Java Memory Model

Roman Elizarov[1] Nikita Koval[2]

[1]Kotlin Team Lead, JetBrains
elizarov@gmail.com

[2]Researcher, JetBrains
PhD student, IST Austria
ndkoval@ya.ru

ITMO 2019

Посмотрим на многопоточное программирование с практической стороны

Модели памяти

Спецификация поведения хранилища данных

- Чтобы писать корректный и безопасный код
- Чтобы эффективно исполнять код

Модели памяти

Спецификация поведения хранилища данных

- Чтобы писать корректный и безопасный код
- Чтобы эффективно исполнять код

Главный вопрос

Какую запись может прочитать **конкретное** чтение?

Операции над абстрактным хранилищем

Чтение значения общей переменной

$x.r : v$ - чтение значения v из переменной x

Запись в общую переменную

$x.w(v)$ - запись значения v в переменную x

Последовательные программы

Все инструкции выполняются одна за одной, последовательно $\Rightarrow$ чтение x видит последнее записанное в x значение

Последовательные программы

Все инструкции выполняются одна за одной, последовательно $\Rightarrow$ чтение x видит последнее записанное в x значение

Обычно имеется ввиду, что модель памяти покрывает семантику **параллельных** программ

Атомарный доступ

Хотим атомарный доступ к базовым типам:

T t = t1;	
t = t2;	T res = t; assert(res == t1 || res == t2);

Атомарный доступ

Хотим атомарный доступ к базовым типам:

	T t = t1;
t = t2;	T res = t; assert(res == t1 || res == t2);

Всё не так просто

Атомарный доступ: проблема

Реальность

Нужна поддержка атомарных чтения/записи со стороны железа

Атомарный доступ: проблема

Реальность

Нужна поддержка атомарных чтения/записи со стороны железа

Что делать с 64-битными long и double на 32-битных архитектурах?

Атомарный доступ: проблема

Реальность

Нужна поддержка атомарных чтения/записи со стороны железа

Что делать с 64-битными long и double на 32-битных архитектурах?

Если данные лежат на пересечении кеш-лайнов, то теряем атомарность

Атомарный доступ: long и double

- Доступ атомарен для всего, кроме long и double
- К **volatile** $\{\mathrm{long}, \mathrm{double}\}$ атомарен

Атомарный доступ: long и double

- Доступ атомарен для всего, кроме long и double
- К **volatile** $\{\mathrm{long}, \mathrm{double}\}$ атомарен

	long l = 0;
l = -1; // 0xFF..F	print(l);

Может напечатать $\mathrm{0xFFFFFFFF00000000}$

Атомарный доступ: long и double

- Доступ атомарен для всего, кроме long и double
- К **volatile** $\{\mathrm{long}, \mathrm{double}\}$ атомарен

volatile long l = 0;	
l = -1; // 0xFF..F	print(l);

Теперь только $0\mathrm{x}00..0$ или $0\mathrm{xFF..F}$

Атомарный доступ: reference

Вопрос

Что делать с 64-битными ссылками?

Атомарный доступ: reference

Вопрос

Что делать с 64-битными ссылками?

Ответ

64-битные ссылки могут быть только на 64-битной* архитектуре

* или больше

Атомарный доступ: кеш-лайны

- Выравниваем данные так, чтобы не лежали на пересечении кеш-лайнов
- Заодно улучшаем перфоманс

Атомарный доступ: кеш-лайны

- Выравниваем данные так, чтобы не лежали на пересечении кеш-лайнов
- Заодно улучшаем перфоманс

Простой класс в разрезе, x86

offset	size	type	description
0	12		(object header)
12	4		(alignment/padding gap)
16	8	long	A.f (volatile)

Word tearing

А ещё хочется вот так!

```
T[] tarr = new T[..];
tarr[0] = tarr[1] = val0;
```

tarr[0] = val1;	tarr[1] = val1;	<join both>
<term>	<term>	assert(tarr[0] == tarr[1]);

Word tearing

А ещё хочется вот так!

T[] tarr = new T[..];
tarr[0] = tarr[1] = val0;

tarr[0] = val1; <term>	tarr[1] = val1; <term>	<join both> assert(tarr[0] == tarr[1]);

Снова нужна поддержка со стороны железа, чтобы делать независимые чтения/записи

Word tearing: boolean

Вопрос

Что делать с 1-битным $\mathrm{boolean}$, если обычно можно записать минимум 1 байт (8 бит)?

Word tearing: boolean

Вопрос

Что делать с 1-битным $\mathrm{boolean}$, если обычно можно записать минимум 1 байт (8 бит)?

Ответ

В JMM word tearing запрещен $\Rightarrow$ в таком случае boolean занимает 1 байт*

* если железо умеет адресовать минимум N бит, то минимальный размер базового типа тоже N бит

Word tearing: задачка

Что напечатает эта программа?

BitSet bs = new BitSet();		

bs.set(1);
<term>
	bs.set(2);
	<term>
		<join both>
		print(bs.get(1));
		print(bs.get(2));

Word tearing: задачка

Что напечатает эта программа?

BitSet bs = new BitSet();

bs.set(1);
<term>
	bs.set(2);
	<term>
		<join both>
		print(bs.get(1));
		print(bs.get(2));

Напечатает (F, T), (F, T) или (T, T)

Word tearing: задачка

Что напечатает эта программа?

BitSet bs = new BitSet();		

bs.set(1);
<term>
	bs.set(2);
	<term>
		<join both>
		print(bs.get(1));
		print(bs.get(2));

Напечатает (F, T), (F, T) или (T, T)

А может напечатать (F, F)?

