Floorplan: Spatial Layout in Memory Management Systems

Karl Cronburg
karl@cs.tufts.edu

Samuel Z. Guyer
sguyer@cs.tufts.edu

Tufts University | School of Engineering | Department of Computer Science
United States

October 21, 2019

Goals of this Talk

- To familiarize you with Floorplan layout operators.

Goals of this Talk

- To familiarize you with Floorplan layout operators.

Grammar 4: Demarcatable atomic units of memory.

⟨*demarc-val*⟩ ::= (‘#’ | ⟨*formal-id*⟩)? (⟨*enum*⟩ | ⟨*bits*⟩ | ⟨*union*⟩
| ⟨*seq*⟩ | ⟨*ptr*⟩ | ⟨*size-arith*⟩ | ⟨*macro*⟩)

⟨*seq*⟩ ::= ‘`seq`’ ‘{’ ⟨*demarc*⟩ (‘,’ ⟨*demarc*⟩)* ‘,’? ‘}’

⟨*union*⟩ ::= ‘`union`’ ‘{’ ⟨*demarc*⟩ (‘`|`’ ⟨*demarc*⟩)* ‘`|`’? ‘}’

⟨*demarc*⟩ ::= ⟨*field*⟩ | ⟨*layer*⟩ | ⟨*demarc-val*⟩

⟨*field*⟩ ::= ⟨*field-id*⟩ ‘:’ ⟨*demarc-val*⟩

⟨*ptr*⟩ ::= (⟨*layer-id*⟩ | ⟨*field-id*⟩) ‘`ptr`’

⟨*enum*⟩ ::= ‘`enum`’ ‘{’ ⟨*flag-id*⟩ (‘`|`’ ⟨*flag-id*⟩)* ‘`|`’? ‘}’

⟨*bits*⟩ ::= ‘`bits`’ ‘{’ ⟨*bits-exp*⟩ (‘,’ ⟨*bits-exp*⟩)* ‘,’? ‘}’

⟨*bits-exp*⟩ ::= ⟨*field-id*⟩ ‘:’ ⟨*size-arith*⟩

⟨*macro*⟩ ::= ⟨*layer-id*⟩ (‘<’ ⟨*args*⟩ ‘>’)?

⟨*arg*⟩ ::= ⟨*formal-id*⟩ | ⟨*literal*⟩

⟨*args*⟩ ::= ⟨*arg*⟩ (‘,’ ⟨*arg*⟩)* ‘,’?

Goals of this Talk

- To familiarize you with Floorplan layout operators.
- You want to read the paper to understand Floorplan mechanics.

Goals of this Talk

- To familiarize you with Floorplan layout operators.
- You want to read the paper to understand Floorplan mechanics.

Memory Layout Model $\bar{\gamma}$

$$
\begin{array}{ll}
1 & \gamma[\![(\alpha, m, \theta, e_1 + e_2)]\!] \\
2 & \quad = \{\, T\ r_1\ r_2 \mid r_1 \in \bigcup_{i=0}^{m} \gamma[\![(\alpha, i, \theta, e_1)]\!] \\
3 & \qquad\qquad , r_2 \in \gamma[\![(\alpha + \mathrm{leaves}(r_1) \\
4 & \qquad\qquad\quad , m - \mathrm{leaves}(r_1), \theta, e_2)]\!] \,\} \\
5 & \gamma[\![(\alpha, m, \theta, e_1 \parallel e_2)]\!] \quad = \quad \gamma[\![(\alpha, m, \theta, e_1)]\!] \\
6 & \qquad\qquad\qquad\qquad \cup\, \gamma[\![(\alpha, m, \theta, e_2)]\!] \\
7 & \gamma[\![(\alpha, 0, \theta, \mathrm{Prim}\ 0)]\!] \quad = \quad \{\, 0 \text{ bytes} \,\} \\
8 & \gamma[\![(\alpha, m, \theta, \mathrm{Prim}\ n)]\!] \\
9 & \quad \mid m \equiv n = \{\, T\ (1 \text{ bytes})_1\ (\cdots\ \mathrm{T}\ (1 \text{ bytes})_n\ (0 \text{ bytes})) \,\} \\
10 & \quad \mid m \neq n = \emptyset \\
11 & \gamma[\![(\alpha, m, \theta, \mathrm{Con}\ n\ e)]\!] \\
12 & \quad \mid m \equiv n = \gamma[\![(\alpha, m, \theta, e)]\!] \\
13 & \quad \mid m \neq n = \emptyset \\
14 & \gamma[\![(\alpha, m, \theta, \mathrm{e}\ @\ \hat{a})]\!] \\
15 & \quad \mid \alpha \bmod \hat{a} \equiv 0 = \gamma[\![(\alpha, m, \theta, e)]\!] \\
16 & \quad \mid \alpha \bmod \hat{a} \neq 0 = \emptyset
\end{array}
$$

$$
\begin{array}{ll}
17 & \gamma[\![(\alpha, m, \theta, \ell :: e)]\!] \quad = \{\, N\ \ell\ r \\
18 & \qquad\qquad\qquad\qquad \mid r \in \gamma[\![(\alpha, m, \theta, e)]\!] \,\} \\
19 & \gamma[\![(\alpha, m, \theta, \exists\ f\ .\ e)]\!] = \\
20 & \qquad \bigcup_{i=0}^{m} \gamma[\![(\alpha, m, \theta\{f \mapsto i\}, e)]\!] \\
21 & \gamma[\![(\alpha, m, \theta, f\ \#\ e)]\!] \\
22 & \quad \mid f \notin \mathrm{dom}(\theta) = \emptyset \\
23 & \quad \mid m \equiv \theta(f) \equiv 0 = \{\, T\ (0 \text{ bytes})\ (0 \text{ bytes}) \,\} \\
24 & \quad \mid \theta(f) \equiv 0 = \emptyset \\
25 & \quad \mid \theta(f) > 0 \\
26 & \quad = \{\, T\ r_1\ r_2 \\
27 & \qquad \mid r_1 \in \bigcup_{i=0}^{m} \gamma[\![(\alpha, i, \theta, e)]\!] \\
28 & \qquad , r_2 \in \gamma[\![(\alpha + \mathrm{leaves}(r_1), m - \mathrm{leaves}(r_1) \\
29 & \qquad\qquad\quad , \theta\{f \mapsto (\theta(f) - 1)\}, f\ \#\ e)]\!] \\
30 & \qquad , m \equiv \mathrm{leaves}(r_1) + \mathrm{leaves}(r_2) \,\} \\
31 & \bar{\gamma}[\![(\alpha, m, e)]\!] \qquad\qquad = \quad \gamma[\![(\alpha, m, \emptyset, e)]\!]
\end{array}
$$

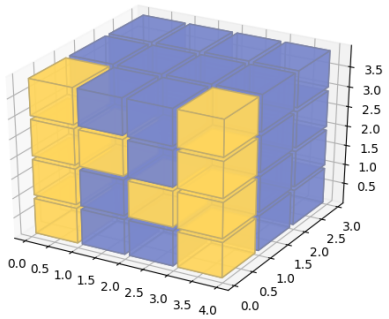
Figure 9. Denotational semantics of Floorplan.

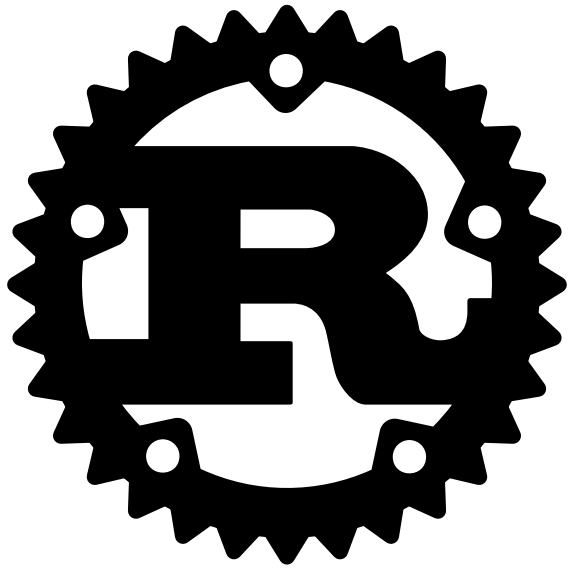
Goals of this Talk

- To familiarize you with Floorplan layout operators.
- You want to read the paper to understand Floorplan mechanics.
- Motivate customized memory managers as a domain lacking in language support.

Goals of this Talk

- To familiarize you with Floorplan layout operators.
- You want to read the paper to understand Floorplan mechanics.
- Motivate customized memory managers as a domain lacking in language support.





Goals of this Talk

What this Talk is *Not*:

- A workshop to learn Floorplan syntax and compilation.

Goals of this Talk

Tufts UNIVERSITY

What this Talk is *Not*:

- A workshop to learn Floorplan syntax and compilation.

$$\mathbb{C}[\![\ \langle \textit{layer-simple} \rangle\ ]\!] =$$
$$\mathbb{C}[\![\ \langle \textit{layer-id} \rangle\ (\text{`<'}\ \langle \textit{formals} \rangle\ \text{`>'})?\ \langle \textit{mag} \rangle?\ \langle \textit{align} \rangle?\ \text{`->'}\ \langle \textit{demarc-val} \rangle\ ]\!], f_i \in \langle \textit{formals} \rangle$$
$$(1) \quad \vDash \langle \textit{layer-id} \rangle :: (\exists\ f_0\ .\ \cdots \exists\ f_n\ .\ (\mathbb{M}[\![\ \langle \textit{mag} \rangle\ ]\!]\ (\mathbb{C}[\![\ \langle \textit{demarc-val} \rangle\ ]\!]\ @\ (\Delta_{\text{byte}}[\![\ \langle \textit{align} \rangle\ ]\!]))))$$

$$\mathbb{C}[\![\ \langle \textit{demarc-val} \rangle\ ]\!] = \mathbb{C}[\![\ \text{`\#'}\ \langle \textit{demarc-val} \rangle\ ]\!]$$
$$(2) \quad \vDash \text{let } f = \texttt{fresh}(\langle \textit{demarc-val} \rangle) \text{ in } \exists\ f\ .\ f\ \#\ \mathbb{C}[\![\ \langle \textit{demarc-val} \rangle\ ]\!]$$

$$\mathbb{C}[\![\ \langle \textit{demarc-val} \rangle\ ]\!] = \mathbb{C}[\![\ \langle \textit{formal-id} \rangle\ \langle \textit{demarc-val} \rangle\ ]\!]$$
$$(3) \quad \vDash \langle \textit{formal-id} \rangle\ \#\ \mathbb{C}[\![\ \langle \textit{demarc-val} \rangle\ ]\!]$$

$$\mathbb{C}[\![\ \langle \textit{seq} \rangle\ ]\!] = \mathbb{C}[\![\ \text{`seq'}\ \text{`\{'}\ \langle \textit{demarc} \rangle_0 \cdots \langle \textit{demarc} \rangle_n\ \text{`\}'}\ ]\!]$$
$$(4) \quad \vDash \mathbb{C}[\![\ \langle \textit{demarc} \rangle_0\ ]\!] + \cdots + \mathbb{C}[\![\ \langle \textit{demarc} \rangle_n\ ]\!]$$

$$\mathbb{C}[\![\ \langle \textit{union} \rangle\ ]\!] =$$
$$\mathbb{C}[\![\ \text{`union'}\ \text{`\{'}\ \langle \textit{demarc} \rangle_0\ \text{`|'} \cdots \text{`|'}\ \langle \textit{demarc} \rangle_n\ \text{`\}'}\ ]\!]$$
$$(5) \quad \vDash \mathbb{C}[\![\ \langle \textit{demarc} \rangle_0\ ]\!] \,||\, \cdots \,||\, \mathbb{C}[\![\ \langle \textit{demarc} \rangle_n\ ]\!]$$

$$\mathbb{C}[\![\ \langle \textit{field} \rangle\ ]\!] = \mathbb{C}[\![\ \langle \textit{field-id} \rangle\ \text{`:'}\ \langle \textit{demarc-val} \rangle\ ]\!]$$
$$(6) \quad \vDash \langle \textit{field-id} \rangle :: \mathbb{C}[\![\ \langle \textit{demarc-val} \rangle\ ]\!]$$

$$(7) \quad \mathbb{C}[\![\ \langle \textit{ptr} \rangle\ ]\!] = \mathbb{C}[\![\ (\langle \textit{layer-id} \rangle \mid \langle \textit{field-id} \rangle)\ \texttt{`ptr'}\ ]\!] \vDash \mathbb{C}[\![\ 1\ \text{word}\ ]\!]$$

$$\mathbb{C}[\![\ \langle \textit{enum} \rangle\ ]\!] = \mathbb{C}[\![\ \texttt{`enum'}\ \text{`\{'}\ \langle \textit{flag-id} \rangle_0 \cdots \langle \textit{flag-id} \rangle_n\ \text{`\}'}\ ]\!]$$
$$(8) \quad \vDash \text{Prim}\left\lceil \log_2(n+1) * \frac{1\ \text{byte}}{8\ \text{bits}} \right\rceil$$

$$\mathbb{C}[\![\ \langle \textit{bits} \rangle\ ]\!] = \mathbb{C}[\![\ \texttt{`bits'}\ \text{`\{'}\ \langle \textit{bits-exp} \rangle_0 \cdots \langle \textit{bits-exp} \rangle_n\ \text{`\}'}\ ]\!]$$
$$(9) \quad \vDash \text{Prim}\left\lceil \left(\sum_{i=0}^{n} (\Delta_{\text{bit}} \langle \textit{bits-exp} \rangle_i) \right) * \frac{1\ \text{byte}}{8\ \text{bits}} \right\rceil$$

$$(10) \quad \mathbb{C}[\![\ \langle \textit{size-arith} \rangle\ ]\!] \vDash \text{Prim}\left(\Delta_{\text{byte}} \langle \textit{size-arith} \rangle\right)$$

Figure 10. Compilation rules for translating surface syntax to a core expression. Syntax inside oxford-like brackets[9] is surface syntax, and syntax after a double-turnstile[10] $\vDash$ is a core expression. Formals support list membership, $\in$.

[9] $[\![\ldots]\!]$ separates raised syntax (inside brackets) from lowered expressions.
[10] A double-turnstile, Foo $[\![\ldots]\!] \vDash$ Bar, reads as "Bar models Foo $[\![\ldots]\!]$".

Goals of this Talk

Tufts UNIVERSITY

What this Talk is *Not*:

- A workshop to learn Floorplan syntax and compilation.
- The paper. Examples are novel, Floorplan syntax herein matches paper.

Goals of this Talk

Tufts UNIVERSITY

What this Talk is *Not*:

- A workshop to learn Floorplan syntax and compilation.
- The paper. Examples are novel, Floorplan syntax herein matches paper.

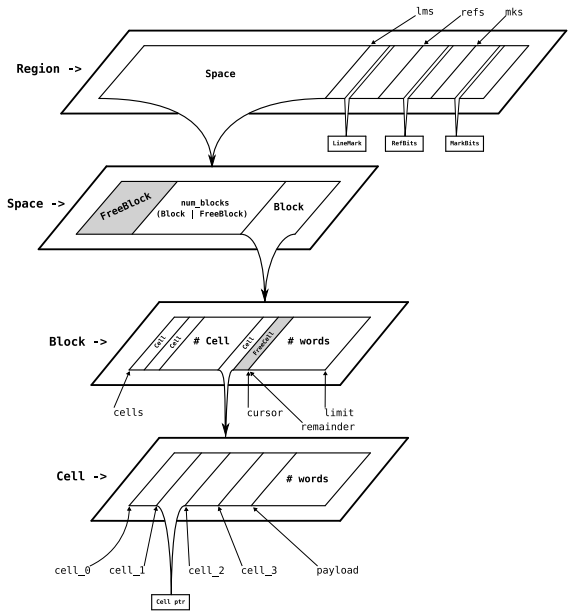


Figure 4. A four-layered demarcation diagram depicting the immix-rust layout. Each layer corresponds to a ⟨*layer*⟩ definition from Figure 5. Each layer is then further demarcated into ⟨*field*⟩ fields, ⟨*demarc-val*⟩ values, and other ⟨*layer*⟩ layers.

Goals of this Talk

Tufts UNIVERSITY

What this Talk is *Not*:

- A workshop to learn Floorplan syntax and compilation.
- The paper. Examples are novel, Floorplan syntax herein matches paper.
- Performance results.

Goals of this Talk

Tufts UNIVERSITY

What this Talk is *Not*:

- A workshop to learn Floorplan syntax and compilation.
- The paper. Examples are novel, Floorplan syntax herein matches paper.
- Performance results.

