5. Непрямое взаимодействие

Сухорослов Олег Викторович

02.10.2023

План лекции

- Непрямое взаимодействие
- Очередь сообщений
- Издатель-подписчик
- Распределенная общая память
- Пространство кортежей

All problems in computer science can be solved by another level of indirection.

David Wheeler

Непрямое взаимодействие

- Indirect communication, косвенное взаимодействие…
- Происходит через некоторого посредника или абстракцию, без прямого связывания между взаимодействующими процессами
- Позволяет упростить решение типовых задач в РС
  - Распространение информации
  - Интеграция компонентов и систем
  - Распределение работы (балансировка нагрузки)
  - Координация между процессами
  - Обновление частей системы
  - …

Связывание процессов

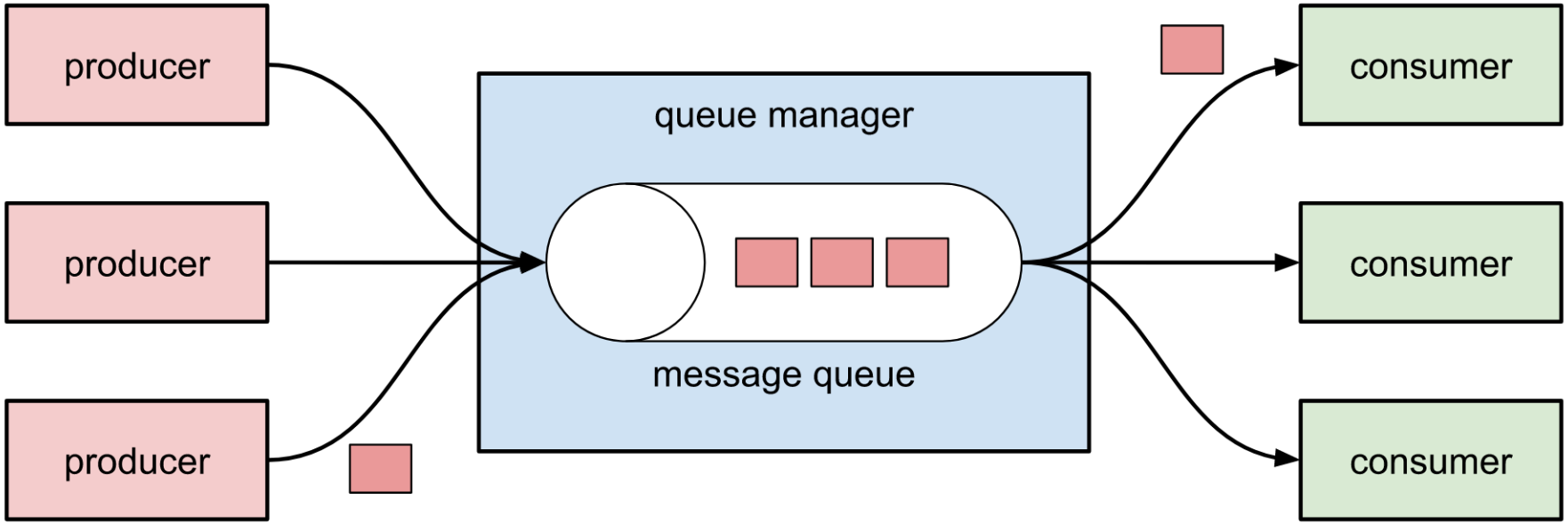
- Связывание по пространству (space coupling)
  - Процессы должны обладать информацией друг о друге
- Связывание по времени (time coupling)
  - Процессы должны выполняться в одно время

	Time Coupling	**Time Uncoupling**
Space Coupling	Message passing, RPC	???
Space Uncoupling	IP multicast	Indirect communication

Модели непрямого взаимодействия

- Групповые взаимодействия (group communication)
  - см. предыдущую лекцию
- Очередь сообщений (message queue)
- Издатель-подписчик (publish-subscribe)
- Распределенная общая память (distributed shared memory)
- Пространство кортежей (tuple space)

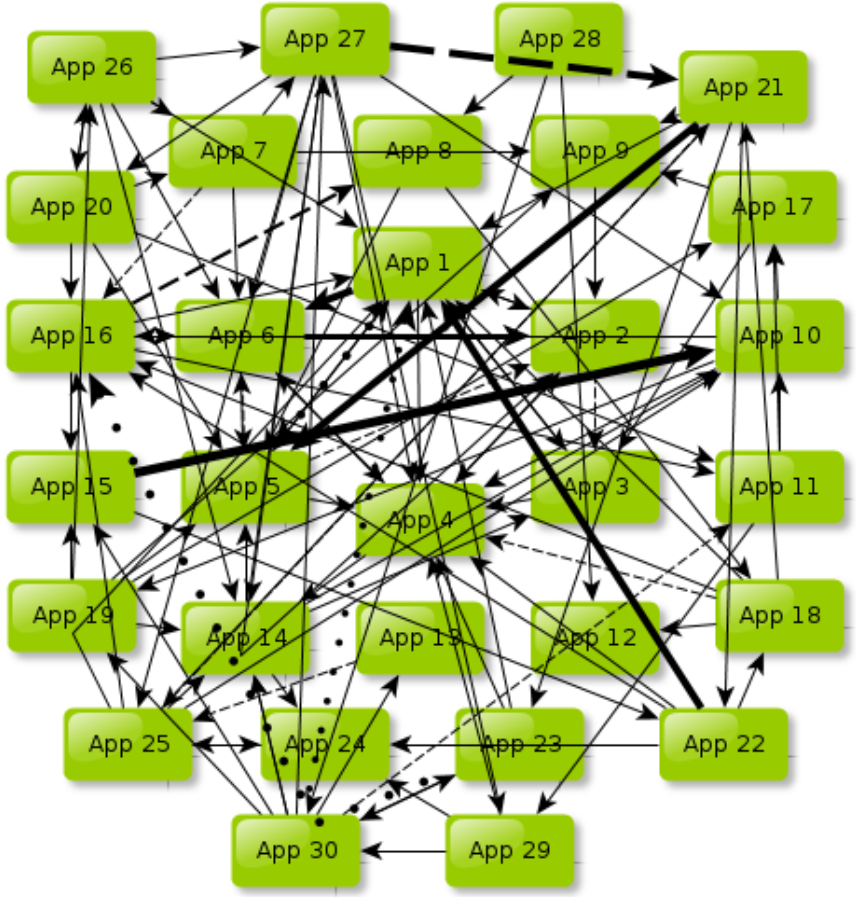
Очередь сообщений



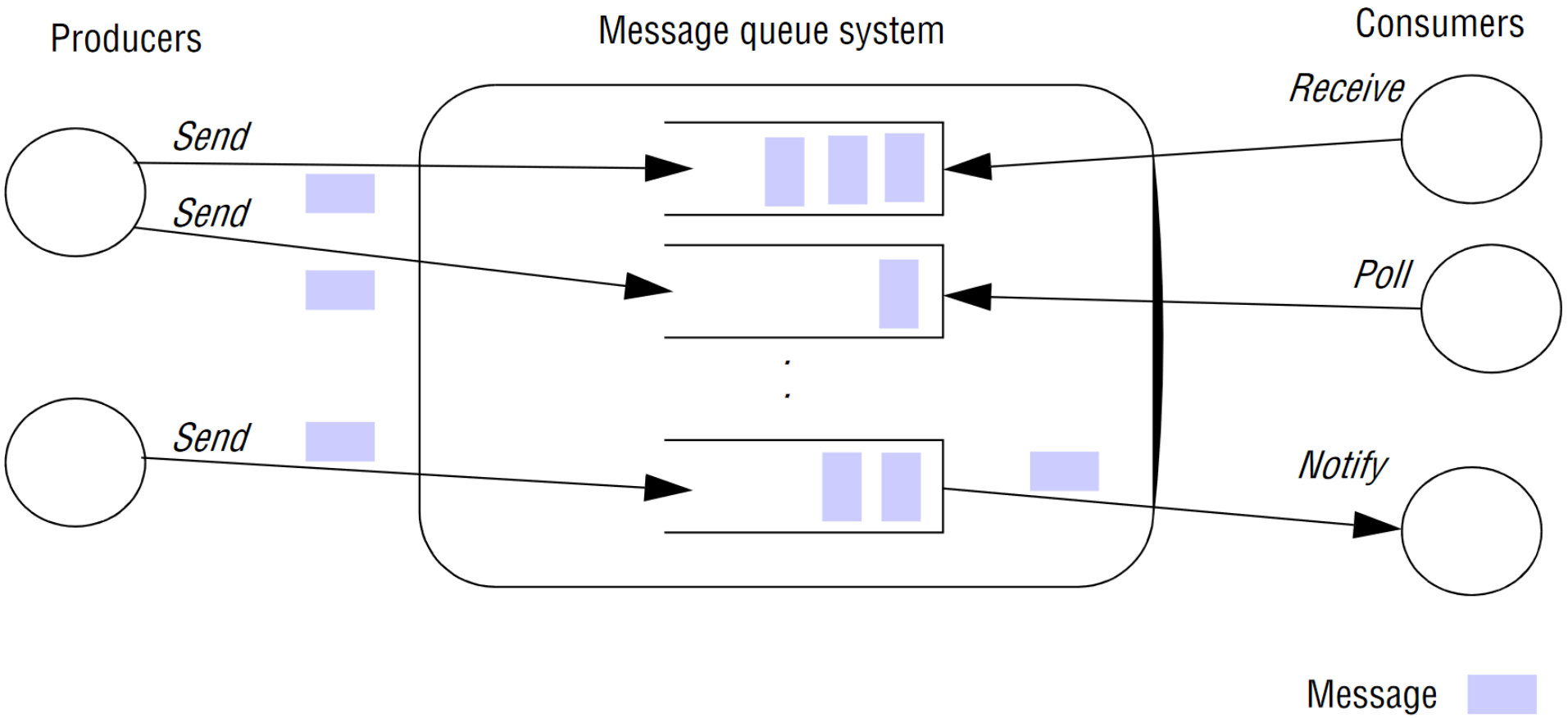
Непрямое взаимодействие на основе абстракции очереди сообщений между отправителями и получателями.

