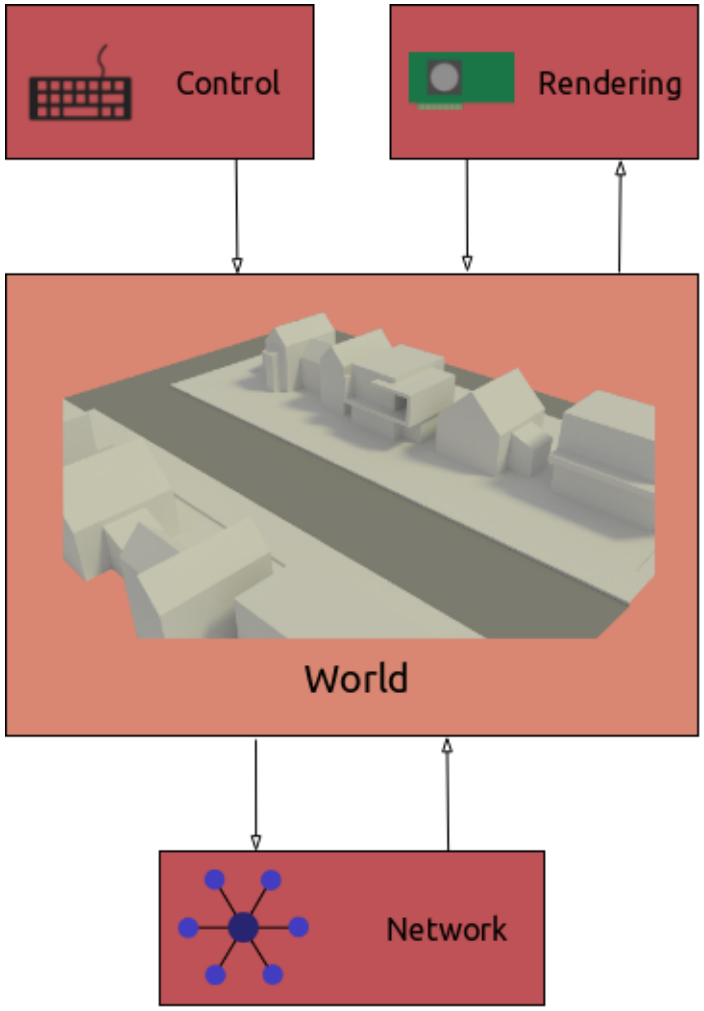
Механизм синхронного доступа к совместно используемым данным

Шляков Антон Константинович

trionprog@gmail.com

10.04.2018г.

Проблема



Необходимо разработать механизм, возволяющий параллельно независимо друг от друга модифицировать здания

Цели и задачи

Характер обработки данных

- Транзакции выполняются долго (до 0.5с)
- При обработке данных затрагивается множество объектов
- Необходимо обеспечить целостность данных
- Возможны взаимные блокировки
- Очень частые обращения к данным

