

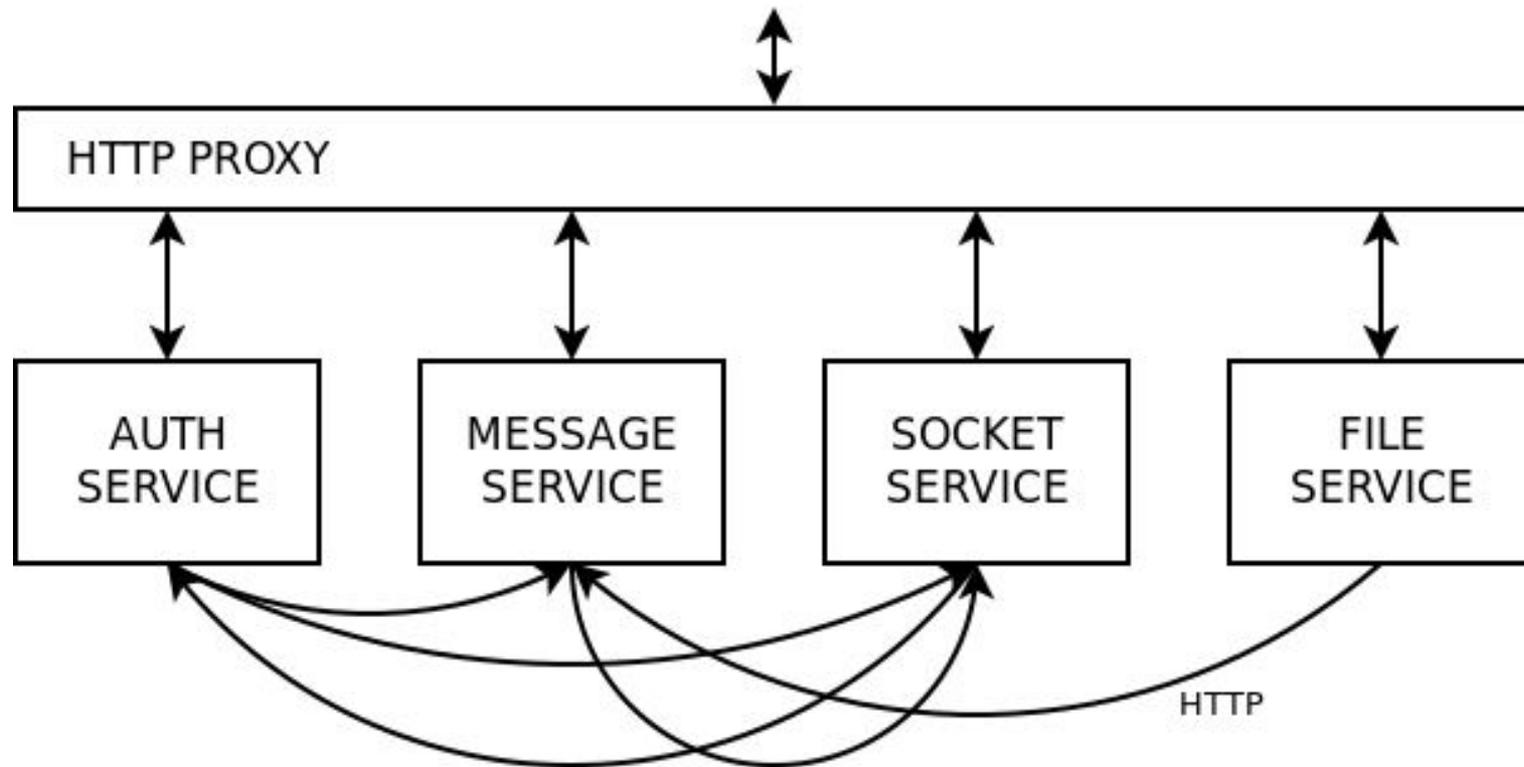
# От микросервисов к микроагентам

Власов Дмитрий

# Развитие микросервисных архитектур

- ✓ 2011 — 2012 появление понятия микросервисной архитектуры (МСА)
- ✓ 2012 — 2014 широкое распространение МСА среди технологических лидеров
- ✓ 2013 — 2014 публикация «Реактивного манифеста» (v.1 и v.2)
- ✓ 2015 — выход книги С.Ньюмена «Создание микросервисов»

