
Optiver

Optiver

Advanced C++ Programming

University of NSW Guest Lecture

Gregory Saunders | 17.04.23

Why am I Here Today?

- Optiver is proud to sponsor Advanced C++ Programming at the University of New South Wales.

- At Optiver developers design, build and maintain a world-class automated trading platform, mostly in C++.
- This means:
  - Designing, developing, testing and deploying their own systems.
  - Choosing appropriate algorithms and data-structures.
  - Optimising their systems for low-latency.
  - Employing up-to-date, industry-best practice.

- This course supports these skills and provides a great foundation for working with Optiver
- Optiver donates $2,250 in prizes for the best performers in the course.

Agenda

How we use C++ at Optiver

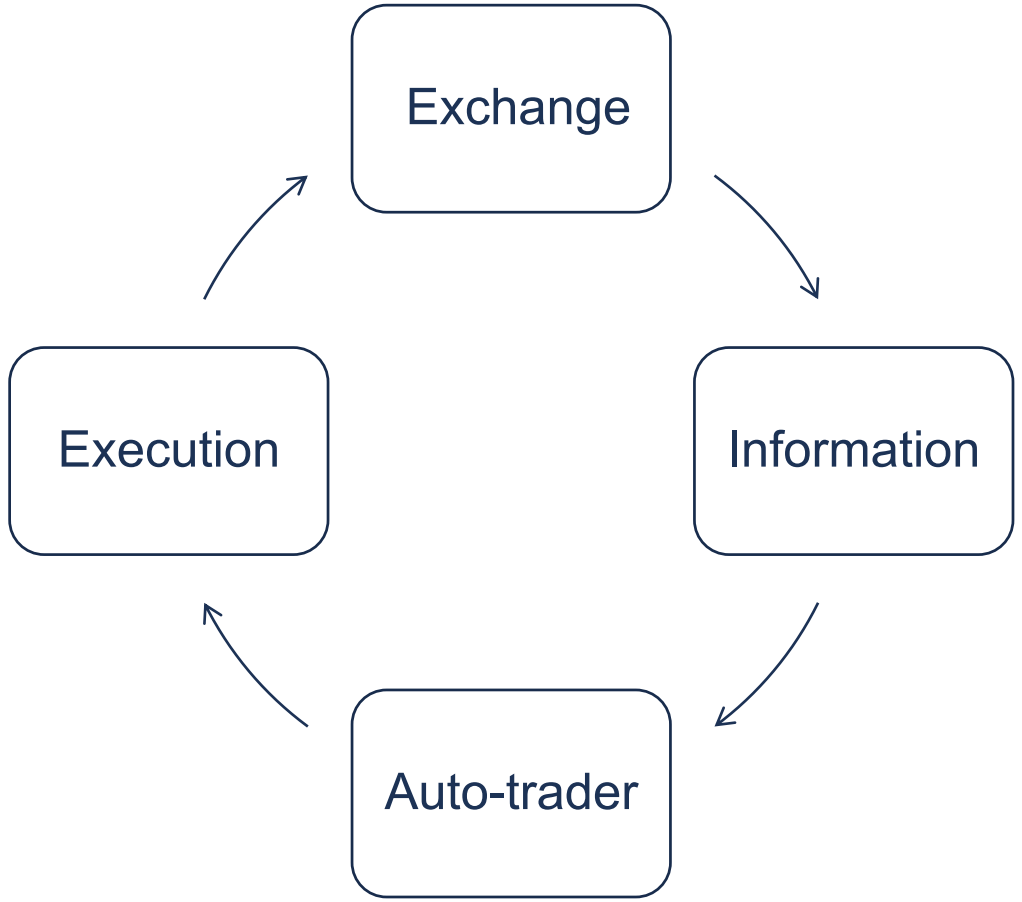
Features of C++20 in Depth

Optiver

How we use C++ at Optiver

How we use C++ at Optiver?

