Дедлоки в корпоративных системах

источники, влияние, управление

Святослав Штумпф.

CROSSOVER GFI Software™

Deadlock

Dead = смертельный, lock = запор.

1. Простая проблема. Все понятно.
2. Как избежать - тоже все знают.
    a. Общество Мертвых Потоков http://www.slideshare.net/23derevo/ss-53186014
    b. Обедающие Философы наносят ответный удар! https://youtu.be/qrQ21myRSRc?list=PLVe-2wcL84b9rmcHD9w5Az5-otb4_V5D-
    c. Где моя память, чувак? https://www.youtube.com/watch?v=sSmQ6W-ovZE
3. Но все равно происходят детективные истории...

О чем же доклад

1. Все правильно сделал. Откуда все равно может взяться дедлок?
2. Каскадные последствия дедлока: «от того, что в кузнице не было гвоздя»
3. Подготовка корпоративного приложения к счастливой жизни заказчика:

- Как протестировать;
- Как выявлять и устранять.

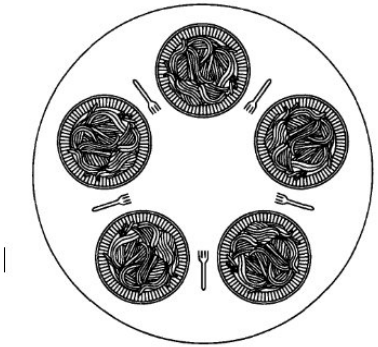
Синхронизация в Java: источники дедлоков.

Пока есть жизнь в ИТ, будут дедлоки.

«Питательная среда» - разделяемый ресурс.

Типы дедлоков:

- Взаимные (пробка на дороге);
- Кольцевые (обедающие философы);
- Рекурсивные (кусаем себя за хвост).

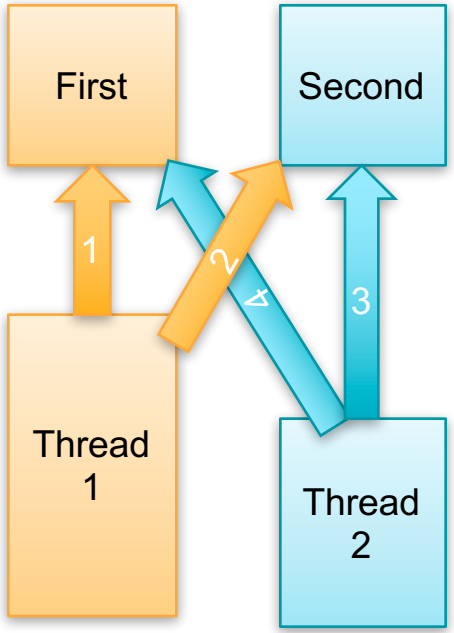


Пример неочевидного дедлока

```
static class First {
    static final Second _second = new Second();
}

static class Second {
    static final First _first = new First();
}

public static void main(String[] args) {
    system.out.println(“Первый пошел…”);
    new Thread(First::new).start();
    system.out.println(“Второй пошел…”);
    new Second();
    system.out.println(“Работа выполнена”);
}
```

Пример неочевидного дедлока

```
static class Fi
    static final Second _second = new Secon
}

static class Second {
    static final First _first = new First();
}

public static void main(String[] args) {
    system.out.println(“Первый пошел…”);
    new Thread(First::new).start();
    system.out.println(“Второй пошел…”);
    new Second();
    system.out.println(“Работа выполнена”);
}
```



Это выдумка, в жизни такого не бывает…

https://github.com/querydsl/querydsl/issues/1237

```
public final class Ops {
    public static final Operator<Boolean> EQ = new OperatorImpl<Boolean>(NS, "EQ");
    public static final Operator<Boolean> NE = new OperatorImpl<Boolean>(NS, "NE");
    ...

public final class OperatorImpl<T> implements Operator<T> {

    static {
        try {
            // initialize all fields of Ops
            List<Field> fields = new ArrayList<Field>();
            fields.addAll(Arrays.asList(Ops.class.getFields()));
            for (Class<?> cl : Ops.class.getClasses()) {
                fields.addAll(Arrays.asList(cl.getFields()));
            }
            ...
```

Пример неочевидного дедлока

```
class Deadlock {
    static {
        IntStream.range(0,100)
           .parallel().map(i -> i).count();
        System.out.println("done");
    }
    public static void main(final String[] args) {}
}
```

- статический инициализатор
- он же + лямбда-выражение

https://bugs.openjdk.java.net/browse/JDK-8036728
https://bugs.openjdk.java.net/browse/JDK-8143380
https://bugs.openjdk.java.net/browse/JDK-8136753

- Not An Issue.
- RTFM © Stuart Marks.

JVM Spec §5.5 - initialization lock

Проблема real-time расчетов

Отложенные расчеты

Просто 

Медленно 

Небезопасно 

Расчеты real-time

Быстро 

Прозрачно 

Безопасно (?) 

➔ Разделяемый ресурс (счета)

➔ Deadlock где-то рядом…

Полностью победить дедлоки

реалистический и фантастический варианты

Вариант 1

- Donate SETI Institute
  SETI = Search for Extra-Terrestrial Intelligence.
- Дожидаемся результатов контакта.
- Проблема решена.

Вариант 2

- Всегда работаем с ресурсами с гарантированным SLA.
- Всегда снимаем лок перед внешним вызовом.
- Всегда соблюдаем иерархию локов в любых (распределенных) командах и проектах любого размера.
- Всегда проектируем архитектуру с учетом будущего масштабирования.

Иерархия локов – переводим средства

```
private void doTransfer(
      final Account fromAcct, final Account toAcct,
      final DollarAmount amount) throws InsufficientFundsException {
        if (fromAcct.getBalance().compareTo(amount) < 0)
                throw new InsufficientFundsException();
        else {
                fromAcct.debit(amount);
                toAcct.credit(amount);
        }
}
```

Иерархия локов – скрытая угроза

```
public void transferMoney(final Account fromAcct,
          final Account toAcct, final DollarAmount amount)
          throws InsufficientFundsException {

        synchronized (fromAcct) {
            synchronized (toAcct) {
                doTransfer(fromAcct, toAcct, amount)
            }
        }
}
```

- А теперь оба окурка давим одновременно…

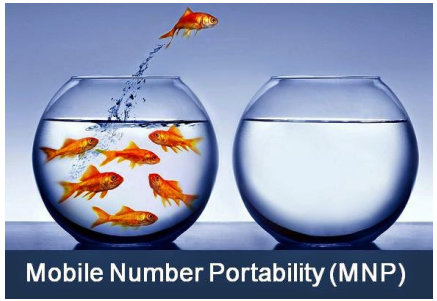
Иерархия локов – скрытая угроза

Порядок взятия лока:

- Основан на инварианте, внешнем по отношению к коду.
- И даже **внешнем по отношению к бизнес-процессу**.

Риск промышленных систем: инварианты со временем могут меняться…

Пример: **Mobile Number Portability**.



