SDC 19

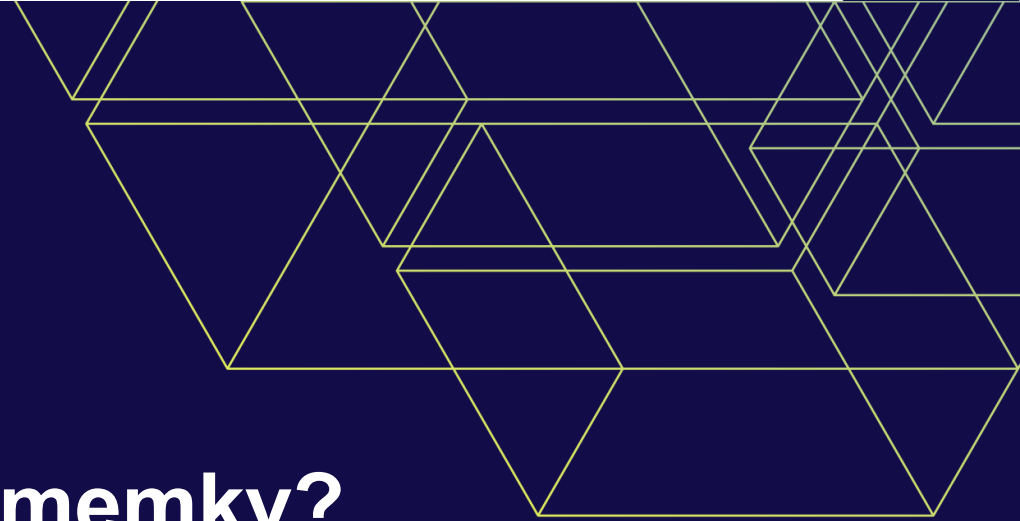
September 23-26, 2019
Santa Clara, CA

Persistent Memory Programming Made Easy with pmemkv

Andy Rudoff
Intel Corporation

Agenda

- Why pmemkv?
- pmemkv Design
- Engines
- Language bindings
- Performance



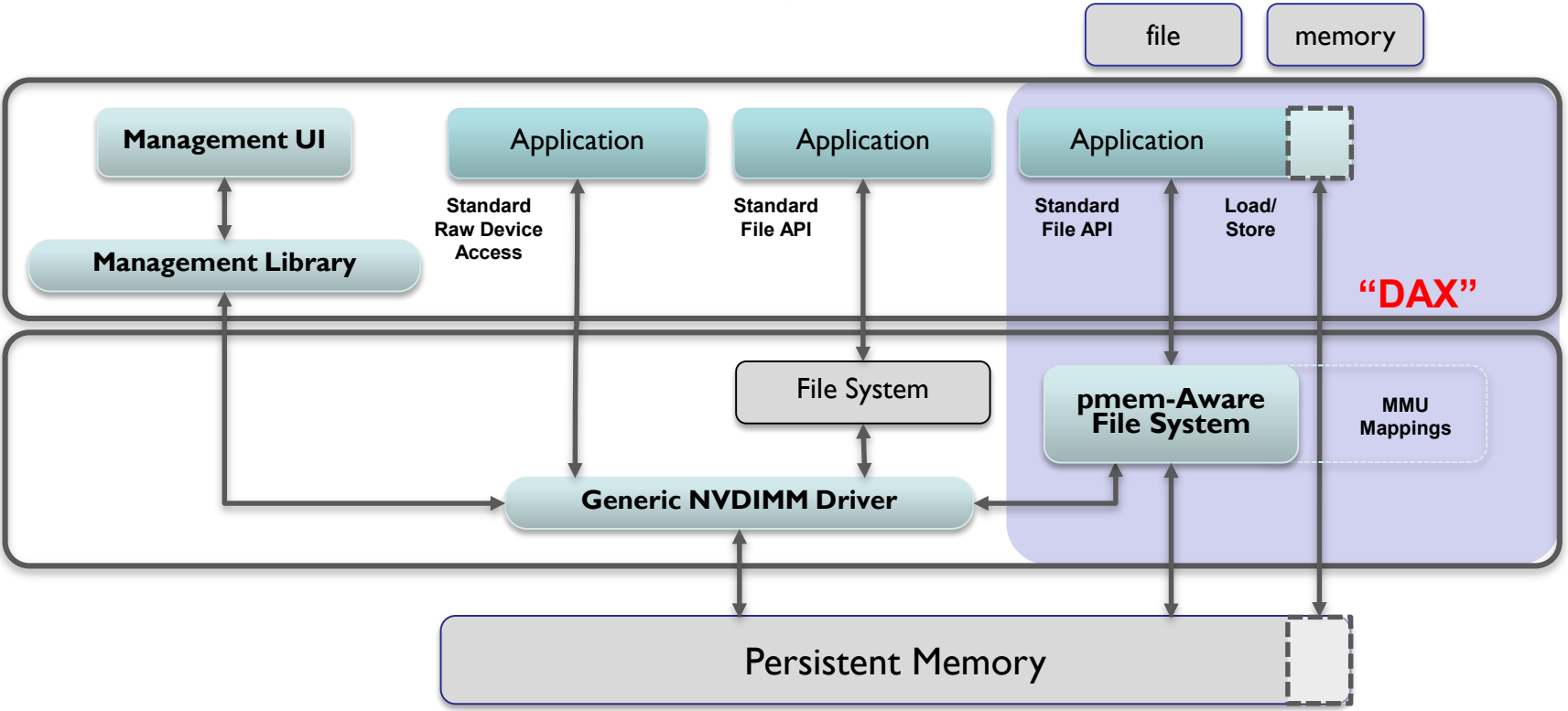
Why pmemkv?

