Многопоточное Программирование:
Алгоритмы без блокировок: Построения на регистрах

Роман Елизаров, JetBrains, elizarov@gmail.com
Никита Коваль, JetBrains, ndkoval@ya.ru

ИТМО 2020

ITMO UNIVERSITY

⚠Контрольный вопрос: Блокировки 1

Какие утверждения верные?

- ✓ a. Используя взаимное исключение (mutex) можно любой последовательный объект сделать линеаризуемым
- ✓ b. Использование множества блокировок внутри одного объекта называется тонкой блокировкой
- ✗ c. Использование грубой блокировки дает гарантию того, что любая операция будет выполнена за конечное время без каких-либо условий на действия, выполняемые другими потоками

⚠Контрольный вопрос: Блокировки 2

Какие утверждения верные?

- ✕ a. При использовании множества блокировок внутри одного объекта линеаризуемость гарантируется только при иерархической блокировке
- ✕ b. Условие корректности алгоритма взаимного исключения заключается в том, что любой поток должен войти в критическую секцию за конечное время, если другие потоки не находятся в ней
- ✕ c. Алгоритм Петерсона для N потоков гарантирует линейное ожидание входа в критическую секцию

⚠Контрольный вопрос: Блокировки 3

Расставьте следующие свойства честности для задачи взаимного исключение в порядке усиления (от более слабого свойства, к более сильному свойству).

3 a. Первым пришел, первым обслужен (**first come, first served**)
2 b. Линейное ожидание (**linear wait**)
1 c. Отсутствие голодание (**starvation-freedom**)

Условные условия прогресса с блокировками

- **Прогресс с блокировками**
  - Отсутствие взаимной блокировки
  - Отсутствие голодания

- **Условно**
  - Только при условии что другие потоки проводят в критической секции конечное время

Алгоритмы без блокировок (lock-free)

Безусловные условия прогресса

Безусловные условия прогресса

- **Отсутствие помех** (obstruction-freedom)
  - Если несколько потоков пытаются выполнить операцию, то любой из них должен выполнить её за конечное время, если все другие потоки остановить в любом месте (чтобы не мешали)

Безусловные условия прогресса

- **Отсутствие помех** (obstruction-freedom)
  - Если несколько потоков пытаются выполнить операцию, то любой из них должен выполнить её за конечное время, если все другие потоки остановить в любом месте (чтобы не мешали)
- **Отсутствие блокировки** (lock-freedom)
  - Если несколько потоков пытаются выполнить операцию, то хотя бы один из них должен выполнить её за конечное время (не зависимо от действия или бездействия других потоков)

Безусловные условия прогресса

- **Отсутствие помех** (obstruction-freedom)
  - Если несколько потоков пытаются выполнить операцию, то любой из них должен выполнить её за конечное время, если все другие потоки остановить в любом месте (чтобы не мешали)
- **Отсутствие блокировки** (lock-freedom)
  - Если несколько потоков пытаются выполнить операцию, то хотя бы один из них должен выполнить её за конечное время (не зависимо от действия или бездействия других потоков)
- **Отсутствие ожидания** (wait-freedom)
  - Если какой-то поток пытается выполнить операцию, то он выполнит её за конечное время (не зависимо от действия или бездействия других потоков)

Объекты без блокировки

Используя **блокировку** (lock)
мы не можем получить объект
без блокировки (lock-free) и даже
без помех (obstruction-free)

Вопросы

- Какой самый простой объект может лежать в основе параллельного программирования?

- Что нужно чтобы писать программы:
  - С блокировками?
  - Без блокировок?

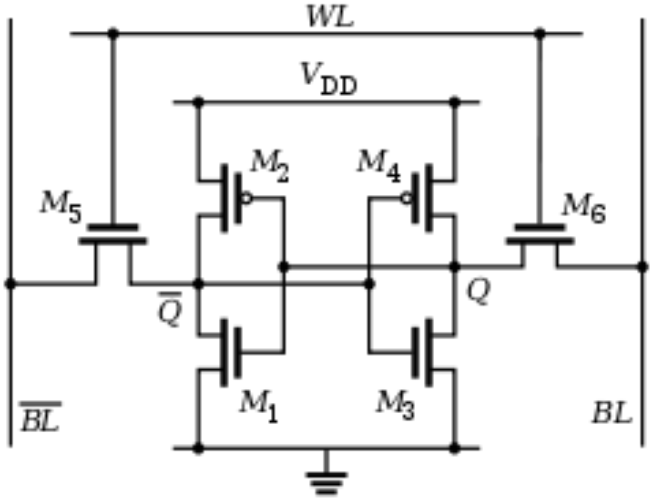
Регистры без блокировки

- Общие регистры – базовый объект для общения потоков между собой

```
// Последовательная спецификация
class Register:
    int r

    def write(x):
        r = x

    def read():
        return r
```

Физические регистры

- Физические регистры **неатомарны**
  - Но работают **без ожидания**
  - **Булевы** (хранят только один бит)
  - Поддерживают только **одного читателя** и **одного писателя**
  - Попытка чтения и записи **одновременно** приводит к непредсказуемым результатам
  - Но они **безопасны** (safe)
    - После завершение записи, будет прочитано последнее записанное значение

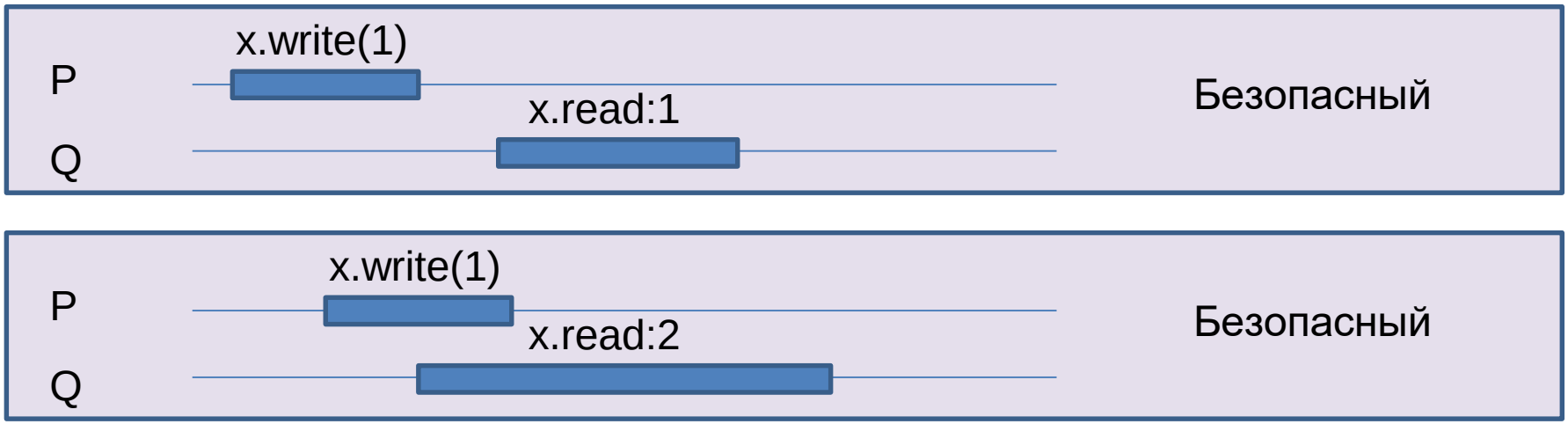


Классификация регистров

- **По условиям согласованности или корректности**
  - Безопасные (safe), регулярные (regular), атомарные (atomic)
- **По количеству потоков**
  - Один читатель, много читателей (SR, MR)
  - Один писатель, много писателей (SW, MW)
- **По количеству значений**
  - Булевские значение (boolean), множественные значения (M-valued)
- Иерархия типов регистров
  - Самый примитивный регистр – **Safe SRSW Boolean register**
  - Самый сложный регистр – **Atomic MRMW M-Valued register**

