Moscow
2019

Dmitry Patsura @ovrweb

Writing a compiler for TypeScript on TypeScript on top of LLVM

Dmitry Patsura @ovrweb

Пишу на много чем)

https://github.com/ovr

GHubber

PHPSA

StaticScript

#безызвестный

JS

Почему нет комилятора JS/TS в нативный код?

Можно ли написать компилятор TypeScript?

Теоретически

Эмпирически

This lecture is about:

2 questions = 2 answers = 2 parts

Why JS is using VM (Interpreter + JIT) instead of AOT compiler?

» Translators (interpretators/compilers/jit compilers)
» ByteCode (intermediate representation)
» Virtual Machine
» JIT Compilers

But what about compiler for TypeScript?

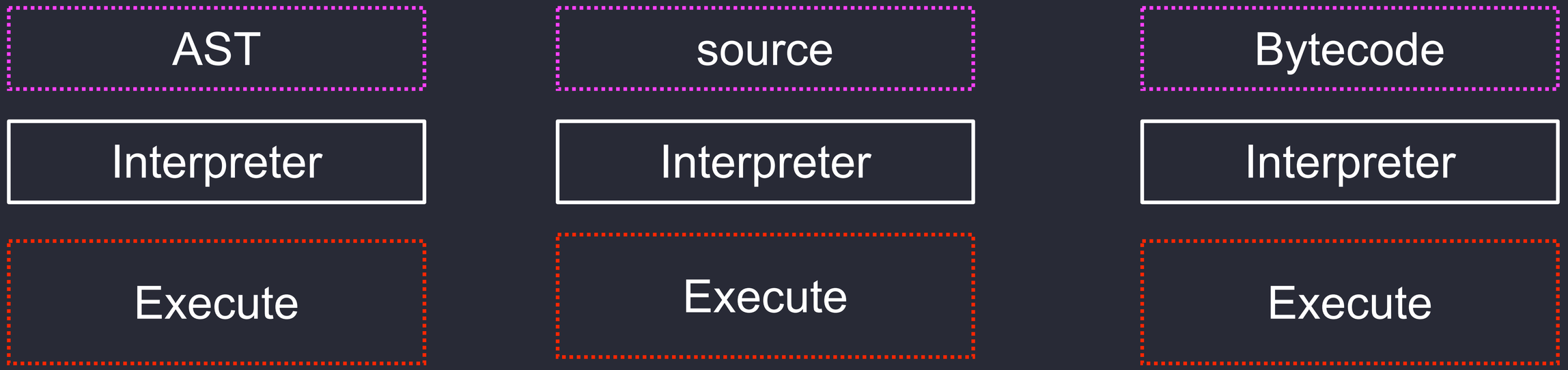
» TypeScript as Frontend
» LLVM
» Type System
» Object Type / Classes
» Array Type
» Branches
» Runtime Library

Translator
Interpreter vs Compiler vs Compiler JIT

Translator

is a generic term that could refer to a compiler, assembler, or interpreter; anything that converts code from one language into another.

Interpreter



Main task: Be fast

Minuses of Interpreter

» Small number of optimizations
» Hot code problem (can be resolved with JIT)
» Program is slower then native program
» Program cannot be executed without Interpreter

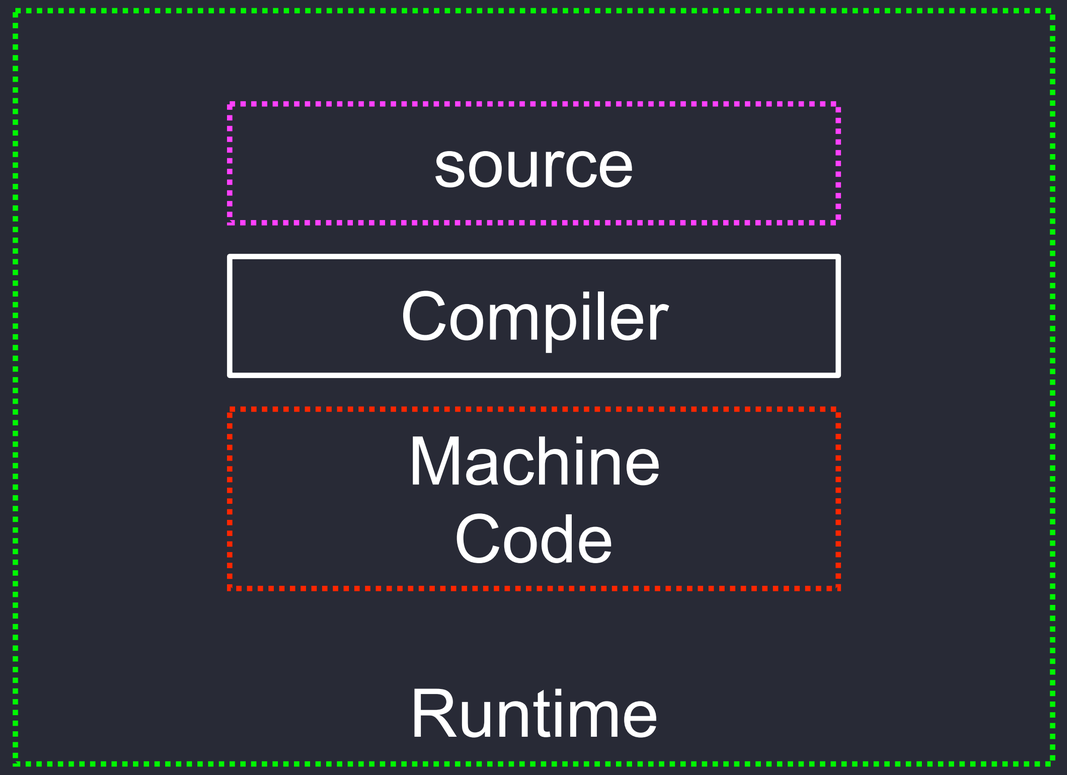
Compiler

Main task: effective translation

Minuses of Compiler

» Not JIT optimizations (without PGO)
» ~~Memory/Time~~

JIT Compiler

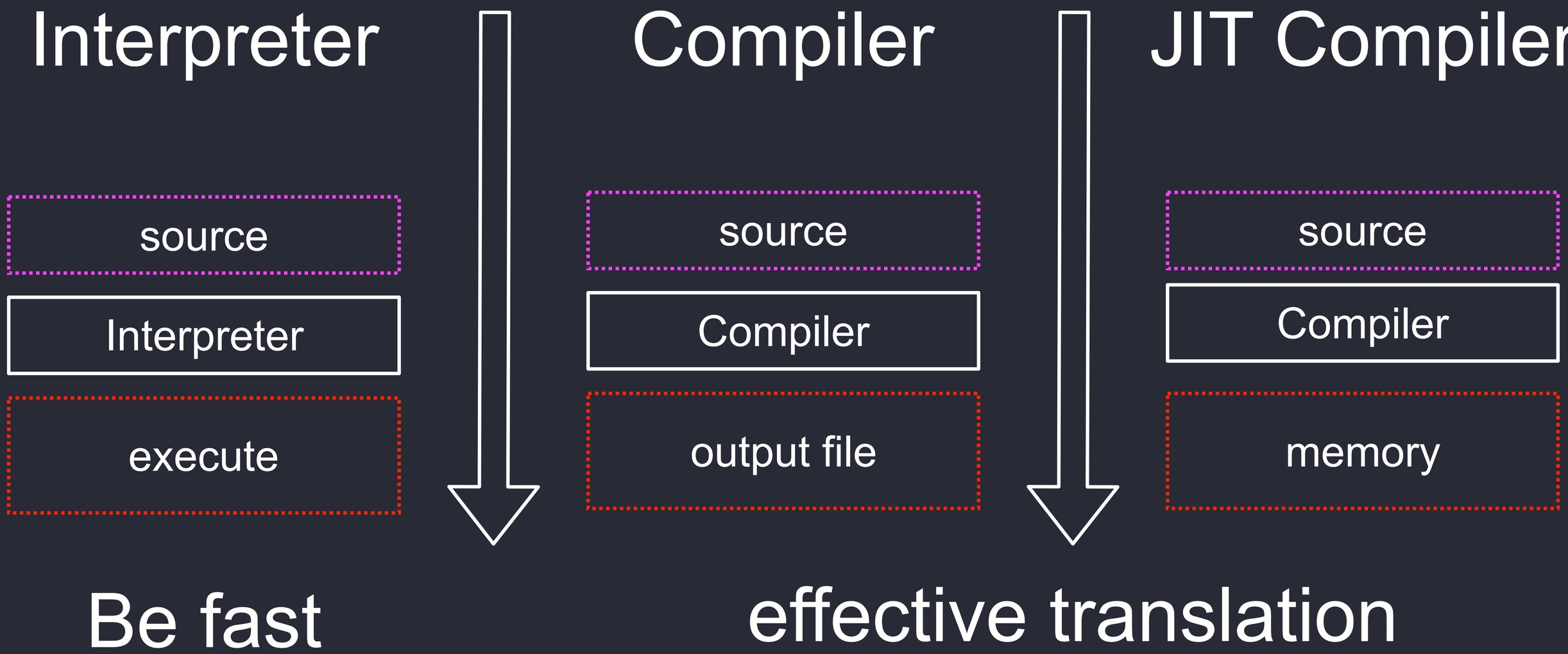


Main task: effective translation

Minuses of JIT Compiler

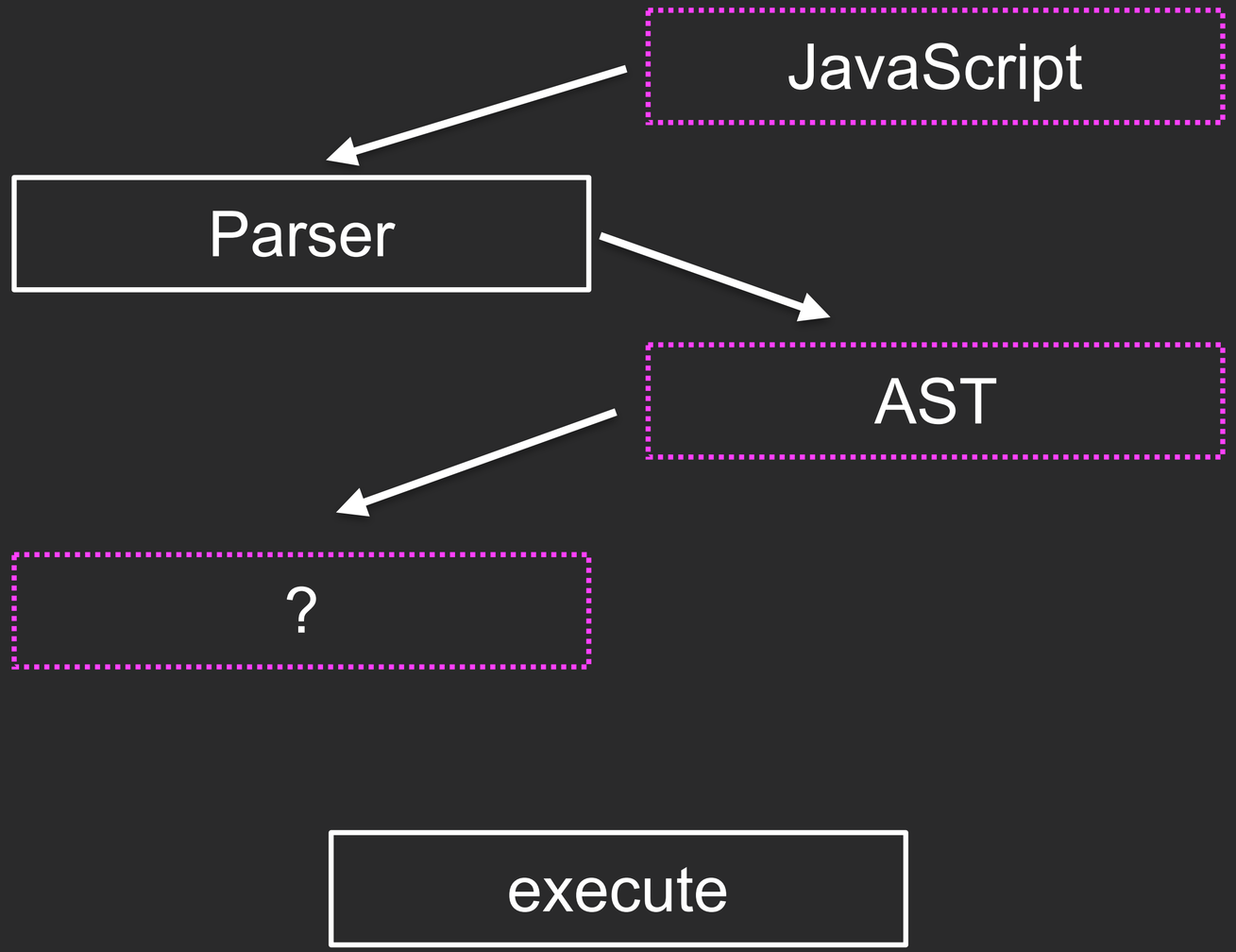
» Security (Executable space protection / W^X)
» Program cannot be executed without compiling step
» More memory usage for compilation

https://en.wikipedia.org/wiki/Executable_space_protection


Interpreter
Compiler
JIT Compiler
source
Interpreter
execute
source
Compiler
output file
source
Compiler
memory
Be fast
effective translation

ByteCode

Why ByteCode is needed?



ByteCode = Intermediate Representation

ASM < IR < JS

- » Should be portable (hardware free)
- » Effective to store
- » Should be simple
- » Easy SSA?
- » Easy CFG?

