Инженерный подход к разработке SW/HW

Лекция 2

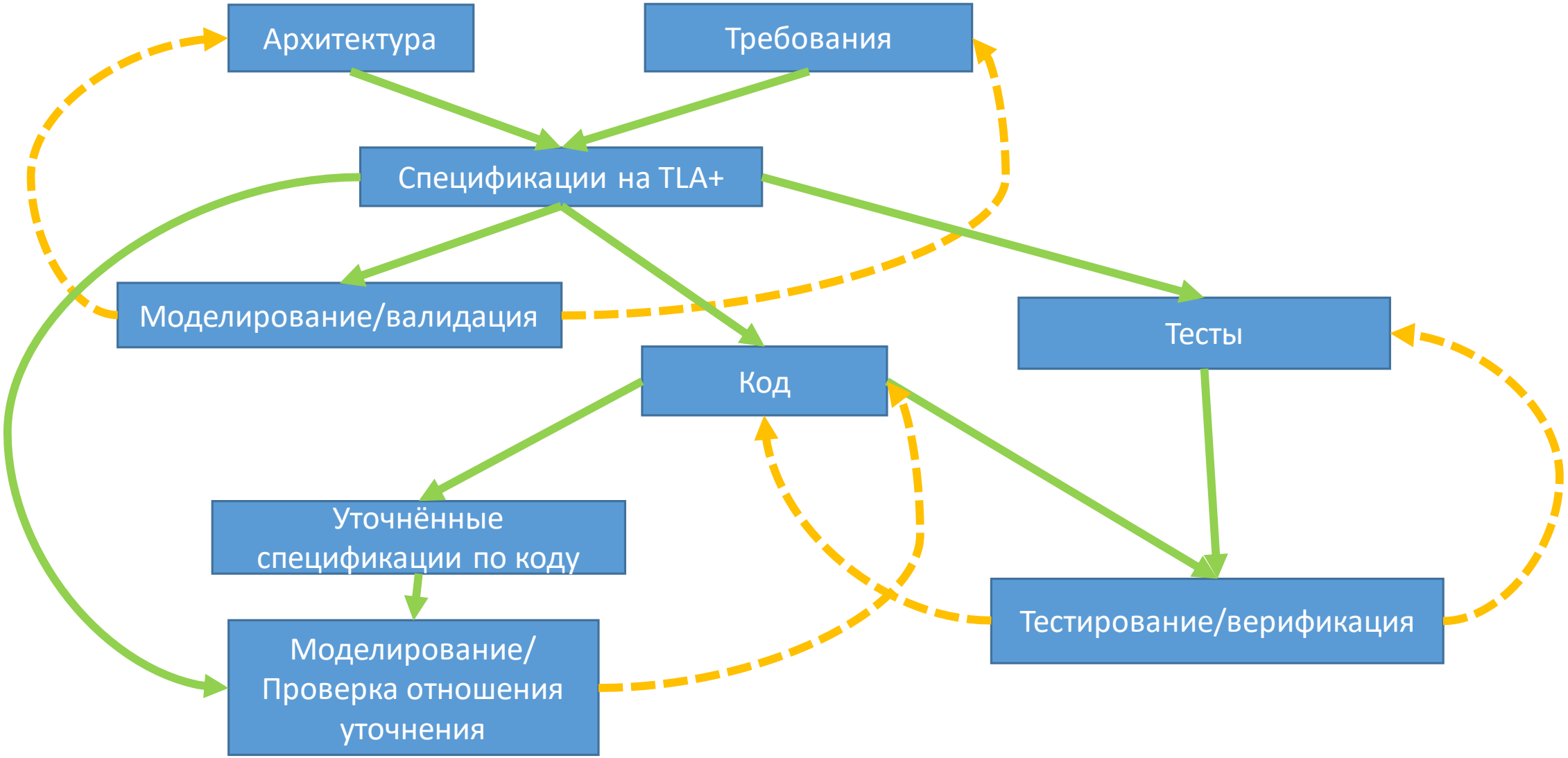
План

- Пример инженерного подхода при разработке OpenCom RTOS
- Основные результаты этого проекта
- Небольшой экскурс в операционные системы: основные компоненты, какие операционки бывают и тд
- Постановка проблематики управления памятью: менеджер памяти – что должен делать? Какими свойствами обладать? Как сформулировать свойства? Как проверить?
- Что такое спецификация? Формальная спецификация? Модель? Валидация?

