align

invisalign® | iTero®

Анатомия Thread Sanitizer

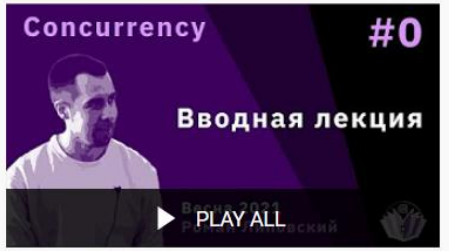
Алексей Веселовский

Align Technology

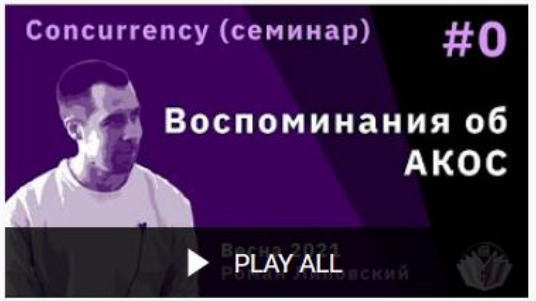
C++ Russia 2021

Материалы

https://youtu.be/V2_80g0eOMc



Теория и практика многопоточной синхронизации (лекции) (2 курс, весна 2021), лектор - Роман Липовский



Теория и практика многопоточной синхронизации (семинары) (2 курс, весна 2021), семинарист - Роман Липовский

https://www.youtube.com/playlist?list=PL4_hYwCyhAvaxKQHe6n8JQcoc7tWxKWRL

https://www.youtube.com/playlist?list=PL4_hYwCyhAvb7P8guwSTaaUS8EcOaWjxF

Зачем?

Зачем?

Потоки (aka Threads), общая память. Сложно.

Зачем?

Потоки (aka Threads), общая память. Сложно.

Зачем потоки?

Зачем?

Потоки (aka Threads), общая память. Сложно.

Производительность и асинхронность.

Зачем потоки?

Зачем?

Потоки (aka Threads), общая память. Сложно.

Зачем потоки?

Производительность и асинхронность.

За счет чего производительность?

Зачем?

Потоки (aka Threads), общая память. Сложно.

Зачем потоки?

Производительность и асинхронность.

За счет чего производительность?

Многоядерность/многопроцессорность.

Зачем?

Потоки (aka Threads), общая память. Сложно.

Зачем потоки?

Производительность и асинхронность.

За счет чего производительность?

Многоядерность/многопроцессорность.

Почему сложно?

Зачем?

Потоки (aka Threads), общая память. Сложно.

Зачем потоки?

Производительность и асинхронность.

За счет чего производительность?

Многоядерность/многопроцессорность.

Почему сложно?

По сути – распределенная система.

Зачем?

Потоки (aka Threads), общая память. Сложно.

Зачем потоки?

Производительность и асинхронность.

За счет чего производительность?

Многоядерность/многопроцессорность.

Почему сложно?

По сути – распределенная система.
Распределенная система пытающаяся работать быстро.

Зачем?

Потоки (aka Threads), общая память. Сложно.

Зачем потоки?

Производительность и асинхронность.

За счет чего производительность?

Многоядерность/многопроцессорность.

Почему сложно?

По сути – распределенная система.
Распределенная система пытающаяся работать быстро.
С кэшами и буферами.

Зачем?

Потоки (aka Threads), общая память. Сложно.

Зачем потоки?

Производительность и асинхронность.

За счет чего производительность?

Многоядерность/многопроцессорность.

Почему сложно?

По сути – распределенная система.
Распределенная система пытающаяся работать быстро.
С кэшами и буферами.
Разные системы имеют разную архитектуру.

Зачем?

Потоки (aka Threads), общая память. Сложно.

Зачем потоки?

Производительность и асинхронность.

За счет чего производительность?

Многоядерность/многопроцессорность.

Почему сложно?

По сути – распределенная система.
Распределенная система пытающаяся работать быстро.
С кэшами и буферами.
Разные системы имеют разную архитектуру.
И методы оптимизации.

Зачем?

Потоки (aka Threads), общая память. Сложно.

Зачем потоки?

Производительность и асинхронность.

За счет чего производительность?

Многоядерность/многопроцессорность.

Почему сложно?

По сути – распределенная система.
Распределенная система пытающаяся работать быстро.
С кэшами и буферами.
Разные системы имеют разную архитектуру.
И методы оптимизации.
А мы хотим портабельный код.

Зачем?

Потоки (aka Threads), общая память. Сложно.

Зачем потоки?

Производительность и асинхронность.

За счет чего производительность?

Многоядерность/многопроцессорность.

Почему сложно?

По сути – распределенная система.
Распределенная система пытающаяся работать быстро.
С кэшами и буферами.
Разные системы имеют разную архитектуру.
И методы оптимизации.
А мы хотим портабельный код. Без ошибок.

Зачем?

Потоки (aka Threads), общая память. Сложно.

Зачем потоки?

Производительность и асинхронность.

За счет чего производительность?

Многоядерность/многопроцессорность.

Почему сложно?

По сути – распределенная система.
Распределенная система пытающаяся работать быстро.
С кэшами и буферами.
Разные системы имеют разную архитектуру.
И методы оптимизации.
А мы хотим портабельный код. Без ошибок.

TSan не поможет тебе

Зачем?

И методы оптимизации.

А мы хотим портабельный код. Без ошибок.

TSan не поможет тебе

Зачем?

И методы оптимизации.

А мы хотим портабельный код. Без ошибок.

TSan не поможет тебе

Тебе нужен ЯП с моделью памяти

Зачем?

И методы оптимизации.

А мы хотим портабельный код. Без ошибок.

TSan не поможет тебе

Тебе нужен ЯП с моделью памяти

И уже потом TSan

Модель Памяти

Модель Памяти

1

```
volatile int a = 42;
void foo() {a = 77;}
int main()
{
    std::thread th(foo);
    std::cout << a << std::endl;
    th.join();
    return 0;
}
```

2

```
volatile int a = 42;
void foo() {a = 13;}
int main()
{
    std::thread th(foo);
    std::this_thread::sleep_for(1000ms);
    std::cout << a << std::endl;
    th.join();
    return 0;
}
```

3

```
int a = 42;
int main()
{
    std::shared_ptr<int> ptr(new int);
    std::thread th([=](){*ptr = a; a = 13;});
    std::this_thread::sleep_for(1000ms);
    std::weak_ptr<int> wp(ptr);
    ptr.reset();
    if (wp.expired())
        a++;
    th.join();
    return 0;
}
```

4

```
int a = 42;
int main()
{
    std::shared_ptr<int> ptr(new int);
    std::thread th([=](){a = 13;});
    std::this_thread::sleep_for(1000ms);
    std::weak_ptr<int> wp(ptr);
    ptr.reset();
    if (wp.expired())
        a++;
    th.join();
    return 0;
}
```

Модель Памяти

1996

1

```
volatile int a = 42;
void foo() {a = 77;}
int main()
{
    std::thread th(foo);
    std::cout << a << std::endl;
    th.join();
    return 0;
}
```

2

```
volatile int a = 42;
void foo() {a = 13;}
int main()
{
    std::thread th(foo);
    std::this_thread::sleep_for(1000ms);
    std::cout << a << std::endl;
    th.join();
    return 0;
}
```

3

```
int a = 42;
int main()
{
    std::shared_ptr<int> ptr(new int);
    std::thread th([=](){*ptr = a; a = 13;});
    std::this_thread::sleep_for(1000ms);
    std::weak_ptr<int> wp(ptr);
    ptr.reset();
    if (wp.expired())
        a++;
    th.join();
    return 0;
}
```

4

```
int a = 42;
int main()
{
    std::shared_ptr<int> ptr(new int);
    std::thread th([=](){a = 13;});
    std::this_thread::sleep_for(1000ms);
    std::weak_ptr<int> wp(ptr);
    ptr.reset();
    if (wp.expired())
        a++;
    th.join();
    return 0;
}
```

Модель Памяти

1996

2004

1

```
volatile int a = 42;
void foo() {a = 77;}
int main()
{
    std::thread th(foo);
    std::cout << a << std::endl;
    th.join();
    return 0;
}
```

2

```
volatile int a = 42;
void foo() {a = 13;}
int main()
{
    std::thread th(foo);
    std::this_thread::sleep_for(1000ms);
    std::cout << a << std::endl;
    th.join();
    return 0;
}
```

3

```
int a = 42;
int main()
{
    std::shared_ptr<int> ptr(new int);
    std::thread th([=](){*ptr = a; a = 13;});
    std::this_thread::sleep_for(1000ms);
    std::weak_ptr<int> wp(ptr);
    ptr.reset();
    if (wp.expired())
        a++;
    th.join();
    return 0;
}
```

4

```
int a = 42;
int main()
{
    std::shared_ptr<int> ptr(new int);
    std::thread th([=](){a = 13;});
    std::this_thread::sleep_for(1000ms);
    std::weak_ptr<int> wp(ptr);
    ptr.reset();
    if (wp.expired())
        a++;
    th.join();
    return 0;
}
```

Модель Памяти

1996 

2004 

1

```
volatile int a = 42;
void foo() {a = 77;}
int main()
{
    std::thread th(foo);
    std::cout << a << std::endl;
    th.join();
    return 0;
}
```

2

```
volatile int a = 42;
void foo() {a = 13;}
int main()
{
    std::thread th(foo);
    std::this_thread::sleep_for(1000ms);
    std::cout << a << std::endl;
    th.join();
    return 0;
}
```

3

```
int a = 42;
int main()
{
    std::shared_ptr<int> ptr(new int);
    std::thread th([=](){*ptr = a; a = 13;});
    std::this_thread::sleep_for(1000ms);
    std::weak_ptr<int> wp(ptr);
    ptr.reset();
    if (wp.expired())
        a++;
    th.join();
    return 0;
}
```

4

```
int a = 42;
int main()
{
    std::shared_ptr<int> ptr(new int);
    std::thread th([=](){a = 13;});
    std::this_thread::sleep_for(1000ms);
    std::weak_ptr<int> wp(ptr);
    ptr.reset();
    if (wp.expired())
        a++;
    th.join();
    return 0;
}
```

Модель Памяти

1996 

2004 

2009 

1
```
volatile int a = 42;
void foo() {a = 77;}
int main()
{
    std::thread th(foo);
    std::cout << a << std::endl;
    th.join();
    return 0;
}
```

2
```
volatile int a = 42;
void foo() {a = 13;}
int main()
{
    std::thread th(foo);
    std::this_thread::sleep_for(1000ms);
    std::cout << a << std::endl;
    th.join();
    return 0;
}
```

3
```
int a = 42;
int main()
{
    std::shared_ptr<int> ptr(new int);
    std::thread th([=](){*ptr = a; a = 13;});
    std::this_thread::sleep_for(1000ms);
    std::weak_ptr<int> wp(ptr);
    ptr.reset();
    if (wp.expired())
        a++;
    th.join();
    return 0;
}
```

4
```
int a = 42;
int main()
{
    std::shared_ptr<int> ptr(new int);
    std::thread th([=](){a = 13;});
    std::this_thread::sleep_for(1000ms);
    std::weak_ptr<int> wp(ptr);
    ptr.reset();
    if (wp.expired())
        a++;
    th.join();
    return 0;
}
```

Модель Памяти

1996 

2004 

2009 

2011 

1
```cpp
volatile int a = 42;
void foo() {a = 77;}
int main()
{
    std::thread th(foo);
    std::cout << a << std::endl;
    th.join();
    return 0;
}
```

2
```cpp
volatile int a = 42;
void foo() {a = 13;}
int main()
{
    std::thread th(foo);
    std::this_thread::sleep_for(1000ms);
    std::cout << a << std::endl;
    th.join();
    return 0;
}
```

3
```cpp
int a = 42;
int main()
{
    std::shared_ptr<int> ptr(new int);
    std::thread th([=](){*ptr = a; a = 13;});
    std::this_thread::sleep_for(1000ms);
    std::weak_ptr<int> wp(ptr);
    ptr.reset();
    if (wp.expired())
        a++;
    th.join();
    return 0;
}
```

4
```cpp
int a = 42;
int main()
{
    std::shared_ptr<int> ptr(new int);
    std::thread th([=](){a = 13;});
    std::this_thread::sleep_for(1000ms);
    std::weak_ptr<int> wp(ptr);
    ptr.reset();
    if (wp.expired())
        a++;
    th.join();
    return 0;
}
```

Модель Памяти

1996 

2004 

2009 

2011 

2014 

1

```
volatile int a = 42;
void foo() {a = 77;}
int main()
{
    std::thread th(foo);
    std::cout << a << std::endl;
    th.join();
    return 0;
}
```

2

```
volatile int a = 42;
void foo() {a = 13;}
int main()
{
    std::thread th(foo);
    std::this_thread::sleep_for(1000ms);
    std::cout << a << std::endl;
    th.join();
    return 0;
}
```

3

```
int a = 42;
int main()
{
    std::shared_ptr<int> ptr(new int);
    std::thread th([=](){*ptr = a; a = 13;});
    std::this_thread::sleep_for(1000ms);
    std::weak_ptr<int> wp(ptr);
    ptr.reset();
    if (wp.expired())
        a++;
    th.join();
    return 0;
}
```

4

```
int a = 42;
int main()
{
    std::shared_ptr<int> ptr(new int);
    std::thread th([=](){a = 13;});
    std::this_thread::sleep_for(1000ms);
    std::weak_ptr<int> wp(ptr);
    ptr.reset();
    if (wp.expired())
        a++;
    th.join();
    return 0;
}
```

Модель Памяти

1996 

2004 

2009 

2011 

2014 

2015 

1
```
volatile int a = 42;
void foo() {a = 77;}
int main()
{
    std::thread th(foo);
    std::cout << a << std::endl;
    th.join();
    return 0;
}
```

2
```
volatile int a = 42;
void foo() {a = 13;}
int main()
{
    std::thread th(foo);
    std::this_thread::sleep_for(1000ms);
    std::cout << a << std::endl;
    th.join();
    return 0;
}
```

3
```
int a = 42;
int main()
{
    std::shared_ptr<int> ptr(new int);
    std::thread th([=](){*ptr = a; a = 13;});
    std::this_thread::sleep_for(1000ms);
    std::weak_ptr<int> wp(ptr);
    ptr.reset();
    if (wp.expired())
        a++;
    th.join();
    return 0;
}
```

4
```
int a = 42;
int main()
{
    std::shared_ptr<int> ptr(new int);
    std::thread th([=](){a = 13;});
    std::this_thread::sleep_for(1000ms);
    std::weak_ptr<int> wp(ptr);
    ptr.reset();
    if (wp.expired())
        a++;
    th.join();
    return 0;
}
```

Модель Памяти

1

```
volatile int a = 42;
void foo() {a = 77;}
int main()
{
    std::thread th(foo);
    std::cout << a << std::endl;
    th.join();
    return 0;
}
```

2

```
volatile int a = 42;
void foo() {a = 13;}
int main()
{
    std::thread th(foo);
    std::this_thread::sleep_for(1000ms);
    std::cout << a << std::endl;
    th.join();
    return 0;
}
```

3

```
int a = 42;
int main()
{
    std::shared_ptr<int> ptr(new int);
    std::thread th([=](){*ptr = a; a = 13;});
    std::this_thread::sleep_for(1000ms);
    std::weak_ptr<int> wp(ptr);
    ptr.reset();
    if (wp.expired())
        a++;
    th.join();
    return 0;
}
```

4

```
int a = 42;
int main()
{
    std::shared_ptr<int> ptr(new int);
    std::thread th([=](){a = 13;});
    std::this_thread::sleep_for(1000ms);
    std::weak_ptr<int> wp(ptr);
    ptr.reset();
    if (wp.expired())
        a++;
    th.join();
    return 0;
}
```

1996 

2004 

2009 

2011 

2014 

2015 

2017 

Модель Памяти

1

```
volatile int a = 42;
void foo() {a = 77;}
int main()
{
    std::thread th(foo);
    std::cout << a << std::endl;
    th.join();
    return 0;
}
```

2

```
volatile int a = 42;
void foo() {a = 13;}
int main()
{
    std::thread th(foo);
    std::this_thread::sleep_for(1000ms);
    std::cout << a << std::endl;
    th.join();
    return 0;
}
```

3

```
int a = 42;
int main()
{
    std::shared_ptr<int> ptr(new int);
    std::thread th([=](){*ptr = a; a = 13;});
    std::this_thread::sleep_for(1000ms);
    std::weak_ptr<int> wp(ptr);
    ptr.reset();
    if (wp.expired())
        a++;
    th.join();
    return 0;
}
```

4

```
int a = 42;
int main()
{
    std::shared_ptr<int> ptr(new int);
    std::thread th([=](){a = 13;});
    std::this_thread::sleep_for(1000ms);
    std::weak_ptr<int> wp(ptr);
    ptr.reset();
    if (wp.expired())
        a++;
    th.join();
    return 0;
}
```

1996 

2004 

2009 

2011 

2014 

2015 

2017 

2020 

Модель Памяти

Ответ на главный вопрос:

Модель Памяти

Ответ на главный вопрос: как гарантировать, что чтение увидит нужную запись?

Модель Памяти

Ответ на главный вопрос: как гарантировать, что чтение увидит нужную запись?

Можно было бы потребовать гарантии глобального упорядочивания всего, но...

Модель Памяти

Ответ на главный вопрос: как гарантировать, что чтение увидит нужную запись?

Можно было бы потребовать гарантии глобального упорядочивания всего, но...

мы хотим быстро!

Модель Памяти

Два типа ячеек памяти:

Модель Памяти

Два типа ячеек памяти:

1. Обычные – переупорядочивания разрешены, глобального порядка нет

Модель Памяти

Два типа ячеек памяти:

1. Обычные - переупорядочивания разрешены, глобального порядка нет
2. Атомарные – переупорядочивания запрещены, глобальный порядок есть (слабые модели памяти не рассматриваем)

Модель Памяти

Два типа ячеек памяти:

1. Обычные - переупорядочивания разрешены, глобального порядка нет
2. Атомарные – переупорядочивания запрещены, глобальный порядок есть (слабые модели памяти не рассматриваем).

Чтение атомика увидит последнюю предшествующую (в глобальном порядке, aka **synchronization order**) запись в этот же атомик.