Применение

- Интеграция систем и приложений внутри организации
- Асинхронный обмен сообщениями между компонентами системы
- Распределение заданий между несколькими обработчиками



Модель программирования



Свойства

- Получатель сообщения только один (one-to-one)
- Хранение сообщения до момента его получения (persistent)
- Поддержка асинхронных взаимодействий, схем push и pull

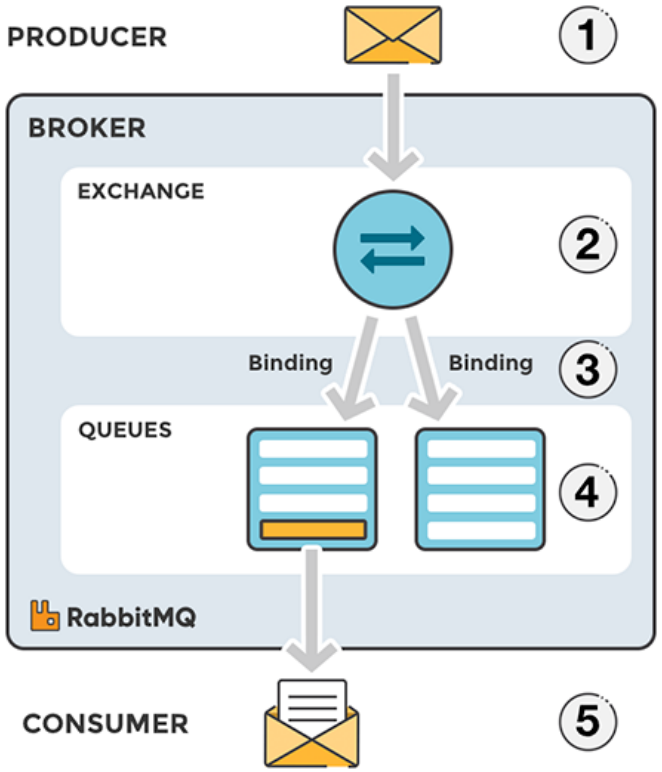
Гарантии

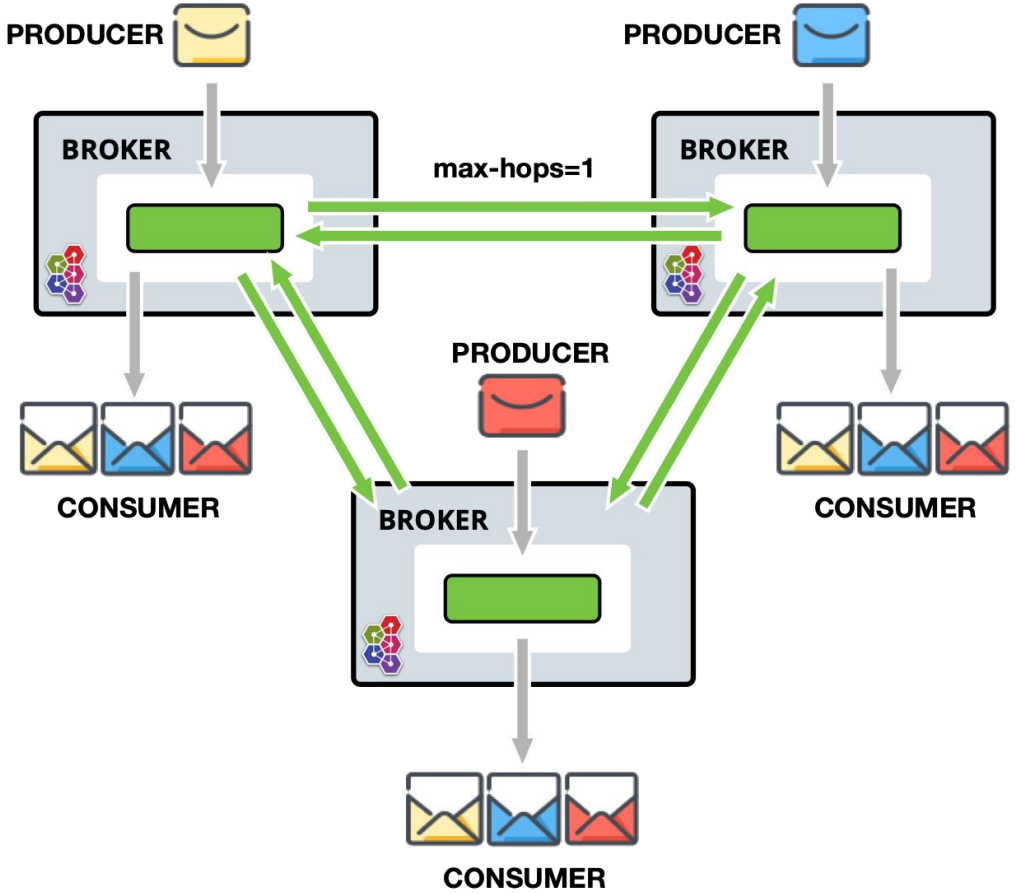
- Сообщение может прочитать только один получатель
- Гарантии по доставке и **обработке** сообщений
- Сохранение порядка сообщений
- Надежное хранение сообщений в процессе доставки

Дополнительные возможности

- Приоритеты сообщений
- Фильтрация сообщений на основе атрибутов
- Преобразование сообщений, конвертация между форматами
- Отправка и получение сообщений в рамках транзакций
- Безопасность

