The new LLVM exception handling scheme

Duncan Sands

DeepBlueCapital / CNRS

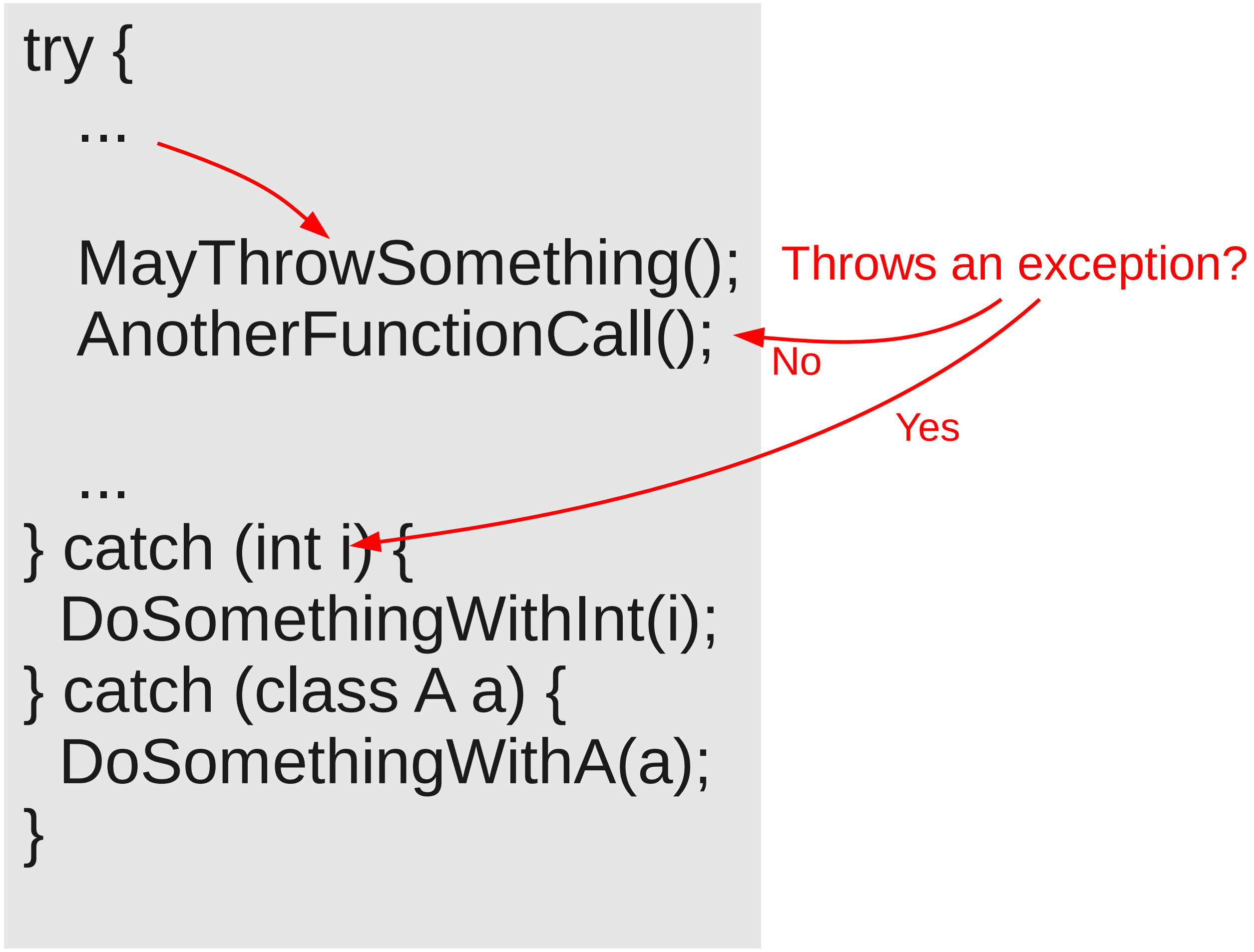
Control flow

```
try {
   ...

   MayThrowSomething();
   AnotherFunctionCall();

   ...
} catch (int i) {
  DoSomethingWithInt(i);
} catch (class A a) {
  DoSomethingWithA(a);
}
```

