Twisted Atomics

www.conceptslearningmachine.com

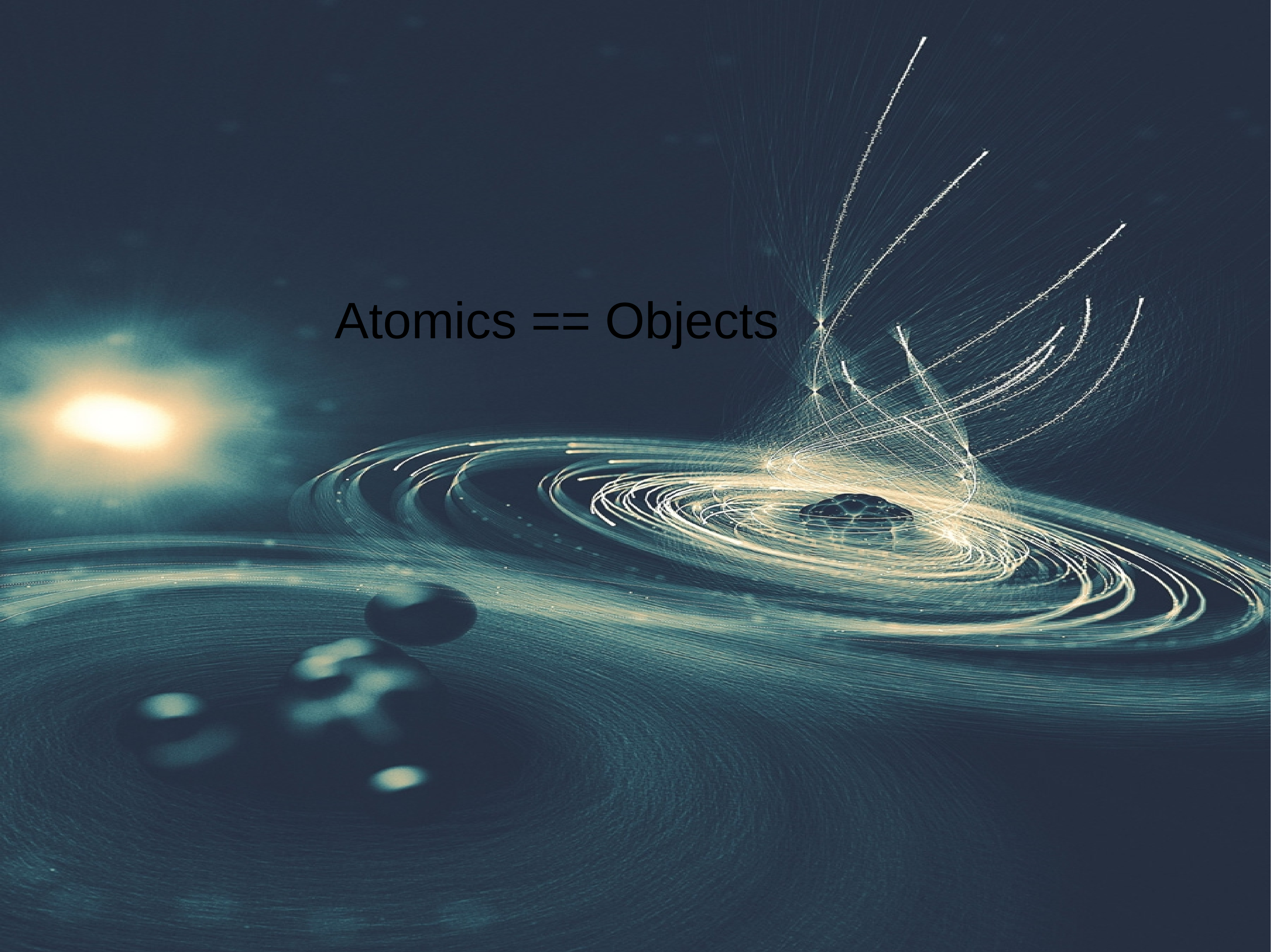
We have multiple slabs of physical and/or virtual memory that needs protecting!

! the need for solutions !

0000-the process

the_process_atomic_x*_x*<*>

We can link objects to atomic operations



Atomics == Objects

SP == OOP

Be System Programmers that have
Object Oriented Programming principles.

Compile[ER’s]

Each with it’s own set of rules, sum very similar to being custom workspaces…

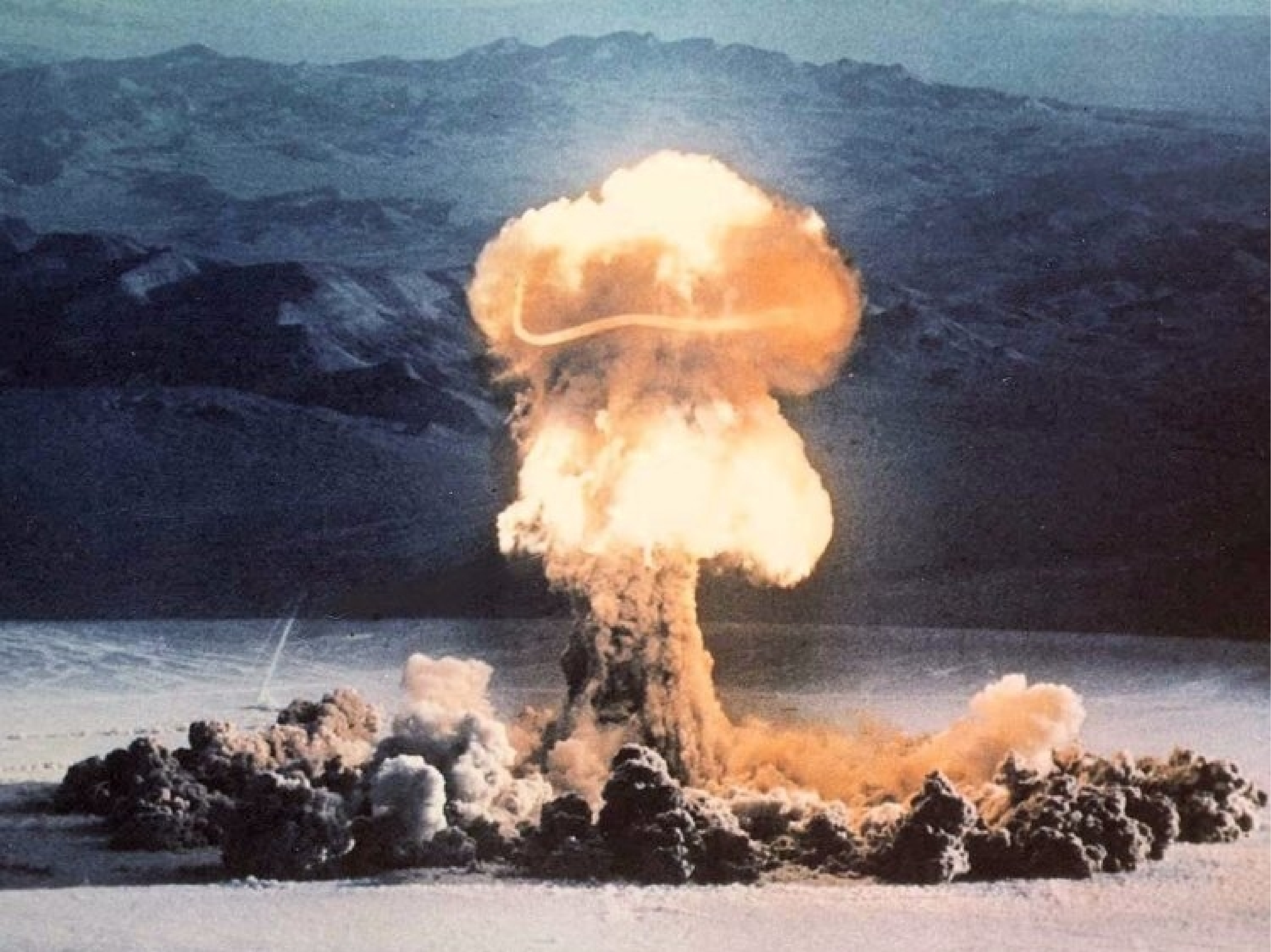
WORD len*

(Explain jokingly & briefly how they
count & define(does false inputs)Ha_Hohoho!)