Модель Памяти

Свяжем обращения к обычным ячейкам памяти с обращениями с атомиками: **program order**

Модель Памяти

Свяжем обращения к обычным ячейкам памяти с обращениями с атомиками: **program order**

Program order – порядок следования обращений к памяти в *тексте программы*

Модель Памяти

Свяжем обращения к обычным ячейкам памяти с обращениями с атомиками: **program order**

Program order – порядок следования обращений к памяти в *тексте программы*

Synchronization order – глобальный порядок обращений к атомика при *исполнении программы*

Модель Памяти

Свяжем обращения к обычным ячейкам памяти с обращениями с атомиками: **program order**

Program order – порядок следования обращений к памяти в *тексте программы*

Synchronization order – глобальный порядок обращений к атомика при *исполнении программы*

Атомики и обычные обращения к памяти существуют в одном тексте программы, между ними есть **program order**

Модель Памяти

Program order – порядок следования обращений к памяти в *тексте программы*

Synchronization order – глобальный порядок обращений к атомика при *исполнении программы*

Модель Памяти

Program order – порядок следования обращений к памяти в *тексте программы*

Synchronization order – глобальный порядок обращений к атомику при *исполнении программы*

Happens-before – наблюдаемая программой причинность

Модель Памяти

```
std::atomic<bool> ready(false);
std::vector<int> vec;
```

```
void thread1() {
    for (int i=0; i<100500; ++i)
        vec.push_back(i);
    ready.store(true);
}
```

```
void thread2() {
    if (ready.load())
        std::cout << vec.size();
}
```

Program order – порядок следования обращений к памяти в *тексте программы*
Synchronization order – глобальный порядок обращений к атомика при *исполнении программы*
Happens-before – наблюдаемая программой причинность

Модель Памяти

```
std::atomic<bool> ready(false);
std::vector<int> vec;
```

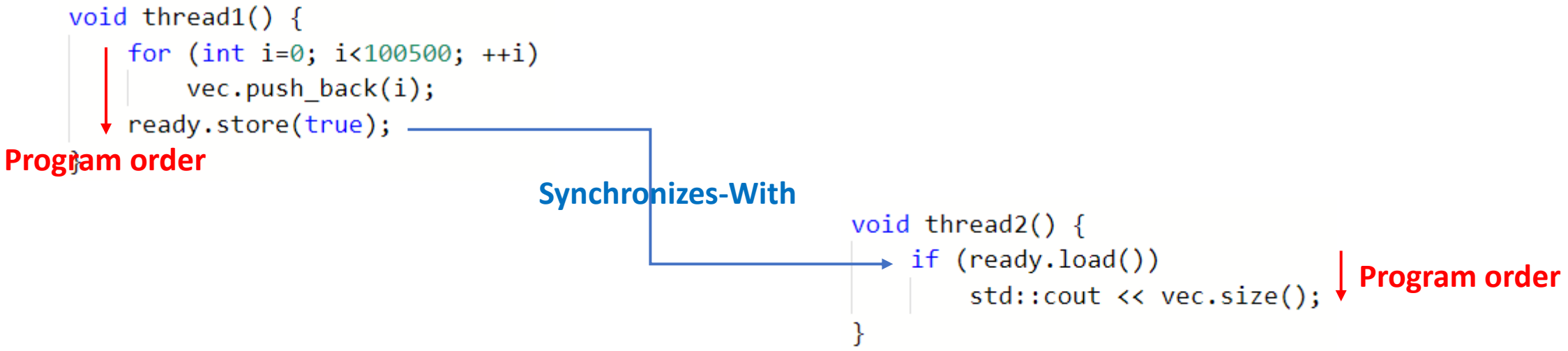
```
void thread1() {
    for (int i=0; i<100500; ++i)
        vec.push_back(i);
    ready.store(true);
}
```

```
void thread2() {
    if (ready.load())
        std::cout << vec.size();
}
```

Program order – порядок следования обращений к памяти в *тексте программы*
Synchronization order – глобальный порядок обращений к атомика при *исполнении программы*
Happens-before – наблюдаемая программой причинность
Synchronizes-With – если из атомика прочитали записанное туда значение из другого потока

Модель Памяти

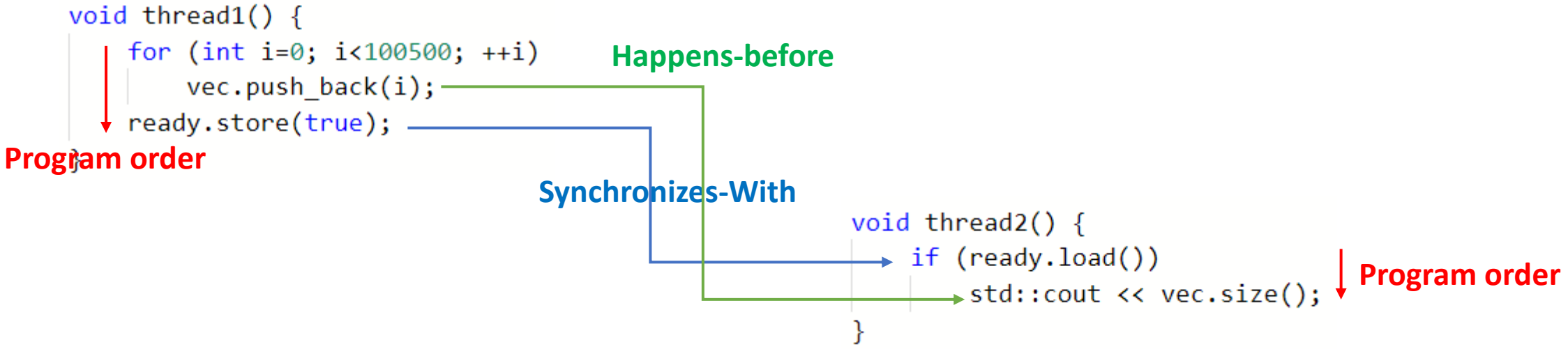
```
std::atomic<bool> ready(false);
std::vector<int> vec;
```



Program order – порядок следования обращений к памяти в *тексте программы*
Synchronization order – глобальный порядок обращений к атомика при *исполнении программы*
Happens-before – наблюдаемая программой причинность
Synchronizes-With – если из атомика прочитали записанное туда значение из другого потока

Модель Памяти

```
std::atomic<bool> ready(false);
std::vector<int> vec;
```



Program order – порядок следования обращений к памяти в *тексте программы*
Synchronization order – глобальный порядок обращений к атомика при *исполнении программы*
Happens-before – наблюдаемая программой причинность
Synchronizes-With – если из атомика прочитали записанное туда значение из другого потока

Модель Памяти

Data Race

Два обращения к памяти **конфликтуют** если

- они обращаются к одной и той же ячейке памяти
- по крайней мере одна из этих обращений - запись

Модель Памяти

Data Race

Два обращения к памяти **конфликтуют** если

- они обращаются к одной и той же ячейке памяти
- по крайней мере одна из этих обращений - запись

Гонка (data race) в исполнении – два конфликтующих неатомарных (не через атомик) обращения к памяти, которые не упорядочены через **happens-before**.

Модель Памяти

Data Race

Два обращения к памяти **конфликтуют** если
- они обращаются к одной и той же ячейке памяти
- по крайней мере одна из этих обращений - запись

Гонка (data race) в исполнении – два конфликтующих неатомарных (не через атомик) обращения к памяти, которые не упорядочены через **happens-before**.

Thread Sanitizer – Data Race detector

Модель Памяти

Data Race

Два обращения к памяти **конфликтуют** если
- они обращаются к одной и той же ячейке памяти
- по крайней мере одна из этих обращений - запись

Гонка (data race) в исполнении – два конфликтующих неатомарных (не через атомик) обращения к памяти, которые не упорядочены через **happens-before**.

Thread Sanitizer – Data Race detector

Thread Sanitizer – Data Race detector

Data Race

Два обращения к памяти **конфликтуют** если

- они обращаются к одной и той же ячейке памяти
- по крайней мере одна из этих обращений - запись

Гонка (data race) в исполнении – два конфликтующих неатомарных (не через атомик) обращения к памяти, которые не упорядочены через **happens-before**.

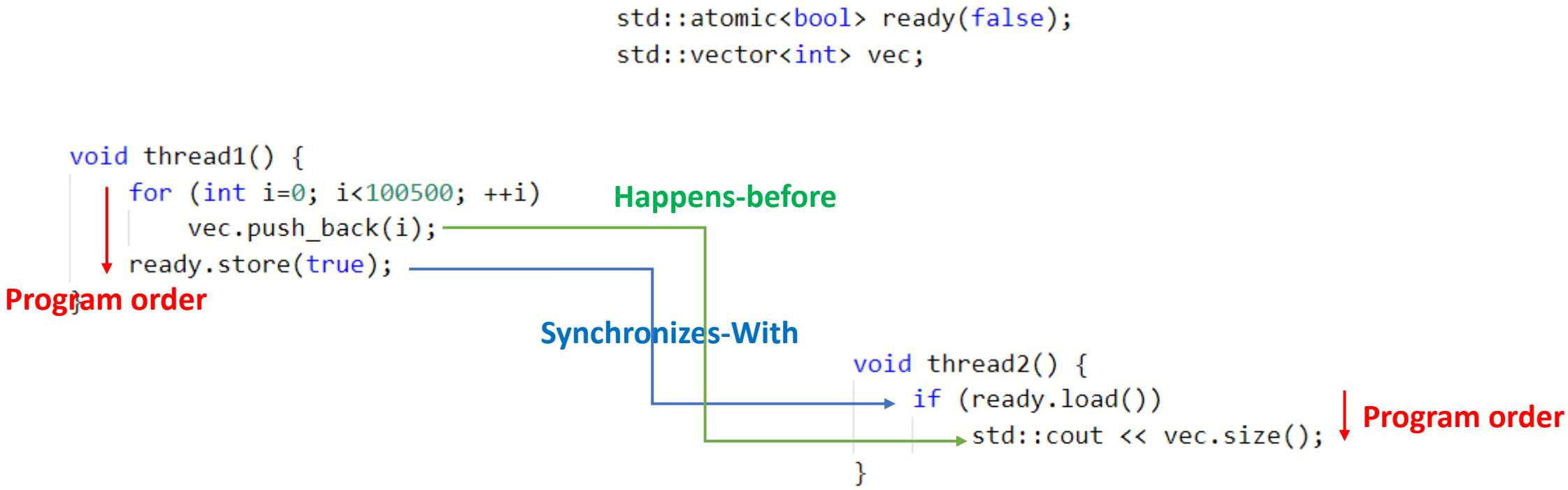
Детектирует Data Race прямо по определению

Thread Sanitizer – Data Race detector

ThreadSanitizer is a tool that detects data races. It consists of a compiler instrumentation module and a run-time library. Typical slowdown introduced by ThreadSanitizer is about **5x-15x**. Typical memory overhead introduced by ThreadSanitizer is about **5x-10x**. (https://clang.llvm.org/docs/ThreadSanitizer.html)

Thread Sanitizer – Data Race detector

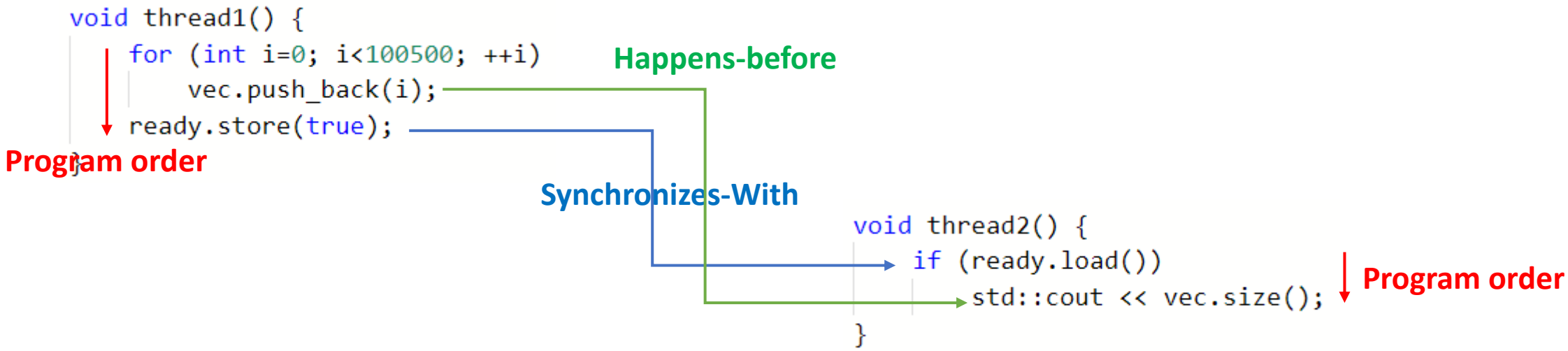
Как он это делает?



Thread Sanitizer – Data Race detector

Как он это делает?

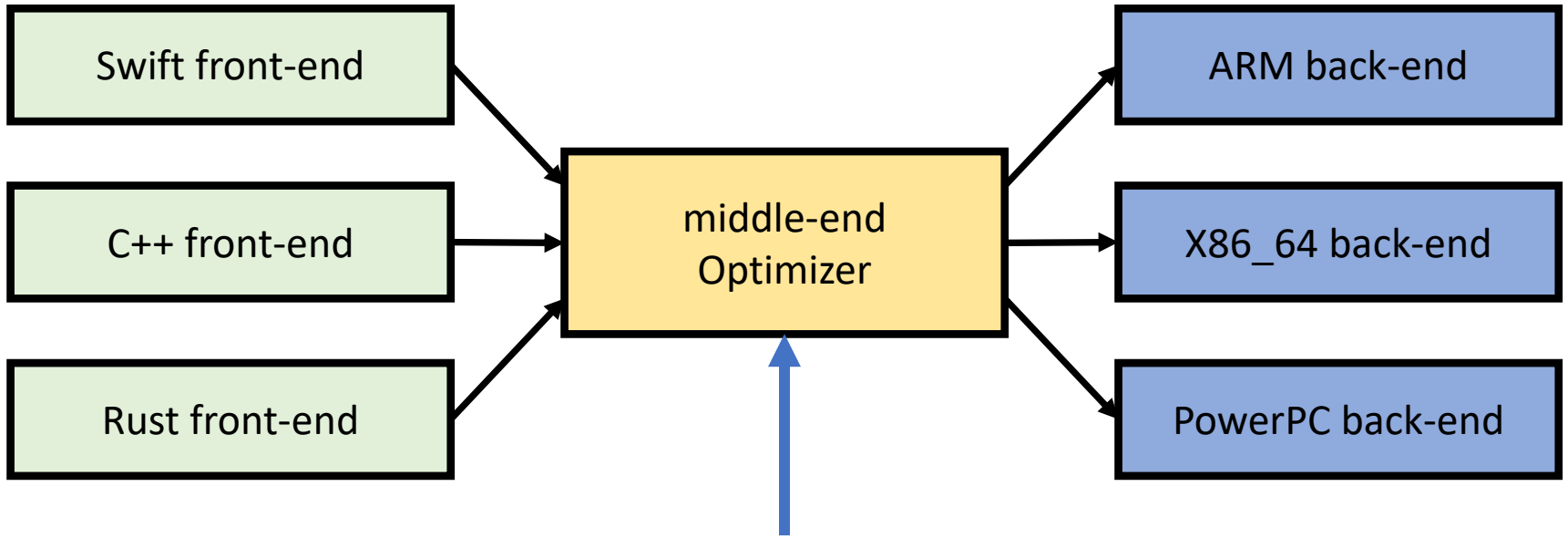
```
std::atomic<bool> ready(false);
std::vector<int> vec;
```



Не так!

Thread Sanitizer – Data Race detector

На фронтенде компилятора TSan не живёт



Sanitizer живет здесь

Thread Sanitizer – Data Race detector

Часы

Thread Sanitizer – Data Race detector

Логические часы

Thread Sanitizer – Data Race detector

Логические часы

- К каждому событию записи/чтения сопоставим временную метку T

Thread Sanitizer – Data Race detector

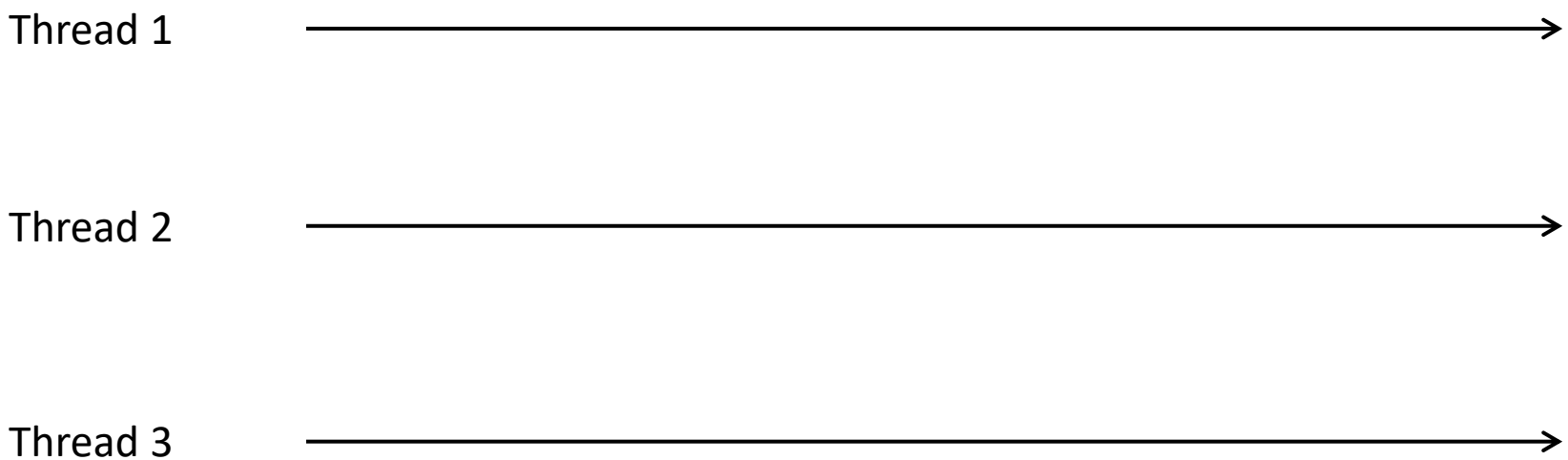
Логические часы

- К каждому событию записи/чтения сопоставим временную метку T
- Событие e1 предшествует событию e2 ⇔ T(e1) < T(e2)

Thread Sanitizer – Data Race detector

Логические ВЕКТОРНЫЕ часы

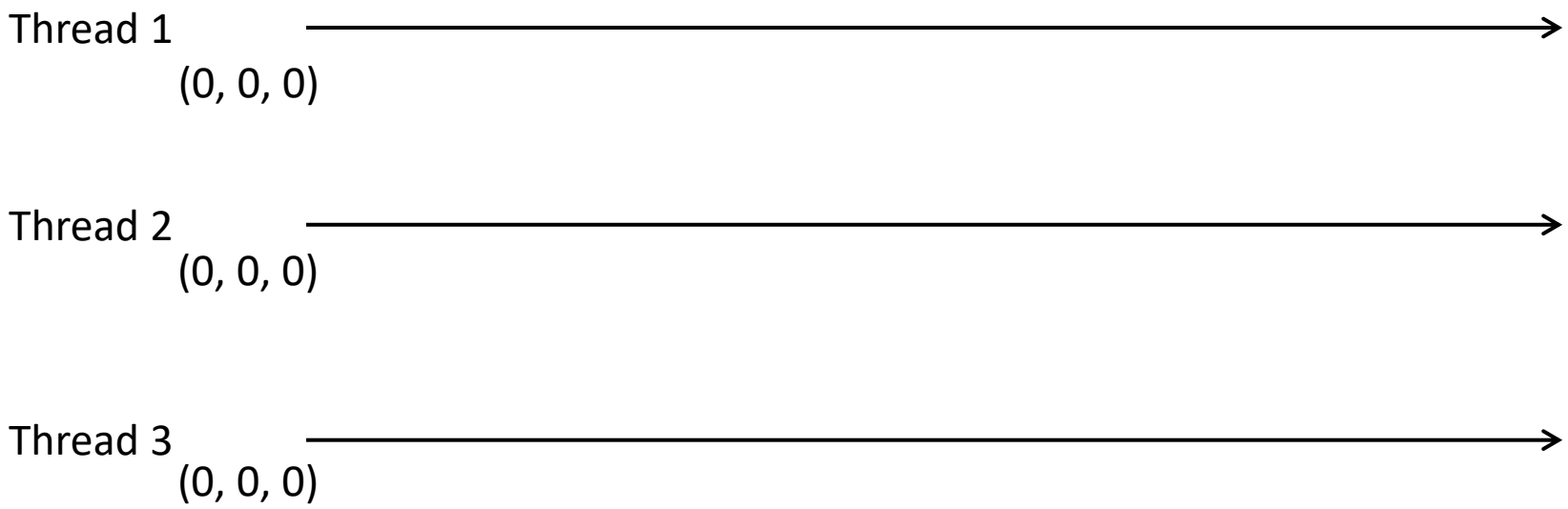
- К каждому событию записи/чтения сопоставим временную метку T
- Событие e1 предшествует событию e2 ⇔ T(e1) < T(e2)



