Cohen, Aksun, Larus. **Object-Oriented Recovery for Non-Volatile Memory**. OOPSLA 2018.

Object-Oriented Recovery for Non-volatile Memory

Nachshon Cohen, David Aksun, **James Larus**

EPFL

10th Annual Non-volatile Memories Workshop

San Diego, CA March 12, 2019

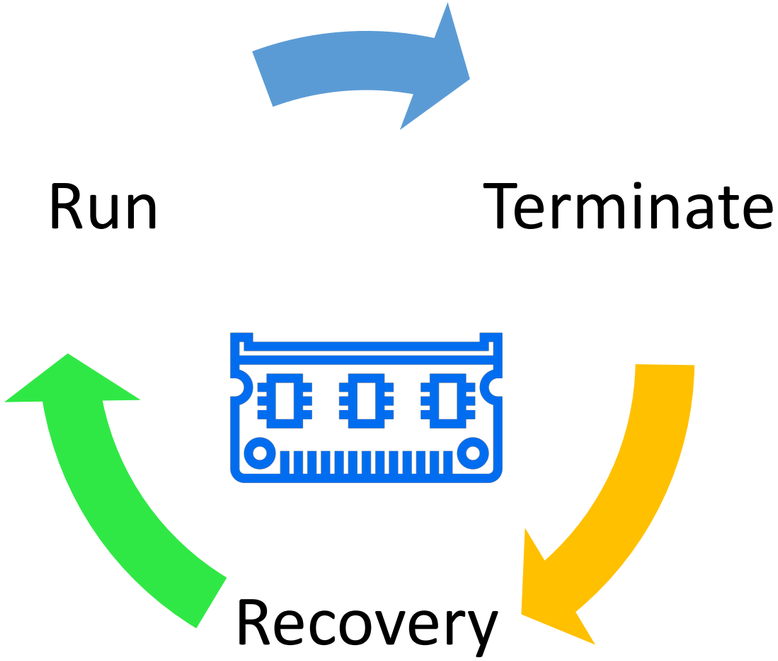
Overview

- Prior NVM recovery mechanisms are incomplete
  - Your carefully stored, consistent data may be unusable

- Object-oriented recovery
  - llvm extension to support complete recovery

NVM Lifecycle

1. Code accesses NVM with load and store instructions



2. NVM must record a consistent memory state before termination, planned or unexpected

3. Ensure NVM state is consistent in **the environment in which execution restarts**

Recovery Problems

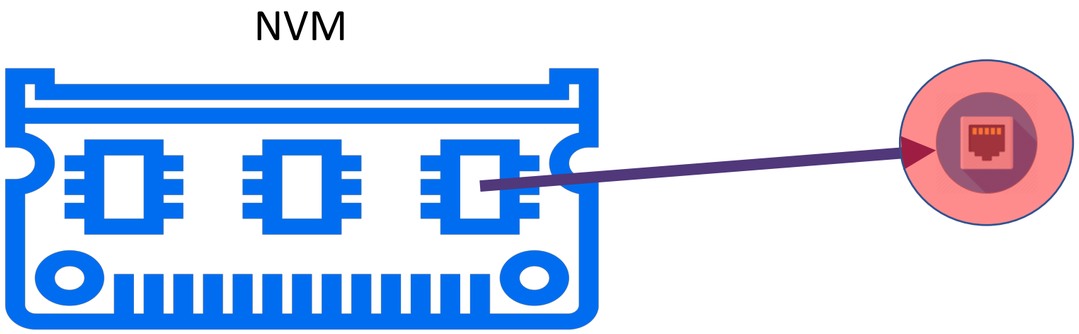
1. Non-persistent data
2. NVM remapping
3. Code remapping

