Flutter における FFI

FFI ?

Foreign function interface

今回は C++/C の呼び出しの話

話すこと

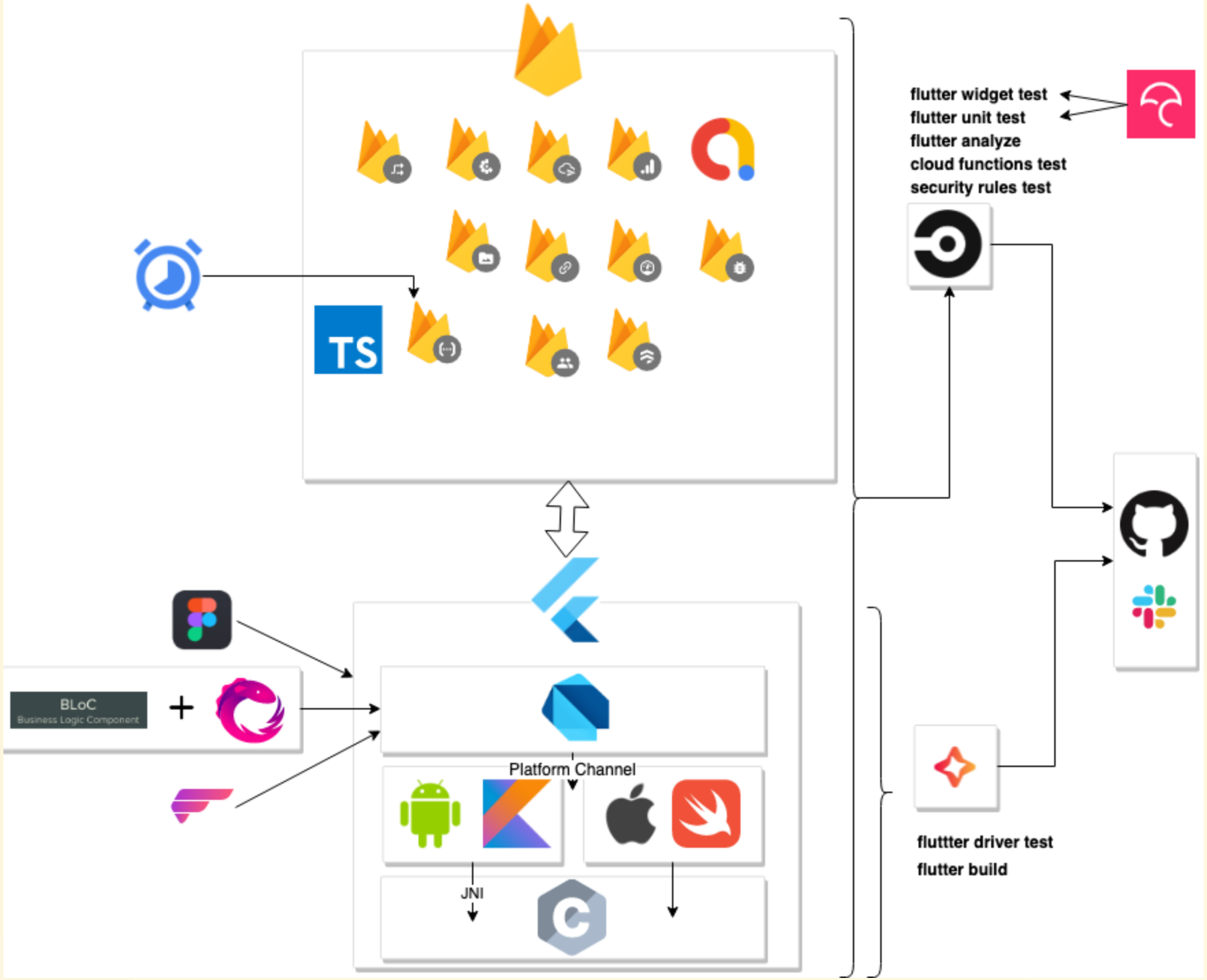
- Dart, Flutter で FFI どうやるか
- (Flutter の) FFI は何が難しいか

