



ЭВОЛЮЦИЯ МОДЕЛЕЙ ПАМЯТИ

МИР Plat.Form



Обо мне

[linkedin.com/in/alantsov](https://www.linkedin.com/in/alantsov)

Мир Plat.Form

mir-platform.ru

Статьи и примеры

github.com/lantalex/mm-evolution



О ЧЕМ ДОКЛАД?

Модели памяти:

- ❏ **x86-64 / ARMv8 / RISC-V**
«железо»
- ❏ **Java / C++ / Rust / Go**
языки программирования
- ❏ **Linux Kernel Memory Model**
ядро Linux
- ❏ **LLVM**
компиляторная инфраструктура
- ❏ **RC11 / Promising Semantics**
академические работы

Почему так сложно?
Зачем это всё?

План

- ❖ **С чего все началось?**

первые статьи и модели памяти в «железе»

- ❖ **Появление моделей памяти в ЯП**

JMM как первая (не)успешная модель памяти; другие языки

- ❖ **Академические работы**

какую проблему пытаются решить?

- ❖ **Как жить дальше?**

Глава 1. Откуда всё берется

первые статьи + «железо»

Первые многопроцессорные системы

1967 IBM System/360 Model 65MP

1971 C.mmp

1972 ILLIAC IV

Проблема : как дать разработчикам гарантии и правила относительно того, как работать с оборудованием?

Sequential Consistency aka **SC**

1979

«How to Make a Multiprocessor Computer That Correctly Executes Multiprocess Programs»

by L. Lamport

SC : «Результат любого исполнения не отличим от случая, когда все операции на всех процессорах исполняются в некотором последовательном порядке, и операции на конкретном процессоре исполняются в порядке, обозначенном программой»

Sequential Consistency aka **SC**

Операции выполняются последовательно одна за другой, иногда переключаясь на другой поток CPU

```
MOV [a] ← 0 ; MOV [b] ← 0
```

vCPU 1

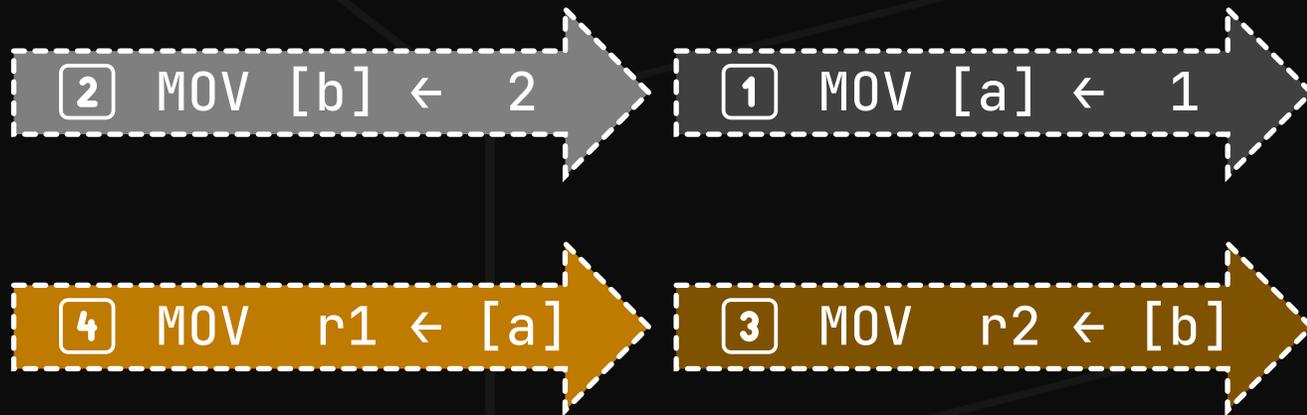
```
MOV [a] ← 1  
MOV [b] ← 2
```

vCPU 2

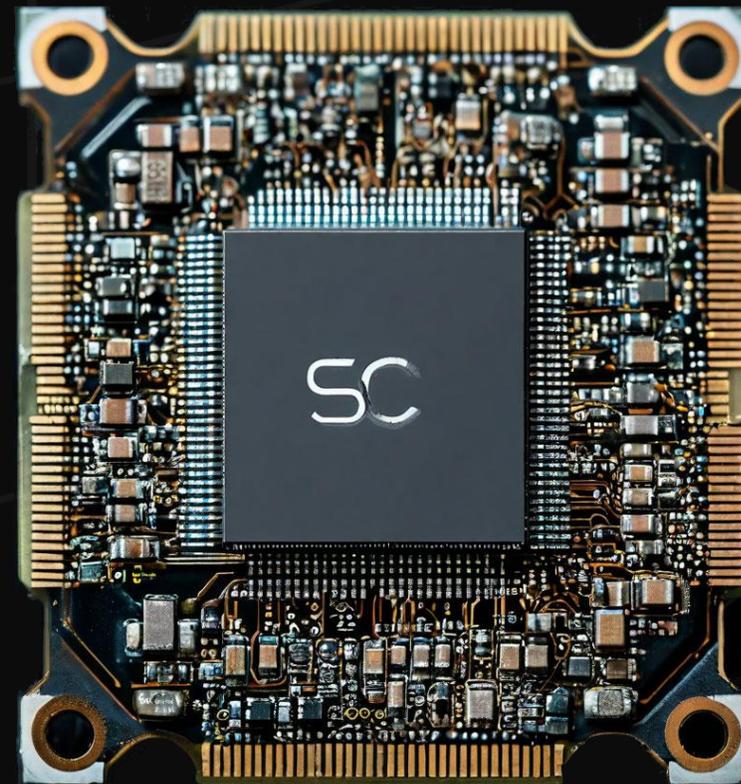
```
MOV r2 ← [b]  
MOV r1 ← [a]
```

Возможен ли исход: $r2 = 2$; $r1 = 0$?

