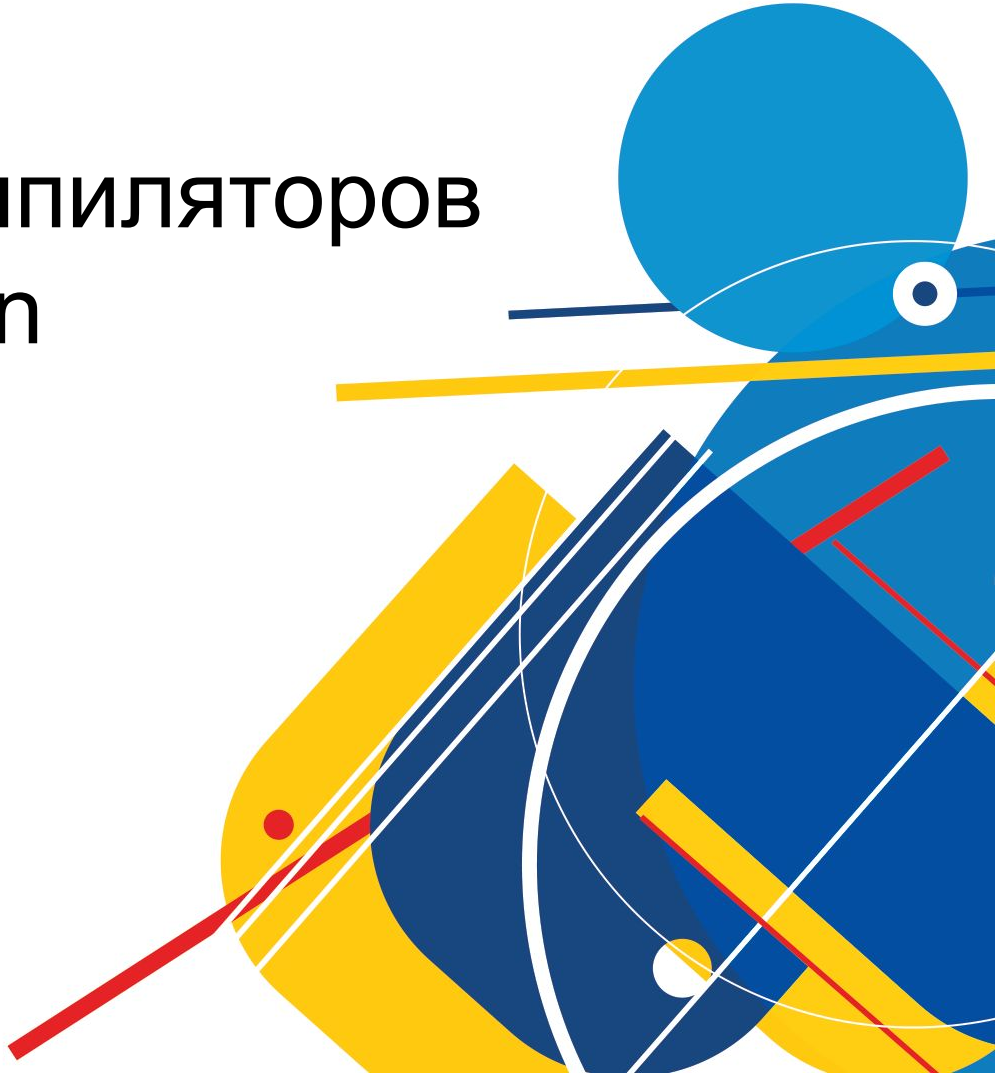
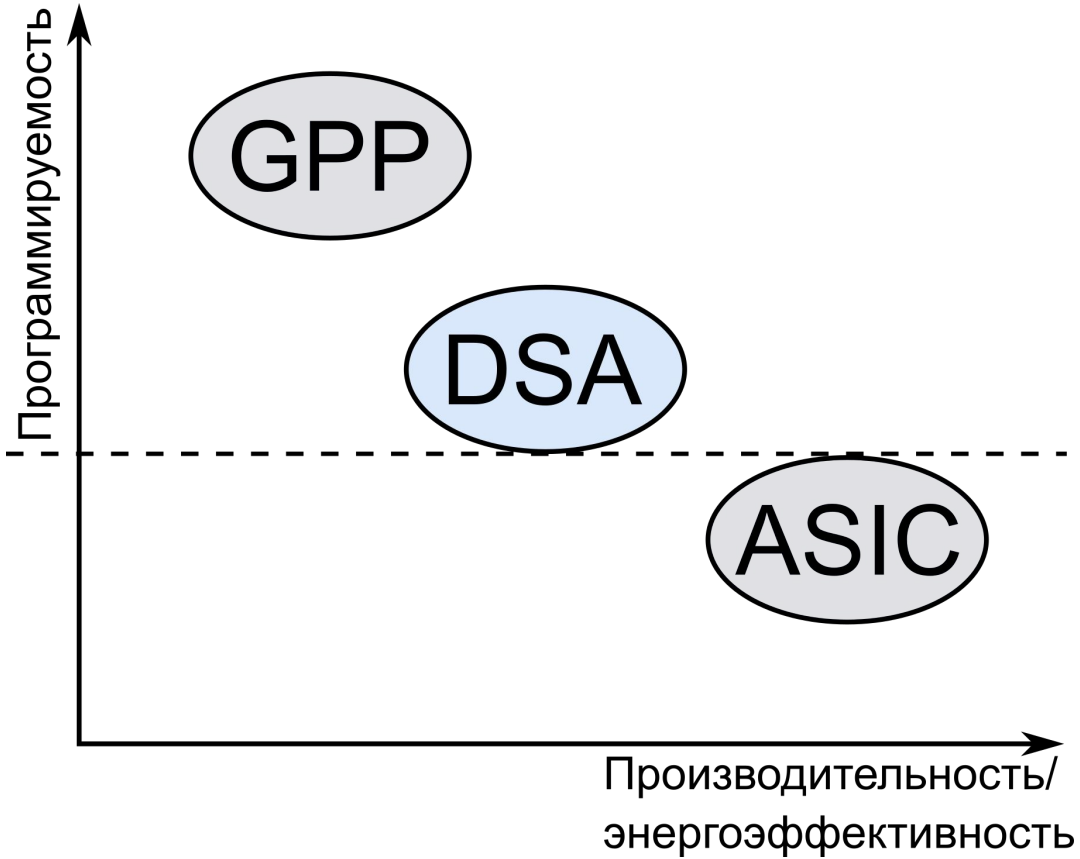
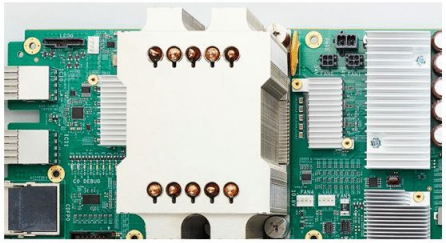
Создание DSL-компиляторов на Python