Иерархия локов – чиним перевод

А нужно всего лишь…

```
if (fromAcct.getId() < toAcct.getId()) {
     synchronized (fromAcct) {
          synchronized (toAcct) {
               doTransfer(fromAcct, toAcct, amount)}
          }
     }
} else  {
     synchronized (toAcct) {
          synchronized (fromAcct) {
               doTransfer(fromAcct, toAcct, amount)}
          }
     }
}
```

'volatile' – **нарушение атомарности**

```
// Jetty 7.1.0, org.eclipse.jetty.io.nio,
// SelectorManager.java, line 105
private volatile int _set;
......
public void register(SocketChannel channel, Object att)
{
        int s=_set++;
        ......
}
......
```

- Используем явную синхронизацию
- или атомарные классы из пакета java.util.concurrent.

'volatile' – **чиним**

```
// Jetty 7.1.0, org.eclipse.jetty.io.nio,
// SelectorManager.java, line 105
private volatile int _set;
......
public void register(SocketChannel channel, Object att)
{
        synchronized (this) {
                int s=_set++;
        }
                ......
}
......
```

‘java.util.concurrent‘ - **многоступенчатый доступ**

```
public class Balance {
        private final ConcurrentHashMap nameToNumber;
        private final ConcurrentHashMap numberToBalance;
        ... различные методы для добавления, удаления и т. д.

        public int geBonusFor(String name) {
                Integer MSISDN = nameToNumber.get(name);
                Balance balance = numberToBalance.get(MSISDN);
                return balance.getCharges();
        }
}
```

‘java.util.concurrent‘ - **многоступенчатый доступ**


```
public class Balance {
        private final HashMap nameToNumber;
        private final HashMap numberToBalance;
        ... различные методы для добавления, удаления и т. д.

        public int synchronized getChargesFor(String name) {
                Integer MSISDN = nameToNumber.get(name);
                Balance balance = numberToBalance.get(MSISDN);
                return balance.getCharges();
        }
}
```

‘java.util.concurrent‘ - **утечка блокировки**

```
private final Lock lock = new ReentrantLock();

public void lockLeak() {
     lock.lock();
     try {
               // доступ к совместно используемому ресурсу
               accessResource();
               lock.unlock();
     }
     catch (Exception e) {}

public void accessResource() throws InterruptedException {...}
```

‘java.util.concurrent‘ - **утечка блокировки**.

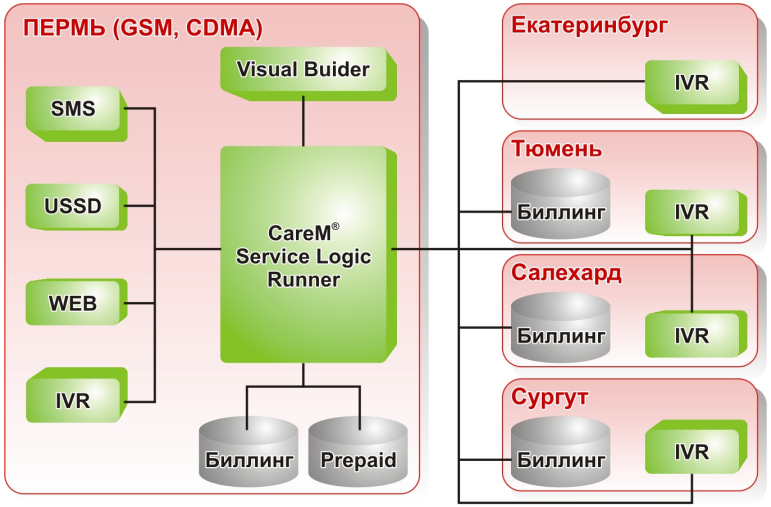
```
public void lockLeak() {
     lock.lock();
     try {
               // доступ к совместно используемому ресурсу
               accessResource();

     }
     catch (Exception e) {}

     finally {
               lock.unlock();
     }
```

Еще об инвариантах…

- Динамическое конфигурирование пользовательских сценариев.
- Возможность подключения внешнего кода в runtime.
- Изоляция доступа к внешним системам.

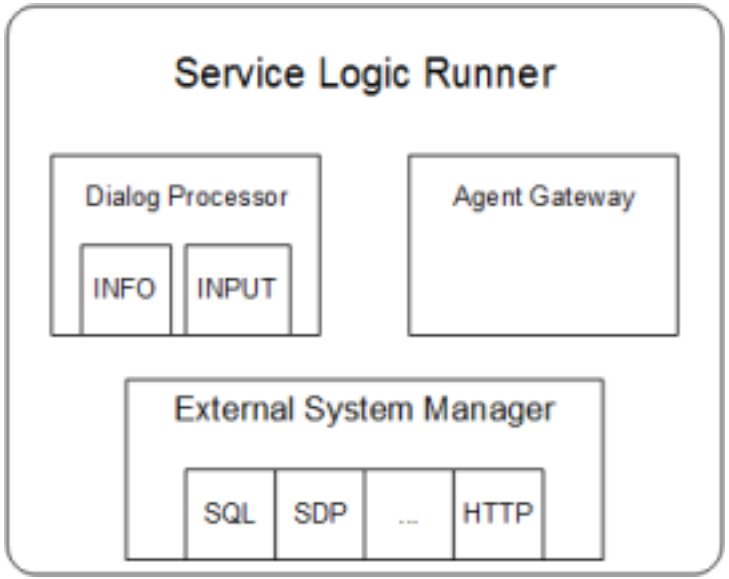
CaReM – Работает с 2003 года.

100 000 000+ уникальных пользователей.

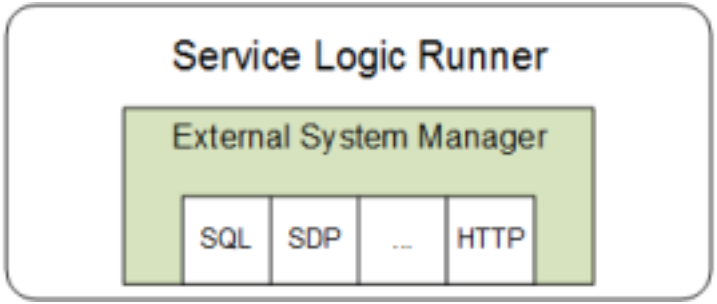


Service Logic Runner

- Взаимная блокировка сценариев
- Работа с одним набором таблиц.
- Блокировка внешних источников данных от изменений.
- Взаимная блокировка сессий в агентах
- Блокировки на источниках данных



External System Manager



- UDP-источник → одна сессия “запрос-ответ”
  - Один поток послал запрос. Источник ответил, но ответ пропал.
  - Параллельный поток послал запрос - источник ждет другого ответа…
  - = два потока ждут, источник ждет.

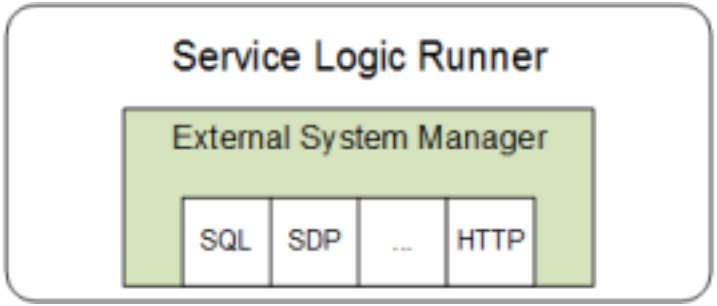
Не гарантировано взятие лока и освобождение лока.

