Лекция 3.

Классификация языков программирования

Для чего нужна классификация?

Классификация – система распределения объектов по классам в соответствии с определенным признаком.

Система классификации позволяет сгруппировать объекты и выделить определенные классы, которые будут характеризоваться рядом общих свойств.

Для чего нужна классификация?

Варианты классификации ЯП

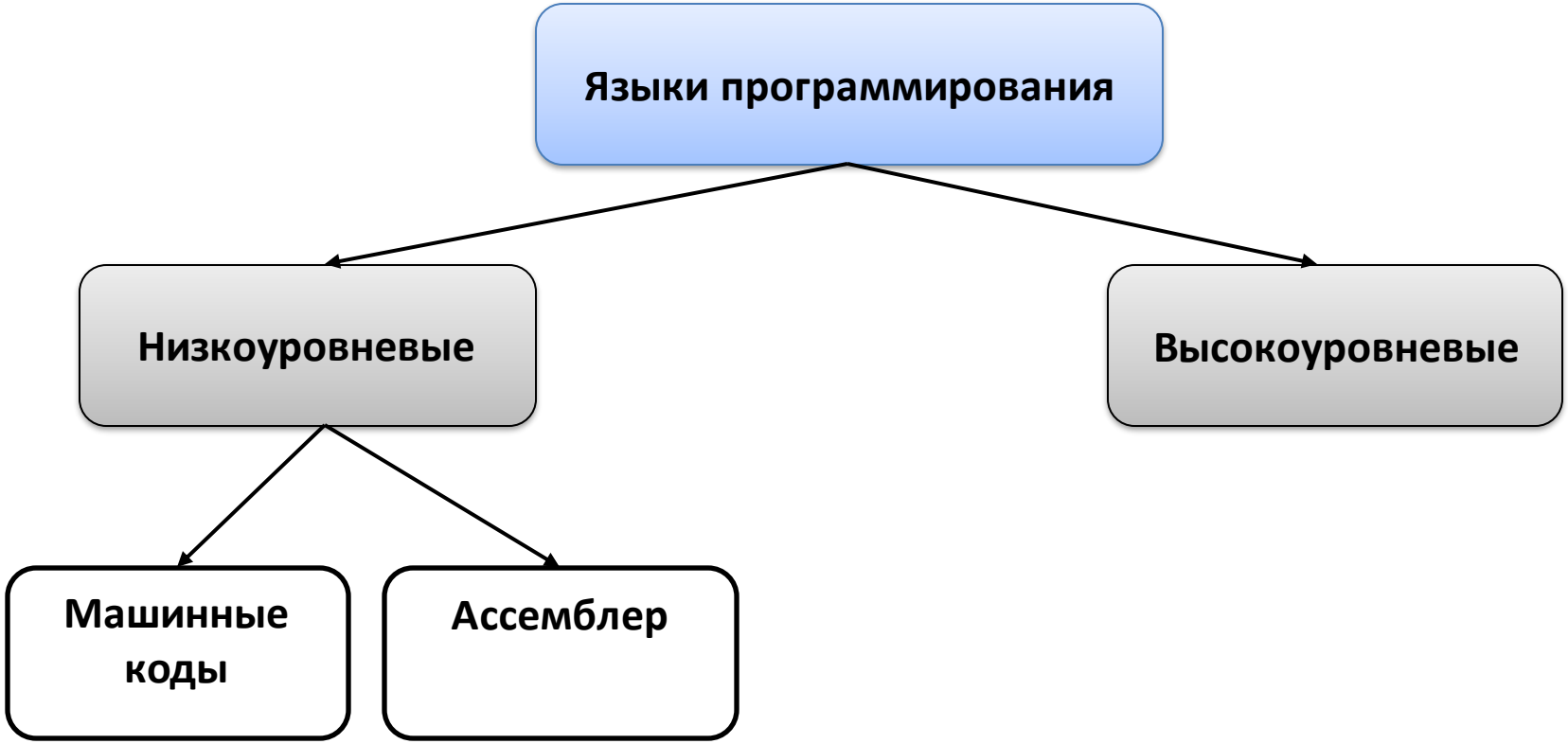
По каким признакам можно классифицировать?

Варианты классификации ЯП

По каким признакам можно классифицировать?

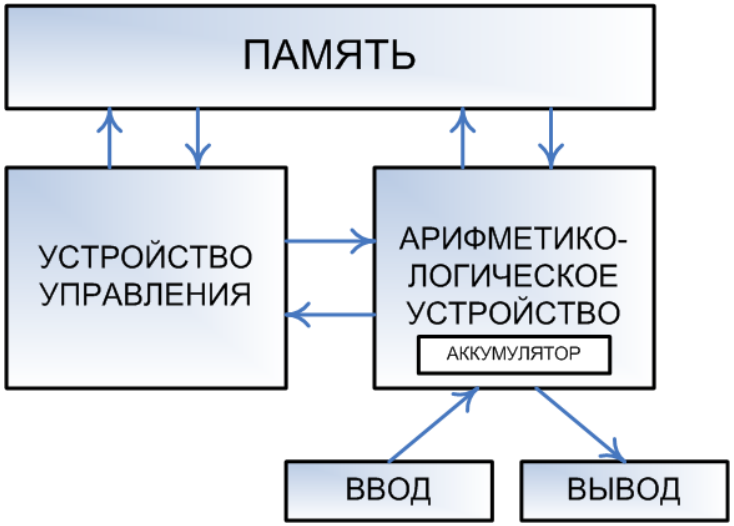
- Близость к аппаратуре
- Способ мышления
- Варианты разбиения задач на подзадачи
- Математический аппарат
- Система типов

Языки программирования – близость к аппаратуре

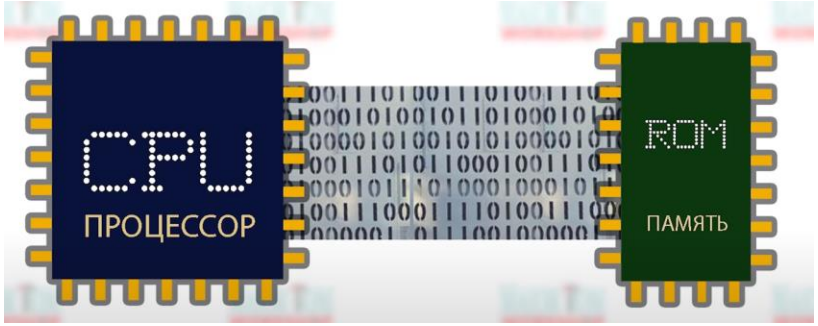


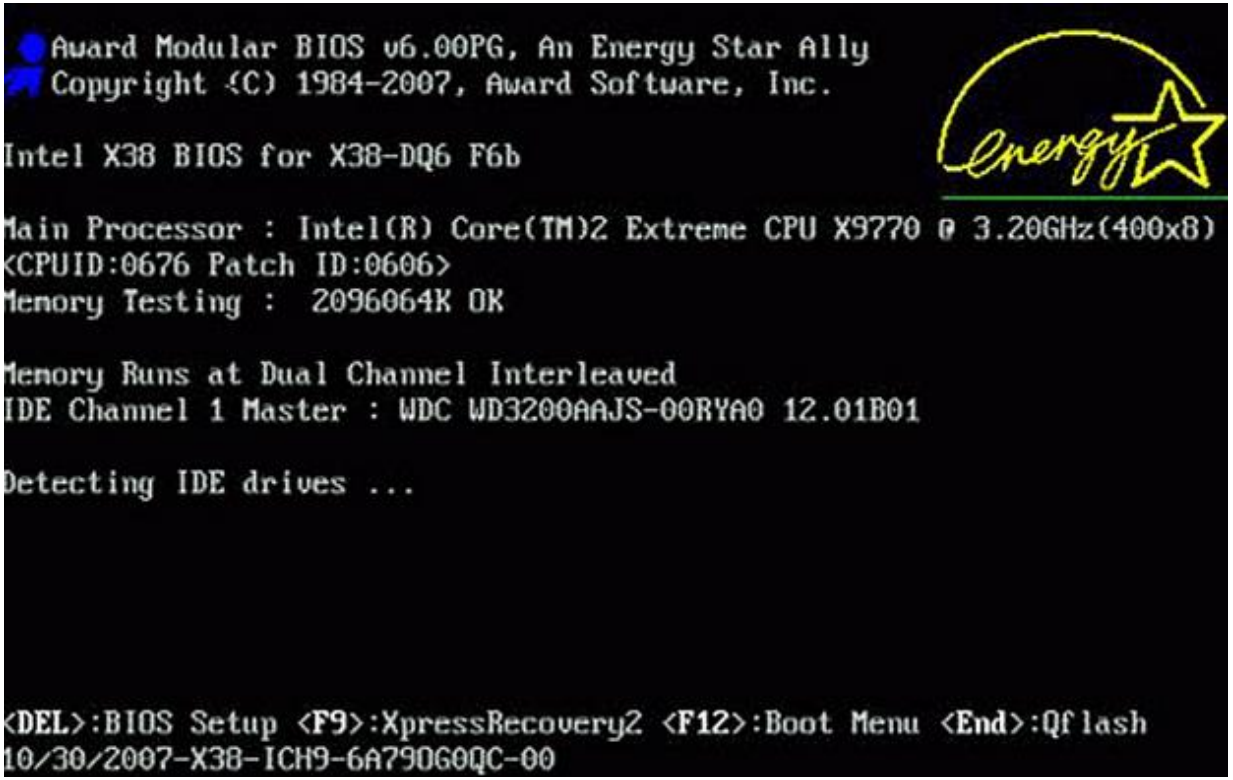
Классические принципы выполнения программ

- **Двоичное кодирование.** Вся информация кодируется с помощью двоичных сигналов (двоичных цифр, битов).
- **Однородность памяти.** Программы и данные хранятся в одной и той же памяти.
- **Адресуемость памяти.** Память состоит из пронумерованных ячеек; процессору в произвольный момент времени доступна любая ячейка.
- **Последовательного программное управления.** Программа состоит из набора команд, которые выполняются процессором автоматически друг за другом. Это **машинный код**.



