# 

# High-Level Design Document for <project name> project

|  |  |
| --- | --- |
| **Author** | Alon Libling & Tal Gelbard |
| **Version** | V0.1 |
| **Created** | 15/04/20 |
| **Last Updated** | 15/04/20 |

# Introduction – this section specifies what you are actually implementing (usually subset of the functional specification)

## Assumptions and Design Constraints List

We assume the client isn’t malicious and will not try to find vulnerabilities in our Data structure. we also assume the maximum thread count to be used in our DS is 128, but this could be easily expanded.

## Dependencies

Our DS will require very little dependencies in its basic form, which std and pthread, we also require the core architecture will have an atomic CAS implemented, but this should be standard in today modern CPUs.

In the future we might add dependencies related to the hash function we are using and smart pointers (GC) for better overall performance.

## Issues List

Currently there are no open issues, we are only in implementation stage.

## To-do List + Expected time-tables

Todos:

Implementation of the data structure with basic required features until the end of April.

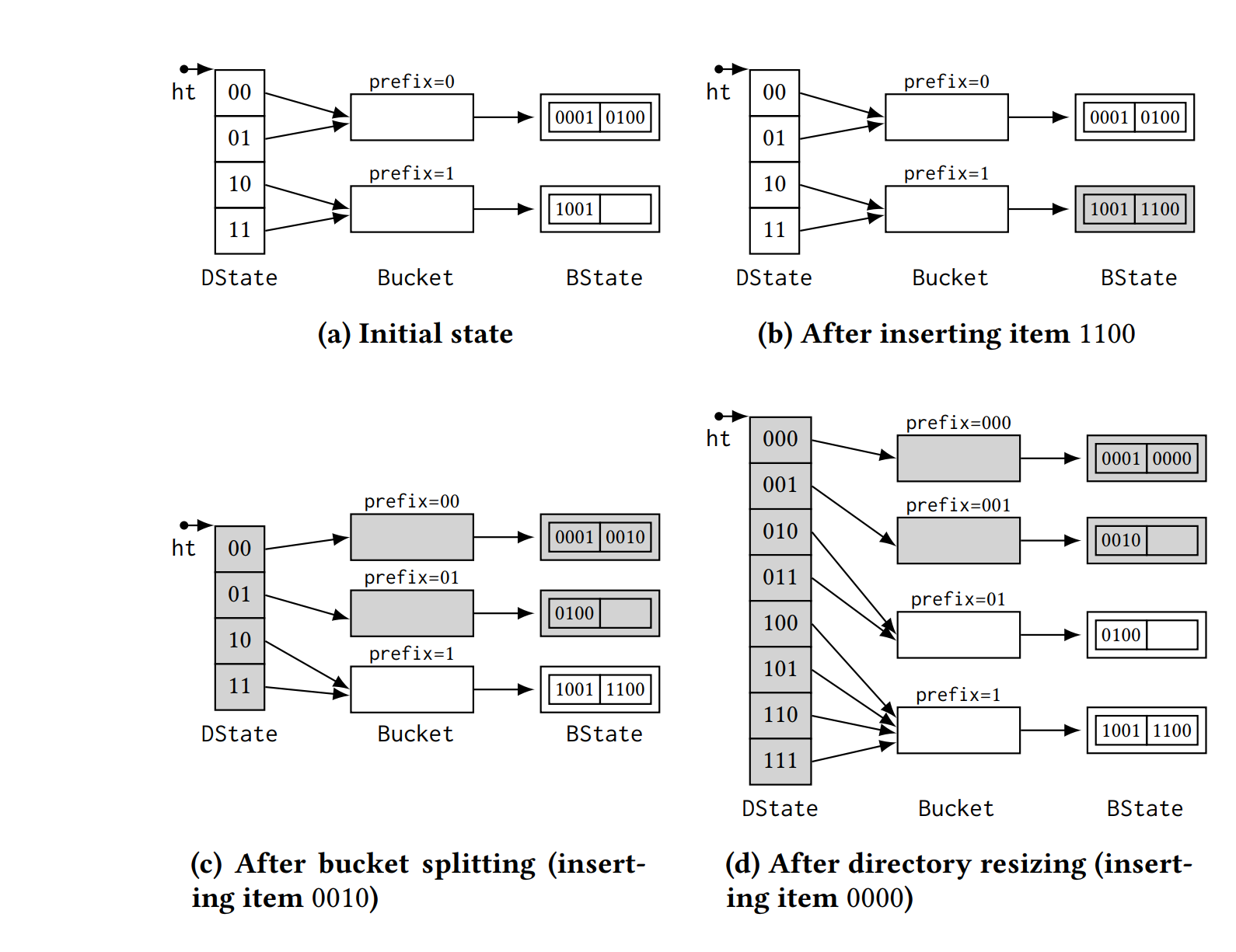
Testing & debugging – end of June.

