

Взаимодействие между процессами в РС

Олег Сухорослов

Распределенные системы

Факультет компьютерных наук НИУ ВШЭ

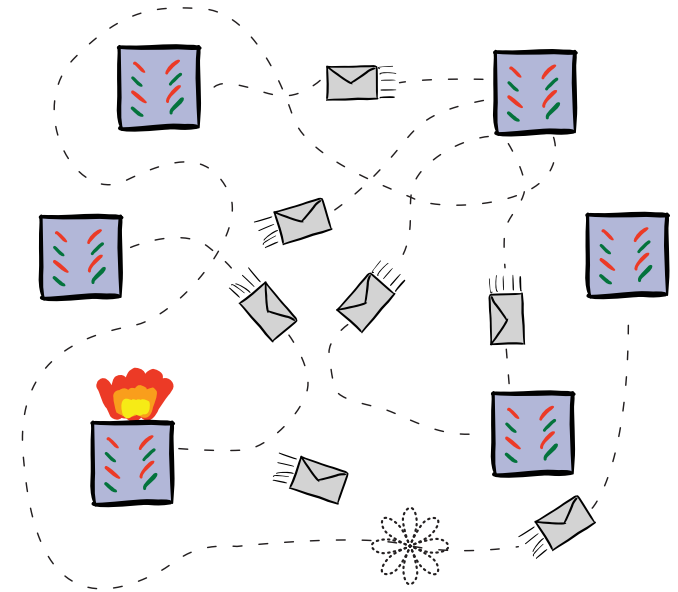
12.09.2022

План

- Возможные разновидности взаимодействий
- Передача сообщений между парой процессов
- Схема "запрос-ответ" и удаленные вызовы процедур (RPC)

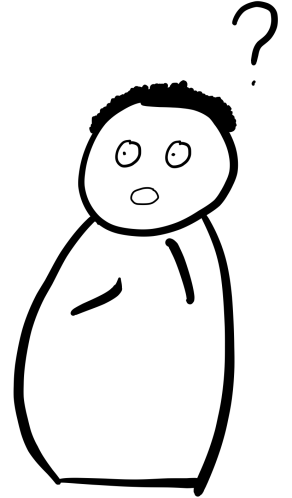
Обмен сообщениями (Message Passing)

- Наиболее общая и универсальная модель взаимодействия процессов в РС
- Сообщения передаются по ненадежным каналам
- Реализация может предоставлять некоторые гарантии по доставке сообщений
- Возможны различные варианты и схемы взаимодействий



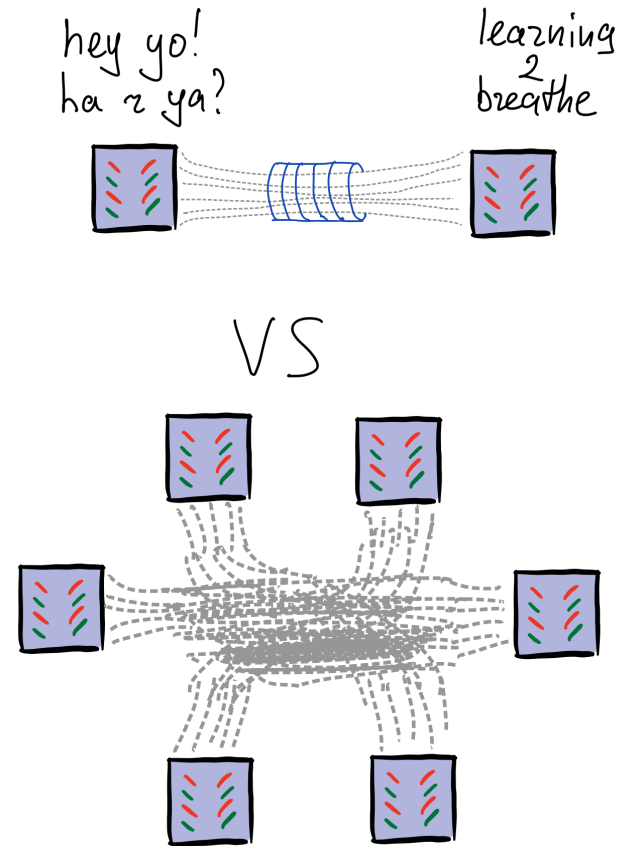
Варианты взаимодействий

- Сколько процессов взаимодействует?
- В каких направлениях передаются сообщения?
- Требуется ли ответ от получателя?
- Кто инициирует передачу сообщения?
- Должны ли отправитель и получатель "знать" друг друга?
- Должны ли процессы работать одновременно?
- Блокируются ли процессы во время отправки/приема сообщений?



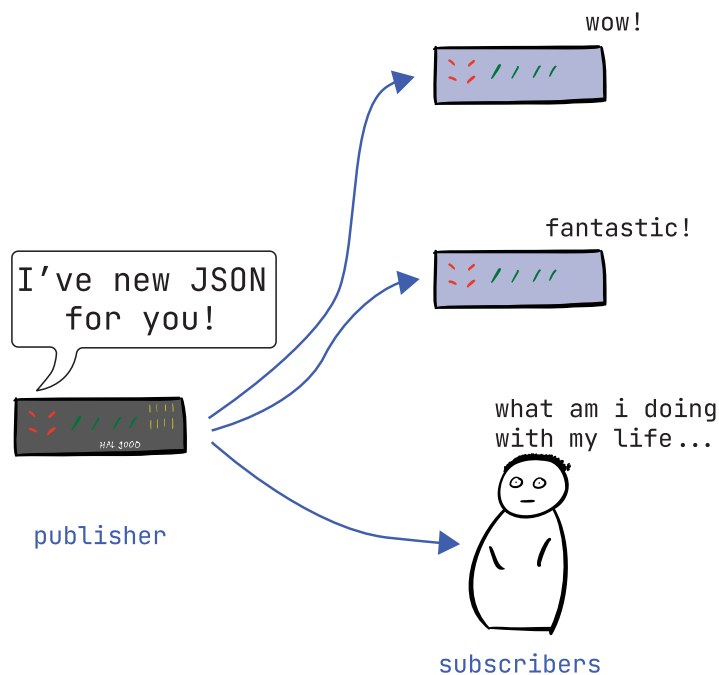
Число процессов

- Парные взаимодействия
- Групповые взаимодействия

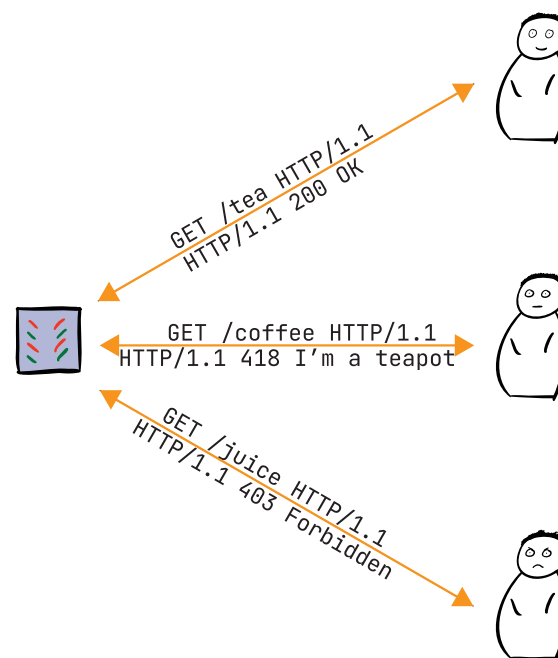


Направления передачи сообщений

- В одну сторону (unidirectional)
 - роли отправителей и получателей зафиксированы
 - пример: издатель-подписчик

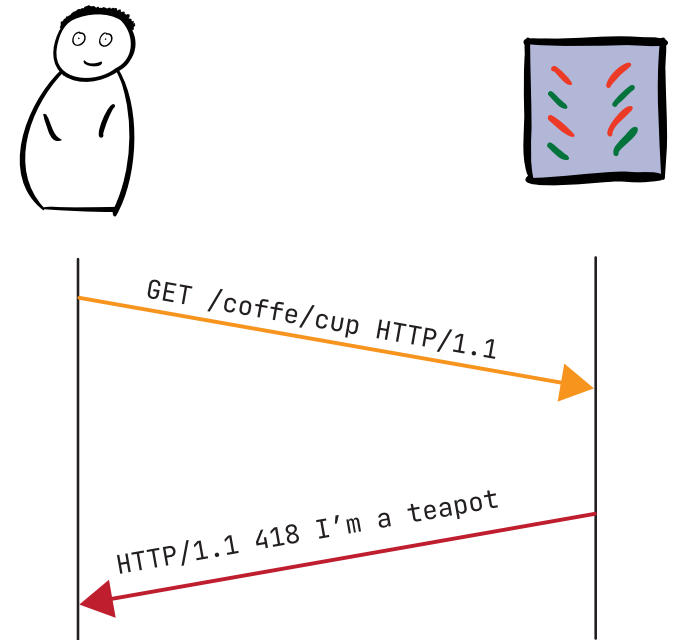


- В обе стороны (bidirectional)
 - процесс может быть как отправителем, так и получателем
 - пример: клиент-сервер



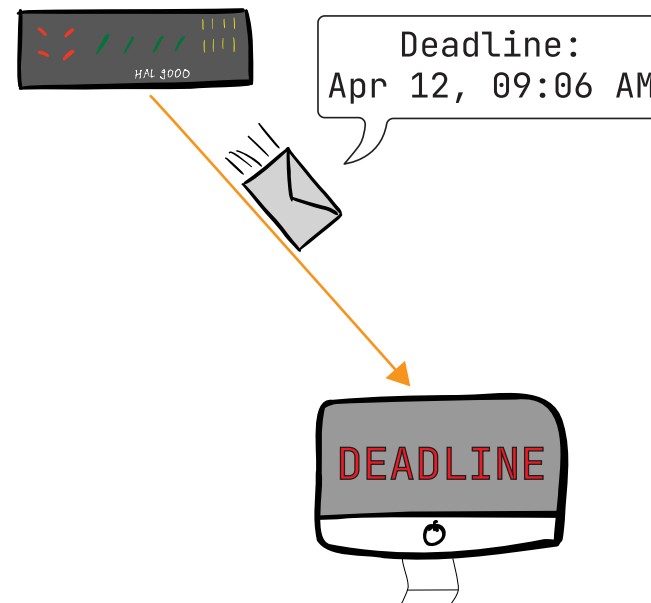
Требуется ли ответ от получателя?

