PERSISTENT MEMORY EMULATION & PROGRAMMING

Diego Cepeda
August 2, 2019
dcepeda@uwaterloo.ca

UNIVERSITY OF WATERLOO

PERSISTENT MEMORY

- Pmem has hybrid properties of volatile memory and disk storage.
- Being a new technology not everyone can have access to machines that use Pmem.
- Pmem emulation allows the development of persistent memory applications.

NON-VOLATILE MEMORY PROGRAMMING MODEL

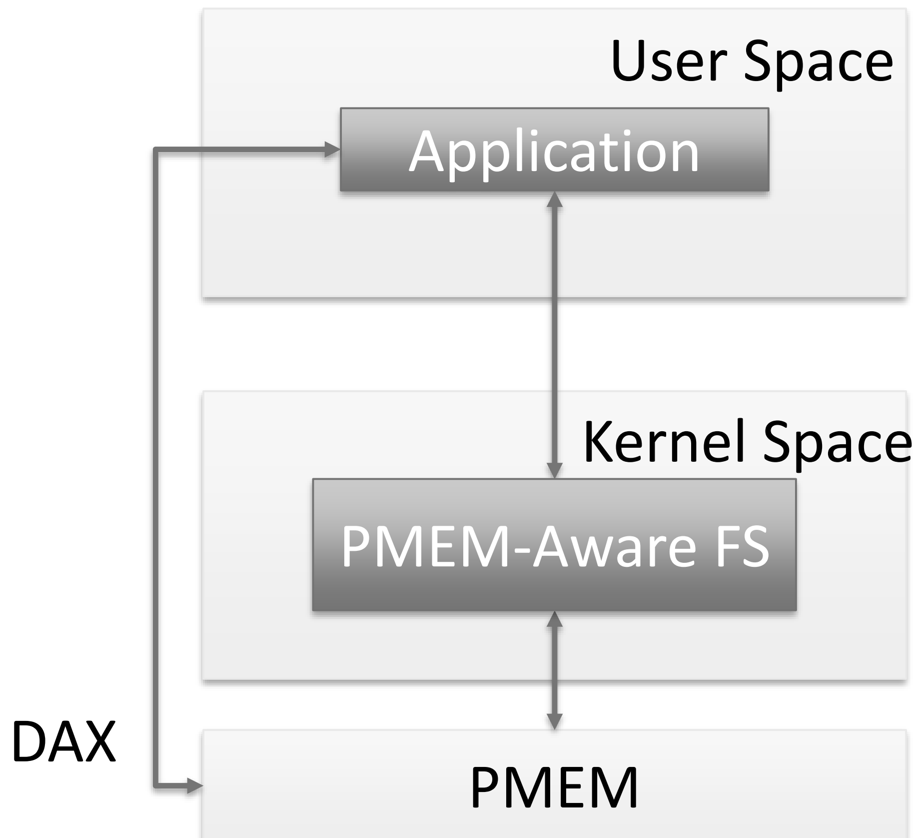
- Memory mapped files
  - » Building block of the Pmem programming model.
  - » Allows accessing the contents of a file in virtual memory.
  - » Allows programs to modify the file by reading and writing memory directly.
  - » To persist the changes on the memory mapped files, they need to be flushed to the storage medium.

https://www.snia.org/tech_activities/standards/curr_standards/npm

NON-VOLATILE MEMORY PROGRAMMING MODEL

- Pmem Aware File system
  - » Direct access (DAX), which is a fast way to access the medium without involving the kernel.
  - » DAX eliminates the use of page cache.
  - » DAX is currently supported by Windows and Linux.

NON-VOLATILE MEMORY PROGRAMMING MODEL



PERSISTENT MEMORY EMULATION

- Currently there are different options for emulating Pmem.
  - » Linux: memmap Kernel Option
  - » Virtual machine
    - QEMU
    - Vmware VSphere

EMULATION ON A LINUX SYSTEM

- What can be done
  - » Development teams can work in parallel on their own emulated system rather than all of them needing access to a machine with persistent memory.
  - » Program crash testing can be done, and logical behavior can be verified.
- What can't be done
  - » Simulating a power failure is still a difficult topic to address in emulation, given the implication of cache loss and the complexity involving the appropriate real-life behavior of a system in these conditions.

