StarMalloc: Verifying a Modern, Hardened Memory Allocator

Antonin Reitz[1], Aymeric Fromherz[1], Jonathan Protzenko[2]
[1]Inria Paris, Prosecco team
[2]Microsoft Research

2024-10-04

The state of critical software

Web browsers, messaging applications, etc, can be considered critical and thus should be:

- secure
- reliable
- fast

The state of critical software

Web browsers, messaging applications, etc, can be considered critical and thus should be:

- secure
- reliable
- fast

In practice, such software remains implemented in low-level, error-prone languages

Microsoft + Google: 70% of CVEs in their software are memory-related issues

Memory issues remain a billion dollar problem

Analyzers, safer languages, ..., already exist.
What can we do better?

Memory issues remain a billion dollar problem

Analyzers, safer languages, ..., already exist.
What can we do better?

"Software needs seatbelts and airbags"[1]

Security-oriented memory allocators can provide mitigations against memory corruptions, reducing their impact

[1] Berger, CACM 2012

Software stack sketch and memory management

web browser, terminal, etc	$\subset$	user applications

Software stack sketch and memory management

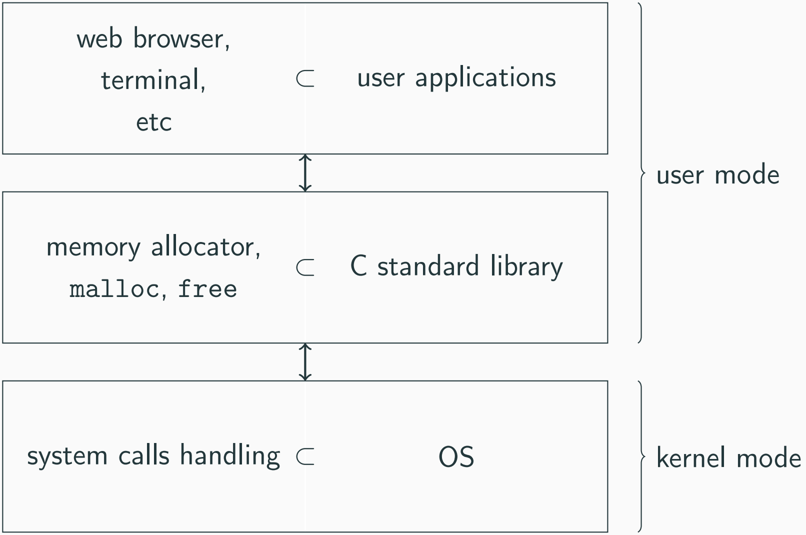
web browser, terminal, etc	⊂	user applications

system calls handling	⊂	OS	} kernel mode

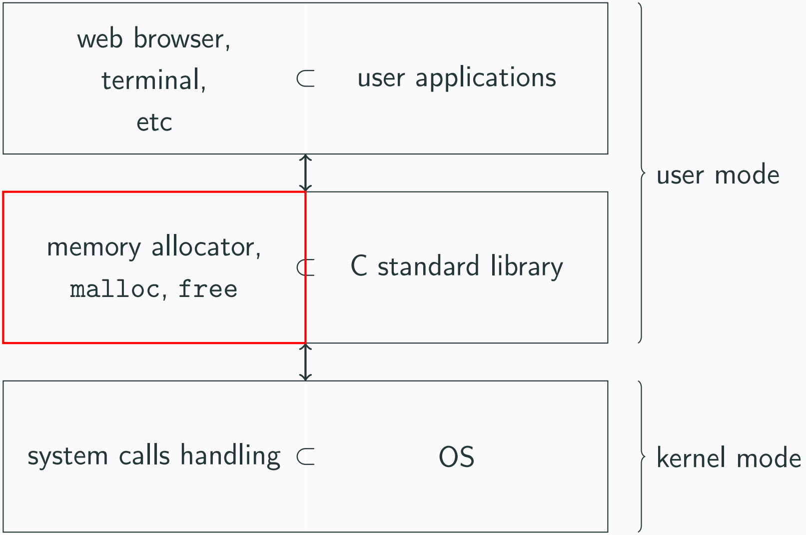
Software stack sketch and memory management

web browser, terminal, etc	⊂	user applications	
memory allocator, `malloc`, `free`	⊂	C standard library	
system calls handling	⊂	OS	} kernel mode

Software stack sketch and memory management



Software stack sketch and memory management



Allocator design space = vast

Allocators can be used in a large diversity of environments

- concurrent allocations
- small available memory space
- need for performance
- need for security

Allocator design space = vast

Allocators can be used in a large diversity of environments

- concurrent allocations
- small available memory space
- need for performance
- need for security $\implies$ security-oriented allocators

Allocator design space = vast

Allocators can be used in a large diversity of environments

- concurrent allocations
- small available memory space
- need for performance
- need for security $\implies$ security-oriented allocators

There is not one allocator design satisfying all possible constraints

What about bugs in the allocator?

Multiple implementation bugs in recent years: GNU malloc (glibc allocator)[2], Scudo (Android allocator)[3], ...

What about bugs in the allocator?

Multiple implementation bugs in recent years: GNU malloc (glibc allocator)[2], Scudo (Android allocator)[3], ...

[2] CVE-2017-17426, high severity
[3] CVE-2023-21367, medium severity

Security-oriented allocators: constraints

Security-oriented memory allocators:

- ✔ can provide mitigations against memory corruptions
- ⚠ can have their own bugs

Security-oriented allocators: constraints

Security-oriented memory allocators:

- ✔ can provide mitigations against memory corruptions
- ⚠ can have their own bugs

$\Longrightarrow$ Need for a verified implementation that meets end users needs, including performance and security

Verifying a security-oriented allocator: challenges

- relating metadata and available memory space
- efficient datastructures without dynamic memory allocation
- interacting with the OS: modelizing syscalls
- modern allocator: concurrency

Verifying a security-oriented allocator: challenges

- relating metadata and available memory space
- efficient datastructures without dynamic memory allocation
- interacting with the OS: modelizing syscalls
- modern allocator: concurrency
- proof engineering: result should be easy to extend, e.g. when implementing new security mechanisms
  $\implies$ composable abstraction layers, **verification methodology**

Contributions

`StarMalloc` is part of our contributions,
as the first **verified** general-purpose userspace memory allocator

Verification theorem states that `StarMalloc` is **functionally correct**

A specific **verification methodology** was used

Contributions

`StarMalloc` is part of our contributions,
as the first **verified** general-purpose userspace memory allocator

Verification theorem states that `StarMalloc` is **functionally correct**

A specific **verification methodology** was used

Out-of-scope:

- memory allocator design
- security theorem

`StarMalloc` architecture

Security-oriented userspace memory allocator: design choices

`StarMalloc` **is heavily inspired from** `hardened_malloc`[4], an **unverified** modern security-focused **general-purpose** memory allocator

