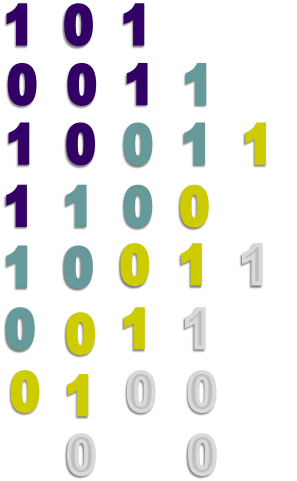
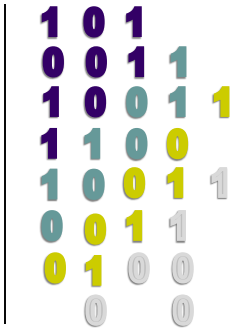
Операционные системы

Архитектура операционных систем

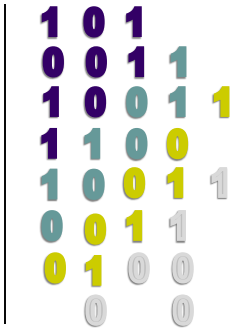


API



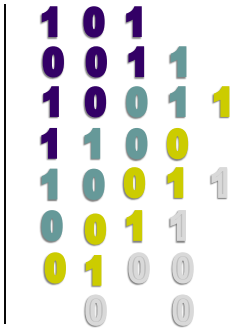
- Application programming interface
- Интерфейс прикладного программирования
- Набор готовых **классов, процедур, функций, структур и констант**, предоставляемых для использования во внешних программных продуктах
- Может быть у любого программного обеспечения

API



- ОС, библиотеки, приложения, веб-приложения
  - DirectX, OpenGL
- Многообразие API
  - У каждой ОС свой API
  - Прослойки — промежуточные API
    - Gtk, Qt, Wine, Cygwin, стандарт C, скриптовые языки
- Строгое описание сигнатур функций
  - Аргументы, их порядок, возвращаемое значение

API



- BOOL GetVersionEx(LPOSVERSIONINFO lpVersionInformation);

- typedef struct _OSVERSIONINFO {
  DWORD dwOSVersionInfoSise;
  DWORD dwMajorVersion;
  DWORD dwMinorVersion;
  DWORD dwBuildNumber;
  DWORD dwPlatformId;
  TCHAR szCSDVersion[128];
  } OSVERSIONINFO;

Windows API

Microsoft Developer Network | Technologies | Downloads | Programs | Community | Documentation | Samples | Sign in | MSDN subscriptions | Get tools

- MSDN Library
- Windows Desktop App Development
- Develop
- Desktop App Technologies
- Desktop App UI
- Windows and Messages
- Window Classes
- Window Class Reference
  - Window Class Functions
    - GetClassInfo
    - GetClassInfoEx
    - GetClassLong
    - GetClassLongPtr
    - **GetClassName**
    - GetClassWord
    - GetWindowLong
    - GetWindowLongPtr
    - RegisterClass
    - RegisterClassEx
    - SetClassLong
    - SetClassLongPtr
    - SetClassWord
    - SetWindowLong
    - SetWindowLongPtr
    - UnregisterClass

GetClassName function

Retrieves the name of the class to which the specified window belongs.

Syntax

C++

```
int WINAPI GetClassName(
  _In_   HWND hWnd,
  _Out_  LPTSTR lpClassName,
  _In_   int nMaxCount
);
```

Parameters

hWnd [in]
Type: **HWND**

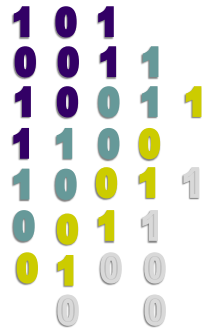
A handle to the window and, indirectly, the class to which the window belongs.

lpClassName [out]
Type: **LPTSTR**

The class name string.

