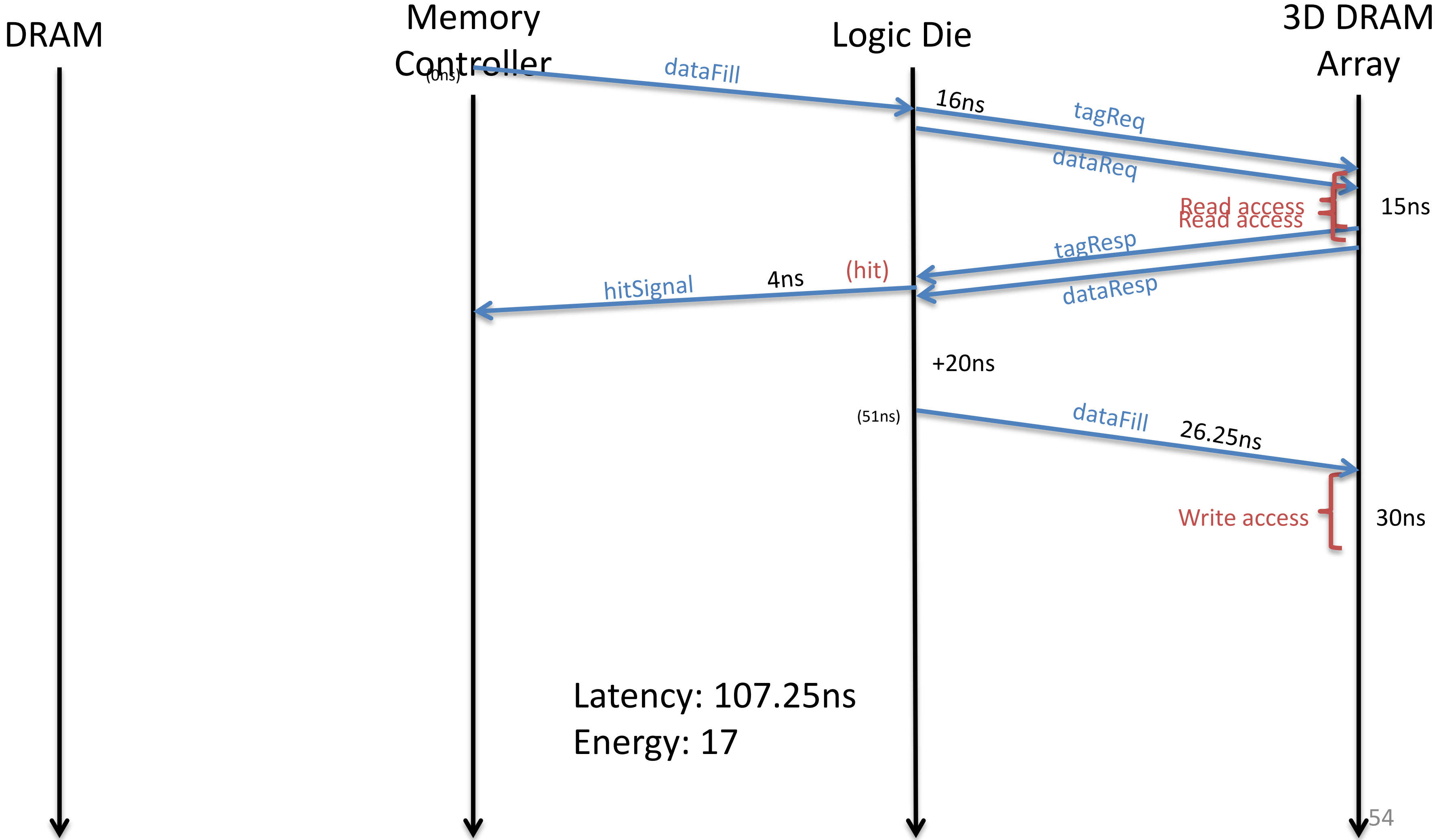
Latency: 131.5ns

Energy: 29



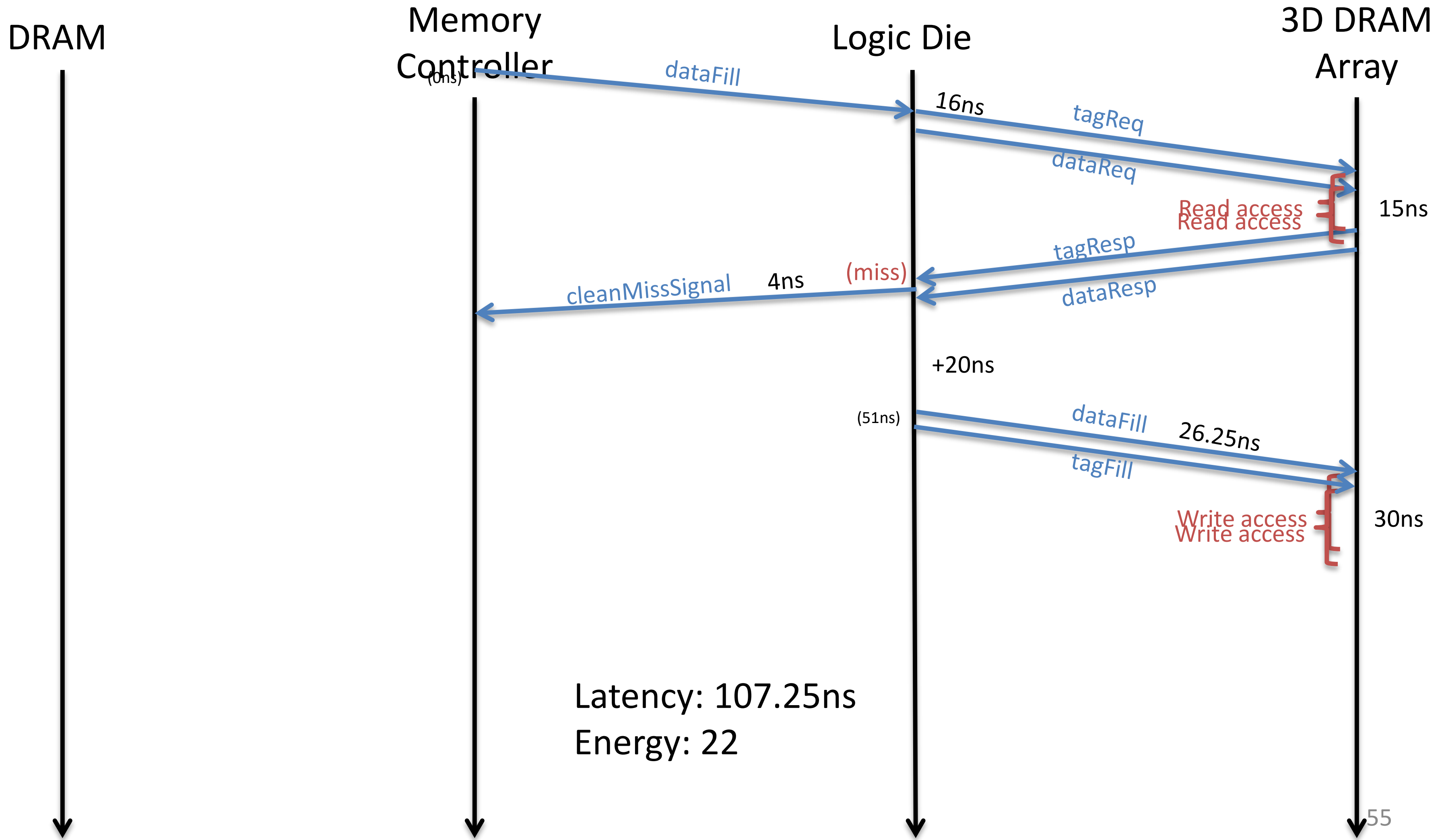
Latency: 146.25ns (131.5ns best case)