Жизнь - боль

Модель памяти - это **компромисс** между

- удобством программирования
- сложностью реализации языка
- сложностью создания железа

Жизнь - боль

Модель памяти - это **компромисс** между

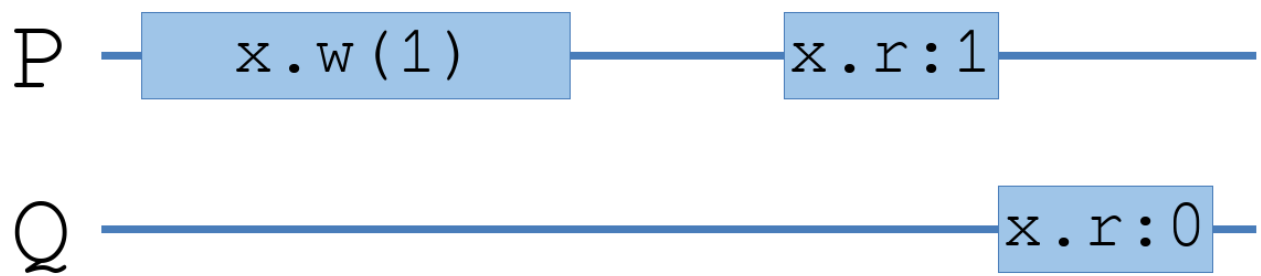
- удобством программирования
- сложностью реализации языка
- сложностью создания железа

Можно потребовать что угодно, но сколько времени и сил уйдет на создание ЯП и железа... Плюс производительность...

Последовательная согласованность

Sequential consistency (SC)

Последовательно согласованно $\Leftrightarrow \exists$ допустимое последовательное исполнение, сохраняющее программный порядок (po)



JMM: что хотим

- Хотим последовательную согласованность
  - Позволяет анализировать программу в модели чередования
  - Сложно делать локальные оптимизации

JMM: что хотим

- Хотим последовательную согласованность
  - Позволяет анализировать программу в модели чередования
  - Сложно делать локальные оптимизации
- Упростим модель
  - Разрешим оптимизации
  - Сохраним разумность и вменяемость

SC-DRF

SC-DRF*

Если в исполнении нет гонок, то, согласно JMM, оно последовательно согласованно

* Sequentially consistent for data race free programs

SC-DRF

SC-DRF*

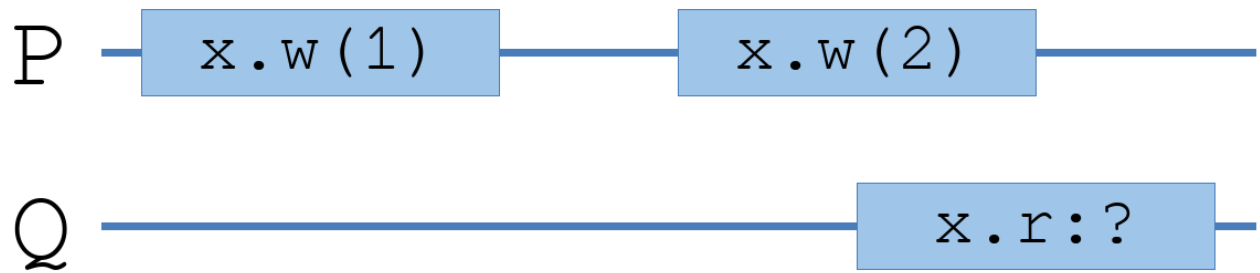
Если в исполнении нет гонок, то, согласно JMM, оно последовательно согласованно

* Sequentially consistent for data race free programs

Что такое гонка?

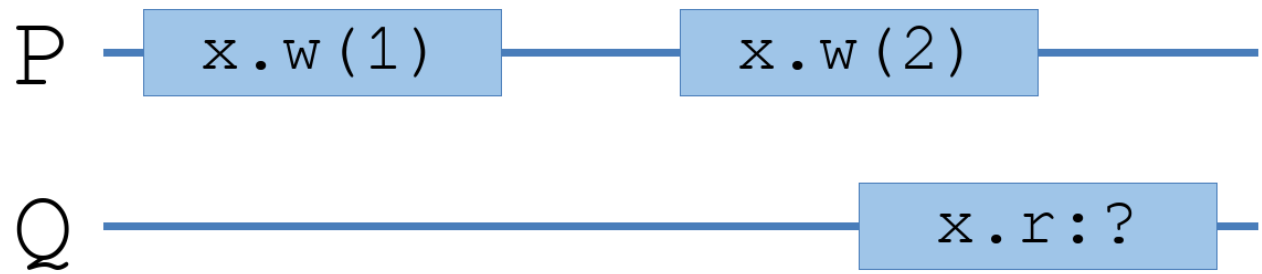
Гонка данных

Гонка данных = два *конфликтующих* доступа, не связанные отношением "произошло до"



Гонка данных

Гонка данных = два *конфликтующих* доступа, не связанные отношением "произошло до"



happens-before order

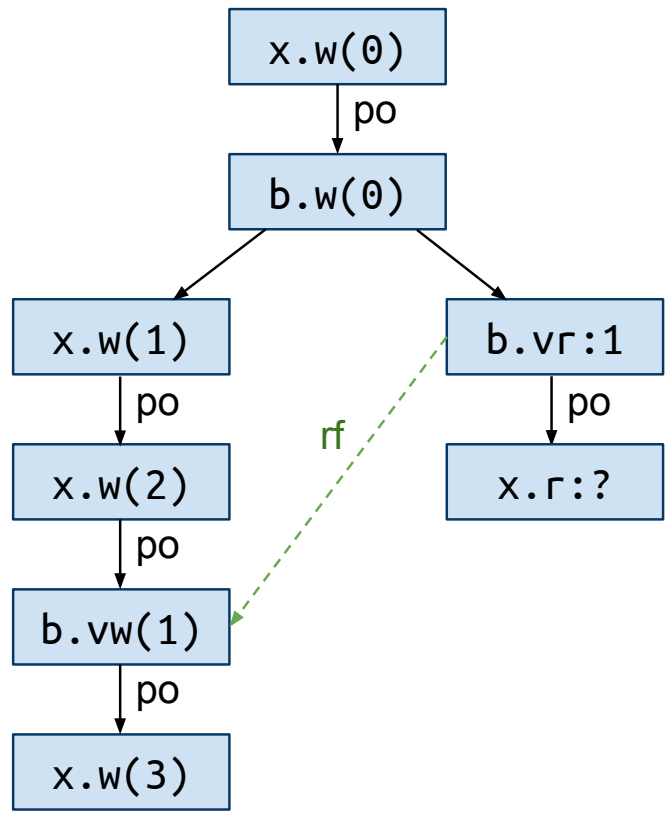
Нужно понять, что такое "произошло до" с точки зрения модели памяти Java!

