N Way Set Associatve Cache

The solution implements n-way set associative cache using a `ConcurrentDictionary` to store individual Cache Lines (terminology borrowed from CPU L1 set associatve cache implementation).

Performance was the cornerstone of the design and this cache operates at O(1) complexity.

Cache line is itself a Dictionary of the Key provided by the clients and LinkedListNode of CacheLineItem. CacheLineItem is object which stores the Key and the Value of any type supplied by the client.

As the solution requires eviction policies of LRU and MRU, LinkedList is used to store the actual data. Every node is stored in a dictionary against the user supplied Key for O(1) access within the CacheLine.

Every new data is stored at the end of this list. Thus newly inserted objects live at the end and older objects move towards the start of the list. Whenever an object is accessed via key, it is deleted from the current position in the linked list (if not already last) and placed at the end to ensure not only newly created but also newly accessed objects (Get or Update) remain at the end of the list. This way eviction policies of LRU and MRU remain O(1) operations.

As performance is the single most important factor in designing in-memory cache the Add, Update, Get and Remove all remain O(1) operations. This is achieved by using above mentioned data structures.

For multi-threaded access fine tuning locking at Cacheline are used with outer dictionary being ConcurrentDictionary. *Picking up policy for locking is out of scope for this solution*. The locking policy can be built around use cases for read heavy, write heavy or mixed mode of access to this cache in future.

Architecture

N-way set associatve cache implements INWaySetAssociatveCache interface to provide basic cache interface including Add, AddOrUpdate, Get, TryGet, Remove and Count. The interface puts following constraints

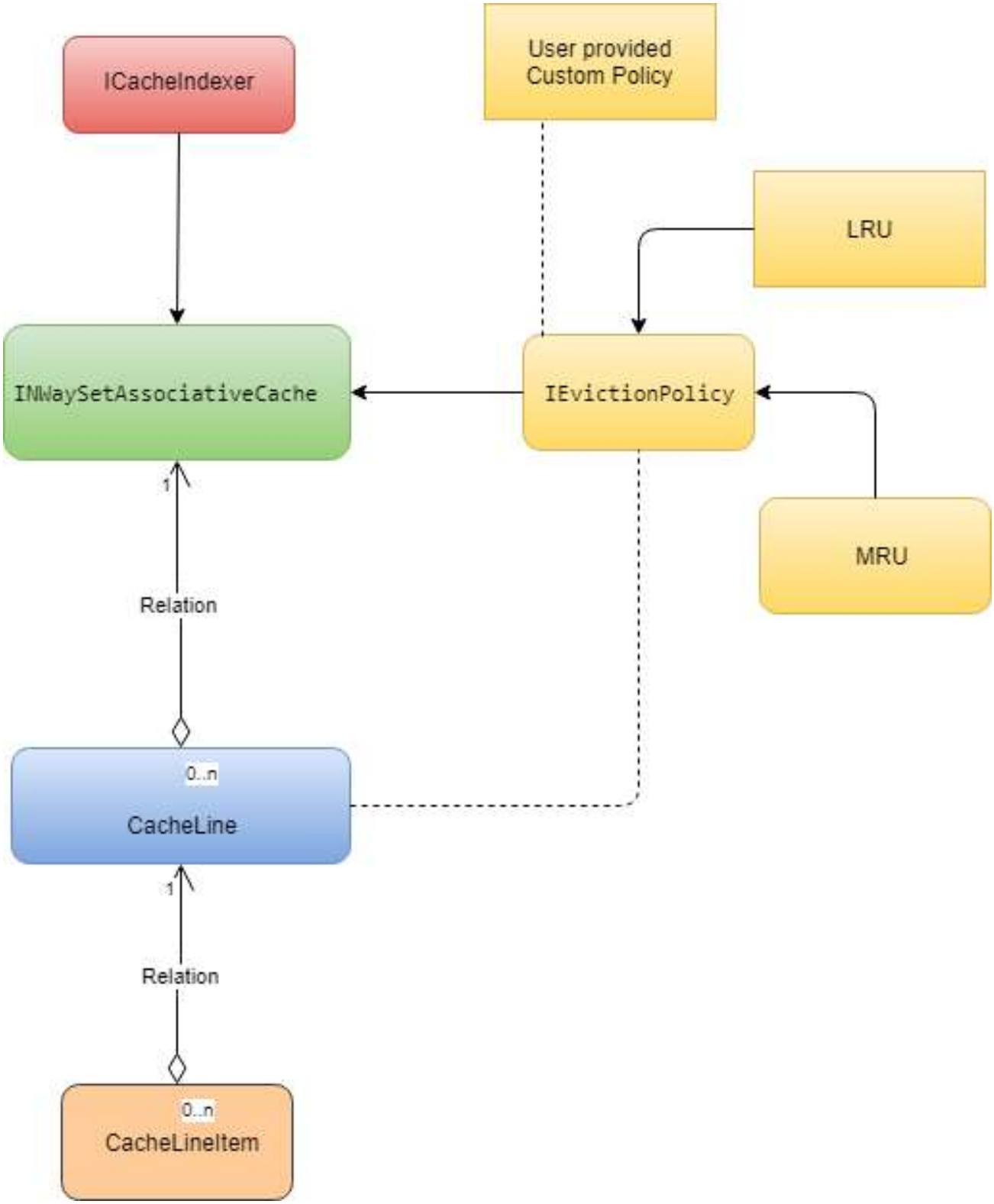
- Keys can be either primitive type or a struct or a class. In case of latter two the **user supplied key must implement IEquatable to work with Dictionaries which are used to maintain O(1) complexity of operations**, this is in line with current .Net associative containers.
- Values can be of any data type without any constraints.
- Eviction Policy is supplied as generics parameter to the INWaySetAssociativeCache interface. It has a constraint to implement IEvictionPolicy and must have a default constructor. This policy object is then injected via constructor to CacheLine.

