THE IEEE INTERNATIONAL CONFERENCE
ON CLOUD COMPUTING

EFFICIENT VERSIONING FOR UNIKERNELS

GAULTHIER GAIN, BENOIT KNOTT & PROF. LAURENT MATHY
University of Liège

IEEE World Congress on SERVICES
July 7-12, 2025, Helsinki, Finland

Dilemma

To deploy microservices, developers commonly rely on **Virtual Machines** (VMs) or **Containers**

Virtual Machines (VMs)	Containers
✓ Strong isolation	✗ Less isolation – More exploits
✗ Heavyweight – Degrade performance	✓ Lightweight – Share underlying kernel

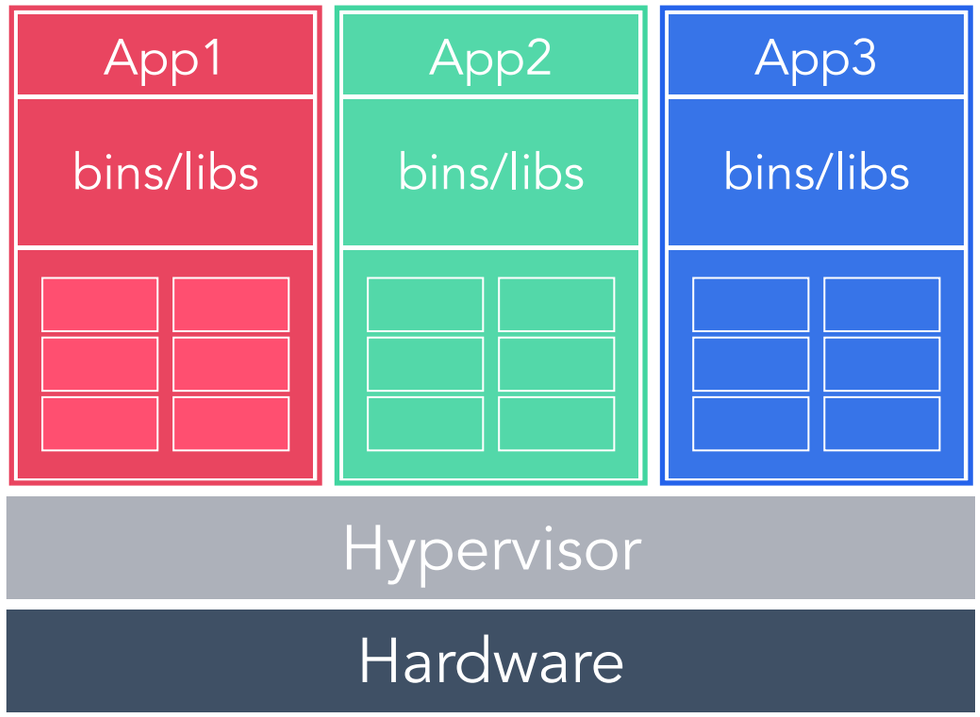
Dilemma

To deploy microservices, developers commonly rely on **Virtual Machines** (VMs) or **Containers**

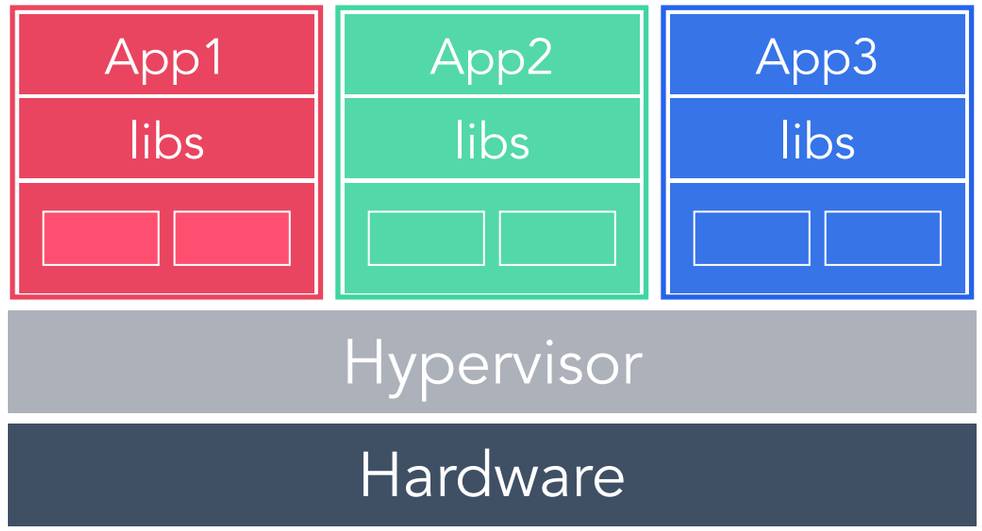
Virtual Machines (VMs)	**Containers**
✓ Strong isolation	✗ Less isolation – More exploits
✗ Heavyweight – Degrade performance	✓ Lightweight – Share underlying kernel

Solution → **Unikernels**

UNIKERNELS



Virtual Machines (VMs)