Program order

Программный порядок (PO) связывает действия внутри одного потока

```
int x = 0;
volatile boolean b = false;
```

x = 1;
x = 2;
b = true;
x = 3;
	if (b)
	  print(x)



Synchronization actions

Слабые модели памяти упорядочивают избранные операции

Synchronization Actions:

- volatile чтение/запись
- взятие/отпускание блокировки
- (синтетические) первое и последнее действие в потоке
- действия, запускающие поток
- действия, обнаруживающие останов потока (Thread.join(), Thread.isInterrupted(), ...)

Synchronization order

Synchronization actions образуют synchronization order (SO)

- SO - линейный порядок
- Консистентен с PO

Synchronization order consistency

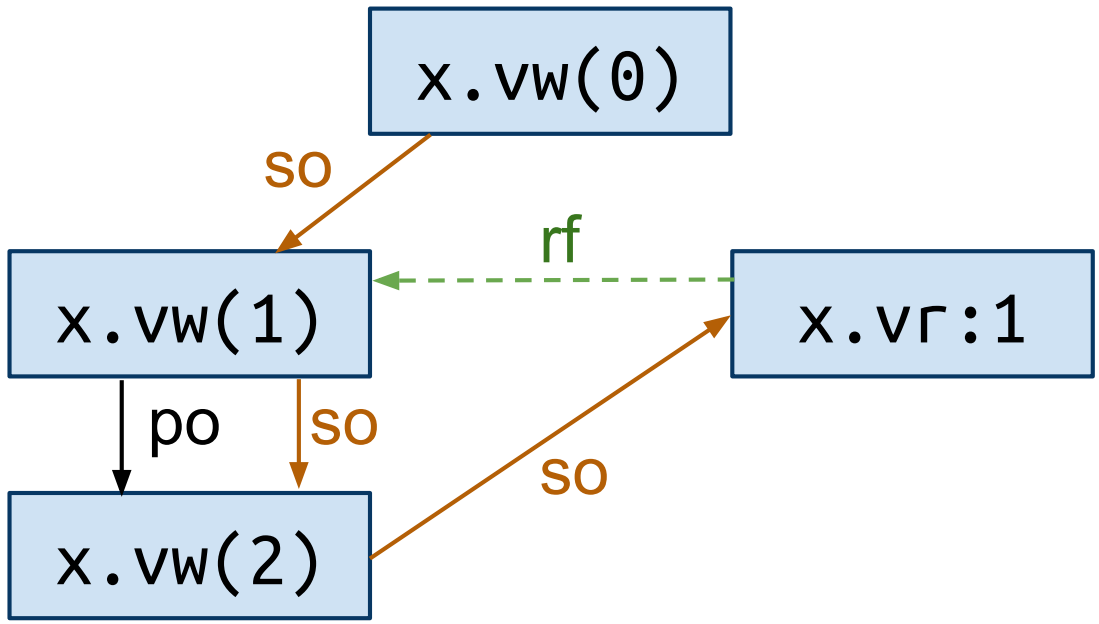
Все чтения в SO видят последние записи в SO

Synchronization order: volatile

- volatile read/write - Synchronization Actions
- над всеми операциями доступа есть SO полный порядок!
- чтения и записи согласованы

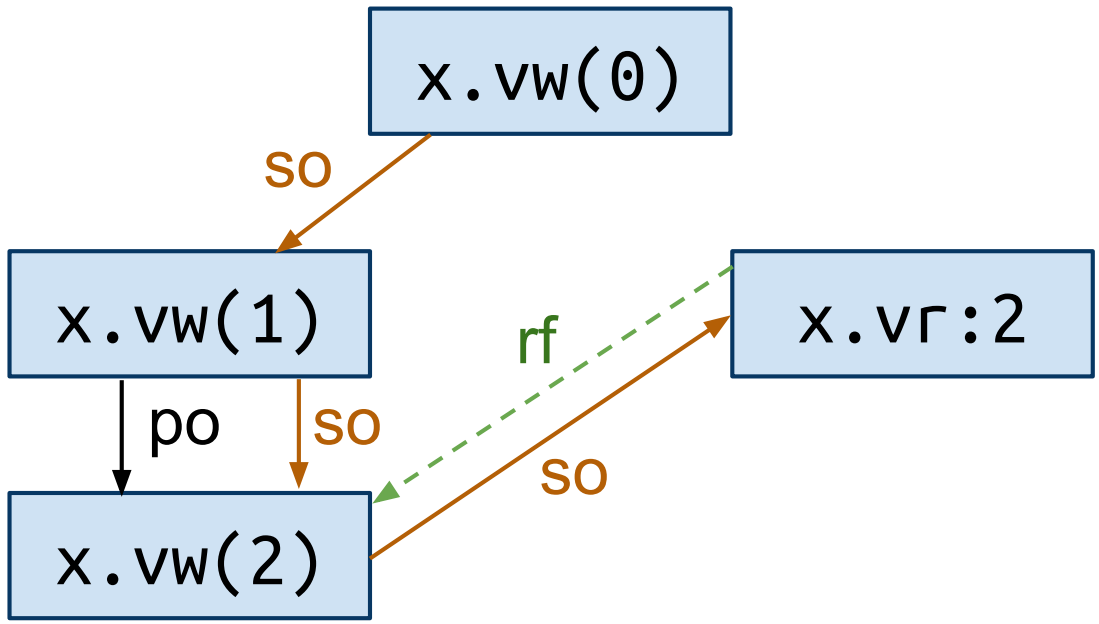
volatile переменные **линеаризуемы**

Synchronization order consistency



Такого быть не может: последняя запись в SO - это "x.vw(2)"

Synchronization order consistency



А вот это уже корректно!

