Zhuque: Failure is Not an Option, it’s an Exception

George Hodgkins*, Yi Xu*, Steven Swanson, Joseph Izraelevitz

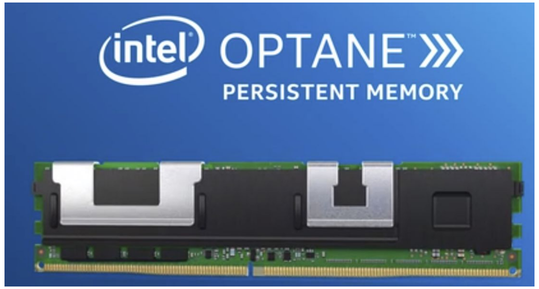
*co-first author

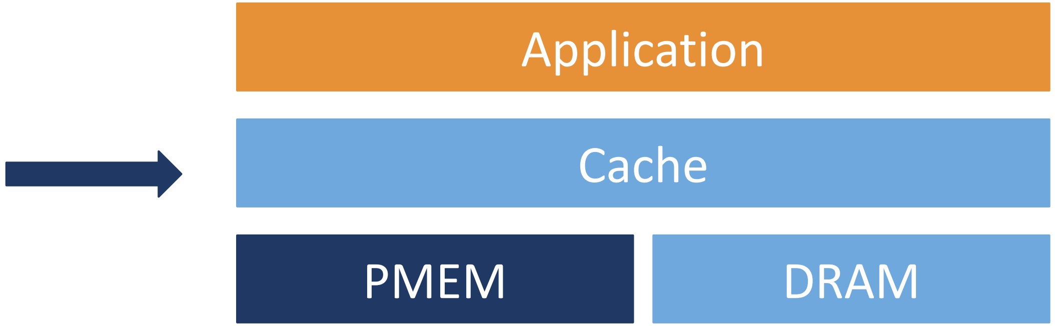
CU Boulder & UCSD

USENIX ATC ‘23

Presenter: George Hodgkins

Persistent Memories (PMEMs)

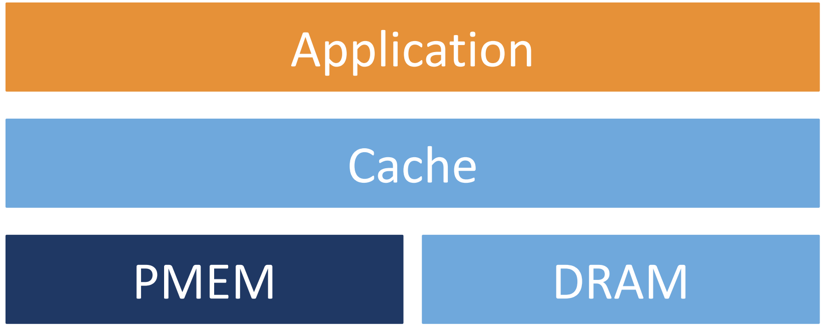




Persistent Memories (PMEMs)

PMEM

- Persistent memory is **byte-addressable**.
- **Persistent** over power failures.
- Delivers **DRAM-class latency/BW.**

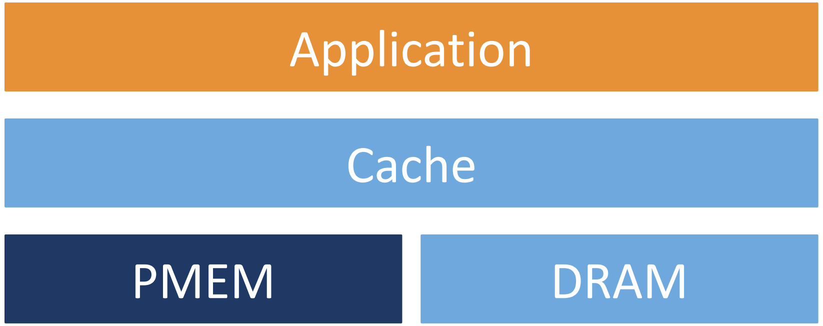


NVSL

Persistent Memories (PMEMs)

PMEM

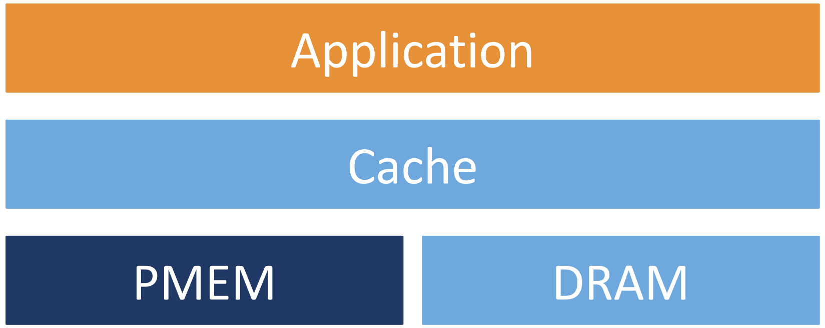
- Persistent memory is **byte-addressable**.
- **Persistent** over power failures.
- Delivers **DRAM-class latency/BW.**



Persistent Memories (PMEMs)

PMEM

- Persistent memory enables an application's in-memory data to live beyond its lifetime.

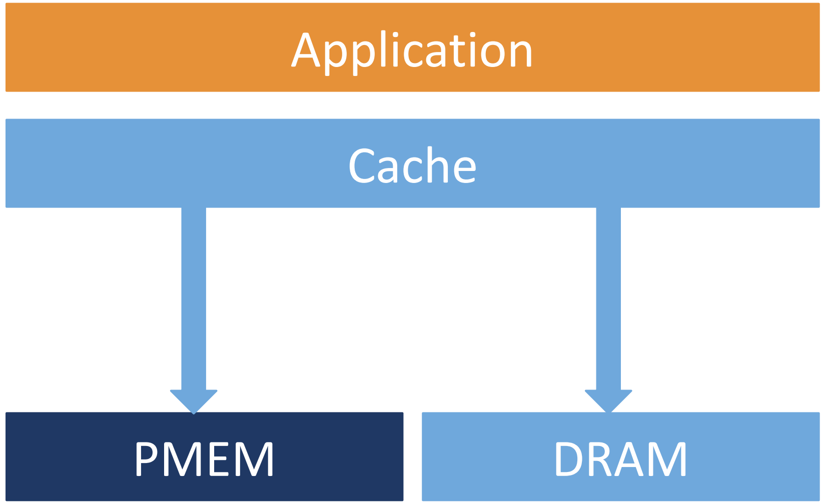


NVSL

The challenge

Cache

- The cache has been volatile.
- **Cached updates will be dropped** after a power loss.

