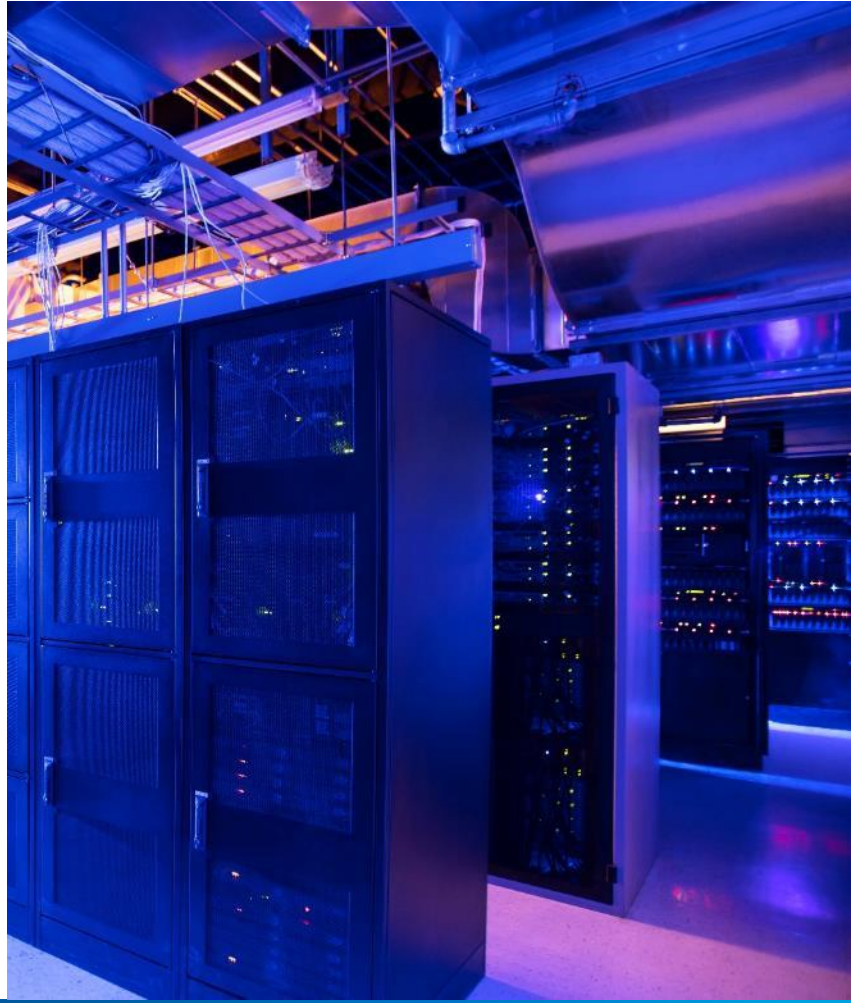
IS YOUR DATA REALLY PERSISTENT? OR DID YOU FORGET TO FLUSH?

Kevin O'Leary
Technical Consulting Engineer

AGENDA

- Persistent memory programming challenges
- Introducing Intel® Inspector – Persistence Inspector
  - Verify your persistent memory applications correctness
    - Works with PMDK



Persistent memory programming challenges

Persistent memory remains intact even after a power outage or system restart. It can be used to create systems that are more reliable and restart faster … but it introduces challenges for software designers.

Data persists only after it is out of the cache hierarchy and written to persistent memory. Processor out-of-order execution and caching also complicate things since the order of persistence may not be the same as the order of store.

What is the Intel® Inspector – Persistence Inspector Tool?

Overview

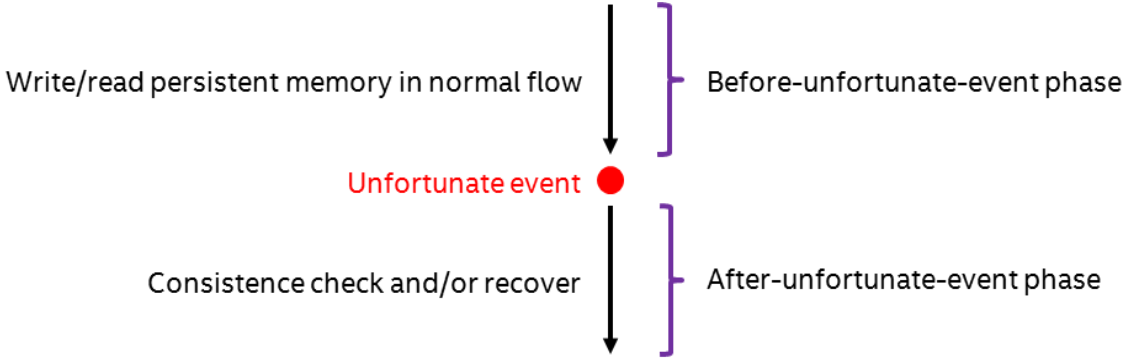
- A new run-time tool developers can use to detect programming errors in persistent memory programs. In addition to cache flush misses, this tool detects
  - Cache flush misses
  - Redundant cache flushes and memory fences
  - Out-of-order persistent memory stores
  - Incorrect undo logging for the Persistent Memory Development Kit (PMDK)

