

4. Групповые взаимодействия

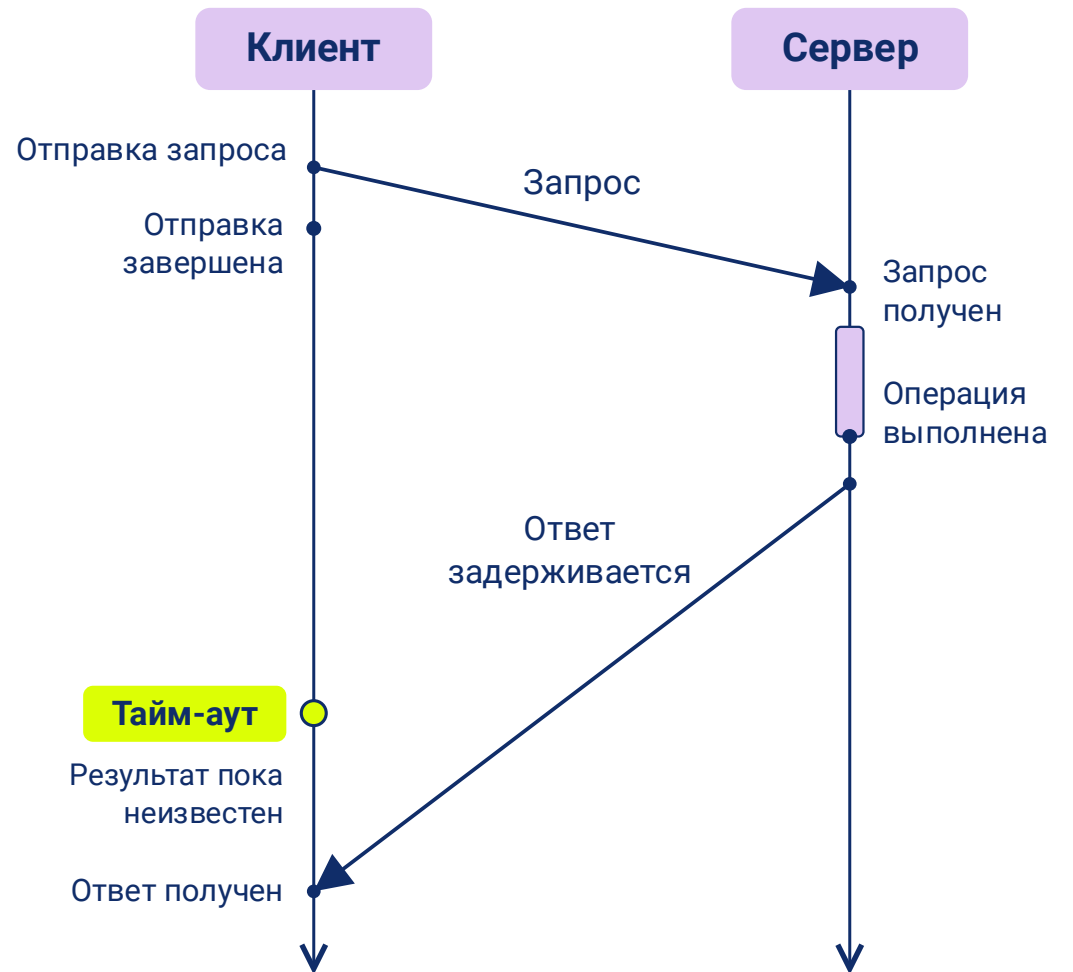
Надёжность, порядок доставки и масштабирование

Сухорослов Олег Викторович

28.09.2026

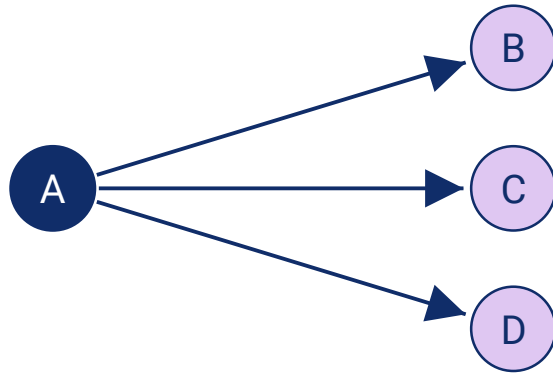
Взаимодействие двух процессов

- Подтверждения, повторы и устранение дубликатов обеспечивают гарантии доставки.
- RPC и HTTP связывают передачу сообщений с выполнением операции.
- Завершение отправки, получение сообщения и выполнение операции – разные события.
Отсутствие ответа не означает, что операция не выполнена.



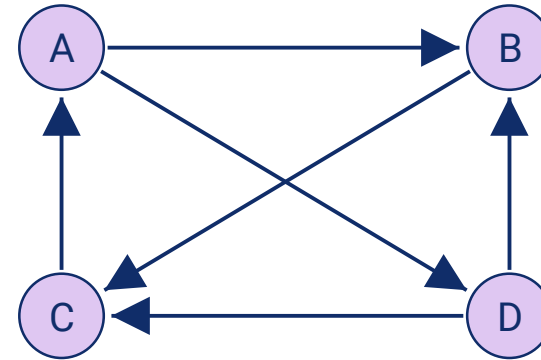
Групповые взаимодействия

Один со многими



Один отправитель, несколько получателей

Многие со многими



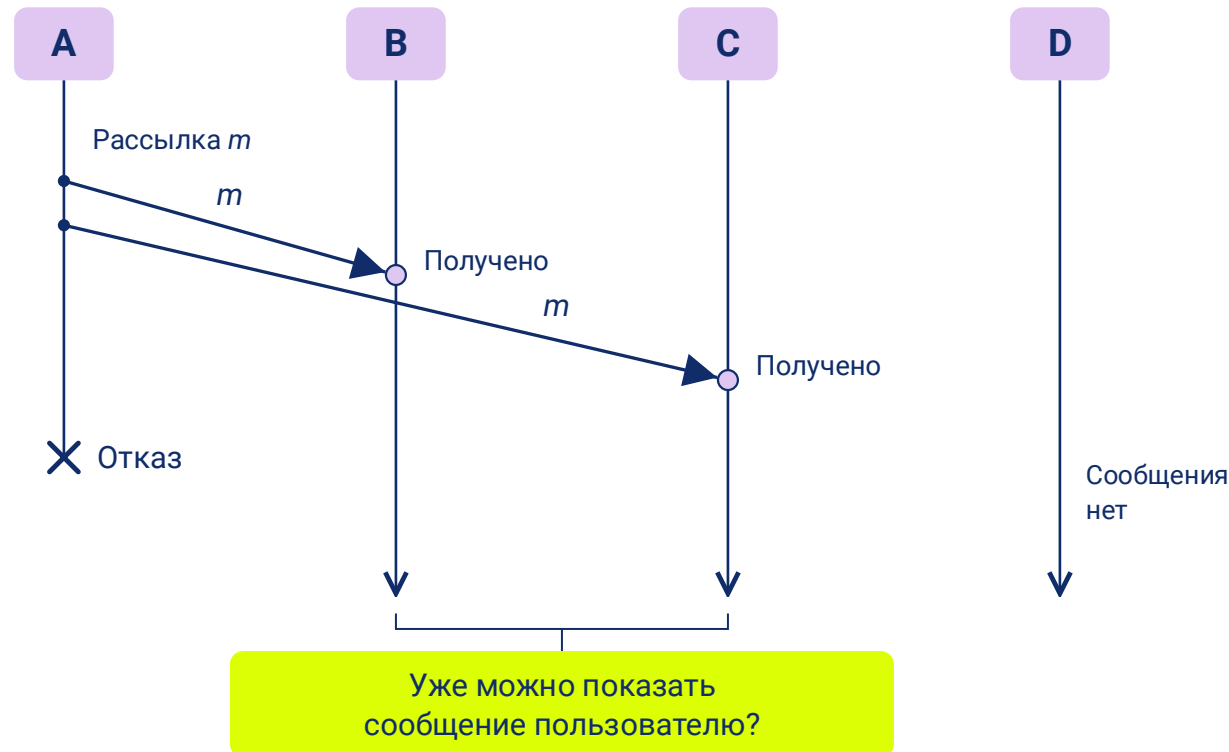
Несколько отправителей и получателей

Стрелки показывают направления обмена сообщениями

- **Уведомления:** передать событие нескольким подписчикам.
- **Контент и вещание:** доставить один поток многим получателям.
- **Обнаружение сервисов:** найти участников, предоставляющих нужный сервис.
- **Репликация:** распространить изменения между копиями данных.
- **Параллельные вычисления:** передать данные рабочим процессам или между ними.

Рассылка сообщения группе

- Каждый участник чата может отправить сообщение всем остальным.
- Отправитель может отказать, успев передать сообщение лишь части участников.
- Надёжность отдельных каналов не гарантирует надёжную рассылку всей группе.

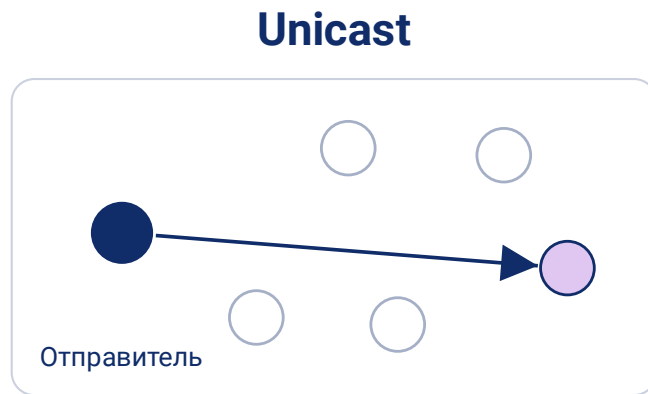


План лекции

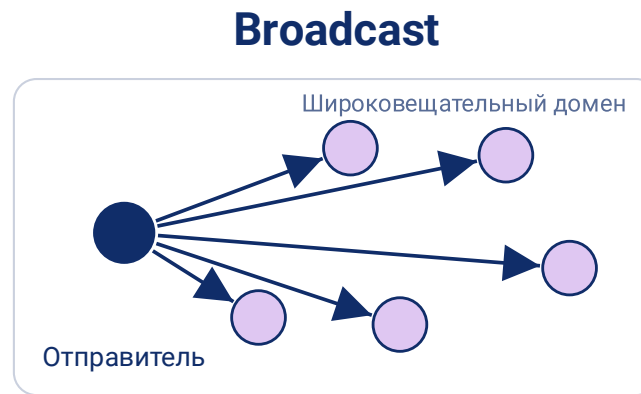
- **Групповые взаимодействия:** возможности сети и рассылка на уровне приложения.
- **Надёжная рассылка:** гарантии доставки при отказах процессов.
- **Порядок сообщений:** FIFO, причинный и общий порядок.
- **Масштабирование рассылки:** дерево, flooding и gossip.

Unicast, broadcast и multicast

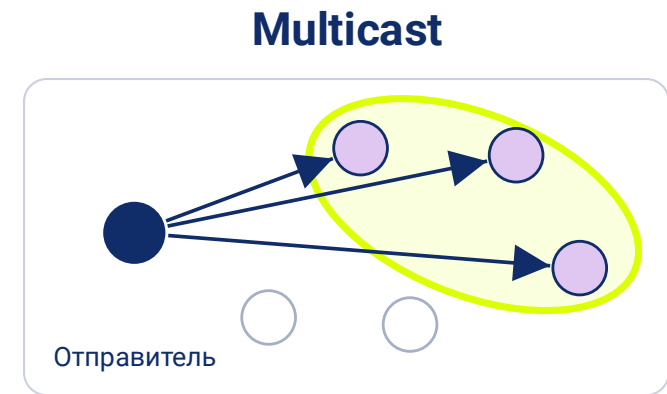
- **Unicast** – передача одному получателю.
- **Broadcast** – передача всем узлам в пределах широковещательного домена.
- **Multicast** – передача участникам выбранной группы.



