Betriebssysteme

5. Tutorium - Memory Management, Segmentation, allocation

Peter Bohner

29. November 2023

ITEC - Operating Systems Group

Feedback 2. Übungsblatt

- Nur 2 Abgaben, das geht besser :) Andere Tuts hatten bis zu 15 Abgaben.

Feedback 2. Übungsblatt

- Nur 2 Abgaben, das geht besser :) Andere Tuts hatten bis zu 15 Abgaben.
- Gibt es etwas, was euch motivieren könnte?

Feedback 2. Übungsblatt

- Nur 2 Abgaben, das geht besser :) Andere Tuts hatten bis zu 15 Abgaben.
- Gibt es etwas, was euch motivieren könnte?
- Fragen zum ÜB?

Feedback 2. Übungsblatt

- Nur 2 Abgaben, das geht besser :) Andere Tuts hatten bis zu 15 Abgaben.
- Gibt es etwas, was euch motivieren könnte?
- Fragen zum ÜB?

Macht Advent of Code in C

Memory Management Basics

Physical And Virtual Addresses

And once again: What is the difference?

Physical And Virtual Addresses

And once again: What is the difference?

- We assume no 1:1 mapping (i.e. we have virtual memory)

Physical And Virtual Addresses

And once again: What is the difference?

- We assume no 1:1 mapping (i.e. we have virtual memory)
- *All program addresses are virtual*

Physical And Virtual Addresses

And once again: What is the difference?

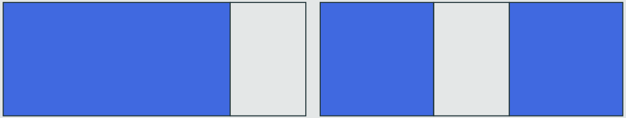
- We assume no 1:1 mapping (i.e. we have virtual memory)
- *All program addresses are virtual*
- Mapped to *physical* addresses as needed by the memory management unit

Fragmentation

What is *internal* fragmentation?

Fragmentation

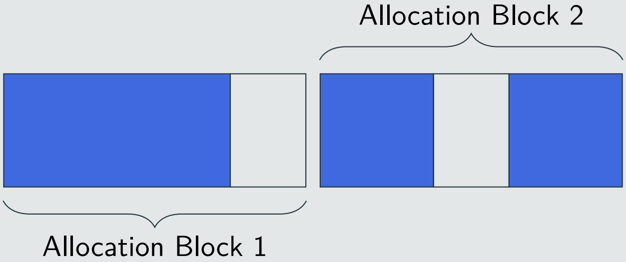
What is *internal* fragmentation?



Internal, i.e. *within* a block

Fragmentation

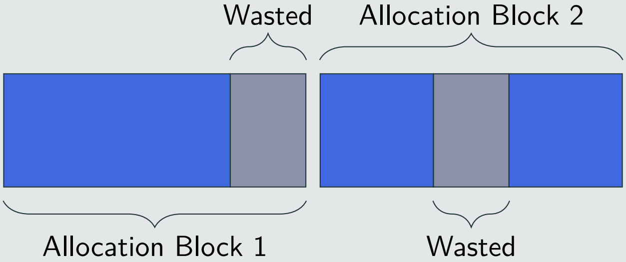
What is *internal* fragmentation?



Internal, i.e. *within* a block

Fragmentation

What is *internal* fragmentation?

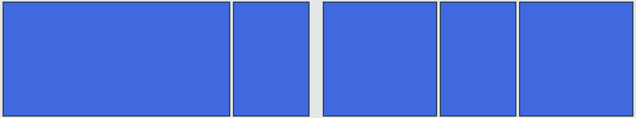


Internal, i.e. *within* a block

What is *external* fragmentation?

Fragmentation

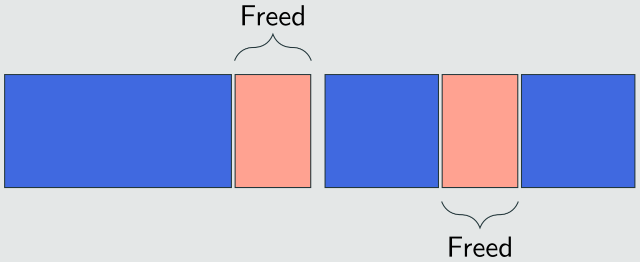
What is *external* fragmentation?



External, i.e. due to *external factors* (different time-to-free)

Fragmentation

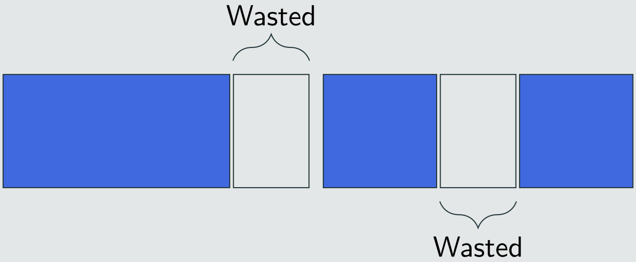
What is *external* fragmentation?



