ILToC

Taking C# where the CLR cannot go

Keith Chapman
Mentor : Herman Venter

Motivation

- Provide the ability to write programs in C# and run it on a platform without the CLR
- A .NET VM to perform some experimentation
  - Try different GC schemes
  - Try different locking schemes
  - Try different ... anything
- Prime directive : Simplest implementation

What have we achieved?

- Translate all IL instructions used by C#
- No fancy optimizations
  - Always stick to the prime directive – Simplest implementation
- Limited functionality in BCL
  - Enough to run programs that do not depend heavily on the BCL
    - Supports collection classes
    - Does not support IO, Reflection

Some Results

Benchmark	CLR	ILToC with PGO	ILToC with PGO and no checks
Linpack	Mflops/s: 263.14 Time: 0.55 s	Mflops/s: 371.095 Time: 0.39 s	Mflops/s: 2067.53 Time: 0.07 s
Raytracer with data structure as a class	2.824 s	2.948 s	2.34 s
Raytracer with data structure as a struct	5.024 s		2.31 s

Some details

- Handling portability
- Structured Exception Handling
- Object layout
- Virtual Method Invocation
- Generics
- Threading
- Locking

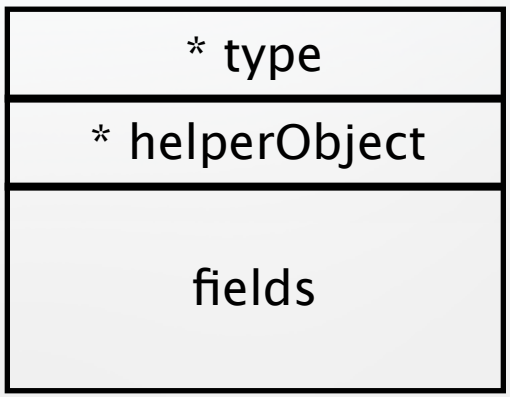
Handling portability

- Use C# whenever possible
- Use C standard library functions whenever possible
  - Special class CRuntime used to call C functions
- Abstract out platform specific functionality to a platform adaptation layer
  - Compare-and-swap
  - Threading primitives
  - Thread Local Storage (TLS)

Separate Compilation

- Name mangling scheme based on structure
- Prevent mangling by adding the [DoNotMangle] attribute on a method
- Type objects constructed during startup
- Each assembly constructs the nominal types that it provides
- Executable constructs all structural types

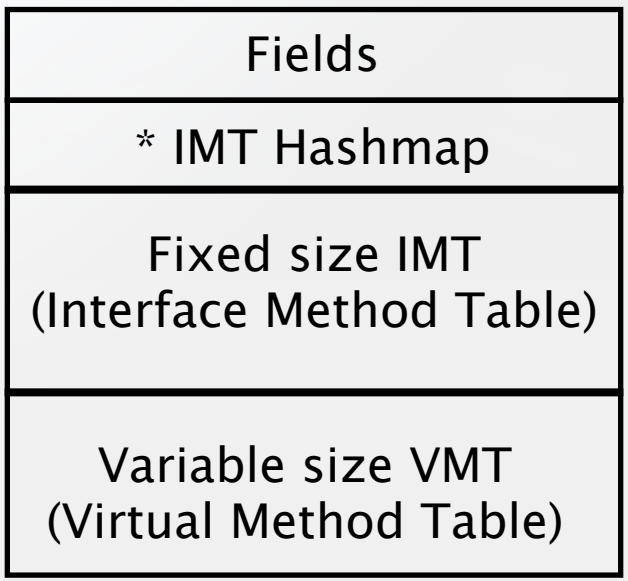
Structure of an Object




```
class SimpleClass {
  int i;
  bool flag;
}
```

```
struct SimpleClass {
  uintptr_t type;
  uintptr_t helperObject;
  int32_t i;
  uint8_t flag;
};
```

Structure of a type object



Creation of type objects

```
class SimpleClass {
  int i;
  bool flag;
}
```

```
SimpleClass_typeObject = (uintptr_t)calloc(1, sizeof(struct System_RuntimeType) +
sizeof(uintptr_t) * (5 + IMTSIZE + 1));
SimpleClass_typeObject += sizeof(struct System_Object);
InitializeRuntimeType(SimpleClass_typeObject, ...);
SetBaseClass(SimpleClass_typeObject, System_Object_typeObject);
```

Type Objects

- Represent System.Type
- Container for virtual method tables
- Holds base class information
- Holds information about interfaces implemented
- And more...

Object creation

```
new SimpleClass()
```

```
uintptr_t object_to_construct;
object_to_construct = (uintptr_t)calloc(1, sizeof(struct  SimpleClass));
object_to_construct += sizeof(struct System_Object);
SetType(object_to_construct, SimpleClass_typeObject);
SimpleClass__ctor(object_to_construct);
```

Structured Exception Handling

- In order to be platform independent
  - Did not use windows exception handling
  - Did not use setjmp/longjmp
- Designed our own simple scheme
  - Transforming method signatures
  - Using goto statements

Structured Exception Handling

- Transform method signatures

```
public int Method(int arg) {…}
```

```
uintptr_t Method(uintptr_t  _this, int32_t arg, uintptr_t _result)
```