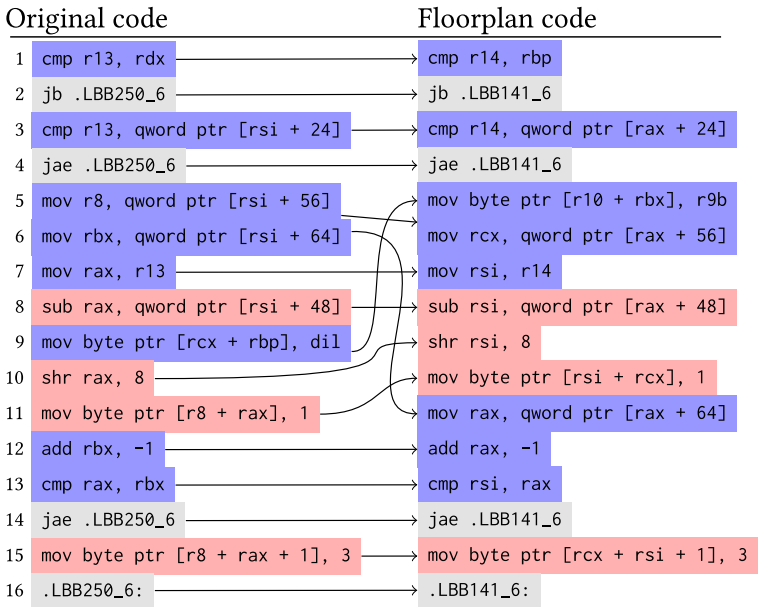
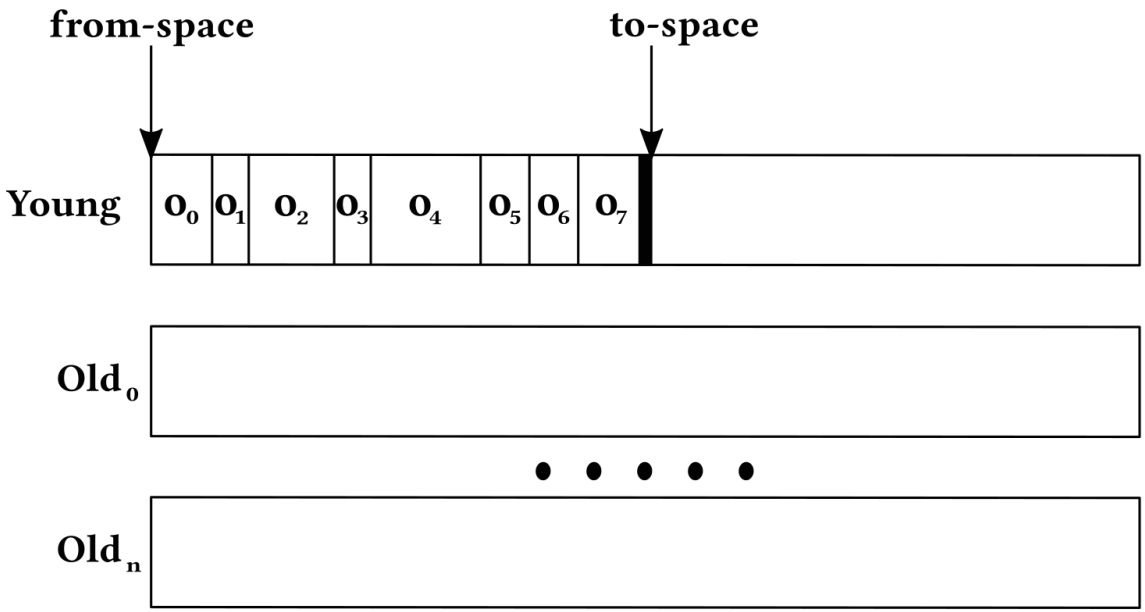


Figure 15. x86 Intel assembly code for marking immix lines.

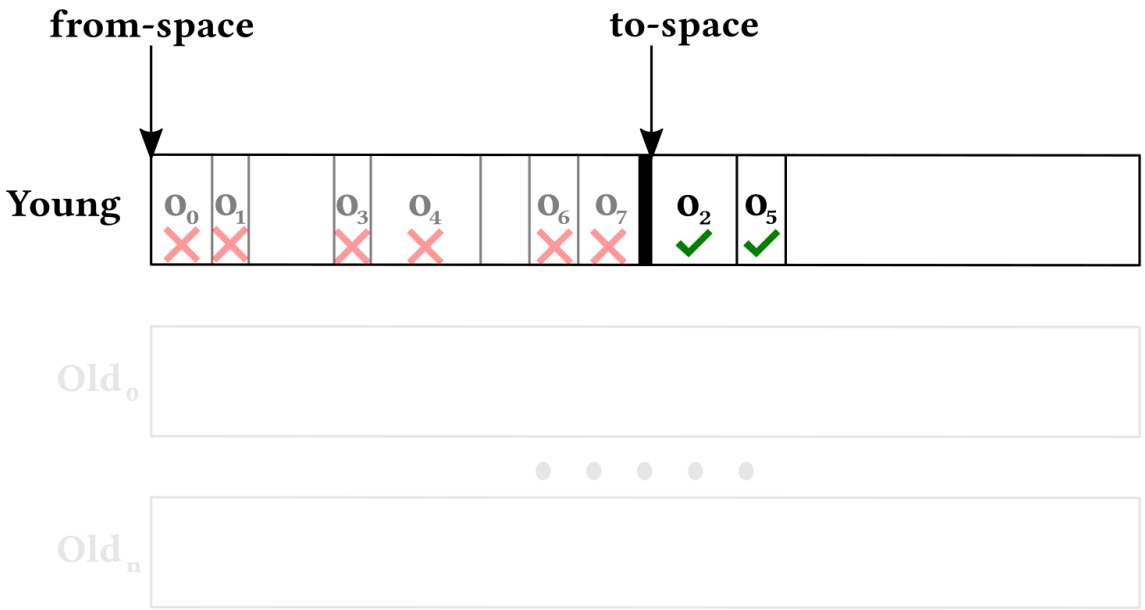
Garbage Collection (GC): The Good

- Modern GCs are *generational*.



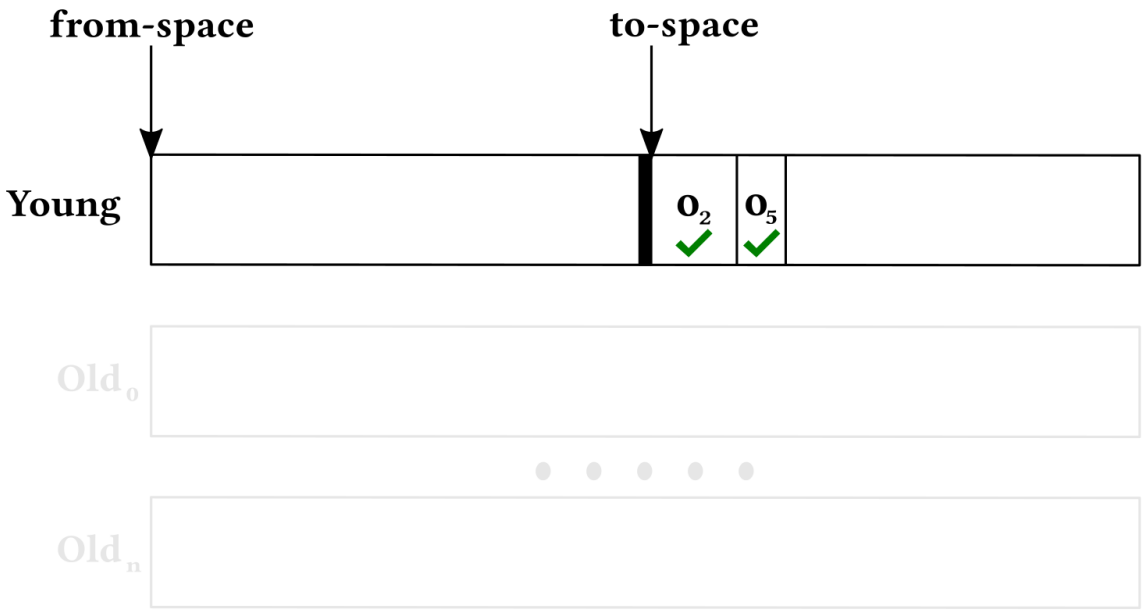
Garbage Collection (GC): The Good

- Modern GCs are *generational*.
- Applications exhibit high allocation rates and objects die young



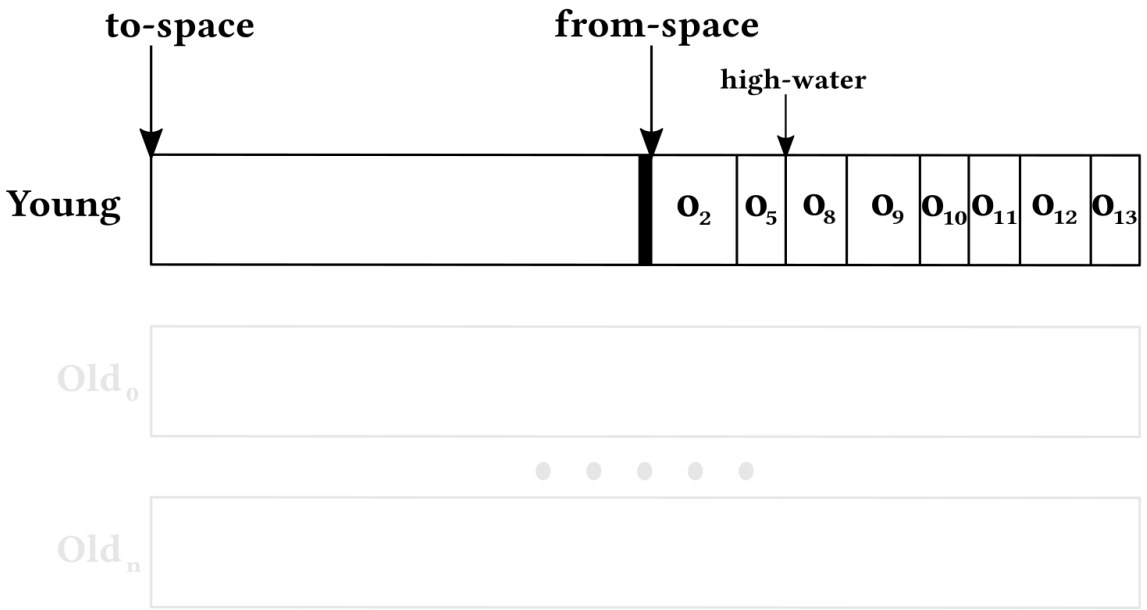
Garbage Collection (GC): The Good

- Modern GCs are *generational*.
- Applications exhibit high allocation rates and objects die young
  - Application constructs lots of ephemeral data.



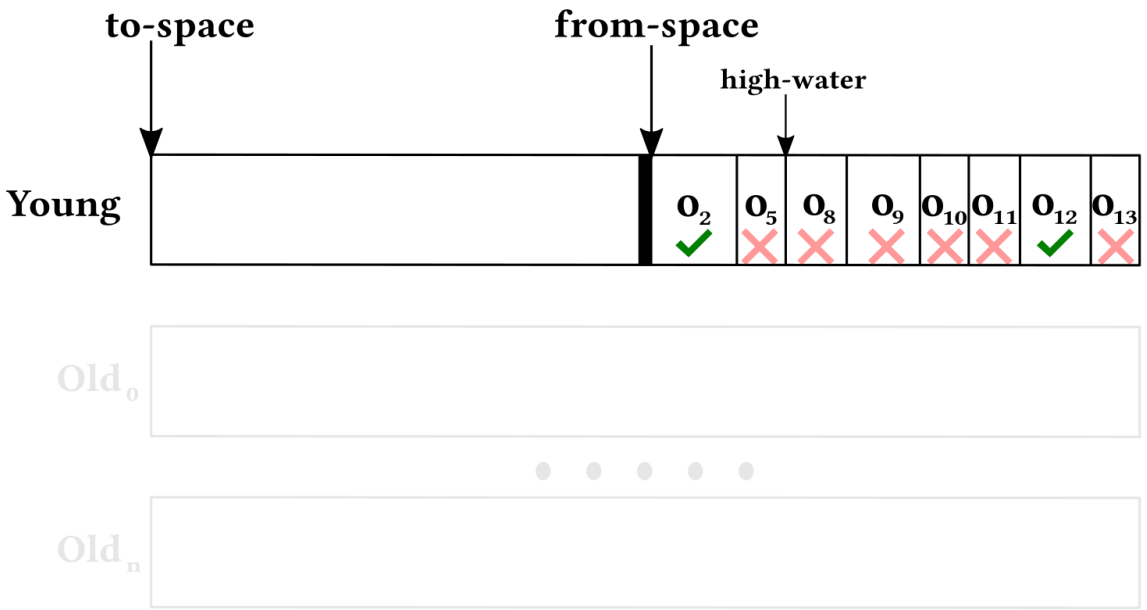
Garbage Collection (GC): The Good

- Modern GCs are *generational*.
- Applications exhibit high allocation rates and objects die young
  - Application constructs lots of ephemeral data.
- Long-lived objects survive multiple minor GCs



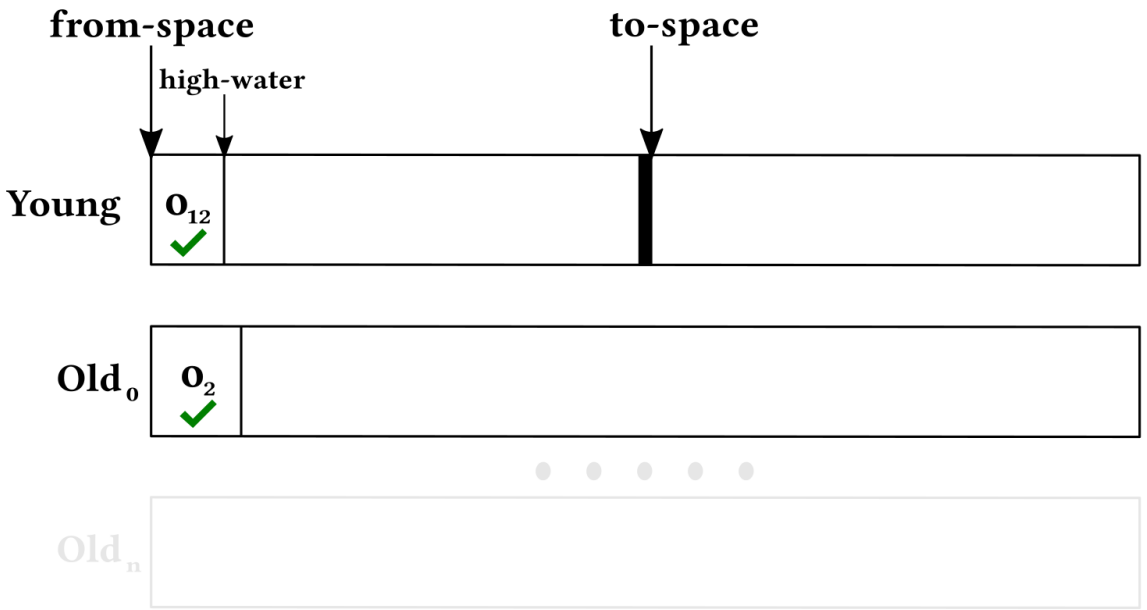
Garbage Collection (GC): The Good

- Modern GCs are *generational*.
- Applications exhibit high allocation rates and objects die young
  - Application constructs lots of ephemeral data.
- Long-lived objects survive multiple minor GCs
  - Application saves small to moderates amount of enduring data.



Garbage Collection (GC): The Good

- Modern GCs are *generational*.
- Applications exhibit high allocation rates and objects die young
  - Application constructs lots of ephemeral data.
- Long-lived objects survive multiple minor GCs
  - Application saves small to moderates amount of enduring data.
- All is right with the world.



Garbage Collection: Motivating Custom Allocators

- Performance is poor on non-generational and specialized workloads.

Garbage Collection: Motivating Custom Allocators

- Performance is poor on **non-generational** and specialized workloads.
  - **Way 1**: Object sizes & layout.
  - **Way 2**: Object lifetimes & allocation.
  - **Way 3**: Locality and access patterns.

Way 1: Object Sizes in a Streaming Science Workload

Tufts UNIVERSITY

- Homogenous object sizes: memory footprint per object limits window size and thus scientific algorithm accuracy.

