CS107 Lecture 20
Reverse Engineering, Privacy and Trust
Managing Heap: Preamble

Reading: None!

Privacy and Trust

How does a computer interpret and execute C programs?

Why is answering this question important?

- Learning how our code is really translated and executed helps us write better code
- **We can learn how to reverse engineer and exploit programs at the assembly level**

Privacy and Trust

- Learning about machine code and program execution helps us better understand computer security.
- Computer security (the protection of data, devices, and networks from disruption, harm, theft, unauthorized access or modification) is important in part because it helps us maintain privacy and trust.

Have you been affected by a data breach/hack or some other unauthorized access to your data?

How did that make you feel?

Privacy

What is privacy? Here are 4 ways of framing it:

- Privacy as **control of information**: controlling how our private information is shared with others.
- Privacy as **autonomy**: the agency to decide for ourselves what is valuable.
- Privacy as **social good**: social life would be severely compromised without privacy.
- Privacy as a display of **trust**: privacy enables trusting relationships.

First two are *individualist* –the value of privacy as an individual right.

Second two are *social* – the value of privacy for a group.

Privacy

Privacy as **control of information**: controlling how our private information is shared with 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 some new online service? What are you surrendering by simply agreeing to it?
- Control over personal data collection and aggregation (e.g., data exports from services you use, privacy dashboards, device privacy protections)

Privacy

Privacy as **autonomy**: the agency to decide for ourselves what is valuable.

- Pertains to the autonomy over our own lives and our ability to lead them as we choose.
- Do you feel your autonomy is consistently 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 on the value of privacy at an individual level.

- Individual privacy can conflict with interests of state or society.
- Many debates over "privacy vs. security" – whether one should be sacrificed for the other
  - Apple v. FBI case re: unlocking iPhones (link)
  - Debates around encryption (link)
  - Privacy in the age of surveillance (link)

Privacy

Privacy as **social good**: social life would be severely compromised without privacy.

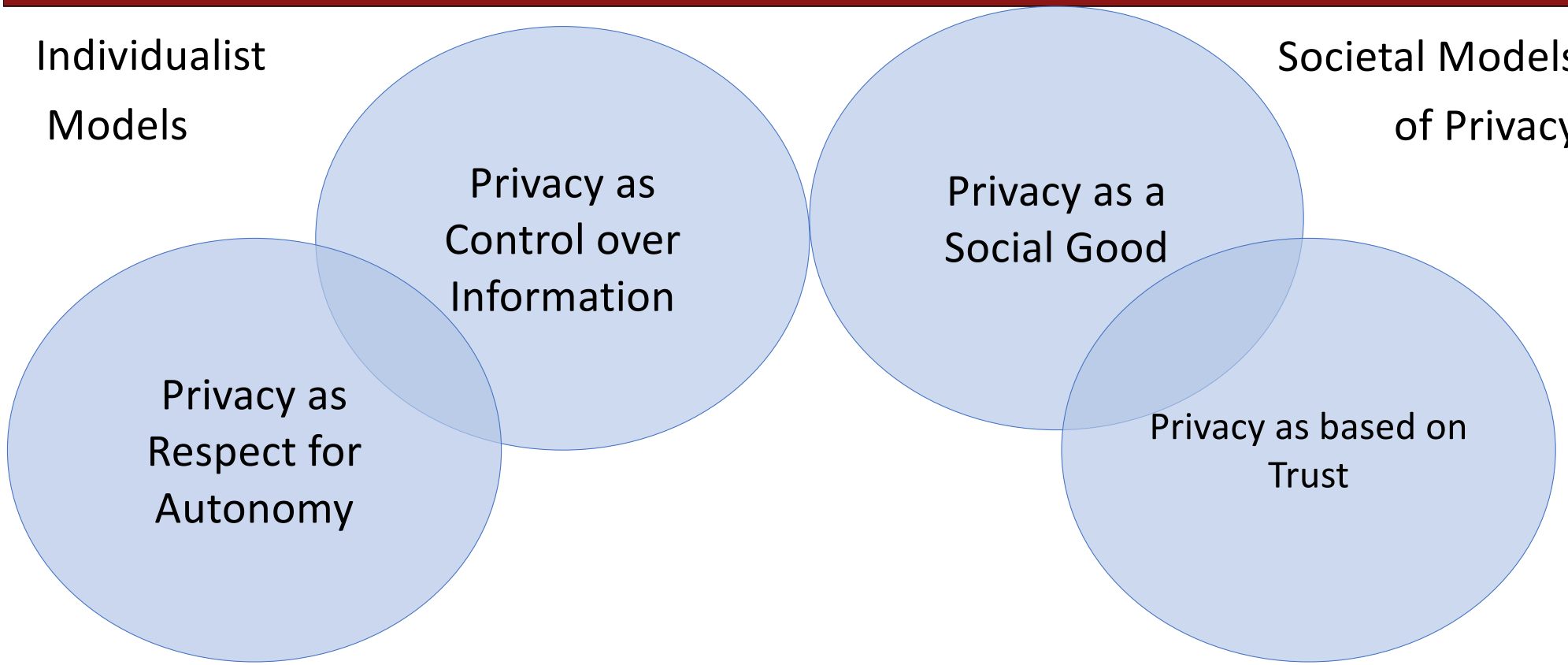
- Privacy provides a social value strongly influencing the kind of society we live in.
- What would society look like without privacy?

