Clobber-NVM: Log Less, Re-execute More

Yi Xu, Joseph Izraelevitz, Steven Swanson

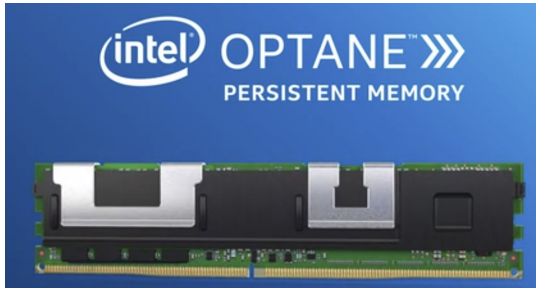
UC San Diego & University of Colorado, Boulder

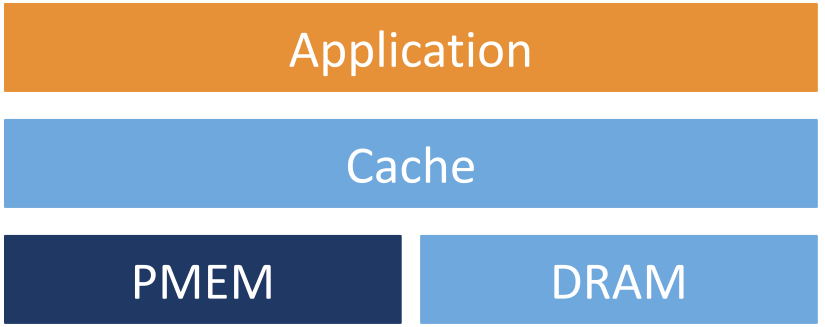
Published on ASPLOS 2021

NVSL

UCSDCSE Computer Science and Engineering

Persistent Memory Programming



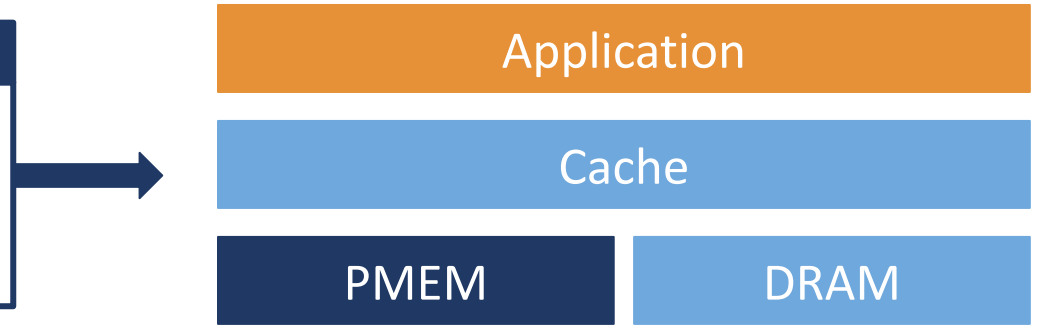


NVSL

Persistent Memory Programming

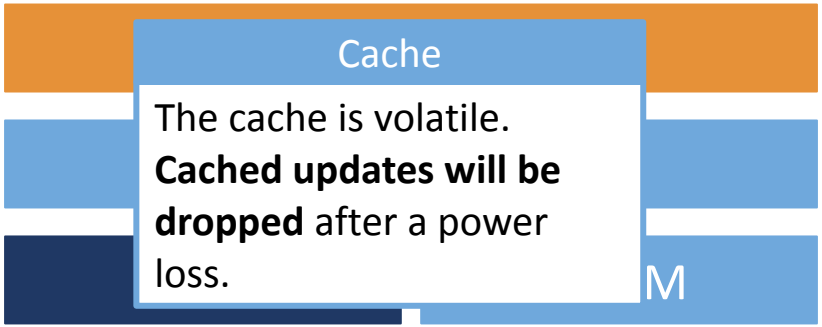
PMEM

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



NVSL

Persistent Memory Programming

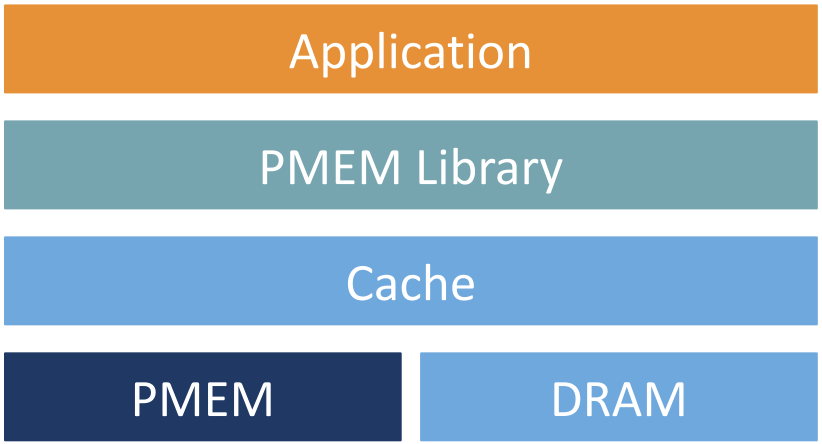


NVSL

Persistent Memory Programming

NVSL

Persistent Memory Programming



PMEM library

- PMEM libraries provide the means to **apply sets of writes to persistent memory atomically.**
- Unfortunately, most current libraries impose **significant overhead.**

NVSL

PMEM Program with Undo Logging

DRAM program

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

PMEM program with undo logging

```
void list_push(list_t *list,char* value){
    undo_log(value, strlen(value));
    persist_barrier();
    memcpy(list->buf[list->size], value, strlen(value);
    undo_log(list->size, sizeof(size_t));
    persist_barrier();
    list->size++;
}
```

NVSL

PMEM Program with Undo Logging

DRAM program

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

PMEM program with undo logging

```
void list_push(list_t *list,char* value){
    undo_log(value, strlen(value));
    persist_barrier();
    memcpy(list->buf[list->size], value, strlen(value);
    undo_log(list->size, sizeof(size_t));
    persist_barrier();
    list->size++;
}
```

NVSL

PMEM Program with Undo Logging

DRAM program

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

PMEM program with undo logging

```
void list_push(list_t *list,char* value){
    undo_log(value, strlen(value));
    persist_barrier();
    memcpy(list->buf[list->size], value, strlen(value);
    undo_log(list->size, sizeof(size_t));
    persist_barrier();
    list->size++;
}
```

NVSL

PMEM Program with Undo Logging

DRAM program

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

PMEM program with undo logging

```
void list_push(list_t *list,char* value){
    undo_log(...);
    persist_...
    memcpy(...                                  len(value);
    undo_log(...);
    persist_barrier();
    list->size++;
}
```

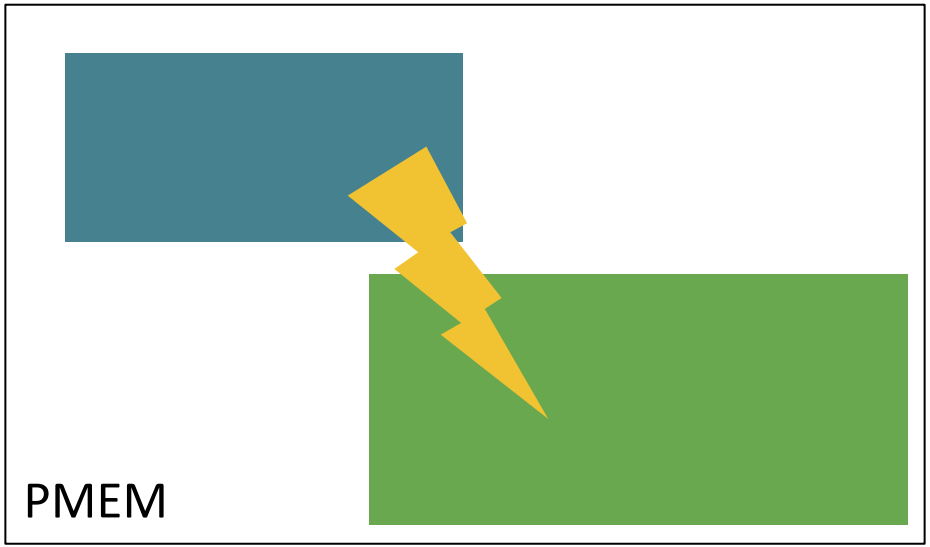
Are the logs and barriers necessary?