External System Manager

- SOAP-источник → перемешивание multipart-запросов.

  Не гарантировано завершение операции,
  выполняемой во время блокировки.

External System Manager



- JDBC-источник → проблемы дедлоков в драйверах.

JDK-6354348 : Deadlock between threads in java/sql/DriverManager code

Workaround:

Было → `Class.forName("oracle.jdbc.driver");`

Стало →
```
synchronized(java.sql.DriverManager.class) {
    Class.forName("oracle.jdbc.driver");
}
```

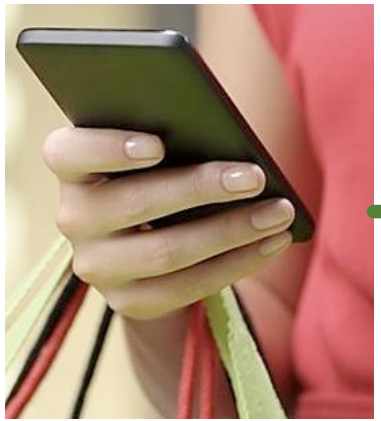
JDK-7106332 : MSSQL JDBC Driver (Type4) deadlock while establishing connection

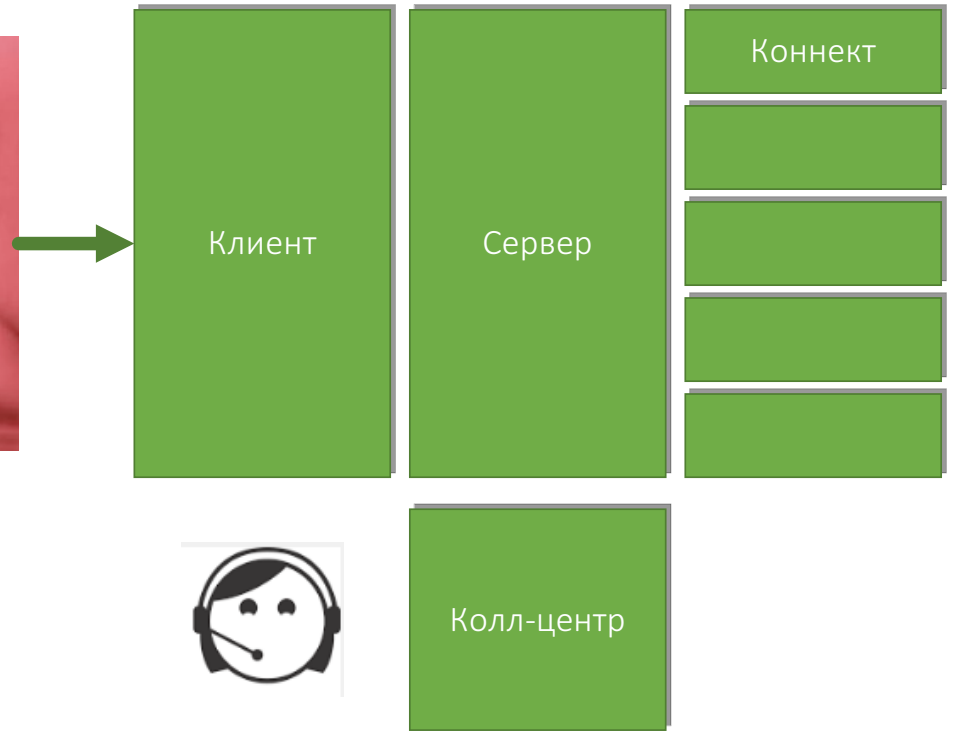
О чем же доклад

1. Все правильно сделал. Откуда все равно может взяться дедлок?
2. **Каскадные последствия дедлока: «от того, что в кузнице не было гвоздя»**
3. Подготовка корпоративного приложения к счастливой жизни заказчика:

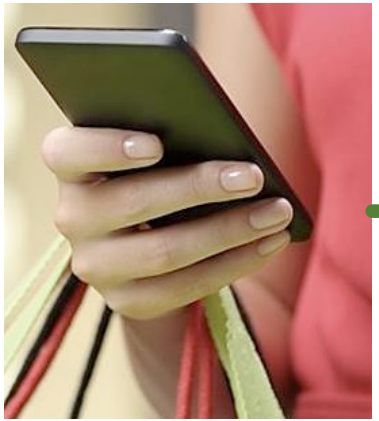
- Как протестировать;
- Как выявлять и устранять.