External, i.e. due to *external factors* (different time-to-free)

Fragmentation

What is *external* fragmentation?



External, i.e. due to *external factors* (different time-to-free)

Fragmentation

Can you have *both* types at the same time?

Fragmentation

Can you have *both* types at the same time?

Yes!

Fragmentation

Can you have *both* types at the same time?

Yes!

- Allocate in *chunks* by e.g. rounding up to 2^x

Fragmentation

Can you have *both* types at the same time?

Yes!

- Allocate in *chunks* by e.g. rounding up to 2^x
- Have different lifetimes

Fragmentation

Can you have *both* types at the same time?

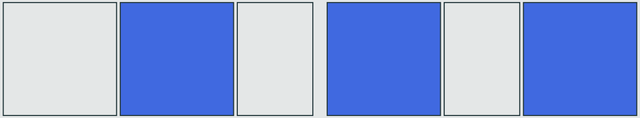
Yes!

- Allocate in *chunks* by e.g. rounding up to 2^x
- Have different lifetimes

$\Rightarrow$ Wasteful allocations scattered throughout RAM

Fragmentation

What do we do now? This sounds bad!



Compaction - Is that even possible?

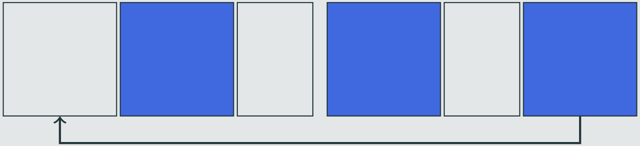
- C uses direct pointers

$\Rightarrow$ They are all garbage now!

- Works just fine in languages with indirections (e.g. garbage collection)
- Also works for segments in physical memory! How? Update base addresses in MMU

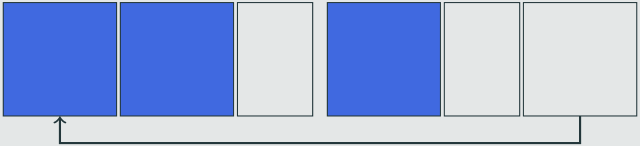
Fragmentation

What do we do now? This sounds bad!



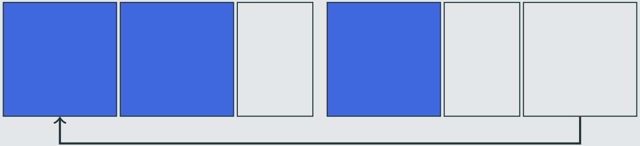
Fragmentation

What do we do now? This sounds bad!



Fragmentation

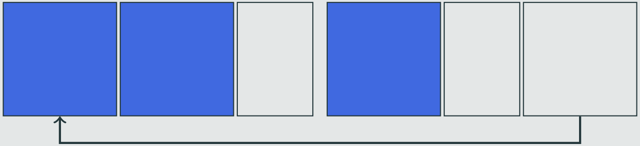
What do we do now? This sounds bad!



This is called Compaction

Fragmentation

What do we do now? This sounds bad!

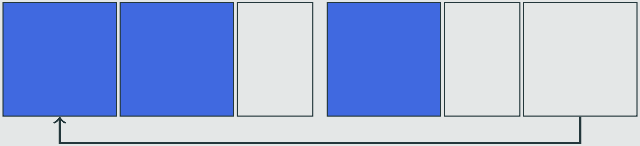


This is called Compaction

Compaction - Is that even possible?

Fragmentation

What do we do now? This sounds bad!



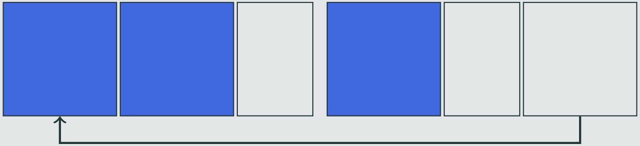
This is called Compaction

Compaction - Is that even possible?

- C uses direct pointers

Fragmentation

What do we do now? This sounds bad!



This is called Compaction

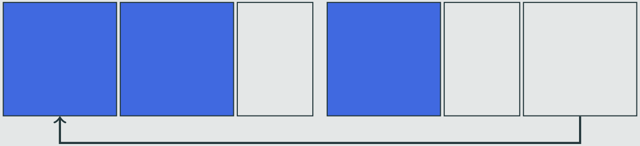
Compaction - Is that even possible?

- C uses direct pointers

$\Rightarrow$ They are all garbage now!

Fragmentation

What do we do now? This sounds bad!



