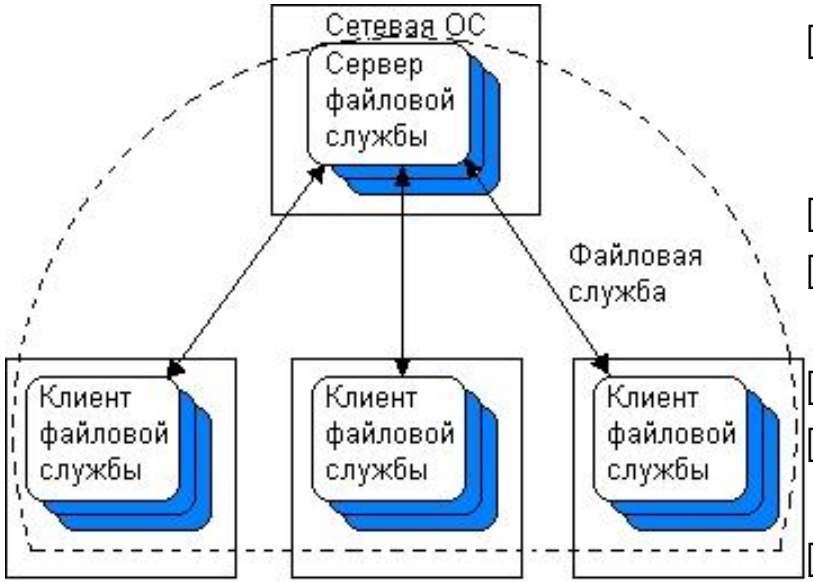
Распределенные информационные системы

Процессы

Реализация взаимодействия между компонентами в распределенных системах

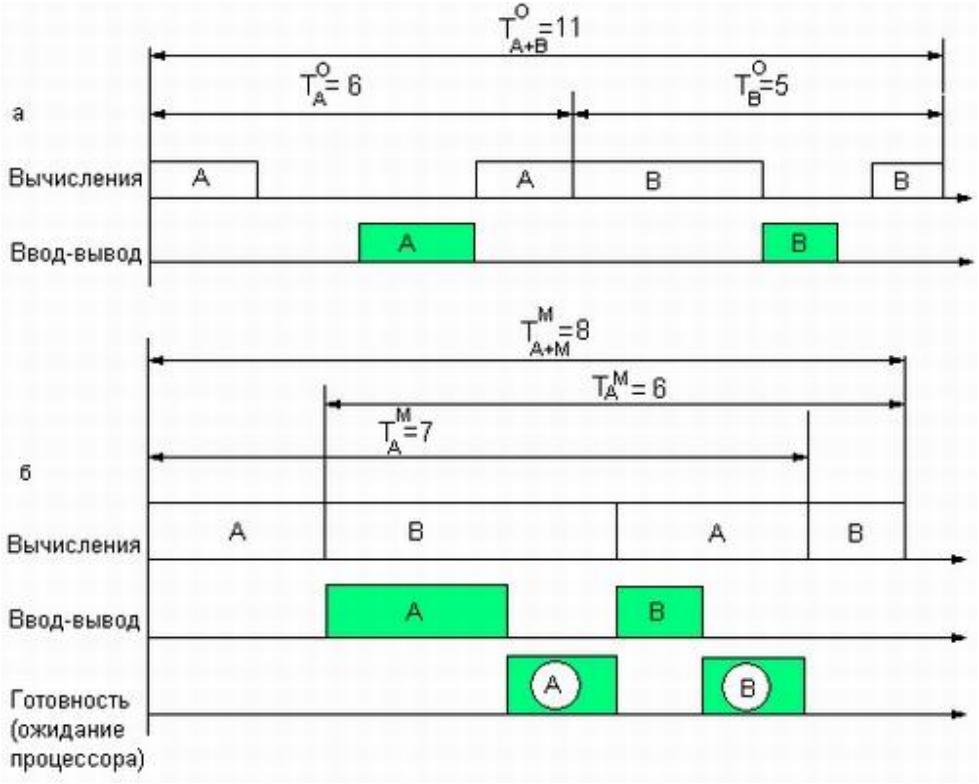
- В распределенных системах взаимодействие между компонентами реализуется средствами операционных систем узлов.
- Основным элементом с помощью которого реализуются функции выполняемые узлами являются процессы.
- Концепция процесса зародилась в операционных системах, где под этим понятием обычно обозначают выполняемую программу.



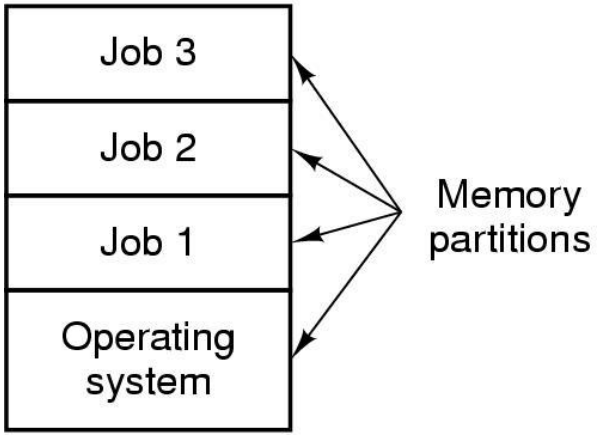
- С точки зрения ОС наиболее важными являются вопросы планирования и управления процессами.
- В РС наиболее важными являются вопросы:
- скорость реакции узла на поступающие запросы;
- минимизации времени ответа;
- обеспечкение прозрачности доступа к ресурсам РС;
- масштабируемость сервиса при возрастании потока запросов и т.п

Мультипрограммирование или многозадачный режим работы

ОС пакетной обработки



Время выполнения двух задач: в однопрограммной системе (а), в мультипрограммной системе (б)

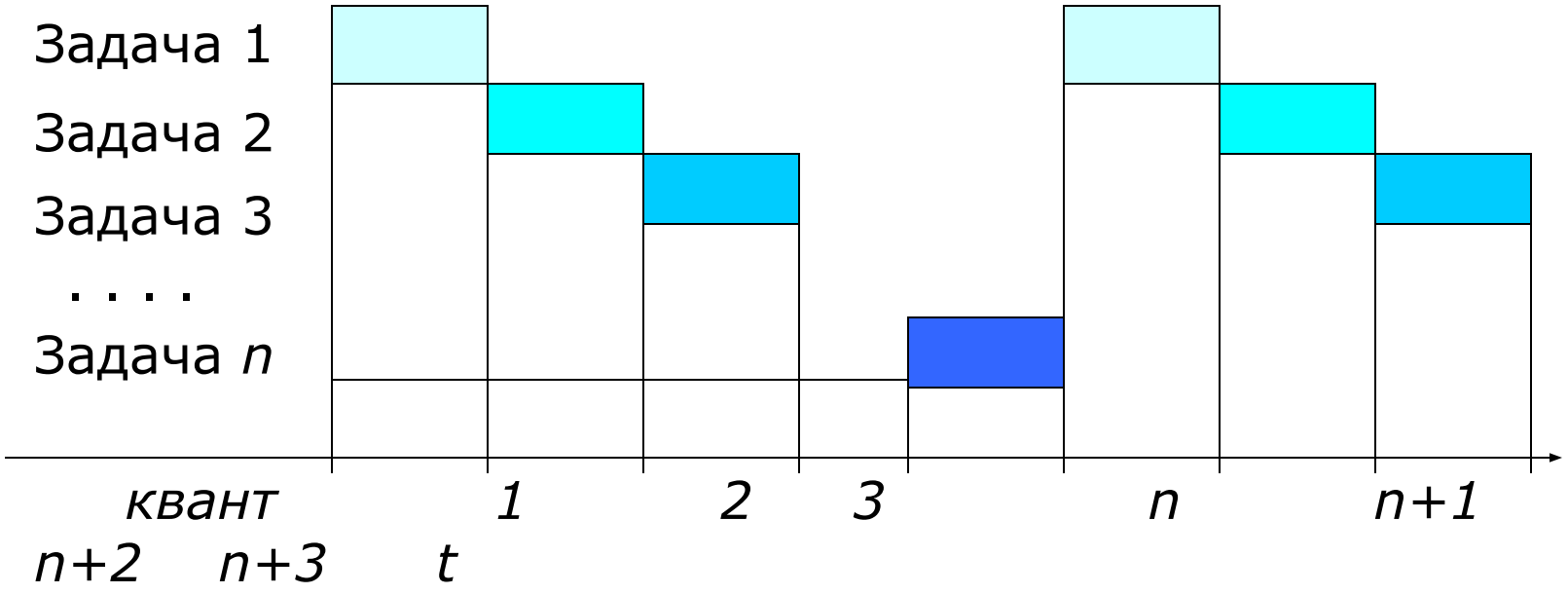


Размещение нескольких заданий в памяти.

Пример ОС:

DOS, OS/360 – фирма IBM

Системы разделения времени
Квантование времени (MULTICS 1968)



Процессы выполнения

- Для выполнения программ операционная система создает несколько виртуальных процессоров, по одному для каждой программы.
- Чтобы отслеживать эти виртуальные процессоры, операционная система поддерживает *таблицу процессов (process table), содержащую записи для сохранения значений регистров* процессора, карт памяти, открытых файлов, учетных записях пользователей, привилегиях и т. п.
- *Процесс (process) часто определяют как выполняемую программу,* то есть программу, которая в настоящее время выполняется на одном из виртуальных процессоров операционной системы.

Понятие процесса

- **Процессом**, называют программу в момент выполнения, в некоторых ОС исполняемую программу называют **задачей**, процесс и задача являются синонимами.
- С каждым процессом связывается его **адресное пространство** — список адресов в памяти от некоторого минимума (обычно нуля) до некоторого максимума, которые процесс может прочесть и в которые он может писать. Адресное пространство содержит саму программу, данные к ней и ее стек.
- Со всяким процессом связывается некий **набор регистров**, включая счетчик команд, указатель стека и другие аппаратные регистры, плюс вся остальная информация, необходимая для запуска программы. Это **информация о процессе**.
- Вся совокупность информации о процессе, необходимая для ее продолжения процессором после прерывания называется **контекстом** процесса. Контекст процесса является частью (подмножеством) информации о процессе.
- Всякий процесс выполняемый в системе запускается от **имени пользователя** инициировавшего запуск программы на выполнение.
- **Права доступа процесса** к ресурсам системы определяются **правами доступа пользователя**, от чьего имени программа была запущена.

Таблица процессов

- Это массив (или связанный список) структур хранящий информацию о процессах исполняемых системой.
- Каждому процессу в таблице процессов соответствует один экземпляр структуры, хранящий информацию о данном процессе.
- Процесс может находиться в одном из двух состояний:
  - выполнение;
  - остановлен.

Составляющие процесса

- В режиме выполнения:
  - адресное пространство процесса в оперативной памяти ЭВМ;
  - информация о процессе и ресурсах которые он использует, например об открытых файлах или установленных сетевых соединениях.
- В режиме останова:
  - образа памяти процесса (адресное пространство процесса);
  - информация о процессе хранящаяся в таблице процессов.

Жизненный цикл процесса в ОС

- Включает в себя следующие стадии:
  - Создание процесса;
  - выполнение процесса;
  - уничтожение процесса.

Системные вызовы управляющие процессами

- Процесс создается родительским процессом с помощью обращения к функции ядра «создать процесс».
- Обращение к функциям ОС называется системным вызовом.
- Главными системными вызовами, управляющими процессами, являются вызовы связанные с созданием и уничтожением процессов.
- Системный вызов выполняется в привилегированном режиме работы процессора.

Смена режимов работы процессора при выполнении системного вызова

Виды системных вызовов связанных с процессами

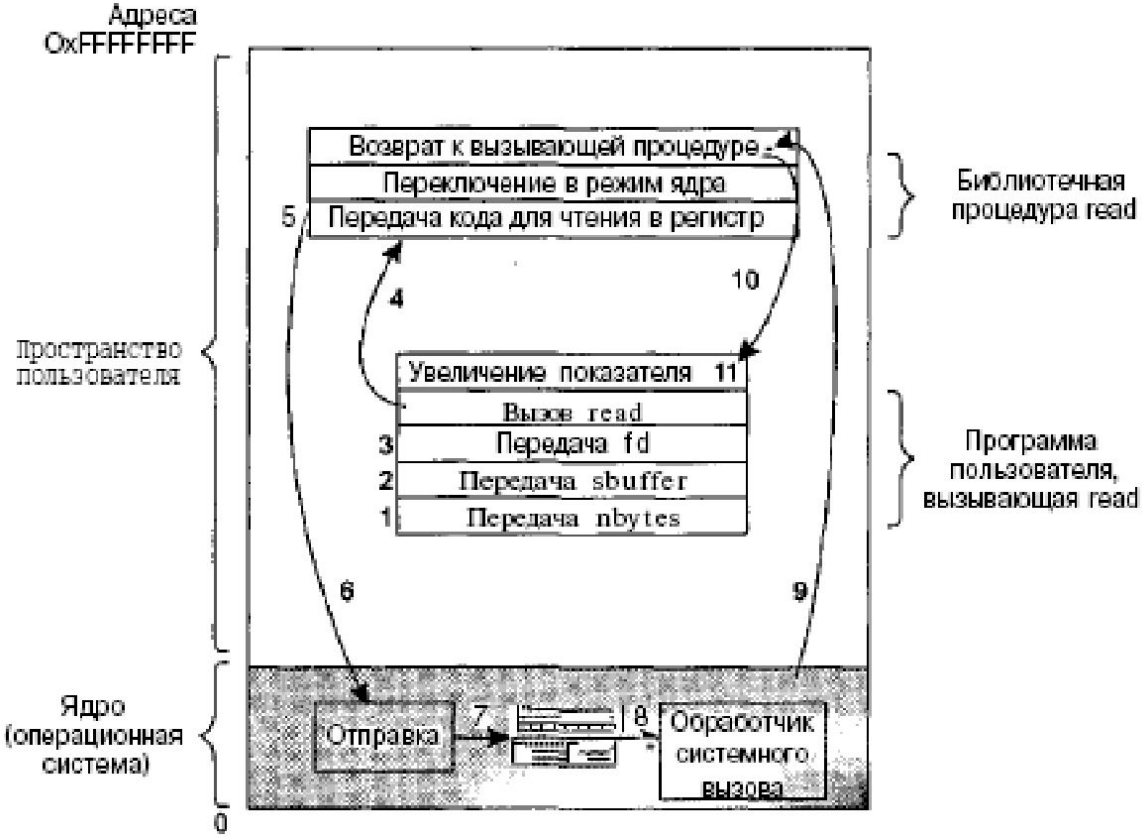
- создание процесса;
- освобождение или выделение дополнительной памяти процессу;
- ожидание завершения какого-либо процесса;
- перекрытие адресных пространств процесса;
- передача сигнала процессу;
- завершение процесса;
- и др.

Системные вызовы

1-3 размещение параметров вызова в стек;

4 - выполнение вызова библиотечной процедуры read;

5 – размещение номера системного вызова в регистр;



count = read(fd, buffer, nbytes);

6 – переключение в режим ядра;

7 – проверка номера системного вызова и передача управления обработчику;

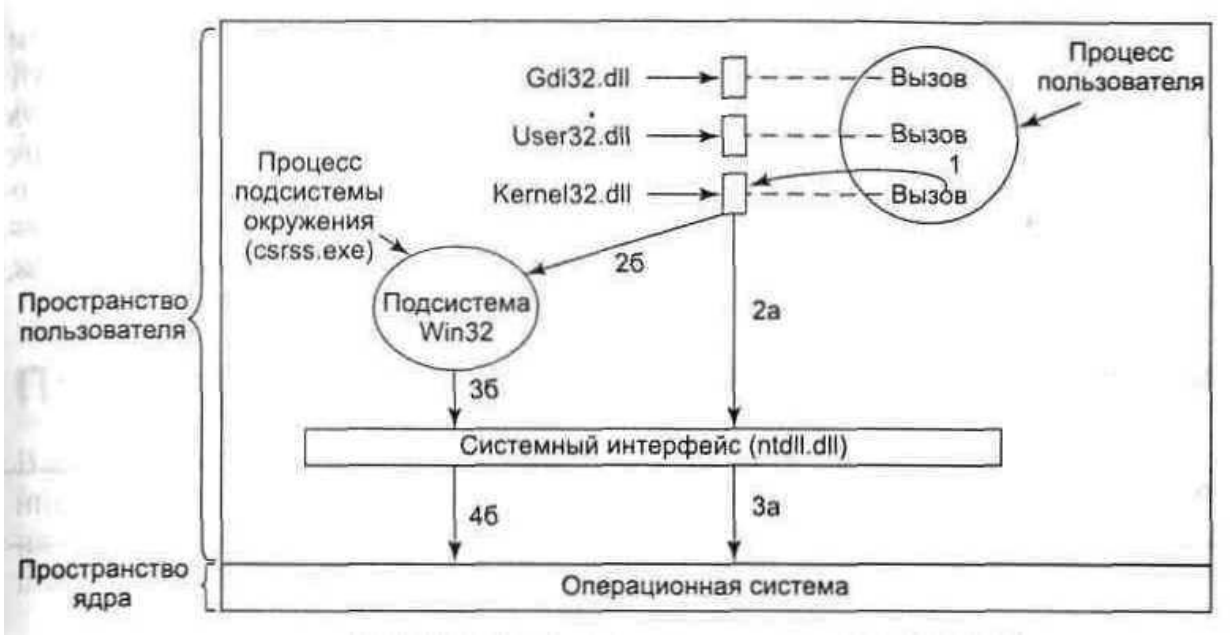
8 – выполнение обработчика;

9 – возврат в режим пользователя;

10 – возврат из процедуры read;

11 – очистка стека (восстановление указателя вершины стека).

Выполнение вызовов Win32 API (Windows 2000)



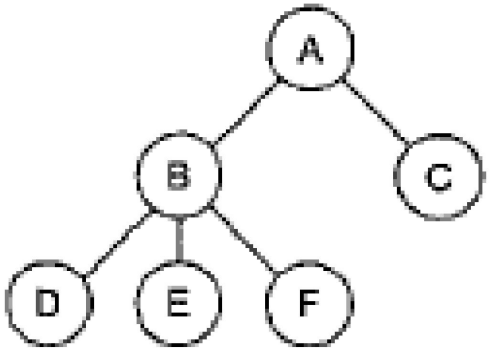
2а и 3а динамические библиотеки обращаются к другой динамической библиотеке {ntdll.dll) которая, в свою очередь, обращается к ядру операционной системы. При этом dll может выполнить всю работу самостоятельно, совсем не обращаясь к системным вызовам;

- 2б, 3б и 4б. Для других вызовов Win32 API выбирает маршрут: процесс пользователя обращается к подсистеме Win32 (csrss.exe), которая выполняет некоторую работу, и затем обращается к системному вызову (ntdll.dll);
- В первой версии Windows NT практически все вызовы Win32 API выбирали маршрут 2б, 3б, 4б, так как большая часть операционной системы (например, графика) была помещена в пространство пользователя.
- начиная с версии NT 4.0, для увеличения производительности большая часть кода была перенесена в ядро (в драйвер Win32/GDI);
- в Windows 2000 только небольшое количество вызовов Win32 API, например вызовы для создания процесса или потока, идут по длинному пути. Остальные вызовы выполняются напрямую, минуя подсистему окружения Win32.

Сигналы передаваемые процессам

- Сигналы являются программными аналогами аппаратных прерываний и могут быть сгенерированы по различным причинам, а не только из-за истечения какого-либо интервала времени.
- Многие аппаратные прерывания (например, вызванные выполнением недопустимой команды или использованием неправильного адреса) также преобразуются в сигналы процессу, в котором произошла ошибка.
- Сигнал вызывает:
  - временную остановку работы процесса независимо от того, что процесс делает в данный момент;
  - сохраняет его регистры в стеке
  - запускает специальную процедуру обработки сигнала (например, передающую повторно предположительно потерянное сообщение).
- После завершения обработки сигнала работающий процесс запускается заново в том состоянии, в котором он находился до сигнала.

Дерево процессов



- Если процесс может создавать несколько других процессов (называющихся **дочерними процессами),** а эти процессы, в свою очередь, тоже могут создать дочерние процессы, таким образом формируется дерево процессов,

Идентификация процессов в системе

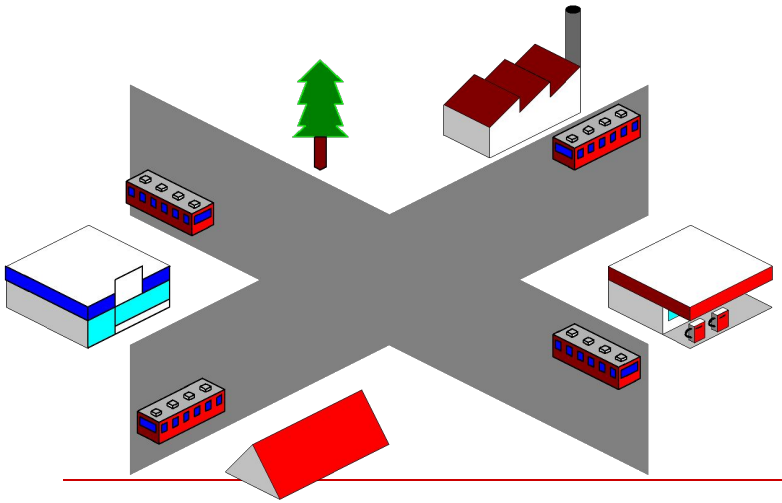
- Для идентификации процессов в системе используются идентификаторы процессов PIDs (Process Identificator)
- PID – это число присваемое процессу при запуске.
- Каждому процессу присваевается идентификатор пользователя (UID – User Idetificator), запустившего данный процесс.

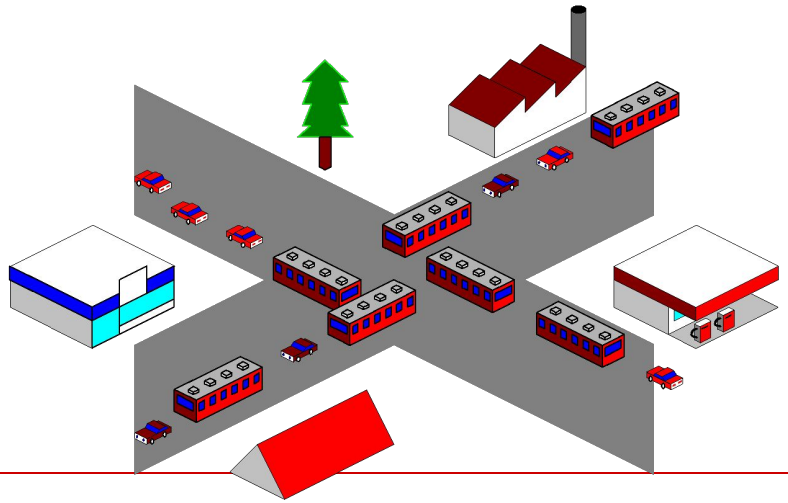
Связанные процессы и межпроцессное взаимодействие

- Связанные процессы — это те, которые объединены для выполнения некоторой задачи, и им нужно часто передавать данные от одного к другому и синхронизировать свою деятельность.
- Такая связь называется межпроцессным взаимодействием.

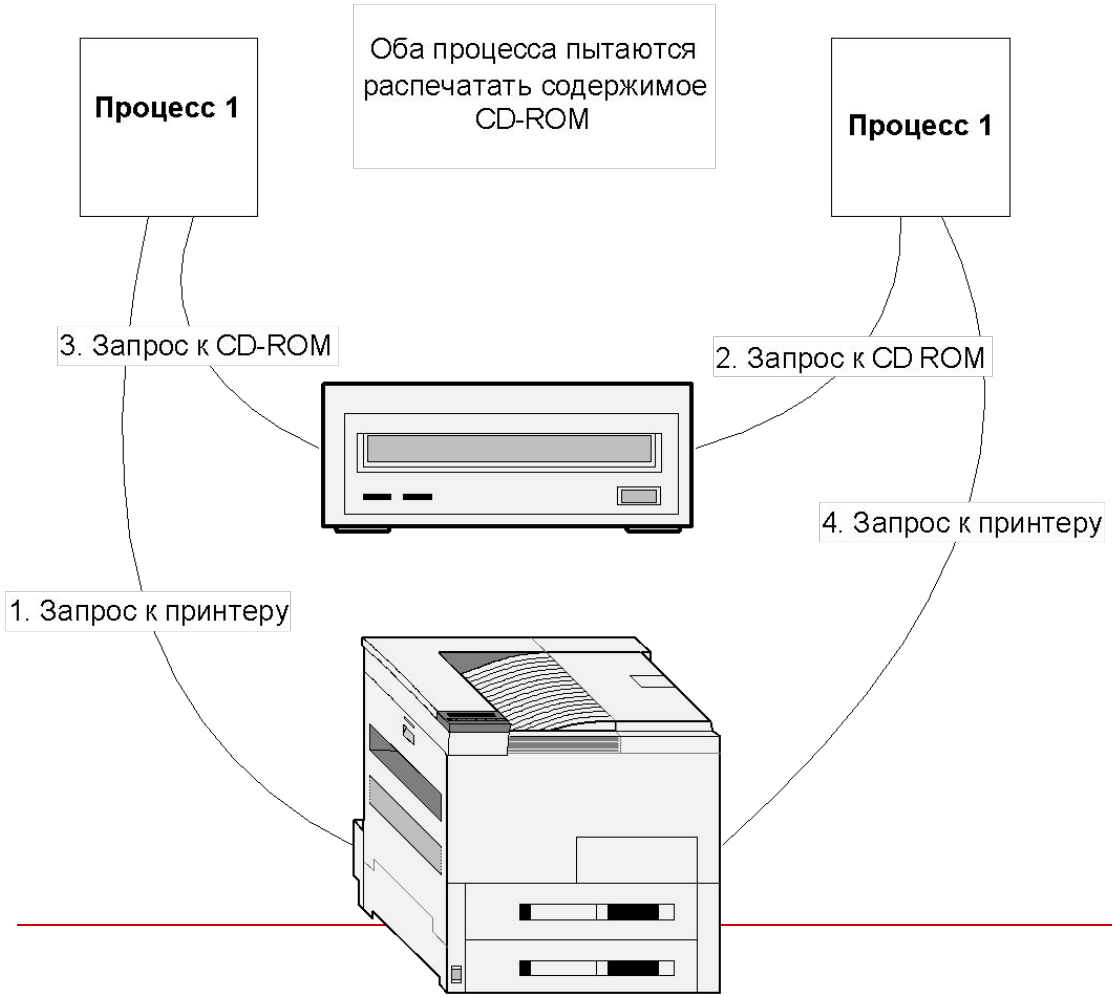
Взаимоблокировка процессов

- Когда взаимодействуют два или более процессов, они могут попадать в патовые ситуации, из которых невозможно выйти без посторонней помощи.
- Такая ситуация называется тупиком, тупиковой ситуацией или взаимоблокировкой.





Пример тупика



Потоки выполнения

- Хотя процессы являются строительными блоками распределенных систем, практика показывает, что дробления на процессы, предоставляемого операционными системами, на базе которых строятся распределенные системы, недостаточно.
- Вместо этого оказывается, что наличие более тонкого дробления в форме нескольких *потоков выполнения {threads) на процесс значительно упрощает построение* распределенных приложений и позволяет добиться лучшей производительности.