Пётр Советов

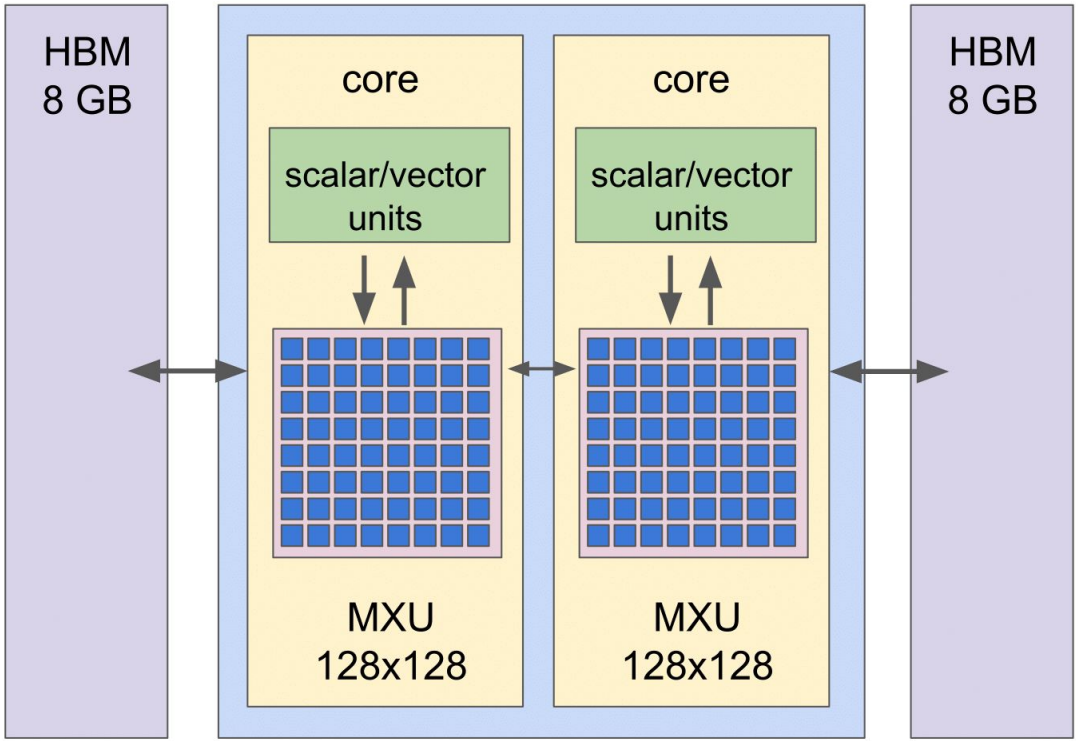




TPUv2 Chip



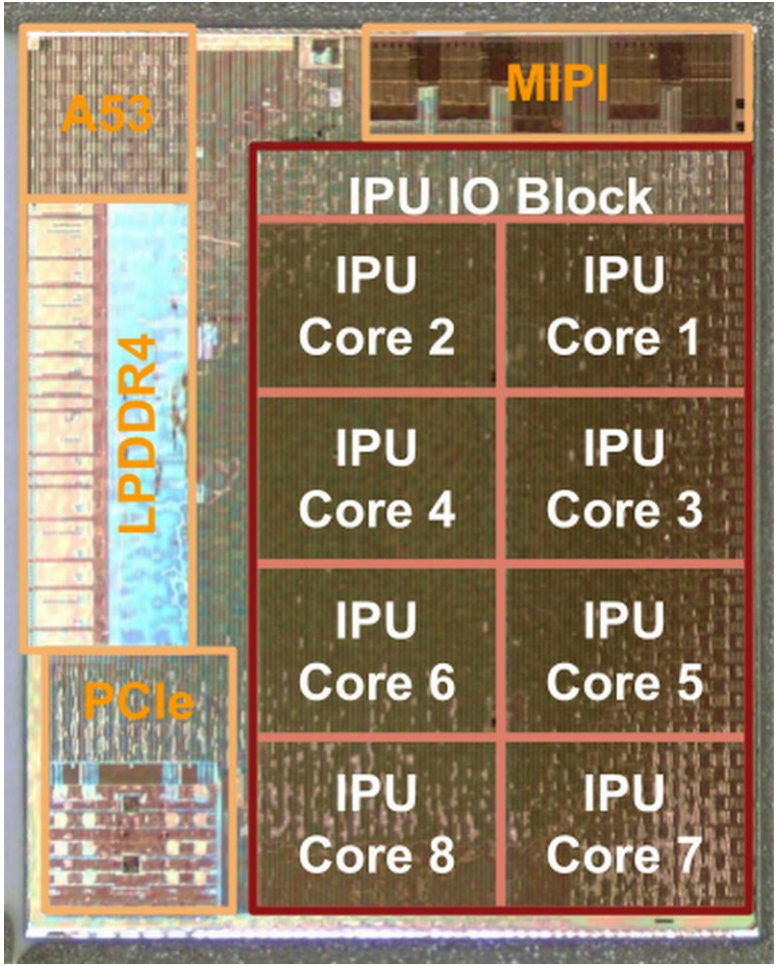
- 16 GB of HBM
- 600 GB/s mem BW
- Scalar/vector units: 32b float
- MXU: 32b float accumulation but reduced precision for multipliers
- 45 TFLOPS