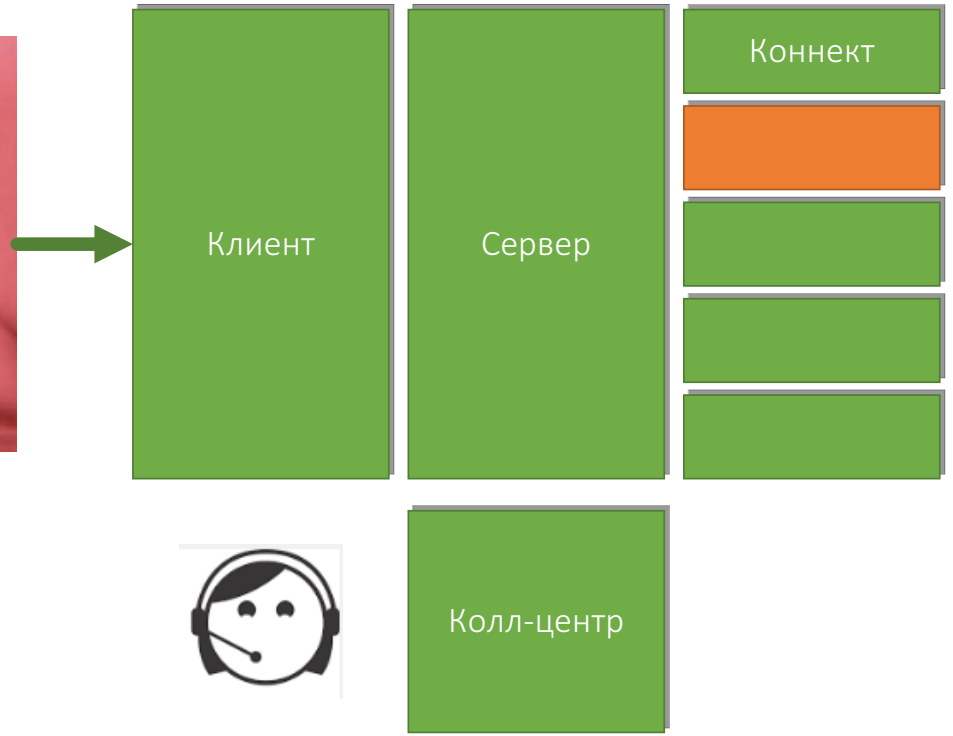
Детектив: почему умер колл-центр



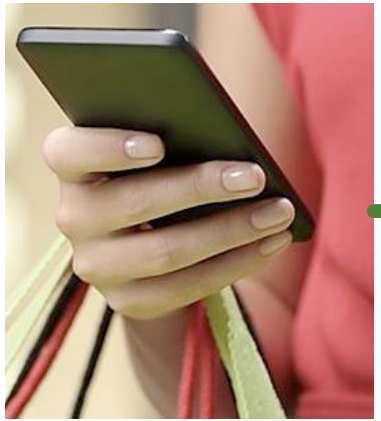


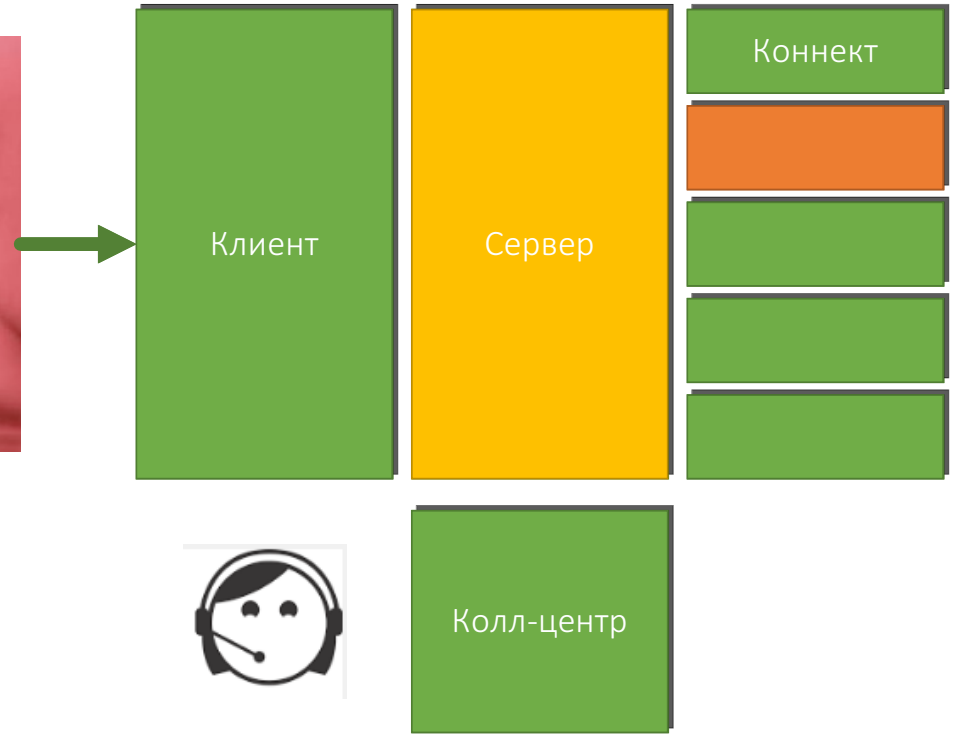
Deadlock на одном из коннектов…





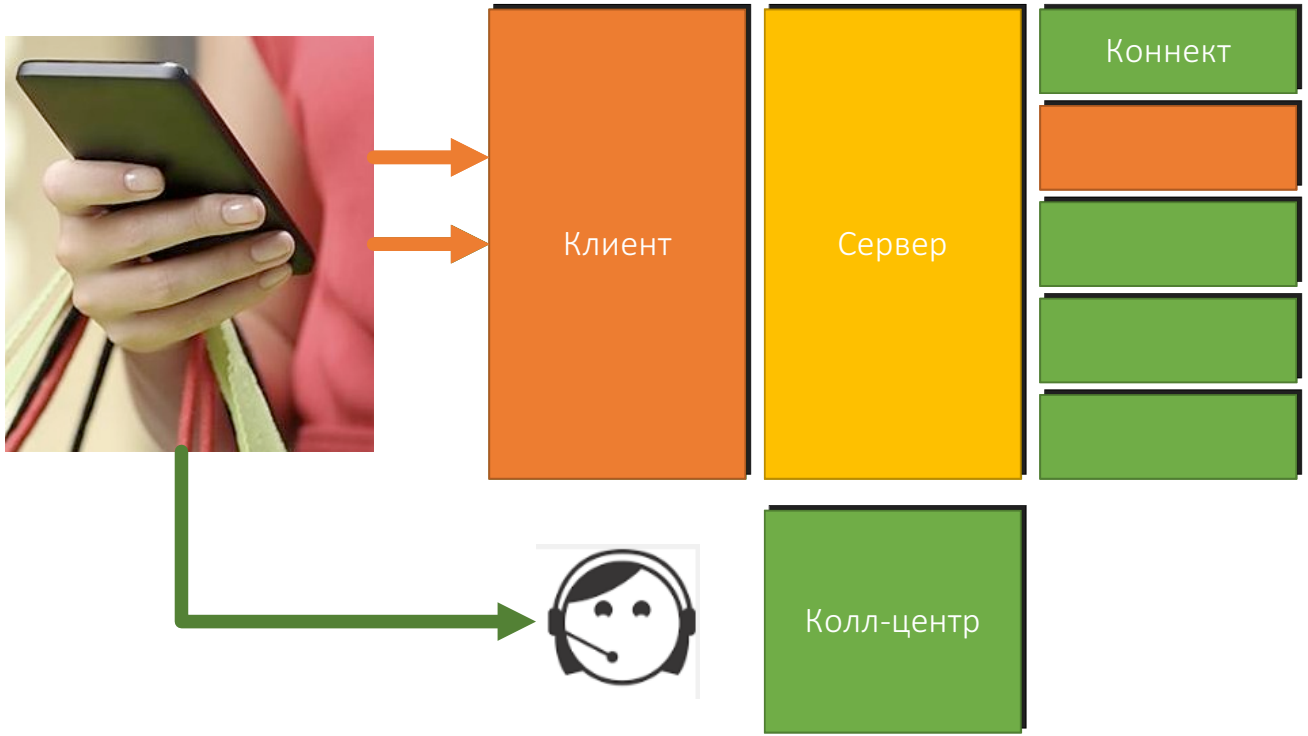
… рост очередей на сервере …



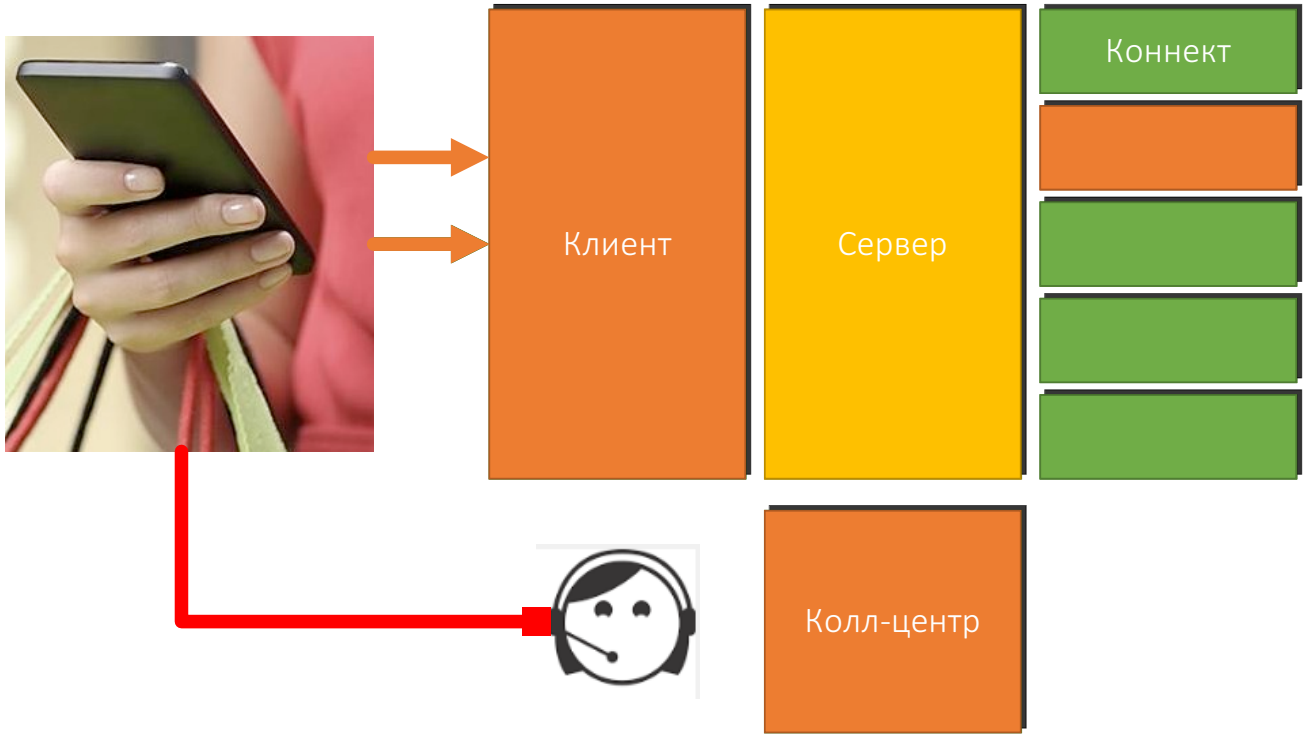


… отбой клиентов…

… массовые звонки в колл-центр…



… и перегрузка колл-центра.



О чем же доклад

1. Все правильно сделал. Откуда все равно может взяться дедлок?
2. Каскадные последствия дедлока: «от того, что в кузнице не было гвоздя»
