# Архитектура параллельных вычислительных систем

к.т.н., доцент
**Костичев Сергей Валентинович**

Введение.

**snenv@mail.ru**

# Учебные вопросы:

1. Цели и задачи дисциплины, методика текущего контроля. Литература.
2. «Задачи большого вызова»
3. Модель и принципы фон Неймана
4. Машина Тьюринга
5. Архитектура и организация ЭВМ
6. Эволюция микропроцессорных архитектур
7. Уровни параллелизма
8. Классификация архитектур по параллельной обработке данных по Флинну
9. MIMD системы (Классификация Хокни)



# 1. Цели и задачи дисциплины. Методика текущего контроля. Литература.

## ЦЕЛИ И ЗАДАЧИ ДИСЦИПЛИНЫ

**Изучение** принципов действия скалярных, потоковых, параллельных и векторных вычислительных устройств.

**Формирование** навыков работы в параллельной вычислительной среде.

**Получение** знаний для последующего изучения методов и средств параллельных и распределенных научных вычислений



## ТЕКУЩИЙ КОНТРОЛЬ

1.Осуществляется в форме оценивания **знаний** и **организованности** работы студентов.

2. Оценка **знаний** осуществляется по результатам:

- контрольных работ в первой и второй половинах семестра,
- выполнения и защиты лабораторных работ.

3. Оценки **организованности** работы студентов выставляется с учетом соблюдения студентами планового графика выполнения лабораторных работ и их защиты

4. Максимальные оценки в баллах за семестр по всем составляющим:

- контрольные работы – 2*10=20 баллов;
- лабораторные работы – 5*10 =50 баллов,
- курсовая работа - 30 баллов

## **ТЕКУЩИЙ КОНТРОЛЬ** (продолжение)

5. Итоговая оценка в баллах формируется суммированием оценок, после чего она переводится в обычную 4-балльную шкалу и проставляется в ведомость и в зачетную книжку.

Шкала перевода:

| **Итоговая сумма в баллах** | **Итоговая оценка по дисциплине** |
|---|---|
| >=85 | отлично |
| 65-84 | хорошо |
| 45-64 | удовлетворительно |
| <45 | неуд |

# Литература


1. В.П.Гергель Теория и практика параллельных вычислений. Учебное пособие – М.: ИНТУИТ.РУ «Интернет-Университет Информационных Технологий», 2007 /
2. К.Ю.Богачев Основы параллельного программирования. – М.: БИНОМ. Лаборатория знаний, 2003.
3. А.С.Антонов Параллельное программирование с использованием технологии MPI. – М.: Изд.МГУ, 2004 /
4. Т.Чан Системное программирование на C++ для UNIX – Издательская группа BHV, 1997
5. И.В.Шошмина, А.К.Зароченцев, А.С.Иванов, Г.А.Феофилов Использование Grid-технологий для крупномасштабных научных экспериментов. Часть 1. Введение в Grid-технологии с примерами практических занятий на базе ARC NorduGrid. – СПб.: Изд.СПбГУ, 2006.
6. Воеводин В.В., Воеводин Вл.В.- Параллельные вычисления. – М.: "БХВ",2002
7. Корнеев В.В. Параллельные вычислительные системы. – М.: “Нолидж”, 1999
8. С.Немнюгин, О.Стесик, Параллельное программирование для многопроцессорных вычислительных систем. СПб:, "БХВ-Петербург", 2002




# 2. «Задачи большого вызова»



Сфера применения многопроцессорных вычислительных систем расширяется:

- обработка транзакций в режиме реального времени (on-line transaction processing)
- создание хранилищ данных для организации систем поддержки принятия решений (Data Mining)

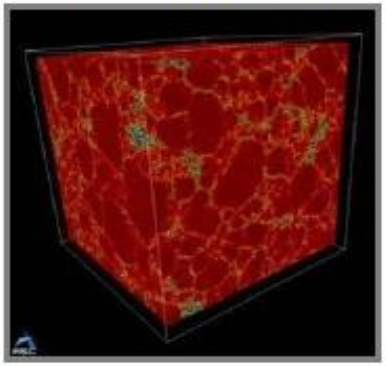
Расширение области применения в традиционных сферах.

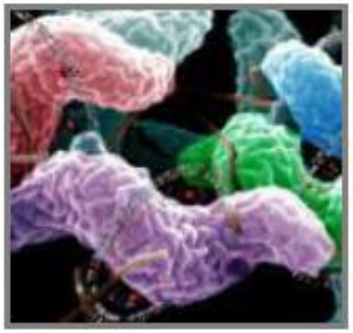
«**Задачи большого вызова**» — «**Grand challenges**» — термин, использованный в конце 1980-х годов в целях обозначения необходимости финансирования исследований в области высокопроизводительных вычислений (**HPC** =High-Performance Computing) и коммуникаций. Введен нобелевским лауреатом Kenneth G. Wilson.

Включал круг проблем, жизненно важных для развития человечества, эффективное решение которых возможно только с использованием сверхмощных вычислительных ресурсов.

# Где без HPC невозможно обойтись ?

Науки о материалах и нанотехнологии



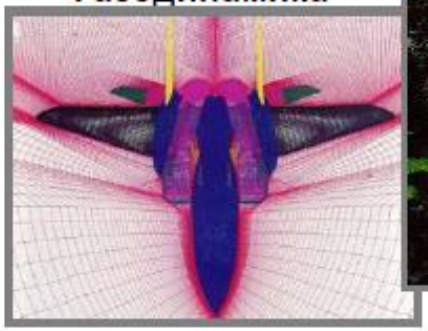


Геномика

Исследование пандемий

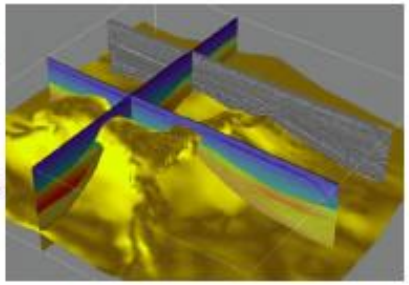

Системная биология

Газодинамика



Моделирование климата и землетрясений

Финансы и оценка рисков

Обработка геофизических данных

Список включает комплексные задачи физики, нанотехнологий, аэронавтики, биологии, национальной безопасности, науки о Земле, энергетики, окружающей среды и т.д.

«Задачи большого вызова» можно разделить на 3 группы

- задачи обработки больших и сверхбольших объемов данных;
- большое количество слабосвязных задач;
- одна сильносвязная задача, требующая большого объема памяти и производительности вычислительных мощностей.

Понимание того, что **одним локальным аппаратным решением невозможно обеспечить решение большинства задач «Grand challenges»,** привело к разработке идей распределенных вычислений, появлению кластерных технологий и GRID – технологии.



# 3. Модель и принципы фон Неймана



В 1946 году учёные Артур Бёркс, Герман Голдстайн и Джон фон Нейман опубликовали статью «Предварительное рассмотрение логического конструирования электронного вычислительного устройства», в которой изложили новые принципы построения и функционирования ЭВМ.

Имя фон Неймана в отличии от соавторов было широко известно в науке ⇒ данные идеи получили название архитектура (модель) и принципы фон Неймана

# Модель фон Неймана

**Система памяти**
хранит как команды, так и данные
Доступ с помощью
***регистра адреса***
(memory address register, MAR)
и ***регистра данных***
(memory data register, MDR)

**Блок обработки данных**
***арифметико-логическое устройство*** (ALU)
или
***центральный процессор*** (CPU)
имеет небольшой объем памяти –
*набор регистров*

**Блок управления**
отвечает за операции между компонентами модели
***счетчик команд***
**+**
***регистр команд***

**Подсистема ввода-вывода**
***(I/O)***
энергонезависимый способ хранения
а также
способ выдачи данных пользователю
и принятие входных данных

**Является абстрактной моделью ЭВМ**



# Принципы фон Неймана

## Основные

Принцип программного управления.
Принцип хранимой программы.
Принцип однородности памяти.
Принцип отсутствия различий в семантике данных.
Принцип линейности и адресуемости памяти.
Принцип последовательного выполнения программы

Джон фон Нейманн (1903-1957)

## Дополнительные

Принцип двоичного кодирования.
Принцип иерархической памяти.
Принцип низкоуровневости машинного языка.
Принцип микропрограммирования.

# Принцип программного управления

Алгоритм, реализуемый вычислительной машиной, должен быть представлен в виде **программы**, представляющей собой последовательность управляющих слов – команд.

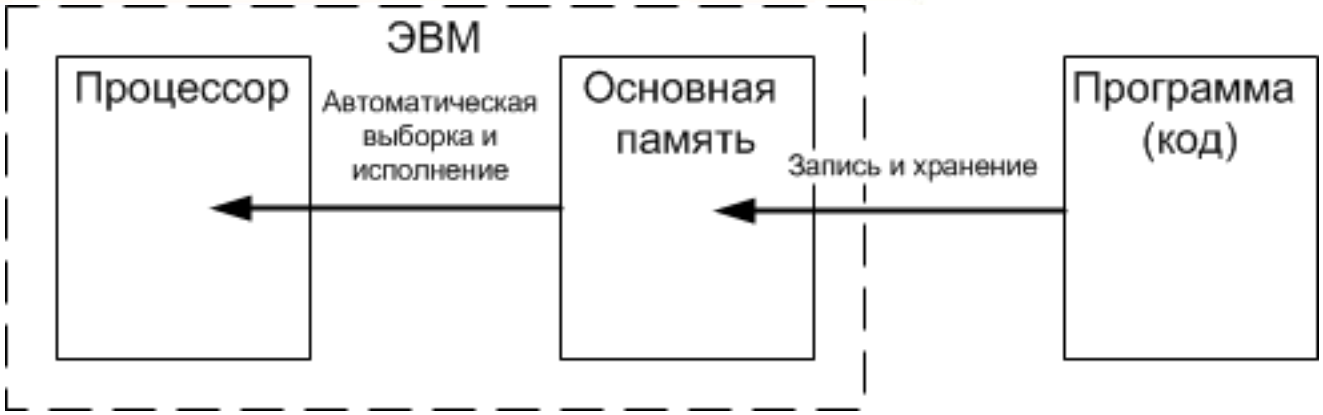
Пример
Алгоритм вычисления площади трапеции с основаниями A и B, высотой H {S=0.5*(A+B)*H} .

| Команды (трехадресные) | Комментарий |
|---|---|
| +, A, B,P1; | P1=A+B |
| *, P1,H,P2; | P2=P1*H |
| *,P2,”0.5”,S; | S=R2*0.5 |



# Принцип хранимой программы

Последовательность команд управления процессом вычисления (**программа**) должна храниться в основной памяти ЭВМ и извлекаться оттуда автоматически в процессе исполнения программы.




**Первые ЭВМ с хранимой в памяти программой**
EDVAC (1944-1951, США**)**: Дж. Экерт, Дж. Мокли, Дж. Фон Нейман, Г. Голдсайн

# Принцип однородности памяти (отсутствия различий между командами и данными )

Команды и данные хранятся в одной и той же памяти и внешне неразличимы – нет идентификаторов. Распознать их можно только по способу использования.


Компиляция, самомодификация

# Принстонская и Гарвардская архитектуры

**Принстонская** – данные и команды программы хранятся в единой памяти; допустима *динамическая модификация команд;* используется во многих микропроцессорах – x86, ARM.

**Гарвардская** – данные и команды хранятся в раздельных (физически) блоках памяти; защита программ от модификации; используется при организации КЭШ-памяти, в некоторых процессорах управляющих систем (микроконтроллерах) для безопасности.



**иногда Принципы фон Неймана дополняется еще одной трактовкой**

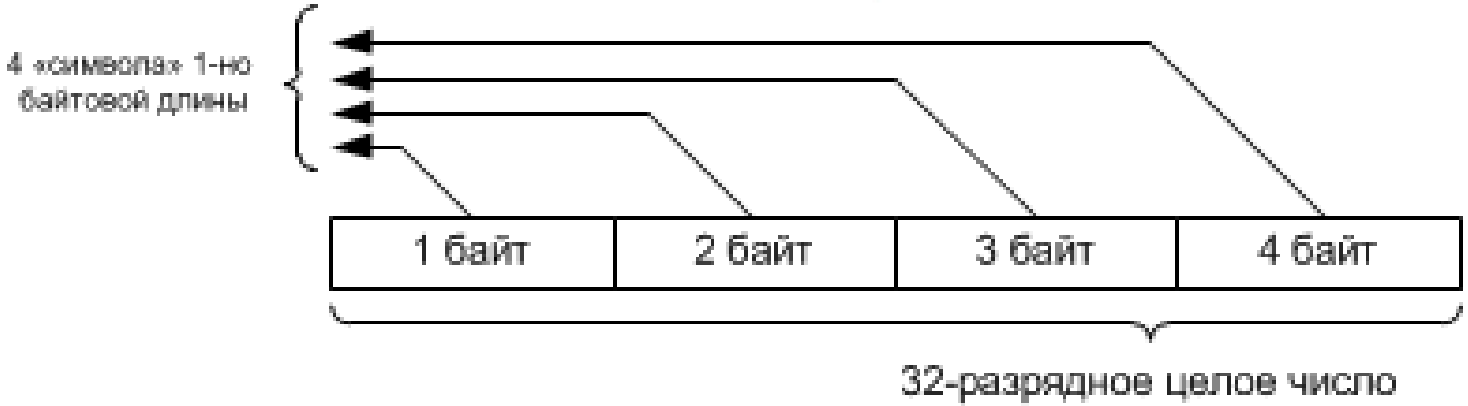
## Принципа однородности памяти

время чтения (записи) из/в любой ячейки одинаково
для всех ячеек

# Принцип отсутствия различий в семантике данных

Данные различных типов и форматов хранятся в одной и той же памяти и внешне неразличимы – нет специальных идентификаторов типов данных. Распознать тип данных можно только по способу использования.

Например: 4 байта в памяти могут быть интерпретированы как 4 независимых символа или как одно 32-х разрядное целое число.