Synchronizes-with order

- PO не связывает действия разных потоков
- Нужен порядок, связывающий действия разных потоков
- SO линеен $\Rightarrow$ накладывает строгие ограничения

Synchronizes-with order

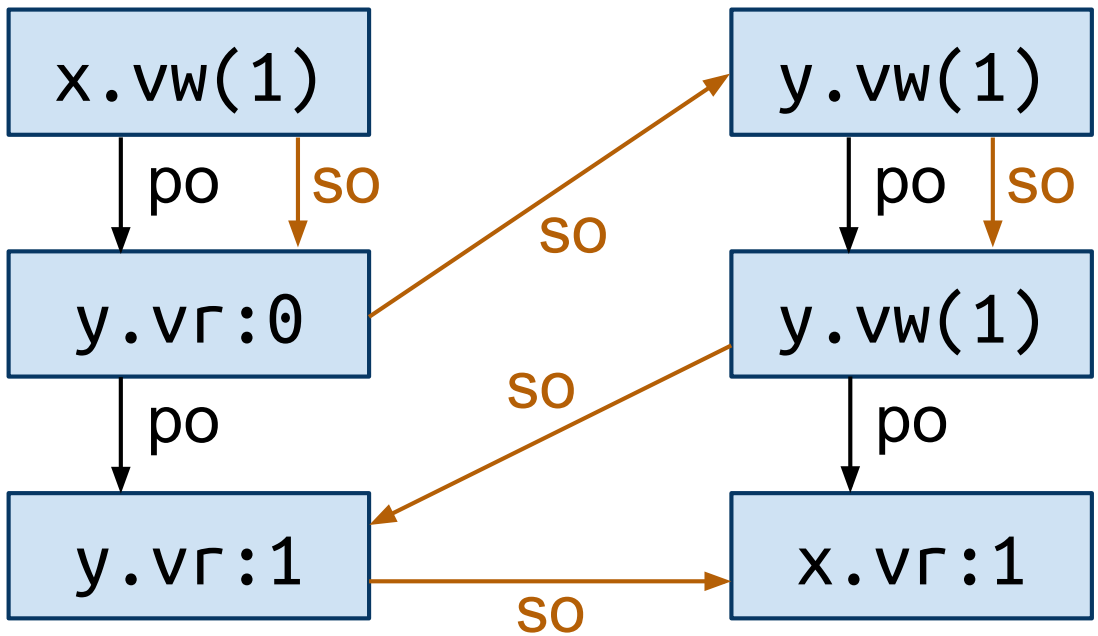
- PO не связывает действия разных потоков
- Нужен порядок, связывающий действия разных потоков
- SO линеен $\Rightarrow$ накладывает строгие ограничения

Synchronizes-with order (SW)

Подпорядок SO, ограниченный конкретными парными действиями синхронизации (volatile read/write, lock/unlock, ...)

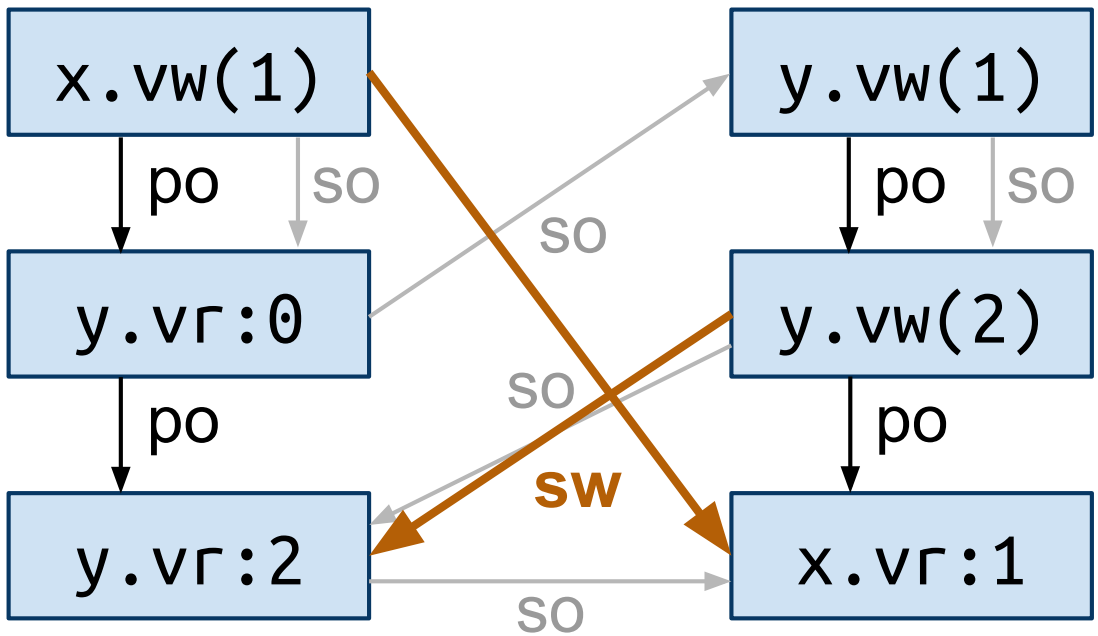
Synchronizes-with order: пример

Пусть есть такое исполнение с выбранным SO:



Synchronizes-with order: пример

Тогда SW будет таким:



Happens-before order

$$HB = (SW \cup PO)^+$$

PO даёт семантику внутри потока, SW - между потоками

Happens-before order

$$HB = (SW \cup PO)^+$$

PO даёт семантику внутри потока, SW - между потоками

Happens-before consistency

Чтения могут увидеть либо последнюю запись в HB, либо что-нибудь ещё через гонку

Happens-before consistency

```
          int x = 0;
volatile boolean b = false;
```

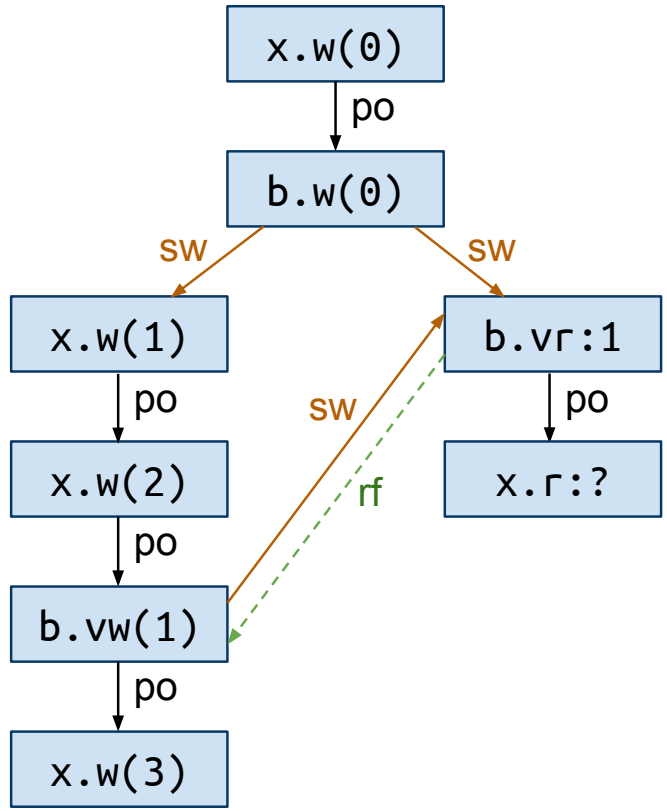
x = 1;	
x = 2;	if (b)
b = true;	print(x)
x = 3;	

Что может увидеть второй поток?

