Type erasure in Python

Kir Chou @ PyCon APAC 2022

Type hint in 2022?

Type erasure

Objective

- Understand what is type erasure
  - Type erasure in static programming language (C++)
  - Type erasure in dynamic programming language (Python)
- Relevant knowledges
  - C++ development experience
- Nice to have knowledges
  - CPython extension module development experience

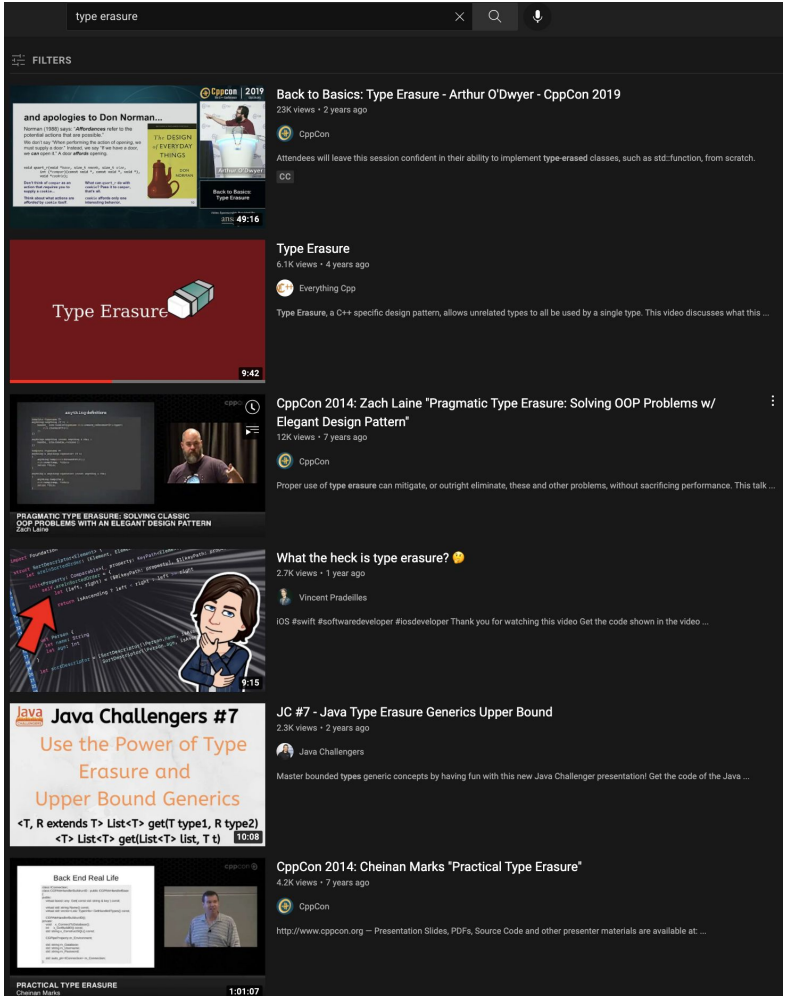
Agenda

- What is type erasure?
- Why do you need type erasure?
- Type erasure in C++
- Type erasure in Python

What is type erasure?

"type erasure is the load-time process by which explicit type annotations are removed from a program, before it is executed at run-time." - Wiki

If you search "type erasure" in Youtube. Top videos are in C++/Swift/Java.



Why do you need type erasure?

- 🙋 I am a pure Python developer
  ➡ ❌ You don't need to care about this 🤟😌👌

- 🙋 I am a CPython extension module developer
- 🙋 I want to run some C code in Python
  ➡ ⭕ Your C library can be written in with a pattern that needs this
  ➡ ⭕ Your module can hide the type complexity behind C
  ➡ ⭕ Your module can hide the type complexity behind Python

Type erasure in C++

Type erasure in C++ [ref]

Full example @ Github

```
struct Alice {
    void say() const { std::cout << "alice\n"; }
};

struct Bob {
    void say() const { std::cout << "bob\n"; }
};

int main() {
    AliceOrBob aliceOrBob{Alice()};
    aliceOrBob.say();  // alice
    aliceOrBob = Bob();
    aliceOrBob.say();  // bob
}
```

