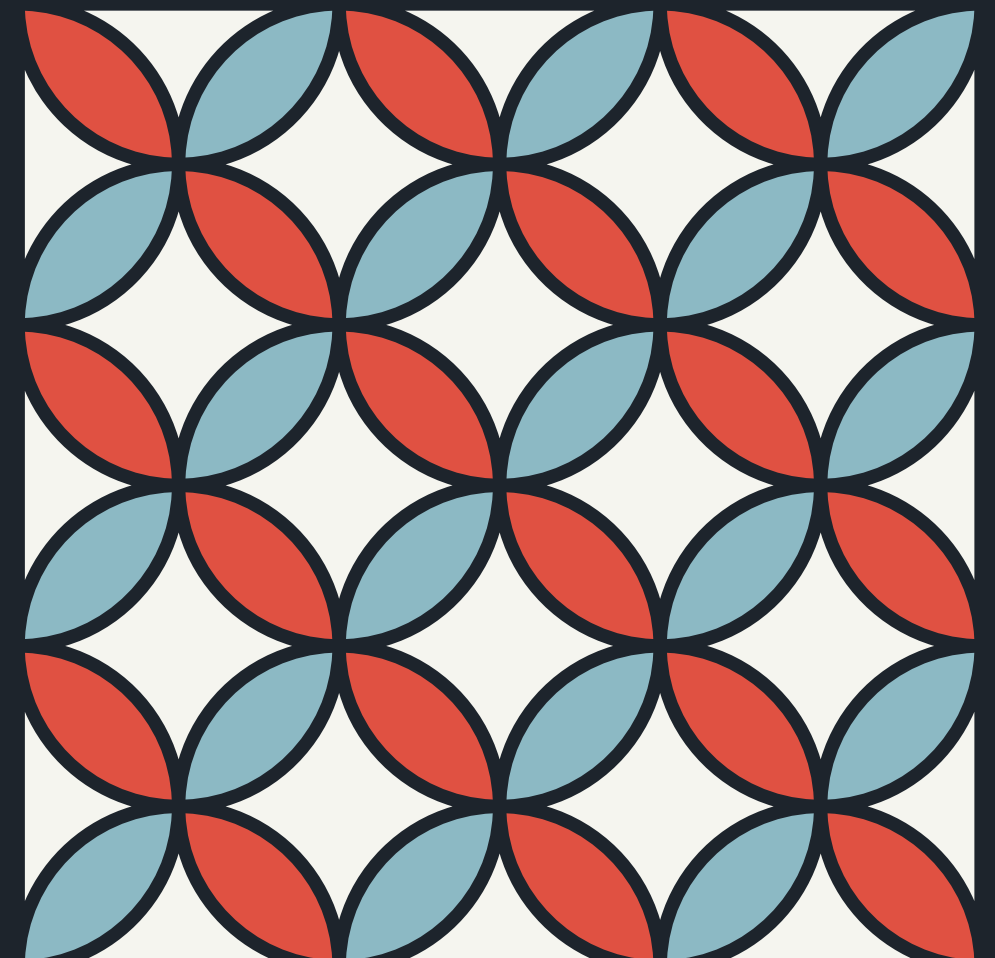
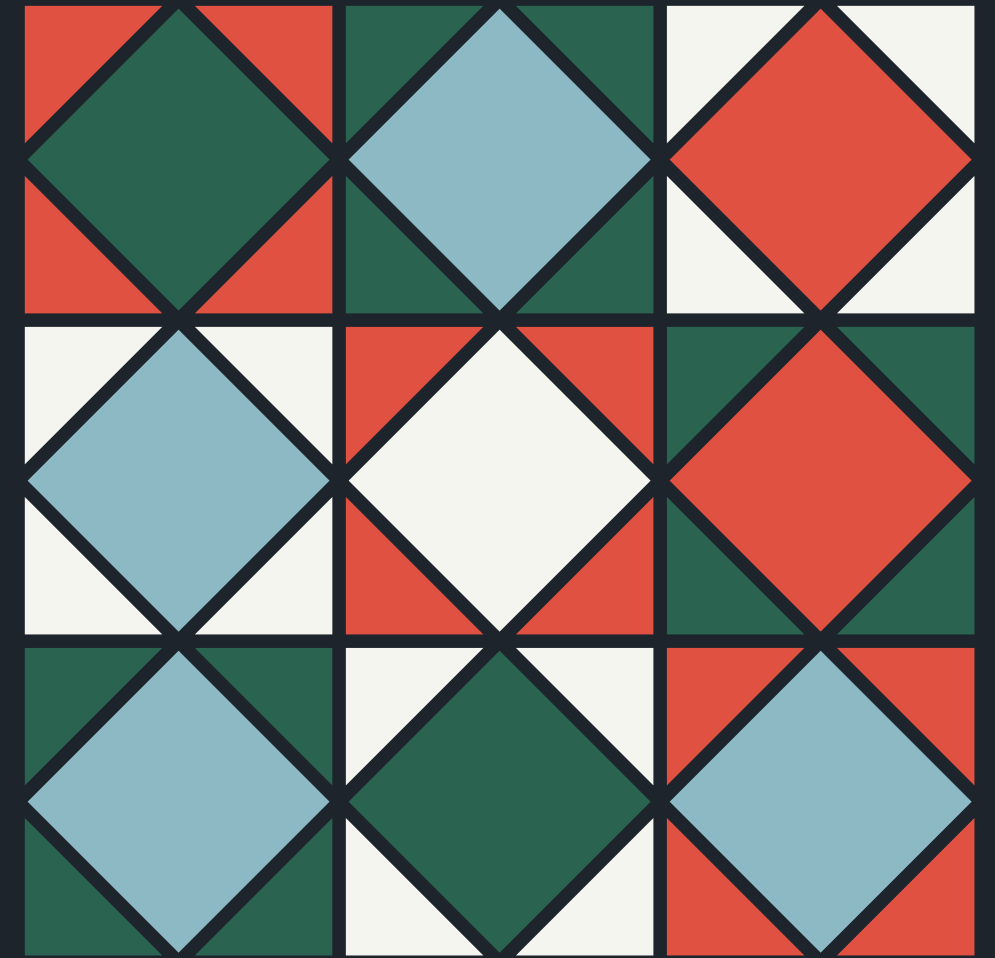
