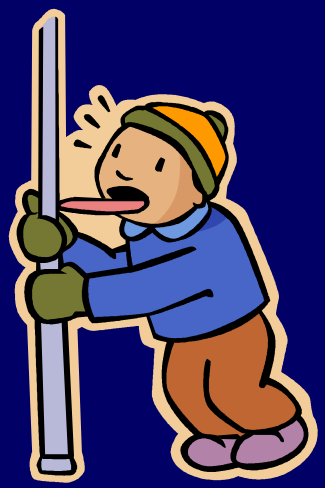
Введение в языки программирования



Границы моего языка означают границы моего мира.

Л. Витгенштейн

Содержание

- О курсе
- Мотивация изучения языков программирования
- Понятие языка программирования
- Основные элементы языков программирования
- Критерии оценки языков программирования
- Классификация языков программирования
- Эволюция языков программирования

О курсе

- Распределение часов:
  - Лекции – 36 час.
  - Практические – 36 час.
- Форма контроля: экзамен
- Страница курса: http://mzym.susu.ru/courses/pl/
- Основная литература:
  - Бен-Ари М. Языки программирования. Практический сравнительный анализ. –М.: Мир, 2000.
  - Себеста Р. Основные концепции языков программирования. –М.: Вильямс, 2001.

Для чего изучать языки программирования?

- Больше возможностей для выражения идей
- Более обоснованный выбор подходящего языка
- Повышение способности к изучению новых языков
- Углубление понимания важности реализации
- Повышение способности к разработке новых языков

Язык программирования

- *Язык программирования* – система обозначений для *абстрактного* (не зависящего от архитектуры конкретного компьютера) описания вычислений.
- Абстрактное описание вычислений можно перевести в детализированную форму для последующего выполнения на компьютере.
- Перевод осуществляется специализированной программой (*ассемблер*, *компилятор*, *интерпретатор*).

Основные элементы языков программирования

- Алфавит, синтаксис, семантика
- Типы данных, значения, литералы, переменные, константы
- Операторы
- Подпрограммы
- Библиотеки
- Стандартизация языков и мобильность программ

Способы задания синтаксиса

- *Расширенные формулы Бэкуса-Наура (РБНФ)*
  - Нетерминальные символы – обозначают синтаксические конструкции языка, заключаются в угловые скобки <...>.
  - Терминальные символы образуют алфавит языка.
  - Метасимволы
    - ::= есть по определению
    - | или
  - Дополнительные метасимволы (расширение БНФ)
    - […] повторение символа 0 или 1 раз
    - {…} повторение символа произвольное число раз (в т.ч. нуль)
    - (…) группировка символов
- *Синтаксические диаграммы* – графическое изображение РБНФ.

Синтаксис: основные ловушки

- Ограничения на длину идентификаторов
  current_winner_number
  current_width
- Регистр символов
  Number, NUMBER, number
- Комментарии
  - Однострочные
    -- в Ada и PL/SQL, // в C++, C в Fortran
  - Многострочные
    /* … */ в C, { … } и (*…*) в Pascal
  - Пропущенные операторы
    { a:=b+c;
    { Комментарий }

Синтаксис: основные ловушки

- Похожие символы

  Присвоить: **:=** в Pascal, **=** в C и Fortran

  Равно: **=** в Pascal, **==** в C, **.eq.** в Fortran

- Значащие символы, или когда взрываются ракеты?

```
DO 10 I = 1,100    C Это оператор цикла
DO 10 I = 1.100    C Это оператор присваивания
                   C DO10I = 1.100
                   C т.к. пробел игнорируется!
```

Семантика

- Пример: семантика оператора
  if C then S1else S2
  - *Формальное описание*
    (C⇒Семантика S1) & (⌉C⇒Семантика S2)
  - *Неформальное описание*
    Проверяется условие после if; если результат Истина, то выполняется оператор, следующий после then, иначе выполняется оператор, следующий за else.
- Формальное описание семантики всех конструкций языка может быть очень сложным и, как правило, не применятся.

Семантика языка и правильность программ

- Формальное описание семантики дает возможность доказать *правильность программы* в следующем смысле:
  - оператор – аксиома, описывающая преобразование состояние, для которого верно некоторое входное утверждение, в состояние, для которого верно некоторое выходное утверждение
  - семантика программы "вычисляется" путем построения входных и выходных утверждений на основе утверждений для отдельных операторов
  - результат вычислений – доказательство того, что если входные данные удовлетворяют утверждению на входе, то выходные данные удовлетворяют утверждению на выходе

Типы данных

- *Тип данных* – множество *значений* и множество *операций* над этими значениями.
- Определение множеств значений и операций зависит от языка программирования и его конкретной реализации.
- Примеры:
  - типы Integer в Pascal и int в C – это конечное множество целочисленных значений (в количестве ≅ 65 тыс. или 4 млрд. – в зависимости от компьютера) и конечного набора операций (+, -, *, >, …)
  - тип "массив" – индексированная совокупность элементов других (ранее определенных) типов с операцией индексации; в Ada над массивами определена также операция присваивания.