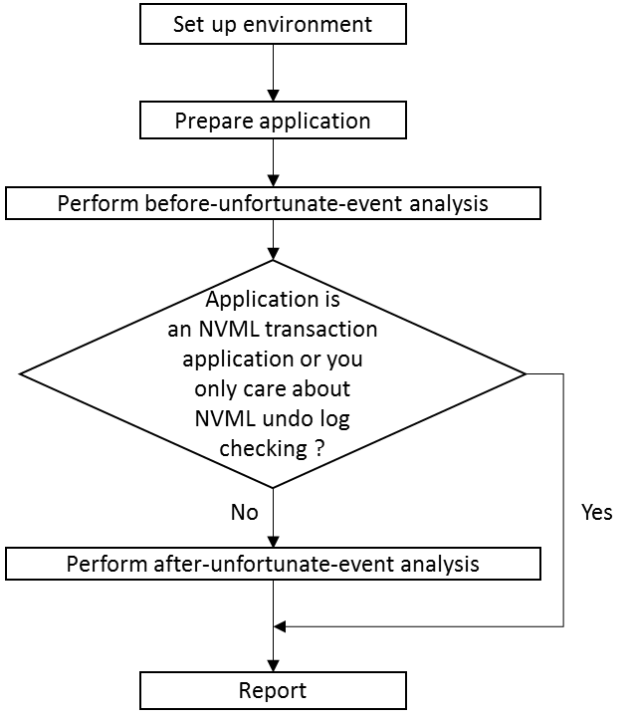
Intel® Inspector – Persistence Inspector workflow

2 Phases

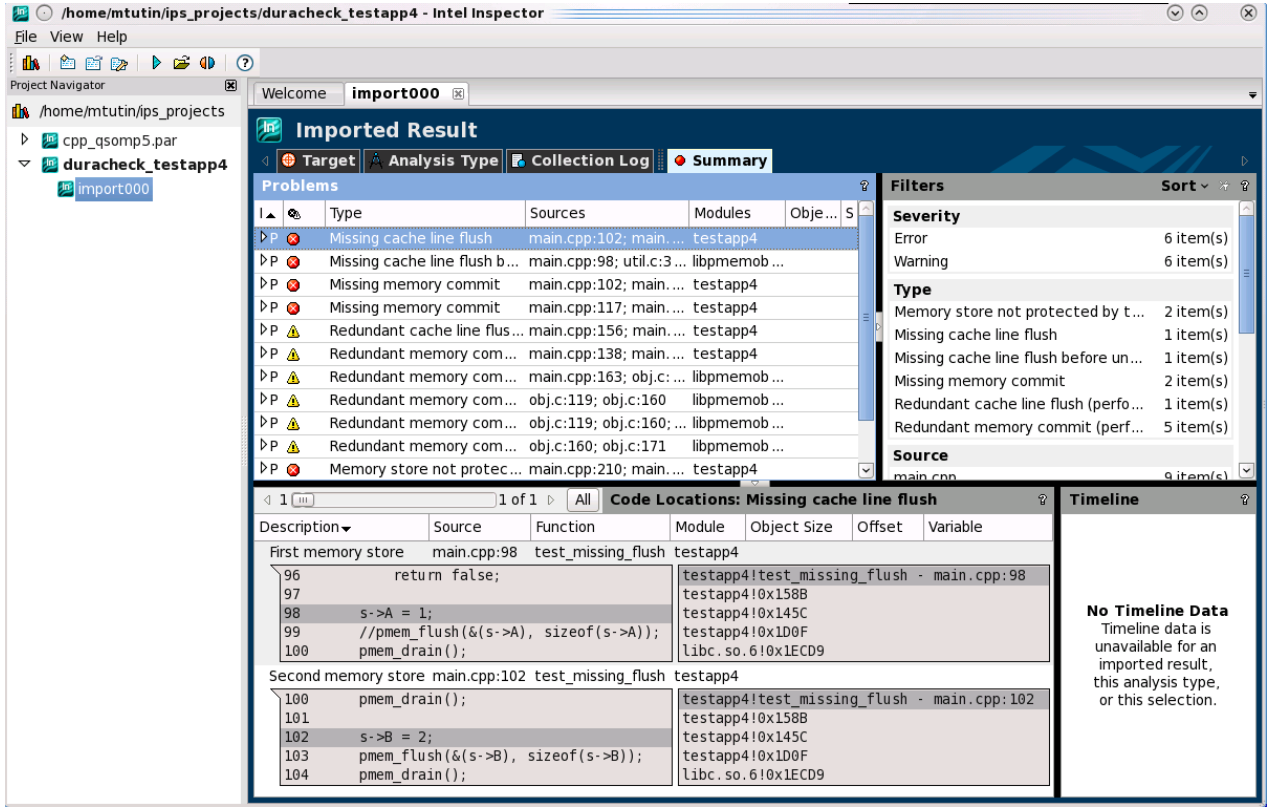
Before Unfortunate event

After Unfortunate event





Inspector report example



Getting the Intel® Inspector – Persistence Inspector tool

- Part of Intel® Parallel Studio 2019
  - Give feedback on the tool!!

