Unwinding the Stack:

Exploring how C++ Exceptions work on Windows

JAMES MCNELLIS

PRINCIPAL SOFTWARE ENGINEER

MICROSOFT WINDOWS DEBUGGERS

JAMES@JAMESMCNELLIS.COM

```cpp
throw std::runtime_error{"oh no"};
```

```cpp
try
{
    // ...dangerous things...
}
catch (std::exception const& ex)
{
    std::cout << ex.what() << '\n';
}
```

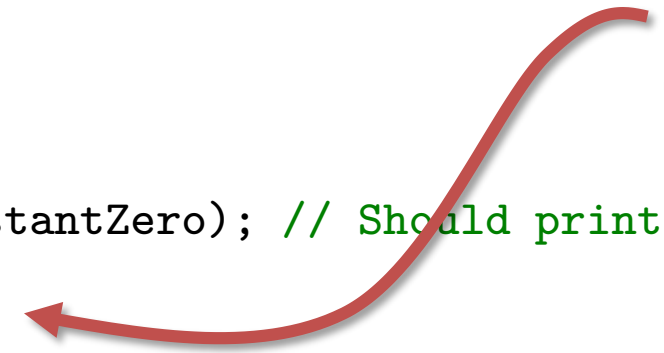
C++ Exceptions

```
__try
{
    // ...dangerous things...
}
__except (MyExceptionFilter(GetExceptionInformation()))
{
    puts("oh no");
}
```

Structured Exceptions

What Happens When Something Goes Wrong?

```
int const ConstantZero = 0;

void MyWonderfulProgram()
{
    printf("ConstantZero is %d\n", ConstantZero); // Should print "ConstantZero is 0"

    ConstantZero = 1;

    printf("ConstantZero is %d\n", ConstantZero); // Should print "ConstantZero is 0"
}
```



error C3892: 'ConstantZero': you cannot assign to a variable that is const, stupid

```
int const ConstantZero = 0;

void MyWonderfulProgram()
{
    printf("ConstantZero is %d\n", ConstantZero);

    const_cast<int&>(ConstantZero) = 1;

    printf("ConstantZero is %d\n", ConstantZero);
}
```

Once Upon A Time…

```
A:\>
```

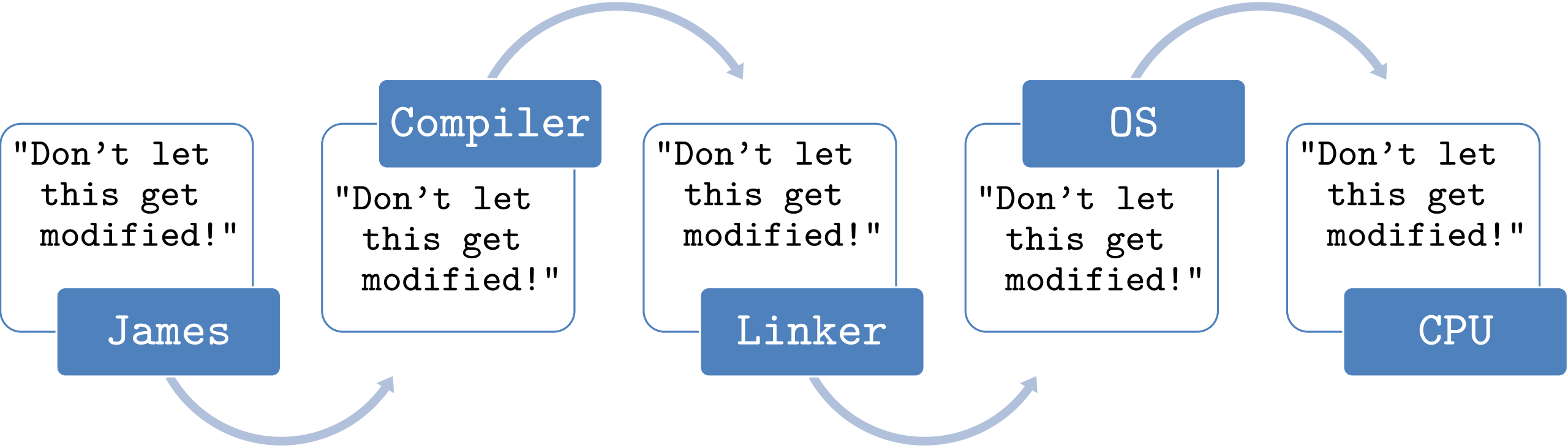
Once Upon A Time…

```
A:\>MyWonderfulProgram.exe
ConstantZero is 0

A:\>
```

Hey, where did my second message go?

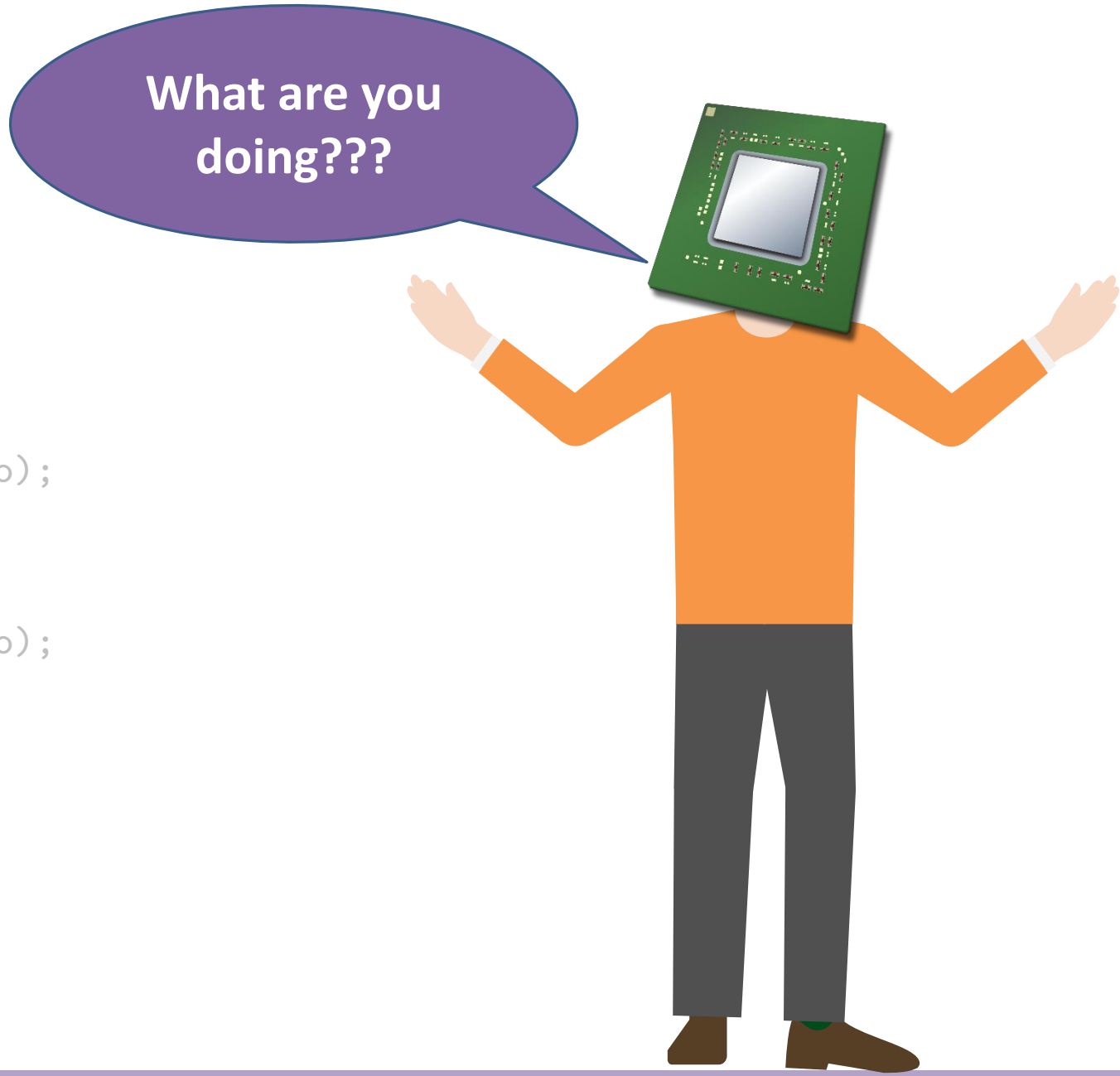
```
A:\>MyWonderfulProgram.exe
ConstantZero is 0

A:\>echo %errorlevel%
-1073741819

A:\>
```

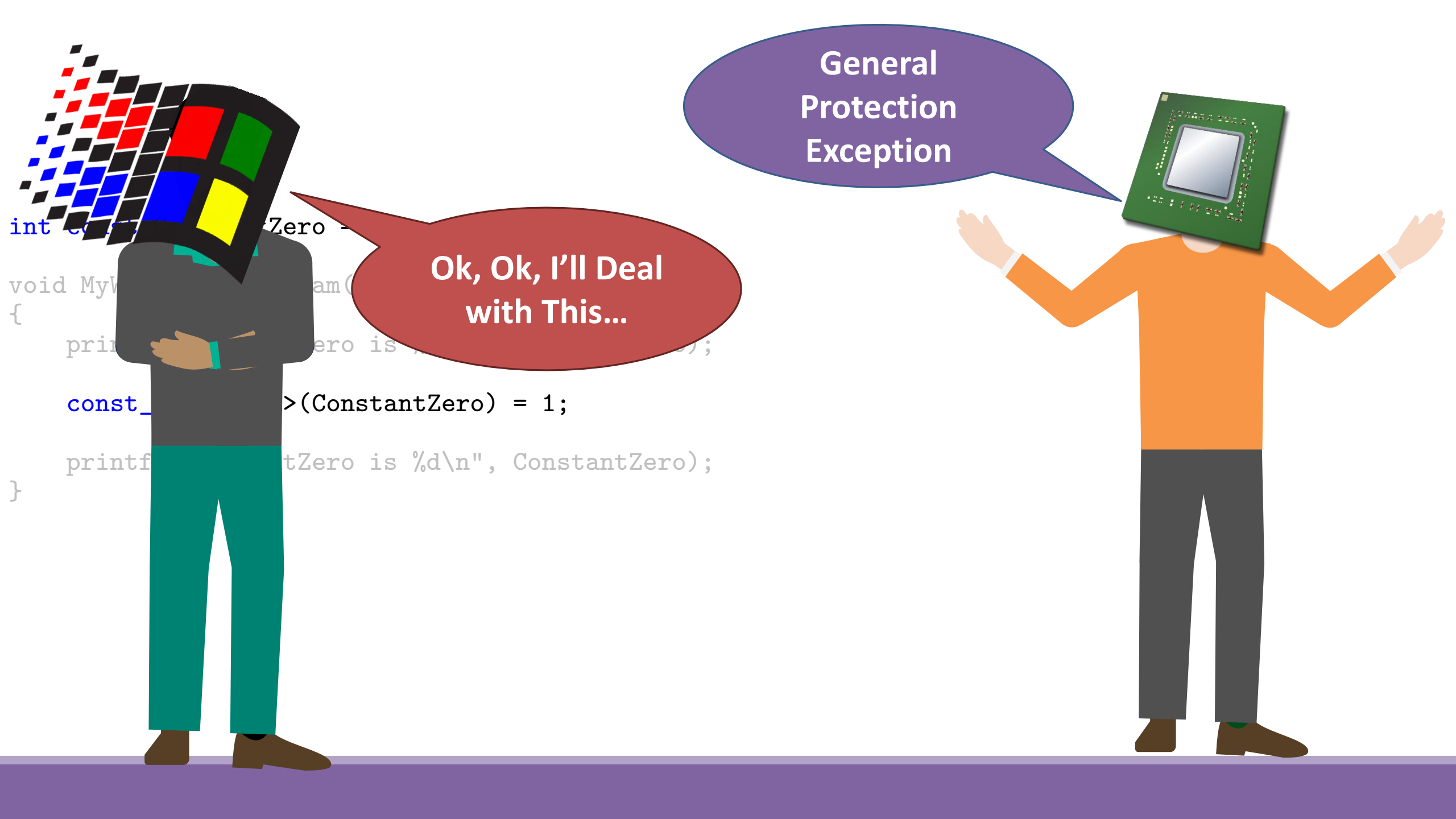
```
int const ConstantZero;
```



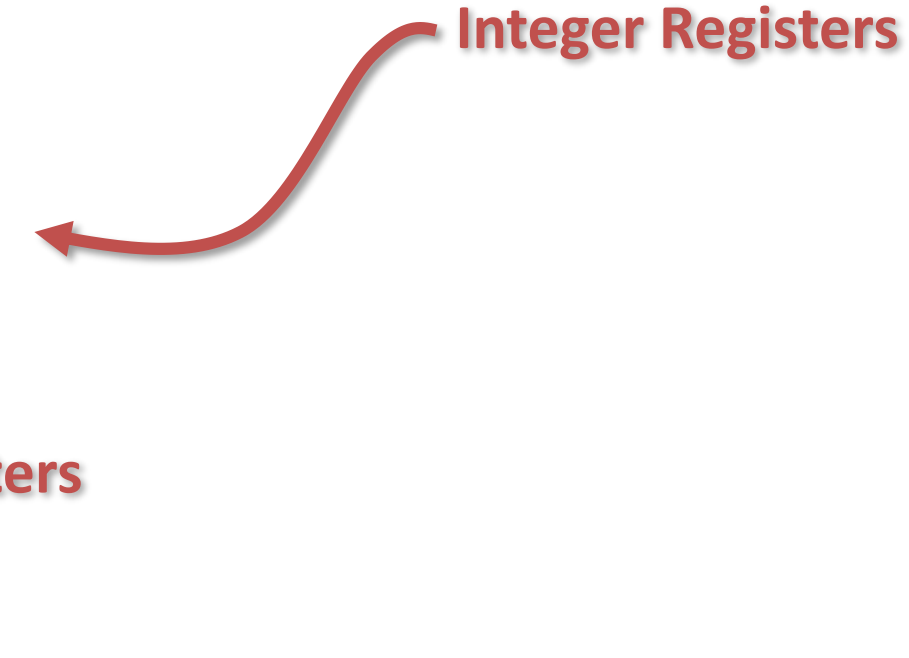
What Happened?

```
int const ConstantZero = 0;

void MyWonderfulProgram()
{
    printf("ConstantZero is %d\n", ConstantZero);

    const_cast<int&>(ConstantZero) = 1;

    printf("ConstantZero is %d\n", ConstantZero);
}
```




General Protection Exception
Ok, Ok, I'll Deal with This...
int
Zero
void My
am(
{
pri
ero is
);
const_
>(ConstantZero) = 1;
printf
tZero is %d\n", ConstantZero);
}

```
struct CONTEXT
{
    DWORD ContextFlags;

    DWORD Dr0, Dr1, Dr2, Dr3, Dr6, Dr7;

    FLOATING_SAVE_AREA FloatSave;

    DWORD SegGs, SegFs, SegEs, SegDs;

    DWORD Edi, Esi, Ebx, Edx, Ecx, Eax;

    DWORD Ebp;
    DWORD Eip;
    DWORD SegCs;
    DWORD EFlags;
    DWORD Esp;
    DWORD SegSs;

    BYTE  ExtendedRegisters[MAXIMUM_SUPPORTED_EXTENSION];
};
```



CONTEXT

```
#define EXCEPTION_MAXIMUM_PARAMETERS 15

struct EXCEPTION_RECORD
{
    DWORD                    ExceptionCode;
    DWORD                    ExceptionFlags;
    EXCEPTION_RECORD*        ExceptionRecord;
    void*                    ExceptionAddress;
    DWORD                    NumberParameters;
    ULONG_PTR                ExceptionInformation[EXCEPTION_MAXIMUM_PARAMETERS];
};
```

EXCEPTION_RECORD

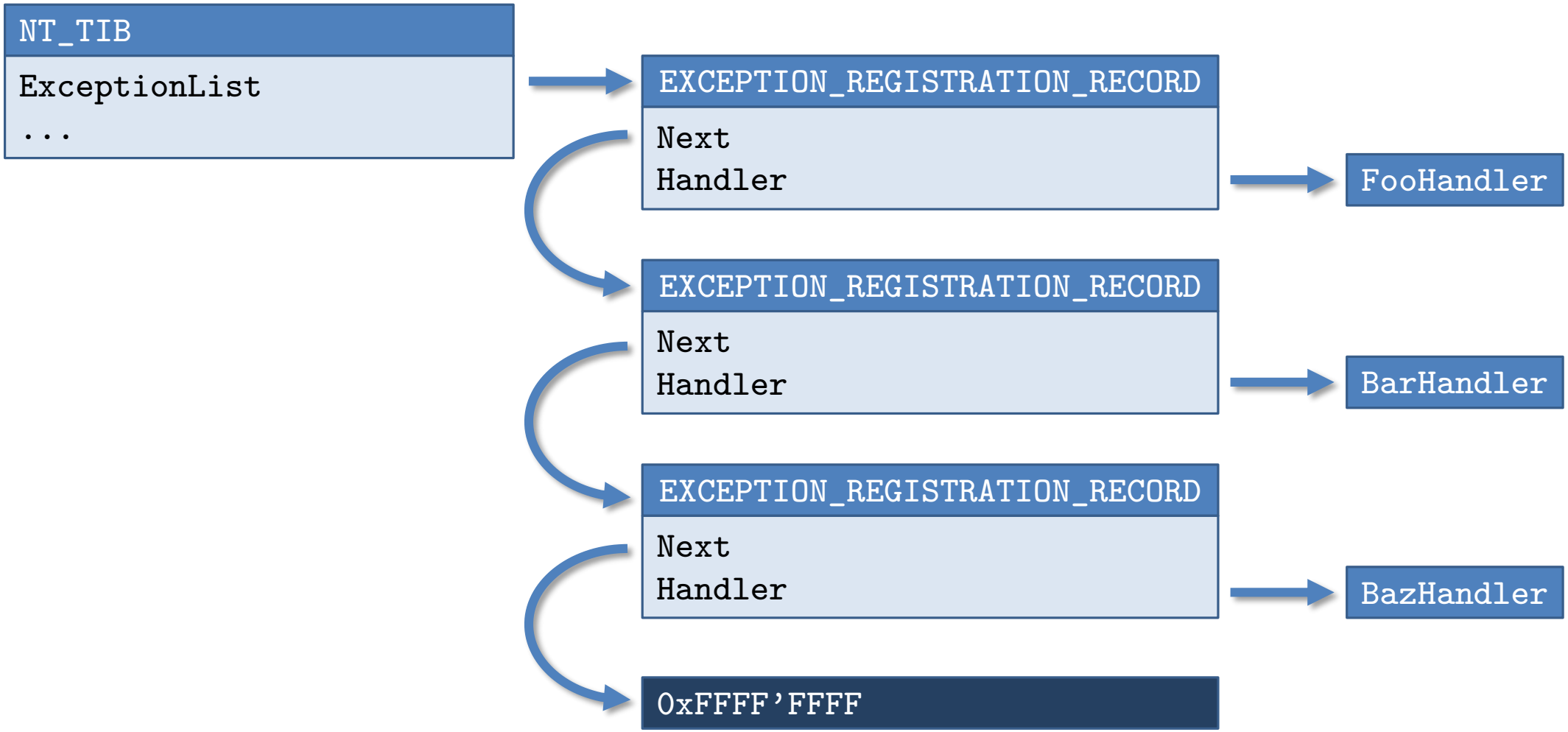
```
EXCEPTION_RECORD Exception;

Exception.ExceptionCode             = STATUS_ACCESS_VIOLATION;
Exception.ExceptionFlags            = 0;
Exception.ExceptionRecord           = nullptr;
Exception.ExceptionAddress          = 0x010D21CD;
```

EXCEPTION_RECORD

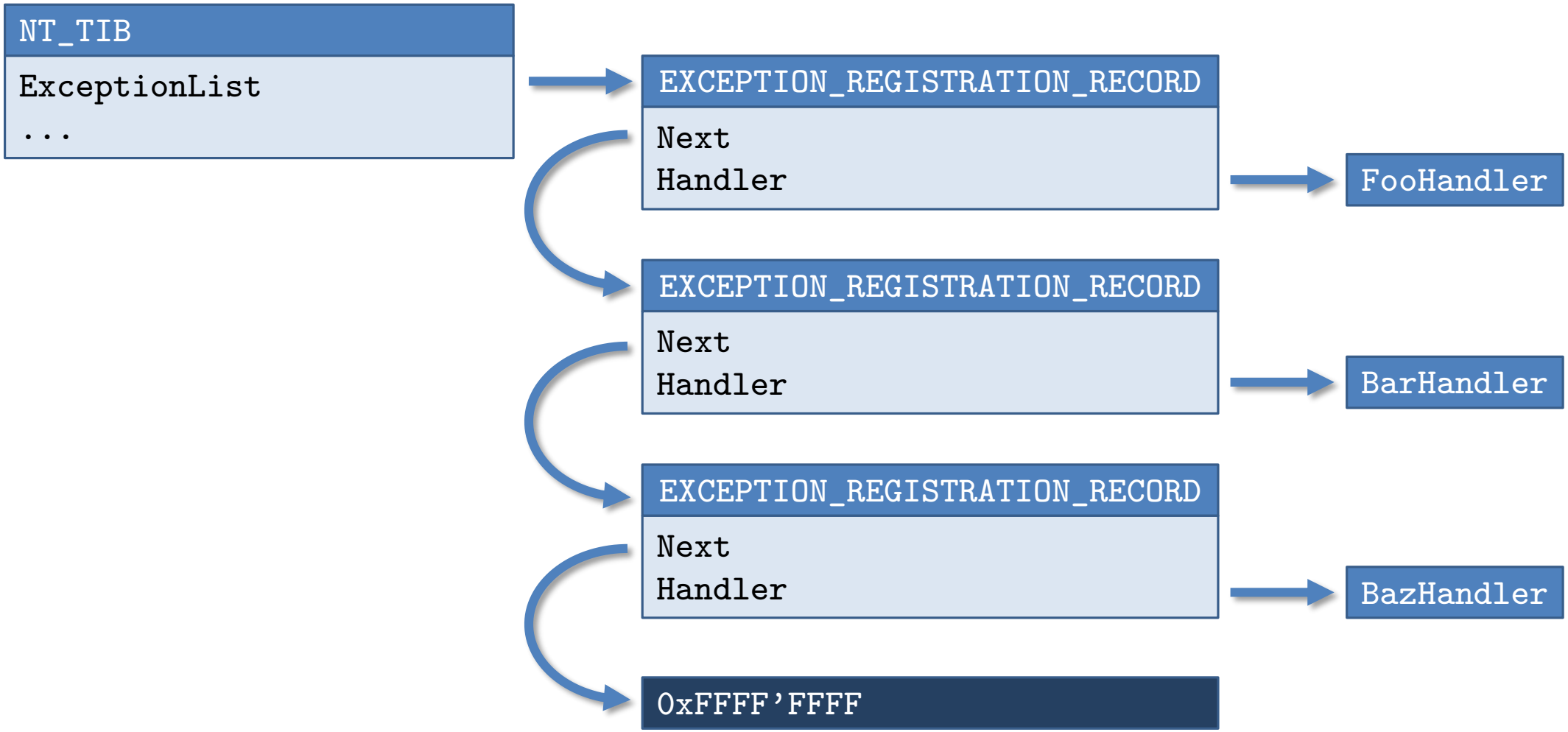
```
EXCEPTION_RECORD Exception;

Exception.ExceptionCode             = STATUS_ACCESS_VIOLATION;
Exception.ExceptionFlags            = 0;
Exception.ExceptionRecord           = nullptr;
Exception.ExceptionAddress          = 0x010D21CD;

Exception.NumberParameters          = 2;
Exception.ExceptionInformation[0] = EXCEPTION_WRITE_FAULT;
Exception.ExceptionInformation[1] = 0x010DAB30;
```

EXCEPTION_RECORD

Making Things Right…



The Thread Information Block's "Exception List"

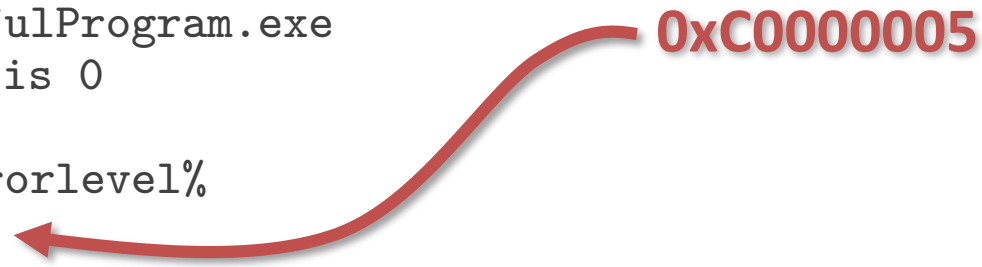


The Thread Information Block's "Exception List"

```
int const ConstantZero = 0;

void MyWonderfulProgram()
{
    printf("ConstantZero is %d\n", ConstantZero);

    const_cast<int&>(ConstantZero) = 1;

    printf("ConstantZero is %d\n", ConstantZero);
}
```

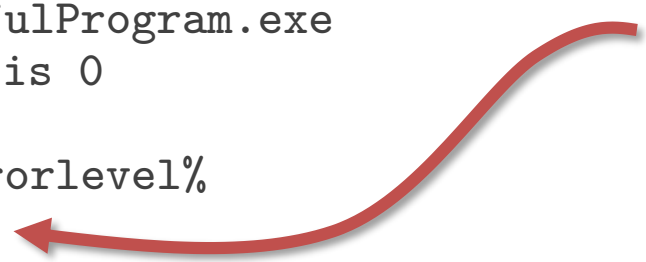
Our Program Doesn’t Register Any Handlers

```
A:\>MyWonderfulProgram.exe
ConstantZero is 0

A:\>echo %errorlevel%
-1073741819

A:\>
```



Our Program Doesn’t Register Any Handlers

```
A:\>MyWonderfulProgram.exe
ConstantZero is 0

A:\>echo %errorlevel%
-1073741819

A:\>
```



0xC0000005 (STATUS_ACCESS_VIOLATION)

Our Program Doesn’t Register Any Handlers

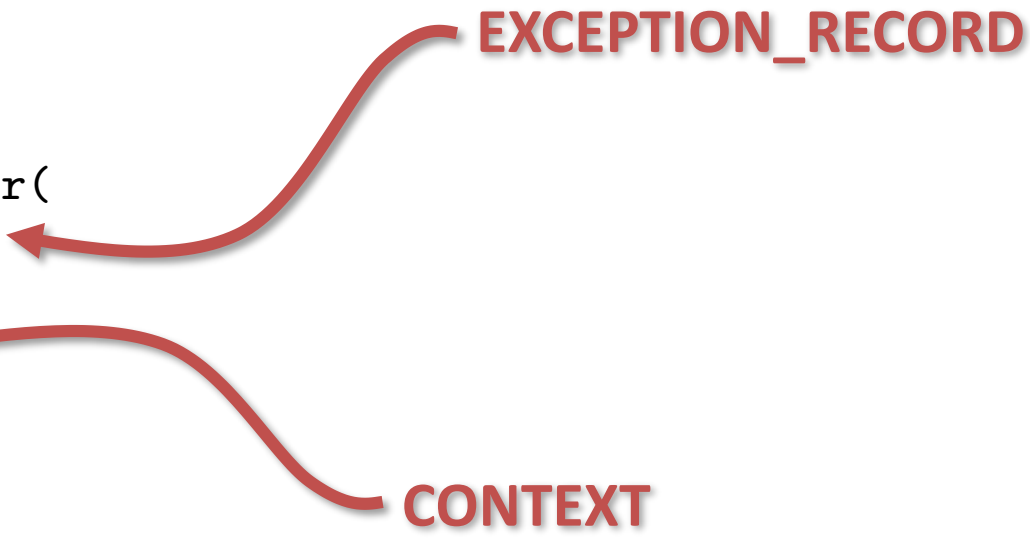
```
void MyWonderfulProgram()
{




    printf("ConstantZero is %d\n", ConstantZero); // Prints "ConstantZero is 0"

    const_cast<int&>(ConstantZero) = 1;

    printf("ConstantZero is %d\n", ConstantZero); // Prints "ConstantZero is 1"


}
```

Let’s Write an Exception Handler

```
void MyWonderfulProgram()
{
    NT_TIB* TIB = (NT_TIB*)NtCurrentTeb();

    EXCEPTION_REGISTRATION_RECORD Registration;
    Registration.Handler = &MyExceptionHandler;
    Registration.Next    = TIB->ExceptionList;
    TIB->ExceptionList = &Registration;

    printf("ConstantZero is %d\n", ConstantZero); // Prints "ConstantZero is 0"

    const_cast<int&>(ConstantZero) = 1;

    printf("ConstantZero is %d\n", ConstantZero); // Prints "ConstantZero is 1"

    TIB->ExceptionList = TIB->ExceptionList->Next;
}
```

Let's Write an Exception Handler

```
EXCEPTION_DISPOSITION MyExceptionHandler(
    EXCEPTION_RECORD* ExceptionRecord,
    void*             EstablisherFrame,
    CONTEXT*          ContextRecord,
    void*             DispatcherContext
    );
```



Let's Write an Exception Handler

```
EXCEPTION_DISPOSITION MyExceptionHandler(
    EXCEPTION_RECORD* ExceptionRecord,
    void*             EstablisherFrame,
    CONTEXT*          ContextRecord,
    void*             DispatcherContext
    )
{
    printf(
        "An exception occurred at address 0x%p, with ExceptionCode = 0x%08x!\n",
        ExceptionRecord->ExceptionAddress,
        ExceptionRecord->ExceptionCode);

    return ExceptionContinueSearch;
}
```

Let's Write an Exception Handler

```
A:\>
```

Let’s Write an Exception Handler

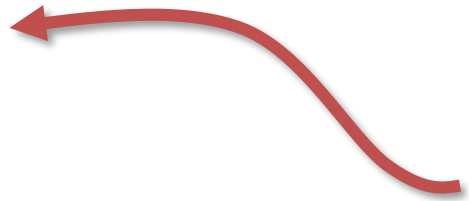
```
A:\>MyWonderfulProgram.exe
ConstantZero is 0
An exception occurred at address 0x00AD2EEC, with ExceptionCode = 0xc0000005!

A:\>
```

Let’s Write an Exception Handler

```
A:\>MyWonderfulProgram.exe
ConstantZero is 0
An exception occurred at address 0x00AD2EEC, with ExceptionCode = 0xc0000005!

A:\>echo %errorlevel%
-1073741819

A:\>
```

Let’s Write an Exception Handler

```
EXCEPTION_DISPOSITION MyExceptionHandler(
    EXCEPTION_RECORD* ExceptionRecord,
    void*             EstablisherFrame,
    CONTEXT*          ContextRecord,
    void*             DispatcherContext
    )
{
    printf(
        "An exception occurred at address 0x%p, with ExceptionCode = 0x%08x!\n",
        ExceptionRecord->ExceptionAddress,
        ExceptionRecord->ExceptionCode);

    return ExceptionContinueSearch;
}
```



We chose not to handle the exception

Let's Write an Exception Handler

```
EXCEPTION_DISPOSITION MyExceptionHandler(
    EXCEPTION_RECORD* ExceptionRecord,
    void*             EstablisherFrame,
    CONTEXT*          ContextRecord,
    void*             DispatcherContext
    )
{
    if (ExceptionRecord->ExceptionCode           != STATUS_ACCESS_VIOLATION ||
        ExceptionRecord->ExceptionInformation[0] != EXCEPTION_WRITE_FAULT)
    {
        return ExceptionContinueSearch; // Not a write access violation
    }

    puts("A write access violation occurred!  Let's see if we can fix it!");

    void* WriteAddress = (void*)ExceptionRecord->ExceptionInformation[1];

    VirtualProtect(WriteAddress, sizeof(int), PAGE_READWRITE);


}
```

Let's "Fix" The Problem

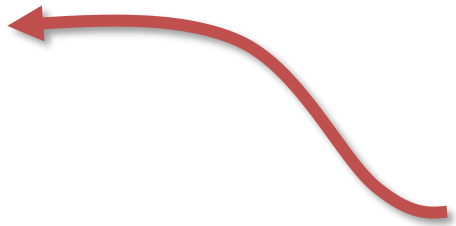
```
EXCEPTION_DISPOSITION MyExceptionHandler(
    EXCEPTION_RECORD* ExceptionRecord,
    void*             EstablisherFrame,
    CONTEXT*          ContextRecord,
    void*             DispatcherContext
    )
{
    if (ExceptionRecord->ExceptionCode           != STATUS_ACCESS_VIOLATION ||
        ExceptionRecord->ExceptionInformation[0] != EXCEPTION_WRITE_FAULT)
    {
        return ExceptionContinueSearch; // Not a write access violation
    }

    puts("A write access violation occurred!  Let's see if we can fix it!");

    void* WriteAddress = (void*)ExceptionRecord->ExceptionInformation[1];

    VirtualProtect(WriteAddress, sizeof(int), PAGE_READWRITE);

    return ExceptionContinueExecution;
}
```

Let’s “Fix” The Problem

```
A:\>
```

Let’s “Fix” The Problem

```
A:\>MyWonderfulProgram.exe
ConstantZero is 0
A write access violation occurred!  Let's see if we can fix it!
ConstantZero is 1

A:\>
```



We got to our second printf!