Energy: 38



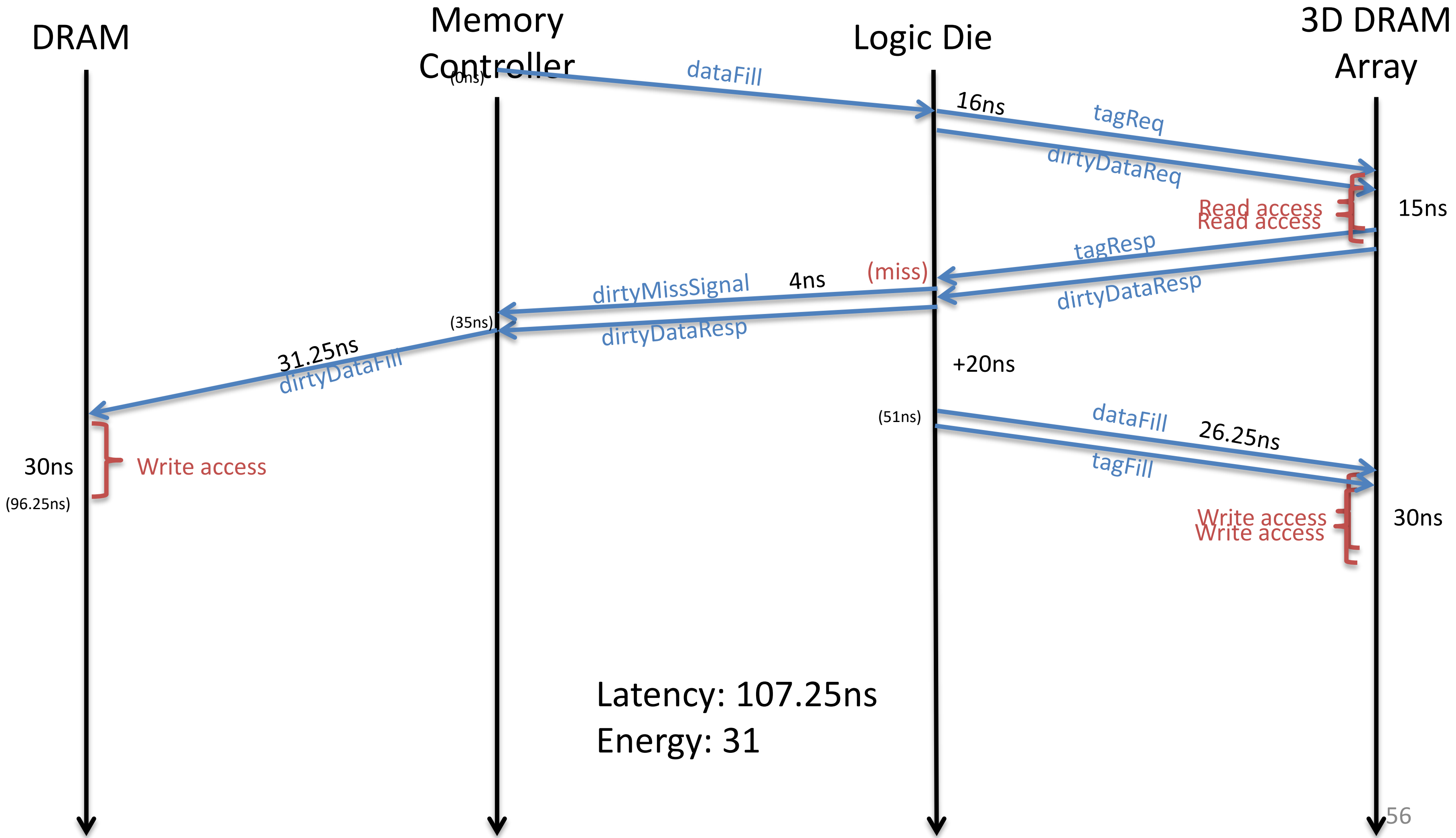
Latency: 107.25ns

Energy: 17



Latency: 107.25ns

Energy: 22



DRAM

Memory Controller

Logic Die

3D DRAM Array

(0ns)

dataFill

16ns

tagReq

dirtyDataReq

Read access

Read access

15ns

tagResp

(miss)

dirtyDataResp

4ns

dirtyMissSignal

(35ns)

dirtyDataResp

31.25ns

dirtyDataFill

+20ns

(51ns)

dataFill

26.25ns

tagFill

30ns

Write access

(96.25ns)