Happens-before consistency

int x = 0;
volatile boolean b = false;

x = 1;
x = 2;
b = true;
x = 3;
	if (b)
	  print(x)

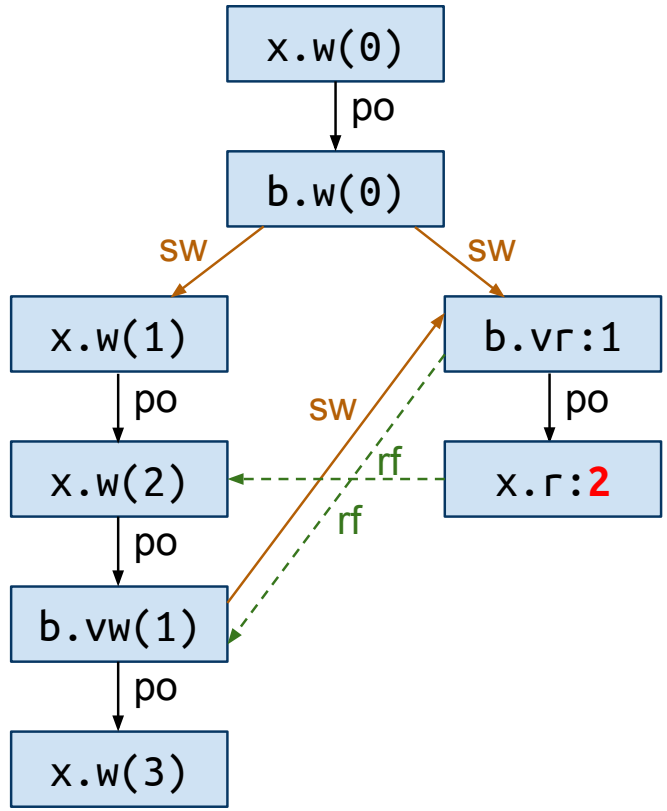


Что может увидеть второй поток?

Happens-before consistency

int x = 0;
volatile boolean b = false;

x = 1;
x = 2;
b = true;
x = 3;
	if (b)
	  print(x)

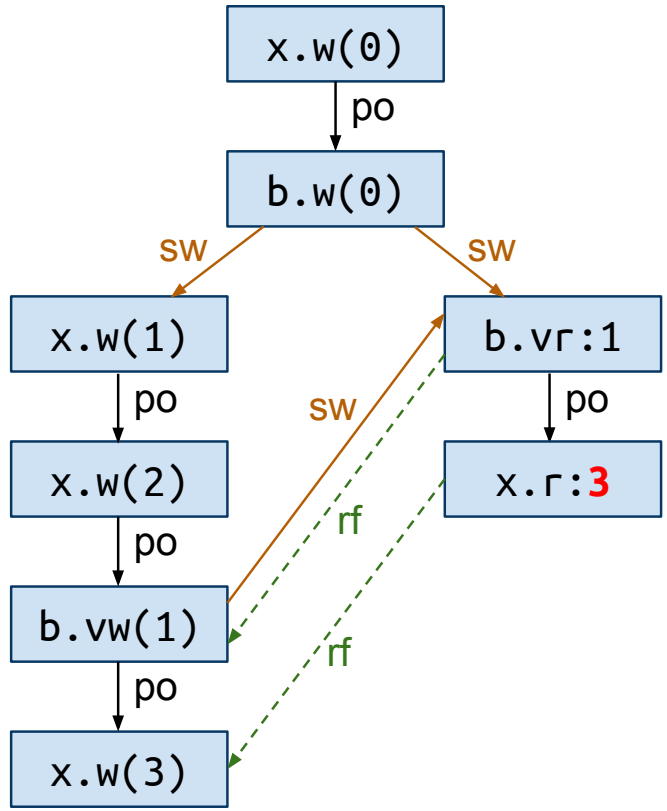


Может увидеть запись "x = 2" как последнюю в HB

Happens-before consistency

int x = 0;
volatile boolean b = false;

x = 1;
x = 2;
b = true;
x = 3;
	if (b)
	  print(x)



Может увидеть запись "x = 3" под гонкой

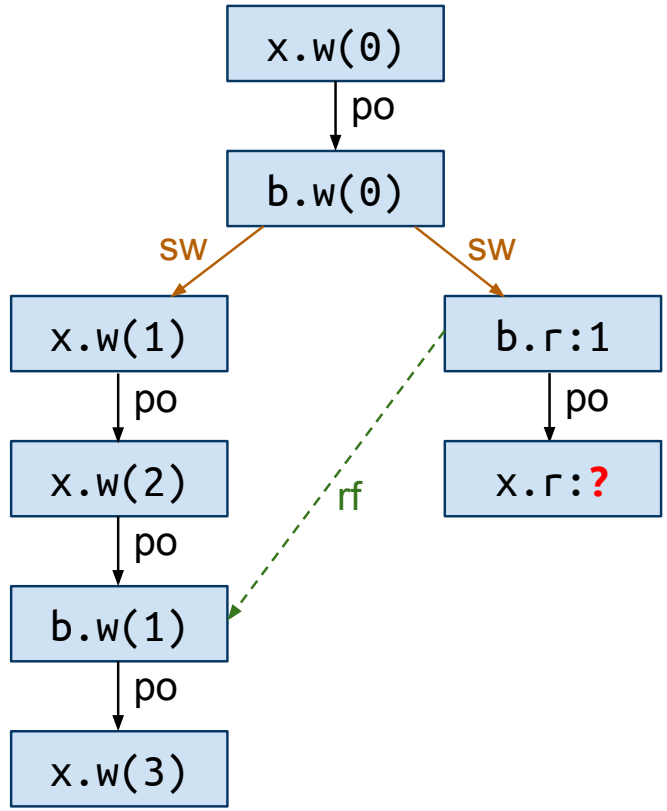
SC-DFR: задачка

```
        int x = 0;
    boolean b = false;
-------------------------------
   x = 1;     |
   x = 2;     |  if (b)
   b = true;  |    print(x)
   x = 3;     |
```