*Альтернативный принцип – это использование идентификаторов типов команд и/или данных -ТЕГОВ*

# Принцип линейности и адресуемости памяти

Память ЭВМ представляет совокупность последовательно пронумерованных ячеек. Номера (адреса) ячеек образуют неразрывную (линейную) последовательность, причем процессору в любой момент времени доступна любая из них по ее номеру (адресу). **Ячейка** — минимально адресуемый элемент (минимальная единица, к которой можно обращаться). Обычно 1 байт. Ячейка состоит из разрядов.

Различают ячейку и **машинное слово.** С точки зрения архитектуры, машинное слово – это **минимальный** объём данных, которым могут обмениваться между собой различные узлы машины или **наибольшая длина данного**, выбираемого за одно обращение (16, 32, 64 бит).

За счет возможности считывания ячеек памяти по порядку, принцип линейности памяти обеспечивает другой ключевой принцип – последовательного исполнения программы.

# Принцип последовательного выполнения команд программы и возможности перехода

- выполнение любой программы в ЭВМ сводится к выполнению ее команд в порядке возрастания номера (адреса) ячейки в памяти, где хранится команда.

- линейная последовательность выполнения команд может быть нарушена при необходимости путем исполнения специальных команд перехода или при обработки исключений

*Данный принцип управления исполнением программы также называют* ***принципом управления потоком команд****.*



# Принцип двоичного кодирования

Данные и команды кодируются в виде двоичных чисел, называемых *словами*.

**Двоичное кодирование обеспечивает:**

- Простоту аппаратной поддержки в виде ключевых электронных схем;
- Помехоустойчивость двухуровневых дискретных сигналов;
- Простоту реализации арифметических операций;
- Простоту формального синтеза цифровых электронных схем.



# Принцип иерархической памяти

- Память ЭВМ представляет собой иерархию запоминающих блоков, которые обладают различными параметрами и строятся на основе запоминающих элементов с различными принципами действия.

- Основными параметрами является пара быстродействие-объем.

- В современных ЭВМ количество уровней памяти составляет 5-8 шт. (регистровая, КЭШ различных уровней, основная, внешняя)



# Принцип низкоуровневости машинного языка

Команды машинного языка (которые способна понимать и выполнять ЭВМ) должны осуществлять только элементарные действия над данными. Поддерживаемое множество таких действий должно быть достаточным, чтобы реализовать более сложные функции в виде программы из машинных команд.

*По уровню машинных команд различают CISC (complex instruction set computer) и RISC (reduced instruction set computer) ЭВМ.*

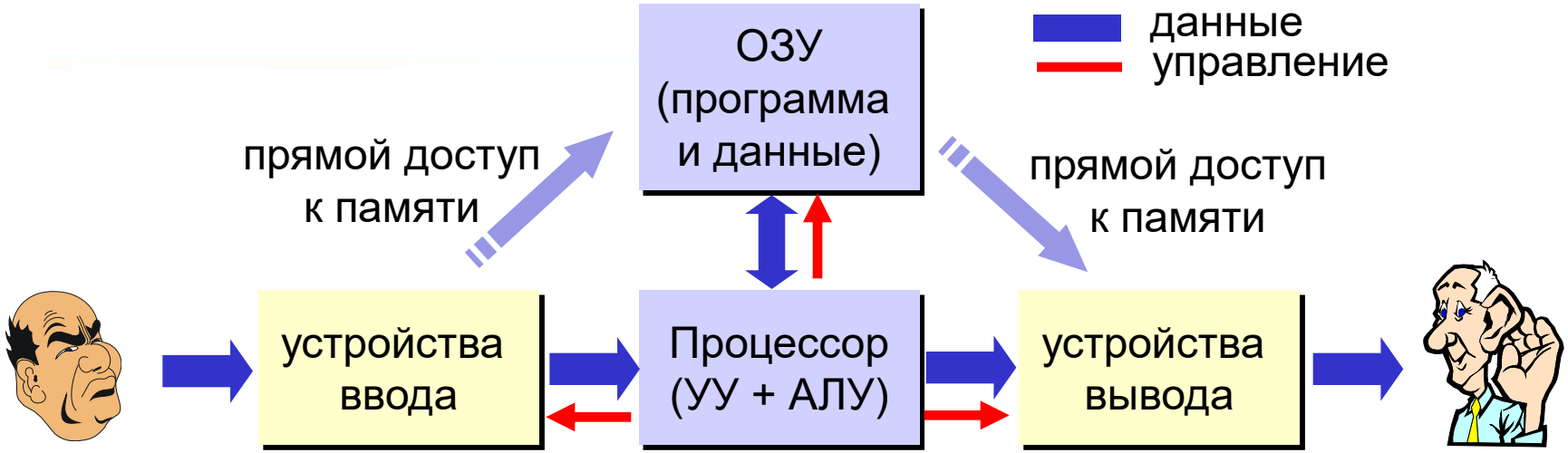


# Принцип микропрограммирования

Машинные команды не должны напрямую управлять аппаратурой (схемами) ЭВМ, что привело бы к усложнению их формата. Каждая машинная команда представляется в виде последовательности еще более простых команд, содержащих данные о состоянии управляющих электрических сигналов – **микропрограммы**.



# Архитектура ЭВМ Дж. фон Неймана



Революционность идей Фон Неймана - **Строгая специализация** устройств (жёстко распределить выполняемые ЭВМ функции между различными устройствами)

**АЛУ** ( вид ОУ) может выполнить следующие действия.

1.Считать содержимое некоторой ячейки памяти
2.Записать машинное слово в некоторую ячейку памяти
3. Выполнить различные операции над данными в своих регистрах

**УУ** (Устройство управления):

Автоматическое исполнение программы (выборка и декодирование команд, управление чтением и записью операндов в память, настройка ОУ);

**Регистры УУ**

**- регистр команд** (IR) хранит текущую выполняемую команду

- **счётчик адреса** =счётчик команд (IP) хранит адрес следующей выполняемой команды

После выполнения очередной команды ЭВМ "не помнит", какую команду она выполнила - основа для **мультипрограммного** режима работы компьютера



**Современные ЭВМ в той или иной степени нарушают практически все принципы Фон Неймана** (кроме принципа автоматической работы)

- различают команды и данные
- нарушают принцип однородности и линейности памяти
- нарушают  принцип последовательного выполнения команд

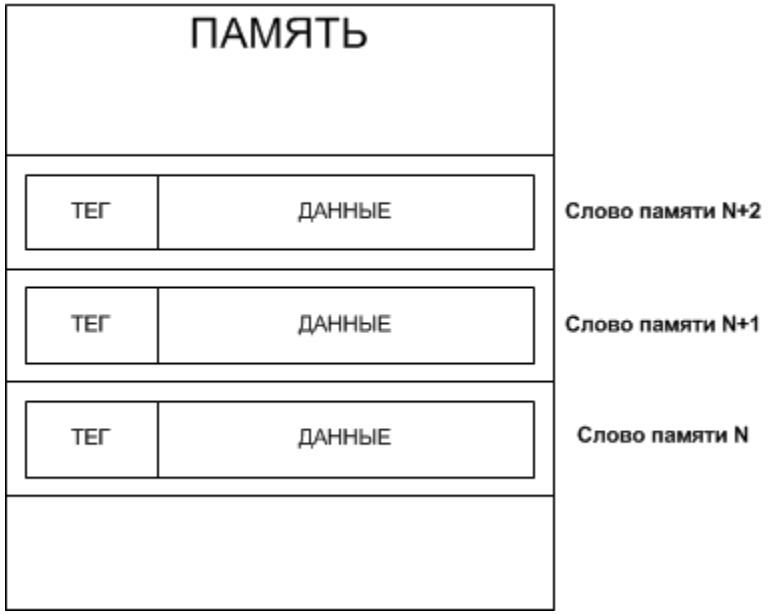
# Примеры ненеймановских ЭВМ

- Теговые ЭВМ - нарушен принцип отсутствия различий в семантике данных
- Потоковые и редукционные ЭВМ – нарушен принцип последовательного выполнения команд программы
- Нейровычислители – работают по принципу сетей узлов обработки с настройкой функций обработки через обратную связь.
- Другие (параллельные, векторные, матричные ЭВМ. Транспьютеры. Систолические матрицы. Теговые и дескрипторные ЭВМ)



# Теговые ЭВМ

В Теговых ЭВМ тип, формат данных записаны в специальном поле – Теге, который хранится вместе с данными как неотъемлемая их часть.

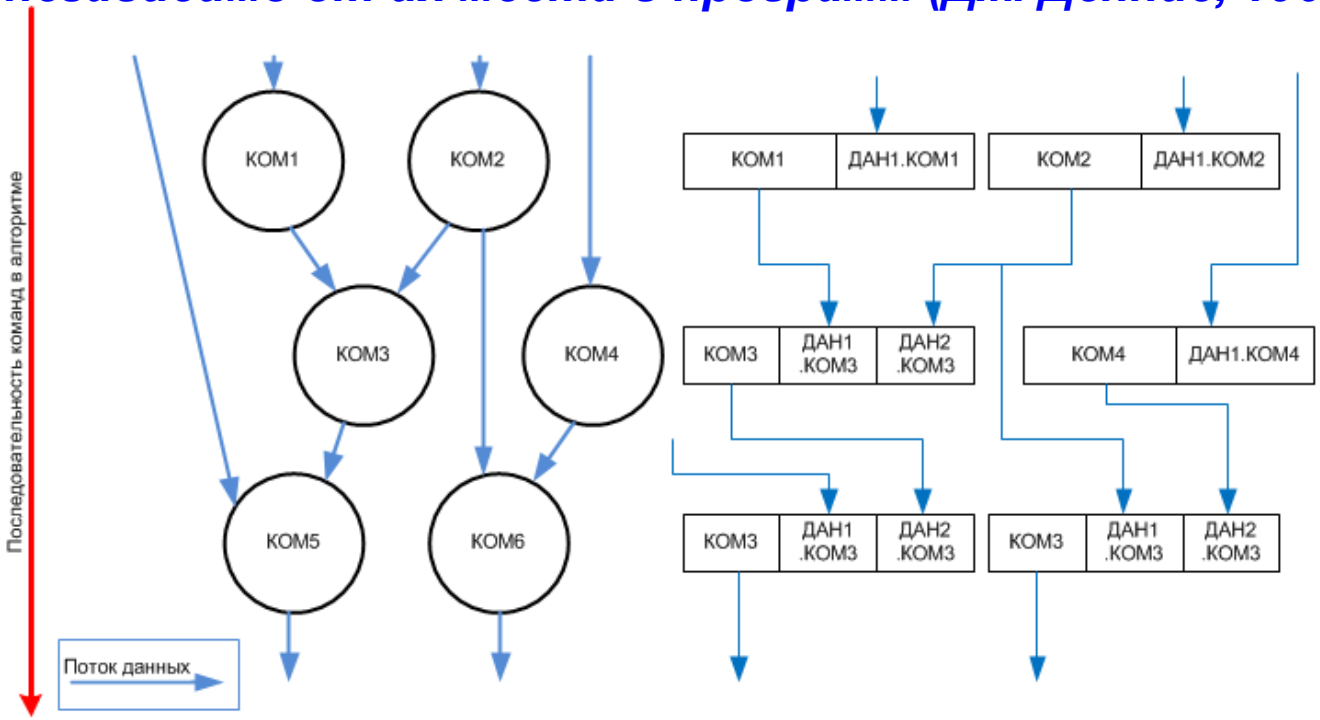



**+** короче команда, меньше размер программ, проще компилятор, более жесткий контроль типов

**-** больший размер данных, более сложное и (возможно) длительное исполнение команд

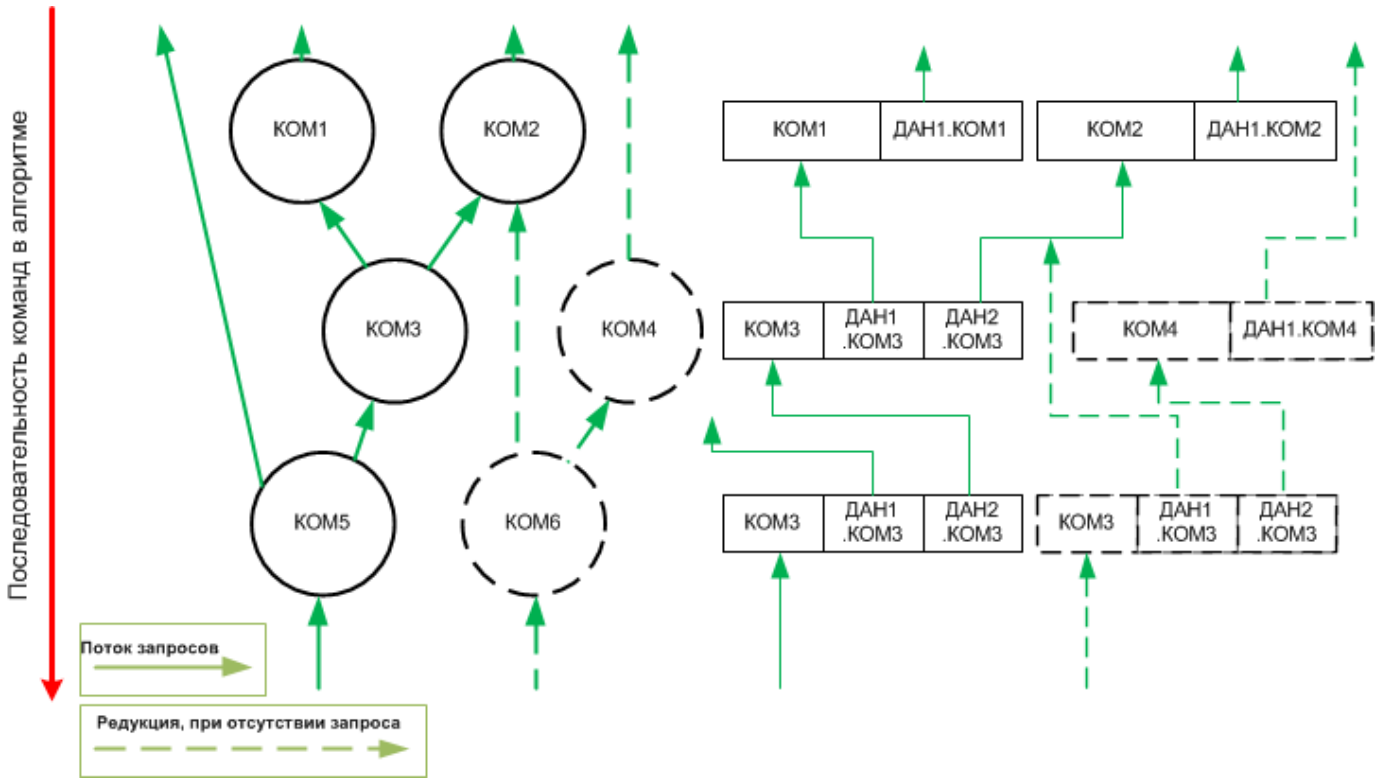
# Потоковые ЭВМ с управлением потоком данных (data driven)

***должны выполняться все команды, для которых есть данные, независимо от их места в программ (Дж. Деннис, 1967 г.)***




Команды выполняются по готовности данных от ПРЕДЫДУЩИХ команд. Если ОДНОВРЕМЕННО готовы данные для нескольких команд, то они выполняются параллельно – для этого предусмотрено несколько операционных блоков. Мультитредовые (гипертредовые) ЭВМ – вариант data driven организации, где вместо команд – нити (треды, thread)

# Редукционные ЭВМ с управлением потоком запросов (demand driven)




Команды выполняются по запросам данных от ПОСЛЕДУЮЩИХ команд. Если данные нигде не используются, то команды, их вырабатывающие, не запрашиваются и не исполняются. Граф потока команд-данных сокращается – редуцируется. Выполнение программы начинается с «последней команды»

# Повышение производительности в фон неймановских машинах

- увеличение **разрядности** обработки данных (16 бит, 32 и 64 бит);
- активное использование **конвейеризации** при выборке и обработке команд;
- активное использование **кэш-памяти**, т.е. модулей памяти, которые являются буферными между процессором и оперативной памятью.

