Непрямое взаимодействие

Олег Сухорослов

Распределенные системы

Факультет компьютерных наук НИУ ВШЭ

04.10.2021

Непрямое (косвенное) взаимодействие

- Indirect communication
- Происходит через некоторого посредника или абстракцию, без прямого связывания между взаимодействующими процессами

Indirection

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

> ...except for the problem of too many layers of indirection.
>
> Kevlin Henney

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

> It is easier to move a problem around (for example, by moving the problem to a different part of the overall network architecture) than it is to solve it.
> Corollary: It is always possible to add another level of indirection.
>
> RFC 1925 The Twelve Networking Truths

Связывание процессов

- Связывание по пространству (space coupling)
  - Процессы должны обладать информацией друг о друге
  - Например, отправитель должен знать адрес получателя
- Связывание по времени (time coupling)
  - Процессы должны выполняться в одно время
- Синхронизация между процессами
  - Отправитель блокируется (например, до приема сообщения)

Хранение сообщений

- Transient communication
  - сообщения не хранятся на промежуточных узлах
  - если получатель недоступен, то сообщение отбрасывается
- Persistent communication
  - сообщение хранится до момента его доставки получателю

Доставка сообщений

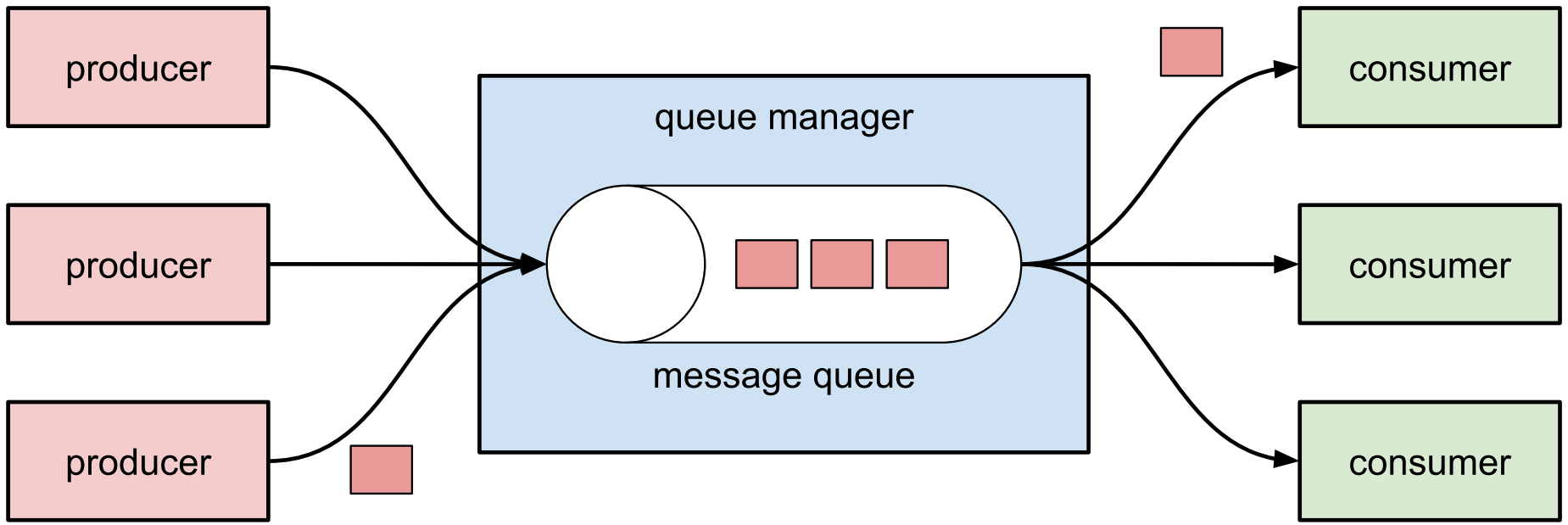
- Push
  - отправитель инициирует доставку
  - получатель пассивен
- Pull
  - получатель инициирует доставку
  - отправитель пассивен

Модели непрямого взаимодействия

- Групповые взаимодействия (group communication)
  - см. предыдущую лекцию
- Очередь сообщений (message queue)
- Издатель-подписчик (publish-subscribe)
- Общая память (shared memory)

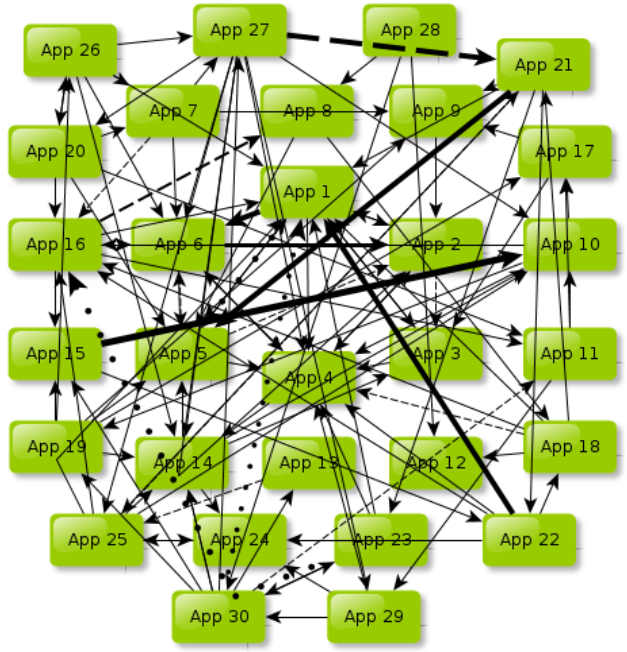
Очередь сообщений

Очередь сообщений



Применение

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



Свойства

- Обеспечивает space и time uncoupling
- Поддержка асинхронного взаимодействия, push и pull
- Хранение сообщения до момента его получения (persistent)
- Получатель сообщения только один (one-to-one)

Модель программирования

- *put(msg, q)*
  - поместить сообщение в указанную очередь
- *get(q) -> msg*
  - извлечь сообщение из указанной очереди (блокируется, если очередь пуста)
- *poll(q) -> msg*
  - извлечь сообщение из указанной очереди (не блокируется)
- *notify(msg)*
  - уведомить о появлении сообщения в очереди