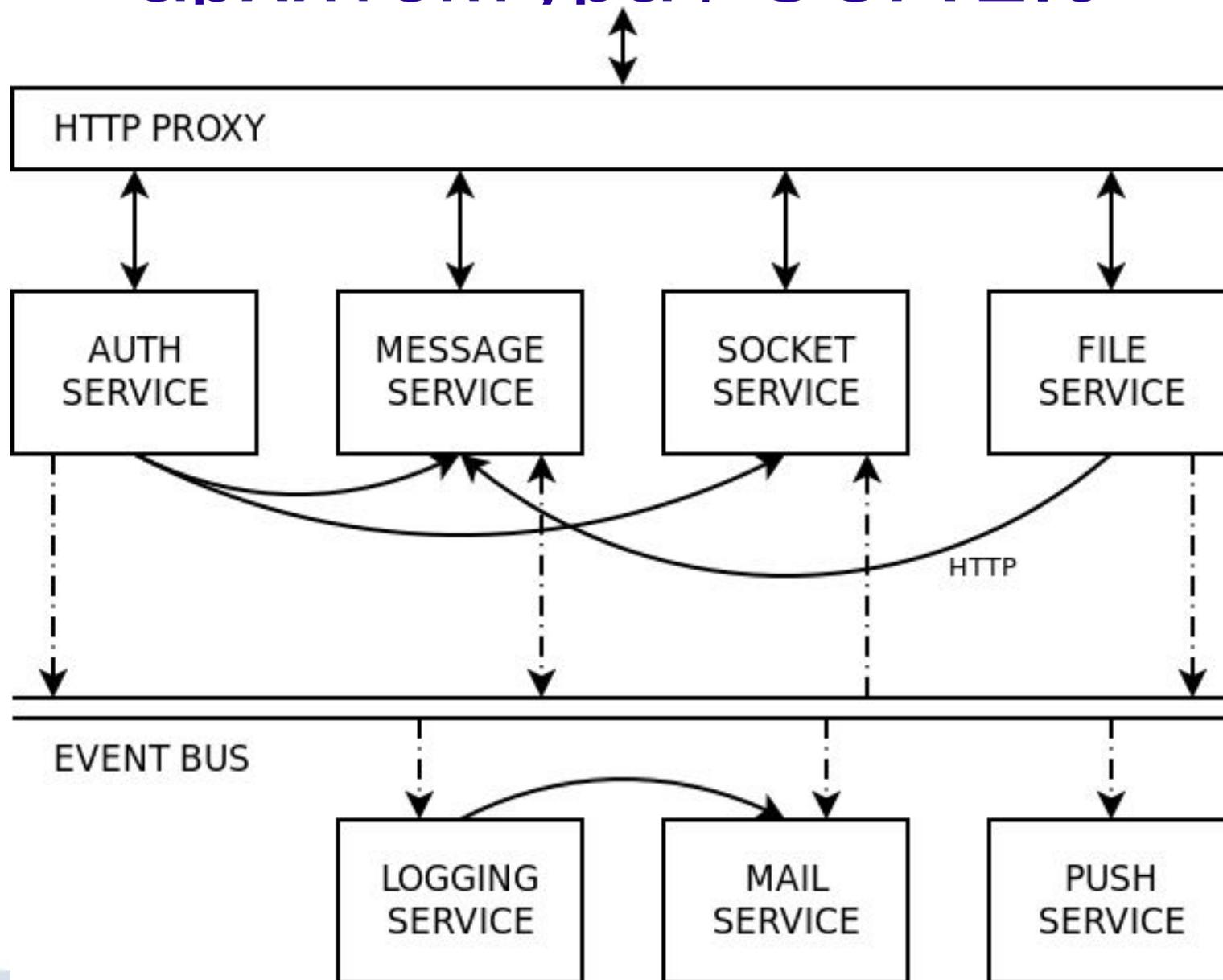
# Клиент-серверная архитектура SOA 1.0



# Клиент-серверная архитектура

- ✓ Система построенная на запрос-ответах по сути своей синхронна
- ✓ Обращение из точки в точку, без посредников
- ✓ Гарантирована доставка или явная ошибка

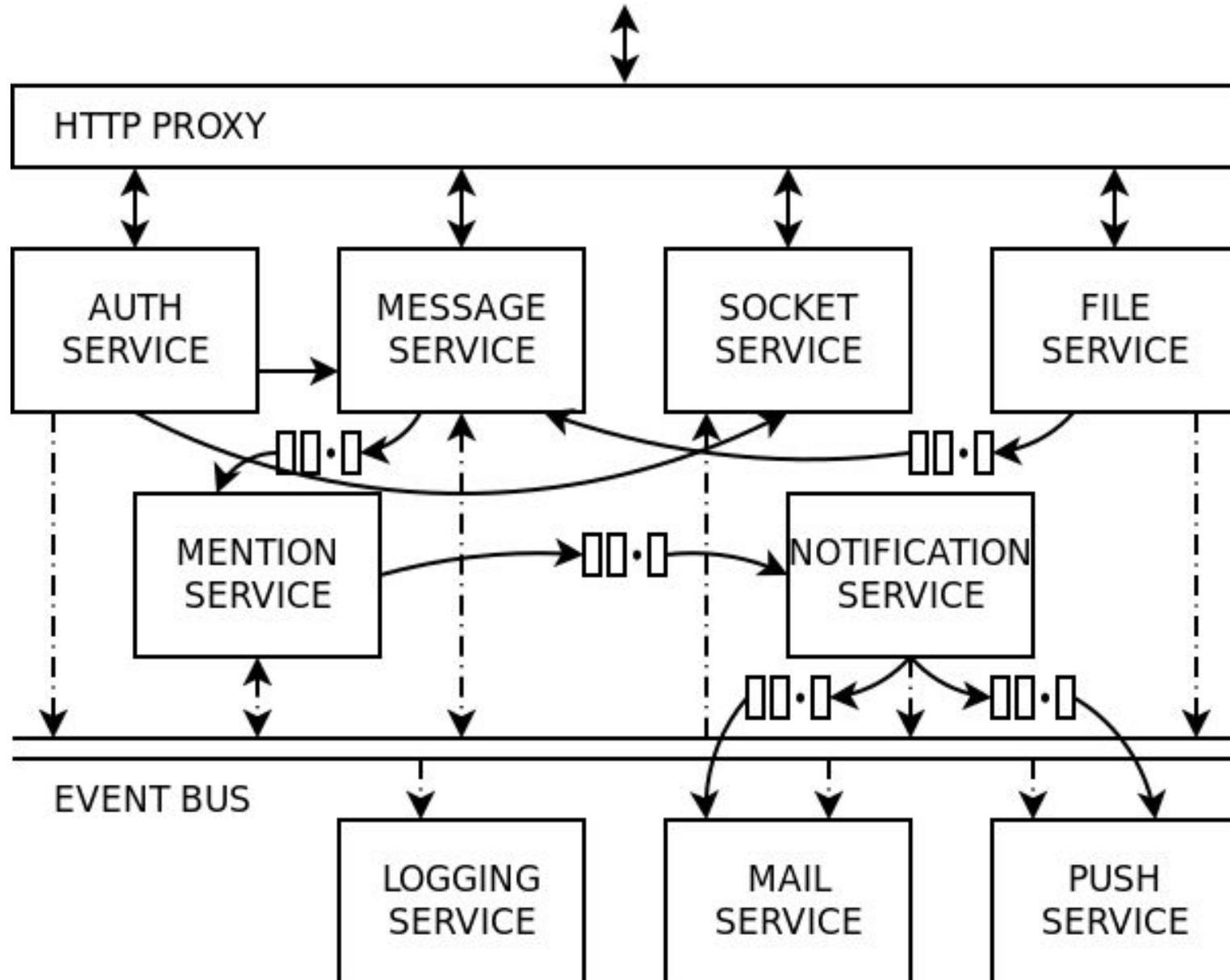
# Событийно-ориентированная архитектура / SOA 2.0



