16TH SYMPOSIUM ON CLOUD COMPUTING (SOCC'25)

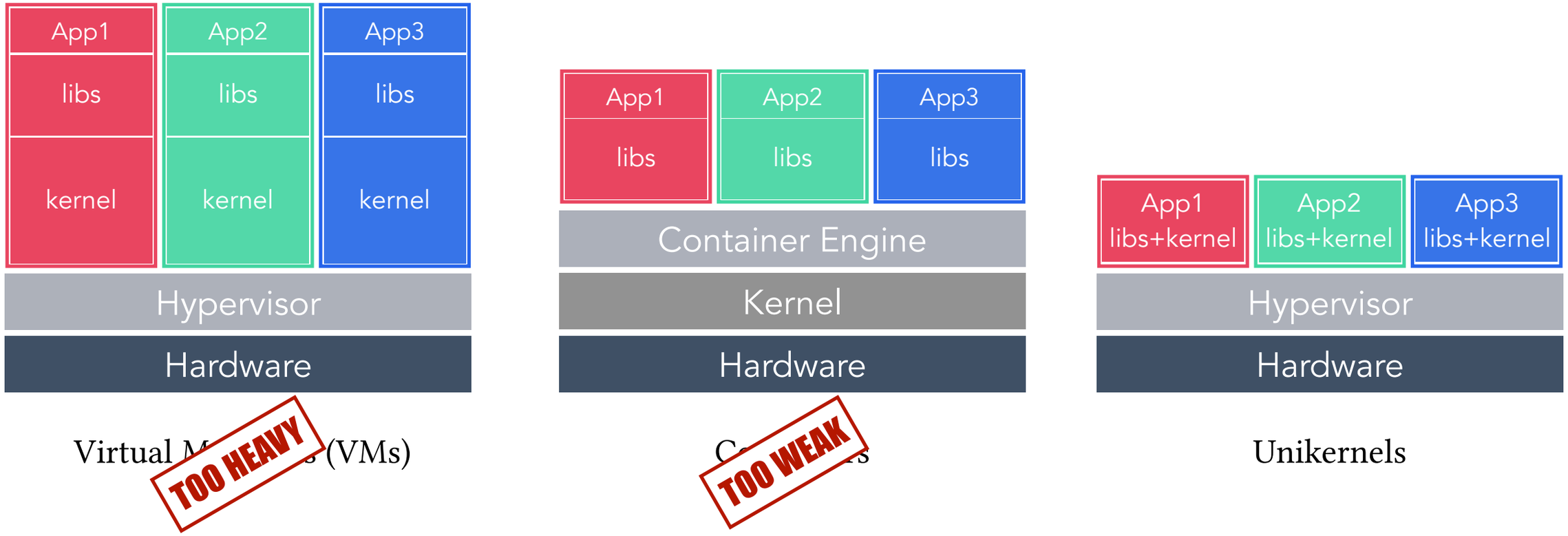
MEMORY MATTERS: LOAD-TIME DEDUPLICATION FOR UNIKERNELS

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

ACM Symposium on Cloud Computing (SoCC' 25),
November 19–21, 2025, Online

UNIKERNELS FOR CLOUD COMPUTING

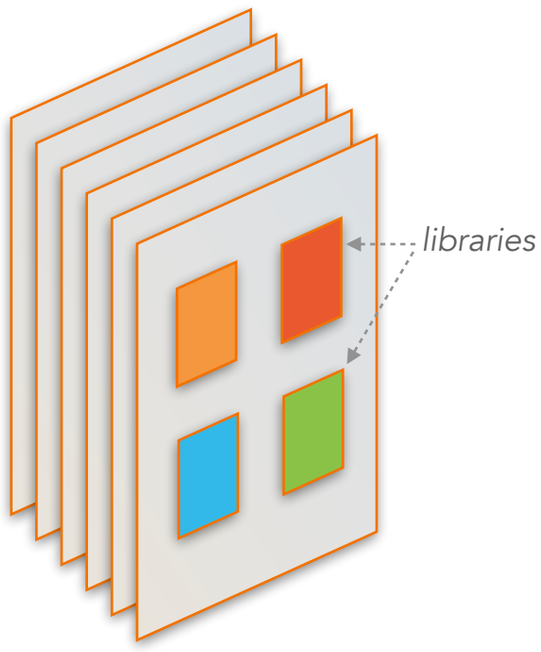
- Cloud workloads increasingly rely on massively parallel, short-lived functions
- Function as a Service (FaaS) platforms demand millisecond-scale startup, high density, and strong isolation



Unikernels are the **ideal** match for **serverless**: fast autoscaling, low overhead, reduced attack surface

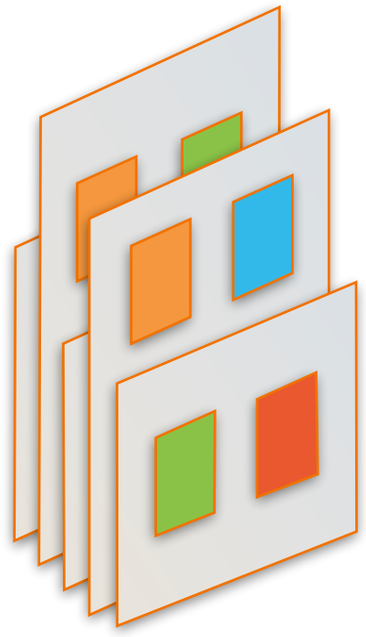
Running a large number of unikernels

Modern cloud platforms may launch **hundreds** or **thousands** of unikernels per machine