EMULATION ON A LINUX SYSTEM

- Memmap Kernel option :
  - » This allows users to define a specific region of DRAM to be reserved.
  - » This will mark the region as a non-standard e820 type of 12.
  - » The kernel will offer these regions to the 'pmem' driver so they can be used for emulated persistent storage.

EMULATION OF PMEM ON A LINUX MACHINE

- Example

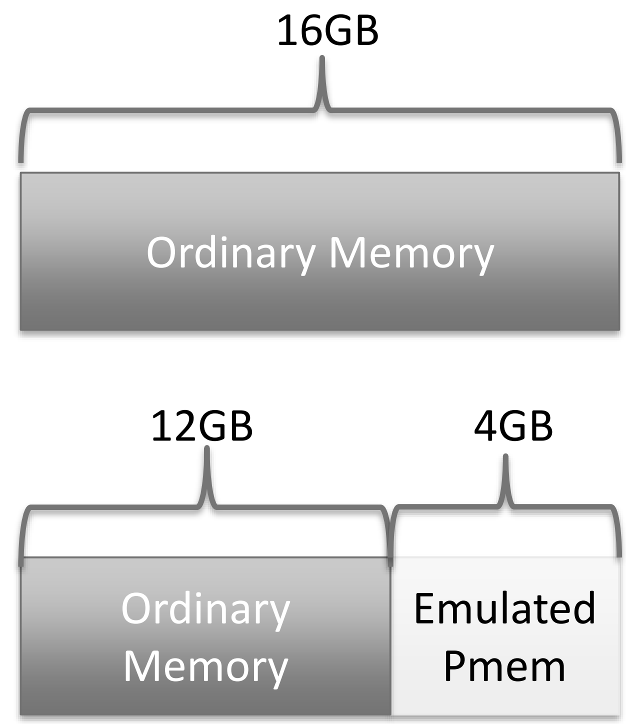
  memmap=**nn**[KMG]!**ss**[KMG]

  nn being region to reserve

  ss starting offset

  memmap=4G!12G

  Reserve 4GB starting from 12GB



EMULATION ON A LINUX SYSTEM

- Emulation checklist:
  1. Make sure the memory region that will be reserved is not overlapping with already reserved memory, as failing to do so might corrupt your system or produce undefined behavior.
     - sudo dmesg | grep BIOS-e820
     - sudo dmesg | sed -n 's/   0.000000] BIOS-e820://p'
  2. Edit the grub configuration to set the memmap option
     - sudo nano /etc/default/grub

EMULATION ON A LINUX SYSTEM

- Emulation checklist:
  3. Update your grub configuration and reboot system
     - sudo update-grub2
  4. Verify the Pmem device is correctly configured
     - sudo dmesg | grep user:
     - sudo dmesg | sed -n 's/   0.000000] user://p'
  5. After correctly setting up your Pmem device, it should appear under /dev/pmem0, and we would be ready to create our **DAX filesystem**.

EMULATION ON A LINUX SYSTEM

- Emulation checklist:
  6. After successfully setting up your device you will be able to mount your device in DAX mode.
  7. Make sure to assign the proper permissions on the mount location so files can be created on the Pmem aware filesystem.

```
sudo mkfs.ext4 /dev/pmem0
sudo mkdir /mnt/mem/
sudo mount -o dax /dev/pmem0 /mnt/mem
sudo mount -v | grep /mnt/mem
sudo chmod 777 /mnt/mem
```