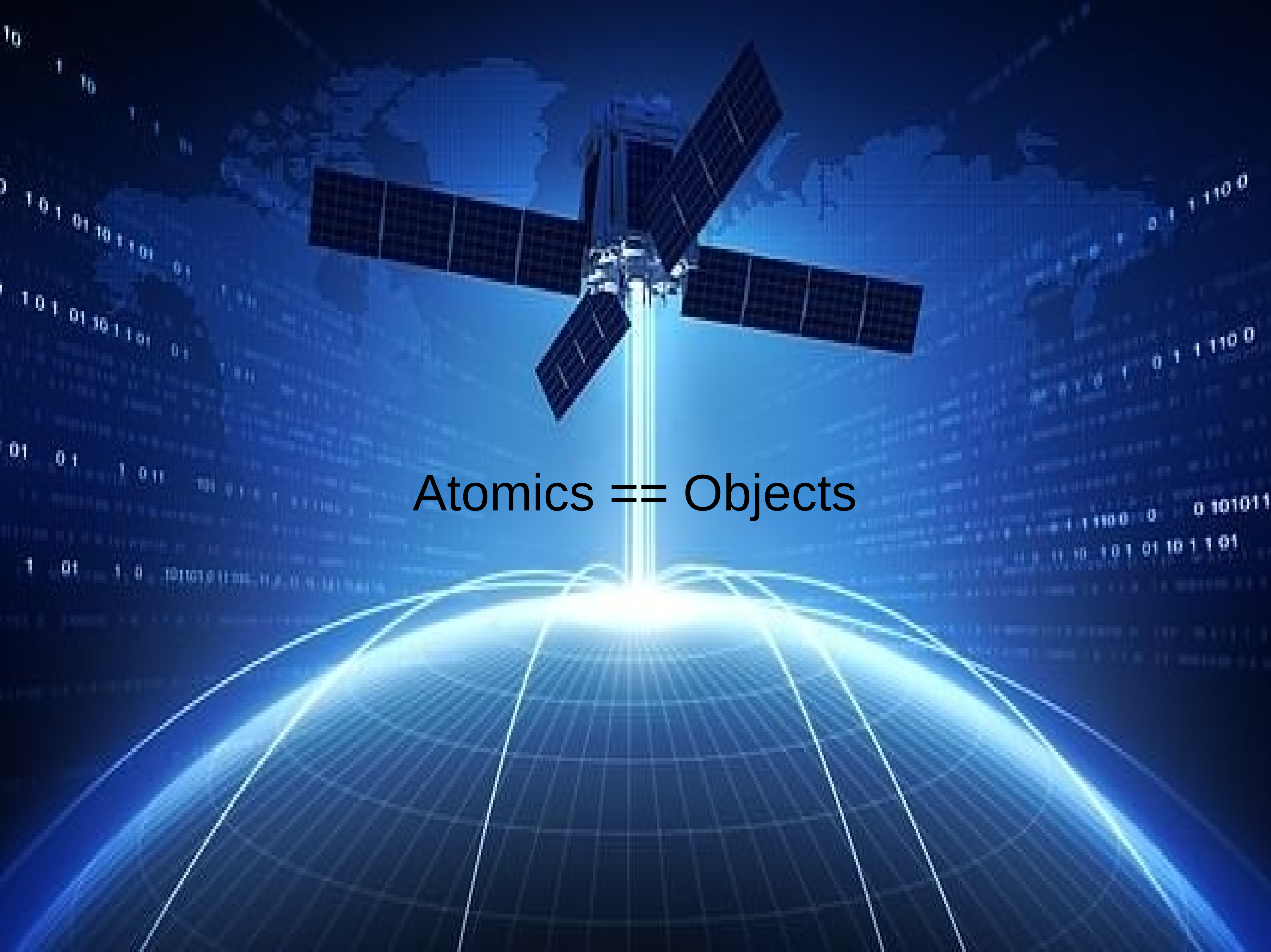
In short, we write coherent directed code that corresponds with arch & compiler specification:::

Atomics?

What your program will do when multiple event counters need to be in sync, comply and can’t verify>>>

