13TH SYMPOSIUM ON CLOUD COMPUTING (SOCC'22)

WANT MORE UNIKERNELS? INFLATE THEM!

GAULTHIER GAIN[1], CYRIL SOLDANI[1], FELIPE HUICI[2], PROF. LAURENT MATHY[1]

[1]University of Liège, [2]Unikraft GmbH

ACM Symposium on Cloud Computing (SoCC '22),
November 8–10, 2022, San Francisco, CA, USA.

Dilemma

Virtual Machines (VMs)

✓ Strong isolation

✗ Heavyweight
– Degrade performance

Containers

✗ Poor isolation
– A lot of exploits

✓ Lightweight
– Share underlying kernel

DILEMMA

Virtual Machines (VMs)

✓ Strong isolation

✗ Heavyweight
– Degrade performance

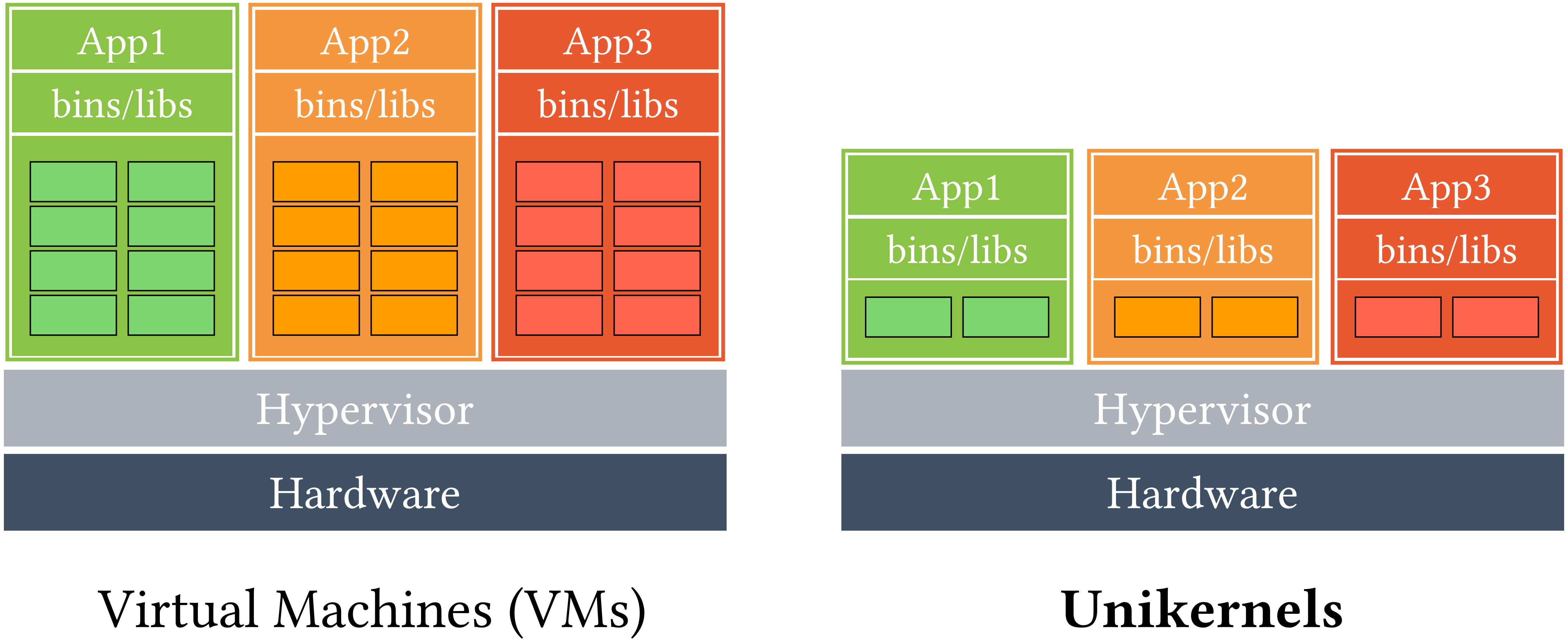