# 4. Машина Тьюринга



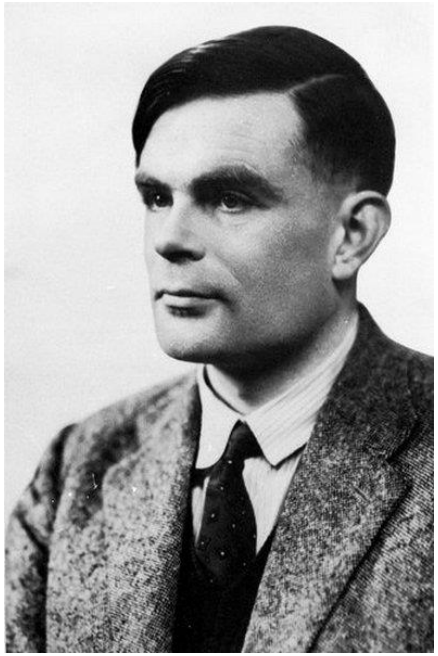
Это абстрактная машина, предложенная Тьюрингом в 1936 г. в качестве универсальной алгоритмической модели.

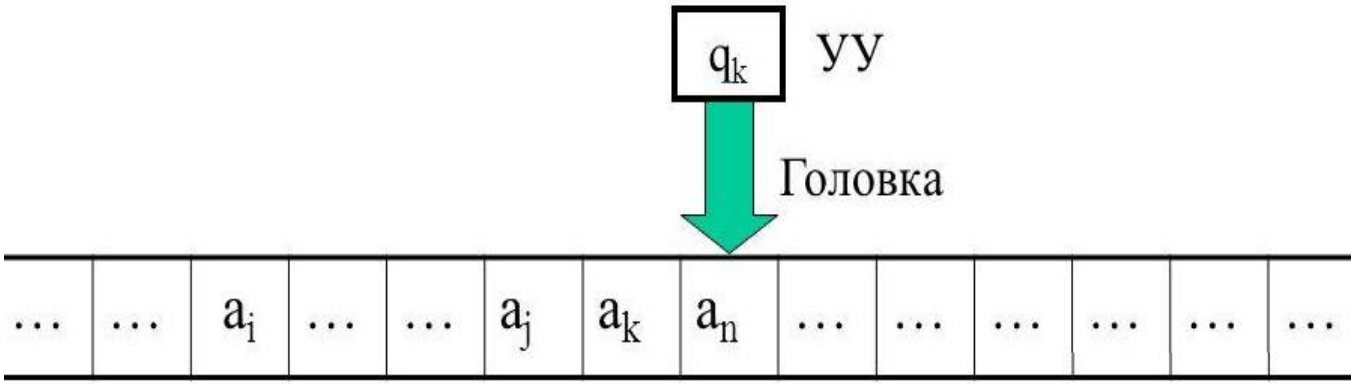
Она состоит из трех частей:

•лента;

•головка;

•управляющее устройство (УУ).





# Машина Тьюринга против машины фон Неймана

Обе машины – абстрактные модели.

**Машина Тьюринга**

- абстрактный исполнитель алгоритмов; может обрабатывать входные данные любого объема;
- определяет вычислительные возможности;
- служит **доказательством проблемы решения**;
- не является архитектурной моделью.

**Машина Фон Неймана**

- абстрактный исполнитель, т.к. детали в архитектуре этого вычислителя не конкретизированы;
- архитектура для построения реальных компьютеров; **руководство, как создавать компьютеры**;
- ничего не говорит о возможностях вычислений.



# 5. Архитектура и организация ЭВМ



# Архитектура ЭВМ

**Архитектура** – это множество ресурсов ЭВМ, доступных пользователю на структурном (логическом, функциональном) уровне, **без детализации** способов взаимодействия процессоров, устройств памяти, внешних устройств и программных средств.

При изучении архитектуры рассматривают:

- состав и характеристики процессоров, включая системы команд;
- состав и характеристики устройств памяти и ВУ;
- состав программных средств разработки ПО;
- вид ОС и режимы обработки данных.

**Принстонская архитектура (фон-неймановская) -** хранение команд и данных в общей памяти. Все современные настольные ПК

**Гарвардская архитектура -** раздельное хранение команд и данных. Большинство микроконтроллеров.

## Достоинства и недостатки Гарвардской архитектуры

Первый компьютер - Марк I. Разработан и построен в 1941 году.
Архитектура не использовалась до конца 70-х годов.

**Достоинства** разработчикам микроконтроллеров (МК)

- **гибкость и универсальность Принстонской архитектуры не имеют большого значения для МК**.
  Анализ реальных программ управления: объем памяти данных МК на порядок меньше требуемого объема памяти программ. => Использование единого адресного пространства приводило к увеличению формата команд за счет увеличения числа разрядов для адресации операндов.
- **обеспечивает более высокую скорость выполнения программы по сравнению с Принстонской за счет возможности реализации параллельных операций**.
- **невозможно производить операцию записи в память программ**, что исключает возможность случайного разрушения управляющей программы

**Недостаток**: усложняется схема реализации доступа к памяти



# Организация ЭВМ

**Организация** – это **способы распределения** функций, установления связи и взаимодействия процессоров, устройств памяти и внешних устройств, **используемые для реализации возможностей, заложенных в архитектуре**. При изучении организации рассматривают:

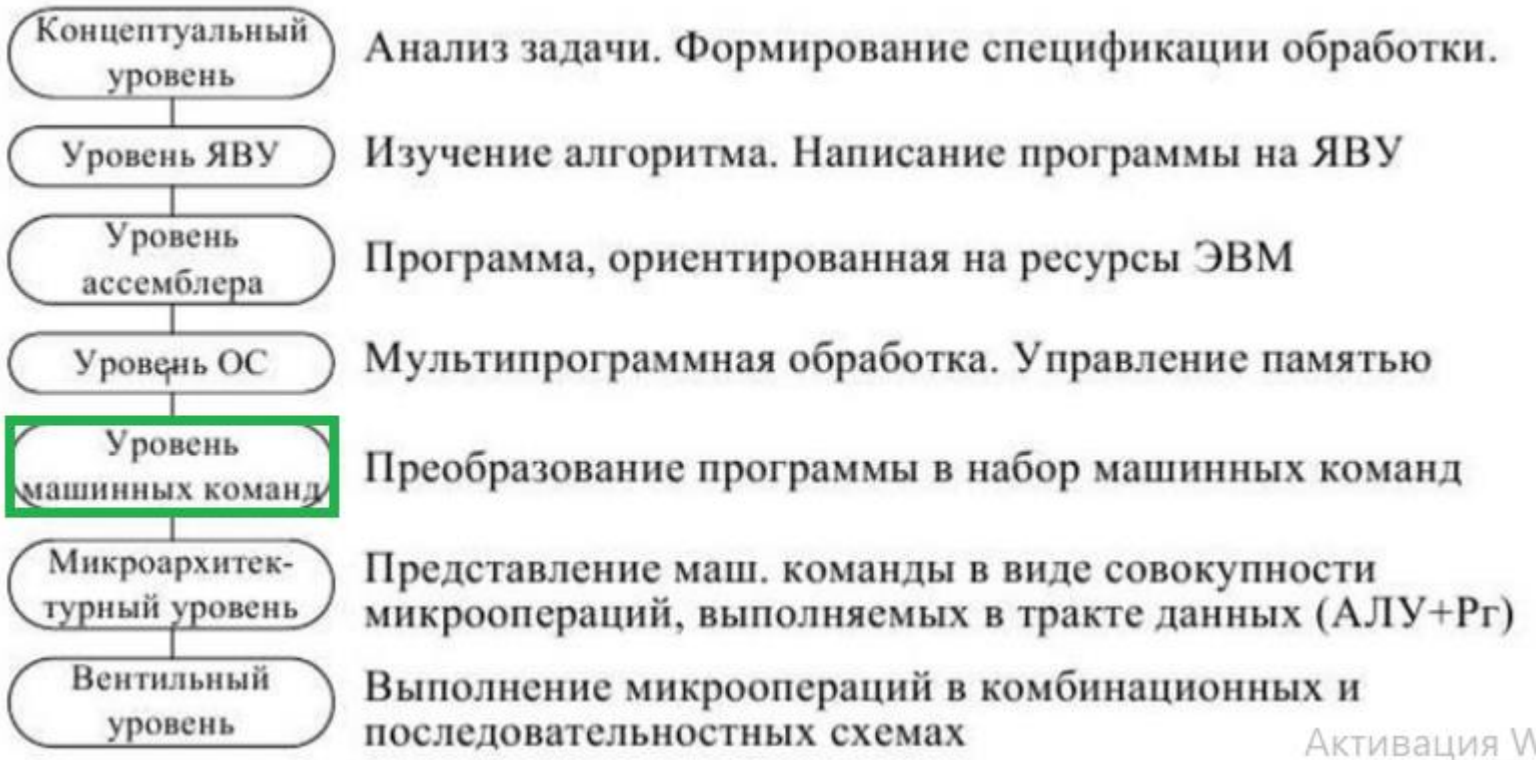
- представление и форматы данных;
- уровни памяти и их взаимодействие;
- состав и форматы машинных команд;
- систему прерываний;
- способы обмена данными.

**Реализация** – **способы** технического **исполнения** конкретных устройств, линий или шин связи и протоколов взаимодействия между ними.

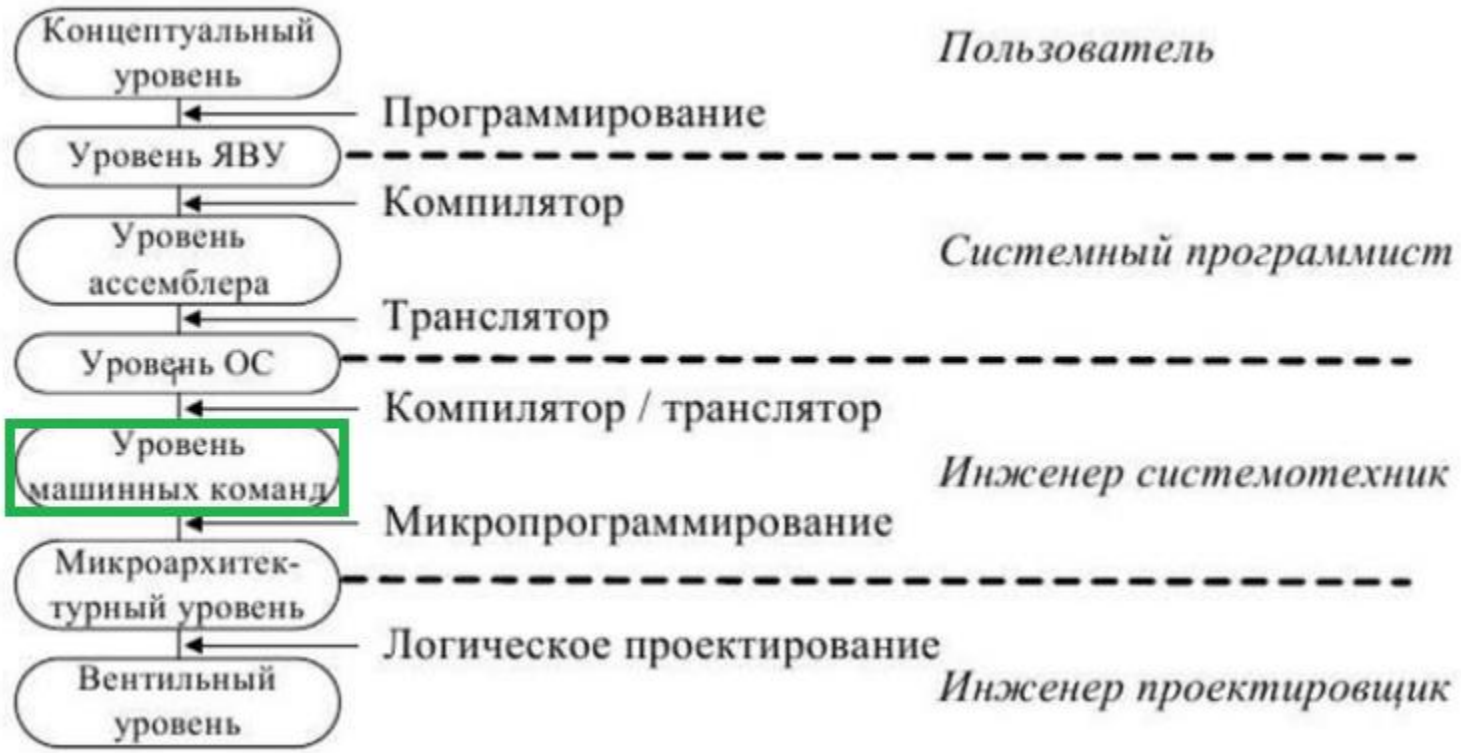
Обычно **на уровнях организации и реализации происходит перераспределение функций** между аппаратными и программными средствами. Это порождает семейство машин одной архитектуры, но разной производительности



# Уровни организации ЭВМ



# Уровни организации ЭВМ

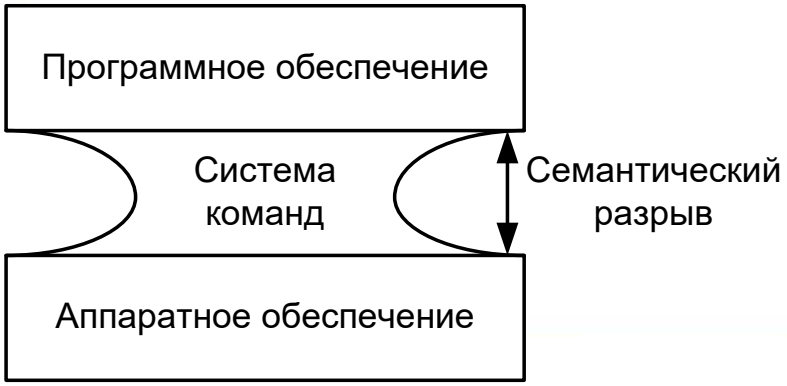


# Особенности многоуровневой организации

- Каждый верхний уровень интерпретируется одним или несколькими нижними уровнями.
- Каждый из уровней можно проектировать независимо.
- Модификация нижних уровней не влияет на реализацию верхних.
- Чем ниже уровень реализации программы, тем более высокая производительность достижима.