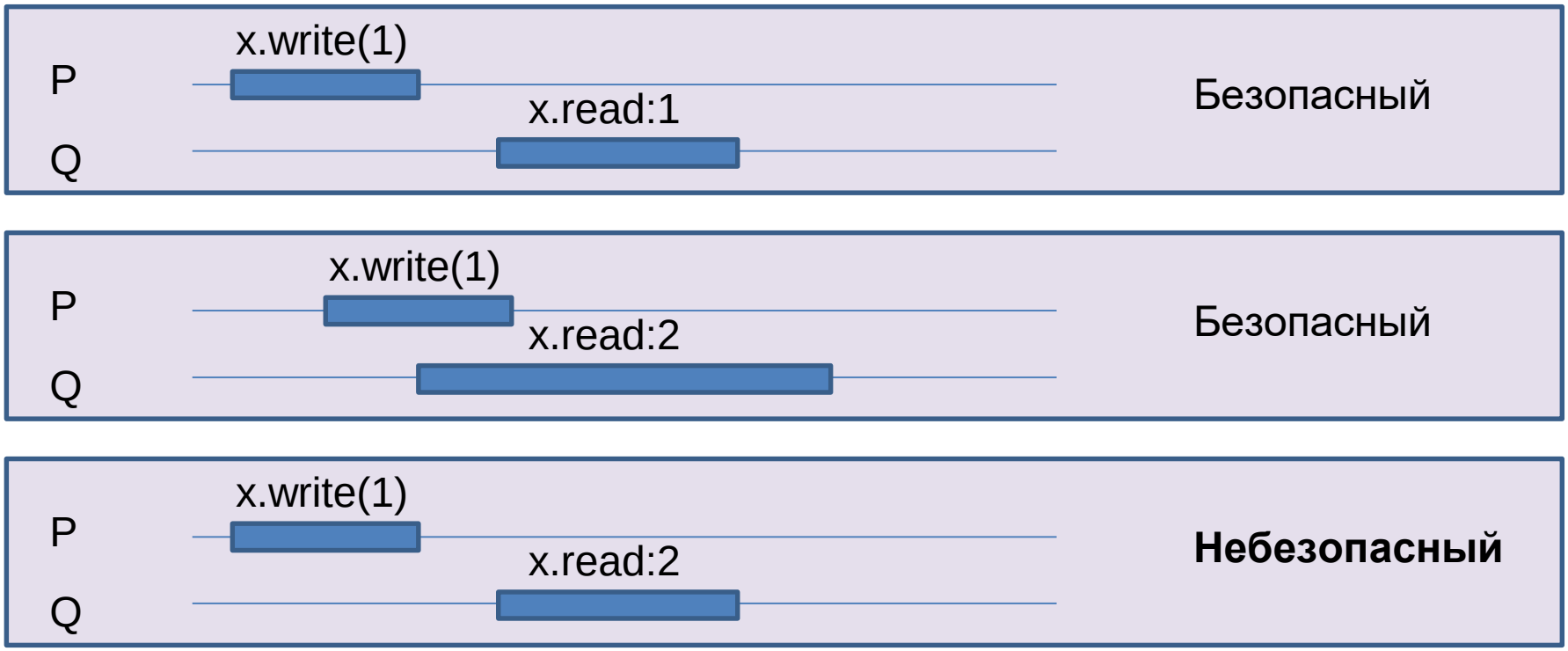
Безопасные (safe) регистры

- Гарантирует получение последнего записанного значения, если операция чтения не параллельна операциям записи



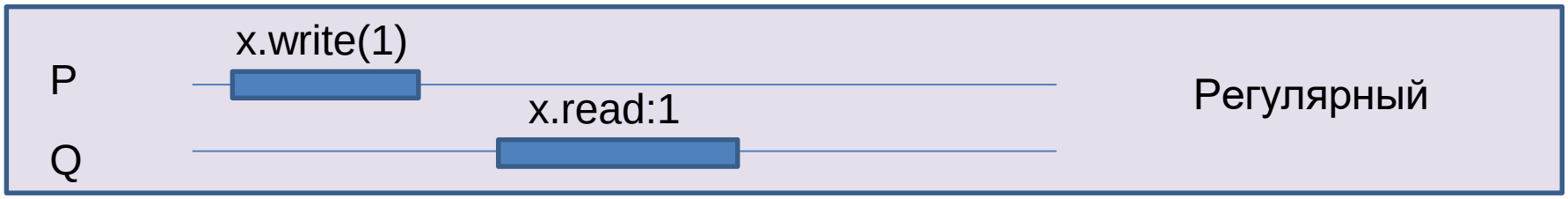
Безопасные (safe) регистры

- Гарантирует получение последнего записанного значения, если операция чтения не параллельна операциям записи



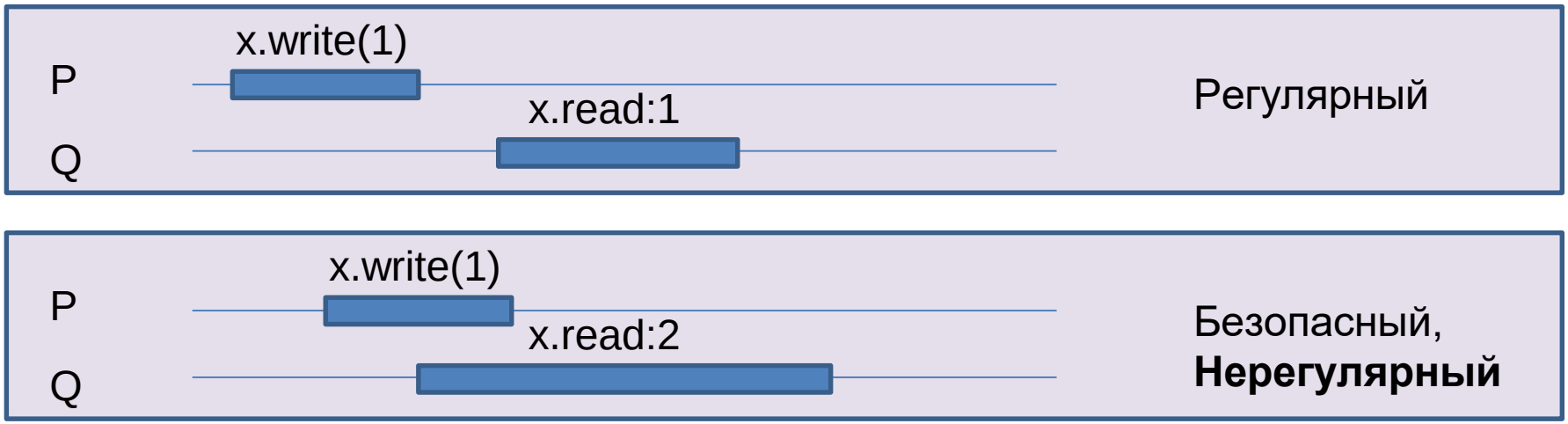
Регулярные (regular) регистры

- При чтении выдает либо последнее записанное, либо одно из тех значений, который сейчас пишутся



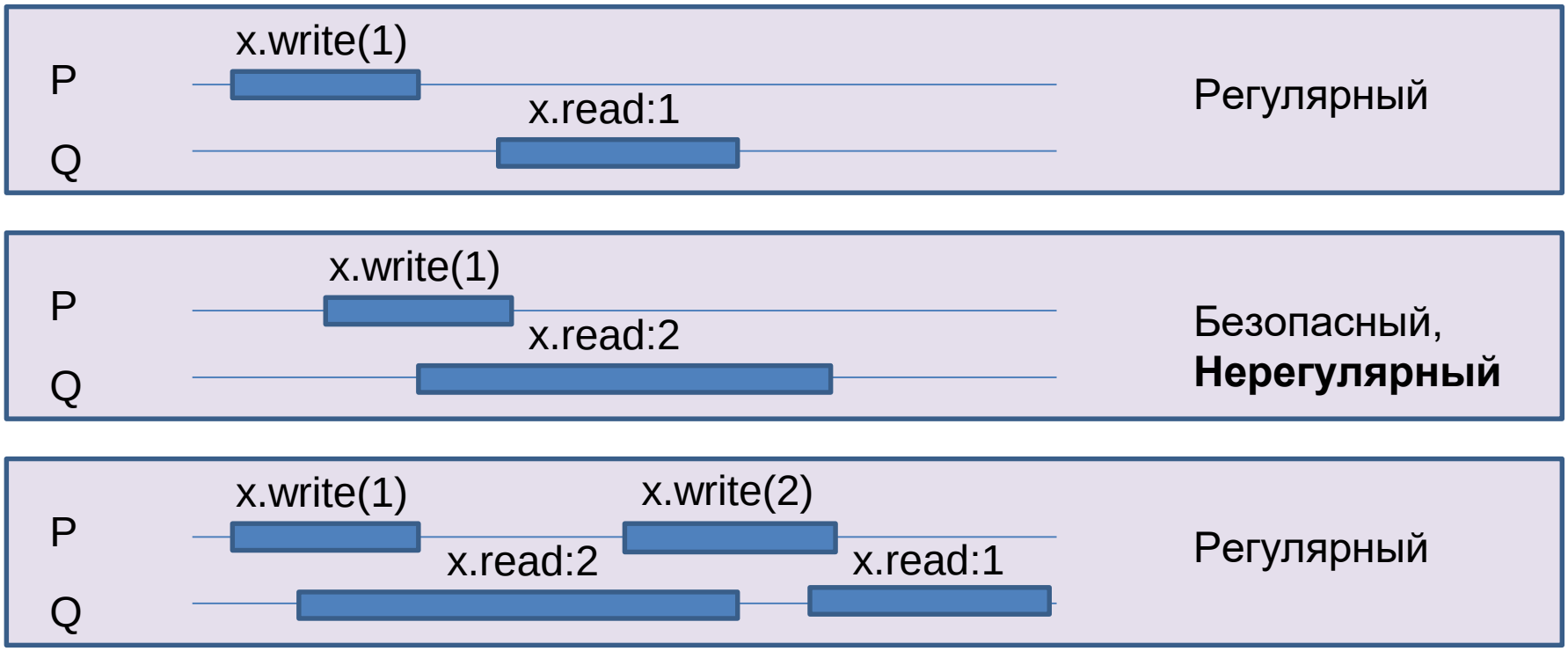
Регулярные (regular) регистры

- При чтении выдает либо последнее записанное, либо одно из тех значений, который сейчас пишутся



