



HEX

Research Team

# О нас

Пилотные проекты в разное время для ряда компаний

- Erlang
- OCaml
- Haskell
- C (embedded)
- PostgreSQL



ZYXEL



# Outline

- Переход от “анемичной” к “богатой” модели предметной области
- Хранение данных
- Навигация по данным (вложенным ADT)
- Модификация данных
- Загрузка данных в память (в т.ч. ленивая)
- Связи между данными
- Версионирование данных
- Полученные результаты
- Как все должно быть на самом деле

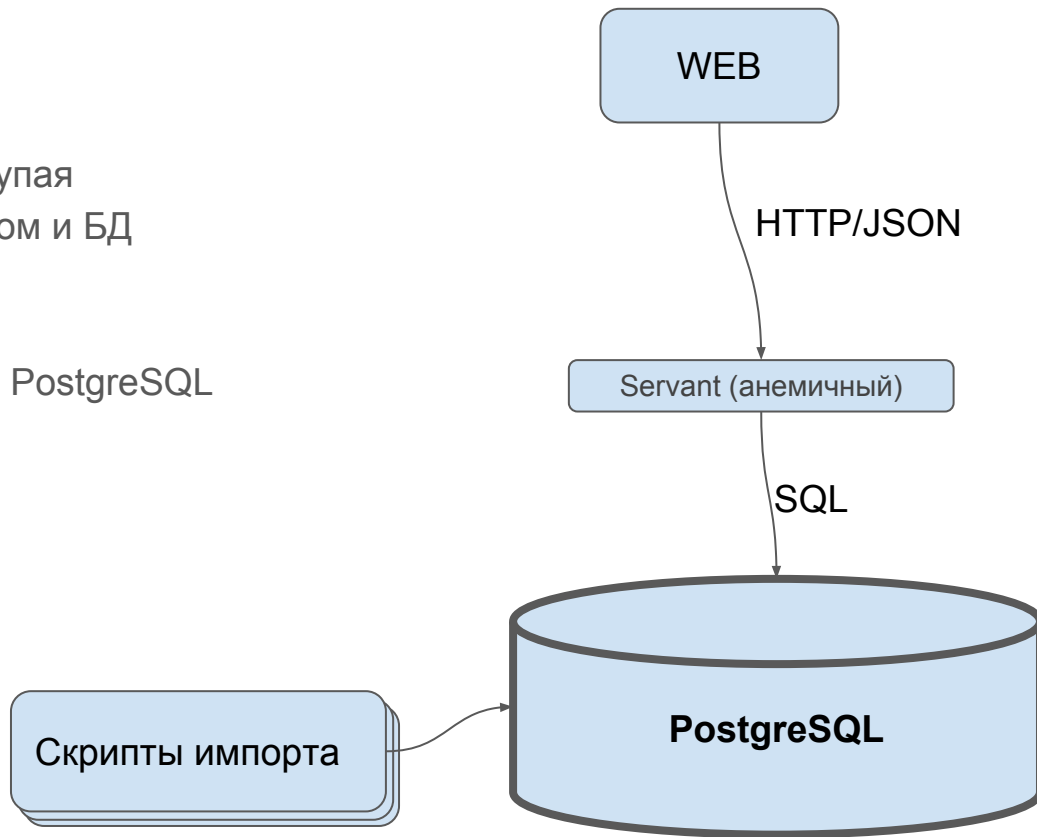
# О проекте

- Туроператор
  - Регионы, отели, авиабилеты, рейсы
  - Поиск
  - Вычисления
  - Большие объемы
  - Плохо структурированные данные
    - $10K * 100 * 365 = 365M$
  - Медленные API агрегаторов
    - 16 часов на импорт - не предел
  - Невозможно построить полную модель
  - Был написан на LAMP
    - Разработчик бежал исчез
    - Поддержка невозможна



# Как было

- “Анемичная” модель
  - Сервер - относительно тупая прослойка между клиентом и БД
  - Много логики в БД
    - СЛИШКОМ МНОГО
  - Максимум возможностей PostgreSQL
    - Функции
    - JSON/XML
    - Materialized view
    - “Типы”
    - Массивы



DISCLAIMER (ответы на неизбежные вопросы)

# DISCLAIMER (ответы на неизбежные вопросы)

- Умеем в PostgreSQL (давно)
  - GPS мониторинг
    - 7 лет
    - 30М записей в сутки
    - Шардинг
    - HA (Pacemaker)
    - Очень тяжелые отчеты
    - PostGIS (гео-данные)
- Знаем про
  - apgdiff, sqitch
  - Redis, Mongo The **WEBSCALE!**, Clickhouse, Vertica, Cassandra, Tarantool, leveldb, acid-state и 9000 прочих
- А вот если данные не поместятся в RAM....