Containers

✗ Poor isolation
– A lot of exploits

✓ Lightweight
– Share underlying kernel

Solution → **Unikernels**

Unikernels



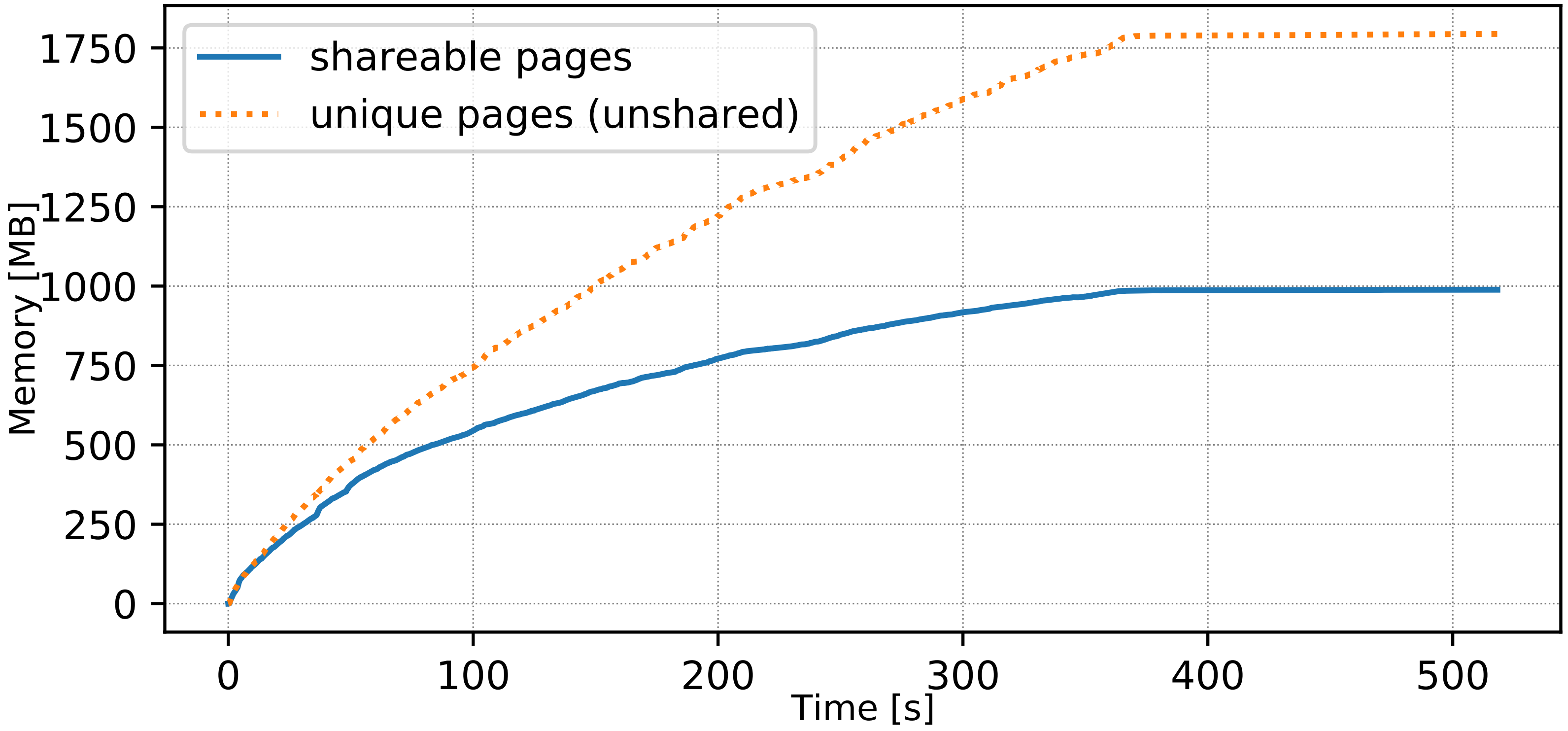
Unikernels are purpose-built:

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

Unikernels Gains

- Fast instantiation, destruction and migration times:
  - Hundred of milliseconds.
- Small per-instance memory footprint:
  - Few MBs or even KBs.
- High performance:
  - 10-40 Gbps throughput.
- Reduced attack surface:
  - Less components.
- High density:
  - Thousand of instances on a single host ⟶ Can we do better ?

