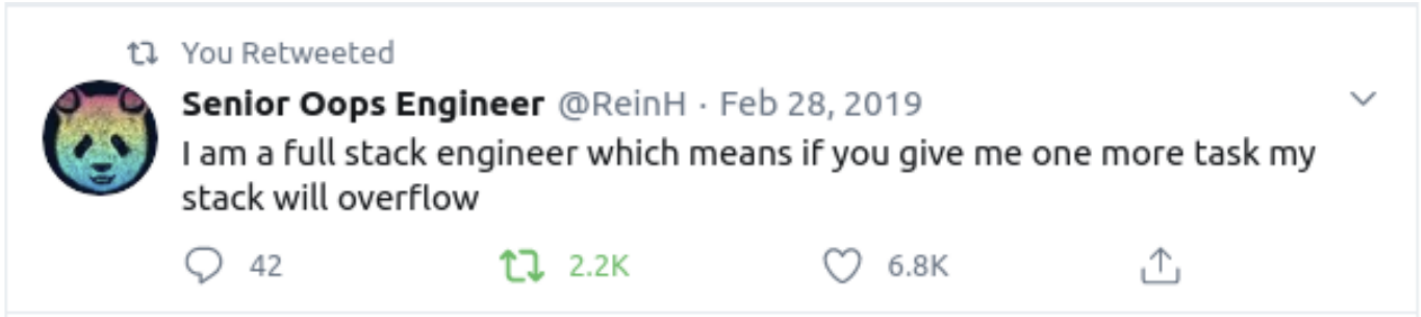
Procedures I

CSE 351 Autumn 2023

Instructor:
Justin Hsia

Teaching Assistants:

Afifah Kashif	Malak Zaki
Bhavik Soni	Naama Amiel
Cassandra Lam	Nayha Auradkar
Connie Chen	Nikolas McNamee
David Dai	Pedro Amarante
Dawit Hailu	Renee Ruan
Ellis Haker	Simran Bagaria
Eyoel Gebre	Will Robertson
Joshua Tan	



Relevant Course Information

- Lab 2 due next Friday (10/27)
  - Can start in earnest after today's lecture!
  - See GDB Tutorial Lesson and and Phase 1 walkthrough in Section 4 Lesson

- Midterm (take home, 11/2–11/4)
  - Make notes and use the midterm reference sheet
  - Form study groups and look at past exams!

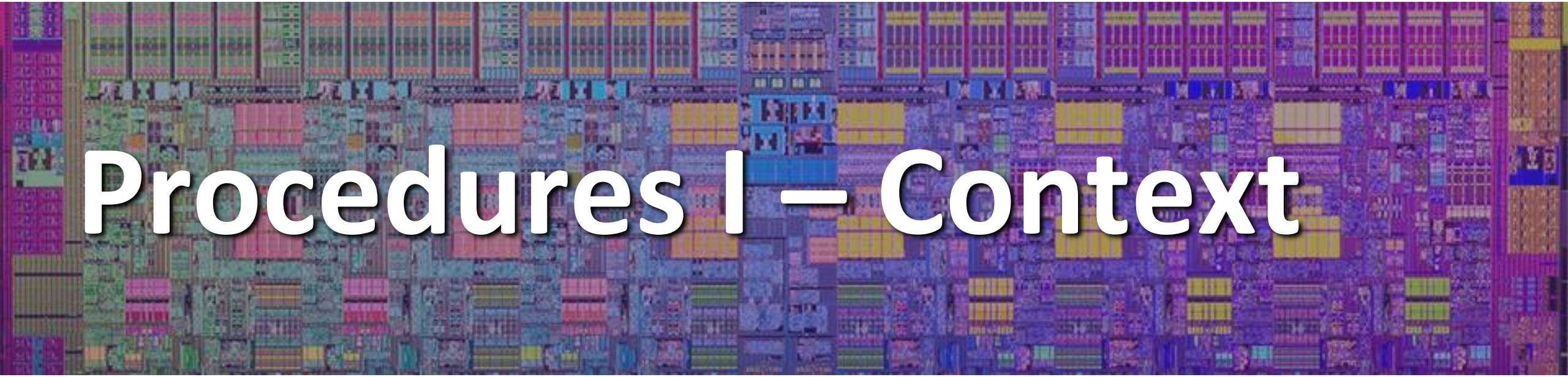
Procedures I

Lesson Summary (1/2)