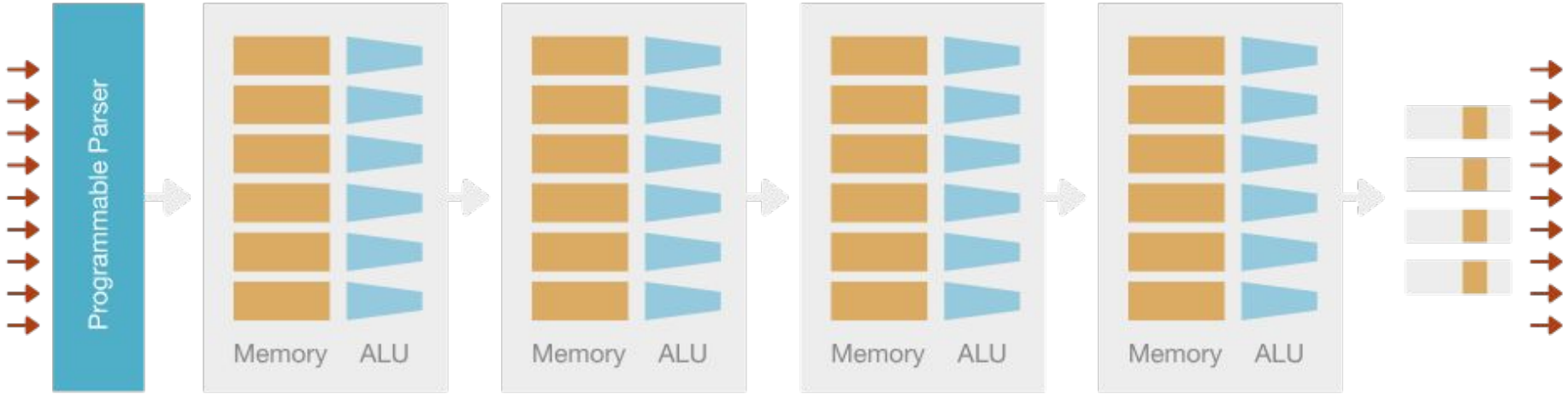
DSA Google TPU2, библиотека TensorFlow, DSL-компилятор XLA

https://www.anandtech.com/show/12429/google-cloud-announces-cloud-tpu-beta-availability

DSA Google Pixel Visual Core,
Halide: DSL и компилятор



https://www.blog.google/products/pixel/pixel-visual-core-image-processing-and-machine-learning-pixel-2/



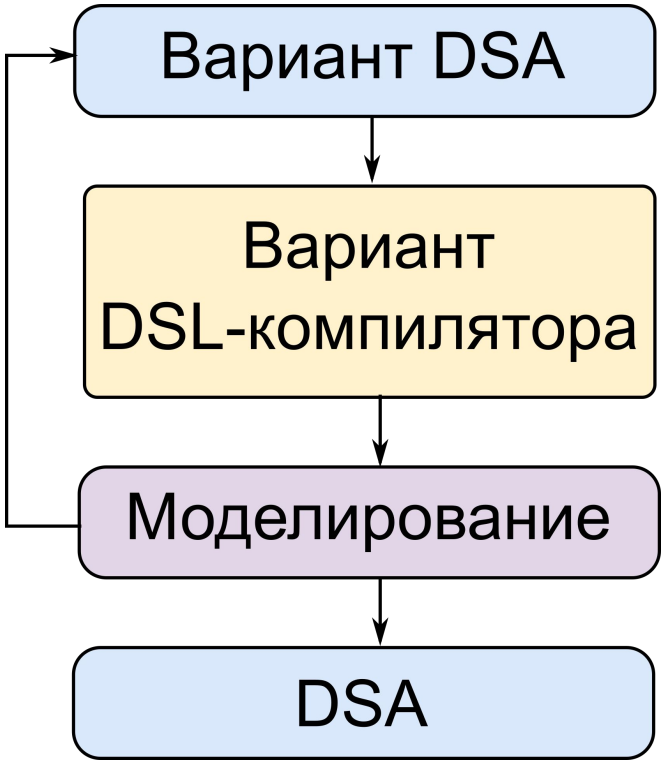
DSA Barefoot Tofino, DSL P4,
DSL-компилятор Barefoot P4 Compiler

