Rethinking Memory System Design for Data-Intensive Computing

Onur Mutlu
onur@cmu.edu
June 20, 2014
ASAP 2014, Zurich

Carnegie Mellon
Main memory is a critical component of all computing systems: server, mobile, embedded, desktop, sensor.

Main memory system must scale (in size, technology, efficiency, cost, and management algorithms) to maintain performance growth and technology scaling benefits.
Memory System: A Shared Resource View

Shared Memory

Shared Memory Control

Shared L3 Cache

Core 1

Shared L2 Cache

Core 2

Shared L2 Cache

Core 3

Shared L2 Cache

Core 4

Shared L2 Cache

Core 5

Shared L2 Cache

Core 6

Shared L2 Cache

Core 7

Shared L2 Cache

Core 8

Shared L2 Cache

Core 9

Shared L2 Cache

Shared Memory Control

Shared L3 Cache

Shared Memory

Shared Memory Control

Shared L3 Cache

Shared Memory

Storage

SAFARI
State of the Main Memory System

- Recent technology, architecture, and application trends
  - lead to new requirements
  - exacerbate old requirements

- DRAM and memory controllers, as we know them today, are (will be) unlikely to satisfy all requirements

- Some emerging non-volatile memory technologies (e.g., PCM) enable new opportunities: memory+storage merging

- We need to rethink the main memory system
  - to fix DRAM issues and enable emerging technologies
  - to satisfy all requirements
Agenda

- Major Trends Affecting Main Memory
- The Memory Scaling Problem and Solution Directions
  - New Memory Architectures
  - Enabling Emerging Technologies: Hybrid Memory Systems
- How Can We Do Better?
- Summary
Major Trends Affecting Main Memory (I)

- Need for main memory capacity, bandwidth, QoS increasing

- Main memory energy/power is a key system design concern

- DRAM technology scaling is ending
Major Trends Affecting Main Memory (II)

- Need for main memory capacity, bandwidth, QoS increasing
  - Multi-core: increasing number of cores/agents
  - Data-intensive applications: increasing demand/hunger for data
  - Consolidation: cloud computing, GPUs, mobile, heterogeneity

- Main memory energy/power is a key system design concern

- DRAM technology scaling is ending
Example: The Memory Capacity Gap

- Memory capacity per core expected to drop by 30% every two years
- Trends worse for memory bandwidth per core!
Major Trends Affecting Main Memory (III)

- Need for main memory capacity, bandwidth, QoS increasing

- Main memory energy/power is a key system design concern
  - ~40-50% energy spent in off-chip memory hierarchy [Lefurgy, IEEE Computer 2003]
  - DRAM consumes power even when not used (periodic refresh)

- DRAM technology scaling is ending
Major Trends Affecting Main Memory (IV)

- Need for main memory capacity, bandwidth, QoS increasing
- Main memory energy/power is a key system design concern
- DRAM technology scaling is ending
  - ITRS projects DRAM will not scale easily below X nm
  - Scaling has provided many benefits:
    - higher capacity (density), lower cost, lower energy
Agenda

- Major Trends Affecting Main Memory
- The Memory Scaling Problem and Solution Directions
  - New Memory Architectures
  - Enabling Emerging Technologies: Hybrid Memory Systems
- How Can We Do Better?
- Summary
The DRAM Scaling Problem

- DRAM stores charge in a capacitor (charge-based memory)
  - Capacitor must be large enough for reliable sensing
  - Access transistor should be large enough for low leakage and high retention time
  - Scaling beyond 40-35nm (2013) is challenging [ITRS, 2009]

- DRAM capacity, cost, and energy/power hard to scale
Repeatedly opening and closing a row induces *disturbance errors* in adjacent rows in most real DRAM chips [Kim+ ISCA 2014]
Most DRAM Modules Are at Risk

A company

86% (37/43)

Up to $1.0 \times 10^7$ errors

B company

83% (45/54)

Up to $2.7 \times 10^6$ errors

C company

88% (28/32)

Up to $3.3 \times 10^5$ errors

Solutions to the DRAM Scaling Problem

- Two potential solutions
  - Tolerate DRAM (by taking a fresh look at it)
  - Enable emerging memory technologies to eliminate/minimize DRAM

- Do both
  - Hybrid memory systems
Solution 1: Tolerate DRAM

- Overcome DRAM shortcomings with
  - System-DRAM co-design
  - Novel DRAM architectures, interface, functions
  - Better waste management (efficient utilization)

