vNV-Heap: An Ownership-Based Virtually Non-Volatile Heap for Embedded Systems

June 17, 2025

Markus Elias Gerber[1] Luis Gerhorst[1] Ishwar Mudraje[2] Kai Vogelgesang[2]
Thorsten Herfet[2] Peter Wägemann[1]

[1] Friedrich-Alexander-University (FAU), Erlangen-Nürnberg, Germany
[2] Saarland Informatics Campus (SIC), Saarbrücken, Germany

Friedrich-Alexander-Universität
Faculty of Engineering

Saarland Informatics Campus

Supported by DFG
Project 502615015 (ResPECT), 502947440 (Watwa)

Energy-Harvesting Systems with Intermittent Power Supply

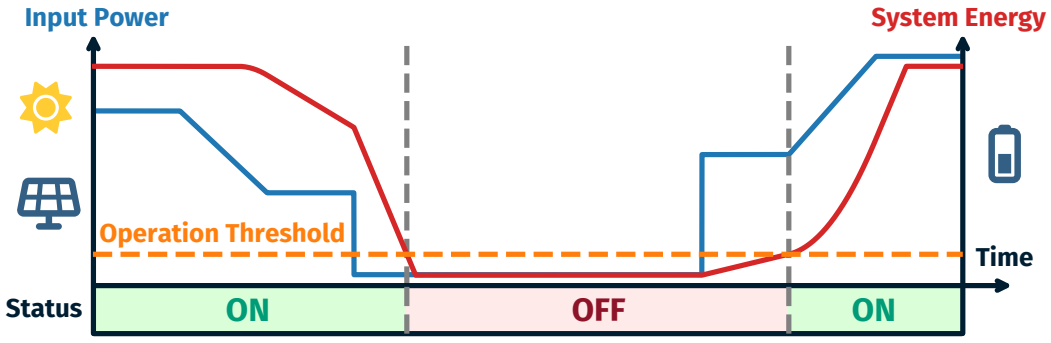
Observation Devices

Remote environmental sensor [1]

IoT Devices

Battery-free BLE smartwatch [3]

Intermittent Computing



- **Hard intermittency**: Device shuts down when *energy* $<$ *threshold*
- **State retention**: Persist state in time! ($\rightarrow$ predictability)

Large Amounts of State

- **Memory-intense applications**
  ($\rightarrow$ DNNs, LLMs)
- **Store work packages to be processed later**
  ($\rightarrow$ sensor data, IP packets)

$\Rightarrow$ **Commonly:** Explicitly allocated memory ($\rightarrow$ heaps)

State Retention Mechanism

Q *How do you persist state before the system shuts down?*

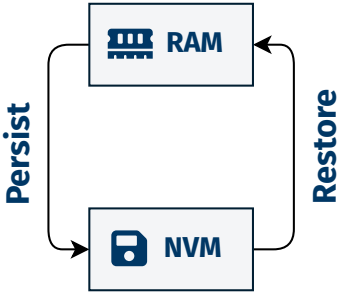
A **Copy data from volatile RAM to NVM**

Q *What data should you persist?*

A ~~Persist all data~~

A **Persist only modified data**

A **Limit amount of modified state**

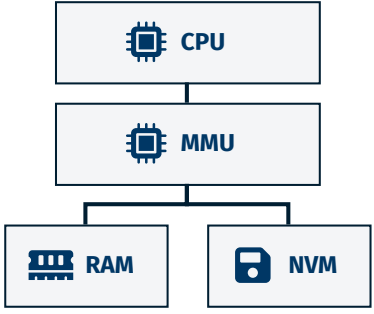


Takeaway #1

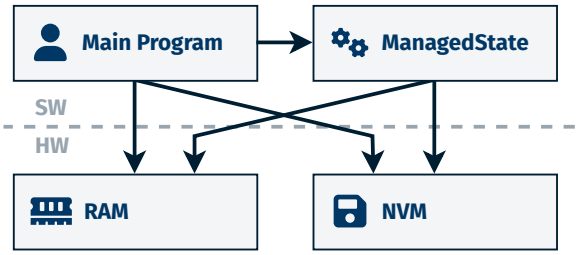
Precise modification tracking is required!

MMU-Based Approaches

	MMU-based
1. NVM safety?	✔
2. Transparent swap?	✔
3. Predictable?	✘
4. HW agnostic?	✘



ManagedState [2] as SW-Only Approach



Modifying Data

```
1 modify_start(ptr);
2 *ptr += 1; // valid
3 modify_end(ptr);
4
5 // integrity at risk
6 *ptr += 1;
```

[2] S. Sliper et al. **"Efficient State Retention through Paged Memory Management for Reactive Transient Computing".** In: DAC '19. ACM, June 2019

Comparison of Existing Approaches

	MMU-based	ManagedState	vNV-Heap
1. NVM safety?	✔	✘	✔
2. Transparent swap?	✔	✘	✔
3. Predictable?	✘	✔	✔
4. HW agnostic?	✘	✔	✔

vNV-Heap: virtually Non-Volatile Heap

Comparison of Existing Approaches

	MMU-based	ManagedState	vNV-Heap
1. NVM safety?	✔	✘	✔
2. Transparent swap?	✔	✘	✔
3. Pointer-based?	✘	✔	✔
4. HW agnostic?	✘	✔	✔