Write access

Write access

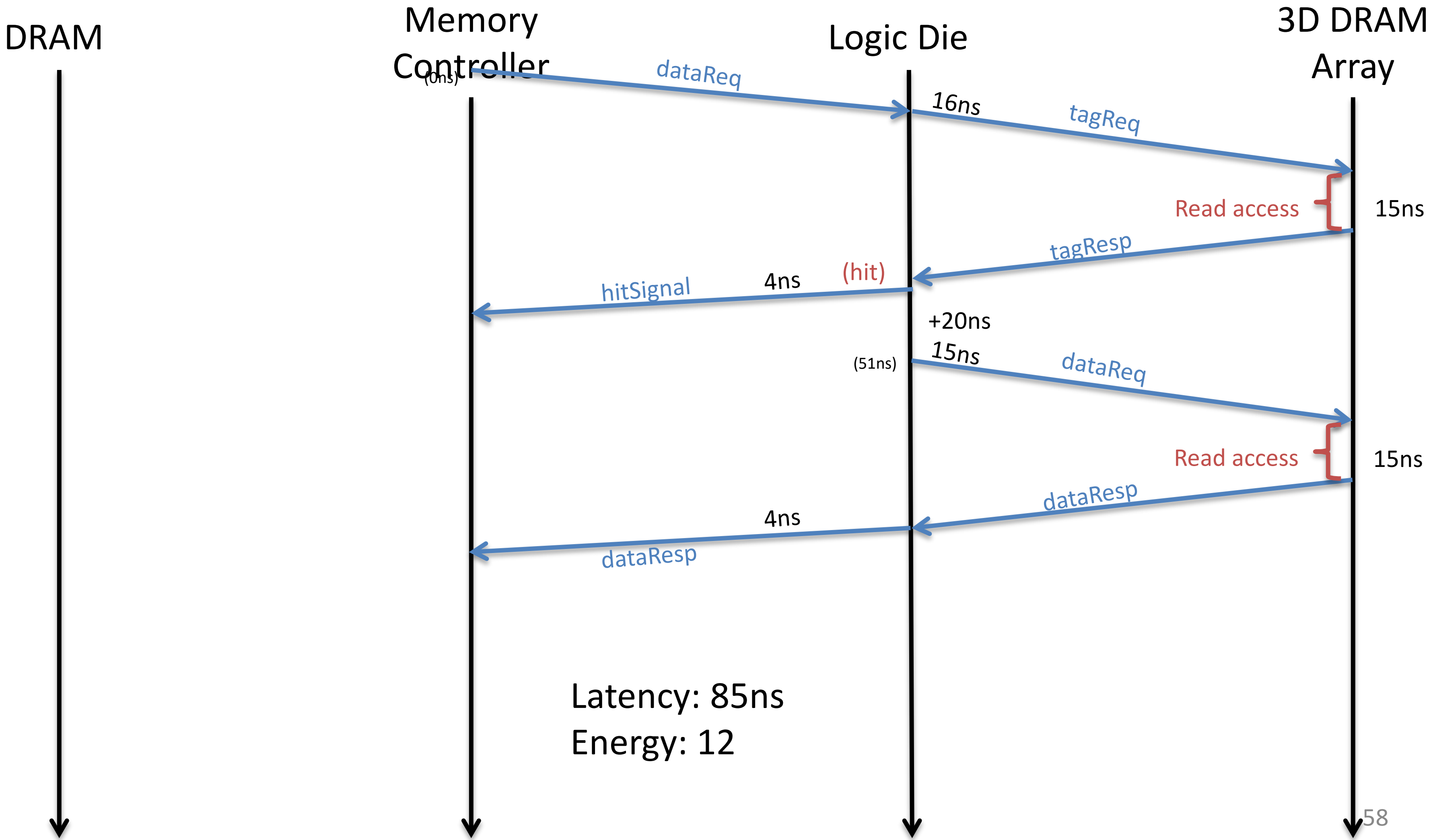
30ns

Latency: 107.25ns

Energy: 31

Energy Optimized

- Read Hit
- Read Miss – Invalid, Read Miss – Valid Clean
- Read Miss – Valid Dirty
- Write Hit
- Write Miss – Invalid, Write Miss – Valid Clean
- Write Miss – Valid Dirty



DRAM

Memory Controller

Logic Die

3D DRAM Array

(0ns)

dataReq

16ns

tagReq

Read access

15ns

tagResp

(hit)

4ns

hitSignal

+20ns

15ns

(51ns)