NVSL

Recover Through Re-execution

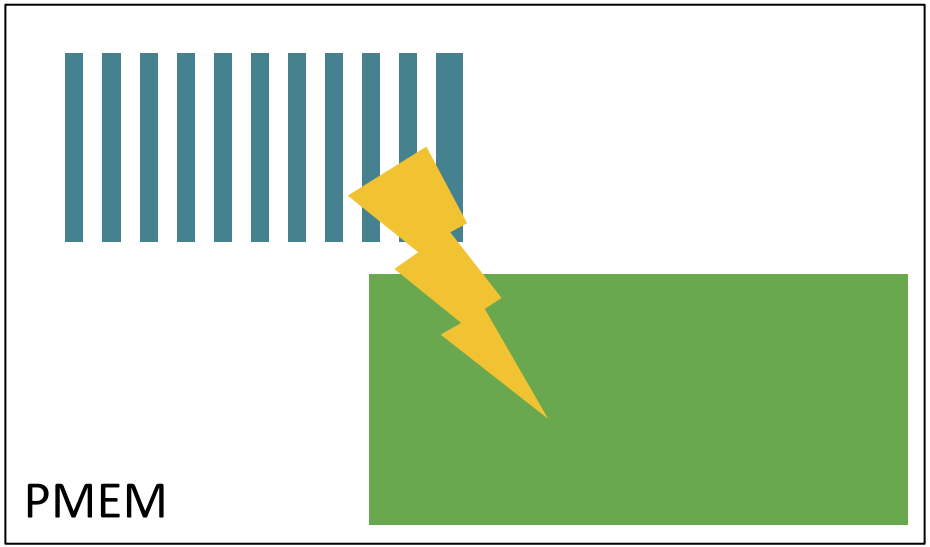
NVSL

Recover Through Re-execution



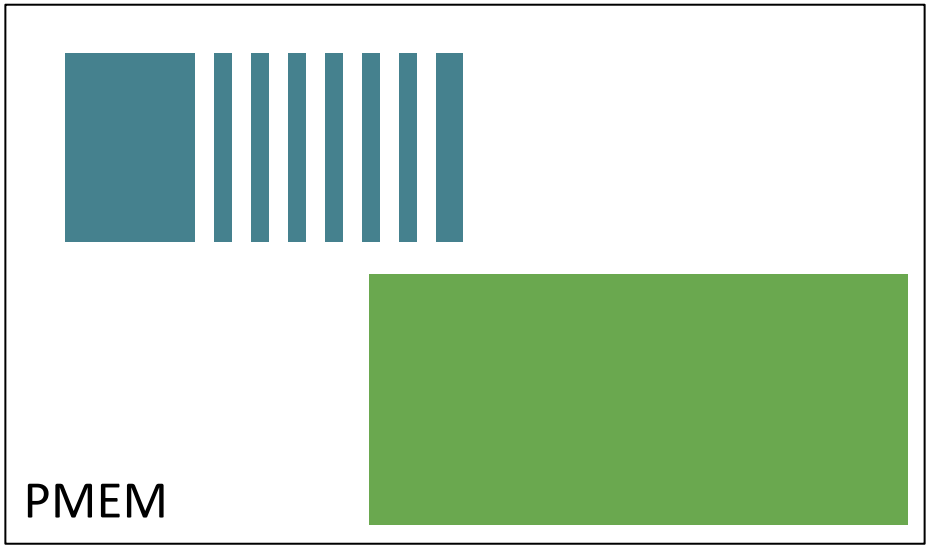
NVSL

Recover Through Re-execution



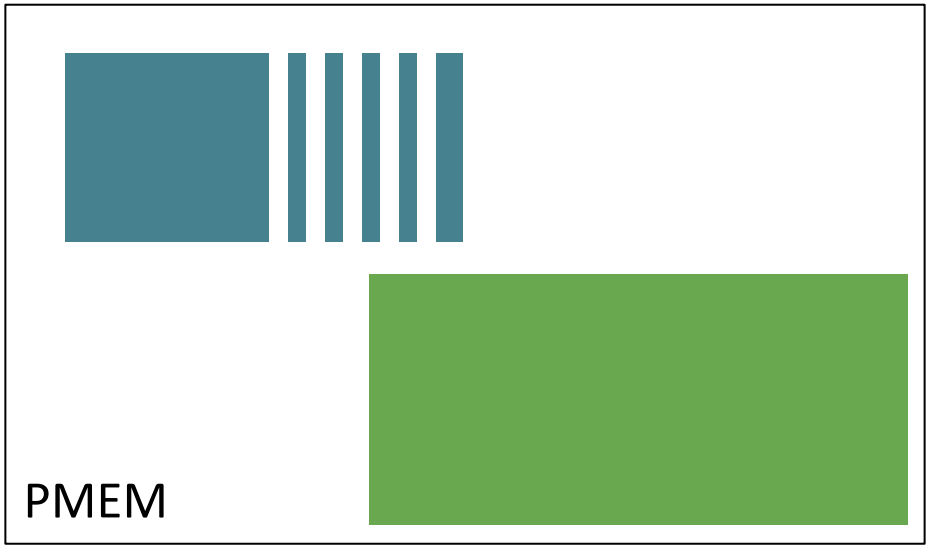
NVSL

Recover Through Re-execution



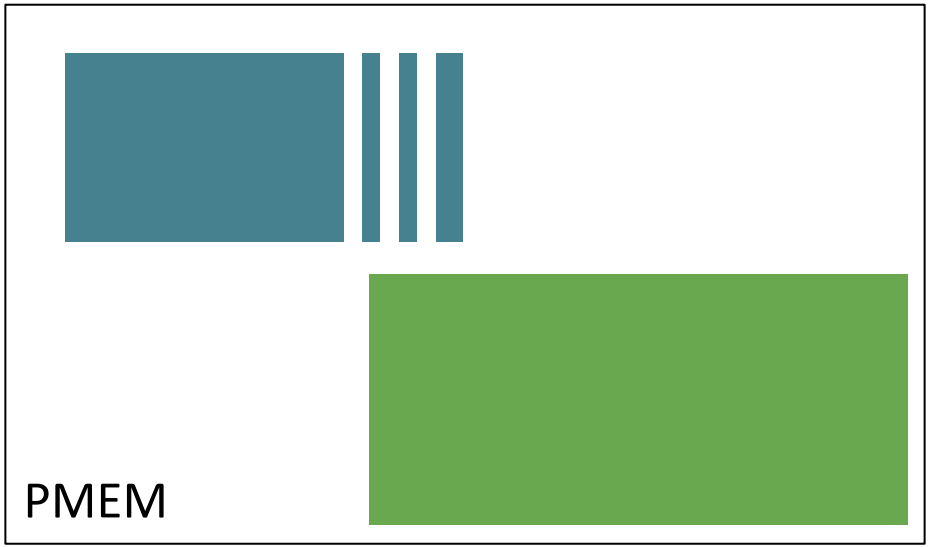
NVSL

Recover Through Re-execution



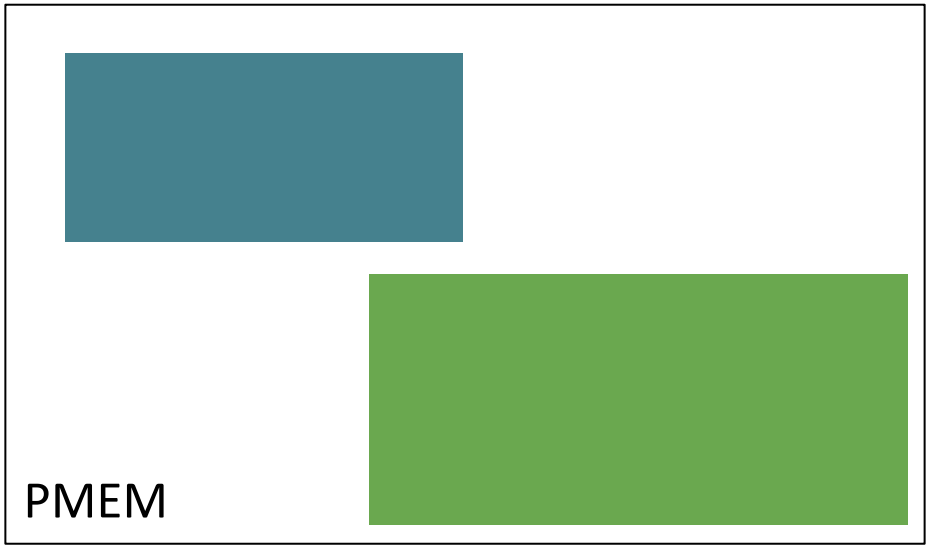
NVSL

Recover Through Re-execution



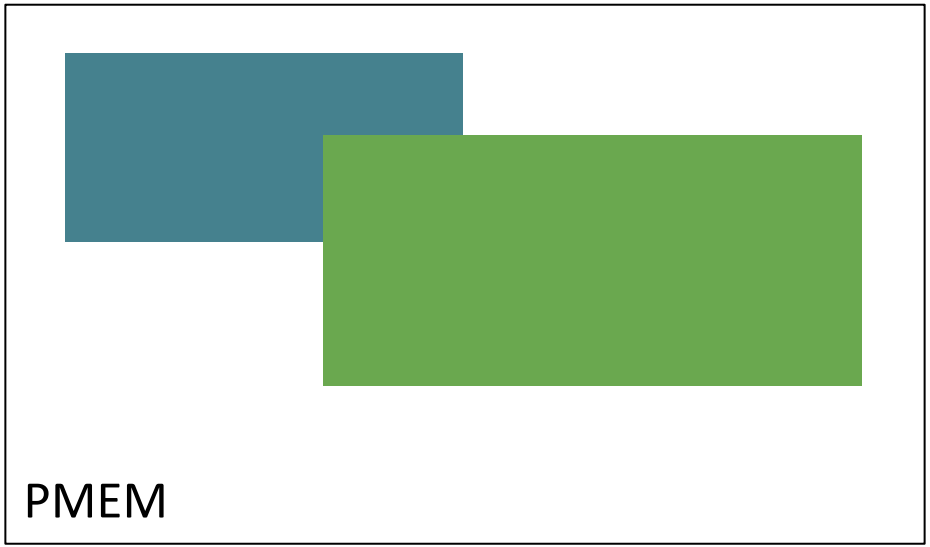
NVSL

Recover Through Re-execution



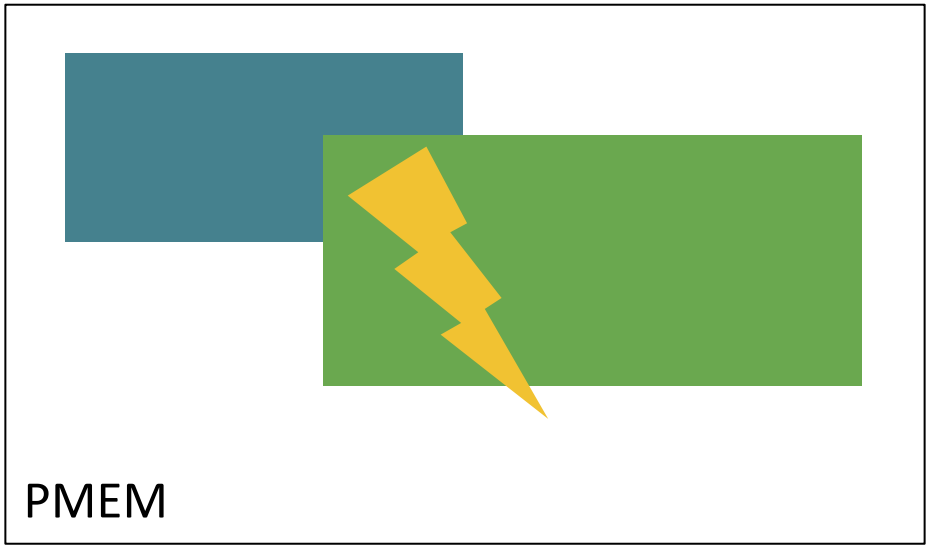
NVSL

Recover Through Re-execution



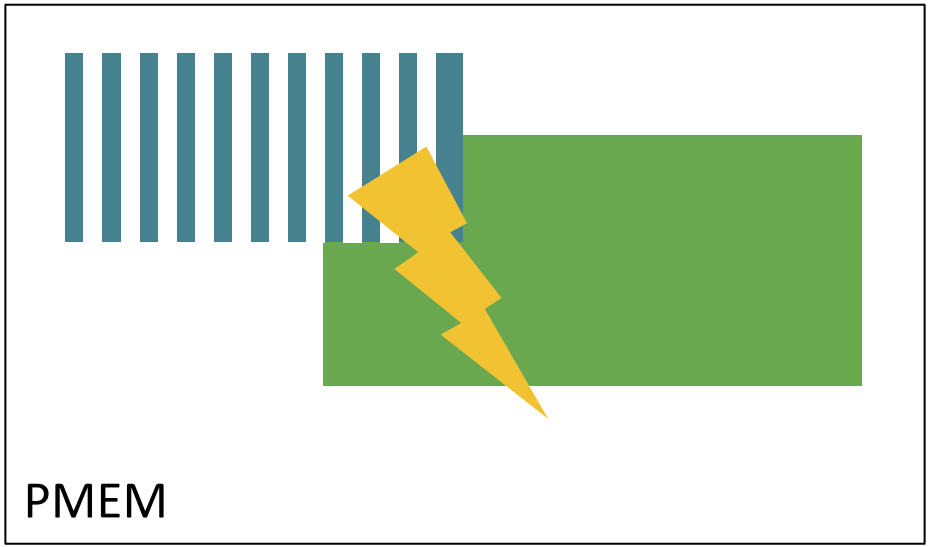
NVSL

Recover Through Re-execution



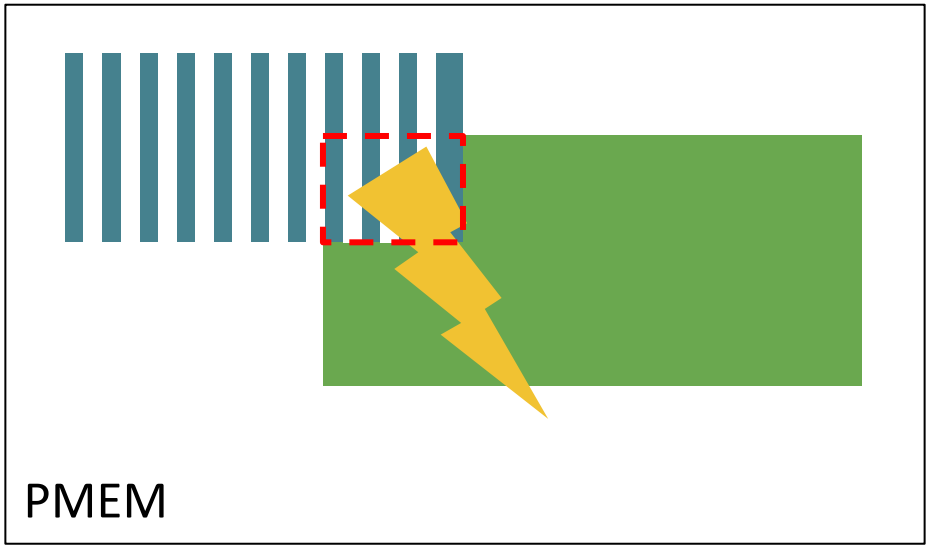
NVSL

Recover Through Re-execution



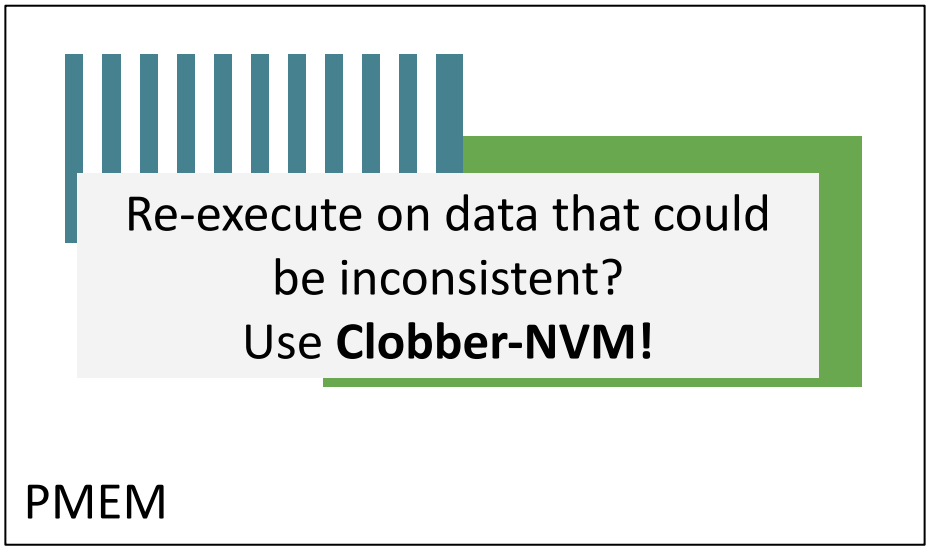
NVSL

Recover Through Re-execution



NVSL

Recover Through Re-execution



NVSL

Clobber-NVM Program

PMEM program with Clobber-NVM

```
void list_push(list_t *list,char* value){
    txbegin();
    memcpy(list->buf[list->size], value, strlen(value);
    list->size++;
