Higher Level Persistent Memory Programming

Adrian Jackson
EPCC
a.jackson@epcc.ed.ac.uk
@adrianjhpc

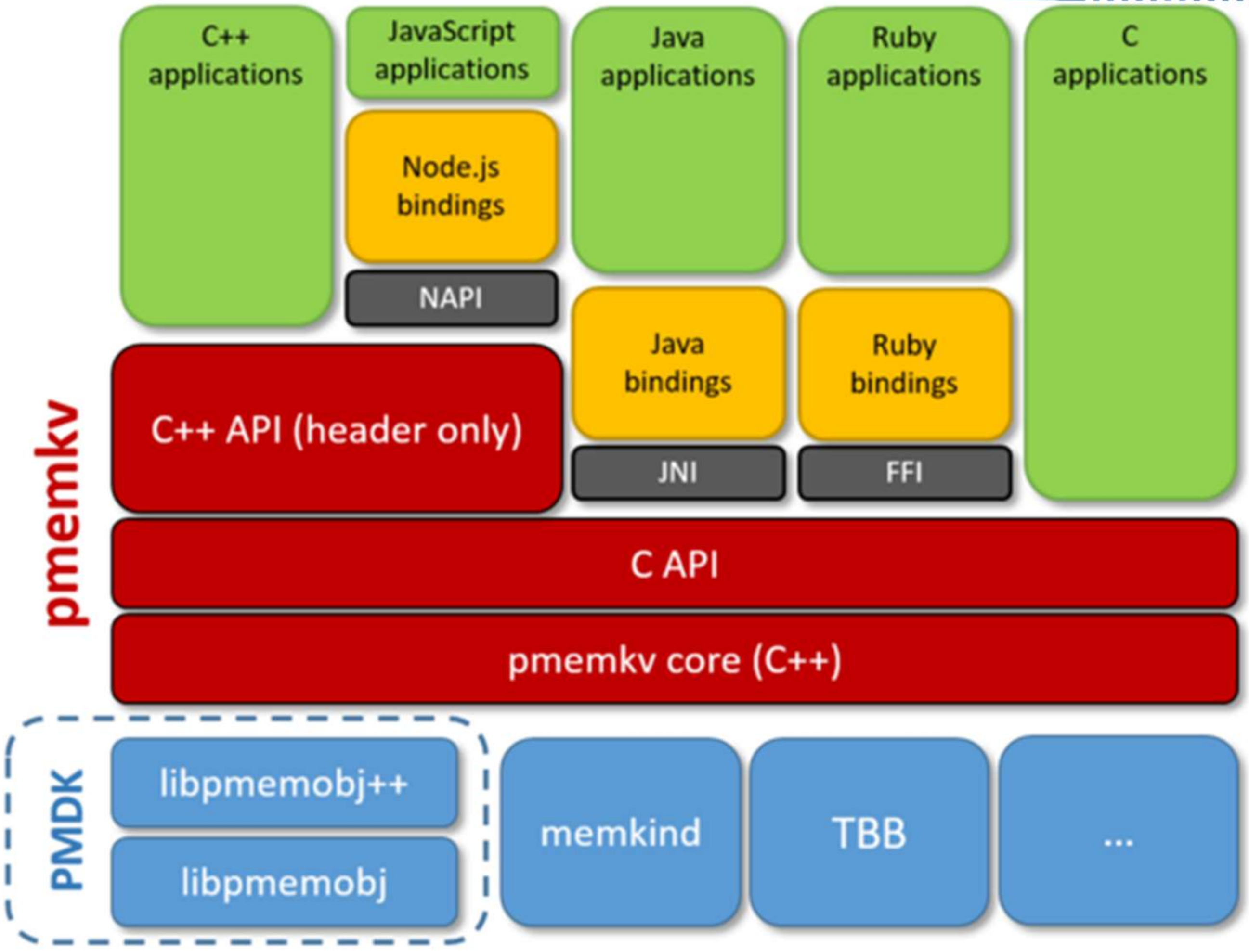
PMDK

- Provides higher level programming functionality
- How people really expect you to be programming NVRAM
- Transactional memory
- Key value stores
- Memory pools
- Manage the persistence nasties for you

