https://forms.gle/xGn9PJcGdbbZMu2X7

Code: heap of fun

CS107, Lecture 15

Privacy and Trust

+

Managing The Heap

Reading: B&O 9.9, 9.11

Privacy and Trust

Our learning about assembly and program execution helps us better understand computer security (the protection of data, devices, and networks from disruption, harm, theft, unauthorized access or modification).

Computer security is important in part because it enables privacy.
In understanding computer security, it’s essential to understand the context in which it comes up (privacy and trust).

Privacy

What is privacy? 4 possible framings in two categories:

Individualist: the value of privacy as an individual right

- Privacy as **control of information** – controlling how our private information is shared with others.
- Privacy as **autonomy** – capacity to choose/decide for ourselves what is valuable.

Social: the value of privacy for a group

- Privacy as **social good** – social life would be unlivable without privacy.
- Privacy (protection) as based in **trust** – privacy enables trusting relationships

Privacy

Privacy as **control of information** – controlling how our information is communicated to others.

- Consent requires *free* choice with available alternatives and *informed* understanding of what is being offered.
    - How many of you just skip past the terms of service for new online services you sign up for?
    - Do you feel in control of your information with the services you choose to use? Why or why not? If you're working on a service, how can you respect privacy while achieving product goals?
- Control over personal data being collected (e.g. data exports from services you use, privacy dashboards, device privacy protections)

Privacy

Privacy as **autonomy** – capacity to choose/decide for ourselves what is valuable.

- Links to autonomy over our own lives and our ability to lead them as we choose.
- Do you feel that your autonomy is always respected when using products and services? Why or why not?

“[P]rivacy is valuable because it acknowledges our respect for persons as autonomous beings with the capacity to love, care and like—in other words, persons with the potential to freely develop close relationships” (Innes 1992)

Individualist Models of Privacy

Privacy as **autonomy** and privacy as **control over information** focus the value of privacy at an individual level.

Individual privacy can conflict with interests of society or the state.
Many debates over ”privacy vs. security” – whether one should be sacrificed for the other

- Apple v. FBI case re: unlocking iPhones
- Debates around encryption

Where do your beliefs fall in balancing privacy and security? When (if at all) is it ok to sacrifice one, and how much?

Privacy

Privacy as **social good** – social life would be unlivable without privacy.

Privacy has a social value in bringing about the kind of society we want to live in. What would society look like without privacy?

Privacy

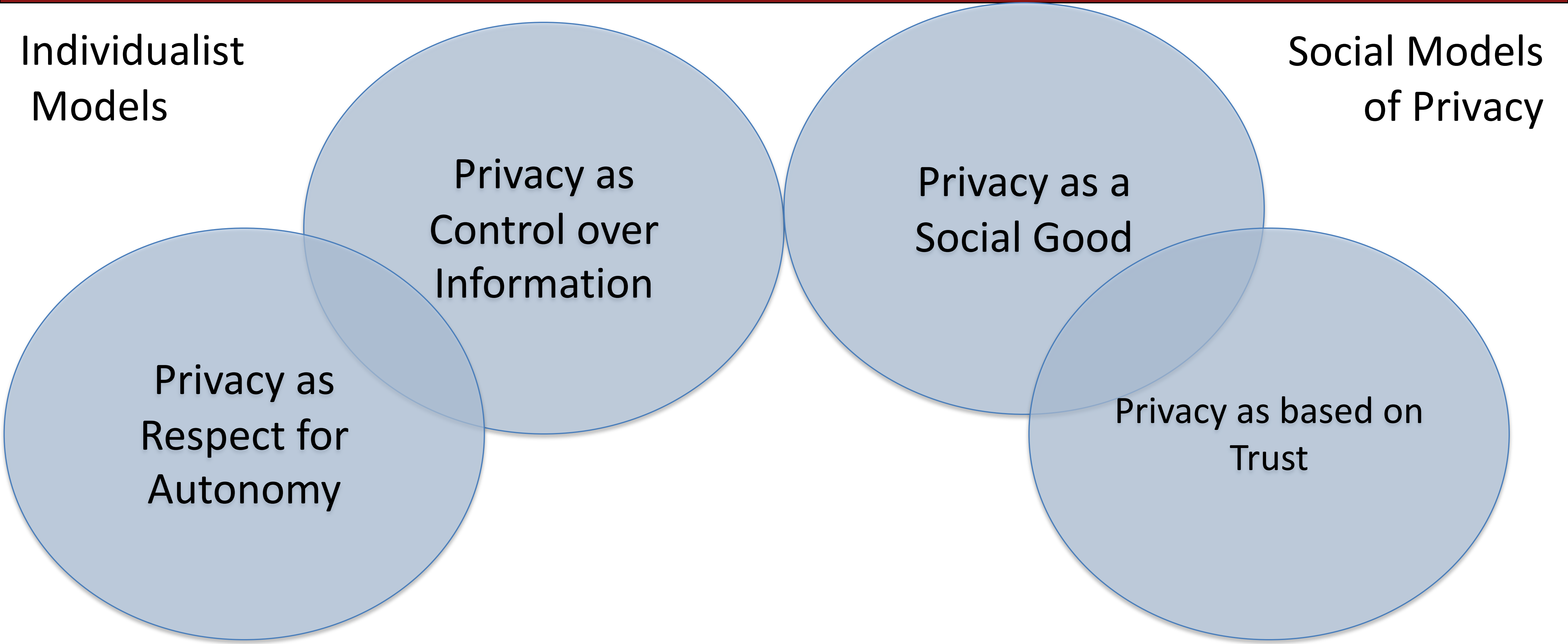
Privacy (protection) as based in **trust** – privacy enables trusting relationships

Privacy may help enable trusting relationships essential for cooperation.

> For instance, a *fiduciary*: someone who stands in a legal or ethical relationship of trust with another person (or group). The fiduciary must act for the benefit of and in the best interest of the other person. E.g. tax filer with access to your bank account
>
> > Should anyone who has access to personal info have a *fiduciary* responsibility? (Richards & Hartzog 2020).

This model of privacy stresses the essential relationship of trust placed in any holder of personal data and the responsibilities that result from this trust.

Models of Privacy


Individualist Models
Social Models of Privacy
Privacy as Control over Information
Privacy as Respect for Autonomy
Privacy as a Social Good
Privacy as based on Trust

Who Should We Trust?

Both security and privacy rely on trusted people (who administer security, perform penetration tests, submit vulnerabilities to databases, or keep private information secret). The final piece of the security puzzle is understanding trust.

Trust = Reliance + Risk of Betrayal

What makes trust unique to relationships between people is that trust exposes one to being *betrayed or being let down* (Baier 1986).

Penetration Testing & Trust

Penetration testing is the practice of encouraging or hiring security researchers / contractors to find vulnerabilities in one's own code or system. Position of trust – tester is given access to the system and encouraged to find exploitable vulnerabilities, expected to share what they have found with you. Means *relying on* their skill at finding vulnerabilities and *trusting* that their ethical compass will lead them to tell you and to act as a trustworthy *fiduciary* (guardian of your interests).

In Assignment5, you will have the opportunity to test your own ethical compass!