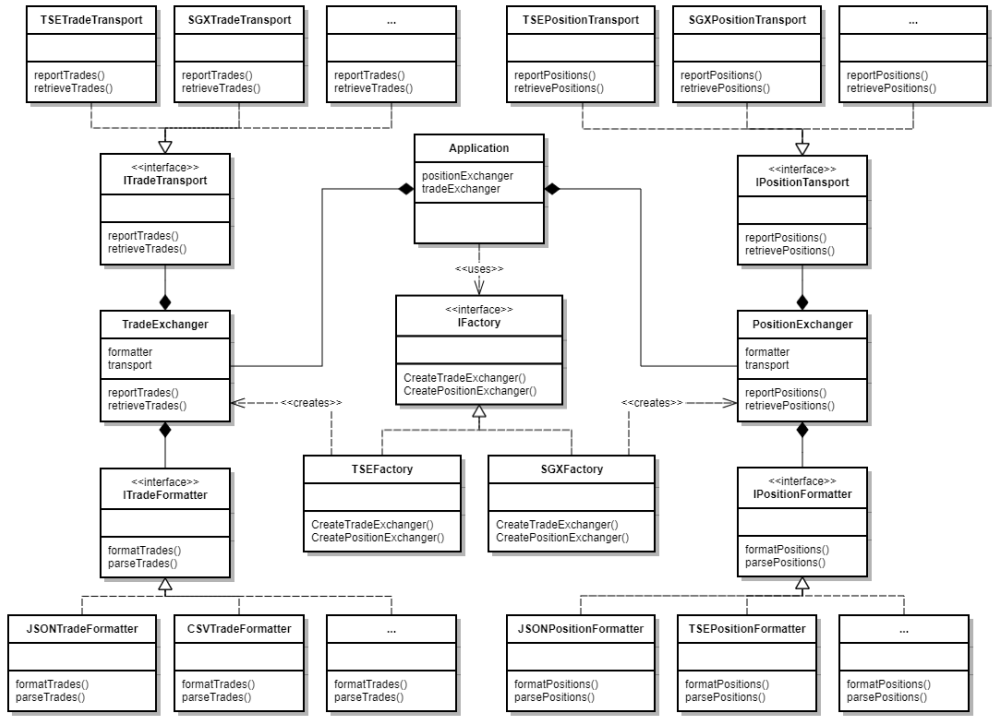
- We've built a world-class trading system, mostly in C++
- *Information* flows to us from an exchange.
- Our auto-traders estimate prices and determine if we wish to *execute* any order operations – that is: place, amend and/or delete orders.
- If so, those order operations are sent to the exchange.
- Rinse and repeat!
- The system is composed of numerous microservices which can be broadly grouped into three categories:
  1. Pricing
  2. Execution
  3. Risk Management



How we use C++ at Optiver

Object-Oriented Programming

- The features of C++ we use most are, of course, those that support object-oriented programming:
  - Classes
  - Encapsulation
  - Inheritance
  - Polymorphism
- We make heavy use of function overloading, and some use of operator overloading (e.g. the >> operator in logging)
- Object-oriented programming is considered critically important at Optiver and is specifically tested in our interview process



STL Containers

- Within our classes we make heavy use of STL containers such as std::vector
- The *latency* of our auto-traders is critically important and therefore we must choose containers very carefully

Let's look at an example:

- An important kind of information we deal with at Optiver is called an *order book*
- It describes the prices at which traders are willing to buy or sell and the volume they're willing to trade

Buy Orders	Price	Sell Orders
	$122.60	21
	$122.59	4
	$122.58	27
	$122.57	
	$122.56	10
	$122.55	3
15	$122.54	
10	$122.53	
20	$122.52	
8	$122.51	
27	$122.50	

An OrderBook Class

```
class OrderBook {
public:
        …

        unsigned long get_volume(const std::string_view price_level) {
                auto it = mVolumes.find({price_level.data(), prive_level.size()});
                if (it != mVolumes.end()) {
                        return it->second;
                }
                return 0;
        }

private:
        std::map<std::string, unsigned long> mVolumes;
};
```