Один получатель



Все узлы домена



Участники группы

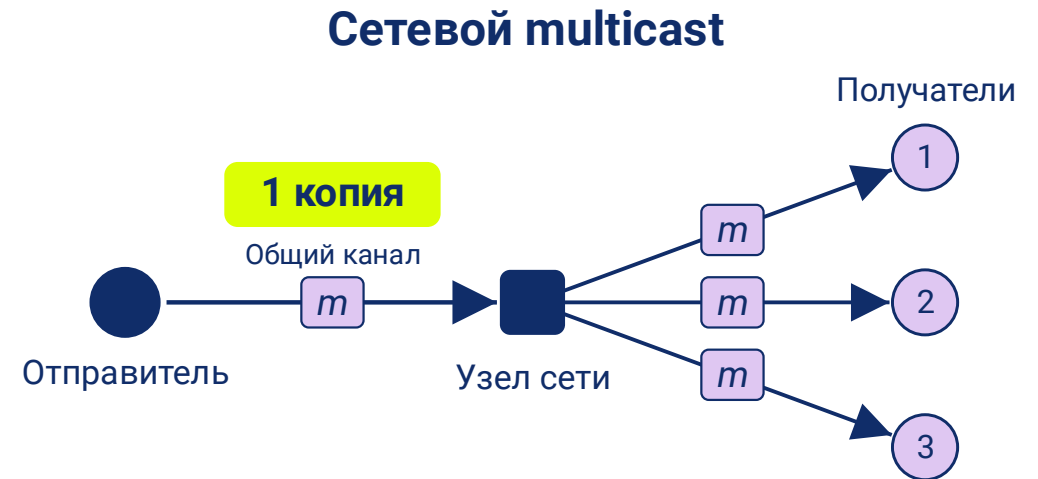
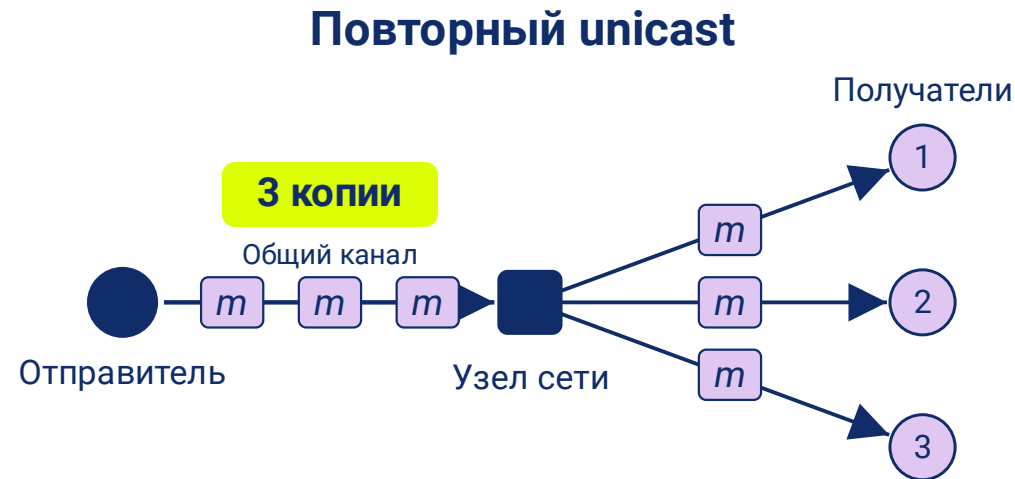
Групповую рассылку можно реализовать отдельными unicast-отправками или использовать поддержку multicast в сети.

Unicast и multicast: стоимость рассылки

Повторный unicast: отправитель передаёт отдельную копию каждому получателю.

Сетевой multicast: отправитель передаёт одну копию; сеть размножает её на разветвлениях маршрута.

- Выигрыш – меньше исходящего трафика отправителя и повторных передач по общим участкам сети.



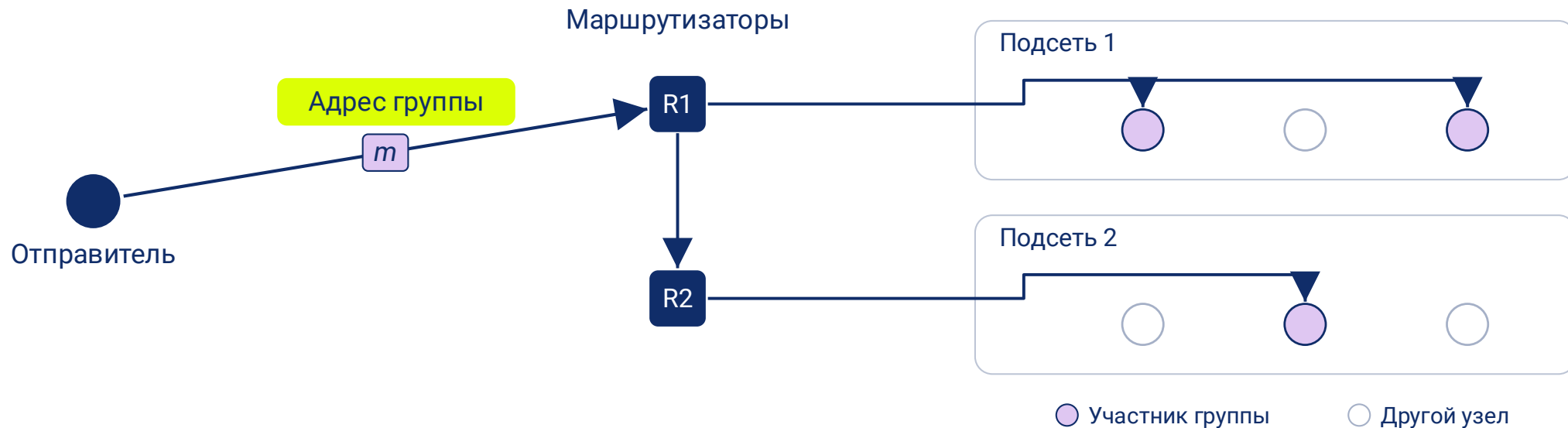
Multicast сокращает сетевой трафик, но сам по себе не гарантирует доставку сообщений.

Поддержка multicast в сети

Ethernet: групповые MAC-адреса для рассылки в локальной сети.

IP multicast: групповой IP-адрес; получатели присоединяются к нужной группе.

- Для передачи между подсетями требуется поддержка **multicast-маршрутизации**.
- Приложения обычно используют IP multicast через **UDP** – доставка и порядок пакетов не гарантируются.

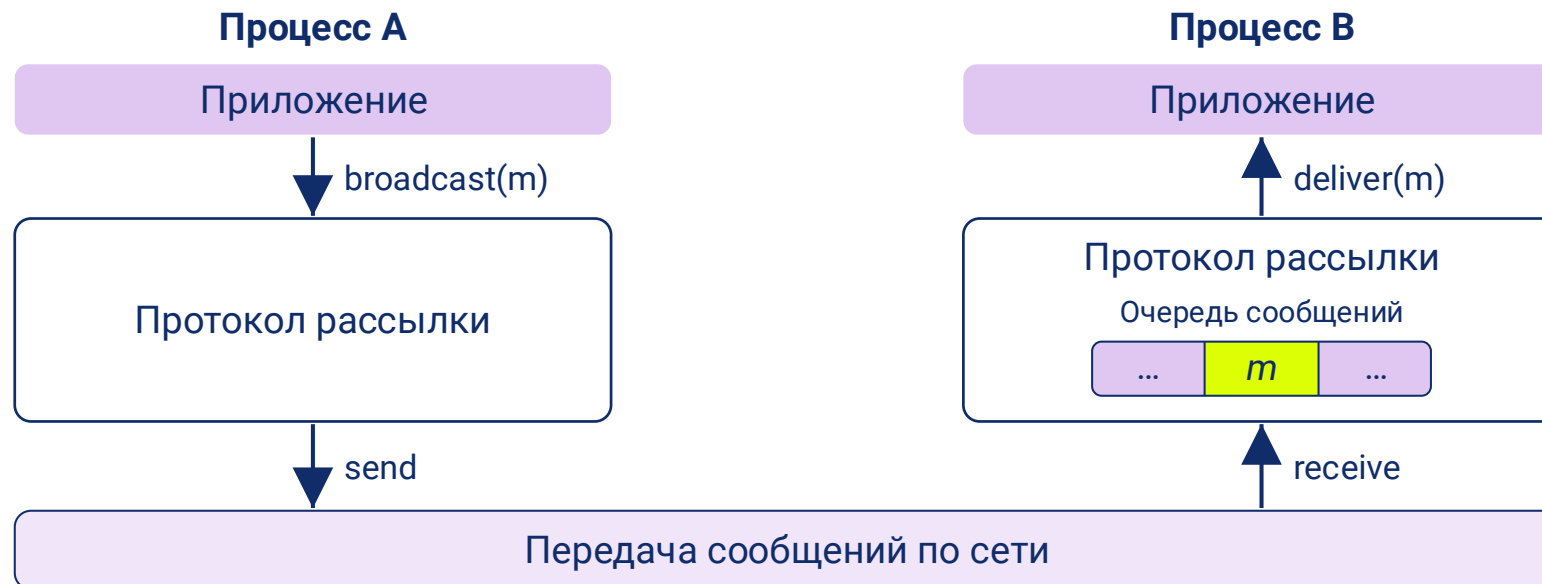


Рассылка на уровне приложения

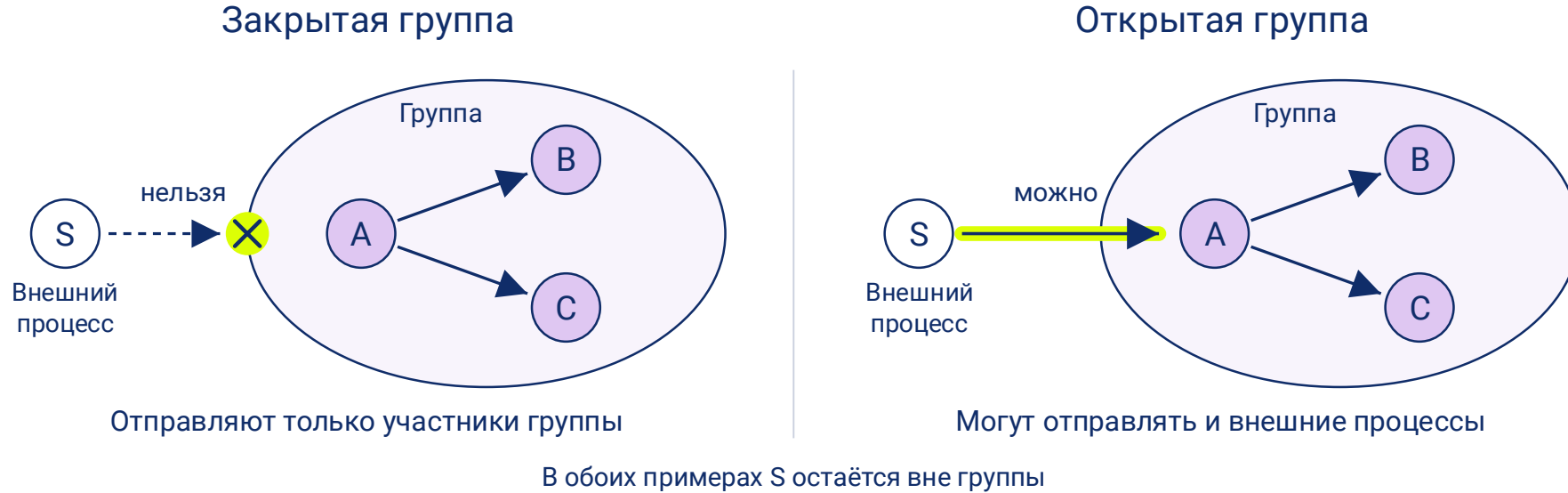
Multicast может быть недоступен между участниками. Групповую рассылку строят поверх отдельных unicast-соединений.