This is called Compaction

Compaction - Is that even possible?

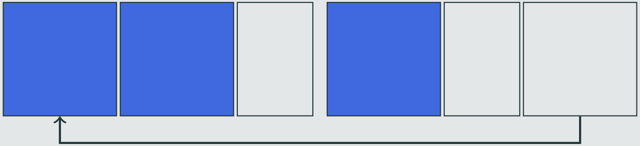
- C uses direct pointers

⇒ They are all garbage now!

- Works just fine in languages with indirections (e.g. garbage collection)

Fragmentation

What do we do now? This sounds bad!



This is called Compaction

Compaction - Is that even possible?

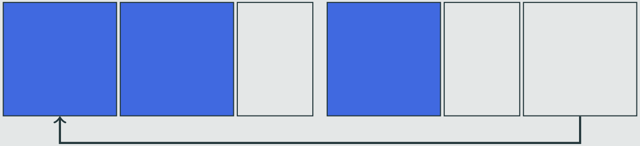
- C uses direct pointers

$\Rightarrow$ They are all garbage now!

- Works just fine in languages with indirections (e.g. garbage collection)
- Also works for segments in physical memory! How?

Fragmentation

What do we do now? This sounds bad!



This is called Compaction

Compaction - Is that even possible?

- C uses direct pointers

$\Rightarrow$ They are all garbage now!

- Works just fine in languages with indirections (e.g. garbage collection)
- Also works for segments in physical memory! How? Update base addresses in MMU

Segmentation

Segmentation

Where have you seen that word before while sadly staring at your screen?

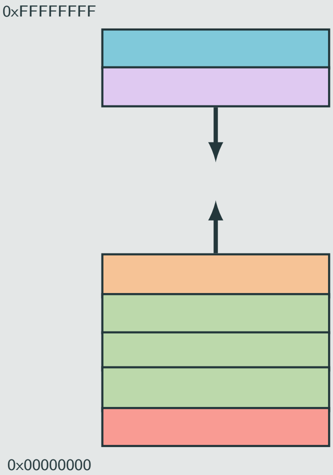
Segmentation

Where have you seen that word before while sadly staring at your screen?

```
> Segmentation fault (core dumped)
```

