# Глава 2
# Прикладной уровень

ЭКСМО

*Компьютерные сети: Нисходящий подход*

Эксмо, 2015

*Примечание по использованию презентаций:*

Данная презентация свободно доступна для всех (преподавателей, студентов, читателей). Вы можете просматривать слайды и использовать информацию из презентации о своему усмотрению. Взамен, авторы просят о следующем:

- При использовании слайдов (например, в аудитории) указывайте источник (чтобы другие люди использовали нашу книгу!)
- Если вы выкладываете слайды на сайт, укажите информацию об авторских правах.

Спасибо и приятного чтения!

# Глава 2: План

2.1 Принципы сетевых приложений
2.2 Всемирная паутина и HTTP
2.3 FTP
2.4 Электронная почта
- SMTP, POP3, IMAP

2.5 DNS
2.6 Одноранговые (P2P) приложения
2.7 Программирование сокетов UDP и TCP

# Глава 2: прикладной уровень

Наша цель:

- Теоретические и практические аспекты сетевых приложений
  - модели обслуживания транспортного уровня
  - клиент-серверная архитектура
  - одноранговая архитектура
- изучение протоколов прикладного уровня на примере самых популярных из них
  - HTTP
  - FTP
  - SMTP / POP3 / IMAP
  - DNS
- разработка сетевых приложений
  - API сокета

# Примеры сетевых приложений

- электронная почта
- Всемирная паутина
- текстовые сообщения
- удаленный доступ
- одноранговый файлообмен
- многопользовательские онлайн-игры
- потоковое видео (YouTube, Hulu, Netflix)
- VOIP-технология (в т.ч. Skype)
- видеоконференции в режиме реального времени
- социальные сети
- поисковые системы
- …
- …

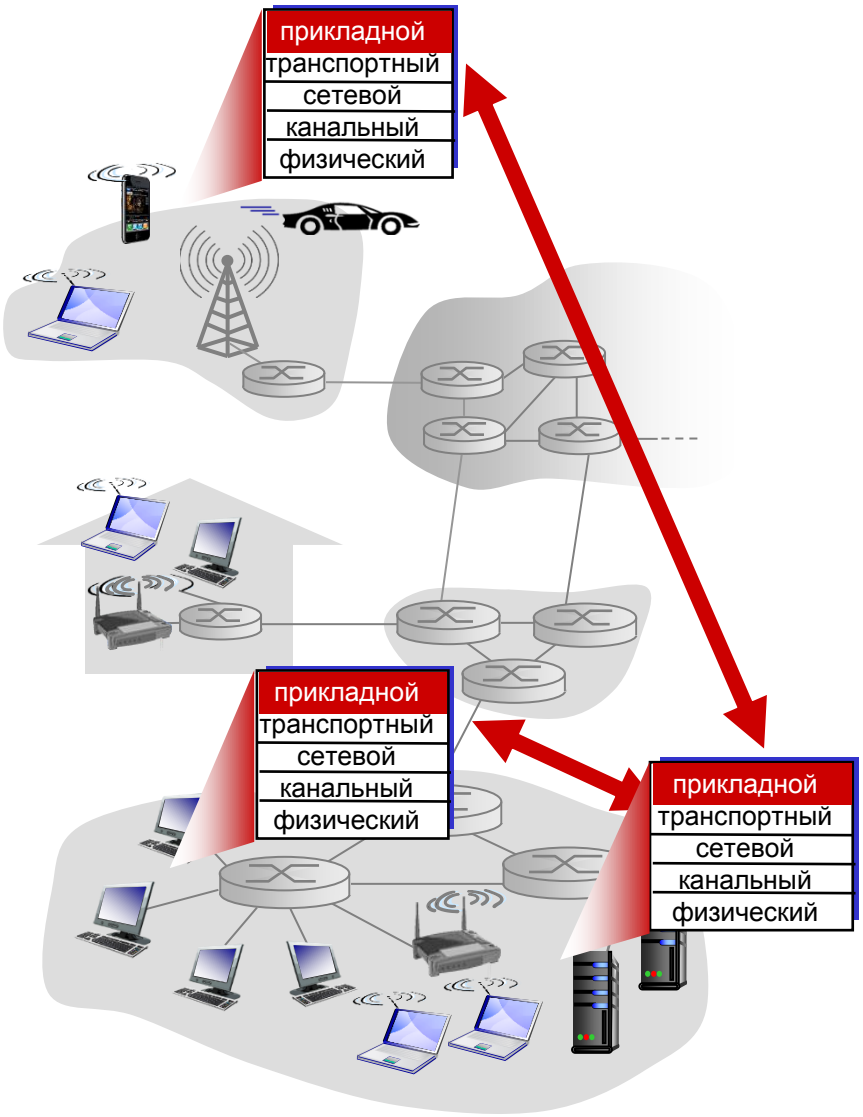
# Создание сетевого приложения

разработка программ, которые:

- запускаются на разных конечных системах
- взаимодействуют по сети
- пример взаимодействия – ПО веб-сервера и ПО браузера

нет необходимости в разработке ПО для устройств ядра сети

- устройства ядра сети не используют приложения
- как следствие, разработка и развертывание приложений на конечных системах происходит еще быстрее

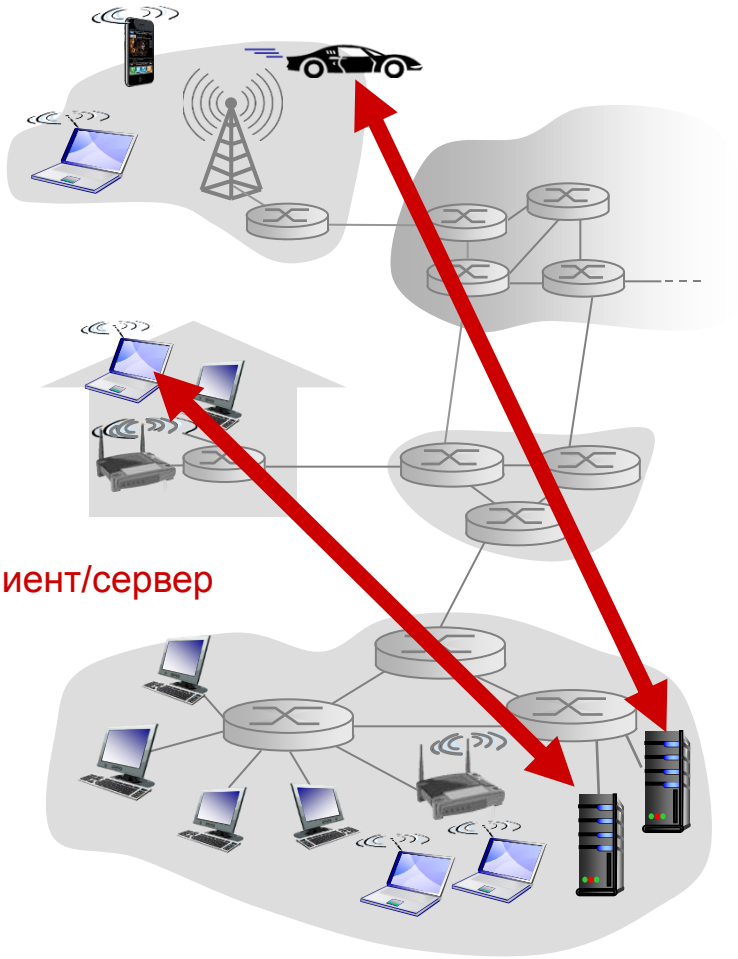


# Архитектура приложения

доступная архитектура приложений:

- клиент-серверная
- одноранговая (P2P)

# Клиент-серверная архитектура



клиент/сервер

**сервер:**

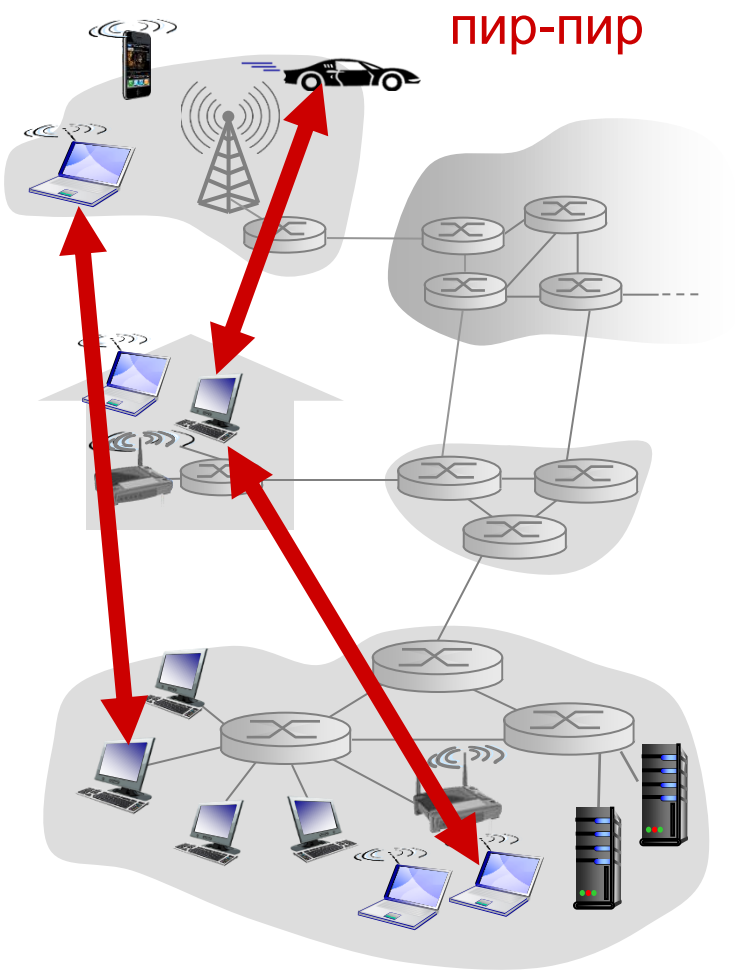
- хост всегда в режиме онлайн
- постоянный IP-адрес
- центры обработки данных с возможностью масштабирования

**клиенты:**

- взаимодействуют с сервером
- периодические соединения
- могут иметь динамические IP-адреса
- не взаимодействуют друг с другом напрямую

# Одноранговая (P2P) архитектура

- *нет* постоянного сервера в режиме онлайн
- произвольные конечные системы взаимодействуют напрямую
- пиры запрашивают услуги у других, предоставляя сами услуги другим пирам
  - *самомасштабируемость* – новые пиры увеличивают как объем, так и потребность в обслуживании
- взаимодействие периодическое и возможна смена адресов
  - сложнее управляемость



# Взаимодействие процессов

*процесс:* программа, запущенная на хосте

- межпроцессное взаимодействие между двумя процессами одного хоста (определяется операционной системой)
- процессы разных хостов взаимодействуют, обмениваясь сообщениями

клиенты, серверы

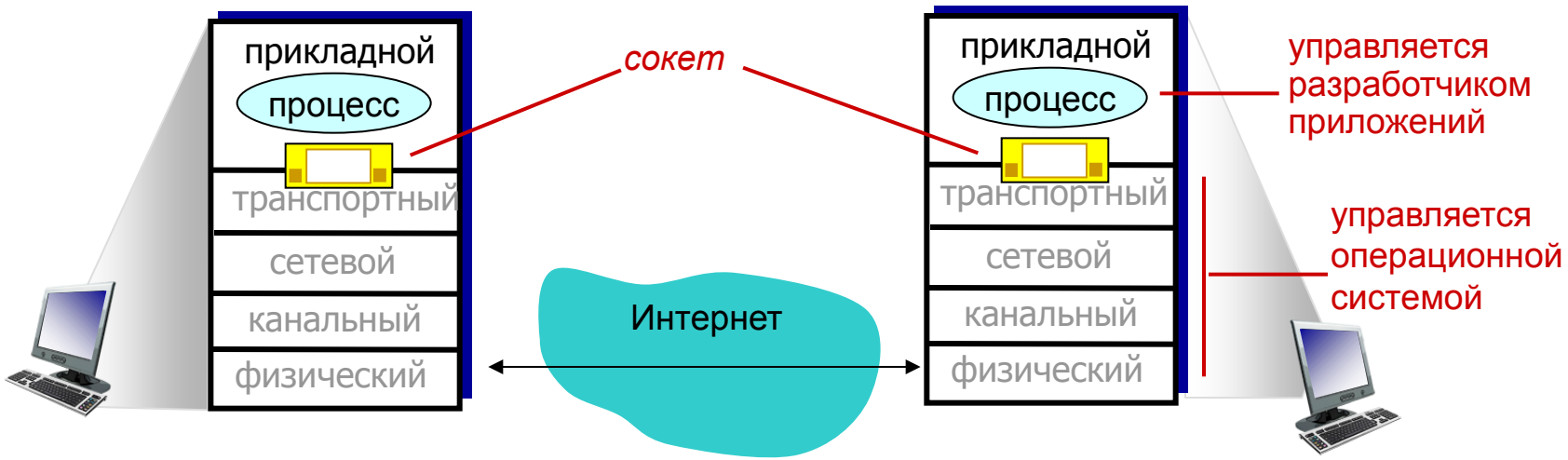
*клиентский процесс:* инициирующий взаимодействие

*серверный процесс:* ожидающий взаимодействия

- *отступление*: в приложениях, использующих одноранговую архитектуру, процесс может выступать как в качестве клиентского, так и серверного

# Сокеты

- процесс отправляет/получает сообщения через свой сокет
- сокет – аналог «двери»
  - передающий процесс «проталкивает» сообщение в «дверь»
  - передающий процесс полагается на транспортную инфраструктуру по ту сторону «двери», доставляющую сообщения в сокет принимающего процесса



# Адресация процессов

- для получения сообщений процесс должен иметь *идентификатор*
- уникальный 32-битный адрес – у каждого хоста
- *В:* достаточно ли IP-адреса хоста для идентификации процесса?
  - О: нет, на одном хосте может быть запущено *много* процессов
- *идентификатор* включает как IP-адрес, так и номера портов, связанные с процессом
- примеры номеров портов:
  - HTTP-сервер: 80
  - почтовый сервер: 25
- для отправки сообщения веб-серверу gaia.cs.umass.edu:
  - IP-адрес: 128.119.245.12
  - номер порта: 80
- подробнее далее…

# Протокол прикладного уровня

определяет:

- типы передаваемых сообщений:
  - запрос, ответ
- синтаксис сообщения:
  - содержание и формат полей в сообщении
- семантика сообщения:
  - значение информации в полях
- правила: определяющие когда и как процессы передают и получают сообщения

открытые протоколы:

- описываются документами RFC
- открыты для взаимодействия
- примеры: HTTP, SMTP

проприетарные протоколы:

- пример: Skype

# Транспортные службы, требуемые приложению

## целостность данных

- некоторые приложения (передача файлов, веб-операции) требуют 100% надежности в передаче данных
- другие приложения(аудио) допускают некоторые потери

## время доставки

- для эффективной работы некоторых приложений (Интернет-телефония, онлайн-игры) требуется низкий уровень задержек

## пропускная способность

- определенный минимальный порог требуется для работы некоторых приложений (мультимедиа)
- другие приложения (эластичные) работают, используя любой канал, не предъявляя особых требований к полосе пропускания

## безопасность

- шифрование, целостность данных, …

# Требования к транспортным службам, предъявляемые сетевыми приложениями

| приложение | потери данных | пропускная способность | чувствительность к времени доставки |
|---|---|---|---|
| передача файлов | не допускаются | эластичное | нет |
| эл. почта | не допускаются | эластичное | нет |
| веб-приложения | не допускаются | эластичное | нет |
| аудио/видео реального времени | допустимы | аудио: 5 Кбит/с – 1 Мбит/с<br>видео:10 Кбит/с – 5 Мбит/с | да, сотни миллисекунд |
| сохраненное аудио/видео | допустимы | как предыдущее | да, несколько секунд |
| интерактивные онлайн-игры | допустимы | от нескольких Кбит/с | да, сотни миллисекунд |
| текстовые сообщения | не допускаются | эластичное | да и нет |

# Транспортные службы Интернета

*службы TCP:*

- *надежная доставка* от передающего процесса к принимающему
- *контроль потока:* отправитель не «завалит» получателя сообщениями
- *контроль перегрузки:* принудительное снижение скорости передачи при перегрузке сети
- *не обеспечивается:* безопасность, гарантированное время доставки и пропускная способность
- *с установлением соединения:* логическое соединение между клиентским и серверным процессами

*службы UDP:*

- *ненадежная передача данных* между процессами
- *не обеспечивается:* надежность, контроль потока, контроль перегрузки, гарантированное время доставки, пропускная способность, безопасность, установление соединения,

# Интернет-приложения и используемые протоколы

| приложение | протокол прикладного уровня | базовый транспортный протокол |
|---|---|---|
| электронная почта | SMTP [RFC 2821] | TCP |
| удаленный доступ | Telnet [RFC 854] | TCP |
| Всемирная паутина | HTTP [RFC 2616] | TCP |
| передача файлов | FTP [RFC 959] | TCP |
| потоковые мультимедиа | HTTP (например, YouTube), | TCP или UDP |
| Интернет-телефония | RTP [RFC 1889] SIP, RTP, частный (например, Skype) | TCP или UDP |

# Безопасность протокола TCP

TCP и UDP

- без шифрования
- текстовые пароли передаются в сокет открытым текстом

SSL

- зашифрованное TCP-соединение
- целостность данных
- конечная аутентификация

SSL на прикладном уровне

- приложения используют SSL для общения с TCP

API-сокет SSL

- текстовые пароли передаются в сокет зашифрованными
- См. главу 7

# Глава 2: План

2.1 Принципы сетевых приложений
- архитектура
- требования

2.2 Всемирная паутина и HTTP

2.3 FTP

2.4 Электронная почта
- SMTP, POP3, IMAP

2.5 DNS

2.6 Одноранговые (P2P) приложения

2.7 Программирование сокетов UDP и TCP

# Всемирная паутина и HTTP

*Сначала общий обзор…*

- *Веб-страница состоит из объектов*
- Объектом может служить файл HTML, изображение JPEG, Java-апплет, аудиофайл,…
- Веб-страница включает *основной HTML-файл,* в котором присутствует *несколько объектов, на которые он ссылается*
- У каждого объекта есть свой *URL-адрес*

`www.someschool.edu/someDept/pic.gif`

Имя хоста | Путь к файлу

# Обзор HTTP

HTTP: протокол передачи гипертекста

- протокол прикладного уровня для веб-приложений
- клиент-серверная модель
  - *клиент:* браузер запрашивает, получает и отображает (используя протокол HTTP) веб-объекты
  - *сервер:* веб-сервер отправляет объекты (используя протокол HTTP) в ответ на запросы

# Обзор HTTP

*использует TCP:*

- клиент инициирует TCP-соединение (создает сокет) с сервером на порту 80
- сервер принимает TCP-соединение от клиента
- Браузер (HTTP-клиент) и веб-сервер (HTTP-сервер) обмениваются сообщениями прикладного уровня
- TCP-соединение закрывается

*HTTP – протокол «без сохранения состояния»*

- сервер не хранит информацию о последних запросах

*примечание*

протоколы с сохранением состояния достаточно сложны!