Privacy

Privacy as display of **trust**: privacy enables trusting relationships.

- Privacy enables trusting relationships essential to 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 between trust placed in any holder of personal data and responsibilities that come with this trust.

Models of Privacy



Loss of Privacy

Loss of privacy can cause us various forms harms, including:

- ***Aggregation*:** combining personal information from various sources to build someone's profile
- ***Exclusion:*** not knowing or understanding 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.

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 even hiring security researchers to find vulnerabilities in one's own code or system.

The tester is placed in a position of trust: they are given access to the system itself and encouraged to find easily and not-so-easily exploited vulnerabilities, with the expectation that the tester will share what they have found with you.

Hiring a penetration tester means *relying on* their skill at finding vulnerabilities but also *trusting* their ethical compass will lead them to tell you and to act as a trustworthy *fiduciary* (guardian of your interests). In assign5, you will have the opportunity to exercise your own ethical compass!

Example: Differential Privacy

Imagine a large database—perhaps a medical one—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 is a formal measure of privacy that attempts to address these concerns. By adding inconsequential noise (changing a birthday from 2001 to 2002, for example) or removing records, differential privacy protects individuals from *aggregation* by making them harder to identify (Dwork 2008).

Differential Privacy's Trust Model

Differential privacy assumes that the only threat to privacy is an *external user accessing 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.

Differential Privacy: The Other Threats

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

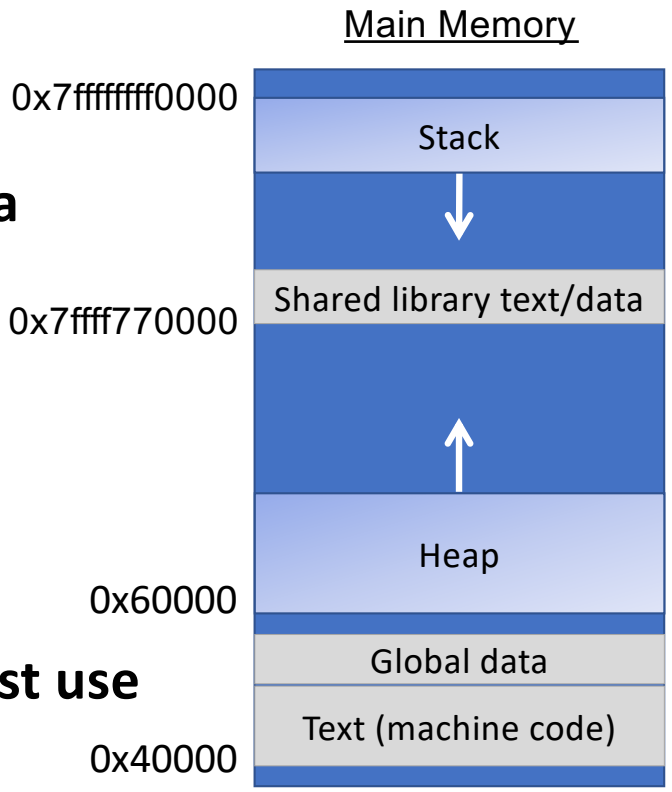
Differential privacy also does not question the assumption that amassing & storing large amounts of personal data is worth the risk of inevitable leaks (Rogaway 2015).