# Понятие семантического разрыва




**Языки высокого уровня (ЯВУ) имеют следующие характеристики**:

- память состоит из набора дискретных именуемых переменных.
- используются многомерные, а не просто линейные данные.
- существует резкое разграничение между данными и командами.
- тип данных определяет и операции, выполняемые над ними

**Принципы, на которых основывается архитектура фон Неймана, не согласуются с принципами ЯВУ**

# Понятие семантического разрыва

Система команд первых 3-х поколений ВТ подстраивалась под потребности технологий программирования, включая в состав новые, более развитые команды. Догнать высокие темпы развития технологий программирования процессоростроители не могли. Образовался семантический разрыв.

**Факт различия принципов, лежащих в основе ЯВУ и тех принципов, которые определяют архитектуру ЭВМ, называется семантическим разрывом.** Проблема выражается в неоправданном падении производительности ВС.

**Способы его преодоления** зависят от типа архитектуры ЭВМ:

1. специализация машин для традиционных ВМ со сложным набором команд (CISC) (аппаратная реализация графических преобразований)
2. переход к ВМ с сокращенным набором команд (RISC) (реализация операторов ЯВУ на основе команд RISC-процессора оказывается столь же эффективной, как и аппаратная реализация)



# 6. Эволюция микропроцессорных архитектур



Узловой момент эволюции - концепция ВМ с хранимой в памяти программой (1945 г. фон Нейман).

Относительно неё история развития ВТ представляется в виде **3-х этапов**:

- донеймановский период,
- эра ВМ с фон-неймановской архитектурой
- постнеймановская эпоха (эпоха параллельных и распределённых вычислений).



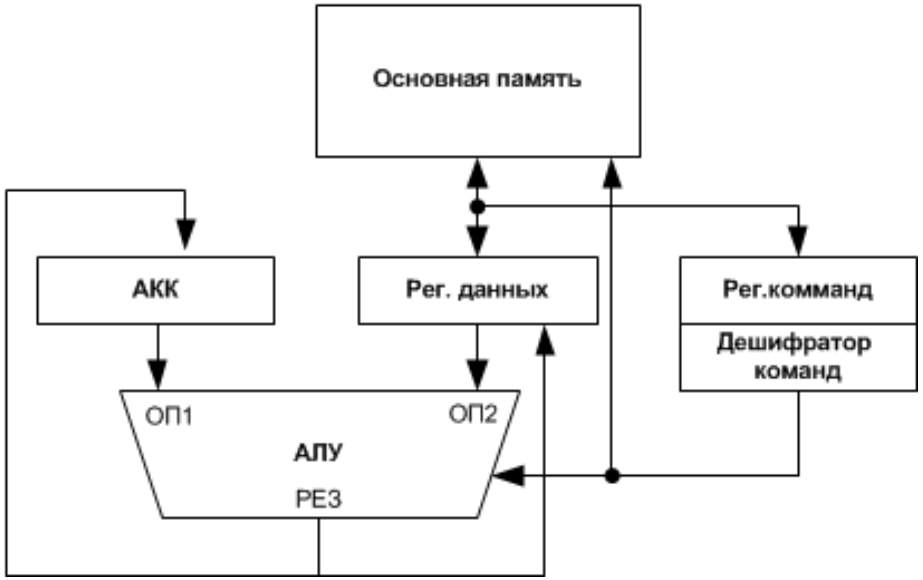
Эволюция МП архитектур ассоциировалась с революционными технологическими прорывами.

**1-е поколение ВТ** (1937–1953):

- элементная база - электронные лампы.
- программирование - машинный код, затем ассемблер.
- **аккумуляторная архитектура**.

# Аккумуляторная архитектура

Один из операндов должен обязательно находиться в специальном регистре-аккумуляторе. Результат также сохраняется в аккумулятору. Извлечение 1-го операнда из аккумулятора, извлечение 2-го операнда из ОП, выполнение действий, помещение результата в аккумулятор).




+ хранение в команде адреса только одного операнда
+ один канал считывания данных из основной памяти

- аккумулятор – «узкое горлышко» обрабатываемого потока данных

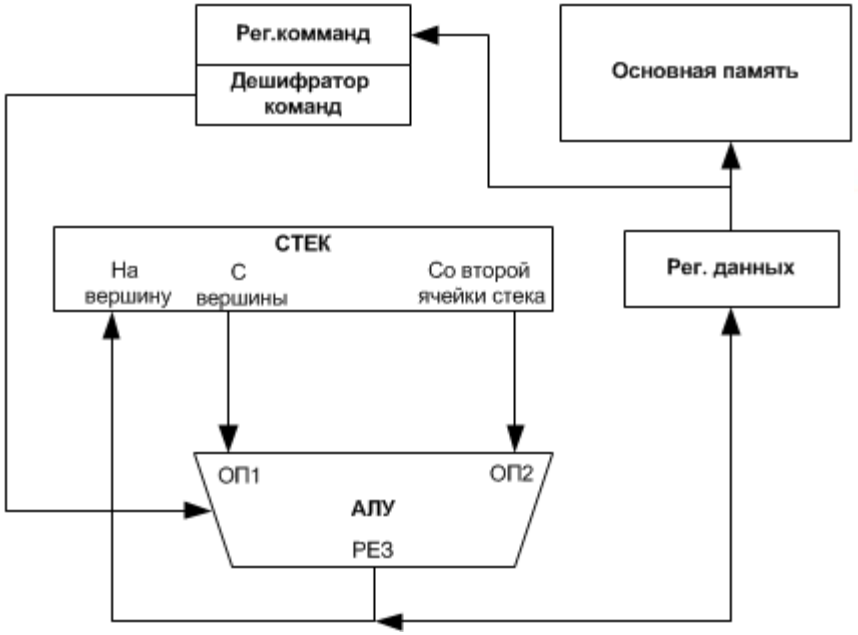
**2-е поколение ВТ** (1954–1962):

- элементная база - транзисторы.
- **стековая архитектура**.
- **регистровая архитектура**.



# Стековая архитектура

Стек - регистровая структура, работающая по принципу LIFO. Стек используется в качестве РОН. Стековая архитектура похожа на аккумуляторную, но вместо аккумулятора – стек.




Извлечение операндов из вершины стека, выполнение действий, помещение результата в вершину стека
При ограниченном объёме доступной памяти стековая архитектура имела преимущество.

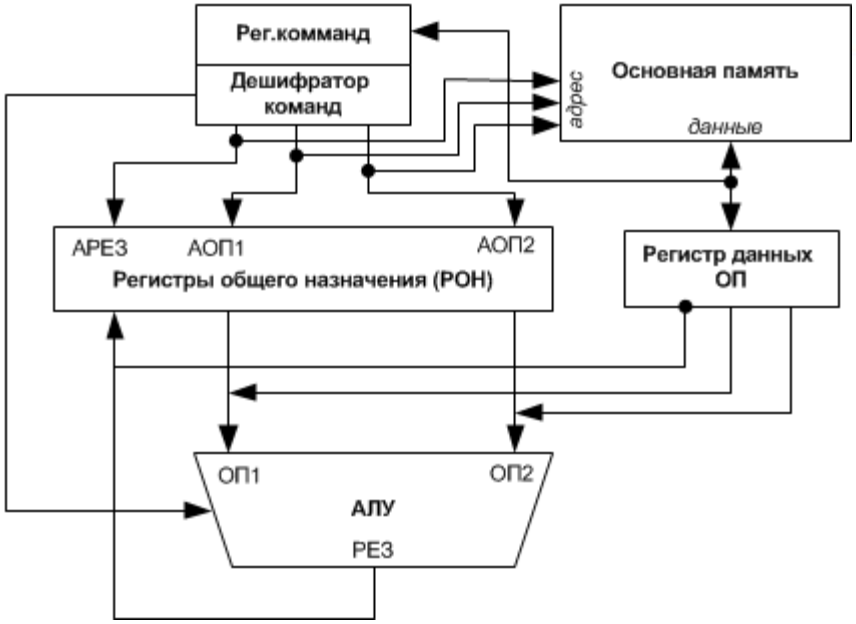
+ простота аппаратной структуры
+ «короткие» команды

- необходимость пересылки данных в стек перед обработкой или их постоянное хранение в стеке
- необходимо четко организовывать данные в стеке под порядок их обработки

# Регистровая архитектура

Появление полупроводниковых ЗУ → Увеличение объема ОП →отказ от стековой архитектуры

В состав процессора входит большое количество РОНов. В команде необходимо указать номера регистров, хранящих операнды, а также номер регистра результата.




\+ нет «узкого горлышка» - аккумулятора

\+ операнды из ОП и регистров используются в произвольной комбинации

\- сложная аппаратная структура с несколькими логическими каналами считывания из основной памяти

**3-е поколение ВТ** (1963–1972):

- элементная база – ИС
- многозадачные ОС
- **CISC** архитектура (ВМ со сложным набором команд)

**4-е поколение ВТ** (1973–1984):

- элементная база - БИС.
- ЯВУ → резкое снижение востребованности CISC инструкций (компиляторы использовали только небольшое подмножество из них) → переход к **RISC** архитектуре (ВМ с сокращенным набором команд)



**90-е годы:**

- **суперскалярные процессоры**
- увеличение числа конвейеров ограничивает дальнейшее распараллеливание на уровне команд
- необходимость **использования более высоких уровней параллелизма (**постнеймановская эпоха )



# Эволюция микропроцессорных архитектур

## Перспективные пути развития

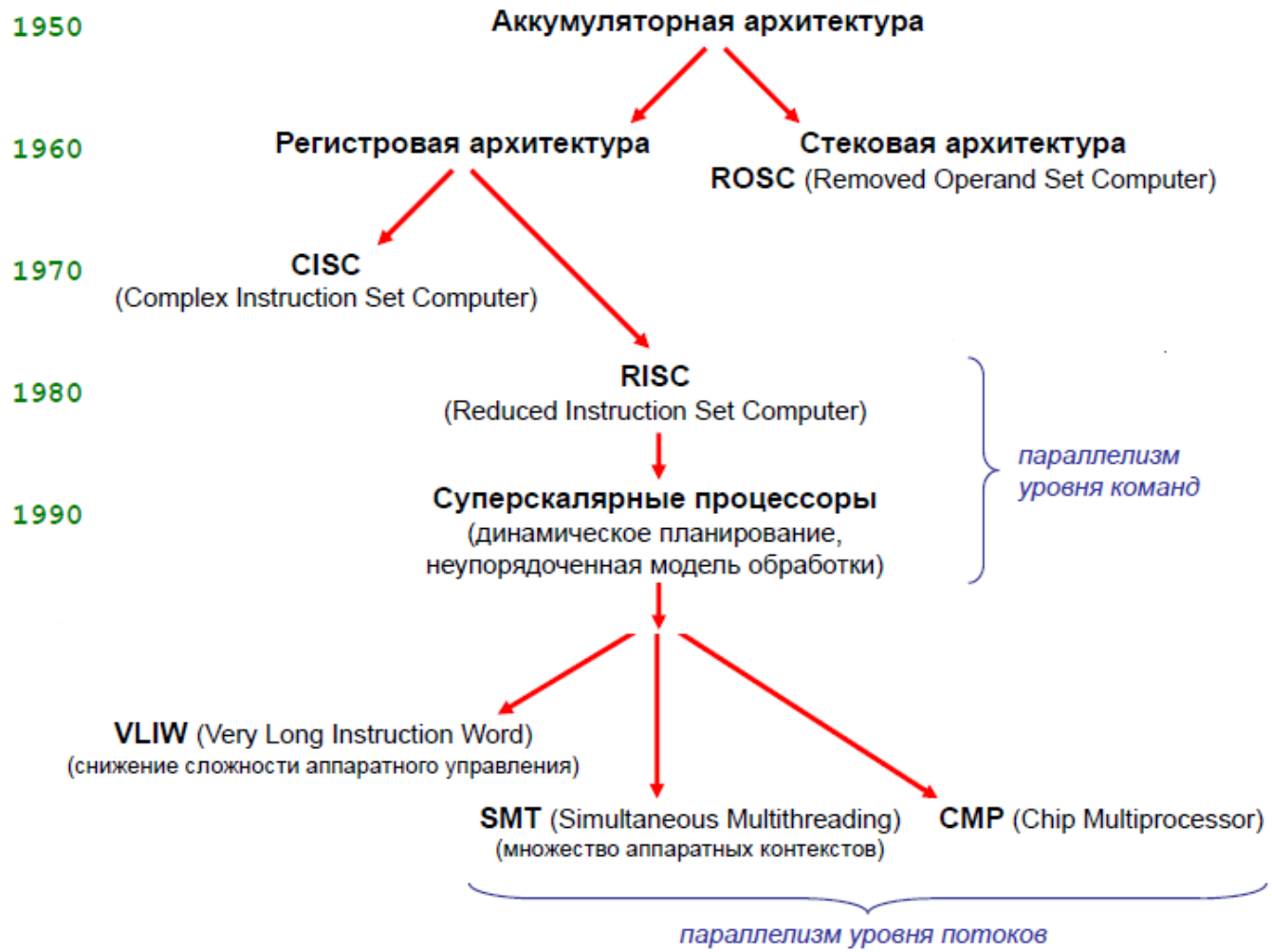
- **VLIW** (Very Long Instruction Word) - снижение сложности аппаратного управления

- **SMT** (Simultaneous Multithreading) - многопоточное процессирование - технология, позволяющая выполнять инструкции из нескольких потоков выполнения (программ) на одном суперскалярном конвейере.
  Потоки разделяют один суперскалярный конвейер процессора (ALU, FPU, Load/Store)

- **CMP** (Chip Multiprocessor) - многоядерное процессирование
  - Процессорные ядра размещены на одном чипе (Processor chip)
  - Ядра процессора могут разделять некоторые ресурсы (например, кеш-память)



# Эволюция микропроцессорных архитектур



# Альтернативные архитектуры ЭВМ

Счетчик команд -"узкое горло".