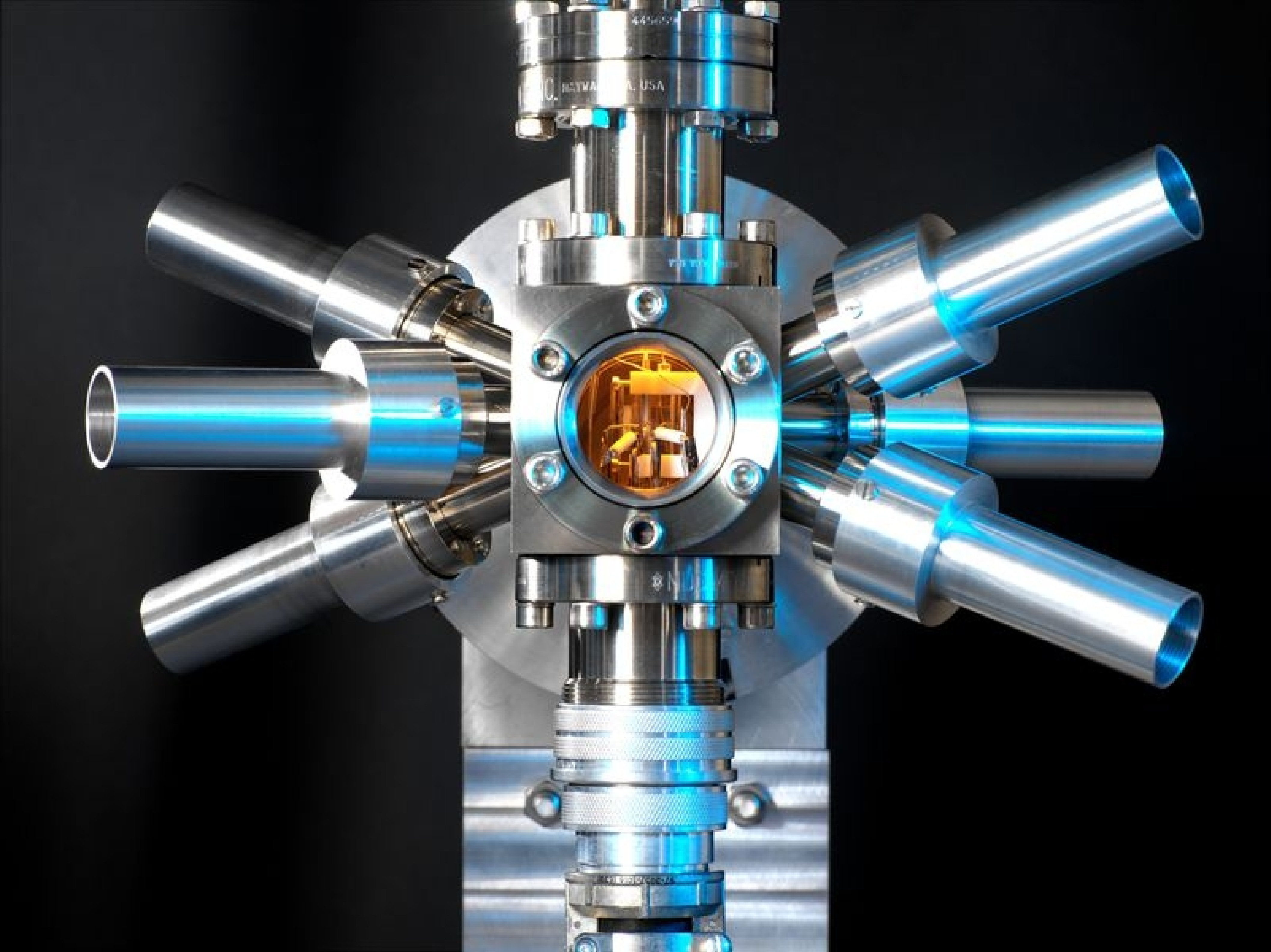


Remember



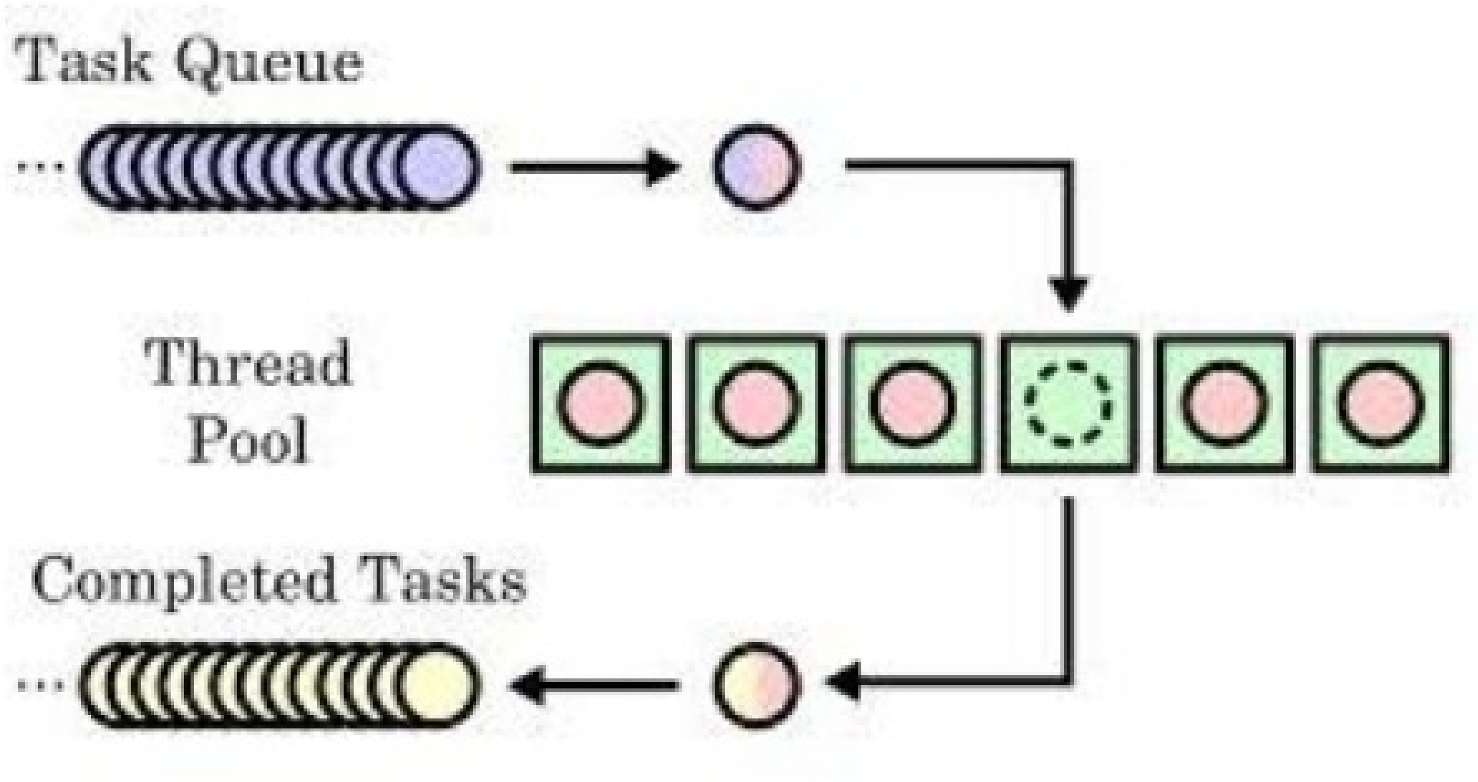
Atomics == Objects

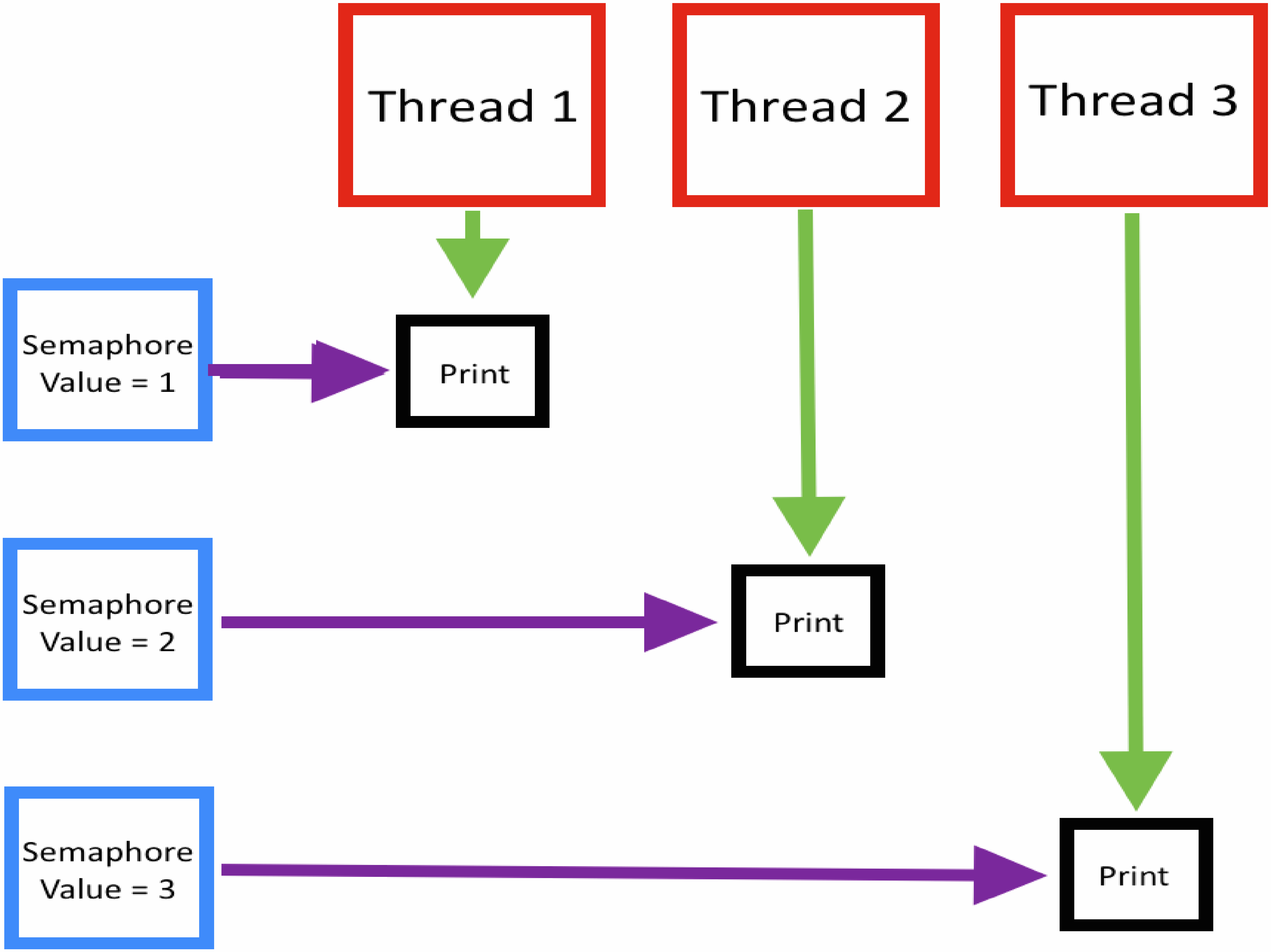
Works like this for now---