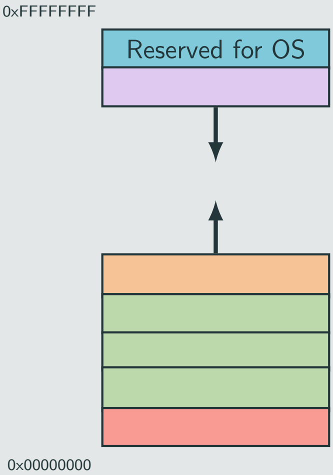
Segmentation

A few example segments



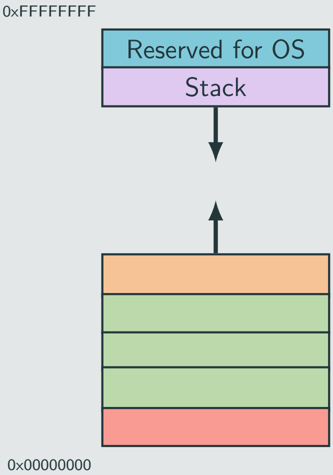
Segmentation

A few example segments



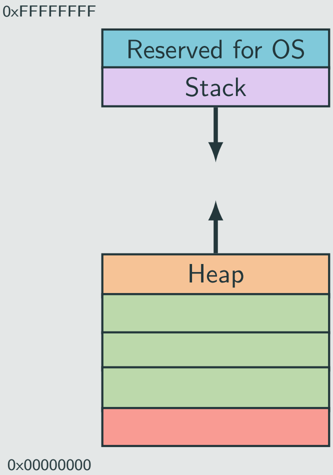
Segmentation

A few example segments



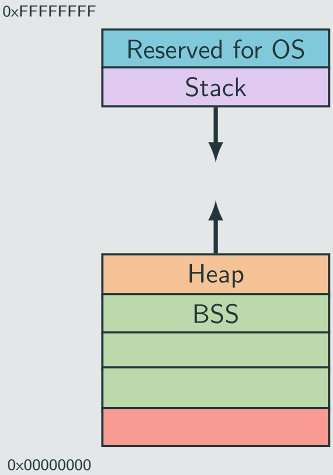
Segmentation

A few example segments



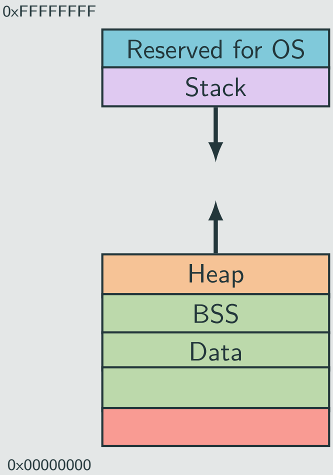
Segmentation

A few example segments

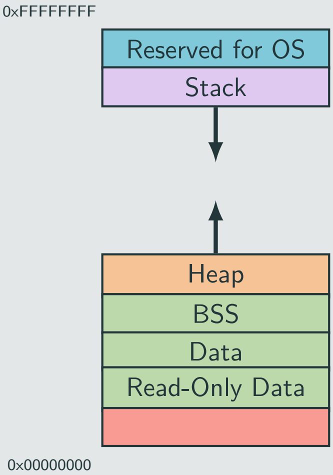


Segmentation

A few example segments

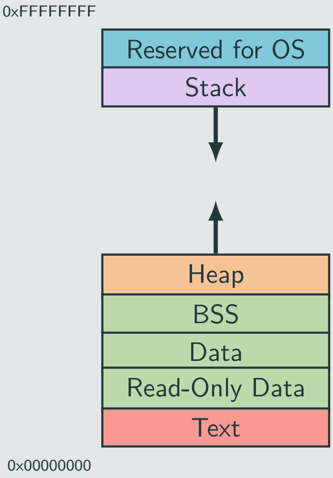


A few example segments



Segmentation

A few example segments



A Virtual Address

What does it look like?

42	`0xCAFE`

Segment Number	Base	Limit
...	...	...
42	`0xB105`	`0xF00D`
...	...	...

A Virtual Address

What does it look like?

42	0xCAFE

Segment Number	Base	Limit
...	...	...
42	0xB105	0xF00D
...	...	...

A Virtual Address

What does it look like?

42	0xCAFE

Segment Number	Base	Limit
...	...	...
42	0xB105	0xF00D
...	...	...

0xCAFE < 0xF00D

A Virtual Address

What does it look like?

42	0xCAFE

Segment Number	Base	Limit
...	...	...
42	0xB105	0xF00D
...	...	...

0xCAFE < 0xF00D

A Virtual Address

What does it look like?

42	0xCAFE

Segment Number	Base	Limit
...	...	...
42	0xB105	0xF00D
...	...	...

`0xCAFE < 0xF00D`

`0xB105 + 0xCAFE = 0x17C03`

And let's try it

Segments

Segment Number	Base	Limit
0	0xdead	0x00ef
1	0xf154	0x013a
2	0x0000	0x0000
3	0x0000	0x3fff

Your task

Virtual Address	Segment Number	Offset	Valid?	Physical Address
	3	0x3999		
0x2020				
		0x0204	yes	
			yes	0xf15f

And let's try it

Solution

Virtual Address	Segment Number	Offset	Valid?	Physical Address
0xf999	3	0x3999	yes	0x3999
0x2020	0	0x2020	no	Offset outside limit
0xc204	3	0x0204	yes	0x0204
0x400b	1	0x000b	yes	0xf15f

Memory allocation policies

Some common strategies

Which strategies for finding free blocks do you know?

First Fit,

Some common strategies

Which strategies for finding free blocks do you know?

First Fit, Best Fit,

Some common strategies

Which strategies for finding free blocks do you know?

First Fit, Best Fit, Worst Fit

Some common strategies

Which strategies for finding free blocks do you know?

First Fit, Best Fit, Worst Fit

First Fit

Pick the first block that is large enough

Some common strategies

Which strategies for finding free blocks do you know?

First Fit, Best Fit, Worst Fit

First Fit

Pick the first block that is large enough

Best Fit

Pick the *smallest* block that fits

Some common strategies

Which strategies for finding free blocks do you know?

First Fit, Best Fit, Worst Fit

First Fit

Pick the first block that is large enough

Best Fit

Pick the *smallest* block that fits

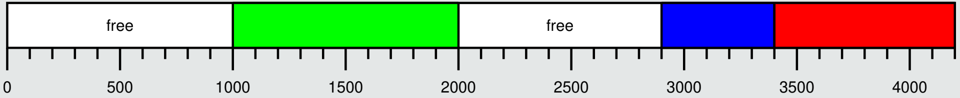
Worst Fit

Pick the *largest* block that fits

Let's try them

Best fit

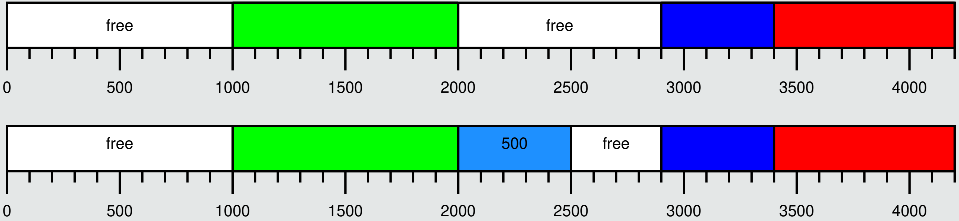
Allocate 500, 1200, and 200, fail if not possible.