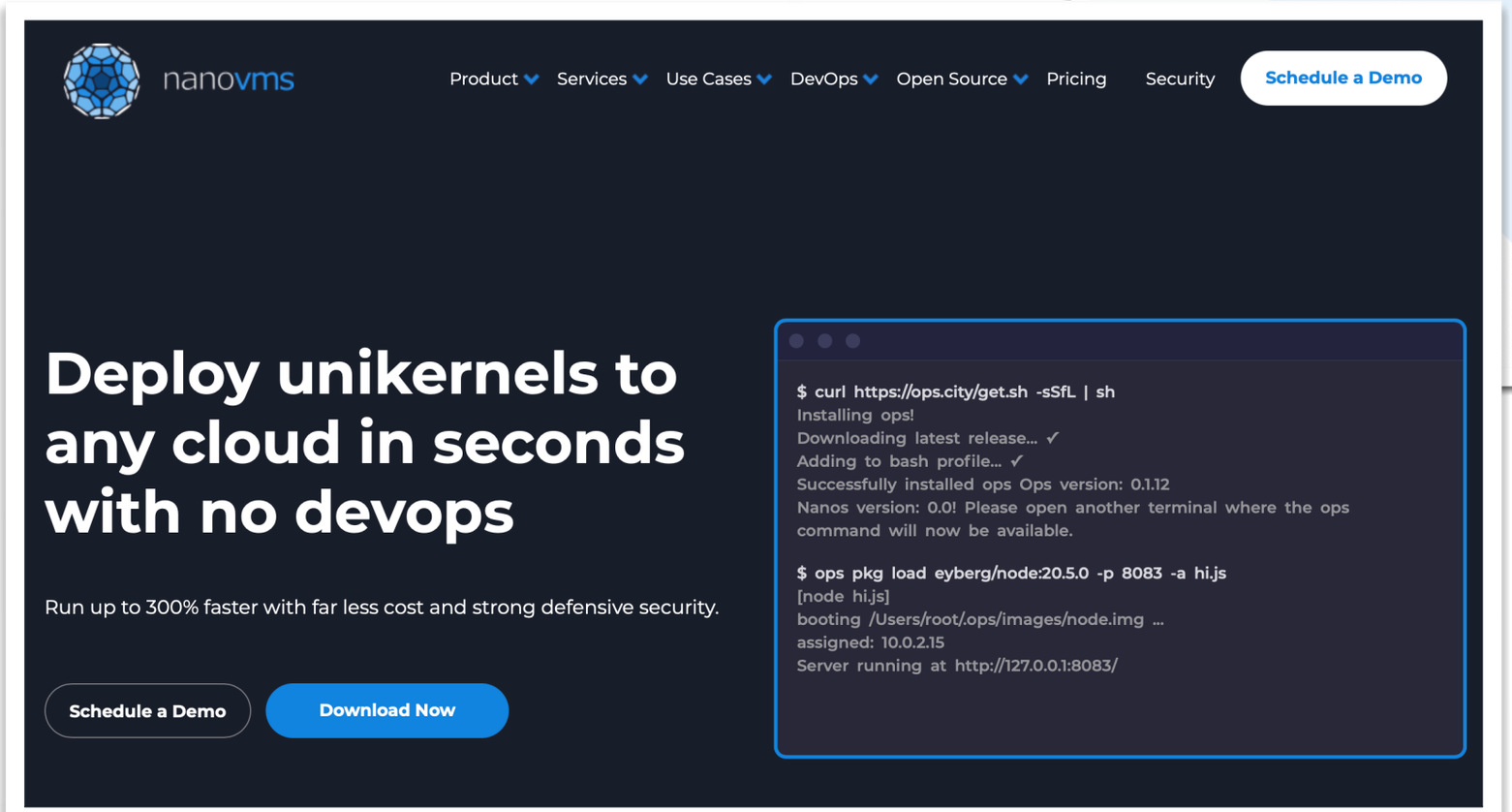
Unikernels

Unikernels are purpose-built:

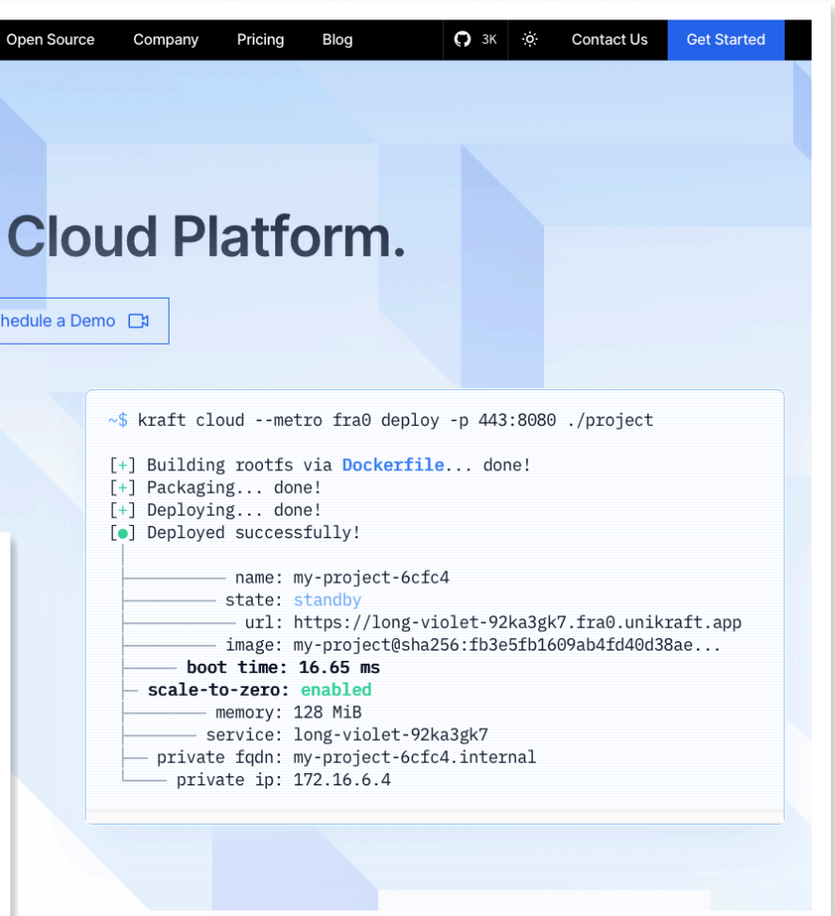
- Thin kernel layer (only the necessary features that the application needs)
- Essential functions are placed into libraries with well-defined behaviour

UNIKERNELS: CLOUD PLATFORMS

Unikernels are gradually being adopted in cloud computing



Nanovms



Unikraft Cloud

The Unikernel Trade-Off

Major unikernel projects use a **static linking** model:

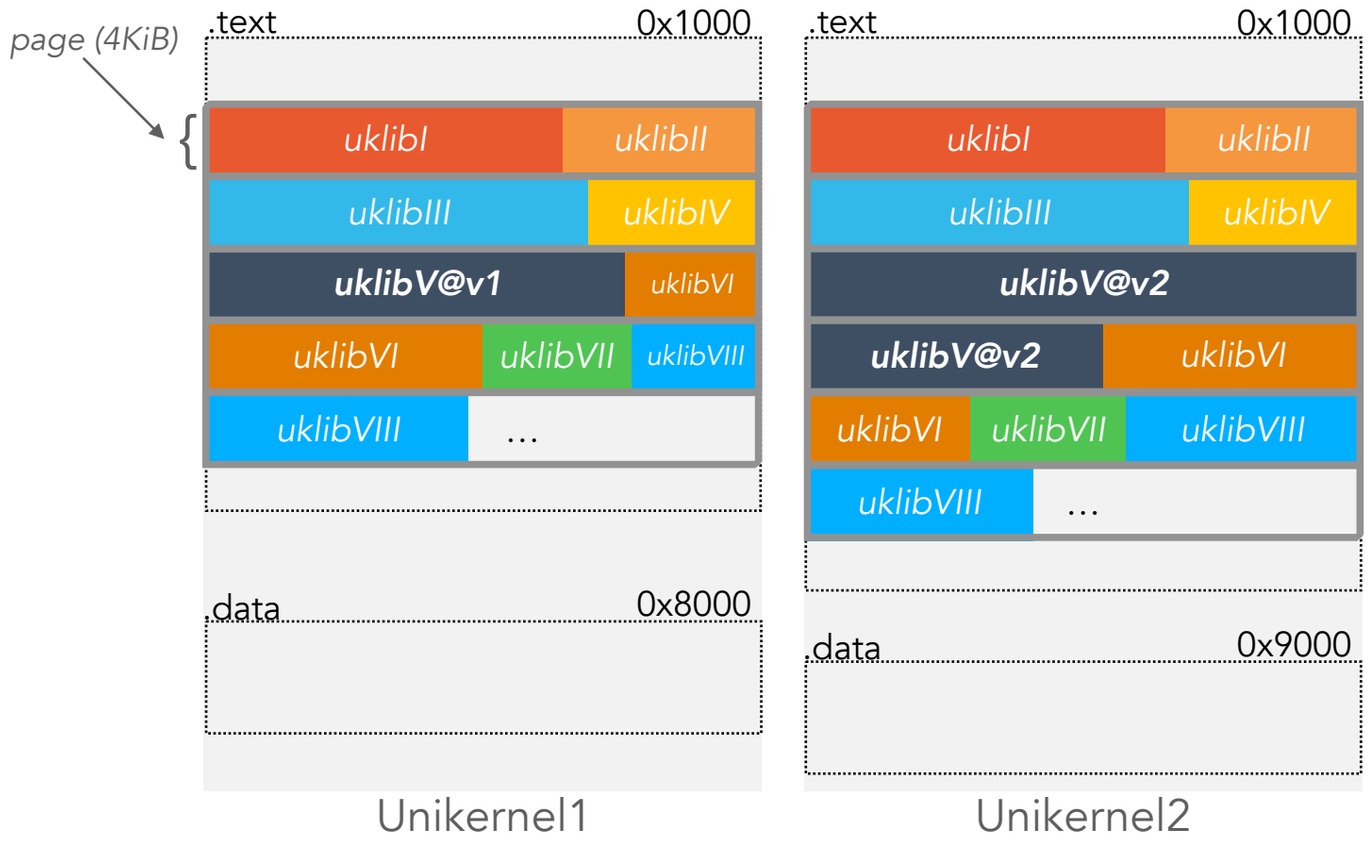
- All application code and required libraries are compiled into a single binary at build time
- No need for a dynamic linker