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?

Running a program

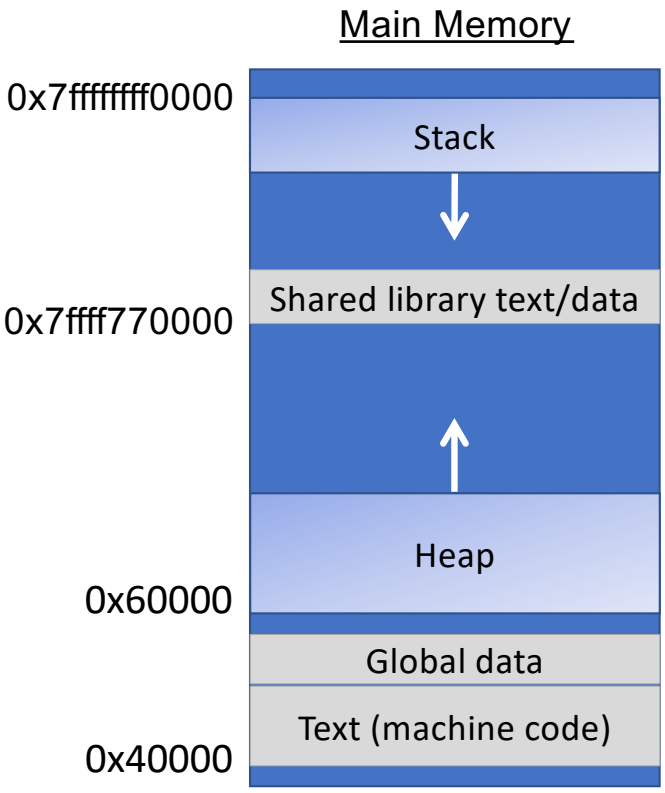
- **Creates new process**
- **Sets up address space/segments**
- **Read executable file, load instructions, global data**
  Mapped from file into gray segments
- **Libraries loaded on demand**

- **Set up stack**
  Reserve stack segment, initialize **%rsp**, **callq main**
- **malloc written in C, heap will initialize itself on first use**
  Asks OS for large memory region,
  parcels out to service requests



The Stack

Review

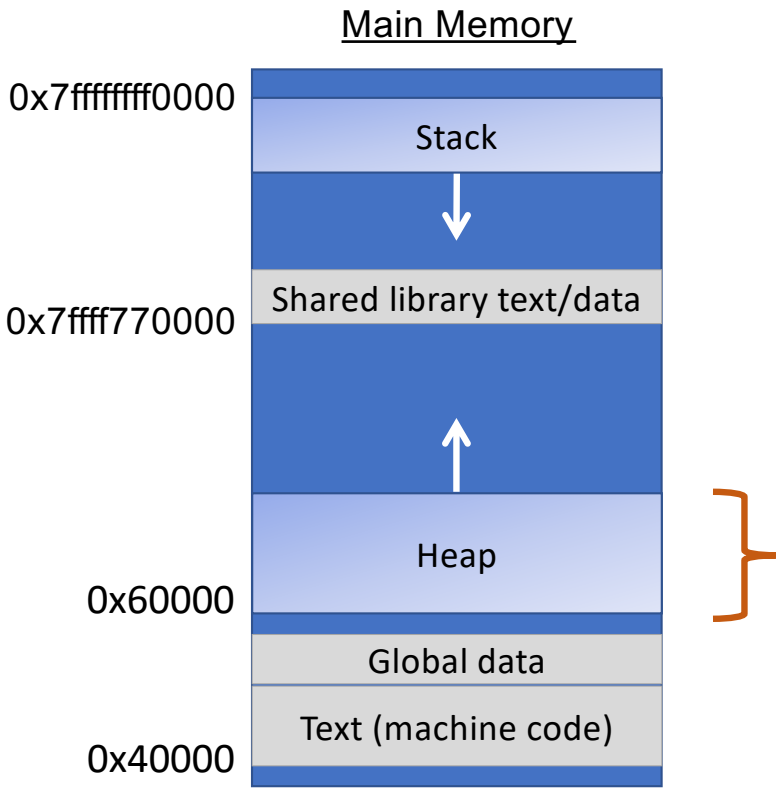


Stack memory "goes away" after function call ends.

