

View как чистая функция от состояния базы данных



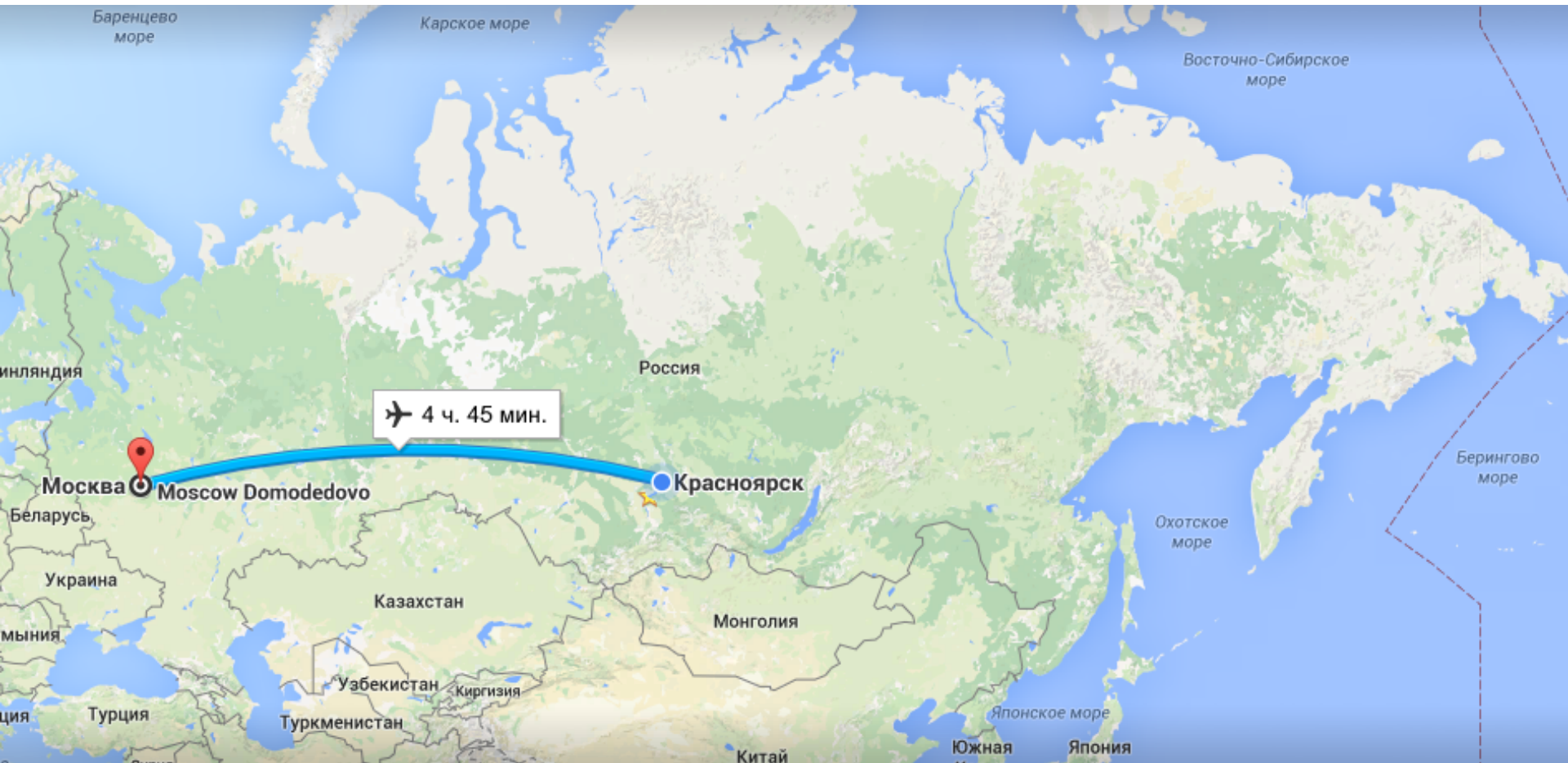
Илья Беда / bro.agency



<http://www.devconf.ru>

Кто я?

- Тимлид и один из основателей bro.agency
- Разрабатываю веб-формочки на python в течении 7 лет
- Специализируюсь на аутсорс-разработке
- Продвигаю функциональное программирование в массы
- Провожу воркшопы и мастер-классы
- С недавнего времени выступаю на конференциях