Ways to Use Persistent Memory

- Memory Mode
  - Transparent to application
  - Transparent to OS
  - Volatile
  - …but not a match for every use case

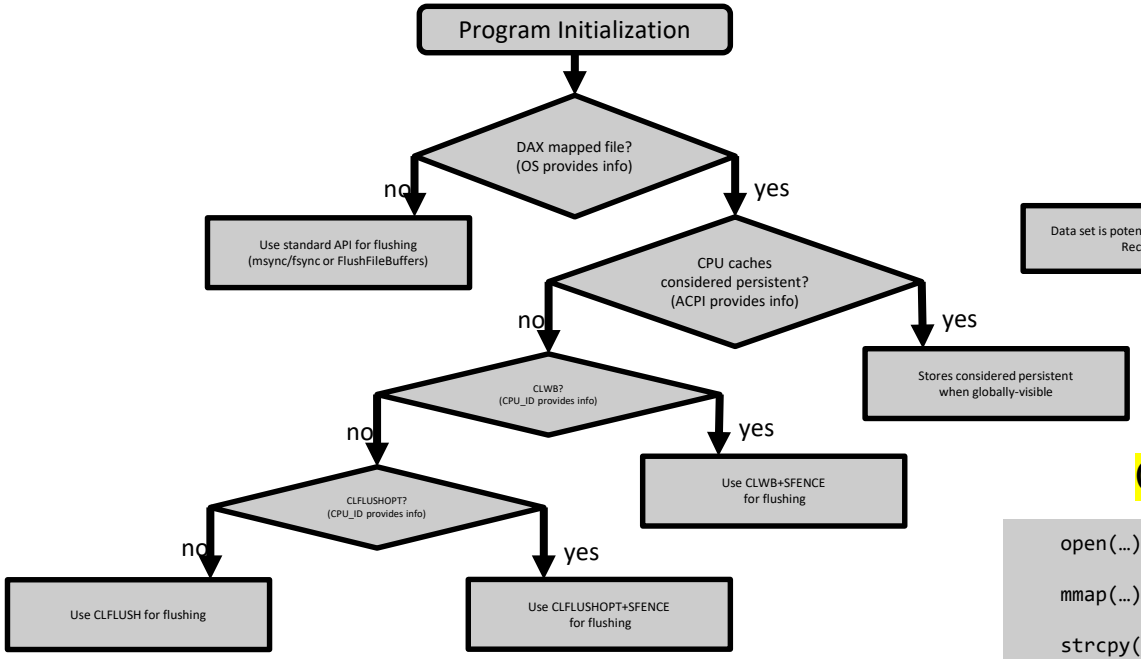
- As Storage…

The SNIA NVM Programming Model

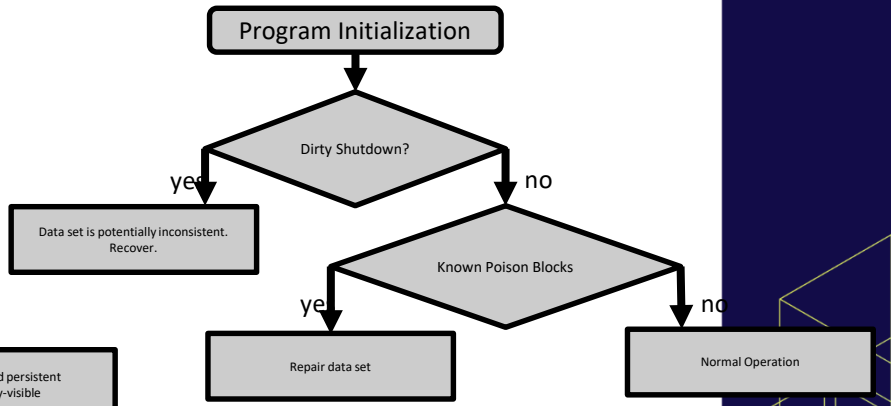


With Great Direct Access Comes Great Responsibility

Flushing…



RAS…



Consistency…

```
open(…);

mmap(…);

strcpy(pmem, "Hello, World!");

msync(…);
```

Crash

PMDK

“make pmem programming easier”

PCJ – Persistent Collection for Java

C++ | C | PCJ/ LLPL | Python

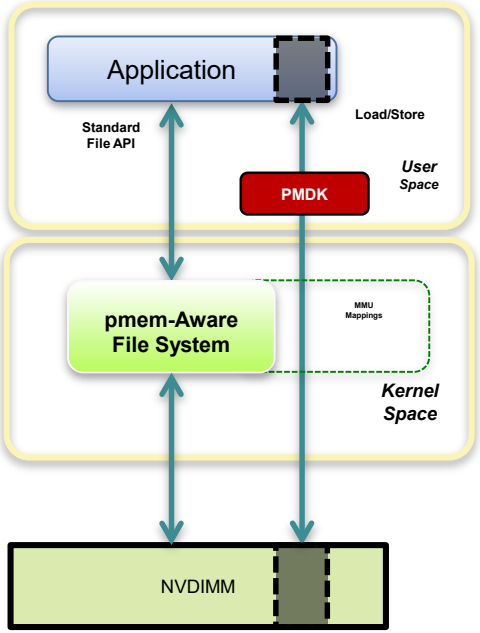
pmemkv

Transaction Support

Interface to create a persistent memory resident log file — libpmemlog

Interface for persistent memory allocation, transactions and general facilities — libpmemobj

Interface to create arrays of pmem-resident blocks, of same size, atomically updated — libpmemblk



