

Аппаратная транзакционная память в Java

Никита Коваль

ndkoval *at* ya *dot* ru
twitter.com/nkoval_

Осторожно! Этот доклад про МНОГОПОТОЧНОСТЬ!*

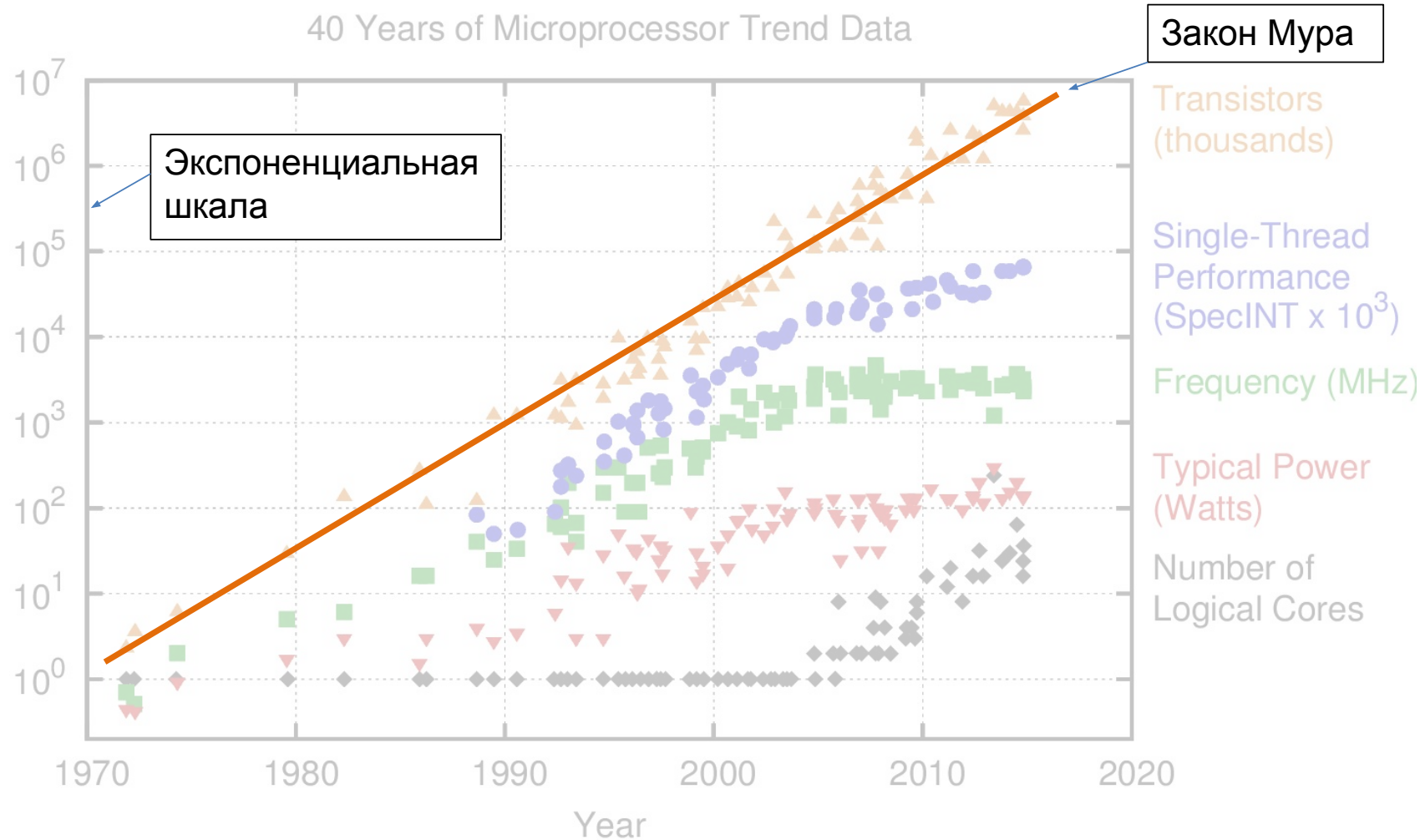
* В других залах: профилирование в AOT (зал 2), микросервисы (зал 3) и машинное обучение (зал 4).

О себе

- Инженер-исследователь в лаборатории dxLab, Devexperts
- Преподаю курс по многопоточному программированию в ИТМО
- Заканчиваю магистратуру ИТМО

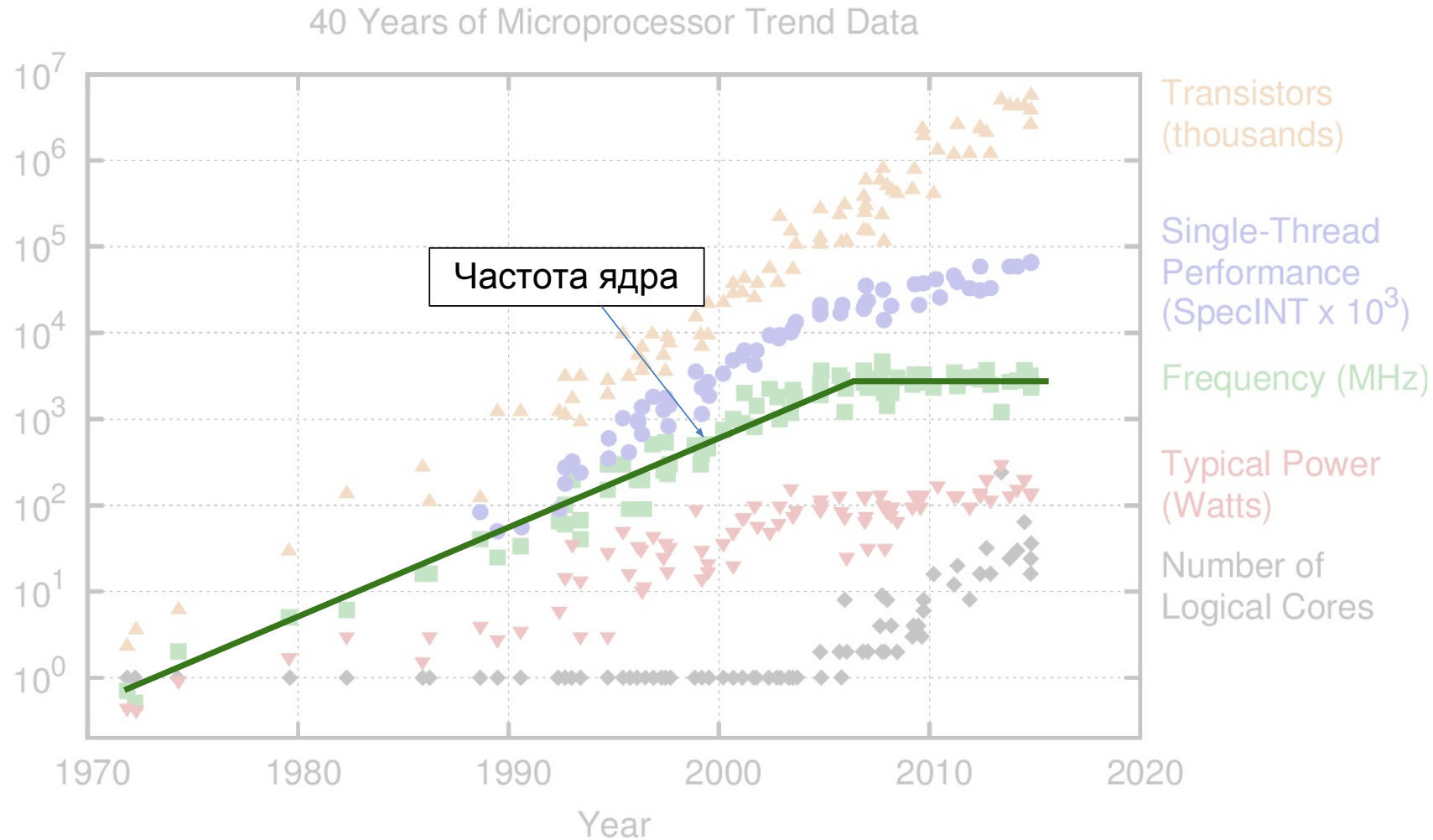


Многопоточность: а надо ли?



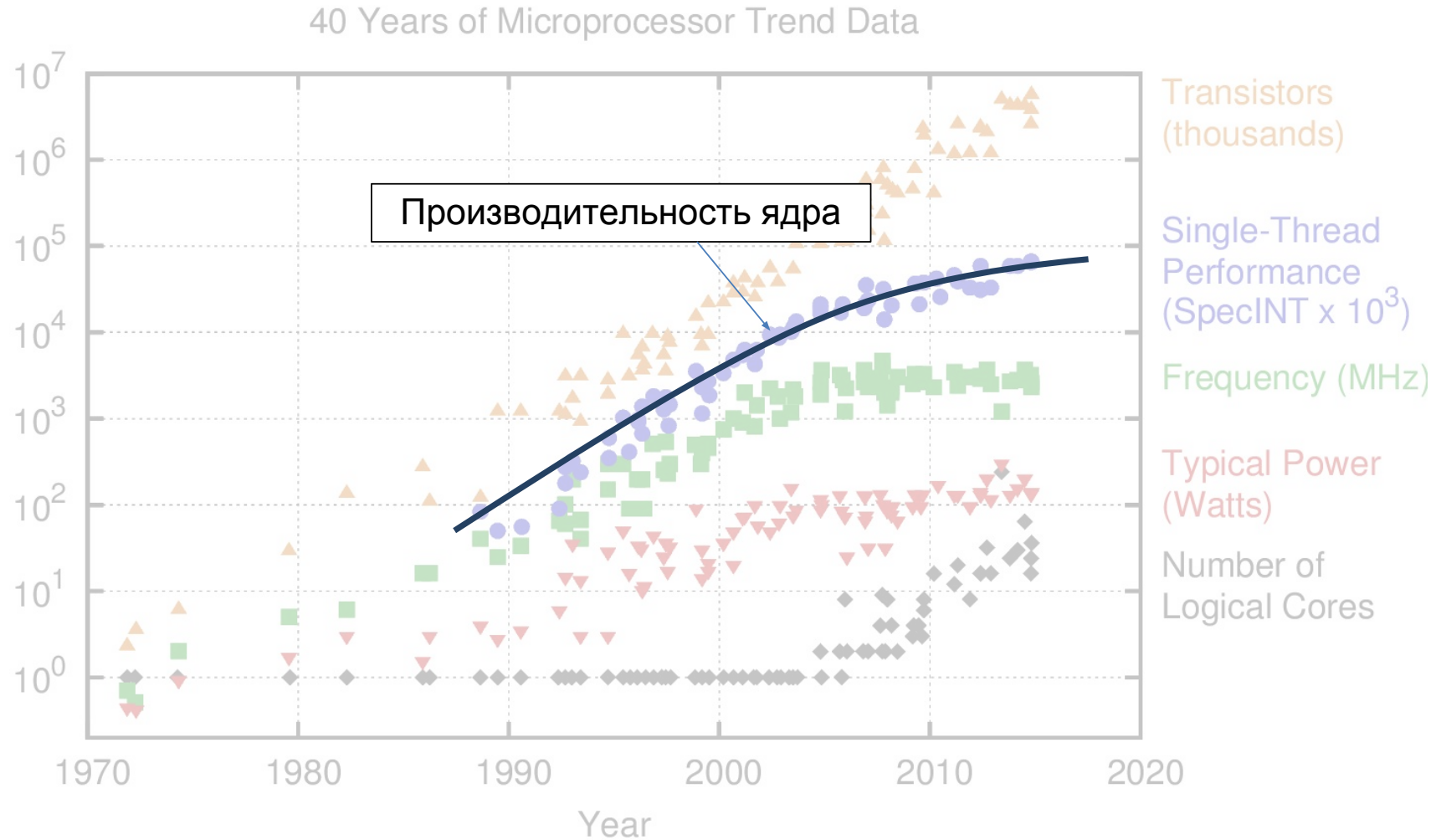
Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2015 by K. Rupp

Многопоточность: а надо ли?



