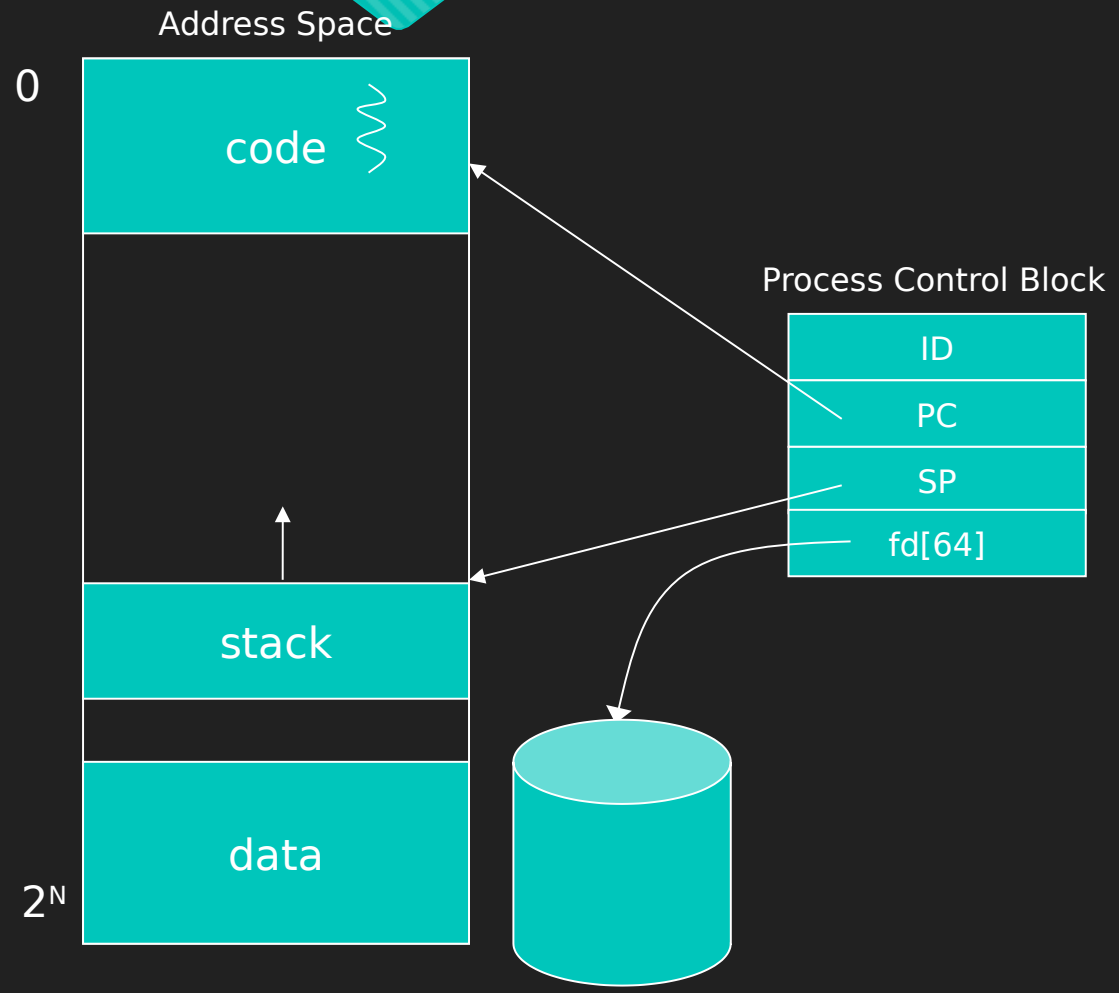
Parallel and Concurrent Programming CS3S666

Threading

Contents of a Process

- An environment to execute a program
- A process consists of:
  - A Process Control Block (Kernel info)
  - A dependent address space including
    - Code.
    - Data (global data)
    - Stack (local variables)
    - Heap (dynamic memory)
  - Files