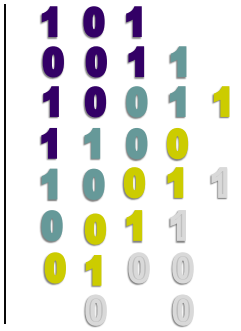
nMaxCount [in]
Type: **int**

Ядро (kernel)



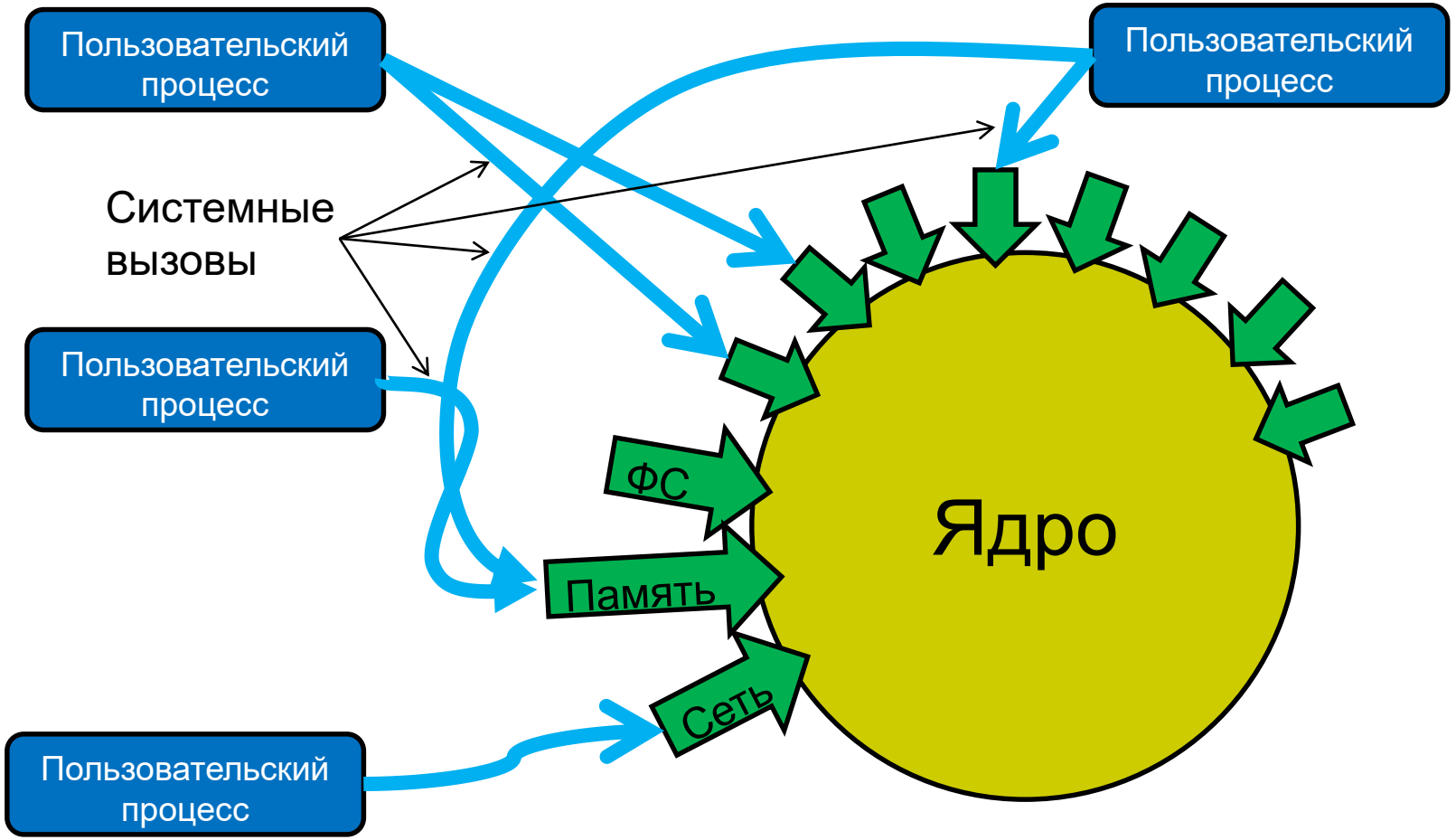
- Центральная часть операционной системы, обеспечивающая приложениям координированный и контролируемый доступ к ресурсам компьютера
  - Процессорное время
  - Оперативная память
  - Внешнее оборудование

Архитектуры ядра



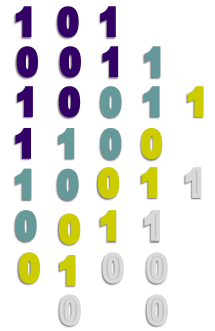
- Монолитное ядро
- Модульное ядро
- Микроядро
- Гибридное ядро