OpenCom RTOS

- Операционная система для широкого промышленного применения: от спутников и радаров до оборудования нефтепромышленности
- Первая ОС реального времени, которая разработана полностью на основе формальных спецификаций на TLA+. Промоделирована с помощью TLC
- В основе лежит теория SCP (communicating sequential processes) Тони Хоара.
- Портирована на все основные процессоры и микроконтроллеры, которые широко используются в промышленности
- Опыт настолько удачный, что последующая ОС PB Virtuoso Next была тоже разработана на основе формальных спецификаций на TLA+

OpenCom RTOS, процесс разработки



Основные результаты проекта OpenCom RTOS

- Чистая, ясная и модульная архитектура
- Объём исходного кода, по сравнению с предыдущей версией RTOS Virtuoso, уменьшился в 10 раз! За счёт формальных спецификаций и моделирования были найдены архитектурные оптимизации, которые позволили многократно снизить объёмы “лишнего” кода.
- Нашли новые алгоритмические решения в области планирования ресурсов системы, что позволило существенно снизить время реакции системы по сравнению с решениями на основе других RTOS
- OpenCom RTOS не подвержена ошибкам переполнения буферов, by-design, так как разработчики придумали и внедрили новый способ управления памятью.
- Проект был завершён в срок. Практически не потребовалось отладки после реализации проекта, так как тесты, разработанные по формальным моделям, хорошо закрыли все “острые углы”.

Операционные системы

- ОС – это базовая программа для управления ресурсами компьютера, которая предоставляет пользовательским программам API, абстрагированный от ньюансов “железа”.
- Примеры таких API: POSIX, Win32, Win64, DOS (ms-dos, dr-dos), BeOS
- Примеры ресурсов, доступом к которым управляет ОС:
  - Память
  - Сеть
  - Диск
  - Таймеры
- Операционные системы существенно упрощают задачи по разработке ПО, предназначенного для работы на разных компьютерах и даже архитектурах

Типы ОС

- По способам предоставления ресурсов:
  - Batch – древние, уже не найти в активном использовании
  - time-sharing – обычные Unix, Windows, etc
  - Distributed – ориентированные на распределённые среды, например Plan9, Inferno, где окружение исполнения можно набрать из ресурсов разных машин: CPU одной, диск – другой, экран – третьей
  - Network – ориентированные на сетецентричную организацию, например Windows NT server + терминальные станции
  - Network-centric (не знаю как грамотно дифференцировать с Network) – ориентированные на сетевую структуру организации вычислений. Основные сущности в таких ОС – это процесс и коммуникационный канал. Пример – OpenCom RTOS
  - И др

Типы ОС

- По гарантиям на ресурсы:
  - Защищённая память / разделяемая процессами память
  - Реального времени / не реального времени – есть ли гарантия на максимальное время отклика системы
  - И др

Типы ОС

- По способу огранизации ядра ОС и предоставлению API
  - Микроядерные: GNU/Hurd, Minix, L3, L4, seL4
  - Монолитные: Linux, Windows
  - Unikernel: MirageOS
  - Экзоядерные: MIT XOK
  - И др

Управление памятью

- Разновидность управления доступом к ресурсам
- Многопоточный/многопроцессорный или однопоточный
- Реального времени или нет
- Фрагментация памяти
- Автоматическое освобождение (сборка мусора) или нет
- И тд.
- Алгоритмов управления памятью множество, как и разных реализаций с разными свойствами.
- Обычно менеджеры памяти подбираются/разрабатываются под определённый класс задач

Менеджер памяти, основные функции

- Выделить блок памяти запрошенного размера
- Освободить блок памяти (в случае со сборщиком мусора этот функционал опциональный)
- Могут быть дополнительные параметры запросов: выравнивание адреса и размера (например кратные 8 байтам), время доступа к блоку памяти (на NUMA архитектурах есть “своя” память, доступ к которой быстрый и “чужая” доступ к которой может быть медленнее во много раз) и тд.

Свойства менеджера памяти

- Структурные свойства:
  - Выделенные блоки памяти не перекрываются
  - Сумма выделенных блоков + сумма свободных = объем памяти (константа)
  - Если освободили все выделенные блоки памяти, независимо от последовательности выделения/освобождения, то сумма размеров свободных блоков = объем памяти. Нет “утечек” памяти.
  - Блоки памяти следуют без “дырок” – нет непродуктивных потерь памяти
  - Выделенный блок не выделяется повторно при запросе.
  - Можно ещё сформулировать требования к максимальной фрагментации памяти, к оверхеду служебных структур и тд.

Свойства менеджера памяти

- ВременнЫе свойства:
  - Реального времени или нет – есть ли гарантированное ограничение сверху на время выполнения запроса

Свойства менеджера памяти

- Надёжность и устойчивость к ошибкам:
  - Например: двойное освобождение блока памяти, попытка освободить блок, передав неправильный указатель и тд
  - Способность к детекции повреждения служебных структур: например, у пользователя произошел выход за границы массива и он записал данные в заголовок следующего блока.