Let’s “Fix” The Problem

```
A:\>MyWonderfulProgram.exe
ConstantZero is 0
A write access violation occurred!  Let's see if we can fix it!
ConstantZero is 1

A:\>echo %errorlevel%
0

A:\>
```

We got to our second printf!

Our program didn't crash!

Let's "Fix" The Problem

Perhaps The Compiler Would Like To Help…

```
void MyWonderfulProgram()
{
    __try
    {
        // Something "dangerous"
    }
    __except (MyExceptionFilter())
    {

    }
}
```

Guarded Region

Exception Filter

__try/__except

```
void MyWonderfulProgram()
{
    __try
    {
        // Something "dangerous"
    }
    __except (EXCEPTION_CONTINUE_SEARCH)
    {

    }
}
```

__try/__except

```
void MyWonderfulProgram()
{
    __try
    {
        // Something "dangerous"
    }
    __except (EXCEPTION_CONTINUE_EXECUTION)
    {

    }
}
```

__try/__except

```
void MyWonderfulProgram(bool ShouldContinue)
{
    __try
    {
        // Something "dangerous"
    }
    __except(ShouldContinue ? EXCEPTION_CONTINUE_EXECUTION : EXCEPTION_CONTINUE_SEARCH)
    {

    }
}
```

__try/__except

```
struct EXCEPTION_POINTERS
{
    EXCEPTION_RECORD* ExceptionRecord;
    CONTEXT*          ContextRecord;
};

EXCEPTION_POINTERS* GetExceptionInformation();



DWORD GetExceptionCode();
```

__try/__except

```
int MyExceptionFilter(EXCEPTION_POINTERS* Pointers);

void MyWonderfulProgram(bool ShouldContinue)
{
    __try
    {
        // Something "dangerous"
    }
    __except (MyExceptionFilter(GetExceptionInformation()))
    {

    }
}
```

__try/__except

```
int const ConstantZero = 0;


void MyWonderfulProgram()
{
    printf("ConstantZero is %d\n", ConstantZero);

    __try
    {
        const_cast<int&>(ConstantZero) = 1;
    }
    __except (MyExceptionFilter(GetExceptionInformation()))
    {
    }

    printf("ConstantZero is %d\n", ConstantZero);
}
```

Adding Some Structure to Our First Example

```
int MyExceptionFilter(EXCEPTION_POINTERS* Pointers)
{
    if (Pointers->ExceptionRecord->ExceptionCode           != STATUS_ACCESS_VIOLATION ||
        Pointers->ExceptionRecord->ExceptionInformation[0] != EXCEPTION_WRITE_FAULT)
    {
        return EXCEPTION_CONTINUE_SEARCH; // Not a write access violation
    }

    puts("A write access violation occurred!  Let's see if we can fix it!");

    void* WriteAddress = (void*)Pointers->ExceptionRecord->ExceptionInformation[1];

    if (!VirtualProtect(WriteAddress, sizeof(int), PAGE_READWRITE))
    {
        return EXCEPTION_CONTINUE_SEARCH;
    }

    return EXCEPTION_CONTINUE_EXECUTION;
}
```

Adding Some Structure to Our First Example

```
int MyExceptionFilter(EXCEPTION_POINTERS* Pointers)
{
    if (Pointers->ExceptionRecord->ExceptionCode           != STATUS_ACCESS_VIOLATION ||
        Pointers->ExceptionRecord->ExceptionInformation[0] != EXCEPTION_WRITE_FAULT)
    {
        return EXCEPTION_CONTINUE_SEARCH; // Not a write access violation
    }

    puts("A write access violation occurred!  Let's see if we can fix it!");

    void* WriteAddress = (void*)Pointers->ExceptionRecord->ExceptionInformation[1];

    if (!VirtualProtect(WriteAddress, sizeof(int), PAGE_READWRITE))
    {
        return EXCEPTION_CONTINUE_SEARCH;
    }

    return EXCEPTION_CONTINUE_EXECUTION;
}
```

Adding Some Structure to Our First Example

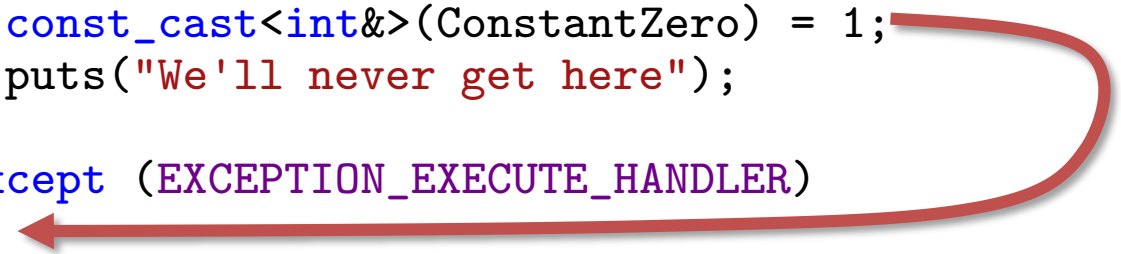
```
void MyWonderfulProgram()
{
    __try
    {
        // Something "dangerous"
    }
    __except (MyExceptionFilter())
    {
        puts("Hello from inside of the __except statement!");
    }
}
```

Let's Take a Closer Look at That __except

```
// Filter Expression Value                  // Handler Return
// -----------------------                  // --------------

#define EXCEPTION_CONTINUE_EXECUTION (-1) // ExceptionContinueExecution

#define EXCEPTION_CONTINUE_SEARCH      0  // ExceptionContinueSearch

#define EXCEPTION_EXECUTE_HANDLER      1  // ???
```

Exception Filter Expression Values

```
void MyWonderfulProgram()
{
    __try
    {
        const_cast<int&>(ConstantZero) = 1;
        puts("We'll never get here");
    }
    __except (EXCEPTION_EXECUTE_HANDLER)
    {
        puts("Oh no, an exception occurred! :'(");
    }
}
```

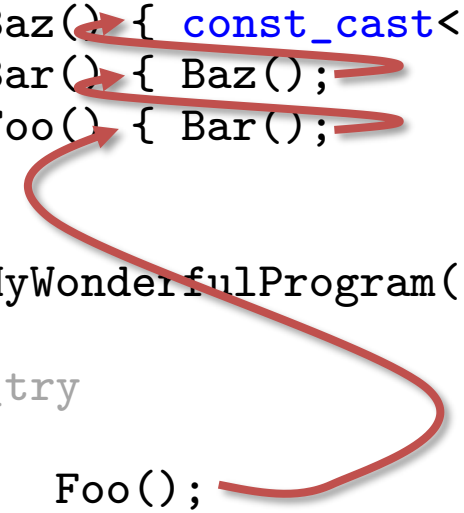


EXCEPTION_EXECUTE_HANDLER

```
void MyWonderfulProgram()
{
    __try
    {
        const_cast<int&>(ConstantZero) = 1;
        puts("We'll never get here");
    }
    __except (EXCEPTION_EXECUTE_HANDLER)
    {
        puts("Oh no, an exception occurred! :'(");
    }
}
```

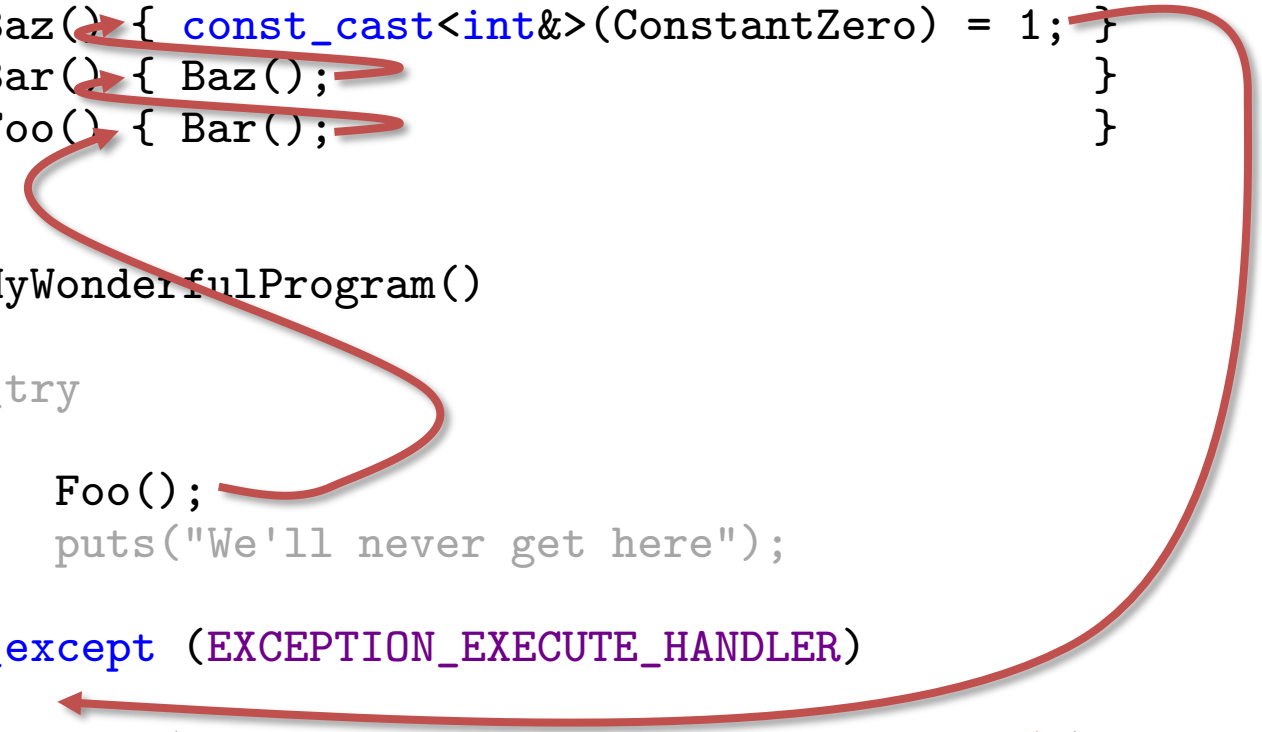
EXCEPTION_EXECUTE_HANDLER

```
void Baz() { const_cast<int&>(ConstantZero) = 1; }
void Bar() { Baz();                              }
void Foo() { Bar();                              }


void MyWonderfulProgram()
{
    __try
    {
        Foo();
        puts("We'll never get here");
    }
    __except (EXCEPTION_EXECUTE_HANDLER)
    {
        puts("Oh no, an exception occurred! :'(");
    }
}
```



EXCEPTION_EXECUTE_HANDLER

```
void Baz() { const_cast<int&>(ConstantZero) = 1; }
void Bar() { Baz();                              }
void Foo() { Bar();                              }


void MyWonderfulProgram()
{
    __try
    {
        Foo();
        puts("We'll never get here");
    }
    __except (EXCEPTION_EXECUTE_HANDLER)
    {
        puts("Oh no, an exception occurred! :'(");
    }
}
```



EXCEPTION_EXECUTE_HANDLER

```
void MyWonderfulProgram()
{
    __try
    {
        ModifyConstantZeroUnderLock();
    }
    __except (EXCEPTION_EXECUTE_HANDLER)
    {
    }
}
```

```
void ModifyConstantZeroUnderLock()
{
    EnterCriticalSection(Lock);

    ModifyConstantZero();

    LeaveCriticalSection(Lock);
}
```

```
void ModifyConstantZero()
{
    const_cast<int&>(ConstantZero) = 1;
}
```

EXCEPTION_EXECUTE_HANDLER

```
void MyWonderfulProgram()
{
    __try
    {
        ModifyConstantZeroUnderLock();
    }
    __except (EXCEPTION_EXECUTE_HANDLER)
    {
    }
}
```

```
void ModifyConstantZeroUnderLock()
{
    EnterCriticalSection(Lock);
    __try
    {
        ModifyConstantZeroUnderLock();
    }
    __finally
    {
        LeaveCriticalSection(Lock);
    }
}

void ModifyConstantZero()
{
    const_cast<int&>(ConstantZero) = 1;
}
```

__try/__finally

Under The Hood…

```
void F()
{
    __try
    {
        // Something dangerous
    }
    __except (FFilterA())
    {
    }

    __try
    {
        // Something else dangerous
    }
    __except (FFilterB())
    {
    }
}
```

Multiple Sequential __try Blocks

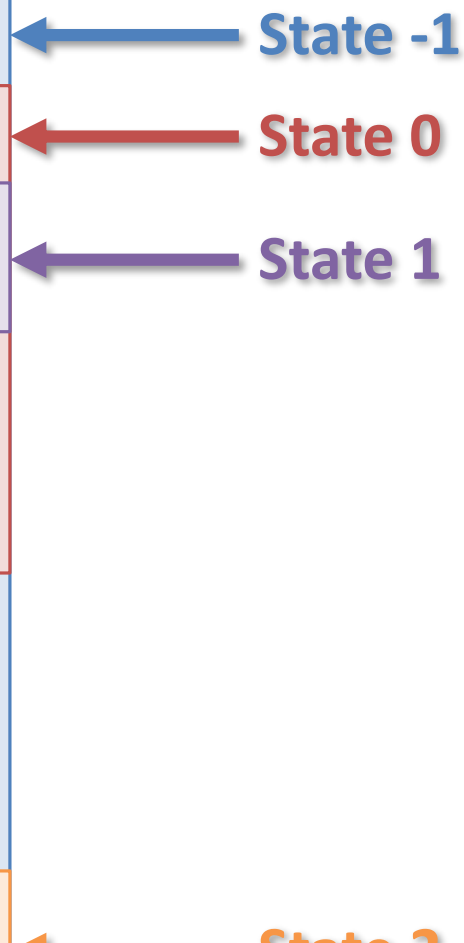
```
void F()
{
    __try
    {
        __try
        {
            // Something dangerous
        }
        __finally
        {

        }
    }
    __except (FFilter1())
    {
    }
}
```

Nested __try Blocks

```
void F()
{
    __try
    {
        __try
        {
            G1();
        }
        __finally
        {
            G2();
        }
    }
    __except (FFilter1())
    {
        G3();
    }

    __try
    {
        G4();
    }
    __except (FFilter2())
    {
        G5();
    }
}
```

A Mixture __try Blocks

```
void F()
{
    __try
    {
        __try
        {
            G1();
        }
        __finally
        {
            G2();
        }
    }
    __except (FFilter1())
    {
        G3();
    }

    __try
    {
        G4();
    }
    __except (FFilter2())
    {
        G5();
    }
}
```



Building a Scope Table

```
void F()
{
    __try
    {
        __try
        {
            G1();
        }
        __finally
        {
            G2();
        }
    }
    __except (FFilter1())
    {
        G3();
    }

    __try
    {
        G4();
    }
    __except (FFilter2())
    {
        G5();
    }
}
```

State -1

State 0

State 1

State 2

```
struct SCOPETABLE_ENTRY
{
    int32_t             EnclosingLevel;
    FILTER_CALLBACK*    Filter;
    HANDLER_CALLBACK*   Handler;
};

SCOPETABLE_ENTRY FScopeTable[3];

[0].EnclosingLevel = -1;
[0].Filter         = &FFilter1;
[0].Handler        = &ExceptBlock1;

[1].EnclosingLevel = 0;
[1].Filter         = nullptr;
[1].Handler        = &FinallyBlock;

[2].EnclosingLevel = -1;
[2].Filter         = &FFilter2;
[2].Handler        = &ExceptBlock2;
```

Building a Scope Table

```
struct EXCEPTION_REGISTRATION_RECORD
{
    EXCEPTION_REGISTRATION_RECORD* Next;
    EXCEPTION_ROUTINE*             Handler;
};

struct C_EXCEPTION_REGISTRATION_RECORD
{
    void*                          StackPointer;
    EXCEPTION_POINTERS*            Exception;
    EXCEPTION_REGISTRATION_RECORD  HandlerRegistration;
    SCOPETABLE_ENTRY*              ScopeTable;
    int                            TryLevel;
};
```

C_EXCEPTION_REGISTRATION_RECORD

```
EXCEPTION_DISPOSITION MyExceptionHandler(
    EXCEPTION_RECORD*               ExceptionRecord,
    void*                           EstablisherFrame,
    CONTEXT*                        ContextRecord,
    void*                           DispatcherContext
    );
```

C_EXCEPTION_REGISTRATION_RECORD

```
EXCEPTION_DISPOSITION MyExceptionHandler(
    EXCEPTION_RECORD*                ExceptionRecord,
    EXCEPTION_REGISTRATION_RECORD*   EstablisherFrame,
    CONTEXT*                         ContextRecord,
    void*                            DispatcherContext
    );
```

C_EXCEPTION_REGISTRATION_RECORD

```
C_EXCEPTION_REGISTRATION_RECORD RN;

RN.StackPointer                 = // TBD
RN.Exception                    = // TBD

RN.HandlerRegistration.Handler = &_except_handler3;
RN.HandlerRegistration.Next    = TIB->ExceptionList;

RN.ScopeTable                   = &FScopeTable;
RN.TryLevel                     = -1;

TIB->ExceptionList = &RN.HandlerRegistration;
```

On Entry into the Function

```
TIB->ExceptionList = TIB->ExceptionList->Next;
```

On Return from the Function

```
void F()
{
    __try
    {
        __try
        {
            G1();
        }
        __finally
        {
            G2();
        }
    }
    __except (FFilter1())
    {
        G3();
    }

    __try
    {
        G4();
    }
    __except (FFilter2())
    {
        G5();
    }
}
```

State -1

State 0

State 1

State 2

```
...
    __try
mov         dword ptr [RN.TryLevel],0
    {
        __try
mov         dword ptr [RN.TryLevel],1
        {
            G1();
call        G1
        }
mov         dword ptr [RN.TryLevel],0
...
```

In The Function: Updating the TryLevel

```
EXCEPTION_DISPOSITION _except_handler3(
    EXCEPTION_RECORD*                ExceptionRecord,
    EXCEPTION_REGISTRATION_RECORD* EstablisherFrame,
    CONTEXT*                         ContextRecord,
    void*                            DispatcherContext
    )
{
    C_EXCEPTION_REGISTRATION_RECORD* RN = RNFromEstablisherFrame(EstablisherFrame);

    RN->ExceptionPointers = &EXCEPTION_POINTERS{ ExceptionRecord, ContextRecord };

    for (int I = RN->TryLevel; I != -1; I = RN->ScopeTable[I].EnclosingLevel)
    {
        if (RN->ScopeTable[I].Filter == nullptr) { continue; }

        int FilterResult = RN->ScopeTable[I].Filter();









    }


}
```

_except_handler3

```
EXCEPTION_DISPOSITION _except_handler3(
    EXCEPTION_RECORD*                ExceptionRecord,
    EXCEPTION_REGISTRATION_RECORD* EstablisherFrame,
    CONTEXT*                         ContextRecord,
    void*                            DispatcherContext
    )
{
    C_EXCEPTION_REGISTRATION_RECORD* RN = RNFromEstablisherFrame(EstablisherFrame);

    RN->ExceptionPointers = &EXCEPTION_POINTERS{ ExceptionRecord, ContextRecord };

    for (int I = RN->TryLevel; I != -1; I = RN->ScopeTable[I].EnclosingLevel)
    {
        if (RN->ScopeTable[I].Filter == nullptr) { continue; }

        int FilterResult = RN->ScopeTable[I].Filter();
        switch (FilterResult)
        {
        case EXCEPTION_CONTINUE_SEARCH:    continue;
        case EXCEPTION_CONTINUE_EXECUTION: return ExceptionContinueExecution;
        case EXCEPTION_EXECUTE_HANDLER:    // TBD
        }
    }

    return ExceptionContinueSearch;
}
```

_except_handler3

Unwinding is split into two parts:

- Global unwinding -- unwinding across frames
- Local unwinding -- unwinding a single frame

Unwinding

```
void RtlUnwind(
    EXCEPTION_REGISTRATION_RECORD* TargetFrame,
    void*                          TargetIp,
    EXCEPTION_RECORD*              ExceptionRecord,
    void*                          ReturnValue
    )
{
    ExceptionRecord->ExceptionFlags |= EXCEPTION_UNWINDING;

    NT_TIB* TIB = (NT_TIB*)NtCurrentTeb();

    while (TIB->ExceptionList != TargetFrame)
    {
        TIB->ExceptionList->Handler(ExceptionRecord, TIB->ExceptionList);

        TIB->ExceptionList = CurrentRecord->Next;
    }
}
```

Global Unwinding in RtlUnwind

```
void _local_unwind(
    C_EXCEPTION_REGISTRATION_RECORD* RN,
    int                              Stop
    )
{
    while (RN->TryLevel != Stop)
    {
        SCOPETABLE_ENTRY* CurrentEntry = &RN->ScopeTable[RN->TryLevel];
        if (CurrentEntry->Filter == nullptr)
        {
            CurrentEntry->Handler();
        }

        RN->TryLevel = CurrentEntry->EnclosingLevel;
    }
}
```

Local Unwinding in _local_unwind

```
EXCEPTION_DISPOSITION _except_handler3(
    EXCEPTION_RECORD*                ExceptionRecord,
    EXCEPTION_REGISTRATION_RECORD* EstablisherFrame,
    CONTEXT*                         ContextRecord,
    void*                            DispatcherContext
    )
{
    C_EXCEPTION_REGISTRATION_RECORD* RN = RNFromEstablisherFrame(EstablisherFrame);

    RN->ExceptionPointers = &EXCEPTION_POINTERS{ ExceptionRecord, ContextRecord };



    for (int I = RN->TryLevel; I != -1; I = RN->ScopeTable[I].EnclosingLevel)
    {
        if (RN->ScopeTable[I].Filter == nullptr) { continue; }

        int FilterResult = RN->ScopeTable[I].Filter();
        // ...
    }



}
```

_except_handler3 Redux

```
EXCEPTION_DISPOSITION _except_handler3(
    EXCEPTION_RECORD*                ExceptionRecord,
    EXCEPTION_REGISTRATION_RECORD* EstablisherFrame,
    CONTEXT*                         ContextRecord,
    void*                            DispatcherContext
    )
{
    C_EXCEPTION_REGISTRATION_RECORD* RN = RNFromEstablisherFrame(EstablisherFrame);

    RN->ExceptionPointers = &EXCEPTION_POINTERS{ ExceptionRecord, ContextRecord };



        for (int I = RN->TryLevel; I != -1; I = RN->ScopeTable[I].EnclosingLevel)
        {
            if (RN->ScopeTable[I].Filter == nullptr) { continue; }

            int FilterResult = RN->ScopeTable[I].Filter();
            // ...
        }



}
```

_except_handler3 Redux

```
EXCEPTION_DISPOSITION _except_handler3(
    EXCEPTION_RECORD*                ExceptionRecord,
    EXCEPTION_REGISTRATION_RECORD* EstablisherFrame,
    CONTEXT*                         ContextRecord,
    void*                            DispatcherContext
    )
{
    C_EXCEPTION_REGISTRATION_RECORD* RN = RNFromEstablisherFrame(EstablisherFrame);

    RN->ExceptionPointers = &EXCEPTION_POINTERS{ ExceptionRecord, ContextRecord };

    if ((ExceptionRecord->ExceptionFlags & EXCEPTION_UNWINDING) == 0)
    {
        for (int I = RN->TryLevel; I != -1; I = RN->ScopeTable[I].EnclosingLevel)
        {
            if (RN->ScopeTable[I].Filter == nullptr) { continue; }

            int FilterResult = RN->ScopeTable[I].Filter();
            // ...
        }
    }


}
```

_except_handler3 Redux

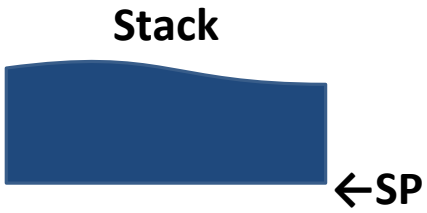
```
EXCEPTION_DISPOSITION _except_handler3(
    EXCEPTION_RECORD*                ExceptionRecord,
    EXCEPTION_REGISTRATION_RECORD* EstablisherFrame,
    CONTEXT*                         ContextRecord,
    void*                            DispatcherContext
    )
{
    C_EXCEPTION_REGISTRATION_RECORD* RN = RNFromEstablisherFrame(EstablisherFrame);

    RN->ExceptionPointers = &EXCEPTION_POINTERS{ ExceptionRecord, ContextRecord };

    if ((ExceptionRecord->ExceptionFlags & EXCEPTION_UNWINDING) == 0)
    {
        for (int I = RN->TryLevel; I != -1; I = RN->ScopeTable[I].EnclosingLevel)
        {
            if (RN->ScopeTable[I].Filter == nullptr) { continue; }

            int FilterResult = RN->ScopeTable[I].Filter();
            // ...
        }
    }
    else
    {
        _local_unwind(RN, -1);
    }
}
```

_except_handler3 Redux

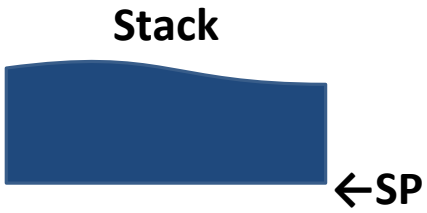
```
for (int I = RN->TryLevel; I != -1; I = RN->ScopeTable[i].EnclosingLevel)
{
    if (RN->ScopeTable[I].Filter == nullptr) { continue; }

    int FilterResult = RN->ScopeTable[I].Filter();
    switch (FilterResult)
    {
    case EXCEPTION_CONTINUE_SEARCH:    continue;
    case EXCEPTION_CONTINUE_EXECUTION: return ExceptionContinueExecution;
    case EXCEPTION_EXECUTE_HANDLER:

        RtlUnwind(EstablisherFrame, ExceptionRecord);

        _local_unwind(RN, RN->TryLevel);

        RN->ScopeTable[I].Handler();

        assert(false);
    }
}
```