Понятие потока выполнения

- Основная причина использования потоков заключается в том, что во многих приложениях одновременно происходит несколько действий, часть которых может периодически быть заблокированной.
- Модель программирования упрощается за счет разделения такого приложения на несколько последовательных потоков, выполняемых в квазипараллельном режиме.

Преимущества потоков выполнения

1. Возможность использования параллельными процессами единого адресного пространства и всех имеющихся данных. Эта возможность играет весьма важную роль для тех приложений, которым не подходит использование нескольких процессов (с их раздельными адресными пространствами).
2. Легкость (то есть быстрота) их создания и ликвидации по сравнению с более «тяжеловесными» процессами. Во многих системах создание потоков осуществляется в 10–100 раз быстрее, чем создание процессов.
3. Когда потоки работают в рамках одного центрального процессора, они не приносят никакого прироста производительности, но когда выполняются значительные вычисления, а также значительная часть времени тратится на ожидание ввода-вывода, наличие потоков позволяет этим действиям перекрываться по времени, ускоряя работу приложения.
4. И наконец, потоки весьма полезны для систем, имеющих несколько центральных процессоров, где есть реальная возможность параллельных вычислений.

Процессы и потоки

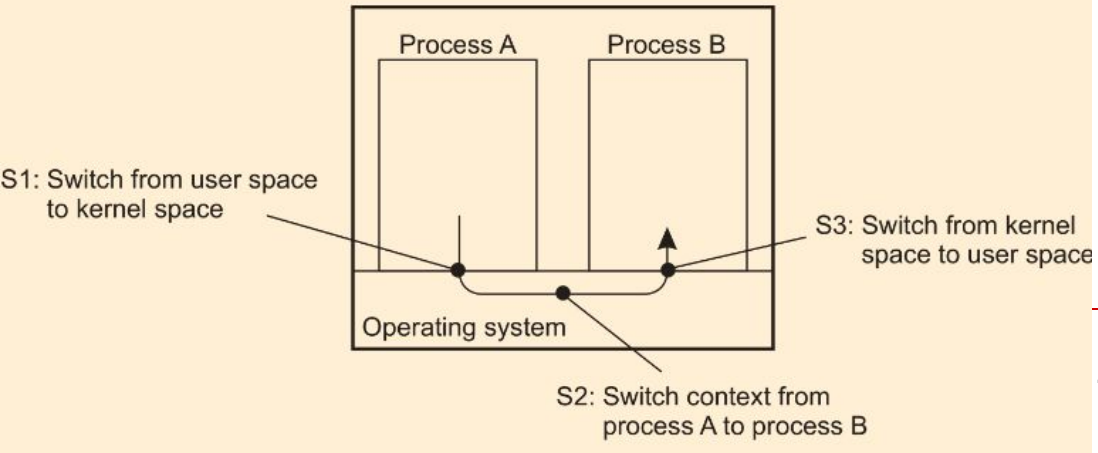
- **Создание потоков требует от ОС меньших накладных расходов, чем процессов.**
- **Все потоки одного процесса всегда принадлежат одному приложению, поэтому ОС изолирует потоки в гораздо меньшей степени,**
- **Поскольку каждый поток может иметь доступ к любому виртуальному адресу процесса, один поток может использовать стек другого потока.**

- Между потоками одного процесса нет полной защиты, потому что, во-первых, это невозможно, а во-вторых, не нужно.
- Чтобы организовать взаимодействие и обмен данными, потокам вовсе не требуется обращаться к ОС, им достаточно использовать общую память — один поток записывает данные, а другой читает их.
- С другой стороны, потоки разных процессов по-прежнему хорошо защищены друг от друга.

Потоки в локальных системах

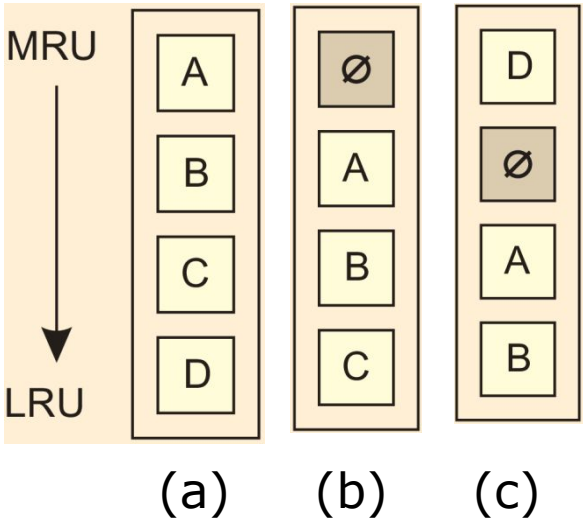
Межпроцессное взаимодействие

- Многопоточная структура часто используется при построении больших приложений. Подобные приложения часто разрабатываются в виде наборов совместно работающих программ, каждая из которых выполняется отдельным процессом.
- Этот подход типичен для среды UNIX. Кооперация между программами реализуется в виде межпроцессного взаимодействия (через механизмы IPC).
- Поскольку IPC требует вмешательства в ядро, процесс обычно вынужден сначала переключиться из пользовательского режима в режим ядра (точка *S1 на* рисунке). Это требует изменения карты памяти в блоке MMU, а также сброса буфера TLB.
- В ядре происходит переключение контекста процесса (точка 52), после чего второй процесс может быть активизирован очередным переключением из режима ядра в пользовательский режим (точка *S3).*



Последнее переключение вновь потребует изменения карты памяти в блоке MMU и буфера сброса TLB.

“Стоимость” переключения контекстов процессов



Рассмотрим состояние кэша процессора:

- а) состояние кэша перед прерыванием, блок D является кандидатом на удаление из кэша при возникновении прерывания (b).
- В дальнейшем, этот блок может опять понадобиться, что приведет его возврату в кэш (c).
- В результате будет разрушена оптимальная структура кэша, что повлечет замедление работы приложения.

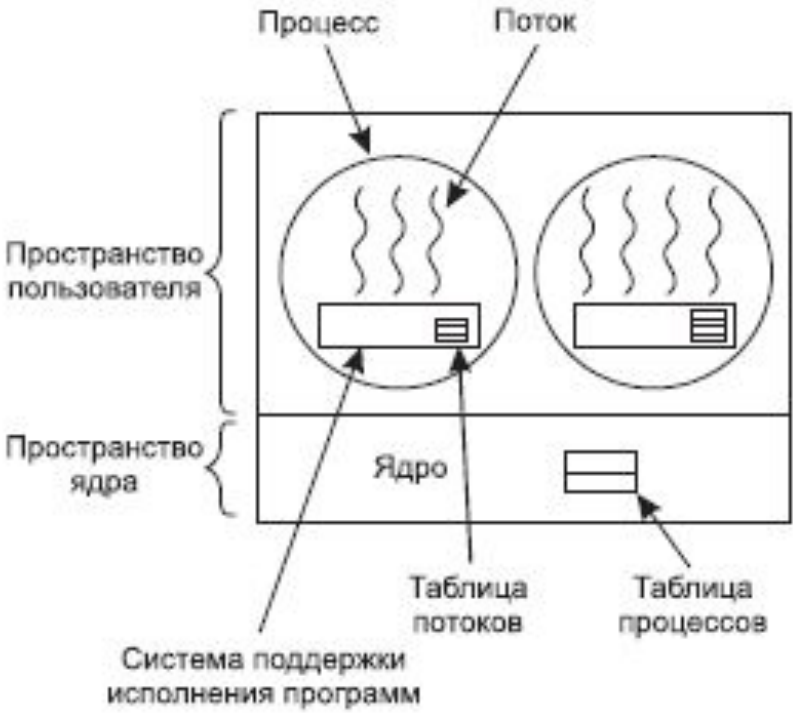
Потоки выполнения в нераспределенных системах

- Имеется особая причина использовать потоки выполнения:
  - ✔ многие приложения просто легче разрабатывать, структурировав их в виде набора взаимосвязанных потоков выполнения.
  - ✔ Например, в случае текстового редактора в отдельные потоки можно выделить обработку ввода пользователя, проверку орфографии и грамматики, оформление внешнего вида документа, создание индекса и т. п.

Реализация потоков выполнения

- Потоки выполнения обычно существуют в виде пакетов. Подобные пакеты содержат механизмы для создания и уничтожения потоков, а также для работы с переменными синхронизации, такими как мьютексы и условные переменные.
- Существует два основных подхода к реализации пакетов для потоков выполнения.
  - Первый из них состоит в создании библиотеки для работы с потоками выполнения, выполняющейся исключительно в режиме пользователя.
  - Второй подход предполагает, что за потоки выполнения отвечает (и управляет ими) ядро.

Преимущества потоков выполнения на пользовательском уровне (1)

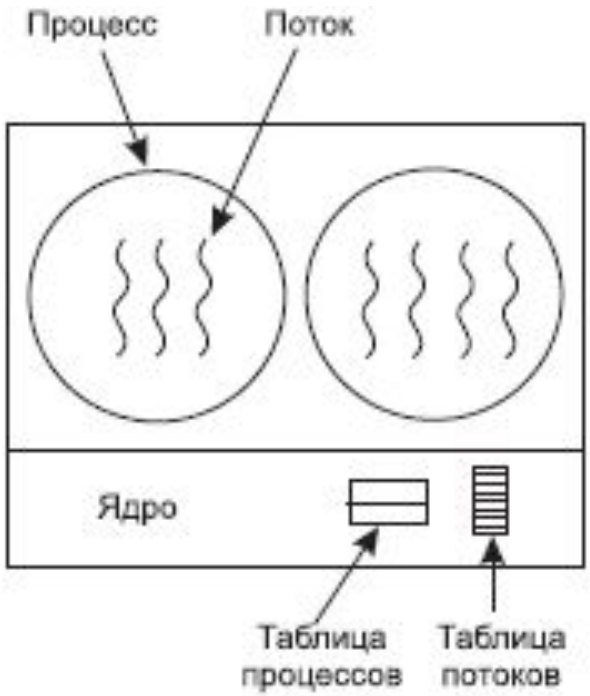


- Библиотека для работы с потоками выполнения на пользовательском уровне имеет множество преимуществ.
- Во-первых, дешевле обходится создание и уничтожение потоков выполнения.
- Поскольку все управление потоками реализуется в адресном пространстве пользователя, стоимость создания потока выполнения определяется в первую очередь затратами на память, выделяемую для создания стека под поток.
- Аналогично и уничтожение потока выполнения в основном состоит в освобождении памяти, задействованной под стек, после того как надобность в потоке отпадает. Обе операции достаточно дешевы.

Преимущества потоков выполнения на пользовательском уровне (2)

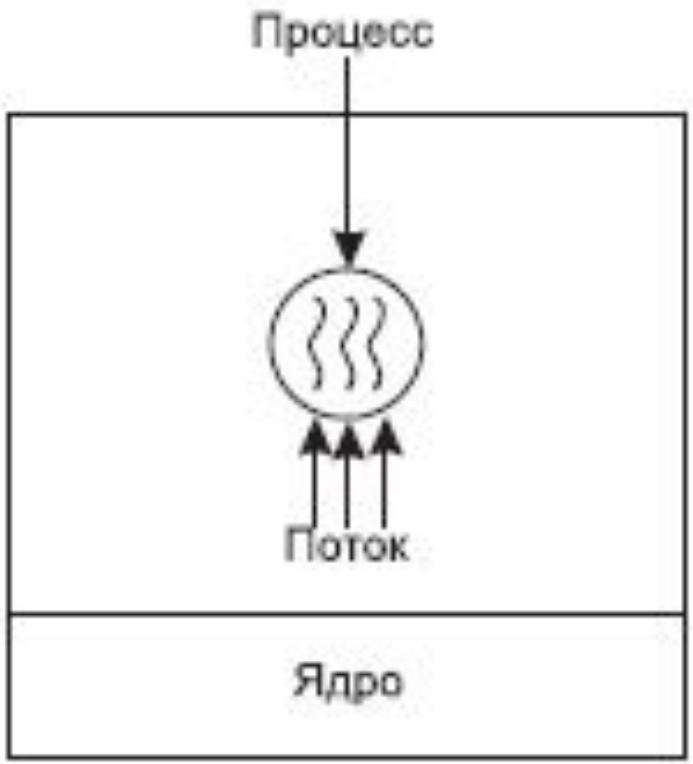
- Второе преимущество потоков выполнения на пользовательском уровне состоит в том, что переключение контекста требует всего нескольких инструкций.
- В основном в сохранении и последующем восстановлении сохраненных значений при переключении с потока на поток нуждаются исключительно значения регистров процессора.
- Нет необходимости изменять карты памяти, сбрасывать буфер TLB, контролировать загрузку процессора и т. д.
- Переключение контекста потоков выполнения производится при необходимости в синхронизации двух потоков, например, при обращении к секции совместно используемых данных.

Реализация потоков в ядре



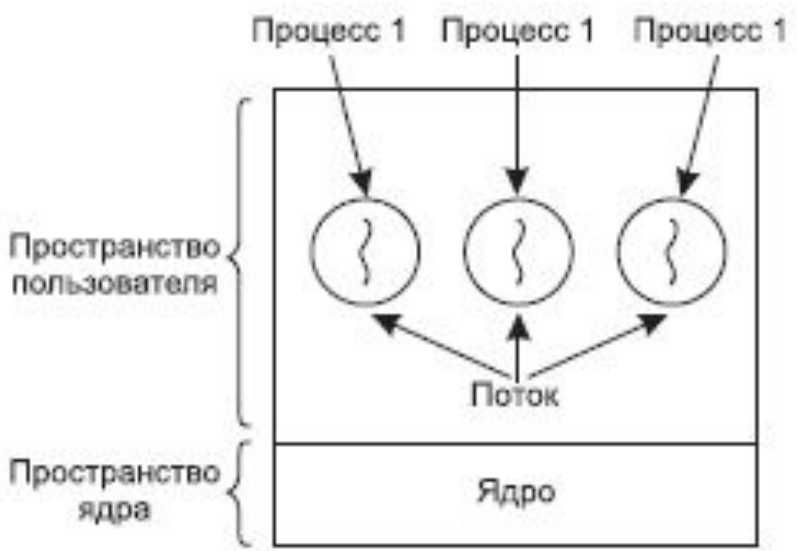
- Для потоков реализуемых на уровне ядра вся информация о потоках аналогична той, что используется для пользовательских потоков.
- При работе в режиме ядра создание и уничтожение потоков требует более существенных затрат.
- Хотя потоки на уровне ядра могут решить многие проблемы, но их главный недостаток в весьма существенных затратах на реализацию системных вызовов, поэтому если потоки создаются/удаляются достаточно часто, то это влечет за собой существенные издержки.
- Другой проблемой являются сигналы. Сигналы предназначены для процессов, а не для птоков.

Модель многие-к-одному



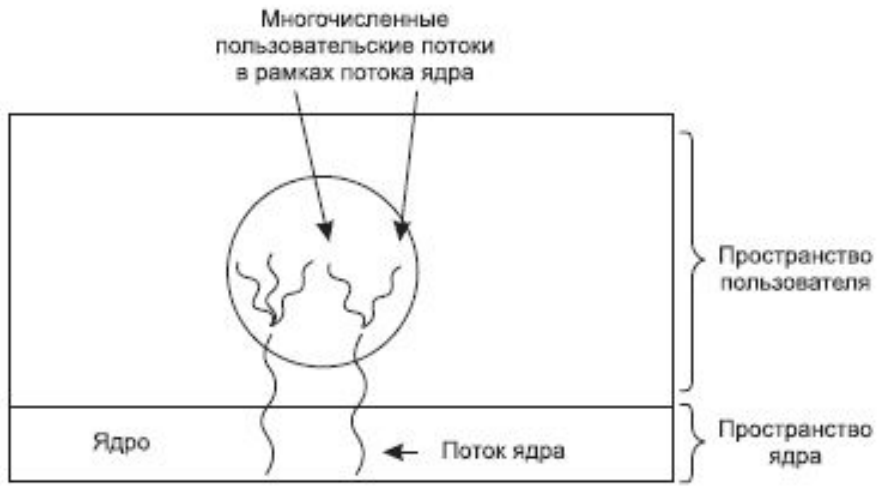
- Many-to-one threading model - когда несколько потоков отображаются на один планируемый процесс.
- В результате при использовании системного вызова блокирующего данный процесс блокируются все потоки связанные с данным процессом.

Модель один-к-одному



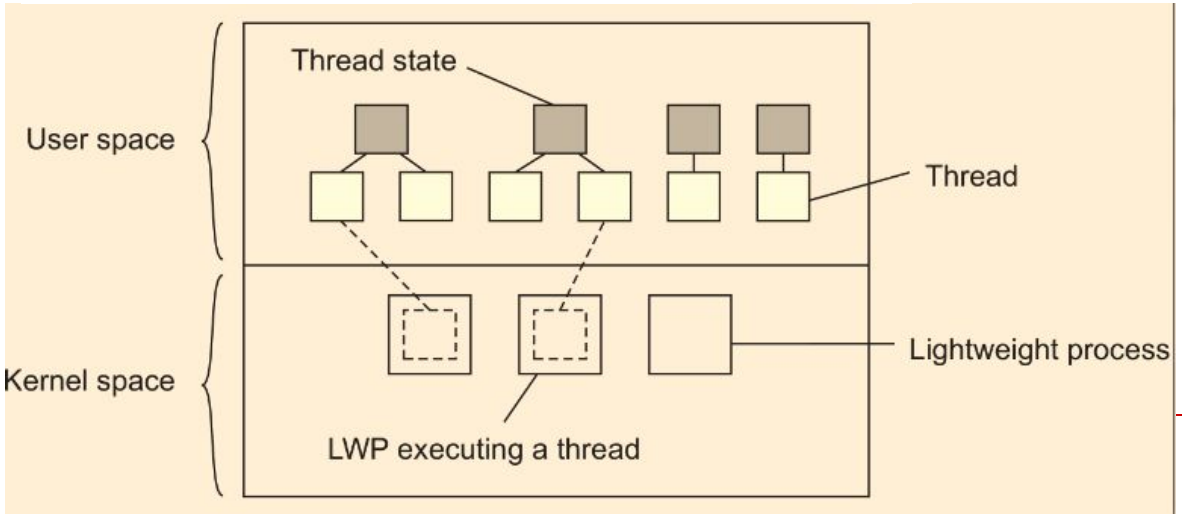
- One-to-one threading model – каждый процесс имеет только один поток, который диспетчируется независимо от других.
- Достоинством этой модели является возможность выполнения операций над потоками без необходимости обращения к ядру.

Гибридная реализация потоков



Здесь используются потоки на уровне ядра и один или несколько потоков на пользовательском уровне.

Эта модель обладает максимальной гибкостью



Использование потоков против применения группы конкурирующих процессов

- Применение потоков является способом одновременного и параллельного исполнения в рамках одного приложения.
- На практике, часто можно встреть реализации когда приложения строятся как коллекция параллельно работающих процессов, объединяемых с помощью средств межпроцессного взаимодействия предлагаемых операционными системами.
  - Примером применения такого подхода является реализация веб-сервера Apache, который по-умолчанию стартует как пятерка процессов, предназначенных для обслуживания поступающих запросов.
  - Каждый процесс является однопотоковой реализацией сервера, способной взаимодействовать с другими экземплярами с помощью стандартных средств ОС.
- Использование процессов вместо потоков имеет одно важное преимущество – обеспечивается разделение пространств данных процессов.

  Разделение процессов и их пространств данных обеспечивается аппаратными средствами процессоров.
- Это преимущество нельзя недооценивать, так как при использовании потоков вся забота о параллельном доступе к разделяемым данным ложиться на разработчика многопоточного приложения.

Потоки в распределенных системах

Почему потоки применяют в РС

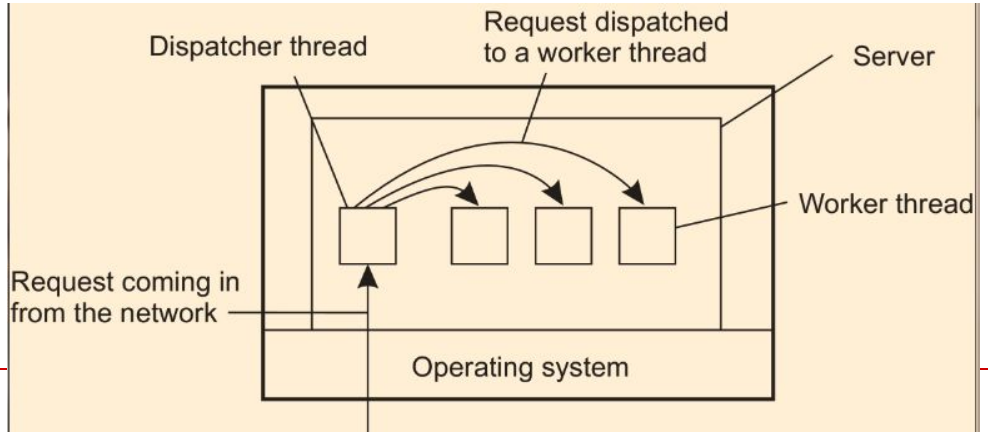
- Важнейшим свойством потоков выполнения является возможность выполнения системных блокирующих вызовов, без блокировки всего процесса в рамках которого работает поток.
- Это свойство является привлекательным для распределенных систем, что делает их более приспособленными к быстрым коммуникациям и обеспечивает одновременное поддержание нескольких логических соединений между компонентами РС.

Многопоточные клиенты

- Для обеспечения высокой степени прозрачности клиенты должны обладать поддержкой работы в многопоточном режиме.
- Например Web-браузер, для сокрытия достаточно больших величин задержек должен поддерживать возможность поддержания нескольких одновременных соединений с веб-вервером для одновременной закачки содержимого различных частей веб страниц.
- Кроме того современные браузеры могут одновременного выполнять доставку нескольких веб страниц с различных серверов, что также требует многопоточности.

Многопоточные серверы

- Рассмотрим файловый сервер. Обычно файловый сервер ожидает обращений от клиентов с запросами на выполнение файловых операций.
- Часто такой сервер реализуется в виде нескольких параллельно исполняемых потоков, один из которых выполняет роль диспетчера запросов (поток диспетчер), а остальные реализуют рабочие потоки.
- Данная реализация обеспечивает необходимую прозрачность и производительность этого файлового сервера.



Три способа построения сервера

Модель	Характеристика
Многопоточная	Параллелизм, блокировка системных вызовов
Однопоточный процесс	Отсутствие параллелизма, блокировка системных вызовов
Многопоточный процесс	Параллелизм, блокировка системных вызовов отсутствует

Виртуализация

Понятие виртуализации

- В компьютерных технологиях под термином "виртуализация" обычно понимается абстракция вычислительных ресурсов и предоставление пользователю системы, которая "инкапсулирует" (скрывает в себе) собственную реализацию.
- Проще говоря, пользователь работает с удобным для себя представлением объекта, и для него не имеет значения, как объект устроен в действительности.

Преимущества виртуализации:

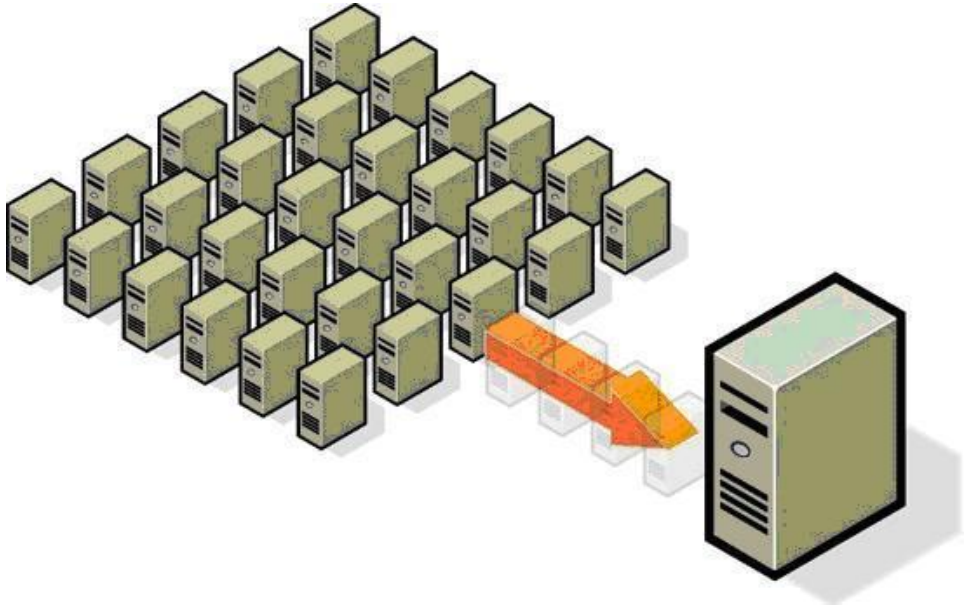
1. Эффективное использование вычислительных ресурсов
2. Сокращение расходов на инфраструктуру
3. Снижение затрат на программное обеспечение.
4. Повышение гибкости и скорости реагирования системы.
5. Несовместимые приложения могут работать на одном компьютере
6. Повышение доступности приложений и обеспечение непрерывности работы предприятия
7. Возможности легкой архивации
8. Повышение управляемости инфраструктуры

Виртуализация ЭВМ

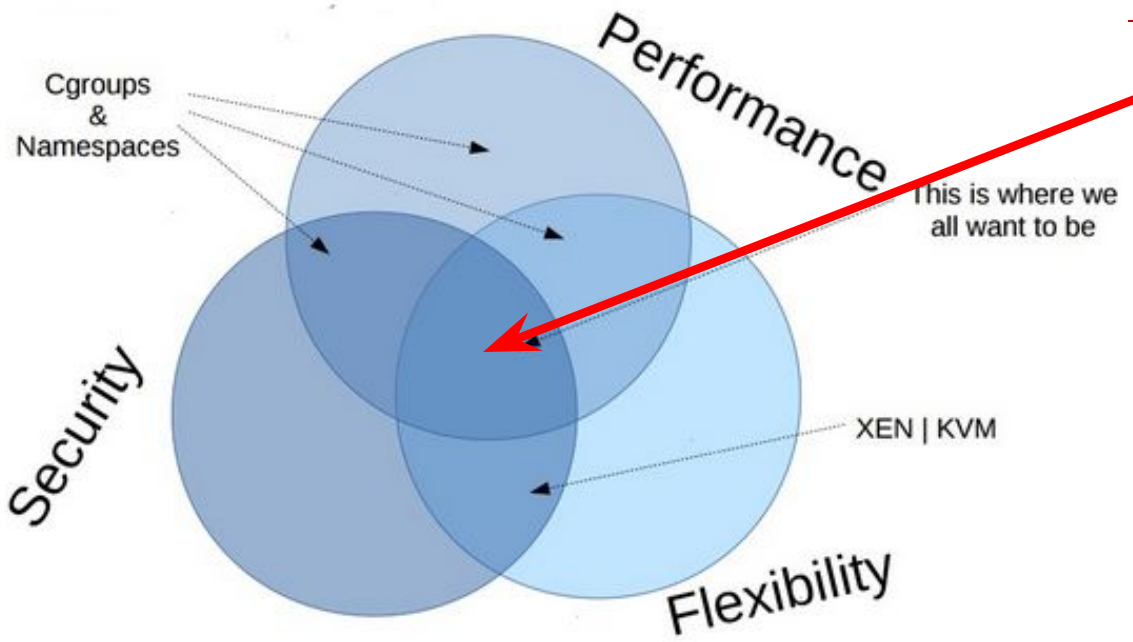
- Виртуализация серверов — размещение нескольких логических серверов в рамках одного физического.
- Цели виртуализации:
  - Предоставить каждому пользователю изолированную среду исполнения приложений.
  - Повысить гибкость использования ресурсов ЭВМ исполняемыми на ней приложениями.
  - Повысить защищенность приложений друг от друга исполняемых на одной и той же ЭВМ.
  - Повысить эффективность использования аппаратных средств ЭВМ.