# Событийно-ориентированная архитектура

- ✓ Система построенная на обмене событиями асинхронна по сути
- ✓ Отправители и получатели не знают друг о друге, отправитель просто посылает сигнал в общую шину
- ✓ Один сигнал может получить множество получателей, получатели могут подключаться и отключаться независимо,
- ✓ Гарантия доставки зависит от реализации посредника - хранит ли он данные

# Реактивная архитектура



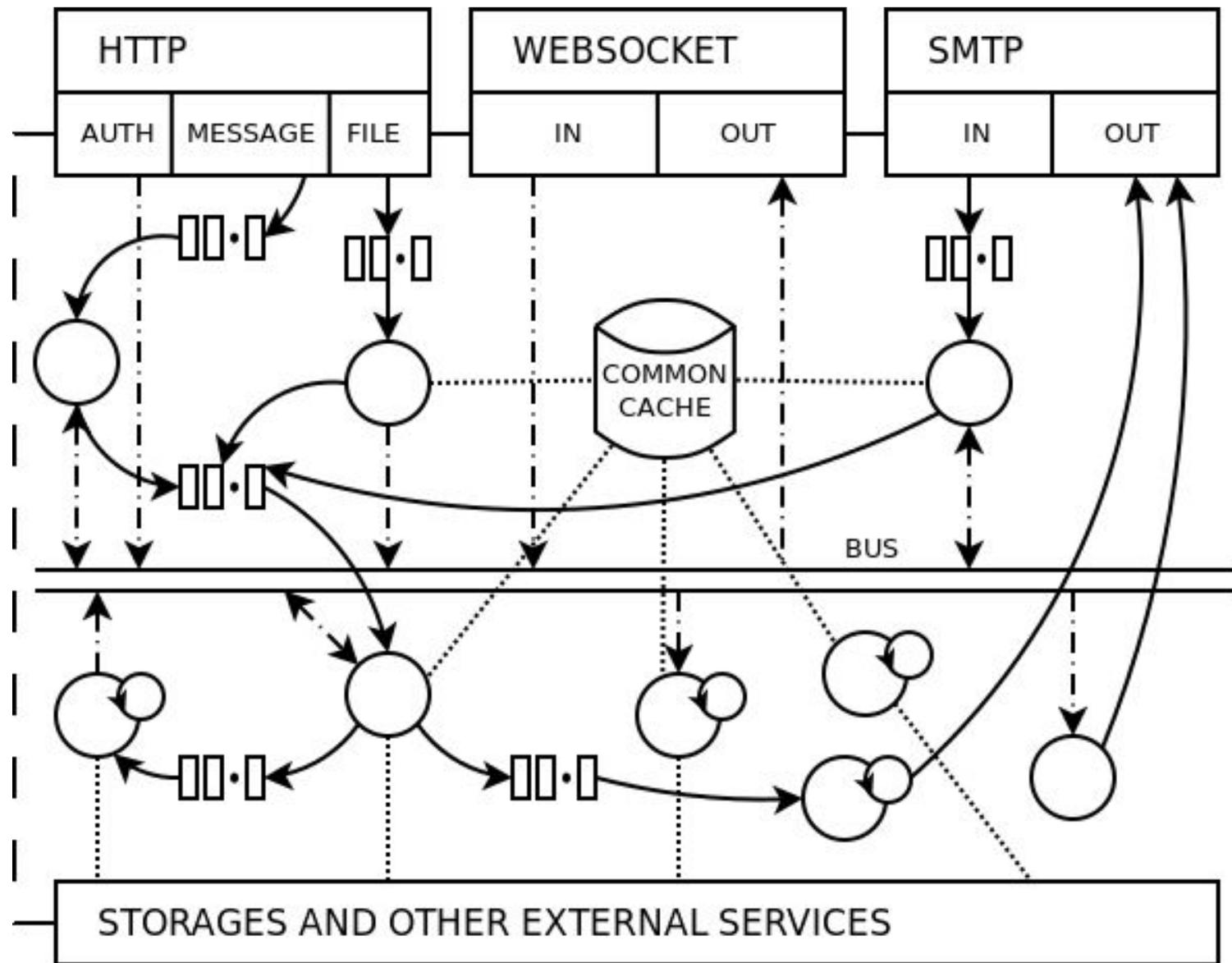
# Реактивная архитектура

- ✓ Система асинхронна, отправитель и получатель не связаны непосредственно