Process Contents

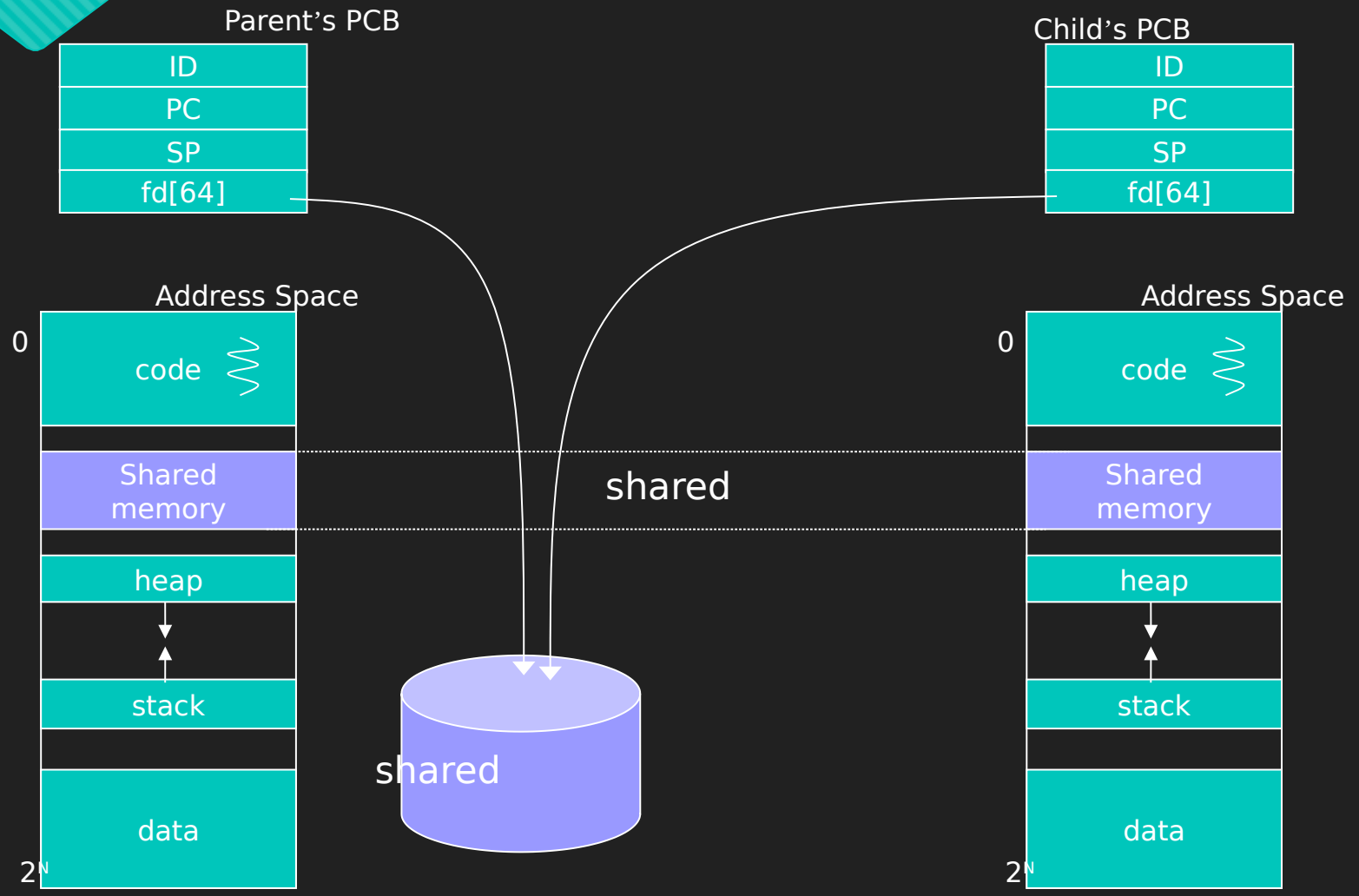


- ID = ID of process
- PC = Program Counter
  - Is a processor register that indicates where a computer is in its program sequence
- SP = Stack Pointer / Stack Register
  - Is a computer central processor register whose purpose is to keep track of a call stack
- FD = File Descriptor