https://barefootnetworks.com/products/brief-tofino/

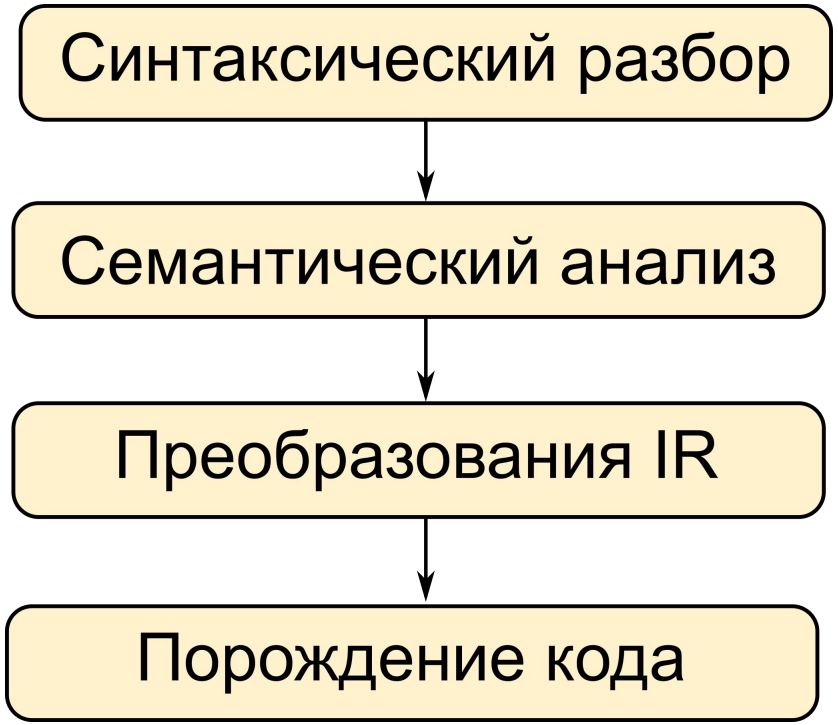
Мой опыт создания DSL-компиляторов

- Компилятор (подмножество C99) для многоядерного ускорителя (28 нм).
- Компиляторы для ограниченных по ресурсам софт-ядер (FPGA).