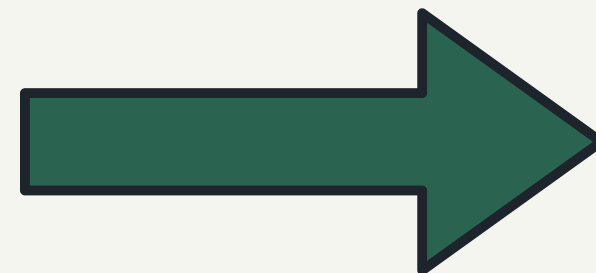
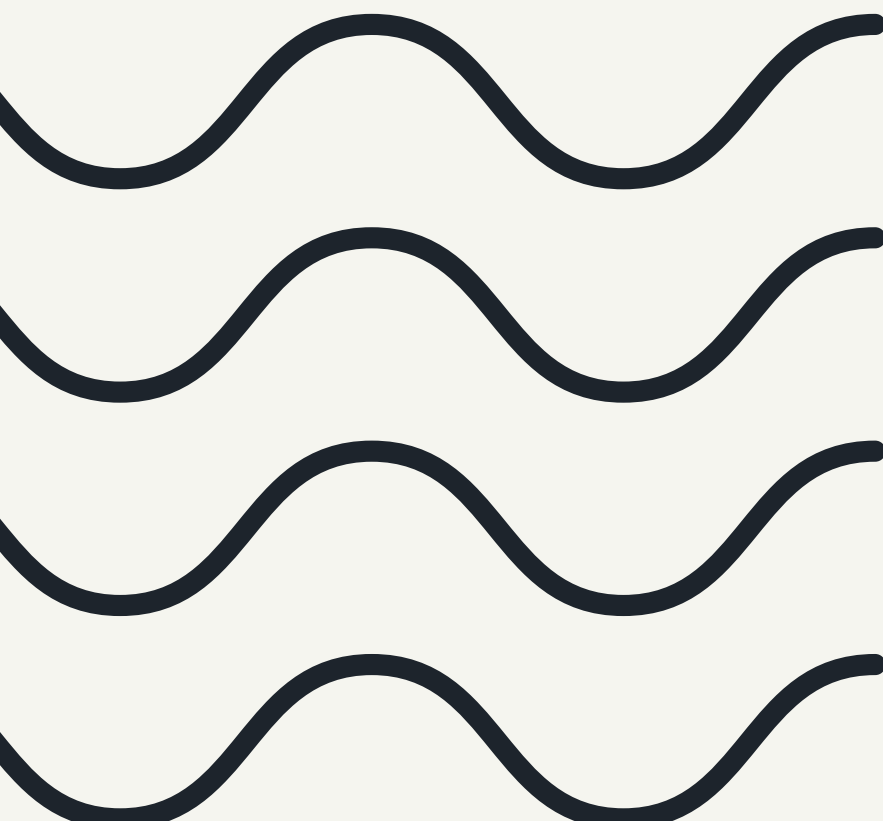