    txend();
}
```

Transaction Boundary

NVSL

Clobber-NVM Program

PMEM program with Clobber-NVM

```
void list_push(list_t *list,char* value){
    txbegin();
    memcpy(list->buf[list->size], value, strlen(value);
    list->size++;
    txend();
}
```

Transaction Boundary

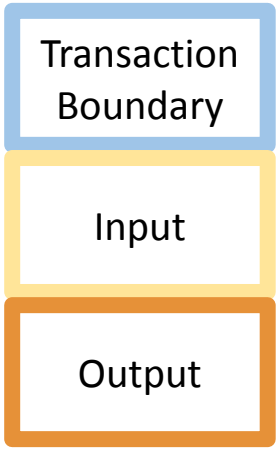
Input

NVSL

Clobber-NVM Program

PMEM program with Clobber-NVM

```
void list_push(list_t *list,char* value){
    txbegin();
    memcpy(list->buf[list->size], value, strlen(value);
    list->size++;
    txend();
}
```

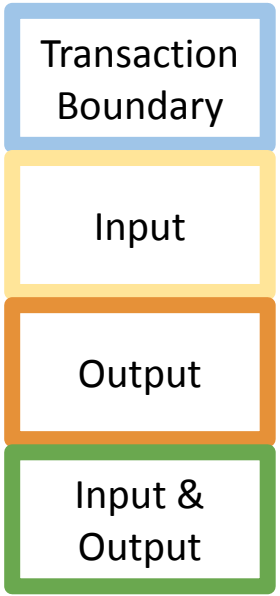


NVSL

Clobber-NVM Program

PMEM program with Clobber-NVM

```
void list_push(list_t *list,char* value){
    txbegin();
    memcpy(list->buf[list->size], value, strlen(value);
    list->size++;
    txend();
}
```

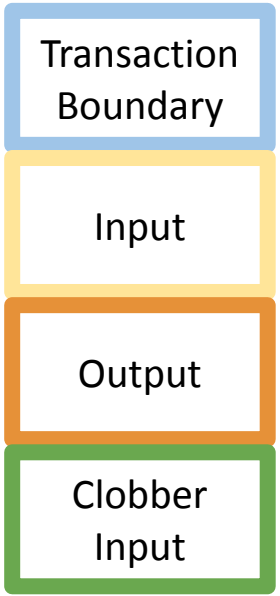


NVSL

Clobber-NVM Program

PMEM program with Clobber-NVM

```
void list_push(list_t *list,char* value){
    txbegin();
    memcpy(list->buf[list->size], value, strlen(value);
    list->size++;
    txend();
}
```