- должна храниться история состояний сеансов
- состояния сеансов должны согласовываться на случай сбоев на сервере или клиенте

# HTTP-соединения

*непостоянные*

- один объект - через одно соединение
  - соединение затем закрывается
- загрузка нескольких объектов требует нескольких соединений

*постоянные*

- через одно соединение клиент-сервер можно отправить много объектов

# Непостоянное HTTP-соединение

Предположим, пользователь набирает в адресной строке:

**`www.someSchool.edu/someDepartment/home.index`** *(содержит текст, ссылки на 10 jpeg-изображений)*

1а. HTTP-клиент инициирует TCP-соединение с HTTP-сервером (с его процессом) хоста www.someSchool.edu на порту 80

1б. HTTP-сервер хоста www.someSchool.edu, ожидающий TCP-соединения на порту 80, "принимает" соединение, уведомляя клиента

2. HTTP-клиент отправляет *сообщение-запрос* (содержащее URL-адрес) в сокет соединения TCP. Сообщение указывает, что клиент запрашивает объект someDepartment/home.index

3. HTTP-сервер получает сообщение-запрос, формирует *ответное сообщение*, содержащее запрашиваемый объект, и отправляет сообщение в свой сокет

время

# Непостоянное HTTP-соединение

4. HTTP-сервер закрывает TCP-соединение

5. HTTP-клиент получает ответное сообщение, содержащее html-файл, и отображает его. Обрабатывая файл, находит ссылки на 10 jpeg-объектов

время

6. Шаги 1-5 повторяются для каждого из 10 jpeg-объектов

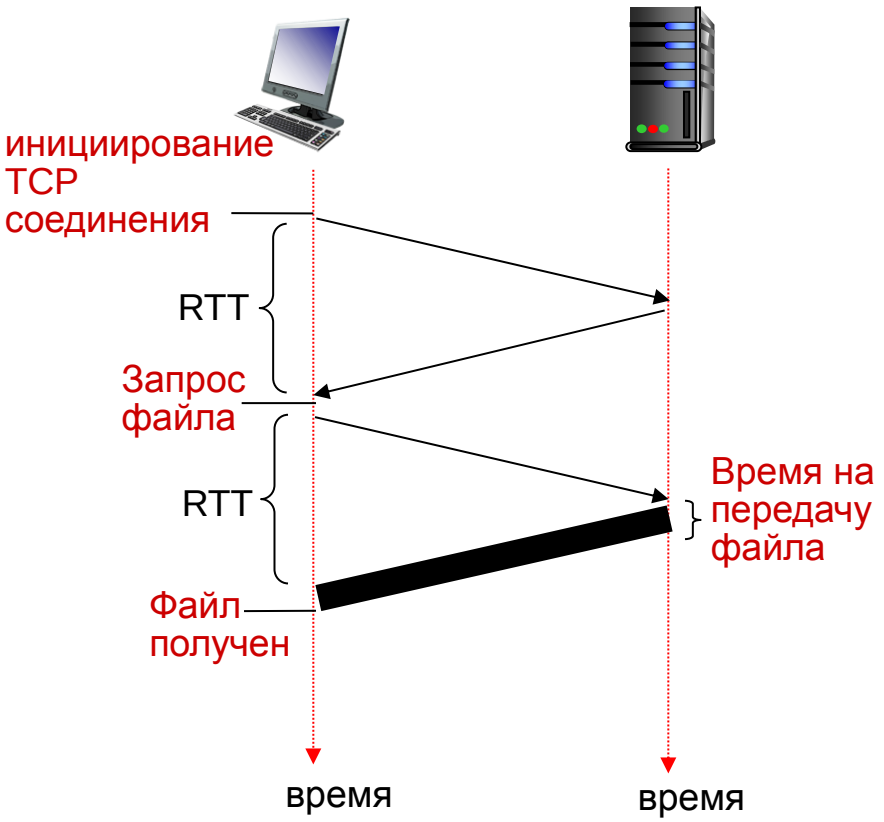
# Непостоянное HTTP-соединение: время ответа

RTT (определение): время прохождения пути от клиента до сервера и обратно для небольшого пакета

Время ответа HTTP :

- один RTT на инициирование TCP-соединения
- один RTT на HTTP-запрос и первые несколько байт HTTP-ответа
- время на передачу файла
- время ответа для непостоянного HTTP-соединения=

  2RTT+ время на передачу файла



# Постоянное HTTP-соединение

*Непостоянное HTTP-соединение :*

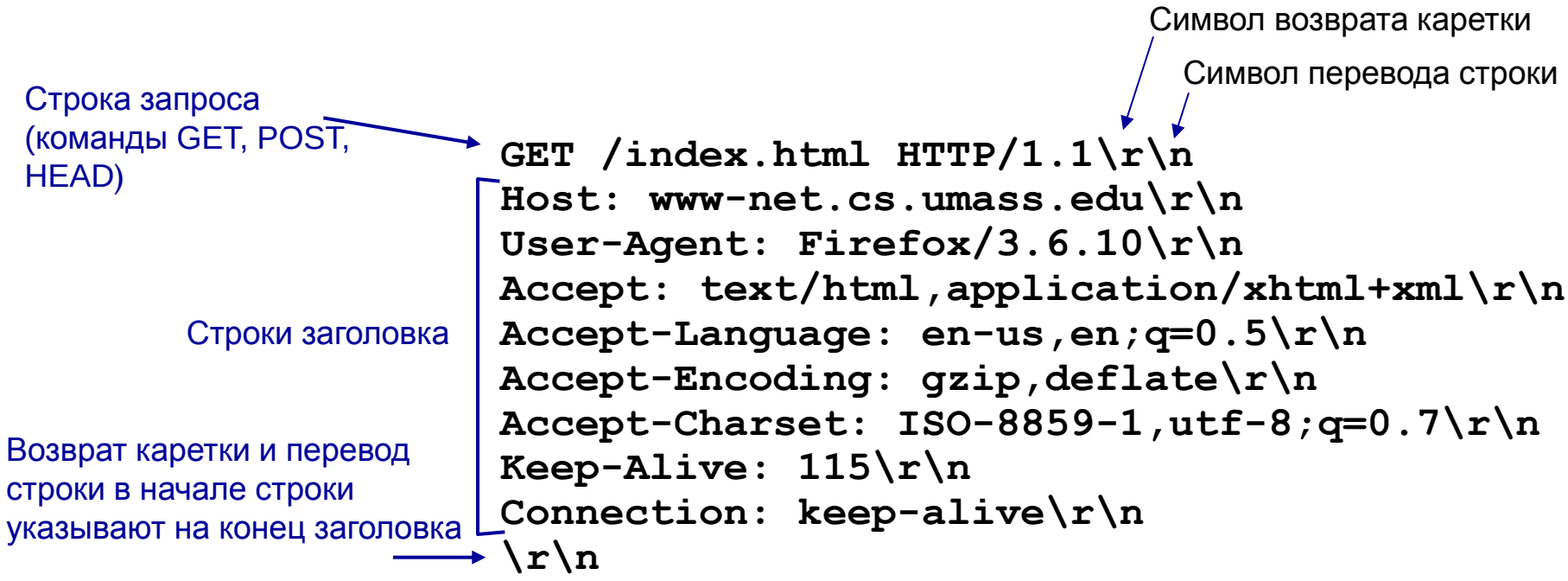
- требуется 2 RTT на 1 объект
- нагрузка на ОС для каждого нового соединения
- браузеры часто открывают параллельные TCP-соединения для извлечения объектов по ссылкам

*Постоянное HTTP-соединение:*

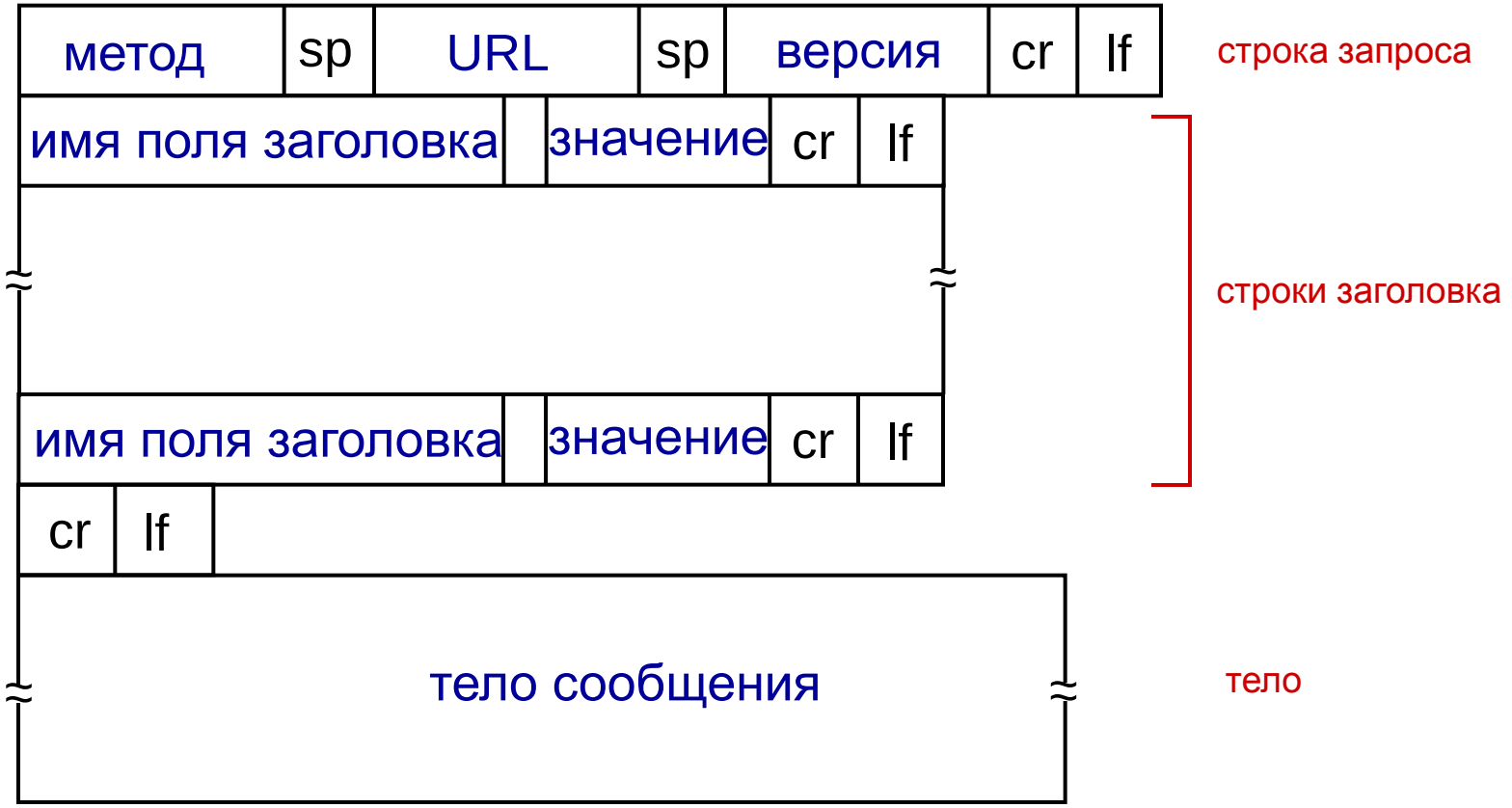
- после отправки ответа сервер оставляет соединение открытым
- через открытое соединение можно отправить последовательность HTTP-сообщений между клиентом и сервером
- клиент отправляет запрос как только встретит содержащий ссылку объект
- всего 1 RTT для каждого из объектов

# Сообщение-запрос HTTP

- Два вида HTTP-сообщений: *запрос, ответ*
- Сообщение-запрос HTTP:
  - ASCII (воспринимаемый человеком формат)



# Основной формат HTTP-запроса



# Ввод данных через формы

метод POST:

- веб-страница часто содержит формы ввода
- введенные данные загружаются на сервер в тело сообщения

метод URL:

- использует метод GET
- введенные данные загружаются в поле URL строки запроса:

```
www.somesite.com/animalsearch?monkeys&banana
```

# Основные методы

HTTP/1.0:

- GET
- POST
- HEAD
  - запрос, при котором тело сообщения в ответе остается пустым

HTTP/1.1:

- GET, POST, HEAD
- PUT
  - выгружает файл из тела сообщения в путь, указанный в поле URL
- DELETE
  - удаляет файл, указанный в поле URL

# Ответное сообщение HTTP

строка состояния
(код состояния протокола и
сообщение состояния)

строки заголовка

данные, например
запрошенный
HTML-файл

```
HTTP/1.1 200 OK\r\n
Date: Sun, 26 Sep 2010 20:09:20 GMT\r\n
Server: Apache/2.0.52 (CentOS)\r\n
Last-Modified: Tue, 30 Oct 2007 17:00:02
   GMT\r\n
ETag: "17dc6-a5c-bf716880"\r\n
Accept-Ranges: bytes\r\n
Content-Length: 2652\r\n
Keep-Alive: timeout=10, max=100\r\n
Connection: Keep-Alive\r\n
Content-Type: text/html; charset=ISO-8859-
   1\r\n
\r\n
данные данные данные данные...
```

# Коды состояния HTTP-ответа

- появляются в первой строке ответа от сервера
- некоторые примеры:

**200 OK**

- запрос выполнен успешно , запрошенный объект следует дальше в этом же сообщении

**301 Moved Permanently**

- запрошенный объект окончательно перемещен, его новое место расположения указано далее в сообщении (Location:)

**400 Bad Request**

- сервер не понял запрос (скорее всего, синтаксическая ошибка)

**404 Not Found**

- запрошенный документ на сервере не найден

**505 HTTP Version Not Supported**

- сервер не поддерживает указанную в запросе версию протокола

# Проверка работы HTTP (клиентская часть)

1. С помощью Telnet подключитесь к веб-серверу:

```
telnet cis.poly.edu 80
```

открывает TCP-соединение по порту 80 (порт по умолчанию для HTTP) с сервером cis.poly.edu. все, что набрано, отсылается на его порт 80

2. введите GET-запрос:

```
GET /~ross/ HTTP/1.1
Host: cis.poly.edu
```

введя это (нажмите дважды Enter), вы отправляете минимальный (но готовый) GET-запрос HTTP -серверу

3. посмотрите на ответное сообщение от HTTP-сервера

(или используйте ПО Wireshark для анализа перехваченных запросов/ответов HTTP)