Возможен ли исход: $r2 = 2; r1 = 0$?

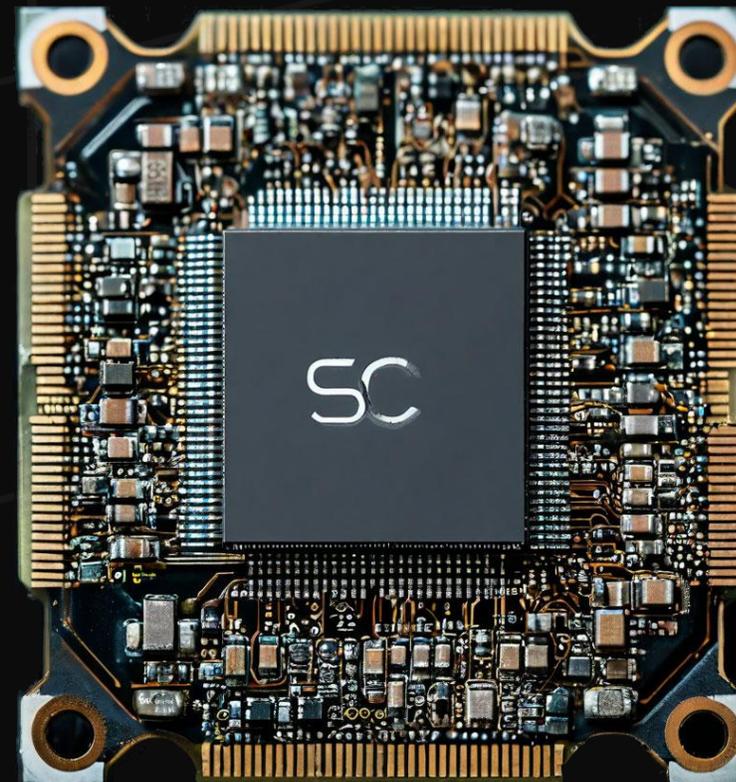
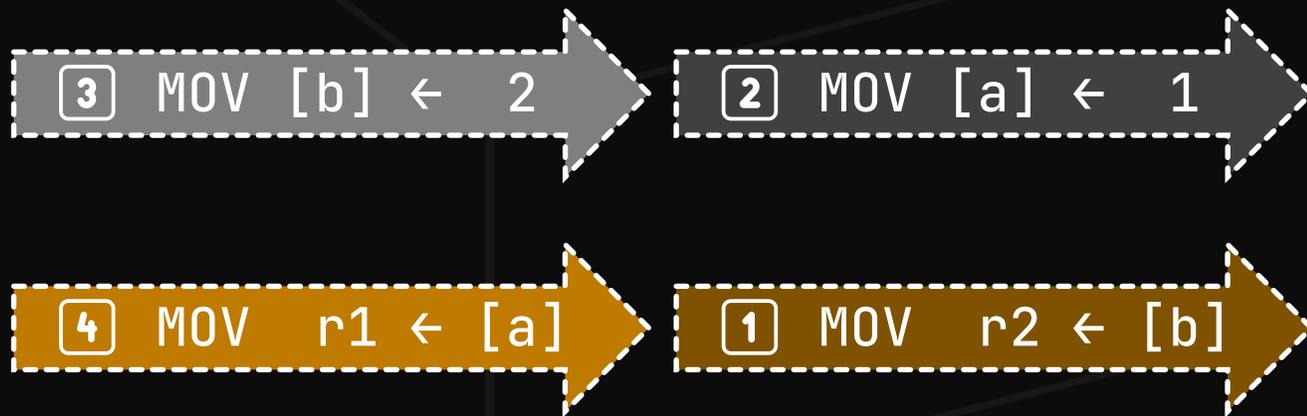


Возможен ли исход: $r_2 = 2$; $r_1 = 0$?



Возможен ли исход: $r2 = 2; r1 = 0$?

SC : нет



Проблема: SC не бесплатен

❖ Хотим ослабить модель, но *как* ?

