STORAGE DEVELOPER CONFERENCE

SNIA ■ SANTA CLARA, 2015

Solving the Challenges of Persistent Memory Programming

Sarah Jelinek
Intel Corporation

Disclaimer

By using this document, in addition to any agreements you have with Intel, you accept the terms set forth below.

You may not use or facilitate the use of this document in connection with any infringement or other legal analysis concerning Intel products described herein. You agree to grant Intel a non-exclusive, royalty-free license to any patent claim thereafter drafted which includes subject matter disclosed herein.

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

A "Mission Critical Application" is any application in which failure of the Intel Product could result, directly or indirectly, in personal injury or death. SHOULD YOU PURCHASE OR USE INTEL'S PRODUCTS FOR ANY SUCH MISSION CRITICAL APPLICATION, YOU SHALL INDEMNIFY AND HOLD INTEL AND ITS SUBSIDIARIES, SUBCONTRACTORS AND AFFILIATES, AND THE DIRECTORS, OFFICERS, AND EMPLOYEES OF EACH, HARMLESS AGAINST ALL CLAIMS COSTS, DAMAGES, AND EXPENSES AND REASONABLE ATTORNEYS' FEES ARISING OUT OF, DIRECTLY OR INDIRECTLY, ANY CLAIM OF PRODUCT LIABILITY, PERSONAL INJURY, OR DEATH ARISING IN ANY WAY OUT OF SUCH MISSION CRITICAL APPLICATION, WHETHER OR NOT INTEL OR ITS SUBCONTRACTOR WAS NEGLIGENT IN THE DESIGN, MANUFACTURE, OR WARNING OF THE INTEL PRODUCT OR ANY OF ITS PARTS.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined". Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

This document contains information on products in the design phase of development. The information here is subject to change without notice. Do not finalize a design with this information.

Results have been estimated based on internal Intel analysis and are provided for informational purposes only. Any difference in system hardware or software design or configuration may affect actual performance.

Intel, the Intel logo, are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States or other countries.

Copyright © 2015 Intel Corporation. All rights reserved

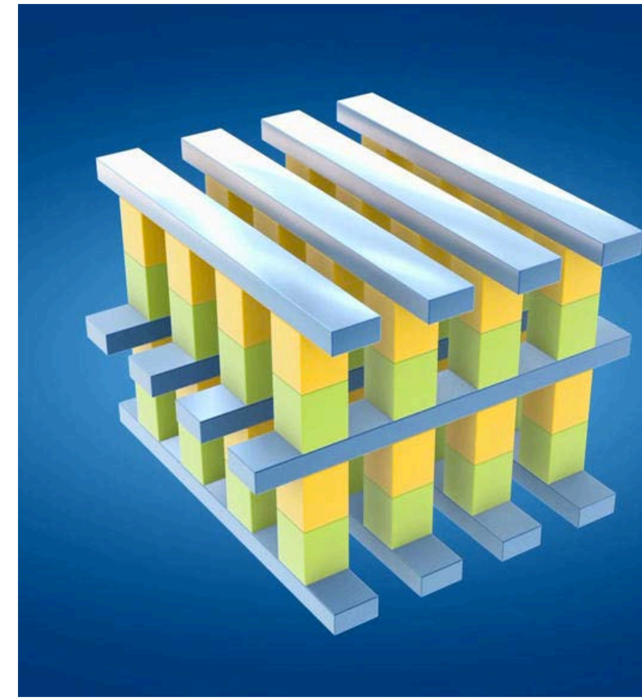
*Other brands and names may be claimed as the property of others.

Agenda

- What is Persistent Memory
  - Visibility vs. Persistence
- Overview of Persistent Memory and the Non-Volatile Memory Library
- Context of current work
- Challenges with integration of volatile mode persistent memory
- Challenges with integration of persistent mode persistent memory

2015 Storage Developer Conference. © Insert Your Company Name. All Rights Reserved.

What Is Persistent Memory?



- Byte-addressable memory, but persistent
- Must be *reasonable* to stall a CPU waiting for a load to finish
  - So, not NAND NVM based
- Can do small I/O
  - DIMMs are 64B cache line accessible
- Can DMA to it
  - Receive data from network directly to persistence!
- At IDF Intel said capacity will be up to 6TB on a two-socket server