Automatically managed at compile-time by gcc

From Assembly:
Stack management amounts to moving **%rsp** up and down (pushq, popq, mov)

Preamble: 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?

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);
```

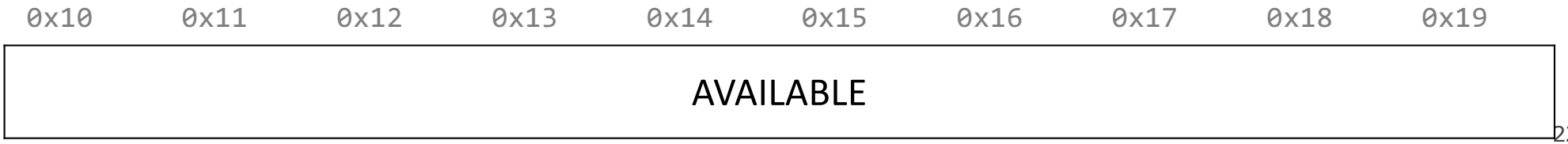
Frees the heap-allocated block starting at the specified address.

```
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 address of new, larger allocated memory region. `realloc(NULL, size)` -> `malloc(size)`

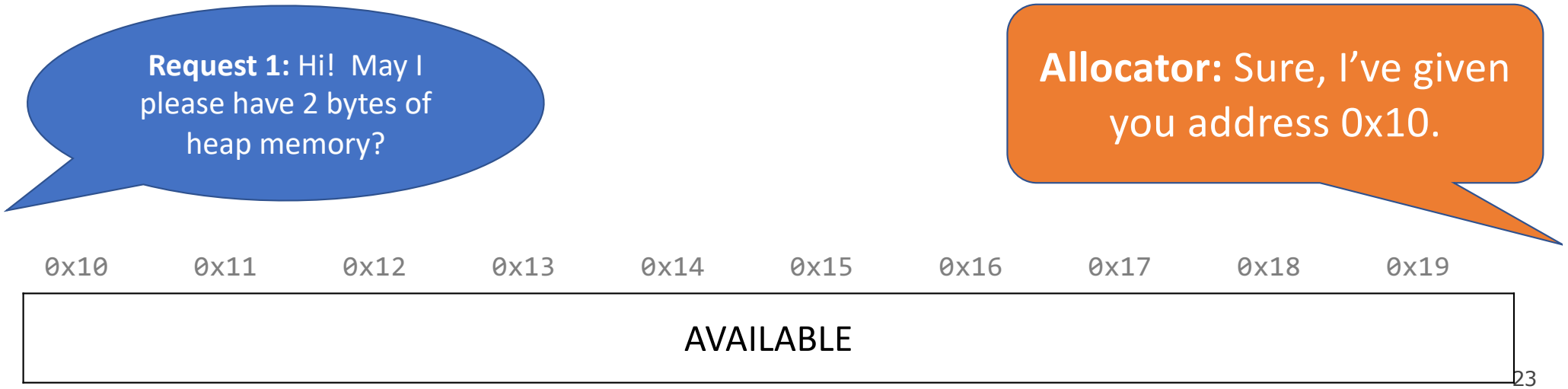
What is a heap allocator?

- A heap allocator is a suite of functions that cooperatively fulfill requests for dynamically allocated memory.
- When initialized, a heap allocator tracks the base address and the size of a large contiguous block of memory. That block of memory is the heap.



What is a heap allocator?

- A heap allocator is a suite of functions that cooperatively fulfill requests for dynamically allocated memory.
- When initialized, a heap allocator tracks the base address and the size of a large contiguous block of memory. That block of memory is the heap.
- The allocator manages the heap as clients request or donate back pieces of it.