Сегодня я расскажу вам о кэшировании



**There are only two hard things in Computer Science:
cache invalidation and naming things.**

Phil Karlton

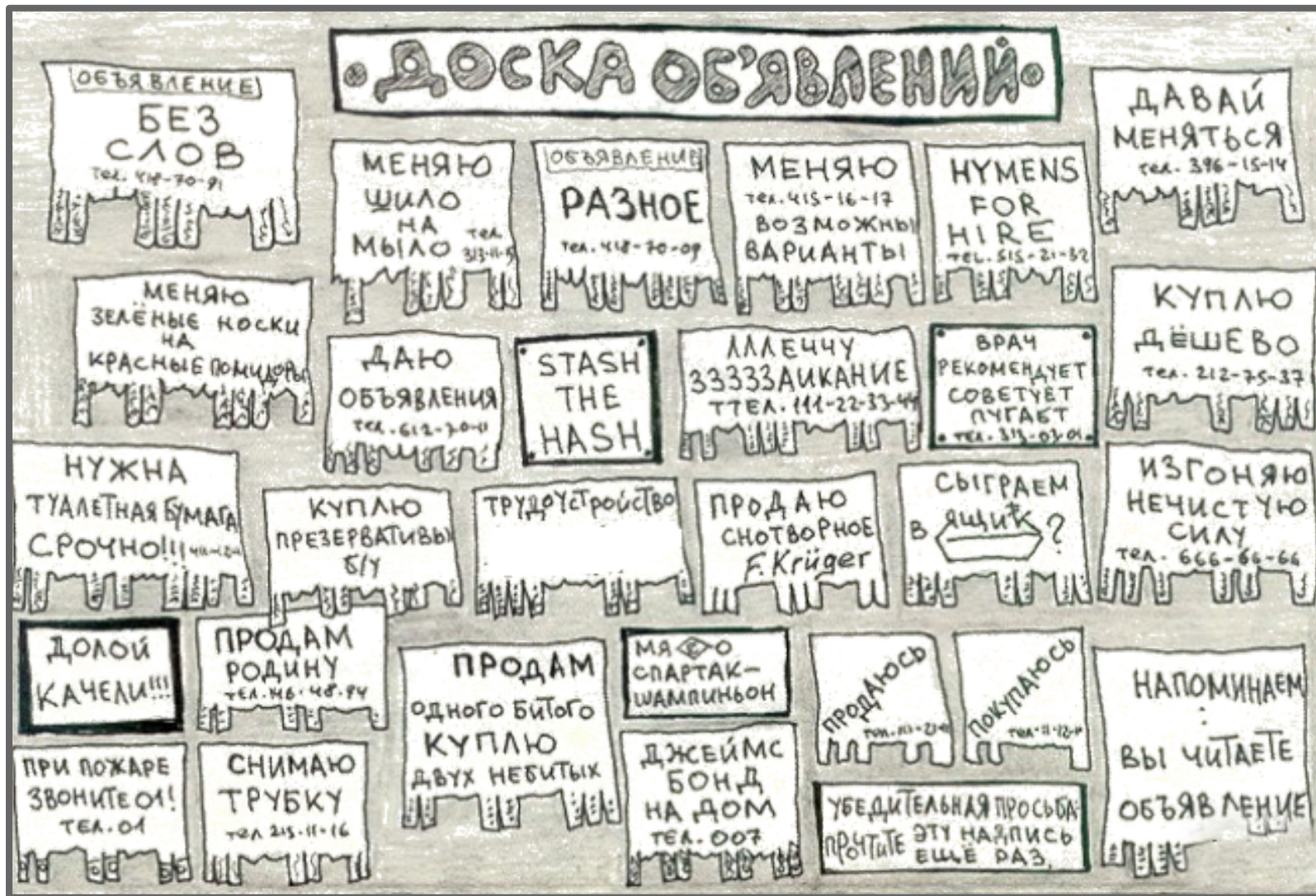
Зачем нужна инвалидация кэша?

Инвалидация по умолчанию (по таймауту)

- кэш становится не актуальным, но сами данные остались неизменными;
- пользователь увидит не актуальную версию страницы.

Инвалидация по умолчанию (по таймауту)

- кэш становится не актуальным, но сами данные остались неизменными;
- **пользователь увидит не актуальную версию страницы.**



**Изменить правила инвалидации кэша
нам помогут чистые функции.**

Что такое чистая функция?

Чистая функция

1. Детерминированная функция
2. Функция не обладающая сторонними эффектами

Детерминированная функция

```
def add(a, b):  
    return a + b
```

Недетерминированная функция

```
import random

def add(a,b):
    return a + b + random.randint(0,100)
```

Недетерминированная функция

```
import random
```

```
def add(a,b):  
    return a + b + random.randint(0,100)
```

Недетерминированная функция

```
import datetime  
  
datetime.datetime.now()
```


Функция с побочными эффектами

```
count = 0
```

```
def add(a,b):
```

```
    count += 1
```

```
    return a + b
```

Функция с побочными эффектами

```
count = 0
```

```
def add(a,b):
```

```
    count += 1
```

```
    return a + b
```

Чистая функция

```
def calculate(a, b, c):  
    return a + b * c
```

Преимущество чистых функций

Если строить cache key на основе аргументов чистой функции, то этот кэш не нужно инвалидировать.