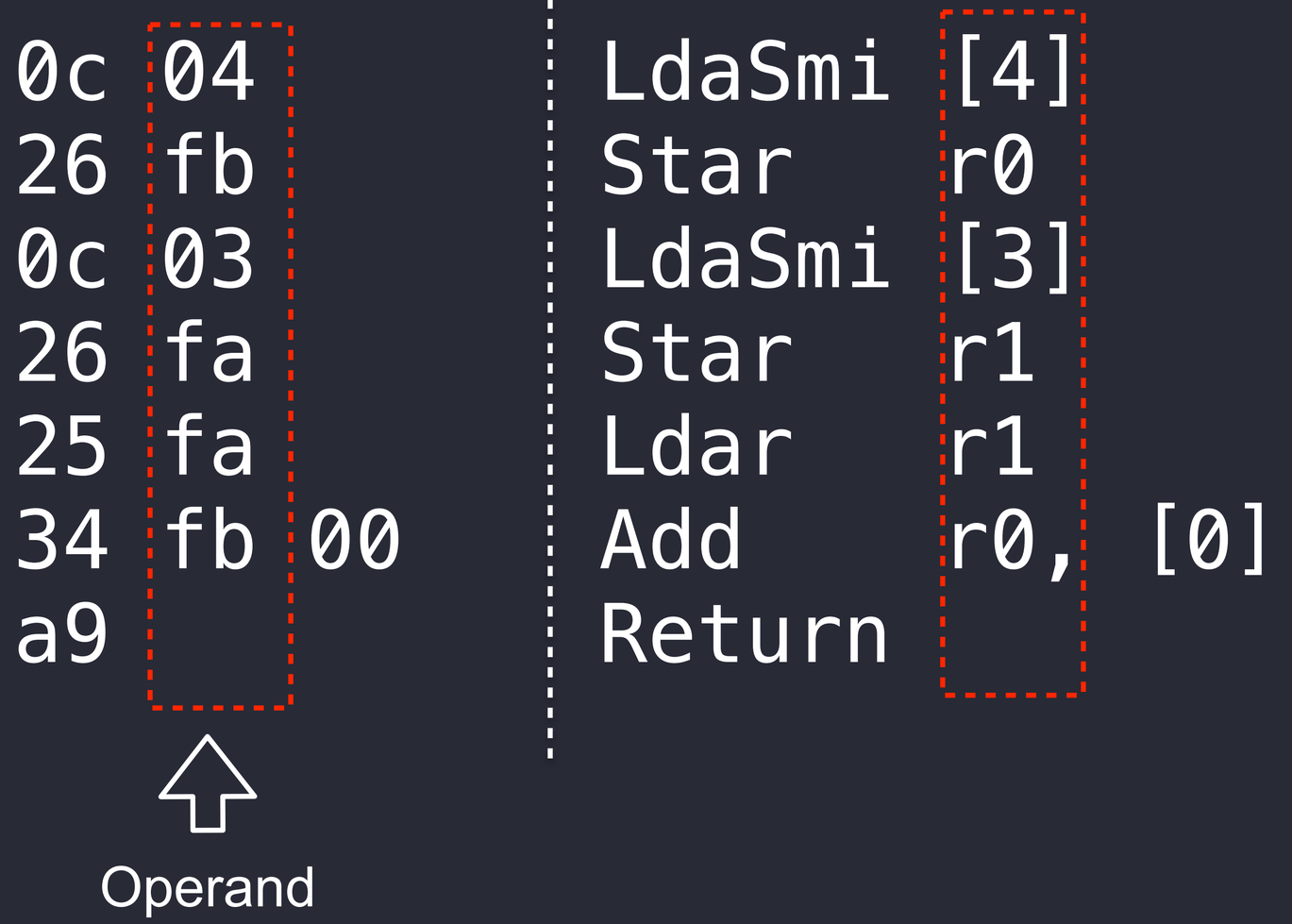
V8 ByteCode Example

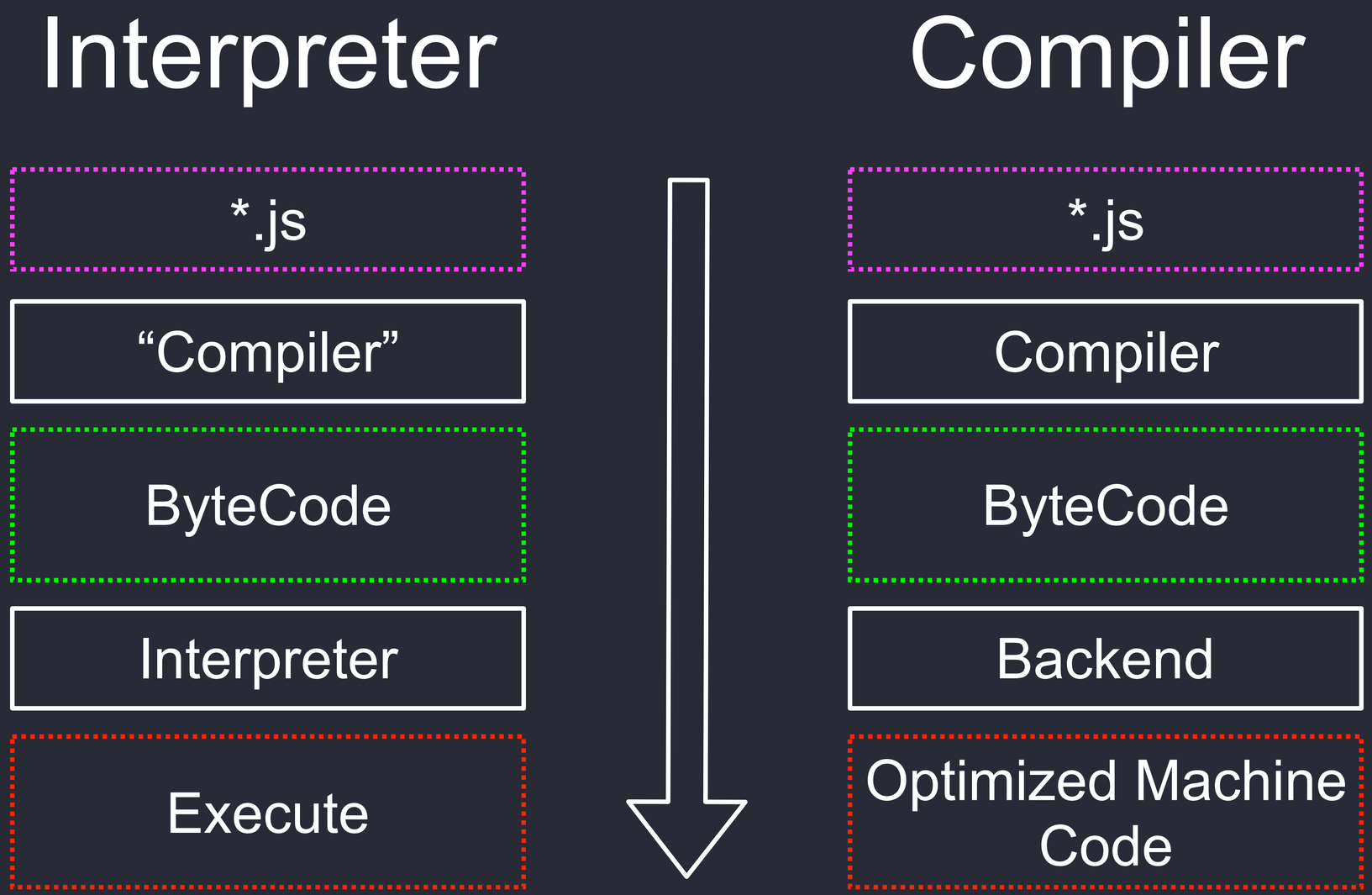
```
0c 04
26 fb
0c 03
26 fa
25 fa
34 fb 00
a9
```

OpCode Number (byte) [0, 255]
(00…FF)

```
LdaSmi [4]
Star   r0
LdaSmi [3]
Star   r1
Ldar   r1
Add    r0, [0]
Return
```

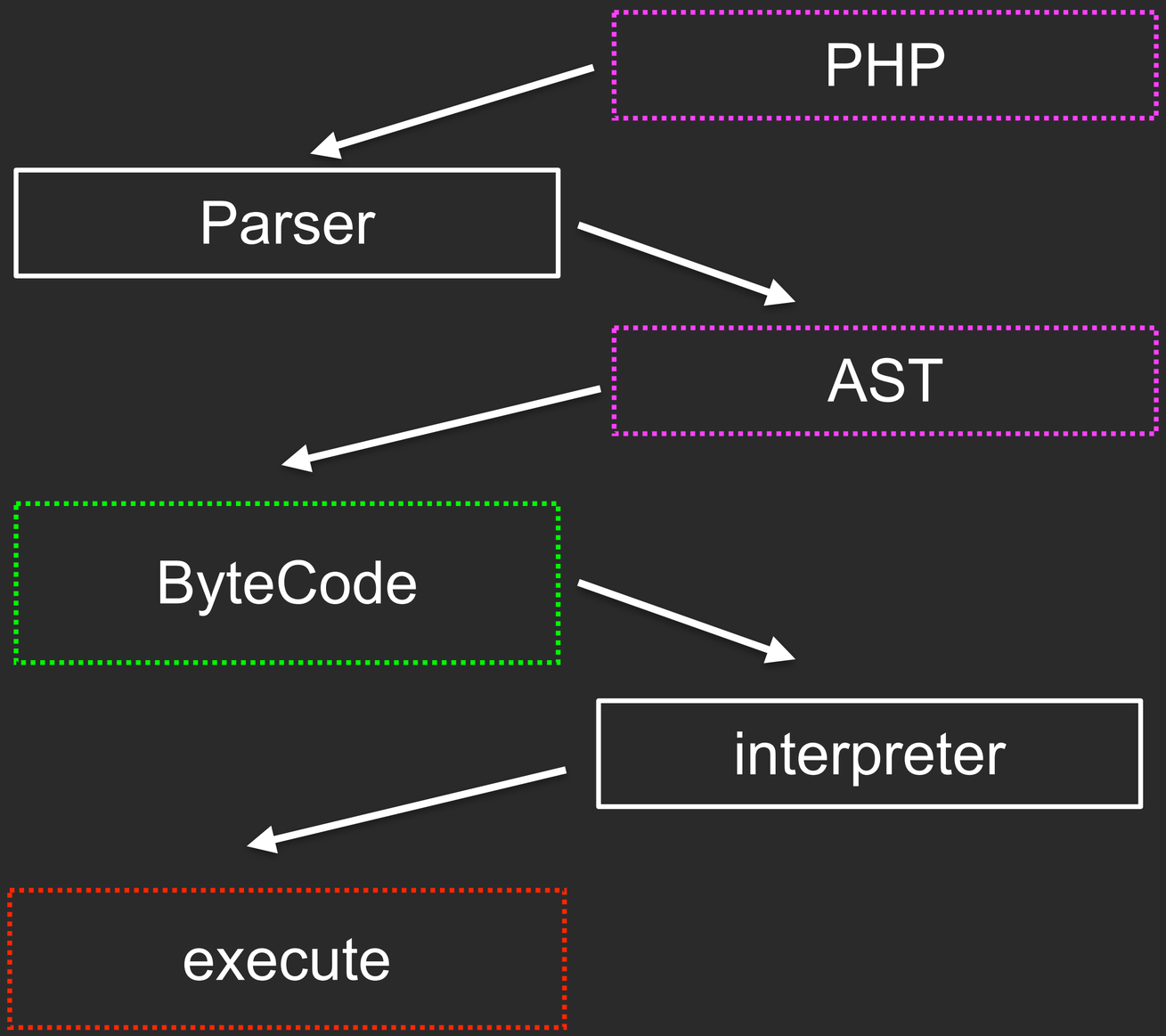
V8 ByteCode Example



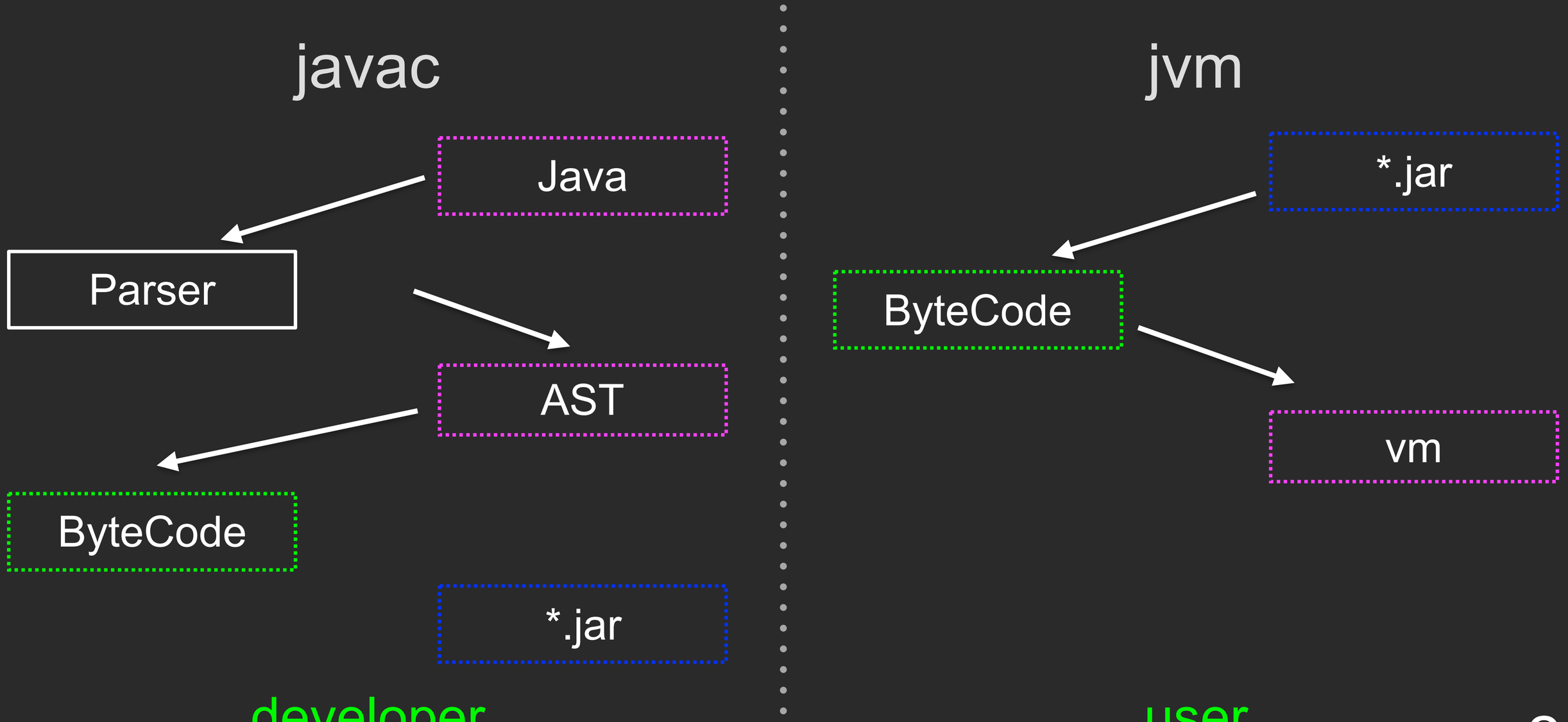

Interpreter
*.js
“Compiler”
ByteCode
Interpreter
Execute
Compiler
*.js
Compiler
ByteCode
Backend
Optimized Machine Code