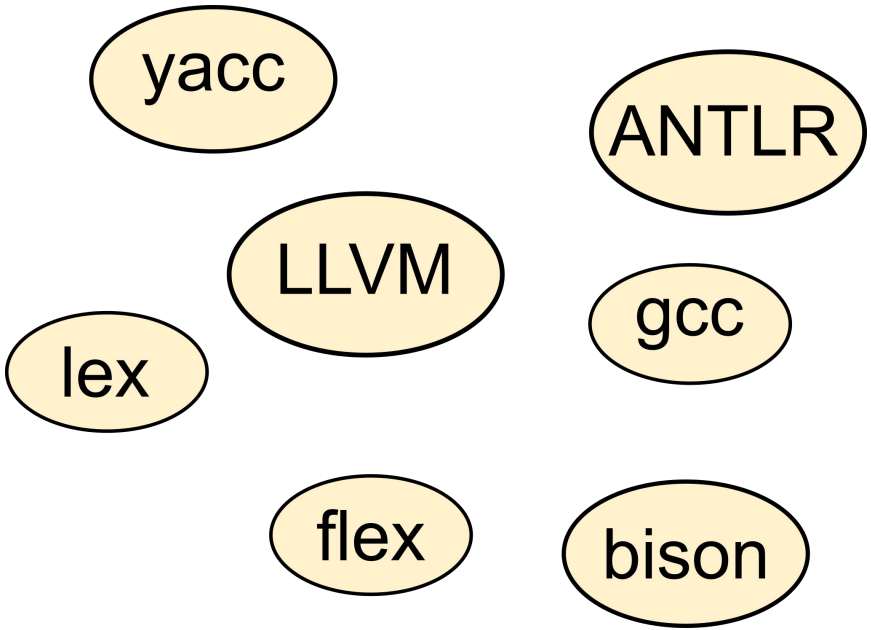
Итеративный подход compiler-in-the-loop



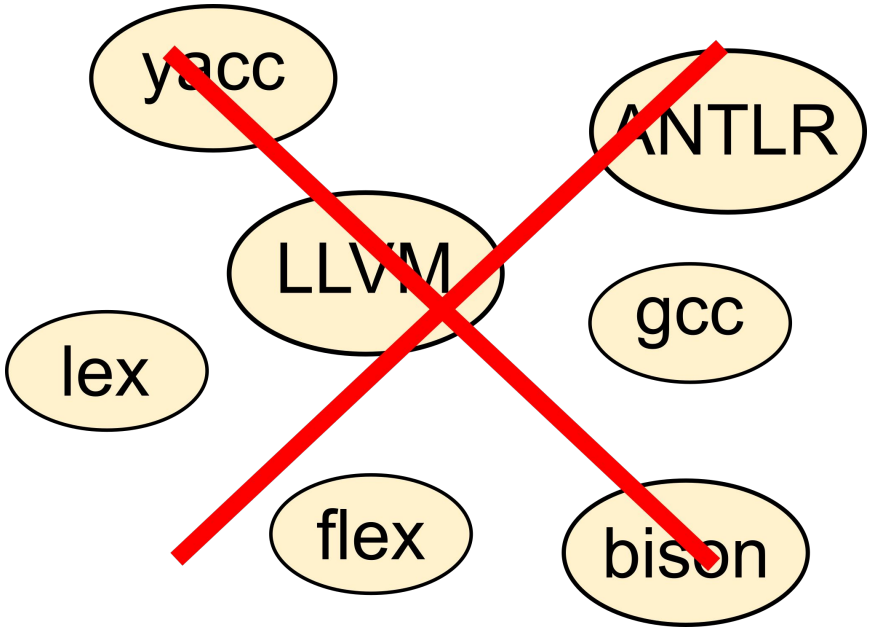
Основные стадии компиляции



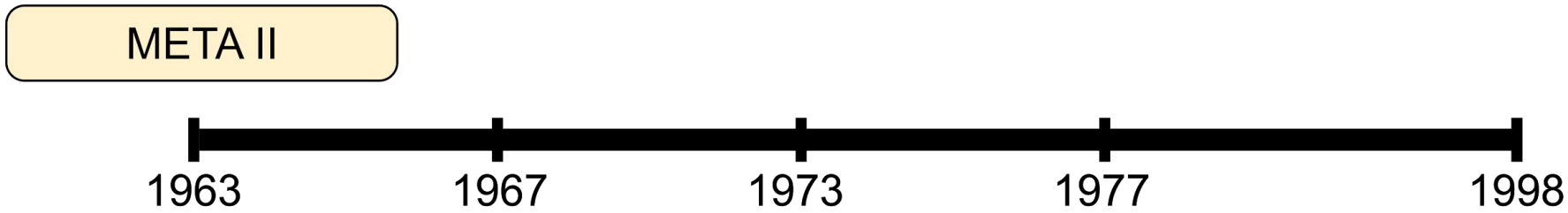
Традиционные средства разработки компиляторов...



...мало помогают при использовании подхода compiler-in-the-loop

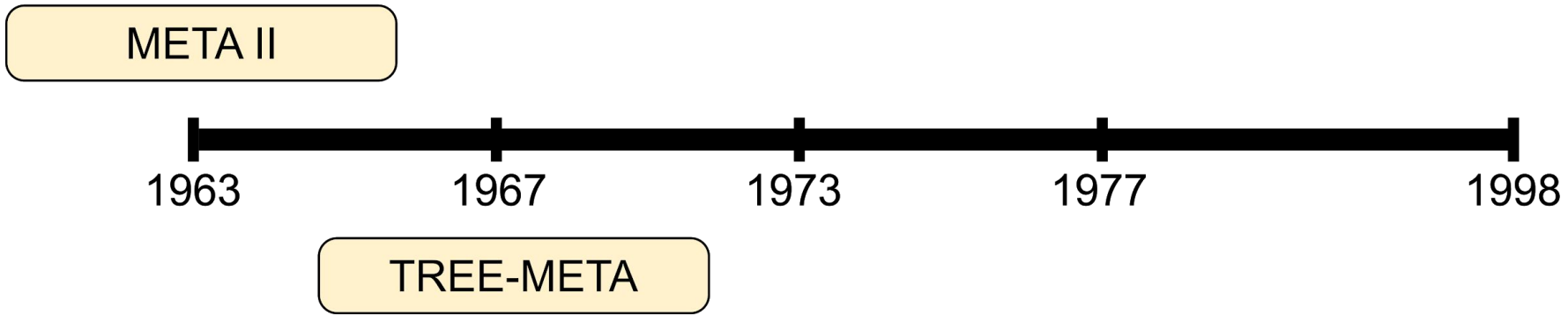


Быстрая разработка компиляторов (история)



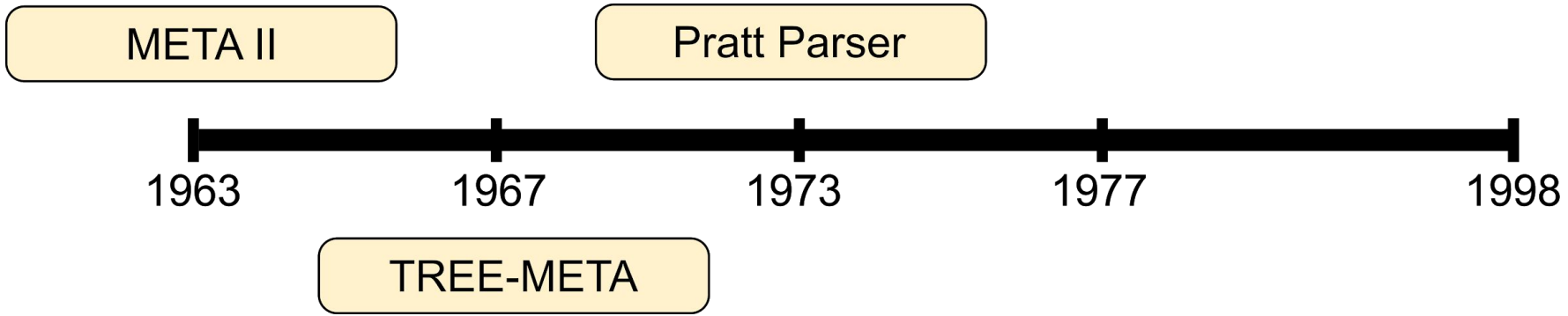
PEG

Быстрая разработка компиляторов (история)



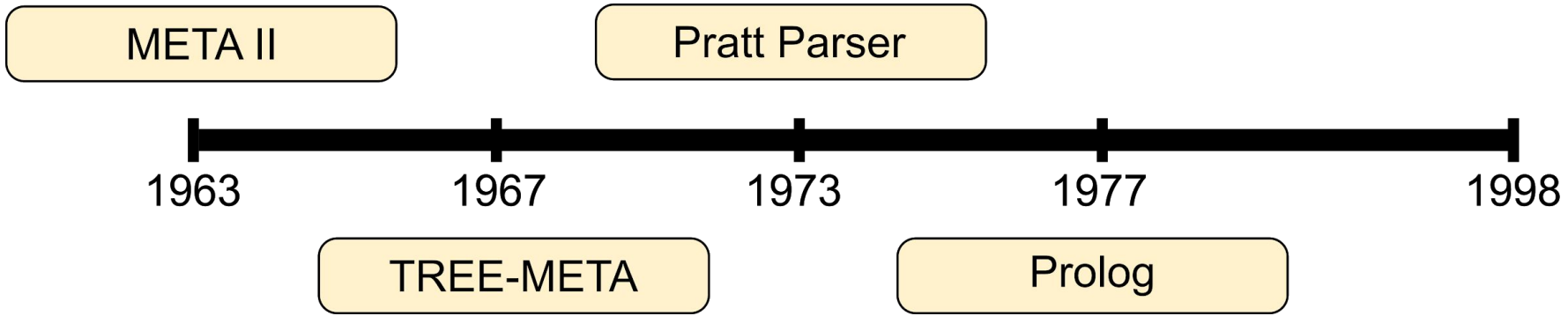
PEG + tree matching

Быстрая разработка компиляторов (история)