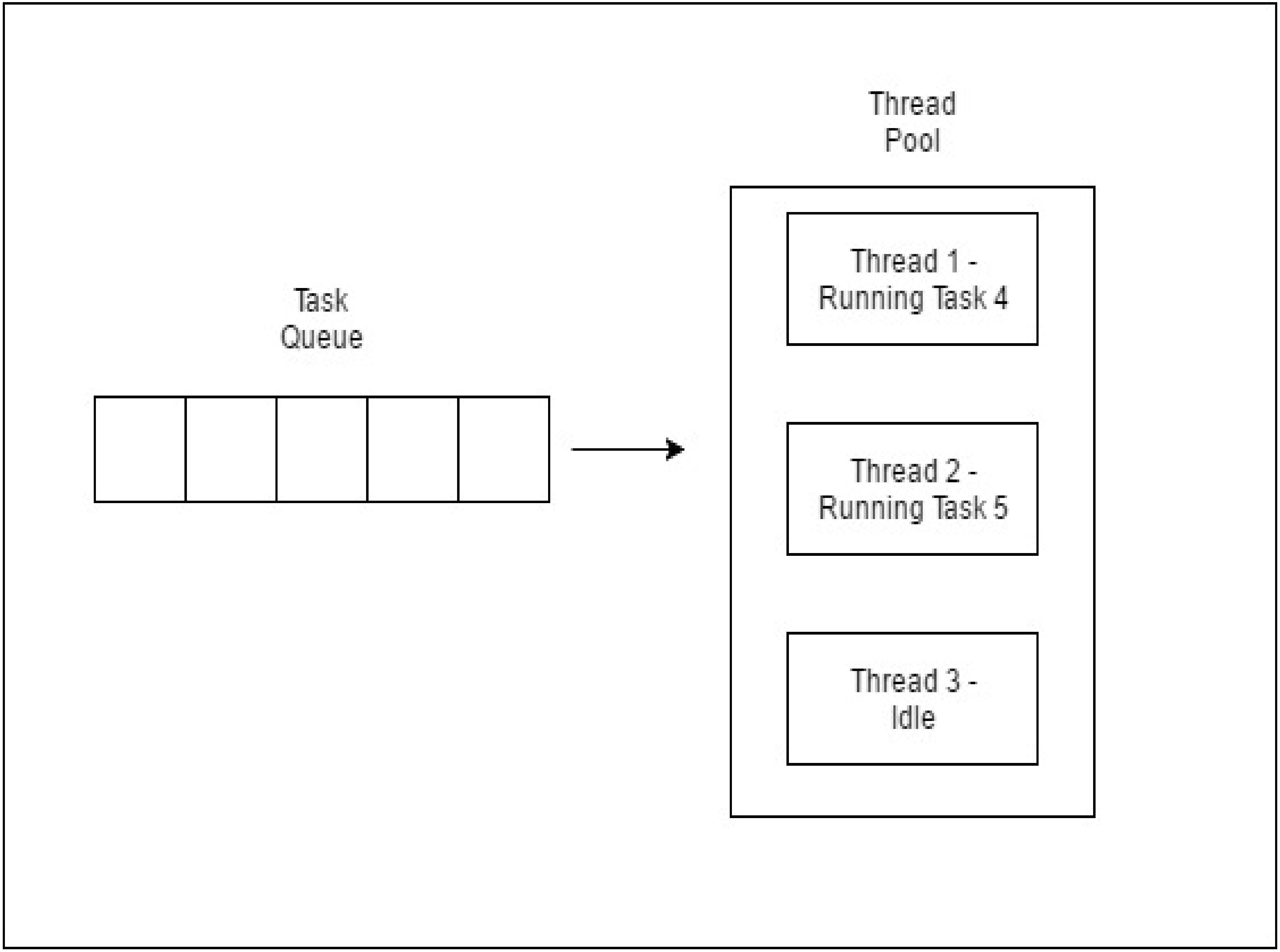


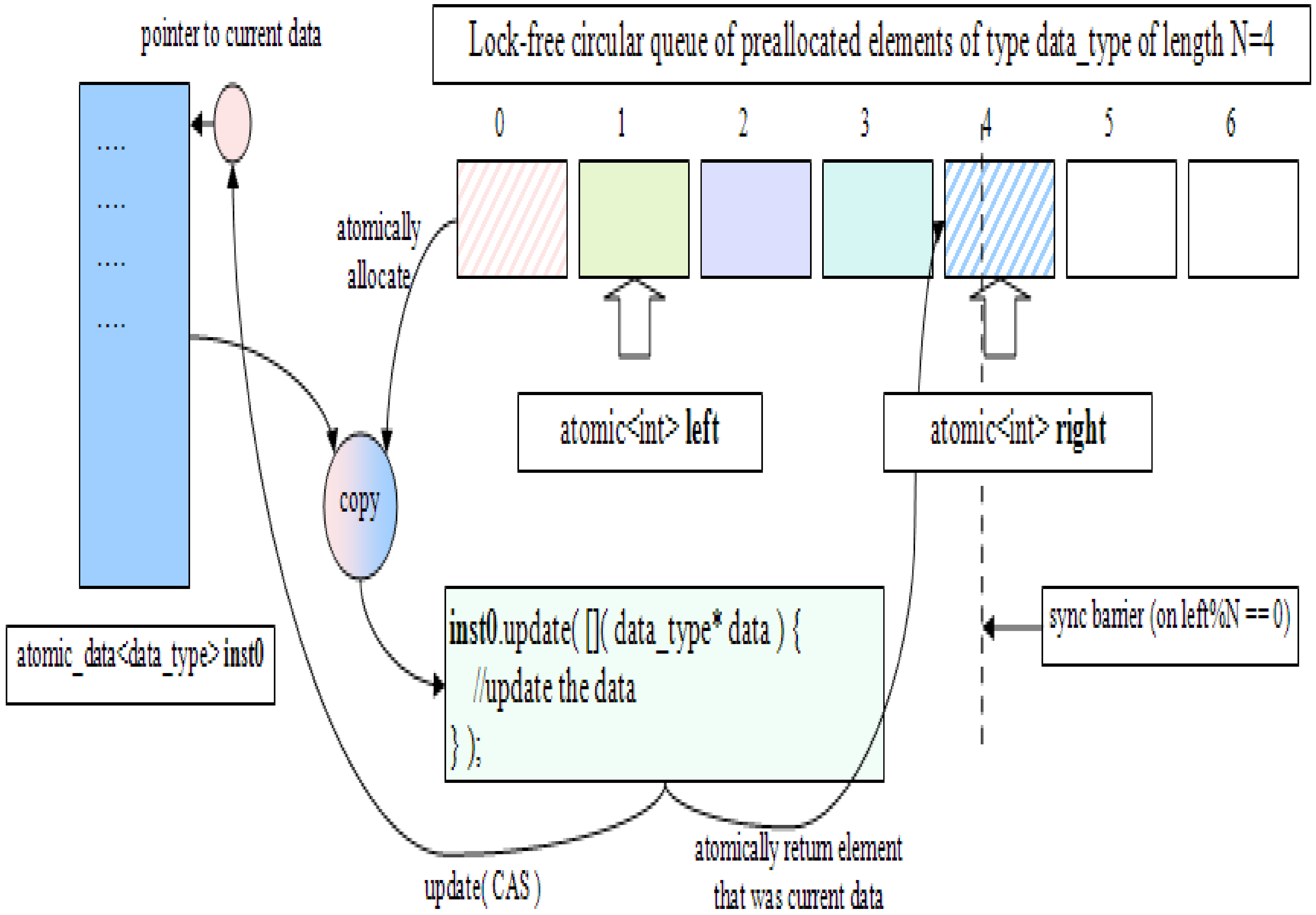
As MOTs on the atomic clock they process as there own threads with data & time syncs traveling through them while being logged.