Возможностей сети может быть недостаточно. Приложению нужны доставка при отказах, определённый порядок сообщений и правила работы с составом группы.

Протокол рассылки использует доступный транспорт и обеспечивает требуемые свойства.



Состав и открытость группы



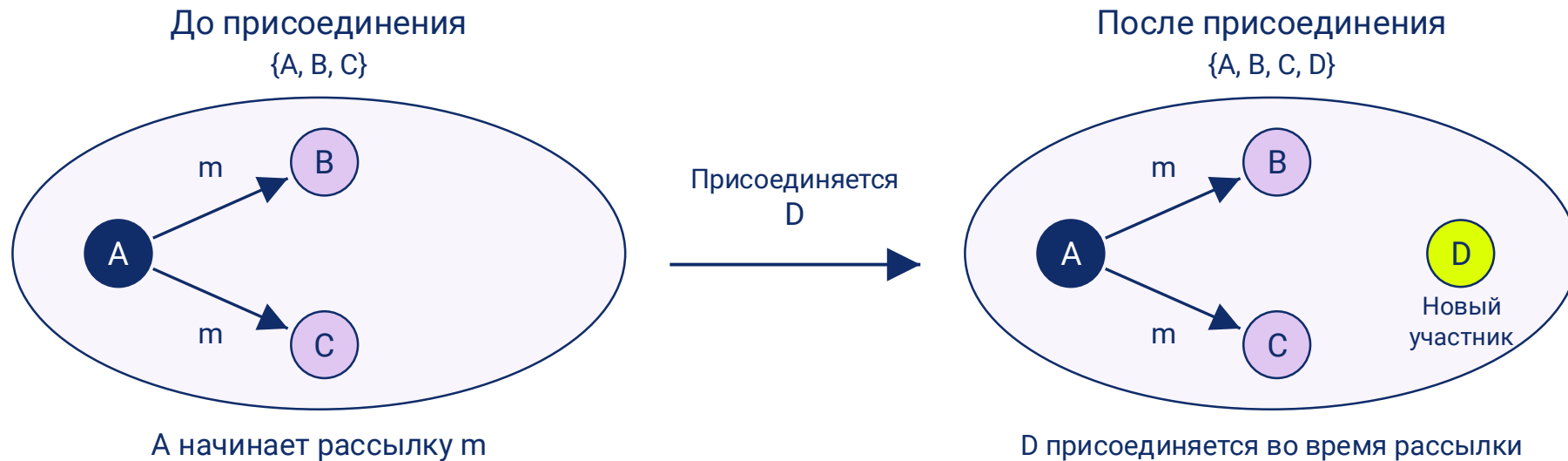
	Варианты
Кто может отправлять сообщения группе?	Закрытая: только участники группы. Открытая: также процессы вне группы.
Может ли меняться состав группы?	Статический: состав фиксирован. Динамический: процессы могут присоединяться и покидать группу.

Открытость и изменение состава – независимые характеристики.

Управление составом группы

- **Изменение состава:** присоединение, выход и исключение участников.
- **Согласование состава:** какие процессы входят в текущую группу.
- **Переход к новому составу:** какие сообщения и состояние должны получить участники.

Если процесс присоединяется во время рассылки, должен ли он получить это сообщение?



Должен ли D получить сообщение m?

Модель системы

- **Фиксированная группа:** каждый процесс знает всех участников.
- **Надёжные каналы:** между корректными процессами сообщения доставляются без потерь, искажений и повторов.
- **Произвольные задержки:** известной верхней границы времени доставки нет; порядок сообщений не гарантируется.
- **Отказы остановкой:** процесс может остановиться навсегда; его сообщения в пути могут быть потеряны.

Корректный процесс – процесс, который не отказывает в рассматриваемом исполнении.

Надёжный канал между процессами

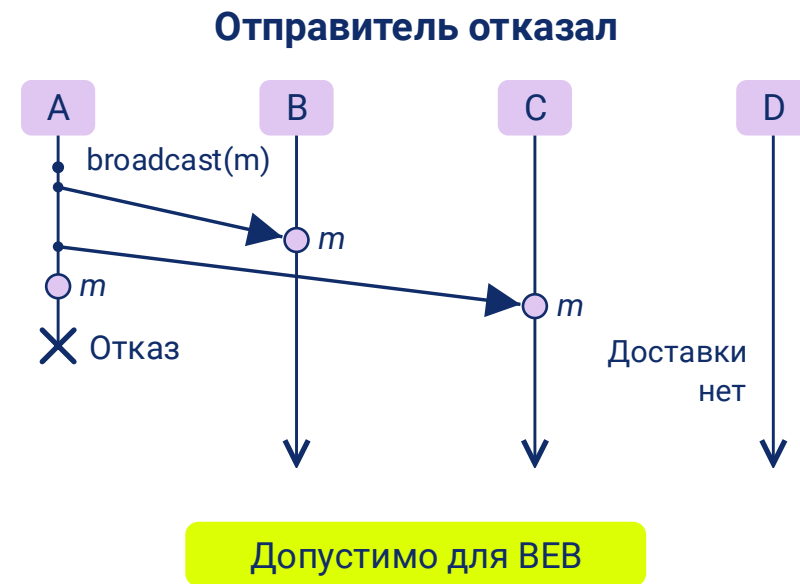
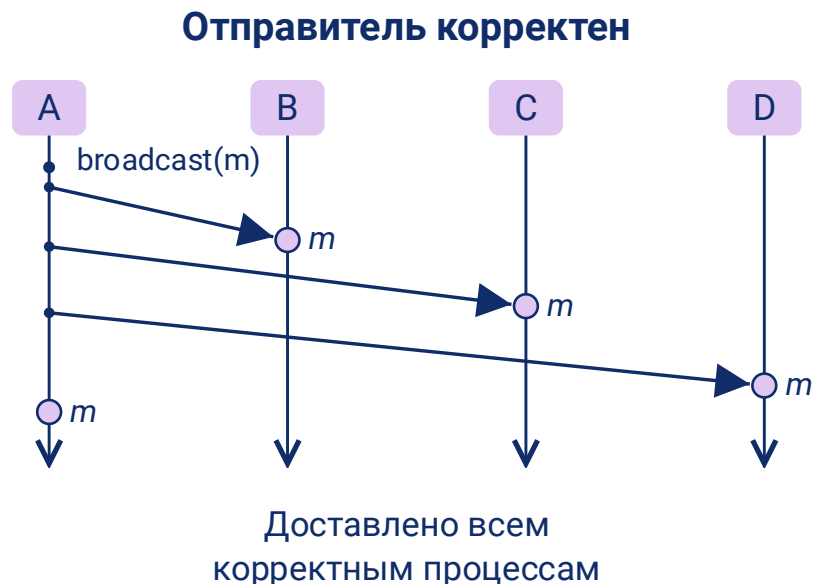
- **Validity:** если корректный процесс p отправил сообщение m корректному процессу q , то q в конце концов доставит m .
- **No Duplication:** каждое сообщение доставляется получателю не более одного раза.
- **No Creation:** если q доставил сообщение m от p , то p ранее отправил именно это сообщение процессу q .

Порядок доставки не гарантируется.

Сообщения отказавшего отправителя могут быть потеряны.

Best-Effort Broadcast (BEB)

- **Validity:** если корректный процесс разослал сообщение, каждый корректный процесс в конце концов его доставит.
- **No Duplication:** каждый корректный процесс доставляет каждое сообщение не более одного раза.
- **No Creation:** если корректный процесс доставил сообщение от некоторого отправителя, тот ранее разослал это сообщение.



○ deliver(m): доставка сообщения приложению

Реализация Best-Effort Broadcast

```
При broadcast(m):  
    для каждого q из группы:  
        send(q, m)
```

```
При receive(m):  
    deliver(m)
```

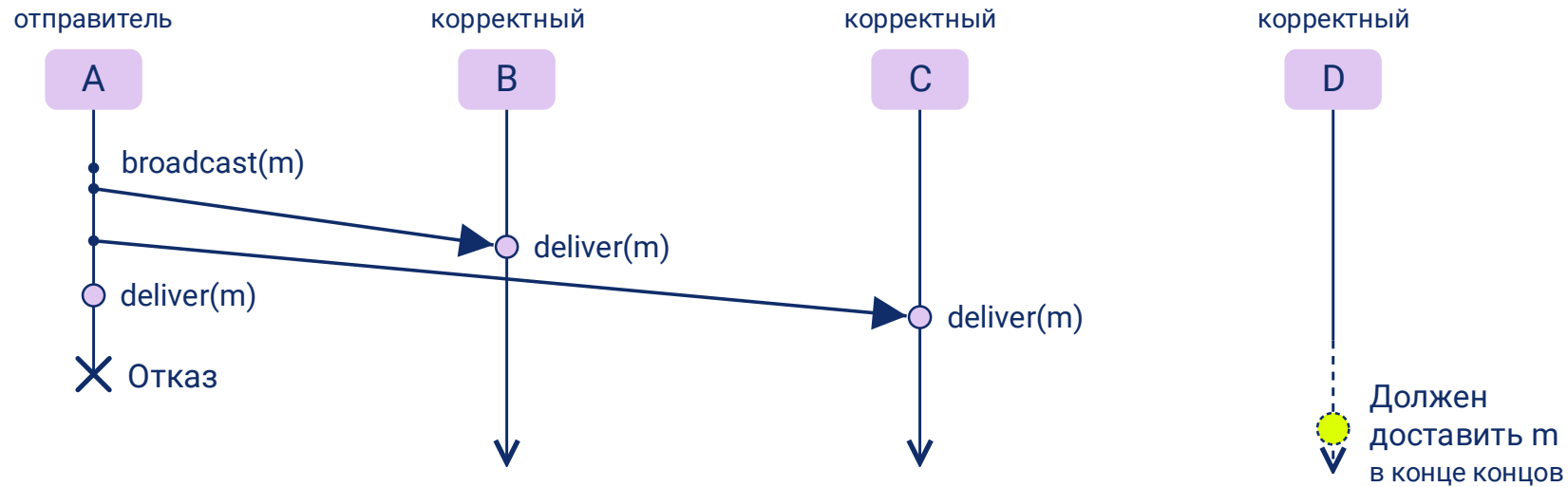
- Отправляем сообщение всем участникам, **включая себя**.
- Полученное сообщение сразу доставляем приложению.
- Используем надёжные каналы между процессами.