- At a call site
  - If the return value of a method call is not null run exception handling code

```
{ ... exception = ...; // Allocate exception }
  throwOffset = 7; expSwitchVal1 = 4;
  goto lcatch_filter_8;
l0007_0008_4:
  if (canHandleExp > 0) {
    canHandleExp = 0;
    goto lrun_finallies_for_catch_8;
  }
  expSwitchVal1 = 5; goto l0010;
l0007_0010_5: return exception;
l0008: goto l000d;
l000d: expSwitchVal1 = 6; goto l0010;
l0013_0010_6: goto l0013;
l0010: goto lexpSwitch1;//Endfinally;
l0013: return 0;
lcatch_filter_8:
  TryCast(exception, System_ApplicationException_typeObject, (uintptr_t)&result);
  if ((void *)result != NULL){
    canHandleExp = 1;
  }
  goto lexpSwitch1;
lrun_finallies_for_catch_8: goto l0008;
lexpSwitch1:
  switch (expSwitchVal1) {
    case 4 : goto l0007_0008_4;
    case 5 : goto l0007_0010_5;
    case 6 : goto l0013_0010_6;
  }
```

```
public static void ThrowCatchSimple() {
    try {
      throw new System.Exception();
    } catch (System.ApplicationException) {

    } finally {

    }
  }
```

```
{ ... exception = ...; // Allocate exception }
  throwOffset = 7; expSwitchVal1 = 4;
  goto lcatch_filter_8;
l0007_0008_4:
  if (canHandleExp > 0) {
    canHandleExp = 0;
    goto lrun_finallies_for_catch_8;
  }
  expSwitchVal1 = 5; goto l0010;
l0007_0010_5: return exception;
l0008: goto l000d;
l000d: expSwitchVal1 = 6; goto l0010;
l0013_0010_6: goto l0013;
l0010: goto lexpSwitch1;//Endfinally;
l0013: return 0;
lcatch_filter_8:
  TryCast(exception, System_ApplicationException_typeObject, (uintptr_t)&result);
  if ((void *)result != NULL){
    canHandleExp = 1;
  }
  goto lexpSwitch1;
lrun_finallies_for_catch_8: goto l0008;
lexpSwitch1:
  switch (expSwitchVal1) {
    case 4 : goto l0007_0008_4;
    case 5 : goto l0007_0010_5;
    case 6 : goto l0013_0010_6;
  }
```

```
public static void ThrowCatchSimple() {
    try {
      throw new System.Exception();
    } catch (System.ApplicationException) {

    } finally {

    }
  }
```

```
{ ... exception = ...; // Allocate exception }
  throwOffset = 7; expSwitchVal1 = 4;
  goto lcatch_filter_8;
l0007_0008_4:
  if (canHandleExp > 0) {
    canHandleExp = 0;
    goto lrun_finallies_for_catch_8;
  }
  expSwitchVal1 = 5; goto l0010;
l0007_0010_5: return exception;
l0008: goto l000d;
l000d: expSwitchVal1 = 6; goto l0010;
l0013_0010_6: goto l0013;
l0010: goto lexpSwitch1;//Endfinally;
l0013: return 0;
lcatch_filter_8:
  TryCast(exception, System_ApplicationException_typeObject, (uintptr_t)&result);
  if ((void *)result != NULL){
    canHandleExp = 1;
  }
  goto lexpSwitch1;
lrun_finallies_for_catch_8: goto l0008;
lexpSwitch1:
  switch (expSwitchVal1) {
    case 4 : goto l0007_0008_4;
    case 5 : goto l0007_0010_5;
    case 6 : goto l0013_0010_6;
  }
```

```
public static void ThrowCatchSimple() {
    try {
      throw new System.Exception();
    } catch (System.ApplicationException) {

    } finally {

    }
  }
```

```
{ ... exception = ...; // Allocate exception }
  throwOffset = 7; expSwitchVal1 = 4;
  goto lcatch_filter_8;
l0007_0008_4:
  if (canHandleExp > 0) {
    canHandleExp = 0;
    goto lrun_finallies_for_catch_8;
  }
  expSwitchVal1 = 5; goto l0010;
l0007_0010_5: return exception;
l0008: goto l000d;
l000d: expSwitchVal1 = 6; goto l0010;
l0013_0010_6: goto l0013;
l0010: goto lexpSwitch1;//Endfinally;
l0013: return 0;
lcatch_filter_8:
  TryCast(exception, System_ApplicationException_typeObject, (uintptr_t)&result);
  if ((void *)result != NULL){
    canHandleExp = 1;
  }
  goto lexpSwitch1;
lrun_finallies_for_catch_8: goto l0008;
lexpSwitch1:
  switch (expSwitchVal1) {
    case 4 : goto l0007_0008_4;
    case 5 : goto l0007_0010_5;
    case 6 : goto l0013_0010_6;
  }
```

```
public static void ThrowCatchSimple() {
    try {
      throw new System.Exception();
    } catch (System.ApplicationException) {

    } finally {

    }
  }
```

```
{ ... exception = ...; // Allocate exception }
  throwOffset = 7; expSwitchVal1 = 4;
  goto lcatch_filter_8;
l0007_0008_4:
  if (canHandleExp > 0) {
    canHandleExp = 0;
    goto lrun_finallies_for_catch_8;
  }
  expSwitchVal1 = 5; goto l0010;
l0007_0010_5: return exception;
l0008: goto l000d;
l000d: expSwitchVal1 = 6; goto l0010;
l0013_0010_6: goto l0013;
l0010: goto lexpSwitch1;//Endfinally;
l0013: return 0;
lcatch_filter_8:
  TryCast(exception, System_ApplicationException_typeObject, (uintptr_t)&result);
  if ((void *)result != NULL){
    canHandleExp = 1;
  }
  goto lexpSwitch1;
lrun_finallies_for_catch_8: goto l0008;
lexpSwitch1:
  switch (expSwitchVal1) {
    case 4 : goto l0007_0008_4;
    case 5 : goto l0007_0010_5;
    case 6 : goto l0013_0010_6;
  }
```

```
public static void ThrowCatchSimple() {
    try {
      throw new System.Exception();
    } catch (System.ApplicationException) {

    } finally {

    }
  }
```

```
{ ... exception = ...; // Allocate exception }
  throwOffset = 7; expSwitchVal1 = 4;
  goto lcatch_filter_8;
l0007_0008_4:
  if (canHandleExp > 0) {
    canHandleExp = 0;
    goto lrun_finallies_for_catch_8;
  }
  expSwitchVal1 = 5; goto l0010;
l0007_0010_5: return exception;
l0008: goto l000d;
l000d: expSwitchVal1 = 6; goto l0010;
l0013_0010_6: goto l0013;
l0010: goto lexpSwitch1;//Endfinally;
l0013: return 0;
lcatch_filter_8:
  TryCast(exception, System_ApplicationException_typeObject, (uintptr_t)&result);
  if ((void *)result != NULL){
    canHandleExp = 1;
  }
  goto lexpSwitch1;
lrun_finallies_for_catch_8: goto l0008;
lexpSwitch1:
  switch (expSwitchVal1) {
    case 4 : goto l0007_0008_4;
    case 5 : goto l0007_0010_5;
    case 6 : goto l0013_0010_6;
  }
```

```
public static void ThrowCatchSimple() {
    try {
      throw new System.Exception();
    } catch (System.ApplicationException) {

    } finally {

    }
  }
```

```
{ ... exception = ...; // Allocate exception }
  throwOffset = 7; expSwitchVal1 = 4;
  goto lcatch_filter_8;
l0007_0008_4:
  if (canHandleExp > 0) {
    canHandleExp = 0;
    goto lrun_finallies_for_catch_8;
  }
  expSwitchVal1 = 5; goto l0010;
l0007_0010_5: return exception;
l0008: goto l000d;
l000d: expSwitchVal1 = 6; goto l0010;
l0013_0010_6: goto l0013;
l0010: goto lexpSwitch1;//Endfinally;
l0013: return 0;
lcatch_filter_8:
  TryCast(exception, System_ApplicationException_typeObject, (uintptr_t)&result);
  if ((void *)result != NULL){
    canHandleExp = 1;
  }
  goto lexpSwitch1;
lrun_finallies_for_catch_8: goto l0008;
lexpSwitch1:
  switch (expSwitchVal1) {
    case 4 : goto l0007_0008_4;
    case 5 : goto l0007_0010_5;
    case 6 : goto l0013_0010_6;
  }
```

```
public static void ThrowCatchSimple() {
    try {
      throw new System.Exception();
    } catch (System.ApplicationException) {

    } finally {

    }
}
```

```
{ ... exception = ...; // Allocate exception }
  throwOffset = 7; expSwitchVal1 = 4;
  goto lcatch_filter_8;
l0007_0008_4:
  if (canHandleExp > 0) {
    canHandleExp = 0;
    goto lrun_finallies_for_catch_8;
  }
  expSwitchVal1 = 5; goto l0010;
l0007_0010_5: return exception;
l0008: goto l000d;
l000d: expSwitchVal1 = 6; goto l0010;
l0013_0010_6: goto l0013;
l0010: goto lexpSwitch1;//Endfinally;
l0013: return 0;
lcatch_filter_8:
  TryCast(exception, System_ApplicationException_typeObject, (uintptr_t)&result);
  if ((void *)result != NULL){
    canHandleExp = 1;
  }
  goto lexpSwitch1;
lrun_finallies_for_catch_8: goto l0008;
lexpSwitch1:
  switch (expSwitchVal1) {
    case 4 : goto l0007_0008_4;
    case 5 : goto l0007_0010_5;
    case 6 : goto l0013_0010_6;
  }
```

```
public static void ThrowCatchSimple() {
    try {
      throw new System.Exception();
    } catch (System.ApplicationException) {

    } finally {

    }
  }
```

```
{ ... exception = ...; // Allocate exception }
  throwOffset = 7; expSwitchVal1 = 4;
  goto lcatch_filter_8;
l0007_0008_4:
  if (canHandleExp > 0) {
    canHandleExp = 0;
    goto lrun_finallies_for_catch_8;
  }
  expSwitchVal1 = 5; goto l0010;
l0007_0010_5: return exception;
l0008: goto l000d;
l000d: expSwitchVal1 = 6; goto l0010;
l0013_0010_6: goto l0013;
l0010: goto lexpSwitch1;//Endfinally;
l0013: return 0;
lcatch_filter_8:
  TryCast(exception, System_ApplicationException_typeObject, (uintptr_t)&result);
  if ((void *)result != NULL){
    canHandleExp = 1;
  }
  goto lexpSwitch1;
lrun_finallies_for_catch_8: goto l0008;
lexpSwitch1:
  switch (expSwitchVal1) {
    case 4 : goto l0007_0008_4;
    case 5 : goto l0007_0010_5;
    case 6 : goto l0013_0010_6;
  }
```

```
public static void ThrowCatchSimple() {
    try {
      throw new System.Exception();
    } catch (System.ApplicationException) {

    } finally {

    }
  }
```

```
{ ... exception = ...; // Allocate exception }
  throwOffset = 7; expSwitchVal1 = 4;
  goto lcatch_filter_8;
l0007_0008_4:
  if (canHandleExp > 0) {
    canHandleExp = 0;
    goto lrun_finallies_for_catch_8;
  }
  expSwitchVal1 = 5; goto l0010;
l0007_0010_5: return exception;
l0008: goto l000d;
l000d: expSwitchVal1 = 6; goto l0010;
l0013_0010_6: goto l0013;
l0010: goto lexpSwitch1;//Endfinally;
l0013: return 0;
lcatch_filter_8:
  TryCast(exception, System_ApplicationException_typeObject, (uintptr_t)&result);
  if ((void *)result != NULL){
    canHandleExp = 1;
  }
  goto lexpSwitch1;
lrun_finallies_for_catch_8: goto l0008;
lexpSwitch1:
  switch (expSwitchVal1) {
    case 4 : goto l0007_0008_4;
    case 5 : goto l0007_0010_5;
    case 6 : goto l0013_0010_6;
  }
```

```
public static void ThrowCatchSimple() {
    try {
      throw new System.Exception();
    } catch (System.ApplicationException) {

    } finally {

    }
  }
```

```
{ ... exception = ...; // Allocate exception }
  throwOffset = 7; expSwitchVal1 = 4;
  goto lcatch_filter_8;
l0007_0008_4:
  if (canHandleExp > 0) {
    canHandleExp = 0;
    goto lrun_finallies_for_catch_8;
  }
  expSwitchVal1 = 5; goto l0010;
l0007_0010_5: return exception;
l0008: goto l000d;
l000d: expSwitchVal1 = 6; goto l0010;
l0013_0010_6: goto l0013;
l0010: goto lexpSwitch1;//Endfinally;
l0013: return 0;