Way 1: Object Sizes in a Streaming Science Workload

- Homogenous object sizes: memory footprint per object limits window size and thus scientific algorithm accuracy.
  - E.g. *k*-means algorithm accuracy correlated with available memory, and inversely correlated with runtime.

k-means: `http://www.cs.columbia.edu/~rjaiswal/ajmNIPS09.pdf`

Way 1: Object Sizes in a Streaming Science Workload

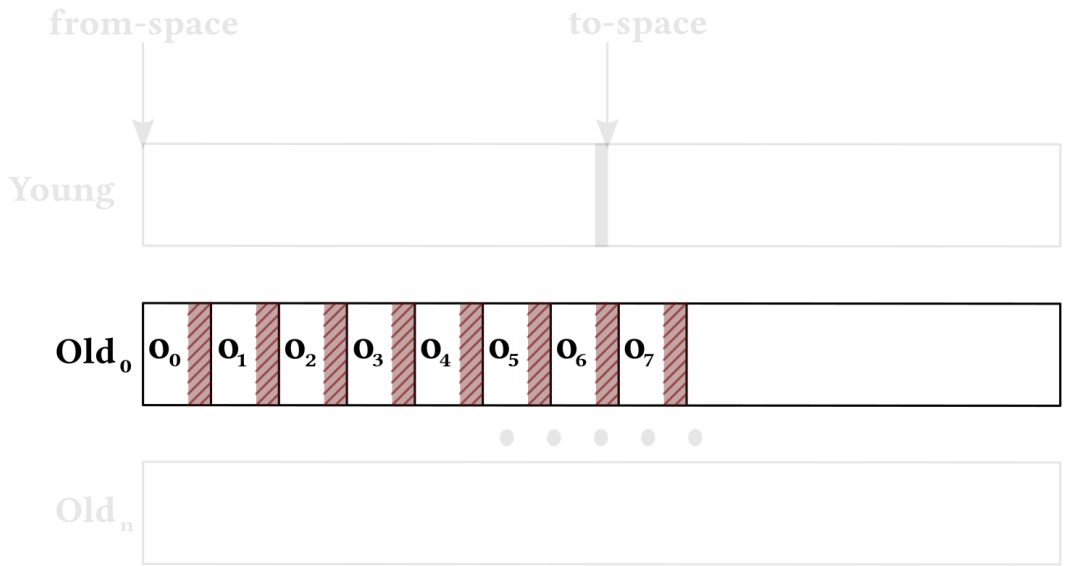
- Homogenous object sizes: memory footprint per object limits window size and thus scientific algorithm accuracy.
  - E.g. *k*-means algorithm accuracy correlated with available memory, and inversely correlated with runtime.
  - Same for approximating entropy of network packets.

k-means: `http://www.cs.columbia.edu/~rjaiswal/ajmNIPS09.pdf`

Entropy: `https://users.ece.cmu.edu/~vsekar/papers/fp013-lall.pdf`

Way 1: Object Sizes in a Streaming Science Workload

- Homogenous object sizes: memory footprint per object limits window size and thus scientific algorithm accuracy.
  - E.g. *k*-means algorithm accuracy correlated with available memory, and inversely correlated with runtime.
  - Same for approximating entropy of network packets.



k-means: `http://www.cs.columbia.edu/~rjaiswal/ajmNIPS09.pdf`

Entropy: `https://users.ece.cmu.edu/~vsekar/papers/fp013-lall.pdf`

Way 1: Object Sizes in a Streaming Science Workload

Tufts UNIVERSITY

```
1   type Mac = [u8; 6];
2   type Data = [u8; 13];
3   pub struct Msg { ptr: Addr, }
4   impl Msg {
5     const SIZE : usize = Self::ID_SIZE + Self::POWER_SIZE
6       + Self::MAC_SIZE + Self::EPOCH_SIZE + Self::DATA_SIZE;
7     const ID_OFFSET : usize = 0;
8     const ID_SIZE : usize = size_of::<u32>();
9     const EPOCH_OFFSET : usize = Self::ID_OFFSET + Self::ID_SIZE;
10    const EPOCH_SIZE : usize = size_of::<u64>();
11    const MAC_OFFSET : usize = Self::EPOCH_OFFSET + Self::EPOCH_SIZE;
12    const MAC_SIZE : usize = size_of::<Mac>();
13    const POWER_OFFSET : usize = Self::MAC_OFFSET + Self::MAC_SIZE;
14    const POWER_SIZE : usize = size_of::<u8>();
15    const DATA_OFFSET : usize = Self::POWER_OFFSET + Self::POWER_SIZE;
16    const DATA_SIZE : usize = size_of::<Data>();
17
18    pub fn get_id(&self) -> u32 { unsafe { *(self.ptr.add(Self::ID_OFFSET) as *mut u32) } }
19    pub fn get_epoch(&self) -> u64 { unsafe { *(self.ptr.add(Self::EPOCH_OFFSET) as *mut u64) } }
20    pub fn get_mac(&self) -> Mac { unsafe { *(self.ptr.add(Self::MAC_OFFSET) as *mut Mac) } }
21    pub fn get_power(&self) -> i8 { unsafe { *(self.ptr.add(Self::POWER_OFFSET) as *mut i8) } }
22    pub fn get_data(&self) -> Data { unsafe { *(self.ptr.add(Self::DATA_OFFSET) as *mut Data) } }
23    pub fn set_id(&self, v: u32) { unsafe { *(self.ptr.add(Self::ID_OFFSET) as *mut u32) = v } }
24    pub fn set_epoch(&self, v: u64) { unsafe { *(self.ptr.add(Self::EPOCH_OFFSET) as *mut u64) = v } }
25    pub fn set_mac(&self, v: Mac) { unsafe { *(self.ptr.add(Self::MAC_OFFSET) as *mut Mac) = v } }
26    pub fn set_power(&self, v: i8) { unsafe { *(self.ptr.add(Self::POWER_OFFSET) as *mut i8) = v } }
27    pub fn set_data(&self, v: Data) { unsafe { *(self.ptr.add(Self::DATA_OFFSET) as *mut Data) = v } }
28  }
```

Way 1: Object Sizes in a Streaming Science Workload

Tufts UNIVERSITY

```
1   type Mac = [u8; 6];
2   type Data = [u8; 13];
3   pub struct Msg { ptr: Addr, }
4   impl Msg {
5     const SIZE : usize = Self::ID_SIZE + Self::POWER_SIZE
6       + Self::MAC_SIZE + Self::EPOCH_SIZE + Self::DATA_SIZE;
7     const ID_OFFSET : usize = 0;
8     const ID_SIZE : usize = size_of::<u32>();
9     const EPOCH_OFFSET : usize = Self::ID_OFFSET + Self::ID_SIZE;
10    const EPOCH_SIZE : usize = size_of::<u64>();
11    const MAC_OFFSET : usize = Self::EPOCH_OFFSET + Self::EPOCH_SIZE;
12    const MAC_SIZE : usize = size_of::<Mac>();
13    const POWER_OFFSET : usize = Self::MAC_OFFSET + Self::MAC_SIZE;
14    const POWER_SIZE : usize = size_of::<u8>();
15    const DATA_OFFSET : usize = Self::POWER_OFFSET + Self::POWER_SIZE;
16    const DATA_SIZE : usize = size_of::<Data>();
17
18    pub fn get_id(&self) -> u32 { unsafe { *(self.ptr.add(Self::ID_OFFSET) as *mut u32) } }
19    pub fn get_epoch(&self) -> u64 { unsafe { *(self.ptr.add(Self::EPOCH_OFFSET) as *mut u64) } }
20    pub fn get_mac(&self) -> Mac { unsafe { *(self.ptr.add(Self::MAC_OFFSET) as *mut Mac) } }
21    pub fn get_power(&self) -> i8 { unsafe { *(self.ptr.add(Self::POWER_OFFSET) as *mut i8) } }
22    pub fn get_data(&self) -> Data { unsafe { *(self.ptr.add(Self::DATA_OFFSET) as *mut Data) } }
23    pub fn set_id(&self, v: u32) { unsafe { *(self.ptr.add(Self::ID_OFFSET) as *mut u32) = v } }
24    pub fn set_epoch(&self, v: u64) { unsafe { *(self.ptr.add(Self::EPOCH_OFFSET) as *mut u64) = v } }
25    pub fn set_mac(&self, v: Mac) { unsafe { *(self.ptr.add(Self::MAC_OFFSET) as *mut Mac) = v } }
26    pub fn set_power(&self, v: i8) { unsafe { *(self.ptr.add(Self::POWER_OFFSET) as *mut i8) = v } }
27    pub fn set_data(&self, v: Data) { unsafe { *(self.ptr.add(Self::DATA_OFFSET) as *mut Data) = v } }
28  }
```

Way 1: Object Sizes in a Streaming Science Workload

Tufts UNIVERSITY

```
1   type Mac = [u8; 6];
2   type Data = [u8; 13];
3   pub struct Msg { ptr: Addr, }
4   impl Msg {
5     const SIZE : usize = Self::ID_SIZE + Self::POWER_SIZE
6       + Self::MAC_SIZE + Self::EPOCH_SIZE + Self::DATA_SIZE;
7     const ID_OFFSET : usize = 0;
8     const ID_SIZE : usize = size_of::<u32>();
9     const EPOCH_OFFSET : usize = Self::ID_OFFSET + Self::ID_SIZE;
10    const EPOCH_SIZE : usize = size_of::<u64>();
11    const MAC_OFFSET : usize = Self::EPOCH_OFFSET + Self::EPOCH_SIZE;
12    const MAC_SIZE : usize = size_of::<Mac>();
13    const POWER_OFFSET : usize = Self::MAC_OFFSET + Self::MAC_SIZE;
14    const POWER_SIZE : usize = size_of::<u8>();
15    const DATA_OFFSET : usize = Self::POWER_OFFSET + Self::POWER_SIZE;
16    const DATA_SIZE : usize = size_of::<Data>();
17
18    pub fn get_id(&self) -> u32 { unsafe { *(self.ptr.add(Self::ID_OFFSET) as *mut u32) } }
19    pub fn get_epoch(&self) -> u64 { unsafe { *(self.ptr.add(Self::EPOCH_OFFSET) as *mut u64) } }
20    pub fn get_mac(&self) -> Mac { unsafe { *(self.ptr.add(Self::MAC_OFFSET) as *mut Mac) } }
21    pub fn get_power(&self) -> i8 { unsafe { *(self.ptr.add(Self::POWER_OFFSET) as *mut i8) } }
22    pub fn get_data(&self) -> Data { unsafe { *(self.ptr.add(Self::DATA_OFFSET) as *mut Data) } }
23    pub fn set_id(&self, v: u32) { unsafe { *(self.ptr.add(Self::ID_OFFSET) as *mut u32) = v } }
24    pub fn set_epoch(&self, v: u64) { unsafe { *(self.ptr.add(Self::EPOCH_OFFSET) as *mut u64) = v } }
25    pub fn set_mac(&self, v: Mac) { unsafe { *(self.ptr.add(Self::MAC_OFFSET) as *mut Mac) = v } }
26    pub fn set_power(&self, v: i8) { unsafe { *(self.ptr.add(Self::POWER_OFFSET) as *mut i8) = v } }
27    pub fn set_data(&self, v: Data) { unsafe { *(self.ptr.add(Self::DATA_OFFSET) as *mut Data) = v } }
28  }
```

Way 1: Object Sizes in a Streaming Science Workload

Tufts UNIVERSITY

```
1   type Mac = [u8; 6];
2   type Data = [u8; 13];
3   pub struct Msg { ptr: Addr, }
4   impl Msg {
5     const SIZE : usize = Self::ID_SIZE + Self::POWER_SIZE
6       + Self::MAC_SIZE + Self::EPOCH_SIZE + Self::DATA_SIZE;
7     const ID_OFFSET : usize = 0;
8     const ID_SIZE : usize = size_of::<u32>();
9     const EPOCH_OFFSET : usize = Self::ID_OFFSET + Self::ID_SIZE;
10    const EPOCH_SIZE : usize = size_of::<u64>();
11    const MAC_OFFSET : usize = Self::EPOCH_OFFSET + Self::EPOCH_SIZE;
12    const MAC_SIZE : usize = size_of::<Mac>();
13    const POWER_OFFSET : usize = Self::MAC_OFFSET + Self::MAC_SIZE;
14    const POWER_SIZE : usize = size_of::<u8>();
15    const DATA_OFFSET : usize = Self::POWER_OFFSET + Self::POWER_SIZE;
16    const DATA_SIZE : usize = size_of::<Data>
17
18    pub fn get_id(&self) -> u32 { unsafe { *(self.ptr.add(Self::ID_OFFSET) as *mut u32) } }
19    pub fn get_epoch(&self) -> u64 { unsafe { *(self.ptr.add(Self::EPOCH_OFFSET) as *mut u64) } }
20    pub fn get_mac(&self) -> Mac { unsafe { *(self.ptr.add(Self::MAC_OFFSET) as *mut Mac) } }
21    pub fn get_power(&self) -> i8 { unsafe { *(self.ptr.add(Self::POWER_OFFSET) as *mut i8) } }
22    pub fn get_data(&self) -> Data { unsafe { *(self.ptr.add(Self::DATA_OFFSET) as *mut Data) } }
23    pub fn set_id(&self, v: u32) { unsafe { *(self.ptr.add(Self::ID_OFFSET) as *mut u32) = v } }
24    pub fn set_epoch(&self, v: u64) { unsafe { *(self.ptr.add(Self::EPOCH_OFFSET) as *mut u64) = v } }
25    pub fn set_mac(&self, v: Mac) { unsafe { *(self.ptr.add(Self::MAC_OFFSET) as *mut Mac) = v } }
26    pub fn set_power(&self, v: i8) { unsafe { *(self.ptr.add(Self::POWER_OFFSET) as *mut i8) = v } }
27    pub fn set_data(&self, v: Data) { unsafe { *(self.ptr.add(Self::DATA_OFFSET) as *mut Data) = v } }
28  }
```

Way 1: Object Sizes in JVM Runtime Code


```
94    int SCALAR_HEADER_SIZE = JAVA_HEADER_BYTES + OTHER_HEADER_BYTES;
95    int ARRAY_HEADER_SIZE = SCALAR_HEADER_SIZE + ARRAY_LENGTH_BYTES;
96    /** offset of object reference from the lowest memory word */
97    Offset TIB_OFFSET = JAVA_HEADER_OFFSET;
98    Offset STATUS_OFFSET = TIB_OFFSET.plus(STATUS_BYTES);
99    Offset AVAILABLE_BITS_OFFSET =
100     VM.LittleEndian ?
101         STATUS_OFFSET
102       : STATUS_OFFSET.plus(STATUS_BYTES - 1);
103   int HASH_CODE_SHIFT = 2;
104   Word HASH_CODE_MASK =
105     Word.one()
106       .lsh(10)
107       .minus(Word.one())
108       .lsh(HASH_CODE_SHIFT);
109   /** How many bits are allocated to a thin lock? */
110   int NUM_THIN_LOCK_BITS = ADDRESS_BASED_HASHING ? 22 : 20;
111   /** How many bits to shift to get the thin lock? */
112   int THIN_LOCK_SHIFT = ADDRESS_BASED_HASHING ? 10 : 12;
```

Way 1: JikesRVM Layout Bug Fix

JikesRVM / **JikesRVM**

Fix incorrect offset calculation for status bytes.

New issue

Merged **erik-brangs** merged 1 commit into `JikesRVM:master` from `cronburg:tib` on May 14, 2018

Conversation 3 | Commits 1 | Checks 0 | Files changed 1 | +2 −1

Changes from **all commits** | **File filter...** | **Jump to...**

rvm/src/org/jikesrvm/objectmodel/JavaHeader.java

```
29  29    import static org.jikesrvm.objectmodel.JavaHeaderConstants.NUM_AVAILABLE_BITS;
30  30    import static org.jikesrvm.objectmodel.JavaHeaderConstants.OTHER_HEADER_BYTES;
31  31    import static org.jikesrvm.objectmodel.JavaHeaderConstants.STATUS_BYTES;
    32  +  import static org.jikesrvm.objectmodel.JavaHeaderConstants.TIB_BYTES;
32  33    import static org.jikesrvm.objectmodel.MiscHeader.REQUESTED_BITS;
33  34    import static org.jikesrvm.runtime.JavaSizeConstants.BYTES_IN_INT;
34  35    import static org.jikesrvm.runtime.UnboxedSizeConstants.LOG_BYTES_IN_ADDRESS;
95  96      /** offset of object reference from the lowest memory word */
96  97      public static final int OBJECT_REF_OFFSET = ARRAY_HEADER_SIZE;  // from start to ref
97  98      protected static final Offset TIB_OFFSET = JAVA_HEADER_OFFSET;
98      -   protected static final Offset STATUS_OFFSET = TIB_OFFSET.plus(STATUS_BYTES);
    99  +   protected static final Offset STATUS_OFFSET = TIB_OFFSET.plus(TIB_BYTES);
99  100     protected static final Offset AVAILABLE_BITS_OFFSET =
100 101         VM.LittleEndian ? (STATUS_OFFSET) : (STATUS_OFFSET.plus(STATUS_BYTES - 1));
```

`https://github.com/JikesRVM/JikesRVM/pull/18/files`

Way 1: "Eat Your Spinach"

Tufts UNIVERSITY

- Static type system = Eat your spinach!
- Talk at SPLASH-I, about Rust, last year in Boston:

  `https://2018.splashcon.org/details/splash-2018-SPLASH-I/5/Rust-Reach-Further`

- Improves code structure and legibility.

