Technische Universität Braunschweig

SPP2037
SCALABLE DATA MANAGEMENT FOR FUTURE HARDWARE

Institute of Operating Systems and Computer Networks

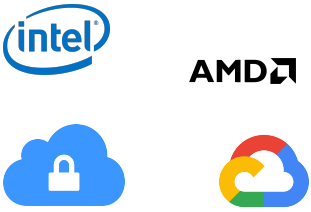
Continuous Code Re-Randomisation at Runtime for Intel SGX Enclaves

Herbsttreffen 2020 Fachgruppe Betriebssysteme

Mohammad Mahhouk, Rüdiger Kapitza, September 24, 2020

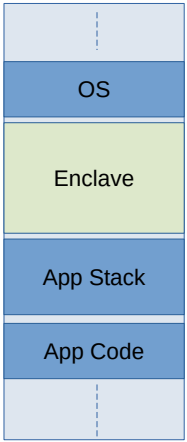
The Rise of Trusted Execution Environments

- Rapid growth of cloud computing services
- Need to protect sensitive data in untrusted environments
- Hence, hardware supported solutions:
  - Intel SGX
  - AMD SEV-VMs
- Commercial secure clouds:
  - Microsoft Azure (Intel SGX)
  - Google confidential VMs (AMD SEV)



Intel SGX in a Nutshell

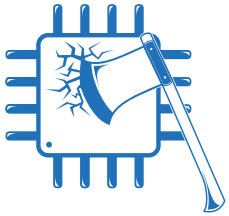
- Hardware supported security feature
- Trusted execution environments called *Enclaves*
- Enclaves characteristics:
  - Isolated and limited memory regions
  - Encrypted by a dedicated hardware unit
  - Contains the sensitive data and critical code
- SGX software development kit (SDK)
  - Eases the work with the enclaves
- SGXv2 lifts up some design restrictions:
  - Enclave Dynamic Memory (EDM)
  - Modifying permissions of enclave's pages at runtime



Attacks on SGX

Despite SGX's strong threat model, it is prone to:

- Return-oriented programming attacks, for example:
  - Dark-ROP [Lee et al.]
  - Just-in-time code reuse [Snow et al.]
- Controlled-channel attacks, such as:
  - Noise-free deterministic side-channel attack [Xu et al.]
  - Stealthy page table-based attacks [Van Bulck et al.]
- Micro-architectural side-channel attacks:
  - Cache attacks
  - Branch shadowing attacks



Source: SGAxe [van Schaik et al.]

Motivation & Our Goal

- Harden the security against ROP and some controlled-channel attacks
- Less enclave limitations through the novel SGXv2 instructions
- Extendable dynamic linker and hot patching framework with SGXv2 support

Continuous Code Re-Randomisation at Runtime for Intel SGX Enclaves

- Mitigating state-of-the-art ROP and controlled channel attacks
- No necessary custom compiler or hardware modification

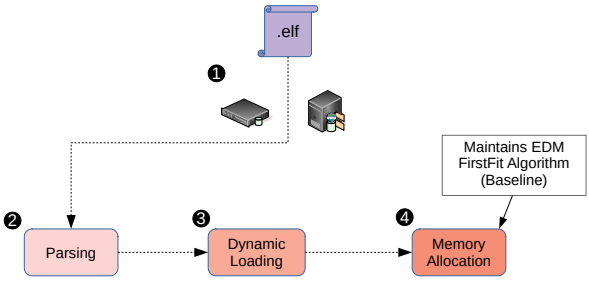
Contents

SGX Dynamic Loader Framework

- Dynamic linker and hot patching framework, sgx-dl, utilising the new instructions. It allows/has:
  - base enclave for elf files parsing
  - adding, un-loading, updating and executing of functions dynamically
  - extensible stand-alone memory management library for the EDM
  - practical performance overhead of $< 1\%$
- Part of the STAN project SPP 2037
- First to use the novel SGXv2 features

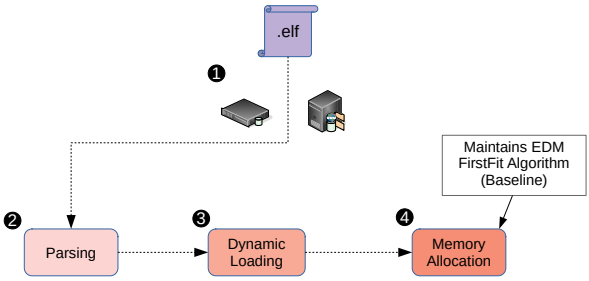
SGX-DL Workflow

1. Compiled source code
2. In-enclave elf parser
3. Dynamic loader
4. Enclave dynamic memory management



SGX-DL Workflow

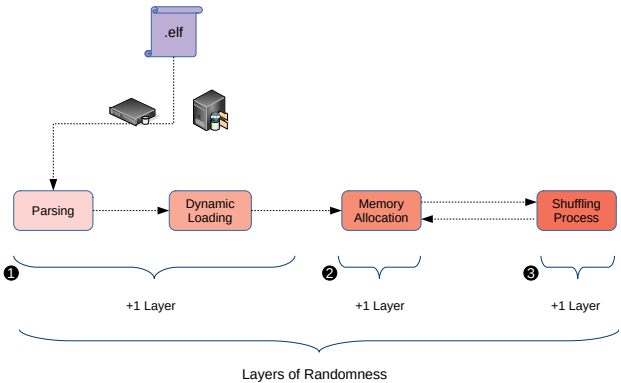
1. Compiled source code
2. In-enclave elf parser
3. Dynamic loader
4. Enclave dynamic memory management



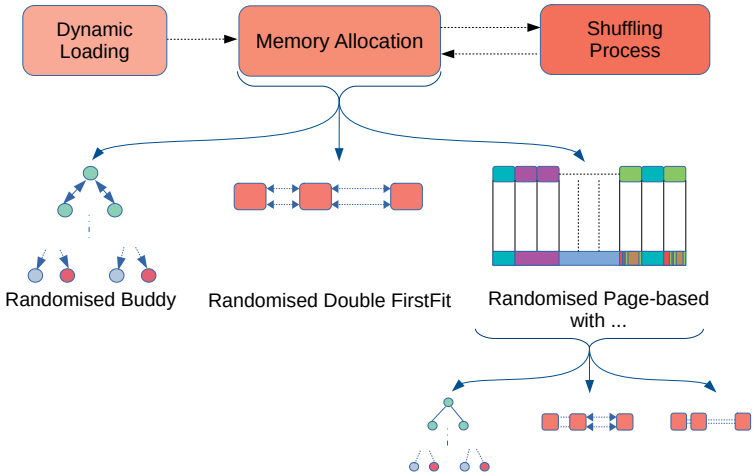
Deterministic process $\Longrightarrow$ Identical address space layout