lcatch_filter_8:
  TryCast(exception, System_ApplicationException_typeObject, (uintptr_t)&result);
  if ((void *)result != NULL){
    canHandleExp = 1;
  }
  goto lexpSwitch1;
lrun_finallies_for_catch_8: goto l0008;
lexpSwitch1:
  switch (expSwitchVal1) {
    case 4 : goto l0007_0008_4;
    case 5 : goto l0007_0010_5;
    case 6 : goto l0013_0010_6;
  }
```

```
public static void ThrowCatchSimple() {
    try {
      throw new System.Exception();
    } catch (System.ApplicationException) {

    } finally {

    }
  }
```

```
{ ... exception = ...; // Allocate exception }
  throwOffset = 7; expSwitchVal1 = 4;
  goto lcatch_filter_8;
l0007_0008_4:
  if (canHandleExp > 0) {
    canHandleExp = 0;
    goto lrun_finallies_for_catch_8;
  }
  expSwitchVal1 = 5; goto l0010;
l0007_0010_5: return exception;
l0008: goto l000d;
l000d: expSwitchVal1 = 6; goto l0010;
l0013_0010_6: goto l0013;
l0010: goto lexpSwitch1;//Endfinally;
l0013: return 0;
lcatch_filter_8:
  TryCast(exception, System_ApplicationException_typeObject, (uintptr_t)&result);
  if ((void *)result != NULL){
    canHandleExp = 1;
  }
  goto lexpSwitch1;
lrun_finallies_for_catch_8: goto l0008;
lexpSwitch1:
  switch (expSwitchVal1) {
    case 4 : goto l0007_0008_4;
    case 5 : goto l0007_0010_5;
    case 6 : goto l0013_0010_6;
  }
```

```
public static void ThrowCatchSimple() {
    try {
      throw new System.Exception();
    } catch (System.ApplicationException) {

    } finally {

    }
  }
```

```
{ ... exception = ...; // Allocate exception }
  throwOffset = 7; expSwitchVal1 = 4;
  goto lcatch_filter_8;
l0007_0008_4:
  if (canHandleExp > 0) {
    canHandleExp = 0;
    goto lrun_finallies_for_catch_8;
  }
  expSwitchVal1 = 5; goto l0010;
l0007_0010_5: return exception;
l0008: goto l000d;
l000d: expSwitchVal1 = 6; goto l0010;
l0013_0010_6: goto l0013;
l0010: goto lexpSwitch1;//Endfinally;
l0013: return 0;
lcatch_filter_8:
  TryCast(exception, System_ApplicationException_typeObject, (uintptr_t)&result);
  if ((void *)result != NULL){
    canHandleExp = 1;
  }
  goto lexpSwitch1;
lrun_finallies_for_catch_8: goto l0008;
lexpSwitch1:
  switch (expSwitchVal1) {
    case 4 : goto l0007_0008_4;
    case 5 : goto l0007_0010_5;
    case 6 : goto l0013_0010_6;
  }
```

```
public static void ThrowCatchSimple() {
    try {
      throw new System.Exception();
    } catch (System.ApplicationException) {

    } finally {

    }
  }
```

```
{ ... exception = ...; // Allocate exception }
  throwOffset = 7; expSwitchVal1 = 4;
  goto lcatch_filter_8;
l0007_0008_4:
  if (canHandleExp > 0) {
    canHandleExp = 0;
    goto lrun_finallies_for_catch_8;
  }
  expSwitchVal1 = 5; goto l0010;
l0007_0010_5: return exception;
l0008: goto l000d;
l000d: expSwitchVal1 = 6; goto l0010;
l0013_0010_6: goto l0013;
l0010: goto lexpSwitch1;//Endfinally;
l0013: return 0;
lcatch_filter_8:
  TryCast(exception, System_ApplicationException_typeObject, (uintptr_t)&result);
  if ((void *)result != NULL){
    canHandleExp = 1;
  }
  goto lexpSwitch1;
lrun_finallies_for_catch_8: goto l0008;
lexpSwitch1:
  switch (expSwitchVal1) {
    case 4 : goto l0007_0008_4;
    case 5 : goto l0007_0010_5;
    case 6 : goto l0013_0010_6;
  }
```

```
public static void ThrowCatchSimple() {
    try {
      throw new System.Exception();
    } catch (System.ApplicationException) {

    } finally {

    }
  }
```

```
{ ... exception = ...; // Allocate exception }
  throwOffset = 7; expSwitchVal1 = 4;
  goto lcatch_filter_8;
l0007_0008_4:
  if (canHandleExp > 0) {
    canHandleExp = 0;
    goto lrun_finallies_for_catch_8;
  }
  expSwitchVal1 = 5; goto l0010;
l0007_0010_5: return exception;
l0008: goto l000d;
l000d: expSwitchVal1 = 6; goto l0010;
l0013_0010_6: goto l0013;
l0010: goto lexpSwitch1;//Endfinally;
l0013: return 0;
lcatch_filter_8:
  TryCast(exception, System_ApplicationException_typeObject, (uintptr_t)&result);
  if ((void *)result != NULL){
    canHandleExp = 1;
  }
  goto lexpSwitch1;
lrun_finallies_for_catch_8: goto l0008;
lexpSwitch1:
  switch (expSwitchVal1) {
    case 4 : goto l0007_0008_4;
    case 5 : goto l0007_0010_5;
    case 6 : goto l0013_0010_6;
  }
```

```
public static void ThrowCatchSimple() {
    try {
      throw new System.Exception();
    } catch (System.ApplicationException) {

    } finally {

    }
  }
```

```
{ ... exception = ...; // Allocate exception }
  throwOffset = 7; expSwitchVal1 = 4;
  goto lcatch_filter_8;
l0007_0008_4:
  if (canHandleExp > 0) {
    canHandleExp = 0;
    goto lrun_finallies_for_catch_8;
  }
  expSwitchVal1 = 5; goto l0010;
l0007_0010_5: return exception;
l0008: goto l000d;
l000d: expSwitchVal1 = 6; goto l0010;
l0013_0010_6: goto l0013;
l0010: goto lexpSwitch1;//Endfinally;
l0013: return 0;
lcatch_filter_8:
  TryCast(exception, System_ApplicationException_typeObject, (uintptr_t)&result);
  if ((void *)result != NULL){
    canHandleExp = 1;
  }
  goto lexpSwitch1;
lrun_finallies_for_catch_8: goto l0008;
lexpSwitch1:
  switch (expSwitchVal1) {
    case 4 : goto l0007_0008_4;
    case 5 : goto l0007_0010_5;
    case 6 : goto l0013_0010_6;
  }
```

```
public static void ThrowCatchSimple() {
    try {
      throw new System.Exception();
    } catch (System.ApplicationException) {

    } finally {

    }
  }
```

```
{ ... exception = ...; // Allocate exception }
  throwOffset = 7; expSwitchVal1 = 4;
  goto lcatch_filter_8;
l0007_0008_4:
  if (canHandleExp > 0) {
    canHandleExp = 0;
    goto lrun_finallies_for_catch_8;
  }
  expSwitchVal1 = 5; goto l0010;
l0007_0010_5: return exception;
l0008: goto l000d;
l000d: expSwitchVal1 = 6; goto l0010;
l0013_0010_6: goto l0013;
l0010: goto lexpSwitch1;//Endfinally;
l0013: return 0;
lcatch_filter_8:
  TryCast(exception, System_ApplicationException_typeObject, (uintptr_t)&result);
  if ((void *)result != NULL){
    canHandleExp = 1;
  }
  goto lexpSwitch1;
lrun_finallies_for_catch_8: goto l0008;
lexpSwitch1:
  switch (expSwitchVal1) {
    case 4 : goto l0007_0008_4;
    case 5 : goto l0007_0010_5;
    case 6 : goto l0013_0010_6;
  }
```

```
public static void ThrowCatchSimple() {
    try {
      throw new System.Exception();
    } catch (System.ApplicationException) {

    } finally {

    }
  }
```

```
{ ... exception = ...; // Allocate exception }
  throwOffset = 7; expSwitchVal1 = 4;
  goto lcatch_filter_8;
l0007_0008_4:
  if (canHandleExp > 0) {
    canHandleExp = 0;
    goto lrun_finallies_for_catch_8;
  }
  expSwitchVal1 = 5; goto l0010;
l0007_0010_5: return exception;
l0008: goto l000d;
l000d: expSwitchVal1 = 6; goto l0010;
l0013_0010_6: goto l0013;
l0010: goto lexpSwitch1;//Endfinally;
l0013: return 0;
lcatch_filter_8:
  TryCast(exception, System_ApplicationException_typeObject, (uintptr_t)&result);
  if ((void *)result != NULL){
    canHandleExp = 1;
  }
  goto lexpSwitch1;
lrun_finallies_for_catch_8: goto l0008;
lexpSwitch1:
  switch (expSwitchVal1) {
    case 4 : goto l0007_0008_4;
    case 5 : goto l0007_0010_5;
    case 6 : goto l0013_0010_6;
  }
```

```
public static void ThrowCatchSimple() {
    try {
      throw new System.Exception();
    } catch (System.ApplicationException) {

    } finally {

    }
  }
```

```
{ ... exception = ...; // Allocate exception }
  throwOffset = 7; expSwitchVal1 = 4;
  goto lcatch_filter_8;
l0007_0008_4:
  if (canHandleExp > 0) {
    canHandleExp = 0;
    goto lrun_finallies_for_catch_8;
  }
  expSwitchVal1 = 5; goto l0010;
l0007_0010_5: return exception;
l0008: goto l000d;
l000d: expSwitchVal1 = 6; goto l0010;
l0013_0010_6: goto l0013;
l0010: goto lexpSwitch1;//Endfinally;
l0013: return 0;
lcatch_filter_8:
  TryCast(exception, System_ApplicationException_typeObject, (uintptr_t)&result);
  if ((void *)result != NULL){
    canHandleExp = 1;
  }
  goto lexpSwitch1;
lrun_finallies_for_catch_8: goto l0008;
lexpSwitch1:
  switch (expSwitchVal1) {
    case 4 : goto l0007_0008_4;
    case 5 : goto l0007_0010_5;
    case 6 : goto l0013_0010_6;
  }
```

```
public static void ThrowCatchSimple() {
    try {
      throw new System.Exception();
    } catch (System.ApplicationException) {

    } finally {

    }
  }
```

```
{ ... exception = ...; // Allocate exception }
  throwOffset = 7; expSwitchVal1 = 4;
  goto lcatch_filter_8;
l0007_0008_4:
  if (canHandleExp > 0) {
    canHandleExp = 0;
    goto lrun_finallies_for_catch_8;
  }
  expSwitchVal1 = 5; goto l0010;
l0007_0010_5: return exception;
l0008: goto l000d;
l000d: expSwitchVal1 = 6; goto l0010;
l0013_0010_6: goto l0013;
l0010: goto lexpSwitch1;//Endfinally;
l0013: return 0;
lcatch_filter_8:
  TryCast(exception, System_ApplicationException_typeObject, (uintptr_t)&result);
  if ((void *)result != NULL){
    canHandleExp = 1;
  }
  goto lexpSwitch1;
lrun_finallies_for_catch_8: goto l0008;
lexpSwitch1:
  switch (expSwitchVal1) {
    case 4 : goto l0007_0008_4;
    case 5 : goto l0007_0010_5;
    case 6 : goto l0013_0010_6;
  }
```

```
public static void ThrowCatchSimple() {
    try {
      throw new System.Exception();
    } catch (System.ApplicationException) {

    } finally {

    }
  }
```

```
{ ... exception = ...; // Allocate exception }
  throwOffset = 7; expSwitchVal1 = 4;
  goto lcatch_filter_8;
l0007_0008_4:
  if (canHandleExp > 0) {
    canHandleExp = 0;
    goto lrun_finallies_for_catch_8;
  }
  expSwitchVal1 = 5; goto l0010;
l0007_0010_5: return exception;
l0008: goto l000d;
l000d: expSwitchVal1 = 6; goto l0010;
l0013_0010_6: goto l0013;
l0010: goto lexpSwitch1;//Endfinally;
l0013: return 0;
lcatch_filter_8:
  TryCast(exception, System_ApplicationException_typeObject, (uintptr_t)&result);
  if ((void *)result != NULL){
    canHandleExp = 1;
  }
  goto lexpSwitch1;
lrun_finallies_for_catch_8: goto l0008;
lexpSwitch1:
  switch (expSwitchVal1) {
    case 4 : goto l0007_0008_4;
    case 5 : goto l0007_0010_5;
    case 6 : goto l0013_0010_6;
  }
```

```
public static void ThrowCatchSimple() {
    try {
      throw new System.Exception();
    } catch (System.ApplicationException) {

    } finally {

    }
  }
```