# Состояние сеанса пользователь-сервер: cookie

многие веб-сайты используют механизм cookie

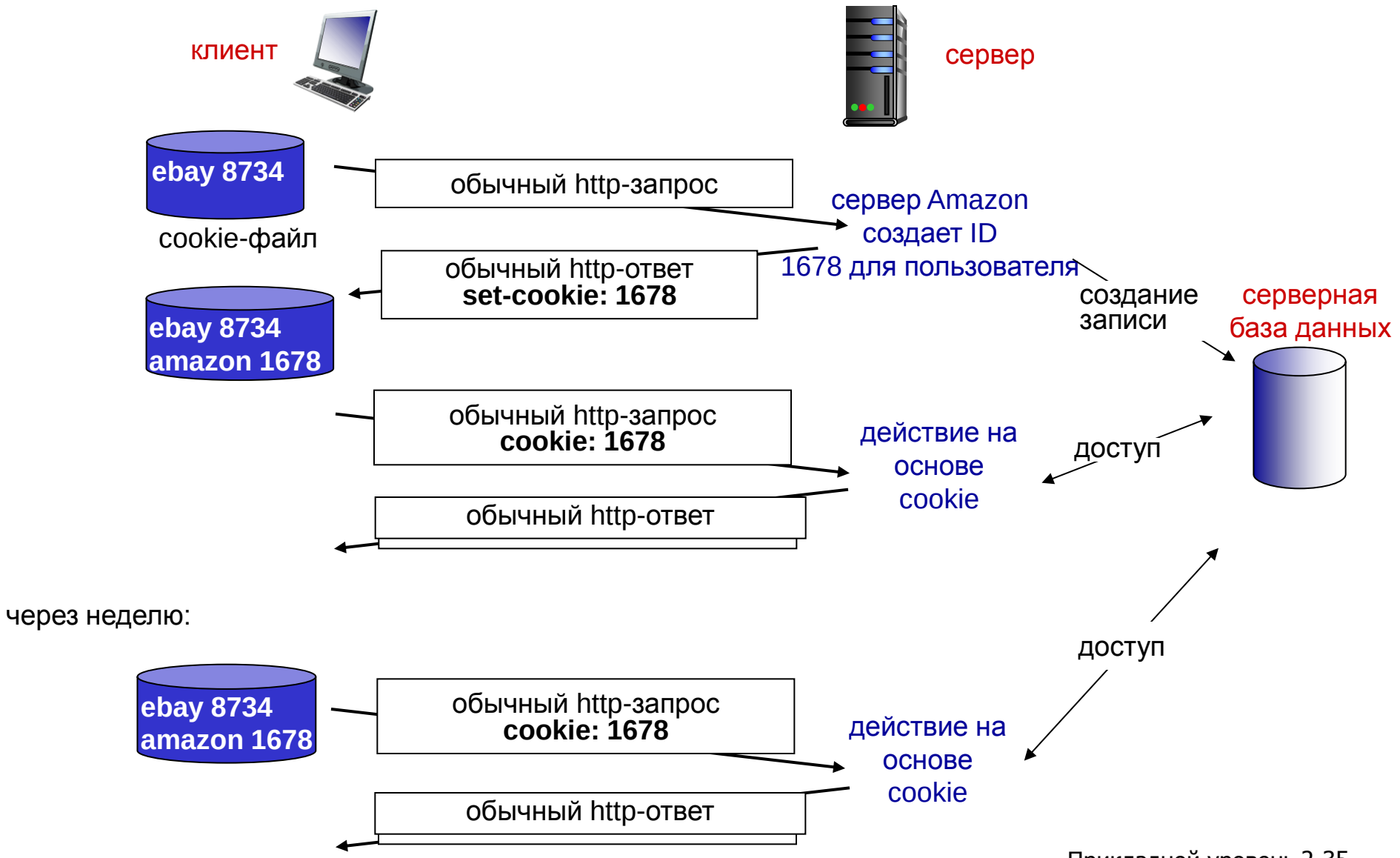
*четыре компонента:*

1) строка заголовка в HTTP-*ответе*
2) строка заголовка в HTTP-*запросе*
3) cookie-файл на хосте пользователя, управляемый его браузером
4) база данных на стороне веб-сервера

пример:

- Светлана выходит в Интернет со своего домашнего ПК
- первый раз заходит на сайт электронных покупок
- когда начальный HTTP-запрос поступает на сайт, веб-сайт создает:
  - уникальный ID
  - запись в базе данных сервера, соответствующую этому ID

# Состояние сеанса пользователь-сервер: cookie



# Состояние сеанса пользователь-сервер: cookie

*для чего используются cookie-файлы:*

- авторизация
- карты покупок
- хранение настроек и предпочтений
- сохранение состояния сеанса пользователя (веб-почта)

отступление

*Cookie и конфиденциальность:*

- Cookie позволяют сайтам получать информацию о пользователе: адрес эл. почты, имя и т.д.

*как сохранять состояние:*

- cookie: http-сообщения несут в себе информацию о состоянии сеанса отправитель-получатель

# Веб-кэш (прокси-сервер)

*цель:* удовлетворять запросы клиента без участия исходного веб-сервера

- пользователь настраивает браузер: веб-доступ через прокси
- браузер отправляет запрос прокси-серверу
  - объект в кэше: прокси-сервер возвращает объект
  - в противном случае прокси-сервер запрашивает объект у исходного сервера и возвращает его клиенту

# Веб-кэш (прокси-сервер)

- кэш выполняет роль клиента и сервера
  - сервер для клиента, выполняющего запрос
  - клиент для исходного сервера
- часто кэш устанавливается Интернет-провайдером

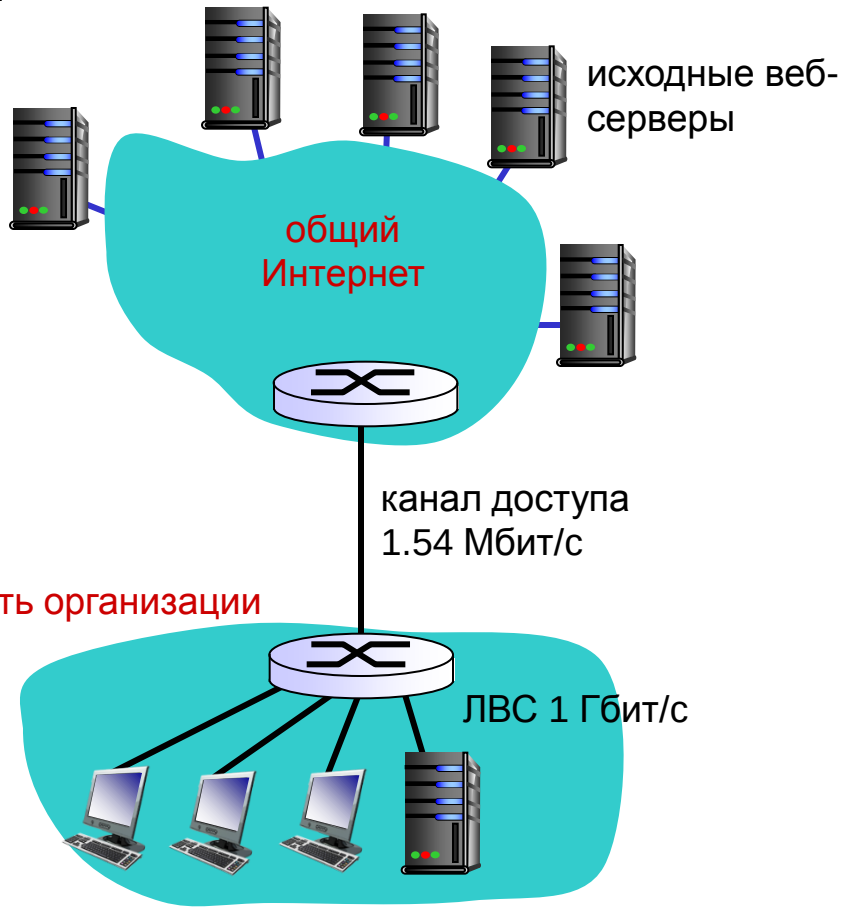
*зачем нужно веб-кэширование?*

- уменьшить время отклика на запросы клиентов
- уменьшить трафик в сети организации
- позволяет провайдерам эффективнее доставлять контент (как и одноранговый файлообмен)

# Пример кэширования

*предположения:*

- средний размер объекта: 100 Кбит
- средняя частота запросов от браузеров к серверам: 15/с
- средняя скорость передачи данных к браузерам: 1,50 Мбит/с
- RTT от маршрутизатора организации до любого веб-сервера: 2 с
- пропускная способность канала: 1.54 Мбит/с




*результаты:*

- загрузка ЛВС: 15%
- загрузка канала доступа= 99%
- общая задержка = задержка Интернета + задержка доступа + задержка ЛВС

*проблема!*

= 2 с + минуты + микросекунды

# Пример кэширования: расширение канала доступа

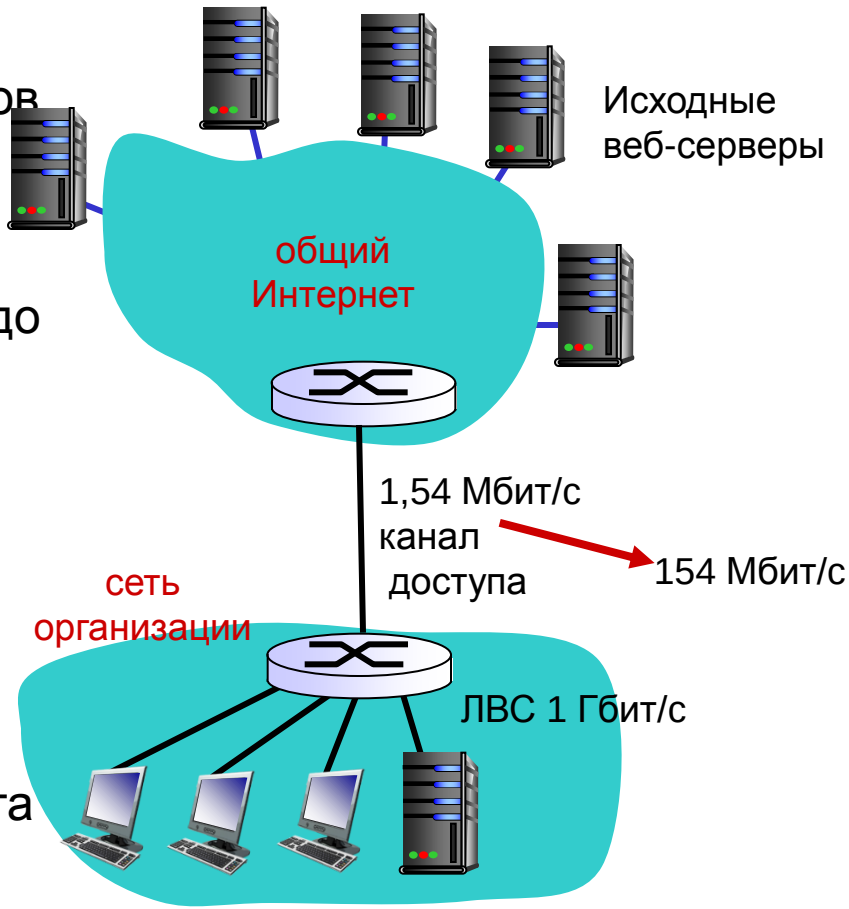
*предположения:*

- средний размер объекта: 100 Кбит
- средняя частота запросов от браузеров к серверам: 15/с
- средняя скорость передачи данных к браузерам: 1,50 Мбит/с
- RTT от маршрутизатора организации до любого веб-сервера: 2 с
- пропускная способность канала:
- 1,54 Мбит/с → 154 Мбит/с

*результаты:*

- загрузка ЛВС :15%
- загрузка канала
- доступа = 99% → 9.9%
- общая задержка = задержка Интернета + задержка доступа + задержка ЛВС

= 2 с + минуты + микросекунды → миллисекунды




*затраты:* увеличенная скорость канала (недешево!)

# Пример кэширования: установка локального прокси

*предположения:*

- средний размер объекта: 100 Кбит
- средняя частота запросов от браузеров к серверам:15/с
- средняя скорость передачи данных к браузерам: 1,50 Мбит/с
- RTT от маршрутизатора организации до любого веб-сервера: 2 с
- пропускная способность канала: 1,54 Мбит/с

*результаты:*

- загрузка ЛВС : 15%
- загрузка канала доступа = ?
- общая задержка = ?

*Как вычислить загрузку канала и задержку?*


*Затраты:* прокси-сервер (недорого!)

# Пример кэширования: установка локального прокси

*Вычисляем загрузку канала и задержку в случае с прокси:*

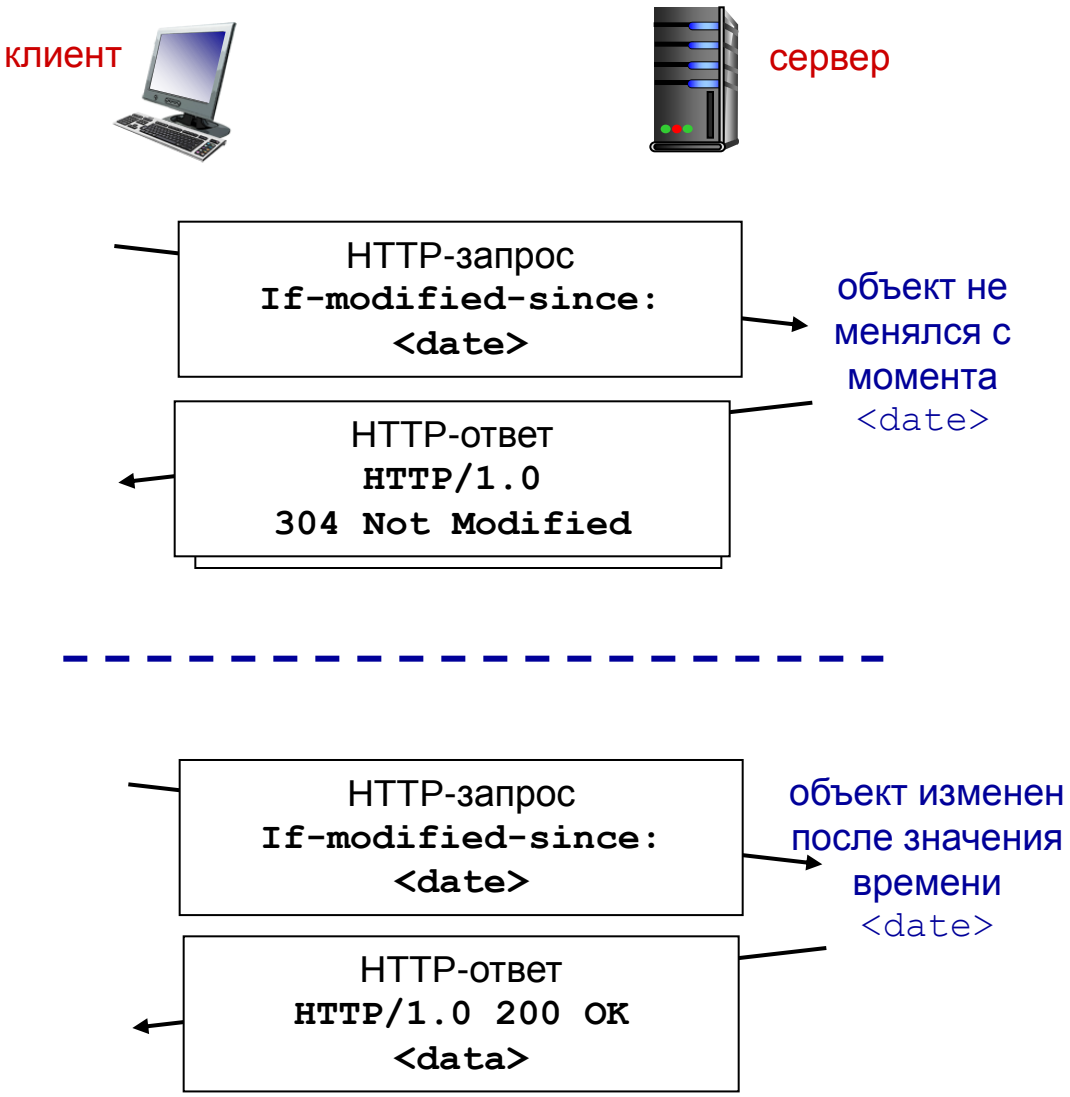
- предполагаем, что коэффициент попадания в кэш 0,4
  - 40% запросов обрабатываются прокси-сервером, 60% - исходными веб-серверами
    - **загрузка канала доступа:**
      - 60% запросов используют канал
    - скорость данных через канал к браузерам= 0,6 × 1,50 Мбит/с = 0,9 Мбит/с
      - загрузка = 0,9/1,54 = 0,58
    - **общая задержка**
      - = 0,6 × (задержка от исходных серверов) + 0,4 × (задержка от прокси)
      - = 0,6 (2,01) + 0,4 (~миллисекунды)
      - = ~ 1,2 секунды
      - меньше, чем при канале в 154 Мбит/с (и к тому же дешевле!)

