Getting Started with NVM Programming

- NVM emulation
- File systems
  - BPFS
  - PMFS
- Persistent heaps
  - Mnemosyne
  - libpmem

Emulation Techniques

- Performance (based on DRAM)
  - Latency
  - Bandwidth
- Functionality
  - Persistent memory (NVM plugged on memory bus)
  - Power failures

Intel Hardware Emulator

- Latency emulation
  - Uses special microcode to inject additional stall cycles every few memory accesses
- Bandwidth emulation
  - Programs memory controller to throttle bandwidth

- Available through Intel

Software-created Delays

- Inject delays via:
  - **`clflush` + delay** for slower memory writes
  - **RAM-disk + delay** for slower block reads/writes
- Create delays using a spin-loop

```
for (start = RDTSCP; // read cpu timestamp counter
      RDTSCP – start < delay; )
```

- Available through Mnemosyne library

PCMSIM: NVM-disk Emulation

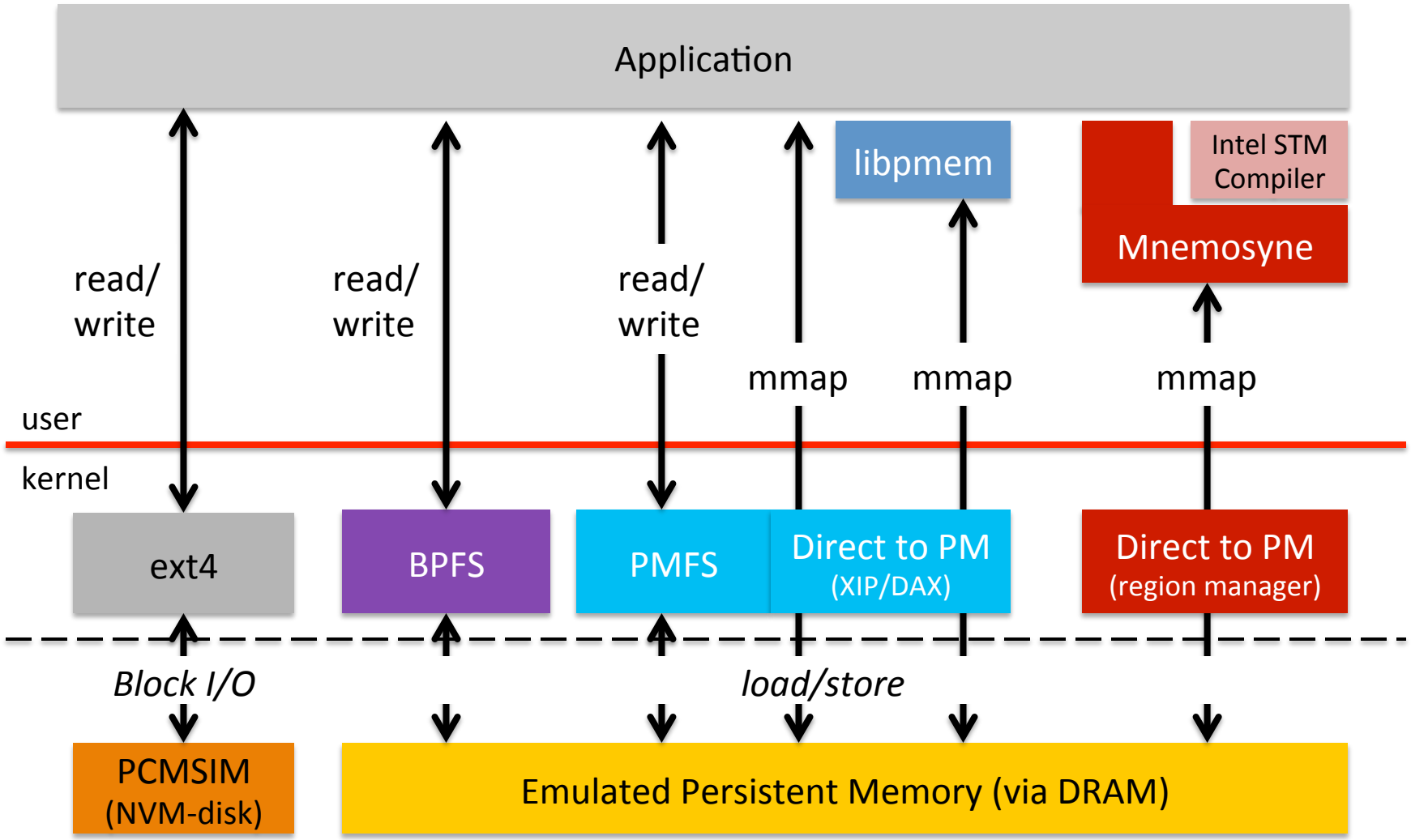
- Emulates a PCM-disk by slowing down DRAM
  - Implemented as a Linux device-driver
  - Exposes PCM-disk as `/dev/pcm0`
- Supports unmodified existing disk-based file systems (e.g. ext4)