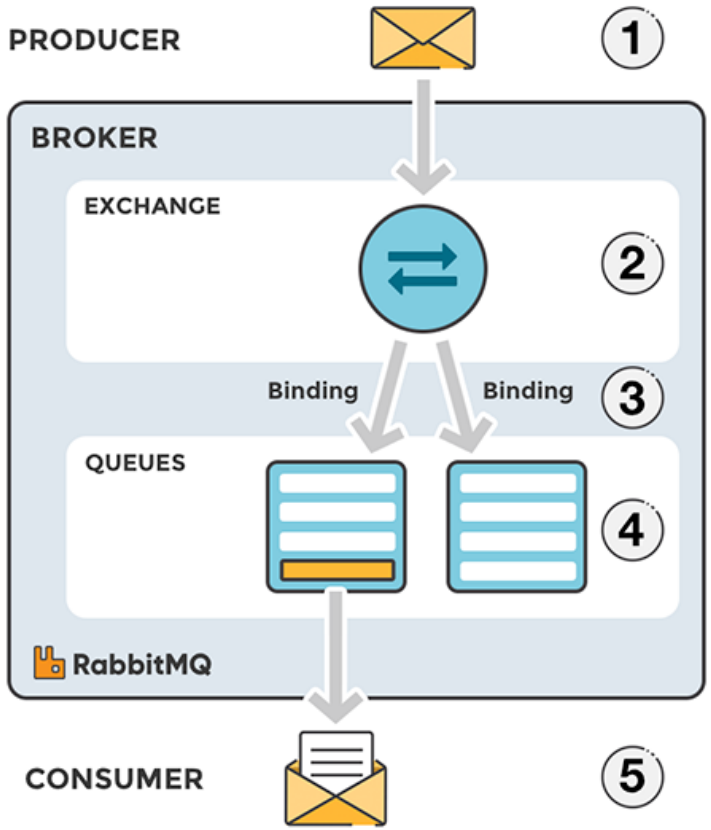
Гарантии

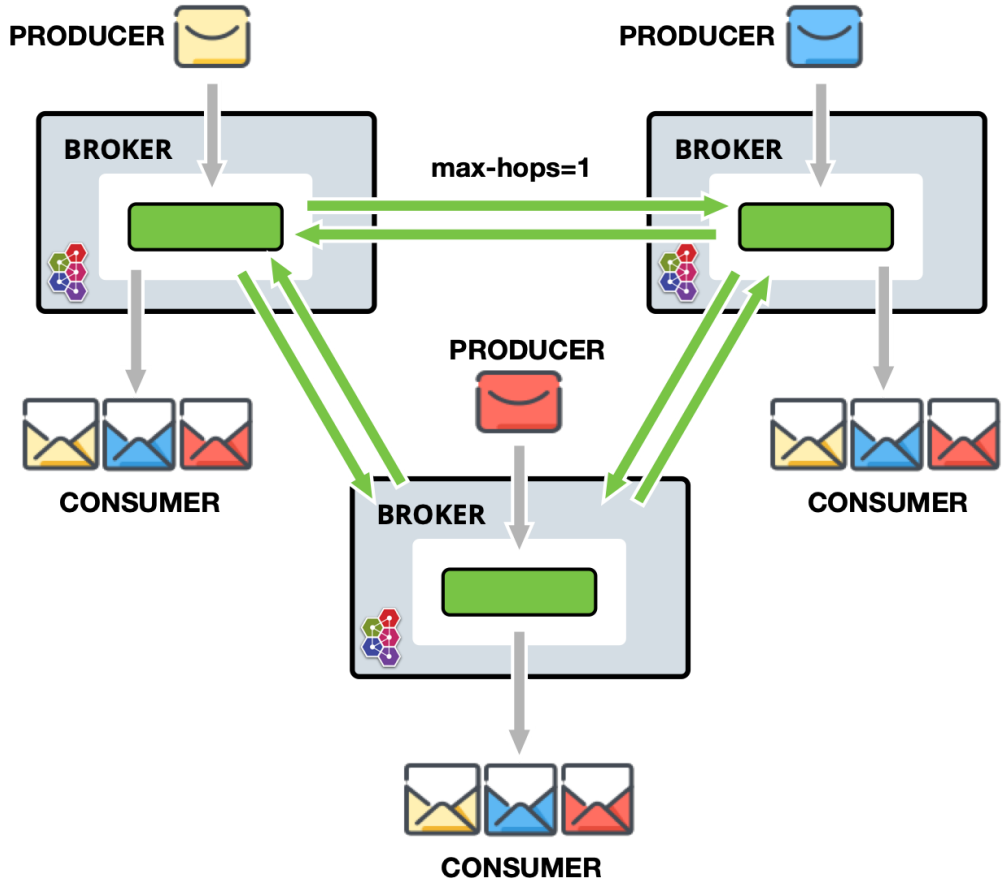
- Доставка только одному получателю
- Гарантии по доставке и обработке сообщений
- Сохранение порядка сообщений
- Надежное хранение сообщений в процессе доставки

Дополнительные функции

- Приоритеты сообщений
- Фильтрация сообщений на основе атрибутов
- Преобразование сообщений, конвертация между форматами
- Отправка и получение сообщений в рамках транзакций
- Безопасность (аутентификация, конфиденциальность, права доступа)

Архитектура





Ключевые моменты реализации

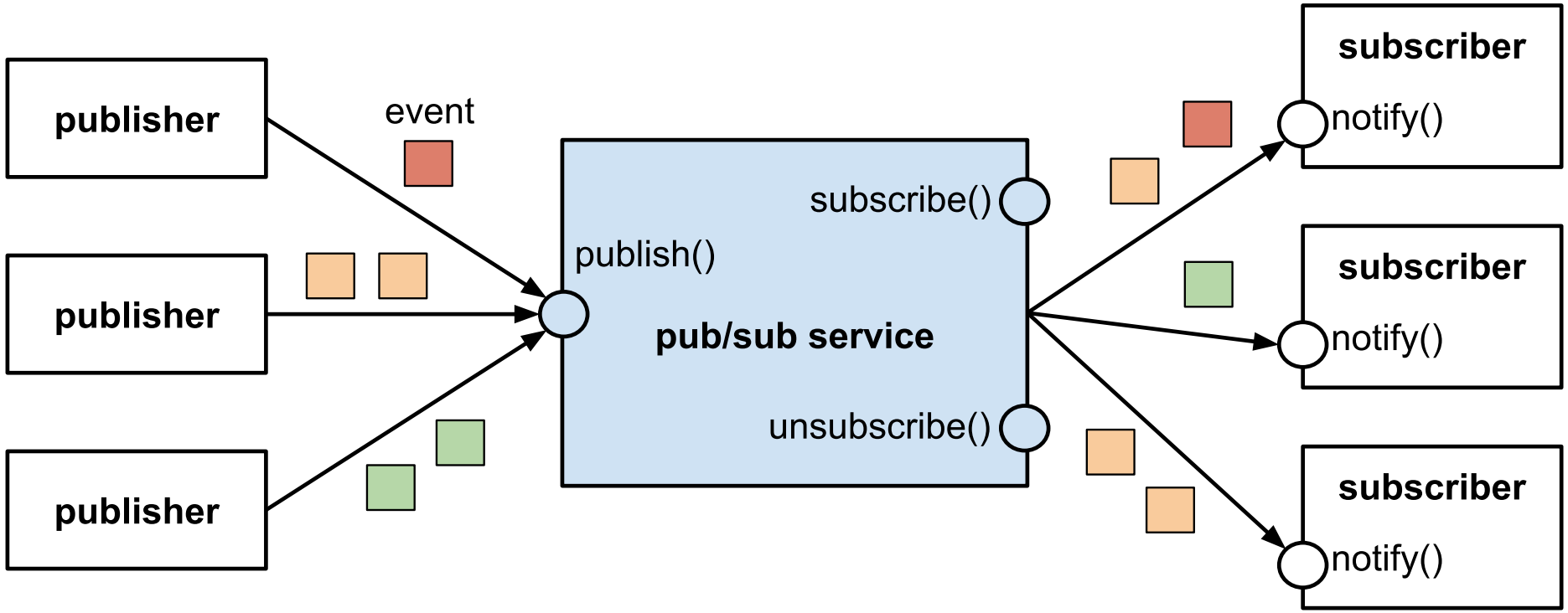
- Надежное хранение сообщений
- Реализация гарантий доставки сообщений, сохранение порядка
- Контроль потребления ресурсов (лимиты, TTL, flow control)