Bytecode
Can be used differently

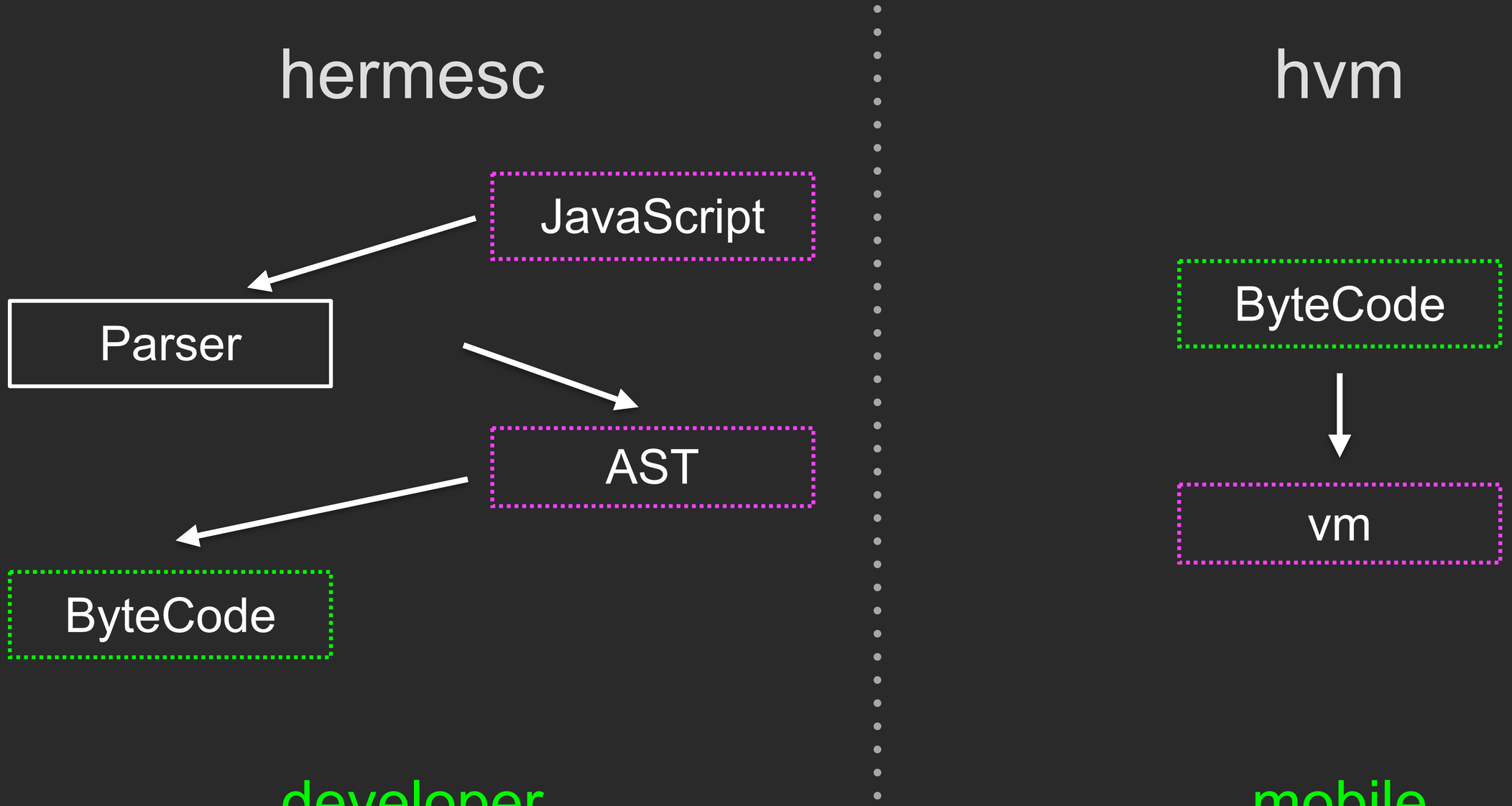
PHP



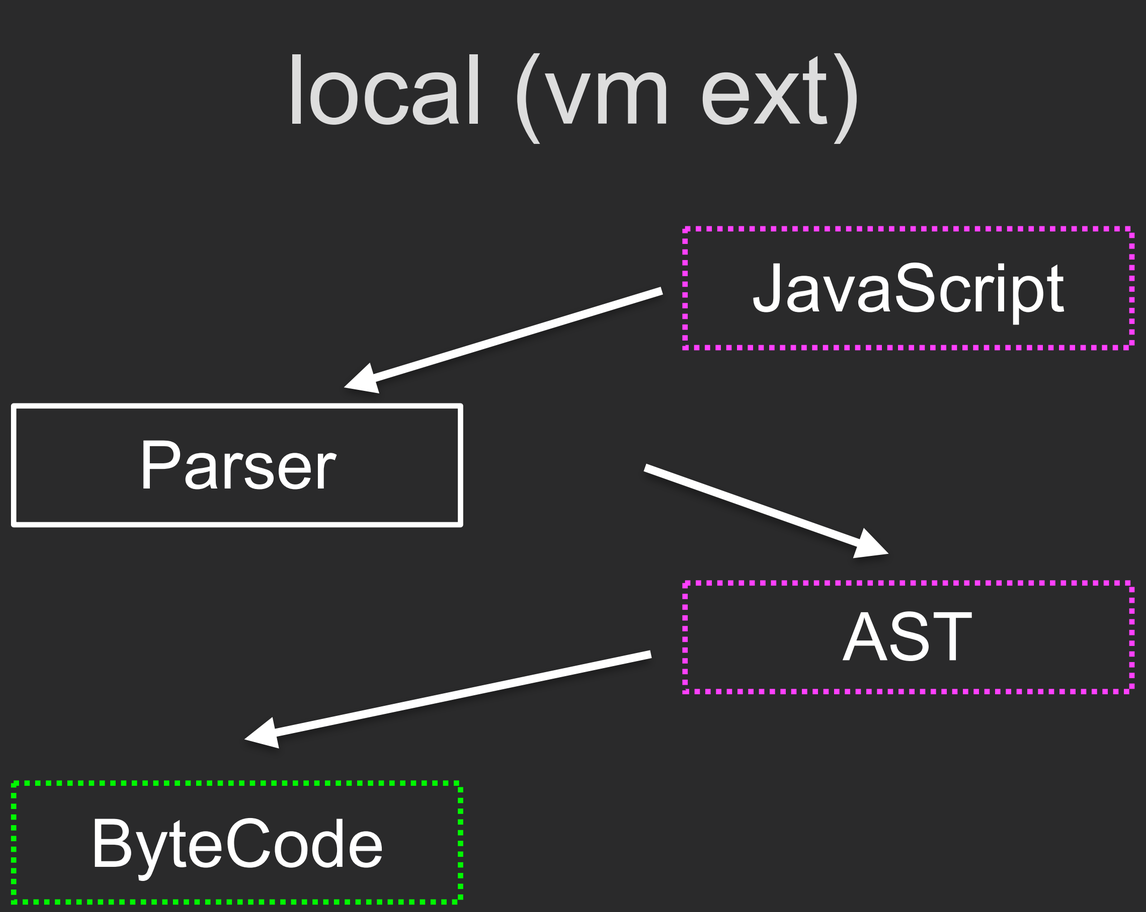
Java

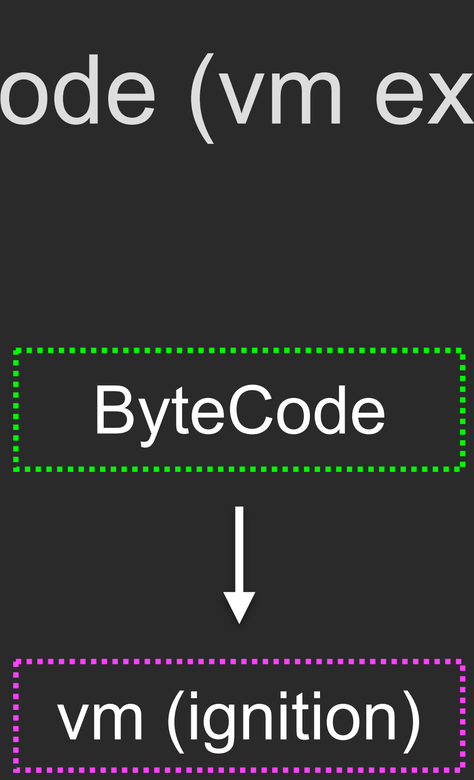

javac
Java
Parser
AST
ByteCode
*.jar
jvm
*.jar
ByteCode
vm
developer
user

Facebook Hermes (JS engine)



NodeJS?





developer

ByteNode

docker image*

V8

JavaScript

Parser

AST

Interpreter
Ignition

Compiler
TurboFan

ByteCode

Optimized Machine
Code

Virtual Machine

VirtualMachine

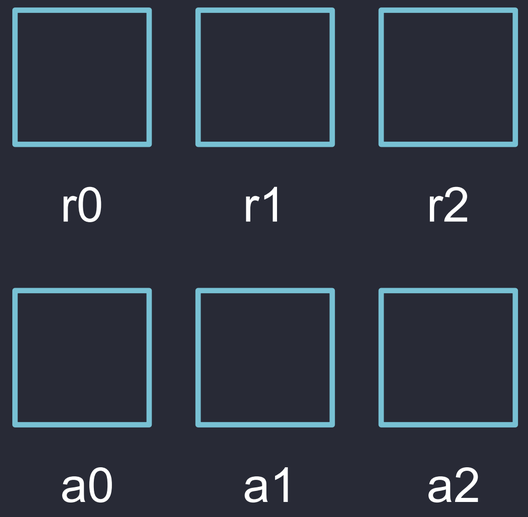
Process virtual machines are designed to execute computer programs in a platform-independent environment.

VirtualMachine

Portable Executable	OS	JS
VM	VM	VM
Execute	Execute	Execute

Virtual Machine Types

Register based




```
Example 3 + 4
```

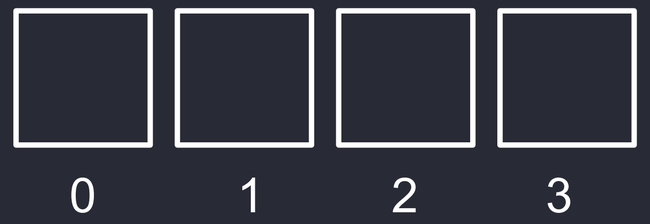
0x00001 SET r0, 3

0x00002 SET r1, 4

0x00003 ADD r1, r0

Stack based

Accumulator
(stack FILO)



“Infinity” Registers


```
Example 3 + 4
```

0x00001 LOAD 3

0x00002 PUSH r0

0x00003 LOAD 4

0x00004 ADD r0

Compare

» Closer to Hardware
» Harder to implement
» Less byte code

Register based

» Architecture Abstract
» Easier to implement

Stack based

Virtual Machine
V8 Interperter ByteCode

```
function test() {
    const a = 4;
    const b = 3;

    return a + b;
}

test();
```

node --print-bytecode index.js > 1.stdout

```
[generated bytecode for function: test]
Parameter count 1
Register count 2
Frame size 16
   39 E> 0x3d8cc7e0556 @    0 : a5          StackCheck
   58 S> 0x3d8cc7e0557 @    1 : 0c 04       LdaSmi [4]
         0x3d8cc7e0559 @    3 : 26 fb       Star r0
   75 S> 0x3d8cc7e055b @    5 : 0c 03       LdaSmi [3]
         0x3d8cc7e055d @    7 : 26 fa       Star r1
   83 S> 0x3d8cc7e055f @    9 : 25 fa       Ldar r1
   92 E> 0x3d8cc7e0561 @   11 : 34 fb 00    Add r0, [0]
   96 S> 0x3d8cc7e0564 @   14 : a9          Return
Constant pool (size = 0)
Handler Table (size = 0)
```

https://github.com/v8/v8/blob/master/src/interpreter/bytecodes.h

```
/* Loading the accumulator */                                   \
V(LdaZero, AccumulatorUse::kWrite)                              \
V(LdaSmi, AccumulatorUse::kWrite, OperandType::kImm)            \
V(LdaUndefined, AccumulatorUse::kWrite)                         \
V(LdaNull, AccumulatorUse::kWrite)                              \
V(LdaTheHole, AccumulatorUse::kWrite)                           \
/* Globals */                                                   \
V(LdaGlobal, AccumulatorUse::kWrite, OperandType::kIdx, …)      \
V(LdaGlobalInsideTypeof, AccumulatorUse::kWrite, …)             \
V(StaGlobal, AccumulatorUse::kRead, …)                          \
/* Context operations */                                        \
V(PushContext, AccumulatorUse::kRead, OperandType::kRegOut)     \
V(PopContext, AccumulatorUse::kNone, OperandType::kReg)         \
V(LdaContextSlot, AccumulatorUse::kWrite, OperandType::kReg,    \
```

```
// LdaZero
//
// Load literal '0' into the accumulator.
IGNITION_HANDLER(LdaZero, InterpreterAssembler) {
  TNode<Number> zero_value = NumberConstant(0.0);
  SetAccumulator(zero_value);
  Dispatch();
}

// LdaSmi <imm>
//
// Load an integer literal into the accumulator as a Smi.
IGNITION_HANDLER(LdaSmi, InterpreterAssembler) {
  TNode<Smi> smi_int = BytecodeOperandImmSmi(0);
  SetAccumulator(smi_int);
  Dispatch();
}
```

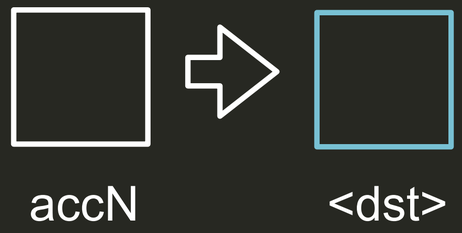
Virtual Machine
V8 ByteCode interpreter on JS

```
type Accumulator = any[];
type Registers = {[key: string]: any};

class VirtualMachine {
    protected acc: Accumulator = [];
    protected registers: Registers = {};
}
```

```
const a = 4;  0x3d8cc7e0557 LdaSmi [4]
              0x3d8cc7e0559 Star r0

const b = 3;  0x3d8cc7e055b LdaSmi [3]
              0x3d8cc7e055d Star r1

      a + b   0x3d8cc7e055f Ldar r1
              0x3d8cc7e0561 Add r0
```

```
0x3d8cc7e0557 LdaSmi [4]
0x3d8cc7e0559 Star r0
```

```
// Load an integer literal into the accumulator
LdaSmi: (op) => {
    this.acc.push(op.operand[0]);
},
// Store accumulator to register <dst>.
Star: (op) => {
    this.putToRegister(
        op.reg,
        this.acc.pop()
    );
},
```



```
// Load accumulator with value from register <src>.
Ldar: (op) => {
    this.acc.push(
        this.getFromRegister(op.reg)
    )
},
// Add register <src> to accumulator.
Add: (op) => {
    const right = this.getFromRegister(op.reg);

    this.acc.push(
        this.acc.pop() + right
    )
},
```

V8 ByteCode Demo 1

Entry:

```
let a = 0;
```

Condition:

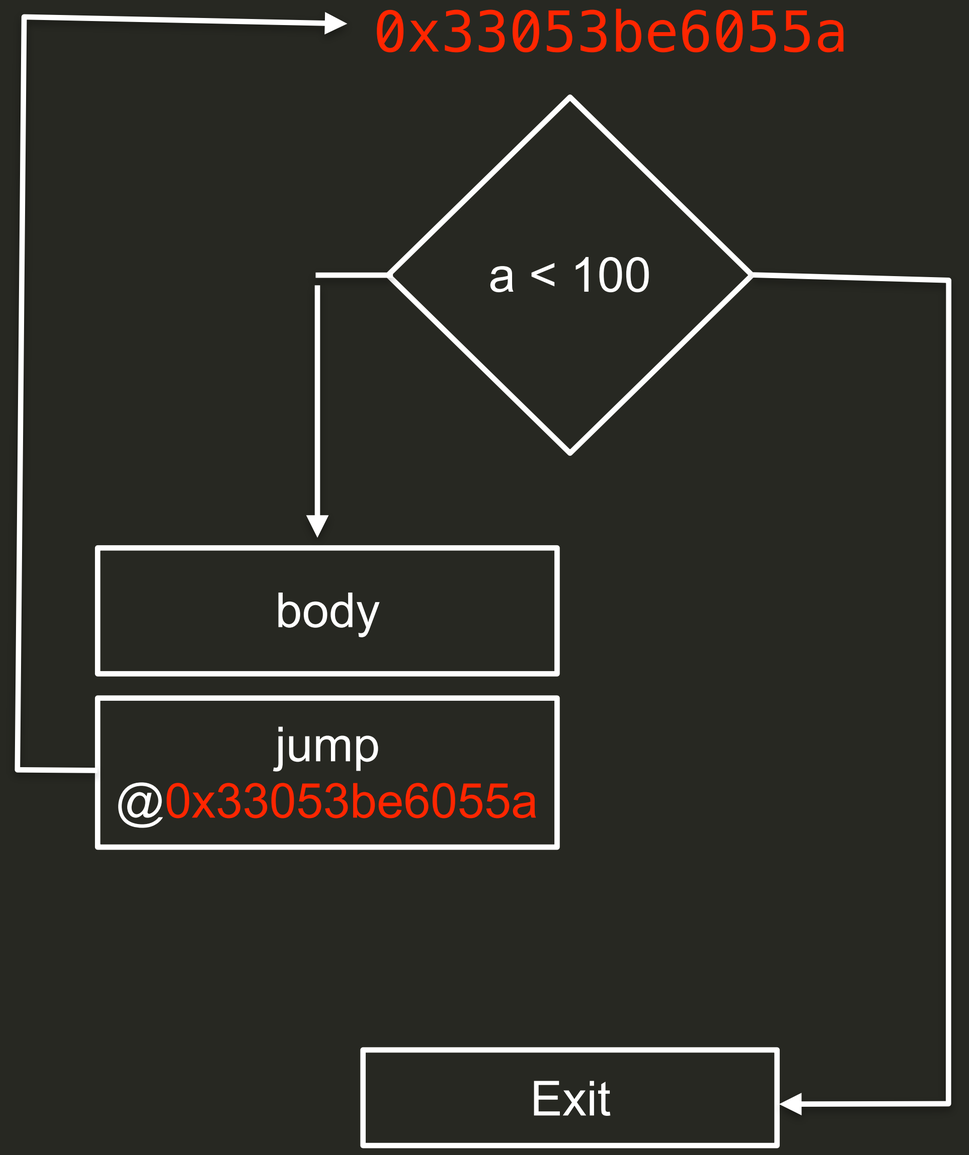
```
while (a < 100) {
```

Body:

```
        a++;
    }
```

Exit:

```
return a;
```



```
0x37a29c260556 StackCheck
0x37a29c260557 LdaZero
0x37a29c260558 Star r0    let a = 0;

0x37a29c26055a LdaSmi [100]
0x37a29c26055c TestLessThan r0        while (a < 100)
0x37a29c26055f JumpIfFalse [12] (0x37a29c26056b@ 21)

0x37a29c260561 StackCheck
0x37a29c260562 Ldar r0
0x37a29c260564 Inc [1]                    a++;
0x37a29c260566 Star r0
0x37a29c260568 JumpLoop [14] (0x37a29c26055a@ 4)

0x37a29c26056b Ldar r0
0x37a29c26056d Return     return a;
```