Thread Sanitizer – Data Race detector

Логические ВЕКТОРНЫЕ часы

- К каждому событию записи/чтения сопоставим временную метку T
- Событие e1 предшествует событию e2 ⇔ T(e1) < T(e2)



Thread Sanitizer – Data Race detector

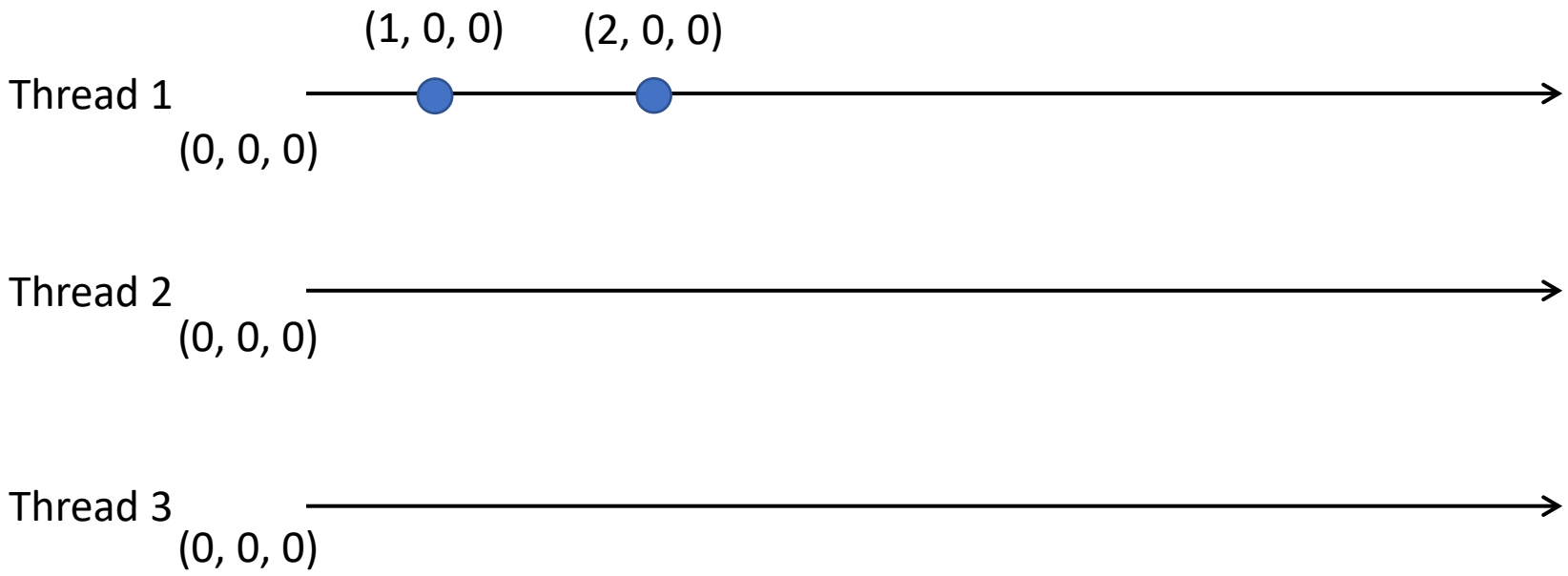
Логические ВЕКТОРНЫЕ часы

- К каждому событию записи/чтения сопоставим временную метку T
- Событие e1 предшествует событию e2 ⇔ T(e1) < T(e2)

Thread Sanitizer – Data Race detector

Логические ВЕКТОРНЫЕ часы

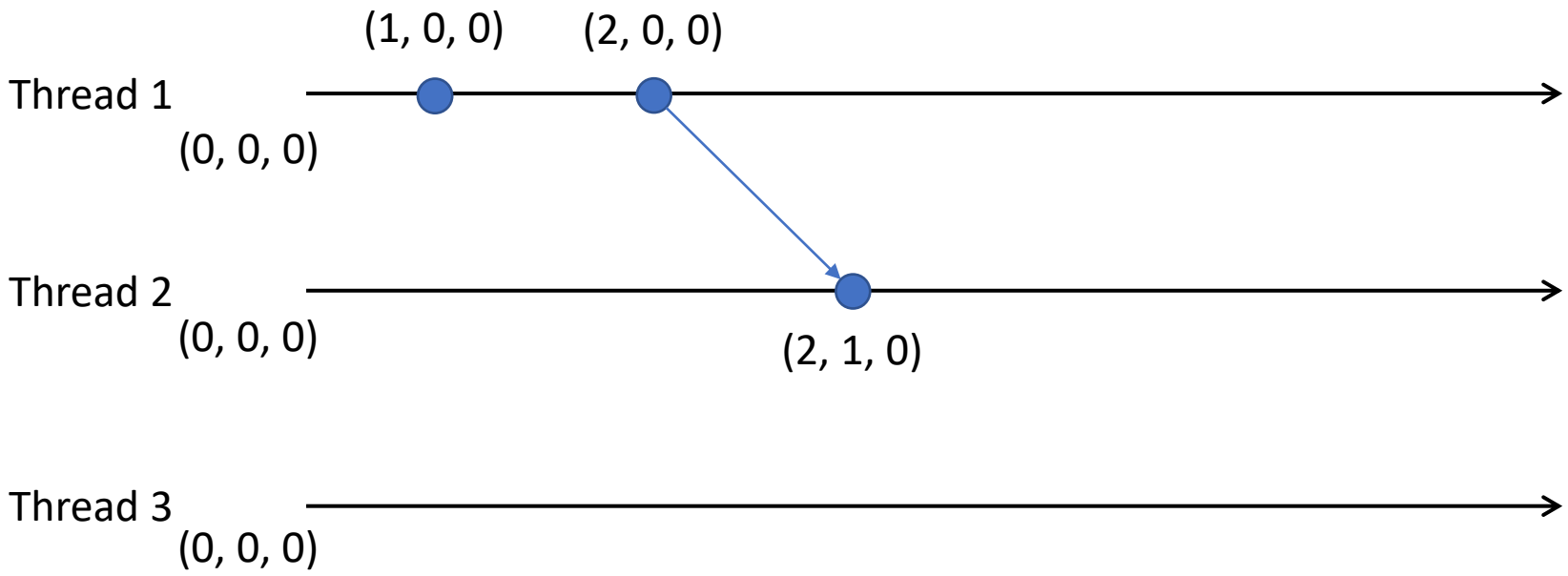
- К каждому событию записи/чтения сопоставим временную метку T
- Событие e1 предшествует событию e2 ⇔ T(e1) < T(e2)



Thread Sanitizer – Data Race detector

Логические ВЕКТОРНЫЕ часы

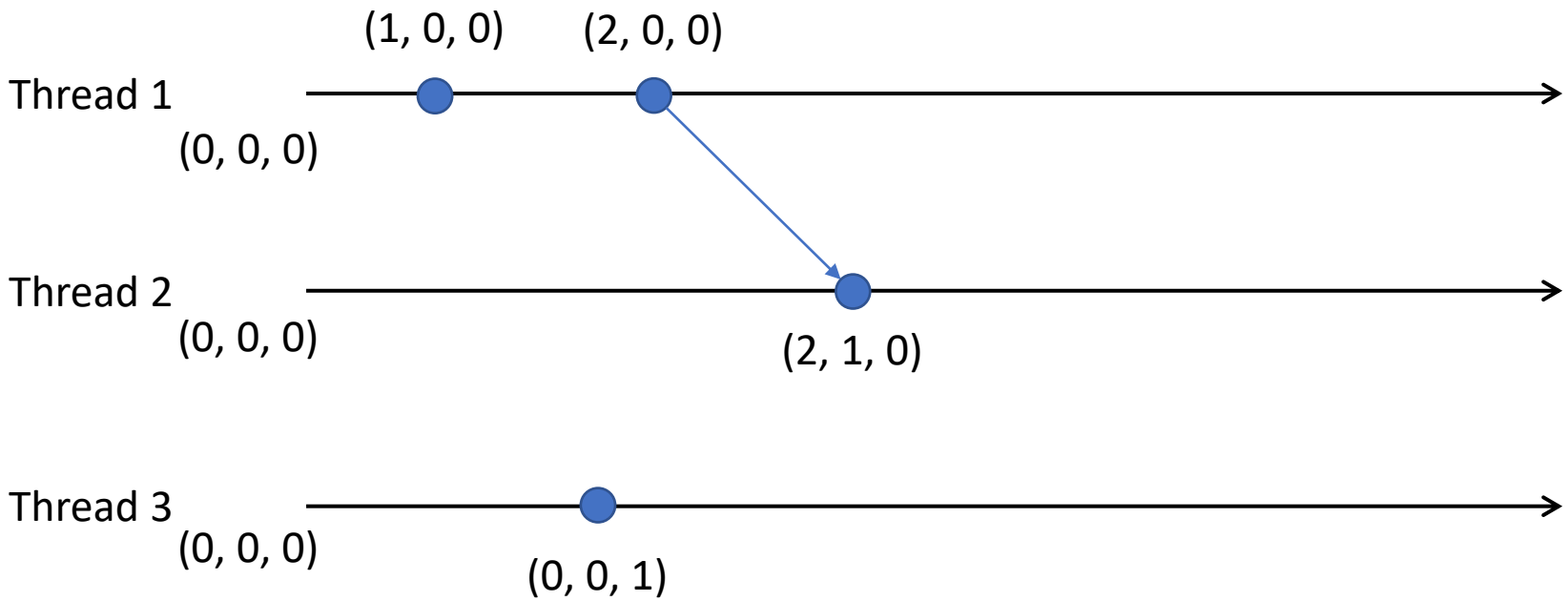
- К каждому событию записи/чтения сопоставим временную метку T
- Событие e1 предшествует событию e2 ⇔ T(e1) < T(e2)



Thread Sanitizer – Data Race detector

Логические ВЕКТОРНЫЕ часы

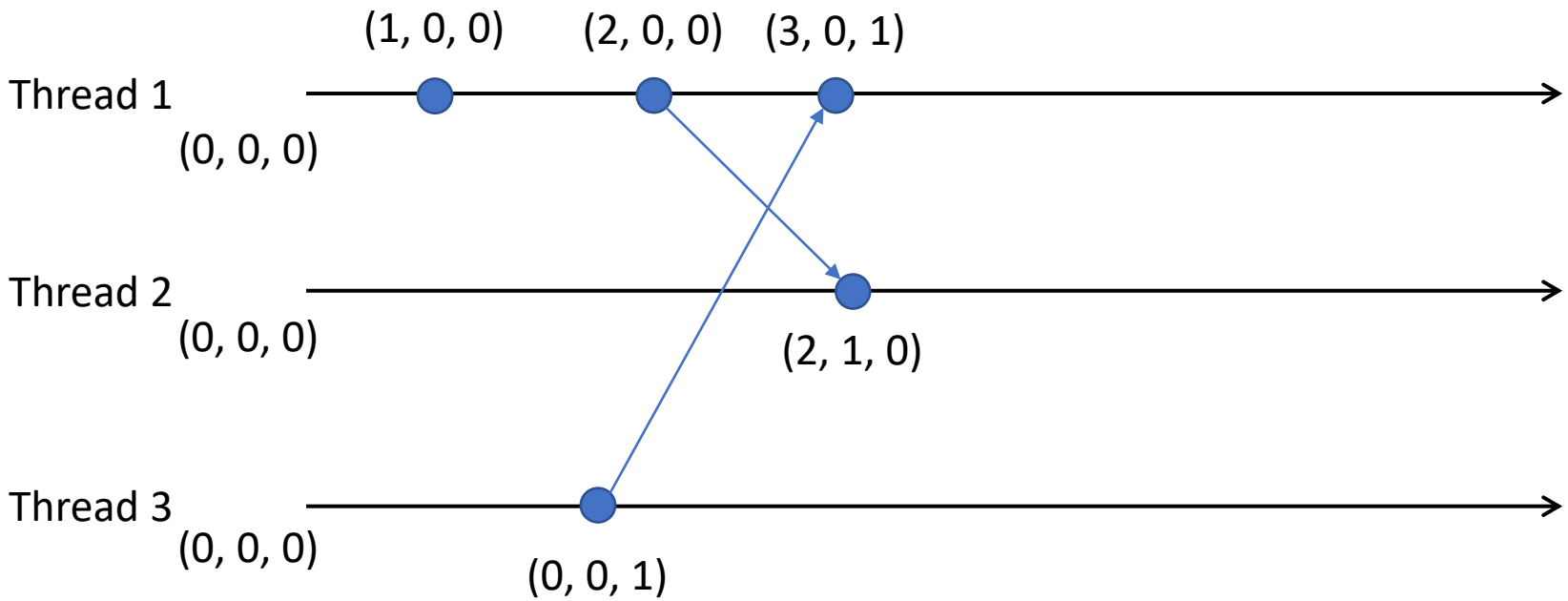
- К каждому событию записи/чтения сопоставим временную метку T
- Событие e1 предшествует событию e2 ⇔ T(e1) < T(e2)



Thread Sanitizer – Data Race detector

Логические ВЕКТОРНЫЕ часы

- К каждому событию записи/чтения сопоставим временную метку T
- Событие e1 предшествует событию e2 ⇔ T(e1) < T(e2)

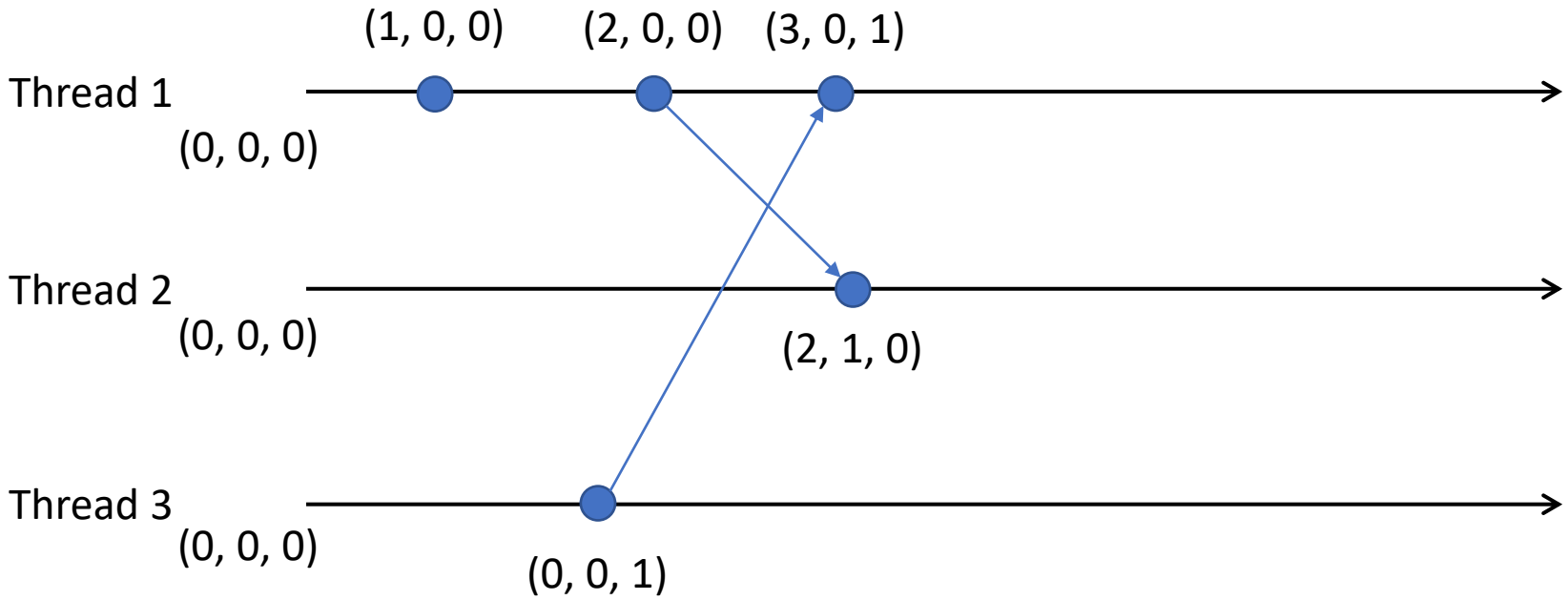


Thread Sanitizer – Data Race detector

Логические ВЕКТОРНЫЕ часы

- К каждому событию записи/чтения сопоставим временную метку T
- Событие e1 предшествует событию e2 ⇔ T(e1) < T(e2)

e1 на Th(i) может быть причиной e2 на Th(j) если T(e1)[i] <= T(e2)[i]

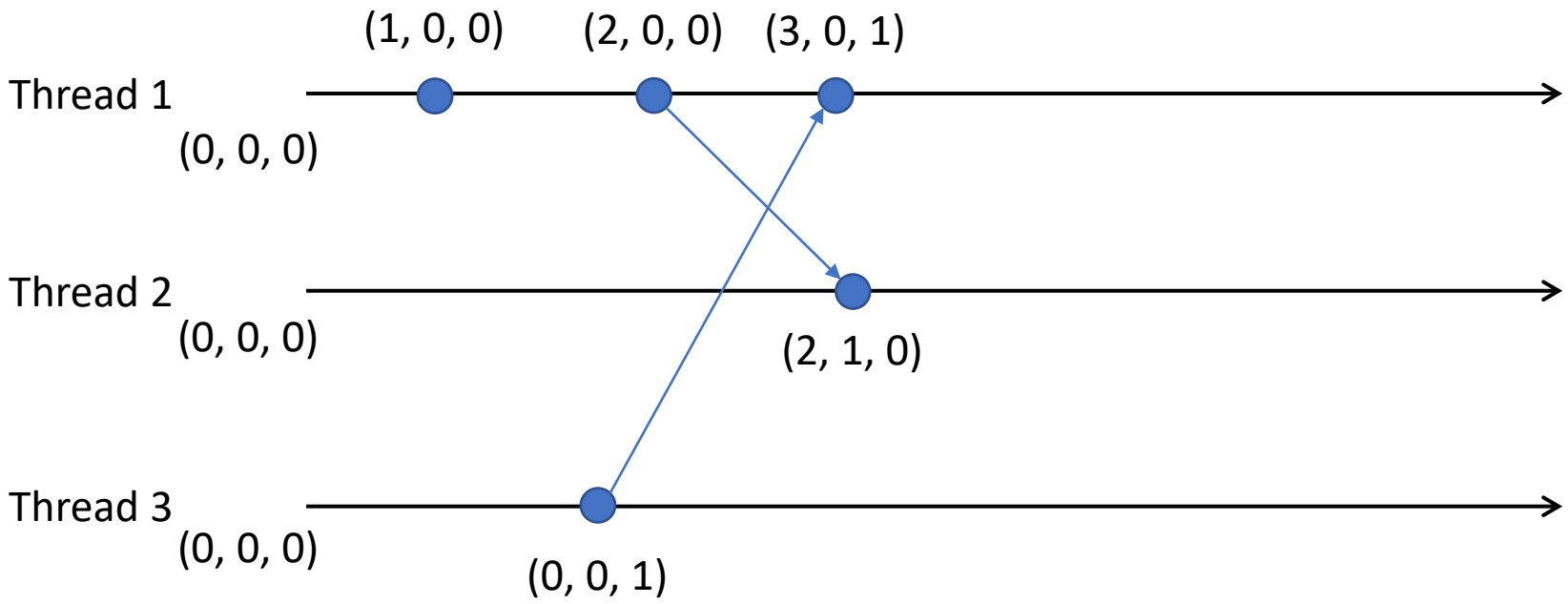


Thread Sanitizer – Data Race detector

Логические ВЕКТОРНЫЕ часы

- К каждому событию записи/чтения сопоставим временную метку T
- Событие e1 предшествует событию e2 ⇔ T(e1) < T(e2)

e1 на Th(i) может быть причиной e2 на Th(j) если T(e1)[i] <= T(e2)[i]