- Available at: https://code.google.com/p/pcmsim/

Functionality

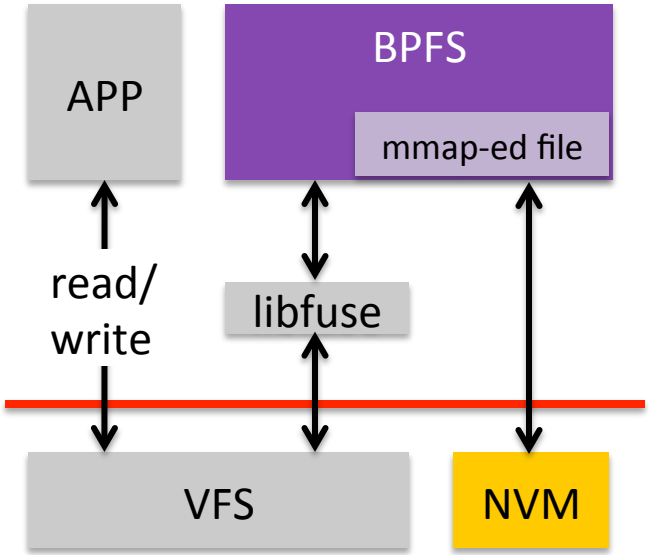
- Emulating PM: NVM plugged on memory bus
  - Use a memory-mapped file
  - Reserve DRAM when the kernel boots
- Validating software against power-failures
  - Kill the program using SIGKILL (but cache survives)
  - Unplug flash-backed NV-DIMMs
  - Inject failures via binary instrumentation (e.g., Yat)

Ecosystem Overview



BPFS (FUSE-based version)

- Supports crash-consistency via copy-on write (CoW)
  - Short circuit CoW
  - No epoch ordering
- Emulates PM through a memory-mapped file



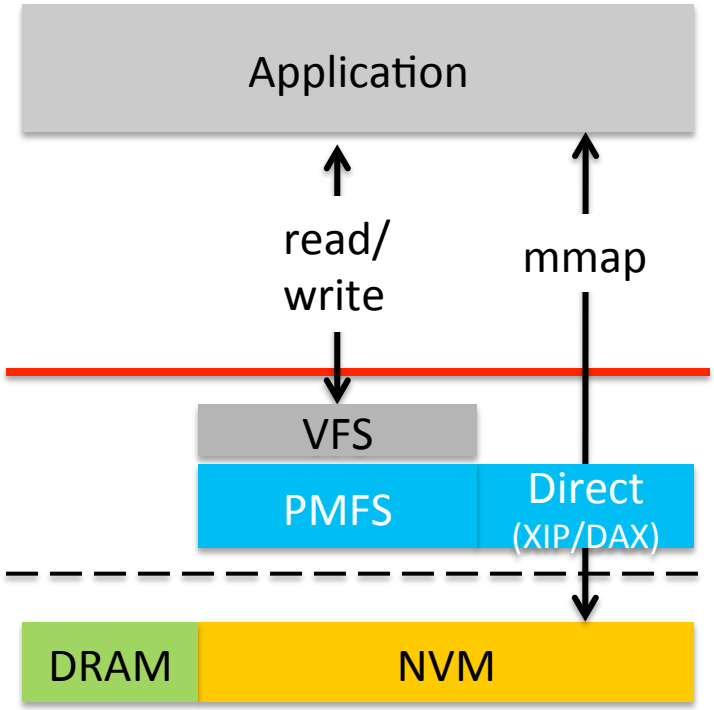
- Available at: http://bpfs.cs.ucla.edu

Getting started with BPFS

- Download
  ```
  > git clone git://bpfs.cs.ucla.edu/bpfs/bpfs
  ```
- Build (requires libfuse-dev installed)
  ```
  > make
  ```
- Create emulated-NVM backing store file
  ```
  > dd if=/dev/zero of=bpram.img bs=1M count=$N
  ```
- Format
  ```
  > ./mkfs.bpfs bpram.img $MNT
  ```
- Mount
  ```
  > ./bpfs –f bpram.mnt $MNT
  ```