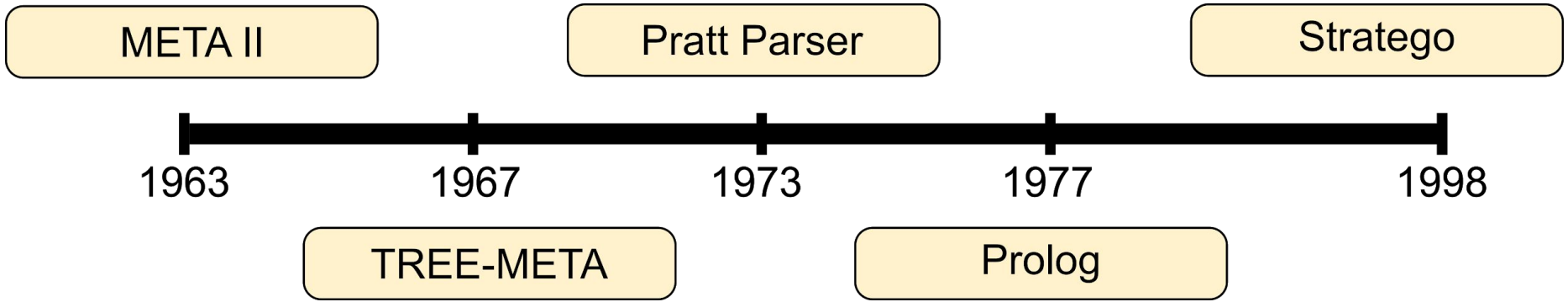
Recursive descent + operator precedence

Быстрая разработка компиляторов (история)



"The specification is the implementation"

Быстрая разработка компиляторов (история)



Strategic term rewriting

Как быстро создать прототип компилятора?

- **Набор встроенных DSL (eDSL)** для выразительного описания фаз компиляции.

Как быстро создать прототип компилятора?

- **Набор встроенных DSL (eDSL)** для выразительного описания фаз компиляции.
- **Язык** реализации **высокого уровня**, удобный для быстрой разработки небольших проектов.

Как быстро создать прототип компилятора?

- **Набор встроенных DSL (eDSL)** для выразительного описания фаз компиляции.
- **Язык** реализации **высокого уровня**, удобный для быстрой разработки небольших проектов.
- **Простые**, легко реализуемые **алгоритмы** компиляции.

Набор инструментов raddsl

Два eDSL (библиотеки комбинаторов) на языке Python.

- **parse.py**: лексический и синтаксический анализ, табличное описание операций с приоритетами (PEG, Pratt parser).
- **rewrite.py**: трансформации и порождение кода (миниатюрный вариант Stratego).

Менее 350 строк кода реализации.

“Halt! It's simulation time!”

```
"""
6
NOT 0
JZ 5 0
OR 0 1
NOT 0
JMP 1
HALT
"""
```


https://www.reddit.com/r/dailyprogrammer/comments/1euacb/052213_challenge_125_intermediate_halt_its/

“Halt! It's simulation time!”

```
"""
6
NOT 0
JZ 5 0
OR 0 1
NOT 0
JMP 1
HALT
"""
```

```
ws = many(space)
number = seq(ws, quote(some(digit)), to(1, int))
name = seq(ws, quote(some(letter)))
instr = group(name, many(number))
program = seq(number, many(instr))
```


https://www.reddit.com/r/dailyprogrammer/comments/1euacb/052213_challenge_125_intermediate_halt_its/

“Halt! It's simulation time!”

```
"""
6
NOT 0
JZ 5 0
OR 0 1
NOT 0
JMP 1
HALT
"""
```

```
ws = many(space)
number = seq(ws, quote(some(digit)), to(1, int))
name = seq(ws, quote(some(letter)))
instr = group(name, many(number))
program = seq(number, many(instr))
```

```
[
  6,
  ['NOT', 0],
  ['JZ', 5, 0],
  ['OR', 0, 1],
  ['NOT', 0],
  ['JMP', 1],
  ['HALT']
]
```


https://www.reddit.com/r/dailyprogrammer/comments/1euacb/052213_challenge_125_intermediate_halt_its/

JSON: лексический разбор

```
token = lambda tag: to(1, lambda x: (tag, x))
token_num = ...
OPERATORS = "[ { ] } : , false true null".split()
operator = seq(quote(match(OPERATORS)), token("op"))
int_part = alt(seq(range_of("1", "9"), many(digit)), a("0"))
frac = seq(a("."), some(digit))
exp = seq(one_of("eE"), opt(one_of("-+")), some(digit))
number = seq(quote(opt(a("-")), int_part, opt(frac), opt(exp)), token_num)
string = ...
token = seq(ws, alt(operator, string, number))
tokens = seq(many(token), ws, end)
```

JSON: лексический разбор

```
token = lambda tag: to(1, lambda x: (tag, x))
token_num = ...
OPERATORS = "[ { ] } : , false true null".split()
operator = seq(quote(match(OPERATORS)), token("op"))
int_part = alt(seq(range_of("1", "9"), many(digit)), a("0"))
frac = seq(a("."), some(digit))
exp = seq(one_of("eE"), opt(one_of("-+")), some(digit))
number = seq(quote(opt(a("-")), int_part, opt(frac), opt(exp)), token_num)
string = ...
token = seq(ws, alt(operator, string, number))
tokens = seq(many(token), ws, end)
```