# Проблема

- РСУБД
  - Высокая стоимость внесения изменений
    - Миграции
  - Сложность командной разработки
    - Желателен DBA
  - Сложность автоматического выкатывания (DevOps!)
    - Желательно разделение сред (Dev, Test, UAT, Prod)
  - Слабая динамическая типизация (неизбежные падения в runtime)
  - Проблемы с производительностью на неудобных данных (ряды)
  - Оверхед (отображение данных, преобразования)



# Проблема

- РСУБД
  - Невыразительные языки
    - Склонные к генерации ошибок

## Нормальный язык

```
1 capacityReqNorm :: CapacityReq -> CapacityReq
2 capacityReqNorm (CapacityReq ages) = CapacityReq $ reverse $ L.sort ages
3
4 capacityNorm :: Capacity -> Capacity
5 capacityNorm (Capacity units) = Capacity $ L.sortBy (comparing capAgeMax) units
6
7 capacityMatch :: CapacityReq -> Capacity -> Bool
8 capacityMatch req' cap' = null $ go rs ss
9   where
10     (CapacityReq rs) = capacityReqNorm req'
11     (Capacity ss) = capacityNorm cap'
12
13 go :: [Age] -> [CapacityUnit] -> [Age]
14
15 go (a:as) (CapacityUnit _ am : cs) | a < am = go as cs
16                                     | otherwise = go (a:as) cs
17
18 go [] _ = []
19 go as [] = as
```

## PL/pgSQL

```
1 CREATE FUNCTION check_accomodation( adults5check smallint
2                                     , ages2check smallint[]
3                                     , adults given smallint
4                                     , ages_given numrange[]) RETURNS boolean
5 AS
6 LANGUAGE plpgsql IMMUTABLE
7 AS $$
8 DECLARE
9   i int; j int;
10  ages_used boolean[];
11  n_ages2check int := array_length(ages2check, 1);
12  n_ages_given int := array_length(ages_given, 1);
13 BEGIN
14 IF adults2check > adults_given THEN
15   RETURN FALSE;
16 END IF;
17 IF n_ages2check > n_ages_given THEN
18   RETURN FALSE;
19 END IF;
20 IF n_ages2check IS NULL THEN
21   RETURN TRUE;
22 END IF;
23 IF n_ages_given IS NULL THEN
24   RETURN FALSE;
25 END IF;
26
27 ages_given := (
28   WITH ages_given AS (SELECT ages FROM unnest(ages_given) AS ages)
29   SELECT array_agg(ages ORDER BY upper(ages)) FROM ages_given
30 );
31 ages2check := (
32   WITH ages2check AS (SELECT ages FROM unnest(ages2check) AS ages)
33   SELECT array_agg(ages ORDER BY ages) FROM ages2check
34 );
35
36 ages_used := array_fill(FALSE, ARRAY[n_ages_given]);
37
38 <<TO_NEXT_AGE>>
39 FOR i IN 1 .. n_ages2check LOOP
40   FOR j IN 1 .. n_ages_given LOOP
41     CONTINUE WHEN ages_used[j];
42
43     IF ages2check[i]::numeric <@ ages_given[j] THEN
44       ages_used[j] := TRUE;
45       CONTINUE TO_NEXT_AGE;
46     END IF;
47   END LOOP;
48
49   RETURN FALSE;
50 END LOOP TO_NEXT_AGE;
51
52 RETURN TRUE;
53 END;
54 $$;
```