- Key issues to tackle
  - Reduce energy
  - Enable reliability at low cost
  - Improve bandwidth and latency
  - Reduce waste
Solution 1: Tolerate DRAM


Avoid DRAM:

Solution 2: Emerging Memory Technologies

- Some emerging resistive memory technologies seem more scalable than DRAM (and they are non-volatile)

  Example: Phase Change Memory
  - Expected to scale to 9nm (2022 [ITRS])
  - Expected to be denser than DRAM: can store multiple bits/cell

- But, emerging technologies have shortcomings as well
  - Can they be enabled to replace/augment/surpass DRAM?

Hybrid Memory Systems

Hardware/software manage data allocation and movement to achieve the best of multiple technologies

Cores’ interfere with each other when accessing shared main memory
An Orthogonal Issue: Memory Interference

- **Problem:** Memory interference between cores is uncontrolled
  -> unfairness, starvation, low performance
  -> uncontrollable, unpredictable, vulnerable system

- **Solution:** QoS-Aware Memory Systems
  - Hardware designed to provide a configurable fairness substrate
    - Application-aware memory scheduling, partitioning, throttling
  - Software designed to configure the resources to satisfy different QoS goals

- QoS-aware memory controllers and interconnects can provide predictable performance and higher efficiency
Designing QoS-Aware Memory Systems: Approaches

- **Smart resources:** Design each shared resource to have a configurable interference control/reduction mechanism
  - QoS-aware memory controllers [Mutlu+ MICRO’07] [Moscibroda+, Usenix Security’07] [Mutlu+ ISCA’08, Top Picks’09] [Kim+ HPCA’10] [Kim+ MICRO’10, Top Picks’11] [Ebrahimi+ ISCA’11, MICRO’11] [Ausavarungnirun+, ISCA’12][Subramanian+, HPCA’13] [Kim+, RTAS’14]
  - QoS-aware interconnects [Das+ MICRO’09, ISCA’10, Top Picks ’11] [Grot+ MICRO’09, ISCA’11, Top Picks ’12]
  - QoS-aware caches

- **Dumb resources:** Keep each resource free-for-all, but reduce/control interference by injection control or data mapping
  - Source throttling to control access to memory system [Ebrahimi+ ASPLOS’10, ISCA’11, TOCS’12] [Ebrahimi+ MICRO’09] [Nychis+ HotNets’10] [Nychis+ SIGCOMM’12]
  - QoS-aware data mapping to memory controllers [Muralidhara+ MICRO’11]
  - QoS-aware thread scheduling to cores [Das+ HPCA’13]
Some Current Directions

- **New memory/storage + compute architectures**
  - Rethinking DRAM and flash memory
  - Processing close to data; accelerating bulk operations
  - Ensuring memory/storage reliability and robustness

- **Enabling emerging NVM technologies**
  - Hybrid memory systems with automatic data management
  - Coordinated management of memory and storage with NVM

- **System-level memory/storage QoS**
  - QoS-aware controller and system design
  - Coordinated memory + storage QoS
Agenda

- Major Trends Affecting Main Memory
- The Memory Scaling Problem and Solution Directions
  - New Memory Architectures
  - Enabling Emerging Technologies: Hybrid Memory Systems
- How Can We Do Better?
- Summary
Tolerating DRAM: Example Techniques

- Retention-Aware DRAM Refresh: Reducing Refresh Impact
- Refresh Access Parallelization: Reducing Refresh Impact
- Tiered-Latency DRAM: Reducing DRAM Latency
- RowClone: Accelerating Page Copy and Initialization
- Subarray-Level Parallelism: Reducing Bank Conflict Impact
- Linearly Compressed Pages: Efficient Memory Compression
DRAM Refresh

- DRAM capacitor charge leaks over time

- The memory controller needs to refresh each row periodically to restore charge
  - Activate each row every N ms
  - Typical N = 64 ms

- Downsides of refresh
  -- Energy consumption: Each refresh consumes energy
  -- Performance degradation: DRAM rank/bank unavailable while refreshed
  -- QoS/predictability impact: (Long) pause times during refresh
  -- Refresh rate limits DRAM capacity scaling
Refresh Overhead: Performance