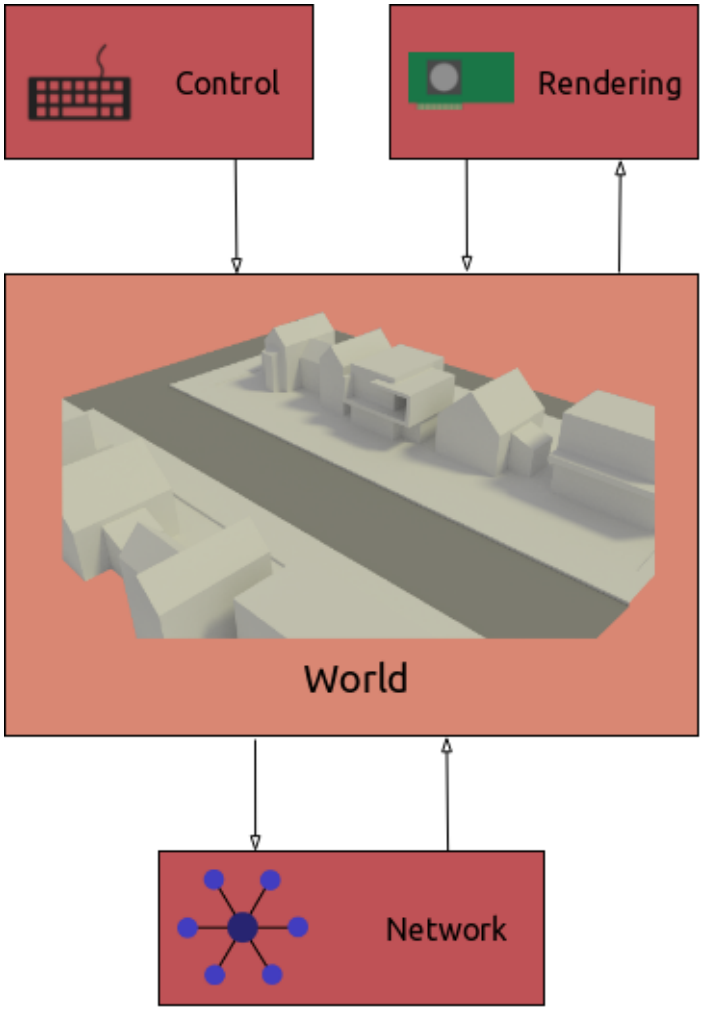
Характер вычислительной сети

- Распределённые обработка и хранение данных
- Высокая надёжность
- Очень критична производительность

Задача

Необходимо разработать механизм, возволяющий параллельно независимо друг от друга модифицировать здания

Недостатки существующих механизмов



- Обернуть мир мутексом / доступ с помощью сообщений
  - ⛔ Производительность ограничена одним ядром
  - ⛔ В случае сообщений невозможно редактировать несколько объектов одновременно
- Обернуть каждый объект/поле мутексом
  - ⛔ В случае ошибки часть изменений будет применена, а часть нет, а значит, нарушется целостность
- STM
  - ⛔ Если коллизии решаются перед фиксацией изменений, при большом потоке обращений транзакции будут отменяться, а значит, будет плохая производительность

  Мои предложения ?

Виды объектов

Vector<T> — сильная связь

Поля:

- buffer: *T
- length: usize
- reserved: usize

Ограничения:

1) length<reserved
2) buffer=0 if reserved=0
3) buffer.length=reserved

Ограничения работают всегда для всех полей объекта.

Building — слабая связь

Поля:

- walls:Vector<Wall>
- floors:Vector<Floor>