- Отправка в одну сторону (one-way)
- Схема "запрос-ответ" (request-reply)



Кто инициирует передачу сообщения?

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



Связывание процессов

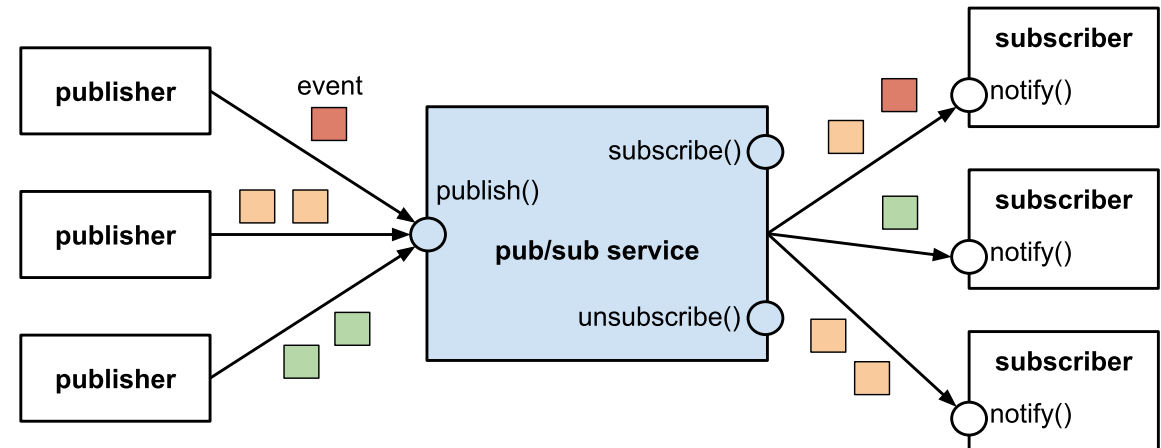
- Связывание по пространству (space coupling)
 - Процессы должны обладать информацией друг о друге
 - Например, отправитель должен знать адрес получателя
- Связывание по времени (time coupling)
 - Процессы должны выполняться в одно время
 - Transient vs persistent communication

Непрямое взаимодействие (indirect)

Происходит через некоторого посредника или абстракцию, без прямого связывания между отправителями и получателями

Примеры

- Очередь сообщений
- Издатель-подписчик
- Общая память

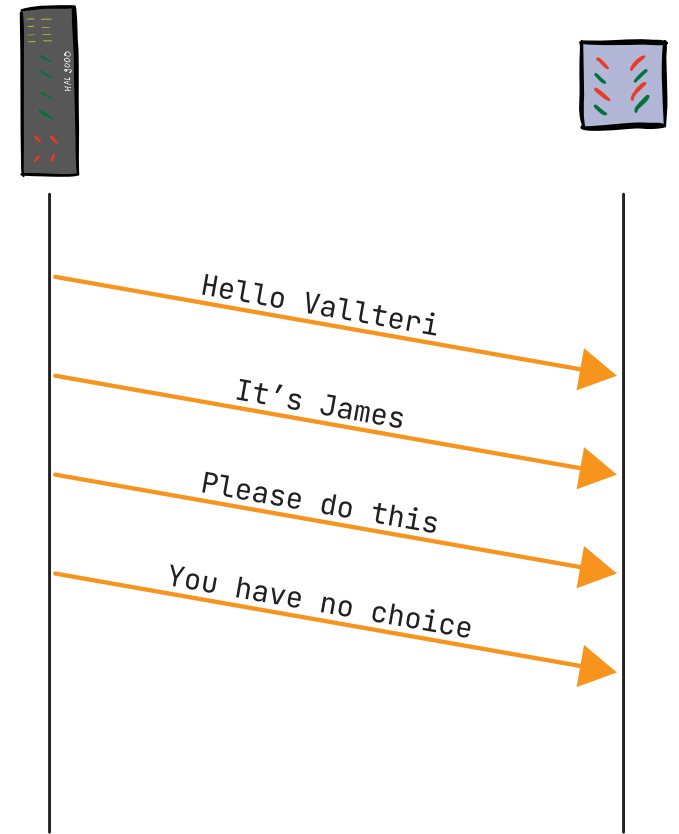


Блокируются ли процессы?

- Синхронное взаимодействие
 - отправитель блокируется (до приема сообщения к доставке, доставки сообщения, окончания обработки сообщения...)
 - получатель блокируется до приема сообщения
- Асинхронное взаимодействие
 - отправитель продолжает выполнение сразу после отправки сообщения
 - получатель может неблокирующим образом проверить наличие сообщений
 - позволяет перекрыть коммуникации и вычисления

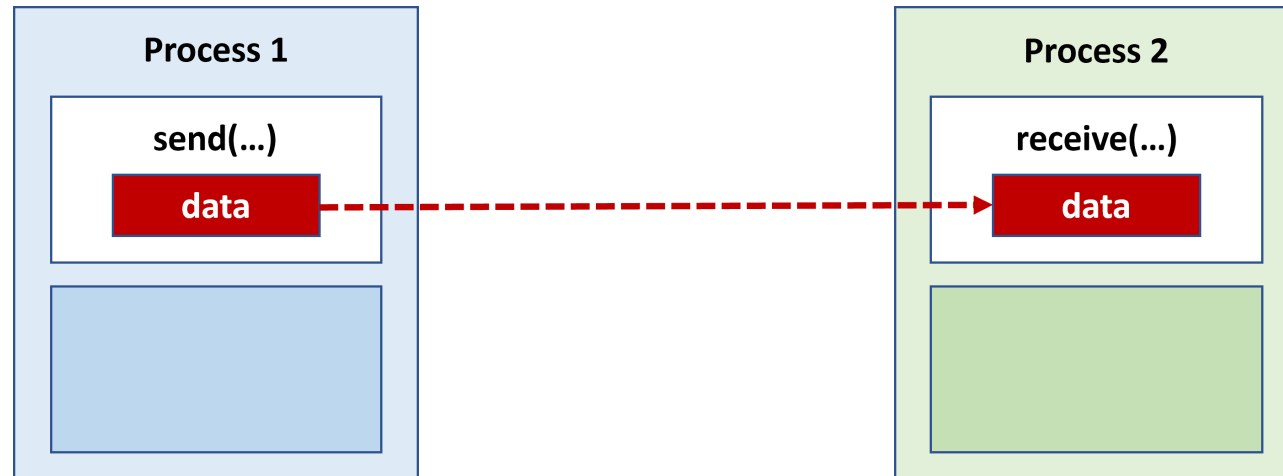
Простейший вариант

- Два процесса
- Один отправитель, другой получатель
- Передачу инициирует отправитель
- Отправитель знает адрес получателя
- Процессы работают одновременно



Реализация

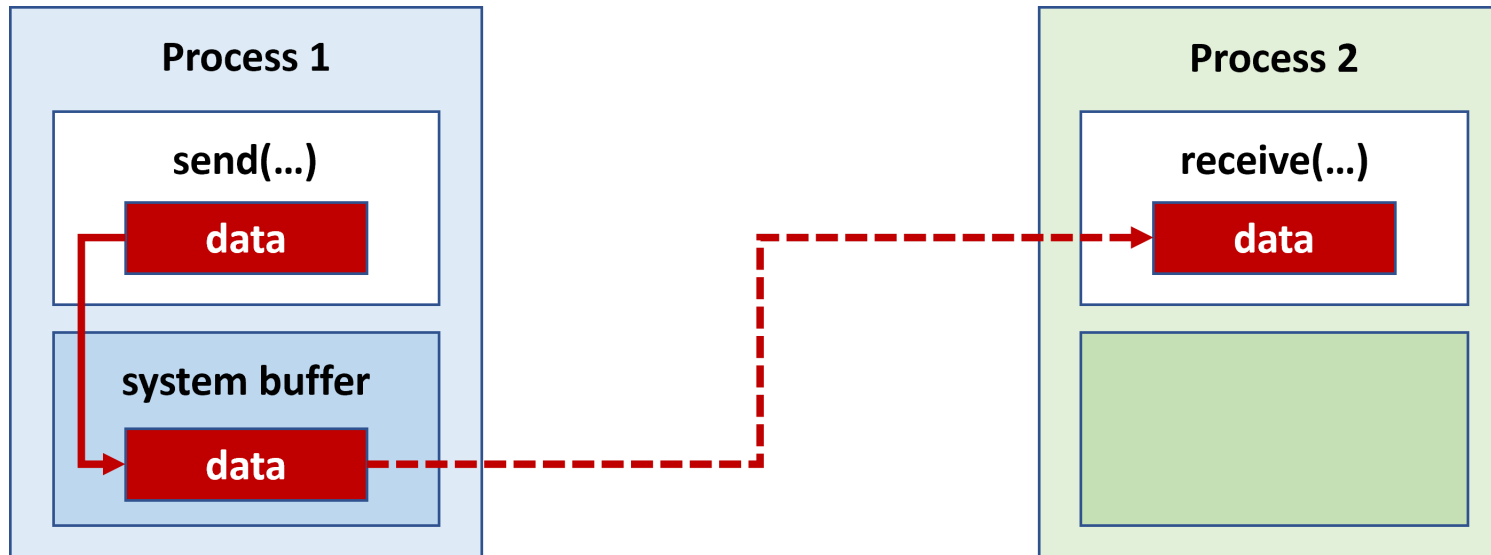
- `send(data_buf, dest)`
 - блокируется до ...
- `receive(data_buf)`
 - блокируется до получения сообщения и размещения его в буфере