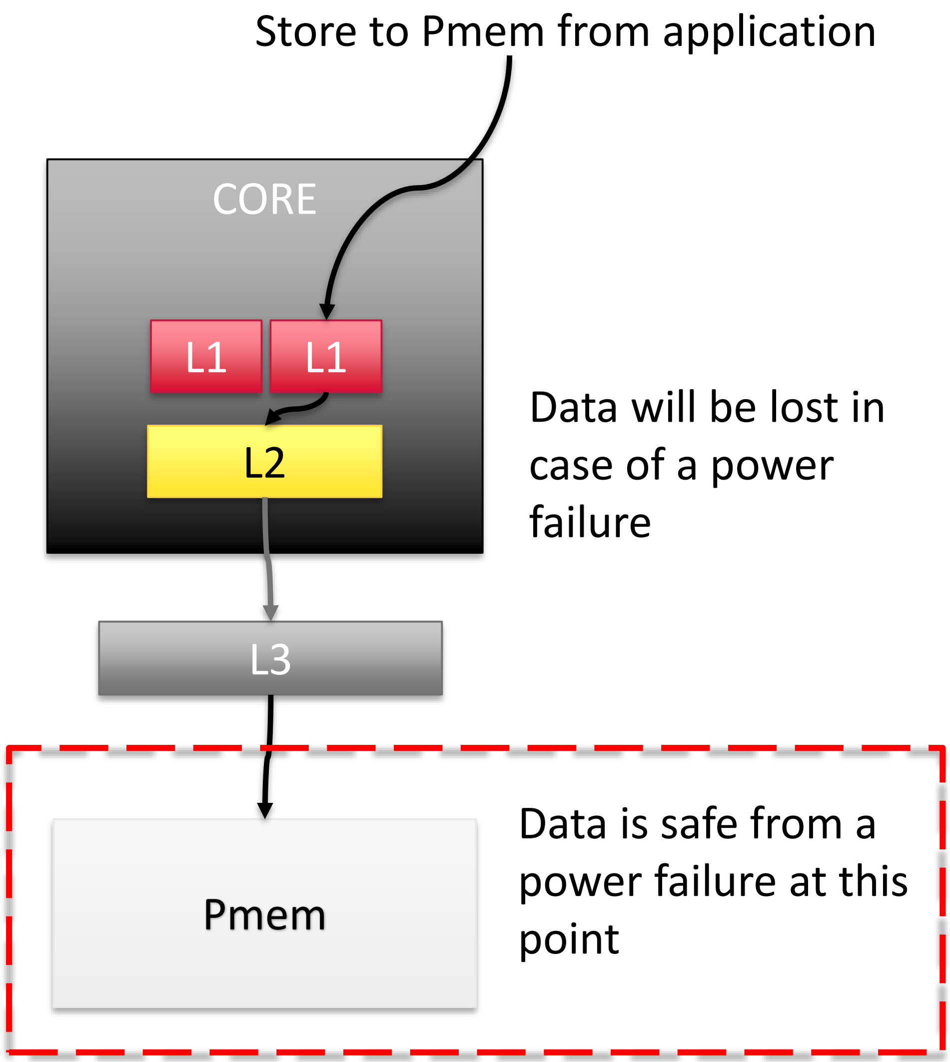
PERSISTENT MEMORY PROGRAMMING

CHALLENGES OF PMEM PROGRAMMING

- Currently, and probably for a long-time, cache memory will remain volatile.

- Flushing Instructions:
  - » CLFLUSH
  - » CLFLUSHOPT
  - » CLWB

- The only store to Pmem guaranteed to be atomic in case of a power failure is an 8-byte store aligned on an eight-byte boundary.



CHALLENGES OF PMEM PROGRAMMING

- Programs using persistent memory should always create/open their corresponding files early on initialization.

- A valid state of your in-memory data structure should always be kept to provide the expected behavior of the program.

SOME AVAILABLE DEVELOPMENT ENVIRONMENTS

- PMDK (Persistent Memory Development Kit)
  - » C
- PCJ (Persistent Collections for Java)
  - » Java 8 or above

PERSISTENT MEMORY DEVELOPMENT KIT

- Collection of libraries and tools and utilities.
  - libpmem
  - libpmemobj
  - libpmemlog

LIBPMEM

- Low level Pmem support.
- Freedom to handle memory allocation and consistency of your program.
- Does not ensure atomicity, even when calling functions that flush data to persistent memory.

LIBPMEM BASIC API

- **pmem_map_file**()
  - » Creates a new read/write mapping for the named file.
- **pmem_unmap**()
  - » Deletes all the mappings for the specified address range.
- **pmem_is_pmem**()
  - » Returns true only if the entire range [*addr*, *addr+len*) consists of persistent memory.