Security-oriented userspace memory allocator: design choices

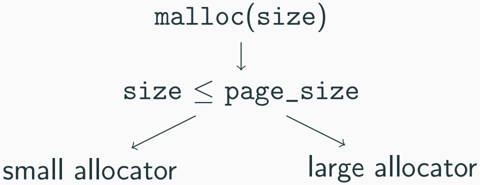
`StarMalloc` **is heavily inspired from** `hardened_malloc`[4], an **unverified** modern security-focused **general-purpose** memory allocator

- most common allocations are small $\Longrightarrow$ they should be fast
- low memory fragmentation
- security by default

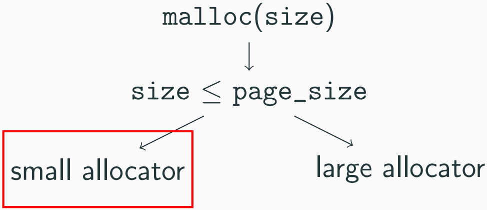
$\Longrightarrow$ dedicated architecture/memory layout

[4] `https://github.com/GrapheneOS/hardened_malloc`

StarMalloc architecture



StarMalloc architecture



today: focus on small allocations

StarMalloc architecture: small allocations

malloc(24)

size≤ page_size

StarMalloc architecture: small allocations

malloc(24)

size≤ page_size

↓

size class selection

16B size class	32B size class	···

StarMalloc architecture: small allocations

`malloc(24)`

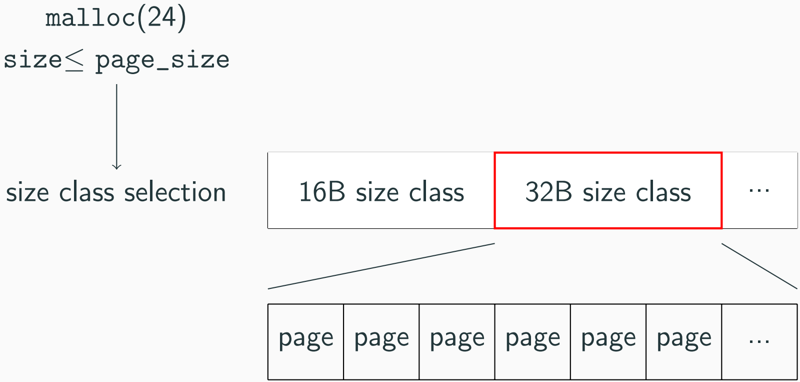
`size≤ page_size`

↓

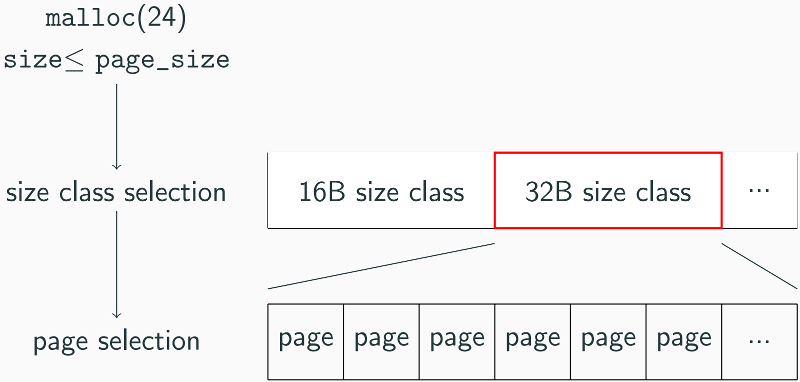
size class selection

16B size class	32B size class	···

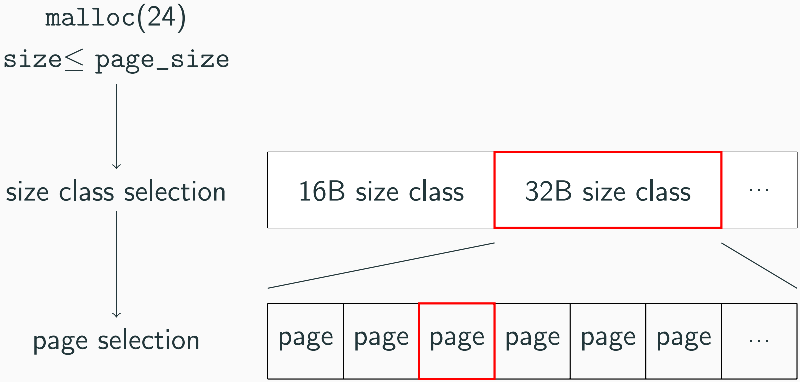
StarMalloc architecture: small allocations



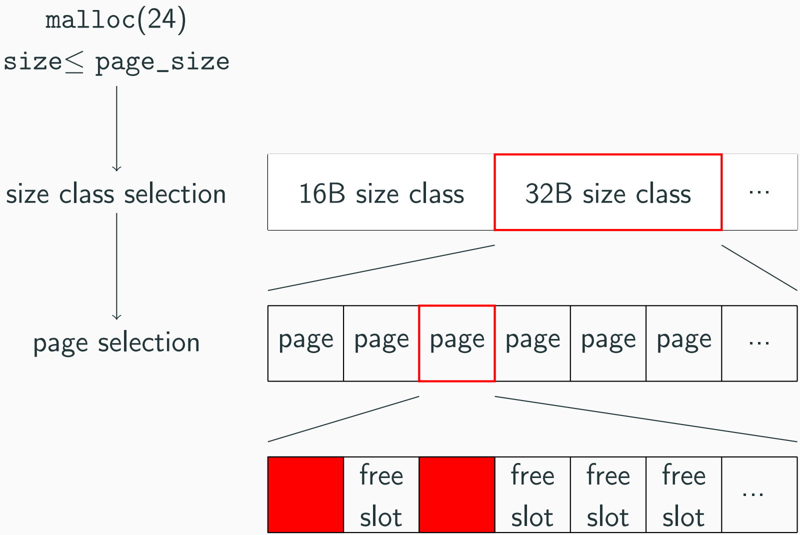
StarMalloc architecture: small allocations



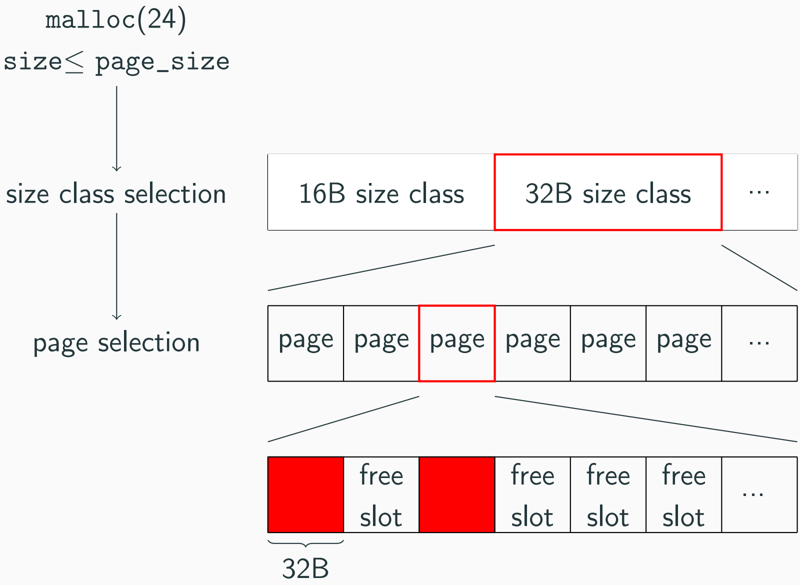
StarMalloc architecture: small allocations