Архитектура

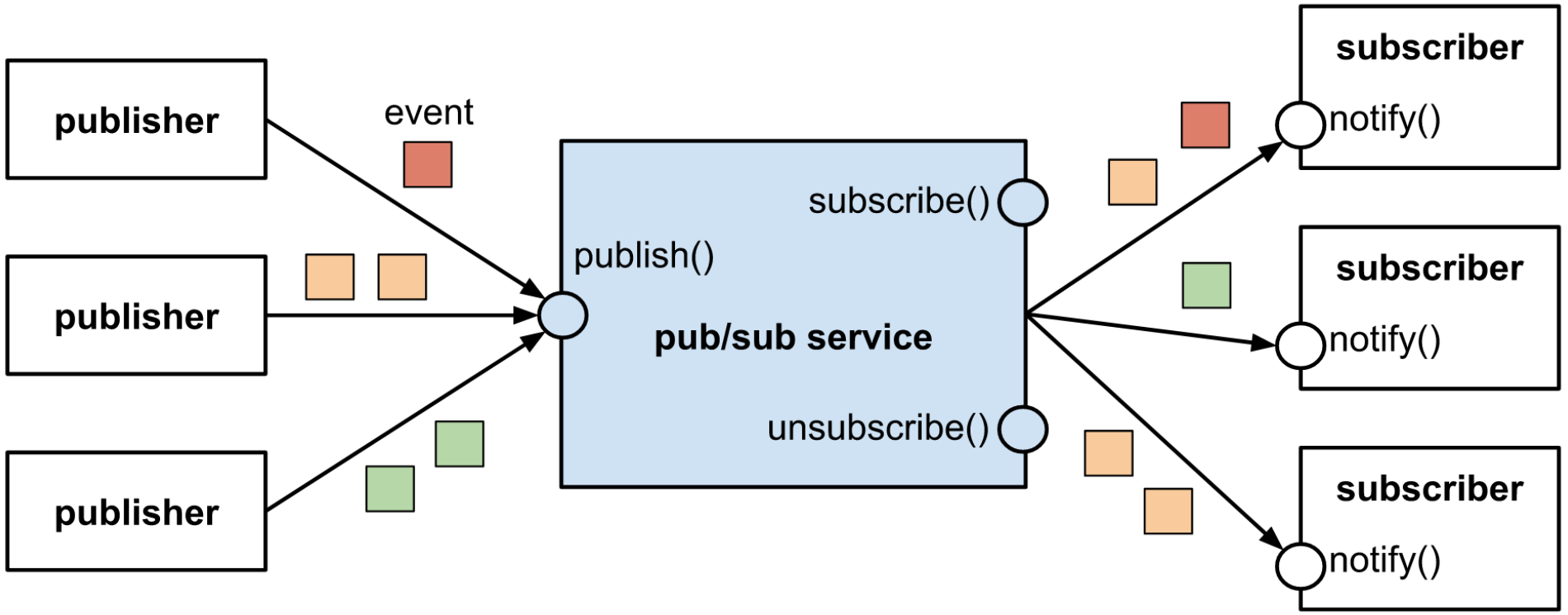




Примеры реализаций

- Enterprise
  - IBM MQ, Java Message Service (JMS)
- Advanced Message Queuing Protocol (AMQP)
  - Открытый протокол для передачи сообщений в MQ-системах
  - Реализации: RabbitMQ, Apache ActiveMQ, Apache Qpid
- Task/Job queues
  - Gearman, Redis
- Облачные сервисы
  - Amazon Simple Queue Service, Yandex Message Queue…

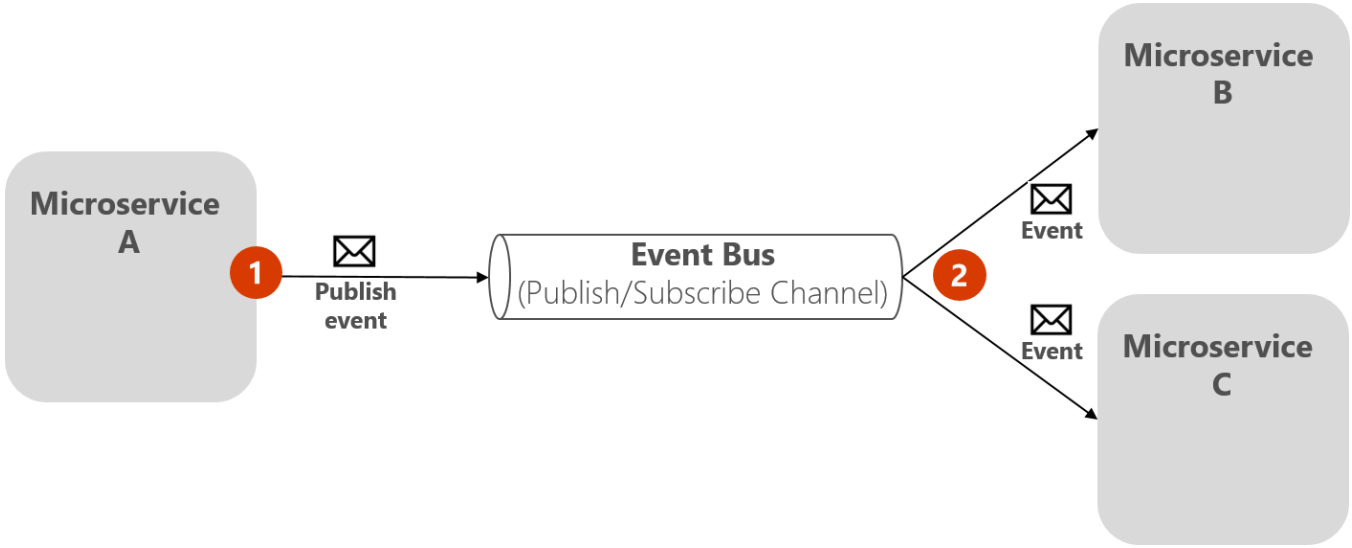
Издатель-подписчик (publish-subscribe)



Непрямое взаимодействие на основе подписки на события, публикуемые компонентами системы.

Применение

- Везде, где требуется организовать передачу и обработку событий
  - Финансовые системы, совместное редактирование, мониторинг...
- Интеграция систем и приложений внутри организации
- Асинхронный обмен сообщениями между компонентами системы

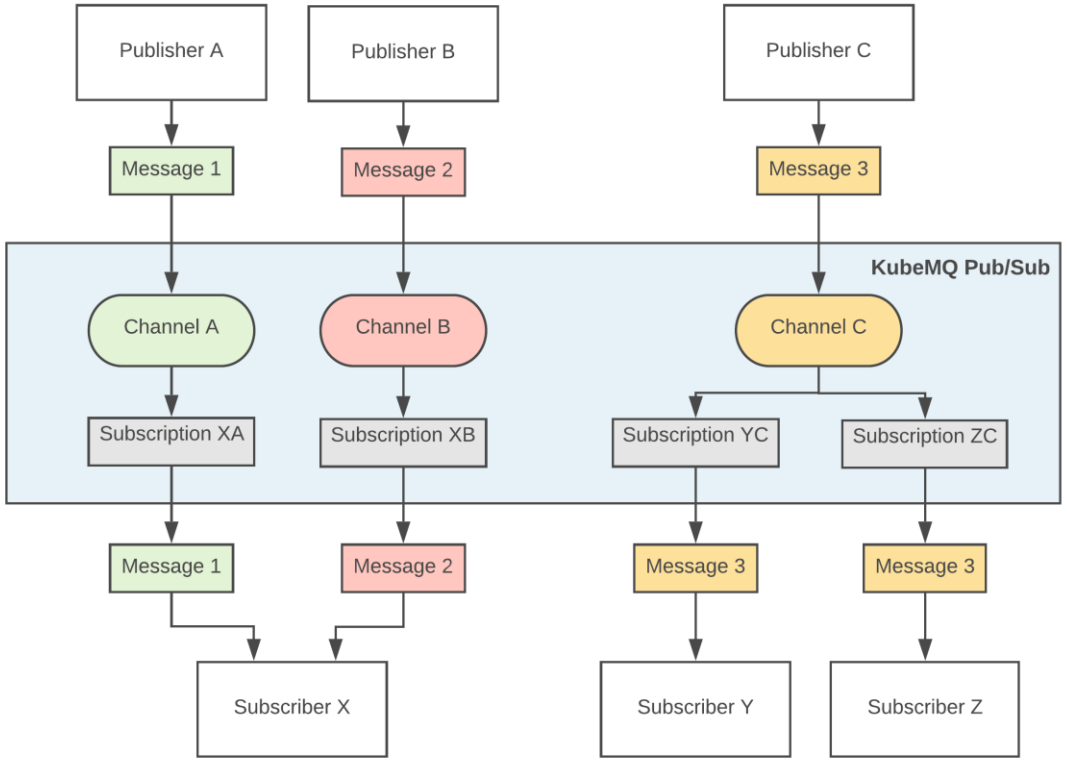


Модель программирования

- **`publish(event)`**
  - опубликовать событие
- **`subscribe(filter, handler)`**
  - подписаться на получение событий по заданному фильтру
- **`unsubscribe(filter)`**
  - отменить подписку на события
- **`advertise(filter)`**
  - распространить информацию о публикуемых типах событий
- **`unadvertise(filter)`**
  - отозвать объявление о публикуемых типах

Модели фильтрации событий

- Канал (channel-based)
- Тема (subject-based)
- Содержимое (content-based)
- Тип (type-based)
- Конкретный объект
- Контекст (местоположение)
- Complex event processing



