Кирилл Волков @ MERA
github.com/vulko/Cpp_Basics_Lectures

C++

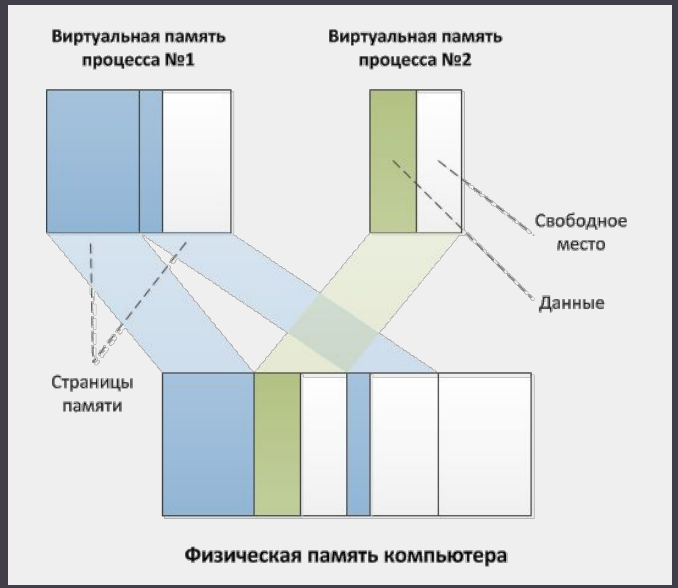
Парадигма исполняемого приложения в современных ОС. Процесс, поток и работа с памятью.

Однопоточность vs многопоточность. Создание и управление потоками. Основные проблемы и способы их решения. Принципы синхронизации и разделяемого доступа.

Парадигма работы ОС

- OS manages memory as pages
- Process is an instance of a program, being executed
- Process has a memory assosiated with it and one or multiple execution threads
- OS runs multiple independent processes to avoid system failures
- Process has MAIN thread of execution

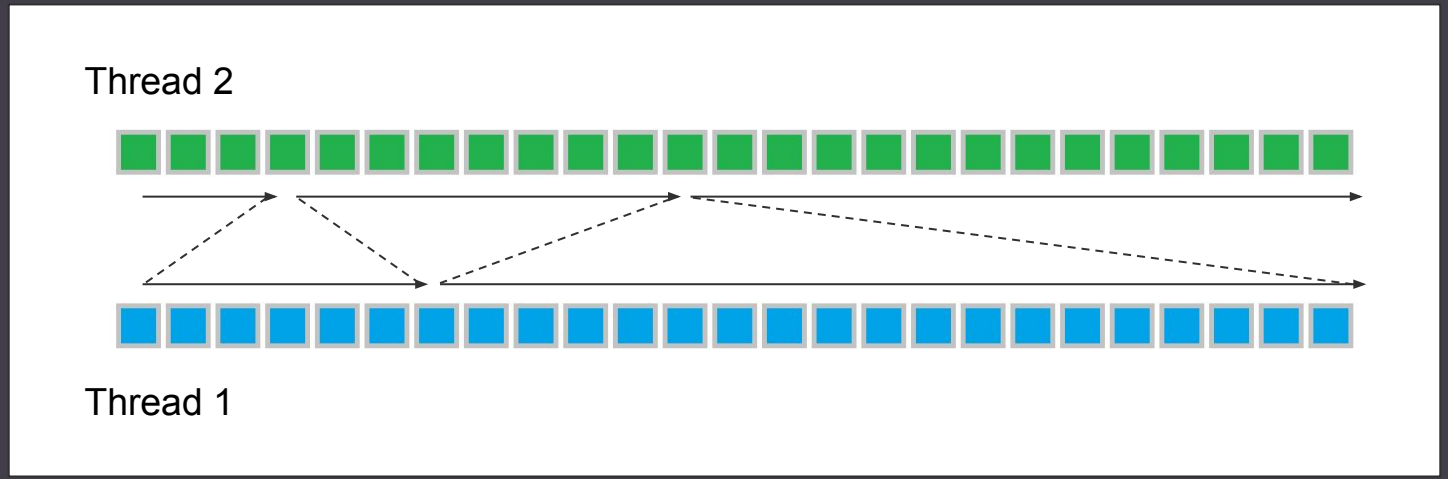
- Thread is a sequence of commands
- Threads exist only as a part of a process
- Multiple threads share state, memory and other resources of a parent process
- Context switching btw threads is faster than context switching btw processes



Процесс

- Memory (typically some region of virtual memory)
    - executable code
    - process-specific data (input and output)
    - call stack
    - heap to hold data generated during run time
- OS descriptors such as file descriptors or handles, data sources and sinks.
- Security attributes, such as the process owner and the set of permissions
- Processor state (context)
    - content of registers and physical memory addressing
    - state is typically stored in computer registers when the process is executing, and in memory otherwise
- OS holds most of this information about active processes in process control blocks
- Most OS have IPC mechanisms

- Multiple threads can run on a single CPU core
- Process switches between threads, so they are executed consequently
- No need for synchronization
- No performance bonus, instead overhead when switching between threads



- Multiple threads can run on several CPU cores simultaneously
- Due to asynchronous behavior, might require synchronization (e.g. use of shared data)
- Potential deadlocks, race conditions, etc
- Performance bonus, no overhead when switching between threads

std::thread

Implements thread API

Runs a function in a separate thread

```
std::thread(FunctionPointer, Args...);
```