Extra features & benchmarks – end of July.

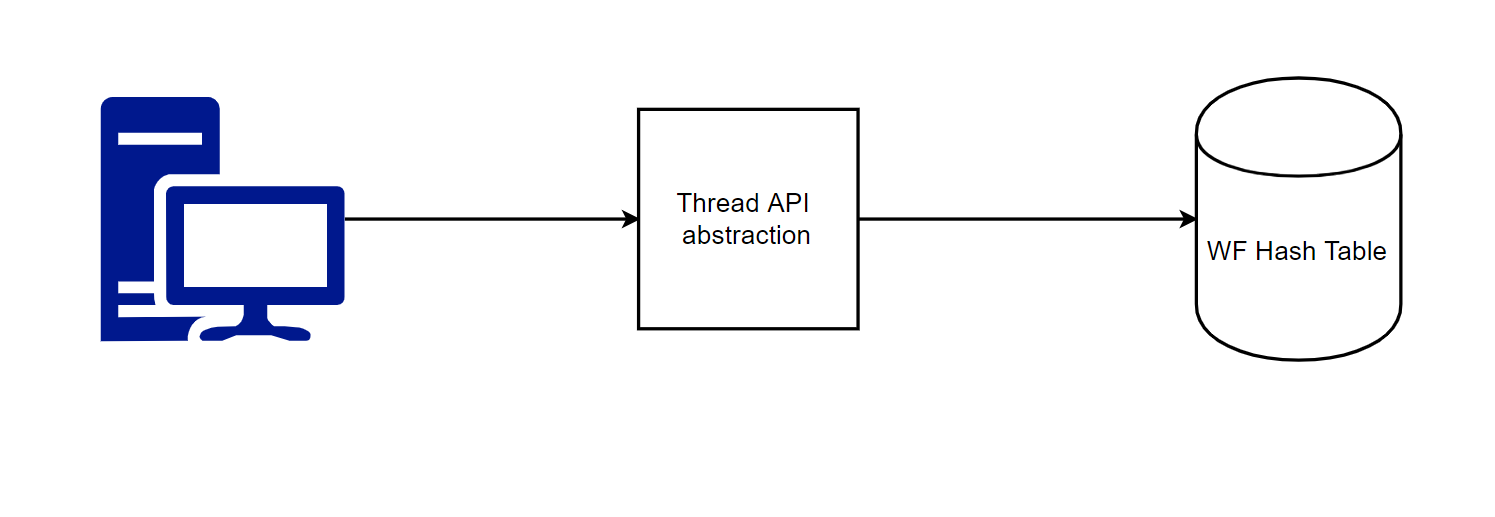
# Logical Architecture

Describe the architecture of your application. Prefer using a diagram like this one.

The paper we base the DS on does a pretty good job at this abstraction:



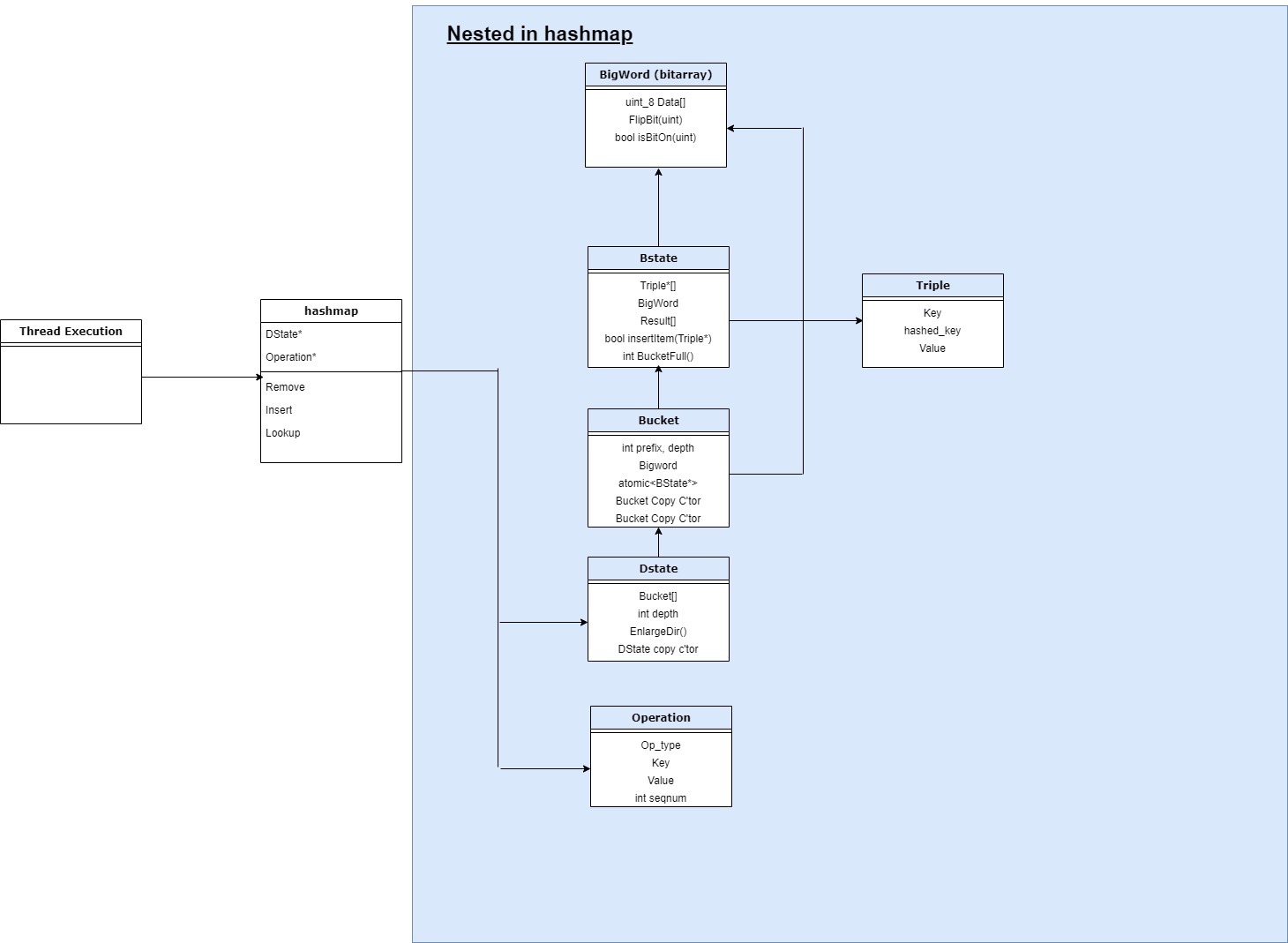
Logical abstraction of the implementation:



# Design

## Classes

### Class Diagram



### Class details

Details about the various classes.

## Setup

Setup should be easily done by including the header of the hashmap or the thread abstraction in the client’s program.

# Use cases

This implementation of a WF hash table surpass any other available implementation by quite a lot in cases that the resize operation doesn’t occur often (). This characteristic is useful to many software applications like those who provide searching as a main feature for example: address searching (Google maps), public transportation searching (Moovit) etc.

# References – to external papers/packages

[An Efficient Wait-free Resizable Hash Table](https://tropars.github.io/downloads/pdf/publications/spaa2018-FKR-WF_ext_hashing.pdf) – Our implementation is mainly based on the DS described in this paper.

[A Highly-Efficient Wait-Free Universal Construction](http://thalis.cs.uoi.gr/tech_reports/publications/TR2011-01.pdf) – The paper above uses the ideas in this paper to implement the hash-table, we use this for gaining deeper understanding of the ideas behind the DS.

[sim-universal-construction](https://github.com/nkallima/sim-universal-construction) – This repo contains many WF/Lockfree implementations of different data structure among WF DS that are based on the P-sim idea we implement in our DS, we use it for ideas on how to design our implementation. And at a later stage of development we will use those DS as benchmarks to compare to our implementation (Contains P-sim based Stack and Queue).