ASLR Workflow

1. Random loading order
2. Randomised memory allocators
3. Periodic in-enclave shuffling process



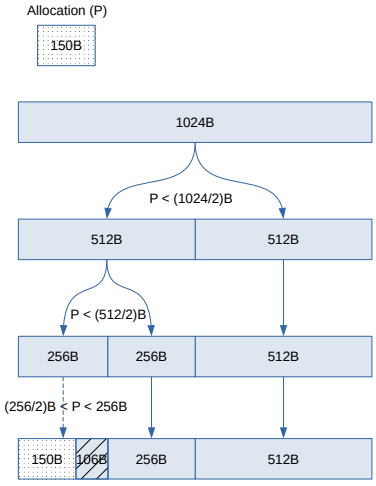
Memory Allocation Mechanisms



Randomised Buddy

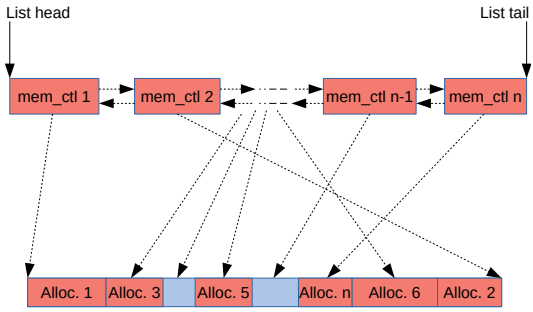
Similar to the standard buddy algorithm, but ..

- randomised allocation path
- Memory reservations rounded up to the 2^n



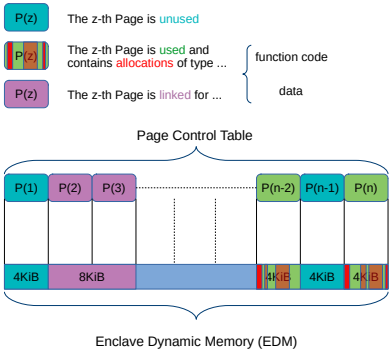
Randomised Double FirstFit

- Randomised entry point.
  - The i-th malloc has $\mathcal{O}(2^i)$ possible locations
- Memory reservation is not limited to the 2^n

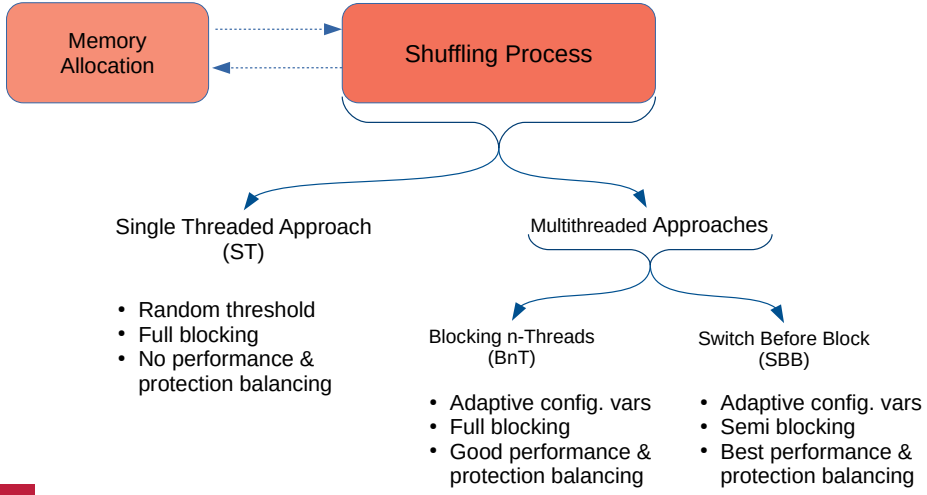


Randomised Page-based

- EDM is chunked in uniform sized pages
- Page choice is randomised
- 3-page states: Used, unused and linked
- Bigger than page size allocations possible



Shuffling Approaches



Existing Defence Mechanisms

Currently only few works provide protection such attacks, for example:

- ASLR implementations for Intel SGX against ROP attacks:
  - SGX-Shield [Seo et al.]
  - SGX-Armor [Shih Dissertation]
- Using hardware features like Intel TSX in T-SGX [Shih et al.]
- Data randomisation and cryptographic obfuscation of memory like in DR.SGX [Brasser et al.]

However, these require either:

- special compilers
- hardware modifications
- or/and induce significant performance overhead

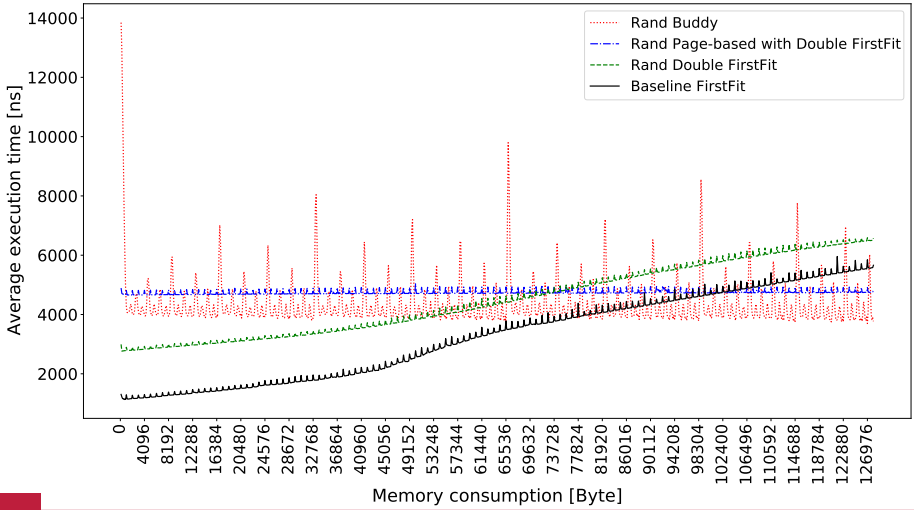
Performance Evaluation Metrics

- 1000 allocations/frees repeated 1000 times
- Fixed allocation size of 128 bytes
- Gathered the duration of each allocation
- Average of the 95th percentile
- 95th confidence-interval

Used hardware specs:

- Intel(R) Celeron(R) J4005 CPU @ 2.00GHz
- 8 GB DDR4 RAM @ 2400Mhz
- SGXv2 supported

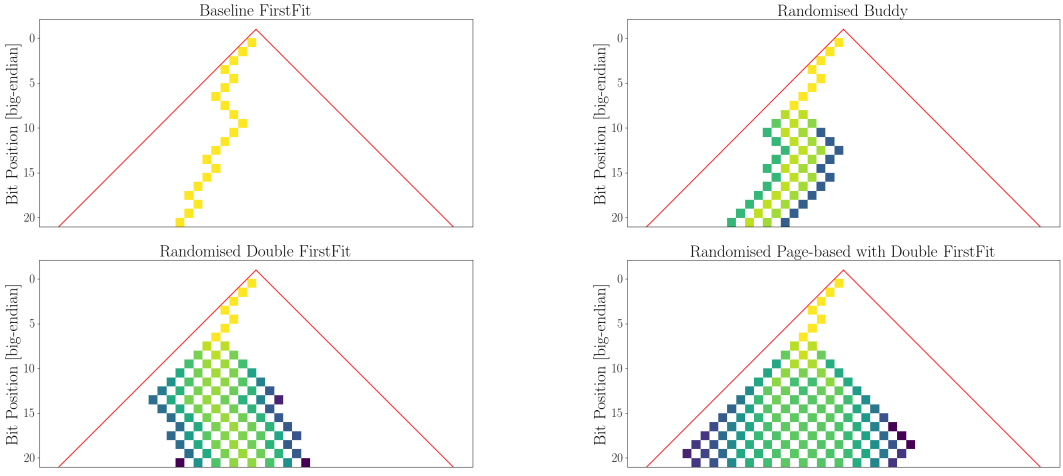
Malloc Evaluation of All Memory Allocators