Формулировка свойств

- Структурные удобнее всего сформулировать в Alloy
- Нам понадобится ввести:
  - Концепцию блока памяти
  - Концепцию адреса
  - Размера блока
- Потребуется определить:
  - Различные нужные нам отношения (порядка, следования и пр) над адресами, размерами и блоками
  - Строго (в виде предиката) свойство корректности структуры памяти
  - Операцию выделения блока
  - Операцию освобождения блока
- Потребуется проверить/доказать:
  - Сохранение свойства корректности структуры памяти операциями выделения и освобождения блока

Проверка свойств

- Для проверки будем использовать Alloy Analyzer:
  - Определим наши концепции адресов, размеров, блоков на языке Alloy
  - Определим отношения на языке Alloy
  - Определим операции выделения/освобождения блоков
  - Определим предикаты корректности структуры памяти
  - Проведём поиск моделей/контр-примеров в Alloy Analyzer
  - Проанализируем полученные результаты

Спецификация

- Это точная формулировка требуемых свойств к разрабатываемому ПО
- Плюс описание архитектуры/функционала/операций (может быть разных уровней абстракции)
- Формулируется на основе требований из предметной области
- Формулируется в рамках терминологии из области разработки
- Можно считать частью спецификации формулировку свойств на слайде 12
- Есть стандатрные языки для спецификаций, например UML, SysML и др

Формальная спецификация

- Это спецификация разработанная в рамках какого-либо формализма
- Основное предназначение:
  - валидация разрабатываемой системы на соответствие требованиям (проверка свойств с помощью model-checking и тд)
  - Формулировка инвариантов, ассертов и тд для кода
  - Разработка направленных тестов
- Пример: спецификация структурных свойств на языке Alloy

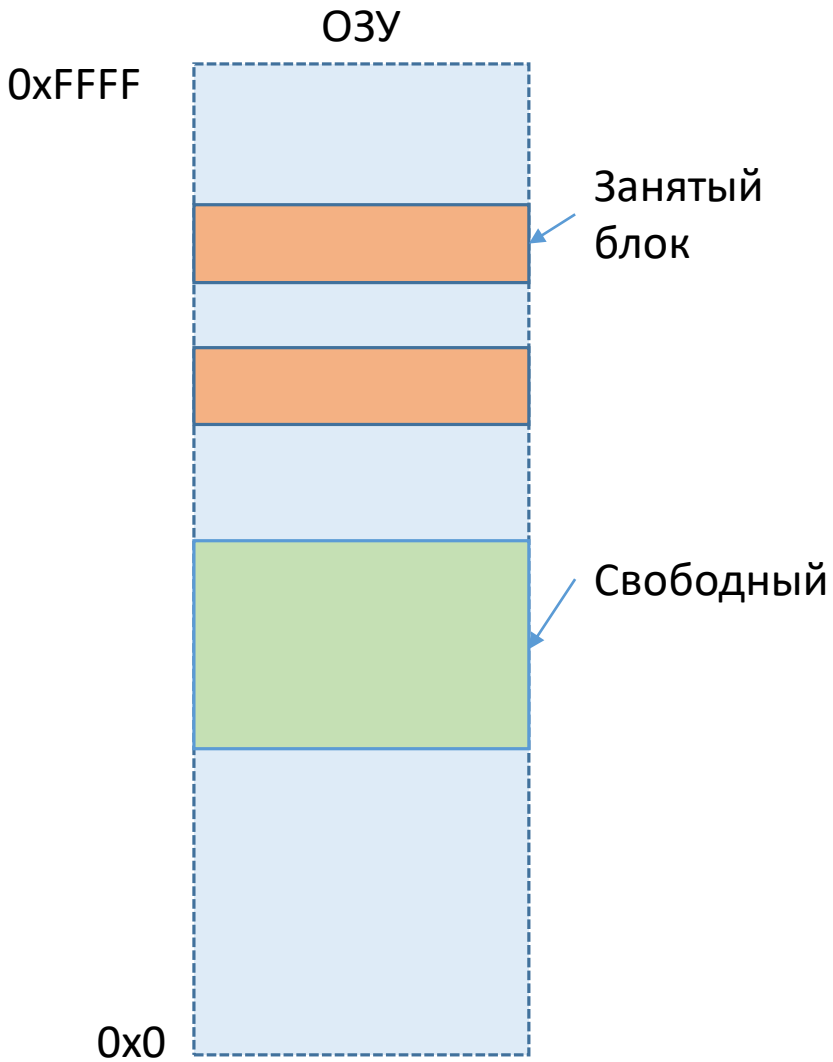
Модель системы

- Формальное абстрактное описание системы, её стурктуры/операций/функционала
- Для model-checking, как правило система описывается в виде некоторого множества взаимодействующих недетерминированных автоматов.
- Можно ещё различать модель структуры системы, когда мы формально описываем структуру системы в виде каких-то сущностей и отношений между ними. Например, будем на Alloy описывать модель структуры памяти для менеджера памяти.