- furniture:Vector<Furniture>
- doors:Vector<Door>
- players:Vector<Player>

Ограничения:

1) Игроки не могут пробегать через стены
2) Игроки могут бегать только по полу
3) Мебель стоит на полу
4) Игроки могут открывать двери
5) Двери располагаются в стенах

Ограничения работают только для некоторых полей. Некоторые поля можно изменять независимо от других, главное сохранить целостность объекта.

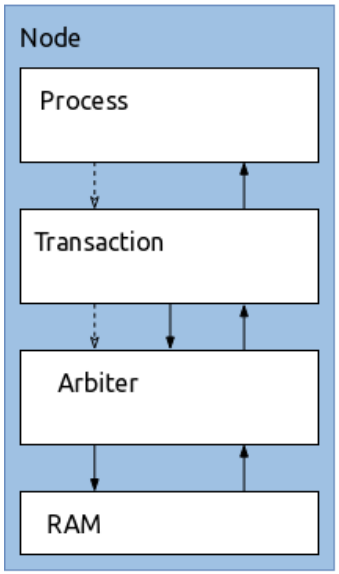
Мои предложения

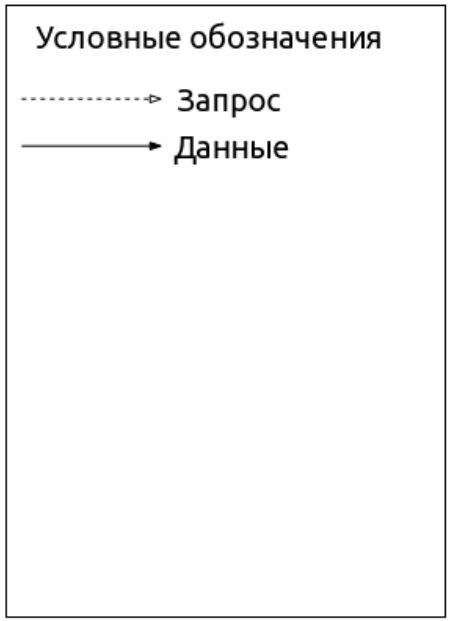
- Обработка данных с помощью транзакций
  - В случае возникновения ошибки транзакция прерывается, а изменения не фиксируются
  - В случае возникновения взаимной блокировки транзакция прерывается, и может быть повторена
  - В случае коллизии, транзакция с большим приоритетом может прервать транзакцию с меньшим
- Применение специального вида блокировок (Арбитр)
  - Блокировка полей объекта по-отдельности
  - В случае коллизии транзакции с меньшим приоритетом ждут окончания блокировки поля транзакцией с большим
  - Блокировака нескольких полей согласно таблицы режимов доступа
  - При окончании транзакции поля разблокируются
- Интеграция с БД и наличие кэша