Loss of Privacy

Loss of privacy can cause us various harms, including:

Aggregation: combining personal information from various sources to build a profile of someone

Exclusion: not knowing how our information is being used, or being unable to access or modify it (Google removing personal info from search – link)

Secondary Use: using your information for purposes other than what was intended without permission.

Mitigation: Differential Privacy

Differential privacy is a formal measure of privacy for datasets to try and protect individuals from aggregation by making them harder to identify (Dwork 2008).

Imagine a large database, e.g., a medical database, with personal information and records of past activity tied to a name.

The records might be useful for research purposes, or to train a machine learning model to predict future health outcomes, but what if giving access to the records exposed the privacy of individual person’s health records?

Differential privacy adds inconsequential noise (e.g., changing a birthday from 2001 to 2002) or removes records to make individuals harder to identify while preserving the utility of the dataset overall.

Trust Models

In every evaluation of privacy, we can ask: who is trusted? Who is distrusted? Does this model concentrate trust (and therefore power) in a single individual or small group, or does it distribute trust?

Differential Privacy’s Trust Model

Differential privacy assumes that the only threat to privacy is an *external user querying the database* who must be prevented from aggregating data that could identify a user.

In other words, the *trust model* of differential privacy is that the database owners and maintainers are to be fully trusted, and no one else.

But is that the only threat? Differential privacy does not protect against improper use by people with full access to data or against leaks of the whole database, which may be the primary data exposure risks.

Differential privacy also does not question the assumption that amassing & storing large amounts of personal data is worth the risk of inevitable leaks (Rogaway 2015).

CS107 Topic 6: How do the core `malloc/realloc/free` memory-allocation operations work?

How do malloc/realloc/free work?

Pulling together all our CS107 topics this quarter:

- Testing
- Efficiency
- Bit-level manipulation
- Memory management
- Pointers
- Generics
- Assembly
- And more…

Learning Goals

- Learn the restrictions, goals and assumptions of a heap allocator
- Understand the conflicting goals of utilization and throughput
- Learn about different ways to implement a heap allocator

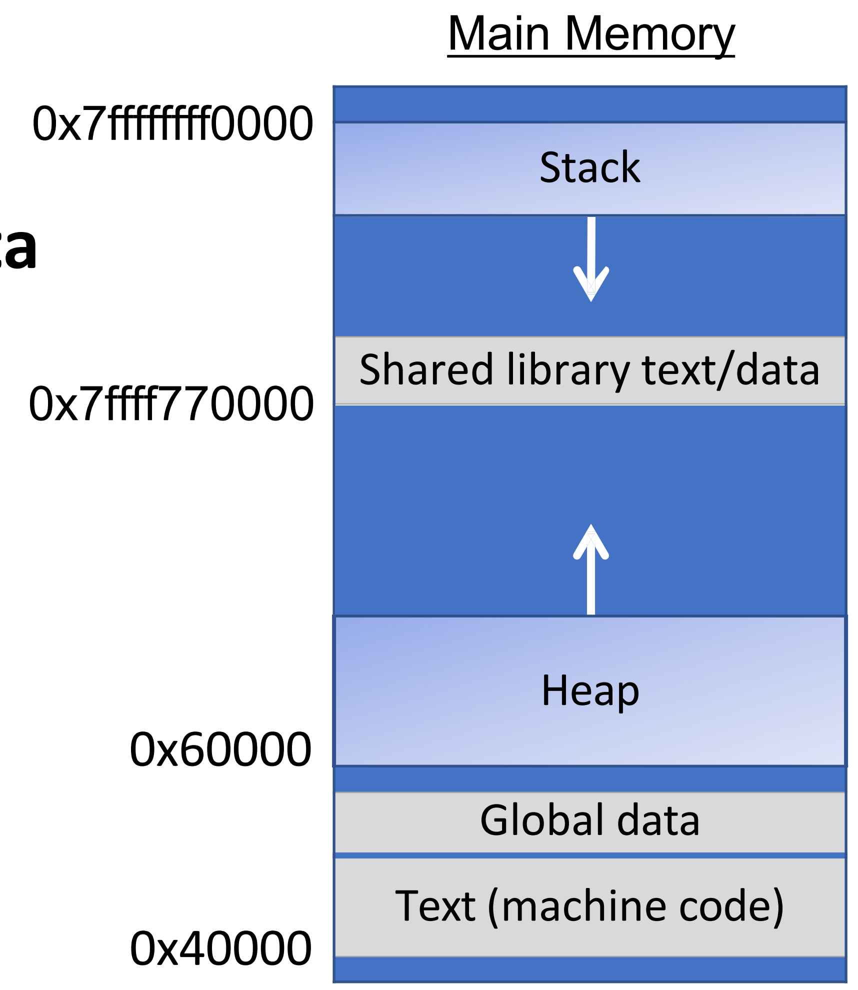
Lecture Plan

Lecture Plan

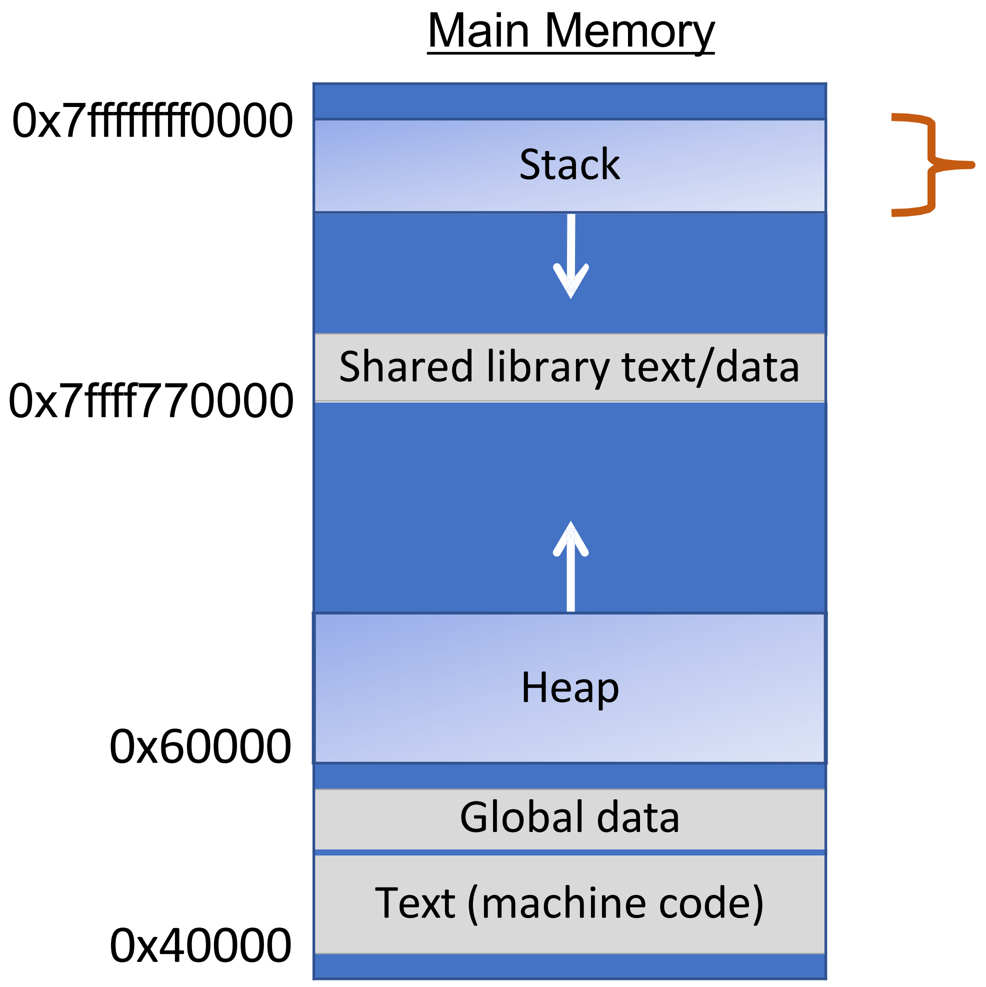
Running a program

- **Creates new process**
- **Sets up address space/segments**
- **Read executable file, load instructions, global data**
  Mapped from file into gray segments
- **Libraries loaded on demand**

