



Архитектура компьютерных сетей



Останков Александр Иванович

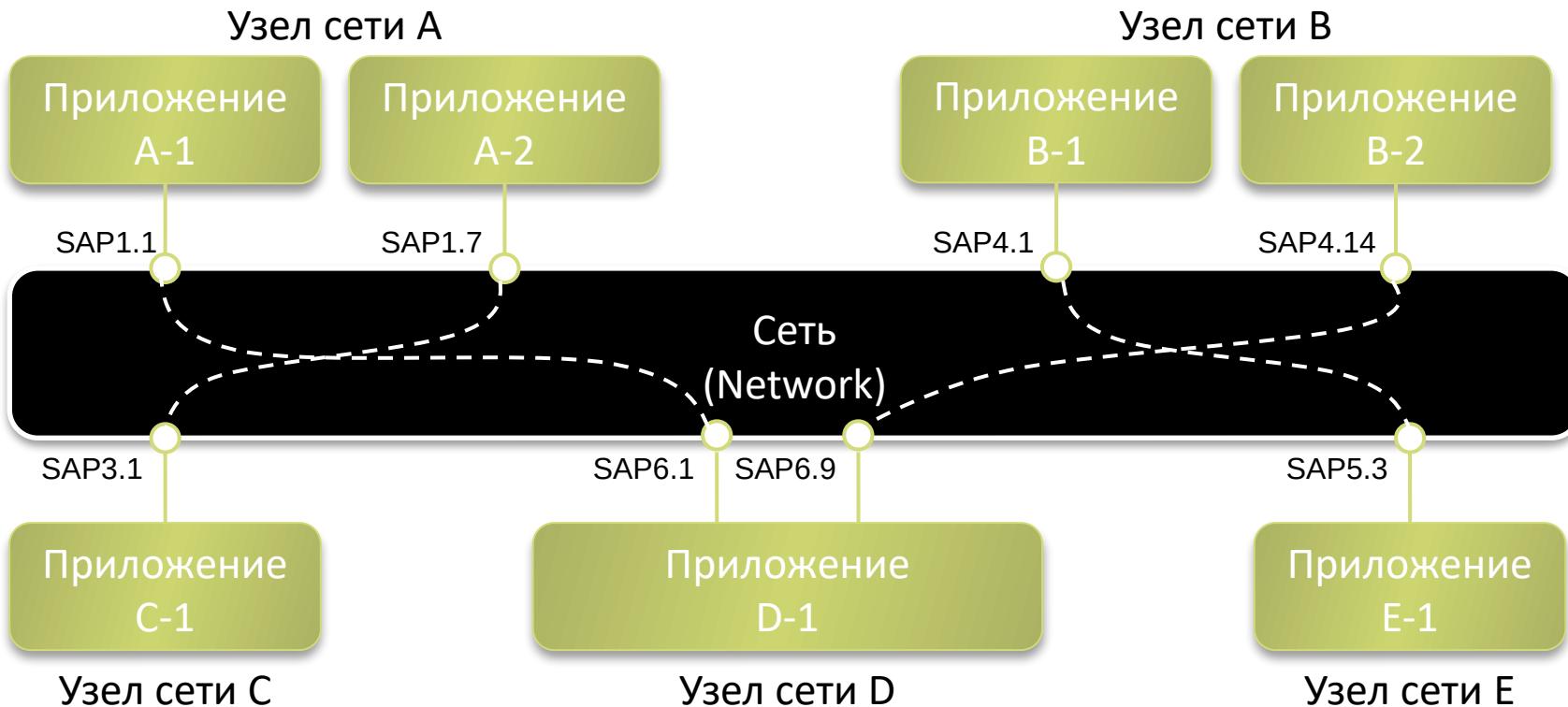
План курса

1. Введение в компьютерные сети
- 2. Основные методы построения СПД**
 - 2.1. Методы коммутации и мультиплексирования
 - 2.2. Программная модель сетевого взаимодействия**
 - 2.4. Общие принципы построения netware (сетевого обеспечения)
 - 2.5. Функциональная совместимость netware и стандарты
3. Архитектура Internet Protocol Suite (TCP/IP)
4. Архитектура модулей физического уровня
5. Технологии беспроводных сетей
6. Архитектура модулей канального уровня
7. Протоколы транспортного уровня
8. Технологии WWW

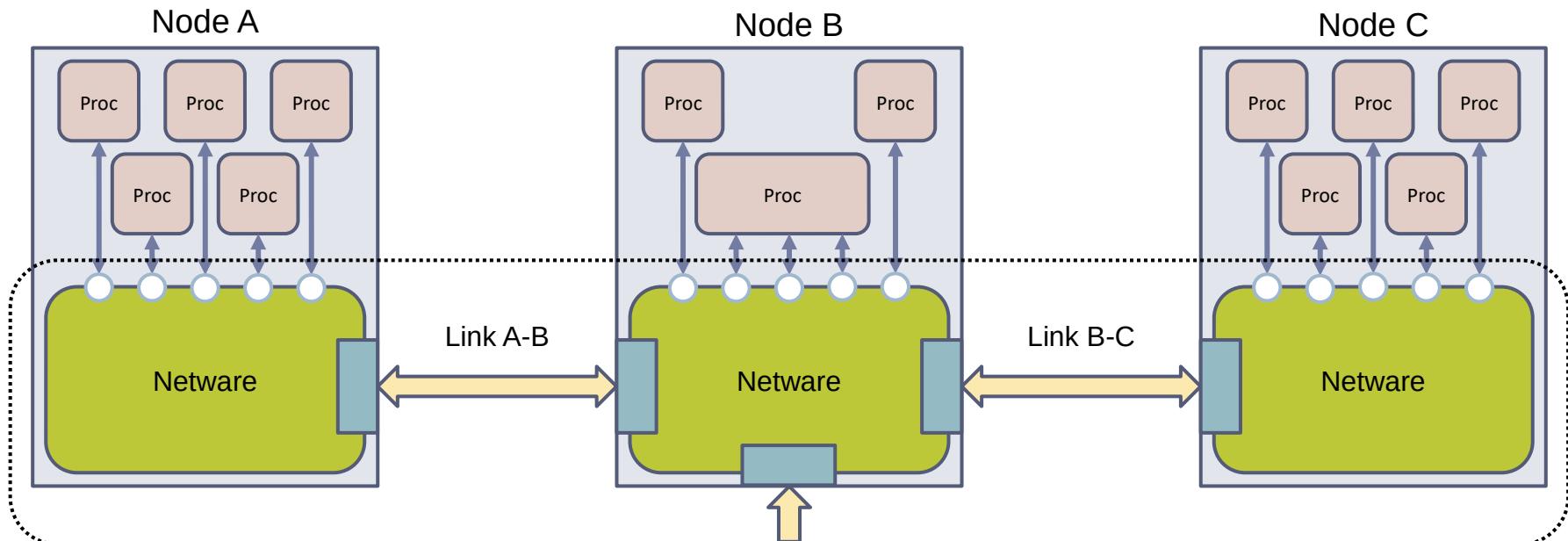


Логическая организация сетевого сервиса

Потребители сетевых услуг — разработчики ПО, использующего сетевой API, рассматривают сеть целиком как некий «черный ящик» предоставляющий услугу (service) по транспортировке данных между точками доступа к услугам (service access points – SAP) независимо от того, на каком узле сети они располагаются. При этом все детали реализации остаются скрытыми внутри черного ящика



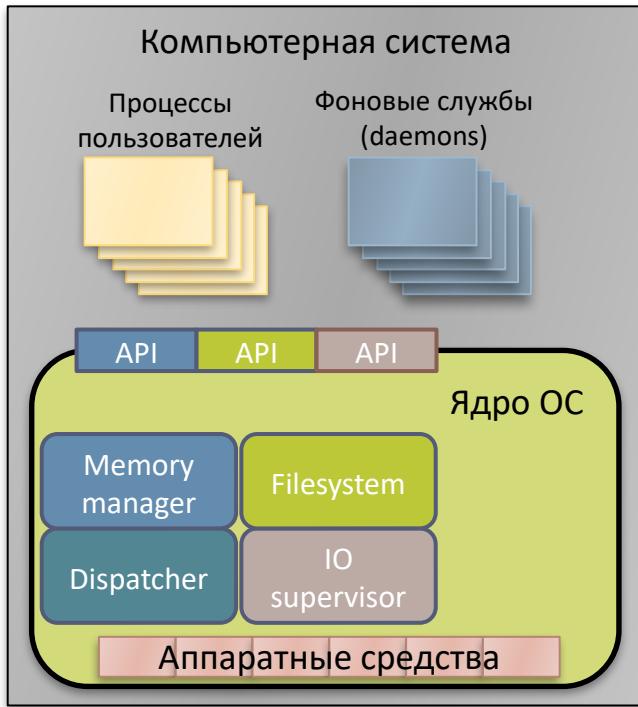
Внутренняя структура «чёрного ящика»



- ✓ «чёрный ящик» сетевого сервиса реализуется совокупностью **модулей сетевого обеспечения** узлов, взаимодействующих друг с другом через **общие звенья передачи данных**
- ✓ как правило, **программные модули Netware обслуживают Endpoint-ы своего узла**, а также выполняют **функции коммутации поступающих порций трафика**, в том числе и **транзит между сетевыми интерфейсами**
- ✓ низкоуровневые процессы **приёма и передачи данных** через **звеня**, реализуются **аппаратными адаптерами сетевых интерфейсов** в **асинхронном режиме**

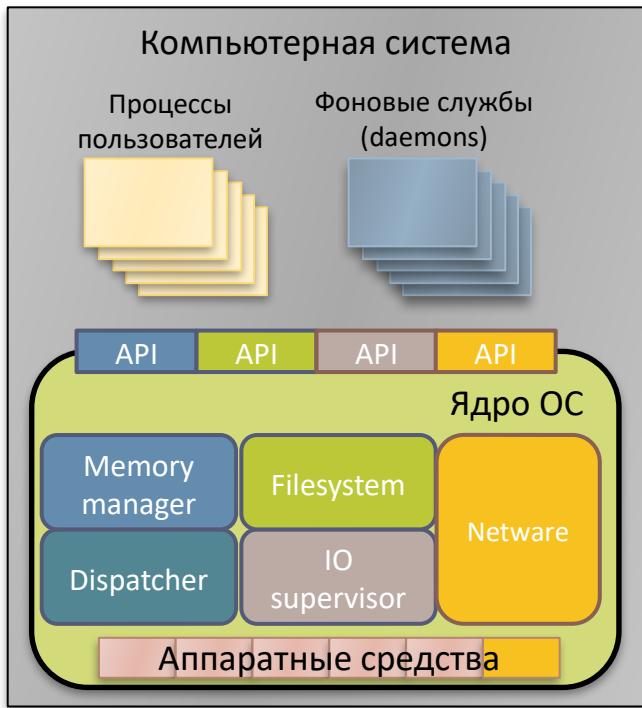


Архитектура компьютерных систем



- ✓ **Процесс** – единица работы в компьютерной системе. Представляет собой набор страниц памяти и один или несколько потоков выполнения инструкций. В системах функционируют пользовательские и фоновые процессы
- ✓ **Ядро ОС** blokiрует прямой доступ процессов к аппаратным средствам системы. Вместо этого **ядро** предоставляет более удобный **высокоуровневый сервис** для работы с **логическими ресурсами системы**, доступный через **API**
- ✓ **API** (application programming interface – интерфейс программирования приложений): **набор программно-доступных объектов** (функций, классов, методов, констант и т.п.) и соглашений по их использованию

Для узла сети — просто добавь Netware



- ✓ Как правило, в качестве **узлов сети** выступают обычные **компьютерные системы**: как специализированные, так и общего назначения (серверы, ПК, смартфоны...)
- ✓ Для реализации **сетевой функциональности** в их архитектуру дополнительно включаются **средства сетевого обеспечения**:
 - аппаратные компоненты (**hardware**) адаптеров сетевых интерфейсов и др.
 - **сетевое программное обеспечение (netware)** в составе ядра ОС
- ✓ **Процессы ОС** посредством **сетевого API** имеют возможность организовать **точки доступа к услугам сети (SAP)** и воспользоваться сетевым **сервисом по транспортировке данных** между конечными точками (**Endpoint-ами**) внутри процессов
- ✓ Вся **деятельность по доставке данных** через **узлы и звенья сети** до адресатов **выполняется модулями Netware** без участия **процессов ОС**



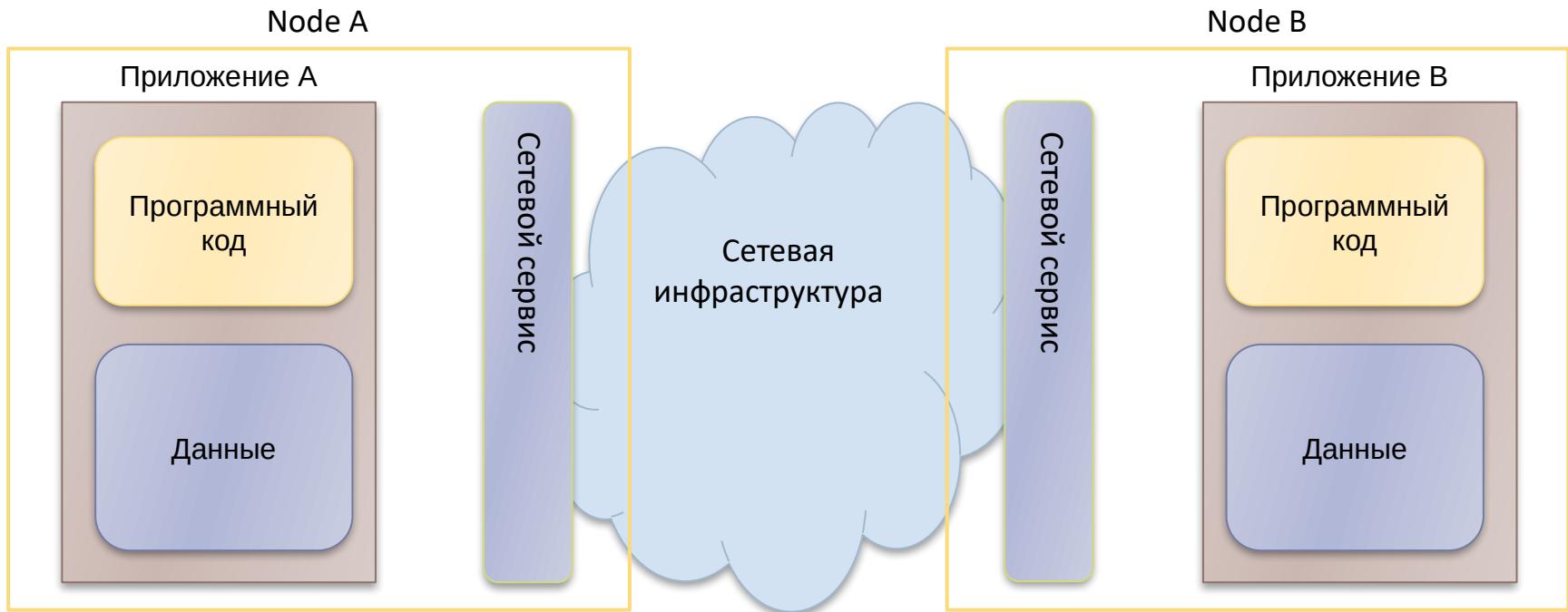
Что такое SAP

- ✓ SAP это логический объект ОС, напоминающий файловый дескриптор. Процессы ОС могут динамически создавать и закрывать SAP посредством сетевого API ОС
- ✓ SAP «олицетворяют» (represent) Endpoint-ы в сети при транспортировке трафика между ними:
 - порции трафика поступают в сеть через SAP процесса отправителя
 - сеть доставляет порции трафика через SAP процесса получателя
- ✓ Для разных применений, ОС может реализовывать несколько методов транспортировки трафика по сети через SAP соответствующего типа:
 - дейтаграммный: независимая транспортировка отдельных (обособленных) порций данных ограниченного размера
 - потокоориентированный: транспортировка (потенциально бесконечного) упорядоченного потока байтов и др.
- ✓ Коммутация трафика на транзитных узлах осуществляется без участия прикладных процессов ОС и поэтому наличие SAP (на этих узлах для транспортировки транзитного трафика) не требуется
- ✓ На уровне сети каждому действующему SAP присваивается уникальный адрес. Обычно он формируется из трёх компонентов:

<Тип SAP> : <Сетевой адрес узла/интерфейса> : <Номер порта>



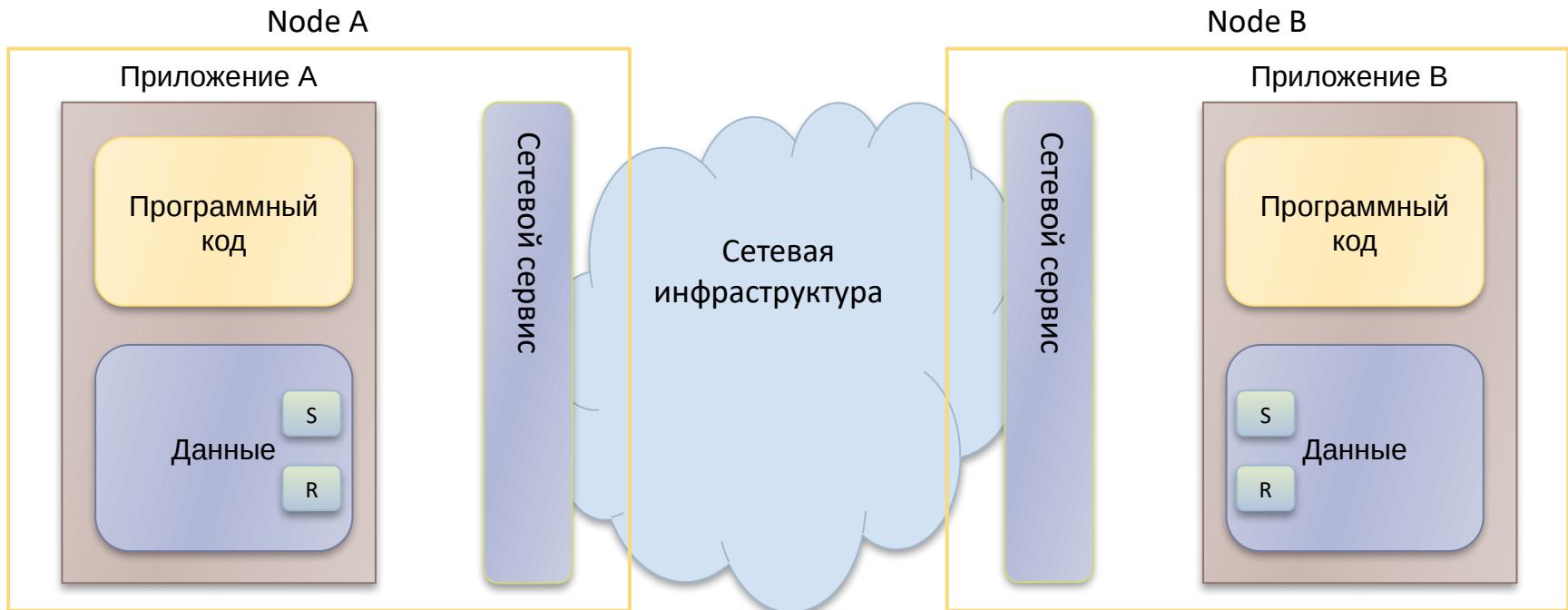
Процедура обмена данными через сеть



- ✓ Взаимодействующие **приложения** функционируют в виде процессов ОС на узлах сети, где развернут **сетевой сервис** и имеются **подключения к сетевой инфраструктуре**, обладающей связностью
- ✓ В каждом **приложении** исполняется **программный код**, который манипулирует **данными в памяти** своих **процессов ОС** и имеет возможность вызывать **методы API** сетевого сервиса



Процедура обмена данными через сеть



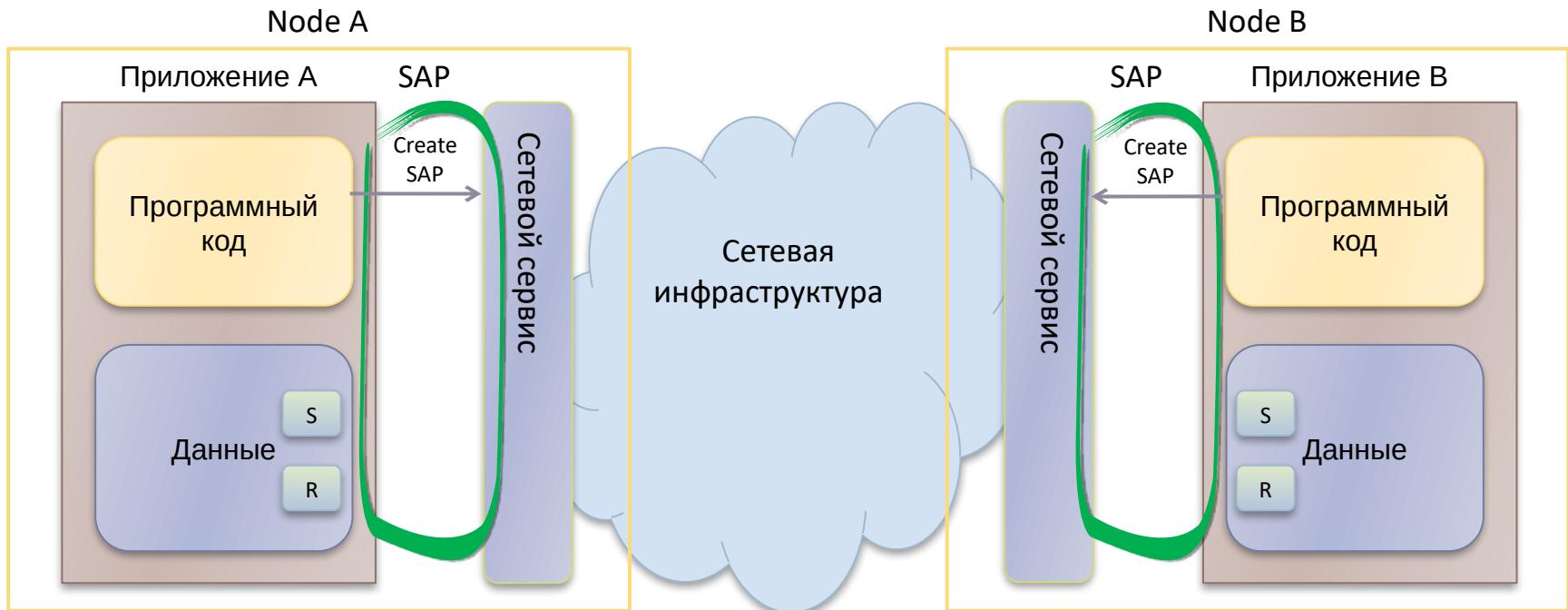
Для **обмена данными по сети** каждое из **приложений** организует в своей памяти **буферы данных**:

- буферы передачи **S**, куда **код приложения** должен скопировать данные, которые **сетевой сервис** будет забирать для передачи по сети
- буферы приема **R**, куда **сетевой сервис** будет копировать принимаемые по сети данные и откуда **код приложения** сможет их забрать для обработки

По сути, эти **буферы** и **выступают в роли Endpoint-ов** при доставке трафика по сети



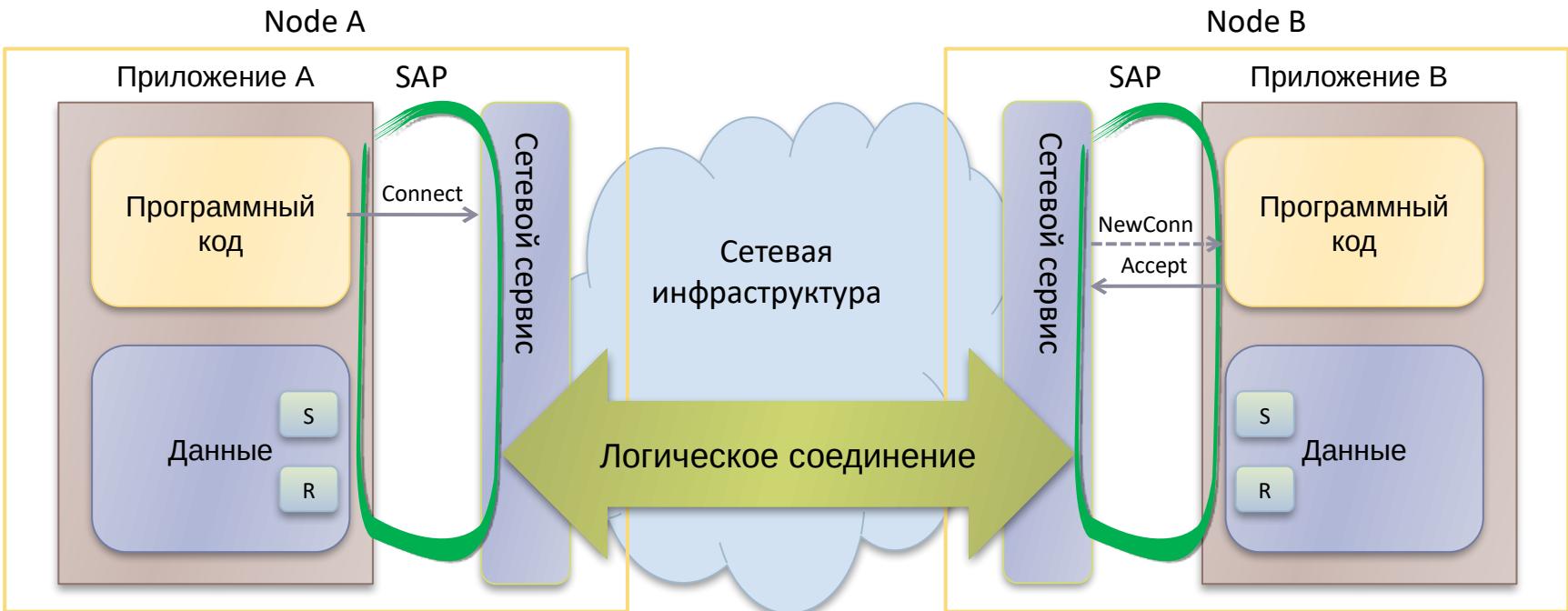
Процедура обмена данными через сеть



- ✓ Программный код приложений через сетевой API запрашивает создание SAP. Сетевой сервис по этому запросу регистрирует логические Endpoint-ы в рамках своего узла
- ✓ При создании SAP им присваивается адрес Endpoint в сети, включающий:
 - уникальный (в пределах сети) адрес, присвоенный данному узлу/интерфейсу
 - номер порта, который сейчас не назначен другому SAP внутри данного узла



Процедура обмена данными через сеть



Программный код одного из приложений (A) инициирует установку логического соединения между SAP:

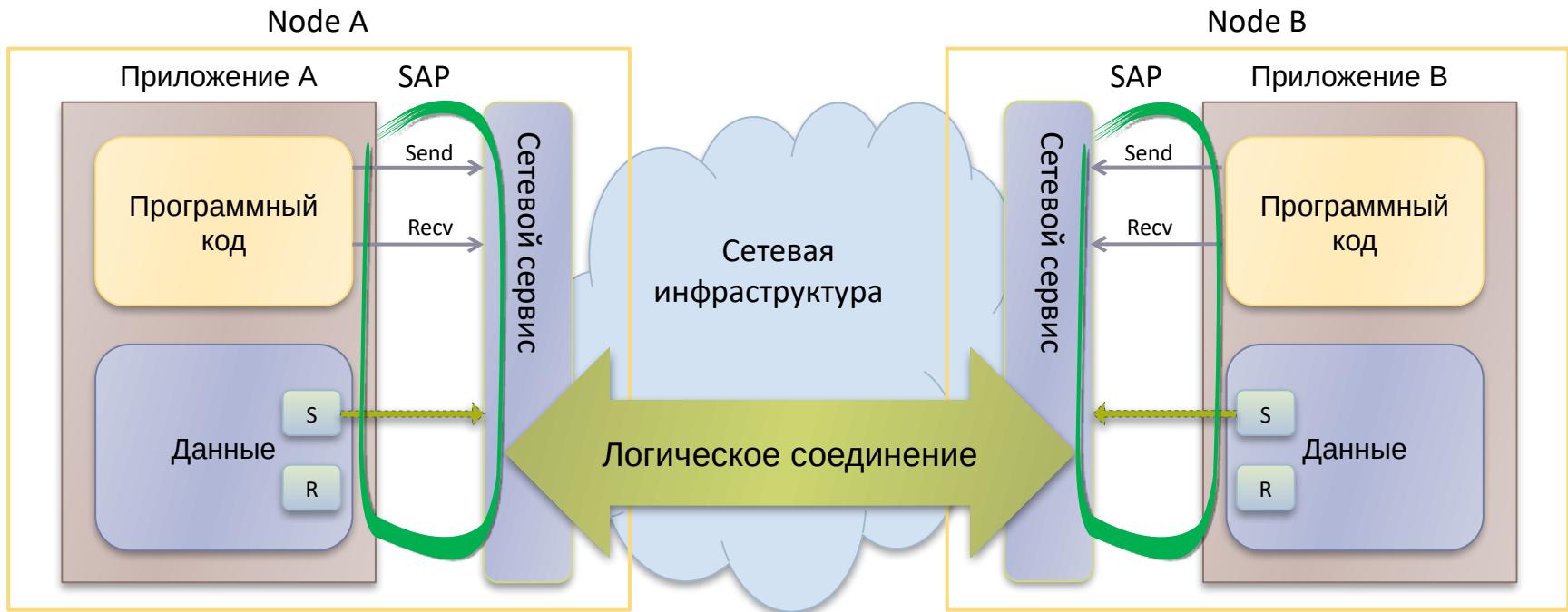
- Запрос **Connect(addr)** – установить логическое соединение с Endpoint, адрес которого указан в параметре **(addr)**