Декоратор для кэширования чистых функций

```
1 def cache_pure(fn):
2     def wrap(*args, **kwargs):
3         cache_key = create_cache_key(*args, **kwargs)
4         result = cache.get(cache_key)
5         if result is None:
6             result = fn(*args, **kwargs)
7             cache.set(cache_key, result)
8         return result
9     return wrap
```

Кэшируем чистую функцию

@cache_pure

```
def calculate(a, b, c):  
    return a + b * c
```

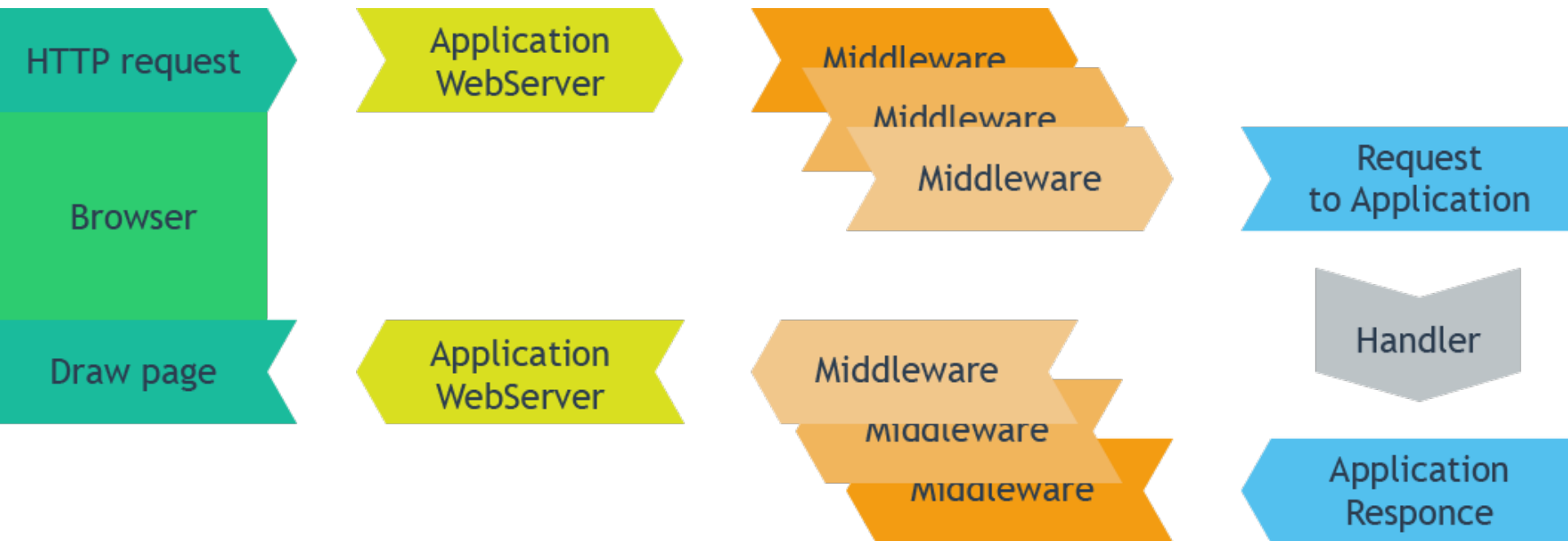
```
r = calculate(1, 2, 3)
```

```
r = calculate(1, 2, 3)
```

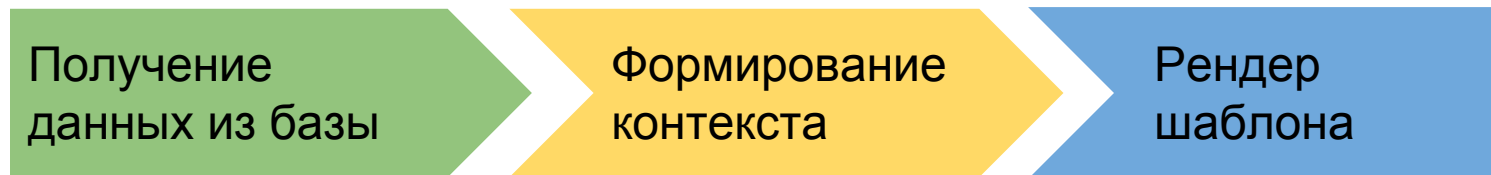
```
r = calculate(1, 2, 3)
```

**Давайте сделаем наш ВЕБ функционально
чистым и забудем про проблемы
инвалидации кэша.**

Как работает ВЕБ



Что делает Handler?



Как выглядит Handler

```
1 def handler(request, database)
2     data = database.get_data(request)
3     context = transform_date_to_context(data)
4     html_string = template.format(**context)
5     return html_string
```

Как выглядит Handler

```
1 def handler(request, database)
2     data = database.get_data(request)
3     context = transform_data_to_context(data)
4     html_string = template.format(**context)
5     return html_string
```

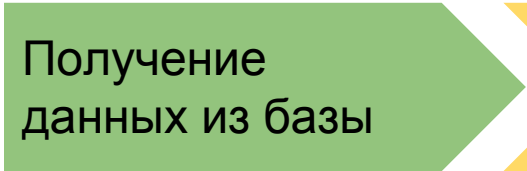
Как выглядит Handler

```
1 def handler(request, database)
2     data = database.get_data(request)
3     context = transform_data_to_context(data)
4     html_string = template.format(**context)
5     return html_string
```

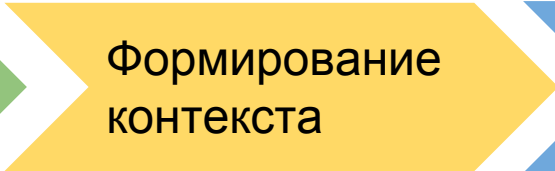
Как выглядит Handler

```
1 def handler(request, database)
2     data = database.get_data(request)
3     context = transform_data_to_context(data)
4     html_string = template.format(**context)
5     return html_string
```

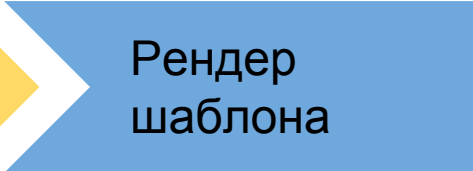
**Мы пишем это каждый день
начиная с 90-х годов**

A green arrow pointing to the right, containing the text "Получение данных из базы".