https://www.flickr.com/photos/salim/8594532469

Way 1: Mechanical Data Formats

Tufts UNIVERSITY

```
type Mac = [u8; 6];
type Data = [u8; 13];
pub struct Msg { ptr: Addr, }
impl Msg {
  const SIZE : usize = Self::ID_SIZE + Self::POWER_SIZE
    + Self::MAC_SIZE + Self::EPOCH_SIZE + Self::DATA_SIZE;
  const ID_OFFSET : usize = 0;
  const ID_SIZE : usize = size_of::<u32>();
  const EPOCH_OFFSET : usize = Self::ID_OFFSET + Self::ID_SIZE;
  const EPOCH_SIZE : usize = size_of::<u64>();
  const MAC_OFFSET : usize = Self::EPOCH_OFFSET + Self::EPOCH_SIZE;
  const MAC_SIZE : usize = size_of::<Mac>();
  const POWER_OFFSET : usize = Self::MAC_OFFSET + Self::MAC_SIZE;
  const POWER_SIZE : usize = size_of::<u8>();
  const DATA_OFFSET : usize = Self::POWER_OFFSET + Self::POWER_SIZE;
  const DATA_SIZE : usize = size_of::<Data>();

  pub fn get_id(&self) -> u32 { unsafe { *(self.ptr.add(Self::ID_OFFSET) as *mut u32) } }
  pub fn get_epoch(&self) -> u64 { unsafe { *(self.ptr.add(Self::EPOCH_OFFSET) as *mut u64) } }
  pub fn get_mac(&self) -> Mac { unsafe { *(self.ptr.add(Self::MAC_OFFSET) as *mut Mac) } }
  pub fn get_power(&self) -> i8 { unsafe { *(self.ptr.add(Self::POWER_OFFSET) as *mut i8) } }
  pub fn get_data(&self) -> Data { unsafe { *(self.ptr.add(Self::DATA_OFFSET) as *mut Data) } }
  pub fn set_id(&self, v: u32) { unsafe { *(self.ptr.add(Self::ID_OFFSET) as *mut u32) = v } }
  pub fn set_epoch(&self, v: u64) { unsafe { *(self.ptr.add(Self::EPOCH_OFFSET) as *mut u64) = v } }
  pub fn set_mac(&self, v: Mac) { unsafe { *(self.ptr.add(Self::MAC_OFFSET) as *mut Mac) = v } }
  pub fn set_power(&self, v: i8) { unsafe { *(self.ptr.add(Self::POWER_OFFSET) as *mut i8) = v } }
  pub fn set_data(&self, v: Data) { unsafe { *(self.ptr.add(Self::DATA_OFFSET) as *mut Data) = v } }
}
```

Way 1: Floorplan Specification


```
1  Msg -> seq {
2    id : 4 bytes,    // Sensor identifier
3    epoch: 8 bytes, // Unix epoch time
4    Mac,             // Bluetooth MAC address
5    power: 1 bytes, // Received Signal Strength Indication
6    data: 13 bytes, // Portion of BLE packet data
7  }
8  Mac -> 6 bytes
```

Way 2: Object Lifetimes in a Streaming Science Workload

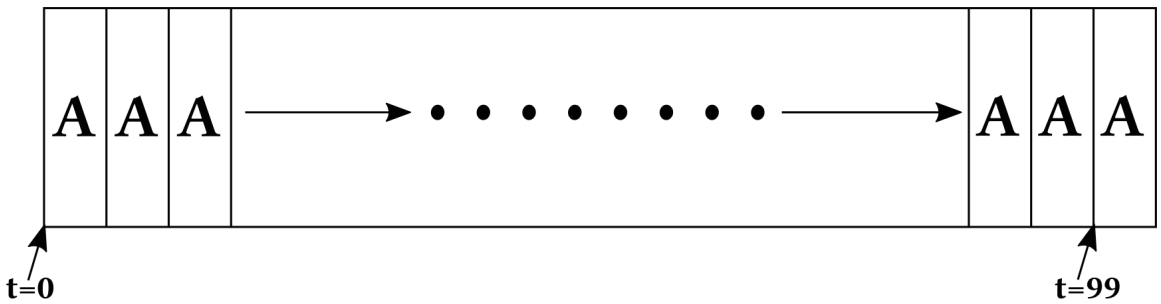
Tufts UNIVERSITY

- Temporal-based sliding window of data: non-generational workload.
  - Window Slide Policy[1] traditionally defined by *n* most recent data entries.

[1] Window sliding policy: `https://ieeexplore.ieee.org/document/8456588`

Way 2: Object Lifetimes in a Streaming Science Workload

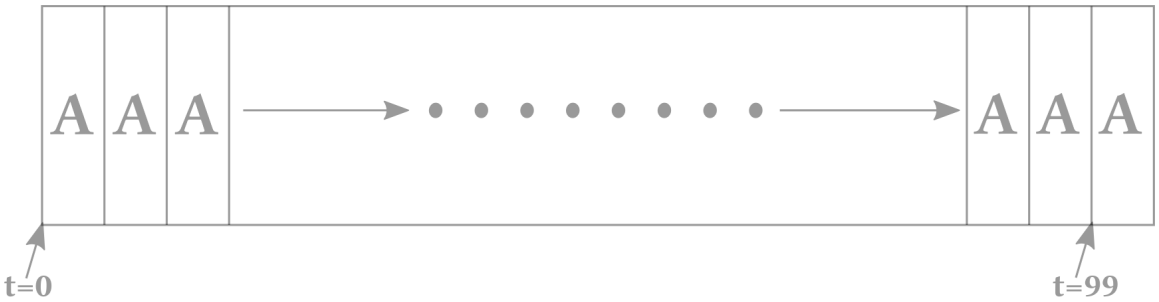
- Temporal-based sliding window of data: non-generational workload.
  - Window Slide Policy[1] traditionally defined by *n* most recent data entries.



[1] Window sliding policy: `https://ieeexplore.ieee.org/document/8456588`

Way 2: Object Lifetimes in a Streaming Science Workload

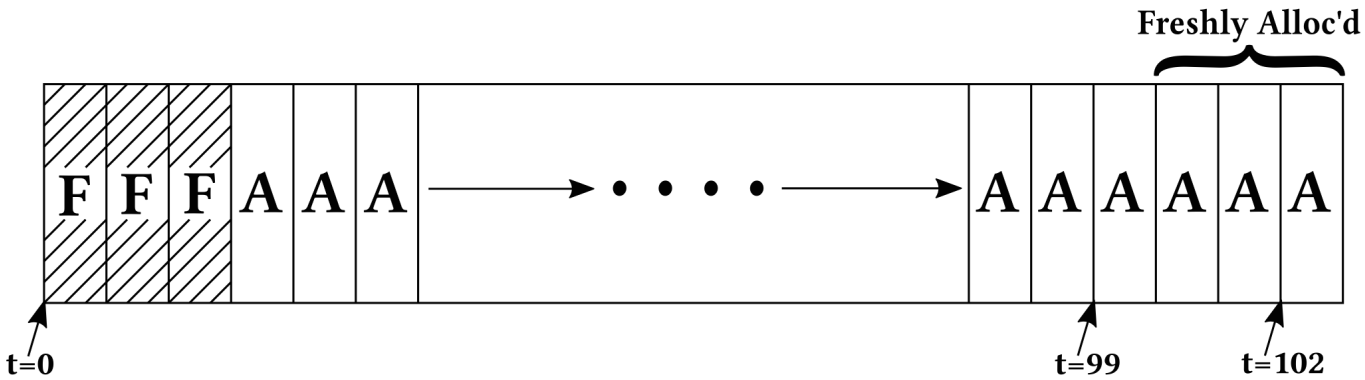
- Temporal-based sliding window of data: non-generational workload.
  - Window Slide Policy[1] traditionally defined by *n* most recent data entries.
  - Evict old entries as new ones come in.



[1] Window sliding policy: `https://ieeexplore.ieee.org/document/8456588`

Way 2: Object Lifetimes in a Streaming Science Workload

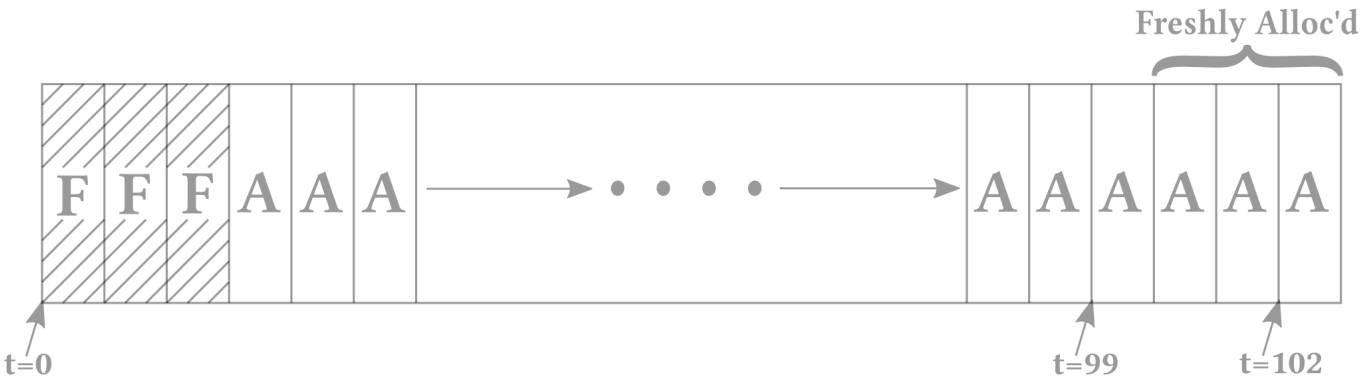
- Temporal-based sliding window of data: non-generational workload.
  - Window Slide Policy[1] traditionally defined by *n* most recent data entries.
  - Evict old entries as new ones come in.



[1] Window sliding policy: `https://ieeexplore.ieee.org/document/8456588`

Way 2: Object Lifetimes in a Streaming Science Workload

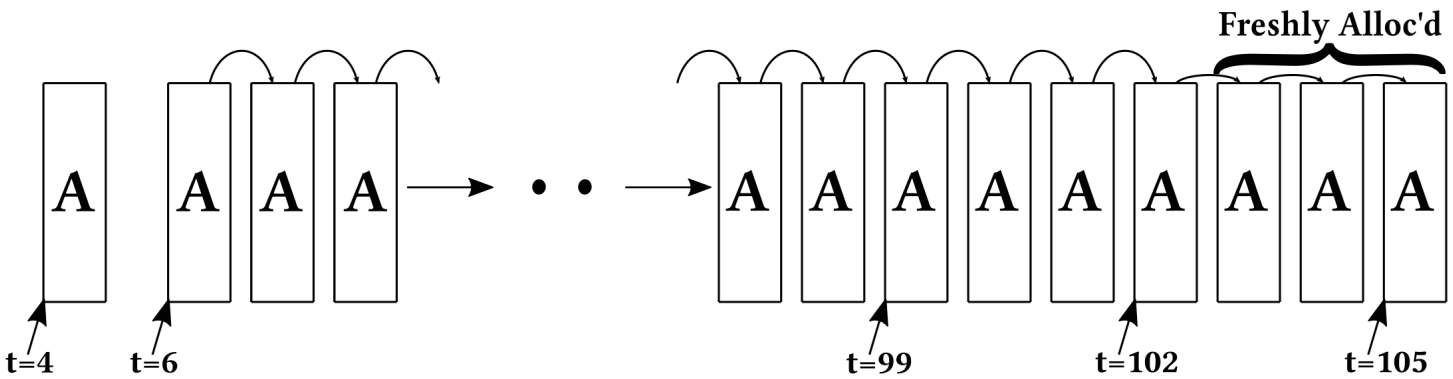
- Temporal-based sliding window of data: non-generational workload.
  - Window Slide Policy[1] traditionally defined by *n* most recent data entries.
  - Evict old entries as new ones come in.
  - Naiive FIFO linked-list experiences O(n) per GC



[1] Window sliding policy: `https://ieeexplore.ieee.org/document/8456588`

Way 2: Object Lifetimes in a Streaming Science Workload

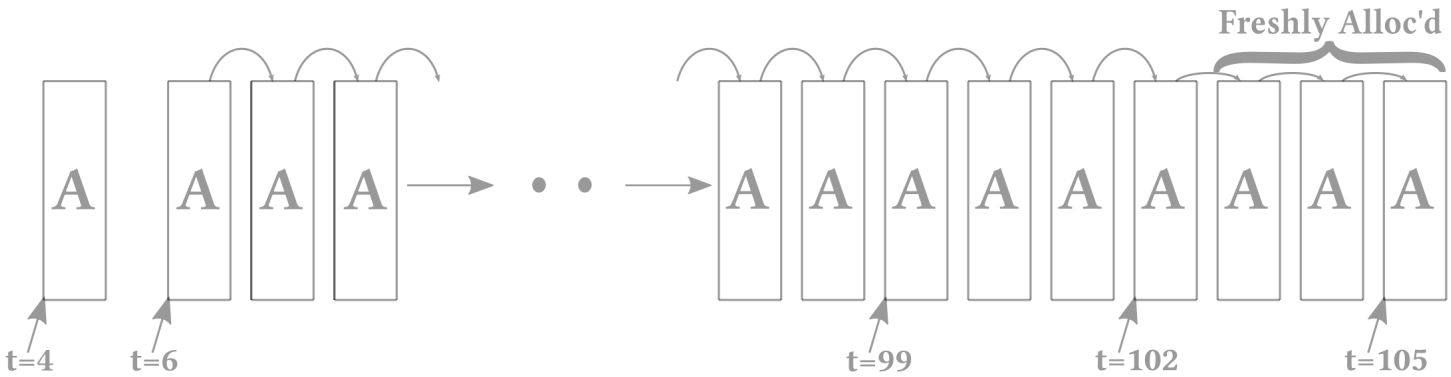
- Temporal-based sliding window of data: non-generational workload.
  - Window Slide Policy[1] traditionally defined by *n* most recent data entries.
  - Evict old entries as new ones come in.
  - Naiive FIFO linked-list experiences O(n) per GC



[1] Window sliding policy: `https://ieeexplore.ieee.org/document/8456588`

Way 2: Object Lifetimes in a Streaming Science Workload

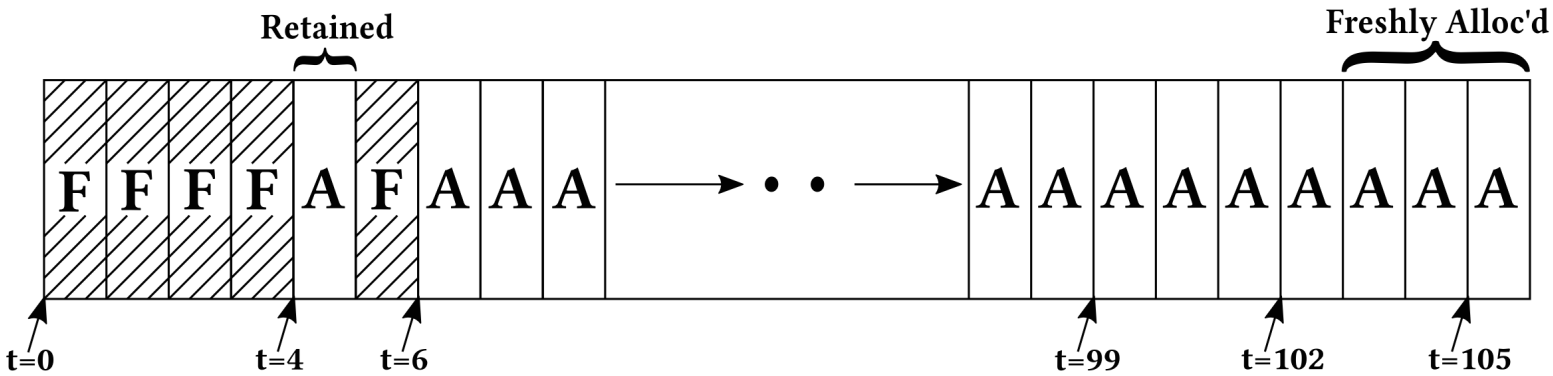
- Temporal-based sliding window of data: non-generational workload.
  - Window Slide Policy[1] traditionally defined by *n* most recent data entries.
  - Evict old entries as new ones come in.
  - Naiive FIFO linked-list experiences O(n) per GC
  - Want GC to act like ring buffer in common case (O(1) allocation and eviction), but still support arbitrary eviction policies.



[1] Window sliding policy: `https://ieeexplore.ieee.org/document/8456588`

Way 2: Object Lifetimes in a Streaming Science Workload

- Temporal-based sliding window of data: non-generational workload.
  - Window Slide Policy[1] traditionally defined by *n* most recent data entries.
  - Evict old entries as new ones come in.
  - Naiive FIFO linked-list experiences O(n) per GC
  - Want GC to act like ring buffer in common case (O(1) allocation and eviction), but still support arbitrary eviction policies.