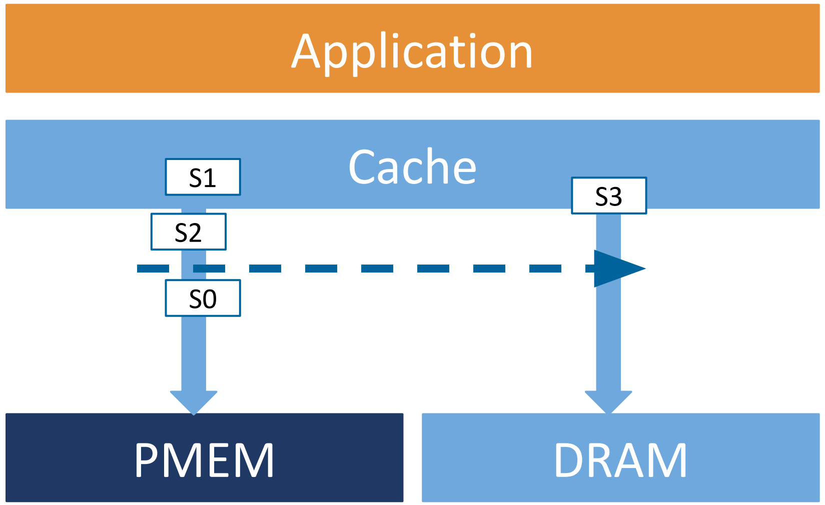


Applications need to explicitly evict cachelines to provide crash consistency.

The challenge

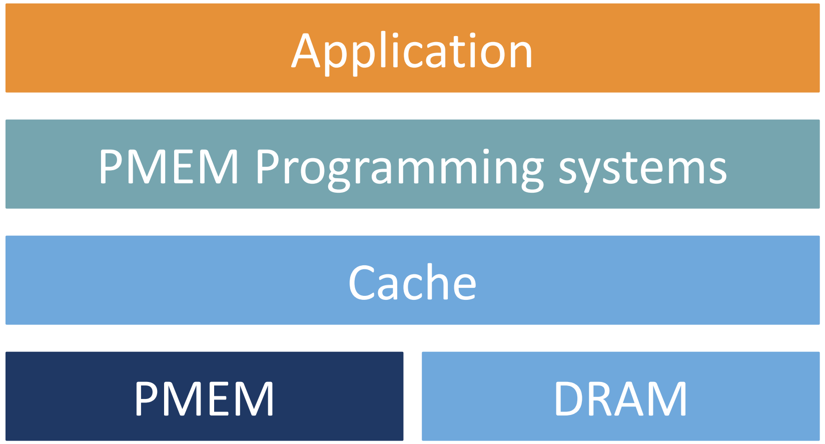
Cacheline eviction

- Evicted cachelines may not reach PMEM in a desired order.
- **Memory barrier enforce an ordering on memory operations.**



NVSL

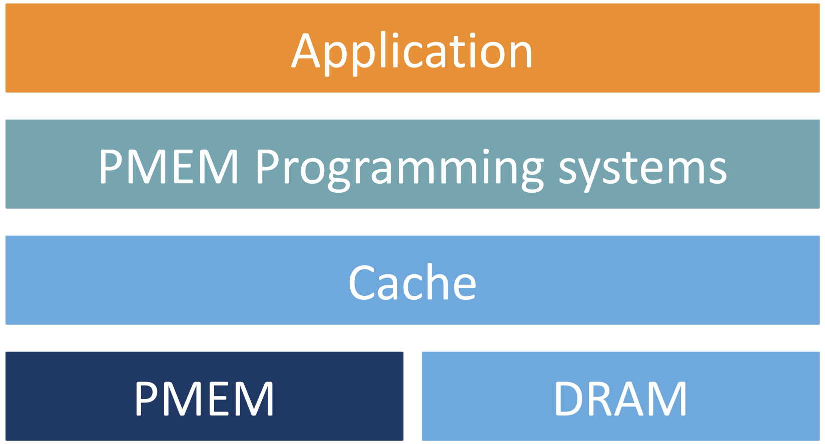
Persistent Memory Programming



PMEM Programming systems

- Libraries, programming models, language support, and compilers.
- Usually allow applications to **apply sets of writes to persistent memory atomically.**
- They usually provide the **interface of *"failure-atomic section"* and log.**
- They usually rely on cacheline eviction and memory barrier instructions.

Persistent Memory Programming



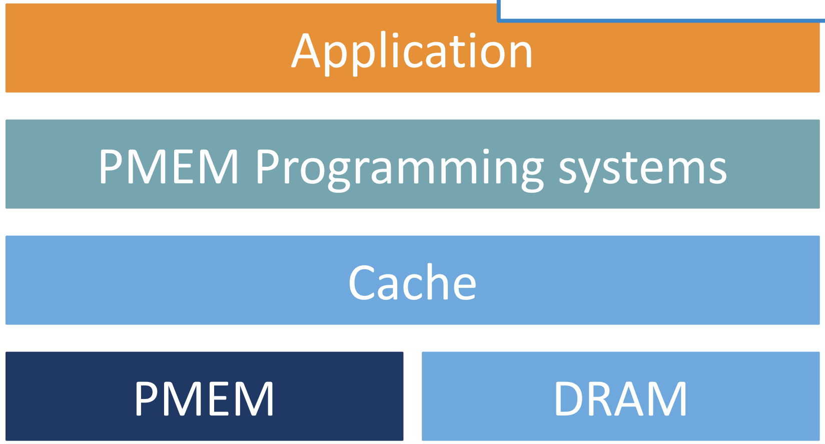
PMEM Programming systems

- Transaction-based.
- Failure-atomic sections (FASEs).
- Whole system persistence (WSP).