Support for **volatile** memory usage

memkind

vmemcache

Low level support for local persistent memory — libpmem

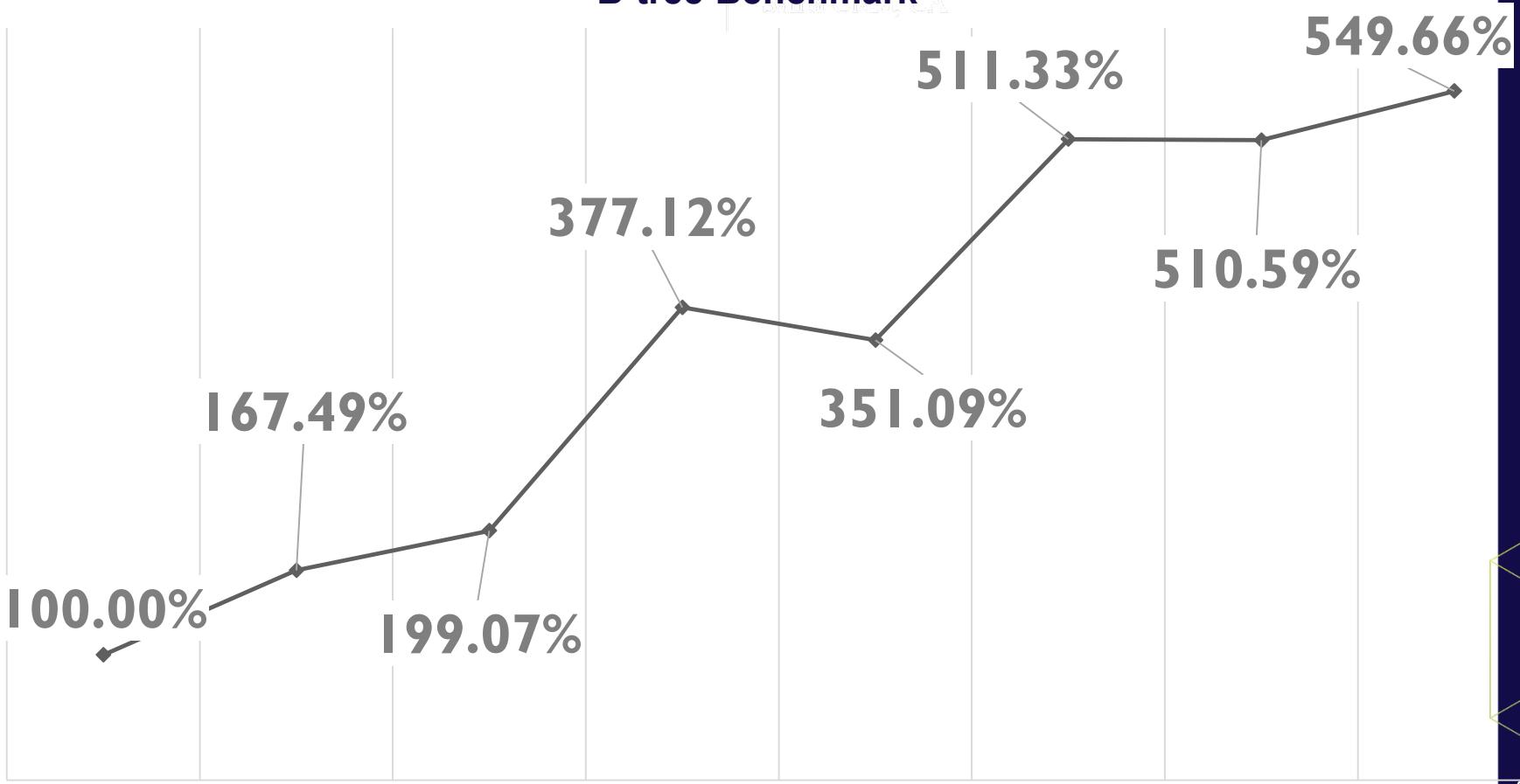
Low level support for remote access to persistent memory — librpmem