std::map – A Red-Black Tree

Use the Appropriate Kind of Map

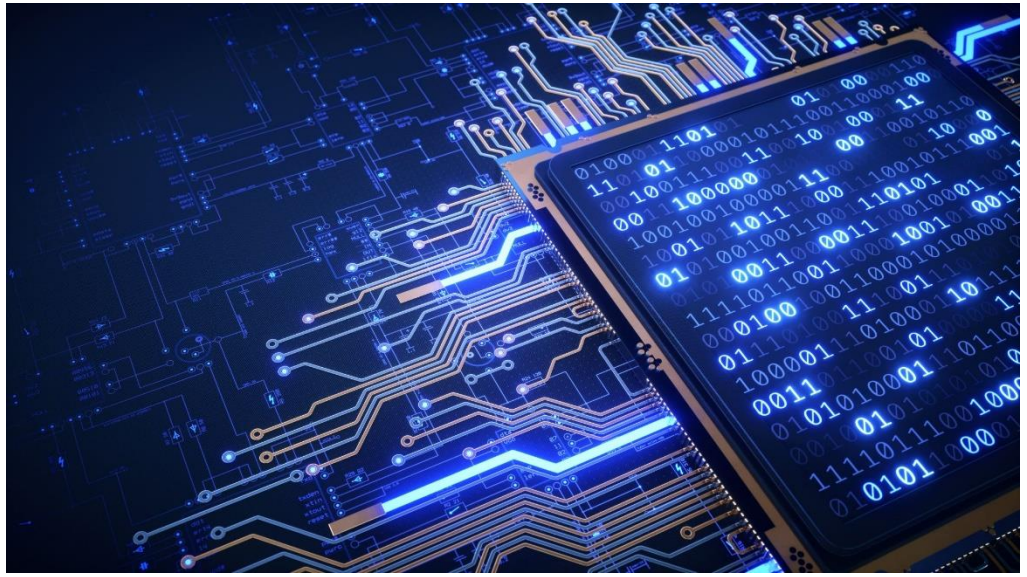
```
class OrderBook {
public:
        …

        unsigned long get_volume(const std::string_view price_level) {
                auto it = mVolumes.find({price_level.data(), prive_level.size()});
                if (it != mVolumes.end()) {
                        return it->second;
                }
                return 0;
        }

private:
        std::unordered_map<std::string, unsigned long> mVolumes;  // <-- O(1) lookup
};
```

Be Aware of Hidden Costs

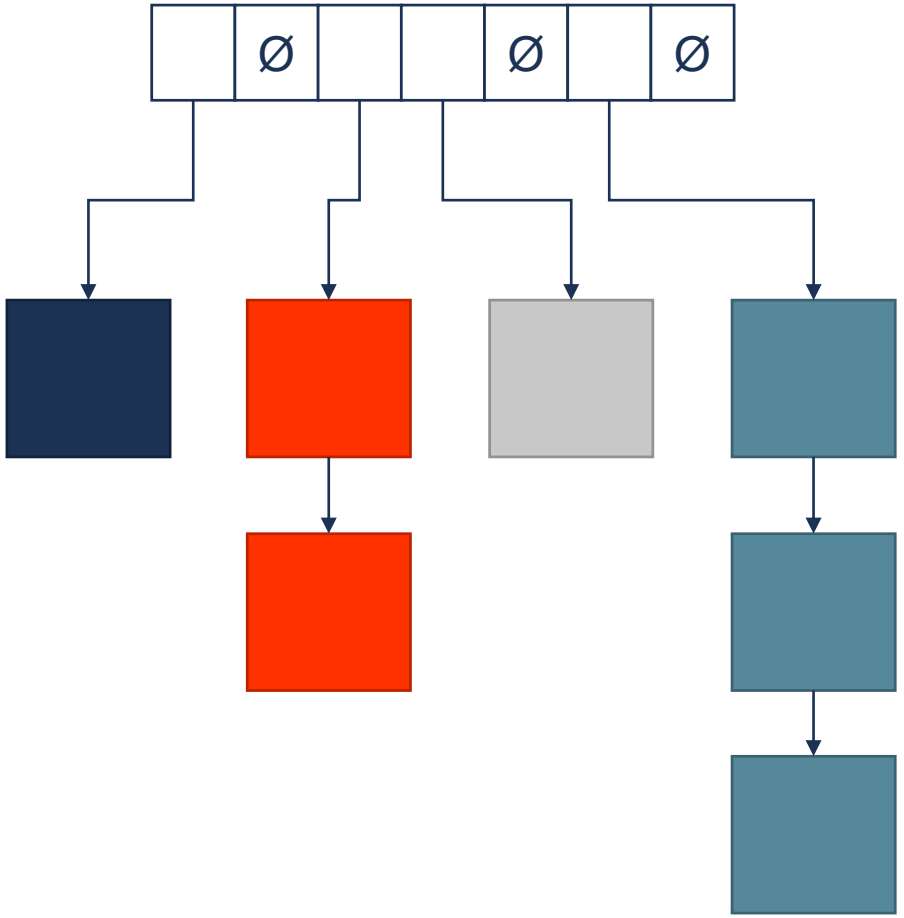
```
class OrderBook {
public:
        …

        unsigned long get_volume(const std::string_view price_level) {
                auto it = mVolumes.find(price_level);     // <- No allocations
                if (it != mVolumes.end()) {
                        return it->second;
                }
                return 0;
        }

private:
        std::vector<std::string> mPrices;
        std::unordered_map<std::string_view, unsigned long> mVolumes;
};
```