Переменная, константа

- *Значение* – интуитивно понятный термин.
- *Представление* – строка битов для указанного значения (например: представление целого числа 201 есть 11001001).
- *Переменная* – имя ячеек, содержащих представление значения указанного типа, которое может изменяться во время выполнения программы.
- *Константа* – имя ячеек, содержащих представление значения указанного типа, которое *не* может изменяться во время выполнения программы.
- *Литерал* – последовательность символов, которая в программе непосредственно задает значение (например: 201, 2.01, '0', "Строка", TRUE).

Оператор присваивания

- *Операторы* – конструкции языка для управления вычислениями.
- *Оператор присваивания* изменяет значение указанной переменной на указанное новое значение.
  Pascal : `A := A+1;`
  C : `m[a*i] = r[k+2] * a;`
- Алгоритм работы оператора присваивания
  1. Вычислить значение выражения в правой части
  2. Вычислить значение выражения в левой части, получить адрес ячейки
  3. Скопировать значение, вычисленное на шаге 1., в ячейки памяти, начиная с адреса, полученного на шаге 2.

Контроль соответствия типов

- *Контроль соответствия типов* – проверка того, что при выполнении присваивания тип выражения совместим с типом адресуемой переменной.
- Подходы к контролю соответствия типов:
  - *Отсутствие контроля*: программист отвечает за последствия выполнения присваивания.
  - *Неявное преобразование* значения выражения к типу адресуемой переменной.
  - *Строгий контроль*: отказ от выполнения присваивания при несовместимости типов.

Управляющие операторы

- *Управляющие операторы* используются для изменения порядка выполнения.
- *Условные операторы* позволяют выбрать одну из нескольких альтернативных последовательностей управления.

```
if A>5 then
        Z:=1
else
        Z:=A+B;
```

```
case K of
0..3 : P:=1;
7, 9 : P:=233;
else
        P:=0;
end;
```

- *Операторы цикла* позволяют повторить выполнение последовательности операторов.

```
for (i=0; i<N; i++)
        A[i]=0;
```

```
while i<N do begin
        F := i*F;
        i:=i+1;
end;
```

Подпрограммы

- *Подпрограмма* – сегмент программы, состоящий из объявлений данных и операторов, которые можно многократно *вызывать* (call) из различных частей программы.
- *Параметры подпрограммы* – последовательность значений, передаваемая в подпрограмму при ее вызове для задания различных вариантов выполнения, передачи исходных данных и получения результатов.

Подпрограммы

- Подпрограмма – важный элемент программной структуры.
  - Каждая подзадача программы должна быть оформлена в виде подпрограммы.
  - Подпрограммы позволяют разделить программу на небольшие, хорошо понятные и читаемые части.
- Виды подпрограмм:
  - Pascal : процедуры (procedure), функции (function)
  - C : функции
  - FORTRAN : суб-рутины (SUBROUTINE)
  - ОО-языки : методы

Библиотеки подпрограмм

- *Библиотека* – вспомогательная неисполняемая программная единица, содержащая определения подпрограмм и данных, которые могут быть вызваны из основной программы.
- Библиотеки необходимы для коллективной разработки больших программных систем (размером от 1 тыс. строк).
- Примеры библиотек:
  - Pascal : нет языковых средств
  - Turbo Pascal : unit
  - Modula : module
  - C : только средства раздельной трансляции
  - Ada : package

Стандартизация языков

- *Стандарт языка* – официальный документ одной из международных организаций по стандартам (ISO – Int. Standards Org., ANSI – Amer. Nat. Standards Inst. и др.), в котором зафиксированы алфавит, синтаксис и семантика языка.
- *Мобильная программа* выполняется одинаково на различных компьютерах и/или операционных системах.
- Мобильность программ возможна, если для языка существует стандарт и различные трансляторы языка поддерживают данный стандарт.

Критерии оценки языков программирования

- Читабельность – легкость чтения и понимания программ
- Легкость и удобство языка для создания программ
- Надежность – выполнение программой предназначенных ей действий при любых условиях
- Стоимость

Читабельность

- Простота
  - Количество элементарных конструкций
  - Множественность свойств
  - Перегрузка операторов
- Ортогональность – наличие относительно небольшого количества элементарных конструкций, с помощью которых выражаются управляющие операторы и структуры данных
- Управляющие операторы и оператор goto
- Адекватные средства определения типов и структур данных
- Синтаксис
  - Правила образования идентификаторов
  - Написание и семантика ключевых слов