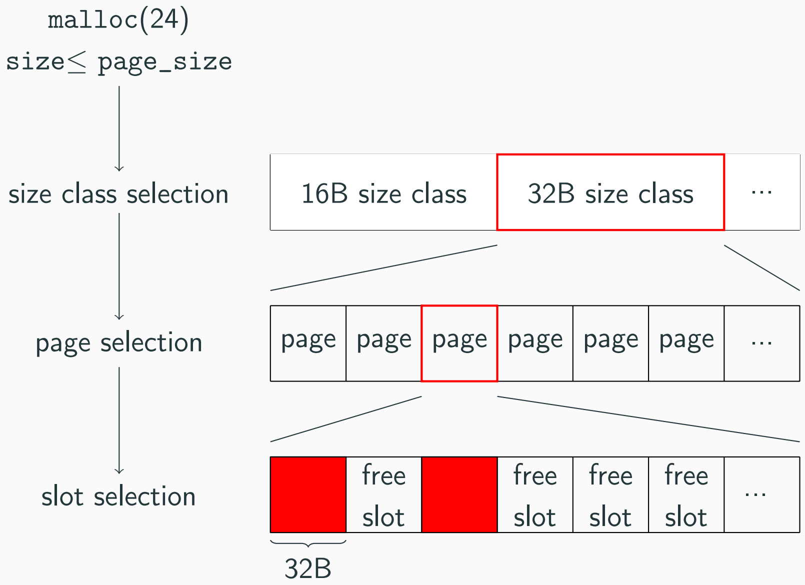
StarMalloc architecture: small allocations



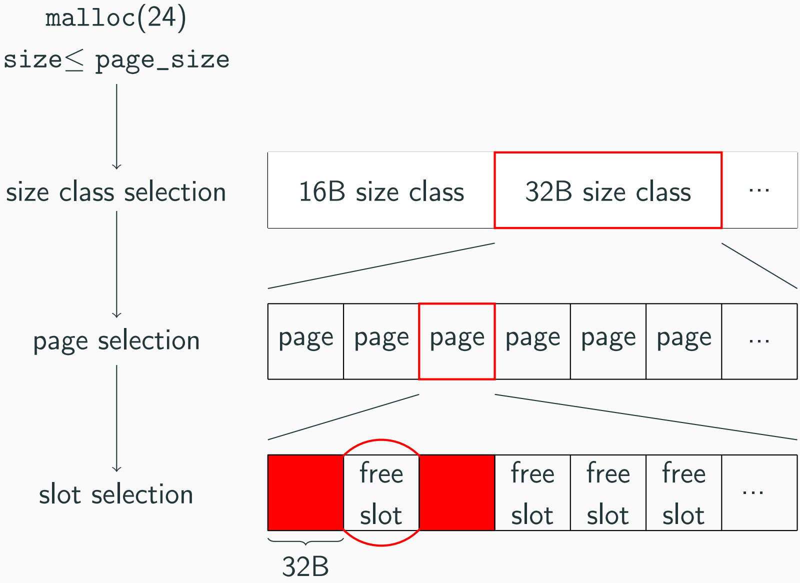
StarMalloc architecture: small allocations



StarMalloc architecture: small allocations

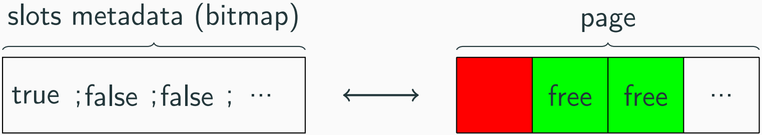


StarMalloc architecture: small allocations



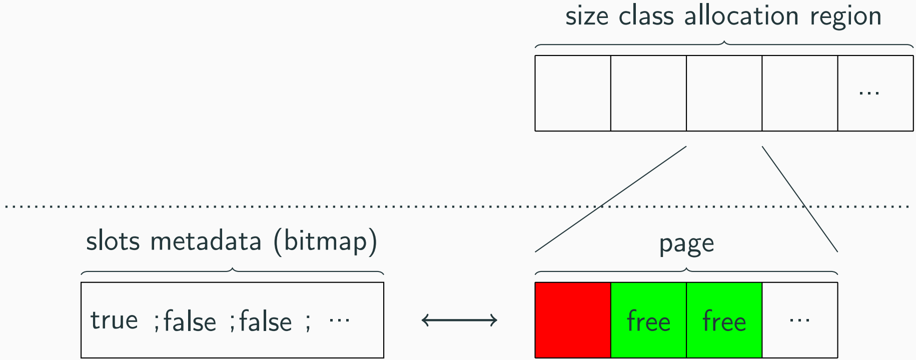
Relating memory and metadata: slots and pages

StarMalloc stores metadata about the allocation region in another disjoint region



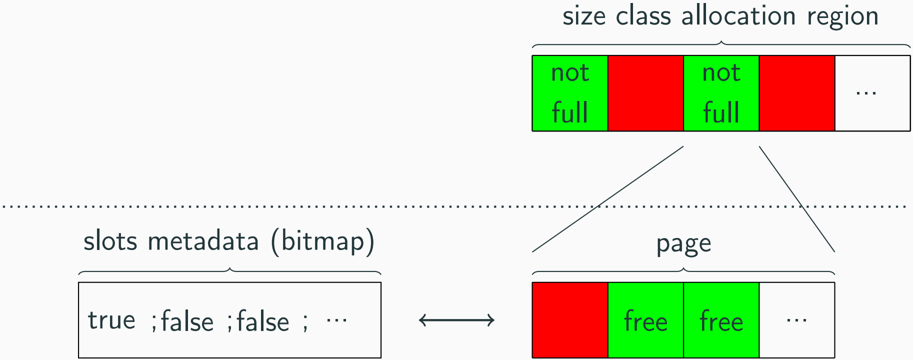
Relating memory and metadata: slots and pages

StarMalloc stores metadata about the allocation region in another disjoint region