EXCEPTION_EXECUTE_HANDLER and Unwinding

Let’s Walk Through That End-to-End…

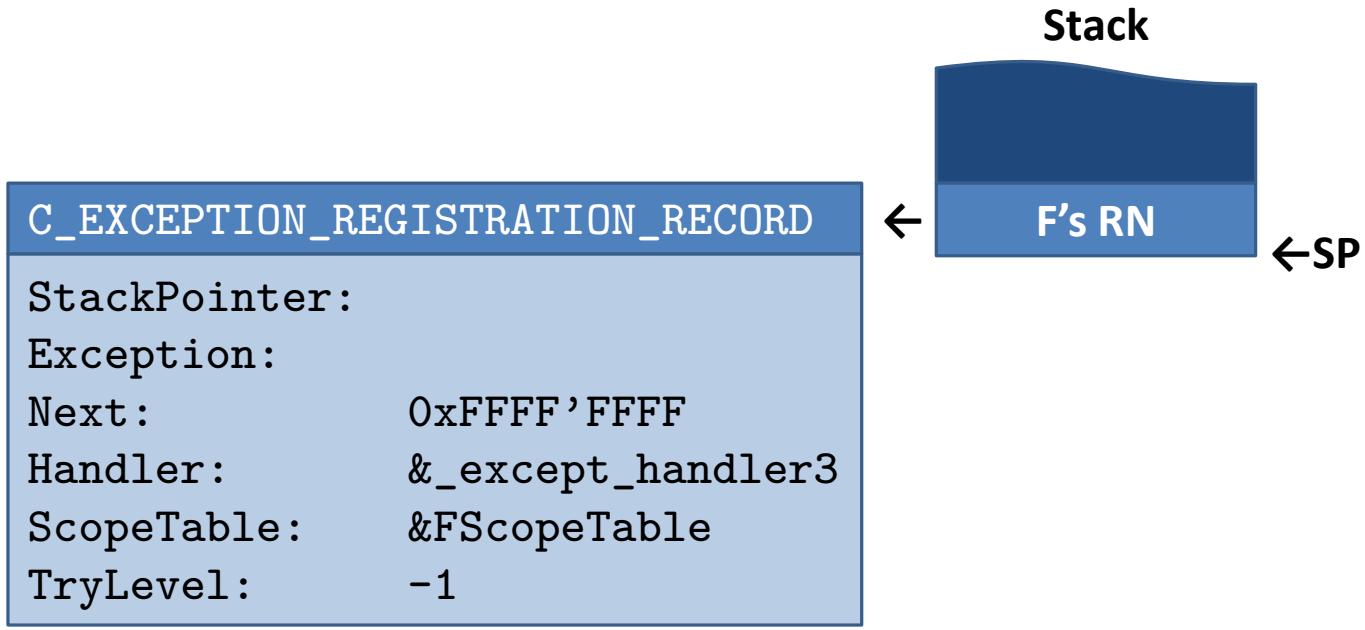

Stack
←SP

```
    void F()
IP→ {
        __try
        {
            __try
            {
                G();
            }
            __except(FFilter())
            {
            }
        }
        __finally
        {
        }
    }
```



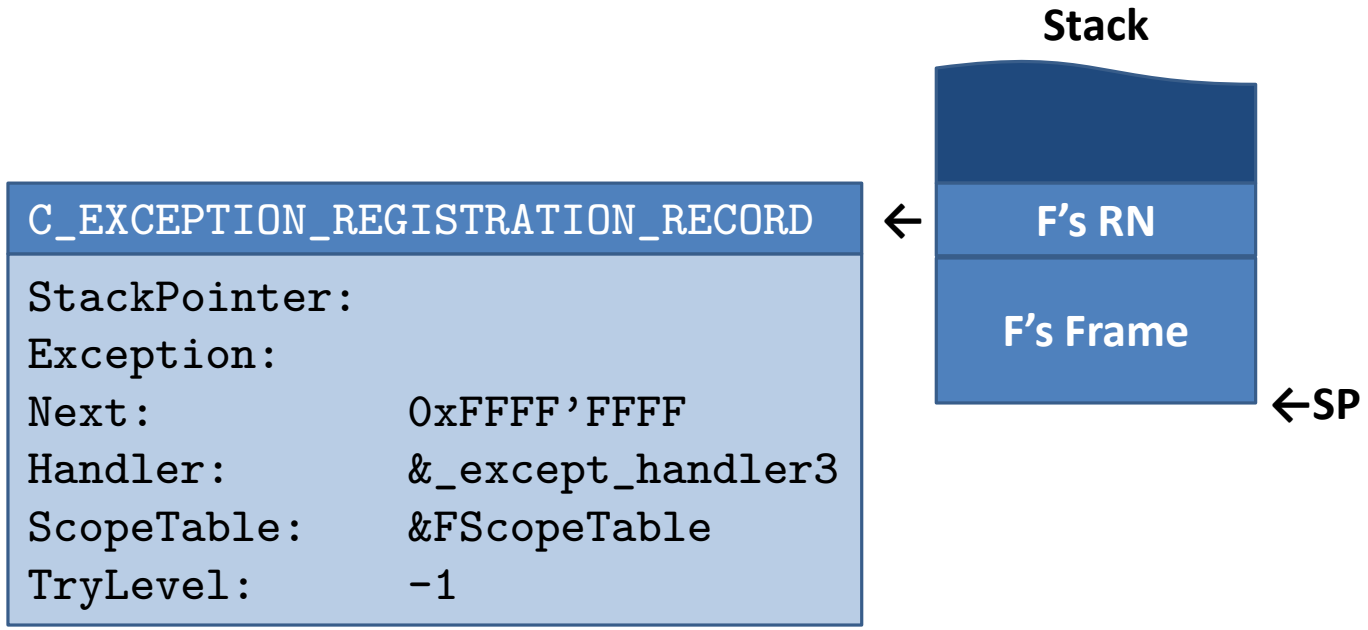
Someone Calls F

```
    void F()
IP→ {
        __try
        {
            __try
            {
                G();
            }
            __except(FFilter())
            {
            }
        }
        __finally
        {
        }
    }
```



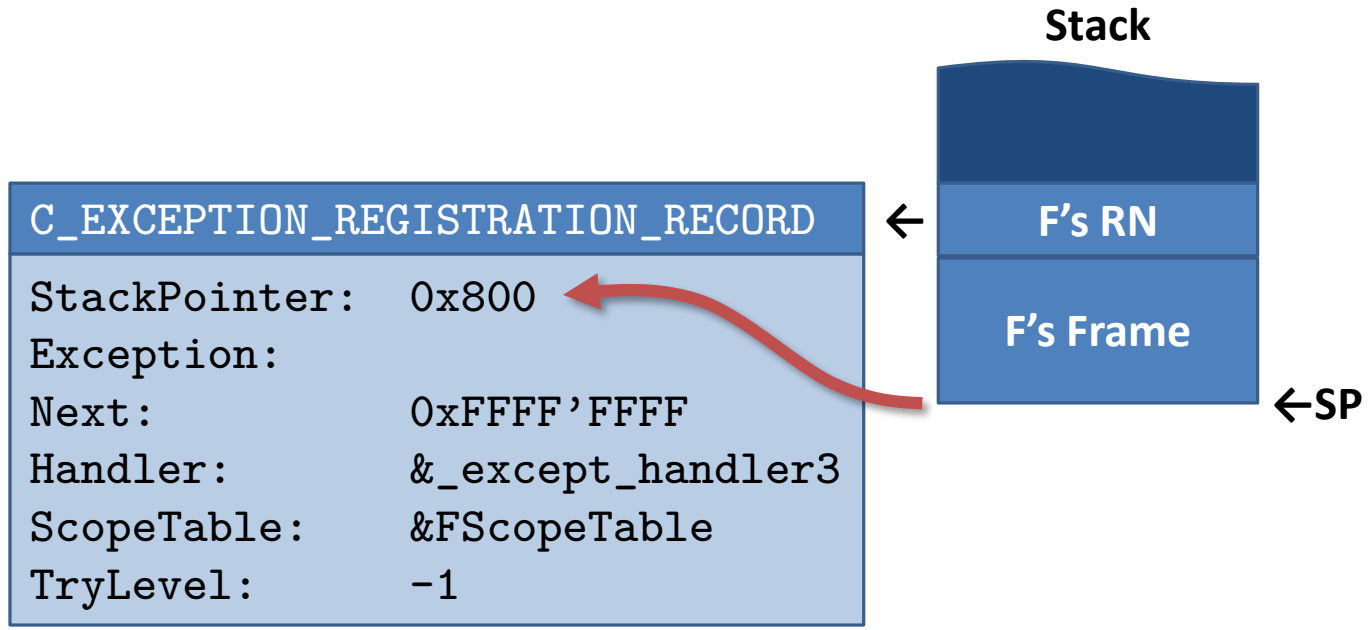
Build a Registration Node for F

```
    void F()
IP→ {
        __try
        {
            __try
            {
                G();
            }
            __except(FFilter())
            {
            }
        }
        __finally
        {
        }
    }
```



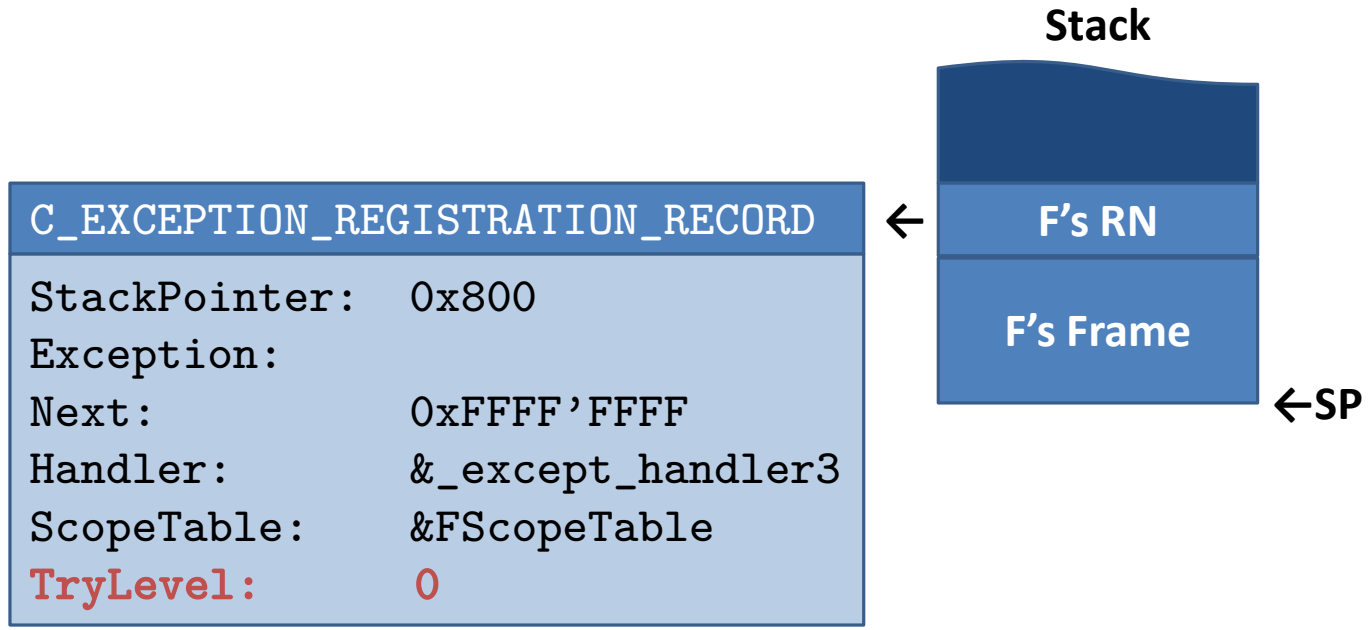
Allocate a Frame for F

```
    void F()
IP→ {
        __try
        {
            __try
            {
                G();
            }
            __except(FFilter())
            {
            }
        }
        __finally
        {
        }
    }
```



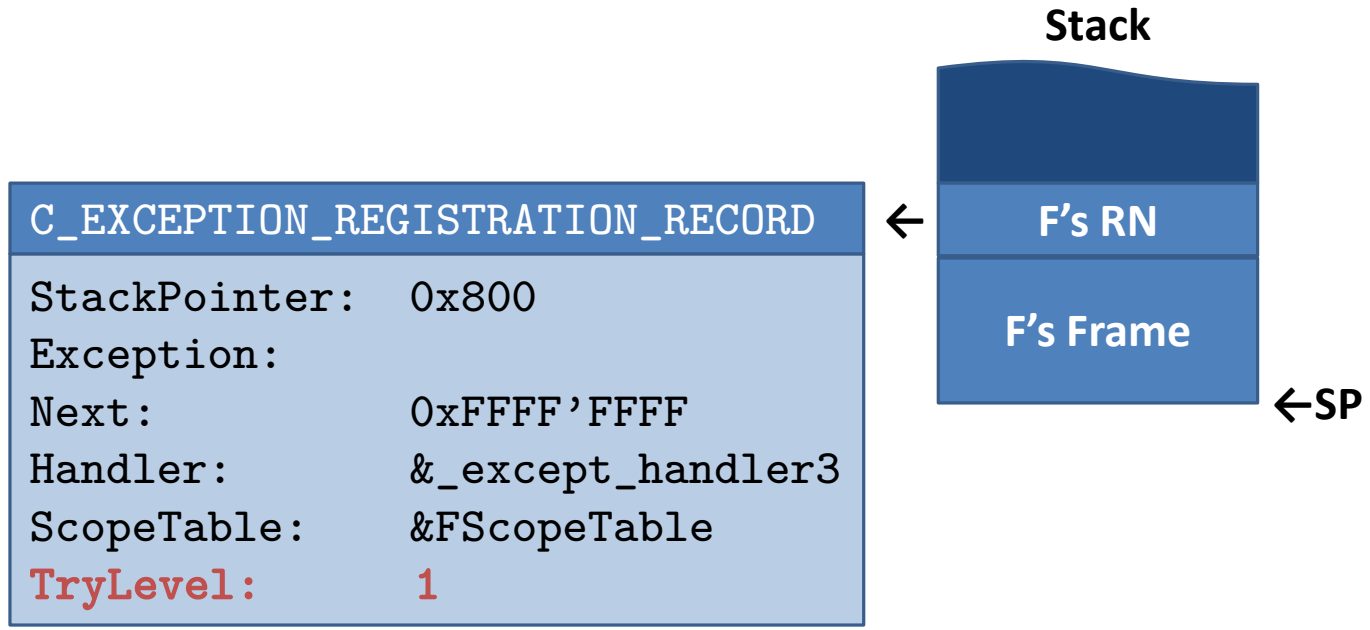
Allocate a Frame for F

```
    void F()
    {
        __try
IP→     {
            __try
            {
                G();
            }
            __except(FFilter())
            {
            }
        }
        __finally
        {
        }
    }
```



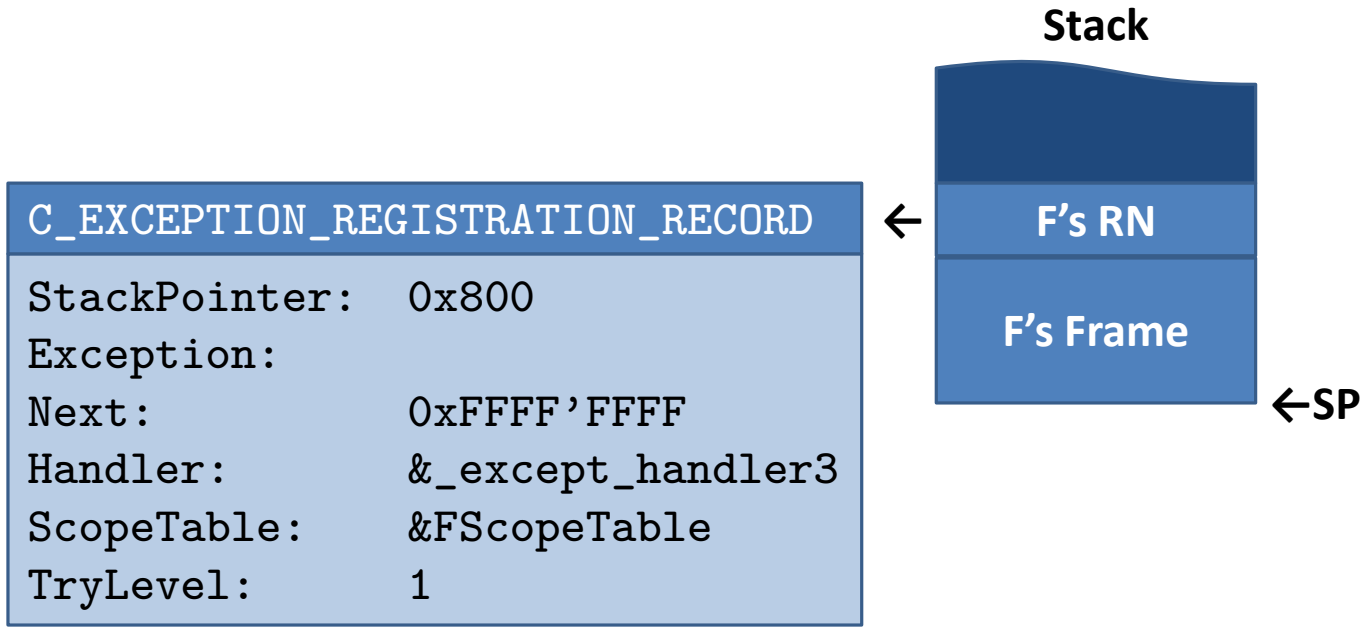
Enter Outer __try/__finally in F (State 0)

```
void F()
{
    __try
    {
        __try
        {
IP→         G();
        }
        __except(FFilter())
        {
        }
    }
    __finally
    {
    }
}
```

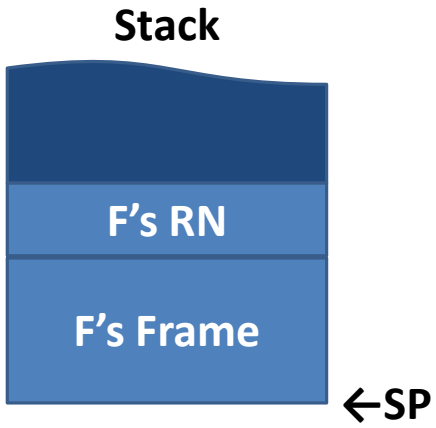


Enter Inner __try/__except in F (State 1)

```
    void F()
    {
        __try
        {
            __try
            {
IP→             G();
            }
            __except(FFilter())
            {
            }
        }
        __finally
        {
        }
    }
```



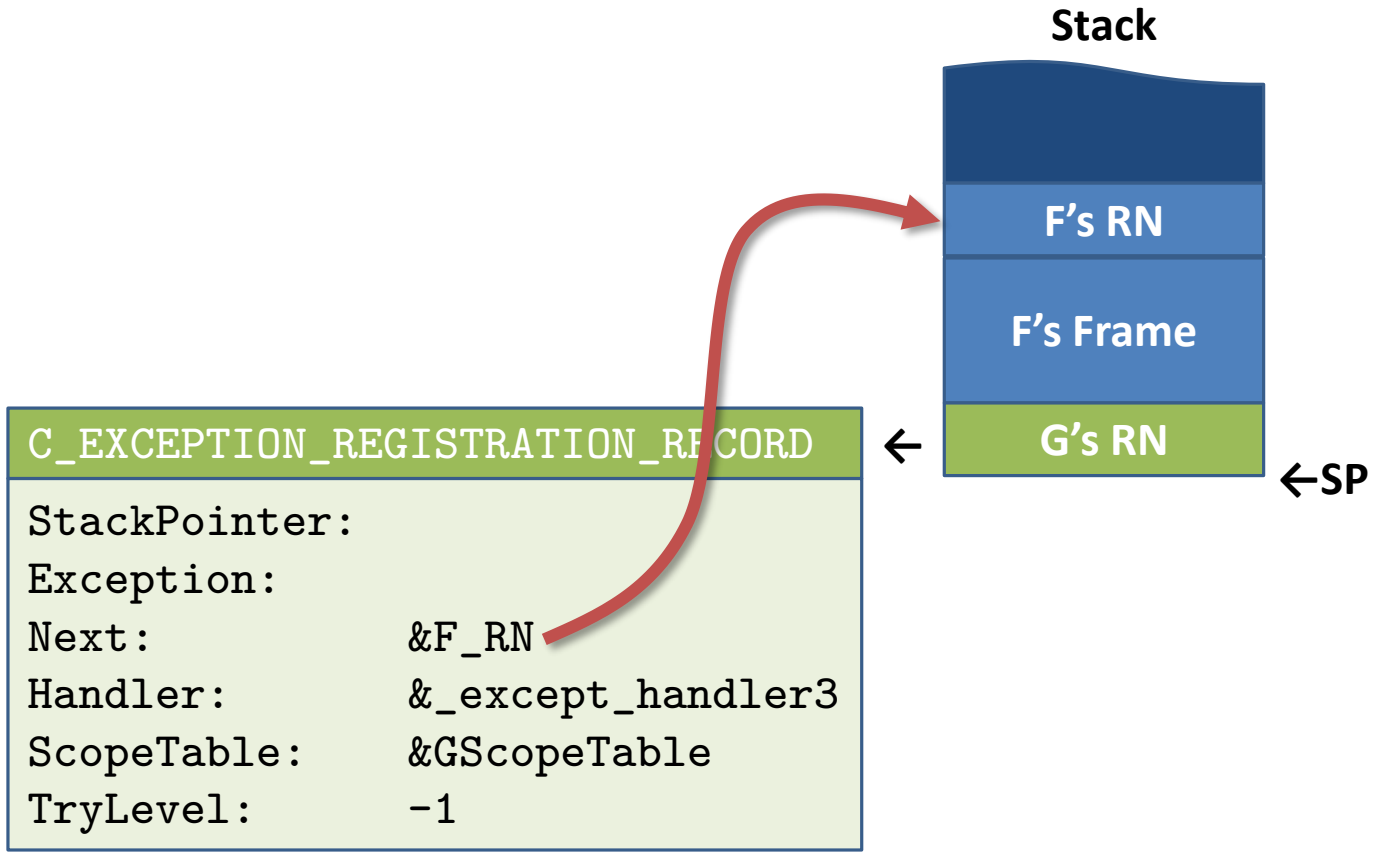
Call G




```
    void G()
IP→ {
        __try
        {
            __try
            {
                H();
            }
            __except(GFilter())
            {
            }
        }
        __finally
        {
        }
    }
```

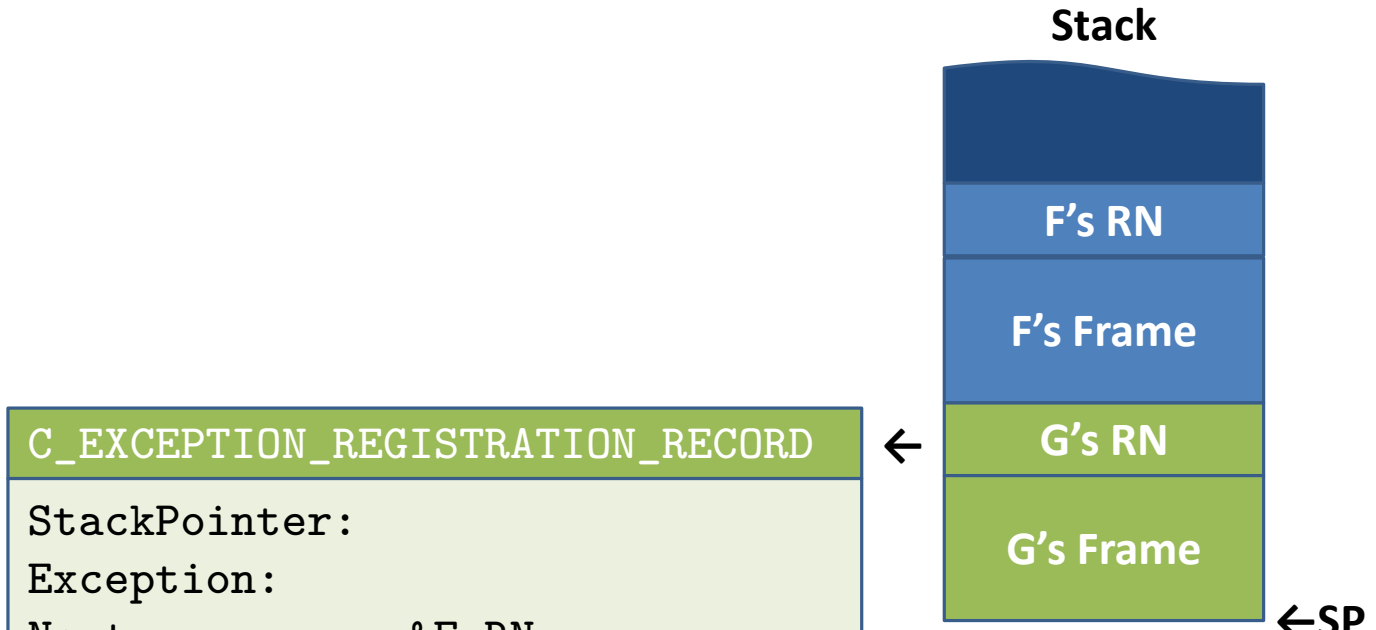
Enter G

```
    void G()
IP→ {
        __try
        {
            __try
            {
                H();
            }
            __except(GFilter())
            {
            }
        }
        __finally
        {
        }
    }
```



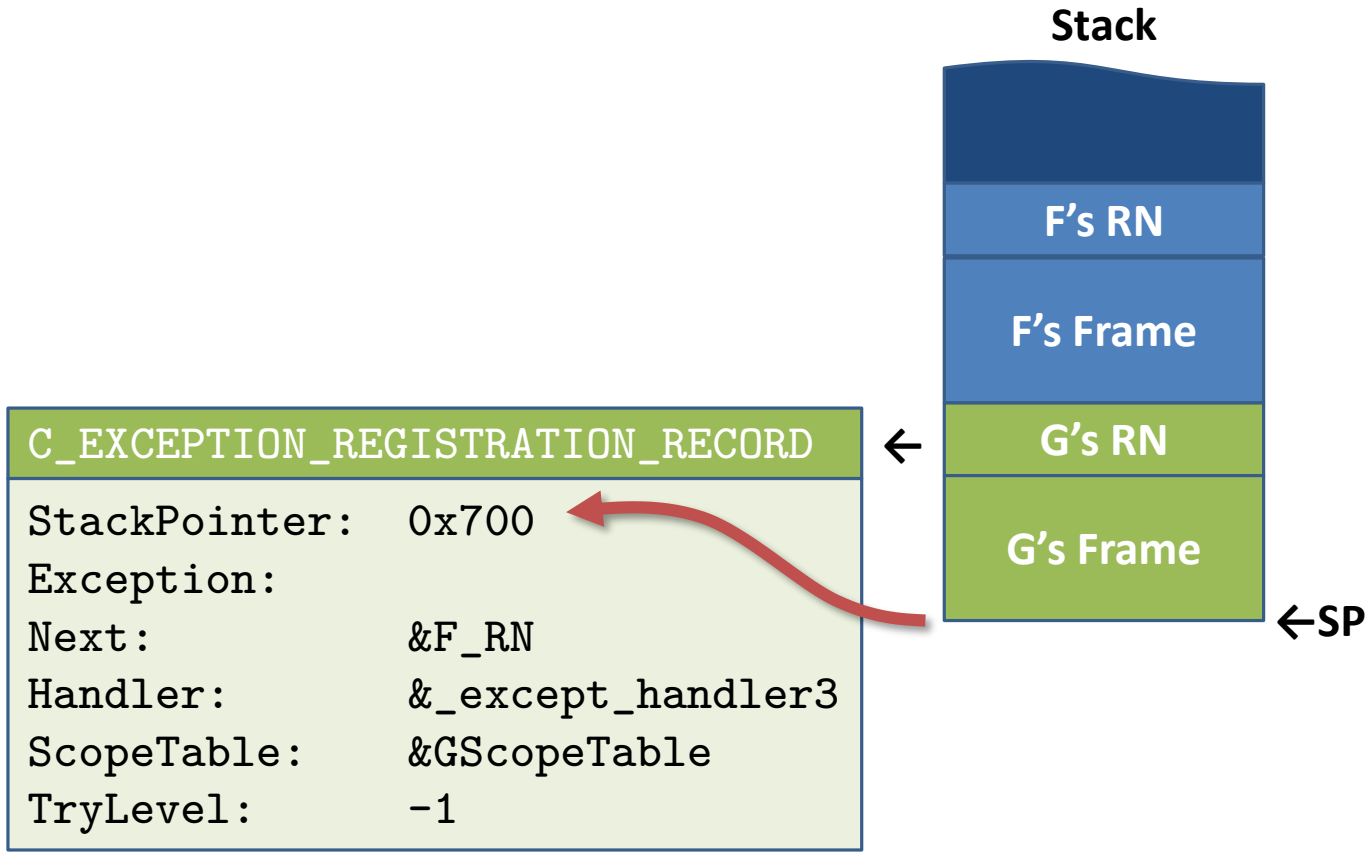
Build a Registration Node for G

```
    void G()
IP→ {
        __try
        {
            __try
            {
                H();
            }
            __except(GFilter())
            {
            }
        }
        __finally
        {
        }
    }
```