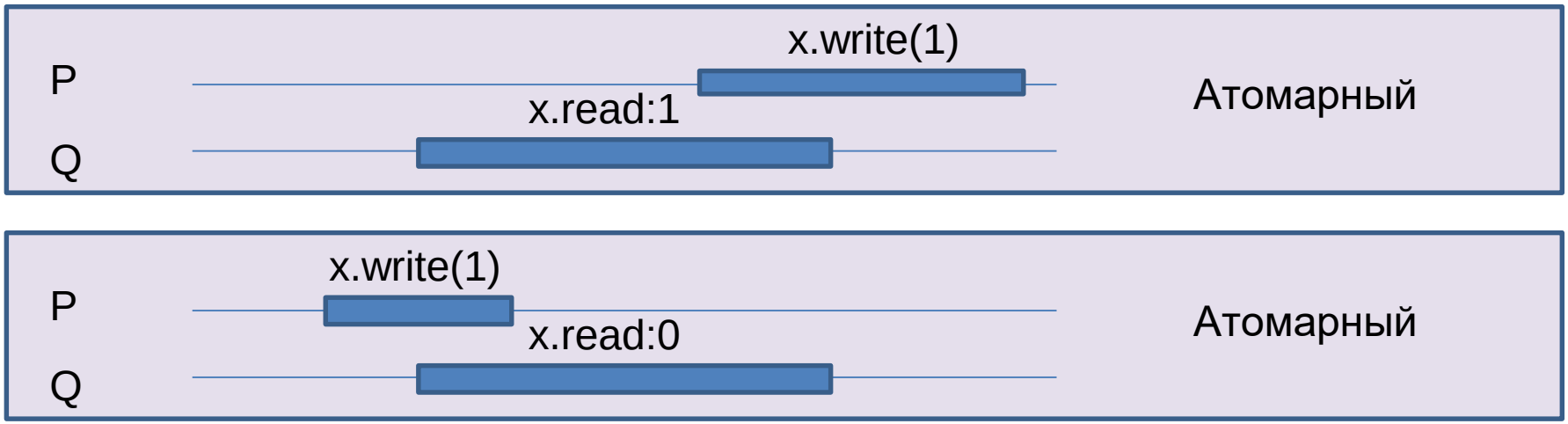
Регулярные (regular) регистры

- При чтении выдает либо последнее записанное, либо одно из тех значений, который сейчас пишутся