# Условный GET-запрос

- *Цель:* не отправлять объект, если в кэше существует его актуальная версия
  - нет задержек передачи
  - ниже загрузка канала
- *cache:* указывается дата кэшированной копии

  **`If-modified-since: <date>`**
- *server:* ответ не содержит объект, если копия в кэше актуальна:

  **`HTTP/1.0 304 Not Modified`**



# Глава 2: План

2.1 Принципы сетевых приложений
- архитектура
- требования

2.2 Всемирная паутина и HTTP

2.3 FTP

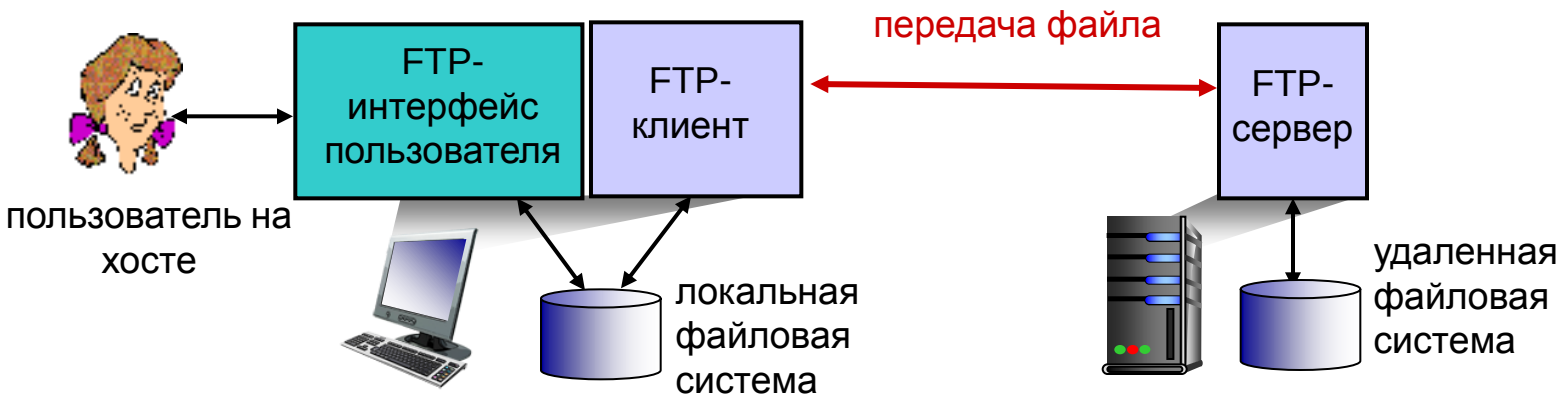
2.4 Электронная почта
- SMTP, POP3, IMAP

2.5 DNS

2.6 Одноранговые (P2P) приложения

2.7 Программирование сокетов UDP и TCP

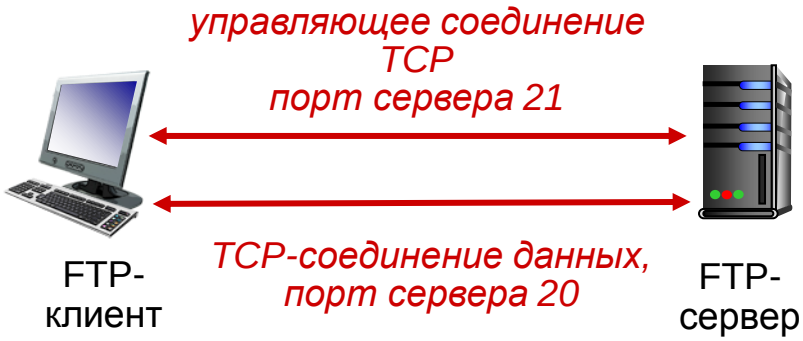
# FTP: протокол передачи файлов




- передача файла на удаленный хост или с него
- клиент-серверная модель
  - *клиент:* сторона, инициирующая передачу (на/с удаленного хоста)
  - *сервер:* удаленный хост
- ftp: документ RFC 959
- ftp-сервер: порт 21

# FTP: отдельные соединения для передачи данных и управления

- FTP-клиент обращается к FTP-серверу по порту через TCP
- авторизация через управляющее соединение
- отправка команд и просмотр удаленного каталога через управляющее соединение
- при получении команды на передачу *сервер* открывает соединение данных для файла
- после передачи одного файла сервер закрывает соединение данных




- для передачи следующего файла сервер открывает еще одно соединение данных
- управляющее соединение: *«вне полосы»*
- FTP-сервер отслеживает состояние соединения: текущий каталог, предыдущая аутентификация

# FTP-команды и ответы

***пример команд:***

- отправляются в ASCII формате через управляющее соединение
- **USER *имя пользователя***
- **PASS *пароль***
- **LIST** возвращает список файлов текущего каталога
- **RETR *имя файла*** вызов (получение) файла
- **STOR *имя файла*** сохранение (отправка) файла на удаленный хост

***пример кодов ответа***

- код состояния и сообщение (как в HTTP)
- **331 Username OK, password required**

  (Имя пользователя корректно, нужен пароль)
- **125 data connection already open; transfer starting**

  (Соединение открыто, обмен данными начат)
- **425 Can't open data connection** (Невозможно открыть соединение)
- **452 Error writing file** (Ошибка при записи файла)

# Глава 2: План

2.1 Принципы сетевых приложений
- архитектура
- требования

2.2 Всемирная паутина и HTTP
2.3 FTP
2.4 Электронная почта
2.5 DNS
2.6 Одноранговые (P2P) приложения
2.7 Программирование сокетов UDP и TCP

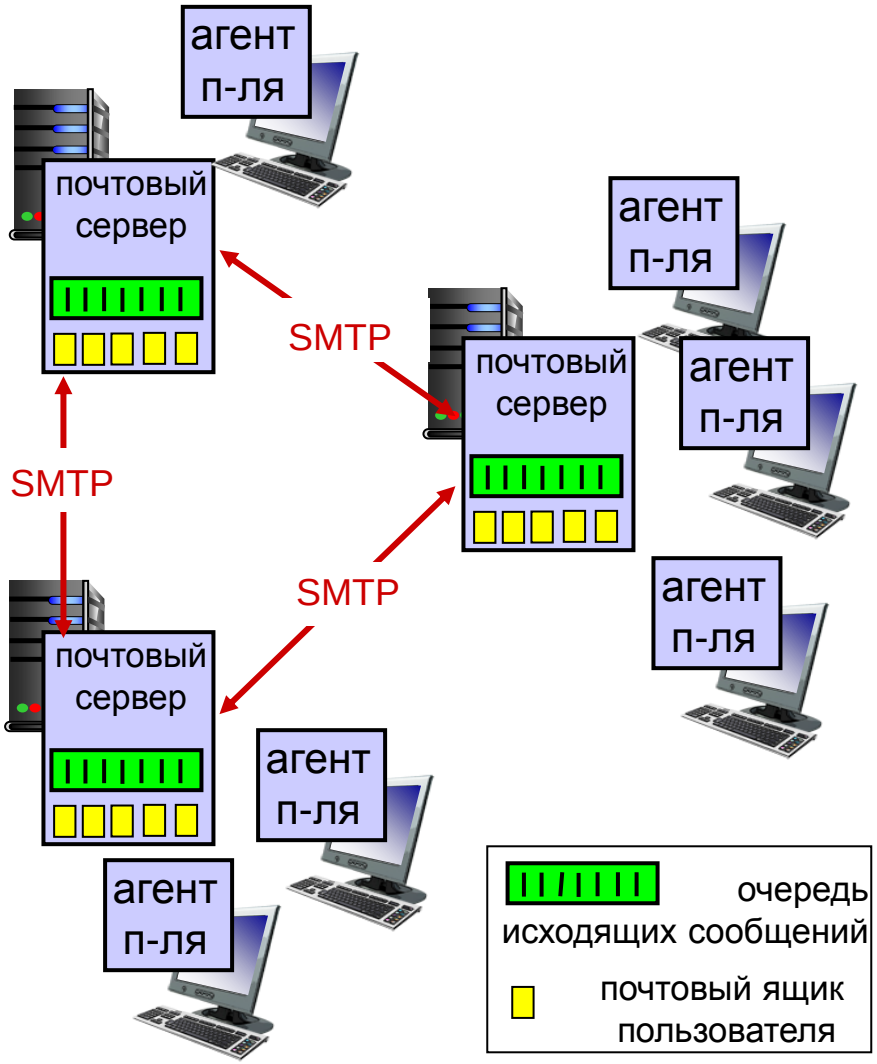
# Электронная почта

*Три основных компонента:*

- агенты пользователя
- почтовые серверы
- протокол SMTP

*Агенты пользователя*

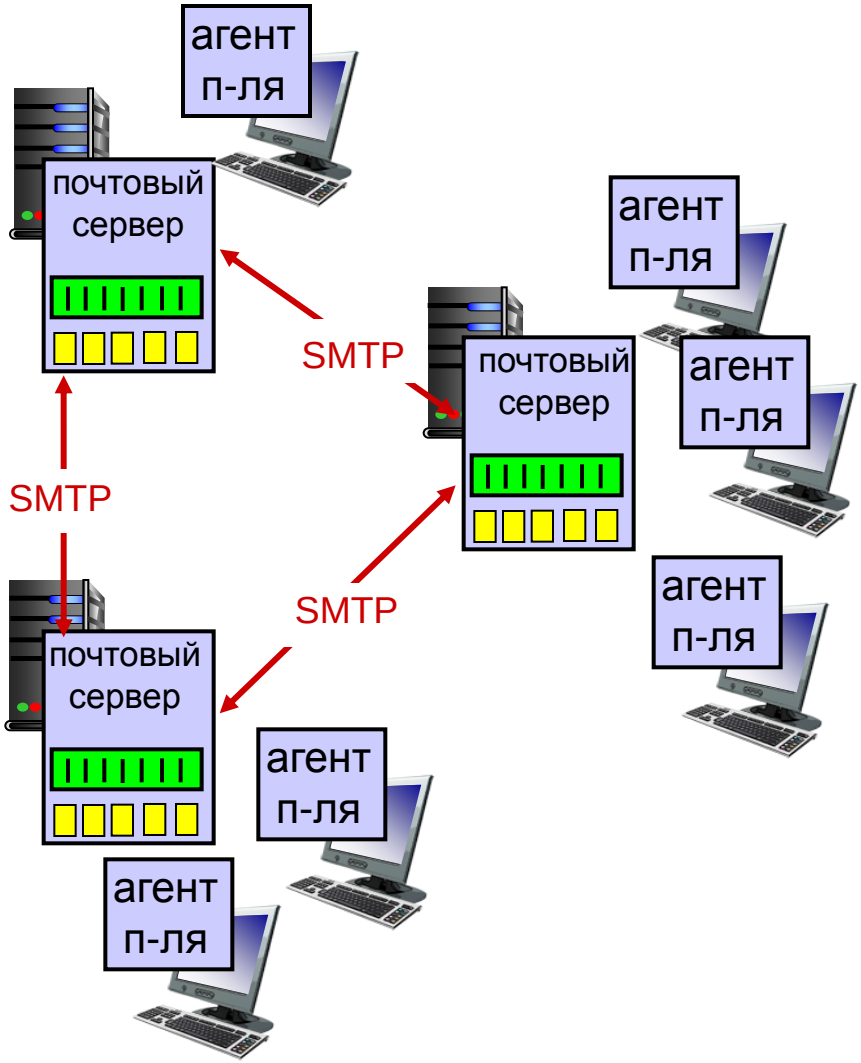
- создают, редактируют, читают сообщения
- примеры: Outlook, Thunderbird, Mail
- исходящие и входящие сообщения хранятся на сервере



# Электронная почта: почтовые серверы

почтовые серверы:

- *почтовый ящик* содержит входящие сообщения пользователя
- *очередь сообщений* для отправки (исходящие)
- *протокол SMTP* между почтовыми серверами для отправки сообщений
  - клиент: отправляющий почтовый сервер
  - сервер: принимающий почтовый сервер

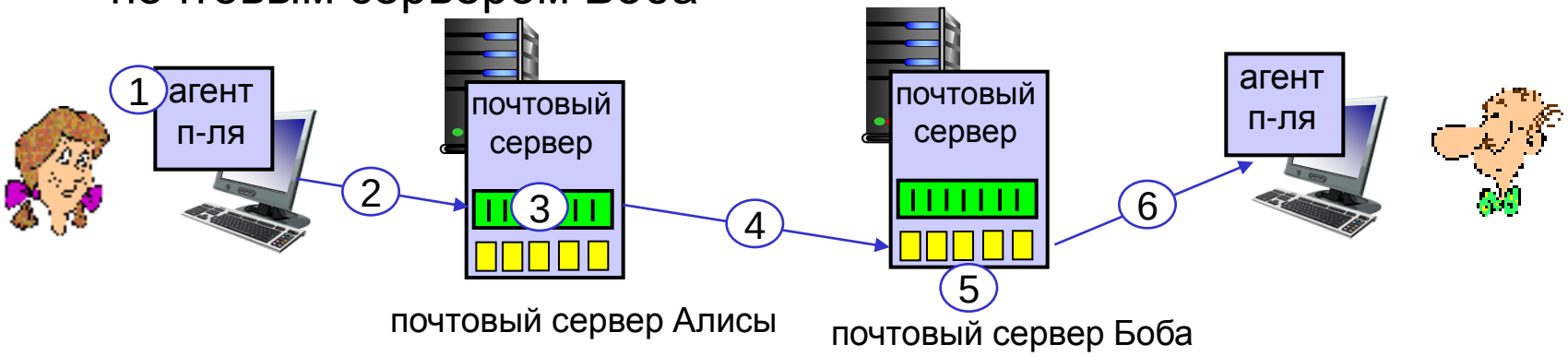


# Электронная почта: протокол SMTP [RFC 2821]

- использует TCP для надежной доставки сообщений от клиента серверу, порт 25
- прямая передача: от отправляющего сервера к принимающему
- три этапа передачи
  - рукопожатие (приветствие)
  - передача сообщений
  - закрытие
- взаимодействие посредством команд/ответов (подобно HTTP, FTP)
  - команды: текст в формате ASCII
  - ответы: код состояния и сообщение
- сообщения должны быть в 7-битном ASCII

# Сценарий: Алиса отправляет сообщение Бобу

1) Алиса использует пользовательский агент для создания сообщения для `bob@someschool.edu`
2) пользовательский агент Алиса отправляет сообщение ее почтовому серверу и помещает в очередь сообщений
3) клиентская часть SMTP открывает TCP-соединение с почтовым сервером Боба
4) клиент SMTP отправляет сообщение через TCP-соединение
5) почтовый сервер Боба помещает сообщение в почтовый ящик Боба
6) Боб запускает свой агент пользователя и читает сообщение



# Пример сеанса SMTP

```
C: 220 hamburger.edu
K: HELO crepes.fr
C: 250 Hello crepes.fr, pleased to meet you
K: MAIL FROM: <alice@crepes.fr>
C: 250 alice@crepes.fr... Sender ok
K: RCPT TO: <bob@hamburger.edu>
C: 250 bob@hamburger.edu ... Recipient ok
K: DATA
C: 354 Enter mail, end with "." on a line by itself
K: Вы любите кетчуп?
K: Как насчет анчоусов?
K: .
C: 250 Message accepted for delivery
K: QUIT
C: 221 hamburger.edu closing connection
```

# Соединение с SMTP-сервером

- выполните команду **`telnet` *`имя_сервера`* `25`**
- ждите ответ 220 от сервера
- вводите команды HELO, MAIL FROM, RCPT TO, DATA, QUIT

это позволит вам отправить сообщение электронной почты без применения пользовательского агента

# SMTP: Заключение

- SMTP использует постоянные соединения
- SMTP требует, чтобы сообщения были (тело и заголовки) в 7-битном формате ACSII
- SMTP использует комбинацию `CRLF.CRLF` для обозначения конца сообщения

*сравнение с HTTP:*

- HTTP: протокол получения (pull)
- SMTP: протокол отправки (push)
- оба используют команды, ответы и коды состояний в формате ASCII
- HTTP: каждый объект инкапсулирован в свое собственное ответное сообщение
- SMTP: все объекты в одном сообщении

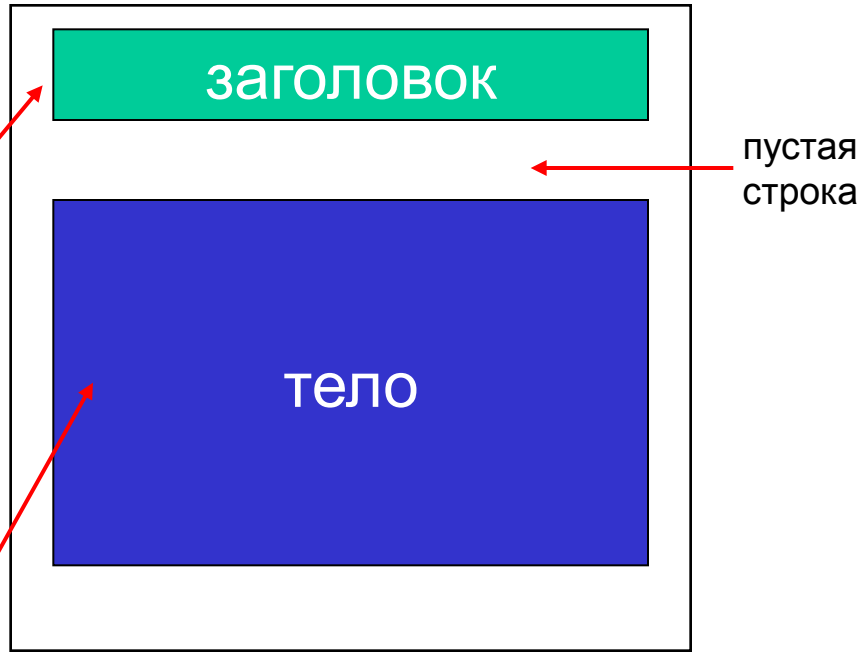
# Формат сообщения электронной почты

SMTP: протокол для обмена сообщениями электронной почты

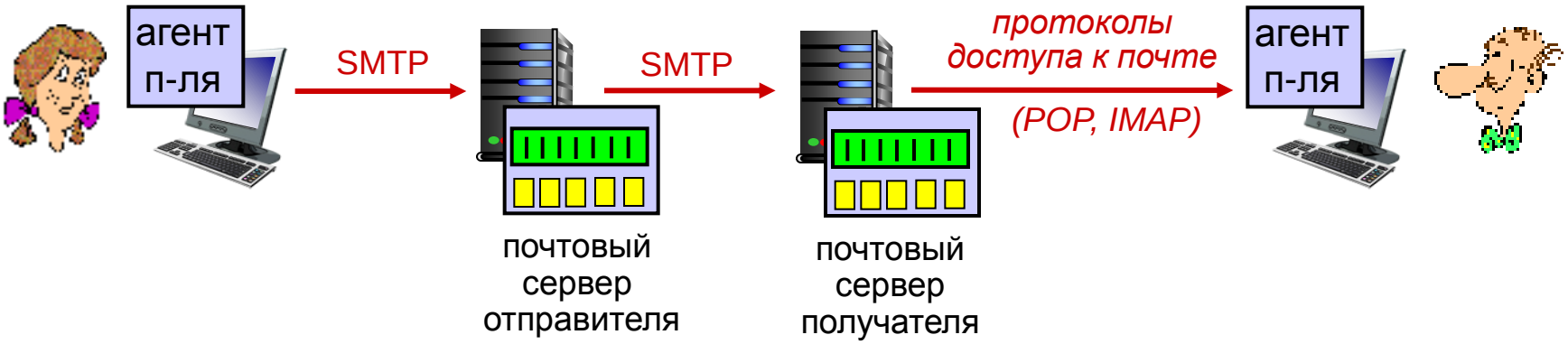
RFC 822: документ, определяющий формат почтового сообщения:

- строки заголовка:
  - To:
  - From:
  - Subject:

  *не путать с* SMTP-командами MAIL FROM, RCPT TO:!
- тело: само сообщение
  - только ASCII-символы



# Протоколы доступа к почте




- SMTP: доставка/хранение до сервера получателя
- протокол почтового доступа: получение сообщений с сервера
  - POP: Post Office Protocol [RFC 1939]: авторизация, загрузка
  - IMAP: Internet Mail Access Protocol [RFC 1730]: расширенные функции, включая манипуляцию сообщениями на сервере
  - HTTP: Gmail, Hotmail, Yahoo! Mail, и т.д.