Running a large number of unikernels

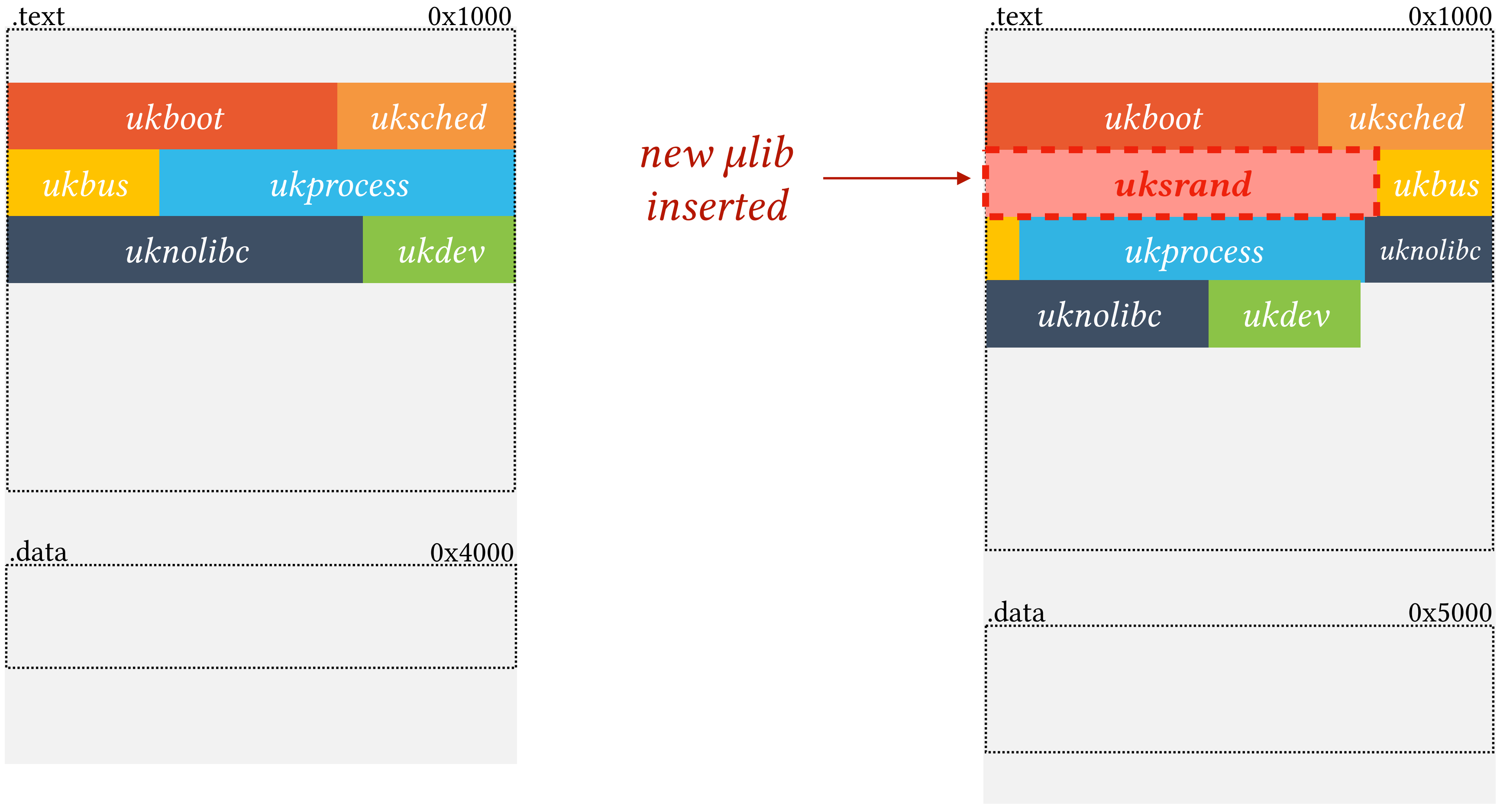


Evolution of unshared and shareable (i.e. having at least one copy) pages when running 1000 different FaaS unikernels (with ASLR) on a single physical server.

- We investigated by running a large number of unikernels on a same physical server and we relied on a memory deduplication scanner (UKSM*).
- Unique pages are much more frequent than shared pages.
- Specialisation? Need further investigation to understand the reason.

*Ultra Kernel Samepage Merging

Memory deduplication with unikernels: Overview

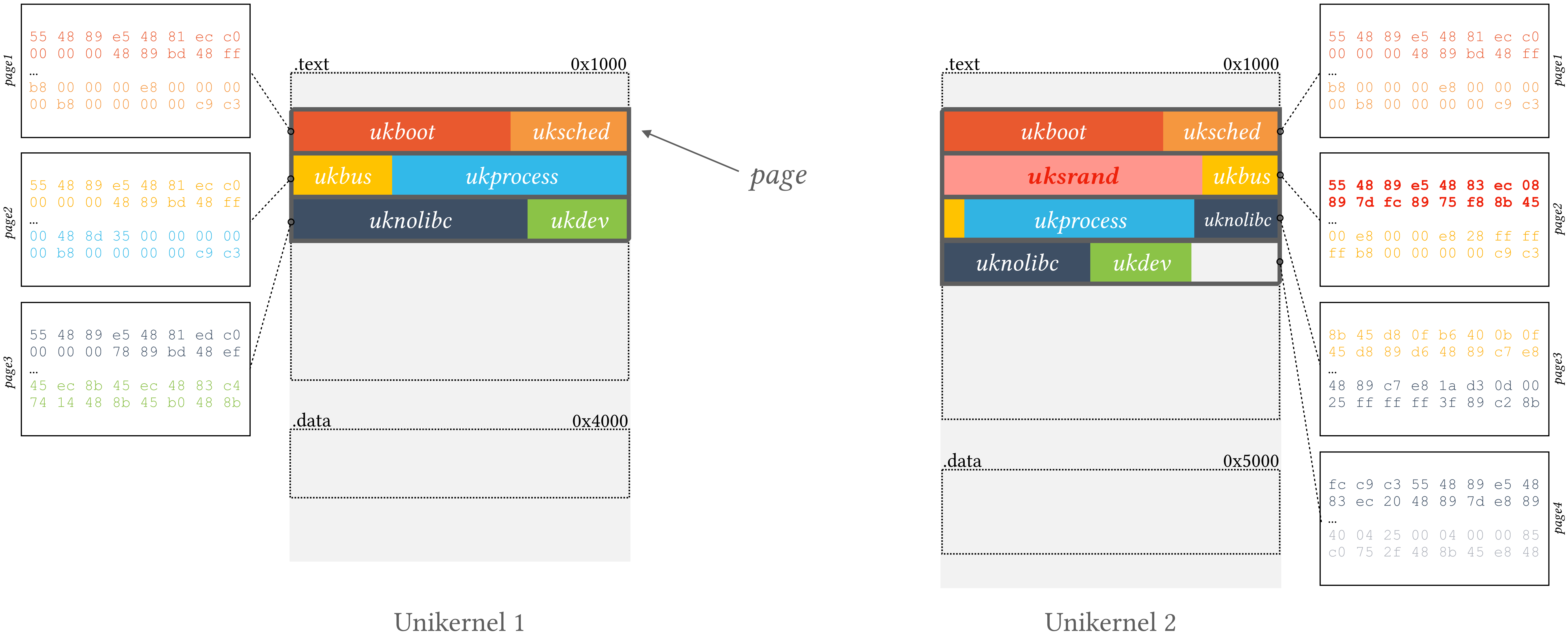


Unikernel 1

Unikernel 2

- Having several instances will result into different µlibs configurations.
- The underlying build system does not have a global overview of the µlibs: Each unikernel is built in an individual way.
- All µlibs are compacted: resulting unikernel consumes as little memory and disk space as possible.

