\Orchestrating a brighter world NEC

Lancaster University

Building Extremely Fast, Specialized Unikernels The Easy Way

Alexander Jung <a.jung@lancs.ac.uk>
Felipe Huici <felipe.huici@neclab.eu>
Sharan Santhanam <sharan.santhanam@neclab.eu>
Simon Kuenzer <simon.kuenzer@neclab.eu>

FOSDEM'21

This work has received funding from the European Union's Horizon 2020 research and innovation program under grant agreements no. 871793 ("ACCORDION") and 825377 ("UNICORE"). This work reflects only the author's views and the European Commission is not responsible for any use that may be made of the information it contains.

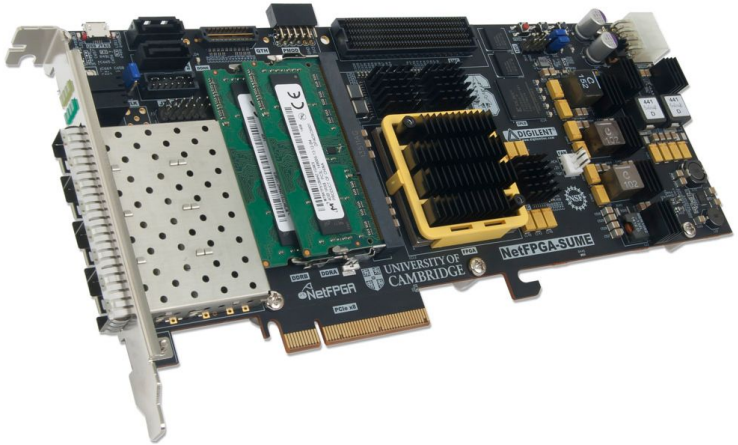
UNICORE ACCORDION

Specialization = High Performance

Specialization = High Performance

Hardware

- TPUs
- Movidius
- FPGAs



Costly…

inherently scoped…

Specialization = High Performance

Networking

- **Sandstorm** \w Marinos, Ilias, Robert NM Watson, and Mark Handley. "*Network stack specialization for performance.*" ACM SIGCOMM Computer Communication Review 44.4 (2014): 175-186.

- Kuenzer, Simon, et al. "*Towards minimalistic, virtualized content caches with **MiniCache**.*" Proceedings of the 2013 workshop on Hot topics in middleboxes and network function virtualization. 2013.

- Martins, Joao, et al. "***ClickOS** and the art of network function virtualization.*" 11th USENIX Symposium on Networked Systems Design and Implementation (NSDI'14). 2014.

Specialization = High Performance

Language-specific runtime environments

- **MirageOS** \w Madhavapeddy, Anil, and David J. Scott. "Unikernels: Rise of the virtual library operating system." Queue 11.11 (2013): 30-44.
- **Erlang on Xen (LING)** http://erlangonxen.org
- **runtime.js** http://runtimejs.org/

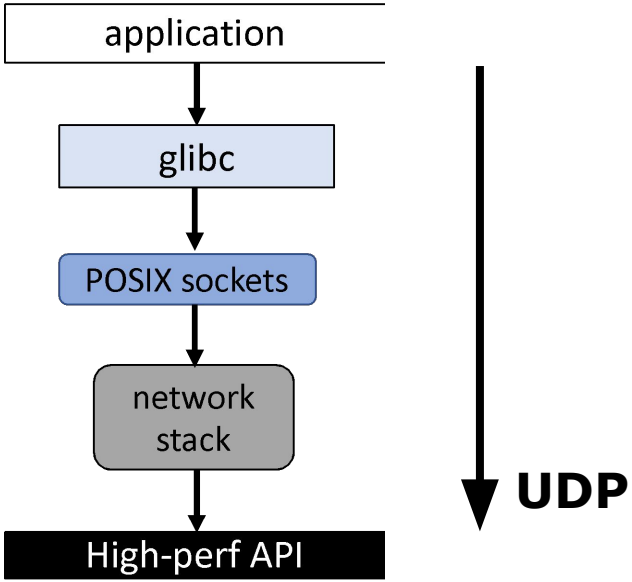
Specialization in Virtualization = Unikernels

1. Small image size
2. Fast boot times
3. Low memory consumption
4. High throughput
5. *Potentially* more secure

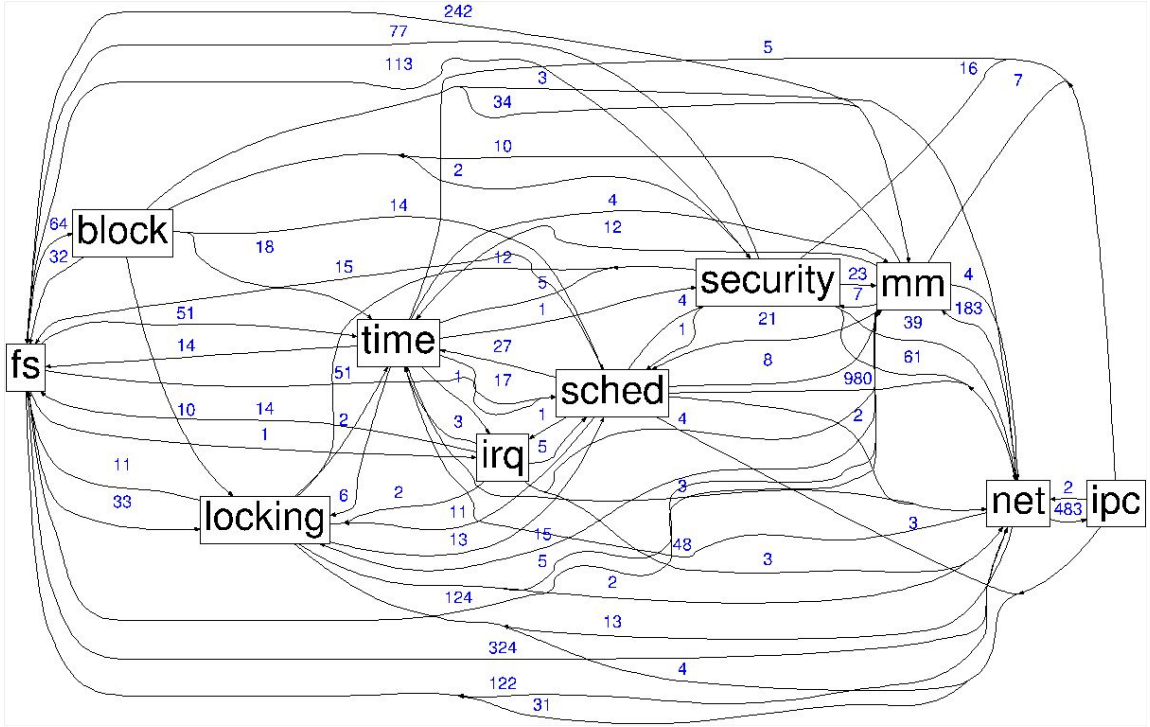
Achieving Unikernel Performance

1. **Transparently:** *applications are ported and automatically benefit from lower boot times, less memory consumption, etc.*

2. **Modified:** *applications are hooked into high performance APIs at the right level in the software stack*



Doing it with Linux?



Doing it with existing unikernels?

1. They **require significant expert work to build** and to extract high performance; such work has to for the most part be redone for each target application.

2. They are **often non-POSIX compliant**, requiring porting of applications and language environments.

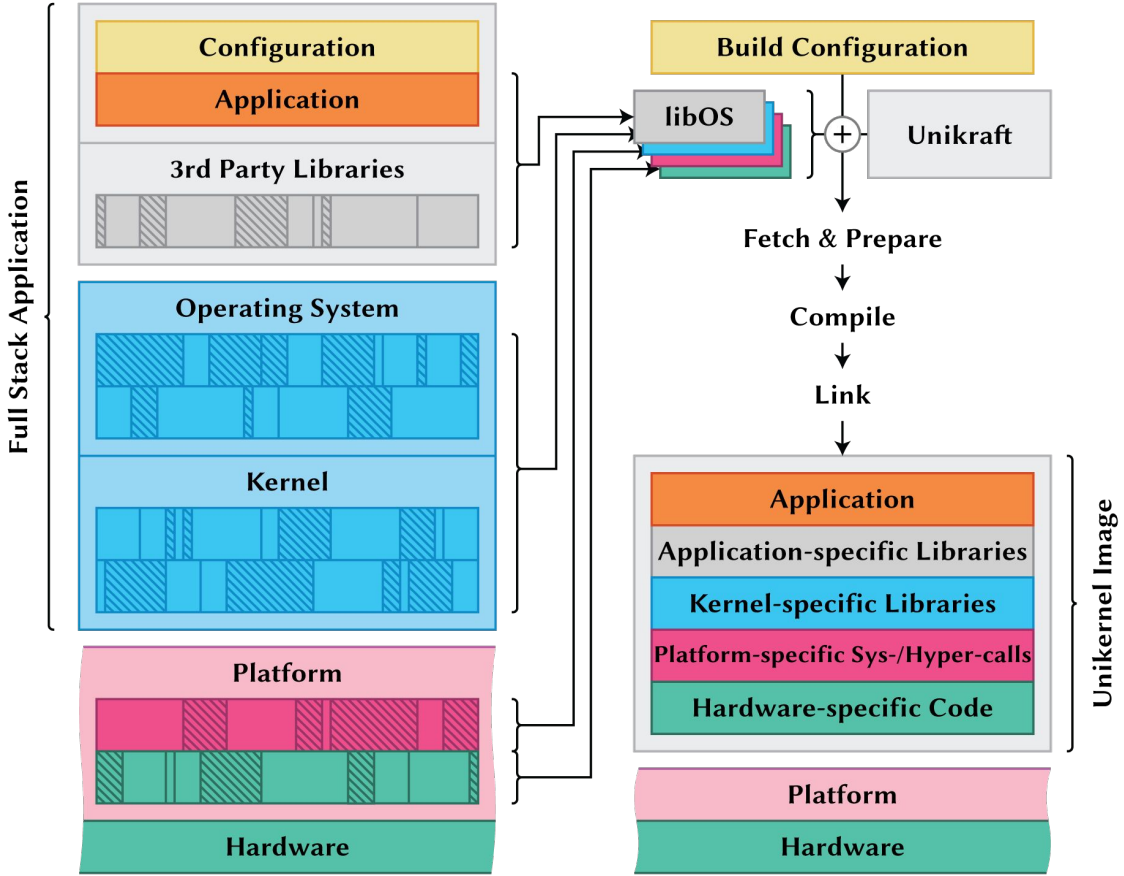
3. The (uni)kernels themselves, while smaller, are *still* monolithic and hard to customize.