- **Set up stack**
  Reserve stack segment, init %rsp, call main
- **malloc written in C, will init self on use**
  Asks OS for large memory region,
  parcels out to service requests



The Stack

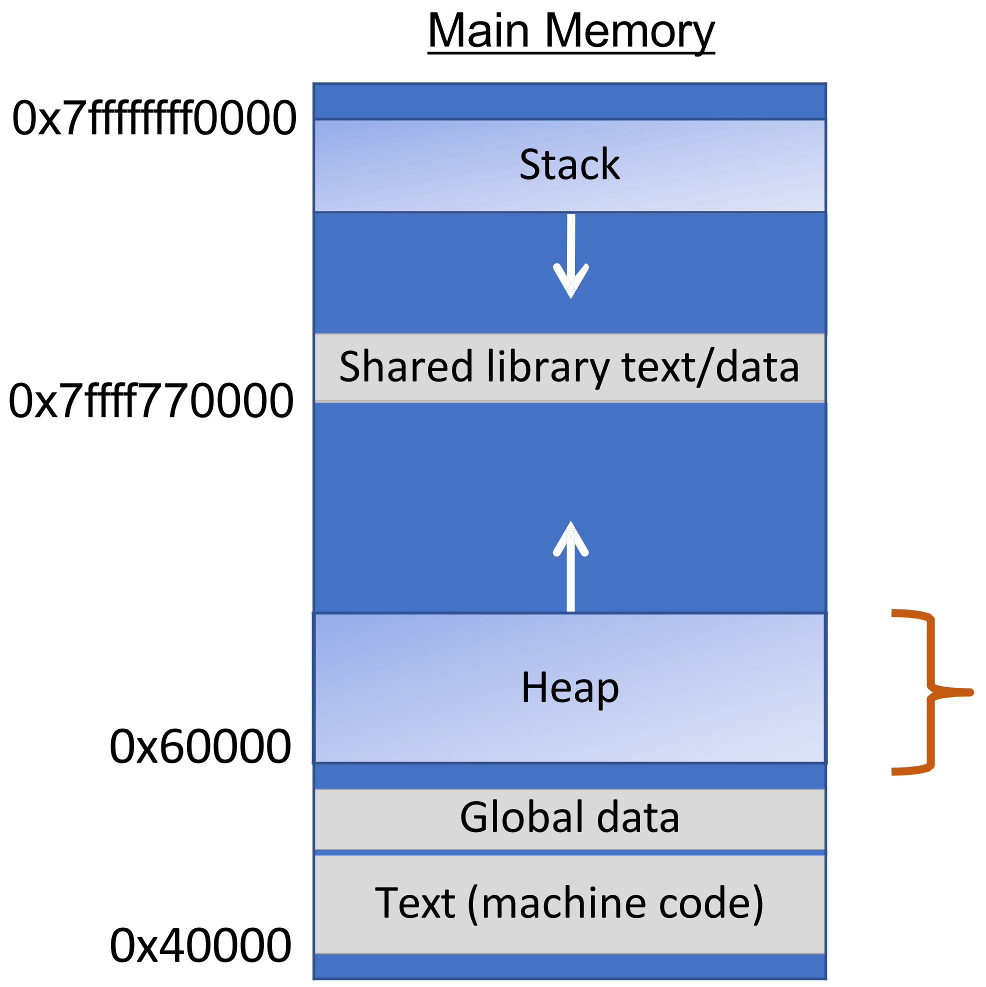


Stack memory ”goes away” after function call ends.

Automatically managed at compile-time by gcc

Last lecture:
Stack management == moving %rsp around (pushq, popq, mov)

Today: The Heap



Heap memory persists until caller indicates it no longer needs it.

Managed by C standard library functions (malloc, realloc, free)

This lecture:
How does heap management work?

Lecture Plan

Your role so far: Client

```
void *malloc(size_t size);
```

Returns a pointer to a block of heap memory of at least size bytes, or NULL if an error occurred.

```
void free(void *ptr);
```

Frees the heap-allocated block starting at the specified address.

```
void *realloc(void *ptr, size_t size);
```

Changes the size of the heap-allocated block starting at the specified address to be the new specified size. Returns the address of the new, larger allocated memory region.

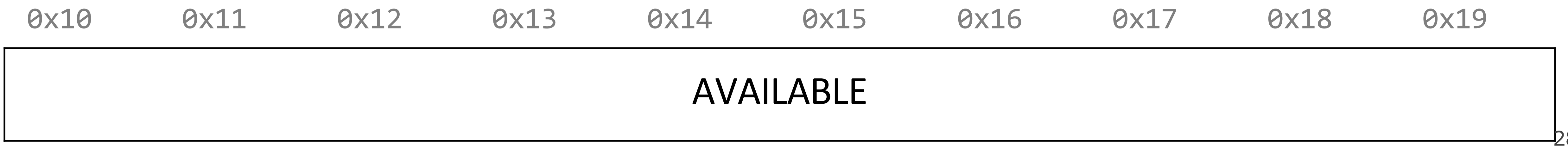
Your role now: Heap Hotel Concierge

(aka **Heap Allocator**)

http://screencrave.com/wp-content/uploads/2014/03/the-grand-budapest-hotel-anderson-image-2.jpg

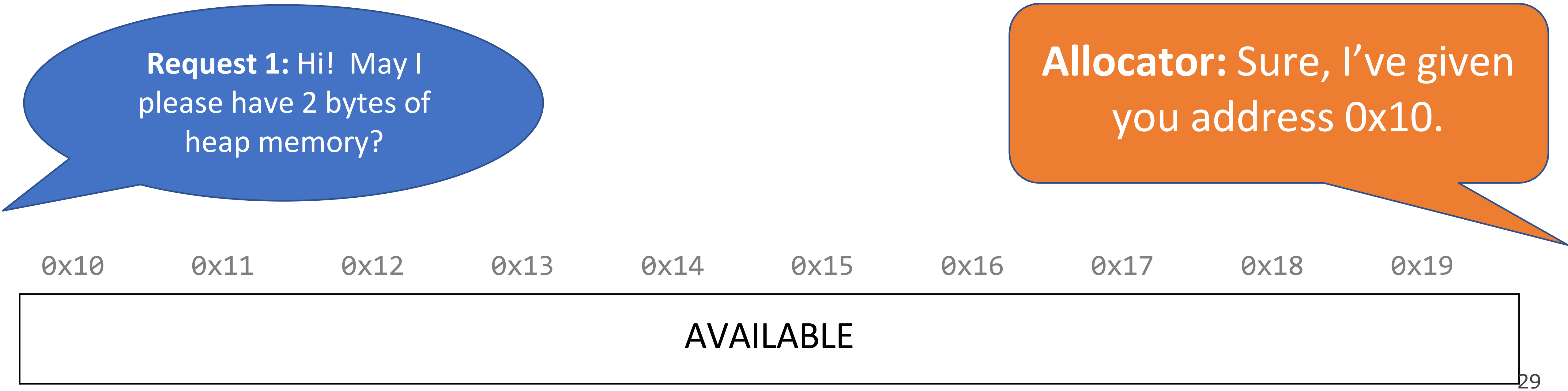
What is a heap allocator?