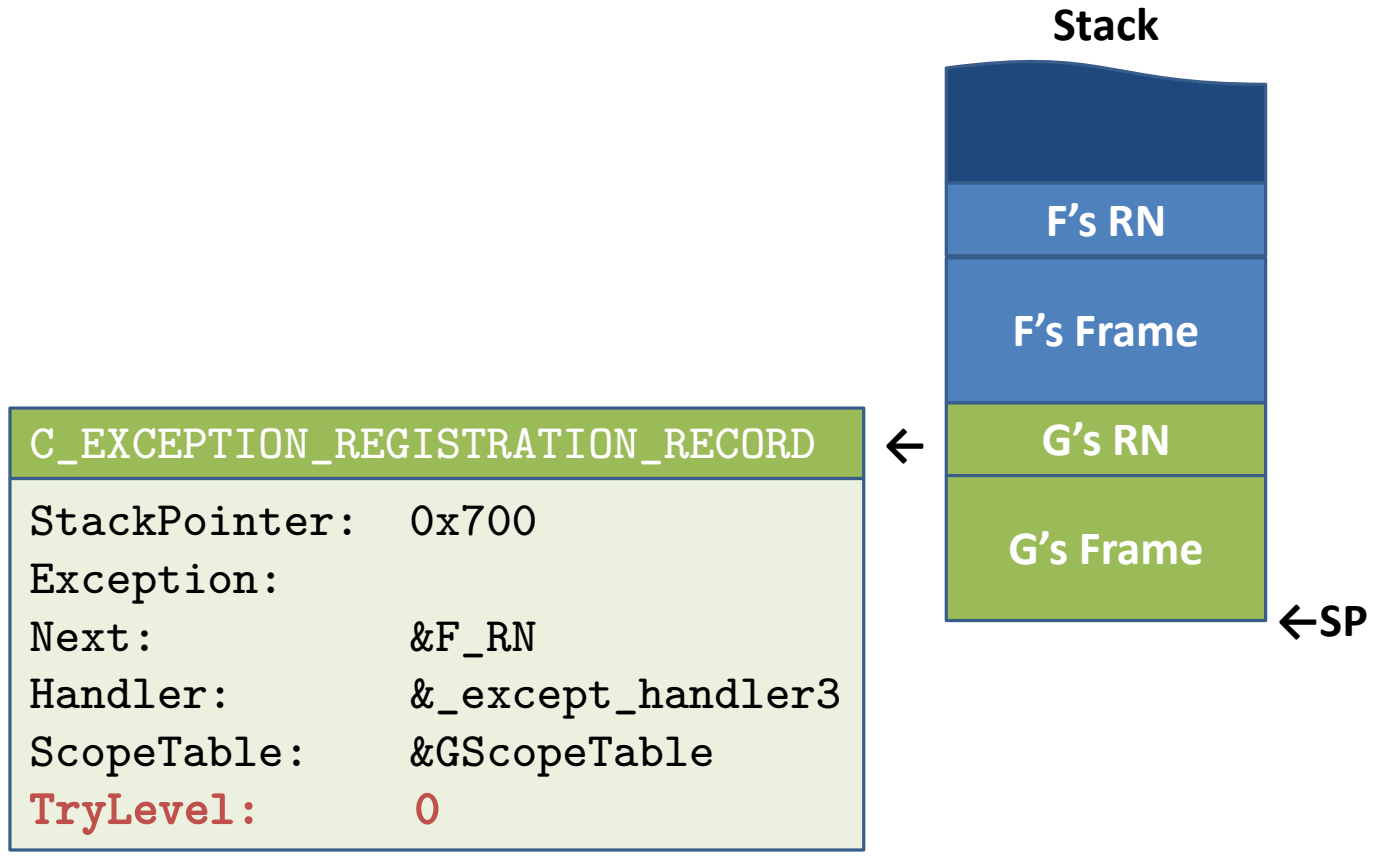
Allocate a Frame for G

```
    void G()
IP→ {
        __try
        {
            __try
            {
                H();
            }
            __except(GFilter())
            {
            }
        }
        __finally
        {
        }
    }
```



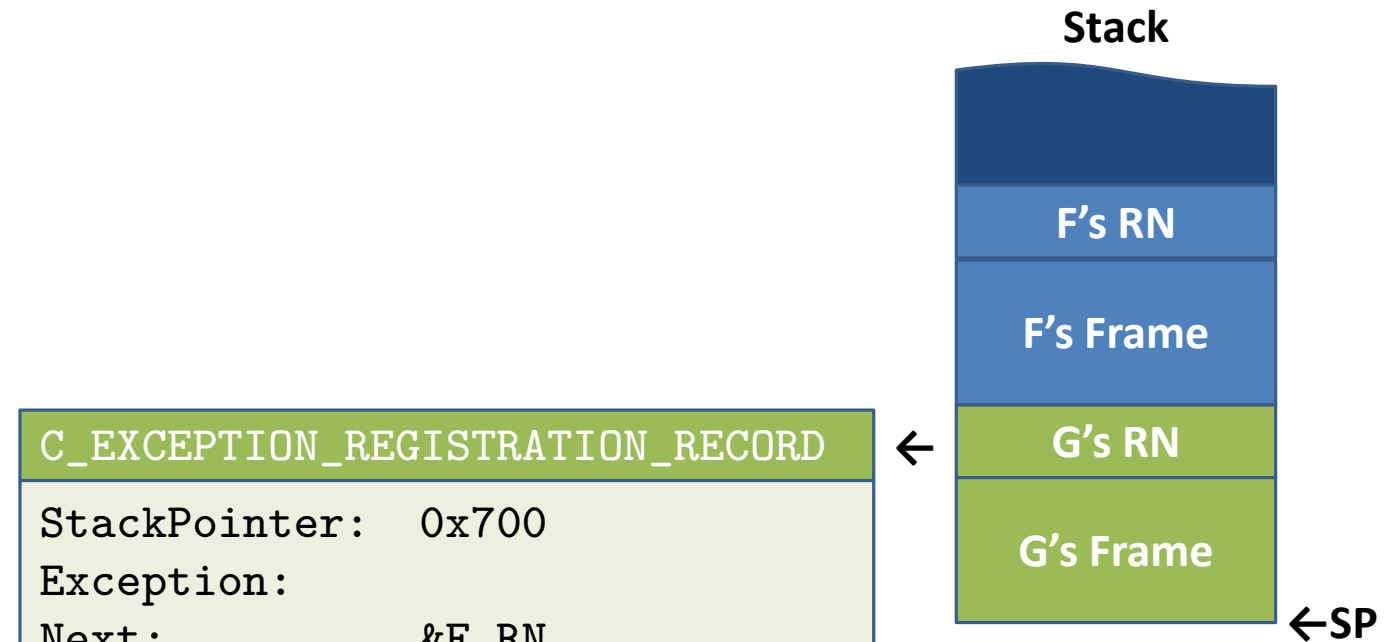
Allocate a Frame for G

```
    void G()
    {
        __try
IP→     {
            __try
            {
                H();
            }
            __except(GFilter())
            {
            }
        }
        __finally
        {
        }
    }
```



Enter Outer __try/__finally in G (State 0)

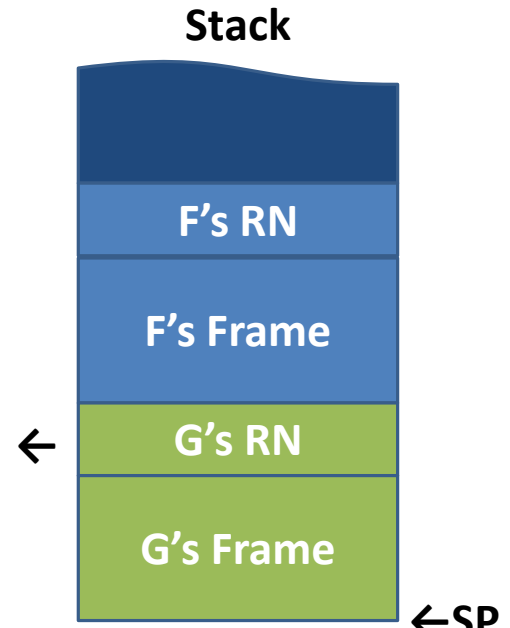
```
    void G()
    {
        __try
        {
            __try
            {
IP→             H();
            }
            __except(GFilter())
            {
            }
        }
        __finally
        {
        }
    }
```



Enter Inner __try/__except in G (State 1)

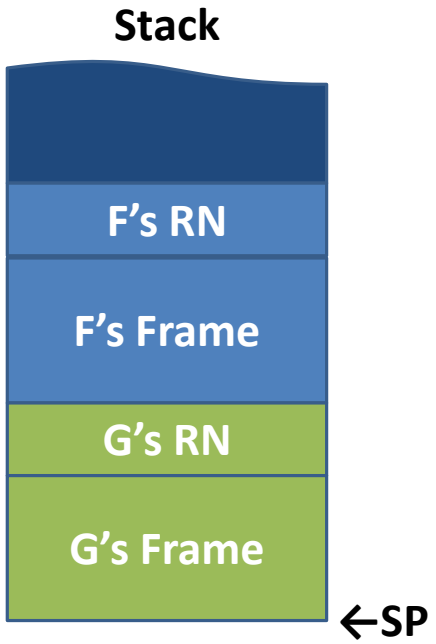
```
    void G()
    {
        __try
        {
            __try
            {
IP→             H();
            }
            __except(GFilter())
            {
            }
        }
        __finally
        {
        }
    }
```

C_EXCEPTION_REGISTRATION_RECORD	
StackPointer:	0x700
Exception:	
Next:	&F_RN
Handler:	&_except_handler3
ScopeTable:	&GScopeTable
TryLevel:	1



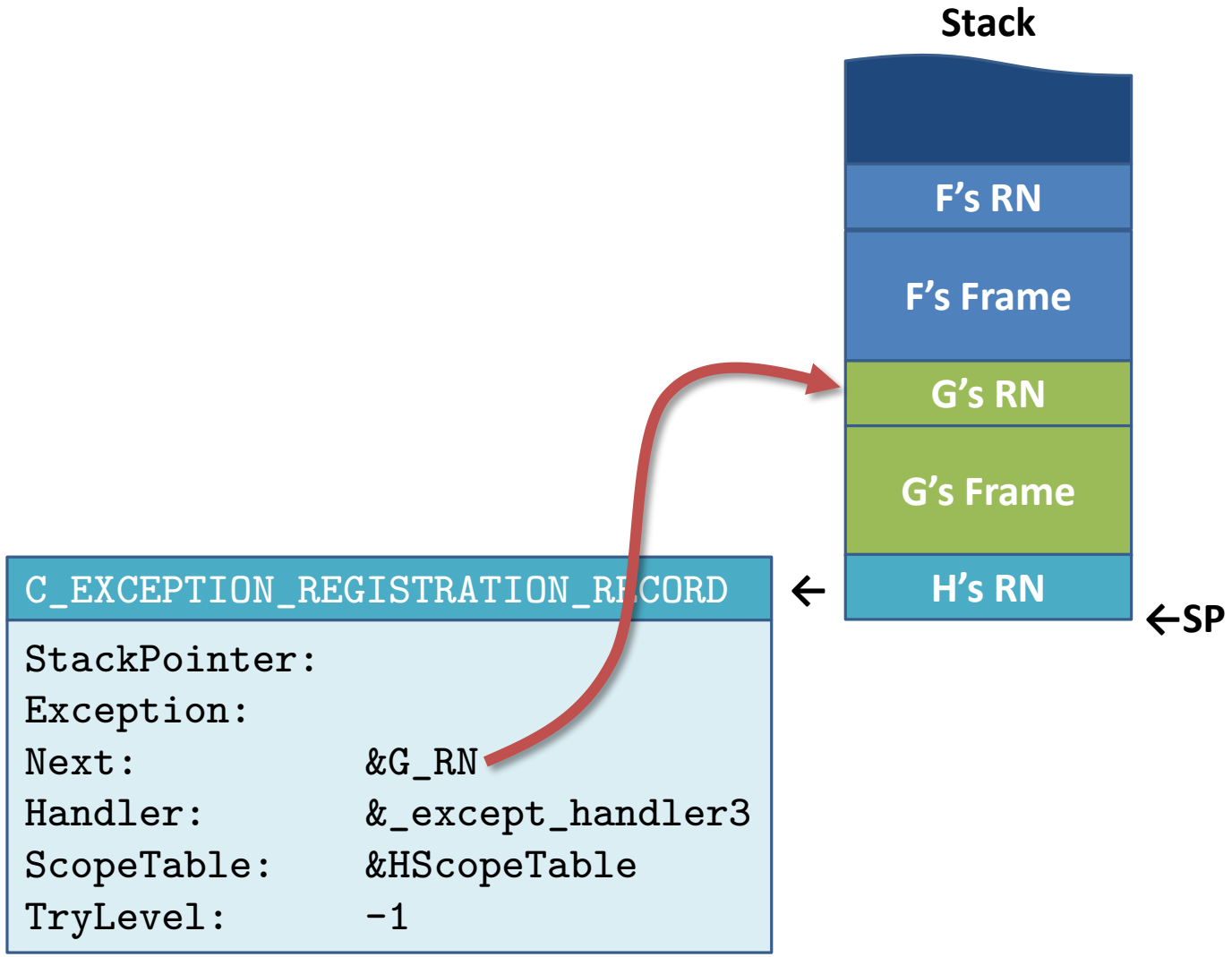
Call H

```
    void H()
IP→ {
        __try
        {
            *(int*)nullptr = 0;
        }
        __finally
        {
        }
    }
```



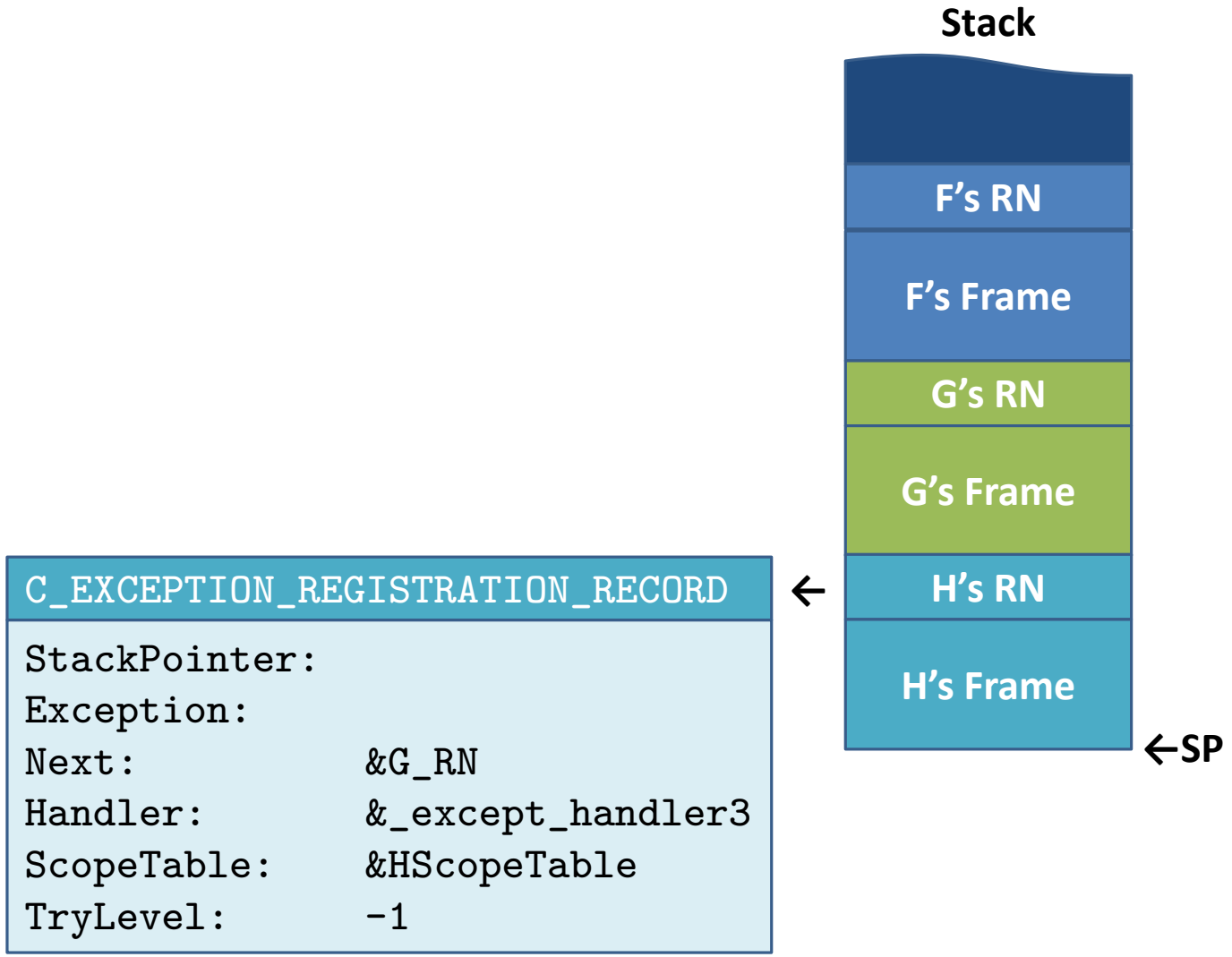
Enter H

```
    void H()
IP→ {
        __try
        {
            *(int*)nullptr = 0;
        }
        __finally
        {
        }
    }
```



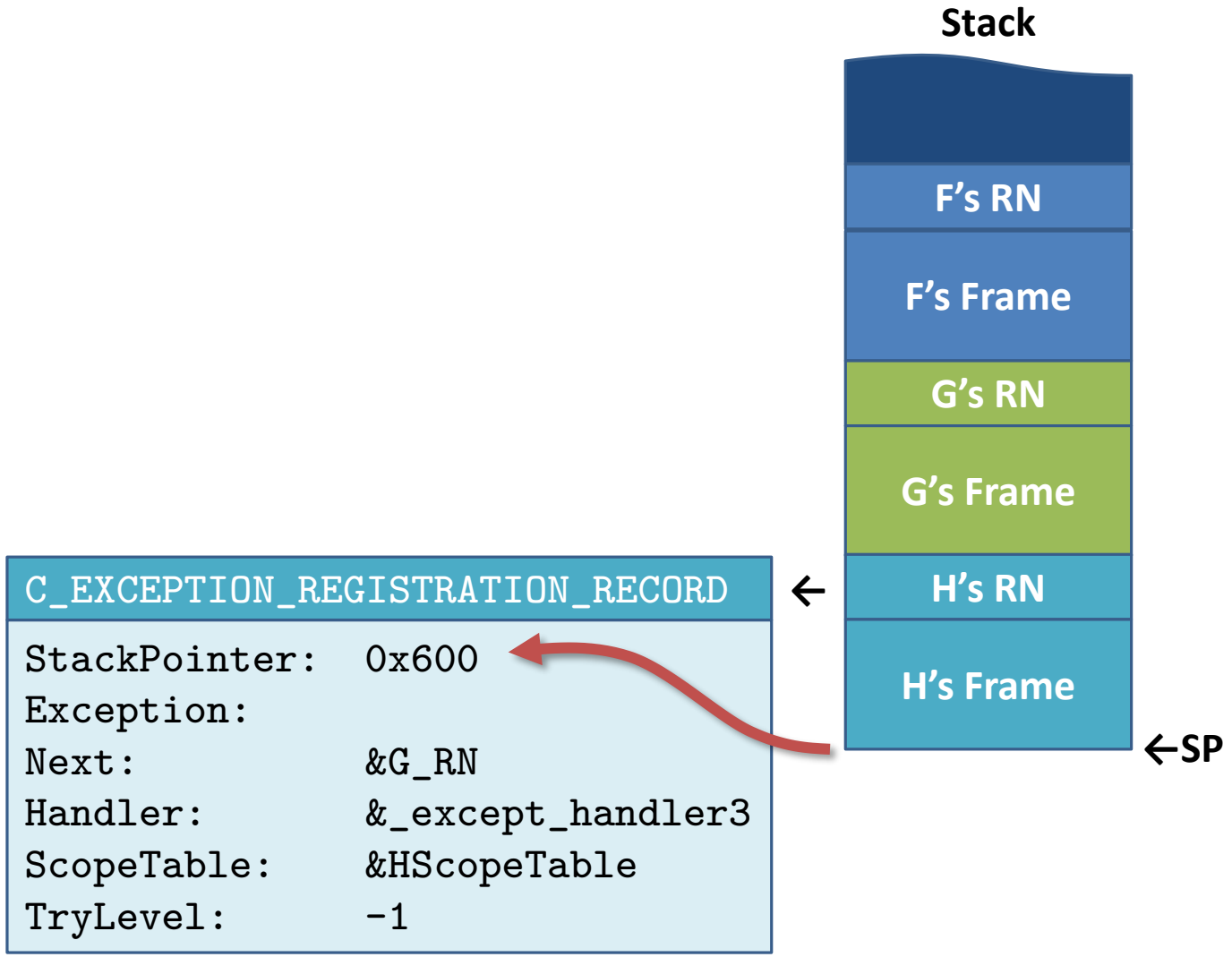
Build a Registration Node for H

```
    void H()
IP→ {
        __try
        {
            *(int*)nullptr = 0;
        }
        __finally
        {
        }
    }
```



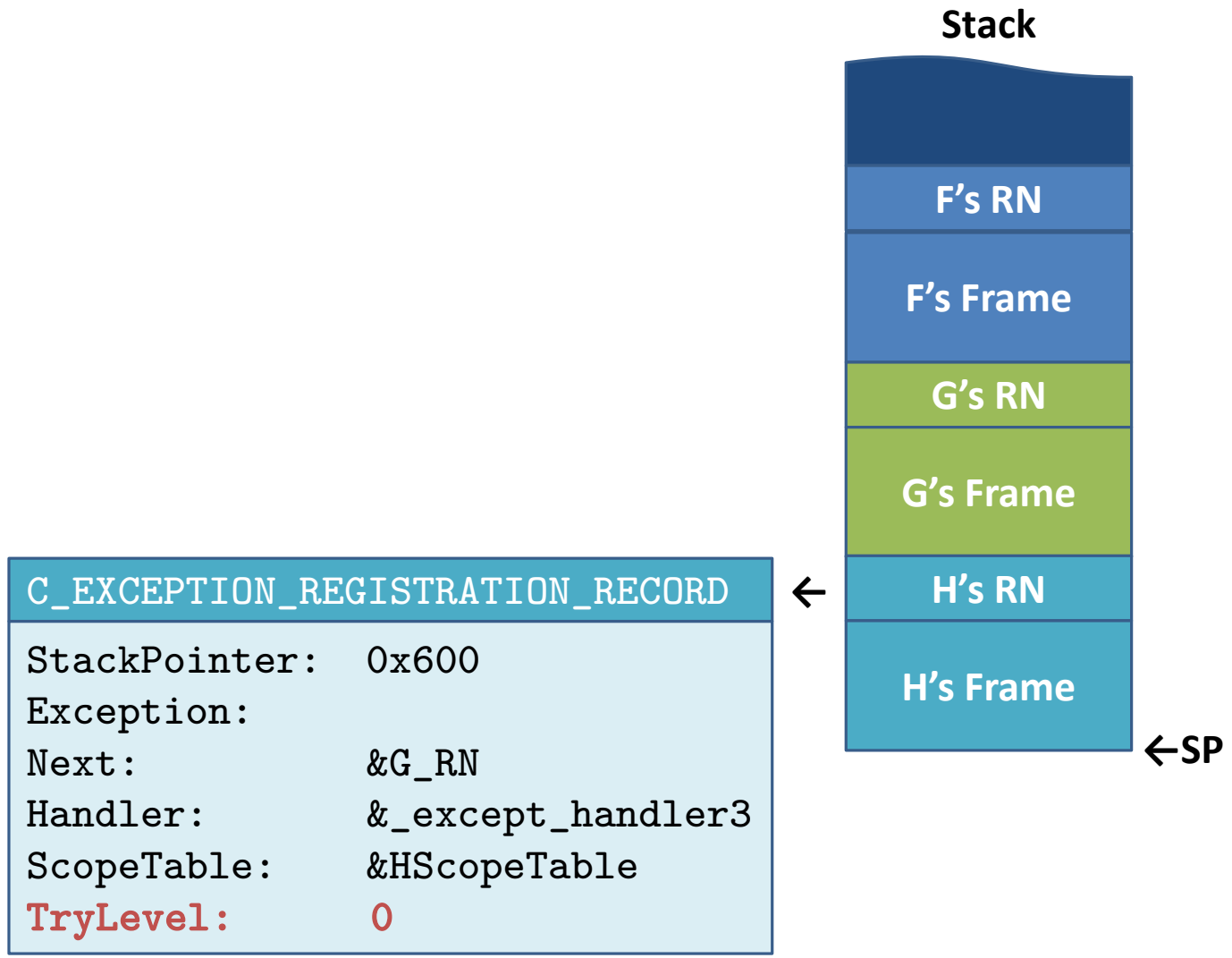
Allocate a Frame for H

```
    void H()
IP→ {
        __try
        {
            *(int*)nullptr = 0;
        }
        __finally
        {
        }
    }
```



Allocate a Frame for H

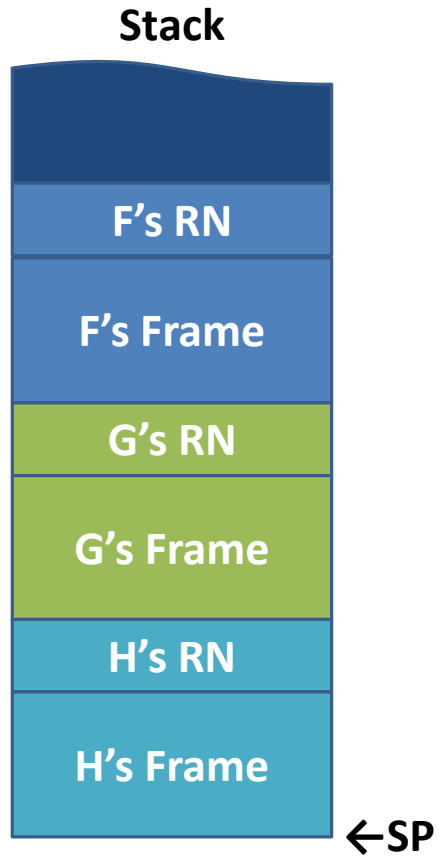
```
    void H()
    {
        __try
IP→     {
            *(int*)nullptr = 0;
        }
        __finally
        {
        }
    }
```



Enter __try/__finally in H (State 0)

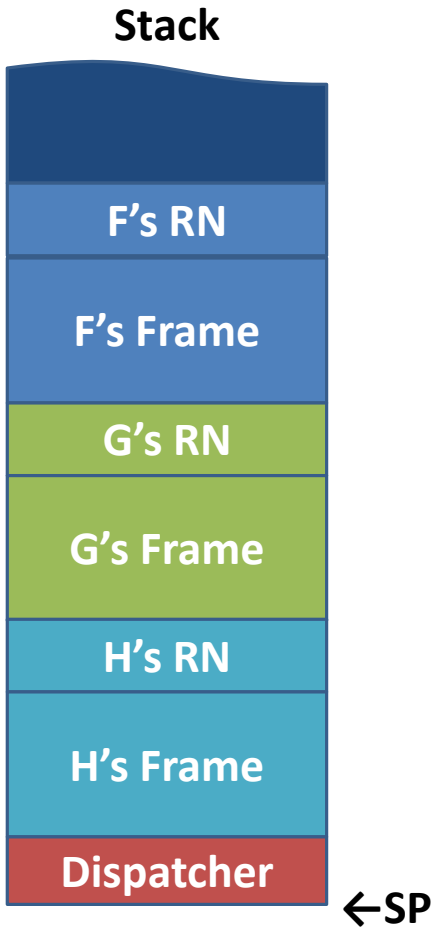
```
void H()
{
    __try
    {
IP→     *(int*)nullptr = 0;
    }
    __finally
    {
    }
}
```

OH NOOOOOOOOOOO

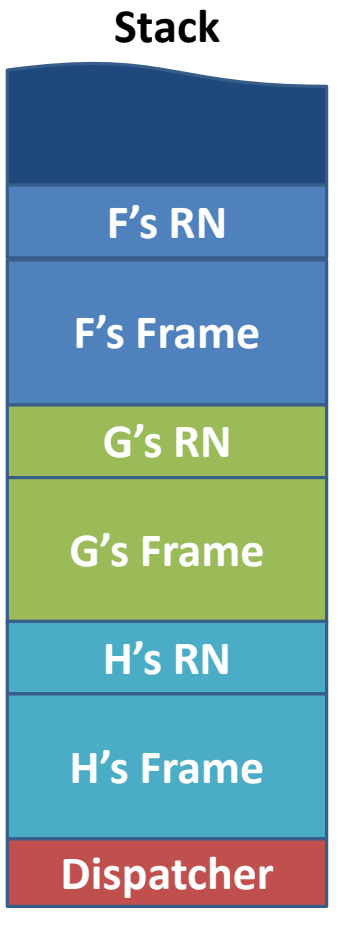


C_EXCEPTION_REGISTRATION_RECORD	
StackPointer:	0x600
Exception:	
Next:	&G_RN
Handler:	&_except_handler3
ScopeTable:	&HScopeTable
TryLevel:	0