Виртуализация ресурсов физического сервера

- Виртуализация ресурсов физического сервера позволяет:
  - гибко распределять их между приложениями, каждое из которых при этом "видит" только предназначенные ему ресурсы и "считает", что ему выделен отдельный сервер, т. е. в данном случае реализуется подход "один сервер — несколько приложений"



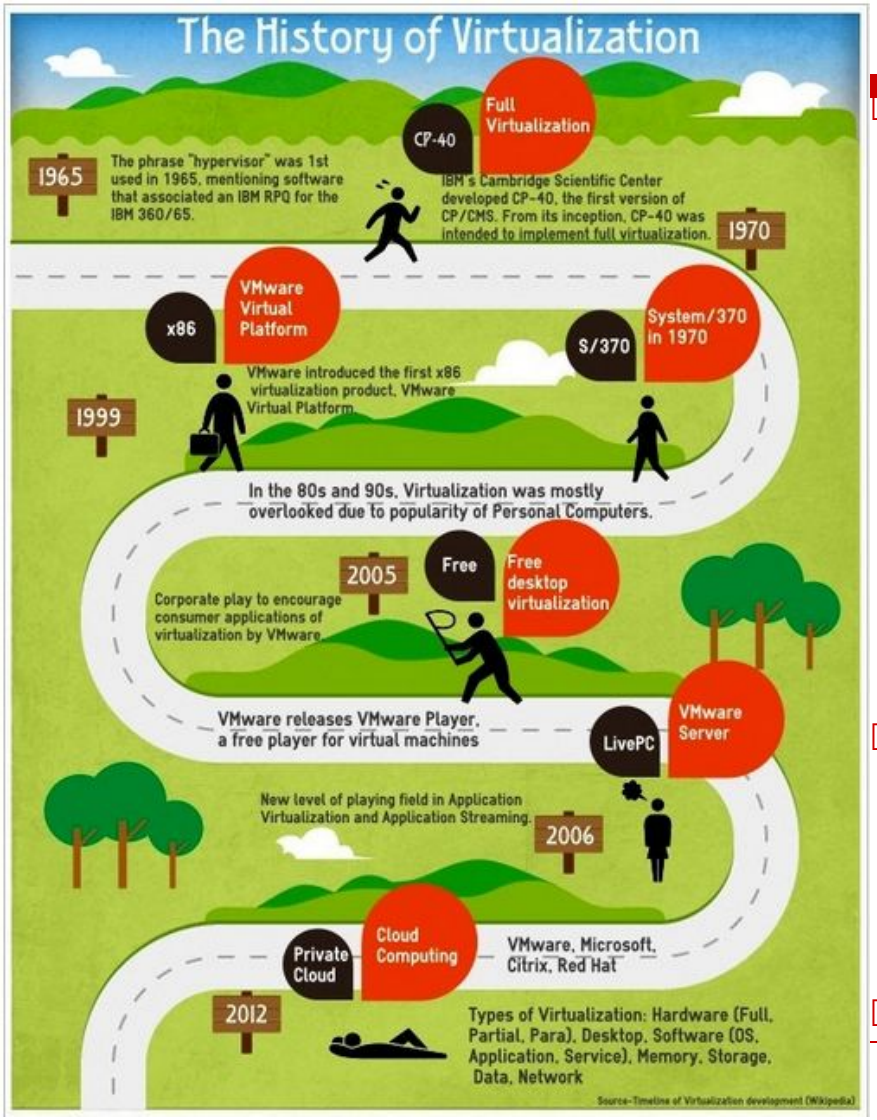
Цели виртуализации



Область эффективной виртуализации

- Первые версии гипервизоров отличались относительной медлительностью и действительно приводили к серьезному снижению производительности по сравнению с тем, как операционные системы и запущенные в них приложения работали на «реальном железе»
- Сегодня появилась модель предоставления виртуального хостинга на базе контейнеров ОС (или «Легких ВМ») – максимум эффективности при минимальной нагрузке на серверы.

История виртуализации (1)



- 1965. Выражение “Hypervisor” впервые появилось применительно к ПО обработки RPQ на ЭВМ IBM 360/65.

- Примерно 1966. В кембриджском научном центре разработан эмулятор CP-40 для S/360-40, явившийся первой попыткой реализации полной программной виртуализации физической ЭВМ.
- 1967. На основе этой разработки была создана CP[-67]/CMS1967. На основе этой разработки была создана CP[-67]/CMS – virtual machine1967. На

История виртуализации (2) Первый гипервизор VM/370

- В начале 70-х гипервизор CP-67, был переработан в виде OS VM/370 для нового семейства машин System/370, выпущенного на рынок в 1972 г.
- Линейка машин System/370 в 1990-х годах была заменена компанией IBM линейкой System/390. Виртуализация в ОС MVS 390 была сохранена.
- В 2000 году IBM выпустила машины z-серии, поддерживающие 64-разрядное виртуальное адресное пространство при сохранении обратной совместимости с System/360 и поддержке виртуализации.
- Все эти системы фирмы IBM поддерживали виртуализацию на десятилетия раньше того момента, когда она приобрела популярность на машинах семейства x86.

Машина IBM System/ 370.

Экран терминала с заставкой VM/370

История виртуализации (3)

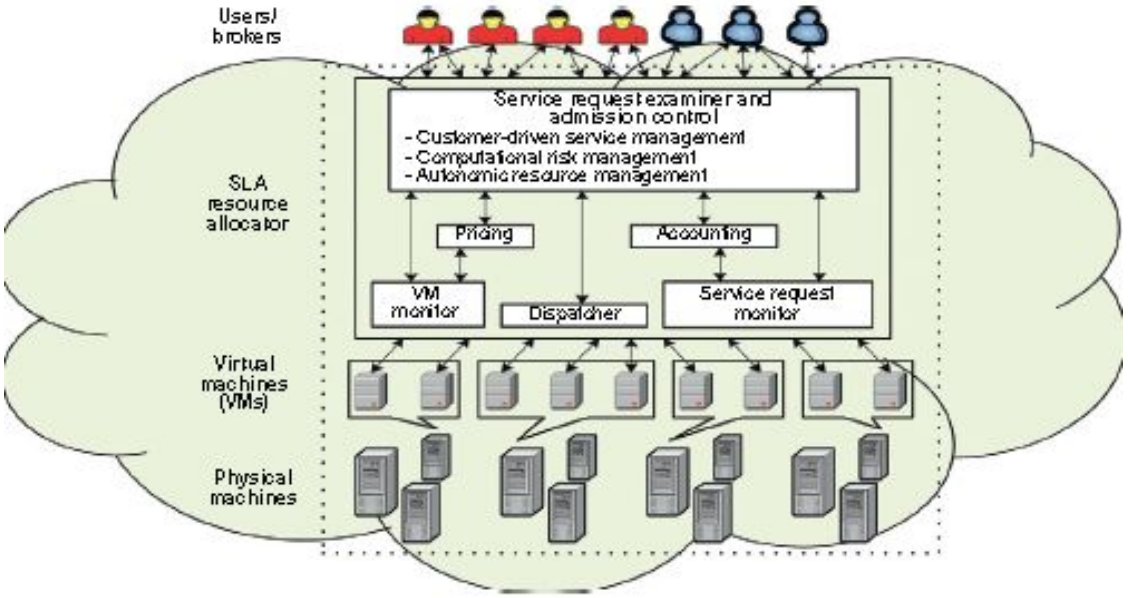
- 1980-90 г.г. В это время основные работы в области виртуализации велись в направлении адаптации этой технологии для персональных ЭВМ и прежде всего для архитектуры Intel x86.
- В 1999 г. компания VMware представила технологию виртуализации систем на базе x86 получившую название VMware Virtual Platform. Первым продуктом реализующим новую технологию было ПО VMware WorkStation (гирервизор на основе хозяйской ЭВМ).
- 2005 г. VMware выпустила первое бесплатное ПО виртуализации десктопов - VMware Player.
- 2006 г. Выпустила ПО VMware ESX Server – первый гипервизор полной виртуализации серверов для архитектуры x86 (native hypervisor – “родной” гипервизор x86)

История виртуализации (3)

- Позднее в "битву"включились такие компании как:
  - Parallels (ранее SWsoft), продукты Parallels Workstation, **Parallels Desktop для Mac** (2006), **Parallels Virtuozzo Containers** (технология виртуализации средствами ОС);
  - Oracle (Sun Microsystems), **VirtualBox** (2008) была приобретена у компании Innotek в 2007;
  - Citrix Systems (**XenSourse**), Xen разработан в кембриджеском университете (там же где и СР-40), первый публичный релиз Xen выпущен в 2003. Гипервизор Xen использует технологию паравиртуализации. В октябре 2007 Citrix купила XenSource и осуществила переименование продуктов Xen;
  - RedHat. Программное обеспечение **KVM** было создано, разрабатывается и поддерживается фирмой Qumranet было создано, разрабатывается и поддерживается фирмой Qumranet, которая была куплена Red Hat за $107 млн 4 сентября 2008 года. После сделки KVM (наряду с системой управления виртуализацией oVirtПосле сделки KVM (наряду с системой управления виртуализацией oVirt) вошла в состав платформы виртуализации RHEV;
  - Корпорация Microsoft вышла на рынок средств виртуализации в 2003 г. с приобретением компании Connectix, выпустив свой первый продукт **Virtual PC** для настольных ПК. Microsoft ***Hyper-V*** (кодовое имя Viridian), — система аппаратной виртуализации для x64-систем на основе гипервизора. Бета-версия *Hyper-V* была включена в x64-версии Windows Server 2008, а законченная версия (автоматически, через Windows Update) была выпущена 26 июня 2008.

История виртуализации (4)

- 2011г. Сформулированы основные модели развертывания и признаки облачных вычислений.
- В основе облачных вычислений лежат технологии виртуализации.



Виртуальная машина

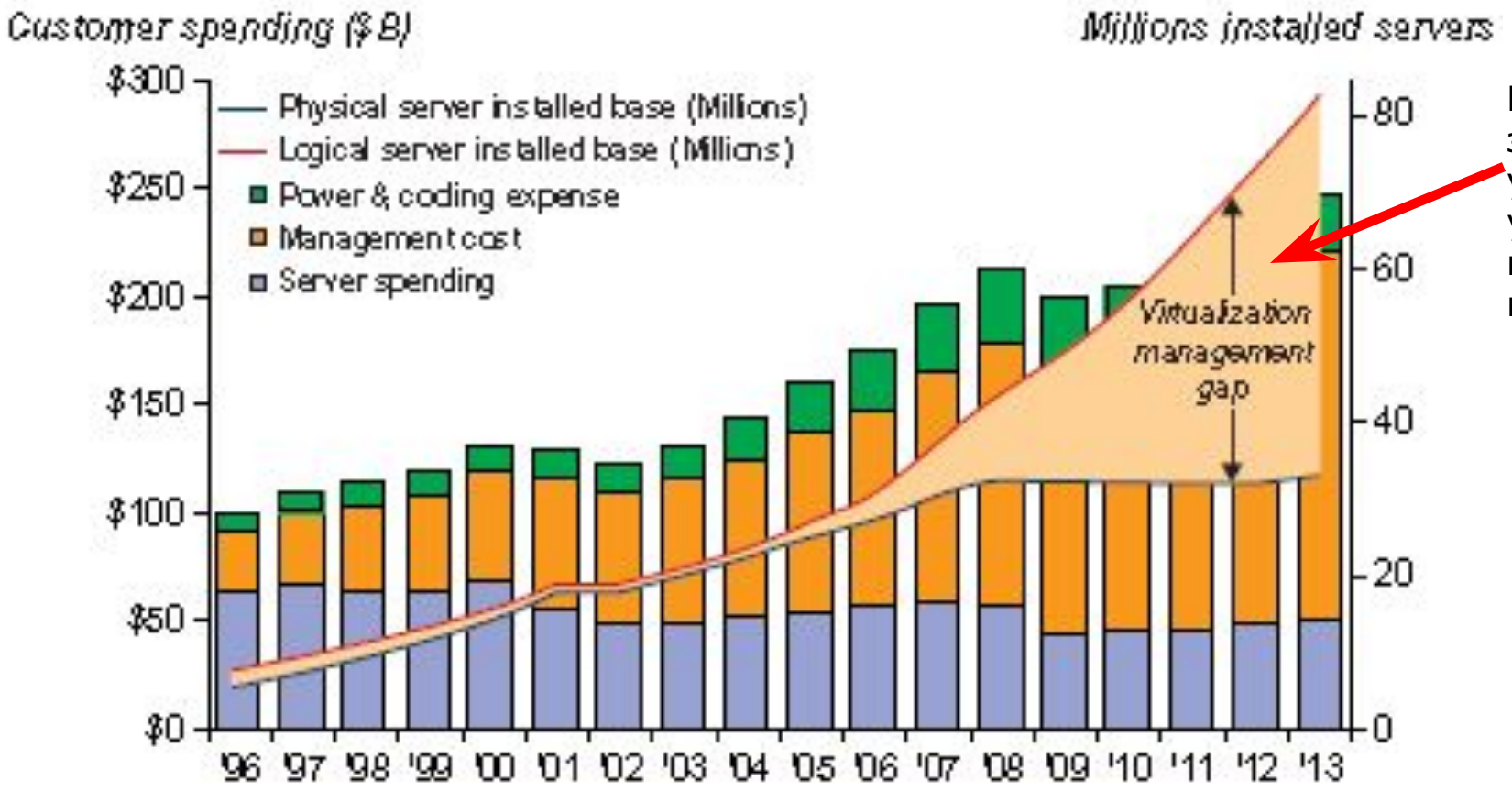
Достоинства:

- Эффективность использования ресурсов
- Масштабируемость
- Простые резервное копирование и миграция
- Гибкость

Недостатки:

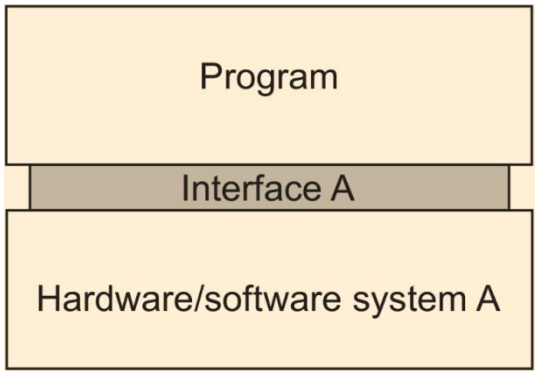
- Проблема в распределении ресурсов при высокой загрузке
- Vendor lockin
- Сложная настройка

Рост стоимости и числа виртуализированных ЦОД в период 1996-2013



Рост доли затрат на управление управления виртуализацией.

Принципы виртуализации

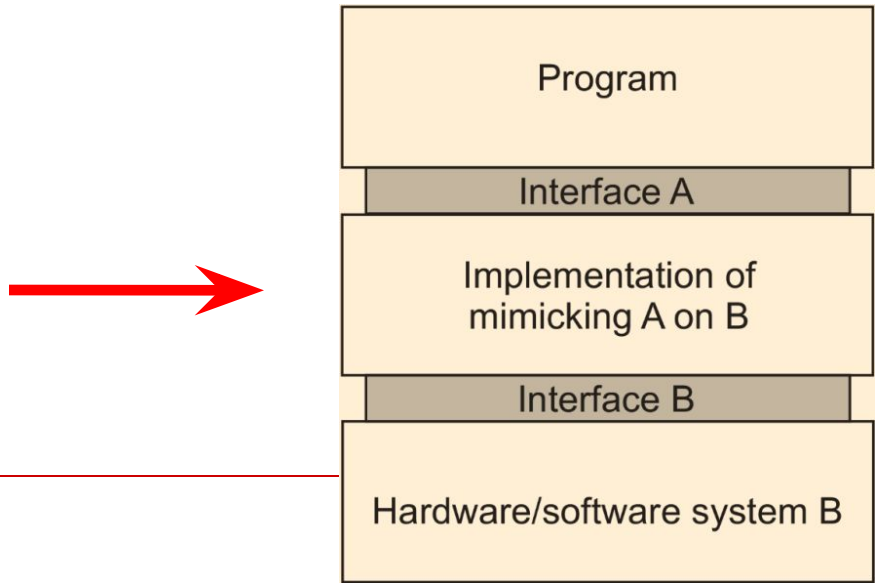


- В реальной программной системе имеется ряд интерфейсов, начиная с базового набора команд ЦПУ и кончая коллекцией API поставляемых с различным ПО современных систем с промежуточным слоем.
- Виртуализация – это по сути замена существующих интерфейсов на интерфейсы имитирующие поведение других систем.

Виртуализация и распределенные системы

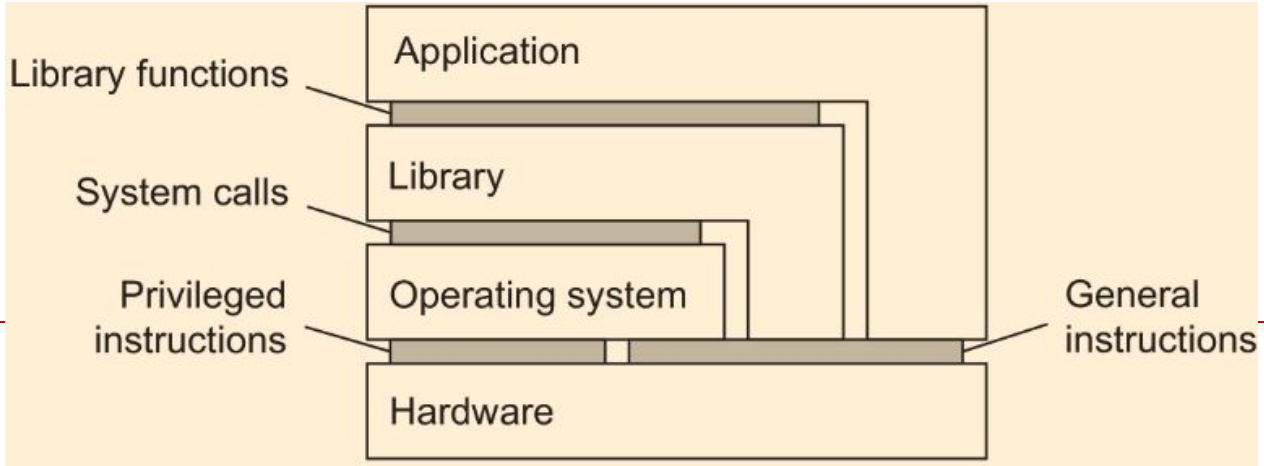
- Рассмотрим случай, когда необдходимо, обеспечить исполнение программы разработанной для аппаратной платформы A, на платформе.
- Для подобной виртуализации (исполнения) программы, созданной для платформы A, для новой платформе B, необходимо обеспечить преобразование интерфейса A (между программой и аппаратной платформой A) в интерфейс для платформы B.

- В распределенных системах, такое преобразование реализуется средствами ПО промежуточного слоя. В том числе и средствами ОС.

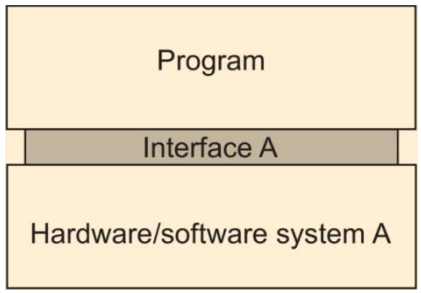


Типы виртуализации.

- Имеется несколько способов реализации виртуализации в зависимости от типа интерфейса компьютерной системы, используемого для этого:
  - Виртуализация на уровне набора команд (ISA). Используется интерфейс между аппаратными средствами и ПО, который представляет собой набор машинных команд;
  - Виртуализация на уровне системных вызовов (эмуляция системных вызовов библиотек функций ОС);
  - Виртуализация на уровне библиотечных вызовов (на уровне API) приложений.

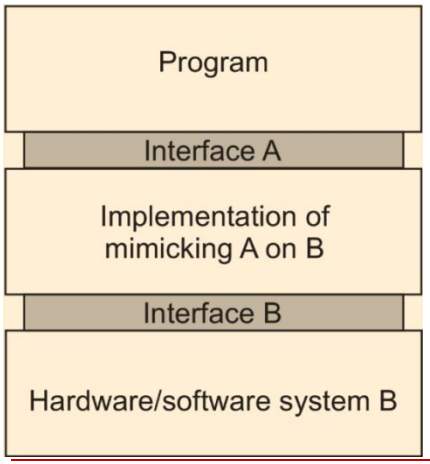


Принципы виртуализации



В компьютерной системе имеется ряд интерфейсов, начиная с базового набора команд ЦПУ и кончая коллекцией API поставляемых с различным ПО современных систем с промежуточным слоем.

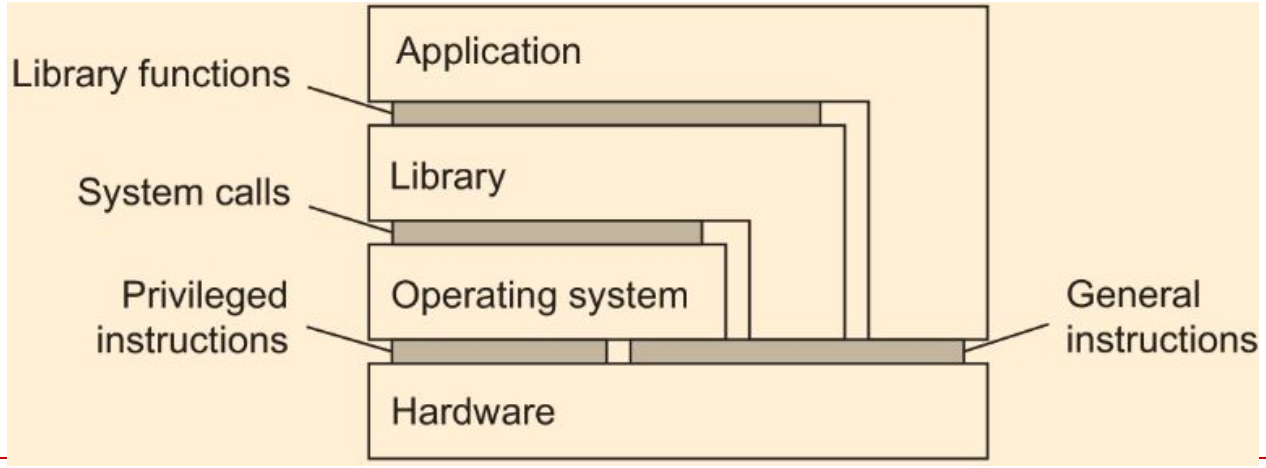
- **Виртуализация – это по сути замена существующих интерфейсов на интерфейсы имитирующие поведение других систем.**



Например, для исполнения под ОС Unix приложений Windows, необходимо использовать эмулятор обеспечивающий преобразование системных вызовов Windows в системные вызовы UNIX.

Виды интерфейсов в компьютерной системе

- В компьютерной системе в общем случае выделяют 4 слоя, связанные тремя типам интерфейсов:
  - Интерфейс между аппаратными средствами и операционной системой (ISA – Instruction Set Archecture).
  - Интерфейс системных вызовов ОС (.
  - Интерфейс вызова библиотечных функций (Library function) (API).



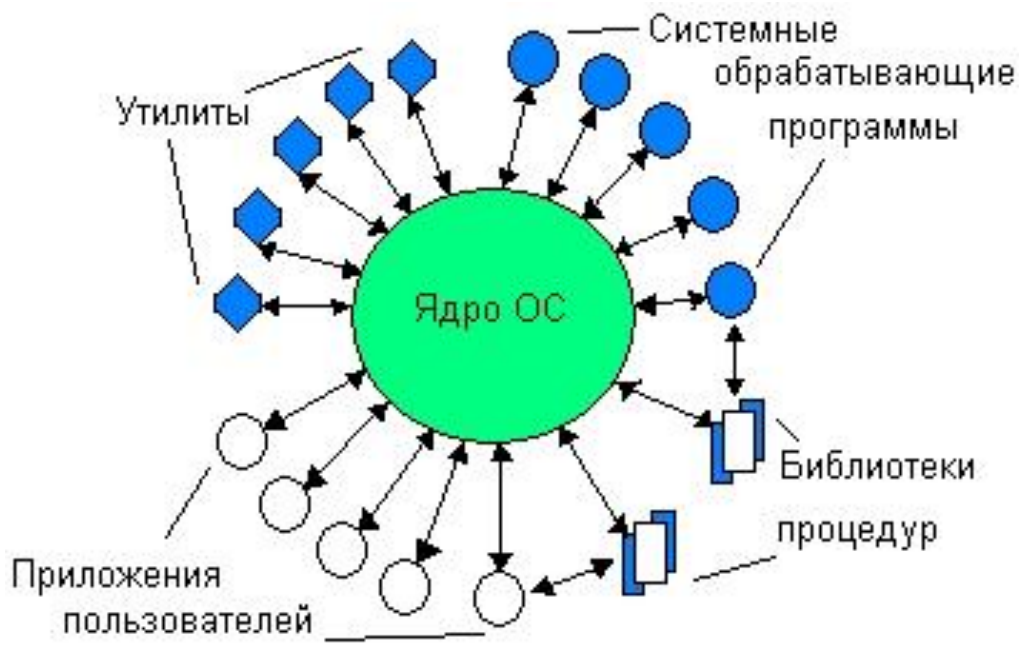
Архитектура операционной системы с ядром в привилегированном режиме

- **Обеспечить привилегии операционной системе невозможно без специальных средств аппаратной поддержки. Аппаратура компьютера должна поддерживать как минимум два режима работы — пользовательский режим (user mode) и привилегированный режим, который также называют режимом ядра (kernel mode), или режимом супервизора (supervisor mode). Подразумевается, что операционная система или некоторые ее части работают в привилегированном режиме, а приложения — в пользовательском режиме.**
- **Так как ядро выполняет все основные функции ОС, то чаще всего именно ядро становится той частью ОС, которая работает в привилегированном режиме**

Смена режимов при выполнении системного вызова к привилегированному ядру

- **Архитектура ОС, основанная на привилегированном ядре и приложениях пользовательского режима, стала, по существу, классической.**
- **Ее используют многие популярные операционные системы, в том числе многочисленные версии UNIX, VAX VMS, IBM OS/390, OS/2, и с определенными модификациями — Windows NT/2000/2003/XP.**

Способы реализации прикладных программных сред



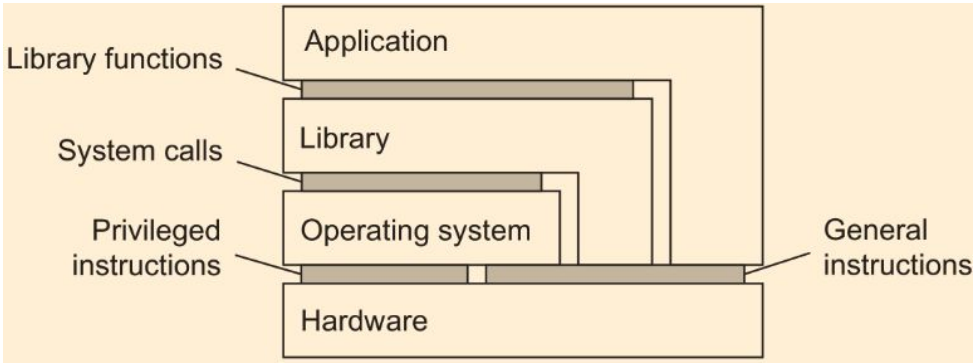
Создание полноценной прикладной среды, полностью совместимой со средой другой операционной системы, является сложной задачей, тесно связанной со структурой операционной системы.