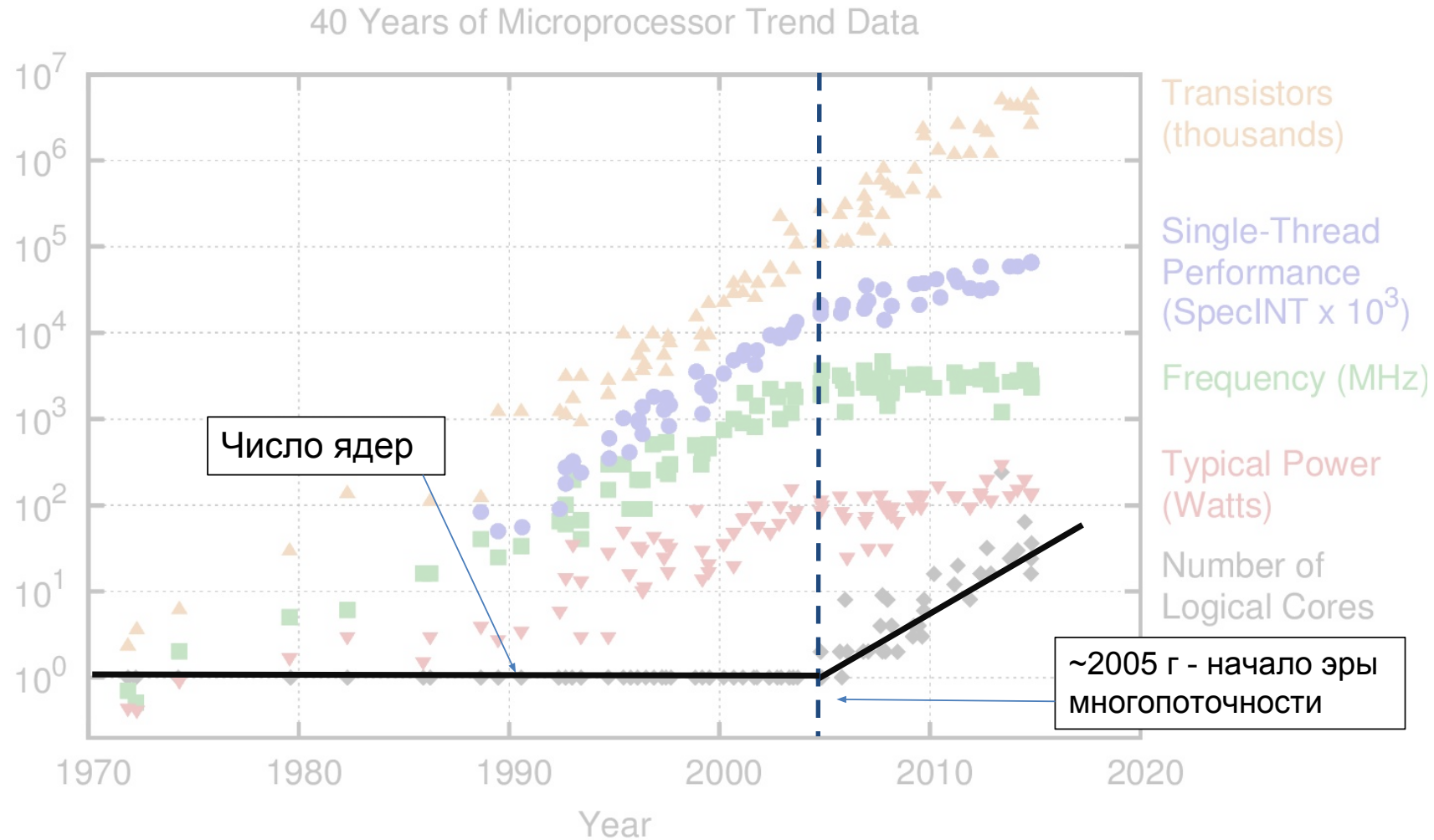
Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2015 by K. Rupp

Многопоточность: а надо ли?



Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2015 by K. Rupp

Многопоточность: а надо ли?



Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2015 by K. Rupp

Подходы к построению алгоритмов

- Грубая блокировка
- Тонкая блокировка
- Неблокирующая синхронизация

Выберем задачу

Игрушечный банк

```
interface Bank {  
    fun getAmount(id: Int): Long  
  
    fun deposit(id: Int, amount: Long): Long  
    fun withdraw(id: Int, amount: Long): Long  
  
    fun transfer(fromId: Int, toId: Int, amount: Long)  
}
```

Игрушечный банк

```
interface Bank {  
    fun getAmount(id: Int): Long  
    fun deposit(id: Int, amount: Long): Long  
    fun withdraw(id: Int, amount: Long): Long  
    fun transfer(fromId: Int, toId: Int, amount: Long)  
}
```

Узнать баланс

Игрушечный банк

```
interface Bank {  
    fun getAmount(id: Int): Long  
  
    fun deposit(id: Int, amount: Long): Long  
    fun withdraw(id: Int, amount: Long): Long  
  
    fun transfer(fromId: Int, toId: Int, amount: Long)  
}
```

Пополнить/снять



Игрушечный банк

```
interface Bank {  
    fun getAmount(id: Int): Long  
  
    fun deposit(id: Int, amount: Long): Long  
    fun withdraw(id: Int, amount: Long): Long  
  
    fun transfer(fromId: Int, toId: Int, amount: Long)  
}
```



Перевести

Подходы к построению алгоритмов

- Грубая блокировка
- Тонкая блокировка
- Неблокирующая синхронизация

Грубая блокировка

- Coarse-Grained locking
- Берём блокировку при вызове каждой операции

Грубая блокировка

```
class CGBank(n: Int) : Bank {  
    private val accounts = Array(n, { Account() })  
    private val gLock = ReentrantLock()  
  
    ...  
  
    private class Account(var amount: Long = 0)  
}
```


Грубая блокировка

```
class CGBank(n: Int) : Bank {  
    private val accounts = Array(n, { Account() })  
    private val gLock = ReentrantLock() // global lock  
  
    ...  
  
    private class Account(var amount: Long = 0)  
}
```

Грубая блокировка

```
private val gLock = ReentrantLock()
```

```
fun getAmount(id: Int): Long = gLock.withLock {  
    return accounts[id].amount  
}
```

```
fun deposit(id: Int, amount: Long) = gLock.withLock {  
    accounts[id].amount += amount  
}
```

```
fun withdraw(id: Int, amount: Long) = gLock.withLock { ... }
```

```
fun transfer(fromId: Int, toId: Int, amount: Long) = gLock.withLock {  
    accounts[fromId].amount -= amount  
    accounts[toId].amount += amount  
}
```

Грубая блокировка

```
private val gLock = ReentrantLock()
```

```
fun getAmount(id: Int): Long = gLock.withLock {  
    return accounts[id].amount  
}
```

```
fun deposit(id: Int,  
    accounts[id].amount  
}
```

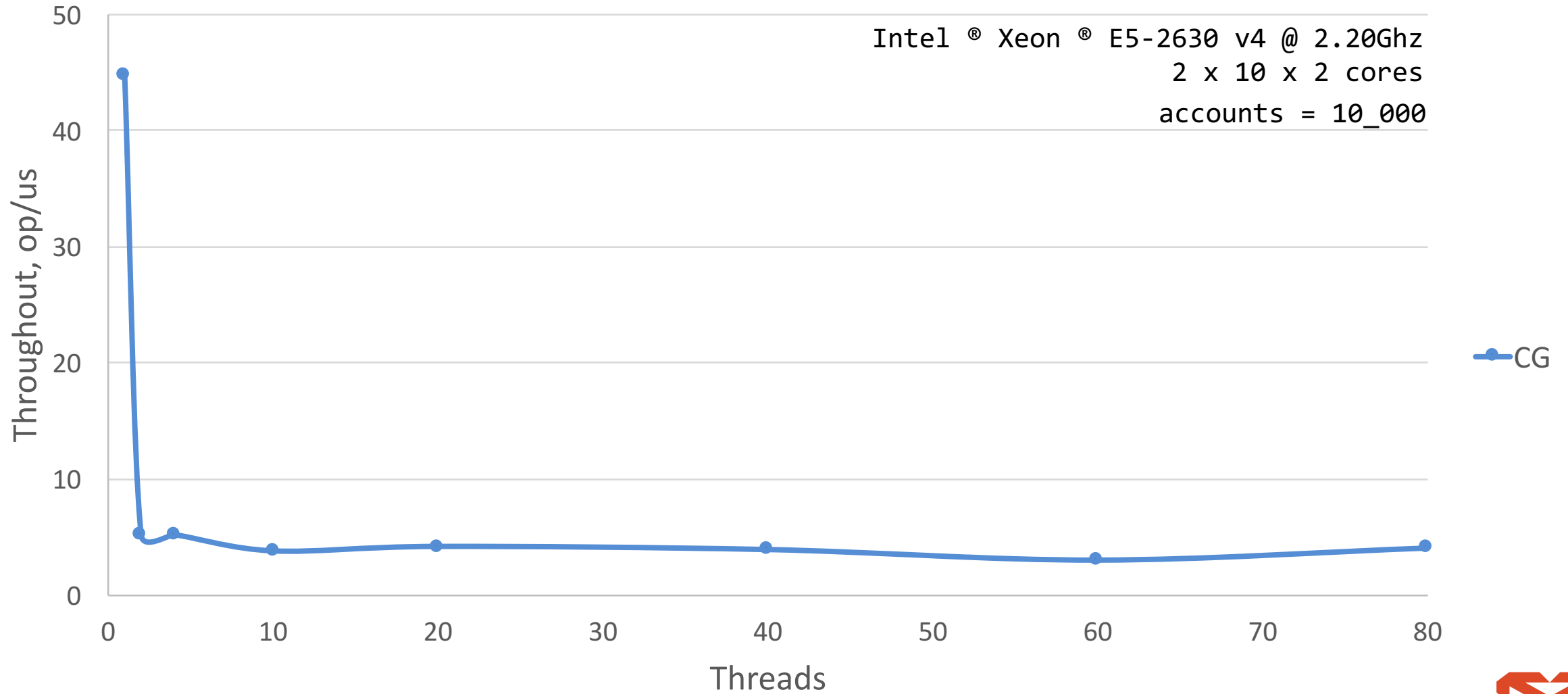
Обеспечили
последовательный доступ

```
fun withdraw(id: Int, amount: Long) = gLock.withLock { ... }
```

```
fun transfer(fromId: Int, toId: Int, amount: Long) = gLock.withLock {  
    accounts[fromId].amount -= amount  
    accounts[toId].amount += amount  
}
```

Грубая блокировка

Intel® Xeon® E5-2630 v4 @ 2.20Ghz
2 x 10 x 2 cores
accounts = 10_000



Грубая блокировка

- Можно сделать любой объект потокобезопасным
- Простой код

Грубая блокировка

- Можно сделать любой объект потокобезопасным
- Простой код
- Плохо масштабируется
 - Исполнение де-факто последовательное
 - Вспоминаем закон Амдала

Подходы к построению алгоритмов

- Грубая блокировка
- **Тонкая блокировка**
- Неблокирующая синхронизация

Тонкая блокировка

- Fine-Grained locking
- Блокируем не весь объект, а только необходимые части
 - Будем блокировать аккаунты по отдельности

Тонкая блокировка

- Fine-Grained locking
- Блокируем не весь объект, а только необходимые части
 - Будем блокировать аккаунты по отдельности

```
private class Account(  
    var amount: Long = 0,  
    val lock: Lock = ReentrantLock()  
)
```

Тонкая блокировка

```
fun getAmount(id: Int): Long {  
    val account = accounts[id]  
    account.lock.withLock {  
        return account.amount  
    }  
}  
  
fun deposit(id: Int, amount: Long) {  
    val account = accounts[id]  
    account.lock.withLock {  
        account.amount += amount  
    }  
}
```

Безопасный доступ
к аккаунтам

Тонкая блокировка

```
fun transfer(fromId: Int, toId: Int, amount: Long) {  
    val from = accounts[fromId]  
    val to = accounts[toId]  
    from.lock.lock()  
    to.lock.lock()  
    try {  
        from.amount -= amount  
        to.amount += amount  
    } finally {  
        from.lock.unlock()  
        to.lock.unlock()  
    }  
}
```

Тонкая блокировка

```
fun transfer(fromId: Int, toId: Int, amount: Long) {  
    val from = accounts[fromId]  
    val to = accounts[toId]  
    from.lock.lock()  
    to.lock.lock()  
    try {  
        from.amount -= amount  
        to.amount += amount  
    } finally {  
        from.lock.unlock()  
        to.lock.unlock()  
    }  
}
```