Reliable Broadcast (RB)

Сохраняются свойства BEB: **Validity, No Duplication, No Creation.**

Добавляется **Agreement (согласие о доставке)**:

Если некоторый **корректный процесс** доставил сообщение, то все корректные процессы в конце концов доставят это сообщение.



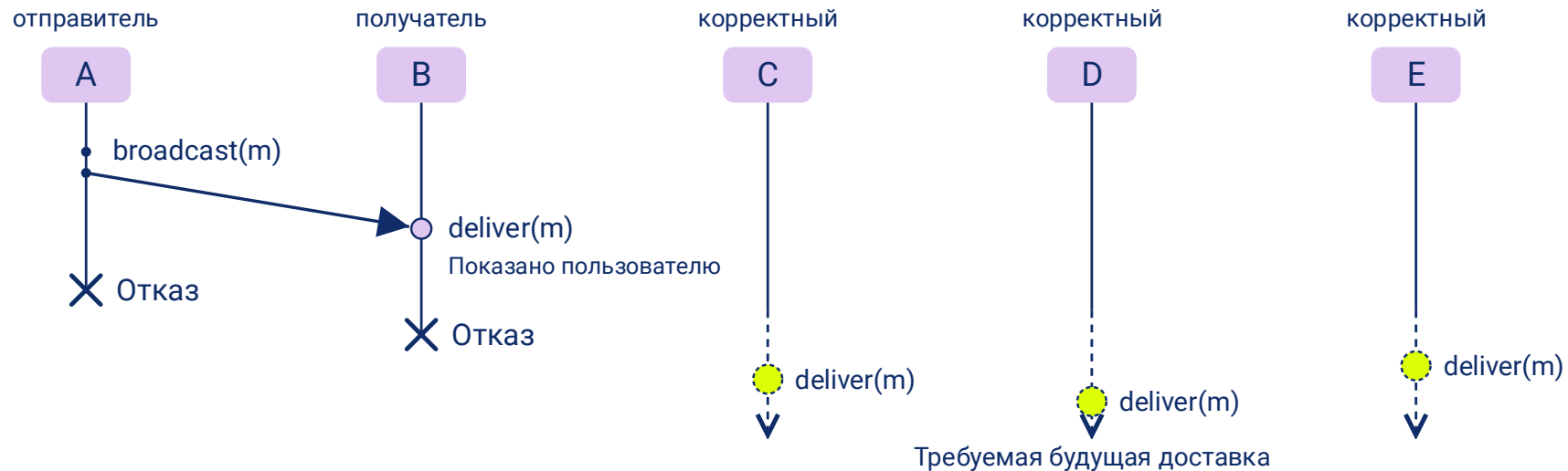
Если B или C доставил m , то D тоже должен в конце концов доставить m

Uniform Reliable Broadcast (URB)

Свойство **Uniform Agreement**:

Если **любой процесс** доставил сообщение, то все корректные процессы в конце концов доставят это сообщение.

Требование сохраняется, даже если доставивший сообщение процесс затем отказал.
Свойства **Validity**, **No Duplication** и **No Creation** сохраняются.



В уже доставил m приложению.
URB требует, чтобы C, D и E тоже в конце концов доставили m.

Eager Reliable Broadcast

```
При broadcast(m):  
    BEB.broadcast(m)
```

```
При BEB.deliver(m):  
    если id(m) уже в seen:  
        завершить обработку  
    добавить id(m) в seen  
    BEB.broadcast(m)  
    deliver(m)
```

- seen – множество идентификаторов уже обработанных сообщений.
- При пересылке сохраняются идентификатор и исходный отправитель.
- Стоимость одной рассылки – **$O(N^2)$** сетевых сообщений.

BEV, RB и URB: гарантии доставки

Во всех трёх случаях выполняются **Validity, No Duplication и No Creation**.

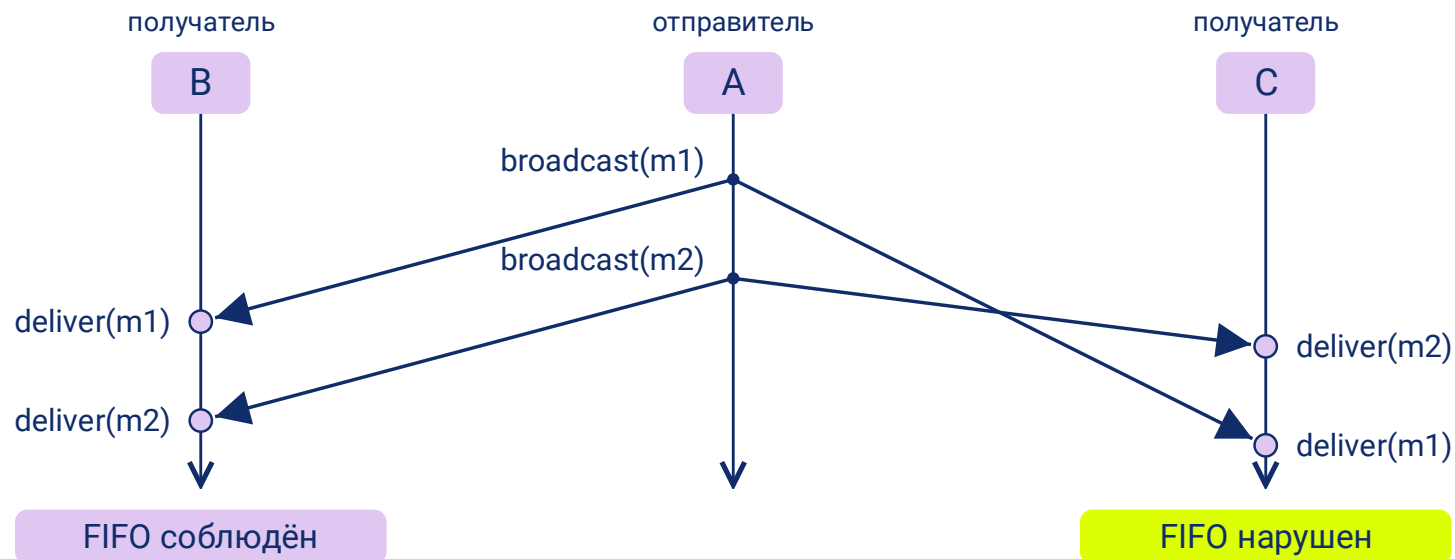
Абстракция	Обязательство по доставке
BEV	Сообщение корректного отправителя доставят все корректные процессы
RB	Если сообщение доставил корректный процесс, его доставят все корректные процессы
URB	Если сообщение доставил любой процесс, его доставят все корректные процессы

Во всех формулировках доставка требуется в конце концов.

FIFO: порядок сообщений одного отправителя

Если корректный процесс разослал m_1 раньше m_2 , то каждый корректный процесс, доставляющий m_2 , должен сначала доставить m_1 .

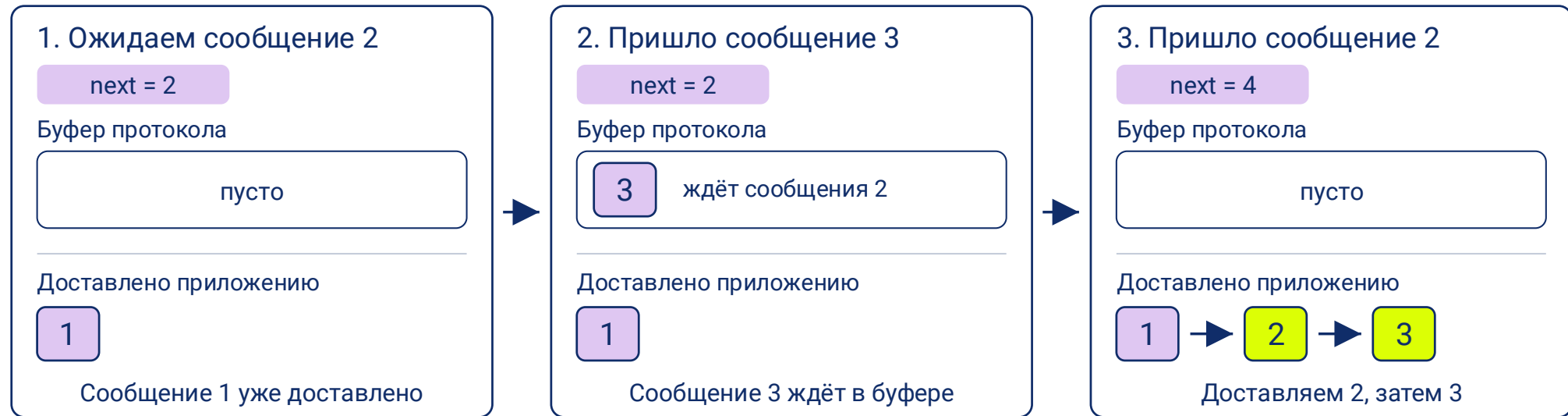
- Порядок сохраняется **отдельно для каждого отправителя**.
- Порядок сообщений **разных отправителей** может различаться.
- Сообщения могут прийти из сети не по порядку – требование относится к **доставке приложению**.



Все сообщения доставлены, но порядок у C неверен

FIFO: порядковые номера и буфер

- Отправитель нумерует свои сообщения.
- Получатель доставляет их по порядку номеров, сохраняя опередившие сообщения в буфере.
- Нумерация и ожидание ведутся отдельно для каждого отправителя.



Все сообщения — от одного отправителя; числа обозначают их порядковые номера

FIFO: алгоритм доставки

Для каждого отправителя s :

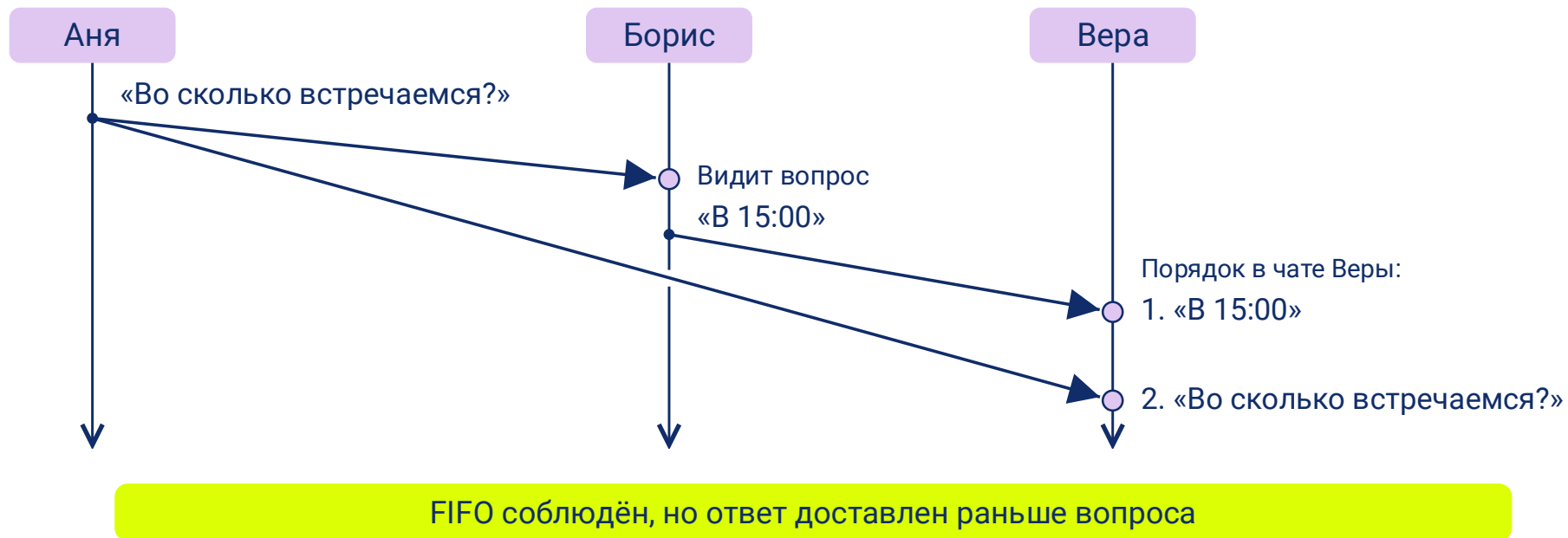
- $\text{next}[s] = 1$ – следующий ожидаемый номер
- $\text{buffer}[s] = \{\}$ – буфер сообщений