Существуют различные варианты построения множественных прикладных сред:

- в виде обычного приложения (UNIX).
- в виде серверов пользовательского режима (Windows NT/2000/2003/XP);
- средства организации прикладных сред встроены глубоко в операционную систему (OS/2)

Типы виртуализации

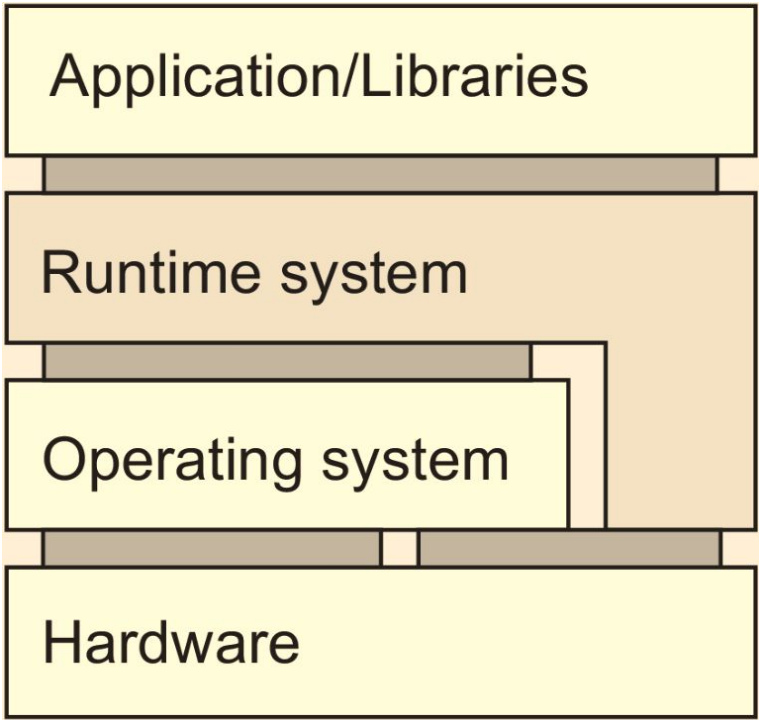
- Тип виртуализации определяются типом интерфейса на котором она выполняется:
- Виртуализация на уровне набора команд (интерфейс ISA). При этом надо учитывать, что в состав набора команд входят привилегированные и непривилегерованные команды;
- Виртуализация на уровне системных вызовов (интерфейс системных вызовов);
- Виртуализация на уровне библиотечных вызовов (на интерфейсе вызова библиотечных функций или API).



Два способа реализации виртуализации

- Суть виртуализации – имитация поведения интерфейса компьютерной системы. Существуют два способа реализации такой имитации:
  - Средствами среды исполнения приложения.
  - Средствами экранирования аппаратных средств машины с помощью ПО преобразования в набор команд той же или другой архитектуры, используемого в качестве интерфейса. Это ПО получило название **VMM – Virtual Machine Monitor** (Монитор виртуальных машин).

Виртуализация на уровне среды исполнения прикладной программы. Виртуальная машина одного процесса.



- В этом случае среда исполнения обеспечивает либо:
  - интерпретацию кода приложения построенного на основе некоторого абстрактного набора команд. Пример: Java VM, реализующая интерпретацию байт кода в машинные команды аппаратной платформы.
  - эмуляцию кода приложения, написанного для одной ОС в код приложения предназначенного для другой ОС. Пример: ПО Wine для ОС Linux, позволяющее исполнять программы для Windows.
- Оба способа исполнения работают только с непривилегированными командами процессора, поэтому они не требуют специальных условий для испонения кода.
- Такая виртуализация получила название **виртуальной машины одного процесса**.

Требования к архитектуре ЭВМ, для поддержки виртуализации

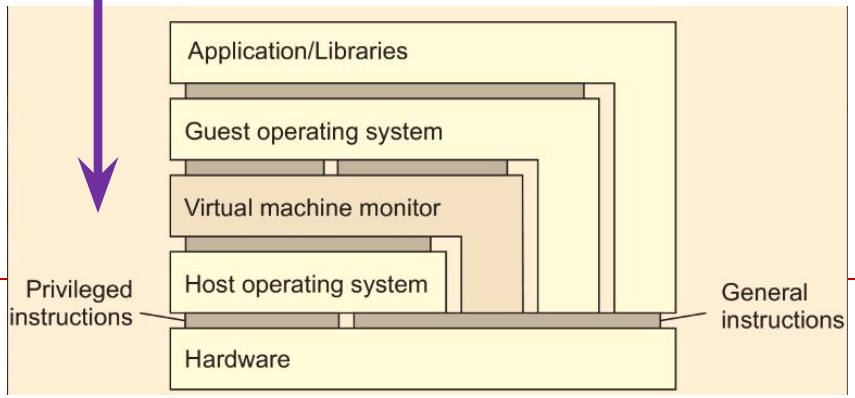
- В 1974 году двое ученых из Калифорнийского университета (Лос-Анджелес), работающих в компьютерной сфере, Геральд Попек (Gerald Popek) и Роберт Голдберг (Robert Goldberg), опубликовали основополагающую статью («Formal Requirements for Virtualizable Third Generation Architectures»), в которой дали точный перечень тех условий, которым должна отвечать компьютерная архитектура, чтобы иметь возможность эффективно поддерживать виртуализацию.
- Такими требованиями являются:
  - 1. **Безопасность** — у гипервизора должно быть полное управление виртуализированными ресурсами.
  - 2. **Эквивалентность** — поведение программы на виртуальной машине должно быть идентичным поведению этой же программы, запущенной на реальном оборудовании.
  - 3. **Эффективность** — основная часть кода в виртуальной машине должна выполняться без вмешательства гипервизора.

Проблемы виртуализации в архитектуре Intel x86

- В наборе команд Intel x86 (включая и x64) имеются команды способные изменить состояние процессора исполняемые в пользовательском режиме. Различают команды:
  - поведение которых зависит от режима исполнения (behavior sensitive);
  - которые влияют на управление (control senitive)
- Например команда **POPF**, которая устанавливает флаг разрешения прерывания, только когда работает в режиме ядра, но относится к общим командам.
- Имеется еще **17** команд являющихся не привилегированными и не перехватываемых операционной системой, но чувствительных к режиму исполнения.
- Для решения проблемы виртуализации архитектуры Intel используется дополнительное ПО, получившее название **монитор виртуальных машин**.

Монитор виртуальных машин

- Идея монитора виртуальных машин не нова. Она была предложена еще в 1974 г. в работе Г. Popek, P. Goldberg, «Formal Requirements for Virtualizable Third Generation Architectures».
- Монитор виртуальных машин обеспечивает интерфейс между гостевой ОС и хозяйской ОС. При этом подавляющая часть кода приложения, гостевой ОС, а также хозяйской ОС выполняется напрямую на аппаратных средствах физической машины (все общие команды и часть привилегированных).
- И лишь небольшая часть кода, которую составляют некоторые привилегированные команды перехватываются монитором виртуальных машин, модифицируются им, и затем передаются на исполнение аппаратным средствам.

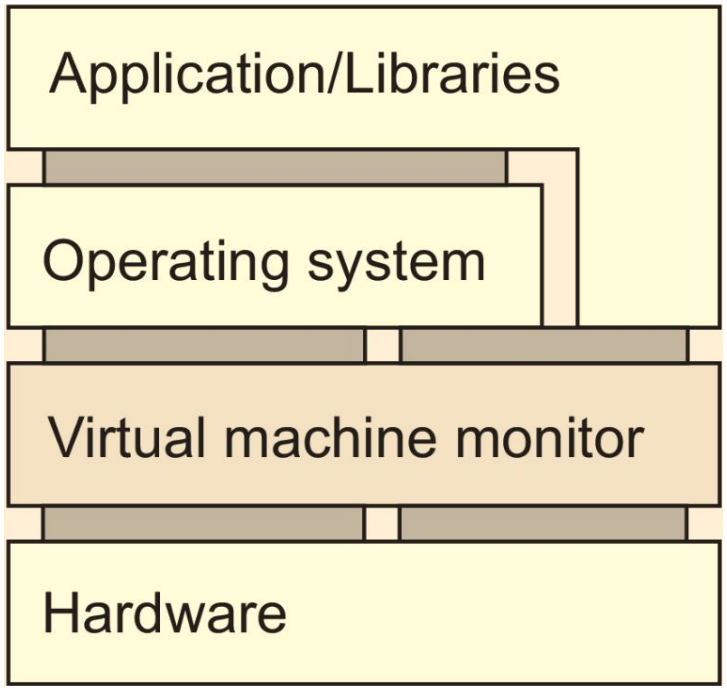


Монитор виртуальных машин получил название Гипервизор.
Различают: гипервизоры I и II типов.

Два подхода к реализации монитора виртуальных машин

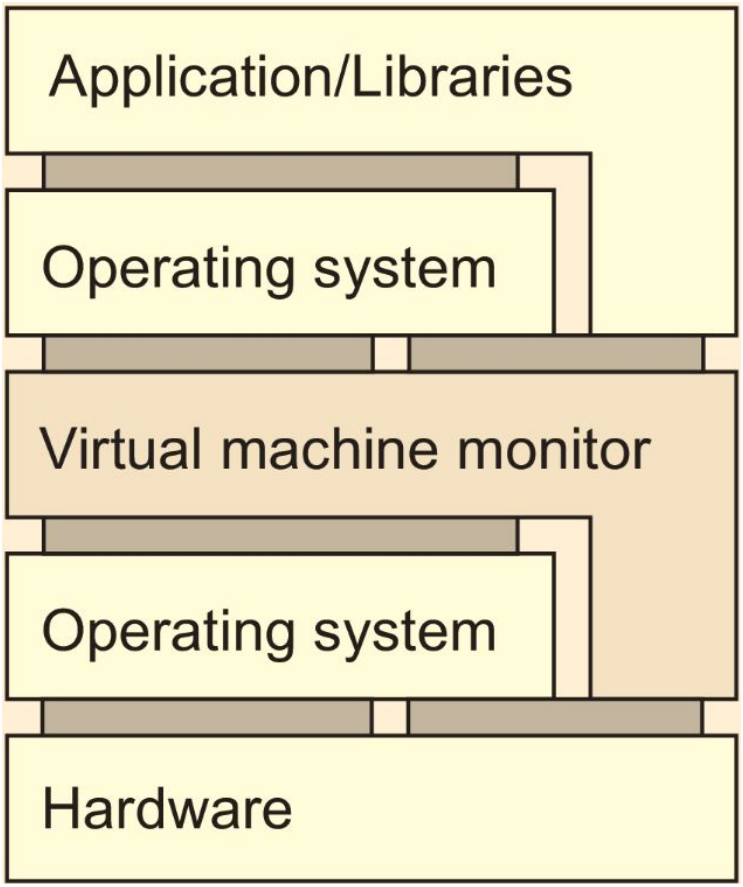
- Имеются два подхода к построению мониторов виртуальных машин:
  - **Эмуляция** всех инструкций, что влечет за собой резкое падение производительности. Этот подход в измененном виде используется современными **мониторами виртуальных машин**.
  - Использование **паравиртуализации**, которая предполагает модификацию гостевых ОС, таким образом, чтобы нейтрализовать, либо полностью исключить влияние “чувствительных” команд, средствами хозяйской ОС.

Монитор виртуальных машин исполняемый как отдельная ОС (Гипервизор первого типа)



- На аппаратные средства устанавливается специализированная ОС (гипервизор) предназначенная для виртуализации на уровне набора команд аппаратной платформы (Vmware ESX).
- Такой монитор получил название “родного” (naitive) монитора виртуальных машин.
- Достоинства данной технологии заключаются в:
  - отсутствии потребности в хостовой ОС – ВМ, устанавливаются фактически на "голое железо", а аппаратные ресурсы используются более эффективно.
- Недостатки:
  - необходимость поддержки в гипервизоре собственных драйверов внешних устройств, из-за чего возникают высокие дополнительные накладные расходы на используемые аппаратные ресурсы,
  - отсутствие учета особенностей гостевых ОС, меньшая, чем нужно, гибкость в использовании аппаратных средств

Монитор виртуальных машин исполняемый в среде хозяйской ОС (Гипервизор второго типа)



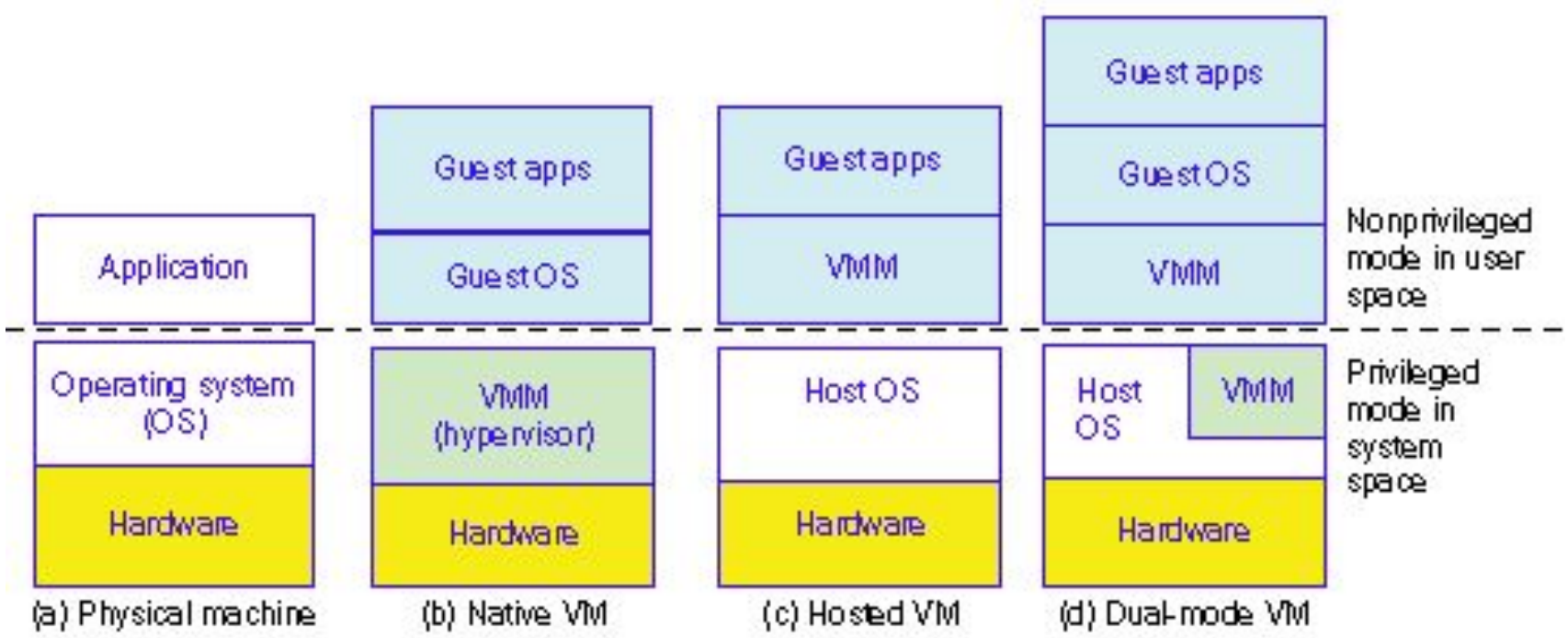
- Монитор виртуальных машин работает в рамках хозяйской ОС.
- В процессе своей работы он модифицирует код гостевых ОС с целью обеспечения их виртуализации на уровне набора команд.
- При этом код содержащий “чувствительные” команды переписывается на лету (Vmware WorkStation).

Паравиртуализация

- техника виртуализации, при которой гостевые операционные системы подготавливаются для исполнения в виртуализированной среде, для чего их ядро незначительно модифицируется.
- Операционная система взаимодействует с программой гипервизора, который предоставляет ей гостевой API, вместо использования напрямую таких ресурсов, как таблица страниц памяти.
- Этот подход не только поддерживает высокую производительность, но и позволяет формировать гетерогенную среду, в которой работает несколько гостевых операционных систем.
- Накладные расходы составляют 5-10%

Архитектура физической машины (a) и три архитектуры виртуализации (b-d)

- Монитор виртуальных машин (VMM – Virtual Machine Monitor)



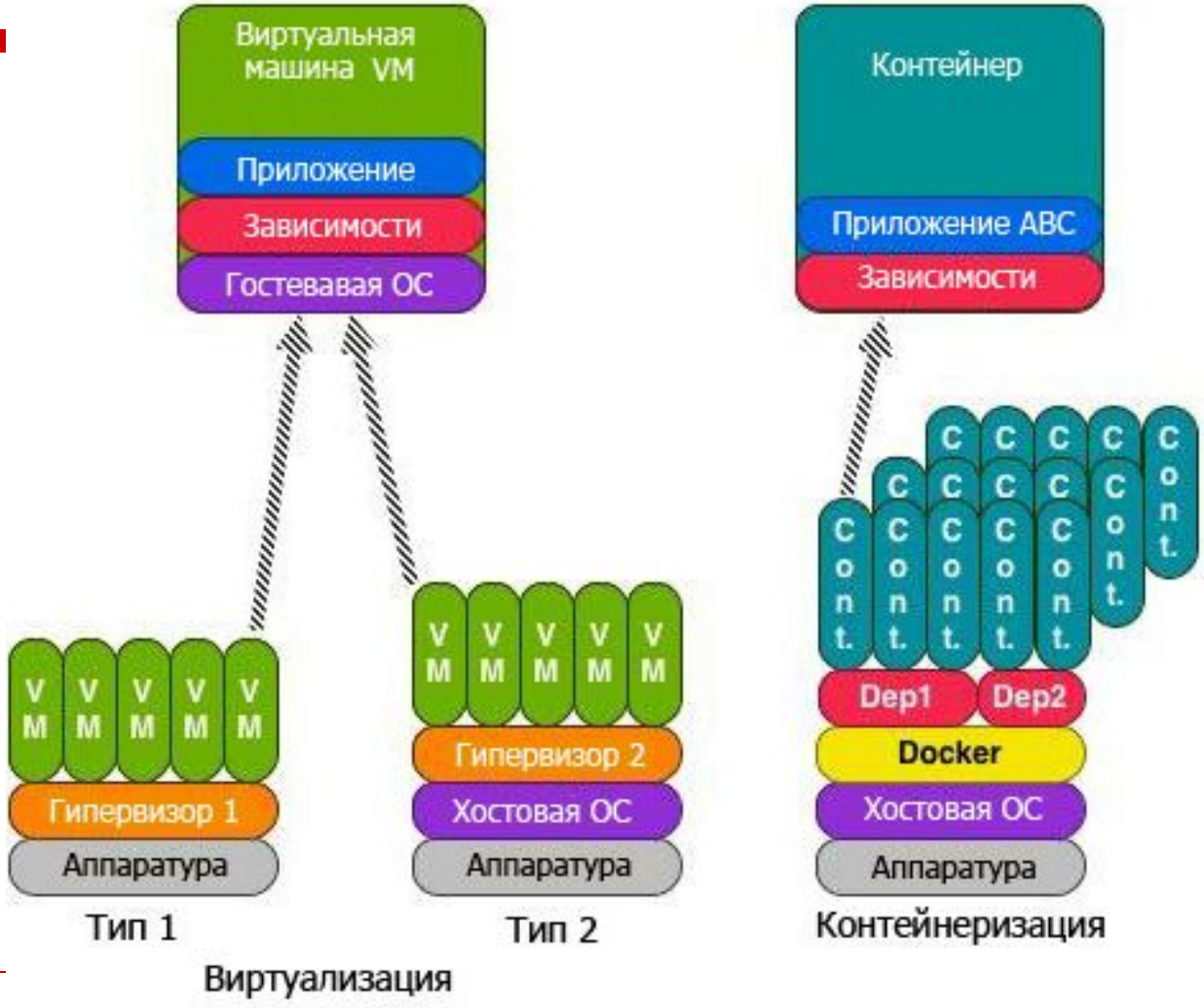
Виртуализация на уровне операционной системы

- — виртуализирует физический сервер на уровне ОС, позволяя запускать изолированные виртуальные серверы называемые Виртуальные Частные Серверы (Virtual Pivate Servers, VPS) или Контейнеры (Container, CT).
- Виртуализация на уровне операционной системы имеет минимальные накладные расходы и обеспечивает самую высокую степень консолидации, однако эта технология не позволяет запускать ОС с ядрами, отличными от ядра базовой ОС.
- **Накладные потери производительности контейнеров составляют 3%**

Контейнерная виртуализация в Linux

- Экземпляры пространств пользователя (часто называемые контейнерами или зонами) с точки зрения пользователя полностью идентичны реальному серверу, но они в своей работе используют один экземпляр ядра операционной системы.
- Для linux-систем, эта технология может рассматриваться как улучшенная реализация механизма chroot.
- Ядро обеспечивает полную изолированность контейнеров, поэтому программы из разных контейнеров не могут воздействовать друг на друга.
- **Наиболее распространены сейчас OpenVZ, LXC, FreeBSD jail и Solaris Containers**

Виртуализация против контейнеризации



Виртуализация и распределенные системы

- Одной из важнейшей причиной, но не единственной, появления технологии виртуализации, явилась необходимость выполнения унаследованного ПО.
- По мере того как стоимость аппаратных средств снижалась, потребность в виртуализации снижалась и не была актуальной до 90-х г.г. XX века.
- Однако в конце 90-х необходимость в виртуализации вновь появилась, в связи с ростом темпов обновления архитектур процессоров, при том, что темпы сменяемости ПО не изменились.
- Виртуализация может обеспечить практически немедленный перенос унаследованного ПО на новые аппаратные платформы. При этом каждое приложение (сервис), включая требуемую ОС и все необходимые ему библиотеки может работать на отдельной виртуальной машине. А группа виртуальных машин может быть развернута на единой аппаратной платформе (физическом сервере).
- При виртуализации обеспечивается высокая степень переносимости ПО между физическими аппаратными платформами, что является важнейшей причиной почему виртуализация играет ключевую роль во многих распределенных системах.

Основы облачных вычислений

Что это такое ? (1)

- **Пользователь:**

  1. Сегодня под облачными вычислениями обычно понимают возможность получения необходимых вычислительных мощностей по запросу из сети.

  2. Облачные вычисления представляют собой динамически масштабируемый способ доступа к внешним вычислительным ресурсам в виде сервиса, предоставляемого посредством Интернета, при этом пользователю не требуется никаких особых знаний об инфраструктуре "облака" или навыков управления этой "облачной" технологией.

Что это такое ?(2)

- **Руководитель ИТ:**

  1. Это новый подход, позволяющий снизить сложность ИТ-систем, благодаря применению широкого ряда эффективных технологий, управляемых самостоятельно и доступных по требованию в рамках виртуальной инфраструктуры, а также потребляемых в качестве сервисов
  2. "Облако" является новой бизнес-моделью для предоставления и получения информационных услуг.

Системный Архитектор:

Cloud computing – это программно-аппаратное обеспечение, доступное пользователю через Интернет или локальную сеть в виде сервиса, позволяющего использовать удобный интерфейс для удаленного доступа к выделенным ресурсам (вычислительным ресурсам, программам и данным).

Поставщик ПО и ИТ услуг:

Это способ продажи программного обеспечения и других информационных ресурсов предполагающий оплату только за полученный и использованный объем обслуживания.

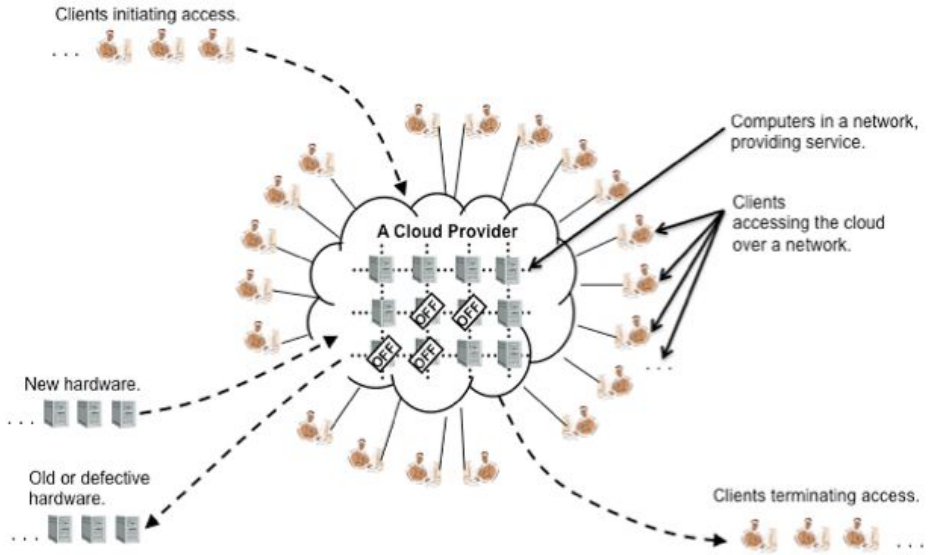
Понятие облачные вычисления

- Под облачными вычислениями мы понимаем программно-аппаратное обеспечение, доступное пользователю через Интерне или локальную сеть в виде сервиса, позволяющего использовать удобный интерфейс для удаленного доступа к выделенным ресурсам (вычислительным ресурсам, программам и данным).

SaaS сервисы Google

Признаки облачных вычислений

- Самообслуживание по требованию в условиях мультиарендности.
- Широкий (универсальный и высокоскоростной) сетевой доступ.
- Объединение ресурсов в пулы.
- Мгновенная эластичность ресурсов (мгновенная масштабируемость).
- Измеряемый сервис (учет реального объема потребляемых ресурсов).



