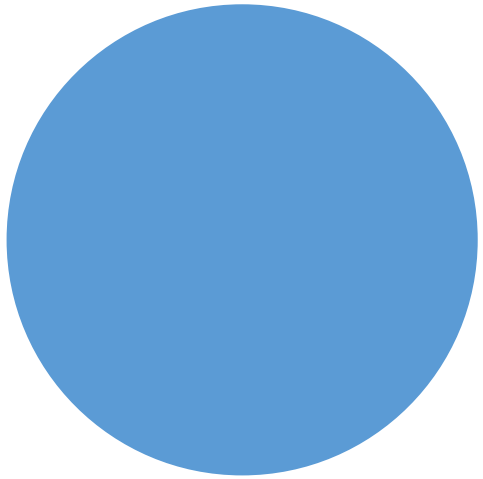
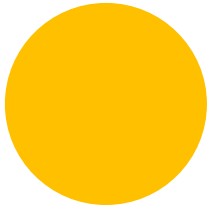



Introduction to Python JIT 101 In 15 Mins

나동희

Introduce myself

- 1월 말까지 합법적(?) 백수
- Pyston & Grumpy 프로젝트 컨트리뷰터
- Cpython에는 패치 한 5개 정도 미니멀하게..
- 컴파일러 전문가는 아닙니다..
- TOROS N2 (https://github.com/kakao/n2)

Today topic

- Discuss about Cpython interpreter
- What is JIT?
- How JIT works?

What is the interpreter

- 여기 계신 분들은 거의 다 아시겠지만..
- 파이썬은 Stack based interpreter 입니다.
- 느려요.. 느려

```
switch (opcode) {

/* BEWARE!
   It is essential that any operation that fails sets either
   x to NULL, err to nonzero, or why to anything but WHY_NOT,
   and that no operation that succeeds does this! */

TARGET(NOP)
    FAST_DISPATCH();

TARGET(LOAD_FAST) {
    PyObject *value = GETLOCAL(oparg);
    if (value == NULL) {
        format_exc_check_arg(PyExc_UnboundLocalError,
                             UNBOUNDLOCAL_ERROR_MSG,
                             PyTuple_GetItem(co->co_varnames, oparg));
        goto error;
    }
    Py_INCREF(value);
    PUSH(value);
    FAST_DISPATCH();
}

PREDICTED(LOAD_CONST);
TARGET(LOAD_CONST) {
    PyObject *value = GETITEM(consts, oparg);
    Py_INCREF(value);
    PUSH(value);
    FAST_DISPATCH();
}
```

ceval.c

No optimization on interpreter level?

- Constant folding 등 가벼운 최적화는 CPython 레벨에서 합니다.
- 많은 최적화는 할 수가 없다. (e.g Dead code elimination)
- AST 레벨에서 최적화를 하려는 많은 노력은 존재합니다. (e.g FAT Python)