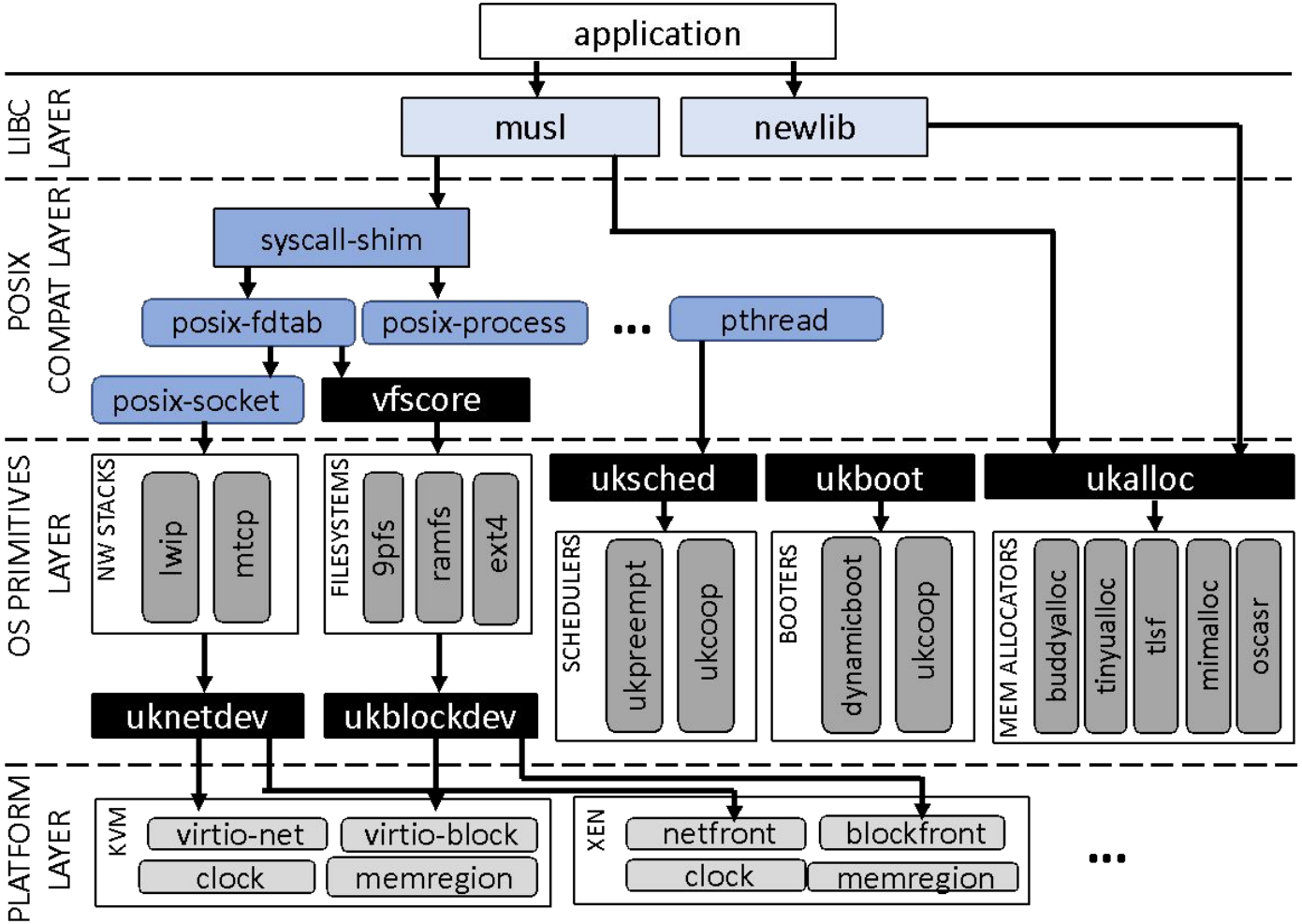
Design Principles

1. **The kernel should be fully modular** in order to allow for the unikernel to be fully and easily customizable.

2. **The kernel should provide a number of performance-minded**, well-defined APIs that can be easily selected and composed in order to meet an application's performance needs.

Unikraft Overview



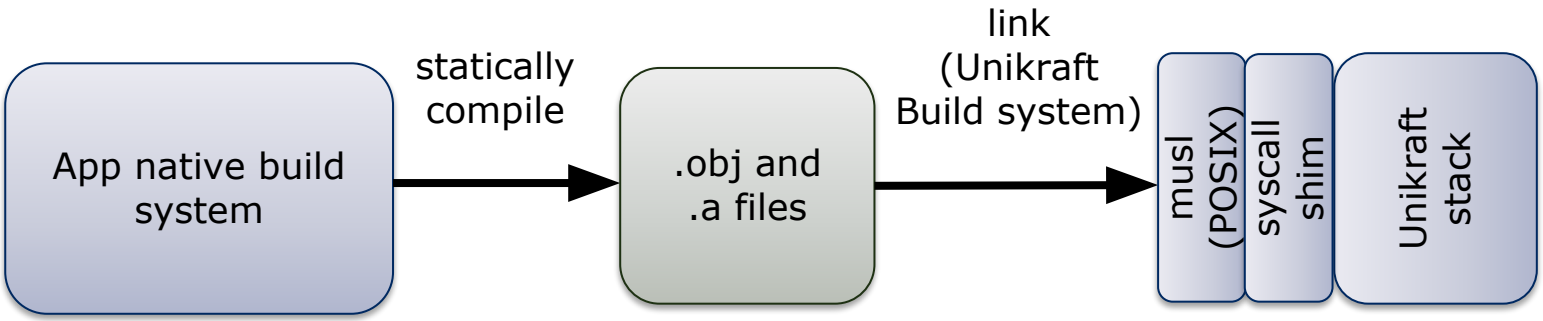


But is it possible to provide transparent application support?

How does Binary Compatibility compare?

Platform	**Routine call**	**# Cycles**	**nsecs**
Linux/KVM	System call	604.62	232.55
	System call (no mitigations)	142.31	54.74
Unikraft/KVM	System call	85.0	32.69
Both	Function call	6.0	2.31

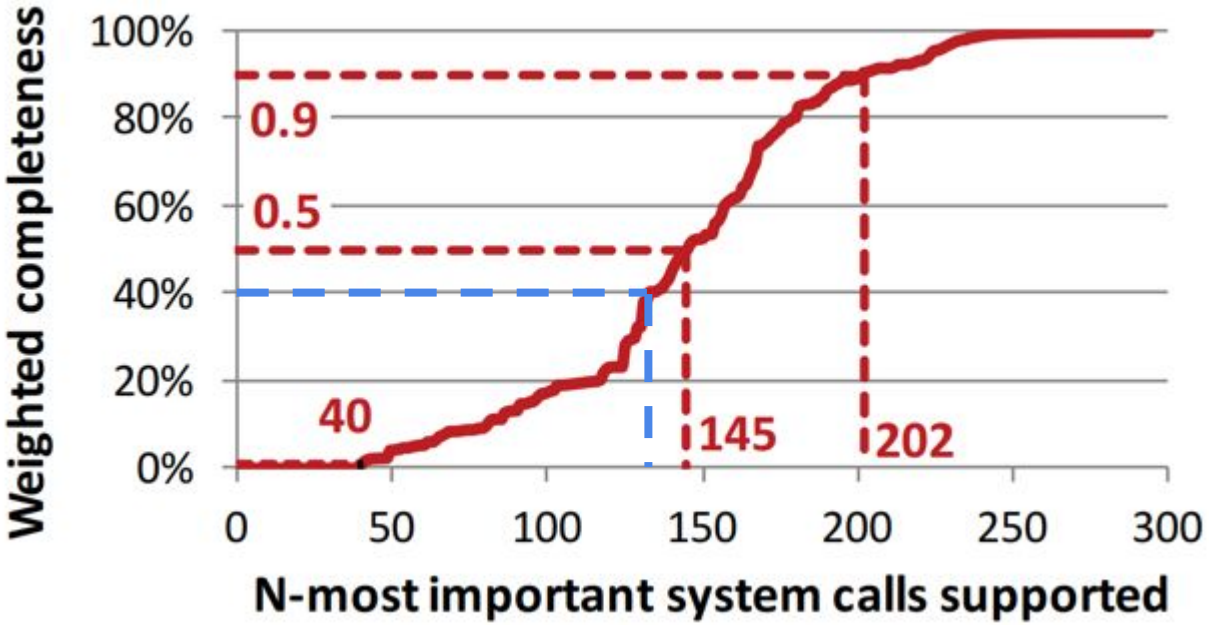
Transparently Building from Source?



Compile time

	musl			newlib		
	Size (MB)	std	compat. layer	Size (MB)	std	compat layer
lib-axtls	0.336	✗	✓	0.432	✗	✓
lib-bzip2	0.296	✓	✓	0.364	✗	✓
lib-c-ares	0.304	✓	✓	0.432	✗	✓
lib-duktape	0.700	✓	✓	0.772	✗	✓
lib-farmhash	0.232	✓	✓	0.276	✓	✓
lib-fft2d	0.356	✓	✓	0.396	✗	✓
lib-helloworld	0.232	✓	✓	0.256	✓	✓
lib-libucontext	0.232	✓	✓	0.276	✓	✓
lib-libunwind	0.232	✓	✓	0.276	✗	✓
lib-lighttpd	0.796	✗	✓	0.916	✗	✓
lib-lighttpreply	0.256	✓	✓	0.296	✓	✓
lib-memcached	0.524	✓	✓	0.672	✗	✓
lib-micropython	0.527	✓	✓	0.628	✗	✓
lib-nginx	1.13	✗	✓	1.20	✗	✓
lib-open62541	0.248	✗	✓	0.804	✗	✓
lib-openssl	2.98	✗	✓	3.01	✗	✓
lib-pcre	0.344	✓	✓	0.380	✗	✓
lib-python	4.75	✗	✓	4.81	✗	✓
lib-redis-client	0.640	✗	✓	0.801	✗	✓
lib-redis-server	1.26	✗	✓	1.42	✗	✓
lib-ruby	6.84	✗	✓	6.93	✗	✓
lib-sqlite	1.22	✓	✓	1.31	✗	✓
lib-zlib	0.348	✓	✓	0.404	✗	✓
lib-zydis	0.276	✓	✓	0.328	✗	✓

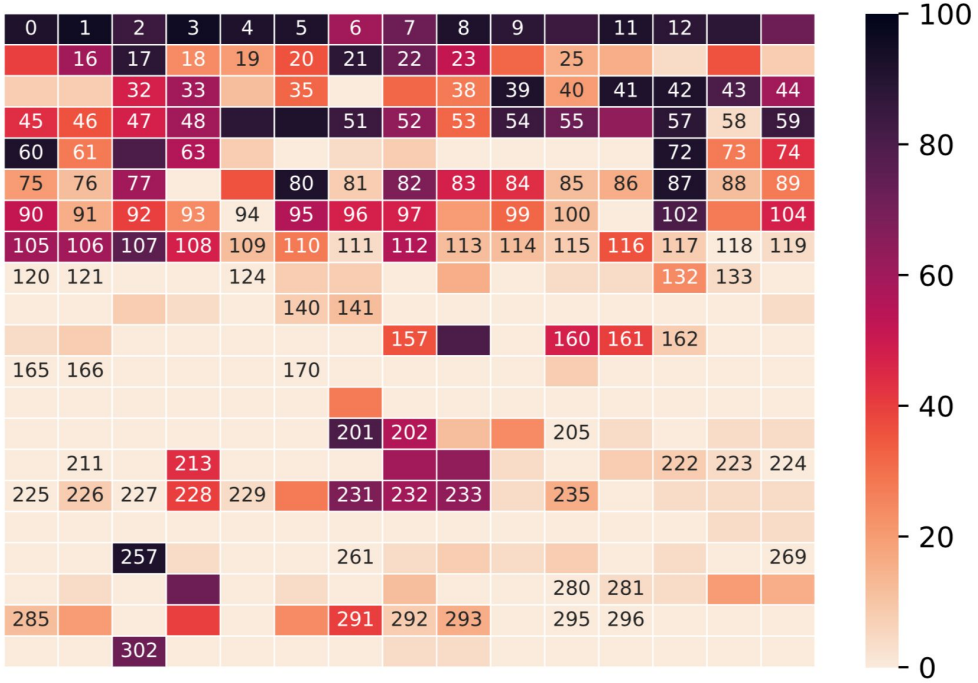
How much syscall support is enough?