How does vNV-Heap achieve this?

vNV-Heap: virtually Non-Volatile Heap

Comparison of Existing Approaches

	MMU-based	ManagedState	vNV-Heap
1. NVM safety?	✔	✘	✔
2. Transparent swap?	✔	✘	✔
3. Fine-grained?	✘	✔	✔
4. HW agnostic?	✘	✔	✔

Idea: Leverage Modern Programming Languages!

vNV-Heap: virtually Non-Volatile Heap

Ownership & Borrowing

✔ **Zero-cost abstraction:** Statically checked

Ownership

- Each value has an owner
- There exists only one owner
- The value is dropped once the owner goes out of scope

Borrowing

There exists either:

- $= 1$ mutable reference or
- ≥ 0 immutable references

Memory Management using Ownership & Borrowing

Previous Example: Access and modify value

(a) Unsafe

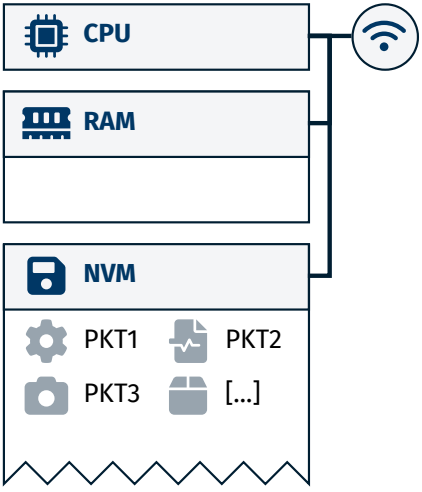
```
1 modify_start(ptr);
2 *ptr += 1; // valid
3 modify_end(ptr);
4
5 // integrity at risk
6 *ptr += 1;
```

(b) Ownership-Checked

```
1 ref = get_mut(handle);
2 *ref += 1; // valid
3 drop(ref);
4
5 // compilation error
6 *ref += 1;
```

⇒ **Overall:** Makes SW-based modification tracking & swapping safe!

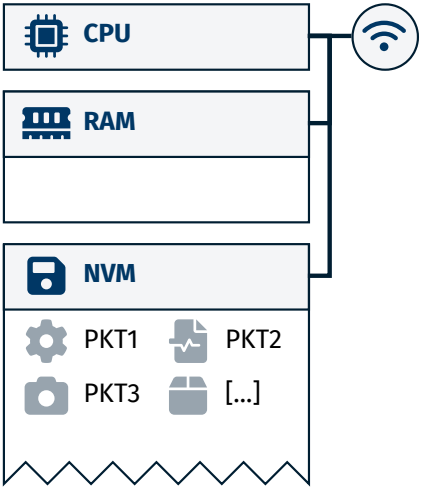
Practical Example



- `send_next_pkt();`

Max modified packets: 1

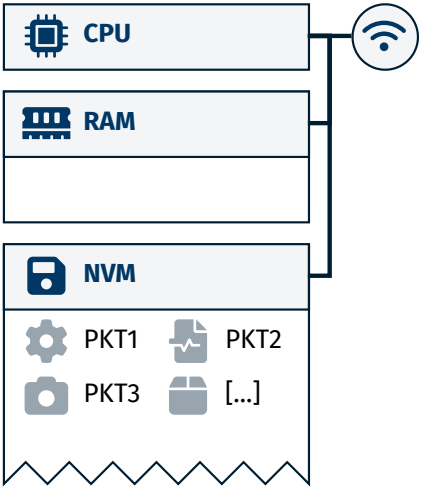
Practical Example




```
  send_next_pkt();
→ pkt1 = dequeue_pkt();
```

Max modified packets: 1

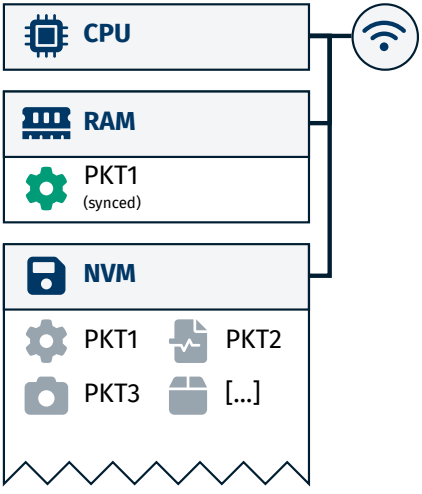
Practical Example




```
send_next_pkt();
→ pkt1 = dequeue_pkt();
→ ref1 = pkt1.get_mut();
```

Max modified packets: 1

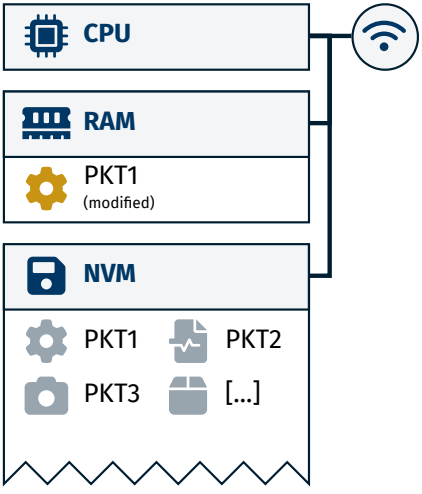
Practical Example




```
  send_next_pkt();
  → pkt1 = dequeue_pkt();
● → ref1 = pkt1.get_mut();
```