Low-level support

In Development

libpmemobj Performance Across Versions

B-tree Benchmark

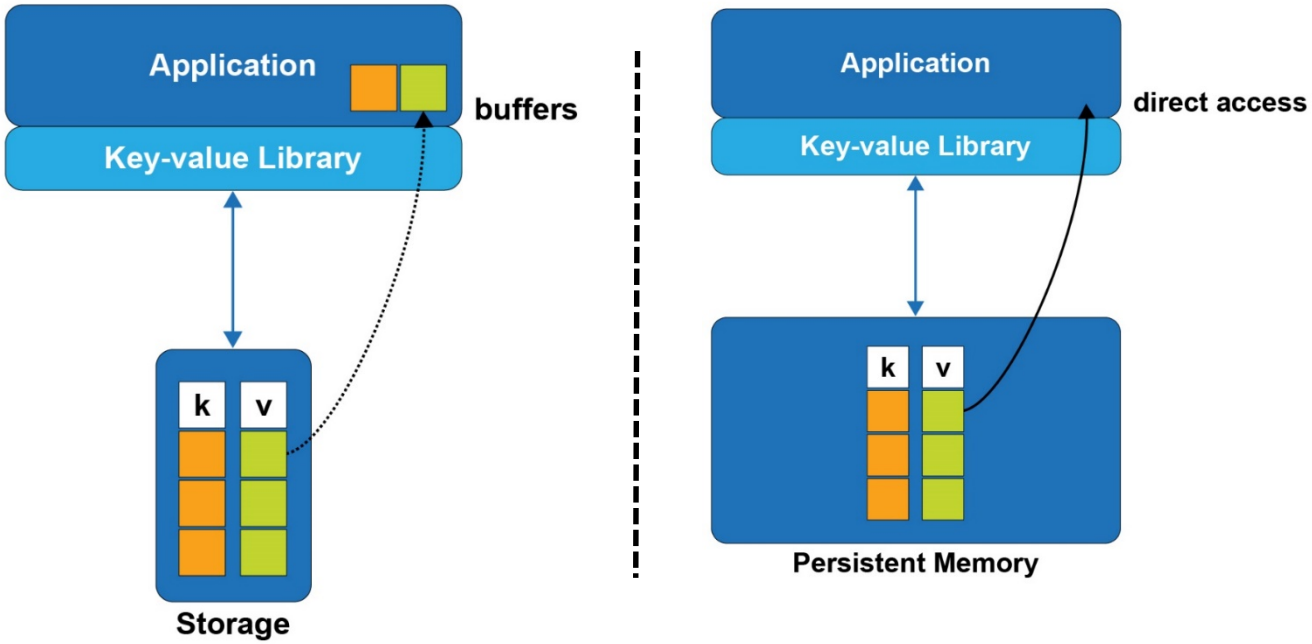


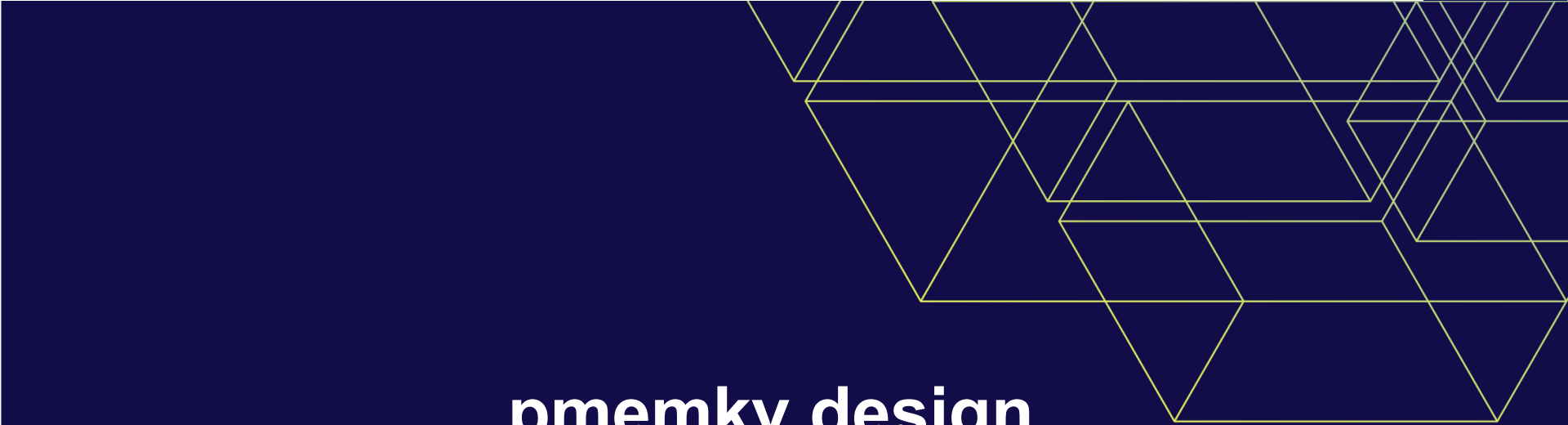
Why pmemkv?

- We have Memory Mode and Storage
  - for legacy use cases
- We have libpmem
  - for raw access
- We have libpmemobj
  - For transactions, allocation, containers
- How about a simple put/get API!
  - Tuned and validated to product quality

Why pmemkv?

- Key-value store can take advantage of access-in-place





pmemkv design

pmemkv design goals

Technical:

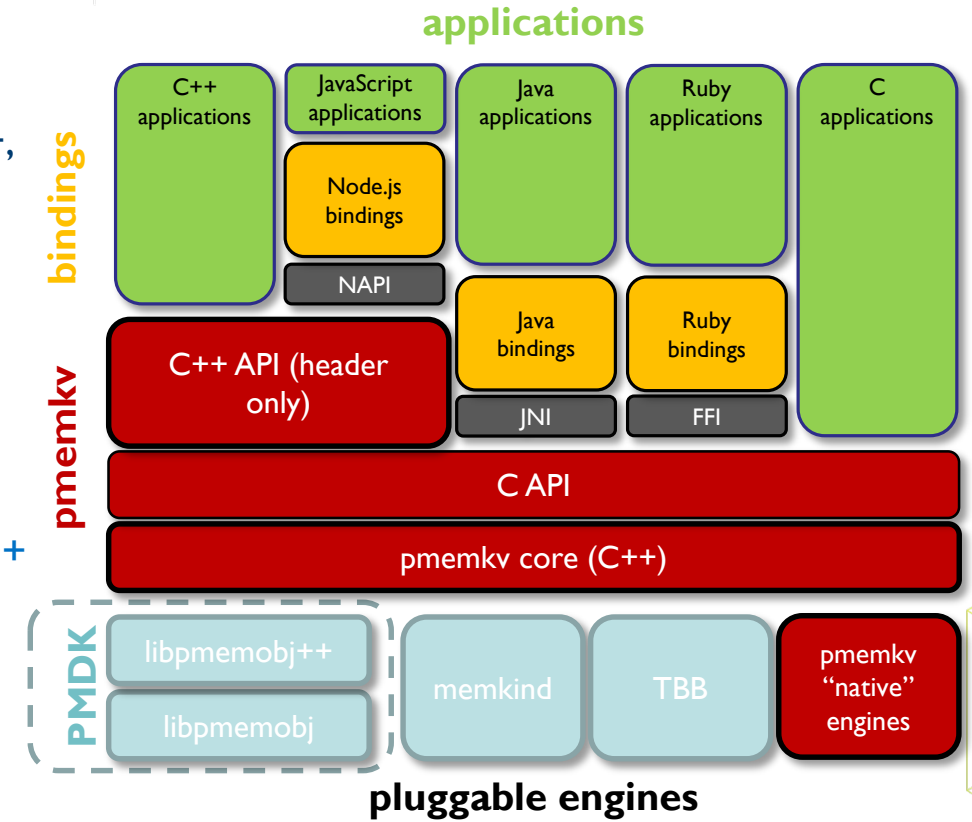
- Local key/value store (no networking)
- Idiomatic language bindings
- Simple, familiar, bulletproof API
- Easily extended with new engines
- Optimized for persistent memory (limit copying to/from DRAM)
- Flexible configuration, not limited to a single storage algorithm
- Generic tests