- ✓ Доставка сообщений гарантирована, как зачастую и порядок
- ✓ Как правило отправитель и получатель одни, если брокер очередей не создает ветвлений.

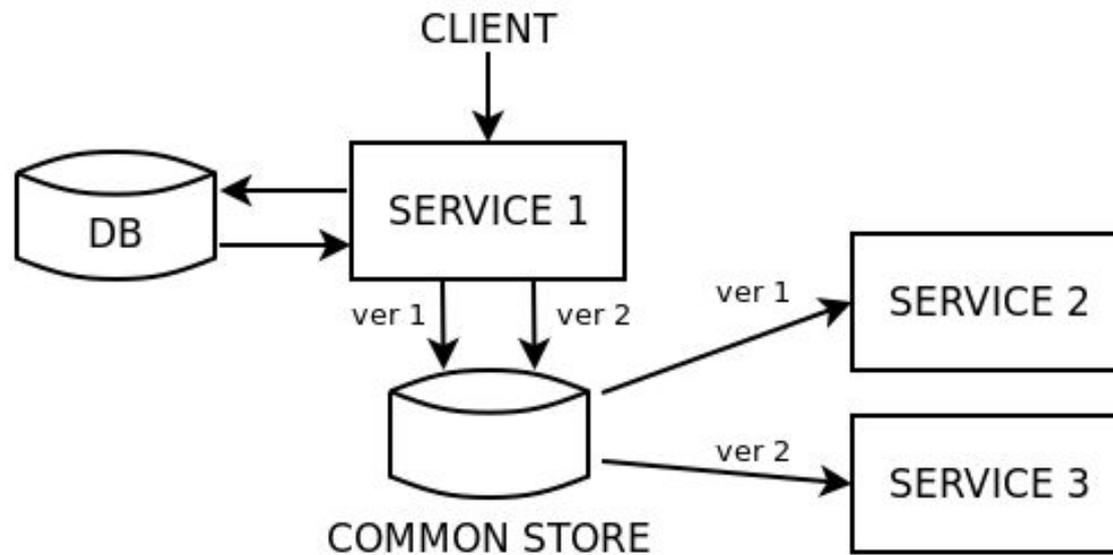
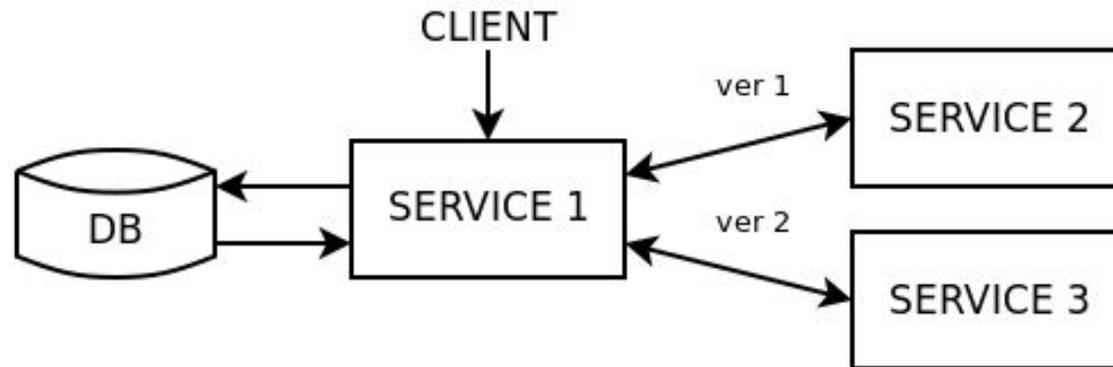
# Периодические задачи

- ✓ Позволяют группировать данные в пакеты для более эффективной обработки
- ✓ Позволяют обновлять и инвалидировать кэши
- ✓ Запускать проверки и обслуживание БД, рассылать отчеты по ним
- ✓ Инициировать запросы к другим сервисам