3. Подготовка корпоративного приложения к счастливой жизни заказчика:

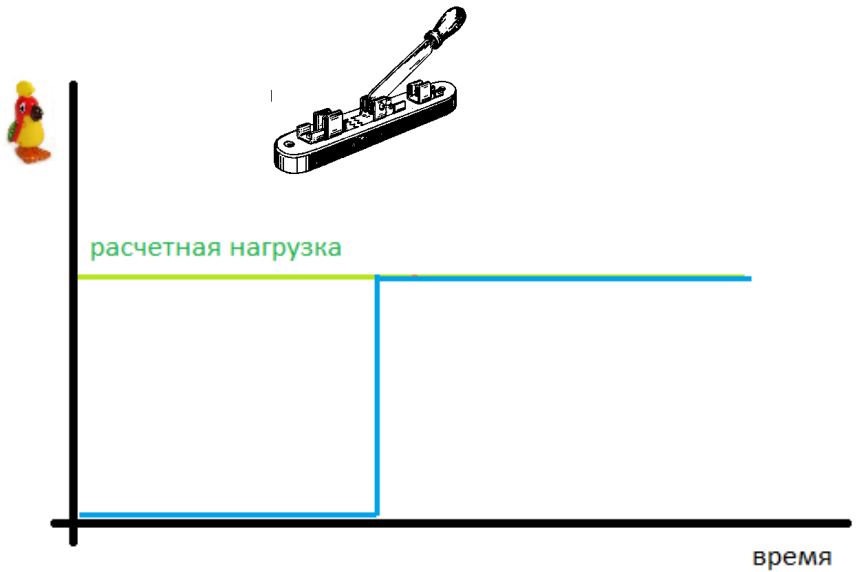
- **Как протестировать;**
- Как выявлять и устранять.

О ловле дедлоков

«Удар» нагрузкой

Проверяем:

- Дедлоки в инициализаторах, загрузчиках, компонентах старта.
- Способность пережить «падение» и восстановление внешних компонентов.

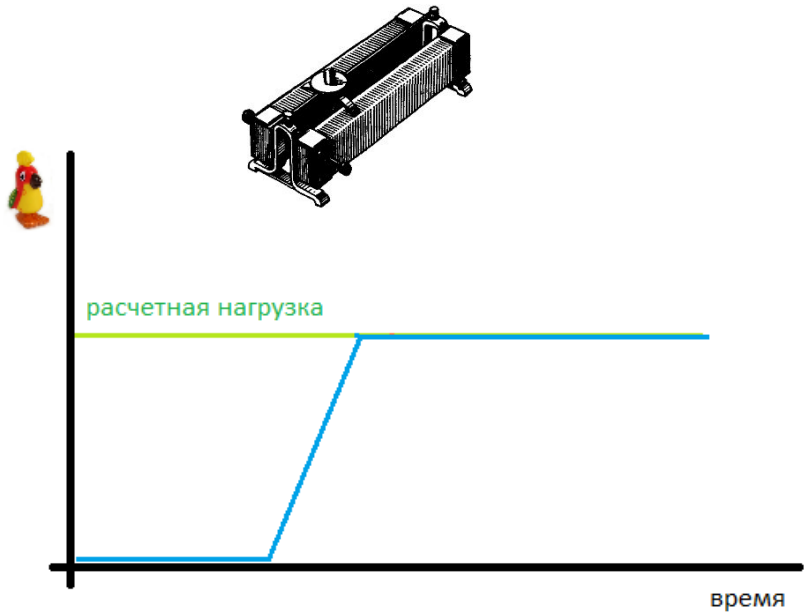


О ловле дедлоков

Плавный старт

Проверяем:

- Дедлоки при обслуживании пользовательских сессий.
- Корректность доступа к данным из разных частей пользовательской сессии.

