CS310 Project Specification

5508367 - Louis Tanak

2025-10-13

Title

Evaluating the performance of lock-free data structures under realistic exchange workloads.

Abstract

Lock-free data structures allow multiple threads to operate safely on shared data without using locks. These have many real-word use cases in high-performance computing, low-latency systems and financial exchanges. Research has been conducted on their theoretical properties, however there is a gap in understanding how lock-free data structures perform under realistic workloads, such as trading exchange-like patterns.

The work aims to implement various lock-free data structures and evaluate their performance against each other and traditional lock-based solutions. The data structures will be benchmarked against simulated exchange traffic to measure throughput, latency, scalability and completion ordering. Results will provide insights into the benefits, drawbacks and the suitable conditions for each data structure.

1 Problem Statement

In 2007, Michael Maged presented the world's first lock-free memory allocator at the PLDI conference (Programming Language Design and Implementation)[3]. Since then, lock-free data structures have been widely used in a range of applications, across multiple industries.

Previous research has been conducted on the theoretical properties of lock-free data structures, and various studies have analysed the performance of the structures under different load testing, such as increasing thread count or increasing the critical work that each thread does.[1][5]

However, there is limited publicly available work investigating the performance of lock-free data structures with domain-specific implementations under realistic workloads. More specifically, little work has examined their performance under bursty and inconsistent flows, similar to those found in financial exchanges. This provides a gap in the research that this project plans to investigate.

2 Objectives

The primary objectives have been outlined below, separating into the essential and extended work.

2.1 Essential Work

The following tasks are essential work that should be completed to consider the project a success. These provide the foundation for the analysis, benchmarking and experimentation of the project.

1. Design and implement code to simulate trade orders such as buy, sell and cancel orders. These are common in a financial exchange and would establish a realistic workload.

2. Implement different implementations of lock-free data structures. Each data structure will be compared to their locked data structure equivalent, to provide a baseline comparison. Sample data structures include a regular queue, a lock-free queue, and an MPMC queue data structure[4].

3. Design and implement a benchmarking framework that will capture the throughput, latency and the *tail latency* of the data structure being tested.

4. Automate the execution and collection of data from benchmarking by creating configurable test scripts. Automating the tests reduces the probability of human error and provides an environment for controlled testing.

2.2 Core Work

The following is work that should be completed to make the overall project's research value stronger.

5. Design and implement a *price-time priority exchange* to simulate the matching behaviour between orders. It will enable the measurement of additional metrics such as order completion time and fairness, bridging the gap between performance and real-world system behaviour.

6. Introduce stochastic behaviour into the order generation model, such as poisson distributions and simulated *market events*, to create more realistic trading flows to further test data structures.

7. Implement variants of lock-free ring buffers. These are structures used actively in exchanges around the world, making the overall project more realistic and offering an insight into their trade-offs.

8. Investigate methods to ensure correct ordering of transactions and the effect of lock-free data structures on ordering. The benchmarking library will provide information such as the amount of out-of-order pairs and the *spearman correlation* between different data structures.

9. Benchmark hardware-level metrics such as CPU utilisation, cache performance and thread efficiency. Hardware-level metrics provide a deeper understanding of where the differences amongst data structures arise.

2.3 Extended Work

Given enough time, this is work that could be completed, however it is not necessary for its completion.

10. Optimise the data structures for *NUMA architectures*. This would be achieved by binding threads to the same NUMA node, which reduces latency as cores access local memory rather than going through shared interconnection channels. NUMA-aware designs reflect real high-performance exchanges and would lead to substantial latency improvements[2].

11. Optimise each data structure by restructuring memory layout, by using padding and other alignment techniques. The result of this would improve the throughput of the data structure.

12. Design and implement new variants of lock-free queues or ring buffers.

13. Implement one or more data structures in another programming language such as `Go` or `Rust`. This would provide insight into the different memory models of different languages and allow for cross-language data structure comparison.

3 Methods & Methodology

3.1 Pre-reading & Project Setup

At the beginning of this phase, some time will be spent reading `A tour of C++`[6] to understand the syntax and the best practices of the C++ language.

Alongside the reading, to ensure the project is modular and extensible, code will be implemented to create an interface-based architecture, to allow for the different modules to communicate with each other. This will be done utilising templates in `C++`.