Control flow

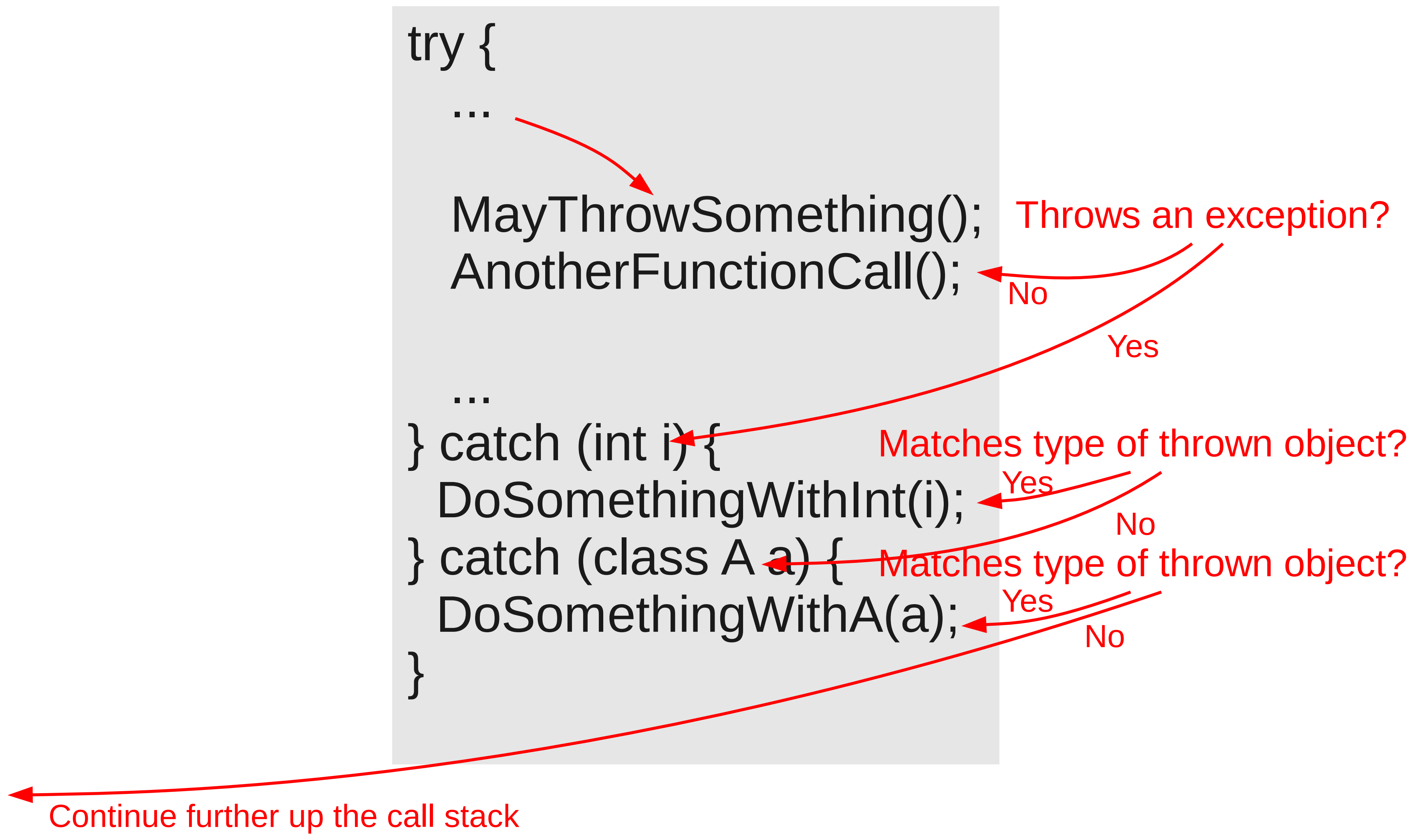
```
try {
   ...

   MayThrowSomething();
   AnotherFunctionCall();

   ...
} catch (int i) {
  DoSomethingWithInt(i);
} catch (class A a) {
  DoSomethingWithA(a);
}
```