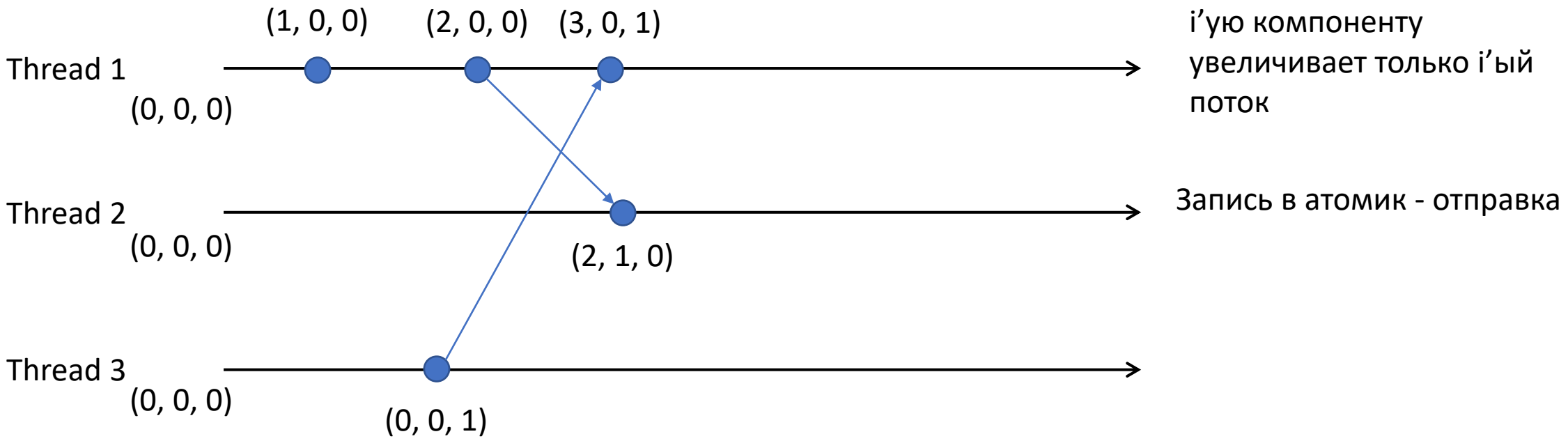
i'ую компоненту увеличивает только i'ый поток

Thread Sanitizer – Data Race detector

Логические ВЕКТОРНЫЕ часы

- К каждому событию записи/чтения сопоставим временную метку T
- Событие e1 предшествует событию e2 ⇔ T(e1) < T(e2)

e1 на Th(i) может быть причиной e2 на Th(j) если T(e1)[i] <= T(e2)[i]

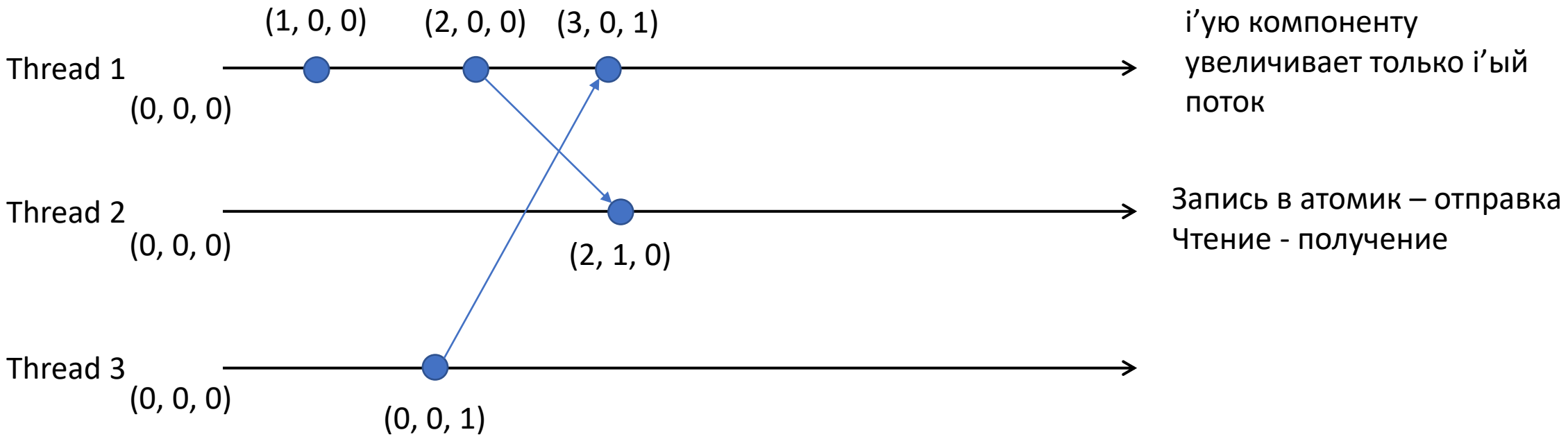


Thread Sanitizer – Data Race detector

Логические ВЕКТОРНЫЕ часы

- К каждому событию записи/чтения сопоставим временную метку T
- Событие e1 предшествует событию e2 ⇔ T(e1) < T(e2)

e1 на Th(i) может быть причиной e2 на Th(j) если T(e1)[i] <= T(e2)[i]



i'ую компоненту увеличивает только i'ый поток

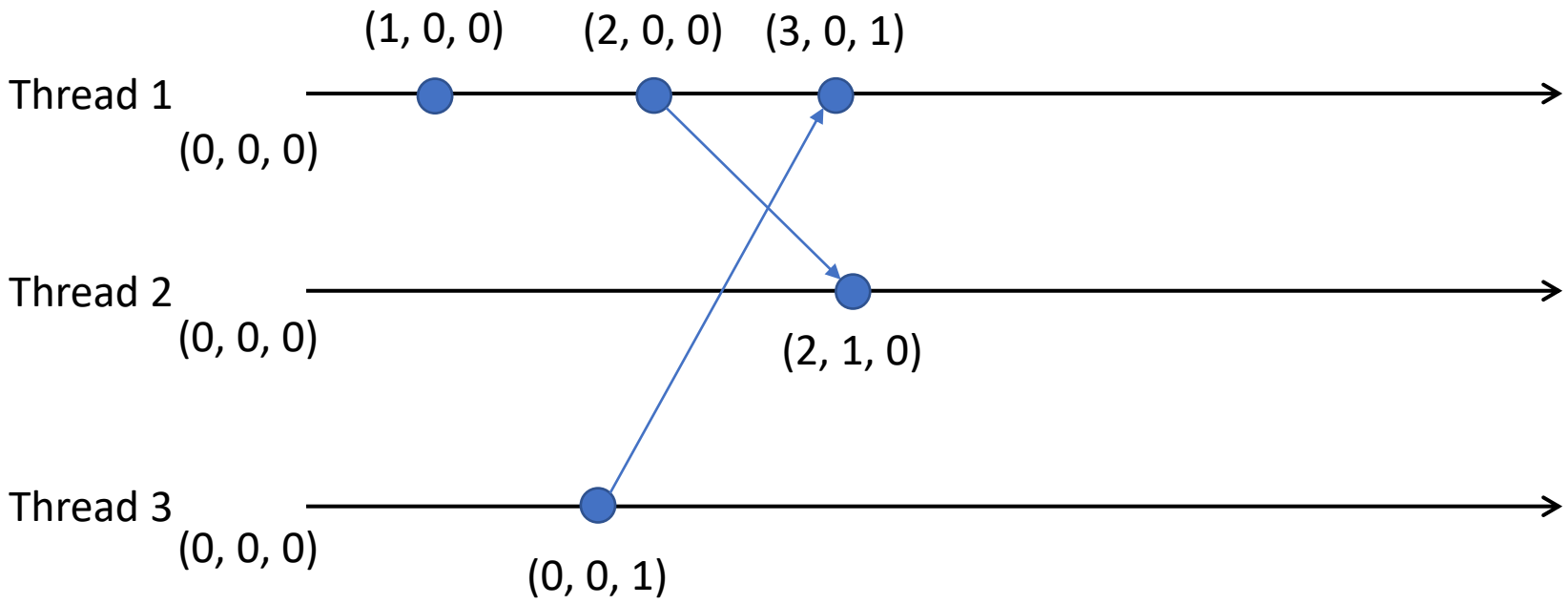
Запись в атомик – отправка
Чтение - получение

Thread Sanitizer – Data Race detector

Логические ВЕКТОРНЫЕ часы

- К каждому событию записи/чтения сопоставим временную метку T
- Событие e1 предшествует событию e2 ⇔ T(e1) < T(e2)

e1 на Th(i) может быть причиной e2 на Th(j) если T(e1)[i] <= T(e2)[i]



i'ую компоненту увеличивает только i'ый поток

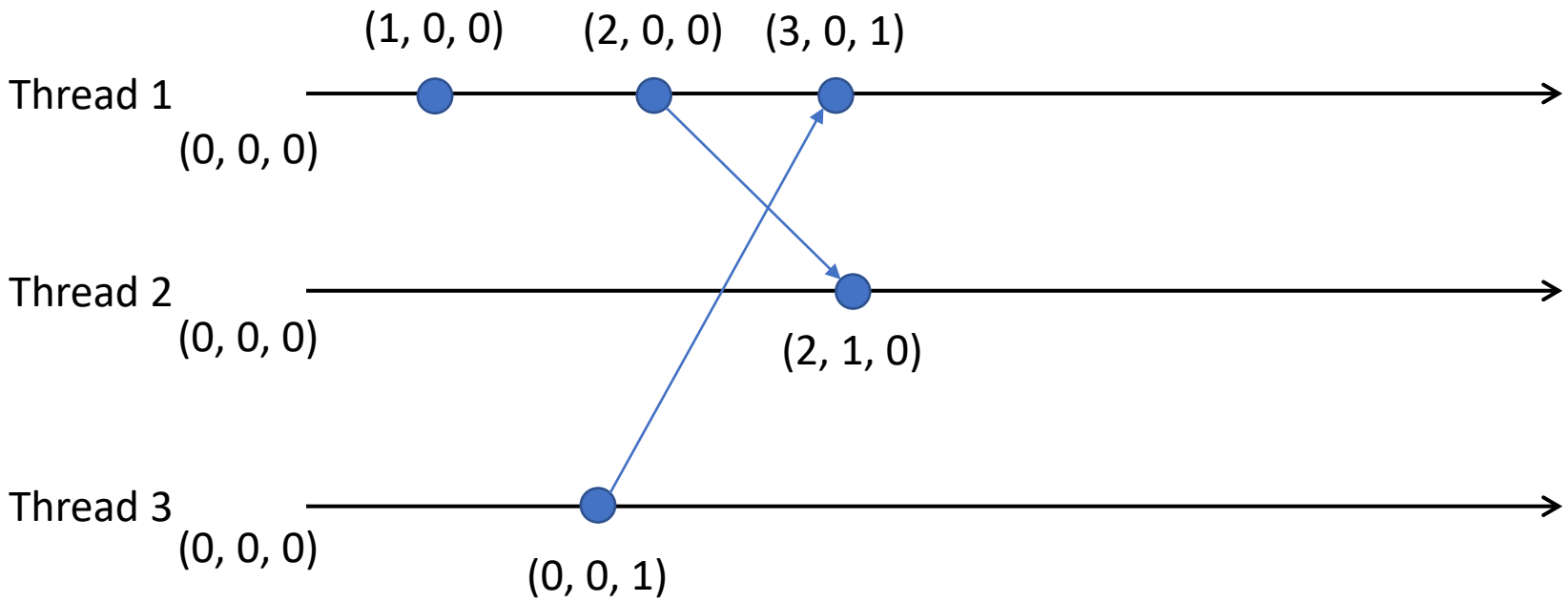
Запись в атомик – отправка
Чтение – получение
К атомику прилагаются часы

Thread Sanitizer – Data Race detector

Логические ВЕКТОРНЫЕ часы

Запись в атомик – отправка
Чтение – получение
К атомику прилагаются часы

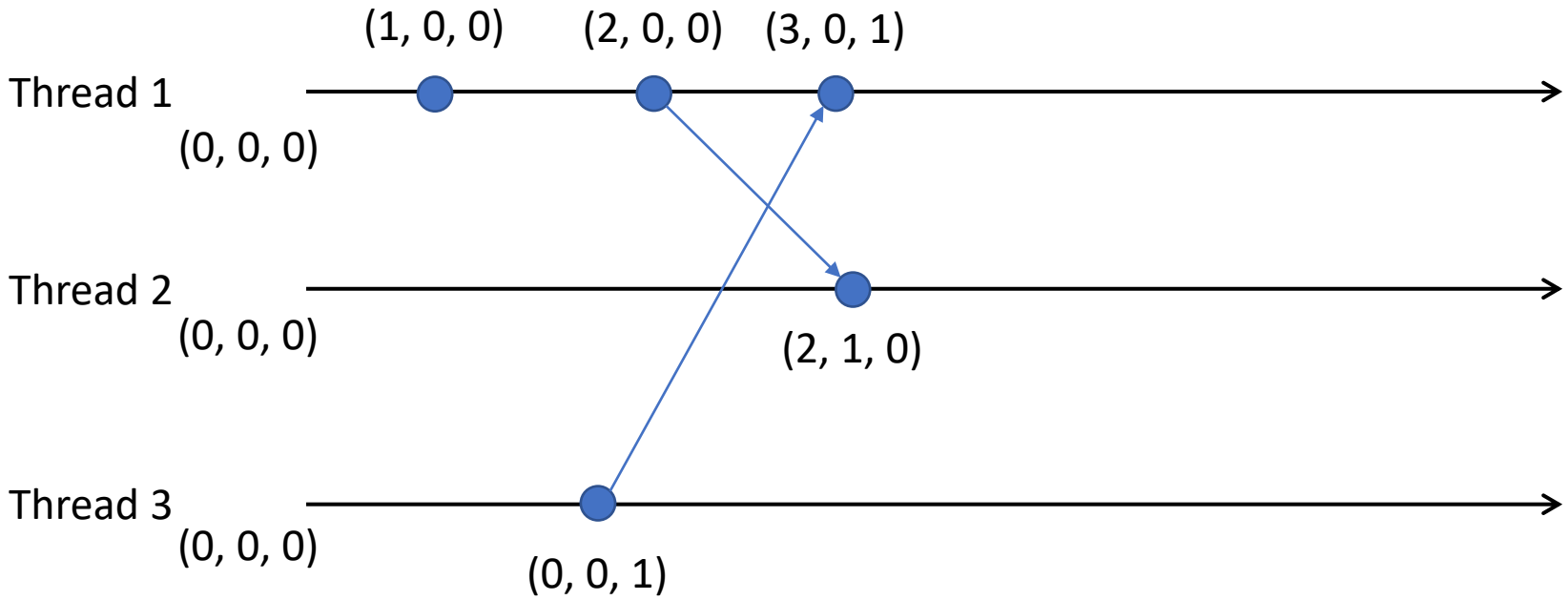
- К каждому событию записи/чтения сопоставим временную метку T
- Событие e1 предшествует событию e2 ⇔ T(e1) < T(e2)



Thread Sanitizer – Data Race detector

Логические ВЕКТОРНЫЕ часы

- К каждому событию записи/чтения сопоставим временную метку T
- Событие e1 предшествует событию e2 ⇔ T(e1) < T(e2)



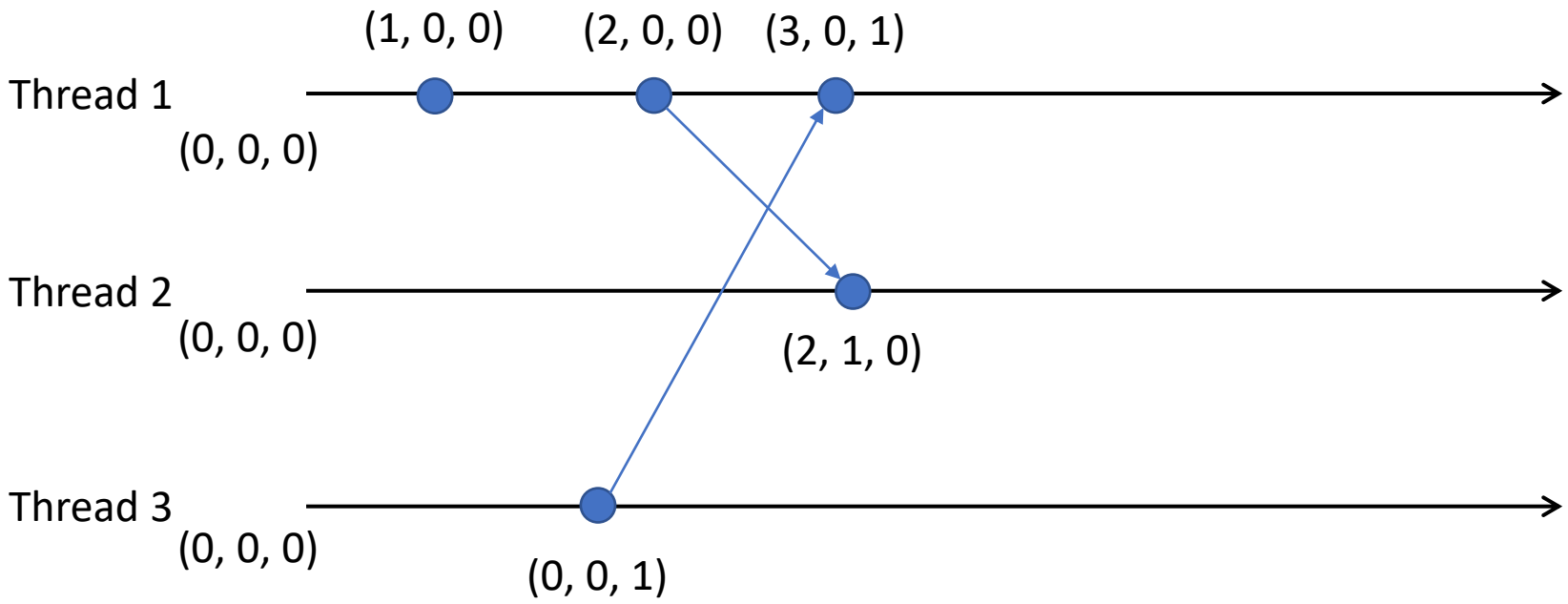
Запись в атомик – отправка
Чтение – получение
К атомику прилагаются часы

Для обращений к памяти будем хранить ThId + epoch

Thread Sanitizer – Data Race detector

Логические ВЕКТОРНЫЕ часы

- К каждому событию записи/чтения сопоставим временную метку T
- Событие e1 предшествует событию e2 ⇔ T(e1) < T(e2)