LIBPMEM BASIC API

- **pmem_memcpy_persist()**
  - » Same functionality as *memcpy(),* but also ensures that the result has been flushed to persistence before returning.
- **pmem_persist**()
  - » Force any changes in the range [addr, addr+len) to be stored durably in persistent memory.
- **pmem_msync**()
  - » Same functionality as *pmem_persist()*, but using msync(), this function works on either Pmem or a memory mapped file on traditional storage.

LIBPMEM EXAMPLE

```c
int main(int argc, char *argv[])
{  char *pmemaddr;
   size_t mapped_len;
   int is_pmem;
   /* create a pmem file and memory map it */
   if ((pmemaddr = pmem_map_file(PATH, PMEM_LEN, PMEM_FILE_CREATE,
         0666, &mapped_len, &is_pmem)) == NULL) {
         perror("pmem_map_file");
         exit(1);
   }
   /* store a string to the persistent memory */
   strcpy(pmemaddr, "hello, persistent memory");
   /* flush above strcpy to persistence */
   if (is_pmem)
      pmem_persist(pmemaddr, mapped_len);
   else
      pmem_msync(pmemaddr, mapped_len);
   /*
    * Delete the mappings. The region is also
    * automatically unmapped when the process is
    * terminated.
    */
   pmem_unmap(pmemaddr, mapped_len);
}
```

LIBPMEMOBJ

- High level library abstracting the complexity of ensuring persistence
  - Flexible object store
  - Transactions
  - Memory management
  - Locking

LIBPMEMOBJ BASIC API

- **pmemobj_tx_add_range()**
  - » takes a “snapshot” of the memory block of given size, located at given offset and saves it to the undo log.
- **pmemobj_create()**
  - » creates a transactional object store with the given total pool-size
- **pmemobj_root()**
  - » creates or resizes the root object for the persistent memory pool.
- **pmemobj_direct()**
  - » returns a pointer to the PMEMoid object

LIBPMEMOBJ EXAMPLE

```
#define LAYOUT_NAME "intro_2"
#define MAX_BUF_LEN 10
struct my_root {
   char buf[MAX_BUF_LEN];
};
int main(int argc, char *argv[])
{
   if (argc != 2) {
      printf("usage: %s file-name\n", argv[0]);
      return 1;
   }
   PMEMobjpool *pop = pmemobj_create(argv[1], LAYOUT_NAME,
      PMEMOBJ_MIN_POOL, 0666);
   if (pop == NULL) {
      perror("pmemobj_create");
      return 1;
   }
```

LIBPMEMOBJ EXAMPLE

```
   PMEMoid root = pmemobj_root(pop, sizeof(struct my_root));
   struct my_root *rootp = pmemobj_direct(root);
   char buf[MAX_BUF_LEN] = { 0 };
   if (scanf("%9s", buf) == EOF) {
       fprintf(stderr, "EOF\n");
       return 1;
   }
   TX_BEGIN(pop) {
       pmemobj_tx_add_range(root, 0, sizeof(struct my_root));
       memcpy(rootp->buf, buf, strlen(buf));
   } TX_END
       pmemobj_close(pop);
   return 0;
}
```

If you know the virtual address the pool is mapped at, a simple addition can be performed to get the direct pointer, like this: (void *)((uint64_t)root + offset)

LIBPMEMOBJ EXAMPLE

```c
#define LAYOUT_NAME "intro_2"
#define MAX_BUF_LEN 10
struct my_root {
  char buf[MAX_BUF_LEN];
};
int main(int argc, char *argv[])
{
  if (argc != 2) {
    printf("usage: %s file-name\n", argv[0]);
    return 1;
  }
  PMEMobjpool *pop = pmemobj_create(argv[1], LAYOUT_NAME,
      PMEMOBJ_MIN_POOL, 0666);
  if (pop == NULL) {
    perror("pmemobj_create");
    return 1;
  }
  PMEMoid root = pmemobj_root(pop, sizeof(struct my_root));
  struct my_root *rootp = pmemobj_direct(root);
  printf("%s\n", rootp->buf);
  pmemobj_close(pop);
  return 0;
}
```