MEMORY DEDUPLICATION WITH UNIKERNELS: ISSUE 1



Unikernel 1, page1:

```
55 48 89 e5 48 81 ec c0
00 00 00 48 89 bd 48 ff
…
b8 00 00 00 e8 00 00 00
00 b8 00 00 00 00 c9 c3
```

Unikernel 1, page2:

```
55 48 89 e5 48 81 ec c0
00 00 00 48 89 bd 48 ff
…
00 48 8d 35 00 00 00 00
00 b8 00 00 00 00 c9 c3
```

Unikernel 1, page3:

```
55 48 89 e5 48 81 ed c0
00 00 00 78 89 bd 48 ef
…
45 ec 8b 45 ec 48 83 c4
74 14 48 8b 45 b0 48 8b
```

Unikernel 2, page1:

```
55 48 89 e5 48 81 ec c0
00 00 00 48 89 bd 48 ff
…
b8 00 00 00 e8 00 00 00
00 b8 00 00 00 00 c9 c3
```

Unikernel 2, page2:

```
55 48 89 e5 48 83 ec 08
89 7d fc 89 75 f8 8b 45
…
00 e8 00 00 e8 28 ff ff
ff b8 00 00 00 00 c9 c3
```

Unikernel 2, page3:

```
8b 45 d8 0f b6 40 0b 0f
45 d8 89 d6 48 89 c7 e8
…
48 89 c7 e8 1a d3 0d 00
25 ff ff ff 3f 89 c2 8b
```

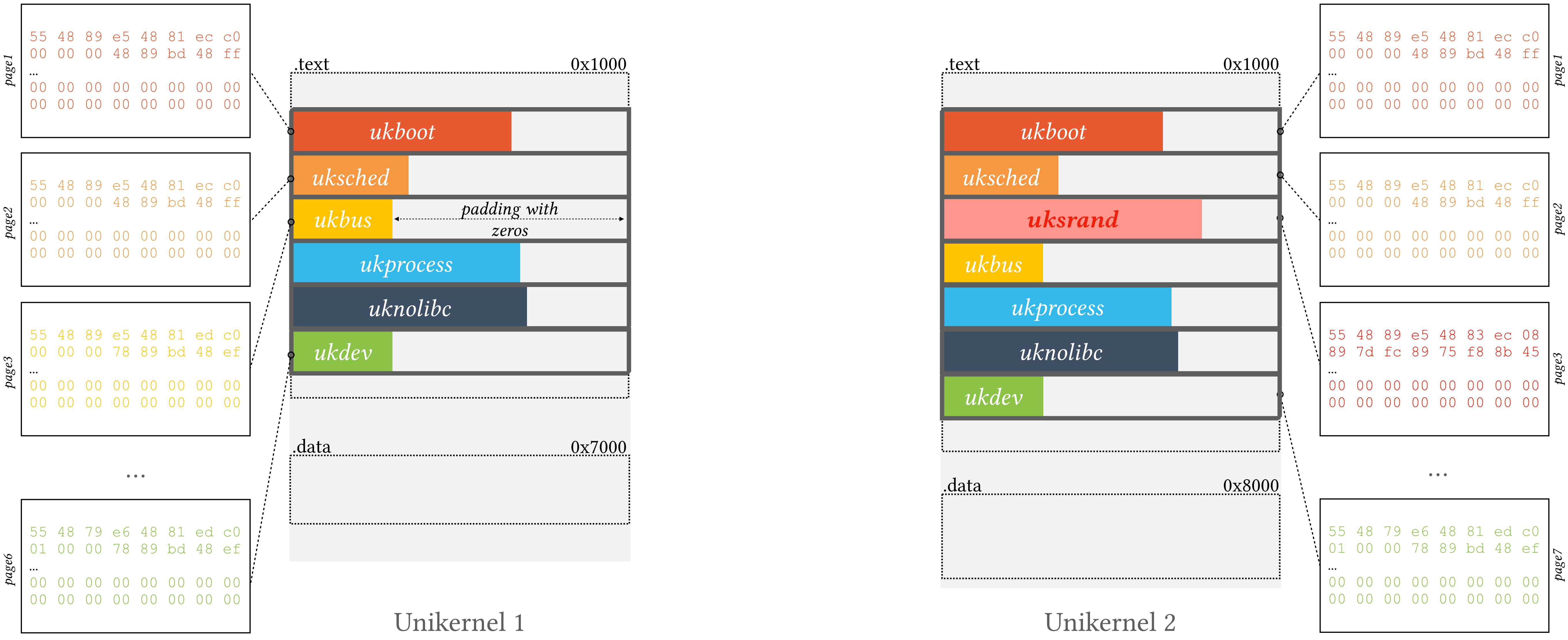
Unikernel 2, page4:

```
fc c9 c3 55 48 89 e5 48
83 ec 20 48 89 7d e8 89
…
40 04 25 00 04 00 00 85
c0 75 2f 48 8b 45 e8 48
```

If a new µlib ‘*uksrand*’ is inserted between other µlibs:

- µlibs’ code will be split across different pages.
- It reduces memory sharing since pages are different.

Memory deduplication with unikernels: A first solution?



```
55 48 89 e5 48 81 ec c0
00 00 00 48 89 bd 48 ff
...
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
```

```
55 48 89 e5 48 81 ec c0
00 00 00 48 89 bd 48 ff
...
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
```

```
55 48 89 e5 48 81 ed c0
00 00 00 78 89 bd 48 ef
...
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
```

...

```
55 48 79 e6 48 81 ed c0
01 00 00 78 89 bd 48 ef
...
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
```

```
55 48 89 e5 48 81 ec c0
00 00 00 48 89 bd 48 ff
...
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
```

```
55 48 89 e5 48 81 ec c0
00 00 00 48 89 bd 48 ff
...
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
```

```
55 48 89 e5 48 83 ec 08
89 7d fc 89 75 f8 8b 45
...
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
```

...