PMDK higher level options

- libpmemobj
  - Transactional object store
  - Providing memory allocation, transaction
- Pools
  - libpmemblk
    - Blocks, all the same size, that are atomically updated
  - libpmemlog
    - Log file on persistent memory
- pmemkv
  - Key value store

pmemkv


C++ applications
JavaScript applications
Node.js bindings
NAPI
Java applications
Ruby applications
C applications
Java bindings
Ruby bindings
JNI
FFI
C++ API (header only)
pmemkv
C API
pmemkv core (C++)
PMDK
libpmemobj++
libpmemobj
memkind
TBB
...

pmemkv

- Storage engine with simple operations
  - Start
  - Stop
  - Put
  - Get
  - Remove
  - Exists
- Also further iterator operations
  - Count
  - All
  - Each

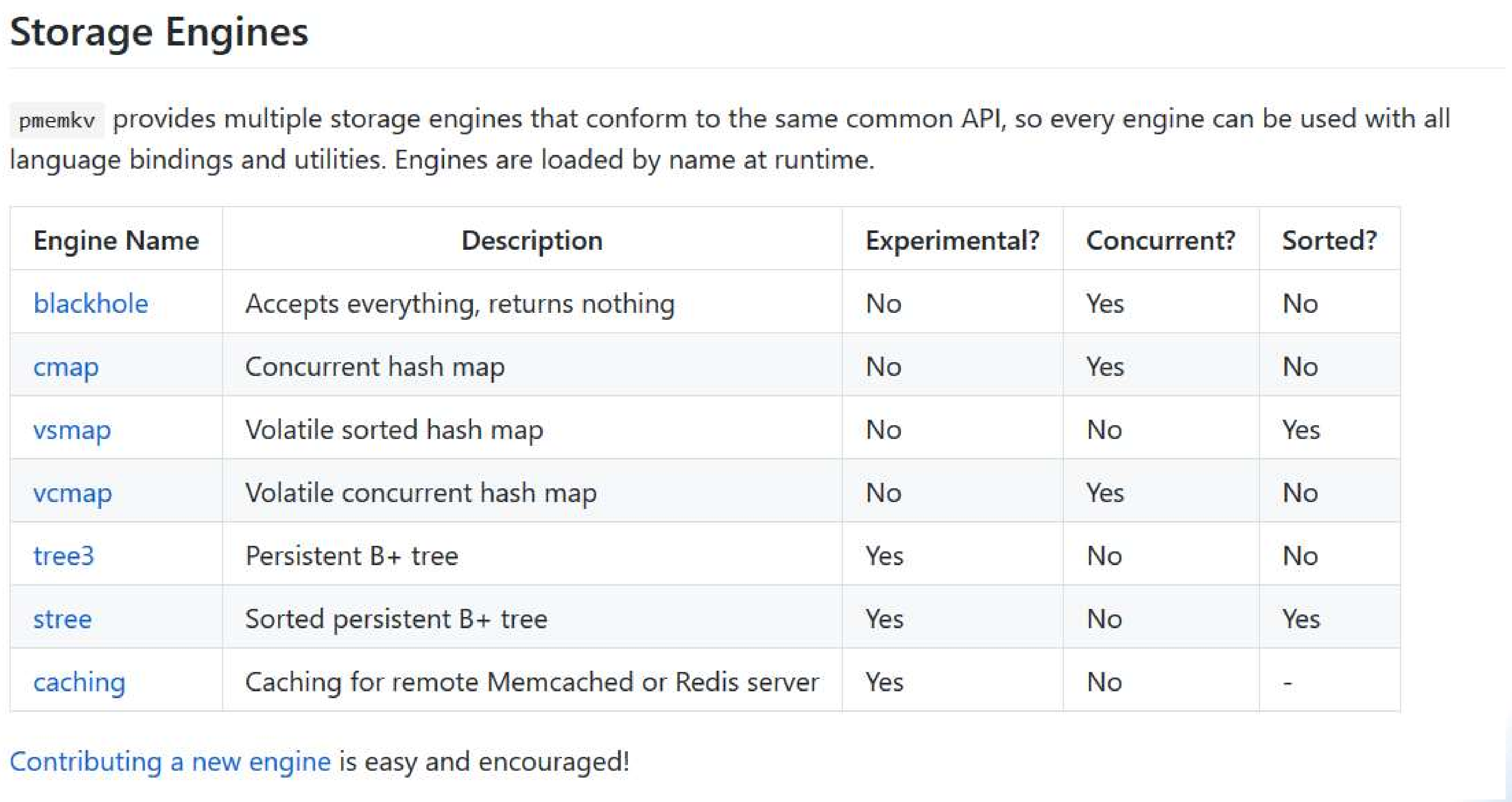
pmemkv

```
int main(){

KVEngine* kv = KVEngine::Start("vsmap","{\"path\":\"/dev/shm/\"}");
KVStatus s = kv->Put("key1", "value1");
string value;
s = kv->Get("key1", &value);
s = kv->Remove("key1");
delete kv;
return 0;

}
```

pmemkv



Storage Engines

`pmemkv` provides multiple storage engines that conform to the same common API, so every engine can be used with all language bindings and utilities. Engines are loaded by name at runtime.