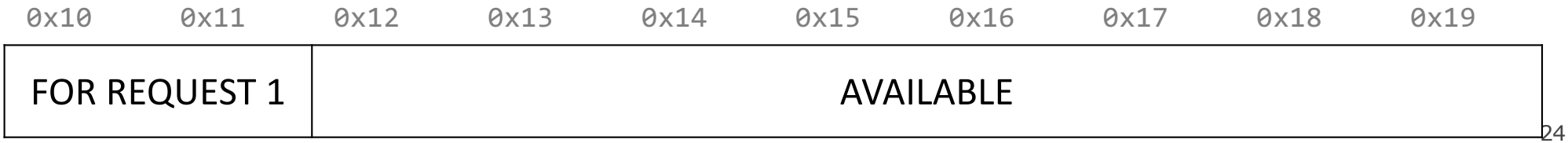


What is a heap allocator?

- A heap allocator is a suite of functions that cooperatively fulfill requests for dynamically allocated memory.
- When initialized, a heap allocator tracks the base address and the size of a large contiguous block of memory. That block of memory is the heap.
- The allocator manages the heap as clients request or donate back pieces of it.

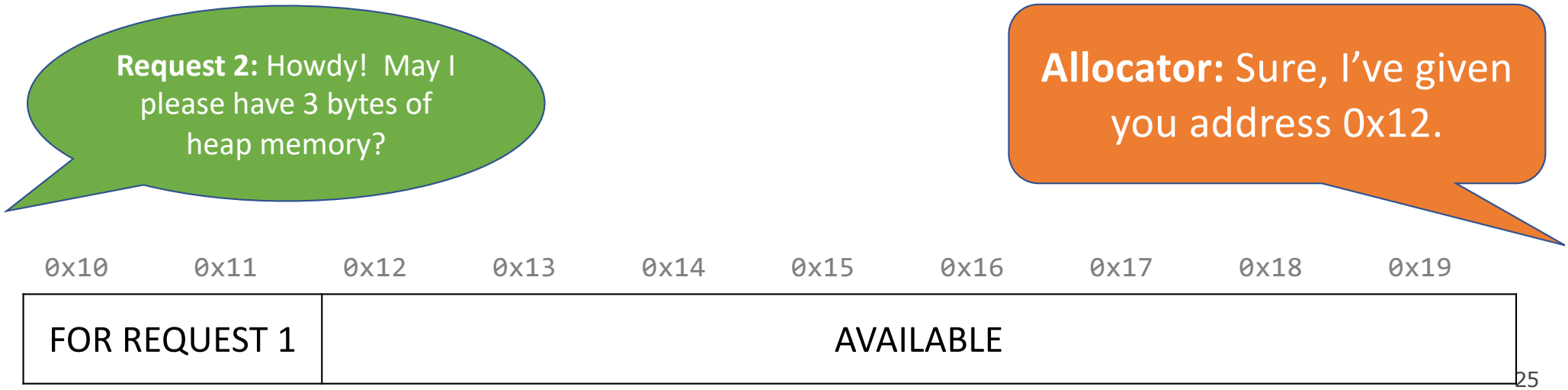
Request 1: Hi! May I please have 2 bytes of heap memory?

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



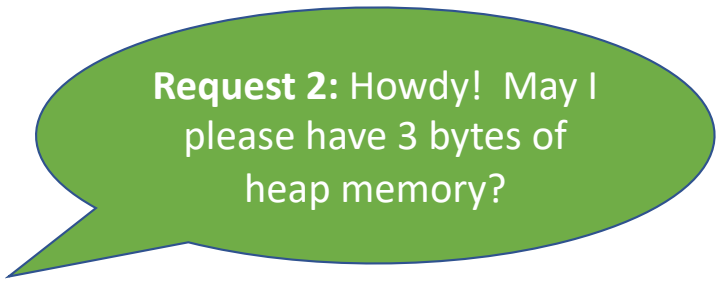
What is a heap allocator?