Virtual Method Invocation

- When a class A is translated
  - Each virtual method is assigned a unique offset in its VMT
  - Methods inherited from a superclass retain the unique offset
  - If a method is overridden its VMT entry is simply overridden
- Results in a dense VMT

```
class BaseClass {
  public virtual void G() {}
}
class DerivedClass : BaseClass {
  public override void G() {}
}
```

```
baseAddress = (void **)(BaseClass_typeObject - sizeof(struct System_Object) +
sizeof(struct System_RuntimeType) + ((1 + IMTSIZE) * sizeof(uintptr_t)));
*(baseAddress++) = (void *)&System_Object_Finalize;
*(baseAddress++) = (void *)&System_Object_Equals_System_Object;
*(baseAddress++) = (void *)&System_Object_GetHashCode;
*(baseAddress++) = (void *)&System_Object_ComputeHashCode;
*(baseAddress++) = (void *)&System_Object_ToString;
*(baseAddress++) = (void *)&BaseClass_G;

baseAddress = (void **)(DerivedClass_typeObject - sizeof(struct System_Object) +
sizeof(struct System_RuntimeType) + ((1 + IMTSIZE) * sizeof(uintptr_t)));
*(baseAddress++) = (void *)&System_Object_Finalize;
*(baseAddress++) = (void *)&System_Object_Equals_System_Object;
*(baseAddress++) = (void *)&System_Object_GetHashCode;
*(baseAddress++) = (void *)&System_Object_ComputeHashCode;
*(baseAddress++) = (void *)&System_Object_ToString;
*(baseAddress++) = (void *)&DerivedClass_G;
```