OSN

TUTORIAL-2

C-SHELL

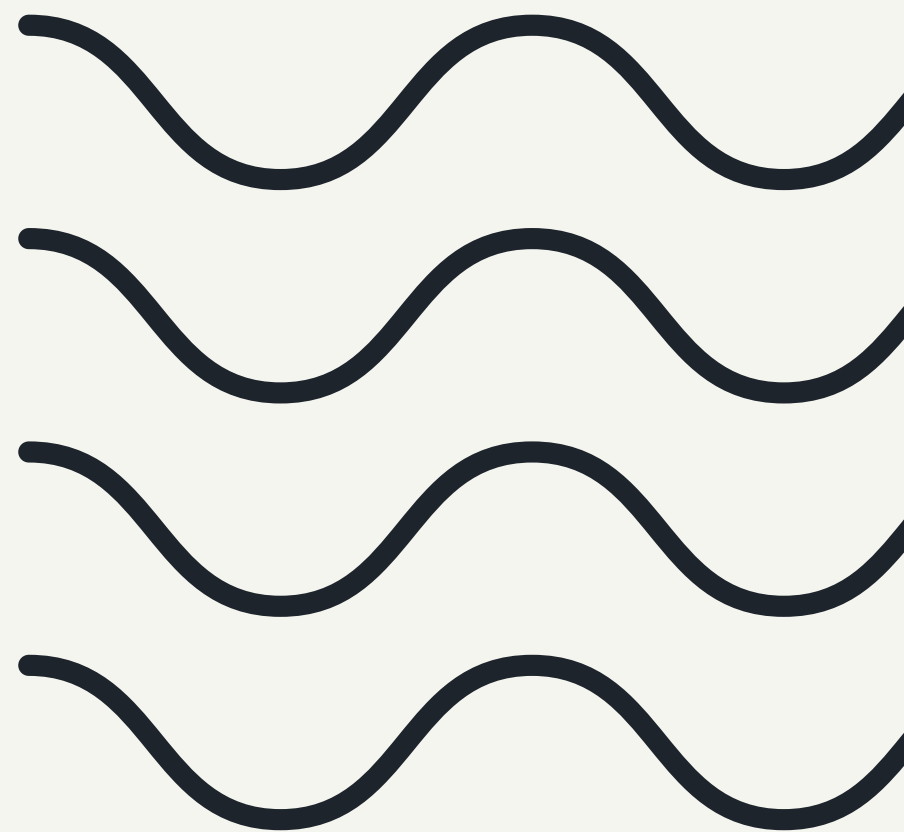




• • •

TODAY'S AGENDA

- PROCESSES
- FOREGROUND AND BACKGROUND PROCESSES
- EXEC COMMANDS
- FILE DESCRIPTORS
- PIPING
- RAW AND COOKED MODE



PROCESSES

01

As its been covered previous time, processes have their own unique ID with which they can be identifies.

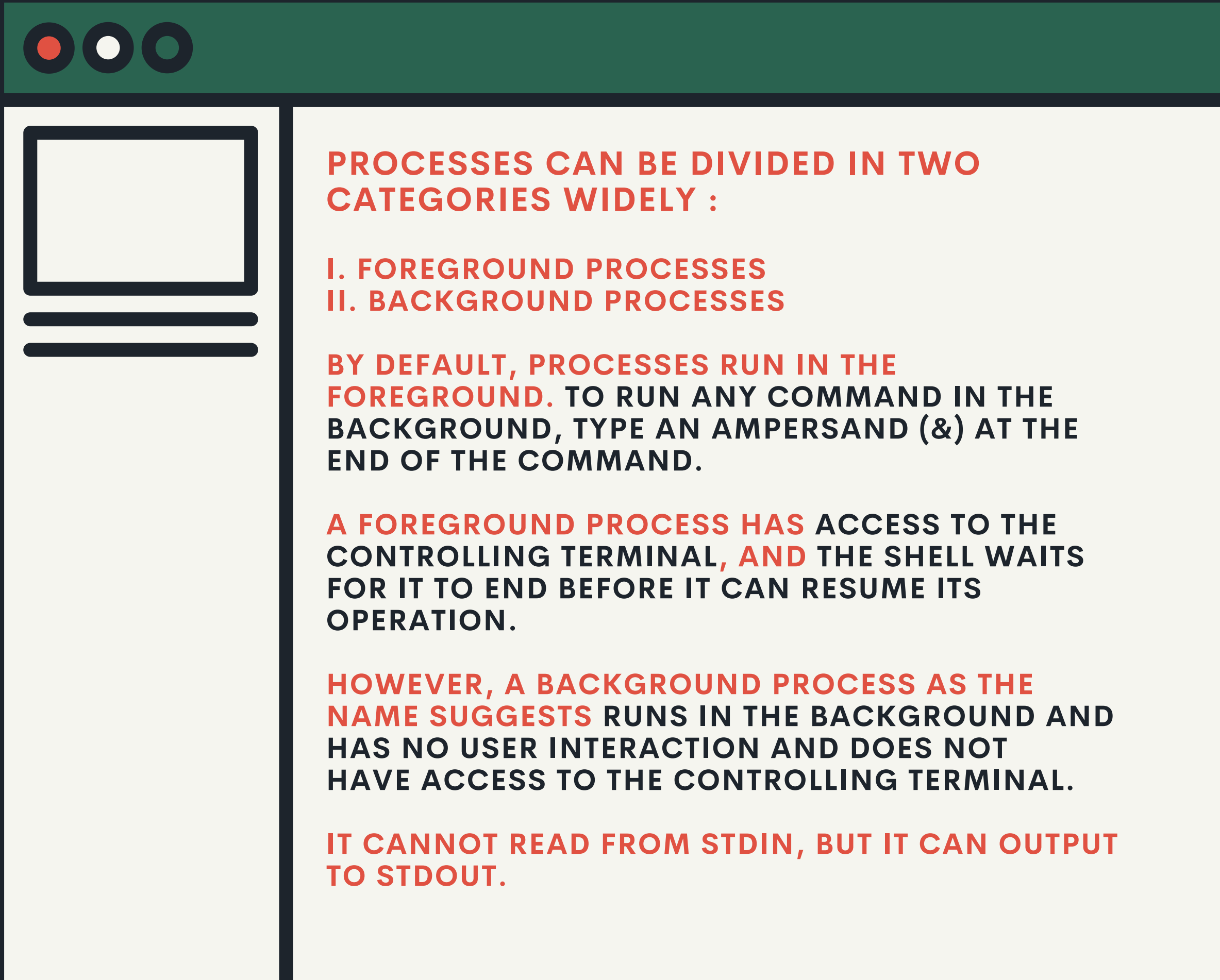
Try typing "ps" on your terminal to find out the processes that our currently running and their pid.