Легкость создания, надежность

- Легкость
  - Простота и ортогональность
  - Поддержка абстракции
  - Выразительность
- Надежность
  - Проверка совместимости типов при компиляции
  - Средства обработки исключительных ситуаций
  - Совмещение имен (несколько имен для одной и той же ячейки памяти)
  - Легкость чтения и использования

Стоимость

- Затраты на обучение программистов
  - Простота и ортогональность языка, опыт человека
- Затраты на создание программ
  - Легкость и удобство языка для конкретного приложения, наличие среды разработки
- Затраты на разработку компилятора
- Стоимость и эффективность выполнения программ

Критерии классификации языков

- *Поколение* языка
- *Уровень абстракции* языка
  Абстракция – выделение существенных и отбрасывание несущественных в данном контексте характеристик объекта.
- *Парадигма* языка
  Парадигма – базовая модель постановки задач и их решения.
- *Назначение* языка
- …

Классификация языков (по уровню абстракции)

Языки программирования

Языки высокого уровня

- Процедурные языки (Fortran, Cobol, PL/1, Algol, Pascal, C, Ada, …)
- Языки обработки данных (LISP, APL, Snobol, Icon, SETL, …)
- OO-языки (Smalltalk, Simula, Eiffel, C++, Ada95, …)
- …

Языки низкого уровня

- Машинные языки
- Языки ассемблеров

Классификация языков (по уровню абстракции)

- *Языки высокого уровня*
  - Наличие понятия *типа данных*.
  - Независимость от архитектуры конкретного компьютера (*мобильность программ*).
  - Развитые управляющие структуры и средства описания структур данных.
  - Близость к естественному языку.
- *Языки низкого уровня*
  - Отсутствие понятия *типа данных*.
  - Зависимость от архитектуры конкретного компьютера (отсутствует *мобильность программ*).
  - Примитивные управляющие структуры и средства описания структур данных.
  - Близость к машинному языку.

Пример: вычисление факториала

Машинный язык	Язык ассемблера	Язык высокого уровня Паскаль

10111010
00001100
00000001
00101110
10001001
00010110
10010001
00000010
10001011
00011110
00101100
00000000
10001110
11011010
10100011 …
	mov AX,1
	mov BX,N
	@2:
	cmp BX,0
	je @1
	imul BX
	dec BX
	jmp @2
	@1:
		function Factorial (N: **Integer**) : **Integer**;
		var
		i, F: **Integer**;
		begin
		F:=1;
		for i:=1 **to** N **do**
		F:=F*i;
		Fact:=F;
		end;

Классификация языков (по парадигме)

Языки программирования *

Императивныеязыки

- Процедурные языки

Fortran, Cobol, PL/1, Algol, Pascal, C, Ada, …

- ОО-языки

Smalltalk, Simula, Eiffel, C++, Ada95, …

Декларативные языки

- Функциональные языки

Lisp, ML, Haskell, …

- Логические языки

Prolog, …

Параллельные языки

occam, Ada95, Linda, Parlog, Lisp2D

* Язык может сочетать в себе более одной парадигмы

Классификация языков (по парадигме)

- *Императивный* язык рассматривает программу как описание последовательности действий по получению искомого результата.
- *Процедурно-ориентированный* язык рассматривает программу как совокупность подпрограмм (процедур), обменивающихся данными в процессе их (подпрограмм) вызова из основной программы.
- *Объектно-ориентированный* язык рассматривает программу как совокупность объектов (имитирующих объекты реального мира), обменивающихся сообщениями в процессе вызова их (объектов) методов в основной программе.

Классификация языков (по парадигме)

- *Декларативный* язык рассматривает программу как совокупность описания входных данных и описания искомого результата.
  - *Функциональный* язык рассматривает программу как совокупность определений функций, которые обмениваются между собой данными без использования промежуточных переменных и присваиваний.
  - *Логический* язык рассматривает программу как описание задачи в терминах фактов и логических формул, а решение задачи выполняет система с помощью механизмов логического вывода.

Классификация языков (по парадигме)