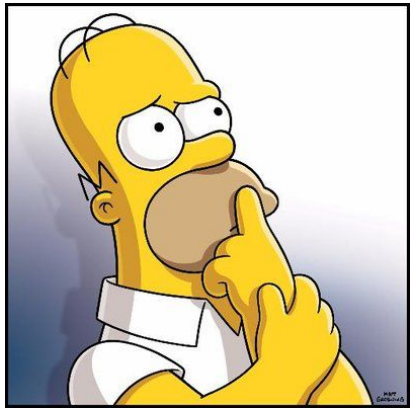
std::any [ref]

```
#include <any>

int main()  {
  // i: 1
  std::any a =  1;
  std::cout << a.type().name() << ": " << std::any_cast<int>(a) << '\n';

  // d: 3.14
  a =  3.14;
  std::cout << a.type().name() << ": " << std::any_cast<double>(a) << '\n';

  // b: 1
  a =  true;
  std::cout << a.type().name() << ": " << std::any_cast<bool>(a) << '\n';
}
```

Type erasure in Python?

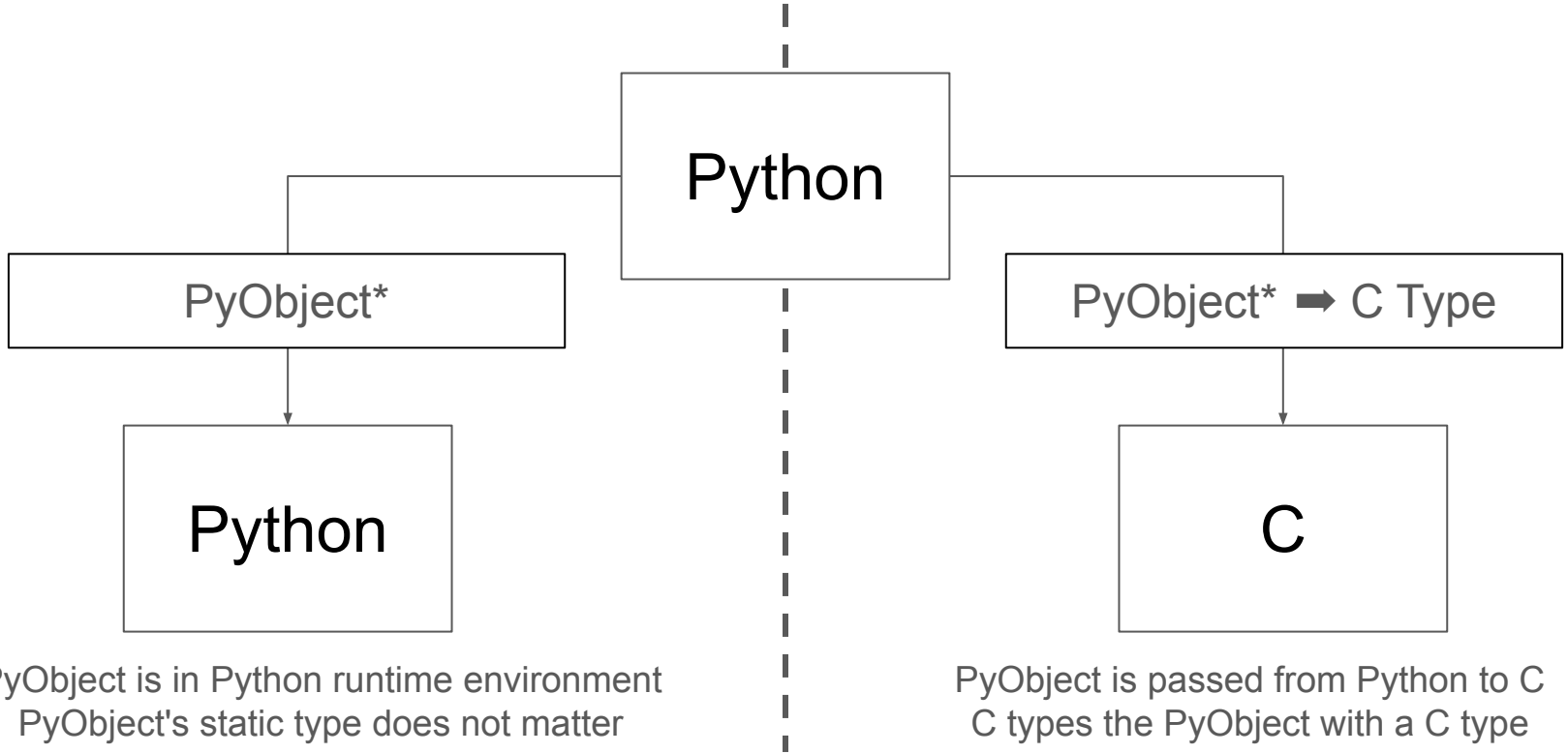
Facts

- In Python level, the type of an object is decided in runtime