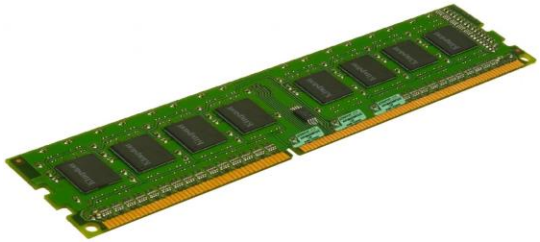
Классические принципы выполнения программ





Виды памяти

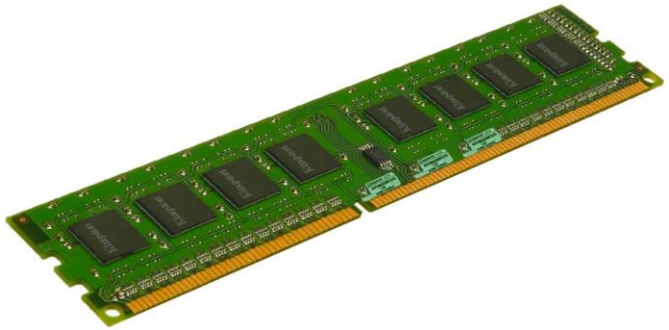
Внутренняя



Внешняя

Внутренняя память

ОЗУ



Временные данные и результаты вычислений

ПЗУ

Заранее записанные программы

Внутренняя память

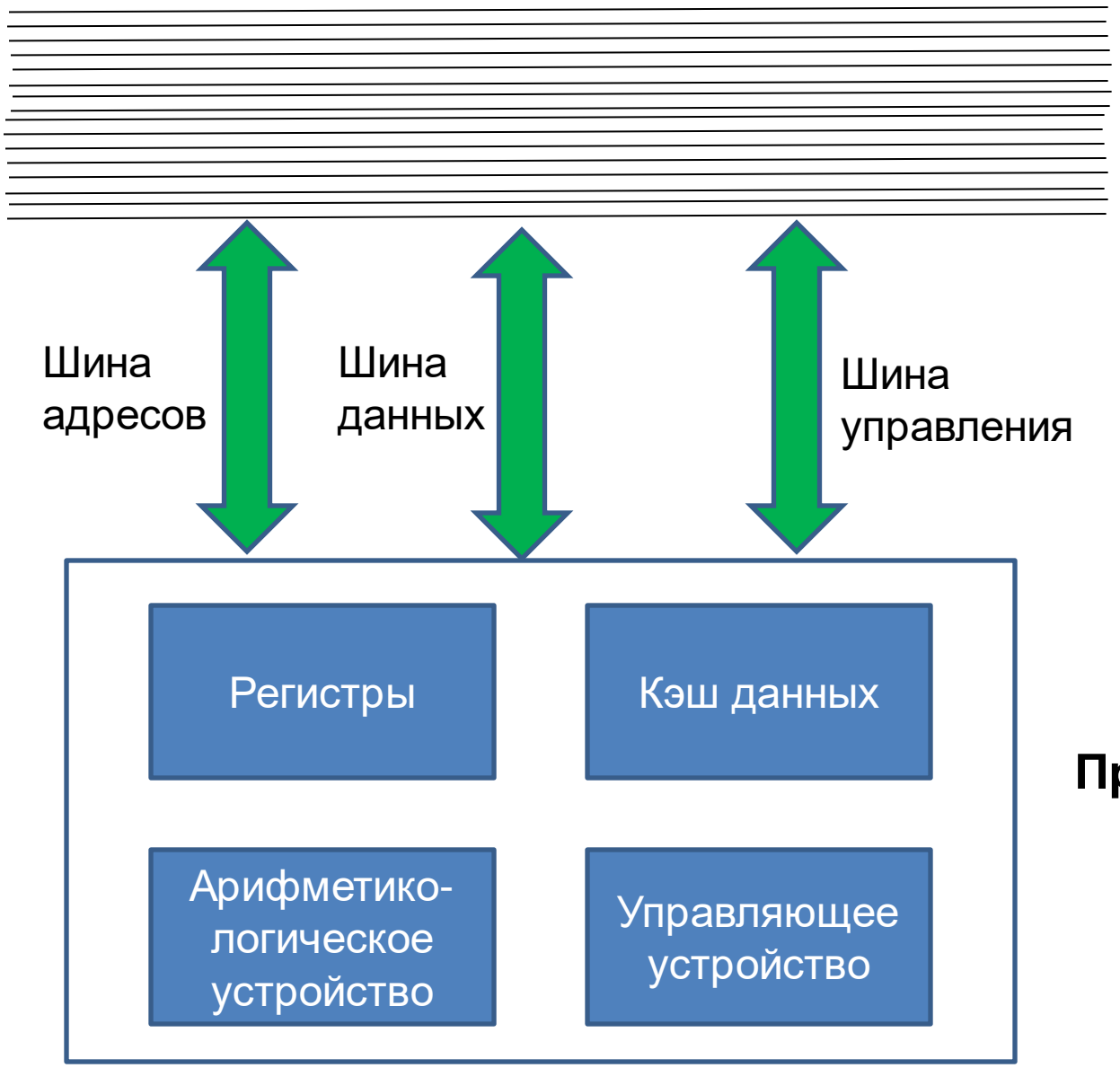
- Единицей памяти является **байт**.
- Память прямоадресуема (каждый байт имеет адрес).
- Процессор выбирает для обработки нужные данные, непосредственно адресуясь к последовательности байтов, содержащих эти данные.

0	1	2	3	4	5	6	7	8	. . .
11	123	35	5	65	77	88	48	253	. . .

Внешняя память

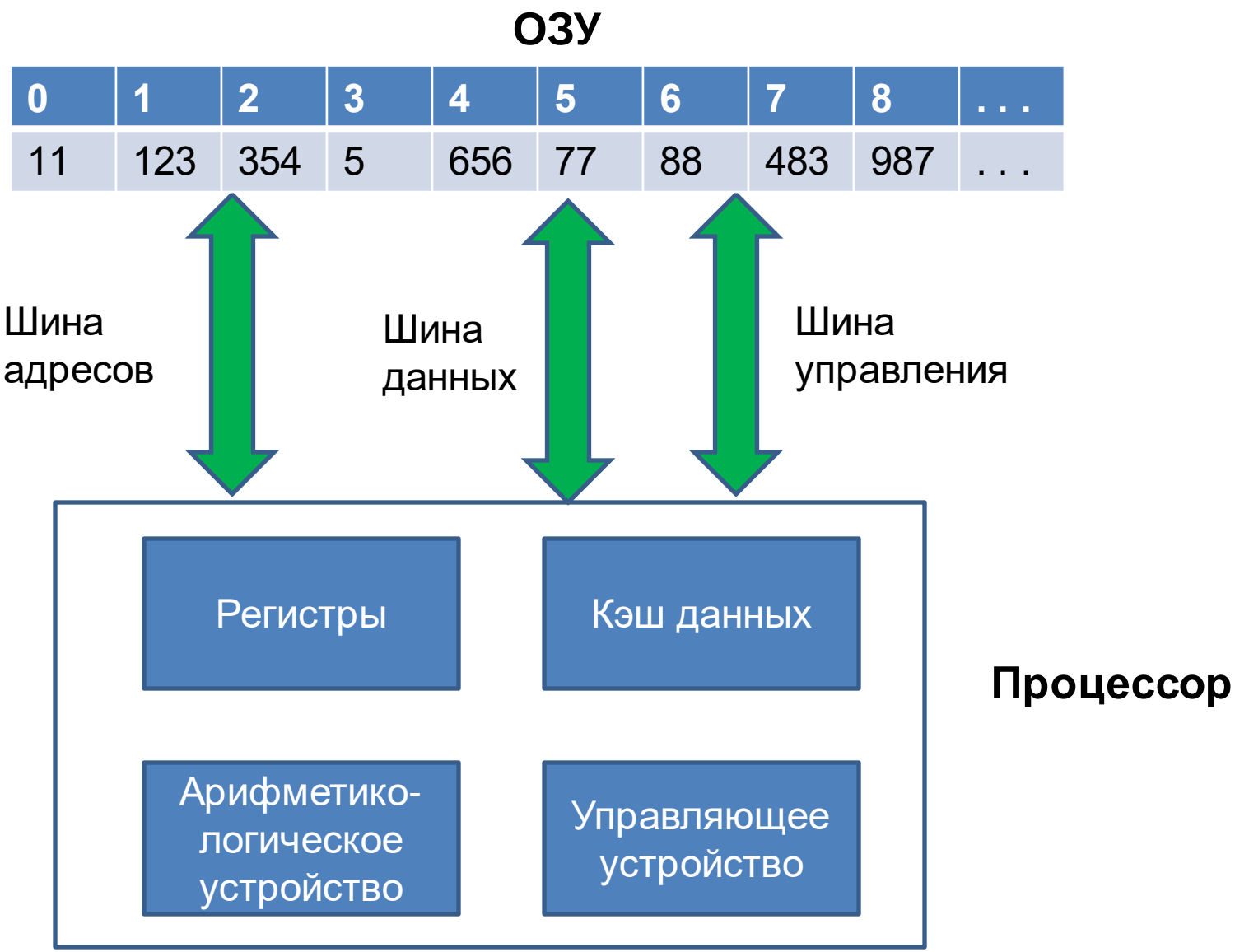
- Минимальная адресуемая единица – **физическая запись**.
- Для обработки (выделения полей) запись должна быть считана в оперативную память (ОП).
- Записи из внешней памяти в ОП читаются намного медленнее, чем обрабатываются процессором в ОП.