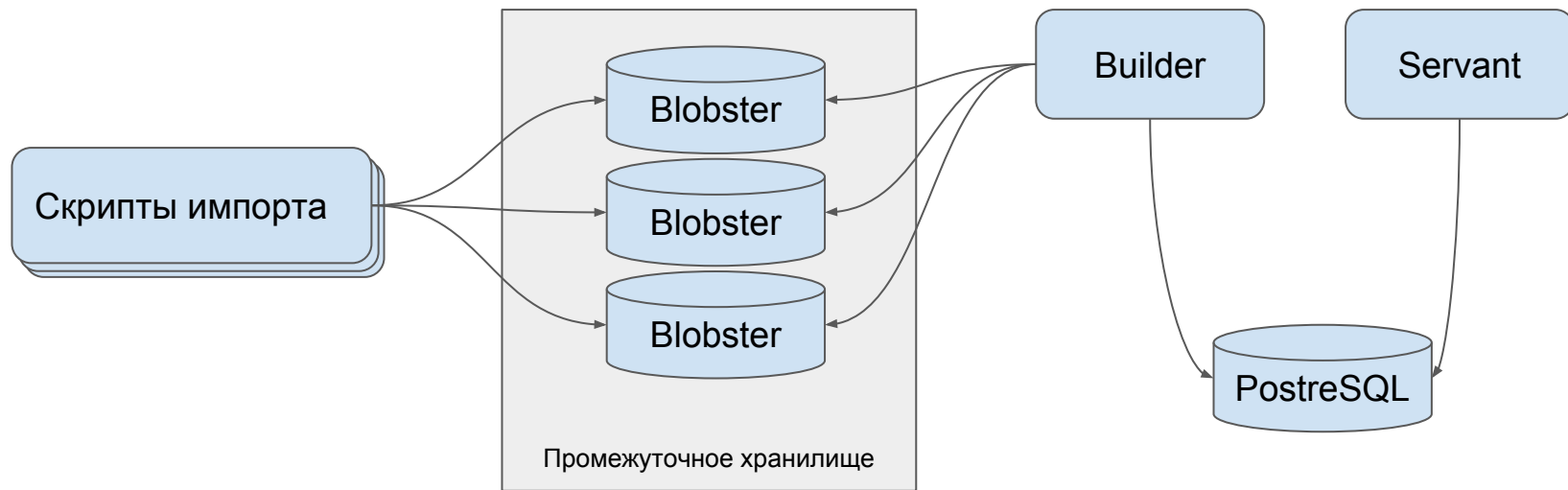
# Что решили сделать

- Переход от анемичной модели к богатой модели
- Представление данных в виде ADT
- Бизнес-логика строго на Haskell
- Хранилище импортированных данных в виде сериализованных ADT

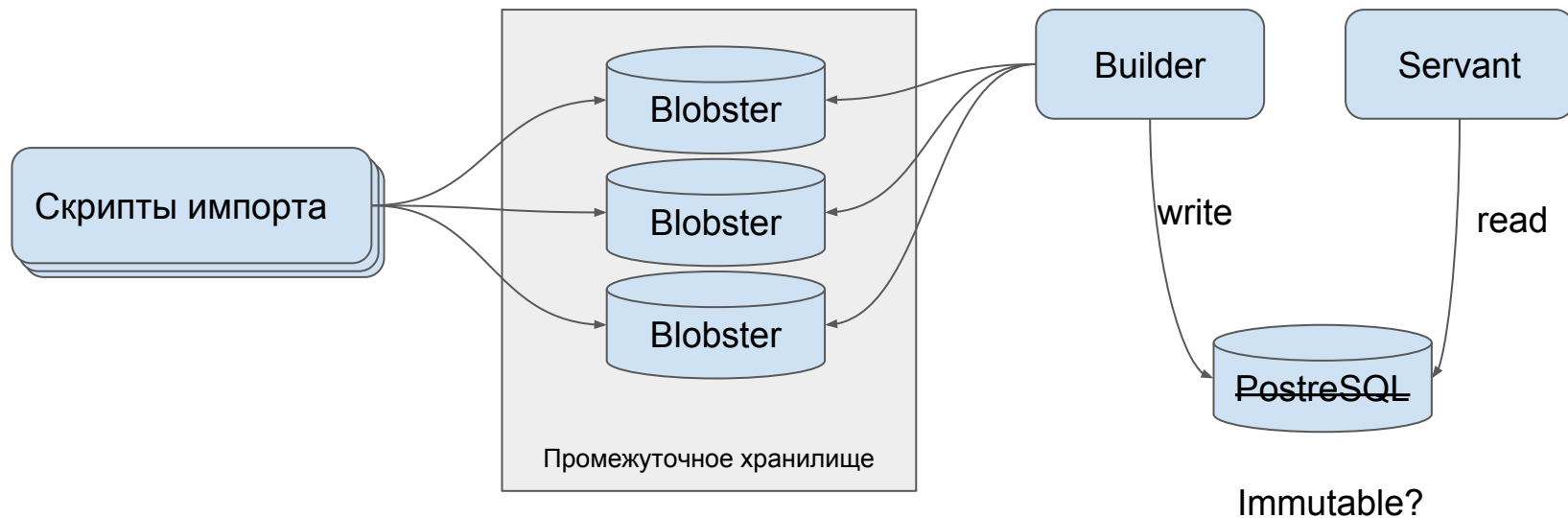
Искрометная шутка про анемичную и богатую модель удалена по соображениям политкорректности