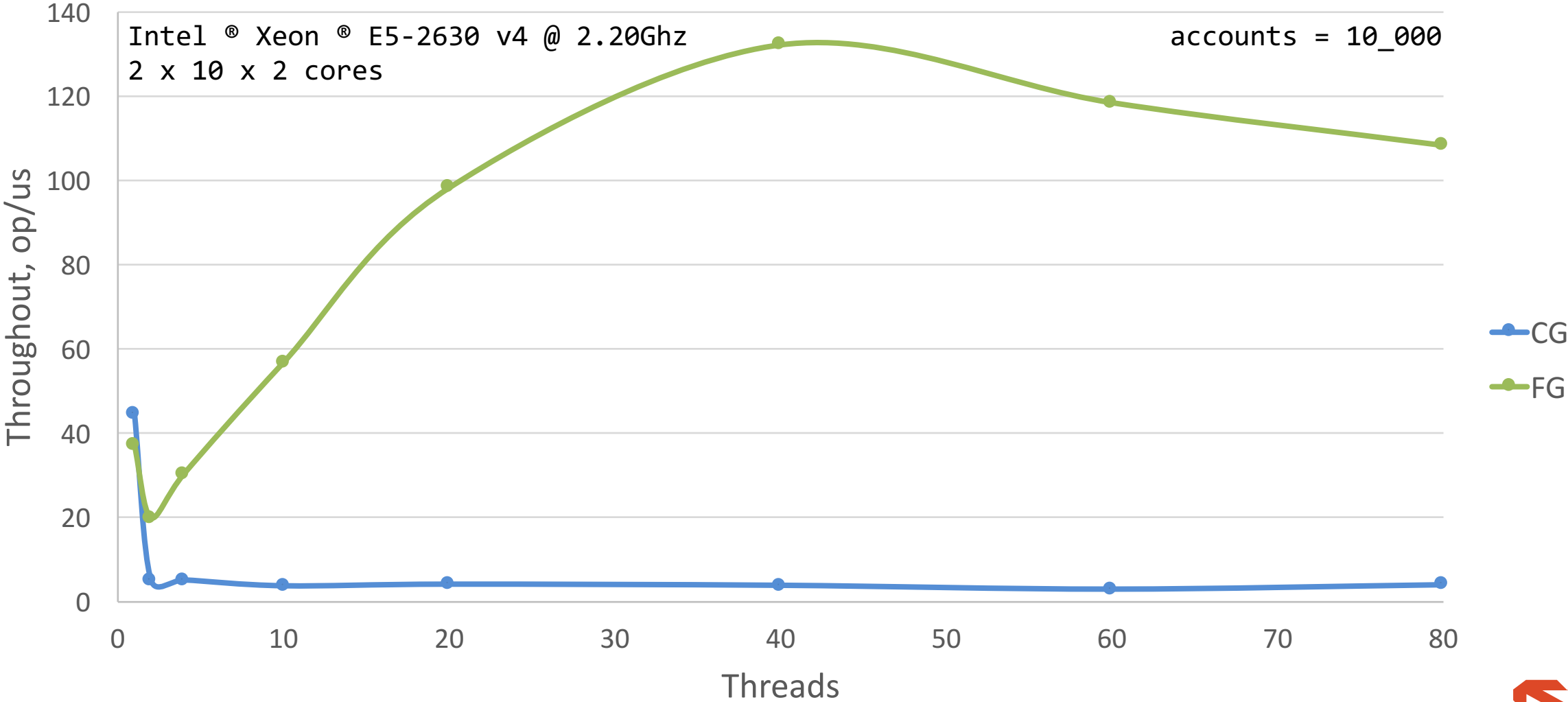
Легко попасть в
дедлок!

Тонкая блокировка

```
fun transfer(fromId: Int, toId: Int, amount: Long) {  
    val from = accounts[fromId]  
    val to = accounts[toId]  
    accounts[if (fromId < toId) fromId else toId].lock.lock()  
    accounts[if (fromId < toId) toId else fromId].lock.lock()  
    try {  
        from.amount -= amount  
        to.amount += amount  
    } finally {  
        from.lock.unlock()  
        to.lock.unlock()  
    }  
}
```

Теперь всегда берём
блокировки в одном
порядке!

Тонкая блокировка



Тонкая блокировка

- Относительно просто увеличивает масштабируемость

Тонкая блокировка

- Относительно просто увеличивает масштабируемость
- Сложно взять блокировки «правильно»
 - Получив безопасный объект и не попав в дедлок
<https://github.com/Devexperts/dlcheck>
- Необходимо открыть протокол блокировки
- Ограничена масштабируемость

Подходы к построению алгоритмов

- Грубая блокировка
- Тонкая блокировка
- Неблокирующая синхронизация

Неблокирующая синхронизация

- Базовый кирпичик: Compare-And-Set
 - `AtomicInteger`, `AtomicReference<T>`, ...

```
Compare-And-Set(reg, expected, updated) = atomic {  
    if (reg.value == expected):  
        reg.value = updated; return true;  
    else:  
        return false;  
}
```

Неблокирующая синхронизация

```
class LFBank(n: Int) : Bank {  
    private val accounts = AtomicReferenceArray(n, { Account() })
```



Новый Account на
каждое изменение

Неблокирующая синхронизация

```
class LFBank(n: Int) : Bank {  
    private val accounts = AtomicReferenceArray(n, { Account() })  
  
    ...  
  
    fun deposit(id: Int, amount: Long) {  
        while (true) {  
            val account = accounts[id]  
            val updated = Account(account.amount + amount)  
            if (accounts.compareAndSet(id, account, updated))  
                return  
        }  
    }  
}
```

CAS-loop



Неблокирующая синхронизация

- Чтение не требует дополнительной синхронизации

```
fun getAmount(id: Int): Long {  
    val account = accounts.get(id)  
    return account.amount  
}
```

Неблокирующая синхронизация

- С transfer-ом всё сложнее

```
fun transfer(fromId: Int, toId: Int, amount: Long) {  
    accounts[fromId].amount -= amount  
    accounts[toId].amount += amount  
}
```

Неблокирующая синхронизация

- С transfer-ом всё сложнее

```
fun transfer(fromId: Int, toId: Int, amount: Long) {  
    accounts[fromId].amount -= amount  
    accounts[toId].amount += amount  
}
```

Как поменять два аккаунта атомарно?

Неблокирующая синхронизация

- Кирпичик №2: дескрипторы [1]

[1] T. Harris et al. "A practical multi-word compare-and-swap operation." ISDC'02

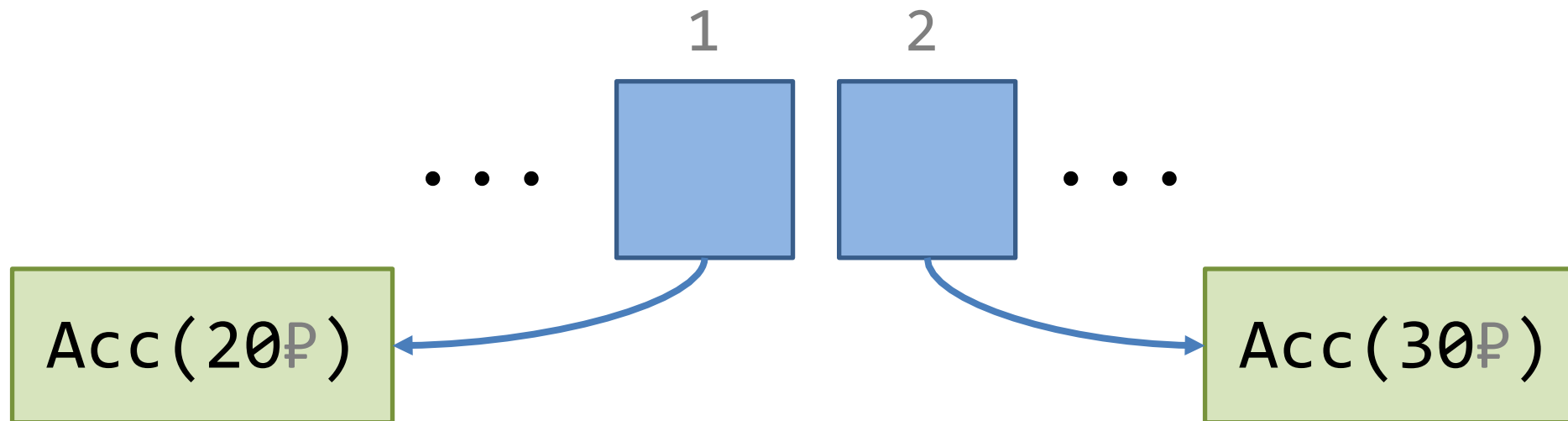
Неблокирующая синхронизация

- Кирпичик №2: дескрипторы [1]
 1. Создаём дескриптор операции (transfer)
 2. Заменяем аккаунты на дескрипторы
 3. Выполняем операцию
 4. Заменяем дескрипторы на новые значения

[1] T. Harris et al. "A practical multi-word compare-and-swap operation." ISDC'02

Неблокирующая синхронизация

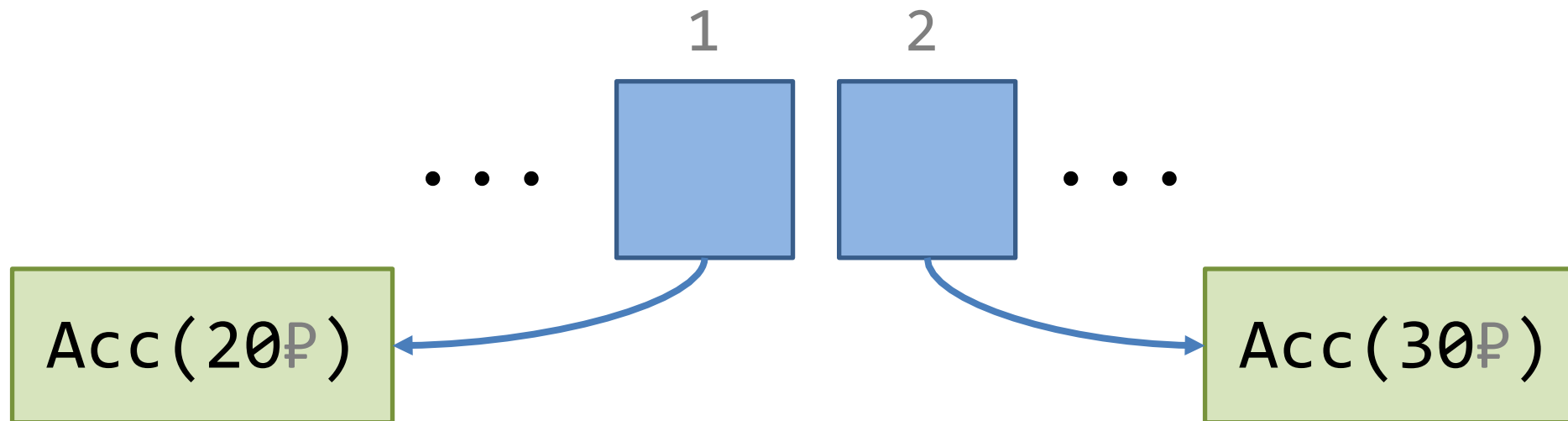
`transfer(1, 2, 10₽):`



[1] T. Harris et al. "A practical multi-word compare-and-swap operation." ISDC'02

Неблокирующая синхронизация

`transfer(1, 2, 10₽):`

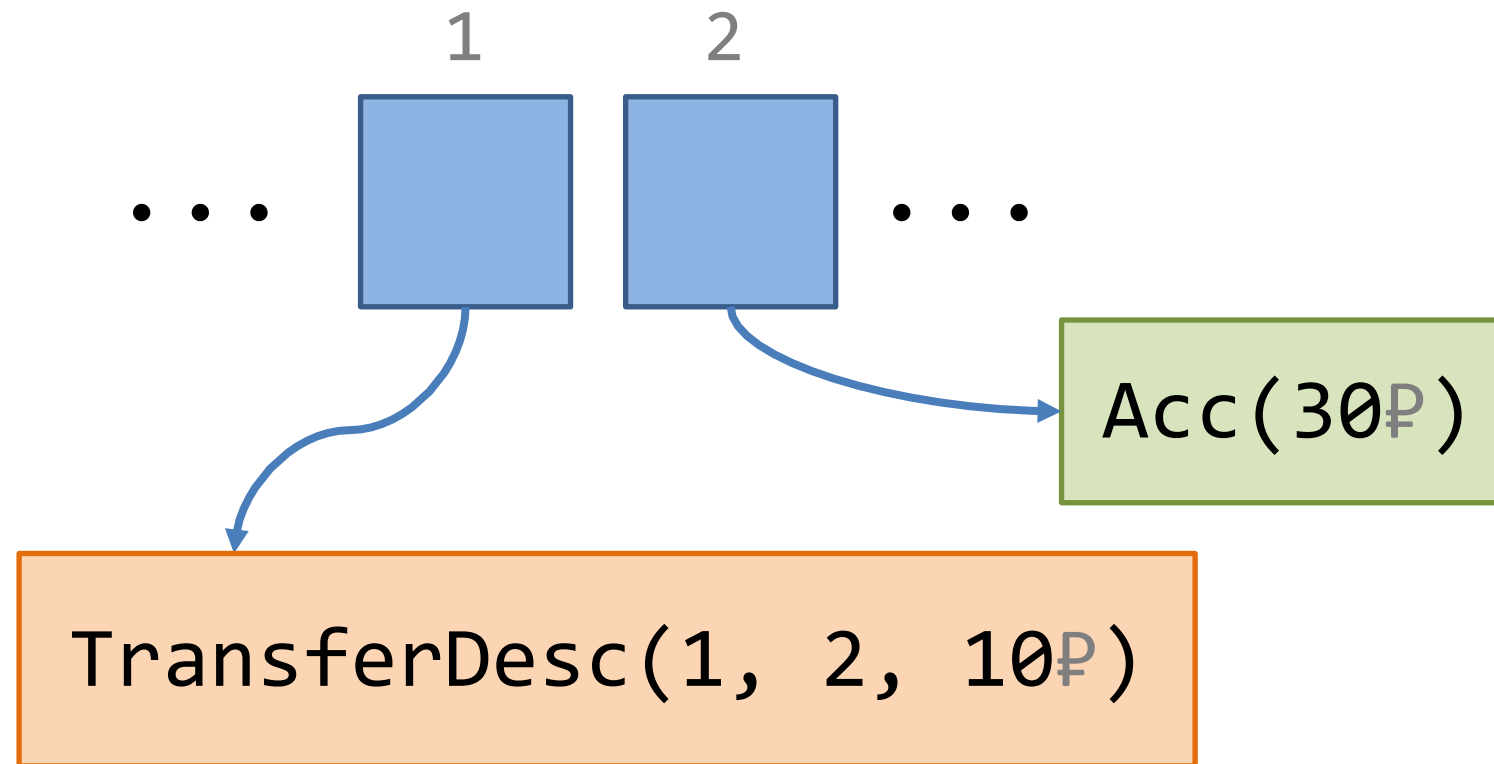


`TransferDesc(1, 2, 10₽)`

[1] T. Harris et al. "A practical multi-word compare-and-swap operation." ISDC'02