Устройство процессора



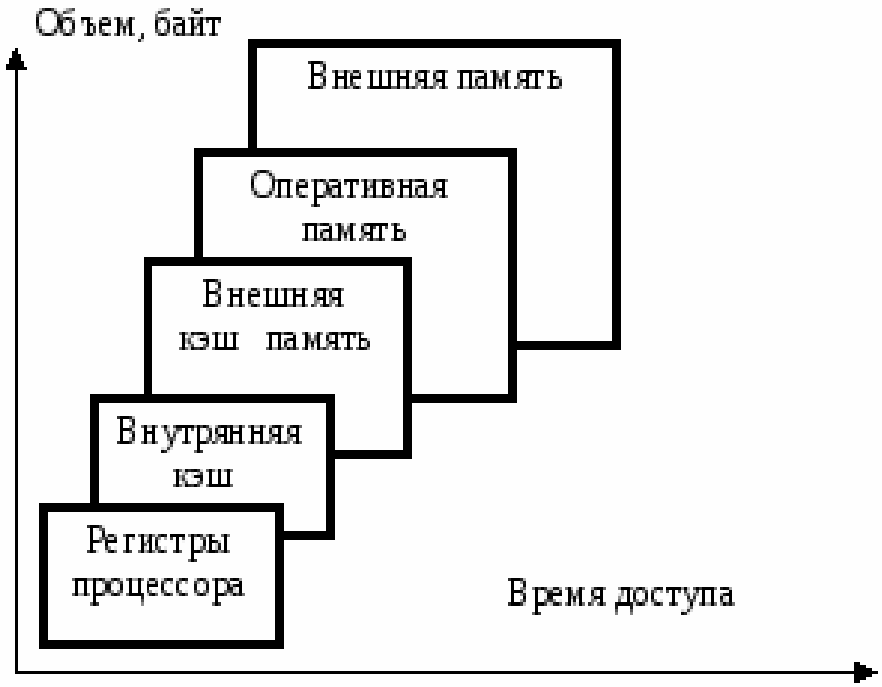
Процессор

Взаимодействие процессора и ОЗУ



Неоднородность памяти

- **Регистры процессора**. Сотни байт, скорость сравнима со скоростью процессора.
- **Оперативная память**. Гигабайты, скорость меньше.
- **Внешняя память**. Терабайты, самая медленная.



Неоднородность памяти

Тип кэша	Время доступа (тактов)	Размер кэша
Регистры	0	десятки штук
L1 кэш	4	32 KB
L2 кэш	10	256 KB
L3 кэш	50	8 MB
Оперативная память	200	8 GB
Буфер диска	100'000	64 MB
Локальный диск	10'000'000	1000 GB

Неоднородность памяти

```
int matrixsum1(int size, int M[][size])
{
    int sum = 0;

    for (int i = 0; i < size; i++) {
        for (int j = 0; j < size; j++) {
            sum += M[i][j];    // обходим построчно
        }
    }
    return sum;
}

int matrixsum2(int size, int M[][size])
{
    int sum = 0;

    for (int i = 0; i < size; i++) {
        for (int j = 0; j < size; j++) {
            sum += M[j][i];    // обходим по столбцам
        }
    }
    return sum;
}
```

Две функции для суммирования элементов матрицы.

Одинаково ли время выполнения этих функций?

Неоднородность памяти

```
int matrixsum1(int size, int M[][size])
{
    int sum = 0;

    for (int i = 0; i < size; i++) {
        for (int j = 0; j < size; j++) {
            sum += M[i][j];    // обходим построчно
        }
    }
    return sum;
}

int matrixsum2(int size, int M[][size])
{
    int sum = 0;

    for (int i = 0; i < size; i++) {
        for (int j = 0; j < size; j++) {
            sum += M[j][i];    // обходим по столбцам
        }
    }
    return sum;
}
```

На больших матрицах вторая функция может работать в 20 раз медленнее

Регистры процессора

Регистры общего назначения: **AX, BX, CX, DX**
Для размещения обрабатываемых данные

Служебные регистры: **SP, BP, IP, SI, FLAGS**
IP (Instruction Pointer) – указатель текущей инструкции

<table>
<tr><td colspan="3">RAX</td><td>64 бита</td></tr>
<tr><td></td><td colspan="2">EAX</td><td>32 бита</td></tr>
<tr><td></td><td></td><td>AX</td><td>16 бит</td></tr>
<tr><td></td><td></td><td>AH AL</td><td>8 бит</td></tr>
</table>

AL, AH, AX, EAX, RAX – самостоятельные регистры

У каждого регистра свой **числовой код**.

Машинные команды

Процессоры могут выполнять команды трех типов:

1. **Вычисления.** Сложение вычитание, умножение, деление целых и вещественных чисел, операции сравнения, побитовые операции. Операнды обычно находятся в регистрах.
2. **Загрузка и хранение.** Передача данных из оперативной памяти в регистры и обратно, обмен данными с другими устройствами.
3. **Управление.** Переключение выполнения к другой точке программного. Условное или безусловное.

Структура машинных команд

КОП	Адрес 1-го операнда	Адрес 2-го операнда	Адрес результата	Адрес след. команды

а

КОП	Адрес 1-го операнда	Адрес 2-го операнда	Адрес результата

б

КОП	Адрес 1-го операнда

г

КОП	Адрес 1-го операнда	Адрес 2-го операнда

в

КОП

д

КОП – код операции

Структуры команд: а – четырехадресная; б – трехадресная; в – двухадресная; г – одноадресная; д – безадресная

Пример машинной команды (32 бита)

0						6					11					16					21	22									31	Позиция бита
0	1	1	1	1	1	0	0	1	0	1	0	0	0	1	1	0	0	1	0	0	0	0	1	0	0	0	0	1	0	1	0	Значение бита
Код команды						Результат					Операнд 1					Операнд 2						Вторичный код (266)										Роль

- **Код команды** - комбинация первичного кода 31 (11111) и вторичного кода 266 (100001010).
- **Операнды** читаются из регистров 3 (011) и 4 (100).
- **Результат** помещается в регистр 5 (101).

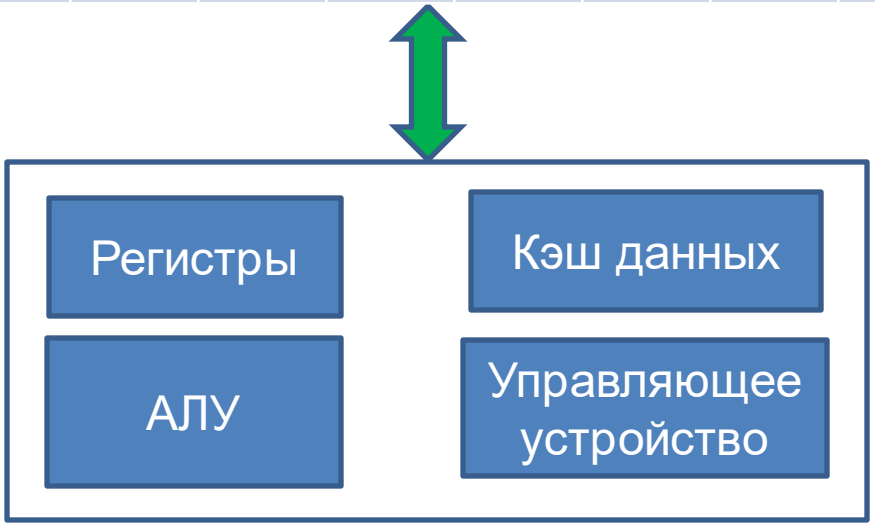
Цикл обработки команд

1. Извлечь код очередной машинной команды из памяти, начиная с ячейки, адрес которой находится в ***указателе инструкции*** (регистр IP, EIP или RIP).

