VolipMem: A System-Level PMEM Runtime

Jana Toljaga

Nicolas Derumigny, Tara Aggoun, Mathieu Bacou, Gaël Thomas

Benagil, Inria Saclay

Télécom SudParis, Institut Polytechnique de Paris

Presented by:

Jana Toljaga

23.05.2024.

Introduction

Persistent memory

Durable, efficient and valuable

- Persistence = resilience to power outage and software failures
- Promising for large databases and big data analytics
- Efficient: byte-addressable and durable with direct loads and stores

Technology

- Intel Optane DC
- New generation supporting CXL

```
user_a.balance -= 100;
user_b.balance += 100;
```

crash

How to provide consistency?

Failure Atomic Sections (FAS)

- A section of code with all-or-nothing semantics
- Logging: mandatory to restore a consistent state after a crash

Revert the first write

```
pstart();
user_a.balance -= 100;
user_b.balance += 100;
pend();
```

crash

Failure Atomic Sections (FAS)

- A section of code with all-or-nothing semantics
- Logging: mandatory to restore a consistent state after a crash
- **Write set:** modified locations in a failure atomic section

Logging is complex!

write set:

```
pstart();
LOG(user_a);
LOG(user_b);
user_a.balance -= 100;
user_b.balance += 100;
pend();
```

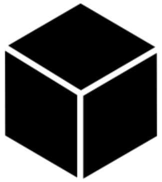
crash

Failure Atomic Sections (FAS)

- A section of code with all-or-nothing semantics
- Logging: mandatory to restore a consistent state after a crash
- **Write set:** modified locations in a failure atomic section

write set is unknown →

```
pstart();
LOG(?);
```



```
pend();
```

Logging is complex!

← crash

Reusing legacy code is almost impossible!

Transparency of logging

- **Write-ahead** redo or undo logging: write set specified manually by application developers

- Logging **embedded in the language**: write set collected wrapping objects with generics or annotations, using operator overloading, etc.

PMDK

```
void insert(MEMobjpool *pop, TOID(struct list) list,
↪  uint64_t key) {
  struct list *head = D_RW(list);
  TX_BEGIN(pop) {
    node = TX_NEW(struct node);
    TOID_TYPEOF(elm) *node_ptr = D_RW(node);
    node_ptr->key = key;
    elm_ptr->next = (head)->first;
    TX_ADD_DIRECT(head);
    (head)->first = node_ptr;
  }
  TX_END
}
```

not transparent
not reusable

Romulus

```
struct Node {
  persist<K> key;
  persist<Node*> next;
  Node(const K& key): key{key},next{nullptr} {}
};
void insert(persist<Node*>& head, const K& key) {
  Romulus::update_transaction([&] () {
    Node* n = Romulus::alloc<Node>(key);
    head->next = n;
    n->next = head;
  });
}
```

transparent
not reusable

What about using hardware?

- Code instrumentation can be avoided using hardware mechanisms

Copy-on-write
- memory protection mechanism
- interrupt handling

PMThreads

PAGE TABLE

Dirty inspection
- dirty bits stored in page table entries

Page Map Logging (PML)
- write-set tracking at the level of a virtual machine

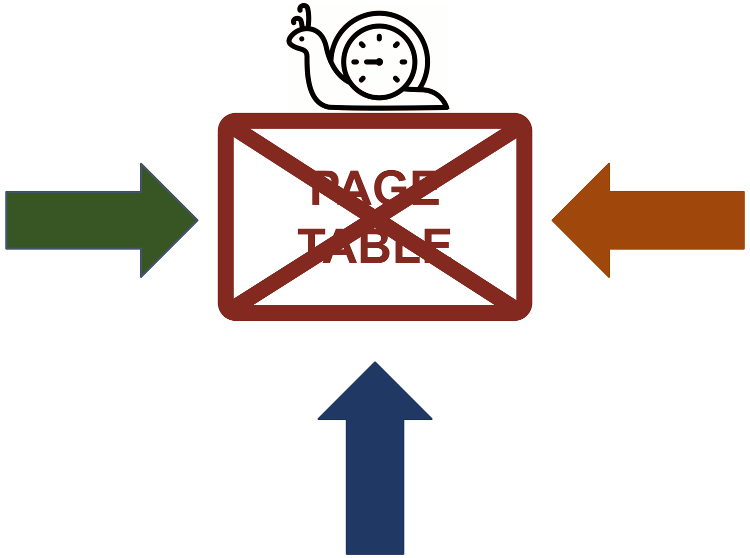
What about using hardware?

- Code instrumentation can be avoided using hardware mechanisms
- **Problem:** Hardware is accessible only through slow **system primitives**

Copy-on-write

- memory protection mechanism
- interrupt handling

PMThreads



Dirty inspection

- dirty bits stored in page table entries

Page Map Logging (PML)

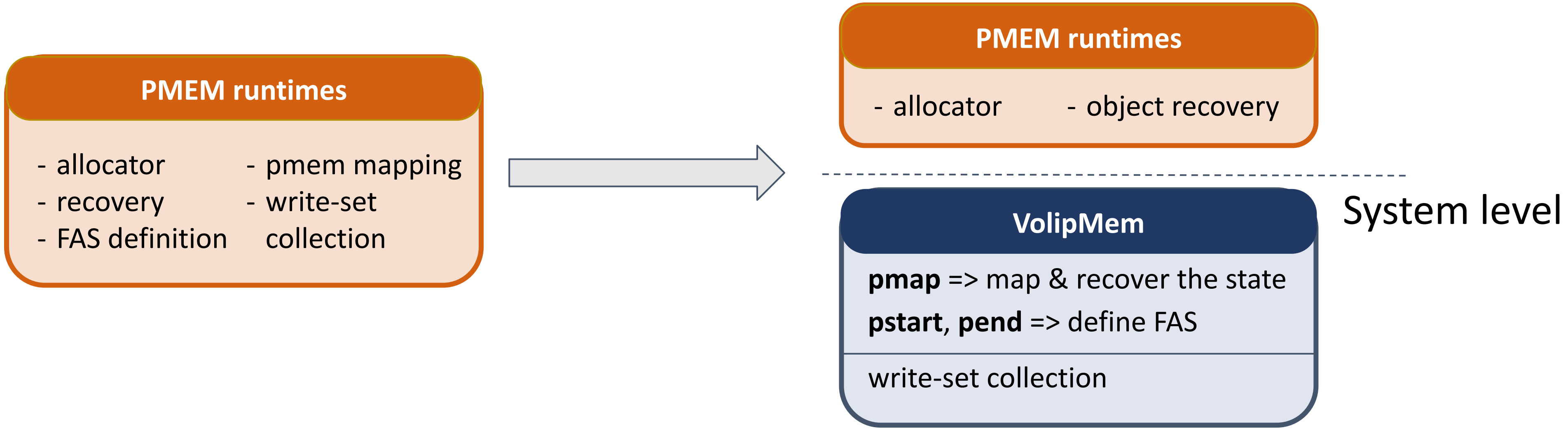
- write-set tracking at the level of a virtual machine

We need new system primitives

VolipMem

Extracting the core of PMEM runtimes into new primitives:

pmap, pstart, pend



VolipMem

Inria

```
pstart();
LOG(user_a);
LOG(user_b);
user_a.balance -= 100;
user_b.balance += 100;
pend();
```

```
pstart();
user_a.balance -= 100;
user_b.balance += 100;
pend();
```

Application level

PMEM runtimes

- allocator
- recovery
- FAS definition
- pmem mapping
- write-set collection

PMEM runtimes

- allocator
- object recovery

System level

VolipMem

pmap => map & recover the state

pstart, pend => define FAS

write-set collection

VolipMem

`pstart();`
~~`LOG(user_a);`~~
~~`LOG(user_b);`~~
`user_a.balance -= 100;`
`user_b.balance += 100;`
`pend();`