Max modified packets: 1

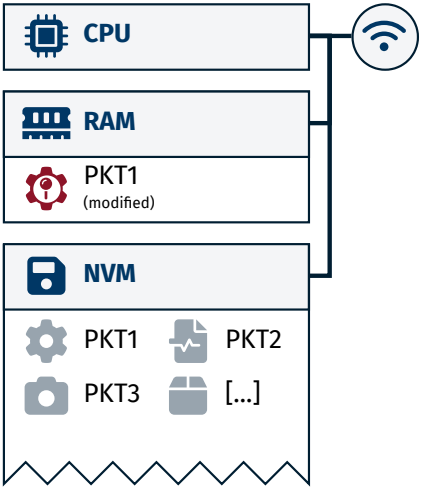
Practical Example




```
  send_next_pkt();
  → pkt1 = dequeue_pkt();
● → ref1 = pkt1.get_mut();
```

Max modified packets: 1

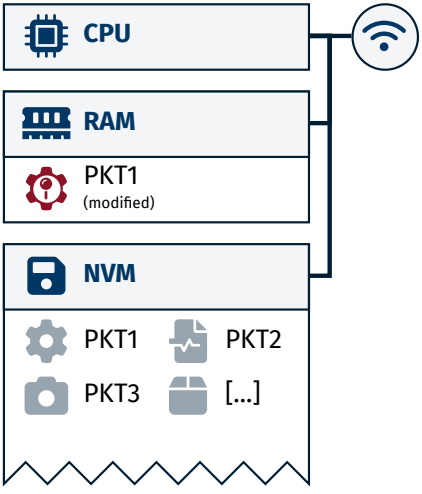
Practical Example




```
  send_next_pkt();
  → pkt1 = dequeue_pkt();
● → ref1 = pkt1.get_mut();
```

Max modified packets: 1

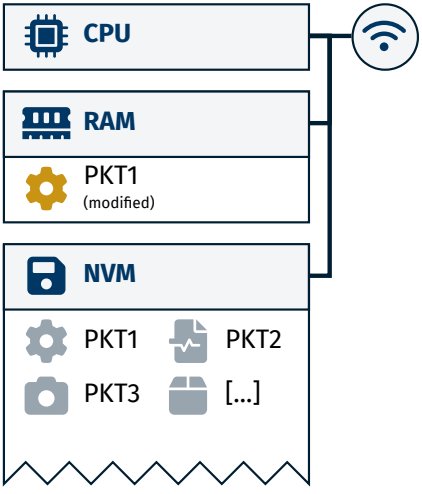
Practical Example




```
send_next_pkt();
→ pkt1 = dequeue_pkt();
→ ref1 = pkt1.get_mut();
→ ref1.crc = calc_crc(ref1);
```

Max modified packets: 1

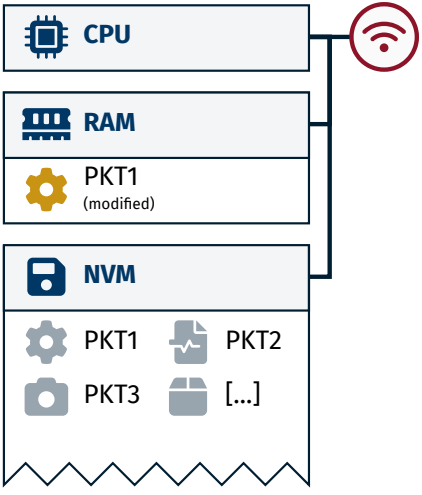
Practical Example




```
send_next_pkt();
→ pkt1 = dequeue_pkt();
→ ref1 = pkt1.get_mut();
→ ref1.crc = calc_crc(ref1);
→ drop(ref1);
```

Max modified packets: 1

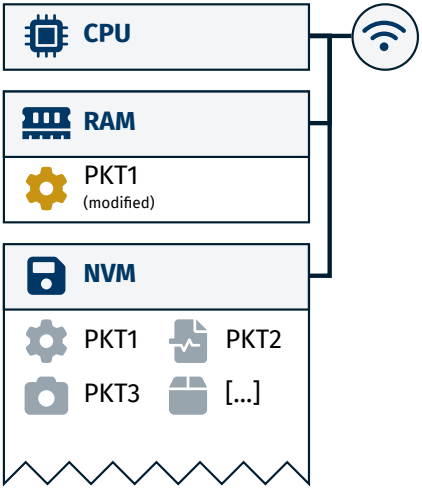
Practical Example




```
send_next_pkt();
→ pkt1 = dequeue_pkt();
→ ref1 = pkt1.get_mut();
→ ref1.crc = calc_crc(ref1);
→ drop(ref1);
→ send(pkt1);
```

Max modified packets: 1

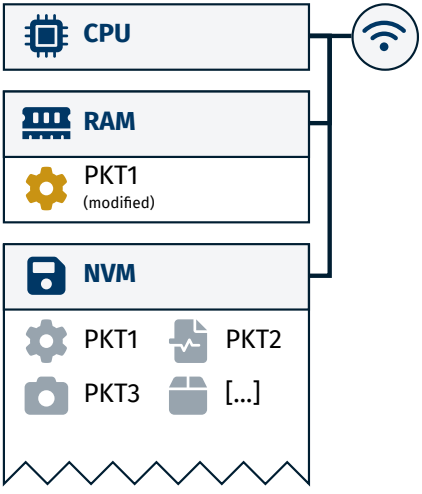
Practical Example




```
  send_next_pkt();
  → [...]
● send_next_pkt();
```