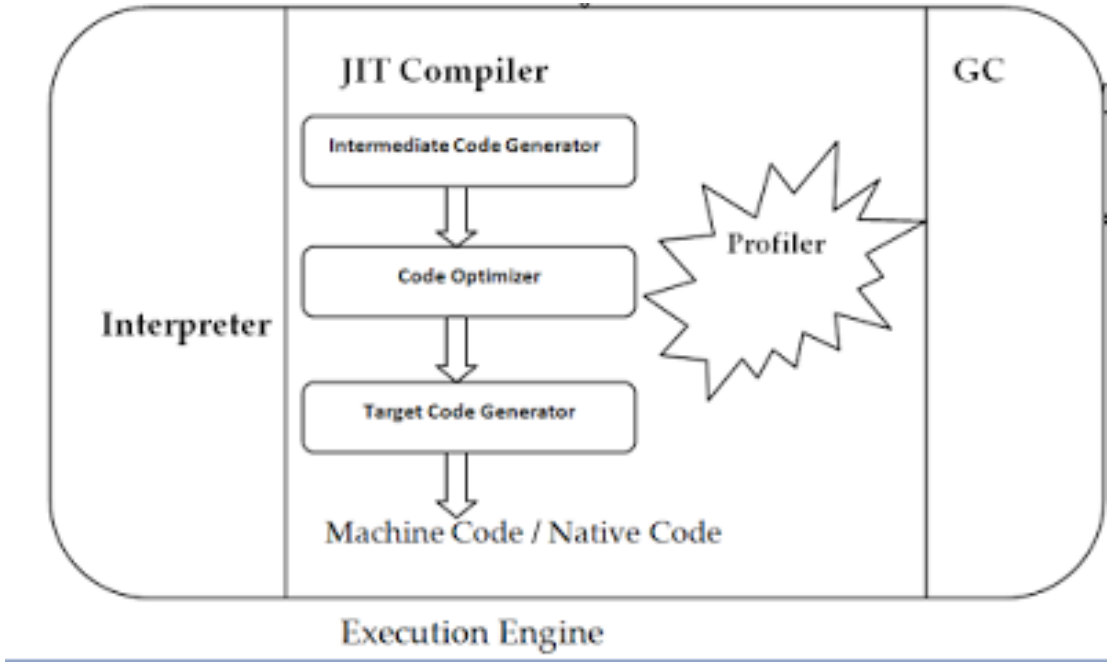
```
    /* Skip over LOAD_CONST trueconst
       POP_JUMP_IF_FALSE xx.  This improves
       "while 1" performance.  */
case LOAD_CONST:
    cumlc = lastlc + 1;
    if (nextop != POP_JUMP_IF_FALSE  ||
        !ISBASICBLOCK(blocks, op_start, i + 1)  ||
        !PyObject_IsTrue(PyList_GET_ITEM(consts, get_arg(codestr, i))))
        break;
    fill_nops(codestr, op_start, nexti + 1);
    cumlc = 0;
    break;

    /* Try to fold tuples of constants.
       Skip over BUILD_SEQN 1 UNPACK_SEQN 1.
       Replace BUILD_SEQN 2 UNPACK_SEQN 2 with ROT2.
       Replace BUILD_SEQN 3 UNPACK_SEQN 3 with ROT3 ROT2. */
case BUILD_TUPLE:
    j = get_arg(codestr, i);
    if (j > 0 && lastlc >= j) {
        h = lastn_const_start(codestr, op_start, j);
        if (ISBASICBLOCK(blocks, h, op_start)) {
            h = fold_tuple_on_constants(codestr, h, i+1, consts, j);
            break;
        }
    }
    if (nextop != UNPACK_SEQUENCE  ||
        !ISBASICBLOCK(blocks, op_start, i + 1) ||
        j != get_arg(codestr, nexti))
        break;
```

peephole.c

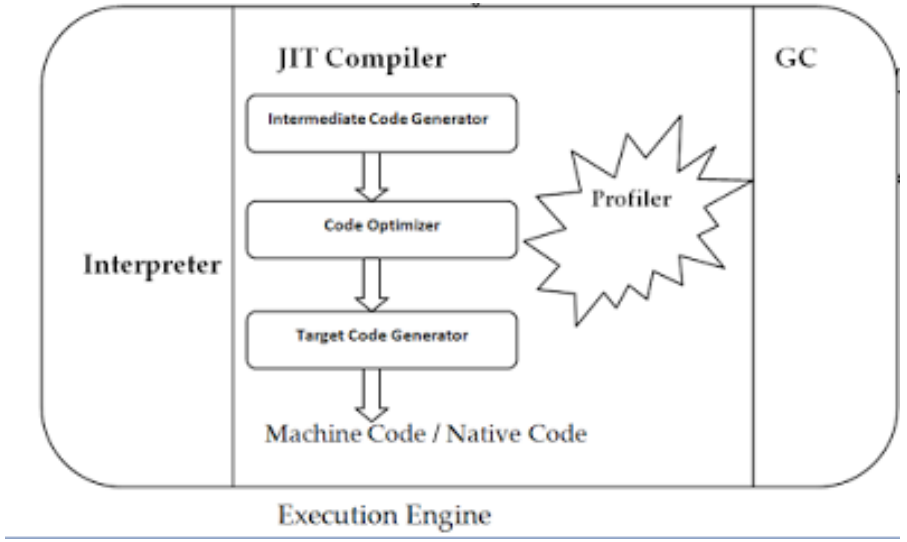
So what is JIT?