- A heap allocator is a set of functions that fulfills requests for heap memory.
- On initialization, a heap allocator is provided the starting address and size of a large contiguous block of memory (the heap).



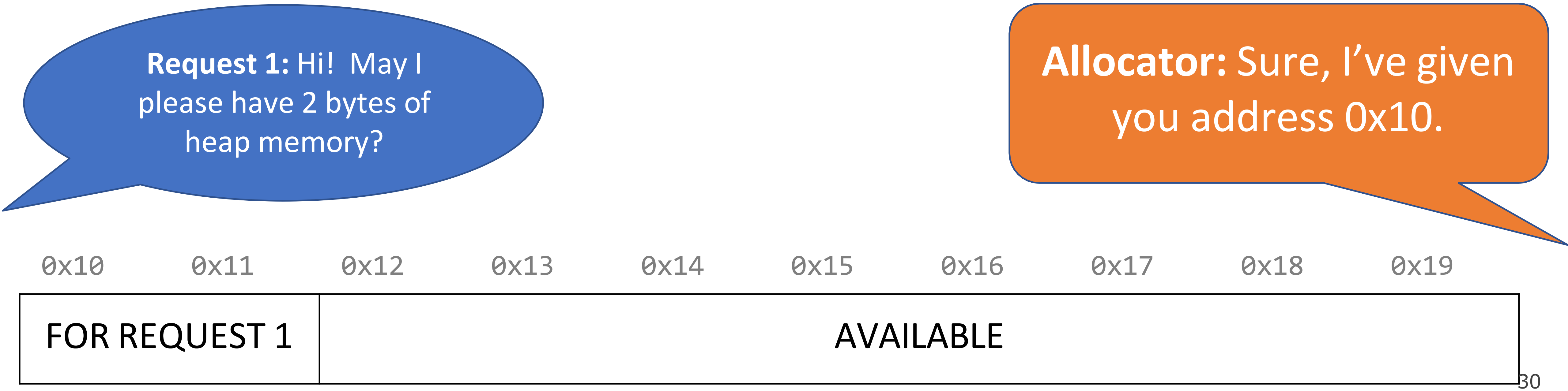
What is a heap allocator?

- A heap allocator is a set of functions that fulfills requests for heap memory.
- On initialization, a heap allocator is provided the starting address and size of a large contiguous block of memory (the heap).
- A heap allocator must manage this memory as clients request or no longer need pieces of it.


Request 1: Hi! May I please have 2 bytes of heap memory?
Allocator: Sure, I've given you address 0x10.
0x10
0x11
0x12
0x13
0x14
0x15
0x16
0x17
0x18
0x19
AVAILABLE

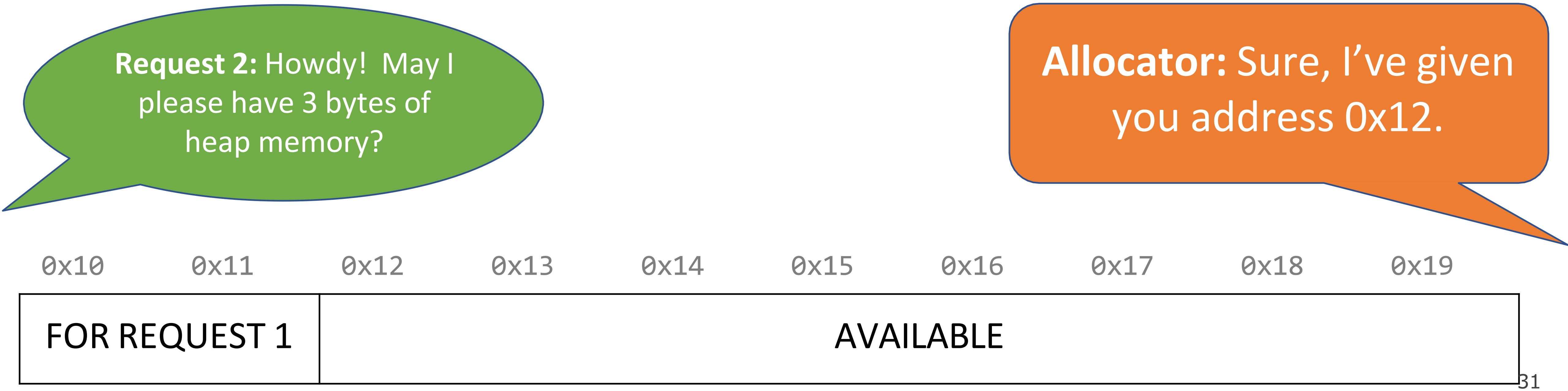
What is a heap allocator?

- A heap allocator is a set of functions that fulfills requests for heap memory.
- On initialization, a heap allocator is provided the starting address and size of a large contiguous block of memory (the heap).
- A heap allocator must manage this memory as clients request or no longer need pieces of it.


Request 1: Hi! May I please have 2 bytes of heap memory?
Allocator: Sure, I've given you address 0x10.
0x10
0x11
0x12
0x13
0x14
0x15
0x16
0x17
0x18
0x19
FOR REQUEST 1
AVAILABLE

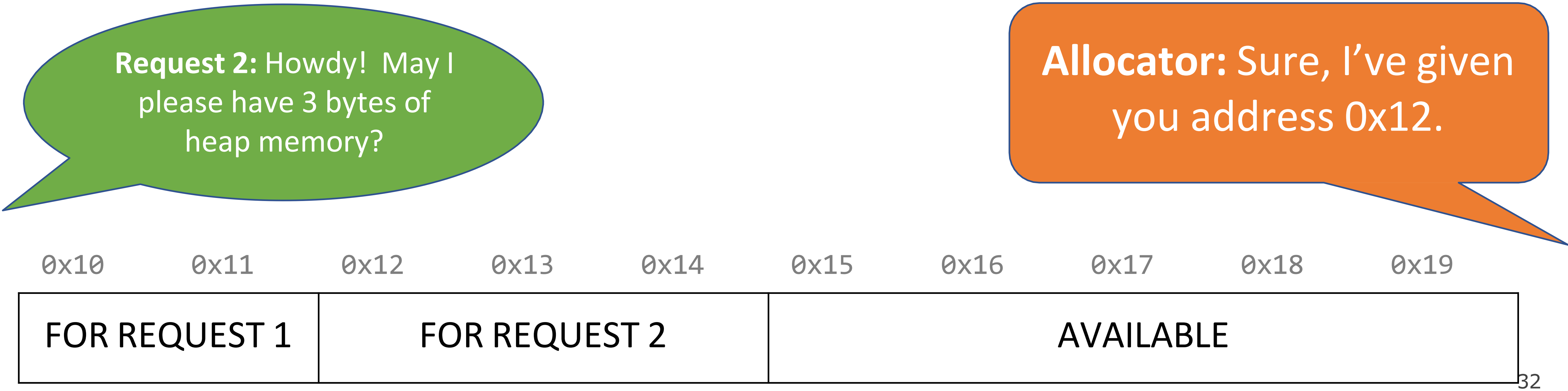
What is a heap allocator?