2015 Storage Developer Conference. © Insert Your Company Name. All Rights Reserved.

Visibility vs. Persistence

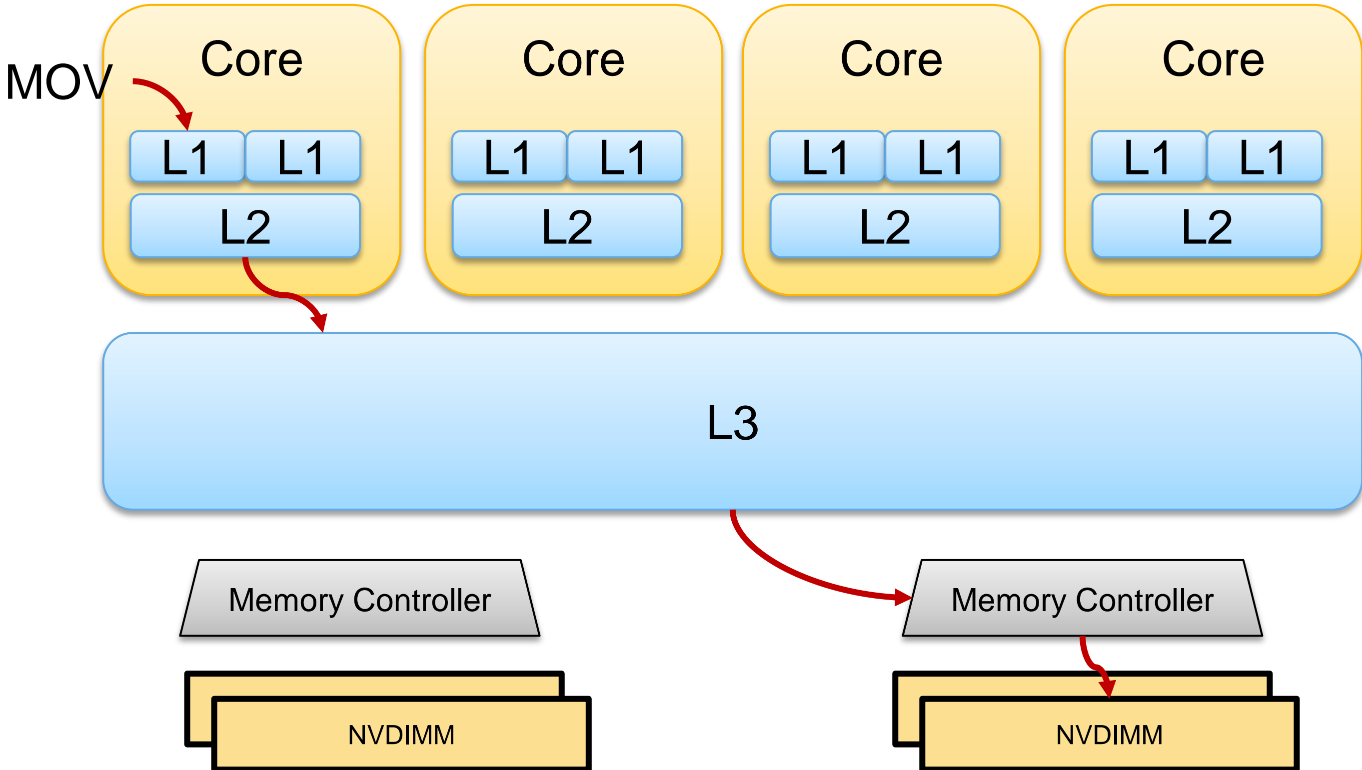
- It has always been thus:
  - open()
  - mmap()
  - store...

  - msync()

- pmem just follows this decades-old model
  - But the stores are cached in a different spot

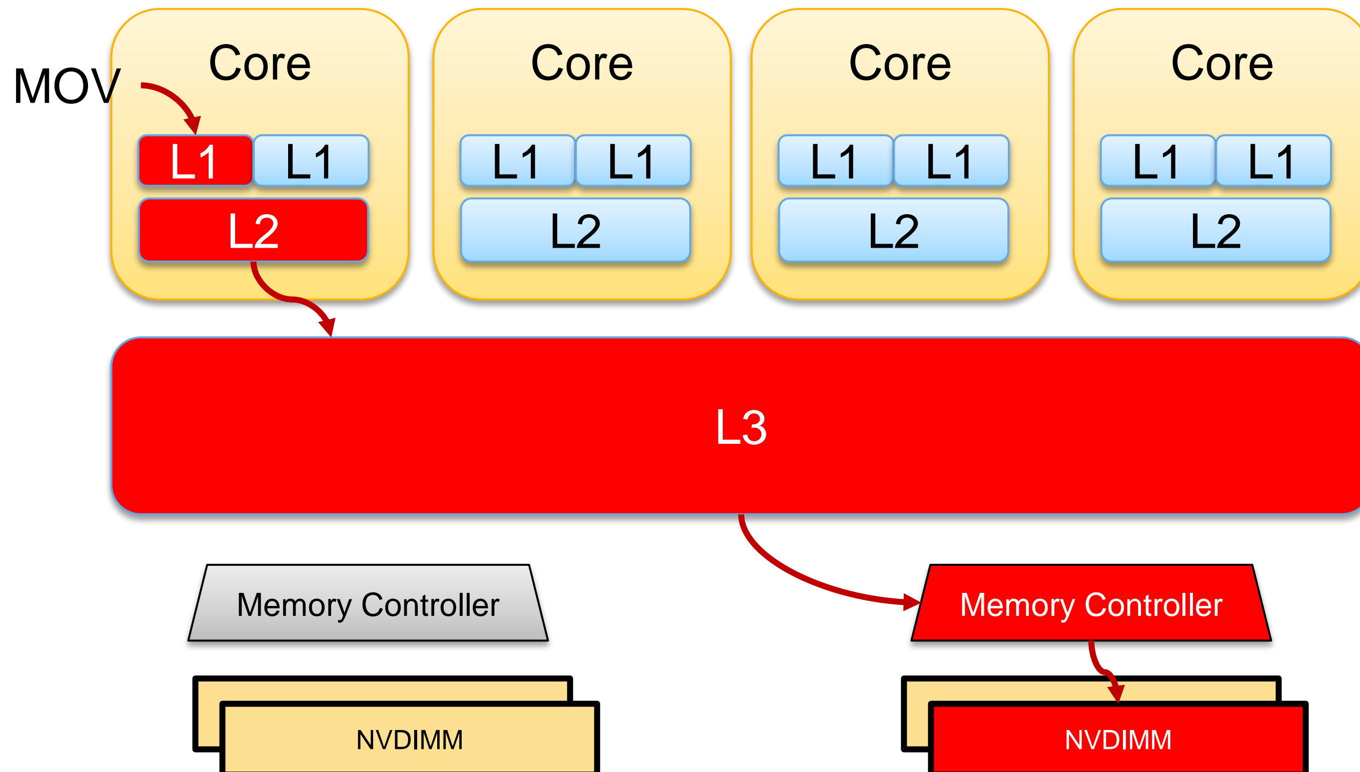
2015 Storage Developer Conference. © Insert Your Company Name. All Rights Reserved.