- 학부수업 때 듣기로는 컴파일러와 인터프리터 중간이다.
- JVM은 JIT으로 돌아간다고 하던데, 같은 방법으로 하면 되지 않을까요?



What is JIT

- 쉽게 설명하면, 런타임에 자주 사용되는(Hot spot) 코드를 머신코드 레벨로 컴파일하고, 메모리에 저장한 후
- 해당 코드가 실행되어야 하면 메모리에 있는 머신코드를 실행한다.
- 실제 코드로 보시죠



```
long add4(long n) {
  return n + 4;
}
```

이놈을 JIT 시켜보겠습니다.

Add4 함수를 다음과 같이 기계어 코드로 바꿉니다.

- 실제로 런타임에 컴파일러 백엔드가 다음과 같이 함수를 기계어로 바꾸면 되겠죠?

- ARM 타겟이면 ARM 머신코드, NVPTX 타겟이면 NVPTX 머신코드

```
void emit_code_into_memory(uns
  unsigned char code[] = {
    0x48, 0x89, 0xf8,
    0x48, 0x83, 0xc0, 0x04,
    0xc3
  };
  memcpy(m, code, sizeof(code)
}
```

기계어 코드를 저장할 메모리를 할당합니다.

- 해당 메모리를 실행가능하게 설정해야하고, 쓰기 권한과 읽기 권한 모드 획득해야 합니다.

- RWX 권한을 모드 획득하면 보안 취약점이 있습니다.
- 실제로는 RW 권한 획득 이후 코드를 삽입 후 RX 권한으로 권한을 변경합니다.

```
void* allocate_executable_memory(size_t size){
  void *ptr = mmap(0, size,
                   PROT_READ | PROT_WRITE | PROT_EXEC,
                   MAP_PRIVATE | MAP_ANONYMOUS, -1 , 0);
  if (ptr == (void*)-1){
    return NULL;
  }
  return ptr;
}
```

참 쉽죠?

- 뭔가 중간에 생략된 것 같지만 기분 탓입니다.
- JIT 참 쉽죠?

```
void emit_to_rw_run_from_rx(){
  void *m = alloc_writable_memory(SIZE);
  emit_code_into_memory(m);
  make_memory_executable(m, SIZE);

  JittedFunc func = m;
  int result = func(2);
  printf("result =%d\n", result);
}
```

그래서 JIT을 구현할 때마다 이짓을 해야하는가?

- 사실은 NO입니다
- 요즘은 훌륭한 프레임워크가 많이 있습니다.

- AsmJit - Complete x86/x64 JIT and Remote Assembler for C++
- DynASM
- LibJIT
- LLVM - MCJIT
- GCC - libgccjit
- GNU lightning - a library that generates assembly language code at run-time
- Xbyak - JIT assembler for x86(IA32), x64(AMD64, x86-64) by C++
- sljit - a stack-less platform independent JIT compiler

그 중에서 슈퍼 스타는 LLVM

- 애플에서 스위프트 개발에 참여하고, 테슬라 갔다가 지금은 구글 AI팀에 합류한 그분..
- 참고로 텐서플로우에도 JIT으로 LLVM이 달려있습니다 ㅎㅎ
- 웹킷도 B3 엔진을 달기 전까지는 FTL 엔진이라고 잠시 LLVM을 쓴 과거가 있습니다. (https://webkit.org/blog/3362/introducing-the-webkit-ftl-jit/)

Chris Lattner's Homepage

About Me

1. My résumé & publications.
2. I am @clattner_llvm on Twitter.
3. I am lattner on GitHub.
4. I am not a web designer.

Apple

I have worked for Apple since 2005, holding a number of different positions over the years (a partial history is available in the Apple section of my résumé). These days, I run the Developer Tools department, which is responsible for Xcode and Instruments, as well as compilers, debuggers, and related tools.