NVSL

Persistent Memory Programming

Failure-atomicity libraries: Allow applications to **apply sets of writes to persistent memory atomically.**



PMEM Programming systems

- Transaction-based.
- Failure-atomic sections (FASEs).
- Whole system persistence (WSP).

Makes everything persistent. From the program's perspective, **crashes never occur.**

Transaction-based Libraries

PMEM program with undo log based transaction-based library

```
void list_push(list_t *list,char* val){
    TX_BEGIN{
    int val_len = strlen(val);
    log(list->buf[list->size], val_len);
    memcpy(list->buf[list->size], val, val_len);
    log(list->size, sizeof(size_t));
    list->size++;
    }TX_END
}
```

Programmers explicitly mark failure atomic transactions.

Traditional DRAM code

It is necessary to log extra information during normal execution to support recovery after a failure.

Once the effects of the code region are guaranteed to survive a crash, the operation is **committed.**

NVSL

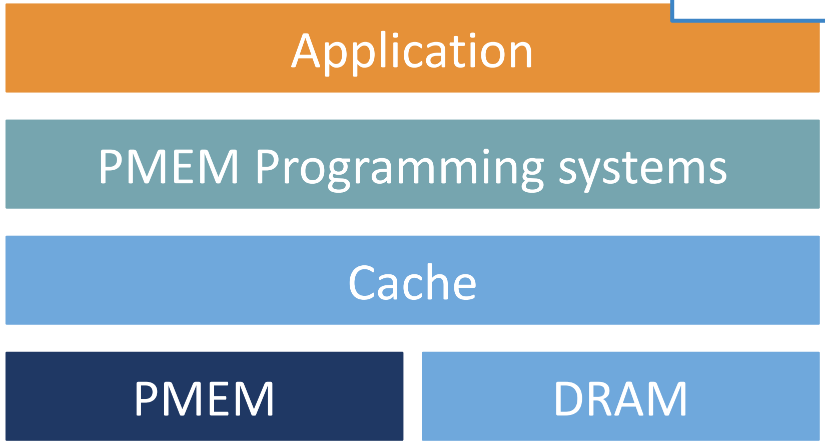
Transaction-based Libraries - Concurrency Control

PMEM program with undo log based transaction-based library

```
void list_push(list_t *list,char* val){
    TX_BEGIN{
    lock(list);
    int val_len = strlen(val);
    log(list->buf[list->size], val_len);
    memcpy(list->buf[list->size], val, val_len);
    log(list->size, sizeof(size_t));
    list->size++;
    unlock(list);
    }TX_END
}
```

Expects programmers to acquire and release locks in a *conservative, strong strict two-phase locking* pattern.

Persistent Memory Programming



Fundamentally incompatible with existing legacy multithreaded code. Low performance.

PMEM Programming systems

- Transaction-based.
- Failure-atomic sections (FASEs).
- Whole system persistence (WSP).

FASE-based Libraries

PMEM program with undo log based FASE-based library

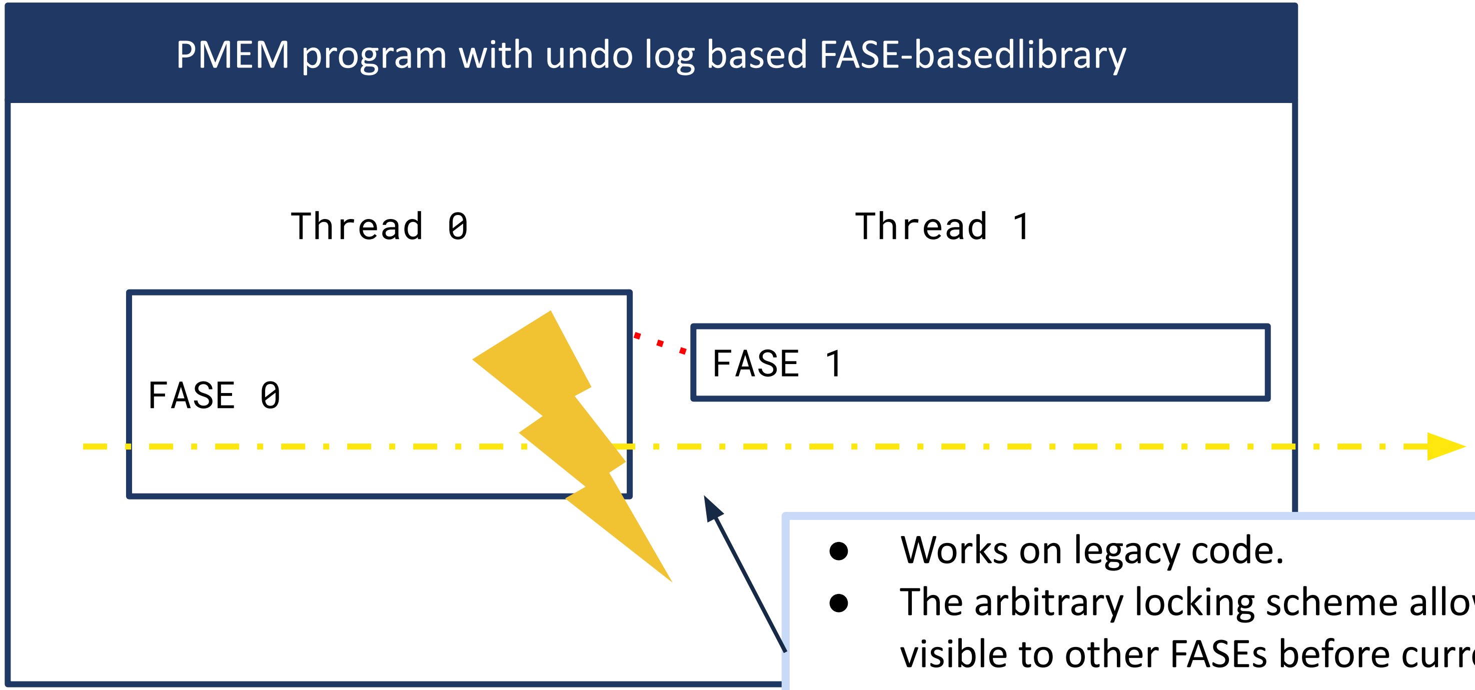
```
void list_push(list_t *list,char* val){
     int val_len = strlen(val);
     lock(list->buf[list->size]);
     log(list->buf[list->size], val_len);
     memcpy(list->buf[list->size].val, val, val_len)
     lock(list);
     unlock(list->buf[list->size]);
     log(list->size, sizeof(size_t));
     list->size++;
     unlock(list);
}
```

Traditional DRAM code

It is necessary to log extra information during normal execution to support recovery after a failure.