2. Увеличить значение указателя инструкции на длину извлеченного кода. Регистр будет содержать адрес инструкции, ***следующей за текущей***.

3. Декодировать извлеченный из памяти код команды и выполнить соответствующее этому коду действие.

0	1	2	3	4	5	6	7	8	. . .
11	123	35	5	65	77	88	48	253	. . .



Передача управления

Для организации **ветвлений**, **циклов** и **вызовов подпрограмм** применяются команды, принудительно изменяющие содержимое указателя инструкции. Выполнение программы продолжается с нового места.

Условная передача управления - сначала проверяется выполнение условия. Регистр FLAGS.

Переход с запоминанием адреса возврата в аппаратном стеке - для вызова подпрограмм.

Язык ассемблера

1952 год. Первый компилятор А-0

Грейс Хоппер (Grace Hopper, 1906-1992)

Американская ученая и контр-адмирал флота США

Язык ассемблера

Предоставляет базовый уровень абстракции:

- **Переменные** и **метки** с символьными именами (в машинном коде на их местах стоят адреса в памяти).
- Независимость **мнемокода команды** от её операндов.

Язык ассемблера

Предоставляет базовый уровень абстракции:

- **Переменные** и **метки** с символьными именами (в машинном коде на их местах стоят адреса в памяти).
- Независимость **мнемокода команды** от её операндов.

“Hello world” (TASM для MS DOS)

```
.MODEL TINY
CODE SEGMENT
ASSUME CS:CODE, DS:CODE
ORG 100h
START:
        mov ah,9
        mov dx,OFFSET Msg
        int 21h
        int 20h
        Msg DB 'Hello World',13,10,'$'
CODE ENDS
END START
```

Ассемблер – низкоуровневый язык программирования

Программа на ассемблере - прообраз содержимого оперативной памяти.

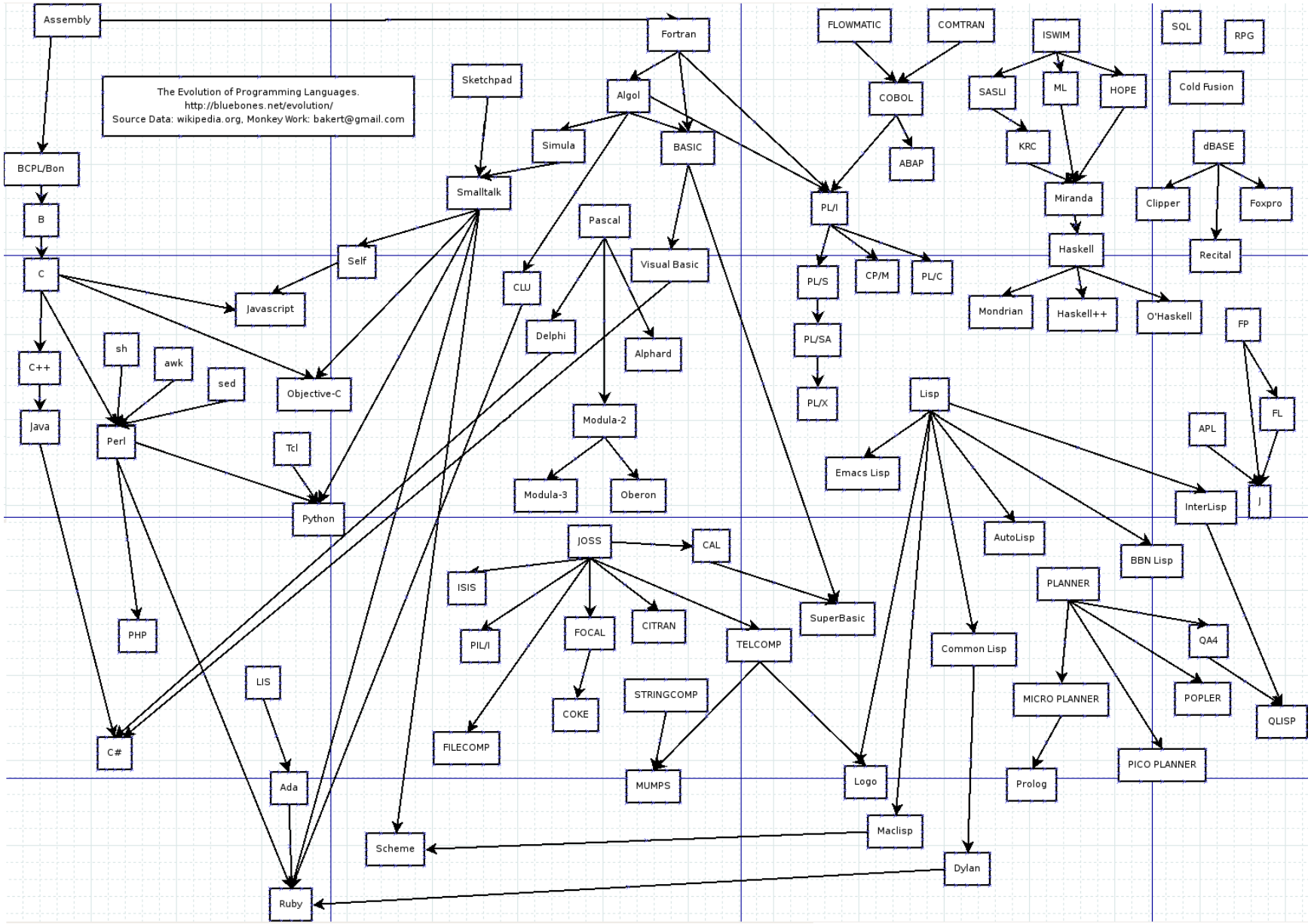
- **Самый быстрый и компактный машинный код**, какой вообще возможен для данного процессора.
- **Непосредственный доступ к аппаратуре** (порты ввода-вывода, регистры процессора, системные области диска).
- **Дизассемблер** – можно исследовать существующие программы при отсутствии исходного кода.

Ассемблер – низкоуровневый язык программирования

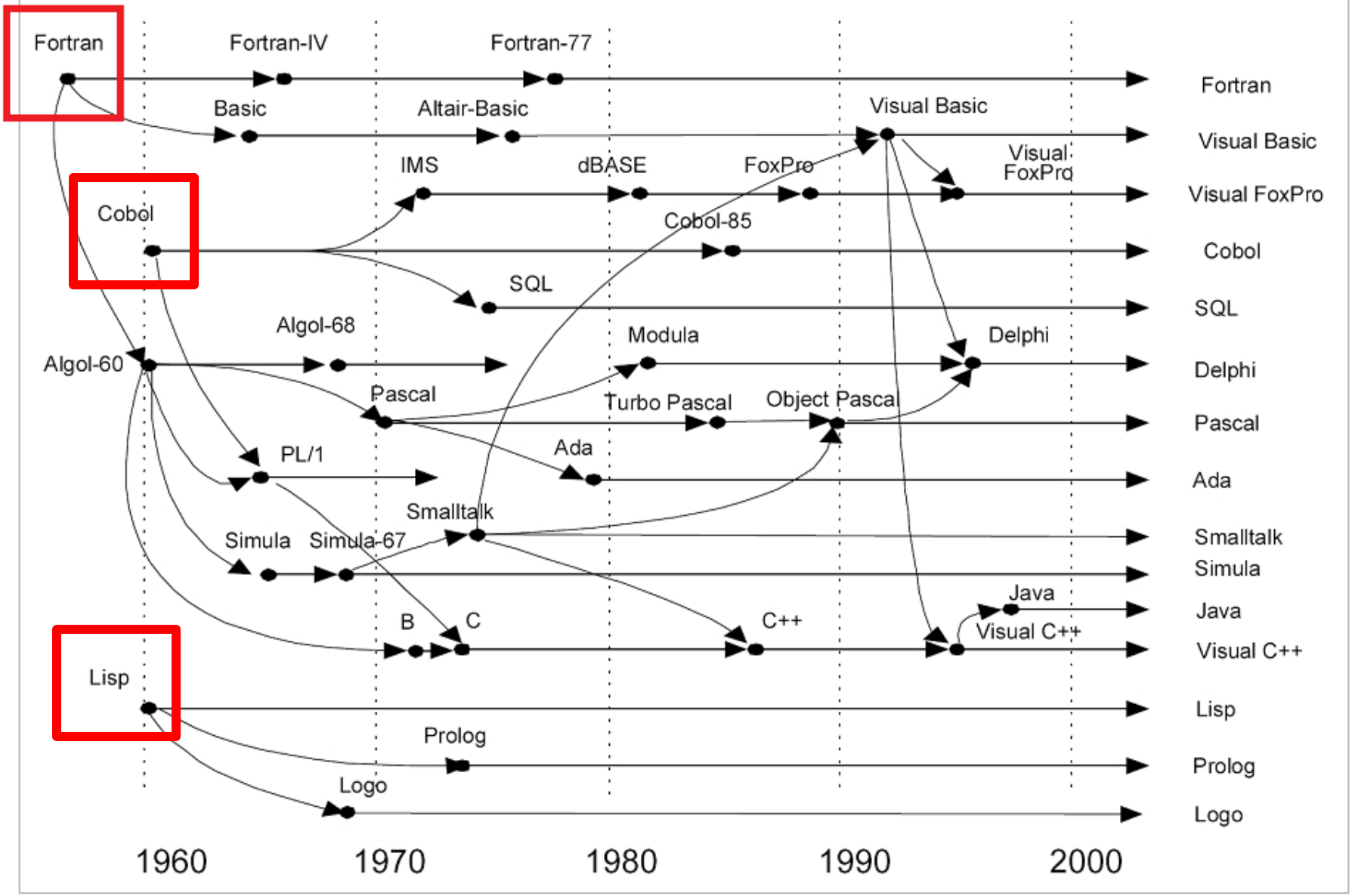
Применяют, когда критически важны:

- **Быстродействие** (драйверы, игры)
- **Объем используемой памяти** (загрузочные секторы, встраиваемое ПО для микроконтроллеров, вирусы, программные защиты).

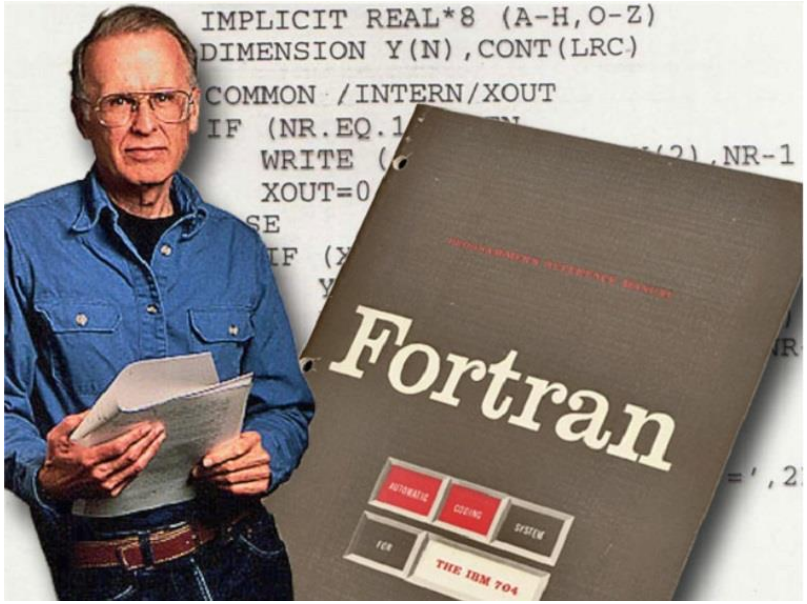
Высокоуровневые языки программирования



ГЕНЕАЛОГИЧЕСКОЕ ДЕРЕВО ЯЗЫКОВ ПРОГРАММИРОВАНИЯ



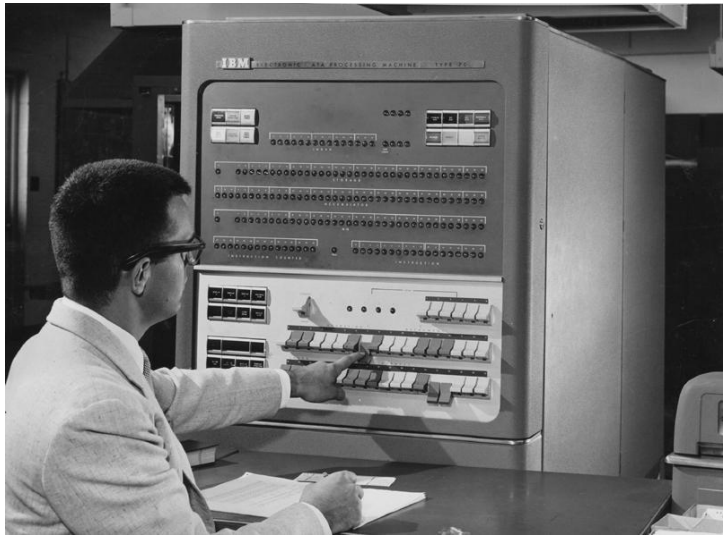
Fortran – первый высокоуровневый ЯП



1954 год

Джон Бэкус (John Backus)
1924-2007

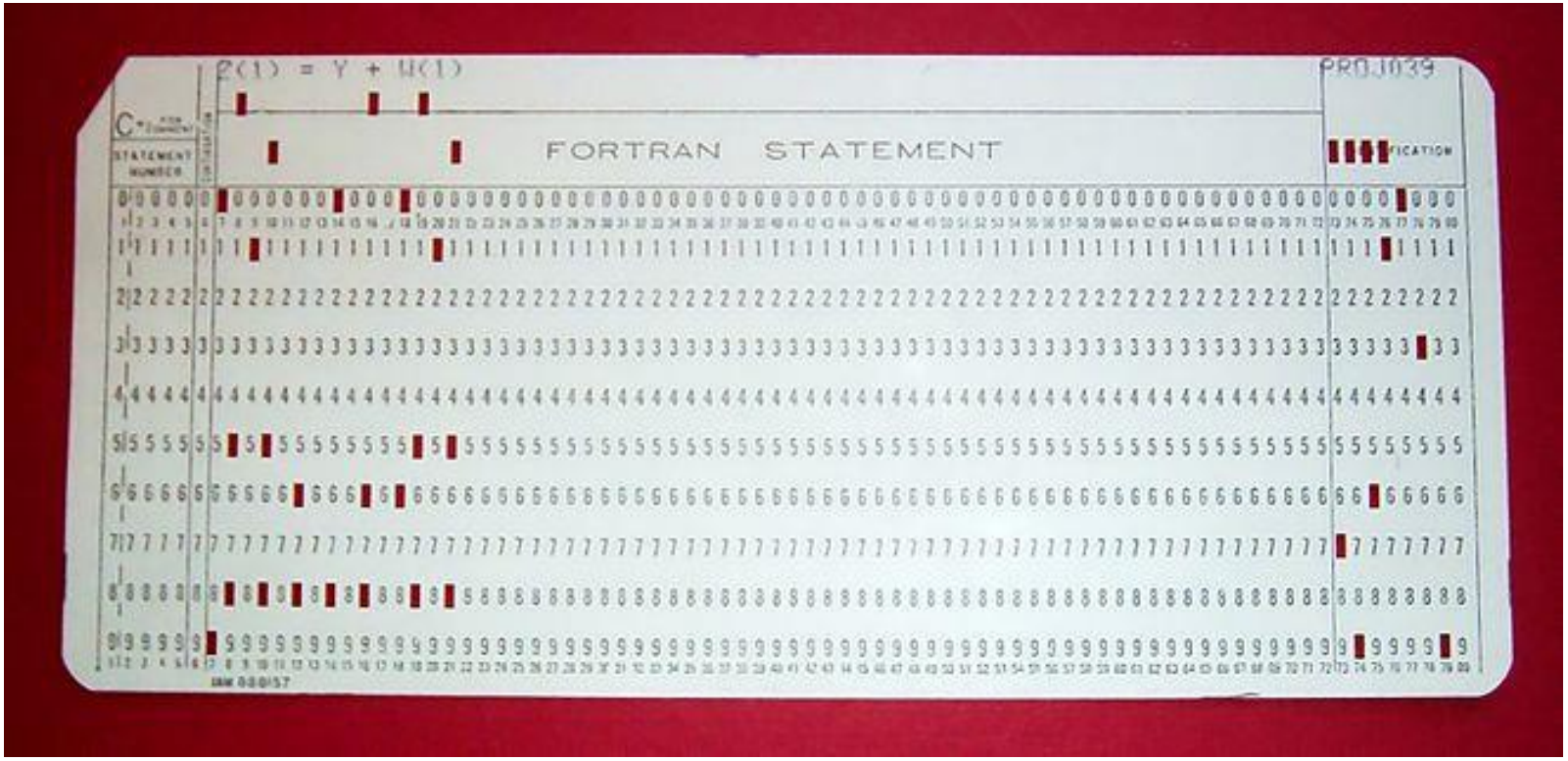
Разрабатывался в фирме IBM
для компьютера IBM 704