Неблокирующая синхронизация

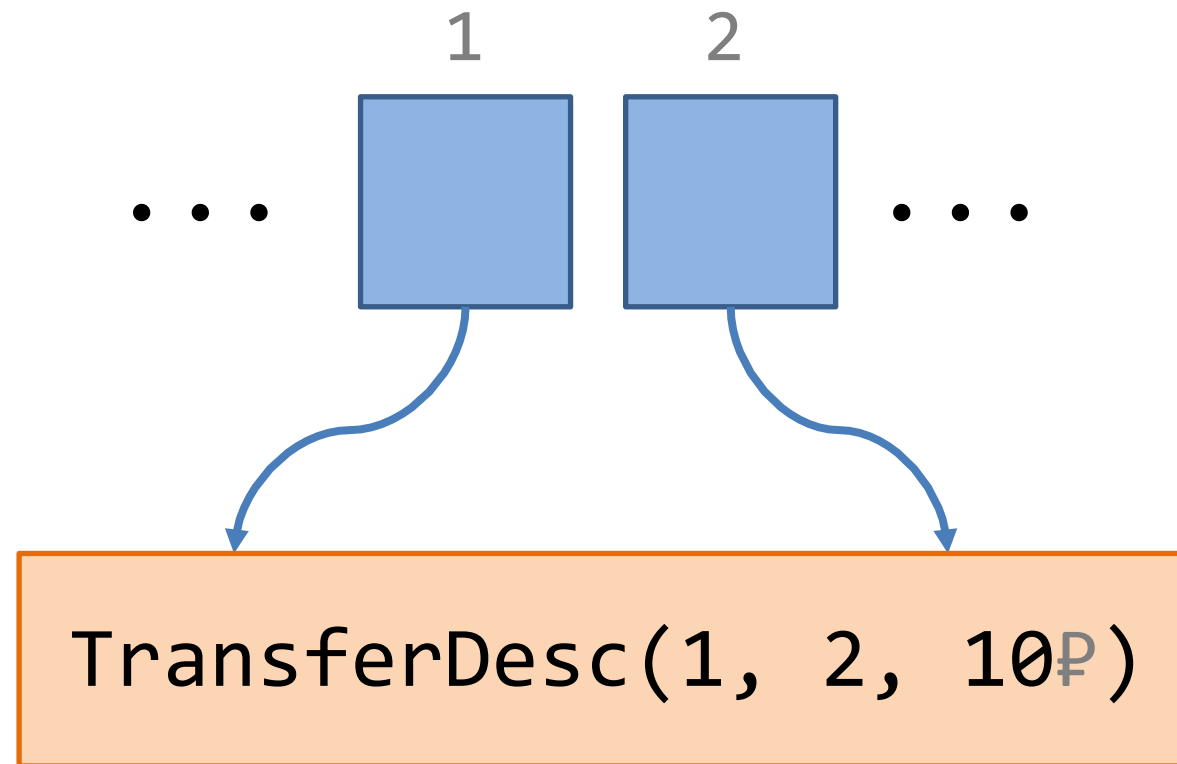
`transfer(1, 2, 10₽):`



[1] T. Harris et al. "A practical multi-word compare-and-swap operation." ISDC'02

Неблокирующая синхронизация

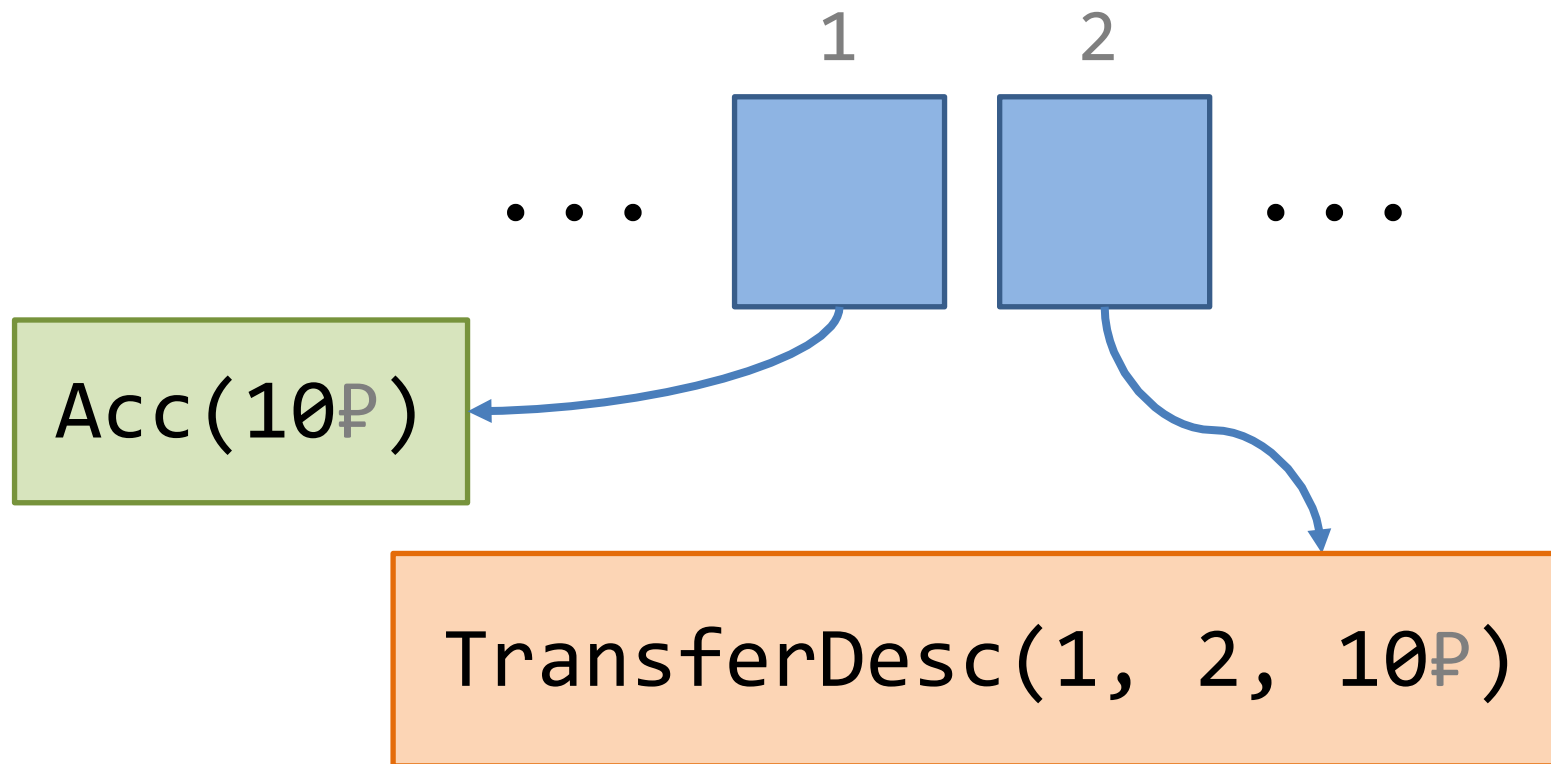
`transfer(1, 2, 10P)`:



[1] T. Harris et al. "A practical multi-word compare-and-swap operation." ISDC'02

Неблокирующая синхронизация

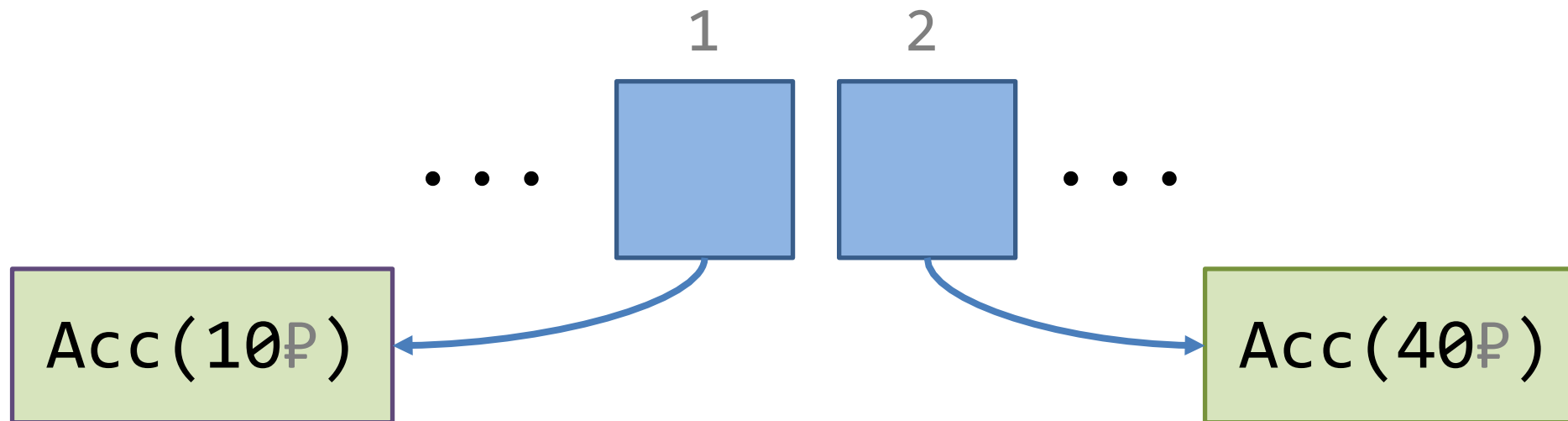
`transfer(1, 2, 10P)`:



[1] T. Harris et al. "A practical multi-word compare-and-swap operation." ISDC'02

Неблокирующая синхронизация

`transfer(1, 2, 10₽):`



[1] T. Harris et al. "A practical multi-word compare-and-swap operation." ISDC'02

Неблокирующая синхронизация

```
private open class Account(val amount: Long)

private class AcquiredAccount(amount: Long, val op: Op,
    var newAmount: Long = amount) : Account(amount)

private abstract class Op {
    @Volatile var completed: Boolean = false
    abstract fun invoke()
}

private fun acquire(id: Int, op: Op): AcquiredAccount? {
    while (true) {
        if (op.completed)
            return null
        val account = accounts[id]
        if (account is AcquiredAccount) {
            if (account.op === op)
                return account
            account.op.invoke()
        } else {
            val acquiredAccount = AcquiredAccount(account.amount, op)
            if (accounts.compareAndSet(id, account, acquiredAccount)) {
                return acquiredAccount
            }
        }
    }
}
```

```
private fun release(id: Int, op: Op) {
    val account = accounts.get(id)
    if (account is AcquiredAccount && account.op === op) {
        val updated = Account(account.newAmount)
        accounts.compareAndSet(id, account, updated)
    }
}

private inner class TransferOp(val fromIndex: Int, val toIndex: Int,
    val amount: Long) : Op() {

    override fun invoke() {
        val from: AcquiredAccount?
        val to: AcquiredAccount?
        if (fromIndex < toIndex) {
            from = acquire(fromIndex, this)
            to = acquire(toIndex, this)
        } else {
            to = acquire(toIndex, this)
            from = acquire(fromIndex, this)
        }
        if (to != null && from != null) {
            from.newAmount = from.amount - amount
            to.newAmount = to.amount + amount
            completed = true
        }
        release(fromIndex, this)
        release(toIndex, this)
    }
}
```