- A heap allocator is a set of functions that fulfills requests for heap memory.
- On initialization, a heap allocator is provided the starting address and size of a large contiguous block of memory (the heap).
- A heap allocator must manage this memory as clients request or no longer need pieces of it.


Request 2: Howdy! May I please have 3 bytes of heap memory?
Allocator: Sure, I've given you address 0x12.
0x10
0x11
0x12
0x13
0x14
0x15
0x16
0x17
0x18
0x19
FOR REQUEST 1
AVAILABLE

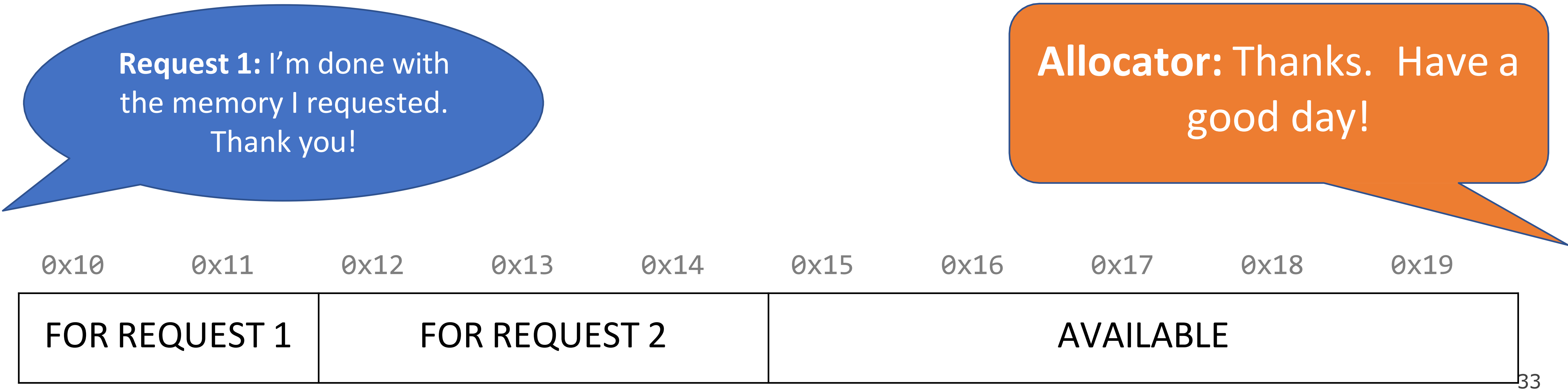
What is a heap allocator?

- A heap allocator is a set of functions that fulfills requests for heap memory.
- On initialization, a heap allocator is provided the starting address and size of a large contiguous block of memory (the heap).
- A heap allocator must manage this memory as clients request or no longer need pieces of it.


Request 2: Howdy! May I please have 3 bytes of heap memory?
Allocator: Sure, I've given you address 0x12.
0x10
0x11
0x12
0x13
0x14
0x15
0x16
0x17
0x18
0x19
FOR REQUEST 1
FOR REQUEST 2
AVAILABLE

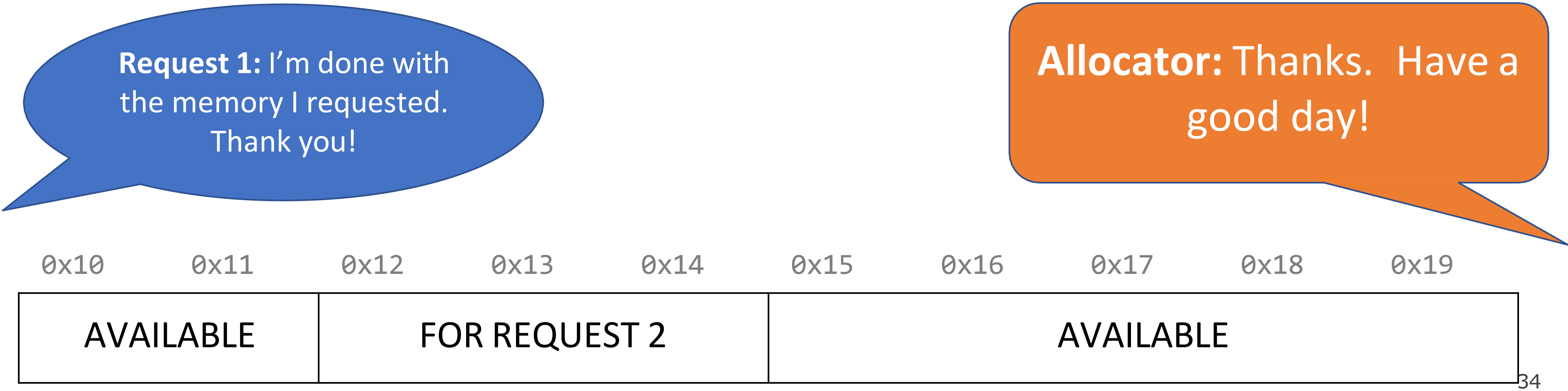
What is a heap allocator?

- A heap allocator is a set of functions that fulfills requests for heap memory.
- On initialization, a heap allocator is provided the starting address and size of a large contiguous block of memory (the heap).
- A heap allocator must manage this memory as clients request or no longer need pieces of it.


Request 1: I'm done with the memory I requested. Thank you!
Allocator: Thanks. Have a good day!
0x10
0x11
0x12
0x13
0x14
0x15
0x16
0x17
0x18
0x19
FOR REQUEST 1
FOR REQUEST 2
AVAILABLE

What is a heap allocator?

- A heap allocator is a set of functions that fulfills requests for heap memory.
- On initialization, a heap allocator is provided the starting address and size of a large contiguous block of memory (the heap).
- A heap allocator must manage this memory as clients request or no longer need pieces of it.

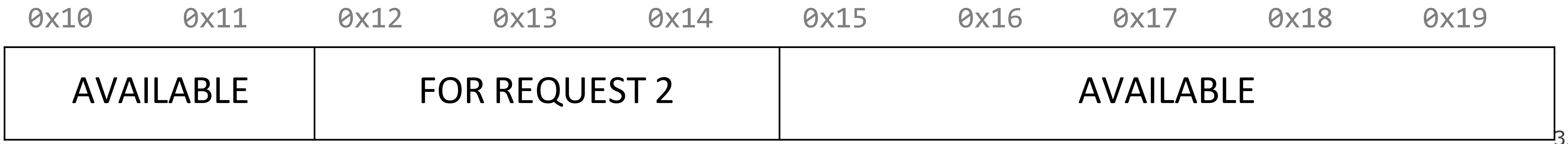

Request 1: I'm done with the memory I requested. Thank you!
Allocator: Thanks. Have a good day!
0x10
0x11
0x12
0x13
0x14
0x15
0x16
0x17
0x18
0x19
AVAILABLE
FOR REQUEST 2
AVAILABLE

What is a heap allocator?