- Memory is organized into 5 segments (Stack, Heap, Static Data, Literals, Instructions/Code) based on data declaration and lifetime
  - Goals: maximize use of space, manage data differently, apply separate permissions
  - The Stack is at the highest addresses and grows downward; can manipulate using `add`, `sub`, `push`, and `pop`
- Procedure calling conventions for passing control and data
  - `call` and `ret` pass control using `%rip` and a return address on the stack
  - Return value: `%rax`, Arguments: `%rdi`, `%rsi`, `%rdx`, `%rcx`, `%r8`, `%r9`, Stack
- Stack organized into stack frames that hold a procedure instance's data

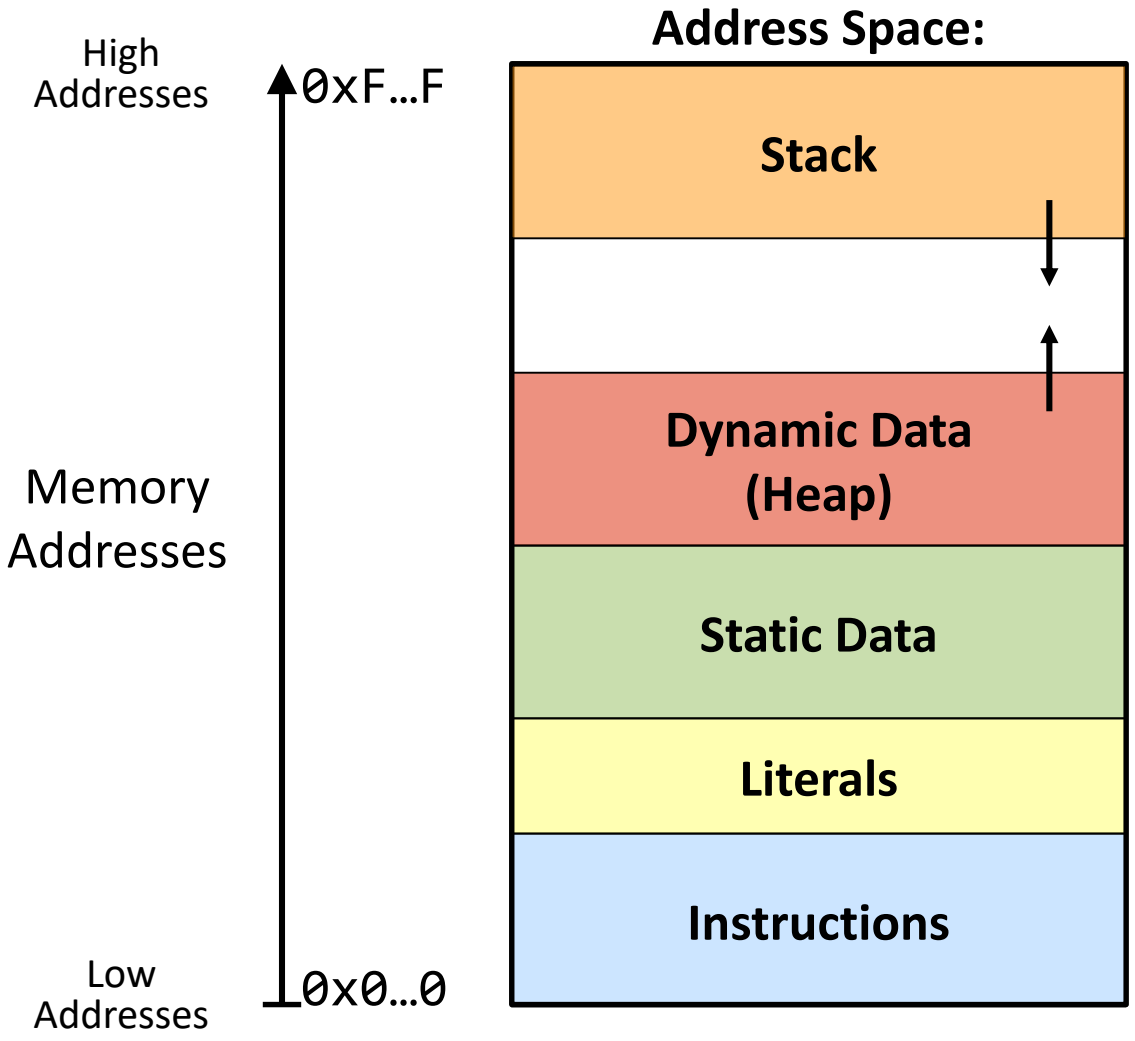
Lesson Summary (2/2)