LIBPMEMOBJ EXAMPLE

- The intended use of the TX_ONCOMMIT and TX_ONABORT macros is to print log information and set return variable of the function.

```c
int do_work() {
    int ret;
    TX_BEGIN(pop) {
    } TX_ONABORT{
        LOG_ERR("work transaction failed");
        ret = 1;
    } TX_ONCOMMIT{
        LOG("work transaction successful");
        ret = 0;
    } TX_END
        return ret;
}
```

LIBPMEMOBJ & CONCURRENCY

- All the libpmemobj library functions are thread-safe.
- Exceptions:
  - » pool management functions (open, close, etc.)
  - » pmemobj_root() when providing different sizes in different threads
- A single transaction block works in the context of a single thread.

LIBPMEMOBJ & CONCURRENCY

- A crash in fetch_and_add, will cause the pthread_mutex_t structure to contain invalid values and the application will most likely segfault when an attempt to use it is made.

```c
struct foo {
   pthread_mutex_t lock;
   int bar;
};
int fetch_and_add(TOID(struct foo) foo, int val) {
   pthread_mutex_lock(&D_RW(foo)->lock);
   int ret = D_RO(foo)->bar;
   D_RW(foo)->bar += val;
   pthread_mutex_unlock(&D_RW(foo)->lock);
   return ret;
}
```

LIBPMEMOBJ & CONCURRENCY

- To put a lock in a structure that resides on persistent memory, libpmemobj provides a pthread-like API.

```
struct foo {
    PMEMmutex lock;
    int bar;
};

int fetch_and_add(TOID(struct foo) foo, int val) {
    pmemobj_mutex_lock(pop, &D_RW(foo)->lock);
    int ret = D_RO(foo)->bar;
    D_RW(foo)->bar += val;
    pmemobj_mutex_unlock(pop, &D_RW(foo)->lock);
    return ret;
}
```

LIBPMEMOBJ & CONCURRENCY

- There’s no need to initialize those locks or to verify their state. When an application crashes, they are all automatically released.

LIBPMEMOBJ & CONCURRENCY

- Using **TX_PARAM_MUTEX** or **TX_PARAM_ RWLOCK** causes the specified lock to be acquired at the beginning of the transaction.

```
/* begin a transaction, and adquire mutex */
   TX_BEGIN_PARAM(Pop, TX_PARAM_MUTEX, &D_RW(ep)->mtx, TX_PARAM_NONE) {
        TX_ADD(ep);
        D_RW(ep)->count++;
   } TX_END
```

LIBPMEMLOG

- Log variable length entries
- Handles the transactional update of the log.
- Useful during development to quickly log information about your program without the overhead of writing to disk

LIBPMEMLOG BASIC API

- **pmemlog_open**()
  - » opens an existing log memory pool
- **pmemlog_create**()
  - » creates a log memory pool with the given total poolsize
- **pmemlog_append()**
  - » appends to the current write offset in the log memory pool
- **pmemlog_walk**()
  - » walks through the log, from beginning to end, calling the callback function

LIBPMEMLOG EXAMPLE

```c
int main(int argc, char *argv[])
{
  const char path[] = "/mnt/mem/log";
  PMEMlogpool *plp;
  size_t nbyte;
  char *str;
  /* create the pmemlog pool or open it if it already exists */
  plp = pmemlog_create(path, POOL_SIZE, 0666);
  if (plp == NULL)
    plp = pmemlog_open(path);

  if (plp == NULL) {
    perror(path);
    exit(1);
  }
  /* how many bytes does the log hold? */
  nbyte = pmemlog_nbyte(plp);
  printf("log holds %zu bytes\n", nbyte);
  /* append to the log... */
  str = "This is the first string appended\n";
  if (pmemlog_append(plp, str, strlen(str)) < 0) {
    perror("pmemlog_append");
    exit(1);
  }
  /* print the log contents */
  printf("log contains:\n");
  pmemlog_walk(plp, 0, printit, NULL);
  pmemlog_close(plp);
}

int printit(const void *buf, size_t len, void *arg)
{
  fwrite(buf, len, 1, stdout);
  return 0;
}
```

PMDK UTILITIES

- Pmemcheck
  - » Tracks stores you make to persistent memory and informs you of possible memory violations.
  - » Integrated with valgrind

valgrind ***--tool****=pmemcheck [valgrind options] <your_app> [your_app options]*

- Pmreorder
  - » Collection of python scripts designed to parse, and replay operations logged by pmemcheck

PERSISTENT COLLECTIONS FOR JAVA (PCJ)

- Provides a set of classes that persist beyond the life of the java VM instance.