CPU Cache

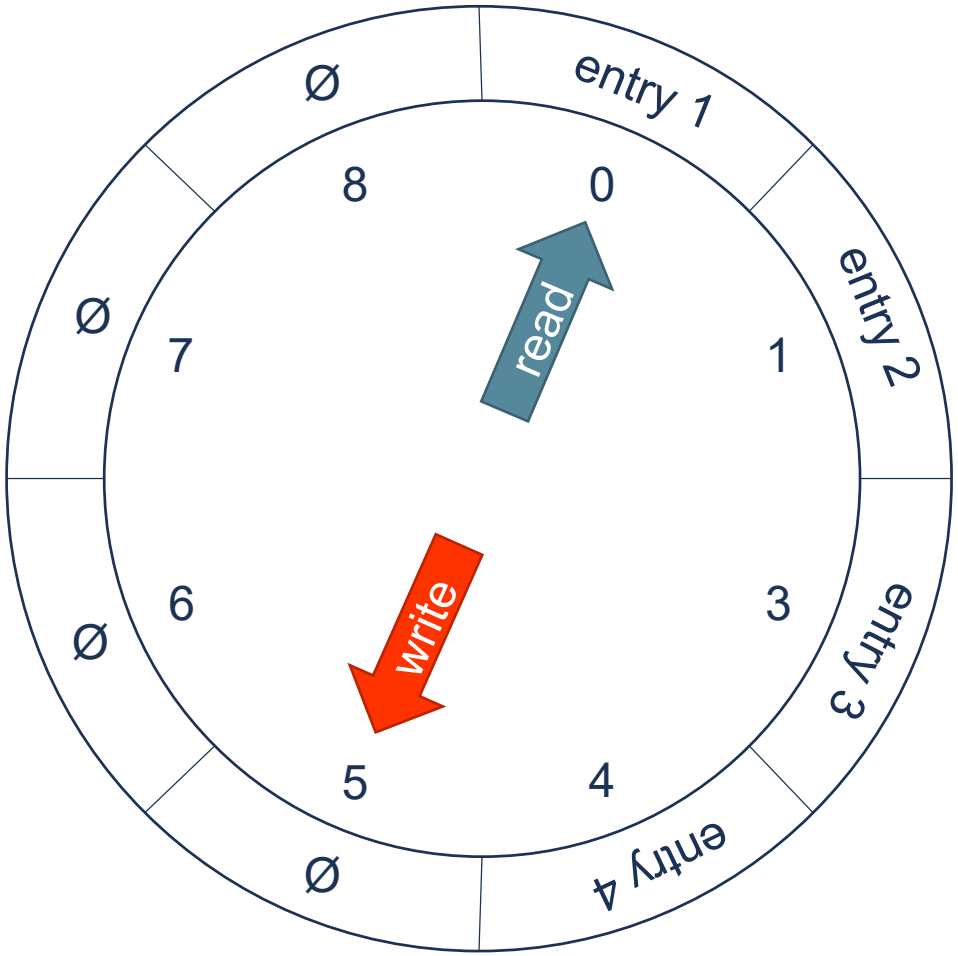
- Modern CPUs employ caches to mitigate the cost of transferring data to/from memory.
- Data is transferred in blocks called cache lines (typically 64-bytes)
- Data structures in which all the data is located close together in memory are faster to access