Execute This Instruction



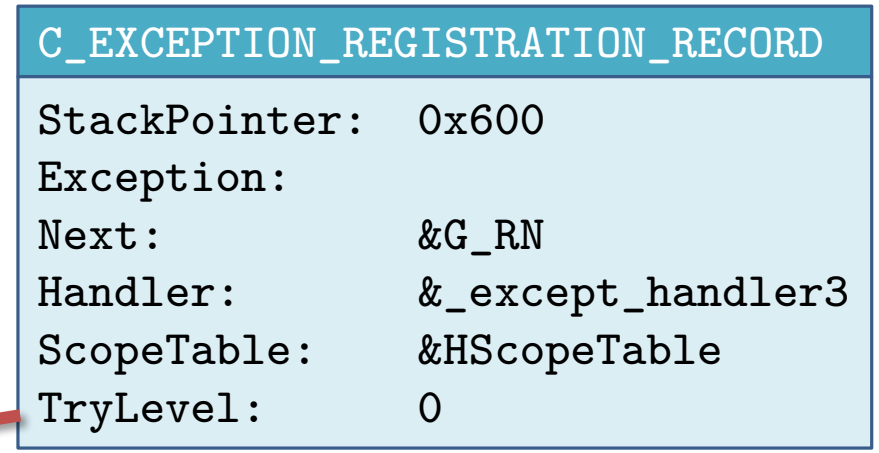
The Exception Gets Dispatched

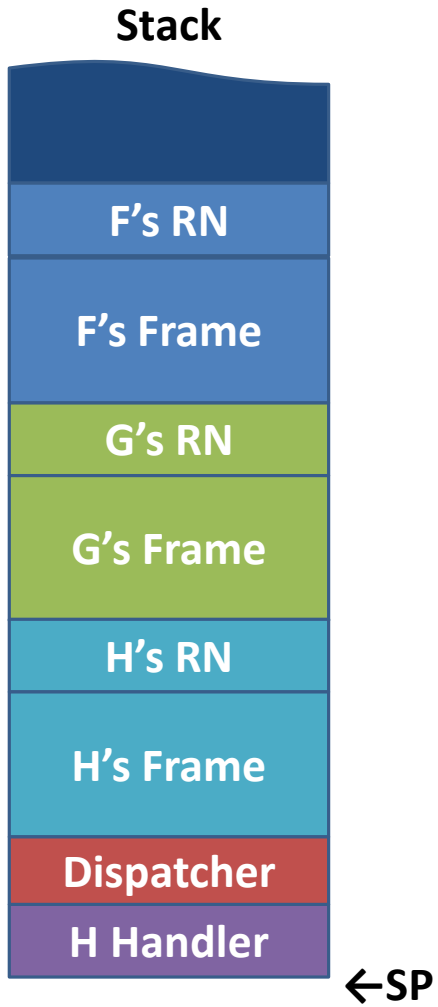


C_EXCEPTION_REGISTRATION_RECORD	
StackPointer:	0x600
Exception:	
Next:	&G_RN
Handler:	&_except_handler3
ScopeTable:	&HScopeTable
TryLevel:	0

The Dispatcher Gets The Most Recent Handler Registration

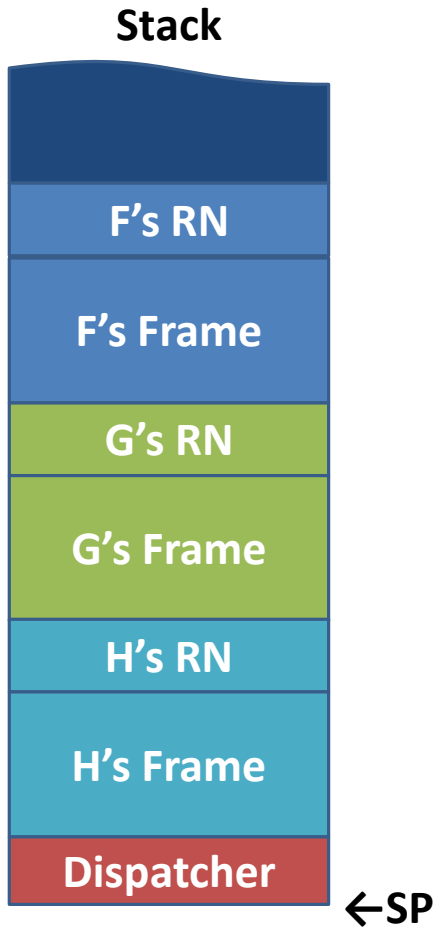
```
void H()
{
    __try
    {
        *(int*)nullptr = 0;
    }
    __finally
    {
    }
}
```





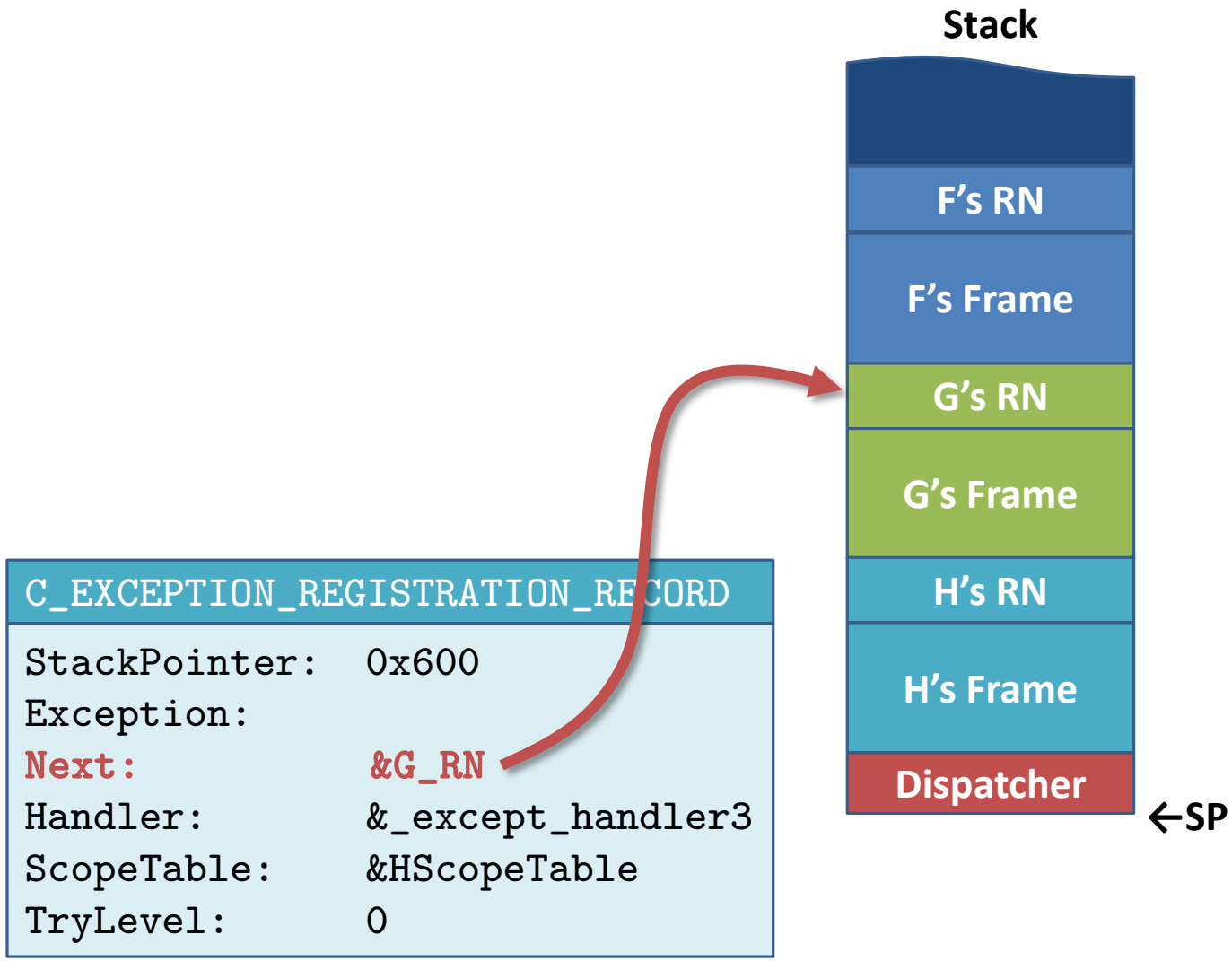
_except_handler3 Called for H

```
void H()
{
    __try
    {
        *(int*)nullptr = 0;
    }
    __finally
    {
    }
}
```

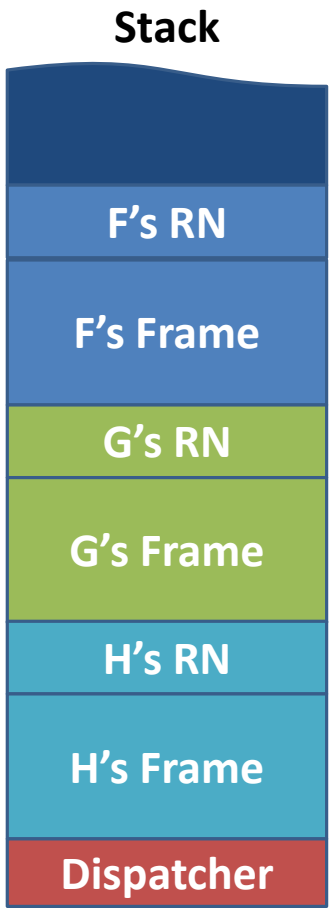


C_EXCEPTION_REGISTRATION_RECORD	
StackPointer:	0x600
Exception:	
Next:	&G_RN
Handler:	&_except_handler3
ScopeTable:	&HScopeTable
TryLevel:	0

_except_handler3 for H Returns ExceptionContinueSearch



The Dispatcher Follows the Link to the Next Handler

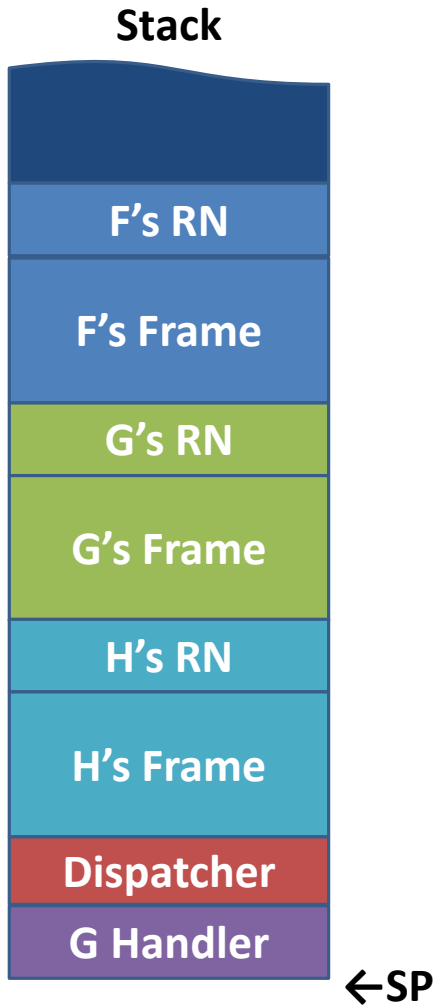


C_EXCEPTION_REGISTRATION_RECORD	
StackPointer:	0x700
Exception:	
Next:	&F_RN
Handler:	&_except_handler3
ScopeTable:	&GScopeTable
TryLevel:	1

The Dispatcher Follows the Link to the Next Handler

```
void G()
{
    __try
    {
        __try
        {
            H();
        }
        __except(GFilter())
        {
        }
    }
    __finally
    {
    }
}
```

C_EXCEPTION_REGISTRATION_RECORD	
StackPointer:	0x700
Exception:	
Next:	&F_RN
Handler:	&_except_handler3
ScopeTable:	&GScopeTable
TryLevel:	1

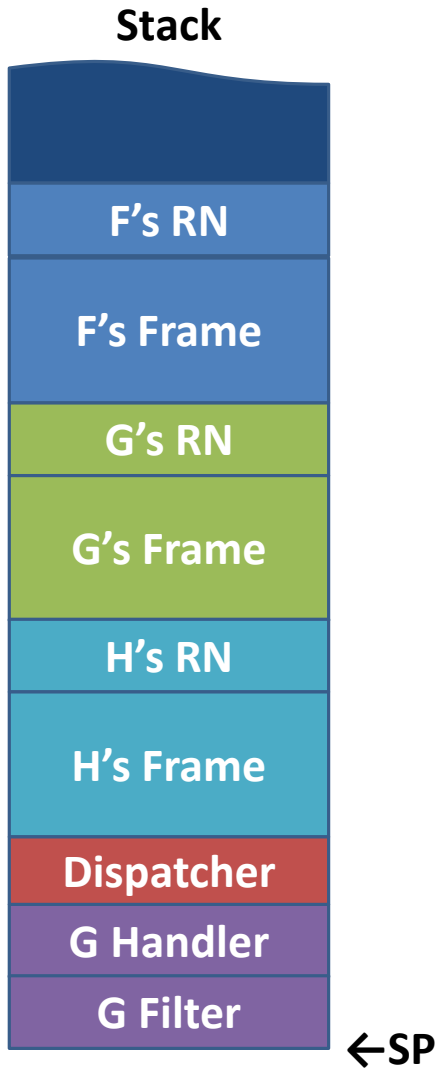


_except_handler3 Called for G

```
int GFilter()
{
    return EXCEPTION_CONTINUE_SEARCH;
}
```

```
void G()
{
    __try
    {
        __try
        {
            H();
        }
        __except(GFilter())
        {
        }
    }
    __finally
    {
    }
}
```

C_EXCEPTION_REGISTRATION_RECORD	
StackPointer:	0x700
Exception:	
Next:	&F_RN
Handler:	&_except_handler3
ScopeTable:	&GScopeTable
TryLevel:	1

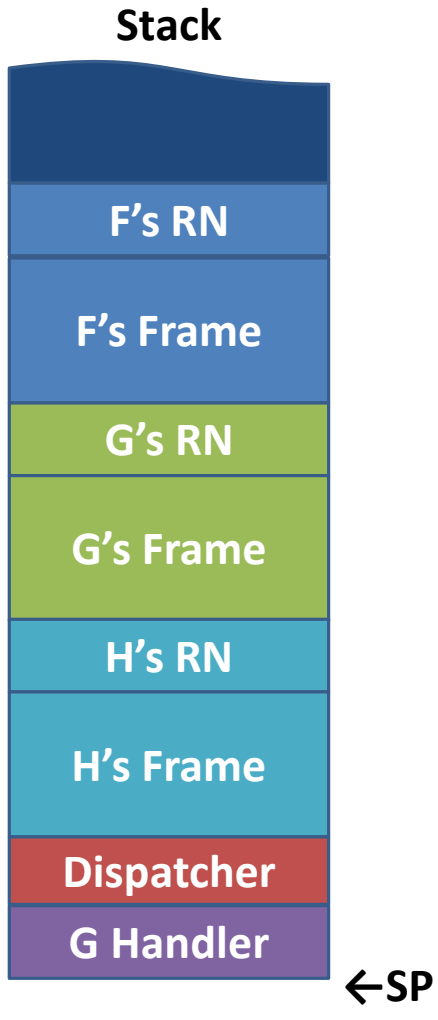


_except_handler3 Calls GFilter

```
int GFilter()
{
    return EXCEPTION_CONTINUE_SEARCH;
}
```

```
void G()
{
    __try
    {
        __try
        {
            H();
        }
        __except(GFilter())
        {
        }
    }
    __finally
    {
    }
}
```

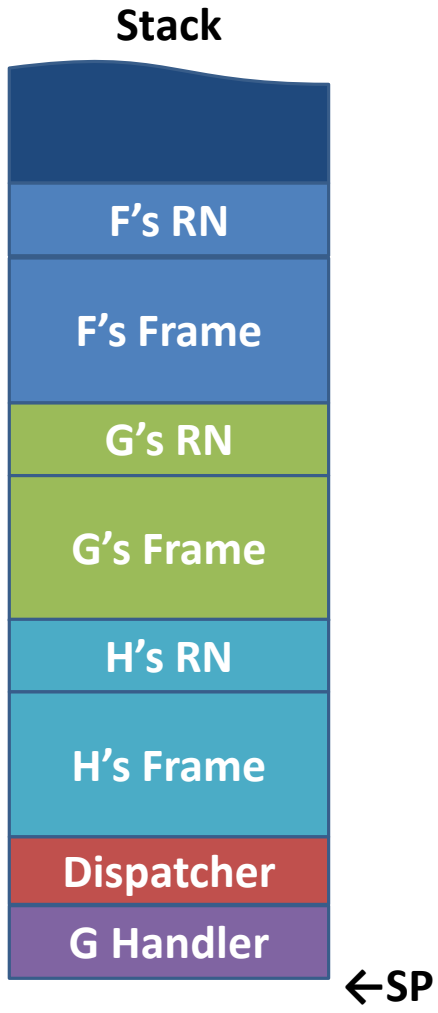
C_EXCEPTION_REGISTRATION_RECORD	
StackPointer:	0x700
Exception:	
Next:	&F_RN
Handler:	&_except_handler3
ScopeTable:	&GScopeTable
TryLevel:	1



GFilter Returns EXCEPTION_CONTINUE_SEARCH

```
void G()
{
    __try
    {
        __try
        {
            H();
        }
        __except(GFilter())
        {
        }
    }
    __finally
    {
    }
}
```

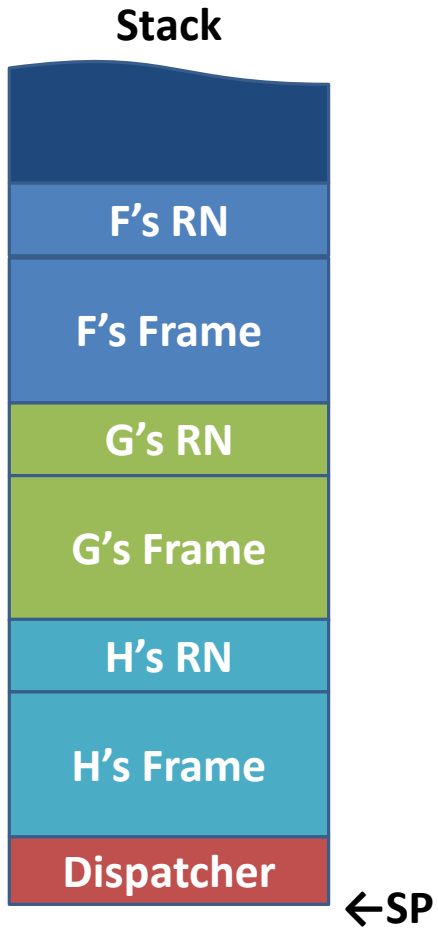
C_EXCEPTION_REGISTRATION_RECORD	
StackPointer:	0x700
Exception:	
Next:	&F_RN
Handler:	&_except_handler3
ScopeTable:	&GScopeTable
TryLevel:	1



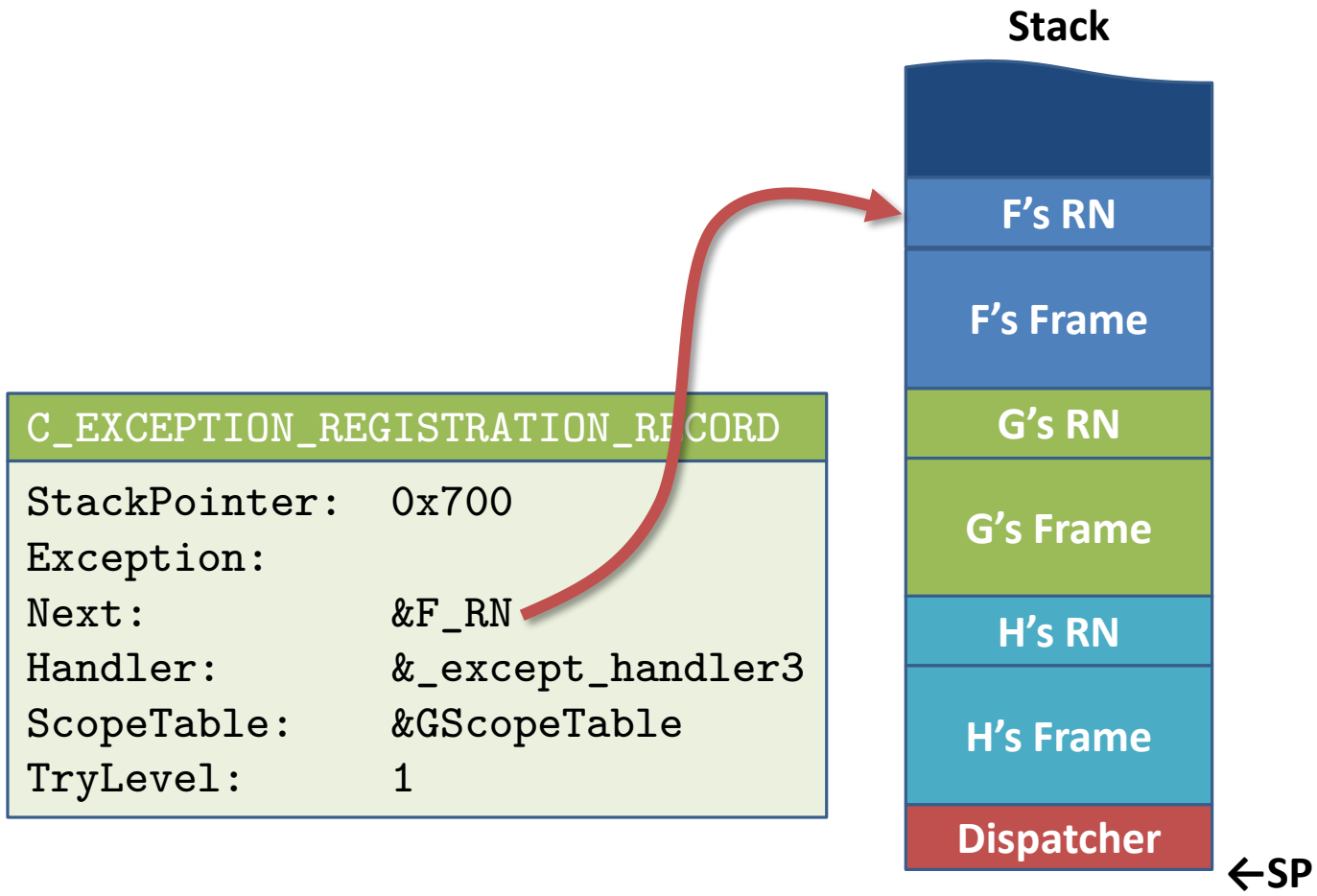
GFilter Advances to Enclosing TryLevel

```
void G()
{
    __try
    {
        __try
        {
            H();
        }
        __except(GFilter())
        {
        }
    }
    __finally
    {
    }
}
```

C_EXCEPTION_REGISTRATION_RECORD	
StackPointer:	0x700
Exception:	
Next:	&F_RN
Handler:	&_except_handler3
ScopeTable:	&GScopeTable
TryLevel:	1

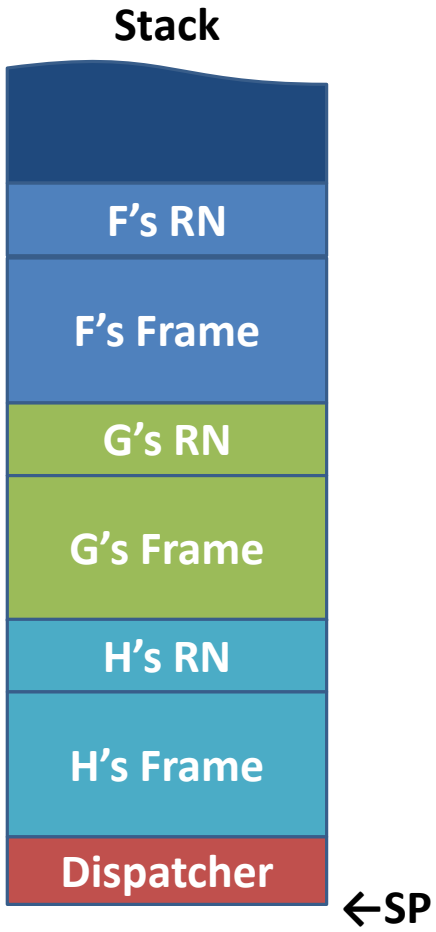


_except_handler3 for G Returns ExceptionContinueSearch



The Dispatcher Follows the Link to the Next Handler

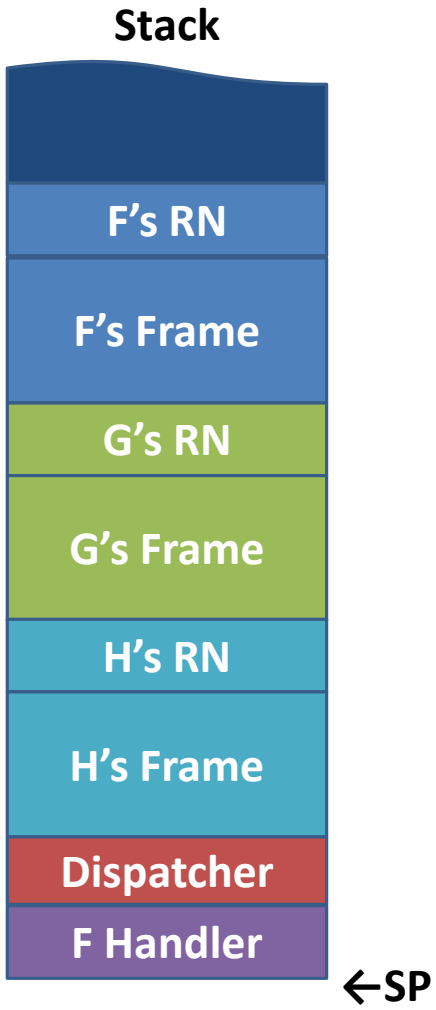
C_EXCEPTION_REGISTRATION_RECORD	
StackPointer:	0x800
Exception:	
Next:	0xFFFF'FFFF
Handler:	&_except_handler3
ScopeTable:	&FScopeTable
TryLevel:	1



The Dispatcher Follows the Link to the Next Handler

```
void F()
{
    __try
    {
        __try
        {
            G();
        }
        __except(FFilter())
        {
        }
    }
    __finally
    {
    }
}
```

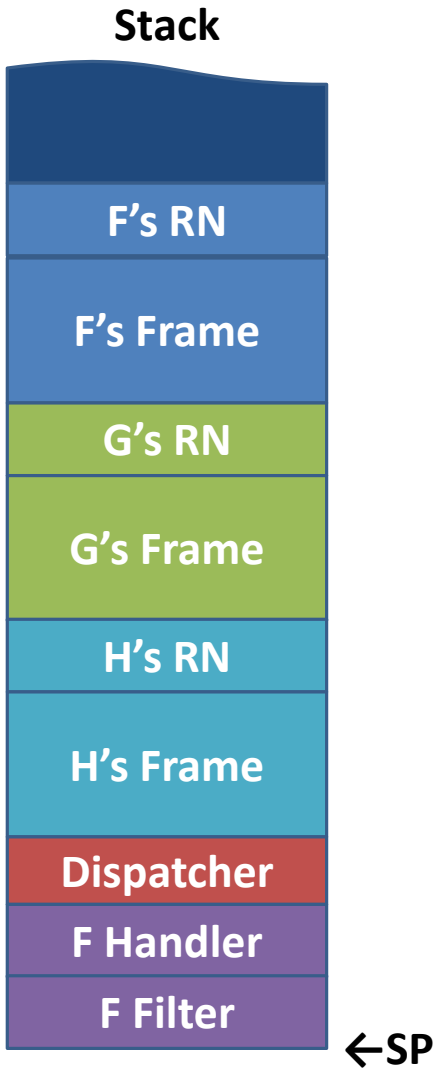
C_EXCEPTION_REGISTRATION_RECORD	
StackPointer:	0x800
Exception:	
Next:	0xFFFF'FFFF
Handler:	&_except_handler3
ScopeTable:	&FScopeTable
TryLevel:	1



_except_handler3 Called for F

```
void F()
{
    __try
    {
        __try
        {
            G();
        }
        __except(FFilter())
        {
        }
    }
    __finally
    {
    }
}



int FFilter()
{
    return EXCEPTION_EXECUTE_HANDLER;
}
```

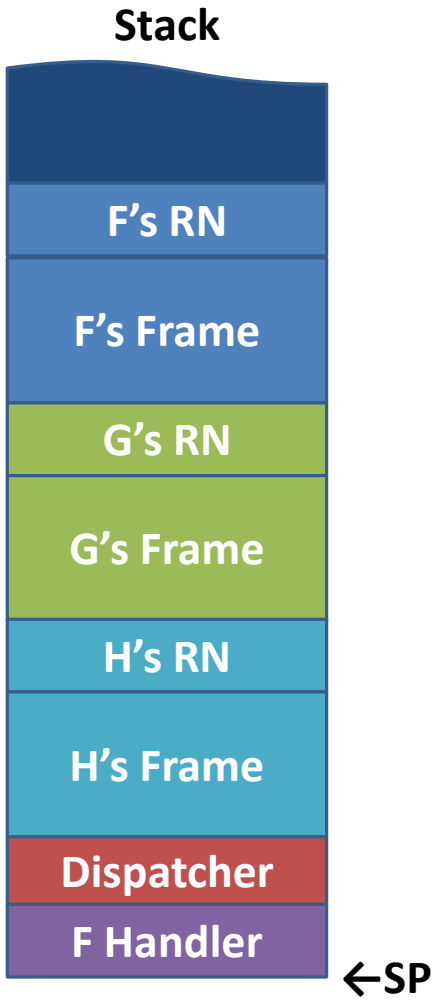
C_EXCEPTION_REGISTRATION_RECORD	
StackPointer:	0x800
Exception:	
Next:	0xFFFF'FFFF
Handler:	&_except_handler3
ScopeTable:	&FScopeTable
TryLevel:	1