V8 ByteCode Demo 2

JIT Compiler
Machine Code

V8

JavaScript

Parser

AST

Interpreter
Ignition

Compiler
TurboFan

ByteCode

Optimized Machine
Code

d8 –code-comments –print-opt-code jit.js

```
function plus(obj) {
    return obj.x + obj.y;
}

console.log(plus({ x: 5, y: 5 }));
console.log(plus({ x: 5, y: 5 }));
console.log(plus({ x: 5, y: 5 }));
```

Interpreter
Ignition

Compiler
TurboFan

A = Time to Execute ByteCode
B = Compile function using JIT compiler
C = Time to Execute Machine ByteCode

$$A < B+C$$

@todo

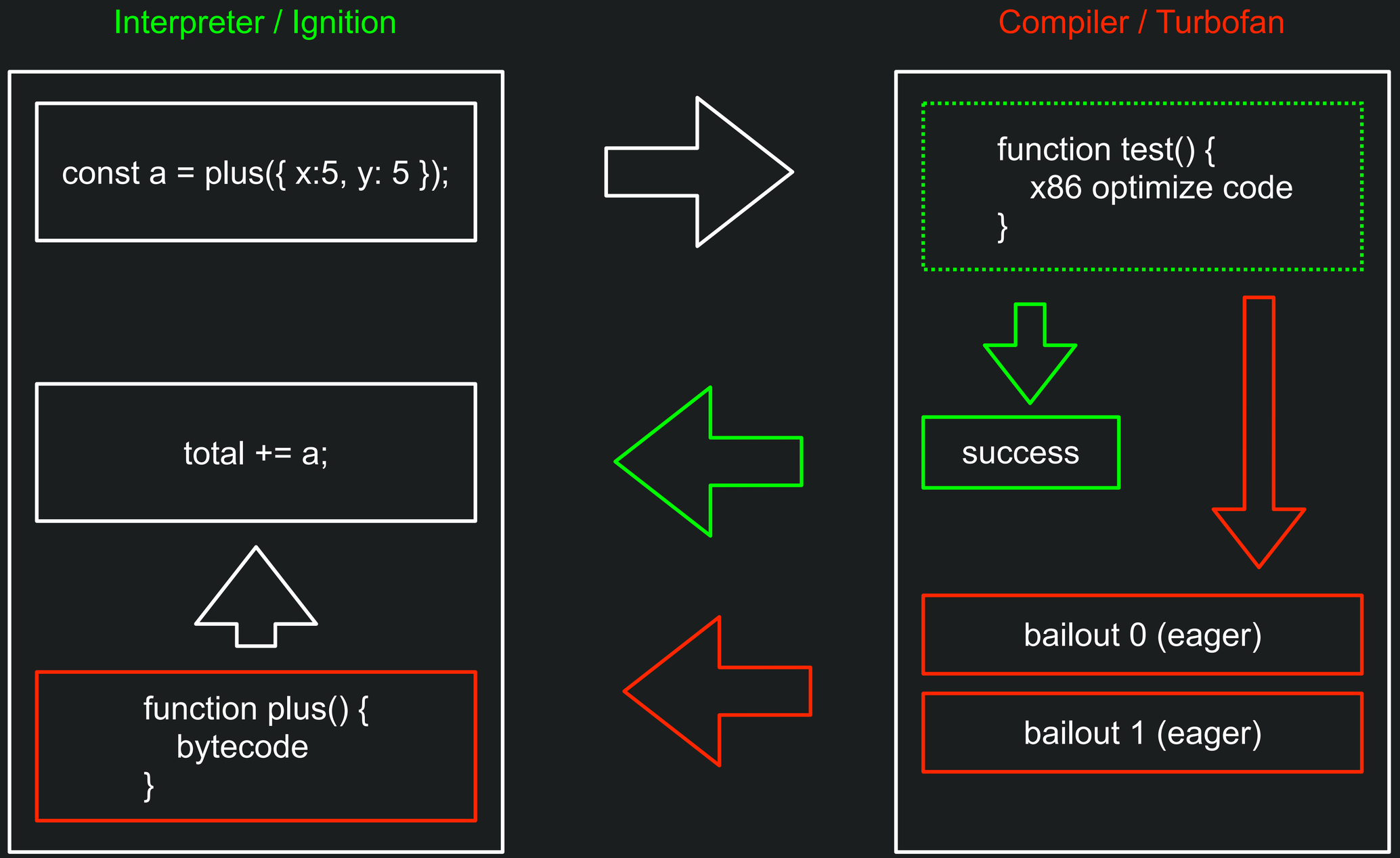
```
function plus(obj) {
    return obj.x + obj.y;
}

let total = 0;

while (total < 100000000) {
    total += plus({ x: 5, y: 5 });
}

console.log(total);
```

```
function plus(obj) {
    return obj.x + obj.y;
}

let total = 0;

while (total < 100000000) {
    total += plus({ x: 5, y: 5 });
}

console.log(total);
```

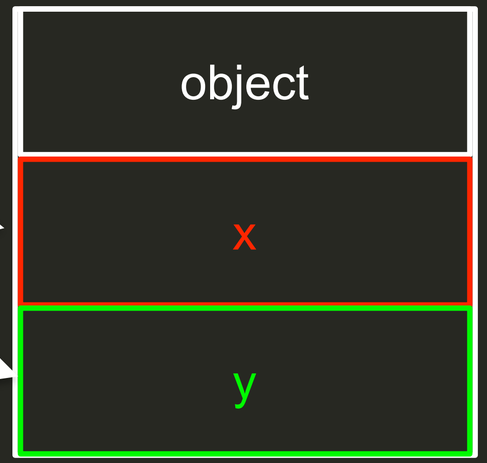
d8 —code-comments —print-opt-code jit.js

```
--- Raw source ---
(obj) {
    return obj.x + obj.y;
}


--- Optimized code ---
optimization_id = 0
source_position = 38
kind = OPTIMIZED_FUNCTION
name = plus
stack_slots = 5
compiler = turbofan
address = 0x3b34494c2ae1
```

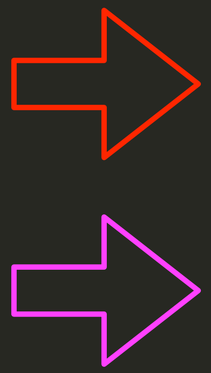
```
Instructions (size = 364)
                -- Prologue: check code start register --
0x3b34494c2b20     0  488d1df9ffffff REX.W leaq rbx,[rip+0xfffffff9]
0x3b34494c2b27     7  483bd9         REX.W cmpq rbx,rcx
0x3b34494c2b2a     a  7418           jz 0x3b34494c2b44  <+0x24>
                Abort message:
                Wrong value in code start register passed
0x3b34494c2b2c     c  48ba6c0000000000000 REX.W movq rdx,0x6c
                -- Inlined Trampoline to Abort --
0x3b34494c2b36    16  49ba207a660c01000000 REX.W movq r10,0x10c667a20  (Abort)    ;; off heap target
0x3b34494c2b40    20  41ffd2         call r10
0x3b34494c2b43    23  cc             int3l
                -- Prologue: check for deoptimization --
0x3b34494c2b44    24  488b59e0       REX.W movq rbx,[rcx-0x20]
0x3b34494c2b48    28  f6430f01       testb [rbx+0xf],0x1
0x3b34494c2b4c    2c  740d           jz 0x3b34494c2b5b  <+0x3b>
                -- Inlined Trampoline to CompileLazyDeoptimizedCode --
0x3b34494c2b4e    2e  49ba40795c0c01000000 REX.W movq r10,0x10c5c7940  (CompileLazyDeoptimizedCode)    ;; off heap target
0x3b34494c2b58    38  41ffe2         jmp r10
                -- B0 start (construct frame) --
0x3b34494c2b5b    3b  55             push rbp
0x3b34494c2b5c    3c  4889e5         REX.W movq rbp,rsp
0x3b34494c2b5f    3f  56             push rsi
0x3b34494c2b60    40  57             push rdi
0x3b34494c2b61    41  4883ec08       REX.W subq rsp,0x8
0x3b34494c2b65    45  488975e8       REX.W movq [rbp-0x18],rsi
0x3b34494c2b69    49  493b65e0       REX.W cmpq rsp,[r13-0x20] (external value (StackGuard::address_of_jslimit()))
0x3b34494c2b6d    4d  0f8668000000   jna 0x3b34494c2bdb  <+0xbb>
                -- B2 start --
                -- B3 start --
```

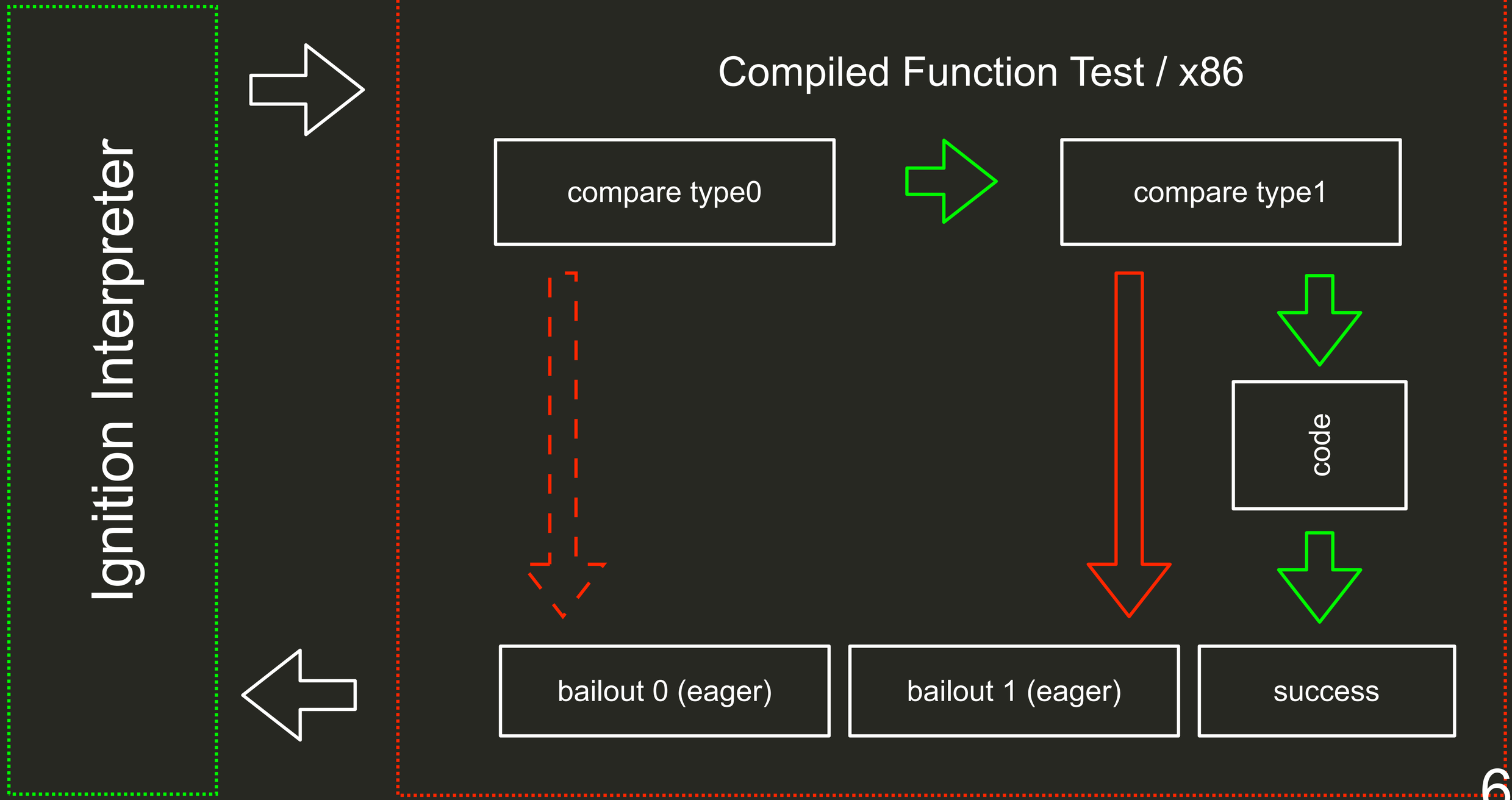

Interpreter / Ignition
Compiler / Turbofan
const a = plus({ x:5, y: 5 });
total += a;
function plus() {
bytecode
}
function test() {
x86 optimize code
}
success
bailout 0 (eager)
bailout 1 (eager)

```
                    -- B3 start -
53  REX.W movq rcx,[rbp+0x10]
57  testb rcx,0x1
5a  jz 0x3b34494c2c66  <+0x146>

60  REX.W movq rdi,0x3fb0727495e9    ;; object: 0x3fb0727495e9
6a  REX.W cmpq [rcx-0x1],rdi
e   jnz 0x3b34494c2c72  <+0x152>

74  REX.W movq rdi,[rcx+0x17]
78  REX.W movq rcx,[rcx+0x1f]
7c  sarl rcx, 1
07e sarl rdi, 1
80  addl rcx,rdi
82  REX.W movq r10,0x100000000
8c  REX.W cmpq r10,rcx
8f  jnc 0x3b34494c2bc6  <+0xa6>

-- A LOT OF HODE IS HERE-
```



```
                    -- B3 start --
53   REX.W movq rcx,[rbp+0x10]
57   testb rcx,0x1
5a   jz 0x3b34494c2c66  <+0x146>

60   REX.W movq rdi, x3fb0727495e9  ;; object: 0x3fb0727495e9
6a   REX.W cmpq [rcx-0x1],rdi
e    jnz 0x3b34494c2c72  <+0x152>

-- A LOT OF HODE IS HERE-

146  REX.W movq r13,0x0
14d  call 0x3b3449502040     ;; eager deoptimization bailout

152  REX.W movq r13,0x1
159  call 0x3b3449502040     ;; eager deoptimization bailout

15e  REX.W movq r13,0x2
165  call 0x3b3449542040     ;; lazy deoptimization bailout
```




Ignition Interpreter
Compiled Function Test / x86
compare type0
compare type1
code
bailout 0 (eager)
bailout 1 (eager)
success

Why JS is using VM (Interpreter + JIT compiler)
instead of AOT compiler?

usage area


node
JS
64

But what about compiler for TypeScript?