1990

«Weak Ordering – A New Definition»

by S. Adve, M. Hill

❖ **SC-DRF** : если нет гонок по данным – то исходы такие же, как в **SC**

❖ Чтобы не было гонок – железо предоставляет *synchronization operations*

❖ **SC-DRF** : «a contract between software and hardware»

x86

Pentium Pro

никаких упоминаний о модели памяти в спецификации

разработка Linux Kernel / Plan 9

неформальные объяснения в публичных обсуждениях

Pentium D

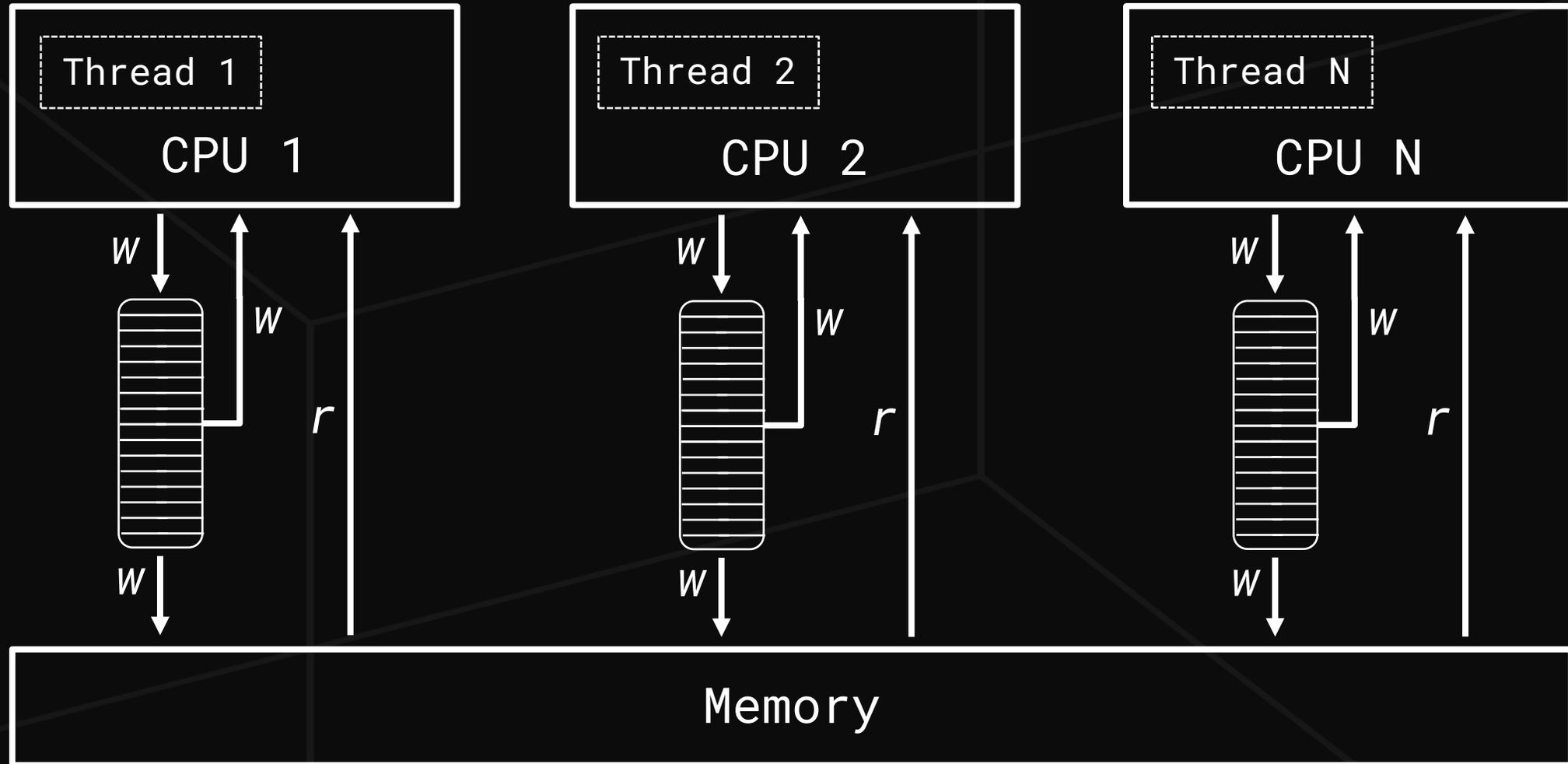
Intel 64 Architecture Memory Ordering White Paper

TLO+CC: total lock order + causal consistency

«A Better x86 Memory Model: x86-TSO»

by S. Owens, S. Sarkar, P. Sewell

x86-TSO: ментальная модель



x86-TSO: message passing

```
MOV [a] ← 0 ; MOV [b] ← 0
```

CPU 1

```
MOV [a] ← 1  
MOV [b] ← 2
```

CPU 2

```
MOV r2 ← [b]  
MOV r1 ← [a]
```

Возможен ли исход: $r2 = 2; r1 = 0$?