The Data Path



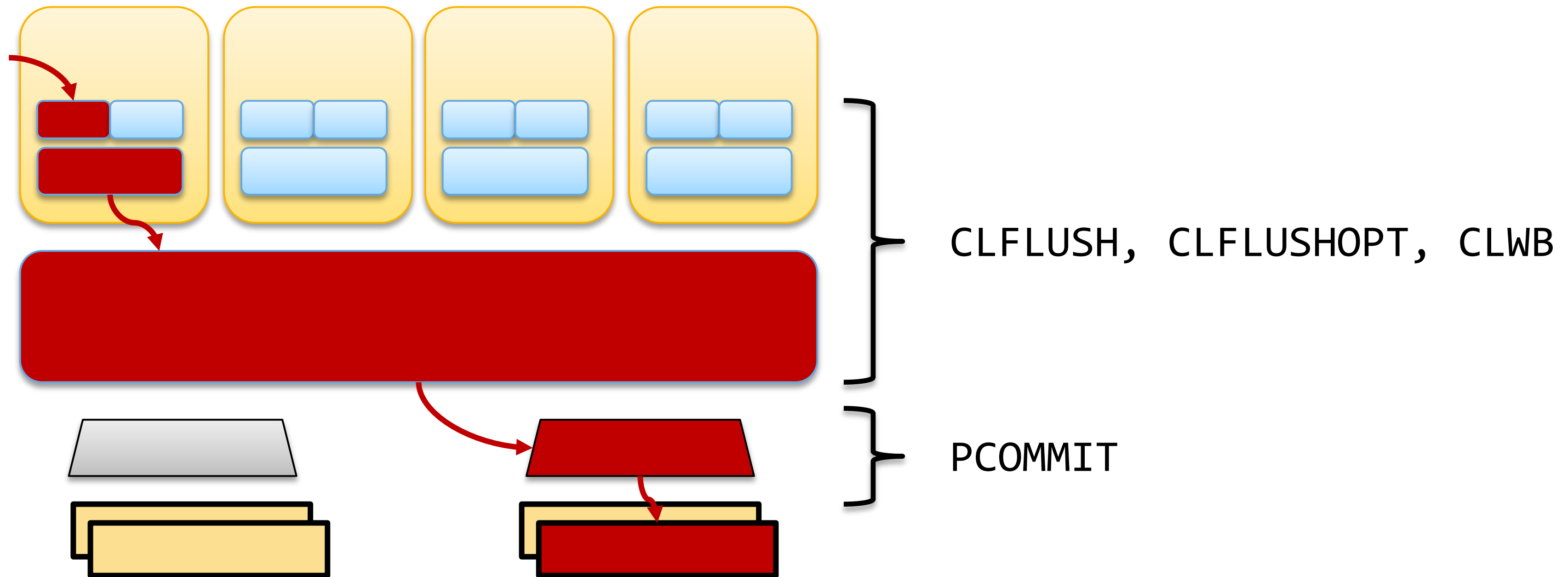
2015 Storage Developer Conference. © Insert Your Company Name. All Rights Reserved.