Let's try them

Best fit

Allocate 500, 1200, and 200, fail if not possible.

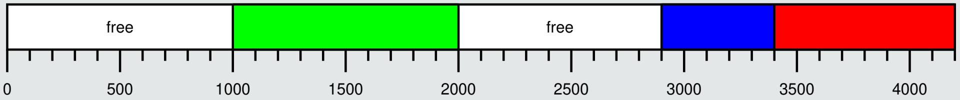


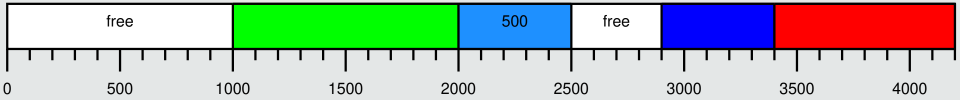
And compact it to fit the next one!

Let's try them

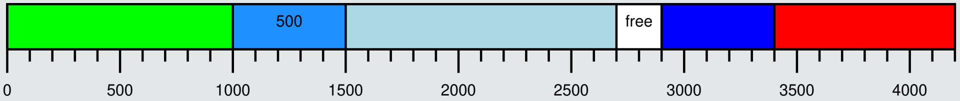
Best fit

Allocate 500, 1200, and 200, fail if not possible.





And compact it to fit the next one!



What allocator would you make up for this?

You are a poor kernel and you need lots of inodes

Every inode has the same size, 64 Byte. Can you think of any fast allocation strategy that does not waste a single bit?

What allocator would you make up for this?

You are a poor kernel and you need lots of inodes

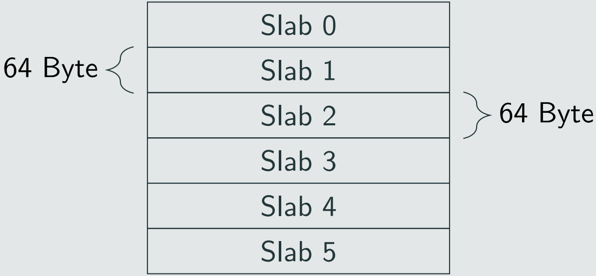
Every inode has the same size, 64 Byte. Can you think of any fast allocation strategy that does not waste a single bit?

Slab 0
Slab 1
Slab 2
Slab 3
Slab 4
Slab 5

What allocator would you make up for this?

You are a poor kernel and you need lots of inodes

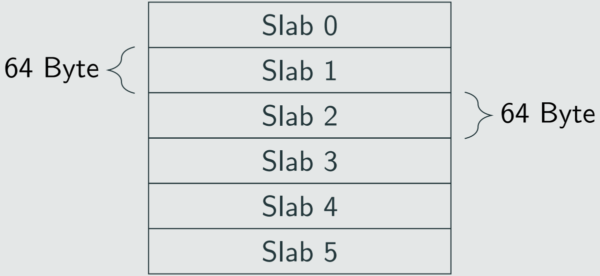
Every inode has the same size, 64 Byte. Can you think of any fast allocation strategy that does not waste a single bit?



What allocator would you make up for this?

You are a poor kernel and you need lots of inodes

Every inode has the same size, 64 Byte. Can you think of any fast allocation strategy that does not waste a single bit?



This is called a *Slab allocator*

Buddy Allocator

So far we've seen

- Consistent large blocks $\Rightarrow$ Low external, high internal fragmentation
- Fitted blocks $\Rightarrow$ High external, low internal fragmentation

Buddy Allocator

So far we've seen

- Consistent large blocks $\Rightarrow$ Low external, high internal fragmentation
- Fitted blocks $\Rightarrow$ High external, low internal fragmentation

Can we do better for some applications? Any ideas?