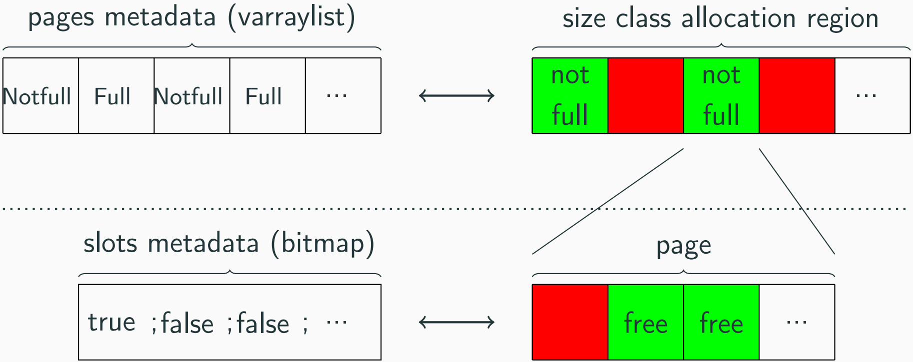
Relating memory and metadata: slots and pages

StarMalloc stores metadata about the allocation region in another disjoint region



Relating memory and metadata: slots and pages

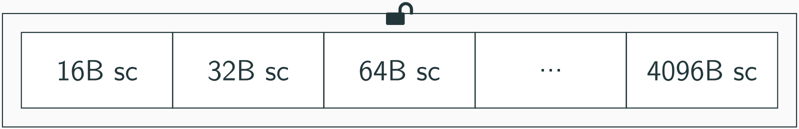
StarMalloc stores metadata about the allocation region in another disjoint region



Fine-grained locks: size classes

allocation must be thread-safe

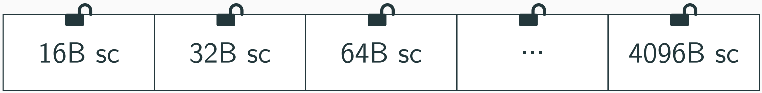
size class selection



Fine-grained locks: size classes

allocation must be thread-safe

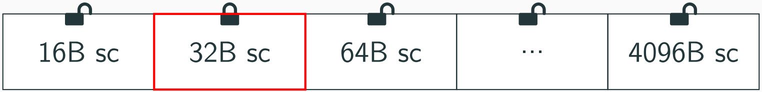
size class selection



Fine-grained locks: size classes

allocation must be thread-safe

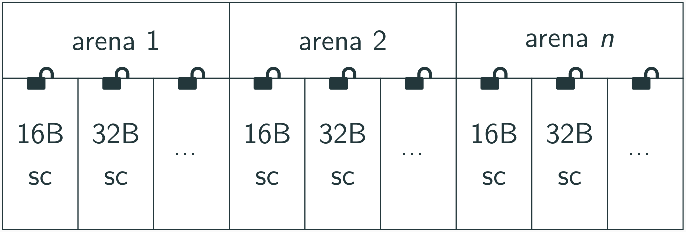
size class selection



Fine-grained locks: arenas

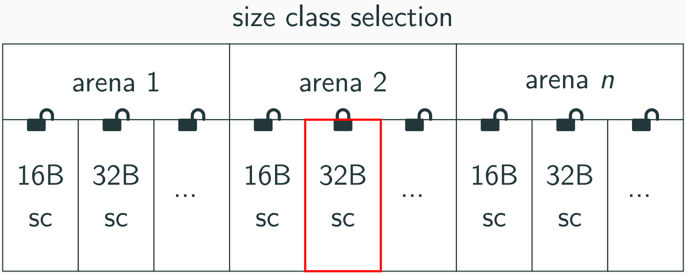
- each thread has an assigned arena
- several threads can share an arena
- threads can free in other arenas

size class selection



Fine-grained locks: arenas

- each thread has an assigned arena
- several threads can share an arena
- threads can free in other arenas



Verification methodology

Verification software: $F^\star$

$F^\star$= a proof-oriented programming language, used as a verification framework, that supports:

- dependent types
- semi-automated verification using an SMT solver

Separation logic 101

- $H_1 \star H_2$ means H_1 and H_2 are heap predicates valid in disjoint memory regions

Separation logic 101

- $H_1 \star H_2$ means H_1 and H_2 are heap predicates valid in disjoint memory regions

- modular reasoning relying on frame rule: $\dfrac{\{\,P\,\}\;c\;\{\,Q\,\}}{\{\,R \star P\,\}\;c\;\{\,Q \star R\,\}}$

Separation logic 101

- $H_1 \star H_2$ means H_1 and H_2 are heap predicates valid in disjoint memory regions

- modular reasoning relying on frame rule: $\dfrac{\{\,P\,\}\ c\ \{\,Q\,\}}{\{\,R \star P\,\}\ c\ \{\,Q \star R\,\}}$

- concurrent separation logic: $\dfrac{\{\,P_1\,\}\ c_1\ \{\,Q_1\,\} \qquad \{\,P_2\,\}\ c_2\ \{\,Q_2\,\}}{\{\,P_1 \star P_2\,\}\ c_1 || c_2\ \{\,Q_1 \star Q_2\,\}}$

Separation logic 101

- $H_1 \star H_2$ means H_1 and H_2 are heap predicates valid in disjoint memory regions

- modular reasoning relying on frame rule: $\dfrac{\{\,P\,\}\ c\ \{\,Q\,\}}{\{\,R \star P\,\}\ c\ \{\,Q \star R\,\}}$

- concurrent separation logic: $\dfrac{\{\,P_1\,\}\ c_1\ \{\,Q_1\,\} \qquad \{\,P_2\,\}\ c_2\ \{\,Q_2\,\}}{\{\,P_1 \star P_2\,\}\ c_1 || c_2\ \{\,Q_1 \star Q_2\,\}}$

- reference assignment example

$$\{\,r \mapsto v\,\}\ r := 42\ \{\,r \mapsto 42\,\}$$

Verification software: `Steel`

`Steel`: a concurrent separation logic embedded in F⋆

key design feature: memory shape and memory content proof obligations discharged separately

---

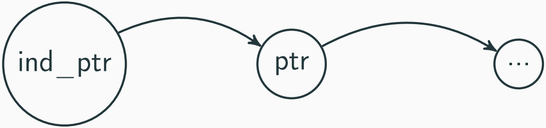
```
val swap (#a:Type) (r1 r2: ref a)
  : Steel unit
  (vptr r1 ⋆ vptr r2)
  (fun _ -> vptr r1 ⋆ vptr r2)
  (requires fun _ ->
    True)
  (ensures fun h0 _ h1 ->
    v_ref r2 h1 == v_ref r1 h0 /\
    v_ref r1 h1 == v_ref r2 h0)
```

Lines 3–4: memory shape (tactic)

Lines 5–9: memory content (SMT solver)

Combinators: sldep

sldep = dependent star



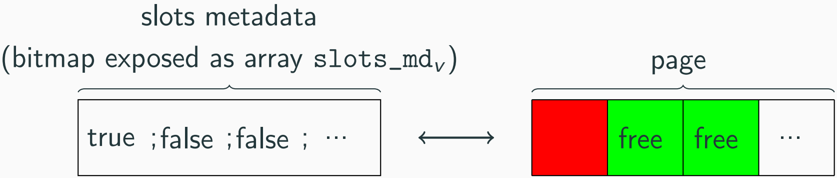

```
let ind_ref (#a: Type)
  (ind_ptr: ref (ref a))
  : slprop
  =
  sldep (vptr ind_ptr) (fun ptr -> vptr ptr)
```