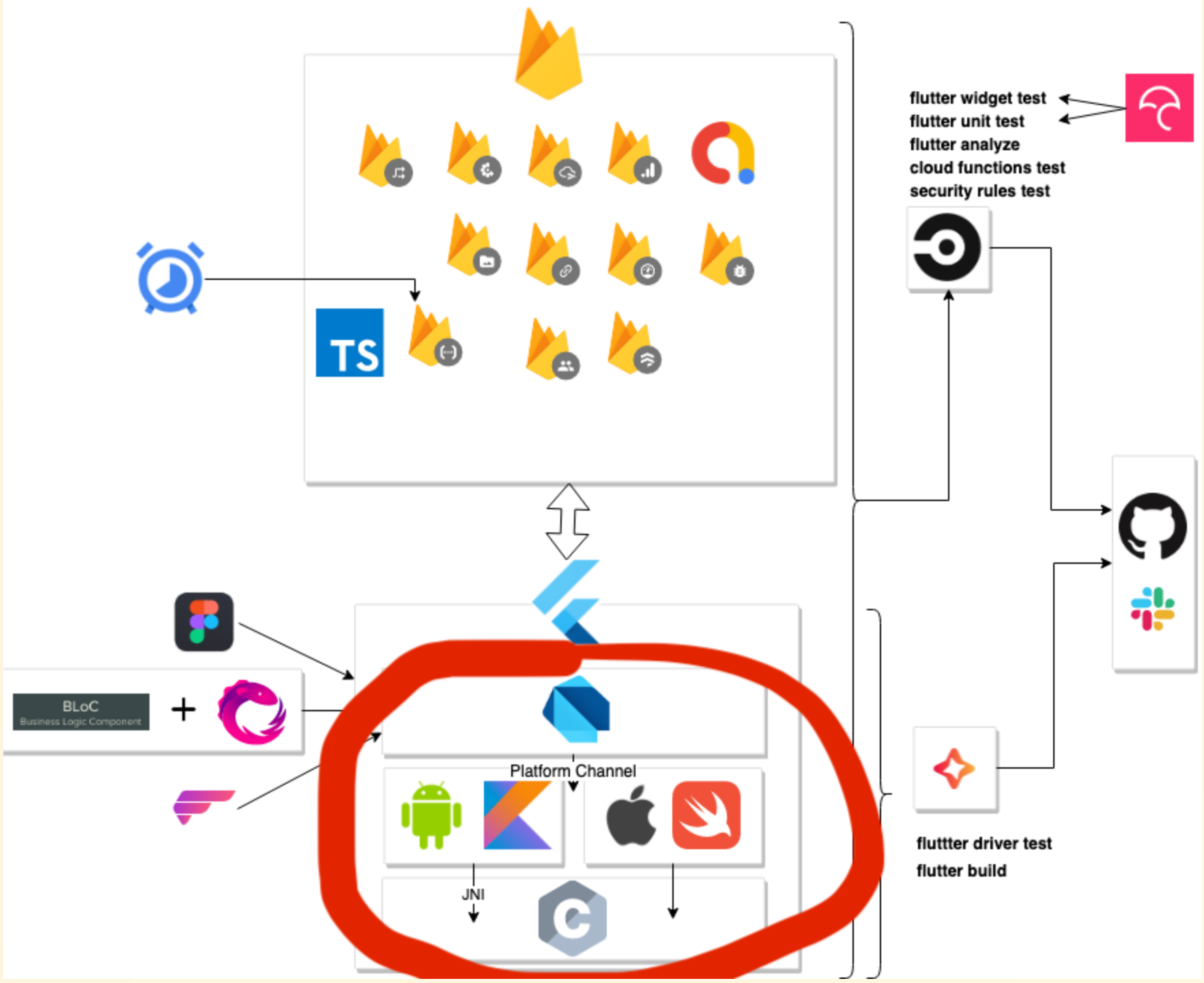
自己紹介

サーバ

オセロ

オセロには常にCが必要


flutter widget test
flutter unit test
flutter analyze
cloud functions test
security rules test
TS
BLoC
Business Logic Component
Platform Channel
JNI
fluttter driver test
flutter build


flutter widget test
flutter unit test
flutter analyze
cloud functions test
security rules test
TS
BLoC
Business Logic Component
Platform Channel
JNI
fluttter driver test
flutter build