- Terminology:
  - Stack, Heap, Static Data, Literals, Instructions/Code
  - Stack pointer (`%rsp`), `push`, `pop`
  - Caller, callee, return address, `call`, `ret`
  - Stack frames and stack discipline
- Learning Objectives:
  - Determine the location/segment in memory that a piece of data will be stored based on the nature of that data (*i.e.*, static, literals, etc.).
  - Trace stack frame movement and creation.
- What lingering questions do you have from the lesson?



Procedures I – Context

Simplified Memory Layout



What Goes Here:

Local variables and procedure context

Variables allocated with new or `malloc`

Static variables (including global variables)

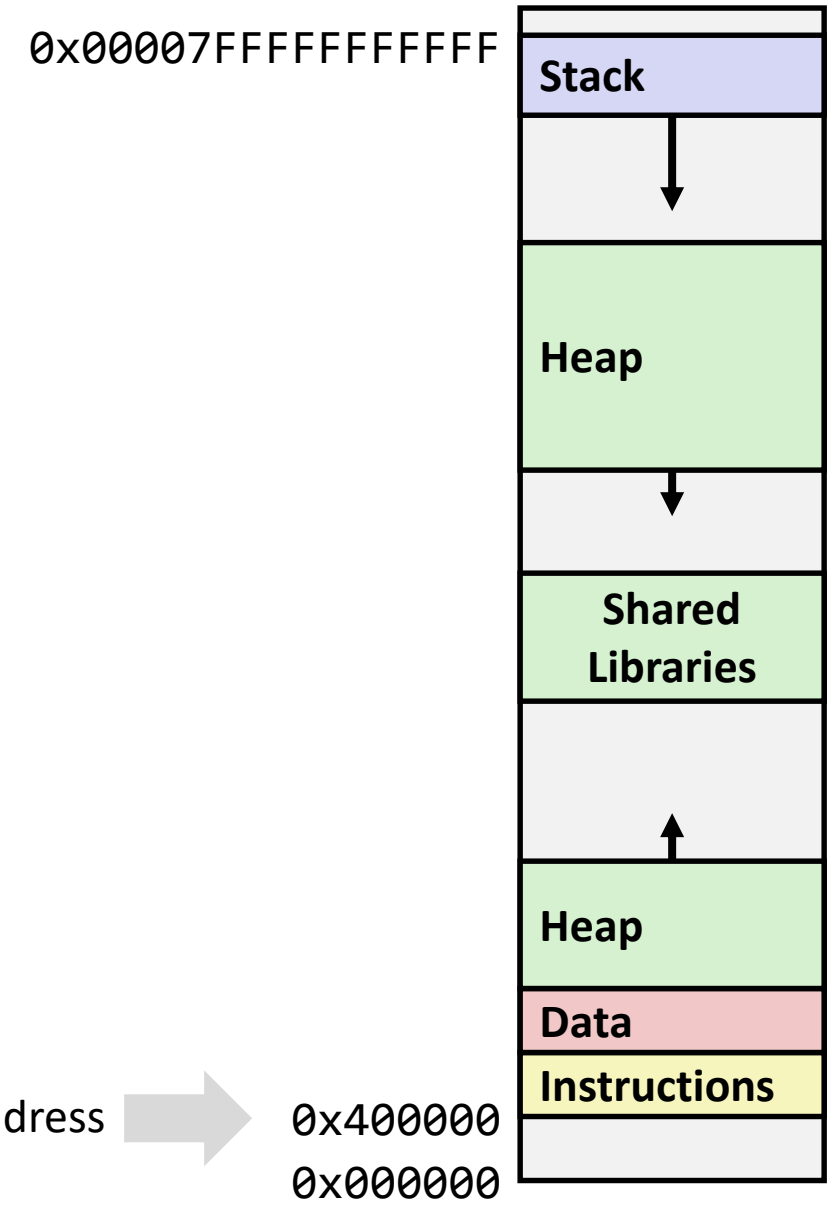
Immutable literals/constants (*e.g.*, `"example"`)

