API-Research: STL, TBB & GCD

Lumina Wang

#0 Overview

Overview

For current OGS:

- Replace duplicated or deprecated tbb modules/APIs with STL;
- Keep TBB's concurrency container;
- Try tbb::task_arena;
- Balance tbb operator work;
- Separate data processing with pure cache data-structure operations.

For GSF:

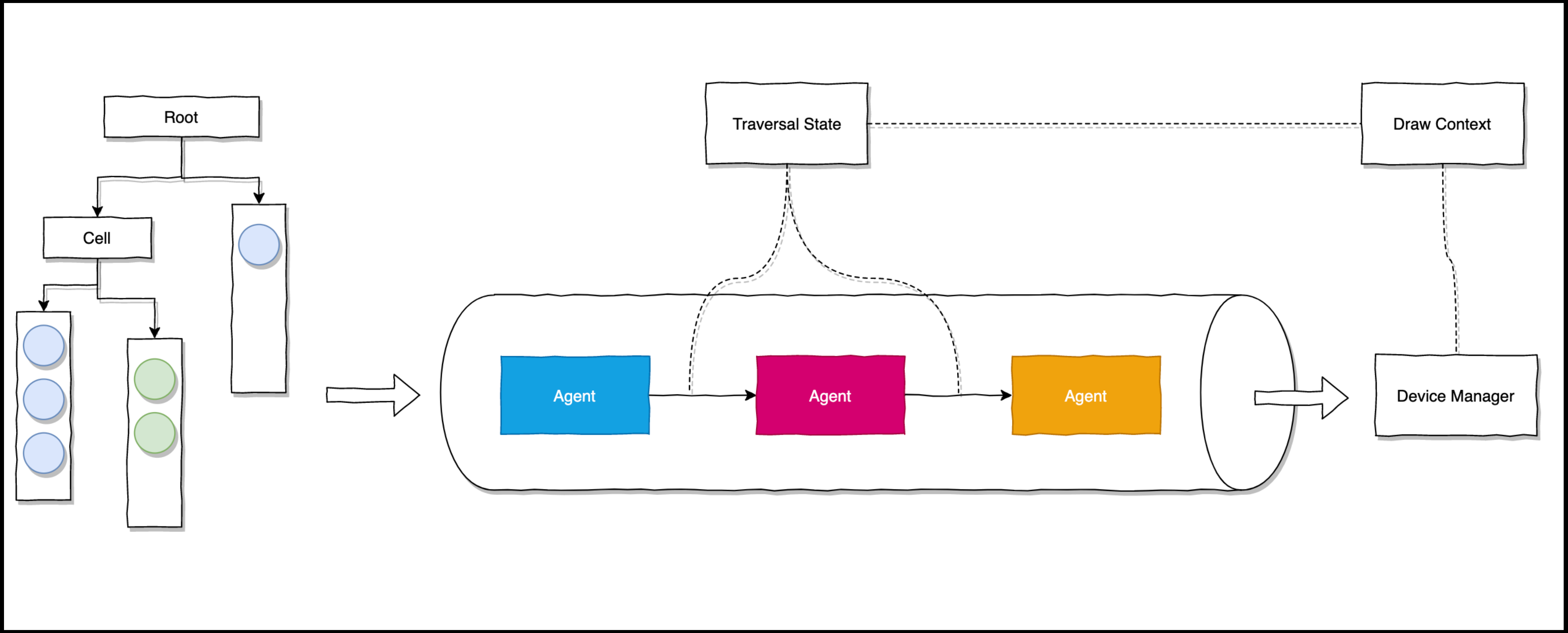
- For cross-platform compatibility, avoid using either GCD or TBB and limit within STL or boost;
- For platform-specific optimization, we could consider WinRT or GCD;
- Redesign parallel traversing with STL;
- Redesign OGS containers so that users can clearly choose the thread-safe or the non-thread-safe version.

Lib	Platform Support	Thread-Safe Container	Parallel Algorithms
TBB	Windows, Linux, macOS	YES	tbb::parallel_do
STL	Windows, Linux, macOS	NO (alternative: boost)	stl::async
GCD	macOS	NO	dispatch_async

Parallel Computing Applications in Graphics / Games

Parallel Computing	Description	Application
Sparse Linear Algebra (e.g., SpMV, OSKI, or SuperLU)	Data sets include many zero values. Data is usually stored in compressed matrices to reduce the storage and bandwidth requirements to access all of the nonzero values. One example is block compressed sparse row (BCSR). Because of the compressed formats, data is generally accessed with indexed loads and stores.	Reverse kinematics; Spring models
Spectral Methods (e.g., FFT)	Data are in the frequency domain, as opposed to time or spatial domains. Typically, spectral methods use multiple butterfly stages, which combine multiply-add operations and a specific pattern of data permutation, with all-to-all communication for some stages and strictly local for others.	Texture maps
Structured Grids (e.g., Cactus or Lattice- Boltzmann Magneto-hydrodynamics)	Represented by a regular grid; points on grid are conceptually updated together. It has high spatial locality. Updates may be in place or between 2 versions of the grid. The grid may be subdivided into finer grids in areas of interest ("Adaptive Mesh Refinement"); and the transition between granularities may happen dynamically.	Smoothing; interpolation
Graph Traversal (e.g., Quicksort)	Visits many nodes in a graph by following successive edges. These applications typically involve many levels of indirection, and a relatively small amount of computation.	Reverse kinematics, collision detection, depth sorting, hidden surface removal
Finite State Machine	A system whose behavior is defined by states, transitions defined by inputs and the current state, and events associated with transitions or states.	Response to collisions

Current OGS Usages



- Coupled Data Structure & Algorithms, typically BSP tree & consolidation
- Numerous Global Cache introducing data race;
- Uncontrollable IteratorStream (e.g. Swapchain) resulting in starting up more than once tbb::parallel_while in a single traversal.