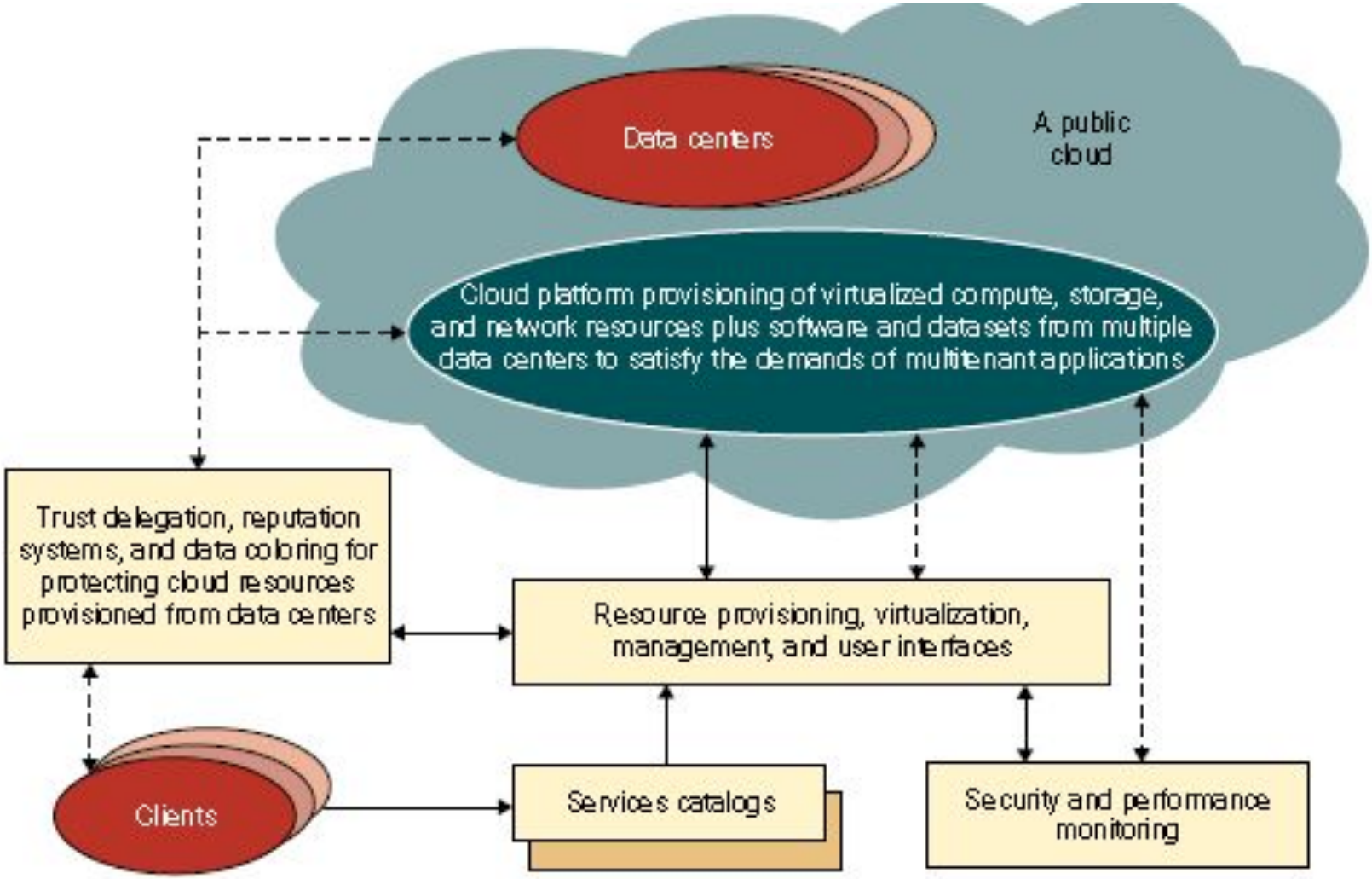
Облачные вычисления – результат синтеза целого ряда технологий и подходов

Облачные технологии основываются на использовании:
- сервис-ориентированной архитектуры ИС;
- технологий динамического Web (Web 2.0);
- современных систем разработки ПО;
- подходов и методов построения систем масштабируемых кластерных вычислений;
- методов и средств создания grid-систем;
- технологий виртуализации;
- модель предоставления сервиса IaaS.

Один облачный сервер обходится дешевле, чем сервер, приобретенный и установленный самой компанией:

- более разумное расходование электроэнергии + размещение дата-центров в регионах с более выгодными тарифами.
- оптимизация использования рабочей силы (один администратор - не 100, а 1000 серверов).
- более высокая безопасность и надежность

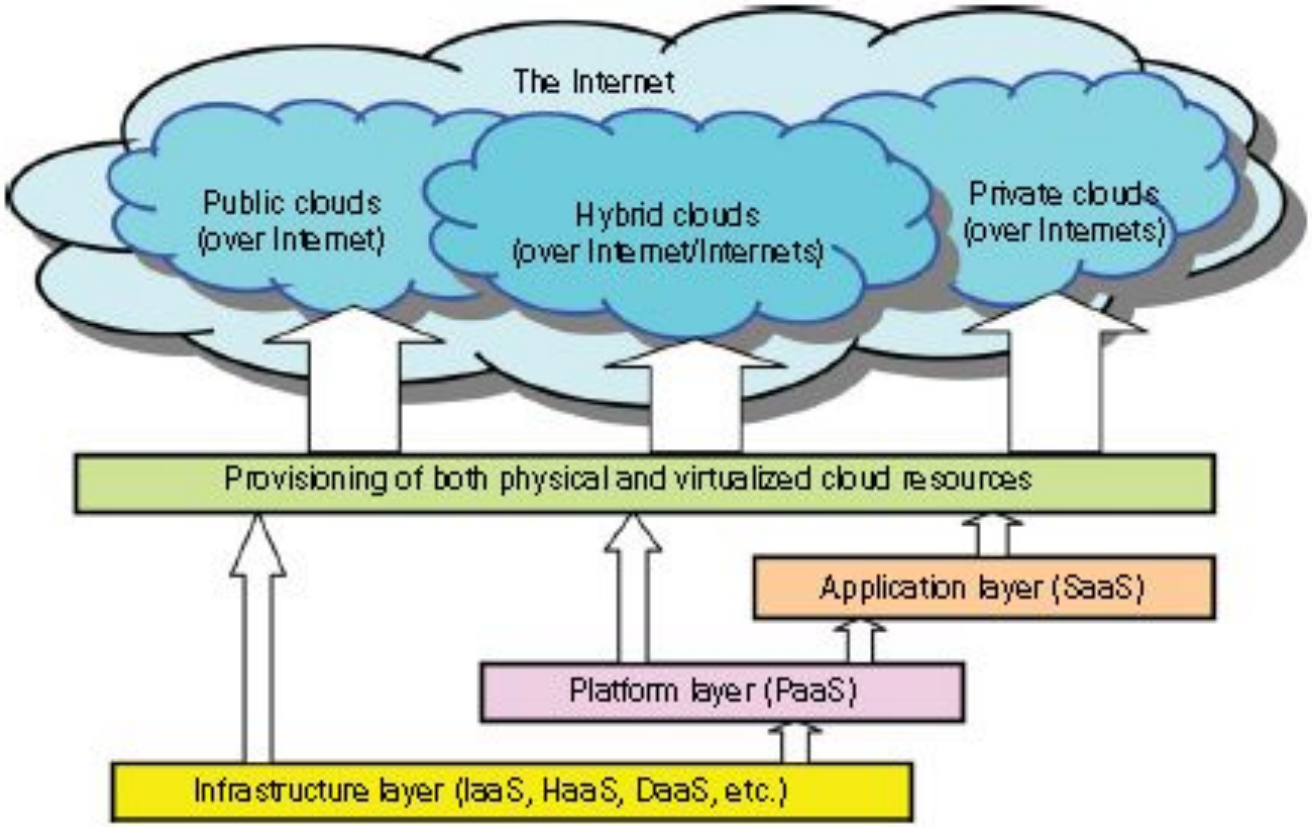
Общая архитектура облачных вычислений



Общая архитектура облаков

- Включает следующие компоненты:
  - ЦОДы – центры обработки данных;
  - платформы снабжения виртуализированными вычислительными, сетевыми ресурсами и ресурсами СХД и СУБД из нескольких ЦОДов для обеспечения запросов от мультиарендных приложений;
  - управление снабжением ресурсами облака;
  - управление доступом к облаку;
  - каталоги предоставляемых облаком сервисов;
  - мониторинг производительности и безопасности облака.

Многоуровневая архитектура облака



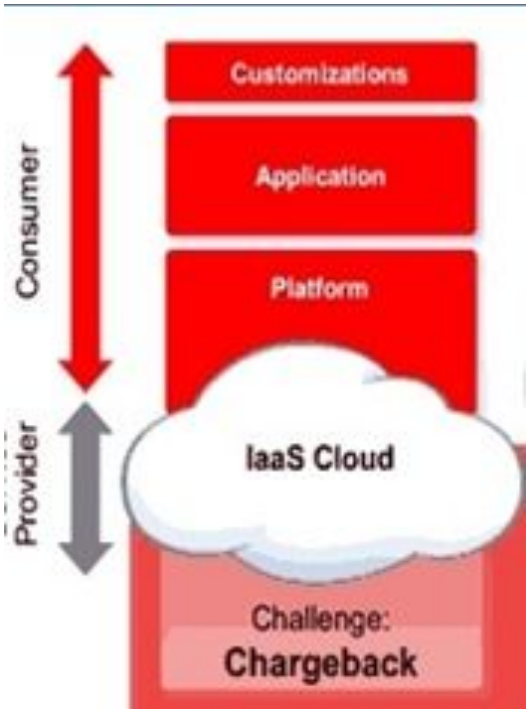
Архитектура облака разработана как 3-х уровневая:

- Уровень инфраструктуры (IaaS);
- Уровень платформы (PaaS);
- Уровень приложений (SaaS).

- Используется для описания моделей обслуживания пользователей и развертывания

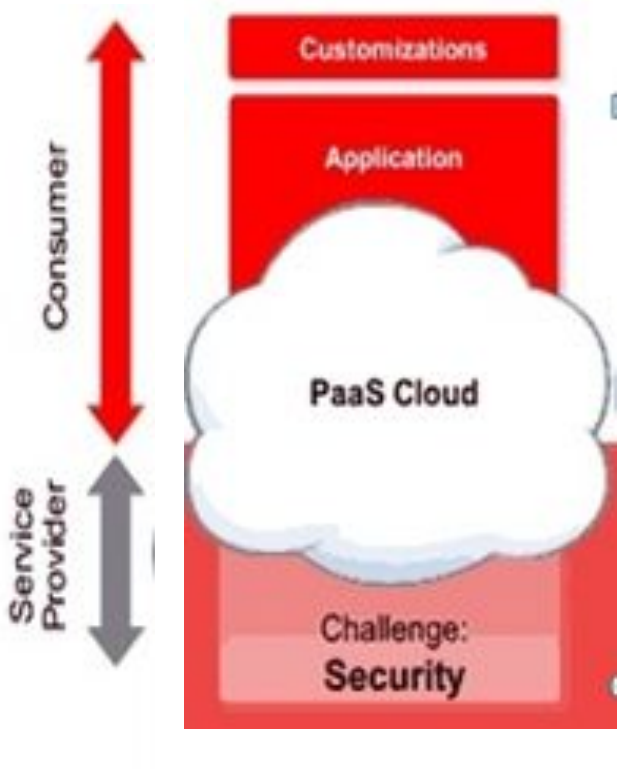
Сервисные модели облачных вычислений

Уровень инфраструктуры (IaaS)



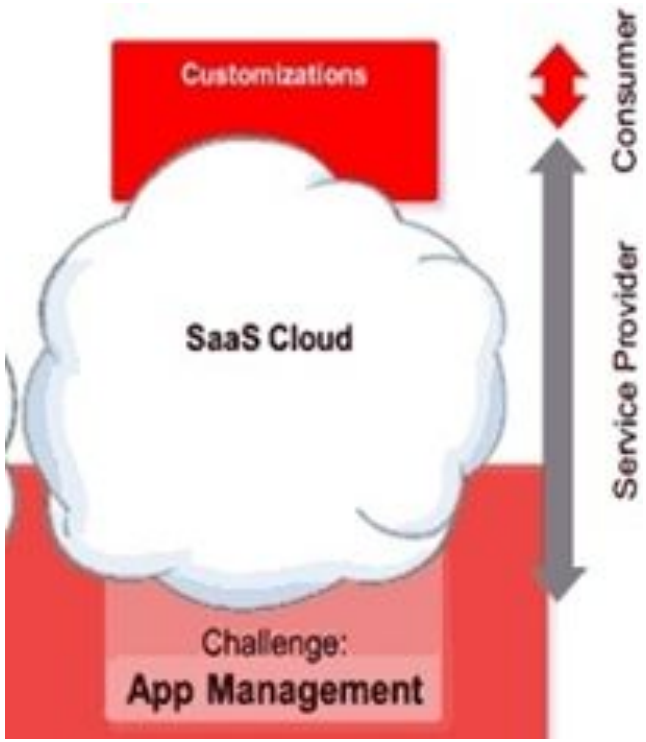
- Служит основой для создания других уровней облака.
- Строится на основе виртуализированных вычислительных, сетевых и ресурсов систем хранения.
- Виртуализация обеспечивает автоматическое снабжение ресурсами и оптимизирует процесс управления ими.

Уровень платформы (PaaS)



- Предназначен для общего в том числе и для повторного использования коллекции программных ресурсов.
- Это уровень обеспечивает для пользователей среду для разработки своих приложений, для тестирования правильности их функционирования, а также для оценки результатов вычислений и уровня производительности.
- Он должен гарантировать пользователям необходимый уровень масштабируемости, независимости и защищенности.
- В некотором смысле виртуализированный уровень платформы служит системным промежуточным слоем между инфраструктурным уровнем и уровнем приложений.

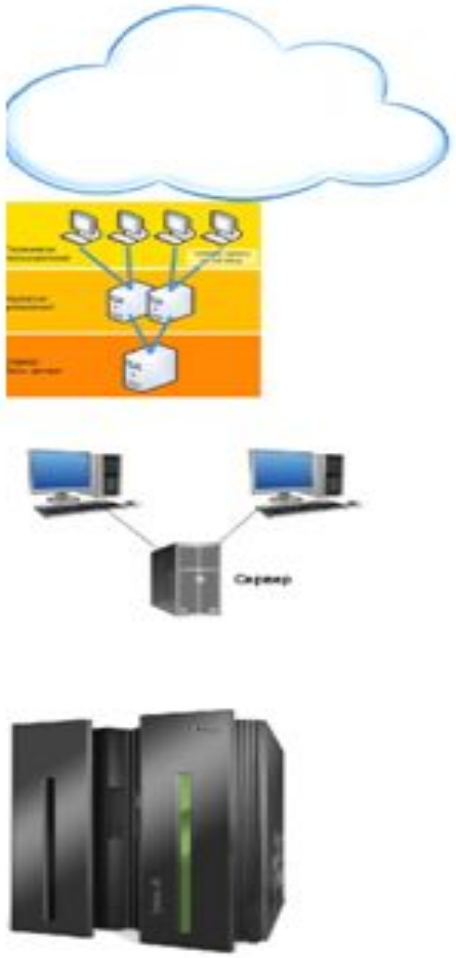
Уровень приложений (SaaS)



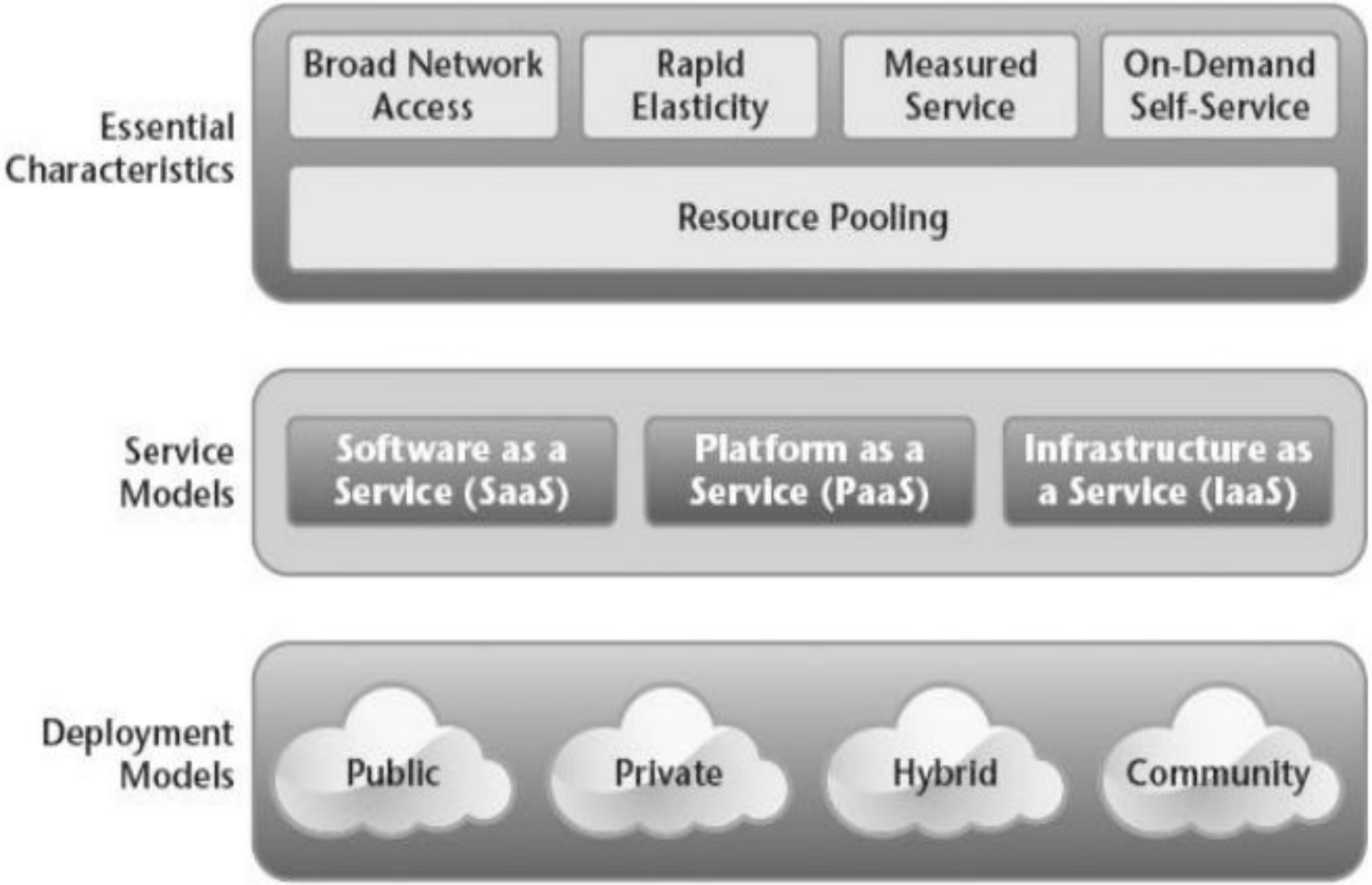
- Формируется совокупностью всех программных модулей необходимых для выполнения приложений SaaS.
- Сервисные приложения обслуживающие этот уровень включают:
  - Управление ежедневными офисными работами, такими как, поиск информации, обработка документов, планирование рабочего времени, аутентификация.
- Это уровень широко используется для автоматизации деловых процессов, работе с потребителями, выполнения финансовых транзакций, управления отношениями с партнерами и потребителями, управлениями цепочками поставок.
- Следует отметить, что не все облачные сервисы ограничиваются только этим одним уровнем. Существует много приложений использующих ресурсы нескольких уровней.

Облачные вычисления как эволюция архитектуры корпоративных приложений

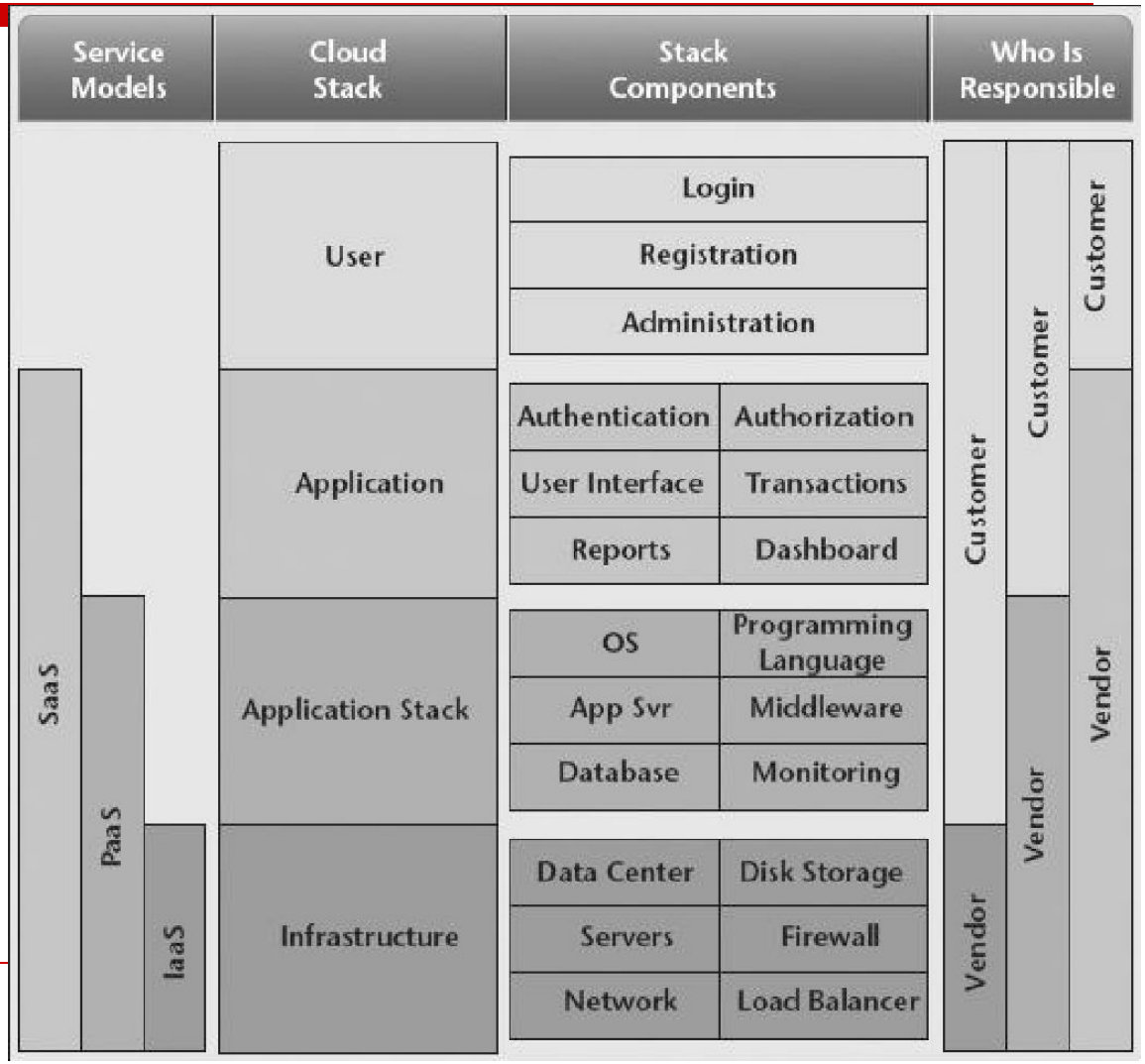
- Облачные вычисления – это следующий шаг в эволюции архитектуры построения информационных систем.
- Стадии эволюции архитектур ИС:
- Монолитная архитектура приложений и целых информационных систем, когда данные и приложения работали на одном компьютере
- Архитектура "клиент-сервер"
- ЦОД - виртуализация ресурсов (серверов, хранилищ данных, сетевого оборудования и клиентских рабочих мест).
- Облачные вычисления – системы позволяющие создавать ИС нужной конфигурации по требованию потребителя.



Определение облачных вычислений по NIST



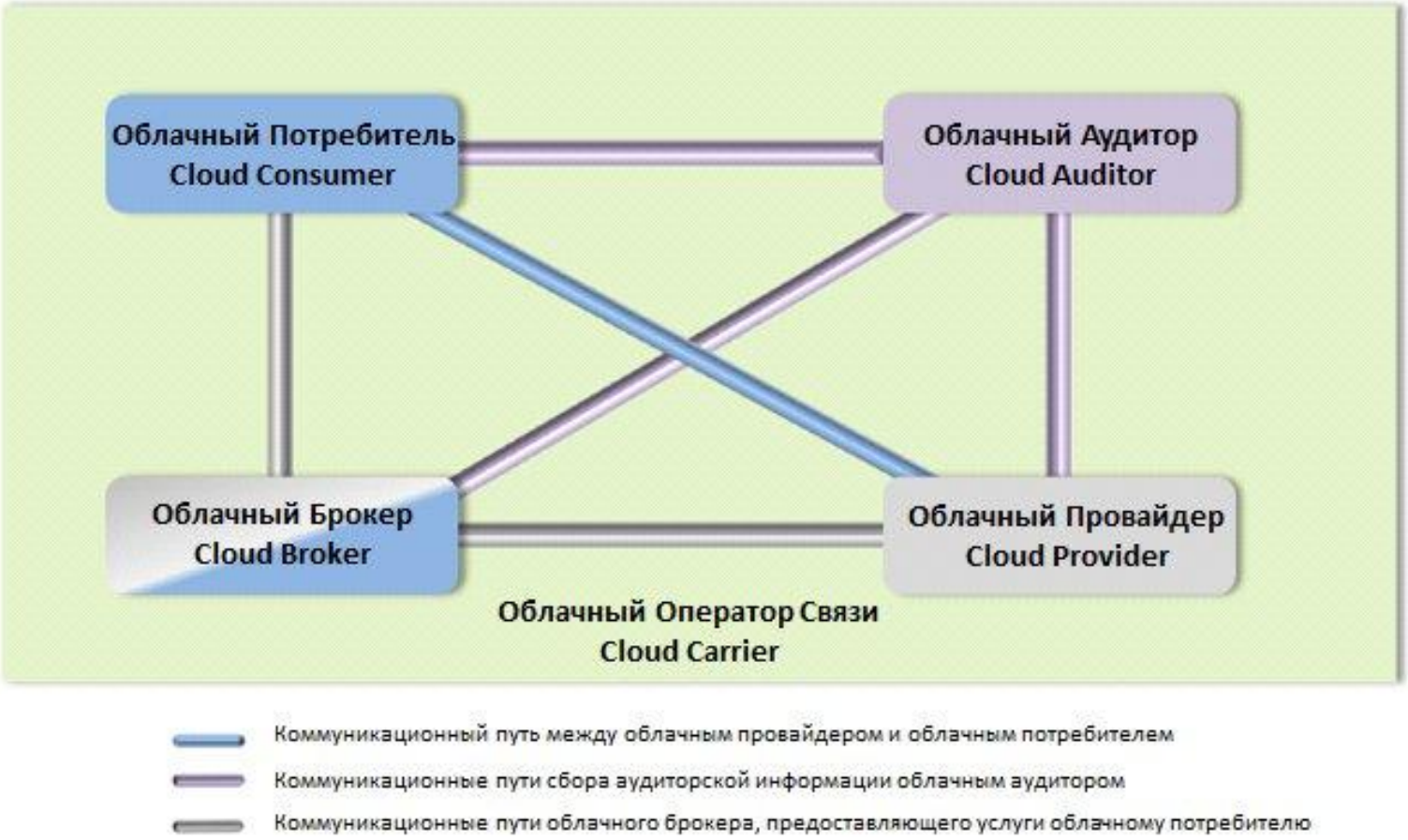
Облачный стек



Архитектурная модель. Участники процесса.