std::unordered_map – A Hash Table



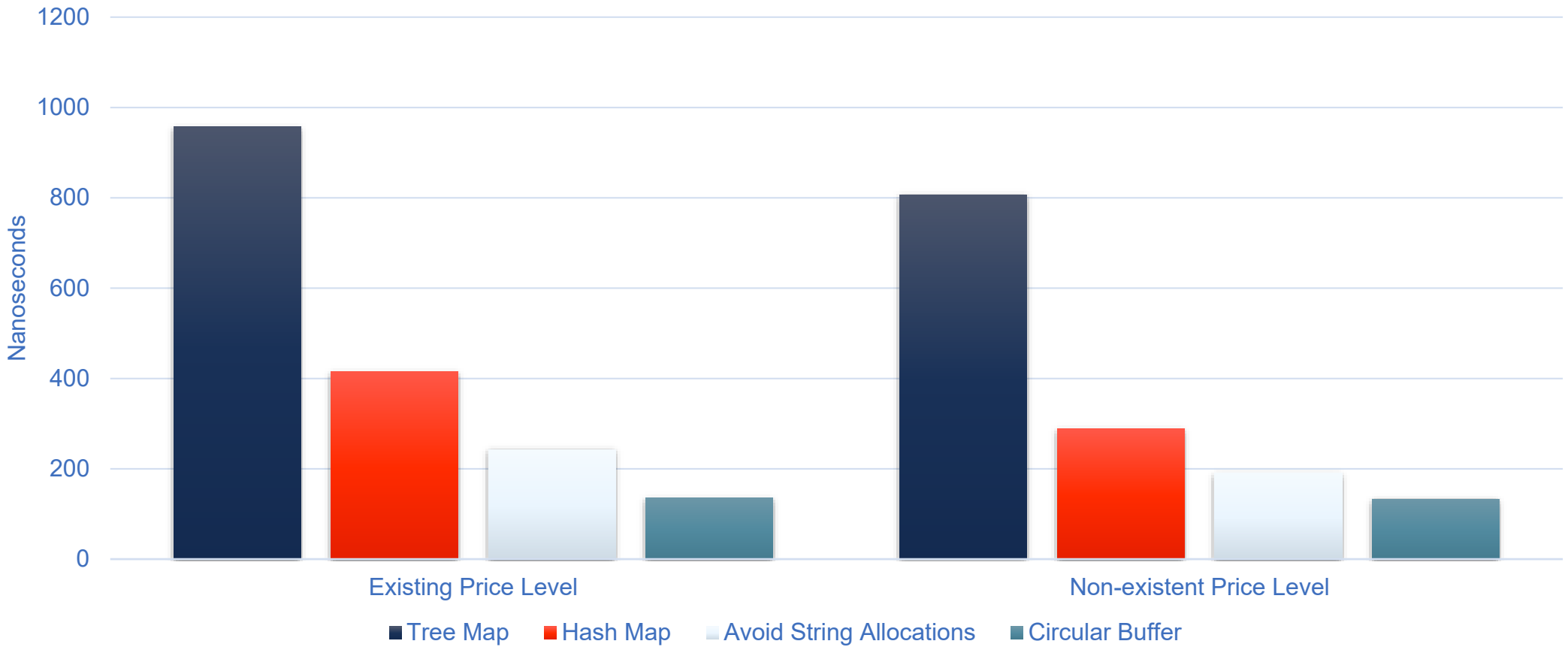
Circular Buffer



Take Advantage of the CPU Cache

```
class OrderBook {
public:
        …

        unsigned long get_volume(const std::string_view price_level) {
                auto price = (unsigned long)(std::strtod(price_level.data(),
                                                         nullptr)*100);
                return mBuffer[price % mBuffer.size()];
        }

private:
        std::vector<unsigned long> mBuffer;
};
```

Benchmark Results of Order Book Variants



STL Algorithms

- We also make heavy use of STL algorithms such as std::find

To give a more complex example:

- One type of trading we do is called *base trading*
- We base our estimate of the fair price of one stock on the price of another (called the *base)*
- Calculating fair prices is time consuming, so we pre-compute them to save time, but the range of possible base prices is very large so we can't feasibly pre-compute them all
- Instead, we pre-compute fair prices for a small number of equidistant base prices that are near the current base price
- How can we quickly get a *theo* given a *base price*?

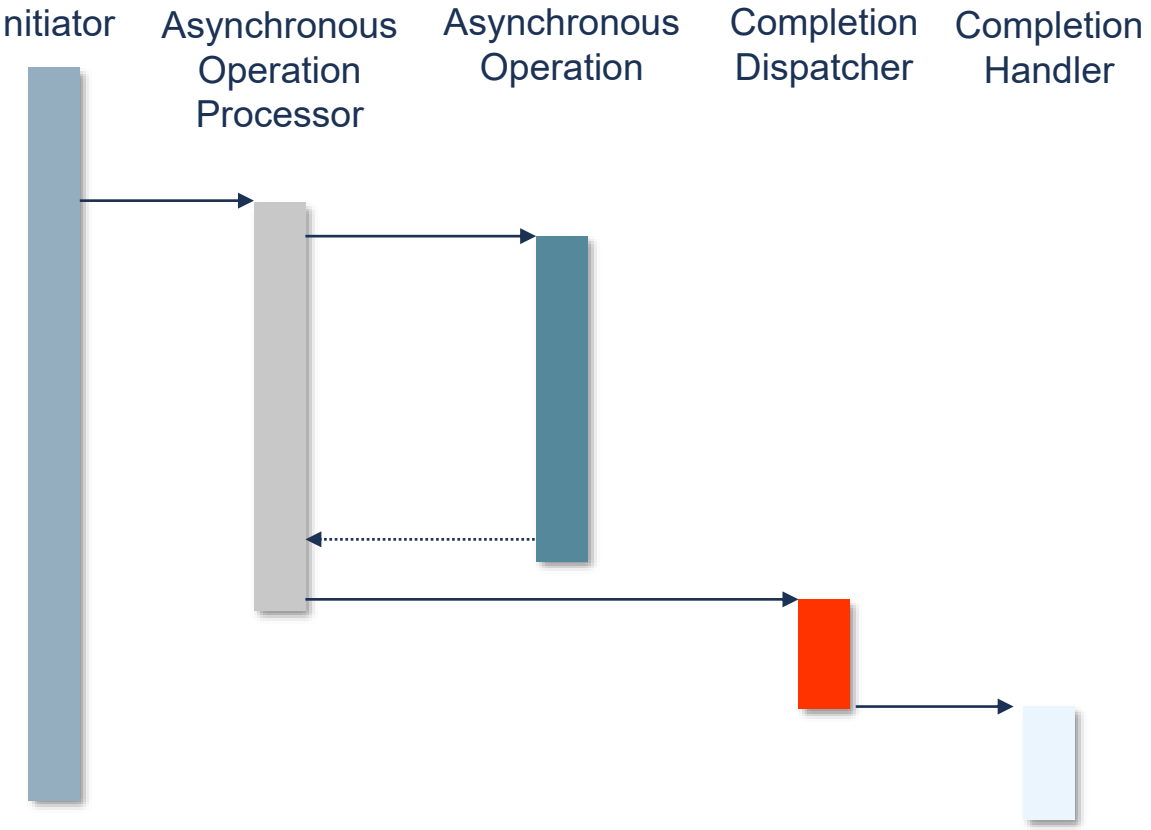
Base Prices				
$42.25	$42.30	$42.35	$42.40	$42.45