dataReq

Read access

15ns

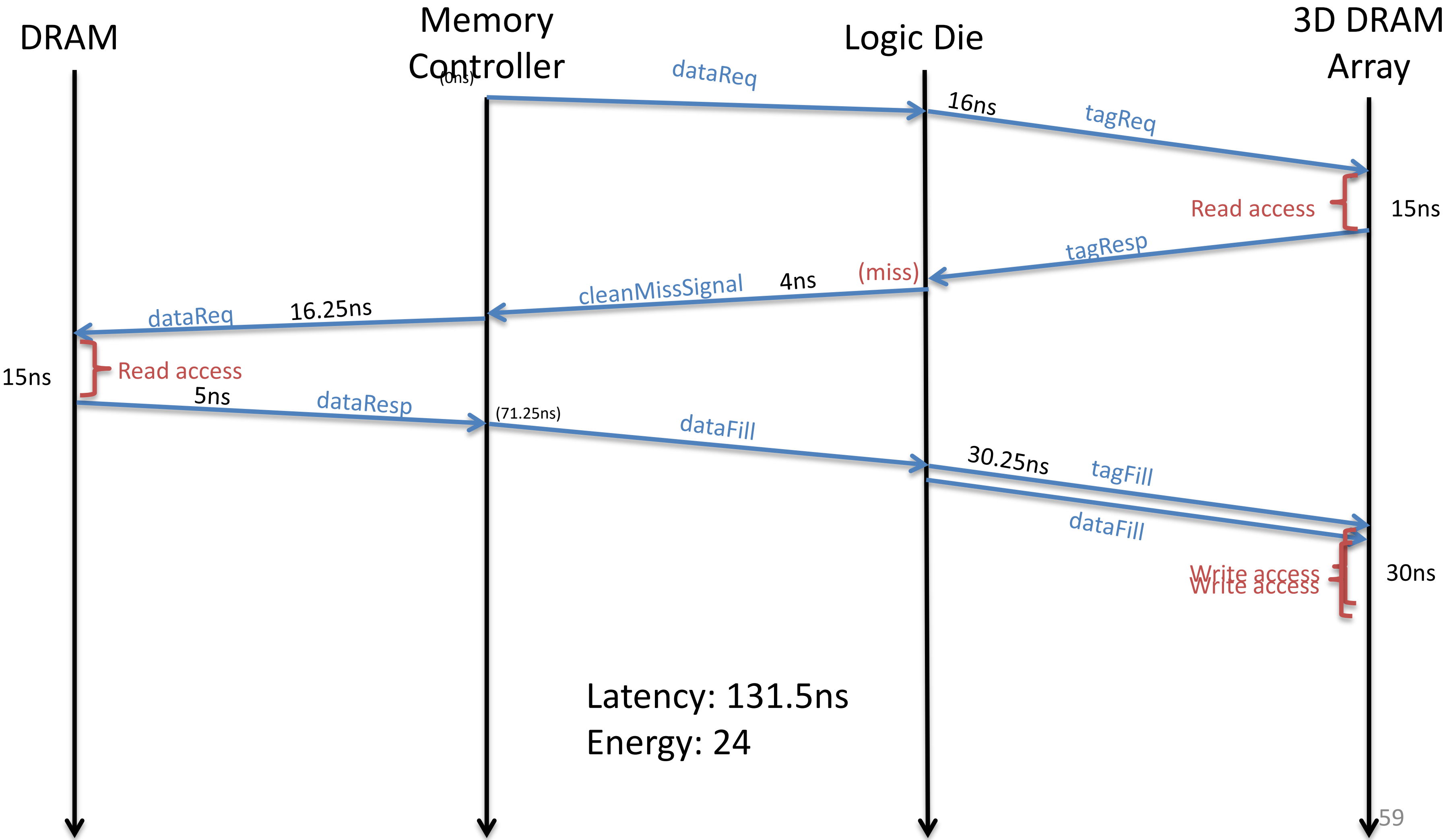
dataResp

4ns

dataResp

Latency: 85ns

Energy: 12



DRAM

Memory Controller

Logic Die

3D DRAM Array

(0ns)

dataReq

16ns

tagReq

Read access

15ns

tagResp

(miss)

4ns

cleanMissSignal

16.25ns

dataReq

15ns

Read access

5ns

dataResp

(71.25ns)