Entropy Evaluation Metrics

- 1000 allocations repeated 1000 times
- Fixed allocation size of 128 bytes
- Gathered the returned addresses in binary form
- Only relevant bits of the shuffling process, i.e. the first 21 bits of 1MB shuffled EDM

Heat Maps of the 1000th Allocation of All Memory Allocators



Evaluation Metrics for the Periodic Shuffling Approaches

- Two stress micro-benchmarks: inclusive & exclusive benchmark
  - Repeated 100 times for each shuffling approach
  - Average and 95% confidence interval

Exclusive:

- Two threads: worker & measurement
- $\approx$ One minute execution runtime
- Enclave transition times are not included

Exclusive Benchmark Results

Approach Name	Ttl. Nr. of Executions [Million]	Executions per ms	Ttl. Nr. of Shuffles	Shuffles per ms	Normalised Performance
Baseline	547.48 ± 1.77	9124.58 ± 29.51	0	0	1×
Single **T**hreaded	260 ± 1.15	4333.29 ± 19.12	519968.04 ± 2298.55	8.67 ± 0.04	≈0.48×
Blocking **n**-**T**hreads	285.46 ± 3.12	4757.67 ± 51.91	369627.10 ± 10032.67	6.16 ± 0.17	≈0.52×
Switch **B**efore **B**lock	374.92 ± 1.15	6248.71 ± 19.20	470665.58 ± 7143.51	7.84 ± 0.12	≈0.69×

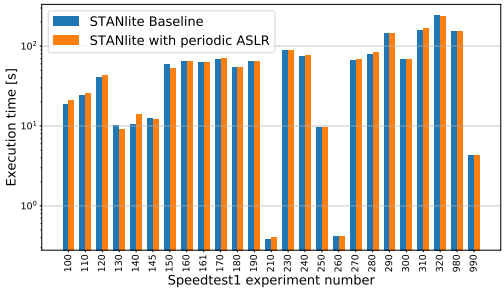
Exclusive Benchmark Results

Approach Name	Ttl. Nr. of Executions [Million]	Executions per ms	Ttl. Nr. of Shuffles	Shuffles per ms	Normalised Performance
Baseline	547.48 ± 1.77	9124.58 ± 29.51	0	0	1×
Single **T**hreaded	260 ± 1.15	4333.29 ± 19.12	519968.04 ± 2298.55	8.67 ± 0.04	≈0.48×
Blocking **n**-**T**hreads	285.46 ± 3.12	4757.67 ± 51.91	369627.10 ± 10032.67	6.16 ± 0.17	≈0.52×
Switch **B**efore **B**lock	374.92 ± 1.15	6248.71 ± 19.20	470665.58 ± 7143.51	7.84 ± 0.12	≈0.69×

SBB is the most balanced but ST is less complex and strictly random

STANlite Speedtest1

- Repeated 5 times w & w/o periodic shuffling
- Stable results with insignificant std. dev.
- Avg. perf. overhead $\approx$ 2.3% per test experiment



Short Summary

- *Problem:* Novel JIT-ROP and controlled channel attacks
- *Motivation:* Unsatisfying existing defensive mechanisms and approaches
- *Our work:* Periodic ASLR for Intel SGX enclaves
  - Randomised memory management mechanisms
  - Various runtime shuffling concepts
  - Complete execution path randomisation
- With periodic ASLR we achieved the following:
  - Better protection against ROP and controlled-channel attacks
  - No need for special compiler or hardware modification
  - Incurs practical performance overhead in real application workloads $\approx 2.3\%$

Evaluation Metrics for the Periodic Shuffling Approaches

- Two stress micro-benchmarks: inclusive & exclusive benchmark
  - Repeated 100 times for each shuffling approach
  - Average and 95% confidence interval

Inclusive:

- Single thread for measurement & execution
- One million consecutive calls
- Enclave transition times included

Exclusive:

- Two threads: worker & measurement
- ≈ One minute execution runtime
- Enclave transition times are not included

Inclusive Benchmark

Approach Name	Avg. Execution Time per Call [ns]	Ttl. Nr. of Shuffles	Shuffles per ms	Normalised Performance
Baseline	14937.98 ± 14.26	0	0	1×
Single **T**hreaded	15185.18 ± 18.20	1991.75 ± 4.36	0.13 ± 0.00	≈ 0.98×
Blocking **n**-**T**hreads	18017.75 ± 115.88	52389.69 ± 118.98	2.91 ± 0.02	≈ 0.83×
Switch **B**efore **B**lock	15614.59 ± 17.40	47614.85 ± 30.83	3.05 ± 0.00	≈ 0.96×

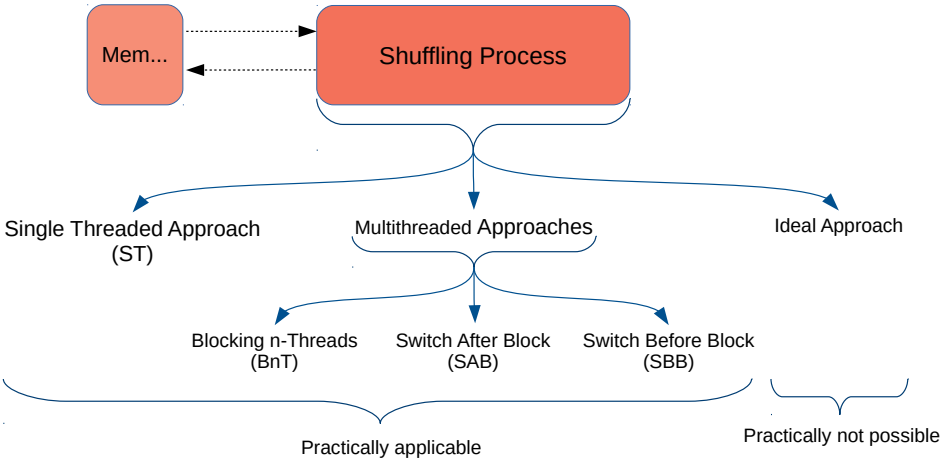
Exclusive Benchmark

Approach Name	Ttl. Nr. of Executions [Million]	Executions per ms	Ttl. Nr. of Shuffles	Shuffles per ms	Normalised Performance
Baseline	547.48 ± 1.77	9124.58 ± 29.51	0	0	1×
ST	260 ± 1.15	4333.29 ± 19.12	519968.04 ± 2298.55	8.67 ± 0.04	≈0.48×
BnT	285.46 ± 3.12	4757.67 ± 51.91	369627.10 ± 10032.67	6.16 ± 0.17	≈0.52×
SBB	374.92 ± 1.15	6248.71 ± 19.20	470665.58 ± 7143.51	7.84 ± 0.12	≈0.69×

