flight

flight

NDK Technical Deep-Dive: Now Gluten Free

Konstantin Mandrika
Engineer, Fabric | @kmandrika

Java Crash Handling

```
Thread.setDefaultUncaughtExceptionHandler(handler);
```

flight

```java
public class Handler implements UncaughtExceptionHandler {
    @Override
    public void uncaughtException(Thread thread, Throwable ex) {
    }
}
```

flight

```
java.lang.RuntimeException: I messed up!
    at java.awt.AWTEventMulticaster.componentShown(AWTEventMulticaster.java:162)
    at java.awt.AWTEventMulticaster.componentShown(AWTEventMulticaster.java:162)
    at java.awt.Component.processComponentEvent(Component.java:6246)
    at java.awt.Component.processEvent(Component.java:6194)
    at java.awt.Container.processEvent(Container.java:2084)
    at java.awt.Component.dispatchEventImpl(Component.java:4776)
    at java.awt.Container.dispatchEventImpl(Container.java:2142)
    at java.awt.Component.dispatchEvent(Component.java:4604)
    at java.awt.EventQueue.dispatchEventImpl(EventQueue.java:717)
    at java.awt.EventQueue.access$400(EventQueue.java:82)
    at java.awt.EventQueue$2.run(EventQueue.java:676)
    at java.awt.EventQueue$2.run(EventQueue.java:674)
    at java.security.AccessController.doPrivileged(Native Method)
    at java.security.AccessControlContext$1.doIntersectionPrivilege(AccessControlContext.java:86)
    at java.security.AccessControlContext$1.doIntersectionPrivilege(AccessControlContext.java:97)
    at java.awt.EventQueue$3.run(EventQueue.java:690)
    at java.awt.EventQueue$3.run(EventQueue.java:688)
    at java.security.AccessController.doPrivileged(Native Method)
    at java.security.AccessControlContext$1.doIntersectionPrivilege(AccessControlContext.java:86)
    at java.awt.EventQueue.dispatchEvent(EventQueue.java:687)
    at java.awt.EventDispatchThread.pumpOneEventForFilters(EventDispatchThread.java:296)
    at java.awt.EventDispatchThread.pumpEventsForFilter(EventDispatchThread.java:211)
    at java.awt.EventDispatchThread.pumpEventsForHierarchy(EventDispatchThread.java:201)
    at java.awt.EventDispatchThread.pumpEvents(EventDispatchThread.java:196)
    at java.awt.EventDispatchThread.pumpEvents(EventDispatchThread.java:188)
    at java.awt.EventDispatchThread.run(EventDispatchThread.java:122)
```

flight

Uncaught exceptions
do not crash the JVM

Native Crash Handling

C

```c
void format_the_clients_hdd()
{
    int* x = NULL;
    *x = 42;
}
```

C++

```cpp
void format_the_clients_hdd()
{
    throw new std::runtime_error(“uh-oh!”);
}
```

flight

The most broad strategy is to install a **signal** handler

```
signal::sigaction_t action = {};
...
action.sa_flags = SA_SIGINFO;
action.sa_sigaction = make_invocation_wrapper(
        std::bind(signal::detail::restore_handlers, saved_handlers),
        std::bind(signal::handler, unwinder, handler_context, _1, _2, _3)
);
…
sigaction(SIGSEGV, action, &previous);
```

flight

Very little **information** is passed into the handler

```
void handler(int signum, siginfo_t* info, void* context)
{
    …
}
```

Additionally, there is a list of **constraints** to consider…

Additionally, there is a list of **constraints** to consider…

Async-Safety

Static Storage Duration

flight

Async-
Safety

All function calls must be **reentrant**

flight

Async-
Safety

Can’t allocate any dynamic or static memory…

flight

Static Storage Duration

… and can’t pass state into the handler

flight

Just **one** though!