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

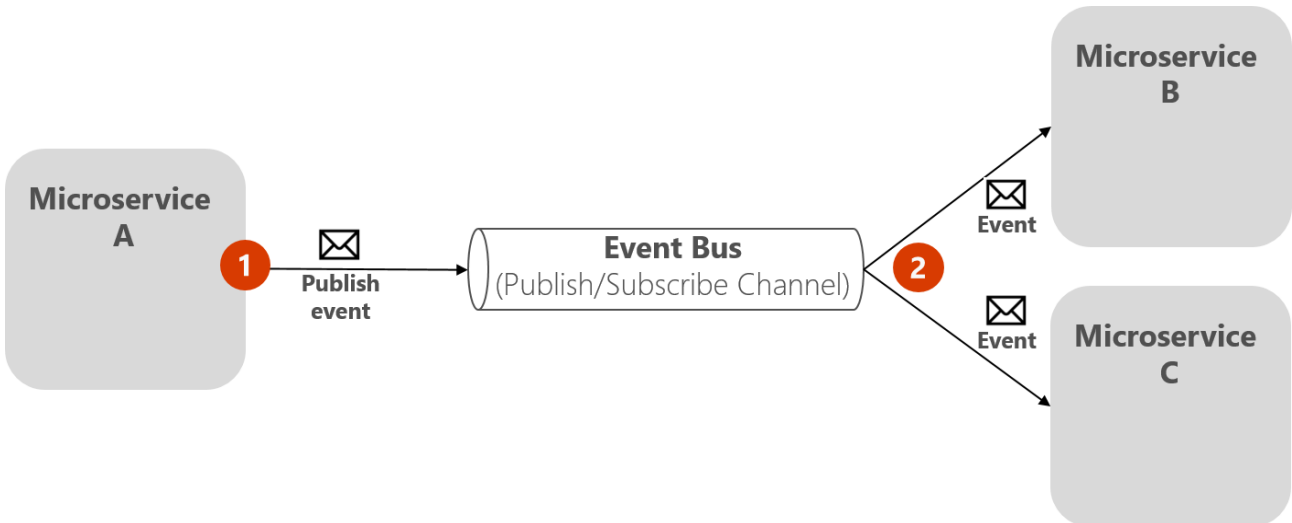
Издатель-подписчик (publish/subscribe)

Publish/Subscribe



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



Свойства

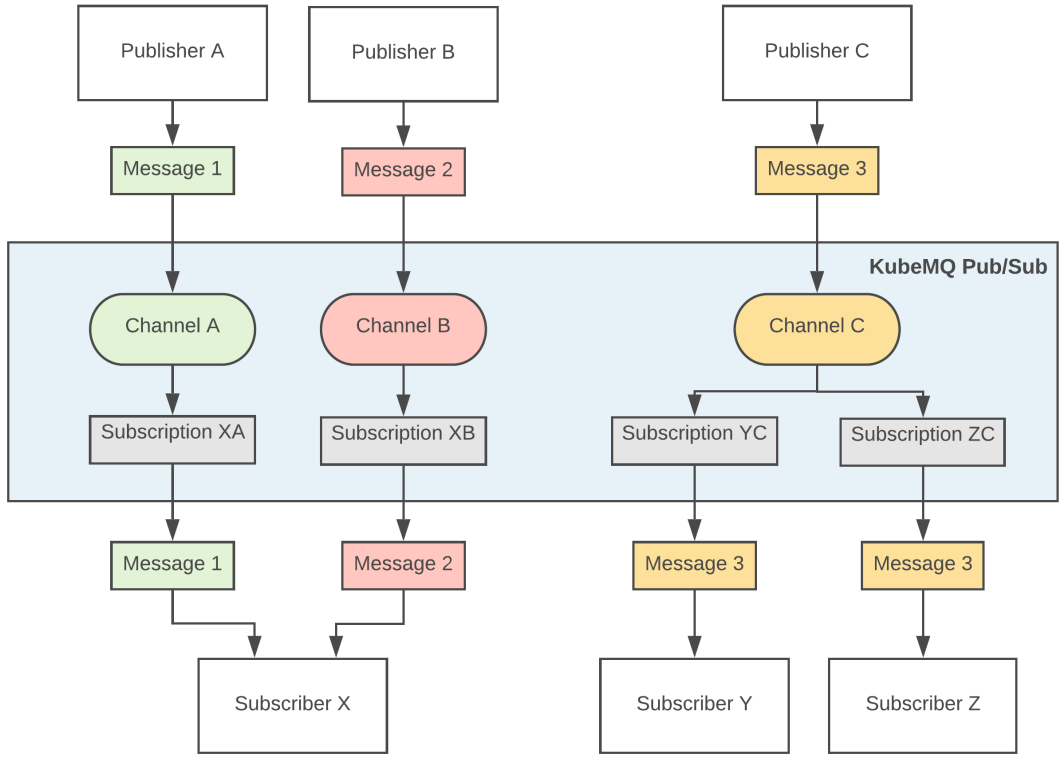
- Обеспечивает space uncoupling
  - достаточно согласовать типы и атрибуты событий
- Возможно обеспечение time uncoupling
  - хранение сообщения до его востребования
- Асинхронные взаимодействия, только push
- Получателей сообщения может быть несколько (one-to-many)

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

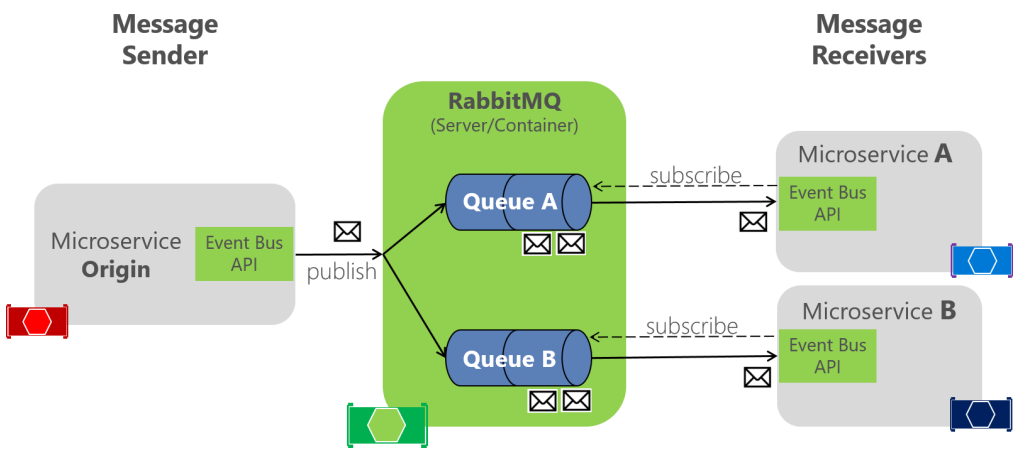
- Channel-based
- Subject-based
- Content-based
- Type-based
- Конкретный объект
- Контекст (местоположение)
- Complex event processing