SC : нет

x86-TSO : нет

x86-TSO: store buffer

```
MOV [a] ← 0 ; MOV [b] ← 0
```

CPU 1

```
MOV [a] ← 1  
MOV r2 ← [b]
```

CPU 2

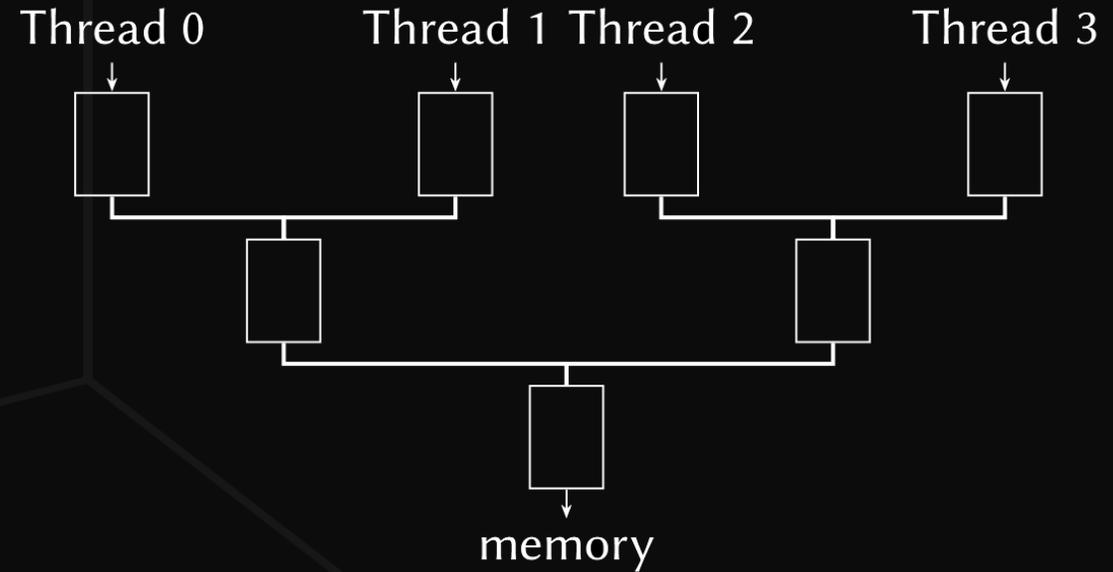
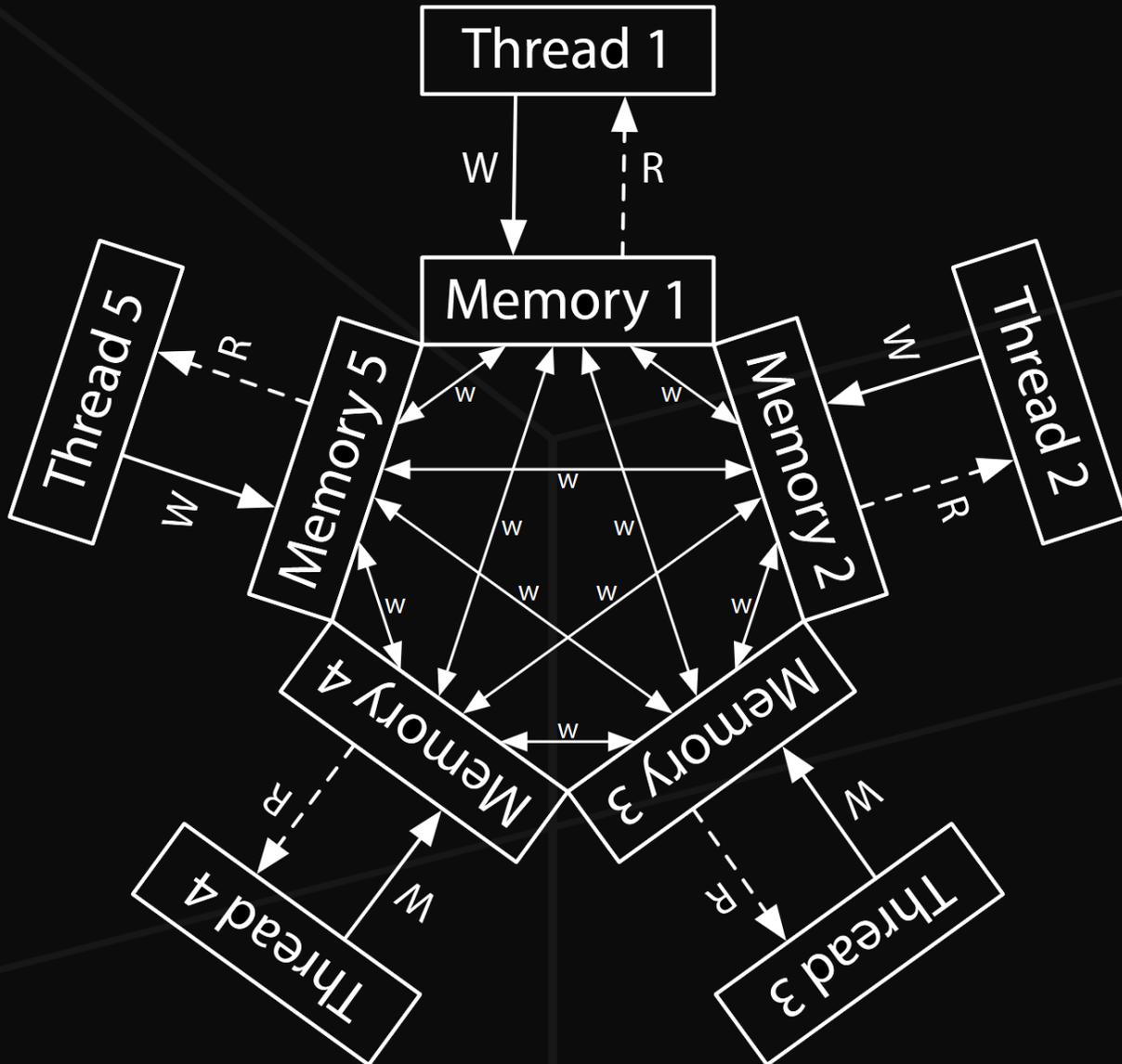
```
MOV [b] ← 1  
MOV r1 ← [a]
```

Возможен ли исход: $r2 = 0$; $r1 = 0$?