SC-DFR: задачка

```
int x = 0;
boolean b = false;
```

x = 1;	if (b)
x = 2;	print(x)
b = true;	
x = 3;	



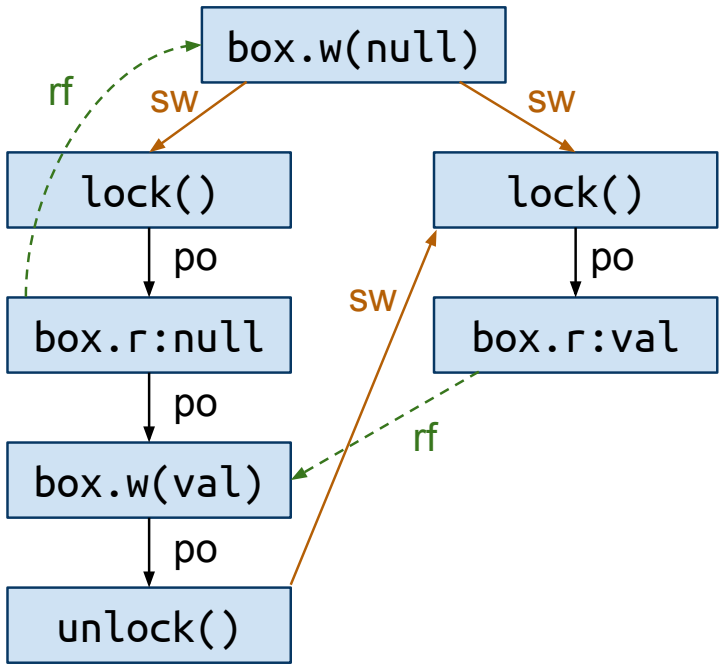
Может увидеть запись "x = 0" как последнюю в HB или всё остальное под гонкой

SC-DRF: ещё задачка

```
 class Box<T>  private T
box;
synchronized void set(T val)
if (box == null) box = val;
// Fix me! I am too hot in
profiler!  synchronized  T
get()  return box;
```

SC-DRF: ещё задачка

```
class Box<T> {
  private T box;
  synchronized void set(T val) {
    if (box == null) box = val;
    // Fix me! I am too hot in profiler!
  }
  synchronized T get() { return box; }
}
```



SC-DRF: ещё задачка

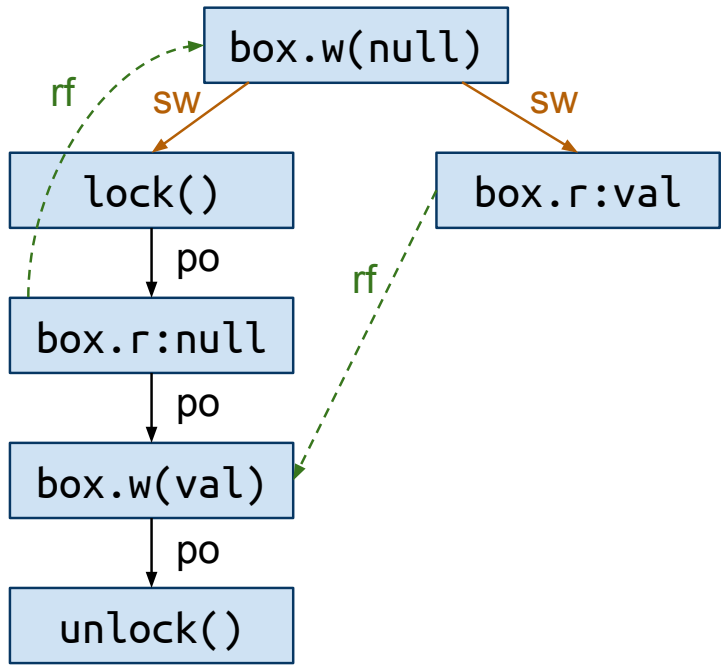
Соптимизируем: уберём synchronized на get

```
class Box<T> {
  private T box;
  synchronized void set(T val) {
    if (box == null) box = val;
  }
  T get() {
    return box;
  }
}
```

SC-DRF: ещё задачка

Соптимизируем: уберём synchronized на get

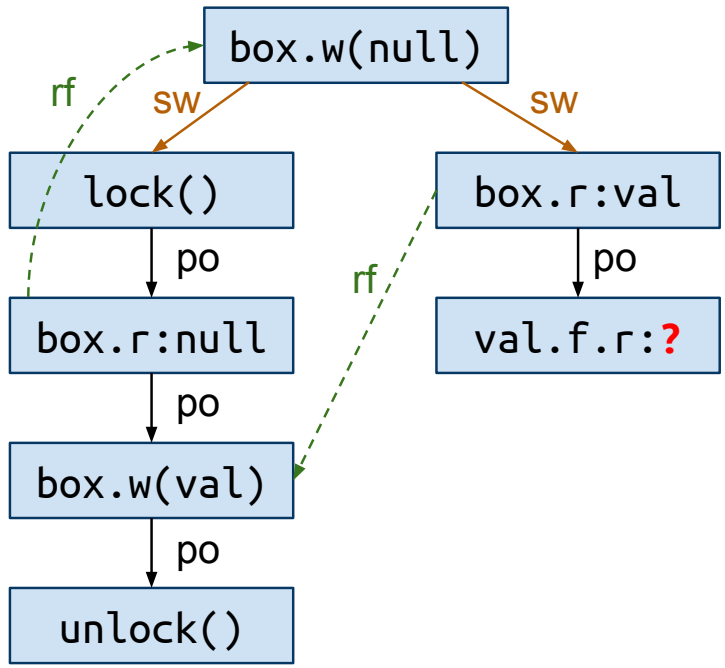
```
class Box<T> {
  private T box;
  synchronized void set(T val) {
    if (box == null) box = val;
  }
  T get() {
    return box;
  }
}
```



SC-DRF: ещё задачка

Соптимизируем: уберём synchronized на get

```
class Box<T> {
  private T box;
  synchronized void set(T val) {
    if (box == null) box = val;
  }
  T get() {
    return box;
  }
}
```



Многие делают так, "это же чтение"!