При получении (s, k, m) от надёжной рассылки:
 $\text{buffer}[s][k] = m$

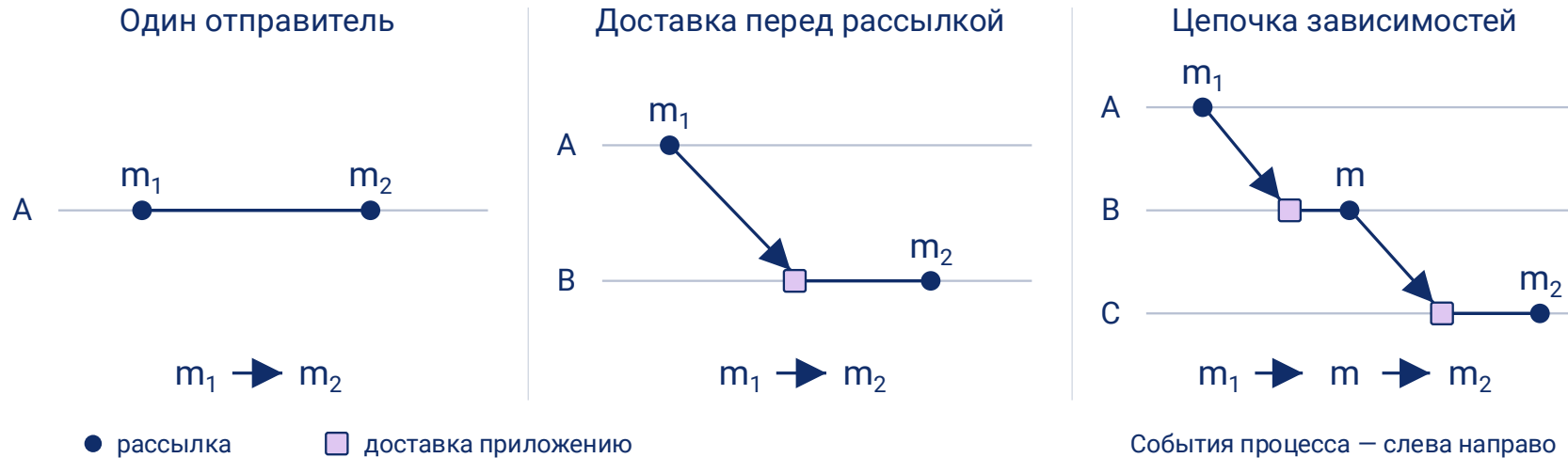
пока $\text{next}[s]$ есть в $\text{buffer}[s]$:
 $m = \text{извлечь } \text{buffer}[s][\text{next}[s]]$
 $\text{deliver}(m)$
 $\text{next}[s] += 1$

Когда FIFO недостаточно

FIFO сохраняет порядок сообщений одного отправителя, но не учитывает зависимости между разными отправителями.



Причинный порядок доставки (Causal Order)



Сообщение m_1 **причинно предшествует** m_2 ($m_1 \rightarrow m_2$), если:

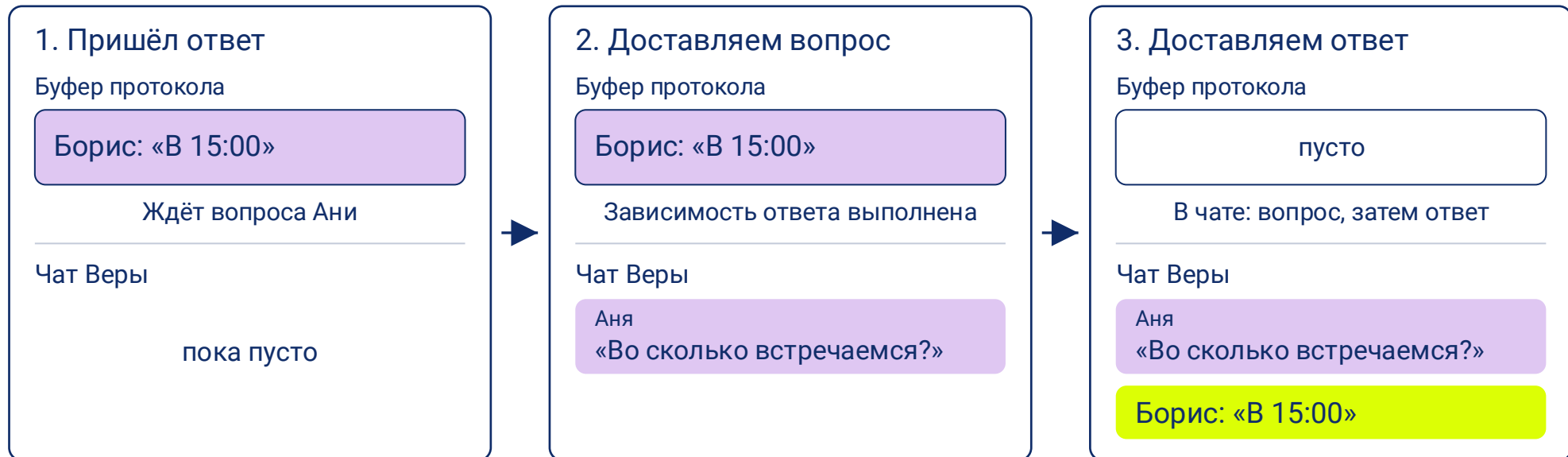
- один процесс разослал m_1 , а затем m_2 ;
- процесс доставил m_1 , а затем разослал m_2 ;
- связь следует по цепочке: $m_1 \rightarrow m \rightarrow m_2$.

Требуемая гарантия: если $m_1 \rightarrow m_2$, то каждый корректный процесс, доставляющий m_2 , должен сначала доставить m_1 . Независимые сообщения могут доставляться в разном порядке.

Причинный порядок: ожидание зависимостей

- Вместе с сообщением передаётся информация о его причинных зависимостях.
- Сообщение ждёт в буфере, пока все его предшественники не доставлены приложению.

Получить ответ можно раньше вопроса. Доставить приложению – только после него.



Причинные зависимости: вектор счётчиков

Для каждого отправителя указываем, **сколько его первых сообщений должно быть доставлено перед текущим сообщением.**

Для ответа Бориса:

Аня	Борис	Вера
1	0	0

Нужно сначала доставить первое сообщение Ани – её вопрос.

Размер вектора – по одному счётчику на участника группы.

Когда можно доставить сообщение?

- $D[s]$ – сколько сообщений отправителя s требуется доставить **перед текущим**.
- $V[s]$ – сколько сообщений отправителя s **уже доставлено**.

Сообщение можно доставить, когда выполнены все его зависимости:

$V[s] \geq D[s]$ для каждого отправителя s .

Пример для ответа Бориса, $D = (1, 0, 0)$:

Доставлено у Веры	Решение
$V = (0, 0, 0)$	Ждём вопрос Ани
$V = (1, 0, 0)$	Можно доставить ответ Бориса

Как сформировать вектор зависимостей

- V – вектор уже доставленных сообщений.
- `sent` – число собственных рассылок, изначально 0.
- `self` – идентификатор текущего процесса.

```
При broadcast(m):  
    D = копия V  
    D[self] = sent  
    sent += 1  
    RB.broadcast(self, D, m)
```

Используем RB, чтобы сообщения, доставленные корректному процессу, в итоге получили все корректные процессы.

Причинная рассылка: алгоритм доставки

Изначально $V[s] = 0$ для каждого отправителя, буфер пуст.

```
При RB.deliver(s, D, m):  
    добавить (s, D, m) в буфер  
  
    пока в буфере есть сообщение с  $D \leq V$ :  
        извлечь такое сообщение (s, D, m)  
         $V[s] += 1$   
        deliver(m)
```

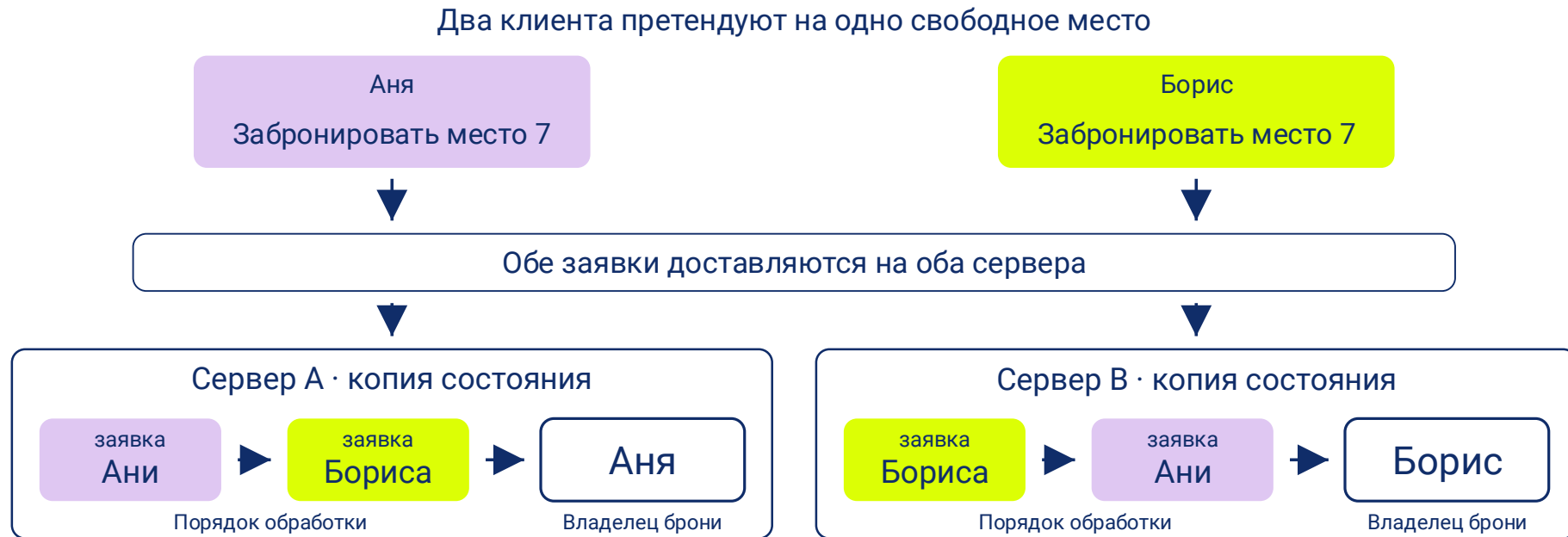
$D \leq V$ означает: **для каждой координаты $D[i] \leq V[i]$.**

Если готовы несколько сообщений, можно выбрать любое.