Монолитное ядро



Точки входа системных вызовов API

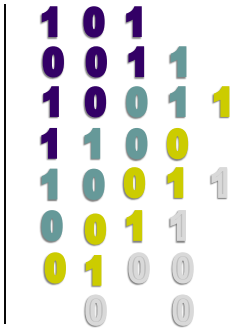
Монолитное ядро



- Все части работают в одном адресном пространстве как одна программа
- Каждая процедура может вызывать каждую
- Все процедуры ядра работают в привилегированном режиме
- Пользовательские программы взаимодействуют с ядром через системные вызовы и переключение уровня привилегий
  - Сервисная программа

Монолитное ядро. Внутреннее строение

Модульное ядро

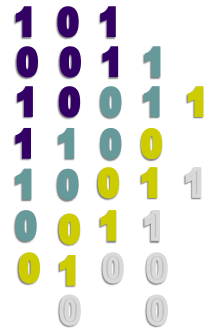


- Модули
- Большинство современных монолитных ядер позволяют во время работы загружать *модули*

- Модули не самостоятельны

- Статически связаны с ядром

Свойства монолитного ядра

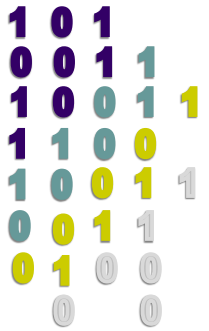


+ Скорость работы
+ Упрощённая разработка модулей

- Сбой в одном из компонентов может нарушить работоспособность всей системы
- Сложно поддерживать и развивать из-за сложной внутренней структуры

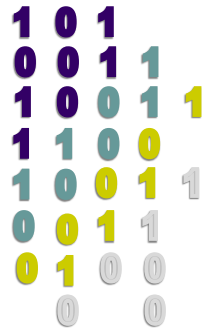
 * Linux, BSD, DOS, …

Многоуровневые системы (Layered systems)



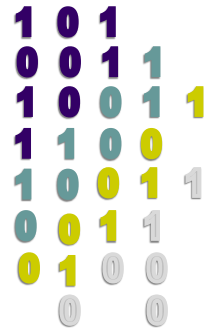
- Реализуется в системах с несколькими уровнями привилегий
- Процедура уровня N может вызывать только процедуры уровня N-1
- Все или почти все уровни работают в привилегированном режиме
- MULTICS, THE

Многоуровневые системы MULTICS



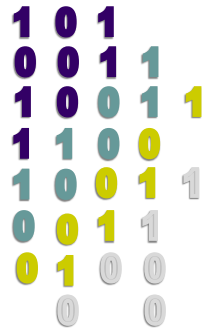
5	Интерфейс пользователя
4	Управление вводом-выводом
3	Драйвер связи с консолью
2	Управление памятью
1	Планирование задач и процессов
0	Hardware

Микроядерная архитектура



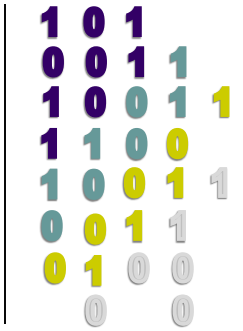
- Минимизация ядра
- Перенос части кода системы в пользовательское пространство
  - Непривилегированный режим
- **Микроядро** предоставляет только элементарные функции управления процессами и минимальный набор абстракций для работы с оборудованием