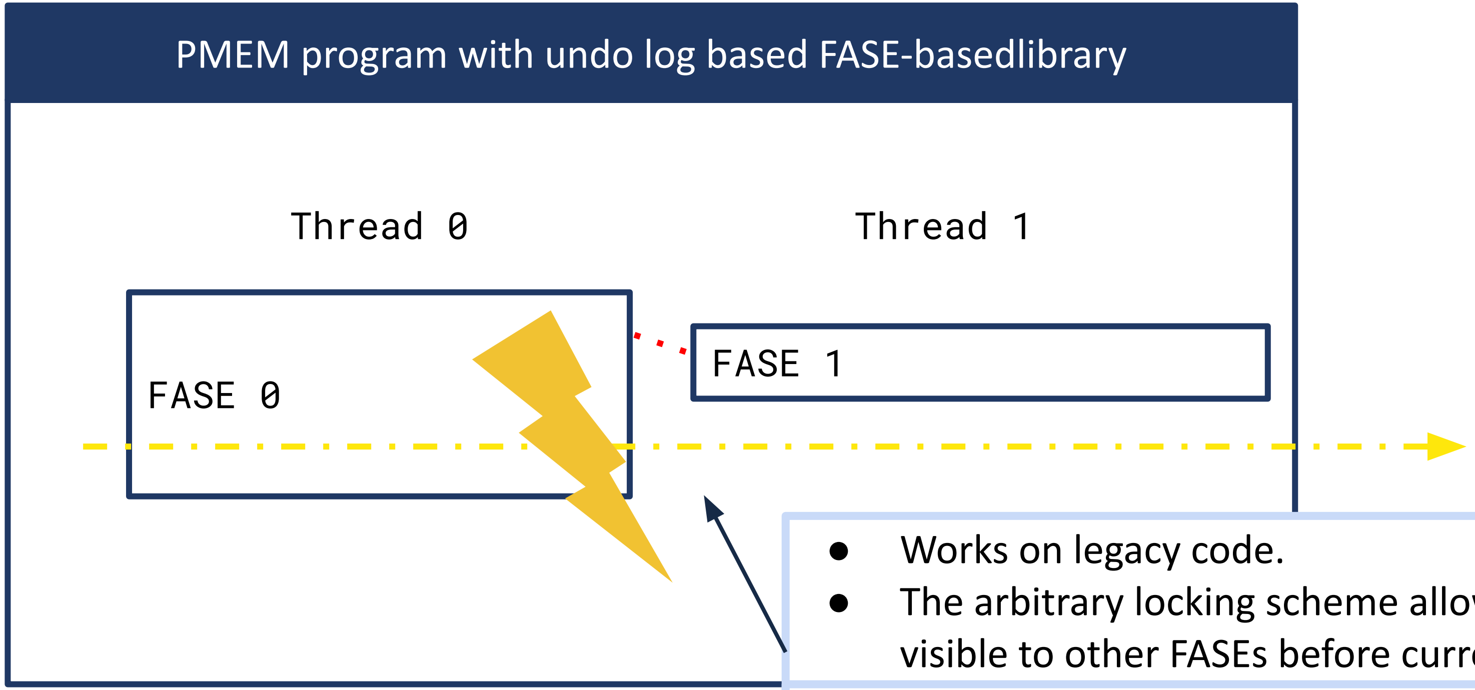
Allows arbitrary locking scheme.
A FASE is a failure-atomic operation protected by its outermost locks.

FASE-based Libraries



- Works on legacy code.
- The arbitrary locking scheme allows updates to be visible to other FASEs before current FASE commit.
- Need to track dependency between threads, and rollback dependent FASEs in case of failure.

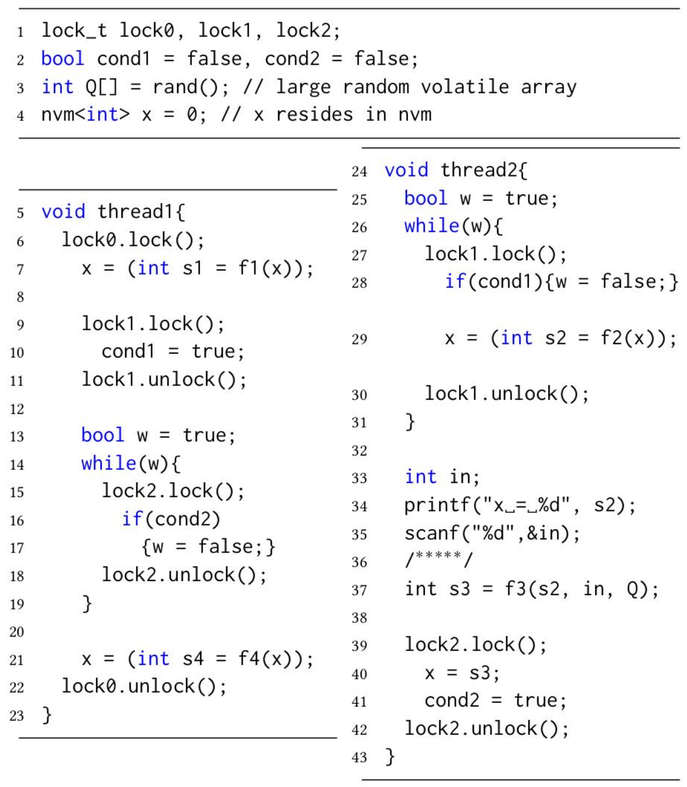
FASE-based Libraries



- Works on legacy code.
- The arbitrary locking scheme allows updates to be visible to other FASEs before current FASE commit.