Thread Pool




Thread 1
Thread 2
Thread 3
Semaphore Value = 1
Print
Semaphore Value = 2
Print
Semaphore Value = 3
Print


Task Queue
Thread Pool
Thread 1 - Running Task 4
Thread 2 - Running Task 5
Thread 3 - Idle



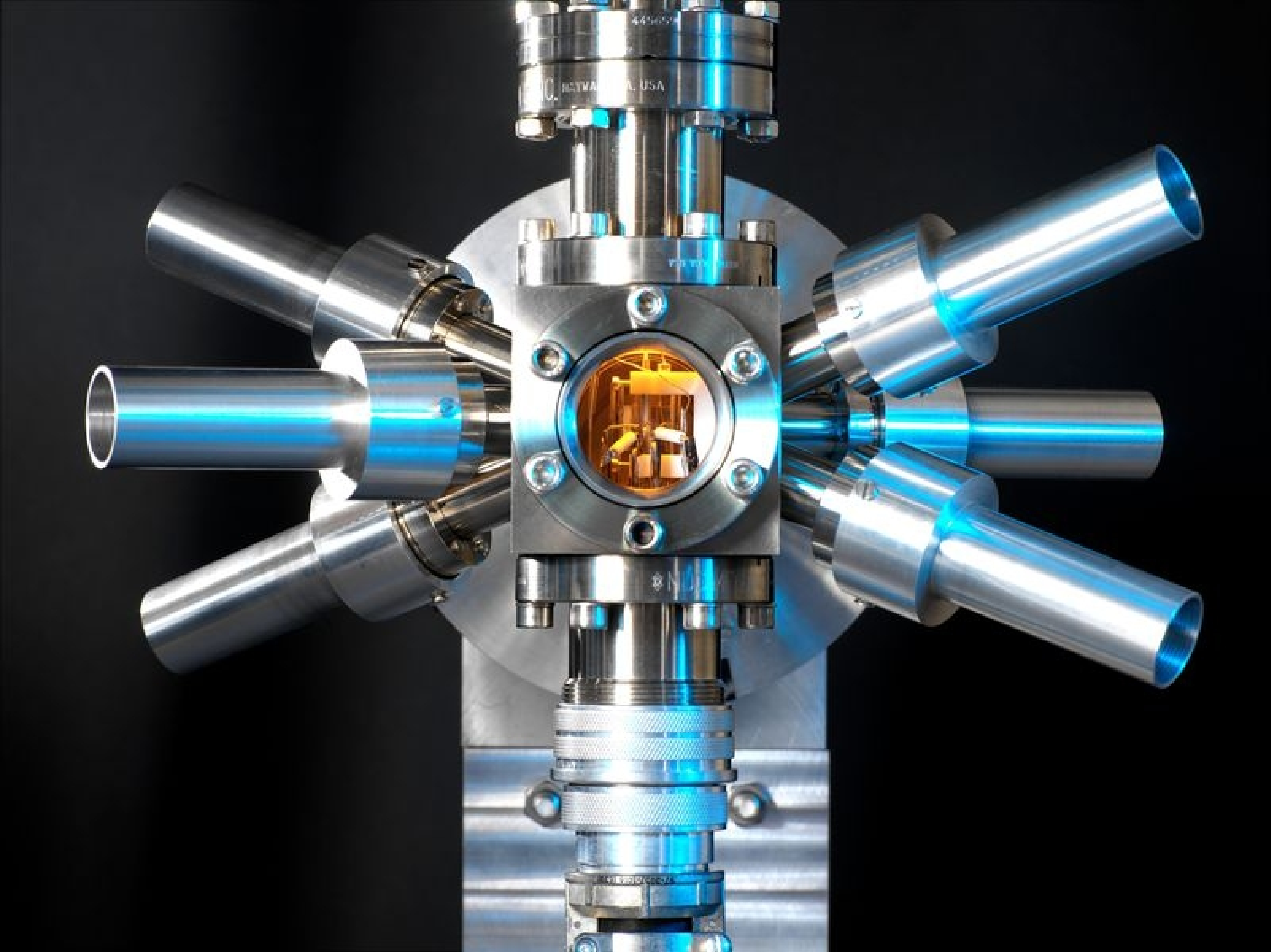
pointer to current data
Lock-free circular queue of preallocated elements of type data_type of length N=4
0
1
2
3
4
5
6
....
....
....
....
atomically allocate
atomic<int> left
atomic<int> right
copy
inst0.update([](data_type* data) {
//update the data
});
sync barrier (on left%N == 0)
atomic_data<data_type> inst0
update(CAS)
atomically return element that was current data

There is only enough space in memory to allocate at any one time.

Even when you reference & dereference, lock & free, you will always have that “gap” in time adverted from its original point.

For hyper-ledgers with consensus it works well. In data science with implied methods you will crash!

Atomics!



GCC is currently not going to invest the time, effort, and required code obfuscation to prevent speculative loads until such hardware exists.

Avoiding Speculation

Most forms of speculation involve introducing loads and/or stores on code paths which may not have executed a load or store to that location before. This new load or store may introduce a data race between threads which was not present before. This is not allowed in the standard, and these types of optimizations must be disabled. This applies to loads and stores of cross-thread visible data (shared memory) only, purely thread local optimizations are still allowed.

The Library interface

In order to satisfy the C++11 atomic functionality, GCC is looking to implement calls to all the required routines utilizing the following pattern:

First, provide a generic routine to handle objects of all sizes. This routine effectively treats the object as a 'slab of memory' and performs the operations on memory which is passed in by the caller.

```
  void __atomic_exchange (size_t
obj_size_in_bytes, void *mem,
const void *value, void
*return_value, enum memory_model
model)
```

A typical caller, such as the C++ atomic template for exchange would provide any required temporaries, relieving the library of memory allocation responsibilities.

```
template class <T>
T exchange (T *mem, T val,
memory_order m)
{
  T tmp;
  __atomic_exchange (sizeof (T),
mem, &val, &tmp, m);
  return tmp;
}
```

Second, also provide a set of optimized routines for 1, 2, 4, 8, and 16 bytes values which forego the slab of memory approach and pass the required information by value. These map to the sizes which different architectures may provide as atomic instruction sequences.

```
  I1  __atomic_exchange_1  (I1
*mem, I1 val, enum memory_model
model)
  I2  __atomic_exchange_2  (I2
*mem, I2 val, enum memory_model
model)
  I4  __atomic_exchange_4  (I4
*mem, I4 val, enum memory_model
model)
  I8  __atomic_exchange_8  (I8
*mem, I8 val, enum memory_model
model)
  I16 __atomic_exchange_16 (I16
*mem, I16 val, enum memory_model
model)
```

The following accumulation transformation:

```
for (p = q; p; p = p->next)
  if (p->data > 0) ++count;
```

can no longer be transformed to:

```
int tmp = count;

for (p = q; p; p = p->next)
  if (p->data > 0) ++tmp;
count = tmp;
```

The rationale is that in the case of q == NULL, the new transformation performs a store of “count = count” at the end of the loop. There was no store executed by this snippet before the transformation, and another thread could be counting on this and write a different value into count when q == NULL. This first thread could now overwrite that value with the original value of count. This was not possible in the program before, thus the semantics have been changed and a new data race introduced by the store.

The transformation could be allowed if it were transformed into:

```
int tmp = 0;


for (p = q; p; p = p->next)
 if (p->data > 0) ++tmp; if (tmp
!= 0) count += tmp;
```

Disallowing speculative loads will impact a number of optimizations, including scheduling, common subexpression elimination, etc. The effect is you can no longer move a load above a control flow construct if there wasn't already a load on that path. Note again that this is only loads of **shared** memory. Local variables do not have these restrictions, assuming their addresses never escape.