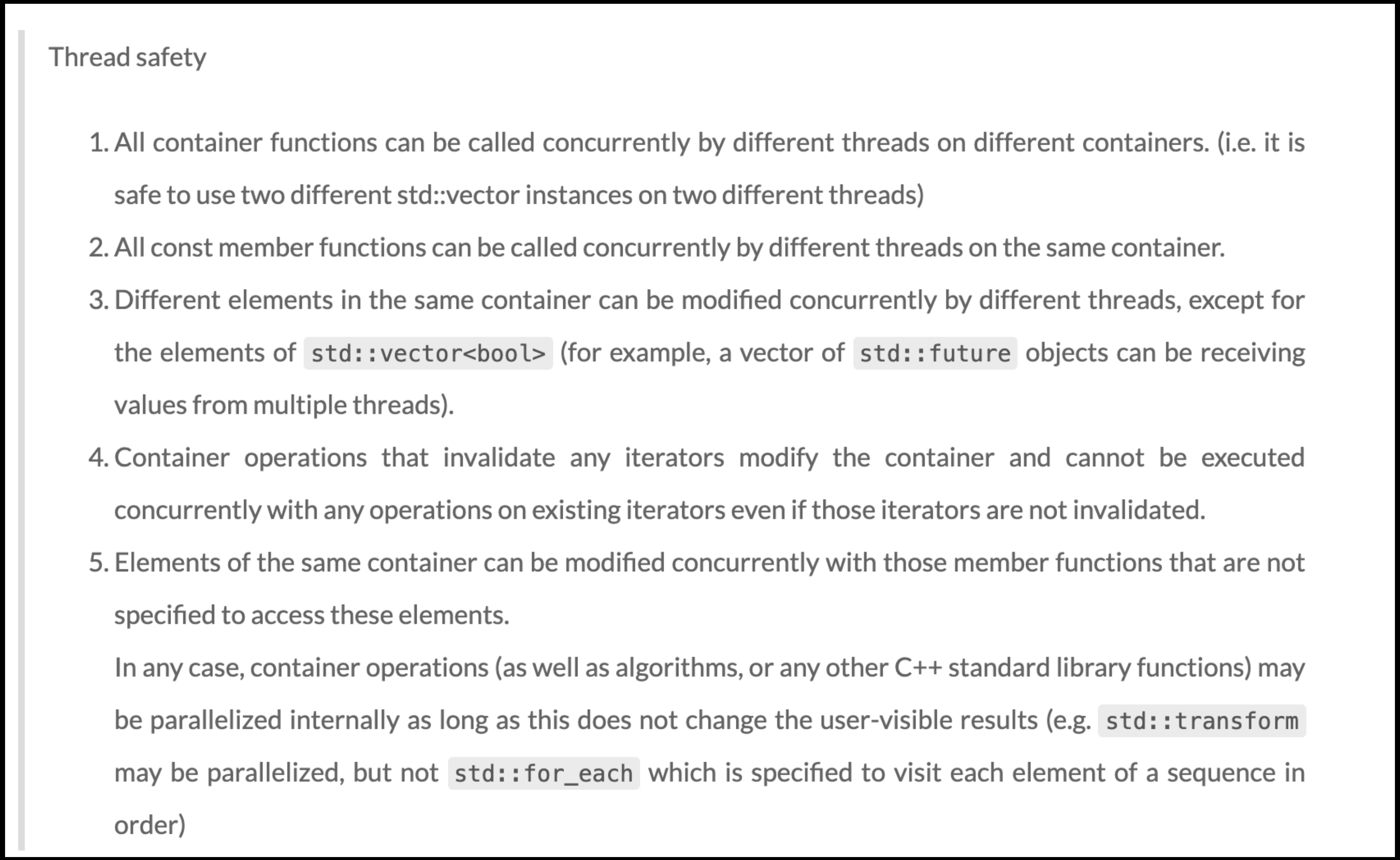
#1 STL

Compiler Support

- C++11 ✓
- C++14 ✓
- C++17 ●
- C++2a ✗

C++17 feature	GCC libstdc++	Clang libc++	MSVC Standard Library
Hardware interference size	-	-	19.11
Elementary string conversions	8 (no FP)	7 (no FP)	19.24
std::shared_ptr / std::weak_ptr with array support	7	-	19.12