02

But the question is : Are all these processes running in a similar fashion?

They are not.

FOR EGGROUND & BACKGROUNDD PROCESSES



PROCESSES CAN BE DIVIDED IN TWO CATEGORIES WIDELY :

- I. FOREGROUND PROCESSES**
- II. BACKGROUND PROCESSES**

BY DEFAULT, PROCESSES RUN IN THE FOREGROUND. TO RUN ANY COMMAND IN THE BACKGROUND, TYPE AN AMPERSAND (&) AT THE END OF THE COMMAND.

A FOREGROUND PROCESS HAS ACCESS TO THE CONTROLLING TERMINAL, AND THE SHELL WAITS FOR IT TO END BEFORE IT CAN RESUME ITS OPERATION.

HOWEVER, A BACKGROUND PROCESS AS THE NAME SUGGESTS RUNS IN THE BACKGROUND AND HAS NO USER INTERACTION AND DOES NOT HAVE ACCESS TO THE CONTROLLING TERMINAL.

IT CANNOT READ FROM STDIN, BUT IT CAN OUTPUT TO STDOUT.



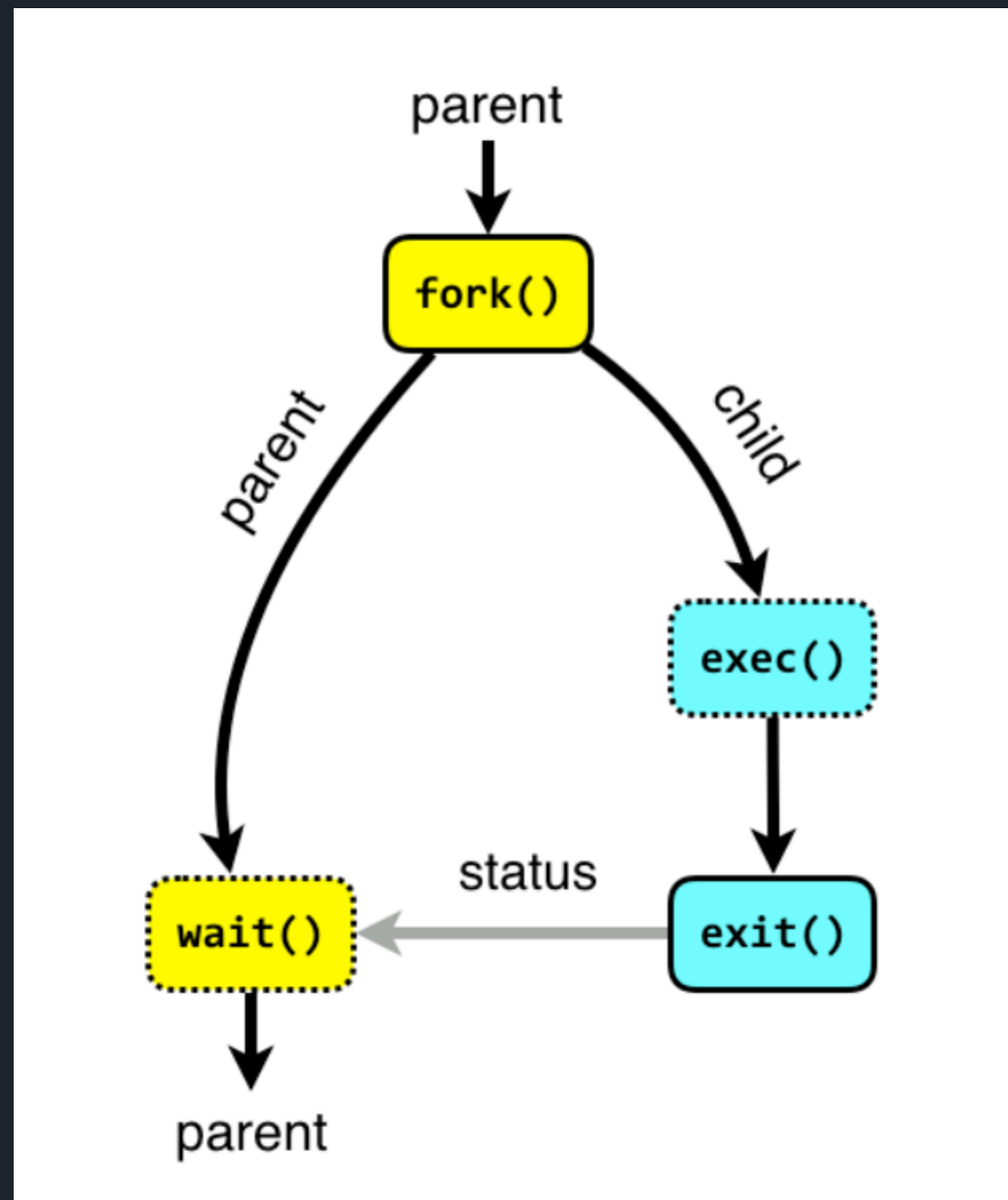
WAIT...