Interface Method Invocation

- Trickier than virtual method invocation
  - Two classes implementing an interface method cannot guarantee the same VMT offset
- Need a mechanism that is comparable in performance to VMT's
- Implementation based on "Efficient implementation of Java interfaces: Invokeinterface considered harmless" – OOPSLA 2001

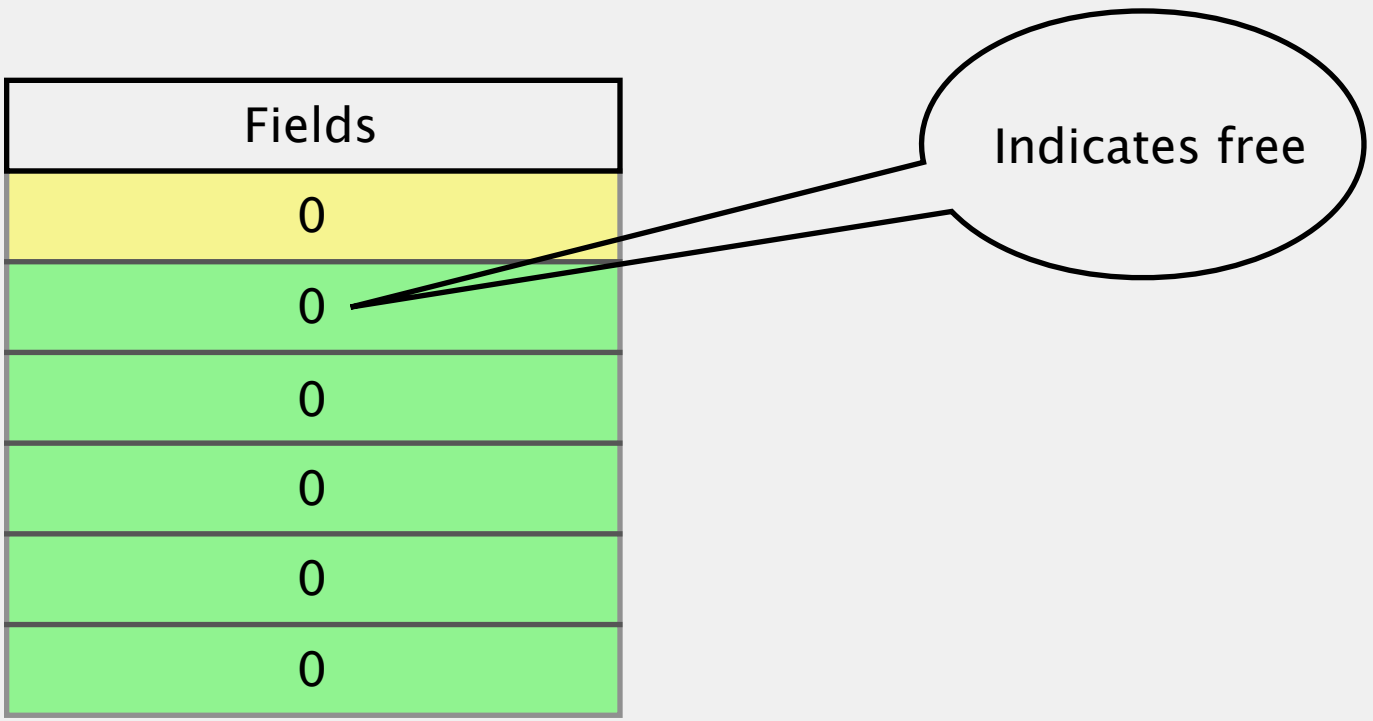
Interface Method Invocation

- Assign a unique ID to each interface method sequentially
- Zero-Initialize all IMT slots
- Each ID is hashed into an IMT offset
- If there is no collision then the IMT slot holds a function pointer to the method
- If there is a collision the function pointer is entered into a Hashtable

```
interface Interface {
  void F();
}
interface Interface2 {
  void G();
}
class ImplementingClass : Interface,
Interface2 {
  public void F() { }
  public void G() { }
}
```

Fields
0
0
0
0
0
0

```
interface Interface {
  void F();
}
interface Interface2 {
  void G();
}
class ImplementingClass : Interface,
Interface2 {
  public void F() { }
  public void G() { }
}
```



```
interface Interface {
  void F();
}
interface Interface2 {
  void G();
}
class ImplementingClass : Interface,
Interface2 {
  public void F() { }
  public void G() { }
}
```

```
Interface_F_id = interfaceMethodIDCounter;
interfaceMethodIDCounter++;
// Assume Interface_F_id = 16
```

Fields
0
0
0
0
0
0

IMTSIZE = 5
Interface_F_id = 16

```
interface Interface {
  void F();
}
interface Interface2 {
  void G();
}
class ImplementingClass : Interface,
Interface2 {
  public void F() { }
  public void G() { }
}
```

Fields
0
0
0
0
0
0

IMTSIZE = 5
Interface_F_id = 16

```
interface Interface {
  void F();
}
interface Interface2 {
  void G();
}
class ImplementingClass : Interface,
Interface2 {
  public void F() { }
  public void G() { }
}
```

```
baseAddress = ...;
IMTOffset = Interface_F_id % IMTSIZE;
// IMTSIZE = 5
// IMTOffset = 1
```

Fields
0
0
0
0
0
0

IMTSIZE = 5
Interface_F_id = 16
IMTOffset = 1

```
interface Interface {
  void F();
}
interface Interface2 {
  void G();
}
class ImplementingClass : Interface,
Interface2 {
  public void F() { }
  public void G() { }
}
```

Fields
0
0
0
0
0
0

IMTSIZE = 5
Interface_F_id = 16
IMTOffset = 1

```
interface Interface {
  void F();
}
interface Interface2 {
  void G();
}
class ImplementingClass : Interface,
Interface2 {
  public void F() { }
  public void G() { }
}
```

Fields
0
0
&ImplementingClass_F
0
0
0

IMTSIZE = 5
Interface_F_id = 16
IMTOffset = 1

Interface2_G_id = interfaceMethodIDCounter;
interfaceMethodIDCounter++;
// Assume Interface2_G_id = 21

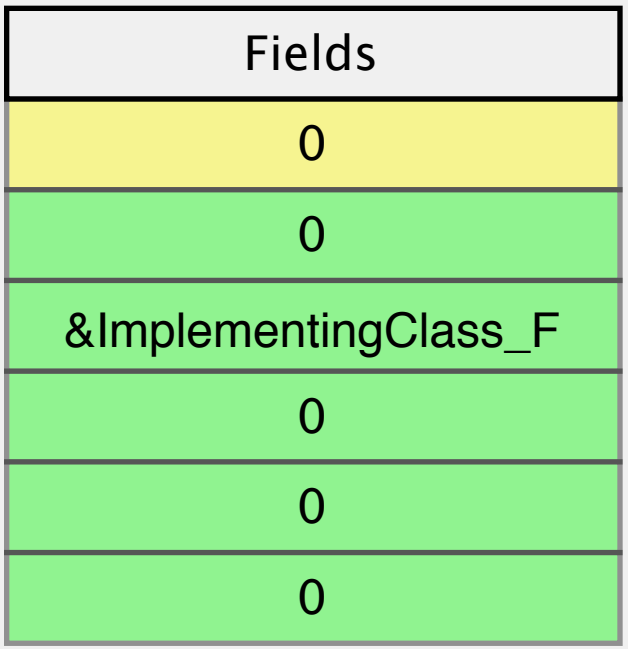
```
interface Interface {
  void F();
}
interface Interface2 {
  void G();
}
class ImplementingClass : Interface,
Interface2 {
  public void F() { }
  public void G() { }
}
```