Unikraft ~140

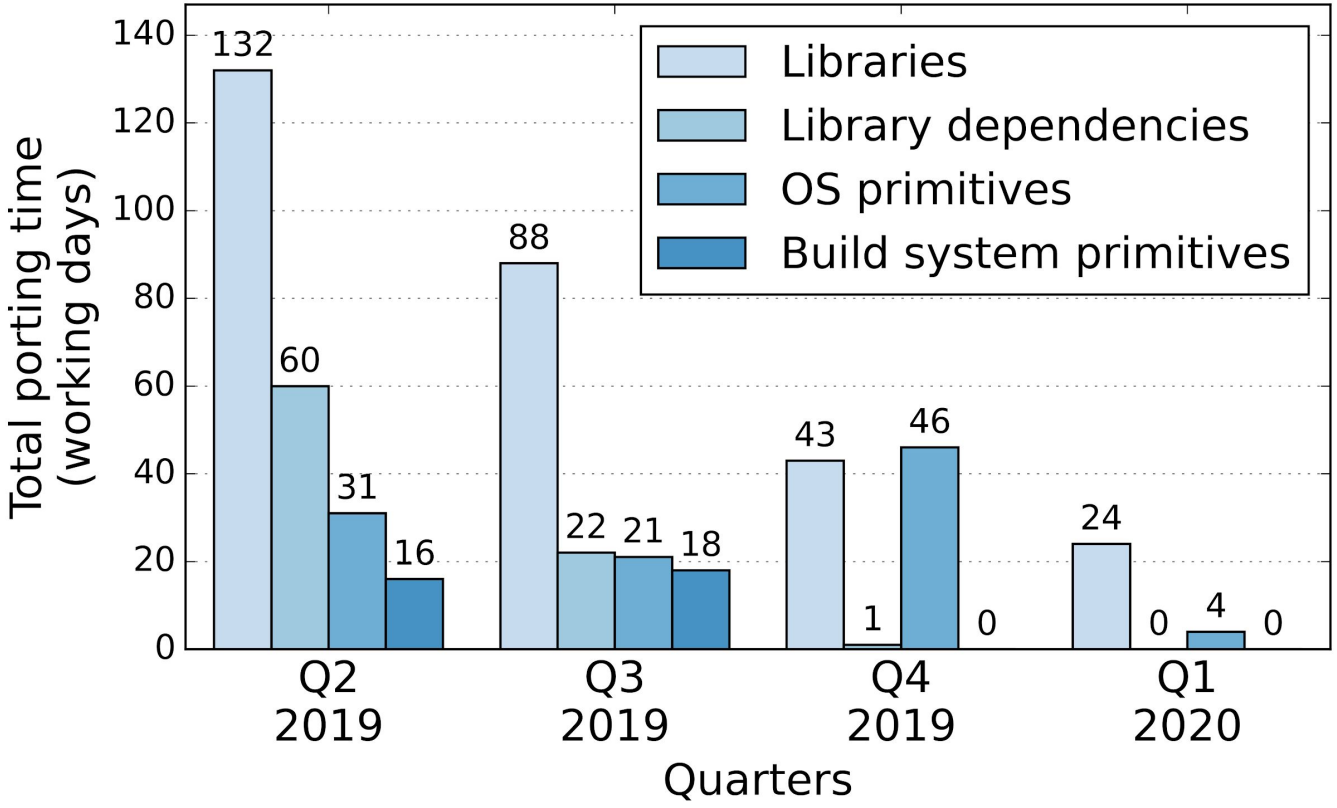
A study of modern Linux API usage and compatibility: what to support when you're supporting.
Tsai et. Al, Eurosys 2016

What Unikraft ***Could Transparently*** Support



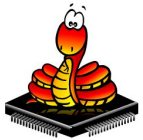
Syscalls required by a set of 30 popular server apps vs. Syscalls currently supported by Unikraft

If all else fails – Manual Porting



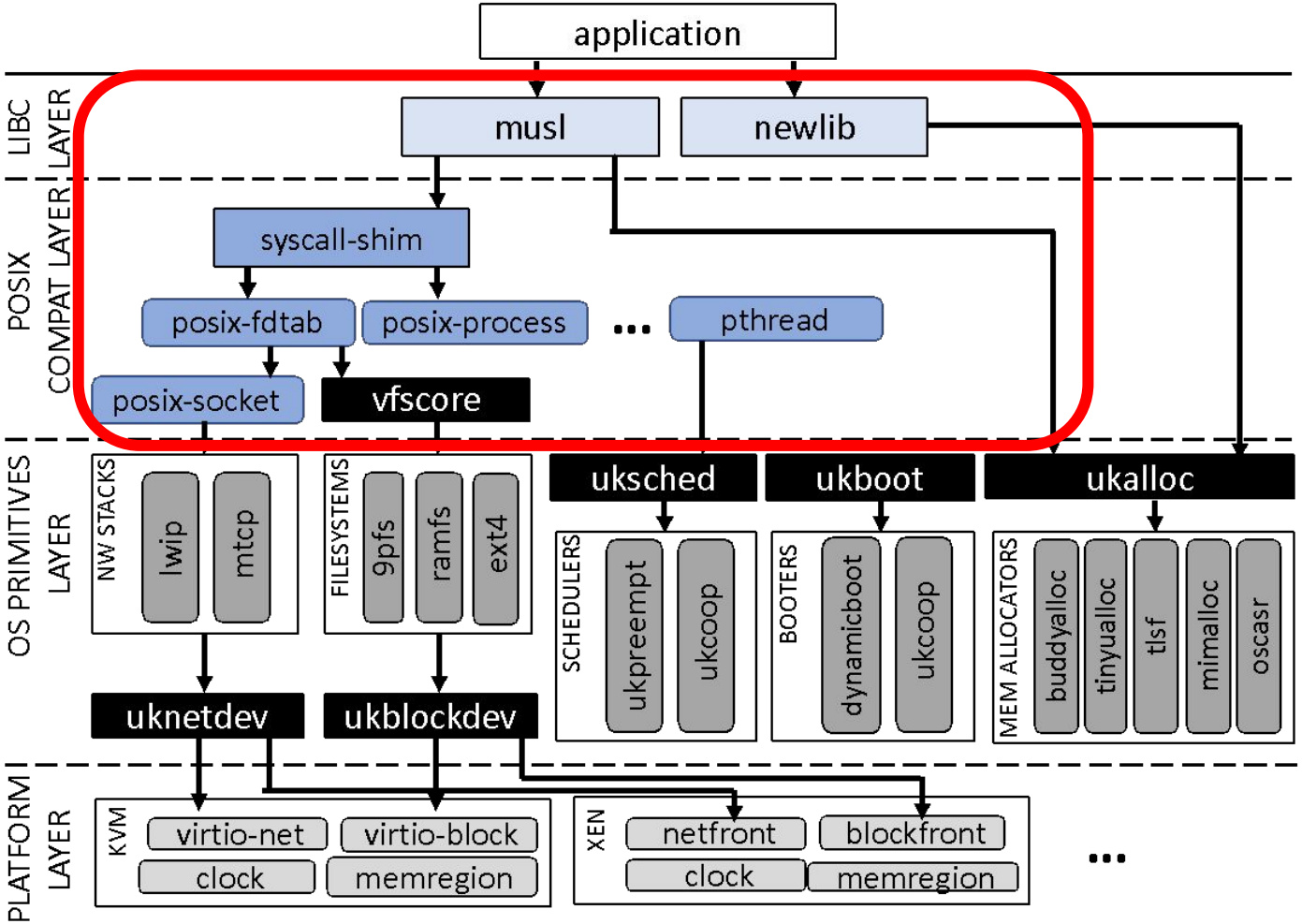
What Unikraft Supports



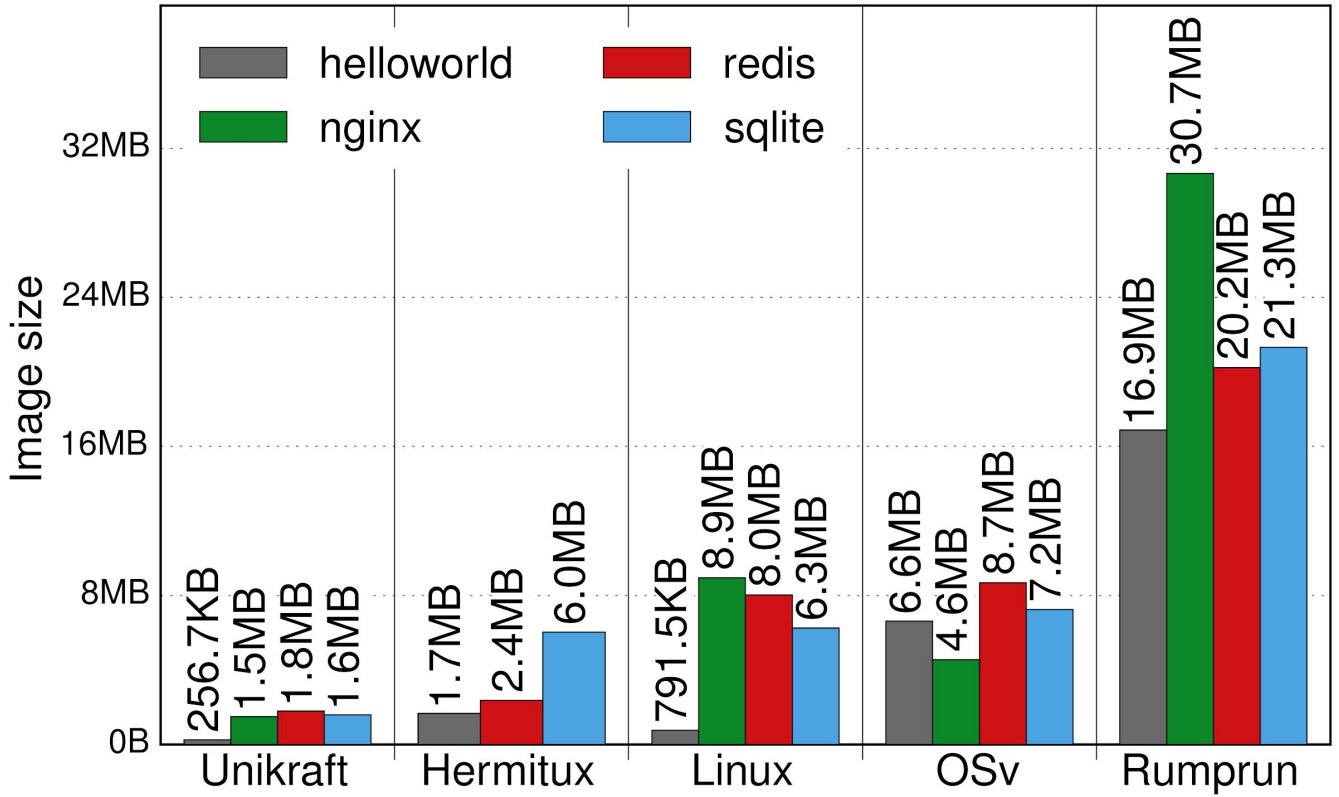




On-going:

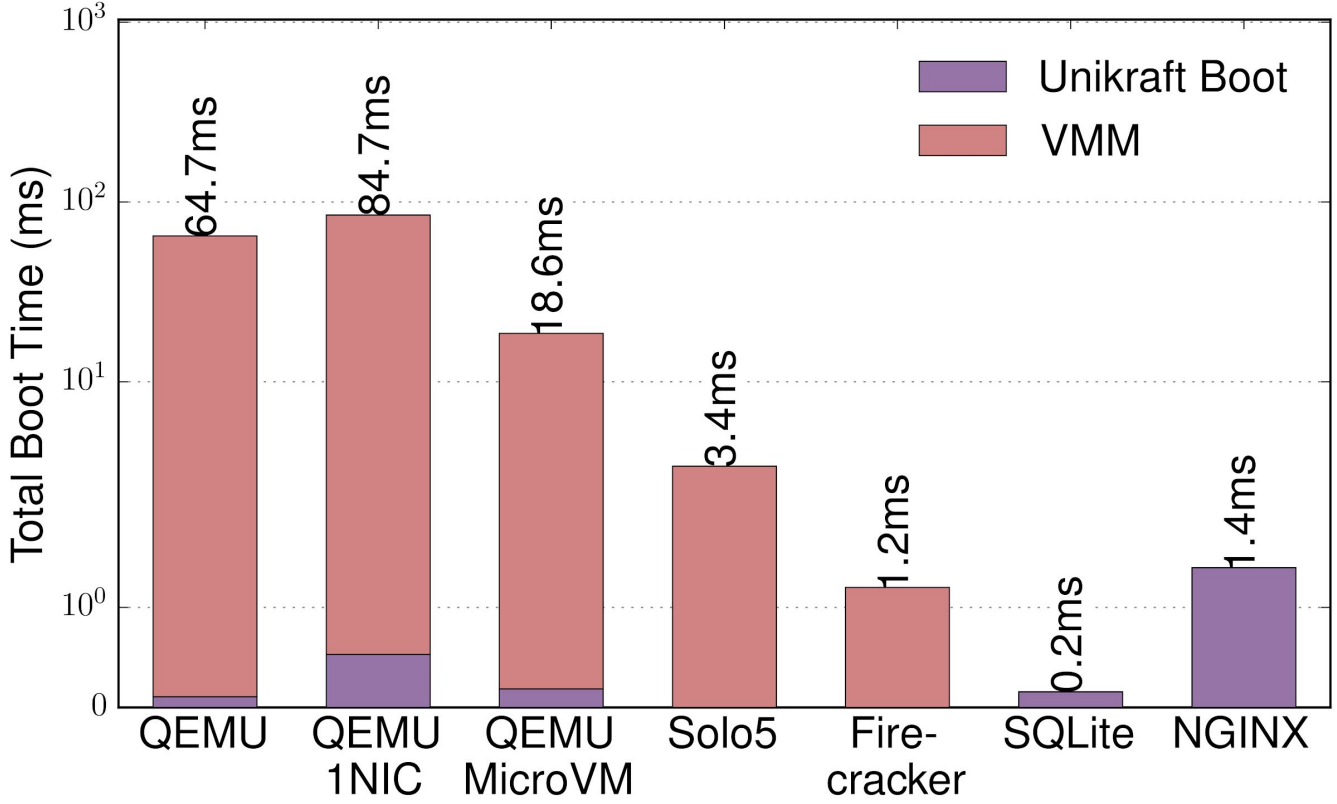

Base Performance Evaluation



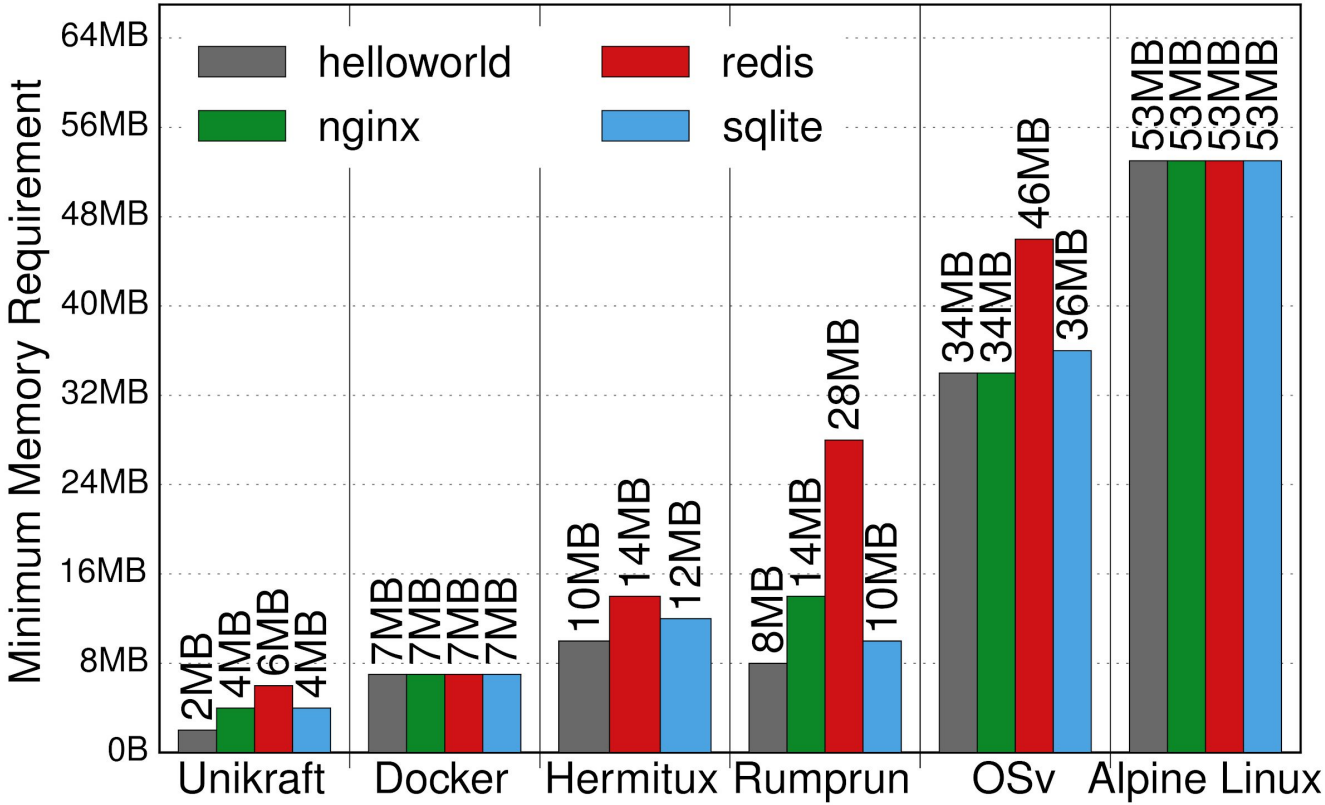
Unikernel image size compared to other projects



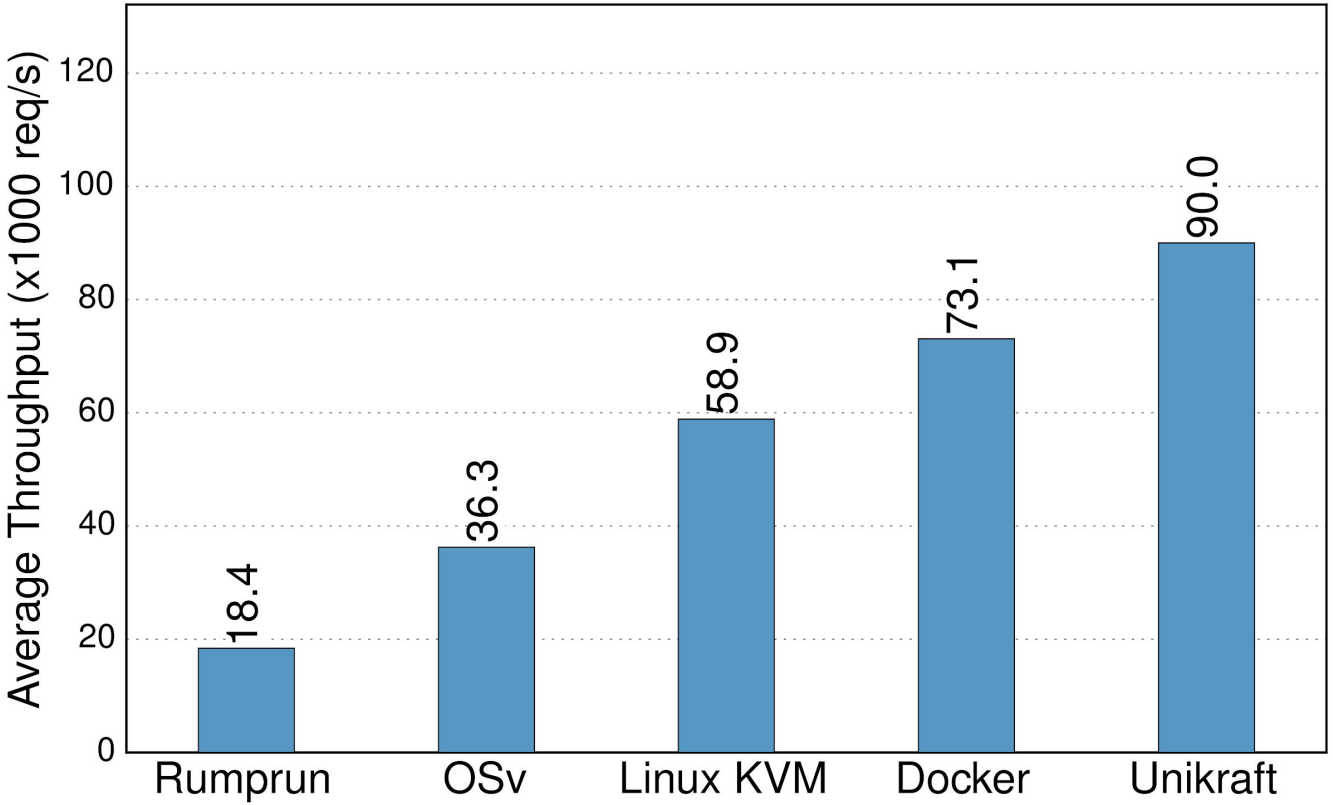
Boot time of Unikraft with different VMMs