Thread-Safe Container

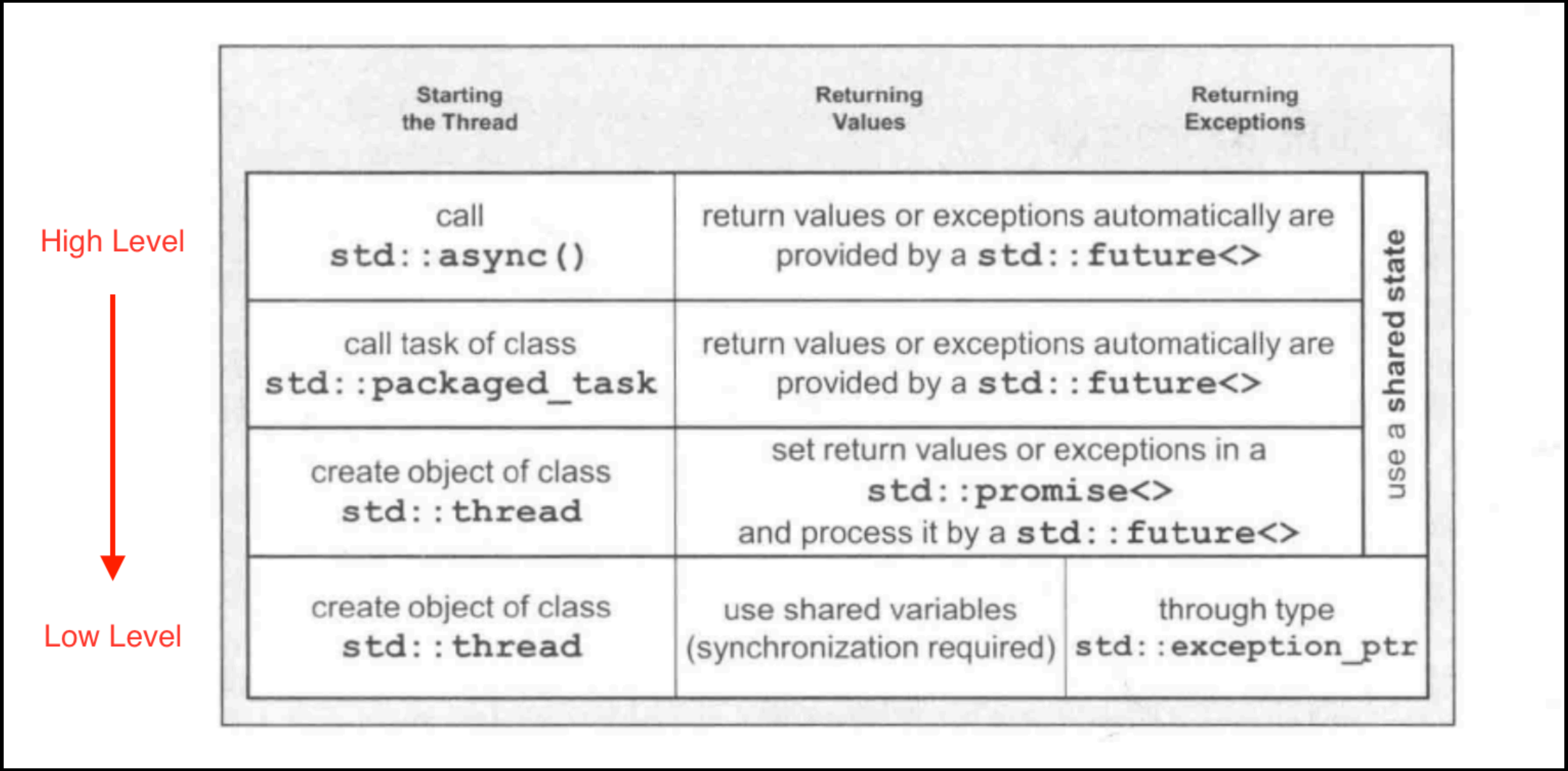


Thread safety

1. All container functions can be called concurrently by different threads on different containers. (i.e. it is safe to use two different std::vector instances on two different threads)
2. All const member functions can be called concurrently by different threads on the same container.
3. Different elements in the same container can be modified concurrently by different threads, except for the elements of `std::vector<bool>` (for example, a vector of `std::future` objects can be receiving values from multiple threads).
4. Container operations that invalidate any iterators modify the container and cannot be executed concurrently with any operations on existing iterators even if those iterators are not invalidated.
5. Elements of the same container can be modified concurrently with those member functions that are not specified to access these elements.
   In any case, container operations (as well as algorithms, or any other C++ standard library functions) may be parallelized internally as long as this does not change the user-visible results (e.g. `std::transform` may be parallelized, but not `std::for_each` which is specified to visit each element of a sequence in order)

- Generally speaking, if we want to modify the container in multiple threads, we need to protect that container to prevent concurrent access.
- For this situation, boost provides a selection of lock-free container alternatives which we can consider for multi-threaded use cases.

Parallel Algorithms


High Level
Low Level
use a shared state

Starting the Thread	Returning Values	Returning Exceptions
call `std::async()`	return values or exceptions automatically are provided by a `std::future<>`	
call task of class `std::packaged_task`	return values or exceptions automatically are provided by a `std::future<>`	
create object of class `std::thread`	set return values or exceptions in a `std::promise<>` and process it by a `std::future<>`	
create object of class `std::thread`	use shared variables (synchronization required)	through type `std::exception_ptr`

- The function template async runs the function f asynchronously (potentially in a separate thread which may be part of a thread pool) and returns a std::future that will eventually hold the result of that function call.