Зачем нужен общий порядок доставки

Серверы сервиса бронирования хранят копии состояния мест – **реплики**.

Первая обработанная заявка занимает свободное место. Следующая отклоняется.

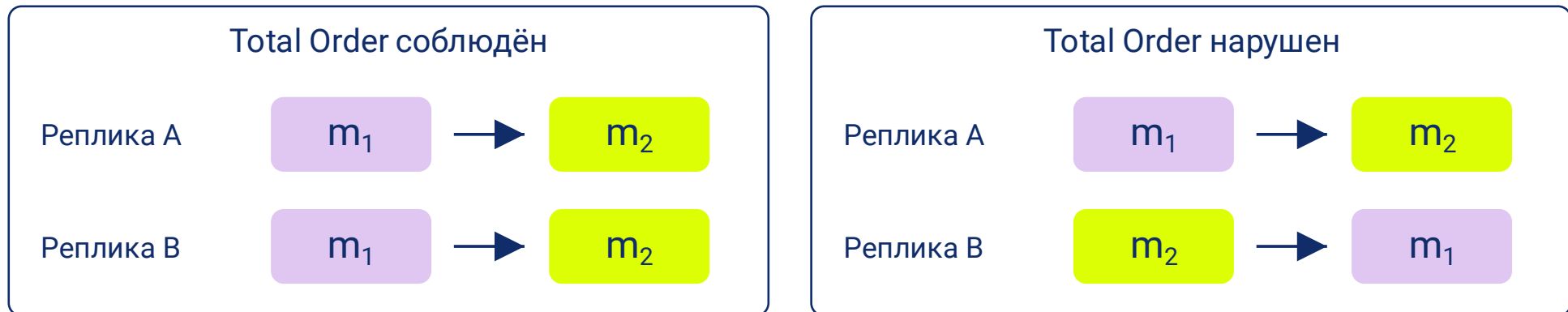


Причинный порядок соблюден, но место закреплено за разными клиентами

Общий порядок доставки – Total Order

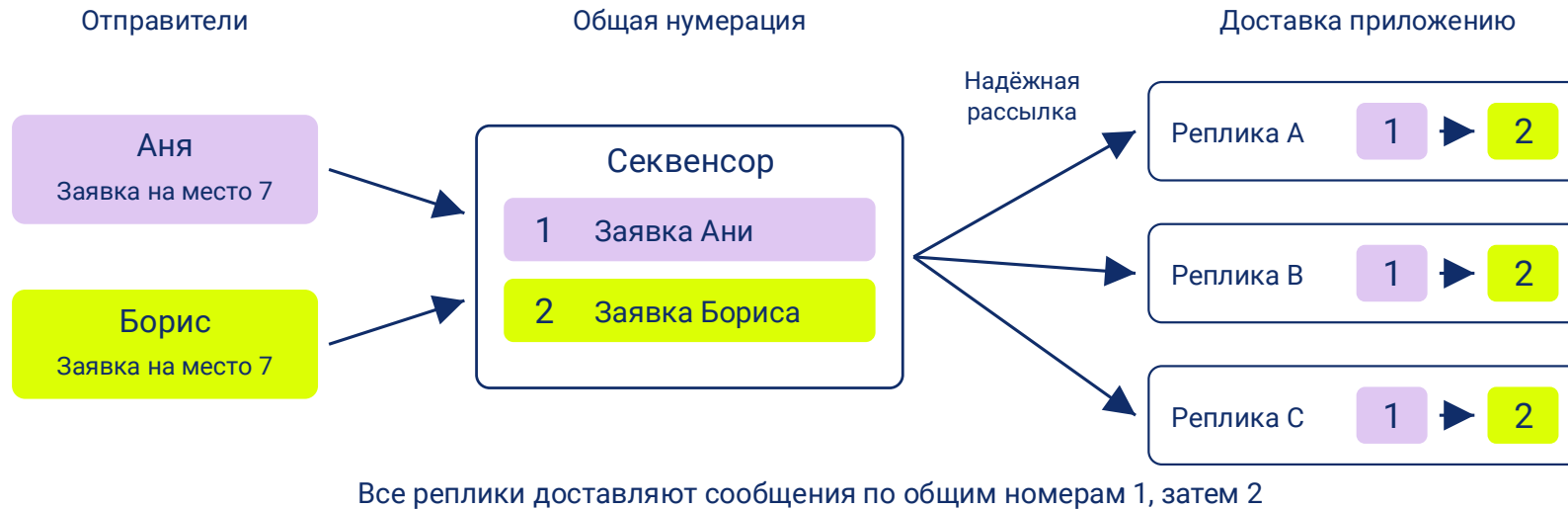
Любые два корректных процесса, доставившие сообщения m_1 и m_2 , доставляют их в одинаковом порядке.

- Порядок должен совпадать у всех корректных получателей.
- Время доставки может различаться.
- Сам по себе Total Order не гарантирует FIFO или причинный порядок.



Показан порядок доставки сообщений приложению

Общий порядок через секвенсор



1. Отправители передают сообщения выделенному процессу – **секвенсору**.
2. Секвенсор назначает общие номера 1, 2, 3, ... и рассылает сообщения группе.
3. Получатели доставляют сообщения **по порядку номеров**, ожидая пропущенные.

Что если секвенсор откажет? Смену лидера с сохранением согласованного порядка рассмотрим при изучении **консенсуса** далее в курсе.

Другие подходы: распределённое согласование порядка с помощью логических часов и подтверждений.

Сравнение гарантий порядка

Свойство	Требование
FIFO Order	Сохраняется порядок сообщений каждого отправителя.
Causal Order	Сохраняются причинные зависимости, включая порядок FIFO.
Total Order	Получатели доставляют сообщения в одинаковом относительном порядке.

FIFO и **Causal** задают *частичный порядок*: некоторые сообщения могут доставляться в разной последовательности.

Total задаёт общий *полный порядок*, но сам по себе не гарантирует FIFO или причинный порядок.

Возможные сочетания:

- **FIFO-Total**: общий порядок с сохранением FIFO.
- **Causal-Total**: общий порядок с сохранением причинных зависимостей.

Надёжность и порядок доставки

- **Надёжность:** кто обязан доставить сообщение при отказах?
- **Порядок:** в какой последовательности доставлять сообщения?

Требования сочетаются:

Название	Гарантии
Reliable FIFO Broadcast	RB + FIFO Order
Reliable Causal Broadcast	RB + Causal Order
Reliable Total Order Broadcast, или Atomic Broadcast	RB + Total Order

Атомарная рассылка и консенсус взаимно сводятся друг к другу в асинхронной модели с отказами остановкой. Связь с консенсусом рассмотрим далее в курсе.

Масштабирование рассылки

До сих пор: **фиксированный состав** группы, каждый процесс **знает всех** участников.

Реальные приложения могут быть сложнее:

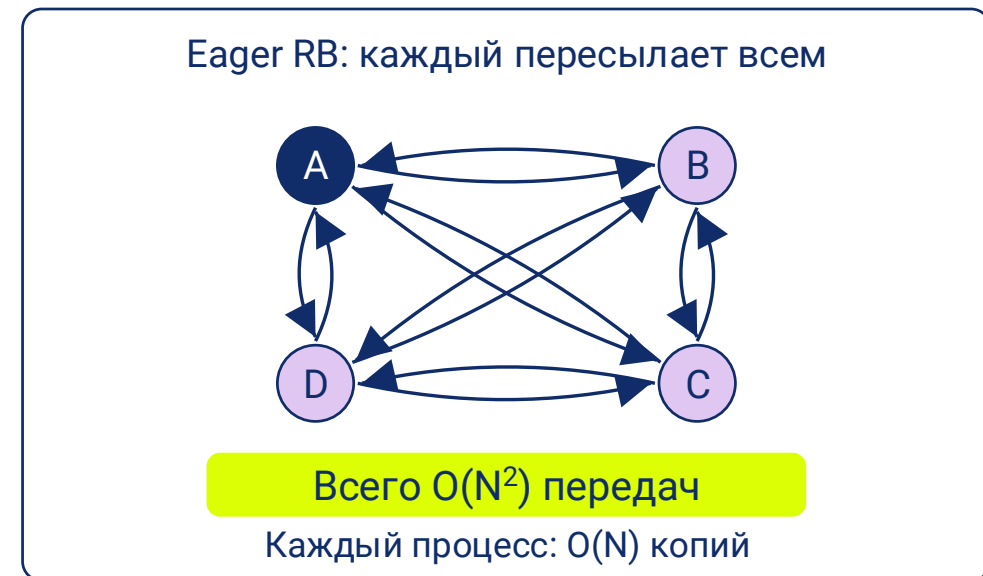
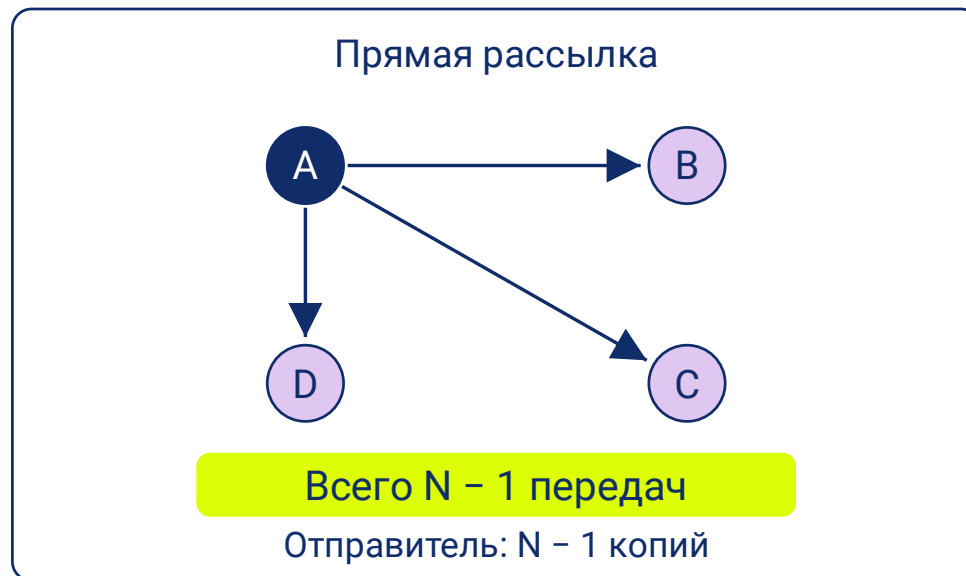
- **Большой канал:** рассылка подписчикам и сбор подтверждений создают нагрузку.
- **Кластер серверов:** состав меняется, сведения об участниках обновляются не одновременно.
- **BitTorrent:** участники подключаются и уходят – нужны доступные источники данных.

Что будем сравнивать: нагрузку на процессы и сеть, скорость распространения, устойчивость к отказам и гарантии доставки.

Как сократить затраты – и чем за это придётся заплатить?

Сколько стоит рассылка?