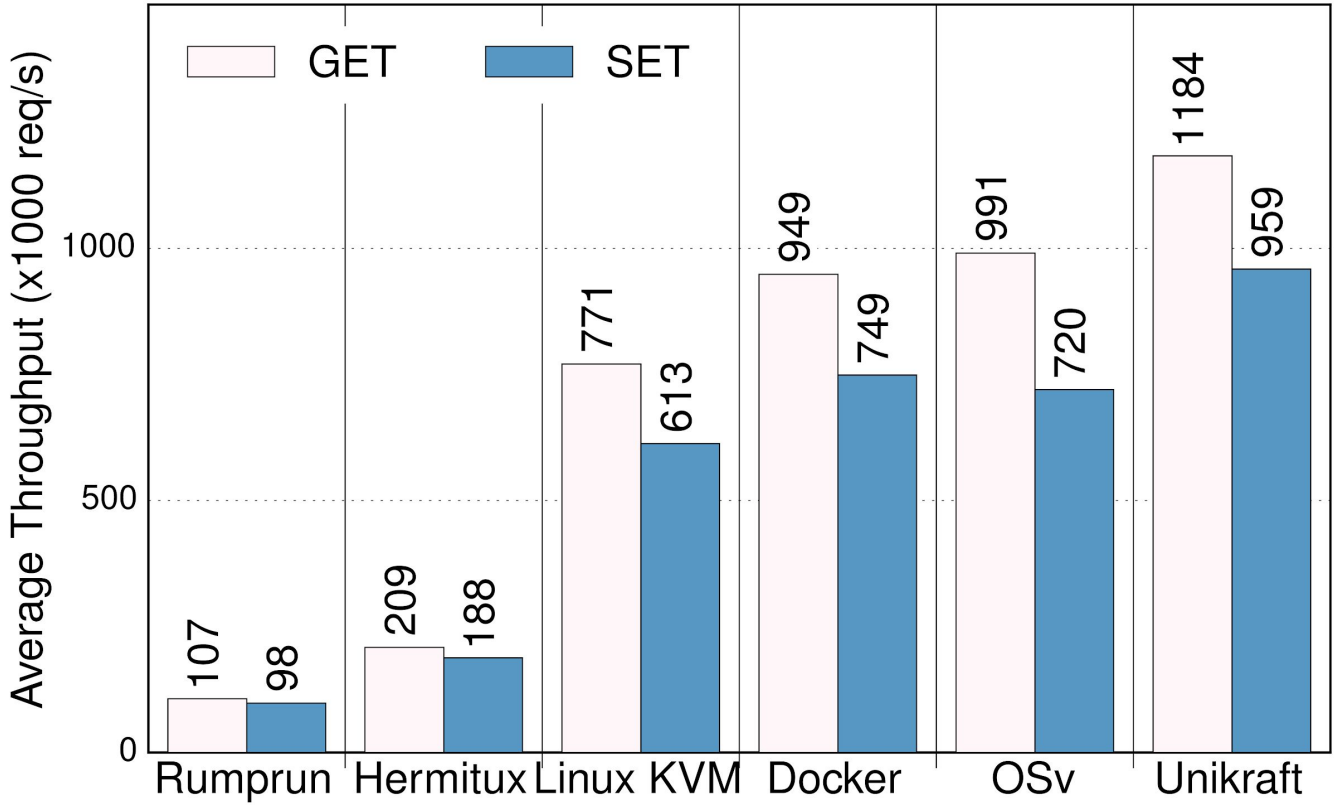
Minimum memory needed to run an application



NGINX performance with wrk

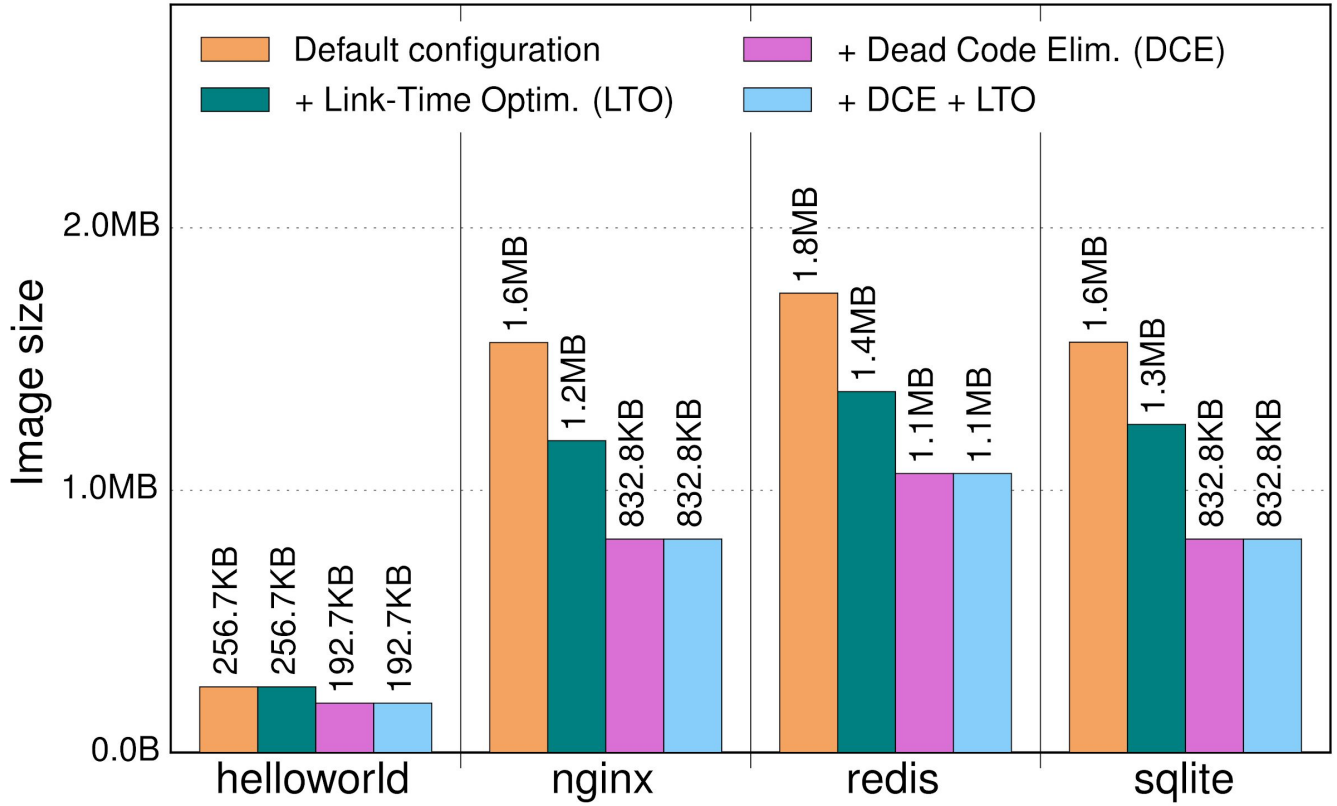


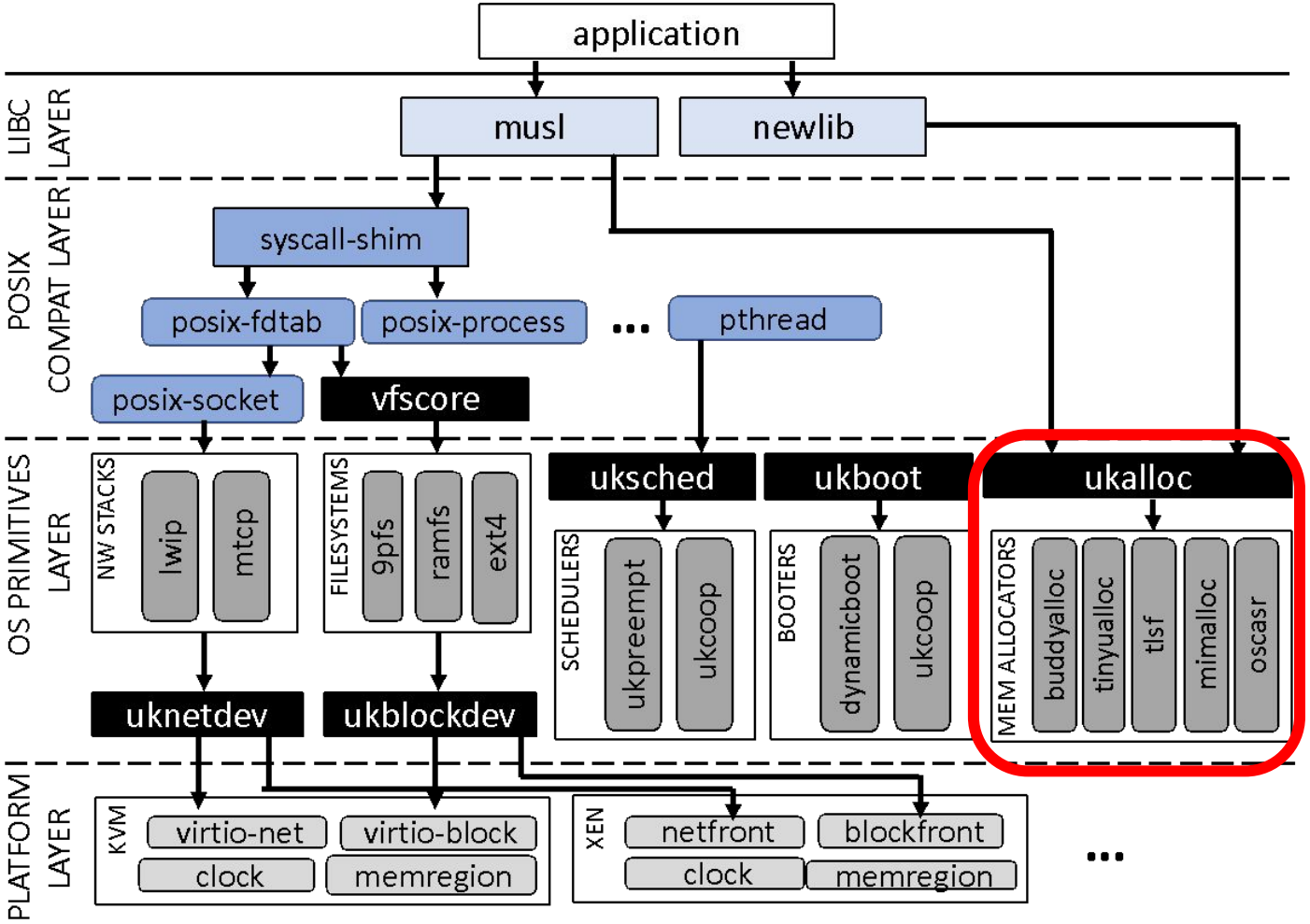
Redis performance tested with `redis-benchmark`