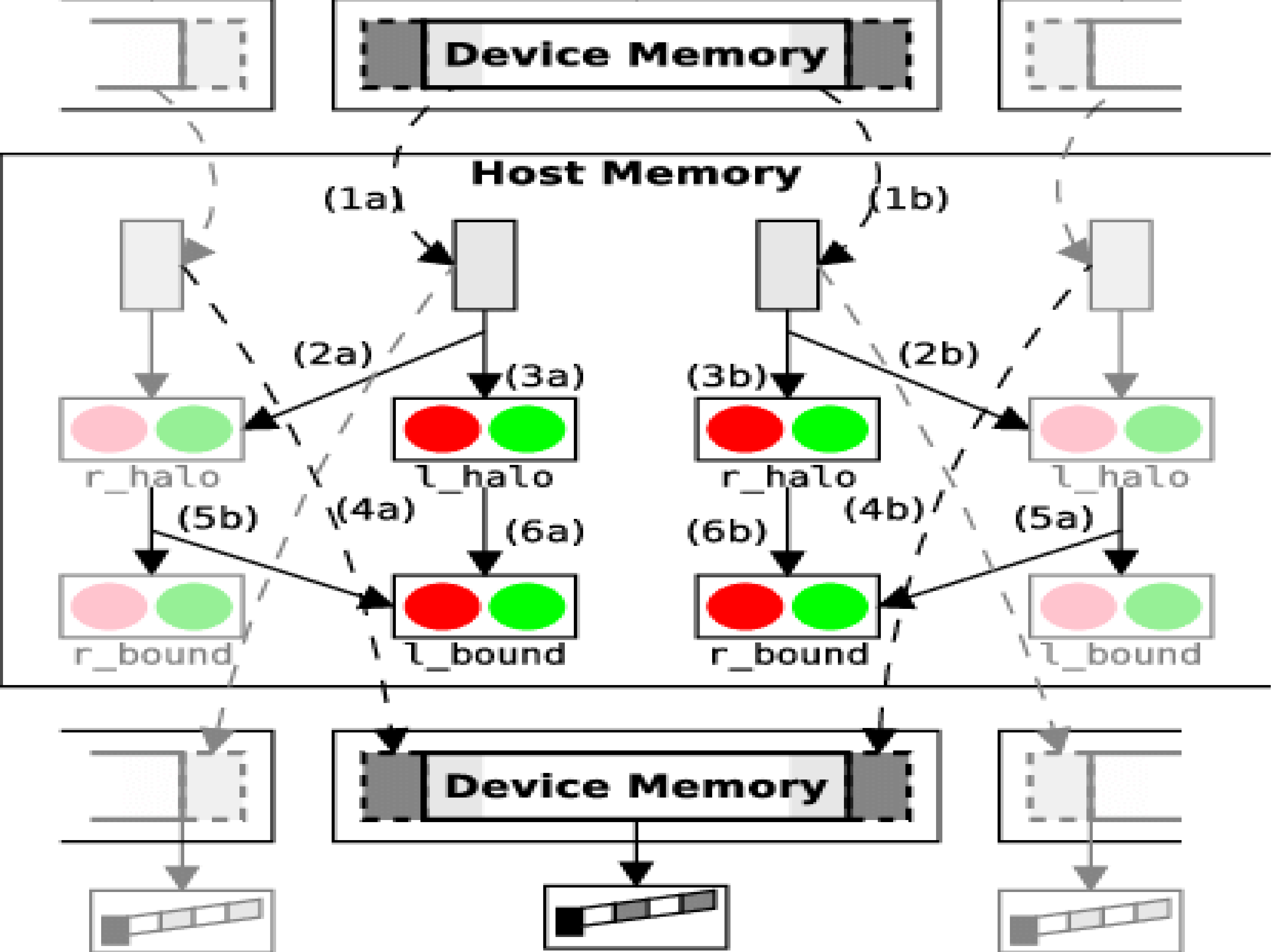
```
  If (x)
     val = global * 2;
```

can no longer be transformed to

```
  reg = global;
  if (x)
   val = reg * 2;
```

unless there is already a load of 'global' on the path leading to the 'if'.

Prev : Executive Summary
Next : Atomic Types


Device Memory
Host Memory
(1a)
(1b)
(2a)
(2b)
(3a)
(3b)
r_halo
l_halo
r_halo
l_halo
(5b)
(4a)
(6a)
(6b)
(4b)
(5a)
r_bound
l_bound
r_bound
l_bound
Device Memory