- **Could be slow**

FASE-based library



```
lock_t lock0, lock1, lock2;
bool cond1 = false, cond2 = false;
int Q[] = rand(); // large random volatile array
nvm<int> x = 0; // x resides in nvm
```

```
void thread1{
  lock0.lock();
    x = (int s1 = f1(x));

    lock1.lock();
      cond1 = true;
    lock1.unlock();

    bool w = true;
    while(w){
      lock2.lock();
        if(cond2)
          {w = false;}
      lock2.unlock();
    }

    x = (int s4 = f4(x));
  lock0.unlock();
}
```

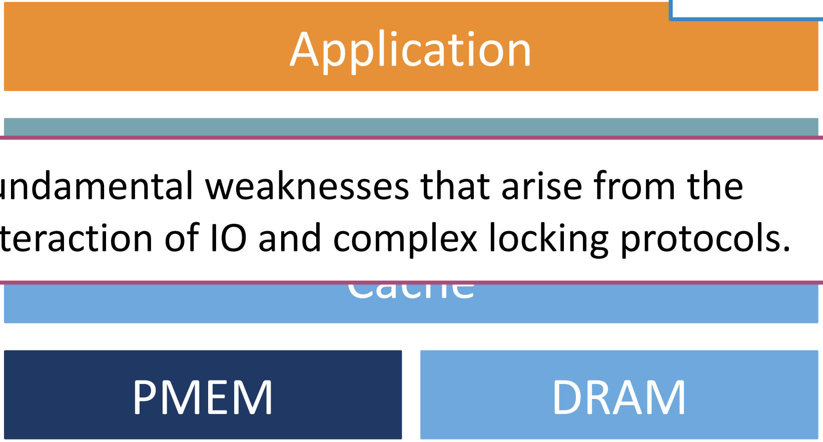
```
void thread2{
  bool w = true;
  while(w){
    lock1.lock();
      if(cond1){w = false;}

      x = (int s2 = f2(x));

    lock1.unlock();
  }

  int in;
  printf("x␣=␣%d", s2);
  scanf("%d",&in);
  /*****/
  int s3 = f3(s2, in, Q);

  lock2.lock();
    x = s3;
    cond2 = true;
  lock2.unlock();
}
```

FASE-based library

- **Is not general enough for some code patterns.**
- Need to persist all volatile states if they want to be general enough to support this example.

Persistent Memory Programming

Fundamentally incompatible with existing legacy multithreaded code.



Fundamental weaknesses that arise from the interaction of IO and complex locking protocols.

PMEM Programming systems

- Transaction-based.
- Failure-atomic sections (FASEs).
- Whole system persistence (WSP).

NVSL

Whole system persistence

PMEM program with whole system persistence

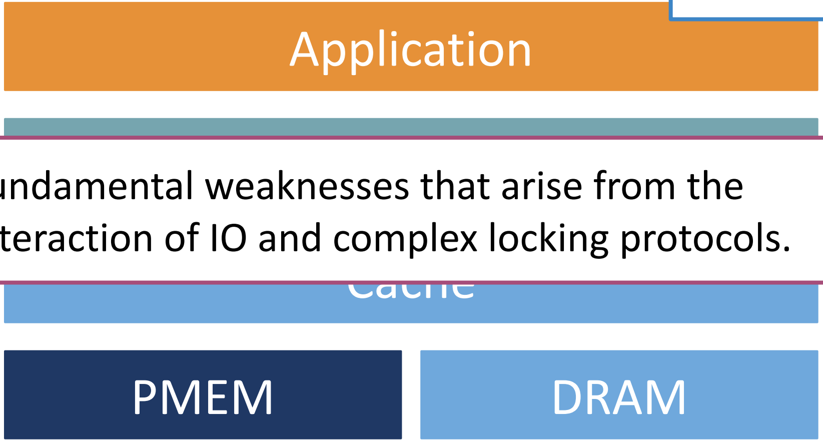
```
void list_push(list_t *list,char* val){
    int val_len = strlen(val);
    memcpy(list->buf[list->size], val,
val_len);
    list->size++;
}
```

Whole system persistence

- Making all of memory persistent has been infeasible because caches has been volatile.
- The benefits of making everything persistent rather than a subset of system state are not justified.

Persistent Memory Programming

Fundamentally incompatible with existing legacy multithreaded code.



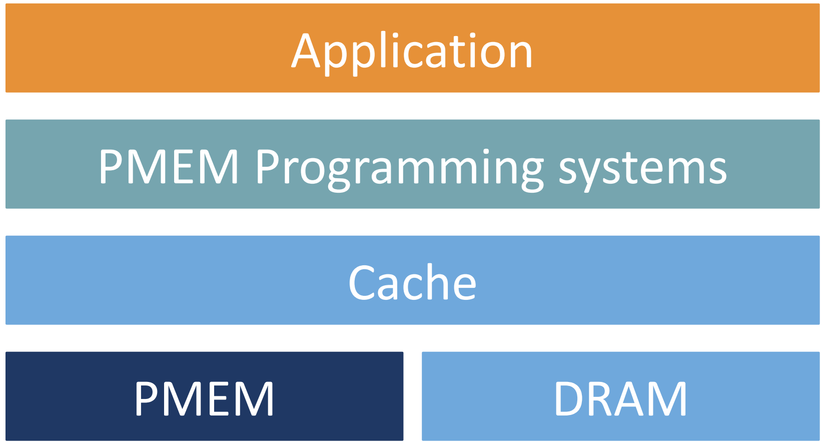
Fundamental weaknesses that arise from the interaction of IO and complex locking protocols.

PMEM Programming systems

- Transaction-based.
- Failure-atomic sections (FASEs).
- Whole system persistence (WSP).

High performance overhead. And the benefits of making everything persistent are not justified.

