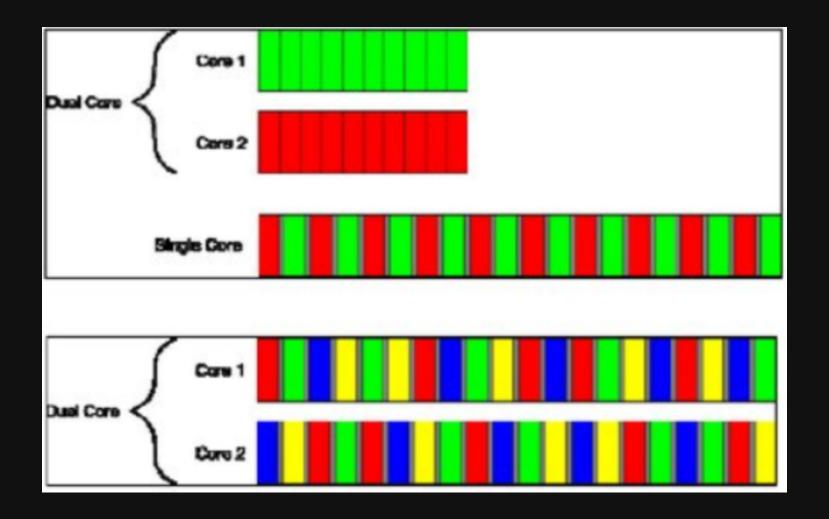
Сибирь II

Многопоточность

Однопоточные процессоры не способны выполнять задачи параллельно, но могут создавать иллюзию этого (получается асинхронность)



Задачи параллелизма:

- клиент-серверные приложенияработа "сложной" ОС
- повышение производительности

std::thread

Позволяет запускать потоки выполнения

Программист *обязан* правильно управлять потоками

Есть 2 пути:

- .join() ожидаем конца выполнения потока и очищаем память потока
- detach() отпускаем поток работать в фоновом режиме, но так можно делать только в "бесконечных" приложениях

```
#include <iostream>
#include <thread>
void foo(const std::string& num)
  for (size t i = 0; i < 10000; ++i) {
    std::cout << i << ' from ' << num << std::endl;
int main() {
  std::thread th1(foo, "Thread 1");
  std::thread th2(foo, "Thread 2");
  th1.join(); // не идём дальше, пока поток не закончит свою рабо
  th2.join();
```

std::jthread (c++20) – автоматически выполняет join при вызове деструктора Потоки некопируемы, но перемещаемы

Средства синхронизации

Для правильного использования многопоточности

std::mutex

Позволяет защитить данные от суеты множества потоков

- mutex.lock() захватывает мьютекс
- mutex.try_lock() захватывает мьютекс, если свободен, иначе возвращает false
- mutex.unlock() освобождение мьютекса

Лучше использовать std::unique_lock и std::lock_guard(мы же все-таки за RAII)

```
#include <iostream>
#include <thread>
#include <mutex>
#include <vector>
std::vector<size t> nums;
size t i;
std::mutex my mutex;
void foo(const std::string& num) {
  for (; i < 100; ++i) {
    std::lock guard<std::mutex> guard(my mutex); // захват мьютек
    nums.push back(i);
    std::cout << i << " from " << num << std::endl;
int main() {
  std::thread th1(foo, "Thread 1");
  th1.detach();
  std::thread th2(foo, "Thread 2");
  th2.join();
```

std::atomic

Предоставляет работу с атомарными операциями

Атомарная – операция, которую невозможно наблюдать в промежуточном состоянии (она либо выполнена, либо нет).

```
#include <atomic>
#include <iostream>
#include <thread>
#include <vector>
std::atomic int acnt;
int cnt;
void f() {
    for (int n = 0; n < 10000; ++n) {
        ++acnt; // acnt.fetch add(1, std::memory order relaxed);
        ++cnt;
int main() {
    std::vector<std::jthread> pool;
    for (int n = 0; n < 10; ++n)
      pool.emplace back(f);
  std::cout << "The atomic counter is " << acnt << '\n'
            << "The non-atomic counter is " << cnt << '\n';
```