Function Size Analysis

Library Name	Average (Byte)	Percentile (Byte)			
		99-th	95-th	90-th	80-th
STANlite	360	4140	1266	671	368
TaLoS	180	2122	779	440	212
LibSEAL	180	2079	779	443	215

Shuffling Approaches



Shuffling Approaches

Single Threaded:

- Random threshold
- Full blocking
- No perf. & prot. balancing
- Less complexity & attack surface

Blocking n-Threads:

- Adaptive threshold
- Full blocking
- Good perf. & prot. balancing
- More complex & attack surface

Switch After Block:

- Adaptive config vars
- Semi blocking
- Better Perf. & prot. balancing
- High complexity & same attack surface

Switch Before Block:

- Adaptive config vars
- Semi blocking
- Best Perf. & prot. balancing
- High complexity & same attack surface

Shuffling Approaches

Approach Name	Configuration Vars.	Process Blocking	Perf. & Prot. Balancing	Complexity & Attack Surface
Single **T**hreaded	±	−−	−−	++
Blocking **n**-**T**hreads	++	−	−	±
Switch **A**fter **B**lock	++	+	++	−−
Switch **B**efore **B**lock	++	++	++	−−

Inclusive Benchmark

Approach Name	Avg. Execution Time per Call [ns]	Ttl. Nr. of Shuffles	Shuffles per ms	Normalised Performance
Baseline	14937.98 ± 14.26	0	0	1×
Single **T**hreaded	15185.18 ± 18.20	1991.75 ± 4.36	0.13	≈ 0.98×
Blocking **n**-**T**hreads	18017.75 ± 115.88	52389.69 ± 118.98	2.91 ± 0.02	≈ 0.83×
Switch **A**fter **B**lock	16125.96 ± 386.94	48353.20 ± 722.92	3.01 ± 0.03	≈ 0.93×
Switch **B**efore **B**lock	15614.59 ± 17.40	47614.85 ± 30.83	3.05 ± 0.002	≈ 0.96×

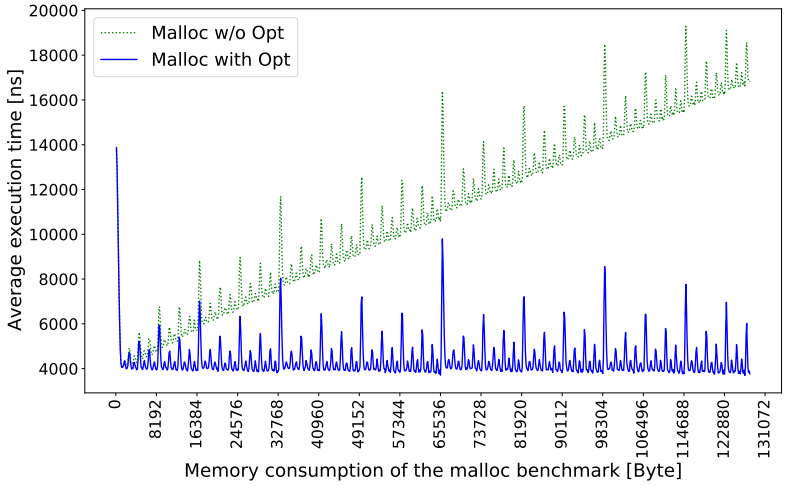
Exclusive Benchmark

Approach Name	Ttl. Nr. of Executions [Million]	Executions per ms	Ttl. Nr. of Shuffles	Shuffles per ms	Normalised Performance
Baseline	547.48 ± 1.77	9124.58 ± 29.51	0	0	1×
ST	260 ± 1.15	4333.29 ± 19.12	519968.04 ± 2298.55	8.67 ± 0.04	≈0.48×
BnT	285.46 ± 3.12	4757.67 ± 51.91	369627.10 ± 10032.67	6.16 ± 0.17	≈0.52×
SAB	375.40 ± 2.27	6256.67 ± 37.90	427891.64 ± 11711.87	7.13 ± 0.20	≈0.69×
SBB	374.92 ± 1.15	6248.71 ± 19.20	470665.58 ± 7143.51	7.84 ± 0.12	≈0.69×

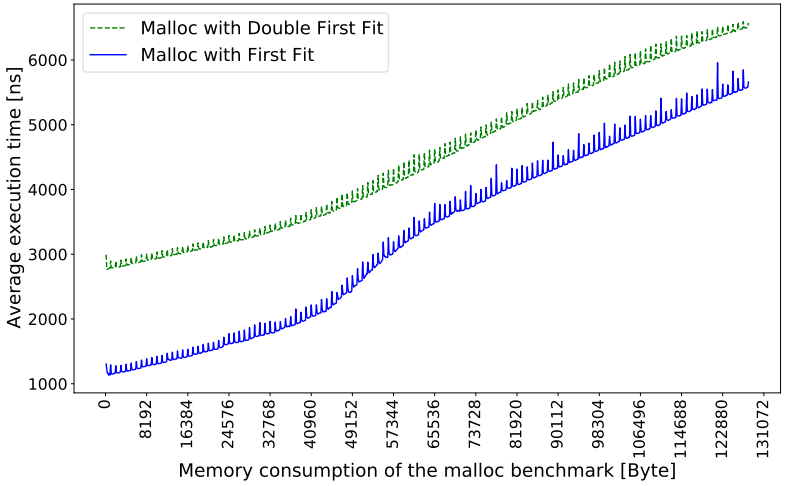
Possible Improvements and Optimisations

- Pre-constructed memory control structures at initialisation time
  - Trade-off space usage $\Longleftrightarrow$ runtime overhead
- Alternative fragmentation handling
- Lock contention $\Longrightarrow$ deadlock at a random point

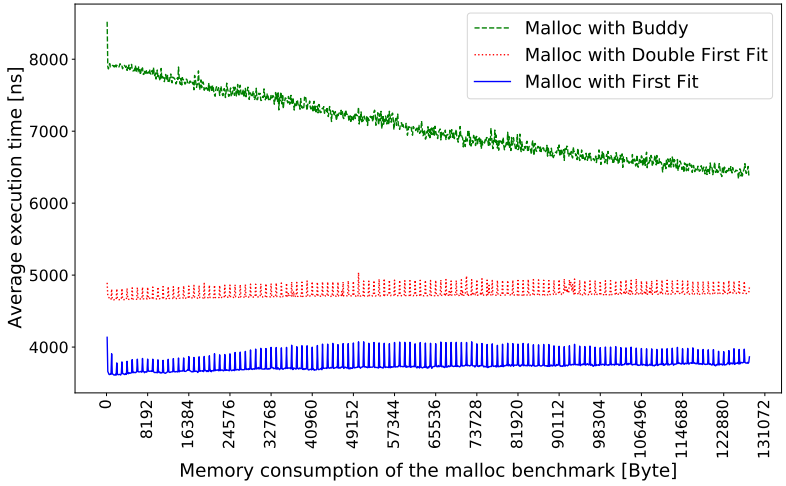
Malloc Evaluation of Randomised Paging Algorithm