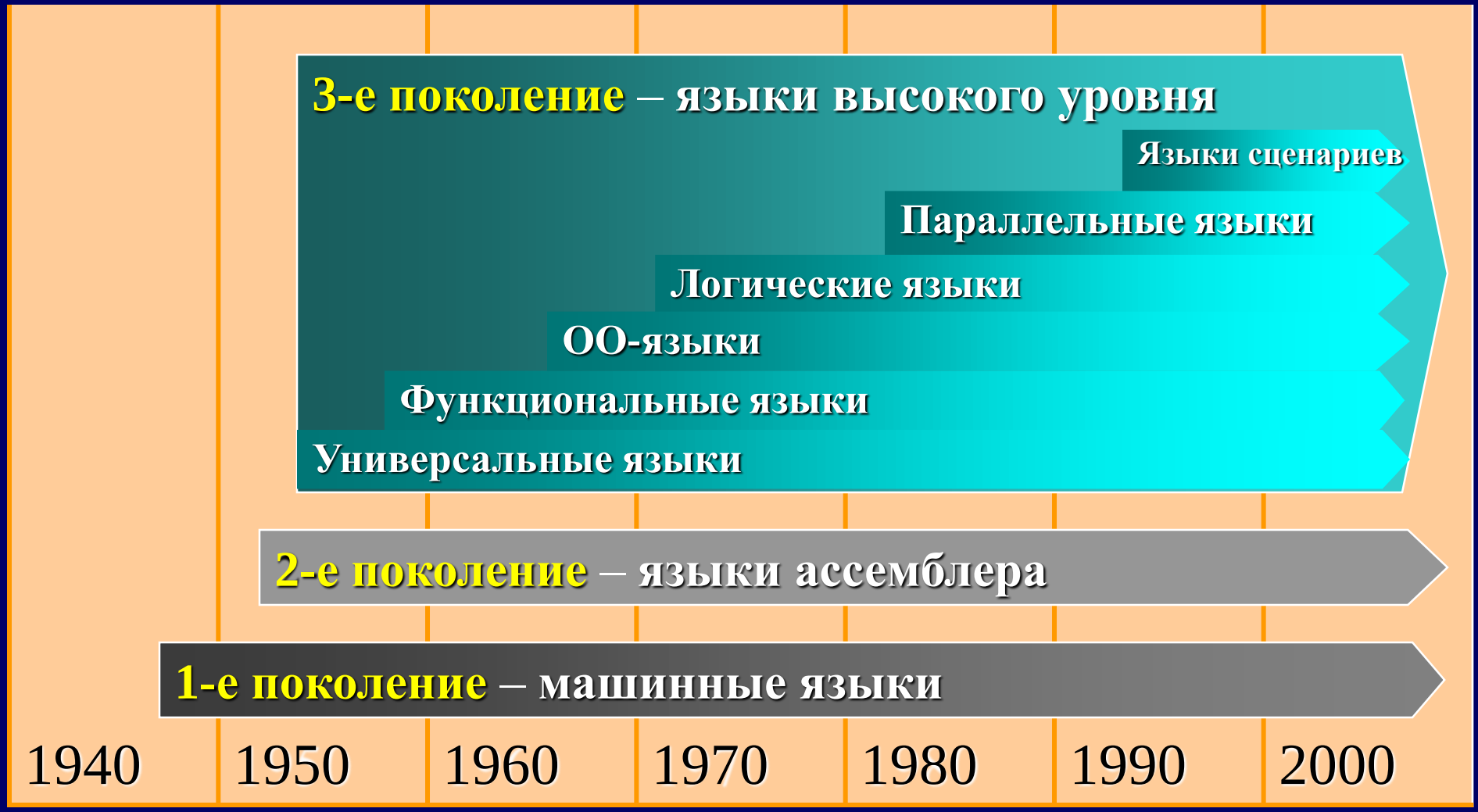
- *Параллельный* язык предполагает использование в программе явных конструкций для параллельного исполнения фрагментов последовательности действий по получению искомого результата.

Пример: вычисление факториала

Императивный процедурный язык Паскаль	Декларативный логический язык ПРОЛОГ

function Factorial (N: **Integer**): **Integer**;
var
  i, F: **Integer**;
begin
  F:=1;
  for i:=1 **to** N **do**
  F:=F*i;
  Fact:=F;
end;
	Factorial (0, 1)**.**
	Factorial (1, 1)**.**
	Factorial (F, N) **:-**
	  N1 **is** N-1,
	  N > 1,
	  Factorial (F1, N1),
	  F **is** N*F1**.**

Эволюция языков программирования



Эволюция языков программирования

- *Язык 1-го поколения (машинный язык)* – система машинных команд конкретного компьютера.
- *Язык 2-го поколения (язык ассемблера)* – система мнемоник для обозначения машинных команд конкретного семейства компьютеров.
- *Язык 3-го поколения (язык высокого уровня)* – система обозначений для абстрактного описания вычислений.
- *Язык 4-го поколения (?)* – система визуального проектирования пользовательских приложений, выполняющая автоматическую генерацию соответствующих программ на языке программирования 3-го поколения.