Программный код второго приложения (B):

- получает индикацию **NewConn** – запрос установки нового соединения
- вызывает функцию **Accept** – принять новое логическое соединение



Процедура обмена данными через сеть



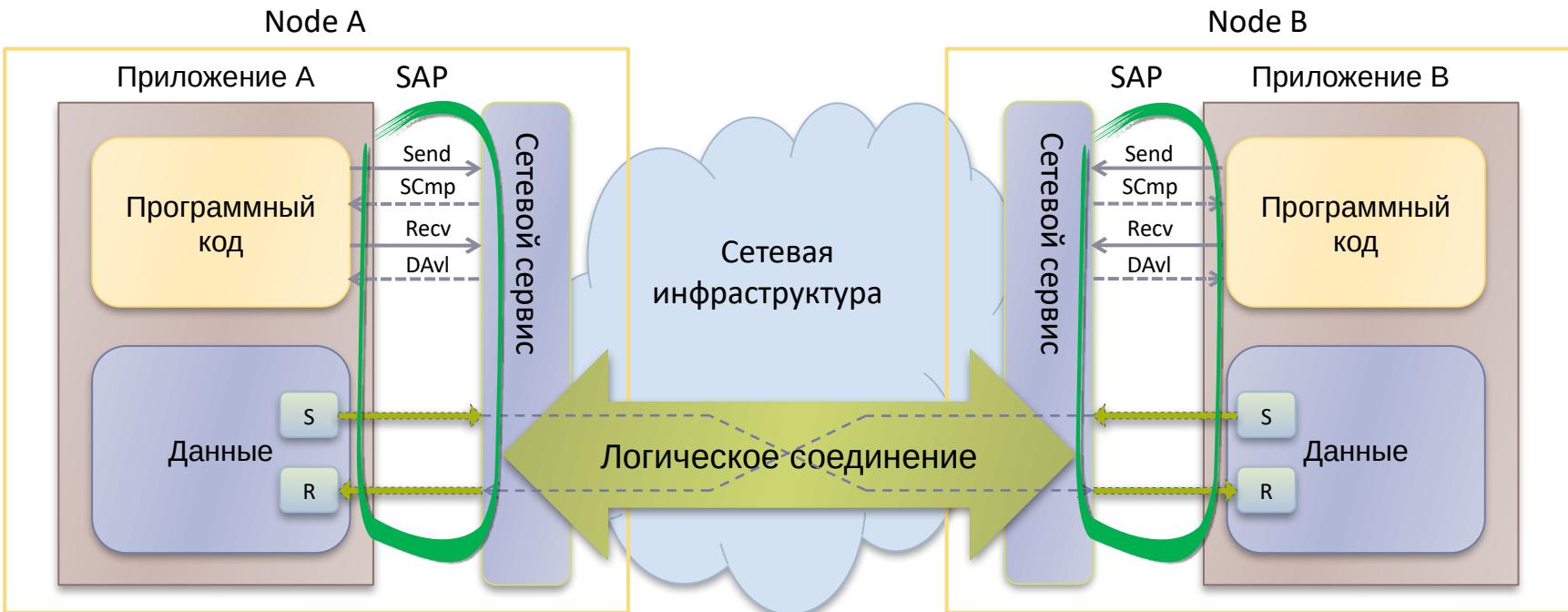
Программный код приложений через SAP вызывает функции API:

- Запрос **Send** – отправить данные из буфера **S** партнеру по соединению
- Запрос **Recv** (**receive**) – поместить в буфер **R** данные, принятые от партнера

Сетевой сервис узлов А и В, а также транзитных узлов по маршруту, выполняет в асинхронном режиме транспортировку (копирование) данных по сети из буферов передачи отправителя в буферы приёма получателя. Этот процесс происходит без участия программного кода приложений



Процедура обмена данными через сеть



По завершению копирования данных **сетевой сервис** через **механизмы ОС оповещает процессы приложений** своего узла о завершении ранее запрошенных действий:

- Индикация SCmp (**send complete**) – ранее отправленные данные переданы
- Индикация DAvl (**data available**) - в буфере R доступны новые принятые данные

Сетевой сервис на уровне API (функций стандартной библиотеки sockets)

`sock_hdl = socket (...)` – создать SAP и розетку (socket) для доступа к ней

- `sock_hdl` – будет содержать `socket handle` («ручку от розетки»), которая соответствует созданной в ОС точке доступа к сетевым услугам (SAP)

`connect (sock_hdl, dst_addr ...)` – соединить данную розетку с другой SAP

- `dst_addr` – адрес буфера в памяти приложения, где записаны `сетевые координаты вызываемой SAP` (что-то вроде набираемого номера телефона)

`send (sock_hdl, buf, len ...)` – отправить порцию данных

- `buf` – адрес первого отправляемого байта в памяти приложения
- `len` – количество отправляемых байт

`recv (sock_hdl, buf, len ...)` – запросить `получение` порции принятых данных

- `buf` – адрес приемного буфера в памяти приложения куда должна будет записана порция принятых данных
- `len` – размер приемного буфера в байтах (максимальный размер получаемой порции)



План курса

1. Введение в компьютерные сети

2. Основные методы построения СПД

2.1. Методы коммутации и мультиплексирования

2.2. Программная модель сетевого взаимодействия

2.4. Общие принципы построения netware (сетевого обеспечения)

2.5. Функциональная совместимость netware и стандарты

3. Архитектура Internet Protocol Suite (TCP/IP)

4. Архитектура модулей физического уровня

5. Технологии беспроводных сетей

6. Архитектура модулей канального уровня

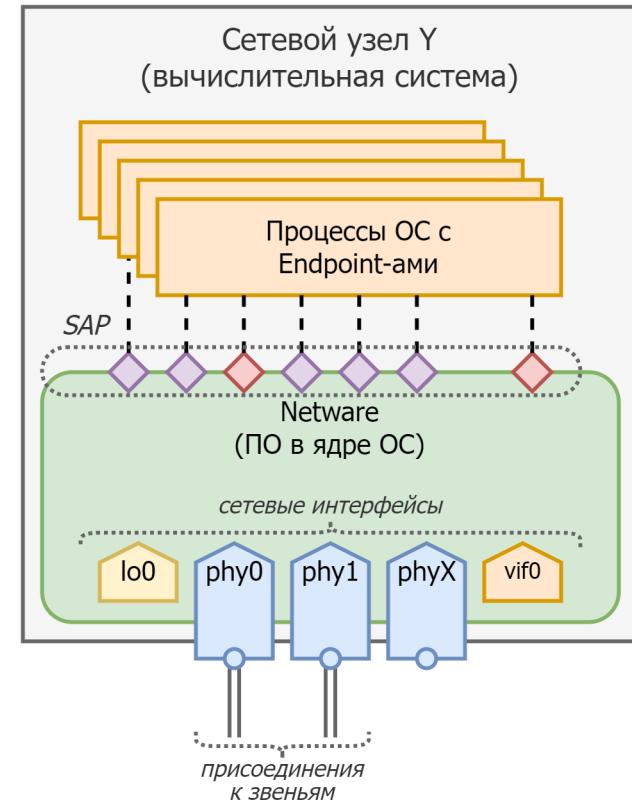
7. Протоколы транспортного уровня

8. Технологии WWW



Логическая архитектура узла сети

- ✓ в составе ядра ОС каждого узла сети функционирует модуль Netware (min один экземпляр...)
- ✓ в соответствии с топологией сети в модуле Netware конфигурируется необходимое количество сетевых интерфейсов, среди которых:
 - ➔ аппаратные сетевые адAPTERы с помощью которых узел может присоединяться к физическим звеньям передачи данных
 - ➔ внутренние логические интерфейсы loopback
 - ➔ разновидности виртуальных интерфейсов, позволяющих программно формировать и/или обрабатывать трафик
- ✓ состав и конфигурация интерфейсов, при необходимости, может динамически изменяться
- ✓ с помощью сетевого API процессы ОС могут динамически создавать и управлять SAP через которые netware реализует доставку сетевого трафика из/в Endpoint-ы процессов



Принципы функционирования Netware

- ✓ Netware получает/формирует и размещает в буферной памяти входящие порции трафика (пакеты, кадры, сегменты...):
 - принятые через сетевые интерфейсы (как правило, от соседних узлов)
 - сформированные из данных Endpoint-ов (этого узла)
- ✓ Netware осуществляет необходимую обработку входящих порций (контроль, преобразование и др.) непосредственно в буферной памяти, а затем, в соответствии с настройками и на основании информации, содержащейся в служебных заголовках, выбирает один из вариантов коммутации данной порции:
 - поставить в очередь передачи определённого интерфейса
 - скопировать в буфер приёма соответствующего Endpoint-а
 - удалить (ошибочную) поцию, возможно, с формированием отчёта об ошибке
- ✓ Netware это часть ядра ОС, которая функционирует не в виде какого-либо процесса, а получает управление на короткое время при определённых событиях:
 - при вызове процессами ОС методов сетевого API
 - при обработке прерываний от сетевых адаптеров
 - при обнаружении таймаутов в процессе ожидания сетевых взаимодействий

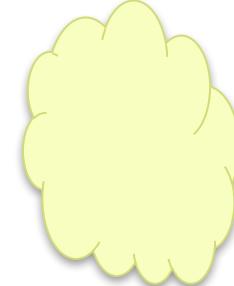
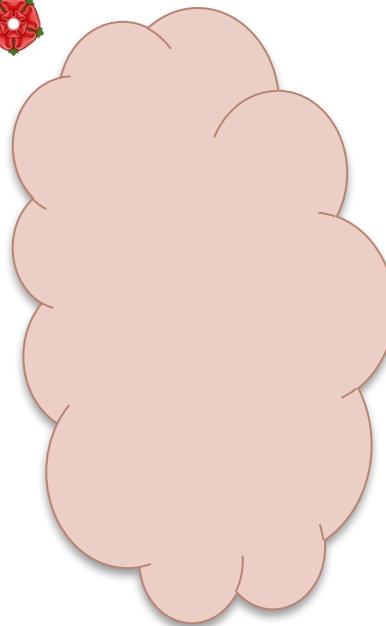
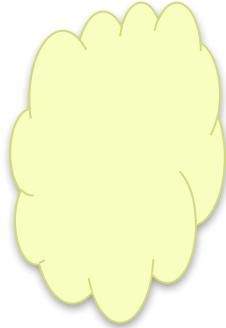
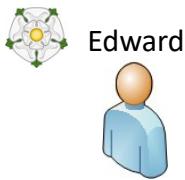


Проблемы при реализации модулей netware

- ✓ Модули netware на разных узлах сети, с одной стороны функционируют относительно самостоятельно в рамках своих узлов, а с другой, должны действовать скоординированно и согласованно друг с другом в процессе транспортировки трафика в масштабе всей сети. Они относятся к классу распределённых алгоритмов (протоколов) и требуют особого подхода при разработке.
- ✓ Функциональная совместимость (interoperability) модулей netware друг с другом. Т.е. способность netware разных компьютерных архитектур (Intel, ARM, MIPS и т.п.), разных операционных систем (Windows, Linux, Android, iOS и т.п.), написанных разными программистами в разное время успешно функционировать вместе в составе одной сети.
- ✓ Модули netware должны реализовывать достаточно большой объём сложной функциональности который сильно зависит от конфигурации сетевых узлов, типов применяемых звеньев, выбранных режимов настройки и др. Это требует применения модульного подхода «конструктора Lego», когда требуемый функционал собирается из отдельных взаизаменяемых модулей.



Понятие о распределенных алгоритмах

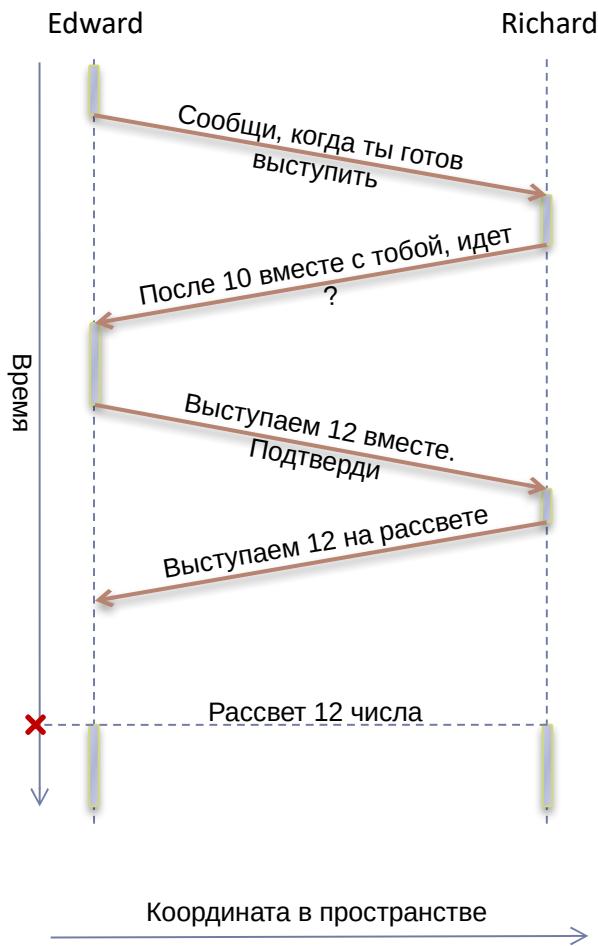


- Участвуют несколько **актеров**, обладающих собственной **способностью действовать**
- Имеется цель/задача, требующая **совместных** скоординированных **действий**

- Актеры самостоятельно **не могут «знать»** что делают другие актеры
- Для координации действий актерам **необходимо вести диалог (conversation)** друг с другом



Изображение диалога



Диалог (**conversation**) – процесс обмена **сообщениями** между актерами в рамках какого-либо распределенного алгоритма.

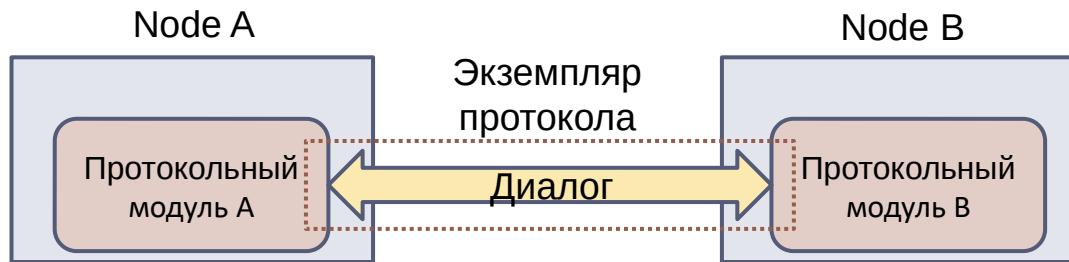
Сообщение (**message**) – законченная порция информации, передаваемая от актера к актеру, имеющая смысл в контексте данного диалога.