Запись в атомик – отправка
Чтение – получение
К атомику прилагаются часы

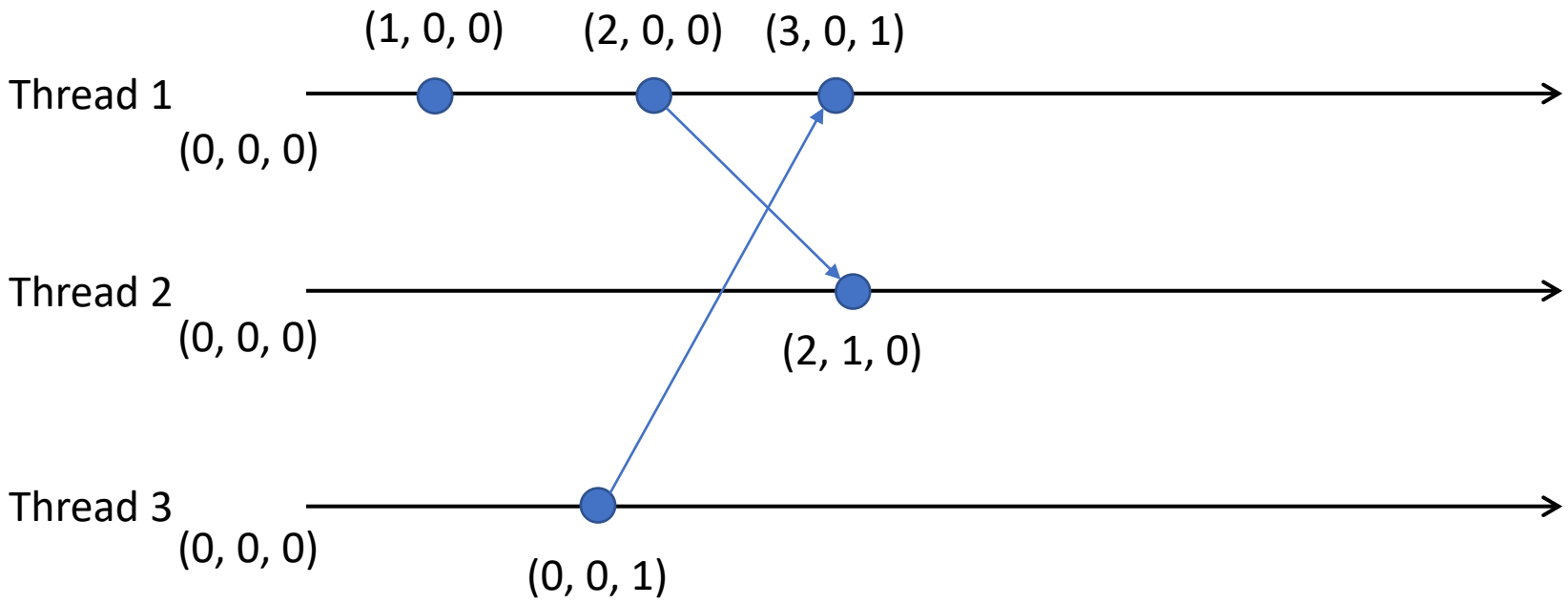
Для обращений к памяти будем хранить ThId + epoch

epoch – локальная (собственная) компонента часов того, кто обращался на момент обращения

Thread Sanitizer – Data Race detector

Логические ВЕКТОРНЫЕ часы

- К каждому событию записи/чтения сопоставим временную метку T
- Событие e1 предшествует событию e2 ⇔ T(e1) < T(e2)



Запись в атомик – отправка
Чтение – получение
К атомику прилагаются часы

Для обращений к памяти будем хранить ThId + epoch

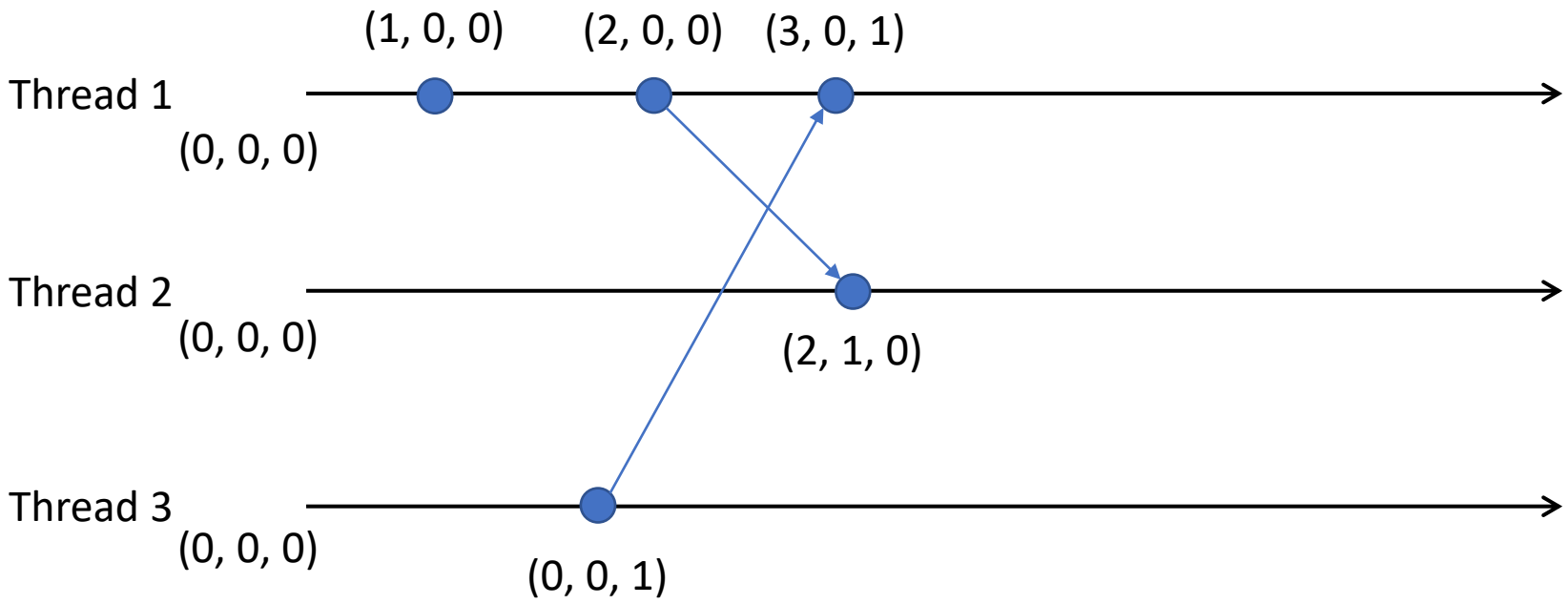
epoch – локальная (собственная) компонента часов того, кто обращался на момент обращения

Thread – имеет векторные часы (полные)

Thread Sanitizer – Data Race detector

Логические ВЕКТОРНЫЕ часы

- К каждому событию записи/чтения сопоставим временную метку T
- Событие e1 предшествует событию e2 ⇔ T(e1) < T(e2)



Запись в атомик – отправка
Чтение – получение
К атомику прилагаются часы

Для обращений к памяти будем хранить ThId + epoch

epoch – локальная (собственная) компонента часов того, кто обращался на момент обращения

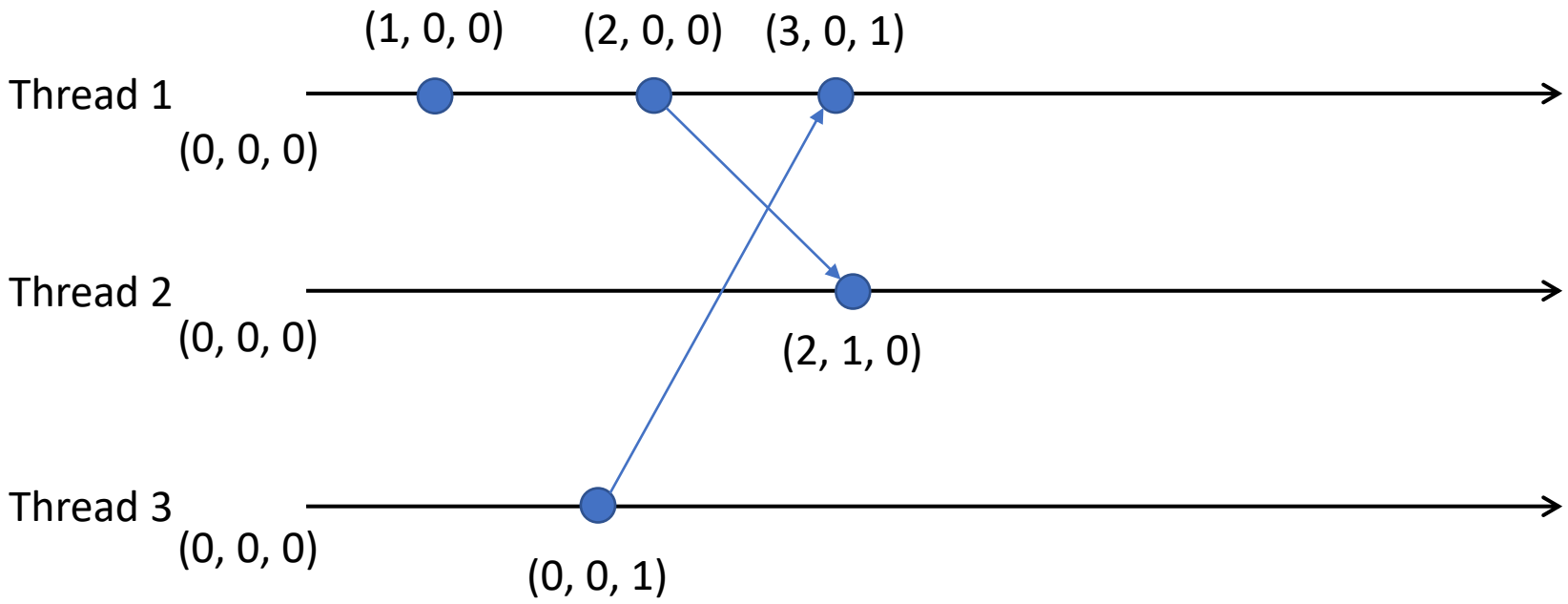
Thread – имеет векторные часы (полные)

Атомик - имеет векторные часы (полные)

Thread Sanitizer – Data Race detector

Логические ВЕКТОРНЫЕ часы

Проверка: current_thread.clock[old.ThId] >= old.epoch



Запись в атомик – отправка
Чтение – получение
К атомику прилагаются часы

Для обращений к памяти будем хранить ThId + epoch

epoch – локальная (собственная) компонента часов того, кто обращался на момент обращения

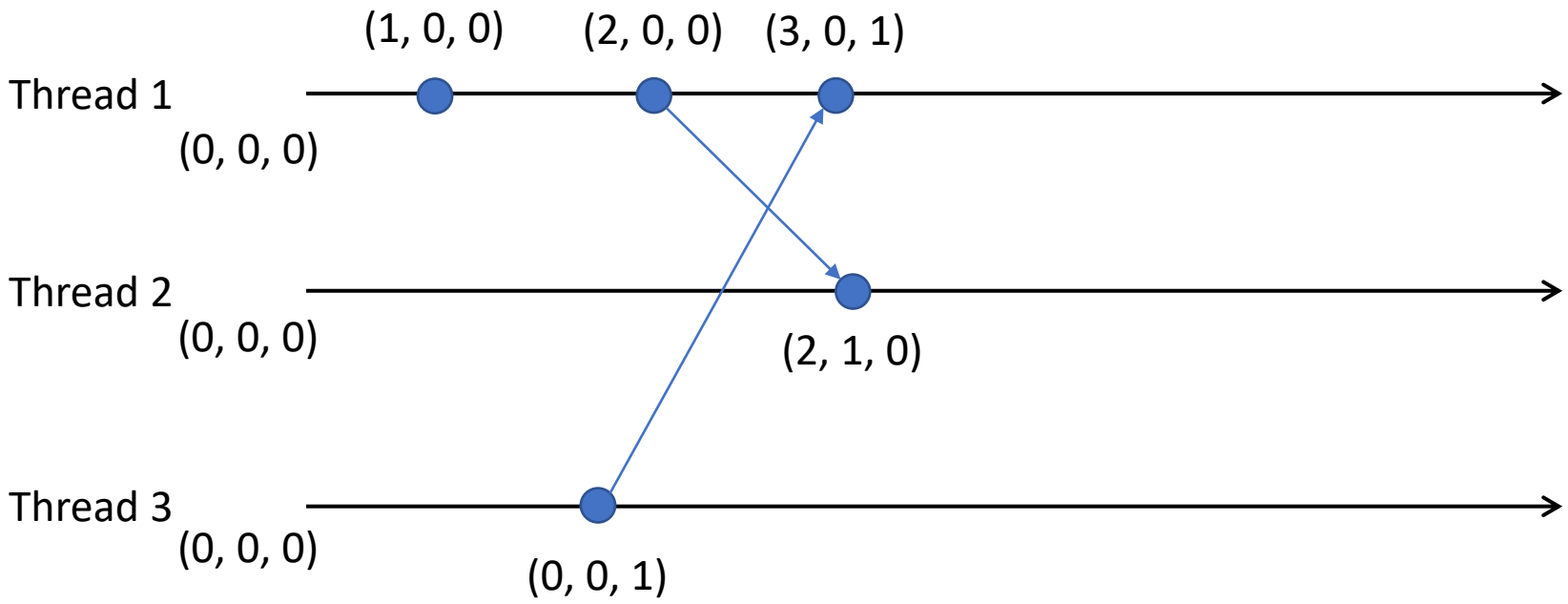
Thread – имеет векторные часы (полные)

Атомик - имеет векторные часы (полные)

Thread Sanitizer – Data Race detector

Логические ВЕКТОРНЫЕ часы

Проверка: current_thread.clock[old.ThId] >= old.epoch
1 запись, и последние чтения из каждого потока



Запись в атомик – отправка
Чтение – получение
К атомику прилагаются часы

Для обращений к памяти будем хранить ThId + epoch

epoch – локальная (собственная) компонента часов того, кто обращался на момент обращения

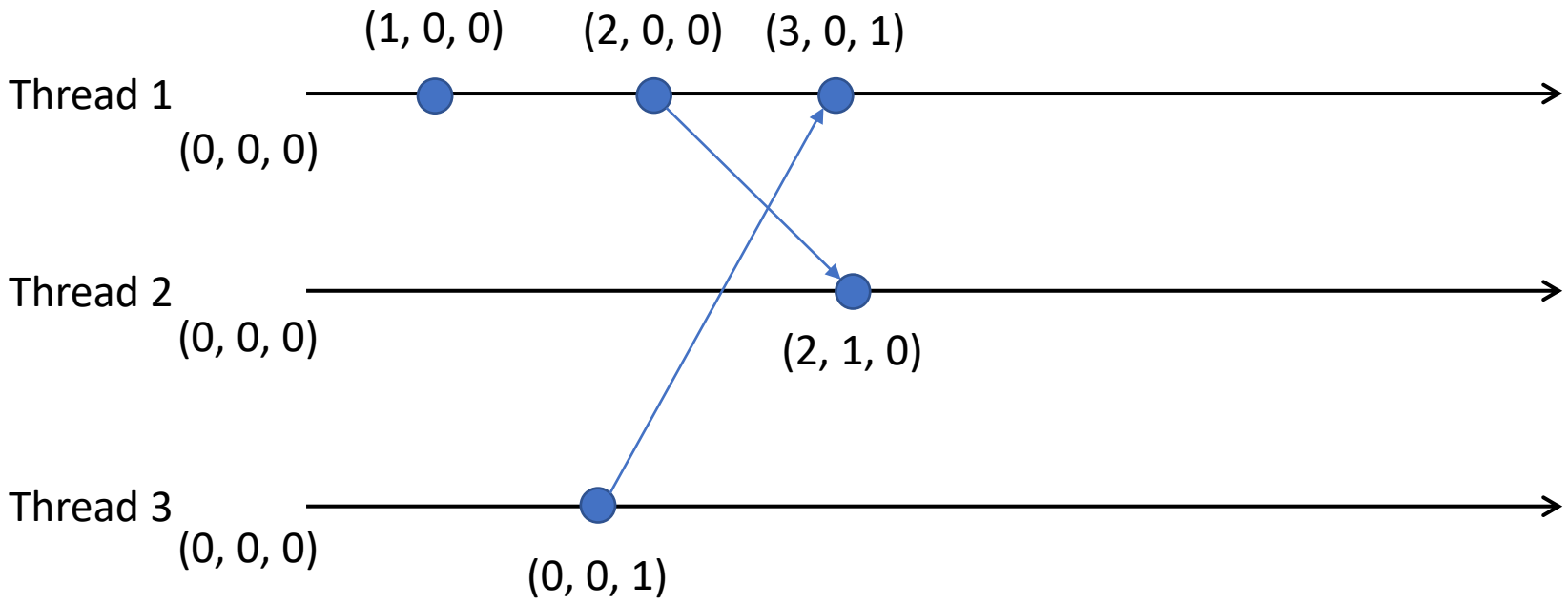
Thread – имеет векторные часы (полные)

Атомик - имеет векторные часы (полные)

Thread Sanitizer – Data Race detector

Логические ВЕКТОРНЫЕ часы

Проверка: current_thread.clock[old.ThId] >= old.epoch
1 запись, и последние чтения из каждого потока
Будем хранить только последние 4 обращения к ячейке памяти



Запись в атомик – отправка
Чтение – получение
К атомику прилагаются часы

Для обращений к памяти будем хранить ThId + epoch

epoch – локальная (собственная) компонента часов того, кто обращался на момент обращения

Thread – имеет векторные часы (полные)

Атомик - имеет векторные часы (полные)

Thread Sanitizer – Data Race detector

Как реализовано - инструментирование

```
std::atomic<bool> a;
bool b;
void foo() {
    a.store(true);
    b = true;
}
```

Thread Sanitizer – Data Race detector

Как реализовано - инструментирование

1. Инструментирование атомиков

```
std::atomic<bool> a;
bool b;
void foo() {
    a.store(true);
    b = true;
}
```

Thread Sanitizer – Data Race detector

Как реализовано - инструментирование

1. Инструментирование атомиков

```
std::atomic<bool> a;
bool b;
void foo() {
    a.store(true);
    b = true;
}
```

```
foo():                                  #
        push    rax
        mov     rdi, qword ptr [rsp + 8]
        call    __tsan_func_entry@PLT
        mov     edi, offset a
        mov     esi, 1
        mov     edx, 5
        call    __tsan_atomic8_store@PLT
        mov     edi, offset b
        call    __tsan_write1@PLT
        mov     byte ptr [rip + b], 1
        call    __tsan_func_exit@PLT
        pop     rax
        ret
```

Thread Sanitizer – Data Race detector

Как реализовано - инструментирование

1. Инструментирование атомиков
2. Инструментирование не атомиков

```
std::atomic<bool> a;
bool b;
void foo() {
    a.store(true);
    b = true;
}
```

```
foo():                                  #
        push    rax
        mov     rdi, qword ptr [rsp + 8]
        call    __tsan_func_entry@PLT
        mov     edi, offset a
        mov     esi, 1
        mov     edx, 5
        call    __tsan_atomic8_store@PLT
        mov     edi, offset b
        call    __tsan_write1@PLT
        mov     byte ptr [rip + b], 1
        call    __tsan_func_exit@PLT
        pop     rax
        ret
```

Thread Sanitizer – Data Race detector

Как реализовано - инструментирование

1. Инструментирование атомиков
2. Инструментирование не атомиков
3. Инструментирование входа/выхода в функцию

```
std::atomic<bool> a;
bool b;
void foo() {
    a.store(true);
    b = true;
}
```

```
foo():                                  #
        push    rax
        mov     rdi, qword ptr [rsp + 8]
        call    __tsan_func_entry@PLT
        mov     edi, offset a
        mov     esi, 1
        mov     edx, 5
        call    __tsan_atomic8_store@PLT
        mov     edi, offset b
        call    __tsan_write1@PLT
        mov     byte ptr [rip + b], 1
        call    __tsan_func_exit@PLT
        pop     rax
        ret
```

Thread Sanitizer – Data Race detector

Как реализовано - инструментирование

1. Инструментирование атомиков
2. Инструментирование не атомиков
3. Инструментирование входа/выхода в функцию

```
std::atomic<bool> a;
bool b;
void foo() {
    a.store(true);
    b = true;
}
```

```
foo():                                  #
        push    rax
        mov     rdi, qword ptr [rsp + 8]
        call    __tsan_func_entry@PLT
        mov     edi, offset a
        mov     esi, 1
        mov     edx, 5
        call    __tsan_atomic8_store@PLT
        mov     edi, offset b
        call    __tsan_write1@PLT
        mov     byte ptr [rip + b], 1
        call    __tsan_func_exit@PLT
        pop     rax
        ret
```

Нет никакой особой логики.