Hiding Places



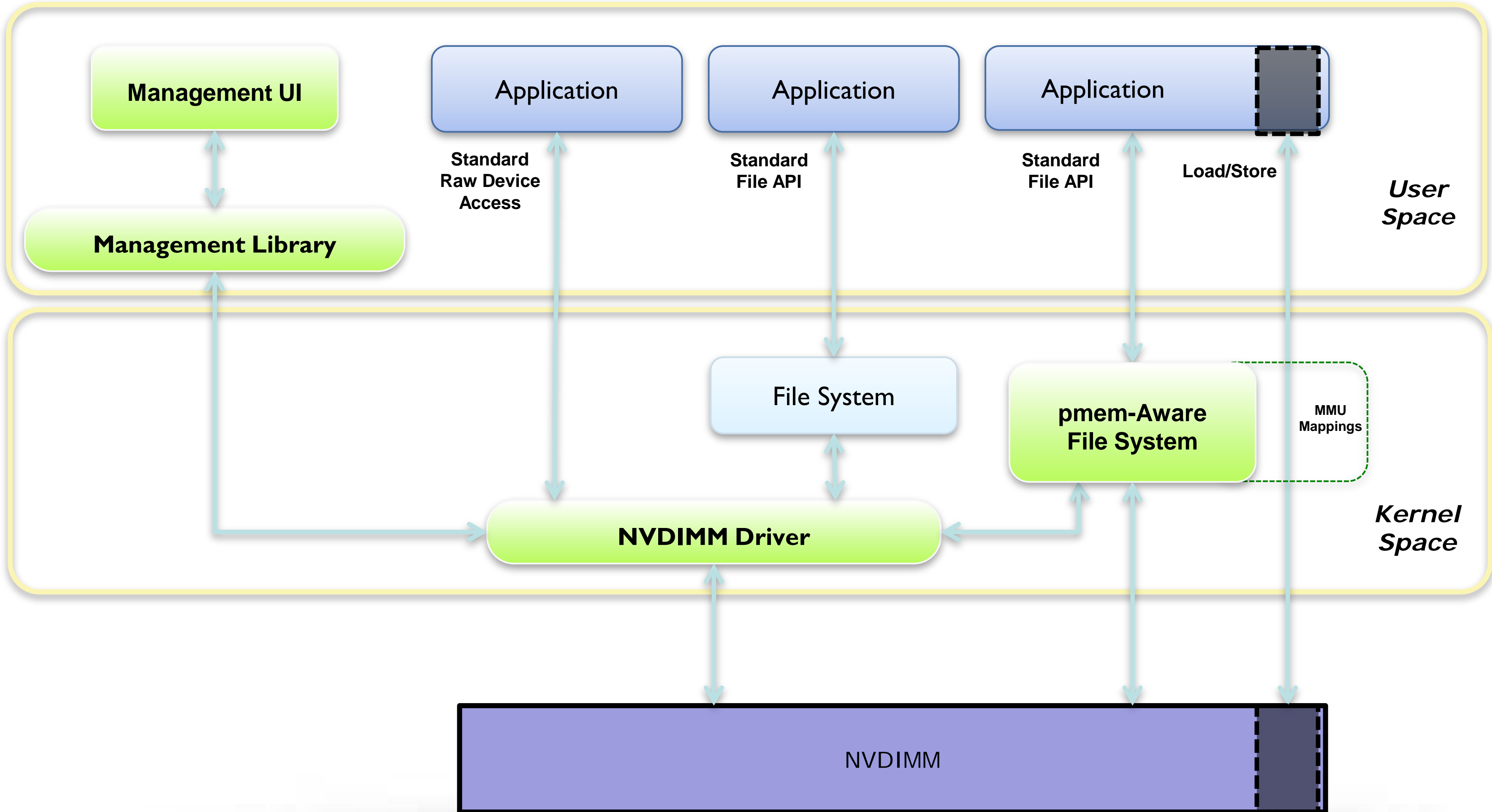
2015 Storage Developer Conference. © Insert Your Company Name. All Rights Reserved.

Two Levels of Flushing Writes(what we can do to ensure persistence)



2015 Storage Developer Conference. © Insert Your Company Name. All Rights Reserved.

Persistent Memory Software Architecture



SDC 15

2015 Storage Developer Conference. © Insert Your Company Name. All Rights Reserved.

NVML Open Source Library

- 6 libraries comprise overall NVML project
  - libpmem:
    - Low level persistent memory support.
  - libpmemblk
    - Supports arrays of pmem-resident blocks, all the same size, that are atomically updated. E.g. cache of fixed sized objects
  - libpmemlog
    - Provides a pmem-resident log file.

SDC 15

2015 Storage Developer Conference. © Insert Your Company Name. All Rights Reserved.

NVML Open Source Library

- NVML libraries cont.
  - libpmemobj
    - Provides transactional object store. Includes memory allocation. Best place to start.
  - libvmem
    - Turns pool of pmem into volatile memory pool.
  - libvmmalloc
    - Transparently converts all memory allocations into persistent memory allocations. No modification of target application required.

More info @pmem.io

2015 Storage Developer Conference. © Insert Your Company Name. All Rights Reserved.

My Current Work

- Integration of persistent memory into an existing block cache
- Goals:
  - Reduce DRAM required
  - Provide warm cache in persistent mode
  - Be within 10-20% performance of DRAM for NVM cache
  - Integration of both volatile and persistent modes

2015 Storage Developer Conference. © Insert Your Company Name. All Rights Reserved.

Persistent Memory Integration Challenges

- Where to integrate Persistent Memory?
- Possible memory management changes
- What to include in Persistent Memory
- Seamless integration of Persistent Memory in existing application
- For “persistent” mode it’s particularly challenging