Локальное действие – последовательность действий, самостоятельно совершаемых актером, как правило, **по наступлению определенного события**: начального запуска диалога, прихода сообщения, наступления определенного времени или истечения таймаута.

Таймаут (**timeout**) – лимит времени в течение которого актер ожидает наступления определенного события, например ответа

Понятие протокола

Компьютерная сеть, вследствие своей распределённой в пространстве (distributed) природы – это «среда обитания» множества **распределенных алгоритмов**, называемых **протоколами (protocols)**.



Протокольный модуль – программный (software) или, реже, аппаратный (hardware) компонент, выступающий в качестве **актера** в рамках некоторого **распределенного алгоритма**: WEB-browser, Telegram-клиент, точка доступа WiFi и т.п.

Способность вести диалог, т.е. принимать и отправлять сообщения, относящиеся к определенному виду диалогов, является **неотъемлемой чертой протокольных модулей**.

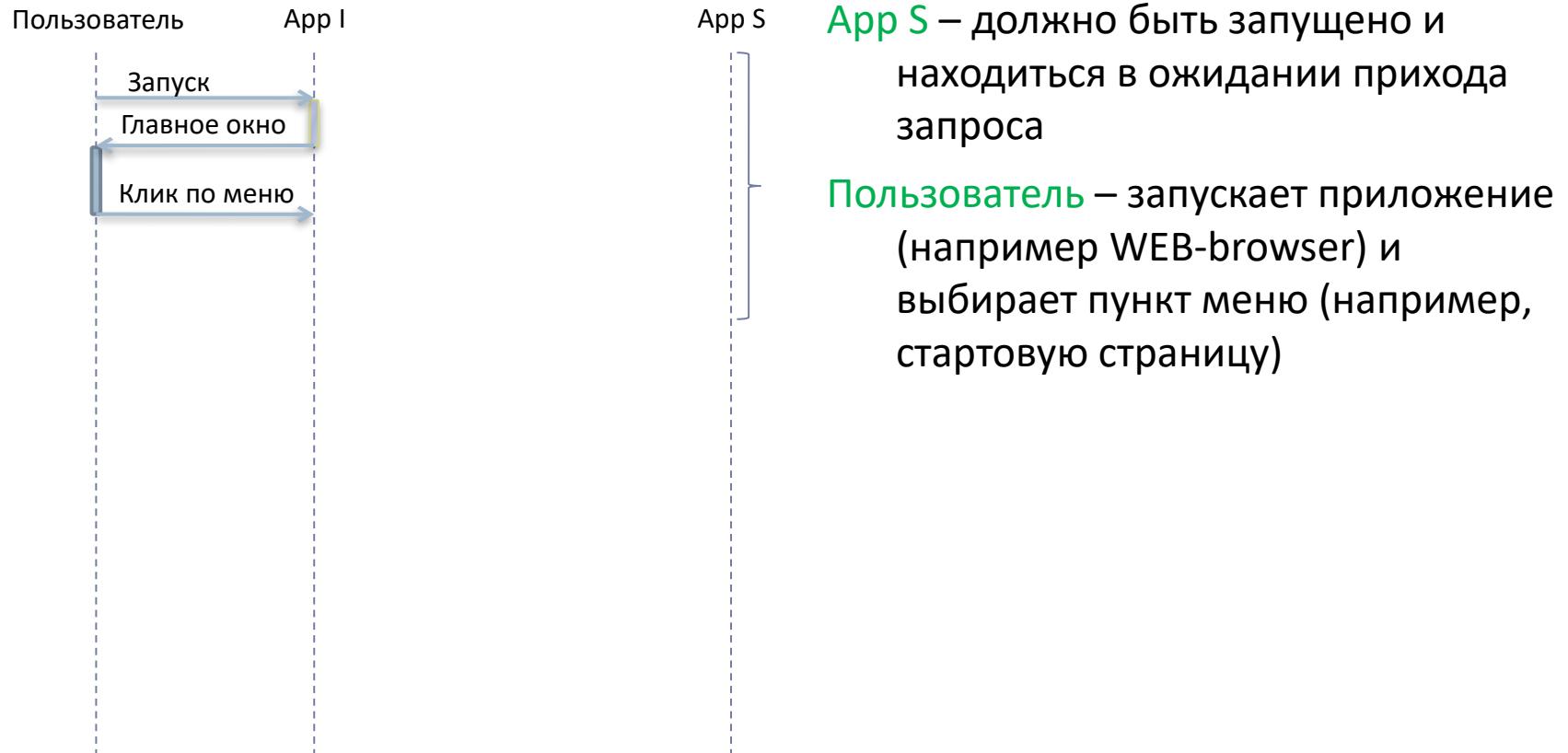
Сообщения, передаваемые в рамках диалога, называются **PDU (protocol data units)**



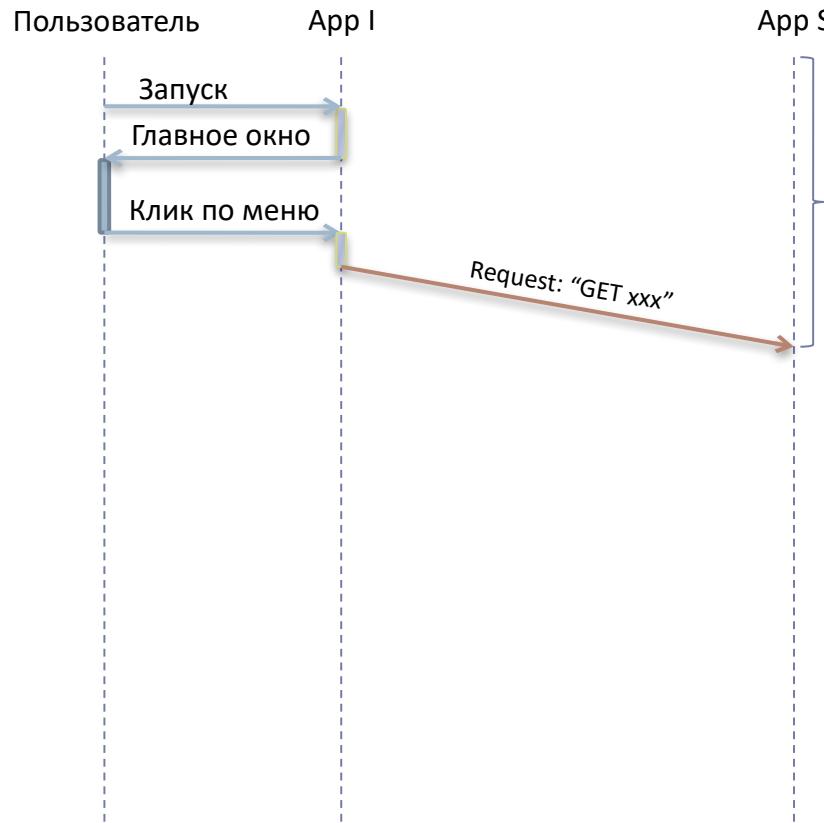
Пример функционирования протокола



Пример функционирования протокола



Пример функционирования протокола



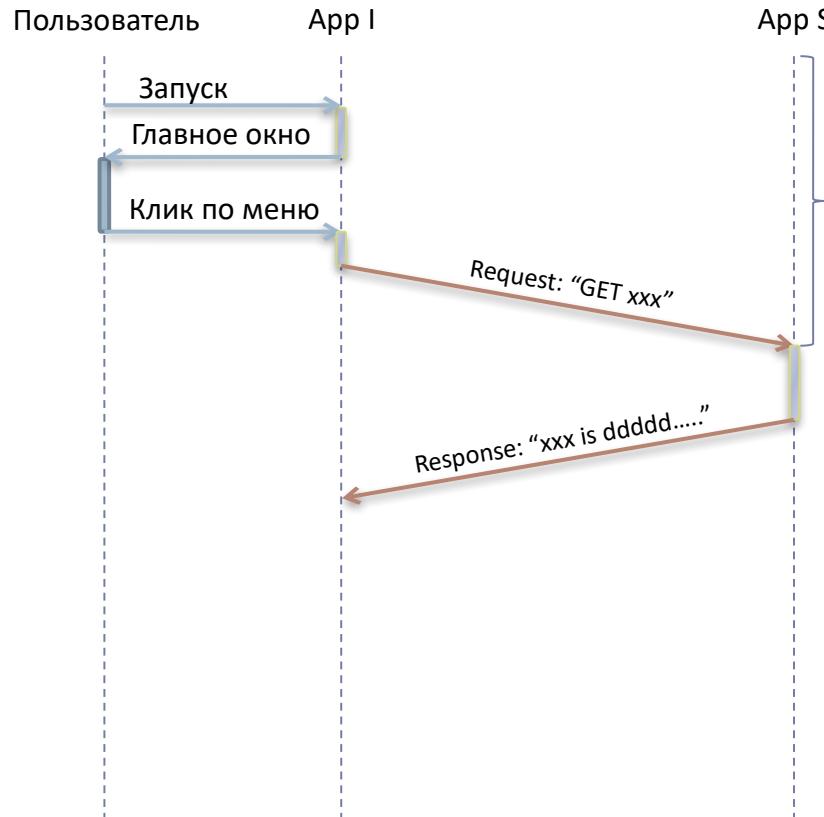
App S – должно быть запущено и находится в ожидании прихода запроса

Пользователь – запускает приложение (например WEB-browser) и выбирает пункт меню (например, стартовую страницу)

App I – начинает (инициирует диалог), отправляя сообщение-запрос (**request**) и ожидает ответа



Пример функционирования протокола



App S – должно быть запущено и находится в ожидании прихода запроса

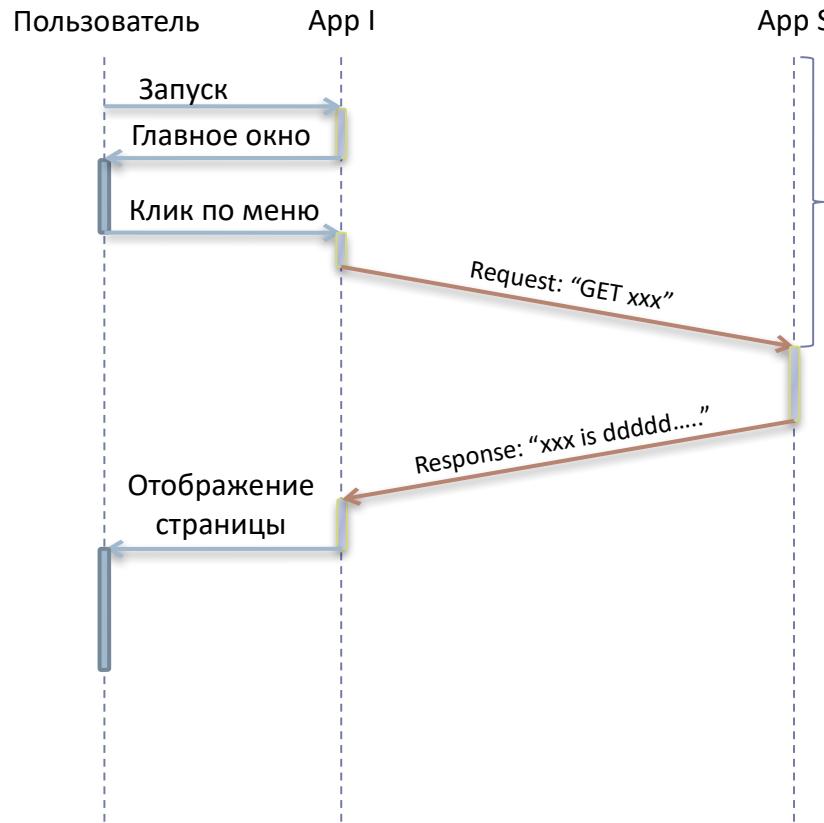
Пользователь – запускает приложение (например WEB-browser) и выбирает пункт меню (например, стартовую страницу)

App I – начинает (инициирует диалог), отправляя сообщение-запрос (**request**) и ожидает ответа

App S – обрабатывает запрос и в результате формирует и направляет ответ (**response**)



Пример функционирования протокола



App S – должно быть запущено и находится в ожидании прихода запроса

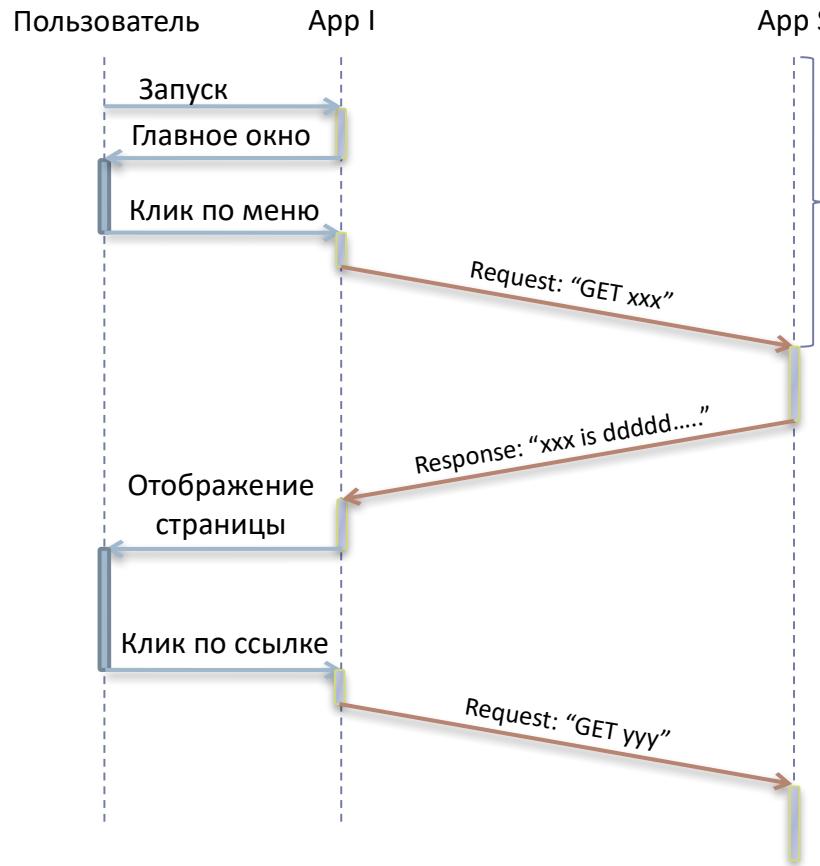
Пользователь – запускает приложение (например WEB-browser) и выбирает пункт меню (например, стартовую страницу)

App I – начинает (инициирует диалог), отправляя сообщение-запрос (**request**) и ожидает ответа

App S – обрабатывает запрос и в результате формирует и направляет ответ (**response**)

App I – получив ответ, отображает его на экране и переходит в ожидание дальнейших действий пользователя

Пример функционирования протокола



App S – должно быть запущено и находится в ожидании прихода запроса

Пользователь – запускает приложение (например WEB-browser) и выбирает пункт меню (например, стартовую страницу)

App I – начинает (инициирует диалог), отправляя сообщение-запрос (**request**) и ожидает ответа

App S – обрабатывает запрос и в результате формирует и направляет ответ (**response**)

App I – получив ответ, отображает его на экране и переходит в ожидание дальнейших действий пользователя

Роли модулей в рамках протокола

Элемент поведения	App I	App S
Как вступает в диалог	Инициирует (начинает) диалог в любой момент времени	Только в ответ на запрос
Что делает в рамках диалога	Может выдавать любые корректные запросы	Отвечает лишь на присланные запросы
Момент запуска и период работы приложения	Может быть запущено пользователем при необходимости и остановлено в любой момент	Должно быть запущено заранее и функционировать все время пока существует необходимость обработки запросов
Наименование роли в протоколе	<u>Клиент (Client)</u>	<u>Сервер (Server)</u>

Клиент и сервер это наименование ролей (шаблонов поведения), которые характерны для модулей в рамках протокола



Модель клиент-сервер

Многие из существующих протоколов являются **клиент-серверными**, т.е. протокольные **модули** четко делятся **по поведению** на **клиенты и серверы** (но встречаются и иные модели, только значительно реже).