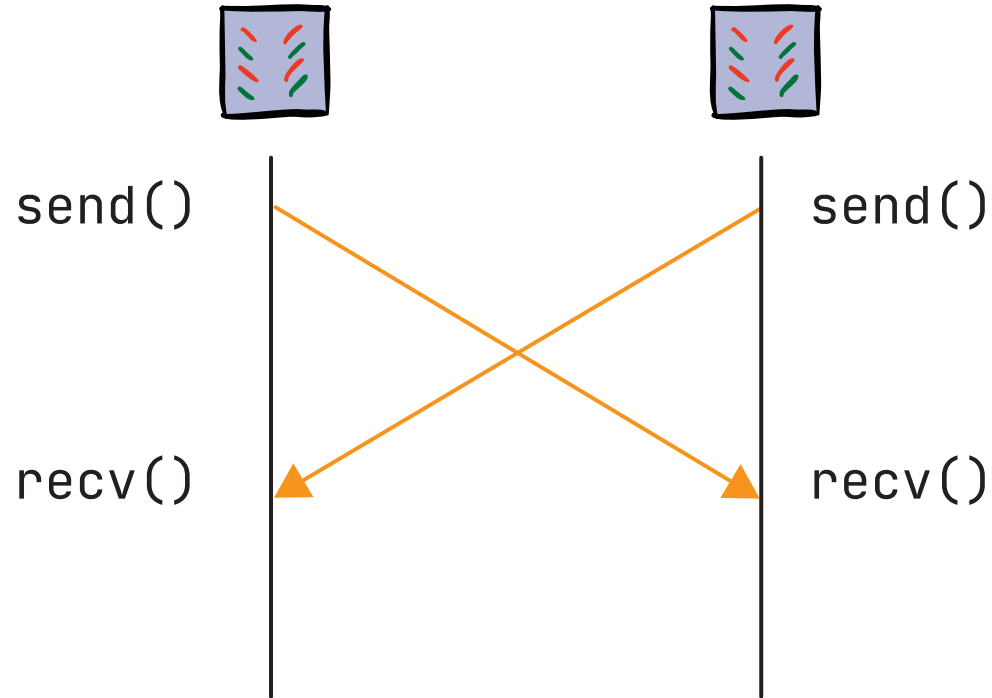
Когда завершается `send()`?

- Буфер можно повторно использовать, не опасаясь испортить передаваемое сообщение?
- Сообщение покинуло узел процесса-отправителя?
- Сообщение принято процессом-получателем?

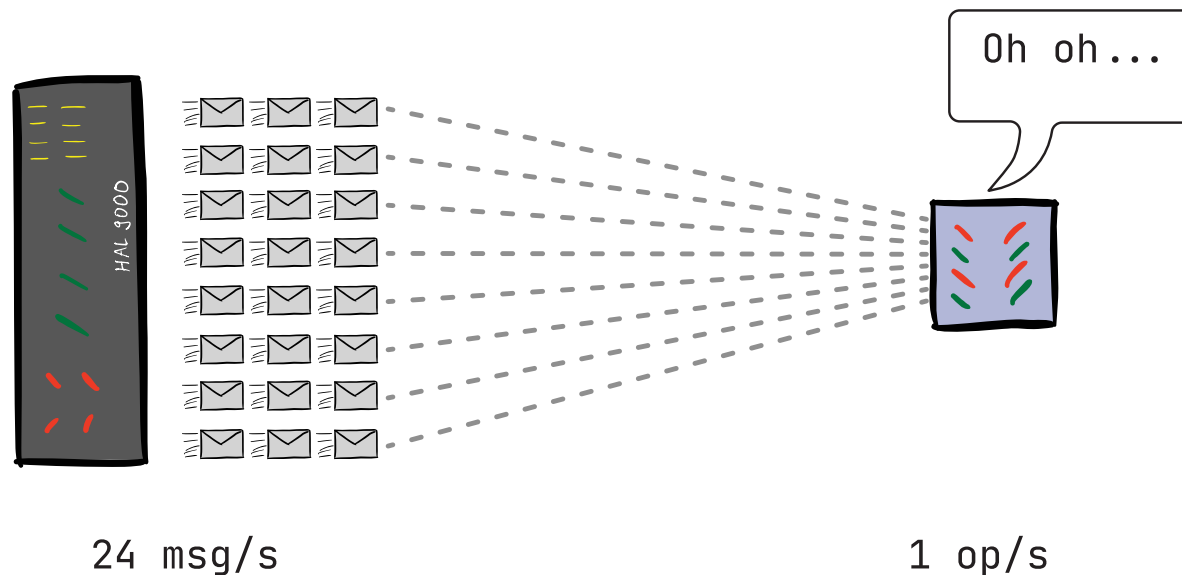
Возможная реализация



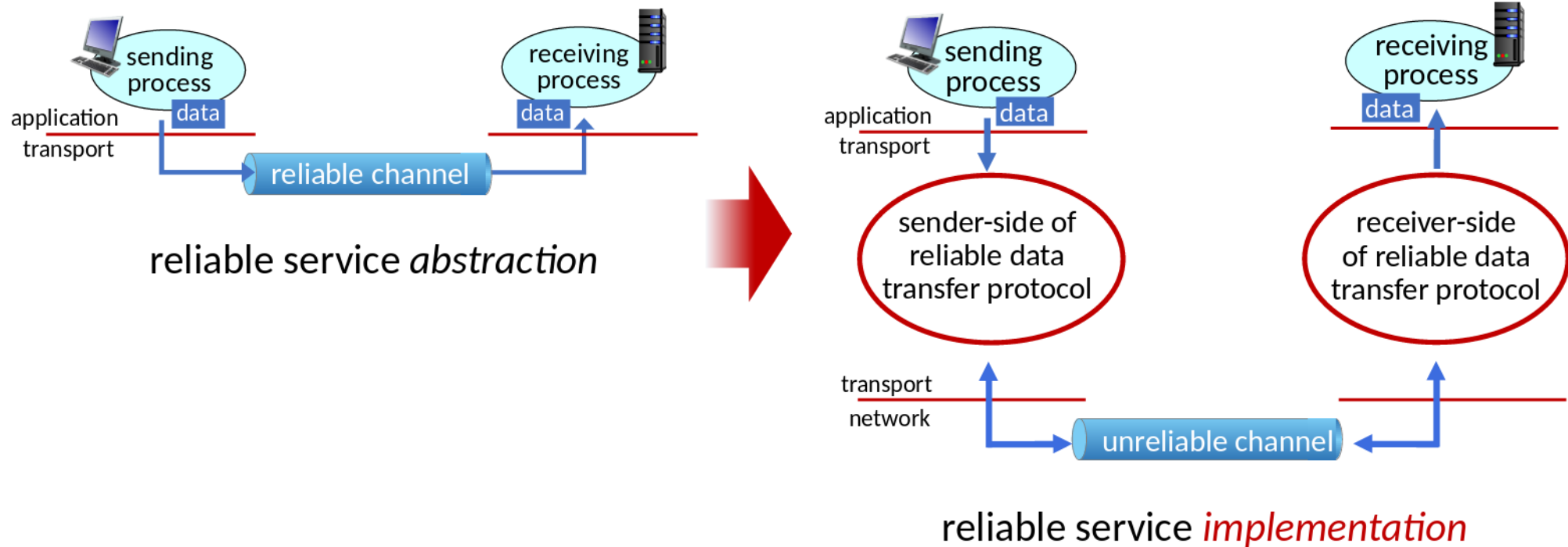
Что здесь может возникнуть?



Слишком быстрый отправитель



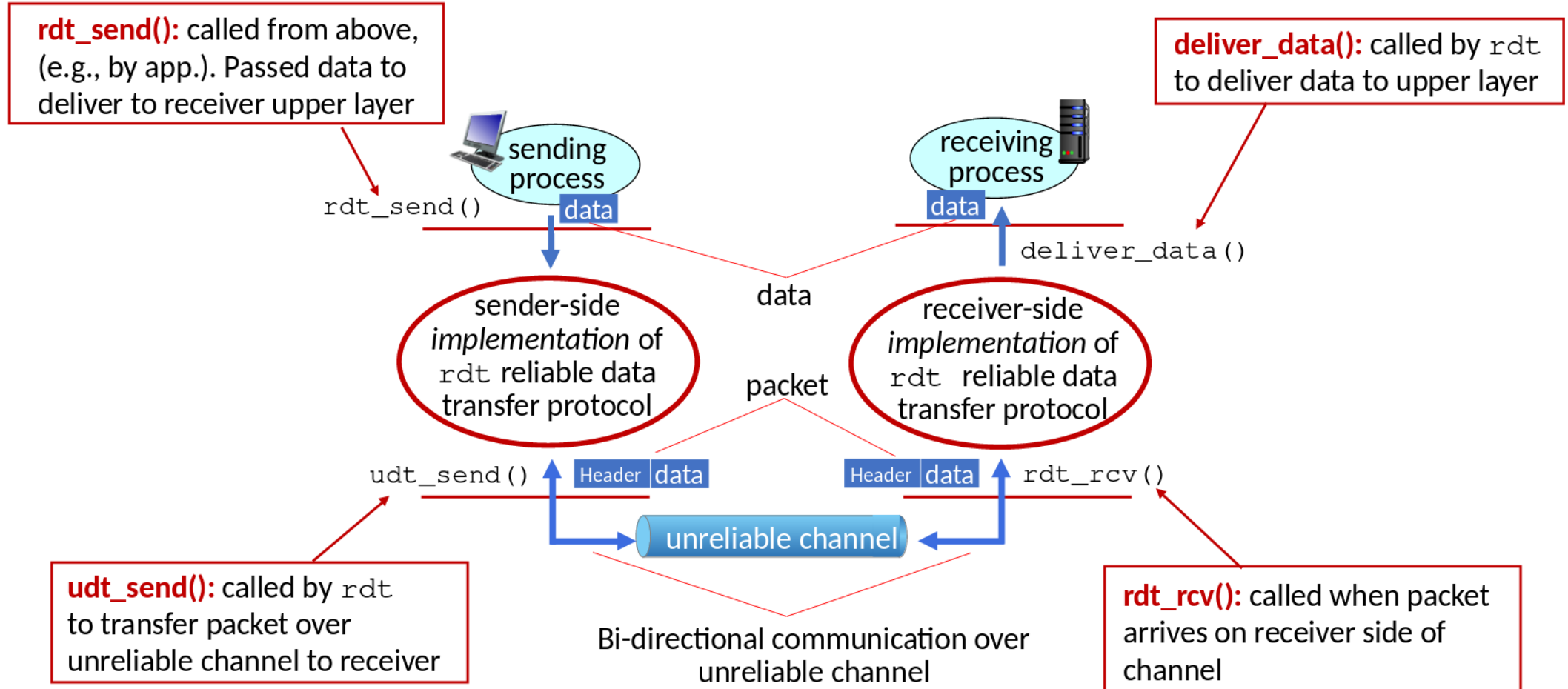
Надежная передача сообщений



Надежная передача сообщений

- Сеть представляет собой ненадежный канал передачи данных
 - Сообщения могут искажаться и теряться в процессе передачи
 - Будем считать, что канал не переупорядочивает сообщения
- Как поверх ненадежного канала реализовать абстракцию надежного канала?
 - Односторонний канал (отправитель, получатель)
 - Все отправленные сообщения доставляются получателю
- Процессы не "видят" состояний друг друга (получено ли сообщение)
 - Единственный способ узнать - передать эту информацию в сообщении
 - Для этого нам понадобится описать протокол