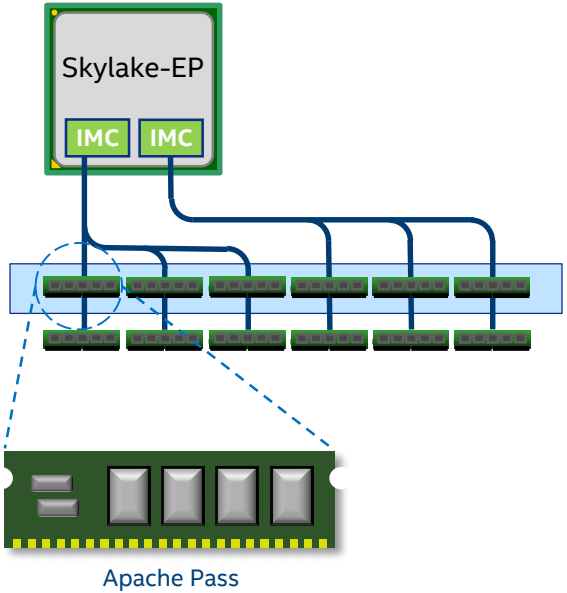
Minimum system requirements:

Processor: Intel architecture processor with clflush/clflushopt/clwb instructions (6thGeneration Intel® processor family code named “Skylake” or newer)

Operating system: Windows* 10 or Linux* (see release notes for details)