Термин **«сервер»** (буквально с английского – тот, который обслуживает) может обозначать разные понятия:

- ✓ **Мощный компьютер**, предназначенный для размещения на нем постоянно функционирующих приложений. Например, сервер HP DL580 G12.
- ✓ **Сетевая служба**, представляющая собой постоянно функционирующий модуль программного обеспечения, предоставляющий **сервис определенного вида** доступный по сети для других узлов (клиентов). Например, файловый сервер, WEB-сервер, игровой сервер, сервер печати и т.п.
- ✓ Наименование **роли в протоколе**.

Все три значения тесно связаны, что может вызывать путаницу: например, **файловый сервер, являющийся сервером SMB/CIFS функционирует на сервере SuperMicro 8026B-6RF**



Смысл слова «протокол»

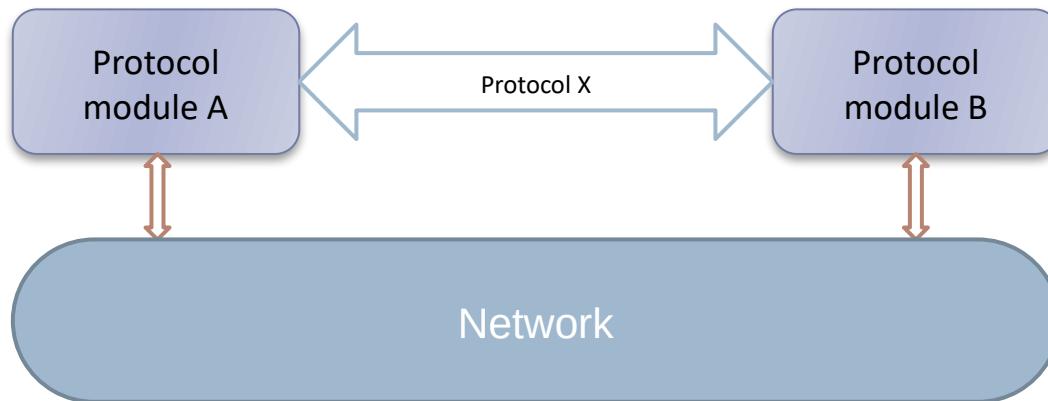


- **Protocol specification:** документ, описывающий детали протокола
- **Protocol implementation:** программный модуль, реализующий функциональность протокольного модуля в соответствии со спецификацией
- **Protocol instance:** функционирующий экземпляр – результат каждой активации (запуска) ПО реализующего протокольный модуль

Понятие функциональной совместимости

Узел: MyPad
CPU: ARM
ОС: Android 7.1
Язык программир.: Java
Программист: J.Smith

Узел: SuperServer
CPU: Intel Xeon x64
ОС: Windows 2012 R2
Язык программир.: C++
Программист: B.Gates



Модуль А и модуль В **функционально совместимы (interoperable)** в том случае, если они могут взаимодействовать (выступать в роли актеров и понимать друг друга) **в** рамках некоторого протокола Х (распределенного алгоритма разновидности Х).

Каким образом можно этого достичь ?



Как достичь функциональной совместимости

Вариант 1 – все модули **разрабатываются одним программистом** →

С функциональной совместимостью модулей нет проблем (теоретически). Все детали, необходимые для реализации, «в одной голове».

Вариант 2 - разные модули **совместно разрабатываются разными людьми**

(например в одной группе) → Участники группы должны договориться друг с другом о правилах ведения диалога между модулями. В процессе разработки детали могут уточняться, а правила ведения диалога изменяться (адаптироваться) по согласованию.

Вариант 3 – модули **разрабатываются независимо, разными людьми**, в разное время. Прямое общение невозможно → Каждому разработчику

перед началом разработки модулей необходимо иметь полную и точную информацию о правилах ведения диалога →

Необходимо иметь документ – спецификацию протокола



Что нужно указать в спецификации протокола

Спецификация протокола – это формализованное, однозначно понимаемое описание правил ведения диалога определенного вида

Аспект	Что определяется	Пример
Синтаксис	Формат сообщений и структура передаваемых элементов данных	Алфавит, лексика и грамматика языка, на котором разговаривают актеры
Семантика	Значение (смысл) элементов данных	Значения слов и метод интерпретации грамматических конструкций языка на котором разговаривают актеры
Процедура диалога	Порядок обмена сообщениями, ожидаемые ответы, подразумеваемые локальные действия, правила ожидания реакции партнера	Правила, описывающие ведение диалога: кто начинает диалог, кто и как продолжает и в ответ на что, кто ждет, что делает по получению сообщений и т.п.



Откуда появляются стандарты

Стандарты де-факто (de-facto standards)

Это некие **технологические традиции**, описываемыми не формальными **нормативными спецификациями**, а широко известным **know-how**, позволяющими добиваться совместимости независимо выпускаемой продукции

Частные стандарты (proprietary standards)

(Обычно) **формальные нормативные документы**, которые не могут быть использованы без согласия **правообладателя** (спецификации держатся в тайне и/или используется защита патентами). Обычно правообладатель (владелец интеллектуальной собственности) либо единолично пользуется исключительным правом выпускать продукцию, в соответствии со своим стандартом, либо собирает **лицензионные отчисления (royalties)**.

Открытые стандарты (open standards)

Имеют правовой статус, позволяющий использовать их любому заинтересованному лицу. Обычно, в качестве **автора правообладателя** открытых стандартов выступает нейтральная некоммерческая организация – **орган стандартизации (Standard Body)**, которая поддерживает **открытый юридический статус документов** и способствует широкому применению выпускаемых ею стандартов. Текст открытых стандартов, обычно, публично доступен (бесплатно или за умеренную фиксированную плату в пользу standard body).



Некоторые важные standard bodies

ISO (International Standardization Organization)

Международная организация по стандартизации. Стандарты «на все случаи жизни», например [ISO 8538:1999 Aerospace – Nuts hexagonal, self-locking, with counterbore and captive washer...](#)

CEN (фр. Comite Europeen de Normalisation)

Региональная организация по стандартизации Евросоюза. Разрабатывает универсальные стандарты серии EN, например, [EN 13537:2002 Requirements for Sleeping Bags](#)

Росстандарт (Федеральное агентство по техническому регулированию и метрологии)

Национальная служба по стандартизации РФ. Универсальные стандарты, например, [ГОСТ Р 50597-93 Автомобильные дороги и улицы. Требования...](#)

ITU (International Telecommunication Union)

Международный союз электросвязи – отраслевая организация, выпускающая специализированные стандарты для связи, например [ITU-T E.164 – Международный план нумерации электросвязи общего пользования](#)

IEEE (Institute of Electrical and Electronics Engineers)

Международная некоммерческая ассоциация по разработке стандартов в области радиоэлектроники, электротехники. Например [IEEE 802.1AX-2008 Local and metropolitan area networks. Link Aggregation](#)

IETF (Internet Engineering Task Force)

Открытое международное сообщество сетевых специалистов (разработчиков, проектировщиков, инженеров, операторов, поставщиков оборудования, исследователей), заинтересованных в эволюции Internet архитектуры и бесперебойном функционировании сетей Internet. Выпускает рекомендации, например [RFC 793 Transmission Control Protocol](#)

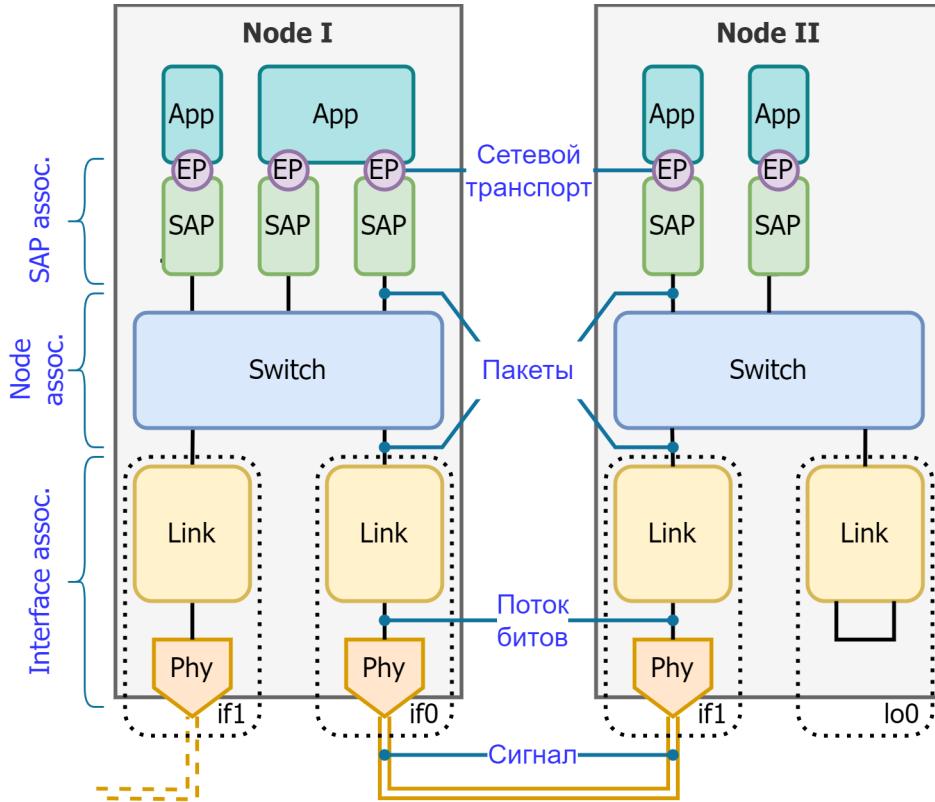


Особенности вокруг сетевых стандартов

- ✓ компьютерные сети это **универсальный механизм коммуникации** в интересах **потребителей**: чем шире охват сетью потенциальных потребителей, тем большую ценность **такая сеть** способна принести
- ✓ функциональная совместимость (**interoperability**) это **наи важнейшая характеристика** сетевого обеспечения (**netware**), позволяющая строить **универсальные гетерогенные сети** произвольного масштаба в интересах широкого круга потребителей (но до этой простой идеи человечество дошло не сразу)
- ✓ **сетевые стандарты** это **ключевой инструмент** для достижения **interoperability** различных вариантов реализации сетевого обеспечения (*и software, и hardware*)
- ✓ возможность **независимой разработки** (разными командами) **компонентов netware** эффективно стимулирует развитие **компьютерных сетей**, поэтому **открытые сетевые стандарты** имеют большое преимущество перед закрытыми
- ✓ большая часть **сетевых стандартов (спецификаций)** описывает различные **протоколы**, однако описываются также **и иные требования**, которые **необходимы для совместимости**: механические (разъёмы), электрические (параметры сигналов), информационные (форматы данных), организационные (процедуры) и др.
- ✓ наблюдаемое **обилие стандартов и протоколов** является прямым следствием **функциональной декомпозиции общего комплекса архитектуры сетевого обеспечения** на отдельные, **более простые элементы — кубики «Lego»**



Структурная декомпозиция функций netware



- ✓ [App] — это функционал, выполняемый в рамках **приложения** (процесса ОС)
- ✓ [EP] — конечный сетевой Endpoint внутри **приложения/процесса**

Структурно, внутри **каждого узла сети** выделяются:

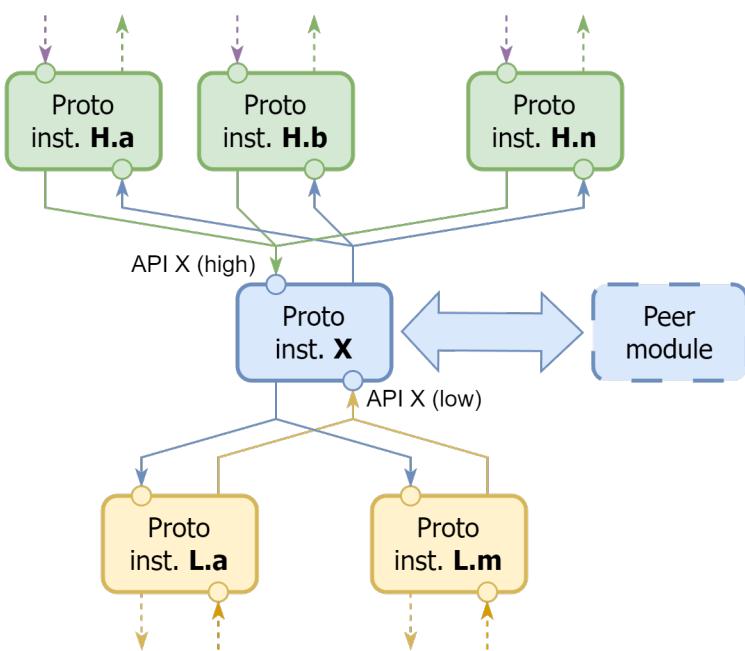
- **логический узел [Node]** — чаще всего один
- **сетевые интерфейсы [If]** — количество переменное, зависит от топологии
- **SAP** — количество динамически определяется **процессами ОС** (много)

С этими элементами **ассоциируются экземпляры функциональных модулей**:

- **[If] Phy** — для каждого физического интерфейса
- **[If] Link** — для каждого физического или логического интерфейса
- **[Node] Switch** — единый для узла
- **[SAP]** — для каждого активного Endpoint-a

Структура модулей и стек протоколов

- Каждый функциональный модуль netware представляет совокупность одного или нескольких protocol instances, которые взаимодействуют друг с другом в пределах своего сетевого узла: вызывают методы API друг друга (для software), либо копируют данные в/из памяти CPU или передают потоки битов (для hardware)
- Протокольные модули выстроены в иерархию (стек), где (почти) у каждого определены нижележащие и вышележащие протокольные модули