Микроядерная архитектура Сервисы



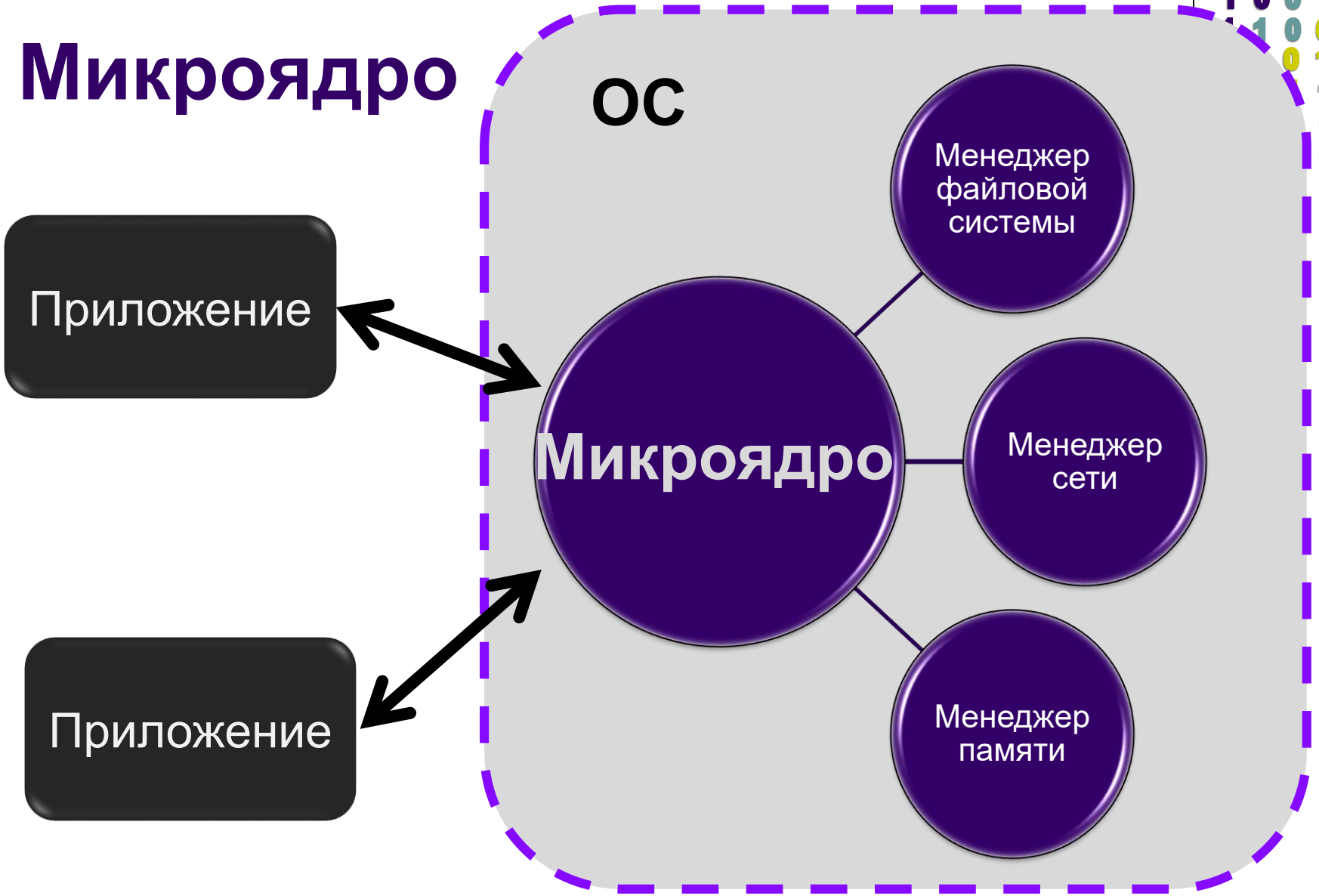
- Процессы в пространстве пользователя выполняющие те или иные системные функции
- Драйвера и модули размещаются не в микроядре, а в сервисах
- Сервисы используют API микроядра
- Сервисные процессы/демоны в монолитных ОС

Микроядро

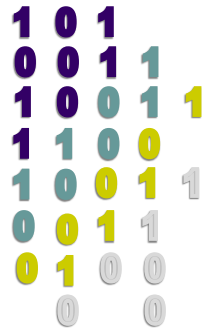


- Выполняет только строгий минимум функций в привилегированном режиме
  - взаимодействие между программами
  - планирование использования процессора
  - первичную обработку прерываний
  - операции ввода-вывода
  - базовое управление памятью
- Компоненты системы взаимодействуют друг с другом путём передачи сообщений через микроядро

Микроядро


ОС
Менеджер файловой системы
Приложение
Микроядро
Менеджер сети
Приложение
Менеджер памяти

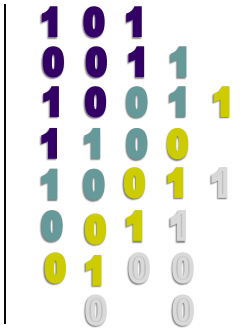
Достоинства и недостатки



- ☺ Высокая степень модульности ядра
  - ☺ Загрузка/выгрузка компонент
  - ☺ Отладка
- ☺ **Существенно** повышает надёжность системы (аппаратные и программные ошибки)

