std::promise/std::future in C++

CODERS SCHOOL

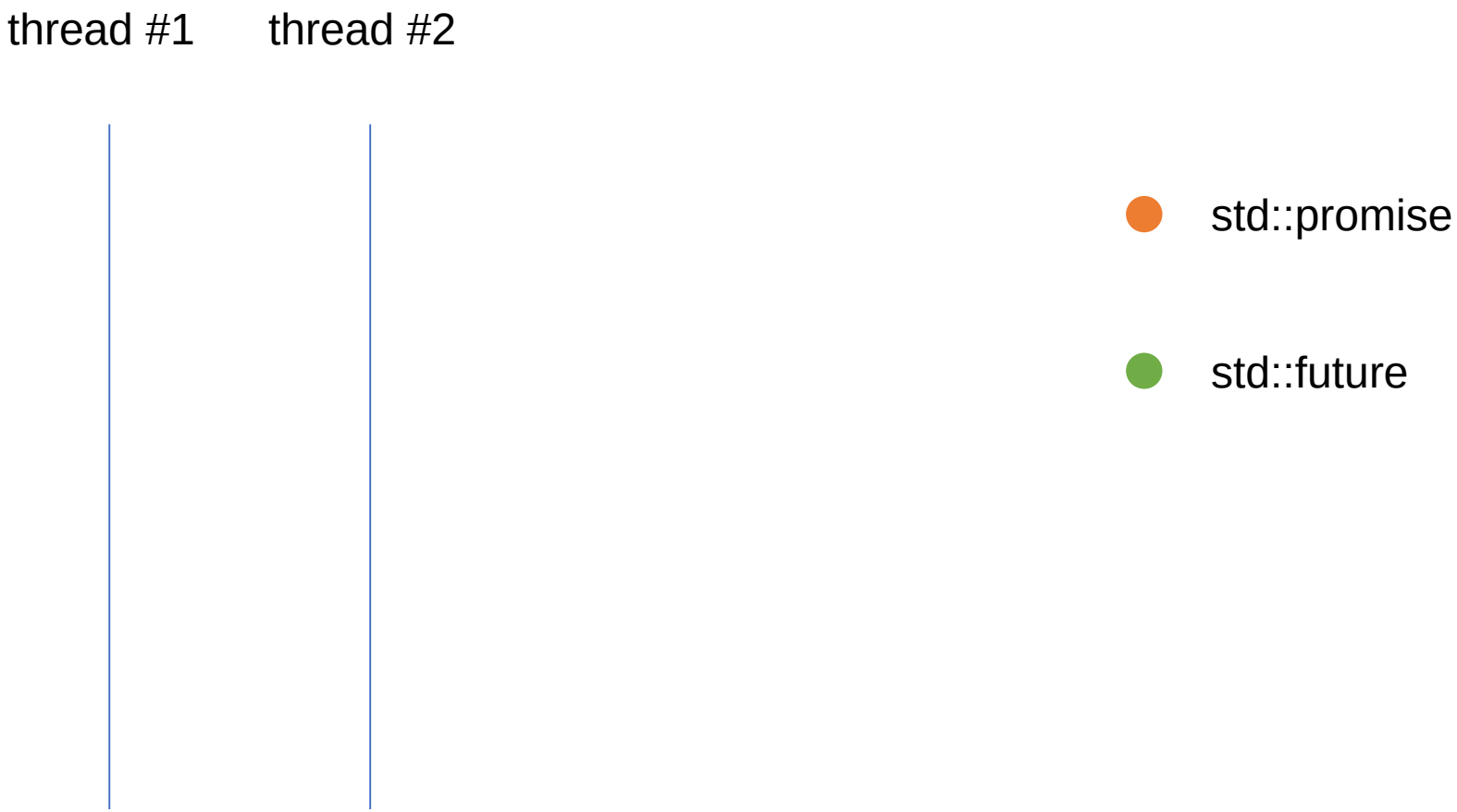
LEVEL UP BEFORE YOU START

Ihor Rudynskyi

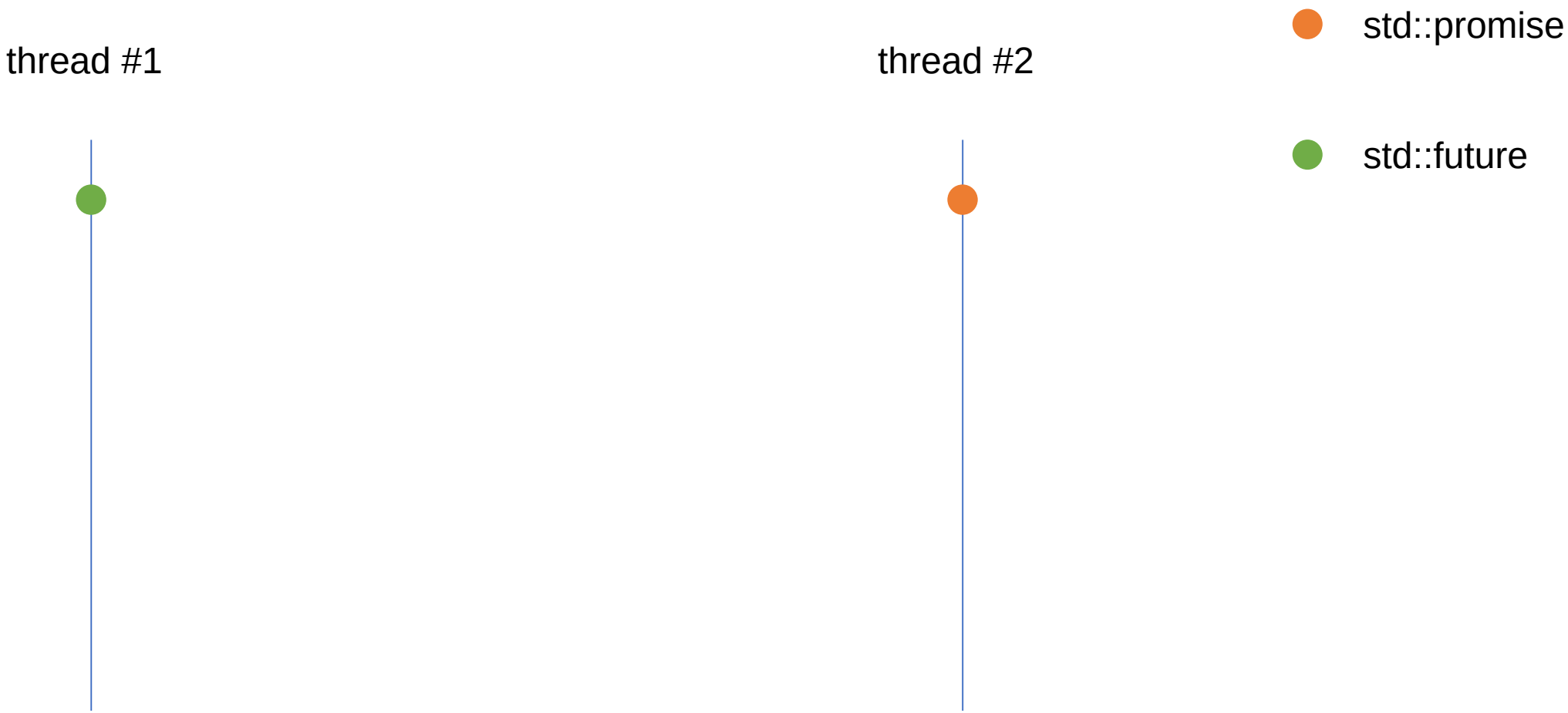
std::thread is a very “low-level” mechanism

- How to return something from it?
- How to forward an exception?
- Should be manually joined/detached