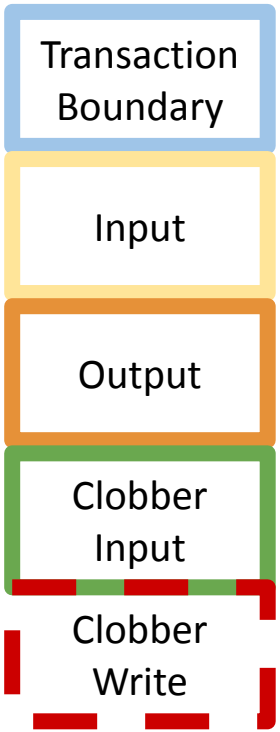
NVSL

Clobber-NVM Program

PMEM program with Clobber-NVM

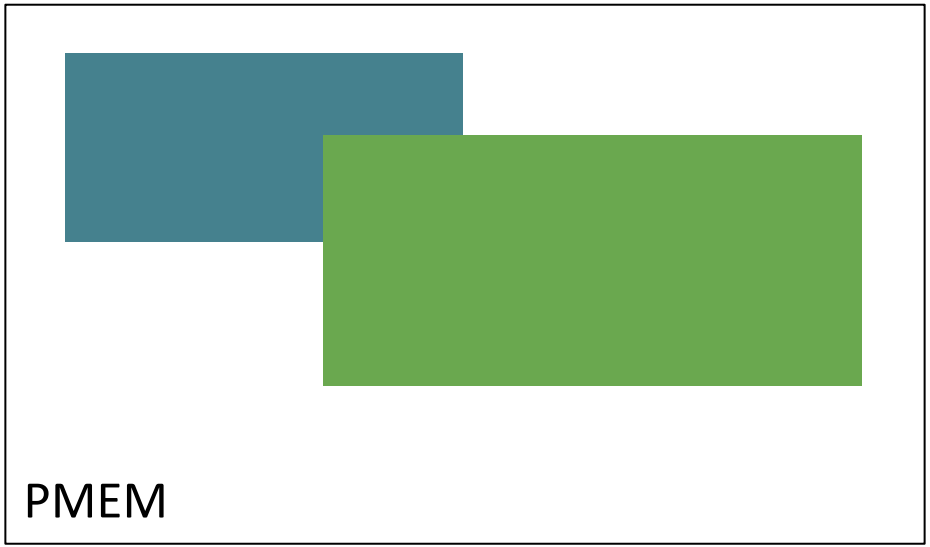
```
void list_push(list_t *list,char* value){
    txbegin();
    memcpy(list->buf[list->size], value, strlen(value);
    list->size++;
    txend();
}
```

A transaction input is a clobbered input if it may be overwritten within this transaction, and this write is a clobber write.



NVSL

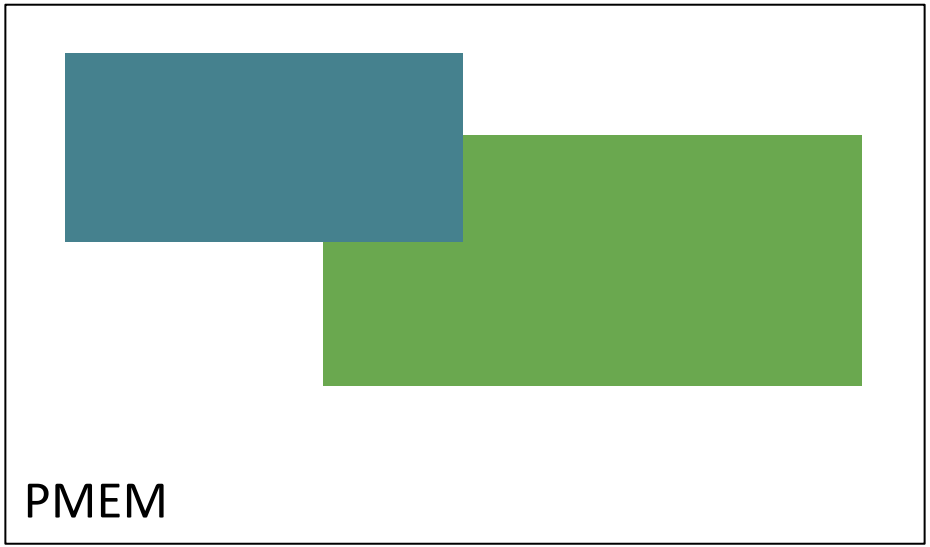
Clobbered Inputs



Clobbered inputs are a problem for re-execution.

NVSL

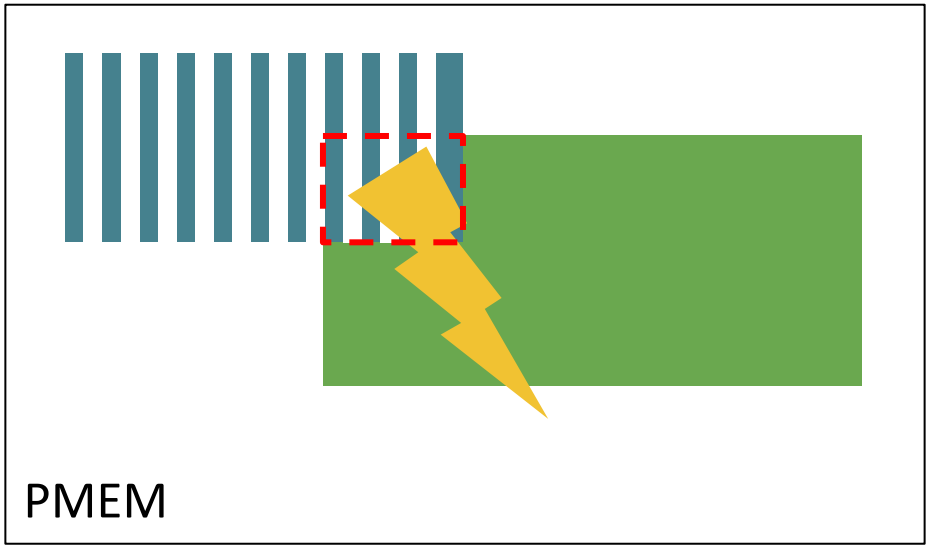
Clobbered Inputs



Clobbered inputs are a problem for re-execution.

NVSL

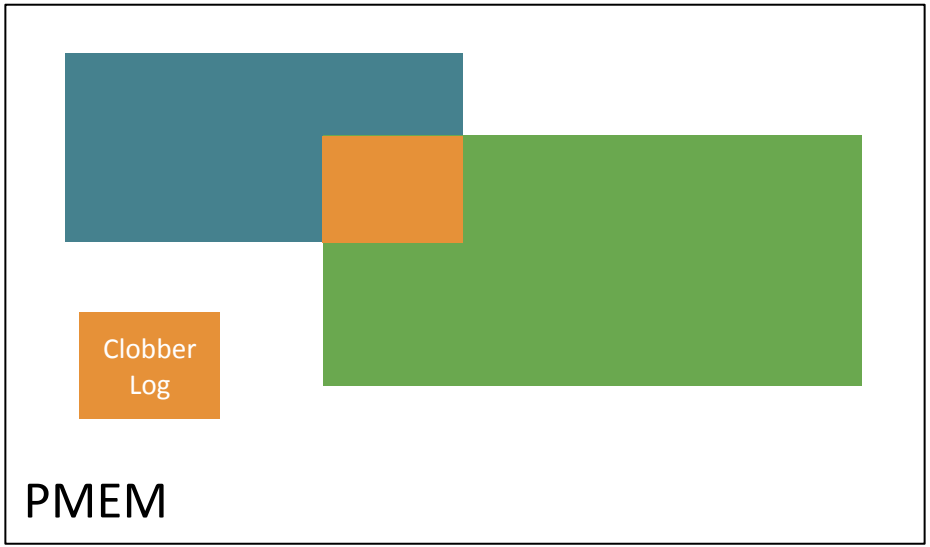
Clobbered Inputs



Clobbered inputs are a problem for re-execution.

NVSL

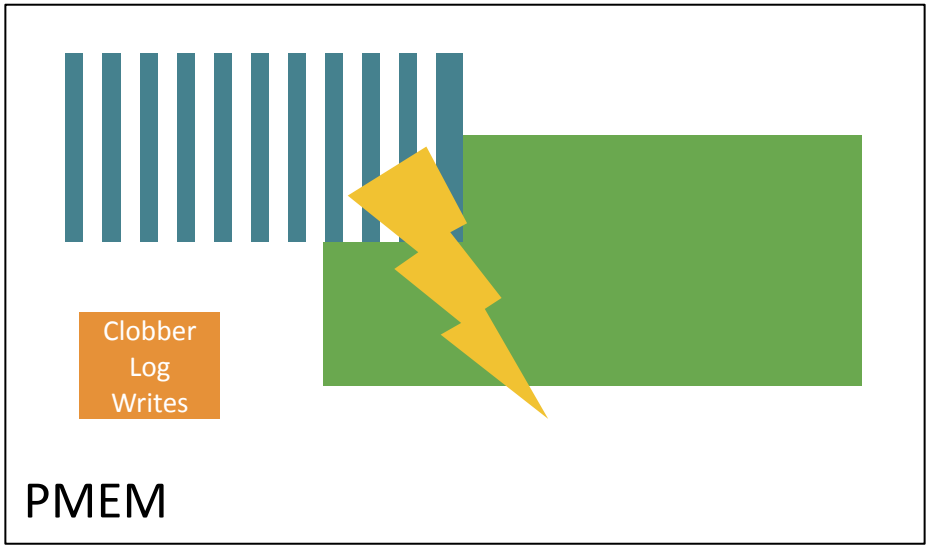
Clobber_Log before Clobber Writes



Clobber_Log --- undo logs before clobber writes.