Persistent Memory Programming



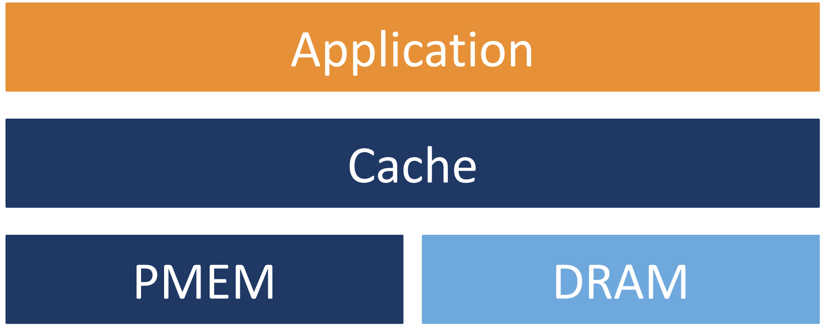
PMEM Programming systems

- High performance overhead.
- Hard to use.

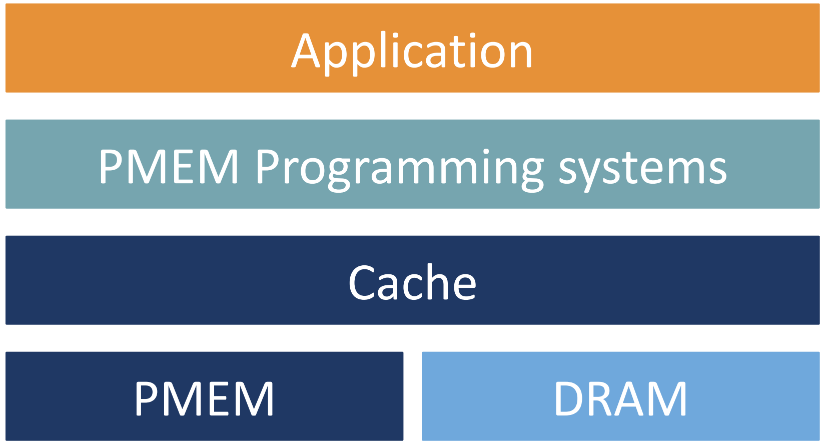
Extended ADR (eADR)

eADR

- eADR ensures that all writes that reach the cache will be written to PMEM in the event of a power outage.
- **Caches are effectively persistent.**



Ideal Persistent Memory Programming Model

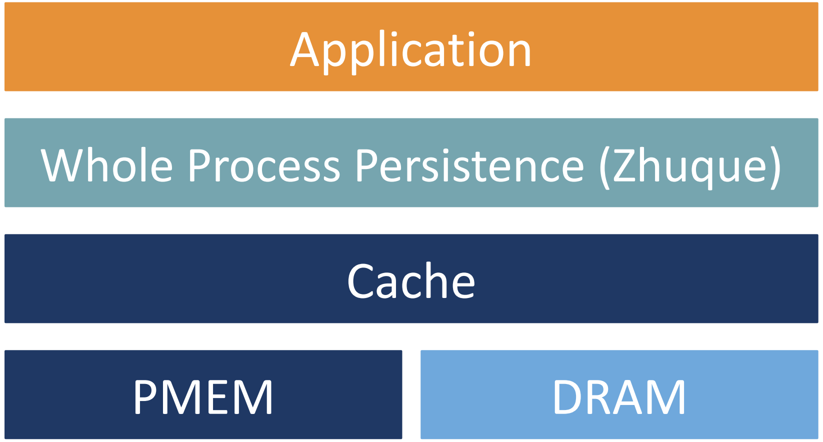


PMEM Programming systems

- Fast.
- Flexible enough to legacy programs, easy to use.

NVSL

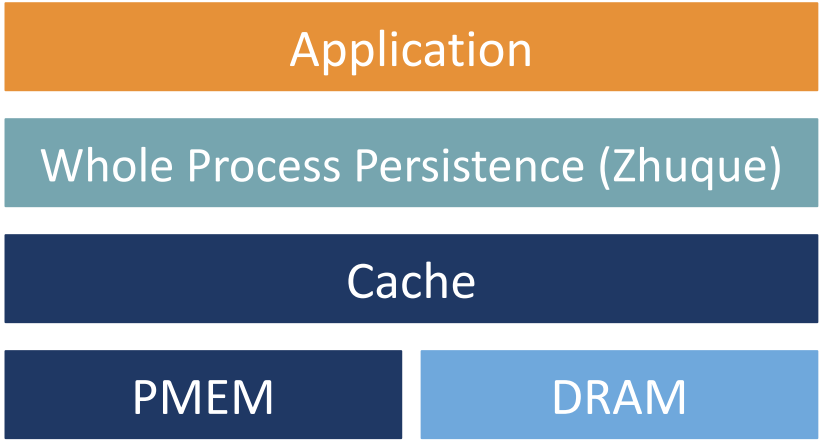
Whole Process Persistence



From the application's perspective, power failure is delivered as an asynchronous signal (recoverable exception).

NVSL

Whole Process Persistence



Whole Process Persistence

- High performance
  - Persistent cache.
  - Limit the scope of persistence to a process (instead of whole system).
- Easy to use.
  - Can run unmodified applications directly on **Zhuque --- musl-based implementation of WPP.**

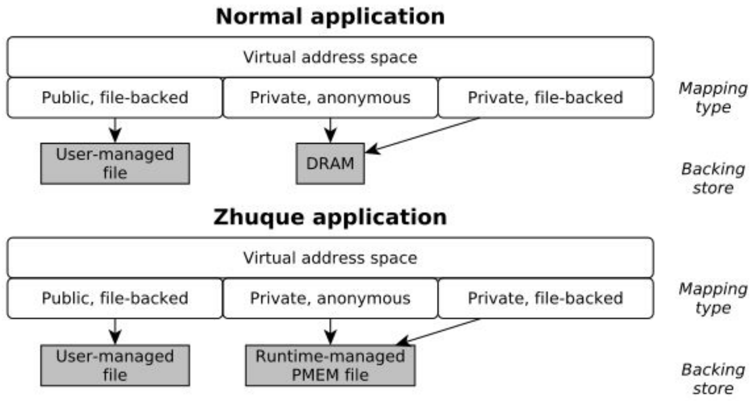
From the application's perspective, power failure is delivered as an asynchronous signal (recoverable exception).