# Протокол POP3

*фаза авторизации*

- команды клиента:
  - **`user`**: имя пользователя
  - **`pass`**: пароль
- ответы сервера
  - **`+OK`**
  - **`-ERR`**

*фаза транзакции*

клиент:

- **`list`**: вывод списка сообщений
- **`retr`**: вызов сообщения по номеру
- **`dele`**: удаление
- **`quit`**

```
C: +OK POP3 server ready
K: user bob
C: +OK
K: pass hungry
C: +OK user successfully logged on
```

```
K: list
C: 1 498
C: 2 912
C: .
K: retr 1
C: <содержимое сообщения 1>
C: .
K: dele 1
K: retr 2
C: <содержимое сообщения 1>
C: .
K: dele 2
K: quit
C: +OK POP3 server signing off
```

# POP3 и IMAP

*еще о POP3*

- в предыдущем примере использовался POP3 в режиме «загрузить-и-удалить»
  - Боб не может заново прочитать сообщение с другого места
- режим POP3 «загрузить-и-сохранить»: можно получать копии сообщения на различных клиентских местах
- POP3 не сохраняет состояние сеанса

*IMAP*

- все сообщения хранятся в одном месте - на сервере
- позволяет организовать хранение в каталогах
- сохраняет состояние пользовательского сеанса:
  - имена каталогов и соответствие между идентификаторами сообщений и каталогами

# Глава 2: План

2.1 Принципы сетевых приложений
- архитектура
- требования

2.2 Всемирная паутина и HTTP

2.3 FTP

2.4 Электронная почта
- SMTP, POP3, IMAP

2.5 DNS

2.6 Одноранговые (P2P) приложения

2.7 Программирование сокетов UDP и TCP

# DNS: система доменных имен

*люди:* различные идентификаторы:

- ИНН, имя, номер паспорта

*хосты в Интернете, маршрутизаторы:*

- IP-адрес (32-битный) – используется для направления дейтаграмм
- «имя», например, www.yahoo.com – используется людьми

*В:* как установить соответствие между именем и IP-адресом?

*Система DNS:*

- *распределенная база данных* ,реализованная в иерархии *серверов имен*
- *протокол прикладного уровня:* хосты, DNS-серверы взаимодействуют с целью *разрешения* имен (трансляции имени в адрес)
  - примечание: функция ядра сети, реализованная на прикладном уровне
  - вся сложность на границе сети

# DNS: службы, структура

*службы DNS*

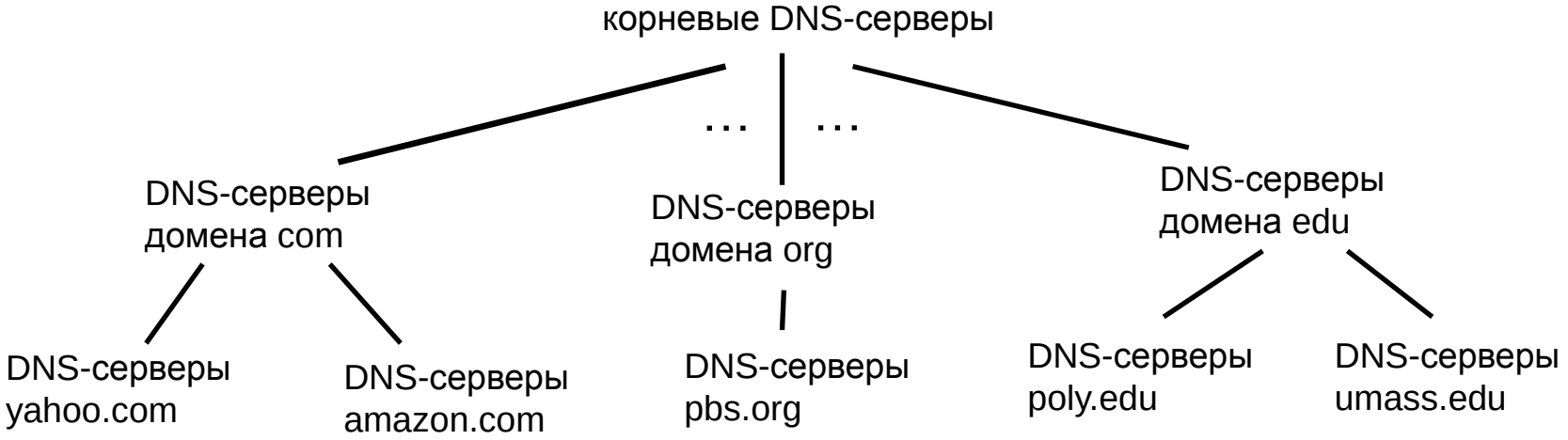
- трансляция имен хостов в IP-адреса
- поддержка псевдонимов хостов
  - канонические имена и псевдонимы
- поддержка псевдонимов почтовых серверов
  - реплицированные веб-серверы: несколько IP-адресов соответствуют одному имени

*В: почему бы не централизовать службу DNS?*

- единая точка отказа
- объем трафика
- удаленная центральная база данных
- обслуживание

*О: нет масштабируемости!*

# DNS: распределенная, иерархическая база данных

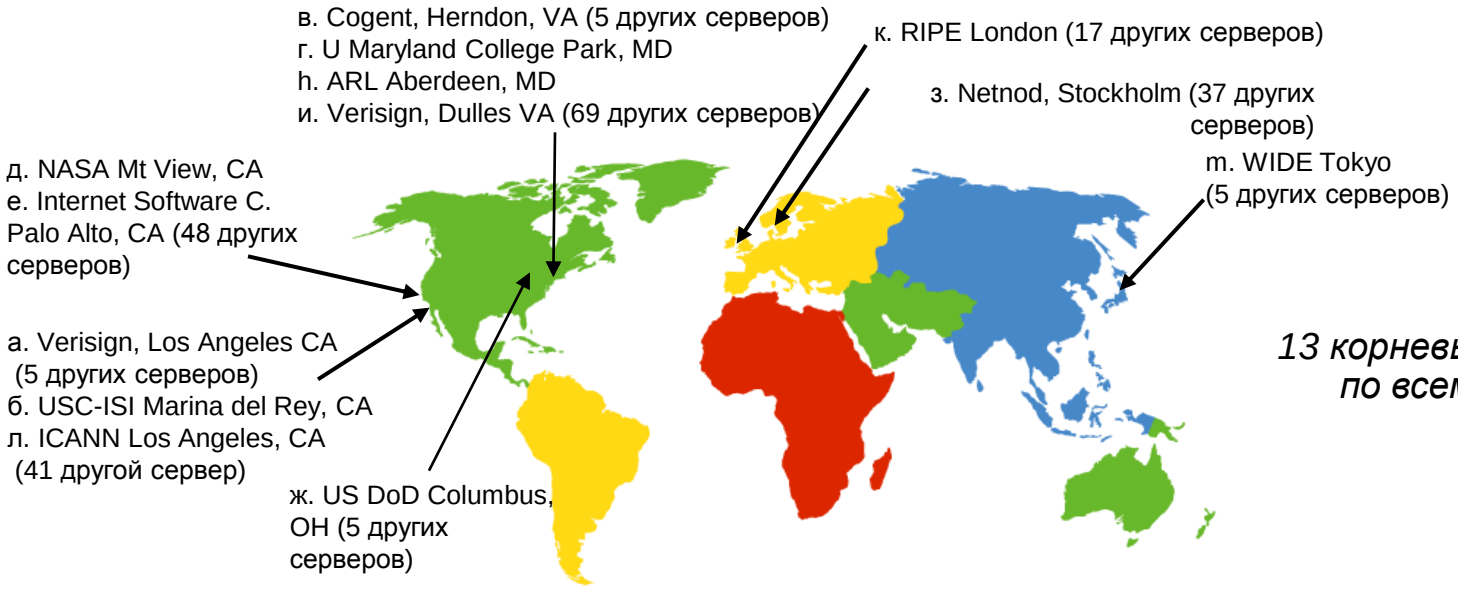



*клиенту нужен IP-адрес для www.amazon.com; приблизительные действия:*

- клиент запрашивает корневой сервер и получает адрес DNS-сервера домена com
- клиент запрашивает DNS-сервер домена com и получает адрес DNS для amazon.com
- клиент запрашивает DNS-сервер amazon.com и получает IP-адрес сервера www.amazon.com

# DNS: корневые серверы

- локальные DNS-серверы обращаются к ним, если не могут разрешить имя
- корневой DNS-сервер
  - обращается к авторитетным DNS-серверам, если сам не может обработать запрос
  - получает соответствующую пару имя-адрес
  - возвращает пару имя-адрес локальному DNS




*13 корневых DNS-серверов по всему миру*

# Серверы верхнего уровня и авторитетные DNS-серверы

*серверы верхнего уровня:*

- отвечают за домены com, org, net, edu, aero, jobs, а также за национальные домены: uk, fr, ca, jp, ru
- Network Solutions обслуживает серверы верхнего уровня для домена .com
- Educause - для домена .edu

*авторитетные DNS-серверы:*

- Собственные DNS-серверы организации, предоставляющие информацию о соответствии имен хостов организации IP-адресам
- могут обслуживаться самой организацией либо провайдером услуг

# Локальный сервер DNS

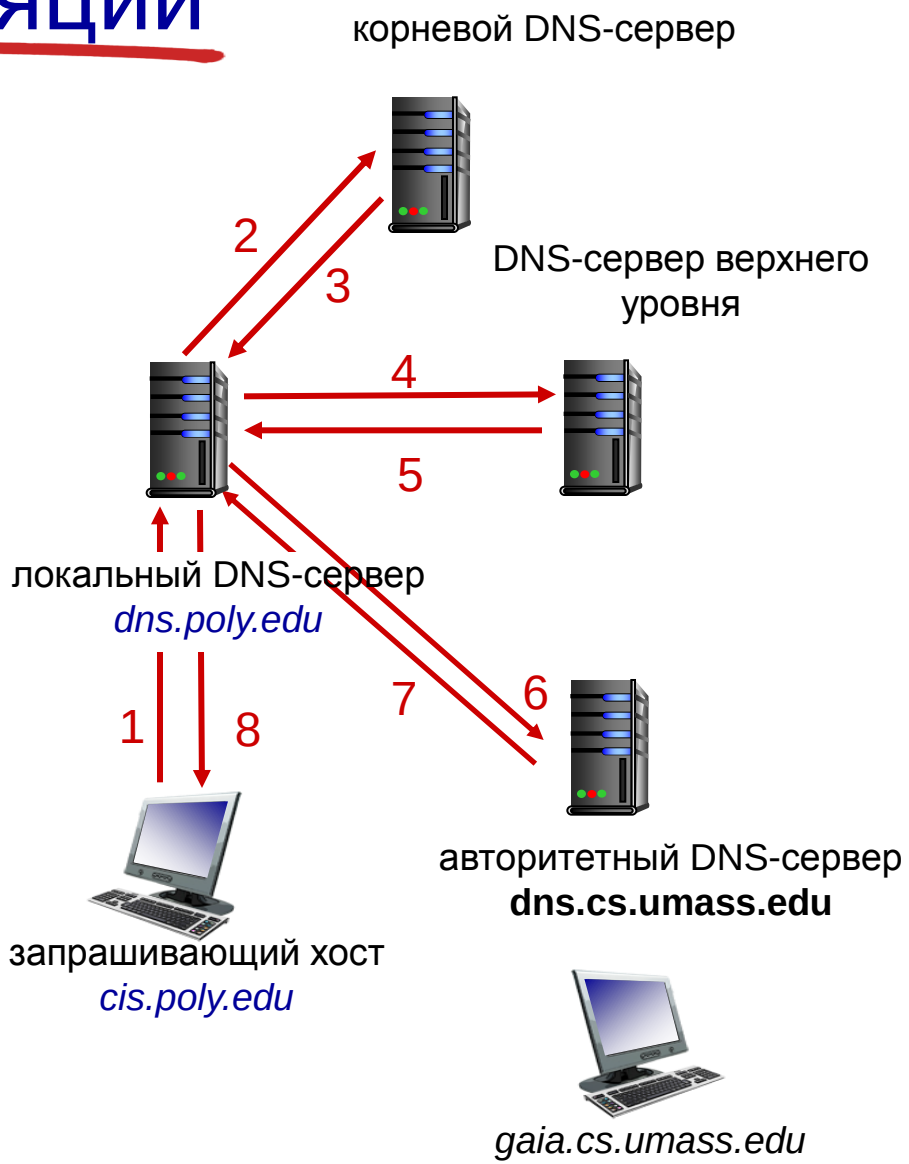
- Может не принадлежать иерархии
- Каждый Интернет-провайдер предоставляет хотя бы один такой сервер
  - Называется также «сервер имен по умолчанию»
- Когда хост производит запрос DNS, он направляется локальному серверу DNS
  - В кэше локального сервера DNS хранятся пары адрес-имя как результаты недавних запросов (хотя могут быть и неактуальными!)
  - Локальный DNS работает как прокси-сервер, перенаправляя запросы в иерархию DNS

# DNS: Пример трансляции имени в адрес

- хосту cis.poly.edu нужен IP-адрес сервера gaia.cs.umass.edu

*итеративный запрос:*

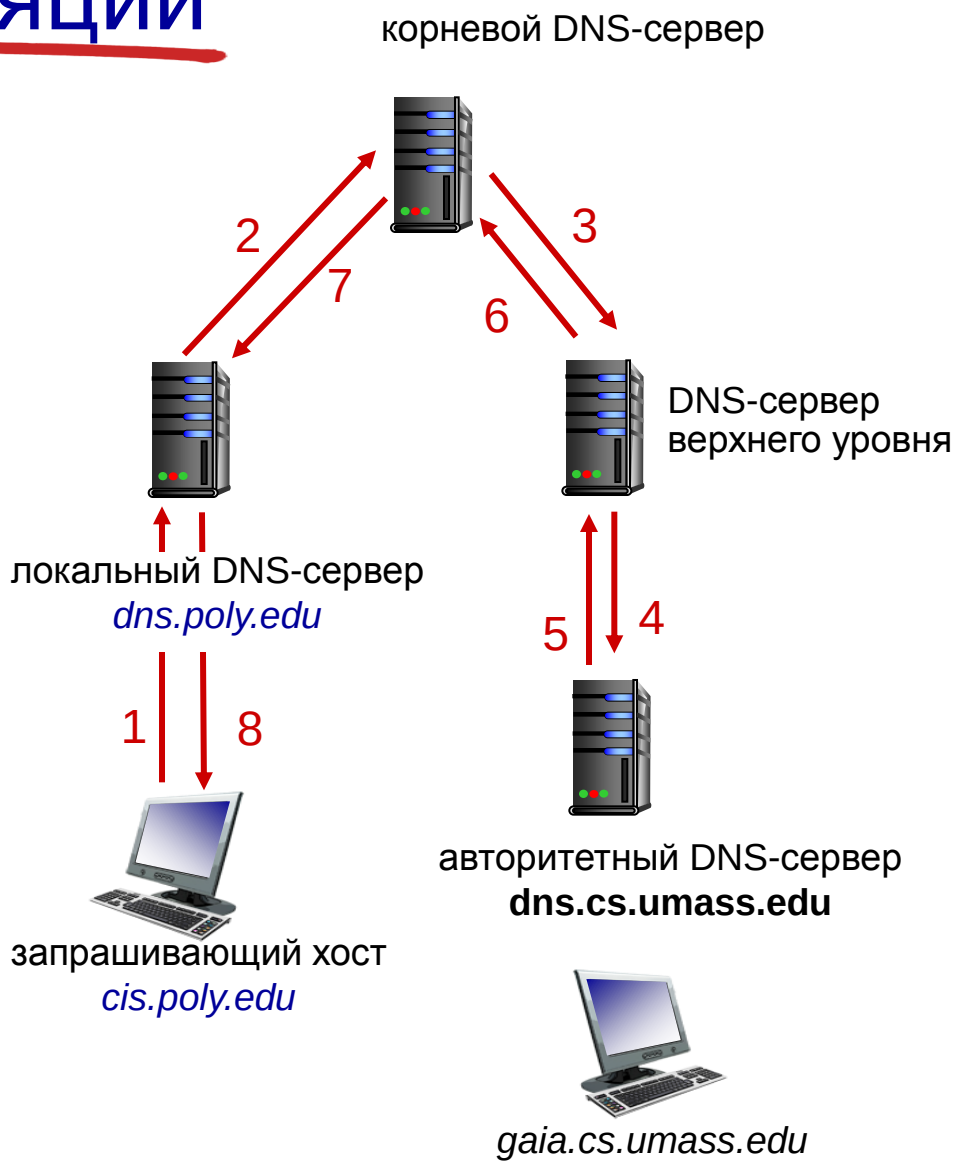
- Сервер, к которому обратились, в ответ дает имена других DNS-серверов
- «Я не знаю ответ, но можете спросить у такого-то сервера»



# DNS: Пример трансляции имени в адрес

*рекурсивный запрос:*

- сервер, к которому обратились, берет на себя всю работу по запросам
- серьезная нагрузка на верхние уровни иерархии?