Results in:

- Faster boot times
- Improved security through reduced attack surface + isolation
- No “dependency hell” issue

But... handling library versioning when deploying many unikernels in cloud platforms introduces challenges

MEMORY DEDUPLICATION WITH VERSIONING



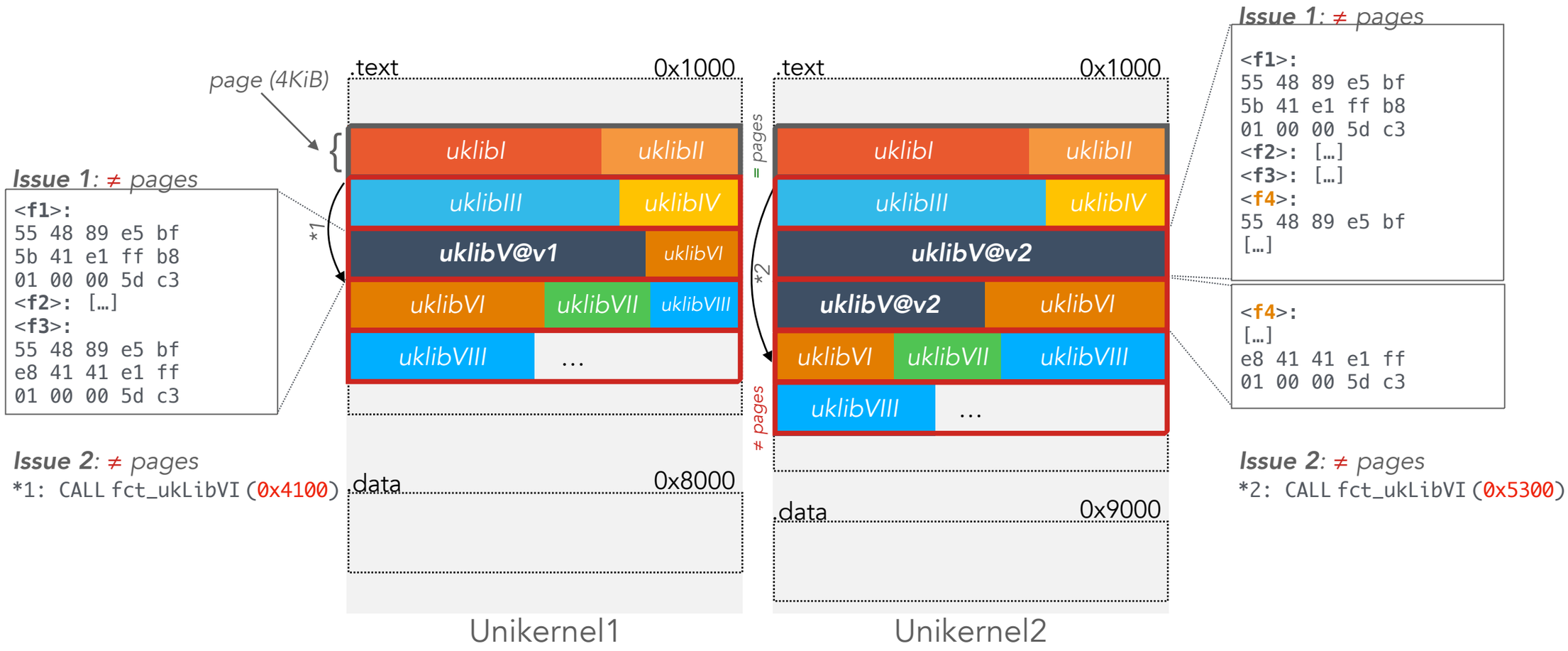
Let's consider 2 unikernels using ≠ versions of the same library (e.g., *uklibV@v1* and *uklibV@v2*)

- In memory, code is divided into pages (the *basic unit of memory management*)

Thanks to **memory deduplication**:

- Identical pages are shared and merged into the same frame
- Reduce the total #frames