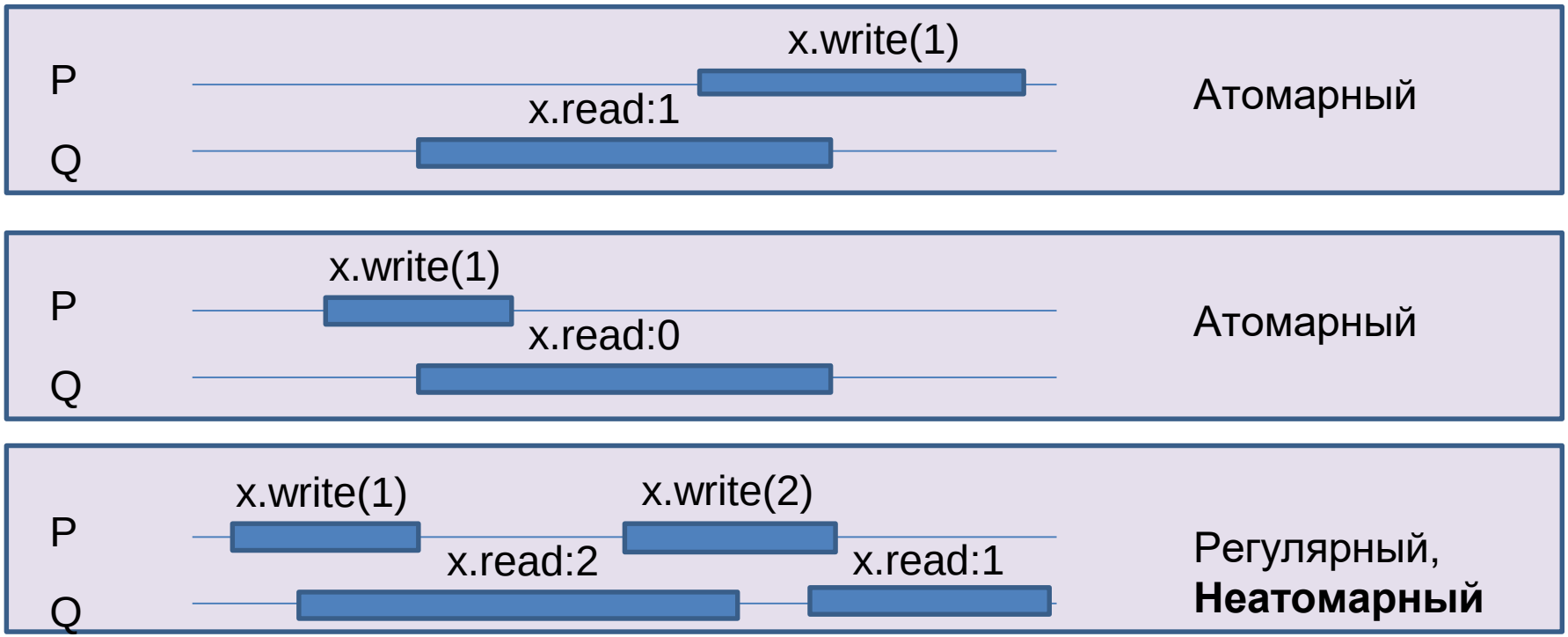
Атомарные (atomic) регистры

- Исполнение линеаризуемо



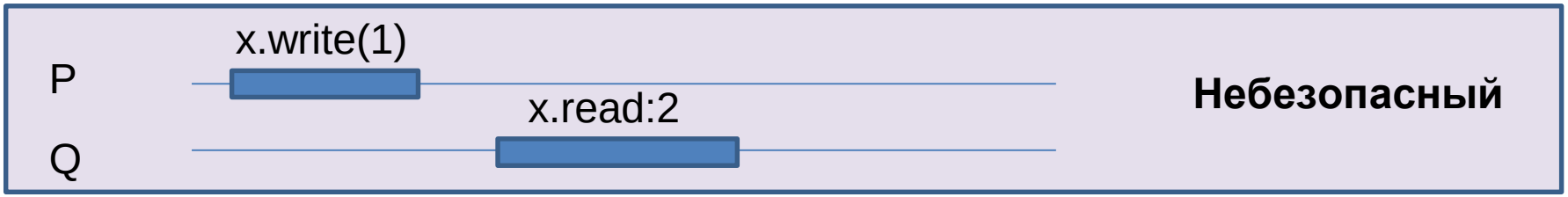
Атомарные (atomic) регистры

- Исполнение линеаризуемо

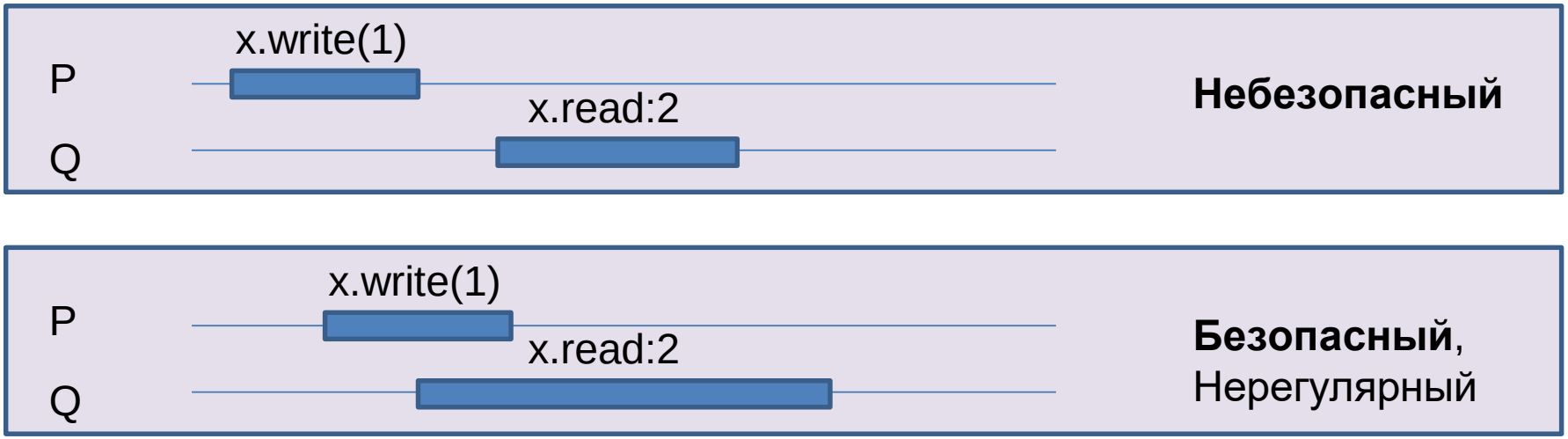


Все условия согласованности регистров

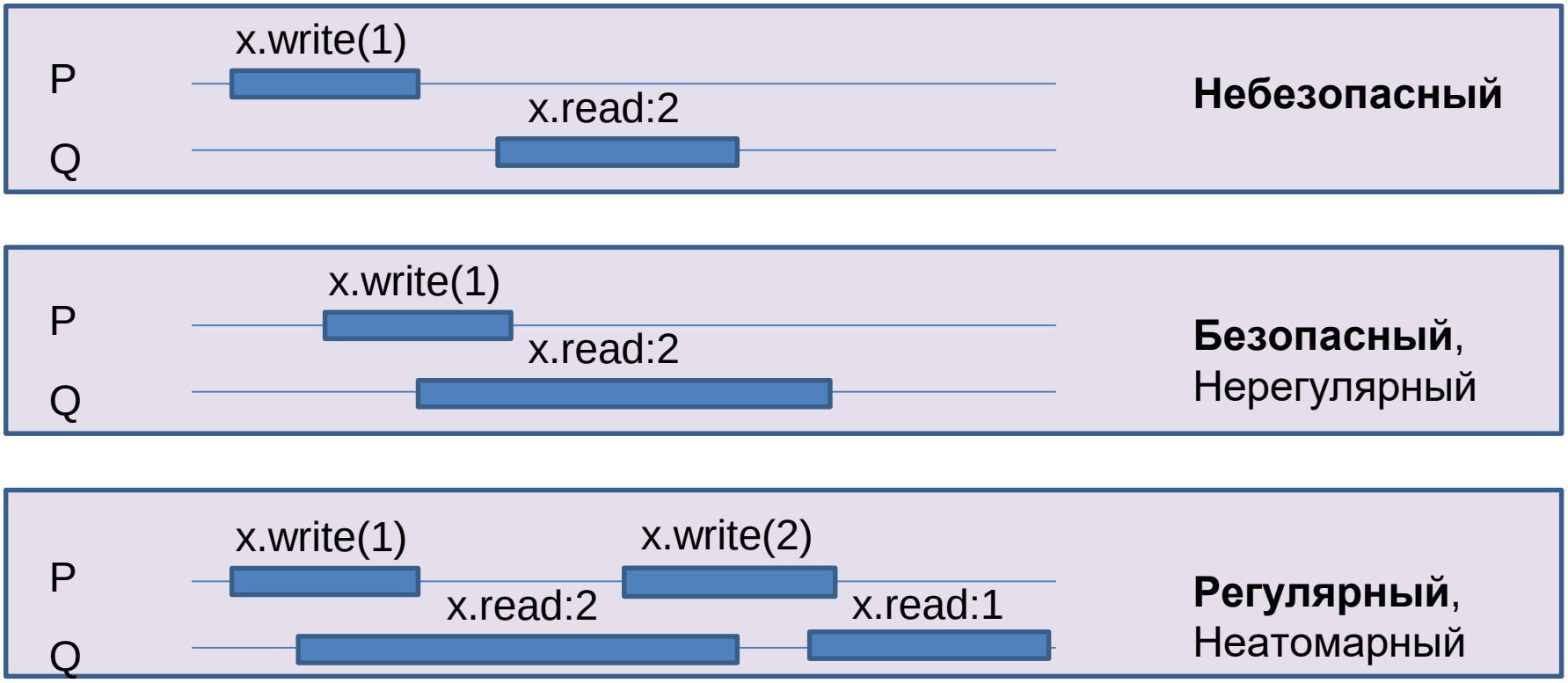
Все условия согласованности регистров



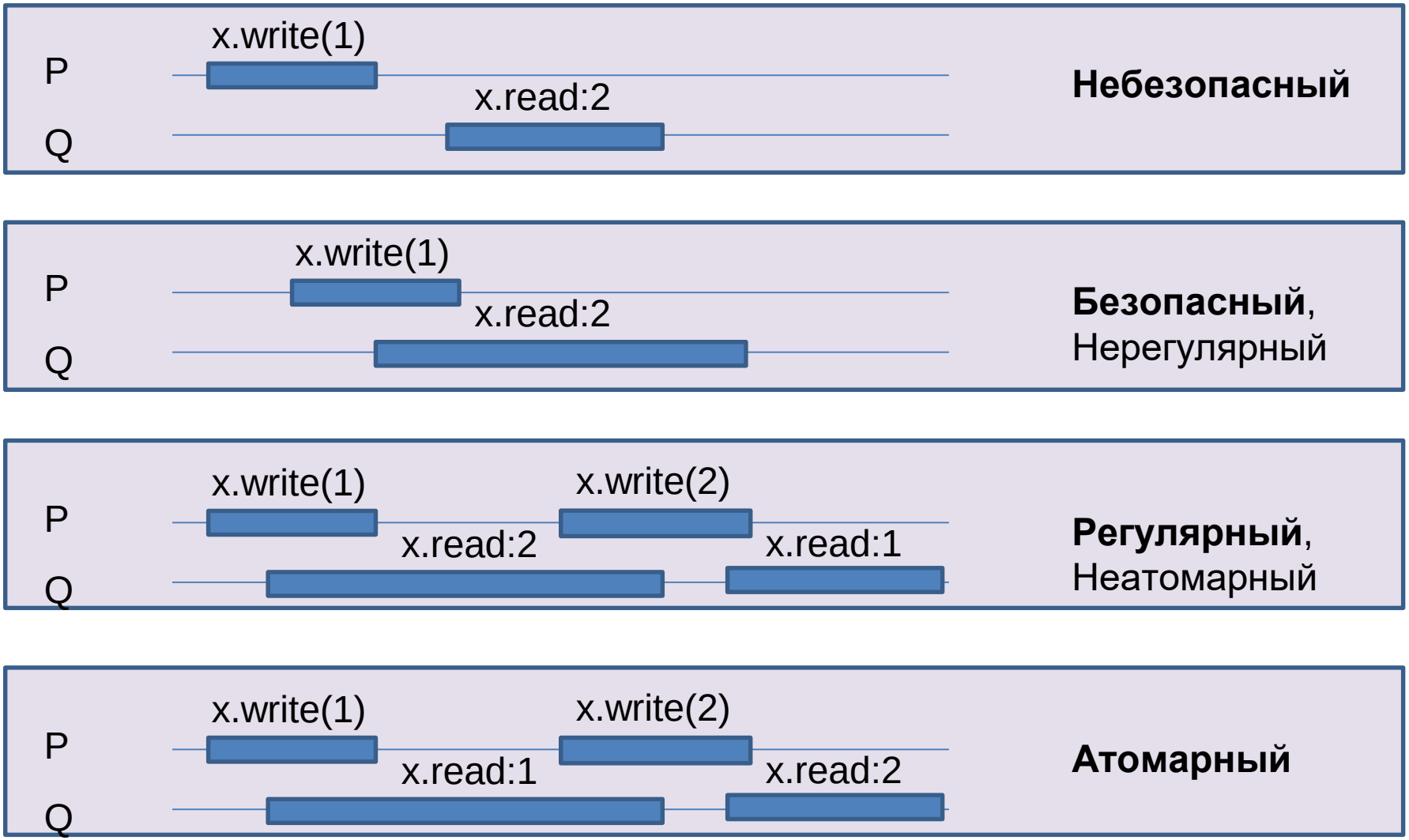
Все условия согласованности регистров



Все условия согласованности регистров

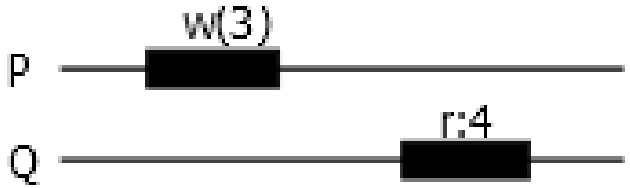


Все условия согласованности регистров

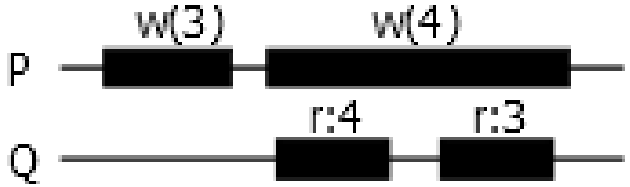


⚠Контрольный вопрос: Регистры 1-3

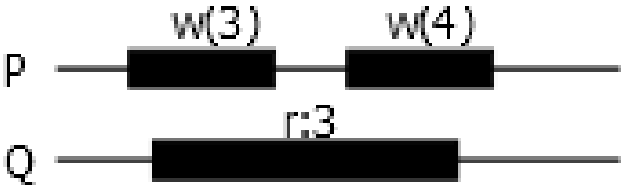
Небезопасный/Безопасный/Регулярный/Атомарный?

1. 

Небезопасный

2. 

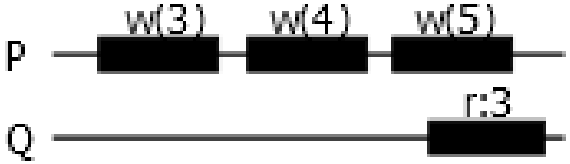
Регулярный

3. 