[1] Window sliding policy: `https://ieeexplore.ieee.org/document/8456588`

Way 2: Object Lifetimes in a Block Allocator

Tufts UNIVERSITY

```
type Addr = *mut u8;
pub fn init(src: &String, heap_sz_mb: usize) -> Result<Ctrl, Error> {
  let file = OpenOptions::new().read(true).write(true).open(src)?;
  let mut mmap_f = unsafe { MmapMut::map_mut(&file)? };
  let fp = mmap_f.as_mut_ptr();

  let heap_sz = heap_sz_mb * BYTES_IN_MB;
  assert!(heap_sz > Block::SIZE, "{}MB not enough for 1 block.", heap_sz_mb);

  let mut mmap_heap = MmapMut::map_anon(heap_sz + Block::SIZE)?;
  let hp : Addr = (((mmap_heap.as_mut_ptr() as usize) >> Block::LOG_SIZE) + 1) << Block::LOG_SIZE;
  let mx = unsafe { hp.add(heap_sz) };

  let mut bs = vec![];
  let first = Block::new(hp);
  let mut curr = first;
  loop {
    let base = unsafe { curr.base.add(Block::SIZE) };
    if unsafe { base.add(Block::SIZE) } > mx { break; }
    curr = Block::new(unsafe { curr.base.add(Block::SIZE) });
    bs.push(curr);
  }

  Ok(Ctrl {
    curr: first,
    next_msg: 0x0 as Addr,
    free_bs: bs,
    used_bs: vec![],
    _fp: mmap_f, fp: fp,
    fp_limit: unsafe { fp.add(file.metadata().unwrap().len() as usize) },
    _hp: mmap_heap, hp: hp, allocd: 0 })
}
```

Way 2: Object Lifetimes in a Block Allocator


```
type Addr = *mut u8;
pub fn init(src: &String, heap_sz_mb: usize) -> Result<Ctrl, Error> {
  let file = OpenOptions::new().read(true).write(true).open(src)?;
  let mut mmap_f = unsafe { MmapMut::map_mut(&file)? };
  let fp = mmap_f.as_mut_ptr();

  let heap_sz = heap_sz_mb * BYTES_IN_MB;
  assert!(heap_sz > Block::SIZE, "{}MB not enough for 1 block.", heap_sz_mb);

  let mut mmap_heap = MmapMut::map_anon(heap_sz + Block::SIZE)?;
  let hp : Addr = (((mmap_heap.as_mut_ptr() as usize) >> Block::LOG_SIZE) + 1) << Block::LOG_SIZE;
  let mx = unsafe { hp.add(heap_sz) };

  let mut bs = vec![];
  let first = Block::new(hp);
  let mut curr = first;
  loop {
    let base = unsafe { curr.base.add(Block::SIZE) };
    if unsafe { base.add(Block::SIZE) } > mx { break; }
    curr = Block::new(unsafe { curr.base.add(Block::SIZE) });
    bs.push(curr);
  }

  Ok(Ctrl {
    curr: first,
    next_msg: 0x0 as Addr,
    free_bs: bs,
    used_bs: vec![],
    _fp: mmap_f, fp: fp,
    fp_limit: unsafe { fp.add(file.metadata().unwrap().len() as usize) },
    _hp: mmap_heap, hp: hp, allocd: 0 })
}
```

Way 2: Object Lifetimes in a Block Allocator

Tufts UNIVERSITY

```
type Addr = *mut u8;
pub fn init(src: &String, heap_sz_mb: usize) -> Result<Ctrl, Error> {
  let file = OpenOptions::new().read(true).write(true).open(src)?;
  let mut mmap_f = unsafe { MmapMut::map_mut(&file)? };
  let fp = mmap_f.as_mut_ptr();

  let heap_sz = heap_sz_mb * BYTES_IN_MB;
  assert!(heap_sz > Block::SIZE, "{}MB not enough for 1 block.", heap_sz_mb);

  let mut mmap_heap = MmapMut::map_anon(heap_sz + Block::SIZE)?;
  let hp : Addr = (((mmap_heap.as_mut_ptr() as usize) >> Block::LOG_SIZE) + 1) << Block::LOG_SIZE;
  let mx = unsafe { hp.add(heap_sz) };

  let mut bs = vec![];
  let first = Block::new(hp);
  let mut curr = first;
  loop {
    let base = unsafe { curr.base.add(Block::SIZE) };
    if unsafe { base.add(Block::SIZE) } > mx { break; }
    curr = Block::new(unsafe { curr.base.add(Block::SIZE) });
    bs.push(curr);
  }

  Ok(Ctrl {
    curr: first,
    next_msg: 0x0 as Addr,
    free_bs: bs,
    used_bs: vec![],
    _fp: mmap_f, fp: fp,
    fp_limit: unsafe { fp.add(file.metadata().unwrap().len() as usize) },
    _hp: mmap_heap, hp: hp, allocd: 0 })
}
```

Way 2: Object Lifetimes in a Block Allocator

Tufts UNIVERSITY

```
type Addr = *mut u8;
pub fn init(src: &String, heap_sz_mb: usize) -> Result<Ctrl, Error> {
  let file = OpenOptions::new().read(true).write(true).open(src)?;
  let mut mmap_f = unsafe { MmapMut::map_mut(&file)? };
  let fp = mmap_f.as_mut_ptr();

  let heap_sz = heap_sz_mb * BYTES_IN_MB;
  assert!(heap_sz > Block::SIZE, "{}MB not enough for 1 block.", heap_sz_mb);

  let mut mmap_heap = MmapMut::map_anon(heap_sz + Block::SIZE)?;
  let hp : Addr = (((mmap_heap.as_mut_ptr() as usize) >> Block::LOG_SIZE) + 1) << Block::LOG_SIZE;
  let mx = unsafe { hp.add(heap_sz) };

  let mut bs = vec![];
  let first = Block::new(hp);
  let mut curr = first;
  loop {
    let base = unsafe { curr.base.add(Block::SIZE) };
    if unsafe { base.add(Block::SIZE) } > mx { break; }
    curr = Block::new(unsafe { curr.base.add(Block::SIZE) });
    bs.push(curr);
  }

  Ok(Ctrl {
    curr: first,
    next_msg: 0x0 as Addr,
    free_bs: bs,
    used_bs: vec![],
    _fp: mmap_f, fp: fp,
    fp_limit: unsafe { fp.add(file.metadata().unwrap().len() as usize) },
    _hp: mmap_heap, hp: hp, allocd: 0 })
}
```

Way 2: Object Lifetimes in a Block Allocator

Tufts UNIVERSITY

```
type Addr = *mut u8;
pub fn init(src: &String, heap_sz_mb: usize) -> Result<Ctrl, Error> {
  let file = OpenOptions::new().read(true).write(true).open(src)?;
  let mut mmap_f = unsafe { MmapMut::map_mut(&file)? };
  let fp = mmap_f.as_mut_ptr();

  let heap_sz = heap_sz_mb * BYTES_IN_MB;
  assert!(heap_sz > Block::SIZE, "{}MB not enough for 1 block.", heap_sz_mb);

  let mut mmap_heap = MmapMut::map_anon(heap_sz + Block::SIZE)?;
  let hp : Addr = (((mmap_heap.as_mut_ptr() as usize) >> Block::LOG_SIZE) + 1) << Block::LOG_SIZE;
  let mx = unsafe { hp.add(heap_sz) };

  let mut bs = vec![];
  let first = Block::new(hp);
  let mut curr = first;
  loop {
    let base = unsafe { curr.base.add(Block::SIZE) };
    if unsafe { base.add(Block::SIZE) } > mx { break; }
    curr = Block::new(unsafe { curr.base.add(Block::SIZE) });
    bs.push(curr);
  }

  Ok(Ctrl {
    curr: first,
    next_msg: 0x0 as Addr,
    free_bs: bs,
    used_bs: vec![],
    _fp: mmap_f, fp: fp,
    fp_limit: unsafe { fp.add(file.metadata().unwrap().len() as usize) },
    _hp: mmap_heap, hp: hp, allocd: 0 })
}
```

Way 2: Object Lifetimes in a Block Allocator


```
let mut mmap_heap = MmapMut::map_anon(heap_sz + Block::SIZE)?;
let hp : Addr = (((mmap_heap.as_mut_ptr() as usize) >> Block::LOG_SIZE) + 1) << Block::LOG_SIZE;
let mx = unsafe { hp.add(heap_sz) };

let mut bs = vec![];
let first = Block::new(hp);
let mut curr = first;
loop {
  let base = unsafe { curr.base.add(Block::SIZE) };
  if unsafe { base.add(Block::SIZE) } > mx { break; }
  curr = Block::new(unsafe { curr.base.add(Block::SIZE) });
  bs.push(curr);
}
```

Way 2: Object Lifetimes in a Block Allocator

Tufts UNIVERSITY

Specification:

```
Heap -> # Block
Block @|2^21 bytes|@ -> # Msg
```

```
let mut mmap_heap = MmapMut::map_anon(heap_sz + Block::SIZE)?;
let hp : HeapAddr = HeapAddr::align_up_to_Block(mmap_heap.as_mut_ptr() as usize);
let mx : HeapEndAddr = hp.init_end(heap_sz);

let mut bs = vec![];
let first : BlockAddr = Block::new(hp);
let mut curr : BlockAddr = first;
loop {
  let base = curr.skip_to_next_Block();
  if (base.skip_to_next_Block().is_after_HeapEnd(mx)) { break; }
  curr = Block::new(curr.base.skip_to_next_Block());
  bs.push(curr);
}
```

Generated Functions

Way 2: Object Lifetimes in a Block Allocator

Tufts UNIVERSITY

Specification:

```
Heap -> # Block
Block @|2^21 bytes|@ -> # Msg
```

```
let mut mmap_heap = MmapMut::map_anon(heap_sz + Block::SIZE)?;
let hp : HeapAddr = HeapAddr::align_up_to_Block(mmap_heap.as_mut_ptr() as usize);
let mx : HeapEndAddr = hp.init_end(heap_sz);

let mut bs = vec![];
let first : BlockAddr = Block::new(hp);
let mut curr : BlockAddr = first;
loop {
  let base = curr.skip_to_next_Block();
  if (base.skip_to_next_Block().is_after_HeapEnd(mx)) { break; }
  curr = Block::new(curr.base.skip_to_next_Block());
  bs.push(curr);
}
```

Way 2: Object Lifetimes in a Block Allocator

Tufts UNIVERSITY

Specification:

```
Heap -> # Block
Block @|2^21 bytes|@ -> # Msg
```

```
let mut mmap_heap = MmapMut::map_anon(heap_sz + Block::SIZE)?;
let hp : HeapAddr = HeapAddr::align_up_to_Block(mmap_heap.as_mut_ptr() as usize);
let mx : HeapEndAddr = hp.init_end(heap_sz);

let mut bs = vec![];
let first : BlockAddr = Block::new(hp);
let mut curr : BlockAddr = first;
loop {
  let base = curr.skip_to_next_Block();
  if (base.skip_to_next_Block().is_after_HeapEnd(mx)) { break; }
  curr = Block::new(curr.base.skip_to_next_Block());
  bs.push(curr);
}
```

Way 2: Object Lifetimes in a Block Allocator

Tufts UNIVERSITY

Specification: Heap -> # Block

Block @|2^21 bytes|@ -> # Msg

```
let mut mmap_heap = MmapMut::map_anon(heap_sz + Block::SIZE)?;
let hp : HeapAddr = HeapAddr::align_up_to_Block(mmap_heap.as_mut_ptr() as usize);
let mx : HeapEndAddr = hp.init_end(heap_sz);

let mut bs = vec![];
let first : BlockAddr = Block::new(hp);
let mut curr : BlockAddr = first;
loop {
  let base = curr.skip_to_next_Block();
  if (base.skip_to_next_Block().is_after_HeapEnd(mx)) { break; }
  curr = Block::new(curr.base.skip_to_next_Block());
  bs.push(curr);
}
```

Way 2: Object Lifetimes in a Block Allocator

Tufts UNIVERSITY

Specification:

```
Heap -> # Block
Block @|2^21 bytes|@ -> # Msg
```

```
let mut mmap_heap = MmapMut::map_anon(heap_sz + Block::SIZE)?;
let hp : HeapAddr = HeapAddr::align_up_to_Block(mmap_heap.as_mut_ptr() as usize);
let mx : HeapEndAddr = hp.init_end(heap_sz);

let mut bs = vec![];
let first : BlockAddr = Block::new(hp);
let mut curr : BlockAddr = first;
loop {
  let base = curr.skip_to_next_Block();
  if (base.skip_to_next_Block().is_after_HeapEnd(mx)) { break; }
  curr = Block::new(curr.base.skip_to_next_Block());
  bs.push(curr);
}
```

Way 2: Object Lifetimes in a Block Allocator

Tufts UNIVERSITY

Specification: Heap -> # Block

Block @|2^21 bytes|@ -> # Msg

```
let mut mmap_heap = MmapMut::map_anon(heap_sz + Block::SIZE)?;
let hp : HeapAddr = HeapAddr::align_up_to_Block(mmap_heap.as_mut_ptr() as usize);
let mx : HeapEndAddr = hp.init_end(heap_sz);

let mut bs = vec![];
let first : BlockAddr = Block::new(hp);
let mut curr : BlockAddr = first;
loop {
  let base = curr.skip_to_next_Block();
  if is_after_HeapEnd(mx) { break; }
  curr = Block::new(curr.base.skip_to_next_Block());
  bs.push(curr);
}
```

Way 2: Object Lifetimes in a Block Allocator

Tufts UNIVERSITY

```
type Addr = *mut u8;
pub fn init(src: &String, heap_sz_mb: usize) -> Result<Ctrl, Error> {
  let file = OpenOptions::new().read(true).write(true).open(src)?;
  let mut mmap_f = unsafe { MmapMut::map_mut(&file)? };
  let fp = mmap_f.as_mut_ptr();

  let heap_sz = heap_sz_mb * BYTES_IN_MB;
  assert!(heap_sz > Block::SIZE, "{}MB not enough for 1 block.", heap_sz_mb);

  let mut mmap_heap = MmapMut::map_anon(heap_sz + Block::SIZE)?;
  let hp : Addr = (((mmap_heap.as_mut_ptr() as usize) >> Block::LOG_SIZE) + 1) << Block::LOG_SIZE;
  let mx = unsafe { hp.add(heap_sz) };

  let mut bs = vec![];
  let first = Block::new(hp);
  let mut curr = first;
  loop {
    let base = unsafe { curr.base.add(Block::SIZE) };
    if unsafe { base.add(Block::SIZE) } > mx { break; }
    curr = Block::new(unsafe { curr.base.add(Block::SIZE) });
    bs.push(curr);
  }

  Ok(Ctrl {
    curr: first,
    next_msg: 0x0 as Addr,
    free_bs: bs,
    used_bs: vec![],
    _fp: mmap_f, fp: fp,
    fp_limit: unsafe { fp.add(file.metadata().unwrap().len() as usize) },
    _hp: mmap_heap, hp: hp, allocd: 0 })
}
```

What We Do Currently

- Data formats: deserialization (copying) required with most GCs.
  - Frameworks like Apache Spark hide individual data entries from the GC.

What We Do Currently: Allocation in Apache Spark

Tufts UNIVERSITY

```java
/**A simple {@link MemoryAllocator} that can allocate up to 16GB using a JVM long primitive array. */
public class HeapMemoryAllocator implements MemoryAllocator {
  private final Map<Long, LinkedList<WeakReference<long[]>>> bufferPoolsBySize = new HashMap<>();
  public MemoryBlock allocate(long size) throws OutOfMemoryError {
    int numWords = (int) ((size + 7) / 8);
    long alignedSize = numWords * 8L;
    assert (alignedSize >= size);
    if (shouldPool(alignedSize)) {
      synchronized (this) {
        final LinkedList<WeakReference<long[]>> pool = bufferPoolsBySize.get(alignedSize);
        if (pool != null) {
          while (!pool.isEmpty()) {
            final WeakReference<long[]> arrayReference = pool.pop();
            final long[] array = arrayReference.get();
            if (array != null) {
              assert (array.length * 8L >= size);
              MemoryBlock memory = new MemoryBlock(array, Platform.LONG_ARRAY_OFFSET, size);
              if (MemoryAllocator.MEMORY_DEBUG_FILL_ENABLED) {
                memory.fill(MemoryAllocator.MEMORY_DEBUG_FILL_CLEAN_VALUE); }
              return memory; } }
          bufferPoolsBySize.remove(alignedSize); } } }
    long[] array = new long[numWords];
    MemoryBlock memory = new MemoryBlock(array, Platform.LONG_ARRAY_OFFSET, size);
    if (MemoryAllocator.MEMORY_DEBUG_FILL_ENABLED) {
      memory.fill(MemoryAllocator.MEMORY_DEBUG_FILL_CLEAN_VALUE); }
    return memory; }
```

https://github.com/apache/spark/blob/0b9ccd55c2986957863dcad3b44ce80403eecfa1/common/unsafe/src/main/java/

org/apache/spark/unsafe/memory/HeapMemoryAllocator.java#L48

What We Do Currently: Allocation in Apache Spark

Tufts UNIVERSITY

```
1   /**A simple {@link MemoryAllocator} that can allocate up to 16GB using a JVM long primitive array. */
2   public class HeapMemoryAllocator implements MemoryAllocator {
3     private final Map<Long, LinkedList<WeakReference<long[]>>> bufferPoolsBySize = new HashMap<>();
4     public MemoryBlock allocate(long size) throws OutOfMemoryError {
5       int numWords = (int) ((size + 7) / 8);
6       long alignedSize = numWords * 8L;
7       assert (alignedSize >= size);
8       if (shouldPool(alignedSize)) {
9         synchronized (this) {
10          final LinkedList<WeakReference<long[]>> pool = bufferPoolsBySize.get(alignedSize);
11          if (pool != null) {
12            while (!pool.isEmpty()) {
13              final WeakReference<long[]> arrayReference = pool.pop();
14              final long[] array = arrayReference.get();
15              if (array != null) {
16                assert (array.length * 8L >= size);
17                MemoryBlock memory = new MemoryBlock(array, Platform.LONG_ARRAY_OFFSET, size);
18                if (MemoryAllocator.MEMORY_DEBUG_FILL_ENABLED) {
19                  memory.fill(MemoryAllocator.MEMORY_DEBUG_FILL_CLEAN_VALUE); }
20                return memory; } }
21            bufferPoolsBySize.remove(alignedSize); } } }
22      long[] array = new long[numWords];
23      MemoryBlock memory = new MemoryBlock(array, Platform.LONG_ARRAY_OFFSET, size);
24      if (MemoryAllocator.MEMORY_DEBUG_FILL_ENABLED) {
25        memory.fill(MemoryAllocator.MEMORY_DEBUG_FILL_CLEAN_VALUE); }
26      return memory; }
```

https://github.com/apache/spark/blob/0b9ccd55c2986957863dcad3b44ce80403eecfa1/common/unsafe/src/main/java/org/apache/spark/unsafe/memory/HeapMemoryAllocator.java#L48

What We Do Currently

- Specialized formats: deserialization (copying) required with most GCs.
  - Frameworks like Apache Spark hide individual data entries from the GC.
  - Libraries like NumPy support memory-mapping out-of-core data.