HARDWARE SIDE OF DATA PERSISTENCE

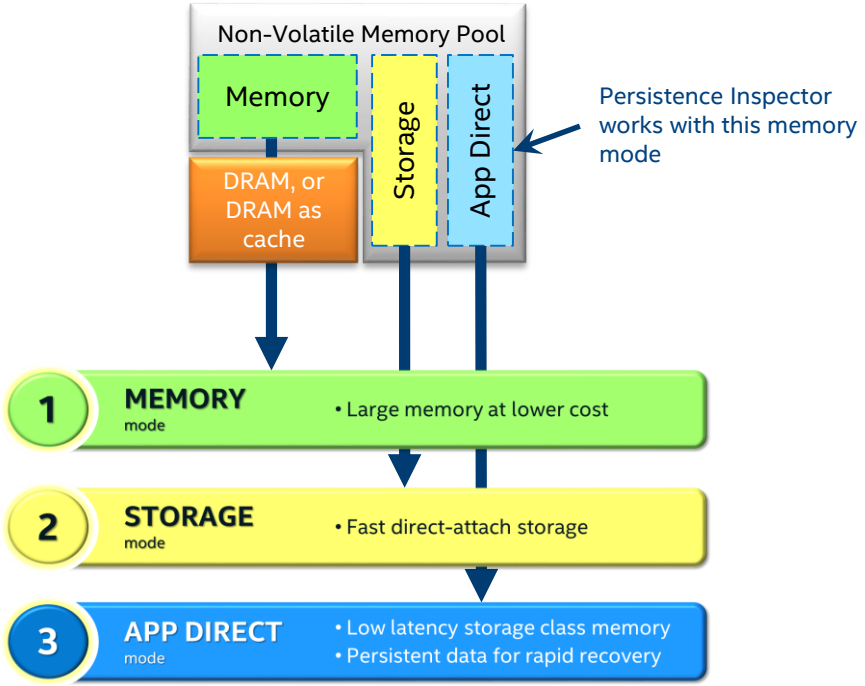
NVM DIMM



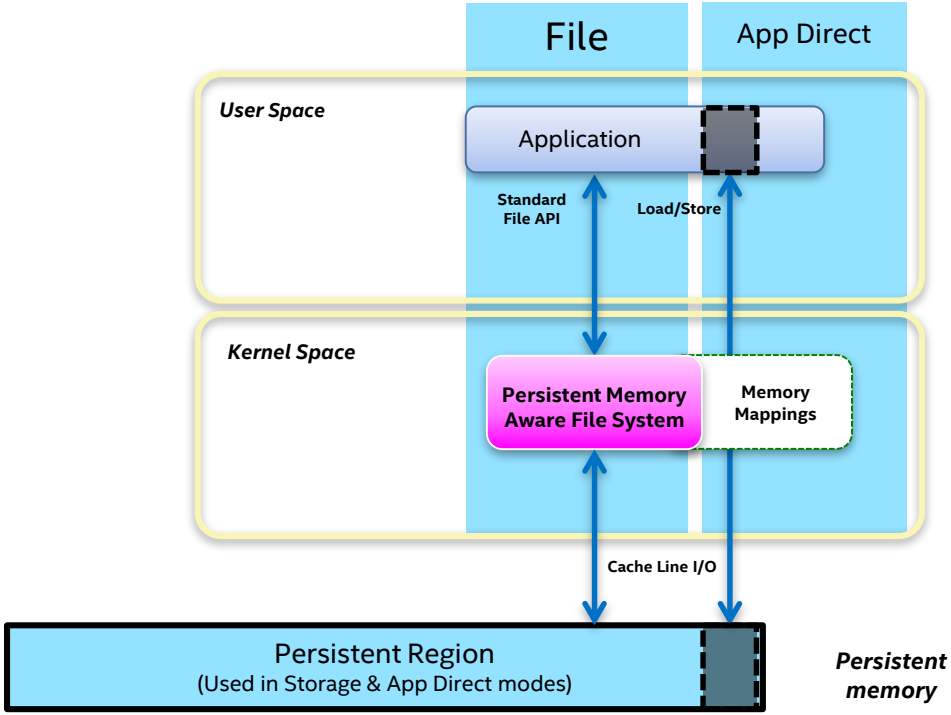
Apache Pass

- DDR4 electrical & physical
- Close to DRAM latency
- Cache line size access

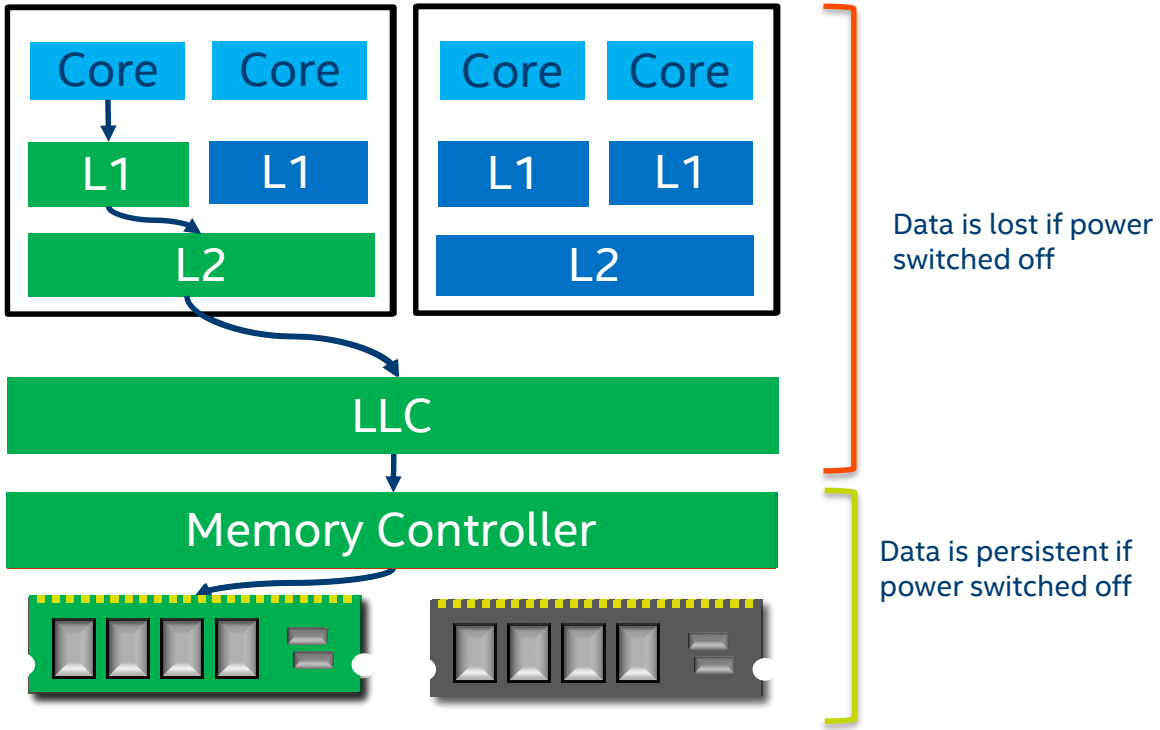
Flexible, Usage Specific Partitions



App Direct Software Architecture



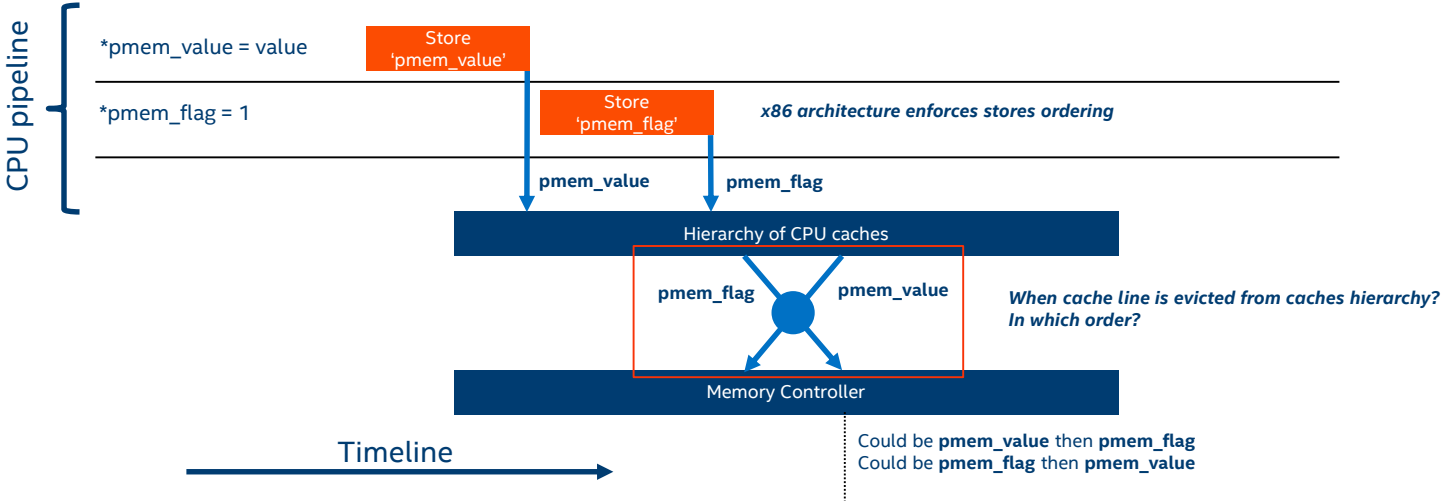
Memory Store != Memory Persistence



CPU cache hierarchy could affect persistence order

```
1  void save_value(uint32_t value, uint8_t* pmem_region)
2  {
3    uint8_t* pmem_flag = pmem_region;
4    uint32_t* pmem_value = (uint32_t*)(pmem_region + 256);
5
6    *pmem_value = value;
7    *pmem_flag = 1;
8  }
```