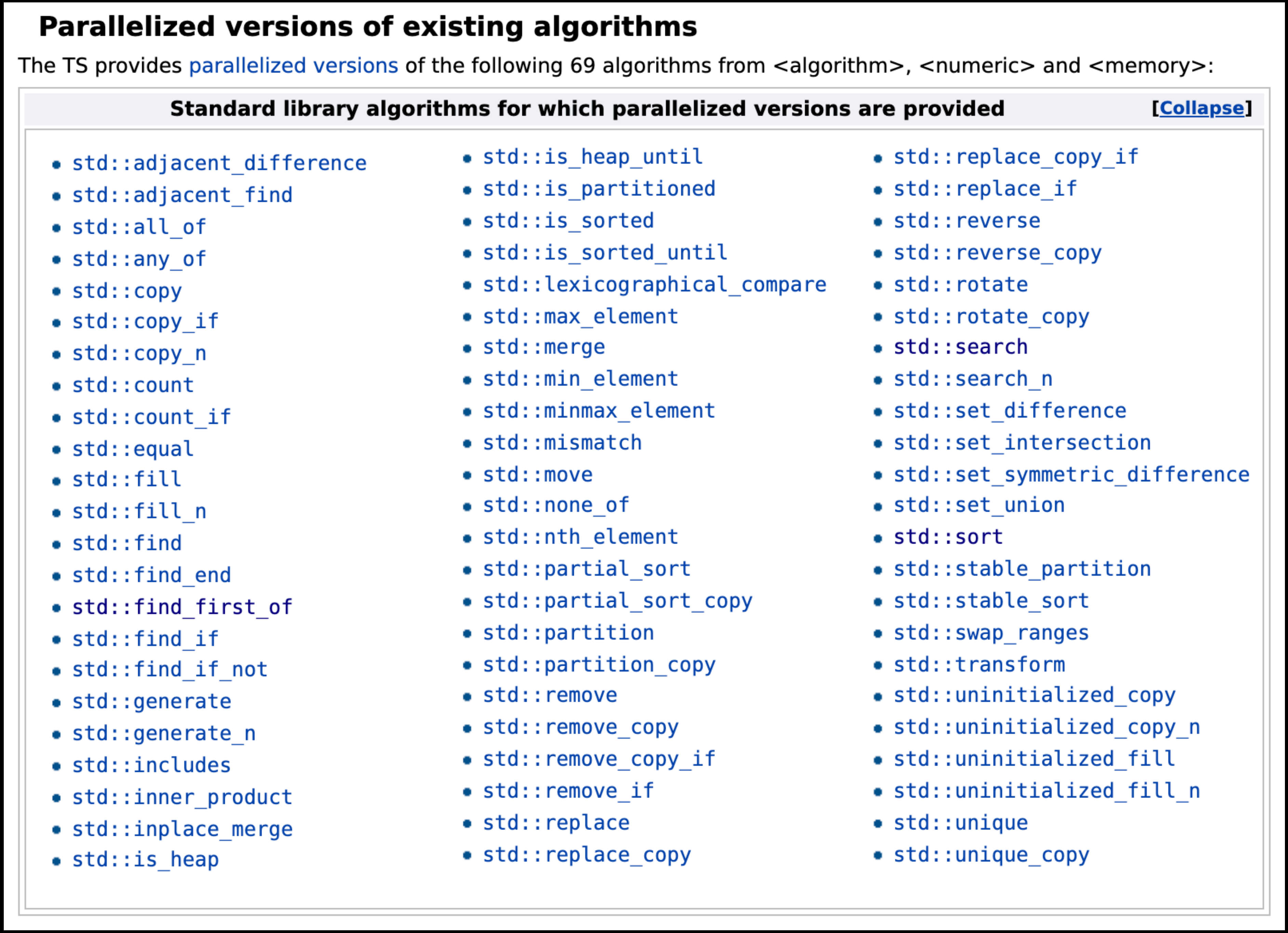
- Refer the Sample Code.

Parallel Algorithms

Further more, in C++17, Standard library implementations (but not the users) may define additional execution policies as an extension:

- std::execution::seq
- std::execution::par
- std::execution::par_unseq
- std::execution::unseq

Refer the Sample Code.



Parallelized versions of existing algorithms

The TS provides parallelized versions of the following 69 algorithms from <algorithm>, <numeric> and <memory>:

Standard library algorithms for which parallelized versions are provided [Collapse]

- std::adjacent_difference
- std::adjacent_find
- std::all_of
- std::any_of
- std::copy
- std::copy_if
- std::copy_n
- std::count
- std::count_if
- std::equal
- std::fill
- std::fill_n
- std::find
- std::find_end
- std::find_first_of
- std::find_if
- std::find_if_not
- std::generate
- std::generate_n
- std::includes
- std::inner_product
- std::inplace_merge
- std::is_heap
- std::is_heap_until
- std::is_partitioned
- std::is_sorted
- std::is_sorted_until
- std::lexicographical_compare
- std::max_element
- std::merge
- std::min_element
- std::minmax_element
- std::mismatch
- std::move
- std::none_of
- std::nth_element
- std::partial_sort
- std::partial_sort_copy
- std::partition
- std::partition_copy
- std::remove
- std::remove_copy
- std::remove_copy_if
- std::remove_if
- std::replace
- std::replace_copy
- std::replace_copy_if
- std::replace_if
- std::reverse
- std::reverse_copy
- std::rotate
- std::rotate_copy
- std::search
- std::search_n
- std::set_difference
- std::set_intersection
- std::set_symmetric_difference
- std::set_union
- std::sort
- std::stable_partition
- std::stable_sort
- std::swap_ranges
- std::transform
- std::uninitialized_copy
- std::uninitialized_copy_n
- std::uninitialized_fill
- std::uninitialized_fill_n
- std::unique
- std::unique_copy