What We Do Currently: NumPy's NPY File Format


```
346     def _wrap_header(header, version):
347         """
348         Takes a stringified header, and attaches the prefix and padding to it
349         """
350         import struct
351         assert version is not None
352         fmt, encoding = _header_size_info[version]
353         if not isinstance(header, bytes):  # always true on python 3
354             header = header.encode(encoding)
355         hlen = len(header) + 1
356         padlen = ARRAY_ALIGN - ((MAGIC_LEN + struct.calcsize(fmt) + hlen) % ARRAY_ALIGN)
357         try:
358             header_prefix = magic(*version) + struct.pack(fmt, hlen + padlen)
359         except struct.error:
360             msg = "Header length {} too big for version={}".format(hlen, version)
361             raise ValueError(msg)
362 
363         # Pad the header with spaces and a final newline such that the magic
364         # string, the header-length short and the header are aligned on a
365         # ARRAY_ALIGN byte boundary.  This supports memory mapping of dtypes
366         # aligned up to ARRAY_ALIGN on systems like Linux where mmap()
367         # offset must be page-aligned (i.e. the beginning of the file).
368         return header_prefix + header + b' '*padlen + b'\n'
```

NPY file-format canonical parser, last modified in the past month

`https://github.com/numpy/numpy/blob/master/numpy/lib/format.py#L346`

What We Do Currently: NumPy's NPY File Format


```
346     def _wrap_header(header, version):
347         """
348         Takes a stringified header, and attaches the prefix and padding to it
349         """
350         import struct
351         assert version is not None
352         fmt, encoding = _header_size_info[version]
353         if not isinstance(header, bytes):  # always true on python 3
354             header = header.encode(encoding)
355         hlen = len(header) + 1
356         padlen = ARRAY_ALIGN - ((MAGIC_LEN + struct.calcsize(fmt) + hlen) % ARRAY_ALIGN)
357         try:
358             header_prefix = magic(*version) + struct.pack(fmt, hlen + padlen)
359         except struct.error:
360             msg = "Header length {} too big for version={}".format(hlen, version)
361             raise ValueError(msg)
362
363         # Pad the header with spaces and a final newline such that the magic
364         # string, the header-length short and the header are aligned on a
365         # ARRAY_ALIGN byte boundary.  This supports memory mapping of dtypes
366         # aligned up to ARRAY_ALIGN on systems like Linux where mmap()
367         # offset must be page-aligned (i.e. the beginning of the file).
368         return header_prefix + header + b' '*padlen + b'\n'
```

NPY file-format canonical parser, last modified in the past month

`https://github.com/numpy/numpy/blob/master/numpy/lib/format.py#L346`

What We Do Currently: Informal Comments

```
 90  Format Version 1.0
 91  -------------------
 92  The first 6 bytes are a magic string: exactly ``\\x93NUMPY``.
 93  The next 1 byte is an unsigned byte: the major version number of the file
 94  format, e.g. ``\\x01``.
 95
 96  The next 1 byte is an unsigned byte: the minor version number of the file
 97  format, e.g. ``\\x00``. Note: the version of the file format is not tied
 98  to the version of the numpy package.
 99
100  The next 2 bytes form a little-endian unsigned short int: the length of
101  the header data HEADER_LEN.
102
103  The next HEADER_LEN bytes form the header data describing the array's
104  format. It is an ASCII string which contains a Python literal expression
105  of a dictionary. It is terminated by a newline (``\\n``) and padded with
106  spaces (``\\x20``) to make the total of
107  ``len(magic string) + 2 + len(length) + HEADER_LEN`` be evenly divisible
108  by 64 for alignment purposes.
```

What We Do Currently

- Specialized formats: deserialization (copying) required with most GCs.
  - Frameworks like Apache Spark hide individual data entries from the GC.
  - Libraries like NumPy support memory-mapping out-of-core data.
  - HDF5 and other general-purpose layout framework solutions have slow adoption across ecosystem.

What We Do Currently: I/O with HDF5 in TensorFlow

Tufts UNIVERSITY

tensorflow / **io**

Support HDF5 in tensorflow-io #174

Open **yongtang** opened this issue on Apr 4 · 20 comments

▶ New issue

yongtang commented on Apr 4 — Member

The follow is from tensorflow/tensorflow#27510

> I am currently using HDF5 files (.h5 or .hdf5) to store my data, which is a data type frequently used in scientific research. (See #2089 for a similar but different request which makes the case for a HDF5 interface nicely). It is very convenient and widely used. MATLAB for example uses HDF5 files for large files. Indeed, it is much more convenient to use than Tensorflow's TFRecord format.
>
> However, in Tensorflow, there is no native support for HDF5 files in the tf.data.Dataset API, which is supposed to be the new API for all data loading. Currently, I am using tf.py_funtion to load my data for the simple reason that tf Dataset is always in graph mode and hence cannot give out the values of the files that I want it to read.
>
> Moreover, I have found that reading an HDF5 file in this way DRAMATICALLY slows down data I/O for unknown reasons. When I used the tf.keras.utils.Sequence API to read HDF5 files without the supposed optimizations that tensorflow is making, an operation that previously took hours now took just a few seconds. (However, I suspect that using tf.defun somehow got tangled up with this. I am not sure why but when I removed some lines, the code sped up, but was still much slower than even a single threaded for loop)

Assignees
No one assigned

Labels
feature

Projects
None yet

Milestone
No milestone

5 participants

What We Do Currently: I/O Allocation with HDF5 in TensorFlow

README.md

TensorFlow I/O

build error | pypi package 0.7.0 | CRAN 0.4.0

TensorFlow I/O is a collection of file systems and file formats that are not available in TensorFlow's built-in support.

At the moment TensorFlow I/O supports the following data sources:

- `tensorflow_io.ignite` : Data source for Apache Ignite and Ignite File System (IGFS). Overview and usage guide here.
- `tensorflow_io.kafka` : Apache Kafka stream-processing support.
- `tensorflow_io.kinesis` : Amazon Kinesis data streams support.
- `tensorflow_io.hadoop` : Hadoop SequenceFile format support.
- `tensorflow_io.arrow` : Apache Arrow data format support. Usage guide here.
- `tensorflow_io.image` : WebP and TIFF image format support.
- `tensorflow_io.libsvm` : LIBSVM file format support.
- `tensorflow_io.ffmpeg` : Video and Audio file support with FFmpeg.
- `tensorflow_io.parquet` : Apache Parquet data format support.
- `tensorflow_io.lmdb` : LMDB file format support.
- `tensorflow_io.mnist` : MNIST file format support.
- `tensorflow_io.pubsub` : Google Cloud Pub/Sub support.
- `tensorflow_io.bigtable` : Google Cloud Bigtable support.
- `tensorflow_io.oss` : Alibaba Cloud Object Storage Service (OSS) support. Usage guide here.
- `tensorflow_io.avro` : Apache Avro file format support.
- `tensorflow_io.audio` : WAV file format support.
- `tensorflow_io.grpc` : gRPC server Dataset, support for streaming Numpy input.
- `tensorflow_io.hdf5` : HDF5 file format support.
- `tensorflow_io.text` : Text file with archive support.

`https://github.com/tensorflow/io`

What We Do Currently: I/O Allocation with HDF5 in TensorFlow

README.md

TensorFlow I/O

build error | pypi package 0.7.0 | CRAN 0.4.0

TensorFlow I/O is a collection of file systems and file formats that are not available in TensorFlow's built-in support.

At the moment TensorFlow I/O supports the following data sources:

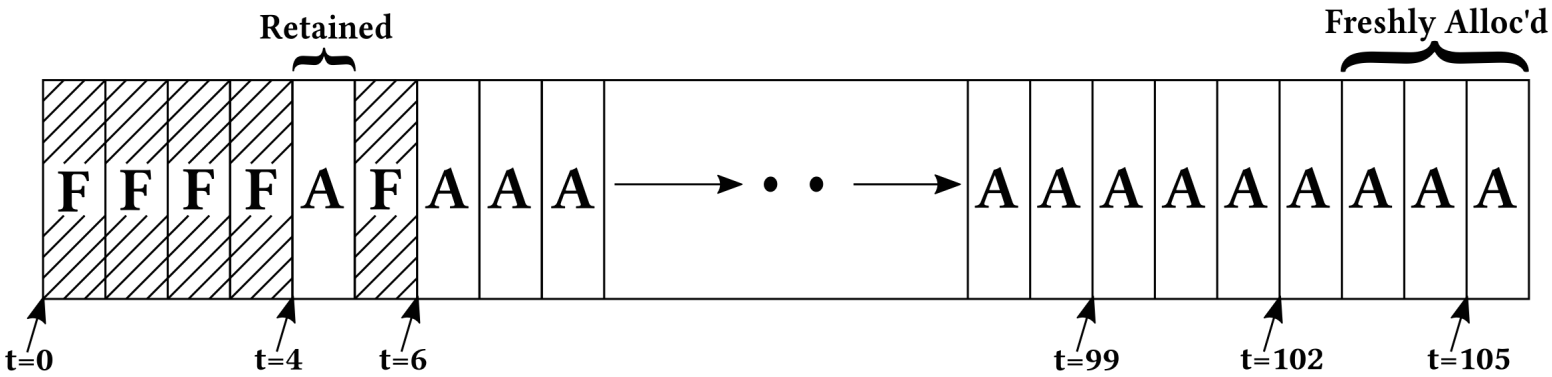
- `tensorflow_io.ignite` : Data source for Apache Ignite and Ignite File System (IGFS). Overview and usage guide here.
- `tensorflow_io.kafka` : Apache Kafka stream-processing support.
- `tensorflow_io.kinesis` : Amazon Kinesis data streams support.
- `tensorflow_io.hadoop` : Hadoop SequenceFile format support.
- `tensorflow_io.arrow` : Apache Arrow data format support. Usage guide here.
- `tensorflow_io.image` : WebP and TIFF image format support.
- `tensorflow_io.libsvm` : LIBSVM file format support.
- `tensorflow_io.ffmpeg` : Video and Audio file support with FFmpeg.
- `tensorflow_io.parquet` : Apache Parquet data format support.
- `tensorflow_io.lmdb` : LMDB file format support.
- `tensorflow_io.mnist` : MNIST file format support.
- `tensorflow_io.pubsub` : Google Cloud Pub/Sub support.
- `tensorflow_io.bigtable` : Google Cloud Bigtable support.
- `tensorflow_io.oss` : Alibaba Cloud Object Storage Service (OSS) support. Usage guide here.
- `tensorflow_io.avro` : Apache Avro file format support.
- `tensorflow_io.audio` : WAV file format support.
- `tensorflow_io.grpc` : gRPC server Dataset, support for streaming Numpy input.
- `tensorflow_io.hdf5` : HDF5 file format support.
- `tensorflow_io.text` : Text file with archive support.

`https://github.com/tensorflow/io`

Garbage Collection: Motivating Custom Allocators

- Performance is poor on non-generational and specialized workloads.
  - Way 1: Object sizes & layout.
  - Way 2: Object lifetimes & allocation.
  - Way 3: Locality and access patterns.

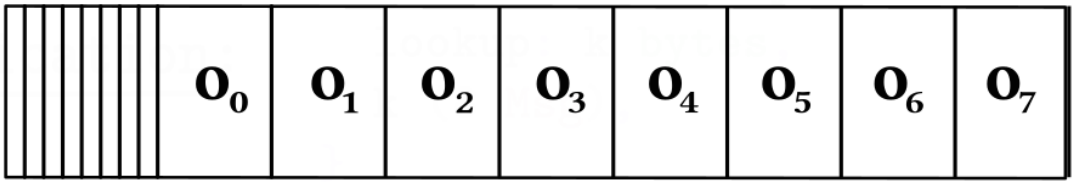
Way 3: Access Patterns



Way 3: Bitmaps

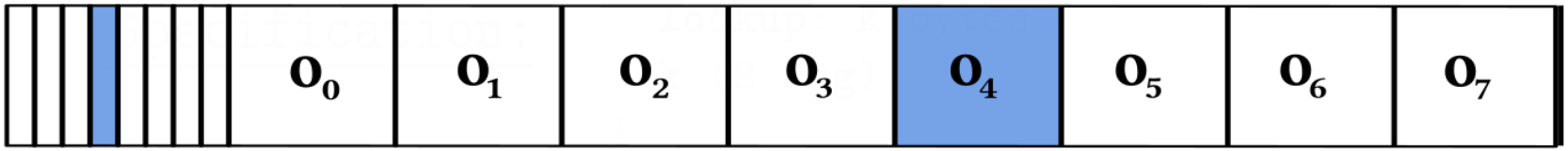

```
61  /* Computes the address of the given Msg ptr's lookup table
62   * entry, as well as the mask for accessing the appropriate
63   * bit of that u8 entry. */
64  #[inline(always)]
65  fn table_addr(ptr: usize) -> (usize, u8) {
66    let block_base = (ptr >> Block::LOG_SIZE) << Block::LOG_SIZE;
67    let msgs = block_base + Block::BYTES_IN_LOOKUP;
68    let idx : usize = (ptr - msgs) / Msg::SIZE;
69    let entry_a = block_base + (idx >> 3);
70    let bit_mask = (1 << (idx & 0b00000111)) as u8;
71    (entry_a, bit_mask)
72  }
```

Way 3: Bitmaps



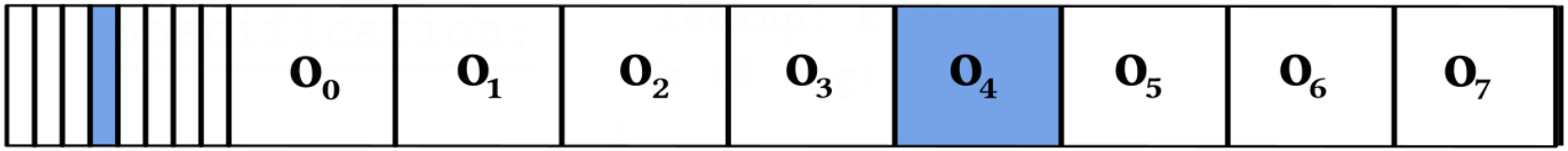

```
61  /* Computes the address of the given Msg ptr's lookup table
62   * entry, as well as the mask for accessing the appropriate
63   * bit of that u8 entry. */
64  #[inline(always)]
65  fn table_addr(ptr: usize) -> (usize, u8) {
66    let block_base = (ptr >> Block::LOG_SIZE) << Block::LOG_SIZE;
67    let msgs = block_base + Block::BYTES_IN_LOOKUP;
68    let idx : usize = (ptr - msgs) / Msg::SIZE;
69    let entry_a = block_base + (idx >> 3);
70    let bit_mask = (1 << (idx & 0b00000111)) as u8;
71    (entry_a, bit_mask)
72  }
```

Way 3: Bitmaps




```
61  /* Computes the address of the given Msg ptr's lookup table
62   * entry, as well as the mask for accessing the appropriate
63   * bit of that u8 entry. */
64  #[inline(always)]
65  fn table_addr(ptr: usize) -> (usize, u8) {
66    let block_base = (ptr >> Block::LOG_SIZE) << Block::LOG_SIZE;
67    let msgs = block_base + Block::BYTES_IN_LOOKUP;
68    let idx : usize = (ptr - msgs) / Msg::SIZE;
69    let entry_a = block_base + (idx >> 3);
70    let bit_mask = (1 << (idx & 0b00000111)) as u8;
71    (entry_a, bit_mask)
72  }
```