Engine Name	Description	Experimental?	Concurrent?	Sorted?
blackhole	Accepts everything, returns nothing	No	Yes	No
cmap	Concurrent hash map	No	Yes	No
vsmap	Volatile sorted hash map	No	No	Yes
vcmap	Volatile concurrent hash map	No	Yes	No
tree3	Persistent B+ tree	Yes	No	No
stree	Sorted persistent B+ tree	Yes	No	Yes
caching	Caching for remote Memcached or Redis server	Yes	No	-

Contributing a new engine is easy and encouraged!

libpmemobj

- Object store with transactions
- Provides functions to create and manage data in persistent memory
- Provides macros and functions to add and remove data from object store
- Node local

libpmemobj

- **PMEMobjpool** *pmemobj_open(**const char** *path, **const char** *layout);
- **void** pmemobj_close(**PMEMobjpool** *pop);
- **PMEMoid** pmemobj_root(**PMEMobjpool** *pop, **size_t** size);
- **int** pmemobj_tx_add_range(**PMEMoid** oid, **uint64_t** off, **size_t** size);
- **int** pmemobj_tx_add_range_direct(**const void** *ptr, **size_t** size);
- **PMEMoid** pmemobj_tx_alloc(**size_t** size, **uint64_t** type_num);
- **int** pmemobj_tx_free(**PMEMoid** oid);
- **void** *pmemobj_direct(**PMEMoid** oid);
- TX_BEGIN(**PMEMobjpool** *pop) / TX_END
- OID_NULL, OID_IS_NULL(PMEMoid oid)

libpmemobj

```
/* TX_STAGE_NONE */

TX_BEGIN(pop) {
      /* TX_STAGE_WORK */
} TX_ONCOMMIT {
      /* TX_STAGE_ONCOMMIT */
} TX_ONABORT {
      /* TX_STAGE_ONABORT */
} TX_FINALLY {
      /* TX_STAGE_FINALLY */
} TX_END

/* TX_STAGE_NONE *
```

libpmemobj C++

```
struct queue {
        void
        push(pmem::obj::pool_base &pop, int value)
        {
                pmem::obj::transaction::run(pop, [&] {
                        auto node = pmem::obj::make_persistent<queue_node>();
                        node->value = value;
                        node->next = nullptr;

                        if (head == nullptr) {
                                head = tail = node;
                        } else {
                                tail->next = node;
                                tail = node;
                        }
                });
        }
}
```

libpmemblk

- Designed for array of blocks
  - updates to a single block are atomic

```
int main(int argc, char *argv[]){
        const char path[] = "/pmem-fs/myfile";
        PMEMblkpool *pbp;
        size_t nelements;
        char buf[ELEMENT_SIZE];
        pbp = pmemblk_create(path, ELEMENT_SIZE, POOL_SIZE, 0666);
        if (pbp == NULL)
                pbp = pmemblk_open(path, ELEMENT_SIZE);
        }
        /* how many ellments fit into the file? */
        nelements = pmemblk_nblock(pbp);
        strcpy(buf, "starting time");
        pmemblk_write(pbp, buf, 5)
        pmemblk_read(pbp, buf, 10);
        pmemblk_set_zero(pbp, 5);
        pmemblk_close(pbp);
}
```

libpmemlog

- NVRAM resident log file
- Designed for append-mostly file
- Variable length entries

```
plp = pmemlog_create(path, POOL_SIZE, 0666);
plp = pmemlog_open(path);
data = "first thing";
pmemlog_append(plp, data, strlen(data)) ;
data = "second thing";
pmemlog_append(plp, data, strlen(data));
pmemlog_walk(plp, 0, printit, NULL);
```

Practical

- Have a look at PMDK further functionality and try implementing some other paradigms based on memory pools

- You may need to create a memory pool, i.e.:

```
pmempool create obj --layout=simplekv -s 100M
/mnt/pmem-fsdax0/simplekv-simple
```

- You can also interact with memory pools, ie.

```
pmempool info /mnt/pmem-fsdax0/simplekv-simple
```