Имя поля	Участвует?	Режим доступа
bounding_box	1	read
players	0	
walls	1	write
doors	0	
floors	1	read

Организация хранения и обработки данных

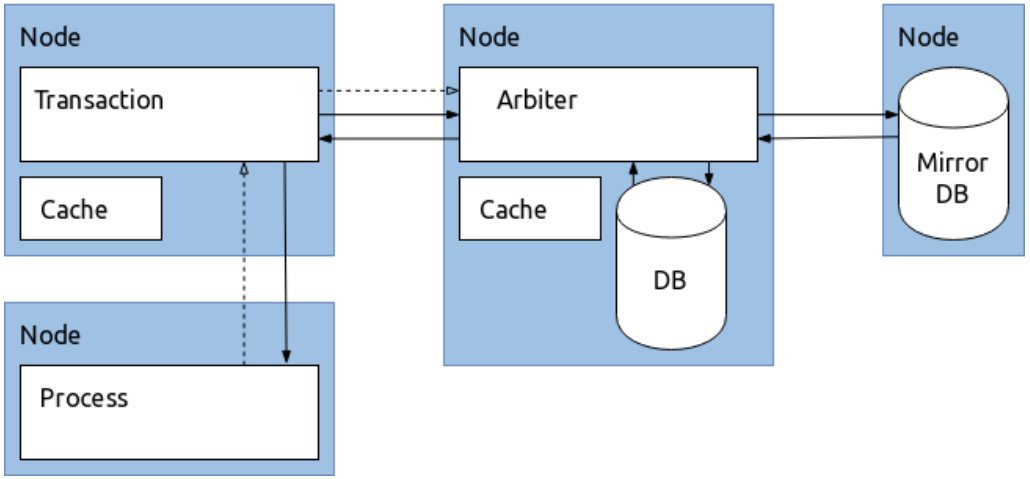
Простая организация





В данной работе рассмотрена простая организация

Сложная организация



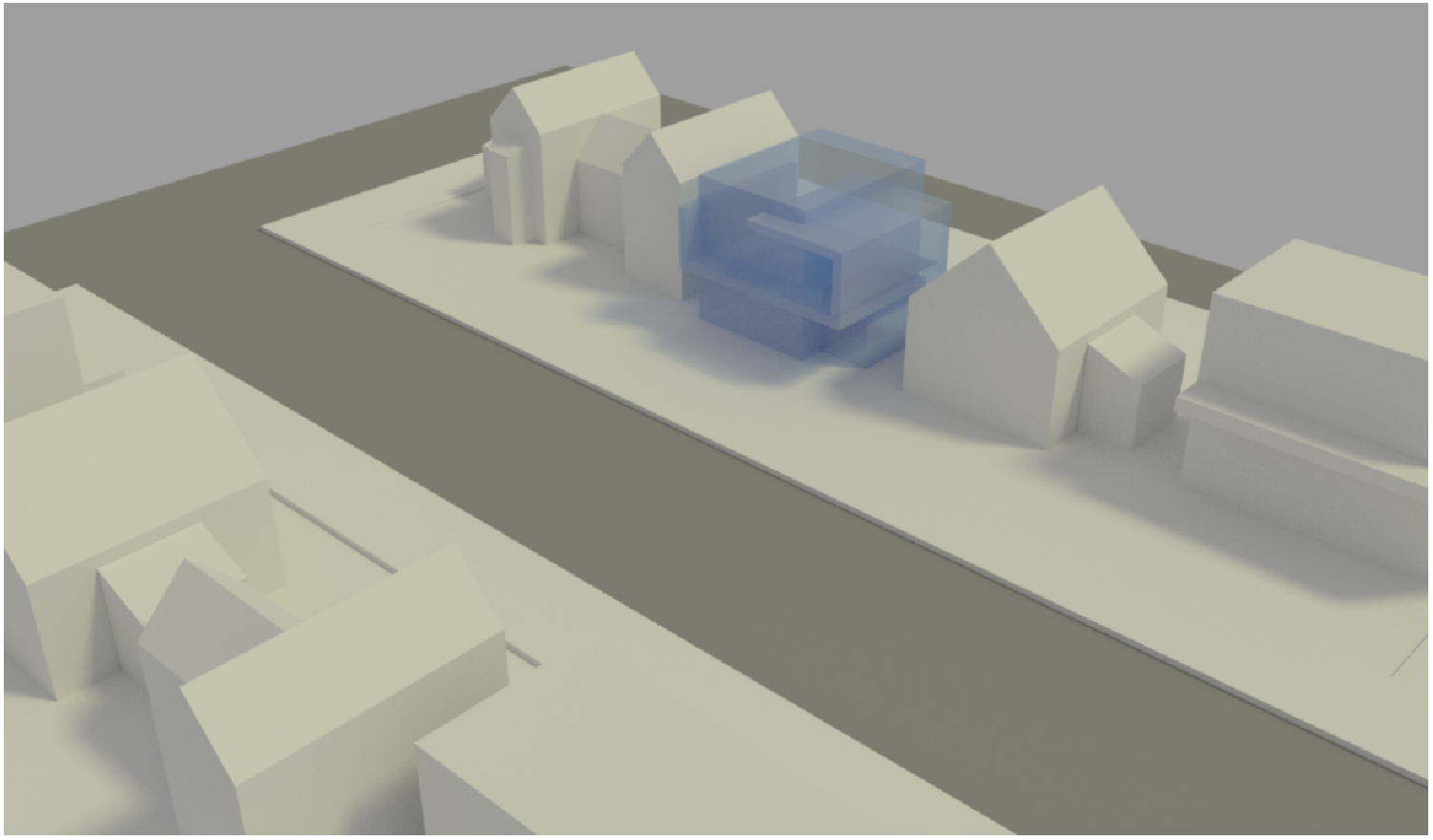
Транзакцию можно выполнить на любом узле согласно балансировщику нагрузки

Ссылки и время жизни объектов

Используются умные указатели

- Подразделяются на сильные и слабые
- Имеют счётчик ссылок
- Объекты создаются и уничтожаются только транзакцией
- В случае завершения транзакции(например с ошибкой), объекты могут быть уничтожены