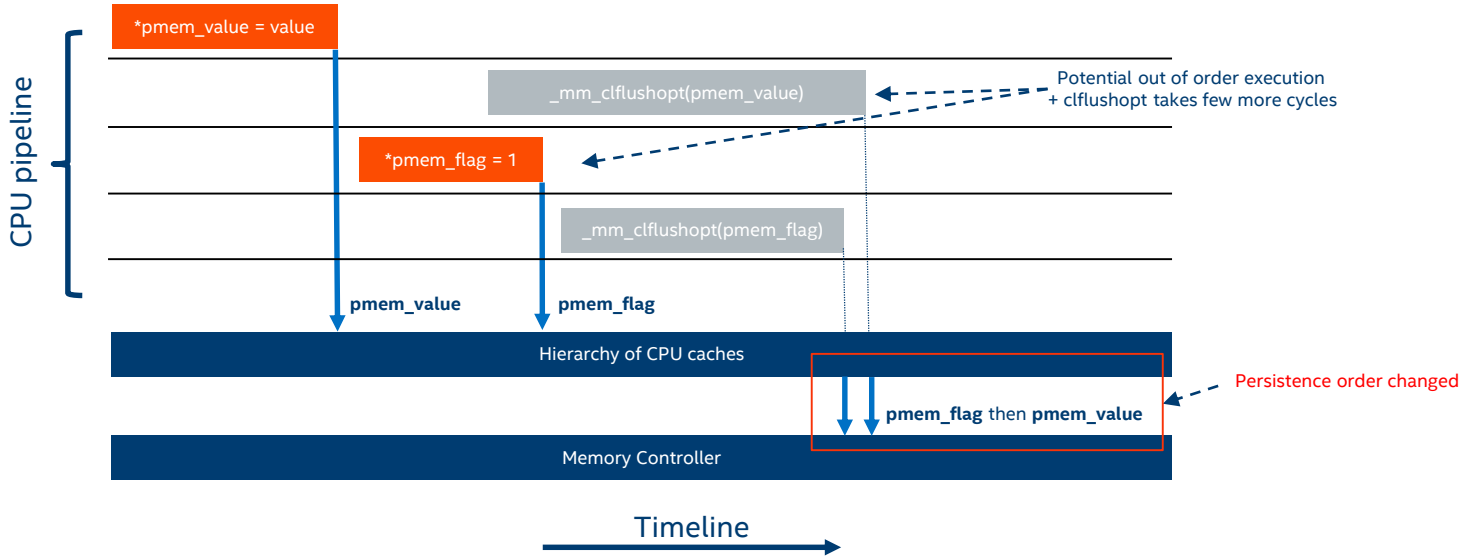
```
1  void load_value(uint32_t value, uint8_t* pmem_region)
2  {
3    uint8_t* pmem_flag = pmem_region;
4    uint32_t* pmem_value = (uint32_t*)(pmem_region + 256);
5
6    if(*pmem_flag)
7      printf("Value = %u", *pmem_value);
8  }
```



Execution reordering affects data correctness

```
1   void save_value(uint32_t value, uint8_t* pmem_region)
2   {
3     uint8_t* pmem_flag = pmem_region;
4     uint32_t* pmem_value = (uint32_t*)(pmem_region + 256);
5
6     *pmem_value = value;
7     _mm_clflushopt(pmem_value);
8     *pmem_flag = 1;
9     _mm_clflushopt(pmem_flag);
10  }
```

- All stores are in order on x86 architecture
- clflushopt is ordered with respect to stores to that cache line

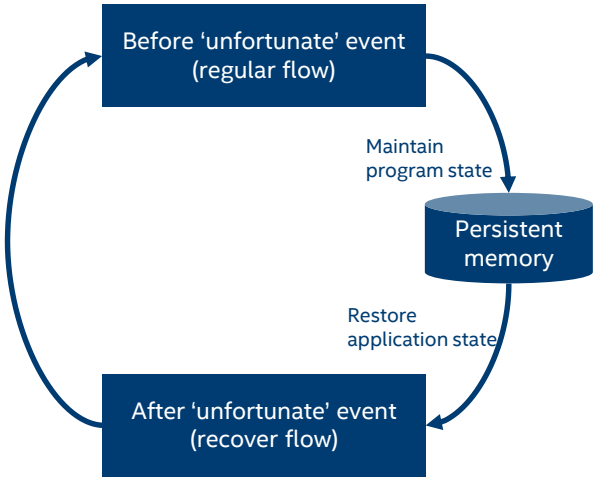


SOFTWARE SIDE OF DATA PERSISTENCE

Persistent Memory Program Workflow

A typical application that is expected to be ‘durable’ has 2 stages in its workflow:

- Normal program flow where application just works as it is by keeping its state in persistent memory.
- Then some ‘unfortunate’ event happens and application, system or hardware stops working. It requires application to ‘recover’ from that and continue to work normally.



Persistent Memory Program

```
struct address {
        char name[64];
        char address[64];
        int valid;
};
int main()
{
        struct address *head = NULL;
        int fd;

        fd = open("addressbook.pmem", O_CREAT|O_RDWR, 0666);
        posix_fallocate(fd, 0, sizeof(struct address));
        head = (struct address *)mmap(NULL, sizeof(struct address),  PROT_READ|PROT_WRITE, MAP_SHARED, fd, 0);
        close(fd);
        strcpy(head->name, "Clark Kent");
        strcpy(head->address, "344 Clinton St, Metropolis, DC 95308");
        head->valid = 1;
        munmap(head, sizeof(struct address));
        return 0;
}
```

Consider power failure after this line

What Could Be In The Memory If Power Fails?