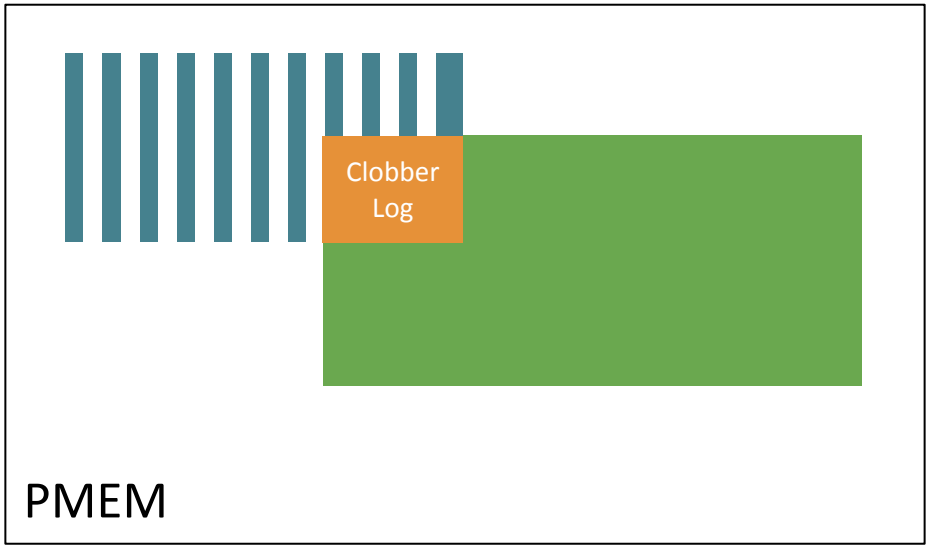
NVSL

Clobber_log before Clobber Writes



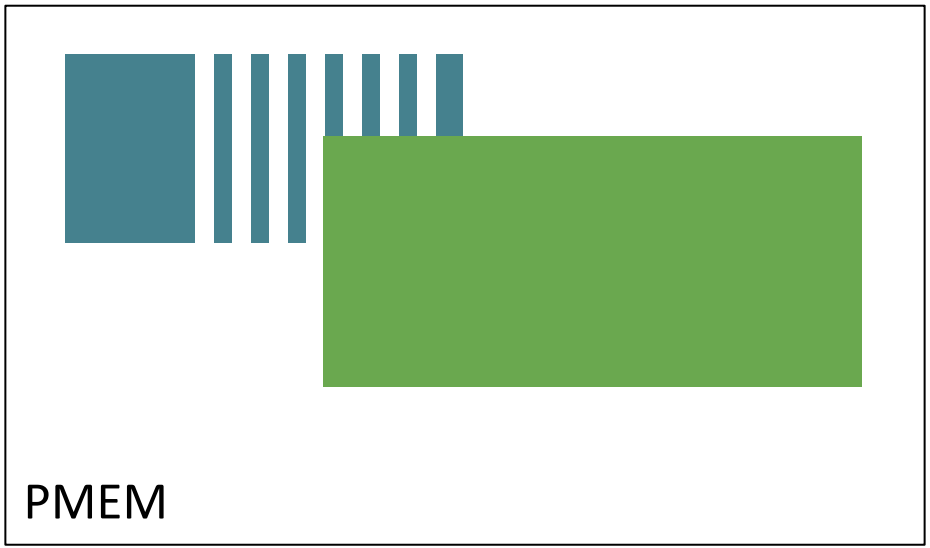
NVSL

`Clobber_log` before Clobber Writes



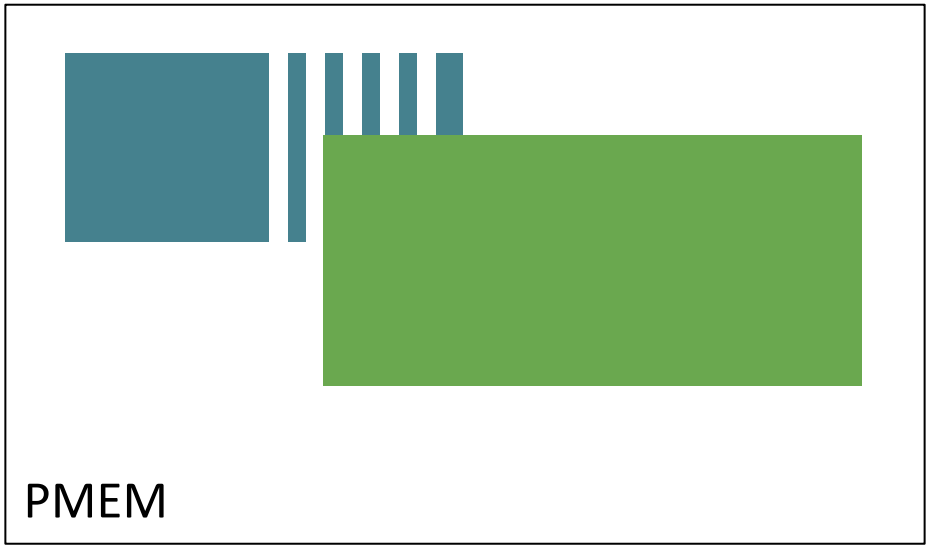
NVSL

Re-execute Based on `Clobber_log`



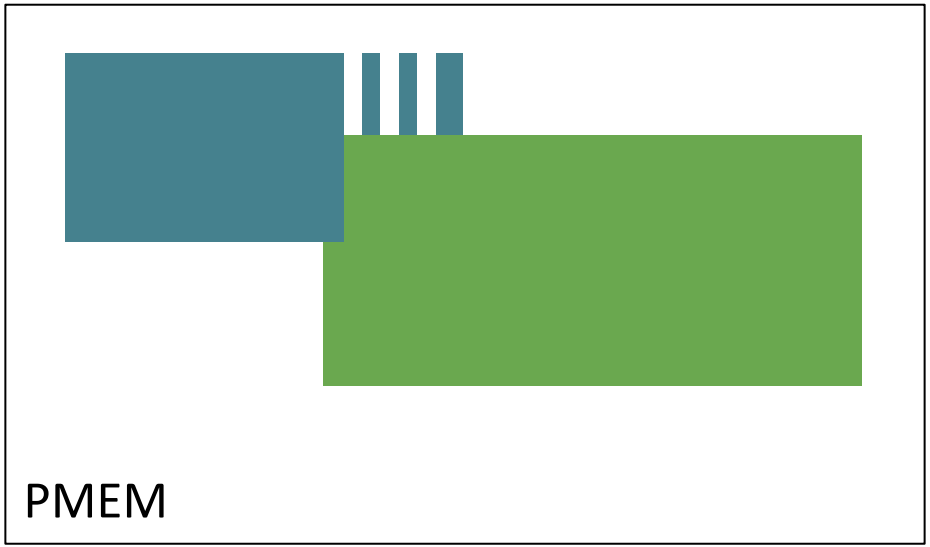
NVSL

Re-execute Based on `Clobber_log`



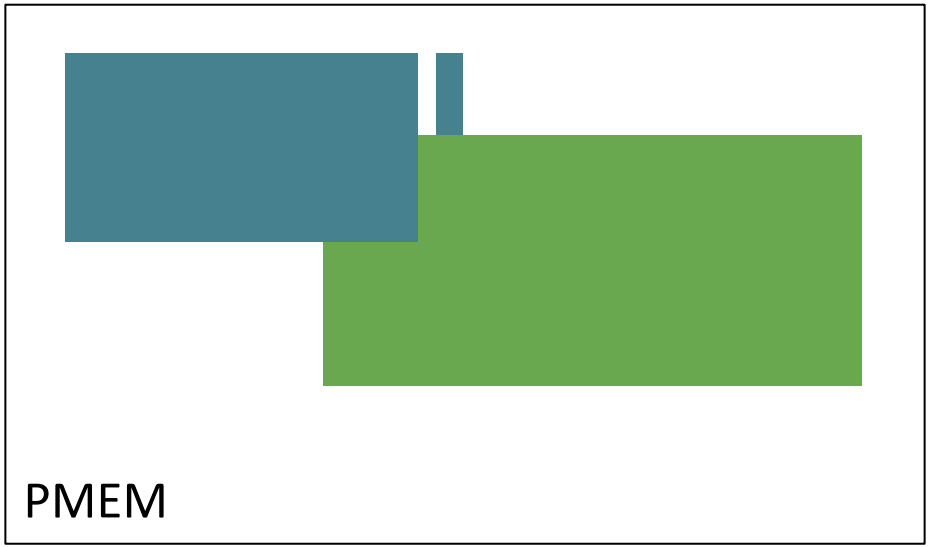
NVSL

Re-execute Based on `Clobber_log`



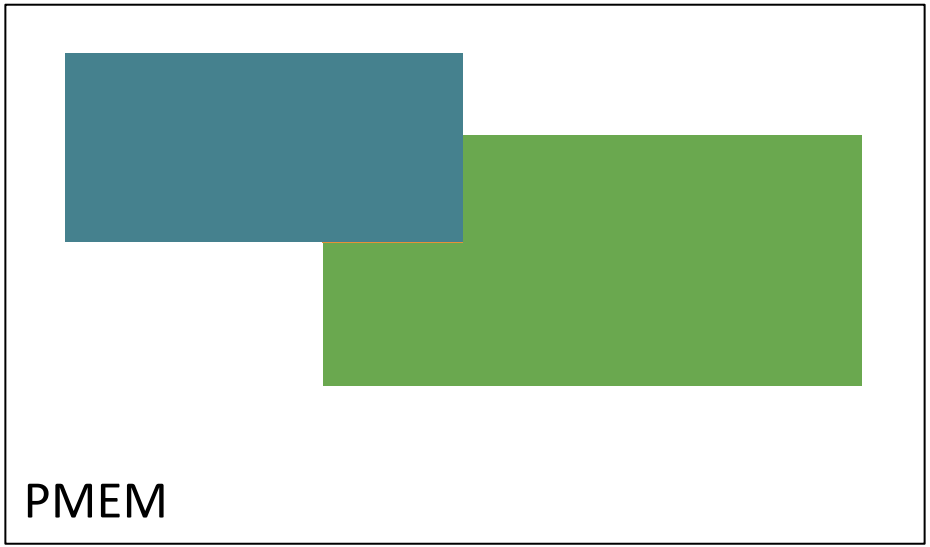
NVSL

Re-execute Based on `Clobber_log`