ConcurrentDictionary<int, CacheLine<K, V>> CacheMap is used to store CacheLines (actual datastructure to store data). This map stores the CacheLine against an integer index created by ICacheIndexer which is injected in the constructor. This act as a load balancer to the set associative cache and can be supplied by the client if required. Default implementation takes modulus of HasCode of the Key around capacity supplied by the client.

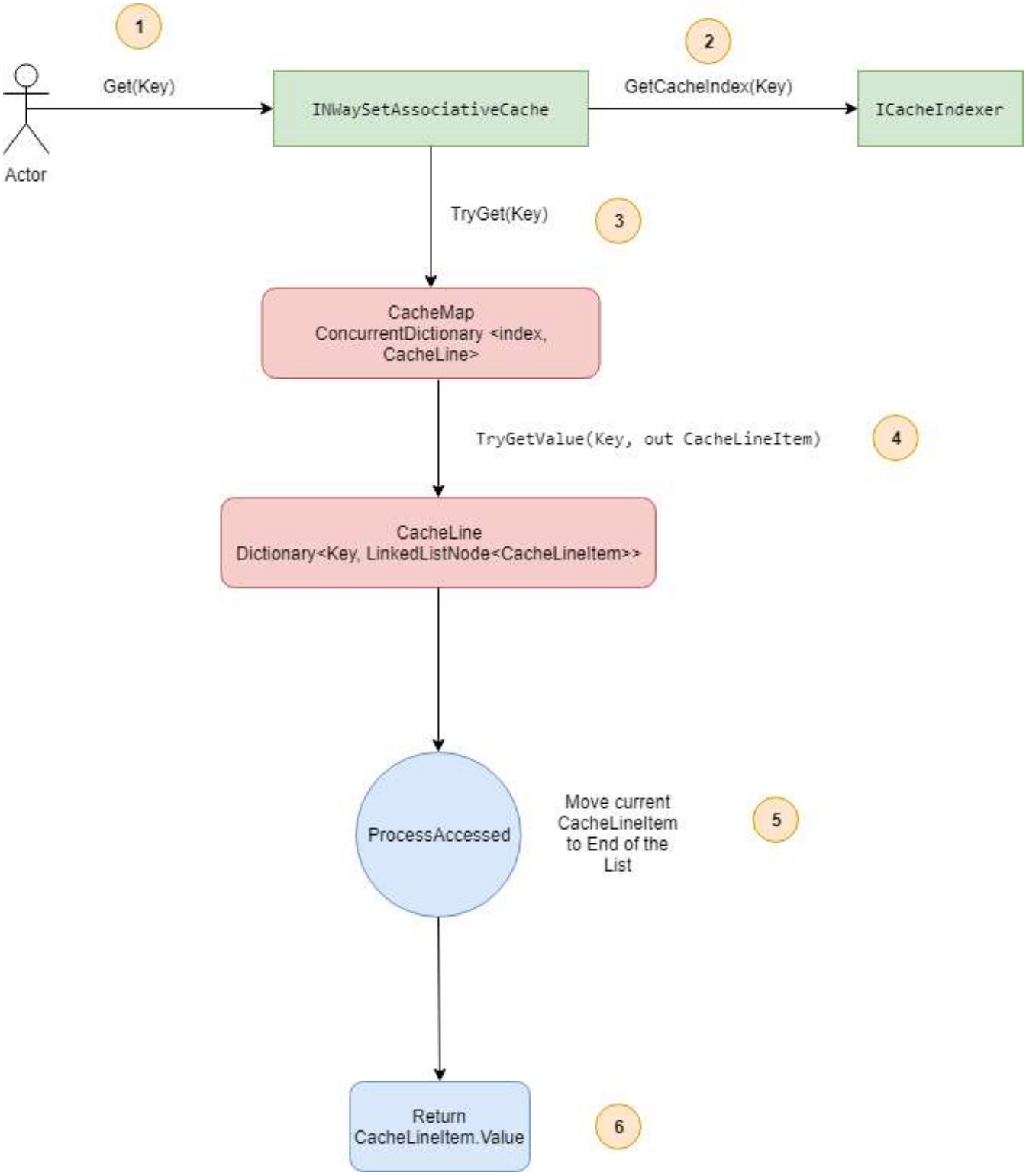
CacheLine<K, V> stores the data using a Dictionary<K, LinkedListNode<CacheLineItem>>. This enables us to access the LinkedListNode with O(1) complexity. Actual data is stored in this node from `LinkedList<CacheLineItem<K, V>>` of CacheLineItem which contains Key and Value.

Following is the top level architecture for the solution.