Program code

x86-64 Linux Memory Layout

This is extra (non-testable) material

- Stack
  - Runtime stack has 8 MiB limit
- Heap
  - Dynamically allocated as needed
  - `malloc()`, `calloc()`, `new`, …
- Statically allocated data (Data)
  - Read-only: string literals
  - Read/write: global arrays and variables
- Code / Shared Libraries
  - Executable machine instructions
  - Read-only



Stack Overflow

- When the stack pointer exceeds the stack bounds (segmentation fault)
  - In theory: when it collides with the Heap
  - In x86-64 Linux, when it exceeds 8 MiB limit

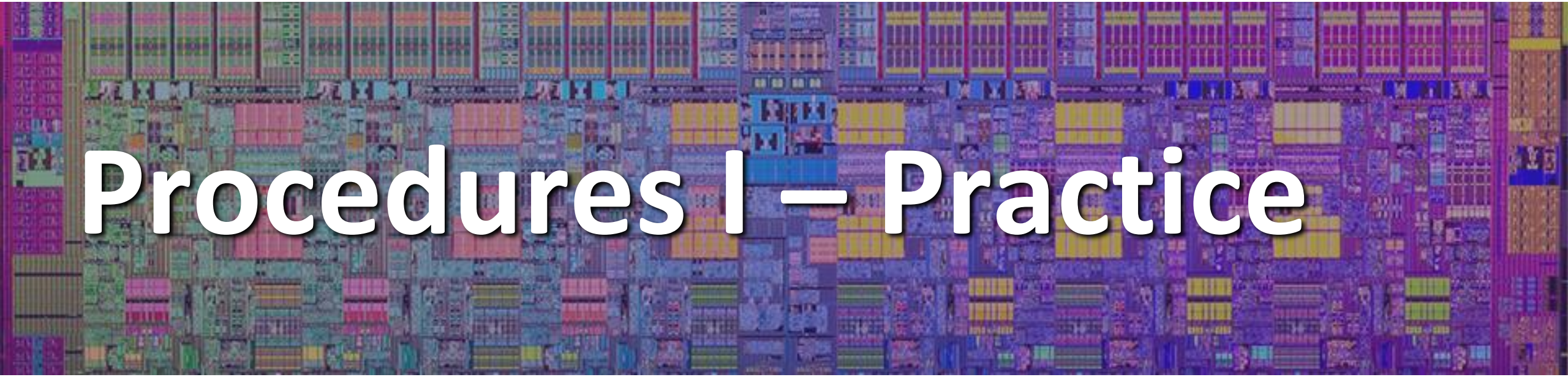
Aside: Stack Overflow

- Has nothing to do with actual stack overflow – named based on poll of blog users; some of the non-winning options:
  - algorithmical
  - bitoriented
  - dereferenced
  - fellowhackers
  - humbleprogrammers
  - privatevoid
  - shiftleft1
  - understandrecursion
- Crowd-sourced their logo for $512

Discussion Questions

- Discuss the following question(s) in groups of 3-4 students
  - I will call on a few groups afterwards so please be prepared to share out
  - Be respectful of others' opinions and experiences

- Naming/etymology plays a big role in learning
  - Which new terms in this class have been the most intuitive for you to learn vs. the most difficult?
  - What do you think goes into a good vs. bad name more generally in computer science?



Procedures I – Practice

Group Work Time

- During this time, you are encouraged to work on the following:
  1) If desired, continue your discussion
  2) Work on the lesson problems (solutions at the end of class)
  3) Work on the homework problems

- Resources:
  - You can revisit the lesson material
  - Work together in groups and help each other out
  - Course staff will circle around to provide support

Practice Questions (1/2)

- How does the stack change after executing the following instructions?

```
pushq %rbp
subq  $0x18, %rsp
```

- For the following function, which registers do we know *must* be used?

```
void* memset(void* ptr, int value, size_t num);
```

Practice Questions (2/2)

- Answer the following questions about when `main()` is run (assume `x` and `y` stored on the Stack):

```
int main() {
    int i, x = 0;
    for(i=0;i<3;i++)
        x = randSum(x);
    printf("x = %d\n",x);
    return 0;
}
```

```
int randSum(int n) {
    int y = rand()%20;
    return n+y;
}
```

  - *Higher/larger address*: `x` or `y`?
  - How many total stack frames are *created*?
  - What is the maximum *depth* (# of frames) of the Stack?

A. 1 B. 2 C. 3 D. 4