Неблокирующая синхронизация

```
private open class Account(val amount: Long)

private class AcquiredAccount(amount: Long, val op: Op,
    var newAmount: Long = amount) : Account(amount)

private abstract class Op {
    @Volatile var completed: Boolean = false
    abstract fun invoke()
}

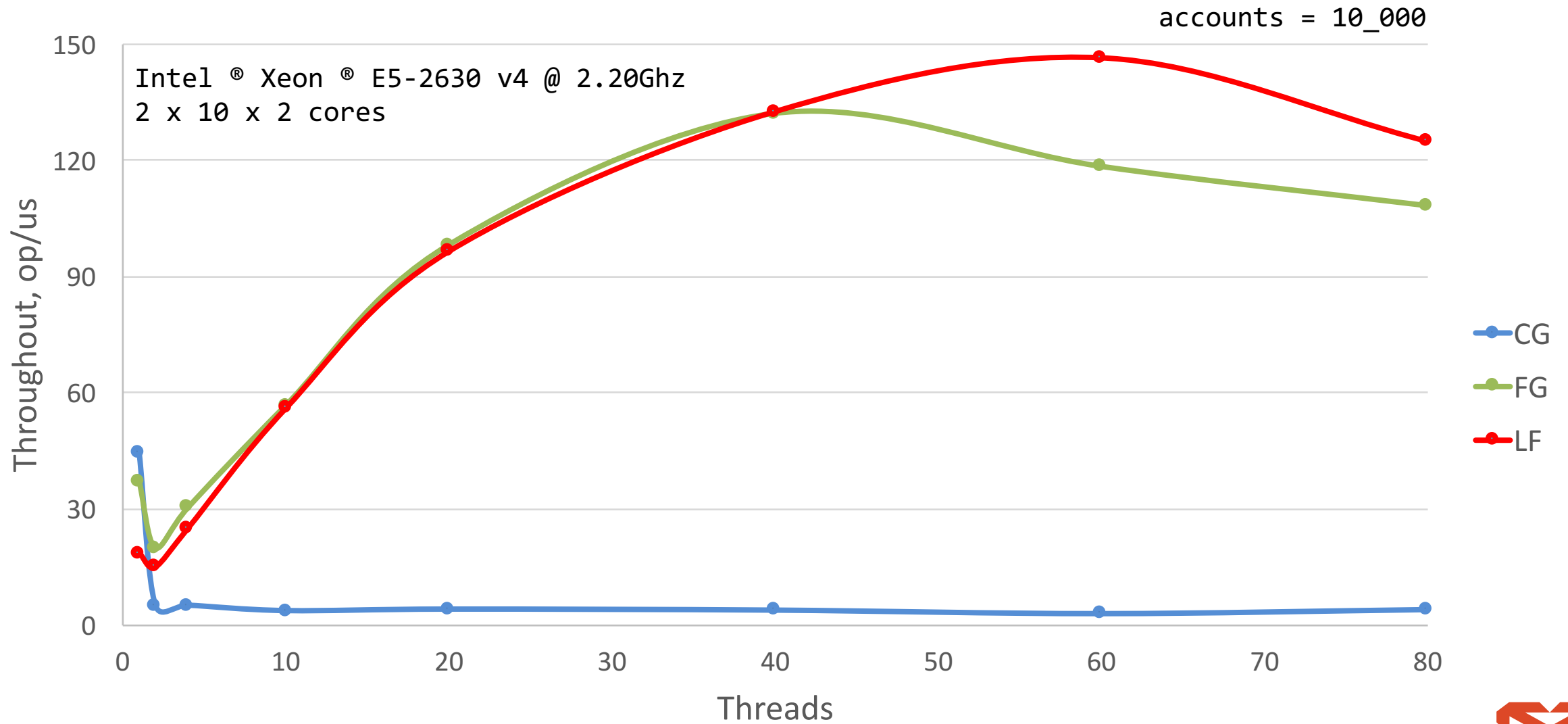
private fun acquire(id: Int, op: Op): AcquiredAccount? {
    while (true) {
        if (op.completed)
            return null
        val account = accounts[id]
        if (account is AcquiredAccount) {
            if (account.op === op)
                return account
            account.op.invoke()
        } else {
            val acquiredAccount = AcquiredAccount(account.amount, op)
            if (accounts.compareAndSet(id, account, acquiredAccount)) {
                return acquiredAccount
            }
        }
    }
}
```

```
private fun release(id: Int, op: Op) {
    val account = accounts.get(id)
    if (account is AcquiredAccount && account.op === op) {
        val updated = Account(account.newAmount)
        accounts.compareAndSet(id, account, updated)
    }
}

private inner class TransferOp(val fromIndex: Int, val toIndex: Int,
    val amount: Long) : Op() {
    override fun invoke() {
        val from = AcquiredAccount?
        val to = AcquiredAccount?
        if (fromIndex < toIndex) {
            from = acquire(fromIndex, this)
            to = acquire(toIndex, this)
        } else {
            to = acquire(toIndex, this)
            from = acquire(fromIndex, this)
        }
        if (to != null && from != null) {
            from.newAmount = from.amount - amount
            to.newAmount = to.amount + amount
            completed = true
        }
        release(fromIndex, this)
        release(toIndex, this)
    }
}
```

Стало сложно...

Неблокирующая синхронизация



Неблокирующая синхронизация

- Позволяет хорошо масштабироваться
- Нет дедлоков и инверсии приоритетов

Неблокирующая синхронизация

- Позволяет хорошо масштабироваться
- Нет дедлоков и инверсии приоритетов
- Эффективный алгоритм – часто научный результат
- Легко допустить ошибку в коде
- Сложно тестировать

<https://github.com/Devexperts/lin-check>

Многопоточность – это сложно!

Идеальный мир

- У нас хорошая масштабируемость
- ... при этом простой код
- ... в котором не нужно думать о дедлоках и пр.
- ... с инкапсулированным контрактом

Идеальный мир: транзакции

```
fun transfer(fromId: Int, toId: Int, amount: Long) = atomic {  
    accounts[fromId].amount -= amount  
    accounts[toId].amount += amount  
}
```



Ну не сказка ли!

Почему транзакции?

- Не нужно думать о порядке блокировок
 - Просто пиши **atomic**
- Не нужно думать о тонкой или толстой блокировке
 - Просто пиши **atomic**
- Не нужно изобретать сложные структуры данных
 - Просто пиши **atomic**

Откуда взять atomic

- Software Transactional Memory (STM)
- Hardware Transactional Memory (HTM)
- Hybrid Transactional Memory

Откуда взять atomic

- Software Transactional Memory (STM)
- Hardware Transactional Memory (HTM)
- Hybrid Transactional Memory

Software Transactional Memory

- Обычно на блокировках/дескрипторах
 - «Самая развивающаяся» для JVM: Scala STM
 - «Самая эффективная» (для C/C++): NOrec

Software Transactional Memory

- Обычно на блокировках/дескрипторах
 - «Самая развивающаяся» для JVM: Scala STM
 - «Самая эффективная» (для C/C++): NOrec
- Применяется для «безопасного» программирования
 - Альтернативный подход: корутины (завтра в 12:15)

Откуда взять atomic

- Software Transactional Memory (STM)
- Hardware Transactional Memory (HTM)
- Hybrid Transactional Memory

Hardware Transactional Memory

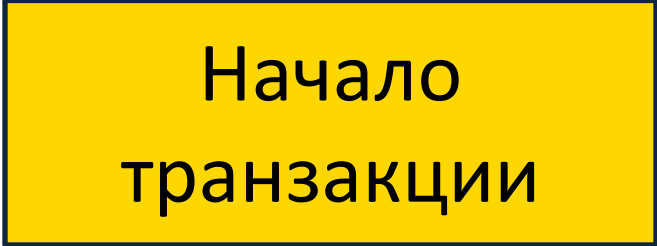
- Почему бы не делать транзакции на уровне железа?
 - Intel Haswell (дорогие модели) и новее
 - IBM Power 8 и новее
 - У AMD только proposal

Hardware Transactional Memory

- Почему бы не делать транзакции на уровне железа?
 - **Intel Haswell (дорогие модели) и новее**
 - IBM Power 8 и новее
 - У AMD только proposal

Intel RTM*

- XBEGIN &on_abort_label
 - Записывает статус в %eax



Начало
транзакции

*<https://software.intel.com/en-us/node/695152>

Intel RTM*

- XBEGIN &on_abort_label
 - Записывает статус в %eax
- XEND



Конец
транзакции

*<https://software.intel.com/en-us/node/695152>

Intel RTM*

- XBEGIN &on_abort_label
 - Записывает статус в %eax
- XEND
- XABORT




Отмена
транзакции

*<https://software.intel.com/en-us/node/695152>

Intel RTM*

- XBEGIN &on_abort_label
 - Записывает статус в %eax
- XEND
- XABORT
- XTEST

Проверка «идёт ли
сейчас транзакция»



*<https://software.intel.com/en-us/node/695152>

Intel RTM: example

Хотим написать
`atomic { /* Transaction body */ }`

Intel RTM: example

```
0x7fe69cd1b377: mov    $0xffffffff, %eax
0x7fe69cd1b37c: xbegin 0x7fe69cd1b382
0x7fe69cd1b382: cmp    $0xffffffff, %eax
0x7fe69cd1b389: jne    0x7fe69cd1b377
```

В %eax будет записан
статус при отмене

Intel RTM: example

```
0x7fe69cd1b377: mov     $0xffffffff, %eax
0x7fe69cd1b37c: xbegin 0x7fe69cd1b382
0x7fe69cd1b382: cmp     $0xffffffff, %eax
0x7fe69cd1b389: jne    0x7fe69cd1b377
```

Куда идти при отмене
транзакции

Intel RTM: example

```
0x7fe69cd1b377: mov    $0xffffffff, %eax
0x7fe69cd1b37c: xbegin 0x7fe69cd1b382
0x7fe69cd1b382: cmp    $0xffffffff, %eax
0x7fe69cd1b389: jne    0x7fe69cd1b377
```

Начинаем сначала, если
транзакция отменилась

Intel RTM: example

```
0x7fe69cd1b377: mov    $0xffffffff, %eax
0x7fe69cd1b37c: xbegin 0x7fe69cd1b382
0x7fe69cd1b382: cmp    $0xffffffff, %eax
0x7fe69cd1b389: jne    0x7fe69cd1b377
```

; Transaction body

```
0x7fe69cd1b3a7: xend
```

Выполняем свой код в
рамках транзакции

Intel RTM: example

```
0x7fe69cd1b377: mov     $0xffffffff, %eax  
0x7fe69cd1b37c: xbegin 0x7fe69cd1b382  
0x7fe69cd1b382: cmp     $0xffffffff, %eax  
0x7fe69cd1b389: jne     0x7fe69cd1b377
```

; Transaction body

```
0x7fe69cd1b3a7: xend
```

Конец транзакции

Intel RTM: example

```
0x7fe69cd1b377: mov    $0xffffffff, %eax  
0x7fe69cd1b37c: xbegin 0x7fe69cd1b382  
0x7fe69cd1b382: cmp    $0xffffffff, %eax  
0x7fe69cd1b389: jne   0x7fe69cd1b377
```

; Transaction body

```
0x7fe69cd1b3a7: xend
```

Придём сюда, если
что-то пойдёт не так

Intel RTM: причины отмены транзакции

- Вызов XABORT

Intel RTM: причины отмены транзакции

- Вызов XABORT
- Другое ядро читает кеш-линию, в которую мы пишем
- Другое ядро пишет в кеш-линию, которую мы читаем

Intel RTM: причины отмены транзакции

- Вызов XABORT
- Другое ядро читает кеш-линию, в которую мы пишем
- Другое ядро пишет в кеш-линию, которую мы читаем
- Слишком много данных
- Слишком много инструкций

Intel RTM: причины отмены транзакции

- Вызов XABORT
- Другое ядро читает кеш-линию, в которую мы пишем
- Другое ядро пишет в кеш-линию, которую мы читаем
- Слишком много данных
- Слишком много инструкций
- ...

Intel RTM: причины отмены транзакции

- Вызов XABORT
- Другое ядро читает кеш-линию, в которую мы пишем
- Другое ядро пишет в кеш-линию, которую мы читаем
- Слишком много операций
- Слишком много инструкций
- ...

По значению в %eax
можно понять причину*

*<https://software.intel.com/en-us/node/695152>

Intel RTM: как использовать

- Хорошо работает для коротких транзакций
- Может не быть прогресса

Intel RTM: как использовать


- Хорошо работает для коротких транзакций
- Может не быть прогресса
- ⇒ нельзя полагаться только на HTM

```
if (xbegin() == XBEGIN_STARTED) {  
    // Simple sequential code  
    xend()  
} else {  
    // Complex synchronization  
}
```

Intel RTM: как использовать

- Хорошо работает для коротких транзакций
- Может не быть прогресса
- ⇒ нельзя полагаться только на HTM

```
if (xbegin() == XBEGIN_STARTED) {  
    // Simple sequential code  
    xend()  
} else {  
    // Complex synchronization  
}
```



Можно делать
несколько попыток

←

Intel RTM + Java

- Lock elision
 - Оптимизация блокировок
- `java.util.concurrent.RTMSupport`
 - Использование RTM примитивов напрямую
 - Experimental

Пока что такого нет!!!