To answer a FAQ: Yes, I do still write code and most of it goes out to open source. However, due to the nature of the work, I usually can't talk about it until a couple of years after it happens. :)

Swift

I started work on the Swift Programming Language (wikipedia) in July of 2010. I implemented much of the basic language structure, with only a few people knowing of its existence. A few other (amazing) people started contributing in earnest late in 2011, and it became a major focus for the Apple Developer Tools group in July 2013.

The Swift language is the product of tireless effort from a team of language experts, documentation gurus, compiler optimization ninjas, and an incredibly important internal dogfooding group who provided feedback to help refine and battle-test ideas. Of course, it also greatly benefited from the experiences hard-won by many other languages in the field, drawing ideas from Objective-C, Rust, Haskell, Ruby, Python, C#, CLU, and far too many others to list.

The Xcode Playgrounds feature and REPL were a personal passion of mine, to make programming more interactive and approachable. The Xcode and LLDB teams have done a phenomenal job turning crazy ideas into something truly great. Playgrounds were heavily influenced by Bret Victor's ideas, by Light Table and by many other interactive systems. I hope that by making programming more approachable and fun, we'll appeal to the next generation of programmers and to help redefine how Computer Science is taught.

As of Dec 2015, Swift is open source! Join its development at http://swift.org.

Compilers

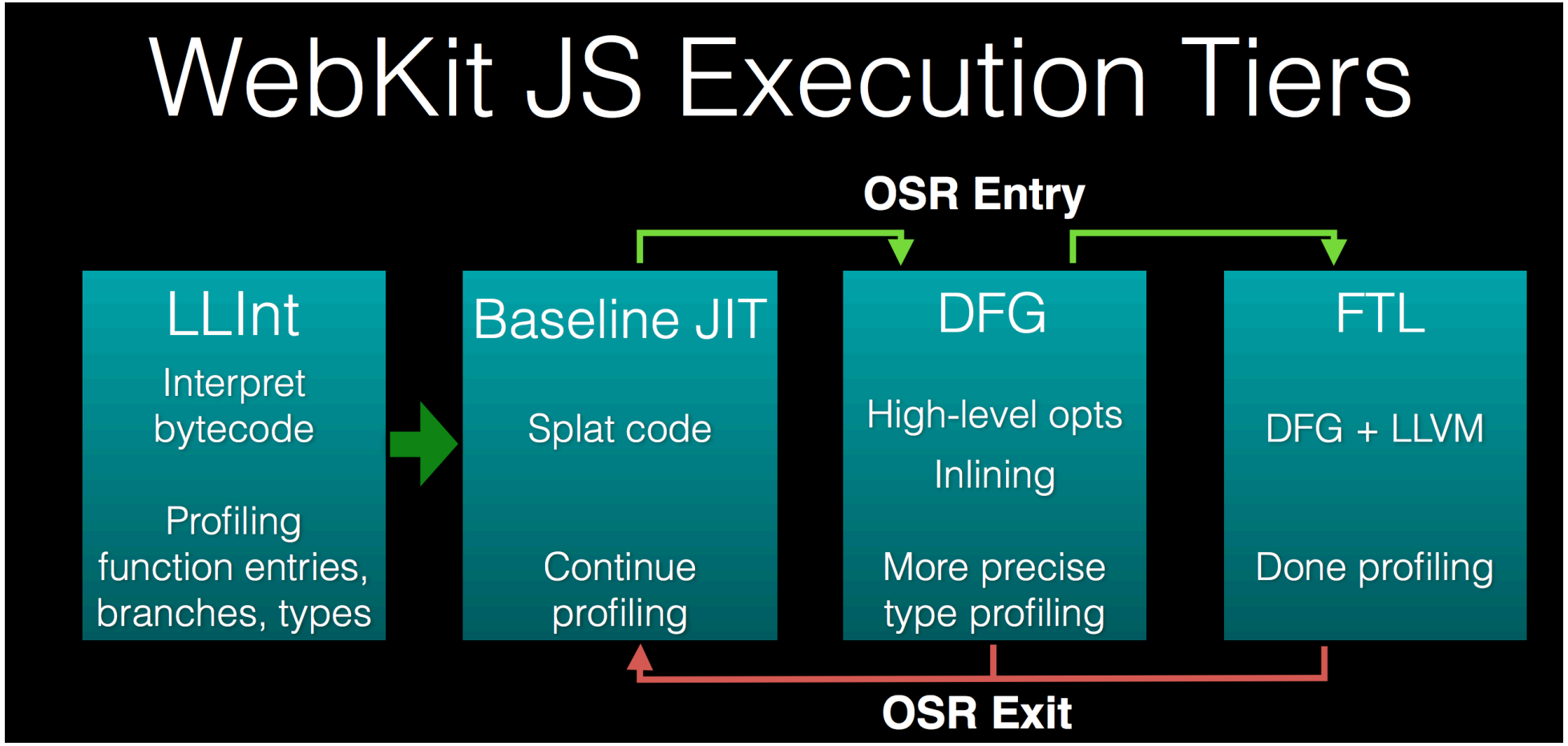
I lead and am the original author of the LLVM Compiler Infrastructure, an open source umbrella project that includes all sorts of toolchain related technology: compilers, debuggers, JIT systems, optimizers, static analysis systems, etc. I started both LLVM and Clang and am still the individual with the most commits. Of course, as the community has grown, my contribution is being dwarfed by those