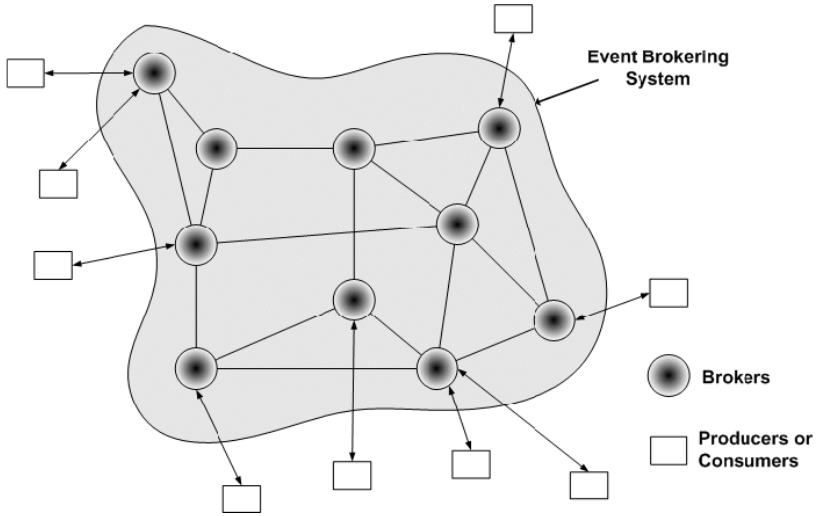


Гарантии

- Гарантии доставки и сохранения порядка событий
- Надежное хранение событий в процессе доставки

Архитектура



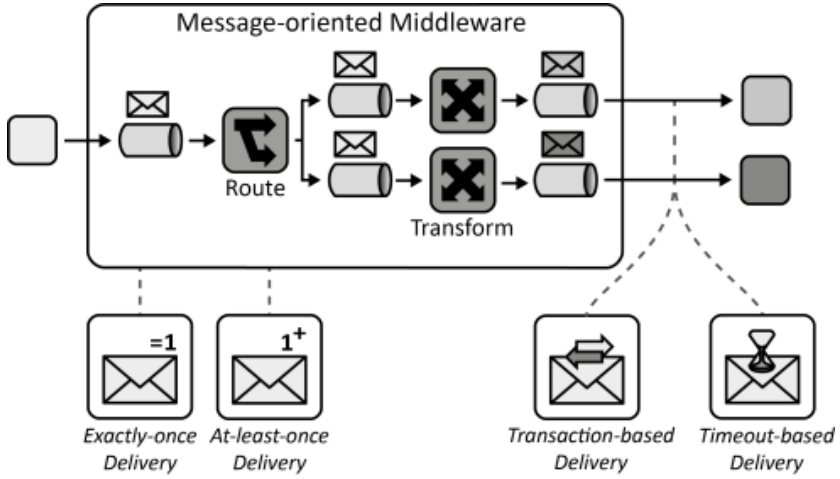


Ключевые моменты реализации

- Рассылка события нескольким подписчикам
- Реализация гарантий доставки, сохранение порядка
- Доставка пропущенных событий отключавшимся подписчикам
- Реализация надежного хранения событий
- Контроль потребления ресурсов на стороне брокера
- Контроль нагрузки на подписчика

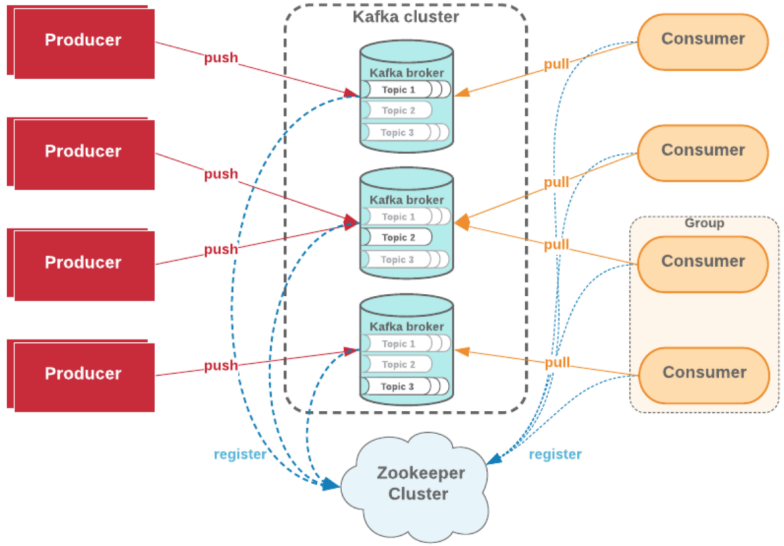
Message-oriented Middleware

- Очереди сообщений и publish-subscribe 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/



Родственные системы

- Complex event processing
  - генерация сложных событий из нескольких обычных
- Distributed messaging / stream processing
  - распределенная обработка непрерывных потоков данных
  - Kafka, MillWheel, Storm, Flink...