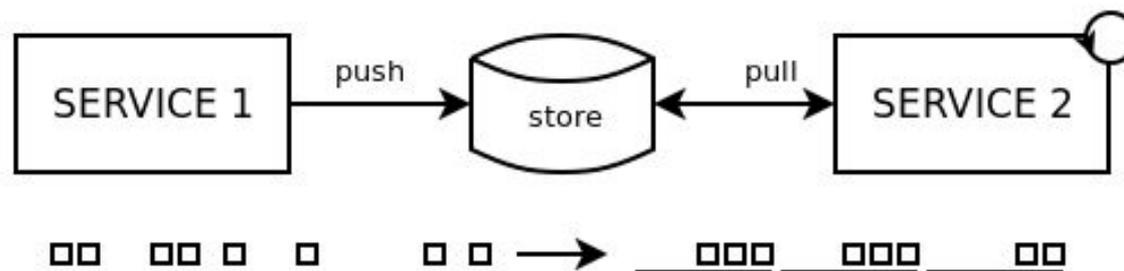
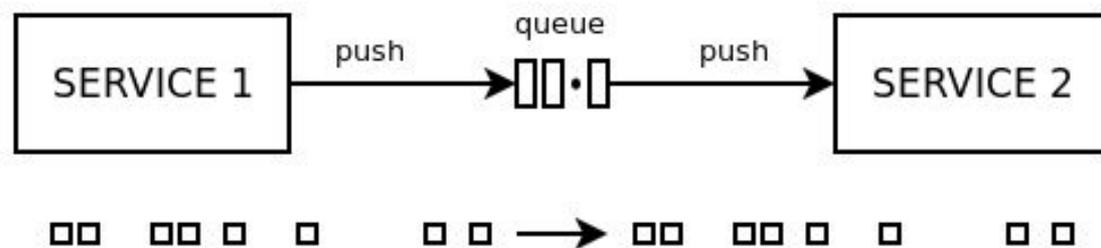
# Смешанная архитектура



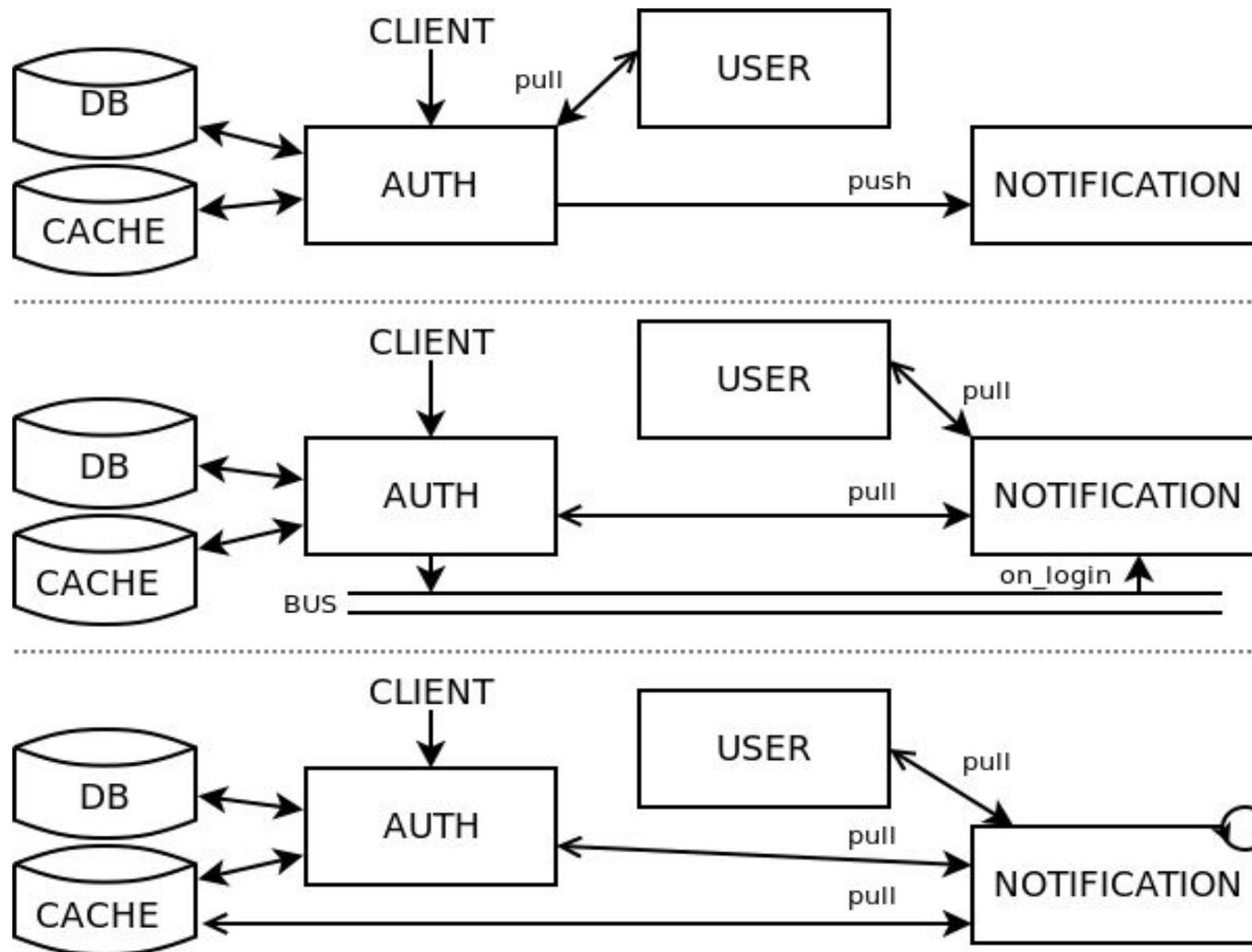
# Сервисы связывают данные, а не используемые технологии