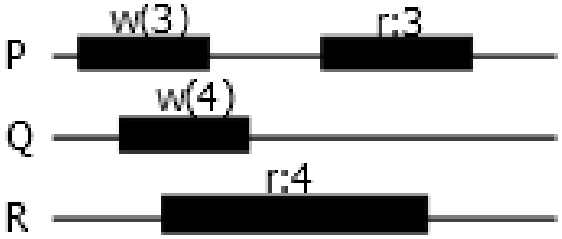
Атомарный

⚠Контрольный вопрос: Регистры 4-6

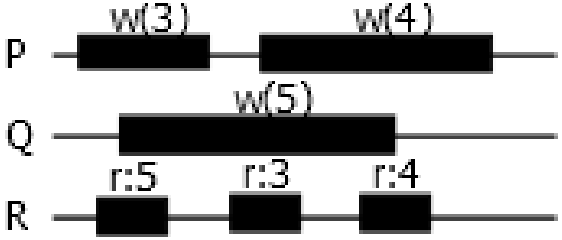
Небезопасный/Безопасный/Регулярный/Атомарный?

4. 

Безопасный

5. 

Атомарный

6. 

Атомарный

Построение регистров

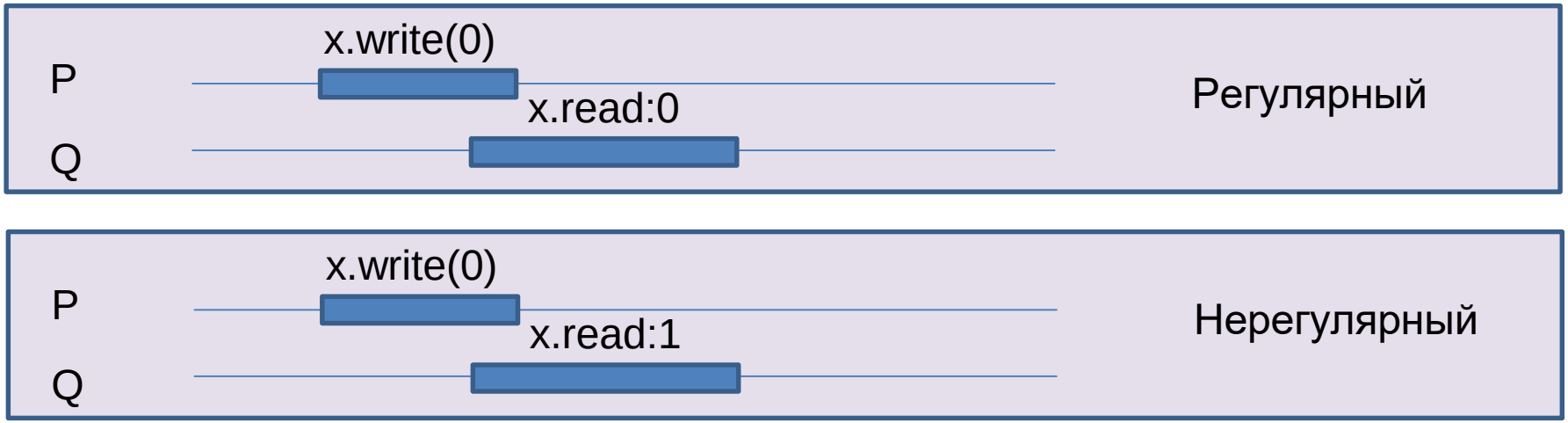
- Будем строить более сложные регистры из более простых требуя, чтобы реализация была **без ожидания** (wait-free образом).
  - **Safe SRSW Boolean register** – дан в начале
  - Regular SRSW Boolean register
  - Regular SRSW M-Valued register
  - Atomic SRSW M-Valued register
  - Atomic MRSW M-Valued register
  - **Atomic MRMW M-Valued register** – хотим построить

Регулярный SRSW булев регистр

Дано: Безопасный SRSW булев регистр

Регулярный SRSW булев регистр

Дано: Безопасный SRSW булев регистр

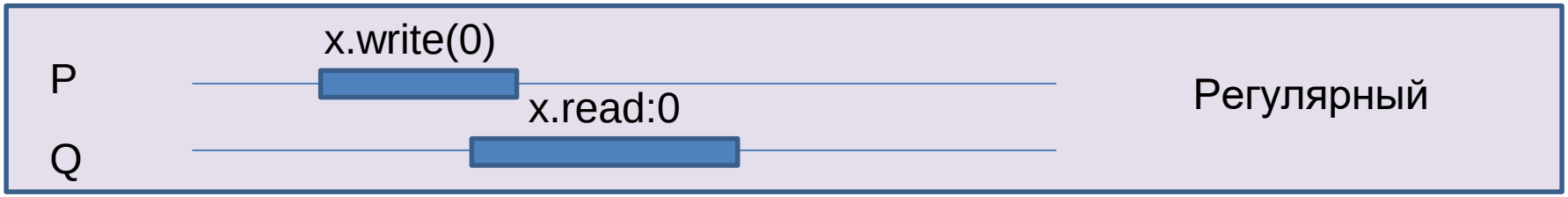


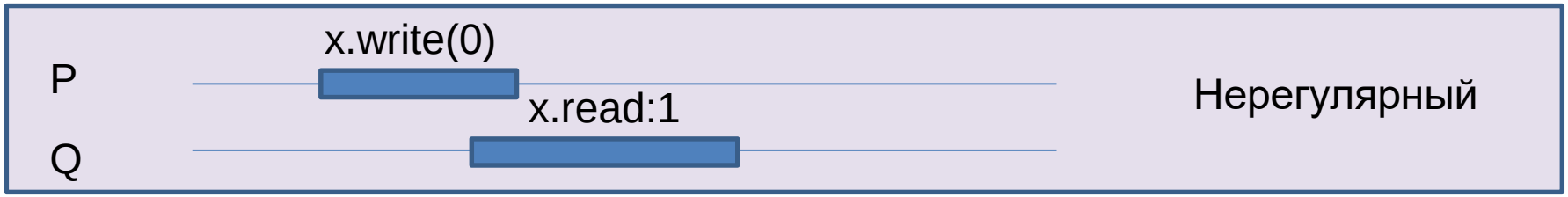
Регулярный SRSW булев регистр

Дано: Безопасный SRSW булев регистр

```
safe shared boolean r
threadlocal  boolean last

def write(x):
    if x != last:
        last = x
        r = x

def read(): return r
```

- У **булева** регистра только два значения – 0 и 1.
- **Один писатель**. Запоминаем последнее записанное значение и не перезаписываем





Регулярный SRSW регистр M значений

Дано: Регулярный SRSW булев регистр

Регулярный SRSW регистр M значений

Дано: Регулярный SRSW булев регистр

Индекс i	**0**	**1**	**2**	**3**	
r[i]	1	1	1	0	Значение 3
r[i]	1	0	0	0	Значение 1

- Запоминаем M значений в унарном коде используя M регистров
  - **Индекс первого нуля** определяет значение

Регулярный SRSW регистр M значений

Дано: Регулярный SRSW булев регистр

```
regular shared boolean[M] r

def write(x): // Справа-на-лево
    r[x] = 0
    for i = x-1 downto 0: r[i] = 1

def read(): // Слева-на-право
    for i = 0 to M-1:
        if r[i] == 0:
            return i
```

- Запоминаем M значение в унарном коде используя M регистров
  - Индекс первого нуля определяет значение
- **Чтение и запись происходят в разном порядке**
- Результирующий регистр регулярен

Индекс i	**0**	**1**	**2**	**3**	
r[i]	1	1	1	0	Значение 3
r[i]	1	0	0	0	Значение 1

Регулярный SRSW регистр M значений

Дано: Регулярный SRSW булев регистр

Индекс i	**0**	**1**	**2**	**3**
r[i]	1	1	1	0

Значение 3

Регулярный SRSW регистр M значений

Дано: Регулярный SRSW булев регистр

Индекс i	**0**	**1**	**2**	**3**
r[i]	1	1	1	0
r[i]	1	0	1	0

Значение 3

Начинаем писать 1

Читатель (слева-на-право) видит
1 или 3 в зависимости от его индекса в
момент этой записи

Регулярный SRSW регистр M значений

Дано: Регулярный SRSW булев регистр

Индекс i	**0**	**1**	**2**	**3**
r[i]	1	1	1	0
r[i]	1	0	1	0

Значение 3

Начинаем писать 1

Читатель (слева-на-право) видит
1 или 3 в зависимости от его индекса в
момент этой записи

Мусор
останется,
не мешает!