![Graph showing the percentage of time spent refreshing for different device capacities. Present: 8% for 2 Gb, 8% for 4 Gb, 8% for 8 Gb, 8% for 16 Gb, and 46% for 64 Gb. Future: 46% for 64 Gb.](image)
Refresh Overhead: Energy

% DRAM energy spent refreshing

Present

Future

Device capacity

2 Gb
4 Gb
8 Gb
16 Gb
32 Gb
64 Gb

15%

47%
Retention Time Profile of DRAM

64-128ms

>256ms

128-256ms
RAIDR: Eliminating Unnecessary Refreshes

- Observation: Most DRAM rows can be refreshed much less often without losing data [Kim+, EDL’09][Liu+ ISCA’13]

- Key idea: Refresh rows containing weak cells more frequently, other rows less frequently
  1. Profiling: Profile retention time of all rows
  2. Binning: Store rows into bins by retention time in memory controller
     *Efficient storage with Bloom Filters* (only 1.25KB for 32GB memory)
  3. Refreshing: Memory controller refreshes rows in different bins at different rates

- Results: 8-core, 32GB, SPEC, TPC-C, TPC-H
  - 74.6% refresh reduction @ 1.25KB storage
  - ~16%/20% DRAM dynamic/idle power reduction
  - ~9% performance improvement
  - Benefits increase with DRAM capacity

Going Forward (for DRAM and Flash)

- **How to find out and expose weak memory cells/rows**

- **Low-cost system-level tolerance of memory errors**

- **Tolerating cell-to-cell interference at the system level**
Experimental Infrastructure (DRAM)


Experimental Infrastructure (DRAM)

Experimental Infrastructure (Flash)

Tolerating DRAM: Example Techniques

- Retention-Aware DRAM Refresh: Reducing Refresh Impact
- Refresh Access Parallelization: Reducing Refresh Impact

- Tiered-Latency DRAM: Reducing DRAM Latency

- RowClone: Accelerating Page Copy and Initialization

- Subarray-Level Parallelism: Reducing Bank Conflict Impact

- Linearly Compressed Pages: Efficient Memory Compression
DRAM latency continues to be a critical bottleneck, especially for response time-sensitive workloads.
What Causes the Long Latency?

$$\text{DRAM Latency} = \text{Subarray Latency} + \text{I/O Latency}$$

Dominant

Subarray
Why is the Subarray So Slow?

- Long bitline
  - Amortizes sense amplifier cost → Small area
  - Large bitline capacitance → High latency & power
Trade-Off: Area (Die Size) vs. Latency

Long Bitline

Faster

Smaller

Short Bitline

Trade-Off: Area vs. Latency
Trade-Off: Area (Die Size) vs. Latency

- Cheaper
- Normalized DRAM Area
- Latency (ns)

Commodity DRAM
Long Bitline

Fancy DRAM
Short Bitline

512 cells/bitline

GOAL

Cheaper
Faster
Approximating the Best of Both Worlds

- **Long Bitline**
  - Small Area
  - High Latency

- **Short Bitline**
  - Large Area
  - Low Latency

- **Our Proposal**
  - Need Isolation
  - Add Isolation Transistors
  - Bitline Fast
Approximating the Best of Both Worlds

Long Bitline Tiered-Latency DRAM Short Bitline

Small Area

High Latency

Small Area

Low Latency

Large Area

Low Latency

Small area using long bitline

Low Latency

42
Tiered-Latency DRAM

• Divide a bitline into two segments with an isolation transistor
Commodity DRAM vs. TL-DRAM

- DRAM Latency ($t_{RC}$)
- DRAM Power

- DRAM Area Overhead
  ~3%: mainly due to the isolation transistors
Trade-Off: Area (Die-Area) vs. Latency

Normalized DRAM Area

Latency (ns)

Cheaper

Near Segment

Faster

Far Segment

64 cells/bitline
Leveraging Tiered-Latency DRAM

• TL-DRAM is a *substrate* that can be leveraged by the hardware and/or software

• Many potential uses

1. Use near segment as hardware-managed *inclusive* cache to far segment
2. Use near segment as hardware-managed *exclusive* cache to far segment
3. Profile-based page mapping by operating system
4. Simply replace DRAM with TL-DRAM
Using near segment as a cache improves performance and reduces power consumption
Tolerating DRAM: Example Techniques