Fields
0
0
&ImplementingClass_F
0
0
0

IMTSIZE = 5
Interface2_G_id = 21
IMTOffset = 1

&ImplementingClass_G

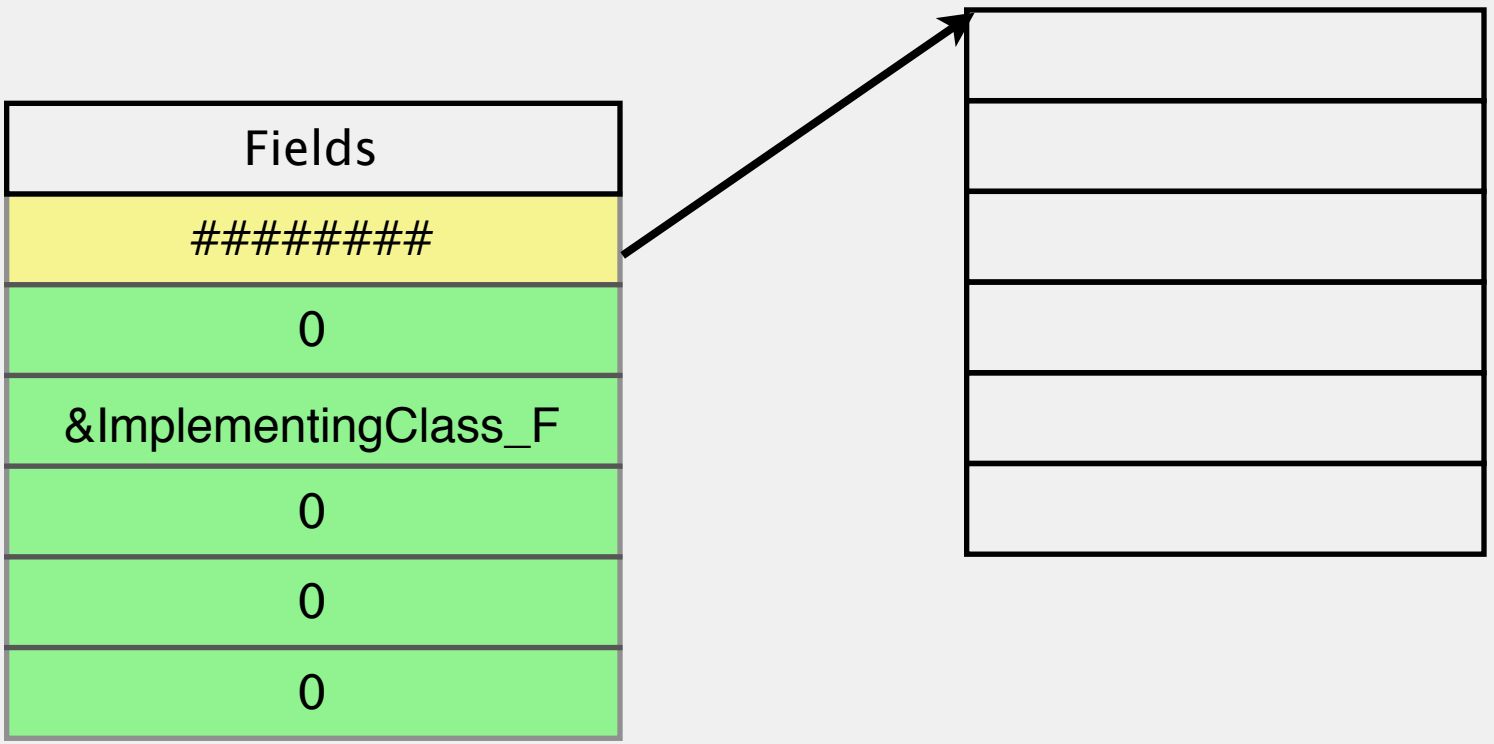
```
interface Interface {
  void F();
}
interface Interface2 {
  void G();
}
class ImplementingClass : Interface,
Interface2 {
  public void F() { }
  public void G() { }
}
```



IMTSIZE = 5
Interface2_G_id = 21
IMTOffset = 1

&ImplementingClass_G

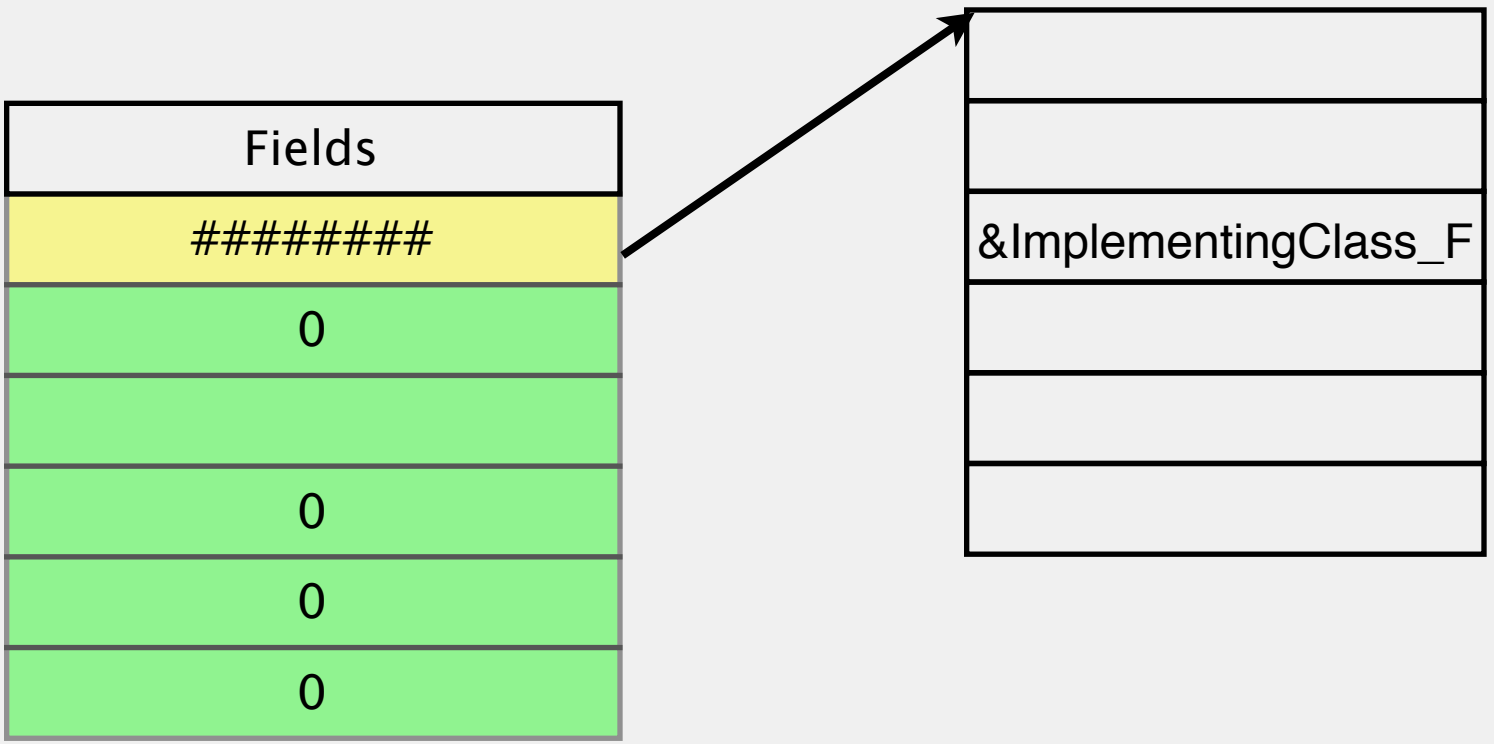
```
interface Interface {
  void F();
}
interface Interface2 {
  void G();
}
class ImplementingClass : Interface,
Interface2 {
  public void F() { }
  public void G() { }
}
```