NVSL

During normal execution

Whole Process Persistence

- Run unmodified ELF binaries linked to Zhuque
- Zhuque ensures that all program memory (stack, heap, etc.) resides in persistent memory.

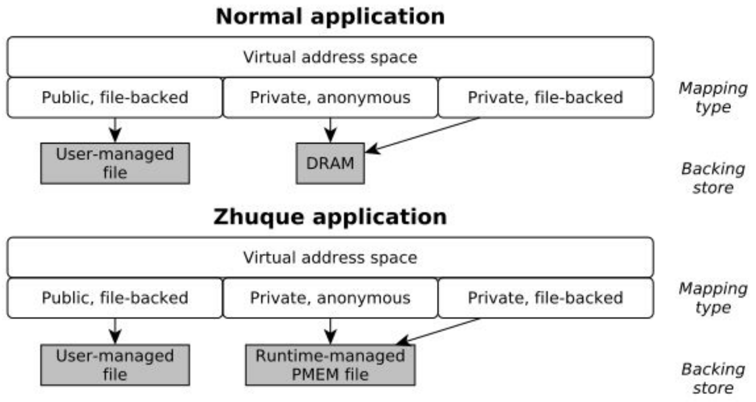


NVSL

During normal execution

Zhuque

- Dynamic memory: return PMEM from `sbrk()` and `mmap()`.
- (Initialized) static memory: transform private, writable file mappings to PMEM.
- Save architectural state to PMEM on kernel entry.



NVSL

At crash

Whole Process Persistence

- When the power failure interrupt is delivered to a thread, it saves its architectural state in a preallocated region:
  - general-purpose registers
  - floating-point unit state
  - vector unit state
- The program receives a normal operating system signal when restarted (e.g. SIGPWR)
- We believe this is supported by the architecture, but firmware is closed to modification…

At recovery

Whole Process Persistence

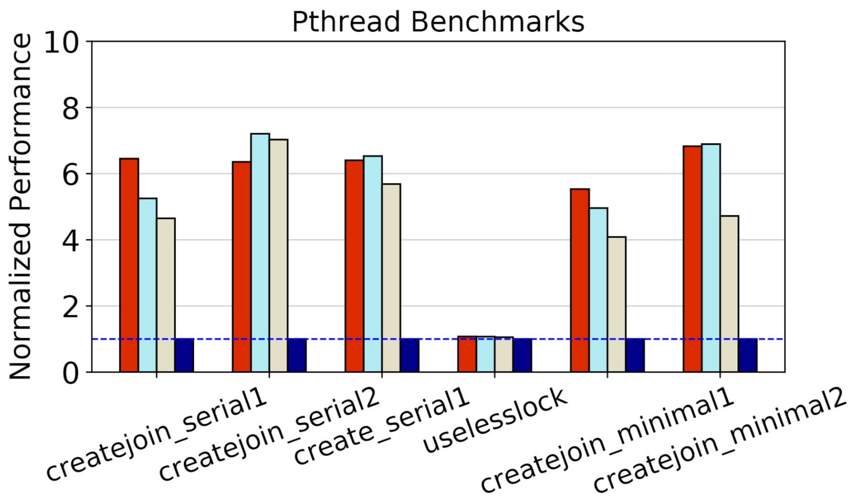
1. Restore application address space: restore the virtual memory mappings.
2. Restore system-specific states: In Zhuque, we track the state of threads and file descriptors and restore them at restart.
3. Restore the architectural state (including stack pointer and program counter).

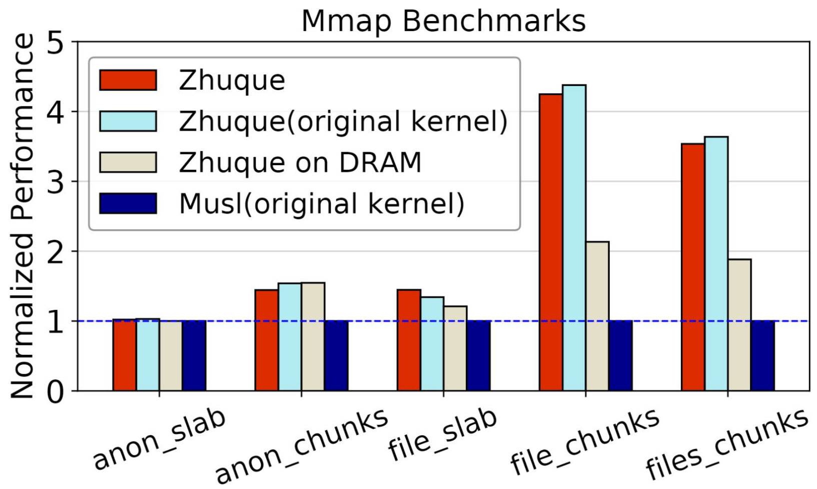
4. Run the application-defined power failure handler, if it exists.
5. Execution of the thread continues at the point where the failure interrupted it.

