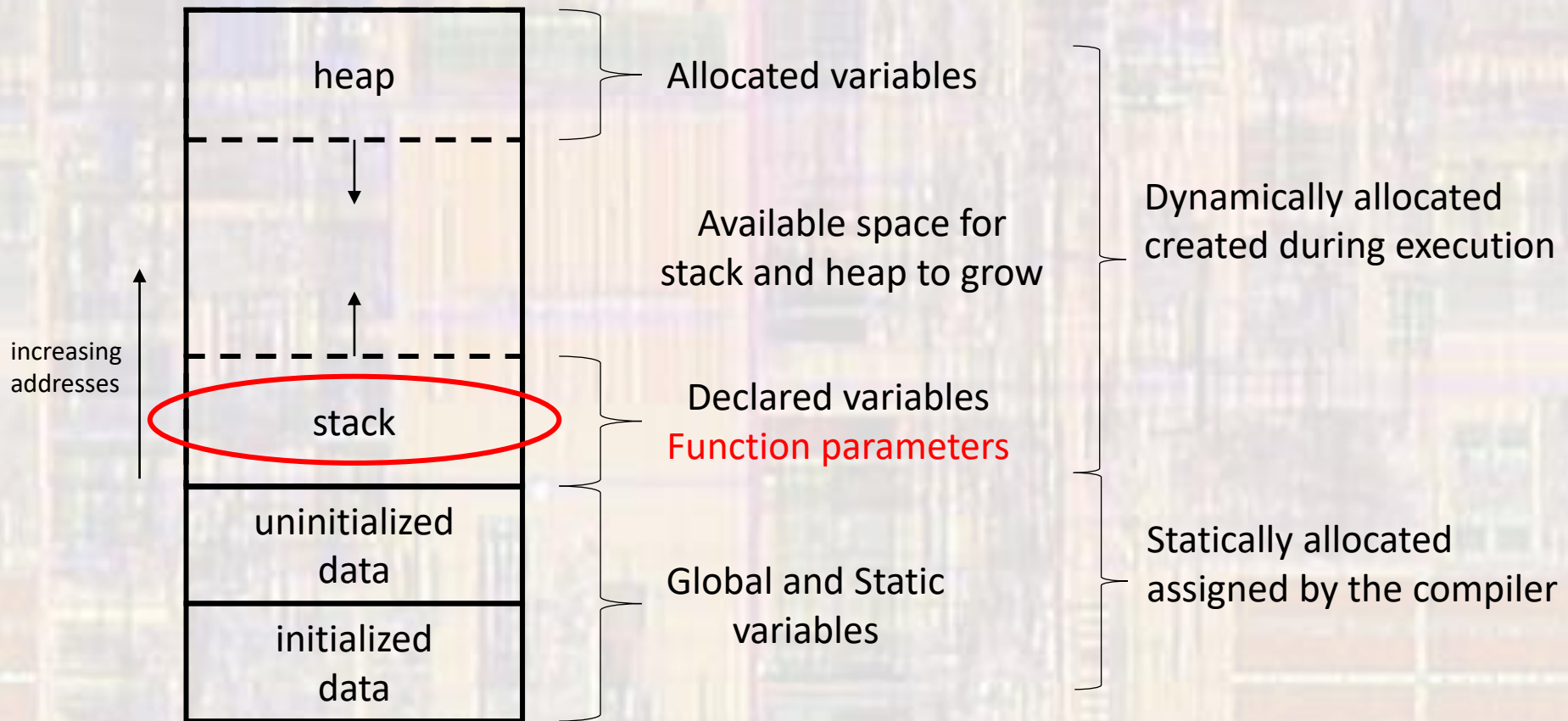# Functions and Data Memory

Last updated 5/13/24

These slides introduce C functions in Data Memory

# Functions and Data Memory

- ## Data Memory Storage
  - ### Typical structure – but variations exist

```
                          ┌─  Allocated variables      ┐
┌──────────────┐          │                            │
│     heap     │          │                            │
│- - - - - - - │          ┘                            │
│      ↓       │                                       │
│              │            Available space for          Dynamically allocated
│              │            stack and heap to grow       created during execution
│      ↑       │                                       │
│- - - - - - - │          ┌─                           │
│  ⟨ stack ⟩   │          │  Declared variables        ┘
│              │          │  Function parameters
├──────────────┤          ┘
│ uninitialized│          ┌─                           ┐
│     data     │          │                            │
├──────────────┤            Global and Static            Statically allocated
│  initialized │          │  variables                   assigned by the compiler
│     data     │          │                            │
└──────────────┘          ┘                            ┘
```

increasing addresses ↑

# Functions and Data Memory

- Process when a function is called

1. A stack frame is created to store the function elements on the stack
   - A stack pointer is created to access the stack (similar to a program counter)
2. The Program Counter (memory location for the Next instruction to be executed) is stored on the stack
3. Memory locations are allocated on the stack for any formal parameters
4. The formal parameter memory spaces are filled with the actual parameter values passed to the function
5. Space is allocated on the stack for any variables that are local to the function
6. The function executes (see notes on Functions in Program Memory)
7. The return value is stored in a special register
8. The local and formal variable memory locations are abandoned, and the stack pointer is updated
9. The Program Counter is reloaded with memory location stored on the stack in step 1 – continuing the program flow

# Functions and Data Memory

- ## Function Example - stack

float average(float val1, float val2);

…

int main(void){
  float ave;
  float try1;
  float try2;

t0 →   // enter try1, try2 (9, 3)

t1 →   …
  ave = average(try1, try2);

t5 →   …
  return 0;
}

float average(float val1, float val2){
t2 →   float tmp;
t3 →   tmp = (val1 + val2)/2;
t4 →   return tmp;
}

### Data Memory - Stack

| | t0 | t1 | | t2 | t3 | t4 | t5 |
|---|---|---|---|---|---|---|---|
| | ? | ? | tmp | ? | 6 | 6 | 6 |
| | ? | ? | val2 | 3 | 3 | 3 | 3 |
| | ? | ? | val1 | 9 | 9 | 9 | 9 |
| | ? | ? | return addr | 0x1000 | 0x1000 | 0x1000 | 0x1000 |
| try2 | ? | 3 | | 3 | 3 | 3 | 3 |
| try1 | ? | 9 | | 9 | 9 | 9 | 9 |
| ave | ? | ? | | ? | ? | ? | 6 |

| result reg | ? | ? | 6 | 6 |
|---|---|---|---|---|

Many variations to this process exist