Получение
данных из базы

A yellow arrow pointing to the right, containing the text "Формирование контекста".

Формирование
контекста

A blue arrow pointing to the right, containing the text "Рендер шаблона".

Рендер
шаблона

Как это выглядит в реальности (Django)

```
1 def some_view(request, url_param):  
2     items = SomeModel.objects.filter(some_field=url_param)  
3     c = Context({'items': items})  
4     response = Template("my_template.html").render(c)  
5     return HttpResponse(response)
```

Как это выглядит в реальности (Django)

```
1 def some_view(request, url_param):  
2     items = SomeModel.objects.filter(some_field=url_param)  
3     c = Context({'items': items})  
4     response = Template("my_template.html").render(c)  
5     return HttpResponse(response)
```


Как это выглядит в реальности (Django)

```
1 def some_view(request, url_param):
2     items = SomeModel.objects.filter(some_field=url_param)
3     c = Context({'items': items})
4     response = Template("my_template.html").render(c)
5     return HttpResponse(response)
```

Как это выглядит в реальности (Django)

```
1 def some_view(request, url_param):  
2     items = SomeModel.objects.filter(some_field=url_param)  
3     c = Context({'items': items})  
4     response = Template("my_template.html").render(c)  
5     return HttpResponse(response)
```

Проанализируем с точки зрения чистоты

```
1 def some_view(request, url_param):  
2     items = SomeModel.objects.filter(some_field=url_param)  
3     c = Context({'items': items})  
4     response = Template("my_template.html").render(c)  
5     return HttpResponse(response)
```

Проанализируем с точки зрения чистоты

```
1 def some_view(request, url_param):  
2     items = SomeModel.objects.filter(some_field=url_param)  
3     c = Context({'items': items})  
4     response = Template("my_template.html").render(c)  
5     return HttpResponse(response)
```

Приведем функцию к чистому виду

```
1 def some_view(request, url_param):  
2     items = list(  
3         SomeModel.objects.filter(some_field=url_param))  
4  
5     @cache_pure  
6     def _some_view_pure(items):  
7         c = Context({'items': items})  
8         response = Template("my_template.html").render(c)  
9         return HttpResponse(response)  
10    return _some_view_pure(items)
```

Приведем функцию к чистому виду

```
1 def some_view(request, url_param):
2     items = list(
3         SomeModel.objects.filter(some_field=url_param))
4
5     @cache_pure
6     def _some_view_pure(items):
7         c = Context({'items': items})
8         response = Template("my_template.html").render(c)
9         return HttpResponse(response)
10    return _some_view_pure(items)
```

Приведем функцию к чистому виду

```
1 def some_view(request, url_param):  
2     items = list(  
3         SomeModel.objects.filter(some_field=url_param))  
4  
5     @cache_pure  
6     def _some_view_pure(items):  
7         c = Context({'items': items})  
8         response = Template("my_template.html").render(c)  
9         return HttpResponse(response)  
10    return _some_view_pure(items)
```

Давайте пофантазируем

```
1 def some_view(request, url_param):
2     @cache_pure
3     def _some_view_pure(url_param, database_state):
4         items = SomeModel.objects.filter(
5             some_field=url_param)
6         c = Context({'items': items})
7         response = Template("my_template.html").render(c)
8         return HttpResponse(response)
9     return _some_view_pure(
10         url_param,
11         get_database_state_from_magic())
```


Давайте пофантазируем

```

1  def some_view(request, url_param):
2      @cache_pure
3      def _some_view_pure(url_param, database_state):
4          items = SomeModel.objects.filter(
5              some_field=url_param)
6          c = Context({'items': items})
7          response = Template("my_template.html").render(c)
8          return HttpResponse(response)
9      return _some_view_pure(
10         url_param,
11         get_database_state_from_magic())

```

Давайте пофантазируем

```
1 def some_view(request, url_param):
2     @cache_pure
3     def _some_view_pure(url_param, database_state):
4         items = SomeModel.objects.filter(
5             some_field=url_param)
6         c = Context({'items': items})
7         response = Template("my_template.html").render(c)
8         return HttpResponse(response)
9     return _some_view_pure(
10         url_param,
11         get_database_state_from_magic())
```

```
def get_database_state_from_magic():
```

Если пренебречь тем, что база данных может вернуться в предыдущее состояние, то можно использовать индекс поколений.

И это практичное решение проблемы, которое можно использовать.