Регулярный SRSW регистр M значений

Дано: Регулярный SRSW булев регистр

Индекс i	**0**	**1**	**2**	**3**
r[i]	1	0	0	0

Значение 1

Регулярный SRSW регистр M значений

Дано: Регулярный SRSW булев регистр

Индекс i	**0**	**1**	**2**	**3**
r[i]	1	0	0	0
r[i]	1	0	0	0

Значение 1

Начинаем писать 3, читатель видит 1

Регулярный SRSW регистр M значений

Дано: Регулярный SRSW булев регистр

Индекс i	**0**	**1**	**2**	**3**
r[i]	1	0	0	0
r[i]	1	0	0	0
r[i]	1	0	1	0

Значение 1

Начинаем писать 3, читатель видит 1

Продолжаем писать 3, читатель видит 1

Регулярный SRSW регистр M значений

Дано: Регулярный SRSW булев регистр

Индекс i	**0**	**1**	**2**	**3**
r[i]	1	0	0	0
r[i]	1	0	0	0
r[i]	1	0	1	0
r[i]	1	1	1	0

Значение 1

Начинаем писать 3, читатель видит 1

Продолжаем писать 3, читатель видит 1

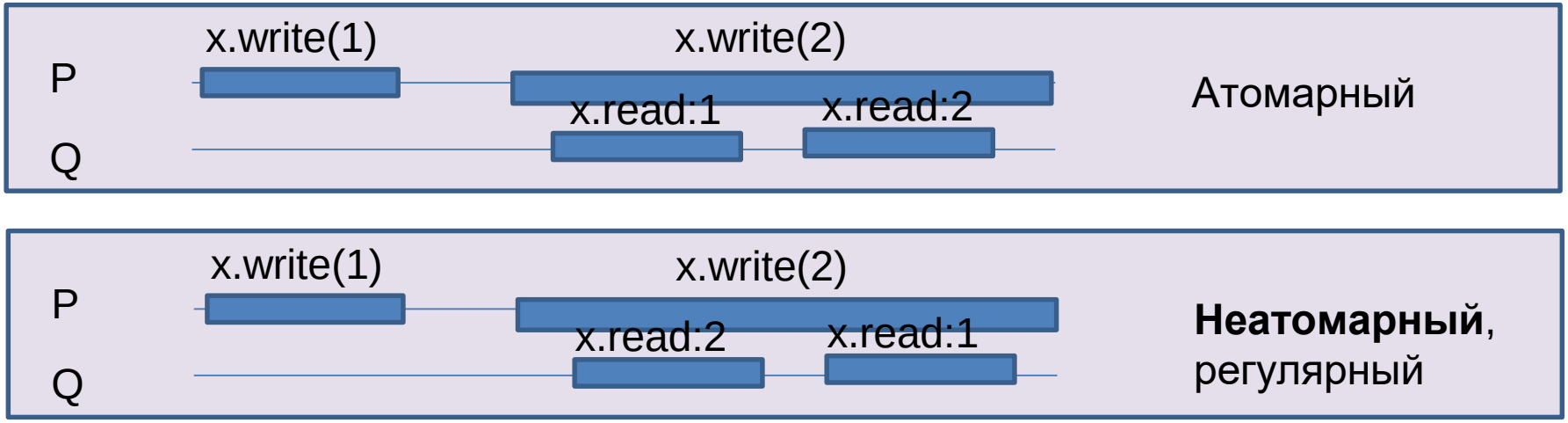
Продолжаем писать 3, **читатель видит 3**

Атомарный SRSW регистр с версиями

Дано: Регулярный SRSW регистр M значений

Атомарный SRSW регистр с версиями

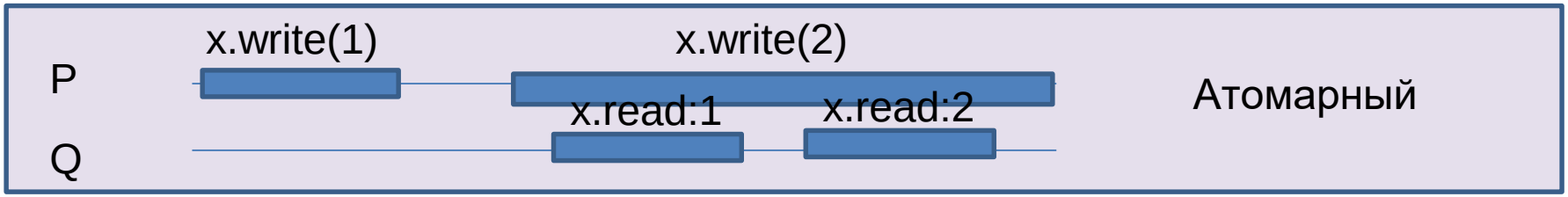
Дано: Регулярный SRSW регистр M значений

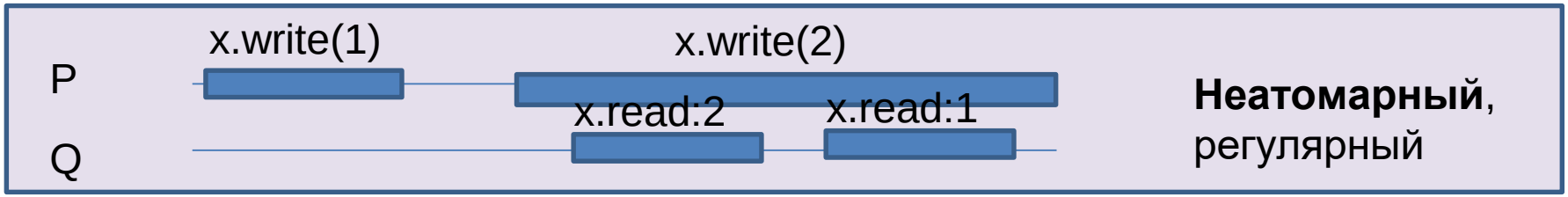


Атомарный SRSW регистр с версиями

Дано: Регулярный SRSW регистр M значений

- Атомарный регистр не может «возвращаться назад во времени» если несколько чтений перекрываются с одной записью
  - Идея: Отследим время через версию значения





Атомарный SRSW регистр с версиями

Дано: Регулярный SRSW регистр M значений

```
regular shared (int x, int v) r

threadlocal (int x, int v) lastRead
threadlocal  int lastWriteV

def write(x):
    lastWriteV++
    r = (x, lastWriteV)

def read():
    cur = r
    if cur.v > lastRead.v:
        lastRead = cur
    return lastRead.x
```

- На практике это отличное решение, ибо размер версии можно разумно ограничить практическими соображениями

Атомарный регистр: проблемы

Дано: Регулярный SRSW регистр M значений

- **Версии**
  - Может хранить пару (версия, значение) в регулярном регистре
  - Но версии растут неограниченно

Атомарный регистр: проблемы

Дано: Регулярный SRSW регистр M значений

- **Версии**
  - Может хранить пару (версия, значение) в регулярном регистре
  - Но версии растут неограниченно
- **Используем блокировки чтобы получить атомарность?**
  - Алгоритм Лампорта будет работать на регулярных регистрах
  - Но это не дает нам алгоритм **без ожидания**

Атомарный регистр: проблемы

Дано: Регулярный SRSW регистр M значений

- **Версии**
  - Может хранить пару (версия, значение) в регулярном регистре
  - Но версии растут неограниченно
- **Используем блокировки чтобы получить атомарность?**
  - Алгоритм Лампорта будет работать на регулярных регистрах
  - Но это не дает нам алгоритм **без ожидания**
- **Теорема:**
  - Не существует алгоритма построения атомарного регистра без ожидания, которые использует конечное число регулярных регистров конечного размера так, чтобы их писал только писатель, а читал только читатель
    - Нужна обратная связь от читателя к писателю!