PMFS

- Supports crash-consistency via metadata journaling
  - Ordering: clflushopt/pcommit
- Enables direct access to PM via XIP/DAX
- Emulates PM by reserving physically contiguous DRAM



- Available at: https://github.com/linux-pmfs/pmfs

Getting Started with PMFS

- Download Linux kernel 3.11 including PMFS

```
> git clone https://github.com/linux-pmfs/pmfs.git
```

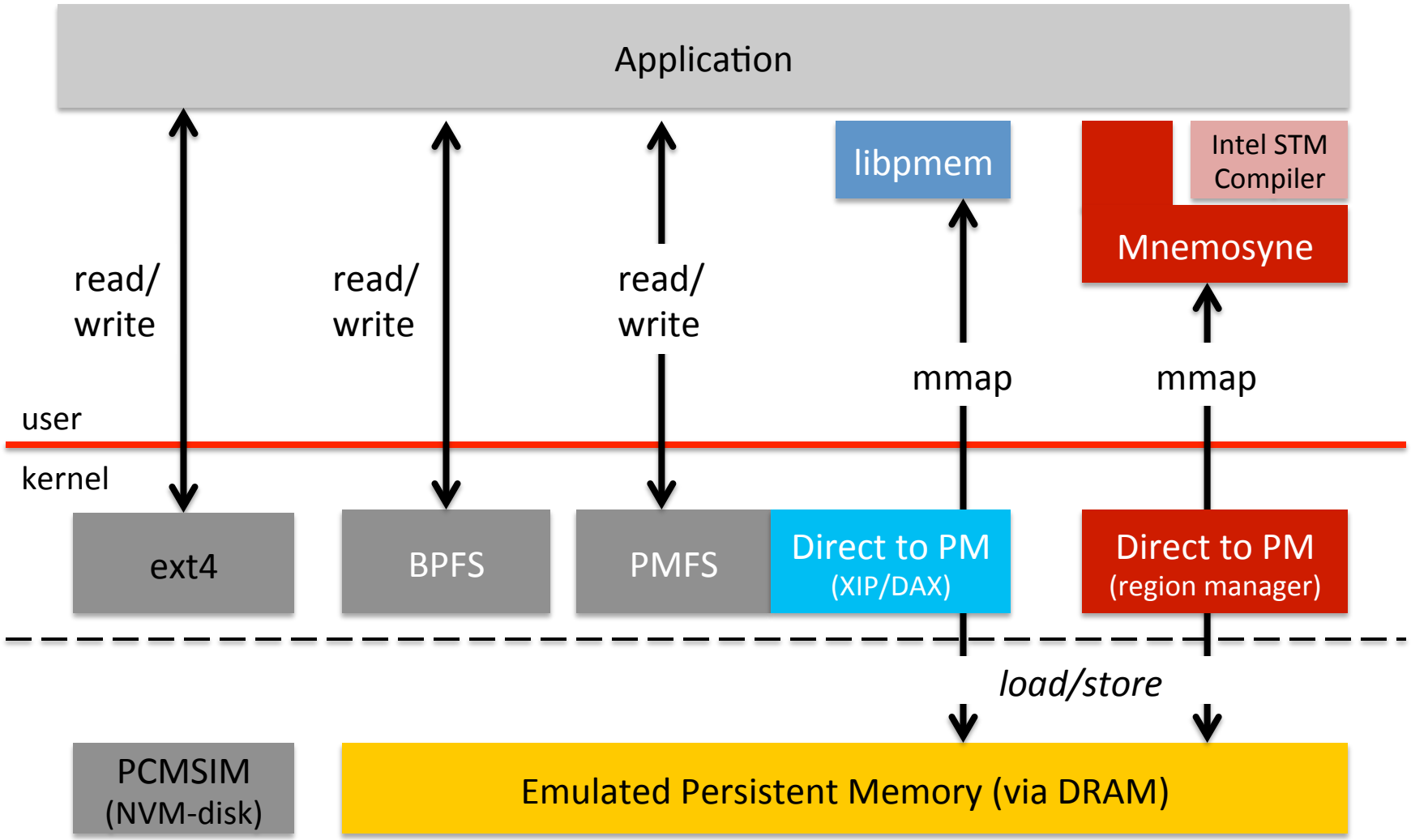
- Build

```
> make menuconfig # Enable PMFS and PMFS XIP
> make && make modules_install install
```