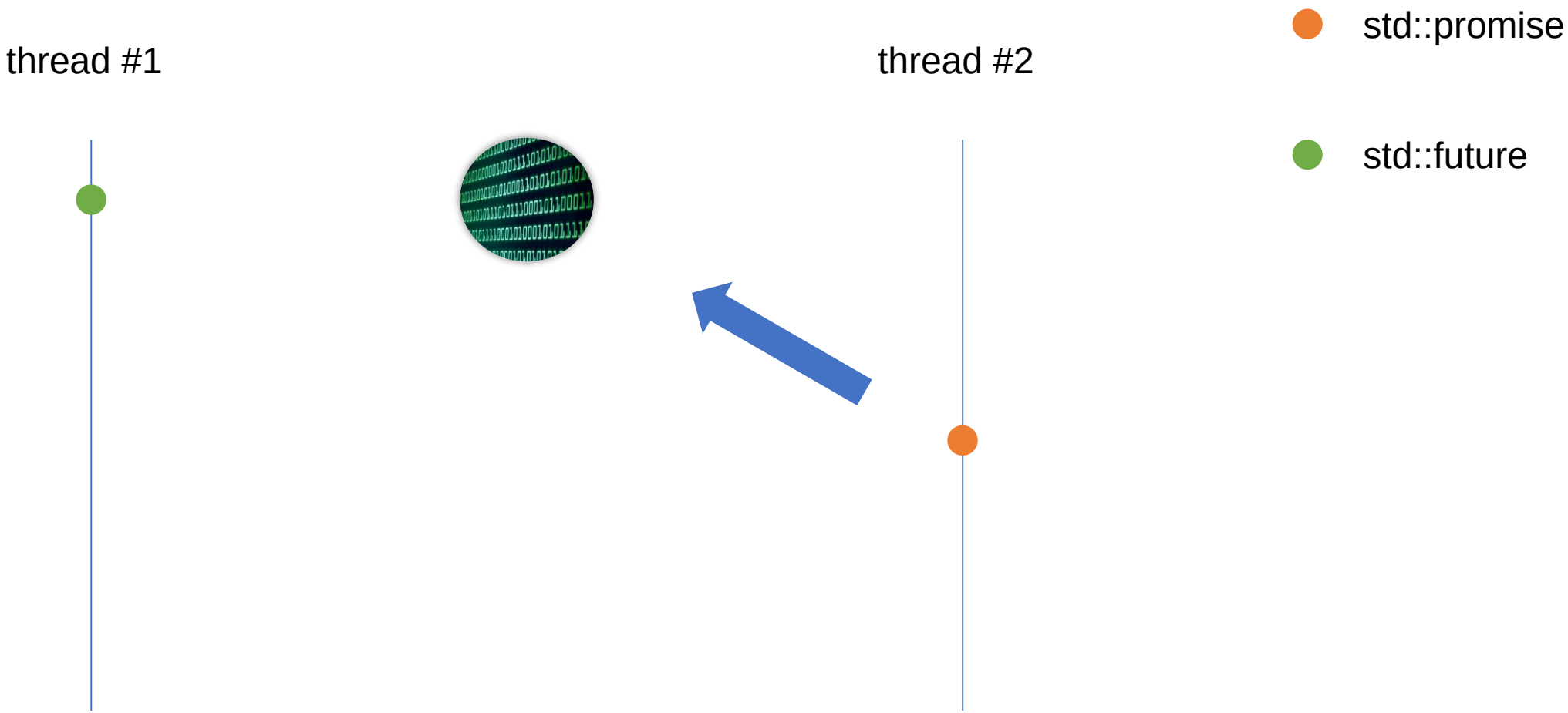
How std::promise/std::future works



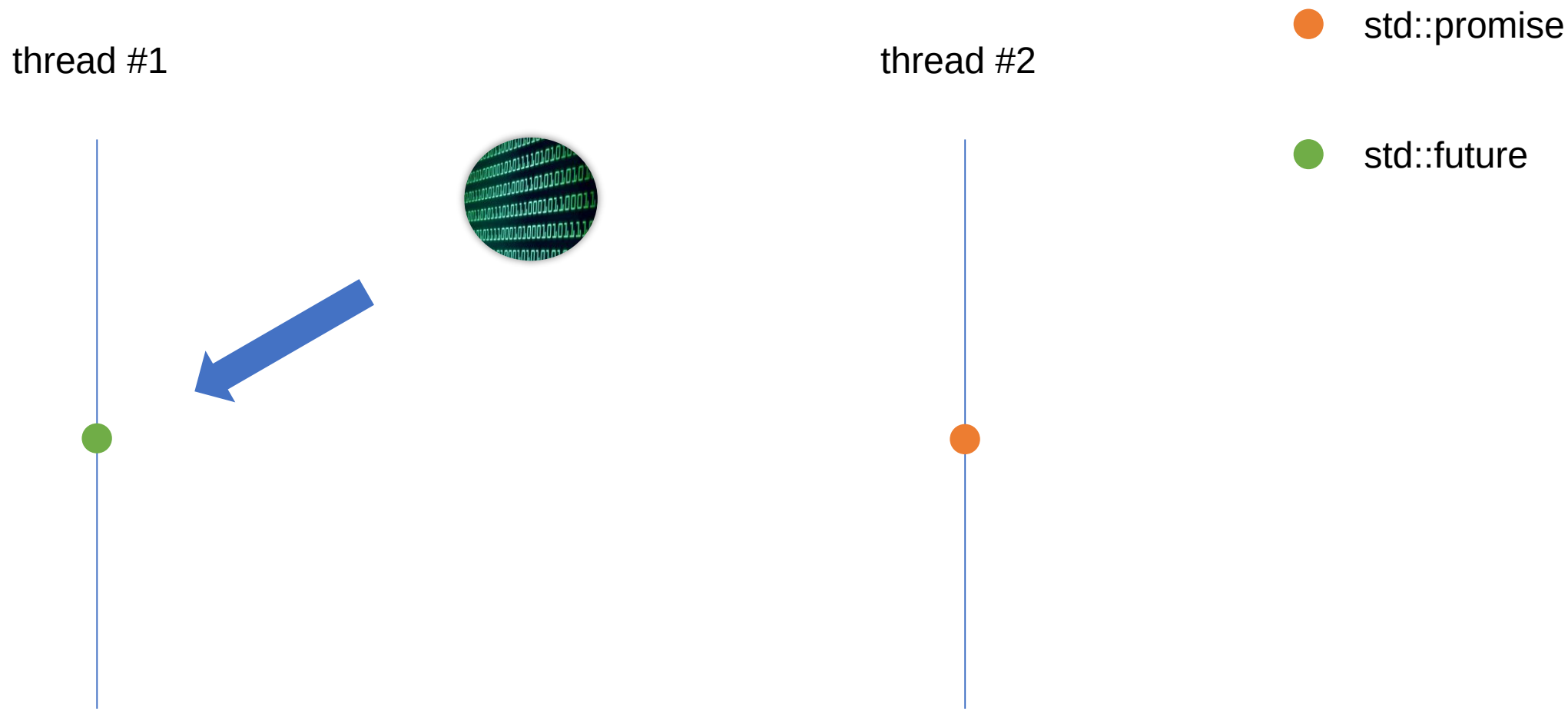
How std::promise/std::future works



How std::promise/std::future works



How std::promise/std::future works



Basic std::promise/std::future usage

```cpp
std::promise<int> promise;
std::future<int> future = promise.get_future();
auto function = [] (std::promise<int> promise)
{
    // ...
    promise.set_value(10);
};
std::thread t(function, std::move(promise));
std::cout << future.get() << std::endl;
t.join();
```