Control flow



```
try {
   ...

   MayThrowSomething();
   AnotherFunctionCall();


   ...
} catch (int i) {
  DoSomethingWithInt(i);
} catch (class A a) {
  DoSomethingWithA(a);
}
```

Control flow



```
try {
   ...

   MayThrowSomething();
   AnotherFunctionCall();

   ...
} catch (int i) {
  DoSomethingWithInt(i);
} catch (class A a) {
  DoSomethingWithA(a);
}
```

Throws an exception?
No
Yes
Matches type of thrown object?
Yes
No
Matches type of thrown object?
Yes
No
Continue further up the call stack

LLVM constructs

```
try {
   ...

   MayThrowSomething();
   AnotherFunctionCall();

   ...
} catch (int i) {
  DoSomethingWithInt(i);
} catch (class A a) {
  DoSomethingWithA(a);
}
```

```
invoke void @_Z17MayThrowSomethingv()
    ...
```

LLVM constructs

```
try {
   ...

   MayThrowSomething();
   AnotherFunctionCall();

   ...
} catch (int i) {
  DoSomethingWithInt(i);
} catch (class A a) {
  DoSomethingWithA(a);
}
```

```
invoke void @_Z17MayThrowSomethingv()
    to label %cont unwind label %lpad
```

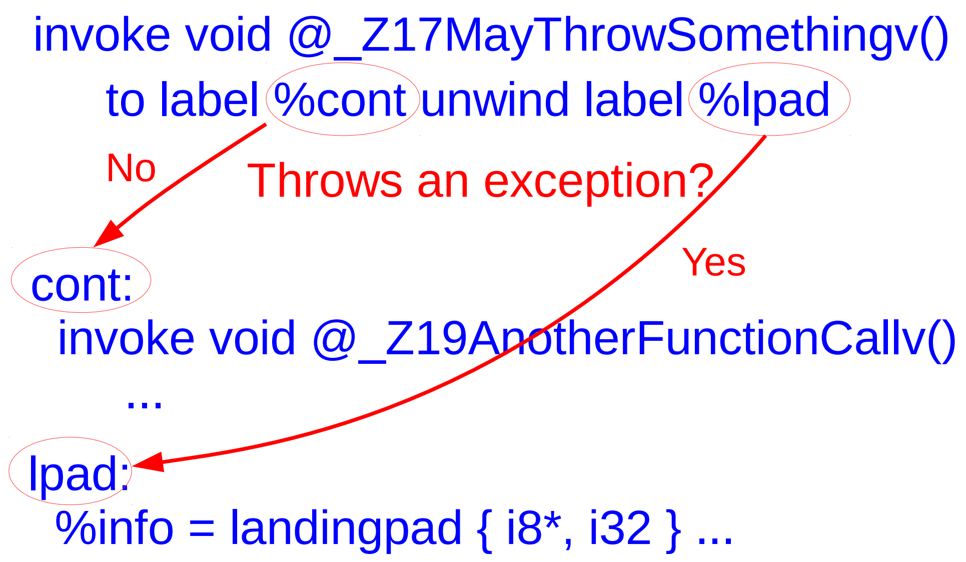
No

Throws an exception?

```
cont:
  invoke void @_Z19AnotherFunctionCallv()
      ...
```