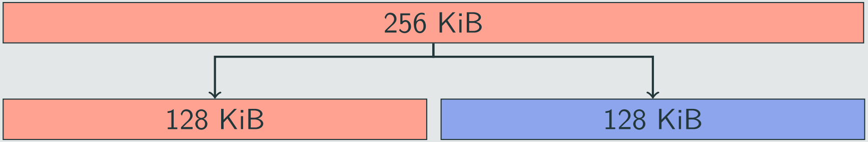
Buddy Allocator

Allocator

256 KiB

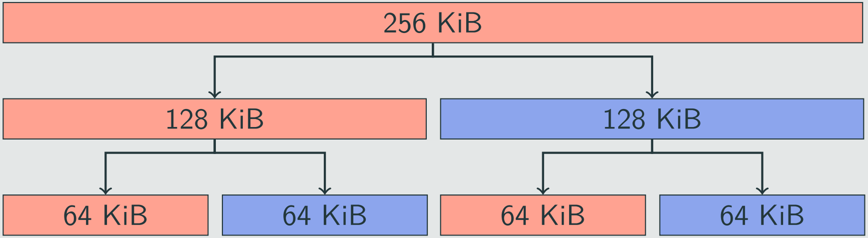
Buddy Allocator

Allocator



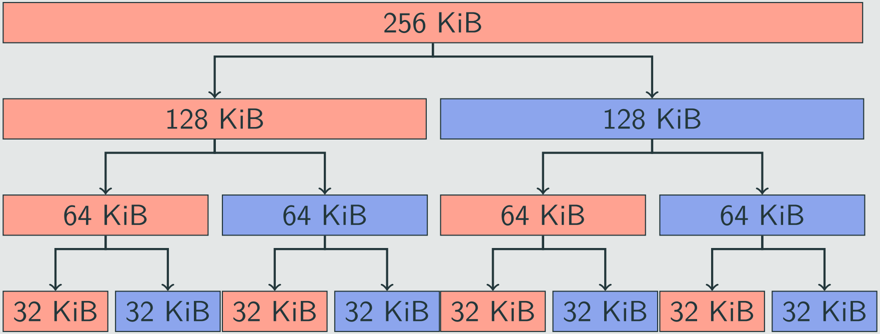
Buddy Allocator

Allocator

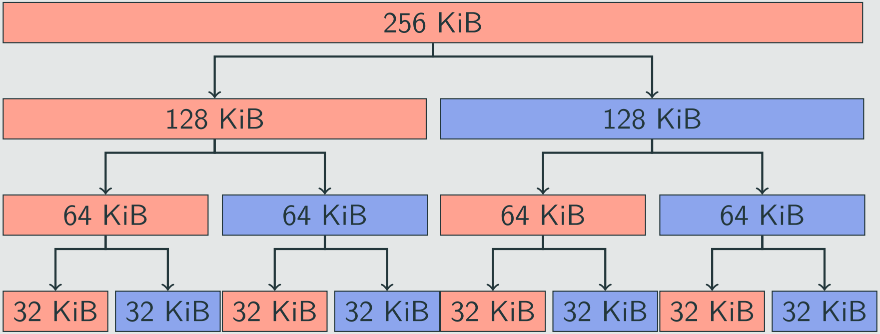


Buddy Allocator

Allocator



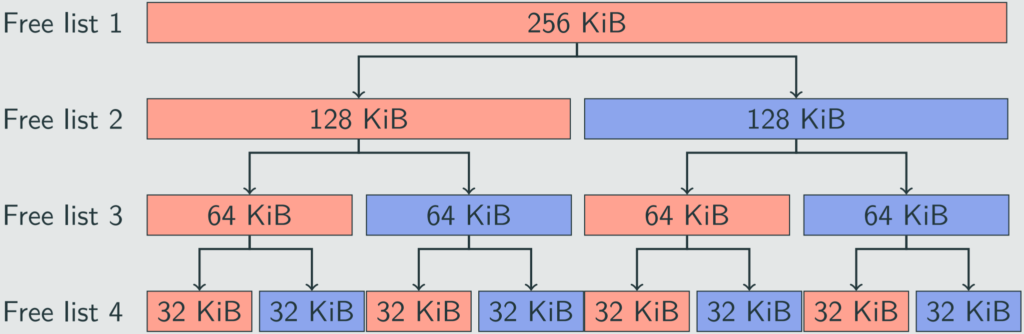
Allocator



How do you find a fitting Element?