- Reserve DRAM for use as NVM via boot option
  - e.g. memmap=2G$4G: reserves 2G starting at 4G
- Mount

```
> mount -t pmfs -o physaddrs=0x100000000,init=2G none /mnt/pmfs
```

Persistent Heaps



Mnemosyne

- Mnemosyne API supports
  - Persistent memory regions
  - Ordering primitives
  - Log and durable memory transactions

- Available at:
  http://research.cs.wisc.edu/sonar/projects/mnemosyne

Mnemosyne Components

Class	API	Module
Persistent regions	pstatic var pmap(addr, len, prot, flags) punmap(addr, len) type persistent * ptr	**libmcore**
Persistent heap	pmalloc(sz, ptr) pfree(ptr)	**libpmalloc**
Ordering primitives	flush(addr) store(addr, val) wtstore(addr, val) fence()	**x86 instructions** **(mnemosyne.h)**
Log	log_create(flags, cbf) log_append(rec) log_flush() log_truncate()	**libmcore**
Durable Transactions	patomic { … }	**libmtm +** **Intel STM Compiler**

Allocation

Consistent Updates

Getting Started with Mnemosyne

- Download Mnemosyne and prerequisites
  - http://research.cs.wisc.edu/sonar/projects/mnemosyne
- Build

```
> cd $MNEMOSYNE/usermode
> scons
```

- Try

```
> scons --build-bench=kvstore
```

- Full documentation available online

libpmem

- Supports SNIA NVM API (Mnemosyne like)
- Builds on Linux Direct Access (DAX)
- Provides a collection of libraries
  - `libpmem`: low-level persistent memory support
  - `libpmemlog`: pmem-resident log file
  - `libpmemblk`: arrays of pmem-resident blocks with atomic updates
  - `libpmemobj`: transactional object store (under dev)

- Available at: http://pmem.io

Getting Started with libpmem

- Install prerequisites
  - Install PMFS
  - Install libuuid: `sudo apt-get install uuid-dev`
- Download NVM library

  ```
  > git clone https://github.com/pmem/nvml.git
  ```
- Build library

  ```
  > make
  ```

Example: Circular Queue

```
if ((fd = open("myqueue", O_RDWR)) < 0) {
  /* allocate queue */
  fd = open("myqueue", O_CREAT|O_RDWR, 0666)
  posix_fallocate(fd, 0, QUEUE_MAX_LEN * QUEUE_ITEM_SIZE)
  fsync(fd);
}

/* memory map it */
struct queue_s* queue = pmem_map(fd);

/* initialize queue (fail-safe against partial init) */
if (queue->valid == 0) {
  queue->head = queue->tail = 0;
  pmem_persist(queue, sizeof(*queue));
  queue->valid = 1;
  pmem_persist(&queue->valid, sizeof(queue->valid));
}
```

Force changes to NVM

Example: Circular Queue

```
enqueue(queue_t* queue, queue_item_t* data)
{
  if ((queue->tail + 1) % QUEUE_MAX_LEN
      != queue->head)
  {
    memcpy(queue->buffer[queue->tail], data,
           QUEUE_ITEM_SIZE);
    pmem_flush(&queue->buffer[queue->tail],
               QUEUE_ITEM_SIZE);
    pmem_drain();
    queue->tail = (tail + 1) % QUEUE_MAX_LEN;
    pmem_flush(&queue->tail, sizeof(queue->tail));
    pmem_drain();
  }
}
```

Flush processor caches

Wait for pmem stores to drain from HW buffers

Equivalent to pmem_persist

Adapting Disk-based Programs to NVM

- Workloads
  - TokyoCabinet: Key-value store
  - OpenLDAP: Directory service
- Approaches
  - Run on ext4 + NVM-disk
  - Run on PMFS/BPFS
  - Convert to use a persistent heap (e.g. Mnemosyne)