_except_handler3 Calls FFilter

```
void F()
{
    __try
    {
        __try
        {
            G();
        }
        __except(FFilter())
        {
        }
    }
    __finally
    {
    }
}
```

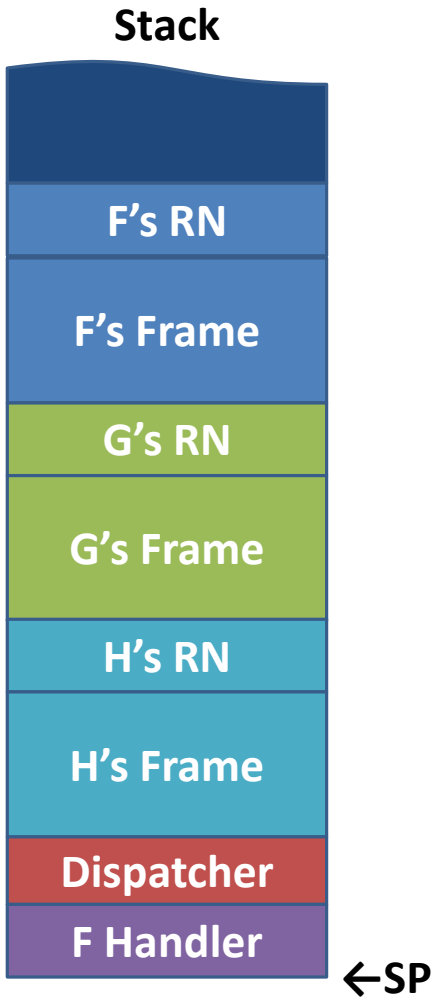
C_EXCEPTION_REGISTRATION_RECORD	
StackPointer:	0x800
Exception:	
Next:	0xFFFF'FFFF
Handler:	&_except_handler3
ScopeTable:	&FScopeTable
TryLevel:	1




```
int FFilter()
{
    return EXCEPTION_EXECUTE_HANDLER;
}
```

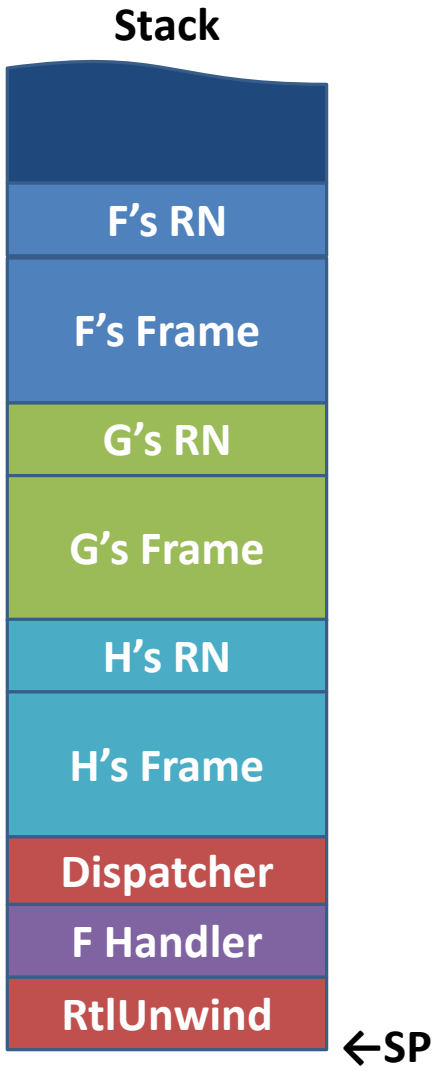
FFilter Returns EXCEPTION_EXECUTE_HANDLER

C_EXCEPTION_REGISTRATION_RECORD	
StackPointer:	0x800
Exception:	
Next:	0xFFFF'FFFF
Handler:	&_except_handler3
ScopeTable:	&FScopeTable
TryLevel:	1

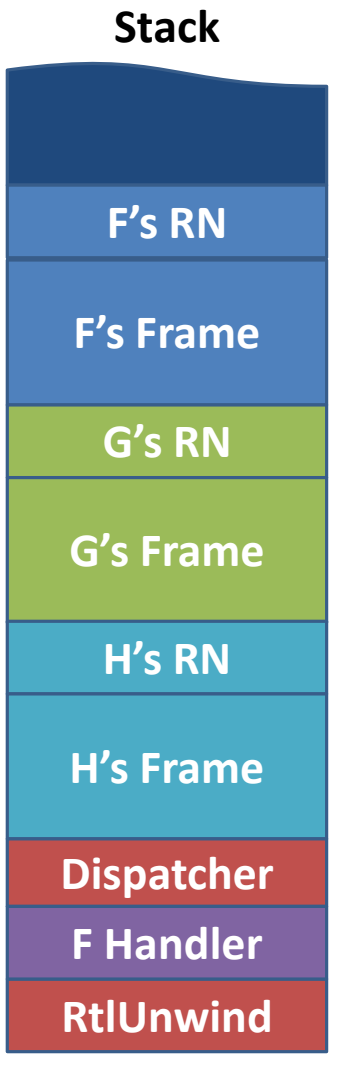



```
case EXCEPTION_EXECUTE_HANDLER:

    RtlUnwind(EstablisherFrame, ExceptionRecord);

    _local_unwind(RN, I);

    RN->ScopeTable[I].Handler();
```

_except_handler3 Needs to Unwind the Stack



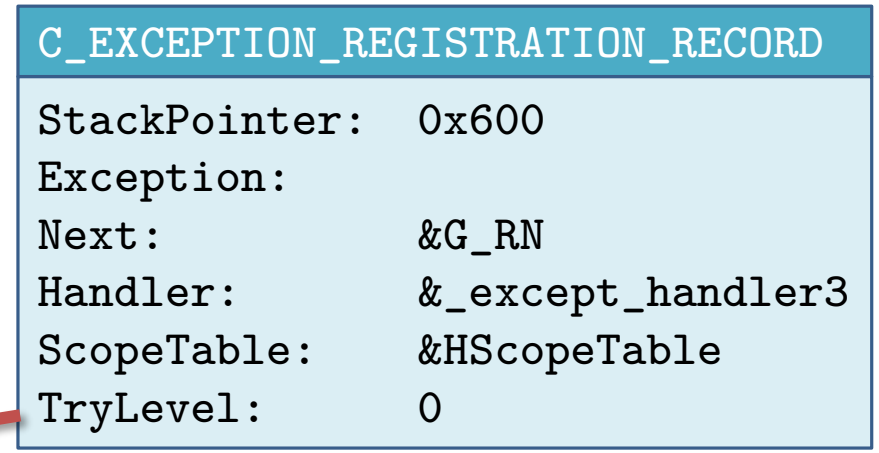
_except_handler3 Calls RtlUnwind(&F_RN)

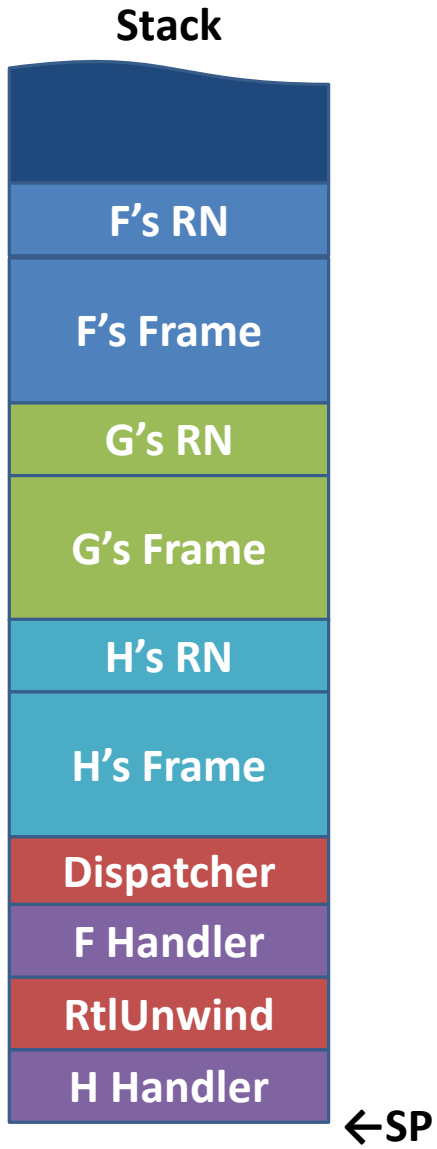



```
C_EXCEPTION_REGISTRATION_RECORD
StackPointer:   0x600
Exception:
Next:           &G_RN
Handler:        &_except_handler3
ScopeTable:     &HScopeTable
TryLevel:       0
```

RtlUnwind Gets The Most Recent Handler Registration

```
void H()
{
    __try
    {
        *(int*)nullptr = 0;
    }
    __finally
    {
    }
}
```

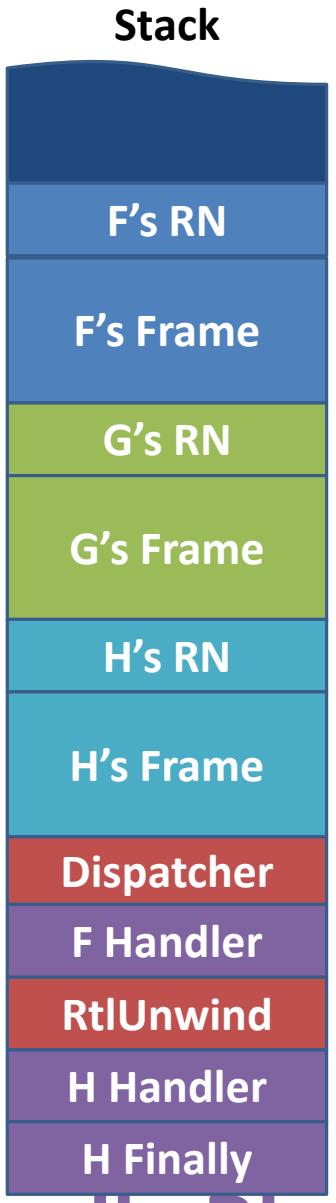




_except_handler3 Called for H

```
void H()
{
    __try
    {
        *(int*)nullptr = 0;
    }
    __finally
    {
    }
}
```

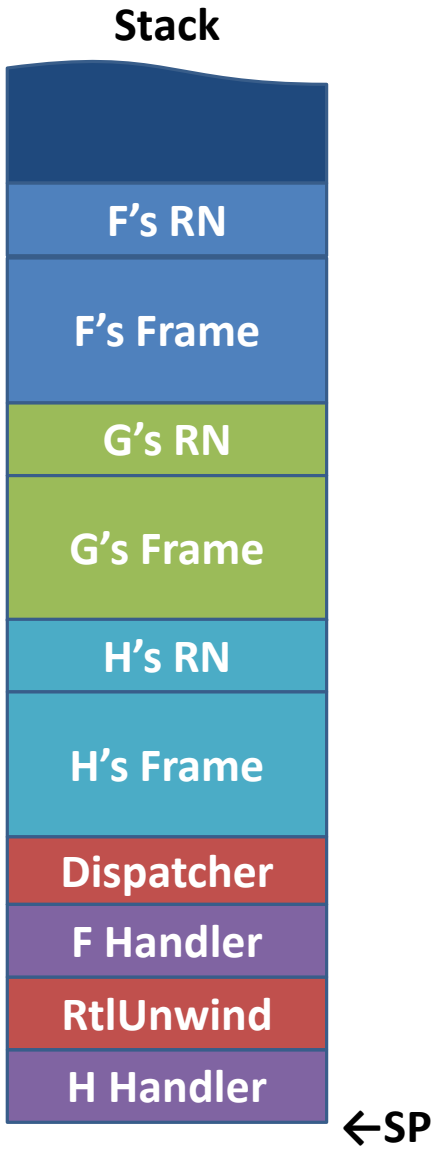
C_EXCEPTION_REGISTRATION_RECORD	
StackPointer:	0x600
Exception:	
Next:	&G_RN
Handler:	&_except_handler3
ScopeTable:	&HScopeTable
TryLevel:	0



_except_handler3 Calls H Finally Block

```
void H()
{
    __try
    {
        *(int*)nullptr = 0;
    }
    __finally
    {
    }
}
```

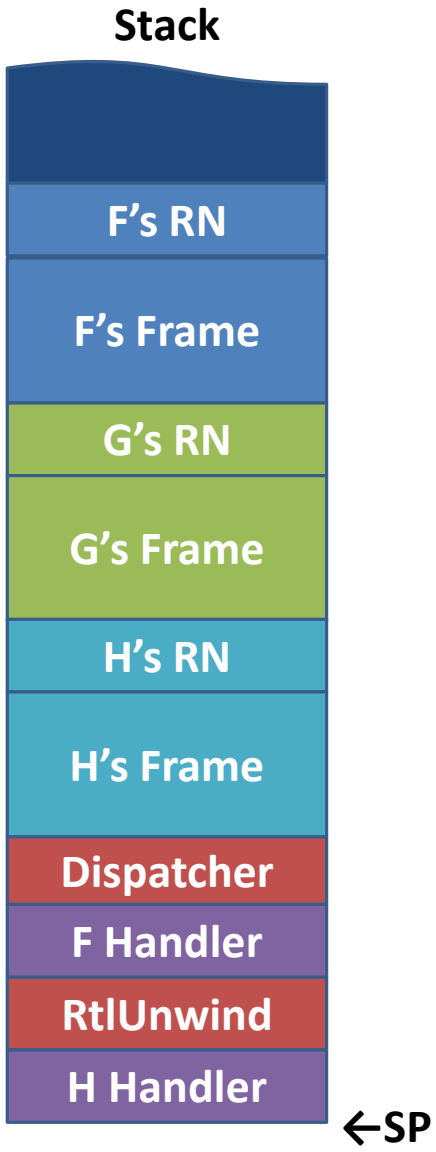
C_EXCEPTION_REGISTRATION_RECORD	
StackPointer:	0x600
Exception:	
Next:	&G_RN
Handler:	&_except_handler3
ScopeTable:	&HScopeTable
TryLevel:	0



H Finally Block Returns

```
void H()
{
    __try
    {
        *(int*)nullptr = 0;
    }
    __finally
    {
    }
}
```

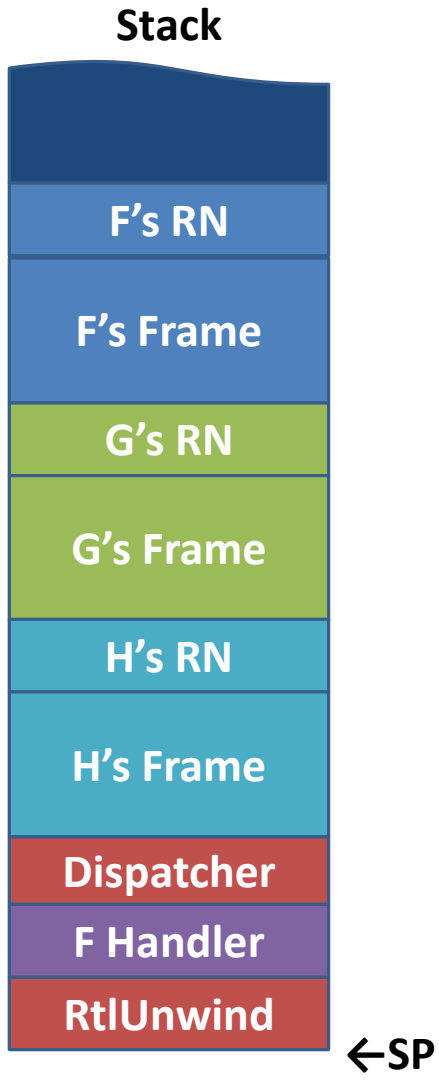
C_EXCEPTION_REGISTRATION_RECORD	
StackPointer:	0x600
Exception:	
Next:	&G_RN
Handler:	&_except_handler3
ScopeTable:	&HScopeTable
TryLevel:	-1



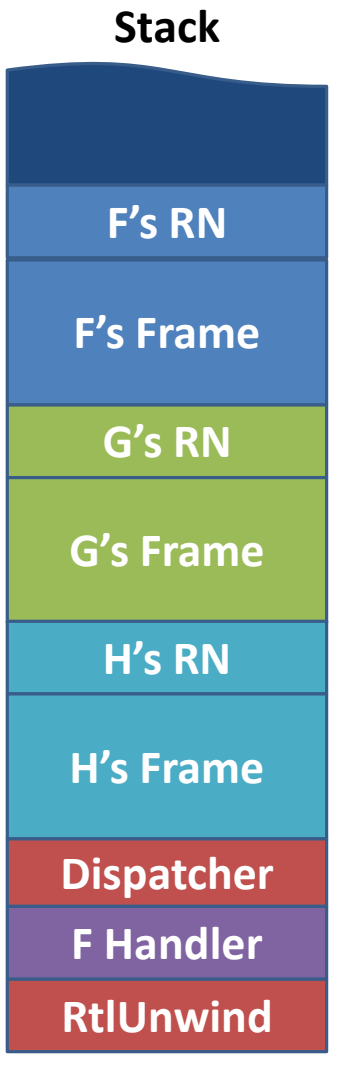
_except_handler3 for H Updates TryLevel

```
void H()
{
    __try
    {
        *(int*)nullptr = 0;
    }
    __finally
    {
    }
}
```

C_EXCEPTION_REGISTRATION_RECORD	
StackPointer:	0x600
Exception:	
Next:	&G_RN
Handler:	&_except_handler3
ScopeTable:	&HScopeTable
TryLevel:	-1

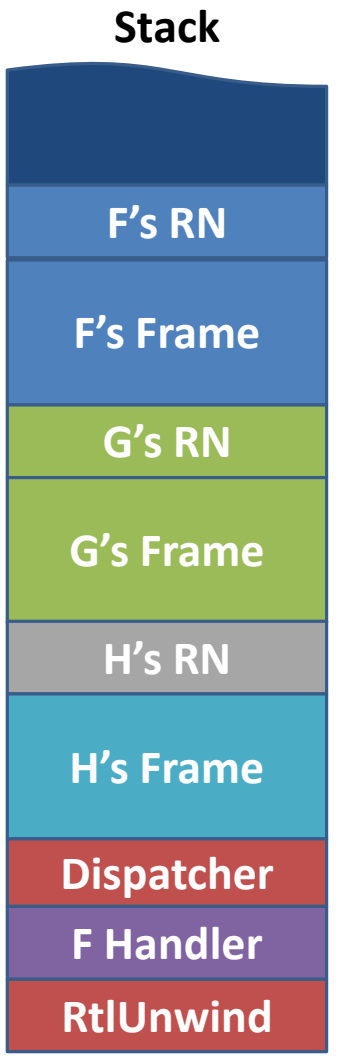


_except_handler3 for H Returns



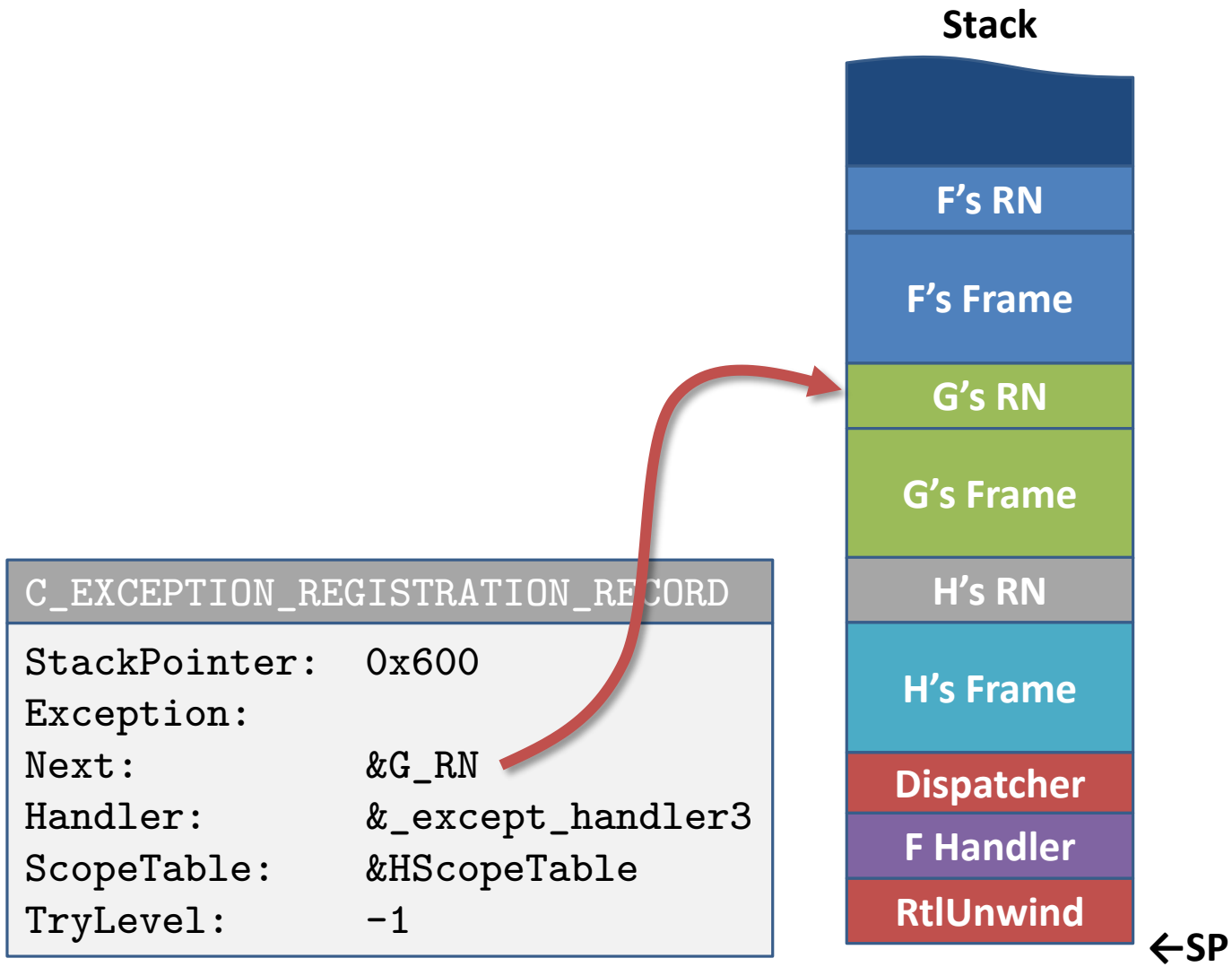
C_EXCEPTION_REGISTRATION_RECORD	
StackPointer:	0x600
Exception:	
Next:	&G_RN
Handler:	&_except_handler3
ScopeTable:	&HScopeTable
TryLevel:	-1

_except_handler3 for H Returns

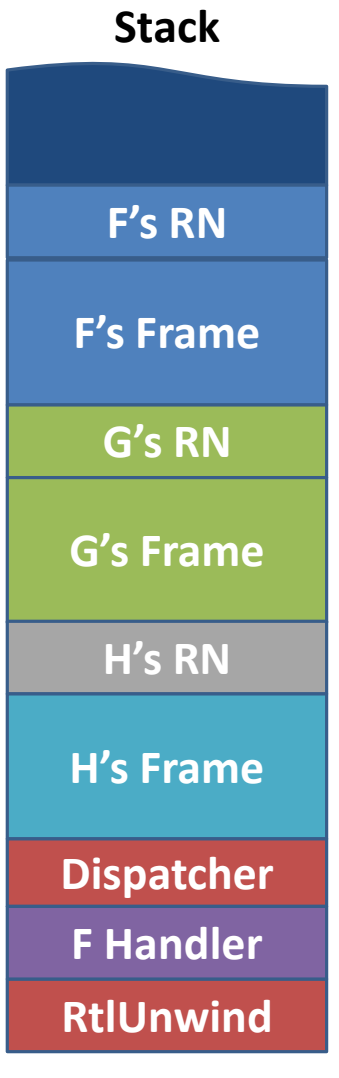


C_EXCEPTION_REGISTRATION_RECORD	
StackPointer:	0x600
Exception:	
Next:	&G_RN
Handler:	&_except_handler3
ScopeTable:	&HScopeTable
TryLevel:	-1

RtlUnwind Pops H's Handler from ExceptionList



RtlUnwind Follows the Link to the Next Handler

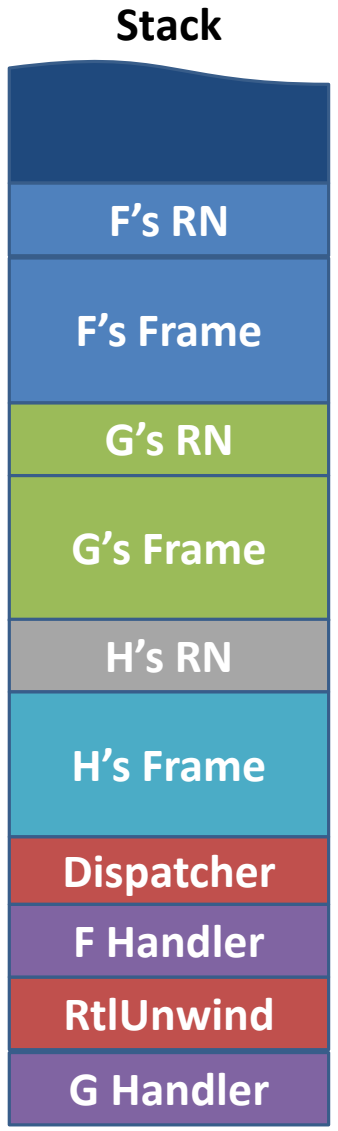



```
C_EXCEPTION_REGISTRATION_RECORD
StackPointer:   0x700
Exception:
Next:           &F_RN
Handler:        &_except_handler3
ScopeTable:     &GScopeTable
TryLevel:       1
```

RtlUnwind Follows the Link to the Next Handler

```
void G()
{
    __try
    {
        __try
        {
            H();
        }
        __except(GFilter())
        {
        }
    }
    __finally
    {
    }
}
```

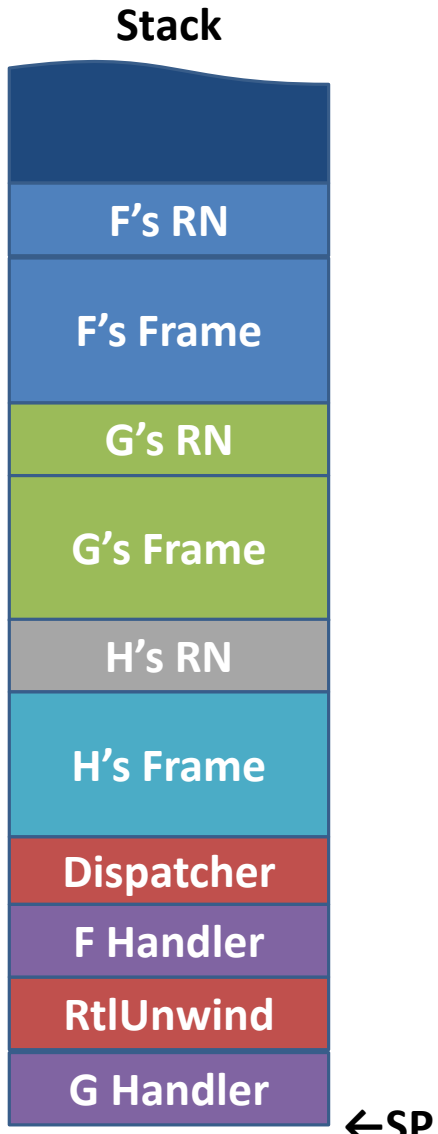
C_EXCEPTION_REGISTRATION_RECORD	
StackPointer:	0x700
Exception:	
Next:	&F_RN
Handler:	&_except_handler3
ScopeTable:	&GScopeTable
TryLevel:	1



_except_handler3 Called for G

```
void G()
{
    __try
    {
        __try
        {
            H();
        }
        __except(GFilter())
        {
        }
    }
    __finally
    {
    }
}
```

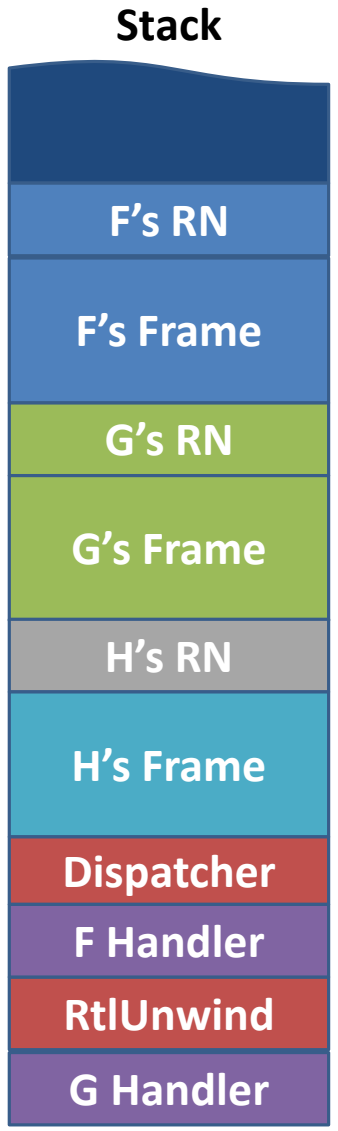
C_EXCEPTION_REGISTRATION_RECORD	
StackPointer:	0x700
Exception:	
Next:	&F_RN
Handler:	&_except_handler3
ScopeTable:	&GScopeTable
TryLevel:	0