dataFill

30.25ns

tagFill

dataFill

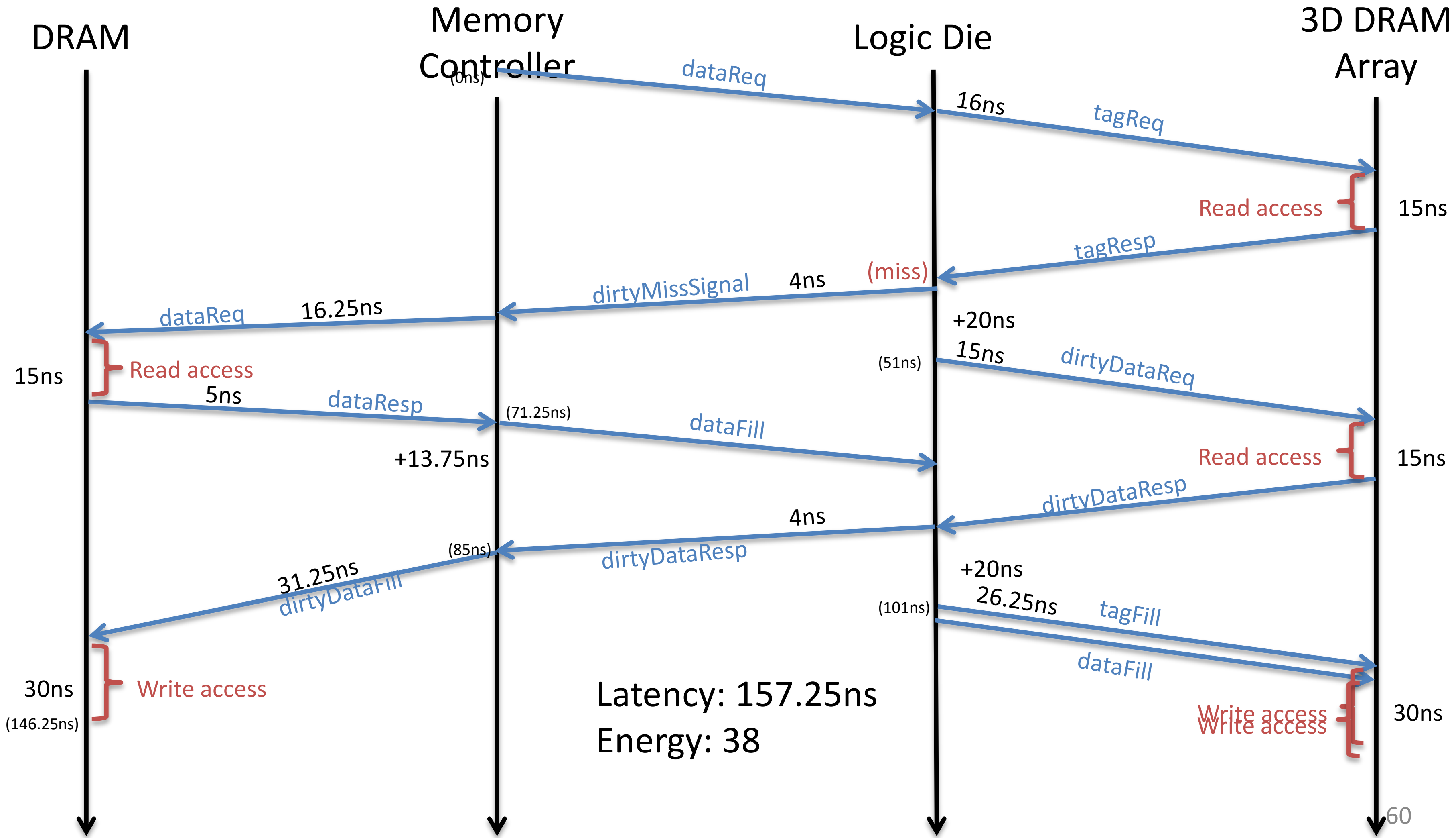
Write access

Write access

30ns

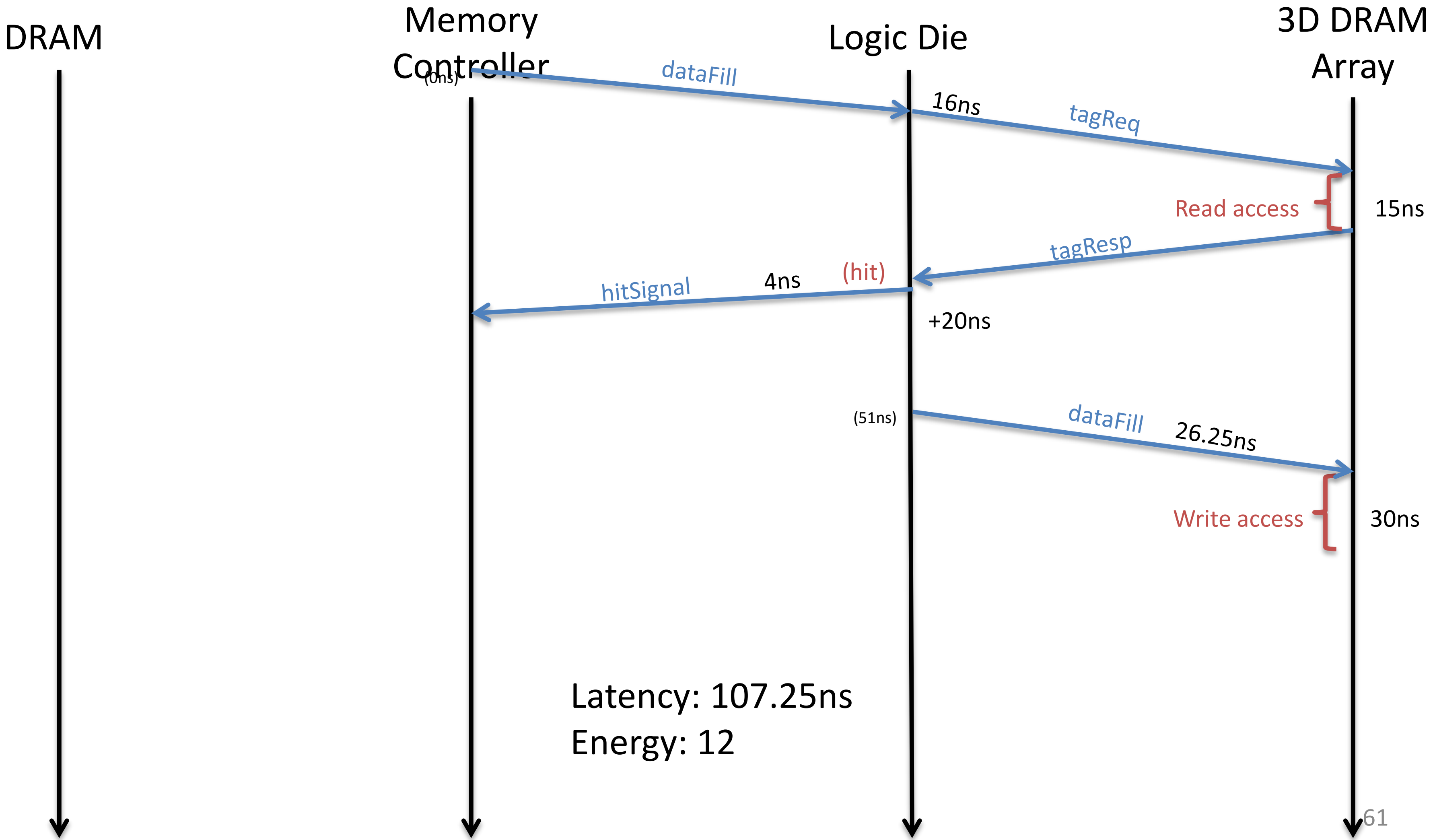
Latency: 131.5ns

Energy: 24



Latency: 157.25ns

Energy: 38



DRAM

Memory Controller

Logic Die

3D DRAM Array

(0ns)

dataFill

16ns

tagReq

Read access

15ns

tagResp

(hit)

4ns

hitSignal

+20ns

(51ns)