SC : нет

x86-TSO : да

Power / ARM: ментальная модель



Power / ARM: message passing

MOV [a] ← 0 ; MOV [b] ← 0

CPU 1

MOV [a] ← 1
MOV [b] ← 2

CPU 2

MOV r2 ← [b]
MOV r1 ← [a]

Возможен ли исход: r2 = 2; r1 = 0 ?

SC : нет

x86-TSO : нет

Power / ARM : да

Когерентность

Все операции по одной конкретной фиксированной локации памяти (*aka переменной*) выполняются в глобальном порядке

Когерентность



Возможен ли исход:

r1 = 1; r2 = 2; r3 = 2; r4 = 1 ?

SC / x86-TSO / Power / ARM : нет

Глава 2. Языки программирования Java и не только

Когерентность

`int x = 0`

Thread 1

`x = 1`

Thread 2

`x = 2`

Thread 3

`r1 = x`
`r2 = x`

Thread 4

`r3 = x`
`r4 = x`

Возможен ли исход:

`r1 = 1; r2 = 2; r3 = 2; r4 = 1 ?`

SC / x86-TSO / Power / ARM: нет
языки программирования: да

Адаптация **SC-DRF** для ЯП

- ❖ Исходно: «a contract between software and hardware»
- ❖ Нужен контракт между ЯП и прикладным разработчиком
- ❖ Можно определить **SC-DRF** для ЯП !
- ❖ Используем ***synchronization operations*** для избегания гонок → **SC**



Java : первая неудачная попытка

- ❖ Появление модели: Java Language Specification (1996)
- ❖ **SC-DRF** не использовалась
- ❖ Когерентное чтение запись через **volatile**
– но никакого **happens-before**

```
int x = 0  
volatile int flag = 0
```

Thread 1

```
x = 1  
flag = 1
```

Thread 2

```
while (flag == 0) { sleep() }  
assert x == 1
```



Java : исправленная модель (2004)

- ❖ Основана на **SC-DRF**
- ❖ Определяются **synchronization operations**, на основании которых строится **happens-before**
- ❖ Специальная семантика для **final**-полей
- ❖ Используя **happens-before**, гарантируем, что гонок нет
- ❖ Определяется семантика для программ с гонками



Проблемы модели памяти Java

«On the Validity of Program Transformations in the Java Memory Model»

by J. Ševčík, D. Aspinall

«Memory Models: A Case For Rethinking Parallel Languages and Hardware»

by Adve, H. Boehm

«Compiling Volatile Correctly in Java»