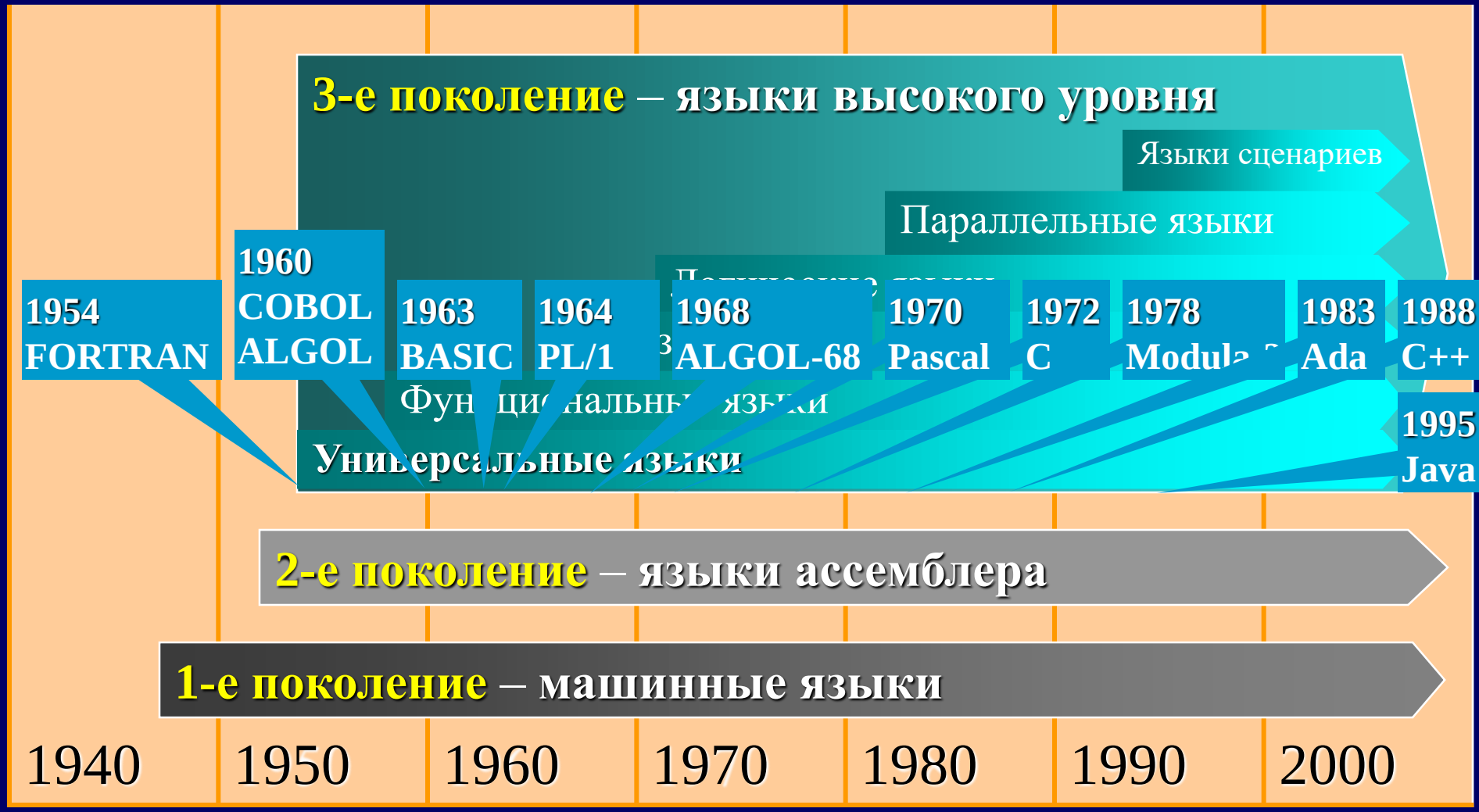
Эволюция: язык Plankalkul

Конрад Цузе
(1910-1995)