Summary

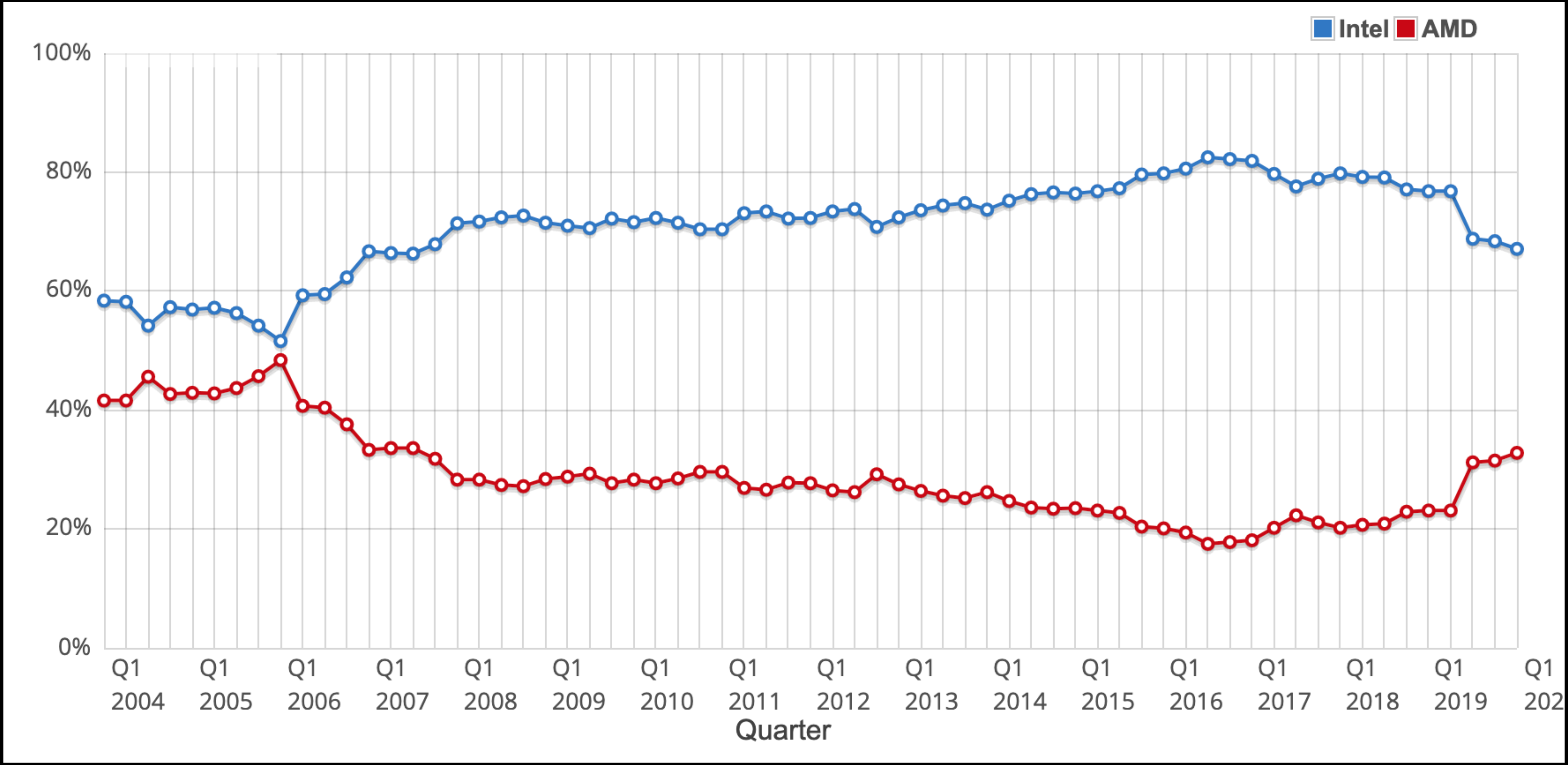
- Pros:
  - Real Cross-platform with stable & continued support;
  - No further 3rd party;
  - Various-level functions to be adopted based on usages;
  - Exception mechanism.
- Cons:
  - Too detailed & specific.
- Problem: Thread-local Cache
  Optional Solution: Refactor Cache Mechanism: Keep Useful Data into Cache and release all the thread-local cache when ending tasks.

#2 TBB

Platform Support

Category	Description
Processors	Optimized for all compatible Intel® processors including Intel Atom®, Intel® Core™, Intel® Xeon®, and Intel® Xeon Phi™ processors.
Language	C++
Portability and Compatibility	Open source under an Apache* license Compatible with multiple compilers (compiler agnostic)
Operating Systems	Windows, Linux, macOS, Android additional with open source

Market Survey



	Base speed	Max speed	TDP	Price
Ryzen 7 3800X	3.9GHz	4.5GHz	105 watts	$399
Core i9-9900K	3.6GHz	5.0GHz	95 watts	$488

Duplicated with STL

TBB	STL
tbb::atomic	std::atomic
tbb::tbb_thread	std::thread
tbb::mutex	std::mutex
tbb::recursive_mutex	std::recursive_mutex
tbb::reader_writer_lock	std::shared_mutex
tbb::hash & tbb:hasher	std::hash

Thread-Safe Container

- Unordered associative containers
  - Unordered map (including unordered multimap)
  - Unordered set (including unordered multiset)
  - Hash table
- Queue (including bounded queue and priority queue)
- Vector