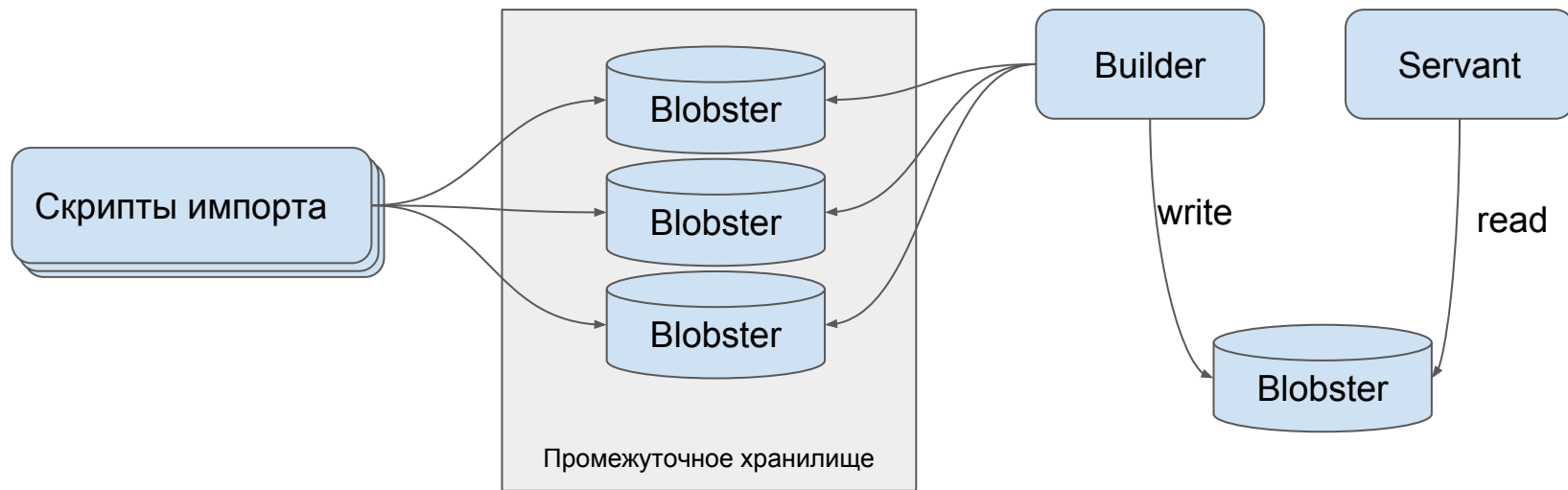
# Что получилось



# Что получилось



# Что получилось



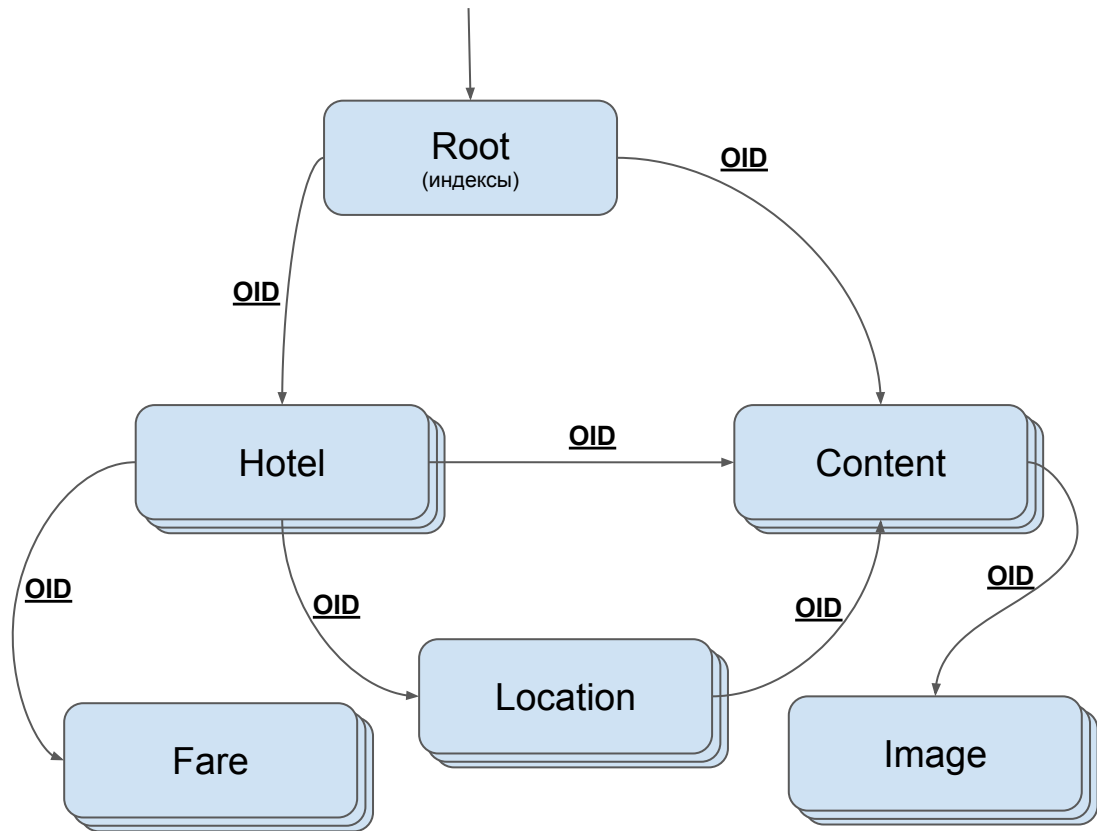
# Что получилось: хранение

- Blobster
  - Написали сами (196 строк)
  - Файловое хранилище
  - Сериализованные в *cereal* значения
  - Структура как в *git*
  - CAS (blobs)
    - Иммутабельные объекты, изменение объекта меняет ключ
  - Indexed (refs)
    - Ссылка + CAS объект