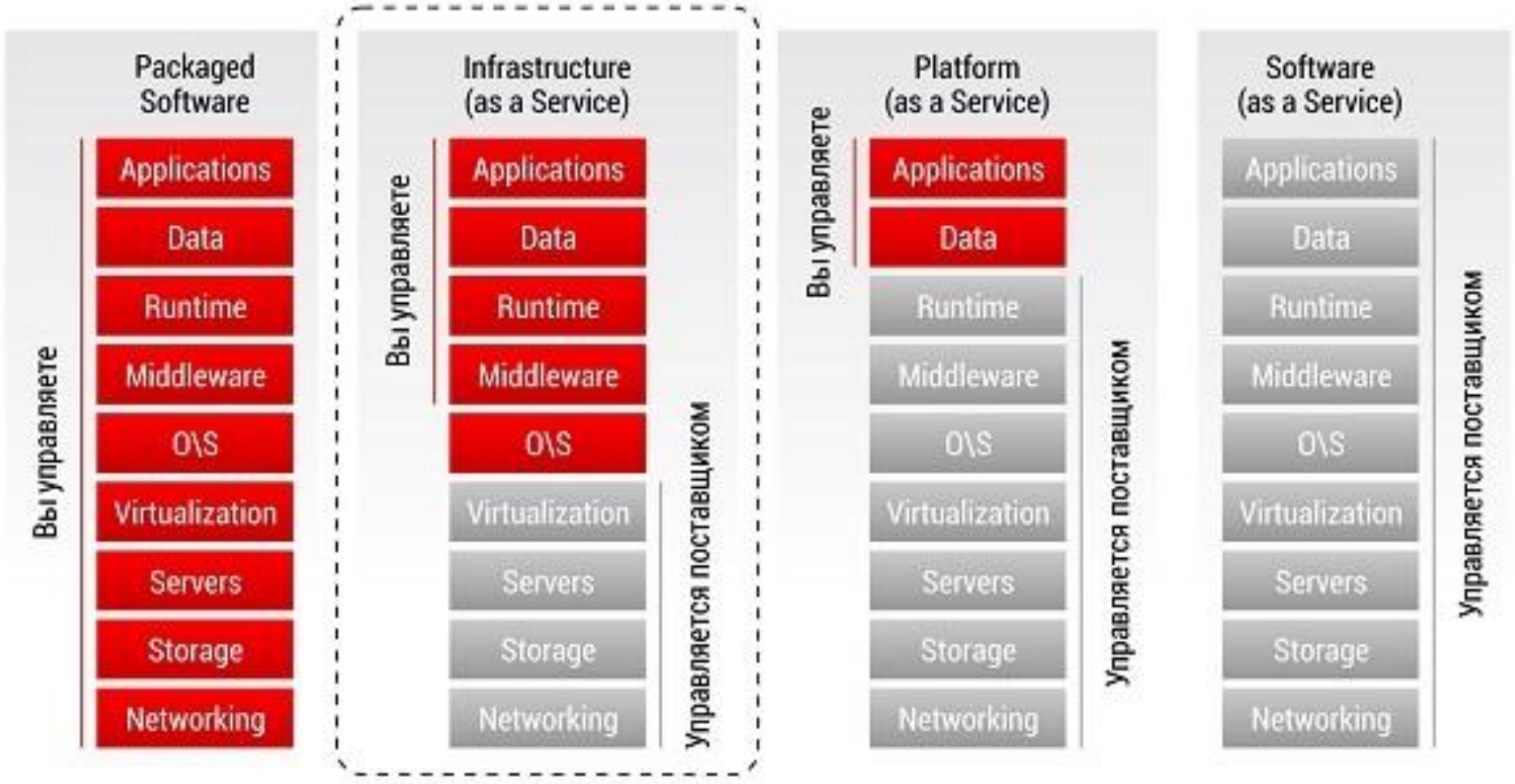
Роль	Определение
Облачный Потребитель **Cloud Consumer**	**Лицо или организация, поддерживающая бизнес-отношения и использующая услуги *Облачных Провайдеров.***
Облачный Провайдер **Cloud Provider**	**Лицо, организация или сущность, отвечающая за доступность облачной услуги для *Облачных Потребителей*.**
Облачный Аудитор **Cloud Auditor**	**Участник, который может выполняет независимую оценку (assessment) облачных услуг, обслуживания информационных систем, производительности и безопасности реализации облака.**
Облачный Брокер **Cloud Broker**	**Сущность, управляющая использованием, производительностью и предоставлением облачных услуг, а также устанавливающая отношения между *Облачными Провайдерами* и *Облачными Потребителями*.**
Облачный Оператор Связи **Cloud Carrier**	**Посредник, предоставляющий услуги подключения и транспорт (услуги связи) доставки облачных услуг от *Облачных Провайдеров* к *Облачным Потребителям*.**

Взаимодействие между ролями в облачных вычислениях



Облачный Провайдер - высокоуровневый взгляд

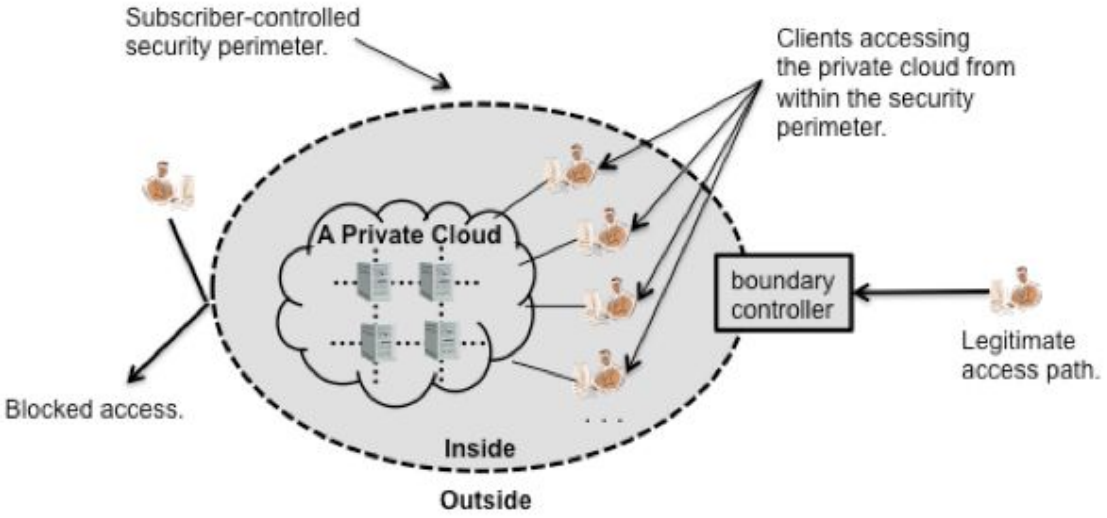
Кто чем управляет в облаке



Типы облаков

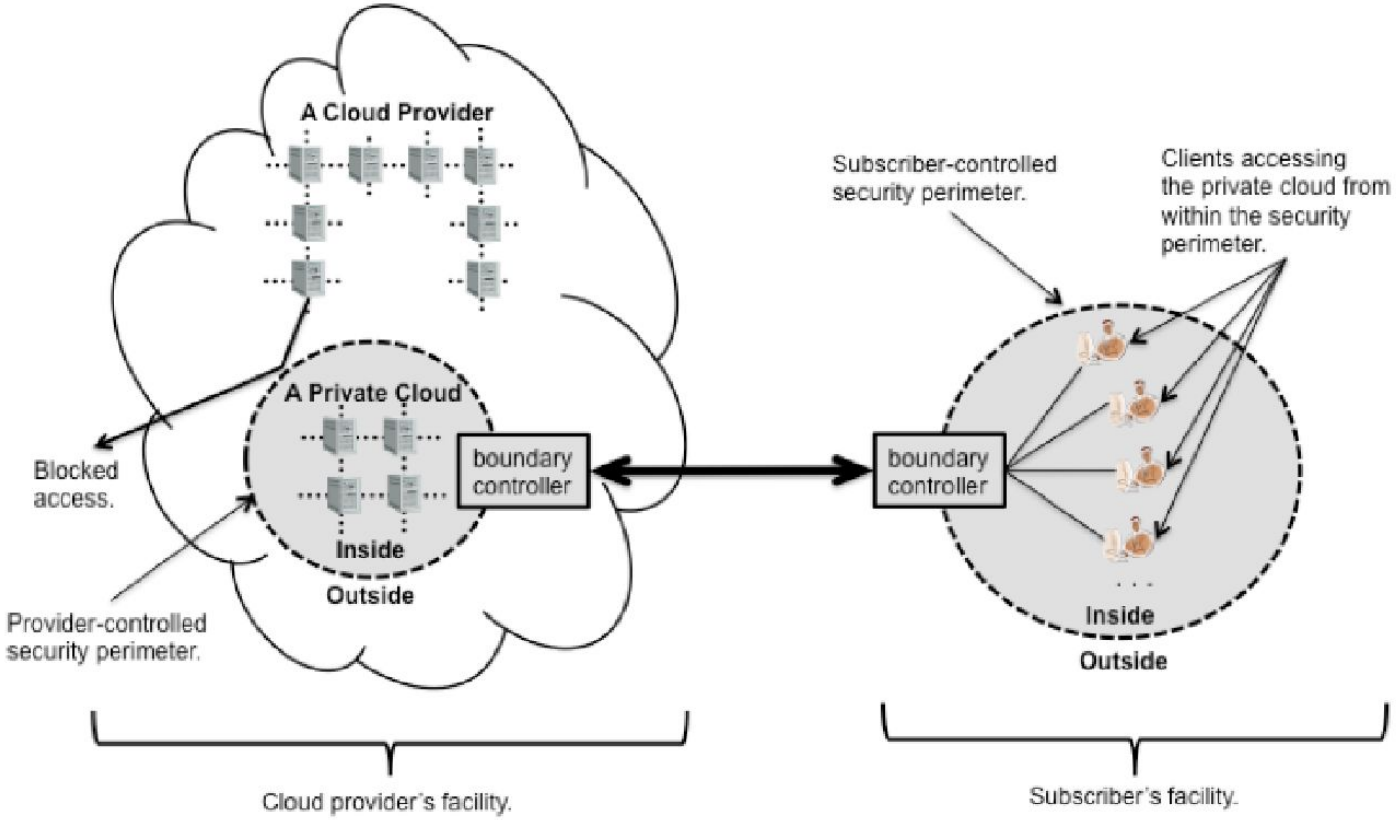
English	Перевод	Определения
Private	**Частное**	1. Реализация модели облачных вычислений на ресурсах, имеющихся в распоряжении у вашей компании, для обслуживания внутренних потребителей 2. Облачная инфраструктура функционирует целиком в целях обслуживания одной организации. Инфраструктура может управляться самой организацией или третьей стороной и может существовать как на стороне потребителя так и у внешнего провайдера.
Community	**Комму-нальное**	Облачная инфраструктура используется совместно несколькими организациями и поддерживает ограниченное сообщество, разделяющими общие принципы (например, миссию, требования к безопасности, политики, требования к соответствию <регламентам и руководящим документам>). Такая облачная инфраструктура может управляться самими организациями или третьей стороной и может существовать как на стороне потребителя так и у внешнего провайдера.
Hybrid	**Гибрид**	Облачная инфраструктура является композицией (сочетанием) двух и более облаков (частных, общих или публичных), остающихся уникальными сущностями, но объединенных вместе стандартизированными или частными (проприетарными) технологиями, обеспечивающими портируемость данных и приложений между такими облаками (например, такими технологиями, как пакетная передача данных для баланса загрузки между облаками).
Public	**Публич-ное**	Облачная инфраструктура создана в качестве общедоступной или доступной для большой группы потребителей не связанной общими интересами, но, например, принадлежащими к одной области деятельности****>. Такая инфраструктура находится во владении организации, продающей соответствующие облачные услуги/ предоставляющей облачные сервисы. ****) принадлежность к одной области деятельности/ индустрии может предполагать специфичные для этой индустрии приложения, потребность в которых испытывают организации, ведущие аналогичную деятельность или работающие на одном рынке.

Внутреннее частное облако



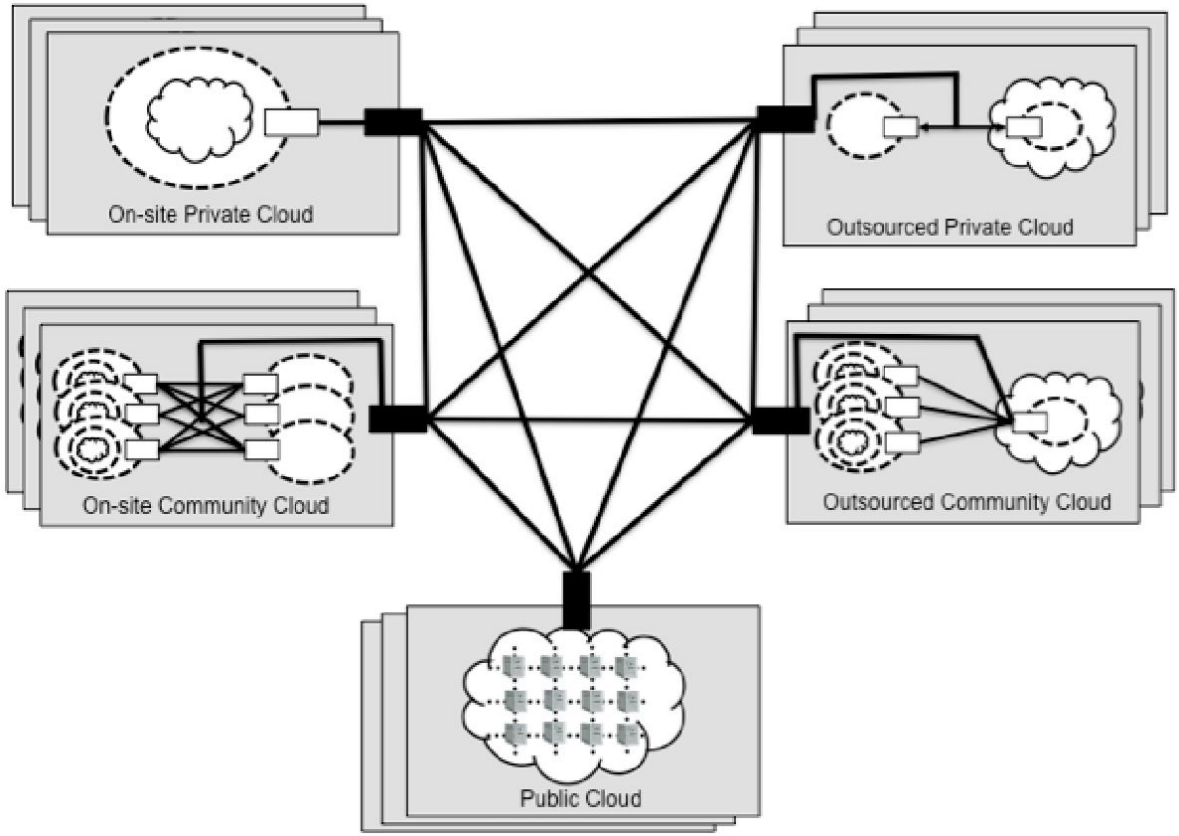
«Облачная инфраструктура функционирует целиком в целях обслуживания одной организации. Инфраструктура управляется самой организацией и существует на стороне потребителя» NIST

Внешнее частное облако

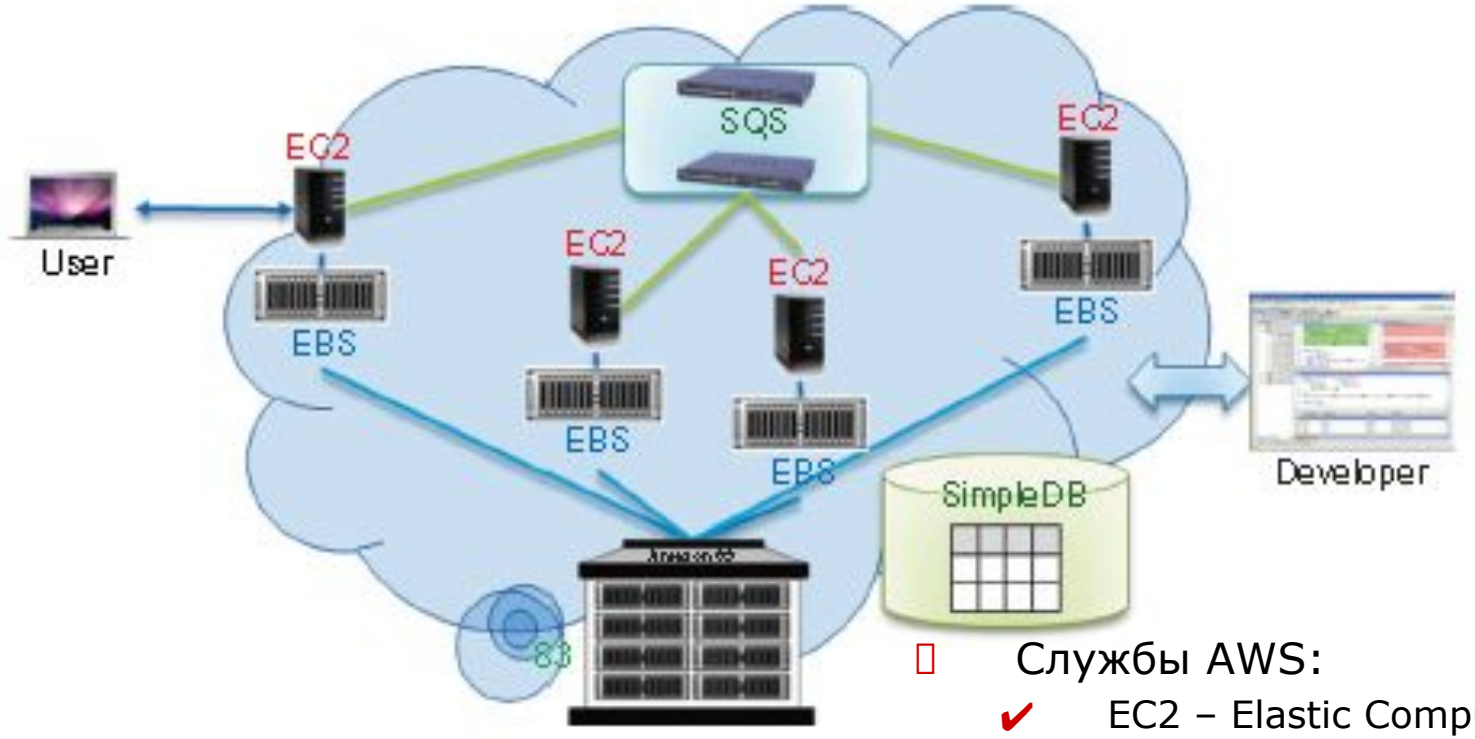


«Облачная инфраструктура функционирует целиком в целях обслуживания одной организации. Инфраструктура управляется третьей стороной и существует на стороне облачного провайдера» NIST

Гибридное облако



Архитектура AWS (Amazon Web Services)



- Службы AWS:
  - ✔ EC2 – Elastic Compute Cloud;
  - ✔ SQS – Simple QueueService;
  - ✔ EBS – Elastic block Service;
  - ✔ S3 – Simple Storage Service;
  - ✔ SNS – Simple Notification Service.

Назначение и функции сервисов AWS

Назначение и функции сервиса	Наименование сервиса AWS
Вычисления	Elastic Compute Cloud(EC2), Elastic MapReduce, Autu Scaling
Передача сообщений	Simple Queue Service(SQS), Simple Notation Service (SNS)
Хранение данных	Simple Storage Service (S3), Elastic Block Storage(EBS), AWS Import/Export
Доставка контента	Amazon CloudFront
Мониторинг	Amazon CloudWatch
Поддержка	AWS Premium Support
Базы данных	Amazon SimpleDB, Relationl Database Service (RDS)
Сетевые коммуникации	Virtual Private Cloud (VPC), Elastic Load Balancing(ELB)
Веб трафик	Alexa Web Information Service, Alexa Web Sites
E-коммерция	Fulfillment Web Service (FWS)
Учет и оплата	Flexible Payments Service (FPS), Amazon DevPay
Персонал	Amazon Mechanical Turk

Подходы к проектированию процессов выполняемых компонентами РС

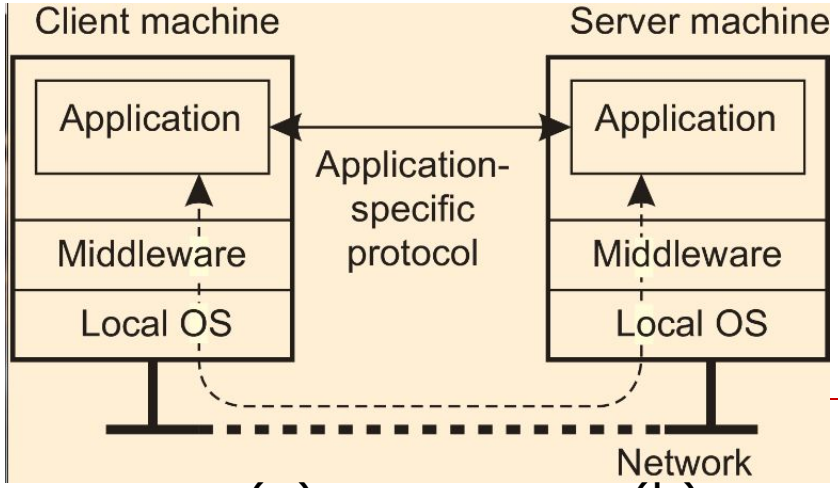
Клиенты PC

Функции клиента PC

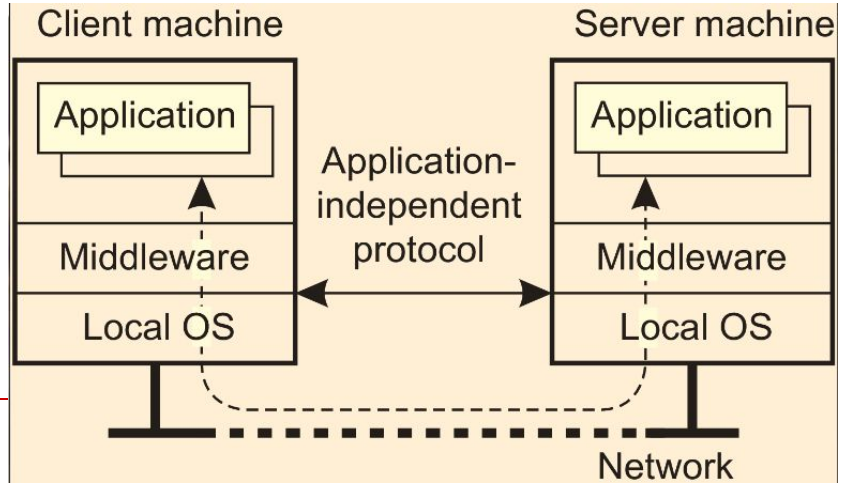
- Основное назначение:
  - Предоставление пользователю возможности взаимодействовать с сервером и получать нужный вид обслуживания.
  - В зависимости от природы (принципов, лежащих в основе) клиент-серверного взаимодействия, имеются различные способы реализации клиентского ПО.

Сетевые пользовательские интерфейсы

- При работе в сети пользователю требуется доступ к ресурсам сервера. Имеется два основных пути достижения этой цели:
  1. Обеспечить для работы с каждым видом ресурсов (видом сервиса) отдельную копию клиентского ПО. Например, файловый клиент, почтовый клиент и т.п. Вариант (a).
  2. Использовать прямой доступ к удаленному сервису (серверу), за счет обеспечения только, подходящего пользовательского интерфейса (терминальный доступ), часто средствами локальной ОС. Это подход получил название **тонкий-клиент**. Внимание к нему не ослабевает, в связи с расширением популярности Интернет, а также использованием мобильных устройств. Вариант (b).



(a)



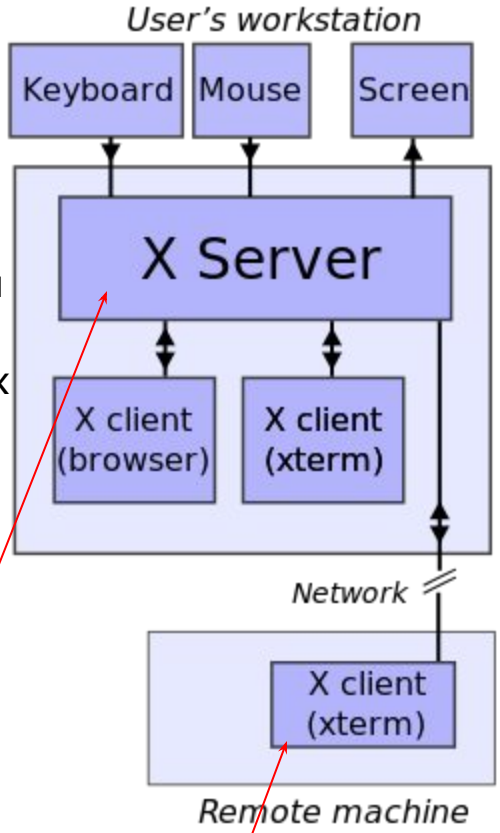
(b)

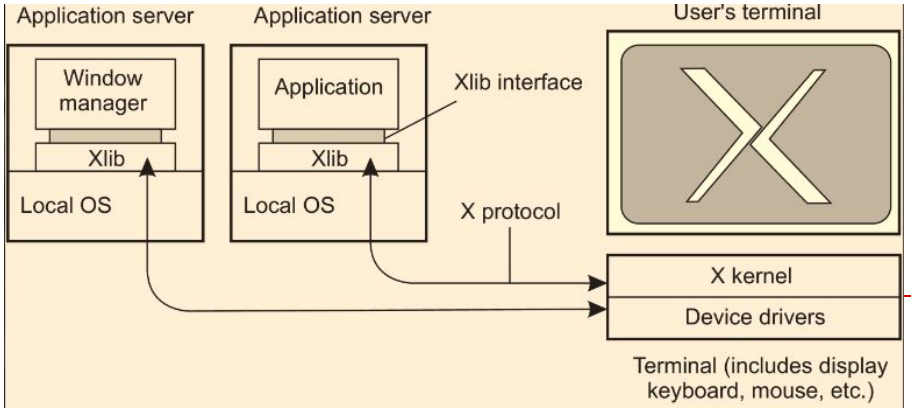
Пример: Система X-Window

- Одним из старейших, но широко распространенных протоколов удаленного обеспечения пользовательского графического интерфейса является **X Window System** или просто **X**. В мире MS Windows аналогом является Terminal Service с использованием протокола RDP (Remote Desktop Protocol). Система была создана в конце 80-х в MIT. С тех пор она развевается без изменения основных принципов, заложенных при ее создании.
- Является основой реализации графического UI в Unix и Unix-подобных ОС.

Организация X Window

- Сердцем X является **X Kernel**. Оно содержит все драйвера устройств относящихся к терминалу (KVM – Keyboard, Video, Mouse). X Kernel обеспечивает низкоуровневый интерфейс управления экраном, а также перехватывает события связанные с работой клавиатуры и мыши. Это интерфейс поддерживается приложением, исполняющемся на сервере, библиотекой **Xlib**.
- **X Kernel** и **X приложение** могут располагаться на разных машинах.
- На сервере одновременно могут исполняться несколько приложений, взаимодействующих с X Kernel клиента. Управляет отображением приложение на X клиенте специальное приложение – Window manager.





- Особенностью X является то, что **X Kernel работает как сервер**, в то время как **X приложение играет роль клиента**.