dataFill

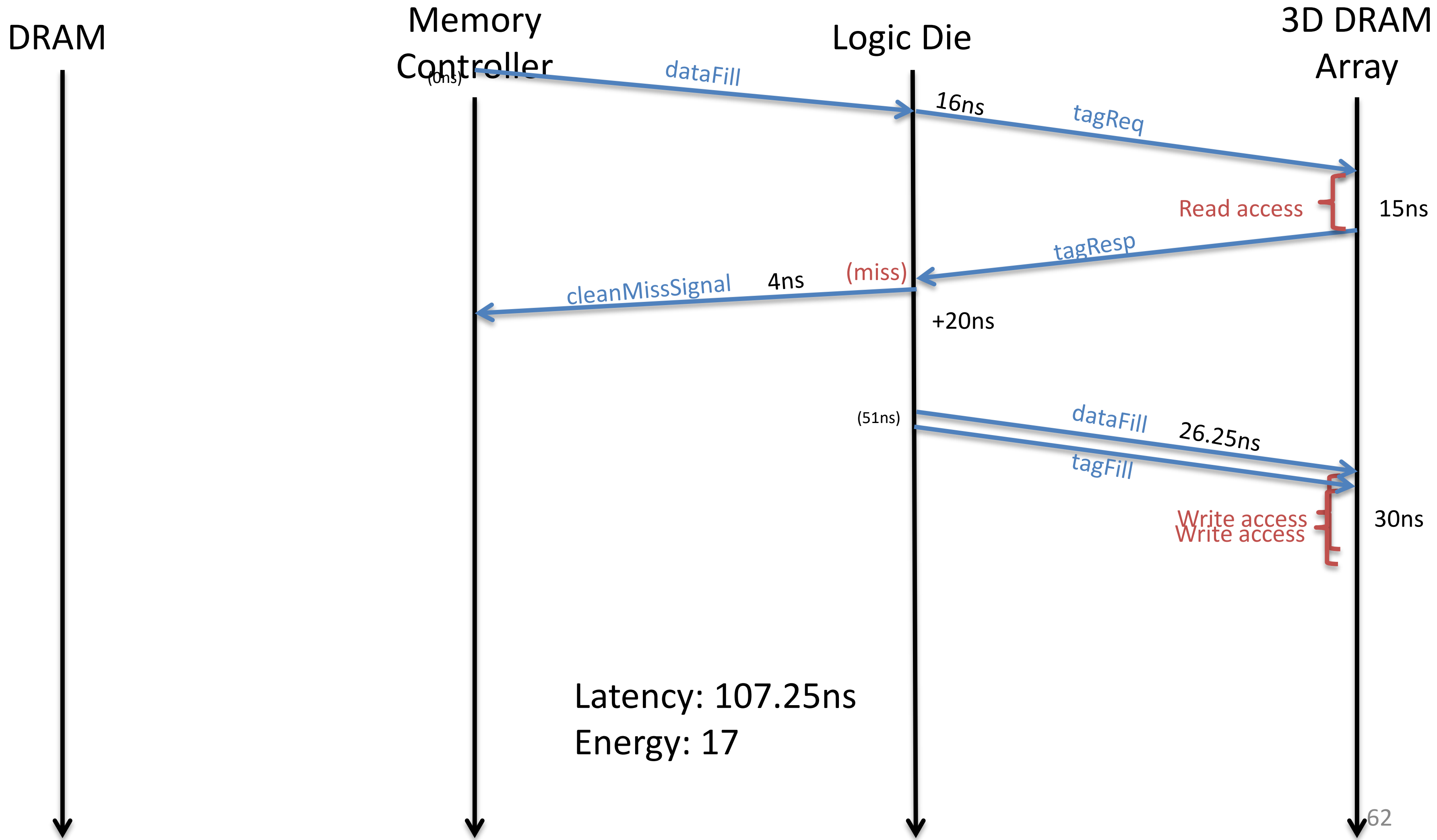
26.25ns

Write access

30ns

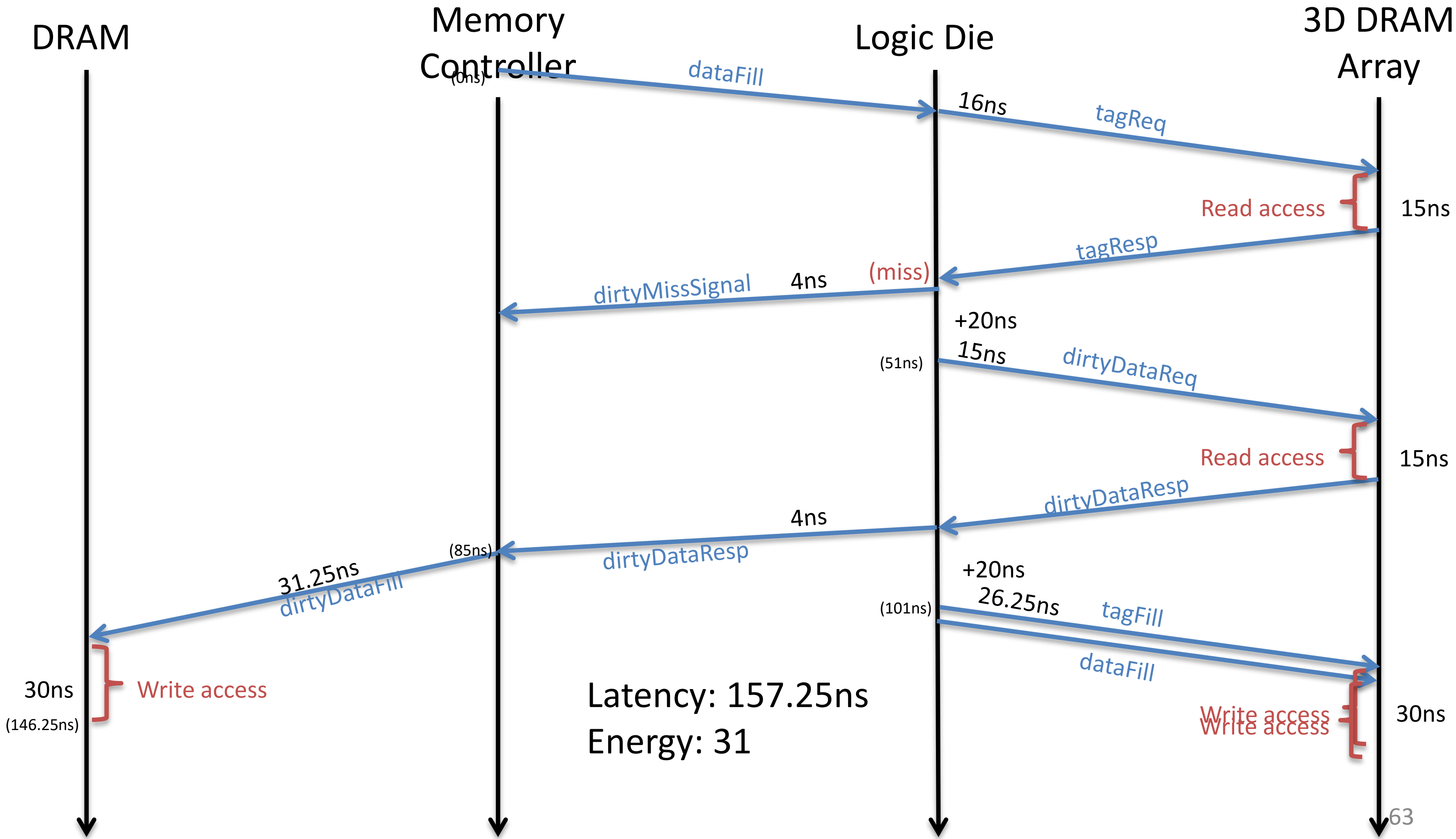
Latency: 107.25ns

Energy: 12



Latency: 107.25ns

Energy: 17



DRAM

Memory Controller

Logic Die

3D DRAM Array

(0ns)

dataFill

16ns

tagReq

Read access

15ns

tagResp

(miss)

4ns

dirtyMissSignal

+20ns

15ns

(51ns)

dirtyDataReq

Read access

15ns

dirtyDataResp

4ns

(85ns)

dirtyDataResp

31.25ns

dirtyDataFill

+20ns

26.25ns

(101ns)

tagFill

dataFill

Write access

Write access

30ns

30ns

Write access

(146.25ns)

Latency: 157.25ns

Energy: 31