- A heap allocator is a set of functions that fulfills requests for heap memory.
- On initialization, a heap allocator is provided the starting address and size of a large contiguous block of memory (the heap).
- A heap allocator must manage this memory as clients request or no longer need pieces of it.

Request 3: Hello there! I'd like to request 2 bytes of heap memory, please.

Allocator: Sure thing. I've given you address 0x10.

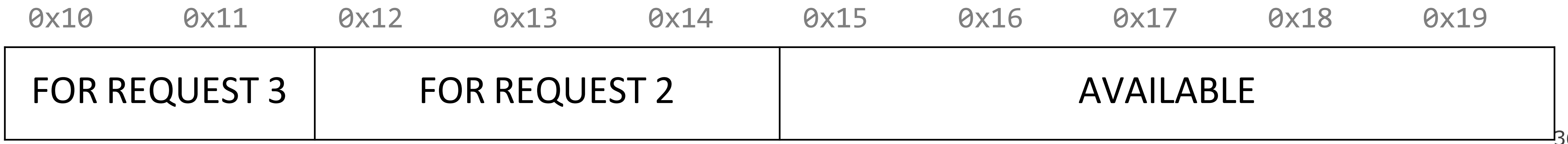


What is a heap allocator?

- A heap allocator is a set of functions that fulfills requests for heap memory.
- On initialization, a heap allocator is provided the starting address and size of a large contiguous block of memory (the heap).
- A heap allocator must manage this memory as clients request or no longer need pieces of it.

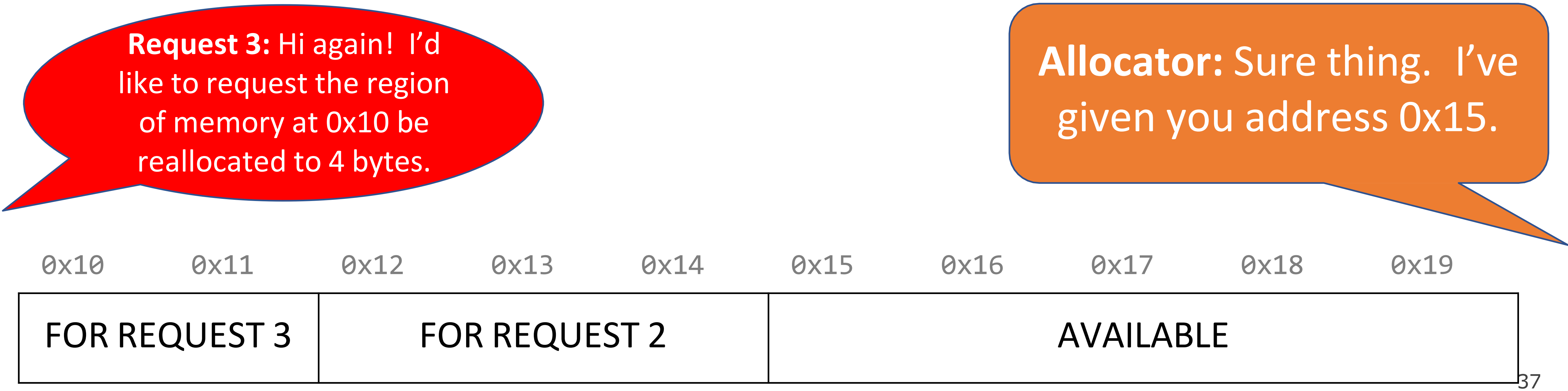
Request 3: Hello there! I'd like to request 2 bytes of heap memory, please.

Allocator: Sure thing. I've given you address 0x10.



What is a heap allocator?

- A heap allocator is a set of functions that fulfills requests for heap memory.
- On initialization, a heap allocator is provided the starting address and size of a large contiguous block of memory (the heap).
- A heap allocator must manage this memory as clients request or no longer need pieces of it.

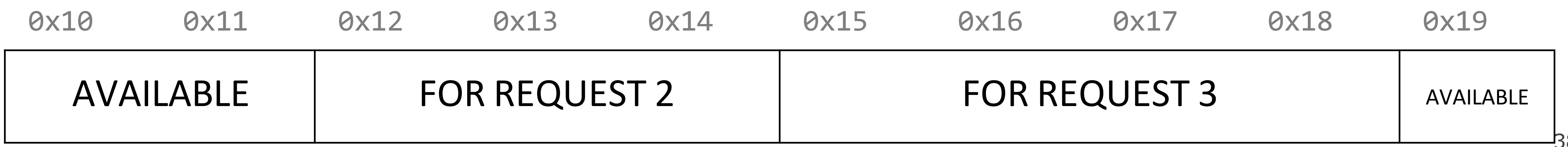

Request 3: Hi again! I'd like to request the region of memory at 0x10 be reallocated to 4 bytes.
Allocator: Sure thing. I've given you address 0x15.
0x10
0x11
0x12
0x13
0x14
0x15
0x16
0x17
0x18
0x19
FOR REQUEST 3
FOR REQUEST 2
AVAILABLE

What is a heap allocator?

- A heap allocator is a set of functions that fulfills requests for heap memory.
- On initialization, a heap allocator is provided the starting address and size of a large contiguous block of memory (the heap).
- A heap allocator must manage this memory as clients request or no longer need pieces of it.

Request 3: Hi again! I'd like to request the region of memory at 0x10 be reallocated to 4 bytes.

Allocator: Sure thing. I've given you address 0x15.



Lecture Plan

Heap Allocator Functions

```
void *malloc(size_t size);

void free(void *ptr);

void *realloc(void *ptr, size_t size);
```

Heap Allocator Requirements

A heap allocator must…

1. Handle arbitrary request sequences of allocations and frees
2. Keep track of which memory is allocated and which is available
3. Decide which memory to provide to fulfill an allocation request
4. Immediately respond to requests without delay

Heap Allocator Requirements

A heap allocator must…

1. **Handle arbitrary request sequences of allocations and frees**
2. Keep track of which memory is allocated and which is available
3. Decide which memory to provide to fulfill an allocation request
4. Immediately respond to requests without delay

A heap allocator cannot assume anything about the order of allocation and free requests, or even that every allocation request is accompanied by a matching free request.

Heap Allocator Requirements

A heap allocator must…

1. Handle arbitrary request sequences of allocations and frees
2. **Keep track of which memory is allocated and which is available**
3. Decide which memory to provide to fulfill an allocation request
4. Immediately respond to requests without delay

A heap allocator marks memory regions as **allocated** or **available**. It must remember which is which to properly provide memory to clients.

Heap Allocator Requirements

A heap allocator must…

1. Handle arbitrary request sequences of allocations and frees
2. Keep track of which memory is allocated and which is available
3. **Decide which memory to provide to fulfill an allocation request**
4. Immediately respond to requests without delay

A heap allocator may have options for which memory to use to fulfill an allocation request. It must decide this based on a variety of factors.