Templates allow a developer to write generic code that works with multiple data type or implementation. Another approach that has been considered is utilising *virtual interfaces*, however virtual interfaces are determined at runtime, which would lead to a higher latency. Therefore, templates will be used as they are determined at compile-time and they provide additional type-safety.

3.2 Order Simulation

To simulate transactions, research will be conducted alongside development investigating different methods for generating realistic order flow. This will involve reviewing existing models that simulate markets, including how to accurately simulate market events such as bursty-traffic or large volume transactions occurring.

Once this research has been completed, the logic for this will be completed in `C++`. Multiple producer threads will be creating orders in different patterns, such as following poisson distributions, pseudo-random orders and other possible strategies / distributions found during the research phase.

The orders will be connected directly to the lock-free data structures, for example pushing directly onto the lock-free queue. This provides a direct medium of access to the data structure. Another approach considered was a networked approach, however this creates an additional overhead of variable transmission delays, potential packet loss and would make it more difficult to isolate and measure the true performance of the lock-free data structure being benchmarked.

3.3 Implementation of Lock-free Data Structures

Using the template architecture aforementioned, each individual data structure will be implemented as its own class and conform to the same interface pattern using the template. All the structures implemented will expose the same function signatures to allow them to be used interchangeably as template parameters within the ecosystem of the code. Each data structure acts as a drop-in replacement for one another because they all adhere to the common interface, simplifying the benchmarking process to compare the different implementations.

3.4 Evaluation Methodology

To evaluate the results produced, we will adopt a scientific experimental approach. Benchmarks will be repeated multiple times under identical conditions, and results will be analysed using statistical methods to assess their variability. Following a strict control over the tests will form a more comprehensive analysis, accurately reflecting the behaviour of each data structure being modelled.

3.5 Software Development Methodology

The project will use an agile methodology, to incrementally build up the solution with each iteration. Early iterations will focus on integrating core components of the system, such as implementing a simple order creation model, working with a regular queue implementation and measuring a small amount of metrics with the benchmarking library. Later iterations will result in more comprehensive products, such as more realistic order creations, different variations of the lock-free data structures and potentially passing through a benchmarking library and a simulated exchange simultaneously.

An agile methodology for this project is the most relevant, as it aligns with the experimental nature of evaluating the lock-free data structures. It provides flexibility which will help manage the project within the time constraints, and adjust objectives based on the current stage of development as required.

The code will be regularly tested through a combination of unit, component and *integration tests*. Writing tests ensures that each section of the project functions as expected, whether that is in isolation or as a wider part of the system. The tests will maintain the reliability of the code and promote well-structured and maintainable code.

4 Timetable

The timetable is depicted as a Gantt chart for the proposed project. It includes information regarding project deliverables & deadlines, internal deadlines for code completion or completed components, and accommodates for commitments external to the project.

The scope of the project may change due to strong progress or setbacks. The aim for the first term is to create a minimum viable product, to provide a foundation for implementing and benchmarking the lock-free data structures. The aim for the winter holidays and the second term is to build upon the foundational work and achieve all the core objectives.

The timetable can be found at the end of this project specification.

5 Resources & Risks

5.1 Resources

Technical implementations of the project will be implemented in `C++ 17/20/23`. `C++` is the most suitable language for the project as it natively provides features that would be useful during development. The language includes access to atomic operations, the ability to tune cache-alignment, memory control and more. The majority of real systems are written in the C++ language, therefore it makes it the most suitable language to use, in comparison to other object-oriented programming languages.

Other notable software and technologies that could be used are:

- **Git** - a distributed version control software system that is capable of managing versions of source code or data
- **CMake** - a cross-platform build system generator. It provides a way to unify builds on different platforms such as Windows, Linux and Mac OS
- **perf** - linux tool for performance analysis. Supports subcommands such as hardware performance counters, tracepoints, monitoring events, etc
- **Intel VTune** - profiler which collects performance statistics for serial and multithreaded code
- **Valgrind** - framework that includes various features to help with memory & thread management
- **Python** - high-level programming language with library implementations useful for graphing data
- **Google Benchmark** - library to benchmark code snippets in C++
- **Docker** - containerisation software that packages code with all its necessary dependencies and configuration

These resources may change over the duration of the project.

5.2 Risks

Below are the risks and mitigations associated with the project.