HAVING THE PARENT WAIT BEFORE THIS CHILD COMPLETES CAN BE BENEFICIAL IN SOME CASES.

WAIT() IS THE SYSTEM CALL THAT YOU ARE LOOKING FOR HERE. THIS IS A BLOCKING COMMAND, AND IT MAKES THE PARENT WAIT UNTIL ONE OF ITS CHILD TERMINATES.

WAITPID(PID_T PID) IS ANOTHER SYSTEM CALL THAT CAN BE USED FOR THIS PURPOSE WHICH WAITS FOR THE CHILD WITH PROCESS ID PID TO TERMINATE.

★ REAL POTENTIAL OF FORK EXEC()



exec replaces the current process image with a new process image that is specified in its arguments.

How is it useful?

Fork a new process from within a process. This process is duplicate of what you were running but then run **exec** and BOOM! You have a totally new process with a different functionality at hand.

- **execl/execv**: When you know the exact path to the executable and want to provide arguments either as a list (execl) or an array (execv).
- **execlp/execvp**: When you want the system to search for the executable in the PATH environment variable.
- **execle/execve**: When you need to provide a custom environment for the new process.





FILE DESCRIPTORS

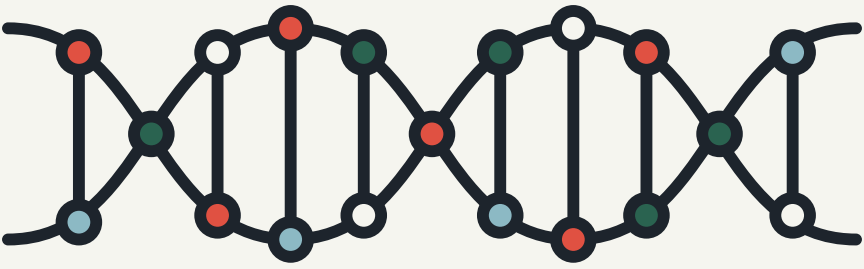
FILE DESCRIPTORS

- A file descriptor is a numeric identifier for an open file, with 0 for stdin, 1 for stdout, and 2 for stderr.
- The kernel maintains a table of open file descriptors for each process, which maps these numbers to structures (struct fd in Linux) containing file details.
- This structure includes a pointer to an open file description.

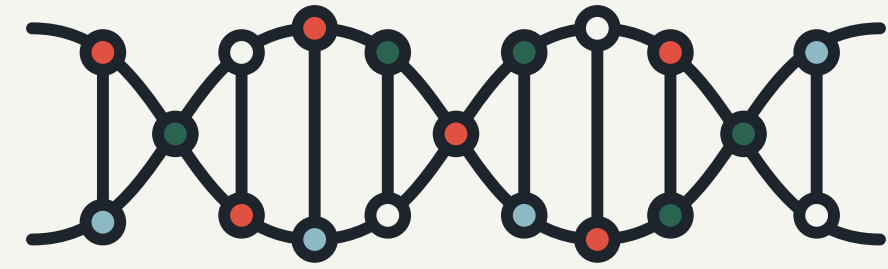
FILE DESCRIPTORS WITH FORK()

- After a fork() call, the child process inherits file descriptors from the parent, but both processes share the same open file description and file buffer.
- Changes to the file (reading or writing) by one process affect the shared file description and buffer, impacting the other process as well.