***“Не-фон-Неймановская" архитектура*** допускает одновременный анализ более одной команды.

Наиболее перспективное направление увеличения скорости решения прикладных задач - широкое внедрение **идей параллелизма** в работу ВС.

***ВС*** - ВМ, позволяющая выполнять параллельную обработку частей программы или различных программ на нескольких процессорных элементах



# 7. Уровни параллелизма



# Уровень заданий (Распараллеливание на уровне задач)

ВС решает различные несвязанные задания.


Решаемая задача естественным образом состоит из **независимых подзадач**, каждую из которых можно решить отдельно.
*Пример*: сжатие аудио-альбома. Каждая запись может обрабатываться отдельно, так как она никак не связана с другими.

Распараллеливание на уровне задач нам демонстрирует ОС, запуская на многоядерной машине программы на разных ядрах

Данный вид распараллеливания не применим к однородной задаче.

# Уровень параллелизма данных

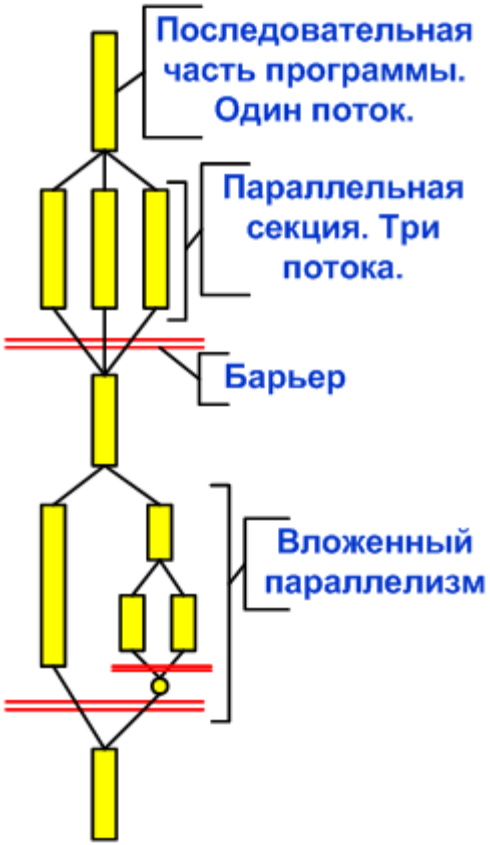
**Параллелизм заключается в применении одной и той же операции к множеству элементов данных**.
ВС позволяет обрабатывать части одной программы на различных процессорах.

*Пример:* архиватор, использующий для упаковки несколько ядер процессора. Данные разбиваются на блоки, которые единообразным образом обрабатываются (упаковываются) на разных узлах.

Используется при решении задач численного моделирования. Счетная область разбивается на геометрические объекты и ячейки, вошедшие в эту область, отдаются на обработку определенному ядру - **геометрический параллелизм**.
Похожим на распараллеливание на уровне задач, но более сложен в реализации.



# Уровень распараллеливания алгоритмов




Распараллеливание отдельных процедур и алгоритмов.

Примеры: алгоритмы параллельной сортировки, умножение матриц, решение системы линейных уравнений.
На этом уровне абстракций удобно использовать технологию параллельного программирования OpenMP.

**OpenMP** (Open Multi-Processing) — это набор директив компилятора, библиотечных процедур и переменных окружения, которые предназначены для программирования многопоточных приложений на многопроцессорных системах.
Используется модель параллельного выполнения «ветвление-слияние».

# Параллелизм на уровне инструкций (Уровень команд)

Параллельная обработка процессором нескольких инструкций. Пакетная обработка нескольких элементов данных одной командой процессора - технологии MMX, SSE, SSE2.

Программа представляет собой поток инструкций выполняемых процессором. Можно изменить порядок этих инструкций, распределить их по группам, которые будут выполняться параллельно, без изменения результата работы всей программы - это параллелизмом на уровне инструкций. Для его реализации используется несколько конвейеров команд, технология предсказания команд

***Параллельные вычисления*** – реализующаяся тем или иным способом параллельная обработка данных на многих вычислительных узлах для повышения общей скорости расчета.



Параллелизм достигается при реализации следующих принципов:

- **Независимость функционирования подсистем ЭВМ**:
  - подсистемы ввода-вывода;
  - подсистема обработки;
  - подсистема памяти.
- **Избыточность элементов ВС**
  - АЛУ;
  - Процессоры;
  - ОЗУ;
  - Контроллеры ввода-вывода;
  - Кэш-память.
- **Конвейерная реализация обработки команд**
  - Обращение к памяти одновременно с обработкой команд
  - Спекулятивная загрузка.



В научной литературе много **различных названий**, характеризующих лишь общие принципы функционирования параллельных машин:

- векторно-конвейерные,
- массивно-параллельные,
- компьютеры с широким командным словом,
- систолические массивы,
- гиперкубы,
- спецпроцессоры и мультипроцессоры,
- иерархические и кластерные компьютеры,
- dataflow,
- матричные ЭВМ и многие другие.



## Предложено много подходов к систематизации всех средств ВТ.

Для классификации компьютеров использовались следующие **классификационные признаки**:

- принцип действия;
- используемая элементная база;
- назначение;
- размеры и вычислительная мощность;
- особенности архитектуры.

- **По принципу действия** ЭВМ делятся на цифровые, аналоговые и гибридные.

- **По этапам создания** ЭВМ условно делятся на поколения с учетом используемой элементной базы.

- **По назначению** ЭВМ можно разделить на: специализированные, универсальные и проблемно-ориентированные.

- **По размерам и вычислительной мощности** ЭВМ делятся на суперЭВМ, большие, малые, сверхмалые. Эта классификация потеряла свою актуальность. Можно говорить только о существовании класса суперЭВМ (суперкомпьютеров).

- Классификация **с учетом особенностей архитектуры**
  Попытки систематизировать все множество архитектур начались после **опубликования М.Флинном** 1-го варианта классификации ВС в конце 60-х годов и непрерывно продолжаются по сей день.



Конец лекции 1



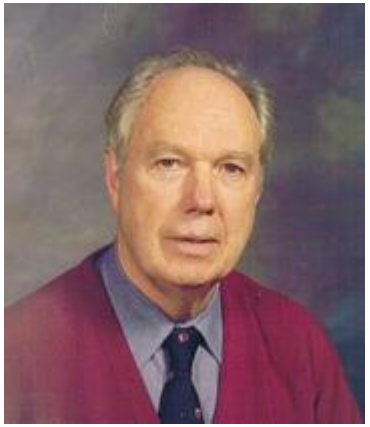
# 8. Классификация архитектур по параллельной обработке данных по Флинну



>10 различных классификаций ВС (*http://www.parallel.ru*)
*Ключевая – классификация Флинна -* классификация, основанная на **различных комбинациях количества потоков команд и данных.**
В 1966 году М.Флинном (Michael J. Flynn) был предложен подход к классификации архитектур ВС.
Основа - понятие **потока** – это последовательность элементов, команд или данных, обрабатываемых процессором.

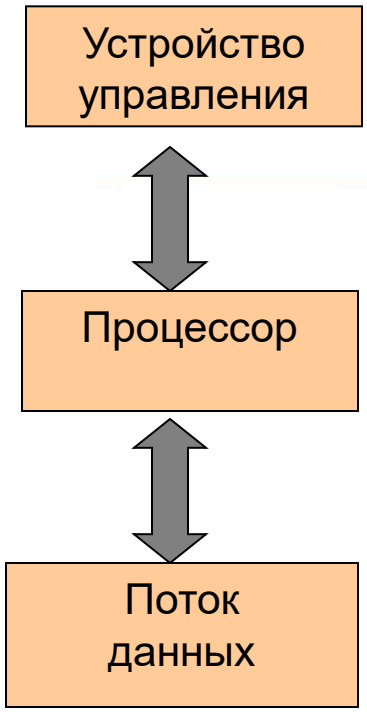
Классификация описывает 4 архитектурных класса:

- SISD = Single  Instruction Single Data
- MISD = Multiple Instruction Single Data
- SIMD = Single Instruction Multiple Data
- MIMD = Multiple Instruction Multiple Data



# SISD

**SISD** (single instruction stream / single data stream) – одиночный поток команд и одиночный поток данных.




К этому классу относятся последовательные компьютерные системы, которые имеют один центральный процессор, обрабатывающий только один поток последовательно исполняемых инструкций (машины фон-неймановского типа).

Для увеличения производительности может применяться **конвейерная** обработка.

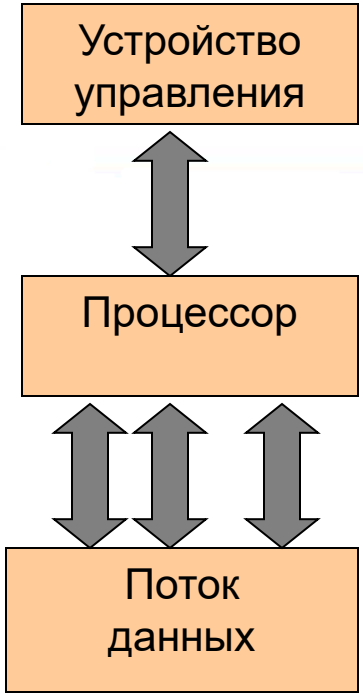
В случае **векторных** систем векторный поток данных следует рассматривать как поток из одиночных неделимых векторов.

Классические языки высокого уровня (C++) также ориентированы на программирование в классе SISD.
В настоящее время выпуск SISD процессоров почти прекращён (низкая производительность из-за низкого уровня параллелизма вычислений)

# SIMD

**SIMD** (single instruction stream / multiple data stream) – одиночный поток команд и множественный поток данных.

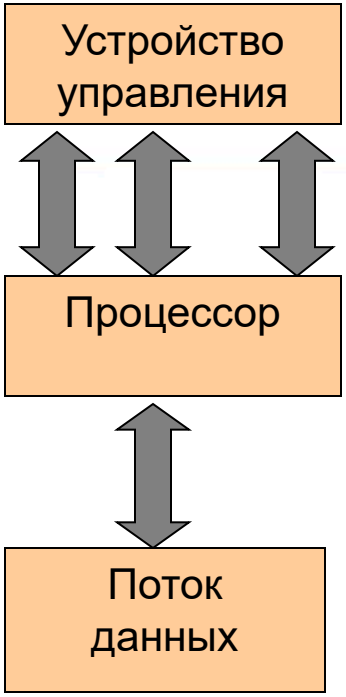
Эти системы обычно имеют большое количество процессоров, от 1024 до 16384, которые могут выполнять одну и ту же инструкцию относительно разных данных в жесткой конфигурации.
Представители класса SIMD **впервые достигли производительности порядка GFLOPS**




*Примеры*:

- **векторные процессоры** (операнд: скаляр или вектор). Векторную обработку внедряют и в процессоры классов SISD и MIMD: **технологии MMX , SSE, 3DNow!** - SIMD расширение IA32
- **матричные процессоры** (массив процессоров с единым потоком команд)
- **архитектура VLIW** (Very Long Instruction Word)/**EPIC** (Explicitly Parallel Instruction Computing)

# MISD

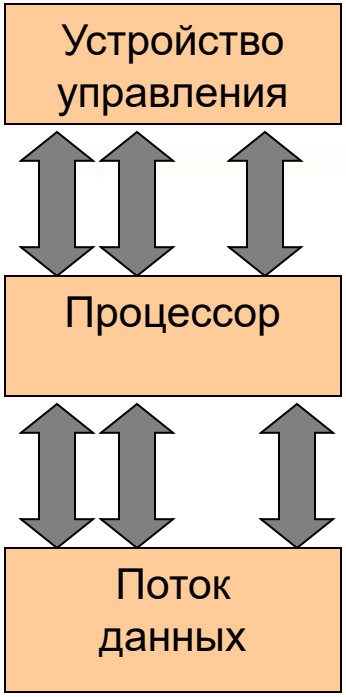



**MISD** (multiple instruction stream / single data stream) – множественный поток команд и одиночный поток данных.

Теоретически в этом типе машин множество инструкций должно выполняться над единственным потоком данных. До сих пор **нет ни одной реальной машины**, попадающей в данный класс

Ряд исследователей относят конвейерные машины к этому классу.
К данному классу можно отнести появившиеся многоядерные компьютеры.

# MIMD




**MIMD** (multiple instruction stream / multiple data stream) – множественный поток команд и множественный поток данных.

Эти машины параллельно выполняют несколько потоков инструкций над различными потоками данных**. Отличие от многопроцессорных SISD-маши**н: команды и данные связаны, потому что они представляют различные части одной и той же задачи.