Валидация

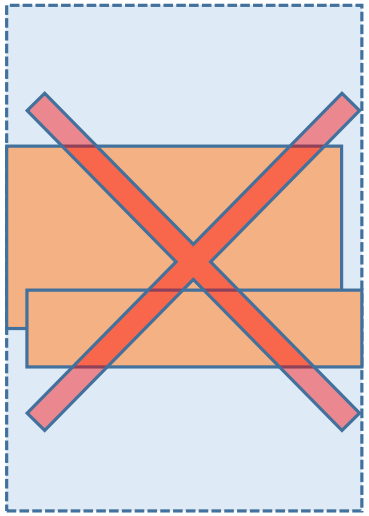
- Это процесс проверки формальной модели системы на соответствие необходимым нам свойствам (тоже формализованным)
- Например, мы проверим в Alloy Analyzer, что операции по работе с блоками в менеджере памяти, сохраняют свойство корректности структуры памяти.

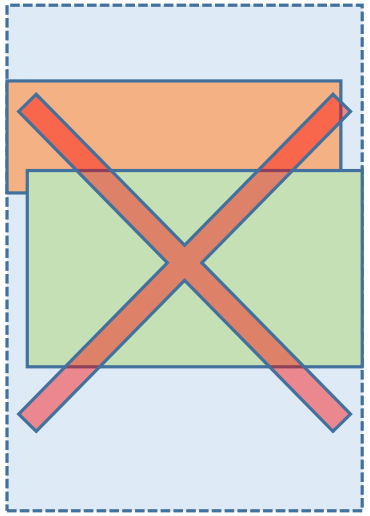
Корректность (валидность) структуры

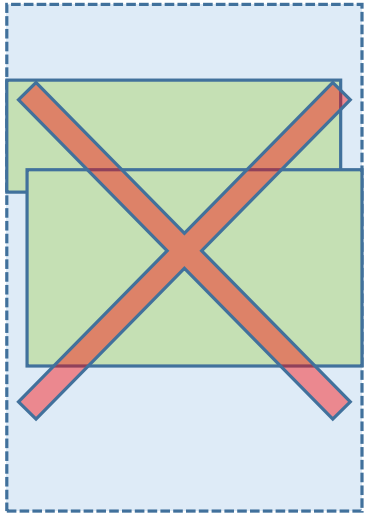


Какую структуру блоков в памяти будем считать корректной?

Наезжающие друг на друга блоки, это, наверное, неправильно

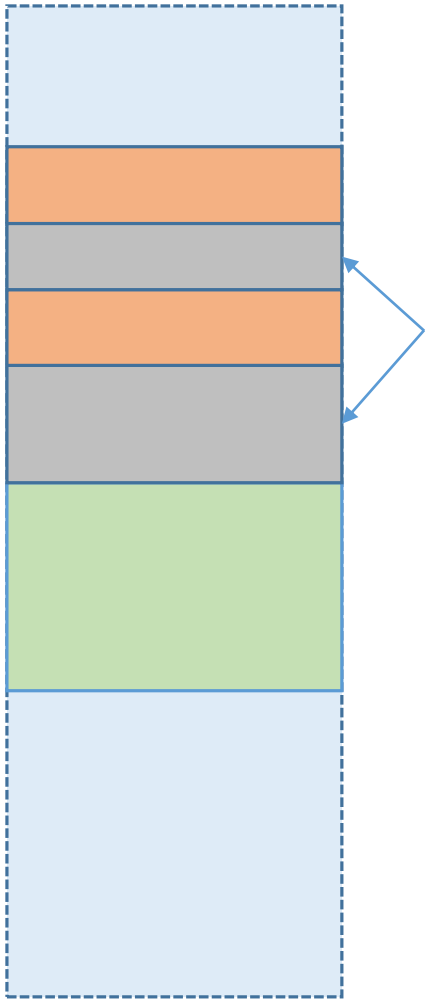






(хотя, в своё время фирма Intel всех убеждала, что наползающие друг на друга сегменты памяти в 8086 процессоре, это не баг, а фича ☺)