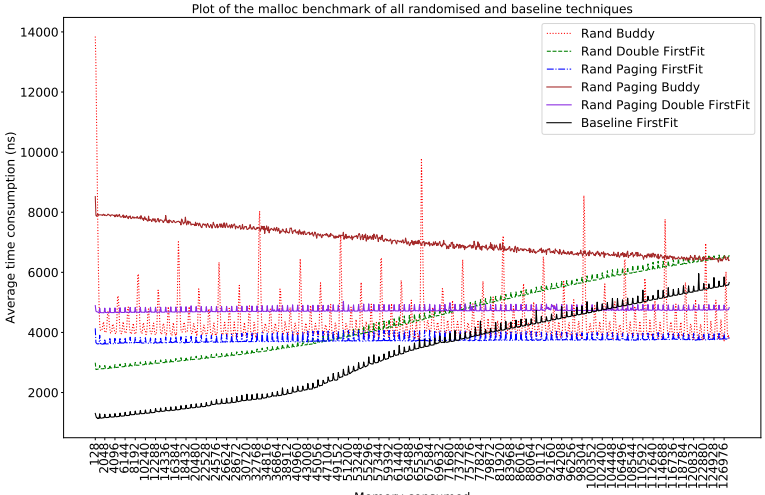
Malloc Evaluation of First Fit & Randomised Double First Fit Algorithm



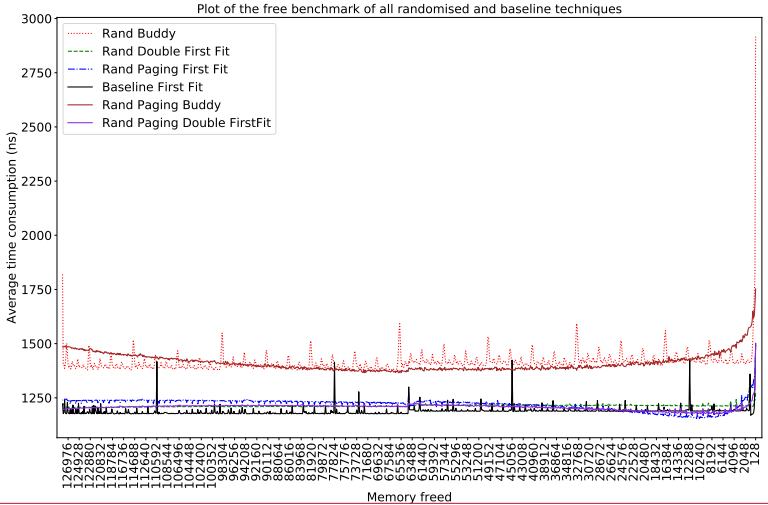
Malloc Evaluation of Randomised Buddy Algorithm



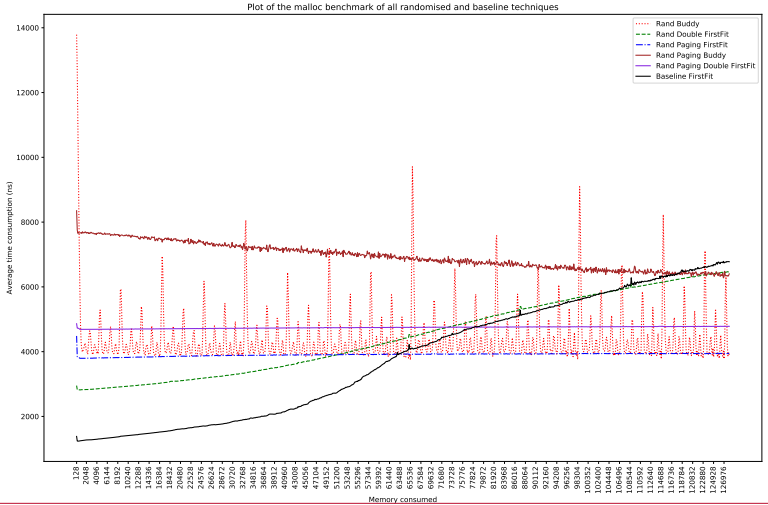
Malloc Evaluation of all Mechanisms



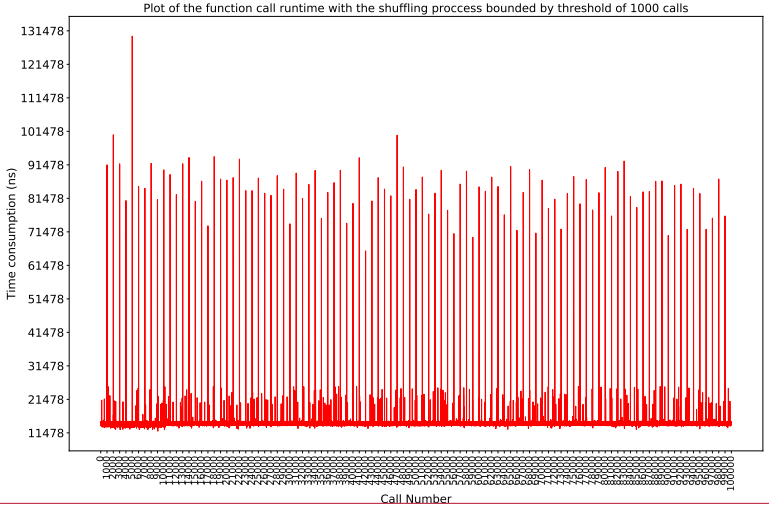
Free Evaluation of all Mechanisms



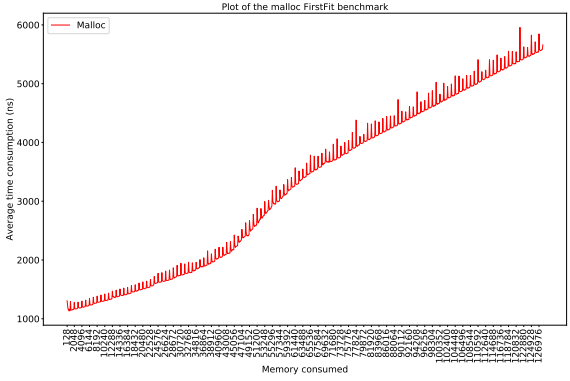
Malloc Evaluation of all Mechanisms using TC-Malloc