Combinators: `slrefine`

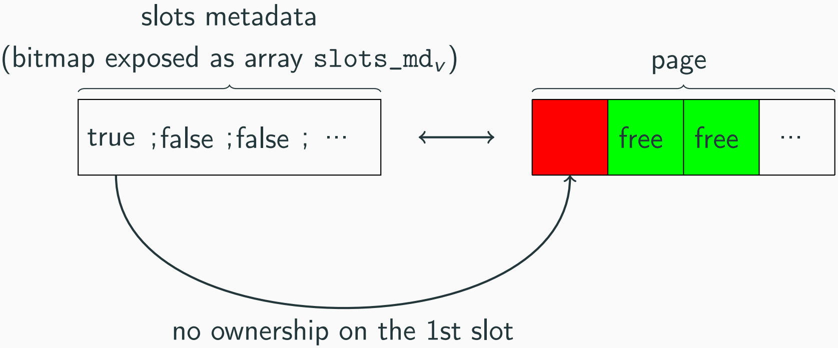
`slrefine` = refinement over `slprop` memory content

```
let refined_ref (ptr: ref int)
  : slprop
  =
  slrefine (vptr ptr) (fun v -> v == 42)
```

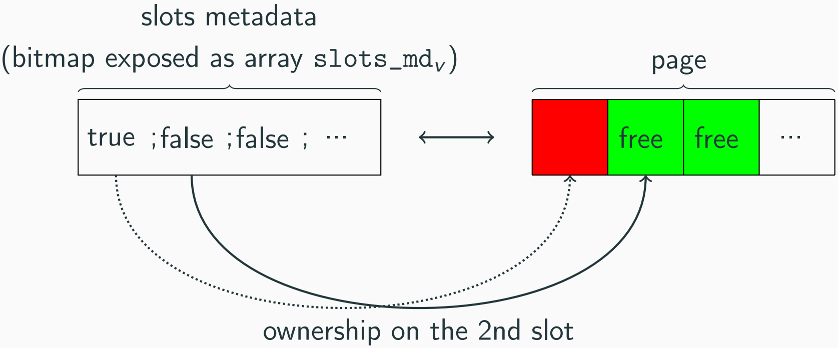
A reusable user-defined, higher-order combinator: `starseq`



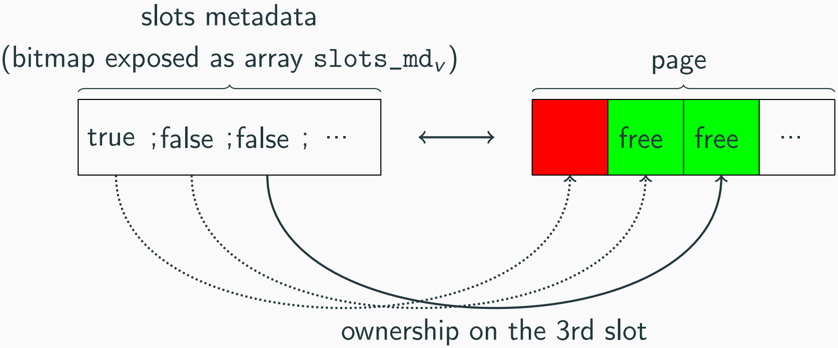
A reusable user-defined, higher-order combinator: `starseq`



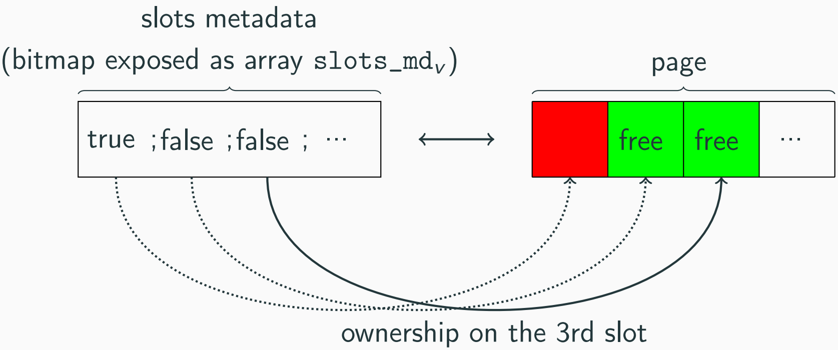
A reusable user-defined, higher-order combinator: `starseq`



A reusable user-defined, higher-order combinator: `starseq`



A reusable user-defined, higher-order combinator: `starseq`




```
let available_slot b slot = if b then emp else slarray slot

let pred slots_md_v page =
  available_slot slots_md_v.[0] (ith_slot page 0) ⋆
  available_slot slots_md_v.[1] (ith_slot page 1) ⋆
  available_slot slots_md_v.[2] (ith_slot page 2) ⋆ ...
```

Combinators: `starseq`

- relating two disjoint arrays
- higher-order
- user-defined

```
type idx (#a: Type) (s: seq a) = i:nat{i <= length s}

let starseq' (p: a -> nat -> slprop) (s: seq a) (i: idx s) =
  if i = length s
  then emp
  else starseq' p s.[i] i `star` starseq' p s (i+1)

let starseq p s : slprop = starseq' p s 0
```

Combinators: `starseq`

- relating two disjoint arrays
- higher-order
- user-defined

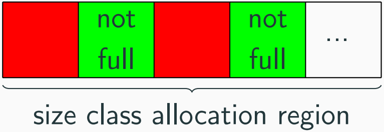
```
type idx (#a: Type) (s: seq a) = i:nat{i <= length s}

let starseq' (p: a -> nat -> slprop) (s: seq a) (i: idx s) =
  if i = length s
  then emp
  else starseq' p s.[i] i `star` starseq' p s (i+1)

let starseq p s : slprop = starseq' p s 0
```

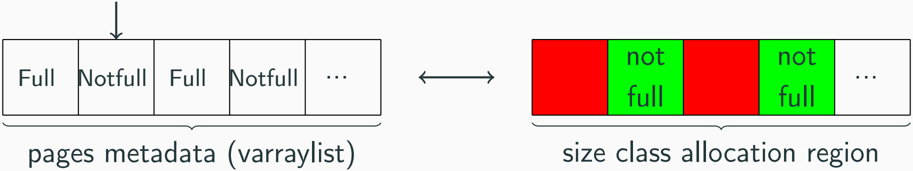
used to relate slots and slots metadata + pages and pages metadata