Intel RTM + Java

- Lock elision
 - Оптимизация блокировок
- `java.util.concurrent.RTMSupport`
 - Использование RTM примитивов напрямую
 - Experimental

Lock elision

```
// 0 -- UNLOCKED, 1 -- LOCKED
val state = AtomicInteger(0)

fun lock() {
    while (true) {
        // UNLOCKED → LOCKED
        if (state.compareAndSet(0, 1))
            break
        // Wait & retry
    }
}
```

```
fun unlock() {
    // LOCKED → UNLOCKED
    state.set(0)
}
```

Lock elision

```
// 0 -- UNLOCKED, 1 -- LOCKED
val state = AtomicInteger(0)

fun lock() {
    if (xbegin() == XBEGIN_STARTED) {
        if (state.get() == 0)
            return
        else
            xabort()
    }
    ...
}
```

Гарантирует, что никто
не держит блокировку


Lock elision

```
// 0 -- UNLOCKED, 1 -- LOCKED
val state = AtomicInteger(0)

fun lock() {
    if (xbegin() == XBEGIN_STARTED) {
        if (state.get() == 0)
            return
        else
            xabort()
    }

    ...
}
```

```
fun unlock() {
    if (state.get() == 0) {
        xend()
        return
    }
    state.set(0)
}
```



Завершаем
транзакцию

Lock elision

- [[JDK-8031320](#)] Use Intel RTM instructions for locks (8u25+)
 - Работает только для **synchronized**

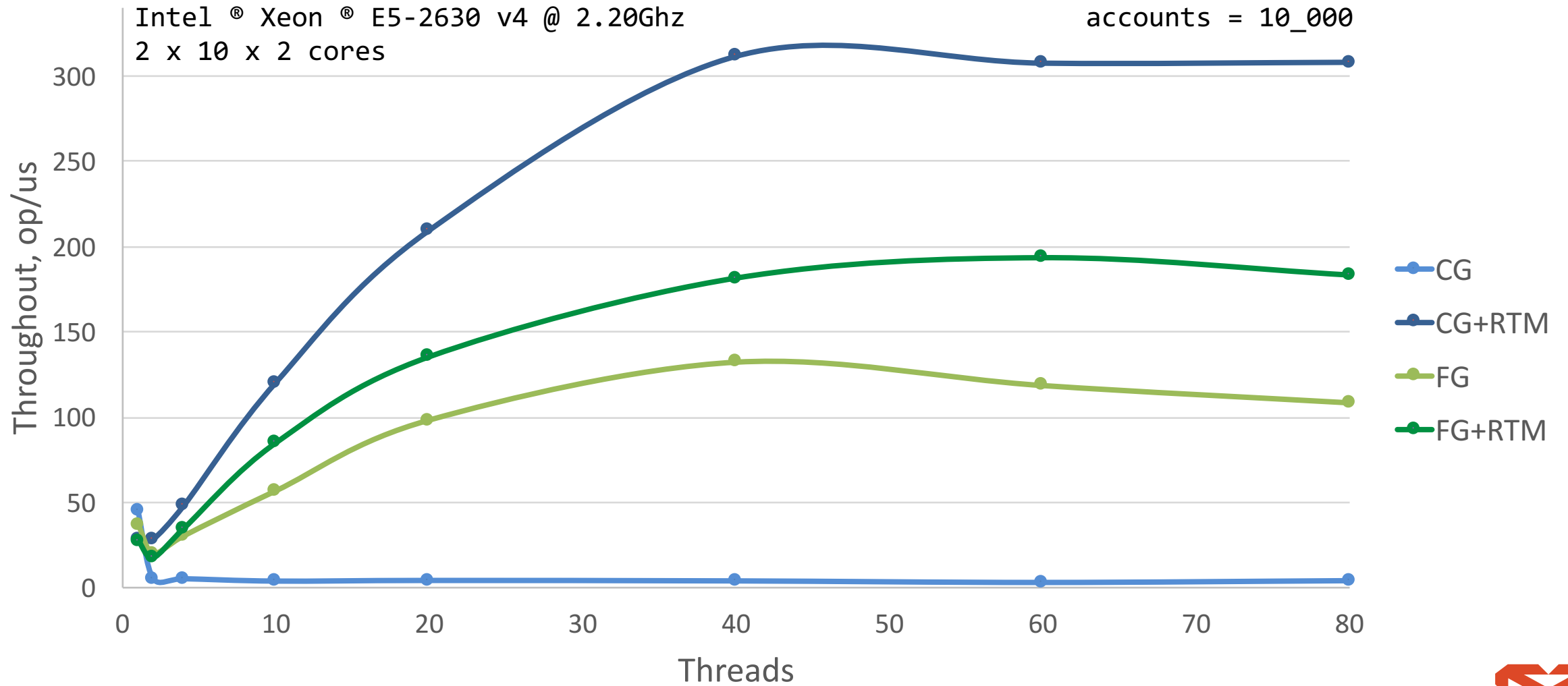
Lock elision

- [[JDK-8031320](#)] Use Intel RTM instructions for locks (8u25+)
 - Работает только для `synchronized`
- Включить: `-XX:+UseRTMLocking`
- Количество попыток: `-XX:RTMRetryCount=retries`

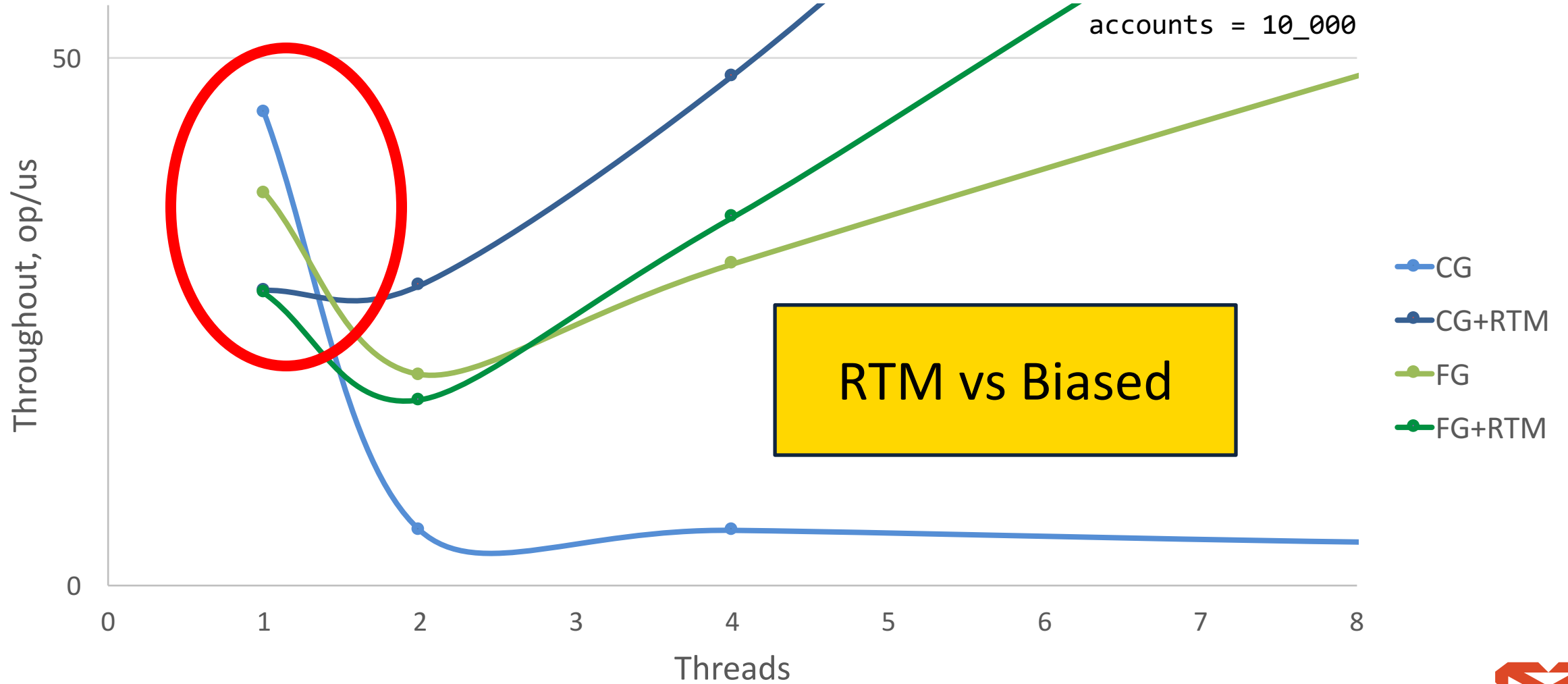
Lock elision

- [[JDK-8031320](#)] Use Intel RTM instructions for locks (8u25+)
 - Работает только для `synchronized`
- Включить: `-XX:+UseRTMLocking`
- Количество попыток: `-XX:RTMRetryCount=retries`
- Деоптимизация: `-XX:+UseRTMDeopt`
- Порог для деоптимизации: `-XX:RTMAbortRatio=ratio`

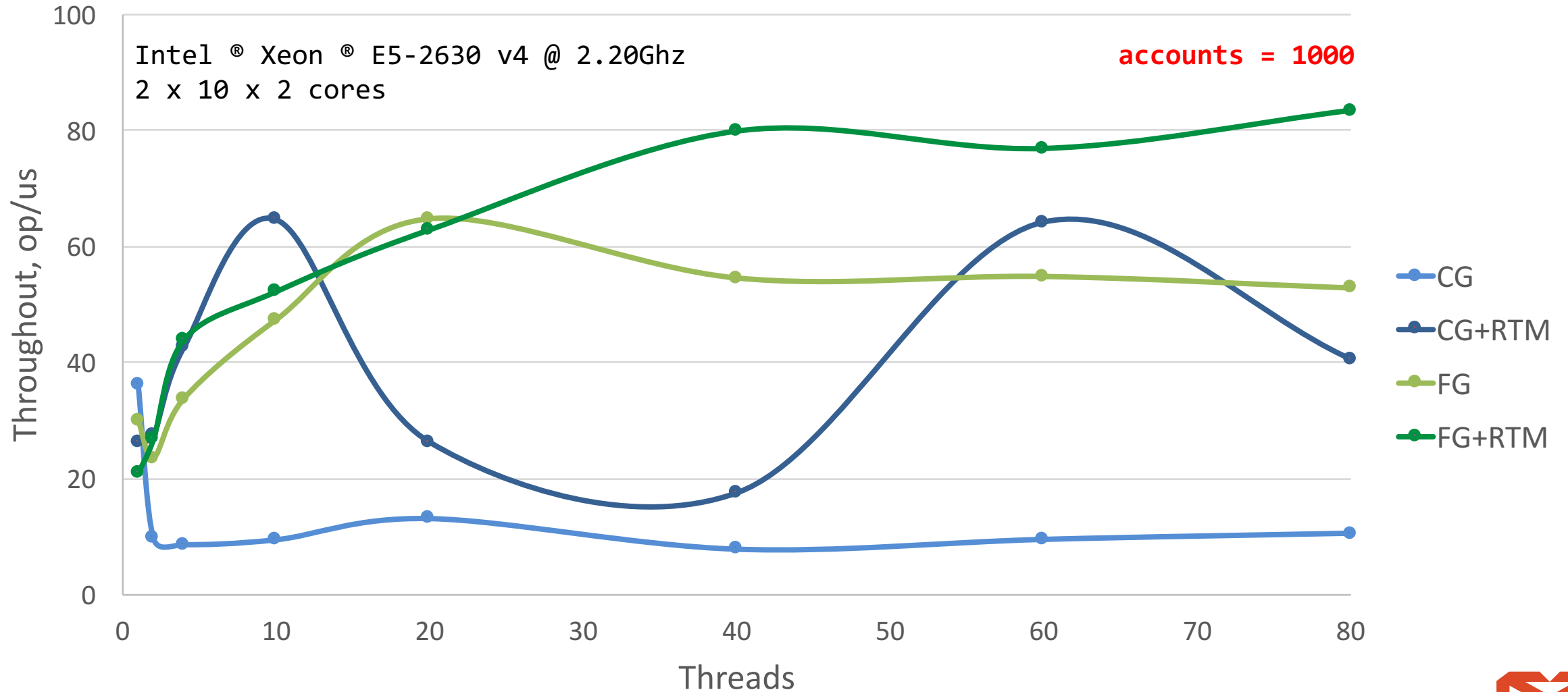
Lock elision



Lock elision



Lock elision: high contention



Intel RTM + Java

- Lock elision
 - Оптимизация блокировок
- `java.util.concurrent.RTMSupport`
 - Использование RTM примитивов напрямую
 - Experimental

Intel RTM: Java API

```
public class RTMSupport {
    public static int RTM_UNSUPPORTED = 0;
    public static int XBEGIN_STARTED = -1;
    public static int XABORT_EXPLICIT = 1 << 0;
    public static int XABORT_RETRY = 1 << 1;
    public static int XABORT_CONFLICT = 1 << 2;
    public static int XABORT_CAPACITY = 1 << 3;
    public static int XABORT_DEBUG = 1 << 4;
    public static int XABORT_NESTED = 1 << 5;

    public static int xbegin() { return RTM_UNSUPPORTED; }
    public static void xend() {}
    public static void xabort() {}
    public static boolean xtest() { return false; }
}
```

Intel RTM: Java API

```
public class RTMSupport {
    public static int RTM_UNSUPPORTED = 0;
    public static int XBEGIN_STARTED = -1;
    public static int XABORT_EXPLICIT = 1 << 0;
    public static int XABORT_RETRY = 1 << 1;
    public static int XABORT_CONFLICT = 1 << 2;
    public static int XABORT_CAPACITY = 1 << 3;
    public static int XABORT_DEBUG = 1 << 4;
    public static int XABORT_NESTED = 1 << 5;

    public static int xbegin() { return RTM_UNSUPPORTED; }
    public static void xend() {}
    public static void xabort() {}
    public static boolean xtest() { return false; }
}
```

Интринсики!