- Каждый модуль X ведёт диалог со своим peer модулем в рамках своего протокола для того, чтобы предоставлять сервис вышележащим модулям
- Для отправки сообщений X использует сервис отправки PDU, предоставляемый нижележащим модулем через API (high)
- При получении PDU протокола X нижележащий модуль вызывает модуль X через его API (low)
- Образуется стек вызовов, соответствующий стеку взаимодействующих протоколов и пути данных в нём: inbound, outbound, transit

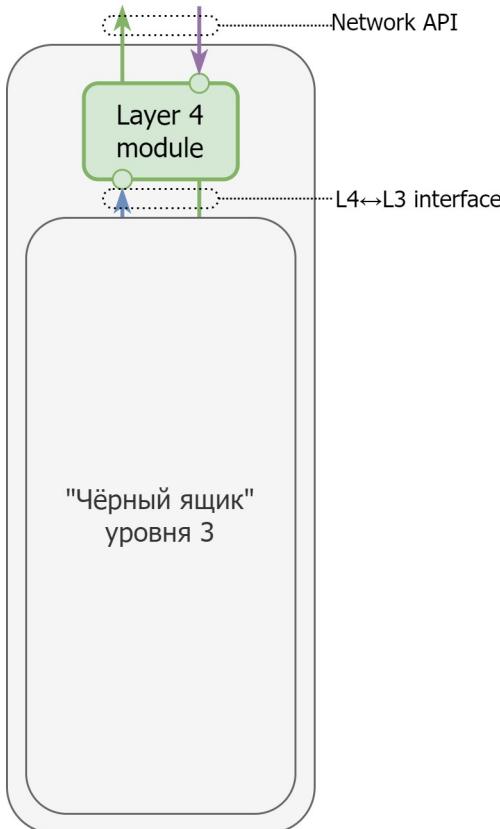
Послойная архитектура стека протоколов



- ✓ Множество функций, которые должны выполняться **netware**, разделено на **функциональные уровни — слои (layers)**. Каждый **уровень** предназначается для реализации **определенного подмножества функций** («*есть слона по кусочкам*»)
- ✓ Каждый уровень предоставляет **сервис по доставке порций данных — PDU**, который доступен **через API** для **вышележащего уровня**
- ✓ При этом **детали реализации этого сервиса скрыты: подход «чёрного ящика»**



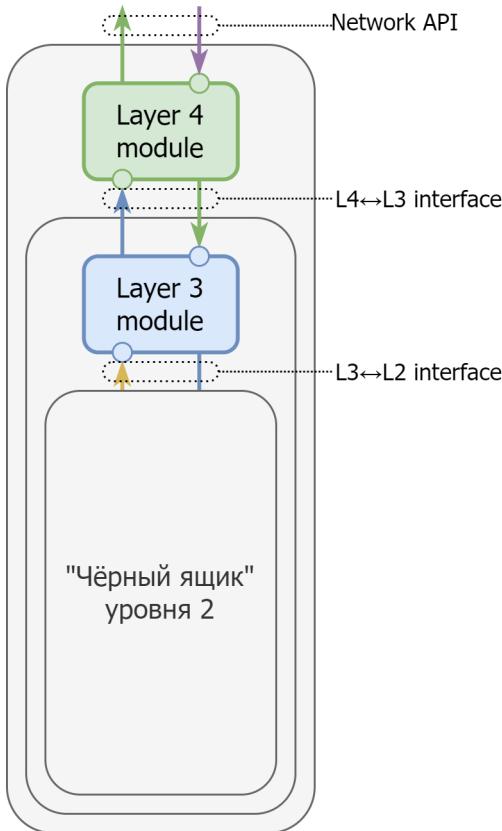
Послойная архитектура стека протоколов



- ✓ Множество функций, которые должны выполняться *netware*, разделено на **функциональные уровни — слои (layers)**. Каждый **уровень** предназначен для реализации **определенного подмножества функций** («*есть слона по кусочкам*»)
- ✓ Каждый уровень предоставляет **сервис по доставке порций данных — PDU**, который доступен **через API** для **вышележащего уровня**
- ✓ При этом **детали реализации этого сервиса скрыты: подход «чёрного ящика»**
- ✓ **«Черные ящики» вложены друг в друга по принципу матрёшки**

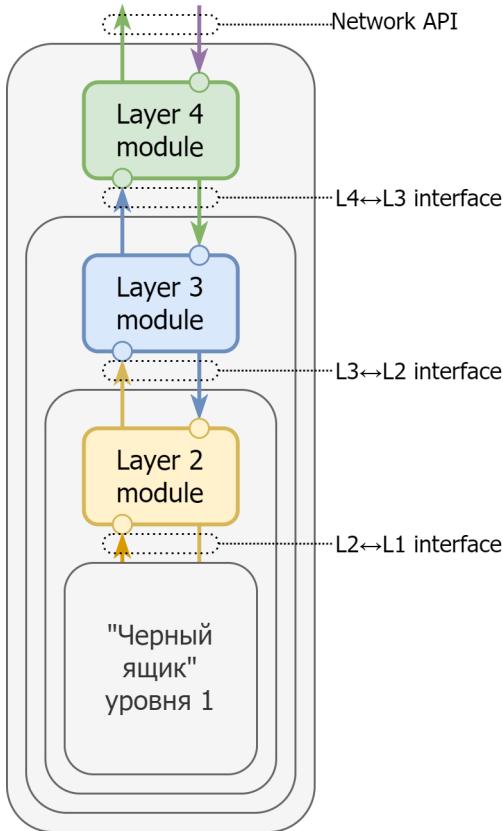


Послойная архитектура стека протоколов



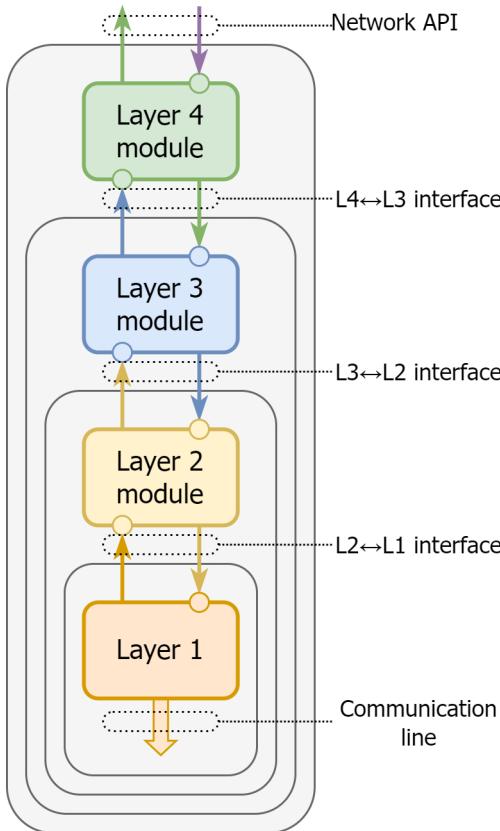
- ✓ Множество функций, которые должны выполняться **netware**, разделено на **функциональные уровни — слои (layers)**. Каждый **уровень** предназначается для реализации **определенного подмножества функций** («*есть слона по кусочкам*»)
- ✓ Каждый уровень предоставляет **сервис по доставке порций данных — PDU**, который доступен **через API** для **вышележащего уровня**
- ✓ При этом **детали реализации этого сервиса скрыты: подход «чёрного ящика»**
- ✓ **«Черные ящики» вложены друг в друга по принципу матрёшки**

Послойная архитектура стека протоколов



- ✓ Множество функций, которые должны выполняться **netware**, разделено на **функциональные уровни — слои (layers)**. Каждый **уровень** предназначается для реализации **определенного подмножества функций** («*есть слона по кусочкам*»)
- ✓ Каждый уровень предоставляет **сервис по доставке порций данных — PDU**, который доступен **через API** для **вышележащего уровня**
- ✓ При этом **детали реализации этого сервиса скрыты: подход «черного ящика»**
- ✓ **«Черные ящики» вложены друг в друга по принципу матрёшки**

Послойная архитектура стека протоколов



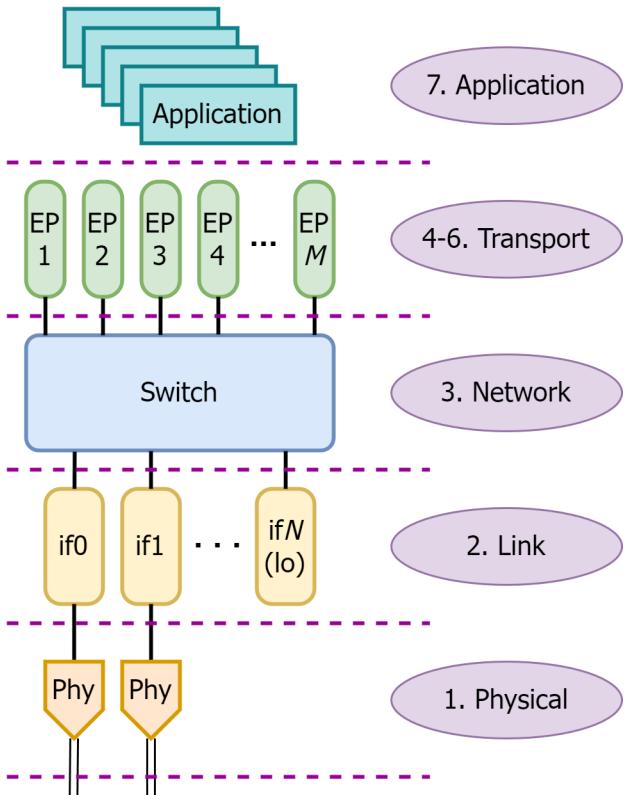
- ✓ Множество функций, которые должны выполняться **netware**, разделено на **функциональные уровни — слои (layers)**. Каждый **уровень** предназначен для реализации **определенного подмножества функций** («*есть слона по кусочкам*»)
- ✓ Каждый уровень предоставляет **сервис по доставке порций данных — PDU**, который доступен **через API** для **вышележащего уровня**
- ✓ При этом **детали реализации этого сервиса скрыты: подход «черного ящика»**
- ✓ **«Черные ящики» вложены друг в друга по принципу матрёшки**
- ✓ Только самый нижний **уровень 1 физически связан с подключенным звеном передачи данных**

Преимущества послойной архитектуры

- ✓ Декомпозиция комплексной функциональности *netware* на более мелкие части позволяет упростить процесс её реализации
- ✓ Разделение интерфейса и реализации позволяет обеспечивать взаимозаменяемость: реализация может быть заменена на другую при сохранении внешнего интерфейса её «чёрного ящика»
- ✓ Наличие специфицированных интерфейсов позволяет вести независимую разработку разных модулей
- ✓ Возможность создания нескольких альтернативных вариантов реализации одной и той-же функциональности, например, для разных сред передачи или моделей оборудования, позволяет гибко собирать требуемый вариант *netware* из набора стандартных модулей (как конструктор Lego)
- ✓ Возможность гибкого расширения функциональности *netware* путём создания новых модулей с улучшенной/изменённой функциональностью, поддерживающих внешние интерфейсы



Функциональные уровни (layers) netware

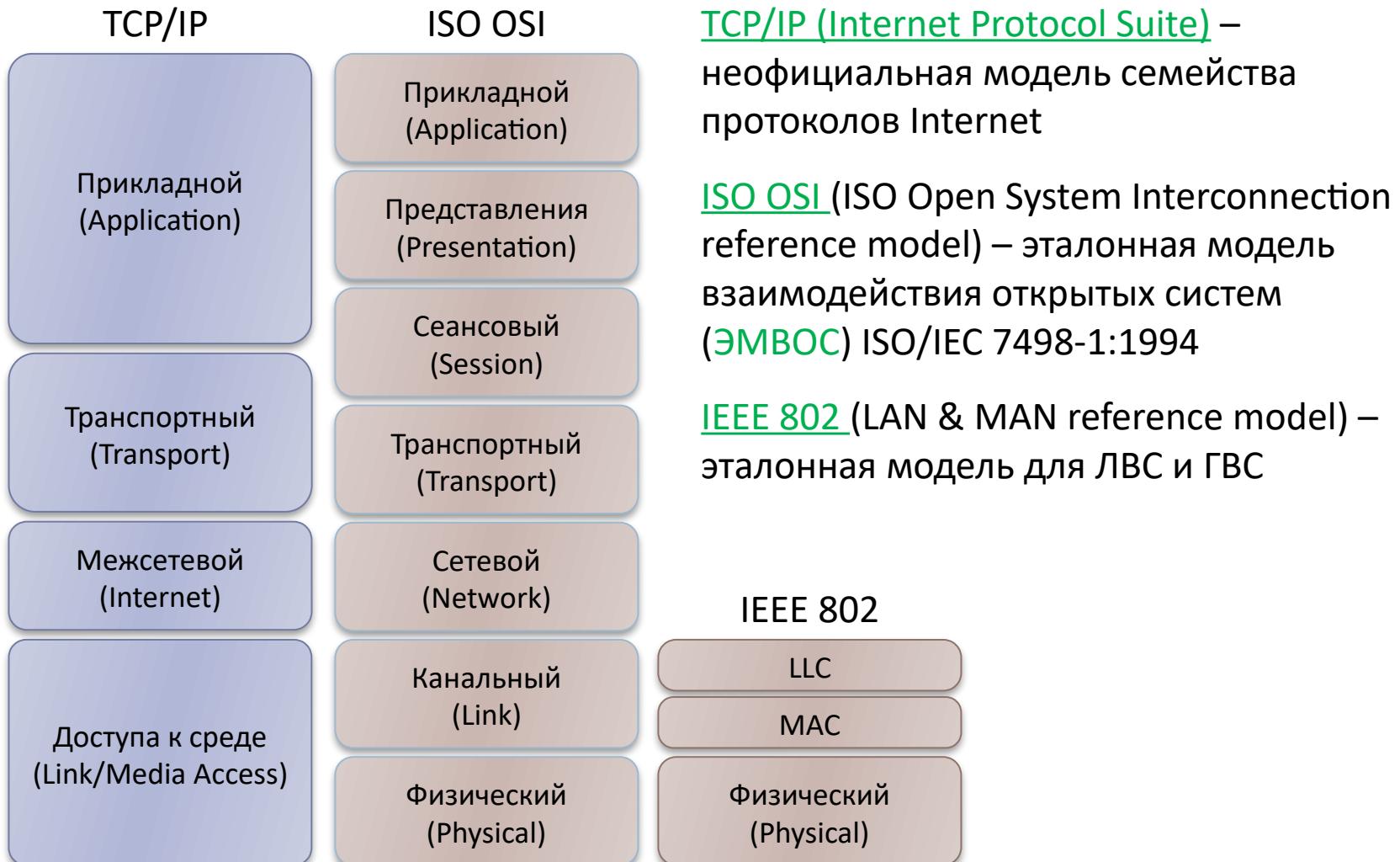


- **Прикладной (application)** — сюда относится всё, что функционирует внутри **приложений**
- **Транспортный (transport)** — сервис **сетевого транспорта данных** на уровне пары взаимодействующих **Endpoint (end-to-end)**
- **Сетевой (network)** — коммутация пакетов между **N интерфейсами** и **M endpoint-ами** (SAP)
- **Канальный (link)** — доставка порций данных по звену между **соседними узлами**
- **Физический (physical)** — преобразование поток битов \leftrightarrow **сигнал**