Same instance (identical unikernels)

same set of libraries



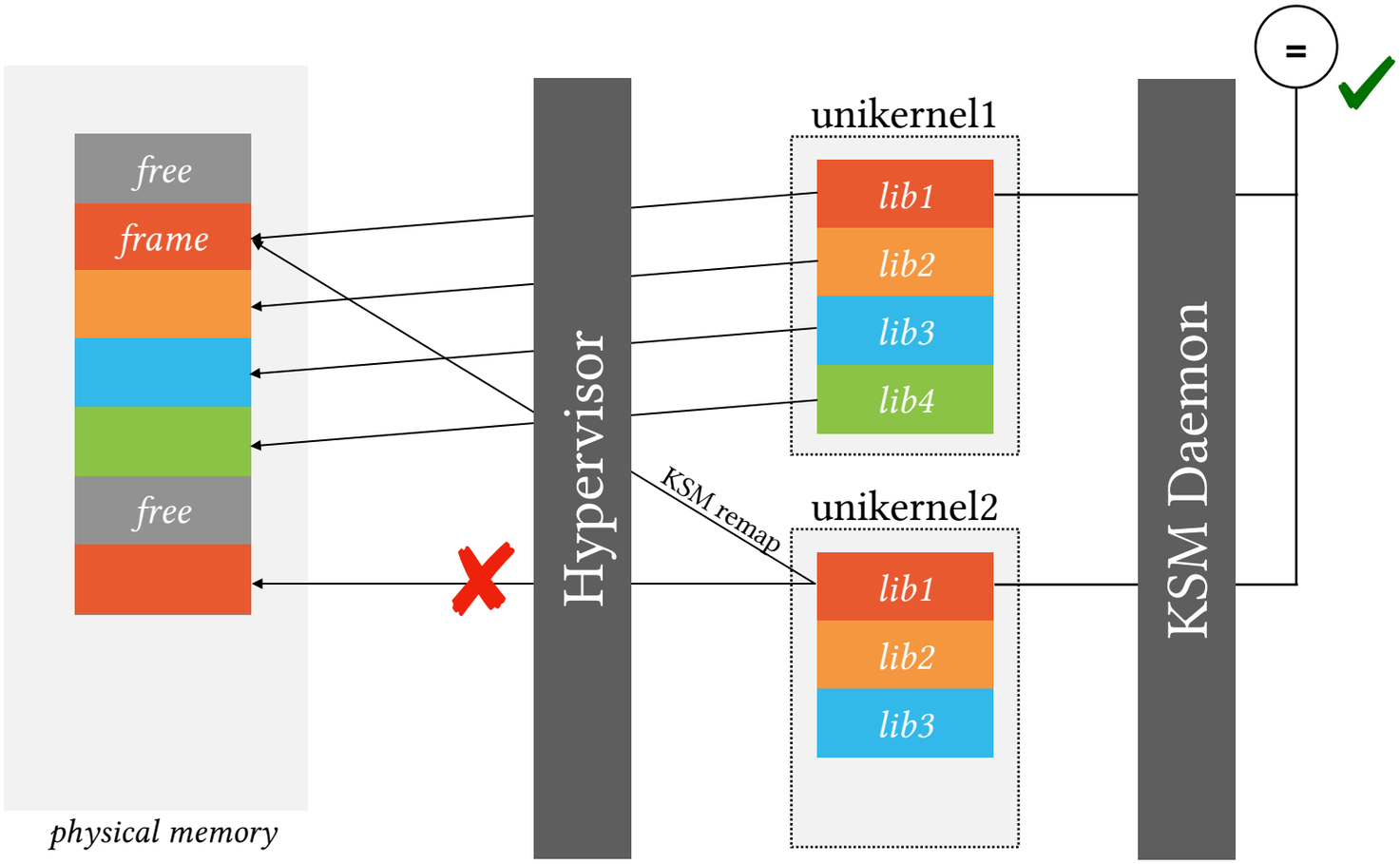
Different instances (≠ unikernels)

≠ sets of libraries

How can we reduce physical memory when dealing with many unikernels?