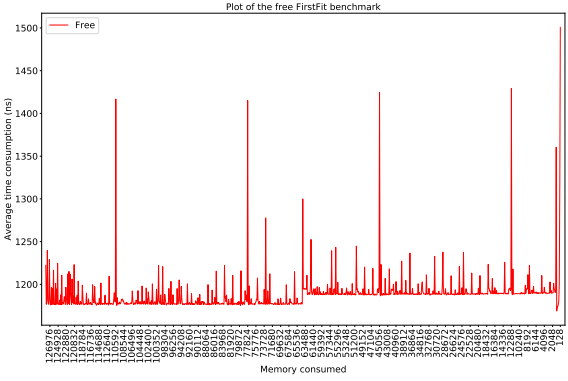


Shuffling Threshold Benchmark

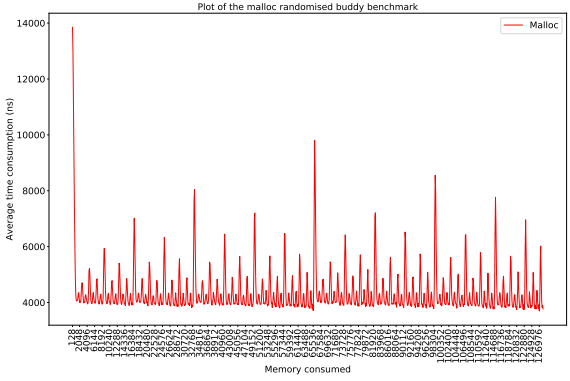


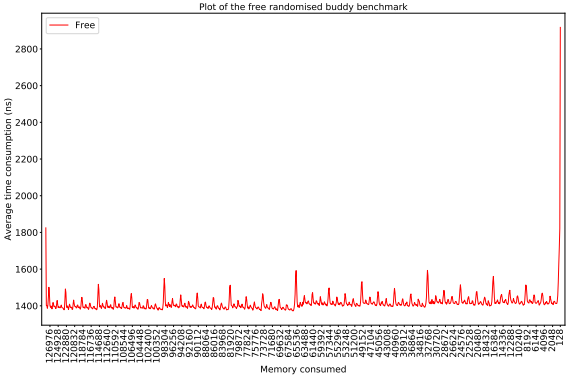
FirstFit (Baseline) Malloc & Free



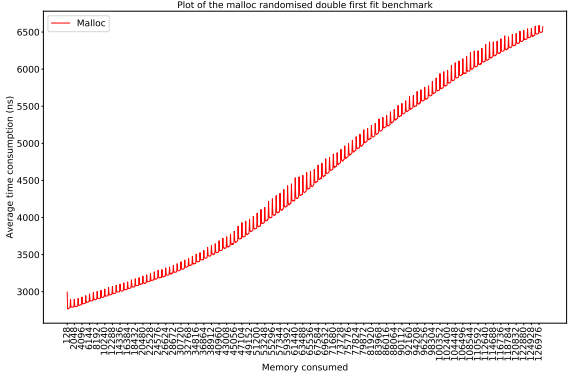


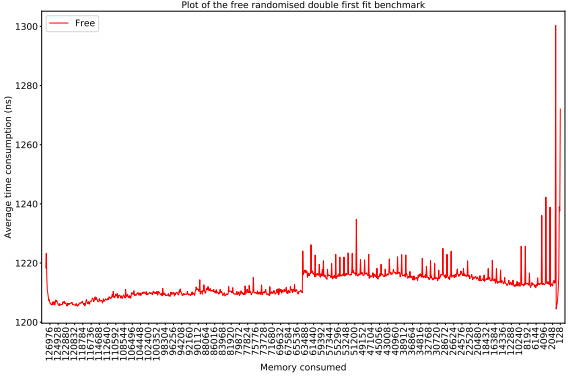
Randomised Buddy Malloc & Free



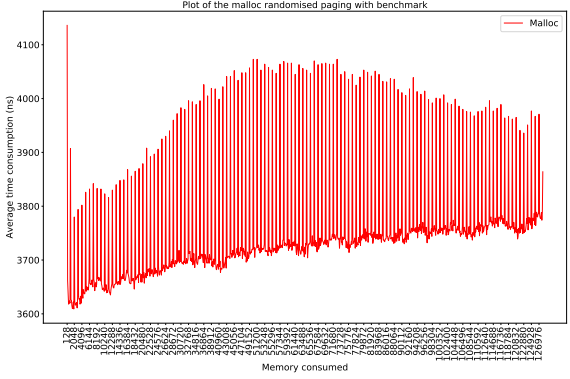


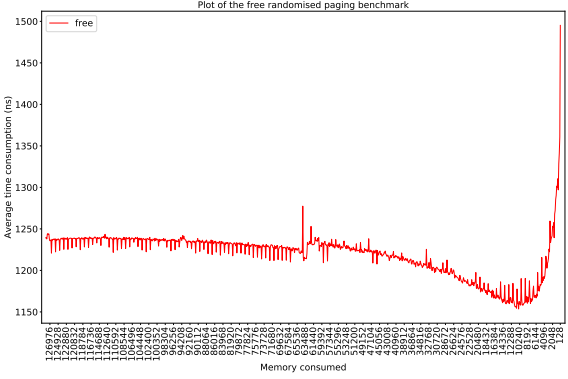
Randomised Double FirstFit Malloc & Free



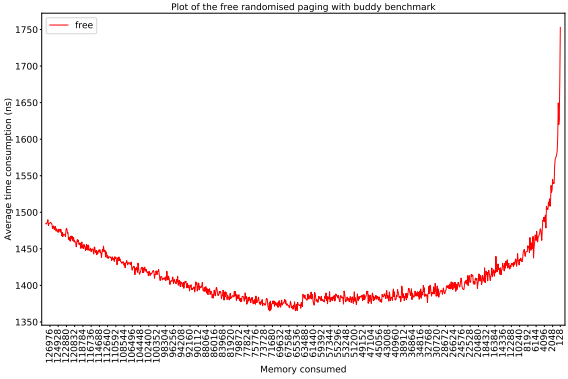


Randomised Paging with FirstFit Malloc & Free

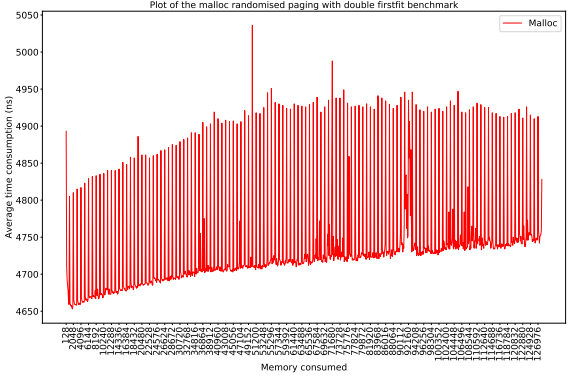


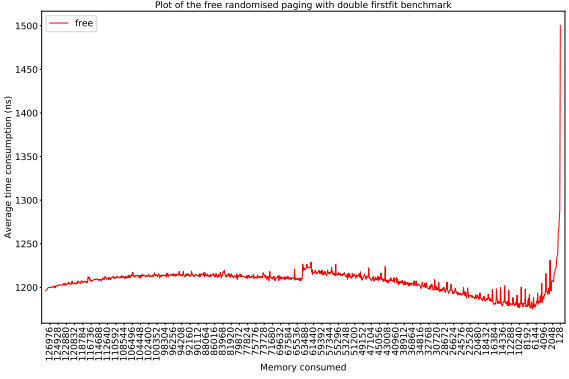


Randomised Paging with Buddy Malloc & Free

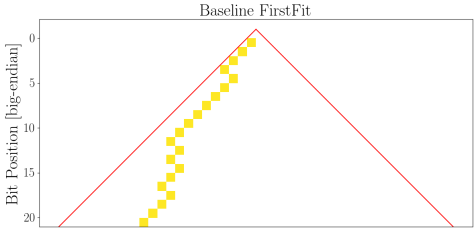


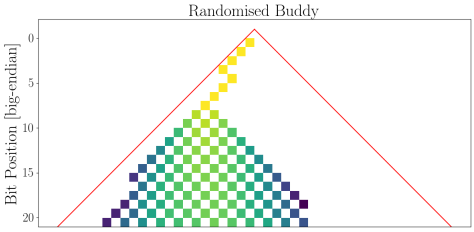
Randomised Paging with Double FirstFit Malloc & Free