Общая память

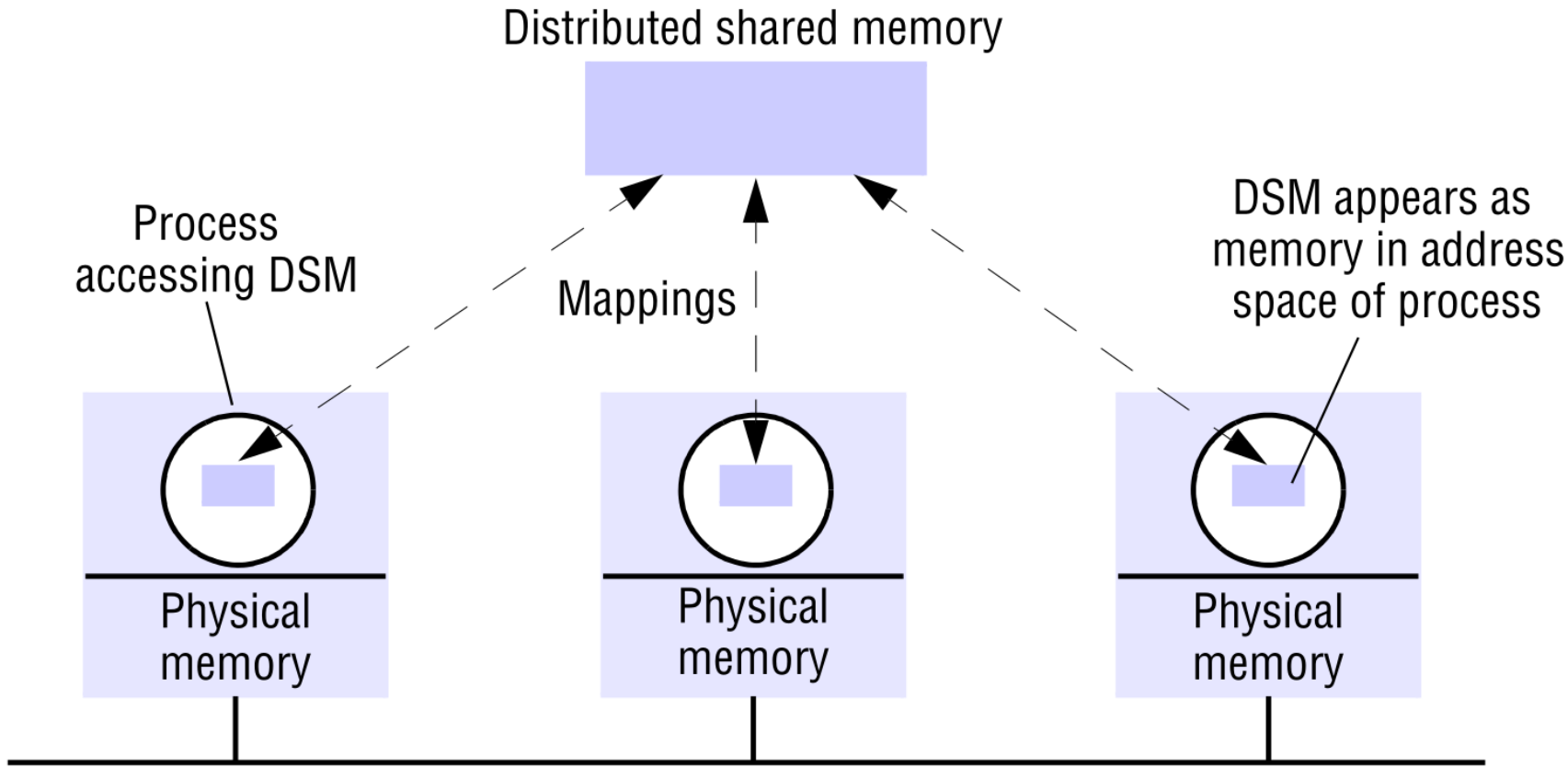
Общая память

Варианты непрямого взаимодействия на основе абстракции общей памяти

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

См. запись 2020 года

Распределенная общая память



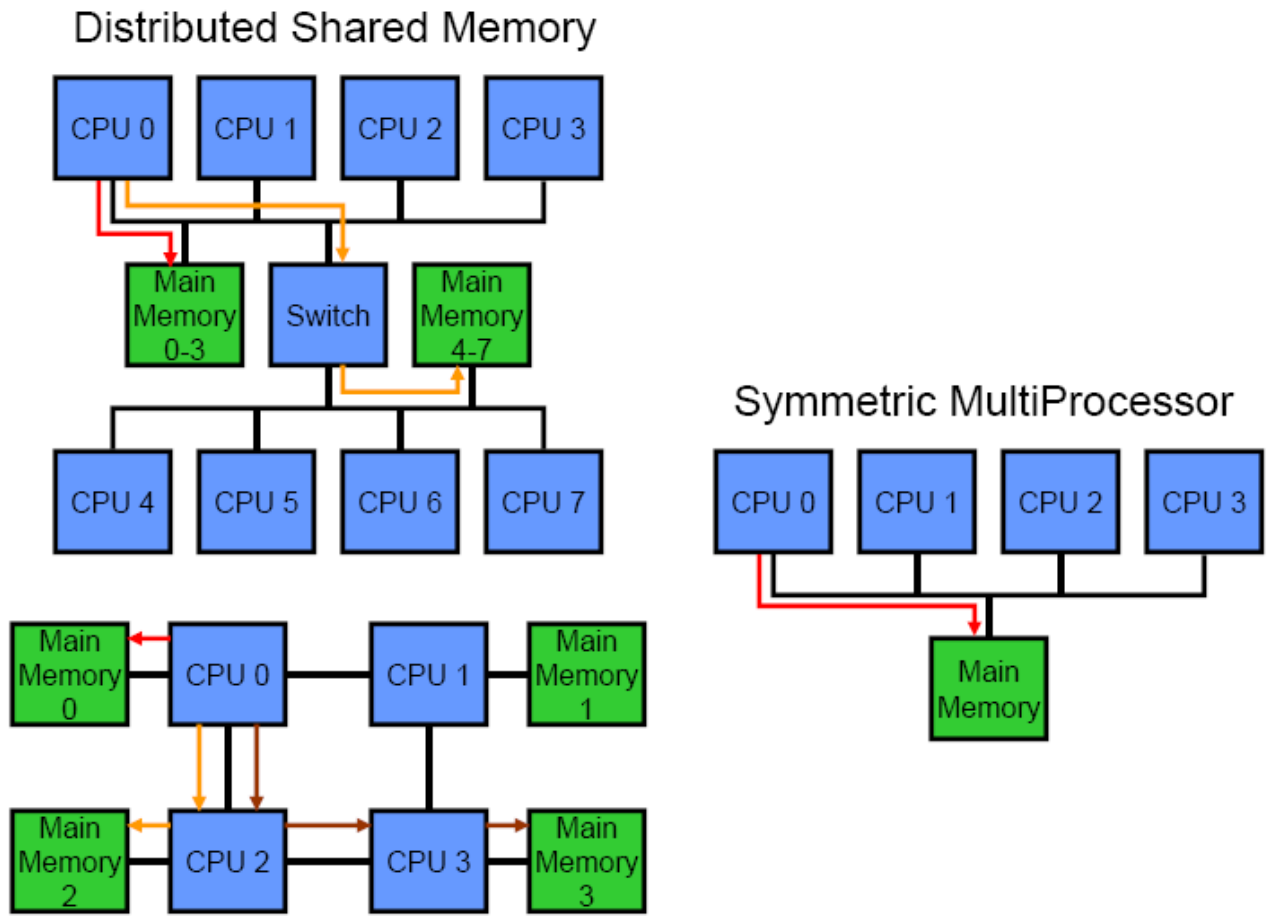
Модель программирования

- Чтение и запись данных в память по адресу
  - *write/read* вместо *send/recv*
- 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

Многопроцессорные системы



Распределенные системы

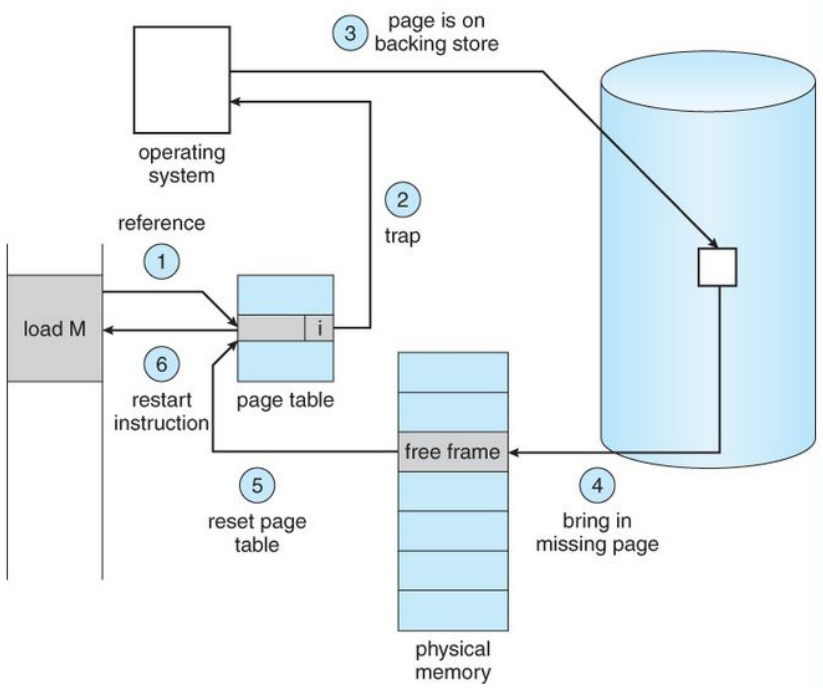
- 1980/90-е: большой интерес к software DSM
  - Попытка перенести подходы, использовавшиеся в hardware shared memory (SMP)
  - Исследовательские системы: Ivy, Mirage, Munin, Mether, Midway, TreadMarks...
  - На системах небольшого размера (~10) показана производительность на уровне реализаций message passing
  - Интерес спал после начала интернет-бума
- Дальнейшее развитие
  - 1990-е: Процессоры, архитектура Non-Uniform Memory Access
  - 1990/2000-е: Кластеры, подход single system image
  - 2000/2010-е: Параллельное программирование, модель Partitioned Global Address Space
  - 2000/2010-е: Распределенные системы хранения данных, модели согласованности