DUP AND PIPE



DUP AND DUP 2



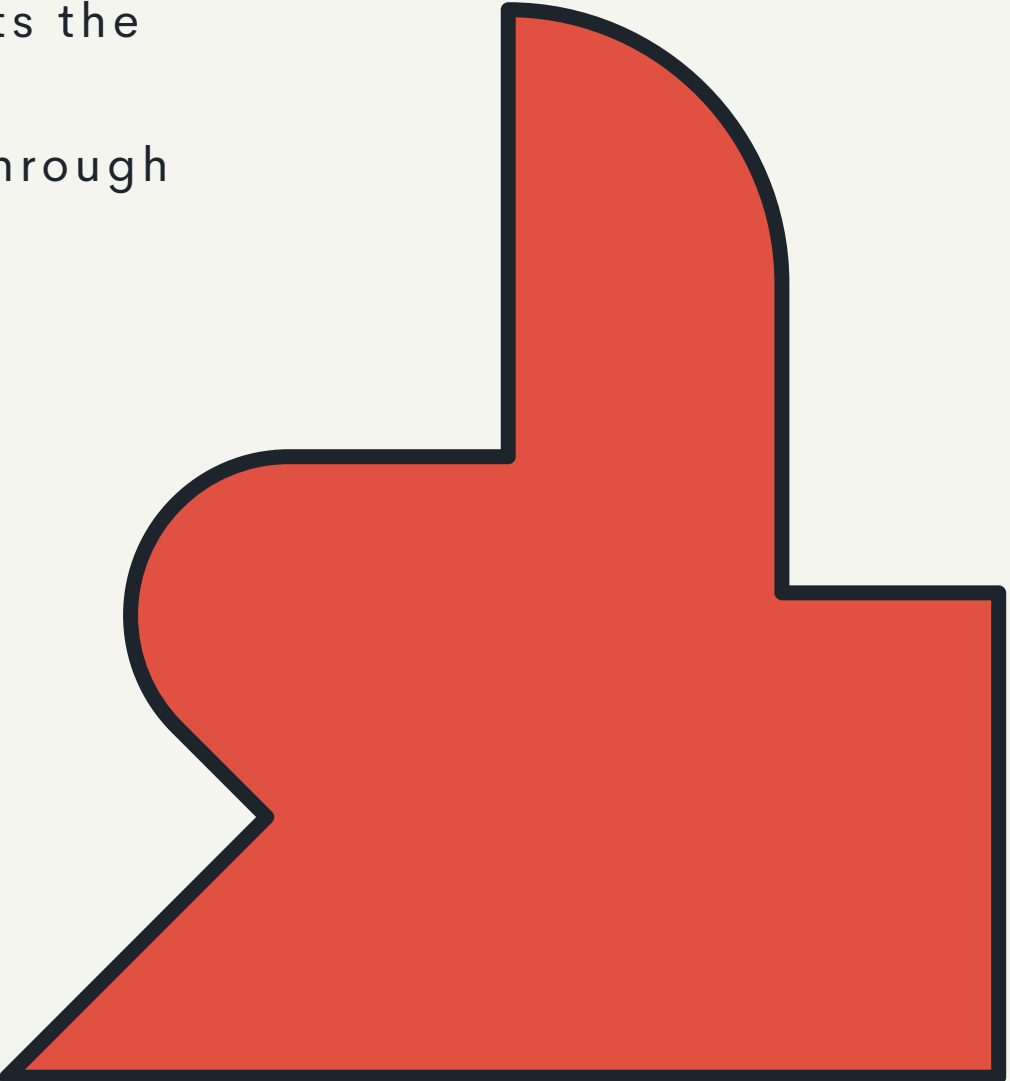
- dup duplicates an existing file descriptor to the lowest available file descriptor.
- dup2 duplicates an existing file descriptor to a specified file descriptor, closing the target if it is already open.
- Both functions allow redirection of standard input/output by duplicating descriptors like stdin, stdout, or stderr to/from files or pipes.

PIPING

- A pipe is a unidirectional communication channel with a read end and a write end.
- It is created using the pipe() system call, which provides two file descriptors: one for reading and one for writing.
- Piping enables data to flow between processes, where one process writes to the pipe, and another reads from it.



USING DUP IN PIPING

- In the writer process: `dup2(pipefd[1], STDOUT_FILENO)` redirects the standard output to the pipe's write end.
 - In the reader process: `dup2(pipefd[0], STDIN_FILENO)` redirects the standard input to the pipe's read end.
 - This setup allows one process to send data directly to another through the pipe, using `dup/dup2` to handle the redirection.
- 
- 
- 