- This library is based on the libpmemobj library, *(transactional operations)*

- NOTE: Pilot project currently under development.
- For more information on this library.
  - » https://github.com/pmem/pcj

LIST OF PERSISTENT CLASSES

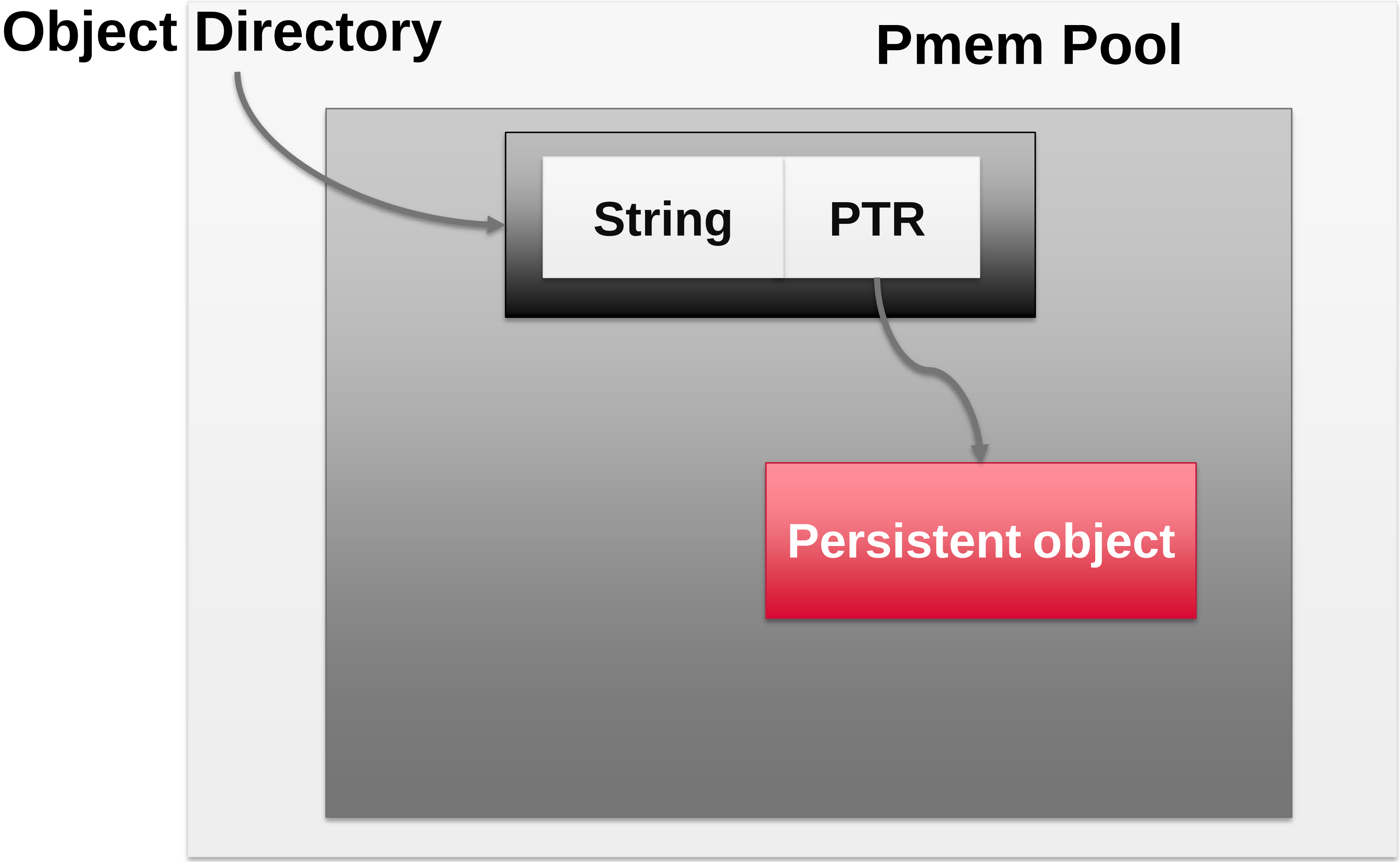
- Primitive arrays
- Generic arrays
- Tuples
- ArrayList
- HashMap
- LinkedList
- LinkedQueue
- SkipListMap
- FPTree
- SIHashMap
- ObjectDirectory
- Boxed primitives
- String
- AtomicReference
- ByteBuffer