- In C level, everything is PyObject* under Python interpreter



Wait? Why does Python need type erasure?

Case 1: Python developers write Python

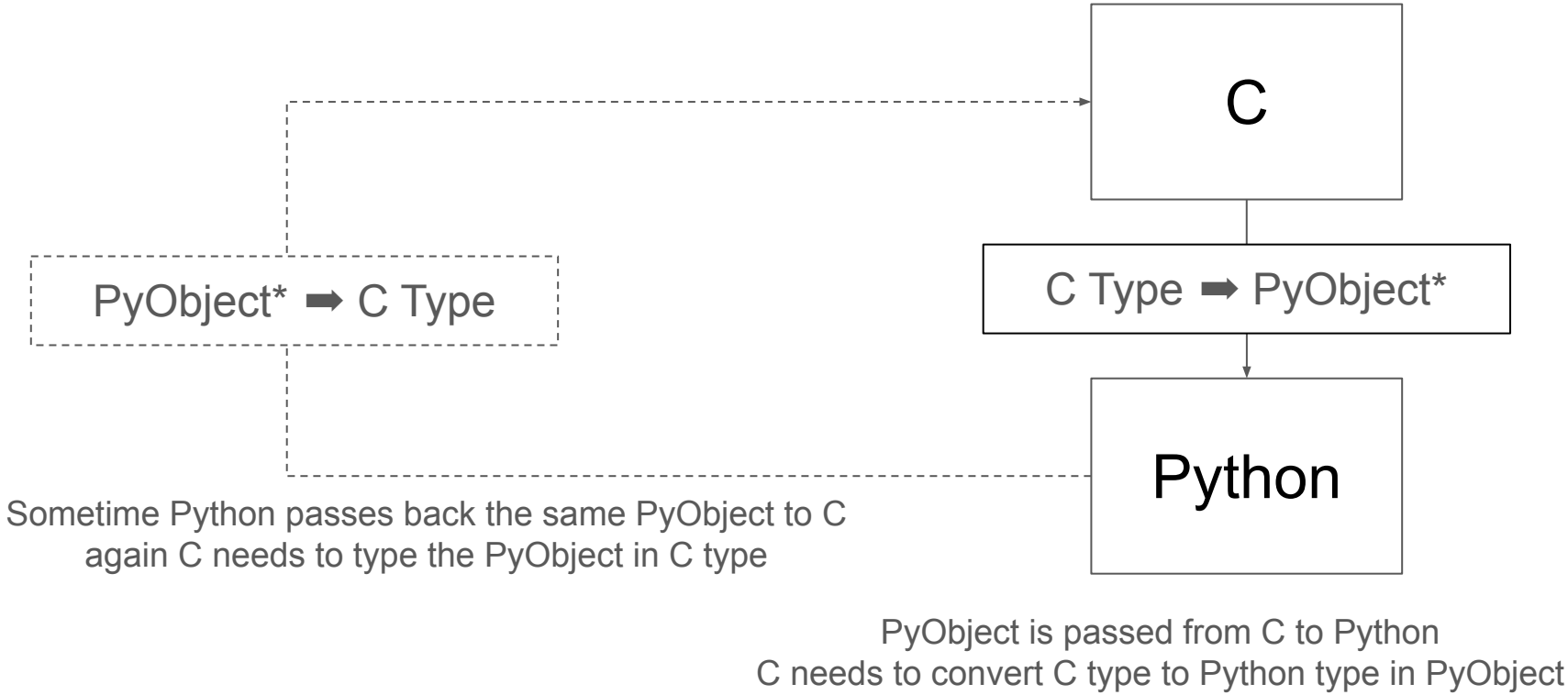


PyObject* ➡ C Type

PyArg_ParseTuple is one commonly-used CAPI to type PyObject* to C type

```
static PyObject* capi_add(PyObject* self, PyObject* args) {
    long a, b;
    if (!PyArg_ParseTuple(args, "ll", &a, &b)) {
        return NULL;
    }
    return PyLong_FromLong(a + b);
}
```