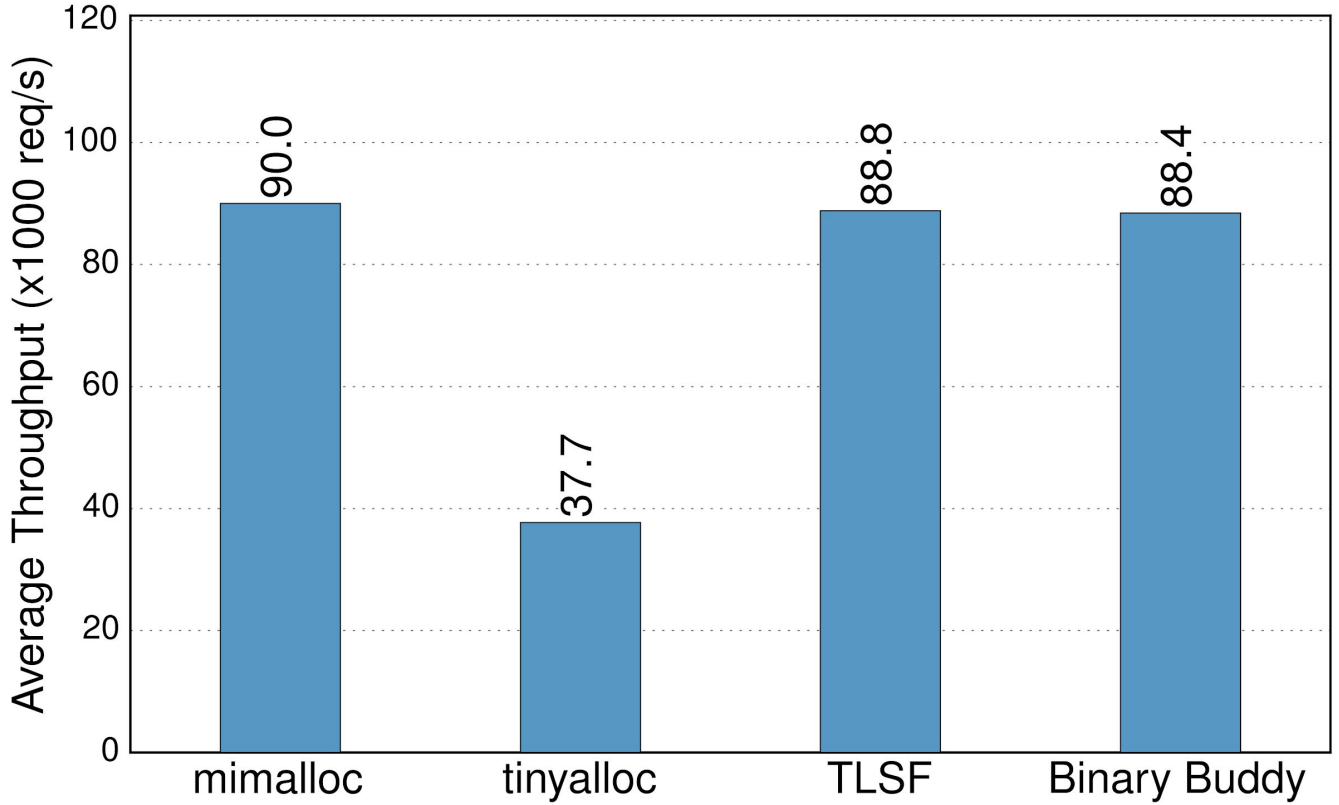
Specialization Performance Evaluation

Unikraft image sizes

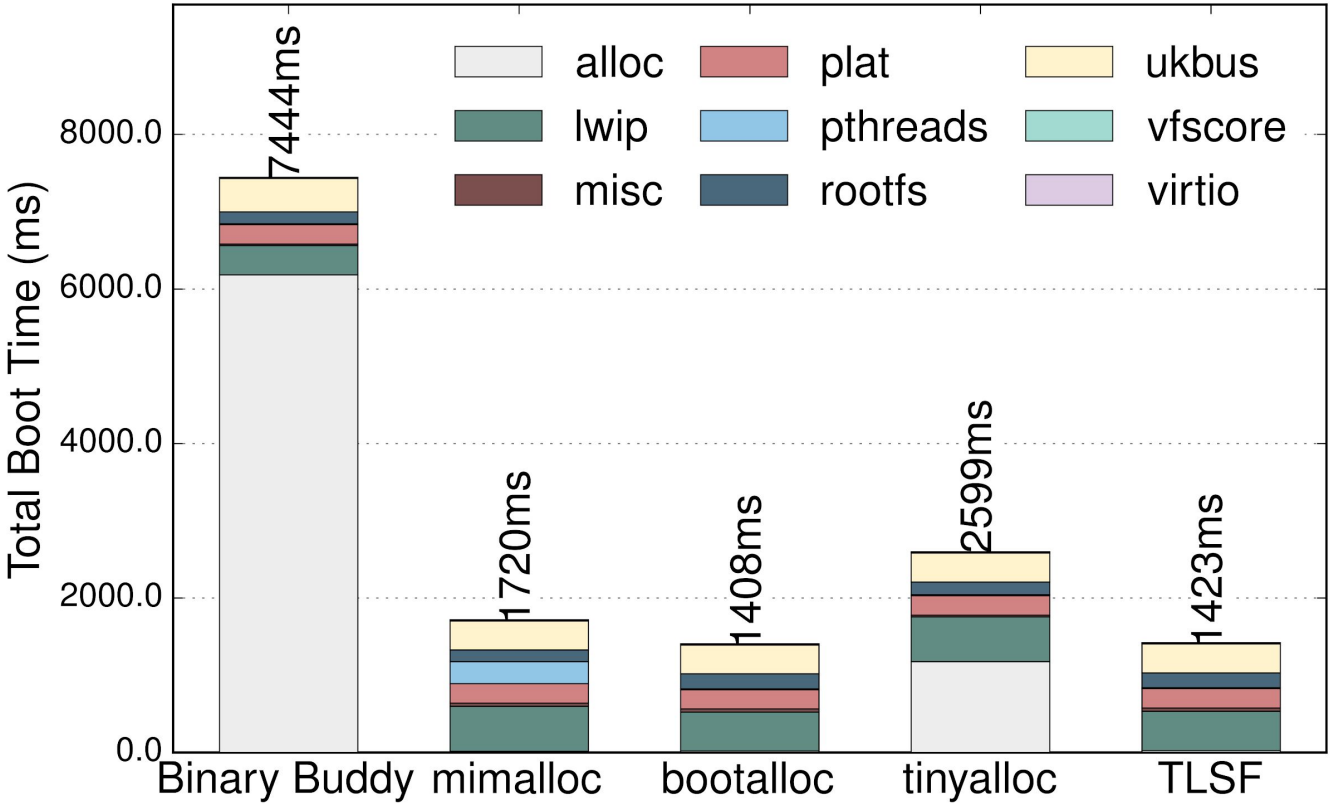




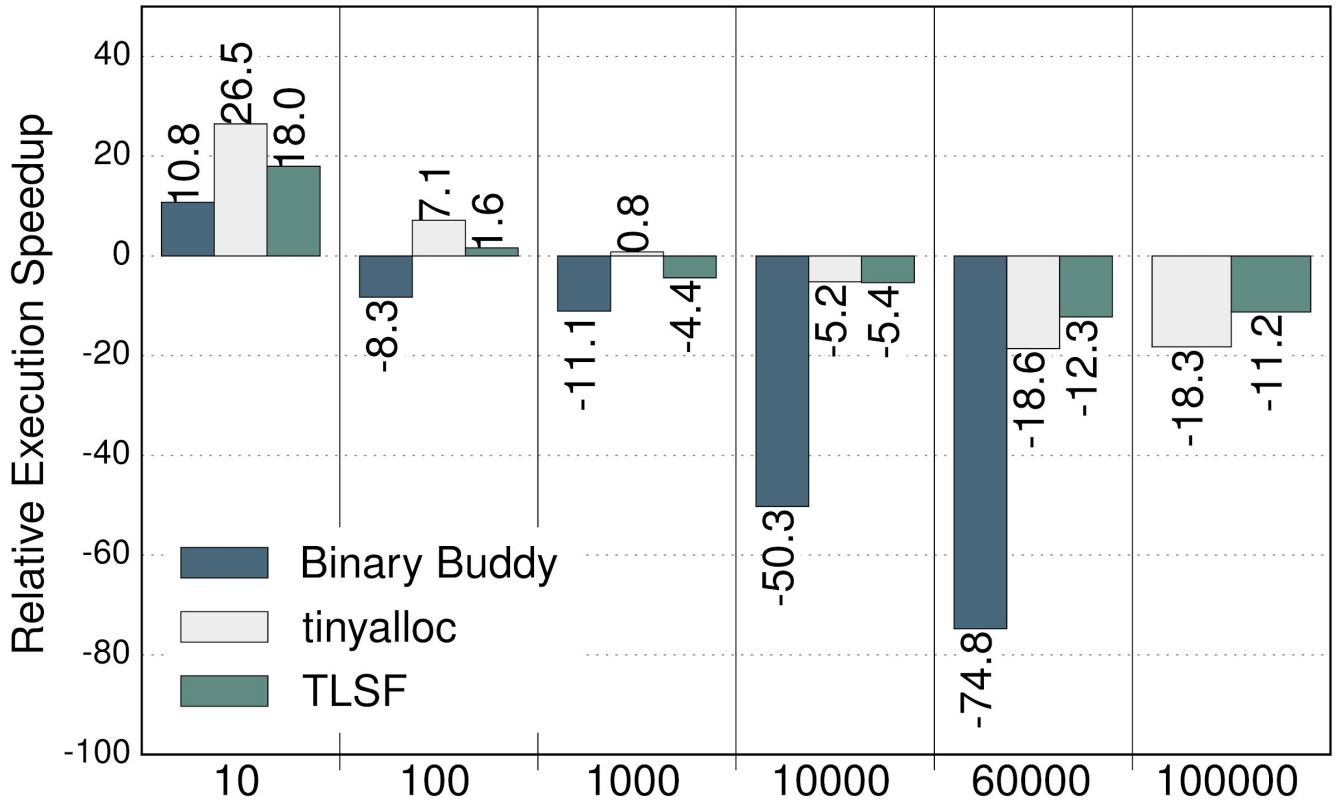
Unikraft NGINX throughput \w diff mem allocators



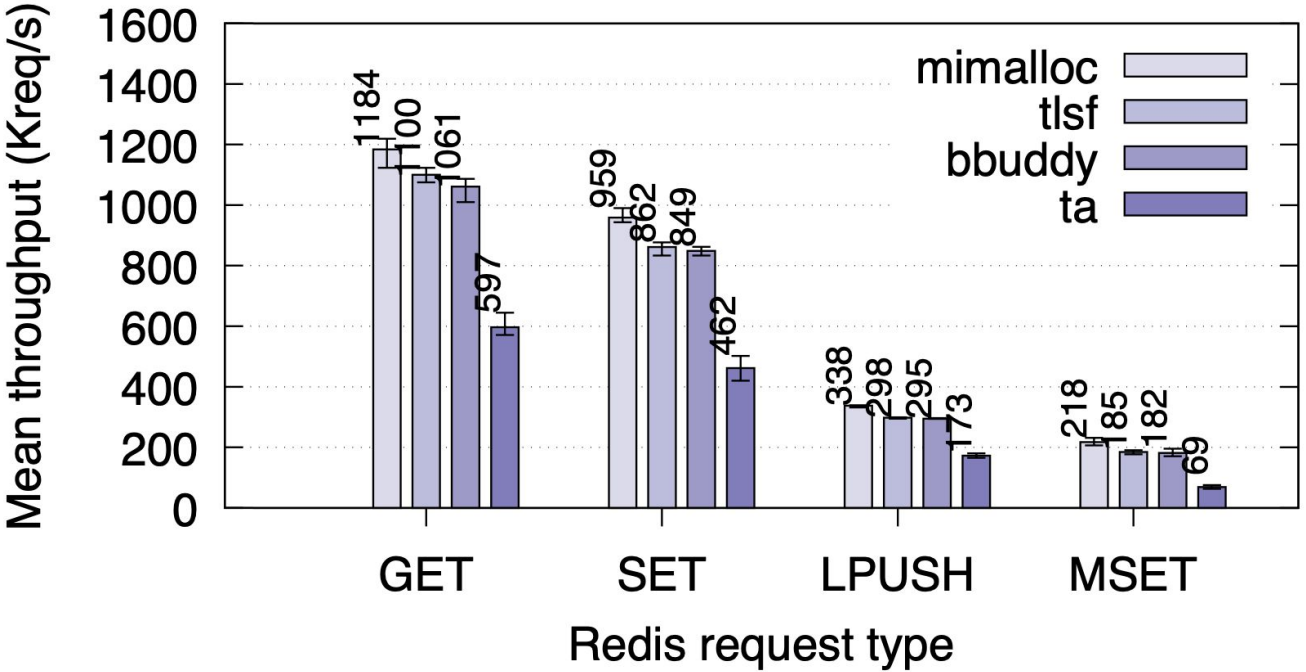
Unikraft NGINX boot time \w diff mem allocators