LLVM constructs

```
try {
   ...

   MayThrowSomething();
   AnotherFunctionCall();


   ...
} catch (int i) {
  DoSomethingWithInt(i);
} catch (class A a) {
  DoSomethingWithA(a);
}
```



```
invoke void @_Z17MayThrowSomethingv()
    to label %cont unwind label %lpad

cont:
  invoke void @_Z19AnotherFunctionCallv()
     ...

lpad:
  %info = landingpad { i8*, i32 } ...
```

LLVM constructs

```
try {
   ...

   MayThrowSomething();
   AnotherFunctionCall();

   ...
} catch (int i) {
  DoSomethingWithInt(i);
} catch (class A a) {
  DoSomethingWithA(a);
}
```

```
invoke void @_Z17MayThrowSomethingv()
    to label %cont unwind label %lpad

cont:
  invoke void @_Z19AnotherFunctionCallv()
     ...
lpad:
  %info = landingpad { i8*, i32 } ...
```

The type of %info

Information describing the exception

LLVM constructs

```
try {
   ...

   MayThrowSomething();
   AnotherFunctionCall();

   ...
} catch (int i) {
  DoSomethingWithInt(i);
} catch (class A a) {
  DoSomethingWithA(a);
}
```

```
invoke void @_Z17MayThrowSomethingv()
    to label %cont unwind label %lpad

cont:
  invoke void @_Z19AnotherFunctionCallv()
      ...
lpad:
  %info = landingpad { i8*, i32 } ...
```

The type of %info

Information describing the exception

Exception pointer part

Selector value part

LLVM constructs

```
try {
   ...

   MayThrowSomething();
   AnotherFunctionCall();


   ...
} catch (int i) {
  DoSomethingWithInt(i);
} catch (class A a) {
  DoSomethingWithA(a);
}
```

```
invoke void @_Z17MayThrowSomethingv()
    to label %cont unwind label %lpad


cont:
  invoke void @_Z19AnotherFunctionCallv()
      ...

lpad:
  %info = landingpad { i8*, i32 }
    personality @__gxx_personality_v0
    catch @_ZTIi
    catch @_ZTI1A
```

Simplified version:
real version has types

LLVM constructs

```
try {
   ...

   MayThrowSomething();
   AnotherFunctionCall();


   ...
} catch (int i) {
  DoSomethingWithInt(i);
} catch (class A a) {
  DoSomethingWithA(a);
}
```

```
invoke void @_Z17MayThrowSomethingv()
     to label %cont unwind label %lpad


cont:
  invoke void @_Z19AnotherFunctionCallv()
       ...
lpad:
  %info = landingpad { i8*, i32 }
    personality @__gxx_personality_v0
    catch @_ZTIi
    catch @_ZTI1A
```

Language specific personality function
Knows how to compare the exception with a catch condition (C++ type)

LLVM constructs

```
try {
   ...

   MayThrowSomething();
   AnotherFunctionCall();

   ...
} catch (int i) {
  DoSomethingWithInt(i);
} catch (class A a) {
  DoSomethingWithA(a);
}
```

```
invoke void @_Z17MayThrowSomethingv()
    to label %cont unwind label %lpad

cont:
  invoke void @_Z19AnotherFunctionCallv()
      ...

lpad:
  %info = landingpad { i8*, i32 }
    personality @__gxx_personality_v0
    catch @_ZTIi
    catch @_ZTI1A
```

} Type infos: language specific global variables that represent the catch condition (C++ types)

LLVM constructs

```
  ...
} catch (int i) {
  DoSomethingWithInt(i);
} catch (class A a) {
  DoSomethingWithA(a);
}
```

```
lpad:
  %info = landingpad { i8*, i32 }
    personality @__gxx_personality_v0
    catch @_ZTIi
    catch @_ZTI1A
  %except = extractvalue { i8*, i32 } %info, 0
  %selector = extractvalue { i8*, i32 } %info, 1
  %typeid = call i32 @llvm.eh.typeid.for(@_ZTIi)
  %match = icmp eq i32 %selector, %typeid
  br i1 %match, label %run_catch, label %try_next

run_catch:
  %thrown = call i8* @__cxa_begin_catch(%except)
  %tmp = bitcast i8* %thrown to i32*
  %i = load i32* %tmp
  call void @_Z18DoSomethingWithInti(i32 %i)
  ...
```