SC-DRF: ещё задачка

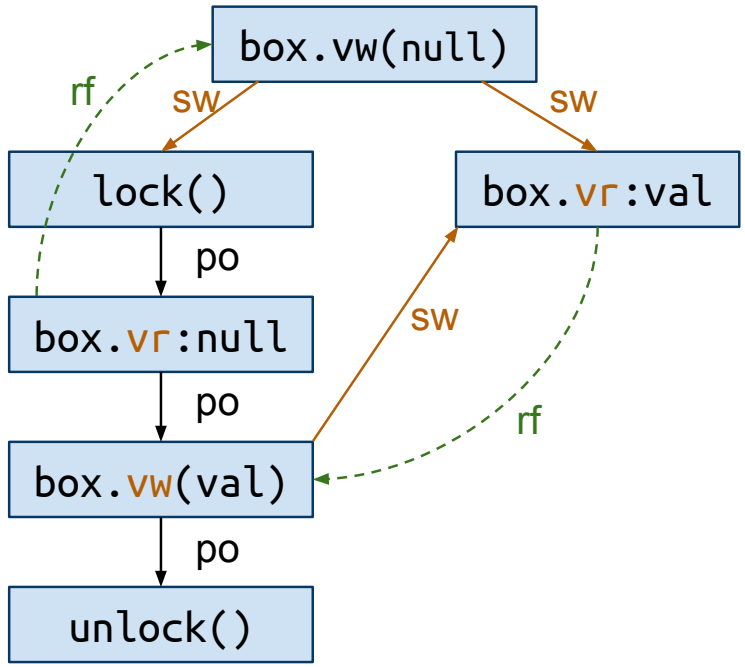
Исправим: добавим volatile

```
class Box<T> {
  private volatile T box;
  synchronized void set(T val) {
    if (box == null) box = val;
  }
  T get() { return box; }
}
```

SC-DRF: ещё задачка

Исправим: добавим volatile

```
class Box<T> {
  private volatile T box;
  synchronized void set(T val) {
    if (box == null) box = val;
  }
  T get() { return box; }
}
```



Вернули SW ребро, восстановили HB

Очень важная мысль

Отношения PO, SW, HB и т.д. не задают реальный порядок выполнения операций, это только модель!

OoTA: логичные мысли

- Локальные трансформации разрешены, пока не встретили синхронизацию
- Локальные трансформации ничего не «ломают», если программа корректно синхронизирована
- Если «сломали», то была гонка и разработчик сам виноват

OoTA: реальность

Есть случаи, когда локальные трансформации всё ломают:

int a = 0, b = 0;

r1 = a;	r2 = b;
if (r1 != 0)	if (r2 != 0)
b = 42;	a = 42;

OoTA: реальность

Есть случаи, когда локальные трансформации всё ломают:

int a = 0, b = 0;	

r1 = a;
if (r1 != 0)
b = 42;
	r2 = b;
	if (r2 != 0)
	a = 42;

Корректно синхронизованна

Возможно только (r1, r2) = (0, 0)

ООТА: оптимизации

Спекулятивная оптимизация:
почему бы не записать в b и откатить «если что»?

int a = 0, b = 0

```
int r1 = a;
if (r1 != 0)
  b = 42;
```

OoTA: оптимизации

Спекулятивная оптимизация:
почему бы не записать в b и откатить «если что»?

int a = 0, b = 0

```
int r1 = a;
if (r1 != 0)
  b = 42;
```

```
int r1 = a;
b = 42;
if (r1 == 0)
  b = 0;
```

OoTA: оптимизации

Спекулятивная оптимизация:
почему бы не записать в b и откатить «если что»?

int a = 0, b = 0

```
int r1 = a;
if (r1 != 0)
  b = 42;
```

```
int r1 = a;
b = 42;
if (r1 == 0)
  b = 0;
```

```
b = 42;
int r1 = a;
if (r1 == 0)
  b = 0;
```

OoTA: дооптимизировались

Запускаем оптимизированный код:

int a = 0, b = 0;	
b = 42;	a = 42;
int r1 = a;	int r2 = b;
if (r1 == 0)	if (r2 == 0)
b = 0;	a = 0;

OoTA: дооптимизировались

Запускаем оптимизированный код:

int a = 0, b = 0;	
b = 42;	a = 42;
int r1 = a;	int r2 = b;
if (r1 == 0)	if (r2 == 0)
b = 0;	a = 0;

Теперь можем получить (r1, r2) = (1, 1)

OoTA: causality

JMM вкратце

Out-of-Thin-Air значения запрещены

- прочитали значение $\Rightarrow$ кто-то его **до нас** записал
- самая сложная часть спецификация: нужно понять, что такое это «до нас»
- JMM определяет специальный процесс валидации исполнений через «commit»-ы действий

OoTA: commit semantics

17.4.8 Executions and Causality Requirements

We use $f|_d$ to denote the function given by restricting the domain of f to d. For all x in d, $f|_d(x) = f(x)$, and for all x not in d, $f|_d(x)$ is undefined.

We use $p|_d$ to represent the restriction of the partial order p to the elements in d. For all x,y in d, $p(x,y)$ if and only if $p|_d(x,y)$. If either x or y are not in d, then it is not the case that $p|_d(x,y)$.

A well-formed execution $E = < P, A, po, so, W, V, sw, hb >$ is validated by *committing* actions from A. If all of the actions in A can be committed, then the execution satisfies the causality requirements of the Java programming language memory model.

Starting with the empty set as C_0, we perform a sequence of steps where we take actions from the set of actions A and add them to a set of committed actions C_i to get a new set of committed actions C_{i+1}. To demonstrate that this is reasonable, for each C_i we need to demonstrate an execution E containing C_i that meets certain conditions.