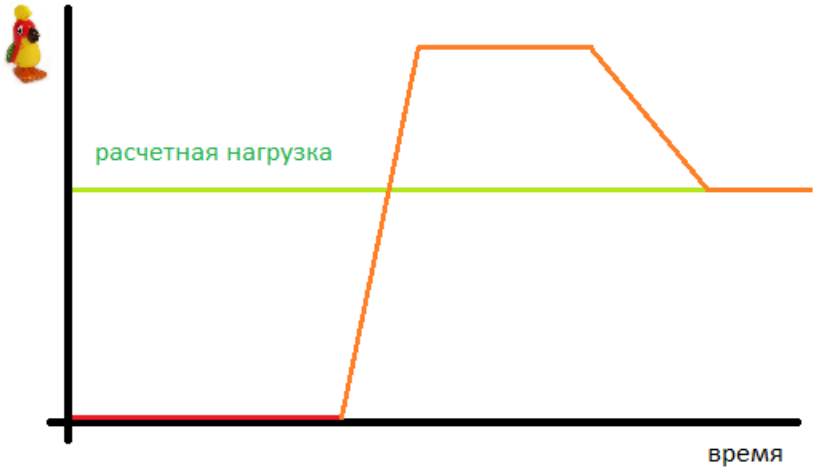


О ловле дедлоков

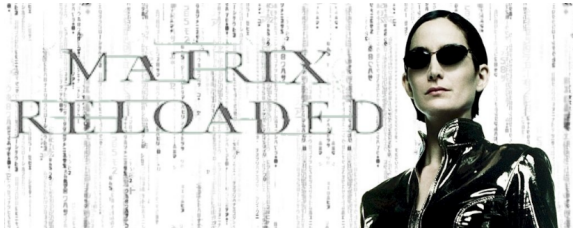
Перегрузка

Проверяем:

- Способность защитных механизмов корректно защитить систему.
- Согласованность ограничений на всех компонентах системы.



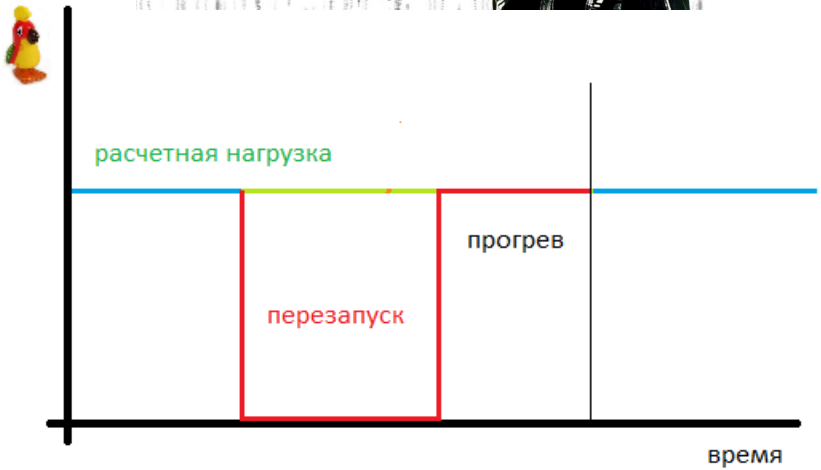
О ловле дедлоков



Перезапуск под нагрузкой

Проверяем:

- Выделение ресурсов, инициализацию кэшей, одновременный старт сервисных потоков на *непрогретой* системе.
- Способность системы справиться с нештатными таймаутами, долгим ожиданием ресурсов, «перемешиванием» порядка доступа.

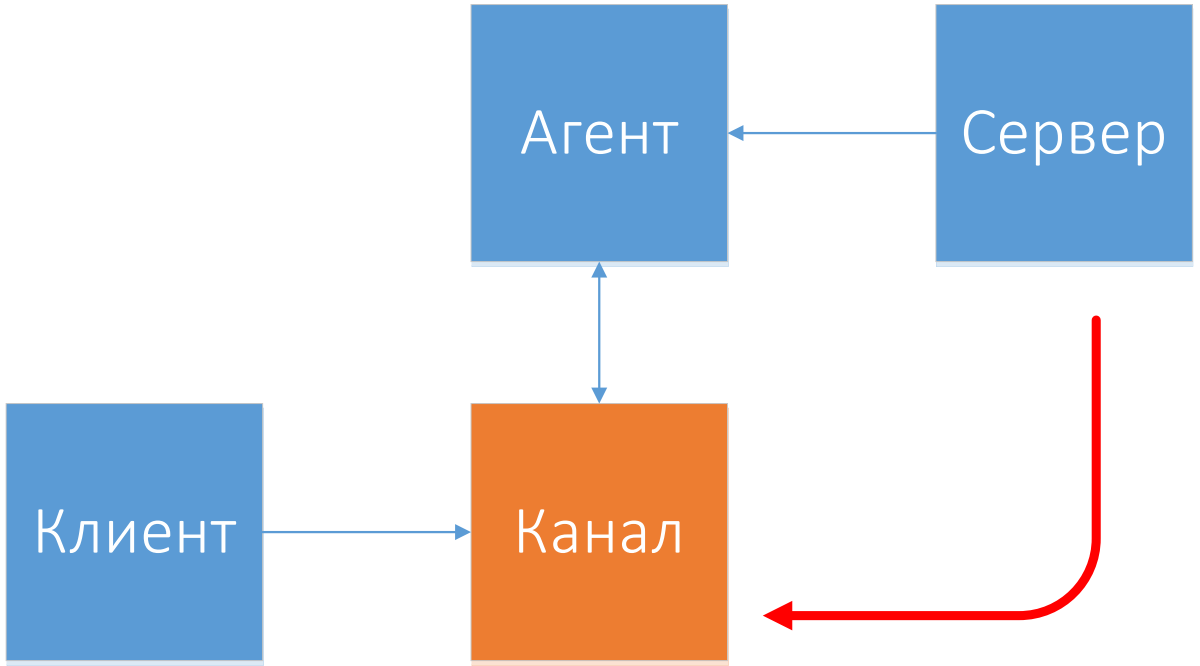


Детектив – куда утекают каналы

IVR-система обслуживания клиентов

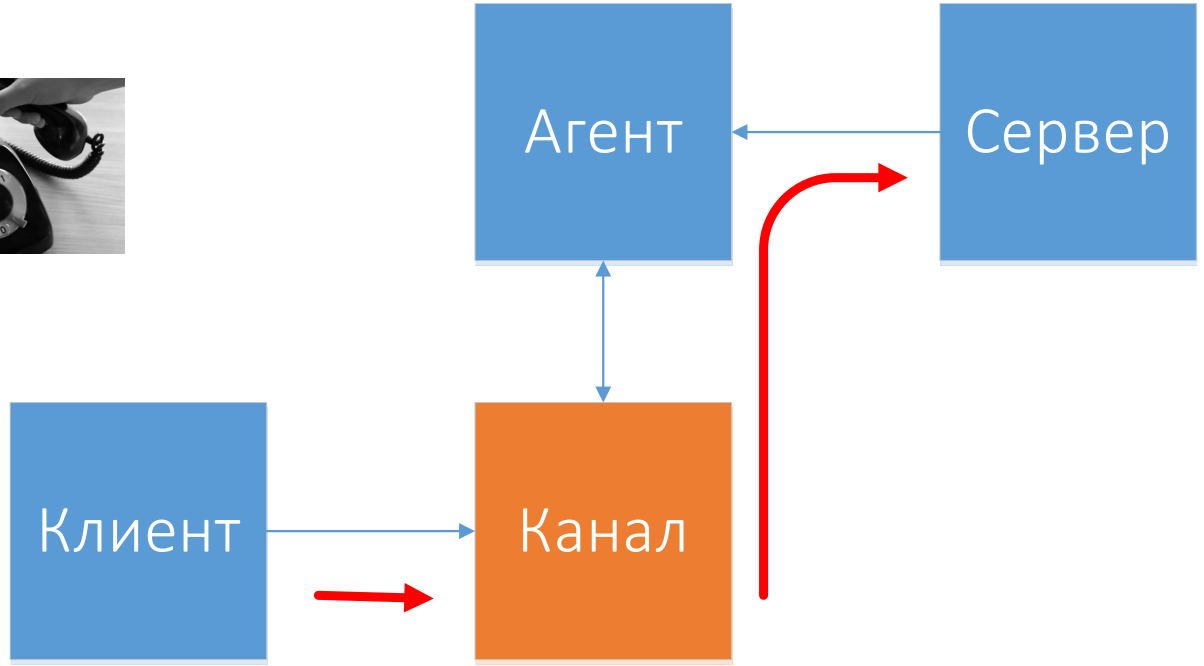
Отбой сессии сервером

Сессия завершена, канал освобожден.