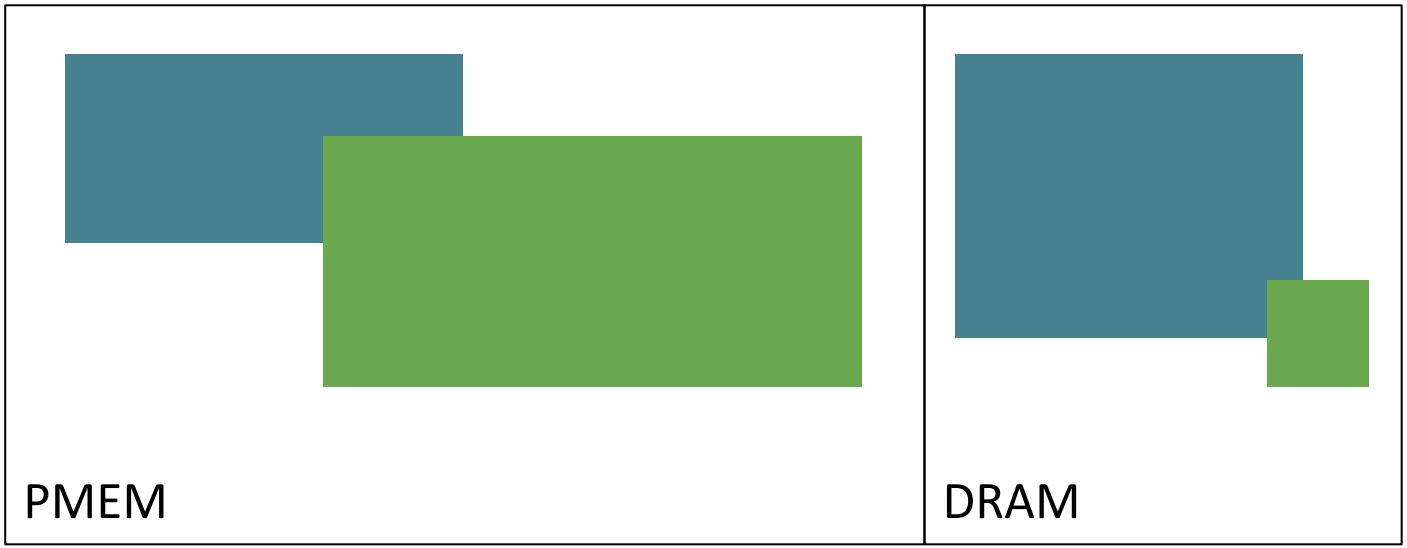
NVSL

Re-execute Based on `Clobber_log`



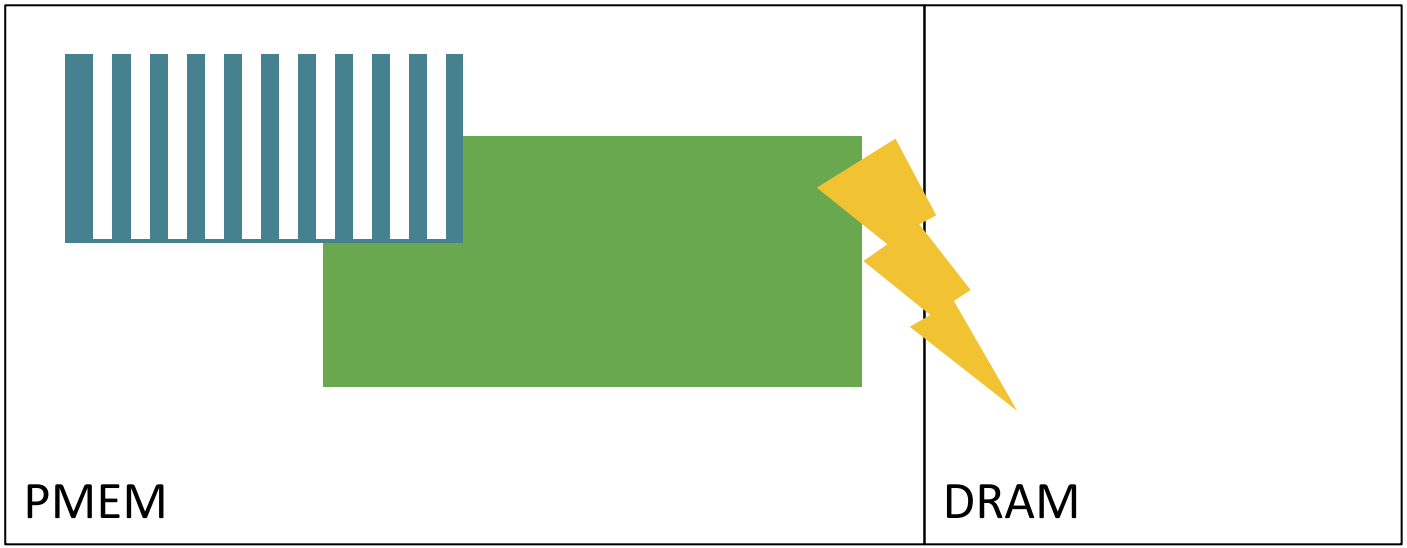
NVSL

Handle DRAM Accesses



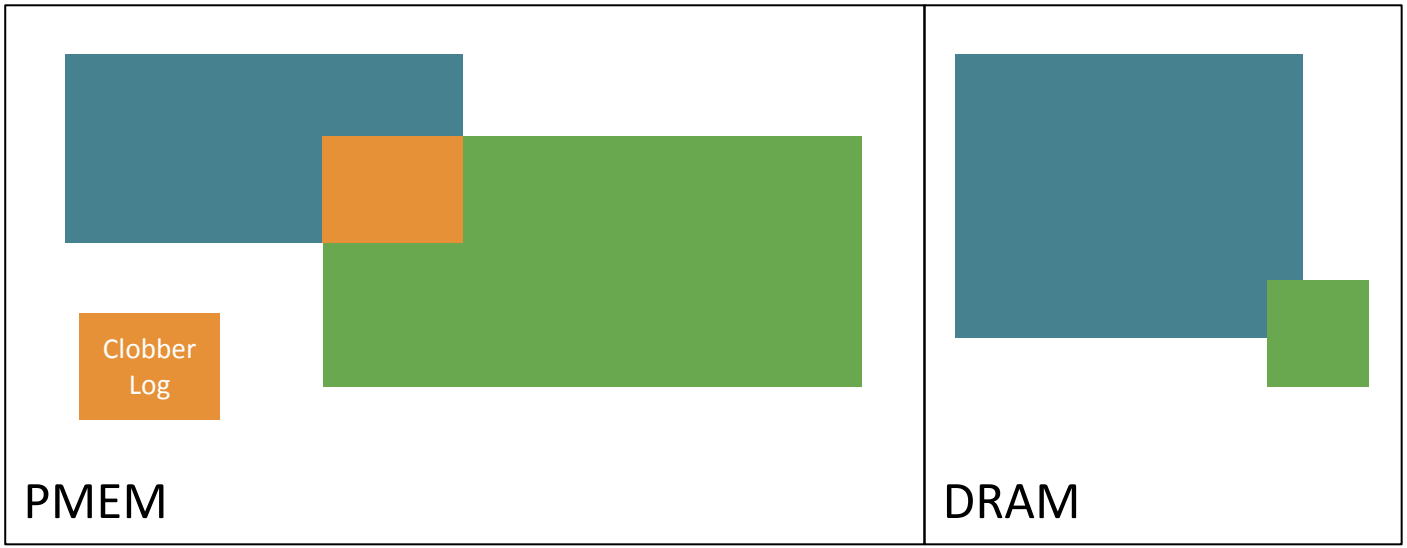
NVSL

Handle DRAM Accesses



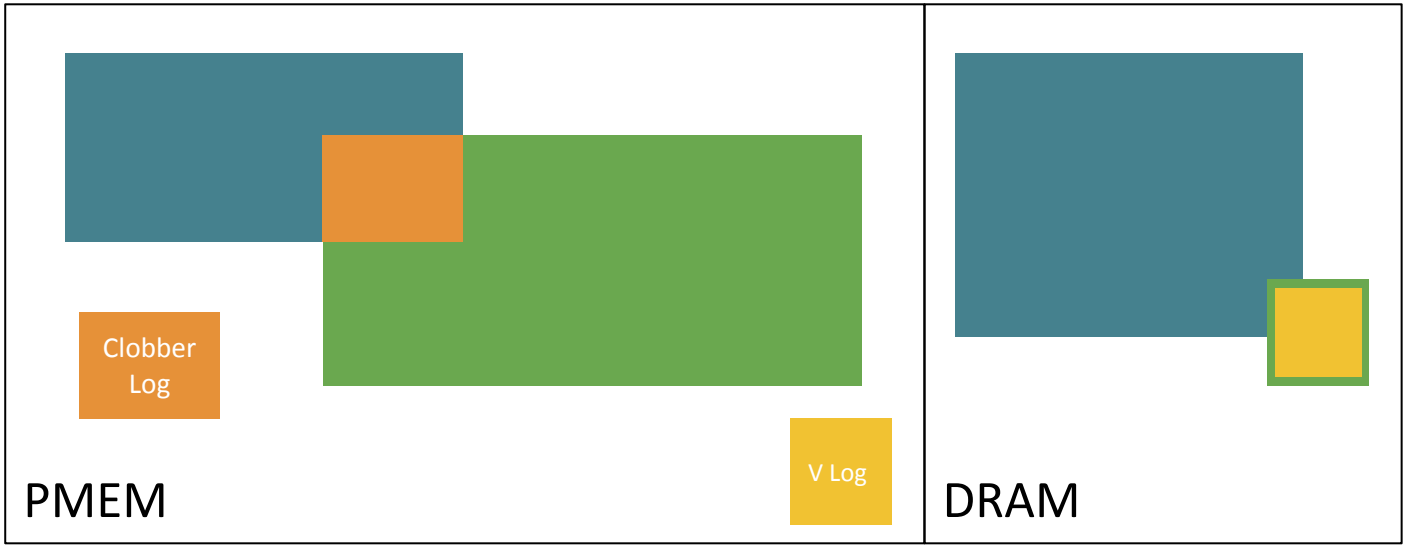
NVSL

Handle DRAM Accesses



NVSL

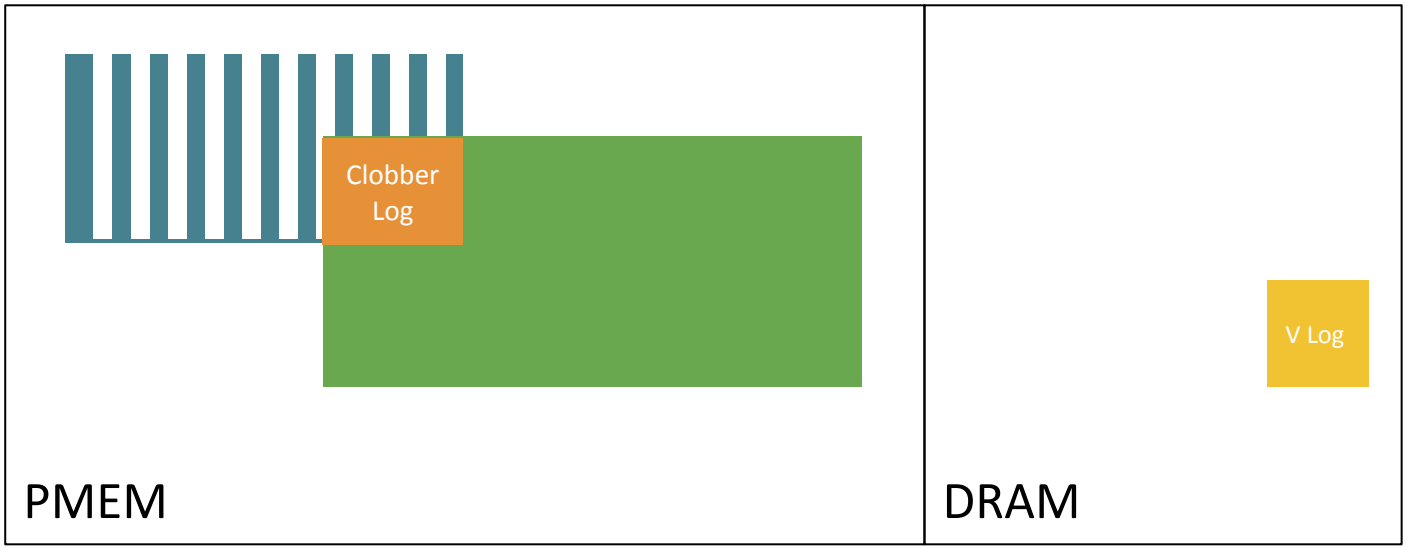
Handle DRAM Accesses on v_log



v_log stores volatile transaction inputs

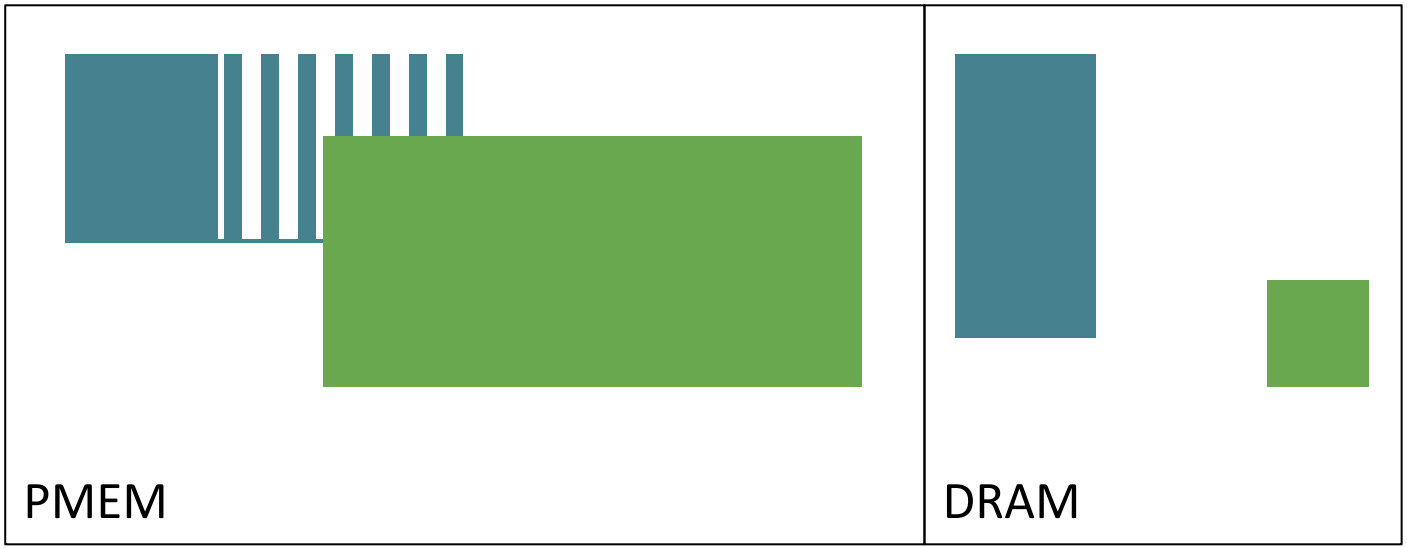
NVSL