2015 Storage Developer Conference. © Insert Your Company Name. All Rights Reserved.

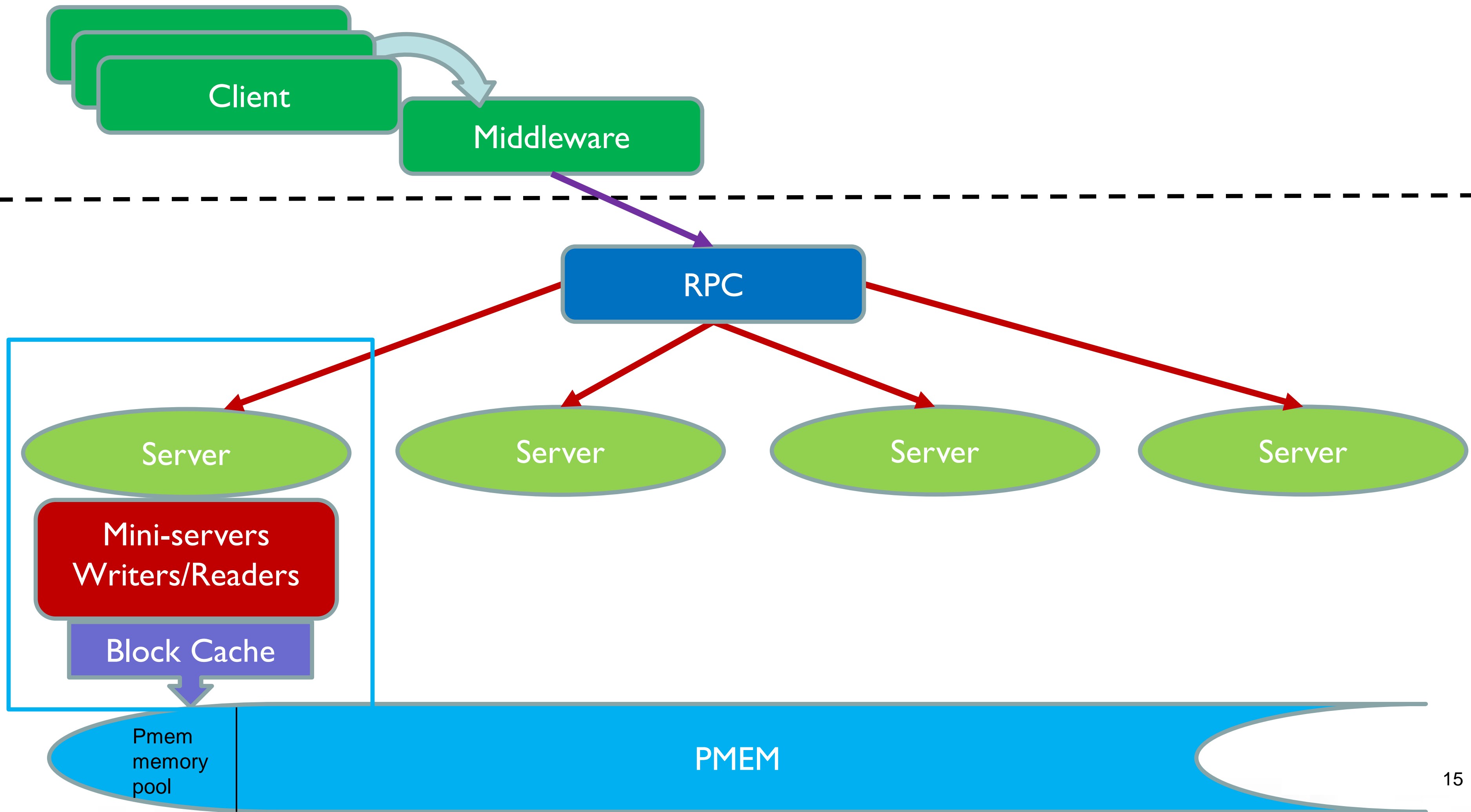
Persistent Memory, Where to Integrate?

- Where will you get the best use and performance?
- Can you keep track of the memory allocated without a lot of additional code?
- How easily does it integrate into existing code?
- Least intrusive

SDC 15

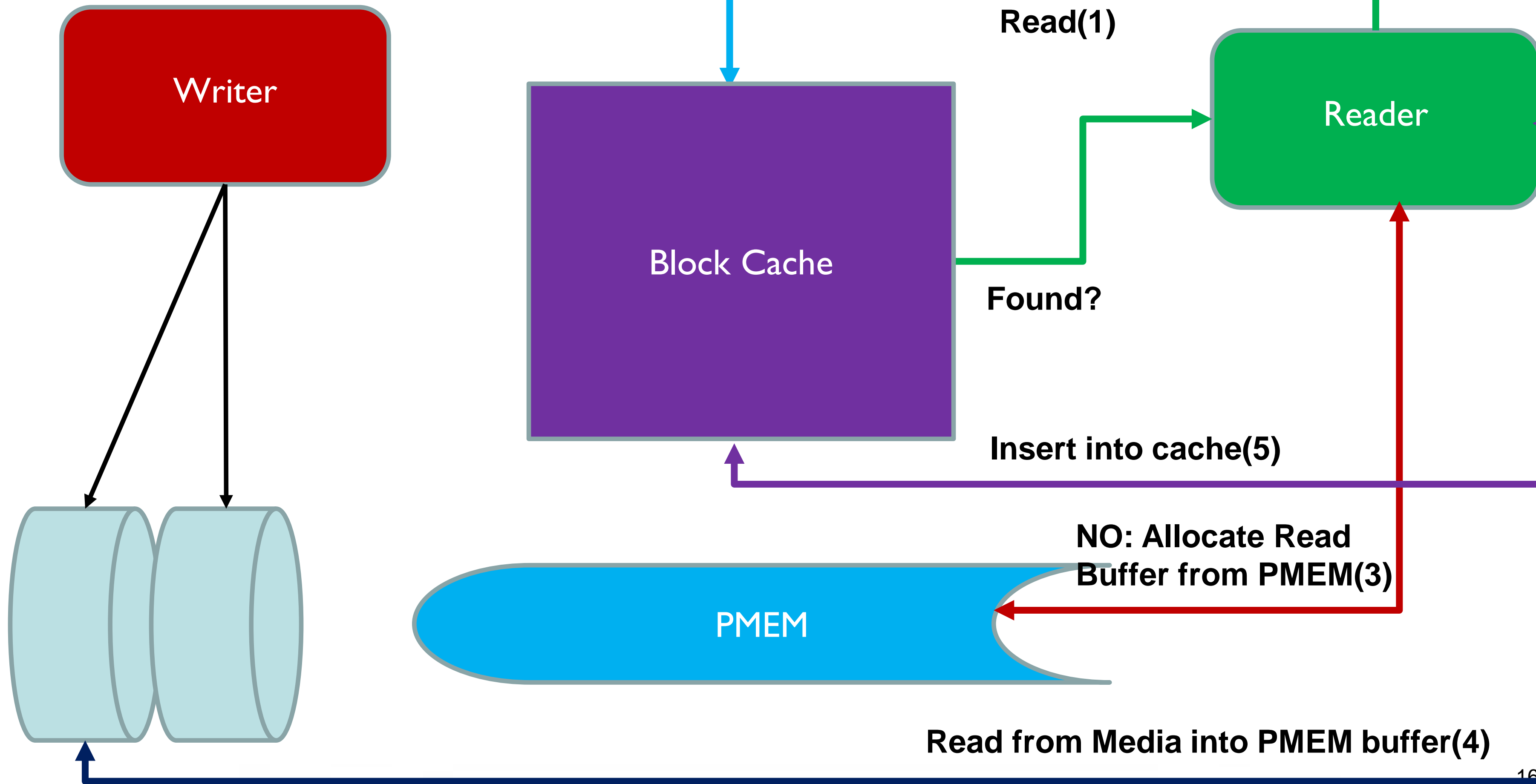
2015 Storage Developer Conference. © Insert Your Company Name. All Rights Reserved.