Theoretical Prices				
$63.80	63.84	$63.90	$63.96	$64.01

Lambda Functions

- One common way we use lambda functions is in the *Proactor Pattern*
- The *initiator* starts an asynchronous operation and defines a *completion handler* (often a lambda)
- The *completion handler* is called at the end of the operation
- One good example is when we wish to update a database:
  - A message is sent to the database requesting the update
  - A lambda function is defined which is called upon notification that the database update has completed successfully

The Proactor Pattern:



How we use C++ at Optiver

Other C++ Features Used Frequently

- Smart pointers, especially std::unique_ptr
  - Memory allocation can be time consuming so we tend to do most allocation at startup

- template metaprogramming
  - e.g., to define a database ORM

- auto

Optiver

C++20 Features in Depth

Modules

- Standardised mechanism for code reuse
- Organise C++ code into logical components

- A module explicitly exports the classes and functions that code outside of the module are allowed to access.

- Other code remains private
- This behaviour will be extended to all c++ library header files (pending compiler support)

```
// helloworld.cppm
export module helloworld;  // module declaration
import <iostream>;         // import declaration

export void hello() {      // export declaration
    std::cout << "Hello world!\n";
}

void goodbye() {
    std::cout << "Goodbye world!\n";
}

//main.cpp
import helloworld;  // import declaration

int main() {
    hello();
    goodbye(); //error not exported
}
```

Modules

- Reduces dependency on the pre-processor and header files
- #include file order no longer matters, less error prone, no cyclic dependencies
- Avoids issues with macro leaking
- Faster build times, modules are pre-compiled only once

```
// helloworld.cppm
export module helloworld;  // module declaration
import <iostream>;         // import declaration

export void hello() {      // export declaration
    std::cout << "Hello world!\n";
}

void goodbye() {
    std::cout << "Goodbye world!\n";
}

//main.cpp
import helloworld;  // import declaration

int main() {
    hello();
    goodbye(); //error not exported
}
```

Coroutines

- A coroutine is a function that can suspend execution to be resumed later
- Control is returned to the caller
- The current state of the coroutine is saved to be resumed where it left off

- Keywords co_yield, co_await, co_return

```
generator<int> iota(int n = 0)
{
    while (true)
    {
        co_yield n++;
    }
}

void printNumbers()
{
    for (const int i : iota())
        std::cout << i << std::endl;
}
```

Coroutines

- Stackless - Coroutine invocations do not have independent stacks, they allocate data for the coroutine on the heap – efficient memory usage and context switching
- Allow for sequential code that executes asynchronously
- No callbacks, can yield control and resume when necessary

However:

- C++ 20 only provides a very low-level api, the generator class used here is from C++23
- Rules of interaction between the caller and the callee are complex
- Can use some third party libraries (e.g. cppcoro)

```
generator<int> iota(int n = 0)
{
    while (true)
    {
        co_yield n++;
    }
}

void printNumbers()
{
    for (const int i : iota())
        std::cout << i << std::endl;
}
```