by S. Liu, J. Bender, J. Palsberg

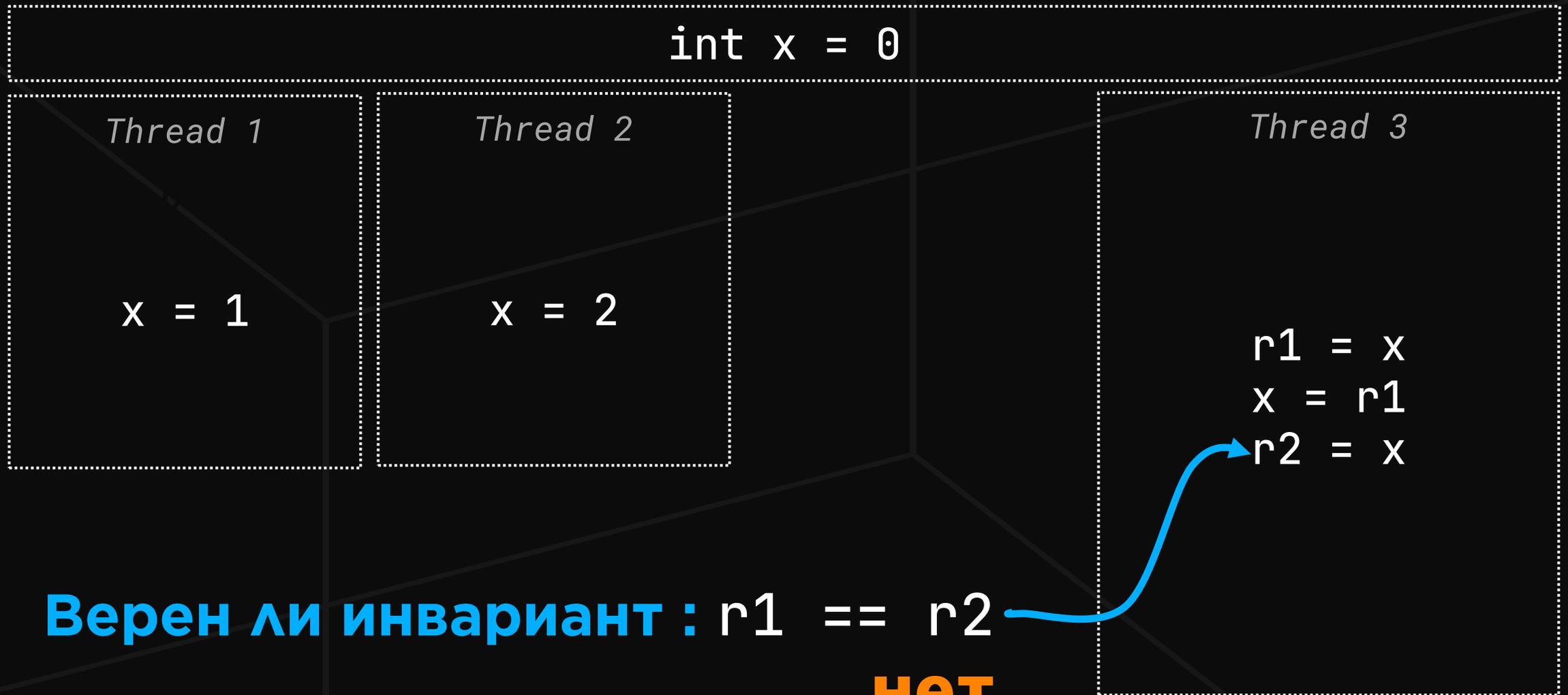


Redundant Write After Read Elimination





Redundant Write After Read Elimination





Redundant Write After Read Elimination

`int x = 0`

Thread 1

`lock a`

`x = 1`

`unlock a`

Thread 2

`lock b`

`x = 2`

`unlock b`

Thread 3

`lock a`

`lock b`

`r1 = x`

`r2 = x`

`unlock b`

`unlock a`

Верен ли инвариант : `r1 == r2`

нет



Redundant Write After Read Elimination

`int x = 0`

Thread 1

`lock a`

`x = 1`

`unlock a`

Thread 2

`lock b`

`x = 2`

`unlock b`

Thread 3

`lock a`

`lock b`

`r1 = x`

`x = r1`

`r2 = x`

`unlock b`

`unlock a`

Верен ли инвариант : `r1 == r2`

Да!



Redundant Write After Read Elimination

`int x = 0`

Thread 1

`lock a`

`x = 1`

`unlock a`

Thread 2

`lock b`

`x = 2`

`unlock b`

Thread 3

`lock a`

`lock b`

`r1 = x`

`x = r1`

`r2 = x`

`unlock b`

`unlock a`

**Компилятор: могу ли я
выкинуть `x = r1` ?**



Проблемы модели памяти Java

- ❖ Насколько сильно реализации ограничены в оптимизациях?
- ❖ [JDK-8262877](#)
«PowerPC sequential consistency problem: volatile stores are too weak»
- ❖ Насколько реальные реализации строже / слабее своих моделей?
- ❖ **VarHandle** : *opaque* и *acquire-release* нарушают принцип **SC-DRF** – как вписать их в существующую модель?



Модель памяти C++ (2011)

- ❏ Основана на **SC-DRF**,
требуется явное использование ***std::atomic***
- ❏ Гонка не по ***std::atomic*** ? ***Undefined behavior***
- ❏ Пример ***UB*** : логирование ситуации гонки
- ❏ Разные режимы доступа к памяти :
memory_order_relaxed
memory_order_consume
memory_order_acquire
memory_order_release
memory_order_seq_cst



Проблемы модели памяти

«Common Compiler Optimisations are Invalid in the C11 Memory Model and what we can do about it»

by V. Vafeiadis, T. Balabonski, S. Chakraborty

```
std::atomic<int> x = 0  
std::atomic<int> y = 0
```

Thread 1

```
if (x.load(relaxed) == 1)  
    y.store(1, relaxed)
```

Thread 2

```
if (y.load(relaxed) == 1)  
    x.store(1, relaxed)
```

Возможен ли исход: $x = 1; y = 1$?

C++: Да, out-of-thin-air (OoTA)



Проблемы модели памяти

```
int a = 0  
std::atomic<int> x = 0  
std::atomic<int> y = 0
```

Thread 0

```
a = 1
```

Thread 1

```
if (x.load(rlx) == 1)  
  if (a == 1)  
    y.store(1, rlx)
```

Thread 2

```
if (y.load(rlx) == 1)  
  x.store(1, rlx)
```

Единственный допустимый исход:

x = 0 ; y = 0



Проблемы модели памяти

```
int a = 0
std::atomic<int> x = 0
std::atomic<int> y = 0
```

Thread 0

`a = 1`

Thread 1

```
if (x.load(rlx) == 1)
  if (a == 1)
    y.store(1, rlx)
```

Thread 2

```
if (y.load(rlx) == 1)
  x.store(1, rlx)
```

Sequentialisation transformation : $C1 \parallel C2 \rightarrow \{ C1; C2; \}$



Проблемы модели памяти

```
int a = 0
std::atomic<int> x = 0
std::atomic<int> y = 0
```

Thread 0

Thread 1

```
a = 1
if (x.load(rlx) == 1)
  if (a == 1)
    y.store(1, rlx)
```

Thread 2

```
if (y.load(rlx) == 1)
  x.store(1, rlx)
```

Sequentialisation transformation : $C1 \parallel C2 \rightarrow \{ C1; C2; \}$

получили OoTA



Rust

- ❖ «Rust does not yet have a defined memory model»
- ❖ Модель памяти «как в C++», `std::sync::atomic`
- ❖ Гонки по данным запрещены* на уровне компилятора благодаря концепции владения
- ❖ Обошли через `unsafe`? *Undefined behavior*
- ❖ Насколько модель памяти `Linux Kernel` совместима с моделью памяти Rust?
(например, *CONTROL DEPENDENCIES*)



Go

- ❖ «Don't be clever» : **SC-DRF**
- ❖ **Есть гонки? Неформально, или остановка через race detector, или чтение вернет какое-то значение**
- ❖ **ОoTA запрещен – некоторые компиляторные оптимизации запрещены**
- ❖ **Конкуренция через координацию, а не разделяемые данные**

Глава 3. Развитие моделей

Текущие модели

Используют **SC-DRF**

Нет точных гарантий для ситуаций с гонками! 