MEMORY DEDUPLICATION WITH VERSIONING

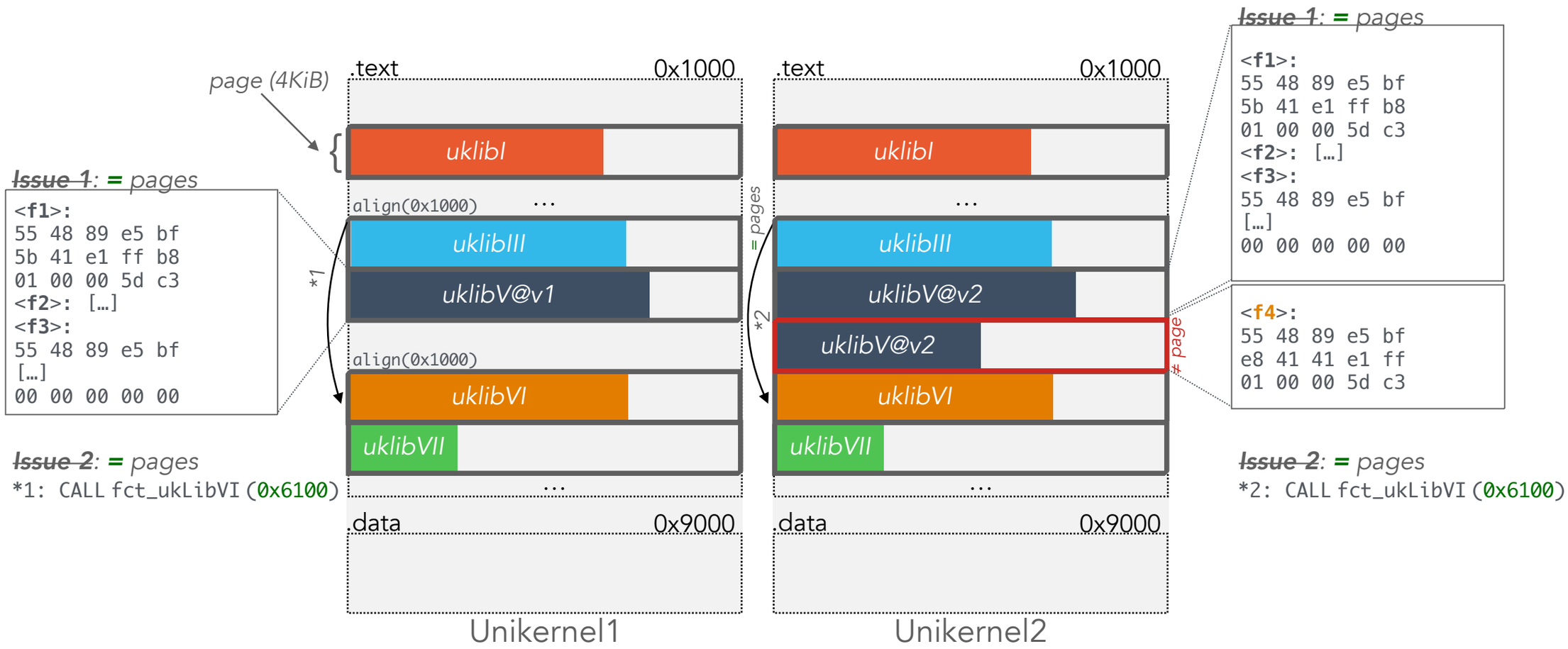


⚠️ **Problem:** Having different versions (*uklibV@v2)* impacts the memory sharing

1. This causes **page misalignment** (≠ pages)
2. This causes **cross-reference addresses** to be different (≠ pages)

Result: Most of the pages are different and cannot be shared

USING ALIGNMENT AS A SOLUTION?



Alignment could be a potential solution:

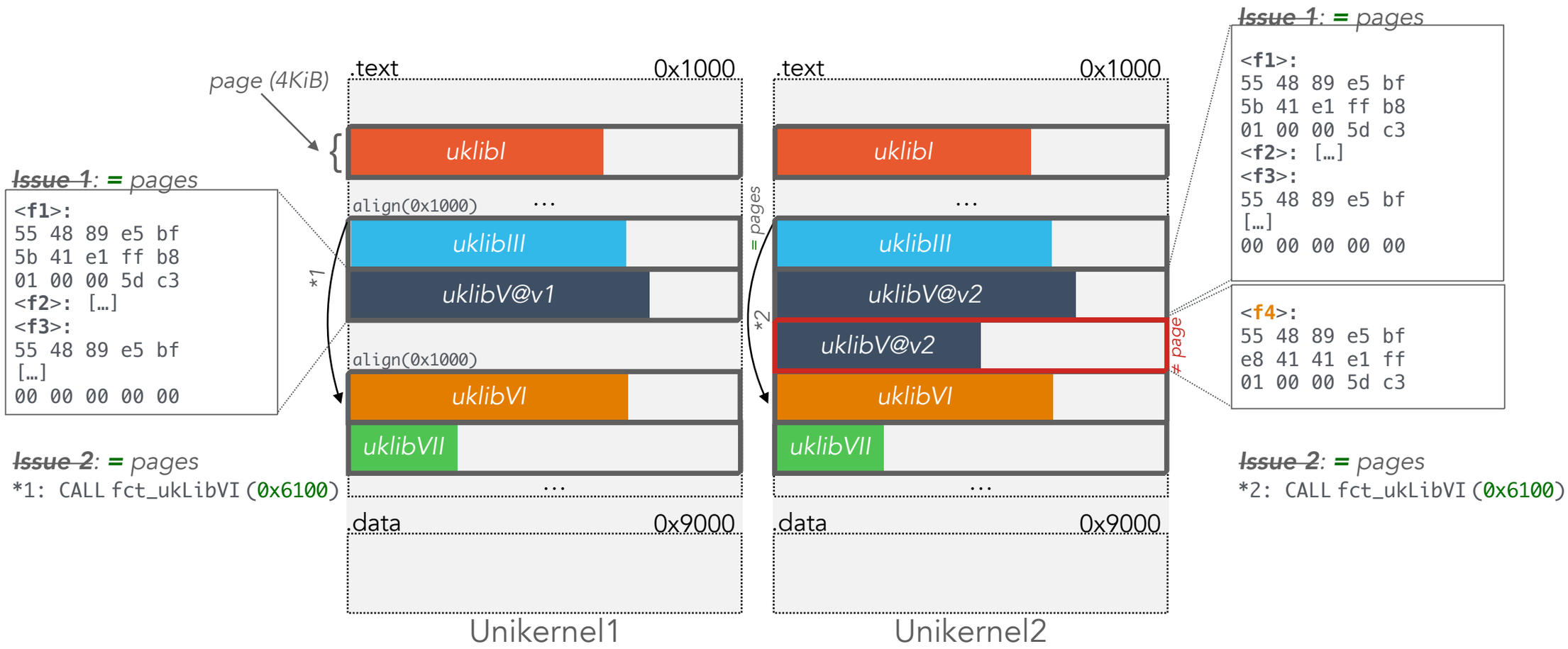
- Libraries are placed at fixed, page-aligned absolute addresses

Seems a good solution:

1. Fixes the **page misalignment** issue?
2. Fixes the **cross-reference addresses** issue?

Is it enough?

USING ALIGNMENT AS A SOLUTION?