Way 3: Bitmaps




```
61  /* Computes the address of the given Msg ptr's lookup table
62   * entry, as well as the mask for accessing the appropriate
63   * bit of that u8 entry. */
64  #[inline(always)]
65  fn table_addr(ptr: usize) -> (usize, u8) {
66    let block_base = (ptr >> Block::LOG_SIZE) << Block::LOG_SIZE;
67    let msgs = block_base + Block::BYTES_IN_LOOKUP;
68    let idx : usize = (ptr - msgs) / Msg::SIZE;
69    let entry_a = block_base + (idx >> 3);
70    let bit_mask = (1 << (idx & 0b00000111)) as u8;
71    (entry_a, bit_mask)
72  }
```

Way 3: Bitmaps

Specification:

```
Block<k> @|2^21 bytes|@ -> seq {
  lookup: k bits, 1 bits,
  msgs:   k Msg,
}
```

```
61 /* Computes the address of the given Msg ptr's lookup table
62  * entry, as well as the mask for accessing the appropriate
63  * bit of that u8 entry. */
64 #[inline(always)]
65 fn table_addr(ptr: usize) -> (usize, u8) {
66   let block_base = (ptr >> Block::LOG_SIZE) << Block::LOG_SIZE;
67   let msgs = block_base + Block::BYTES_IN_LOOKUP;
68   let idx : usize = (ptr - msgs) / Msg::SIZE;
69   let entry_a = block_base + (idx >> 3);
70   let bit_mask = (1 << (idx & 0b00000111)) as u8;
71   (entry_a, bit_mask)
72 }
```

Way 3: Bitmaps

Specification:

```
Block<k> @|2^21 bytes|@ -> seq {
  lookup: k bits, 1 bits,
  msgs:   k Msg,
}
```

```
61 /* Computes the address of the given Msg ptr's lookup table
62  * entry, as well as the mask for accessing the appropriate
63  * bit of that u8 entry. */
64 #[inline(always)]
65 fn table_addr(ptr: usize) -> (usize, u8) {
66   let block_base = (ptr >> Block::LOG_SIZE) << Block::LOG_SIZE;
67   let msgs = block_base + Block::BYTES_IN_LOOKUP;
68   let idx : usize = (ptr - msgs) / Msg::SIZE;
69   let entry_a = block_base + (idx >> 3);
70   let bit_mask = (1 << (idx & 0b00000111)) as u8;
71   (entry_a, bit_mask)
72 }
```

Way 3: Bitmaps

Specification:

```
Block<k> @|2^21 bytes|@ -> seq {
  lookup: k bits, 1 bits,
  msgs:   k Msg,
}
```

```
61 /* Computes the address of the given Msg ptr's lookup table
62  * entry, as well as the mask for accessing the appropriate
63  * bit of that u8 entry. */
64 #[inline(always)]
65 fn table_addr(ptr: MsgAddr) -> (BitsAddr, BitsMask) {
66   let block : BlockAddr = ptr.get_containing_BlockAddr();
67   let msgs : ObjsAddr = block.msgs();
68   let idx : usize = msgs.get_idx_of(ptr);
69   let entry_a : BitsAddr = block.lookup().get_BitsAddr_of_idx
70   let bit_mask : BitsMask = BitsMask::from_idx(idx);      (idx);
71   (entry_a, bit_mask)
72 }
```

Way 3: Bitmaps

Specification:

```
Block<k> @|2^21 bytes|@ -> seq {
  lookup: k bits, 1 bits,
  msgs:   k Msg,
}
```

```
61  /* Computes the address of the given Msg ptr's lookup table
62   * entry, as well as the mask for accessing the appropriate
63   * bit of that u8 entry. */
64  #[inline(always)]
65  fn table_addr(ptr: MsgAddr) -> (BitsAddr, BitsMask) {
66    let block : BlockAddr = ptr.get_containing_BlockAddr();
67    let msgs : ObjsAddr = block.msgs();
68    let idx : usize = msgs.get_idx_of(ptr);
69    let entry_a : BitsAddr = block.lookup().get_BitsAddr_of_idx
                                                    (idx);
70    let bit_mask : BitsMask = BitsMask::from_idx(idx);
71    (entry_a, bit_mask)
72  }
```

Way 3: Bitmaps

Tufts UNIVERSITY

Specification:

```
Block<k> @|2^21 bytes|@ -> seq {
    lookup: k bits, 1 bits,
    msgs:   k Msg,
}
```

```
61  /* Computes the address of the given Msg ptr's lookup table
62   * entry, as well as the mask for accessing the appropriate
63   * bit of that u8 entry. */
64  #[inline(always)]
65  fn table_addr(ptr: MsgAddr) -> (BitsAddr, BitsMask) {
66    let block : BlockAddr = ptr.get_containing_BlockAddr();
67    let msgs : ObjsAddr = block.msgs();
68    let idx : usize = msgs.get_idx_of(ptr);
69    let entry_a : BitsAddr = block.lookup().get_BitsAddr_of_idx
70                                                    (idx);
71    let bit_mask : BitsMask = BitsMask::from_idx(idx);
72    (entry_a, bit_mask)
  }
```

Way 3: Bitmaps

Tufts UNIVERSITY

Specification:

```
Block<k> @|2^21 bytes|@ -> seq {
  lookup: k bits, 1 bits,
  msgs:   k Msg,
}
```

```
61  /* Computes the address of the given Msg ptr's lookup table
62   * entry, as well as the mask for accessing the appropriate
63   * bit of that u8 entry. */
64  #[inline(always)]
65  fn table_addr(ptr: MsgAddr) -> (BitsAddr, BitsMask) {
66    let block : BlockAddr = ptr.get_containing_BlockAddr();
67    let msgs : ObjsAddr = block.msgs();
68    let idx : usize = msgs.get_idx_of(ptr);
69    let entry_a : BitsAddr = block.lookup().get_BitsAddr_of_idx
70                                                    (idx);
71    let bit_mask : BitsMask = BitsMask::from_idx(idx);
72    (entry_a, bit_mask)
  }
```

Way 3: Bitmaps

Tufts UNIVERSITY

Specification:

```
Block<k> @|2^21 bytes|@ -> seq {
  lookup: k bits, 1 bits,
  msgs:   k Msg,
}
```

```
61  /* Computes the address of the given Msg ptr's lookup table
62   * entry, as well as the mask for accessing the appropriate
63   * bit of that u8 entry. */
64  #[inline(always)]
65  fn table_addr(ptr: MsgAddr) -> (BitsAddr, BitsMask) {
66    let block : BlockAddr = ptr.get_containing_BlockAddr();
67    let msgs : ObjsAddr = block.msgs();
68    let idx : usize = msgs.get_idx_of(ptr);
69    let entry_a : BitsAddr = block.lookup().get_BitsAddr_of_idx
70    let bit_mask : BitsMask = BitsMask::from_idx(idx);          (idx);
71    (entry_a, bit_mask)
72  }
```

Way 3: Bitmaps

Tufts UNIVERSITY

Specification:

```
Block<k> @|2^21 bytes|@ -> seq {
  lookup: k bits, 1 bits,
  msgs:   k Msg,
}
```

```
61  /* Computes the address of the given Msg ptr's lookup table
62   * entry, as well as the mask for accessing the appropriate
63   * bit of that u8 entry. */
64  #[inline(always)]
65  fn table_addr(ptr: MsgAddr) -> (BitsAddr, BitsMask) {
66    let block : BlockAddr = ptr.get_containing_BlockAddr();
67    let msgs : ObjsAddr = block.msgs();
68    let idx : usize = msgs.get_idx_of(ptr);
69    let entry_a : BitsAddr = block.lookup().get_BitsAddr_of_idx
70    let bit_mask : BitsMask = BitsMask::from_idx(idx);      (idx);
71    (entry_a, bit_mask)
72  }
```

Semantics Un-mentioned in this Talk

Tufts UNIVERSITY

- Byte order.
- Macros.
- Architectural constants.
- Performance characteristics.
- Reduction semantics defining a Floorplan type.
- Compiler implementation details.
- Coq proofs verifying certain language-level properties.

What do we Lose?

- New spatially optimized data structures must be written with custom collection in mind => but Floorplan takes care of the types.

Common Pitfalls with Custom Allocators and Collectors

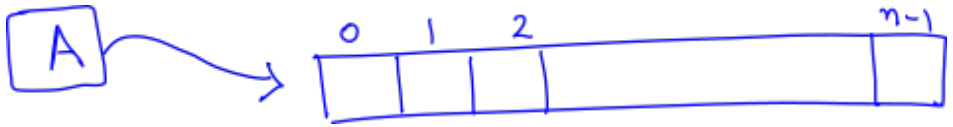
Tufts UNIVERSITY

- Ignoring of diminishing returns on ever-improving modern commodity hardware.
- Pre-emptively optimizing microbenchmarks deemed representative.
- Incurring bookkeeping burdens for each custom memory manager.
- Need to write custom interfaces to memory debuggers like Valgrind, GDB, and Purify.
- Sunk costs due to how difficult they are to write in the first place!

Questions?

Example: if we need an array of n ints, then we can do

```
int* A = malloc(n*sizeof(int));
```



A holds the address of the first element of this block of 4n bytes, and A can be used as an array. For example,

```
if (A != NULL)
 for (i=0;i<n;i++)
 A[i] = 0;
```

will initialize all elements in the array to 0. We note that A[i] is the content at address (A+i). Therefore we can also write

```
for (i=0;i<n;i++)
 *(A+i) = 0;
```

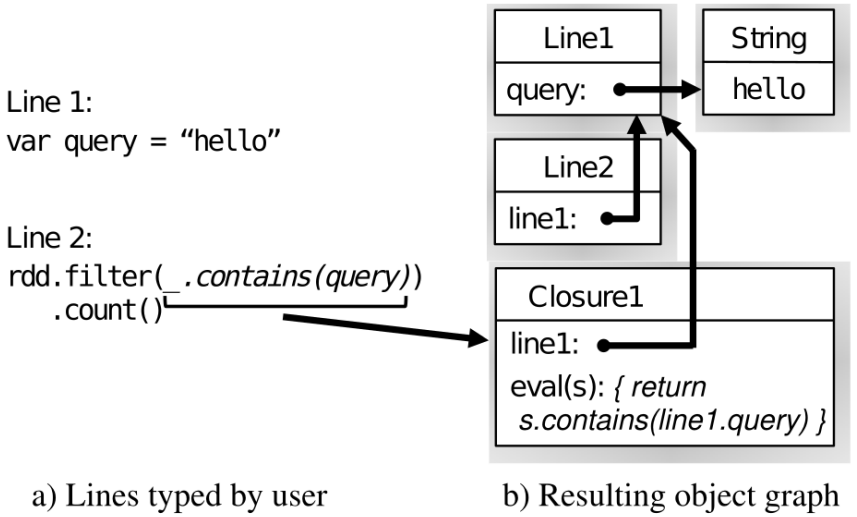
https://www.cs.cmu.edu/~guna/15-123S11/Lectures/Lecture08.pdf

5.3 Memory Management

Spark provides three options for storage of persistent RDDs: in-memory storage as deserialized Java objects, in-memory storage as serialized data, and on-disk storage. The first option provides the fastest performance, because the Java VM can access each RDD element natively. The second option lets users choose a more memory-efficient representation than Java object graphs when space is limited, at the cost of lower performance.[8] The third option is useful for RDDs that are too large to keep in RAM but costly to recompute on each use.

To manage the limited memory available, we use an LRU eviction policy at the level of RDDs. When a new RDD partition is computed but there is not enough space to store it, we evict a partition from the least recently accessed RDD, unless this is the same RDD as the one with the new partition. In that case, we keep the old partition in memory to prevent cycling partitions from the same RDD in and out. This is important because most operations will run tasks over an entire RDD, so it is quite likely that the partition already in memory will be needed in the future. We found this default policy to work well in all our applications so far, but we also give users further control via a "persistence priority" for each RDD.

Finally, each instance of Spark on a cluster currently has its own separate memory space. In future work, we plan to investigate sharing RDDs across instances of Spark through a unified memory manager.



a) Lines typed by user b) Resulting object graph

Figure 6: Example showing how the Spark interpreter translates two lines entered by the user into Java objects.

https://people.csail.mit.edu/matei/papers/2012/nsdi_spark.pdf

Extras: Related Work

1. Implementation of Immix in Rust
2. Block-structured heaps
3. GHC block descriptors
4. Quadtree layout visualization
5. Heap Layers
6. Flash Relate: Flare
7. PADS-Haskell parsing
8. LoCal location calculus

Abusing Rust

- Safety model too restrictive at times, so must use *unsafe* code
- Implementing the Line Mark Table
  - Remember the state of every line in memory
  - Map of unsigned bytes (*u8*) for every 256-bytes of memory
- Allocation: Multiple allocators may access line mark table
  - Rust array of *u8* disallows concurrent writing
- Collection: Set lines to *live* by atomically storing to a byte
  - Rust does not support Atomic unsigned bytes
- Work Around: Generalize Line Mark Table as *AddressMapTable*
- Wrap *unsafe* code into impl of *AddressMapTable*
- Rely on compiler to generate x86 Byte store which is atomic

```
pub struct AddressMapTable {
  start : Address,
  end   : Address,

  len : usize,
  ptr : *mut u8
}
// allow sharing of AddressMapTable across threads
unsafe impl Sync for AddressMapTable {};
unsafe impl Send for AddressMapTable {};

impl AddressMapTable {
  pub unsafe fn set (&self, addr: Address, value: u8)
  {
    let index = addr.diff(self.start) >> LOG_PTR_SIZE;
    unsafe {
      let ptr = self.ptr.offset(index);
      // intrinsics::atomic_store_relaxed(ptr, value);
      *ptr = value;
    }
  }
}
```

https://www.eidos.ic.i.u-tokyo.ac.jp/~tau/lecture/programming_languages/presentations/2017/valentino.pdf

Block-Structured Heaps

Thus motivated, GHC's storage manager uses a block-structured heap. Although this is not a new idea [DEB94, Ste77], the literature is surprisingly thin, so we pause to describe how it works.

- The heap is divided into fixed-size B-byte blocks. Their exact size B is not important, except that it must be a power of 2. GHC uses 4kbytes blocks by default, but this is just a compile-time constant and is easily changed.
- Each block has an associated block descriptor, which describes the generation and step of the block, among other things. Any address within a block can be mapped to its block descriptor with a handful of instructions - we discuss the details of this mapping in Section 2.3.
- Blocks can be linked together, through a field in their descriptors, to form an area. For example, after garbage collection the mutator is provided with an allocation area of free blocks in which to allocate fresh objects.
- The heap contains heap objects, whose layout is shown in Figure 1. From the point of view of this paper, the important point is that the first word of every heap object, its info pointer, points to its statically-allocated info table, which in turn contains layout information that guides the garbage collector.
- A heap pointer always addresses the first word of a heap object; we do not support interior pointers.
- A large object, whose size is greater than a block, is allocated in a block group of contiguous blocks.