Pages metadata: the `varraylist` datastructure

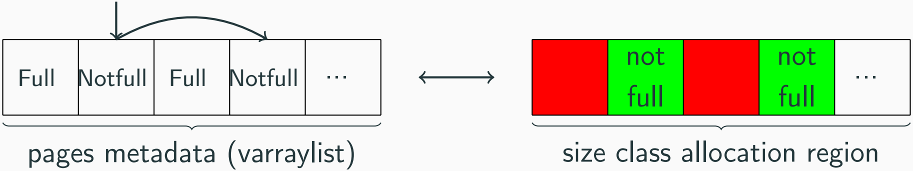


Pages metadata: the `varraylist` datastructure

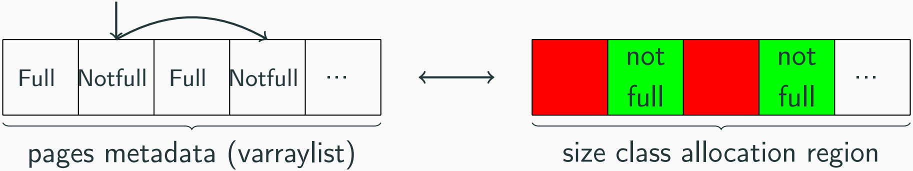
Pages metadata: the `varraylist` datastructure



Pages metadata: the `varraylist` datastructure



Pages metadata: the varraylist datastructure

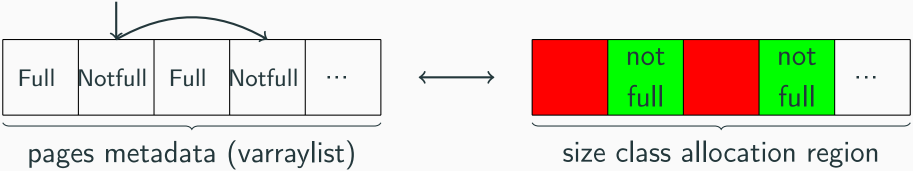



```
val is_list (#a:Type)
  (hd:nat) (s:Seq.seq (cell a)) : prop

let varraylist_refine (#a:Type)
  (hd:nat) (s:Seq.seq (cell a)) : prop =
  is_list hd s

let varraylist (#a:Type) (r:A.array (cell a))
  (hd:nat) : slprop
  = A.varray r `slrefine`
      (fun s -> varraylist_refine hd s)
```

Pages metadata: the `varraylist` datastructure

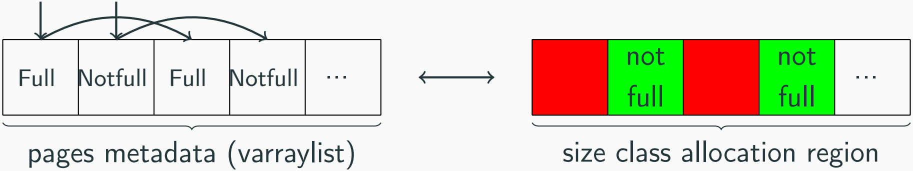



```
val is_list (#a:Type) (pred: a -> prop)
  (hd:nat) (s:Seq.seq (cell a)) : prop

let varraylist_refine (#a:Type) (pred1: a -> prop)
  (hd1:nat) (s:Seq.seq (cell a)) : prop =
  is_list pred1 hd1 s

let varraylist (#a:Type) (r:A.array (cell a))
  (pred1: a -> prop) (hd1:nat) : slprop
  = A.varray r `slrefine`
      (fun s -> varraylist_refine pred1 hd1 s)
```

Pages metadata: the `varraylist` datastructure

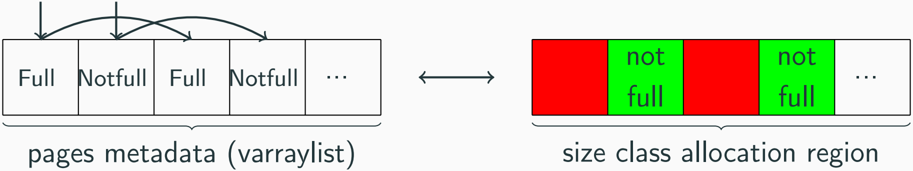



```
val is_list (#a:Type) (pred: a -> prop)
  (hd:nat) (s:Seq.seq (cell a)) : prop

let varraylist_refine (#a:Type) (pred1 pred2: a -> prop)
  (hd1 hd2:nat) (s:Seq.seq (cell a)) : prop =
  is_list pred1 hd1 s /\ is_list pred2 hd2 s

let varraylist (#a:Type) (r:A.array (cell a))
  (pred1 pred2: a -> prop) (hd1 hd2:nat) : slprop
  = A.varray r `slrefine`
      (fun s -> varraylist_refine pred1 pred2 hd1 hd2 s)
```

Pages metadata: the varraylist datastructure




```
val is_dlist (#a:Type) (pred: a -> prop)
  (hd:nat) (s:Seq.seq (cell a)) : prop

let varraylist_refine (#a:Type) (pred1 pred2: a -> prop)
  (hd1 hd2:nat) (s:Seq.seq (cell a)) : prop =
  is_dlist pred1 hd1 s /\ is_dlist pred2 hd2 s

let varraylist (#a:Type) (r:A.array (cell a))
  (pred1 pred2: a -> prop) (hd1 hd2:nat) : slprop
  = A.varray r `slrefine`
      (fun s -> varraylist_refine pred1 pred2 hd1 hd2 s)
```

Pages metadata: the varraylist datastructure

```
let varraylist_refine (#a:Type)
  (hd1 hd2 hd3 hd4 hd5 last5 size5:nat)
  (s:Seq.seq (cell a)) : prop
  =
  is_dlist is_partial hd1 s /\
  is_dlist is_full hd2 s /\
  is_dlist is_empty hd3 s /\
  is_dlist is_guard hd4 s /\
  is_queue is_quarantined hd5 last5 s /\
  cardinality (ptrs_in hd5 s) == size5 /\
  size5 <= SizeT.v Config.quarantine_queue_length /\
  disjoint5 s hd1 hd2 hd3 hd4 hd5
```

5 doubly-linked lists:

- not full (partial), full, ...

Pages metadata: the varraylist datastructure

```
let varraylist_refine (#a:Type)
  (hd1 hd2 hd3 hd4 hd5 last5 size5:nat)
  (s:Seq.seq (cell a)) : prop
  =
  is_dlist is_partial hd1 s /\
  is_dlist is_full hd2 s /\
  is_dlist is_empty hd3 s /\
  is_dlist is_guard hd4 s /\
  is_queue is_quarantined hd5 last5 s /\
  cardinality (ptrs_in hd5 s) == size5 /\
  size5 <= SizeT.v Config.quarantine_queue_length /\
  disjoint5 s hd1 hd2 hd3 hd4 hd5
```

---

5 doubly-linked lists:

- not full (partial), full, ...

Pages metadata: the `varraylist` datastructure

```
let varraylist_refine (#a:Type)
  (hd1 hd2 hd3 hd4 hd5 last5 size5:nat)
  (s:Seq.seq (cell a)) : prop
  =
  is_dlist is_partial hd1 s /\
  is_dlist is_full hd2 s /\
  is_dlist is_empty hd3 s /\
  is_dlist is_guard hd4 s /\
  is_queue is_quarantined hd5 last5 s /\
  cardinality (ptrs_in hd5 s) == size5 /\
  size5 <= SizeT.v Config.quarantine_queue_length /\
  disjoint5 s hd1 hd2 hd3 hd4 hd5
```

5 doubly-linked lists:

- not full (partial), full, empty ($\neq$ partial, fragmentation)
- ...