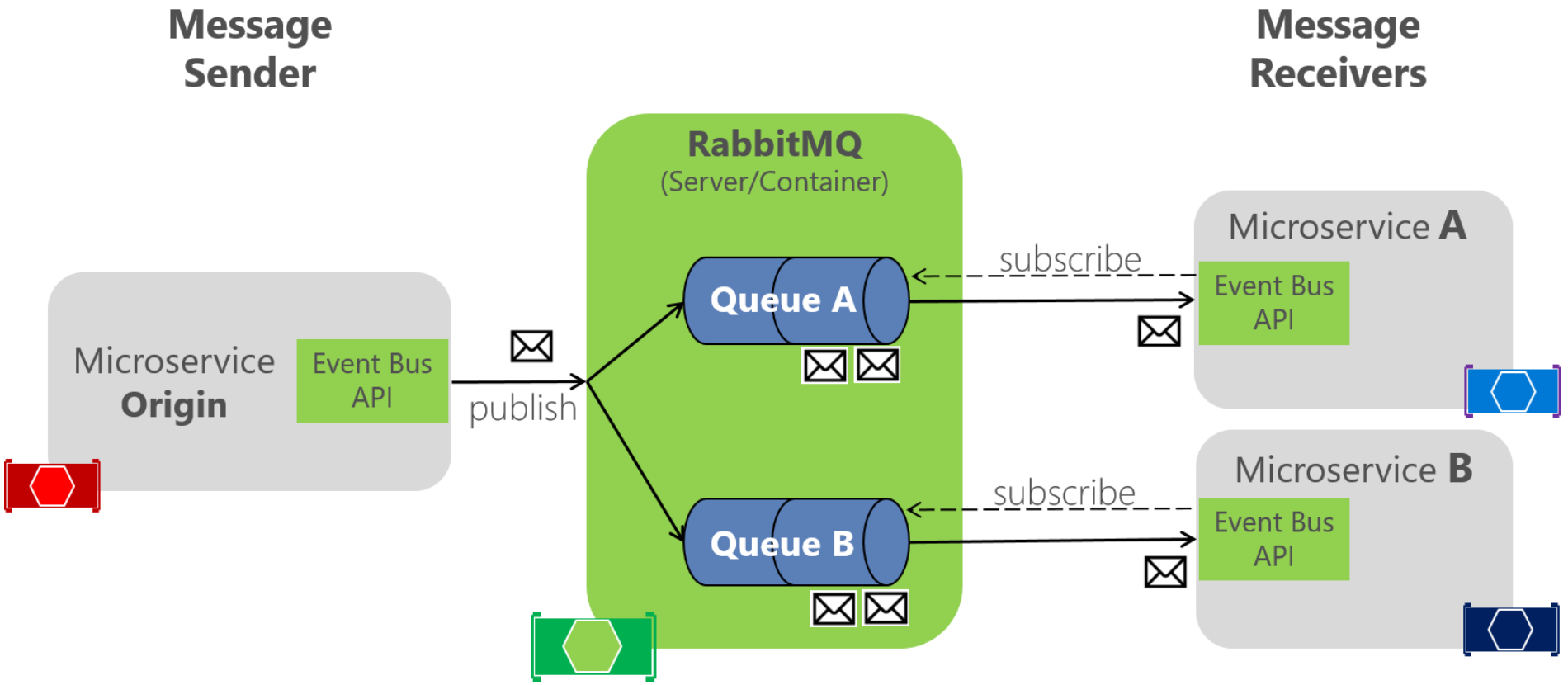
Свойства

- Обеспечивает space uncoupling
  - достаточно согласовать типы и атрибуты событий
- Возможно обеспечение time uncoupling
  - хранение сообщения до его востребования
- Получателей сообщения может быть несколько (one-to-many)
- Поддержка асинхронных взаимодействий, только push

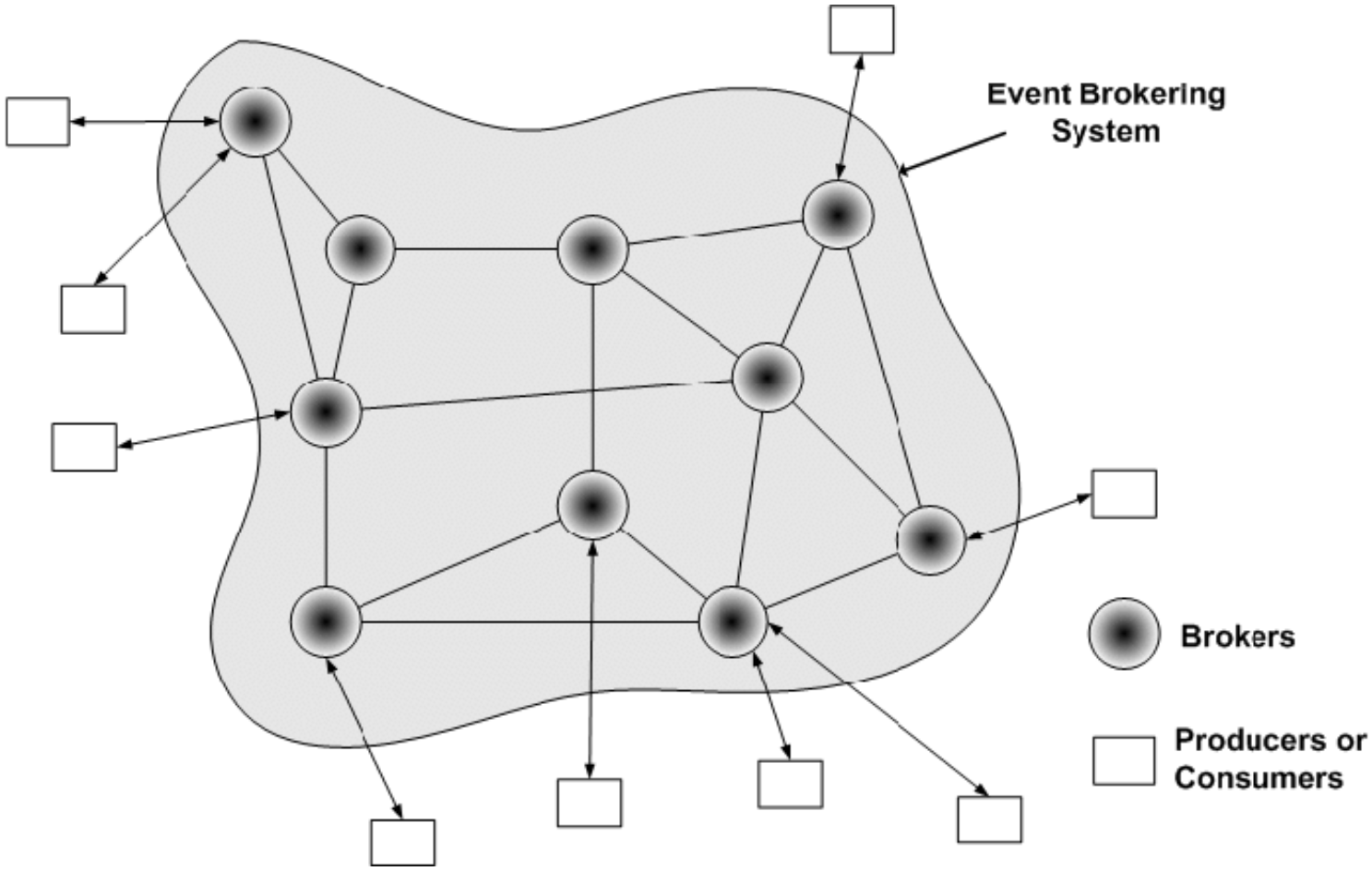
Гарантии

- Гарантии доставки и сохранения порядка событий
- Надежное хранение событий в процессе доставки
- Доставка пропущенных событий отключавшимся подписчикам

Централизованная архитектура

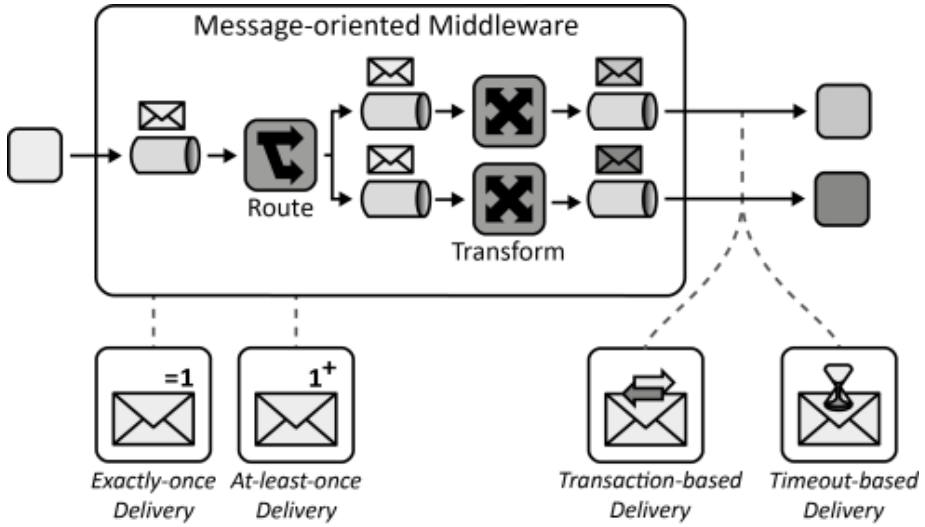


Децентрализованная архитектура



Message-oriented Middleware

- Очереди сообщений и издатель-подписчик похожи и часто реализуются вместе с сопутствующим функционалом в рамках одной технологии промежуточного ПО
  - один или несколько подписчиков
  - push и pull
- Примеры
  - IBM MQ, TIBCO Enterprise Message Service
  - RabbitMQ, ActiveMQ
  - Google Cloud Pub/Sub, Azure Service Bus
  - https://taskqueues.com/