» Type System
» Add more special native types: uint8
» Dont support all JS features

StaticScript

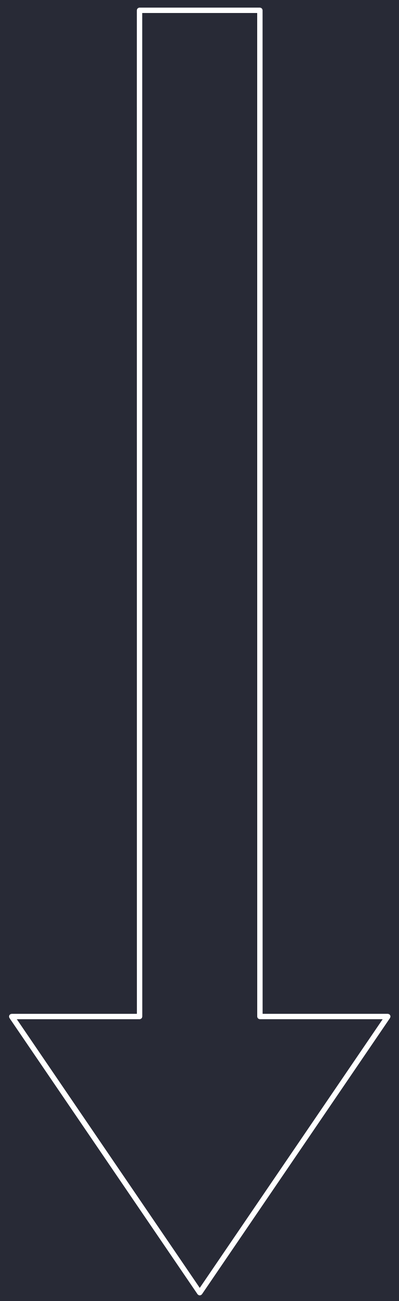
Compiler

FRONTEND

PARSER/AST/PRE-EMIT/
CLI

BACKEND

EMIT/GENERATE OUTPUT
CODE



OUTPUT

Compiler Frontend

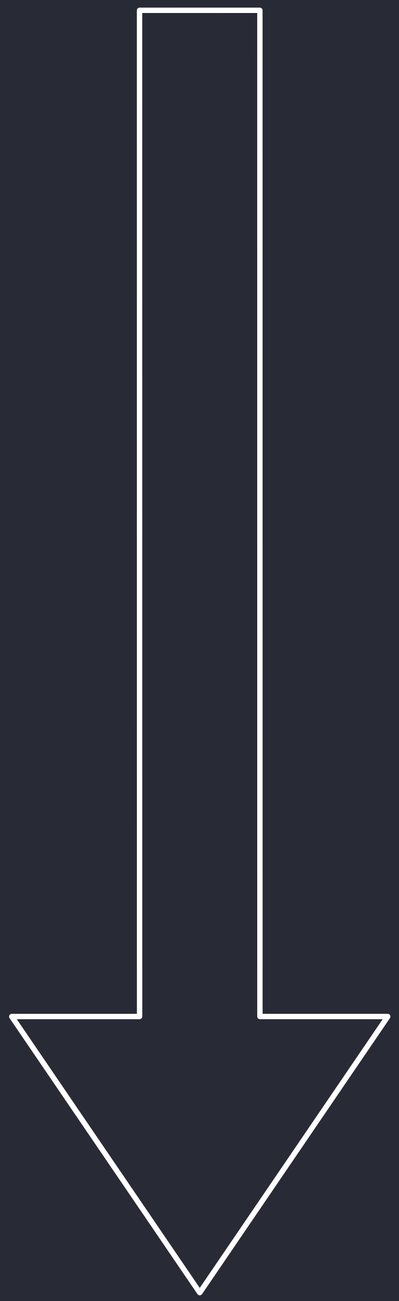
TypeScript as Frontend

FRONTEND

PARSER/AST/PRE-EMIT/
CLI

BACKEND

EMIT/GENERATE OUTPUT
CODE



The simplest problem

```
{
    console_log(“Hello HolyJS!”);
}
```

TypeScript as Framework

```
import * as ts from 'typescript';

const options = {
    lib: [],
    types: []
};

const files = [‘sandbox/hello.ts'];

const host = ts.createCompilerHost(options);
const program = ts.createProgram(files, options, host);
```

Use diagnostics from TypeScript

```
const diagnostics = ts.getPreEmitDiagnostics(program);
if (diagnostics.length) {
    ts.sys.write(
        ts.formatDiagnosticsWithColorAndContext(
            diagnostics,
            DiagnosticHostInstance
        )
    );
    ts.sys.exit(
        ts.ExitStatus.DiagnosticsPresent_OutputsSkipped
    );
}
```

```
ovr@MBP-Dmitry ~hlvm master node ./build/cli
Cannot find global type 'Array'.
Cannot find global type 'Boolean'.
Cannot find global type 'Function'.
Cannot find global type 'IArguments'.
Cannot find global type 'Number'.
Cannot find global type 'Object'.
Cannot find global type 'RegExp'.
Cannot find global type 'String'.
sandbox/do-simple-math.ts (3,5): Cannot find name ‘console_log'.
```

staticscript.d.ts

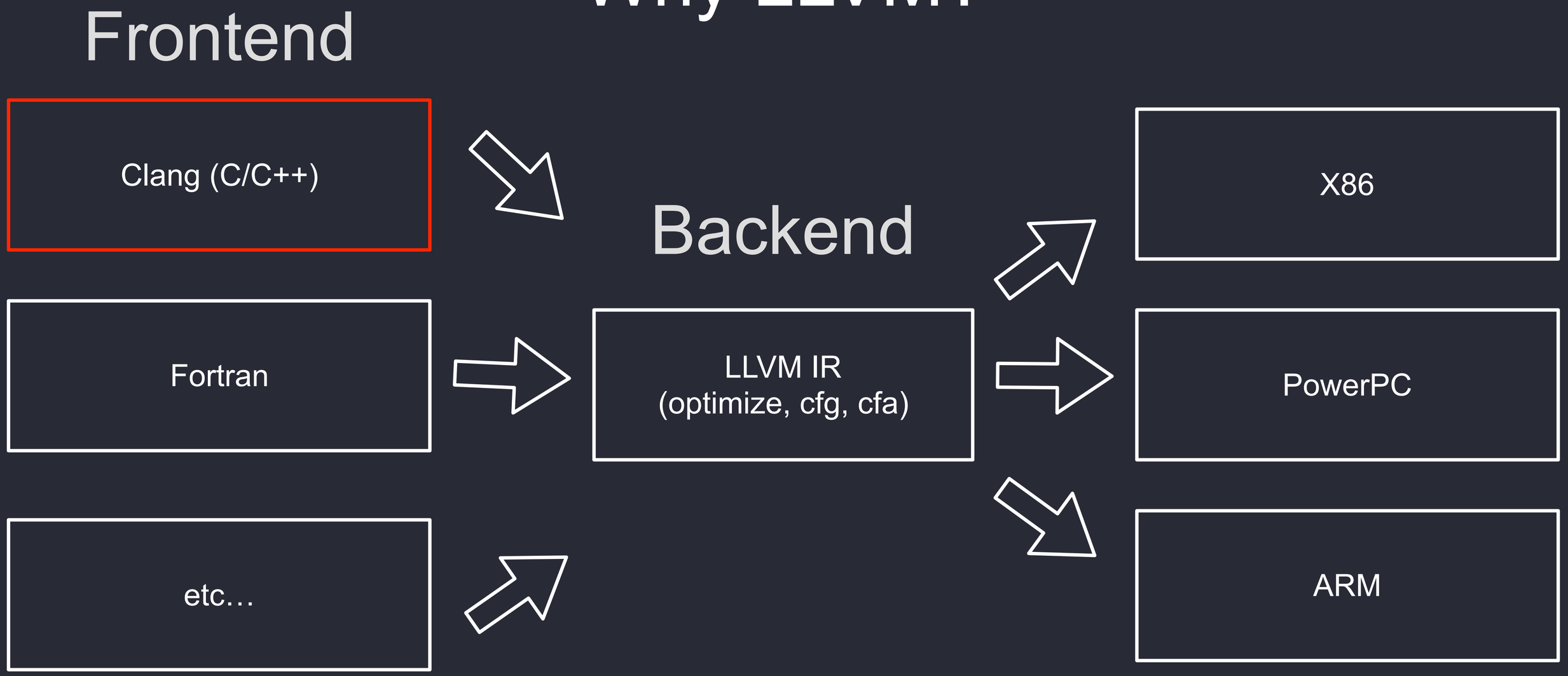
```
interface Boolean {}
interface Function {}
interface IArguments {}
interface Number {}
interface Object {}
interface RegExp {}
interface String {}
interface Array<T = any> {}
```

Compiler Backed
LLVM

LLVM - Low Level Virtual Machine

The LLVM Project is a collection of modular and reusable compiler and toolchain technologies. Despite its name, LLVM has little to do with traditional virtual machines. The name "LLVM" itself is not an acronym; it is the full name of the project.

Why LLVM?


Frontend
Clang (C/C++)
Fortran
etc…
Backend
LLVM IR
(optimize, cfg, cfa)
X86
PowerPC
ARM

LLVM Language Reference Manual

LLVM IR - LLVM intermediate representation

https://llvm.org/docs/LangRef.html

clang -S -emit-llvm main.cpp -O0

```cpp
#include <cstdio>

int main() {
    auto a = 3;
    auto b = 4;
    auto c = a + b;
}
```

```
; ModuleID = ‘main.cpp'
source_filename = "main.cpp"
target datalayout = "e-m:o-i64:64-f80:128-n8:16:32:64-S128"
target triple = “x86_64-apple-macosx10.14.0"

define i32 @main() #0 {
  %1 = alloca i32, align 4
  %2 = alloca i32, align 4
  %3 = alloca i32, align 4
  store i32 3, i32* %1, align 4
  store i32 4, i32* %2, align 4
  %4 = load i32, i32* %1, align 4
  %5 = load i32, i32* %2, align 4
  %6 = add nsw i32 %4, %5
  store i32 %6, i32* %3, align 4
  ret i32 0
}
```

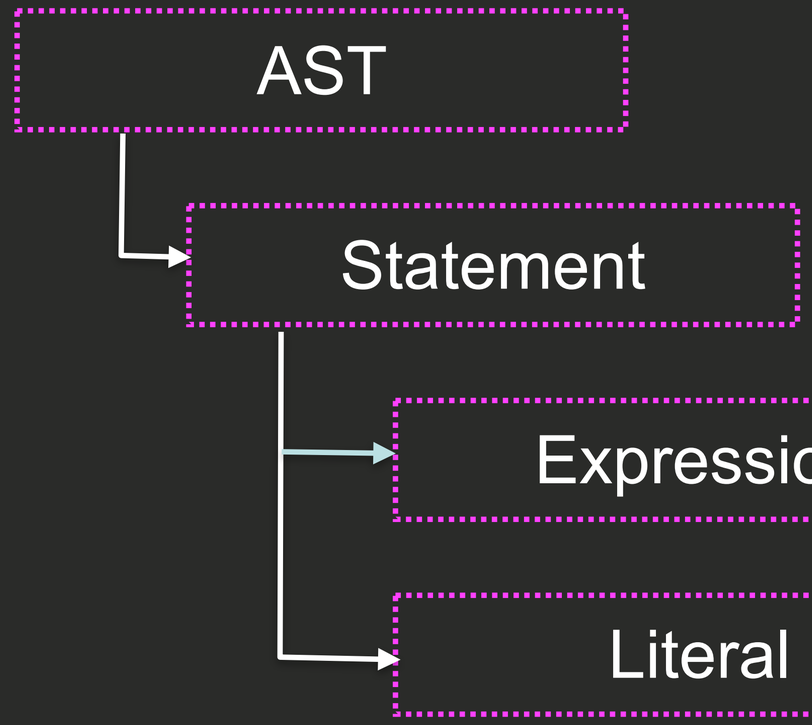
LLVM Types

- Integers, iN integer with any Number of bits (i1/i31/i999999) with signed bit
- Floating (half/float/double/fp128/x86_fp80/ppc_fp128)
- Pointers (i32*)
- Aggregate types: Vector (4 x i32), Array ([40 x i32]), Structure (i32, i32, i32)
- Opaque

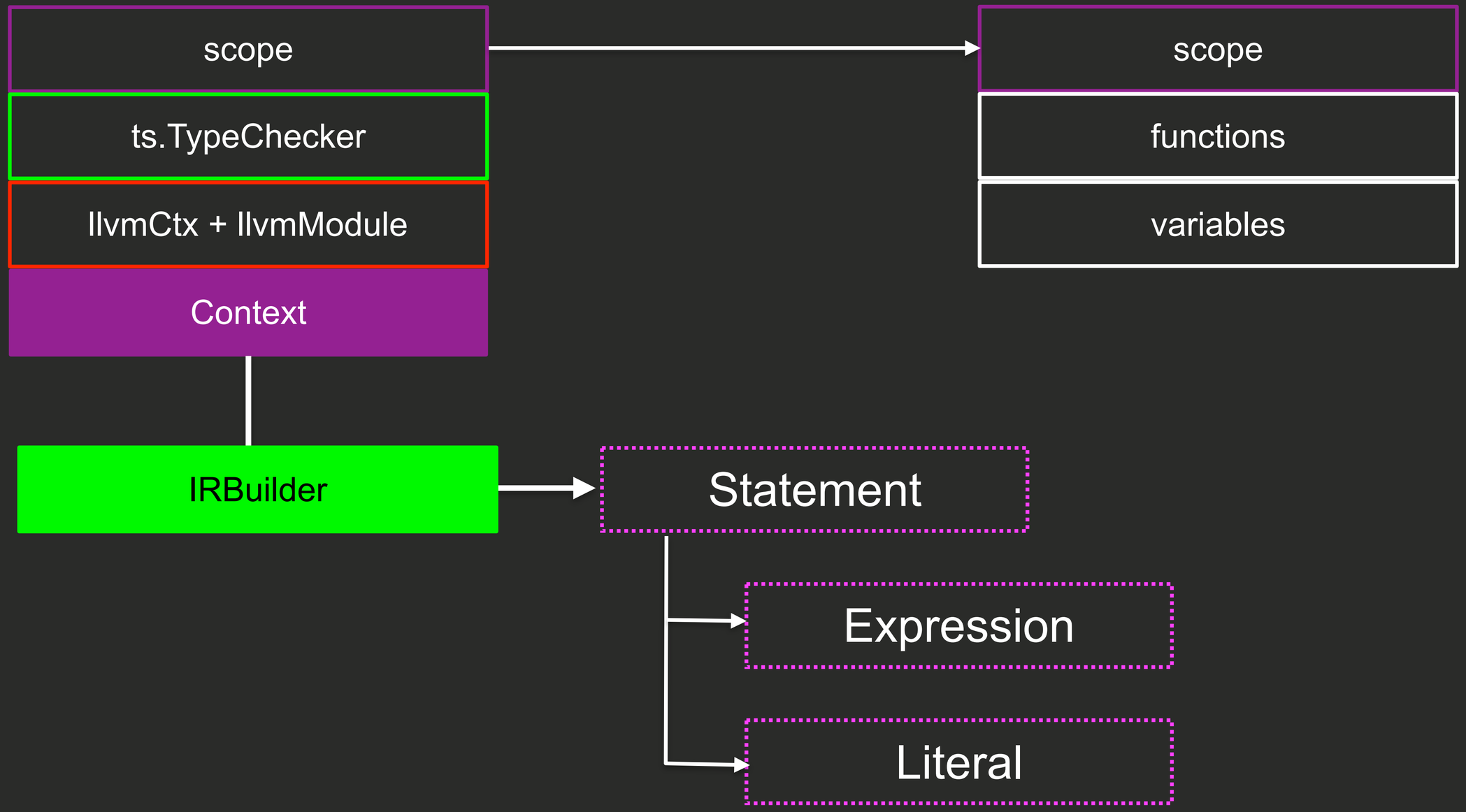
Каждый уважающий себя программист хочет сделать свой собственный компилятор.
Мечты сбываются! LLVM — важный шаг, позволяющий избежать велосипедостроения.
(C) @gridem

```
import * as llvm from 'llvm-node';

const llvmContext = new llvm.LLVMContext();
const llvmModule = new llvm.Module("test", this.llvmContext);

const mainFnType = llvm.FunctionType.get(
  llvm.Type.getVoidTy(llvmContext),
  false
);
const mainFn = llvm.Function.create(
  mainFnType,
  LinkageTypes.ExternalLinkage,
  "main",
  llvmModule
);

const block = llvm.BasicBlock.create(llvmContext, "entry", mainFn);
const builder = new llvm.IRBuilder(block);
```