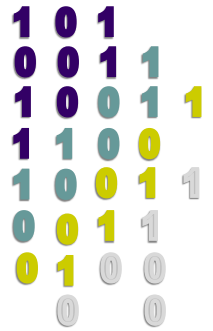
- ☹ Вносит дополнительные накладные расходы (при передаче сообщений)
- ☹ Сложное проектирование для минимизации потерь

Микроядерная архитектура Примеры



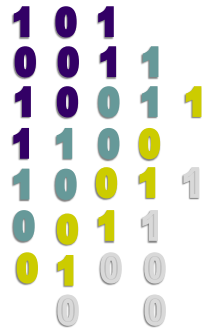
- QNX
- Mach
  - GNU/Hurd и MacOS X
- AIX
- Minix
- ChorusOS
- AmigaOS
- MorphOS

Гибридная архитектура



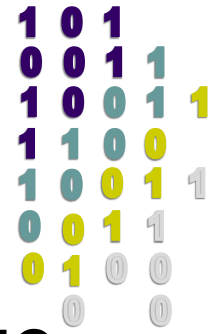
- Все подходы обладают недостатками и достоинствами
- Современные ОС общего назначения объединяют основные подходы для уменьшения недостатков
- Например, многоуровневый подход применяется практически везде

Гибридная архитектура. Примеры



- Linux = монолитное ядро + модульная архитектура
- Системы с монолитным ядром под управлением микроядра
  - 4.4BSD и MkLinux, на микроядре *Mach*
- Windows
  - Микроядро + элементы монолитной архитектуры
  - Части системы взаимодействуют сообщениями
    - Но используют общие структуры данных

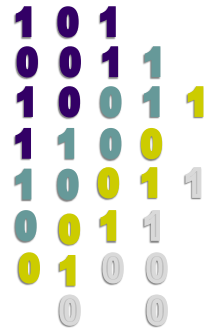
Виртуальные машины



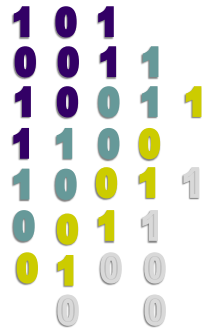
- Виртуальные машины в широком смысле слова
  - Виртуальная исполнительная среда с определённым набором команд
  - Скриптовые языки
  - Языки компилируемые в промежуточный код
    - Java, C#, …
  - ОС предоставляющие виртуальное оборудование
  - Гипервизоры

JVM

```
 iget-object v0, p0, Lcom/a/a/e;->e:Ljava/lang/String;
    return-object v0
.end method


# virtual methods
.method public final run()V
    .locals 10

    .prologue
    const/4 v9, 0x0
    const/4 v8, 0x0
    const/16 v7, 0x400
    const-string v0, "Localytics_uploader"
    const-string v0, "/"

    .line 90
    .line 94
    :try_start_0
    iget-object v0, p0, Lcom/a/a/e;->b:Ljava/io/File;

    if-eqz v0, :cond_2

    iget-object v0, p0, Lcom/a/a/e;->b:Ljava/io/File;

    invoke-virtual {v0}, Ljava/io/File;->exists()Z

    move-result v0

    if-eqz v0, :cond_2

    .line 96
    iget-object v0, p0, Lcom/a/a/e;->b:Ljava/io/File;

    invoke-virtual {v0}, Ljava/io/File;-
>getAbsolutePath()Ljava/lang/String;

    move-result-object v0

    .line 100
    new-instance v1, Lcom/a/a/g;
    invoke-direct {v1, p0}, Lcom/a/a/g;-><init>(Lcom/a/a/e;)V
    iget-object v2, p0, Lcom/a/a/e;->b:Ljava/io/File;

    invoke-virtual {v2, v1}, Ljava/io/File;-
>list(Ljava/io/FilenameFilter;)[Ljava/lang/String;

    move-result-object v1
    move v2, v9
    :goto_0
    array-length v3, v1
```