Block Cache Design



SDC 15
2015 Storage Developer Conference. © Insert Your Company Name. All Rights Reserved.

Server/Block Cache Design



SDC 15
2015 Storage Developer Conference. © Insert Your Company Name. All Rights Reserved.

Key Points In Block Cache Design

- 1 block cache per server
- When PMEM is integrated cache will now be divided between DRAM and PMEM(volatile or persistent mode)
- 2 stage persistent memory allocation
  - Speeds up reads/reduces double copy
  - But, complicates memory management
- Key data structures:
  - Containing structure that holds key/value among other metadata
  - Hash table for lists of containing structures

SDC 15

2015 Storage Developer Conference. © Insert Your Company Name. All Rights Reserved.

Volatile Mode PMEM Integration

- In general do not have to worry about how you integrate volatile mode
  - The goal is to reduce DRAM, so store at least cache data(which could be very large) in PMEM. Account for PMEM correctly.
  - No worry about maintaining persistence.
    - Except that the combination of DRAM and PMEM data structures must still be consistent.
  - PMEM space is limited. There is no paging so when you cannot allocate from a PMEM pool you have to do something to manage this.

SDC 15

2015 Storage Developer Conference. © Insert Your Company Name. All Rights Reserved.

Volatile Mode PMEM Integration

- Store Key, Value and Containing structure in PMEM.
- Some DRAM structures point to PMEM
  - It is **not** ok to have PMEM point to DRAM.
- Used libvmem
- When doing any buffer allocation we evict and retry for *N* times.
- Memory functions:
  - vmem_malloc, vmem_free
  - Once allocated memcpy, memset, memmove are used to modify buffer

2015 Storage Developer Conference. © Insert Your Company Name. All Rights Reserved.

Volatile Mode PMEM Integration

- Critical point is that when you are freeing memory you must call correct *'free'* function since data structures are a mix of DRAM and PMEM.
- Isolated DRAM from PMEM as much as possible.

SDC 15 2015 Storage Developer Conference. © Insert Your Company Name. All Rights Reserved.

Persistent Mode PMEM Integration

- Choosing which method to use
  - Transactions(TX)
  - Atomic LISTS
  - Atomic Memory management
- What works best in implementation
  - I chose atomic memory management based on three factors:
    - Transactional model is complicated for this use case.
    - Performance. Did not want to traverse data structures linearly to find the one we need to finish insertion
    - Atomic memory management fits best in the two stage allocation model that this application requires.

SDC 15

2015 Storage Developer Conference. © Insert Your Company Name. All Rights Reserved.

Persistent Mode PMEM(TX cache allocation)

- TX example – Initial PMEM buffer allocation

```
TX_BEGIN(pop) { // Outer transaction(outer method), allocating from PMEM
    Allocate Memory From Cache {
        TX_BEGIN(pop) { // Inner transaction
            Try allocation; including containing structure
              If fail {
                pmemobj_tx_abort() // Could not allocate from cache. Nothing to do on abort, will revert to
            heap in outer transaction
          }  TX_END(pop) // End of inner transaction
    If no abort called, read data from disk
    Do cache insertion // NEXT page has transaction details for insertion
} TX_ONABORT {// Back to Outer transaction(if abort in inner transaction)
     allocate from heap for read buffer.
     read data from disk
     Return data to caller with indicating cache does not own
```


SDC 15

2015 Storage Developer Conference. © Insert Your Company Name. All Rights Reserved.

Persistent Mode PMEM(TX cache insertion)

- TX example – PMEM cache insertion

```
TX_BEGIN // start a new transaction to do insertion. In outer ReadBlock method.
   Insert into cache {
        TX_BEGIN(pop) { // Inner transaction
              Find data pointer that we are inserting. Requires pointer math. The key_val->value
              member is the 1st member of the structure. Find that address(non linear search)
              If fail to find
                  pmemobj_tx_abort() // Return to outer transaction which aborts
              If found, reallocate struct key_val because we account for key size now.
              Set valid bit
              Call pmem_persist() // No failures possible past this point
              Update DRAM data structures
          }  TX_END(pop) // End of inner transaction

   } TX_ONABORT {// Back to Outer transaction(if abort in inner transaction)
        Clean up previously allocated memory
         return error
   } TX_END
} TX_END(pop)
```