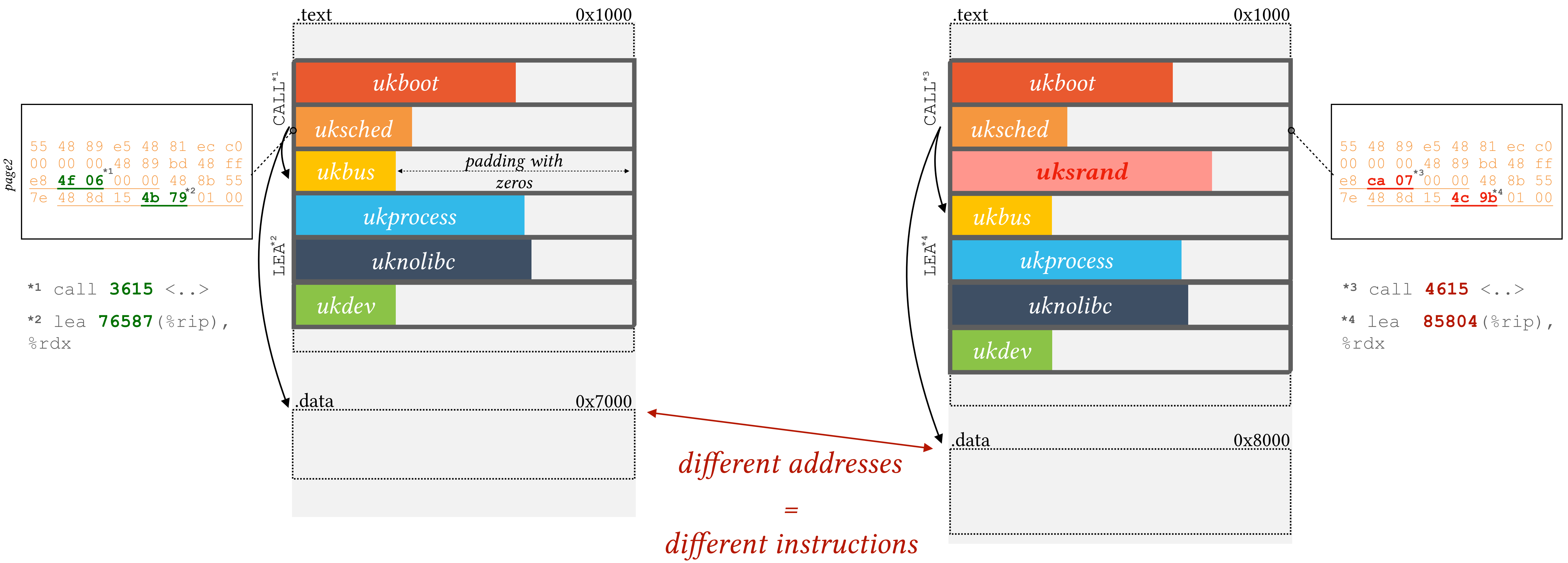
```
55 48 79 e6 48 81 ed c0
01 00 00 78 89 bd 48 ef
...
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
```

To circumvent this issue:

- Align each µlib to a page boundary address.
- Pad the µlib code with zeros to fill a complete page.

} → Is it enough?

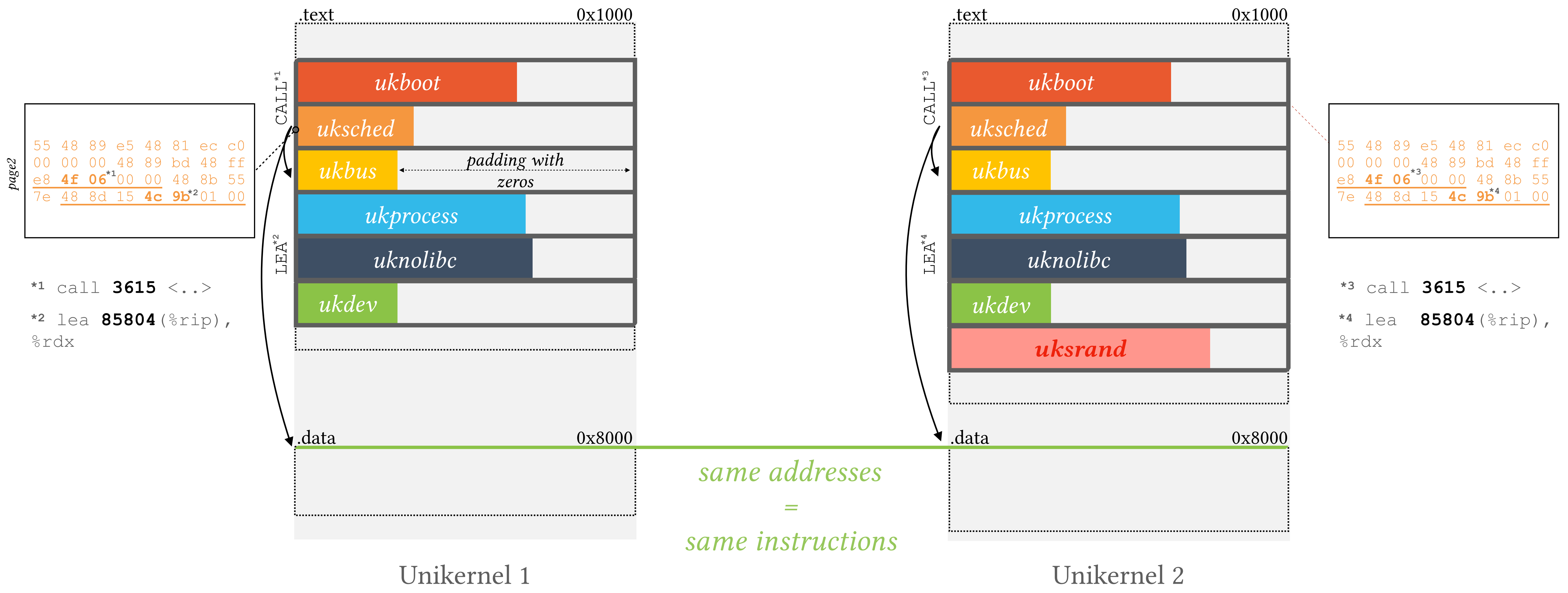
Memory deduplication with unikernels: Issue 2