Примеры использования

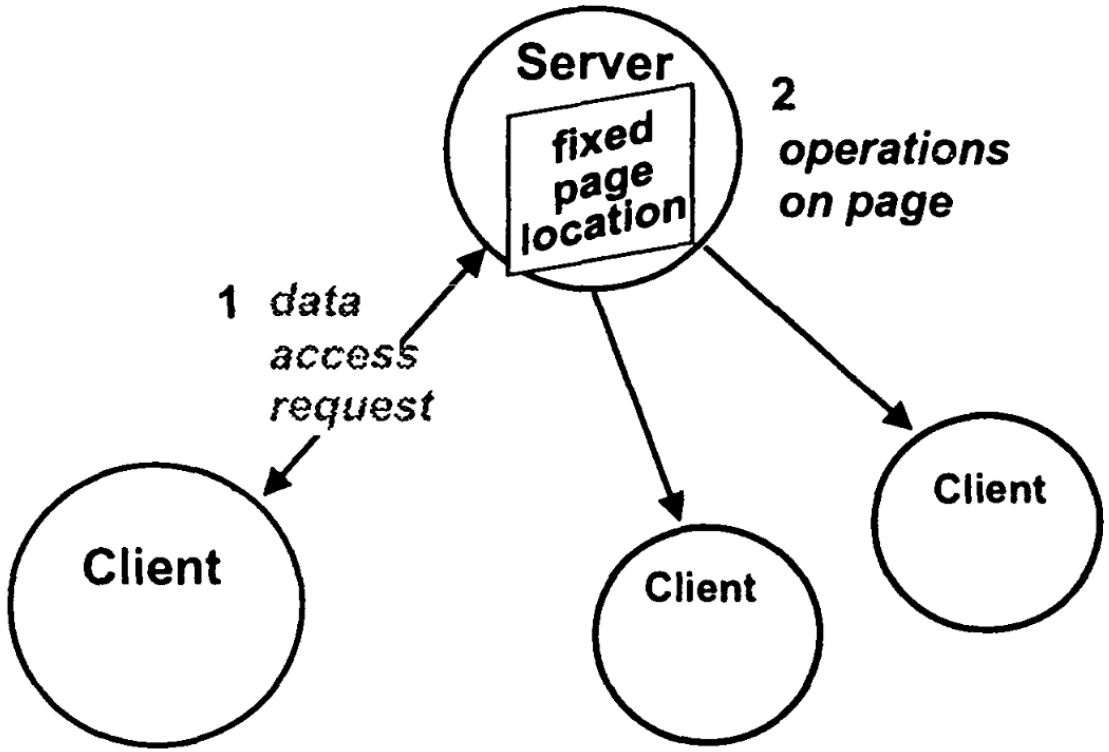
- Обеспечение прямого доступа к разделяемым данным
- Параллельные вычисления
  - Работа с общими структурами данных, тесная координация процессов
  - Перенос существующих программ с одной машины на кластер
- Клиент-серверные приложения
  - Размещение копий данных в кэше на стороне клиента
  - В общем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

Реализация

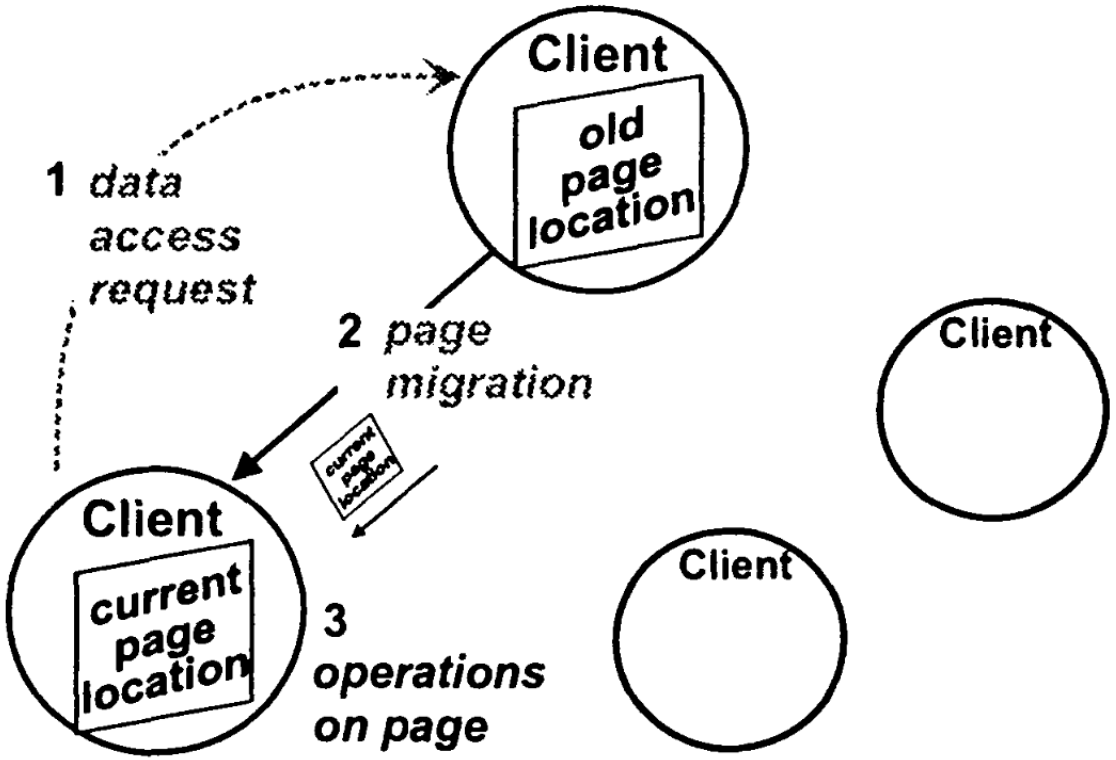
- Основные подходы
  - Использование виртуальной памяти (page based)
  - Операции над разделяемыми переменными или объектами
- Структура памяти
  - Массив байтов или слов, доступ по адресу
  - Коллекция объектов, доступ через методы
  - Неизменяемые данные (см. пространство кортежей)