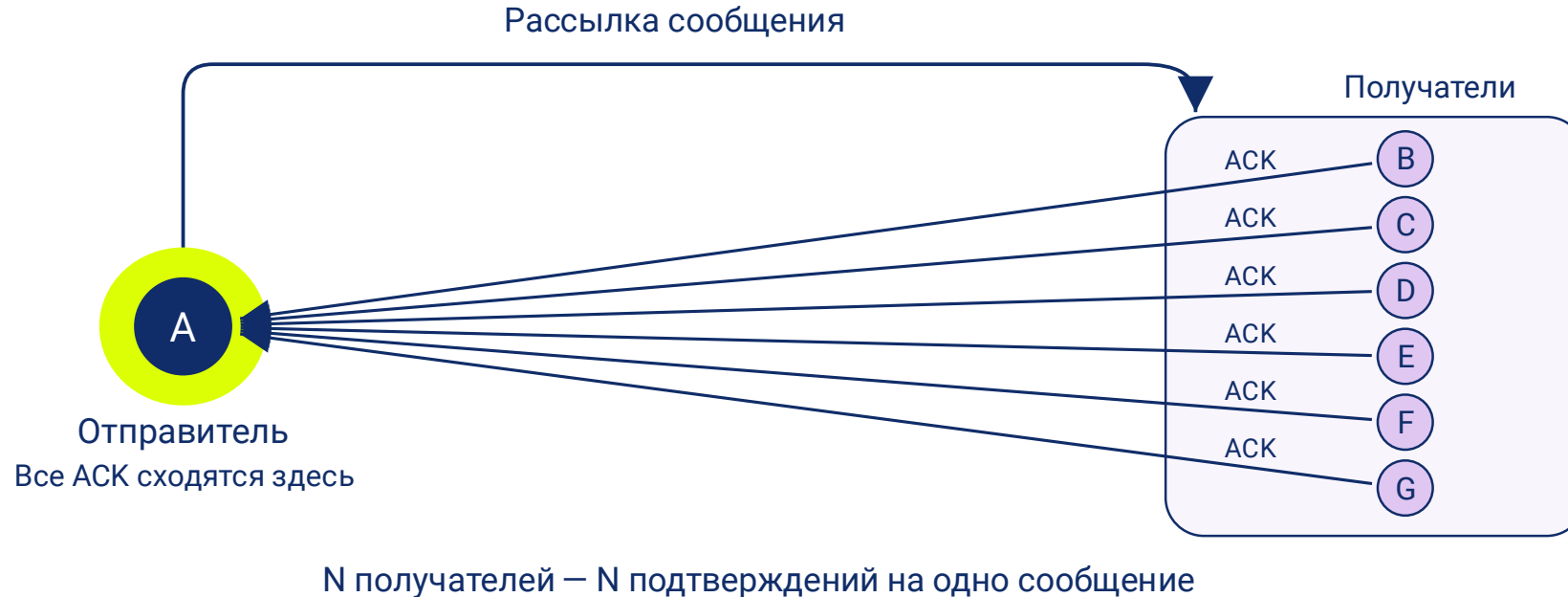
Одно сообщение, **N участников**. Считаем передачи между процессами.



Одно сообщение, без отказов; A — отправитель, N — число участников

Пересылки помогают пережить отказ отправителя, но увеличивают трафик.
Подтверждения, повторные передачи транспорта и поддержание связей здесь не учитываем.

Лавина подтверждений (ACK implosion)

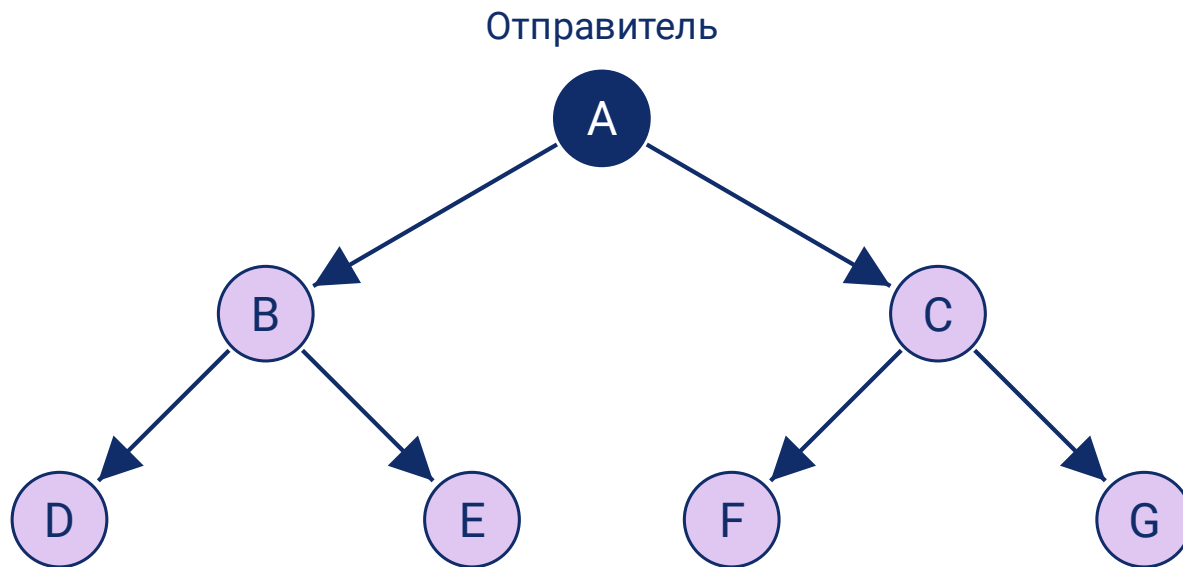


- Каждый получатель подтверждает сообщение отправителю.
- Входящий трафик и обработка ACK сосредоточены на одном процессе.

Как снизить нагрузку: объединять подтверждения, собирать их через посредников или запрашивать только пропущенные сообщения.

Рассылка по дереву

Каждый процесс пересылает сообщение только своим детям в дереве.



$N - 1$ передач всего

На рисунке: 6 передач

$O(1)$ отправок на процесс

На рисунке: не более 2 копий

$O(\log N)$ переходов

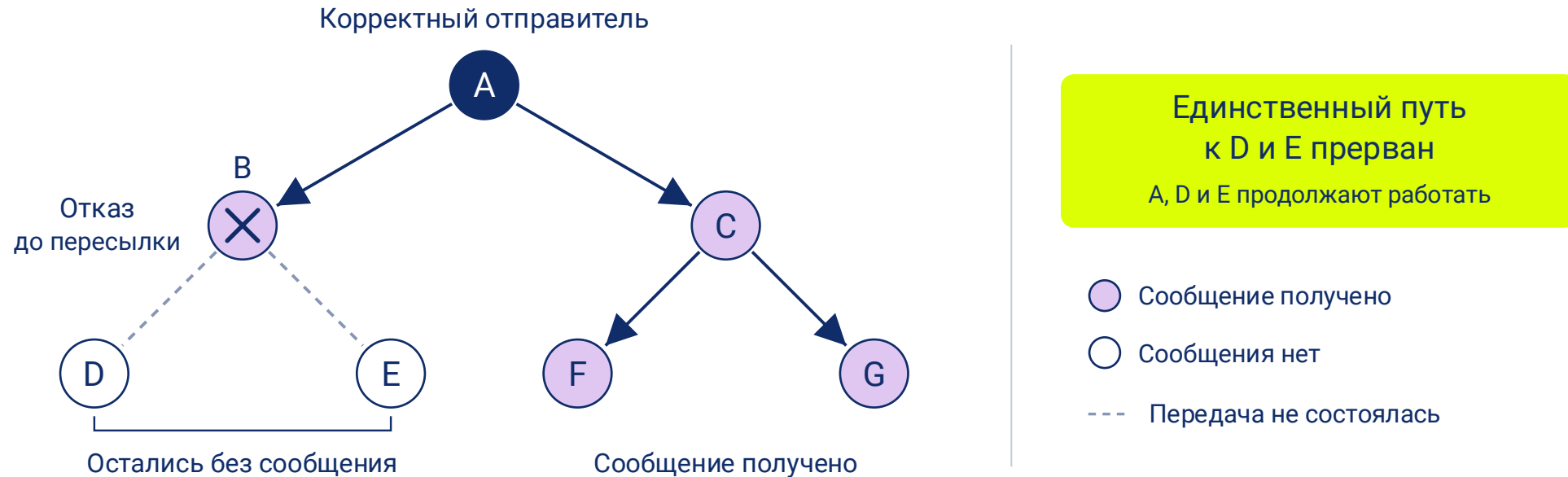
На рисунке: 2 до самого дальнего узла

Сбалансированное дерево с ограниченным числом детей N — число процессов

Одно сообщение, без отказов; без учёта подтверждений, повторов транспорта и построения дерева

Нагрузка распределяется, но сообщение проходит через посредников.

Отказ в дереве рассылки



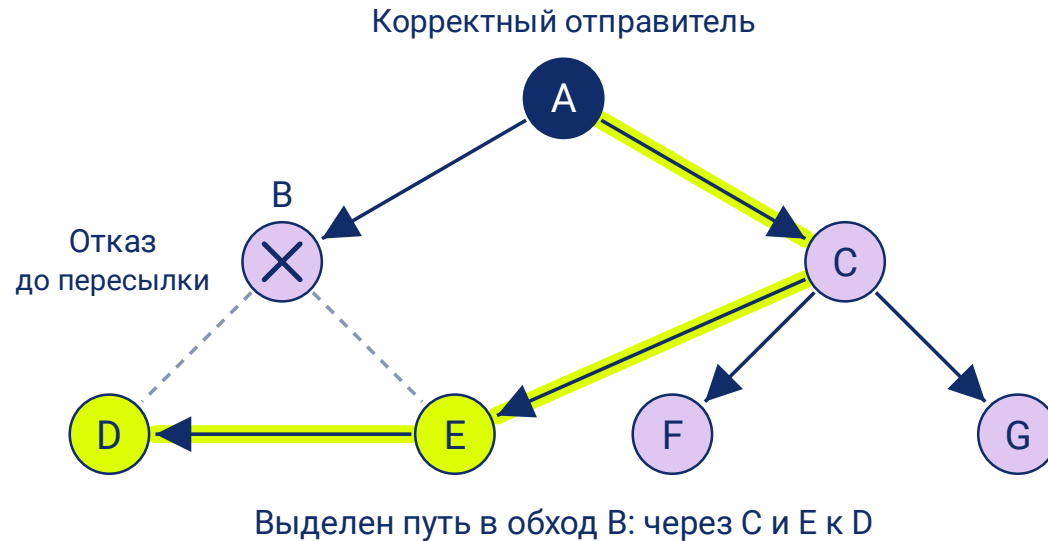
Промежуточный процесс получил сообщение и отказал **до пересылки детям**.

Его потомки могут остаться без сообщения, даже если отправитель корректен.

Единственный путь экономит передачи, но делает доставку зависимой от посредников.

Для восстановления нужны другой путь и доступная копия сообщения.