Heap Allocator Requirements

A heap allocator must…

1. Handle arbitrary request sequences of allocations and frees
2. Keep track of which memory is allocated and which is available
3. Decide which memory to provide to fulfill an allocation request
4. **Immediately respond to requests without delay**

A heap allocator must respond immediately to allocation requests and should not e.g. prioritize or reorder certain requests to improve performance.

Heap Allocator Requirements

A heap allocator must…

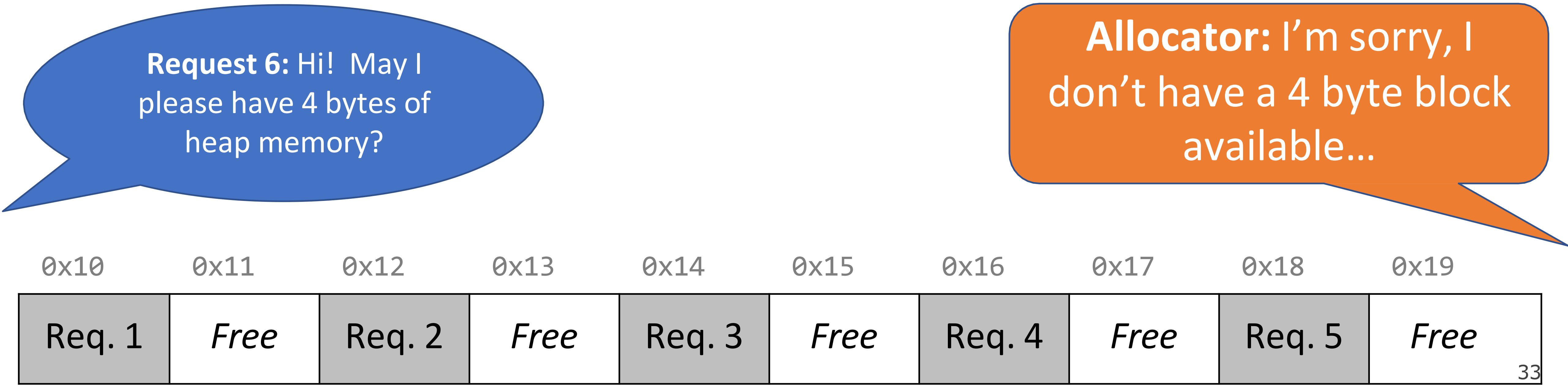
1. Handle arbitrary request sequences of allocations and frees
2. Keep track of which memory is allocated and which is available
3. Decide which memory to provide to fulfill an allocation request
4. Immediately respond to requests without delay
5. **Return addresses that are 8-byte-aligned (must be multiples of 8).**

Heap Allocator Goals

- Goal 1: Maximize **throughput**, or the number of requests completed per unit time. This means minimizing the average time to satisfy a request.
- Goal 2: Maximize memory **utilization**, or how efficiently we make use of the limited heap memory to satisfy requests.

Utilization

- The primary cause of poor utilization is **fragmentation**. **Fragmentation** occurs when otherwise unused memory is not available to satisfy allocation requests.
- In this example, there is enough aggregate free memory to satisfy the request, but no single free block is large enough to handle the request.
- In general: we want the largest address used to be as low as possible.


Request 6: Hi! May I please have 4 bytes of heap memory?
Allocator: I'm sorry, I don't have a 4 byte block available…
0x10
0x11
0x12
0x13
0x14
0x15
0x16
0x17
0x18
0x19
Req. 1
Free
Req. 2
Free
Req. 3
Free
Req. 4
Free
Req. 5
Free

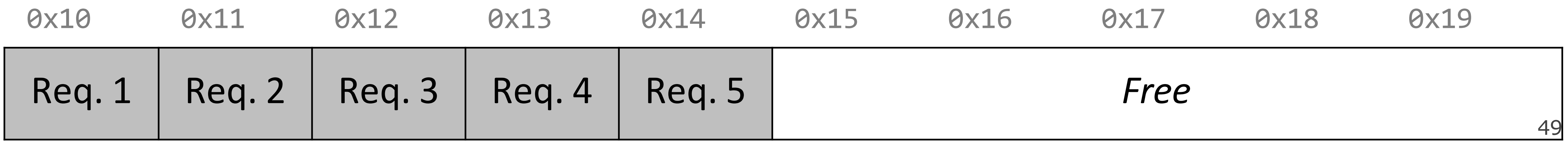
Utilization

Question: what if we shifted these blocks down to make more space? Can we do this?

A. YES, great idea!

B. YES, it can be done, but not a good idea for some reason (e.g. not efficient use of time)

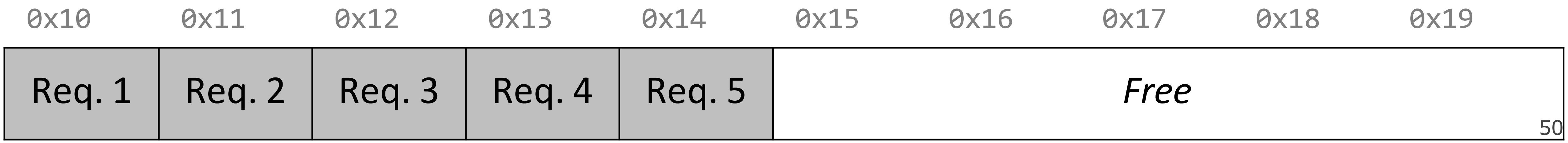
C. NO, it can't be done!



Utilization

Question: what if we shifted these blocks down to make more space? Can we do this?

- **No -** we have already guaranteed these addresses to the client. We cannot move allocated memory around, since this will mean the client will now have incorrect pointers to their memory!



Fragmentation

- **Internal Fragmentation**: an allocated block is larger than what is needed (e.g. due to minimum block size)
- **External Fragmentation**: no single block is large enough to satisfy an allocation request, even though enough aggregate free memory is available

Heap Allocator Goals

- Goal 1: Maximize **throughput**, or the number of requests completed per unit time. This means minimizing the average time to satisfy a request.
- Goal 2: Maximize memory **utilization**, or how efficiently we make use of the limited heap memory to satisfy requests.

These are seemingly conflicting goals – for instance, it may take longer to better plan out heap memory use for each request.

Heap allocators must find an appropriate balance between these two goals!

Heap Allocator Goals

- Goal 1: Maximize **throughput**, or the number of requests completed per unit time. This means minimizing the average time to satisfy a request.
- Goal 2: Maximize memory **utilization**, or how efficiently we make use of the limited heap memory to satisfy requests.

Other desirable goals:
Locality ("similar" blocks allocated close in space)
Robust (handle client errors)
Ease of implementation/maintenance

Lecture Plan

Bump Allocator

Let’s say we want to entirely prioritize throughput, and do not care about utilization at all. This means we do not care about reusing memory. How could we do this?