1. Non-Persistent Data in NVM

Network socket referenced from NVM is valid in current environment

Environment Can Change on Restart

Network socket is no longer usable



Environmentally-Specific Data

- Network sockets
- Locks
- Process and thread IDs
- File handles
- …

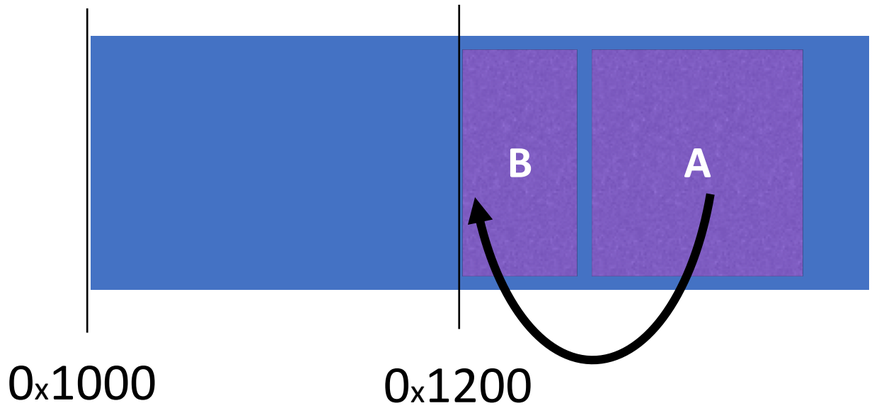
Lesson 8: Initialization of semantically nonpersistent data colocated with persistent data is tricky. Programmers frequently find it convenient to co-locate nonpersistent data in persistent objects.

--- *Persistent Memcached: Bringing Legacy Code to Byte-Addressable Persistent Memory*, HotStorage '17.

- Common practice to store [pointers to] these objects in NVM
  - Fast access
- Must restore / reinitialize during recovery
  - Traverse all objects in NVM (= GC)

2. NVM (Re)Mapping

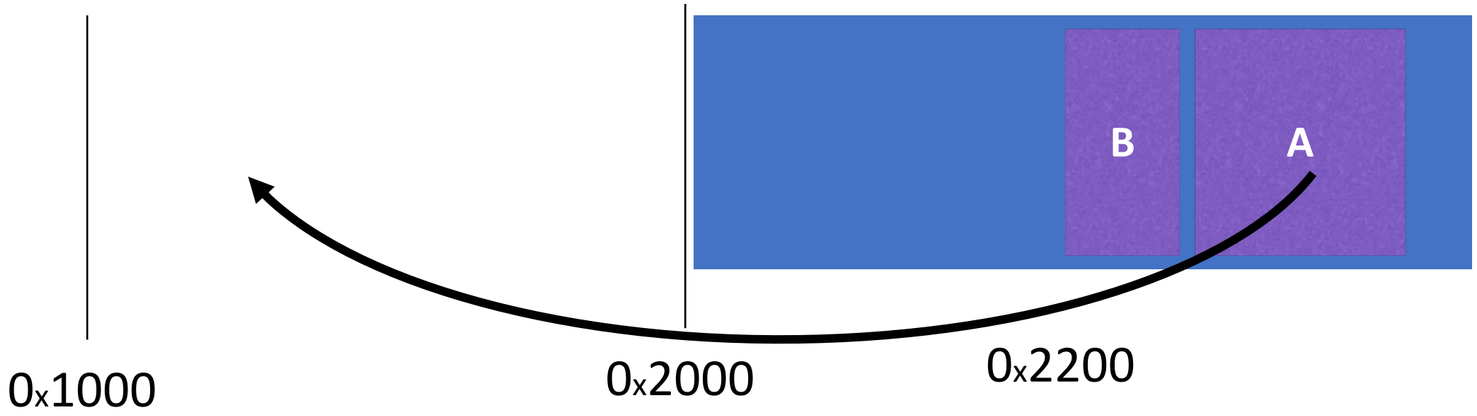
```
base = mmap(0x1000, …, nvm_fd);
```



Remapped To Different Address

```
base = mmap(0x1000, …, nvm_fd);
```

But, kernel may mmap to a different address



mmap

> "If *addr* is not NULL, then the kernel takes it as a **hint** about where to place the mapping..
>
> -- MMAP(2) man page

- Always map to specified virtual address? **NO**
  - OS upgrade
  - NVM grows/shrinks
  - Execution under debugger/profiler/etc.
  - Earlier actions during recovery
  - Mapping in several NVM segment
  - …

3. Code and Literal Pointers

- Function pointers and virtual pointers are also execution specific
  - Address Space Layout Randomization (ASLR)
- C++ objects contain method pointers

- Object may not be well formed after restart

Published Solutions

Data is not durable
if it cannot survive system changes

- Forbid NVM to DRAM pointers [ASPLOS'11]
  - Impractical in real systems [HotStorage'17]