Реализация индексирования поколений

```
1 import redis
2 from django.db.models.signals import post_save
3
4 r = redis.StrictRedis(host='localhost', port=6379, db=0)
5
6 def inc_generation_index(sender, **kwargs):
7     r.inc('generation_index')
8 post_save.connect(inc_generation_index)
9
10 def get_database_state_from_magic():
11     return r.get('generation_index')
```

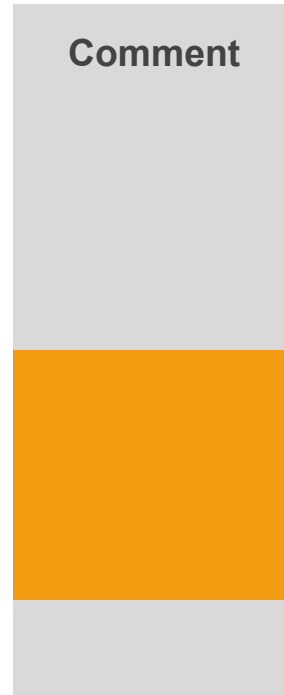
И теперь это реально работающий код

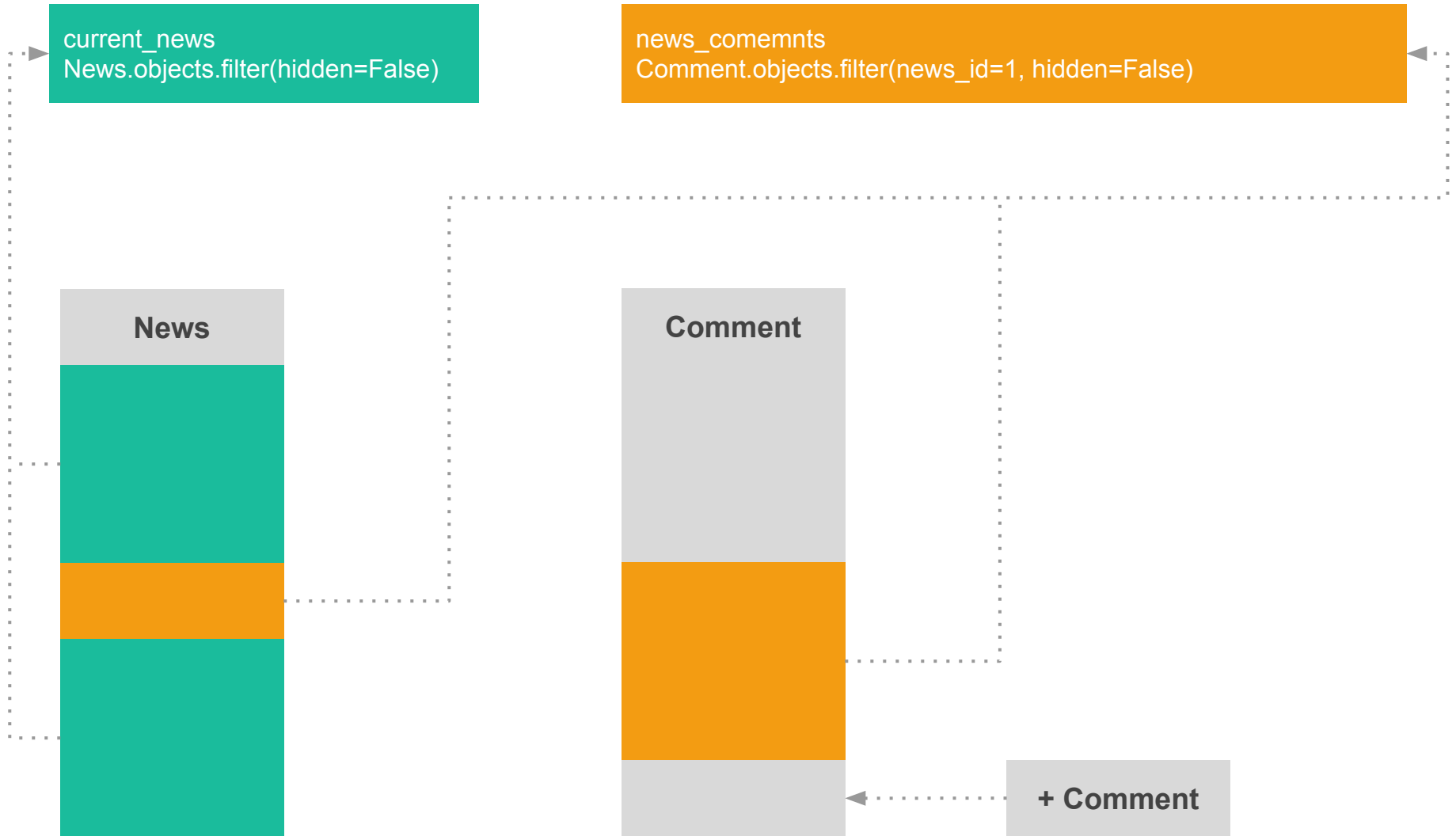
```
1 def some_view(request, url_param):
2     @cache_pure
3     def _some_view_pure(url_param, database_state):
4         items = SomeModel.objects.filter(
5             some_field=url_param)
6         c = Context({'items': items})
7         response = Template("my_template.html").render(c)
8         return HttpResponse(response)
9     return _some_view_pure(
10         url_param,
11         get_database_state_from_magic())
```

**Какие могут быть проблемы
при индексировании поколений?**

current_news
News.objects.filter(hidden=False)

news_comemnts
Comment.objects.filter(news_id=1, hidden=False)





**Индексирование поколений не оптимально
если не учитывать гранулярность данных.**

Гранулярная инвалидация

Нужно использовать не все состояние базы данных,
а только её часть.