Clark Kent\0
344 Clinton St, Metropolis, DC 95308\0
1

Clark Kent\0
344 Clinton St, Metropolis, DC 95308\0
0

Clark Kent\0
\0
1

\0
\0
1

\0
344 Clinton St, Metropolis, DC 95308\0
1

.....

Persistent Memory Program (Corrected)

```
struct address {
        char name[64];
        char address[64];
        int valid;
};
int main()
{
        struct address *head = NULL;
        int fd;

        fd = open("addressbook.pmem", O_CREAT|O_RDWR, 0666);
        posix_fallocate(fd, 0, sizeof(struct address));
        head = (struct address *)mmap(NULL, sizeof(struct address),  PROT_READ|PROT_WRITE, MAP_SHARED, fd, 0);
        close(fd);
        strcpy(head->name, "Clark Kent");
        strcpy(head->address, "344 Clinton St, Metropolis, DC 95308");
        _mm_clflushopt(&(head->name));
        _mm_clflushopt(&(head->address));
        _mm_sfence();
        head->valid = 1;
        _mm_clflushopt(&(head->valid));
        _mm_sfence();
        munmap(head, sizeof(struct address));
        return 0;
}
```

Persistent Memory Programming Challenges

- When to flush stored data out of cache hierarchy?
  - Memory store does not become persistent immediately
  - Data gets persistent only after it is out of cache and arrives at the memory subsystem
- Missing or incorrect cache flushes can leave data in inconsistent or unrecoverable state in case of power failure or system crash
- Excessive cache flushes hurt performance
- Testing and existing development tools do not find missing/incorrect/ excessive cache flushes

VERIFY PROGRAM CORRECTNESS

Intel® Inspector - Persistence Inspector

- Find correctness and performance issues (missing/redundant cache flushes, missing store/fences, out-of-order stores, etc.) in app-direct programs
- Target developers of libraries and/or applications directly managing persistence
- Can be used as a design tool (find places to insert flushes)
- Issues do not need to actually occur
- Pinpoint issues to exact source locations with stack traces
- No source changes required
- Analysis of 64-bit application on Linux and Windows
- PMDK (http://pmem.io/nvml/) support

PMDK Transactions Support

- Update without undo log detection
- Undo log without update detection

Before unfortunate event

Use the following command to trace all stores into persistent memory:

```
pmeminsp cb -- <application command>
```

If your application direct memory mapping instead of NVM library, then you have to additionally specify file paths in persistence memory to look at. In order to do that, add '-pmem-file' option. E.g.

```
pmeminsp cb -pmem-file /tmp/database.dat -- <application command>
```

After unfortunate event

Use the following command to analyze data dependencies when load data from persistent memory:

```
pmeminsp ca -- <application command>
```

Similarly to ‘before unfortunate event’ command, if your application direct memory mapping instead of NVM library, then you have to additionally specify file paths in persistence memory to look at. In order to do that, add ‘-pmem-file’ option. E.g.

```
pmeminsp ca -pmem-file /tmp/database.dat -- <application command>
```

Generating text report

- If your application uses transaction API from NVM library and never access persistence memory directly, then you don't need to do analysis of applications' recovery step. Persistent Inspector can check correctness of this API usage as well as find unprotected regions.
- If your applications has direct accesses to persistent memory, then Persistent Inspector needs data from both application stages for correctness analysis.

In either case, in order to generate report, the following command should be used:

```
pmeminsp report -- <application name>
```

Missing Cache Flush

The first memory store
  in /tests/switch/src/writeswitch!writeswitch at writeswitch.c:33 - 0x692
  in /tests/switch/src/writeswitch!main at writeswitch.c:63 - 0x74D
  in /tests/switch/src/writeswitch!main at writeswitch.c:58 - 0x729
  in /lib/x86_64-linux-gnu/libc.so.6!__libc_start_main at <unknown_file>:<unknown_line> - 0x21F43
  in /tests/switch/src/writeswitch!_start at <unknown_file>:<unknown_line> - 0x594

is not flushed before

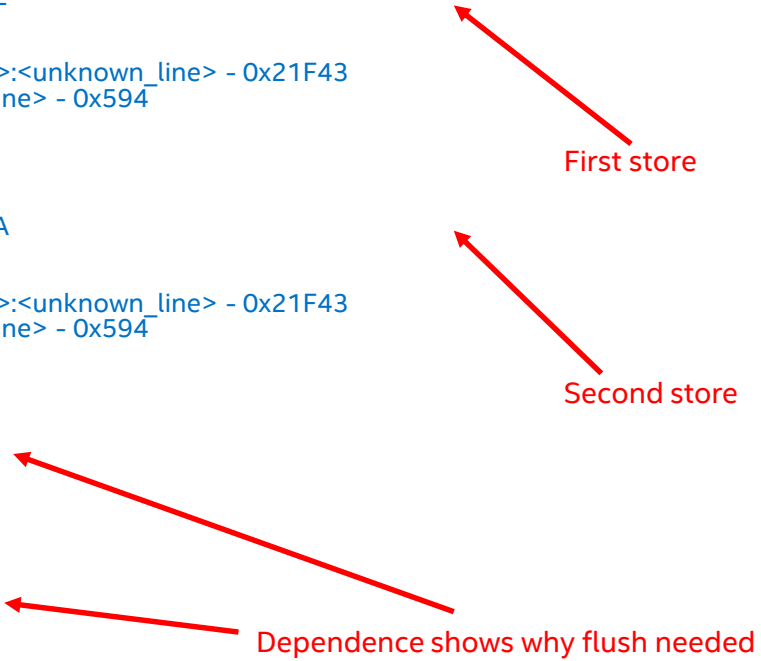
the second memory store
  in /tests/switch/src/writeswitch!writeswitch at writeswitch.c:49 - 0x6FA
  in /tests/switch/src/writeswitch!main at writeswitch.c:63 - 0x74D
  in /tests/switch/src/writeswitch!main at writeswitch.c:58 - 0x729
  in /lib/x86_64-linux-gnu/libc.so.6!__libc_start_main at <unknown_file>:<unknown_line> - 0x21F43
  in /tests/switch/src/writeswitch!_start at <unknown_file>:<unknown_line> - 0x594