Интерфейсы



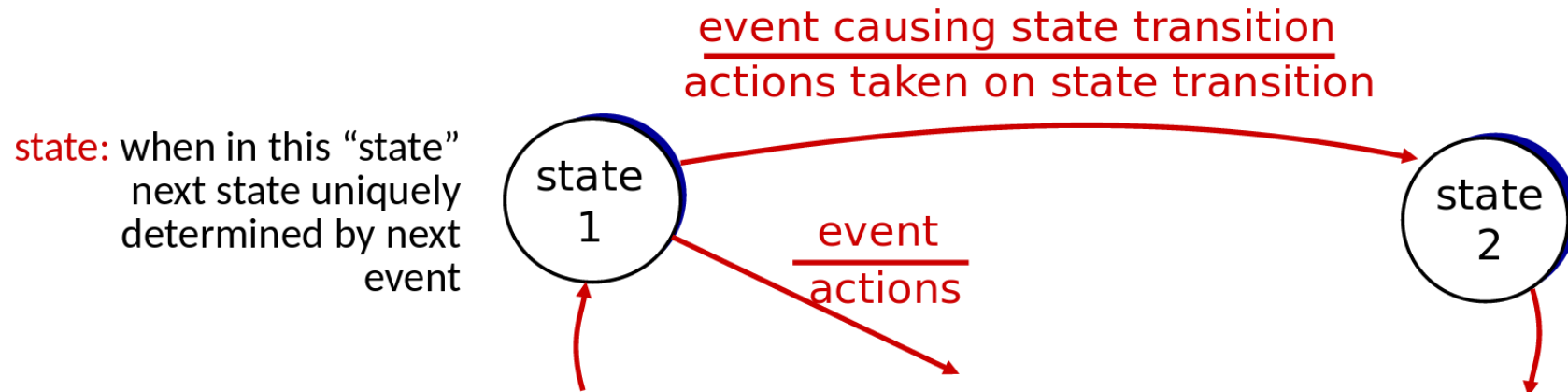
Протокол

- Описание правил взаимодействия компонентов системы
- Типы, семантика и структура сообщений
- Форматы передачи данных
- Правила обработки сообщений
- Адресация компонентов
- Управление соединением
- Обнаружение и обработка ошибок
- ...

Описание протокола

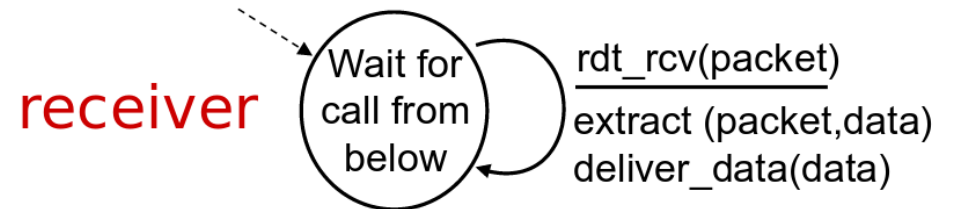
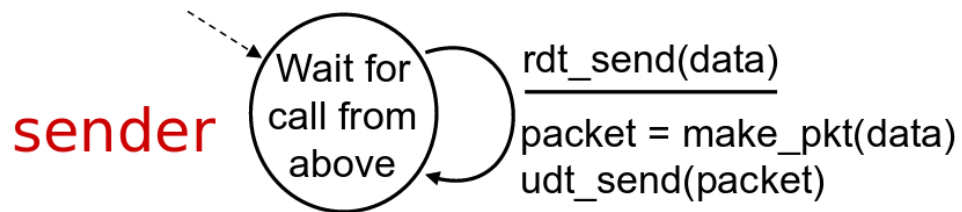
Будем использовать конечные автоматы для описания поведения отправителя и получателя в нашем протоколе

- Состояния, в которых может находиться процесс
- События, приводящие к переходу между состояниями
- Действия, выполняемые во время переходов между состояниями



Протокол 1.0

- Нижележащий канал - надежный
 - сообщения передаются без ошибок (bit errors)
 - сообщения не теряются (packet loss)
- Конечные автоматы для отправителя и получателя:



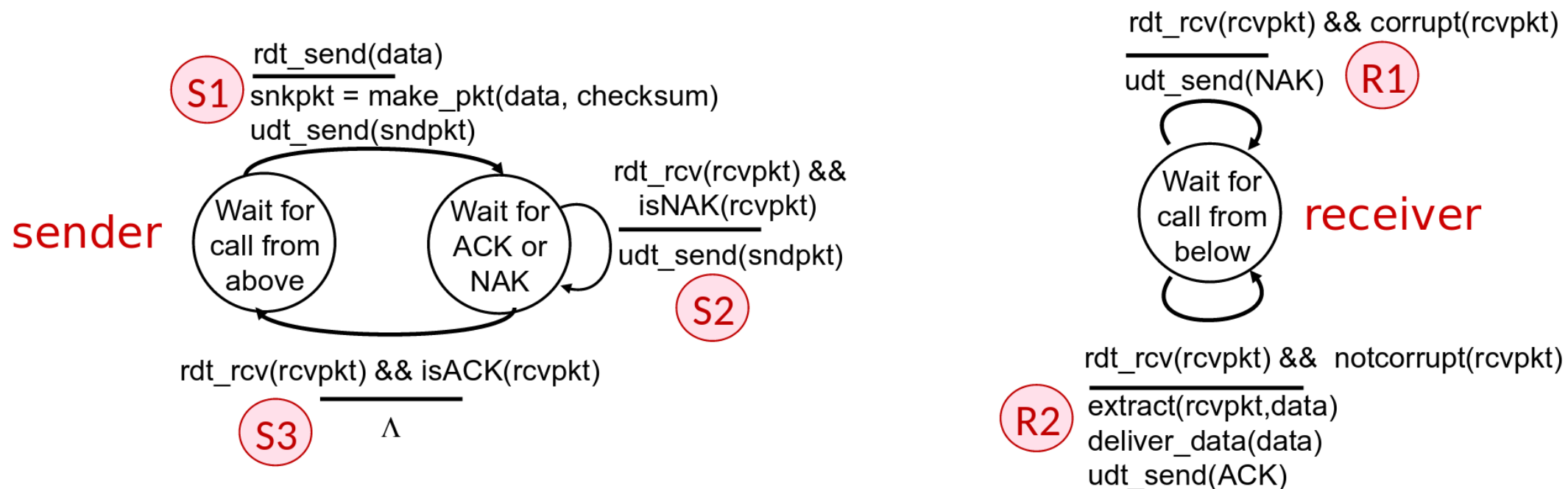
Канал с ошибками

- Содержимое сообщения при передаче может повредиться
 - например, бит 1 вместо 0
- Как обнаруживать и обрабатывать такие ошибки?

Служебные сообщения

- ACK (acknowledgement)
 - сообщение получено в целостности
- NAK (negative acknowledgement)
 - сообщение получено с ошибками