- Retention-Aware DRAM Refresh: Reducing Refresh Impact
- Refresh Access Parallelization: Reducing Refresh Impact
- Tiered-Latency DRAM: Reducing DRAM Latency
- RowClone: Accelerating Page Copy and Initialization
- Subarray-Level Parallelism: Reducing Bank Conflict Impact
- Linearly Compressed Pages: Efficient Memory Compression
Today's Memory: Bulk Data Copy

1) High latency
2) High bandwidth utilization
3) Cache pollution
4) Unwanted data movement

1046ns, 3.6uJ
Future: RowClone (In-Memory Copy)

1) Low latency
2) Low bandwidth utilization
3) No cache pollution
4) No unwanted data movement
5) Memory

100ns, 3.6uJ
90ns, 0.04uJ
DRAM Subarray Operation (load one byte)

Step 1: Activate row

Step 2: Read Transfer byte onto bus
RowClone: In-DRAM Row Copy (and Initialization)

- Step 1: Activate row A
- Step 2: Activate row B

0.01% area cost
RowClone: Latency and Energy Savings

End-to-End System Design

- Application
- Operating System
- ISA
- Microarchitecture
- DRAM (RowClone)

How does the software communicate occurrences of bulk copy-initialization to hardware?

How to ensure cache coherence?

How to maximize latency and energy savings?

How to handle data reuse?
RowClone: Overall Performance

- IPC Improvement
- Energy Reduction

% Compared to Baseline

- bootup
- compile
- forkbench
- mcached
- mysql
- shell
RowClone: Multi-Core Performance

Normalized Weighted Speedup

Baseline
RowClone

50 Workloads (4-core)
Goal: Ultra-Efficient Processing Close to Data

Goal: Memory similar to a “conventional” accelerator

Slide credit: Prof. Kayvon Fatahalian, CMU
Enabling Ultra-Efficient Search

- What is the right partitioning of computation capability?
- What is the right low-cost memory substrate?
- What memory technologies are the best enablers?
- How do we rethink/ease (visual) search?

Picture credit: Prof. Kayvon Fatahalian, CMU
Tolerating DRAM: Example Techniques

- Retention-Aware DRAM Refresh: Reducing Refresh Impact
- Refresh Access Parallelization: Reducing Refresh Impact
- Tiered-Latency DRAM: Reducing DRAM Latency
- RowClone: Accelerating Page Copy and Initialization
- Subarray-Level Parallelism: Reducing Bank Conflict Impact
- Linearly Compressed Pages: Efficient Memory Compression
Agenda

- Major Trends Affecting Main Memory
- The Memory Scaling Problem and Solution Directions
  - New Memory Architectures
  - Enabling Emerging Technologies: Hybrid Memory Systems
- How Can We Do Better?
- Summary
Solution 2: Emerging Memory Technologies

- Some emerging resistive memory technologies seem more scalable than DRAM (and they are non-volatile)

- Example: Phase Change Memory
  - Data stored by changing phase of material
  - Data read by detecting material’s resistance
  - Expected to scale to 9nm (2022 [ITRS])
  - Prototyped at 20nm (Raoux+, IBM JRD 2008)
  - Expected to be denser than DRAM: can store multiple bits/cell

- But, emerging technologies have (many) shortcomings
  - Can they be enabled to replace/augment/surpass DRAM?
Phase Change Memory: Pros and Cons

- Pros over DRAM
  - Better technology scaling (capacity and cost)
  - Non volatility
  - Low idle power (no refresh)

- Cons
  - Higher latencies: ~4-15x DRAM (especially write)
  - Higher active energy: ~2-50x DRAM (especially write)
  - Lower endurance (a cell dies after ~10^8 writes)

- Challenges in enabling PCM as DRAM replacement/helper:
  - Mitigate PCM shortcomings
  - Find the right way to place PCM in the system
PCM-based Main Memory (I)

- How should PCM-based (main) memory be organized?

- Hybrid PCM+DRAM [Qureshi+ ISCA’09, Dhiman+ DAC’09]:
  - How to partition/migrate data between PCM and DRAM
PCM-based Main Memory (II)

- How should PCM-based (main) memory be organized?