Intel RTM: intrinsics

```
public class RTMSupport {  
    @HotSpotIntrinsicCandidate  
    public static int xbegin() { return RTM_UNSUPPORTED; }  
  
    @HotSpotIntrinsicCandidate  
    public static void xend() {}  
  
    @HotSpotIntrinsicCandidate  
    public static void xabort() {}  
  
    @HotSpotIntrinsicCandidate  
    public static boolean xtest() { return false; }  
}
```

Intel RTM: intrinsics

- Общая настройка
- Interpreter интринсик
- C1 интринсик
- C2 интринсик
- Graal интринсик
 - Сейчас не поддерживается

Отличный доклад от Volker Simonis на JPoint'16

<https://youtu.be/76gyiCteq-I> + <https://github.com/simonis/JBreak2016>

Intel RTM: intrinsics

- Общая настройка
- Interpreter интринсик
- C1 интринсик
- C2 интринсик

Intel RTM: intrinsics

Добавим в `vmSymbols.hpp`:

```
template(java_util_concurrent_RTMSupport, "java/util/concurrent/RTMSupport") \
...
do_intrinsic(_rtm_xbegin, java_util_concurrent_RTMSupport, xbegin_name, "()I", F_S) \
  do_name(xbegin_name, "xbegin") \
do_intrinsic(_rtm_xabort, java_util_concurrent_RTMSupport, xabort_name, "()V", F_S) \
  do_name(xabort_name, "xabort") \
do_intrinsic(_rtm_xend, java_util_concurrent_RTMSupport, xend_name, "()V", F_S) \
  do_name(xend_name, "xend") \
do_intrinsic(_rtm_xtest, java_util_concurrent_RTMSupport, xtest_name, "()Z", F_S) \
  do_name(xtest_name, "xtest") \
```

Intel RTM: intrinsics

Добавим примитивы в `assembler_x86.cpp`:

```
void Assembler::xbegin(Label& abort) { ... }
```



Уже было для
lock elision

Intel RTM: intrinsics

Добавим примитивы в `assembler_x86.cpp`:

```
void Assembler::xbegin(Label& abort) { ... }
```

```
void Assembler::xbegin(Register dst) {  
    movl(rax, -1);  
    Label L_on_abort;  
    xbegin(L_on_abort);  
    bind(L_on_abort);  
    movl(dst, rax);  
}
```

Результат
`RTMSupport.xbegin()`

Intel RTM: intrinsics

Добавим примитивы в `assembler_x86.cpp`:

```
void Assembler::xbegin(Label& abort) { ... }
```

```
void Assembler::xend(
    movl(rax, -1);
    Label L_on_abort;
    xbegin(L_on_abort);
    bind(L_on_abort);
    movl(dst, rax);
}
```

Аналогично с
`xend()`, `xabort()`,
`xtest(Register dst)`

Intel RTM: intrinsics

- Общая настройка
- **Interpreter интринсик**
- C1 интринсик
- C2 интринсик

Intel RTM: interpreter intrinsic

Добавляем идентификаторы в `abstractInterpreter.hpp`:

```
class AbstractInterpreter: AllStatic {  
    ...  
    enum MethodKind {  
        ...  
        java_util_concurrent_RTMSupport_xbegin,  
        java_util_concurrent_RTMSupport_xend,  
        java_util_concurrent_RTMSupport_xabort,  
        java_util_concurrent_RTMSupport_xtest  
    }  
}
```

Intel RTM: interpreter intrinsic

Сопоставляем идентификаторы с интринсиками в `AbstractInterpreter.cpp`:

```
MethodKind AbstractInterpreter::method_kind(const methodHandle& m) {  
    if (VM_Version::supports_rtm()) {  
        switch (m->intrinsic_id()) {  
            case vmIntrinsics::_rtm_xbegin : return java_util_concurrent_RTMSupport_xbegin;  
            case vmIntrinsics::_rtm_xend   : return java_util_concurrent_RTMSupport_xend  ;  
            case vmIntrinsics::_rtm_xabort : return java_util_concurrent_RTMSupport_xabort;  
            case vmIntrinsics::_rtm_xtest  : return java_util_concurrent_RTMSupport_xtest  ;  
        }  
    }  
    ...  
}
```

Intel RTM: interpreter intrinsic

Вызываем генерацию своей реализации в `templateInterpreterGenerator.cpp`:

```
address TemplateInterpreterGenerator::generate_method_entry(MethodKind kind) {
    switch(kind) {
        case Interpreter::java_util_concurrent_RTMSupport_xbegin
            : entry_point = generate_RTMSupport_xbegin();    break;
        case Interpreter::java_util_concurrent_RTMSupport_xend
            : entry_point = generate_RTMSupport_xend();      break;
        case Interpreter::java_util_concurrent_RTMSupport_xabort
            : entry_point = generate_RTMSupport_xabort();    break;
        case Interpreter::java_util_concurrent_RTMSupport_xtest
            : entry_point = generate_RTMSupport_xtest();     break;
    }
}
```

Intel RTM: interpreter intrinsic

Добавляем реализацию в

templateInterpreterGenerator_x86_64.cpp:

```
address TemplateInterpreterGenerator::generate_RTMSupport_xbegin() {  
    address entry = __ pc();  
    __ xbegin(rax);  
    __ pop(rdi);  
    __ mov(rsp, r13);  
    __ jmp(rdi);  
    return entry;  
}
```

Intel RTM: interpreter intrinsic

Добавляем реализацию в

templateInterpreterGenerator_x86_64.cpp:

```
address TemplateInterpreterGenerator::generate_RTMSupport_xbegin() {  
    address entry = __ pc();  
    __ xbegin(rax);  
    __ pop(rdi);  
    __ mov(rsp, r13);  
    __ jmp(rdi);  
    return entry;  
}
```



Наш xbegin()

Intel RTM: interpreter intrinsic

Добавляем реализацию в

templateInterpreterGenerator_x86_64.cpp:

```
address TemplateInterpreterGenerator::generate_RTMSupport_xbegin() {  
    address entry = __ pc();  
    __ xbegin(rax);  
    __ pop(rdi);  
    __ mov(rsp, r13); ← Генерируем return  
    __ jmp(rdi);  
    return entry;  
}
```

Intel RTM: interpreter intrinsic

Добавляем реализацию в

templateInterpreterGenerator_x86_64.cpp:

```
address TemplateInterpreterGenerator::Support_xbegin() {  
    address entry = ...  
    __ xbegin(rax)  
    __ pop(rdi);  
    __ mov(rsp, r15);  
    __ jmp(rdi);  
    return entry;  
}
```

Аналогично с
xend(), xabort(), xtest()

Intel RTM: interpreter intrinsic

Добавляем реализацию в

templateInterpreterGenerator_x86_64.cpp:

```
address TemplateInterpreterGenerator::generate_RTMSupport_xbegin() {  
    address entry = __ pc();  
    __ xbegin(rax);  
    __ pop(rdi);  
    __ mov(rsp, r13);  
    __ jmp(rdi);  
    return entry;  
}
```



Текущий program
counter

Intel RTM: interpreter intrinsic

Применяем интринсики в

templateInterpreterGenerator.cpp:

```
method_entry(java_lang_math_fmaD )  
method_entry(java_util_concurrent_RTMSupport_xbegin )  
method_entry(java_util_concurrent_RTMSupport_xend )  
method_entry(java_util_concurrent_RTMSupport_xabort )  
method_entry(java_util_concurrent_RTMSupport_xtest )  
method_entry(java_lang_ref_reference_get)
```

Intel RTM: intrinsics

- Общая настройка
- Interpreter интринсик
- C1 интринсик*
- C2 интринсик*

* Объяснение в дополнительных слайдах

Coarse-grained + RTMSupport

```
var attempt = 0
while (attempt < MAX_RTM_ATTEMPTS) {
    if (xbegin() == XBEGIN_STARTED) {
        if (gLock.isLocked()) xabort()
        transferImpl(idFrom, idTo, amount)
        xend()
        return
    } else attempt++
}
gLock.withLock { transferImpl(idFrom, idTo, amount) }
```

Несколько попыток

Coarse-grained + RTMSupport

```
var attempt = 0
while (attempt < MAX_RTM_ATTEMPTS) {
    if (xbegin() == XBEGIN_STARTED) {
        if (gLock.isLocked()) xabort()
        transferImpl(idFrom, idTo, amount)
        xend()
        return
    } else attempt++
}
gLock.withLock { transferImpl(idFrom, idTo, amount) }
```

Увеличиваем счётчик,
если не удалось

Coarse-grained + RTMSupport

```
var attempt = 0
while (attempt < MAX_RTM_ATTEMPTS) {
    if (xbegin() == XBEGIN_STARTED) {
        if (gLock.isLocked()) xabort()
        transferImpl(idFrom, idTo, amount)
        xend()
        return
    } else attempt++
}
gLock.withLock { transferImpl(idFrom, idTo, amount) }
```

1. Кто-то держит
блокировку?

Coarse-grained + RTMSupport

```
var attempt = 0
while (attempt < MAX_RTM_ATTEMPTS) {
    if (xbegin() == XBEGIN_STARTED) {
        if (gLock.isLocked()) xabort()
        transferImpl(idFrom, idTo, amount)
        xend()
        return
    } else attempt++
}
gLock.withLock { transferImpl(idFrom, idTo, amount) }
```

2. Делаем transfer

Coarse-grained + RTMSupport

```
var attempt = 0
while (attempt < MAX_RTM_ATTEMPTS) {
    if (xbegin() == XBEGIN_STARTED) {
        if (gLock.isLocked()) xabort()
        transferImpl(idFrom, idTo, amount)
        xend()
        return
    } else attempt++
}
gLock.withLock { transferImpl(idFrom, idTo, amount) }
```