В составе **netware** узла каждый слой представлен одним или несколькими **протокольными модулями, взаимодействующими** друг с другом **внутри узла**, а также **поддерживающими диалог со своими peer modules (паритетными модулями)** других узлов



Эталонные модели сетевой архитектуры

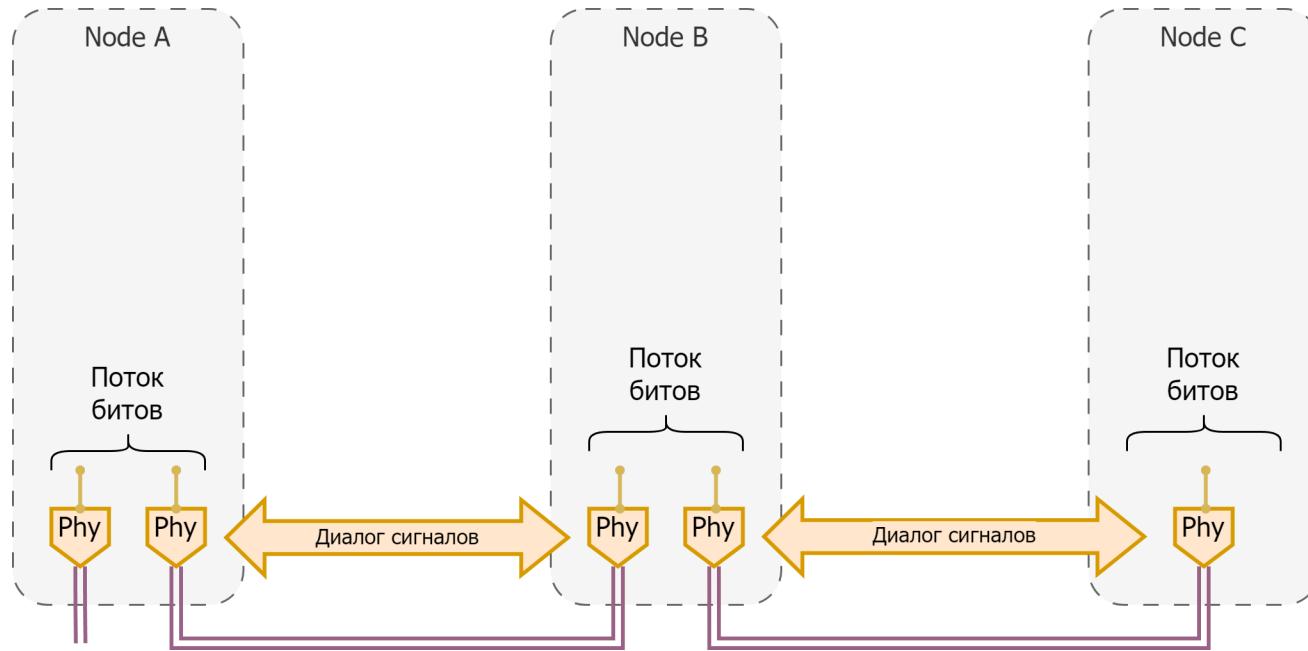


Функциональное описание уровней OSI

	Наименование уровня	Вид PDU	Назначение	Основные функции
7	Уровень приложений Application Layer	Диалог/ сообщения	Уровень для функционирования приложений	<ul style="list-style-type: none"> Сервис для пользователя, реализуемый приложениями
6	Уровень представления Presentation Layer	Данные	Преобразование данных в подходящий формат	<ul style="list-style-type: none"> Преобразование кодов Сериализация/десериализация Шифрование
5	Сеансовый уровень Session Layer	Поток данных	Управление логическими соединениями между SAP на окончных узлах	<ul style="list-style-type: none"> Установка/завершение /контроль соединения Аутентификация/авторизация
4	Транспортный уровень Transport Layer	Сегмент (Segment)	Доставка данных между SAP на окончных узлах, возможно с гарантией	<ul style="list-style-type: none"> Сегментация данных Управление темпом передачи Восстановление данных
3	Сетевой уровень Network Layer	Пакет (Datagram)	Доставка пакетов на указанный узел сети (без гарантии)	<ul style="list-style-type: none"> Адресация в сети Коммутация пакетов Индикация и диагностика
2	Уровень звена передачи Data Link Layer	Кадр (Frames)	Передача между соседними узлами последовательности порций данных (кадров)	<ul style="list-style-type: none"> Кадрирование Контроль правильности передачи Восстановление ошибочно переданных данных (опц.) Управление звеном передачи
1	Физический уровень Physical Layer	Поток битов	Организация тракта передачи битов между соседними узлами	<ul style="list-style-type: none"> Преобразование Данные<->Сигнал Физический соединитель (разъем) Управление физ.соединением



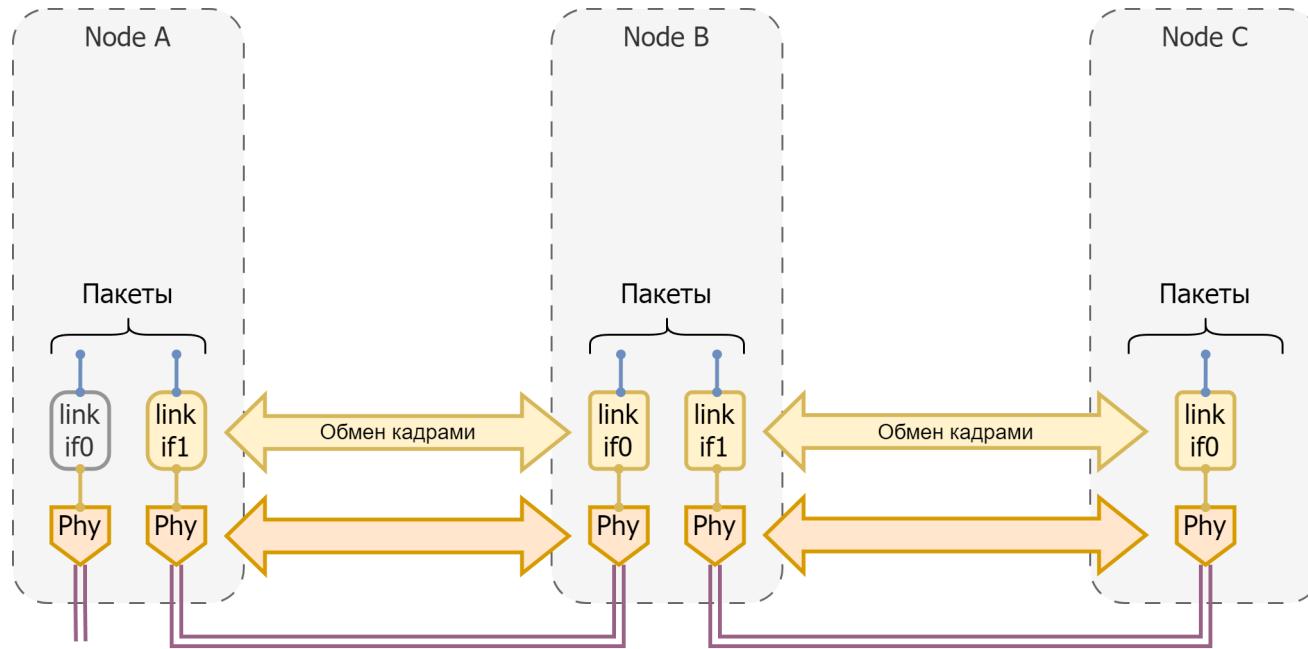
Взаимодействие физических модулей



- Соседние **физические модули** связаны друг с другом **средой передачи** и имеют возможность вести друг с другом **диалог** путём **передачи** (transmission) определённых **сигналов** в эту **среду** с последующим **приёмом** by peer module
- Физические модули** **модулируют/демодулируют** **сигналы**, преобразуя их из/в **поток битов**, который поступает через **интерфейс физического уровня** на **канальные модули**



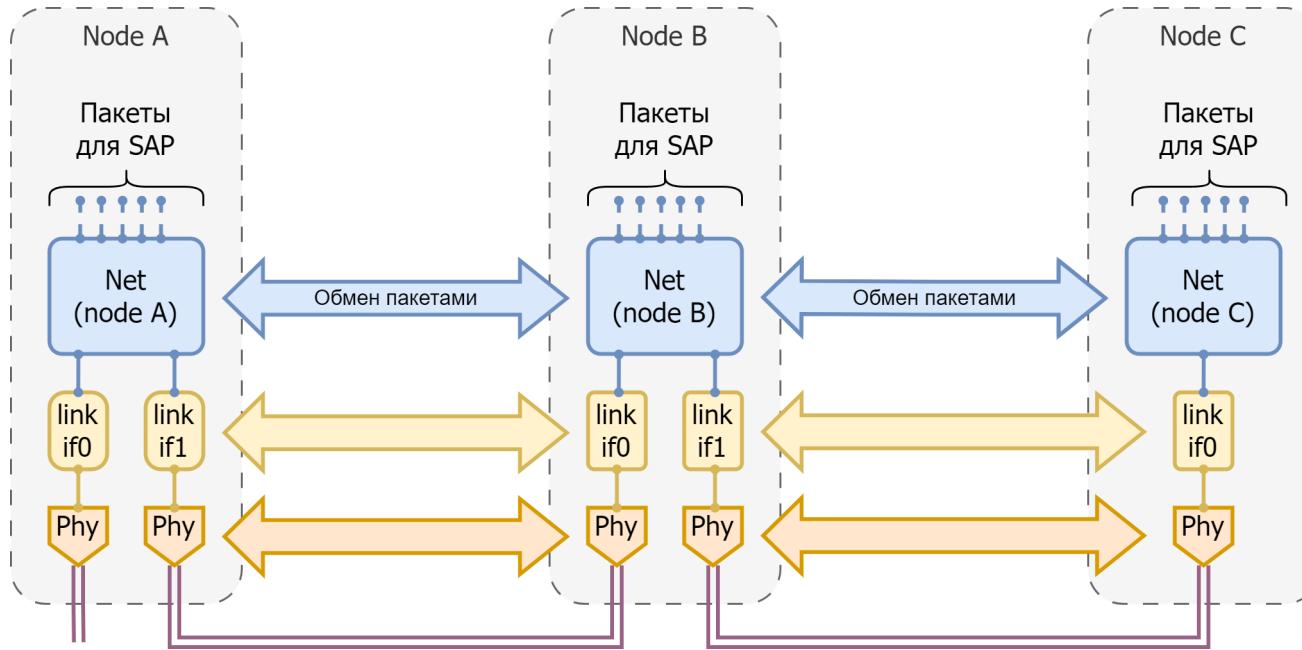
Взаимодействие канальных модулей



- ✓ Канальные модули взаимодействуют с физическими модулями своего сетевого интерфейса (*ifX*), через интерфейс физического уровня (aka $L1 \leftrightarrow L2$!), упаковывая в поток битов кадры (*frames*) — порции данных канального уровня
- ✓ Благодаря сервису своих физических модулей, соседние (peer) канальные модули ведут друг с другом диалог, осуществляя обмен кадрами в пределах своего звена передачи данных. В качестве *payload* в кадры могут включаться данные пакетов, получаемых/доставляемых через интерфейс канального уровня

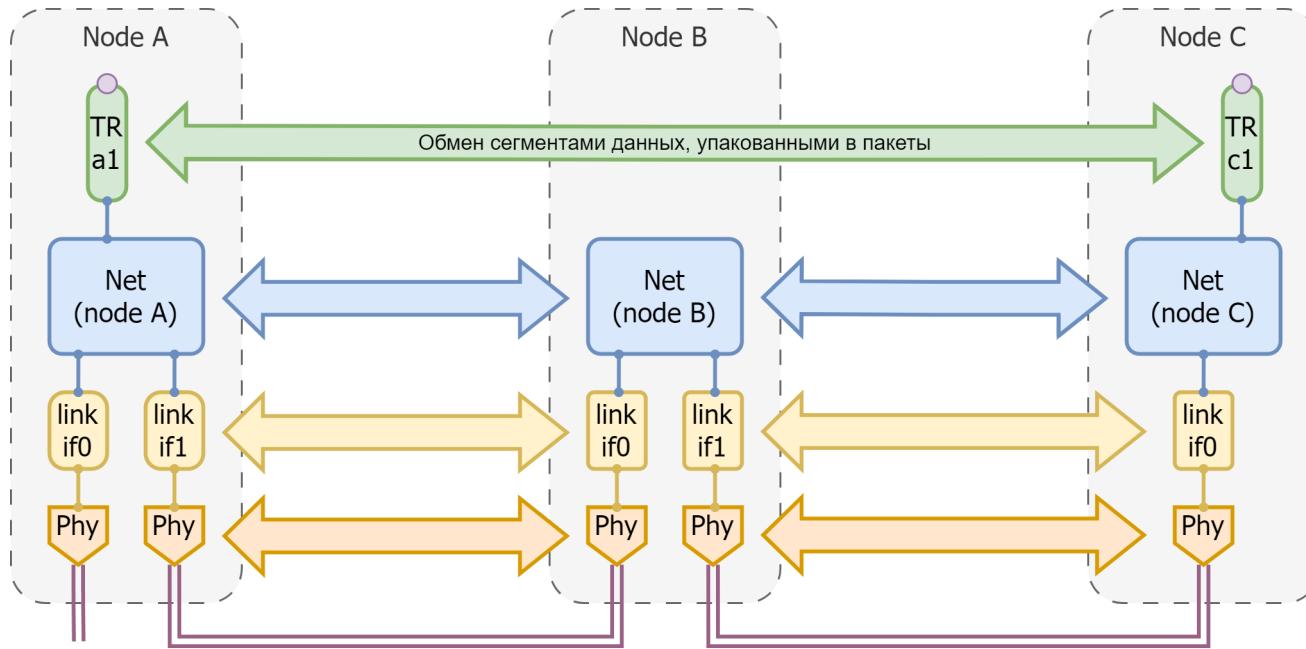


Взаимодействие сетевых модулей



- Сетевые модули поддерживают диалог(и) со всеми своими соседями (neighbors) с помощью канальных модулей сетевых интерфейсов (ifX) своего узла, получая и передавая им пакеты для доставки через подключенные звенья передачи данных
- Кроме этого сетевые модули получают и передают пакеты в направлении локальных SAP через интерфейс сетевого уровня (aka L3 ↔ L4)