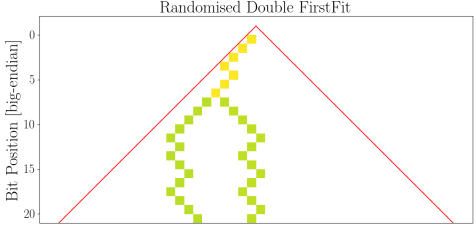


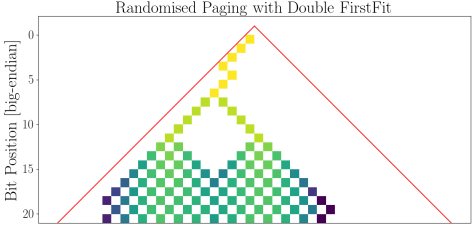


Heat Maps of the 1st Allocation of All Memory Allocators

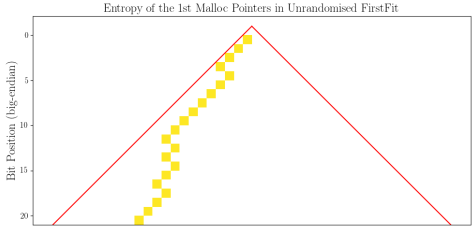


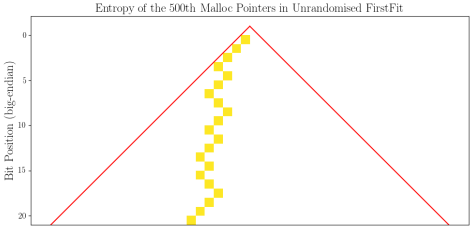


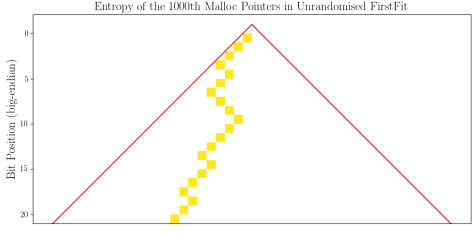


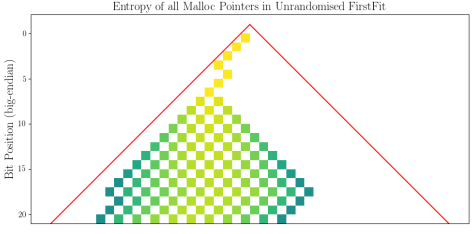


FirstFit (Baseline)