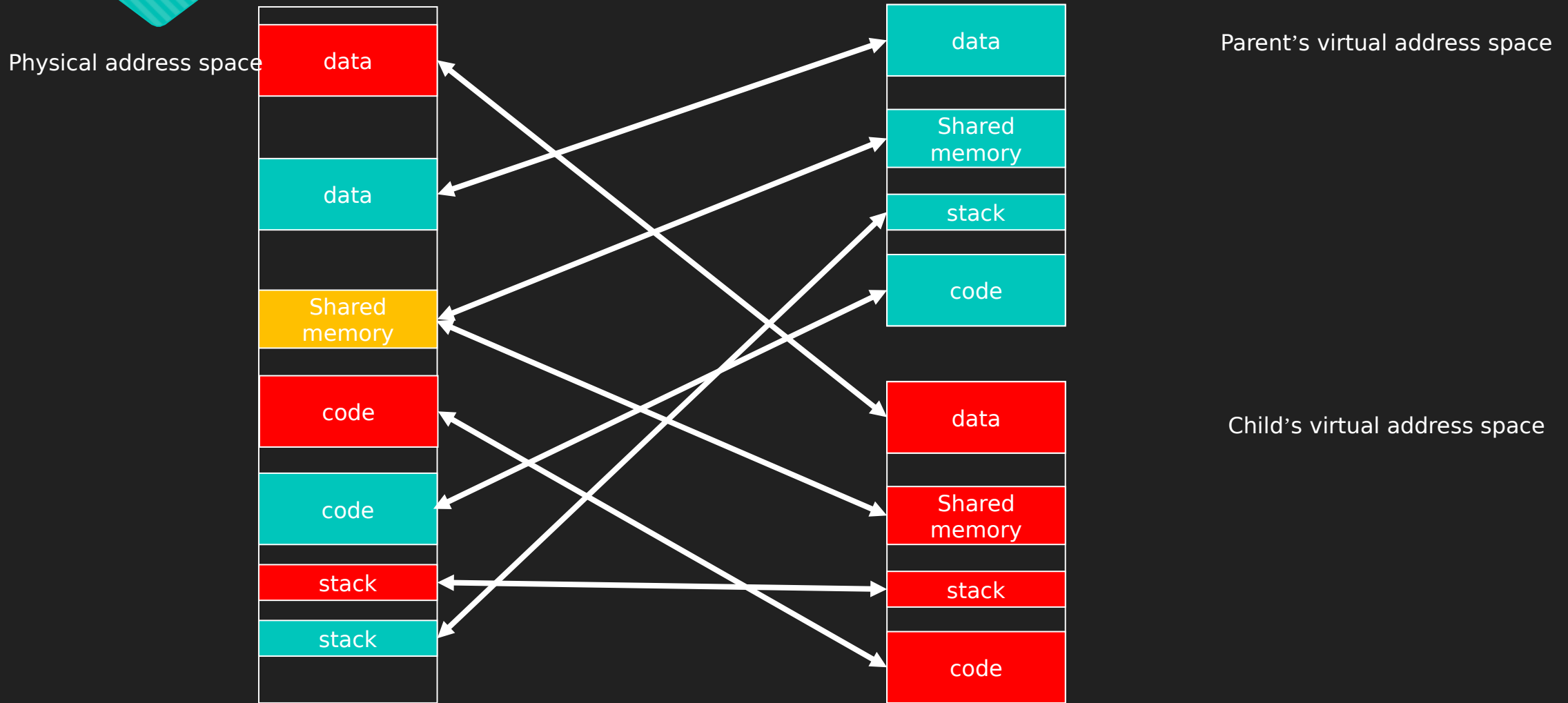
Process

- A parent process creates child processes through ***fork( )***.
- Resource sharing
  - Resource inherited by children:
    - file descriptors
    - shared memory
  - Resource not inherited by children:
    - address space

Process Memory



Process Memory



Process Memory

- The operating system prevents a process accessing (or corrupting) the memory of another process (except shared memory).

