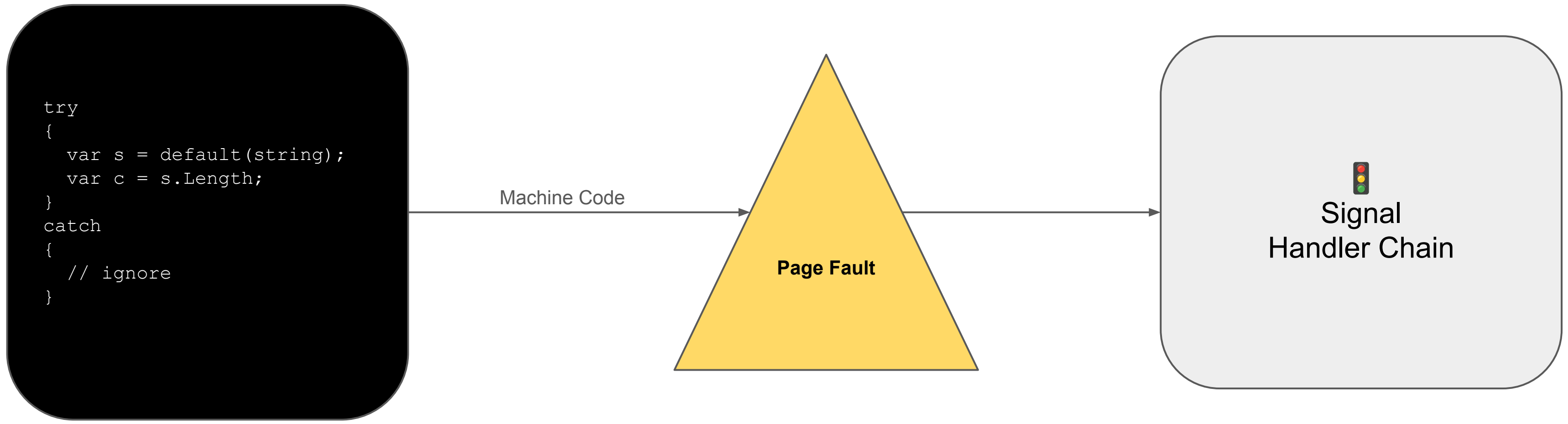
`NullReferenceException` in .NET applications compiled to Native code

From detailed analysis by supervacuus [here]

Native Compilation



```
try
{
  var s = default(string);
  var c = s.Length;
}
catch
{
  // ignore
}
```

Signal Handler Chain

Since the Native SDK installs its signal handler last, it will be the first in the signal chain.

- Other handlers
- .NET SDK
- Native SDK

↑ Order of execution

Signal Handler Chain

Since the Native SDK installs its signal handler last, it will be the first in the signal chain.

- Other handlers
- .NET SDK
- Native SDK

↑ Order of execution

SIGSEGV reported as native crash
(unaware it's running in the CLR)

Signal Handler Chain

Since the Native SDK installs its signal handler last, it will be the first in the signal chain.

- Other handlers
- .NET SDK
- Native SDK

↑ Order of execution

Raises a managed code exception (NullReferenceException)

Signal Handler Chain

Since the Native SDK installs its signal handler last, it will be the first in the signal chain.

- ~~Other handlers~~
- .NET SDK
- Native SDK

Discontinues Signal Chain
(not reported in logcat)

↑ Order of execution

Attempted Solution

Theory

- Invoke the dotnet runtime handler before the NDK handler
- NDK handler would never execute for handled managed code exceptions
  - Only invoked if the runtime handler continues the signal chain (unintended CLR crash or native code crash)

- ~~Other handlers~~
- ~~Native SDK~~
- .NET SDK

↑ Order of execution

Discontinues Signal Chain

Attempted Solution

Reality

- Invoke the dotnet runtime handler before the NDK handler
- NDK handler still executes (even for handled managed code exceptions)

Unhandled SIGSEGV crashes the application ⚠️

- Other handlers
- Native SDK
- .NET SDK

↑ Order of execution