# Что получилось: представление данных

- “Чистые” ADT для представления бизнес-объектов
- Containers: Map, Set
- Root-объект
- OID (типично SHA-х<sup>(\*)</sup> от сериализованного объекта)



# Что получилось: представление данных

```
data Root = Root
{ hotels      :: Map OID HotelEntry
, locationsAll :: Map OID LocationEntry
, miscContent :: [Content]
}
```

```
data Hotel = Hotel
{ hoOID      :: OID
, hoSupplierID :: SupplierID
, hoWebLink  :: Maybe String
, hoClass    :: HotelClass
, hoType     :: Map Language Name
, hoFeatures :: Map OID FeatureDesc
, hoLocation :: MetaLocation
, hoDescription :: Map Language Content
, hoRooms     :: Map RoomId Room
, hoCheckin   :: Checkin
, hoCheckout  :: Maybe Checkout
, hoOffers    :: Set HotelIOffer
}
```

```
data HotelStars = S0 | S1 | S2 | S3 | S4 | S5 | S5Plus
```

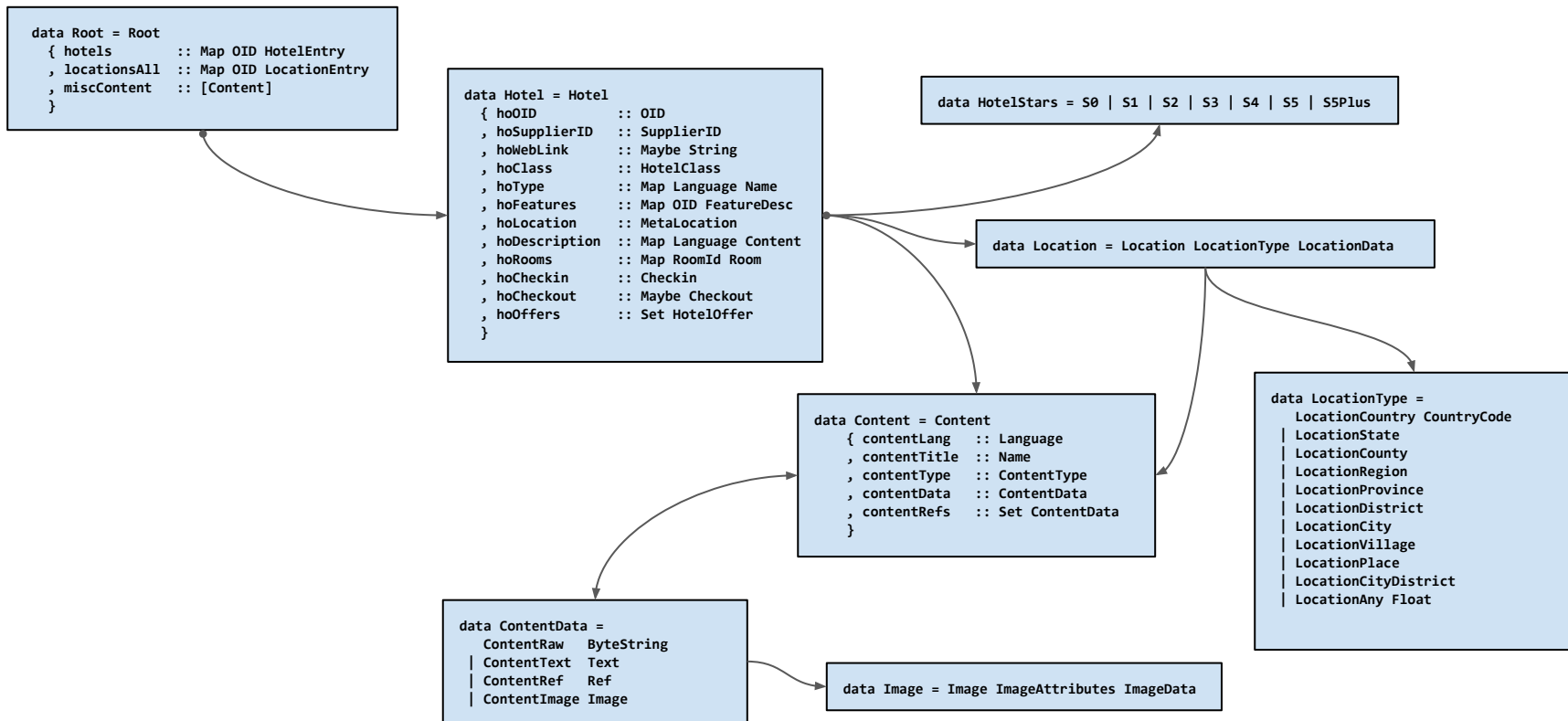
```
data Location = Location LocationType LocationData
```

```
data Content = Content
{ contentLang  :: Language
, contentTitle :: Name
, contentType  :: ContentType
, contentData  :: ContentData
, contentRefs  :: Set ContentData
}
```

```
data LocationType =
  LocationCountry CountryCode
| LocationState
| LocationCounty
| LocationRegion
| LocationProvince
| LocationDistrict
| LocationCity
| LocationVillage
| LocationPlace
| LocationCityDistrict
| LocationAny Float
```

```
data ContentData =
  ContentRaw ByteString
| ContentText Text
| ContentRef Ref
| ContentImage Image
```

```
data Image = Image ImageAttributes ImageData
```