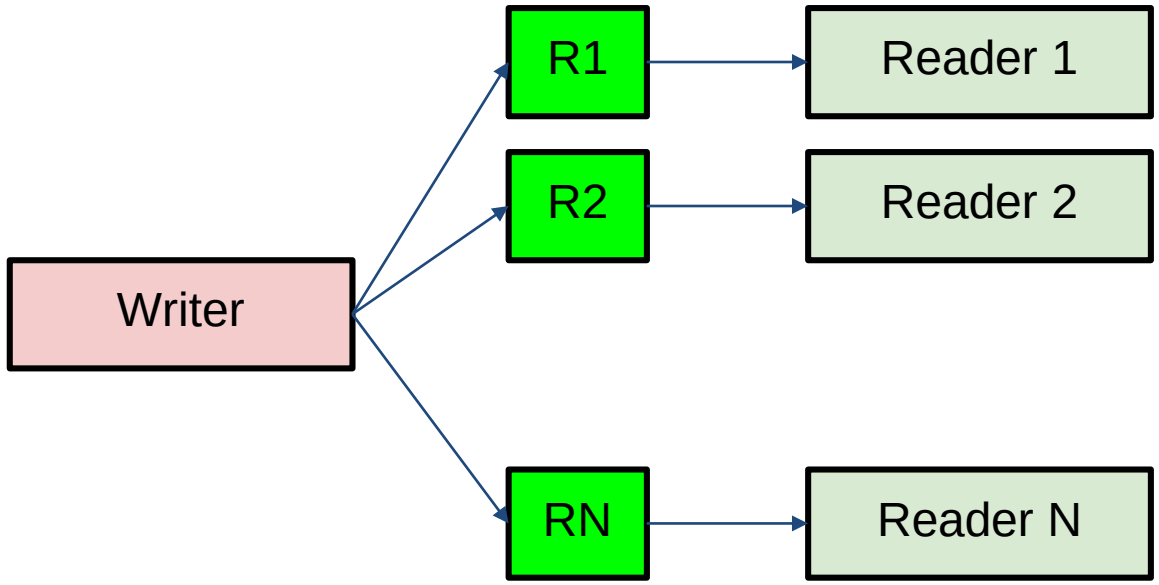
Атомарный MRSW регистр

Дано: Атомарный SRSW регистр M значений

Атомарный MRSW регистр

Дано: Атомарный SRSW регистр M значений

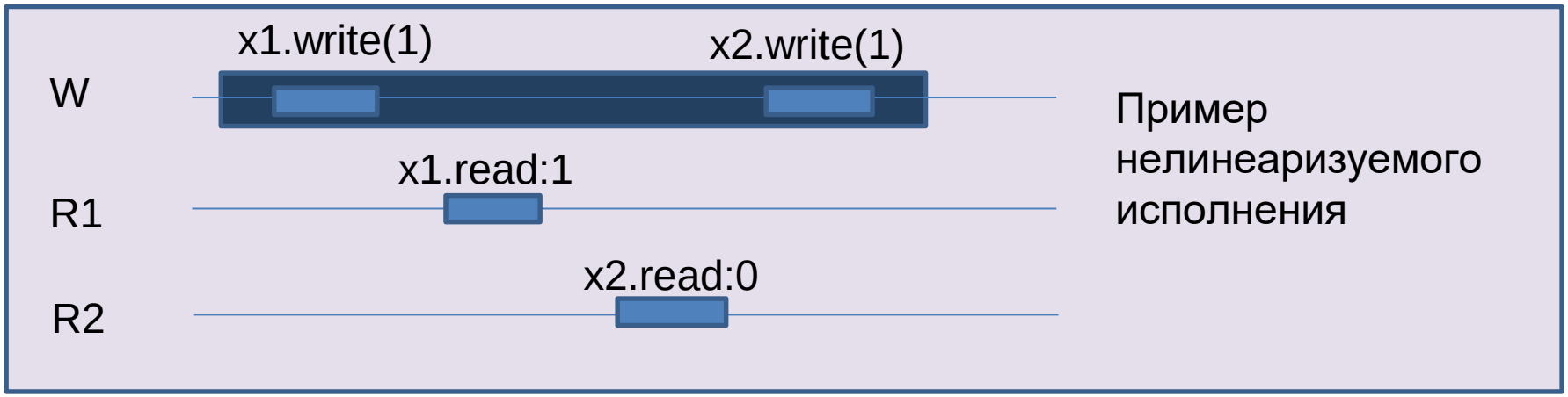
- Нужна поддержка N **читателей**
- Заводим по SRSW регистру для каждого читателя и пишем в каждый из них.



Атомарный MRSW регистр

Дано: Атомарный SRSW регистр M значений

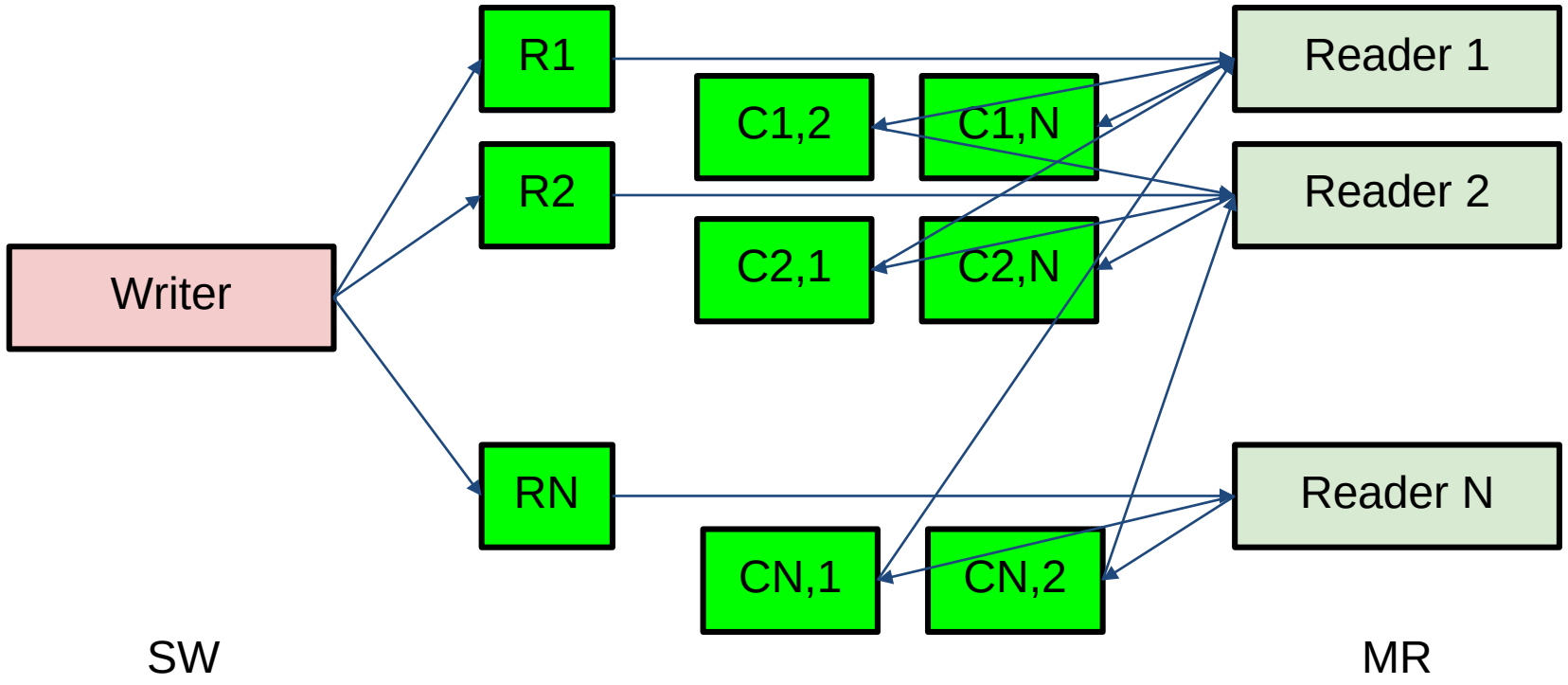
- Нужна поддержка N **читателей**
- Заводим по SRSW регистру для каждого читателя и пишем в каждый из них.
  - Не получается линеаризуемый (атомарный) алгоритм



Атомарный MRSW регистр с версиями

Дано: Атомарный SRSW регистр M значений

- Отслеживаем версию записанного значения храня пару (x,v) в каждом из N регистров в которые пишет писатель
- Заводим N*(N-1) регистров для общения между читателями



Атомарный MRSW регистр с версиями

Дано: Атомарный SRSW регистр M значений

- Отслеживаем версию записанного значения храня пару (x,v) в каждом из N регистров в которые пишет писатель
- Заводим N*(N-1) регистров для общения между читателями
  - Каждый читатель выбирает более позднее значение из записанного писателем и из прочитанных значений других читателей.
  - После этого читатель записывает свое прочитанное значение и версию для всех остальных читателей.