Alignment could be a potential solution:

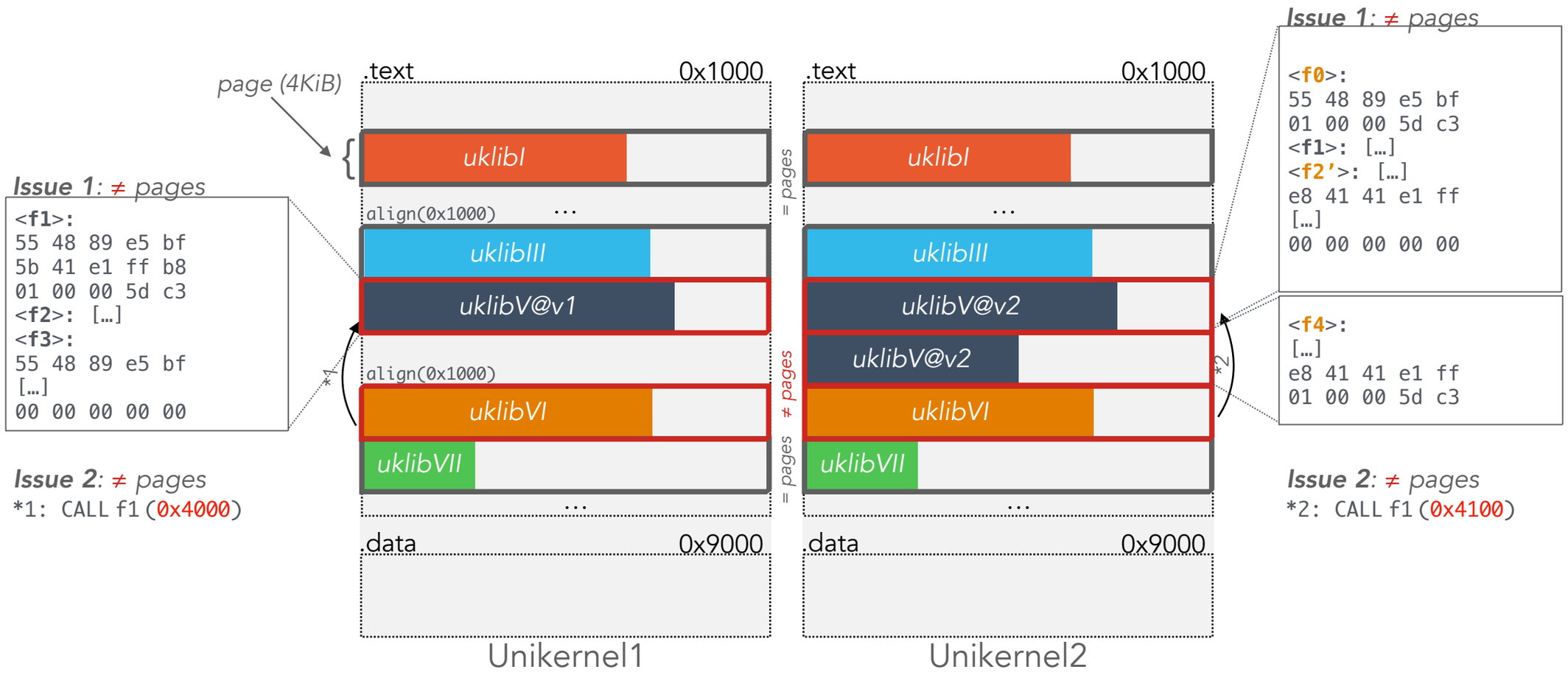
- Libraries are placed at fixed, page-aligned absolute addresses

Seems a good solution:

1. Fixes the **page misalignment** issue?
2. Fixes the **cross-reference addresses** issue?

✗ Need to consider most **complex** cases

USING ALIGNMENT AS A SOLUTION?



⚠ **Problem:** If code has bigger **changes** like function removal, reordering, or modification, alignment is not enough:

1. Still page misalignment due to functions placement (≠ pages)
2. Still different cross-references addresses (≠ pages)

Result: Pages are different and cannot be shared ⟶ Need a **better** function placement

TRYING TO IMPROVE MEMORY SHARING

We first focus on **function misalignment** *(issue 1)*:

lib@v1	*lib@v2*	*lib@v3*
f1 (.5)	f1 (.5)	f1 (.5)
f2 (.3)	/	f2 (.3)
f3 (.1)	f3' (.3)	f3' (.3)
f4 (.6)	f4 (.6)	f4' (.5)
f5 (.4)	f5 (.4)	f5 (.4)
/	f6 (.3)	f6 (.3)
/	f7 (.2)	f7' (.1)

Table 1: Functions per versioned libraries

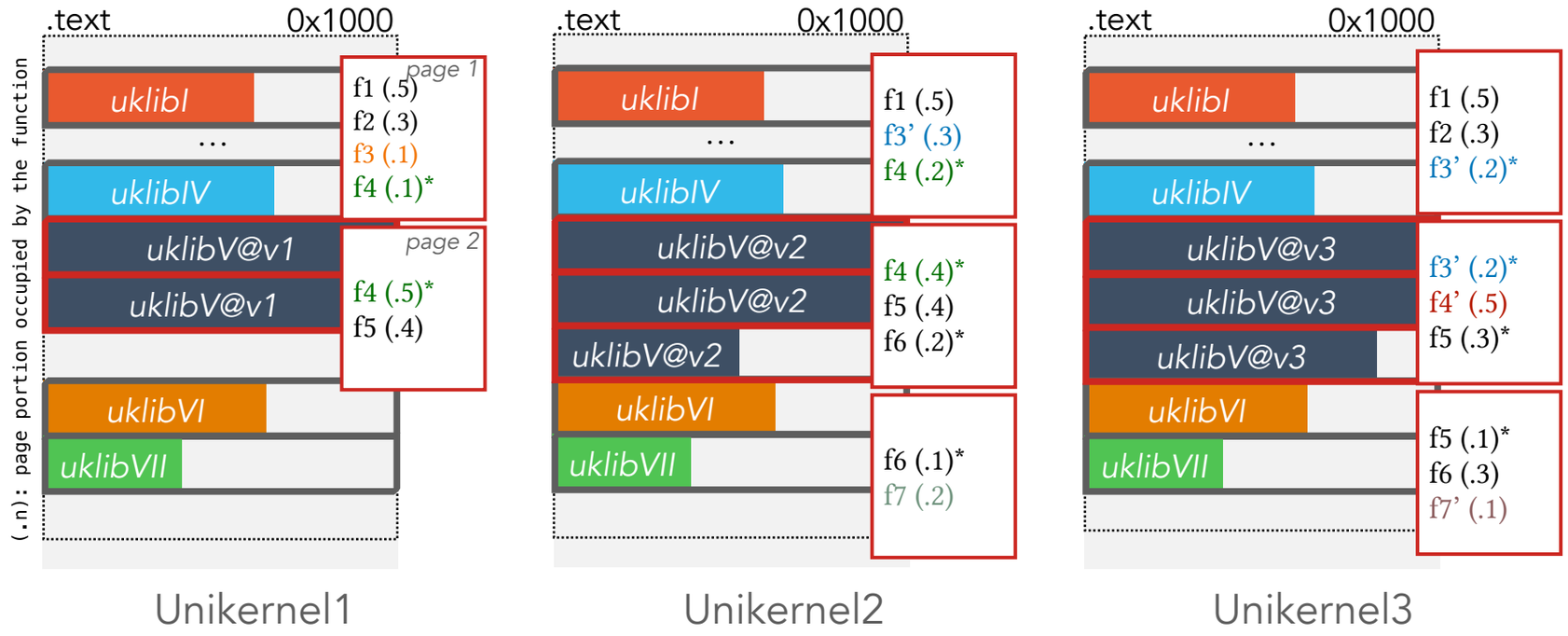


Table 1 shows the functions used by versioned libraries of 3 unikernels

- Each function is assigned a fraction of a page

Without consistent function placement:

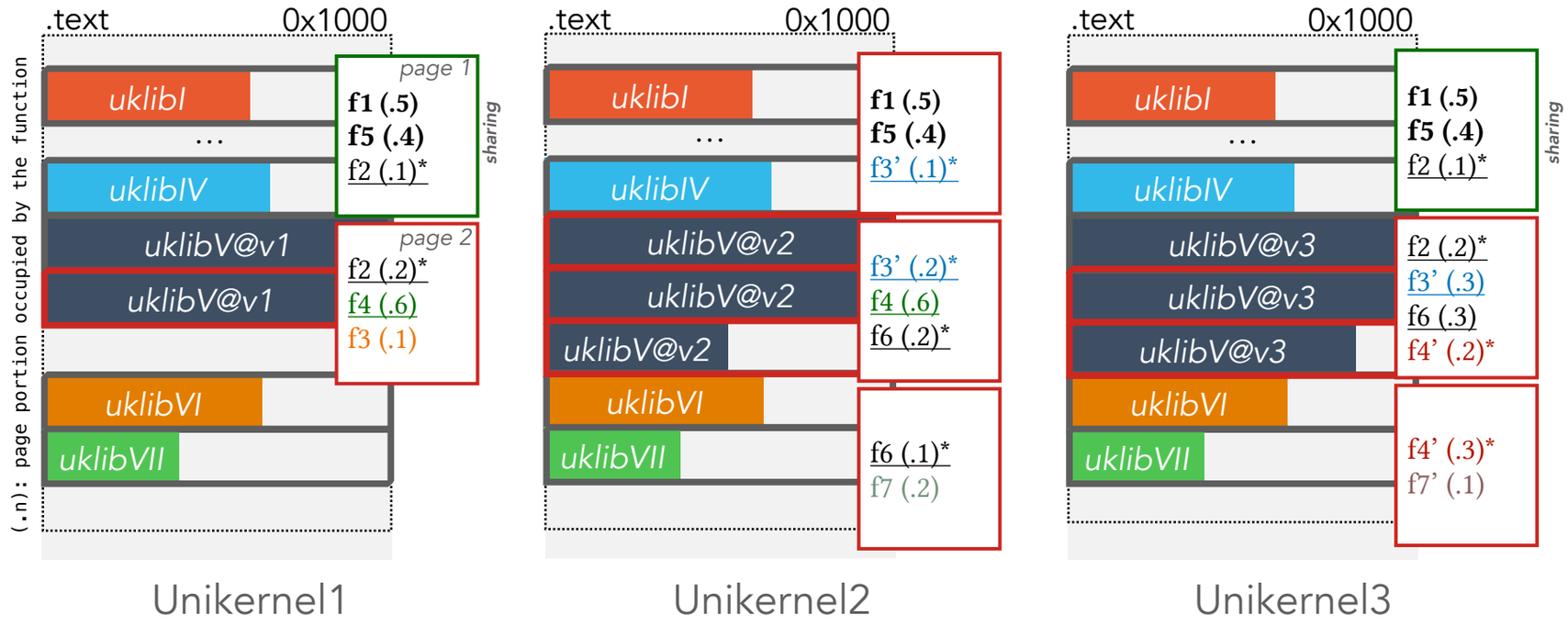
- All pages are unique, which leads to **no deduplication** between instances

Total: 8 pages $\xrightarrow{\text{0 pages shared}}$ 8 frames

TRYING TO IMPROVE MEMORY SHARING

lib@v1	*lib@v2*	*lib@v3*
f1 (.5)	**f1 (.5)**	**f1 (.5)**
f2 (.3)	/	f2 (.3)
f3 (.1)	f3’ (.3)	f3’ (.3)
f4 (.6)	f4 (.6)	f4’ (.5)
f5 (.4)	**f5 (.4)**	**f5 (.4)**
/	f6 (.3)	f6 (.3)
/	f7 (.2)	f7’ (.1)

Table 1: Functions per versioned libraries



Track functions across library versions and organize them in a consistent order (e.g., *per occurrence*)

- Sharing is a bit improved but still not great…
- **Bad idea:** Too complex to **optimally place functions** when multiple instances/versions

Total: 8 pages $\xrightarrow{\text{2 pages shared}}$ 7 frames

Trying to improve memory sharing

We also explore:

1. Optimization methods, such as bin-packing, strip-packing, etc.

   ⚠️ **Problem:** Requires significant computation, making it impractical at scale

2. Aligning each function to a separate page

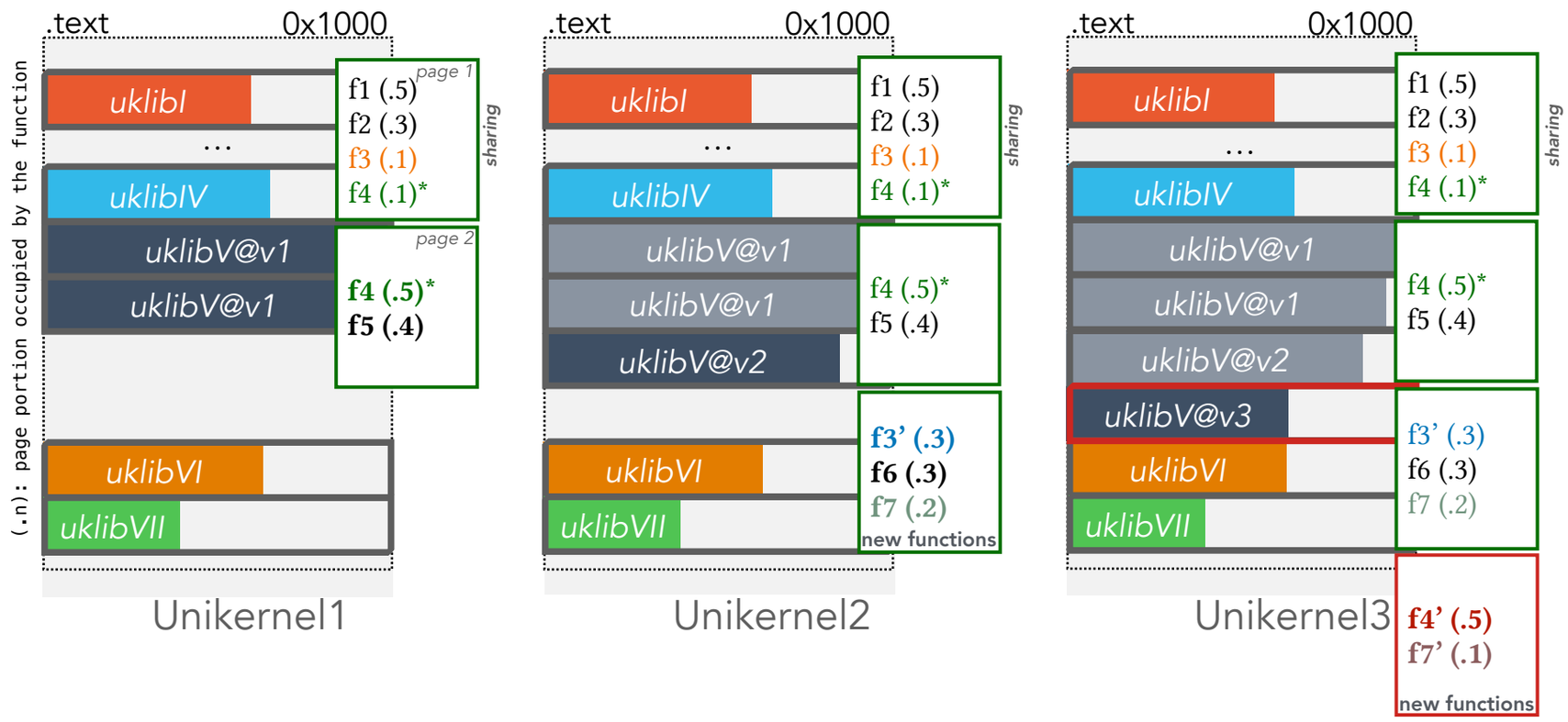
   ⚠️ **Problem:** significant internal fragmentation if functions are small