→

```
pstart();
user_a.balance -= 100;
user_b.balance += 100;
pend();
```

Application level

PMEM runtimes

- allocator
- recovery
- FAS definition
- pmem mapping
- write-set collection

→

PMEM runtimes

- allocator
- object recovery

System level

VolipMem

pmap => map & recover the state

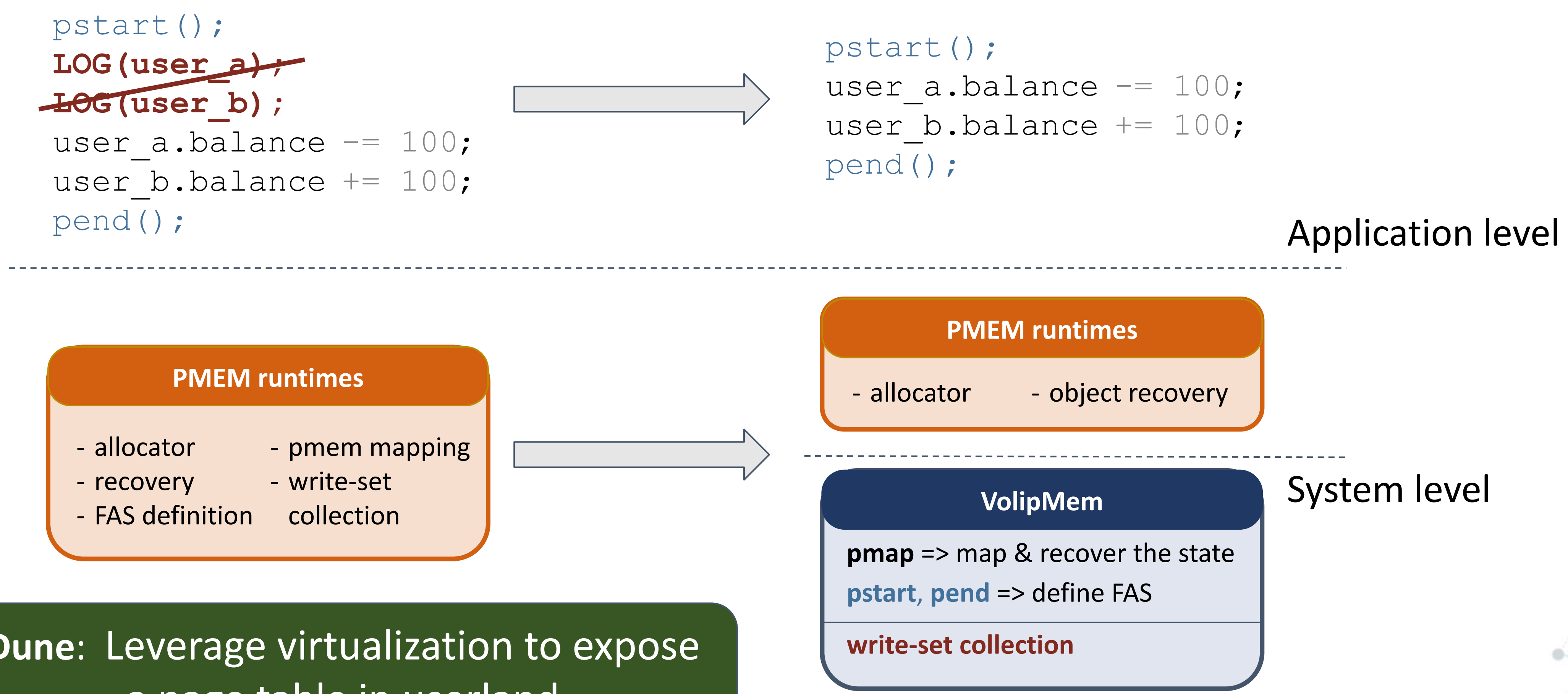
pstart, pend => define FAS

write-set collection

Engineering problem:

Modifying OS paging system is difficult

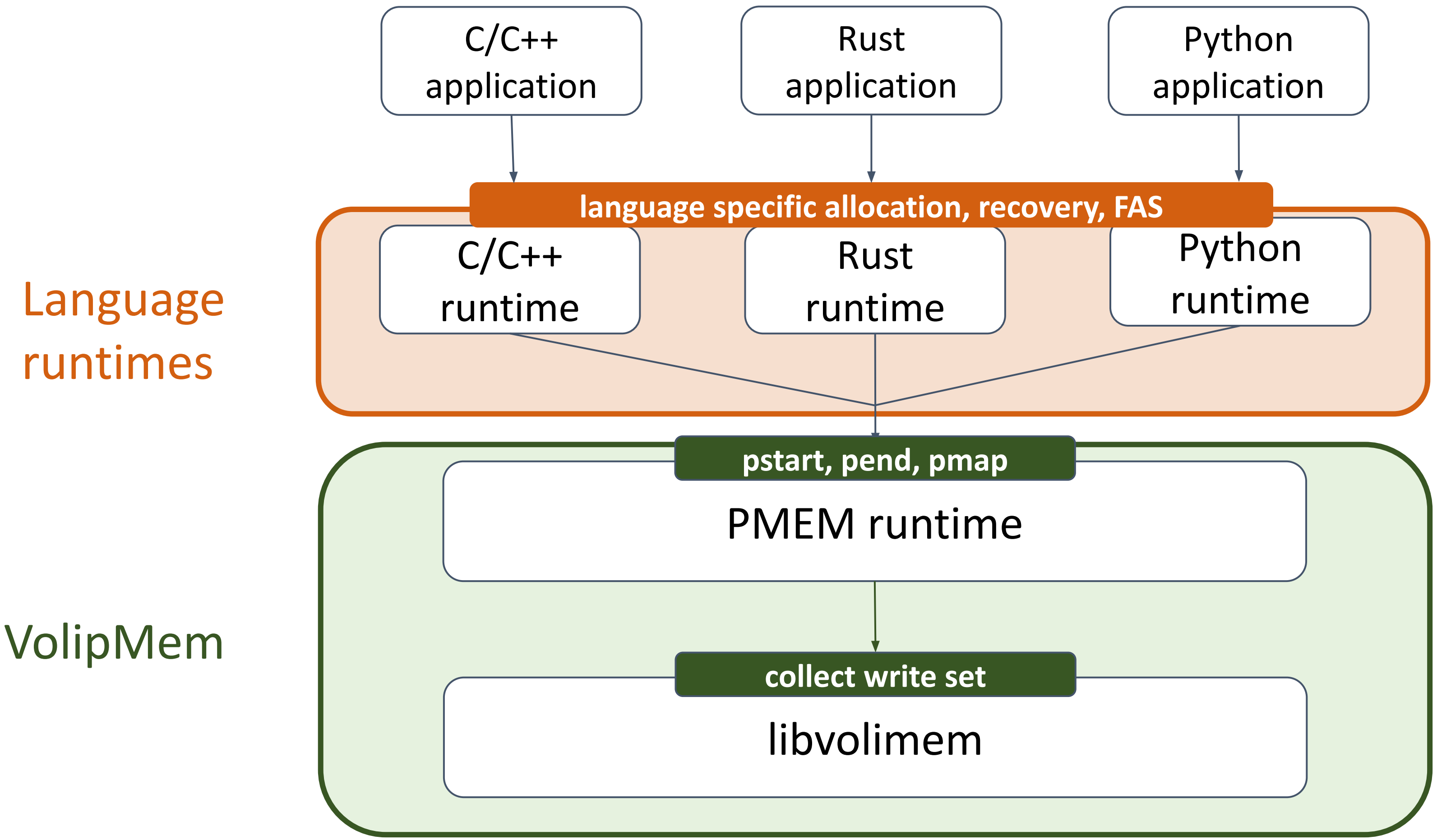
VolipMem



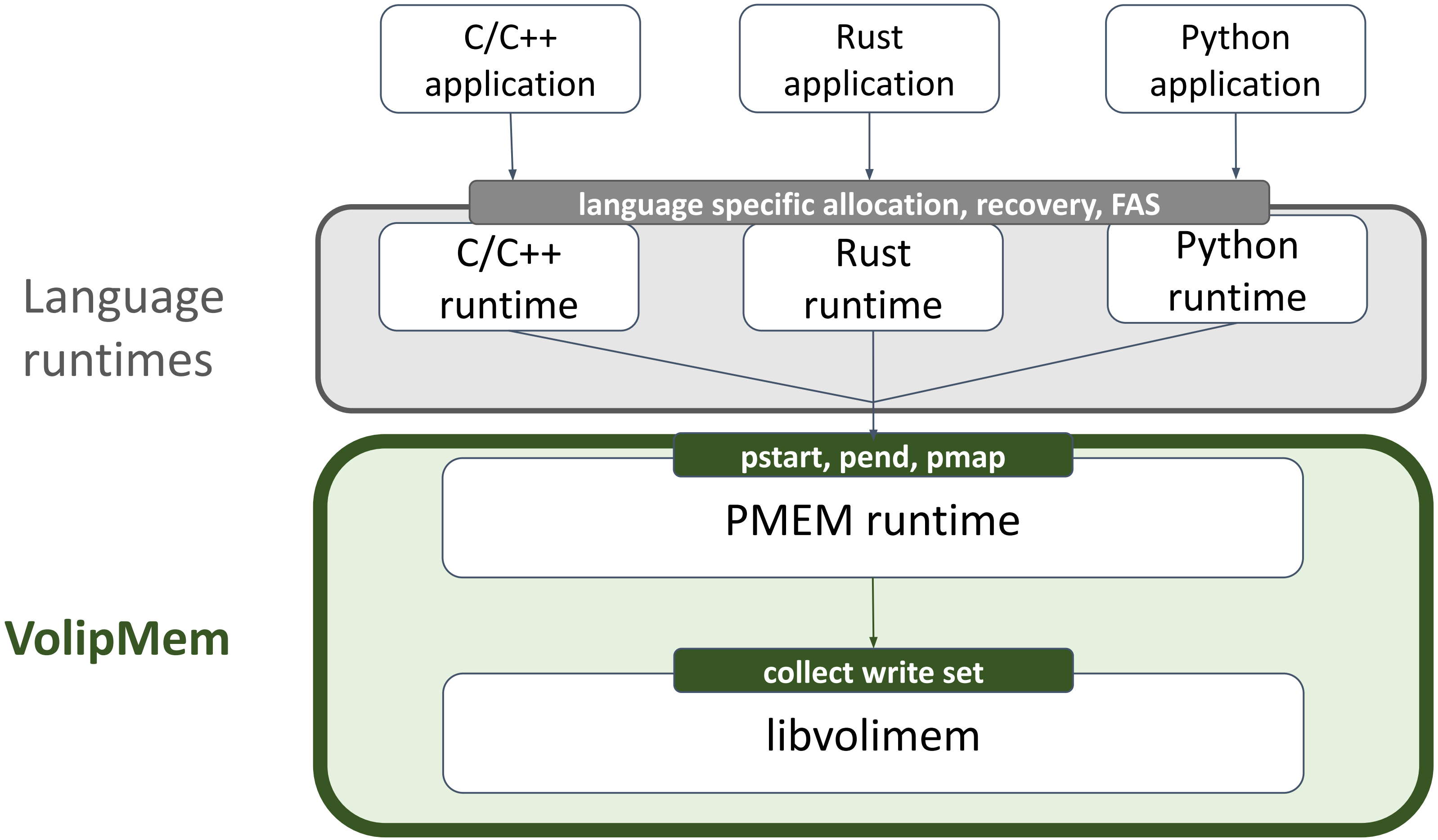
Dune: Leverage virtualization to expose a page table in userland