Case 2: C developers write CPython extension modules



C Type ➡ PyObject*

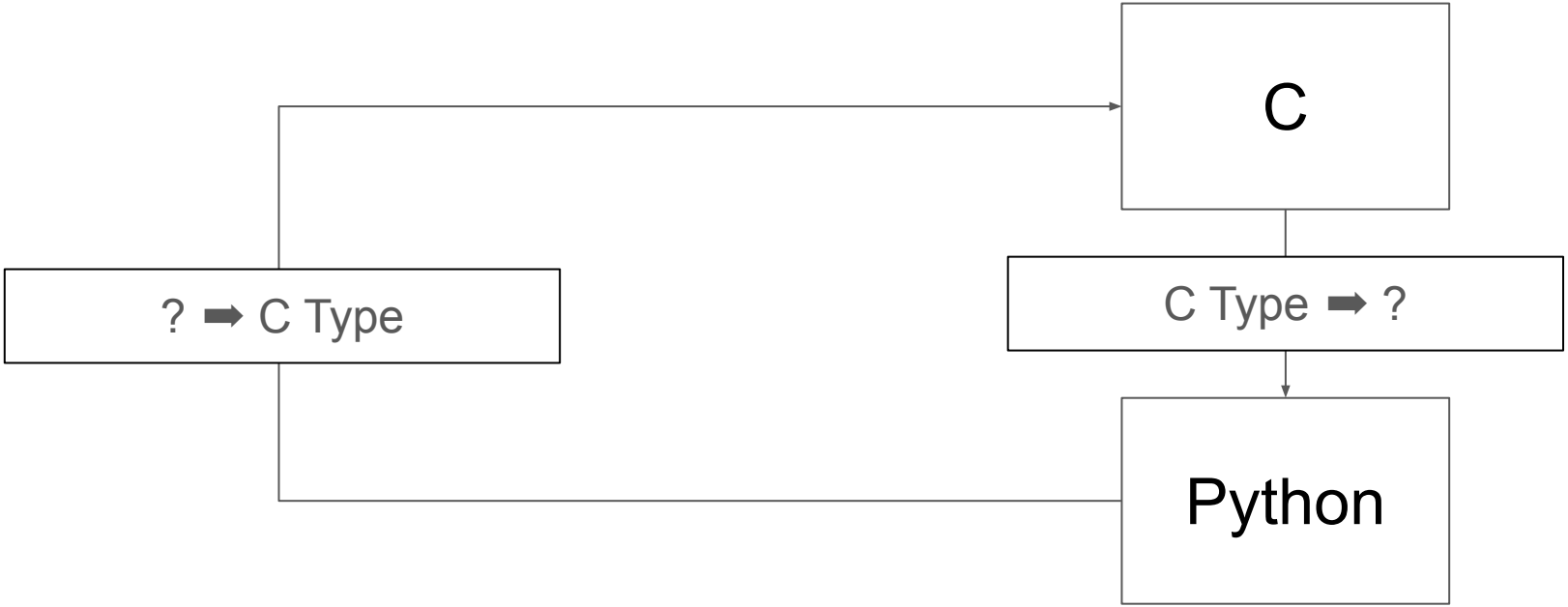
C-API supports the primitive types in Python: [Py<Python Type>_From<C Type>]

```
static PyObject* capi_add(PyObject* self, PyObject* args) {
    long a, b;
    if (!PyArg_ParseTuple(args, "ll", &a, &b)) {
        return NULL;
    }
    return PyLong_FromLong(a + b);
}
```

```
>>> capi_add(1, 2)
>>> 3
```

Type erasure in Python!

Type erasure example in Python



Then, let's erase the type in Python!