«Призрачное» редактирование

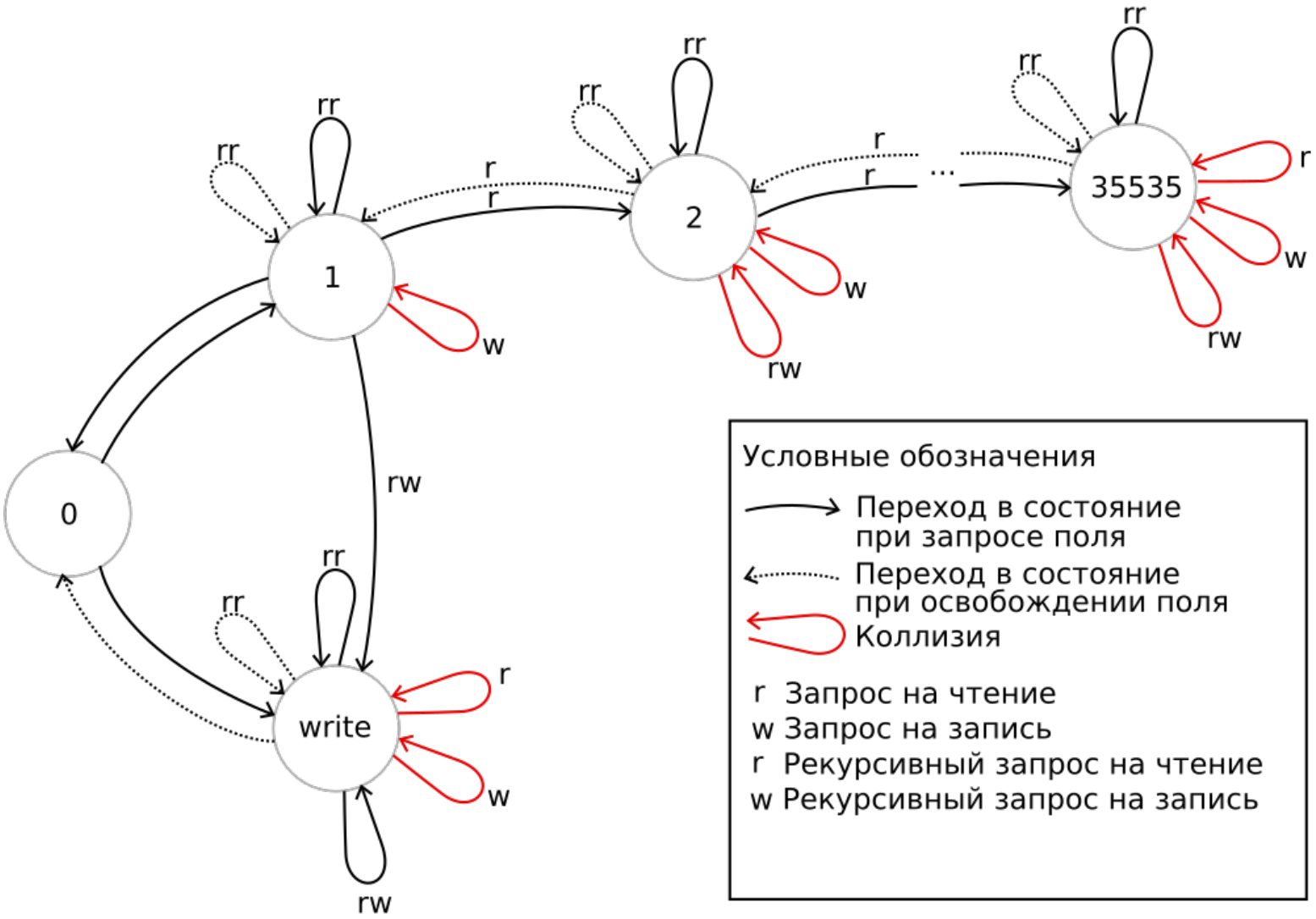


- Применяется, если транзакция не воздействует на мир за пределами объекта
- Позволяет не блокировать списки объектов
- Если объект был удалён из списка, он будет жить до тех пор, пока жива транзакция