Взаимодействие транспортных модулей

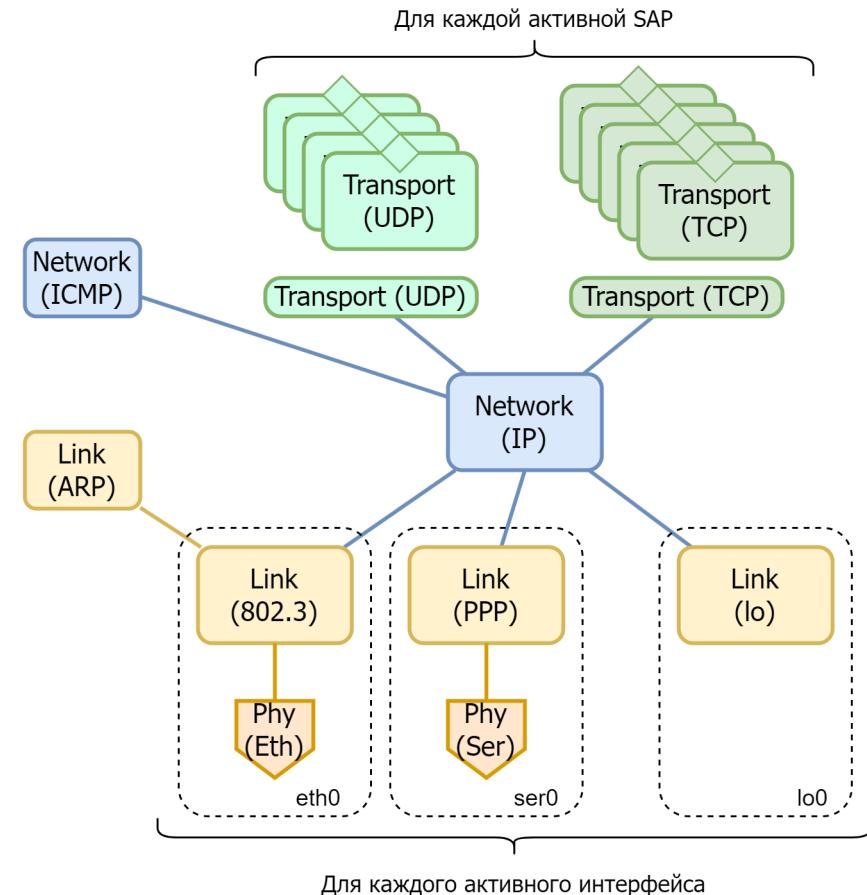


- ✓ Каждый **экземпляр транспортного модуля** поддерживает (как правило двухсторонний) **диалог** с **модулем peer endpoint** путём получения и передачи через **интерфейс сетевого уровня** пакетов, в которые инкапсулируются **порции данных транспортного уровня**, называемые **сегментами**
- ✓ Экземпляры транспортных модулей **создаются только на конечных узлах логических соединений**, поэтому транспортный сервис называют **end-to-end**



Структура протокольного стека узла

- Протокольным стеком (protocol stack) узла будем называть совокупность экземпляров протокольных модулей, функционирующих в составе netware узла, изображённую в виде графа (дерева), с указанием взаимодействия отдельных модулей друг с другом
- Состав protocol instances в стеке динамический и, прежде всего, зависит от:
 - топологии узла (количество и типы интерфейсов)
 - количество и типов открытых SAP
 - применяемого семейства протоколов (protocol suite)
- На примере показана структура стека узла с двумя физическими интерфейсами Eth0 и Ser0 и TCP/IP v4 suite
- В составе стека присутствуют экземпляры служебных протоколов (ARP, ICMP в примере), которые не задействованы в транспорте payload, но необходимы для функционирования сети



Устройство PDU

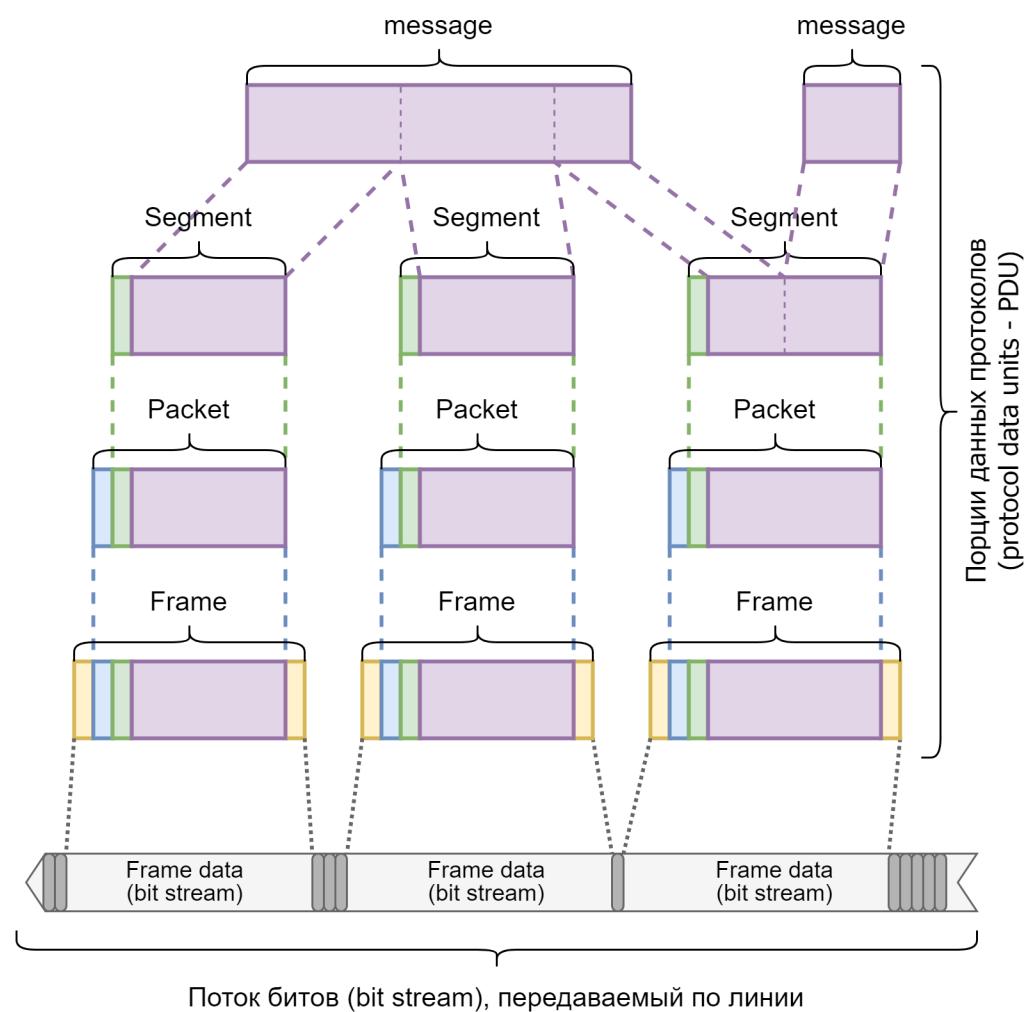
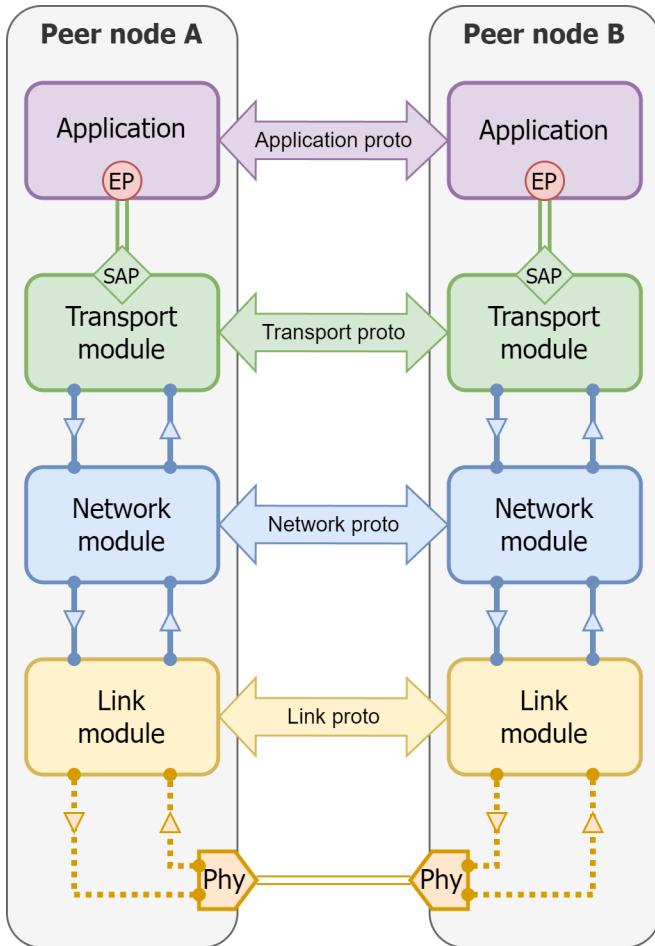
- ✓ PDU (protocol data unit) — порция данных в форме массива байтов некоторой (известной) длины L
- ✓ PDU, как правило, формируется модулем-отправителем путём записи определённых значений в байты буфера передачи S
- ✓ PDU для модуля-получателя доставляются сетевым транспортом в буфер приёма R
- ✓ peer модули ведут диалог друг с другом, используя заголовок
- ✓ структура (назначение) байтов заголовка определяется спецификацией протокола
- ✓ обычно в заголовке всегда присутствует фиксированная часть, содержащая набор обязательных полей



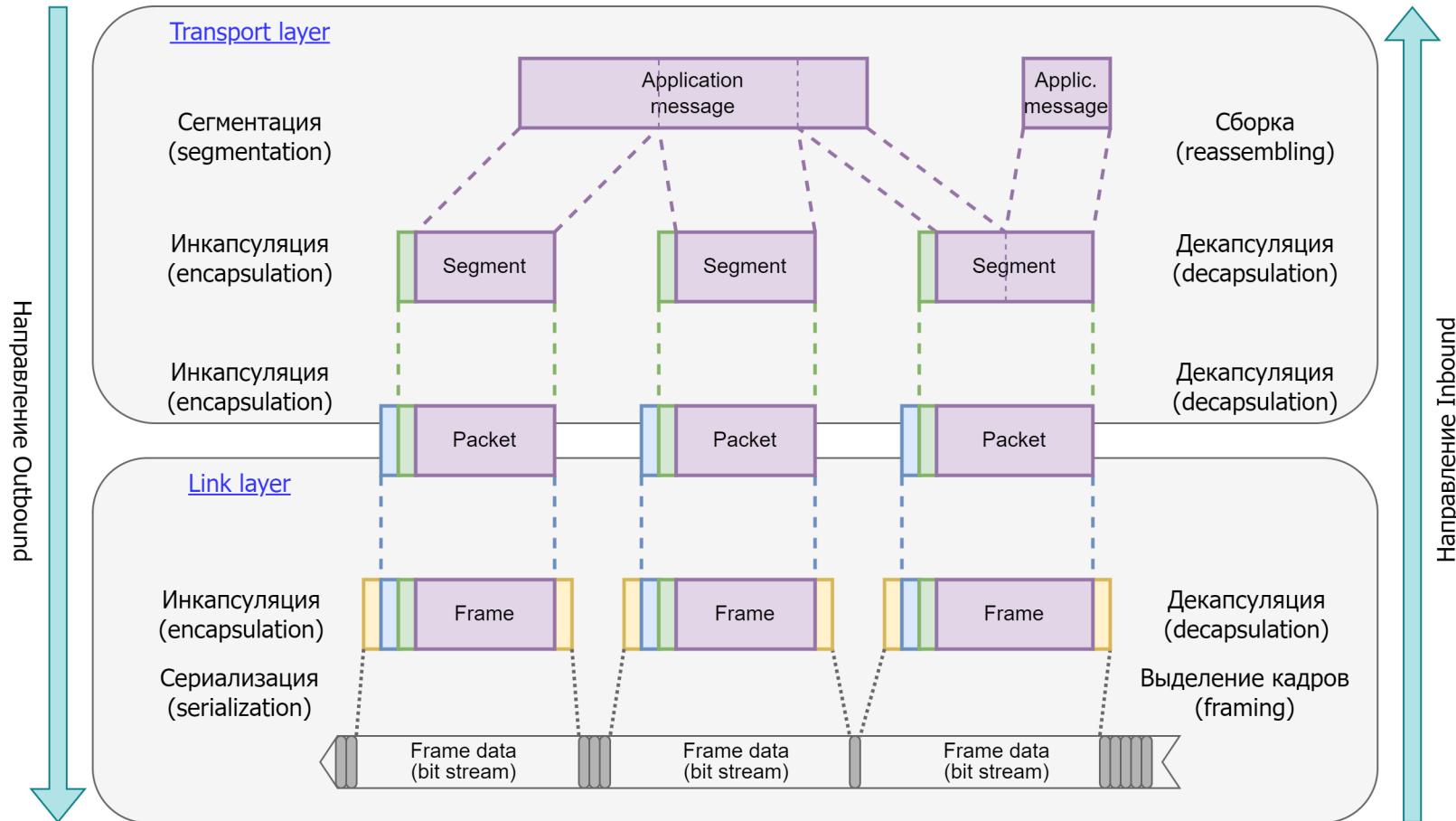
Структурно PDU может включать до трёх логических частей:

- заголовка (header), содержимое которого полностью определяется рассматриваемым протоколом (содержит собственные служебные данные этого протокола)
- полезной нагрузки (payload), содержащей инкапсулированные данные, транспортируемые по сети (для рассматриваемого протокола это просто набор байтов)
- концевика (trailer), обычно содержащего служебные данные, необходимые для контроля правильности передачи

Инкапсуляция протоколов и PDU



Преобразования данных в стеке протоколов



Мультиплексирование потоков в стеке

(Де-)мультиплексирование протоколов:

- ✓ В **PDU** одного протокола (в примере — сетевой протокол **IP**) **могут инкапсулироваться PDU** разных **вышележащих протоколов** (в примере — **ICMP, TCP, UDP...**) и при приёме их **необходимо направлять в соответствующие протокольные модули**
- ✓ Для этого в **заголовке IP** предусмотрено **поле дискриминатора протокола (protocol)**, в котором передаётся код **вышележащего протокола**, к которому относится инкапсулированный **payload**

(Де-)мультиплексирование экземпляров:

- ✓ Поступающие **сегменты TCP** необходимо корректно доставлять к **соответствующему protocol instance** (которых на узле, вероятнее всего, много)
- ✓ В рамках TCP, **protocol instances** идентифицируются по адресу **SAP**, включающему комбинацию из нескольких полей (в примере **port**). Идентификацией требуемого **instance** занимается **компонент протокола TCP (dispatcher)**, единственный экземпляр которого получает все входящие сегменты