_except_handler3 for G Updates TryLevel

```
void G()
{
    __try
    {
        __try
        {
            H();
        }
        __except(GFilter())
        {
        }
    }
    __finally
    {
    }
}
```

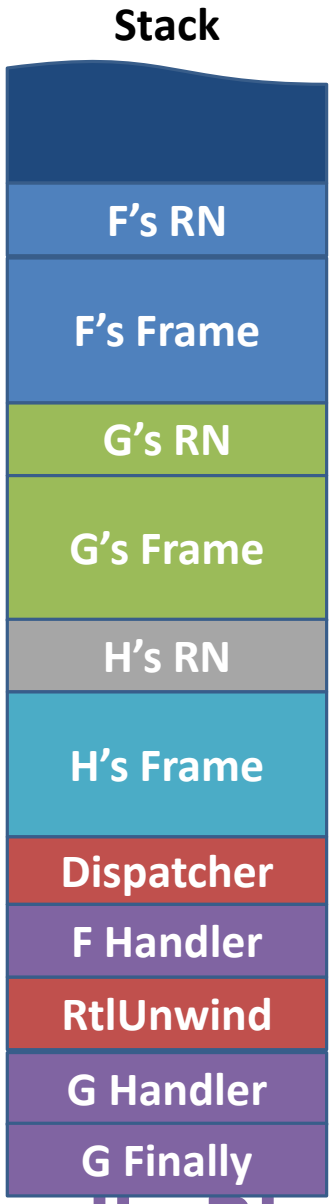
C_EXCEPTION_REGISTRATION_RECORD	
StackPointer:	0x700
Exception:	
Next:	&F_RN
Handler:	&_except_handler3
ScopeTable:	&GScopeTable
TryLevel:	0



_except_handler3 for G Updates TryLevel

```
void G()
{
    __try
    {
        __try
        {
            H();
        }
        __except(GFilter())
        {
        }
    }
    __finally
    {
    }
}
```

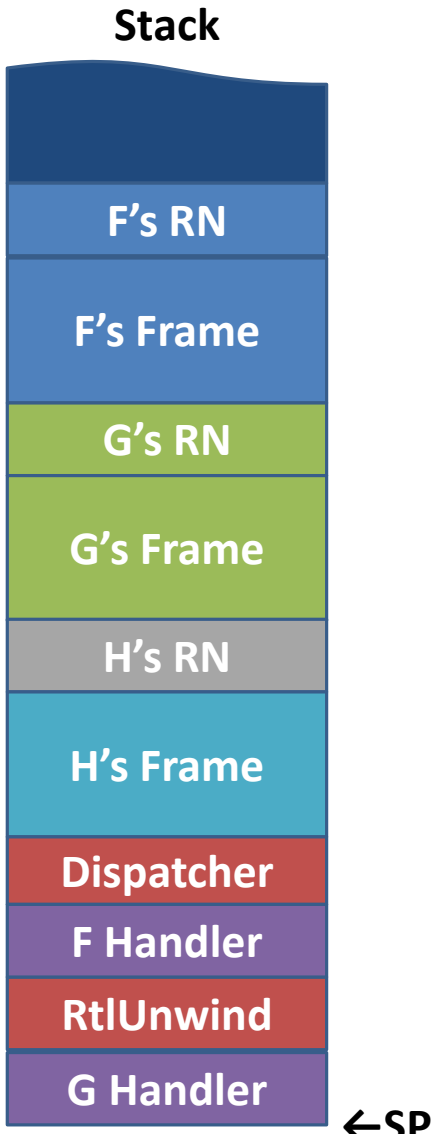
C_EXCEPTION_REGISTRATION_RECORD	
StackPointer:	0x700
Exception:	
Next:	&F_RN
Handler:	&_except_handler3
ScopeTable:	&GScopeTable
TryLevel:	0



_except_handler3 for G Calls Finally Block

```
void G()
{
    __try
    {
        __try
        {
            H();
        }
        __except(GFilter())
        {
        }
    }
    __finally
    {
    }
}
```

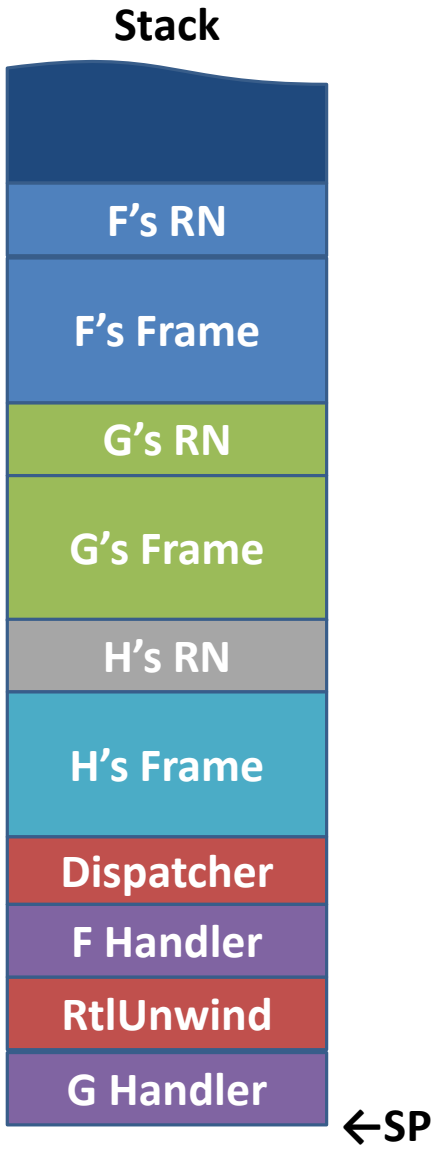
C_EXCEPTION_REGISTRATION_RECORD	
StackPointer:	0x700
Exception:	
Next:	&F_RN
Handler:	&_except_handler3
ScopeTable:	&GScopeTable
TryLevel:	0



G Finally Block Returns

```
void G()
{
    __try
    {
        __try
        {
            H();
        }
        __except(GFilter())
        {
        }
    }
    __finally
    {
    }
}
```

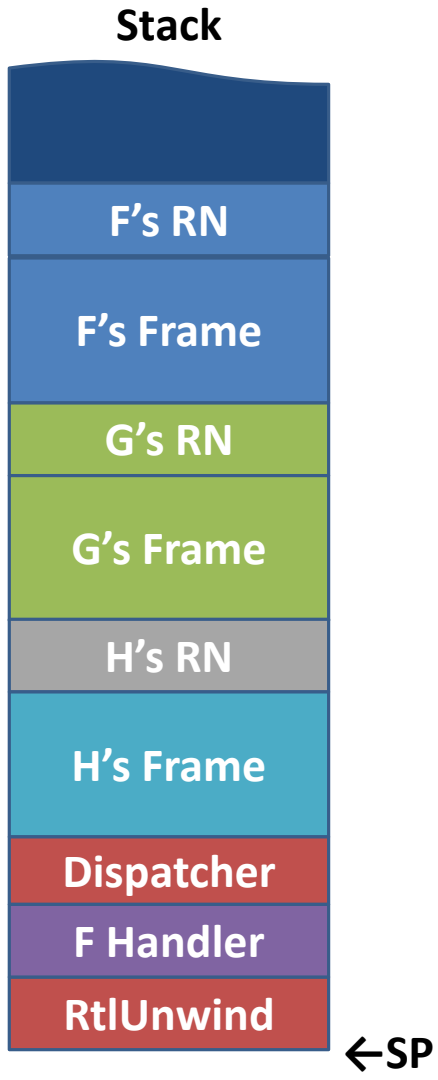
C_EXCEPTION_REGISTRATION_RECORD	
StackPointer:	0x700
Exception:	
Next:	&F_RN
Handler:	&_except_handler3
ScopeTable:	&GScopeTable
TryLevel:	-1



_except_handler3 for G Updates TryLevel

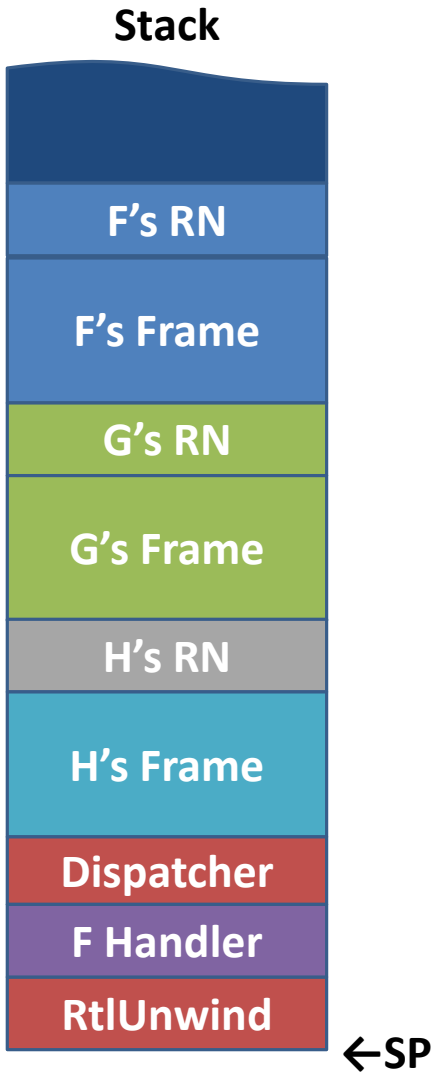
```
void G()
{
    __try
    {
        __try
        {
            H();
        }
        __except(GFilter())
        {
        }
    }
    __finally
    {
    }
}
```

C_EXCEPTION_REGISTRATION_RECORD	
StackPointer:	0x700
Exception:	
Next:	&F_RN
Handler:	&_except_handler3
ScopeTable:	&GScopeTable
TryLevel:	-1



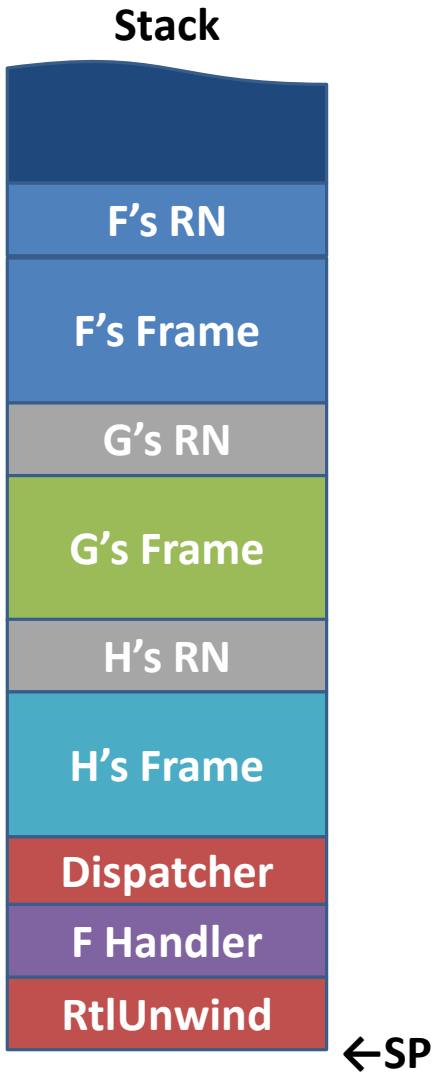
Handler for G Returns

C_EXCEPTION_REGISTRATION_RECORD	
StackPointer:	0x700
Exception:	
Next:	&F_RN
Handler:	&_except_handler3
ScopeTable:	&GScopeTable
TryLevel:	-1

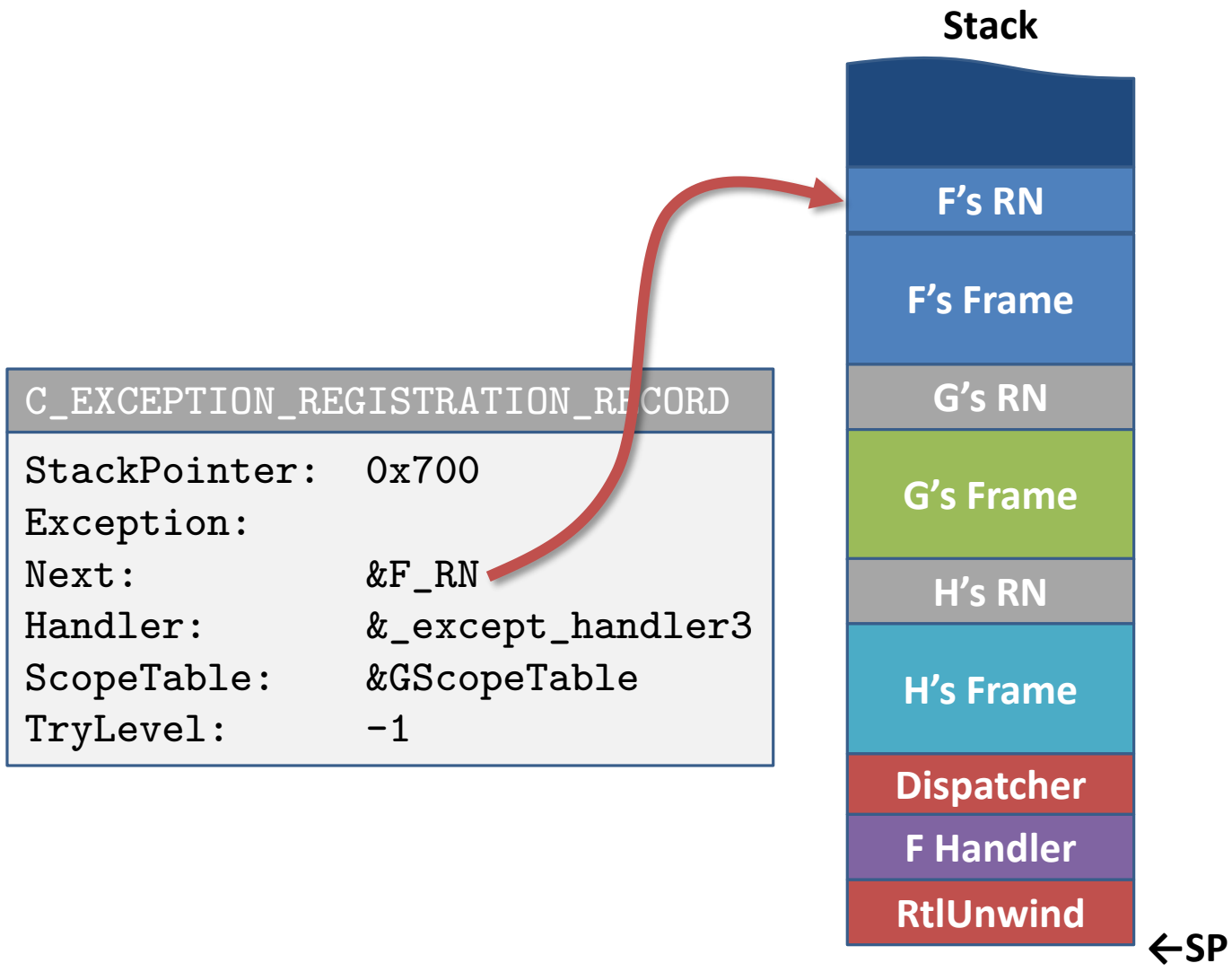


Handler for G Returns

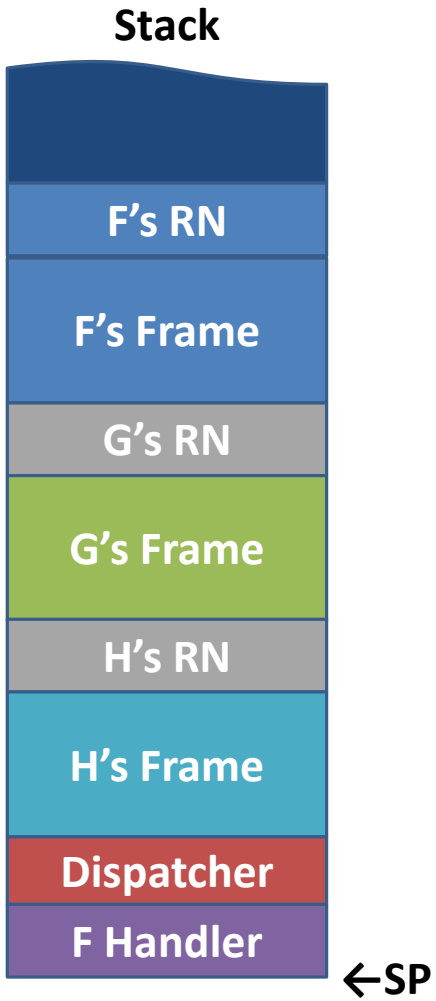
C_EXCEPTION_REGISTRATION_RECORD	
StackPointer:	0x700
Exception:	
Next:	&F_RN
Handler:	&_except_handler3
ScopeTable:	&GScopeTable
TryLevel:	-1



RtlUnwind Pops G's Handler from ExceptionList

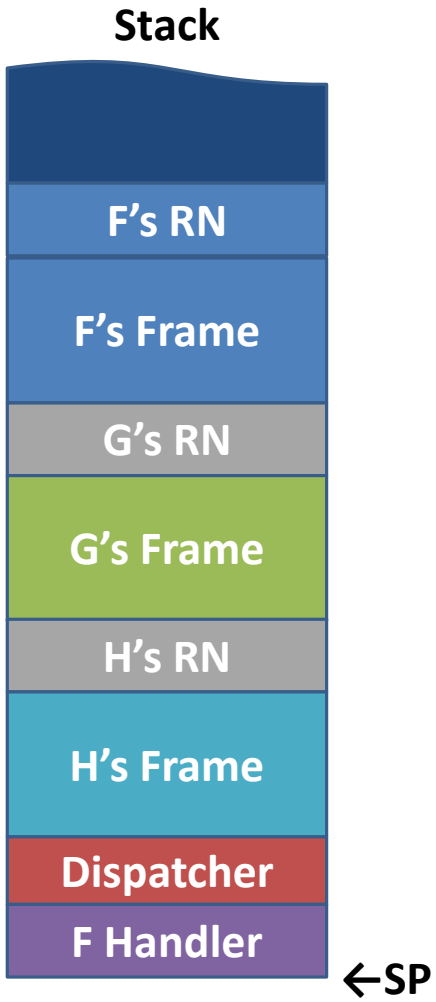


RtlUnwind Follows the Link to the Next Handler



RtlUnwind Returns

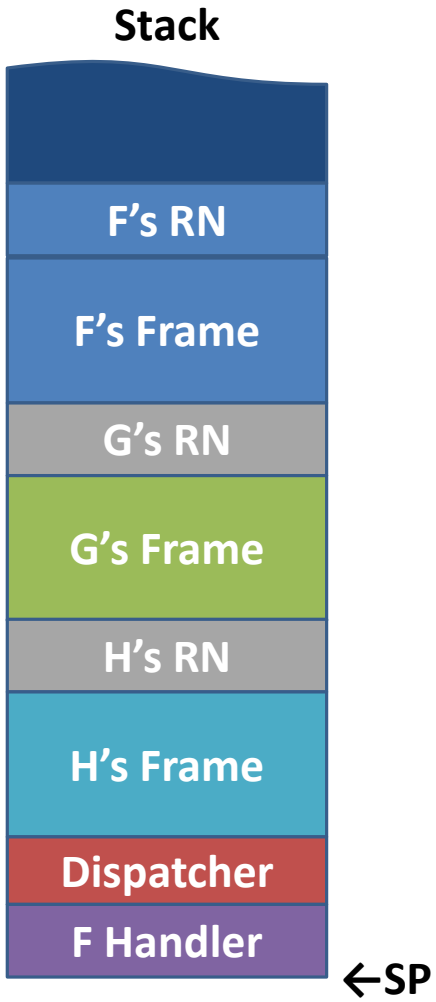
C_EXCEPTION_REGISTRATION_RECORD	
StackPointer:	0x800
Exception:	
Next:	0xFFFF'FFFF
Handler:	&_except_handler3
ScopeTable:	&FScopeTable
TryLevel:	1




```
case EXCEPTION_EXECUTE_HANDLER:

    RtlUnwind(EstablisherFrame, ExceptionRecord);

    _local_unwind(RN, I);

    RN->ScopeTable[I].Handler();
```

Back Inside F's Handler…

C_EXCEPTION_REGISTRATION_RECORD	
StackPointer:	0x800
Exception:	
Next:	0xFFFF'FFFF
Handler:	&_except_handler3
ScopeTable:	&FScopeTable
TryLevel:	1

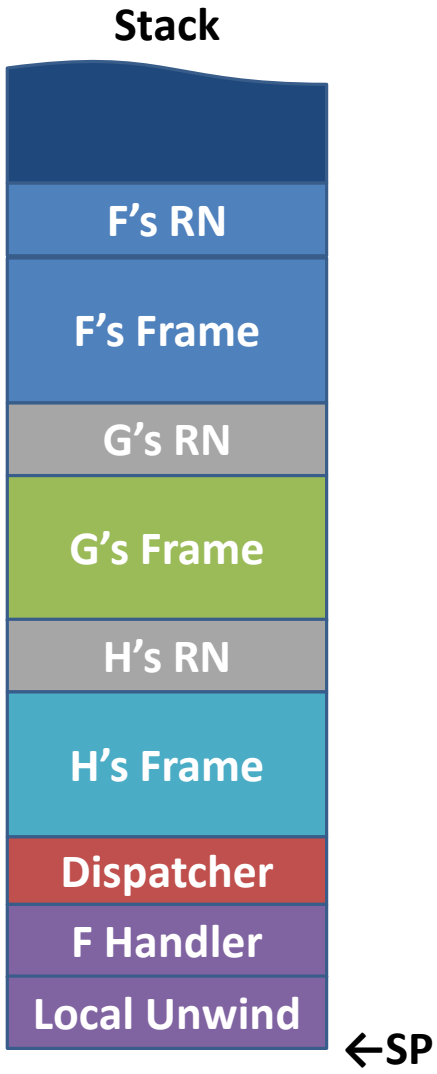



```
case EXCEPTION_EXECUTE_HANDLER:

    RtlUnwind(EstablisherFrame, ExceptionRecord);

    _local_unwind(RN, I);

    RN->ScopeTable[I].Handler();
```

Back Inside F's Handler…

```
void F()
{
    __try
    {
        __try
        {
            G();
        }
        __except(FFilter())
        {
        }
    }
    __finally
    {
    }
}
```

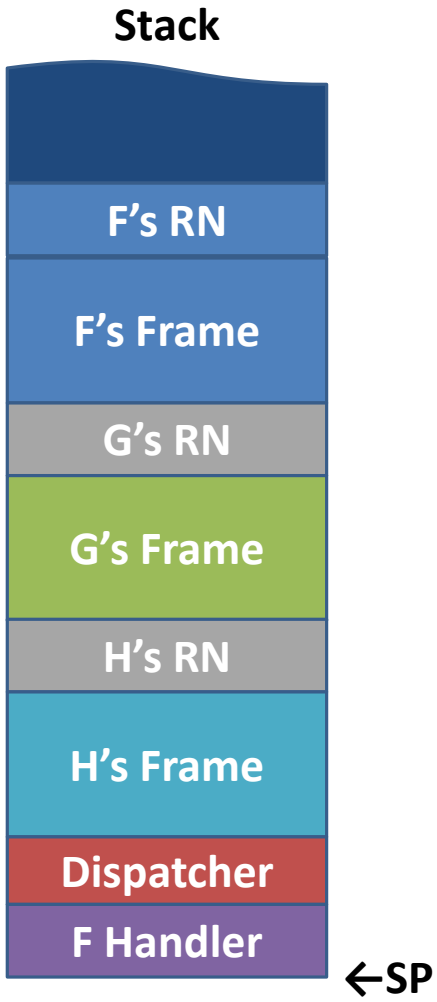
C_EXCEPTION_REGISTRATION_RECORD	
StackPointer:	0x800
Exception:	
Next:	0xFFFF'FFFF
Handler:	&_except_handler3
ScopeTable:	&FScopeTable
TryLevel:	1



F Calls _local_unwind to unwind its state

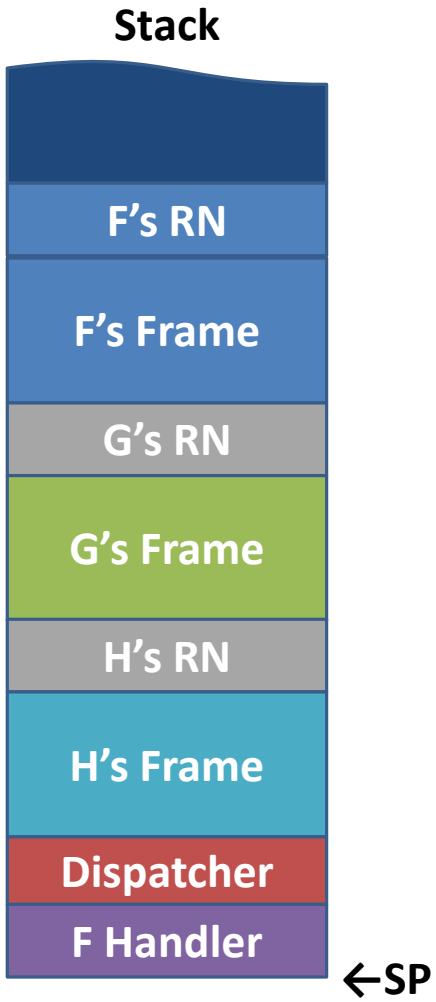
```
void F()
{
    __try
    {
        __try
        {
            G();
        }
        __except(FFilter())
        {
        }
    }
    __finally
    {
    }
}
```

C_EXCEPTION_REGISTRATION_RECORD	
StackPointer:	0x800
Exception:	
Next:	0xFFFF'FFFF
Handler:	&_except_handler3
ScopeTable:	&FScopeTable
TryLevel:	1



Local Unwinder Returns

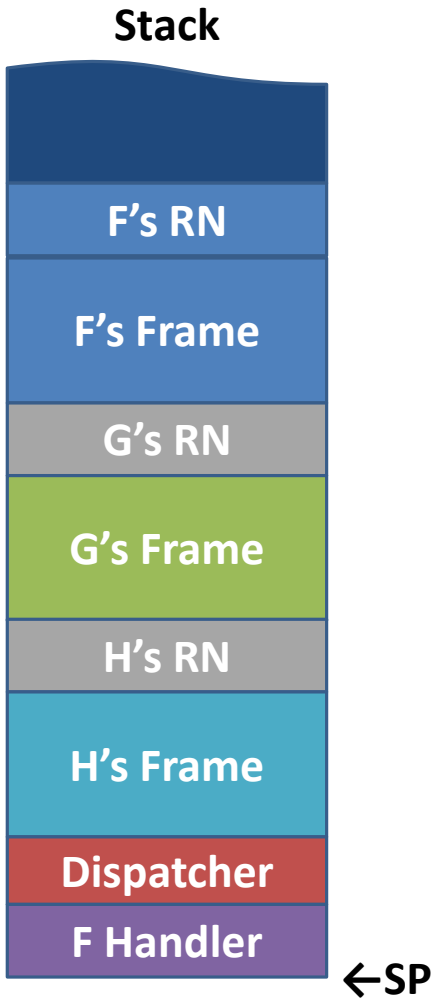
C_EXCEPTION_REGISTRATION_RECORD	
StackPointer:	0x800
Exception:	
Next:	0xFFFF'FFFF
Handler:	&_except_handler3
ScopeTable:	&FScopeTable
TryLevel:	1




```
case EXCEPTION_EXECUTE_HANDLER:

    RtlUnwind(EstablisherFrame, ExceptionRecord);

    _local_unwind(RN, RN->TryLevel);

    RN->ScopeTable[I].Handler();
```

Back Inside F's Handler…

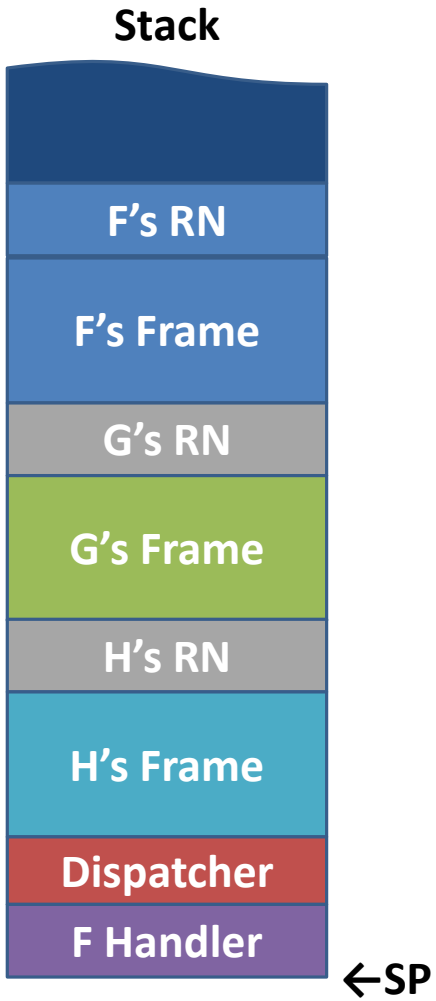
C_EXCEPTION_REGISTRATION_RECORD	
StackPointer:	0x800
Exception:	
Next:	0xFFFF'FFFF
Handler:	&_except_handler3
ScopeTable:	&FScopeTable
TryLevel:	1




```
case EXCEPTION_EXECUTE_HANDLER:

    RtlUnwind(EstablisherFrame, ExceptionRecord);

    _local_unwind(RN, RN->TryLevel);

    RN->ScopeTable[I].Handler();
```