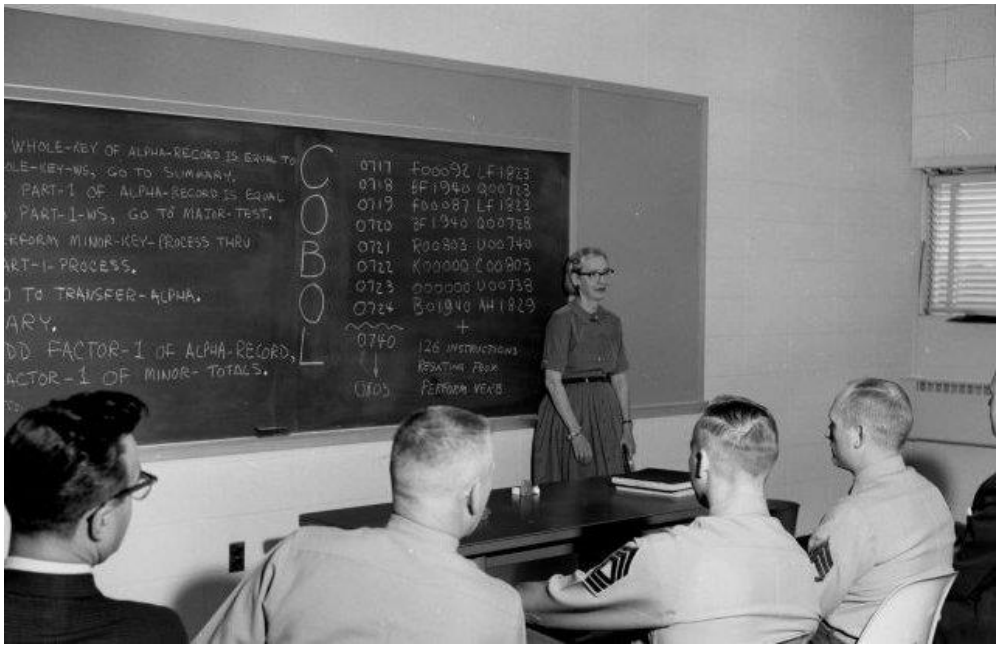
Formula **Tran**slation

- Оператор присваивания
- Массивы
- Оператор цикла DO

Fortran – первый высокоуровневый ЯП



COBOL – первый язык для бизнес-приложений



1959 год

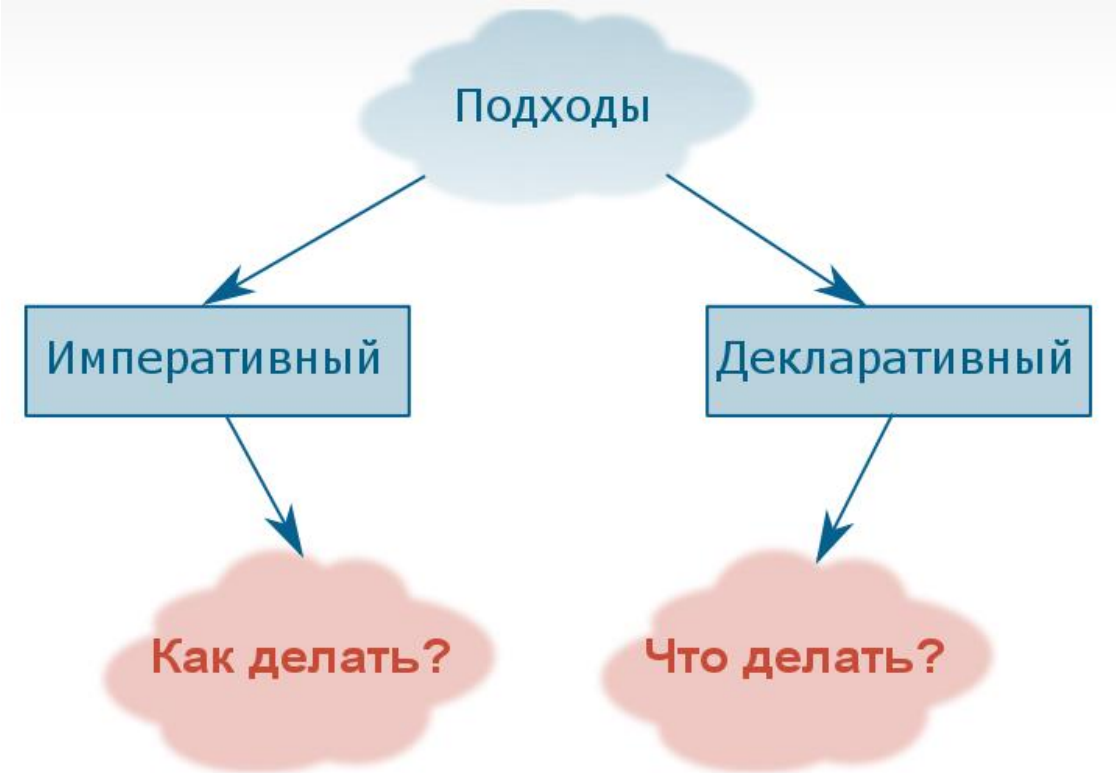
Грейс Хоппер – руководитель проекта (язык Flow-Matic)

Common **B**usines
Oriented **L**anguage

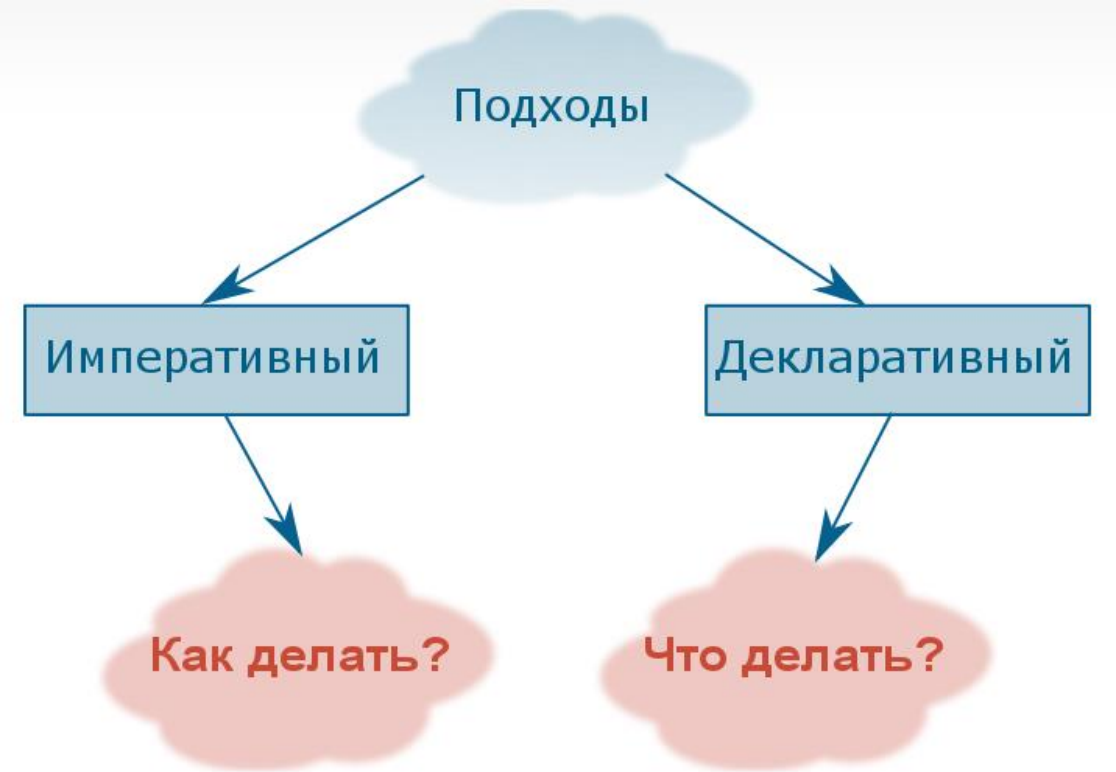
- Структуры данных (записи)
- Файлы

```
PERFORM UNTIL EndOfStudentFile
        ADD 1 TO MonthCount(MOBirth)
        READ StudentFile
           AT END SET EndOfStudentFile TO TRUE
        END-READ
    END-PERFORM
```