Max modified packets: 1

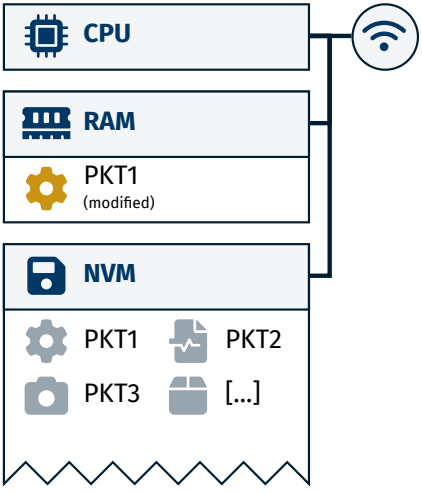
Practical Example




```
send_next_pkt();
→ [...]
send_next_pkt();
→ pkt2 = dequeue_pkt();
```

Max modified packets: 1

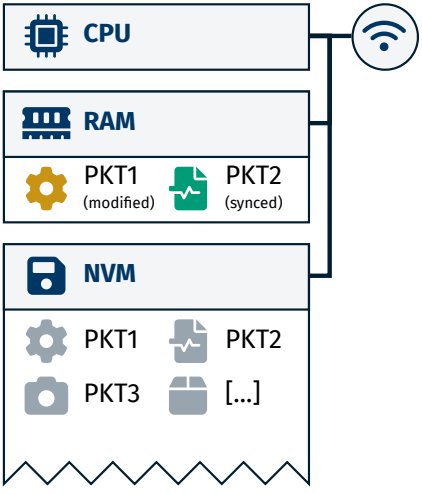
Practical Example




```
send_next_pkt();
→ [...]
send_next_pkt();
→ pkt2 = dequeue_pkt();
→ ref2 = pkt2.get_mut();
```

Max modified packets: 1

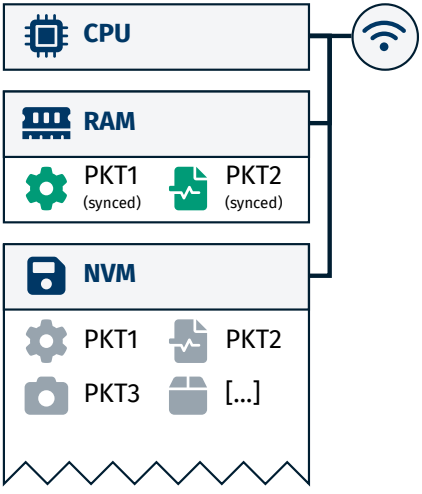
Practical Example




```
send_next_pkt();
→ [...]
send_next_pkt();
→ pkt2 = dequeue_pkt();
→ ref2 = pkt2.get_mut();
```

Max modified packets: 1

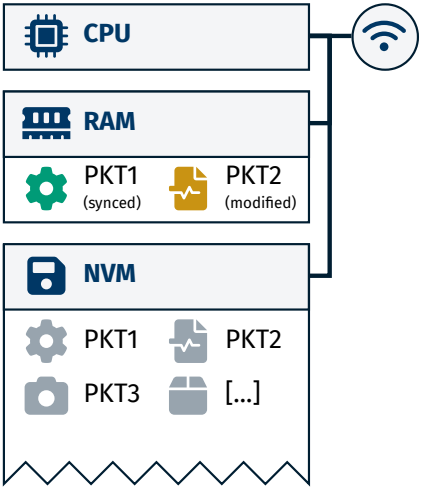
Practical Example




```
send_next_pkt();
→ [...]
send_next_pkt();
→ pkt2 = dequeue_pkt();
→ ref2 = pkt2.get_mut();
```

Max modified packets: 1

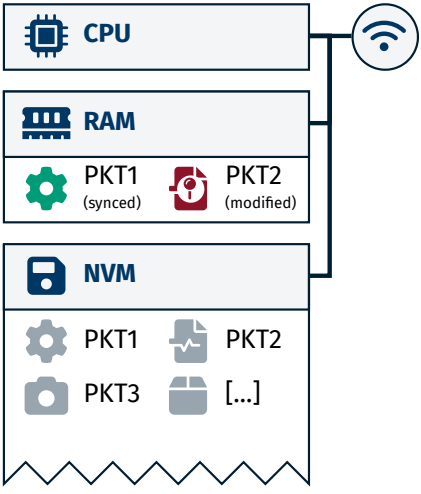
Practical Example




```
send_next_pkt();
→ [...]
send_next_pkt();
→ pkt2 = dequeue_pkt();
→ ref2 = pkt2.get_mut();
```