IMTSIZE = 5
Interface2_G_id = 21
IMTOffset = 1

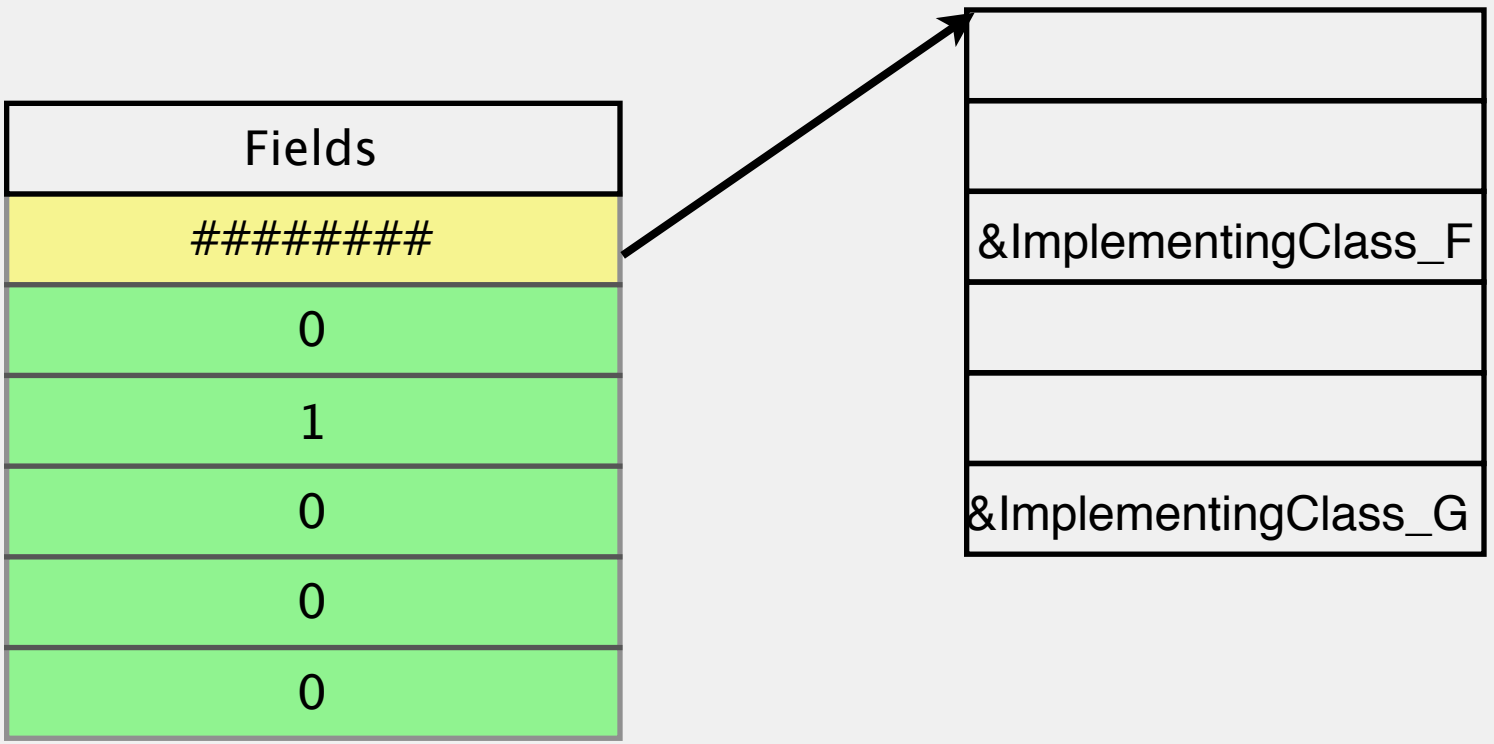
&ImplementingClass_G

```
interface Interface {
  void F();
}
interface Interface2 {
  void G();
}
class ImplementingClass : Interface,
Interface2 {
  public void F() { }
  public void G() { }
}
```



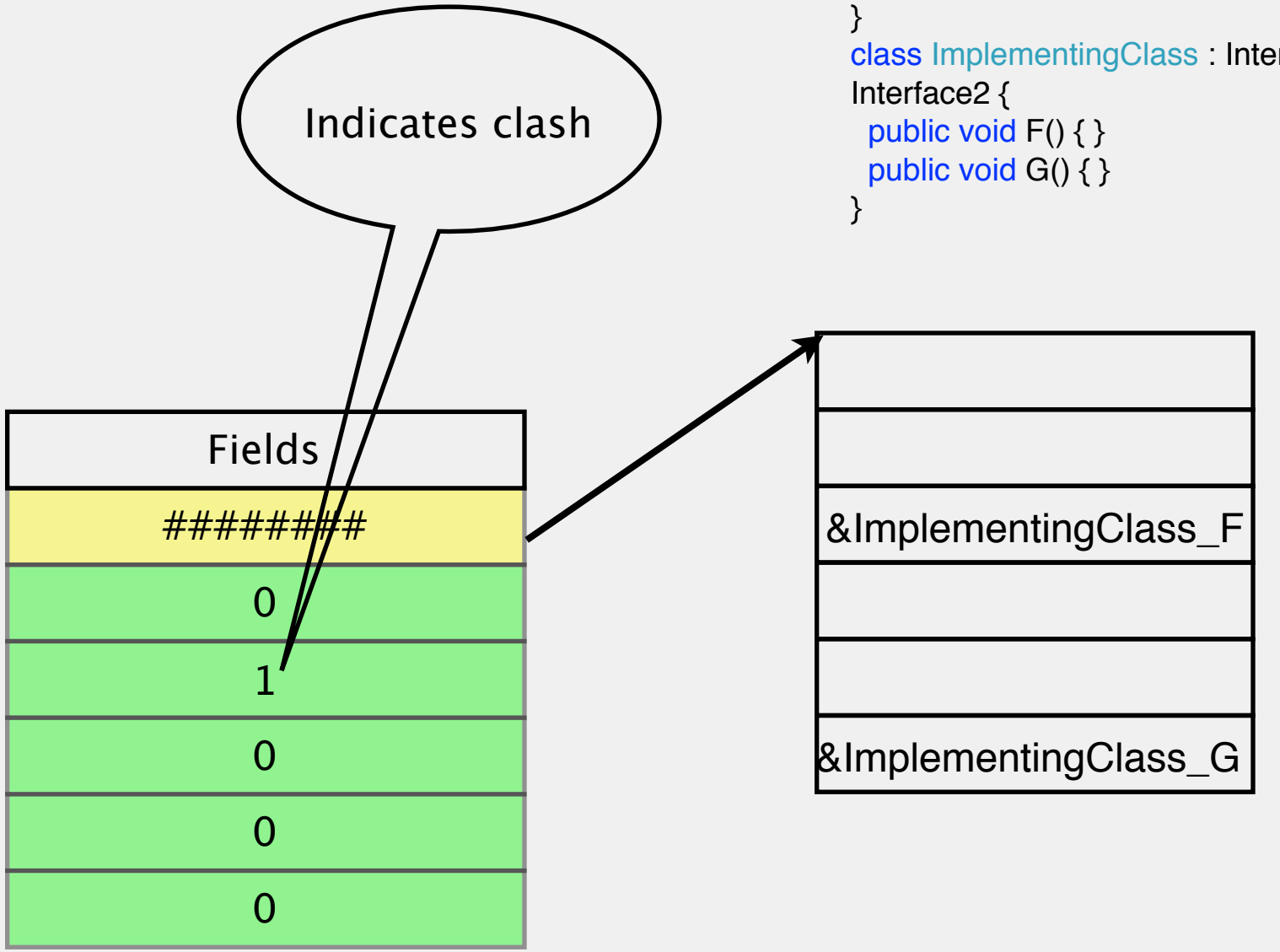
IMTSIZE = 5
Interface2_G_id = 21
IMTOffset = 1

```
interface Interface {
  void F();
}
interface Interface2 {
  void G();
}
class ImplementingClass : Interface,
Interface2 {
  public void F() { }
  public void G() { }
}
```