Buddy Allocator

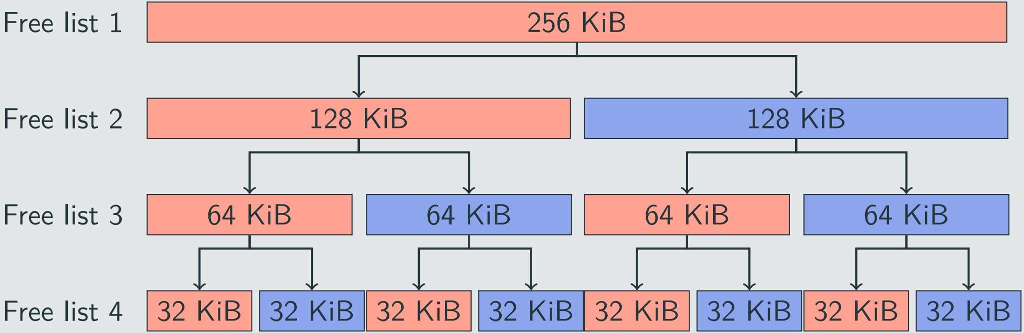
Allocator



How do you find a fitting Element? *Freelist!*

Buddy Allocator

Allocator

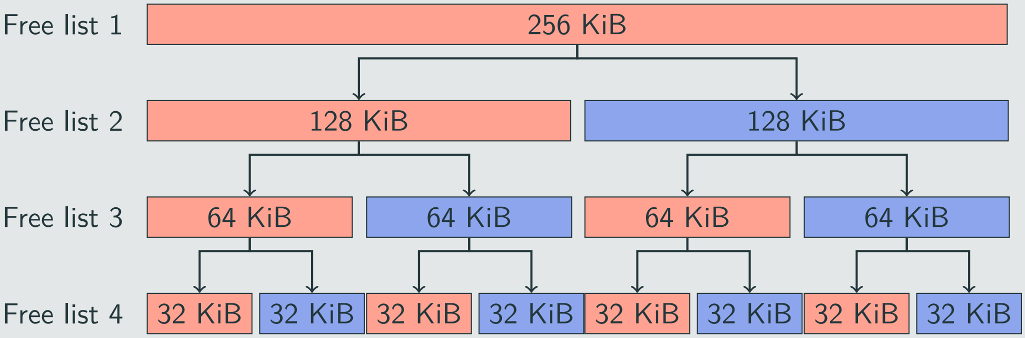


How do you find a fitting Element? *Freelist!*

And if there is no such block?