Backend

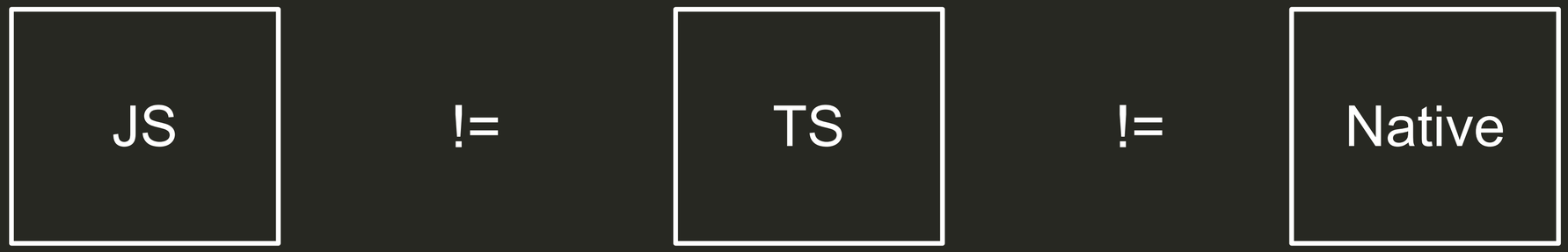



```
-> llvm.BasicBlock
-> llvm.value
-> llvm.value
```


scope
ts.TypeChecker
llvmCtx + llvmModule
Context
IRBuilder
scope
functions
variables
Statement
Expression
Literal

Compiler
Type System

The main problem




```
export class NativeTypeResolver {
    static getType(type: ts.Type, ctx: Context): NativeType|null {
        if (type.isNumberLiteral()) {
            return new NativeType(
                llvm.Type.getDoubleTy(
                    ctx.llvmContext
                )
            );
        }

        // Магия
    }
}
```

staticscript.d.ts

```
declare type int8 = {};
declare type int16 = {};
declare type int32 = {};
declare type int64 = {};
declare type int128 = {};
```

Support some non-structural (nominal) type matching #202

Open **iislucas** opened this issue on 23 Jul 2014 · 377 comments · May be fixed by #33038

Nominal `unique type` brands #33038

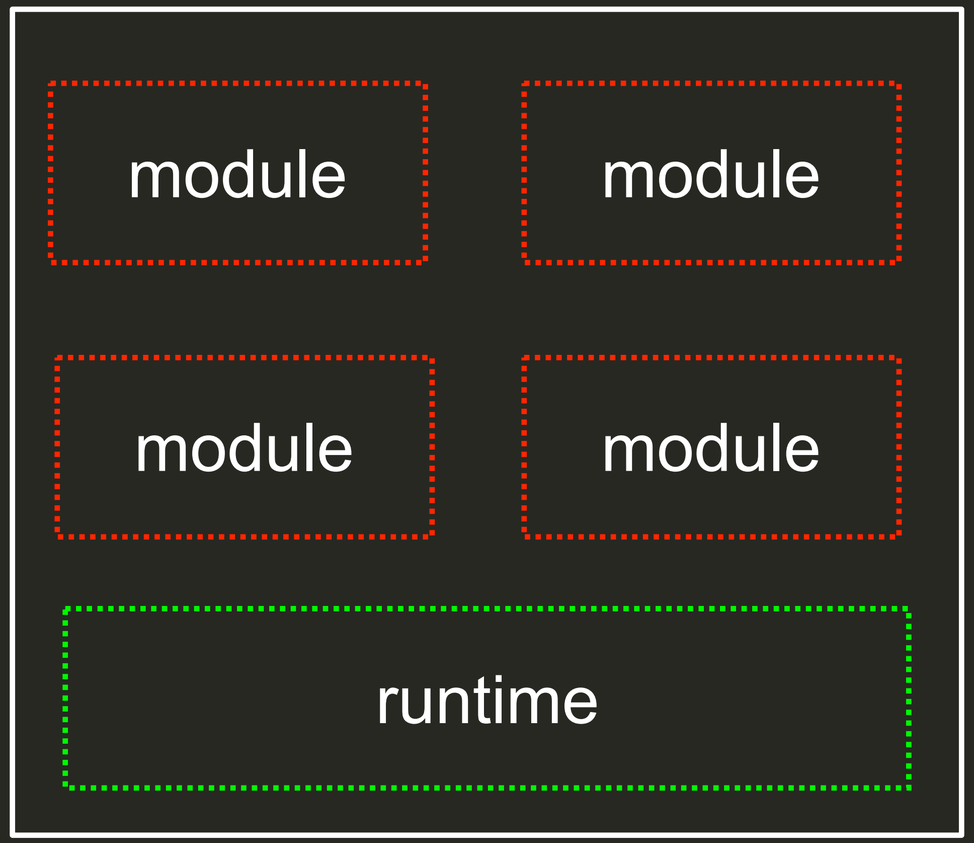
Open **weswigham** wants to merge 1 commit into `microsoft:master` from `weswigham:unique-types`

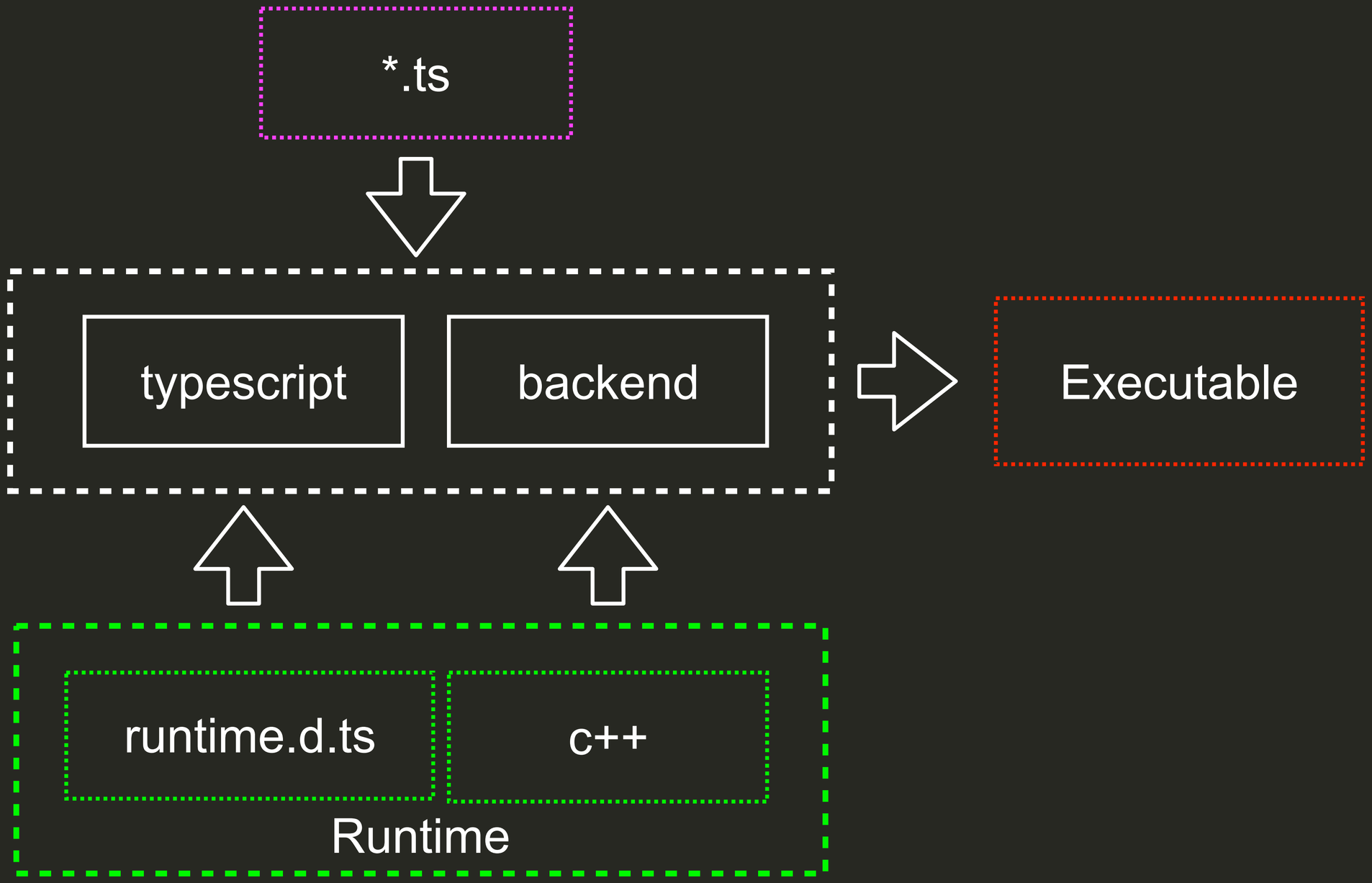
Conversation 35 | Commits 1 | Checks 4 | Files changed 31

https://basarat.gitbooks.io/typescript/docs/tips/nominalTyping.html

Compilter
Runtime Library

What is runtime library?




*.ts
typescript
backend
Executable
runtime.d.ts
c++
Runtime

runtime.d.ts

```
/// <reference no-default-lib="true"/>

declare function console_log(value: number): void;
declare function console_log(value: string): void;
declare function console_log(value: boolean): void;
```

```
LIBRARY_EXPORT void console_log(double number) {
    puts(number2string(number));
}

LIBRARY_EXPORT void console_log(const char *str) {
    puts(str);
}

LIBRARY_EXPORT void console_log(bool boolean) {
    if (boolean) {
        puts("true");
    } else {
        puts("false");
    }
}
```

Name mangling

In compiler construction, **name mangling** (also called **name decoration**) is a technique used to solve various problems caused by the need to resolve unique names for programming entities in many modern programming languages.

A single C++ translation unit might define two functions named `f()`:

```
int  f (void) { return 1; }
int  f (int)  { return 0; }
void g (void) { int i = f(), j = f(0); }
```

These are distinct functions, with no relation to each other apart from the name. The C++ compil

```
int  __f_v (void) { return 1; }
int  __f_i (int)  { return 0; }
void __g_v (void) { int i = __f_v(), j = __f_i(0); }
```

https://en.wikipedia.org/wiki/Name_mangling

./bin/ssc sandbox/do-simple-math.ts --printIR

```
console_log(1.0);
console_log(true);
console_log("str");
```

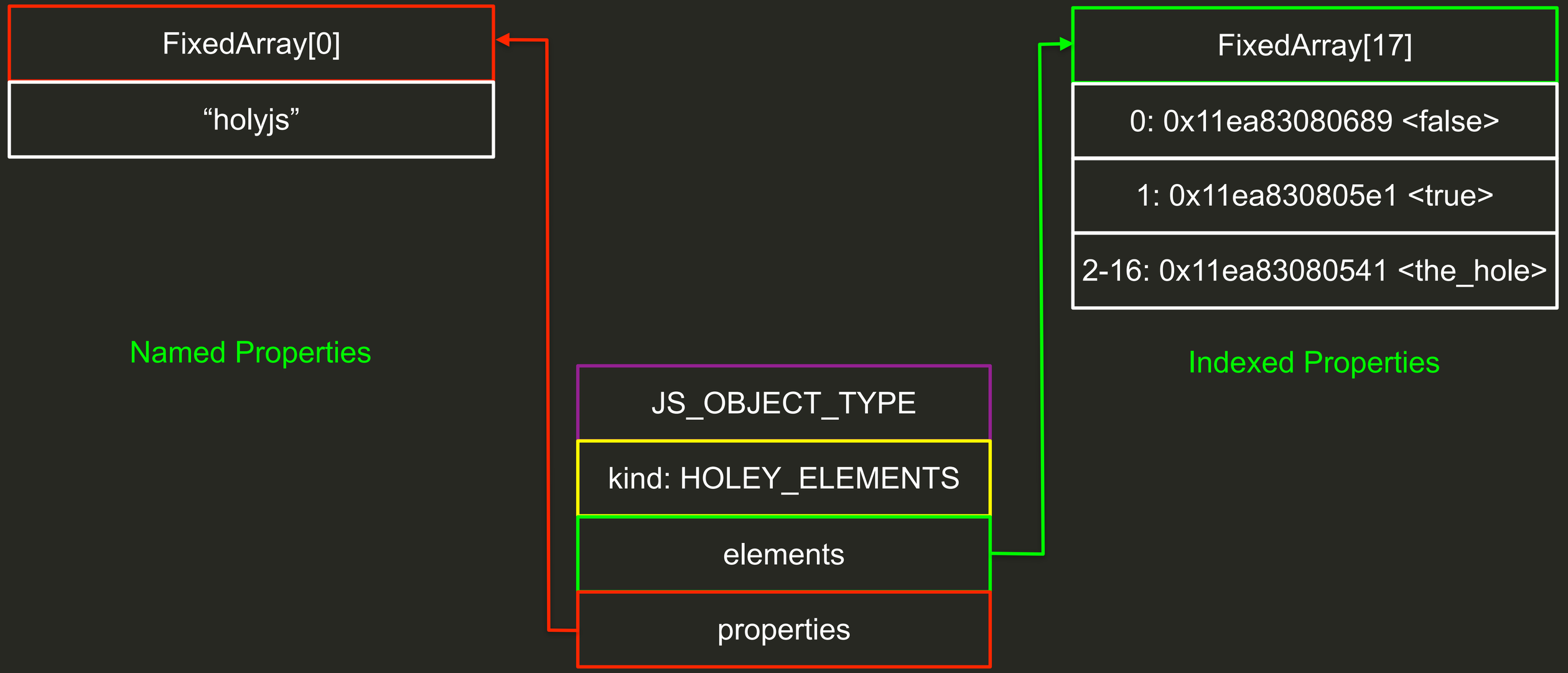
```
@0 = private constant [4 x i8] c”str\00"

define i64 @main() {
entry:
  call void @_Z11console_logd(double 1.000000e+00)
  call void @_Z11console_logb(i1 true)
  call void @_Z11console_logPKc(i8* getelementptr …)
  ret i64 0
}

declare void @_Z11console_logd(double)
declare void @_Z11console_logb(i1)
declare void @_Z11console_logPKc(i8*)
```