Bump Allocator Performance

1. Utilization

😱

Never reuses memory

2. Throughput

⭐

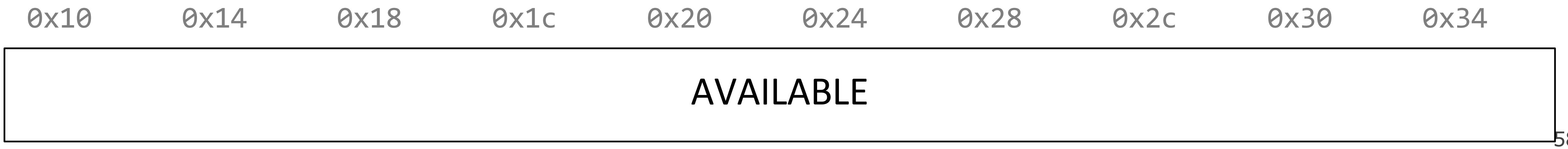
Ultra fast, short rouines

Bump Allocator

- A **bump allocator** is a heap allocator design that simply allocates the next available memory address upon an allocate request and does nothing on a free request.
- Throughput: each **malloc** and **free** execute only a handful of instructions:
  - It is easy to find the next location to use
  - Free does nothing!
- Utilization: we use each memory block at most once. No freeing at all, so no memory is ever reused. ®
- We provide a bump allocator implementation as part of the final project as a code reading exercise.

Bump Allocator

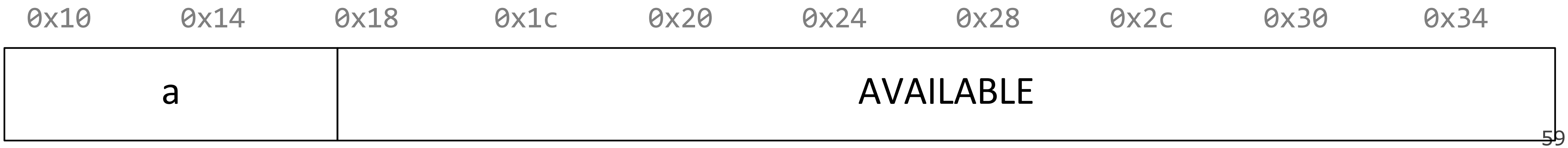
```
void *a = malloc(8);
void *b = malloc(4);
void *c = malloc(24);
free(b);
void *d = malloc(8);
```



Bump Allocator

```
void *a = malloc(8);
void *b = malloc(4);
void *c = malloc(24);
free(b);
void *d = malloc(8);
```

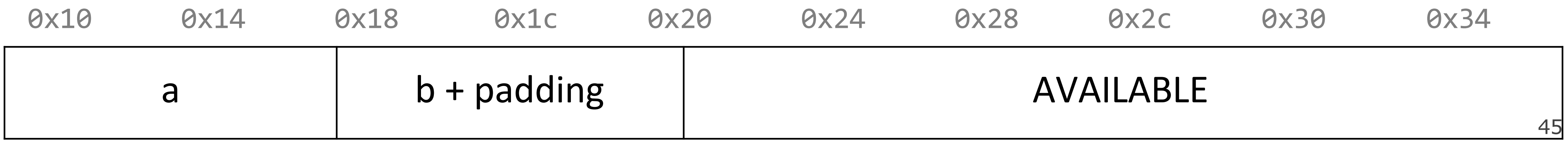
Variable	Value
a	0x10



Bump Allocator

```
void *a = malloc(8);
void *b = malloc(4);
void *c = malloc(24);
free(b);
void *d = malloc(8);
```

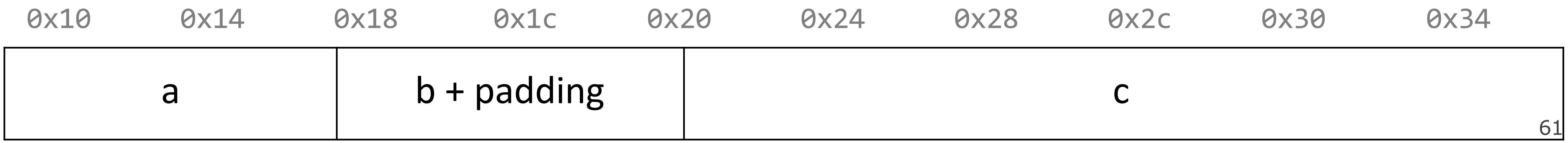
Variable	Value
a	0x10
b	0x18



Bump Allocator

```
void *a = malloc(8);
void *b = malloc(4);
void *c = malloc(24);
free(b);
void *d = malloc(8);
```

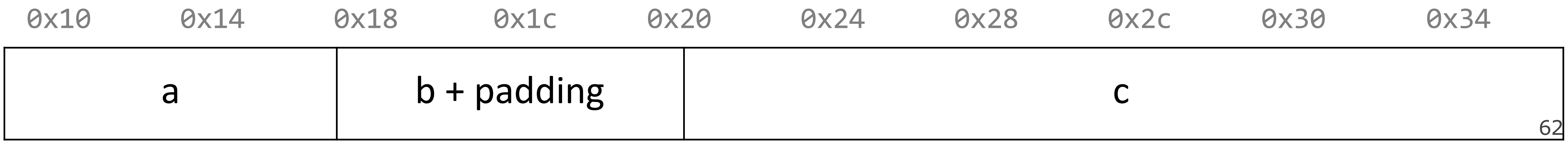
Variable	Value
a	0x10
b	0x18
c	0x20



Bump Allocator

```
void *a = malloc(8);
void *b = malloc(4);
void *c = malloc(24);
free(b);
void *d = malloc(8);
```

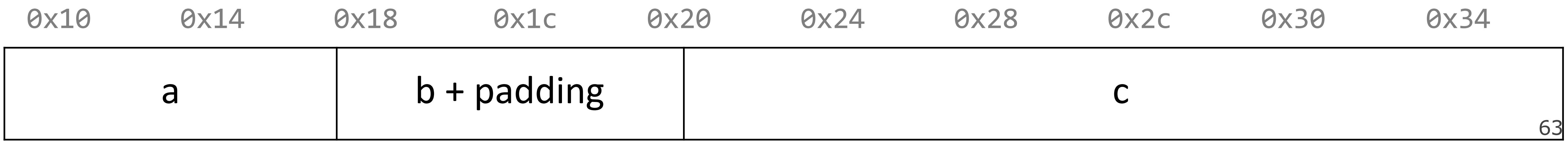
Variable	Value
a	0x10
b	0x18
c	0x20



Bump Allocator

```
void *a = malloc(8);
void *b = malloc(4);
void *c = malloc(24);
free(b);
void *d = malloc(8);
```

Variable	Value
a	0x10
b	0x18
c	0x20
d	NULL



Summary: Bump Allocator

- A bump allocator is an extreme heap allocator – it optimizes only for **throughput**, not **utilization**.
- Better allocators strike a more reasonable balance.   How can we do this?

Questions to consider:

1. How do we keep track of free blocks?
2. How do we choose an appropriate free block in which to place a newly allocated block?
3. After we place a newly allocated block in some free block, what do we do with the remainder of the free block?
4. What do we do with a block that has just been freed?

Lecture Plan

Implicit Free List Allocator

- **Key idea:** in order to reuse blocks, we need a way to track which blocks are allocated and which are free.
- We could store this information in a separate global data structure, but this is inefficient.
- Instead: let's allocate extra space before each block for a **header** storing its payload size and whether it is allocated or free.
- When we allocate a block, we look through the blocks to find a free one, and we update its header to reflect its allocated size and that it is now allocated.
- When we free a block, we update its header to reflect it is now free.
- The header should be 8 bytes (or larger).
- By storing the block size of each block, we *implicitly* have a *list* of free blocks.