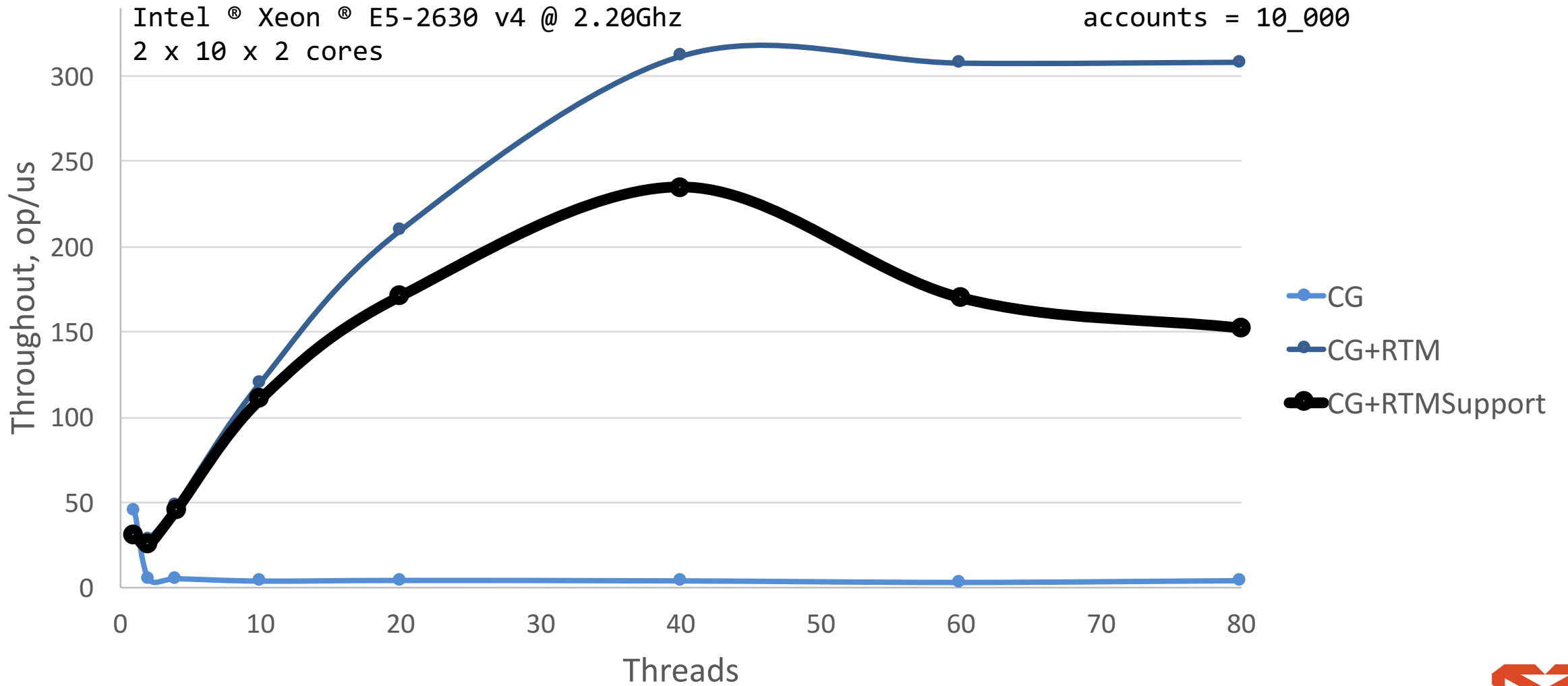
3. Завершаем
транзакцию

Coarse-grained + RTMSupport

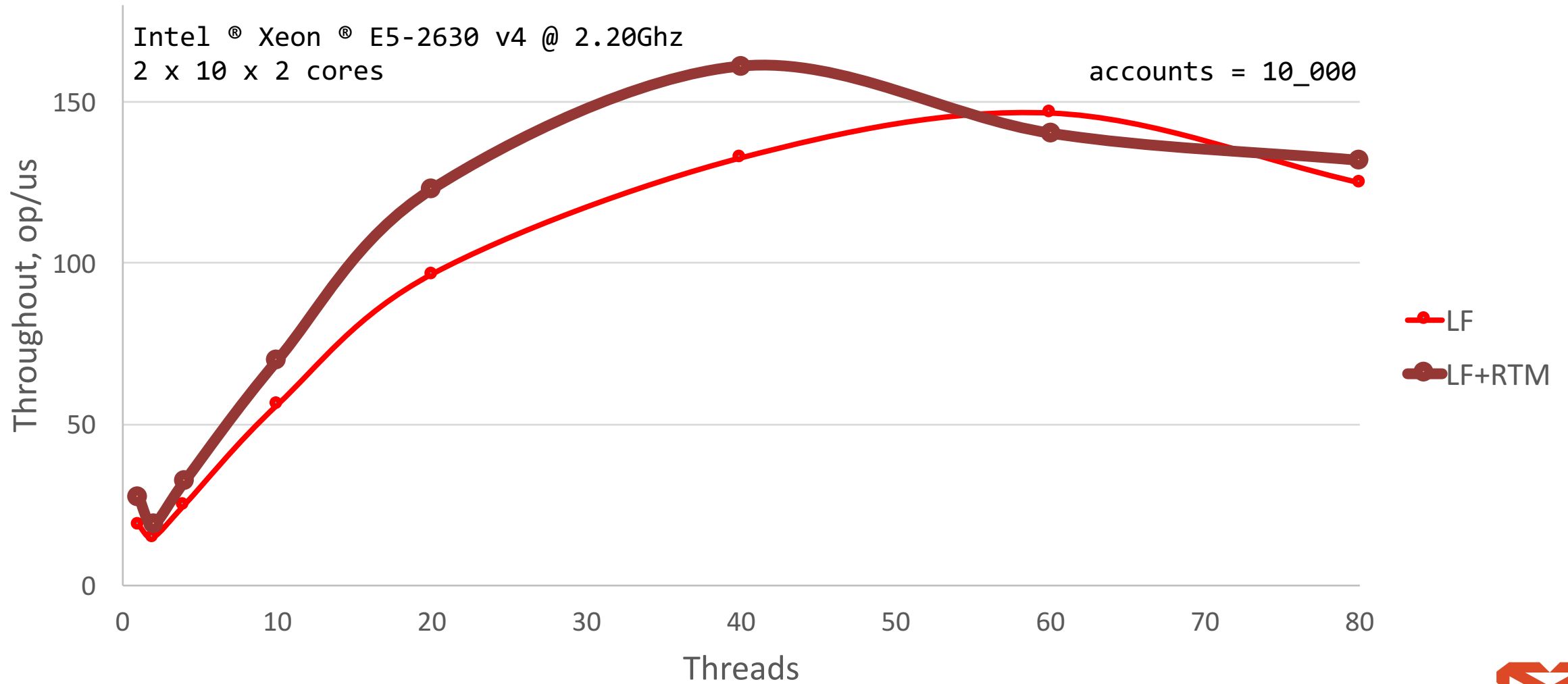
```
var attempt = 0
while (attempt < MAX_RTM_ATTEMPTS) {
    if (xbegin() == XBEGIN_STARTED) {
        if (gLock.isLocked()) xabort()
        transferImpl(idFrom, idTo, amount)
        xend()
        return
    } else attempt++
}
gLock.withLock { transferImpl(idFrom, idTo, amount) }
```

Fallback реализация

Coarse-grained + RTMSupport



Lock-free + RTMSupport



Откуда взять atomic

- Software Transactional Memory (STM)
- Hardware Transactional Memory (HTM)
- Hybrid Transactional Memory
 - RH-NOrec, HY-NOrec
 - TL2

Итого

- Транзакционная память позволяет увеличивать производительность «нахаляву»
 - Lock elision оптимизация
 - `java.util.concurrent.RTMSupport`
- Материалы (патч для OpenJDK + бенчмарки)
 - <https://github.com/ndkoval/jpoint2018>
- Hybrid TM – наше светлое будущее!

Спасибо за внимание!

Никита Коваль

ndkoval *at* ya *dot* ru
twitter.com/nkoval_



Закон Амдала

$$S = \frac{\text{время на 1 ядре}}{\text{время на } N \text{ ядрах}}$$

S – ускорение кода

Закон Амдала

$$S = \frac{1}{1 - P + P/N}$$

S – ускорение кода

P – доля параллельного кода

N – количество ядер

Закон Амдала

- 10 ядер
- **60%** параллельного кода
- **40%** последовательного кода

Закон Амдала

- 10 ядер
- **60%** параллельного кода
- **40%** последовательного кода

$$S = \frac{1}{0.4 + 0.6/10} \approx 2.17$$

Закон Амдала

- 10 ядер
- **80%** параллельного кода
- **20%** последовательного кода

Закон Амдала

- 10 ядер
- **80%** параллельного кода
- **20%** последовательного кода

$$S = \frac{1}{0.2 + 0.8/10} \approx 3.57$$

Закон Амдала

- 10 ядер
- **90%** параллельного кода
- **10%** последовательного кода

Закон Амдала

- 10 ядер
- **90%** параллельного кода
- **10%** последовательного кода

$$S = \frac{1}{0.1 + 0.9/10} \approx 5.26$$

Intel RTM: C1 intrinsics

Добавим идентификаторы в `c1_LIR.cpp`:

```
enum LIR_Code {  
    ...  
    , lir_on_spin_wait  
    , lir_rtm_xbegin  
    , lir_rtm_xend  
    , lir_rtm_xabort  
    , lir_rtm_xtest  
}
```

Intel RTM: C1 intrinsics

Сопоставляем идентификаторы с интринсиками в `c1_LIRGenerator.cpp` и `c1_LIR.hpp`:

```
void LIRGenerator::do_Intrinsic(Intrinsic* x) {  
    switch (x->id()) {  
        case vmIntrinsics::_rtm_xbegin:  
            __ rtm_xbegin(rlock_result(x));    break;  
        case vmIntrinsics::_rtm_xend:  
            __ rtm_xend();                    break;  
        case vmIntrinsics::_rtm_xabort:  
            __ rtm_xabort();                  break;  
        case vmIntrinsics::_rtm_xtest:  
            __ rtm_xtest(rlock_result(x));    break;  
    }
```

Intel RTM: C1 intrinsics

Сопоставляем идентификаторы с интринсиками в `c1_LIRGenerator.cpp` и `c1_LIR.hpp`:

```
void rtm_xbegin(LIR_Opr result) {  
    append(new LIR_Op0(Lir_rtm_xbegin, result)); }
```

```
void rtm_xend() { append(new LIR_Op0(Lir_rtm_xend)); }
```

```
void rtm_xabort() { append(new LIR_Op0(Lir_rtm_xabort)); }
```

```
void rtm_xtest(LIR_Opr result) {  
    append(new LIR_Op0(Lir_rtm_xtest, result)); }
```


Intel RTM: C1 intrinsics

Вызываем генерацию своей реализации в `c1_LIRAssembler.cpp`:

```
void LIR_Assembler::emit_op1(LIR_Op1* op) {  
    switch (op->code()) {  
        ...  
        case Lir_rtm_xbegin: rtm_xbegin(op->result_opr());    break;  
        case Lir_rtm_xend:   rtm_xend();                      break;  
        case Lir_rtm_xabort: rtm_xabort();                   break;  
        case Lir_rtm_xtest:  rtm_xtest(op->result_opr());    break;  
    }  
}
```

Intel RTM: C1 intrinsics

Добавляем реализацию в `c1_LIRAssembler_x86.cpp`:

```
void LIR_Assembler::rtm_xbegin(LIR_Opr result_reg) {  
    __ xbegin(result_reg->as_register());  
}
```

```
void LIR_Assembler::rtm_xend() {  
    __ xend();  
}
```

Результат
`RTMSupport.xbegin()`



Intel RTM: C1 intrinsics

Добавляем реализацию в `c1_LIRAssembler_x86.cpp`:

```
void LIR_Assembler::rtm_xbegin(LIR_Opr result_reg) {  
    __ xbegin(r  
}  
  
void LIR_Asse  
    __ xend();  
}
```

Аналогично с `rtm_xabort()` и
`rtm_xtest(LIR_Opr result)`

Intel RTM: C2 intrinsics

Добавим compiler nodes в `intrinsicnode.hpp`:

```
class XBeginNode: public Node {
public:
    XBeginNode(Node* control): Node(control) {};
    virtual int Opcode() const;
    virtual const Type* bottom_type() const {
        return TypeInt::INT;
    }
    virtual uint ideal_reg() const { return Op_RegI; }
};
```

Intel RTM: C2 intrinsics

Добавим compiler nodes в `intrinsicnode.hpp`:

```
class XBeginNode: public Node {
public:
    XBeginNode(Node* pControl) {};
    virtual int Opcode() const { return Op_Comp; }
    virtual const Type& GetOpType() const {
        return TypeInt::INT;
    }
    virtual uint ideal_reg() const { return Op_RegI; }
};
```

XTestNode
по аналогии

Intel RTM: C2 intrinsics

Добавим compiler nodes в `memnode.hpp`:

```
class XEndNode: public MemBarNode {  
public:  
    XEndNode(Compile* C, int alias_idx, Node* precedent)  
        : MemBarNode(C, alias_idx, precedent) {}  
    virtual int Opcode() const;  
};
```

По аналогии с интринсиком для `Thread.onSpinWait()`

Intel RTM: C2 intrinsics

Добавим compiler nodes в memnode.hpp:

```
class XEndNode: public MemBarNode {  
public:  
    XEndNode(Compile* C, MemBarNode* precedent)  
        : MemBarNode(C, precedent) {}  
    virtual int Opcode() const;  
};
```

XAbortNode
по аналогии

По аналогии с интринсиком для Thread.onSpinWait()

Intel RTM: C2 intrinsics

Matching rules in `x86.ad`:

```
const bool Matcher::match_rule_supported(int opcode) {  
...
```

```
    case Op_XBegin:
```

```
    case Op_XAbort:
```

```
    case Op_XEnd:
```

```
    case Op_XTest:
```

```
        if (VM_Version::supports_rtm() == false)
```

```
            ret_value = false;
```

```
        break;
```

Defined in `classes.hpp`

...

Intel RTM: C2 intrinsics

Matching code в **x86.ad**:

```
instruct xBegin(rRegI dst) %{  
    match(Set dst (XBegin));  
    format %{ "xbegin" %}  
    ins_encode %{  
        __ xbegin($dst$$Register);  
    %}  
    ins_pipe( pipe_slow );  
%}
```

Intel RTM: C2 intrinsics

Matching code в `x86.ad`:

```
instruct xBegin(rRegI dst) %{  
  match(Set dst (   
  format %{ "xbeg  
  ins_encode %{  
    __ xbegin($ds  
  %}  
  ins_pipe( pipe_slow );  
  %}
```

Аналогично с
`xend()`, `xabort()`,
`xtest(rRegI dst)`

Intel RTM: C2 intrinsics

Логика интринсификации в `library_call.cpp`:

```
bool LibraryCallKit::inline_rtm_xbegin() {  
    Node *res = new XBeginNode(control());  
    set_result(_gvn.transform(res));  
    return true;  
}
```

Intel RTM: C2 intrinsics

Логика интринсификации в `library_call.cpp`:

```
bool LibraryCallKit::inline_rtm_xbegin() {  
    Node *res = ne  
    set_result(_gv  
    return true;  
}
```

`inline_rtm_xtest()`

по аналогии

Intel RTM: C2 intrinsics

Логика интринсификации в `library_call.cpp`:

```
bool LibraryCallKit::inline_rtm_xend() {  
    insert_mem_bar(Op_XEnd);  
    return true;  
}
```

Intel RTM: C2 intrinsics

Логика интринсификации в `library_call.cpp`:

```
bool LibraryCallKit::inline_rtm_xend() {  
    insert_mem_barriers();  
    return true;  
}
```

`inline_rtm_xabort()`
по аналогии