Pages metadata: the varraylist datastructure

```
let varraylist_refine (#a:Type)
  (hd1 hd2 hd3 hd4 hd5 last5 size5:nat)
  (s:Seq.seq (cell a)) : prop
  =
  is_dlist is_partial hd1 s /\
  is_dlist is_full hd2 s /\
  is_dlist is_empty hd3 s /\
  is_dlist is_guard hd4 s /\
  is_queue is_quarantined hd5 last5 s /\
  cardinality (ptrs_in hd5 s) == size5 /\
  size5 <= SizeT.v Config.quarantine_queue_length /\
  disjoint5 s hd1 hd2 hd3 hd4 hd5
```

5 doubly-linked lists:

- not full (partial), full, empty ($\neq$ partial, fragmentation)
- security: guard pages + ...

Pages metadata: the `varraylist` datastructure

```
let varraylist_refine (#a:Type)
  (hd1 hd2 hd3 hd4 hd5 last5 size5:nat)
  (s:Seq.seq (cell a)) : prop
  =
  is_dlist is_partial hd1 s /\
  is_dlist is_full hd2 s /\
  is_dlist is_empty hd3 s /\
  is_dlist is_guard hd4 s /\
  is_queue is_quarantined hd5 last5 s /\
  cardinality (ptrs_in hd5 s) == size5 /\
  size5 <= SizeT.v Config.quarantine_queue_length /\
  disjoint5 s hd1 hd2 hd3 hd4 hd5
```

5 doubly-linked lists:

- not full (partial), full, empty ($\neq$ partial, fragmentation)
- security: guard pages + quarantined pages (forming a queue)

Other challenges

So far, we focused on verification reasoning

Other challenges

So far, we focused on verification reasoning

Actually, our code also requires:

- genericity: easily-configurable allocator
    - concurrency with the number of arenas
    - security with the set of enabled security mechanisms
    - different data put into memory: different set of sizeclasses

Other challenges

So far, we focused on verification reasoning

Actually, our code also requires:

- genericity: easily-configurable allocator
    - concurrency with the number of arenas
    - security with the set of enabled security mechanisms
    - different data put into memory: different set of sizeclasses
- mutexes: thread-safety

Genericity

we want to write:

```
let sc_list = [16; 32; 64; 80; ...]

let rec init_size_classes memory sizes i = match sizes with
  | [] -> ()
  | hd::tl -> init_size_class memory hd i;
              init_size_classes memory tl (i+1)
```

Genericity

we want to write:

```
let sc_list = [16; 32; 64; 80; ...]

let rec init_size_classes memory sizes i = match sizes with
  | [] -> ()
  | hd::tl -> init_size_class memory hd i;
              init_size_classes memory tl (i+1)
```

after extraction, through normalization and partial evaluation:

```
size_class* size_classes = [...];
init_size_class(memory, size_classes[0U], 16ul);
init_size_class(memory, size_classes[1U], 32ul);
init_size_class(memory, size_classes[2U], 64ul);
init_size_class(memory, size_classes[3U], 80ul);
[...]
```

Ensuring thread-safety

```
val acquire (#p: slprop) (l:lock p)
  : Steel unit
  emp (fun _ -> p)

let f [...] =
  [...] L.acquire l; [...]
```

```
void f(...) =
  [...] pthread_lock(&l); [...]
```

Mismatch:

- Steel: uses mutex as a value
- C: uses mutex's address

$\Longrightarrow$ relies on already existing, conservative **trusted** compilation passes

Verification guarantees

Theorems about user-facing APIs

user-facing APIs = library defined symbols such as `malloc` or `free`

Theorems about user-facing APIs

user-facing APIs = library defined symbols such as `malloc` or `free`

Theorem: `StarMalloc` **is functionally correct with respect to our translation to Steel of the C standard requirements.**

Theorems about user-facing APIs

user-facing APIs = library defined symbols such as `malloc` or `free`

Theorem: `StarMalloc` **is functionally correct with respect to our translation to Steel of the C standard requirements.**

Corollary: `StarMalloc` **is memory safe.**

Theorems about user-facing APIs

user-facing APIs = library defined symbols such as `malloc` or `free`

Theorem: **`StarMalloc` is functionally correct with respect to our translation to Steel of the C standard requirements.**

Corollary: **`StarMalloc` is memory safe.**

`malloc` case, `ptr` being the returned pointer:

- `ptr` can be null
- if not null
    - of at least the requested size
    - client program has total ownership on the corresponding array
    - 16-bytes aligned
    - if the zeroing security mechanism is enabled, contains zeroes

Theorems about user-facing APIs

user-facing APIs = library defined symbols such as `malloc` or `free`

Theorem: `StarMalloc` **is functionally correct with respect to our translation to Steel of the C standard requirements.**

Corollary: `StarMalloc` **is memory safe.**

`malloc` case, `ptr` being the returned pointer:

- `ptr` can be null
- if not null
  - of at least the requested size
  - client program has total ownership on the corresponding array
  - 16-bytes aligned
  - if the zeroing security mechanism is enabled, contains zeroes

Security properties are out-of-scope

What is the TCB?

$$\text{F}^\star\ \text{files} \xrightarrow{\text{F}^\star + \texttt{Steel}} \text{verified F}^\star\ \text{files} \xrightarrow{\texttt{KaRaMeL}} \text{extracted C files} \xrightarrow{\text{C compiler}} \text{shared library}$$

- F*, `Steel`, `KaRaMeL`

What is the TCB?

$$\text{F}^\star\ \text{files} \xrightarrow{\text{F}^\star + \texttt{Steel}} \text{verified F}^\star\ \text{files} \xrightarrow{\texttt{KaRaMeL}} \text{extracted C files} \xrightarrow{\text{C compiler}} \text{shared library}$$

- F*, `Steel`, `KaRaMeL`
- a C compiler

What is the TCB?

$$\text{F}^\star \text{ files} \xrightarrow{\text{F}^\star + \texttt{Steel}} \text{verified F}^\star \text{ files} \xrightarrow{\texttt{KaRaMeL}} \text{extracted C files} \xrightarrow{\text{C compiler}} \text{shared library}$$

- $\text{F}^\star$, `Steel`, `KaRaMeL`
- a C compiler
- our specifications and axiomatizations:
  - specifications: user-facing APIs, e.g. `malloc`, `free`, ...
  - axiomatizations: OS modeling, e.g. the `mmap` syscall
- C glue code (300 LoC)

What is the TCB?

$$\text{F}^\star\ \text{files} \xrightarrow{\text{F}^\star + \texttt{Steel}} \text{verified F}^\star\ \text{files} \xrightarrow{\texttt{KaRaMeL}} \text{extracted C files} \xrightarrow{\text{C compiler}} \text{shared library}$$