Design & Implementation

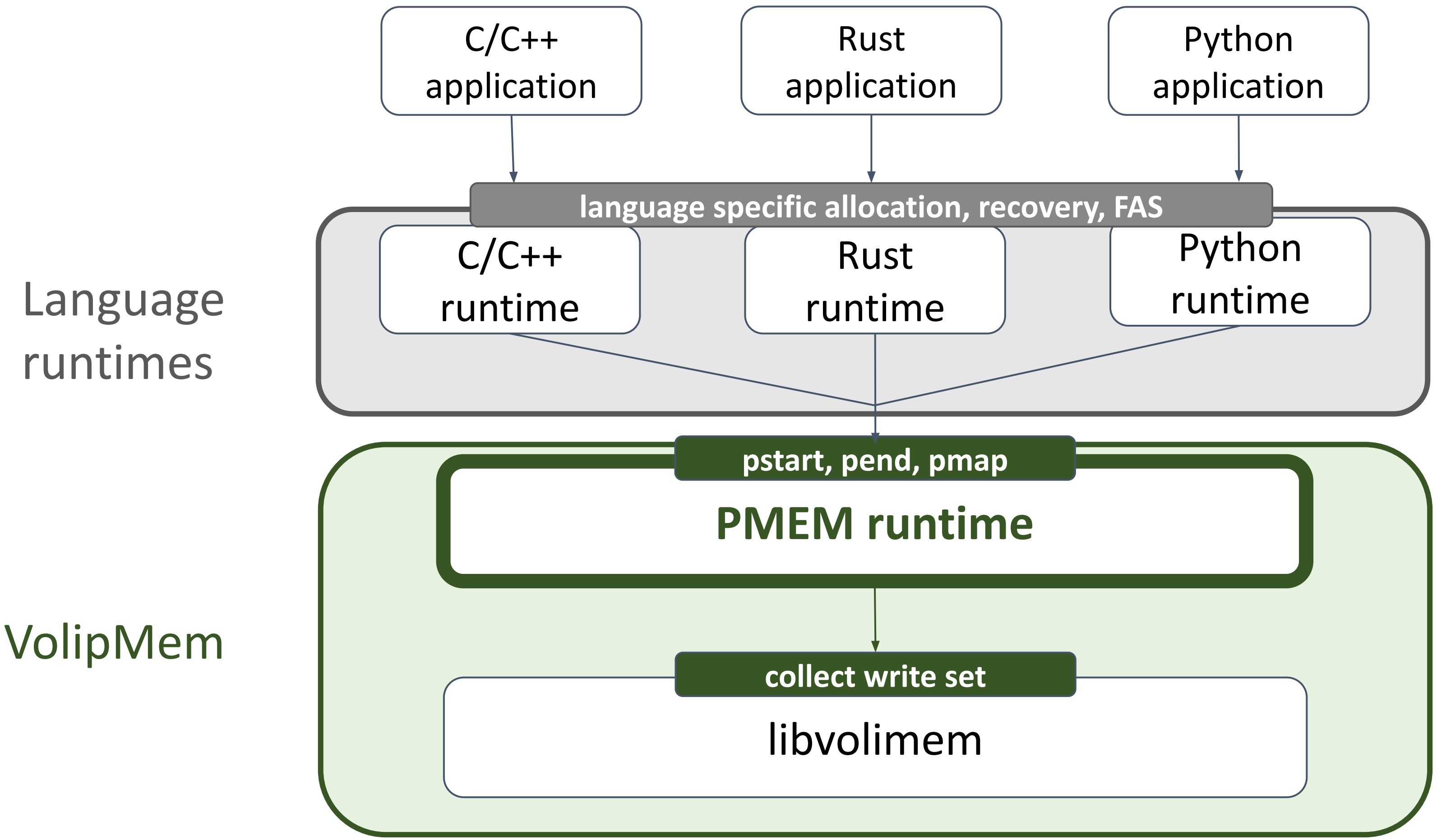
Overall design



Overall design

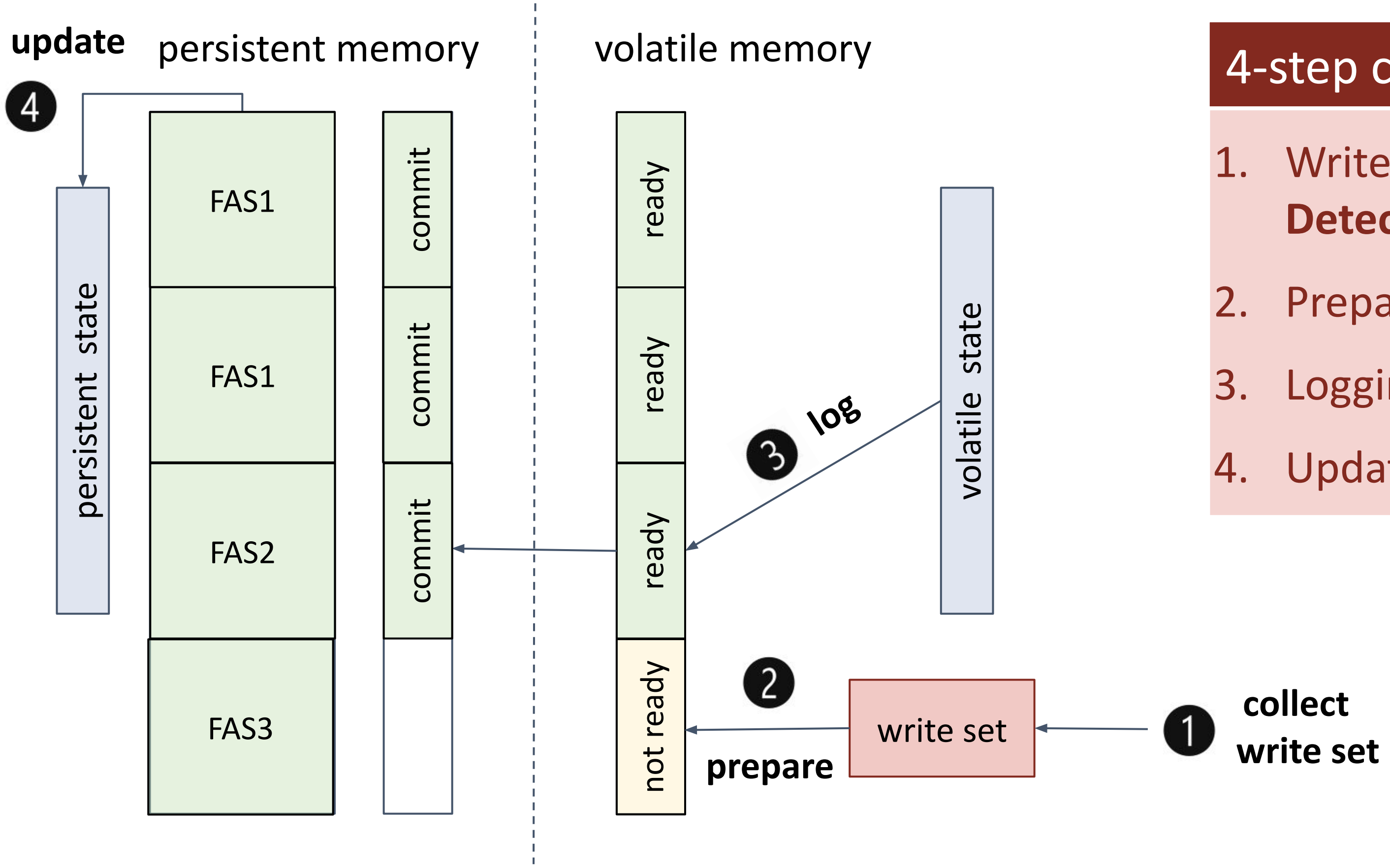


VolipMem


C/C++ application
Rust application
Python application
language specific allocation, recovery, FAS
C/C++ runtime
Rust runtime
Python runtime
Language runtimes
pstart, pend, pmap
PMEM runtime
collect write set
libvolimem
VolipMem

PMEM runtime: commit

- Environment that provides consistent PMEM operations through three primitives



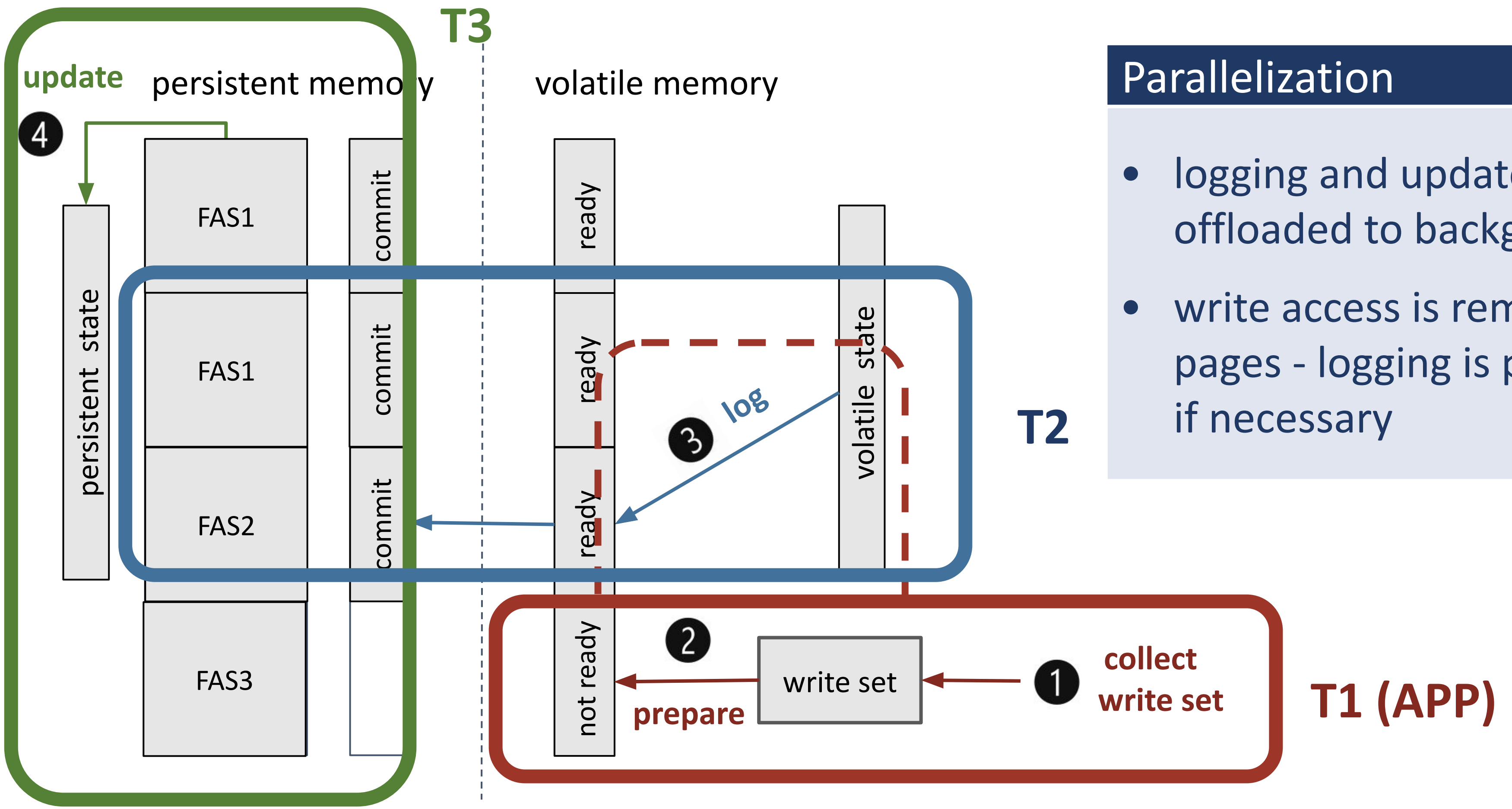
4-step commits

1. Write set collection relies on hardware: **Detect-On-Write** and **Dirty Bit Inspection**
2. Preparation: occupies log entry for each dirty page
3. Logging: copies log content from volatile
4. Update: copies modification to PMEM state

```
pstart();
user_a.balance -= 100;
user_b.balance += 100;
pend();
```