Thread Sanitizer – Data Race detector

Как реализовано - инструментирование

1. Инструментирование атомиков
2. Инструментирование не атомиков
3. Инструментирование входа/выхода в функцию

```
std::atomic<bool> a;
bool b;
void foo() {
    a.store(true);
    b = true;
}
```

```
foo():                                  #
        push    rax
        mov     rdi, qword ptr [rsp + 8]
        call    __tsan_func_entry@PLT
        mov     edi, offset a
        mov     esi, 1
        mov     edx, 5
        call    __tsan_atomic8_store@PLT
        mov     edi, offset b
        call    __tsan_write1@PLT
        mov     byte ptr [rip + b], 1
        call    __tsan_func_exit@PLT
        pop     rax
        ret
```

Нет никакой особой логики.
Только вызовы интерфейсных функций

Thread Sanitizer – Data Race detector

Как реализовано - инструментирование

1. Инструментирование атомиков
2. Инструментирование не атомиков
3. Инструментирование входа/выхода в функцию

```
std::atomic<bool> a;
bool b;
void foo() {
    a.store(true);
    b = true;
}
```

```
foo():                                  #
        push    rax
        mov     rdi, qword ptr [rsp + 8]
        call    __tsan_func_entry@PLT
        mov     edi, offset a
        mov     esi, 1
        mov     edx, 5
        call    __tsan_atomic8_store@PLT
        mov     edi, offset b
        call    __tsan_write1@PLT
        mov     byte ptr [rip + b], 1
        call    __tsan_func_exit@PLT
        pop     rax
        ret
```

Нет никакой особой логики.

Только вызовы интерфейсных функций

Не как в ASan

```
1  #include <cstdint>
2  void foo(uint64_t* a) {
3      *a = 42;
4  }
```

```
1  foo(unsigned long*):
2          mov     rax, rdi
3          shr     rax, 3
4          cmp     BYTE PTR [rax+2147450880], 0
5          jne     .L7
6          mov     QWORD PTR [rdi], 42
7          ret
8  .L7:
9          push    rax
10         call    __asan_report_store8
```

Thread Sanitizer – Data Race detector

Как реализовано - инструментирование

LLVM

main/llvm/include/llvm/Transforms/Instrumentation/ThreadSanitizer.h

main/llvm/lib/Transforms/Instrumentation/ThreadSanitizer.cpp

GCC

master/gcc/tsan.h

master/gcc/tsan.c

Thread Sanitizer – Data Race detector

Как реализовано - рантайм

https://github.com/llvm/llvm-project/tree/main/compiler-rt/lib/tsan/rtl

Thread Sanitizer – Data Race detector

Как реализовано - рантайм

https://github.com/llvm/llvm-project/tree/main/compiler-rt/lib/tsan/rtl

https://github.com/llvm/llvm-project/blob/main/compiler-rt/lib/tsan/rtl/tsan_interface.h

```
SANITIZER_INTERFACE_ATTRIBUTE void __tsan_read1(void *addr);
SANITIZER_INTERFACE_ATTRIBUTE void __tsan_read2(void *addr);
SANITIZER_INTERFACE_ATTRIBUTE void __tsan_read4(void *addr);
SANITIZER_INTERFACE_ATTRIBUTE void __tsan_read8(void *addr);
SANITIZER_INTERFACE_ATTRIBUTE void __tsan_read16(void *addr);

SANITIZER_INTERFACE_ATTRIBUTE void __tsan_write1(void *addr);
SANITIZER_INTERFACE_ATTRIBUTE void __tsan_write2(void *addr);
SANITIZER_INTERFACE_ATTRIBUTE void __tsan_write4(void *addr);
SANITIZER_INTERFACE_ATTRIBUTE void __tsan_write8(void *addr);
SANITIZER_INTERFACE_ATTRIBUTE void __tsan_write16(void *addr);
```

Thread Sanitizer – Data Race detector

Как реализовано - рантайм

https://github.com/llvm/llvm-project/tree/main/compiler-rt/lib/tsan/rtl

https://github.com/llvm/llvm-project/blob/main/compiler-rt/lib/tsan/rtl/tsan_interface.h

```
SANITIZER_INTERFACE_ATTRIBUTE void __tsan_read1(void *addr);
SANITIZER_INTERFACE_ATTRIBUTE void __tsan_read2(void *addr);
SANITIZER_INTERFACE_ATTRIBUTE void __tsan_read4(void *addr);
SANITIZER_INTERFACE_ATTRIBUTE void __tsan_read8(void *addr);
SANITIZER_INTERFACE_ATTRIBUTE void __tsan_read16(void *addr);

SANITIZER_INTERFACE_ATTRIBUTE void __tsan_write1(void *addr);
SANITIZER_INTERFACE_ATTRIBUTE void __tsan_write2(void *addr);
SANITIZER_INTERFACE_ATTRIBUTE void __tsan_write4(void *addr);
SANITIZER_INTERFACE_ATTRIBUTE void __tsan_write8(void *addr);
SANITIZER_INTERFACE_ATTRIBUTE void __tsan_write16(void *addr);
```

Thread Sanitizer – Data Race detector

Как реализовано - рантайм

https://github.com/llvm/llvm-project/tree/main/compiler-rt/lib/tsan/rtl

https://github.com/llvm/llvm-project/blob/main/compiler-rt/lib/tsan/rtl/tsan_interface.h

```
SANITIZER_INTERFACE_ATTRIBUTE void __tsan_read1(void *addr);
SANITIZER_INTERFACE_ATTRIBUTE void __tsan_read2(void *addr);
SANITIZER_INTERFACE_ATTRIBUTE void __tsan_read4(void *addr);
SANITIZER_INTERFACE_ATTRIBUTE void __tsan_read8(void *addr);
SANITIZER_INTERFACE_ATTRIBUTE void __tsan_read16(void *addr);

SANITIZER_INTERFACE_ATTRIBUTE void __tsan_write1(void *addr);
SANITIZER_INTERFACE_ATTRIBUTE void __tsan_write2(void *addr);
SANITIZER_INTERFACE_ATTRIBUTE void __tsan_write4(void *addr);
SANITIZER_INTERFACE_ATTRIBUTE void __tsan_write8(void *addr);
SANITIZER_INTERFACE_ATTRIBUTE void __tsan_write16(void *addr);
```

```
SANITIZER_INTERFACE_ATTRIBUTE
void __tsan_atomic8_store(volatile a8 *a, a8 v, morder mo);
SANITIZER_INTERFACE_ATTRIBUTE
void __tsan_atomic16_store(volatile a16 *a, a16 v, morder mo);
SANITIZER_INTERFACE_ATTRIBUTE
void __tsan_atomic32_store(volatile a32 *a, a32 v, morder mo);
SANITIZER_INTERFACE_ATTRIBUTE
void __tsan_atomic64_store(volatile a64 *a, a64 v, morder mo);
#if __TSAN_HAS_INT128
SANITIZER_INTERFACE_ATTRIBUTE
void __tsan_atomic128_store(volatile a128 *a, a128 v, morder mo);
```

Thread Sanitizer – Data Race detector

Как реализовано - рантайм

https://github.com/llvm/llvm-project/tree/main/compiler-rt/lib/tsan/rtl

https://github.com/llvm/llvm-project/blob/main/compiler-rt/lib/tsan/rtl/tsan_interface.h

```
SANITIZER_INTERFACE_ATTRIBUTE void __tsan_read1(void *addr);
SANITIZER_INTERFACE_ATTRIBUTE void __tsan_read2(void *addr);
SANITIZER_INTERFACE_ATTRIBUTE void __tsan_read4(void *addr);
SANITIZER_INTERFACE_ATTRIBUTE void __tsan_read8(void *addr);
SANITIZER_INTERFACE_ATTRIBUTE void __tsan_read16(void *addr);

SANITIZER_INTERFACE_ATTRIBUTE void __tsan_write1(void *addr);
SANITIZER_INTERFACE_ATTRIBUTE void __tsan_write2(void *addr);
SANITIZER_INTERFACE_ATTRIBUTE void __tsan_write4(void *addr);
SANITIZER_INTERFACE_ATTRIBUTE void __tsan_write8(void *addr);
SANITIZER_INTERFACE_ATTRIBUTE void __tsan_write16(void *addr);
```

```
SANITIZER_INTERFACE_ATTRIBUTE
void __tsan_atomic8_store(volatile a8 *a, a8 v, morder mo);
SANITIZER_INTERFACE_ATTRIBUTE
void __tsan_atomic16_store(volatile a16 *a, a16 v, morder mo);
SANITIZER_INTERFACE_ATTRIBUTE
void __tsan_atomic32_store(volatile a32 *a, a32 v, morder mo);
SANITIZER_INTERFACE_ATTRIBUTE
void __tsan_atomic64_store(volatile a64 *a, a64 v, morder mo);
#if __TSAN_HAS_INT128
SANITIZER_INTERFACE_ATTRIBUTE
void __tsan_atomic128_store(volatile a128 *a, a128 v, morder mo);
```

Thread Sanitizer – Data Race detector

Как реализовано - рантайм

https://github.com/llvm/llvm-project/blob/main/compiler-rt/lib/tsan/rtl/tsan_interface.h
https://github.com/llvm/llvm-project/blob/main/compiler-rt/lib/tsan/rtl/tsan_interface.inc

```
SANITIZER_INTERFACE_ATTRIBUTE void __tsan_read1(void *addr);
SANITIZER_INTERFACE_ATTRIBUTE void __tsan_read2(void *addr);
SANITIZER_INTERFACE_ATTRIBUTE void __tsan_read4(void *addr);
SANITIZER_INTERFACE_ATTRIBUTE void __tsan_read8(void *addr);
SANITIZER_INTERFACE_ATTRIBUTE void __tsan_read16(void *addr);

SANITIZER_INTERFACE_ATTRIBUTE void __tsan_write1(void *addr);
SANITIZER_INTERFACE_ATTRIBUTE void __tsan_write2(void *addr);
SANITIZER_INTERFACE_ATTRIBUTE void __tsan_write4(void *addr);
SANITIZER_INTERFACE_ATTRIBUTE void __tsan_write8(void *addr);
SANITIZER_INTERFACE_ATTRIBUTE void __tsan_write16(void *addr);
```

```
37  void __tsan_write1(void *addr) {
38    MemoryAccess(cur_thread(), CALLERPC, (uptr)addr, 1, kAccessWrite);
39  }
```

Thread Sanitizer – Data Race detector

Как реализовано - рантайм

https://github.com/llvm/llvm-project/blob/main/compiler-rt/lib/tsan/rtl/tsan_interface.inc
https://github.com/llvm/llvm-project/blob/main/compiler-rt/lib/tsan/rtl/tsan_rtl_access.cpp

```
37  void __tsan_write1(void *addr) {
38    MemoryAccess(cur_thread(), CALLERPC, (uptr)addr, 1, kAccessWrite);
39  }
```

```
350  ALWAYS_INLINE USED void MemoryAccess(ThreadState *thr, uptr pc, uptr addr,
351                                       int kAccessSizeLog, bool kAccessIsWrite,
352                                       bool kIsAtomic) {
353    RawShadow *shadow_mem = MemToShadow(addr);
378    FastState fast_state = thr->fast_state;
383     Shadow cur(fast_state);
400    MemoryAccessImpl1(thr, addr, kAccessSizeLog, kAccessIsWrite, kIsAtomic,
401                      shadow_mem, cur);
```

```
175  void MemoryAccessImpl1(ThreadState *thr, uptr addr, int kAccessSizeLog,
176                         bool kAccessIsWrite, bool kIsAtomic, u64 *shadow_mem,
177                         Shadow cur) {
201    for (int idx = 0; idx < 4; idx++) {
202  #  include "tsan_update_shadow_word.inc"
203    }
```

```
229    if (LIKELY(stored))
230      return;
231    // choose a random candidate slot and replace it
232    StoreShadow(shadow_mem + (cur.epoch() % kShadowCnt), store_word);
233    return;
234  RACE:
235    HandleRace(thr, shadow_mem, cur, old);
236    return;
237  }
```

Thread Sanitizer – Data Race detector

Как реализовано - рантайм

https://github.com/llvm/llvm-project/blob/main/compiler-rt/lib/tsan/rtl/tsan_rtl_access.cpp
https://github.com/llvm/llvm-project/blob/main/compiler-rt/lib/tsan/rtl/tsan_update_shadow_word.inc

```
15   do {
16     const unsigned kAccessSize = 1 << kAccessSizeLog;
17     u64 *sp = &shadow_mem[idx];
18     old = LoadShadow(sp);
```

```
175  void MemoryAccessImpl1(ThreadState *thr, uptr addr, int kAccessSizeLog,
176                         bool kAccessIsWrite, bool kIsAtomic, u64 *shadow_mem,
177                         Shadow cur) {
201    for (int idx = 0; idx < 4; idx++) {
202  #   include "tsan_update_shadow_word.inc"
203    }
```

```
229    if (LIKELY(stored))
230      return;
231    // choose a random candidate slot and replace it
232    StoreShadow(shadow_mem + (cur.epoch() % kShadowCnt), store_word);
233    return;
234  RACE:
235    HandleRace(thr, shadow_mem, cur, old);
236    return;
237  }
```

Thread Sanitizer – Data Race detector

Как реализовано - рантайм

https://github.com/llvm/llvm-project/blob/main/compiler-rt/lib/tsan/rtl/tsan_rtl_access.cpp
https://github.com/llvm/llvm-project/blob/main/compiler-rt/lib/tsan/rtl/tsan_update_shadow_word.inc

```
15  do {
16    const unsigned kAccessSize = 1 << kAccessSizeLog;
17    u64 *sp = &shadow_mem[idx];
18    old = LoadShadow(sp);
26    // is the memory access equal to the previous?
27    if (LIKELY(Shadow::Addr0AndSizeAreEqual(cur, old))) {
28      // same thread?
29      if (LIKELY(Shadow::TidsAreEqual(old, cur))) {
30        if (LIKELY(old.IsRWWeakerOrEqual(kAccessIsWrite, kIsAtomic))) {
31          StoreIfNotYetStored(sp, &store_word);
32          stored = true;
33        }
34        break;
35      }
36      if (HappensBefore(old, thr)) {
37        if (old.IsRWWeakerOrEqual(kAccessIsWrite, kIsAtomic)) {
38          StoreIfNotYetStored(sp, &store_word);
39          stored = true;
40        }
41        break;
42      }
43      if (LIKELY(old.IsBothReadsOrAtomic(kAccessIsWrite, kIsAtomic)))
44        break;
45      goto RACE;
46    }
```

```
175  void MemoryAccessImpl1(ThreadState *thr, uptr addr, int kAccessSizeLog,
176                         bool kAccessIsWrite, bool kIsAtomic, u64 *shadow_mem,
177                         Shadow cur) {
201    for (int idx = 0; idx < 4; idx++) {
202  #   include "tsan_update_shadow_word.inc"
203    }
229    if (LIKELY(stored))
230      return;
231    // choose a random candidate slot and replace it
232    StoreShadow(shadow_mem + (cur.epoch() % kShadowCnt), store_word);
233    return;
234  RACE:
235    HandleRace(thr, shadow_mem, cur, old);
236    return;
237  }
```

Thread Sanitizer – Data Race detector

Как реализовано - рантайм

https://github.com/llvm/llvm-project/blob/main/compiler-rt/lib/tsan/rtl/tsan_rtl_access.cpp
https://github.com/llvm/llvm-project/blob/main/compiler-rt/lib/tsan/rtl/tsan_update_shadow_word.inc

```
15  do {
16    const unsigned kAccessSize = 1 << kAccessSizeLog;
17    u64 *sp = &shadow_mem[idx];
18    old = LoadShadow(sp);
26   // is the memory access equal to the previous?
27   if (LIKELY(Shadow::Addr0AndSizeAreEqual(cur, old))) {
28     // same thread?
29     if (LIKELY(Shadow::TidsAreEqual(old, cur))) {
30       if (LIKELY(old.IsRWWeakerOrEqual(kAccessIsWrite, kIsAtomic))) {
31         StoreIfNotYetStored(sp, &store_word);
32         stored = true;
33       }
34       break;
35     }
36     if (HappensBefore(old, thr)) {
37       if (old.IsRWWeakerOrEqual(kAccessIsWrite, kIsAtomic)) {
38         StoreIfNotYetStored(sp, &store_word);
39         stored = true;
40       }
41       break;
42     }
43     if (LIKELY(old.IsBothReadsOrAtomic(kAccessIsWrite, kIsAtomic)))
44       break;
45     goto RACE;
46   }
```

```
175  void MemoryAccessImpl1(ThreadState *thr, uptr addr, int kAccessSizeLog,
176                         bool kAccessIsWrite, bool kIsAtomic, u64 *shadow_mem,
177                         Shadow cur) {
201    for (int idx = 0; idx < 4; idx++) {
202  #   include "tsan_update_shadow_word.inc"
203    }
229    if (LIKELY(stored))
230      return;
231    // choose a random candidate slot and replace it
232    StoreShadow(shadow_mem + (cur.epoch() % kShadowCnt), store_word);
233    return;
234  RACE:
235    HandleRace(thr, shadow_mem, cur, old);
236    return;
237  }
```