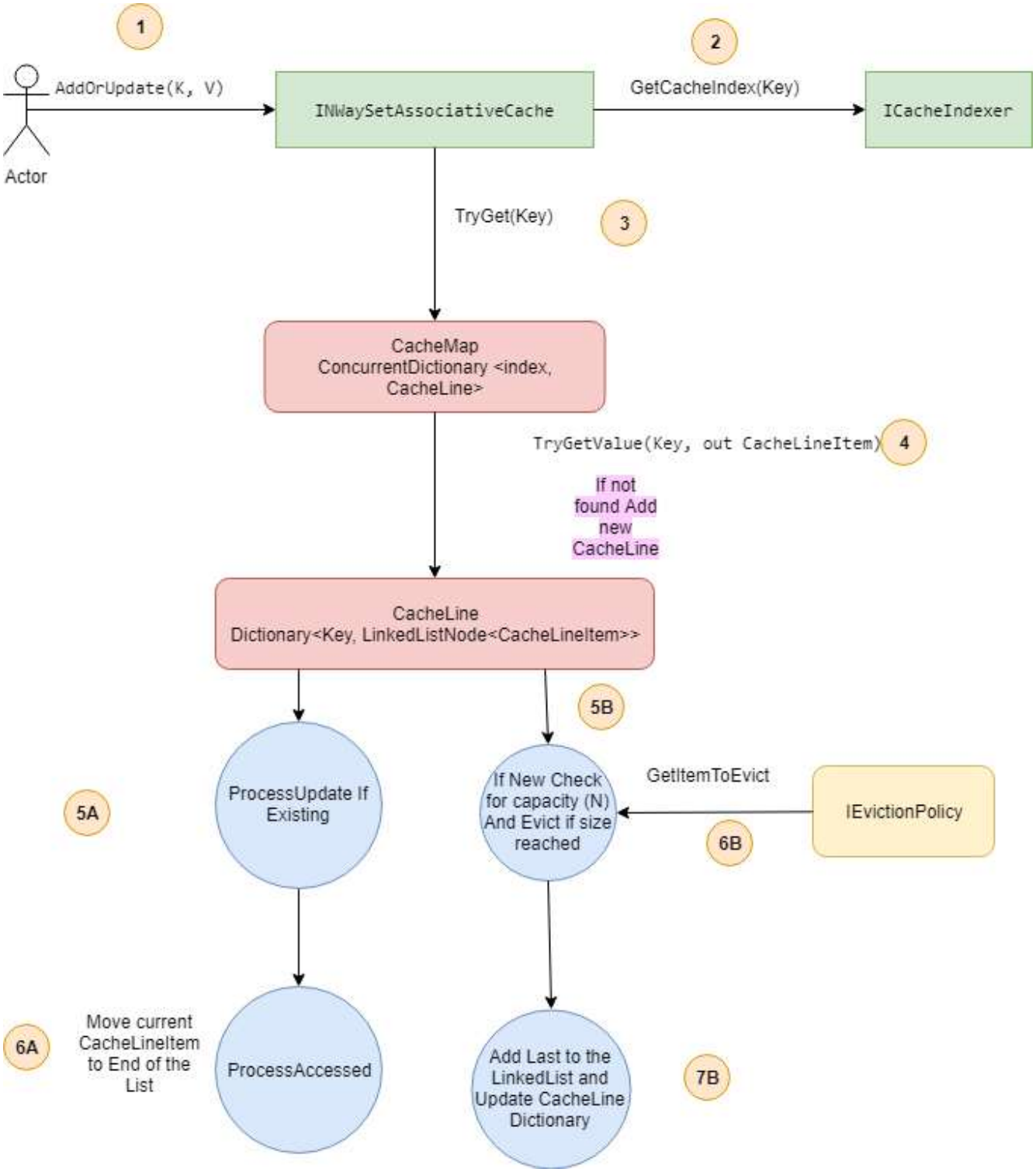
Top Level Architecture



Flow diagram for Get operation



Flow Diagram for AddOrUpdate operation



Testing

17 unit tests are added to the solution to test INWaySetAssociatveCache

These test cache indexing and N-way set associative cache operations with LRU, MRU and one client supplied custom eviction policies. Tests also ensure cache can be utilized for any given data type for the Keys and Values. Only constraints for Struct and Class types are that they must implement IEquatable interface in line with current .Net associative containers.

Usage

```
var N = 50;
var cacheLineCount = 10;
INWaySetAssociativeCache<Key, DataValue, LRUEvictionPolicy<Key, DataValue
>> cache = new NWaySetAssociativeCache< Key, DataValue, LRUEvictionPolicy<
Key, DataValue >>(N, new BoundedCacheIndexer< Key >(cacheLineCount));

var key = new Key (2);
var val = new DataValue (20);
cache.Add(key, val);
var returnedVal = cache.Get(key);
```

Unit Tests

Visual Studio 2017

Set Test Settings – Default Processing Architecture to x64