while

memory load from the location of the first store
  in /tests/switch/src/readswitch!readswitch at readswitch.c:37 - 0x801

depends on

memory load from the location of the second store
  in /tests/switch/src/readswitch!readswitch at readswitch.c:34 - 0x7D8



Redundant Cache Flush

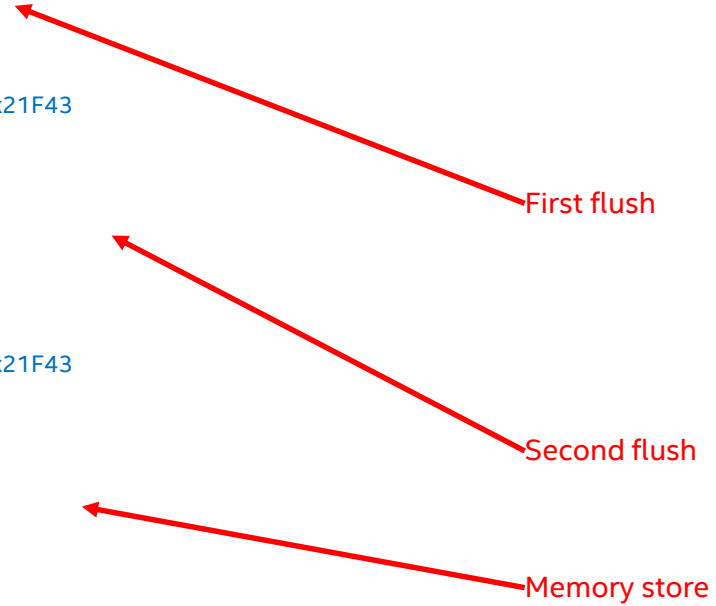
Cache line flush
in /tests/pmemdemo/src/pemmdemo!test_redundant_flush at main.cpp:134 - 0x1721
in /tests/pmemdemo/src/pemmdemo!create_data_file at main.cpp:52 - 0x151F
in /tests/pmemdemo/src/pemmdemo!create_data_file at main.cpp:48 - 0x14FD
in /tests/pmemdemo/src/pemmdemo!main at main.cpp:231 - 0x1C74
in /lib/x86_64-linux-gnu/libc.so.6!__libc_start_main at <unknown_file>:<unknown_line> - 0x21F43
in /tests/pmemdemo/src/pemmdemo!_start at <unknown_file>:<unknown_line> - 0x12A4

is redundant with regard to cache line flush

in /tests/pmemdemo/src/pemmdemo!test_redundant_flush at main.cpp:135 - 0x1732
in /tests/pmemdemo/src/pemmdemo!create_data_file at main.cpp:52 - 0x151F
in /tests/pmemdemo/src/pemmdemo!create_data_file at main.cpp:48 - 0x14FD
in /tests/pmemdemo/src/pemmdemo!main at main.cpp:231 - 0x1C74
in /lib/x86_64-linux-gnu/libc.so.6!__libc_start_main at <unknown_file>:<unknown_line> - 0x21F43
in /tests/pmemdemo/src/pemmdemo!_start at <unknown_file>:<unknown_line> - 0x12A4

of memory store

in /tests/pmemdemo/src/pemmdemo!test_redundant_flush at main.cpp:133 - 0x170F
in /tests/pmemdemo/src/pemmdemo!create_data_file at main.cpp:52 - 0x151F
in /tests/pmemdemo/src/pemmdemo!create_data_file at main.cpp:48 - 0x14FD
in /tests/pmemdemo/src/pemmdemo!main at main.cpp:231 - 0x1C74
in /lib/x86_64-linux-gnu/libc.so.6!__libc_start_main at <unknown_file>:<unknown_line> - 0x21F43
in /tests/pmemdemo/src/pemmdemo!_start at <unknown_file>:<unknown_line> - 0x12A4