# Хранилище может быть сервисом, очередь тоже хранилище



# Независимый сервис сам извлекает информацию



# Агентно-ориентированное программирование

- ✓ Агенты – небольшие, автономные, совместно работающие программы
- ✓ Поведение агента определяется целью а не конкретным функционалом
- ✓ Агент может модифицировать свое поведение в зависимости от окружения
- ✓ Агентно-ориентированное программирование развитие ООП

# Микроагент

- ✓ Может подключаться к событийной шине, отправлять и получать события
- ✓ Может быть узлом обработки потока данных, работая с очередями
- ✓ Может выполнять синхронные вызовы к другим микроагентам
- ✓ Может содержать периодические функции
- ✓ Может запускать зависимые сервисы и подключаться к внешним сервисам

<https://pypi.org/project/microagent/>

# Сигнал / Signal

- ✓ Моментальное уведомление через шину событий
- ✓ Основан на Publish / Subscribe паттерне
- ✓ Лаконичен, параметры строго определены
- ✓ Не гарантирует доставку и порядок
- ✓ Обеспечивает максимальную скорость, в ущерб надежности

# Сигнал / Signal

```
17 class PostAgent(MicroAgent):  
18 +--- 2 строки: counter = 0-----  
20 → @receiver(signals.post) # Подписываемся на сигнал 'post', timeout по-умолчанию 60 сек  
21 | async def post_handler(self, post, user, **kw):  
22 +----- 4 строки: self.counter += 1-----  
25 ← await self.bus.new_post.send('post_agent', num=self.counter) # Отправляем сигнал 'new_post'  
27 |  
28 |  
29 class UserAgent(MicroAgent):  
30 +---- 11 строк: @on('pre_start')-----  
41 | async def users_watcher(self):  
42 |     users = set(await self.redis.smembers('users'))  
43 |  
44 |     for user in users - self.users:  
45 |         await self.bus.user_enter.send('user_agent', name=user) # Отправляем сигнал 'user_enter'  
46 |         self.log.info('User enter: %s', user)  
47 |  
48 |     for user in self.users - users:  
49 |         await self.bus.user_exit.send('user_agent', name=user) # Отправляем сигнал 'user_exit'  
50 |         self.log.info('User exit: %s', user)  
51 +--- 7 строк: self.users = users-----  
58 |  
59 class MentionAgent(MicroAgent):  
60 +----- 4 строки: @on('pre_start')-----  
64 → @receiver(signals.user_enter, signals.user_exit, timeout=10) # Подписываемся сразу на 2 сигнала  
65 | async def update_users(self, signal, name, **kw):  
66 |     self.log.info('Update users: %s', name)  
67 |  
68 |     if signal == signals.user_enter: # signal - полученный сигнал  
69 |         self.users.add(name)  
70 |  
71 |     if signal == signals.user_exit:  
72 |         self.users.remove(name)
```

# Сообщение / Message

- ✓ Пакет данных из очереди, который требуется обработать
- ✓ Основан на Producer / Consumer паттерне
- ✓ Содержит произвольный набор данных
- ✓ Сообщение храниться в очереди, доставка и порядок гарантированы

# Сообщение / Message

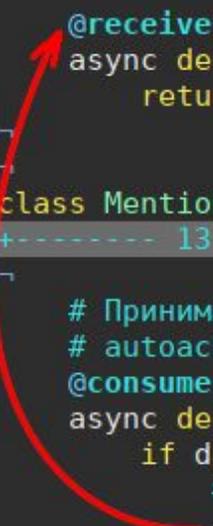
```
17 class PostAgent(MicroAgent):-
18     counter = 0
19
20     @receiver(signals.post)
21     async def post_handler(self, post, user, **kw):-
22         self.counter += 1
23         self.log.info('Catch post: %d %s', self.counter, post)
24         # Отправляем данные поста в очередь 'post'
25         await self.broker.post.send(
26             {'post': post, 'author': user, 'num': self.counter})
27         await self.bus.new_post.send('post_agent', num=self.counter)
28
29 +----- 30 строк: class UserAgent(MicroAgent):-----
59
60 class MentionAgent(MicroAgent):-
61 +----- 13 строк: @on('pre_start')-----
74
75     # Принимаем данные из очереди 'post'
76     # autoack - автоматически удаляем/помечаем обработанным пакет из очереди
77     @consumer(queues.post, autoack=True)
78     async def main_handler(self, *args, **data):-
79 +----- 8 строк: if data['author'] not in self.users:-----
87         for name in self.users:
88             if f'@{name}' in data['post']:
89                 # Отправляем данные уведомления в очередь 'email'
90                 await self.broker.email.send({'user': name, 'post': data['post']})
```