Протокол 2.0



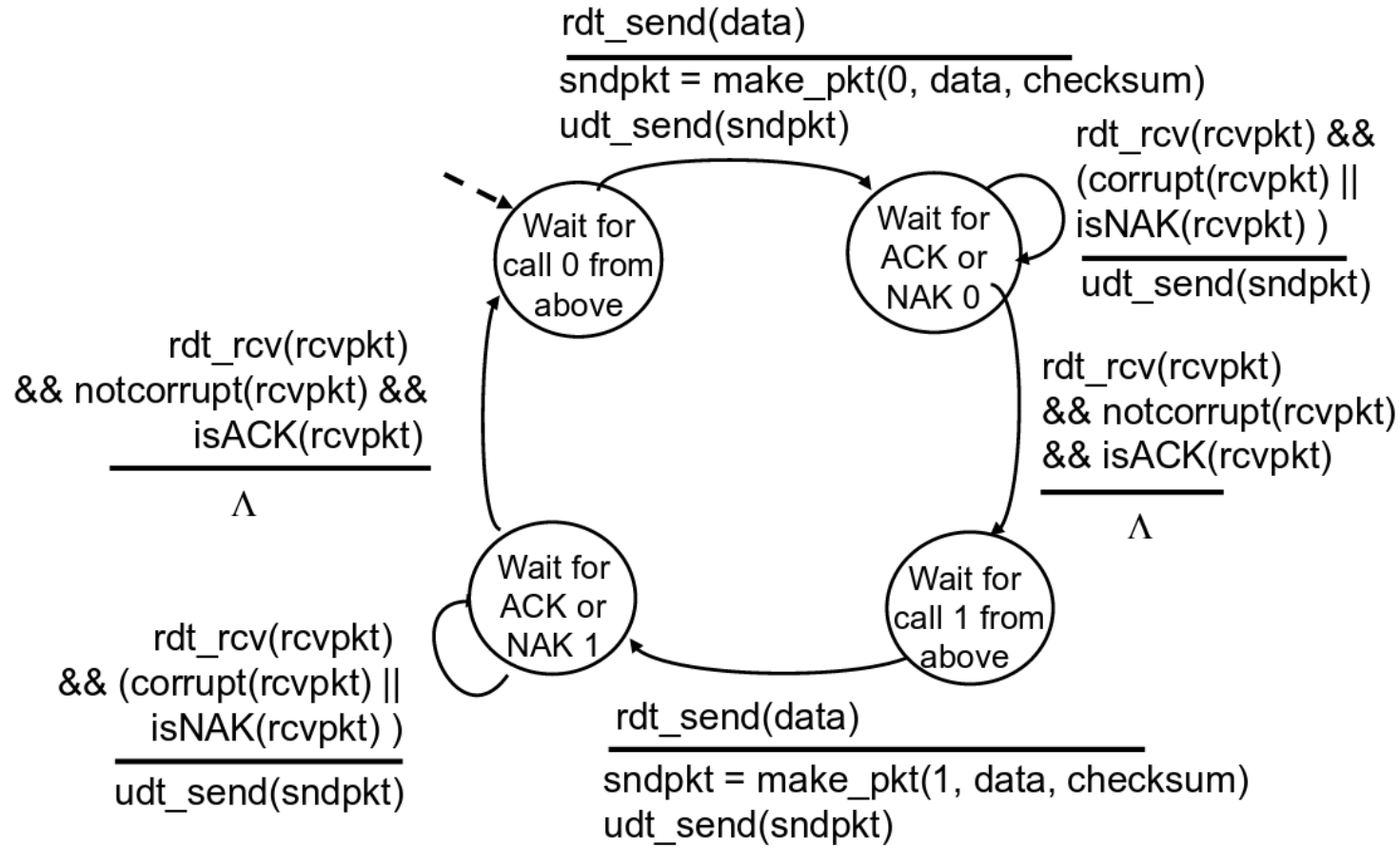
Проблема

- Что если окажется повреждено сообщение с ACK или NAK?
- Можно ли безопасно отправить исходное сообщение еще раз?

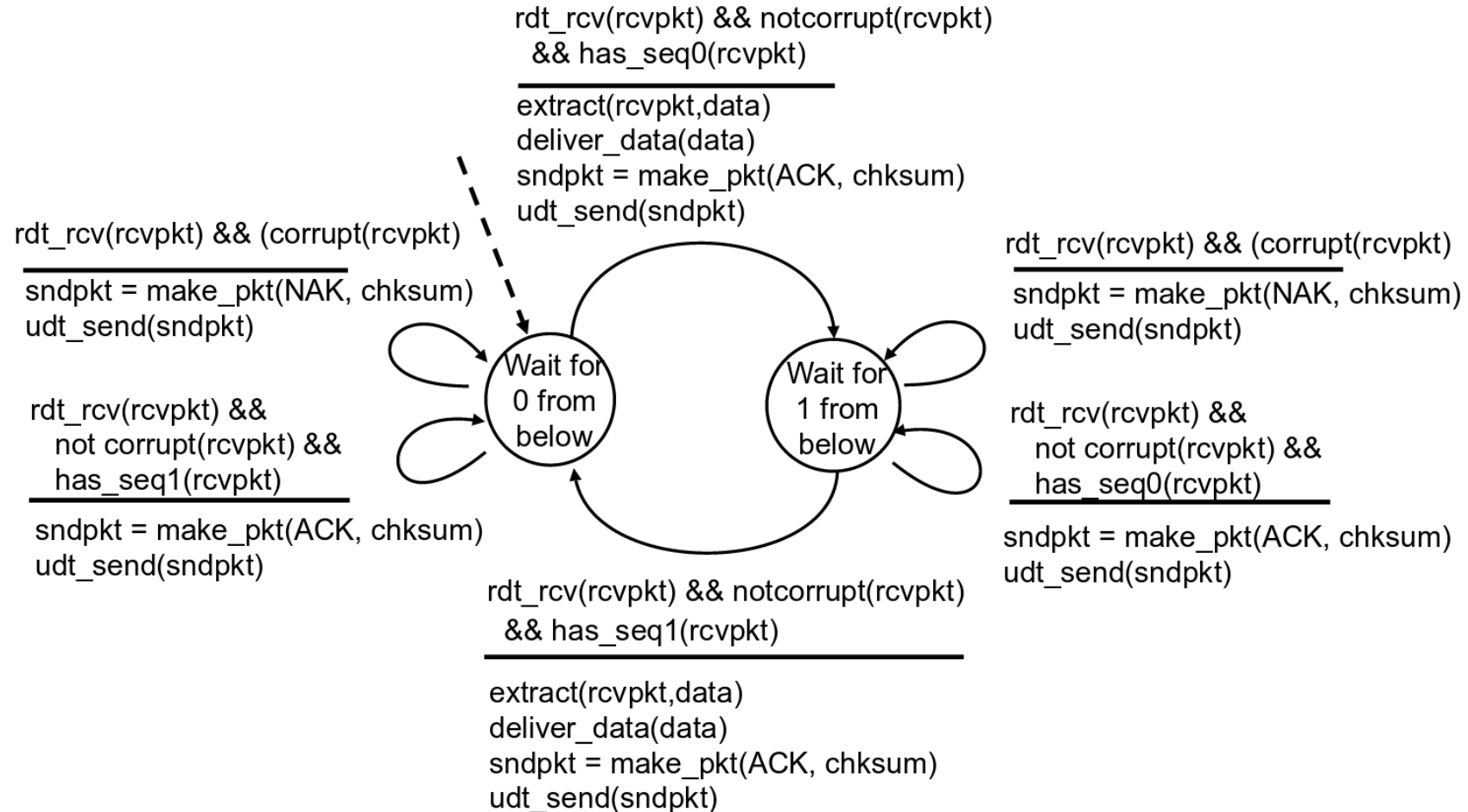
Дедупликация сообщений

- Отправитель добавляет в сообщение sequence number (SN)
 - В нашем случае достаточно чередовать 0 и 1
- Получатель ожидает сообщения с определенным SN
 - Сообщения с другим SN игнорируются (не доставляются выше)

Протокол 2.1: отправитель



Протокол 2.1: получатель



Упражнение

Протокол, который использует только ACKs (NAK-free)

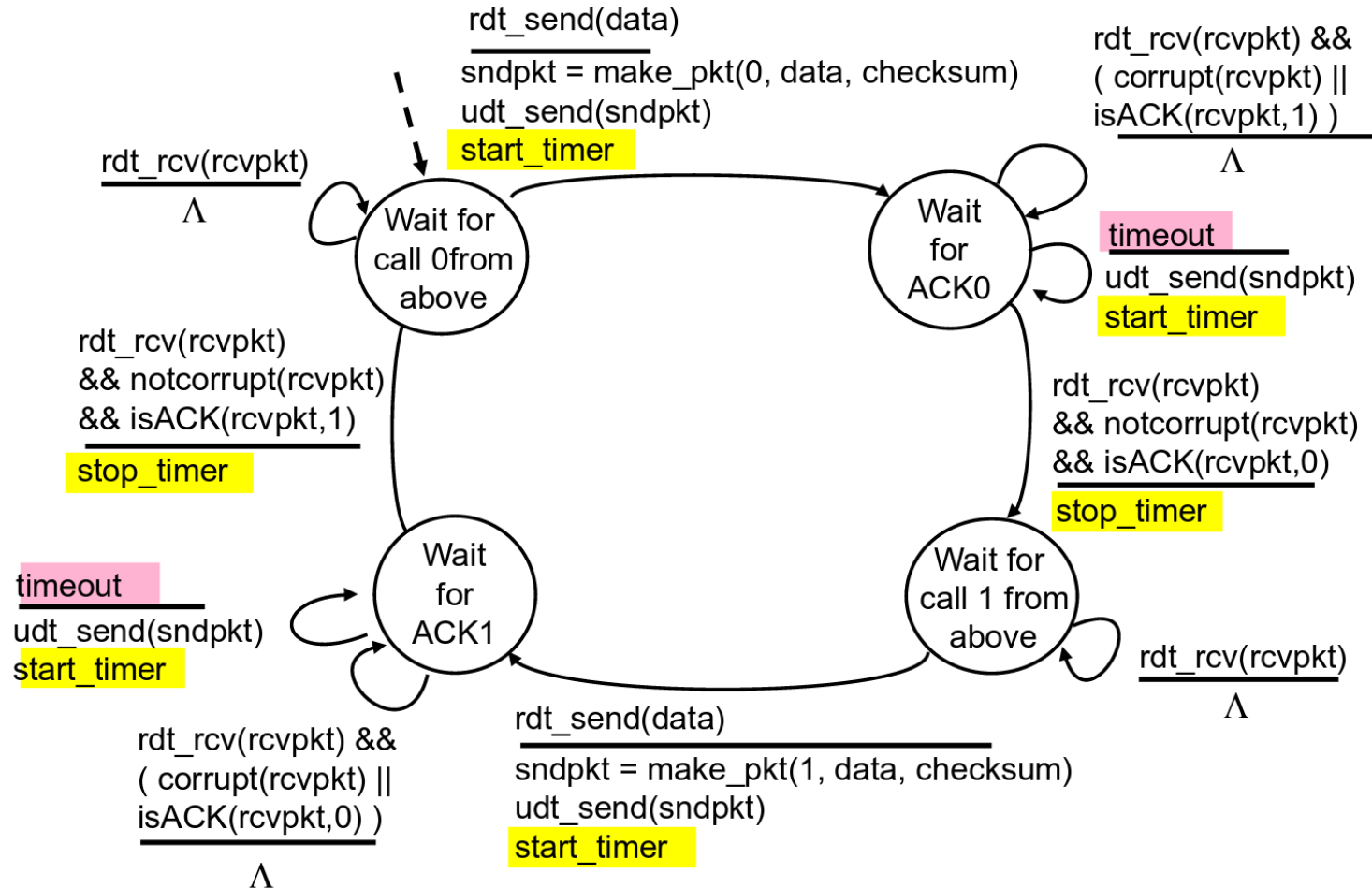
Канал с ошибками и потерей сообщений

- Теперь сообщения могут не только повреждаться, но и теряться в процессе передачи
- Как обнаруживать и обрабатывать потерю сообщения?