**Можно индексировать поколения каждой таблицы
отдельно**

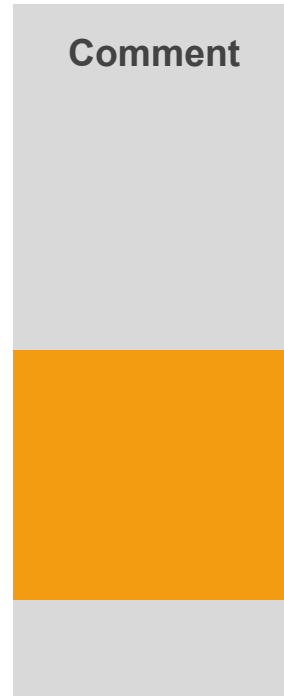
<https://github.com/jmoiron/johnny-cache>

**Можно ивалидировать кэш, зависящий от
определенного состояния базы данных**

Тогда нам нужен инструмент для его описания

current_news
News.objects.filter(hidden=False)

news_comemnts
Comment.objects.filter(news_id=1, hidden=False)

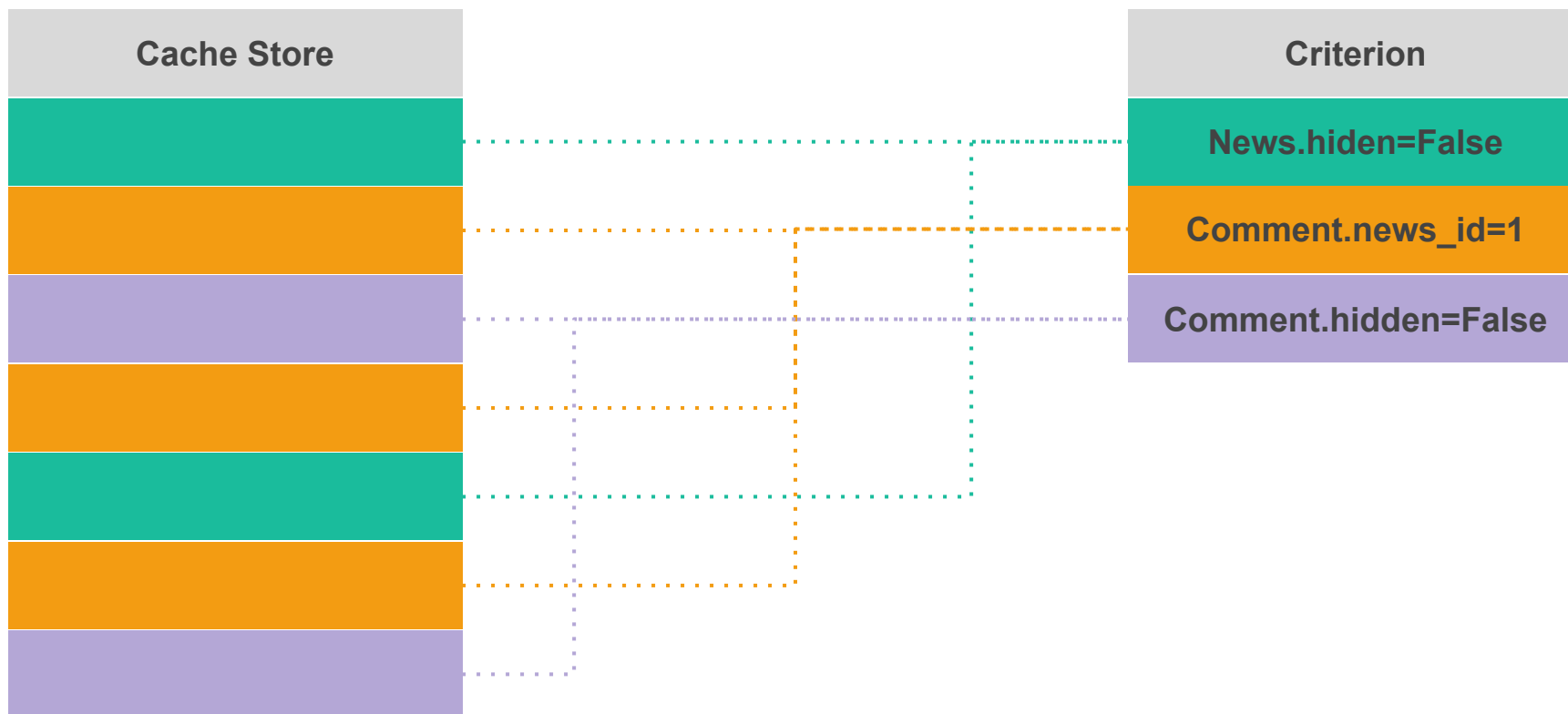


```
current_news  
News.objects.filter(hidden=False)
```