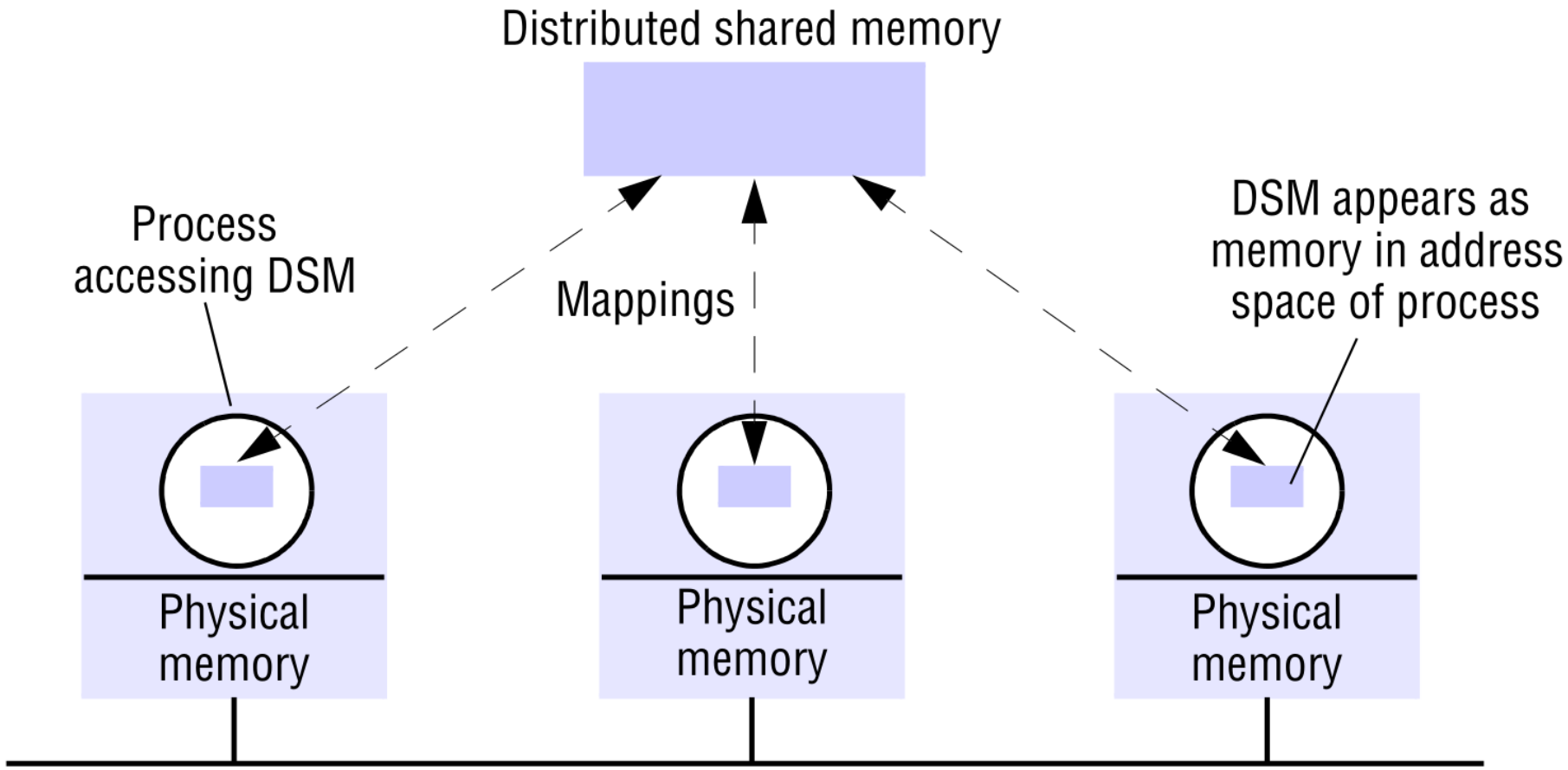


Общая память

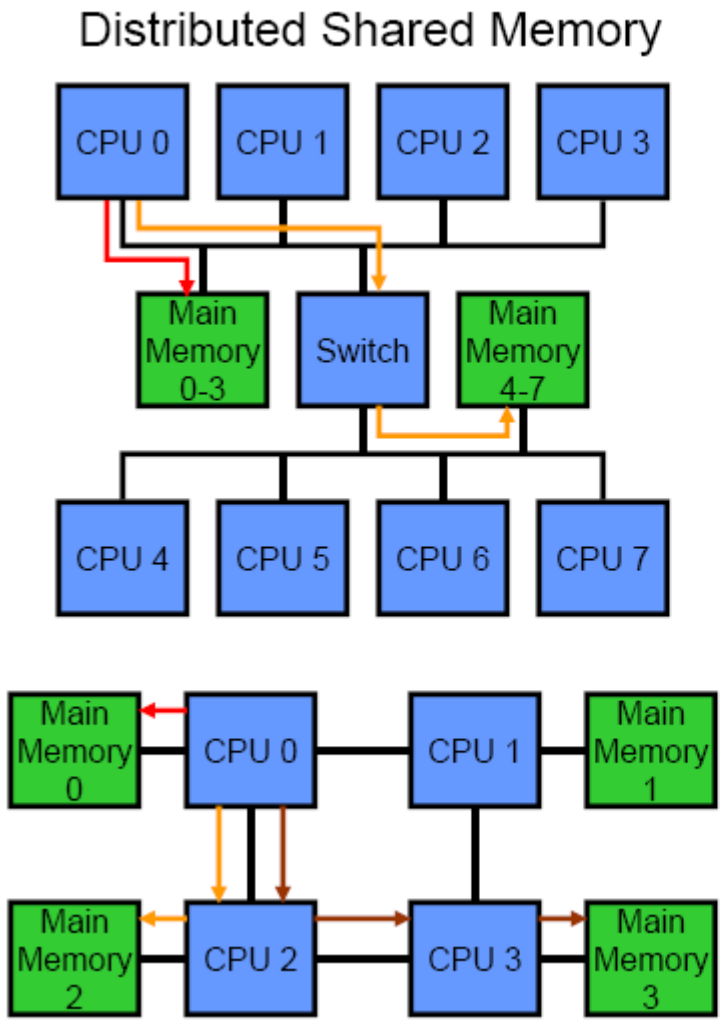
Формы непрямого взаимодействия на основе “общей памяти”:

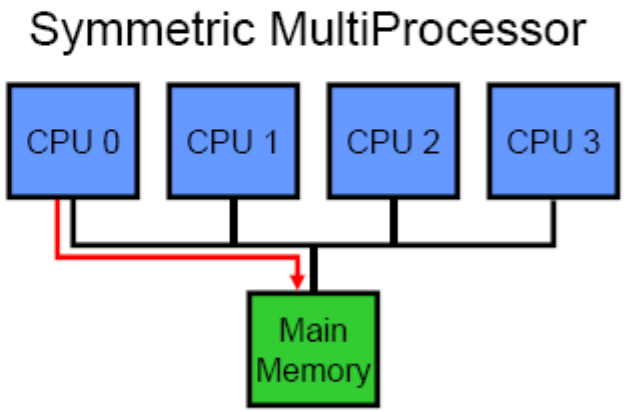
- Распределенная общая память (distributed shared memory)
- Пространство кортежей (tuple space)

Распределенная общая память



Многопроцессорные системы





Использование в PC

- Обеспечение прямого доступа к разделяемым данным
- Параллельные вычисления
  - Работа с общими структурами данных, тесная координация процессов
  - Перенос существующих программ с одной машины на кластер
- Клиент-серверные приложения
  - Размещение копий данных в кэше на стороне клиента
  - Не всегда подходит в силу требований инкапсуляции и безопасности