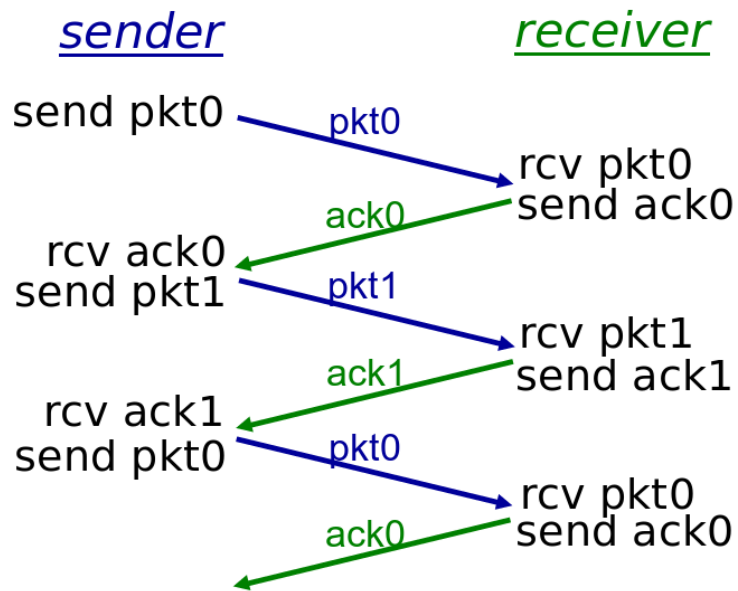
Базовые принципы

- Отправитель ожидает ACK в течение некоторого времени
 - требуется таймер на стороне отправителя
- Если ACK не получен вовремя, сообщение отправляется повторно
- Если сообщение (или ACK) не было потеряно, а задержалось
 - сообщение будет продублировано, но это решается с помощью SN
 - получатель должен указывать SN внутри ACK

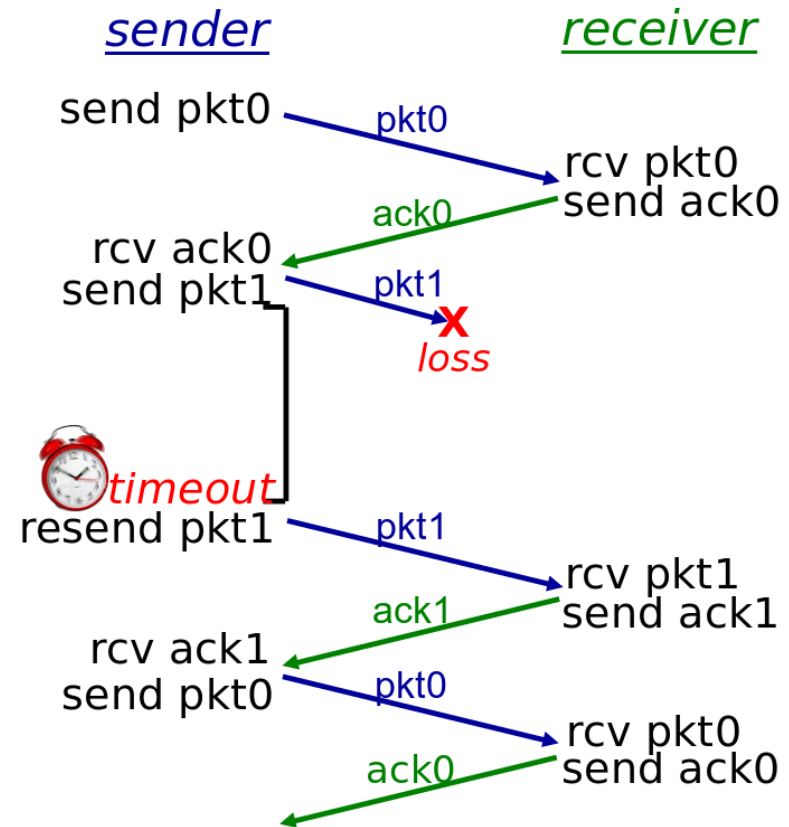
Протокол 3.0: отправитель



Возможные сценарии

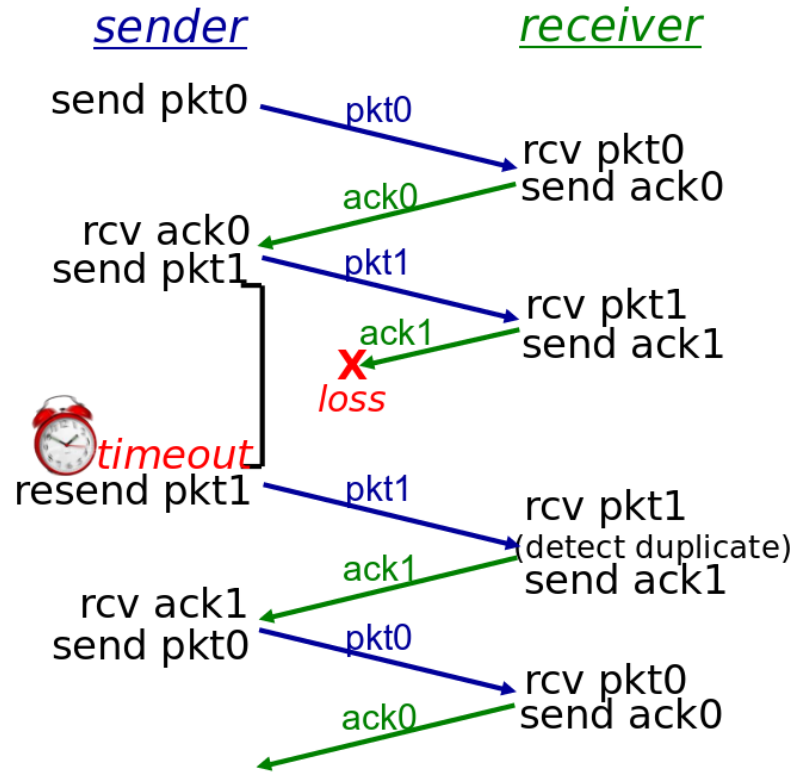


(a) no loss

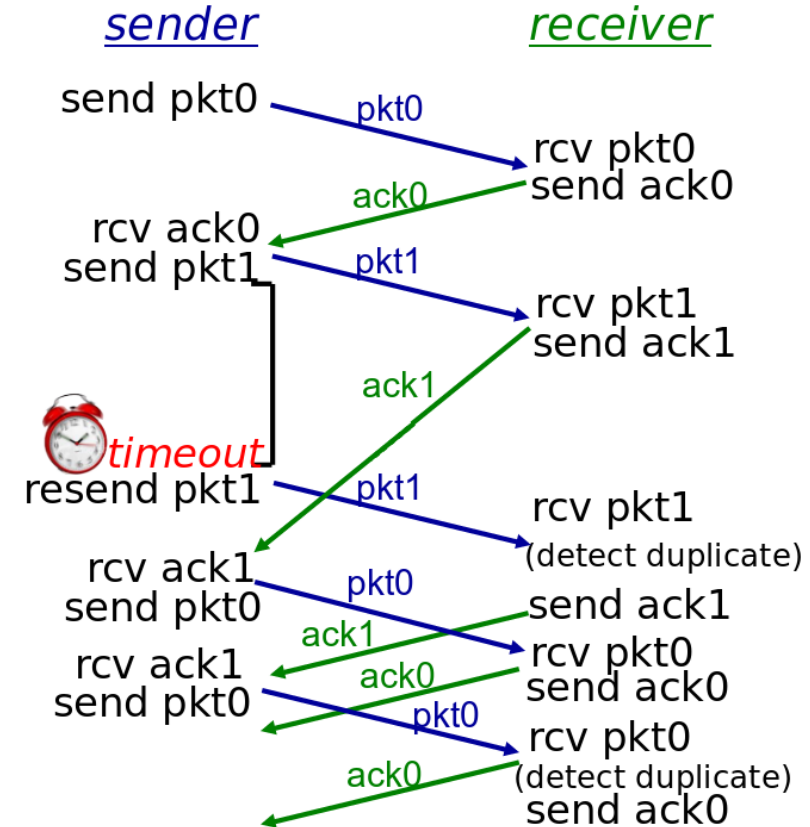


(b) packet loss

Возможные сценарии (2)