PMEM runtime: parallel configurations

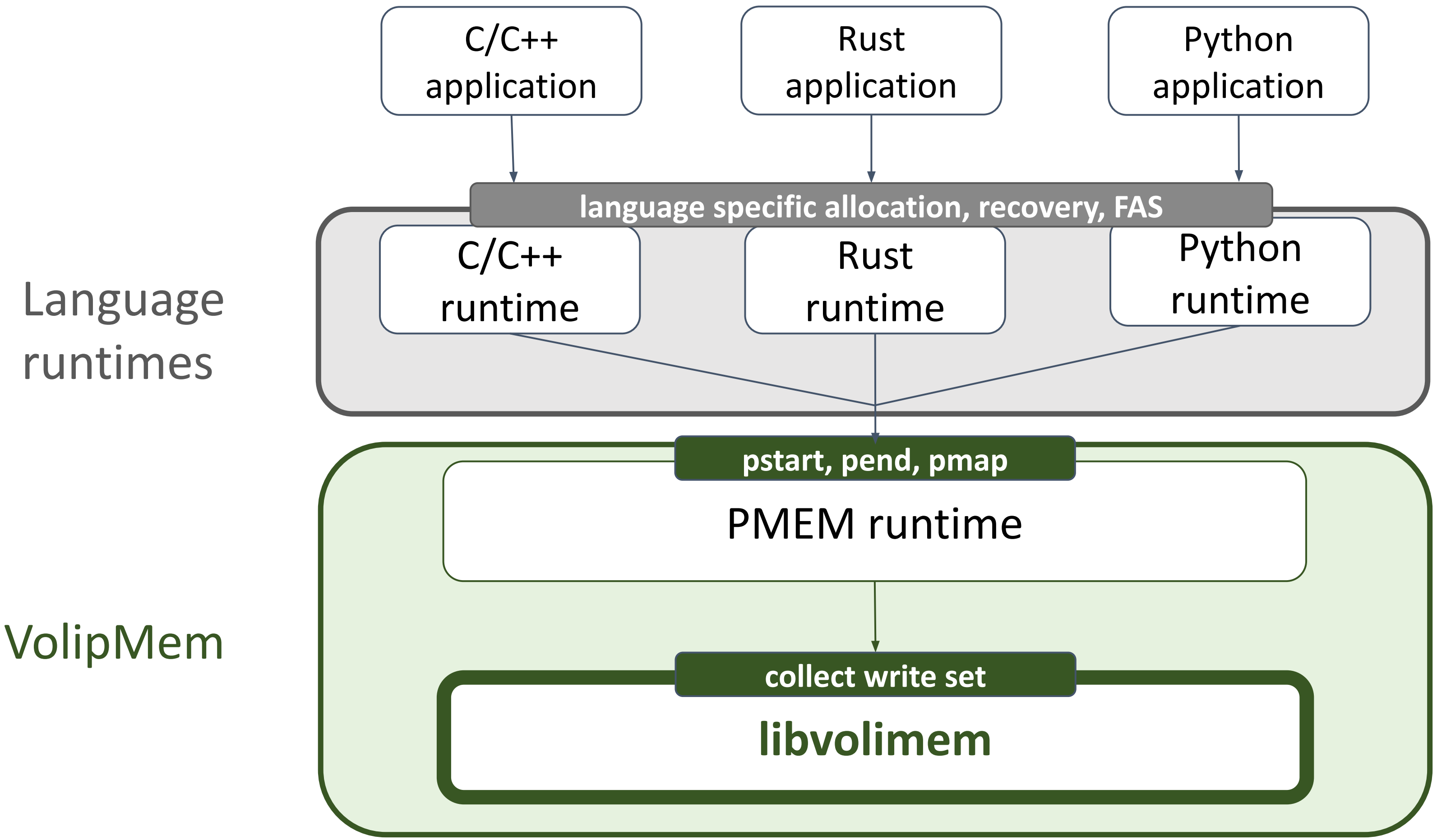
- Environment that provides consistent PMEM operations through three primitives



Parallelization

- logging and update phases can be offloaded to background threads
- write access is removed for 'not-ready' pages - logging is performed by APP only if necessary

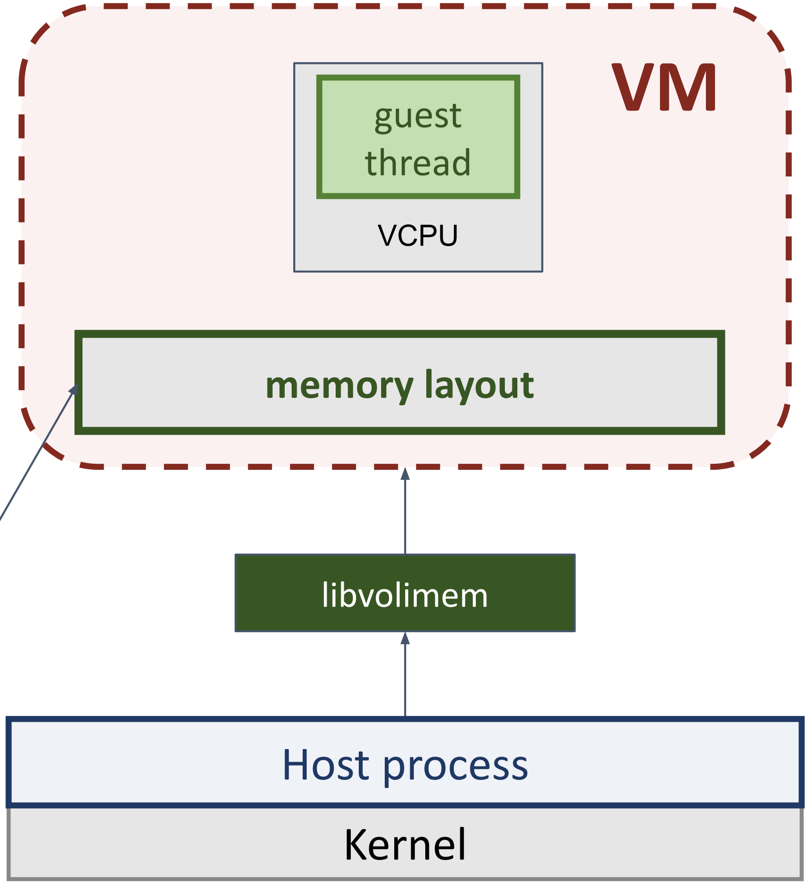
VolipMem


C/C++ application
Rust application
Python application
language specific allocation, recovery, FAS
Language runtimes
C/C++ runtime
Rust runtime
Python runtime
pstart, pend, pmap
VolipMem
PMEM runtime
collect write set
libvolimem