Языки программирования – разное мышление



Языки программирования – разное мышление

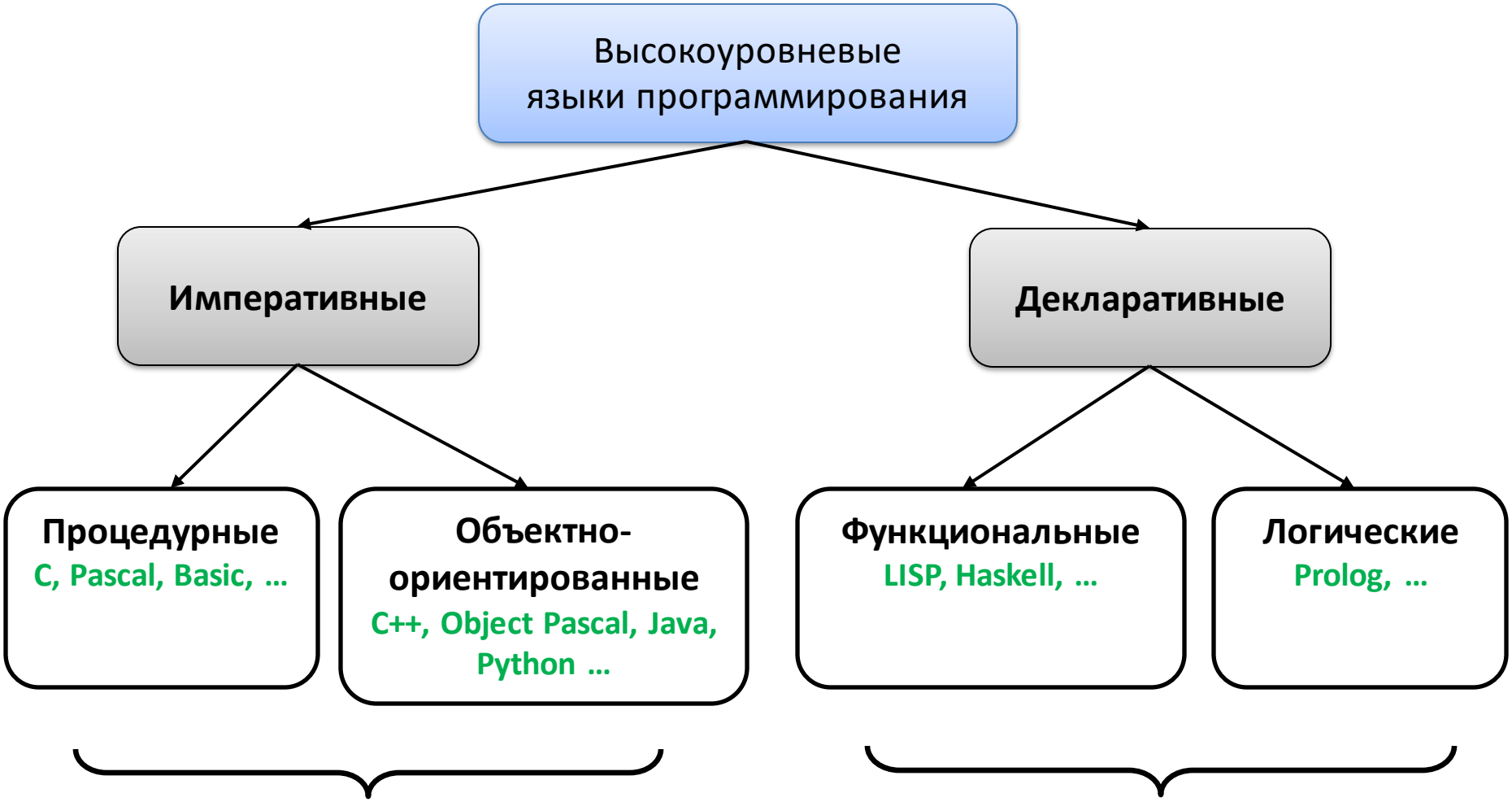



```
USE Customers
GO TOP
DO WHILE .NOT. EOF()
   IF Customers.LastName = “Иванов” …
   SKIP
ENDDO
```

dBase

```
Select * from Customers
Where LastName = “Иванов”
```

SQL



Способы реализации структуры, т.е. разбиения задачи на подзадачи

Математический аппарат: функции или формальная логика

Императивные языки программирования

Программа – последовательность команд, определяющих алгоритм решения задачи.

- Основная идея – использование памяти для хранения данных в виде переменных.
- Основная команда – присвоение. В процессе вычислений меняется значение переменных.
- Программный код логически отделяется от данных.

Императивные языки программирования

Программа – последовательность команд, определяющих алгоритм решения задачи.

- Основная идея – использование памяти для хранения данных в виде переменных.
- Основная команда – присвоение. В процессе вычислений меняется значение переменных.
- Программный код логически отделяется от данных.

Процедурное программирование – отражение фон Неймановской архитектуры компьютера.

+ Понятно человеку, несложно перевести в машинные коды, работает быстро.

- Большой объем кода.

Функциональный язык программирования LISP

1958 год

Джон Маккарти (John McCarthy) 1922-2011

Язык для поддержки задач искусственного интеллекта - машинной обработки "символических выражений".

- Условные конструкции
- Рекурсия
- Переменные как указатели
- Сборка мусора
- Макросы для расширения синтаксиса языка

Функциональные языки программирования

Программа – последовательность описания функций и выражений.

- Выражения записываются в виде списков (Lisp = List Processing).
- Вместо циклов – рекурсия.
- Данные и программный код представляются одинаково. Параметром функции может быть другая функция.

Функциональные языки программирования

Программа – последовательность описания функций и выражений.

- Выражения записываются в виде списков (Lisp = List Processing).
- Вместо циклов – рекурсия.
- Данные и программный код представляются одинаково. Параметром функции может быть другая функция.

+ Алгоритмы записываются в компактном виде. Увеличение производительности программиста.

- Сложно для понимания, сложнее перевести в машинный код, работает медленнее .

Императивный и декларативный подходы

Пример. Вычисление суммы целых чисел от 1 до 10.

Императивный и декларативный подходы

Пример. Вычисление суммы целых чисел от 1 до 10.

C:

```
int total = 0;
for (i=1; i<=10; ++i)
  total += i;
```

Императивный и декларативный подходы

Пример. Вычисление суммы целых чисел от 1 до 10.

C:

```
int total = 0;
for (i=1; i<=10; ++i)
  total += i;
```

Язык Scheme (Lisp):

```
(define (sum from total)
   (if (= 0 from)
      total
      (sum (- from 1) (+ total from))))

( sum 10 0 )
```

Императивный и декларативный подходы

Пример. Вычисление суммы целых чисел от 1 до 10.

C:

```
int total = 0;
for (i=1; i<=10; ++i)
  total += i;
```

Haskell: sum [1..10]

Язык Scheme (Lisp):

```
(define (sum from total)
   (if (= 0 from)
      total
      (sum (- from 1) (+ total from))))

( sum 10 0 )
```

Язык PowerShell:

```
$sum=0
1..10 | ForEach-Object {$sum+=$_}

1..10 | Measure-Object -Sum
```

```
qsort( a, lo, hi ) int a[], hi, lo;
{
  int h, l, p, t;

  if (lo < hi) {
    l = lo;
    h = hi;
    p = a[hi];

    do {
      while ((l < h) && (a[l] <= p))
          l = l+1;
      while ((h > l) && (a[h] >= p))
          h = h-1;
      if (l < h) {
          t = a[l];
          a[l] = a[h];
          a[h] = t;
      }
    } while (l < h);

    t = a[l];
    a[l] = a[hi];
    a[hi] = t;

    qsort( a, lo, l-1 );
    qsort( a, l+1, hi );
  }
}
```

Алгоритм быстрой сортировки на языке С

Алгоритм быстрой сортировки на языке Haskell

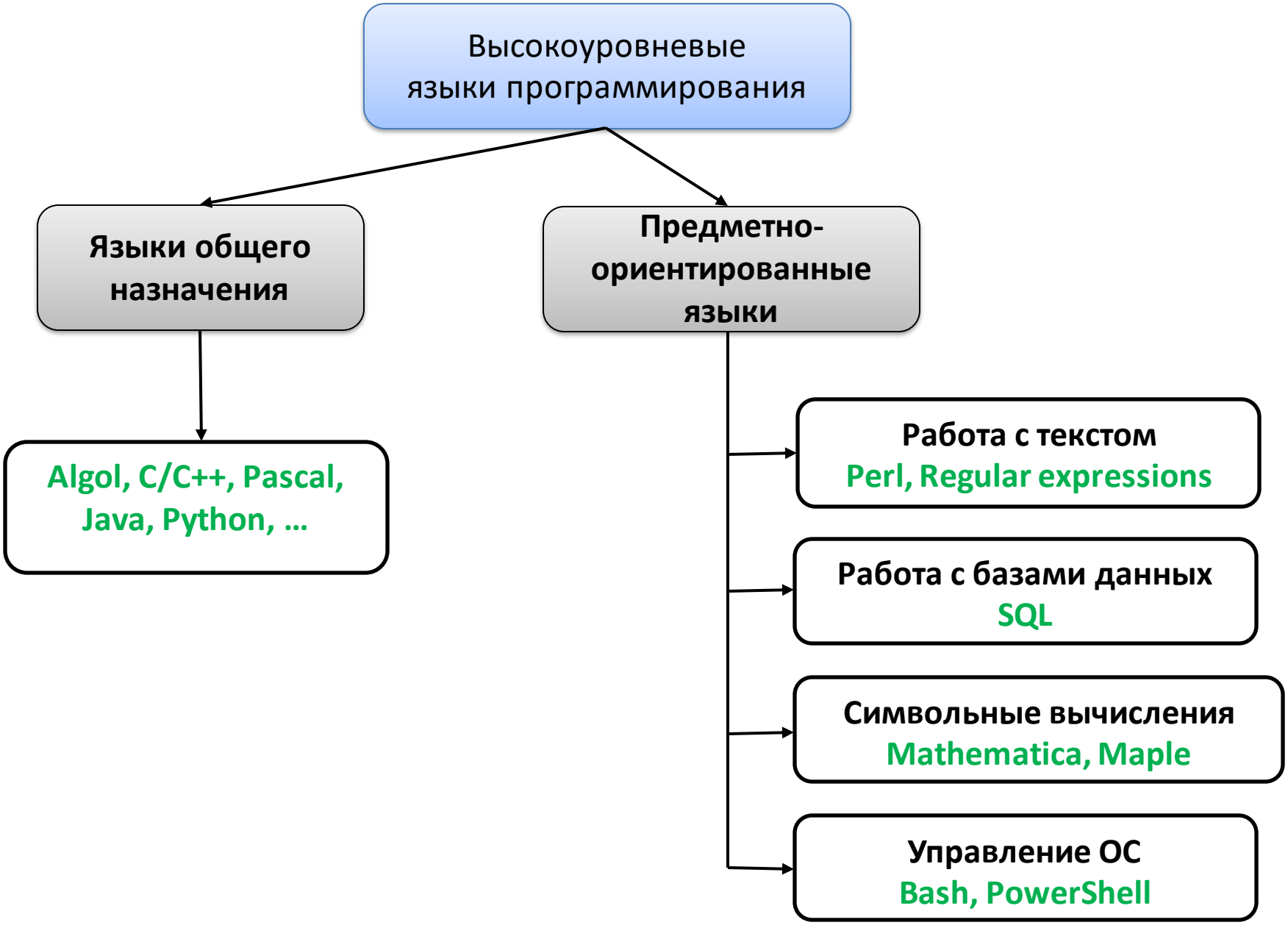
```
qsort []     = []
qsort (x:xs) = qsort elts_lt_x ++ [x] ++ qsort elts_greq_x
                 where
                   elts_lt_x   = [y | y <- xs, y < x]
                   elts_greq_x = [y | y <- xs, y >= x]
```