(c) ACK loss



(d) premature timeout/ delayed ACK

Переупорядочивание сообщений

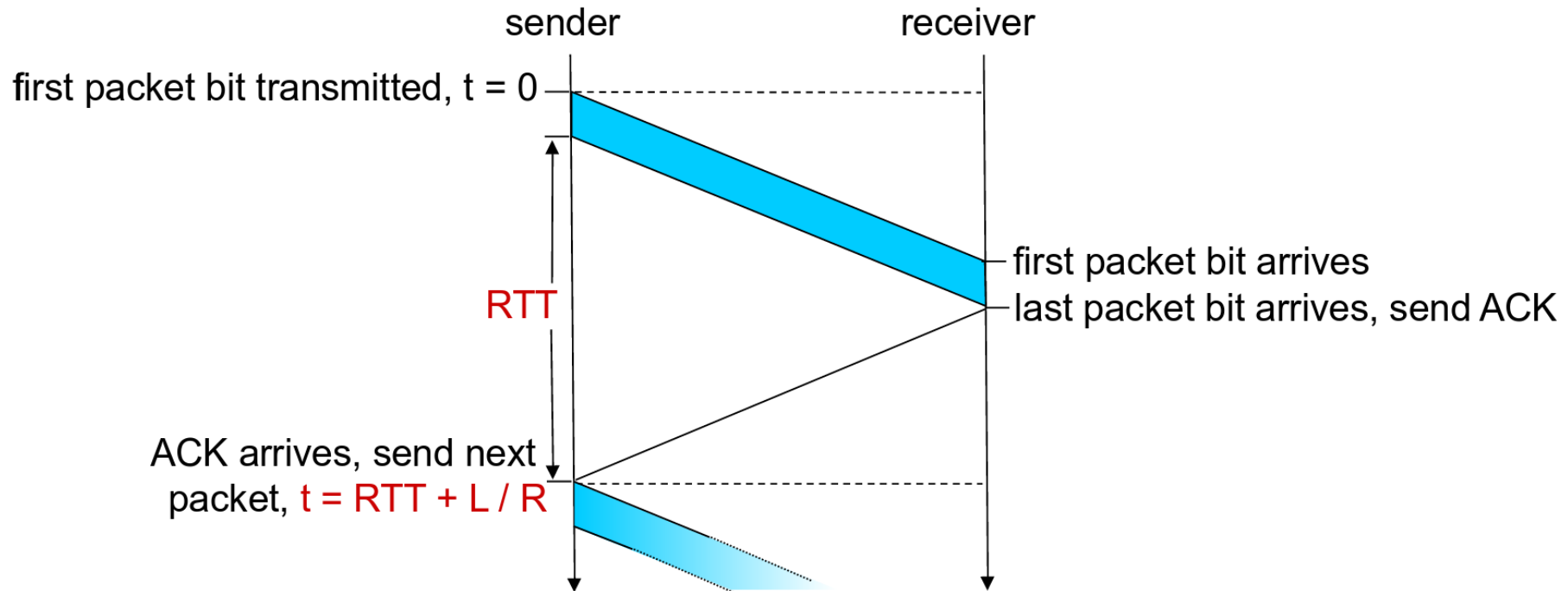
Что если канал (помимо повреждения и потерь) может также переупорядочивать сообщения в процессе передачи?

- Сломает ли это наш протокол?
- Приведите пример сценария

Возможные гарантии транспорта

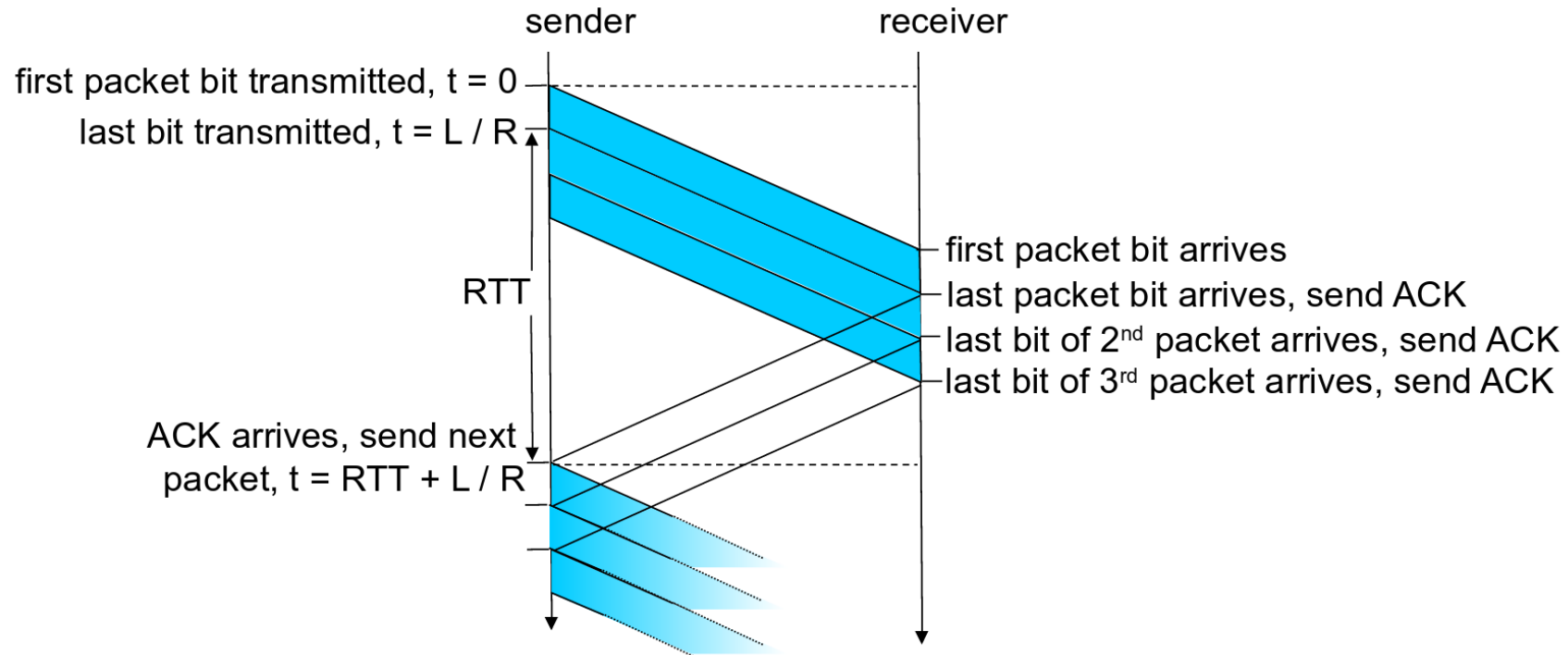
- Контроль целостности сообщений
- Доставка сообщения
 - возможно будет доставлено, возможно несколько раз (zero or more, best effort)
 - будет доставлено не более 1 раза (at most once)
 - будет доставлено как минимум 1 раз (at least once)
 - будет доставлено ровно 1 раз (exactly once)
- Порядок доставки сообщений
 - в произвольном порядке
 - в порядке их отправки

Производительность



Принцип stop-and-wait приводит к низкой утилизации сети

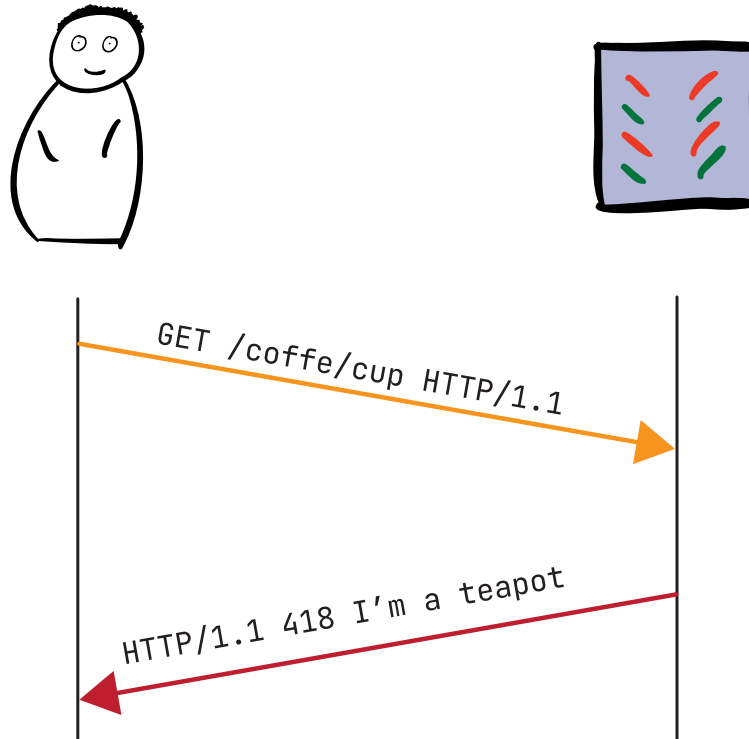
Pipelining



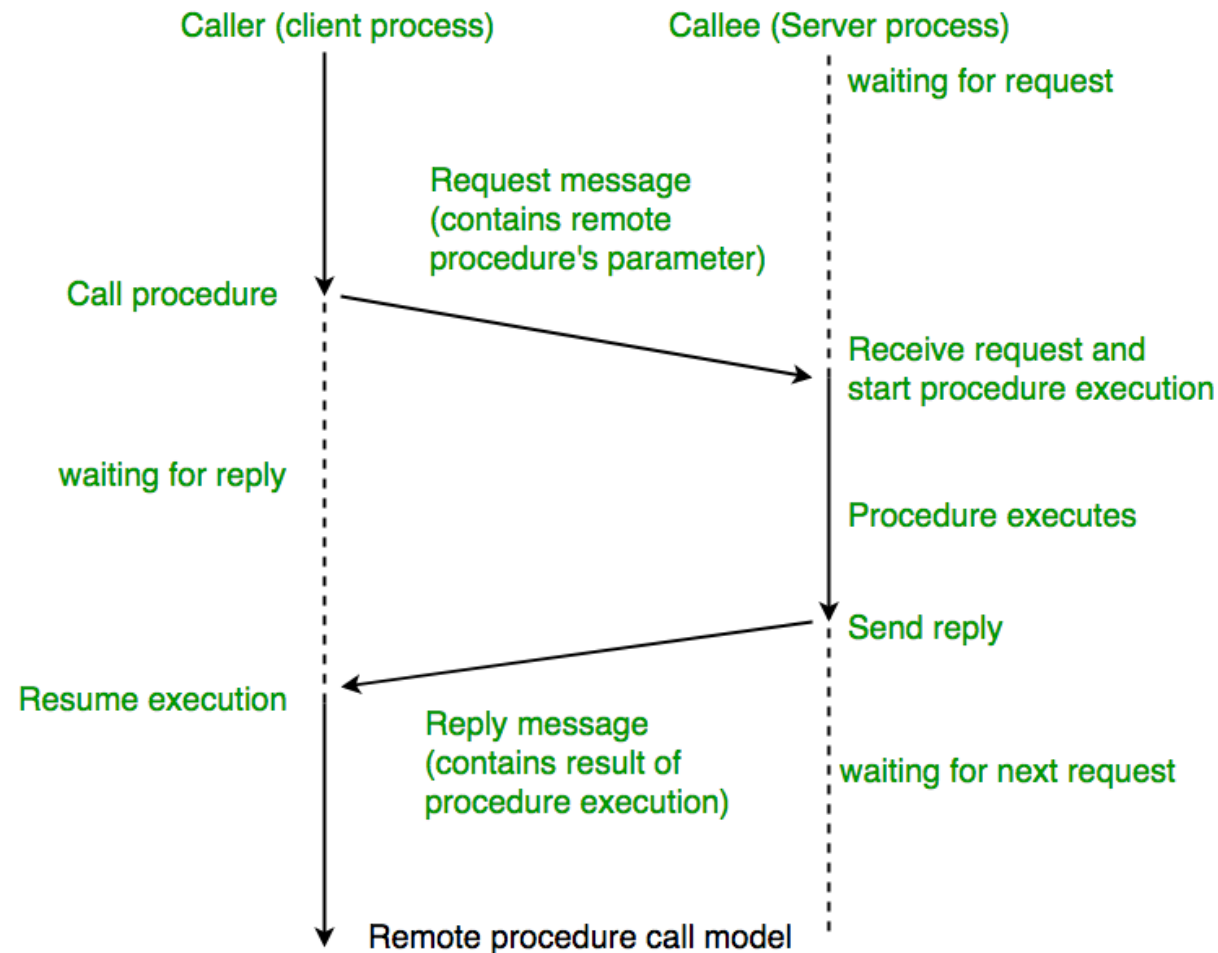
L - размер сообщения, R - пропускная способность сети, RTT - время передачи сигнала туда и обратно