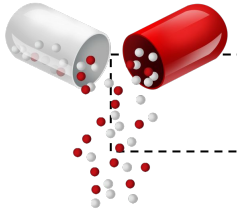
C Type ➡ ? ➡ C Type

Type erasure in Python
Python does NOT need to care about the type of the object passed from C

Workflow:

1. C function returns a PyCapsule to Python
2. The capsule is **type erased** and **unused** in Python
3. Python function passes the capsule back to another C function to make C function use it

Workflow



? ➡ C Type

C

C Type ➡ ?

Python

Example

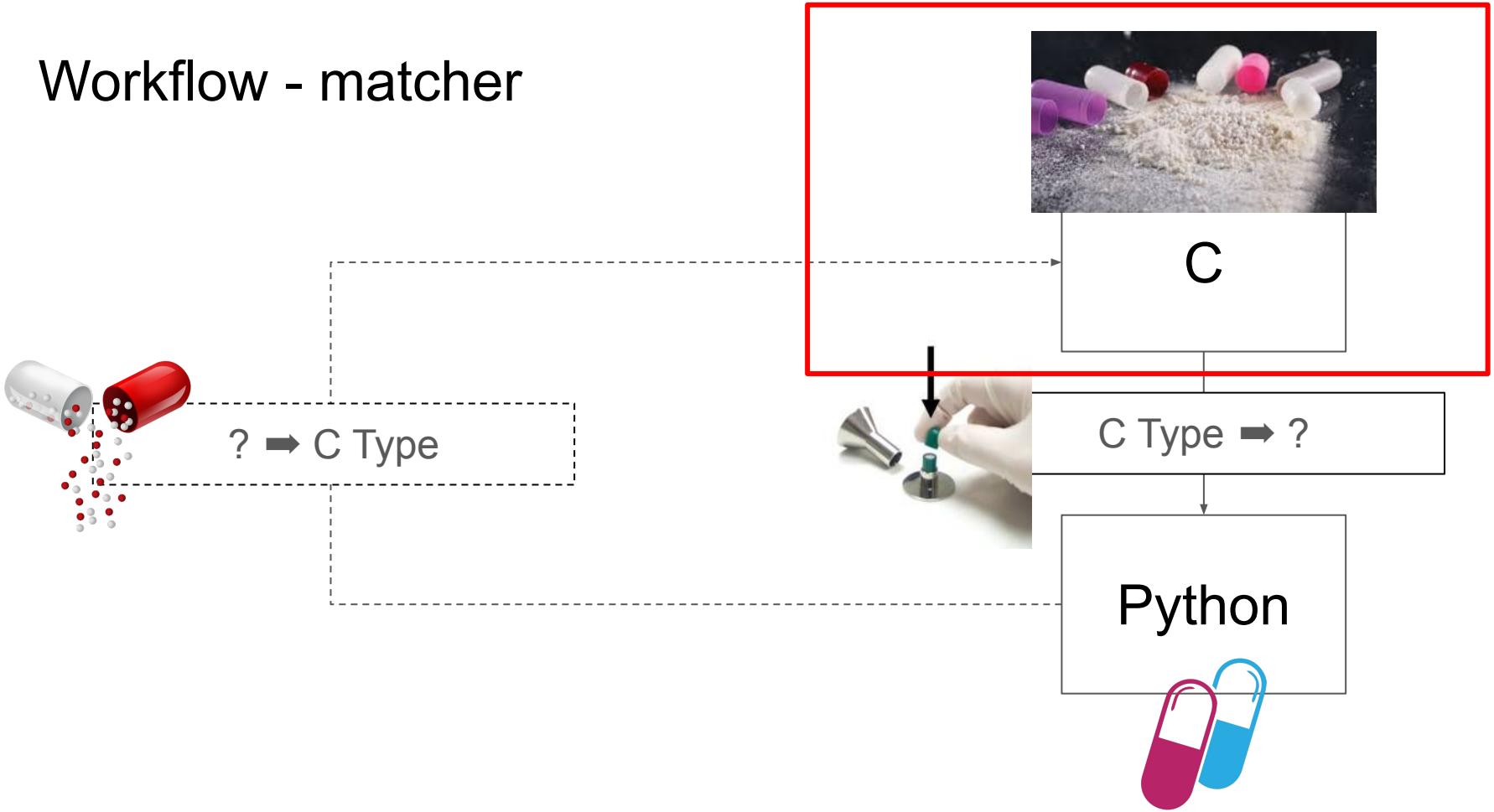
Full example @ Github

Say we want to develop a **string matcher** module in C++ with following methods:

- `def get_matcher_xxx() -> object`
  - C++ can implement different matchers here, let's say we have "exactly" and "partially" matcher
- `def is_matcher(matcher: object) -> bool`
  - Type is erased in Python, but Python can still check the type by this C method
- `def match(s1: str, s2: str, matcher: object) -> bool`
  - Uses the matcher to check if s1 and s2 matches

This use cases satisfy the requirement:
Python does **NOT** need to care about the type of the matcher passed from C

Workflow - matcher



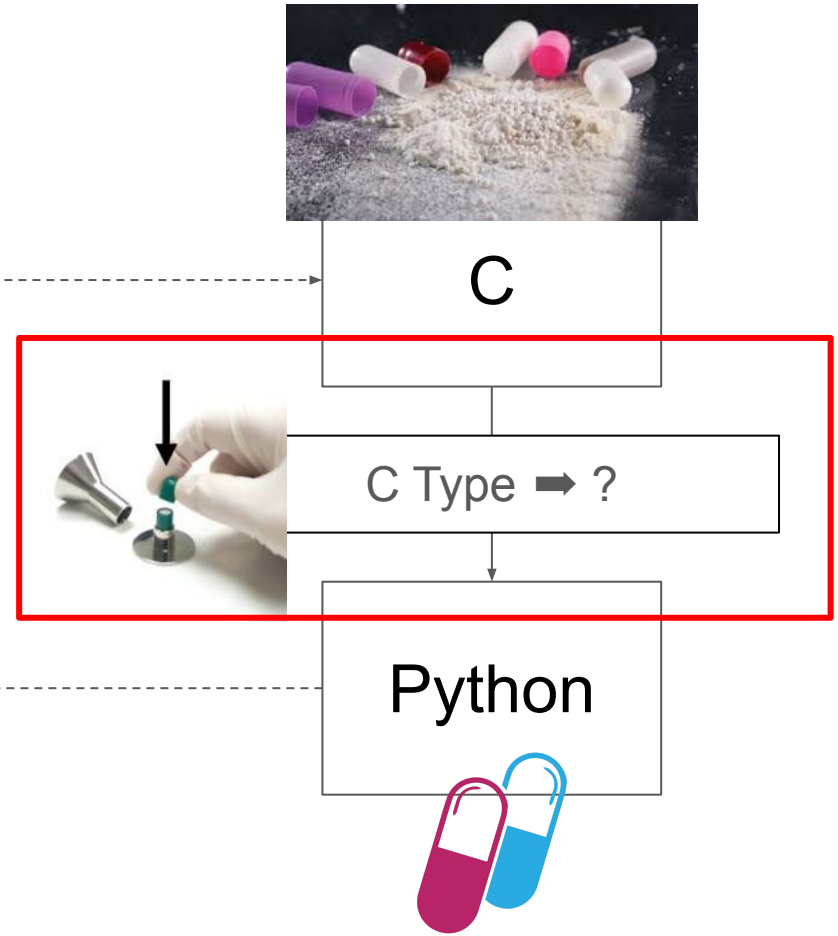
matcher - header

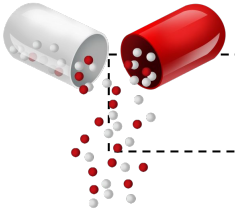
```
struct Matcher {
public:
    virtual bool match(std::string a, std::string b) const = 0;
    virtual ~Matcher() = default;

    // Pre-defines matchers.
    static const Matcher& EXACTLY;
    static const Matcher& PARTIALLY;
}
```

matcher - implementation

```
struct Exactly : public Matcher {
  bool match(std::string a, std::string b) const override {
    return a == b;
  }
};
static Exactly ExactlyObject;
const Matcher& Matcher::EXACTLY(ExactlyObject);

struct Partially : public Matcher {
  bool match(std::string a, std::string b) const override {
    // Returns true if a contains b or vice versa.
    return a.find(b) != std::string::npos or b.find(a) != std::string::npos;
  }
};
static Partially PartiallyObject;
const Matcher& Matcher::PARTIALLY(PartiallyObject);
```