flight

```
volatile sig_atomic_t signal; // At global scope
```

This is something that ~~most~~ all crash handlers **violate**, including us

~~sig_atomic_t~~

std::atomic<state *>

Turns out we also have to manually **unwind** the stack

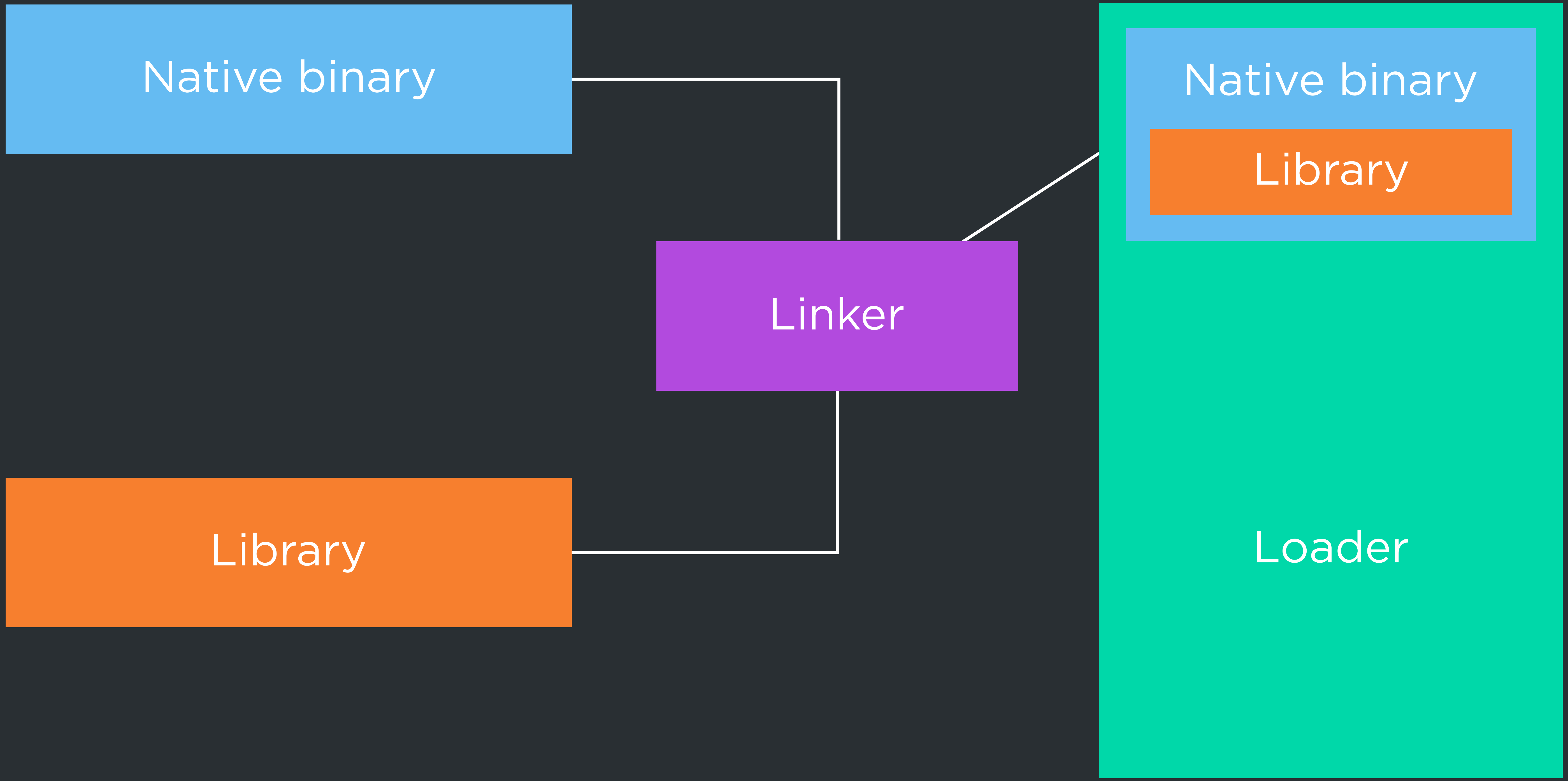
Luckily, some Android API levels ship with unwinding libraries