Start an std::thread

```
void f1(int n)
{
    for (int i = 0; i < 5; ++i) {
        std::cout << "Thread 1 executing\n";
        ++n;

        std::this_thread::sleep_for(
            std::chrono::milliseconds(10));
    }
}
```

```
// pass by value
std::thread t1(f1, n + 1);
```

Start an std::thread

```
void f2(int& n)
{
    for (int i = 0; i < 5; ++i) {
        std::cout << "Thread 2 executing\n";
        ++n;

        std::this_thread::sleep_for(
            std::chrono::milliseconds(10));
    }
}
```

```
// pass by reference
std::thread t2(f2, std::ref(n));

// t4 is now running f2().
// t3 is no longer a thread
std::thread t3(std::move(t2));
```

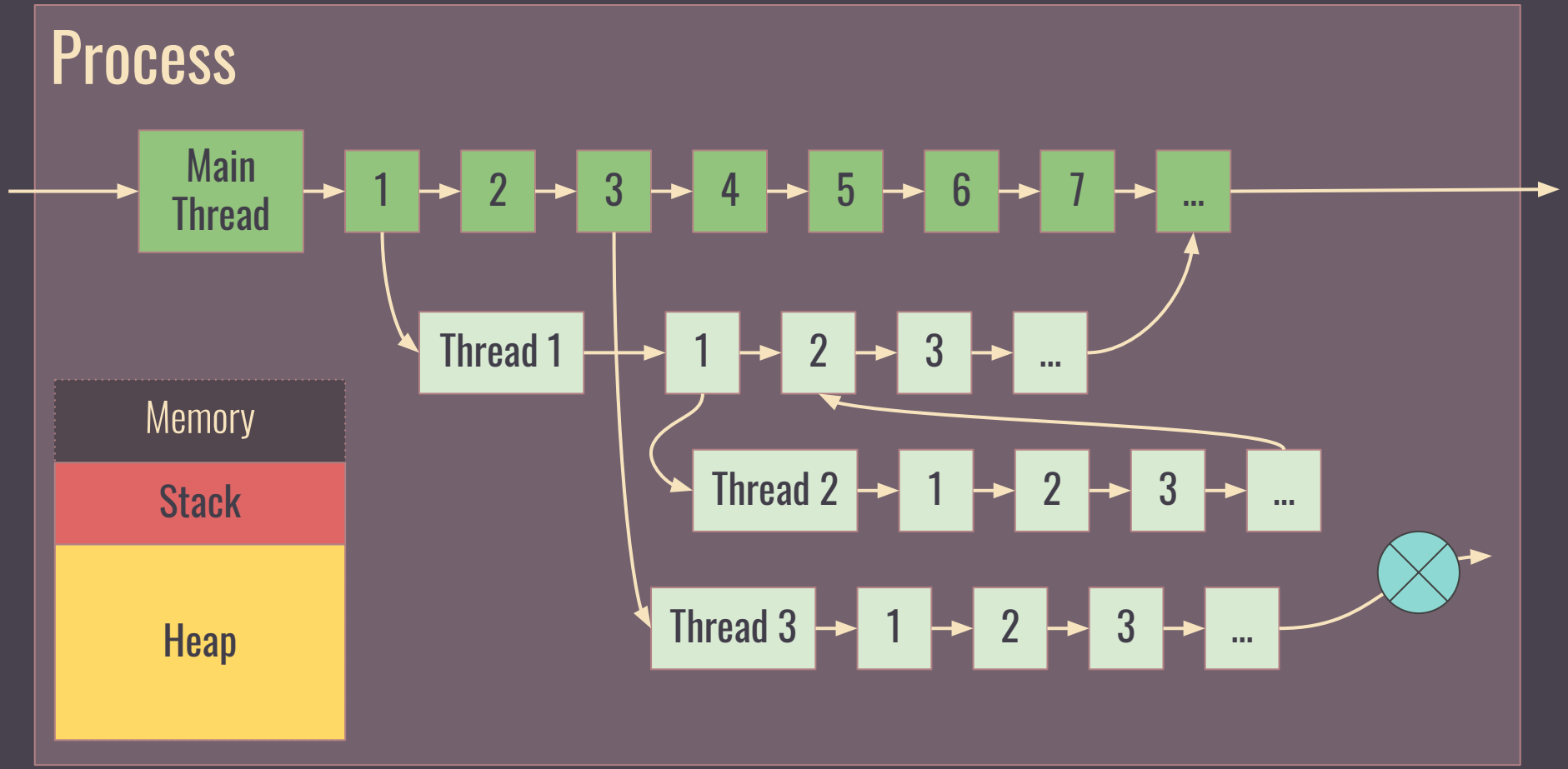
Start an std::thread

```
class foo {
public:
    void bar() {
        for (int i = 0; i < 5; ++i) {
            std::cout << "Thread 3 executing\n";
            ++n;
            std::this_thread::sleep_for(
                std::chrono::milliseconds(10));
        }
    }
    int n = 0;
};
```

```
foo f;
foo f1;

// t4 runs foo::bar() on object f
std::thread t4(&foo::bar, &f);

// t5 runs foo::bar() on object f1
std::thread t4(&foo::bar, &f1);
```


Process
Main Thread
1
2
3
4
5
6
7
...
Thread 1
1
2
3
...
Thread 2
1
2
3
...
Thread 3
1
2
3
...
Memory
Stack
Heap