И эти гарантии не будут бесплатными... 

«Bounding data races in space and time»

by S. Dolan, K.C. Sivaramakrishnan, A. Madhavapeddy

Есть гонки?

Нужно рассматривать всю программу **глобально** ;
локальные рассуждения **невозможны**

2018



Невозможность локальных рассуждений

Thread θ

$b = a + 10$

Условие:

нет других операций с b , a

ожидаемый инвариант: $b == a + 10$



Невозможность локальных рассуждений

```
int c = 0
```

Thread 0

```
c = a + 10
```

```
//some code
```

```
b = a + 10
```

Thread 1

```
c = 1
```

Условие:

нет других операций с b, a

ожидаемый инвариант: $b == a + 10$



Невозможность локальных рассуждений

```
int c = 0
```

Thread 0

```
t = a + 10  
c = t  
//some code  
b = t
```

Thread 1

```
c = 1
```

Условие:

нет других операций с b, а

ожидаемый инвариант: $b == a + 10$



Невозможность локальных рассуждений

```
int c = 0
```

Thread 0

```
t = a + 10  
c = t  
//some code  
b = c
```

Thread 1

```
c = 1
```

Условие:

нет других операций с b, а

ожидаемый инвариант: $b == a + 10$



Невозможность локальных рассуждений

```
int c = 0
```

Thread 0

```
t = a + 10  
c = t  
//some code  
b = c
```

Thread 1

```
c = 1
```

Условие:

нет других операций с b, а

нет инварианта : $b == a + 10$



Невозможность локальных рассуждений

```
class C { int x }
```

Thread θ

```
C c = new C()  
c.x = 42  
int a = c.x
```

нет других операций с переменной a
ожидаемый инвариант: a == 42



Невозможность локальных рассуждений

```
class C { int x }
```

```
C global = new C()
```

Thread 0

```
C c = new C()
```

```
c.x = 42
```

```
int a = c.x
```

```
global = c
```

Thread 1

```
global.x = 7
```

нет других операций с переменной a
ожидаемый инвариант: a == 42



Невозможность локальных рассуждений

```
class C { int x }
```

```
C global = new C()
```

Thread 0

```
C c = new C()
```

```
c.x = 42
```

```
global = c
```

```
int a = c.x
```

Thread 1

```
global.x = 7
```

нет других операций с переменной a

нет инварианта : a == 42



OCaml Memory Model

- ❖ OCaml 5.0.0: отказ от global runtime lock (2022)
- ❖ Только два режима доступа (атомарный и не атомарный)
- ❖ **Local DRF** :
результат чтения **X** не зависит от гонки по переменной **Y**
- ❖ Чтения в одном потоке из **A** вернут одно и то же значение (если нет конкурирующей записи)
- ❖ Нет **OoTA**
- ❖ Не бесплатно: **load-to-store reordering** запрещен

Promising semantics

2020

«Promising 2.0: Global Optimizations in Relaxed Memory Concurrency»

by Sung-Hwan Lee, Minki Cho, Anton Podkopaev, Soham Chakraborty, Chung-Kil Hur, Ori Lahav, Viktor Vafeiadis



Операционная модель, верифицированная на **Coq**



Нет **OoTA** ! Нет ограничений на оптимизации, минимальные накладные расходы !



Нет ***undefined behavior*** при гонках !



Одно но...

Memory Helpers:

(MEMORY: NEW)

$$\frac{}{\langle P, M \rangle \xrightarrow{m} \langle P, M \triangleleft m \rangle}$$

(MEMORY: FULFILL)

 $m \in P$

$$\frac{}{\langle P, M \rangle \xrightarrow{m} \langle P \setminus \{m\}, M \rangle}$$

Thread Helpers:

(READ-HELPER)

$$m = \langle x : _@(_, t], R \rangle \in M \quad V(x) \leq t$$

$$o = r1x \Rightarrow V' = V \sqcup \{x@t\}$$

$$o = ra \Rightarrow V' = V \sqcup \{x@t\} \sqcup R$$

$$\frac{}{\langle V, M \rangle \xrightarrow{o,m}_R \langle V', M \rangle}$$

(WRITE-HELPER)

$$m = \langle x : _@(_, t], R \rangle \quad V(x) < t$$

$$V' = V \sqcup \{x@t\}$$

$$o = r1x \Rightarrow R = \perp$$

$$o = ra \Rightarrow P(x) = \emptyset \wedge R = V'$$

$$\langle P, M \rangle \xrightarrow{m} \langle P', M' \rangle$$

$$\frac{}{\langle V, P, M \rangle \xrightarrow{o,m}_W \langle V', P', M' \rangle}$$

Thread Steps:

(PROMISE)

$$m = \langle _ : _@(_, _], R \rangle$$

$$M' = M \triangleleft m \quad R \in M'$$

$$\frac{}{\langle \langle \sigma, \mathcal{V}, P \rangle, M \rangle \rightarrow \langle \langle \sigma, \mathcal{V}, P \cup \{m\} \rangle, M' \rangle}$$

(RESERVE)

$$m = \langle _ : (_, _] \rangle \quad M' = M \triangleleft m$$

$$\frac{}{\langle \langle \sigma, \mathcal{V}, P \rangle, M \rangle \rightarrow \langle \langle \sigma, \mathcal{V}, P \cup \{m\} \rangle, M' \rangle}$$

(CANCEL)

$$m = \langle _ : (_, _] \rangle \in P$$

$$\frac{}{\langle \langle \sigma, \mathcal{V}, P \rangle, M \rangle \rightarrow \langle \langle \sigma, \mathcal{V}, P \setminus \{m\} \rangle, M \setminus \{m\} \rangle}$$

(READ)

$$\sigma \xrightarrow{R(o,x,v)} \sigma'$$

$$m = \langle x : v@(_, _], _ \rangle$$

$$\langle V, M \rangle \xrightarrow{o,m}_R \langle V', M \rangle$$

$$\frac{}{\langle \langle \sigma, V, P \rangle, M \rangle \rightarrow \langle \langle \sigma', V', P \rangle, M \rangle}$$

(WRITE)

$$\sigma \xrightarrow{W(o,x,v)} \sigma'$$

$$m = \langle x : v@(_, _], _ \rangle$$

$$\langle V, P, M \rangle \xrightarrow{o,m}_W \langle V', P', M' \rangle$$

$$\frac{}{\langle \langle \sigma, V, P \rangle, M \rangle \rightarrow \langle \langle \sigma', V', P' \rangle, M' \rangle}$$

(UPDATE)

$$\sigma \xrightarrow{U(o_r, o_w, x, o_r, o_w)} \sigma''$$

$$m_r = \langle x : v_r@(_, t], _ \rangle$$

$$m_w = \langle x : v_w@(_, t], _ \rangle$$

$$\langle V, M \rangle \xrightarrow{o_r, m_r}_R \langle V', M \rangle$$

$$\langle V', P, M \rangle \xrightarrow{o_w, m_w}_W \langle V'', P'', M'' \rangle$$

$$\frac{}{\langle \langle \sigma, V, P \rangle, M \rangle \rightarrow \langle \langle \sigma'', V'', P'' \rangle, M'' \rangle}$$

(SILENT)

$$\sigma \xrightarrow{\text{Silent}} \sigma'$$

$$\frac{}{\langle \langle \sigma, \mathcal{V}, P \rangle, M \rangle \rightarrow \langle \langle \sigma', \mathcal{V}, P \rangle, M \rangle}$$

(SYSTEM CALL)

$$\sigma \xrightarrow{\text{Sys}(v)} \sigma' \quad P = \emptyset$$

$$\frac{}{\langle \langle \sigma, \mathcal{V}, P \rangle, M \rangle \xrightarrow{\text{Sys}(v)} \langle \langle \sigma', \mathcal{V}, P \rangle, M \rangle}$$

(FAILURE)

$$\sigma \xrightarrow{\text{Fail}} \perp$$

 $\langle \sigma, \mathcal{V}, P \rangle$ is promise-consistent

$$\frac{}{\langle \langle \sigma, \mathcal{V}, P \rangle, M \rangle \xrightarrow{\text{Fail}} \perp}$$

Machine Steps:

(MACHINE NORMAL)

$$\langle \mathcal{TS}(i), M \rangle \rightarrow^+ \langle \mathcal{TS}', M' \rangle$$

 $\langle \mathcal{TS}', M' \rangle$ is consistent

$$\frac{}{\langle \mathcal{TS}, M \rangle \rightarrow \langle \mathcal{TS}[i \mapsto \mathcal{TS}'], M' \rangle}$$

(MACHINE SYSTEM CALL)

$$\langle \mathcal{TS}(i), M \rangle \rightarrow^* \xrightarrow{\text{Sys}(v)} \langle \mathcal{TS}', M' \rangle$$

 $\langle \mathcal{TS}', M' \rangle$ is consistent

$$\frac{}{\langle \mathcal{TS}, M \rangle \xrightarrow{\text{Sys}(v)} \langle \mathcal{TS}[i \mapsto \mathcal{TS}'], M' \rangle}$$

(MACHINE FAIL)

$$\langle \mathcal{TS}(i), M \rangle \rightarrow^* \xrightarrow{\text{Fail}} \perp$$

$$\frac{}{\langle \mathcal{TS}, M \rangle \xrightarrow{\text{Fail}} \perp}$$

ИТОГИ

- ❖ **Посмотрели на эволюцию моделей памяти**
- ❖ **Во всех популярных ЯП есть проблемы с моделями памяти**
- ❖ **У каждого ЯП есть свой принятый подход к решению этой проблемы – найдите его**
- ❖ **Гонки зло! Баги есть везде – в железе, спеках, компиляторах и рантайме !**



Спасибо

за внимание!



Статьи и примеры

github.com/lantalex/mm-evolution/