# Что получилось: навигация по данным

## Boilerplate Removal with Uniplate

<https://github.com/ndmitchell/uniplate>

```
{-# LANGUAGE DeriveDataTypeable #-}
module Expr where
import Data.Data
import Data.Generics.Uniplate.Data

data Expr = Val Int
          | Add Expr Expr
          | Sub Expr Expr
          | Div Expr Expr
          | Mul Expr Expr
          | Neg Expr
          deriving (Show, Eq, Data, Typeable)
```

### Finding the constant values

```
universe :: Uniplate on => on -> [on]

constants :: Expr -> [Int]
constants x = nub [y | Val y <- universe x]
```

### Basic optimisation

```
transform :: Uniplate on => (on -> on) -> on -> on

optimise :: Expr -> Expr
optimise = transform f
  where f (Neg (Val i)) = Val (negate i)
        f x = x
```

# Что получилось: навигация по данным

- uniplate
  - **universeBi**
  - List comprehensions
  - Pattern matching

```
getImages :: Data a => a -> [Image]
getImages w = L.nub [ i | i@(Image _ _) <- L.nub (universeBi w) ]

getImagesFor :: Data a => Maybe Language -> a -> [Image]
getImagesFor ml wut = L.nub (foldMap getImages cs)
  where
    cs = [ c | c@(Content { contentLang = cl }) <- universeBi wut
          , fromMaybe cl ml == cl ]

getHotelDesc :: WebKey -> Language -> FrontPageEditM Text
getHotelDesc wk lang = withRoot $ do
  db <- asks db
  lookupHotel wk >>= \h' -> do
    h <- summonContent db h'
    let content = M.lookup lang (hoDescription h)
    let s = headMay [ txt | ContentText txt <- universeBi content ]
    return $ fromMaybe "" s
```

# Что получилось: ленивая загрузка

- Virtual Proxy (?)
  - Ленивая загрузка значений
- Materialize
  - ValueOf
  - Абстрагирование хранилища

```
class SafeCopy (ValueOf a) => Materialize a where
  type ValueOf a :: *

  toEntry  :: a -> MaterializedEntry (ValueOf a)
  fromEntry :: MaterializedEntry (ValueOf a) -> Maybe a
```

```
data MaterializedEntry a where

  EntryIndexed :: Maybe ObjectID
               -> Maybe a
               -> MaterializedEntry a

  EntryDirect  :: Maybe BlobID
               -> Maybe a
               -> MaterializedEntry a
```

```
dematerialize :: forall a m . (MonadIO m, Materialize a)
              => Blobster
              -> a
              -> m a
```

```
materialize :: forall a m . (MonadIO m, Materialize a)
            => Blobster
            -> a
            -> m (Either String a)
```

```
data DataObject = DataRef OID | DataValue OID Value
```

```
instance Materialize DataValue where
  type ValueOf DataValue = Value

  toEntry (DataRef _ ) = ...
  toEntry (DataValue _ _) = ...

  fromEntry (EntryDirect _ _) = ...
  fromEntry (EntryIndexed _ _ ) = ...
```

# Что получилось: трансформация

- Ленивая загрузка данных любой вложенности
  - uniplate + materialize

```
data MyVeryBusinessObject = ConstructorOne  
    | ConstructorN DataObject
```

```
doSomethingUseful :: MyServantMonad ()  
doSomethingUseful = do  
    db      <- asks database  
    root    <- asks root  
    loaded  <- transformBiM (summon db) root  
    updated <- transformBiM mutateDataObject loaded  
    updateRoot db updated  
  
where  
    summon db x = materialize db x >>= either die return  
    die _ = throwError err500  
  
mutateDataObject v@(Value{..}) =  
    -- сделать что-нибудь со значением
```

# Что получилось: версионность

## - safecopy

```
safePut :: SafeCopy a => a -> Put a
```

Serialize a data type by first writing out its version tag. This is much simpler than the corresponding `safeGet` since previous versions don't come into play.

```
safeGet :: SafeCopy a => Get a
```

Parse a version tagged data type and then migrate it to the desired type. Any serialized value has been extended by the return type can be parsed.