Валидность стурктуры

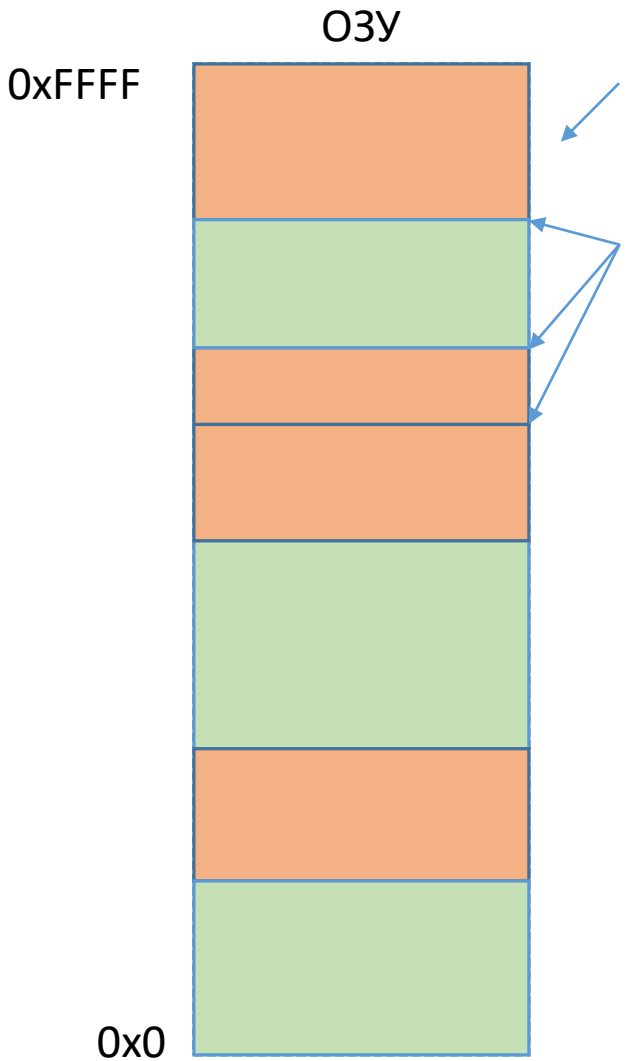


Неучтённые области памяти – это перерасход памяти, поэтому хотелось бы, чтобы их не было

В общем, основные условия корректности можно сформулировать так:

1. Блоки не должны заползать друг на друга
2. Между блоками не должно быть неучтённых областей памяти

Валидность структуры



Такую структуру памяти будем считать правильной (валидной, корректной)

Все блоки идут “встык” и между ними нет “дырок”

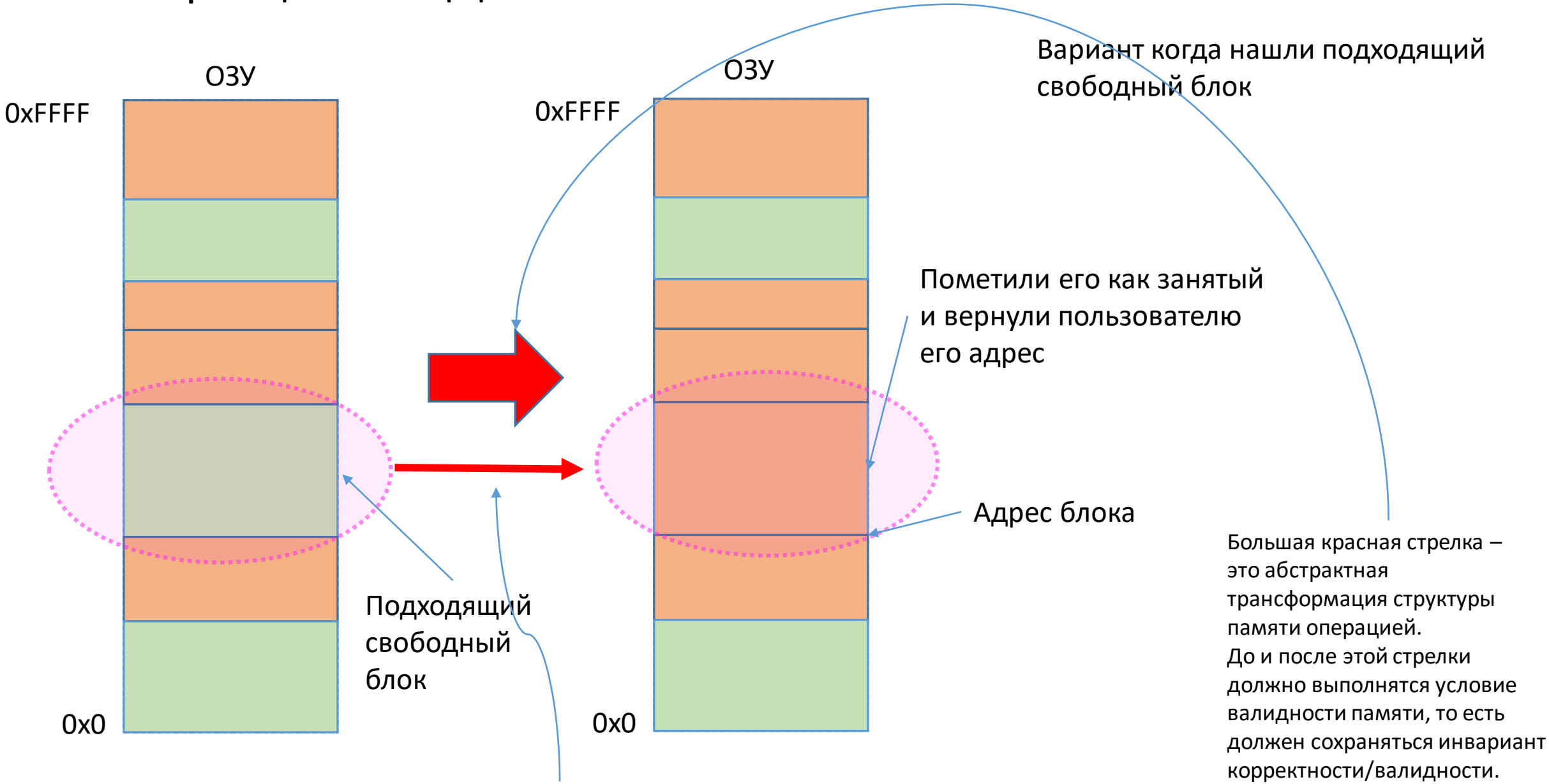
Все операции над памятью должны сохранять валидность стурктуры памяти

Такие свойства, которые должны быть всегда истинными, называют “инвариантами”, то есть неизменными.