**Все современные процессоры** (общего и специального назначения) попадают в класс MIMD.

## Недостатки классификации Флинна:

- многие архитектуры (dataflow, векторно-конвейерные машины) четко не вписываются в данную классификацию
- класс MIMD чрезвычайно заполнен

**Для класса MIMD предложена классификация Хокни,** которая основана на используемых **способах организации оперативной памяти**.

Такой подход позволяет различать два важных типа многопроцессорных систем:

**Мультипроцессоры (multiprocessors)** или системы с общей разделяемой памятью

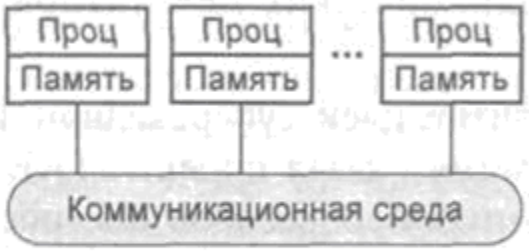
**Мультикомпьютеры (multicomputers)** или системы с распределенной памятью.



# 2 типа многопроцессорных систем


Параллельные компьютеры
с общей памятью

**Мультипроцессоры** – системы с общей памятью

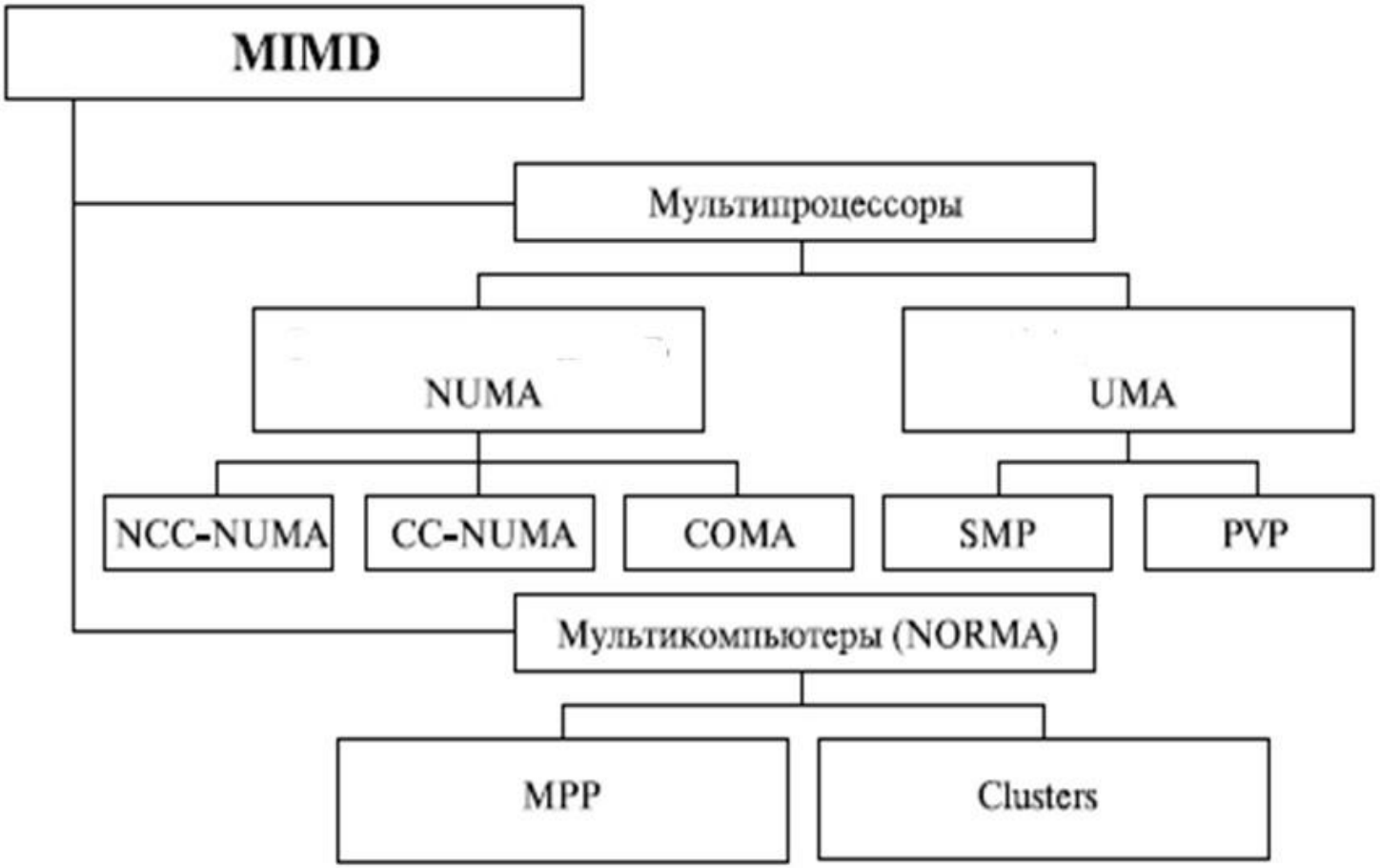



Параллельные компьютеры
с распределенной памятью

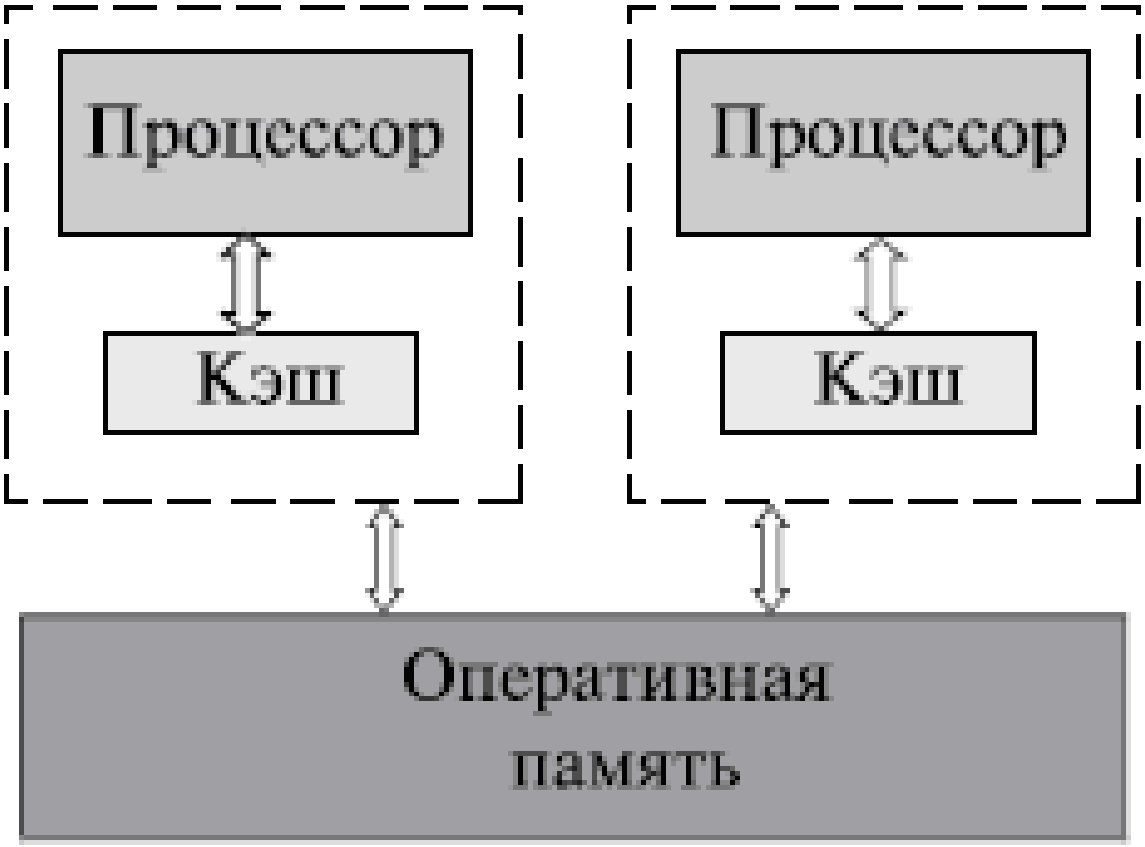
**Мультикомпьютеры -** системы с распределенной памятью

# 9. MIMD системы (Классификация Хокни)




MIMD
Мультипроцессоры
NUMA
UMA
NCC-NUMA
CC-NUMA
COMA
SMP
PVP
Мультикомпьютеры (NORMA)
MPP
Clusters

# Мультипроцессоры с однородным доступом к памяти (UMA)



Среди систем UMA (Unified Memory Access) выделяют:

- симметричные мультипроцессорные системы (**SMP** – simmetric multiproceccor)

- параллельные векторные системы(**PVP** – parallel vector processor)

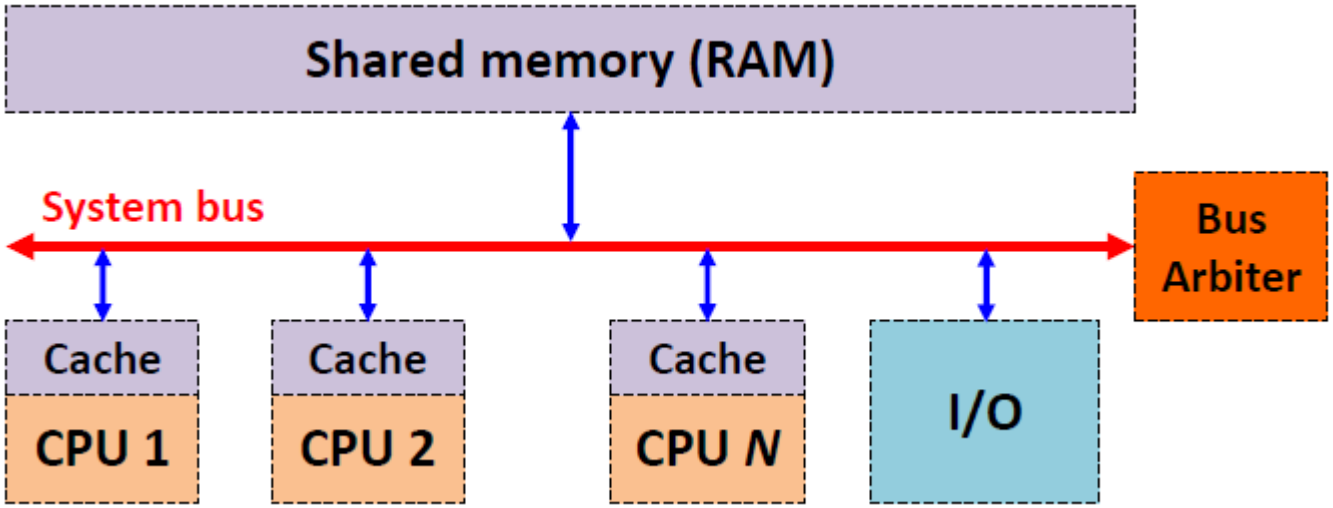


# *SMP-система*

Состоит из

- нескольких **однородных** процессоров
- массива общей памяти (обычно из нескольких независимых блоков).

Все процессоры имеют доступ к любой точке памяти с одинаковой скоростью.




Работает под управлением единой ОС (UNIX-подобной). ОС автоматически распределяет нити по процессорам, но возможна и явная привязка.

*Примеры:* рабочие станции на базе процессоров Intel (IBM, HP, Compaq, Dell)

## *Достоинства SMP-систем:*

- *простота и универсальность для программирования*. Архитектура SMP не накладывает ограничений на модель программирования - программирование в модели общей памяти (POSIX threads, OpenMP, …)

- *использование общей памяти увеличивает скорость такого обмена*, пользователь также имеет доступ сразу ко всему объему памяти. Для SMP-систем существуют довольно эффективные средства автоматического распараллеливания;

- *простота эксплуатации*. Как правило, SMP-системы используют систему кондиционирования, основанную на воздушном охлаждении, что облегчает их техническое обслуживание;

- *относительно невысокая цена*.



## ***Недостатки SMP-систем:***

- *доступ с разных процессоров к общим данным и* ***обеспечение***, в связи с этим, ***однозначности*** (когерентности) содержимого разных кэшей (cache coherence problem). Наличие общих данных при параллельных вычислениях приводит к необходимости синхронизации взаимодействия одновременно выполняемых потоков команд.

- системы с общей памятью ***плохо масштабируются*** (не более 32 *процессоров* в реальных системах). Контроллер памяти – узкое место.

Этот существенный недостаток SMP-систем не позволяет считать их по-настоящему перспективными. *Причиной плохой масштабируемости является то, что в каждый момент времени к памяти может обращаться только один процессор*.