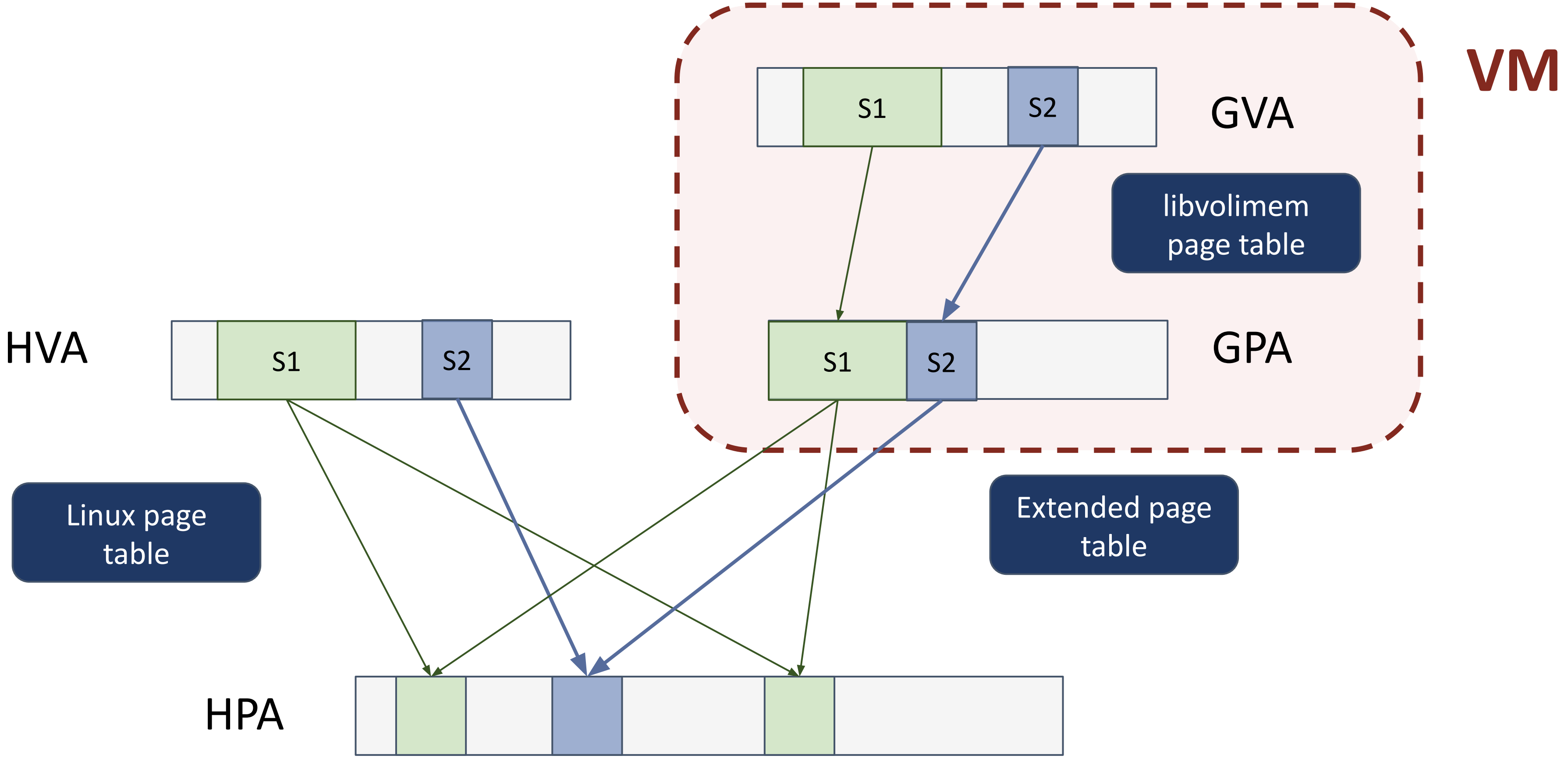
Libvolimem

- Library that transforms any process into a lightweight virtual machine (VM)
- Advantage: fast fault processing inside VM
- Each **thread** on the host = a **VCPU** in VM
- Libvolimem implements a layer to efficiently forward system calls to the host

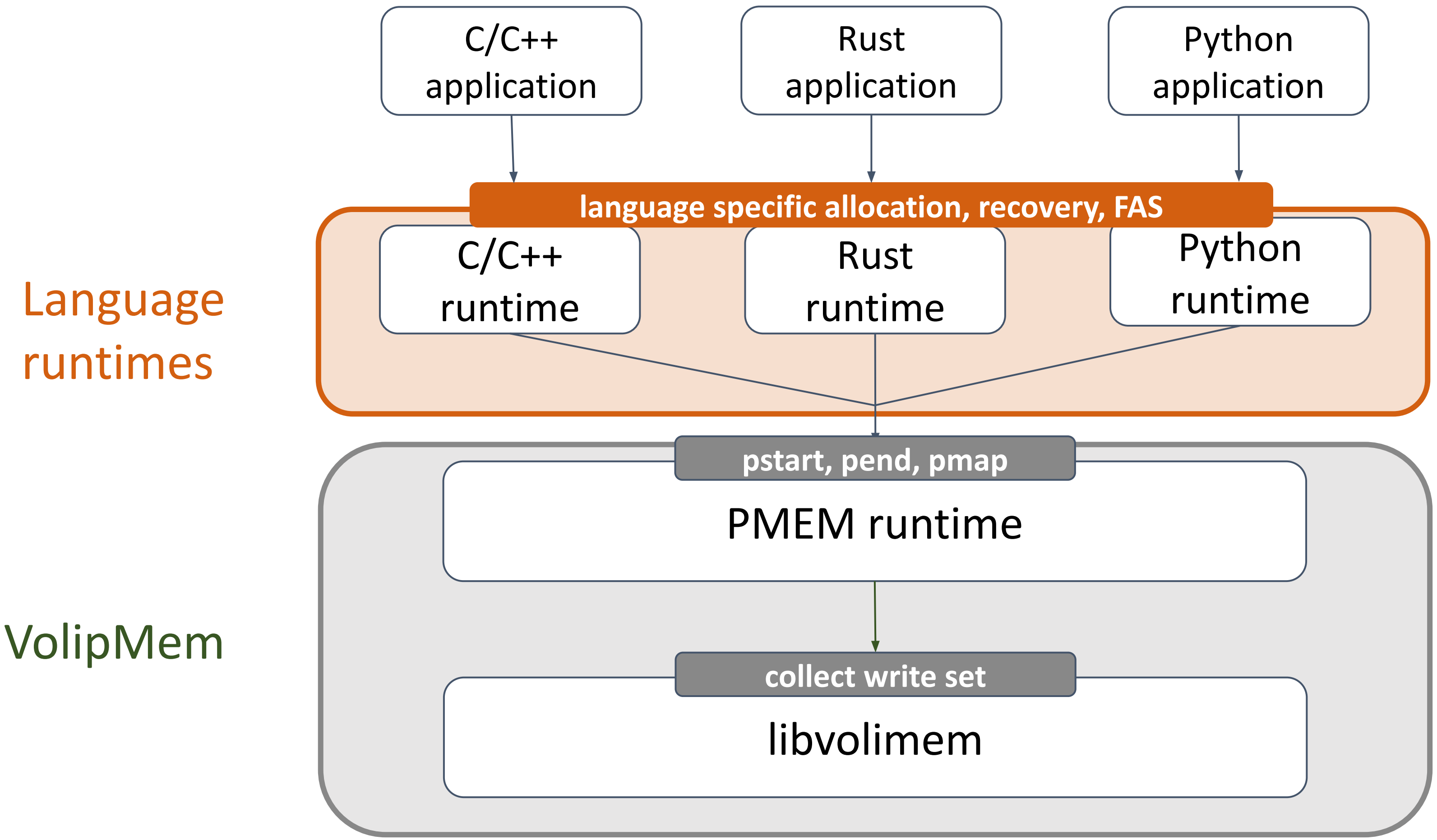
GOAL: Exposing a page table in user space to collect a write set without modifying Linux paging system



Memory layout



Overall design



Language runtimes

- C/C++, Python and Rust - different programming models based on language properties
- **Safety** - no dangling pointers to old volatile objects after a reboot
- **Ease of use** - legacy code is easy to reuse after switching to PMEM
- **Performance** - it is possible to limit number of PMEM objects during execution

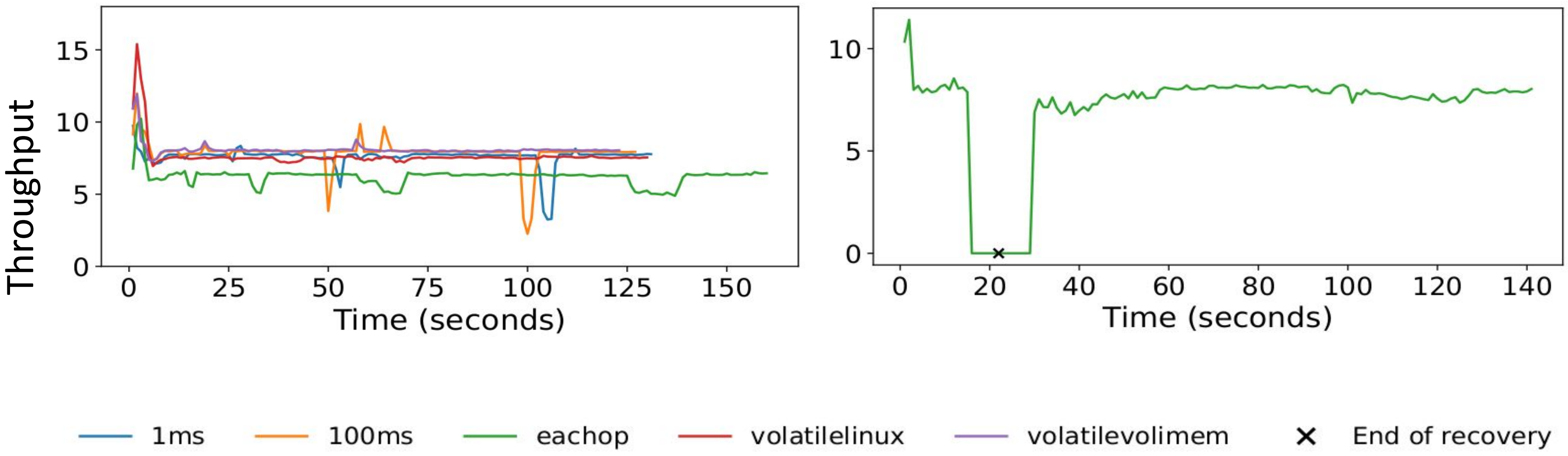
	Language philosophy			**PMEM interface**		
	Python	C/C++	Rust	Python	C/C++	Rust
Safety	~	-	+	+	-	+
Easy of use	+	~	-	+	~	-
Performance	-	+	+	-	+	+