Return to Python

- 그래서 파이썬에다가 JIT을 달면 되겠네라는 생각을 한 사람들이 많았습니다. 아니 당연한거죠.

- Dropbox Pyston, Numba, PyPy

Webkit FTL Pipeline



Dropbox Pyston pipeline

- Python source code → Python AST→ Python CFG → Execution

- AST Interpreter → Baseline JIT → LLVM JIT

Numba

- Python function bytecode -> numba IR -> type 추론 -> LLVM IR -> LLVM JIT

그러면

- 일단 AST를 이용해서든 뭐든 LLVM IR로 만들어서 JIT 코드를 생성하면 되는거네요!

- 이론상으로는 그런데!!!

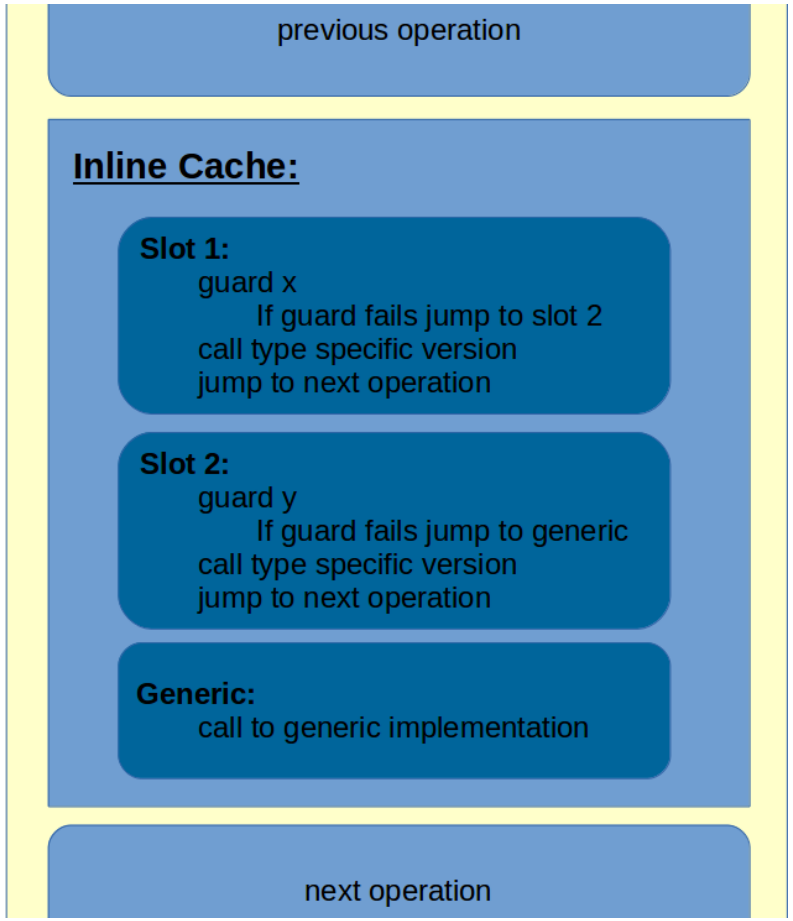
동적언어의 함정

```
def add(lhs, rhs):
    return lhs + rhs

add(1, 2)
add("Lunch", "Class")
add(3.5, 2.4)
add(1, 3.4)
add([1,2,3],[4,5,6])
```