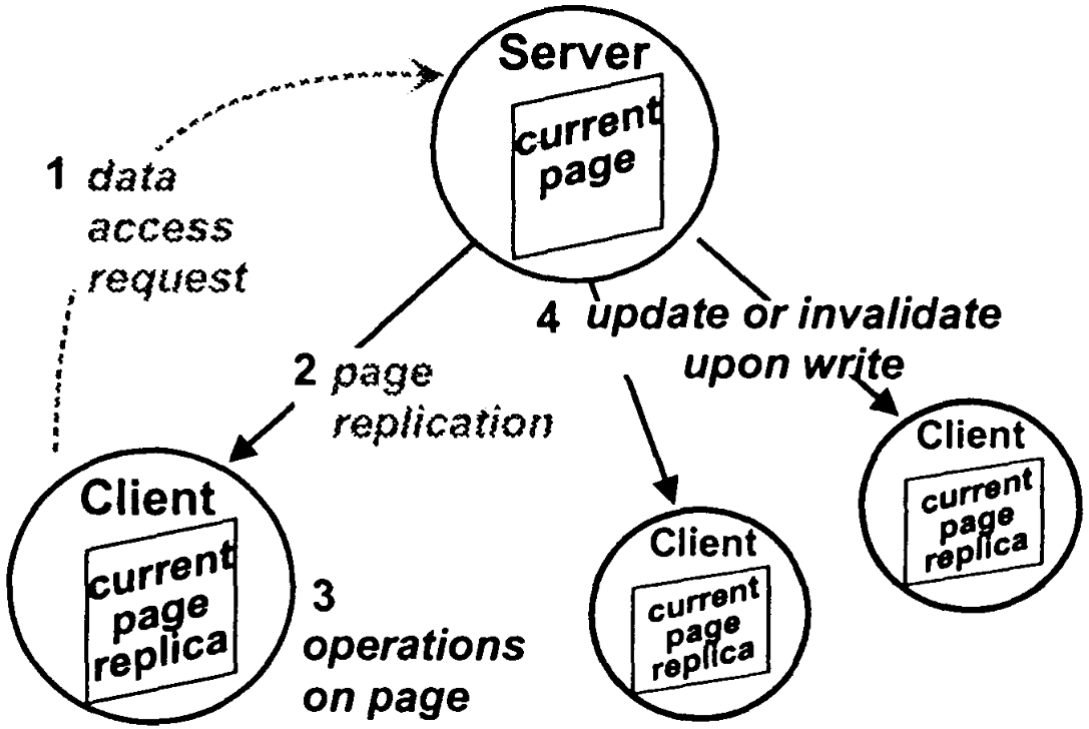
Архитектура 1 (центральный сервер)



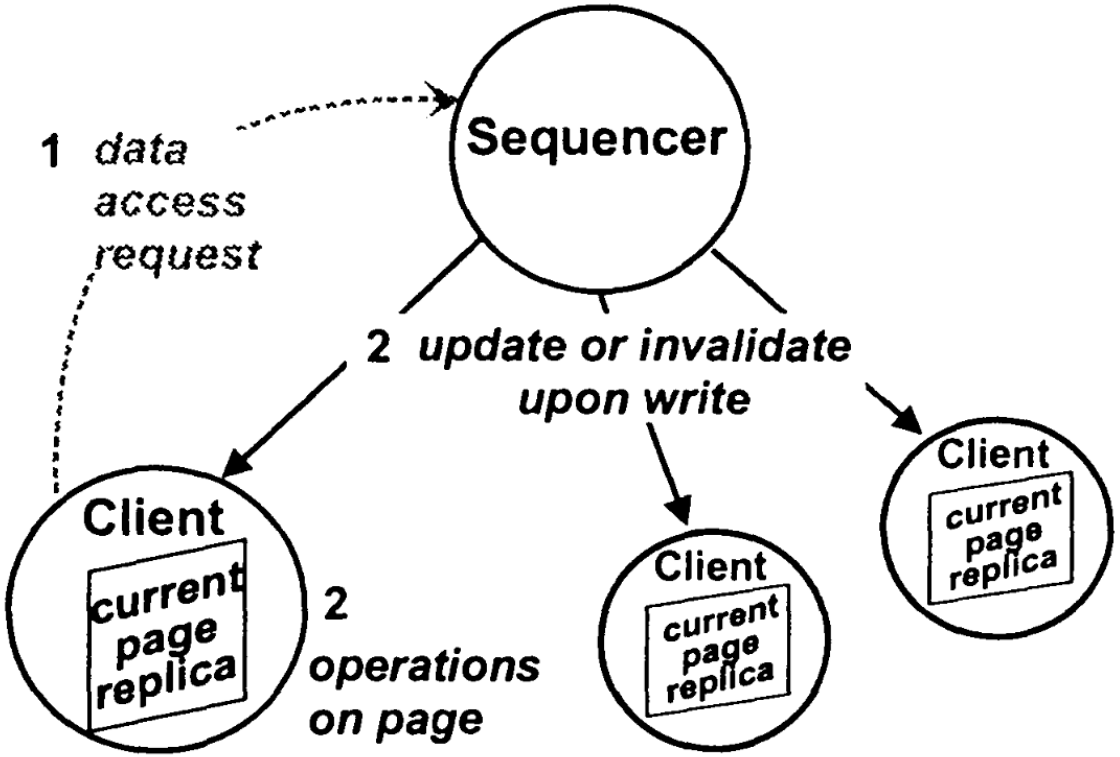
Архитектура 2 (миграция страниц)



Архитектура 3 (репликация для чтения)



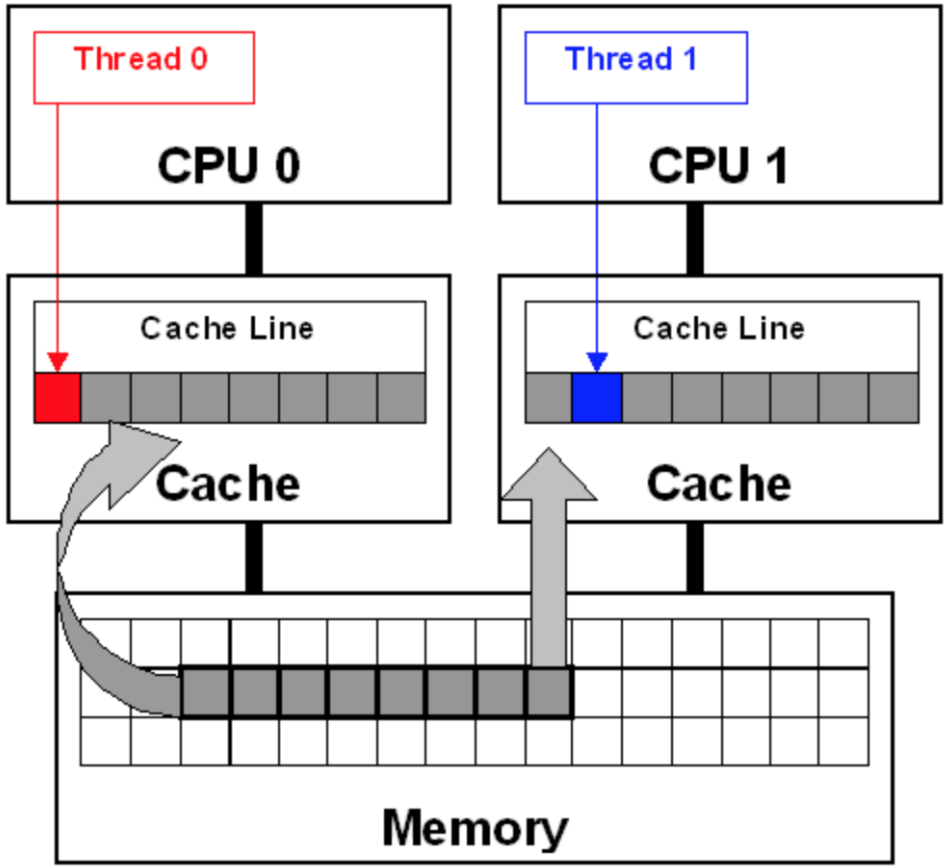
Архитектура 4 (полная репликация)