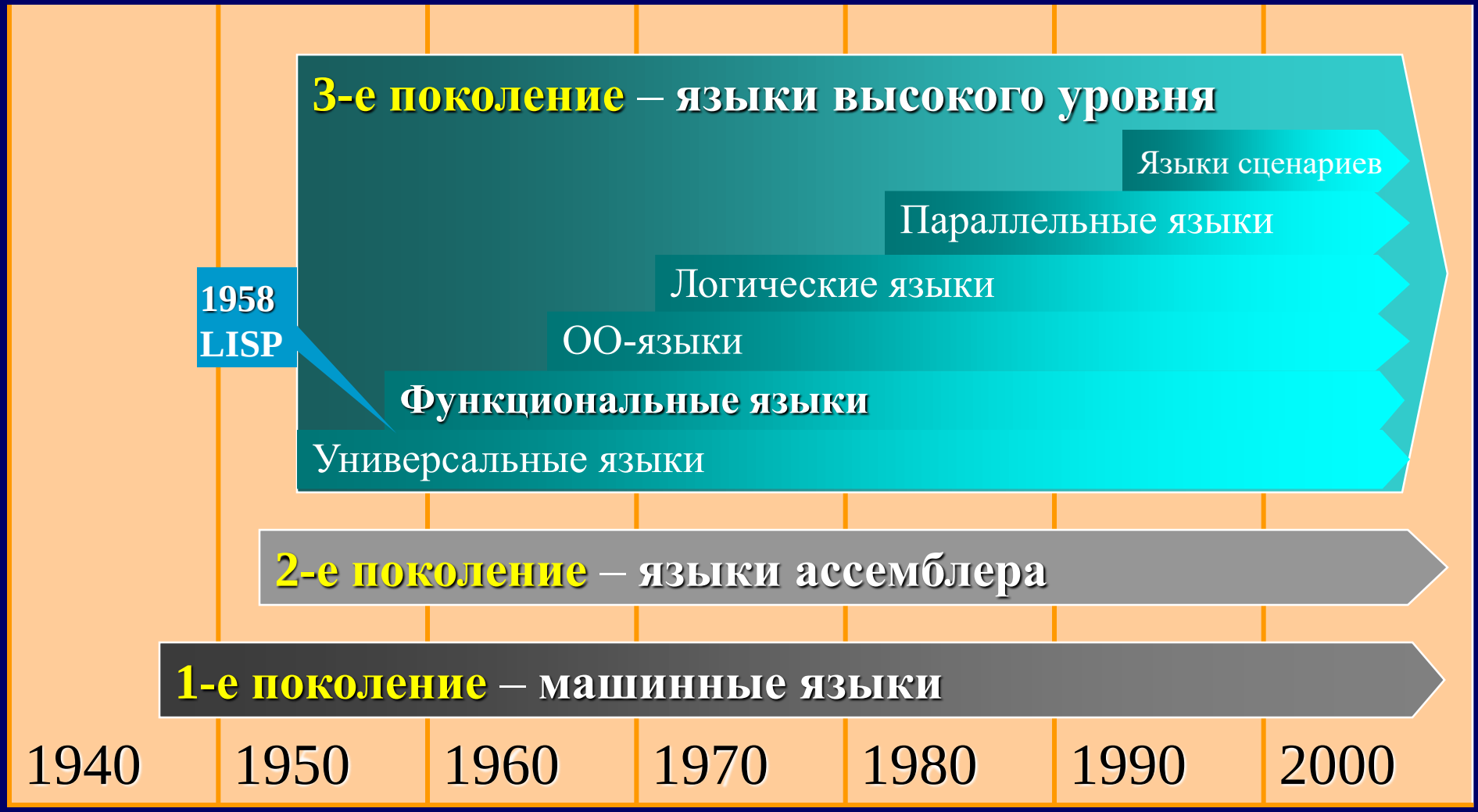
- Простейший тип данных – бит, на основе которого создавались типы для представления целых чисел и чисел с плавающей точкой.
- Язык содержал:
  - массивы и записи (допускалась рекурсия при определении записей);
  - for-подобный оператор цикла;
  - условный оператор без части else.
- Язык не имел оператора goto.
- Пример оператора A[5]:=A[4]+1

```
    |  A + 1 => A        операторная строка
V   |  4       1         строка индексов
S   |  1.n     1.n       строка типов (n битов)
```