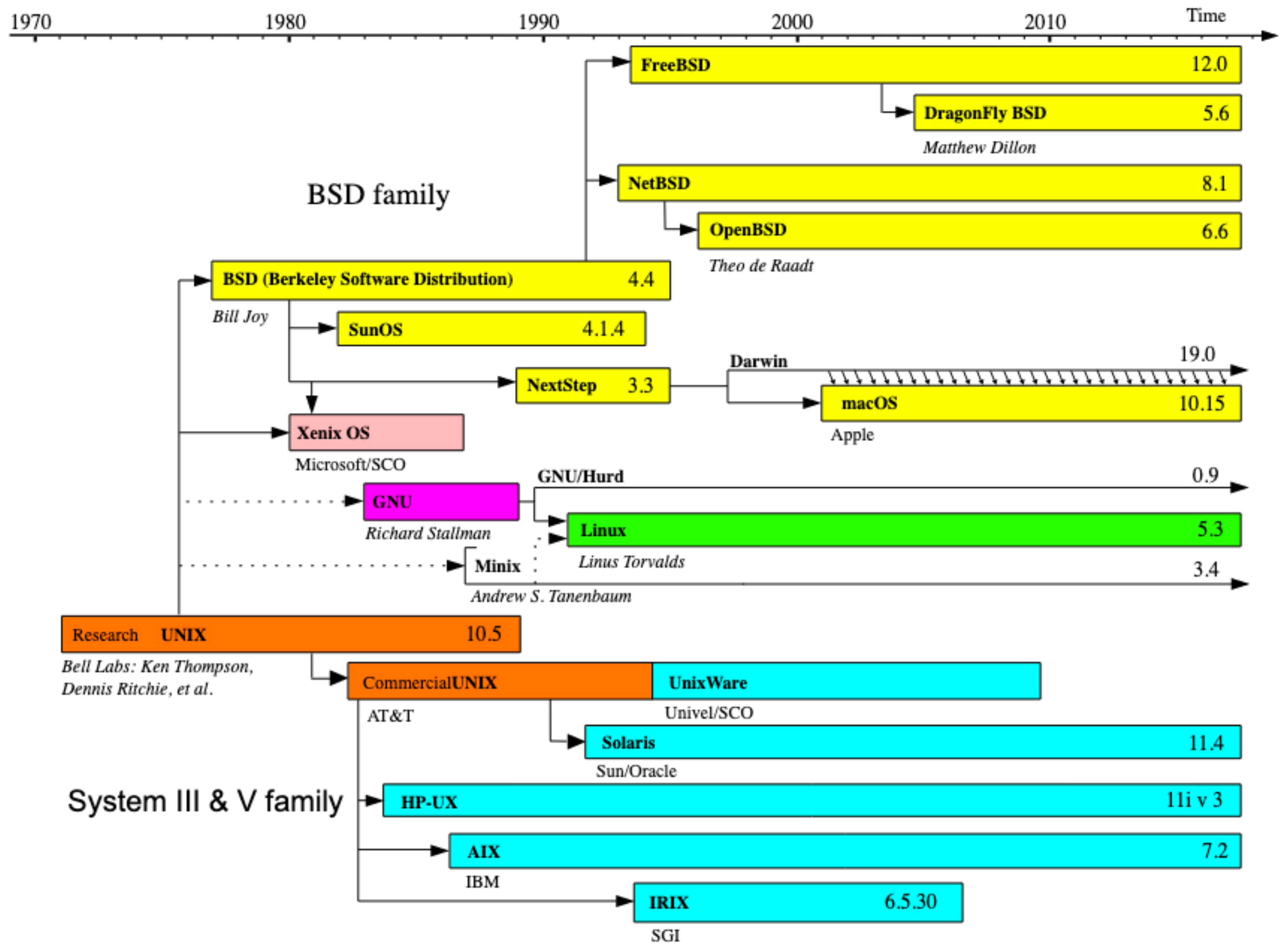
Class Name	Concurrent traversal and insertion	Keys have a value associated with them	Support concurrent erasure	Built-in locking	No visible locking (lock-free interface)	Identical items allowed to be inserted	[] and at accessors
tbb::concurrent_hash_map	YES	YES	YES	YES	NO	NO	NO
tbb::concurrent_unordered_map	YES	YES	NO	NO	YES	NO	YES
tbb::concurrent_unordered_multimap	YES	YES	NO	NO	YES	YES	NO
tbb::concurrent_unordered_set	YES	NO	NO	NO	YES	NO	NO
tbb::concurrent_unordered_multiset	YES	NO	NO	NO	YES	YES	NO

Detailed Usage

- Refer to Docs.