Some instructions use different addresses in the *.text* section:

- Related to other sections (e.g., *.data*, *.rodata*): `MOV` and `LEA`.
- Related to another part of the *.text* section: `CALL`.

MEMORY DEDUPLICATION WITH UNIKERNELS: A WORKING SOLUTION



```
55 48 89 e5 48 81 ec c0
00 00 00 48 89 bd 48 ff
e8 4f 06*1 00 00 48 8b 55
7e 48 8d 15 4c 9b*2 01 00
```

*1 call **3615** <..>

*2 lea **85804**(%rip), %rdx

```
55 48 89 e5 48 81 ec c0
00 00 00 48 89 bd 48 ff
e8 4f 06*3 00 00 48 8b 55
7e 48 8d 15 4c 9b*4 01 00
```

*3 call **3615** <..>

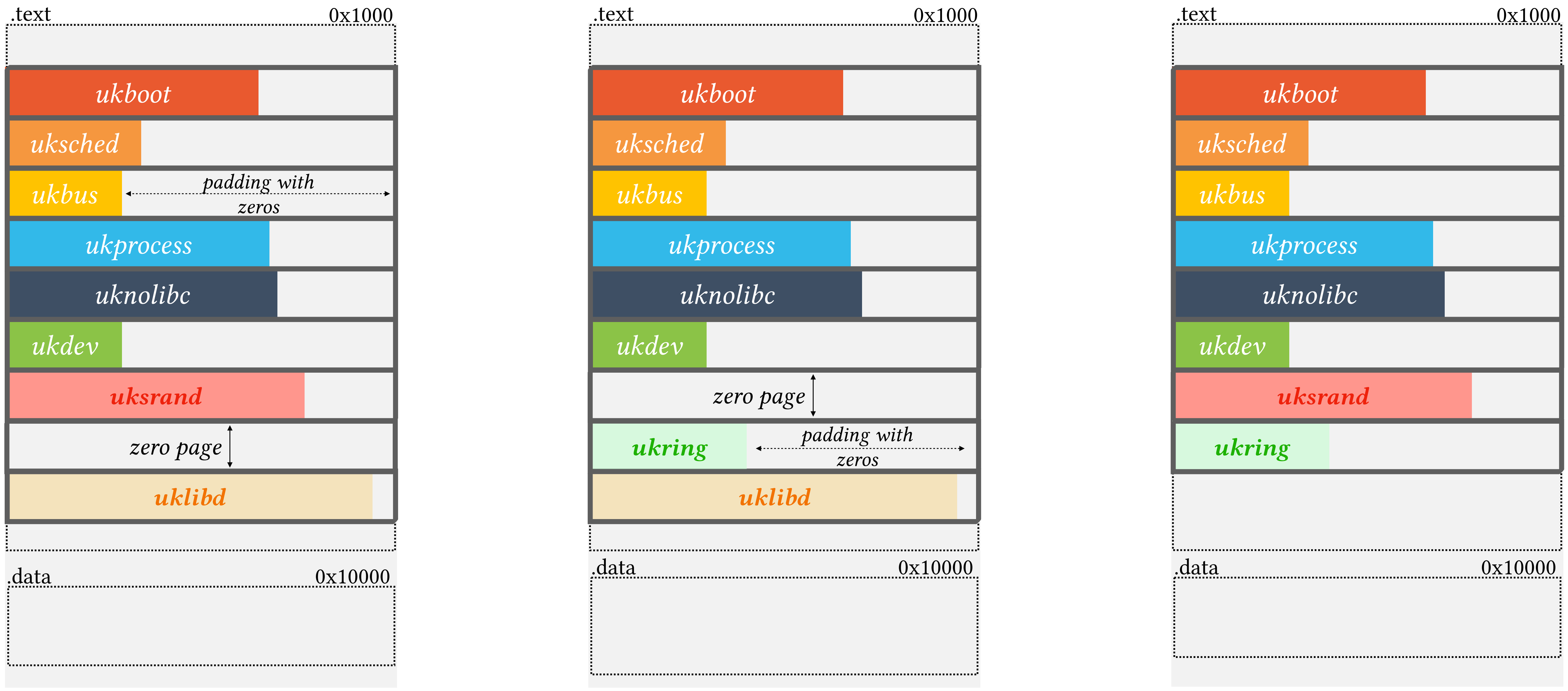
*4 lea **85804**(%rip), %rdx

1. Placing µlibs at page boundary addresses.