## *Пример SMP-системы: м*ногоядерные процессоры (Multi-core processors)
## (CMP – Chip MultiProcessor)

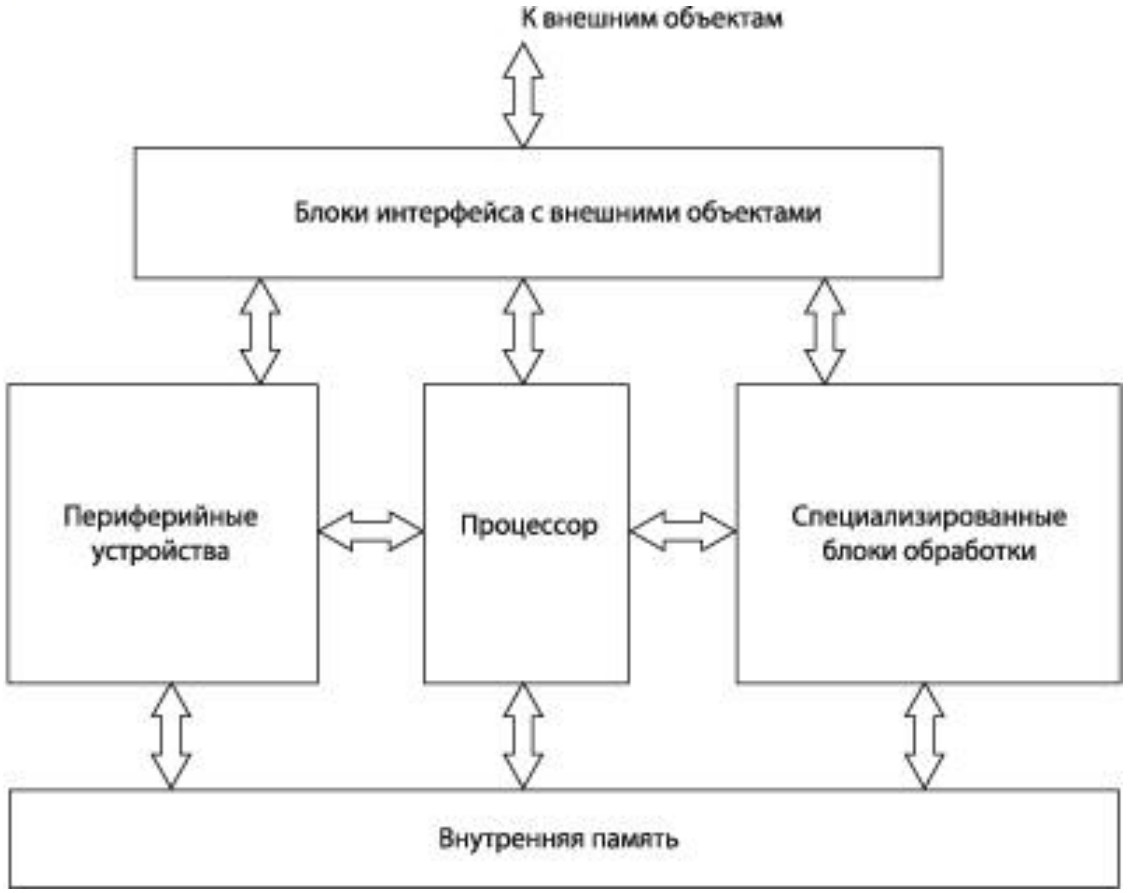
**Закон Мура: мощность последовательных процессоров возрастает практически в 2 раза каждые 18-24 месяцев** →Возможность размещения на одном чипе сразу нескольких процессоров.

**Многоядерный процессор** с точки зрения ОС - **это несколько одноядерных и он допускает симметричное мультипроцессирование**.

Иногда различные ядра имеют общий кеш второго или третьего уровней.

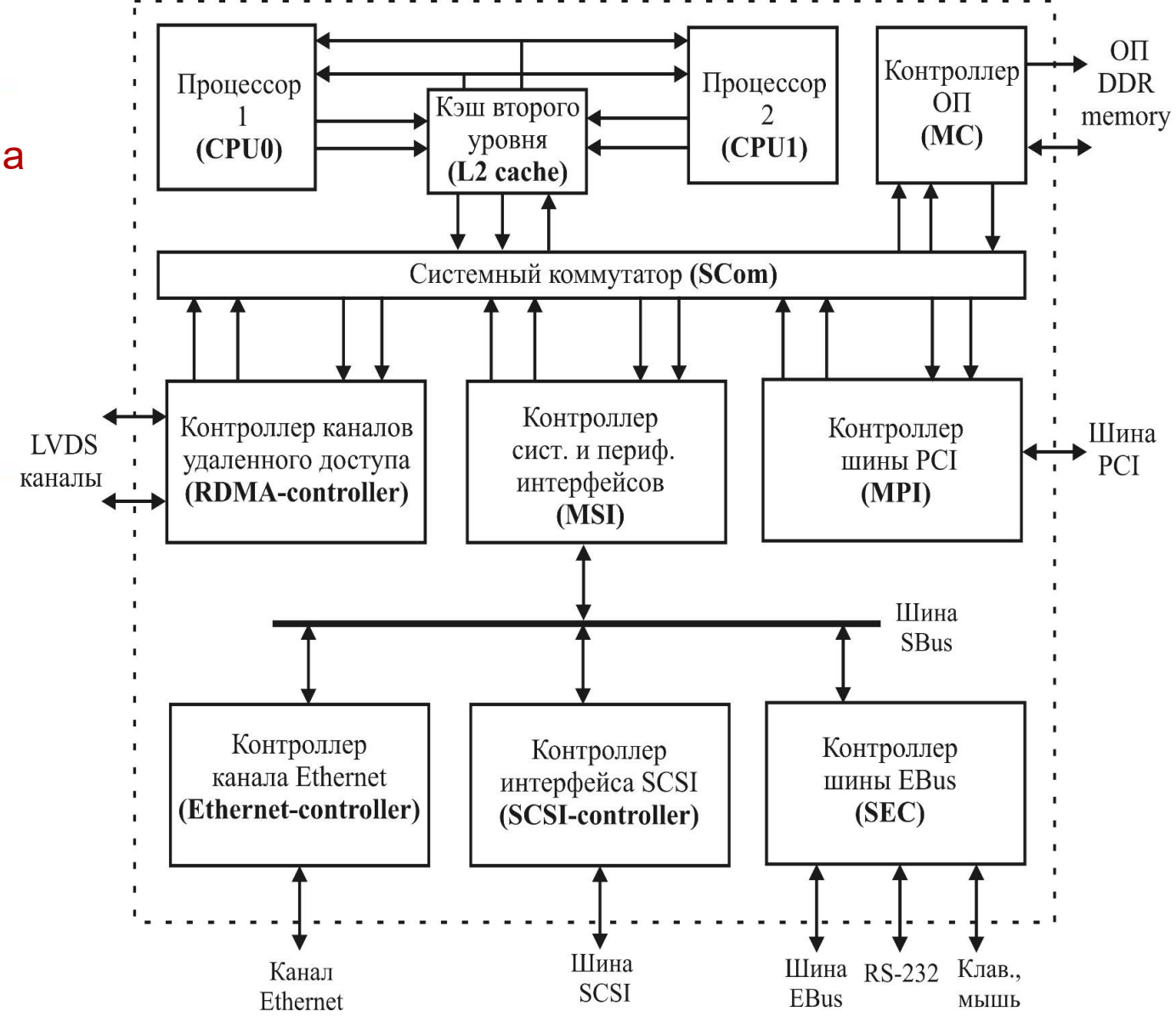


# Системы на кристалле.




СНК (**Soc** - System on chip) — это СБИС, интегрирующая на кристалле различные функциональные блоки, которые образуют законченное изделие для автономного применения в электронной аппаратуре.

Пример: Система на кристалле «МЦСТ-R500S»



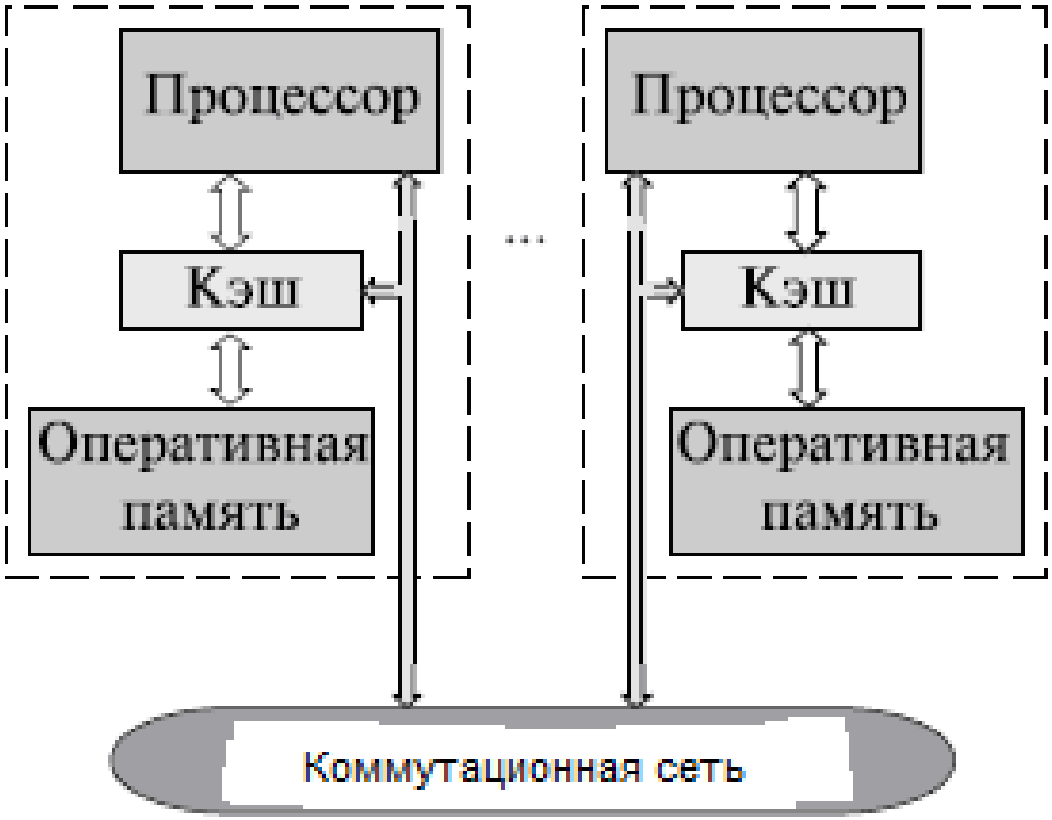
## **PVP (**параллельные векторные системы) ***система***

Основным признаком PVP-систем является **наличие специальных векторно-конвейерных процессоров**, в которых предусмотрены команды однотипной обработки векторов независимых данных.
Несколько таких процессоров (узел из 1-16) работают одновременно над общей памятью (**аналогично SMP**).
Несколько узлов могут быть объединены с помощью коммутатора (**аналогично *MPP*** - Массово-параллельные системы).

*Примеры*: линия векторно-конвейерных компьютеров CRAY



# Мультипроцессоры с неоднородным доступом к памяти (NUMA)



**NUMA** (Non Unified Memory Access) система – набор однородных базовых модулей, состоящих из небольшого числа процессоров и блока памяти.

- Модули объединены с помощью высокоскоростного коммутатора.
- Поддерживается единое адресное пространство, аппаратно поддерживается доступ к удаленной памяти.
- **Доступ к локальной памяти в несколько раз быстрее, чем к удаленной**.
- Масштабируемость NUMA-систем ограничивается
  - объемом адресного пространства,
  - возможностями аппаратуры поддержки когерентности кэшей
  - возможностями ОС по управлению большим числом процессоров.



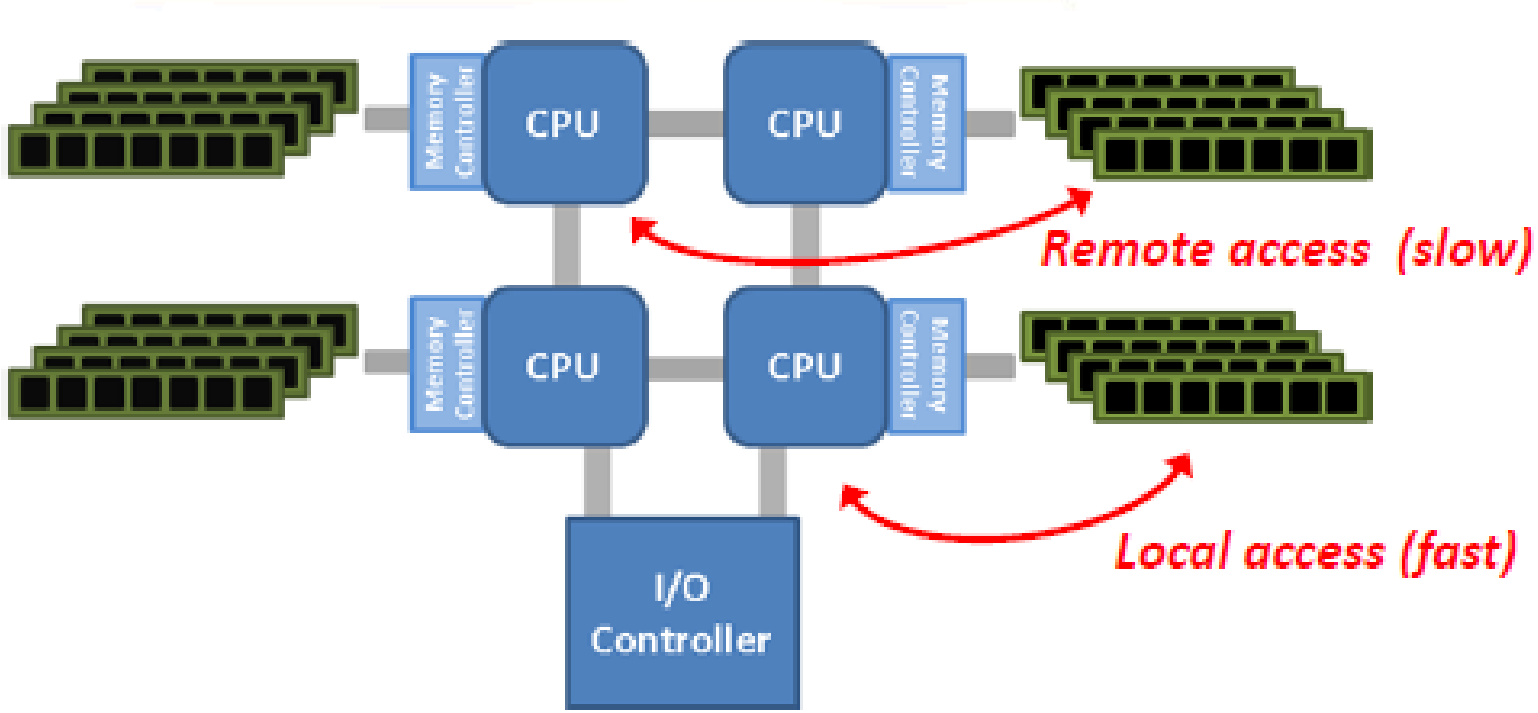
- NUMA-системы имеют один из наиболее **высоких показателей по масштабируемости** и, соответственно, по производительности Максимальное число процессоров в NUMA-системах составляет 1000 (Origin3000).