Workflow - capsule





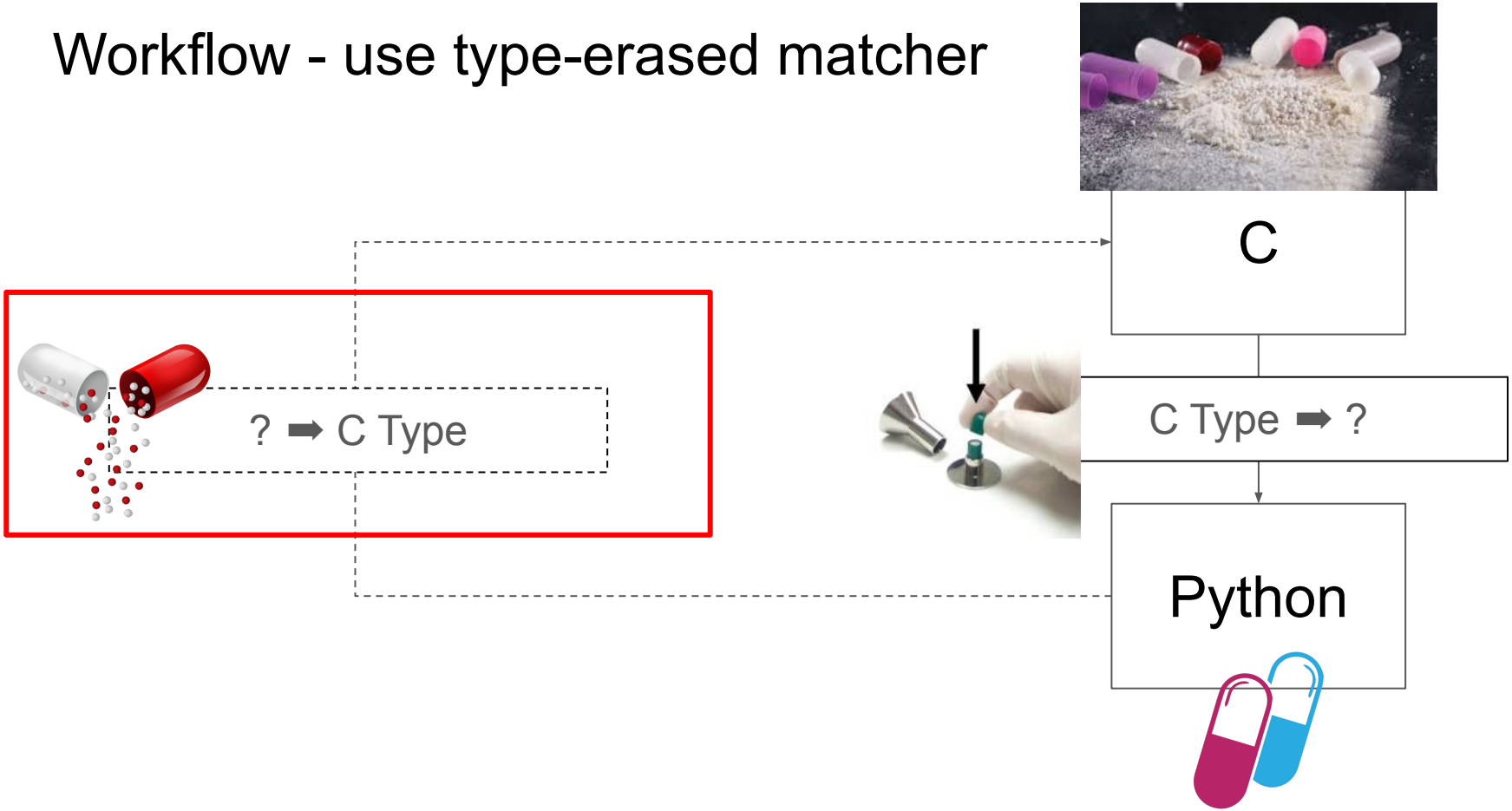
Capsule creator

```
// Defines the name and context for the type erased matcher object.
inline const char MATCHER_NAME[] = "::namespace::matcher";
inline void *MATCHER_CONTEXT = malloc(1);

static PyObject *CreateMatcherCapsule(void *vptr) {
    // A helper function to create matcher capsule.
    PyObject *capsule = PyCapsule_New(vptr, MATCHER_NAME, nullptr);
    PyCapsule_SetContext(capsule, MATCHER_CONTEXT);
    return capsule;
}
```