32.x Atomic operations library

.4 Order & Consistency

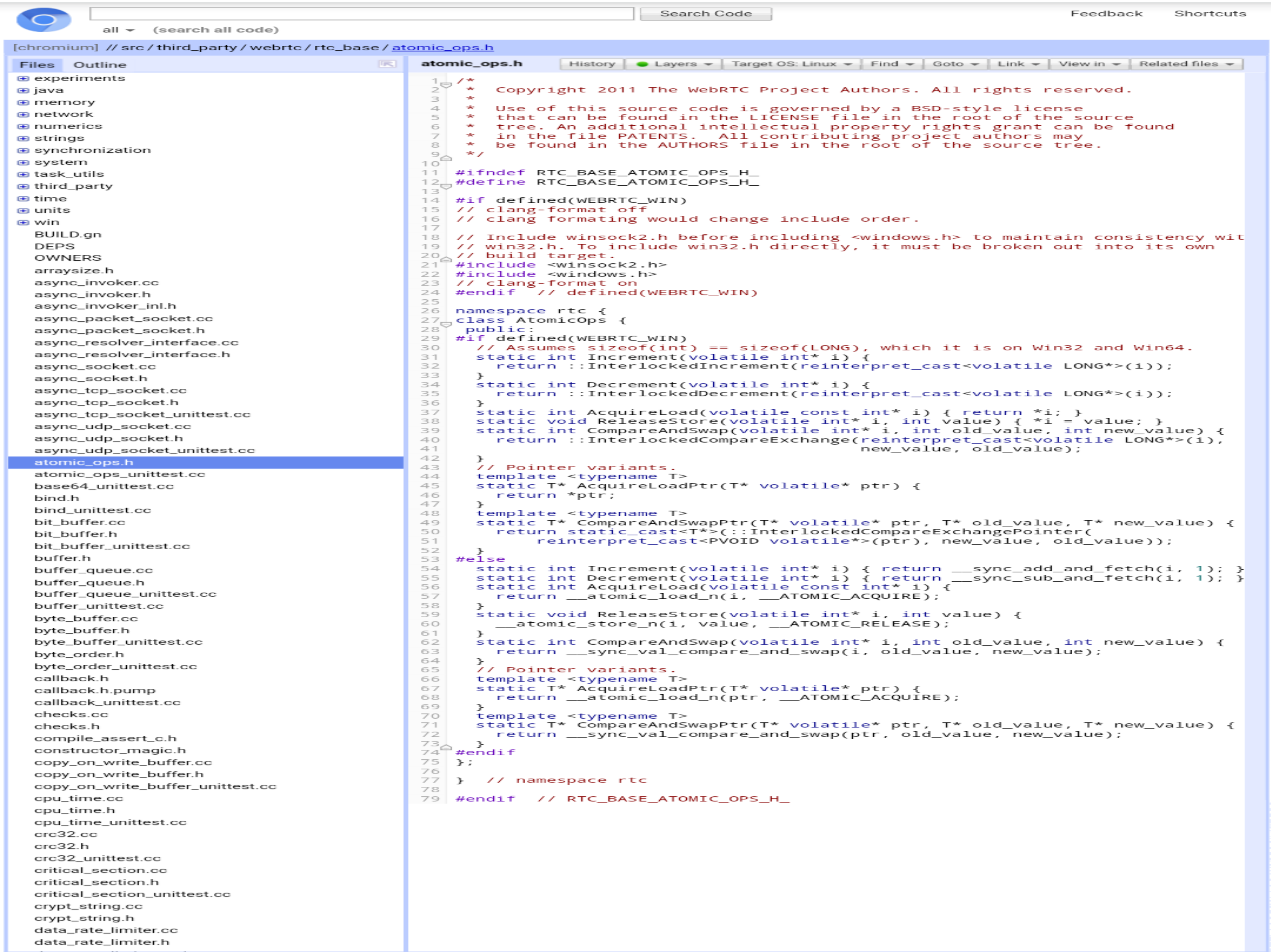
.5 Lock-free Property

.6 Atomic Types

.7 Operations on Atomic T

. 8 Flag Type & Operation

.9 Fences



[chromium] // src / third_party / webrtc / rtc_base / atomic_ops.h

experiments
java
memory
network
numerics
strings
synchronization
system
task_utils
third_party
time
units
win
BUILD.gn
DEPS
OWNERS
arraysize.h
async_invoker.cc
async_invoker.h
async_invoker_inl.h
async_packet_socket.cc
async_packet_socket.h
async_resolver_interface.cc
async_resolver_interface.h
async_socket.cc
async_socket.h
async_tcp_socket.cc
async_tcp_socket.h
async_tcp_socket_unittest.cc
async_udp_socket.cc
async_udp_socket.h
async_udp_socket_unittest.cc
atomic_ops.h
atomic_ops_unittest.cc
base64_unittest.cc
bind.h
bind_unittest.cc
bit_buffer.cc
bit_buffer.h
bit_buffer_unittest.cc
buffer.h
buffer_queue.cc
buffer_queue.h
buffer_queue_unittest.cc
buffer_unittest.cc
byte_buffer.cc
byte_buffer.h
byte_buffer_unittest.cc
byte_order.h
byte_order_unittest.cc
callback.h
callback.h.pump
callback_unittest.cc
checks.cc
checks.h
compile_assert_c.h
constructor_magic.h
copy_on_write_buffer.cc
copy_on_write_buffer.h
copy_on_write_buffer_unittest.cc
cpu_time.cc
cpu_time.h
cpu_time_unittest.cc
crc32.cc
crc32.h
crc32_unittest.cc
critical_section.cc
critical_section.h
critical_section_unittest.cc
crypt_string.cc
crypt_string.h
data_rate_limiter.cc
data_rate_limiter.h

```
/*
 *  Copyright 2011 The WebRTC Project Authors. All rights reserved.
 *
 *  Use of this source code is governed by a BSD-style license
 *  that can be found in the LICENSE file in the root of the source
 *  tree. An additional intellectual property rights grant can be found
 *  in the file PATENTS.  All contributing project authors may
 *  be found in the AUTHORS file in the root of the source tree.
 */

#ifndef RTC_BASE_ATOMIC_OPS_H_
#define RTC_BASE_ATOMIC_OPS_H_

#if defined(WEBRTC_WIN)
// clang-format off
// clang formating would change include order.

// Include winsock2.h before including <windows.h> to maintain consistency wit
// win32.h. To include win32.h directly, it must be broken out into its own
// build target.
#include <winsock2.h>
#include <windows.h>
// clang-format on
#endif  // defined(WEBRTC_WIN)

namespace rtc {
class AtomicOps {
 public:
#if defined(WEBRTC_WIN)
  // Assumes sizeof(int) == sizeof(LONG), which it is on Win32 and Win64.
  static int Increment(volatile int* i) {
    return ::InterlockedIncrement(reinterpret_cast<volatile LONG*>(i));
  }
  static int Decrement(volatile int* i) {
    return ::InterlockedDecrement(reinterpret_cast<volatile LONG*>(i));
  }
  static int AcquireLoad(volatile const int* i) { return *i; }
  static void ReleaseStore(volatile int* i, int value) { *i = value; }
  static int CompareAndSwap(volatile int* i, int old_value, int new_value) {
    return ::InterlockedCompareExchange(reinterpret_cast<volatile LONG*>(i),
                                        new_value, old_value);
  }
  // Pointer variants.
  template <typename T>
  static T* AcquireLoadPtr(T* volatile* ptr) {
    return *ptr;
  }
  template <typename T>
  static T* CompareAndSwapPtr(T* volatile* ptr, T* old_value, T* new_value) {
    return static_cast<T*>(::InterlockedCompareExchangePointer(
        reinterpret_cast<PVOID volatile*>(ptr), new_value, old_value));
  }
#else
  static int Increment(volatile int* i) { return __sync_add_and_fetch(i, 1); }
  static int Decrement(volatile int* i) { return __sync_sub_and_fetch(i, 1); }
  static int AcquireLoad(volatile const int* i) {
    return __atomic_load_n(i, __ATOMIC_ACQUIRE);
  }
  static void ReleaseStore(volatile int* i, int value) {
    __atomic_store_n(i, value, __ATOMIC_RELEASE);
  }
  static int CompareAndSwap(volatile int* i, int old_value, int new_value) {
    return __sync_val_compare_and_swap(i, old_value, new_value);
  }
  // Pointer variants.
  template <typename T>
  static T* AcquireLoadPtr(T* volatile* ptr) {
    return __atomic_load_n(ptr, __ATOMIC_ACQUIRE);
  }
  template <typename T>
  static T* CompareAndSwapPtr(T* volatile* ptr, T* old_value, T* new_value) {
    return __sync_val_compare_and_swap(ptr, old_value, new_value);
  }
#endif
};

}  // namespace rtc

#endif  // RTC_BASE_ATOMIC_OPS_H_
```

Join GitHub today

GitHub is home to over 40 million developers working together to host and review code, manage projects, and build software together.

Sign up

Dism

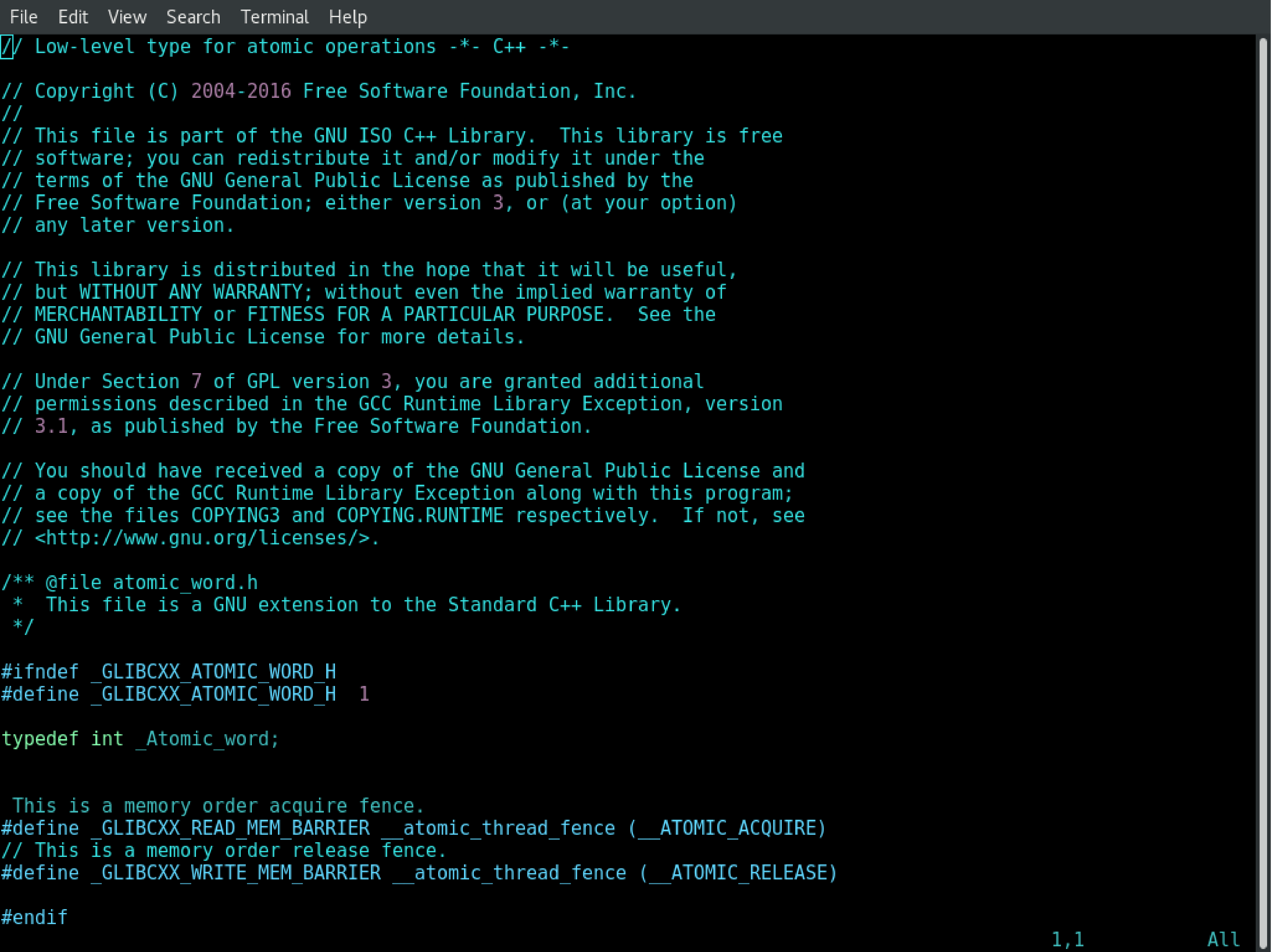
: master ▾ STL / stl / src / atomic.cpp Find file Cop