Compiler
Object/Classes

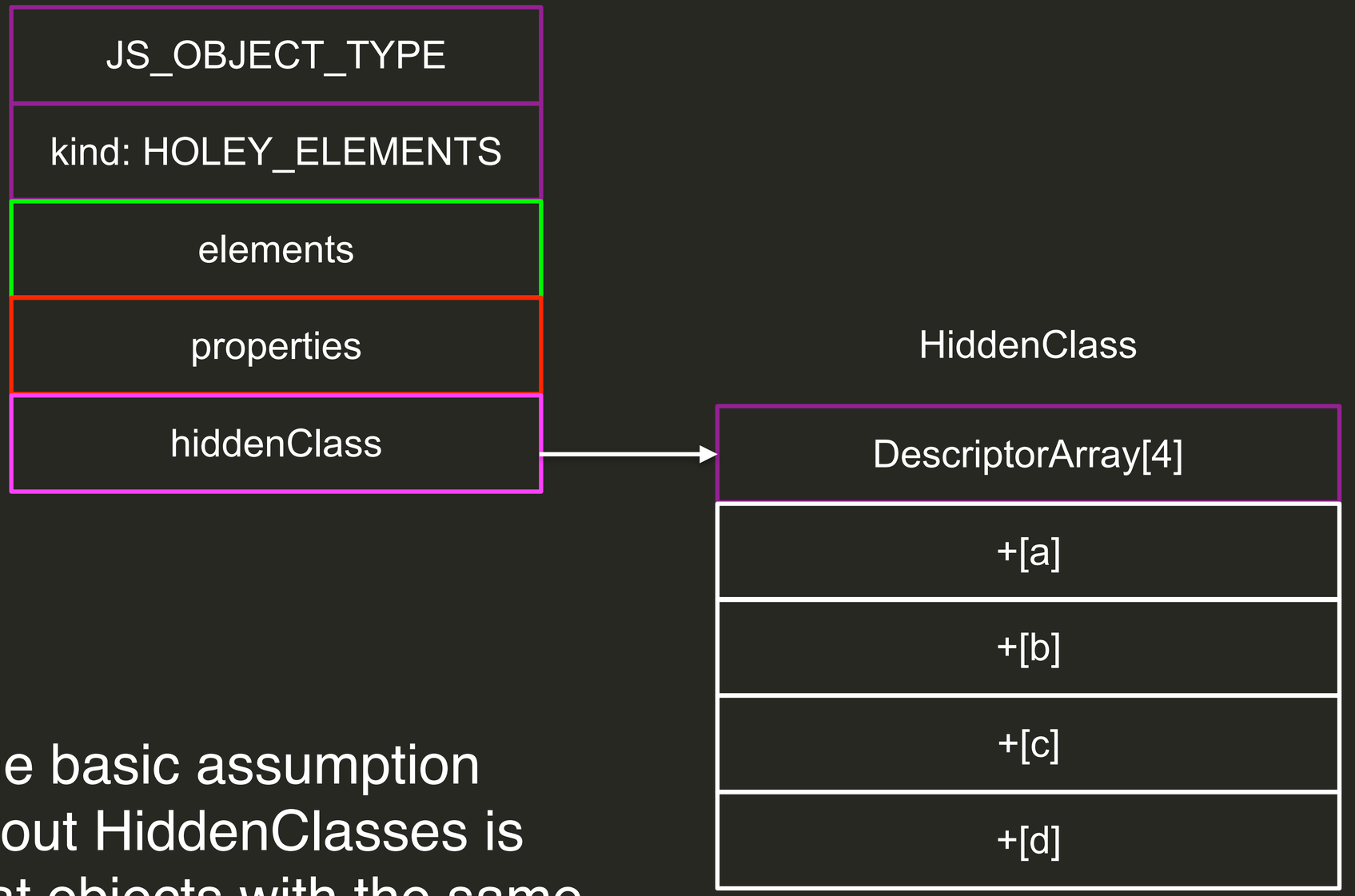
```
d8 --allow-natives-syntax
```

```
%DebugPrint({0: false, 1: true, a: “holyjs”});
```

%DebugPrint({0: false, 1: true, a: “holyjs”});



```
var a = {};

a.a = "hello";
a.b = "holyjs";
a.c = "moscow";
a.d = "2019";

%DebugPrint(a);
```



The basic assumption about HiddenClasses is that objects with the same structure

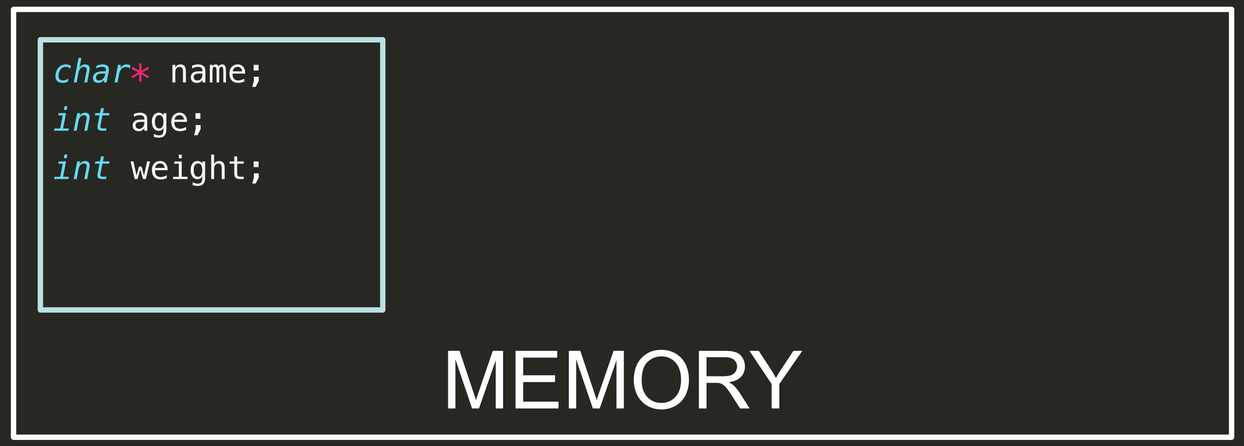
Compiler
Native Classes

TypeScript Class + methods

```
class User {
    protected name: string;
    protected age: number;
    protected weight: number;

    constructor(name: string, age: number, weight: number) {
        this.name = name;
        this.age = age;
        this.weight;
    }

    public getWeight(): number {
        return this.weight;
    }
}
```

```
%struct.User = type { i8*, double, double }
```

```
struct User {
    char* name;
    double age;
    double weight;
};
```




```
User* construct_User(char *name, double age, double weight)
{
    // code
}

double getWeight(User *ptr) {
    return ptr->weight;
}
```

Compiler

```
const struct = llvm.StructType.create(ctx.llvmContext, structName);

struct.setBody(properties.map(
    (property: ts.Symbol) => {
        const nativeType = NativeTypeResolver.getType(
            ctx.typeChecker.getTypeOfSymbolAtLocation(property, node),
            ctx
        );

        return nativeType.getType();
    }
));
```

Compiler
Dynamic Type

Dynamic typing

```
{
    let result;

    if (Math.random() > 0) {
        result = {};
    } else {
        result = "string";
    }

    console.log(result);
}
```

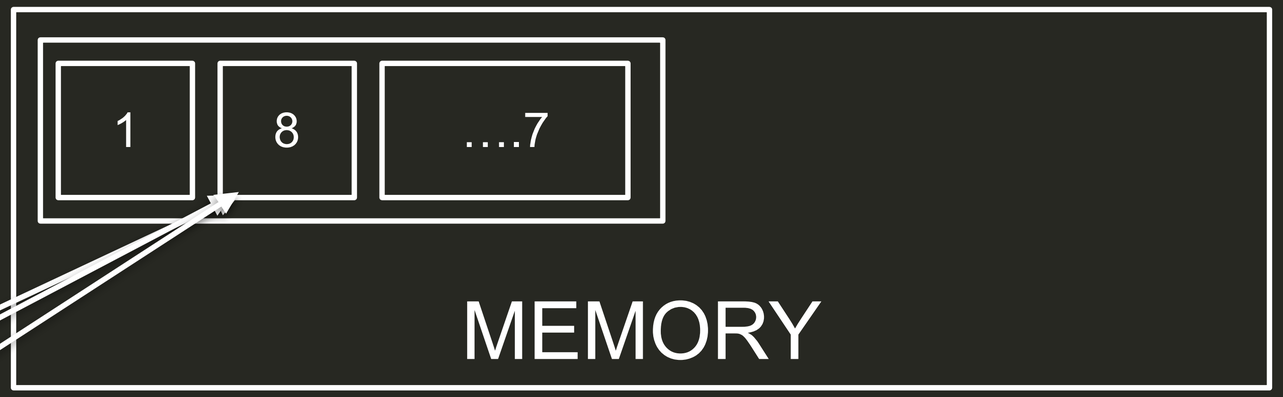
```
let result;
let result: any
```

Dynamic (C++ code)

```
enum DynamicType: int8_t {
    BOOLEAN = 1,
    NUMBER = 2,
    INT64 = 3,
    UNDEFINED = 3,
};
```

```
class Dynamic {
private:
    DynamicType type;

    union {
        double number;
        bool boolean;
        int64_t int64;
    };

};
```

```cpp
enum DynamicType: int8_t {
    BOOLEAN = 1,
    NUMBER = 2,
    INT64 = 3,
    UNDEFINED = 3,
};

class Dynamic {
private:
    DynamicType type;

    union {
        double number;
        bool boolean;
        int64_t int64;
    };
};
```




```cpp
sizeof(Dynamic) = 16
```

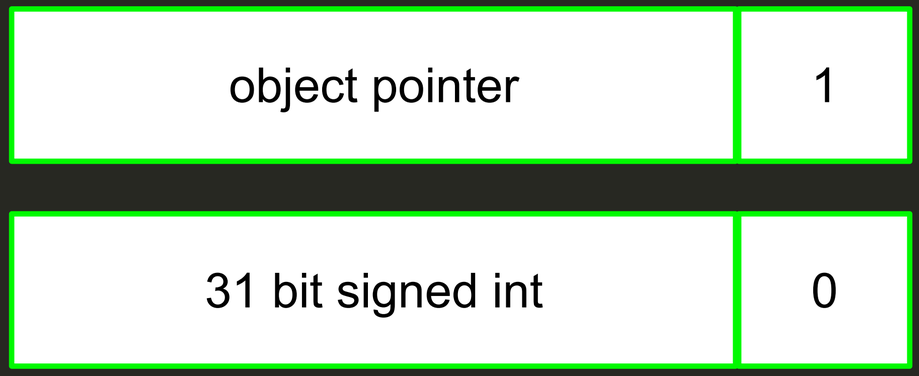
V8 Tagging / Smi

```
// Smi represents integer Numbers that can be stored in 31 bits.
// Smis are immediate which means they are NOT allocated in the heap.
// The ptr_ value has the following format: [31 bit signed int] 0
// For long smis it has the following format:
//     [32 bit signed int] [31 bits zero padding] 0
// Smi stands for small integer.
class Smi : public Object {
 public:
  // This replaces the OBJECT_CONSTRUCTORS macro, because Smis are special
  // in that we want them to be constexprs.
  constexpr Smi() : Object() {}
  explicit constexpr Smi(Address ptr) : Object(ptr) {
#if V8_CAN_HAVE_DCHECK_IN_CONSTEXPR
    DCHECK(HAS_SMI_TAG(ptr));
#endif
  }
```

Smi

```
// Smi represents integer Numbers that can be stored in 31 bits.
// Smis are immediate which means they are NOT allocated in the heap.
// The ptr_ value has the following format: [31 bit signed int] 0
// For long smis it has the following format:
//     [32 bit signed int] [31 bits zero padding] 0
// Smi stands for small integer.
```

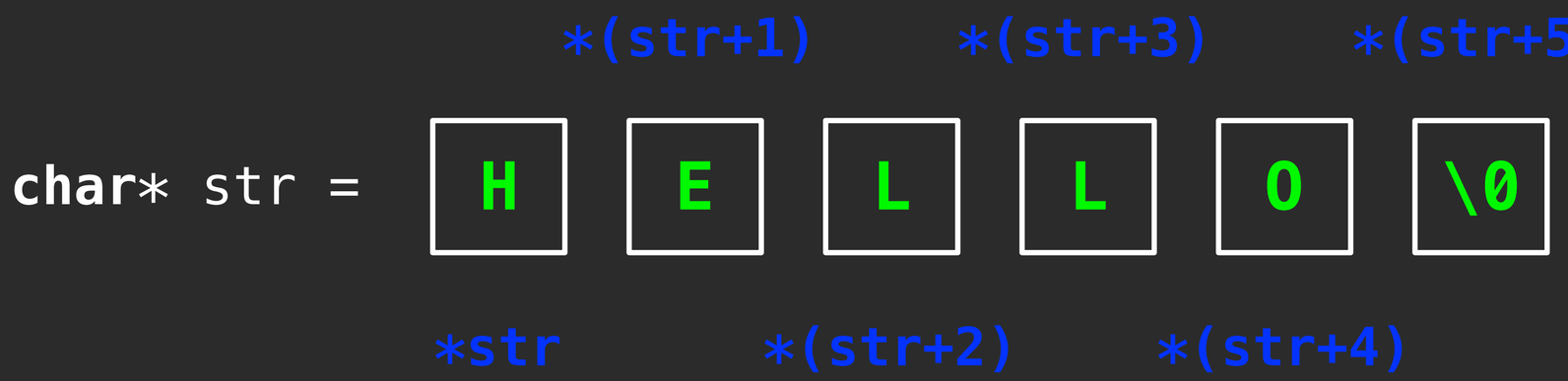
Address = uintptr_t = x64 machine = 8 byte = 64 bit