Max modified packets: 1

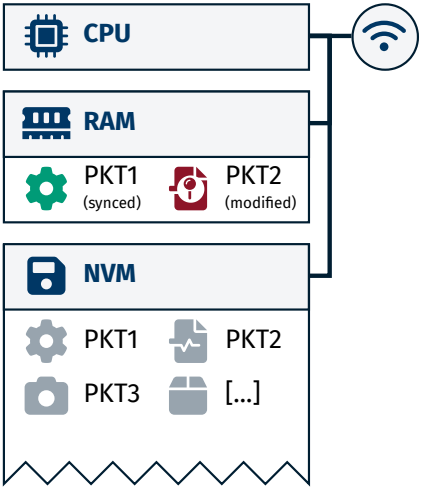
Practical Example




```
send_next_pkt();
→ [...]
send_next_pkt();
→ pkt2 = dequeue_pkt();
→ ref2 = pkt2.get_mut();
```

Max modified packets: 1

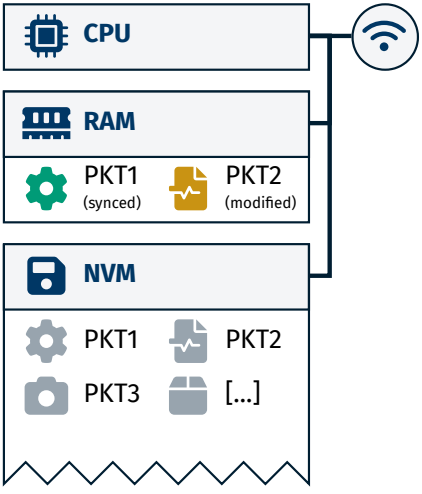
Practical Example




```
send_next_pkt();
→ [...]
send_next_pkt();
→ pkt2 = dequeue_pkt();
→ ref2 = pkt2.get_mut();
→ ref2.crc = calc_crc(ref2);
```

Max modified packets: 1

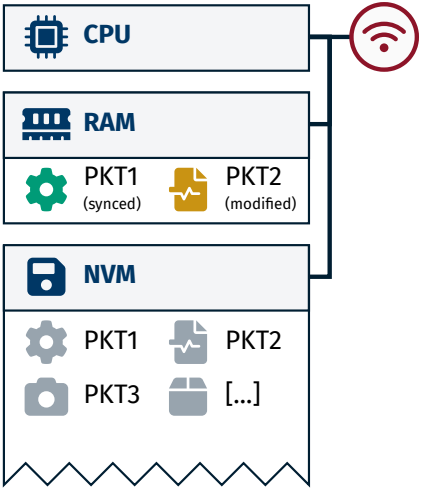
Practical Example




```
send_next_pkt();
→ [...]
send_next_pkt();
→ pkt2 = dequeue_pkt();
→ ref2 = pkt2.get_mut();
→ ref2.crc = calc_crc(ref2);
→ drop(ref2);
```

Max modified packets: 1

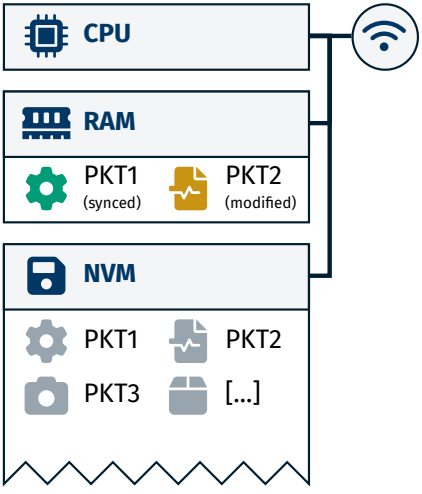
Practical Example




```
send_next_pkt();
→ [...]
send_next_pkt();
→ pkt2 = dequeue_pkt();
→ ref2 = pkt2.get_mut();
→ ref2.crc = calc_crc(ref2);
→ drop(ref2);
→ send(pkt2);
```

Max modified packets: 1

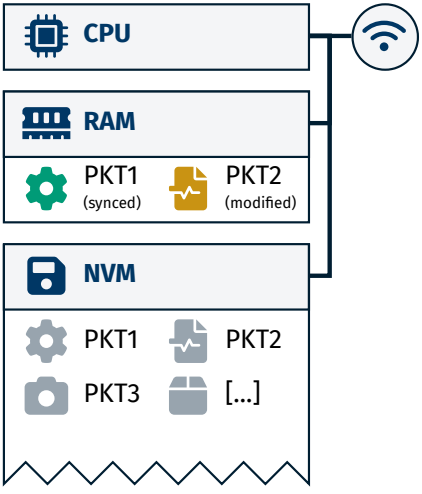
Practical Example




```
send_next_pkt();
→ [...]
send_next_pkt();
→ [...]
send_next_pkt();
```

Max modified packets: 1

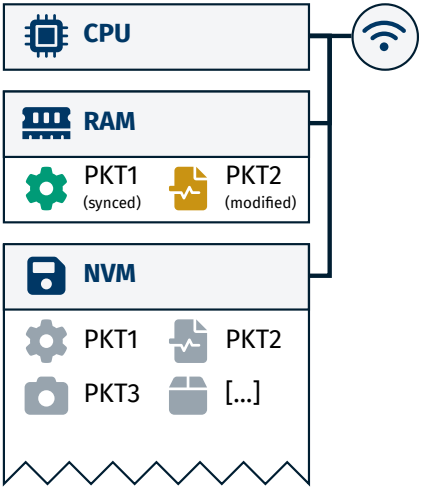
Practical Example




```
send_next_pkt();
→ [...]
send_next_pkt();
→ [...]
send_next_pkt();
→ pkt3 = dequeue_pkt();
```