Тонкий сетевой клиент

- Недостатки X Window:
  - X приложение в процессе своей работы посылает по сети команды, касающиеся управления дисплеем на X клиенте, которые последовательно выполняются X Kernel. Это синхронное поведение компонентов X в глобальных сетях, влечет за собой снижение производительности и увеличение задержек в работе X приложений.
  - В идеале, X приложение должно разделять логику работы приложения (бизнес логику) и функции управления пользовательским интерфейсом, однако, часто это требование не выполняется, что также, влечет за собой замедление работы X клиентов и увеличение задержек в их работе.
  - Имеется несколько решений этих проблем X Window:
    - **Технология NX**: повсеместно известна как “NX”, разработана компанией NoMachine (2003). ;
    - **VNC**: Virtual Network Computing (конец 90-х). система удалённого доступа к рабочему столу компьютера, использующая протокол: Virtual Network Computing (конец 90-х). система удалённого доступа к рабочему столу компьютера, использующая протокол RFB: Virtual Network Computing (конец 90-х). система удалённого доступа к рабочему столу компьютера, использующая протокол RFB (англ. *Remote FrameBuffer*, *удалённый кадровый буфер*).;
    - **THINC**: Thin-Client Internet Computing (2007). Архитектура системы

Технология NX

- Основана на исходном протоколе X Window, со следующими улучшениями:
  - Использование сжатия данных при передаче;
  - Реализована передача только изменений, между смежными по времени образами.
  - Использование протокола SSH для обеспечения шифрования соединения.
- Начиная с 2013, с версии 4.0, NX технология стала закрытой.

VNC

- Является альтернативой X Window, в которой приложение полностью управляет отображением на удаленном дисплее, в плоть до пикселя.
- Управление осуществляется путём передачи нажатий клавиш на клавиатуре и движений мыши с одного компьютера на другой и ретрансляции содержимого экрана через компьютерную сеть.
- Система VNC платформонезависима: VNC-клиент, называемый VNC viewer, запущенный на одной операционной системе, может подключаться к VNC-серверу, работающему на любой другой ОС.
- Существуют реализации клиентской и серверной части практически для всех операционных систем, в том числе и для Java (включая мобильную платформу J2ME).
- К одному VNC-серверу одновременно могут подключаться множественные клиенты. Наиболее популярные способы использования VNC — удалённая техническая поддержка и доступ к рабочему компьютеру из дома.

THINC

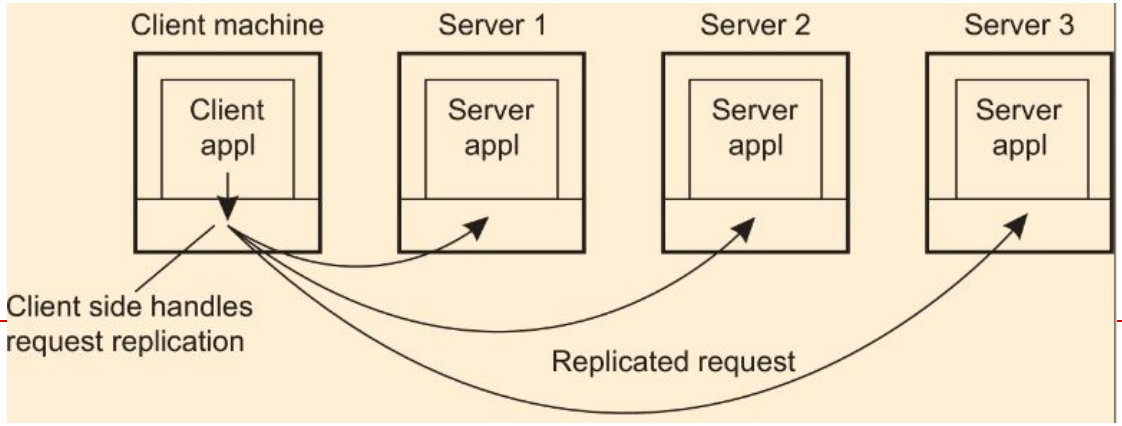
- Недостатком передачи сырых данных пикселей по сравнению с высоуровневыми протоколами такими как X Window является, то что невозможно использование семантики приложения.
- В 2005г. Баррато другой подход, который получил название THINC. Было предложено использовать ограниченное число высокоуровневых команд для управления отображением на дисплее клиента на уровне драйверов устройств.
- Это более эффективно, чем прямое управление пикселями, но мене эффективно, чем обеспечивает X Window.

Клиентское ПО прозрачного доступа

- Во многих случаях требуется обеспечить прозрачное выполнение обработки данных на клиенте и передачу результатов на сторону сервера, например передача штрих кодов, оплаченных наличными сумм и т.п. В этих случаях пользовательский интерфейс составляет небольшую часть клиентского ПО.
- Прозрачный доступ обычно реализуется с помощью генерации клиентских заглушек на основе описаний интерфейса предоставляемого сервером.
- Заглушка предоставляет такой же интерфейс, что и сервер, но она скрывает возможную разницу в архитектуре конкретных серверов, а также разницу в способах коммуникации с серверами.
- В идеале клиент не должен знать где расположен сервер и даже тот факт, что он взаимодействует с удаленным сервером.
- Имеются различные способы обеспечения прозрачности местоположения, миграции и перемещения сервера. Удобная схема именования является ключевым элементом прозрачности.

Прозрачная транзакция с помощью решений размещаемых на клиенте

- Часто прозрачность репликации реализуется с помощью ПО размещаемом на стороне клиента.
- Например, представим систему в которой обеспечивается репликация данных между серверами.
- Такая репликация легко могла бы быть выполнена путем форвардирования запросов на очередную репликацию на сервера.
- Клиент собирает все ответы серверов на запросы репликации и предает их приложению.
- Обработку всех исключительных ситуаций, касающихся возникающих в работе сервера, выполняет промежуточное ПО клиента (прозрачность к сбоям).



Сервера PC

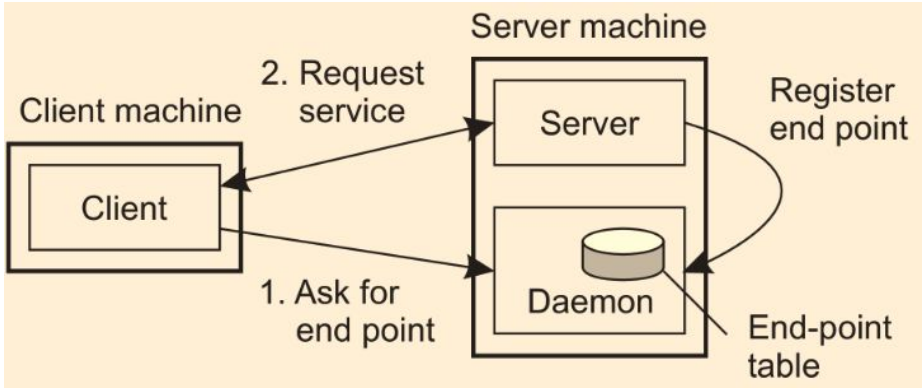
Общие проблемы создания серверов PC

- Сервер реализуется средствами исполнения процессов обеспечивающих реализацию сервиса, потребителями которого являются клиенты.
- Имеется несколько вариантов построения сервера:
  - Параллельная или интеративная обработка запросов.
  - Реализация сервера с отслеживание состояние или без.

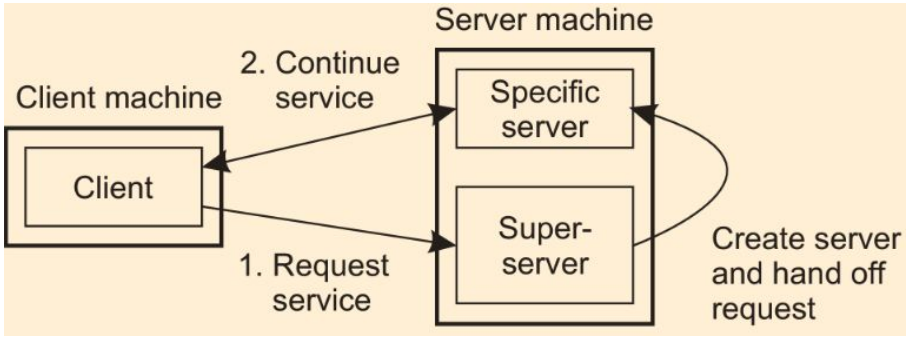
Параллельный сервер против итерационного

- В случае использования параллельной работы сервера, поступающий запрос перехватывается соответствующим процессом приема запросов.
- Для обработки запросов и формирования ответных сообщений используются отдельные потоки, запускаемые при получении очередного запроса.
- После обработки запроса и отправки ответа, поток обработчик переходит в ожидание или закрывается.
- Этот метод реализован в большинстве UNIX систем.
- При интерактивной работе сервера серверный процесс сам реализует обработку запроса, а также отправку ответа клиенту.

Конечная точка подключения к серверу



(a)



(b)

- Конечная точка подключения – это URL, при обращении к которому выполняется подключение к серверу.
- Сетевая адресация, номера портов служб, а также номера протоколов на уровне сетевого протокола назначаются IANA.
- Возможны следующие варианты реализации связи между поступившим запросом и процессом выполняющим обработку запросов.
  - В ОС Unix:
    - Модель одиночных серверов обработки – демонов.
    - Модель использования суперсервера inetd (xinetd).
  - В мире Windows:
    - Сервис зарегистрированный в системе;
    - Приложение запущенное как сервис;
    - Исполняемая программа, запущенная в системе.

Прерываемый сервер

- При создании сервера следует принимать во внимание когда и как работа сервера может быть прервана. Существует несколько способов сделать это:
  - Один из подходов, кстати единственный, надежно работающий в современном Интернете (а иногда и единственно возможный), — пользователь немедленно закрывает клиентское приложение (что автоматически вызывает разрыв соединения с сервером), тут же перезапускает его и продолжает работу. Сервер, естественно, разрывает старое соединение, полагая, что клиент прервал работу.
  - Более правильный способ - разрабатывать клиент и сервер так, чтобы они могли пересылать друг другу сигнал *конца связи {out-ofband),* который должен обрабатываться сервером раньше всех прочих передаваемых клиентом данных. Здесь также возможны разные варианты:
    - Потребовать от сервера просматривать отдельную управляющую конечную точку, на которую клиент будет отправлять сигнал конца связи и одновременно (с более низким приоритетом) — конечную точку, через которую передаются нормальные данные (Пример FTP сервис).
    - Другой вариант — пересылать сигнал конца связи через то же соединение, через которое клиент пересылал свой запрос. В TCP, например, можно посылать срочные данные. Когда срочные данные достигают сервера, он прерывает свою работу (в UNIX- системах — по сигналу), после чего может просмотреть эти данные и обработать их.

Сервер с фиксацией состояния

- Сервер с фиксацией состояния (stateful server) хранит и обрабатывает информацию о своих клиентах.
- Типичным примером такого сервера является файловый сервер, позволяющий клиенту создавать локальные копии файлов, скажем, для повышения производительности операций обновления.
- Подобный сервер поддерживает таблицу, содержащую записи пар (клиент, файл). Такая таблица позволяет серверу отслеживать, какой клиент с каким файлом работает и, таким образом, всегда определять самую «свежую» версию файла. Подобный подход повышает производительность операций чтения-записи, осуществляемых на клиенте.

Достоинства и недостатки сервера с фиксацией состояния

- Достоинства:
  - ✔ Рост производительности по сравнению с серверами без фиксации состояния, что является основной причиной разработки серверов с фиксацией состояния.
- Недостатки:
  - ✔ В случае сбоя сервера он вынужден восстанавливать свою таблицу с записями пар (клиент, файл), в противном случае не будет никакой гарантии в том, что работа происходит с последней обновленной версией файла. Как правило, серверы с фиксацией состояния нуждаются в восстановлении своего состояния в том виде, в котором оно было до сбоя.

Сервер без состояния

- Сервер без фиксации состояния (stateless sewer) не сохраняет информацию о состоянии своих клиентов и может менять свое собственное состояние, не информируя об этом своих клиентов
- Web-сервер, например, это сервер без фиксации состояния. Он просто отвечает на входящие HTTP-запросы, которые могут требовать загрузки файла как на сервер, так и (гораздо чаще) с сервера. После выполнения запроса web-сервер забывает о клиенте. Кроме того, набор файлов, которыми управляет web-сервер (возможно, в комбинации с файловым сервером), может быть изменен без уведомления клиентов об этом действии.
- Достоинства:
  - В случае архитектуры без фиксации состояния вообще нет необходимости принимать какие-то специальные меры по восстановлению серверов после сбоя. Они просто перезапускаются и работают, ожидая запросов клиента.
- Недостатки:
  - Отсутствие возможности отслеживания состояния сессии клиента.

Серверы объектов

- Сервер объектов {object sewer) — это сервер, ориентированный на поддержку распределенных объектов.
- Важная разница между стандартным сервером объектов и другими (более традиционными) серверами состоит в том, что сам по себе сервер объектов не предоставляет конкретной службы.
- Конкретные службы реализуются объектами, расположенными на сервере. Сервер предоставляет только средства обращения к локальным объектам на основе запросов от удаленных клиентов. Таким образом, можно относительно легко изменить набор служб, просто добавляя или удаляя объекты.

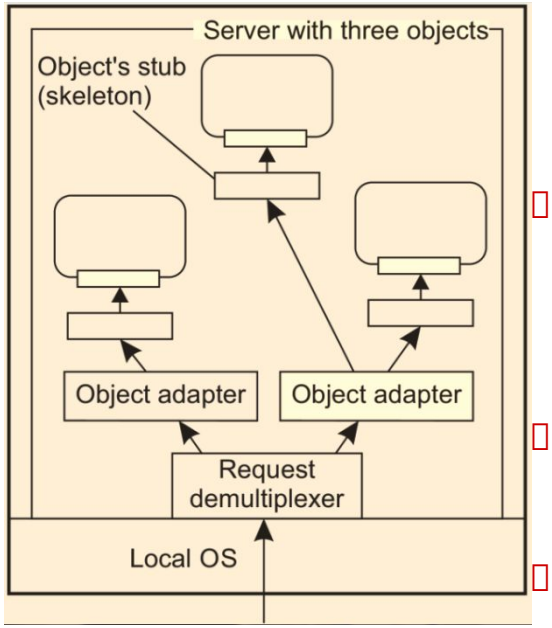
Обращение к объектам

- Объект состоит из двух частей:
  - данных, отражающих его состояние,
  - и кода, образующего реализацию его методов.
- Будут ли эти части храниться раздельно, а также смогут ли методы совместно использоваться несколькими объектами, зависит от сервера объектов.
- Кроме того, существует разница и в способе обращения сервера объектов к его объектам. Например, в многопоточном сервере объектов отдельный поток выполнения может быть назначен каждому объекту или каждому запросу на обращение к объекту.

Способы обращению к объектам. Политика активизации

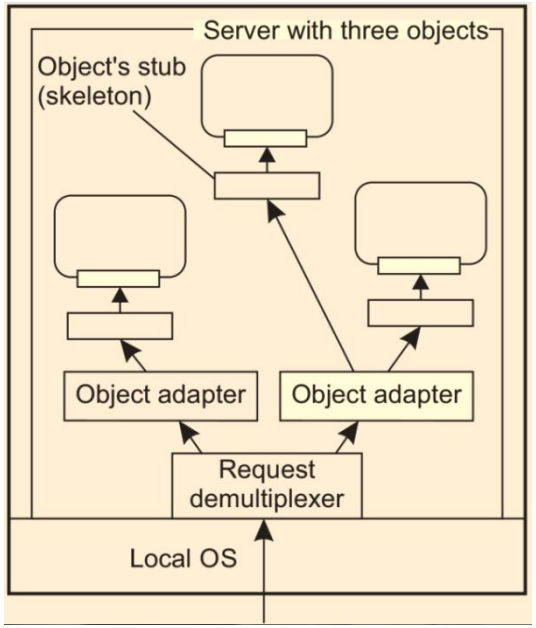
- Для любого объекта, к которому происходит обращение, сервер объектов должен знать:
  - какой код выполнять,
  - с какими данными работать,
  - запускать ли отдельный поток выполнения для поддержки обращения
  - и т. д.
- Возможны следующие подходы к реализации обращений к объектам:
  - Использовать единые правила при обращении ко всем объектам;
  - Использовать различные правила обработки объектов.
- Правила обращения к объекту обычно называют политикой активизации {activation policies), так как во многих случаях объект, перед тем как к нему можно будет обратиться, должен быть перемещен в адресное пространство сервера (то есть активизирован).

Адаптер объектов



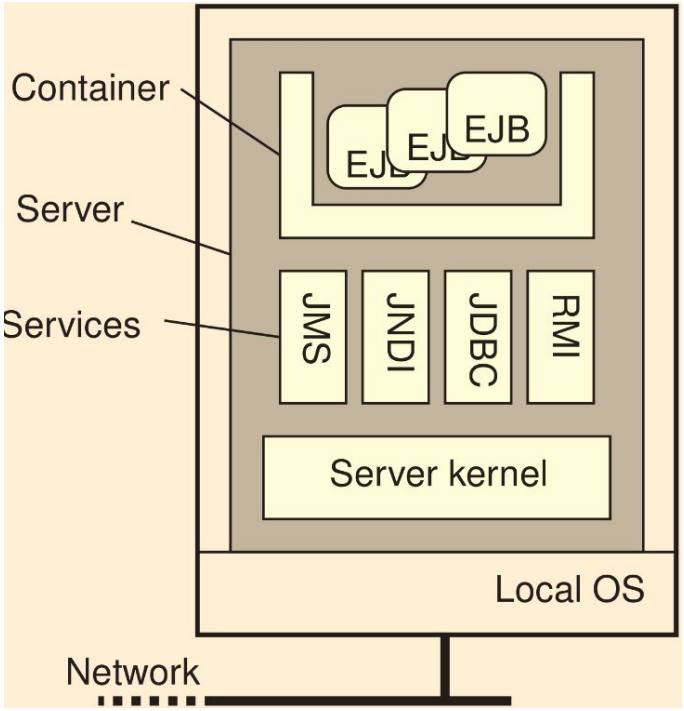
- Нужен механизм группирования объектов в соответствии с политикой активизации каждого из них. Этот механизм называют *адаптером объектов (object adapter), или упаковщиком объектов (object wrapper), но чаще всего его существование* скрыто в наборе средств построения сервера объектов.
- Адаптер объектов контролирует один или несколько объектов. Поскольку сервер должен одновременно поддерживать объекты с различной политикой активизации, на одном сервере может одновременно работать несколько адаптеров объектов.
- При получении сервером запроса с обращением к объекту этот запрос сначала передается соответствующему адаптеру объектов.
- Вместо прямой передачи запроса объекту адаптер передает запрос серверной заглушке этого объекта.
- Заглушка, называемая еще скелетоном и обычно генерируемая из определения интерфейса объекта, выполняет демаршалинг запроса (получает параметры запроса) и обращается к соответствующему методу.

Реализация адаптера



- Реализация адаптера не зависит от объектов, обращения к которым он обрабатывает. Соответственно, можно создать обобщенный адаптер и поместить его на промежуточный уровень программного обеспечения. После этого разработчикам серверов объектов можно сконцентрироваться исключительно на разработке объектов, просто указывая при необходимости, какой адаптер обращения к каким объектам должен контролировать.
- Хотя на рисунке показан специальный компонент, который занимается распределением поступивших запросов соответствующим адаптерам (демультиплексор), он не обязателен. Для этой цели мы вполне могли бы использовать адаптер объектов.

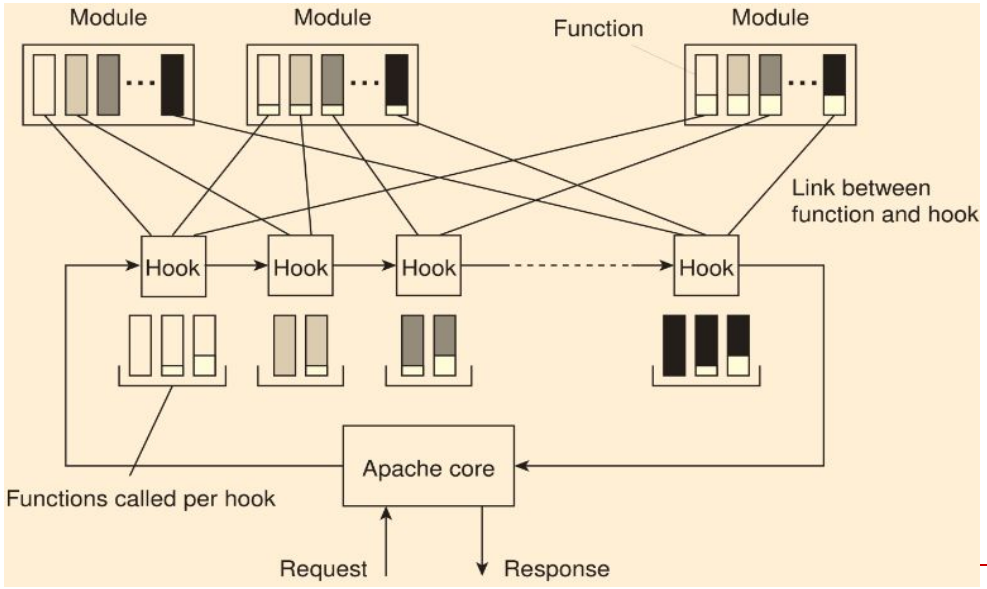
Пример: EJB



- Java Enterprise Beans по сути является объектом, который располагается на сервере и предоставляет для удаленных клиентов различные способы вызова этого объекта.
- Ключевым свойством сервера является, то что он поддерживает для ряда приложений различные виды функций.
- EJB размещается в контейнере, который предоставляет интерфейсы для нижележащих сервисов, которые реализуются сервером приложений.
- Такими сервисами являются:
  - RMI – удаленный вызов процедур;
  - JDBC – доступ к б/д;
  - JNDI – сервис именования;
  - JMS – сервис обмена сообщениями.
- Имеется возможность создания JEB 4-х видов:
  - Stateless session beans - Объект без отслеживания состояния.
  - Stateful session beans – Объект с отслеживанием состояния.
  - Entity beans – Объект сущность;
  - Message-driven beans – Объект управляемый сообщениями.

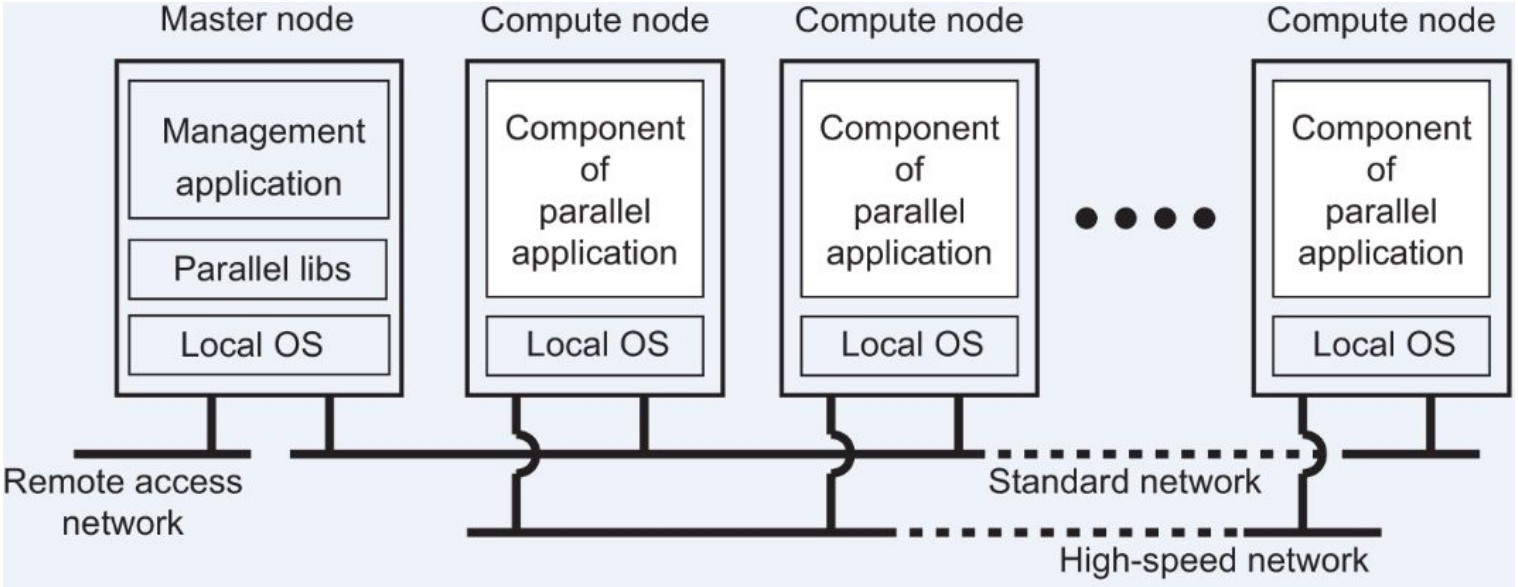
Пример: Веб-сервер Apache

- Сервер Apache – является примером того как могут быть сбалансированы между собой политики активизации и механизмы исполнения.
- Он может рассматриваться как сервер полностью приспособленный под обработку входящих запросов на предоставление Web документов.
- В основе организации работы сервера лежит понятие «ловушки» (hook).



- Сервер Apache исполняется в среде APR - Apache Portable Runtime, которая предоставляет платформо-независимый интерфейс для:
  - Управления файлами;
  - Сетевой работы;
  - Блокировки;
  - Использования потоков и т.п.
- Каждая «ловушка» представляет собой перехватчик группы однотипных действий выполняемых при обработке соотвествующих запросов.
- Функции связанные с той или иной заглушками реализуются с помощью различных модулей
- Имеется несколько десятков модулей Apache каждый из которых предназначен для выполнения определенных функций.

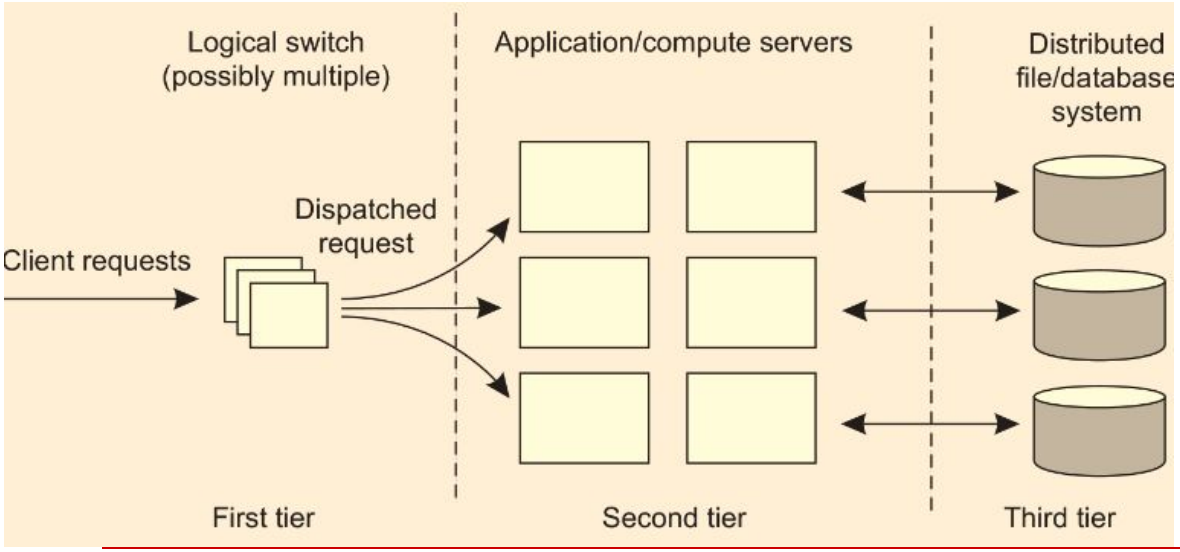
Кластера серверов



- В вычислительных кластерах один узел играет роль управляющего. Остальные узлы являются вычислительными.
- Управляющий кластер исполняет ПО промежуточного слоя, обеспечивающее управление кластером и выполнение вычислительными узлами параллельных фрагментов срвместно решаемой задачи.
- Вычислительные кластера могут быть как локальными так и глобальными.