- std::promise/std::future are used to create one-way communication channel
- std::promise is used for setting value
- std::future is used for getting value

std::promise/std::future can be used only once

Find broken piece of code

```
#1 auto future = promise.get_future();
#2 promise.set_value(10);
#1 future.get();
```

```
#2 promise.set_value(10);
#1 auto future = promise.get_future();
#1 future.get();
```

```
#1 auto future = promise.get_future();
#1 future.get();
#2 promise.set_value(10);
```

```
#1 auto future = promise.get_future();
#1 future.get();
#2 // set wasn't called
```

Find broken piece of code

OK

```
#1 auto future = promise.get_future();
#2 promise.set_value(10);
#1 future.get();
```

```
#2 promise.set_value(10);
#1 auto future = promise.get_future();
#1 future.get();
```

```
#1 auto future = promise.get_future();
#1 future.get();
#2 promise.set_value(10);
```

```
#1 auto future = promise.get_future();
#1 future.get();
#2 // set wasn't called
```

Find broken piece of code

OK

```
#1 auto future = promise.get_future();
#2 promise.set_value(10);
#1 future.get();
```

OK

```
#2 promise.set_value(10);
#1 auto future = promise.get_future();
#1 future.get();
```

```
#1 auto future = promise.get_future();
#1 future.get();
#2 promise.set_value(10);
```

```
#1 auto future = promise.get_future();
#1 future.get();
#2 // set wasn't called
```

Find broken piece of code

OK

```
#1 auto future = promise.get_future();
#2 promise.set_value(10);
#1 future.get();
```

OK

```
#2 promise.set_value(10);
#1 auto future = promise.get_future();
#1 future.get();
```

OK

```
#1 auto future = promise.get_future();
#1 future.get();
#2 promise.set_value(10);
```

```
#1 auto future = promise.get_future();
#1 future.get();
#2 // set wasn't called
```

Find broken piece of code

OK

```
#1 auto future = promise.get_future();
#2 promise.set_value(10);
#1 future.get();
```

OK

```
#2 promise.set_value(10);
#1 auto future = promise.get_future();
#1 future.get();
```

OK

```
#1 auto future = promise.get_future();
#1 future.get();
#2 promise.set_value(10);
```

OK

```
#1 auto future = promise.get_future();
#1 future.get();
#2 // set wasn't called
```

More about std::promise

```
std::thread t(function, std::move(promise));
```

- std::promise/std::future can be only moved

```
std::future<int> future = promise.get_future();
```

- returns a future associated with the promised result
- second call will throw

How to „set"?

- set value

```
promise.set_value(10);
```

- set exception

```
promise.set_exception(std::make_exception_ptr(e));
```

```
try {
    // ...
} catch (...) {
   promise.set_exception(std::current_exception());
}
```

std::promise can be „set" only once

More about std::future

```
future.valid();
```

- checks if the future can be used

```
future.wait();
```

- waits for the result to become available

do not use „invalid" future

How to „get"?

```
future.get();
```

- waits for the result to become available and returns the result
- will automatically throw stored exception
- will invalidate the future

std::future can be „get" only once

Package this function #1

```
int get_number()
{
    return 10;
}

int main()
{
    auto future = get_number_async();
    return future.get();
}
```

Package this function #1

```
std::future<int> get_number_async()
{
    std::promise<int> p;
    std::future<int> f = p.get_future();
    auto wrapped_func = [] (std::promise<int> p)
    {
        p.set_value(get_number());
    };
    std::thread t(wrapped_func, std::move(p));
    t.detach();
    return f;
}
```