SDC 15

2015 Storage Developer Conference. © Insert Your Company Name. All Rights Reserved.

Persistent Mode PMEM Atomic Memory Allocation

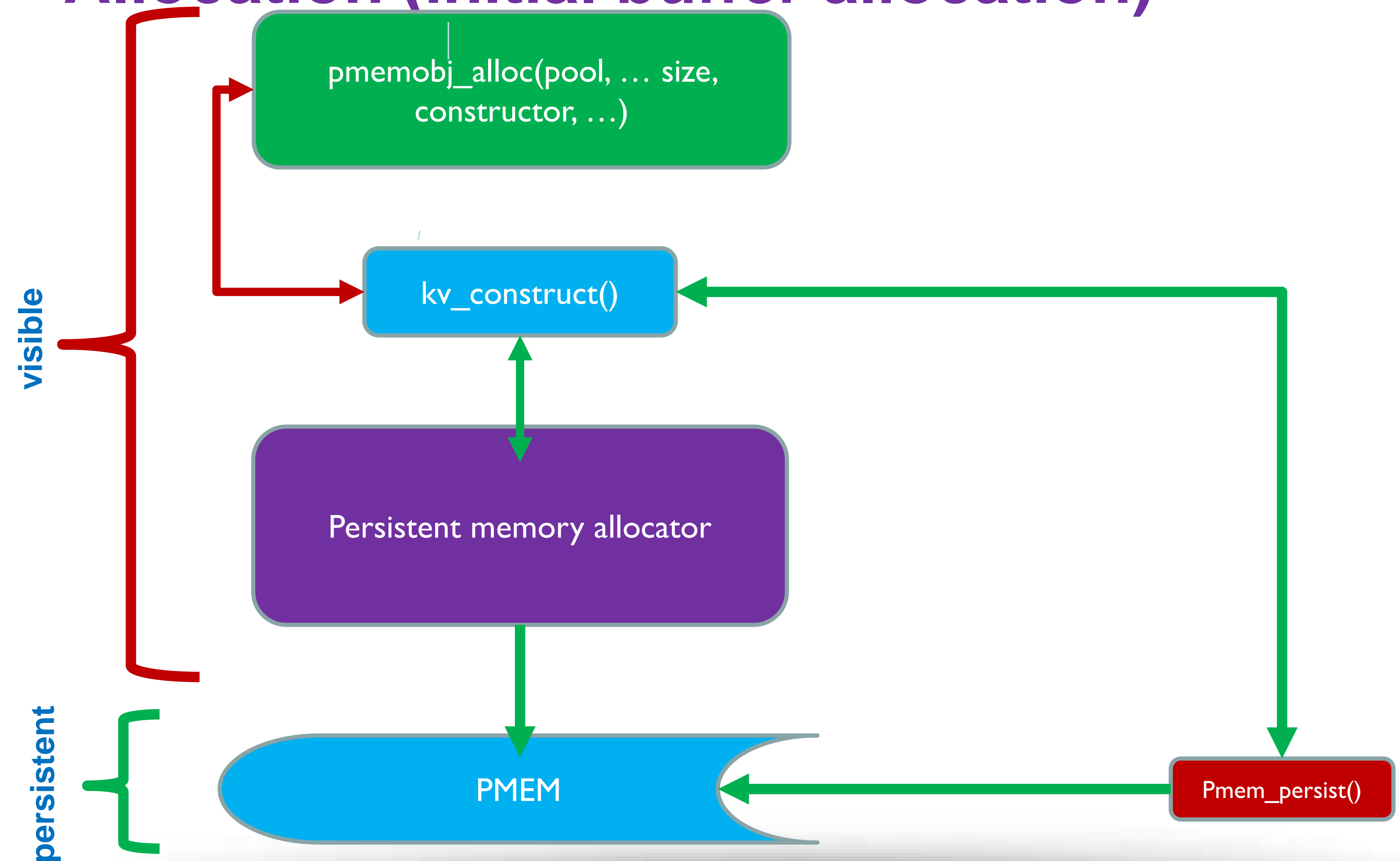
- Initial buffer allocation
- struct key_val is used to track the PMEM key/val addresses. And, it’s also used to find the *‘value’.*

```
struct key_val {
  uint8_t   *value;
  size_t    value_size;
  size_t    key_length;
  uint8_t   key[1];
  size_t    kv_size; // Final size of the allocated structure.
  bool      valid; // Set once all data has been set and inserted into
cache.
};
```


SDC 15

2015 Storage Developer Conference. © Insert Your Company Name. All Rights Reserved.

Persistent Mode PMEM, Atomic Memory Allocation (initial buffer allocation)



2015 Storage Developer Conference. © Insert Your Company Name. All Rights Reserved.

Persistent Mode PMEM, Atomic Memory Allocation(Cache insertion)

- At insertion time we have data value and key value and size.
- Need to reallocate key_val structure to account for key data size
- We must search for value pointer to locate correct key_val structure.