IMTSIZE = 5
Interface2_G_id = 21
IMTOffset = 1

```
interface Interface {
  void F();
}
interface Interface2 {
  void G();
}
class ImplementingClass : Interface,
Interface2 {
  public void F() { }
  public void G() { }
}
```



Generics

- Specialize everything
  - Not the best but it's the simplest solution
  - Results in some code bloat
- Need to identify type objects at translation time
  - When an executable is translated traverse all modules and create type objects for all generic instances

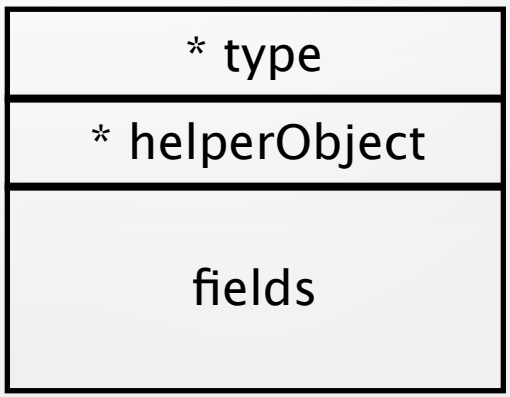
```
public class MyClass {
}

public class MyClass2 : MyClass {
}

public class Stack<T> {
}

public class NewGenericInstance {
  public static int Main() {
    Stack<MyClass> s = new Stack<MyClass>();
    Stack<MyClass2> ss = new Stack<MyClass2>();
    return 0;
  }
}
```

```
uintptr_t Stack_MyClass2__typeObject;
uintptr_t Stack_MyClass__typeObject;

AllocateForGenericArguments(Stack_MyClass2__typeObject, 1);
SetGenericArgument(Stack_MyClass2__typeObject, MyClass2_typeObject, 0);

AllocateForGenericArguments(Stack_MyClass__typeObject, 1);
SetGenericArgument(Stack_MyClass__typeObject, MyClass_typeObject, 0);
```

Threading

- Create an OS thread for each thread object
- Wrap the startup delegate with a generic delegate
  - Store the handle of the current thread to TLS
  - Clean up after termination

Recall : Structure of an Object



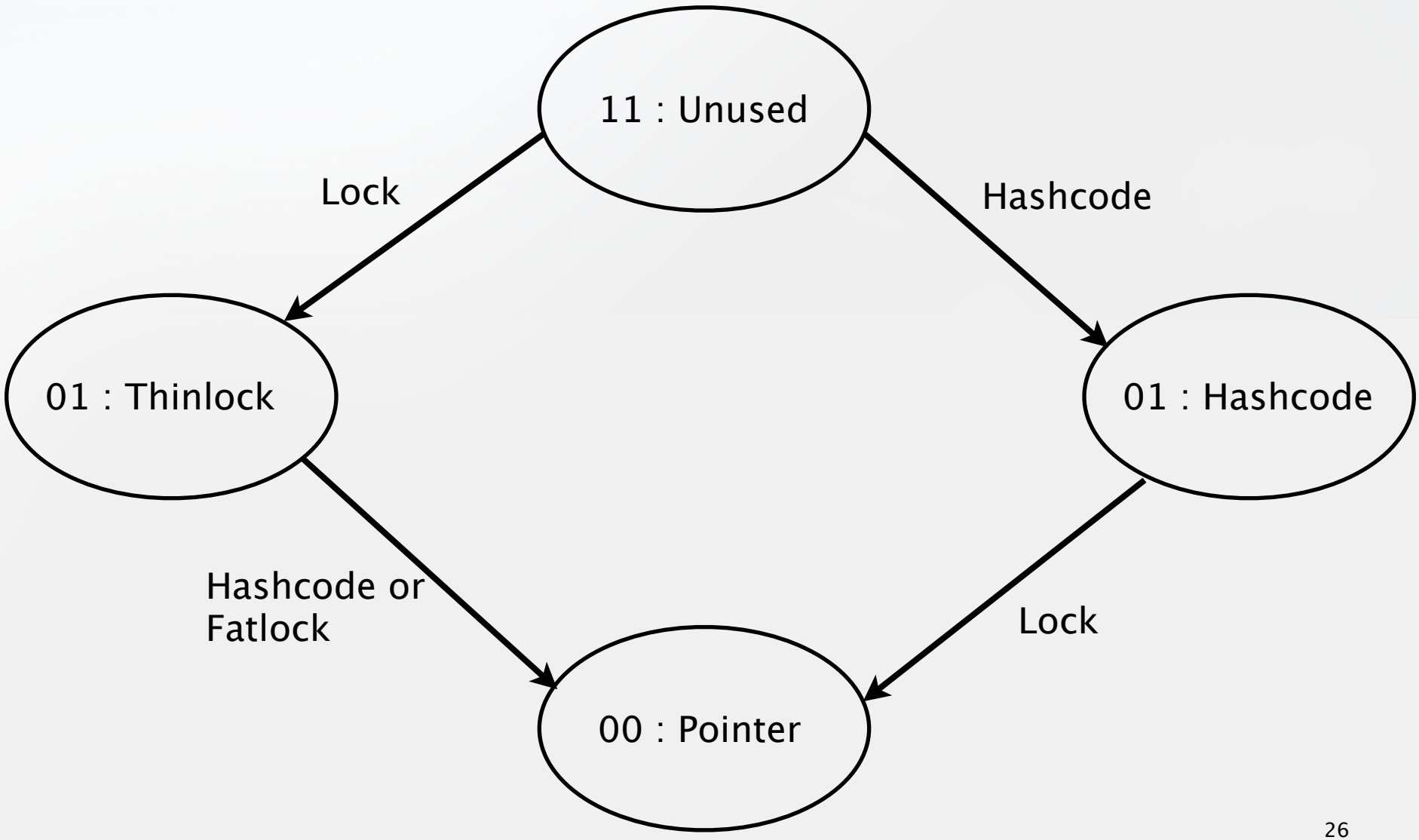

```
class SimpleClass {
  int i;
  bool flag;
}
```

```
struct _3762501631_SimpleClass {
  uintptr_t _2457770256_type;
  uintptr_t _3181468816_helperObject;
  int32_t _4630790_i;
  uint8_t _1288642005_flag;
};
```

HelperObject

- Allocated on demand
- The helper object pointer is used to store multiple pieces of information
- Use high two bits of the helper object pointer to indicate mode
  - 11 : Field is unused
  - 00 : Pointer to a helper object
  - 01 : Hashcode is stored in helperObject address
  - 10 : ThinLock is stored in helperObject address

HelperObject – Modes



Thinlocks

- Lightweight locking scheme to cater to the 2 most common cases
  - Locking an unlocked Object
  - Shallow nested locking
- Structure the lock word such that,
  - n bits are used for the thread id (16)
  - m bits are used for the nested lock count (8)

Transition to a fatlock

- If thread B tries to lock an object while thread A holds a lock
- If the nested lock count overflows
- If GetHashCode is called while the object is locked
- If an object is locked after GetHashCode has been called on it (The helper object pointer contains a hashcode)

Fatlocks

- The helper object maintains a queue for waiting threads
  - Each thread has a next filed (which points to another thread)
  - Helper object holds pointers to the head and the tail
- When a thread unlocks an object, get the head of the queue and wake it up

Future Work

- Use type erasure when possible for generics
  - Needed to handle some corner cases
- Implement missing parts of the BCL
  - Reflection
  - Dynamic code loading
- Memory management
- Platform adaptation layers

Questions