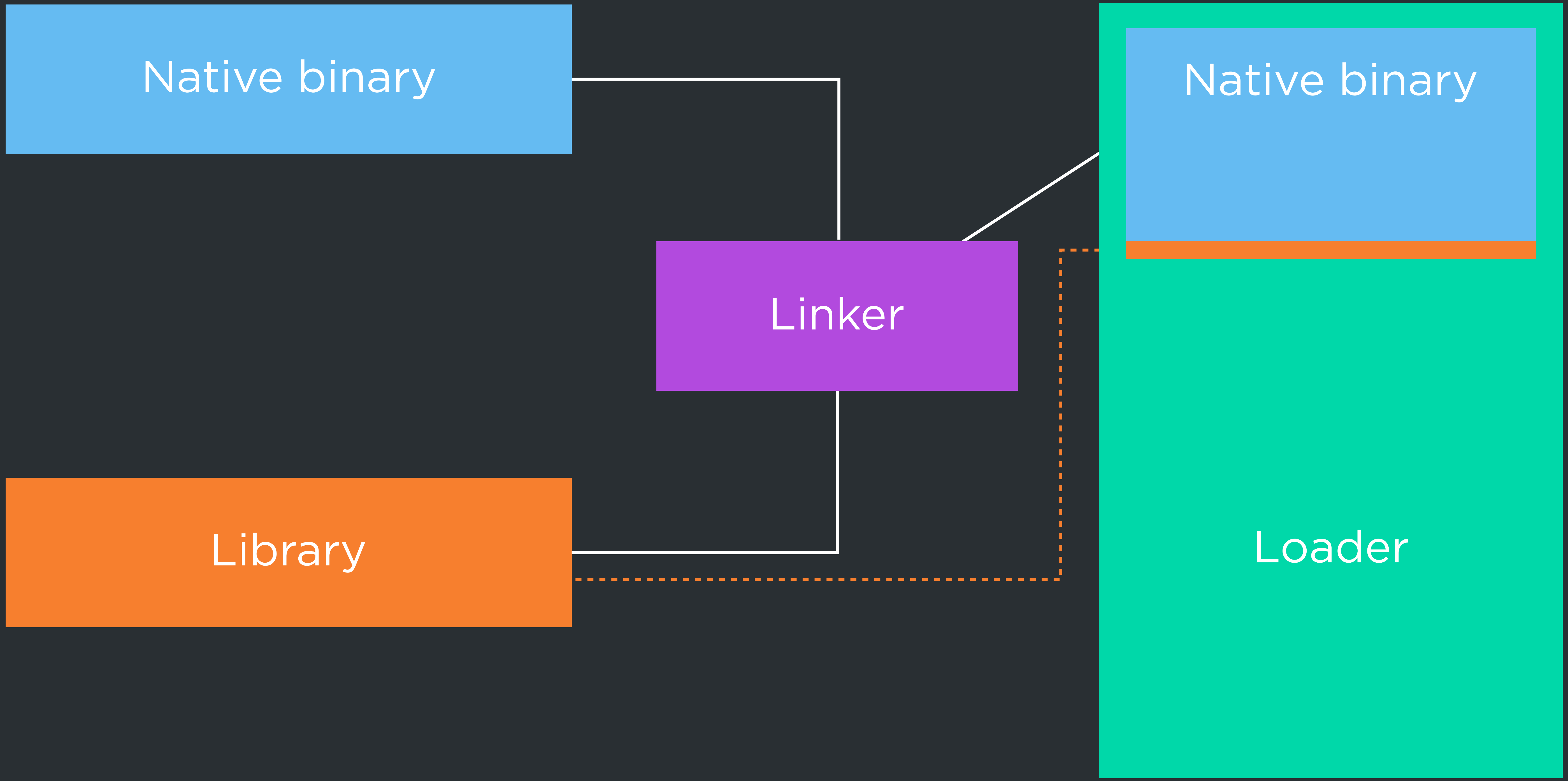
flight

Types Of Linking

STATIC LINKING

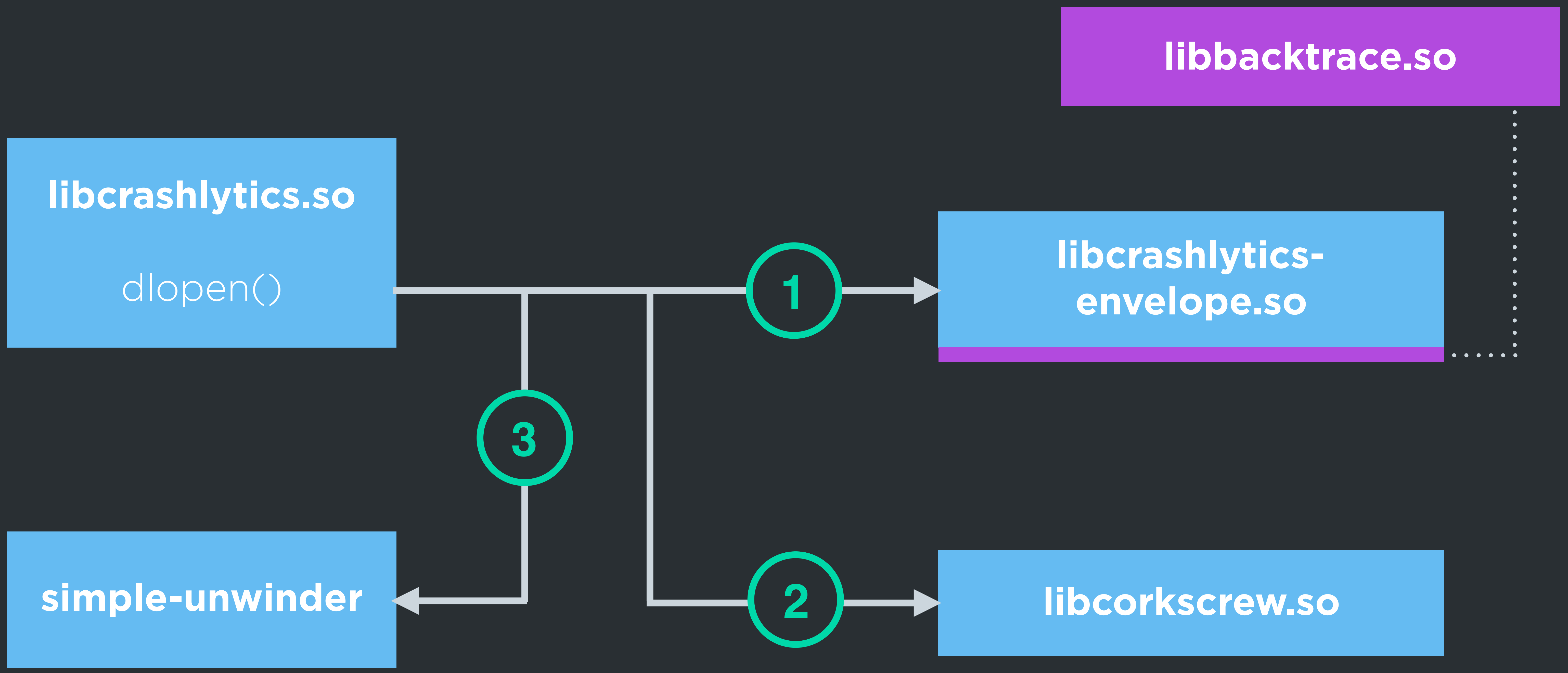
DYNAMIC LINKING


Native binary
Library
Linker
Native binary
Library
Loader



flight

```
dlopen();
```


libbacktrace.so
libcrashlytics.so
dlopen()
1
libcrashlytics-envelope.so
3
simple-unwinder
2
libcorkscrew.so

flight

```
ssize_t impl::simple::unwind_impl(
    pid_t pid, pid_t tid, frames_t& frames, siginfo_t* siginfo, void* context
)
{
    if (detail::is_crashed_thread(pid, tid)) {
        frames[0].pc = static_cast<std::make_unsigned<greg_t>::type>(
            detail::pc_from_context(reinterpret_cast<ucontext_t *>(context))
        );
        return 1;
    }

    return 0;
}
```

flight

```
inline bool dladdr(greg_t pc, Dl_info& dl_info)
{
    return ::dladdr(reinterpret_cast<void *>(pc), &dl_info) != 0;
}

inline const char* symbolicate(greg_t pc)
{
#if defined (CRASHLYTICS_ON_DEVICE_SYMBOLICATION)
    greg_t pc_normalized = normalize(pc);

    Dl_info dl_info = { NULL, NULL, NULL, NULL };
    if (dladdr(pc_normalized, dl_info) && dl_info.dli_sname != NULL) {
        return dl_info.dli_sname;
    }
#endif
    return "";
}
```

flight

Collected information is **dumped** to a file tl;dr - native crash handling is much **trickier** than its Java equivalent

Thank You

@kmandrika

flight