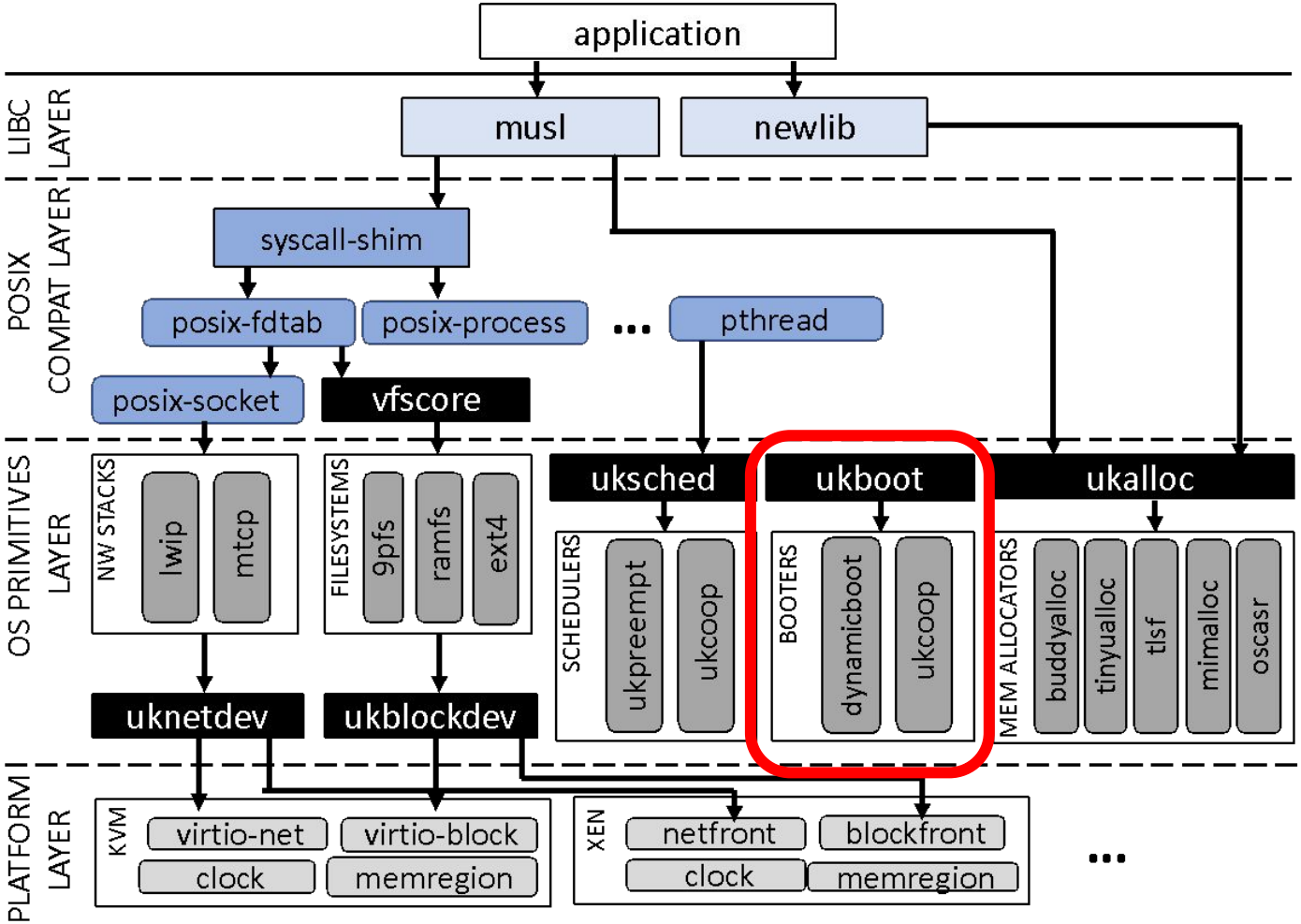


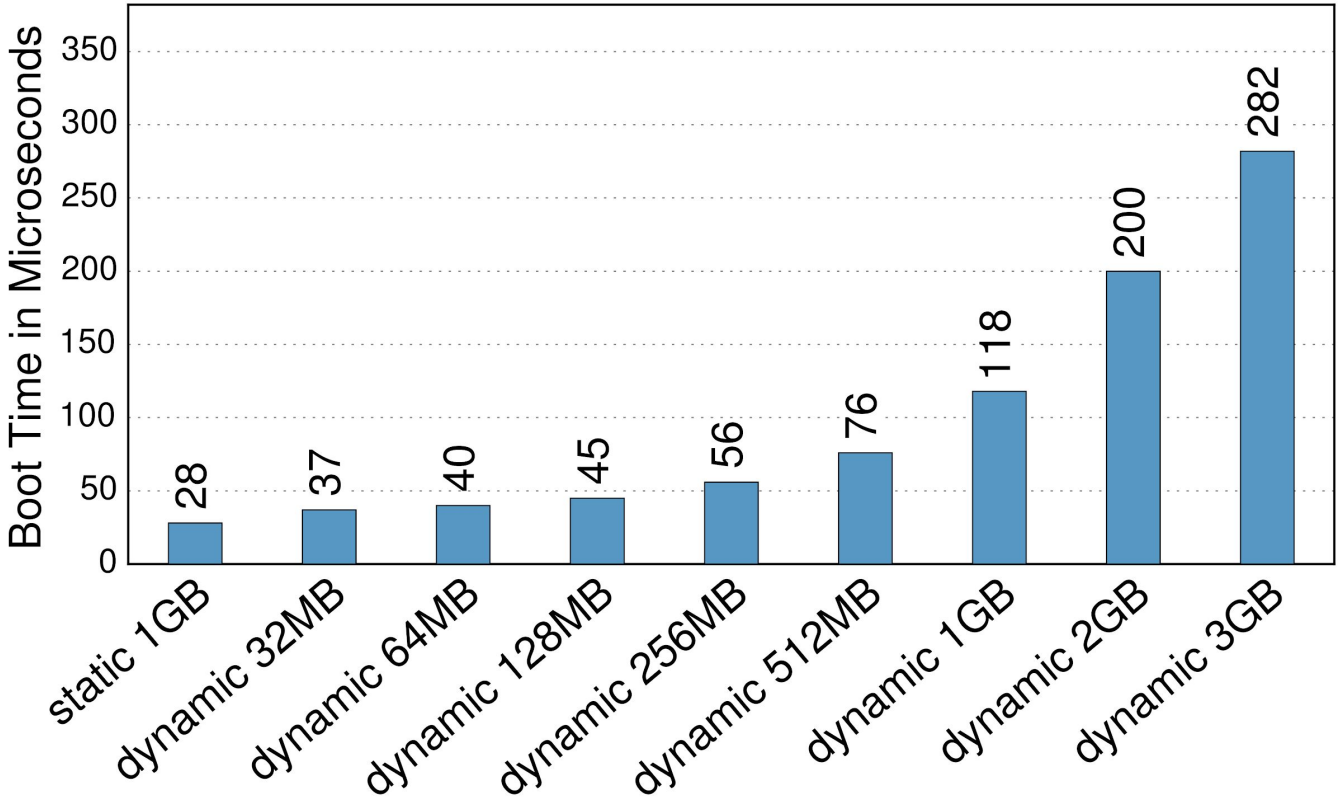
Execution speedup of SQLite relative to mimalloc