Thread Sanitizer – Data Race detector

Как реализовано - рантайм

https://github.com/llvm/llvm-project/blob/main/compiler-rt/lib/tsan/rtl/tsan_rtl_access.cpp
https://github.com/llvm/llvm-project/blob/main/compiler-rt/lib/tsan/rtl/tsan_update_shadow_word.inc

```
15  do {
16    const unsigned kAccessSize = 1 << kAccessSizeLog;
17    u64 *sp = &shadow_mem[idx];
18    old = LoadShadow(sp);
26    // is the memory access equal to the previous?
27    if (LIKELY(Shadow::Addr0AndSizeAreEqual(cur, old))) {
28      // same thread?
29      if (LIKELY(Shadow::TidsAreEqual(old, cur))) {
30        if (LIKELY(old.IsRWWeakerOrEqual(kAccessIsWrite, kIsAtomic))) {
31          StoreIfNotYetStored(sp, &store_word);
32          stored = true;
33        }
34        break;
35      }
36      if (HappensBefore(old, thr)) {
37        if (old.IsRWWeakerOrEqual(kAccessIsWrite, kIsAtomic)) {
38          StoreIfNotYetStored(sp, &store_word);
39          stored = true;
40        }
41        break;
42      }
43      if (LIKELY(old.IsBothReadsOrAtomic(kAccessIsWrite, kIsAtomic)))
44        break;
45      goto RACE;
46    }
```

```
175  void MemoryAccessImpl1(ThreadState *thr, uptr addr, int kAccessSizeLog,
176                         bool kAccessIsWrite, bool kIsAtomic, u64 *shadow_mem,
177                         Shadow cur) {
201    for (int idx = 0; idx < 4; idx++) {
202  #   include "tsan_update_shadow_word.inc"
203    }
229    if (LIKELY(stored))
230      return;
231    // choose a random candidate slot and replace it
232    StoreShadow(shadow_mem + (cur.epoch() % kShadowCnt), store_word);
233    return;
234  RACE:
235    HandleRace(thr, shadow_mem, cur, old);
236    return;
237  }
```

Thread Sanitizer – Data Race detector

Как реализовано - рантайм

https://github.com/llvm/llvm-project/blob/main/compiler-rt/lib/tsan/rtl/tsan_rtl_access.cpp
https://github.com/llvm/llvm-project/blob/main/compiler-rt/lib/tsan/rtl/tsan_update_shadow_word.inc

```
15  do {
16    const unsigned kAccessSize = 1 << kAccessSizeLog;
17    u64 *sp = &shadow_mem[idx];
18    old = LoadShadow(sp);
26   // is the memory access equal to the previous?
27   if (LIKELY(Shadow::Addr0AndSizeAreEqual(cur, old))) {
28     // same thread?
29     if (LIKELY(Shadow::TidsAreEqual(old, cur))) {
30       if (LIKELY(old.IsRWWeakerOrEqual(kAccessIsWrite, kIsAtomic))) {
31         StoreIfNotYetStored(sp, &store_word);
32         stored = true;
33       }
34       break;
35     }
36     if (HappensBefore(old, thr)) {
37       if (old.IsRWWeakerOrEqual(kAccessIsWrite, kIsAtomic)) {
38         StoreIfNotYetStored(sp, &store_word);
39         stored = true;
40       }
41       break;
42     }
43     if (LIKELY(old.IsBothReadsOrAtomic(kAccessIsWrite, kIsAtomic)))
44       break;
45     goto RACE;
46   }
```

```
175  void MemoryAccessImpl1(ThreadState *thr, uptr addr, int kAccessSizeLog,
176                         bool kAccessIsWrite, bool kIsAtomic, u64 *shadow_mem,
177                         Shadow cur) {
201    for (int idx = 0; idx < 4; idx++) {
202  #   include "tsan_update_shadow_word.inc"
203    }
229    if (LIKELY(stored))
230      return;
231    // choose a random candidate slot and replace it
232    StoreShadow(shadow_mem + (cur.epoch() % kShadowCnt), store_word);
233    return;
234  RACE:
235    HandleRace(thr, shadow_mem, cur, old);
236    return;
237  }
```

```
170  static inline bool HappensBefore(Shadow old, ThreadState *thr) {
171    return thr->clock.get(old.TidWithIgnore()) >= old.epoch();
172  }
```

Thread Sanitizer – Data Race detector

Как реализовано - рантайм

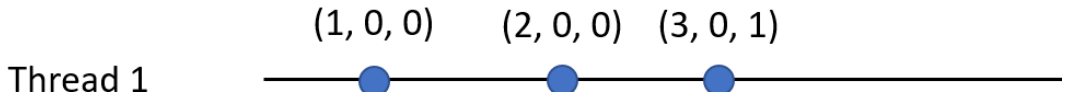
https://github.com/llvm/llvm-project/blob/main/compiler-rt/lib/tsan/rtl/tsan_rtl_access.cpp
https://github.com/llvm/llvm-project/blob/main/compiler-rt/lib/tsan/rtl/tsan_update_shadow_word.inc

```
15  do {
16    const unsigned kAccessSize = 1 << kAccessSizeLog;
17    u64 *sp = &shadow_mem[idx];
18    old = LoadShadow(sp);
26   // is the memory access equal to the previous?
27   if (LIKELY(Shadow::Addr0AndSizeAreEqual(cur, old))) {
28     // same thread?
29     if (LIKELY(Shadow::TidsAreEqual(old, cur))) {
30       if (LIKELY(old.IsRWWeakerOrEqual(kAccessIsWrite, kIsAtomic))) {
31         StoreIfNotYetStored(sp, &store_word);
32         stored = true;
33       }
34       break;
35     }
36     if (HappensBefore(old, thr)) {
37       if (old.IsRWWeakerOrEqual(kAccessIsWrite, kIsAtomic)) {
38         StoreIfNotYetStored(sp, &store_word);
39         stored = true;
40       }
41       break;
42     }
43     if (LIKELY(old.IsBothReadsOrAtomic(kAccessIsWrite, kIsAtomic)))
44       break;
45     goto RACE;
46   }
```

```
175  void MemoryAccessImpl1(ThreadState *thr, uptr addr, int kAccessSizeLog,
176                         bool kAccessIsWrite, bool kIsAtomic, u64 *shadow_mem,
177                         Shadow cur) {
201    for (int idx = 0; idx < 4; idx++) {
202  #   include "tsan_update_shadow_word.inc"
203    }
229    if (LIKELY(stored))
230      return;
231    // choose a random candidate slot and replace it
232    StoreShadow(shadow_mem + (cur.epoch() % kShadowCnt), store_word);
233    return;
234  RACE:
235    HandleRace(thr, shadow_mem, cur, old);
236    return;
237  }
```

```
170  static inline bool HappensBefore(Shadow old, ThreadState *thr) {
171    return thr->clock.get(old.TidWithIgnore()) >= old.epoch();
172  }
```

Проверка: current_thread.clock[old.ThId] >= old.epoch
1 запись, и последние чтения из каждого потока
Будем хранить только последние 4 обращения к ячейке памяти



Thread Sanitizer – Data Race detector

Как реализовано - рантайм

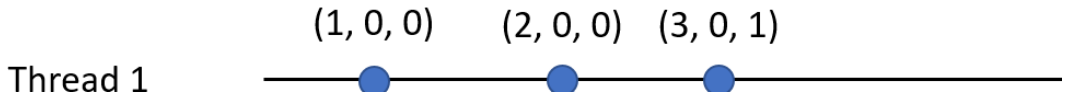
https://github.com/llvm/llvm-project/blob/main/compiler-rt/lib/tsan/rtl/tsan_rtl_access.cpp
https://github.com/llvm/llvm-project/blob/main/compiler-rt/lib/tsan/rtl/tsan_update_shadow_word.inc

```
15  do {
16    const unsigned kAccessSize = 1 << kAccessSizeLog;
17    u64 *sp = &shadow_mem[idx];
18    old = LoadShadow(sp);

47    // Do the memory access intersect?
48    if (Shadow::TwoRangesIntersect(old, cur, kAccessSize)) {
49      if (Shadow::TidsAreEqual(old, cur))
50        break;
51      if (old.IsBothReadsOrAtomic(kAccessIsWrite, kIsAtomic))
52        break;
53      if (LIKELY(HappensBefore(old, thr)))
54        break;
55      goto RACE;
56    }
57    // The accesses do not intersect.
58    break;
59  } while (0);
```

```
175  void MemoryAccessImpl1(ThreadState *thr, uptr addr, int kAccessSizeLog,
176                         bool kAccessIsWrite, bool kIsAtomic, u64 *shadow_mem,
177                         Shadow cur) {
201    for (int idx = 0; idx < 4; idx++) {
202  #   include "tsan_update_shadow_word.inc"
203    }

229    if (LIKELY(stored))
230      return;
231    // choose a random candidate slot and replace it
232    StoreShadow(shadow_mem + (cur.epoch() % kShadowCnt), store_word);
233    return;
234  RACE:
235    HandleRace(thr, shadow_mem, cur, old);
236    return;
237  }
```

```
170  static inline bool HappensBefore(Shadow old, ThreadState *thr) {
171    return thr->clock.get(old.TidWithIgnore()) >= old.epoch();
172  }
```

Проверка: current_thread.clock[old.ThId] >= old.epoch
1 запись, и последние чтения из каждого потока
Будем хранить только последние 4 обращения к ячейке памяти



Thread Sanitizer – Data Race detector

Как реализовано - рантайм

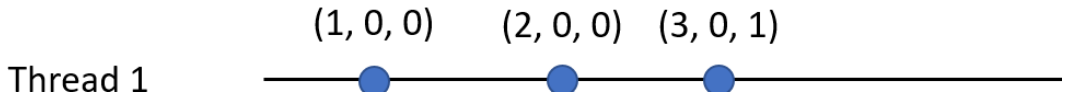
https://github.com/llvm/llvm-project/blob/main/compiler-rt/lib/tsan/rtl/tsan_rtl_access.cpp
https://github.com/llvm/llvm-project/blob/main/compiler-rt/lib/tsan/rtl/tsan_update_shadow_word.inc

```
15  do {
16    const unsigned kAccessSize = 1 << kAccessSizeLog;
17    u64 *sp = &shadow_mem[idx];
18    old = LoadShadow(sp);

47    // Do the memory access intersect?
48    if (Shadow::TwoRangesIntersect(old, cur, kAccessSize)) {
49      if (Shadow::TidsAreEqual(old, cur))
50        break;
51      if (old.IsBothReadsOrAtomic(kAccessIsWrite, kIsAtomic))
52        break;
53      if (LIKELY(HappensBefore(old, thr)))
54        break;
55      goto RACE;
56    }
57    // The accesses do not intersect.
58    break;
59  } while (0);
```

```
175  void MemoryAccessImpl1(ThreadState *thr, uptr addr, int kAccessSizeLog,
176                         bool kAccessIsWrite, bool kIsAtomic, u64 *shadow_mem,
177                         Shadow cur) {
201    for (int idx = 0; idx < 4; idx++) {
202  #   include "tsan_update_shadow_word.inc"
203    }

229    if (LIKELY(stored))
230      return;
231    // choose a random candidate slot and replace it
232    StoreShadow(shadow_mem + (cur.epoch() % kShadowCnt), store_word);
233    return;
234  RACE:
235    HandleRace(thr, shadow_mem, cur, old);
236    return;
237  }
```

```
170  static inline bool HappensBefore(Shadow old, ThreadState *thr) {
171    return thr->clock.get(old.TidWithIgnore()) >= old.epoch();
172  }
```

Проверка: current_thread.clock[old.ThId] >= old.epoch
1 запись, и последние чтения из каждого потока
Будем хранить только последние 4 обращения к ячейке памяти



Thread Sanitizer – Data Race detector

Как реализовано - рантайм

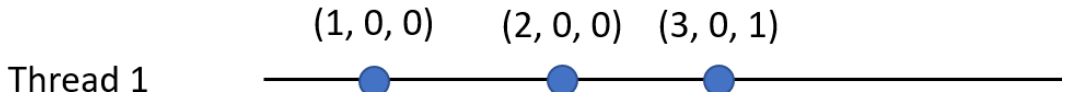
https://github.com/llvm/llvm-project/blob/main/compiler-rt/lib/tsan/rtl/tsan_rtl_access.cpp
https://github.com/llvm/llvm-project/blob/main/compiler-rt/lib/tsan/rtl/tsan_update_shadow_word.inc

```
15  do {
16    const unsigned kAccessSize = 1 << kAccessSizeLog;
17    u64 *sp = &shadow_mem[idx];
18    old = LoadShadow(sp);

47    // Do the memory access intersect?
48    if (Shadow::TwoRangesIntersect(old, cur, kAccessSize)) {
49      if (Shadow::TidsAreEqual(old, cur))
50        break;
51      if (old.IsBothReadsOrAtomic(kAccessIsWrite, kIsAtomic))
52        break;
53      if (LIKELY(HappensBefore(old, thr)))
54        break;
55      goto RACE;
56    }
57    // The accesses do not intersect.
58    break;
59  } while (0);
```

```
175  void MemoryAccessImpl1(ThreadState *thr, uptr addr, int kAccessSizeLog,
176                         bool kAccessIsWrite, bool kIsAtomic, u64 *shadow_mem,
177                         Shadow cur) {
201    for (int idx = 0; idx < 4; idx++) {
202  #  include "tsan_update_shadow_word.inc"
203    }

229    if (LIKELY(stored))
230      return;
231    // choose a random candidate slot and replace it
232    StoreShadow(shadow_mem + (cur.epoch() % kShadowCnt), store_word);
233    return;
234  RACE:
235    HandleRace(thr, shadow_mem, cur, old);
236    return;
237  }
```

```
170  static inline bool HappensBefore(Shadow old, ThreadState *thr) {
171    return thr->clock.get(old.TidWithIgnore()) >= old.epoch();
172  }
```

Проверка: current_thread.clock[old.ThId] >= old.epoch
1 запись, и последние чтения из каждого потока
Будем хранить только последние 4 обращения к ячейке памяти



Thread Sanitizer – Data Race detector

Как реализовано - рантайм

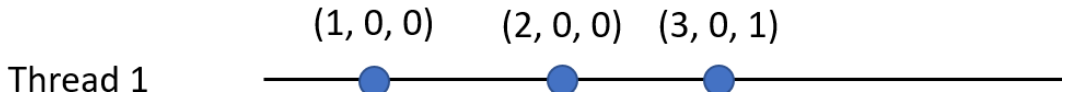
https://github.com/llvm/llvm-project/blob/main/compiler-rt/lib/tsan/rtl/tsan_rtl_access.cpp
https://github.com/llvm/llvm-project/blob/main/compiler-rt/lib/tsan/rtl/tsan_shadow.h

```
94   // Shadow (from most significant bit):
95   //   freed           : 1
96   //   tid             : kTidBits
97   //   is_atomic       : 1
98   //   is_read         : 1
99   //   size_log        : 2
100  //   addr0           : 3
101  //   epoch           : kClkBits
102  class Shadow : public FastState {
103   public:
```

```
175  void MemoryAccessImpl1(ThreadState *thr, uptr addr, int kAccessSizeLog,
176                         bool kAccessIsWrite, bool kIsAtomic, u64 *shadow_mem,
177                         Shadow cur) {
201    for (int idx = 0; idx < 4; idx++) {
202  #  include "tsan_update_shadow_word.inc"
203    }

229    if (LIKELY(stored))
230      return;
231    // choose a random candidate slot and replace it
232    StoreShadow(shadow_mem + (cur.epoch() % kShadowCnt), store_word);
233    return;
234  RACE:
235    HandleRace(thr, shadow_mem, cur, old);
236    return;
237  }
```

```
170  static inline bool HappensBefore(Shadow old, ThreadState *thr) {
171    return thr->clock.get(old.TidWithIgnore()) >= old.epoch();
172  }
```

Проверка: current_thread.clock[old.ThId] >= old.epoch
1 запись, и последние чтения из каждого потока
Будем хранить только последние 4 обращения к ячейке памяти



Thread Sanitizer – Data Race detector

Как реализовано - рантайм

https://github.com/llvm/llvm-project/blob/main/compiler-rt/lib/tsan/rtl/tsan_shadow.h

```
94   // Shadow (from most significant bit):
95   //   freed           : 1
96   //   tid             : kTidBits
97   //   is_atomic       : 1
98   //   is_read         : 1
99   //   size_log        : 2
100  //   addr0           : 3
101  //   epoch           : kClkBits
102  class Shadow : public FastState {
103   public:
```

```
23   class FastState {
24    public:
52     void IncrementEpoch() {
53       u64 old_epoch = epoch();
54       x_ += 1;
55       DCHECK_EQ(old_epoch + 1, epoch());
56       (void)old_epoch;
57     }
```

```
if (kCollectHistory) {
  fast_state.IncrementEpoch();
  thr->fast_state = fast_state;
  TraceAddEvent(thr, fast_state, EventTypeMop, pc);
  cur.IncrementEpoch();
}
```

Найдем где инкрементируется

Thread Sanitizer – Data Race detector

Как реализовано - рантайм

https://github.com/llvm/llvm-project/blob/main/compiler-rt/lib/tsan/rtl/tsan_shadow.h
https://github.com/llvm/llvm-project/blob/main/compiler-rt/lib/tsan/rtl/tsan_rtl_access.cpp

```
94   // Shadow (from most significant bit):
95   //   freed           : 1
96   //   tid             : kTidBits
97   //   is_atomic       : 1
98   //   is_read         : 1
99   //   size_log        : 2
100  //   addr0           : 3
101  //   epoch           : kClkBits
102  class Shadow : public FastState {
103   public:
```

```
23   class FastState {
24    public:
52     void IncrementEpoch() {
53       u64 old_epoch = epoch();
54       x_ += 1;
55       DCHECK_EQ(old_epoch + 1, epoch());
56       (void)old_epoch;
57     }
```

```
350  ALWAYS_INLINE USED void MemoryAccess(ThreadState *thr, uptr pc, uptr addr,
351                                       int kAccessSizeLog, bool kAccessIsWrite,
352                                       bool kIsAtomic) {
       if (kCollectHistory) {
         fast_state.IncrementEpoch();
         thr->fast_state = fast_state;
         TraceAddEvent(thr, fast_state, EventTypeMop, pc);
         cur.IncrementEpoch();
       }
```

Найдем где инкрементируется

Thread Sanitizer – Data Race detector

Как реализовано - рантайм

https://github.com/llvm/llvm-project/blob/main/compiler-rt/lib/tsan/rtl/tsan_shadow.h
https://github.com/llvm/llvm-project/blob/main/compiler-rt/lib/tsan/rtl/tsan_rtl_access.cpp

```
94   // Shadow (from most significant bit):
95   //   freed           : 1
96   //   tid             : kTidBits
97   //   is_atomic       : 1
98   //   is_read         : 1
99   //   size_log        : 2
100  //   addr0           : 3
101  //   epoch           : kClkBits
102  class Shadow : public FastState {
103   public:
```

```
23   class FastState {
24    public:
52     void IncrementEpoch() {
53       u64 old_epoch = epoch();
54       x_ += 1;
55       DCHECK_EQ(old_epoch + 1, epoch());
56       (void)old_epoch;
57     }
```

```
350  ALWAYS_INLINE USED void MemoryAccess(ThreadState *thr, uptr pc, uptr addr,
351                                       int kAccessSizeLog, bool kAccessIsWrite,
352                                       bool kIsAtomic) {
       if (kCollectHistory) {
         fast_state.IncrementEpoch();
         thr->fast_state = fast_state;
         TraceAddEvent(thr, fast_state, EventTypeMop, pc);
         cur.IncrementEpoch();
       }
```

```
261  static void AtomicStore(ThreadState *thr, uptr pc, volatile T *a, T v,
262                          morder mo) {

276    thr->fast_state.IncrementEpoch();
```

https://github.com/llvm/llvm-project/blob/main/compiler-rt/lib/tsan/rtl/tsan_interface_atomic.cpp