Coroutines

- At Optiver, we make heavy use of callbacks to handle events
- Coroutines allow us to declare business logic in-line rather than split across multiple callbacks
- Coroutines have application defined context switch points, which gives us concurrency with a single thread

```
task<> tcp_echo_server() {
    char data[1024];
    while (true)
    {
      std::size_t n = co_await
          socket.async_read_some(buffer(data));
      co_await async_write(
          socket, buffer(data, n));
    }
}
```

Concepts

- Concepts allow us to specify what is needed from a template argument so this can be checked by the compiler
- Constraints model semantic requirements

- In this example the parameter *T* is unconstrained, but it won't compile for any type that doesn't have a + operator
- These error messages can be very complex

```
template <typename T>
auto add(T const a, T const b)
{
    return a + b;
}

int main()
{
    std::cout << add(1, 3) << std::endl;
}
```

Concepts

- Add a *requires* clause

```
template <typename T>
requires std::integral<T>
auto add(T const a, T const b)
{
    return a + b;
}

int main()
{
    std::cout << add(1, 3) << std::endl;

}
```

Concepts

- Creating our own concept

- Generates meaningful error messages that are much easier to understand
- Clearly documents expectations
- Much easier to use than previous **enable_if** syntax

```
template <typename T>
concept Number = std::integral<T> || std::floating_point<T>;

template <typename T, typename U>
requires Number<T> && Number<U>
auto add(T const a, U const b)
{
    return a + b;
}

int main()
{
    std::cout << add(1, 3.1) << std::endl;
}
```

Ranges

- Ranges are an abstraction of a "collection of items" or "something iterable"
- Containers are ranges, they own their elements
- Views are ranges that that are usually defined on another range
- Views do not own any data beyond their algorithm

- Less error prone than using iterators

```
std::vector v;
std::sort(v.begin(), v.end());
std::ranges::sort(v);
```

Ranges

- Allow us to lazily filter and transform data through a pipeline
- Views are applied when an element is requested, not when the view is created

```cpp
#include <iostream>
#include <ranges>
#include <vector>

int main() {
  std::vector<int> numbers = { 1, 2, 3, 4, 5, 6 };

  auto is_even = [](int n) { return n % 2 == 0; };

  auto results = numbers | std::views::filter(is_even)
    | std::views::transform([](int n) { return n++; })
    | std::views::reverse;

  for (auto v: results) {
    std::cout << v << " ";   // Output: 7 3 5
  }
}
```

Optiver

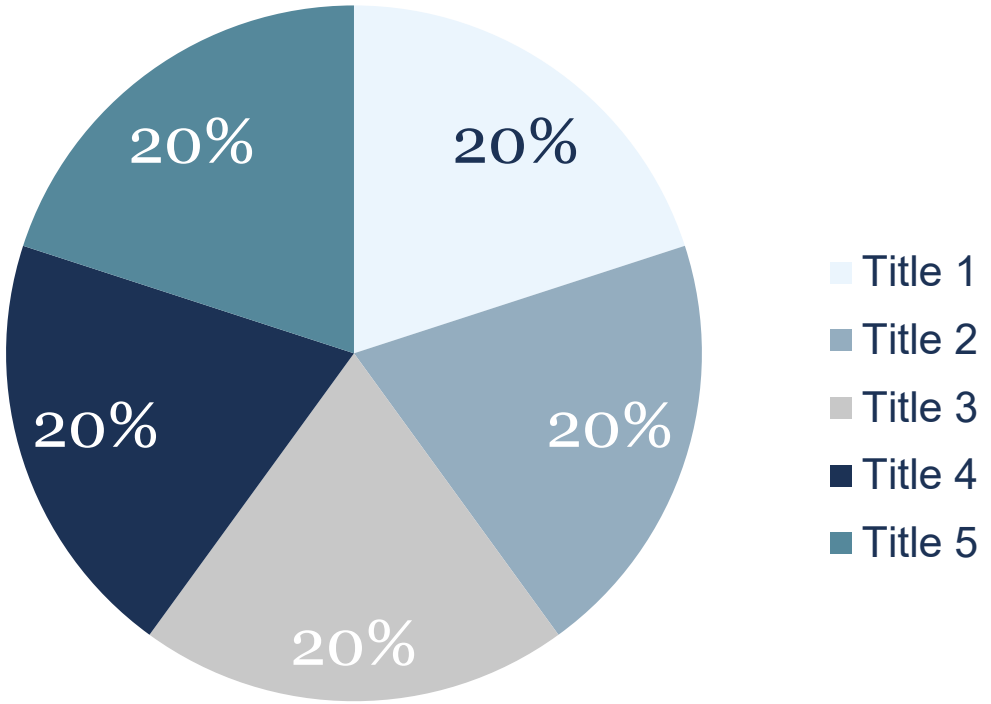
Questions?