各言語のＣ呼び出し

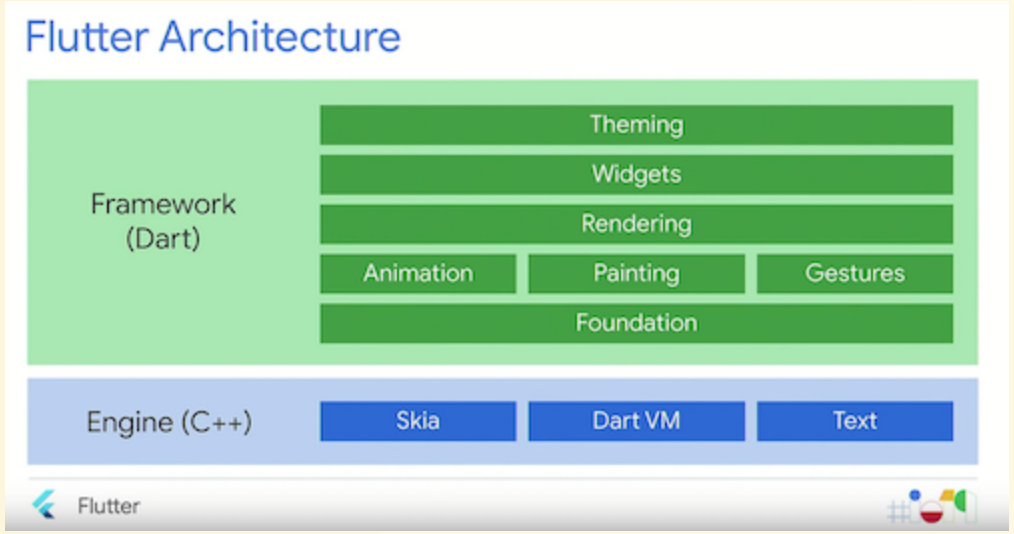
言語	実装方法
C++	`extern "C"` で C++ の名前マングリングを無効にできる。
Java	JNI や JNA, SWIG を使う
Python	ctypes や cffi を使う
Rust	extern キーワードで容易に呼べる
Ruby	Ruby-FFI を使う
Javascript	WebAssembly を使う
Swift	そのままいけるし、カスタムも可能

例: Go -> C

```
package main

/*
#include <stdlib.h>
#include <stdio.h>

void hello() {
    printf("Hello\n");
}
*/
import "C"

func main() {
    C.hello()
}
```

Dart は？

Google I/O'19 でも言及あり



> We are working on a new foreign function interface. This should help you reuse existing C and C++ code, which is important for some critical stuff

① Native Extension

② dart : ffi

① Native Extension

C++ 側

```cpp
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include "include/dart_api.h"
#include "include/dart_native_api.h"

Dart_NativeFunction ResolveName(Dart_Handle name, int argc, bool* auto_setup_scope);

DART_EXPORT Dart_Handle sample_hello_Init(Dart_Handle parent_library) {
  if (Dart_IsError(parent_library)) return parent_library;
  Dart_Handle result_code = Dart_SetNativeResolver(parent_library, ResolveName, NULL);
  if (Dart_IsError(result_code)) return result_code;
  return Dart_Null();
}

void hello(Dart_NativeArguments arguments) {
  Dart_EnterScope();
  printf("Hello\n");
  Dart_ExitScope();
}

Dart_NativeFunction ResolveName(Dart_Handle name, int argc, bool* auto_setup_scope) {
  if (!Dart_IsString(name) || auto_setup_scope == NULL) return NULL;
  Dart_EnterScope();
  const char *cname;
  Dart_StringToCString(name, &cname);
  Dart_NativeFunction result = NULL;
  if (strcmp(cname, "Hello") == 0) result = Hello;
  Dart_ExitScope();
  return result;
}
```

Dart 側

```
library sample_hello;
import 'dart-ext:sample_hello';
void hello() native "Hello";
```

参考: dart-lang sample_extension

② dart:ffi

> The extension mechanism discussed in this page is for deep integration of the VM.
> If you just need to call existing code written in C or C++, see C & C++ interop using FFI.

引用元: Native extensions for the standalone Dart VM

```dart
import "dart:ffi" as ffi;
import 'dart:io' show Platform;

void main() {
  final libHelloWorld = ffi.DynamicLibrary.open(
        "./libHelloWorld.dylib");
  final helloWorld = libHelloWorld.lookupFunction
        <ffi.Void Function(), void Function()>("helloWorld");

  helloWorld();
}
```

https://github.com/sensuikan1973/Dart_FFI_Hello_World

さて、Flutterでは？

dart : ffi のサポートが進んでいる

Dart VM FFI VISION について

そもそも FFI の実装て何が難しいの？

Android では dart:ffi がすでに使える

Flutter における FFI の展望

ありがとうございました

参考

- Dart VM FFI Vision
  - Design and implement Dart VM FFI
  - Flutter Support integrating with C/C++ in plugin framework
  - Native extensions for the standalone Dart VM
  - Support for Dart Extensions
- C & C++ interop using FFI
  - Dart Native platform
  - dart:ffi sqllite sample
  - sensuikan1973/Dart_FFI_Hello_World
- The Engine architecture
  - Writing custom platform-specific code
  - Custom Flutter Engine Embedders
- sensuikan1973/flutter-ffi-slide