tephanTLavavej Initial commit. 2195148 15 days

tributor

ines (28 sloc) | 982 Bytes Raw Blame History

```cpp
// Copyright (c) Microsoft Corporation.
// SPDX-License-Identifier: Apache-2.0 WITH LLVM-exception

// implement shared_ptr spin lock

#include <yvals.h>

#include <intrin.h>
#pragma warning(disable : 4793)

_EXTERN_C

// SPIN LOCK FOR shared_ptr ATOMIC OPERATIONS
volatile long _Shared_ptr_flag;

_CRTIMP2_PURE void __cdecl _Lock_shared_ptr_spin_lock() { // spin until _Shared_ptr_flag successfully set
#ifdef _M_ARM
    while (_InterlockedExchange_acq(&_Shared_ptr_flag, 1)) {
        __yield();
    }
#else // _M_ARM
    while (_interlockedbittestandset(&_Shared_ptr_flag, 0)) { // set bit 0
    }
#endif // _M_ARM
}

_CRTIMP2_PURE void __cdecl _Unlock_shared_ptr_spin_lock() { // release previously obtained lock
#ifdef _M_ARM
    __dmb(_ARM_BARRIER_ISH);
    __iso_volatile_store32((volatile int*) &_Shared_ptr_flag, 0);
#else // _M_ARM
    _interlockedbittestandreset(&_Shared_ptr_flag, 0); // reset bit 0
#endif // _M_ARM
}

_END_EXTERN_C
```



File Edit View Search Terminal Help

```
// Low-level type for atomic operations -*- C++ -*-

// Copyright (C) 2004-2016 Free Software Foundation, Inc.
//
// This file is part of the GNU ISO C++ Library.  This library is free
// software; you can redistribute it and/or modify it under the
// terms of the GNU General Public License as published by the
// Free Software Foundation; either version 3, or (at your option)
// any later version.

// This library is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
// GNU General Public License for more details.

// Under Section 7 of GPL version 3, you are granted additional
// permissions described in the GCC Runtime Library Exception, version
// 3.1, as published by the Free Software Foundation.

// You should have received a copy of the GNU General Public License and
// a copy of the GCC Runtime Library Exception along with this program;
// see the files COPYING3 and COPYING.RUNTIME respectively.  If not, see
// <http://www.gnu.org/licenses/>.

/** @file atomic_word.h
 *  This file is a GNU extension to the Standard C++ Library.
 */

#ifndef _GLIBCXX_ATOMIC_WORD_H
#define _GLIBCXX_ATOMIC_WORD_H  1

typedef int _Atomic_word;


 This is a memory order acquire fence.
#define _GLIBCXX_READ_MEM_BARRIER __atomic_thread_fence (__ATOMIC_ACQUIRE)
// This is a memory order release fence.
#define _GLIBCXX_WRITE_MEM_BARRIER __atomic_thread_fence (__ATOMIC_RELEASE)

#endif
                                                                                1,1           All
```

To demand guarantee’s on proofs for infinite counts based on tread pooling and/or multi-threading is == NULL

Remember this()

Cause a data race as you are

in data proofs

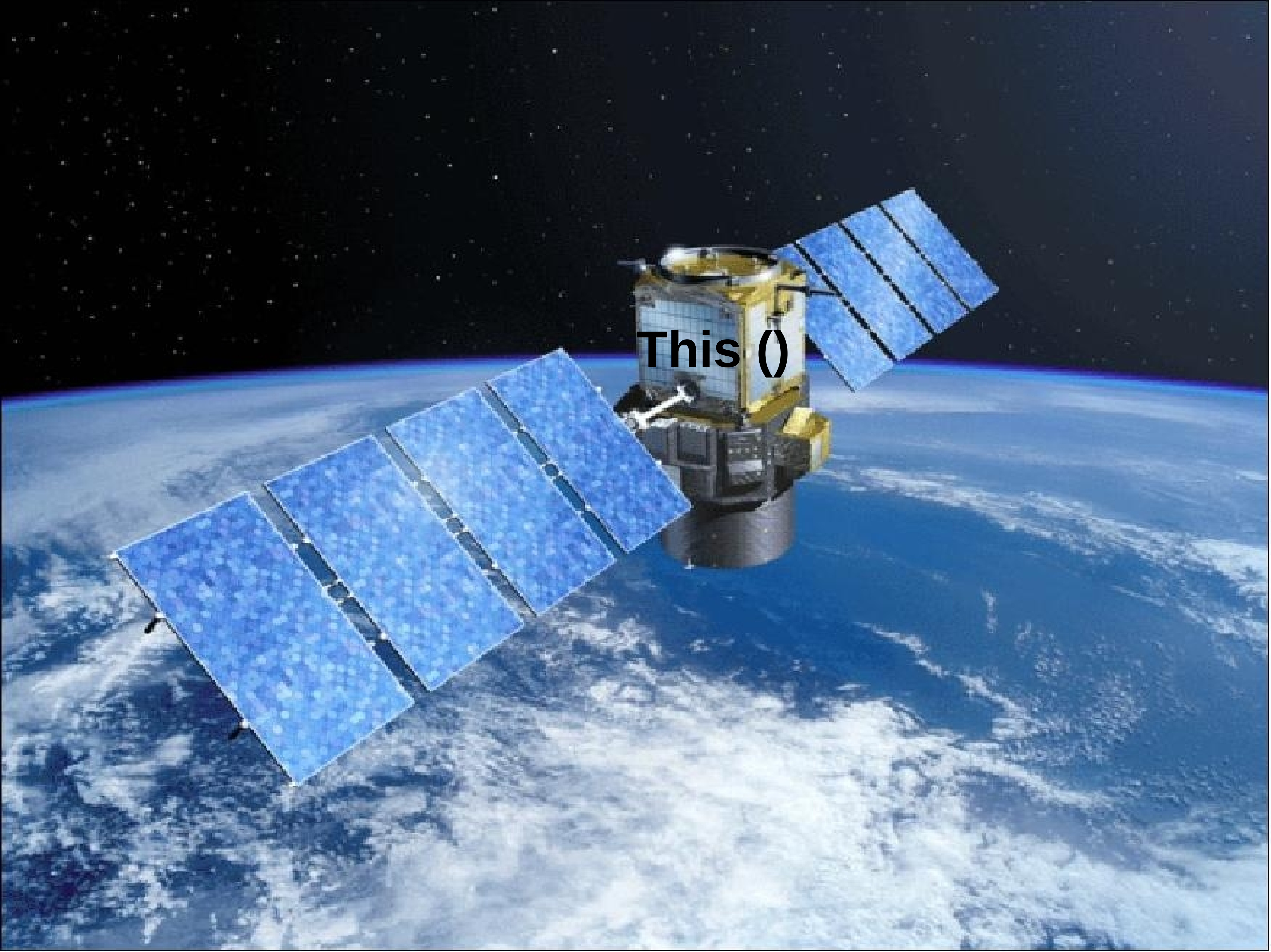
or

our minds & world

We will send this()

& disprove all your data

& implement with all of our proofs.



This ()

To be continued…

Twisted Computing
(float-it-AllWayRound)
(><0000.0000><)

[github.com/conceptslearningmachine/Slides]

Copyright 2013 Thomas Catalano’s Intellectual Property