Greg Saunders
Head of IT Education

Greg.Saunders@optiver.com.au

www.optiver.com/working-at-optiver


Amsterdam
London
Chicago
Austin
Shanghai
Taipei
Hong Kong
Singapore
Sydney


20%
20%
20%
20%
20%
Title 1
Title 2
Title 3
Title 4
Title 5

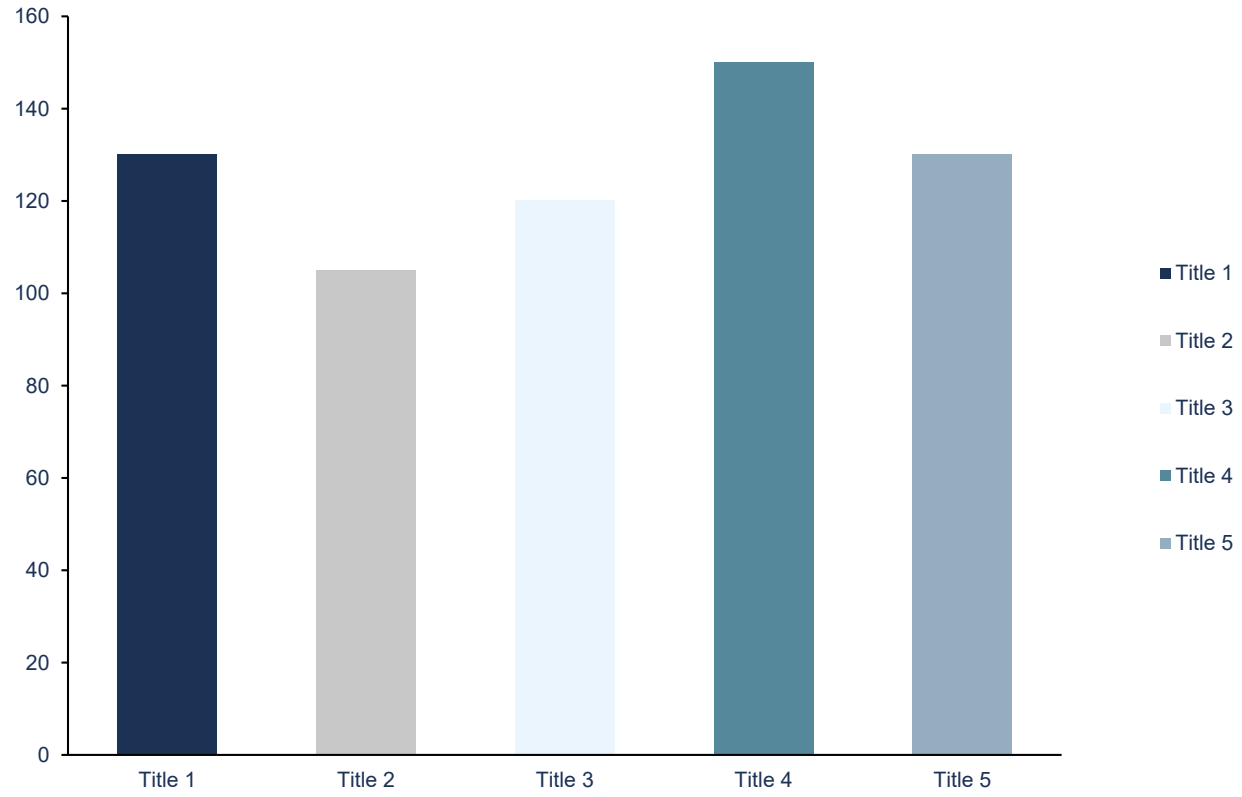

160
140
120
100
80
60
40
20
0
Title 1
Title 2
Title 3
Title 4
Title 5
Title 1
Title 2
Title 3
Title 4
Title 5

Table header first column in italic (manually)	Header row	(right aligned)	Last column
Subtotal bold and lightblue hatching must be set manually			
Total row	**Total row**	**Total row**	**Total row**

Table header first column in italic (manually)	Header row	(right aligned)	Last column
Subtotal bold and lightblue hatching must be set manually			
Total row	**Total row**	**Total row**	**Total row**

FOOTNOTES
Text

Optiver

Do you have any questions?

Name Surname
Job title

namesurname@optiver.com
+31 (0) 625755624
Strawinskylaan 3095
1077 ZX Amsterdam