⇒ *Memory Deduplication*

Memory Deduplication & KSM

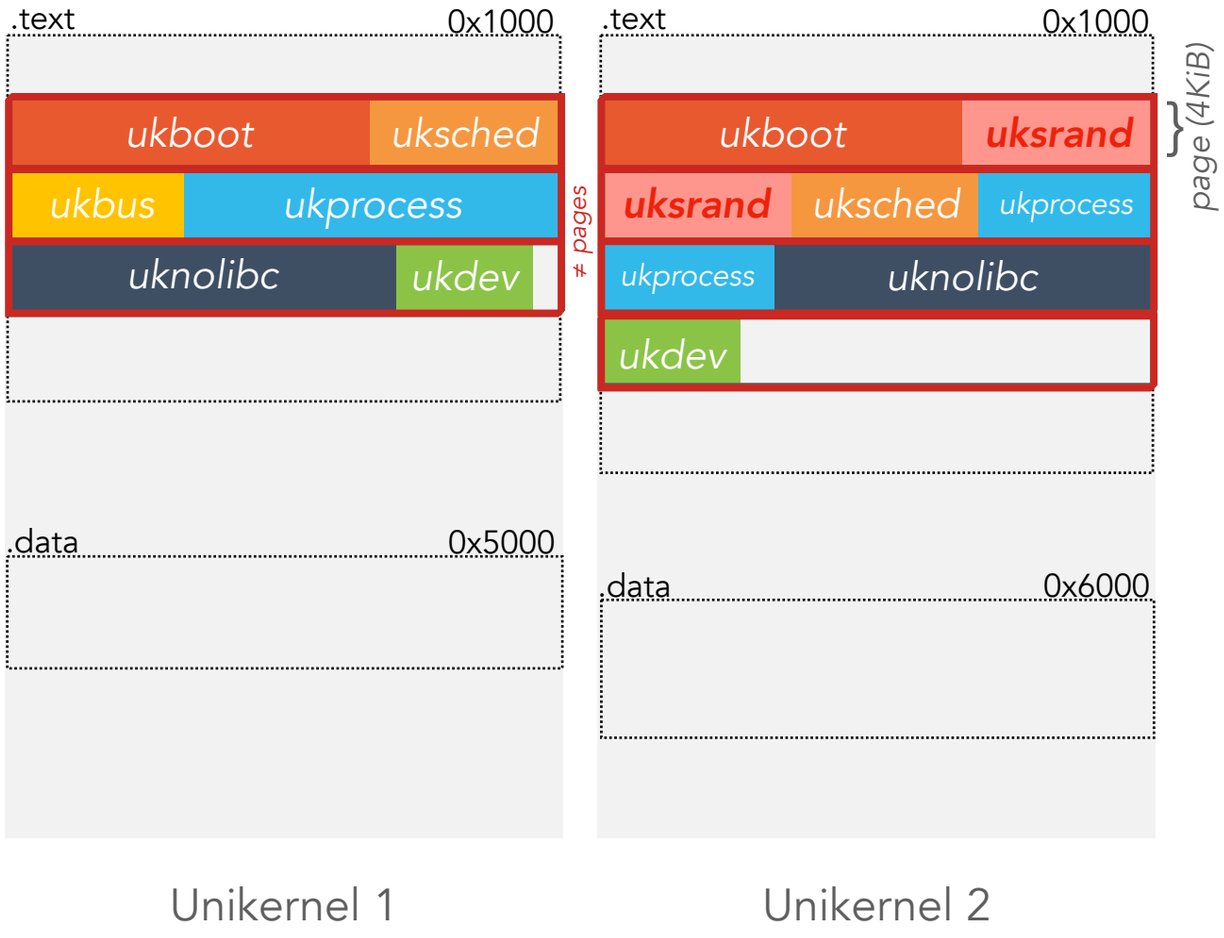


- Identical pages are merged via a memory deduplication runtime scanner
- Linux uses **KSM** (Kernel Samepage Merging) to detect duplicate pages
- KSM periodically scans memory, compares pages (checksum) and merges identical pages into the same single frame

DEALING WITH DIFFERENT UNIKERNELS

Unikernels are highly specialized and compact:

- Having **different** unikernels will result in **different** library configurations
- The library's code is compacted and spread across different pages between unikernels

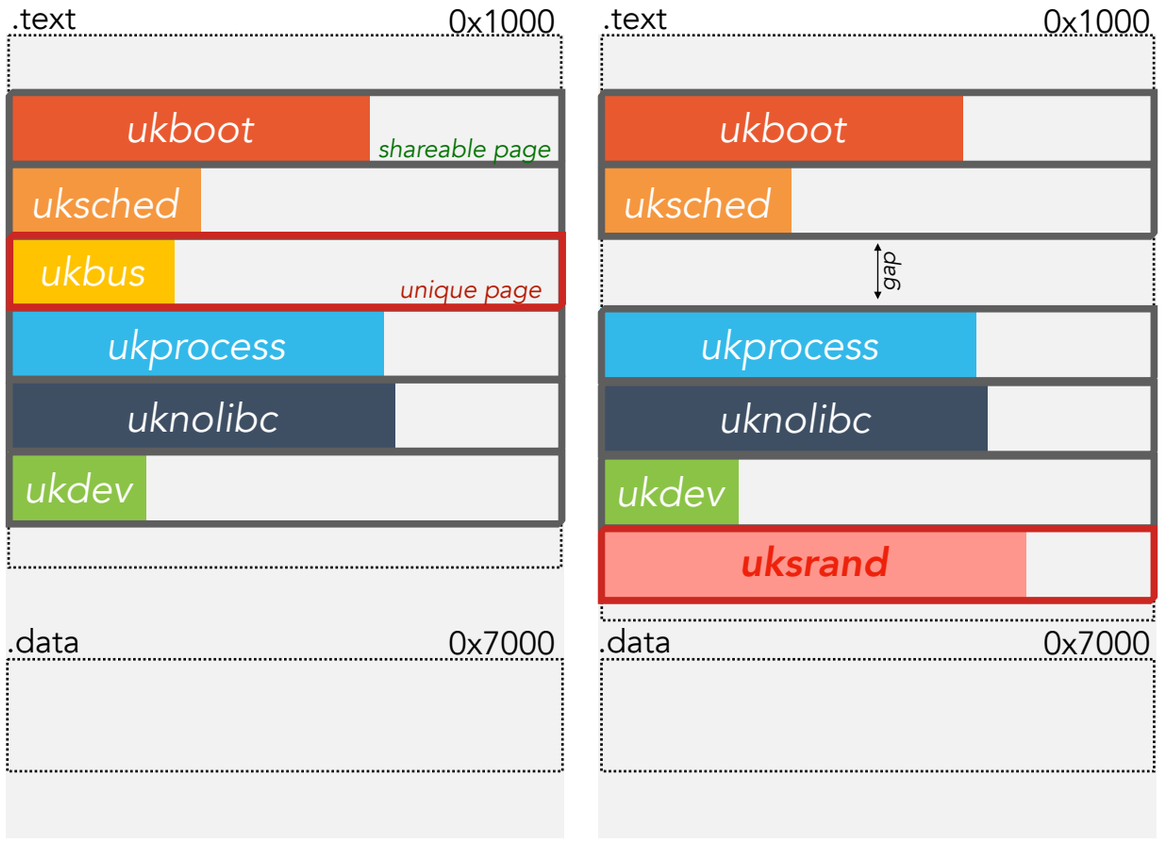


⟹ Different pages: **cannot** be shared between different unikernels, and thus, **higher** memory consumption