Max modified packets: 1

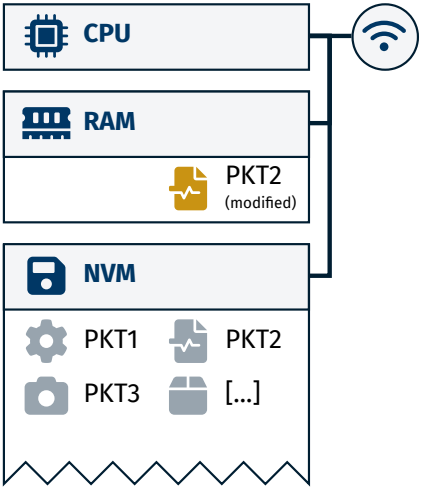
Practical Example




```
send_next_pkt();
→ [...]
send_next_pkt();
→ [...]
send_next_pkt();
→ pkt3 = dequeue_pkt();
→ ref3 = pkt3.get_mut();
```

Max modified packets: 1

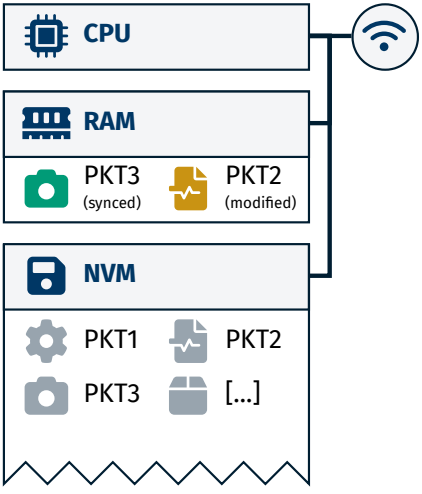
Practical Example




```
send_next_pkt();
→ [...]
send_next_pkt();
→ [...]
send_next_pkt();
→ pkt3 = dequeue_pkt();
→ ref3 = pkt3.get_mut();
```

Max modified packets: 1

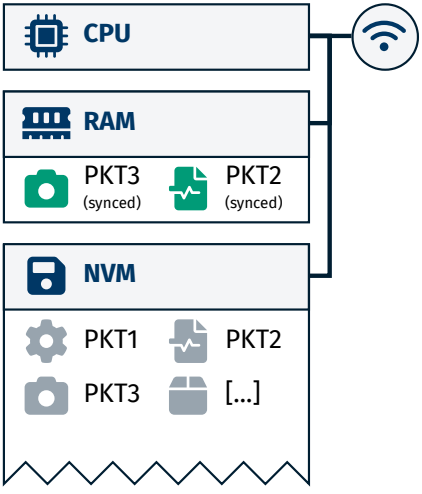
Practical Example




```
send_next_pkt();
→ [...]
send_next_pkt();
→ [...]
send_next_pkt();
→ pkt3 = dequeue_pkt();
→ ref3 = pkt3.get_mut();
```

Max modified packets: 1

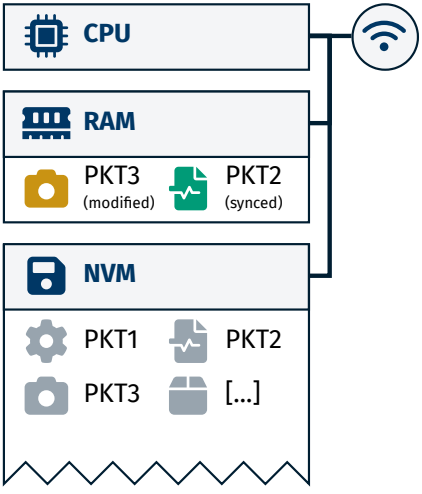
Practical Example




```
send_next_pkt();
→ [...]
send_next_pkt();
→ [...]
send_next_pkt();
→ pkt3 = dequeue_pkt();
→ ref3 = pkt3.get_mut();
```

Max modified packets: 1

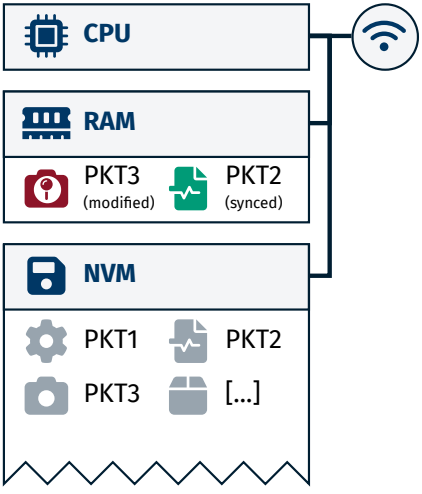
Practical Example




```
send_next_pkt();
→ [...]
send_next_pkt();
→ [...]
send_next_pkt();
→ pkt3 = dequeue_pkt();
→ ref3 = pkt3.get_mut();
```