Community:

- Open source, developed in the open and friendly licensing
  - https://github.com/pmem/pmemkv
- Outside contributions are welcome
- Intel provides stewardship, validation on real hardware, and code reviews
- Standard/comparable benchmarks

pmemkv architecture

- pmemkv core is a frontend for engines
  - Core implementation written in C++, not related to Persistent Memory
- Pluggable engines
  - Some engines are implemented in pmemkv, some engines are imported from external projects
  - Persistent engines are implemented with libpmemobj (PMDK)
- Native API for pmemkv is written C/C++
- pmemkv design allows for easy integration with high-level language bindings



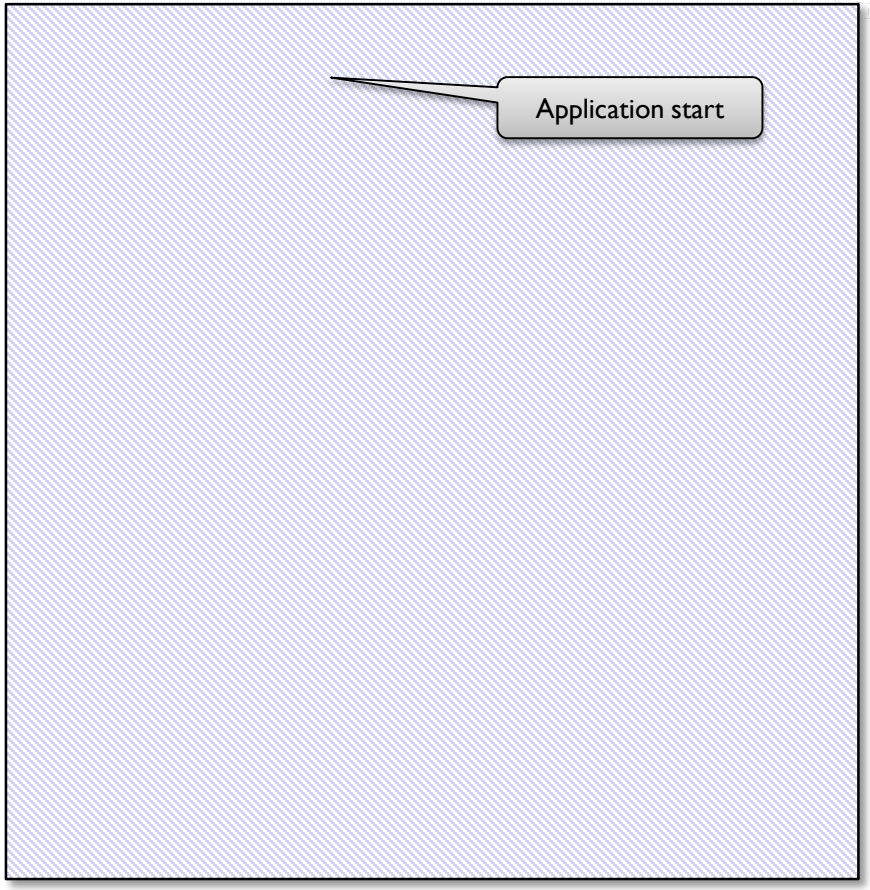
pmemkv configuration

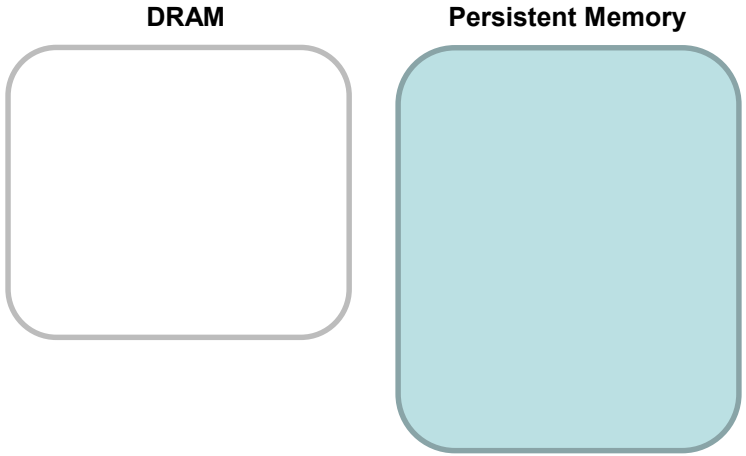
- Flexible configuration API
  - Works with different kinds of engines
- Every engine has documented supported config parameters individually
- Unordered map
  - Takes name configuration value as a k-v pair
- Supported configuration types:
  - int64/uint64/double
  - string
  - Arbitrary data (pointer and size)
- Resides on stack
  - Takes optional destructor as an additional parameter if custom configuration parameter allocates memory

Typical config structure example for libpmemobj-based engines

```
config cfg;

status s = cfg.put_string("path", path);
assert(s == status::OK);

s = cfg.put_uint64("size", SIZE);
assert(s == status::OK);

s = cfg.put_uint64("force_create", 1);
assert(s == status::OK);
```