- Ad-hoc, specific solutions
  - Generational locks [ASPLOS'11]
  - Self-relative pointers [NVML, NVM-Direct]
  - Comment code (and hope someone reads it)
  - Custom (re)initialization code

```
111     // The following are considered logically transient
112     // They must be reset at init time.
113     pthread_mutex_t Lock_;
114     FreeList *FreeList_;
```

NVM Reconstruction

- Compiler support for object-level recovery
- Recovery procedure for each object in NVM
- Ensures that object is well-formed after recovery
- Transparent to application

llvm Language Extension

Standard pointer
(no relative addresses)

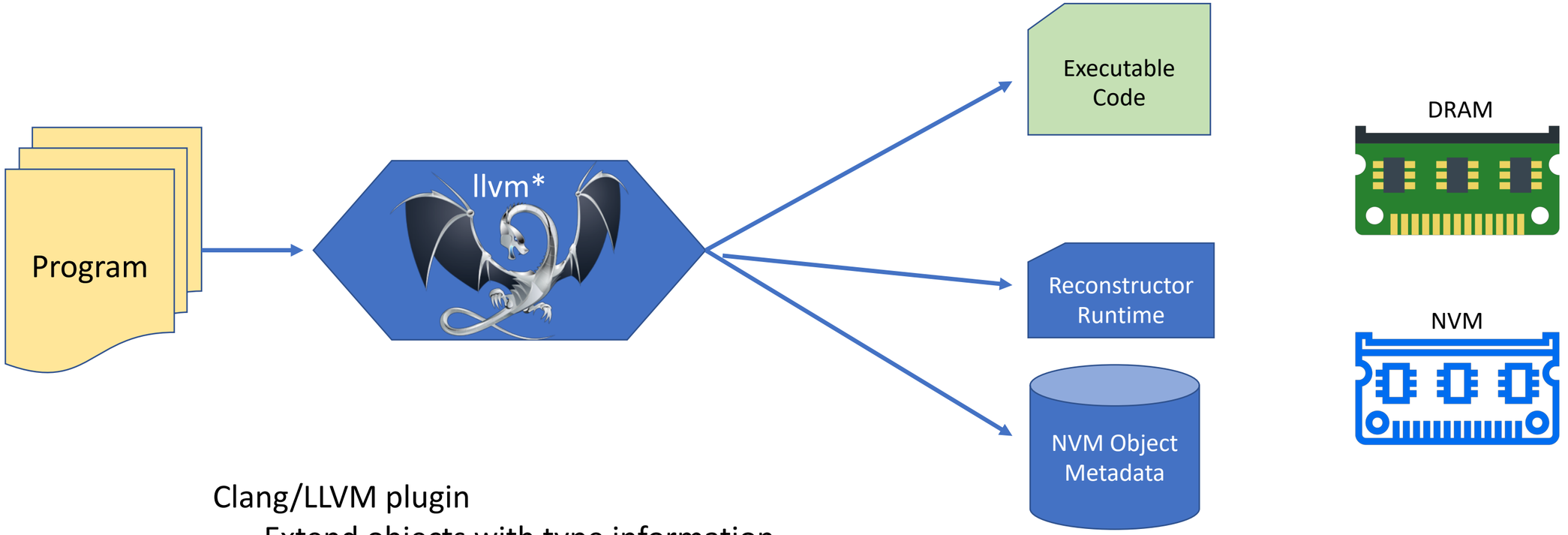
```
struct …{
  void *CurrAllocAddr_;
  transient pthread_mutex_t lock;
  reconstructor(node *n){
          pthread_mutex_init(&n->lock);
     }
    void addChild(long k){
          left = pnew node(k);
```

Zero on restart

Custom initialization code

Allocate in NVM

NVM Reconstruction Workflow



Clang/LLVM plugin

- Extend objects with type information
- Collect metadata

Runtime

- Records runtime information, e.g., mapping address
- Allocates header for each durable object

Reconstruction After Failure

During recovery

- Use type information from previous execution
- Compute address space delta per page

For each live object:

- Zero transient fields
- Rebase NVM pointers
- Fix code pointers
- Invoke user-provided reconstructor