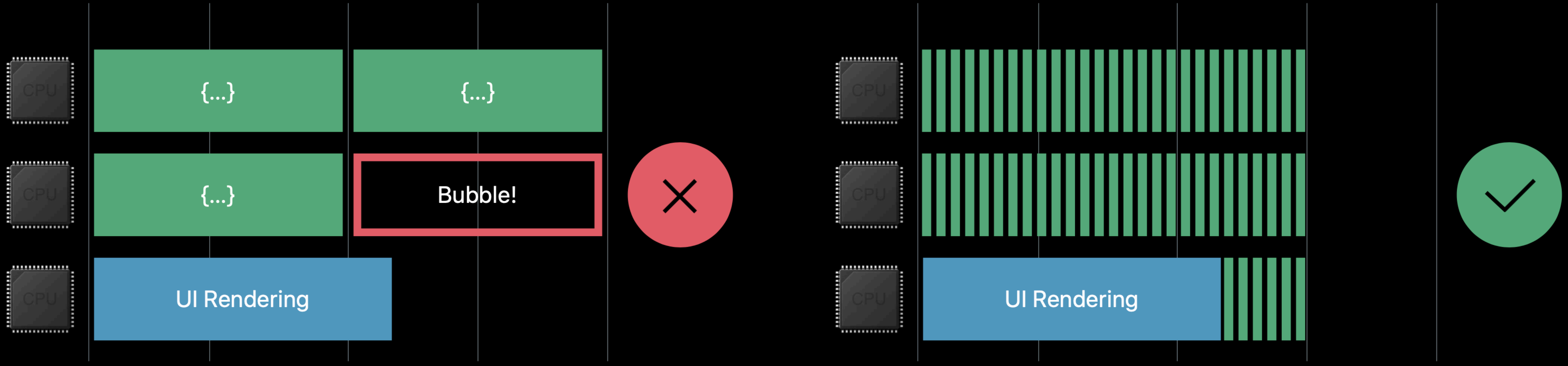
#3 GCD

Platform Support



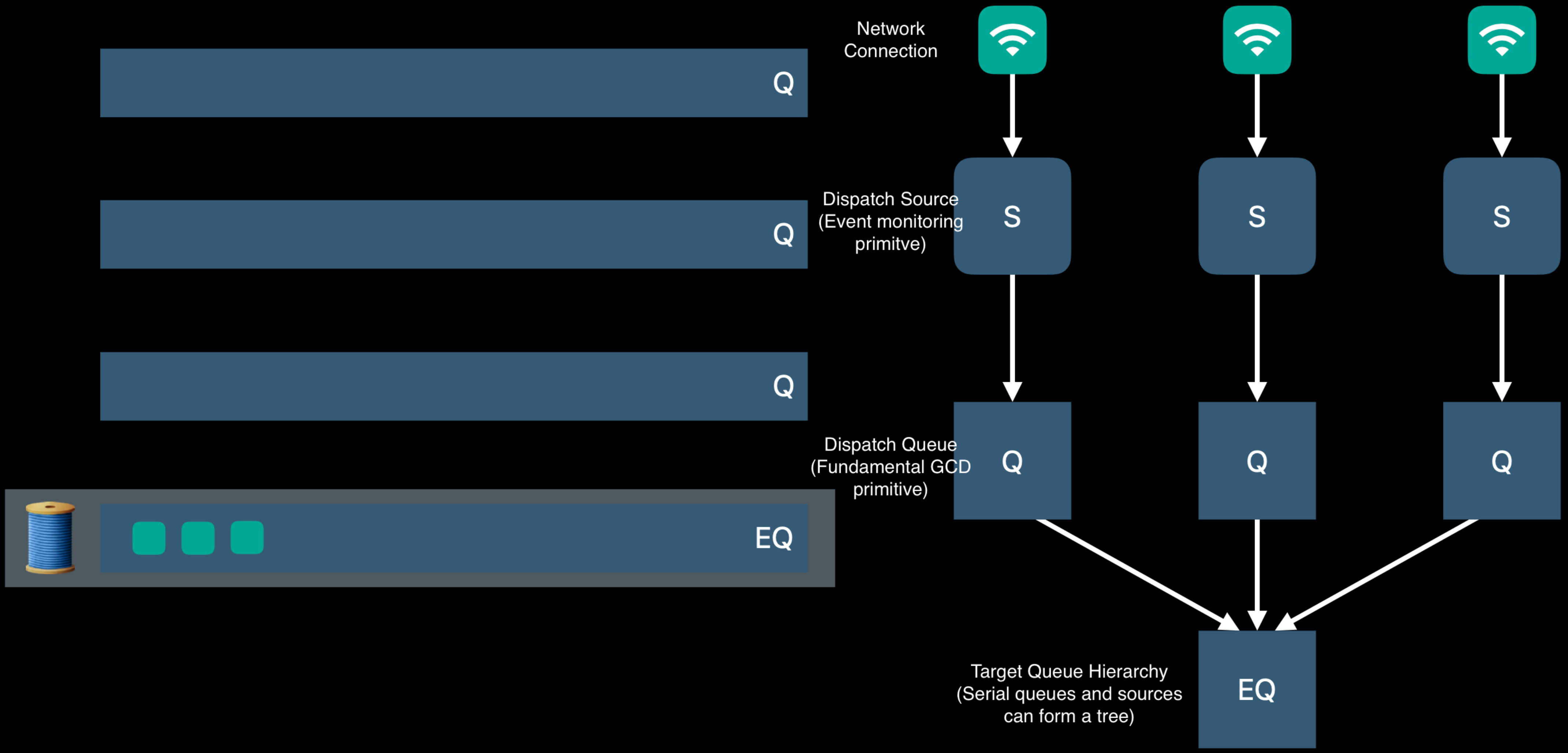
High-Level Usages

- Parallelism with GCD

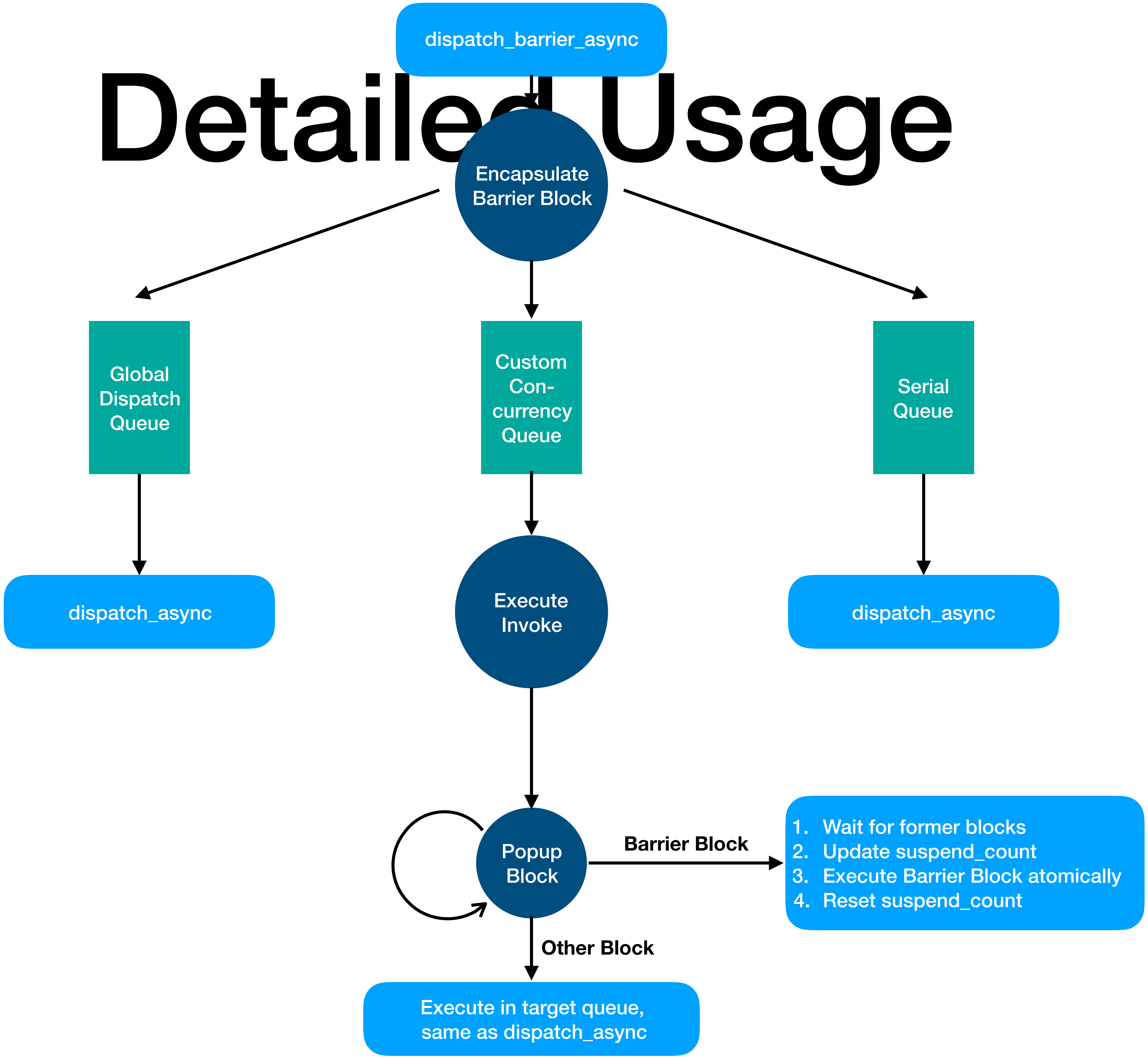


High-Level Usages

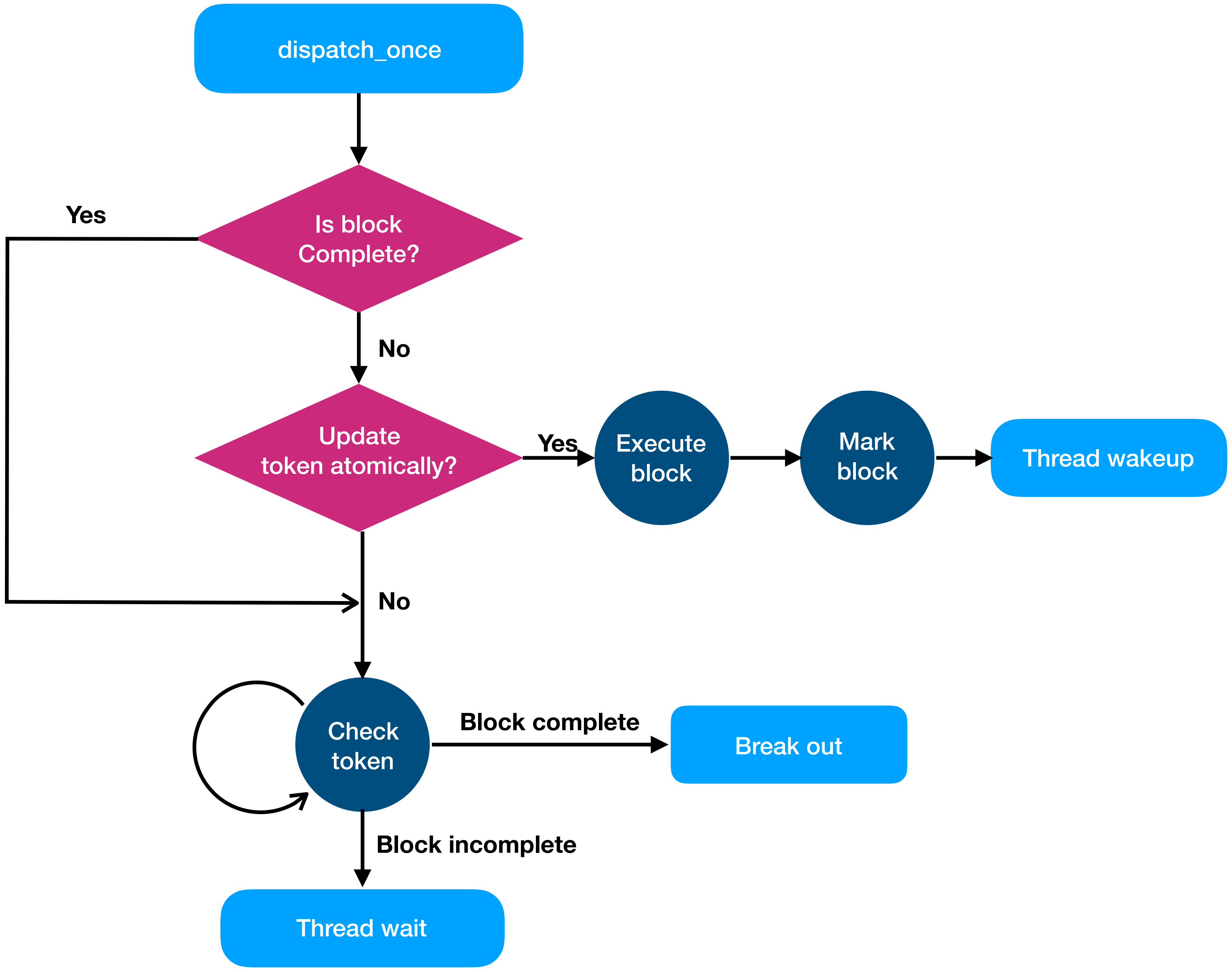
- Concurrency for Independently Executed Tasks


Network Connection
Dispatch Source (Event monitoring primitve)
Dispatch Queue (Fundamental GCD primitive)
Target Queue Hierarchy (Serial queues and sources can form a tree)
Q
Q
Q
EQ
S
S
S
Q
Q
Q
EQ

Detailed Usage



Detailed Usage



Detailed Usage

- Refer to Docs.