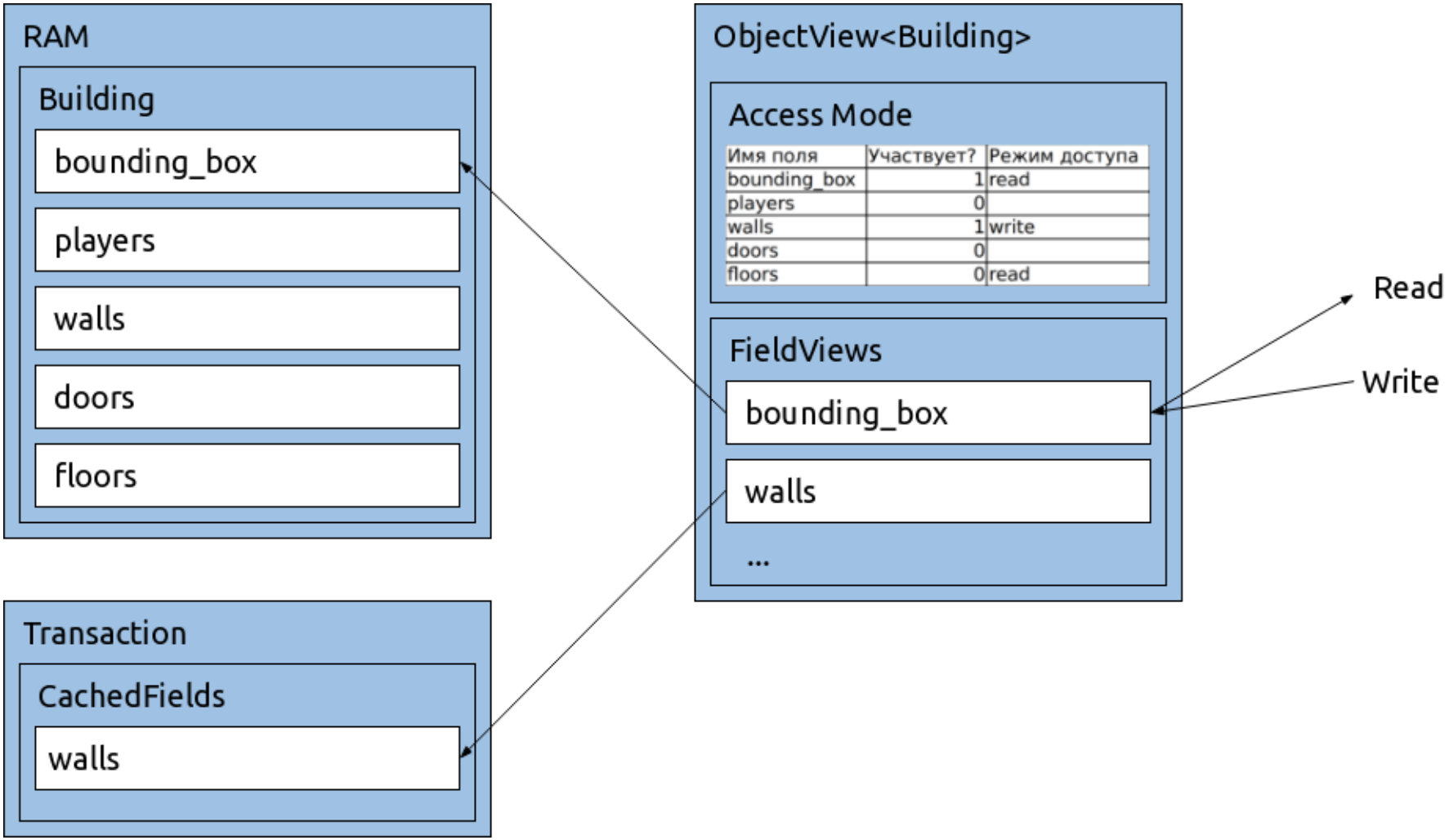
Таблица состояний полей/режимов доступа

Имя поля	Участвует?	Режим доступа
bounding_box	1	read
players	0	
walls	1	write
doors	0	
floors	1	read

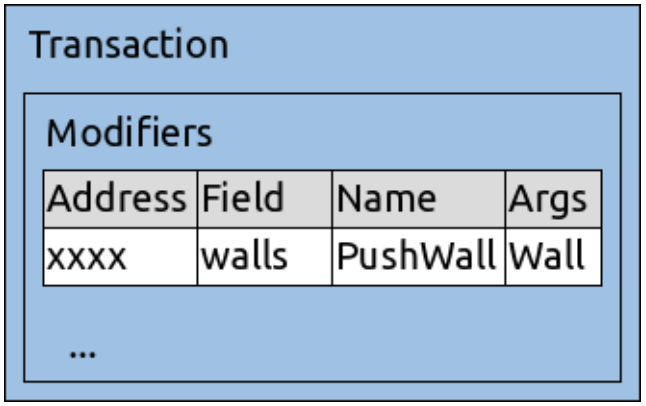
Граф состояний поля



Представление(View)