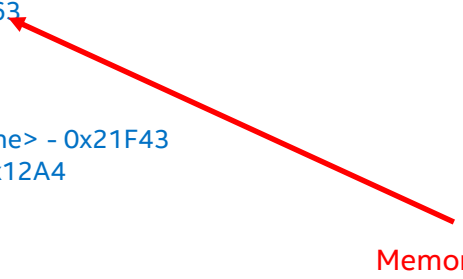


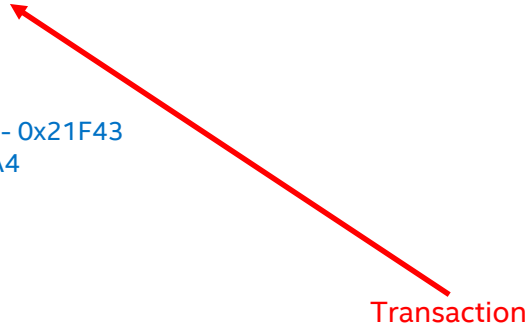
Undo Log Checking

Memory store
in /tests/pmemdemo/src/pemmdemo!test_tx_without_undo at main.cpp:190 - 0x1963
in /tests/pmemdemo/src/pemmdemo!run_tx_test at main.cpp:175 - 0x1877
in /tests/pmemdemo/src/pemmdemo!run_tx_test at main.cpp:163 - 0x180D
in /tests/pmemdemo/src/pemmdemo!main at main.cpp:245 - 0x1CF4
in /lib/x86_64-linux-gnu/libc.so.6!__libc_start_main at <unknown_file>:<unknown_line> - 0x21F43
in /tests/pmemdemo/src/pemmdemo!_start at <unknown_file>:<unknown_line> - 0x12A4

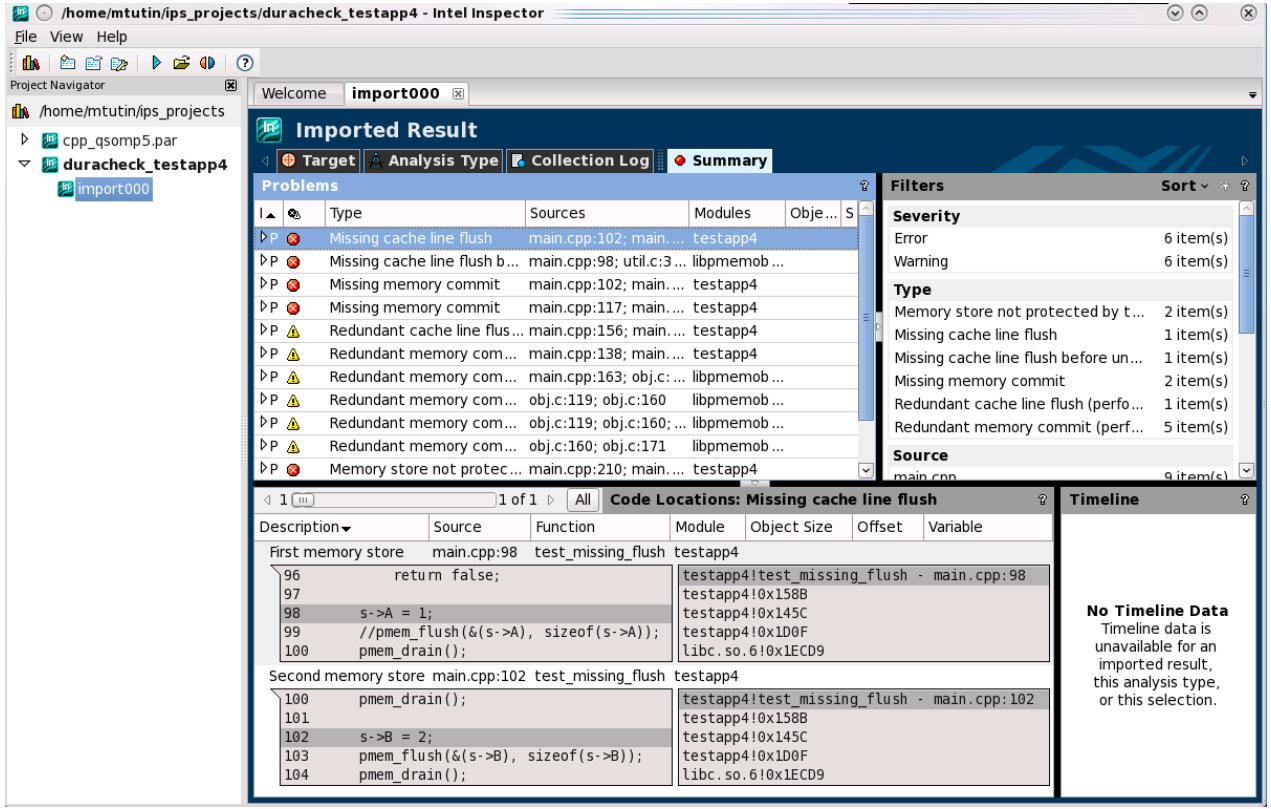
not undo logged in transaction

in /tests/pmemdemo/src/pemmdemo!test_tx_without_undo at main.cpp:185 - 0x1921
in /tests/pmemdemo/src/pemmdemo!run_tx_test at main.cpp:175 - 0x1877
in /tests/pmemdemo/src/pemmdemo!run_tx_test at main.cpp:163 - 0x180D
in //tests/pmemdemo/src/pemmdemo!main at main.cpp:245 - 0x1CF4
in /lib/x86_64-linux-gnu/libc.so.6!__libc_start_main at <unknown_file>:<unknown_line> - 0x21F43
in /tests/pmemdemo/src/pemmdemo!_start at <unknown_file>:<unknown_line> - 0x12A4





Inspector report example



Perspective future development

- Memory leaks and unused memory blocks detection
- Improve out of order stores analysis to detect potential data durability issues

Legal Disclaimer & Optimization Notice

INFORMATION IN THIS DOCUMENT IS PROVIDED “AS IS”. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO THIS INFORMATION INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products.

Optimization Notice

Intel’s compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804


intel