- Pure PCM main memory [Lee et al., ISCA’09, Top Picks’10]:
  - How to redesign entire hierarchy (and cores) to overcome PCM shortcomings
An Initial Study: Replace DRAM with PCM

  - Surveyed prototypes from 2003-2008 (e.g. IEDM, VLSI, ISSCC)
  - Derived “average” PCM parameters for F=90nm

**Density**
- $9 - 12F^2$ using BJT
- $1.5 \times$ DRAM

**Latency**
- 50ns Rd, 150ns Wr
- $4 \times$, $12 \times$ DRAM

**Endurance**
- $1 \times 10^8$ writes
- $1 \times 10^{-8} \times$ DRAM

**Energy**
- $40 \mu A$ Rd, $150 \mu A$ Wr
- $2 \times$, $43 \times$ DRAM
Results: Naïve Replacement of DRAM with PCM

- Replace DRAM with PCM in a 4-core, 4MB L2 system
- PCM organized the same as DRAM: row buffers, banks, peripherals
- 1.6x delay, 2.2x energy, 500-hour average lifetime

Architecting PCM to Mitigate Shortcomings

- **Idea 1:** Use multiple narrow row buffers in each PCM chip → Reduces array reads/writes → better endurance, latency, energy

- **Idea 2:** Write into array at cache block or word granularity → Reduces unnecessary wear
Results: Architected PCM as Main Memory

- 1.2x delay, 1.0x energy, 5.6-year average lifetime
- Scaling improves energy, endurance, density

- Caveat 1: Worst-case lifetime is much shorter (no guarantees)
- Caveat 2: Intensive applications see large performance and energy hits
- Caveat 3: Optimistic PCM parameters?
Hybrid Memory Systems

Hardware/software manage data allocation and movement to achieve the best of multiple technologies

Yoon, Meza et al., “Row Buffer Locality Aware Caching Policies for Hybrid Memories,” ICCD 2012 Best Paper Award.
One Option: DRAM as a Cache for PCM

- PCM is main memory; DRAM caches memory rows/blocks
  - Benefits: Reduced latency on DRAM cache hit; write filtering
- Memory controller hardware manages the DRAM cache
  - Benefit: Eliminates system software overhead

Three issues:
- What data should be placed in DRAM versus kept in PCM?
- What is the granularity of data movement?
- How to design a huge (DRAM) cache at low cost?

Two solutions:
- Locality-aware data placement [Yoon+, ICCD 2012]
- Cheap tag stores and dynamic granularity [Meza+, IEEE CAL 2012]
DRAM vs. PCM: An Observation

- Row buffers are the same in DRAM and PCM
- Row buffer hit latency **same** in DRAM and PCM
- Row buffer miss latency **small** in DRAM, **large** in PCM

- Accessing the row buffer in PCM is fast
- What incurs high latency is the PCM array access → avoid this
Row-Locality-Aware Data Placement

- **Idea:** Cache in DRAM only those rows that
  - Frequently cause row buffer conflicts → because row-conflict latency is smaller in DRAM
  - Are reused many times → to reduce cache pollution and bandwidth waste

- **Simplified rule of thumb:**
  - Streaming accesses: Better to place in PCM
  - Other accesses (with some reuse): Better to place in DRAM

Row-Locality-Aware Data Placement: Results

Memory energy-efficiency and fairness also improve correspondingly
Hybrid vs. All-PCM/DRAM

31% better performance than all PCM, within 29% of all DRAM performance
Aside: STT-MRAM as Main Memory

- Magnetic Tunnel Junction (MTJ)
  - Reference layer: Fixed
  - Free layer: Parallel or anti-parallel

- Cell
  - Access transistor, bit/sense lines

- Read and Write
  - Read: Apply a small voltage across bitline and senseline; read the current.
  - Write: Push large current through MTJ. Direction of current determines new orientation of the free layer.

Aside: STT-MRAM: Pros and Cons

- Pros over DRAM
  - Better technology scaling
  - Non volatility
  - Low idle power (no refresh)

- Cons
  - Higher write latency
  - Higher write energy
  - Reliability?