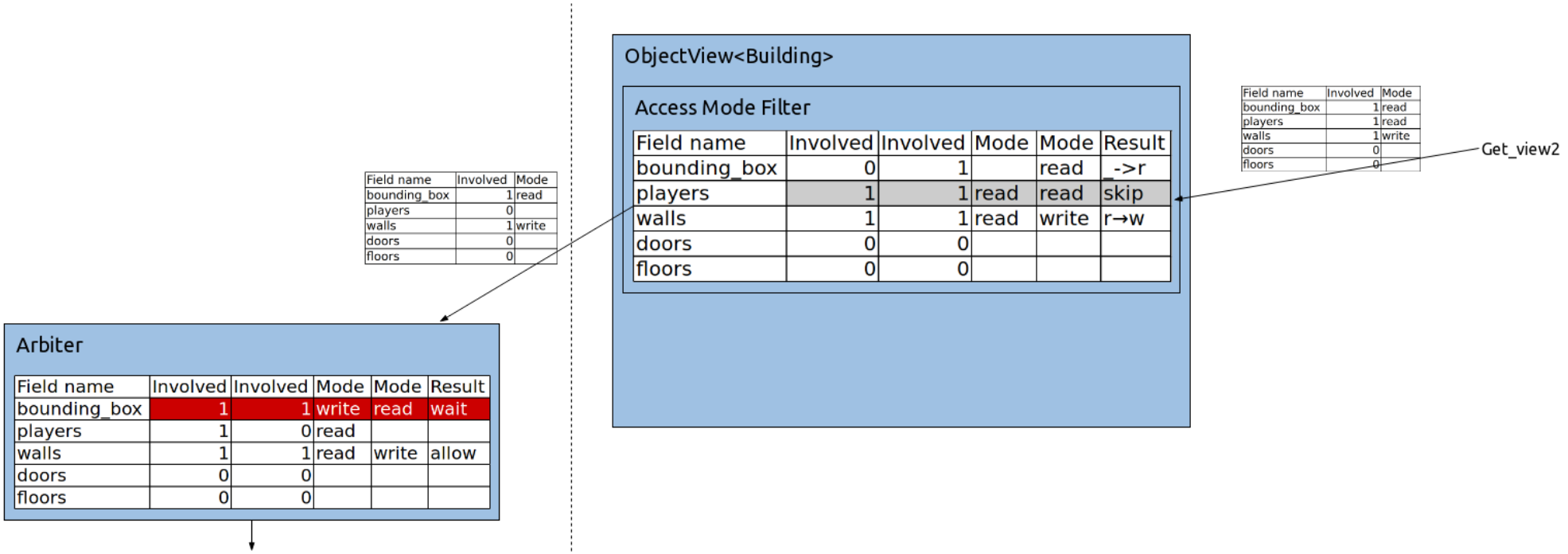


Модификатор

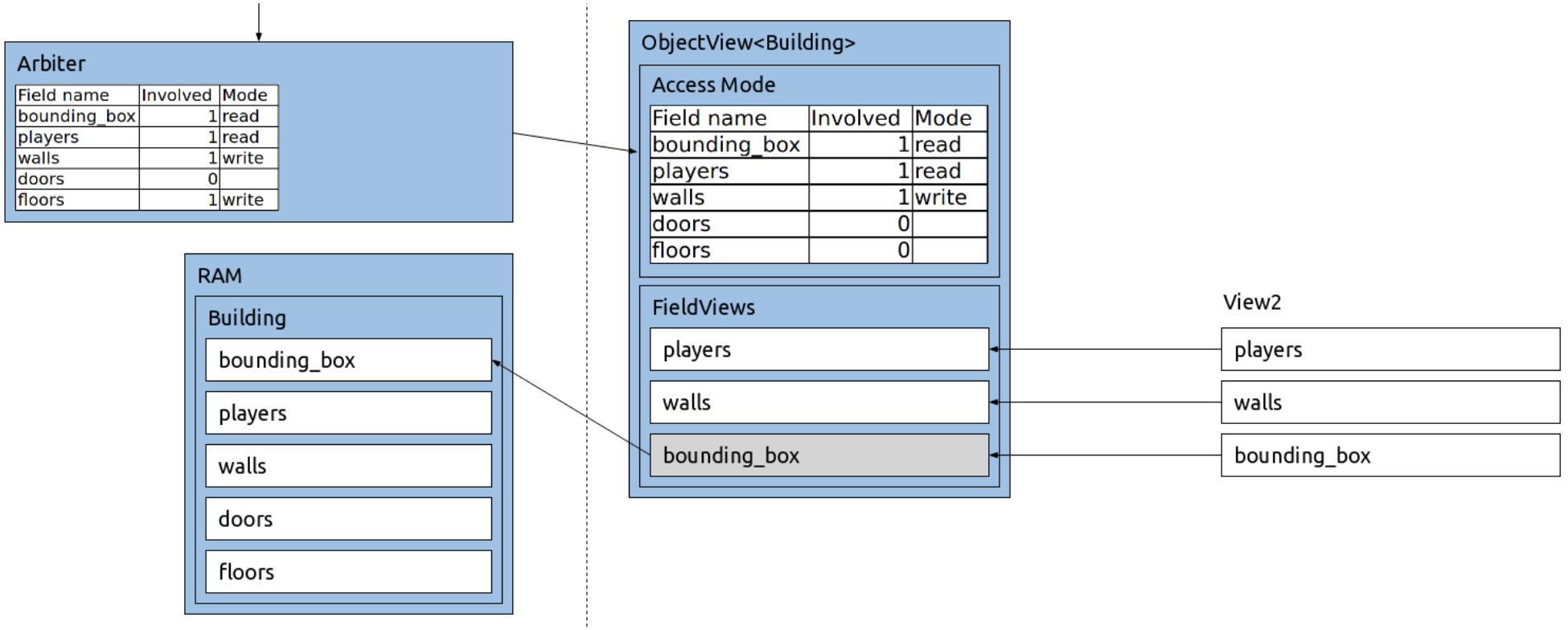


- Фактически удалённый вызов процедуры, напоминает метод
- Содержит адрес объекта, адрес поля, аргументы
- Применяется только в случае успешного завершения транзакции
- Может быть временно применён на виртуальное поле, если это поле требуется прочесть