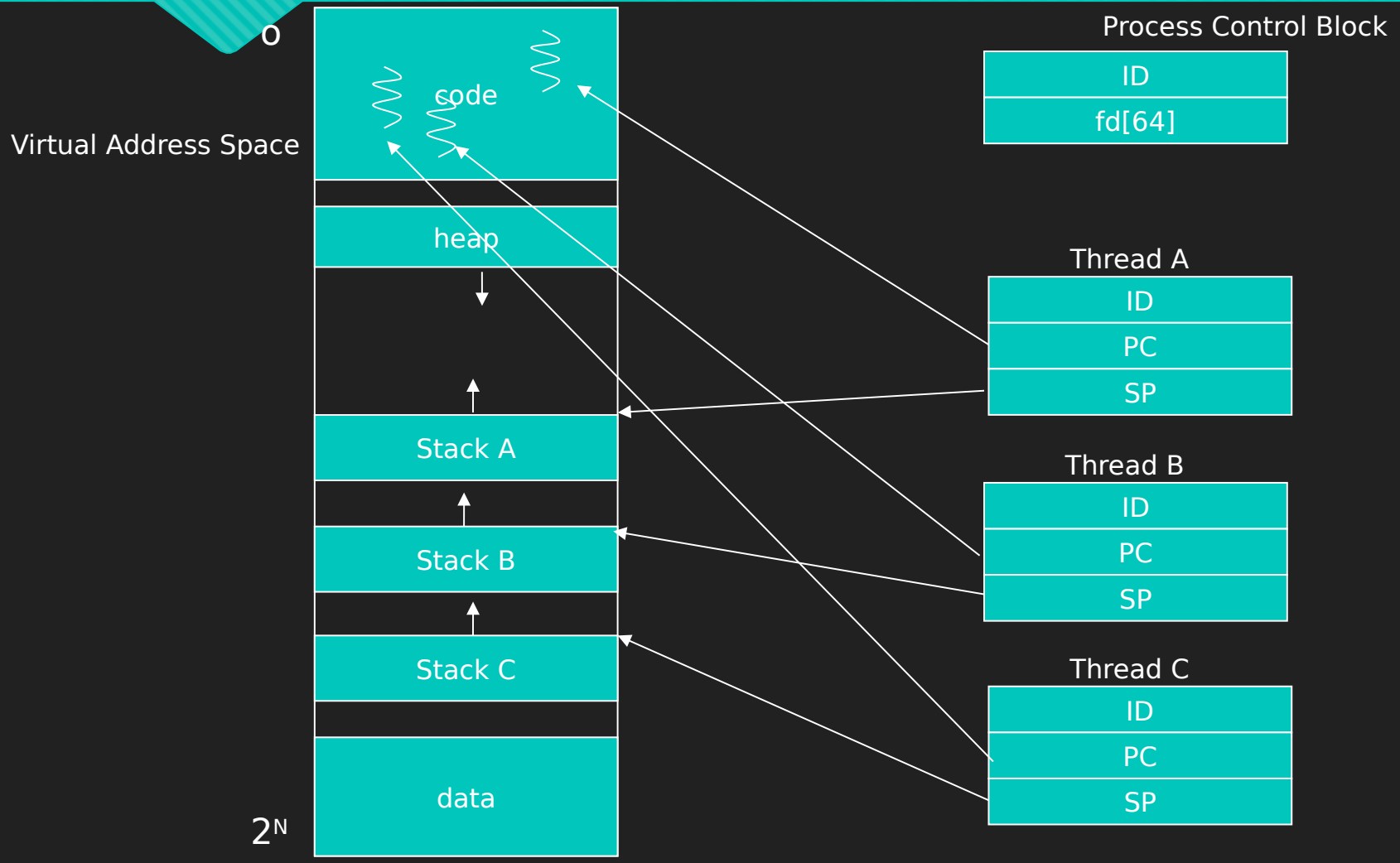
- If a process corrupts it’s own memory (e.g. stack), other processes shouldn’t be impacted.