Распространение изменений

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

Ложное разделение данных (false sharing)



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
- Обеспечение согласованности данных
  - Гарантии на значения, читаемые процессами

Пример

```
a = 0
b = 0
```

Процесс 1

```
a = a + 1
b = b + 1
```

Процесс 2

```
br = b
ar = a
if (ar >= br) {
    print ("OK")
}
```

Модель согласованности

- В данном случае - memory consistency model
- Определяет гарантии, которые система предоставляет относительно читаемых процессами значений некоторых объектов
  - Объекты могут реплицироваться (храниться в копиях) на нескольких процессах
  - Объекты могут одновременно обновляться несколькими процессами
  - Определяет порядок, в котором процессы видят операции записи над объектами
  - Относится к нескольким объектам, а не к одному (coherence)
- Возможны различные модели согласованности с разной степенью строгости предоставляемых гарантий
  - Чем строже модель, тем дороже она в плане реализации, производительности, масштабируемости, доступности...
  - См. далее в курсе

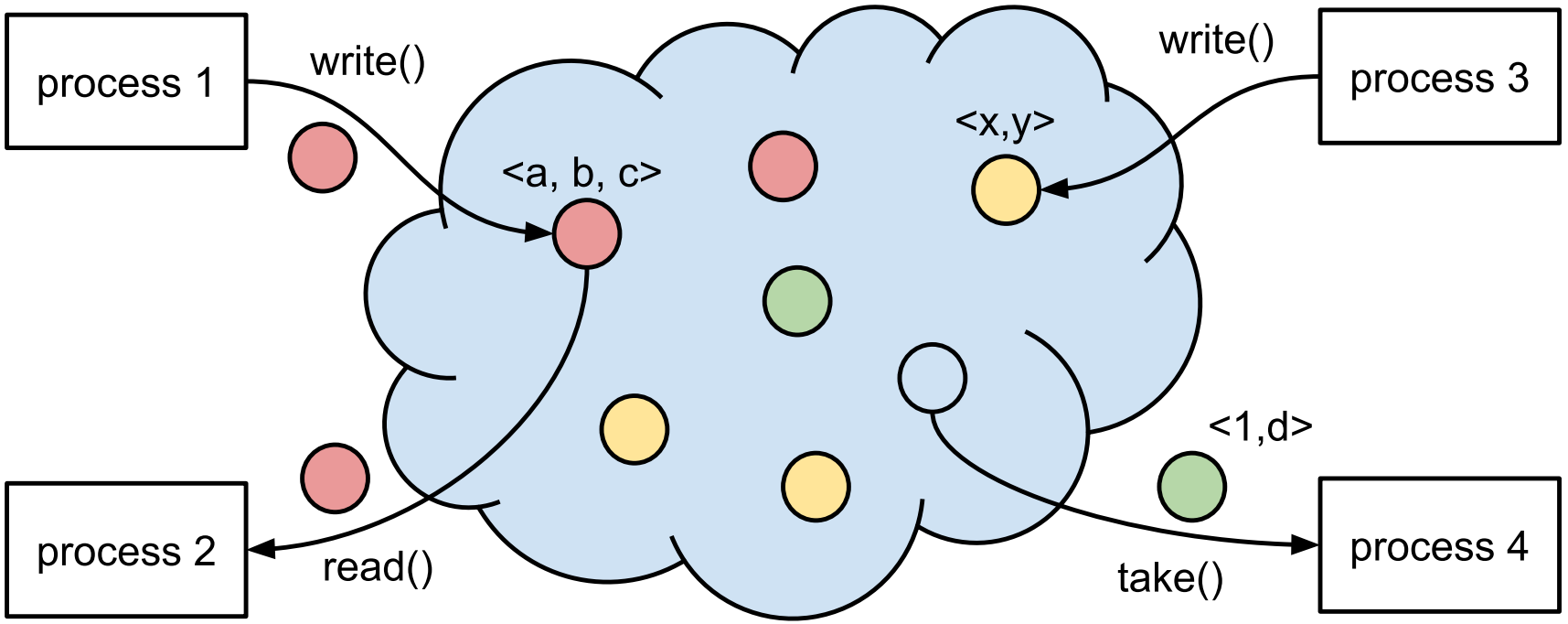
Примеры реализаций DSM

- Ivy
- Munin
- См. литературу

Распределенная общая память: проблемы

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

Пространство кортежей



Gelernter D. Generative communication in Linda // ACM TOPLAS, 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/write

Счётчик на пространстве кортежей

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

Свойства

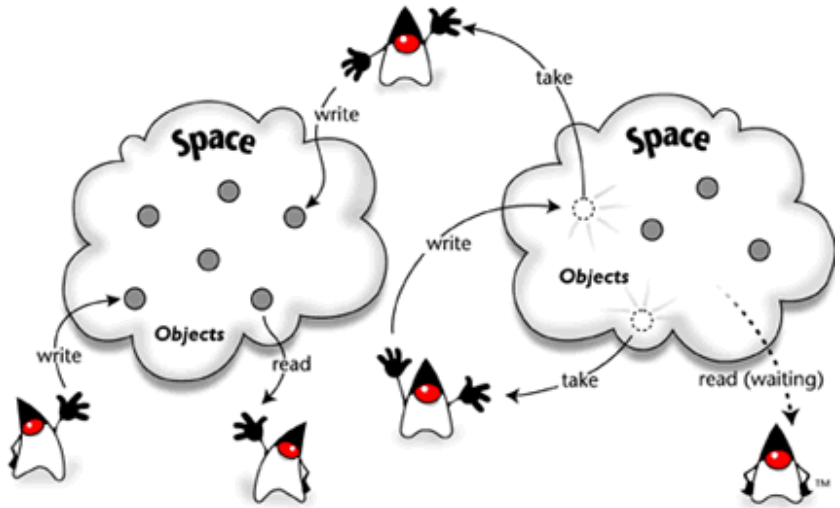
- Обеспечивает space и time uncoupling
  - достаточно согласовать типы кортежей
- Синхронные и асинхронные взаимодействия, pull и push
- Взаимодействия one-to-one и one-to-many

Расширения

- Несколько пространств вместо одного глобального
- Распределенные реализации
  - Репликация данных
  - Разбиение данных
  - 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

Сравнение описанных моделей

- Поддержка space и time uncoupling
- Подход
  - на основе взаимодействий
  - на основе общего состояния
- Схемы взаимодействия
  - one-to-one, one-to-many
- Цели и области применения
- Масштабируемость

Непрямое взаимодействие

Преимущества и недостатки?

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

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

Литература

- Coulouris G.F. et al. Distributed Systems: Concepts and Design (разд. 6.1, 6.3-6.5)
  - плюс глава Distributed Shared Memory (разд. 18.1-18.2)
- Eugster P. T. et al. The Many Faces of Publish/Subscribe
- RabbitMQ vs Kafka Part 4 - Message Delivery Semantics and Guarantees

Литература (дополнительно)

- van Steen M., Tanenbaum A.S. Distributed Systems: Principles and Paradigms
  - раздел 2.1 (Publish-subscribe architectures)
  - раздел 4.3 (Message-oriented persistent communication)
- RabbitMQ vs Kafka Series
- Глава Distributed Shared Memory (разд. 18.3-18.5)
- Hennessy J. L., Patterson D. A. Computer architecture: a quantitative approach (разделы 5.2, 5.4, 5.6)