Отбой сессии клиентом

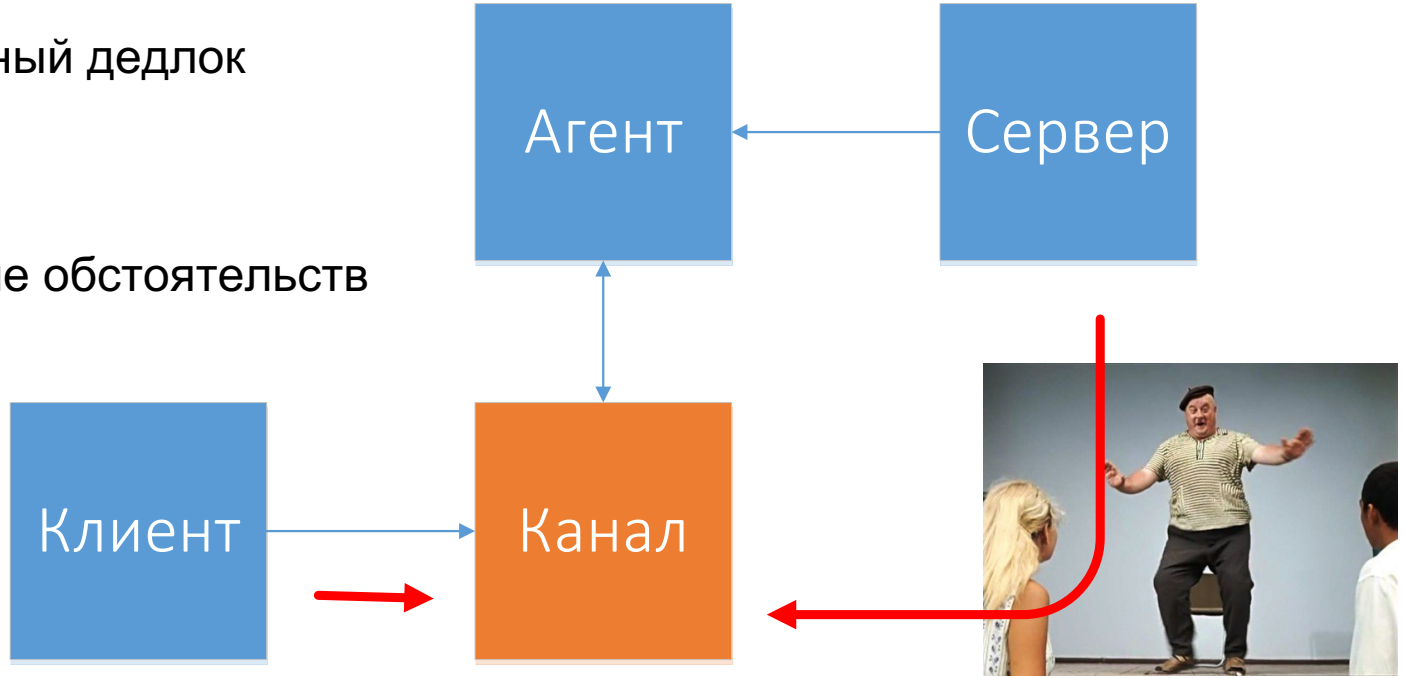
Канал освободился, сессия завершена.



Deadlock – канал остался заблокирован

Одновременный отбой с двух сторон.

- банальный дедлок
- JNI
- стечение обстоятельств



О чем же доклад

1. Все правильно сделал. Откуда все равно может взяться дедлок?
2. Каскадные последствия дедлока: «от того, что в кузнице не было гвоздя»
3. Подготовка корпоративного приложения к счастливой жизни заказчика:

- Как протестировать;
- **Как выявлять и устранять.**

Обнаружение дедлока - просто?

Это код

```
static class First {
    static final Second _second = new Second();
}

static class Second {
    static final First _first = new First();
}

public static void main(String[] args) {
    system.out.println(“Первый пошел…”);
    new Thread(First::new).start();
    system.out.println(“Второй пошел…”);
    new Second();
    system.out.println(“Работа выполнена”);
}
```

Обнаружение дедлока - просто?

Это Thread Dump

```
"Thread-0" #12 prio=5 os_prio=0 tid=0x000000001a098800 nid=0x1cf8 in
Object.wait() [0x000000001a95e000]
   java.lang.Thread.State: RUNNABLE
         at Deadlock$First.<clinit>(Example1.java:4)
         at Deadlock$$Lambda$1/1418481495.run(Unknown Source)
         at java.lang.Thread.run(Thread.java:745)

"main" #1 prio=5 os_prio=0 tid=0x000000000098e800 nid=0x23b4 in Object.wait()
[0x000000000228e000]
   java.lang.Thread.State: RUNNABLE
         at Deadlock$Second.<clinit>(Example1.java:8)
         at Deadlock.main(Example1.java:13)
```

А что еще не так с Thread Dump?

Доступ к корпоративным системам.

1. Система под нагрузкой – мониторинг может перегрузить
2. Безопасность.
3. Права доступа и изоляция.

А если выявили дедлок? Не остановить… не перезапустить…

Обнаружение дедлока - проверяем результат

Анализ функциональных точек.

- Alan Albrecht - 1979
- International Function Point User Group

Функциональные точки:

- Данные.
- Транзакции.