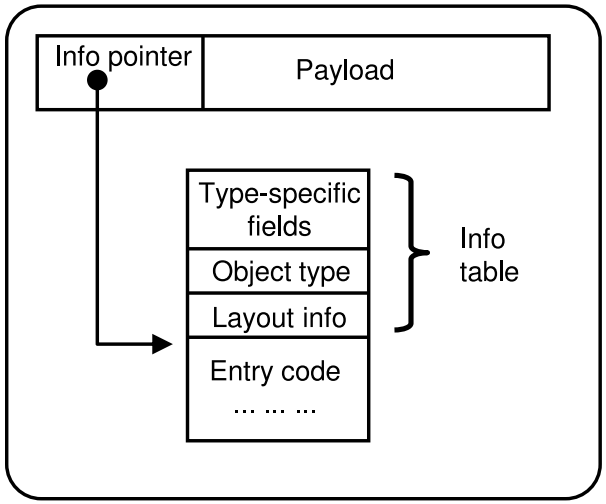
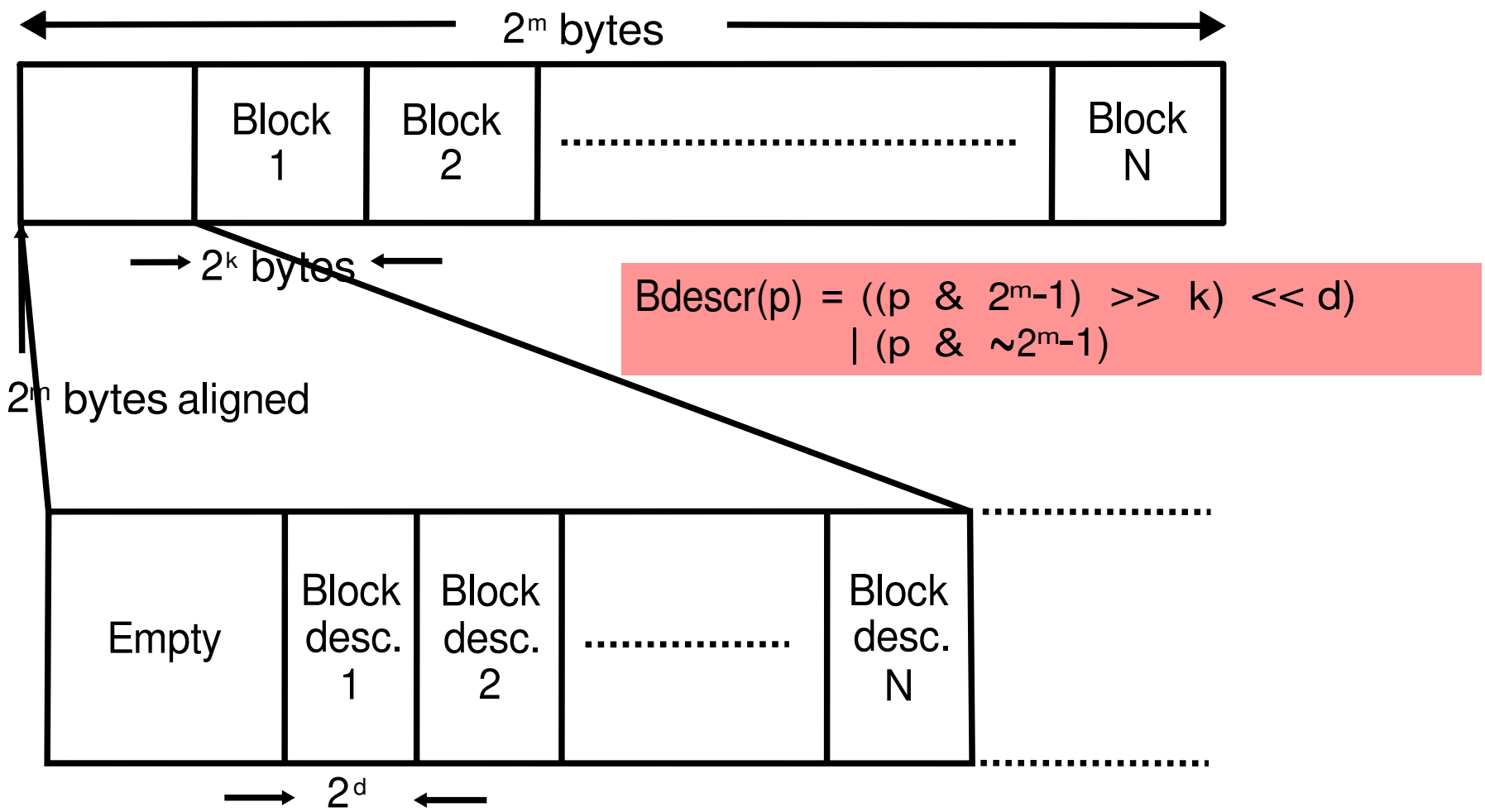


Figure 1. A heap object

https://dl.acm.org/citation.cfm?id=1375637

Highly Optimized Lookup Tables



https://gitlab.haskell.org/ghc/ghc/wikis/commentary/rts/storage/block-alloc

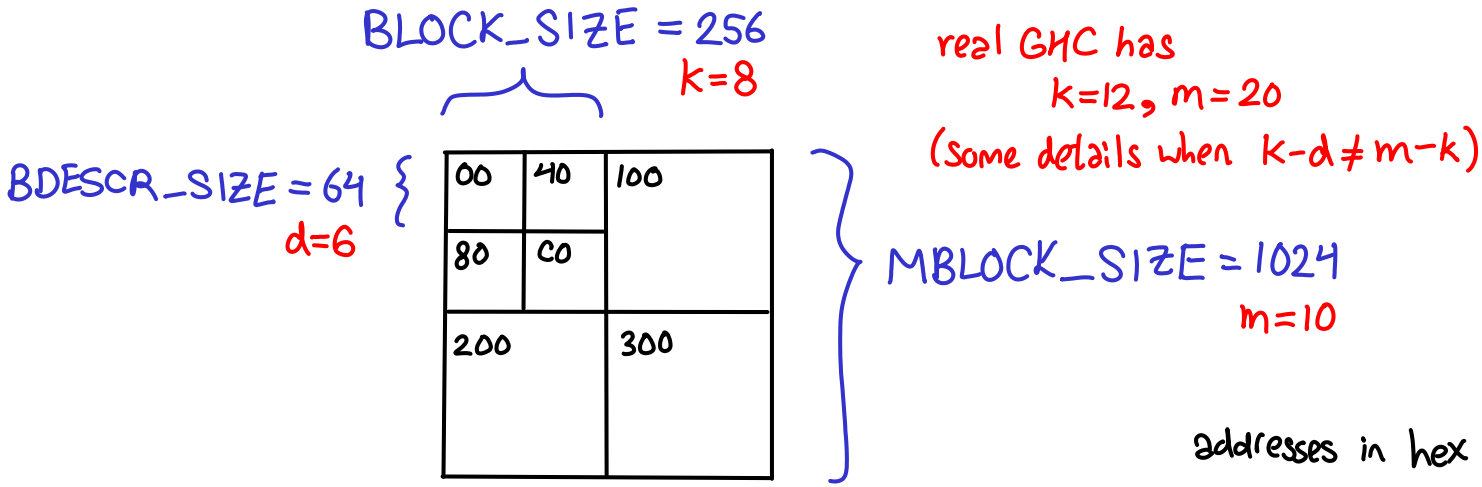
Quadtree Layout Visualization

THE GHC BLOCK ALLOCATOR ON A 64-BIT MACHINE (d=6)

— Edward Z. Yang

The block allocator allows us to manage our heap (and other allocations) with multiple blocks, rather than a single contiguous region. This scheme was presented in the paper "Parallel generational-copying garbage collection with a block-structured heap" and there is also some good commentary at
http://ghc.haskell.org/trac/ghc/wiki/Commentary/Rts/Storage/BlockAlloc

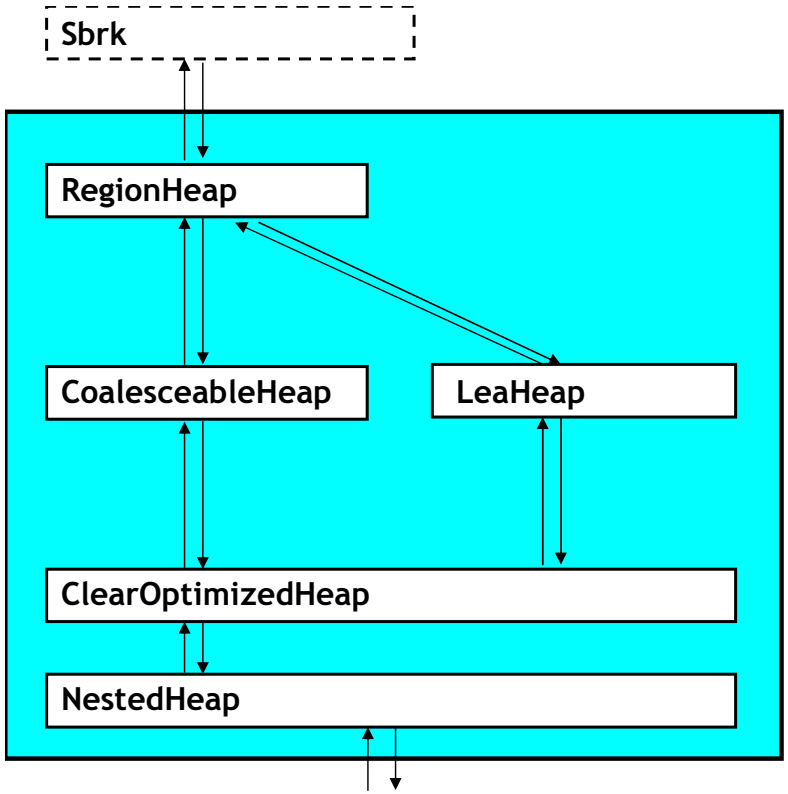
In this document, I want to visualize block allocation using small choices for block size and megablock size.



We start by looking at the memory layout of a single megablock. We've laid out the memory like a quadtree, so that given any block, the order of memory can be read out by dividing it into four sub-blocks and recursively reading the sub-blocks in the usual order (each shown block has been annotated with a hexadecimal address for your convenience). The smallest "block" is 64 bytes large (appropriate for a 64-bit machine), and represents a

`http://web.mit.edu/~ezyang/Public/blocks.pdf`

Logical Heap Layers (vs Spatial Floorplans)



(b) A diagram of the heap layers that comprise our implementation of reaps. Reaps adapt to their use, acting either like regions or heaps (see Section 5). The CoalesceableHeap layer adds per-object metadata that enable a heap to subsequently manage memory obtained from a region.

Figure 3: A description of the API and implementation of reaps.

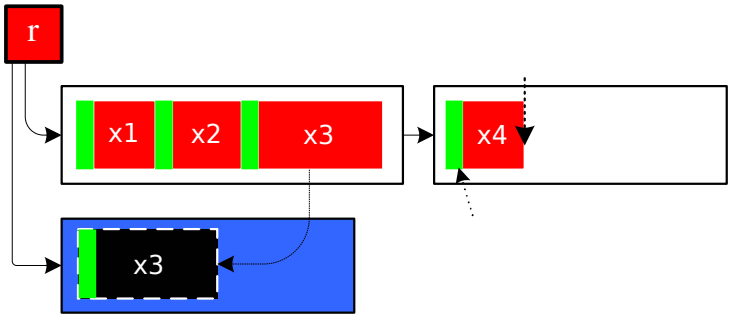
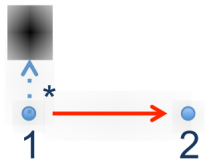


Figure 4: An example of reap allocation and deallocation. Reaps add metadata to objects so that they can be freed onto a heap.

`http://www.cs.umass.edu/~emery/pubs/berger-oopsla2002.pdf`

```
Node(1,"^[0-9]+$",Anchor("value",Vert(*),Horiz(0)))
Node(2,"^[0-9]+$",⊥)
Edge(1,2,Vert(0),Horiz(1),Select(All,All))
```

(a)

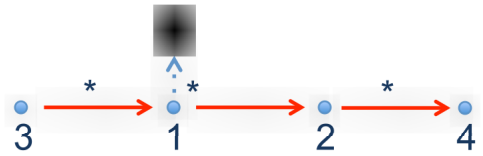


(b)

Figure 3: (a) FLARE program for first example extraction task with (b) schematic illustration. **Node** constraints are shown as dots, **Edge** constraints are shown as solid arrows, and **Anchor** constraints are shown with a dashed arrow and anchor symbol. **Edge** s and **Anchor**s of non-constant length are labeled with a Kleene star. Node numbers correspond to attribute IDs in the desired relational tuple.

```
Node(3,"^[a-zA-Z ]+$",⊥)
Node(4,"^[a-zA-Z ]+$",⊥)
Edge(3,1,Vert(0),Horiz(*),Select(All,All))
Edge(2,4,Vert(0),Horiz(*),Select(All,All))
```

(a)



(b)

Figure 4: (a) A set of additional constraints added to Figure 3a; (b) schematic illustration.

	A	B	C	D	E	...	R
1		value	year	value	year		Comments
2	Albania	1,000	1950	930	1981		FRA 1
3	Austria	3,139	1951	3,177	1955		FRA 3
4	Belgium	541	1947	601	1950		
5	Bulgaria	2,964	1947	3,259	1958		FRA 1
6	Czech ...	2,416	1950	2,503	1960		NC

...

(a)

	A	B	C	D
1	Albania	1,000	1950	FRA 1
2	Albania	930	1981	FRA 1
	...			
5	Austria	3,139	1951	FRA 3
6	Austria	3,177	1955	FRA 3
	...			
9	Belgium	541	1947	
10	Belgium	601	1950	

...

(b)

Figure 1: (a) A semi-structured spreadsheet with two example tuples highlighted. The first tuple (red) represents the timber harvest (per 1000 hectares) for Albania in 1950. The second tuple (blue) represents the timber harvest for Austria in 1950. (b) An extracted relational table with the same two tuples highlighted as in Fig. 1a

https://doi.org/10.1145/2813885.2737952

PADS-Haskell: Layouts for Parsing [1/2]

Tufts UNIVERSITY

```
1   newtype MapsFile =
2     MapsFile ([Line Region] terminator EOF)
3
4   data Region = Region
5     {           start_addr :: Hex
6     , '-',      end_addr   :: Hex
7     , ' ',      perms      :: Permissions
8     , ' ',      offset     :: Int
9     , ' ',      device     :: (Hex, ':', Hex)
10    , ' ',      inode      :: Int
11    , ws ,      path       :: RegionName
12    }
13
14  type Hex = StringME '[0-9A-Fa-f]+'
15
16  data RP = READ  'r' | NOREAD  '-'
17  data WP = WRITE 'w' | NOWRITE '-'
18  data XP = EXEC  'x' | NOEXEC  '-'
19  data SP = SHARE 's' | PRIVATE 'p'
```

```
20  data RegionName =
21      Heap      "[heap]"
22    | Stack     "[stack]"
23    | VDSO      "[vdso]"
24    | VVAR      "[vvar]"
25    | VSyscall  "[vsyscall]"
26    | Path
27      ([Char] terminator EOR)
28    | Anonymous ""
29
30  data Permissions = Permissions
31    { permRead  :: RP
32    , permWrite :: WP
33    , permExec  :: XP
34    , permShare :: SP
35    }
```

https://github.com/padsproj/pads-haskell/blob/master/examples/Proc.hs

PADS-Haskell: Layouts for Parsing [2/2]

Tufts UNIVERSITY

```
00400000-00422000 r-xp 00000000 fe:02 4968                               /usr/bin/less
00622000-00623000 r--p 00022000 fe:02 4968                               /usr/bin/less
00623000-00627000 rw-p 00023000 fe:02 4968                               /usr/bin/less
00627000-0062b000 rw-p 00000000 00:00 0
00aa0000-00ac1000 rw-p 00000000 00:00 0                                  [heap]
7f1ae9999000-7f1ae99b2000 r-xp 00000000 fe:02 93235                      /usr/lib/libpthread-2.25.so
7f1ae99b2000-7f1ae9bb1000 ---p 00019000 fe:02 93235                      /usr/lib/libpthread-2.25.so
7f1ae9bb1000-7f1ae9bb2000 r--p 00018000 fe:02 93235                      /usr/lib/libpthread-2.25.so
7f1ae9bb2000-7f1ae9bb3000 rw-p 00019000 fe:02 93235                      /usr/lib/libpthread-2.25.so
7f1ae9bb3000-7f1ae9bb7000 rw-p 00000000 00:00 0
7f1ae9bb7000-7f1ae9d52000 r-xp 00000000 fe:02 93077                      /usr/lib/libc-2.25.so
7f1ae9d52000-7f1ae9f51000 ---p 0019b000 fe:02 93077                      /usr/lib/libc-2.25.so
7f1ae9f51000-7f1ae9f55000 r--p 0019a000 fe:02 93077                      /usr/lib/libc-2.25.so
7f1ae9f55000-7f1ae9f57000 rw-p 0019e000 fe:02 93077                      /usr/lib/libc-2.25.so
7f1ae9f57000-7f1ae9f5b000 rw-p 00000000 00:00 0
7f1ae9f5b000-7f1ae9fcd000 r-xp 00000000 fe:02 108084                     /usr/lib/libpcre.so.1.2.8
7f1ae9fcd000-7f1aea1cc000 ---p 00072000 fe:02 108084                     /usr/lib/libpcre.so.1.2.8
7f1aea1cc000-7f1aea1cd000 r--p 00071000 fe:02 108084                     /usr/lib/libpcre.so.1.2.8
7f1aea1cd000-7f1aea1ce000 rw-p 00072000 fe:02 108084                     /usr/lib/libpcre.so.1.2.8
7f1aea1ce000-7f1aea235000 r-xp 00000000 fe:02 106343                     /usr/lib/libncursesw.so.6.0
7f1aea235000-7f1aea434000 ---p 00067000 fe:02 106343                     /usr/lib/libncursesw.so.6.0
7f1aea434000-7f1aea438000 r--p 00066000 fe:02 106343                     /usr/lib/libncursesw.so.6.0
7f1aea438000-7f1aea43a000 rw-p 0006a000 fe:02 106343                     /usr/lib/libncursesw.so.6.0
7f1aea43a000-7f1aea45d000 r-xp 00000000 fe:02 93078                      /usr/lib/ld-2.25.so
7f1aea490000-7f1aea629000 r--p 00000000 fe:02 96759                      /usr/lib/locale/locale-archive
7f1aea629000-7f1aea62d000 rw-p 00000000 00:00 0
7f1aea65c000-7f1aea65d000 r--p 00022000 fe:02 93078                      /usr/lib/ld-2.25.so
7f1aea65d000-7f1aea65e000 rw-p 00023000 fe:02 93078                      /usr/lib/ld-2.25.so
7f1aea65e000-7f1aea65f000 rw-p 00000000 00:00 0
7fff3cc14000-7fff3cc35000 rw-p 00000000 00:00 0                          [stack]
7fff3cc3a000-7fff3cc3c000 r--p 00000000 00:00 0                          [vvar]
```