Handle DRAM Accesses



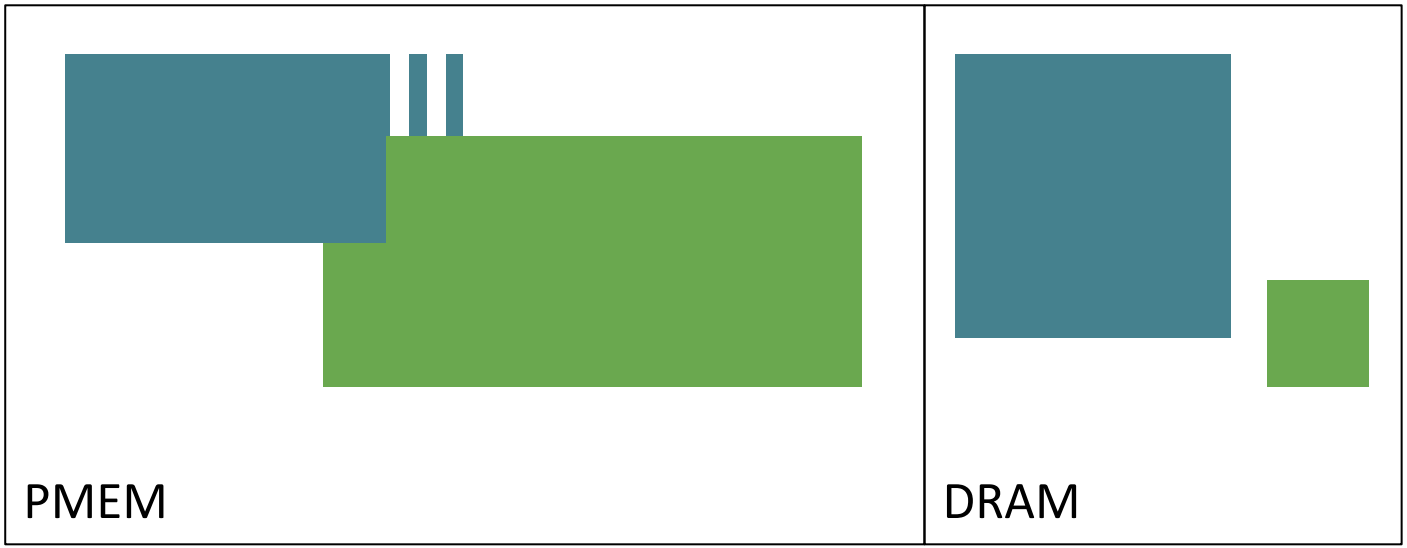
NVSL

Handle DRAM Accesses



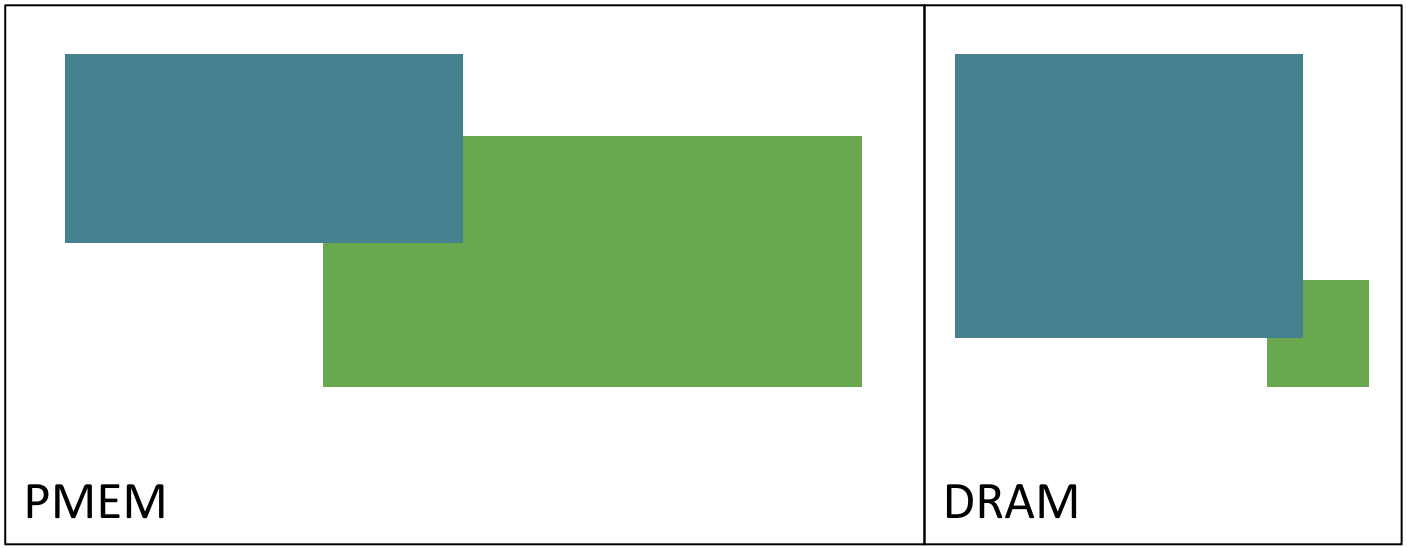
NVSL

Handle DRAM Accesses



NVSL

Handle DRAM Accesses



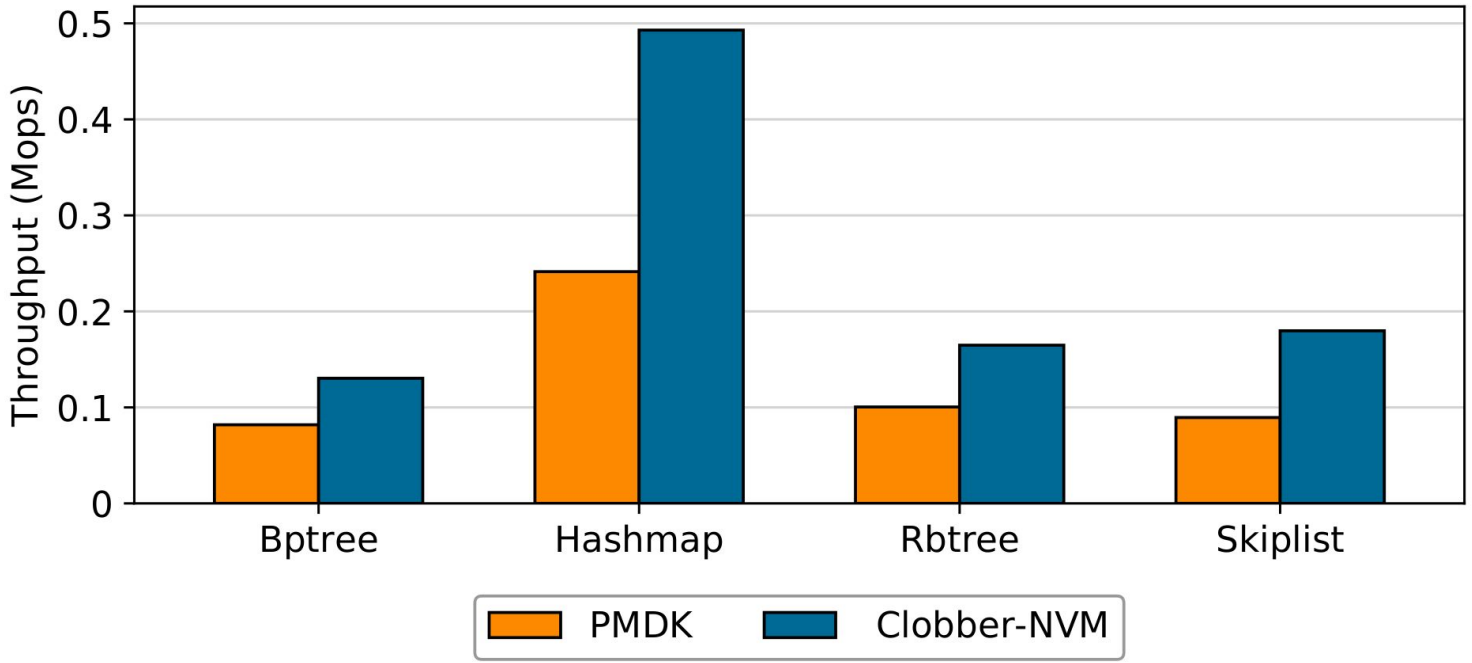
NVSL

Evaluation Setup

- Platform: two 24-core Intel Cascade Lake SP processors, running at 2.2 GHz. The platform has a total of 192 GB of DRAM and 1.5 TB (6 ×256 GB) of Intel Optane DC Persistent Memory directly attached to each processor.