# Вызов / Call

- ✓ Удаленный вызов процедур на базе сигналов (запрос-ответ) с timeout-ом
- ✓ Служит для синхронизации распределенных вычислений
- ✓ Может вернуть простой тип (int, str) в ответ
- ✓ Может получить несколько ответов из разных источников

# Вызов / Call

```
30 class UserAgent(MicroAgent):↵
31 +----- 11 строк: @on('pre_start')-----↵
42     async def users_watcher(self):↵
43         users = set(await self.redis.smembers('users'))↵
44 ↵
45 +----- 3 строки: for user in users - self.users:-----↵
48 ↵
49 +----- 3 строки: for user in self.users - users:-----↵
52 ↵
53     self.users = users↵
54 ↵
55     @receiver(signals.check_user) # Принимаем сигнал 'check_user' и отправляем ответ↵
56     async def check_user(self, user, **kw):↵
57         return user in self.users↵
58 ↵
59 ↵
60 class MentionAgent(MicroAgent):↵
61 +----- 13 строк: @on('pre_start')-----↵
74 ↵
75     # Принимаем данные из очереди 'post'↵
76     # autoack - автоматически удаляем/помечаем обработанным пакет из очереди↵
77     @consumer(queues.post, autoack=True)↵
78     async def main_handler(self, *args, **data):↵
79         if data['author'] not in self.users:↵
80             # Отправляем сигнал 'check_user' и ожидаем первого ответа↵
81             resp = await self.bus.check_user.call('mention_agent', user=data['author'])↵
82 ↵
83             if resp:↵
84                 self.users.add(data['author'])↵
85             else:↵
86                 raise Exception('Unknown user')↵
```



# Периодические задачи

## Periodical, cron

- ✓ Периодически вызываемые функции с timeout-ом и устойчивые к ошибкам
- ✓ Расписание может задаваться через интервал, и по расписанию в формате cron
- ✓ Основан на отложенных вызовах asyncio

# Периодические задачи

## Periodic & cron

```
30 class UserAgent(MicroAgent):↵
31 +----- 10 строк: @on('pre_start')-----↵
41 → @periodic(period=1, timeout=1) # Раз в секунду проверяю данные в редисе, timeout - 1 сек↵
42     async def users_watcher(self):↵
43         users = set(await self.redis.smembers('users'))↵
44 ↵
45         for user in users - self.users:↵
46             await self.bus.user_enter.send('user_agent', name=user) # Отправляем сигнал 'user_enter'↵
47             self.log.info('User enter: %s', user)↵
48 ↵
49         for user in self.users - users:↵
50             await self.bus.user_exit.send('user_agent', name=user) # Отправляем сигнал 'user_exit'↵
51             self.log.info('User exit: %s', user)↵
52 ↵
53         self.users = users↵
54 +----- 5 строк: @receiver(signals.check_user) Принимаем сигнал 'check_user' и отправл↵
59 ↵
60 +--- 33 строки: class MentionAgent(MicroAgent):-----↵
93 ↵
94 class StatsAgent(MicroAgent):↵
95     posts_qty = [0, 0]↵
96 ↵
97 +--- 3 строки: @receiver(signals.new_post)-----↵
100 ↵
101 → @cron('0 */4 * * *', timeout=30) # Раз в 4 часа создаем отчет с собранной статистикой↵
102     async def send_report(self):↵
103         text = 'Get %d new posts by 4 hours' % (self.posts_qty[0] - self.posts_qty[1])↵
104         self.posts_qty[1] = self.posts_qty[0]↵
```

# Зависимые функции / Hook

- ✓ Пред-, пост- обработчики, обработчики ошибок
- ✓ Запускаются на запуск / остановку микроагента, вызов функций
- ✓ Позволяют инициализировать пулы соединений, подключаться к портам, собирать статистику и т.п.

# Зависимые функции / Hook

```
30 class UserAgent(MicroAgent):  
31 → @on('pre_start') # Вызывается перед подключение обработчиков к шине / брокеру  
32     async def setup(self):  
33         self.redis = await aioredis.create_redis('redis://localhost:6379/7')  
34         self.users = set(await self.redis.smembers('users'))  
35       
36 → @on('post_stop') # Вызывается после отключения обработчиков от шины / брокера  
37     async def shutdown(self):  
38         self.redis.close()  
39         await self.redis.wait_closed()  
40       
41     @periodic(period=1, timeout=1) # Раз в секунду проверяю данные в редисе, timeout - 1 сек  
42     async def users_watcher(self):  
43         users = set(await self.redis.smembers('users'))  
44       
45 +----- 3 строки: for user in users - self.users:-----  
48       
49 +----- 3 строки: for user in self.users - users:-----  
52       
53         self.users = users  
54       
55 → @on('error_users_watcher') # Вызывается при исключении в методе users_watcher  
56     async def err_sender(self, *args, **kwargs):  
57         self.log.error('Error %s %s', args, kwargs)  
58       
59 +----- 3 строки: @receiver(signals.check_user) Принимаем сигнал 'check_user' и отпр  
62       
63       
64 class MentionAgent(MicroAgent):  
65 → @on('pre_start')  
66     async def setup(self):  
67         self.users = set()
```