- F*, `Steel`, `KaRaMeL`
- a C compiler
- our specifications and axiomatizations:
  - specifications: user-facing APIs, e.g. `malloc`, `free`, ...
  - axiomatizations: OS modeling, e.g. the `mmap` syscall
- C glue code (300 LoC)

`StarMalloc`:

- 42k LoC for verified code ($\geq$ 30% libraries)
- 6k LoC for extracted C

Specifications part of the TCB: user-facing APIs

user-facing APIs
= symbols defined in our library (`malloc`, `free`, ...)

```
val malloc (size: SizeT.t)
  : Steel (array uint8)
  emp
  (fun r -> null_or_slarray r)
  (requires fun _ -> True)
  (ensures fun _ r h1 ->
    let s : seq uint8 = v_null_or_slarray r h1 in
    not (is_null r) ==> (
      length r >= SizeT.v size /\
      [...]
  ))
```

Axiomatizations part of the TCB: modeling external C code

among used syscalls: `mmap`, `munmap`

```
assume val mmap_u8_init (len: SizeT.t)
  : Steel (array uint8)
  emp
  (fun r -> A.varray r)
  (requires fun _ -> SizeT.v len > 0)
  (ensures fun _ r h1 ->
    A.length r == SizeT.v len /\
    A.asel r h1 == Seq.create (SizeT.v len) 0u /\
    array_u8_alignment r page_size
  )
```

Axiomatizations part of the TCB: modeling external C code

among used syscalls: `mmap`, `munmap`

```
assume val mmap_u8_init (len: SizeT.t)
  : Steel (array uint8)
  emp
  (fun r -> A.varray r)
  (requires fun _ -> SizeT.v len > 0)
  (ensures fun _ r h1 ->
    A.length r == SizeT.v len /\
    A.asel r h1 == Seq.create (SizeT.v len) 0u /\
    array_u8_alignment r page_size
  )
```

used at initialization: additional check: if mmap fails, fatal error

Axiomatizations part of the TCB: modeling external C code

among used syscalls: `mmap`, `munmap`

```
assume val mmap_u8_init (len: SizeT.t)
  : Steel (array uint8)
  emp
  (fun r -> A.varray r)
  (requires fun _ -> SizeT.v len > 0)
  (ensures fun _ r h1 ->
    A.length r == page_rounding (SizeT.v len) /\
    A.asel r h1 == Seq.create (SizeT.v len) 0u /\
    array_u8_alignment r page_size
  )
```

additional refinement: mmap returns pages

Experimental evaluation

Experimental evaluation

Assuming a functionally correct implementation, what should be measured?

Experimental evaluation

Assuming a functionally correct implementation, what should be measured?

`StarMalloc` = a verified implementation whose design is heavily inspired by `hardened_malloc`'s design

Experimental evaluation

Assuming a functionally correct implementation, what should be measured?

`StarMalloc` = a verified implementation whose design is heavily inspired by `hardened_malloc`'s design

Measuring the cost of verification: `hardened_malloc` = baseline

mimalloc-bench results

mimalloc-bench[5]: framework for userspace allocator evaluation

- StarMalloc execution time: within 0.70x-1.30x range of that of hardened_malloc (geomean on all 31 benches = 0.97x)

[5] https://github.com/daanx/mimalloc-bench

mimalloc-bench results

mimalloc-bench[5]: framework for userspace allocator evaluation

- StarMalloc execution time: within 0.70x-1.30x range of that of hardened_malloc (geomean on all 31 benches = 0.97x)
- not all implementations work on all benchmarks! (FreeGuard[6], Guarder[7])

[5] https://github.com/daanx/mimalloc-bench
[6] CCS'17
[7] USENIX'18

mimalloc-bench[5]: framework for userspace allocator evaluation

- StarMalloc execution time: within 0.70x-1.30x range of that of hardened_malloc (geomean on all 31 benches = 0.97x)
- not all implementations work on all benchmarks! (FreeGuard[6], Guarder[7])
- security-oriented allocators are slower than performance-oriented ones

[5] https://github.com/daanx/mimalloc-bench
[6] CCS'17
[7] USENIX'18

mimalloc-bench results

mimalloc-bench[5]: framework for userspace allocator evaluation

- StarMalloc execution time: within 0.70x-1.30x range of that of hardened_malloc (geomean on all 31 benches = 0.97x)
- not all implementations work on all benchmarks! (FreeGuard[6], Guarder[7])
- security-oriented allocators are slower than performance-oriented ones
- high variance among results: no allocator outperforming others on all benchmarks

[5] https://github.com/daanx/mimalloc-bench
[6] CCS'17
[7] USENIX'18

Further testing real-world workloads

Firefox ships its own memory allocator: `mozjemalloc`

Further testing real-world workloads

Firefox ships its own memory allocator: `mozjemalloc`

Firefox$^{\bullet}$ = specific build of Firefox using the environment allocator[8]

[8] `disable-jemalloc` build flag

Further testing real-world workloads

Firefox ships its own memory allocator: `mozjemalloc`

Firefox• = specific build of Firefox using the environment allocator[8]

Firefox• with `StarMalloc` as environment allocator

with respect to

Firefox• with `hardened_malloc` as environment allocator:

0.98x on JetStream2[9]

[8] `disable-jemalloc` build flag

[9] `https://browserbench.org/JetStream`

Further testing real-world workloads

Firefox ships its own memory allocator: `mozjemalloc`

Firefox$^{\bullet}$ = specific build of Firefox using the environment allocator[8]

Firefox$^{\bullet}$ with `StarMalloc` as environment allocator
with respect to
Firefox$^{\bullet}$ with `hardened_malloc` as environment allocator:
0.98x on JetStream2[9]

JetStream2 does not specifically test allocator performance, this mostly tells us that `StarMalloc` is a realistic allocator.

[8] `disable-jemalloc` build flag
[9] `https://browserbench.org/JetStream`

Conclusion

Our contributions:

- `StarMalloc` = first **verified** general-purpose **hardened** userspace memory allocator, usable on real-world workloads

Conclusion

Our contributions:

- `StarMalloc` = first **verified** general-purpose **hardened** userspace memory allocator, usable on real-world workloads
- a verification methodology that enabled this work, building upon `Steel`'s methodology

Conclusion

Our contributions:

- `StarMalloc` = first **verified** general-purpose **hardened** userspace memory allocator, usable on real-world workloads
- a verification methodology that enabled this work, building upon `Steel`'s methodology
- `StarMalloc`'s performance is comparable to that of `hardened_malloc`, whose design was used as a basis

Conclusion

Our contributions:

- `StarMalloc` = first **verified** general-purpose **hardened** userspace memory allocator, usable on real-world workloads
- a verification methodology that enabled this work, building upon `Steel`'s methodology
- `StarMalloc`'s performance is comparable to that of `hardened_malloc`, whose design was used as a basis

Antonin Reitz antonin.reitz@inria.fr

Aymeric Fromherz aymeric.fromherz@inria.fr

Jonathan Protzenko protz@microsoft.com