Модель программирования

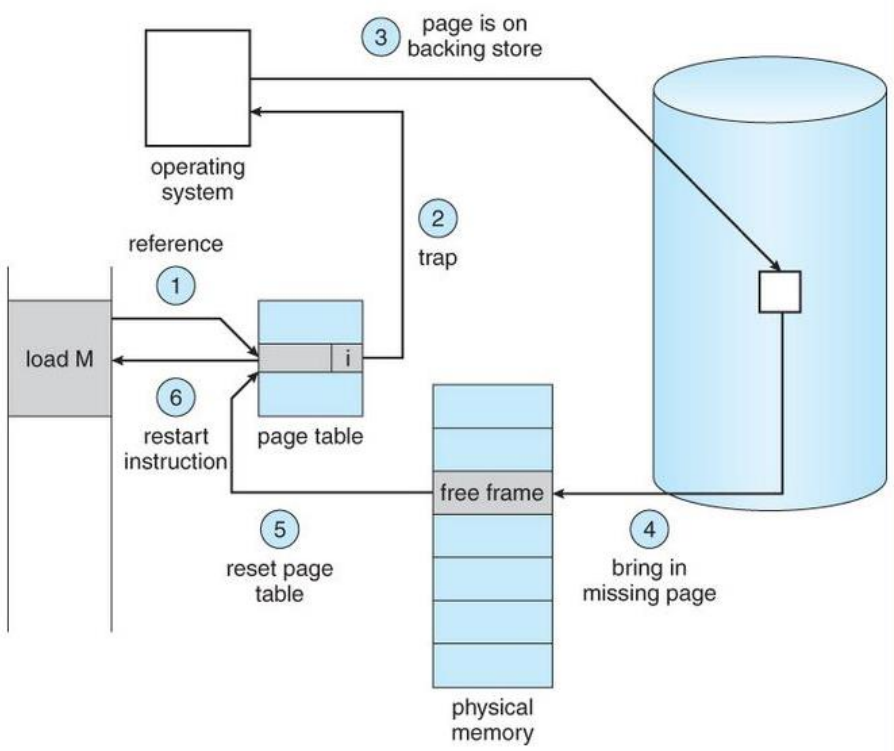
- Чтение и запись данных в память по адресу
  - **write/read** вместо **send/receive**
- Нет явного обмена сообщениями и (де)маршаллинга данных
- Синхронизация с помощью блокировок и аналогичных примитивов
- Позволяет запускать существующие программы без их модификации

Свойства

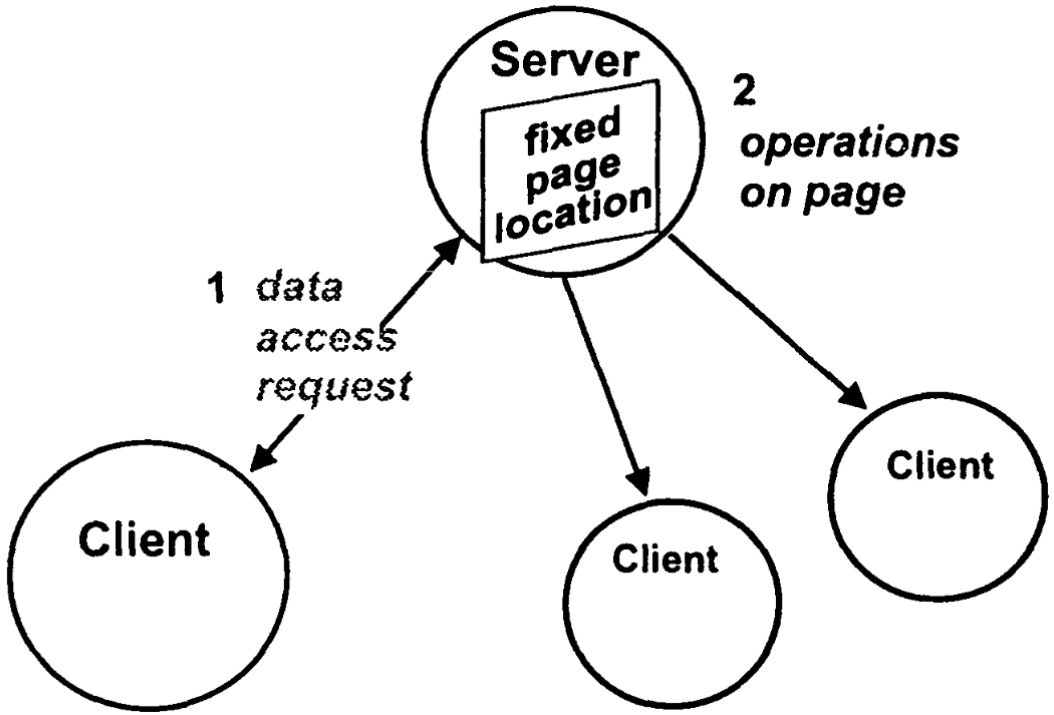
- Обеспечивает space и time uncoupling
- Поддержка асинхронных взаимодействий, только pull
- Взаимодействия one-to-one и one-to-many

Реализация

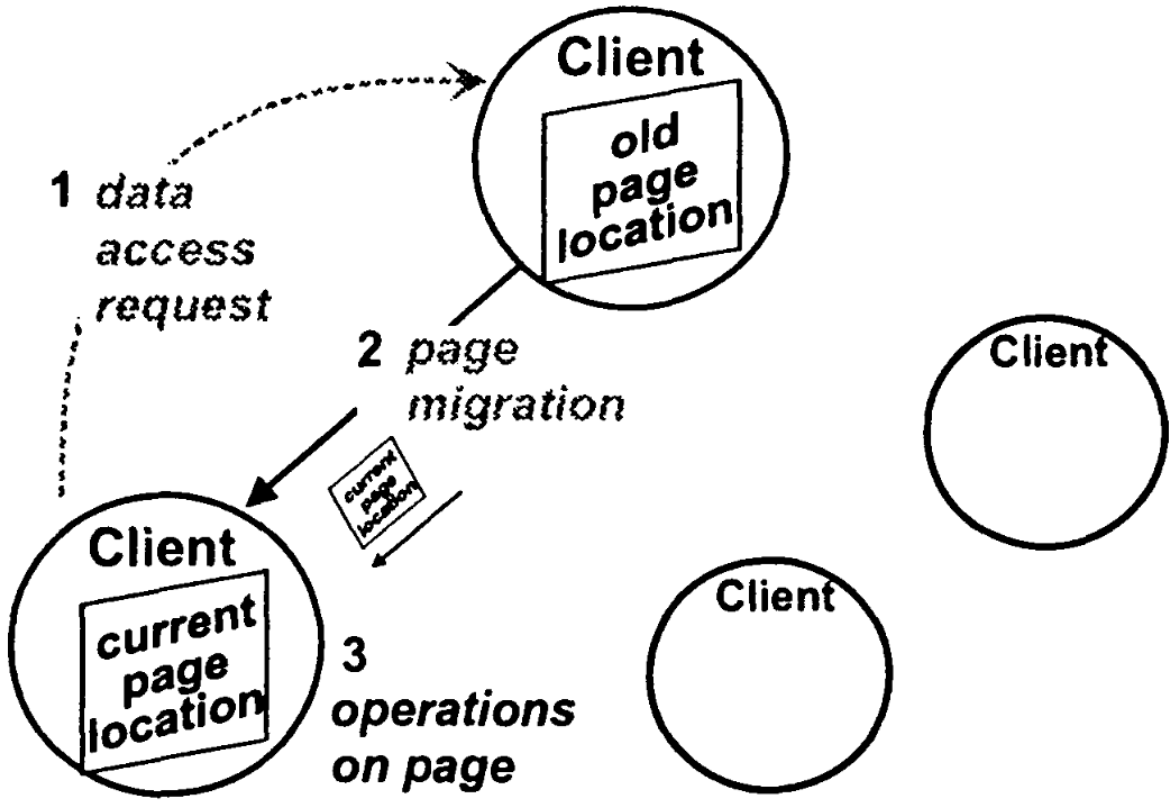
- Основные подходы
  - На основе виртуальной памяти (page based)
  - Операции над разделяемыми переменными или объектами
- Структура памяти
  - Массив байтов или слов, доступ по адресу
  - Коллекция объектов, доступ через методы
  - Неизменяемые данные (см. пространство кортежей)