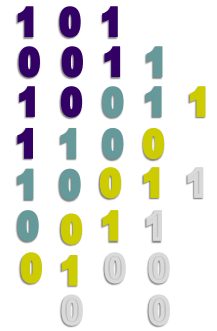


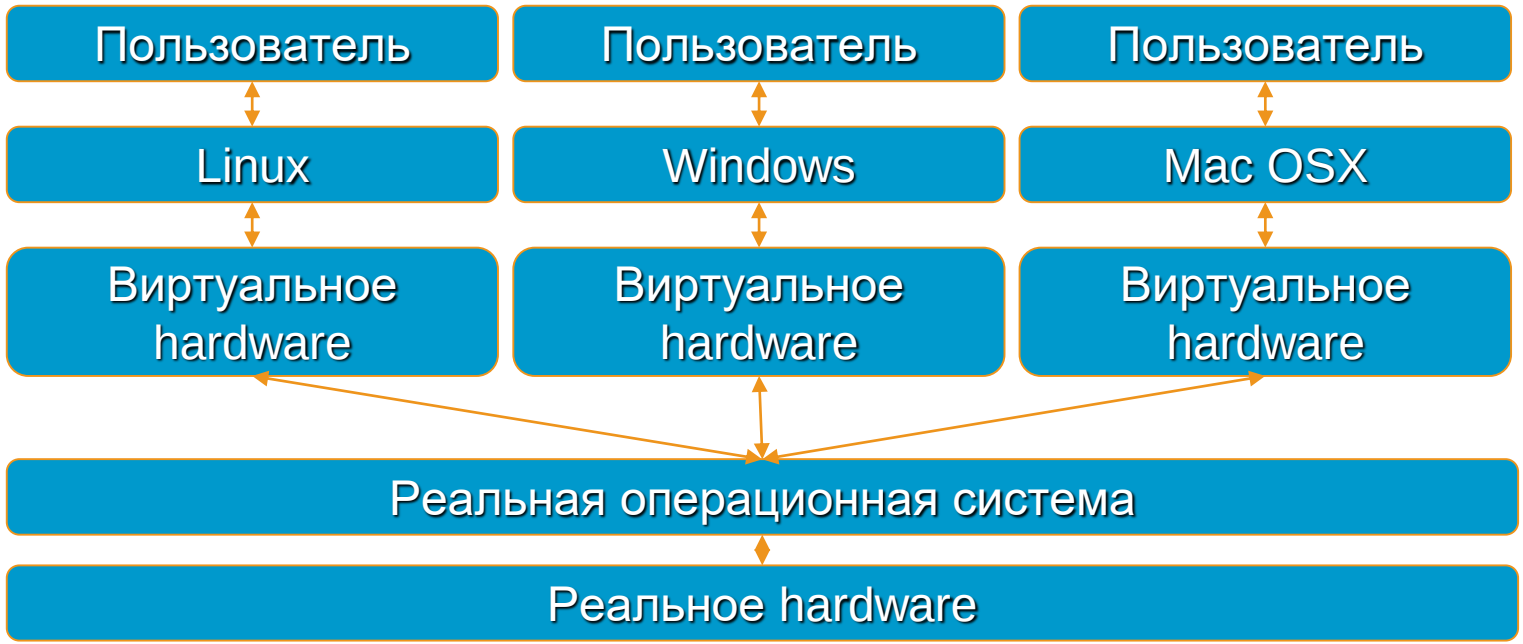
ОС – виртуальные машины



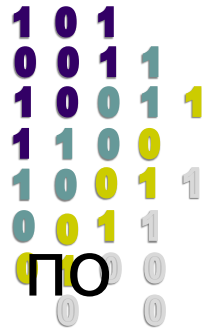
- VM/370 (IBM/370)
- Реализует виртуальную машину для каждого пользователя
  - Пустое оборудование, с полным набором команд
  - Задача пользователя запустить ПО
  - Привилегированные команды перехватываются системой и исполняются

Виртуальные машины





Гипервизоры



- Используют возможности оборудования по виртуализации аппаратного обеспечения
  - Добавляется «-1» кольцо безопасности
  - Процессор позволяет перехватывать все привилегированные команды гипервизором и обрабатывать их
  - Гипервизор запускается из ОС/перед запуском основной ОС
  - Для гипервизора все ОС равноценны
  - Могут работать практически без потерь производительности

Вопросы?