Create a new Gist

Inline Cache (FTL & Pyston approach)



Numba approach by Numba IR

In [3]:
```
@jit
def add(a, b):
    return a + b
```

In [4]:
```
add(1,1)
```

Out[4]: 2

Numba approach by Numba IR

```
In [11]: add(3.5, 3.3)

Out[11]: 6.8
```

```
def add(a, b):

    # --- LINE 3 ---
    #   a = arg(0, name=a)  :: int64
    #   b = arg(1, name=b)  :: int64
    #   $0.3 = a + b  :: int64
    #   $0.4 = cast(value=$0.3)  :: int64
    #   return $0.4

    return a + b


================================================================
add (float64, float64)
----------------------------------------------------------------
# File: <ipython-input-3-7a2ac56f16b6>
# --- LINE 1 ---
# label 0
#   del b
#   del a
#   del $0.3

@jit

# --- LINE 2 ---

def add(a, b):

    # --- LINE 3 ---
    #   a = arg(0, name=a)  :: float64
    #   b = arg(1, name=b)  :: float64
    #   $0.3 = a + b  :: float64
    #   $0.4 = cast(value=$0.3)  :: float64
    #   return $0.4

    return a + b
```

Numba approach by Numba IR

```
In [12]: add("Lunch", "Class")

Out[12]: 'LunchClass'
```

```
def add(a, b):

    # --- LINE 3 ---
    #   a = arg(0, name=a)  :: float64
    #   b = arg(1, name=b)  :: float64
    #   $0.3 = a + b  :: float64
    #   $0.4 = cast(value=$0.3)  :: float64
    #   return $0.4

    return a + b


================================================================================
add (str, str)
--------------------------------------------------------------------------------
# File: <ipython-input-3-7a2ac56f16b6>
# --- LINE 1 ---
# label 0
#   del b
#   del a
#   del $0.3

@jit

# --- LINE 2 ---

def add(a, b):

    # --- LINE 3 ---
    #   a = arg(0, name=a)  :: pyobject
    #   b = arg(1, name=b)  :: pyobject
    #   $0.3 = a + b  :: pyobject
    #   $0.4 = cast(value=$0.3)  :: pyobject
    #   return $0.4

    return a + b


================================================================================
```

Numba approach by No Python mode

```
* (int64, int64) -> int64
* (int64, uint64) -> int64
* (uint64, int64) -> int64
* (uint64, uint64) -> uint64
* (float32, float32) -> float32
* (float64, float64) -> float64
* (complex64, complex64) -> complex64
* (complex128, complex128) -> complex128
* (uint8,) -> uint64
* (uint32,) -> uint64
* (uint16,) -> uint64
* (uint64,) -> uint64
* (int32,) -> int64
* (int64,) -> int64
* (int8,) -> int64
* (int16,) -> int64
* (float32,) -> float32
* (float64,) -> float64
* (complex64,) -> complex64
```

오늘 다루지 못한 Topic

- Garbage collector
- OSR opt & deopt (https://cs.chromium.org/chromium/src/v8/src/bailout-reason.h)
- JIT 프레임워크 사용법

Q&A