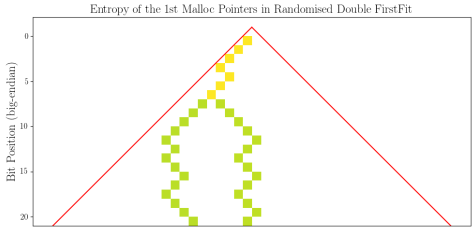


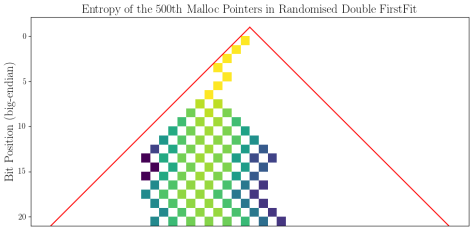


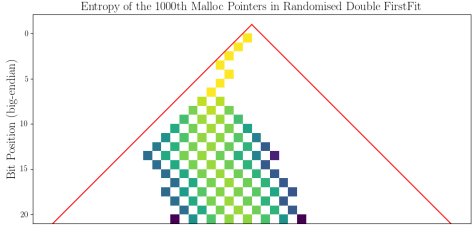


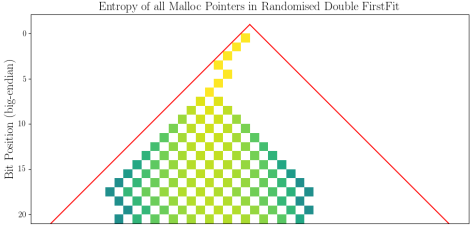


Randomised Double FirstFit

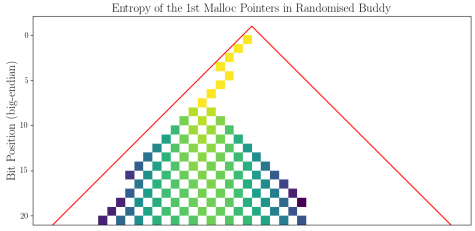


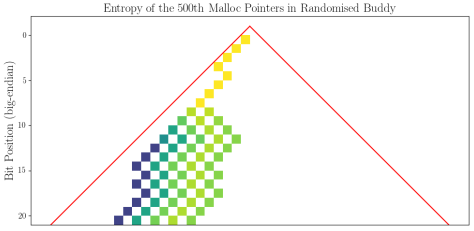


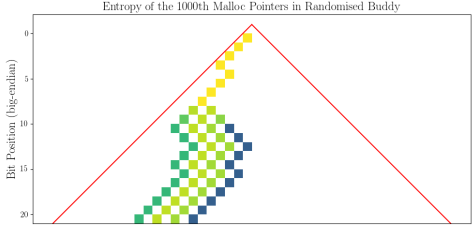


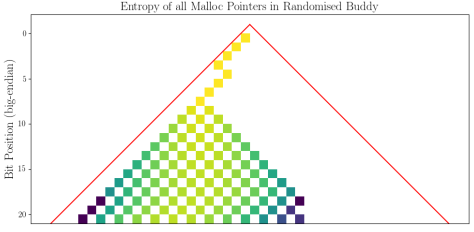


Randomised Buddy

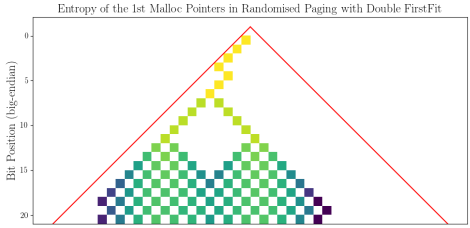


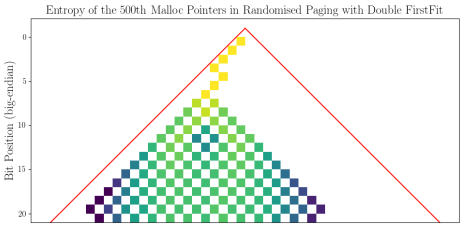


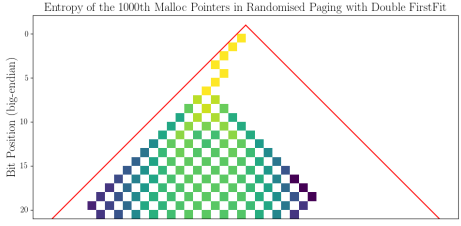


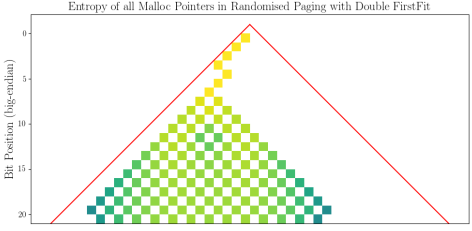


Paging With Double FirstFit

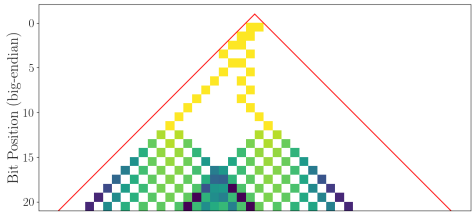


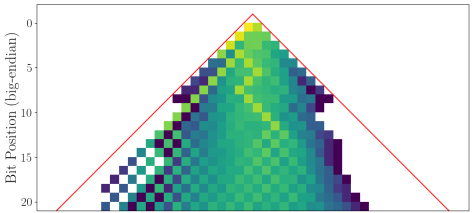


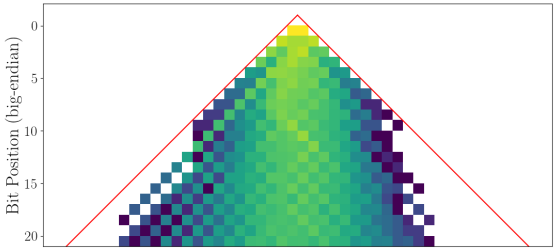




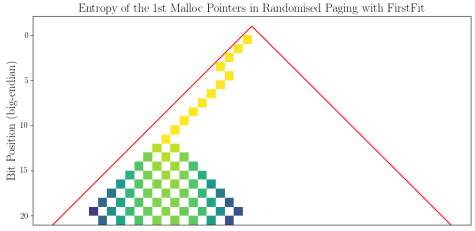
Paging With Double FirstFit (Randomised Allocation Sizes)

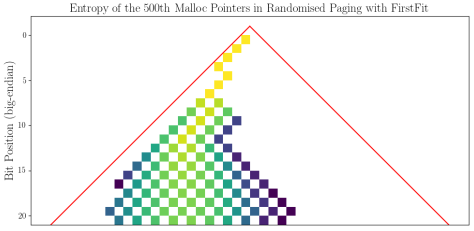


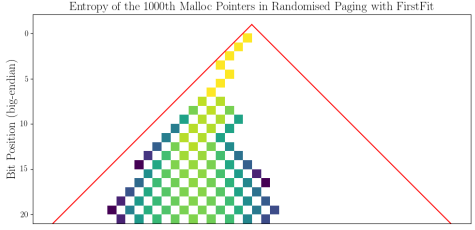


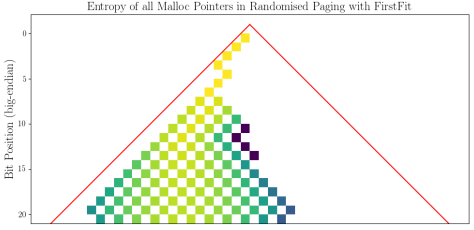


Paging With FirstFit

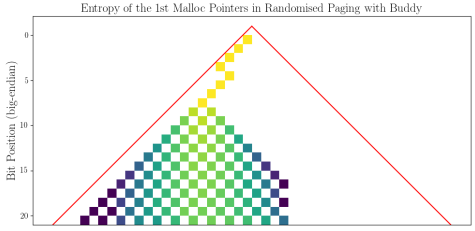


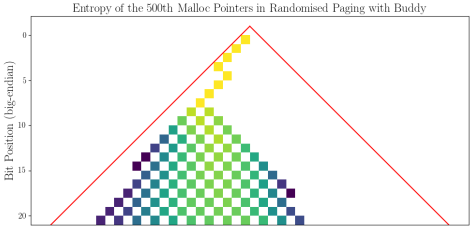


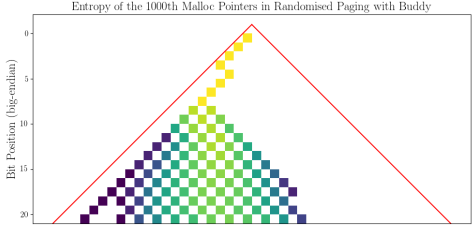


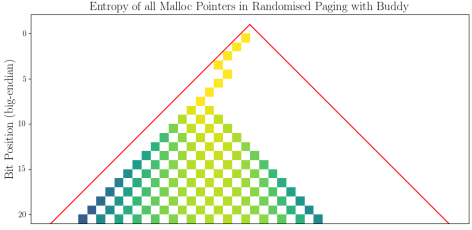


Paging With Buddy

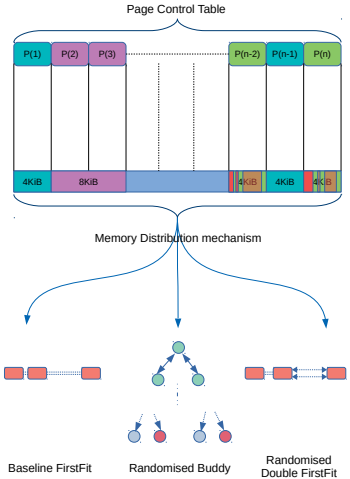




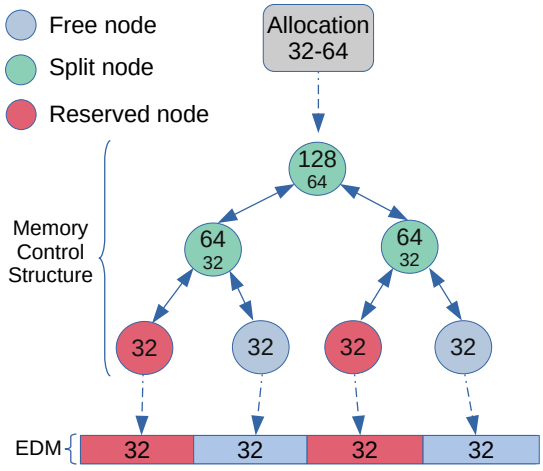




Paging Memory Distribution Mechanisms



Buddy Fragmentation Problem



Randomised Buddy in Action