JSON: лексический разбор

```
token = lambda tag: to(1, lambda x: (tag, x))
token_num = ...
OPERATORS = "[ { ] } : , false true null".split()
operator = seq(quote(match(OPERATORS)), token("op"))
int_part = alt(seq(range_of("1", "9"), many(digit)), a("0"))
frac = seq(a("."), some(digit))
exp = seq(one_of("eE"), opt(one_of("-+")), some(digit))
number = seq(quote(opt(a("-")), int_part, opt(frac), opt(exp)), token_num)
string = ...
token = seq(ws, alt(operator, string, number))
tokens = seq(many(token), ws, end)
```

JSON: лексический разбор

```
token = lambda tag: to(1, lambda x: (tag, x))
token_num = ...
OPERATORS = "[ { ] } : , false true null".split()
operator = seq(quote(match(OPERATORS)), token("op"))
int_part = alt(seq(range_of("1", "9"), many(digit)), a("0"))
frac = seq(a("."), some(digit))
exp = seq(one_of("eE"), opt(one_of("-+")), some(digit))
number = seq(quote(opt(a("-")), int_part, opt(frac), opt(exp)), token_num)
string = ...
token = seq(ws, alt(operator, string, number))
tokens = seq(many(token), ws, end)
```

JSON: синтаксический разбор

```
op = lambda o: eat(lambda x: x == ("op", o))
tok = ...
true = seq(op("true"), to(0, lambda: True))
false = ...
null = ...
array = group(op("["), opt(list_of(value, op(","))), op("]"))
member = group(tok("str"), op(":"), value)
obj = seq(op("{"), group(opt(list_of(member, op(",")))), op("}"), to(1, dict))
value = alt(tok("num"), tok("str"), true, false, null, obj, array)
json = alt(obj, array)
```

JSON: синтаксический разбор

```
op = lambda o: eat(lambda x: x == ("op", o))
tok = ...
true = seq(op("true"), to(0, lambda: True))
false = ...
null = ...
array = group(op("["), opt(list_of(value, op(","))), op("]"))
member = group(tok("str"), op(":"), value)
obj = seq(op("{"), group(opt(list_of(member, op(",")))), op("}"), to(1, dict))
value = alt(tok("num"), tok("str"), true, false, null, obj, array)
json = alt(obj, array)
```

JSON: синтаксический разбор

```
op = lambda o: eat(lambda x: x == ("op", o))
tok = ...
true = seq(op("true"), to(0, lambda: True))
false = ...
null = ...
array = group(op("["), opt(list_of(value, op(","))), op("]"))
member = group(tok("str"), op(":"), value)
obj = seq(op("{"), group(opt(list_of(member, op(",")))), op("}"), to(1, dict))
value = alt(tok("num"), tok("str"), true, false, null, obj, array)
json = alt(obj, array)
```

JSON: синтаксический разбор

```
op = lambda o: eat(lambda x: x == ("op", o))
tok = ...
true = seq(op("true"), to(0, lambda: True))
false = ...
null = ...
array = group(op("["), opt(list_of(value, op(","))), op("]"))
member = group(tok("str"), op(":"), value)
obj = seq(op("{"), group(opt(list_of(member, op(",")))), op("}"), to(1, dict))
value = alt(tok("num"), tok("str"), true, false, null, obj, array)
json = alt(obj, array)
```

JSON: сравнение производительности

Тестовый файл twitter.json (631 Кбайт)

raddsl

```
>>> timeit(raddsl_scan, number=4)
4.21377895557
>>> timeit(raddsl_parse, number=4)
0.630117927753
```

funcparserlib

```
>>> timeit(funcparserlib_scan, number=4)
1.33097530149
>>> timeit(funcparserlib_parse, number=4)
2.08991119573
```

Лексический разбор в funcparserlib реализован с помощью стандартной библиотеки re.

Устройство на базе недорогой FPGA-микросхемы

- Считывание данных с различных датчиков.
- Управление множеством вентиляторов и ALED-лент.

DSA для простых задач реального времени

Специализированное софт-ядро (автор И.Е. Тарасов) для FPGA:

- 32-битная стековая архитектура.
- Многопоточность.
- 8-16 Кбайт памяти кода и данных.

DSL-компилятор “Уж”

Входной язык — компилируемое подмножество Python.

DSA и DSL-компилятор разрабатывались совместно.

Размер компилятора: **1000** строк.

Стадии компиляции

Результат работы стандартной библиотеки ast

```
def fact(n):
  r = 1
  while n > 1:
    r *= n
    n -= 1
  return r
```

```
[
  Assign(targets=[Name(id='r', ctx=Store())], value=Num(n=1)),
  While(test=Compare(left=Name(id='n', ctx=Load()),
      ops=[Gt()], comparators=[Num(n=1)]), body=[
    AugAssign(target=Name(id='r', ctx=Store()), op=Mult(),
      value=Name(id='n', ctx=Load())),
    AugAssign(target=Name(id='n', ctx=Store()), op=Sub(),
      value=Num(n=1))
  ], orelse=[]),
  Return(value=Name(id='r', ctx=Load()))
]
```

Термы для всех видов IR

```
def fact(n):
  r = 1
  while n > 1:
    r *= n
    n -= 1
  return r
```

```
[
  (Assign(Id('r'), Int(1)),
  While(
    BinOp('>', (Id('n'), Int(1)),
    [
      Assign(Id('r'), BinOp('*', Id('r'), Id('n'))),
      Assign(Id('n'), BinOp('-', Id('n'), Int(1)))
    ]
  ),
  Return(Id('r'))
]
```