# DNS: кэширование, обновление записей

- любой DNS-сервер, получив информацию о соответствии имя-адрес, хранит ее в *кэше*
  - информация в кэше периодически сбрасывается (время жизни TTL)
  - в кэше локальных DNS присутствуют данные о серверах верхнего уровня
    - поэтому к корневым серверам можно не обращаться
- записи в кэше могут быть *устаревшими*
  - при изменении IP-адреса какого-нибудь хоста, это не будет известно другим хостам в Интернете, пока не пройдет время TTL
- механизм обновления DNS предложен IETF
  - документ RFC 2136

# Записи DNS

*DNS:* распределенная база данных, хранящая ресурсные записи (RR)

формат ресурсной записи: **(имя, значение, тип, ttl)**

тип=A

- **имя** – имя хоста
- **значение** – IP-адрес

тип=NS

- **имя** – имя домена (например, foo.com)
- **значение** – имя авторитетного DNS-сервера для этого домена

тип=CNAME

- **имя** -псевдоним для реального (канонического) имени
- **www.ibm.com** – на самом деле имеет имя

**servereast.backup2.ibm.com**

- **значение** – каноническое имя

тип=MX

- **значение** – имя почтового сервера для соответствующего **имени**

# Сообщения DNS

- *запрос* и *ответ* – с одинаковым *форматом сообщения*

заголовок сообщения

- идентификатор: 16 – битный номер запроса; копируется в ответное сообщение
- флаги:
  - запрос/ответ
  - требуется рекурсия
  - рекурсия возможна
  - авторитетный ответ

# Сообщения DNS

# Добавление DNS-записей

- пример: новый проект Network Utopia
- регистрация имени networkuptopia.com у *DNS-регистратора* (например, в Network Solutions)
  - предоставление имен, IP-адресов авторитетных DNS-серверов (первичного и вторичного)
  - добавление регистратором двух ресурсных записей в DNS-серверы домена верхнего уровня .com :
    **(networkutopia.com, dns1.networkutopia.com, NS)**
    **(dns1.networkutopia.com, 212.212.212.1, A)**
- создание записи типа A авторитетного сервера для www.networkuptopia.com; записи типа MX для networkutopia.com

# Атаки на DNS

## DDoS-атаки

- бомбардировка корневых серверов огромным числом запросов
  - Неуспешны до сегодняшнего дня
  - Фильтрация трафика
  - Локальные DNS-серверы кэшируют адреса серверов верхнего уровня, позволяя делать запросы в обход корневых DNS
- бомбардировка серверов верхнего уровня
  - потенциально более опасна

## Атаки с перенаправлением

- Man-in-middle (человек посередине)
  - Перехват запросов
- отравление DNS
  - Отправка поддельных пакетов DNS-серверу для кэширования

## Использование DNS для DDoS-атаки

- отправка запросов с подменой адреса источника
- «усиление» DNS

# Глава 2: План

2.1 Принципы сетевых приложений
- архитектура
- требования

2.2 Всемирная паутина и HTTP

2.3 FTP

2.4 Электронная почта
- SMTP, POP3, IMAP

2.5 DNS

2.6 Одноранговые (P2P) приложения

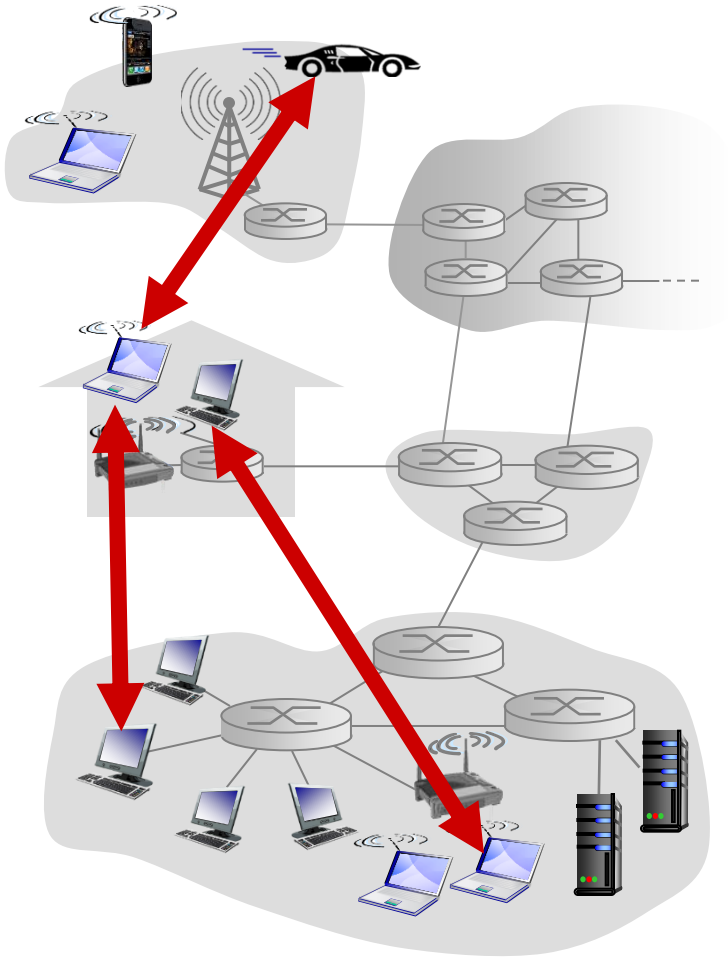
2.7 Программирование сокетов UDP и TCP

# Одноранговая (P2P) архитектура

- *никакого* постоянно работающего сервера
- произвольные конечные системы взаимодействуют напрямую
- узлы подключаются периодически и могут менять IP-адреса

*примеры:*

- файловый обмен (BitTorrent)
- Потоковое видео (KanKan)
- IP-телефония (Skype)



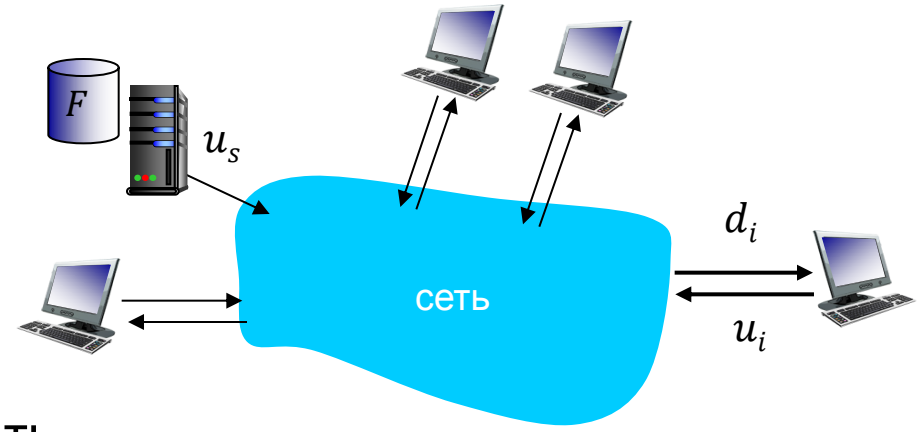
# Файловая раздача: сравнение клиент-серверного и одноранговых вариантов

*Вопрос:* сколько времени уходит на раздачу файла (размером $F$) от одного сервера к $N$ *узлам*?

- исходящая/входящая скорость узла ограничены

# Время раздачи: клиент-серверный вариант

- *сервер:* должен последовательно отправить (выгрузить) $N$ копий файла:
  - время на отправку одной копии: $F/us$
  - для $N$ копий: $NF/us$
- *клиент:* каждый должен загрузить копию файла
  - $d_{min}$ = минимальная скорость загрузки среди клиентов
  - минимальное время загрузки среди клиентов: $F/d_{min}$



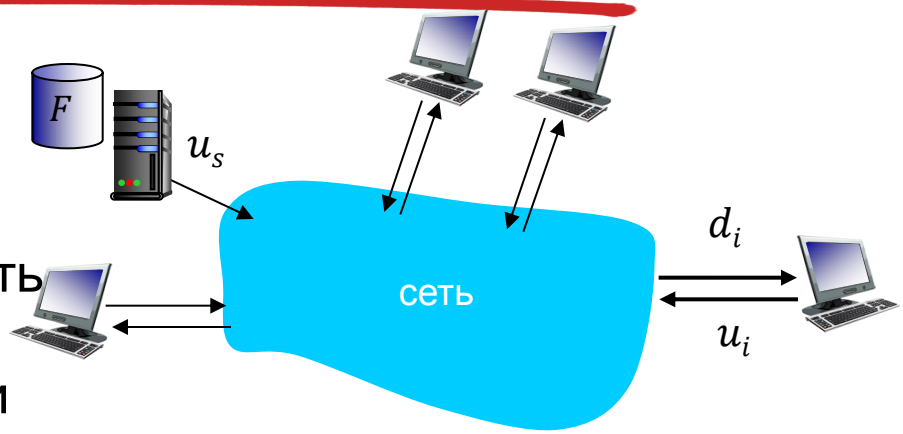

*время на раздачу $F$ бит $N$ клиентам при клиент-серверном подходе*

$$D_{c-s} \geq \max\{NF/u_s, F/d_{min}\}$$

увеличивается прямо пропорционально $N$

# Время раздачи: одноранговый вариант

- *сервер:* должен выгрузить хотя бы одну копию файла
  - время на отправку одной копии: $F/us$
- *клиент:* каждый должен загрузить копию файла
  - минимальное время загрузки среди клиентов: $F/d_{min}$
- *клиенты:* должны суммарно загрузить $NF$ бит
  - максимальная скорость выгрузки (ограничивающая максимальную скорость загрузки) равна $u_s + \sum u_i$




*время на раздачу $F$ бит $N$ клиентам при одноранговом подходе*

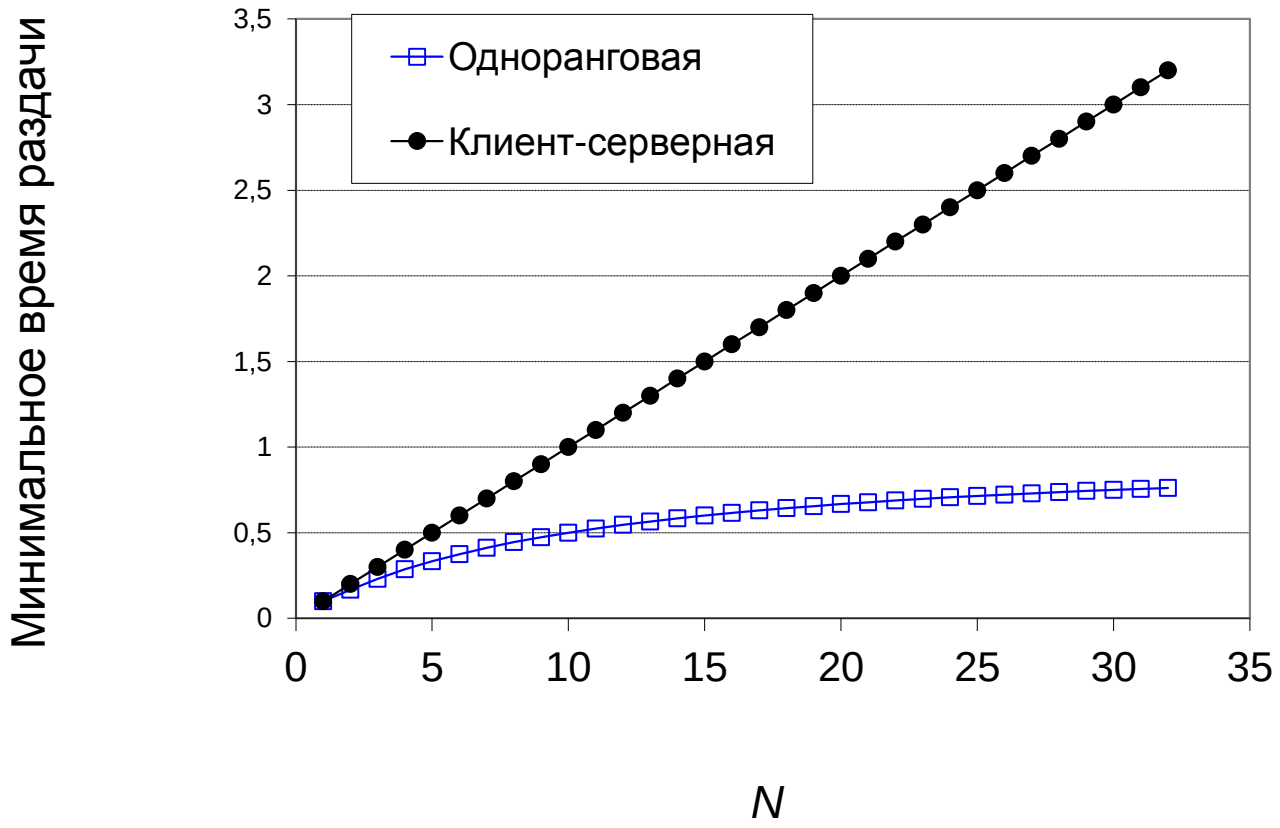
$$D_{P2P} \geq \max\{F/u_s, F/d_{min}, NF/(u_s + \sum u_i)\}$$

увеличивается прямо пропорционально $N$...

… но и с увеличением $N$ возрастает суммарная скорость раздачи

# Сравнение схем раздачи: одноранговая и клиент-серверная

скорость отдачи клиентом = $u$, $F/u$ = 1 ч, $u_s = 10u$, $d_{min} \geq us$

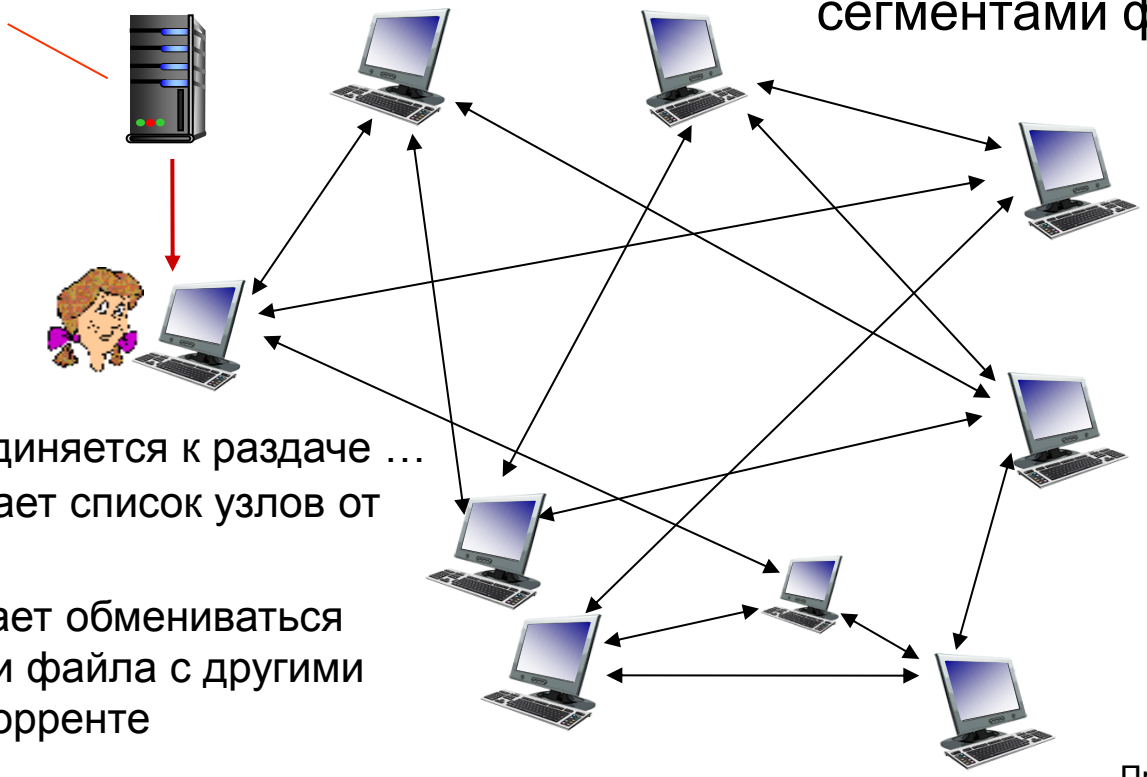


# Одноранговая раздача файлов: BitTorrent

- файл делится на сегменты по 256 Кб
- узлы в торренте получают/отправляют сегменты