Risk	Mitigation
Personal laptop getting lost or broken	Utilise a cloud-based platform for hosting code such as `GitHub`, and regularly commit code changes. Store relevant documents and research papers in cloud storage. Utilise lab computers in the department of computer science whilst waiting for a replacement.
Unexpected illness or personal circumstances	Contingency time has been reflected in the Timetable.
Data loss or corruption	Regularly backup data such as test logs / benchmarks to cloud storage such as Google Drive. Additionally, store data on an external HDD or SSD.
Time management or overambitious objectives	Regularly produce deliverables and update objectives based on current progress. Discuss with supervisor any proposed changes to the scope of the project.

Table 1: Project risks and their mitigation strategy

6 Legal, Social, Ethical and Professional Issues & Considerations

One professional consideration is utilising proprietary exchange information. This could cause issues with data licensing or the data may have restrictions with how it can be used. If proprietary information is being used, checks will be made to ensure that the data is used legally and within any restrictions stated.

One ethical consideration would be the presence of *personally identifiable information* (PII) in datasets. This raises issues around privacy, consent, and data protection. To mitigate against this, datasets and the corresponding dataset provider will be thoroughly checked to ensure we do not use any dataset containing PII.

There are no further legal, social, ethical or professional concerns with this project.

7 Glossary

Tail latency - the performance experienced by the slowest percentile of tests.

Price-time priority exchange - an algorithmic principle in exchanges that matches orders with the most favourable price and in situations of equal price, the order with the earliest arrival timestamp.

Market events - significant occurrences that can cause price volatility and influence market direction severely in a certain direction.

Spearmen correlation - a value ranging from -1 to 1 indicating how strongly correlated two sets of data points are.

NUMA Architectures - a memory architecture where memory access time is different for all processors in a system, therefore processors access their local memory rather than memory in a remote NUMA node.

Virtual interfaces - a class that declares virtual functions, that must be overriden in any derived class.

Integration tests - form of software testing that verifies different modules or components work together when they are combined.

Personally identifiable information (PII) - any data that can be used to identify an individual. The handling of PII is strictly regulated and the mishandling of it can adhere to penalties and fines.

References

[1] Aras Atalar, Paul Renaud-Goud, and Philippas Tsigas. Analyzing the performance of lock-free data structures: A conflict-based model. In Yoram Moses, editor, *Distributed Computing*, pages 341–355, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg. ISBN 978-3-662-48653-5. URL https://link.springer.com/chapter/10.1007/978-3-662-48653-5_23#citeas.

[2] Irina Calciu, Siddhartha Sen, Mahesh Balakrishnan, and Marcos K. Aguilera. Black-box concurrent data structures for numa architectures. In *Proceedings of 22nd ACM International Conference on Architectural Support for*