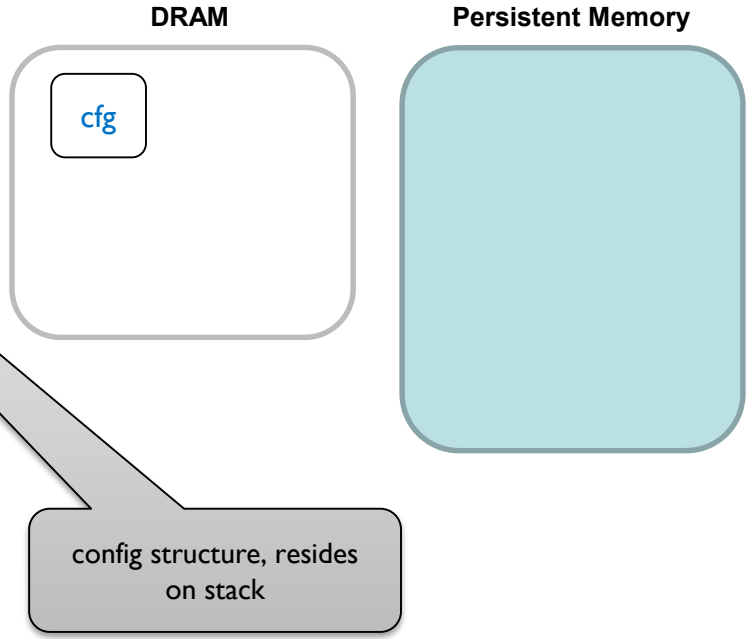
pmemkv design





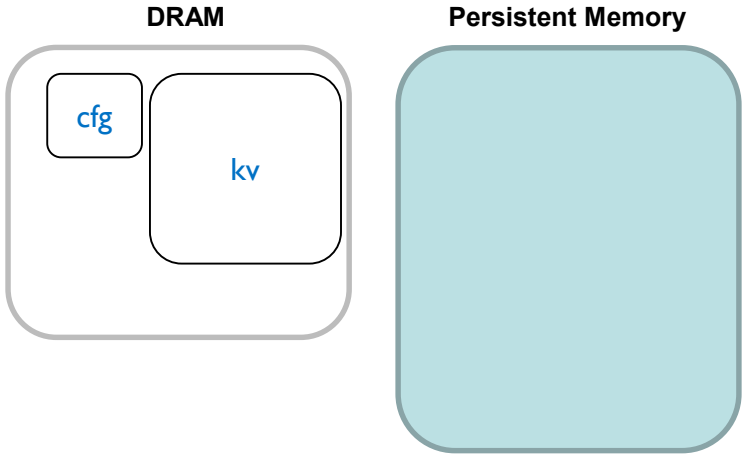
pmemkv design

```
config cfg;

status s =
    cfg.put_string("path", "/daxfs/file");
assert(s == status::OK);

s = cfg.put_uint64("size", SIZE);
assert(s == status::OK);
```



pmemkv design

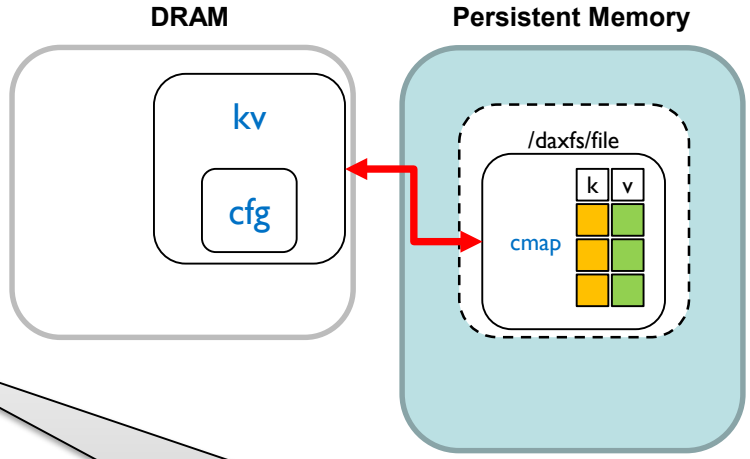
```
config cfg;

status s =
    cfg.put_string("path", "/daxfs/file");
assert(s == status::OK);

s = cfg.put_uint64("size", SIZE);
assert(s == status::OK);

db *kv = new db();
```



db object – volatile object for managing engine

pmemkv design

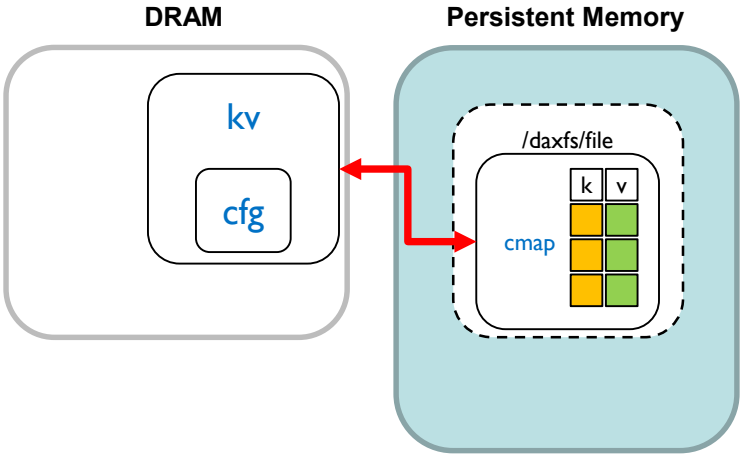
```
config cfg;

status s =
    cfg.put_string("path", "/daxfs/file");
assert(s == status::OK);

s = cfg.put_uint64("size", SIZE);
assert(s == status::OK);

db *kv = new db();

if (kv->open("cmap", cfg) != status::OK) {
    std::cerr << db::errormsg() << std::endl;
    return 1;
}
```



kv.open()
- creates/opens persistent memory pool
- checks consistency and perform recovery
- takes ownership for cfg structure