*трекер:* отслеживает участников раздачи

*торрент:* группа узлов, обменивающихся сегментами файла



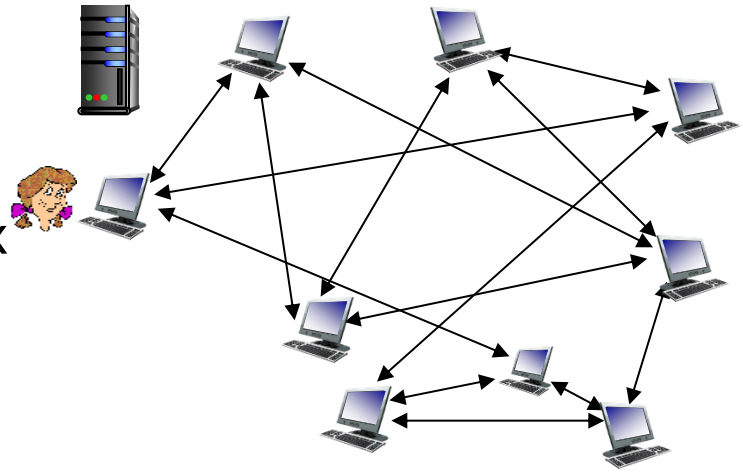
Алиса присоединяется к раздаче …

… получает список узлов от трекера

… и начинает обмениваться сегментами файла с другими узлами в торренте

# Одноранговая раздача файлов: BitTorrent

- присоединяющийся к торренту узел (пир):
  - не имеет сегментов, но начинает получать их от других узлов
  - регистрируется трекером и получает список узлов, соединяется с набором узлов («соседями»)



- узлы одновременно загружают и отдают сегменты
- узел может менять партнеров по обмену сегментами
- *отток:* узлы появляются и исчезают
- при получении всего файла, узел может (эгоистично) покинуть торрент или остаться в раздаче (бескорыстно)

# BitTorrent: запрос и отправка сегментов

*запрос сегментов:*

- в любой момент времени различные узлы хранят разный набор сегментов
- Алиса периодически запрашивает у каждого узла список сегментов, которые у него есть
- Алиса запрашивает отсутствующие у нее сегменты по принципу «сначала редкие»

*отправка: ты-мне-я-тебе*

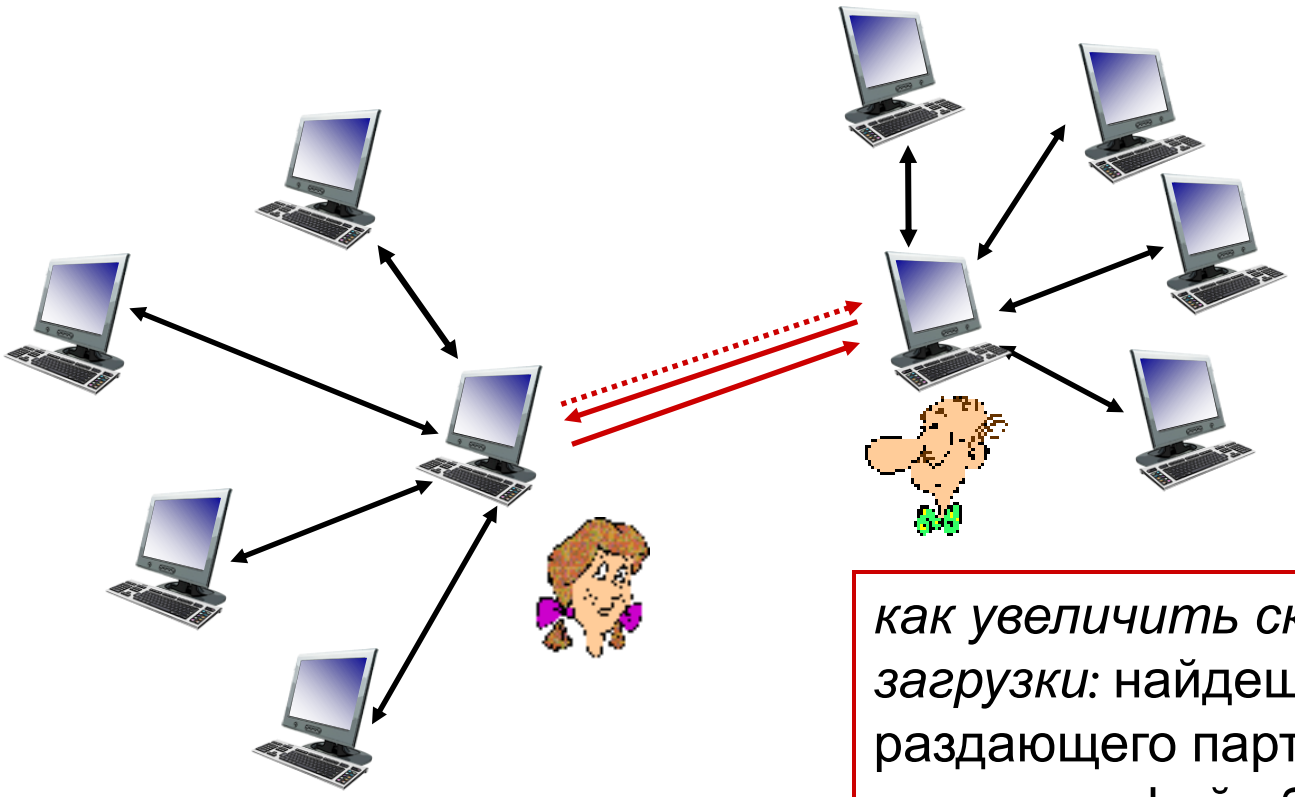
- Алиса отправляет сегменты четырем узлам, которые снабжают ее на *максимальной скорости*
  - другие узлы Алиса блокирует (не получает от них сегменты)
  - каждые 10 с четверка топ-узлов пересматривается
- каждые 30 с: случайно выбирается новый узел и начинается отправка ему сегментов
  - «оптимистическая разблокировка» этого узла
  - новый узел может присоединиться к топ-четверке

# BitTorrent: ты-мне-я-тебе

(1) Алиса оптимистически разблокирует Боба

(2) Алиса становится одним из четырех лучших раздающих для Боба; Боб отвечает взаимностью

(3) Боб становится одним из четырех лучших раздающих для Алисы



*как увеличить скорость загрузки:* найдешь лучшего раздающего партнера - получишь файл быстрее!

# Распределенная хеш-таблица (DHT)

- DHT: распределенная база данных одноранговой сети
- база данных содержит пары (ключ, значение); примеры:
  - ключ: ИНН; значение: имя владельца
  - ключ: название видеофайла; значение: IP-адрес хоста
- пары (ключ, значение) распределены среди миллионов узлов
- узел делает запрос к таблице с помощью ключа
  - DHT возвращает значение, соответствующее ключу
- узлы добавляют пары (ключ, значение) в таблицу

# Как назначать ключи узлам?

- главная задача:
  - присваивание пар (ключ, значение) узлам
- основная идея:
  - преобразовать ключ в целое число
  - каждому узлу присвоить целочисленный идентификатор
  - сохранить пару (ключ, значение) в узле с ближайшим к ключу идентификатором

# Идентификаторы в хеш-таблицах

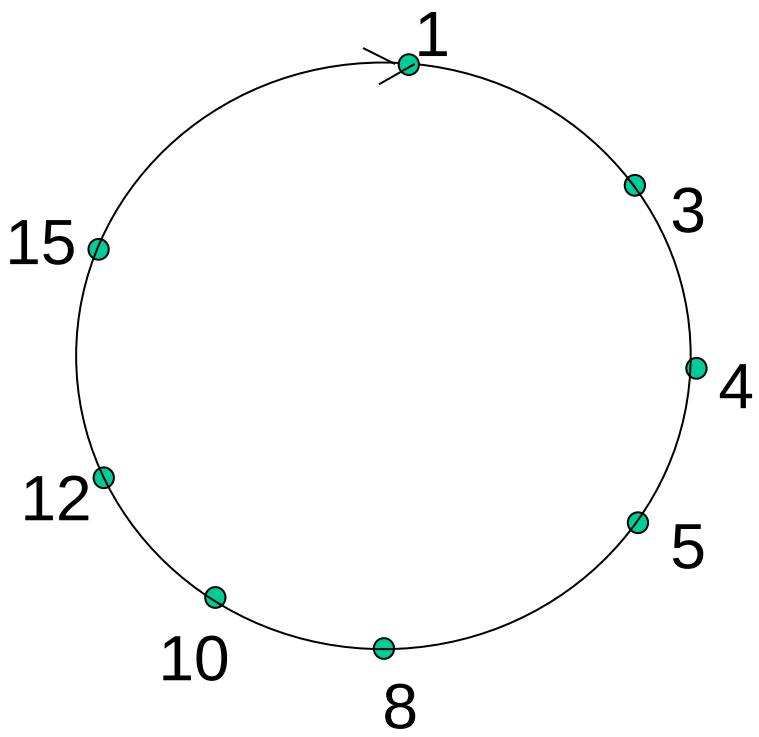
- каждому узлу присваивается целочисленный идентификатор в диапазоне $[0,2^n - 1]$
  - идентификатор выражается $n$-битным представлением

- каждый ключ – целое число в том же диапазоне
- используем хеш-функцию для получения целочисленных из исходных ключей
  - ключ = хеш(Led Zeppelin IV)
  - поэтому используем слово *«хеш»* в термине хеш-таблица

# Присваивание ключей узлам

- правило: присваивать ключ узлу с *ближайшим* идентификатором.
- договоренность: ближайший –*непосредственный последователь*
- пример $n$=4; узлы: 1,3,4,5,8,10,12,14;
  - ключ = 13, последователь = 14
  - ключ = 15, последователь = 1

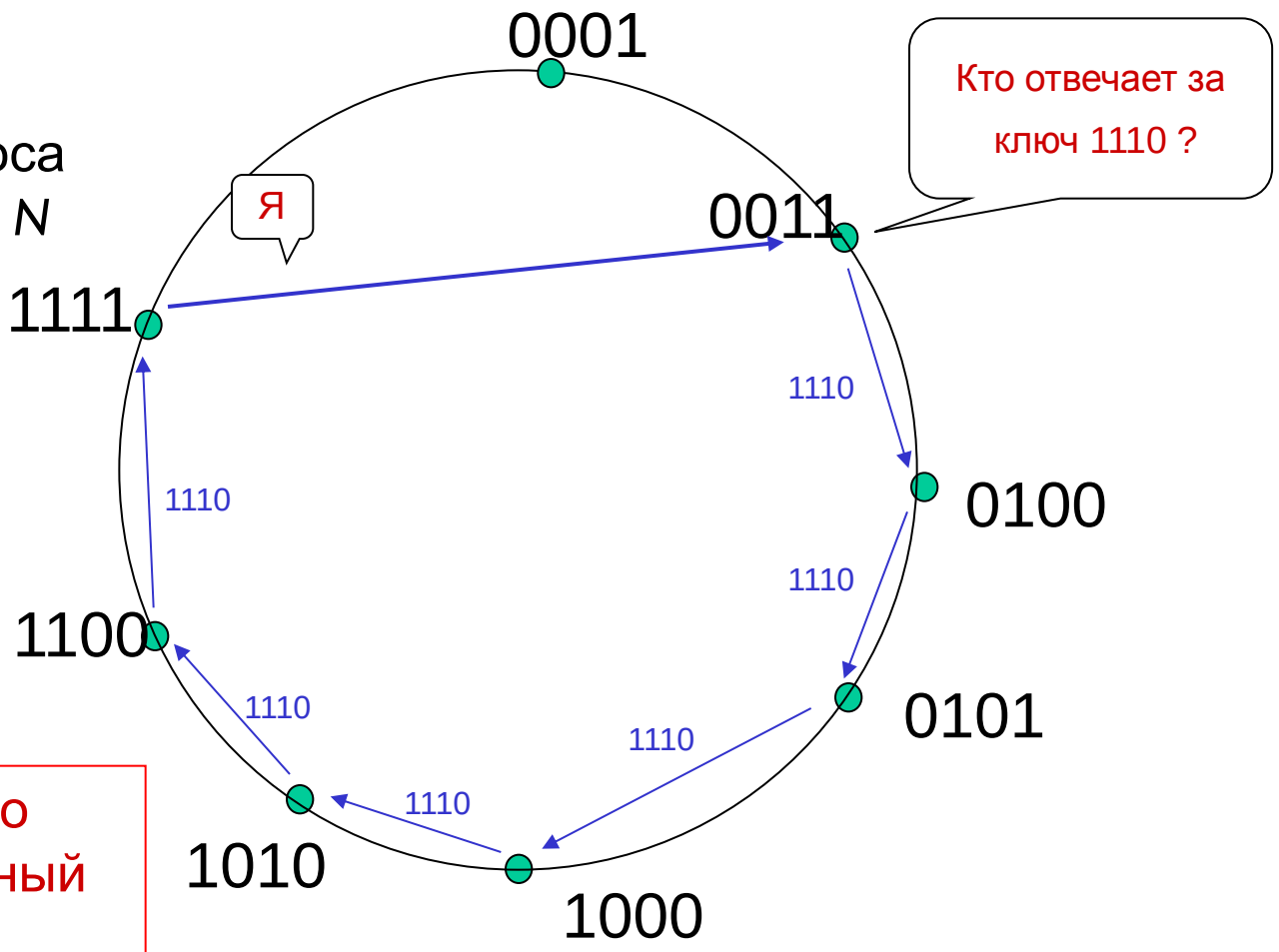
# Циркулярные хеш-таблицы




- каждый узел заботится только о непосредственных последователе и предшественнике
- «оверлейная» сеть

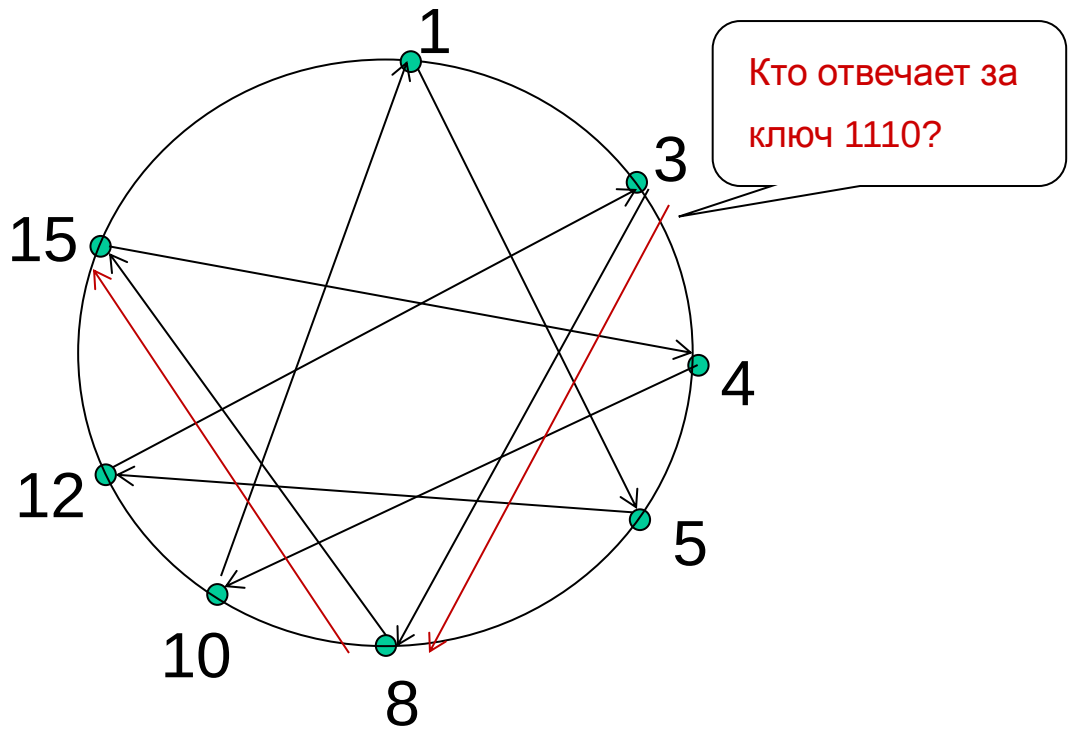
# Циркулярные хеш-таблицы

*в среднем O(N)*
переданных
сообщений для
обработки запроса
при числе узлов *N*



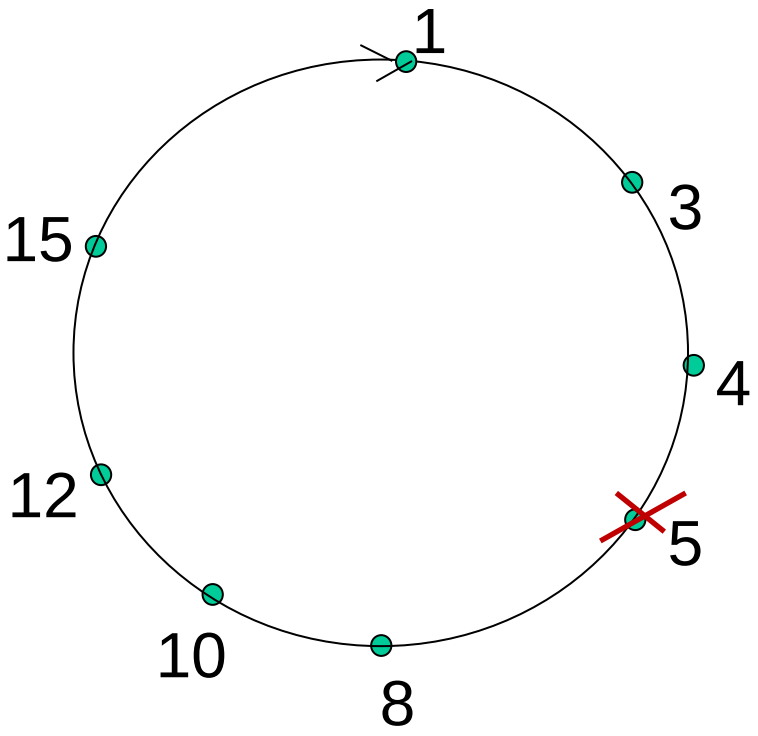

ближайший - это непосредственный последователь

# Циркулярная хеш-таблица со ссылками




- каждый узел отслеживает последователя, предшественника и соседа, на которого указывает ссылка
- число сообщений уменьшилось с 6 до 2
- возможный вариант построения таблицы: $O(\log N)$ соседей и $O(\log N)$ сообщений на запрос

# Отток пиров




**обработка оттока пиров:**

- узлы появляются и исчезают (происходит отток)
- каждый узел отслеживает двух своих последователей
- каждый узел периодически отправляет эхо-запрос двум последователям, проверяя «живы» ли они
- при исчезновении первого последователя второй становится на его место

*пример: пир 5 исчезает*

- пир 4 обнаруживает, что 5-й пропал; его действия: делает 8-го своим первым последователем; спрашивает у 8-го о его первом последователе (10); делает 10-го своим вторым последователем
- что будет когда пир 13 захочет присоединиться к таблице?