Package this function #2

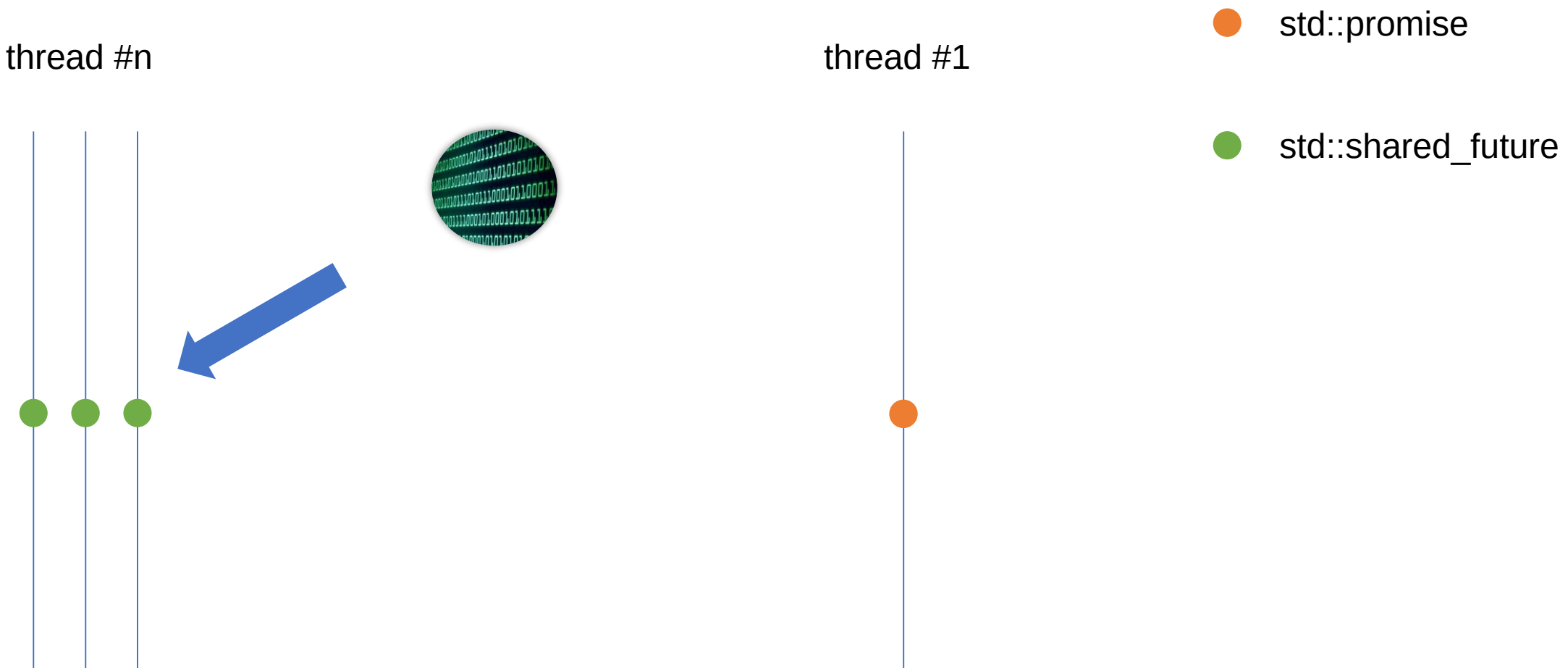
```
int get_number()
{
    return 10;
}

int main()
{
    auto future = schedule(get_number);
    return future.get();
}
```

Package this function #2

```
std::future<int> schedule(std::function<int()> func)
{
    std::promise<int> p;
    std::future<int> f = p.get_future();
    auto wrapped_func = [func] (std::promise<int> p) {
        try {
            p.set_value(func());
        } catch(...) {
            p.set_exception(std::current_exception());
        }
    };
    std::thread t(wrapped_func, std::move(p));
    t.detach();
    return f;
}
```

One-to-many connection



One-to-many connection

```
std::shared_future<int> sfuture = promise.get_future().share();
```

- allows multiple getting
- copyable and movable
- each thread should have its own shared_future object

std::shared_promise does not exist

Pytania?

CODERS SCHOOL
www.coders.school