Max modified packets: 1

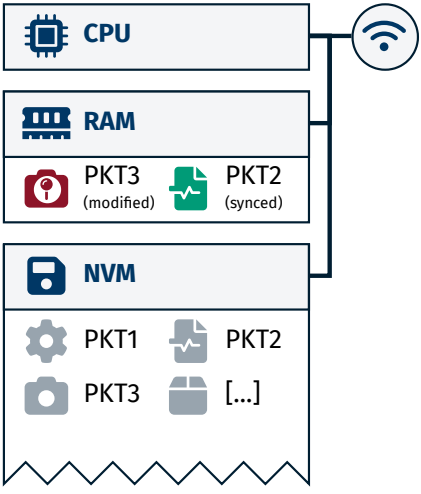
Practical Example




```
send_next_pkt();
→ [...]
send_next_pkt();
→ [...]
send_next_pkt();
→ pkt3 = dequeue_pkt();
→ ref3 = pkt3.get_mut();
```

Max modified packets: 1

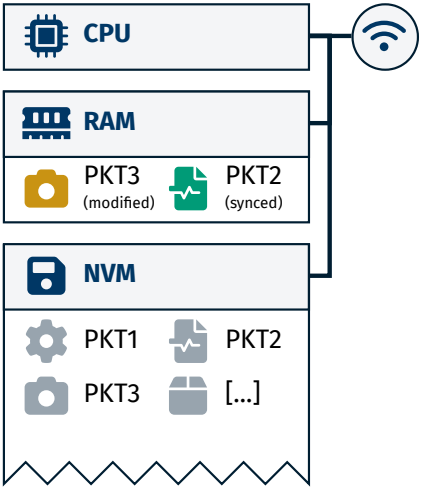
Practical Example




```
send_next_pkt();
→ [...]
send_next_pkt();
→ [...]
send_next_pkt();
→ pkt3 = dequeue_pkt();
→ ref3 = pkt3.get_mut();
→ ref3.crc = calc_crc(ref3);
```

Max modified packets: 1

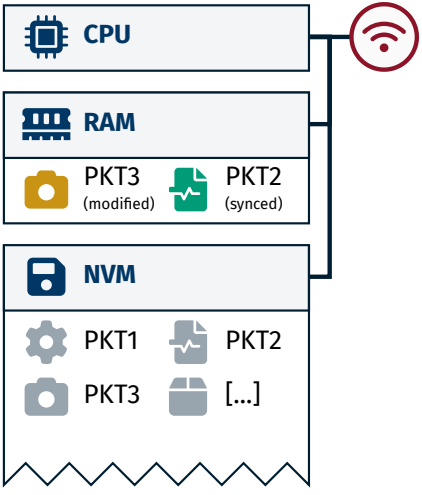
Practical Example




```
send_next_pkt();
→ [...]
send_next_pkt();
→ [...]
send_next_pkt();
→ pkt3 = dequeue_pkt();
→ ref3 = pkt3.get_mut();
→ ref3.crc = calc_crc(ref3);
→ drop(ref3);
```

Max modified packets: 1

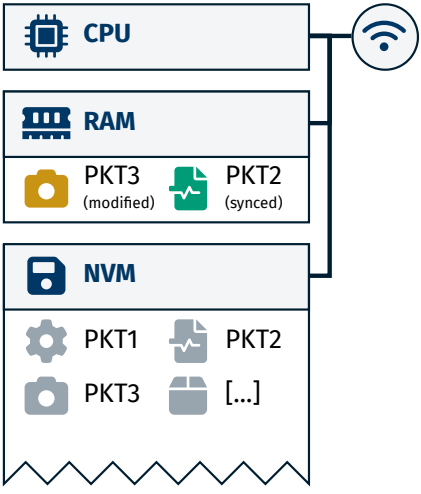
Practical Example




```
send_next_pkt();
→ [...]
send_next_pkt();
→ [...]
send_next_pkt();
→ pkt3 = dequeue_pkt();
→ ref3 = pkt3.get_mut();
→ ref3.crc = calc_crc(ref3);
→ drop(ref3);
→ send(pkt3);
```