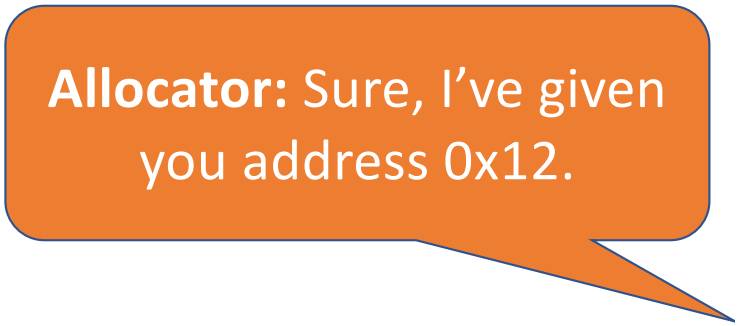
- A heap allocator is a suite of functions that cooperatively fulfill requests for dynamically allocated memory.
- When initialized, a heap allocator tracks the base address and the size of a large contiguous block of memory. That block of memory is the heap.
- The allocator manages the heap as clients request or donate back pieces of it.

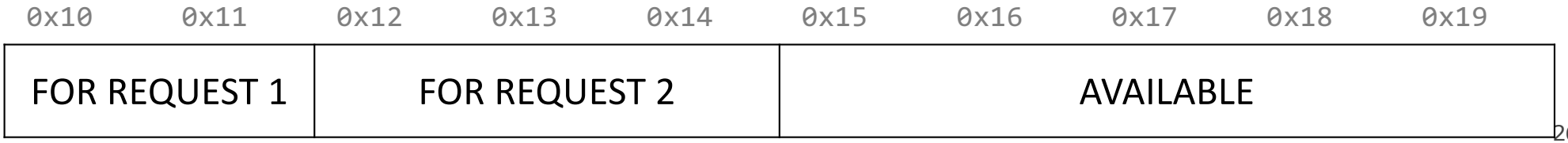


What is a heap allocator?

- A heap allocator is a suite of functions that cooperatively fulfill requests for dynamically allocated memory.
- When initialized, a heap allocator tracks the base address and the size of a large contiguous block of memory. That block of memory is the heap.
- The allocator manages the heap as clients request or donate back pieces of it.





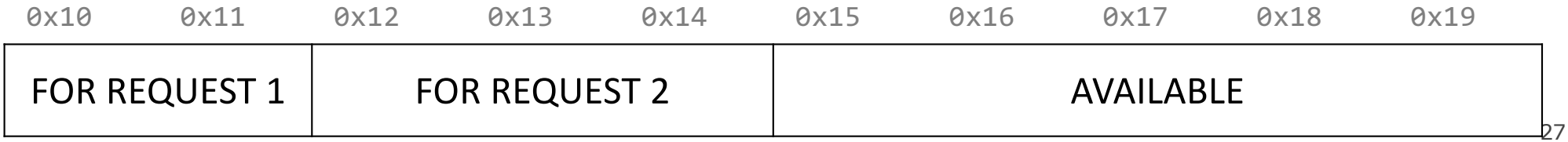


What is a heap allocator?

- A heap allocator is a suite of functions that cooperatively fulfill requests for dynamically allocated memory.
- When initialized, a heap allocator tracks the base address and the size of a large contiguous block of memory. That block of memory is the heap.
- The allocator manages the heap as clients request or donate back pieces of it.

Request 1: I'm done with the memory I requested. Thank you!

Allocator: Thanks. Have a good day!

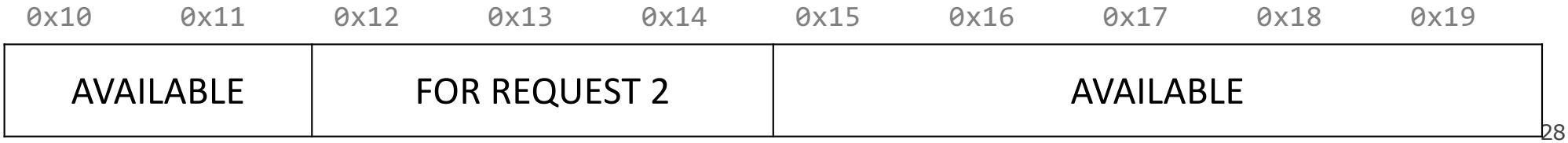


What is a heap allocator?