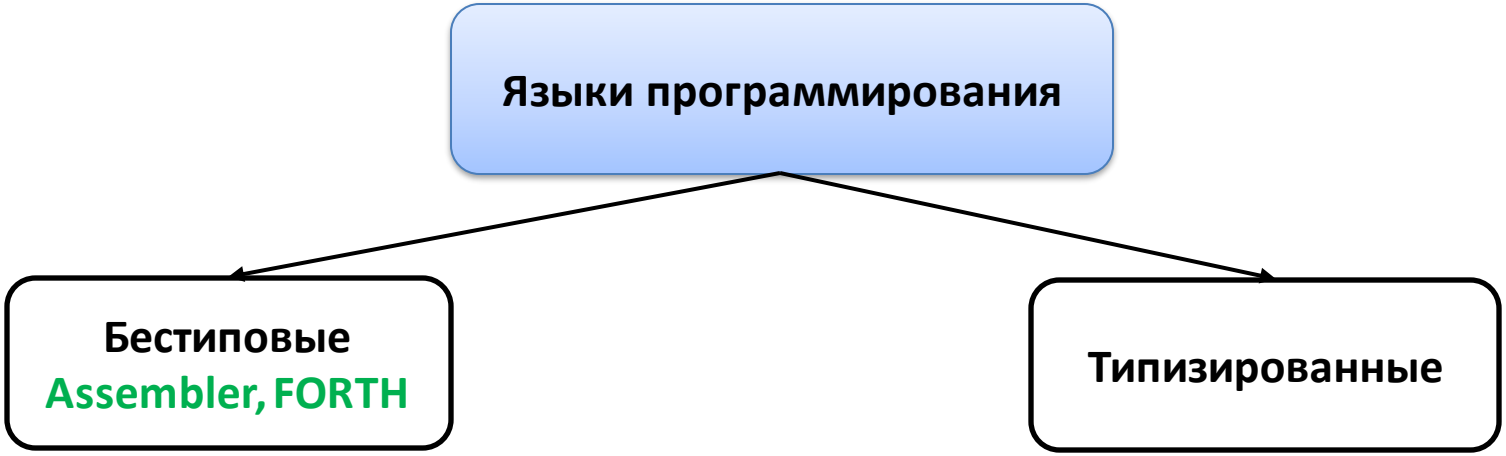
Алгоритм быстрой сортировки на языке Haskell

```
qsort []      = []
qsort (x:XS) = qsort Elts_lt_x ++ [x] ++ qsort Elts_greq_x
                 where
                   Elts_lt_x   = [y | y <- XS, y < x]
                   Elts_greq_x = [y | y <- XS, y >= x]
```

- Результат сортировки пустого списка – пустой список.
- (x:XS) - список с первым элементом x и хвостом XS. Для его сортировки нужно:
  1. Отсортировать все элементы из XS, которые меньше x (Elts_lt_x).
  2. Отсортировать все элементы из XS, которые больше либо равны x (Elts_greq_x).
  3. Объединить (++) результаты, поставив x между ними.

"**|**" означает "**такие, что**"
"**<-**" означает "**принадлежит множеству** (является элементом списка)"


Высокоуровневые языки программирования
Языки общего назначения
Предметно-ориентированные языки
Algol, C/C++, Pascal, Java, Python, …
Работа с текстом
Perl, Regular expressions
Работа с базами данных
SQL
Символьные вычисления
Mathematica, Maple
Управление ОС
Bash, PowerShell


Языки программирования
Бестиповые
Assembler, FORTH
Типизированные

Виды типизации

Когда проверять типы?

- **Статическая.** Конечные типы переменных и функций устанавливаются на этапе компиляции. Писать программу дольше, но компилятор делает проверки. *C, Java, C#*.
- **Динамическая.** Быстрее писать, проще сломать. *Python, JavaScript, Ruby*

Виды типизации

- **Статическая.** Конечные типы переменных и функций устанавливаются на этапе компиляции. Писать программу дольше, но компилятор делает проверки. *C, Java, C#*.
- **Динамическая.** Быстрее писать, проще сломать. *Python, JavaScript, Ruby*

Насколько серьезно проверять типы?

- **Сильная.** Нельзя смешивать в выражениях разные типы, нет неявных автоматических преобразований. *Java, Python, Haskell, Lisp*
- **Слабая.** *JavaScript, Visual Basic, PHP*

Виды типизации

- **Статическая.** Конечные типы переменных и функций устанавливаются при объявлении переменной. Писать программу дольше, но компилятор делает проверки. *C, Java, C#*.
- **Динамическая.** Переменная связывается с типом в момент присваивания значения. Быстрее писать, проще сломать. *Python, JavaScript, Ruby*

- **Сильная.** Нельзя смешивать в выражениях разные типы, нет неявных автоматических преобразований. *Java, Python, Haskell, Lisp*
- **Слабая.** *JavaScript, Visual Basic, PHP*

- **Явная.** Тип новых переменных и функций нужно задавать явно. *C++, Pascal, C#*
- **Неявная.** *JavaScript, PHP, Python*

Типизация в языках программирования

JavaScript	Динамическая	Слабая	Неявная
Ruby	Динамическая	Сильная	Неявная
Python	Динамическая	Сильная	Неявная
Java	Статическая	Сильная	Явная
PHP	Динамическая	Слабая	Неявная
C	Статическая	Слабая	Явная
C++	Статическая	Слабая	Явная
Perl	Динамическая	Слабая	Неявная
Objective-C	Статическая	Слабая	Явная
C#	Статическая	Сильная	Явная
Haskell	Статическая	Сильная	Неявная
Common Lisp	Динамическая	Сильная	Неявная
D	Статическая	Сильная	Явная
Delphi	Статическая	Сильная	Явная

Эзотерические языки программирования

Brainfuck (Урбан Мюллер, 1993). Восемь односимвольных команд, записываются подряд.

Brainfuck	C
>	++ptr;
<	--ptr;
+	++*ptr;
-	--*ptr;
.	putchar(*ptr);
,	*ptr = getchar();
[	while (*ptr) {
]	}

Hello, world!

```
++++++++++[>+++++++>++++++++++>+++>+<<<<-]>++
.>+.+++++++..+++.>++.<<+++++++++++++++.>.+++.
------.--------.>+.>.
```

Эзотерические языки программирования

Язык Petooh

```
KoKoKoKoKoKoKoKoKoKo Kud-Kudah
KoKoKoKoKoKoKoKo kudah kO kud-Kudah Kukarek kudah
KoKoKo Kud-Kudah
kOkOkOkO kudah kO kud-Kudah Ko Kukarek kudah
KoKoKoKo Kud-Kudah KoKoKoKo kudah kO kud-Kudah kO Kukarek
kOkOkOkOkO Kukarek Kukarek kOkOkOkOkOkOkO
Kukarek
```

```
sub cycle {
    while ( $result[$current_cell] > 0 ) {
        foreach my $item ( @{$stack->{$level}} ) {
            if ($item eq "Ko") { $result[$current_cell]++ }
            elsif ($item eq "kO") { $result[$current_cell]-- }
            elsif ($item eq "Kudah") { $current_cell++ }
            elsif ($item eq "kudah") { $current_cell-- }
            elsif ($item eq "Kukarek") { print chr $result[$current_cell]
```