Back Inside F's Handler…

C_EXCEPTION_REGISTRATION_RECORD	
StackPointer:	0x800
Exception:	
Next:	0xFFFF'FFFF
Handler:	&_except_handler3
ScopeTable:	&FScopeTable
TryLevel:	1

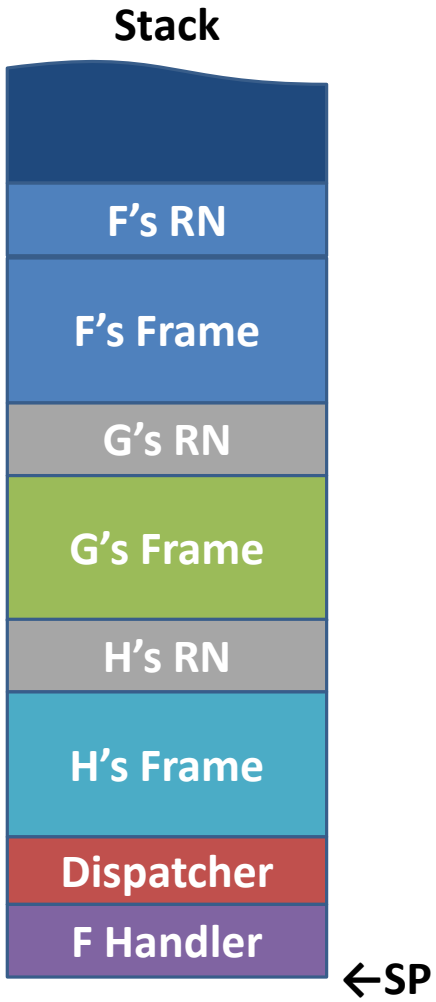



```
case EXCEPTION_EXECUTE_HANDLER:

    RtlUnwind(EstablisherFrame, ExceptionRecord);

    _local_unwind(RN, RN->TryLevel);

    RN->ScopeTable[I].Handler();
```

Handler for F Transfers Control to __except Block

```
void F()
{
    __try
    {
        __try
        {
            G();
        }
        __except(FFilter())
        {
        }
    }
    __finally
    {
    }
}
```

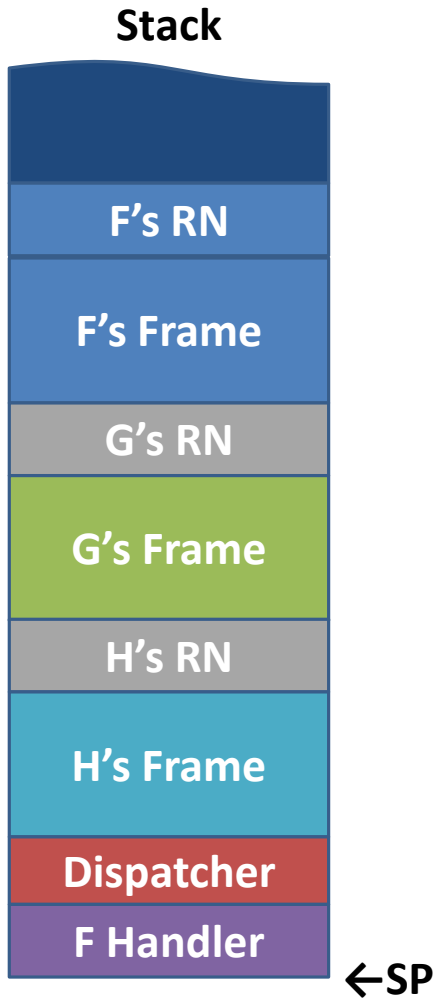
C_EXCEPTION_REGISTRATION_RECORD	
StackPointer:	0x800
Exception:	
Next:	0xFFFF'FFFF
Handler:	&_except_handler3
ScopeTable:	&FScopeTable
TryLevel:	1



Handler for F Transfers Control to __except Block

```
void F()
{
    __try
    {
        __try
        {
            G();
        }
        __except(FFilter())
        {
        }
    }
    __finally
    {
    }
}
```

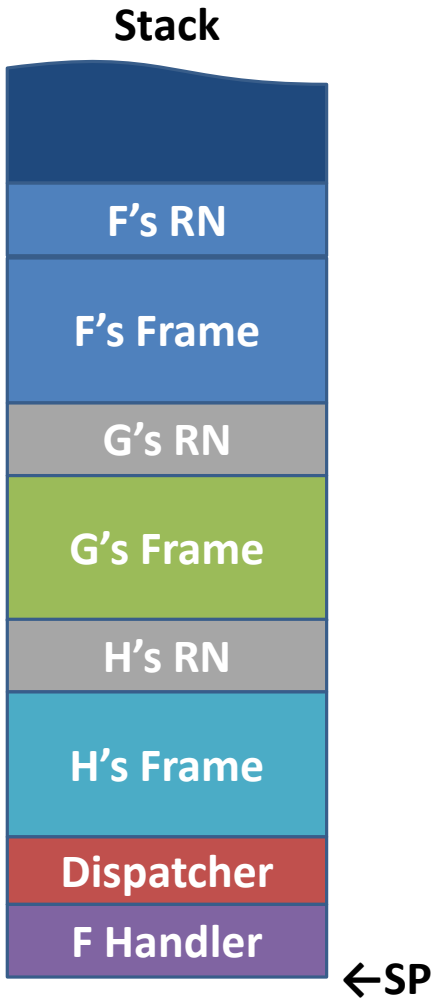
C_EXCEPTION_REGISTRATION_RECORD	
StackPointer:	0x800
Exception:	
Next:	0xFFFF'FFFF
Handler:	&_except_handler3
ScopeTable:	&FScopeTable
TryLevel:	0



__except Block Updates TryLevel

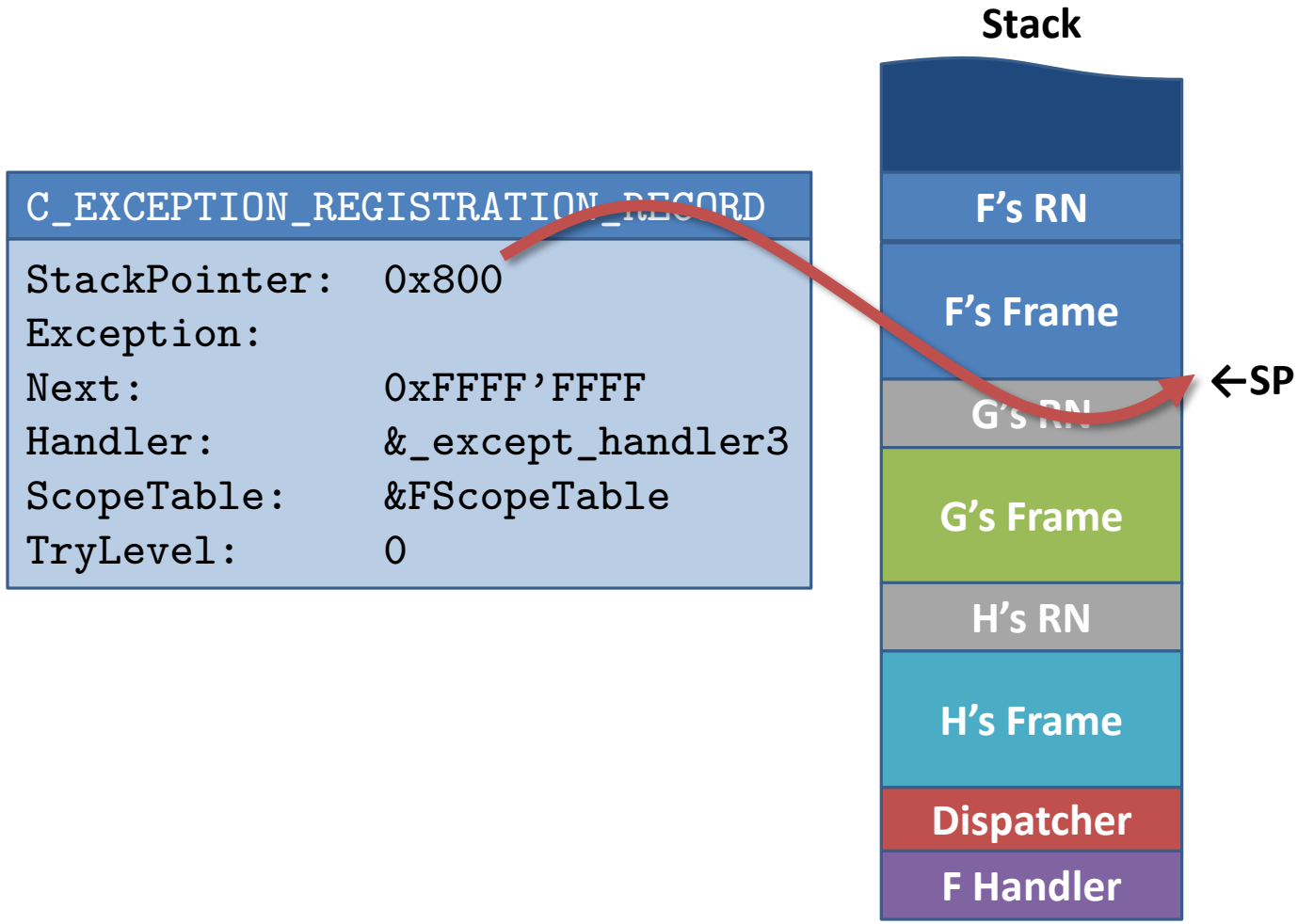
```
void F()
{
    __try
    {
        __try
        {
            G();
        }
        __except(FFilter())
        {
        }
    }
    __finally
    {
    }
}
```

C_EXCEPTION_REGISTRATION_RECORD	
StackPointer:	0x800
Exception:	
Next:	0xFFFF'FFFF
Handler:	&_except_handler3
ScopeTable:	&FScopeTable
TryLevel:	0



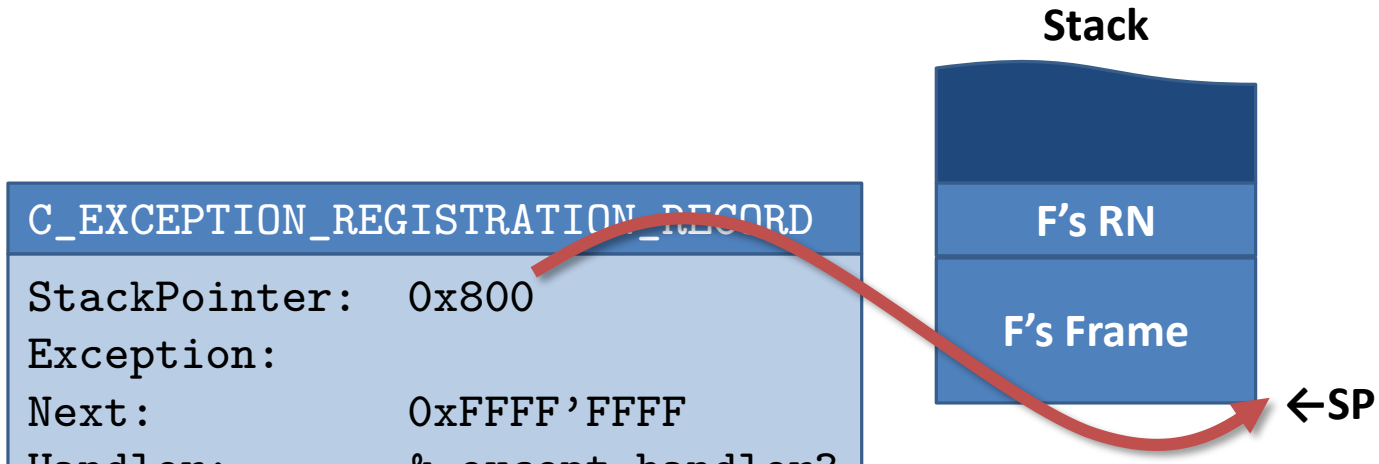
__except Block Updates Stack Pointer

```
void F()
{
    __try
    {
        __try
        {
            G();
        }
        __except(FFilter())
        {
        }
    }
    __finally
    {
    }
}
```



__except Block Updates Stack Pointer

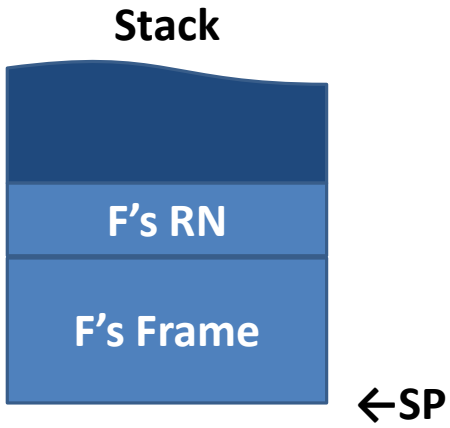
```
void F()
{
    __try
    {
        __try
        {
            G();
        }
        __except(FFilter())
        {
        }
    }
    __finally
    {
    }
}
```



__except Block Updates Stack Pointer

```
void F()
{
    __try
    {
        __try
        {
            G();
        }
        __except(FFilter())
IP→     {
        }
    }
    __finally
    {
    }
}
```

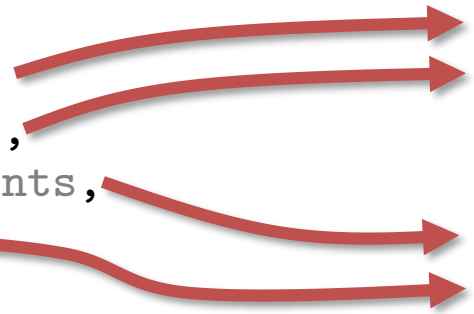
C_EXCEPTION_REGISTRATION_RECORD	
StackPointer:	0x800
Exception:	
Next:	0xFFFF'FFFF
Handler:	&_except_handler3
ScopeTable:	&FScopeTable
TryLevel:	0



__except Block Begins Execution

An Exception of Your Own…

```
void RaiseException(
    DWORD             ExceptionCode,
    DWORD             ExceptionFlags,
    DWORD             NumberOfArguments,
    ULONG_PTR const*  Arguments
    );
```



```
struct EXCEPTION_RECORD
{
    DWORD             ExceptionCode;
    DWORD             ExceptionFlags;
    EXCEPTION_RECORD* ExceptionRecord;
    void*             ExceptionAddress;
    DWORD             NumberParameters;
    ULONG_PTR         ExceptionInformation[15];
};
```

RaiseException

```
constexpr DWORD STATUS_COFFEE_SHORTAGE = 0xCOFFEE;

void RaiseCoffeeShortage()
{
    RaiseException(
        STATUS_COFFEE_SHORTAGE,
        0,
        0,
        nullptr);
}
```

RaiseException

```
int main()
{
    __try
    {
        RaiseCoffeeShortage();
    }
    __except(MainFilter(GetExceptionInformation()))
    {
        puts("Oh no!  A coffee shortage has occurred!");
    }
}



int MainFilter(EXCEPTION_POINTERS const* Exception)
{
    if (Exception->ExceptionRecord->ExceptionCode != STATUS_COFFEE_SHORTAGE)
    {
        return EXCEPTION_CONTINUE_SEARCH;
    }

    return EXCEPTION_EXECUTE_HANDLER;
}
```

RaiseException

When are we finally going to talk about C++?

```
throw                          =>   RaiseException

try/catch                      =>   __try/__except

Local variable destruction     =>   __try/__finally

_CxxFrameHandler               =>   _except_handler3
```

C++ Exception Handling

Throwing

```
[[noreturn]] void _CxxThrowException(
    void*       ExceptionObject,
    _ThrowInfo* ThrowInfo
    );
```

```
throw MyBeautifulException{};
```

```
MyBeautifulException ExceptionObject{};

_CxxThrowException(
    &ExceptionObject,
    &_ThrowInfoFor<MyBeautifulException>);
```

throw

```
struct _ThrowInfo
{
    unsigned int         attributes;
    Destructor*          pmfnUnwind;
    CatchableTypeArray* pCatchableTypeArray;
};

struct CatchableTypeArray
{
    int             nCatchableTypes;
    CatchableType* arrayOfCatchableTypes[nCatchableTypes];
};

struct CatchableType
{
    unsigned int     properties;
    std::type_info*  pType;
    PMD              thisDisplacement;
    int              sizeOrOffset;
    CopyConstructor* copyFunction;
};
```

_ThrowInfo

```
struct BaseException
{
    int BaseData;
};
```

```
struct DerivedException : BaseException
{
    std::string DerivedData;
};
```

ThrowInfo for BaseException
(__TI1?AUBaseException@@)

```
 * attributes             = 0;
 * pmfnUnwind             = nullptr;
 * pCatchableTypeArray[1]
```

ThrowInfo for DerivedException
(__TI2?AUDerivedException@@)

```
 * attributes             = 0;
 * pmfnUnwind             = &~DerivedException;
 * pCatchableTypeArray[2]
```

CatchableType for BaseException
(__CT??_R0?AUBaseException@@@84)

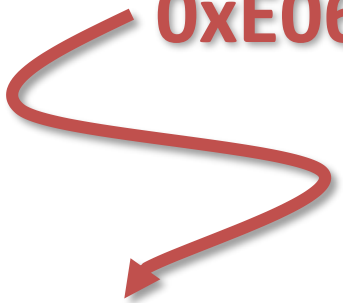
```
 * properties   = 0;
 * pType        = &typeid(BaseException);
 * sizeOrOffset = 4;
 * copyFunction = nullptr;
```

CatchableType for DerivedException
(__CT??_R0?AUDerivedException@@@88)

```
 * properties   = 0
 * pType        = &typeid(DerivedException);
 * sizeOrOffset = 32;
 * copyFunction = &DerivedException(Copy Ctor);
```

_ThrowInfo for our Exception Types

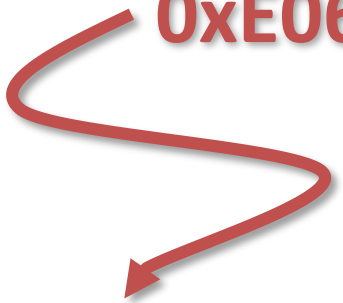
```
[[noreturn]] void _CxxThrowException(
    void*       ExceptionObject,
    _ThrowInfo* ThrowInfo
    )
{
    EXCEPTION_RECORD Exception;

    Exception.ExceptionCode             = EH_EXCEPTION_NUMBER;









}
```



0xE06D7363

_CxxThrowException

```
[[noreturn]] void _CxxThrowException(
    void*       ExceptionObject,
    _ThrowInfo* ThrowInfo
    )
{
    EXCEPTION_RECORD Exception;

    Exception.ExceptionCode             = EH_EXCEPTION_NUMBER;
    Exception.ExceptionFlags            = EXCEPTION_NONCONTINUABLE;

    Exception.NumberParameters          = 3;
    Exception.ExceptionInformation[0] = EH_MAGIC_NUMBER1;
    Exception.ExceptionInformation[1] = (ULONG_PTR)ExceptionObject;
    Exception.ExceptionInformation[2] = (ULONG_PTR)ThrowInfo;

    RaiseException(
        Exception.ExceptionCode,
        Exception.ExceptionFlags,
        Exception.NumberParameters,
        Exception.ExceptionInformation);
}
```



0xE06D7363 ('msc' | 0xE0000000)

_CxxThrowException

```
int main()
{
    __try
    {
        throw DerivedException{};
    }
    __except (VisualCppExceptionFilter(GetExceptionCode()))
    {
    }
}

int VisualCppExceptionFilter(DWORD ExceptionCode)
{
    if (ExceptionCode == 0xE06D7363)
    {
        return EXCEPTION_EXECUTE_HANDLER;
    }
    else
    {
        return EXCEPTION_CONTINUE_SEARCH;
    }
}
```

Handling a C++ Exception, SEH-Style

Catching and Unwinding

```
// SEH
struct C_EXCEPTION_REGISTRATION_RECORD
{
    void*                           StackPointer;
    EXCEPTION_POINTERS*             Exception;
    EXCEPTION_REGISTRATION_RECORD   HandlerRegistration;
    SCOPETABLE_ENTRY*               ScopeTable;
    int                             TryLevel;
};

// C++
struct EHRegistrationNode
{
    void*                            StackPointer;
    EXCEPTION_REGISTRATION_RECORD    HandlerRegistration;
    int                              State;
};
```

EHRegistrationNode

```
__ehhandler$?F@@YAXXZ:
    mov     eax, &FuncInfoForF
    jmp     ___CxxFrameHandler3 (0E51055h)
```

Handler Thunks

```
int main()
{
    std::string A = "A";
    std::string B = "B";

    try
    {
        std::string C = "C";
        F();
        // C.~std::string();
    }
    catch (BaseException)
    {
        std::string D = "D";
        G();
        // D.~std::string();
    }

    // B.~std::string();
    // A.~std::string();
}
```

```
    std::string A = "A";
push offset string "A" (0287B34h)
lea  ecx,[A]
call std::string::string (0281528h)
mov  byte ptr [RN.State], 0
    std::string B = "B";
push offset string "B" (0287B38h)
lea  ecx,[B]
call std::string::string (0281528h)
mov  byte ptr [RN.State], 1

    try
mov  byte ptr [RN.State], 2
    {
```

We'll Split the Function Into States

```
struct FuncInfo
{
    int                 maxState;
    UnwindMapEntry*     pUnwindMap;

    unsigned int        nTryBlocks;
    TryBlockMapEntry*   pTryBlockMap;
};
```

FuncInfo

```
struct FuncInfo
{
    int                 maxState;
    UnwindMapEntry*     pUnwindMap;

    unsigned int        nTryBlocks;
    TryBlockMapEntry*   pTryBlockMap;
};

struct TryBlockMapEntry
{
    int tryLow;     // Lowest state index of try
    int tryHigh;    // Highest state index of try
    int catchHigh;  // Highest state index of any associated catch

    int           nCatches;      // Number of entries in array
    HandlerType*  pHandlerArray; // List of handlers for this try
};
```

FuncInfo

```
struct FuncInfo
{
    int                 maxState;
    UnwindMapEntry*     pUnwindMap;

    unsigned int        nTryBlocks;
    TryBlockMapEntry*   pTryBlockMap;
};

struct TryBlockMapEntry
{
    int tryLow;    // Lowest state index of try
    int tryHigh;   // Highest state index of try
    int catchHigh; // Highest state index of any associated catch

    int           nCatches;      // Number of entries in array
    HandlerType*  pHandlerArray; // List of handlers for this try
};

struct HandlerType
{
    std::type_info* pType;             // Pointer to the corresponding type descriptor
    ptrdiff_t       dispCatchObj;      // Displacement of catch object from base
    void*           addressOfHandler; // Address of 'catch' code
};
```

FuncInfo

- Ignores non-C++ exceptions (returns ExceptionContinueSearch)

- Using the FuncInfo and the current State, computes the range of try blocks whose catch blocks should be considered

- For each try block (from innermost to outermost), it enumerates the associated catch blocks

- For each catch block, it checks to see if the type is a “match” for the type of the thrown object
  - The HandlerType has a std::type_info
  - The ThrowInfo has a CatchableTypeArray, which is basically an array of std::type_infos

__CxxFrameHandler

- If a catch block matches, it:
  - Initializes the catch object
  - Performs a global unwind (RtlUnwind) to unwind nested frames
  - Performs a local unwind to unwind local frames
  - Calls the catch block

__CxxFrameHandler

- If a catch block matches, it:
  - Initializes the catch object
  - Performs a global unwind (RtlUnwind) to unwind nested frames
  - Performs a local unwind to unwind local frames
  - Calls the catch block

- There are two ways for the catch block to exit:
  - It can return normally, in which case the continuation is executed
  - It can rethrow (via a ‘throw;’), in which case the original exception is re-raised

- If no catch block matches, ExceptionContinueSearch is returned

__CxxFrameHandler

```
void __scrt_set_unhandled_exception_filter()
{
    SetUnhandledExceptionFilter(__scrt_unhandled_exception_filter);
}


LONG __scrt_unhandled_exception_filter(EXCEPTION_POINTERS* Pointers)
{
    if (Pointers->ExceptionRecord->ExceptionCode == EH_EXCEPTION_NUMBER)
    {
        std::terminate();
    }

    return EXCEPTION_CONTINUE_SEARCH;
}
```

When There's No Matching Catch Block

```
void f() throw(BaseException);
```

Exception Specifications

```
void f() noexcept;
```

noexcept

/Eha

/Ehs

/EHsc

https://docs.microsoft.com/en-us/cpp/build/reference/eh-exception-handling-model

Exception Handling Models

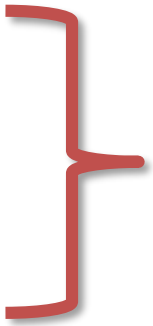
What about other architectures?

What about other architectures?

Each function that registers an exception handler always registers the same handler

An Observation

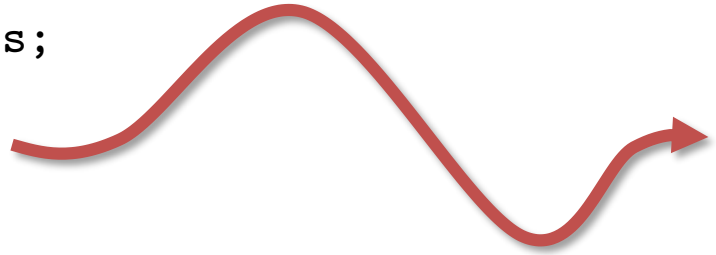
```
F => FHandler

G => GHandler

H => HHandler
```



.pdata

A Static Table of Handlers

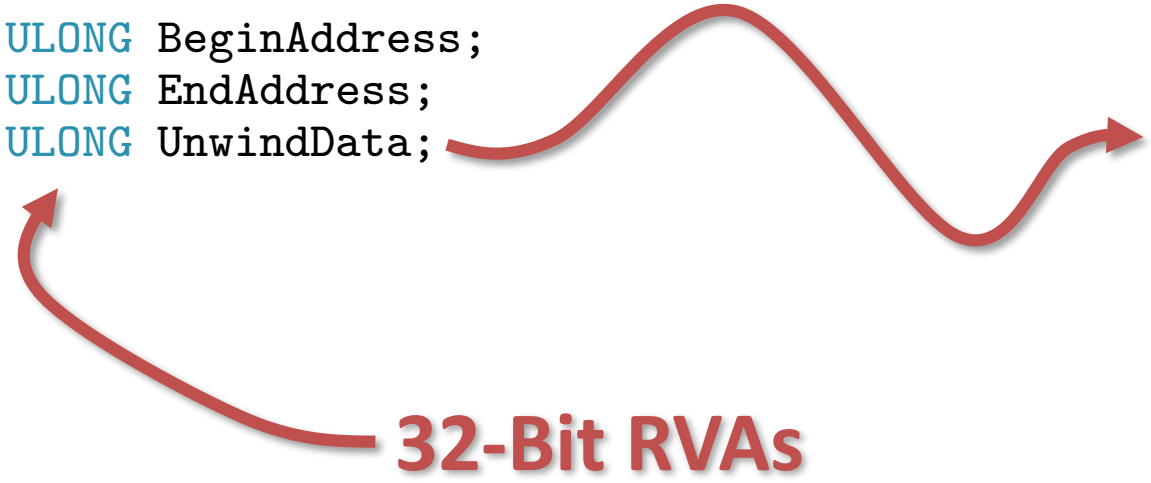
```
struct RUNTIME_FUNCTION
{
    ULONG BeginAddress;
    ULONG EndAddress;
    ULONG UnwindData;
};
```



```
UNWIND_INFO:
 * Stack walking instructions
 * Exception handler address
 * Language-specific metadata
```

.pdata

```
struct RUNTIME_FUNCTION
{
    ULONG BeginAddress;
    ULONG EndAddress;
    ULONG UnwindData;
};
```




```
UNWIND_INFO:
 * Stack walking instructions
 * Exception handler address
 * Language-specific metadata
```

.pdata

Resumable C++ Exceptions?

```
void bar()
{
    if (condition)
    {
        throw resume int(7); // Throw a resumable exception
    }
    else
    {
        throw int(7);        // Throw a terminating exception
    }
}
```

```
void foo()
{
    try
    {
        bar();
    }
    catch (resume int)
    {
        if (condition)
        {
            resume; // Resume the resumable exception
        }
        else if (condition)
        {
            throw resume; // Rethrow the exception as resumable
        }
        else
        {
            throw; // Rethrow as terminating
        }
    }
}
```

Resumable Exceptions in C++ (X3J16/90-0042)

```
void foo()
{
    try
    {
        bar();
    }
    catch (int)
    {
        if (condition)
        {
            throw resume; // Rethrow as resumable (to here)
        }
        else if (condition)
        {
            throw; // Rethrow as terminating
        }
        else
        {
            resume; // Resume (Error!)
        }
    }
}
```

Resumable Exceptions in C++ (X3J16/90-0042)

The End.

```
C:\Program Files (x86)\Microsoft Visual Studio\2017\Enterprise\VC\Tools\MSVC\[build]\crt\src\
```

For More Information, Consult The Sources…