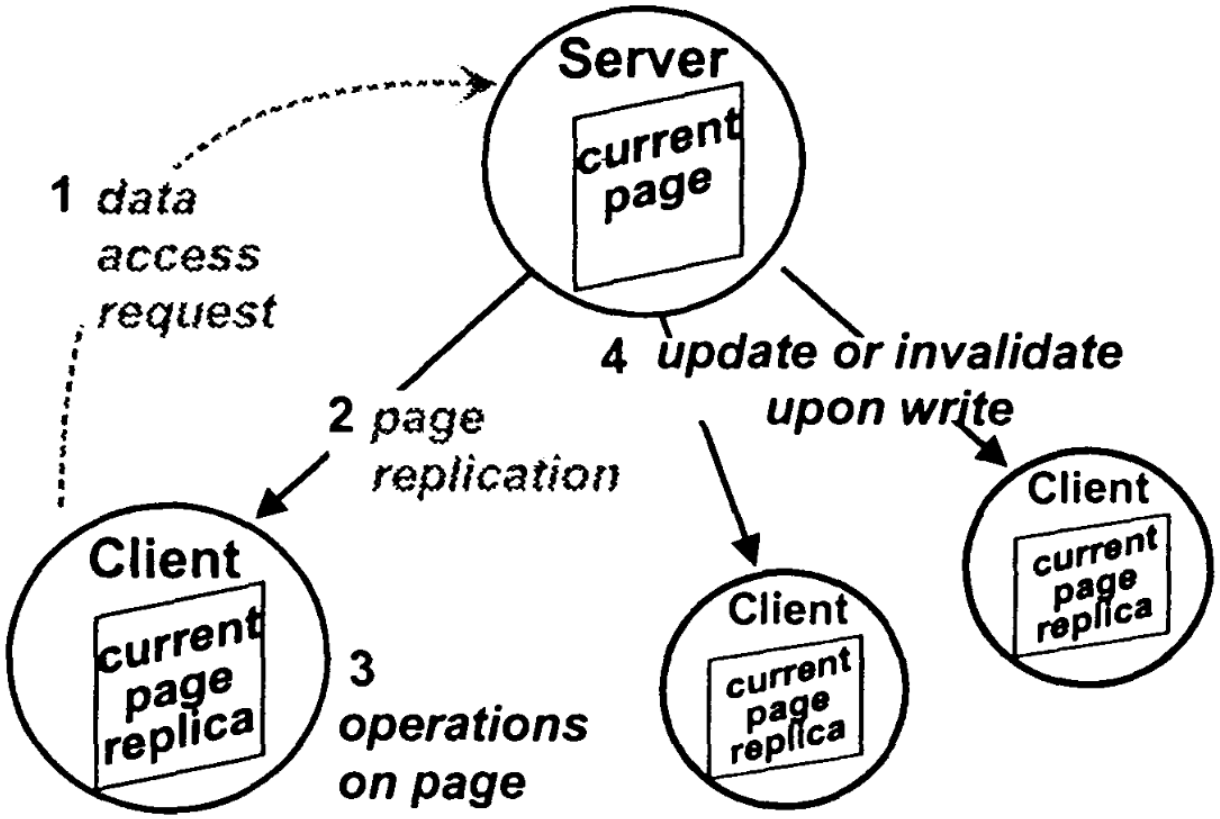
Архитектура 1 (центральный сервер)



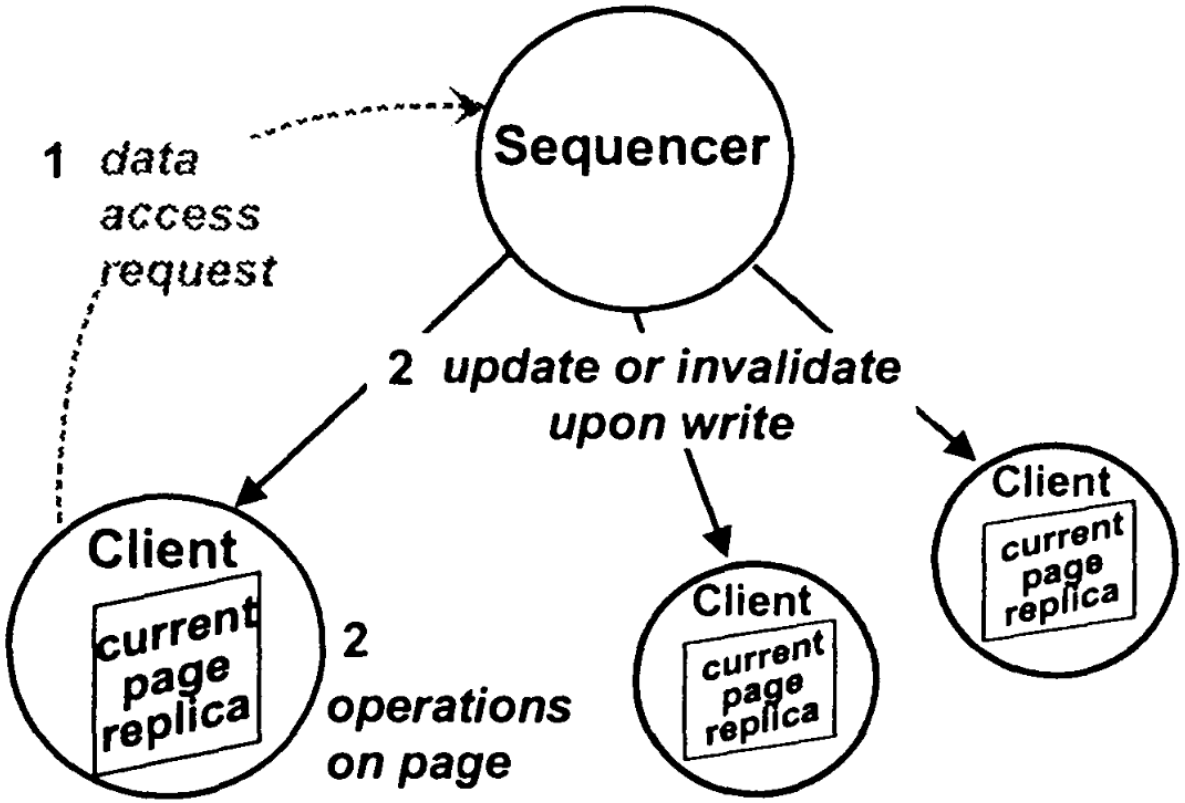
Архитектура 2 (миграция страниц)



Архитектура 3 (репликация для чтения)



Архитектура 4 (полная репликация)



Распространение изменений

- Подход Write-Invalidate
  - копии данных инвалидируются перед записью
  - изменения доставляются при чтении (pull)
  - дорогие чтения в случае интенсивной записи
  - проблема thrashing
- Подход Write-Update
  - изменения сразу применяются ко всем копиям данных (push)
  - дешевые чтения, дорогая запись

Гранулярность памяти

- Размер блока памяти, передаваемого по сети
- В классических системах - страница
- Какой размер страницы выбрать?

Другие аспекты реализации

- Синхронизация при доступе к общим данным
  - Критические секции, блокировки
- Модель согласованности (memory consistency model)
  - Гарантии относительно читаемых процессами значений из памяти
  - Определяет порядок, в котором процессы видят операции записи
  - Относится к нескольким объектам в памяти, а не к одному (coherence)
  - Чем строже модель, тем “дороже” она в плане сложности реализации, производительности, масштабируемости и т.д.