Threads

- A thread is similar to a process (conceptually at least), except it shares a common address space.
  - All memory is shared memory*!!
- The downside is it’s much easier to corrupt your own memory.

* In that it’s accessible from all threads

Thread Memory Space



There’s nothing stopping thread A accessing thread B’s stack data.

Thread Benefits

- Advantages:
  - Light creation
  - Light context switch
  - Suitable to parallel computing
  - Natural form of resource sharing

Threads

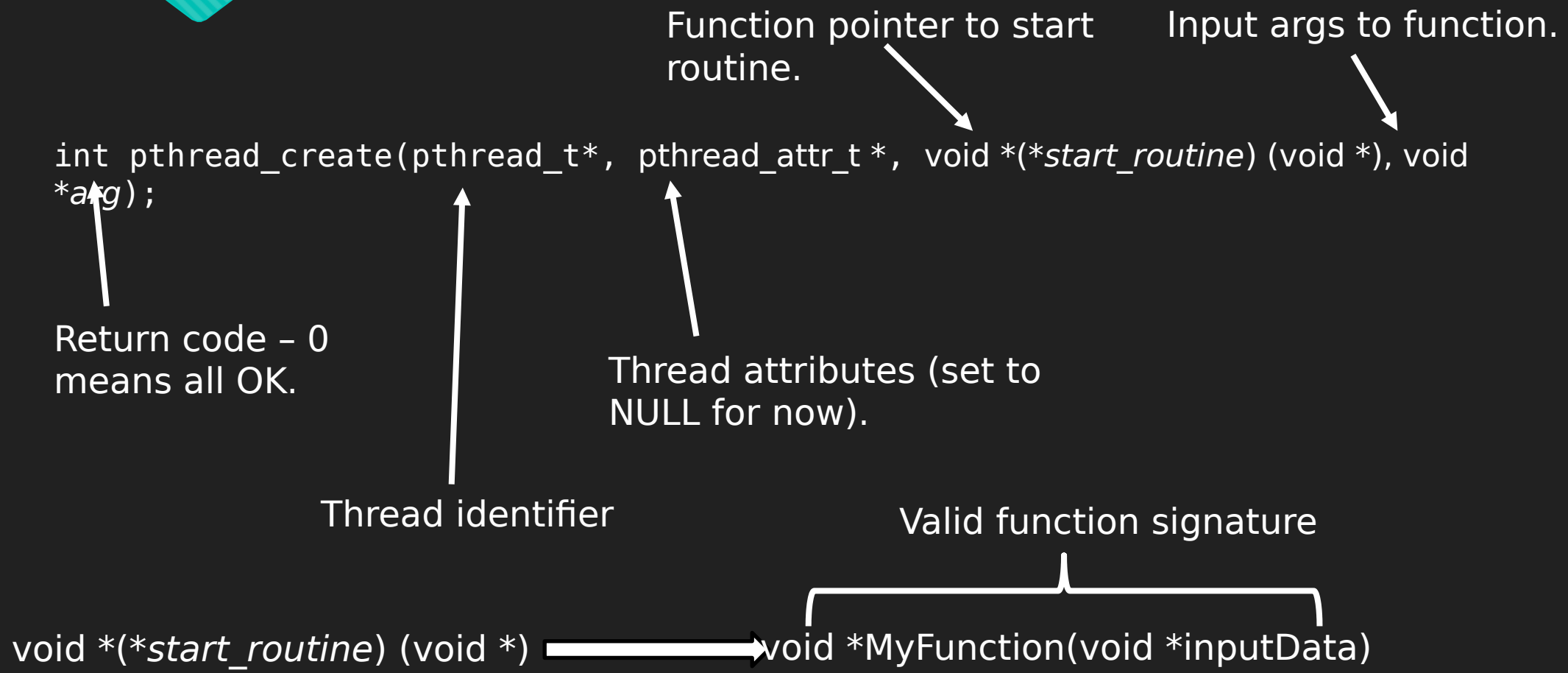
- The initial program will contain one thread.
- A thread can spawn other threads.
- Threads are created equal!
- There is no hierarchy or dependency.