Найдем где инкрементируется

Thread Sanitizer – Data Race detector

Как реализовано - рантайм

https://github.com/llvm/llvm-project/blob/main/compiler-rt/lib/tsan/rtl/tsan_interface_atomic.cpp

```
261  static void AtomicStore(ThreadState *thr, uptr pc, volatile T *a, T v,
262                          morder mo) {
263    DCHECK(IsStoreOrder(mo));
264    MemoryAccess(thr, pc, (uptr)a, AccessSize<T>(), kAccessWrite | kAccessAtomic);
265    // This fast-path is critical for performance.
266    // Assume the access is atomic.
267    // Strictly saying even relaxed store cuts off release sequence,
268    // so must reset the clock.
269    if (!IsReleaseOrder(mo)) {
270      NoTsanAtomicStore(a, v, mo);
271      return;
272    }
273    __sync_synchronize();
274    SyncVar *s = ctx->metamap.GetSyncOrCreate(thr, pc, (uptr)a, false);
275    Lock l(&s->mtx);
276    thr->fast_state.IncrementEpoch();
277    // Can't increment epoch w/o writing to the trace as well.
278    TraceAddEvent(thr, thr->fast_state, EventTypeMop, 0);
279    ReleaseStoreImpl(thr, pc, &s->clock);
280    NoTsanAtomicStore(a, v, mo);
281  }
```

Thread Sanitizer – Data Race detector

Как реализовано - рантайм

https://github.com/llvm/llvm-project/blob/main/compiler-rt/lib/tsan/rtl/tsan_interface_atomic.cpp

```
261  static void AtomicStore(ThreadState *thr, uptr pc, volatile T *a, T v,
262                          morder mo) {
263    DCHECK(IsStoreOrder(mo));
264    MemoryAccess(thr, pc, (uptr)a, AccessSize<T>(), kAccessWrite | kAccessAtomic);
265    // This fast-path is critical for performance.
266    // Assume the access is atomic.
267    // Strictly saying even relaxed store cuts off release sequence,
268    // so must reset the clock.
269    if (!IsReleaseOrder(mo)) {
270      NoTsanAtomicStore(a, v, mo);
271      return;
272    }
273    __sync_synchronize();
274    SyncVar *s = ctx->metamap.GetSyncOrCreate(thr, pc, (uptr)a, false);
275    Lock l(&s->mtx);
276    thr->fast_state.IncrementEpoch();
277    // Can't increment epoch w/o writing to the trace as well.
278    TraceAddEvent(thr, thr->fast_state, EventTypeMop, 0);
279    ReleaseStoreImpl(thr, pc, &s->clock);
280    NoTsanAtomicStore(a, v, mo);
281  }
```

Thread Sanitizer – Data Race detector

Как реализовано - рантайм

https://github.com/llvm/llvm-project/blob/main/compiler-rt/lib/tsan/rtl/tsan_interface_atomic.cpp

```
261  static void AtomicStore(ThreadState *thr, uptr pc, volatile T *a, T v,
262                          morder mo) {
263    DCHECK(IsStoreOrder(mo));
264    MemoryAccess(thr, pc, (uptr)a, AccessSize<T>(), kAccessWrite | kAccessAtomic);
265    // This fast-path is critical for performance.
266    // Assume the access is atomic.
267    // Strictly saying even relaxed store cuts off release sequence,
268    // so must reset the clock.
269    if (!IsReleaseOrder(mo)) {
270      NoTsanAtomicStore(a, v, mo);
271      return;
272    }
273    __sync_synchronize();
274    SyncVar *s = ctx->metamap.GetSyncOrCreate(thr, pc, (uptr)a, false);
275    Lock l(&s->mtx);
276    thr->fast_state.IncrementEpoch();
277    // Can't increment epoch w/o writing to the trace as well.
278    TraceAddEvent(thr, thr->fast_state, EventTypeMop, 0);
279    ReleaseStoreImpl(thr, pc, &s->clock);
280    NoTsanAtomicStore(a, v, mo);
281  }
```

Thread Sanitizer – Data Race detector

Как реализовано - рантайм

https://github.com/llvm/llvm-project/blob/main/compiler-rt/lib/tsan/rtl/tsan_interface_atomic.cpp
https://github.com/llvm/llvm-project/blob/main/compiler-rt/lib/tsan/rtl/tsan_sync.h

```
261  static void AtomicStore(ThreadState *thr, uptr pc, volatile T *a, T v,
262                          morder mo) {
263    DCHECK(IsStoreOrder(mo));
264    MemoryAccess(thr, pc, (uptr)a, AccessSize<T>(), kAccessWrite | kAccessAtomic);
265    // This fast-path is critical for performance.
266    // Assume the access is atomic.
267    // Strictly saying even relaxed store cuts off release sequence,
268    // so must reset the clock.
269    if (!IsReleaseOrder(mo)) {
270      NoTsanAtomicStore(a, v, mo);
271      return;
272    }
273      __sync_synchronize();
274    SyncVar *s = ctx->metamap.GetSyncOrCreate(thr, pc, (uptr)a, false);
275    Lock l(&s->mtx);
276    thr->fast_state.IncrementEpoch();
277    // Can't increment epoch w/o writing to the trace as well.
278    TraceAddEvent(thr, thr->fast_state, EventTypeMop, 0);
279    ReleaseStoreImpl(thr, pc, &s->clock);
280    NoTsanAtomicStore(a, v, mo);
281  }
```

```
49   // SyncVar is a descriptor of a user synchronization object
50   // (mutex or an atomic variable).
51   struct SyncVar {
52     SyncVar();
53
67     SyncClock clock;

23   // The clock that lives in sync variables (mutexes, atomics, etc).
24   class SyncClock {
25    public:
26     SyncClock();
27     ~SyncClock();

134   class ThreadClock {
135    public:

221     // Number of active elements in the clk_ table (the rest is zeros).
222     uptr nclk_;
223     u64 clk_[kMaxTidInClock];  // Fixed size vector clock.
```

https://github.com/llvm/llvm-project/blob/main/compiler-rt/lib/tsan/rtl/tsan_clock.h

Thread Sanitizer – Data Race detector

Как реализовано - рантайм

https://github.com/llvm/llvm-project/blob/main/compiler-rt/lib/tsan/rtl/tsan_interface_atomic.cpp
https://github.com/llvm/llvm-project/blob/main/compiler-rt/lib/tsan/rtl/tsan_sync.h

```
261  static void AtomicStore(ThreadState *thr, uptr pc, volatile T *a, T v,
262                          morder mo) {
263    DCHECK(IsStoreOrder(mo));
264    MemoryAccess(thr, pc, (uptr)a, AccessSize<T>(), kAccessWrite | kAccessAtomic);
265    // This fast-path is critical for performance.
266    // Assume the access is atomic.
267    // Strictly saying even relaxed store cuts off release sequence,
268    // so must reset the clock.
269    if (!IsReleaseOrder(mo)) {
270      NoTsanAtomicStore(a, v, mo);
271      return;
272    }
273    __sync_synchronize();
274    SyncVar *s = ctx->metamap.GetSyncOrCreate(thr, pc, (uptr)a, false);
275    Lock l(&s->mtx);
276    thr->fast_state.IncrementEpoch();
277    // Can't increment epoch w/o writing to the trace as well.
278    TraceAddEvent(thr, thr->fast_state, EventTypeMop, 0);
279    ReleaseStoreImpl(thr, pc, &s->clock);
280    NoTsanAtomicStore(a, v, mo);
281  }
```

```
49   // SyncVar is a descriptor of a user synchronization object
50   // (mutex or an atomic variable).
51   struct SyncVar {
52     SyncVar();
53
67     SyncClock clock;
```

```
23   // The clock that lives in sync variables (mutexes, atomics, etc).
24   class SyncClock {
25    public:
26     SyncClock();
27     ~SyncClock();
```

```
134   class ThreadClock {
135    public:

221     // Number of active elements in the clk_ table (the rest is zeros).
222     uptr nclk_;
223     u64 clk_[kMaxTidInClock];  // Fixed size vector clock.
```

https://github.com/llvm/llvm-project/blob/main/compiler-rt/lib/tsan/rtl/tsan_clock.h

Thread Sanitizer – Data Race detector

Как реализовано - рантайм

https://github.com/llvm/llvm-project/blob/main/compiler-rt/lib/tsan/rtl/tsan_interface_atomic.cpp
https://github.com/llvm/llvm-project/blob/main/compiler-rt/lib/tsan/rtl/tsan_sync.h

```
261  static void AtomicStore(ThreadState *thr, uptr pc, volatile T *a, T v,
262                          morder mo) {
263    DCHECK(IsStoreOrder(mo));
264    MemoryAccess(thr, pc, (uptr)a, AccessSize<T>(), kAccessWrite | kAccessAtomic);
265    // This fast-path is critical for performance.
266    // Assume the access is atomic.
267    // Strictly saying even relaxed store cuts off release sequence,
268    // so must reset the clock.
269    if (!IsReleaseOrder(mo)) {
270      NoTsanAtomicStore(a, v, mo);
271      return;
272    }
273    __sync_synchronize();
274    SyncVar *s = ctx->metamap.GetSyncOrCreate(thr, pc, (uptr)a, false);
275    Lock l(&s->mtx);
276    thr->fast_state.IncrementEpoch();
277    // Can't increment epoch w/o writing to the trace as well.
278    TraceAddEvent(thr, thr->fast_state, EventTypeMop, 0);
279    ReleaseStoreImpl(thr, pc, &s->clock);
280    NoTsanAtomicStore(a, v, mo);
281  }
```

```
49  // SyncVar is a descriptor of a user synchronization object
50  // (mutex or an atomic variable).
51  struct SyncVar {
52    SyncVar();
53
67    SyncClock clock;
```

```
23  // The clock that lives in sync variables (mutexes, atomics, etc).
24  class SyncClock {
25   public:
26    SyncClock();
27    ~SyncClock();
```

```
134   class ThreadClock {
135    public:

221     // Number of active elements in the clk_ table (the rest is zeros).
222     uptr nclk_;
223     u64 clk_[kMaxTidInClock];  // Fixed size vector clock.
```

https://github.com/llvm/llvm-project/blob/main/compiler-rt/lib/tsan/rtl/tsan_clock.h

Thread Sanitizer – Data Race detector

Как реализовано - рантайм

https://github.com/llvm/llvm-project/blob/main/compiler-rt/lib/tsan/rtl/tsan_interface_atomic.cpp
https://github.com/llvm/llvm-project/blob/main/compiler-rt/lib/tsan/rtl/tsan_sync.h

```
261  static void AtomicStore(ThreadState *thr, uptr pc, volatile T *a, T v,
262                          morder mo) {
263    DCHECK(IsStoreOrder(mo));
264    MemoryAccess(thr, pc, (uptr)a, AccessSize<T>(), kAccessWrite | kAccessAtomic);
265    // This fast-path is critical for performance.
266    // Assume the access is atomic.
267    // Strictly saying even relaxed store cuts off release sequence,
268    // so must reset the clock.
269    if (!IsReleaseOrder(mo)) {
270      NoTsanAtomicStore(a, v, mo);
271      return;
272    }
273    __sync_synchronize();
274    SyncVar *s = ctx->metamap.GetSyncOrCreate(thr, pc, (uptr)a, false);
275    Lock l(&s->mtx);
276    thr->fast_state.IncrementEpoch();
277    // Can't increment epoch w/o writing to the trace as well.
278    TraceAddEvent(thr, thr->fast_state, EventTypeMop, 0);
279    ReleaseStoreImpl(thr, pc, &s->clock);
280    NoTsanAtomicStore(a, v, mo);
281  }
```

```
513  void ReleaseStoreImpl(ThreadState *thr, uptr pc, SyncClock *c) {
514    if (thr->ignore_sync)
515      return;
516    thr->clock.set(thr->fast_state.epoch());
517    thr->fast_synch_epoch = thr->fast_state.epoch();
518    thr->clock.ReleaseStore(&thr->proc()->clock_cache, c);
519  }
```

Thread Sanitizer – Data Race detector

Как реализовано - рантайм

https://github.com/llvm/llvm-project/blob/main/compiler-rt/lib/tsan/rtl/tsan_interface_atomic.cpp
https://github.com/llvm/llvm-project/blob/main/compiler-rt/lib/tsan/rtl/tsan_sync.h

```
261  static void AtomicStore(ThreadState *thr, uptr pc, volatile T *a, T v,
262                          morder mo) {
263    DCHECK(IsStoreOrder(mo));
264    MemoryAccess(thr, pc, (uptr)a, AccessSize<T>(), kAccessWrite | kAccessAtomic);
265    // This fast-path is critical for performance.
266    // Assume the access is atomic.
267    // Strictly saying even relaxed store cuts off release sequence,
268    // so must reset the clock.
269    if (!IsReleaseOrder(mo)) {
270      NoTsanAtomicStore(a, v, mo);
271      return;
272    }
273    __sync_synchronize();
274    SyncVar *s = ctx->metamap.GetSyncOrCreate(thr, pc, (uptr)a, false);
275    Lock l(&s->mtx);
276    thr->fast_state.IncrementEpoch();
277    // Can't increment epoch w/o writing to the trace as well.
278    TraceAddEvent(thr, thr->fast_state, EventTypeMop, 0);
279    ReleaseStoreImpl(thr, pc, &s->clock);
280    NoTsanAtomicStore(a, v, mo);
281  }
```

```
513  void ReleaseStoreImpl(ThreadState *thr, uptr pc, SyncClock *c) {
514    if (thr->ignore_sync)
515      return;
516    thr->clock.set(thr->fast_state.epoch());
517    thr->fast_synch_epoch = thr->fast_state.epoch();
518    thr->clock.ReleaseStore(&thr->proc()->clock_cache, c);
519  }

//   void ThreadClock::ReleaseStore(SyncClock *dst) const {
//     for (int i = 0; i < kMaxThreads; i++)
//       dst->clock[i] = clock[i];
//   }
```

Thread Sanitizer – Data Race detector

Как реализовано - рантайм

https://github.com/llvm/llvm-project/blob/main/compiler-rt/lib/tsan/rtl/tsan_interface_atomic.cpp
https://github.com/llvm/llvm-project/blob/main/compiler-rt/lib/tsan/rtl/tsan_sync.h

```
261  static void AtomicStore(ThreadState *thr, uptr pc, volatile T *a, T v,
262                          morder mo) {
263    DCHECK(IsStoreOrder(mo));
264    MemoryAccess(thr, pc, (uptr)a, AccessSize<T>(), kAccessWrite | kAccessAtomic);
265    // This fast-path is critical for performance.
266    // Assume the access is atomic.
267    // Strictly saying even relaxed store cuts off release sequence,
268    // so must reset the clock.
269    if (!IsReleaseOrder(mo)) {
270      NoTsanAtomicStore(a, v, mo);
271      return;
272    }
273    __sync_synchronize();
274    SyncVar *s = ctx->metamap.GetSyncOrCreate(thr, pc, (uptr)a, false);
275    Lock l(&s->mtx);
276    thr->fast_state.IncrementEpoch();
277    // Can't increment epoch w/o writing to the trace as well.
278    TraceAddEvent(thr, thr->fast_state, EventTypeMop, 0);
279    ReleaseStoreImpl(thr, pc, &s->clock);
280    NoTsanAtomicStore(a, v, mo);
281  }
```

```
513  void ReleaseStoreImpl(ThreadState *thr, uptr pc, SyncClock *c) {
514    if (thr->ignore_sync)
515      return;
516    thr->clock.set(thr->fast_state.epoch());
517    thr->fast_synch_epoch = thr->fast_state.epoch();
518    thr->clock.ReleaseStore(&thr->proc()->clock_cache, c);
519  }
```

```
//   void ThreadClock::ReleaseStore(SyncClock *dst) const {
//     for (int i = 0; i < kMaxThreads; i++)
//       dst->clock[i] = clock[i];
//   }

//   void ThreadClock::acquire(const SyncClock *src) {
//     for (int i = 0; i < kMaxThreads; i++)
//       clock[i] = max(clock[i], src->clock[i]);
//   }
```

Thread Sanitizer – Data Race detector

Нюансы

1. Нет false positives*

Thread Sanitizer – Data Race detector

Нюансы

1. Нет false positives* (* -- при условии, что TSan “видит” все примитивы синхронизации – может либо перехватить, либо инструментировать)

Thread Sanitizer – Data Race detector

Нюансы

1. Нет false positives* (* -- при условии, что TSan “видит” все примитивы синхронизации – может либо перехватить, либо инструментировать)
2. Можно менять рантайм TSan независимо от компилятора (в случае ASan – не всегда)

Thread Sanitizer – Data Race detector

Нюансы

1. Нет false positives* (* -- при условии, что TSan “видит” все примитивы синхронизации – может либо перехватить, либо инструментировать)
2. Можно менять рантайм TSan независимо от компилятора (в случае ASan – не всегда)
3. Можно использовать инструментирование под свои нужды

Thread Sanitizer – Data Race detector

Нюансы

1. Нет false positives* (* -- при условии, что TSan “видит” все примитивы синхронизации – может либо перехватить, либо инструментировать)
2. Можно менять рантайм TSan независимо от компилятора (в случае ASan – не всегда)
3. Можно использовать инструментирование под свои нужды
4. Кажется инструментирование TSan и инструментирование для coverage взаимно избыточно.

Thread Sanitizer – Data Race detector

Нюансы

1. Нет false positives* (* -- при условии, что TSan “видит” все примитивы синхронизации – может либо перехватить, либо инструментировать)
2. Можно менять рантайм TSan независимо от компилятора (в случае ASan – не всегда)
3. Можно использовать инструментирование под свои нужды
4. Кажется инструментирование TSan и инструментирование для coverage взаимно избыточно.
5. На самом деле обнаруживает не только data races, но и некорректное обращение с мьютексами, иногда дедлоки

Thread Sanitizer – Data Race detector

Нюансы

1. Нет false positives* (* -- при условии, что TSan “видит” все примитивы синхронизации – может либо перехватить, либо инструментировать)
2. Можно менять рантайм TSan независимо от компилятора (в случае ASan – не всегда)
3. Можно использовать инструментирование под свои нужды
4. Кажется инструментирование TSan и инструментирование для coverage взаимно избыточно.
5. На самом деле обнаруживает не только data races, но и некорректное обращение с мьютексами, иногда дедлоки
6. Есть (возможны всегда) false negatives

Thread Sanitizer – Data Race detector

Нюансы

1. Нет false positives* (* -- при условии, что TSan “видит” все примитивы синхронизации – может либо перехватить, либо инструментировать)
2. Можно менять рантайм TSan независимо от компилятора (в случае ASan – не всегда)
3. Можно использовать инструментирование под свои нужды
4. Кажется инструментирование TSan и инструментирование для coverage взаимно избыточно.
5. На самом деле обнаруживает не только data races, но и некорректное обращение с мьютексами, иногда дедлоки
6. Есть (возможны всегда) false negatives
7. Не пытается ломать ваш код. Может найти data race только если он реально случился во время работы

Thread Sanitizer – Data Race detector

Нюансы

1. Нет false positives* (* -- при условии, что TSan “видит” все примитивы синхронизации – может либо перехватить, либо инструментировать)
2. Можно менять рантайм TSan независимо от компилятора (в случае ASan – не всегда)
3. Можно использовать инструментирование под свои нужды
4. Кажется инструментирование TSan и инструментирование для coverage взаимно избыточно.
5. На самом деле обнаруживает не только data races, но и некорректное обращение с мьютексами, иногда дедлоки
6. Есть (возможны всегда) false negatives
7. Не пытается ломать ваш код. Может найти data race только если он реально случился во время работы
8. TSan v3. Многое из рассказанного станет не актуальным.

Спасибо!

Алексей Веселовский

Align Technology

Telegram: @I_vlxy_I
E-mail: aveselovskiy@aligntech.com