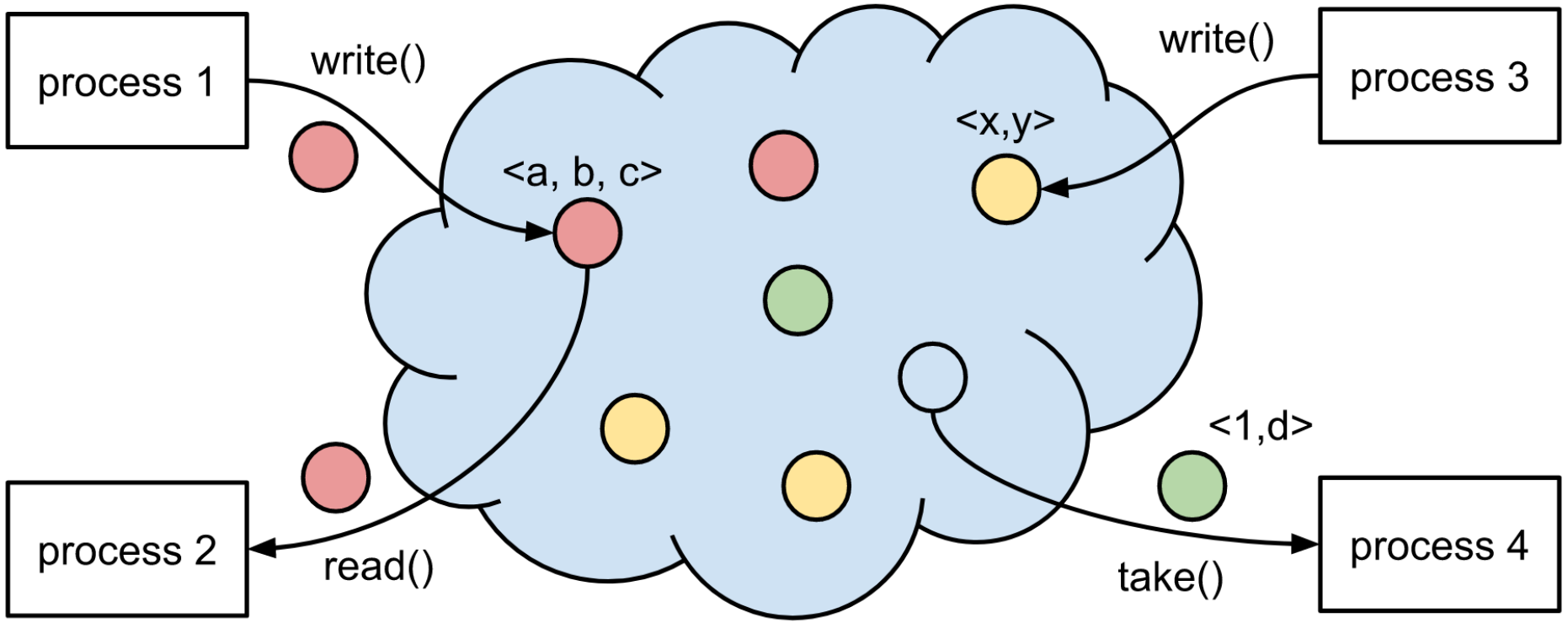
Примеры реализации DSM

- Экспериментальные системы 1980/90-х годов
  - Ivy, Mirage, Munin, Mether, Midway, TreadMarks...
  - На системах небольшого размера (~10) показана производительность на уровне реализаций message passing
- См. литературу

Проблемы DSM

- Низкий уровень изоляции процессов
- Синхронизация при работе с общими данными
- Поддержка гетерогенных систем
- Накладные расходы скрыты от программиста
- Эффекты типа thrashing
- Ограниченная масштабируемость

Пространство кортежей



Gelernter D. Generative communication in Linda (1985)

Модель программирования

- Кортеж
  - упорядоченный набор типизированных значений
  - `<"job", 12, 1.23>`
- Ассоциативная память
  - доступ к кортежам не по адресу, а по их содержимому и типу
- Шаблон
  - `<string, int, float>`
  - `<"job", 12, float>`

Модель программирования

- Операции
  - **`write(tuple)`** - добавление кортежа в пространство (push)
  - **`read(template)`** - чтение кортежа по шаблону (блокирует, pull)
  - **`take(template)`** - чтение с удалением кортежа (блокирует, pull)
  - **`try_read/try_take(template)`** - неблокирующие варианты
- Кортежи являются неизменяемыми сущностями
  - повторная запись добавит новый кортеж
  - для изменения кортежа требуется выполнить **`take`** и **`write`**

```
<s, count> = myTS.take(<"counter", int>)
myTS.write(<"counter", count+1>)
```

Свойства

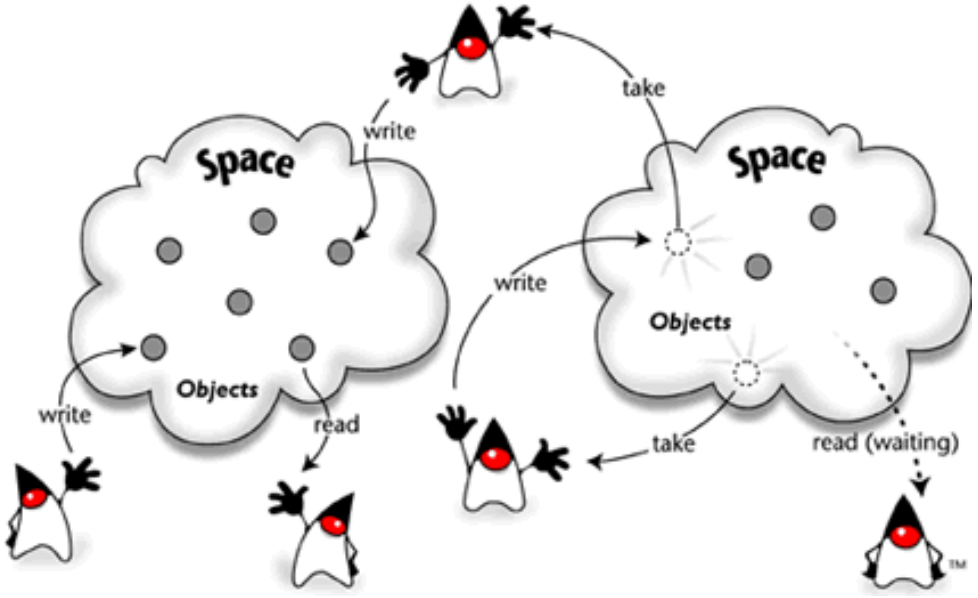
- Синхронные и асинхронные взаимодействия, pull и push
- Взаимодействия one-to-one и one-to-many

Расширения

- Несколько пространств вместо одного глобального
- Распределенные реализации
  - Репликация данных
  - Разбиение данных (шардинг)
  - Peer-to-peer
- Модификации набора операций
- Хранение объектов (object spaces)

JavaSpaces

- Разработка Sun Microsystems (1998)
- Динамически создаваемые пространства
- В пространстве хранятся записи (entry)
- С записью связана аренда (lease) с продлеваемым сроком
- Таймаут при чтении, неблокирующие операции
- Уведомления о новых записях (notify)
- Транзакции



Пространство кортежей

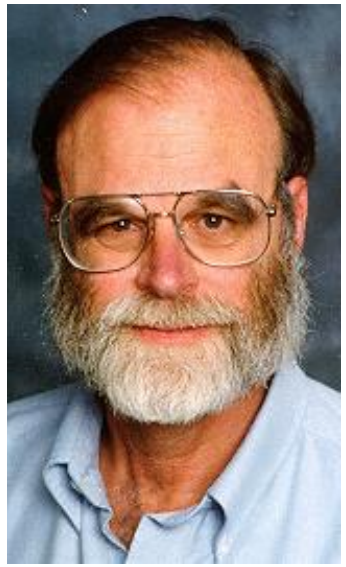
- Преимущества
  - Простая, гибкая и выразительная модель программирования
  - Привычная абстракция общей памяти
  - Отсутствует необходимость синхронизации
- Недостатки
  - Ограниченная масштабируемость (операция take)
  - Мало реализаций

Современные “аналоги” общей памяти

- Разделённое глобальное адресное пространство (PGAS)
- Удалённый прямой доступ к памяти (RDMA)
- Отказоустойчивые хранилища данных
- Отказоустойчивые сервисы координации

Модели непрямого взаимодействия

	Groups	MQ	Pub-Sub	DSM	Tuple Space
Space uncoupled	Yes	Yes	Yes	Yes	Yes
Time uncoupled	Possible	Yes	Possible	Yes	Yes
Style	Comm.-based	Comm.-based	Comm.-based	State-based	State-based
Comm. Pattern	1-to-many	1-to-1	1-to-many	1-to-many	1-to-1, 1-to-many
Push/Pull	Push/Pull	Push/Pull	Push	Pull	Push/Pull
Main intent	Reliable distributed computing	Information dissemination, integration, transaction processing, load balancing	Information dissemination, integration, mobile computing	Parallel and distributed computing	Parallel and distributed computing, mobile computing
Scalability	Possible	Possible	Possible	Limited	Limited



There is no performance problem that cannot be solved by eliminating a level of indirection.

Jim Gray

Материалы

- Distributed Systems: Concepts and Design (разделы 6.1, 6.3-6.5)
  - плюс глава Distributed Shared Memory
- Distributed Systems: Principles and Paradigms (разделы 2.1, 4.3)
- The Many Faces of Publish/Subscribe
- RabbitMQ vs Kafka Series