Zhuque -- Requirement to applications

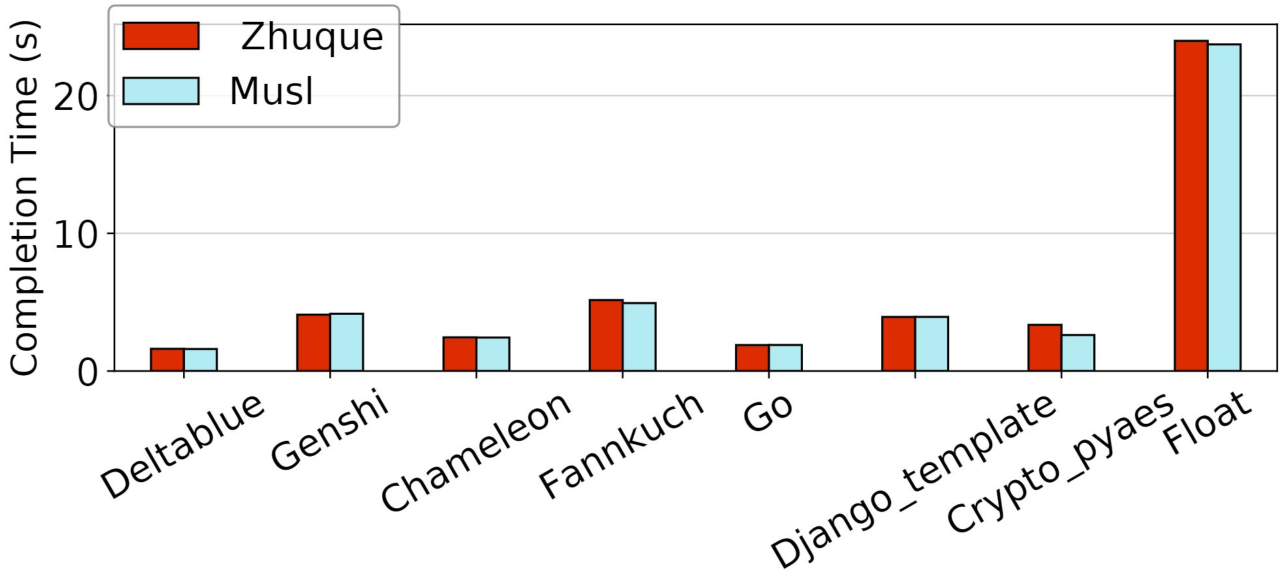
- Threading and virtual memory must be managed using the POSIX-specified APIs.
- Applications must check error returns from system calls and other POSIX APIs.

Performance - microbenchmarks



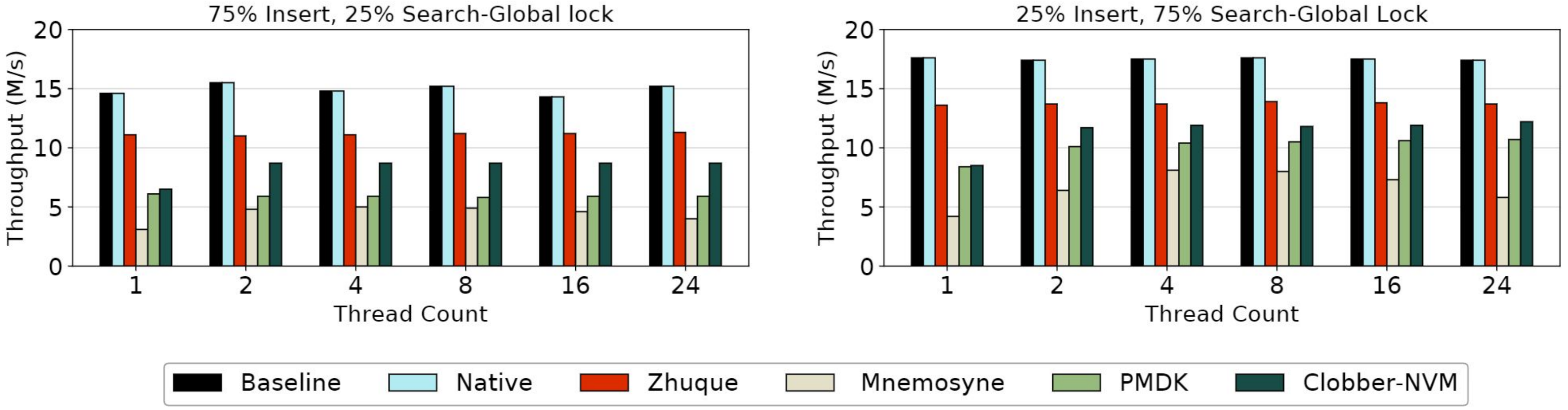


Performance - python benchmarks



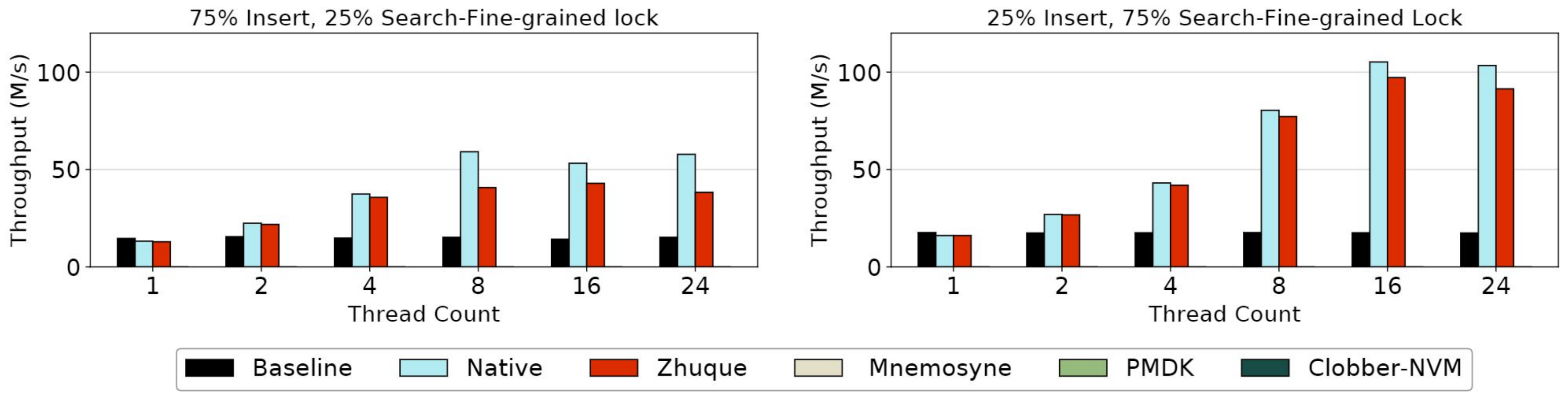
NVSL

Performance - memcached 1.2.5



NVSL

Performance - memcached 1.6.10



Zhuque: Failure is Not an Option, it's an Exception

Thank you!

NVSL

Engineering

UCSDCSE
Computer Science and Engineering