RELYING ON EXISTING WORK: SPACER

Increase sharing opportunities by using *Spacer*[1]:

- Unikernels are inflated by aligning libraries at fixed, page-aligned addresses



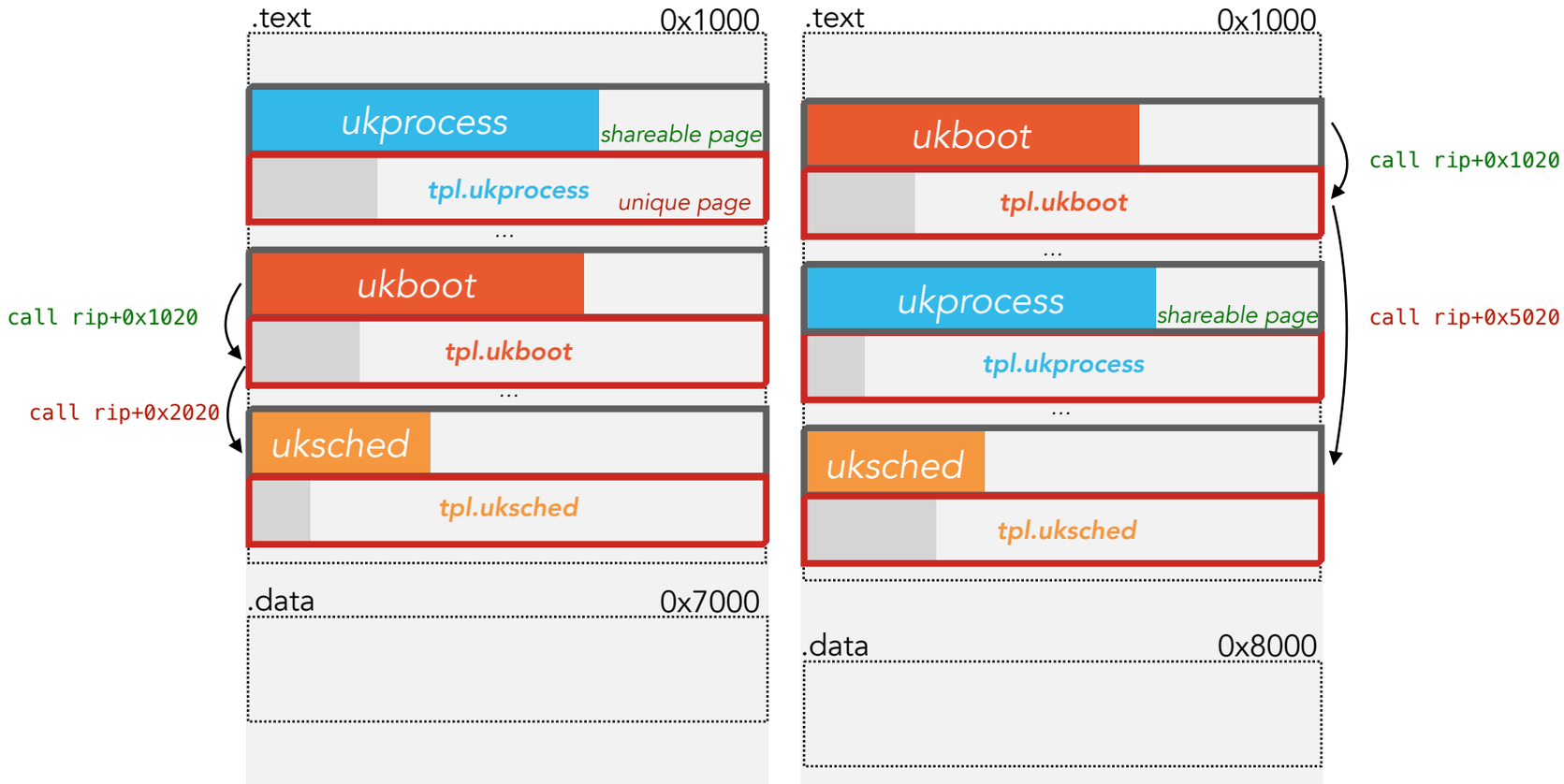
Unikernel 1 Unikernel 2

[1] "*Want More Unikernels? Inflate Them!*", In SoCC'22

Relying on existing work: Spacer

Increase sharing opportunities by using *Spacer*[1]:

- Unikernels are inflated by aligning libraries at fixed, page-aligned addresses



- Supports ASLR through the use of trampoline tables (*tpl*): problematic instructions
- Relies on **KSM** to actually merge identical pages at runtime