- Another level of freedom
  - Can trade off non-volatility for lower write latency/energy (by reducing the size of the MTJ)
Architected STT-MRAM as Main Memory

- 4-core, 4GB main memory, multiprogrammed workloads
- ~6% performance loss, ~60% energy savings vs. DRAM

Agenda

- Major Trends Affecting Main Memory
- The Memory Scaling Problem and Solution Directions
  - New Memory Architectures
  - Enabling Emerging Technologies: Hybrid Memory Systems
- How Can We Do Better?
- Summary
Principles (So Far)

- Better cooperation between devices and the system
  - Expose more information about devices to upper layers
  - More flexible interfaces

- Better-than-worst-case design
  - Do not optimize for the worst case
  - Worst case should not determine the common case

- Heterogeneity in design
  - Enables a more efficient design (No one size fits all)
Other Opportunities with Emerging Technologies

- **Merging of memory and storage**
  - e.g., a single interface to manage all data

- **New applications**
  - e.g., ultra-fast checkpoint and restore

- **More robust system design**
  - e.g., reducing data loss

- **Processing tightly-coupled with memory**
  - e.g., enabling efficient search and filtering
The traditional two-level storage model is a bottleneck with NVM

- **Volatile** data in memory → a **load/store** interface
- **Persistent** data in storage → a **file system** interface
- Problem: Operating system (OS) and file system (FS) code to locate, translate, buffer data become performance and energy bottlenecks with fast NVM stores
Coordinated Memory and Storage with NVM (II)

- **Goal:** Unify memory and storage management in a single unit to eliminate wasted work to locate, transfer, and translate data
  - Improves both energy and performance
  - Simplifies programming model as well

![Unified Memory/Storage Diagram]

**SAFARI**
The Persistent Memory Manager (PMM)

- Exposes a load/store interface to access persistent data
  - Applications can directly access persistent memory → no conversion, translation, location overhead for persistent data

- Manages data placement, location, persistence, security
  - To get the best of multiple forms of storage

- Manages metadata storage and retrieval
  - This can lead to overheads that need to be managed

- Exposes hooks and interfaces for system software
  - To enable better data placement and management decisions

The Persistent Memory Manager (PMM)

PMM uses access and hint information to allocate, locate, migrate and access data in the heterogeneous array of devices.

```c
int main(void) {
    // data in file.dat is persistent
    FILE myData = "file.dat";
    myData = new int[64];
}

void updateValue(int n, int value) {
    FILE myData = "file.dat";
    myData[n] = value; // value is persistent
}
```

Load \[\rightarrow\] Store \[\downarrow\] Hints from SW/OS/runtime

Software Persistent Memory Manager Hardware

Data Layout, Persistence, Metadata, Security, ...

DRAM \[\uparrow\] Flash \[\uparrow\] NVM \[\uparrow\] HDD
Performance Benefits of a Single-Level Store

Results for PostMark

- HDD 2-level
- NVM 2-level
- Persistent Memory

Normalized Execution Time

- User CPU
- User Memory
- Syscall CPU
- Syscall I/O

- ~24X
- ~5X

SAFARI

85
Energy Benefits of a Single-Level Store

Results for PostMark

- HDD 2-level: ~16X
- NVM 2-level: ~5X
- Persistent Memory: 0.013

SAFARI
Agenda

- Major Trends Affecting Main Memory
- The Memory Scaling Problem and Solution Directions
  - New Memory Architectures
  - Enabling Emerging Technologies: Hybrid Memory Systems
- How Can We Do Better?
- Summary
Summary: Memory/Storage Scaling

- Memory/storage scaling problems are a critical bottleneck for system performance, efficiency, and usability

- New memory/storage + compute architectures
  - Rethinking DRAM; processing close to data; accelerating bulk operations

- Enabling emerging NVM technologies
  - Hybrid memory systems with automatic data management
  - Coordinated management of memory and storage with NVM

- System-level memory/storage QoS

- Three principles are essential for scaling
  - Software/hardware/device cooperation
  - Better-than-worst-case design
  - Heterogeneity (specialization, asymmetry)
These slides are a shortened and revised version of the *Scalable Memory Systems* course at ACACES 2013.

**Website for Course Slides, Papers, and Videos**
- [http://users.ece.cmu.edu/~omutlu/acaces2013-memory.html](http://users.ece.cmu.edu/~omutlu/acaces2013-memory.html)
- [http://users.ece.cmu.edu/~omutlu/projects.htm](http://users.ece.cmu.edu/~omutlu/projects.htm)
- Includes extended lecture notes and readings

**Overview Reading**
- Onur Mutlu, *Memory Scaling: A Systems Architecture Perspective*  
  Slides (pptx) (pdf)
Thank you.

Feel free to email me with any questions & feedback

onur@cmu.edu
Rethinking Memory System Design for Data-Intensive Computing

Onur Mutlu
onur@cmu.edu
June 20, 2014
ASAP 2014, Zurich

Carnegie Mellon
Another Talk: NAND Flash Scaling Challenges