Evaluation

Memcached

Load phase

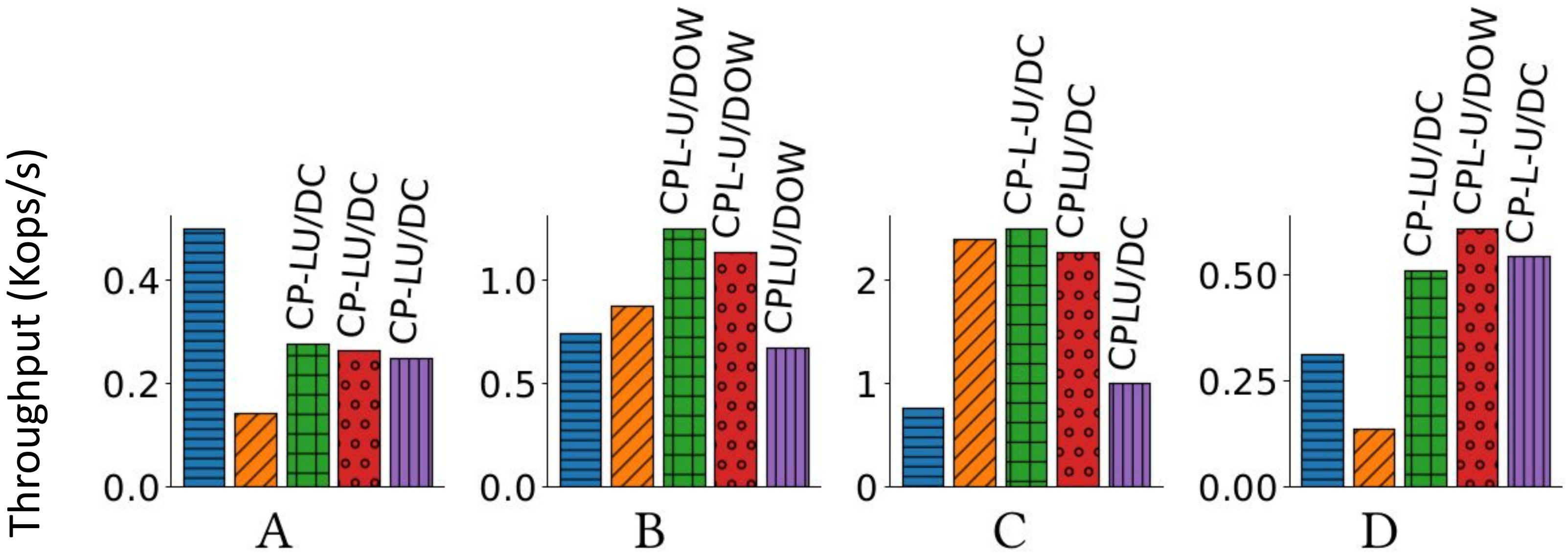
Run phase - recovery



YCSB A uniform distribution

- VolipMem reduces performance at most 20% comparing to native memcached
- When period of commits is higher (>1ms), VolipMem is at the same scale

Hashmap



	PMDK	VolipMem	Rust	PMThread	VoliCPP
Code	PMDK	PMDK	Rust	PMDK	C++
Runtime	PMDK	VolipMem	VolipMem	PMThread	VolipMem

- VolipMem is up to x4 faster than PMThreads

Conclusion

VolipMem: A system-level interface for PMEM

- Leverages hardware to collect the write set of a failure-atomic section
- Extracts core PMEM features with three system primitives: **pstart, pend, pmap**
- Implemented by leveraging virtualization
- VolipMem is **generic**:
  - Integrated into three programming languages, libraries and applications
- VolipMem is **efficient**:
  - Up to more than **4x** faster than PMThreads (transparent)
  - At the same scale as PMDK (manual logging)

Python dictionary

```
import root from volipmem
if not "hashmap" in root:
    hashmap={}
    hashmap[1]="1"
    root["hashmap"] = hashmap
else:
    hashmap=root["hashmap"]
    hashmap[2]="2"
self.pcommit()
```

Appendix

Standard C++ linked list

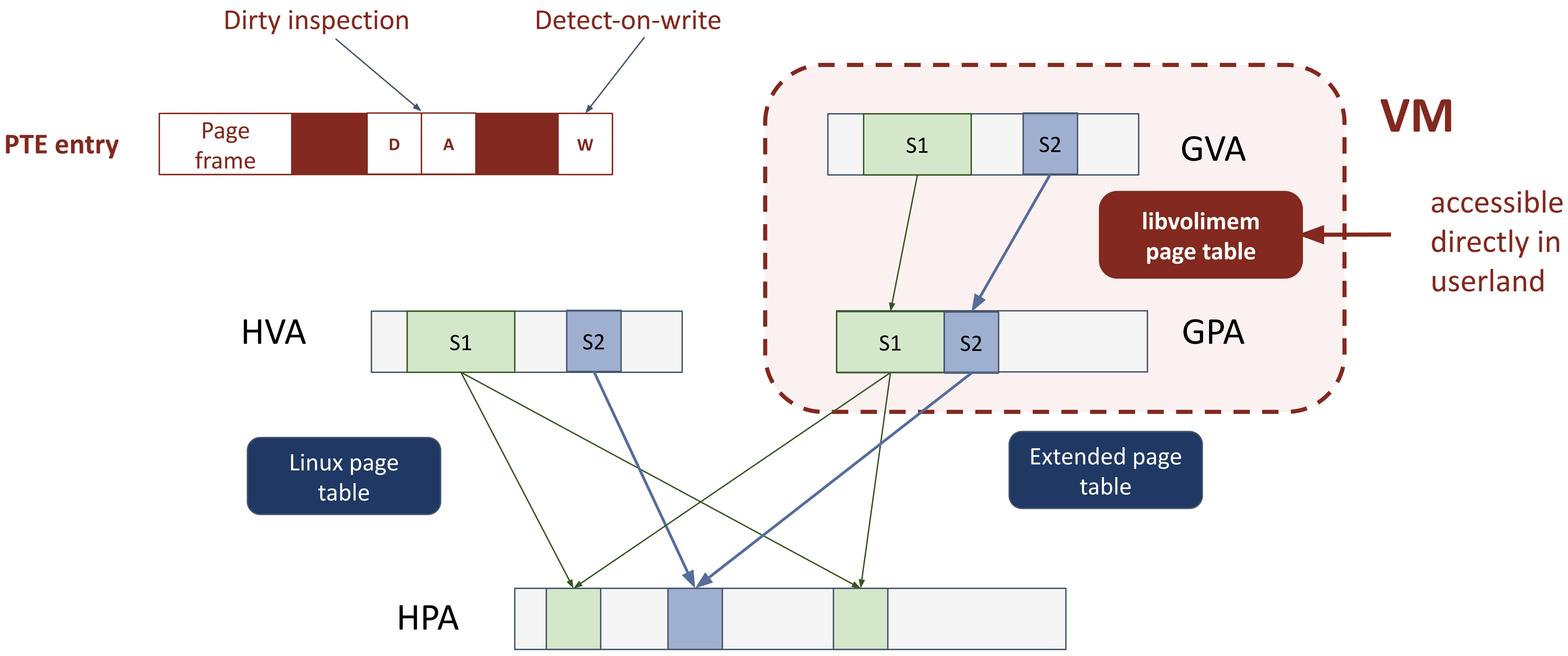
```
std::list<K>* create_list() {
    return provide_pobject<std::list<K>>("list");
}

void insert(std::list<K>& myList, K& element) {
    pmem_api::pstart();
    myList.push_front(element);
    pmem_api::pend();
}
```