Отправка N пакетов за раз увеличивает утилизацию в N раз

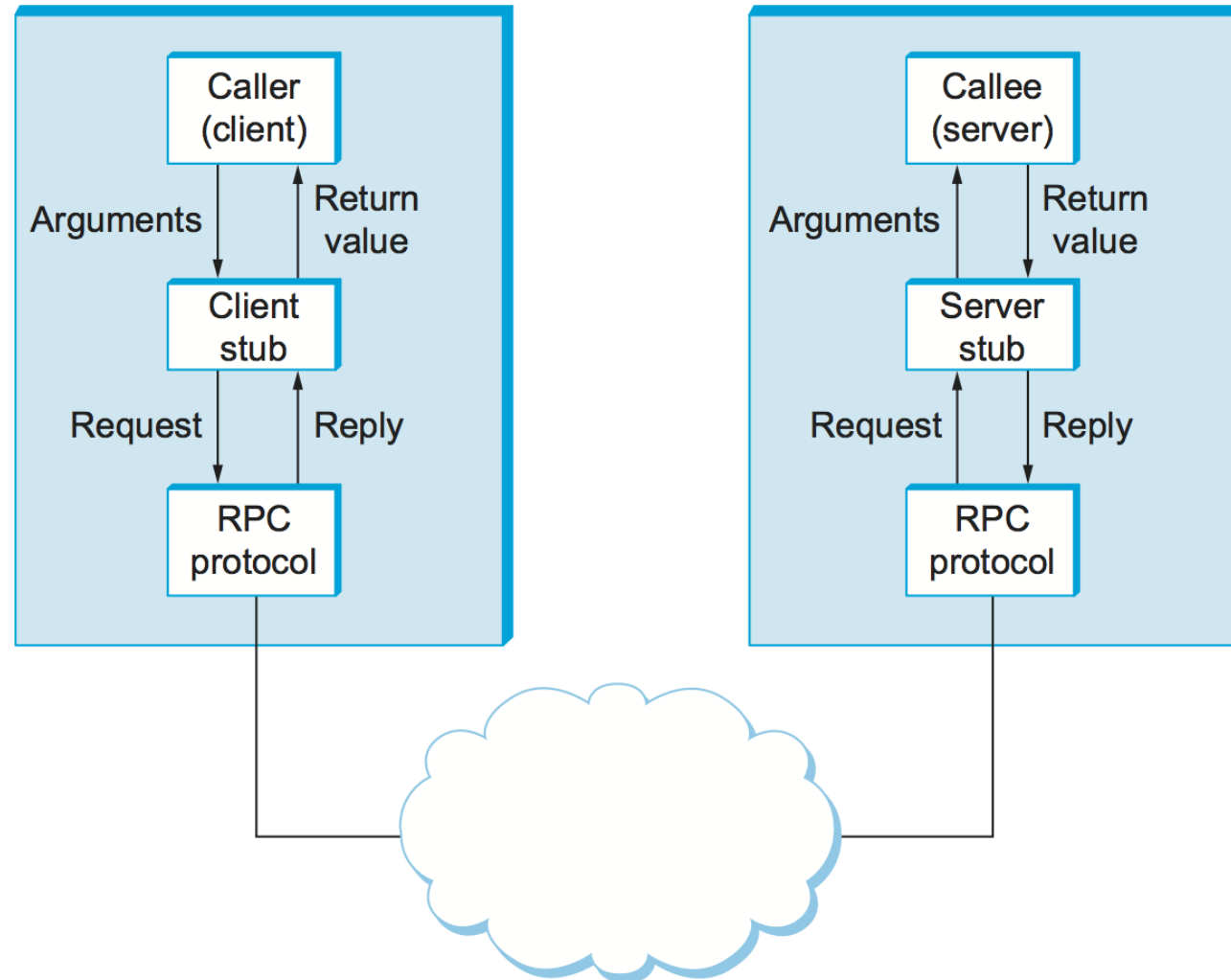
Схема взаимодействия "запрос-ответ"



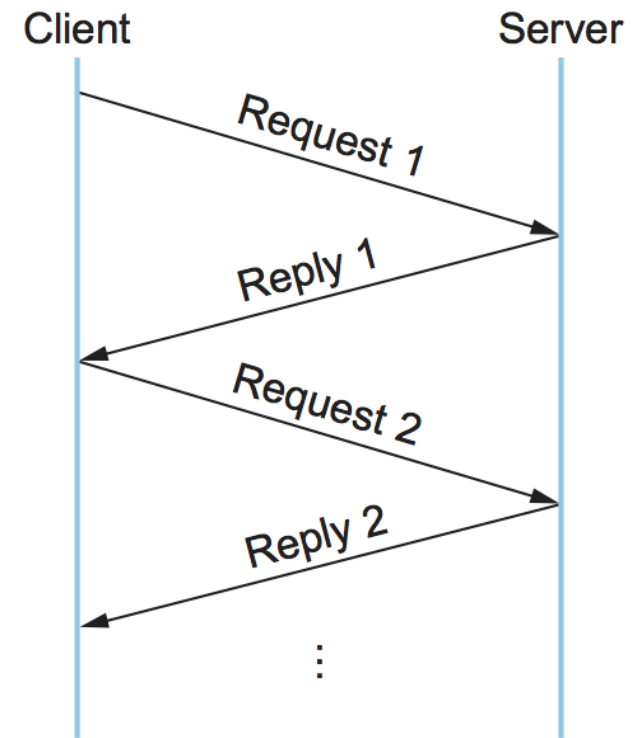
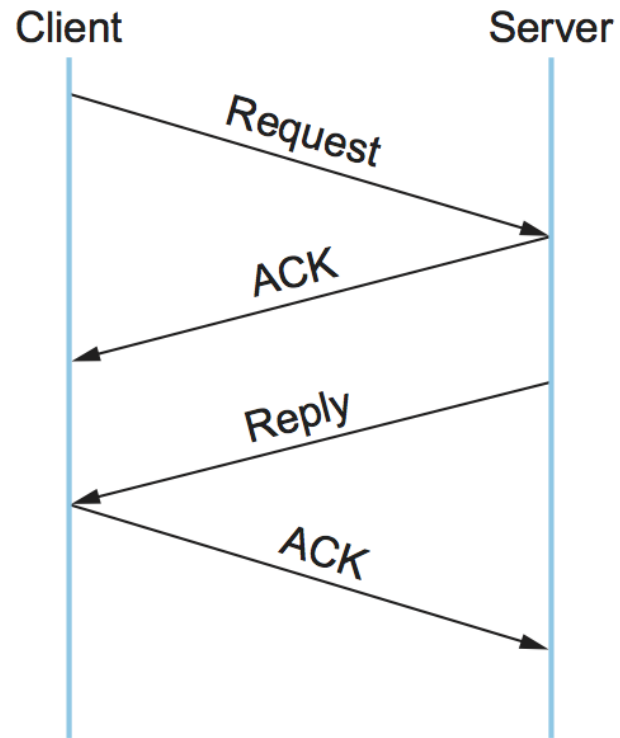
Remote Procedure Call (RPC)



Реализация RPC



Обеспечение надежности



Семантика (гарантии) RPC-вызова

- zero or more (best effort, maybe)
- at most once
- at least once
- exactly once

- ср. с семантикой локального вызова

Идемпотентные операции

- Результат операции не изменяется при многократном её применении
 - состояние сервера не изменяется или изменяется одинаковым образом
 - многократные вызовы эквивалентны однократному
- Примеры?
- В чем преимущество использования таких операций?

Отличия удаленных вызовов от локальных

- Производительность
- Выполняются удаленно, нет нагрузки на клиента
- Новые типы ошибок (исключений)
- Семантика вызова
- Передача параметров, адресное пространство
- Блокирующие вызовы и таймауты
- Что если сервер перезагрузился/обновил версию?
- Новые виды атак и угроз безопасности

Литература

- van Steen M., Tanenbaum A.S. Distributed Systems: Principles and Paradigms (разделы 4.1-4.3)
- Kurose J., Ross K. Computer Networking: A Top-Down Approach (раздел 3.4)
- Peterson L., Davie B. Computer Networks: A Systems Approach (раздел 5.3).

Дополнительно

- Birrell A.D., Nelson B.J. Implementing Remote Procedure Calls (1984)
- Tanenbaum A.S., Van Renesse R. A Critique of the Remote Procedure Call Paradigm (1988)
- Jim Waldo et al. A Note on Distributed Computing (1994)
- Henning M. Another Note on Distributed Computing (2008)
- Vinoski S. Mythbusting Remote Procedure Calls (2012)
- Meiklejohn C., McCaffrey C. A Brief History of Distributed Programming: RPC (2016)
- Kalia A. Datacenter RPCs can be General and Fast (2019)