Локальные кластера серверов. Общая организация.

- Кластер серверов представляет собой распределенную систему имеющую логическую в 3-х уровневую структуру:
  - 1-й уровень – логический коммутатор, распределяющий запросы клиентов между серверами кластера;
  - 2-й уровень – сервера приложений/вычислений, обрабатывающие запросы клиентов;
  - 3-й уровень – сервера обработки данных.



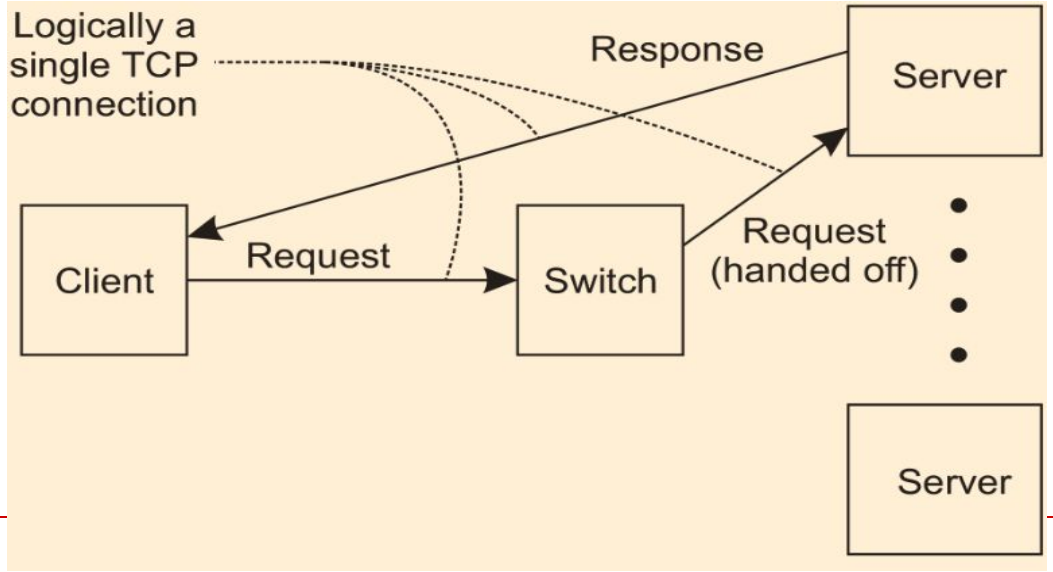
Когда кластер предлагает выполнение нескольких сервисов, они могут быть распределены на различных машинах кластера.

Как результат, некоторые машины могут находиться в режиме ожидания запросов, в то время как другие могут испытывать перегрузки.

Решением может быть – объединение машин в кластер высокой готовности и использование виртуальных машин.

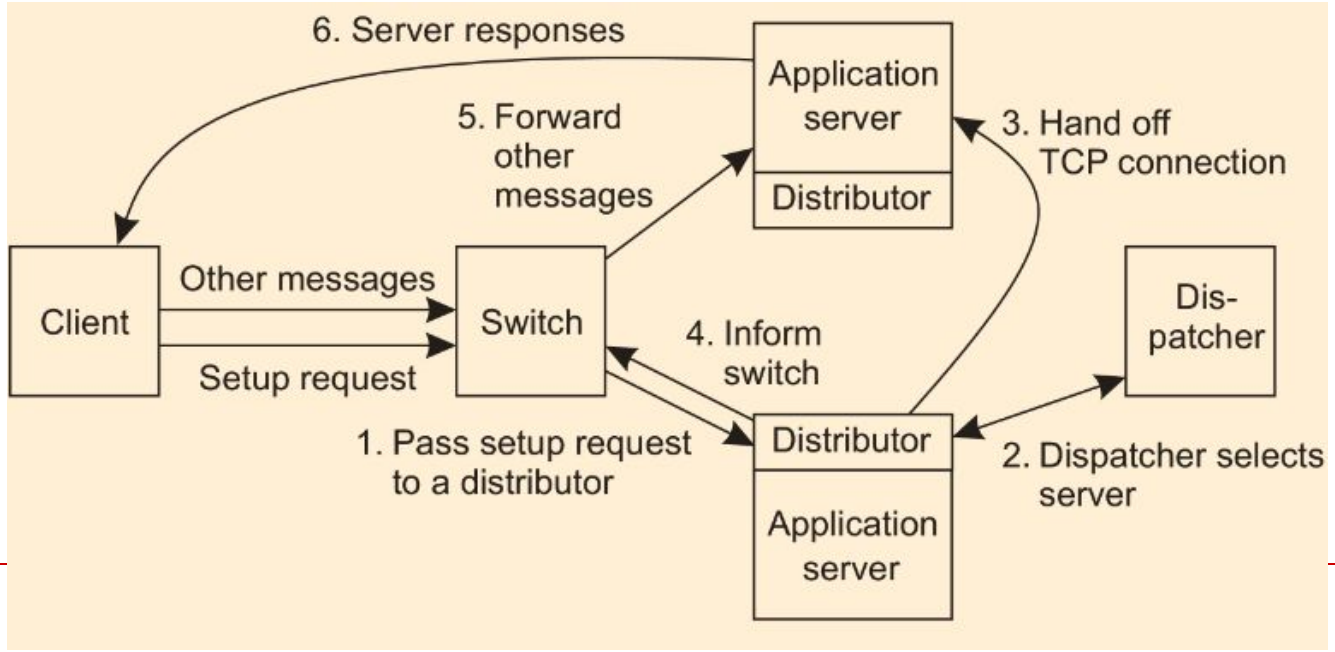
Диспетчирование запросов в локальных кластерах серверов

- Стандартным способом получения доступа к кластеру установление TCP соединения в рамках установления сессии на прикладном уровне. За установление такого соединения отвечает коммутатор входных TCP соединений между пользователями и серверами кластера (frontend).
- Имеется 2 способа организации работы коммутатора TCP соединений:
  - Трансляция сетевых адресов (NAT – Network Address Translations);
  - Переадресация вызова TCP (TCP handoff) см. рис. ниже.



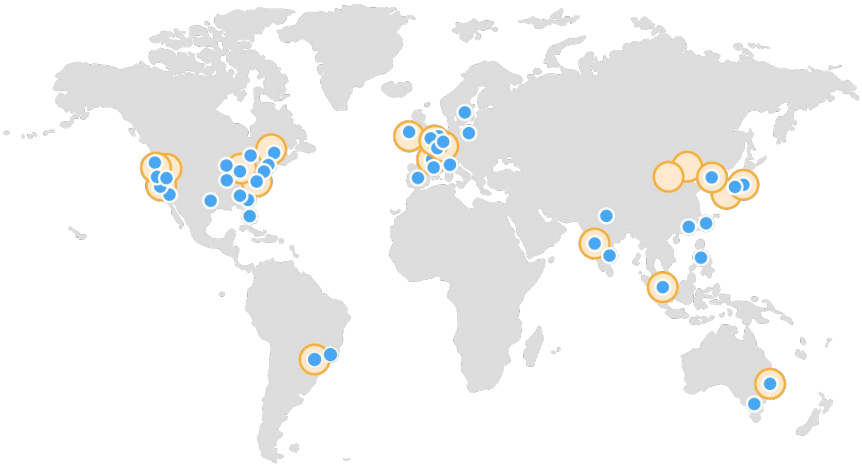
Эффективное распределение на основе контента запроса

- В случаях, когда разные сервера кластера предоставляют услуги различных служб, то возникает необходимость выбора соответствующего сервера. Здесь тоже имеется 2 пути:
  - Выбор на основе номера порта транспортного протокола.
  - На основе инспектирования содержимого пакета транспортного уровня (content-aware т.е. на основе контента).



Глобальные кластеры серверов

- Облачные провайдеры Amazon и Google имеют собственные ЦОД в разных частях мира, в которых размещаются сервера доступа к облачным сервисам.
- В этих ЦОД размещаются виртуальные машины на базе которых поставщики облачных услуг предоставляют возможность пользователям создавать свои собственные глобальные распределенные системы, содержащие большое количество виртуальных машин объеденных сетью поверх Интернет.



(а) Карта размещения ЦОД AWS

✔ Важнейшей характеристикой такой системы является место расположения серверов служб, которые должны располагаться по близости с потенциальными пользователями системы.

✔ Если близость расположения серверов к местам расположения пользователей не является важной, то можно разместить все ВМ в одном ЦОД и получить преимущество в производительности за счет скорости исполнения межпроцессных коммуникаций по локальной сети ЦОД с низкими величинами задержек.

✔ Если близость пользователей к серверам кластера важна, то важность приобретает и способ диспетчирования запросов, так как они должны направляться ближайшему серверу кластера.

Политика перенаправления

- Необходимость выбора сервера ближайшего к источнику запроса порождает проблему политики перенаправления.
- Если предположить, что по аналогии с локальными кластерами серверов диспетчирование будет осуществляться с помощью коммутатора запросов, то диспетчер должен иметь способность оценивать величину задержки передачи между клиентом и различными серверами.
- Как только сервер будет выбран, диспетчер должен уведомить об этом клиента и перенаправить его запрос серверу. Для этого могут использоваться различные механизмы:
  - Использование в качестве диспетчера DNS сервер;

    Клиент делает запрос на разрешение имени сервера и ему возвращается адрес ближайшего к нему сервера кластера (конечно диспетчер должен знать IP адрес клиента).

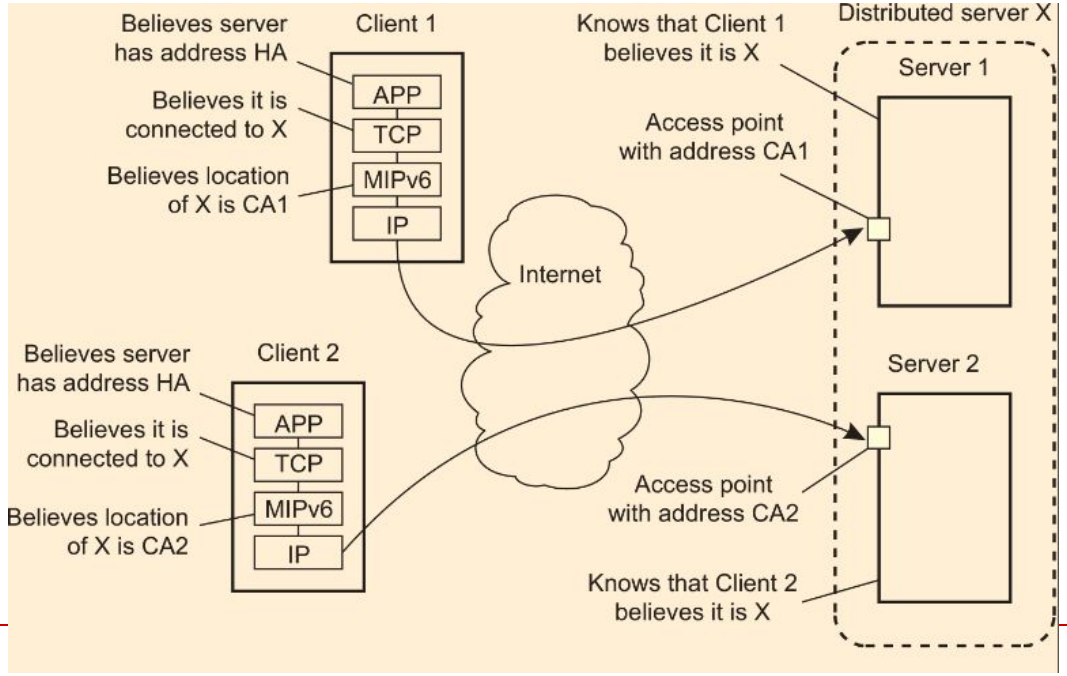
    Недостатки:

    1) клиент может работать через локальный DNS, адрес которого и будет представлен в запросе к диспетчеру на основе DNS сервера.

    2) огромные накладные затраты по времени при использовании локальных DNS, которые могут оказаться не локальными.

Диспетчирование запросов в глобальных кластерах серверов

- Надежное определение требуемого сервера может быть достигнуто с помощью механизма поддержки мобильности используемой в IPv6.
- Каждый мобильный узел сети IPv6 имеет домашний адрес (HA - home address)который определяет где действительно располагается узел. И уникальный адрес контакта (CA – contact address), который назначается ему и который может использоваться для подключения к узлу из внешней сети.



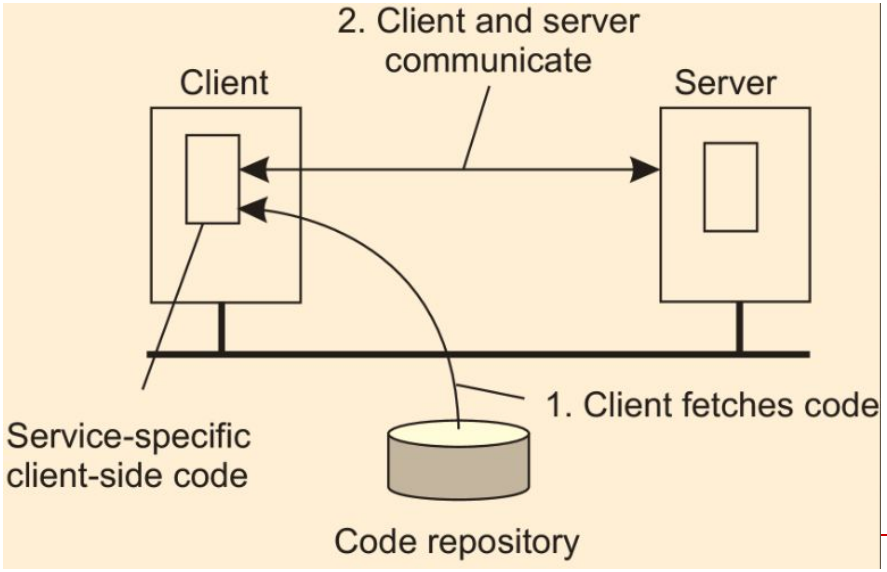
Миграция кода

Основания для переноса кода

- Традиционно перенос кода в распределенных системах происходит в форме переноса процессов (process migration)у в случае которых процесс целиком переносится с одной машины на другую.
- Поддержка переноса кода может также помочь повысить производительность на основе параллелизма, но без обычных сложностей, связанных с параллельным программированием. Типичным примером может быть поиск информации в Web.
- Относительно несложно реализовать поисковый запрос в виде небольшой мобильной программы, переносимой с сайта на сайт. Создав несколько копий этой программы и разослав их по разным сайтам, мы можем добиться линейного возрастания скорости поиска по сравнению с единственным экземпляром программы.
- Помимо повышения производительности существуют и другие причины поддержания переноса кода. Наиболее важная из них — это гибкость.

Пример: Файловый сервер

- Например, рассмотрим сервер, реализующий стандартизованный интерфейс к файловой системе.
- Чтобы предоставить удаленному клиенту доступ к файловой системе, сервер использует специальный протокол. В обычном варианте клиентская реализация интерфейса с файловой системой, основанная на этом протоколе, должна быть скомпонована с приложением клиента. Этот подход предполагает, что подобное программное обеспечение для клиента должно быть доступно уже тогда, когда создается клиентское приложение.



✔ Альтернативой является предоставление сервером клиенту реализации не ранее, чем это будет действительно необходимо, то есть в момент привязки клиента к серверу.

✔ В этот момент клиент динамически загружает реализацию, производит необходимые действия по инициализации, а затем обращается к серверу.

Модели переноса кода

- Перенос кода в широком смысле связан с переносом программ с машины на машину с целью исполнения этих программ в нужном месте.
- Для лучшего понимания различных моделей переноса кода используем шаблон, состоящий из:
  - *Сегмент кода — это часть, содержащая набор инструкций, которые выполняются* в ходе исполнения программы.
  - *Сегмент ресурсов содержит ссылки на* внешние ресурсы, необходимые процессу, такие как файлы, принтеры, устройства, другие процессы и т. п.
  - *Сегмент исполнения используется для хранения* текущего состояния процесса, включая закрытые данные, стек и счетчик программы.
- Различают 2 модели переноса кода, характеризующиеся степенью мобильности:
  - модель слабой мобильности (weak mobility). Согласно этой модели допускается перенос только сегмента кода, возможно вместе с некоторыми данными инициализации.
  - Модель сильной мобильности (strong mobility). В этом случае переносится также и сегмент исполнения.

Модель слабой мобильности

- Согласно этой модели допускается перенос только сегмента кода, возможно вместе с некоторыми данными инициализации. Характерной чертой слабой мобильности является то, что перенесенная программа всегда запускается из своего исходного состояния.
- Достоинство подобного подхода в его простоте.
- Слабая мобильность требуется только для того, чтобы машина, на которую переносится код, была в состоянии его исполнять. Этого вполне достаточно, чтобы сделать код переносимым.

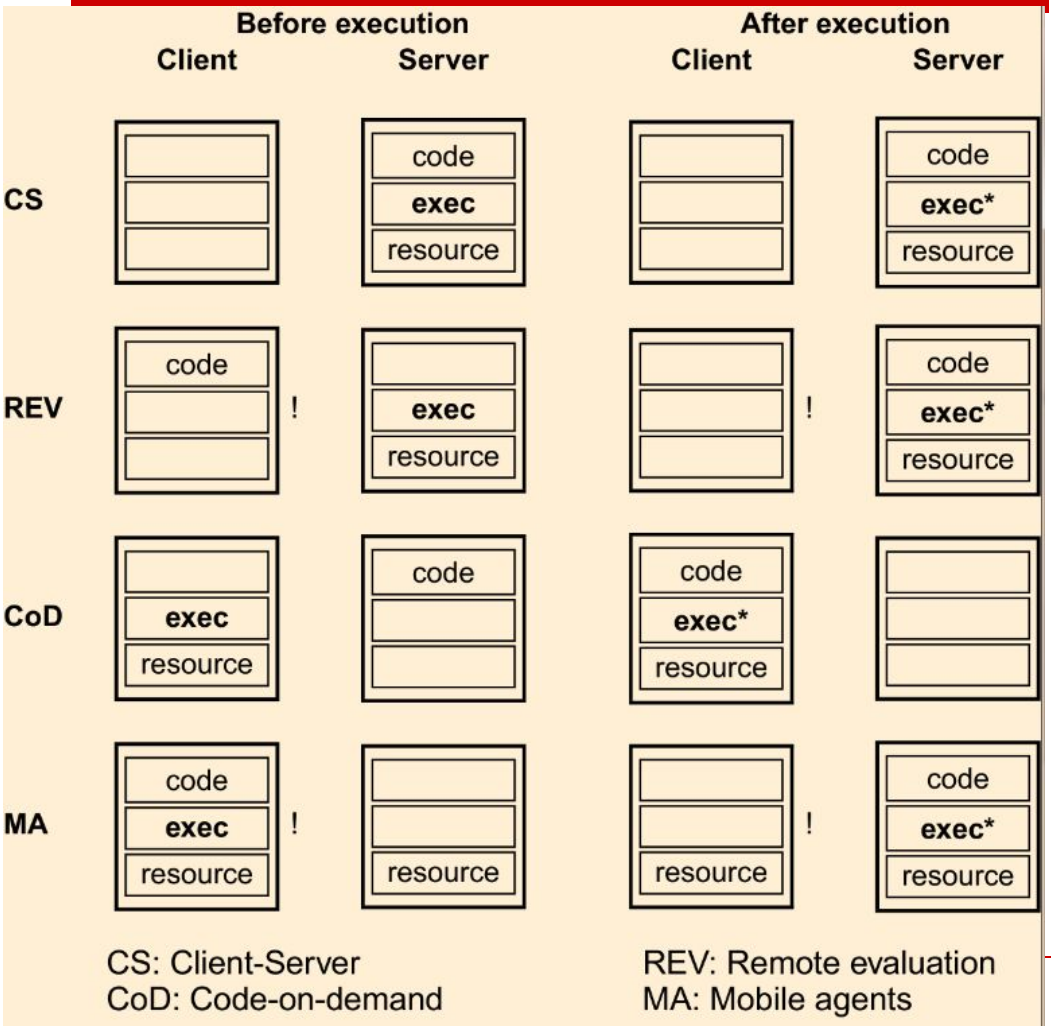
Модель сильной мобильности

- Характерная черта сильной мобильности — то, что работающий процесс может быть приостановлен, перенесен на другую машину и его выполнение продолжено с того места, на котором оно было приостановлено.
- Ясно, что сильная мобильность значительно мощнее слабой, но и значительно сложнее в реализации.

Инициатор переносимости кода

- Независимо от того, является мобильность слабой или сильной, следует провести разделение на:
  - системы с переносом, инициированным отправителем,
  - системы с переносом, инициированным получателем.
- При переносе, инициированном отправителем, перенос инициируется машиной, на которой переносимый код постоянно размещен или выполняется.
  - Обычно перенос, инициированный отправителем, происходит при загрузке программ на вычислительный сервер.
  - Другой пример — передача поисковых программ через Интернет на сервер баз данных в Web для выполнения запроса на этом сервере.
  - Безопасный перенос кода на сервер, как это происходит при переносе, инициированном отправителем, часто требует, чтобы клиент был сначала зарегистрирован и опознавался сервером.
- При переносе, инициированном получателем, инициатива в переносе кода принадлежит машине-получателю. Пример такого подхода — Java-апплеты.

Четыре варианта переноса кода



- Различные варианты переноса кода в зависимости от:
  - вида мобильности (сильная/слабая);
  - типа инициатора (отправитель/получатель);
- порождают следующие парадигмы (модели):
  - Клиент-сервер (CS) (нет мобильности кода);
  - Удаленные вычисления (REV) (слабая/отправителем);
  - Код по требованию (CoD) (слабая/получателем);
  - Мобильный агент (MA) (сильная/отправителем).

Удаленное клонирование процессов

- Помимо переноса работающего процесса, называемого также миграцией процесса, сильная мобильность может также осуществляться за счет удаленного клонирования.
- В отличие от миграции процесса клонирование создает точную копию исходного процесса, которая выполняется на удаленной машине. Клон процесса выполняется параллельно оригиналу. В UNIX-системах удаленное клонирование имеет место при ответвлениях дочернего процесса в том случае, когда этот дочерний процесс продолжает выполнение на удаленной машине.
- Преимущество клонирования — в схожести со стандартными процедурами, осуществляемыми в многочисленных приложениях.
- Единственная разница между ними состоит в том, что клонированный процесс выполняется на другой машине. С этой точки зрения миграция путем клонирования — самый простой способ повышения прозрачности распределения.

Миграция кода в гетерогенных системах

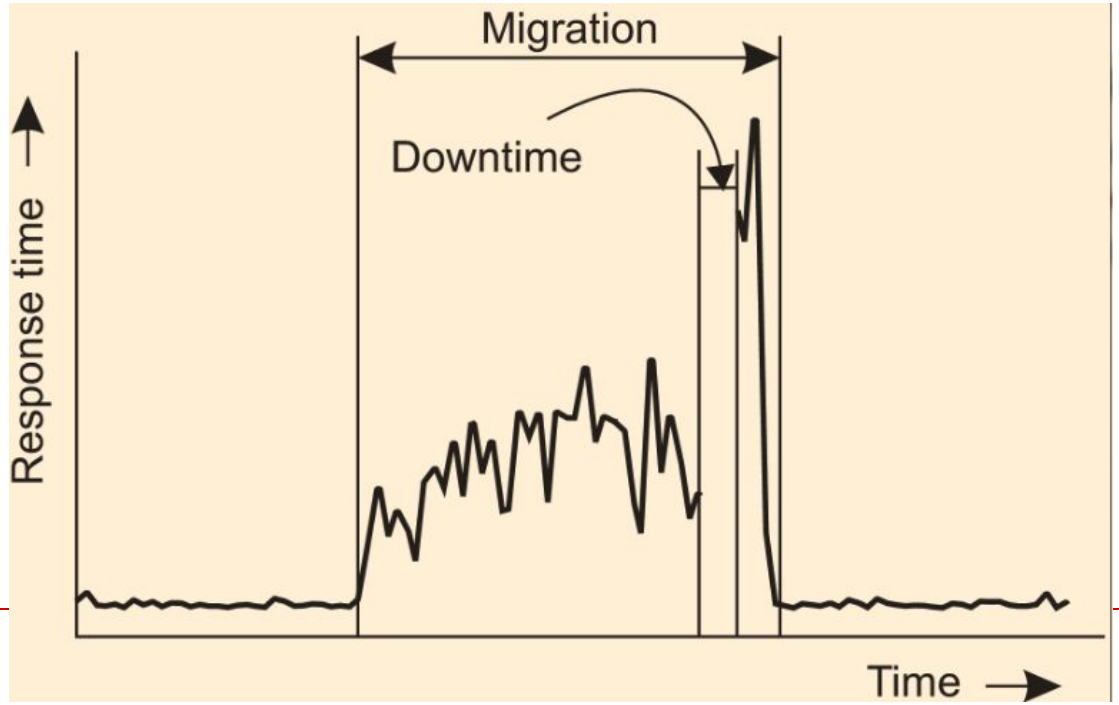
- Перенос в гетерогенных системах требует, чтобы поддерживались все эти платформы, то есть сегмент кода должен выполняться на всех этих платформах без перекомпиляции текста программы. Кроме того, мы должны быть уверены, что сегмент исполнения на каждой из этих платформ будет представлен правильно.
- Проблемы могут быть частично устранены в том случае, если мы ограничимся слабой мобильностью. В этом случае обычно не существует такой информации времени исполнения, которую надо было бы передавать от машины к машине. Это означает, что достаточно скомпилировать исходный текст программы, создав различные варианты сегмента кода — по одному на каждую потенциальную платформу. В конце 1970-х такой подход применялся для переноса Pascal программ.
- В случае сильной мобильности основной проблемой, которую надо будет решить, является перенос сегмента исполнения. Проблема заключается в том, что этот сегмент в значительной степени зависит от платформы, на которой выполняется задача.
- На самом деле перенести сегмент исполнения, не внося в него никаких изменений, можно только в том случае, если машина-приемник имеет ту же архитектуру и работает под управлением той же операционной системы.

Методы портирования процессов в гетерогенных системах.

- В разные годы были предложены следующие способы обеспечения мобильности кода в распределенных системах:
  - Генерация промежуточного кода для абстрактной виртуальной машины (1970-е для программ на Паскале);
  - Использование скриптовых языков и мобильных языков программирования, например Java;
  - Использование переноса кода вместе с окружением времени исполнения, включая операционную систему, с помощью технологии виртуализации (виртуальных машин).

Влияние миграции кода на время ответа ВМ

- При миграции виртуальной машины время неактивности этой ВМ составляет всего несколько минут.



Выводы