PCJ EXAMPLE

```
import lib.util.persistent.*;
public class EmployeeList {
    static PersistentArray<Employee> employees;
    public static void main(String[] args) {
        // fetching back main employee list (or creating it if it is not there)
        if ((employees = ObjectDirectory.get("employees", PersistentArray.class)) == null) {
            employees = new PersistentArray<Employee>(64);
            ObjectDirectory.put("employees", employees);
            // creating objects
            for (int i = 0; i < 64; i++) {
                Employee employee = new Employee(i,
                    new PersistentString("Fake Name"),
                    new PersistentString("Fake Department"));
                employees.set(i, employee);
            }
        }
        else {
            // reading objects
            for (int i = 0; i < 64; i++)
                if ((employees.get(i).getId() == i) == true)
                    System.out.print("OK ");
                else {
                    System.out.print("FAIL ");
                    break;
                }
            System.out.print("\n");
}}}
```

```
cat config.properties
path=/mnt/mem/persistent_heap
size=2147483648
```

PCJ EXAMPLE



PCJ EXAMPLE

```java
import lib.util.persistent.*;
import lib.util.persistent.types.*;

public final class Employee extends PersistentObject {
   private static final LongField ID = new LongField();
   private static final StringField NAME = new StringField();
   private static final StringField DEPARTMENT = new StringField();
   private static final ObjectType<Employee> TYPE =
   ObjectType.withFields(Employee.class, ID, NAME, DEPARTMENT);

   public Employee(long id, PersistentString name, PersistentString department) {
      super(TYPE);
      setLongField(ID, id);
      setName(name);
      setDepartment(department);
   }
   private Employee(ObjectPointer<Employee> p) {super(p);}
   public long getId() { getLongField(ID);}
   public PersistentString getName() {return getObjectField(NAME);}
   public PersistentString getDepartment() {return getObjectField(DEPARTMENT);}
   public void setName(PersistentString name) {setObjectField(NAME, name);}
   public void setDepartment(PersistentString department) {setObjectField(DEPARTMENT, department);
   public int hashCode() {return Long.hashCode(getId());}
   public String toString() {return String.format("Employee(%d, %s)", getId(), getName());}
   public boolean equals(Object obj) {
      if (!(obj instanceof Employee)) return false;
      Employee emp = (Employee)obj;
      return emp.getId() == getId() && emp.getName().equals(getName());
   }
}
```

These are not fields in the traditional way, but *meta fields*.
They serve as a guidance to PersistentObject.

REFERENCES

1. Rudoff A. (2017, June). *Persistent Memory Programming*. Retrieved from:https://www.usenix.org/system/files/login/articles/login_summer17_07_rudoff.pdf/

2. Rudoff A., USHARANI U., Andy M.(2017, August). *Introduction to Programming with Intel® Optane™ DC Persistent Memory*. Retrieved from:https://software.intel.com/en-us/articles/introduction-to-programming-with-persistent-memory-from-intel

3. Eduardo B., (2018, May). *Introduction to Java* API for Persistent Memory Programming*. Retrieved from:https://software.intel.com/en-us/articles/introduction-to-programming-with-persistent-memory-from-intel

4. *Persistent Memory Programming*. Retrieved from: https://pmem.io/

5. *Code Samples PMDK*. Retrieved from: https://github.com/pmem/pmdk

6. *Code Samples PCJ*. Retrieved from: https://github.com/pmem/pcj