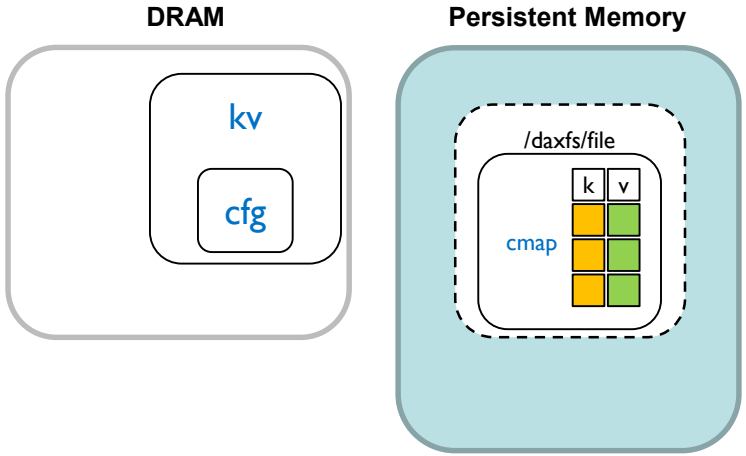
pmemkv design

```
config cfg;

status s =
    cfg.put_string("path", "/daxfs/file");
assert(s == status::OK);

s = cfg.put_uint64("size", SIZE);
assert(s == status::OK);

db *kv = new db();

if (kv->open("cmap", cfg) != status::OK) {
    std::cerr << db::errormsg() << std::endl;
    return 1;
}

// do work here
```



pmemkv design

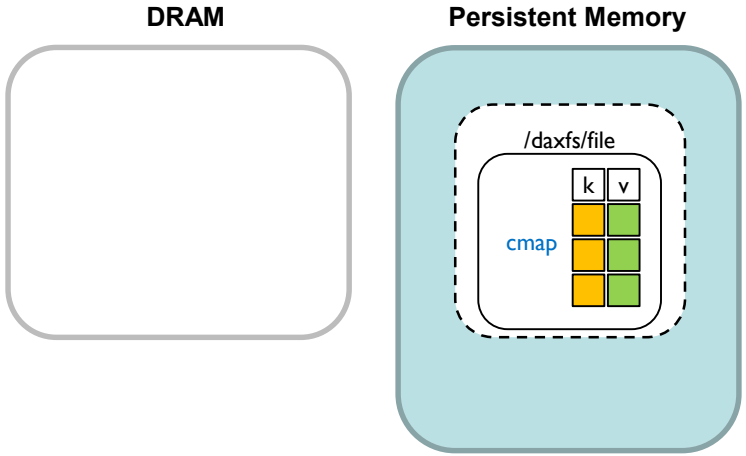
```
config cfg;

status s =
    cfg.put_string("path", "/daxfs/file");
assert(s == status::OK);

s = cfg.put_uint64("size", SIZE);
assert(s == status::OK);

db *kv = new db();

if (kv->open("cmap", cfg) != status::OK) {
    std::cerr << db::errormsg() << std::endl;
    return 1;
}

// do work here

kv->close();
```



kv.close()
- close database connection
- Persistent Memory data remain saved

pmemkv design

```
config cfg;

status s =
    cfg.put_string("path", "/daxfs/file");
assert(s == status::OK);

s = cfg.put_uint64("size", SIZE);
assert(s == status::OK);

db *kv = new db();

if (kv->open("cmap", cfg) != status::OK) {
    std::cerr << db::errormsg() << std::endl;
    return 1;
}

// do work here

kv->close();

delete kv;
```



Safe deletion of volatile data

Engines

Engines

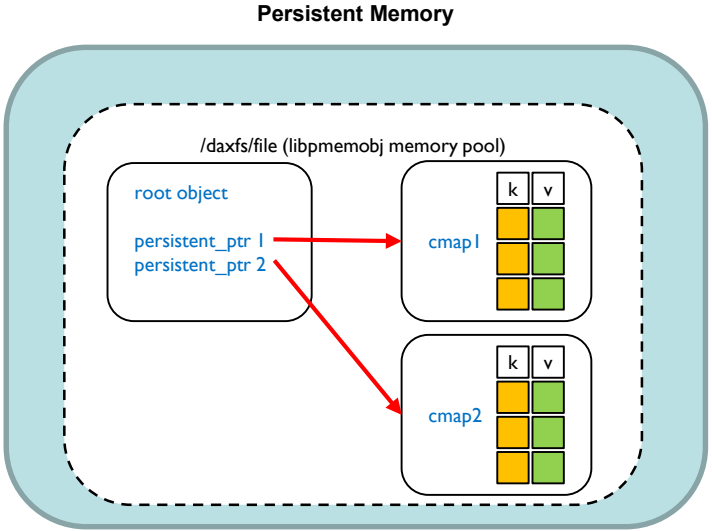
Engine Name	Description	Experimental?	Persistent?	Concurrent?	Sorted?
blackhole	Accepts everything, returns nothing	No	-	-	-
cmap	Concurrent hash map	No	Yes	Yes	No
vsmap	Volatile sorted hash map	No	No	No	Yes
vcmap	Volatile concurrent hash map	No	No	Yes	No
tree3	Persistent B+ tree	Yes	Yes	No	No
stree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

Engines

- Engine contributions are welcome!
- Types:
  - ordered/unordered
  - persistent/volatile
  - concurrent/single threaded
- Engines are optimized for different workloads & capabilities
- All engines work with all language bindings
- Generic tests for engines include:
  - memcheck
  - helgrind/drd
  - pmemcheck
  - pmemreorder