Max modified packets: 1

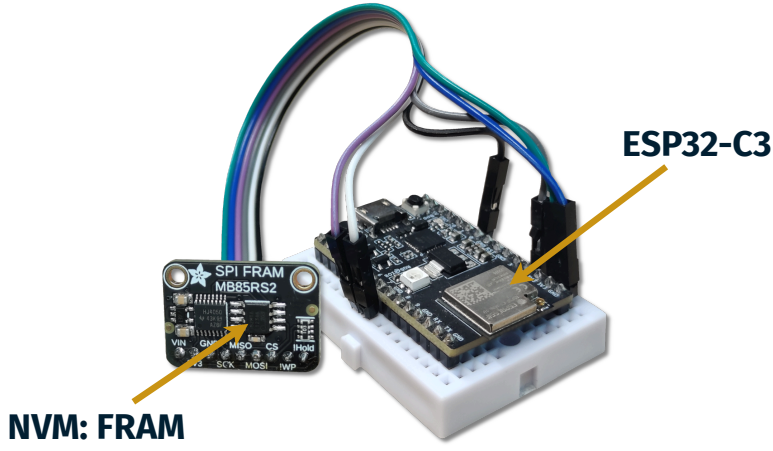
Practical Example



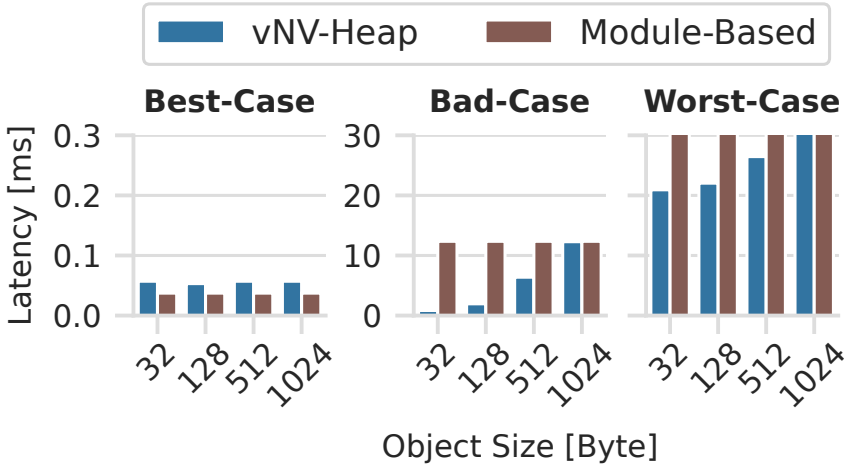

```
send_next_pkt();
→ [...]
send_next_pkt();
→ [...]
send_next_pkt();
→ pkt3 = dequeue_pkt();
→ ref3 = pkt3.get_mut();
→ ref3.crc = calc_crc(ref3);
→ drop(ref3);
→ send(pkt3);
```

Max modified packets: 1

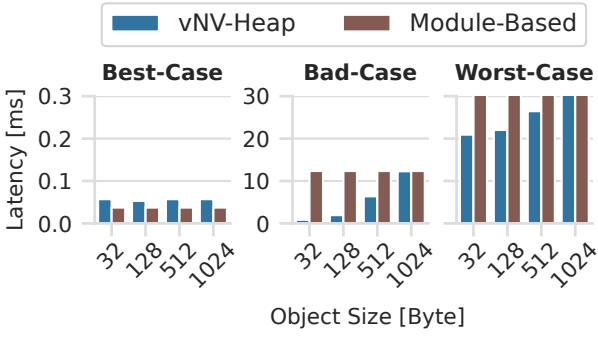
Evaluation Setup



Reference Retrieval



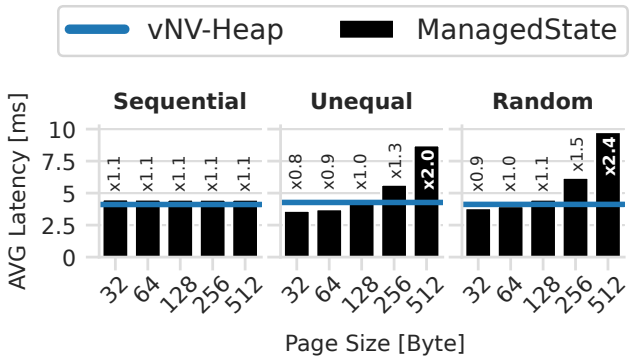
Reference Retrieval



Takeaway #2

More **fine grained management** of objects **reduces swapping overhead up to 93%** in the worst/relevant case!

Key-Value Store



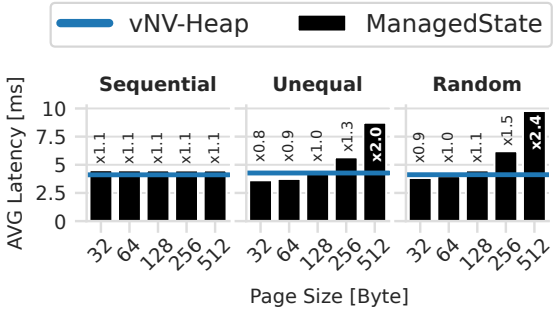
KVS Keys

32-bit integers

KVS Values (58 KiB)

- 64×32 bytes
- 128×128 bytes
- 32×256 bytes
- 32×1024 bytes

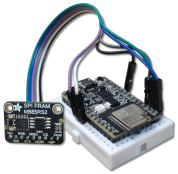
Key-Value Store



Takeaway #3

The vNV-Heap provides **similar/balanced performance**, while providing more features: **NVM safety & transparent swapping**!

Conclusion



vNV-Heap: virtually Non-Volatile Heap

- ✔ **NVM safety** ✔ **Transparent swapping**
  → Treat memory as resource & guard it with ownership/borrowing
- ✔ **Predictable**
  → Provides worst-case guarantees
  → Improves the worst case with object-wise modification tracking
- ✔ **HW agnostic**
  → Implemented & employable as a software library

Thank you for listening!

Link to Paper

Link to Code

References (1)

[1] M. Afanasov et al. **“Battery-less zero-maintenance embedded sensing at the mithræum of circus maximus”.** In: SenSys ’20. Virtual Event, Japan: Association for Computing Machinery, 2020, pp. 368–381. ISBN: 9781450375900. DOI: 10.1145/3384419.3430722.

[2] S. Sliper et al. **“Efficient State Retention through Paged Memory Management for Reactive Transient Computing”.** In: DAC ’19. ACM, June 2019. DOI: 10.1145/3316781.3317812.

References (2)

[3] J. de Winkel, H. Tang, and P. Pawełczak. **“Intermittently-powered bluetooth that works”.** In: MobiSys ’22. Portland, Oregon, 2022, pp. 287–301. DOI: 10.1145/3498361.3538934.