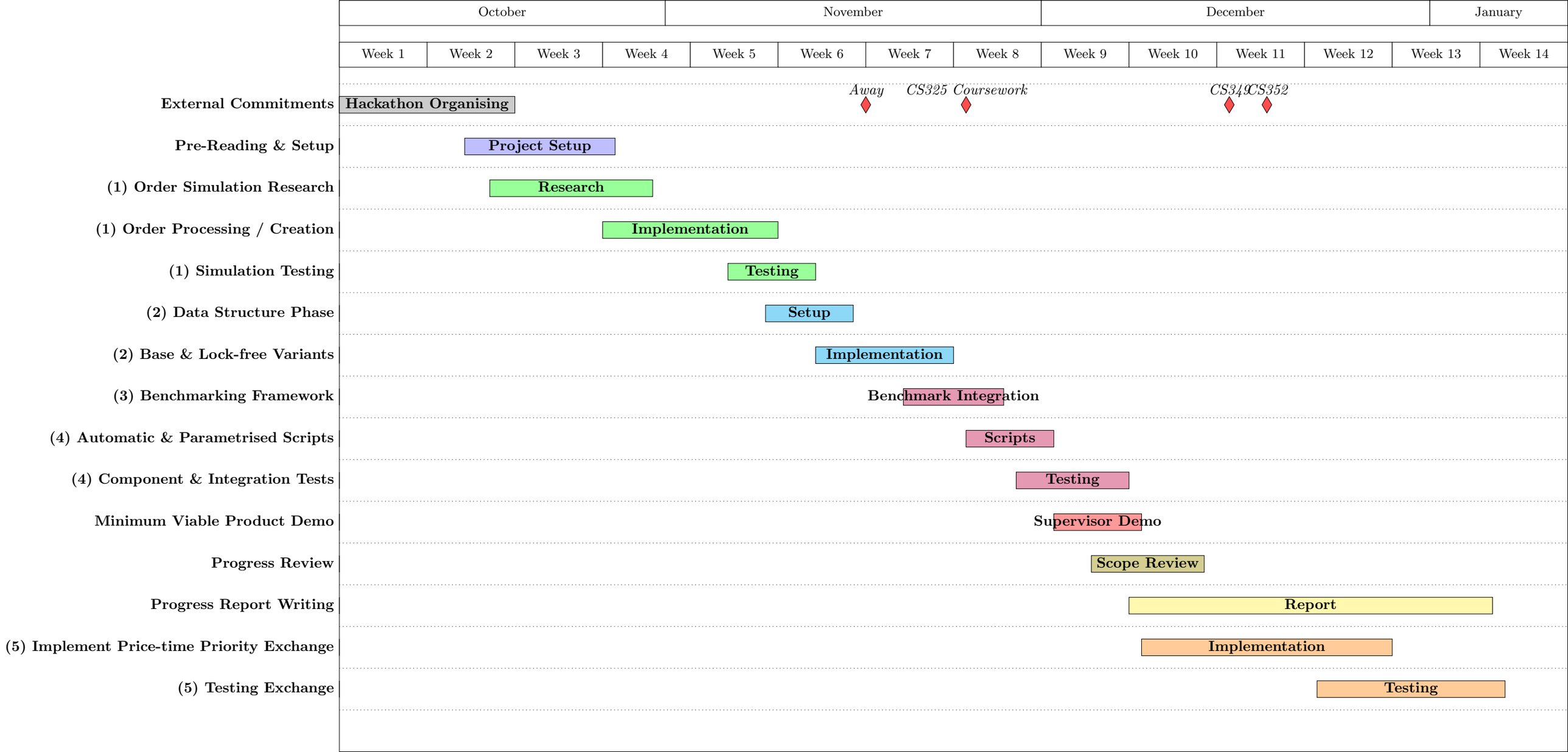
Programming Languages and Operating Systems (ASPLOS), April 2017. URL https://www.microsoft.com/en-us/research/publication/black-box-concurrent-data-structures-numa-architectures-2/. Best Paper Award.

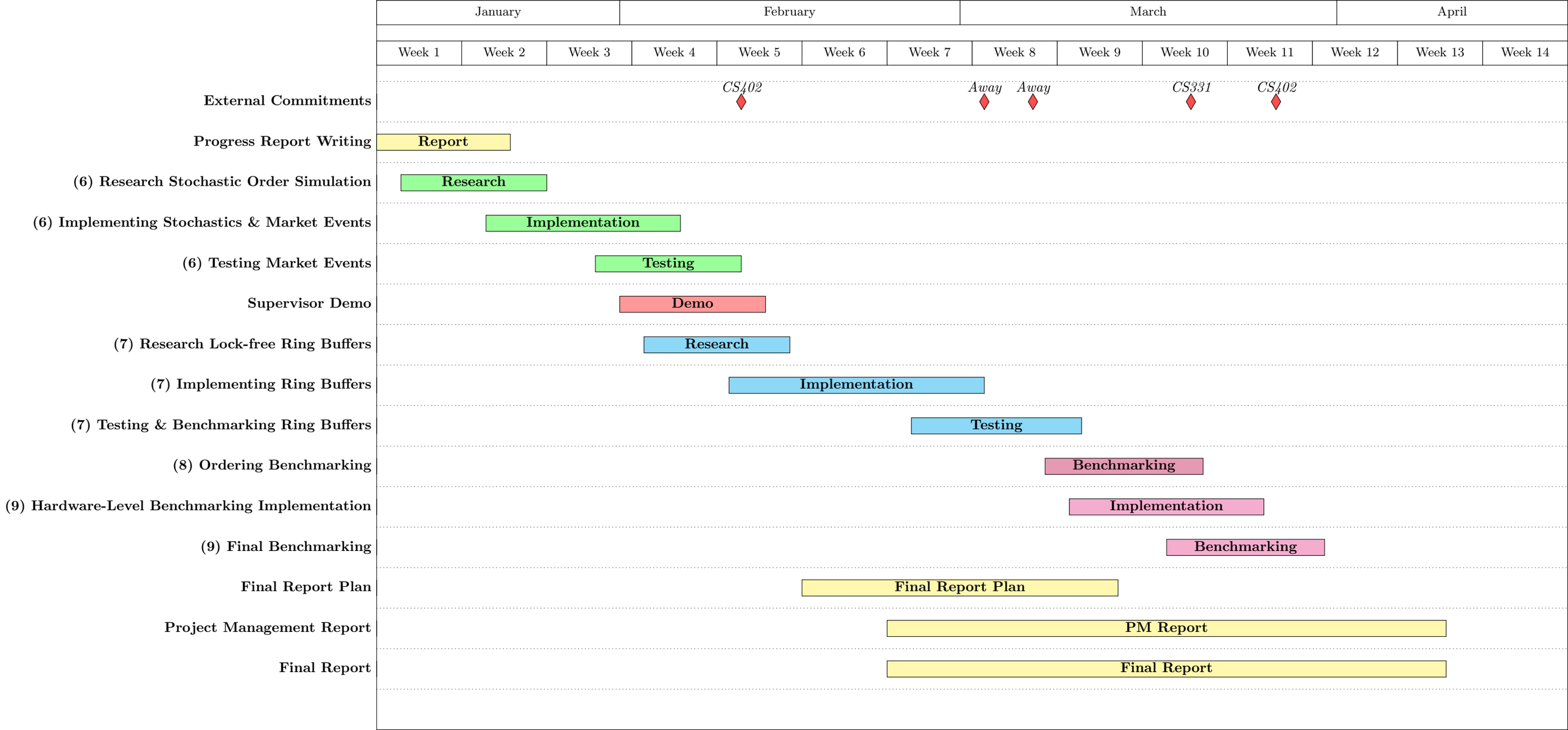
[3] Maged M. Michael. Scalable lock-free dynamic memory allocation. In *Proceedings of the ACM SIGPLAN 2004 Conference on Programming Language Design and Implementation (PLDI '04)*, pages 35–46. ACM Press, 2004. ISBN 1-58113-807-5. doi: 10.1145/996841.996848. URL https://www.cs.tufts.edu/~nr/cs257/archive/neal-glew/mcrt/Non-blocking%20data%20structures/p35-michael.pdf.