2. Keep a same µlibs order.

3. Align sections (e.g., *.data*, *.rodata*, ...) at same addresses.

Memory deduplication with unikernels



Unikernel 1 Unikernel 2 Unikernel 3

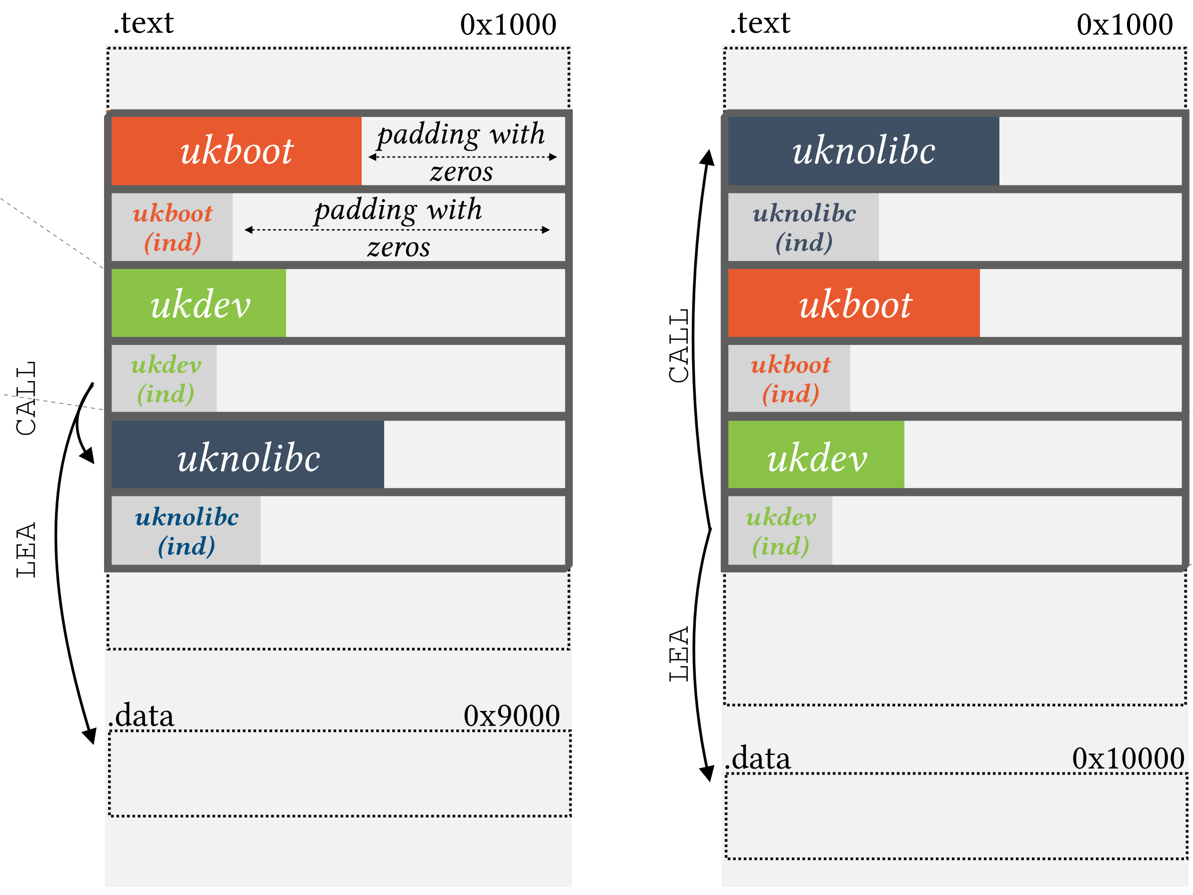
If there are more than two instances with different µlibs subsets:

- It is necessary to align them to specific addresses.
- This leads to ‘gaps’ of zero pages in the memory virtual space.

TOWARDS ASLR SUPPORT

```
ukdev (0x2100):
    jmp eip+1100(0x3200)
    …
ukdev(ind) (0x3200):
    call 0x4200
    jmp 0x2105
```

1. Relative jump to indirection table;
2. Execute the problematic instruction (indirection);
3. Jump back to the microlib code.