- Configured Optane DCPMM in 100% App Direct mode.
- All experiments use Ext4 to manage persistent pools and directly access NVM pages via DAX.

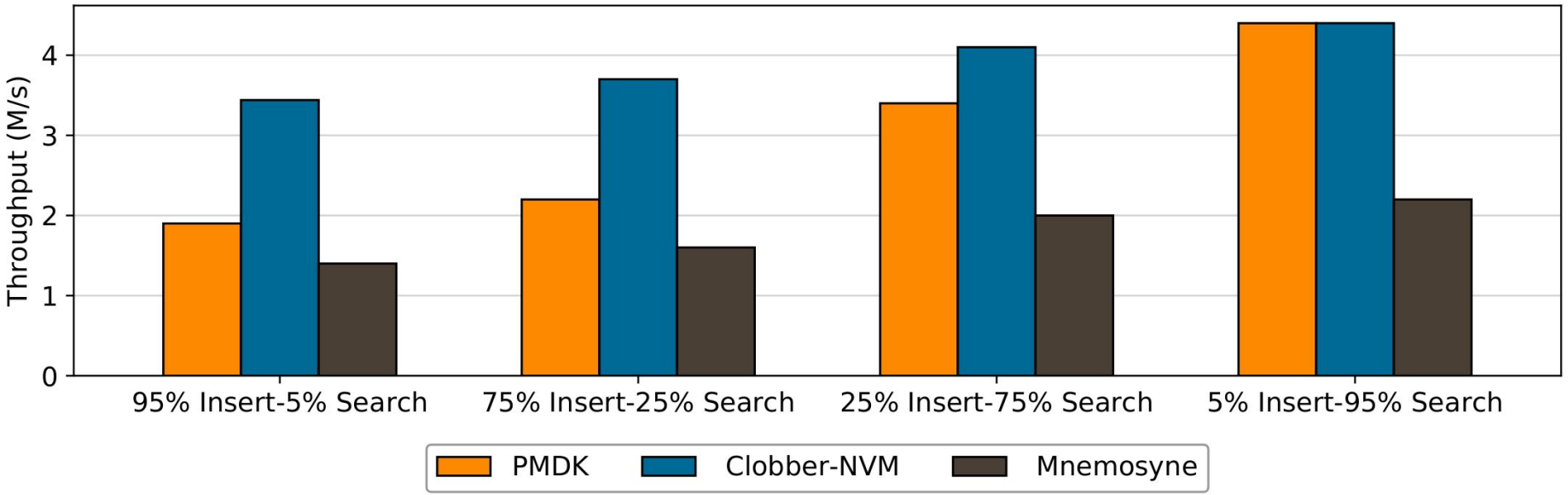
NVSL

Data structure Benchmarks



NVSL

Memcached Performance



NVSL

Conclusion

- Clobber-NVM: Recovers by re-executing interrupted transactions.
- Clobber-NVM compiler: Identifies necessary log entries, and automatically adds logging for selected variables.
- Evaluation shows that Clobber-NVM has high performance.

NVSL