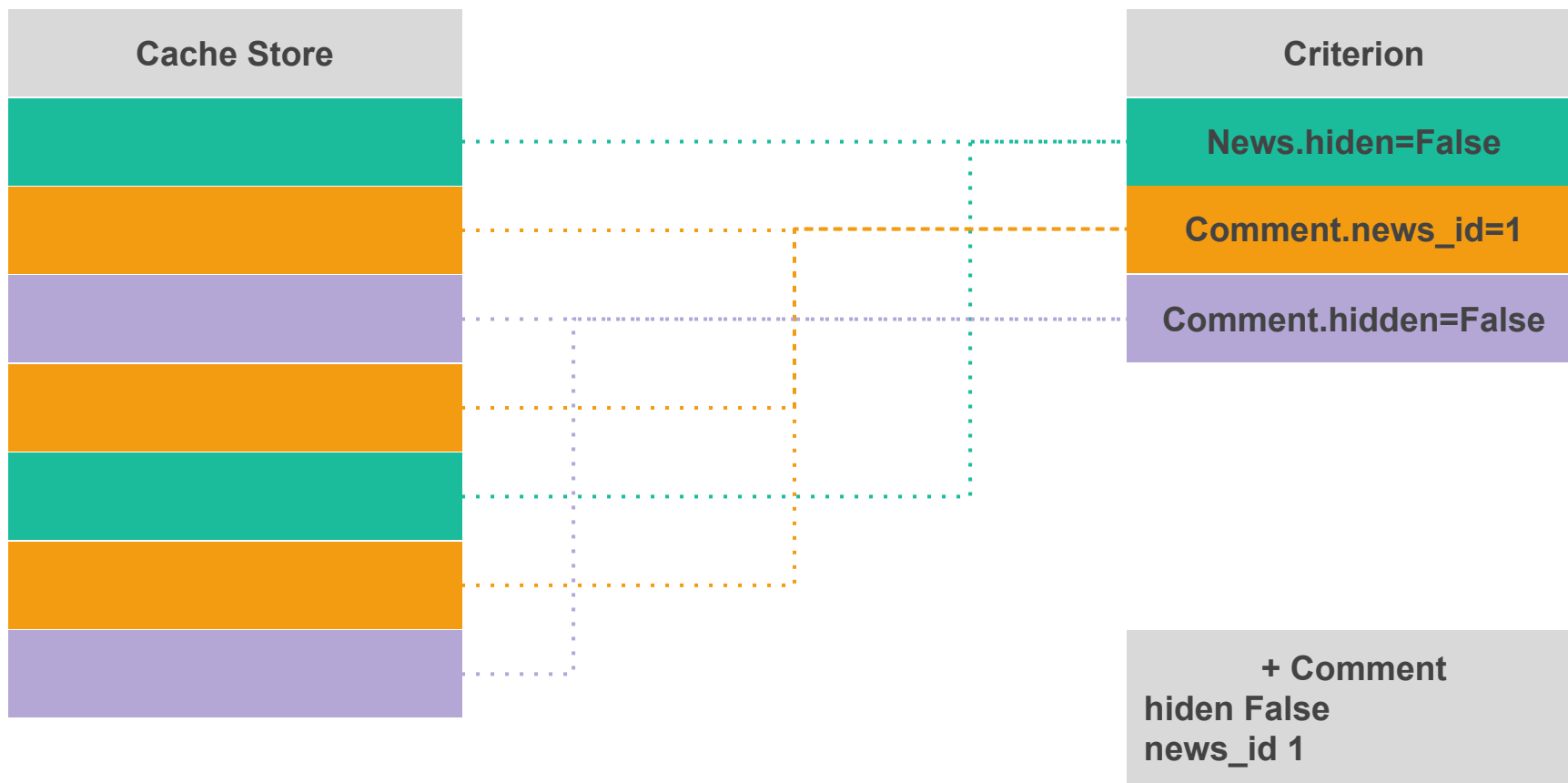
```
news_comemnts  
Comment.objects.filter(news_id=1, hidden=False)
```