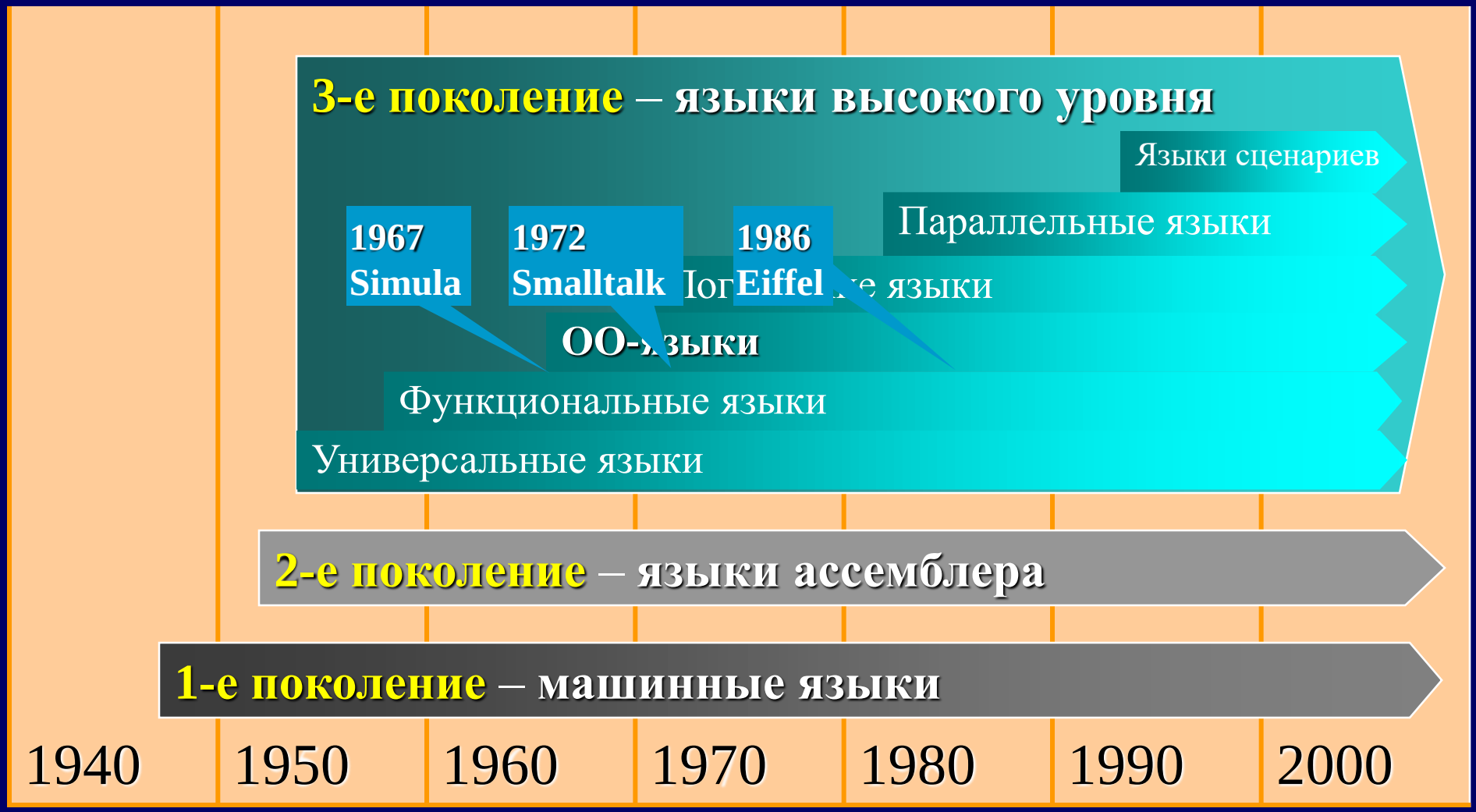
Эволюция: универсальные языки



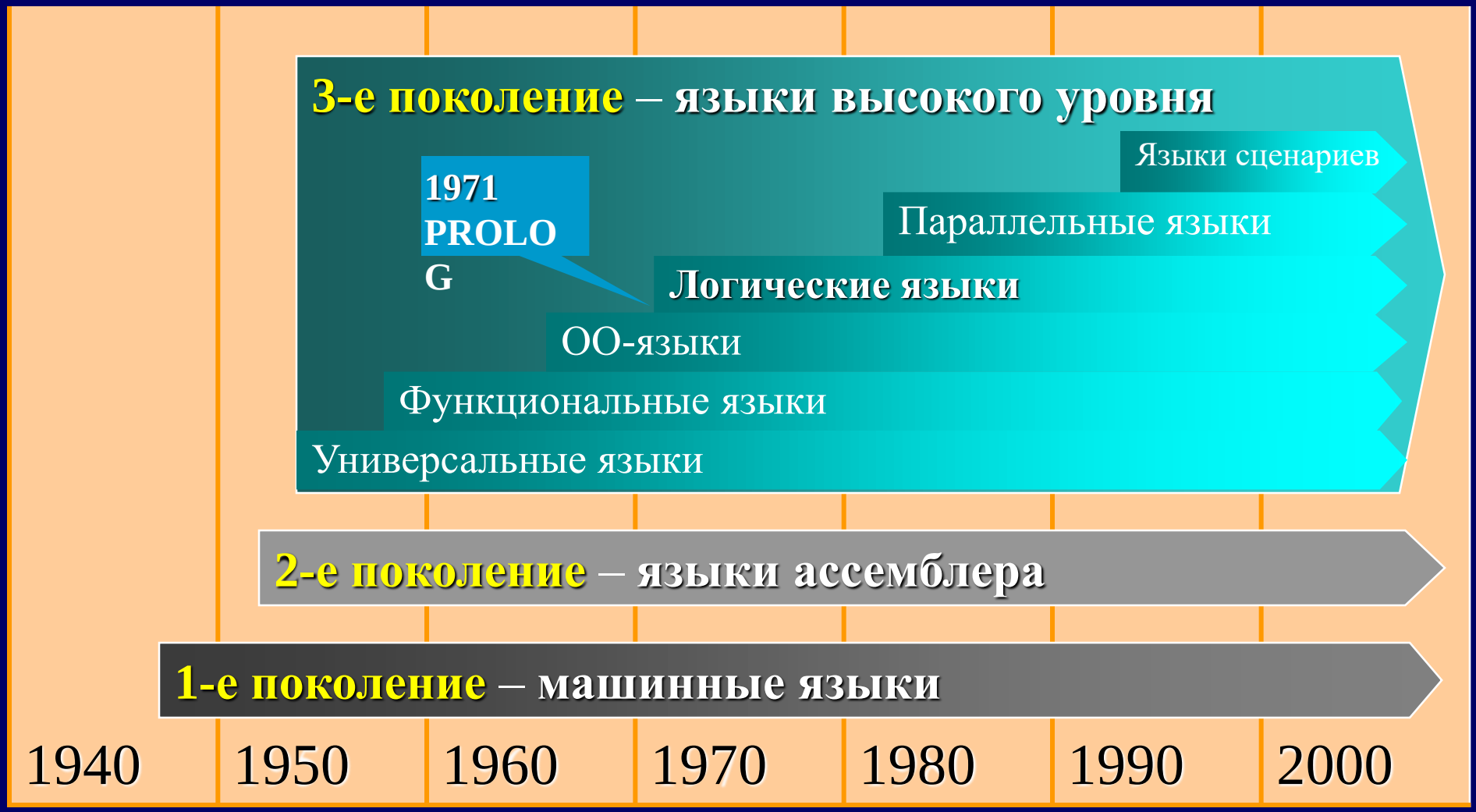
Эволюция: функциональные языки