[4] Ruslan Nikolaev. A scalable, portable, and memory-efficient lock-free FIFO queue. *CoRR*, abs/1908.04511, 2019. URL http://arxiv.org/abs/1908.04511.

[5] Ruslan Nikolaev. A scalable, portable, and memory-efficient lock-free fifo queue. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi: 10.4230/LIPICS.DISC.2019.28. URL https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.DISC.2019.28.

[6] Bjarne Stroustrup. *A Tour of C++*. Addison-Wesley Professional, Boston, 3rd edition, 2022. ISBN 978-0-13-681648-5.


October
November
December
January
Week 1
Week 2
Week 3
Week 4
Week 5
Week 6
Week 7
Week 8
Week 9
Week 10
Week 11
Week 12
Week 13
Week 14
External Commitments
Pre-Reading & Setup
(1) Order Simulation Research
(1) Order Processing / Creation
(1) Simulation Testing
(2) Data Structure Phase
(2) Base & Lock-free Variants
(3) Benchmarking Framework
(4) Automatic & Parametrised Scripts
(4) Component & Integration Tests
Minimum Viable Product Demo
Progress Review
Progress Report Writing
(5) Implement Price-time Priority Exchange
(5) Testing Exchange
Hackathon Organising
Away
CS325 Coursework
CS349
CS352
Project Setup
Research
Implementation
Testing
Setup
Implementation
Benchmark Integration
Scripts
Testing
Supervisor Demo
Scope Review
Report
Implementation
Testing


January
February
March
April
Week 1
Week 2
Week 3
Week 4
Week 5
Week 6
Week 7
Week 8
Week 9
Week 10
Week 11
Week 12
Week 13
Week 14
External Commitments
CS402
Away
Away
CS331
CS402
Progress Report Writing
Report
(6) Research Stochastic Order Simulation
Research
(6) Implementing Stochastics & Market Events
Implementation
(6) Testing Market Events
Testing
Supervisor Demo
Demo
(7) Research Lock-free Ring Buffers
Research
(7) Implementing Ring Buffers
Implementation
(7) Testing & Benchmarking Ring Buffers
Testing
(8) Ordering Benchmarking
Benchmarking
(9) Hardware-Level Benchmarking Implementation
Implementation
(9) Final Benchmarking
Benchmarking
Final Report Plan
Final Report Plan
Project Management Report
PM Report
Final Report
Final Report