SDC 15

2015 Storage Developer Conference. © Insert Your Company Name. All Rights Reserved.

Persistent Mode PMEM, Atomic Memory Allocation (Cache insertion)

Cache Insertion

PMEMoid newoid; newoid.off = uintptr_t(slice->data()) - uintptr_t(pop_);
newoid.pool_uuid_lo = root_.pool_uuid_lo;

Find existing entry

Red lines are crash points

Calculate new size to include key.
int status = pmemobj_zrealloc(pop_, &newoid, new_size, 0);

Set key length

Set key

Set structure size

Set valid bit

pmemobj_persist

Valid bit set, now valid entry

SDC 15

2015 Storage Developer Conference. © Insert Your Company Name. All Rights Reserved.

Summary and Next Steps

- It's challenging to program to Persistent Memory
- Whether integrating into existing app or writing new app it's critical to consider when something is visible and when it's made persistent
- It's important to consider the state of the data in Persistent Memory at any point a crash can occur
- Using the NVML library makes programming to persistent memory much easier. Consider the possibilities without it(programming directly to the device, load/store)

2015 Storage Developer Conference. © Insert Your Company Name. All Rights Reserved.

Summary and Next Steps

- More information about Persistent Memory and the NVM Library at http://pmem.io
- SNIA NVM Programming TWG
  - http://snia.org/forums/sssi/nvmp
- Linux Pmem Examples:
  - https://github.com/pmem/linux-examples
- Use these tools to convert applications for Intel DIMMs!

2015 Storage Developer Conference. © Insert Your Company Name. All Rights Reserved.

Q & A

SDC 15

2015 Storage Developer Conference. © Insert Your Company Name. All Rights Reserved.

Backup

2015 Storage Developer Conference. © Insert Your Company Name. All Rights Reserved.

Flushing Writes From Caches

Instruction	Meaning
`CLFLUSH addr`	Cache Line Flush: Available for a long time
`CLFLUSHOPT addr`	Optimized Cache Line Flush: New to allow concurrency
`CLWB addr`	Cache Line Write Back: Leave value in cache for performance of next access

SDC 15

2015 Storage Developer Conference. © Insert Your Company Name. All Rights Reserved.

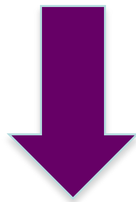
Flushing Writes From Memory Controller

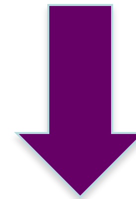
Instruction	Meaning
`PCOMMIT`	Persistent Commit: Flush stores accepted by memory subsystem
`Asynchronous DRAM Refresh`	Flush outstanding writes on power failure Platform-Specific Feature

2015 Storage Developer Conference. © Insert Your Company Name. All Rights Reserved.

Two Movements Afoot

- Why now?
  - Basic programming model is decades old!





Block Mode Innovations

- Atomics
- Access hints
- NVM-oriented operations

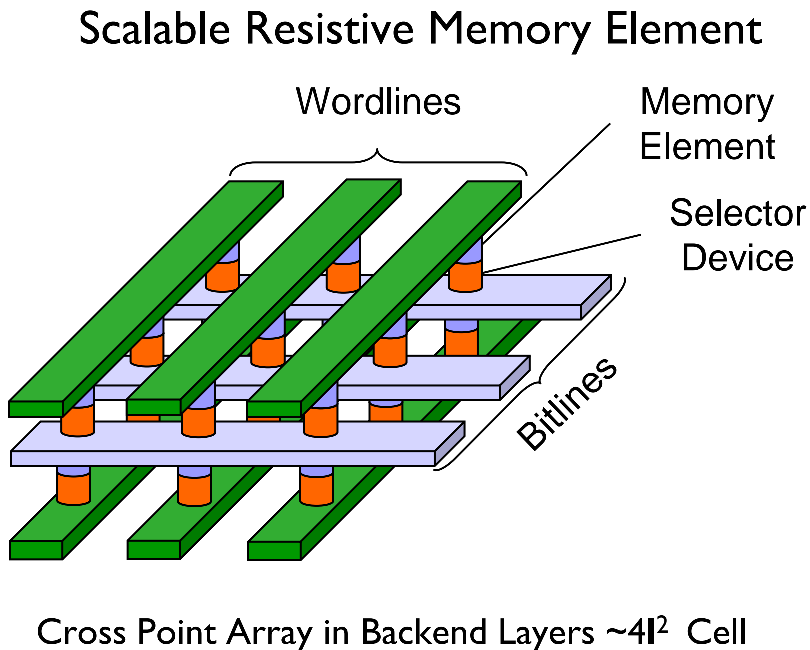
Emerging NVM Technologies

- Performance
- Performance
- Perf… okay, Cost

SDC 15

2015 Storage Developer Conference. © Insert Your Company Name. All Rights Reserved.

Next Generation Scalable NVM



Cross Point Array in Backend Layers ~4l² Cell

Resistive RAM NVM Options

Family	Defining Switching Characteristics
Phase Change Memory	Energy (heat) converts material between crystalline (conductive) and amorphous (resistive) phases
Magnetic Tunnel Junction (MTJ)	Switching of magnetic resistive layer by spin-polarized electrons
Electrochemical Cells (ECM)	Formation / dissolution of "nano-bridge" by electrochemistry
Binary Oxide Filament Cells	Reversible filament formation by Oxidation-Reduction
Interfacial Switching	Oxygen vacancy drift diffusion induced barrier modulation

Many candidate next generation NVM technologies.
Offer ~ 1000x speed-up over NAND, closer to DRAM

2015 Storage Developer Conference. © Insert Your Company Name. All Rights Reserved.