[1] "*Want More Unikernels? Inflate Them!*", In SoCC'22

KSM Drawbacks

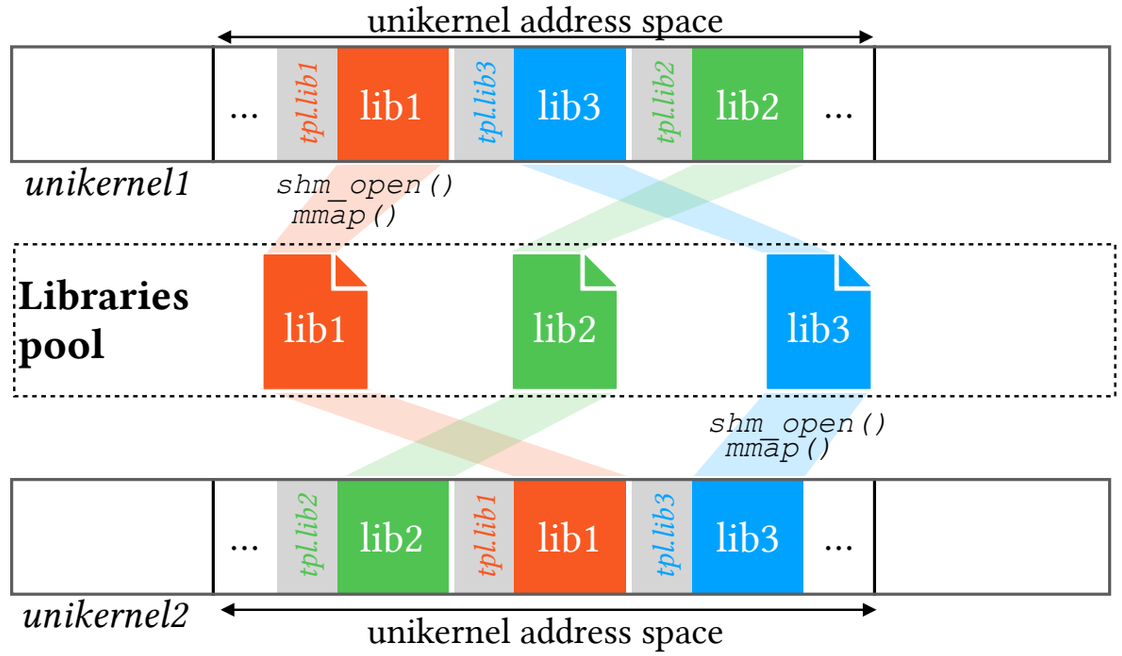
- **Slow convergence**: scanning memory takes time; short-lived functions may finish before pages are merged
- **High CPU overhead**: aggressive scanning consumes significant CPU cycles
- **Non-deterministic savings**: deduplication varies depending on timing and system pressure
- **Security risks**: merging writable pages introduces CoW-based timing side channels
- **I/O and TLB overhead**: merging pages forces extra page faults, copies, and TLB shootdowns

$\Rightarrow$ *Need a better approach*

SPACER-SLT: KEY IDEA

Spacer-SLT: "*Share at Load Time (SLT)*"

- Extends *Spacer* by performing deduplication at load time
- Uses a pool of libraries containing library code & read-only data
- Relies on a custom Firecracker-based loader to map identical pages directly
- Eliminates the need for a memory deduplication runtime scanner (KSM)



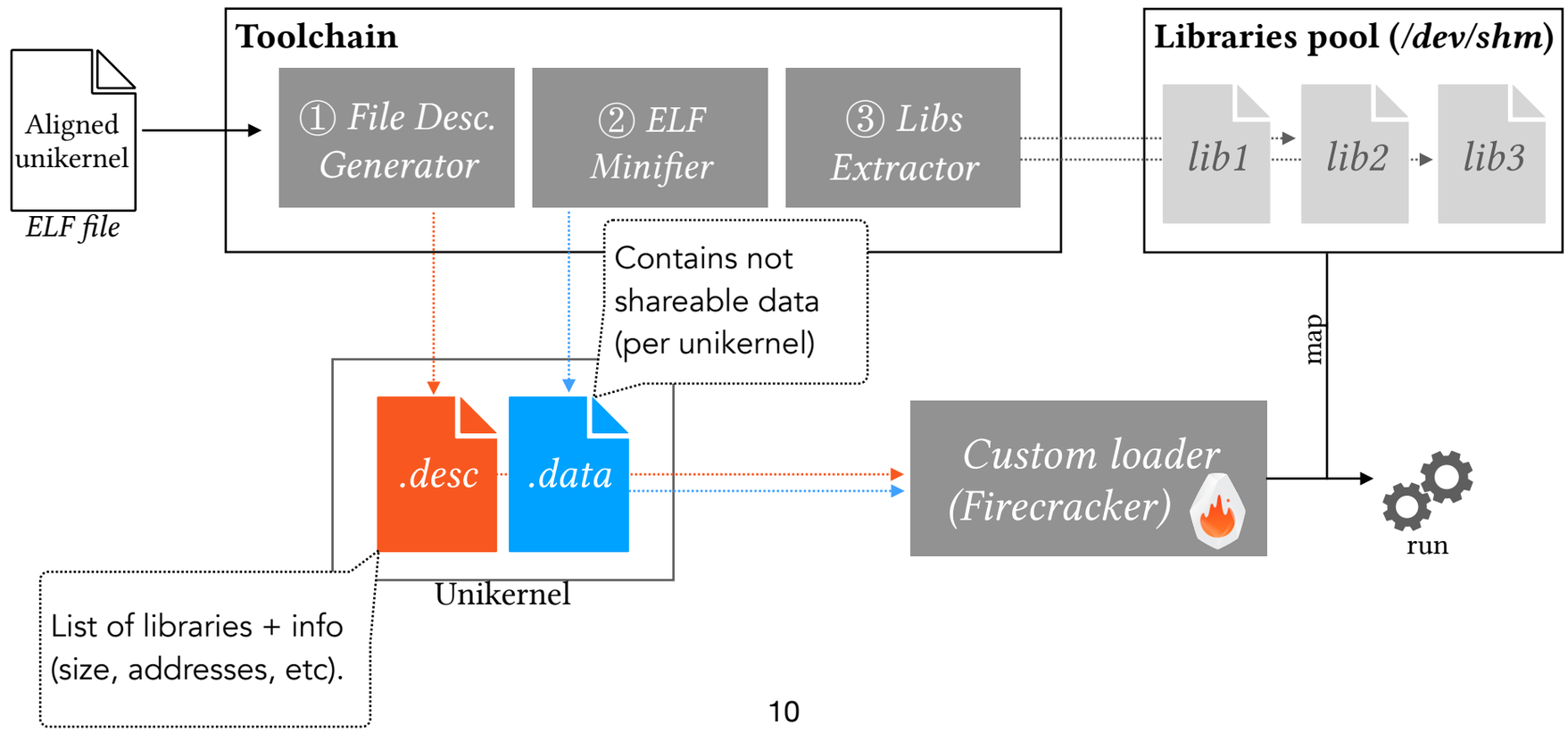
SPACER-SLT: TOOLCHAIN

- Performs all operations offline, eliminating overhead during unikernel runtime
- Use a toolchain that consists of several components:

① **File Description Generator**: Produces a description file (*.desc*) for each unikernel by analyzing its ELF layout

② **ELF Minifier**: Strips the ELF down to a minimal form, retaining only the aggregated writable data section (and trampoline tables for ASLR) into a *.data* file

③ **Libraries Extractor**: Extracts compiled libraries as standalone binary files and places them into a shared library pool

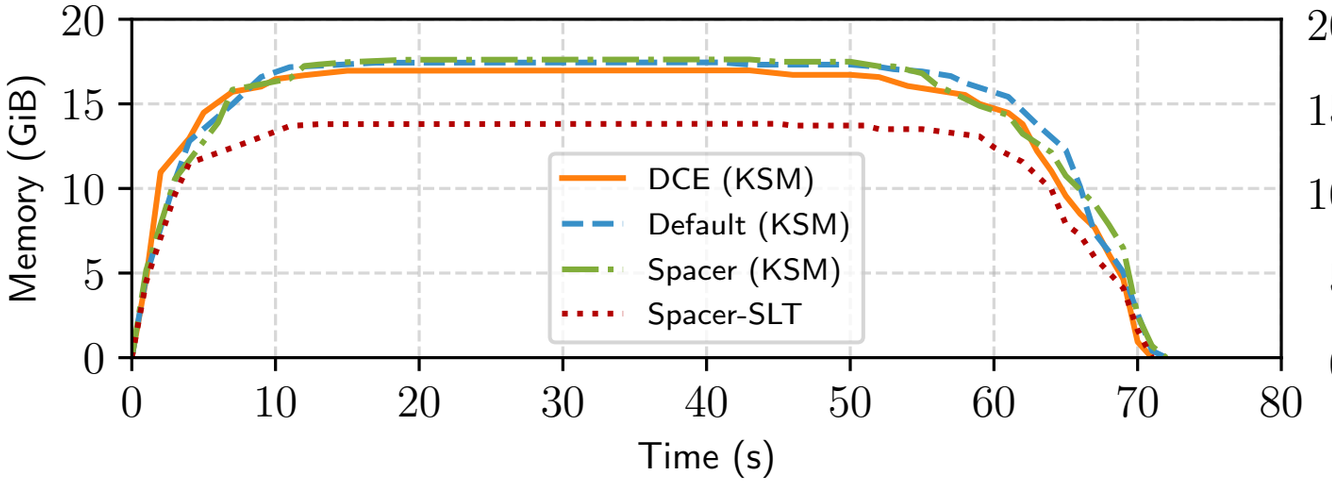


Evaluation: Methodology

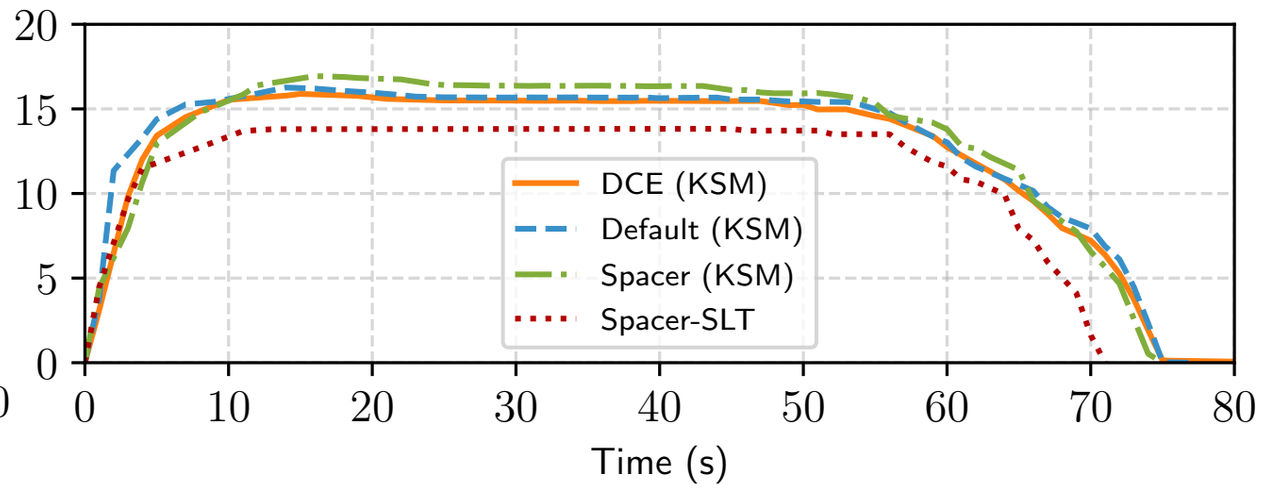
- Rely on the Unikraft Framework and Firecracker as a custom loader
- Compare Spacer-SLT against **DCE (Dead Code Elimination)**, **Default Unikraft**, and **Spacer**
- Evaluate both **homogeneous** and **heterogeneous** workloads:
  - 128 Go-based FaaS unikernels (identical instance)
  - 20 different applications ported as unikernels NGINX SQLite …
- Also, measure the **KSM behavior** under 2 modes:
  - *KSM-quiet (default): scan 100 pages every 20 ms*
  - *KSM-full (aggressive): scan 20000 pages every 1 ms*
- On several dimensions: memory consumption, file size and performance

Please refer to the paper for the full evaluation.