Основных операций у нас две: выделить блок памяти заданного размера и освободить блок памяти (пометить как свободный)

Алгоритм поиска наиболее подходящего свободного блока при выделении мы пока не рассматриваем, пока мы рассматриваем абстрактные операции

Операция выделения блока памяти

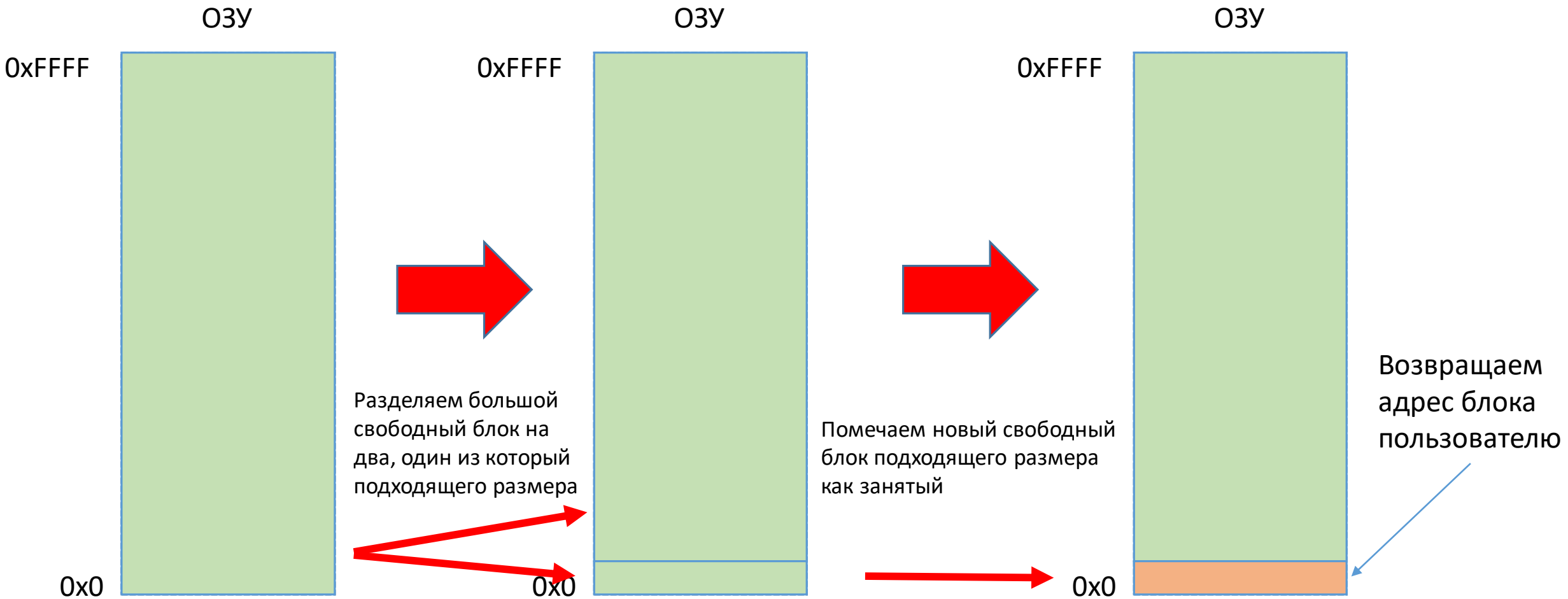


Маленькая стрелочка – это трансформация блока и соответвтующих служебных структур (заголовка и тд)