Formally, an execution E *satisfies the causality requirements of the Java programming language memory model* if and only if there exist:

- Sets of actions $C_0, C_1, ...$ such that:
  - C_0 is the empty set
  - C_i is a proper subset of C_{i+1}
  - $A = \cup (C_0, C_1, ...)$

  If A is finite, then the sequence $C_0, C_1, ...$ will be finite, ending in a set $C_n = A$.

  If A is infinite, then the sequence $C_0, C_1, ...$ may be infinite, and it must be the case that the union of all elements of this infinite sequence is equal to A.
- Well-formed executions $E_1, ...$, where $E_i = < P, A_i, po_i, so_i, W_i, V_i, sw_i, hb_i >$.

Given these sets of actions $C_0, ...$ and executions $E_1, ... ,$ every action in C_i must be one of the actions in E_i. All actions in C_i must share the same relative happens-before order and synchronization order in both E_i and E. Formally:

1. C_i is a subset of A_i
2. $hb_i|_{C_i} = hb|_{C_i}$
3. $so_i|_{C_i} = so|_{C_i}$

The values written by the writes in C_i must be the same in both E_i and E. Only the reads in C_{i-1} need to see the same writes in E_i as in E. Formally:

4. $V_i|_{C_i} = V|_{C_i}$
5. $W_i|_{C_{i-1}} = W|_{C_{i-1}}$

All reads in E_i that are not in C_{i-1} must see writes that happen-before them. Each read r in C_i - C_{i-1} must see writes in C_{i-1} in both E_i and E, but may see a different write in E_i from the one it sees in E. Formally:

6. For any read r in A_i - C_{i-1}, we have $hb_i(W_i(r), r)$
7. For any read r in (C_i - C_{i-1}), we have $W_i(r)$ in C_{i-1} and $W(r)$ in C_{i-1}

Given a set of sufficient synchronizes-with edges for E_i, if there is a release-acquire pair that happens-before (§17.4.5) an action you are committing, then that pair must be present in all E_j, where $j \geq i$. Formally:

8. Let ssw_i be the sw_i edges that are also in the transitive reduction of hb_i but not in po. We call ssw_i the *sufficient synchronizes-with edges for* E_i. If $ssw_i(x, y)$ and $hb_i(y, z)$ and z in C_i, then $sw_j(x, y)$ for all $j \geq i$.

   If an action y is committed, all external actions that happen-before y are also committed.
9. If y is in C_i, x is an external action and $hb_i(x, y)$, then x in C_i.

Finals

Finals: задачка

Что напечатает этот код?

```
class A {
  int f;
  A() { f = 42; }
}
A a = null;
```

`a = new A();`	`if (a != null)` `print(a.f)`

Finals: задачка

Что напечатает этот код?

```
class A {
  int f;
  A() { f = 42; }
}
A a = null;
```

`a = new A();`	`if (a != null)` `print(a.f)`

<ничего>, 0, 42 или бросит NullPointerException

Finals: задачка

Исправим, чтобы не было NPE

```
class A {
  int f;
  A() { f = 42; }
}
A a = null;
```

`a = new A();`	`A ta = a;`
	`if (ta != null)`
	`  print(ta.f)`

Finals: задачка

Исправим, чтобы не было NPE

```
class A {
  int f;
  A() { f = 42; }
}
A a = null;
```

`a = new A();`	`A ta = a;`
	`if (ta != null)`
	`  print(ta.f)`

<ничего>, 0 или 42

Finals: задачка

Хотелось бы только <ничего> или 42

```
class A {
  ????? int f;
  A() { f = 42; }
}
A a = null;
```

`a = new A();`	`A ta = a;`
	`if (ta != null)`
	`  print(ta.f)`

Хотелось бы иметь объекты, которые можно безопасно публиковать через гонки...

Final семантика "на пальцах"

- Пометим поле как **final**, чтобы «зафиксировать» значение

Final семантика "на пальцах"

- Пометим поле как **final**, чтобы «зафиксировать» значение
- В конце конструктора происходит freeze action
- Freeze action «замораживает» поля

Final семантика "на пальцах"

- Пометим поле как **final**, чтобы «зафиксировать» значение
- В конце конструктора происходит freeze action
- Freeze action «замораживает» поля
- Поток прочитал ссылку на объект $\Rightarrow$ увидит «замороженные» значения
- Поток прочитал ссылку на другой объект из final поля $\Rightarrow$ увидит состояние настолько же свежее, как и на время freeze action-а

Finals: задачка

А что получим теперь?

```
class A {
  final int f;
  A() { f = 42; }
}
A a = null;
```

`a = new A();`	`A ta = a;`
	`if (ta != null)`
	`  print(ta.f)`

Finals: задачка

А что получим теперь?

```
class A {
  final int f;
  A() { f = 42; }
}
A a = null;
```

`a = new A();`	`A ta = a;`
	`if (ta != null)`
	`  print(ta.f)`

<ничего> или 42

Finals: проблема

Всё хорошо пока мы публикуем ссылку на объект **после** freeze action-a.

А что будет, если опубликовать **до**?

```
class A {
  final int x;
  A() { x = 42; }
}
A a = null, b = null;
```

A a1 = <**new**>
  a1.x = 42
  a = a1; // a = this
<**freeze a1.x**>
b = a1;
	A a2 = b;
	r2 = a2.x;
		A a3 = a;
		r3 = a.x;
		A a4 = b;
		r4 = a.x;

Finals: проблема

Всё хорошо пока мы публикуем ссылку на объект **после** freeze action-a.

А что будет, если опубликовать **до**?

```
class A {
  final int x;
  A() { x = 42; }
}
A a = null, b = null;
```

A a1 = <**new**>
  a1.x = 42
  a = a1; // a = this
<**freeze a1.x**>
b = a1;
	A a2 = b;
	r2 = a2.x;
		A a3 = a;
		r3 = a.x;
		A a4 = b;
		r4 = a.x;

В r4 может быть 0!

Материал для чтения

- JLS Chapter 17. Threads and Locks
  - https://docs.oracle.com/javase/specs/jls/se7/html/jls-17.html
- The Java memory model
  - http://dl.acm.org/citation.cfm?id=1040336
- Java Memory Model Pragmatics by Alexey Shipilev
  - https://shipilev.net/blog/2014/jmm-pragmatics/

Вопросы?