TERMINAL

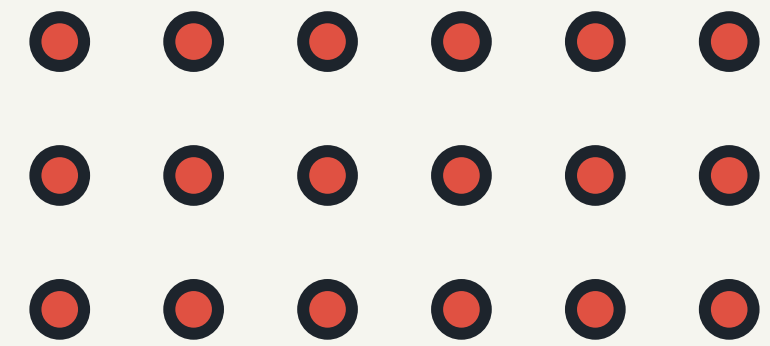
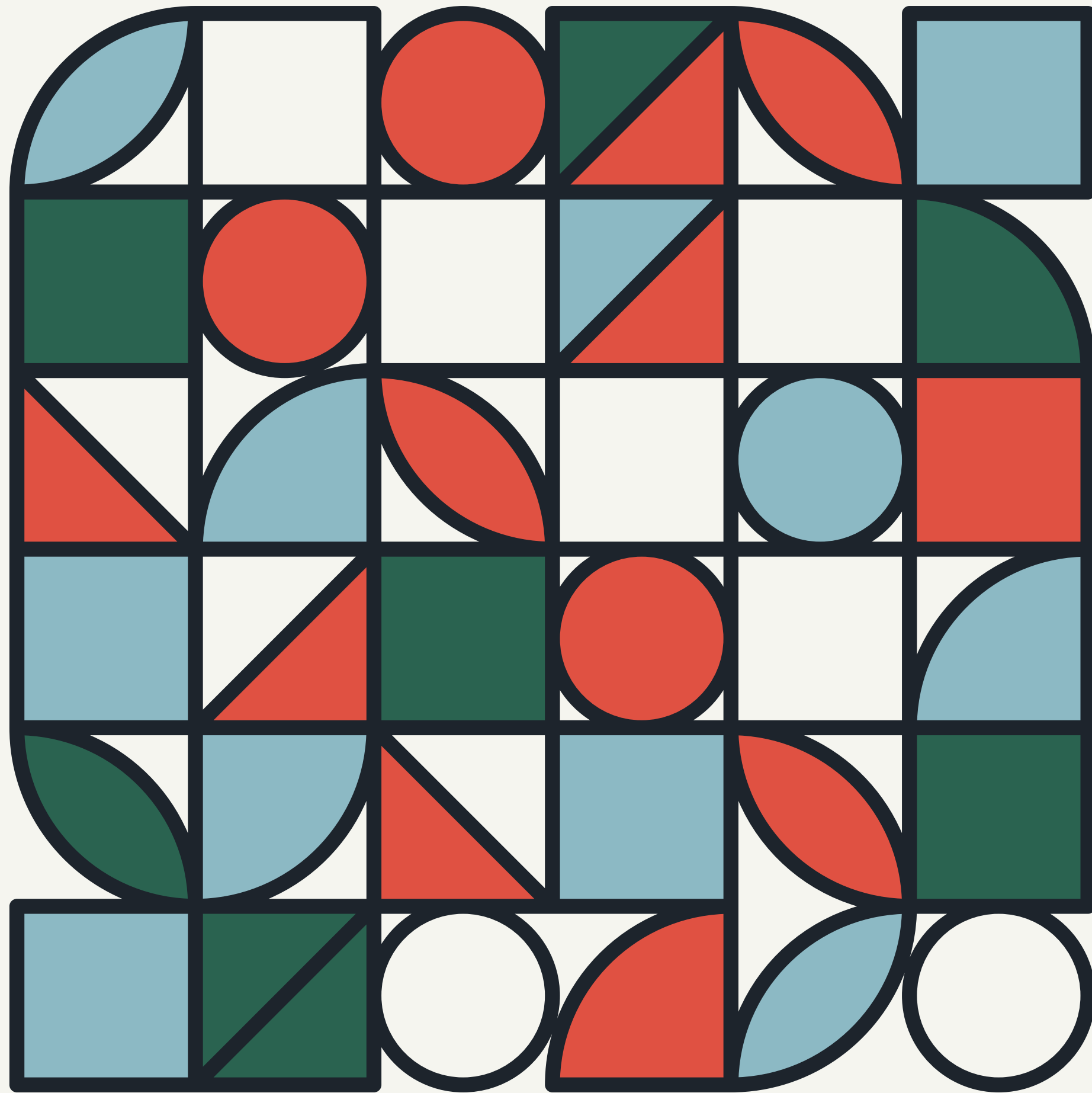
RAW MODE AND COOKED MODE

RAW MODE

- **Definition:** In raw mode, the input is passed directly to the program without any processing by the terminal driver. This means that characters are delivered to the program immediately as they are typed, without waiting for a newline (Enter) and without any interpretation (e.g., no special handling of Ctrl+C, Ctrl+Z, backspace, etc.).
- **Use Cases:** Raw mode is typically used in programs that need real-time input handling, like games, text editors, or command-line tools that require immediate response to each keystroke.