Standard Rust vector

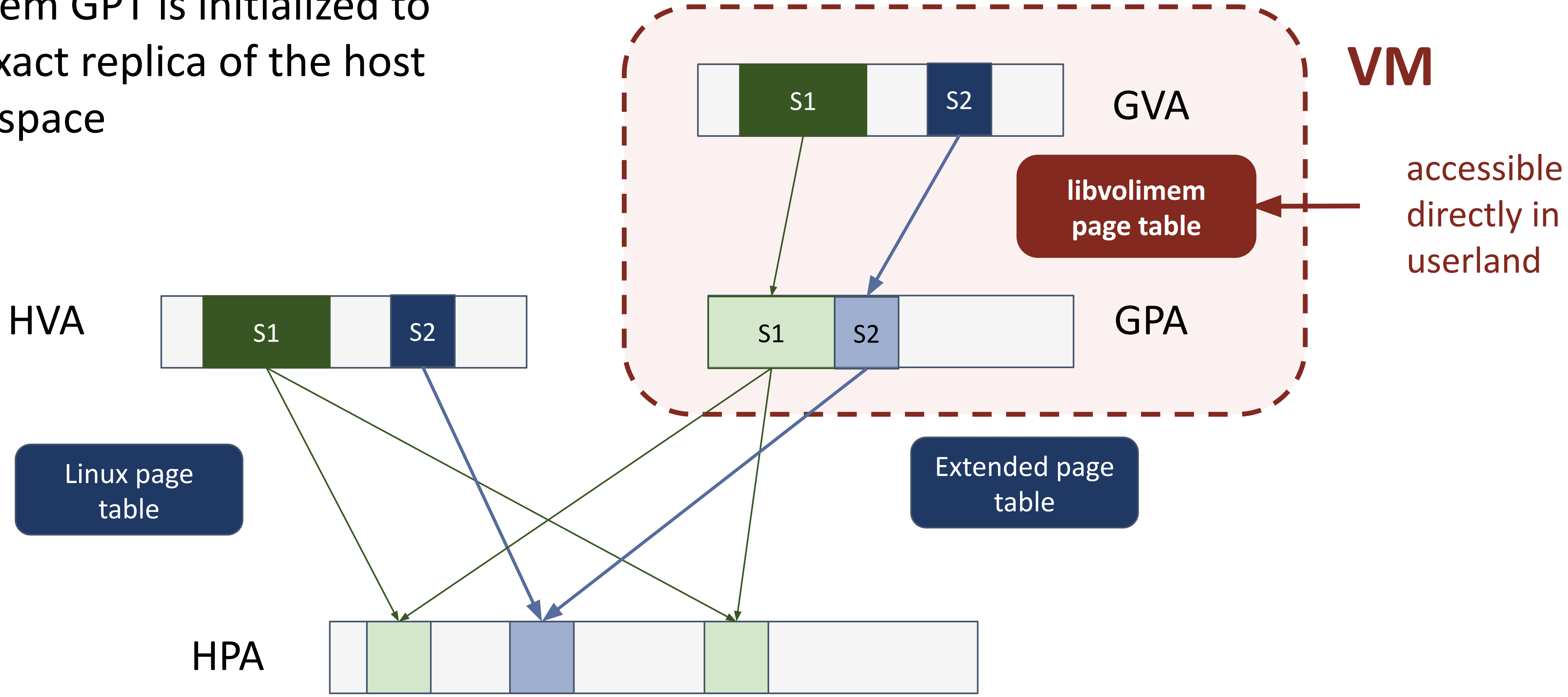
```
use volipmem::{PmemAllocator, s_pcommit};
type PVec<T> = Vec<T, PmemAllocator>;
type PBoxedVec<T> = Box<PVec<T>, PmemAllocator>;
...
let mut alloc = PmemAllocator::new();
let ret = alloc.retrieve_root::<PVec<u32>>();
let mut bvec: PBoxedVec<u32>;
if ret.is_none() {
    let mut v: Vec<u32> = Vec::new(); //volatile vector
    let bvec: PBoxedVec<u32> = PBoxedVec::new_in(v,
    ↪  alloc.clone()); //! compiler error
    bvec.push(1);
    alloc.save_root(&mut bvec);
    s_pcommit();
}
```

Memory layout

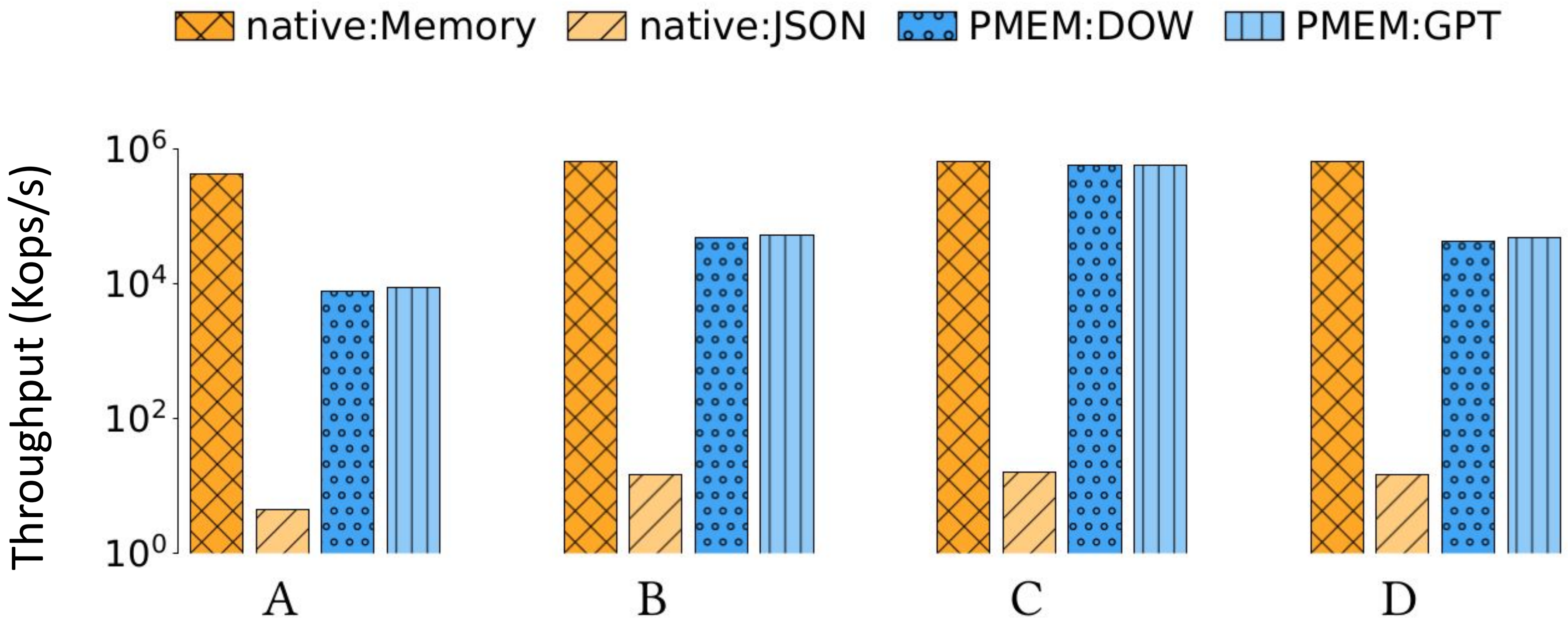


Memory layout

- Libvolimem GPT is initialized to create exact replica of the host address space



Python: Tiny DB



- Lightweight document oriented database implemented fully in Python
- VolipMem is between 2,200 to 40,000 faster than JSON backed DB