The exception object

LLVM constructs

```
  ...
} catch (int i) {
  DoSomethingWithInt(i);
} catch (class A a) {
  DoSomethingWithA(a);
}
```

```
lpad:
  %info = landingpad { i8*, i32 }
    personality @__gxx_personality_v0
    catch @_ZTIi
    catch @_ZTI1A
  %except = extractvalue { i8*, i32 } %info, 0
  %selector = extractvalue { i8*, i32 } %info, 1
  %typeid = call i32 @llvm.eh.typeid.for(@_ZTIi)
  %match = icmp eq i32 %selector, %typeid
  br i1 %match, label %run_catch, label %try_next

run_catch:
  %thrown = call i8* @__cxa_begin_catch(%except)
  %tmp = bitcast i8* %thrown to i32*
  %i = load i32* %tmp
  call void @_Z18DoSomethingWithInti(i32 %i)
  ...
```

The selector value
(which condition matched)

LLVM constructs

```
   ...
} catch (int i) {
  DoSomethingWithInt(i);
} catch (class A a) {
  DoSomethingWithA(a);
}
```

```
lpad:
  %info = landingpad { i8*, i32 }
    personality @__gxx_personality_v0
    catch @_ZTIi
    catch @_ZTI1A
  %except = extractvalue { i8*, i32 } %info, 0
  %selector = extractvalue { i8*, i32 } %info, 1
  %typeid = call i32 @llvm.eh.typeid.for(@_ZTIi)
  %match = icmp eq i32 %selector, %typeid
  br i1 %match, label %run_catch, label %try_next

run_catch:
  %thrown = call i8* @__cxa_begin_catch(%except)
  %tmp = bitcast i8* %thrown to i32*
  %i = load i32* %tmp
  call void @_Z18DoSomethingWithInti(i32 %i)
  call void @__cxa_end_catch()
  br label %finished
```

The selector value for a match with type “int”

Check if the selector has this value

LLVM constructs

```
    ...
} catch (int i) {
  DoSomethingWithInt(i);
} catch (class A a) {
  DoSomethingWithA(a);
}
```

Did the exception match the “int” catch clause?

```
lpad:
  %info = landingpad { i8*, i32 }
    personality @__gxx_personality_v0
    catch @_ZTIi
    catch @_ZTI1A
  %except = extractvalue { i8*, i32 } %info, 0
  %selector = extractvalue { i8*, i32 } %info, 1
  %typeid = call i32 @llvm.eh.typeid.for(@_ZTIi)
  %match = icmp eq i32 %selector, %typeid
  br i1 %match, label %run_catch, label %try_next
```

Yes

No

```
run_catch:
  %thrown = call i8* @__cxa_begin_catch(%except)
  %tmp = bitcast i8* %thrown to i32*
  %i = load i32* %tmp
  call void @_Z18DoSomethingWithInti(i32 %i)
  call void @__cxa_end_catch()
  br label %finished

try_next:
  ...
```

LLVM constructs

```
   ...
} catch (int i) {
  DoSomethingWithInt(i);
} catch (class A a) {
  DoSomethingWithA(a);
}
```

```
lpad:
  %info = landingpad { i8*, i32 }
    personality @__gxx_personality_v0
    catch @_ZTIi
    catch @_ZTI1A
  %except = extractvalue { i8*, i32 } %info, 0
  %selector = extractvalue { i8*, i32 } %info, 1
  %typeid = call i32 @llvm.eh.typeid.for(@_ZTIi)
  %match = icmp eq i32 %selector, %typeid
  br i1 %match, label %run_catch, label %try_next
```