# Что получилось: версионность

```
{-# version = 1 #-}

data LocationType = ...

data LocationData = LocationData {...}

data Location =   Location OID LocationType LocationData
                  | LocationRef OID

data MetaLocation where

    MetaLocation :: Maybe OID
                  -> Set Location
                  -> MetaLocation

    MetaLocationRef :: OID
                    -> MetaLocation

deriveSafeCopy 1 'base ''LocationType
deriveSafeCopy 1 'base ''LocationData
deriveSafeCopy 1 'base ''Location
deriveSafeCopy 1 'base ''MetaLocation
```

```
{-# version = 2 #-}

data Location =   Location LocationType LocationData

data LocationEntry = LocationEntry Location | LocationEntryRef OID

data MetaLocation where

    MetaLocation :: Maybe OID
                  -> Set LocationEntry
                  -> MetaLocation

    MetaLocationRef :: OID
                    -> MetaLocation

deriveSafeCopy 1 'base ''LocationEntry
deriveSafeCopy 2 'extension ''Location
deriveSafeCopy 2 'extension ''MetaLocation
```

# Что получилось: версионность

```
{-# version = 2 #-}

import qualified Data.V1.Location as Previous

instance Migrate Location where
  type MigrateFrom Location = Previous.Location
  migrate (Previous.Location oid lt ld) = Location lt ld

instance Migrate MetaLocation where
  type MigrateFrom MetaLocation = Previous.MetaLocation
  migrate (Previous.MetaLocation oid s) = MetaLocation oid (S.map (LocationEntry . migrate) s)
  migrate (Previous.MetaLocationRef oid) = MetaLocationRef oid
```

# Что получилось: версионность

- safecopy
  - RecordWildCards

```
instance Migrate Hotel where
  type MigrateFrom Hotel = Previous.Hotel
  migrate h@(Previous.Hotel{..}) =
    Hotel { hoOID          = Previous.hoOID h
          , hoSupplierID   = Previous.hoSupplierID h
          , hoWebLink      = Previous.hoWebLink h
          , hoClass        = Previous.hoClass h
          , hoType         = Previous.hoType h
          , hoLocation     = migrate (Previous.hoLocation h)
          , hoDescription  = Previous.hoDescription h
          , hoCurrency      = Previous.hoCurrency h
          , hoBoard        = Previous.hoBoard h
          , hoCapacity     = Previous.hoCapacity h
          , hoRooms        = Previous.hoRooms h
          , hoPricesBegin  = Previous.hoPricesBegin h
          , hoPricesEnd    = Previous.hoPricesEnd h
          , hoPrices       = Previous.hoPrices h
          }
```



# Что получилось: версионность

- safecopy
  - **RecordWildCards**
  - Почему не линзы?

```
instance Migrate Hotel where
  type MigrateFrom Hotel = Previous.Hotel
  migrate (Previous.Hotel{..}) =
    Hotel { hoLocation      = migrate (Previous.hoLocation h)
          , ..
          }
```

# Что получилось: версионность

- safecopy
  - Явная нумерация версий
    - Один тип - один файл
    - Разделение модели и логики по модулям
    - Реэкспорт нужных версий
    - **Написание миграций для всех типов, куда входит изменившийся тип**
    - **Контроль версий объектов при code review и rebase**

```
module Data.Model.Internal.v0_1_3_1.Location where
```

```
import Data.Model.Internal.v0_1_3_0.Language
import Data.Model.Internal.v0_1_3_0.Media
import Data.Model.Internal.v0_1_3_0.OID
import Data.Model.Internal.v0_1_3_0.Types
```

```
module Data.Model.Internal ( module Hotel
                           , module Room
                           , module Location
                           , module Language
                           , module Media
                           , module OID
                           , module Types
                           , module Feature
                           ) where
```

```
import Data.Model.Internal.v0_1_3_14.Room as Room
import Data.Model.Internal.v0_1_3_8.Location as Location
import Data.Model.Internal.v0_1_3_0.Language as Language
import Data.Model.Internal.v0_1_3_0.Media as Media
import Data.Model.Internal.v0_1_3_0.OID as OID
import Data.Model.Internal.v0_1_3_0.Types as Types
import Data.Model.Internal.v0_1_3_6.Feature as Feature
import Data.Model.Internal.v0_1_4_1.Hotel as Hotel
```