- **Definition:** In cooked mode (also known as canonical mode), the terminal driver processes the input before it is sent to the program. Input is typically line-buffered, meaning that it is only sent to the program after the user presses Enter. The terminal driver also interprets special characters, such as backspace for deleting characters and Ctrl+C for sending an interrupt signal.
- **Use Cases:** Cooked mode is suitable for most standard command-line applications where line-by-line input is sufficient, and where special handling (like signal generation on Ctrl+C) is desired.

COOKED MODE

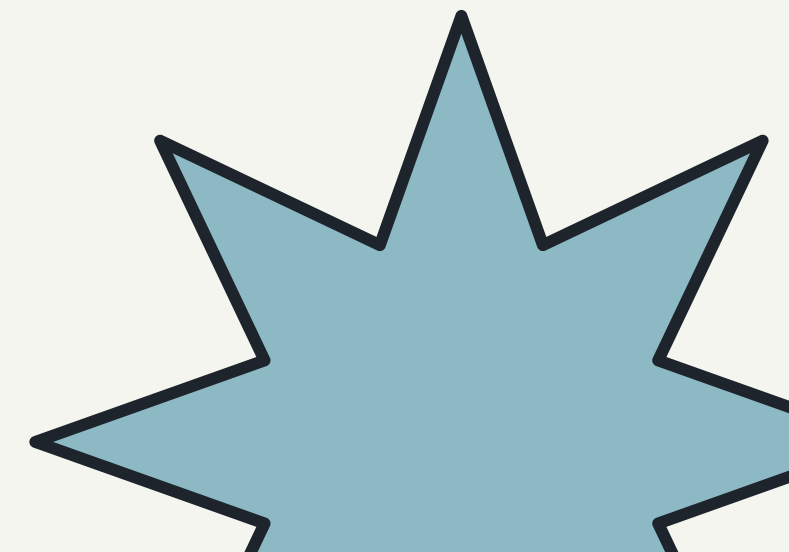


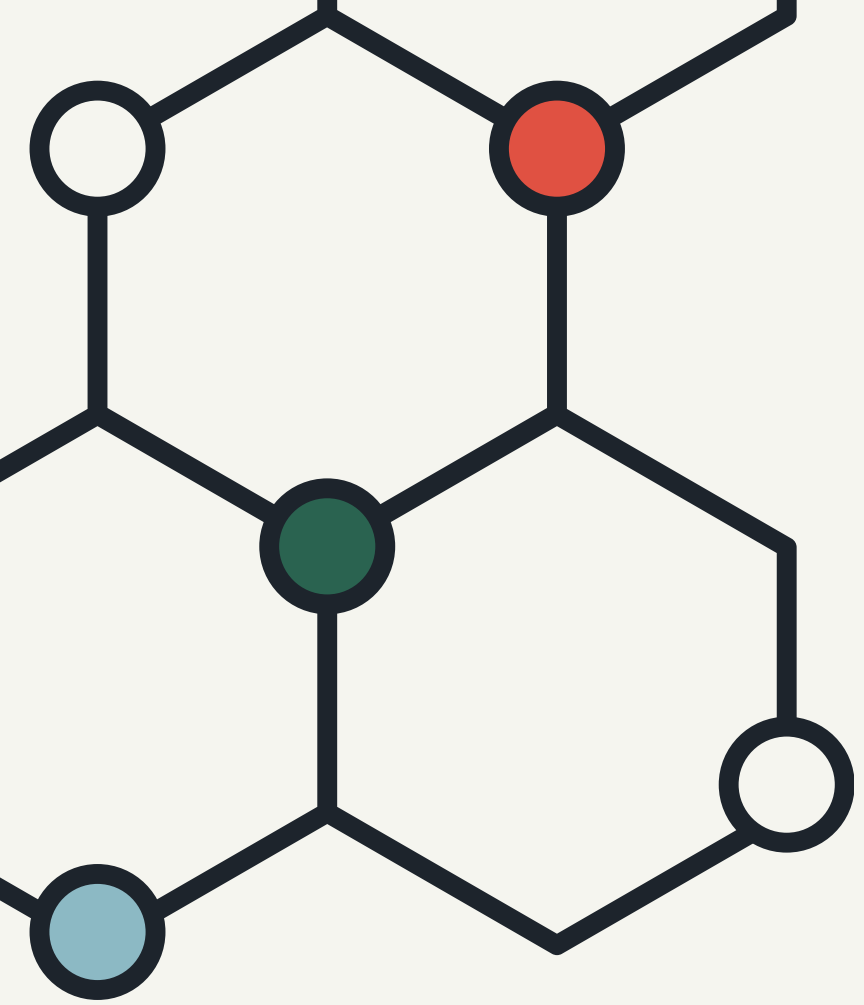
PRO TIP

READ MAN PAGES.

WRITE MODULAR CODE.

START ON TIME.





THANK YOU!