- A heap allocator is a suite of functions that cooperatively fulfill requests for dynamically allocated memory.
- When initialized, a heap allocator tracks the base address and the size of a large contiguous block of memory. That block of memory is the heap.
- The allocator manages the heap as clients request or donate back pieces of it.

Request 1: I'm done with the memory I requested. Thank you!

Allocator: Thanks. Have a good day!

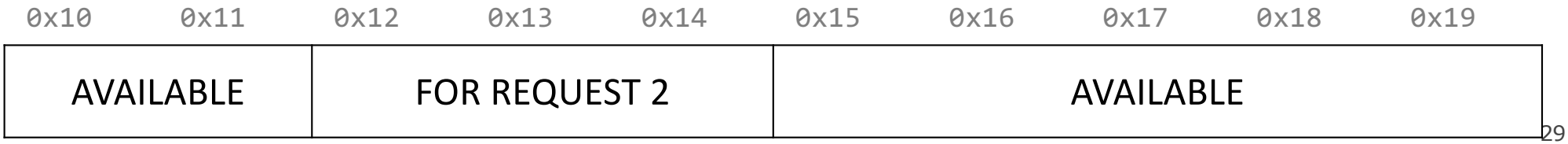


What is a heap allocator?

- A heap allocator is a suite of functions that cooperatively fulfill requests for dynamically allocated memory.
- When initialized, a heap allocator tracks the base address and the size of a large contiguous block of memory. That block of memory is the heap.
- The allocator manages the heap as clients request or donate back 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.

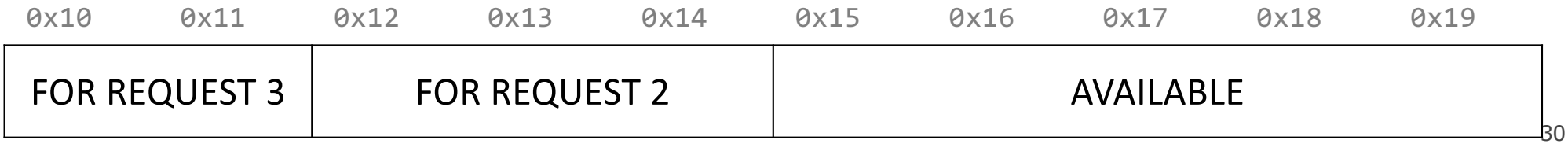


What is a heap allocator?

- A heap allocator is a suite of functions that cooperatively fulfill requests for dynamically allocated memory.
- When initialized, a heap allocator tracks the base address and the size of a large contiguous block of memory. That block of memory is the heap.
- The allocator manages the heap as clients request or donate back 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.

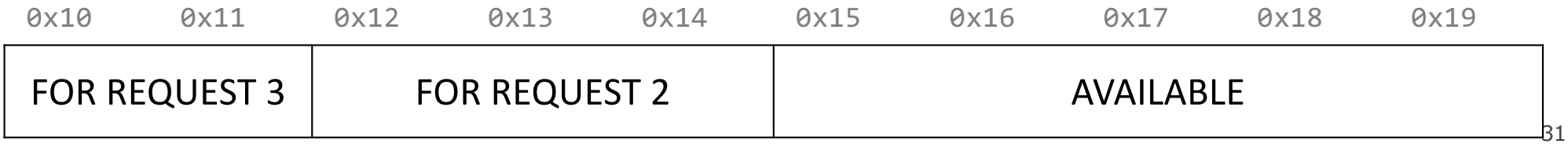


What is a heap allocator?

- A heap allocator is a suite of functions that cooperatively fulfill requests for dynamically allocated memory.
- When initialized, a heap allocator tracks the base address and the size of a large contiguous block of memory. That block of memory is the heap.
- The allocator manages the heap as clients request or donate back 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.



What is a heap allocator?

- A heap allocator is a suite of functions that cooperatively fulfill requests for dynamically allocated memory.
- When initialized, a heap allocator tracks the base address and the size of a large contiguous block of memory. That block of memory is the heap.
- The allocator manages the heap as clients request or donate back 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.