def get_matcher_xxx() -> object

```
static PyObject* get_matcher_exactly(PyObject* self, PyObject *unused) {
    void *vptr = const_cast<void *>(static_cast<const void *>(
        &(matcher::Matcher::EXACTLY)));
    return matcher::CreateMatcherCapsule(vptr);
}

static PyObject* get_matcher_partially(PyObject* self, PyObject *unused) {
    void *vptr = const_cast<void *>(static_cast<const void *>(
        &(matcher::Matcher::PARTIALLY)));
    return matcher::CreateMatcherCapsule(vptr);
}
```

Workflow - use type-erased matcher



def is_matcher(matcher: object) -> bool

```
static PyObject* is_matcher(PyObject* self, PyObject *args) {
    PyObject* pymatcher;
    if (!PyArg_ParseTuple(args, "O", &pymatcher)) {
        return NULL;
    }
    return PyCapsule_IsValid(pymatcher, matcher::MATCHER_NAME) != 0 &&
        PyCapsule_GetContext(pymatcher) == matcher::MATCHER_CONTEXT ?
        Py_True : Py_False;
}
```

def match(s1: str, s2: str, matcher: object) -> bool

```
static PyObject* match(PyObject* self, PyObject* args) {
    char *a, *b;
    PyObject* pymatcher;
    if (!PyArg_ParseTuple(args, "ssO", &a, &b, &pymatcher)) {
        return NULL;
    }
    void* capsule_payload_matcher = PyCapsule_GetPointer(
        pymatcher, matcher::MATCHER_NAME);
    matcher::Matcher* matcher = static_cast<matcher::Matcher*>(
        capsule_payload_matcher);

    std::string sa(a);
    std::string sb(b);
    return matcher->match(sa, sb) ? Py_True : Py_False;
}
```

Unit test: functionality

```
def setUp(self):
    self.exactly_matcher = matcher.get_matcher_exactly()
    self.partially_matcher = matcher.get_matcher_partially()

def test_is_matcher(self):
    self.assertTrue(
        matcher.is_matcher(self.exactly_matcher))

def test_match_by_matcher_exactly(self):
    self.assertTrue(
        matcher.match('apple', 'apple', self.exactly_matcher))

def test_match_by_matcher_partially(self):
    self.assertTrue(
        matcher.match('apple', 'applepie', self.partially_matcher))
```

Reference count matters

Unit test: reference count in Python

```
def setUp(self):
    self.exactly_matcher = matcher.get_matcher_exactly()

def test_matcher_reference_count(self):
    first_count = sys.getrefcount(self.exactly_matcher)
    matcher.match('apple', 'banana', self.exactly_matcher)
    second_count = sys.getrefcount(self.exactly_matcher)

    self.assertTrue(first_count == second_count)
```

Memory safety matters

C Type ➡ ? ➡ C Type (Type erasure in Python)

- You are a CPython extension module developer
  - Your program in C assures the C type's memory safety

PyObject* ➡ ? ➡ PyObject* (Type erasure in C 🤨) Full example @ Github

- You are a Python developer
  - The module you use should handle the PyObject*'s memory safety
- You are a CPython extension module developer
  - Consider known solutions such as pybind11's object
  - Implement your own PyObjectWrapper to wrap the PyObject* and handle reference count
  - Store the PyObject* in Python's built-in containers (eg: PyDict, PyList, …)

More?

Check note35/TypeErasure-Learning for today's code example!

Thank you!
Any questions?

Credit

Special thanks for Ralf W. Grosse-Kunstleve

- Revising this slide
- Educating me to learn CLIF

This slide is not part of the recording.