Упрощение выражений

```
fold_rules = alt(
  rule(BinOp("+", Int(X), Int(Y)), to(lambda v: Int(v.X + v.Y))),
  rule(BinOp("-", Int(X), Int(Y)), to(lambda v: Int(v.X - v.Y))),
  ...
  rule(BinOp(one_of("+ - | ^ << >>"), X, Int(0)), to(lambda v: v.X)),
  rule(BinOp(one_of("+ - | ^ << >>"), Int(0), X), to(lambda v: v.X)),
  rule(BinOp("*", X, Int(1)), to(lambda v: v.X)),
  rule(BinOp("*", Int(1), X), to(lambda v: v.X))
)

fold = bottomup(opt(fold_rules))
```

Упрощение выражений

```
fold_rules = alt(
  rule(BinOp("+", Int(X), Int(Y)), to(lambda v: Int(v.X + v.Y))),
  rule(BinOp("-", Int(X), Int(Y)), to(lambda v: Int(v.X - v.Y))),
  ...
  rule(BinOp(one_of("+ - | ^ << >>"), X, Int(0)), to(lambda v: v.X)),
  rule(BinOp(one_of("+ - | ^ << >>"), Int(0), X), to(lambda v: v.X)),
  rule(BinOp("*", X, Int(1)), to(lambda v: v.X)),
  rule(BinOp("*", Int(1), X), to(lambda v: v.X))
)

fold = bottomup(opt(fold_rules))
```

Упрощение выражений

```
fold_rules = alt(
  rule(BinOp("+", Int(X), Int(Y)), to(lambda v: Int(v.X + v.Y))),
  rule(BinOp("-", Int(X), Int(Y)), to(lambda v: Int(v.X - v.Y))),
  ...
  rule(BinOp(one_of("+ - | ^ << >>"), X, Int(0)), to(lambda v: v.X)),
  rule(BinOp(one_of("+ - | ^ << >>"), Int(0), X), to(lambda v: v.X)),
  rule(BinOp("*", X, Int(1)), to(lambda v: v.X)),
  rule(BinOp("*", Int(1), X), to(lambda v: v.X))
)

fold = bottomup(opt(fold_rules))
```

Замена **for** на **while**

```
for_rule = rule(
  For(Id(X), Call(Id("range"), [Y]), Z), to(lambda v: [
    Assign(Id(v.X), Int(0)),
    While(BinOp("<", Id(v.X), v.Y), v.Z + [
      Assign(Id(v.X), BinOp("+", Id(v.X), Int(1)))])
  ])
)

trans_for = topdown(opt(for_rule))
```

Замена **for** на **while**

```
for_rule = rule(
  For(Id(X), Call(Id("range"), [Y]), Z), to(lambda v: [
    Assign(Id(v.X), Int(0)),
    While(BinOp("<", Id(v.X), v.Y), v.Z + [
      Assign(Id(v.X), BinOp("+", Id(v.X), Int(1)))])
  ])
)

trans_for = topdown(opt(for_rule))
```

Замена **for** на **while**

```
for_rule = rule(
  For(Id(X), Call(Id("range"), [Y]), Z), to(lambda v: [
    Assign(Id(v.X), Int(0)),
    While(BinOp("<", Id(v.X), v.Y), v.Z + [
      Assign(Id(v.X), BinOp("+", Id(v.X), Int(1)))])
  ])
)

trans_for = topdown(opt(for_rule))
```

Порождение стекового IR

```
while_stmt = rule(
  While(let(X=expr), let(Y=block)),
  let(L1=label), let(L2=label), to(lambda v: [
    Label(v.L1),
    v.X,
    GotoIf0(v.L2),
    v.Y,
    Goto(v.L1),
    Label(v.L2)
  ])
)
```

Порождение стекового IR

```
while_stmt = rule(
  While(let(X=expr), let(Y=block)),
  let(L1=label), let(L2=label), to(lambda v: [
    Label(v.L1),
    v.X,
    GotoIf0(v.L2),
    v.Y,
    Goto(v.L1),
    Label(v.L2)
  ])
)
```

Peephole-оптимизация

```
opt_rules = alt(
  rule(peep([X, Push(Int(0)), BinOp("+")]), to(lambda v: [v.X])),
  rule(peep([Nop()]), to(lambda v: []))
)
```

Пример программы

```
@host
def make_sine_table(size):
  return [127 * math.sin(2 * math.pi * i / size) + 127 for i in range(size)]

sine_table = make_sine_table(32)

buf = [0, 0, 0] * 54
```

Пример программы

```
def main():
  phase = 0
  ws_set_port(0)
  while True:
    for i in range(len(buf)):
      buf[i * 3] = sine_table[(i + phase) & (len(sine_table) - 1)]
      ws_send_buf(buf, len(buf))
    phase += 1
    delay(100000)
```

Пример компиляции

```
def main():
  phase = 0
  ws_set_port(0)
  while True:
    for i in range(len(buf)):
      buf[i * 3] = sine_table[(i + phase) & (len(sine_table) - 1)]
      ws_send_buf(buf, len(buf))
    phase += 1
    delay(100000)
```

```
...
Func('main'),
Push(Int(0)),
Push(Local(0)),
Store(),
Push(Int(0)),
Push(Global('ws_set_port')),
Call(),
Label(12),
Push(Int(1)),
GotoIf0(13),
Push(Int(0)),
Push(Local(1)),
Store(),
Label(10),
Push(Local(1)),
Load(),
Push(Int(162)),
BinOp('<'),
GotoIf0(11),
Push(Global('sine_table')),
Push(Local(1)),
...
```

Вопросы?

https://github.com/true-grue/raddsl