Yes

No

```
run_catch:
  %thrown = call i8* @__cxa_begin_catch(%except)
  %tmp = bitcast i8* %thrown to i32*
  %i = load i32* %tmp
  call void @_Z18DoSomethingWithInti(i32 %i)
  call void @__cxa_end_catch()
  br label %finished
```

Run the catch code

```
try_next:
  ...
```

Language specific library calls

LLVM constructs

```
    ...
} catch (int i) {
  DoSomethingWithInt(i);
} catch (class A a) {
  DoSomethingWithA(a);
}
```

```
lpad:
  %info = landingpad { i8*, i32 }
    personality @__gxx_personality_v0
    catch @_ZTIi
    catch @_ZTI1A
  %except = extractvalue { i8*, i32 } %info, 0
  %selector = extractvalue { i8*, i32 } %info, 1
  %typeid = call i32 @llvm.eh.typeid.for(@_ZTIi)
  %match = icmp eq i32 %selector, %typeid
  br i1 %match, label %run_catch, label %try_next
```

No

```
  ...
try_next:
  %typeid2 = call i32 @llvm.eh.typeid.for(@_ZTI1A)
  %match2 = icmp eq i32 %selector, %typeid2
  br i1 %match2, label %run_catch2, label %end
```

Yes

No

```
run_catch2:
  ...

end:
```

Did the exception match the “int” catch clause?

Did the exception match the “class A” catch clause?

LLVM constructs

```
    ...
} catch (int i) {
  DoSomethingWithInt(i);
} catch (class A a) {
  DoSomethingWithA(a);
}
```

```
lpad:
  %info = landingpad { i8*, i32 }
    personality @__gxx_personality_v0
    catch @_ZTIi
    catch @_ZTI1A

  ...

  br i1 %match2, label %run_catch2, label %end

  ...

end:
  resume { i8*, i32 } %info
```

The exception didn't match
any of the catch clauses

LLVM constructs

```
  ...
} catch (int i) {
  DoSomethingWithInt(i);
} catch (class A a) {
  DoSomethingWithA(a);
}
```

```
lpad:
  %info = landingpad { i8*, i32 }
    personality @__gxx_personality_v0
    catch @_ZTIi
    catch @_ZTI1A

  ...

  br i1 %match2, label %run_catch2, label %end

  ...

end:
  resume { i8*, i32 } %info
```

The exception didn't match any of the catch clauses

Continue unwinding the exception further up the call stack

Summary

invoke	Function call inside “try” block

Summary

invoke	Function call inside “try” block
landingpad	Lists catch clauses Returns exception info

Summary

invoke	Function call inside “try” block	
landingpad	Lists catch clauses Returns exception info	*New!*

Summary

invoke	Function call inside “try” block	
landingpad	Lists catch clauses Returns exception info	*New!*
llvm.eh.typeid.for	Map from typeinfo to selector value	

Summary

invoke	Function call inside “try” block
landingpad	Lists catch clauses Returns exception info *New!*
llvm.eh.typeid.for	Map from typeinfo to selector value
resume	Keep unwinding

Summary

invoke	Function call inside “try” block	
landingpad	Lists catch clauses Returns exception info	*New!*
llvm.eh.typeid.for	Map from typeinfo to selector value	
resume	Keep unwinding	*New!*

Summary

invoke	Function call inside “try” block	
landingpad	Lists catch clauses Returns exception info	*New!*
llvm.eh.typeid.for	Map from typeinfo to selector value	
resume	Keep unwinding	*New!*

Nested try

```
try {
  ...
  try {
    ...
    MayThrowSomething();
    ...
  } catch (int i) {
    DoSomethingWithInt(i);
  } catch (class A a) {
    DoSomethingWithA(a);
  }
  ...
} catch (class B b) {
  DoSomethingWithB(b);
} catch (...) {
  DoSomethingElse();
}
```

```
lpad:
  %info = landingpad { i8*, i32 }
   personality @__gxx_personality_v0
   catch @_ZTIi
   catch @_ZTI1A
   catch @_ZTI1B
   catch null
```