Multiple Engines Within the Same Memory Pool

- pmemkv does not limit you to a single engine to a single memory pool

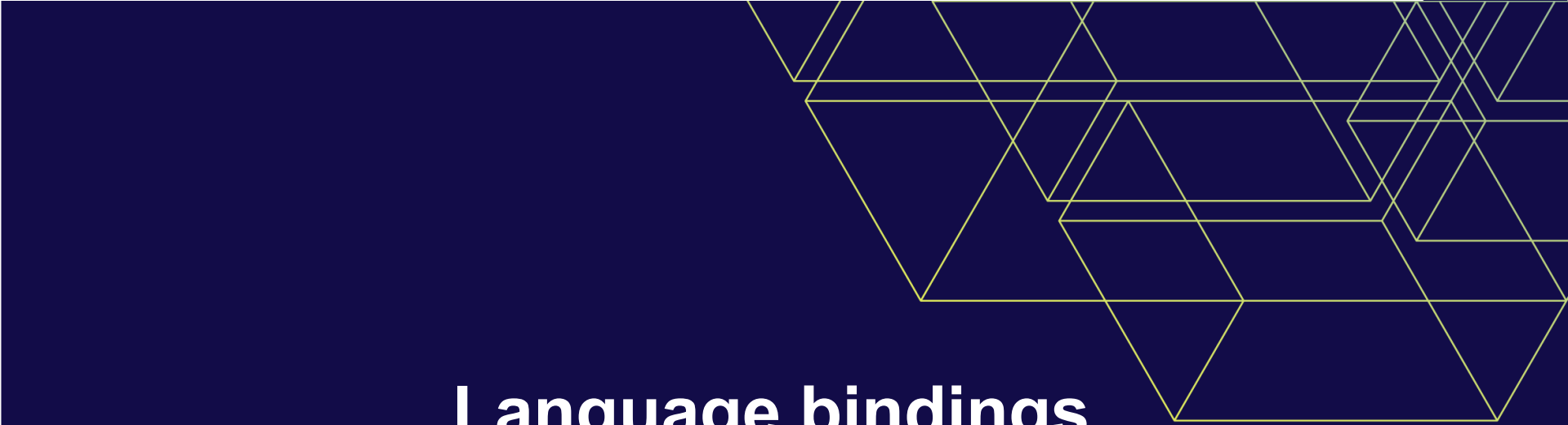


- Engines are reachable from root object

Multiple Engines Within the Same Memory Pool


```
struct Root {
            pmem::obj::persistent_ptr<PMEMoid> ptr1;
            pmem::obj::persistent_ptr<PMEMoid> ptr2;
};
// libpmemobj setup here
config cfg_1;
config cfg_2;
status ret = cfg_1.put_object("oid", &(pop.root()->ptr1), nullptr);
assert(ret == status::OK);
ret = cfg_2.put_object("oid", &(pop.root()->ptr2), nullptr);
assert(ret == status::OK);

db *kv_1 = new db();
status s = kv_1->open("cmap", std::move(cfg_1));
assert(s == status::OK);

db *kv_2 = new db();
s = kv_2->open("cmap", std::move(cfg_2));
assert(s == status::OK);
```

Prototyped API for using pmemkv with libpmemobj++ simultaneously (implementation work ongoing)



Language bindings

Language bindings

A Simple API mean:

easy to add high-level language bindings with low performance overhead

- Currently 4 available language bindings for pmemkv:
  - Java https://github.com/pmem/pmemkv-java
  - NodeJS https://github.com/pmem/pmemkv-nodejs
  - Ruby https://github.com/pmem/pmemkv-ruby
  - Python https://github.com/pmem/pmemkv-python

Simple API

API

- Well understood key-value API
  - Nothing new to learn
  - Inspired by rocksDB and levelDB
- Life-cycle API
  - open()/close()
- Operations API
  - put(key, value)
  - get(key, value/v_callback)
  - remove(key)
  - exists(key)
- other
  - errormsg()
- Iteration API
  - count_all()
  - get_all(kv_callback)
- range versions of above for ordered engines
  - below/above/between

pmemkv is Simple!

C++ example

```
config cfg;
// setup config here

status ret = kv.open("cmap", cfg);
assert(ret == status::OK);

ret = kv.put("John", "123-456-789");
assert(ret == status::OK);

std::string number;
ret = kv.get("John", &number);
assert(ret == status::OK);

ret = kv.get_all([](string_view name, string_view num) {
        std::cout << name.data() << " " << num.data() << std::endl;
    });
assert(ret == status::OK);

assert(kv.exists("John") == status::OK);

ret = (kv.remove("John");
assert(ret == status::OK);

kv.close();
```

Get value by copying to DRAM

Direct access

pmemkv is Simple!

NodeJS example

```
const db = new Database('cmap', '{"path":"/daxfs/kvfile","size":1073741824}');

db.put('John', '123-456-789');

assert(db.get('John') === '123-456-789');

db.get_all((k, v) => console.log(`name: ${k}, number: ${v}`));

db.remove('John');

assert(!db.exists('John'));

db.stop();
```

- Similar simplicity for other high-level language bindings

Latencies and performance

Latencies and performance

- Language bindings
  - number of round trips between high-level language & native code
  - Create high-level object (string, byte[], reference type, callback/closure)
  - Translate bytes to UTF-8
  - String interning, reference counting or GC

- pmemkv core (native code)
  - Searching indexes in DRAM
  - Updating indexes in DRAM
  - Managing transactions
  - Allocating persistent memory

- Persistent Memory
  - HW read and write latency

- Performance varies based on traffic pattern
  - Contiguous 4 cacheline (256B) granularity vs. single random cacheline (64B) granularity
  - Read vs. writes

Latencies and performance

cmap performance

- pmemkv_tools is a separate github repository with benchmark tool inspired by db_bench
  - https://github.com/pmem/pmemkv-tools

- Test results for cmap (persistent concurrent hashmap)
  - Throughput scales with a number of threads
  - P99 latency – flat

- Quote from a not-ready-for-prime-time porting effort:

  The performance numbers are just incredible. Using our persistence engine with full durability its running at ~80% the speed of RAM.

Questions?