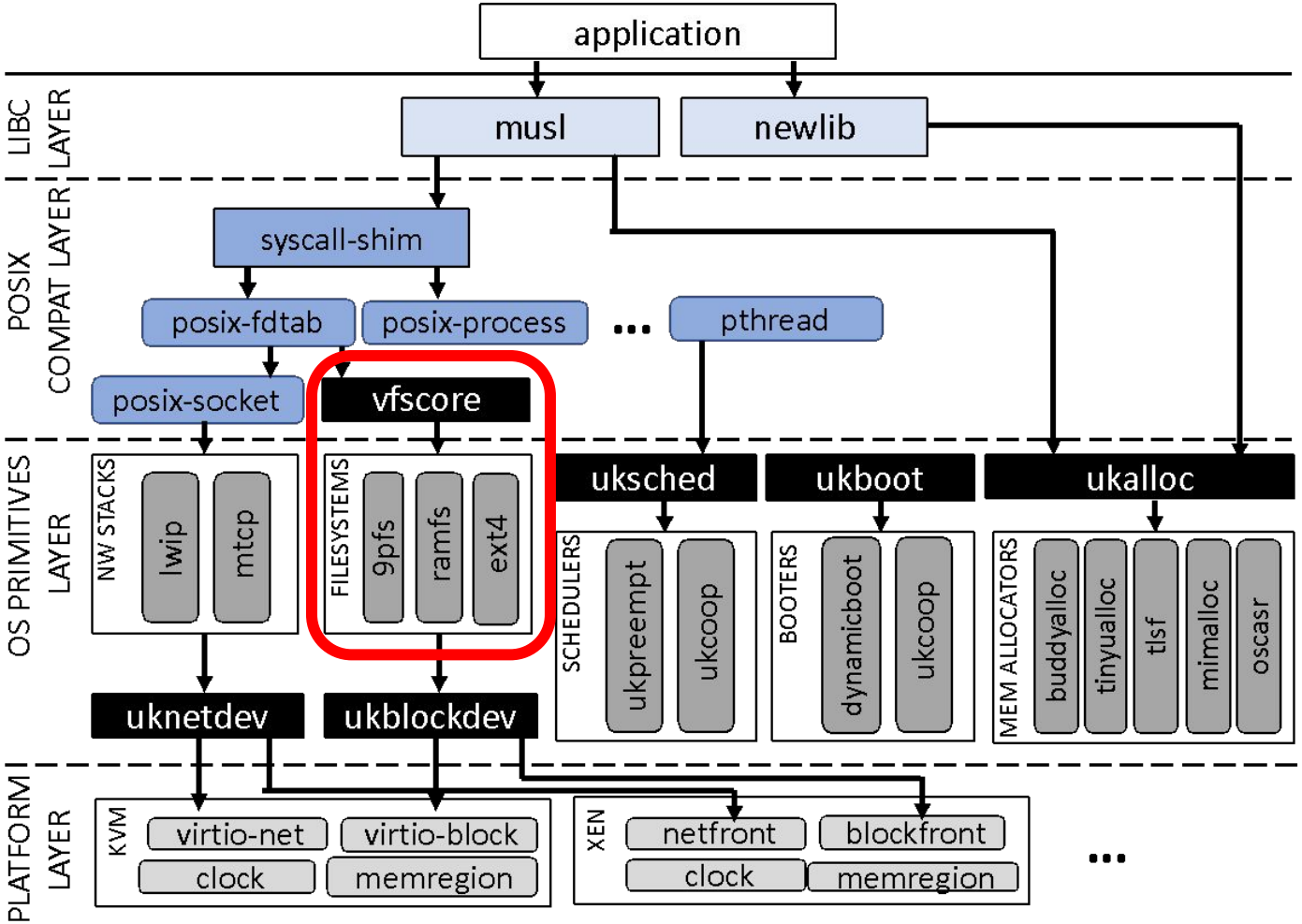


Throughput with Redis using `redis-benchmark`

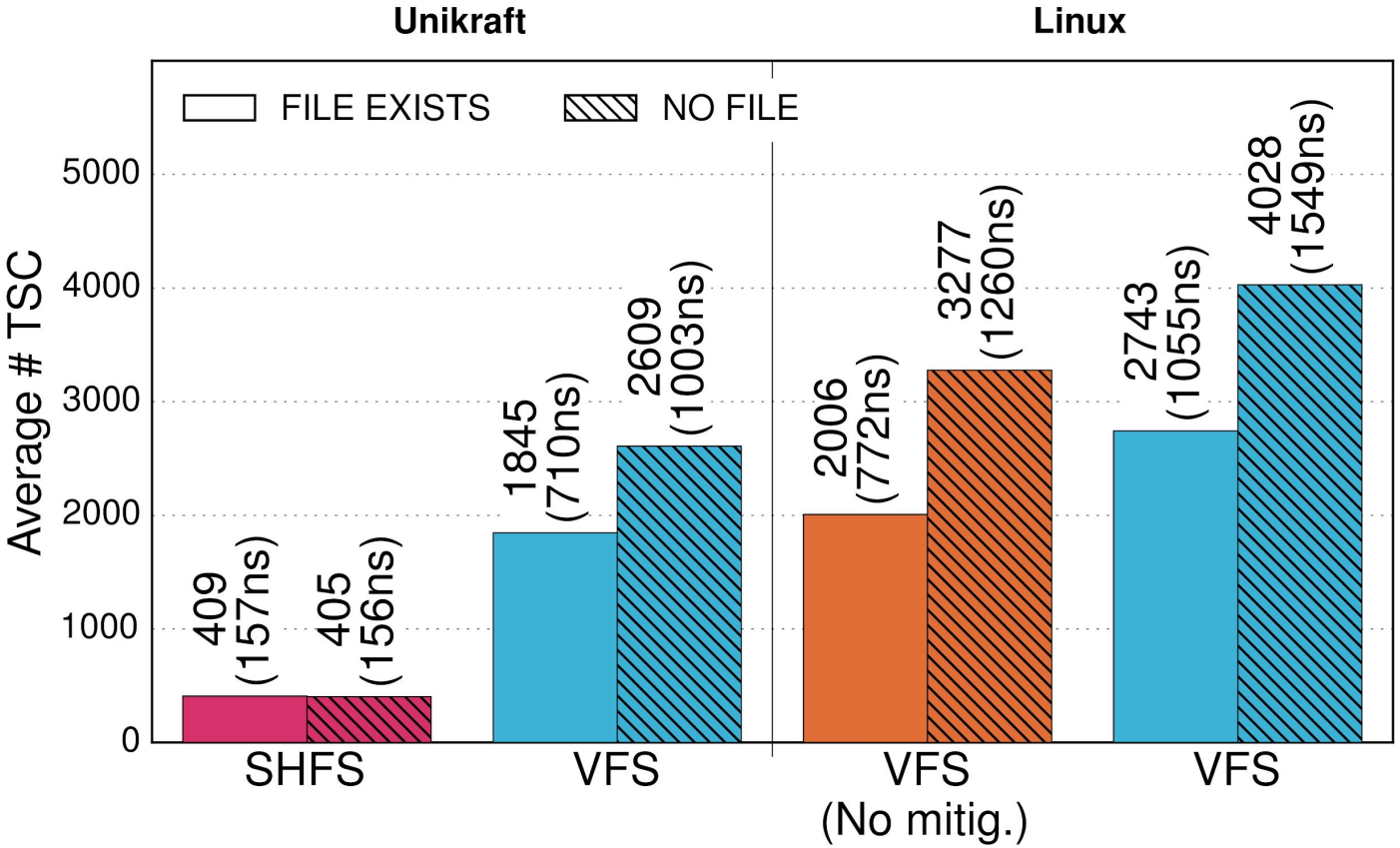


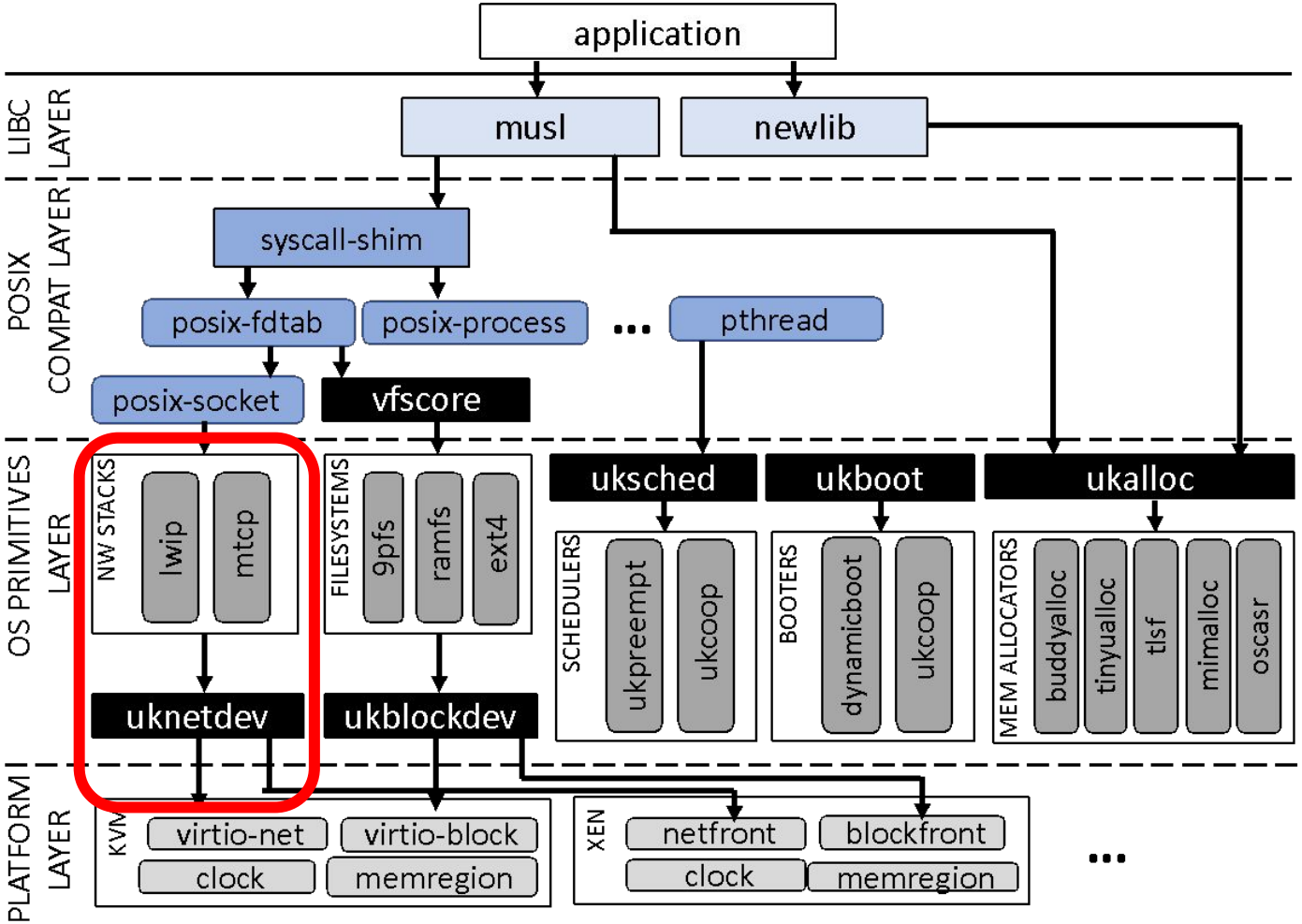






Filesystem Specialization for web caching

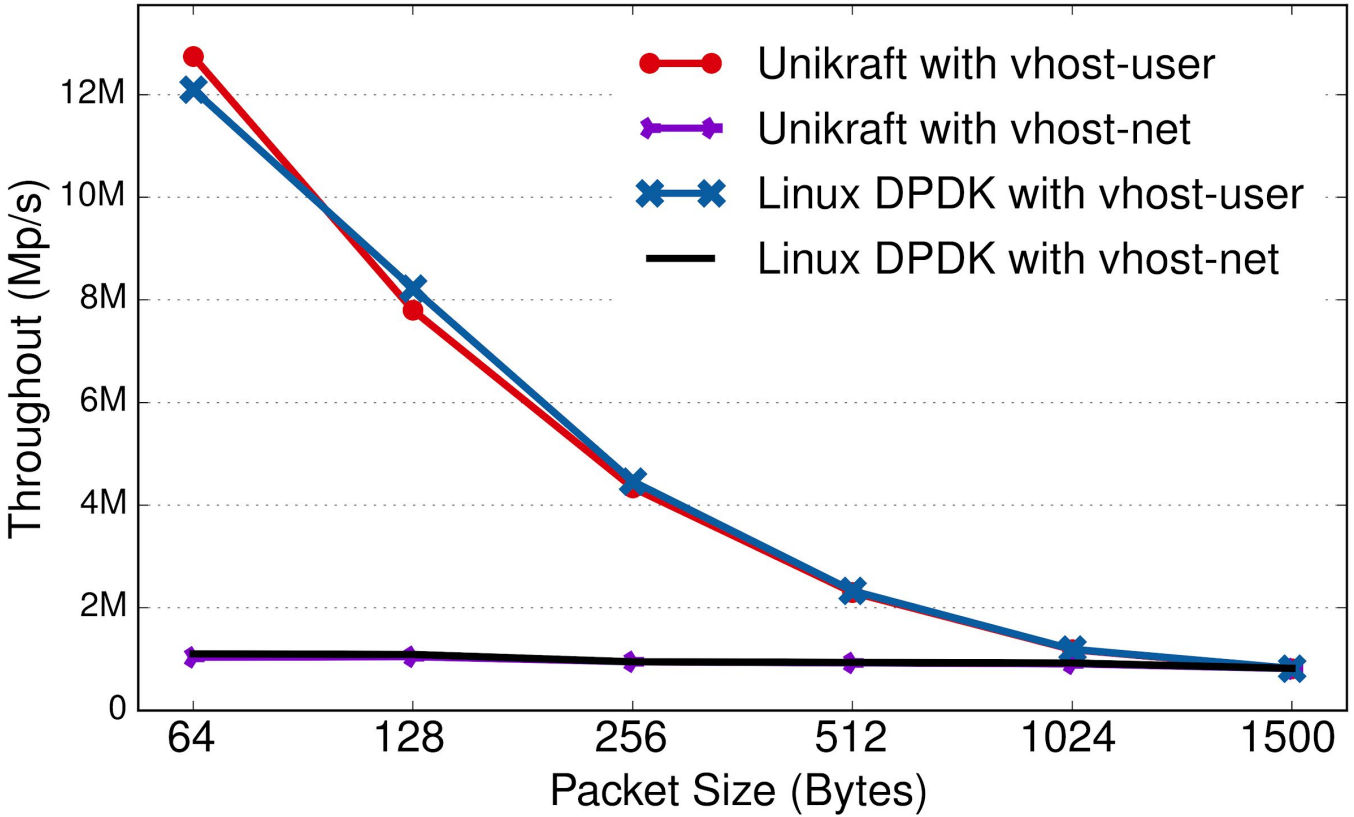




Key-value store application specialization

Linux Baremetal		**Linux KVM Guest**			**Unikraft KVM Guest**		
Single	*Batch*	*Single*	*Batch*	*DPDK*	*LwIP*	*Netdev*	*DPDK*
509K/s	985K/s	105K/s	276K/s	6.1M/s	250K/s	6M/s	6.1M/s

TX throughput of Unikraft vs. Linux KVM VM



Future directions on Specialization

- Compartmentalization
  1. write critical micro-libs in memory safe, race condition safe or statically verifiable languages
  2. compile and link them together
  3. use HW assisted memory separation (CHERI, Intel MPKs, etc.) to retain languages' properties
- Code reduction
- Sealing (hypervisor call to set pages as read-only or execute-only after boot)
- Upstream standard features (ASLR, stack protection etc.)
- Fuzzing (for verification of above)

Find us online

https://github.com/unikraft

http://unikraft.org

<minios-devel@lists.xenproject.org>
<unikraft@listserv.neclab.eu>

@UnikraftSDK

Thanks