List all catch clauses that the exception may meet

Filters

```
int foo() throw () {
  bar();
  return 0;
}
```

```
  invoke void @_Z3barv()
          to label %cont unwind label %lpad

cont:
  ret i32 0

lpad:
  %info = landingpad { i8*, i32 }
          personality @__gxx_personality_v0
          filter [0 x i8*] zeroinitializer
  %except = extractvalue { i8*, i32 } %info, 0
  tail call void @__cxa_call_unexpected(%except)
  unreachable
```

Destructors

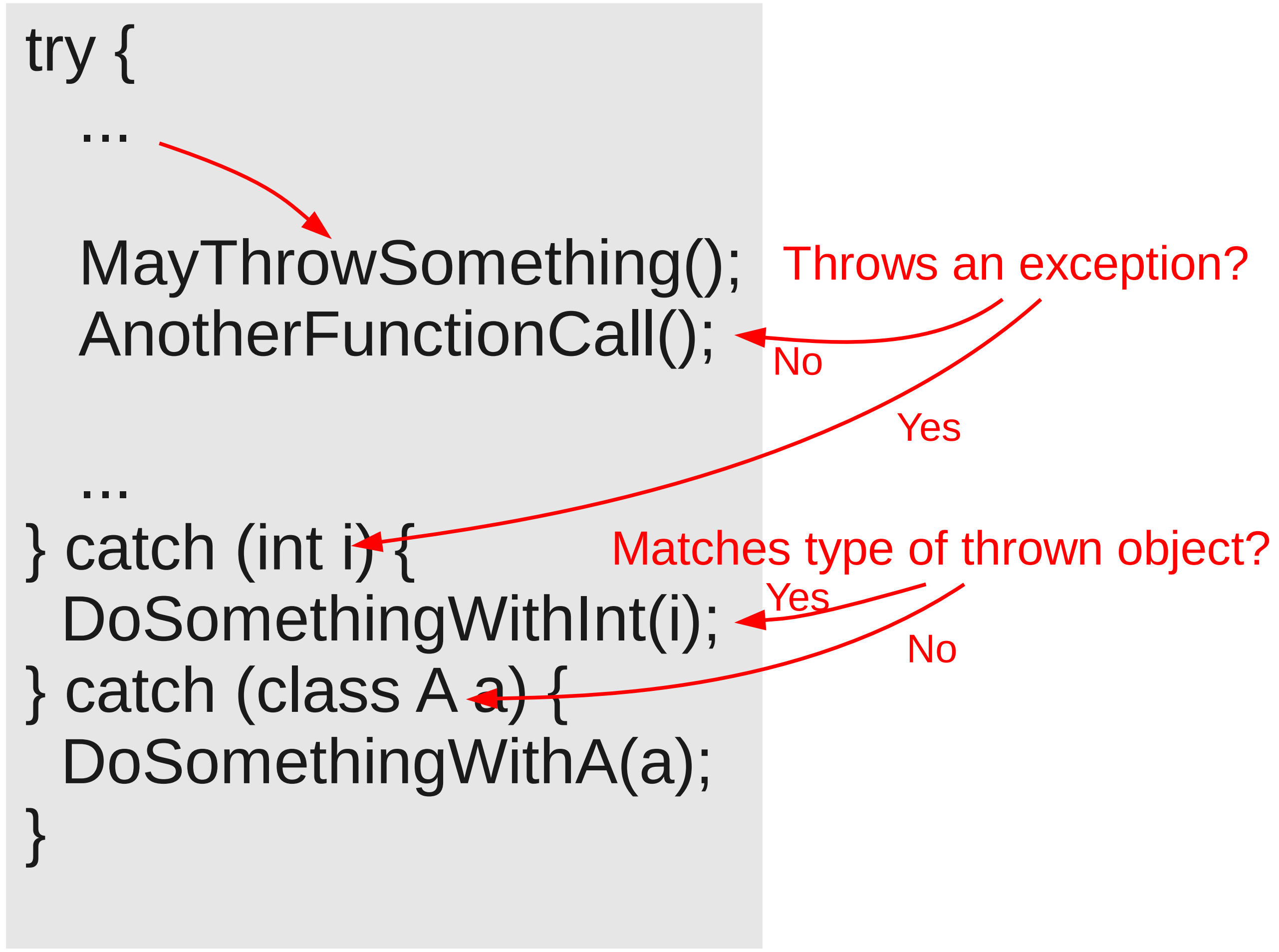
```c
void oof(void *);

void bar(void) {
  int x
    __attribute__((cleanup(oof)));
  foo();
  ...
}
```

```llvm
define void @bar() {
entry:
  %x = alloca i32
  invoke void @foo()
        to label %cont unwind label %lpad

cont:
  ...

lpad:
  %info = landingpad { i8*, i32 }
        personality @__gcc_personality_v0
        cleanup
  %var_ptr = bitcast i32* %x to i8*
  call void @oof(i8* %var_ptr)
  resume { i8*, i32 } %info
}
```