# Результаты

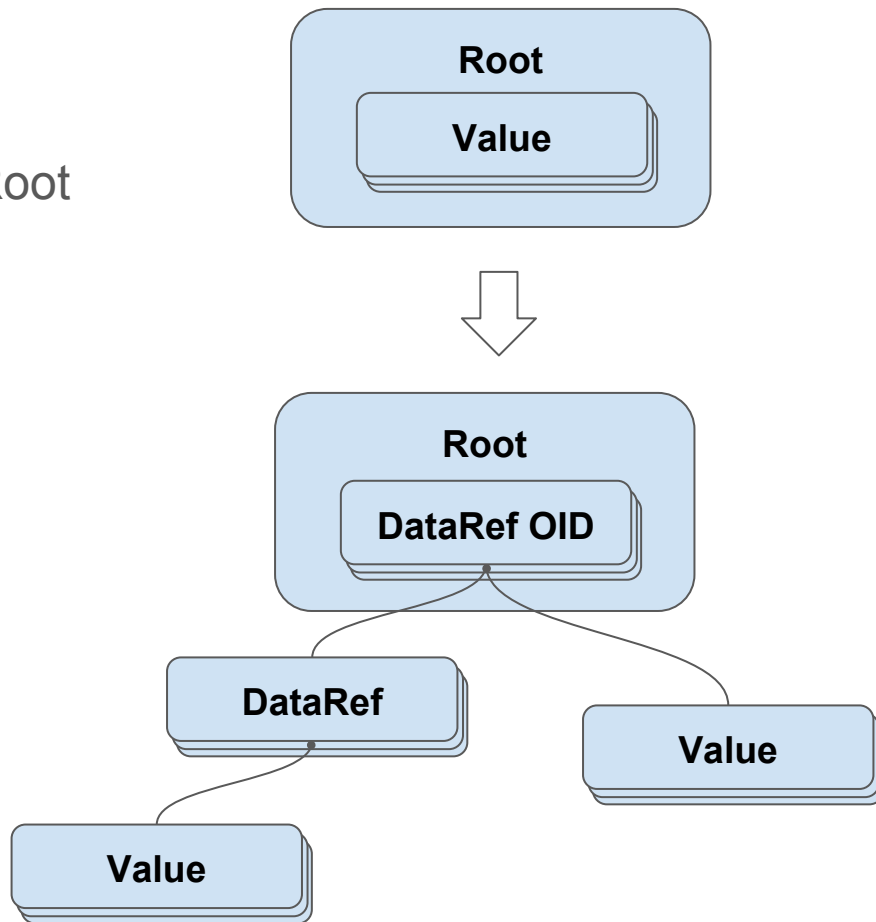
- Стартовала командная разработка
- Увеличилась скорость разработки
- Уменьшилась сложность
- Автоматическая миграция данных
- Производительность...

# Производительность

Было		Стало	
Построение materialized views (импорт)	~1200с	База Blobster (импорт)	~10с
Основной запрос	100мс..5с	Основной запрос	<10мс

# Производительность

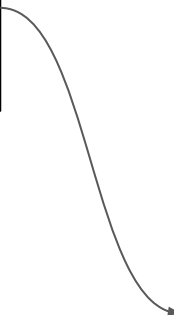
- Явное управление индексами в Root
- Разбиение на более мелкие элементы
- Частичная загрузка
  - На разных уровнях



# Как все должно быть

- Данные и метаданные

```
data BusinessObject
{ attrib1 :: String
, ...
, attribN :: Int
, relatedAttrib :: Other
}
```



```
instance HasSchema BusinessObject
  data Metadata BusinessObject = Schema (SimpleAttribute ...)
  ...
  (SimpleAttribute ...)
  (Relation RelationType ...)

  materialize = genericMaterialize ...
  dematerialize = genericDematerialize ...
```

# Как все должно быть

- Миграция
  - Andres Löh
    - Datatype-generic data migrations
      - <https://www.andres-loeh.de/dgdm-ifip.pdf>
      - Typelevel eDSL для описания миграций
  - HEX Research Team
    - Прозрачные Plain Vanilla ADT
      - без явных версий
    - Type families для метаданных
    - Рантайм-метаданные
    - Git
    - Автоматическая генерация скрипта миграции из предыдущих пунктов + запрос спецификации миграции для нетривиальных случаев (???)

# Как все должно быть

- Хранение
  - Сборка мусора/компактизация (WiP)
  - Отложенная запись (WiP)
  - Log-structured
  - Non-GC'd, contiguous storage for immutable data structures
    - <https://github.com/ezyang/compact>



# Как все должно быть

- Навигация по данным
  - Линзы (использование индексов)
  - eDSL + Generic (???)

# Благодарности

- <https://gitter.im/ruHaskell/forall>
- <https://t.me/haskellru>

# Хотите писать на Haskell за деньги?

 [dzuikov@gmail.com](mailto:dzuikov@gmail.com)