Evaluation: Memory



(a) ***KSM-quiet***: KSM takes time to identify and merge duplicate pages, resulting in minimal memory savings.



(b) ***KSM-full***: KSM achieves better memory reduction by using more CPU cycles, but it is overwhelmed by the rapid arrival of many unikernels.

Running many identical instances:

- Run 128 identical FaaS unikernels (Go) and benchmarked according to 2 KSM modes: ***KSM-quiet*** & ***KSM-full***
- Spacer-SLT achieves the **lowest** memory consumption in both cases **without** any performance penalties
- Up to 30% additional memory **savings** with Spacer-SLT compared to KSM-based configurations

Evaluation: Memory

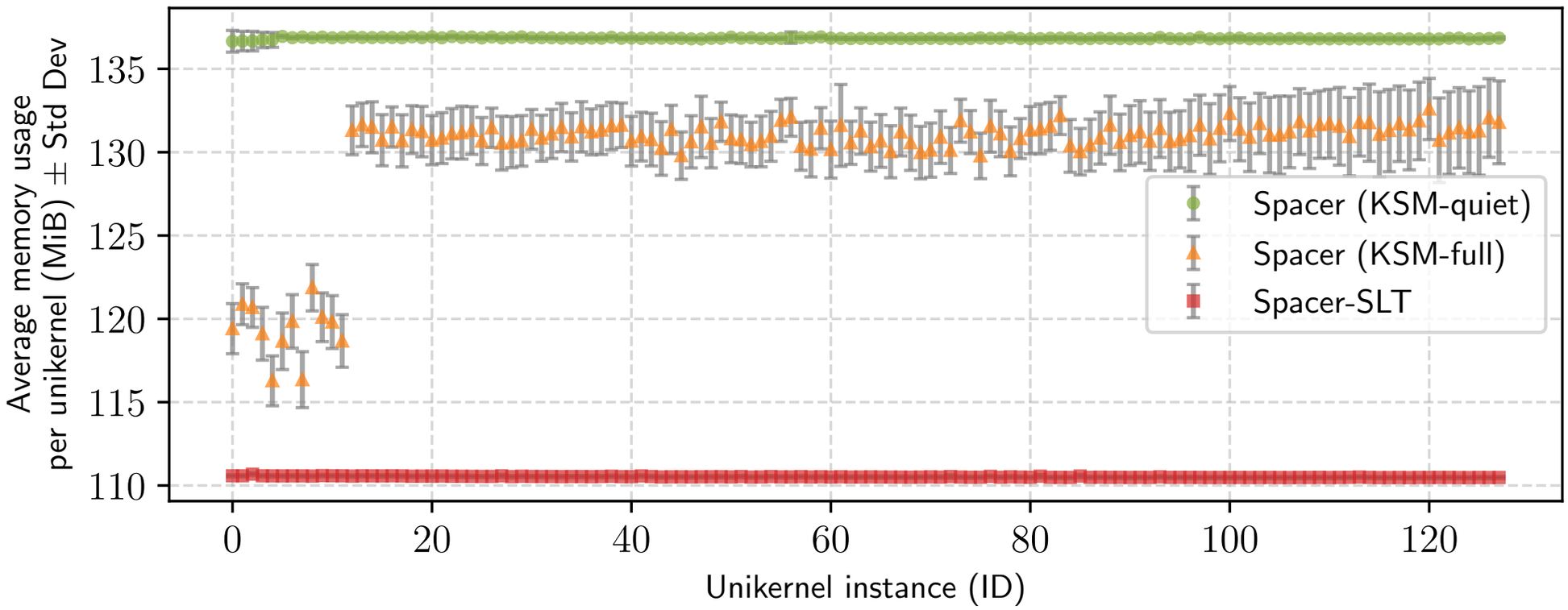


Figure: Average physical memory usage per instance for 128 benchmarked FaaS unikernels (including hypervisor).

- KSM-based configurations show **unstable** behaviour
  - Under ***KSM-full***, the first few unikernels benefit from memory deduplication
  - As the number of instances grows, KSM becomes **overloaded** and cannot merge effectively
- Spacer-SLT maintains **consistent** and **predictable** memory usage across all instances