Need a **better** approach

Improving memory sharing with Spacer-Δ

lib@v1	*lib@v2*	*lib@v3*
f1 (.5)	f1 (.5)	f1 (.5)
f2 (.3)	/	f2 (.3)
f3 (.1)	**f3' (.3)**	f3' (.3)
f4 (.6)	f4 (.6)	**f4' (.5)**
f5 (.4)	f5 (.4)	f5 (.4)
/	**f6 (.3)**	f6 (.3)
/	f7 (.2)	**f7' (.1)**

Table 1: Functions per versioned libraries



Spacer-Δ: Backward-Compatible Alignment across versions

- Each new library version is expressed as a **delta** from previous ones
- Reuses **existing** functions to preserve alignment and improve sharing
- **Modified** and **new** functions are added at the start of a new page

Is it enough?

Total: 9 pages $\xrightarrow{\textbf{8} \text{ pages shared}}$ 4 frames ✓

LIÈGE université

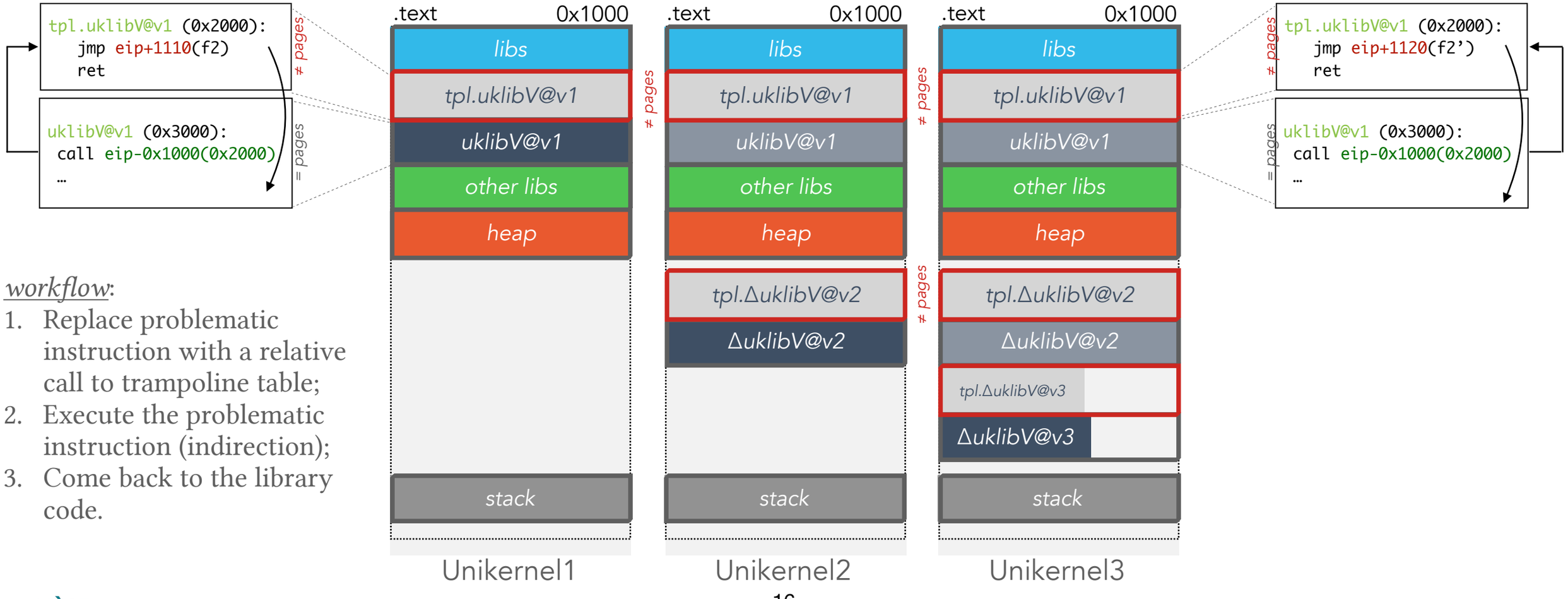
Managing Versions: Additional Steps

Still needs to handle the cross-references issue:

- Use **Trampoline tables (.tpl)**:
  - To isolate problematic instructions (e.g., `f1` calls `f2` in *v1* then `f1` calls `f2'` in *v3*)
  - Use **binary rewriting** to replace problematic instructions with a combination of `call`, `ret` and `jmp` instructions (see *workflow*)
  - Ensures the library's page remains identical across instances

To also avoid overlap with adjacent memory pages

- Each new version (or delta) is placed between **the heap** and **the stack**



workflow:

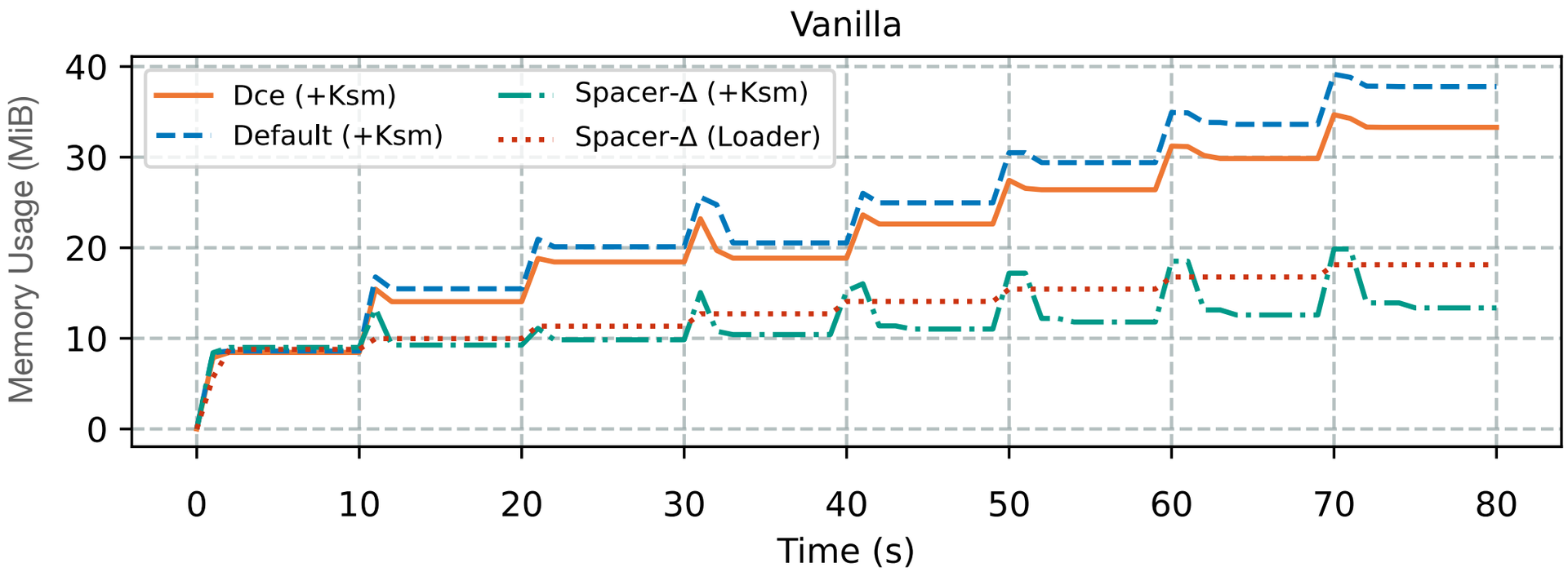
1. Replace problematic instruction with a relative call to trampoline table;
2. Execute the problematic instruction (indirection);
3. Come back to the library code.

EVALUATION: METHODOLOGY

- We compared Spacer-Δ with DCE (Dead Code Elimination) and the Default Unikraft configuration
  - 5 applications ported as unikernels
    - Library versions selected based on specific GitHub tags
  - We rely on 2 different memory deduplication mechanisms:
    1. *KSM*: **Runtime** scanner that merges identical pages
    2. *Loader*: **Load-time** deduplication integrated into the hypervisor, which uses a library pool
- On several dimensions: memory consumption, filesize and performance

Please refer to the paper for the full evaluation.

Evaluation (1)

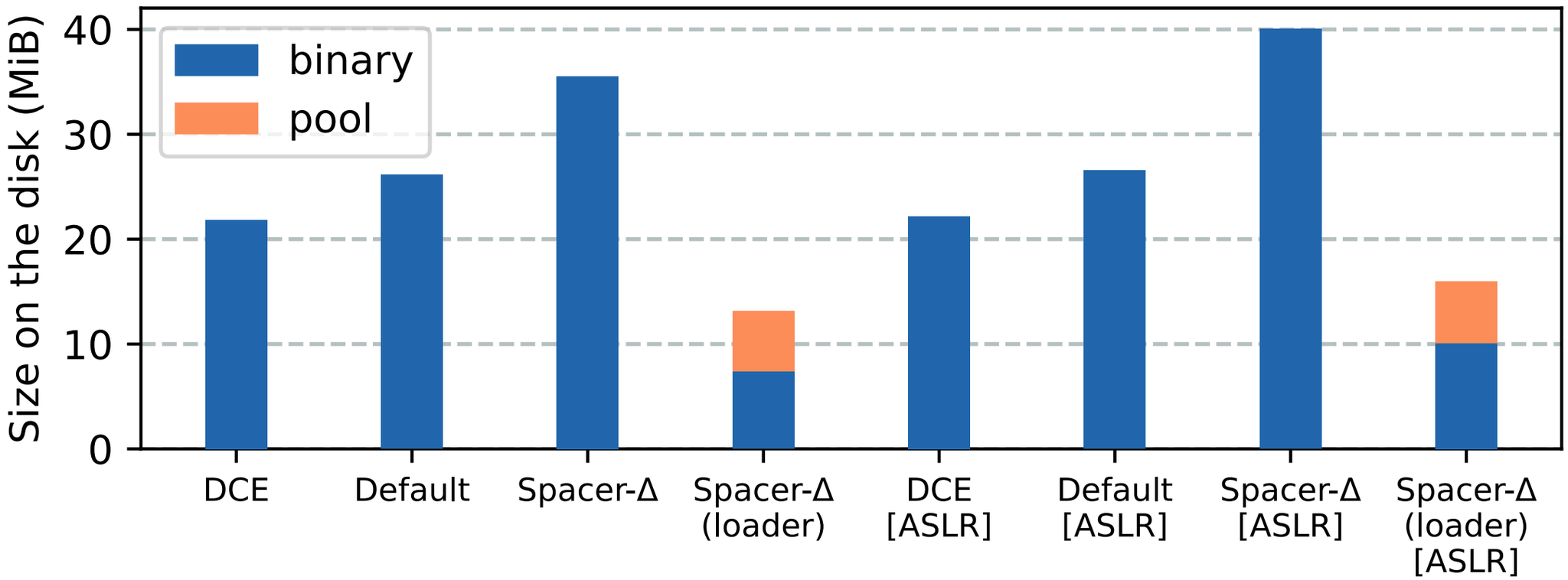


Memory consumption:

- 8 Nginx unikernel instances, each using a different version of lib-sqlite
- Spacer-Δ (+**KSM**) reduces memory usage by up to **2.8×**
- Spacer-Δ (**loader**) slightly higher memory usage ⟶ merges only read-only pages to prevent CoW and side channel attacks[1]

[1] Software Side Channel Attack on Memory Deduplication, *Suzaki et al.*

Evaluation (2)



Filesize consumption:

- 8 Nginx unikernel instances, each using a different version of lib-sqlite
- Spacer-Δ (+**KSM**) increases file size by up to **1.6×**
- Spacer-Δ (**loader**) achieves optimal reduction via **shared libraries pool**

Evaluation (3)

Setup:

- Several experiments: different cores, same core, different workloads
- Total execution time of short-lived and long-lived unikernels

Performance:

- Spacer-Δ **(+KSM):** Slight overhead due to inflation and trampoline tables
  - KSM also has a slight impact on scanning and merging pages
- Spacer-Δ **(loader):** Best performance (~5–12%) uses a preloaded library pool (code + read-only data) and avoids KSM overhead

Conclusion

- Unikernels are lightweight and high-performance, ideal for the cloud
- Versioning & updates are **challenging** due to static linking
- Spacer-Δ enables efficient library versioning via **alignment** & **backward** compatibility
- Yields notable gains in memory, disk usage and execution time (especially when coupled with our loader)

Thank you!

Questions?

SPACER-Δ & UNIKRAFT

Spacer-Δ: https://github.com/gaulthiergain/Spacer-delta

Unikraft: https://unikraft.org/

Contact by email: gaulthier.gain@uliege.be

LIÈGE université

Unikraft