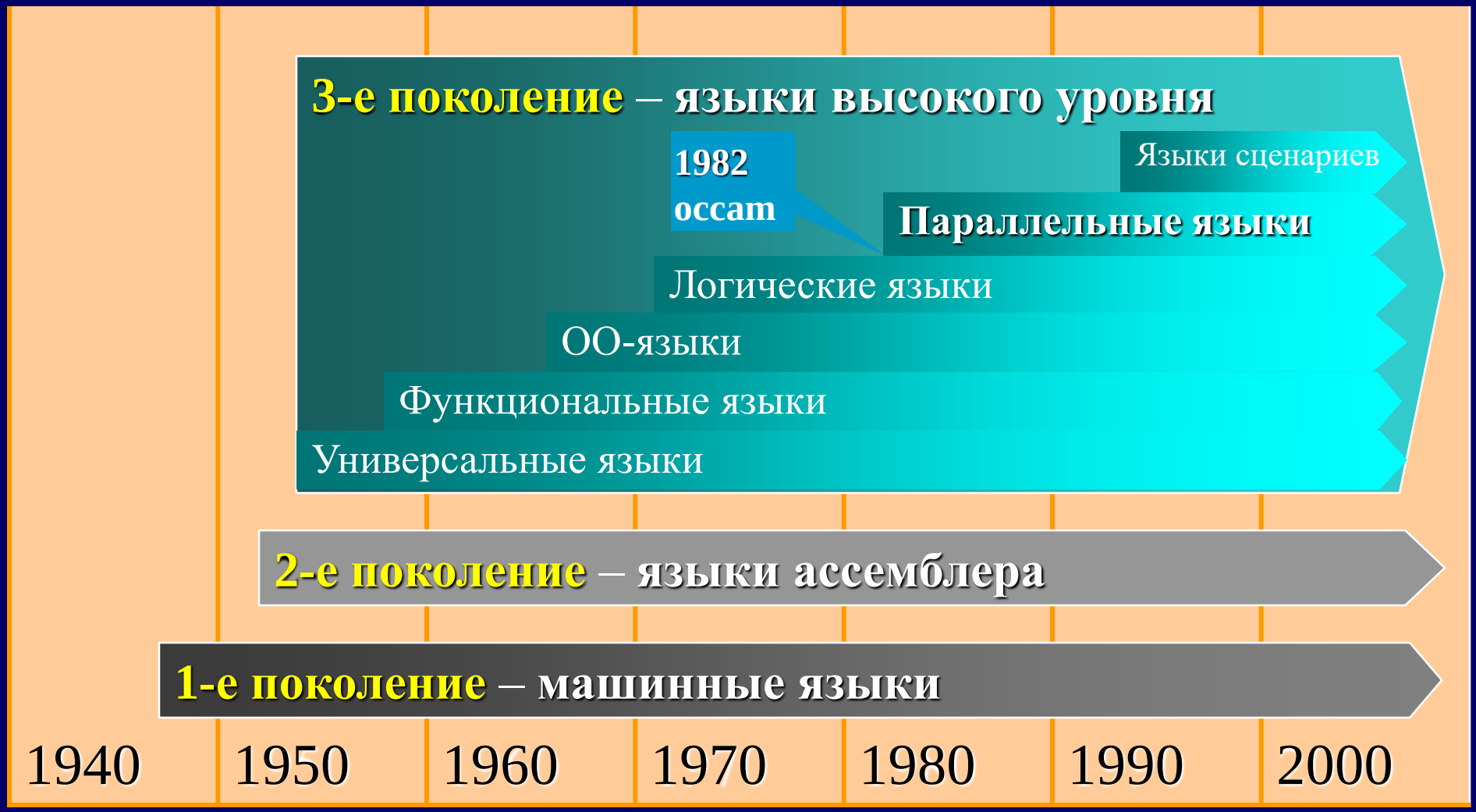
Эволюция: ОО-языки



Эволюция: логические языки



Эволюция: параллельные языки



Эволюция: языки сценариев