Evaluation: Memory

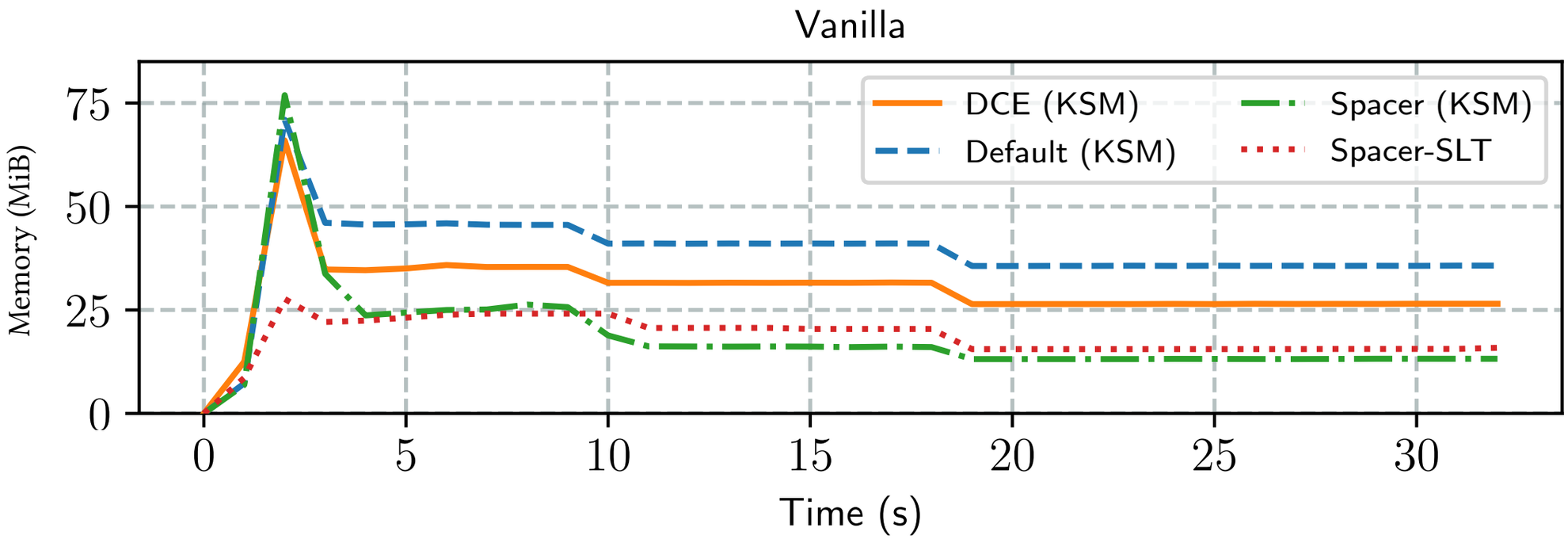


Figure: Evolution of the physical memory used by 20 different unikernels (all started at t=0s)

Running different instances:

- With KSM, memory peaks appear during startup
- KSM also merges identical writable pages, such as those related to the heap and video memory
  - KSM might achieve higher memory reduction than Spacer-SLT, but it comes with inherent issues

Evaluation: Performance & Filesize

Spacer-SLT Performance:

Library pool acting as a cache:

- Fewer page faults: up to 7× less
- Faster load times: up to 1.4× faster

No KSM daemon:

- No runtime CPU cost and no CoW faults
- 10–13× fewer TLB flushes, avoiding costly TLB shootdowns

$\Longrightarrow$ Better execution time: $\approx$ 10 % gain compared with *KSM-quiet*

$\Longrightarrow$ even **better** with *KSM-full*

Spacer-SLT Elf Size:

- Thanks to the library pool, the unikernels filesize is reduced by up to 5.8×

Why Not Dynamic Linking?

Ported Spacer-SLT and the Unikraft codebase to **userspace** *(reduced effort)*

- Extended Unikraft: support dynamically linked unikernels (extensive modifications)
- Implemented a lightweight userspace loader to run them

Apps	Spacer-SLT	SLT [ASLR]	Dynamic
helloworld	0,0000211	0,0000227	0,00003
nginx	0,174	0,177	0,211
mandelbrot	0,358	0,359	0,37
lambda	0,539	0,55	0,56
python	2,114	2,137	2,271
zlib	3,814	3,853	4,095
sqlite-speed	47,503	47,857	48,529

Table: Total execution time (in seconds)

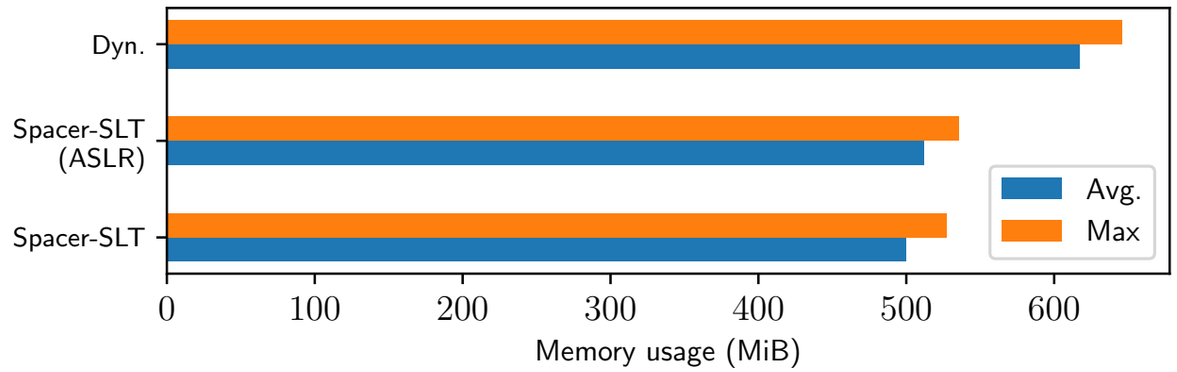


Figure: Memory usage of different unikernels (and the loader)

Dynamically linked unikernels consume **more memory** and have **higher** execution time:

- Memory penalty: overhead of the dynamic linker and additional structures such as PLT/GOT
- Time penalty: library relocations and symbol resolutions

\+ ***Security issues*** *(e.g., GOT overwritten, larger attack surface, etc.)*

Conclusion

- **Spacer-SLT:** deterministic **load-time** deduplication for statically linked unikernels
- Delivers **big wins** across the board: instant memory deduplication, fewer page faults, faster startup & execution time
- Enhances the efficiency of cloud infrastructures, especially for Microservices, FaaS, and short-lived workloads
- Easily **adaptable** to other unikernel frameworks and hypervisors
- Open source and available on GitHub

THANK YOU!

QUESTIONS?

gaulthier.gain@uliege.be

LIÈGE université