TokyoCabinet: Key-Value Store

- Original: msyncs B-tree to a mmap'd file
- Modified: keeps B-tree in persistent memory

Original version

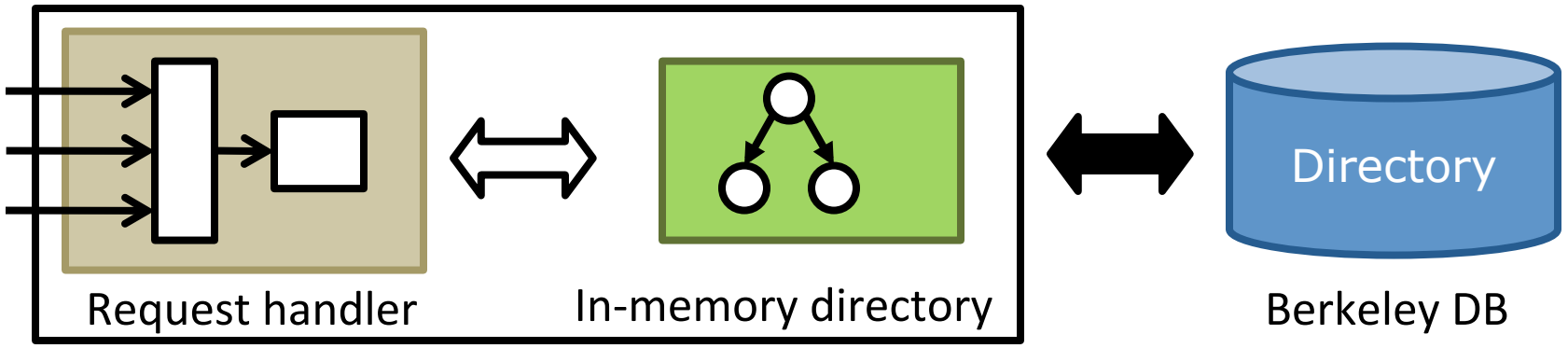
```
tcbdbopen(bdb, ...);

rec =
  TCMALLOC(sizeof(*nrec));

tcbdbput(bdb, &obj->key,
         ...);
tcbdbsync(bdb);
```

Mnemosyne version

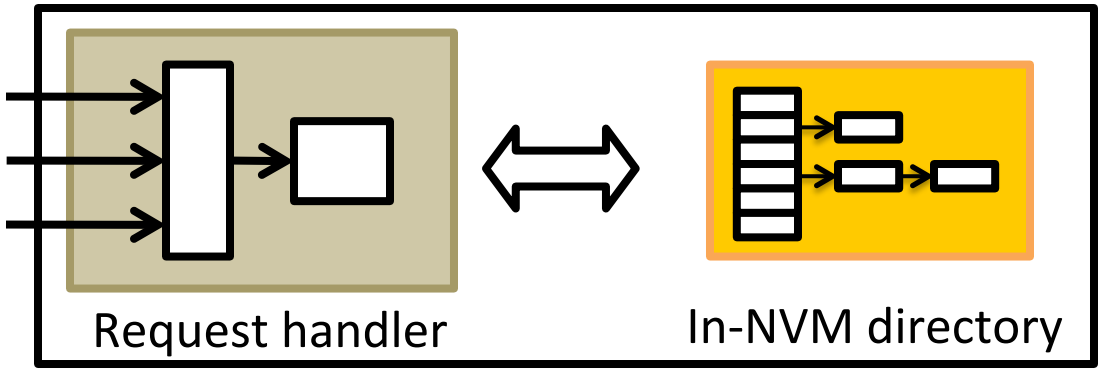
```
pstatic TCBDB* bdb;

pmalloc(&rec, sizeof(*nrec));

patomic {
 tcbdbput(bdb, &obj->key, ...)
}
```

OpenLDAP: Directory Service

- Original: stores dir-entries in Berkeley DB



- Modified: keeps dir-entries in persistent memory



Future Plans

- Port Mnemosyne to GCC-TM compiler and PMFS
- Form a benchmark suite
  - TokyoCabinet
  - OpenLDAP
  - memcached (?)
  - …