Buddy Allocator

Allocator

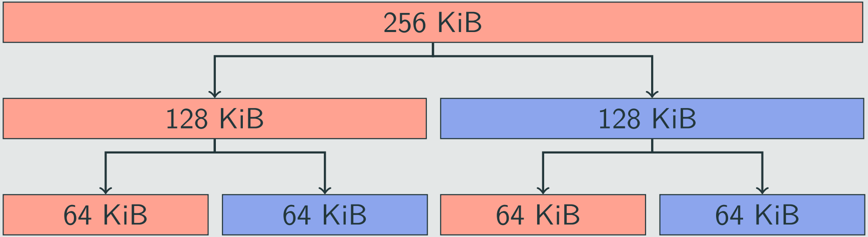


How do you find a fitting Element? *Freelist!*

And if there is no such block? *Recursively split a higher-up block*

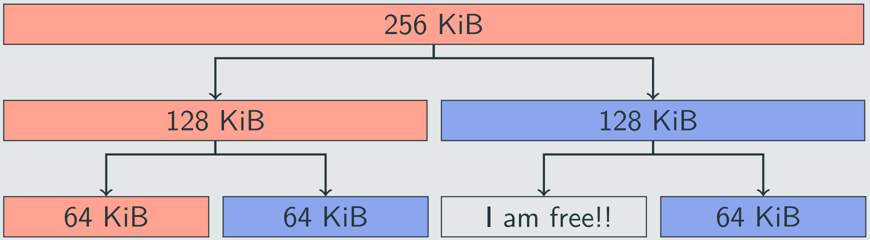
Buddy Allocator - Merging

Merging



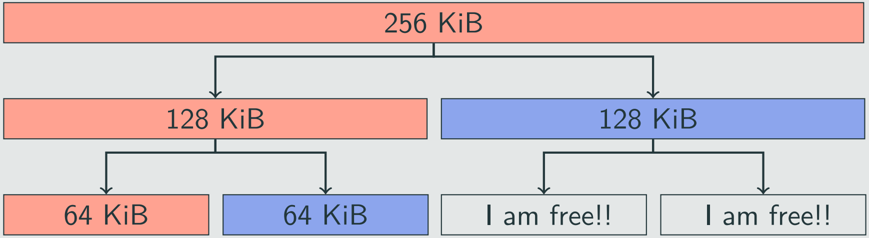
Buddy Allocator - Merging

Merging



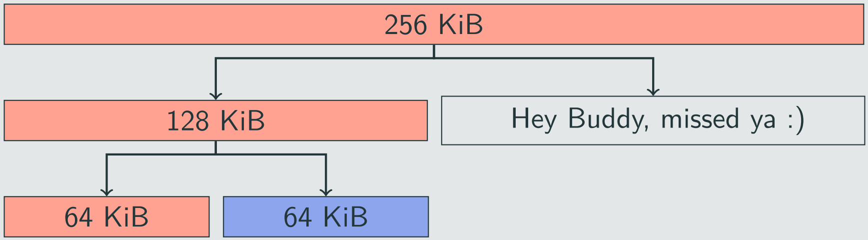
Buddy Allocator - Merging

Merging



Buddy Allocator - Merging

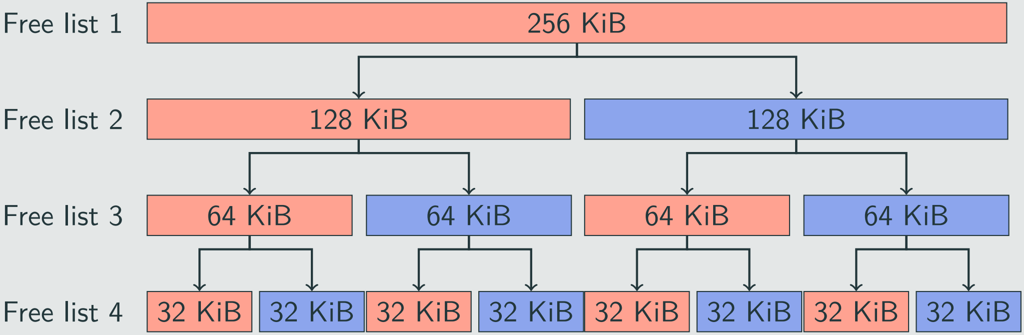
Merging



Buddy Allocator

How small/large can the free list be?

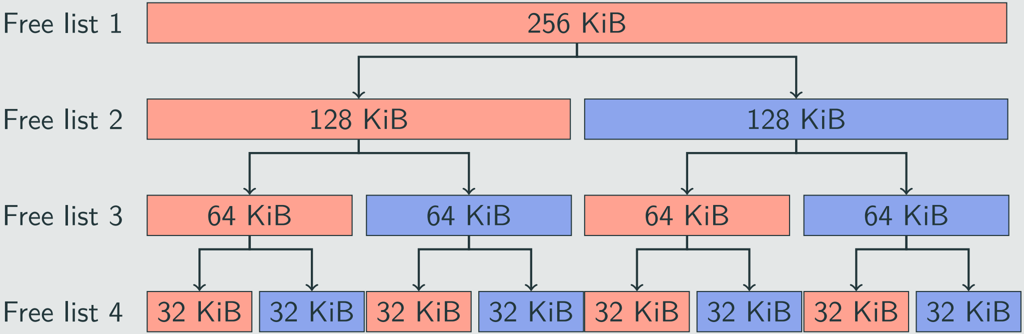
Allocate 2^m chunk of memory in a managed Block of 2^k (here: $k = 18$, as $256\ KiB = 2^{18}$)



Buddy Allocator

How small/large can the free list be?

Allocate 2^m chunk of memory in a managed Block of 2^k (here: $k = 18$, as $256\ KiB = 2^{18}$)



$\Rightarrow$ Max size $\frac{1}{2} \cdot 2^{k-m}$

$\Rightarrow$ Min size 0

What kind of fragmentation can occur?

Internal fragmentation

What kind of fragmentation can occur?

Internal fragmentation

- Power of two blocks

$\Rightarrow$ Request memory of size $2^k + 1, k \in \mathbb{N}_0$

What kind of fragmentation can occur?

Internal fragmentation

- Power of two blocks

$\Rightarrow$ Request memory of size $2^k + 1, k \in \mathbb{N}_0$

External fragmentation

What kind of fragmentation can occur?

Internal fragmentation

- Power of two blocks

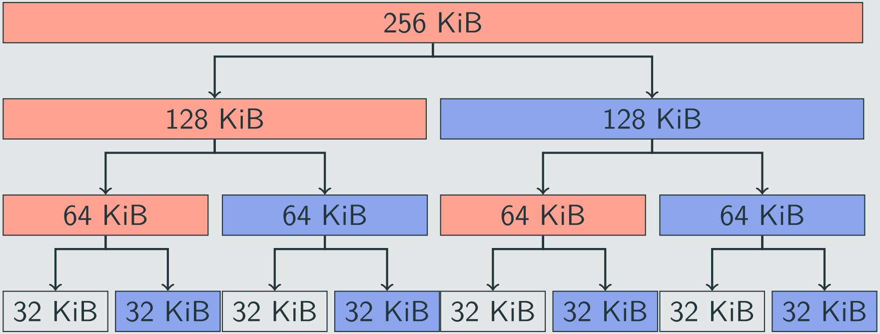
$\Rightarrow$ Request memory of size $2^k + 1, k \in \mathbb{N}_0$

External fragmentation

- Free every other block in a level

Buddy Allocator - Fragmentation

External fragmentation



Buddy Allocator - Fragmentation

But this works alright for larger sizes. So combine it with...

...

Buddy Allocator - Fragmentation

But this works alright for larger sizes. So combine it with...

...the Slab allocator! Allocate large chunks with the buddy allocator and small chunks within them using the slab allocators

Ende

XKCD 138 - Pointers

F R A G E N?

https://forms.gle/9CwJSKidKibubran9

Bis nächste Woche :)