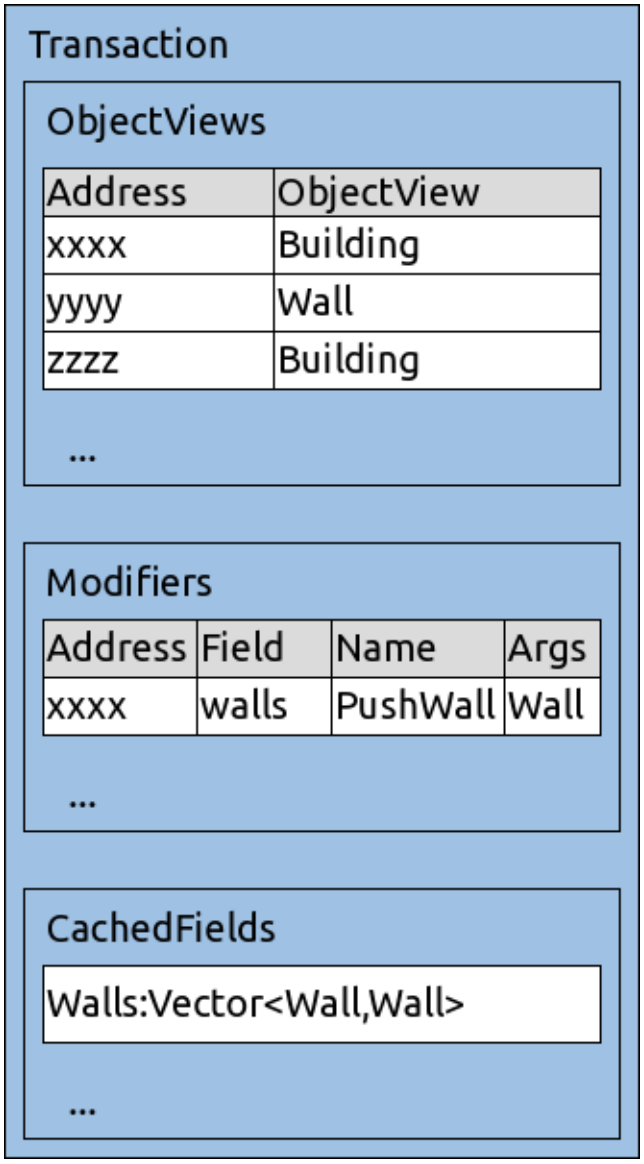
Запрос. Часть 1



Запрос. Часть 2



Транзакция



- Имеет множество буферов, хранящих модификаторы, поля и тд.
- Представления объектов хранятся в хэш-таблице, обсепречивающей поиск по адресу объекта

Псевдокод

```
let buildings=Transaction{
      let buildings=get_world_view1(&world, transaction);//(read)

      for building in buildings.get().iter() {
         let bounding_box=get_building_view1(building,transaction);//(read)

         if bounding_box.get().collide(Point::new(5.0,5.0)) {
             thread::sleep_ms(10); //долгая операция
             buildings1.push(building.clone());
         }
      }
}


for building in buildings.iter() {
    process_building(building,i);
}

fn process_building(building:&ObjectReference<Building>) {
    Transaction{
       let walls=get_building_view3(building,transaction);

       for wall in walls.get().iter() {
          let (bounding_box, health)=get_wall_view1(wall,transaction);//(write,write)

          if bounding_box.get().collide(Point::new(5.0,5.0)) {
             thread::sleep_ms(100); //долгая операция
             health.add_modifer(Box::new(SetHealthModifier{health:4.0})); //модифицируем
          }
       }
    }
}
```

Итоги

- Реализован действующий прототип: https://github.com/TrionProg/transaction_test

Спасибо за внимание

Шляков Антон Константинович

trionprog@gmail.com

10.04.2018г.