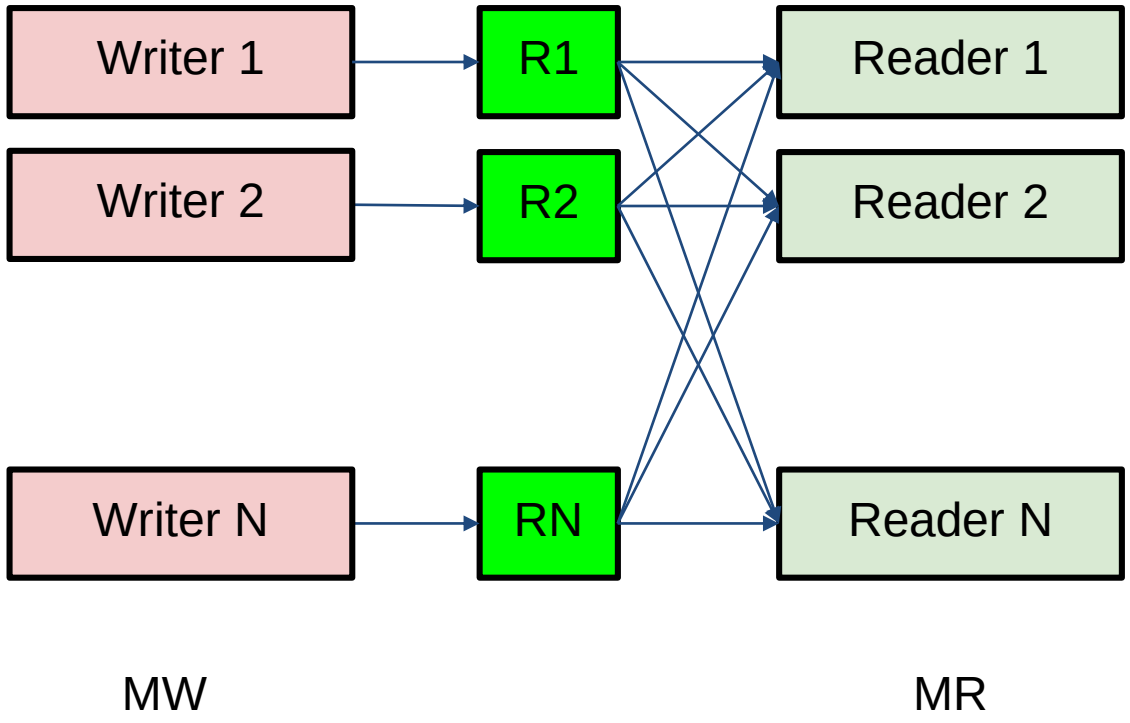
Атомарный MRMW регистр с версиями

Дано: Атомарный MRSW регистр M значений

Атомарный MRMW регистр с версиями

Дано: Атомарный MRSW регистр M значений

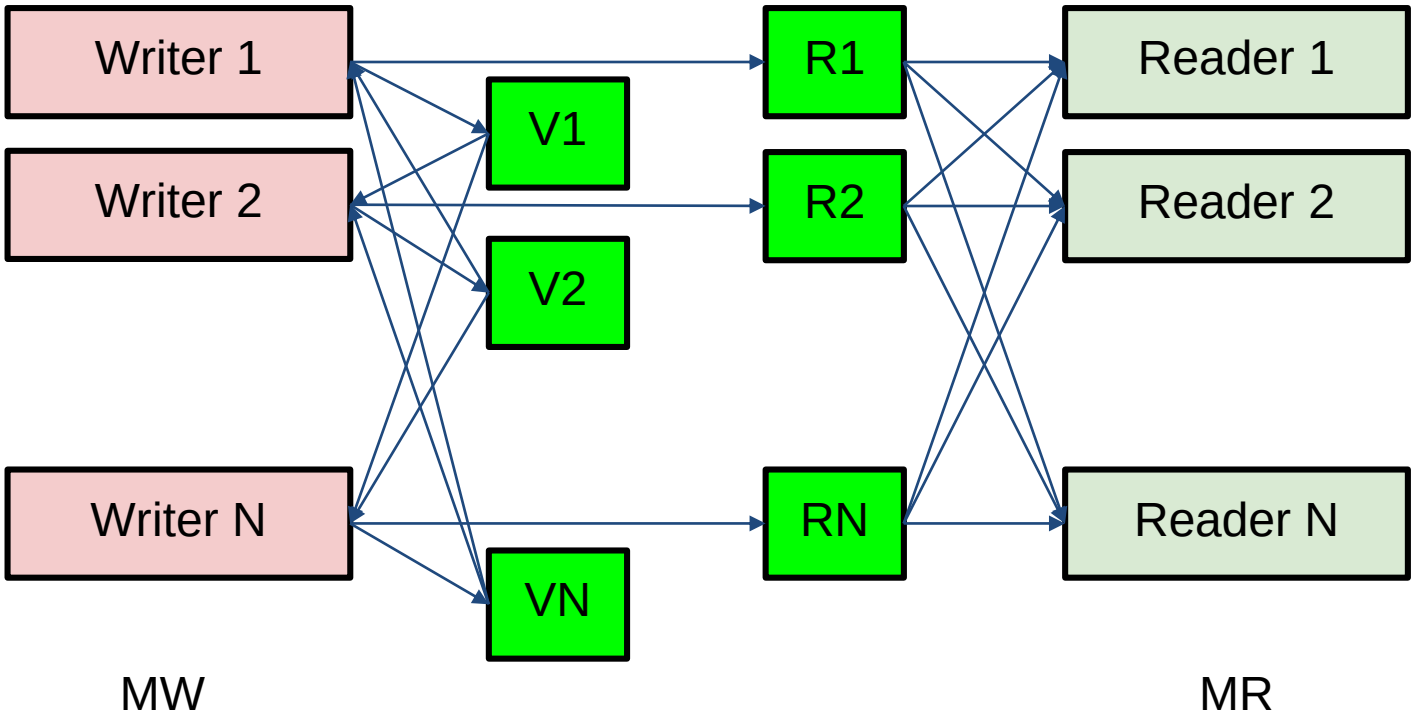
- Нужна поддержка N **писателей**



Атомарный MRMW регистр с версиями

Дано: Атомарный MRSW регистр M значений

- Нужна поддержка N **писателей**
- Отслеживаем версию записанного значения
  - Каждый читатель выбирает более позднюю версию



Атомарный MRMW регистр с версиями

Дано: Атомарный MRSW регистр M значений

- Нужна поддержка N **писателей**
- Отслеживаем версию записанного значения
  - Каждый читатель выбирает более позднюю версию
  - Для проставления версий писателями используем doorway секцию из алгоритма булочника (алгоритма взаимного исключения Лампорта)
    - Версия будет состоять из пары номера потока писателя и собственно числа.

Атомарный снимок состояния N регистров

Атомарный снимок состояния N регистров

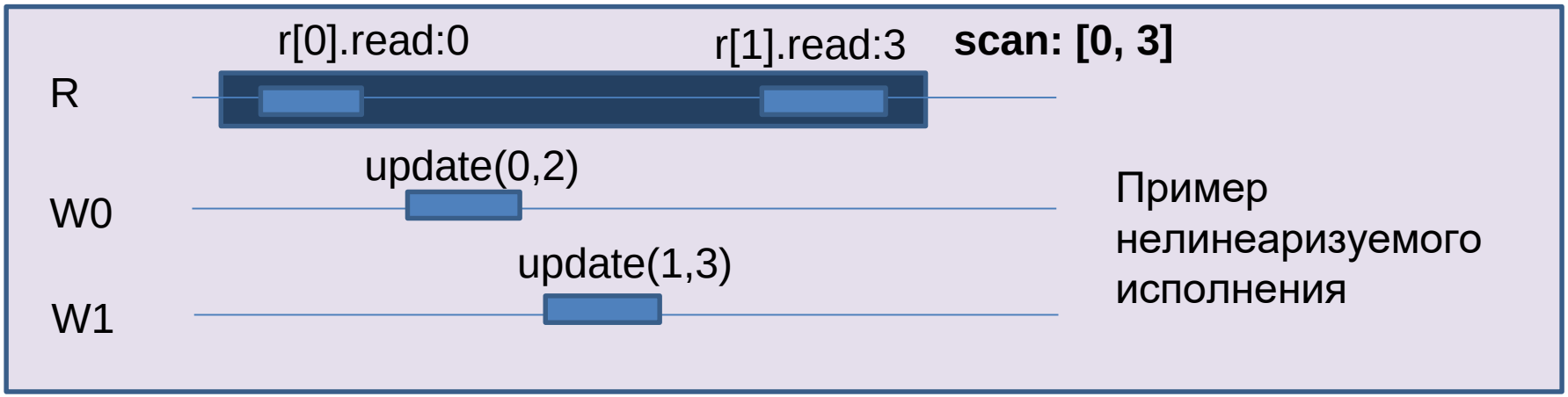
```
// Последовательная спецификация
class Snapshot:
    shared int r[N]

    def update(i, x):
        r[i] = x

    def scan():
        return copy()

    private def copy():
        res = new int[N]
        for i = 0..N-1: res[i] = r[i]
        return res
```

- Набор SW атомарных регистров (по регистру на поток)
- Любой поток может вызвать scan() чтобы получить снимок состояния всех регистров
- Методы должны быть атомарными (линеаризуемыми)

Атомарный снимок состояния N регистров

- Наивная реализация не обеспечивает атомарность

Операция	r[0]	r[1]
update(0,2)	2	0
update(1,3)	2	3



Атомарный снимок состояния N регистров, без блокировок (lock free)

```
shared (int x, int v) r[N]

// wait-free
def update(i, x):
    r[i] = (x, r[i].v + 1)

// lock-free
def scan():
    old = copy()
    loop:
        cur = copy()
        if forall i: cur[i].v == old[i].v:
            return cur.x
        old = cur
```

- Каждый регистр хранит версию
- При обновлении версию увеличиваем на один
- При чтении читает до тех пор, пока не получит два подряд одинаковых снимка

Атомарный снимок состояния N регистров, без ожидания (wait-free) – update

```
shared (int x, int v, int[N] s) r[N]

def update(i, x):
    s = scan()
    r[i] = (x, r[i].v + 1, s)
```

- Для реализации **без ожидания** надо чтобы потоки **помогали** друг другу
- Каждый регистр также хранит копию снимка s
- При обновлении делаем вложенный scan, чтобы помочь параллельно работающим операциям

Атомарный снимок состояния N регистров, без ожидания (wait-free) – scan

```
shared (int x, int v, int[N] s) r[N]

// wait-free, O(N^2)
def scan():
    old = copy()
    boolean updated[N]
    loop:
        cur = copy()
        for i = 0..N-1:
            if cur[i].v != old[i].v:
                if updated[i]: return cur[i].s
                else:
                    update[i] = true
                    old = cur
                    continue loop
        return cur.x
```

- **Лемма**: Если значение изменилось второй раз, то хранящаяся там копия снимка s была получена вложенной операцией scan.
- **Следствие**: Этот алгоритм выдает атомарный снимок

Атомарный снимок состояния N регистров, без ожидания (wait-free) – лемма

- **Лемма**: Если значение изменилось второй раз, то хранящаяся там копия снимка s была получена вложенной операцией scan.