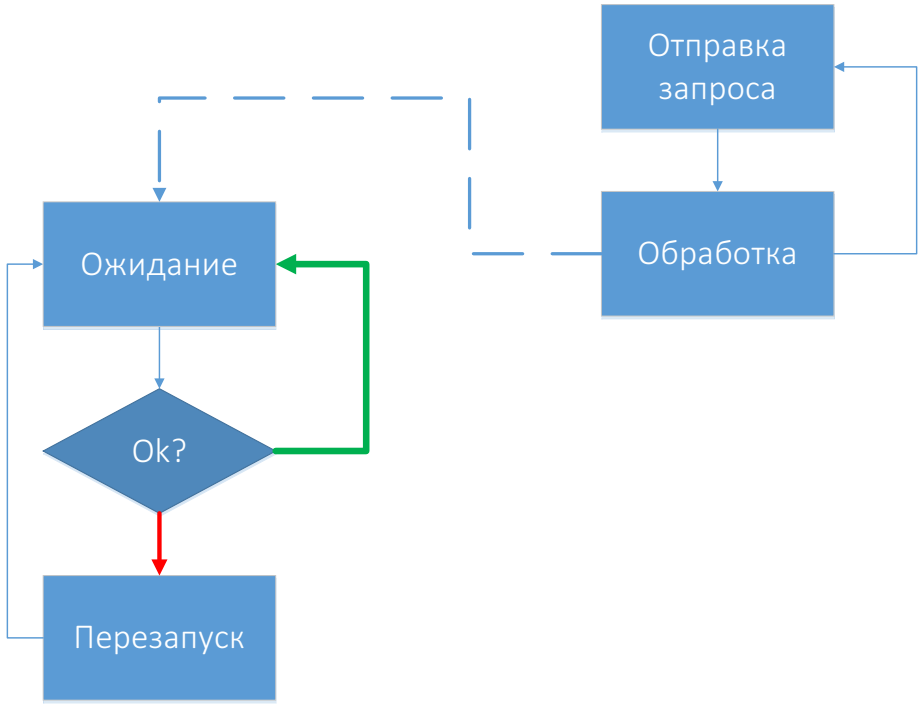
Контроль:

- Alive request;

Обнаружение дедлока - проверяем результат

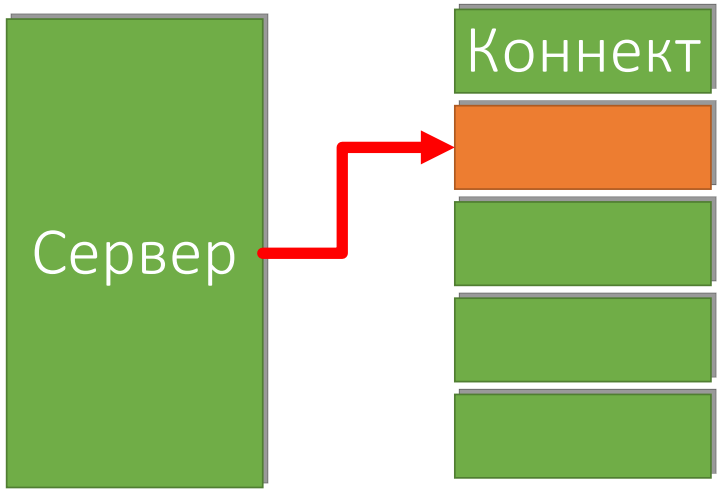
Контроль:

- Alive request;
- Counter;
- Heartbeat.



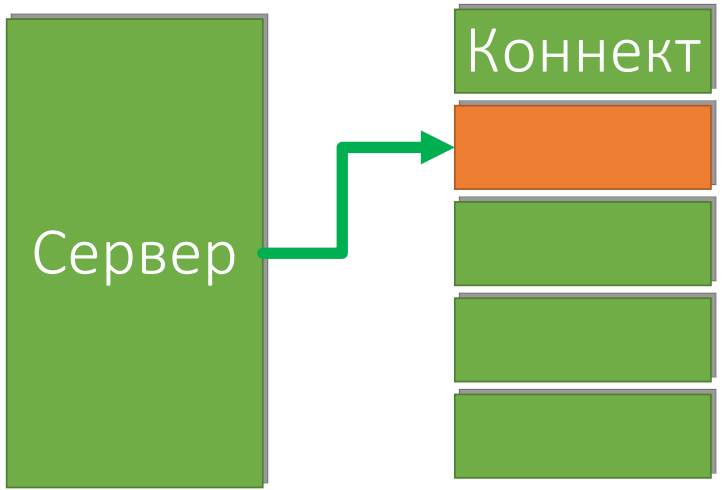
Обнаружили deadlock – и что?

- Необходимо заложить в архитектуру
- Возможность перезапуска функциональных точек (извне и автоматически).



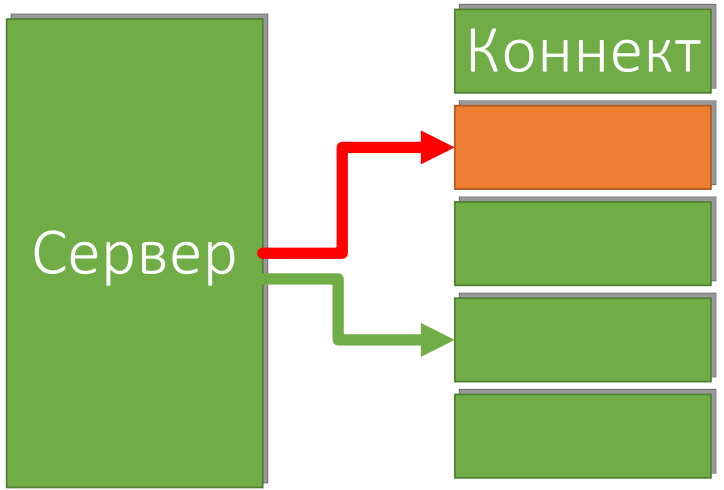
Обнаружили deadlock – и что?

- Необходимо заложить в архитектуру
- Возможность перезапуска функциональных точек (извне и автоматически).
- Автоматический реконнект при перезапуске.



Обнаружили deadlock – и что?

- Необходимо заложить в архитектуру
- Возможность перезапуска функциональных точек (извне и автоматически).
- Автоматический реконнект при перезапуске.
- Таймауты.



О чем же доклад

1. Все правильно сделал. Откуда все равно может взяться дедлок?
2. Каскадные последствия дедлока: «от того, что в кузнице не было гвоздя»
3. Подготовка корпоративного приложения к счастливой жизни заказчика:

- Как протестировать;
- Как выявлять и устранять.

4. Что-то ещё?

Проблема решена?

Deadlock обнаружен.

- Ситуация **устранена**, компонент **перезапущен**.
- Разделяемый ресурс **освободился**.
- Можно **двигаться дальше**.
- **Всем сразу?**

А еще бывает live lock…

Live lock:

- Тоже есть **разделяемый ресурс**.
- Тоже есть **конфликт** за него.
- Но не ждем, а **активно конкурируем**.

«Бег по кругу». Зато просохнем.

Выводы

- Будет многопоточность - будет и дедлок.
- Рекомендации на 100% невыполнимы
  - Это не повод их не выполнять.
- Некоторые дедлоки - “документированный баг = фича”
  - RTFM.

Выводы

- Предотвращай дедлоки при проектировании
  - Но отслеживай их и по функциональным точкам.
  - И закладывай в архитектуру возможность легко перезапустить компоненты.
- Ловля дедлоков - творческий процесс
  - Нужно попробовать множество вариантов
  - Нужно вносить случайность
  - Нужно провоцировать “накладки”

Выводы

- Избегание дедлоков <> решение проблемы
  - Параллельность не ускоряет, а лишь оптимизирует.
  - Борьба с дедлоками не должна поглощать заметный % производительности.
  - Можно создать live lock – при сверхаккуратности.

Всё. Спасибо!

CROSSOVER GFI Software™