  Один из наиболее производительных суперкомпьютеров — Тега 10 — имеет производительностью 60 Тфлопс и состоит из 544 SMP-узлов, в каждом из которых находится от 8 до 16 процессоров Itanium 2

- Относительно **простое создание параллельных программ** (POSIX threads, OpenMP)



# Пример 4-х процессорной NUMA системы



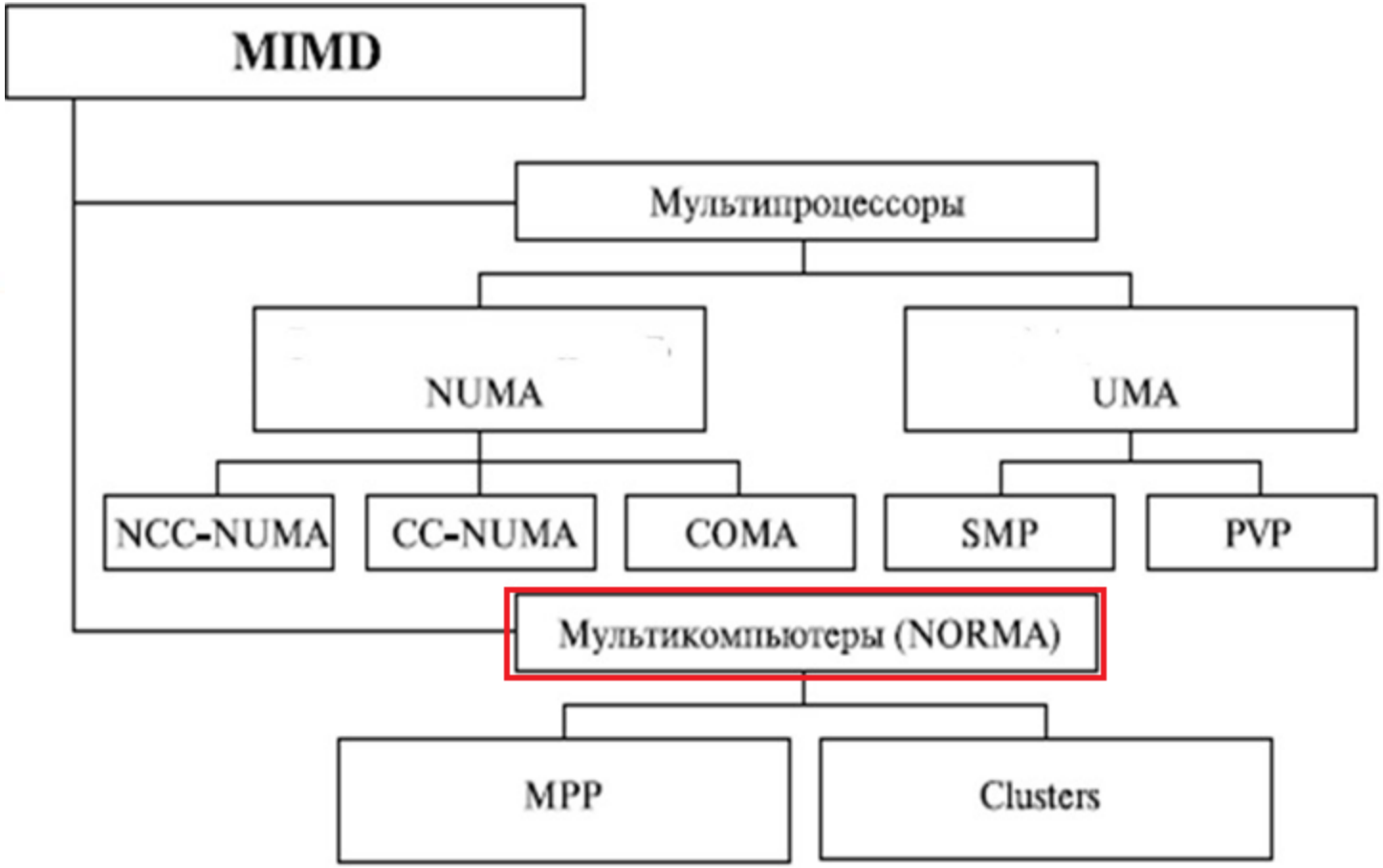

Использование распределенной общей памяти (*distributed shared memory* или DSM) упрощает проблемы создания мультипроцессоров, однако *проблемы эффективного использования распределенной памяти* (время доступа к локальной и удаленной памяти может различаться на несколько порядков) приводят к **существенному повышению сложности параллельного программирования**.

Среди **NUMA** (Non Unified Memory Access) систем выделяют:

- системы, в которых для представления данных используется только локальная кэш-память имеющихся процессоров (cache-only memory architecture или **COMA**); примеры: KSR-1 и DDM;
- *NUMA с когерентностью локальных кэшей разных процессоров* (cache-coherent NUMA или CC-NUMA); примеры: SGI Origin 2000, Sun HPC 10000, IBM/Sequent NUMA-Q 2000;
- *NUMA без поддержки на аппаратном уровне когерентности кэша* (non-cache coherent NUMA или **NCC-NUMA**); пример: Cray T3E.



# Мультикомпьютеры



- Не обеспечивают общего доступа ко всей имеющейся в системах памяти (no-remote memory access или **NORMA**)

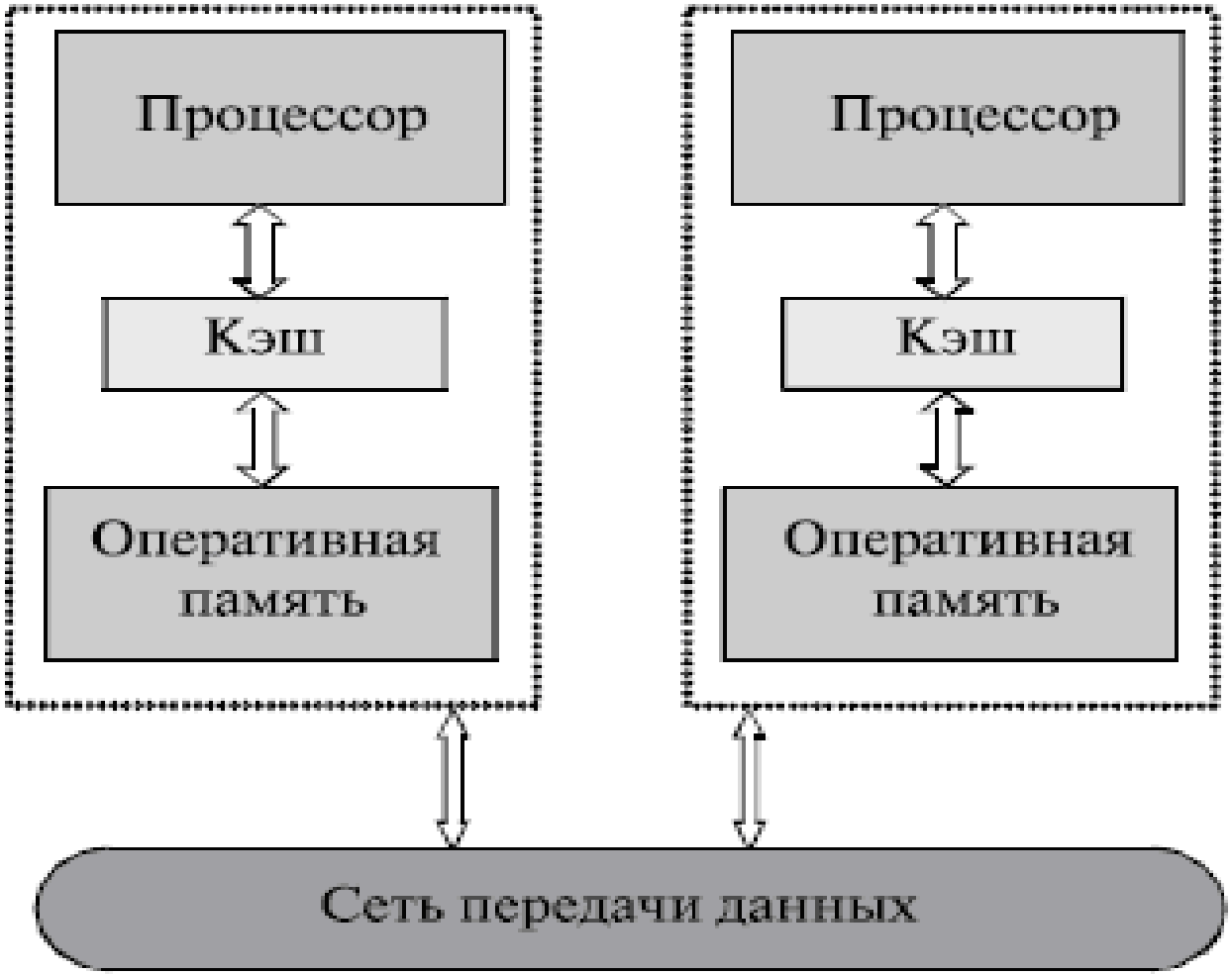
- **Принципиальное отличие с системами с распределенной общей памятью (мультипроцессоров):**
  каждый процессор системы может использовать только свою локальную память, в то время как для доступа к данным, располагаемым на других процессорах, необходимо явно выполнить операции передачи сообщений (message passing operations).

- Причина разработки -техническая сложность создания мультипроцессоров

- Этот подход применяется при построении 2-х типов мультикомпьютеров:
  - массово-параллельных систем (massively parallel processor или **MPP**)
  - кластеров (**clusters**). Связанный набор полноценных компьютеров, используемый в качестве единого ресурса.



# Мультикомпьютеры (MPP, clusters)



# Массово-параллельные системы (MPP)

- **Однородные** мультикомпьютеры с распределенной ОП
- Память физически разделена.
- Система строится из отдельных модулей, содержащих процессор, локальный банк ОП, коммуникационные процессоры или сетевые адаптеры, иногда – жесткие диски и/или другие устройства ввода/вывода.

- Используются **два варианта работы ОС** на машинах MPP-архитектуры.
  - полноценная ОС работает только на управляющей машине (front-end), на каждом отдельном модуле функционирует урезанный вариант ОС, обеспечивающий работу только расположенной в нем ветви параллельного приложения.
  - на каждом модуле работает полноценная UNIX-подобная ОС, устанавливаемая отдельно.

- Программирование производится в рамках **модели передачи сообщений (MPI)**
- В литературе иногда **к MPP системам относят как мультипроцессоры, так и мультикопьютеры**



**Достоинство** : хорошая масштабируемость

Общее число процессоров в реальных системах достигает нескольких тысяч. **Все рекорды по производительности на сегодня устанавливаются на машинах такой архитектуры**

**Недостатки** :

- *отсутствие* общей памяти заметно снижает скорость межпроцессорного обмена
- требуется *быстродействующее коммуникационное* оборудование, обеспечивающее среду передачи сообщений
- при создании программ необходимо учитывать топологию системы и специальным образом *распределять данные* между процессорами, чтобы минимизировать число пересылок и объем пересылаемых данных

Примеры: транспьютерные системы



# Кластерные системы

**Кластер** – это группа взаимно соединенных вычислительных систем (узлов), работающих совместно, составляя единый вычислительный ресурс и создавая иллюзию наличия единственной ВМ

- Используется в качестве **дешевого варианта MPP** компьютера (**связанный набор полноценных компьютеров, используемый в качестве единого ресурса**).
- Для связи узлов используется одна из *стандартных сетевых технологий* (Fast/Gigabit Ethernet)
- Программирование ведётся в рамках модели передачи сообщений (MPI).
- При объединении в кластер компьютеров разной мощности или разной архитектуры, говорят о **гетерогенных** (неоднородных) кластерах.



**Достоинства**

- при сбое узла другой узел берет нагрузку неисправного узла;
- масштабируемость;
- относительная дешевизна, т.к. собираются на базе стандартных комплектующих элементов;

**Недостатки:**

- организация взаимодействия вычислительных узлов кластера при помощи передачи сообщений обычно приводит к значительным временным задержкам.

# Основные области применения :

- **системы высокой надежности**

  Катастрофоустойчивые решения создаются на основе разнесения узлов МП системы на сотни километров и обеспечения механизмов глобальной синхронизации данных между такими узлами.

  В такой системе на аппаратном уровне фактически поддерживается **основной механизм повышения надежности — резервирование**. Причем узлы находятся в так называемом «горячем» резерве, и каждый из них в любой момент готов продолжить вычисления при выходе из строя какого-либо узла. При этом все приложения с отказавшего узла автоматически переносятся на другие машины комплекса.



## Основные области применения :

- **параллельные серверы баз данных**

- **системы для высокопроизводительных вычислений** предназначены для параллельных расчетов. Параллельная работа нескольких недорогих процессоров позволяет обеспечить одновременное проведение большого числа операций для научных расчетов.



# Классификации архитектур ВС

*http://www.parallel.ru*

***Классификация Флинна:***
единственность или множественность потоков данных и команд

**Дополнения Ванга и Бриггса к классификации Флинна**:
конкретизация классов SISD, SIMD, MIMD

**Классификация Фенга**:
две простые численные характеристики параллелизма (пословный и поразрядный параллелизм)

**Классификация Шора**:
шесть "типичных архитектур" ВС



**Классификация Хендлера**:
количественное описание параллелизма на трех различных уровнях обработки данных (выполнение программы, выполнение команд, обработка битов)

***Классификация Хокни:***
конкретизация класса MIMD

**Классификация Шнайдера**:
конкретизация класса SIMD (основная идея - выделение этапов выборки и непосредственно исполнения в потоках команд и данных)

**Классификация Джонсона:**
четыре класса MIMD-компьютеров (компьютеры с общей или распределенной памятью, программируемые с помощью передачи сообщений или разделяемых переменных)



**Классификация Базу:**
любую параллельную ВС можно однозначно описать последовательностью решений, принятых на этапе проектирования архитектуры.

**Классификация Кришнамарфи:**
четыре качественные характеристики параллелизма (степень гранулярности параллелизма, способ реализации, топология процессоров, способ управления процессорами)

**Классификация Скилликорна:**
описание архитектуры компьютера как абстрактной структуры, состоящей из компонент 4-х типов (процессор команд, процессор данных, иерархия памяти, коммутатор)

**Классификация Дазгупты**
построение схем архитектур из семи базовых понятий

**Классификация Дункана**



# Вопросы?