Run the cleanup {

Continue unwinding →

Control flow



```
try {
   ...

   MayThrowSomething();
   AnotherFunctionCall();

   ...
} catch (int i) {
  DoSomethingWithInt(i);
} catch (class A a) {
  DoSomethingWithA(a);
}
```

LLVM constructs

```
  ...
} catch (int i) {
  DoSomethingWithInt(i);
} catch (class A a) {
  DoSomethingWithA(a);
}
```

```
lpad:
  %info = landingpad { i8*, i32 }
    personality @__gxx_personality_v0
    catch @_ZTIi
    catch @_ZTI1A
```

Control reaches this point if:

- The exception matched one of catch clauses in the landingpad instruction. The selector indicates which clause matched.
- The exception didn't match any clauses but the unwinder chose to resume execution here anyway. The selector holds a value which does not correspond to any of the catch clauses.

LLVM constructs

```
  ...
} catch (int i) {
  DoSomethingWithInt(i);
} catch (class A a) {
  DoSomethingWithA(a);
}
```

```
lpad:
  %info = landingpad { i8*, i32 }
    personality @__gxx_personality_v0
    catch @_ZTIi
    catch @_ZTI1A
  %except = extractvalue { i8*, i32 } %info, 0
  %selector = extractvalue { i8*, i32 } %info, 1
  %typeid = call i32 @llvm.eh.typeid.for(@_ZTIi)
  %match = icmp eq i32 %selector, %typeid
  br i1 %match, label %run_catch, label %try_next

run_catch:
  %thrown = call i8* @__cxa_begin_catch(%except)
  %tmp = bitcast i8* %thrown to i32*
  %i = load i32* %tmp
  call void @_Z18DoSomethingWithInti(i32 %i)
  ...
```

LLVM constructs

```
   ...
} catch (int i) {
  DoSomethingWithInt(i);
} catch (class A a) {
  DoSomethingWithA(a);
}
```

Did the exception match the “int” catch clause?

```
lpad:
  %info = landingpad { i8*, i32 }
    personality @__gxx_personality_v0
    catch @_ZTIi
    catch @_ZTI1A
  %except = extractvalue { i8*, i32 } %info, 0
  %selector = extractvalue { i8*, i32 } %info, 1
  %typeid = call i32 @llvm.eh.typeid.for(@_ZTIi)
  %match = icmp eq i32 %selector, %typeid
  br i1 %match, label %run_catch, label %try_next
```

Yes

No

```
run_catch:
  %thrown = call i8* @__cxa_begin_catch(%except)
  %tmp = bitcast i8* %thrown to i32*
  %i = load i32* %tmp
  call void @_Z18DoSomethingWithInti(i32 %i)
  call void @__cxa_end_catch()
  br label %finished

try_next:
  ...
```