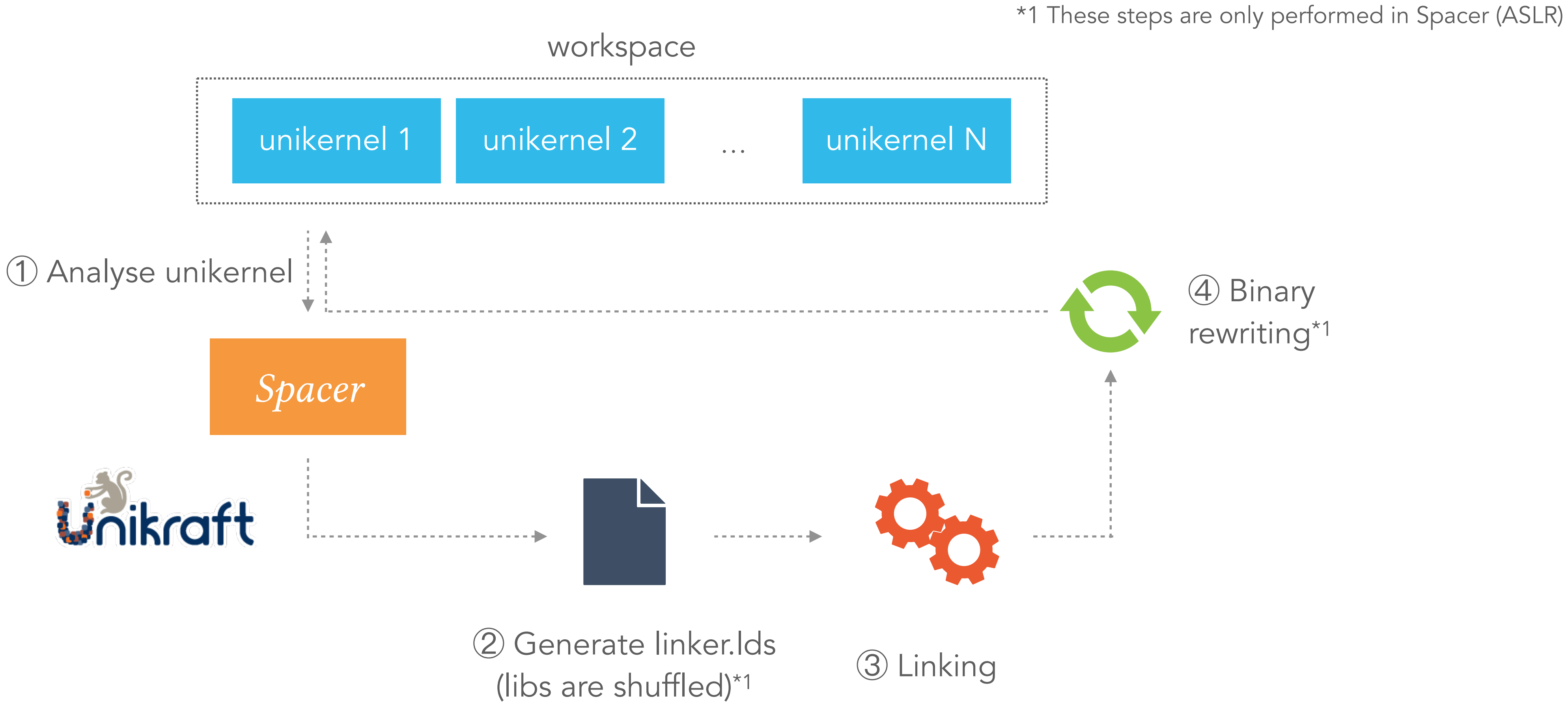
Unikernel1

Unikernel2

```
ukdev (0x4100):
    jmp eip+1100(0x5200)
    …
ukdev(ind) (0x5200):
    call 0x1200
    jmp 0x4105
```

- Using fixed absolute addresses leads to security issues (no ASLR).
- Create an indirection table per µlib which contains problematic instructions (using addresses from other sections/µlibs). Such instructions are replaced by relative jump to their new position.

Spacer High-Level Architecture



- From our methodology, we derive Spacer, a tool aims to have a global knowledge of all the µlibs used by all unikernels on the same workspace.
- Spacer performs a new linking by associating µlibs with absolute addresses according to a map (by rewriting the linker script).
- For Spacer (ASLR), µlibs are shuffled during the linker file generation. Furthermore, there is one extra step of binary rewriting (move problematic instructions to indirection tables).

Evaluation: methodology

- We compared Spacer with DCE (Dead Code Elimination) and Default configuration.
  - 10 applications ported as unikernels.
  - 1000 FaaS unikernels.
  - On several dimensions: memory consumption, file size and performance.

Please refer to the paper for the full evaluation.

Evaluation (1)

Memory consumption:

- Without memory deduplication, Spacer and Spacer (ASLR) consume significantly more memory (zero pages and indirection tables).
- With memory deduplication, the benefits of alignment increases as we run more applications. Spacer and Spacer (ASLR) consume less memory than default and DCE.
- Up to a 3x gain compared to DCE.

Heap-intensive applications:

- The gain is less noticeable (e.g., in-memory databases).
- If there are thousands of applications, Spacer still allows to reduce the memory consumed (code and read-only data are shared).
- But if they are only some instances: do not apply Spacer on it.

EVALUATION (2)

Elf Size:

- Spacer and Spacer (ASLR) have a slight impact on file size:
  - The inflation of the header string table (ELF section).
  - Indirection tables (problematic instructions).
- Spacer ELF files do not have inflation due to zeros, it is only in memory.

Performance:

- Total execution time of short-lived and long-lived unikernels.
- UKSM has a slight impact on scanning and merging pages.
- Spacer performance degradation is minimal: having zero pages and indirection tables introduces a slight overhead (± 5-7% compared to DCE).

Conclusion & Future work

- Unikernels are small and have impressive performance, but they show few opportunities for VM page sharing (specialisation).
- We brought a new methodology that rearranges and inflates unikernels by using µlibs alignment.
  - Aligning µlibs may lead up to a big reduction in memory consumption, even when compared to unikernels built with DCE (Dead Code Elimination).
  - Furthermore, the alignment does not introduce significant overhead in terms of ELF size, nor does it impairs application performance.

Future work:

- *Loader*: A loader that performs deduplication at load time could make µlib pages point directly to the corresponding frames when loading the kernel image into main memory.

THANK YOU!

QUESTIONS?

Spacer & Unikraft

Spacer: https://people.montefiore.uliege.be/gain/spacer

Unikraft: https://unikraft.org/

Contact by email: gaulthier.gain@uliege.be

License: *3-Clause BSD License*

Gaulthier Gain, Cyril Soldani, Felipe Huici, Prof. Laurent Mathy