POSIX Threads (Pthreads)

- UNIX system specific threading interface.
- Required header pthead.h.
- To compile

  g++ -pthread –o objectname cppname.cpp

Getting Started



Exiting a Thread

- Several options:
  - Start routine returns.
  - Thread calls pthread_exit to terminate itself.
  - Another thread calls pthread_cancel.
  - Exit() is called.
  - Main terminates.
- Good news – no zombie threads!

Join()

- Equivalent to processes Join (wait for thread to finish).

```
int pthread_join(pthread_t, void **retval);
```

0 means no error

Thread to wait for.

Gets status of thread (use NULL for now).

```cpp
#include<cstdlib>
#include<iostream>
#include<unistd.h>
#include<pthread.h>


using namespace std;

void *HelloWorld(void *)
{
  sleep(2);
  cout << "Hello
  return NULL;
}

int main()
{
  pthread_t thread;
  int rc = pthread_create(&thread, NULL, HelloWorld, NULL);
  if (rc != 0)
  {
    cout << "Thread creation failed." << endl;
  }

  cout << "Hello World from main thread!" << endl;

  pthread_join(thread, NULL);
  return 0;
}
```

```
simon@simon-VirtualBox:~$ g++ -pthread -o Example Example.cpp
simon@simon-VirtualBox:~$ ./Example
Hello World from main thread!
Hello World from the new thread!
simon@simon-VirtualBox:~$
```

Function to call on thread.

Thread Creation.

Wait for specified thread.

Passing Data

- Data must be passed in as a void ptr (void*).
- You must therefore do the relevant casting.

```
1  void *HelloWorld(void *data)
2  {
3    sleep(2);
4
5    int *num = (int*)da
6
7    cout << "Hello World from new thread number: " << *num << endl;
8    return NULL;
9  }
10
11 int main()
12 {
13   pthread_t thread;
14   int num_in = 5;
15   int rc = pthread_create(&thread, NULL, HelloWorld, (void *)&num_in);
16   if (rc != 0)
17   {
18     cout << "Thread failed." << endl;
19   }
20
21   cout << "Hello World from main thread number: " << num_in << endl;
22
23   pthread_join(thread, NULL);
24   return 0;
25 }
```

Cast to int pointer

```
simon@simon-VirtualBox:~$ g++ -pthread -o Example Example.cpp
simon@simon-VirtualBox:~$ ./Example
Hello World from main thread number: 5
Hello World from the new thread number: 5
simon@simon-VirtualBox:~$
```

Cast to null pointer

```cpp
void *HelloWorld(void *data)
{
  int *num = (int*)data;
  *num = 4;
  Sleep(2);

  cout << "Hello World from new thread number: " << *num << endl;
  return NULL;
}

int main()
{
  pthread_t thread;
  int num_in = 5;
  int rc = pthread_create(&thread, NULL, HelloWorld, (void *)&num_in);
  if (rc != 0)
  {
    cout << "Thread failed." << endl;
  }
  sleep(1);
  cout << "Hello World from main thread number: " << num_in << endl;

  pthread_join(thread, NULL);
  return 0;
}
```

Change here.

```
simon@simon-VirtualBox:~$ g++ -pthread -o Example Example.cpp
simon@simon-VirtualBox:~$ ./Example
Hello World from main thread number: 4
Hello World from the new thread number: 4
simon@simon-VirtualBox:~$
```

Changes here. Why?

Summary

- You now know how to create threads.
- They share similarities with processes.
- Next Week synchronisation and critical sections.