-2^{31} to $2^{31}-1$

Compiler
Array Type

Native C/C++ Arrays

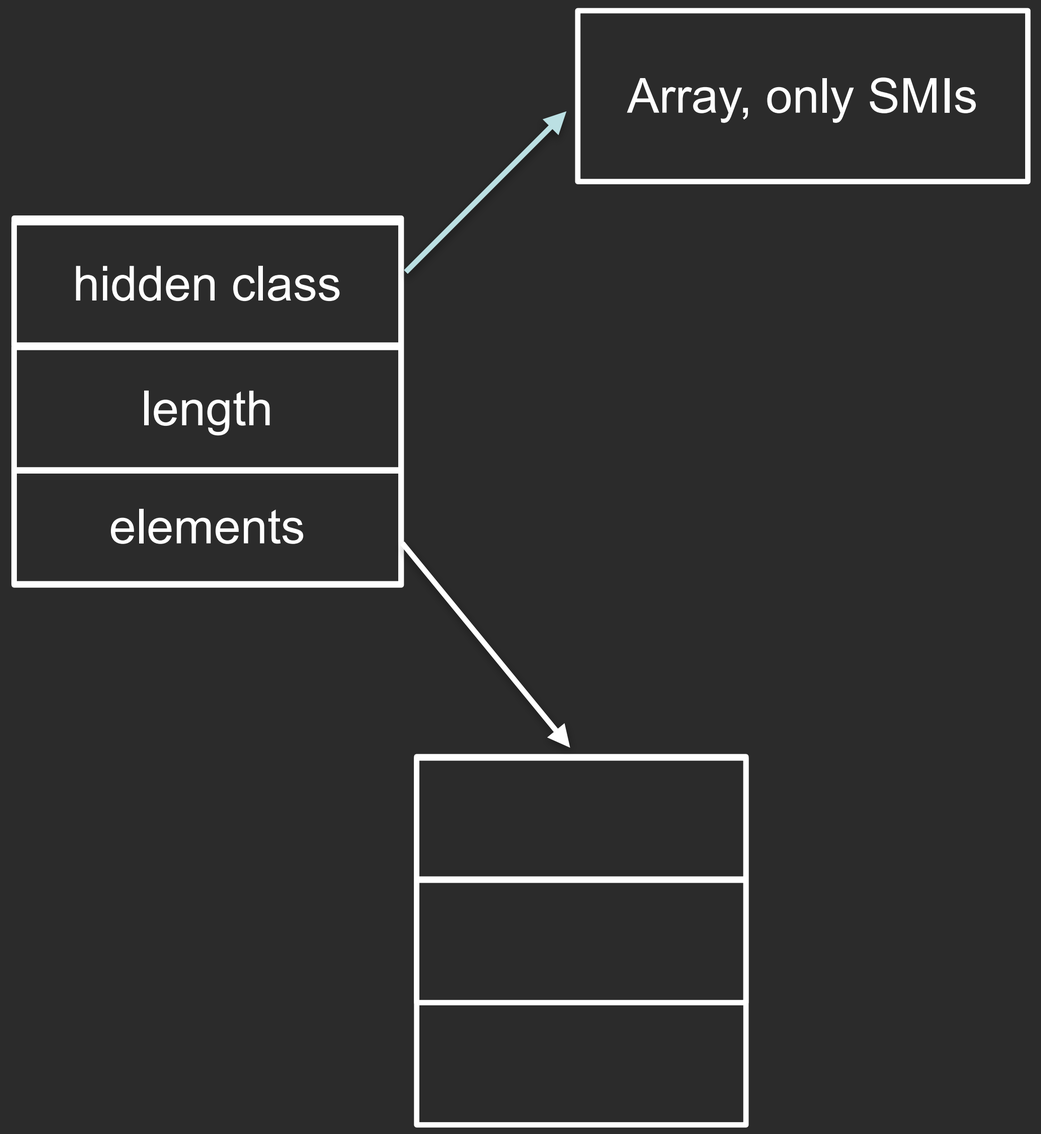


Dynamic C/C++ Arrays

```
struct Vector {
    void* elements;
    int32_t size;
    int32_t capacity;
};
```

```
this.scope.classes.set(
    'Uint8Array',
    ArrayLiteralExpressionCodeGenerator.buildTypedArrayStructLLVMType(
        llvm.Type.getInt8Ty(this.llvmContext),
        this,
        'array<uint8>'
    )
);

static buildTypedArrayStructLLVMType(elementType, ctx, name): llvm.StructType
{
    const structType = llvm.StructType.create(ctx.llvmContext, name);
    return structType.setBody([
        elementType,
        // size
        llvm.Type.getInt32Ty(ctx.llvmContext),
        // capacity
        llvm.Type.getInt32Ty(ctx.llvmContext),
    ]);
}
```

V8 Arrays



V8

```
{
    let result = [];

    for (let i = 0; i < 10000; i++) {
        result.push(Math.random())
    }
}
```

64-bit floating point values = 8 byte

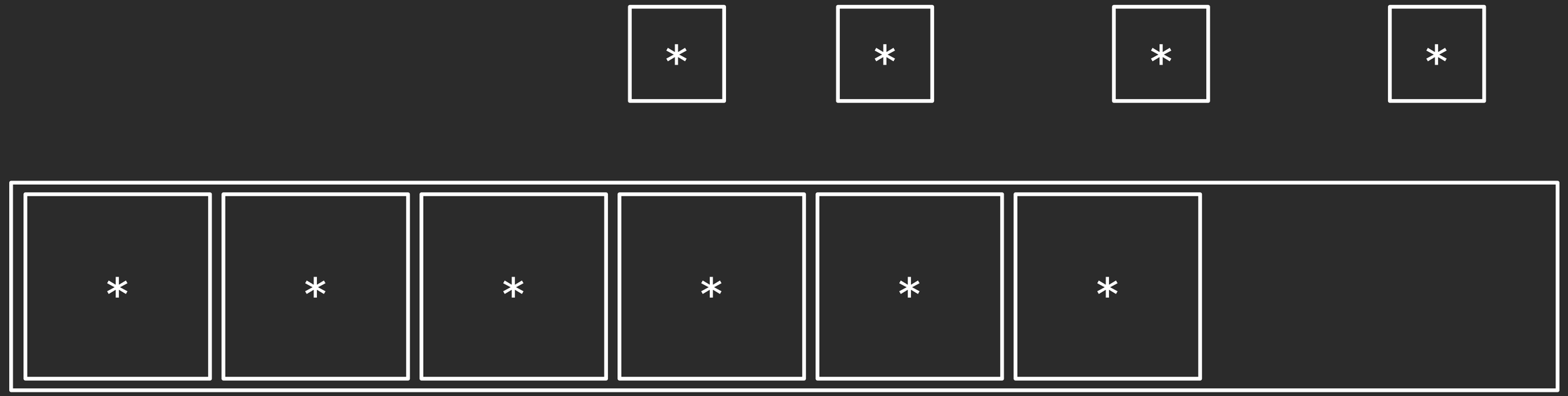
V8

```
{
    let result = [];

    for (let i = 0; i < 10000; i++) {
        result.push(Math.random())
    }

    result.push("Hehehehe Another Type");
}
```

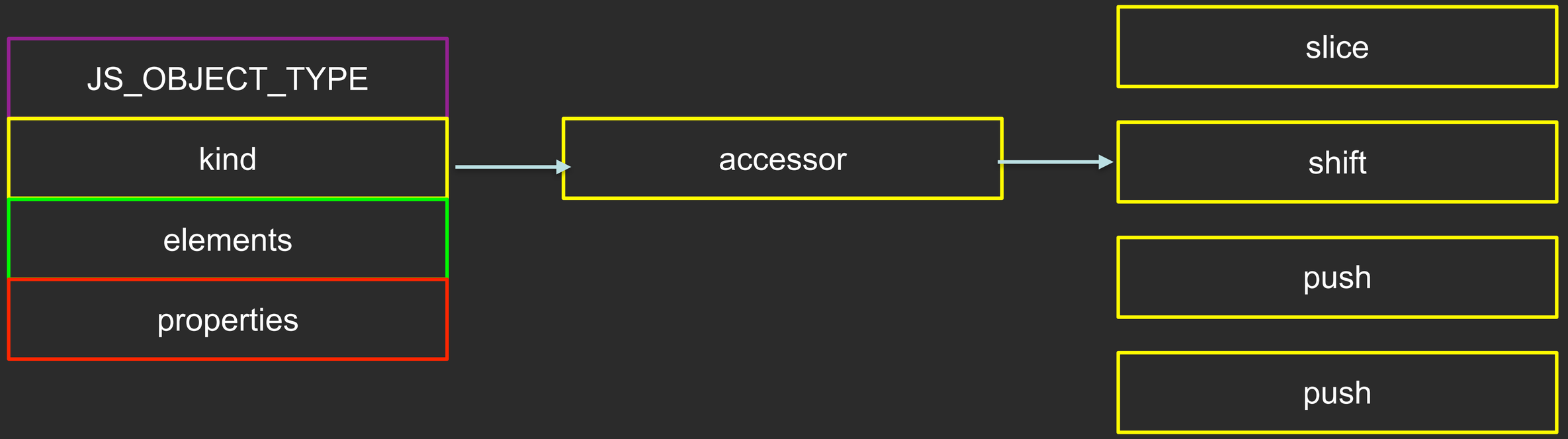
24 byte

Unaligned object & memory in array

```
{
    let values = [true, null, "holyjs", []];
}
```



V8 Element Kind



https://v8.dev/blog/elements-kinds

Compiler Branches

clang -S -emit-llvm main.cpp -O0

```cpp
#include <cstdio>

int main() {
    auto a = 5;
    auto b = 4;

    if (a > b) {
        puts("5 > 4");
    } else {
        puts("5 < 4");
    }
}
```

```
@.str = private unnamed_addr constant [6 x i8] c"5 > 4\00", align 1
@.str.1 = private unnamed_addr constant [6 x i8] c"5 < 4\00", align 1

define i32 @main() #0 {
  %a.0 = alloca i32, align 4
  %b.0 = alloca i32, align 4
  store i32 5, i32* %a.0, align 4
  store i32 4, i32* %b.0, align 4

  %a.1 = load i32, i32* %a.0, align 4
  %b.1 = load i32, i32* %b.0, align 4
  %6 = icmp sgt i32 %a.1, %b.1
  br i1 %6, label %7, label %9

; <label>:7: ; preds = %0
  %8 = call i32 @puts(i8* @.str)
  br label %11

; <label>:9: ; preds = %0
  %10 = call i32 @puts(i8* @.str.1))
  br label %11

; <label>:11:
  ret i32 0
}
```

```
auto a = 5;
auto b = 4;

if (a > b) {

puts("5 > 4");

puts("5 < 4");

 ; preds = %9, %7
```

```
generate(node: ts.IfStatement, ctx: Context, builder: llvm.IRBuilder): void {
    const positiveBlock = llvm.BasicBlock.create(ctx.llvmContext, "if.true");
    ctx.scope.enclosureFunction.llvmFunction.addBasicBlock(positiveBlock);

    const negativeBlock = llvm.BasicBlock.create(ctx.llvmContext, "if.false");
    ctx.scope.enclosureFunction.llvmFunction.addBasicBlock(negativeBlock);

    const next = llvm.BasicBlock.create(ctx.llvmContext, "if.end");
    ctx.scope.enclosureFunction.llvmFunction.addBasicBlock(next);
}
```

```
generate(node: ts.IfStatement, ctx: Context, builder: llvm.IRBuilder): void {
    const positiveBlock = llvm.BasicBlock.create(ctx.llvmContext, "if.true");
    ctx.scope.enclosureFunction.llvmFunction.addBasicBlock(positiveBlock);

    const negativeBlock = llvm.BasicBlock.create(ctx.llvmContext, "if.false");
    ctx.scope.enclosureFunction.llvmFunction.addBasicBlock(negativeBlock);

    const next = llvm.BasicBlock.create(ctx.llvmContext, "if.end");
    ctx.scope.enclosureFunction.llvmFunction.addBasicBlock(next);

    emitCondition(
        node.expression,
        ctx,
        builder,
        positiveBlock,
        negativeBlock
    );
}
```

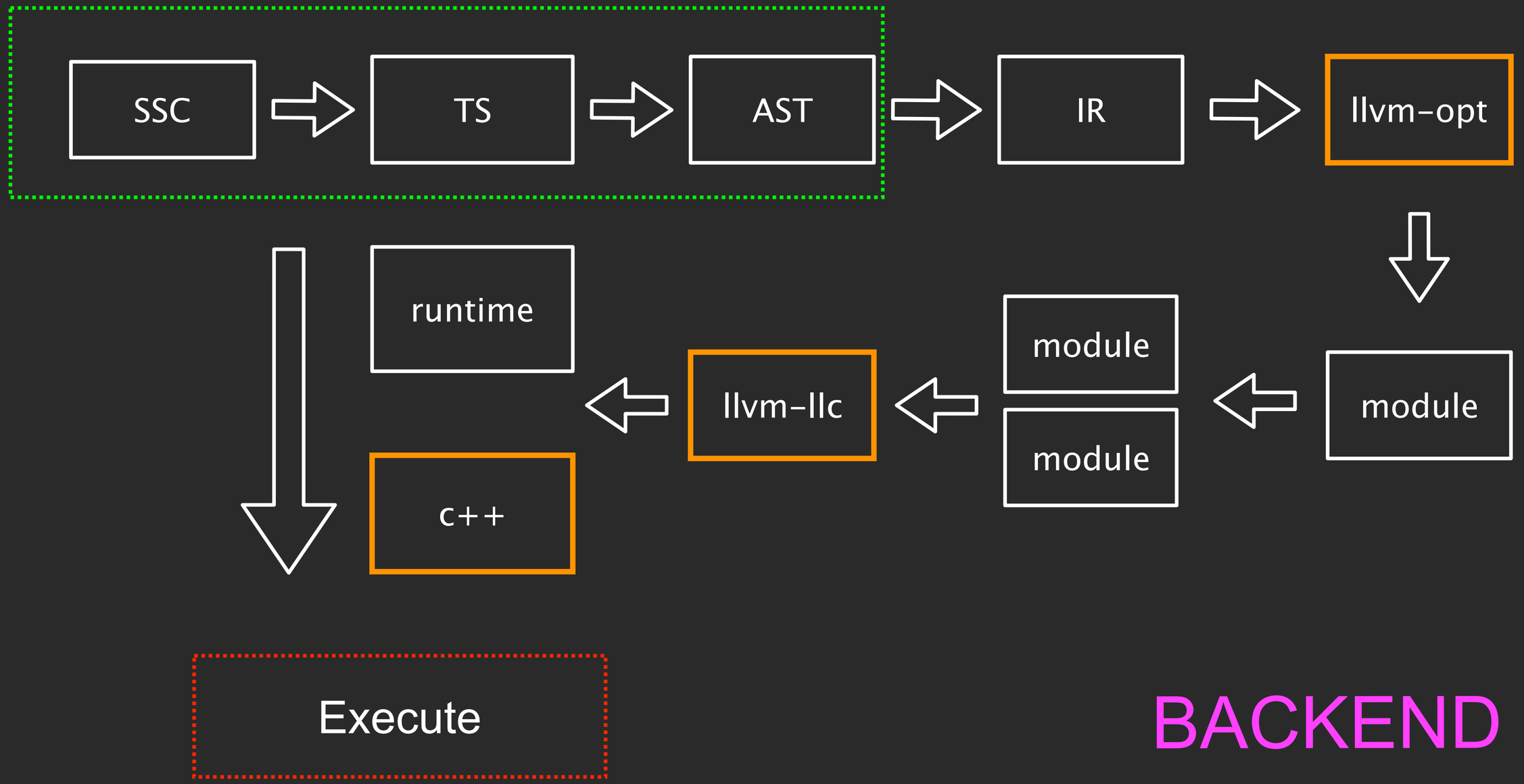
if (expression) {

```
export function emitCondition(
    condition: ts.Expression,
    ctx: Context,
    builder: llvm.IRBuilder,
    positiveBlock: llvm.BasicBlock,
    negativeBlock: llvm.BasicBlock,
) {
    const left = buildFromExpression(condition, ctx, builder);

    const conditionBoolValue = left.toBoolean(ctx, builder, condition);
    builder.createCondBr(conditionBoolValue.getValue(), positiveBlock, negativeBlock);
}
```

```
generate(node: ts.IfStatement, ctx: Context, builder: llvm.IRBuilder): void {
    //.. .. ..

    emitCondition(
        node.expression,
        ctx,
        builder,
        positiveBlock,
        negativeBlock
    );

    builder.setInsertionPoint(positiveBlock);
    passNode(node.thenStatement, ctx, builder);
    builder.createBr(next);

    builder.setInsertionPoint(negativeBlock);
    passNode(node.elseStatement, ctx, builder);
    builder.createBr(next);

    builder.setInsertionPoint(next);
}
```

Final Architecture

FRONTEND



BACKEND

But what about compiler for TypeScript?

JS over dynamic typed language with dynamic types

Dart

Вопросы?)

https://github.com/ovr

talk@dmtry.me

https://telegram.me/ovrweb