Операция выделения блока памяти

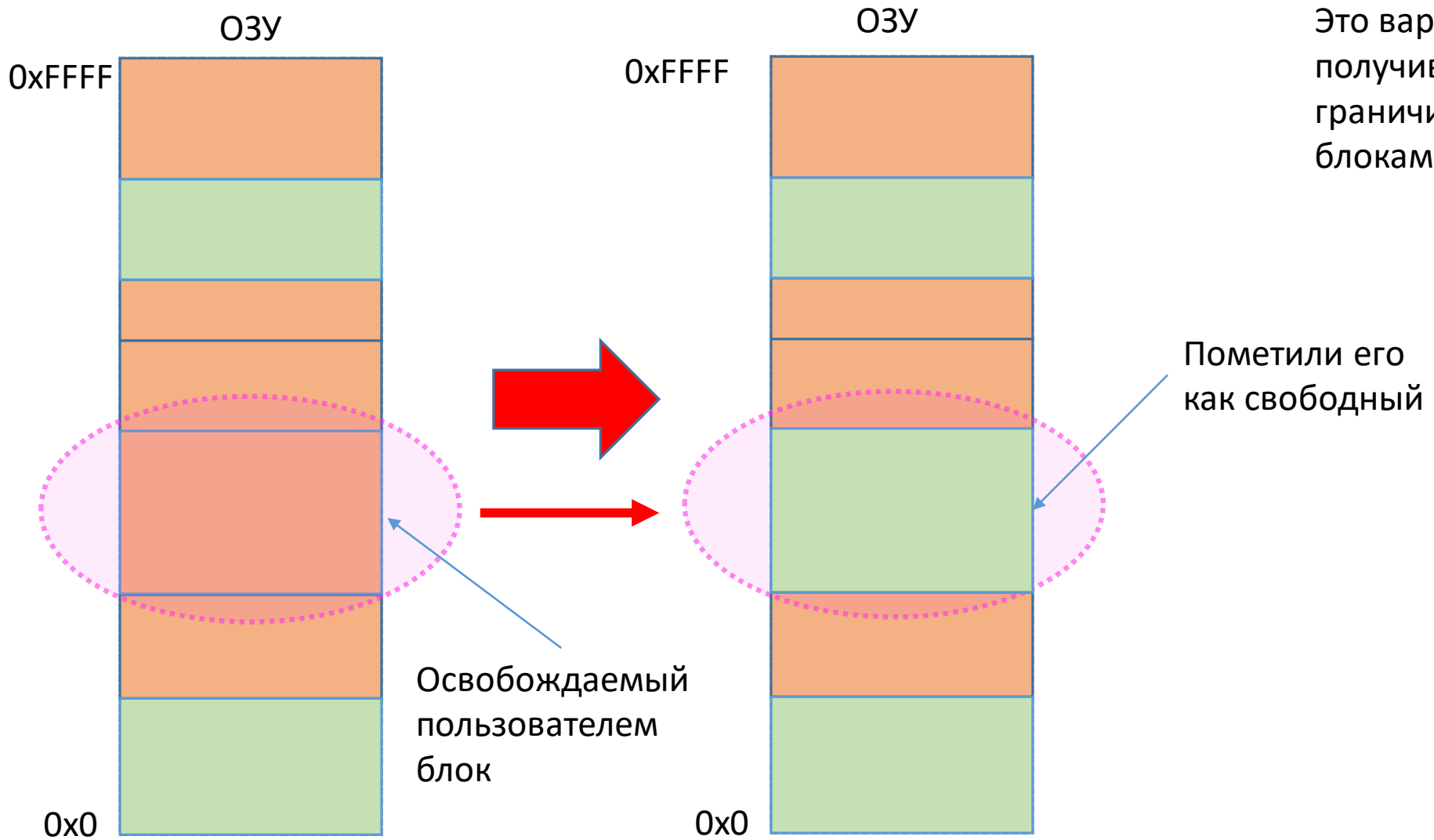
Запросили 32 байта, а свободный блок – это вся память. Что делать?

Ответ – разрезать большой свободный блок на кусочки.