# Шина сигналов / SignalBus

```
from microagent import load_stuff↵
from microagent.tools.aioredis import AIORedisSignalBus # aioredis based backend↵
from microagent.tools.pulsar import RedisSignalBus # pulsar based backend↵
↵
from agents import UserAgent↵
↵
↵
# Загружаем доступные сигналы и очереди из файла↵
cur_dir = os.path.dirname(os.path.realpath(__file__))↵
signals, queues = load_stuff('file://' + os.path.join(cur_dir, 'signals.json'))↵
↵
↵
async def main():↵
    # Инициализируем шины с разными бекендами↵
    aioredis_bus = AIORedisSignalBus('redis://localhost/7', prefix='MYAPP')↵
    pulsar_bus = RedisSignalBus('redis://localhost/7', prefix='MYAPP')↵
    ↵
    # Используем шину для отправки сигналов независимо от микроагента↵
    await pulsar_bus.started.send('user_agent')↵
    ↵
    agent = UserAgent(bus=aioredis_bus) # Инициализируем микроагент и ...↵
    await agent.start() # ... запускаем↵
    ↵
    ↵
if __name__ == '__main__':↵
    loop = asyncio.get_event_loop()↵
    loop.call_soon(asyncio.ensure_future(main))↵
    loop.run_forever()↵
```

```
{↵
    "version": 1,↵
    "signals": [↵
        {"name": "post", "providing_args": ["post", "user"]},↵
        {"name": "user_enter", "providing_args": ["name"]},↵
        {"name": "user_exit", "providing_args": ["name"]},↵
        {"name": "new_post", "providing_args": ["num"]},↵
        {"name": "check_user", "providing_args": ["user"]}↵
    ],↵
    "queues": [↵
        ↵
    ]↵
}
```

+---- 4 строки: "queues": [-----

# Брокер очередей / QueueBroker

```
from microagent import MicroAgent, periodic, receiver, consumer, load_stuff
from microagent.tools.aioredis import AIORedisSignalBus, AIORedisBroker
from microagent.tools.amqp import AMQPBroker
from microagent.tools.kafka import KafkaBroker
-
-
from agents import MentionAgent
-
-
# Загружаем доступные сигналы и очереди из файла
cur_dir = os.path.dirname(os.path.realpath(__file__))
signals, queues = load_stuff('file://' + os.path.join(cur_dir, 'signals.json'))
-
-
async def main():
    aioredis_bus = AIORedisSignalBus('redis://localhost/7', prefix='MYAPP')
    -
    # Инициализируем брокеры с разными бекендами
    aioredis_broker = AIORedisBroker('redis://localhost/7')
    amqp_broker = AMQPBroker('amqp://user:31415@localhost:5672/prod')
    kafka_broker = KafkaBroker('kafka://localhost:9092')
    -
    # Используем шину для отправки сообщений независимо от микроагента
    await aioredis_broker.started.send({'service_name': 'my_app'})
    -
    agent = MentionAgent(bus=aioredis_bus, broker=aioredis_broker)
    await agent.start() # запускаем
    -
    -
if __name__ == '__main__':
    loop = asyncio.get_event_loop()
    loop.call_soon(asyncio.ensure_future(main))
    loop.run_forever()
```

```
{
  "version": 1,
+--- 7 строк: "signals":
  "queues": [
    {"name": "post"},
    {"name": "email"}
  ]
}
```

# Экосистема микроагентов

- Микроагент упрощает интеграцию предоставляя готовый набор интерфейсов
- Обмен данными создает зависимости, но должен быть односторонним, лучше асинхронным и через посредника
- Общие хранилища — часть экосистемы, они не являются антипаттерном, но требуют аккуратного обращения
- Периодические функции позволяют передавать инициативу (поток) управления микроагентам

**ВСЁ!**

# Источники

- Бёрнс Б. Распределенные системы. Паттерны проектирования. — СПб.: Питер, 2019. — 224 с.: ил. — (Серия «Бестселлеры O'Reilly»).
- Ньюмен С. Создание микросервисов. — СПб.: Питер, 2016. — 304 с.: ил. — (Серия «Бестселлеры O'Reilly»).
- Манифест Реактивных Систем — 16 сентября 2014 — <https://www.reactivemanifesto.org/ru>
- Noonan A., Goodbye Microservices: From 100s of problem children to 1 superstar, 10 июля 2018 — <https://segment.com/blog/goodbye-microservices/>
- Иванов Д., Реактивные микросервисы с Apache Kafka, HighLoad++ 2017 — <https://www.youtube.com/watch?v=rCOKmZ8VqCU>
- Иванов Д., Строим распределённое реактивное приложение и решаем задачи согласованности — 8 февраля 2018 — <https://habr.com/ru/company/2gis/blog/348510/>
- Столяров Д., Микросервисы: размер имеет значение, даже если у вас Kubernetes — 12 октября 2018 — <https://habr.com/ru/company/flant/blog/424531/>
- Волошин М. Микросервисы на практике — 31 августа 2015 — <https://www.youtube.com/watch?v=QmGe8Zpm6uk>
- Орлов С. Микросервисная Архитектура: проблемы и решения — 5 июня 2017 — <https://www.youtube.com/watch?v=GIslwdE2DWY>



- Микроагент на PyPI

<https://pypi.org/project/microagent/>

- Наш уютный чат

<https://app.pararam.io/>