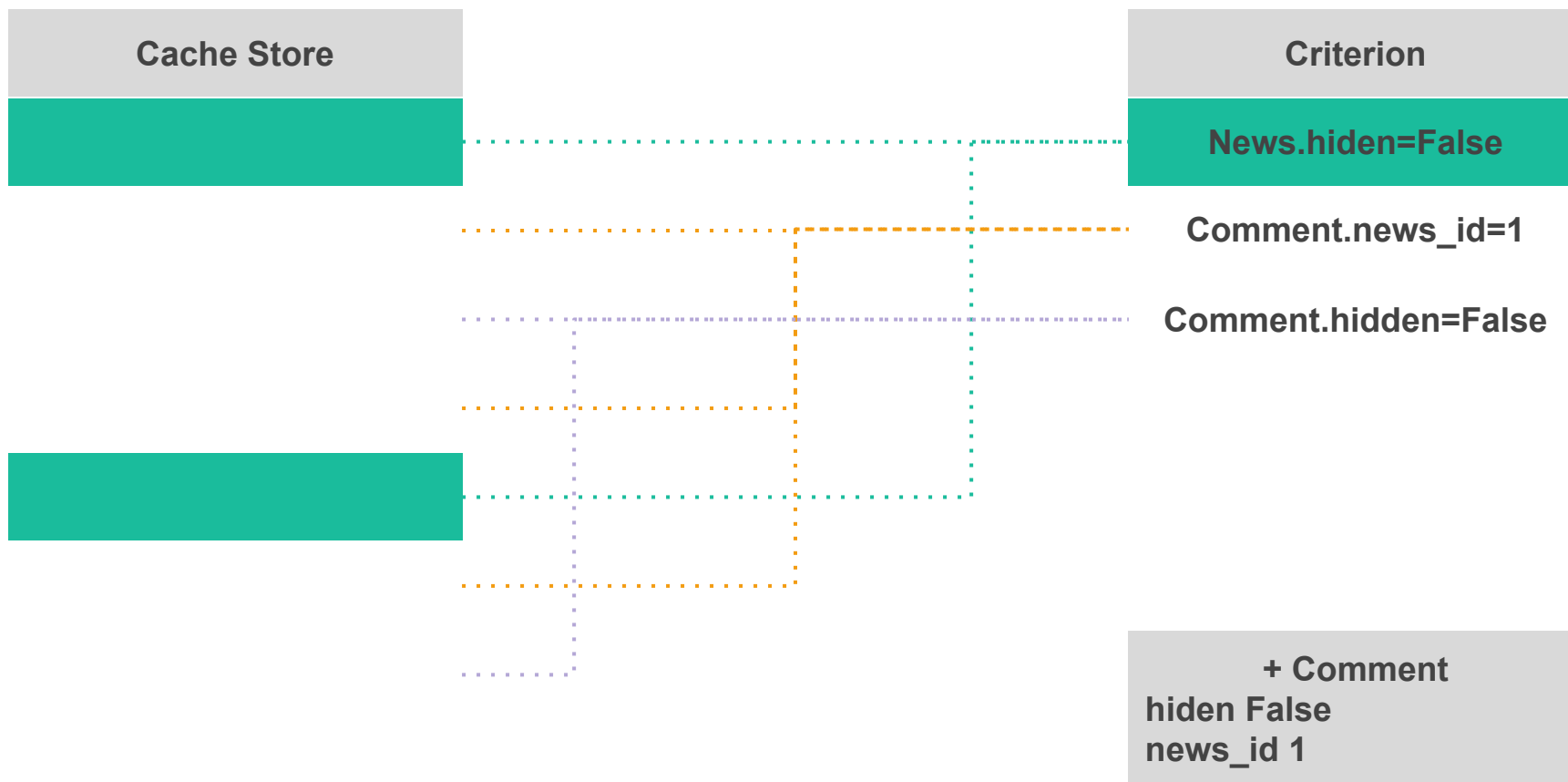
Можно индексировать части запросов



Можно индексировать части запросов



Можно индексировать части запросов



Тоже есть готова реализация

<http://hackflow.com/blog/2014/03/09/on-orm-cache-invalidation/>

<https://github.com/suor/django-cacheops>

Эффективное кэширование с помощью cacheops

```
1 def some_view(request, url_param):
2     items = SomeModel.objects.filter(some_field=url_param)
3
4     @cached_view_as(items)
5     def _render(request, url_param):
6         c = Context({'items': items})
7         response = Template("my_template.html").render(c)
8         return HttpResponse(response)
9
10    return _render(request, url_param)
```

Эффективное кэширование с помощью cacheops

```
1 def some_view(request, url_param):
2     items = SomeModel.objects.filter(some_field=url_param)
3
4     @cached_view_as(items)
5     def _render(request, url_param):
6         c = Context({'items': items})
7         response = Template("my_template.html").render(c)
8         return HttpResponse(response)
9
10    return _render(request, url_param)
```

Эффективное кэширование с помощью cacheops

```
1 def some_view(request, url_param):
2     items = SomeModel.objects.filter(some_field=url_param)
3
4     @cached_view_as(items)
5     def _render(request, url_param):
6         c = Context({'items': items})
7         response = Template("my_template.html").render(c)
8         return HttpResponse(response)
9
10    return _render(request, url_param)
```

Class Based View

```
1 class SomeModelListView(ListView)
2     def get_queryset(self):
3         return SomeModel.objects.all()
4
5     def get(self, request, *args, **kwargs):
6         self.object_list = self.get_queryset()
7         context = self.get_context_data()
8         return self.render_to_response(context)
```

Class Based View

```
1 class SomeModelListView(ListView)
2     def get_queryset(self):
3         return SomeModel.objects.all()
4
5     def get(self, request, *args, **kwargs):
6         self.object_list = self.get_queryset()
7         context = self.get_context_data()
8         return self.render_to_response(context)
```

```
1 class CachedListView(ListView):
2     def querysets(self):
3         return [self.get_queryset()]
4
5     def get(self, request, *args, **kwargs):
6         @cached_view_as(*self.querysets())
7         def _get(request, *args, **kwargs):
8             res = super(CachedListView, self).get(
9                 request, *args, **kwargs)
10            res.render()
11            return res
12    return _get(request, *args, **kwargs)
```


Действуйте с умом

- Нужно понимать бизнес-логику.
- Исходя из бизнес-логики принимать упрощения и идти компромиссы.
- Все варианты хороши — вам решать какой использовать.

Спасибо за внимание!
Я готов ответить на ваши вопросы.