Операция освобождения блока памяти

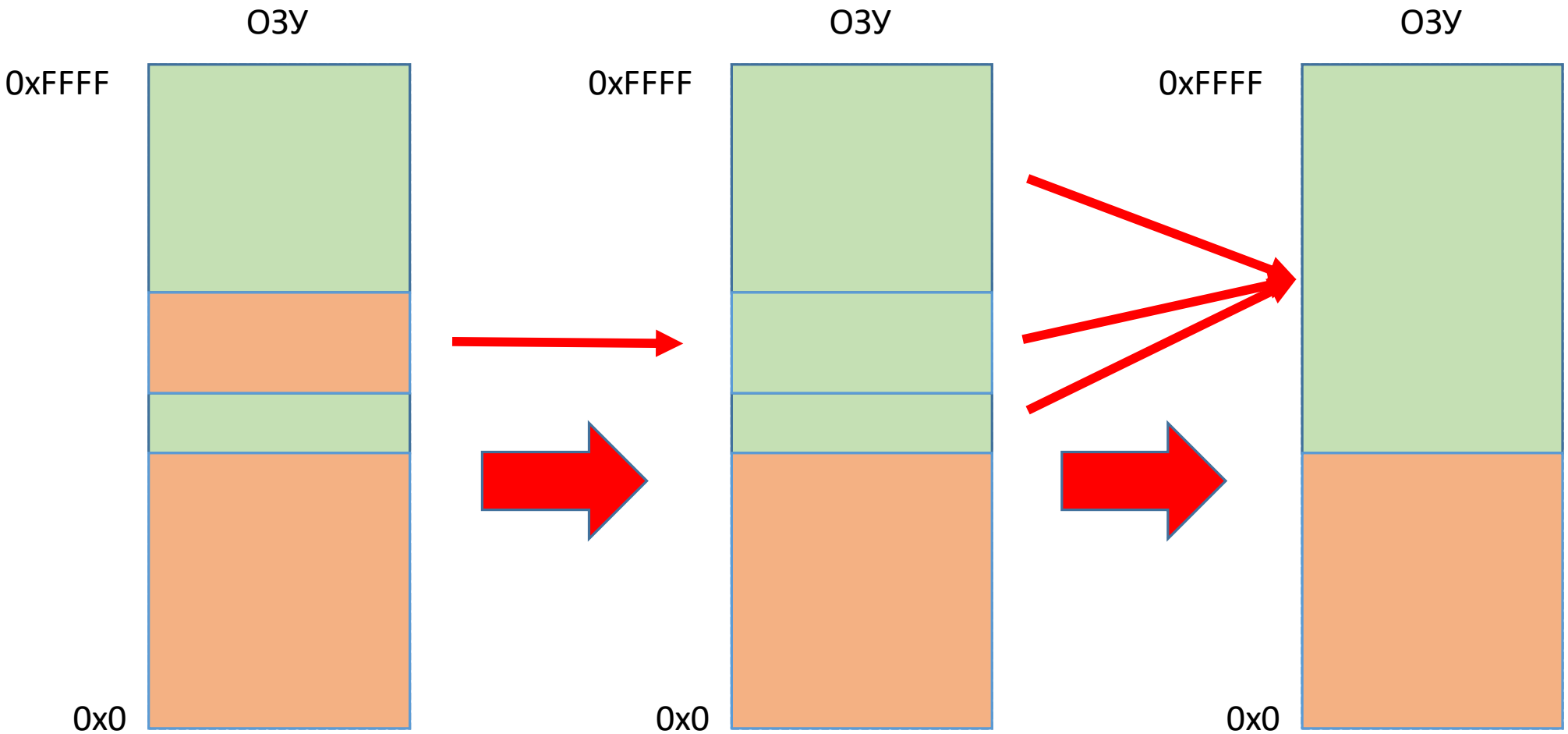
В смысле, что блок освобождает пользователь и отдаёт его нам обратно



Это вариант операции, когда вновь получившийся свободный блок не граничит с другими свободными блоками

Операция освобождения блока

Если вдруг свободные блоки оказались рядом?
Просто объединим их в один большой.



Summary

- У нас две освновных операции: выделение блока и освобождение
- У каждой операции есть две разновидности: нужно делить блок или нет при выделении блока, нужно объединять смежные блоки или нет при освобождении блока
- Все операции, изменяющие структуру памяти должны сохранять инвариант валидности структуры памяти. Инвариант – это предикат корректности ВСЕЙ структуры памяти, именно поэтому были нарисованы большие красные стрелки, для обозначения абстрактной трансформации всей структуры памяти, и карты памяти с блоками целиком.
- На данный момент мы рассматриваем самый простой вариант: работа с памятью однопоточная, абстрагировались от служебных структур менеджера памяти, не рассматриваем никаких выравниываний адресов и размеров и тд.

To be continued…

- На следующей лекции мы формализуем описание структуры памяти в Alloy
- Формализуем свойство корректности структуры памяти (инвариант)
- Формализуем операции над структурой памяти
- Проверим сохранение инварианта при выполнении операций
- Посмотрим, как отлаживать спецификации на Alloy, как читать и понимать контр-примеры, которые Alloy Analyzer будет нам выдавать в случае ошибок в операциях (нарушение инварианта, например)
- Обсудим вопросы, как отразить инварианты и предикаты из модели в коде

Рекомендуемое домашнее задание

- Для небольшого ознакомления с Alloy, рекомендую:
  - Поставить Alloy Analyzer: https://github.com/AlloyTools/org.alloytools.alloy/releases
  - Посмотреть небольшое введение https://alloytools.org/tutorials/day-course/ и небольшое руководство https://alloytools.org/quickguide/, если ничего не понятно, не пугайтесь, это нормально ☺ Главное, чтобы появились хотя бы небольшие навыки по работе с IDE Alloy Analyzer.
  - Неплохие слайды про Alloy можно посмотреть тут: http://www.cse.msu.edu/~cse814/Lectures/ (msu – это Мичиган State University, если что ☺)
  - Вот тут: http://homepage.cs.uiowa.edu/~tinelli/classes/181/Fall14/lectures.shtml хорошее введение в множества, сигнатуры, атомы Alloy и тд.
  - К сожалению, курс у нас очень сжатый и многие вещи детально рассмотреть на лекциях просто нет физической возможности.
  - Поэтому, чтобы было хотя бы общее понимание того, о чём речь пойдёт на следующих лекциях, большая просьба хотя бы бегло просмотреть слайды про Alloy из вышеприведённых ссылок. Чтобы по крайней мере понимать основные термины: отношение, атом, сигнатура, предикат, замыкание, join и тд.