# Глава 2: План

2.1 Принципы сетевых приложений
- архитектура
- требования

2.2 Всемирная паутина и HTTP

2.3 FTP

2.4 Электронная почта
- SMTP, POP3, IMAP

2.5 DNS

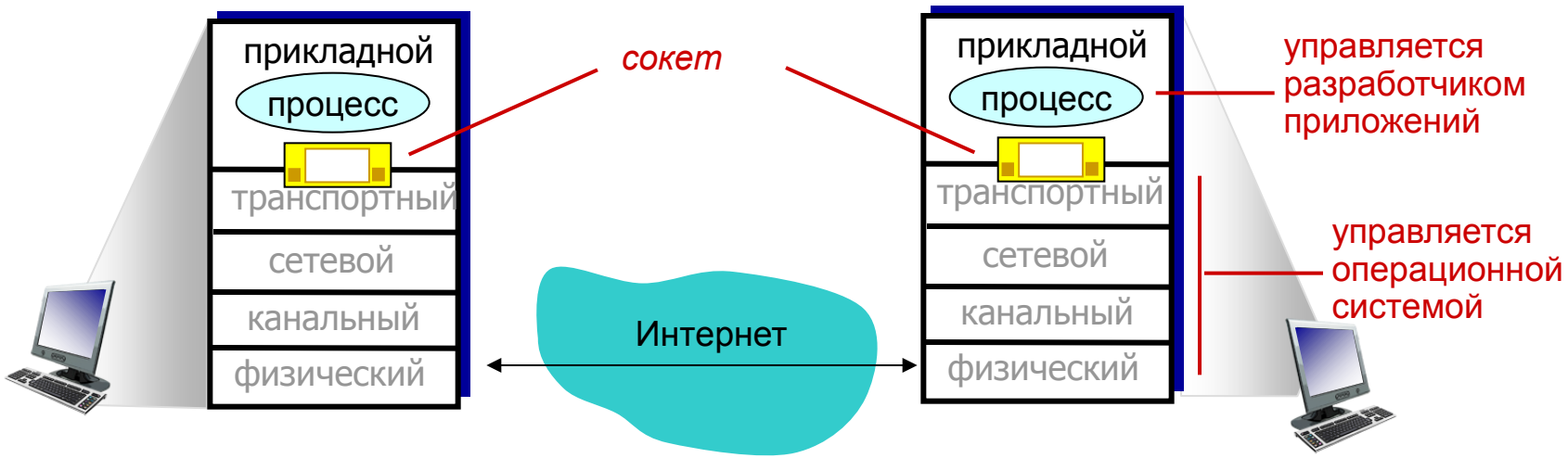
2.6 Одноранговые (P2P) приложения

2.7 Программирование сокетов UDP и TCP

# Программирование сокетов

*цель:* научиться разрабатывать клиент-серверные приложения, которые взаимодействуют при помощи сокетов

*сокет:* дверь между процессом приложения и транспортным протоколом



# Программирование сокетов

*Два типа сокетов для двух транспортных служб:*

- *UDP:* дейтаграммы без гарантии доставки
- *TCP:* гарантия доставки, данные в виде потока байтов

*Пример приложения:*

1. Клиент читает строку символов (данные) с клавиатуры и отправляет данные серверу.
2. Сервер получает данные и преобразует символы в верхний регистр.
3. Сервер отправляет измененные данные клиенту.
4. Клиент получает измененные данные и отображает строку на своем экране.

# Программирование сокетов *с использованием UDP*

UDP: без установления соединения

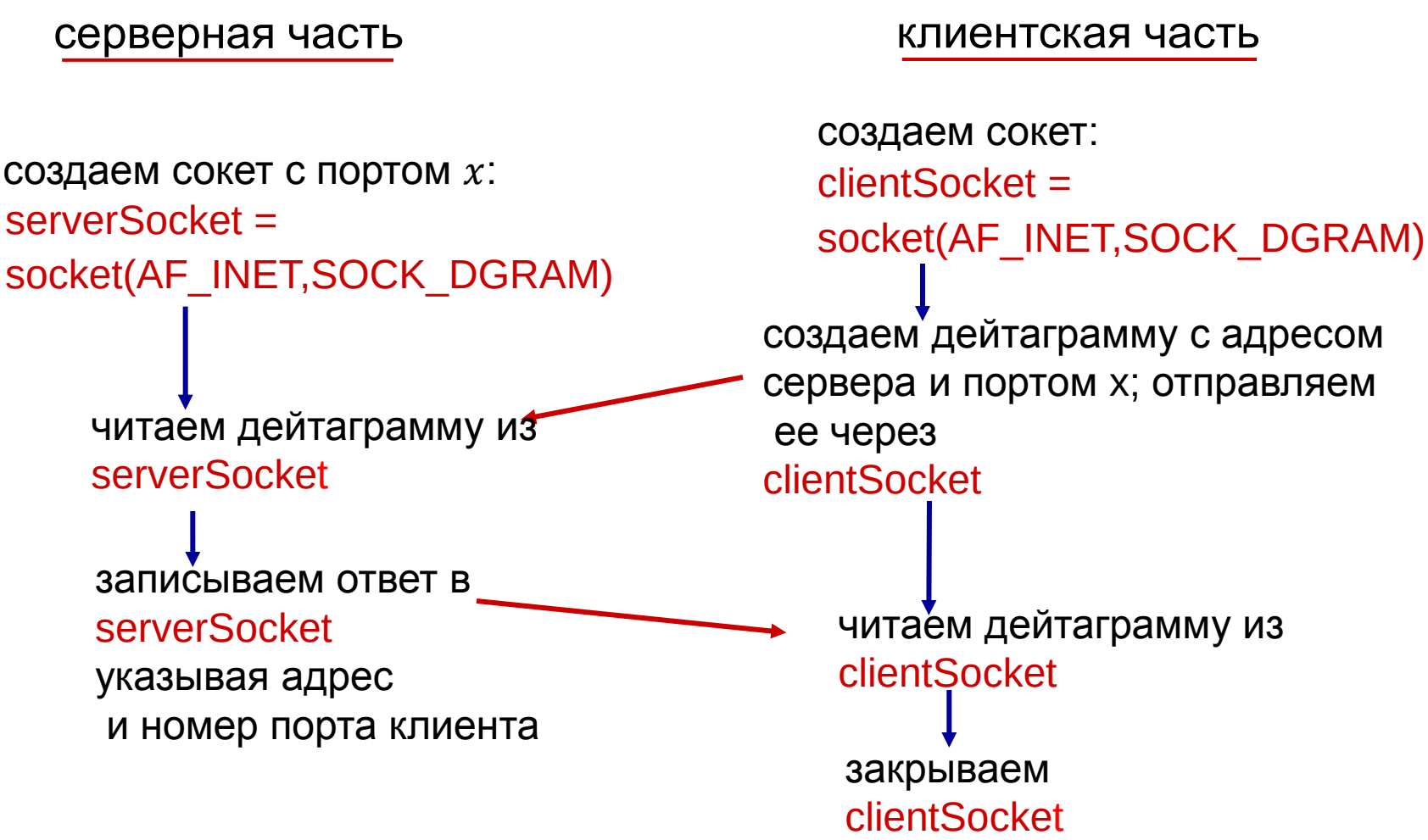
- никакого рукопожатия перед отправкой данных
- отправитель прикрепляет к каждому пакету IP-адрес и номер порта хоста назначения
- получатель извлекает из принятого пакета IP-адрес и номер порта отправителя

UDP: передаваемые данные могут быть потеряны или получены не в том порядке

С точки зрения приложения:

- Протокол UDP обеспечивает *ненадежную* передачу сгруппированных байт (дейтаграмм) между клиентом и сервером

# Взаимодействие клиента и сервера через сокеты: UDP



# Пример: UDP-клиент

*UDPClient на языке Python*

включаем библиотеку для работы с сокетами

создаем сокет клиента

получаем вводимые пользователем символы

прикрепляем к сообщению имя и порт сервера; отправляем в сокет

читаем принятые символы из сокета в переменную

выводим полученную строку и закрываем сокет

```
from socket import *

serverName = 'hostname'

serverPort = 12000

clientSocket = socket(socket.AF_INET,

 socket.SOCK_DGRAM)

message = raw_input('Input lowercase sentence:')

clientSocket.sendto(message,(serverName, serverPort))

modifiedMessage, serverAddress =

 clientSocket.recvfrom(2048)

print modifiedMessage

clientSocket.close()
```

# Пример: UDP-сервер

*UDPServer на языке Python*

```
from socket import *

serverPort = 12000

serverSocket = socket(AF_INET, SOCK_DGRAM)

serverSocket.bind(('', serverPort))

print "The server is ready to receive"

while 1:

 message, clientAddress = serverSocket.recvfrom(2048)

 modifiedMessage = message.upper()

 serverSocket.sendto(modifiedMessage, clientAddress)
```

создаем сокет

связываем с сокетом порт 12000

бесконечный цикл

читаем из сокета в переменную *message*, получаем адрес и порт клиента

отправляем измененную строку обратно клиенту

# Программирование сокетов *с использованием TCP*

условия для обращения клиента к серверу

- серверный процесс должен быть уже запущен
- на сервере должен быть создан входной сокет (дверь) для обращения клиента

как происходит контакт:

- создается TCP-сокет с указанием IP-адреса и номера порта серверного процесса
- *создавая сокет,* клиент устанавливает TCP-соединение с сервером

- при обращении клиента

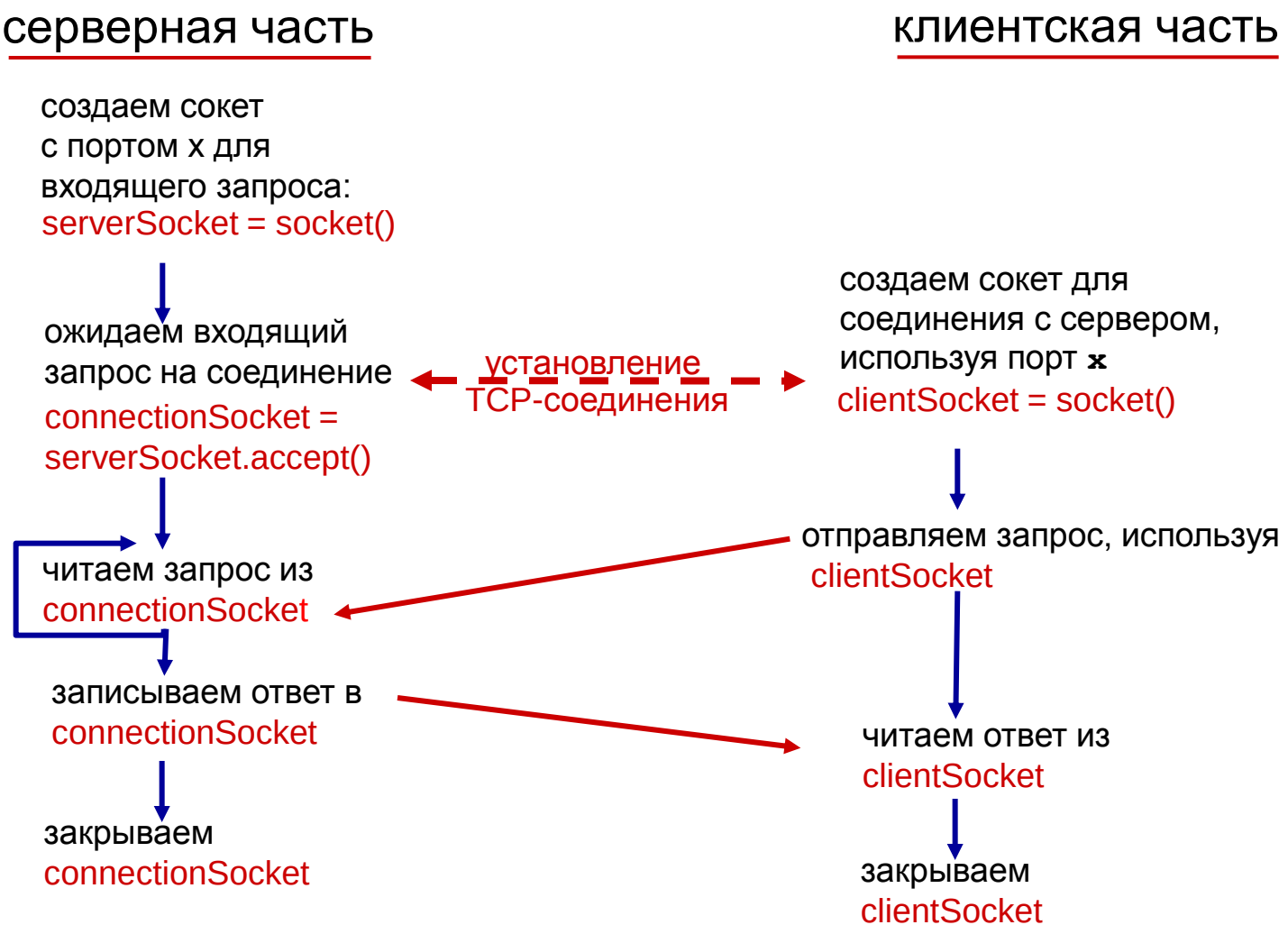
*TCP-сервер создает новый сокет* для серверного процесса, чтобы взаимодействовать конкретно с этим клиентом

  - таким образом сервер может общаться с множеством клиентов
  - по номерам портов можно различать клиентов (подробнее в главе 3)

С точки зрения приложения:

TCP обеспечивает надежную передачу упорядоченного потока байтов между сервером и клиентом

# Взаимодействие клиента и сервера через сокеты: TCP



# Пример: TCP-клиент

*TCPClient на языке Python*

```
from socket import *

serverName = 'servername'

serverPort = 12000

clientSocket = socket(AF_INET, SOCK_STREAM)

clientSocket.connect((serverName,serverPort))

sentence = raw_input('Input lowercase sentence:')

clientSocket.send(sentence)

modifiedSentence = clientSocket.recv(1024)

print 'From Server:', modifiedSentence

clientSocket.close()
```

создаем TCP-сокет для связи с сервером по порту 12000

нет необходимости прикреплять порт и адрес сервера

# Пример: TCP-сервер

*TCPServer на языке Python*

создаем входной TCP-сокет

сервер начинает слушать входящие TCP-запросы

бесконечный цикл

создаем новый сокет соединения для входящего запроса

читаем байты из сокета (не используя адрес , как в UDP)

закрываем соединение с этим клиентом (но *не* входной сокет)

```
from socket import *
serverPort = 12000
serverSocket = socket(AF_INET,SOCK_STREAM)
serverSocket.bind(('',serverPort))
serverSocket.listen(1)
print 'The server is ready to receive'
while 1:
 connectionSocket, addr = serverSocket.accept()

 sentence = connectionSocket.recv(1024)
 capitalizedSentence = sentence.upper()
 connectionSocket.send(capitalizedSentence)
 connectionSocket.close()
```

# Заключение

*изучение сетевых приложений завершено!*

- архитектура приложений
  - клиент-серверная
  - одноранговая (P2P)
- требования к службам:
  - надежность, пропускная способность, время доставки
- транспортные службы Интернета
  - надежная, с установлением соединения: TCP
  - ненадежная с использованием дейтаграмм: UDP
- используемые протоколы:
  - HTTP
  - FTP
  - SMTP, POP, IMAP
  - DNS
  - P2P: BitTorrent, DHT
- программирование сокетов: TCP-, UDP-сокеты

# Заключение

*самое важное о протоколах!*

- обмен стандартными сообщениями запрос-ответ:
  - запрос клиента (информация или услуга)
  - ответ сервера (данные, код состояния)
- форматы сообщения:
  - заголовки: поля с информацией о данных
  - данные: передаваемая информация

*важные темы:*

- управляющая информация и данные
  - в полосе, вне полосы
- централизованные и распределенные системы
- сохранение состояния и без сохранения
- надежная и ненадежная передача сообщений
- «сложность на границе сети»