Flooding: рассылка по графу соседей



$O(E)$ передач всего

Не более $2E$ передач

E — число двусторонних связей

$O(d)$ отправок на процесс

Не более d копий своим соседям

d — число соседей процесса

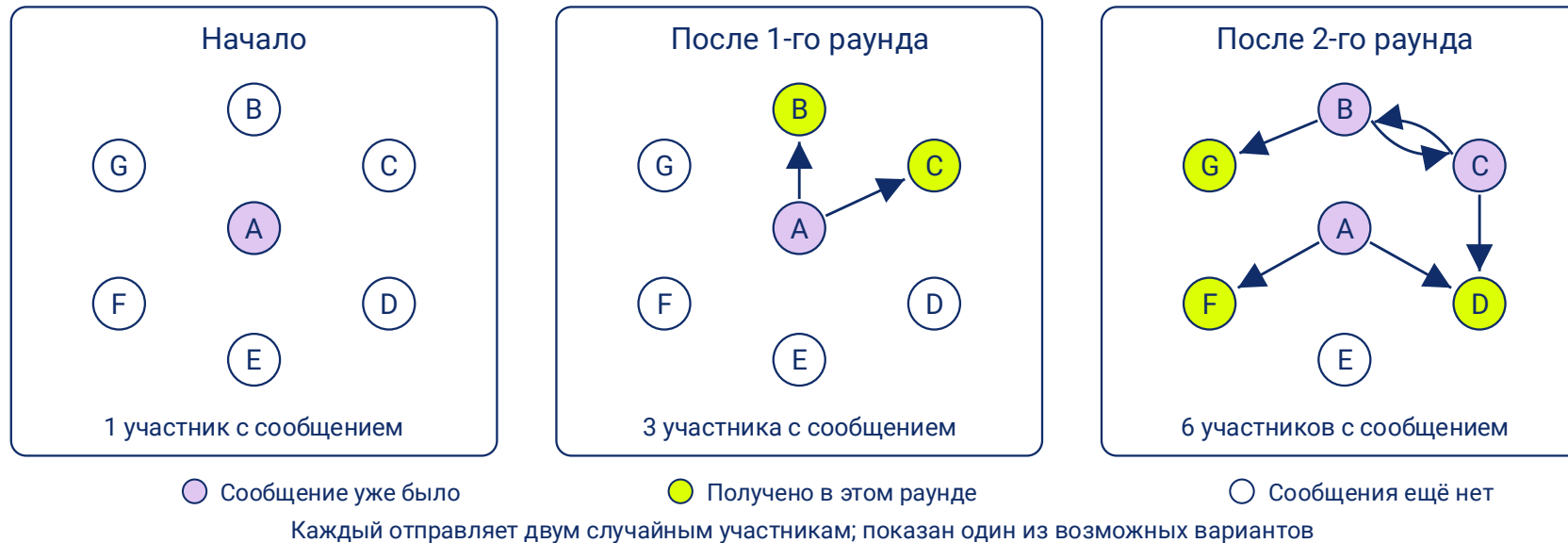
Для доставки нужен путь
через работающие процессы

- Каждый процесс поддерживает связи с несколькими **соседями**.
- Первую копию сообщения процесс **пересылает всем соседям**.
- Повторные копии распознаются по идентификатору и не пересылаются.

Дополнительные пути помогают обходить отказы, но увеличивают трафик.

Для доставки должен сохраниться путь через работающие процессы от обладателя сообщения к получателю.

Gossip: случайный выбор получателей

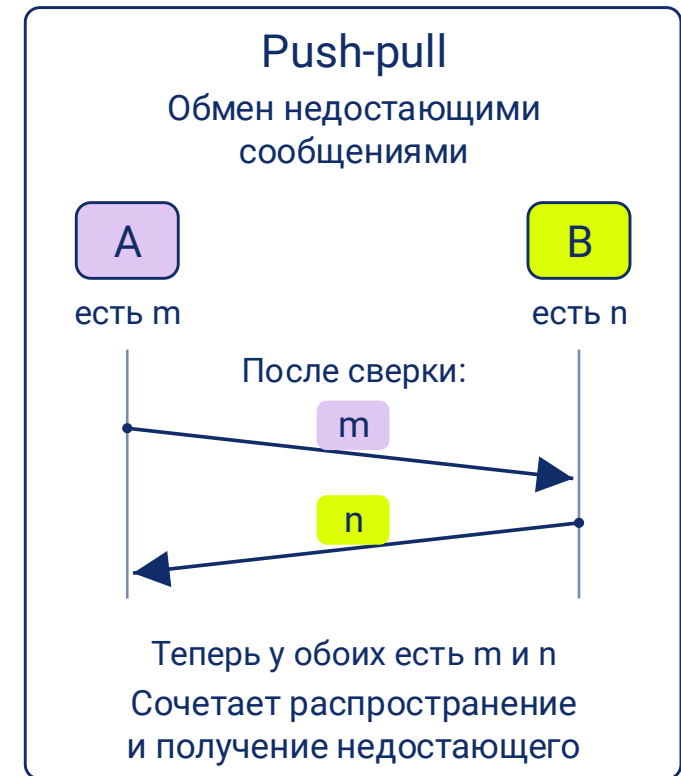
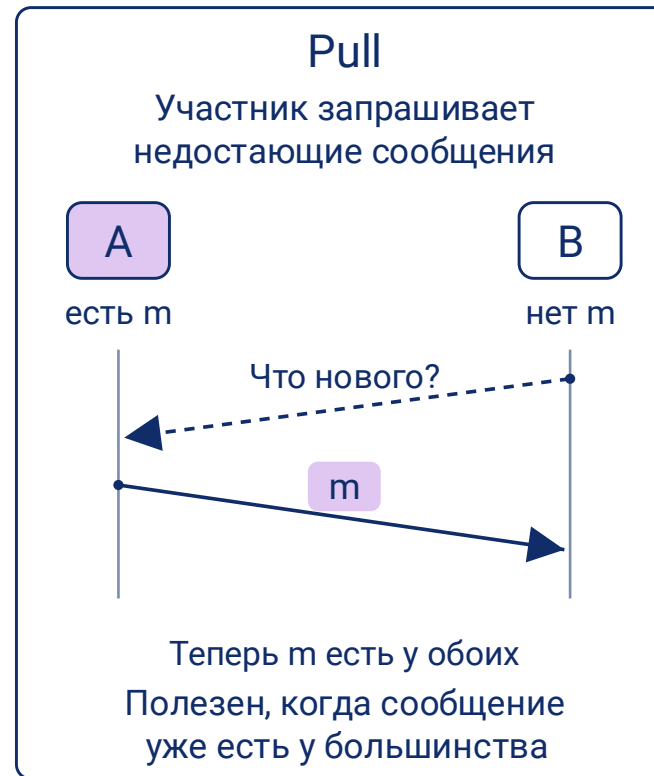
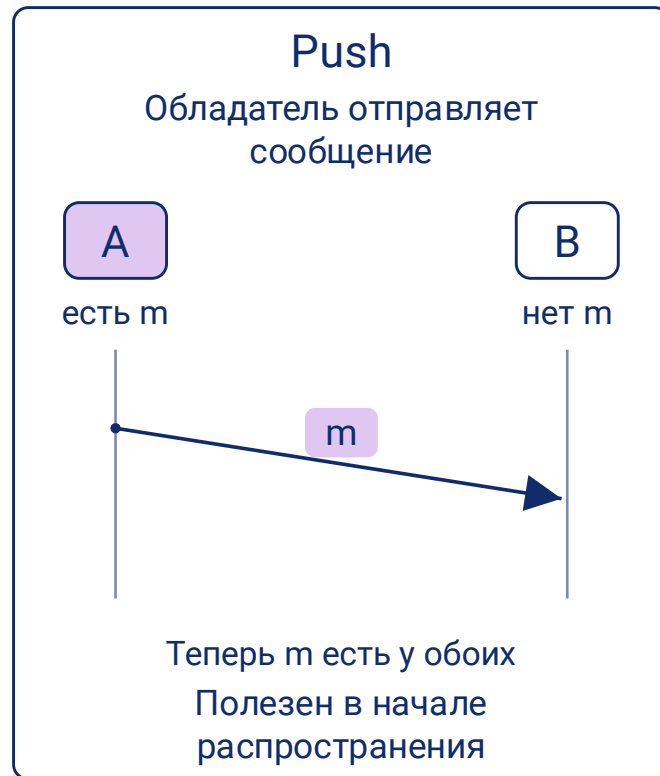


- Процесс с сообщением периодически отправляет его **k случайным участникам**.
- Получатели тоже начинают распространять сообщение.
- В каждом раунде получатели выбираются заново.

Fanout k – число получателей за один раунд.

Повторные копии распознаются по идентификатору и не доставляются приложению повторно.

Gossip: push и pull



Gossip: стоимость и скорость

Одно сообщение, **N участников**, push с **fanout k**. Рассмотрим **R раундов**.

Нагрузка	Число передач
От одного процесса за раунд	Не более k
От одного процесса за R раундов	Не более kR
Во всей группе за R раундов	Не более kNR

Скорость: в простой модели – в среднем **$O(\log N)$** раундов до полного охвата.*

Большой fanout и более частые обмены могут ускорять распространение, но увеличивают нагрузку.

** Фиксированная группа, без отказов и потерь, равномерный выбор из всех участников, постоянный k.*

Gossip: завершение и гарантии

Распространение отдельного сообщения:

ограничение числа раундов сокращает затраты, но допускает пропуски.

Постоянный обмен состоянием:

gossip работает непрерывно, новые сведения постепенно заменяют устаревшие.

Периодическая сверка помогает восстанавливать пропуски, пока нужные данные доступны.

Ограниченный gossip сам по себе не обеспечивает RB.

Гарантии порядка требуют дополнительных механизмов.

Сравнение стоимости рассылки

Одно сообщение, **N участников**. Считаем передачи между процессами.

Подход	Всего передач	Отправки одним процессом
Отправитель – всем	$N - 1$	$N - 1$ у отправителя
Eager RB	$O(N^2)$	$O(N)$
Дерево с ограниченным числом детей	$N - 1$	$O(1)$
Flooding в графе	$O(E)$	$O(d)$
Push-gossip за R раундов	Не более kNR	Не более kR

E – число связей; **d** – число соседей процесса; **k** – fanout.

Без отказов; не учитываем подтверждения, повторы транспорта и поддержание связей.

Для gossip указан бюджет передач, который не гарантирует полный охват.

Сравнение доставки и устойчивости

Подход	Скорость распространения*	Доставка при отказах
Отправитель – всем	1 раунд	Отказ отправителя может прервать рассылку
Eager RB	$O(1)$ раундов	Обеспечивает RB в принятой модели
Сбалансированное дерево	$O(\log N)$ раундов	Отказ посредника отрезает поддерево
Flooding в графе	$O(D)$ раундов	Нужны пути через корректные процессы
Push-gossip	В среднем $O(\log N)$ раундов при продолжении обменов*	При ограничении числа раундов возможны пропуски даже без отказов

D – диаметр графа.

* Скорость – в модели раундов без отказов и потерь. Для дерева число детей ограничено; для gossip – равномерный выбор из всей группы, постоянный fanout и продолжение обменов до полного охвата.

Ни один из подходов сам по себе не обеспечивает FIFO, причинный или общий порядок.

Итоги лекции

Кому доставлять?

Состав группы и гарантии надёжности: BEB, RB, URB.

В каком порядке?

FIFO, причинный и общий порядок – разные требования к доставке.

Как распространять?

Прямая рассылка, дерево, flooding, gossip – разные затраты и поведение при отказах.

Получение сообщения протоколом и доставка приложению – разные события.

Выбор алгоритма определяется требованиями приложения и моделью системы.

Материалы

- [Distributed Systems: Concepts and Design](#) (разделы 4.4, 4.5, 6.2, 15.4)
- [Distributed Systems Course](#) (разделы 4.2-4.3)
- [Distributed Systems](#) (раздел 4.4)
- [Computer Networks: A Systems Approach](#) (разделы 4.3, 9.4)
- [Сети для самых маленьких](#) (часть 9)
- [Симулятор Gossip](#)

Углубленное чтение

- [Introduction to Reliable and Secure Distributed Programming](#) (3.2-3.4, 3.8-3.9)
- [Epidemic Broadcast Trees](#)
- [Epidemic Algorithms for Replicated Database Maintenance](#)