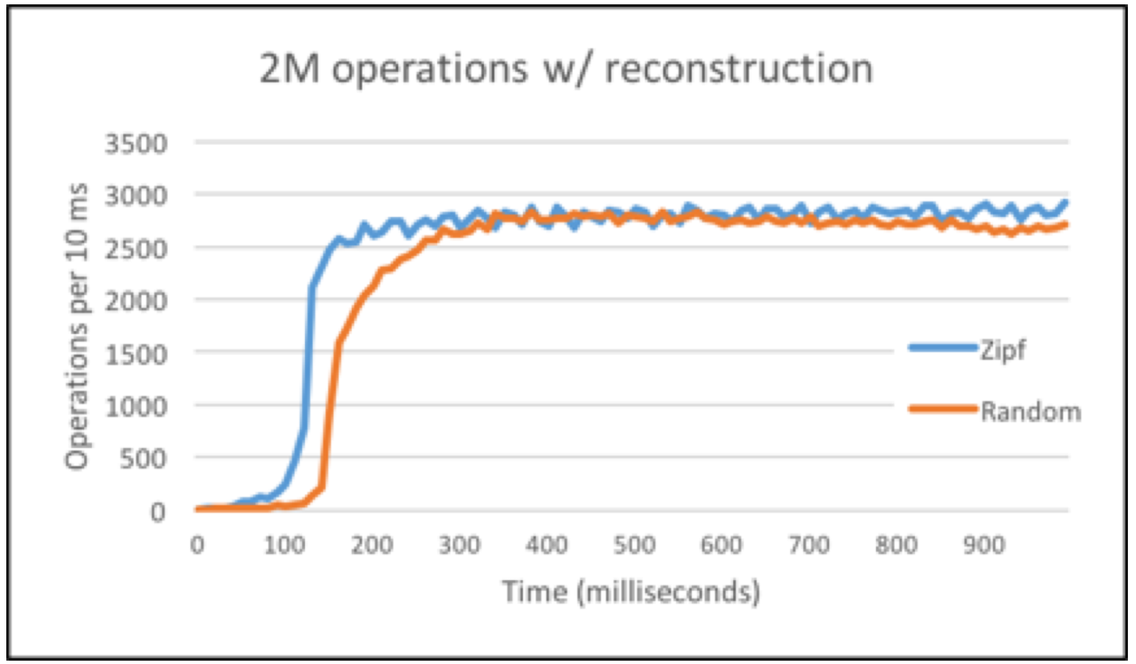
Lazy Reconstruction

```
Initially: memory protect NVM region

On page-fault:
    For each live object in page
    Apply system reconstruction
            Zero transient fields
            Fix NVM pointers, code pointers
        Apply User-provided reconstruction
```

Performance Measurements: Atlas

- Applied NVM-Reconstruction to Atlas [Chakrabarti 2014]
  - Support for transient fields, different mapping addresses, etc.
  - Negligible runtime cost
- Measured simple Key-Value Store
  - Recovery time: up to 200ms/GB, depends on number of items

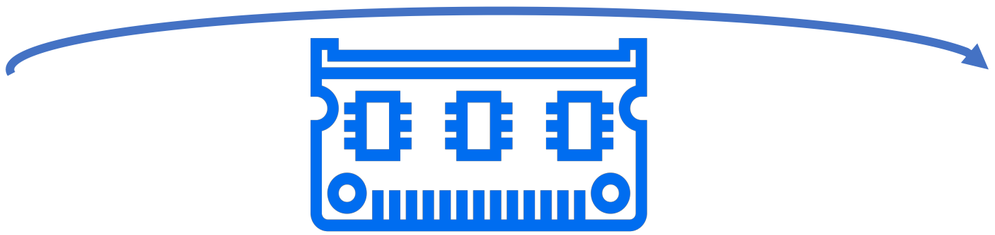


Code Change Measurements: Echo KV Store

- Incorporated NVM-Reconstruction into Echo Key-Value Store
  - Original code: 22,503 SLOC, no recovery
  - NVMReconstructor: added 214 SLOC, full recovery

	pnew	pdelete	realloc extra	transient	reconstructor	total
Added SLOC	38	68	25	19	64	**214**

Reconstruction Test



Environment:
1. gcc –O3
2. Original classes
3. mmap @ 2^{40}

Environment:
1. gcc –O0
2. Add field to each class
3. mmap @ 3 x 2^{40}

Conclusions

- Execution environment may differ after restart
  - Need to recover execution-specific data and adjust for environment changes

- NVM-Reconstruction: system-level approach for object-level recovery
  - Transient fields, virtual address pointers, custom reconstructor functions
  - Low overhead
  - Easy to use