VBA Internals and Technology

Ilan Caron
Software Design Engineer
VBA Development
Microsoft

Topics

Design Goals
Implementation
VBA and OLE2
VBA Overview and Architecture
VBA Compiler
VBA Engine

Design Goals

VB language compatibility
Efficiency: compile-time and run-time
Application hostability
High Productivity
Development Environment
Portability
Localizable
Standard
Cross-application Programmability

Implementation

- C++ for OO features and portability
- Assembler as appropriate: engine parts
  - VBA and VB3 performance comparable.
- Modular Engine
- Complete redesign/“Written from scratch”
- Scalable technology
  - Seamless OLE2 interaction

OLE Automation

- Ole Automation (OA) is a set of standard interfaces supporting programmable objects.
- IDispatch: supports run-time binding and run-time invocation.
  - IDispatch::GetIDsOfNames: run-time binding
  - IDispatch::Invoke: run-time invocation
  - needed for untyped variables: Object/Variant
- ITypeLib: provides compile-time descriptions of set of programmable objects
  - ITypeLib contains set of ITypeInfos
  - ITypeInfo describes object: methods, properties, variables.
  - Supports compile-time binding

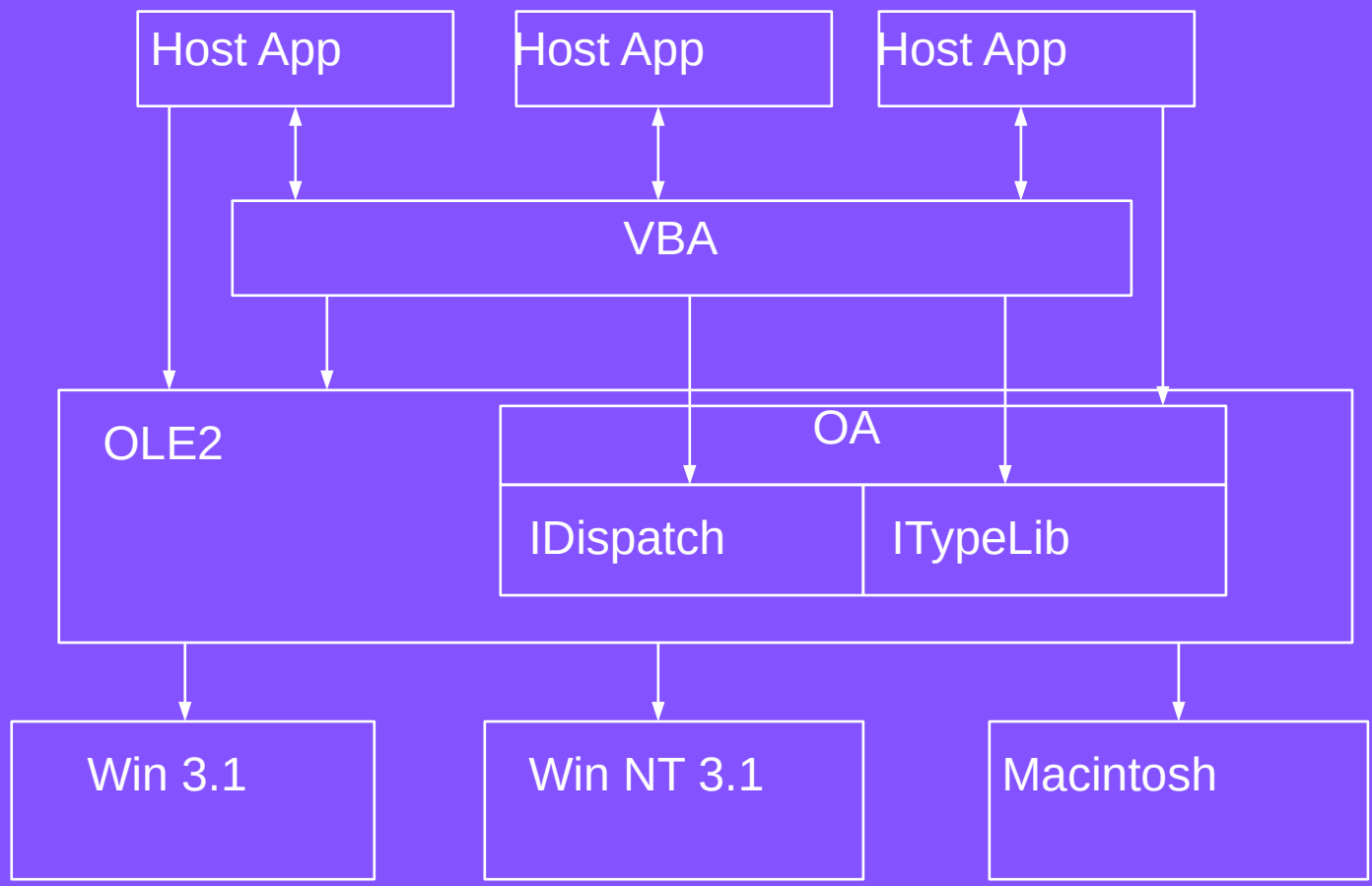
VBA and OLE2

- VBA uses Ole Automation
  - uses ITypeLib/ITypeInfo as well unlike VB3.
- VBA can control any application that exposes programmable objects via IDispatch
- VBA is more efficient if application provides a type library as well for its programmable objects
  - Also provides compile-time type checking
  - And object browsing
- VBA uses OA String, Variant, Array package to implement run-time
  - Exceptions: numerical conversions

VBA and OLE2

- VBA uses OLE2 to implement cross-process object invocation (function call/property access)
  - Extensible to cross-machine/net calls!
- VBA uses OLE2 standard interfaces for memory management and file storage
  - Host-customizable: VBA program can be stored anywhere, e.g. in a spreadsheet.
- Big Market for OA components
  - Language independent
  - THE Microsoft programmability solution

Programmability Architecture



VBA and Host Integration

- VBA works and appears identically in all host applications.
- VBA interacts with host-supplied objects in similar ways across multiple applications.
  - Since host applications implement similar Object Models (by convention)

VBA Components

Incremental Compiler

- syntax and semantic analysis
- code generation
- inter-module dependency analysis

User Interface

- code editor
- object browser
- debugger
- immediate window
- watch expressions

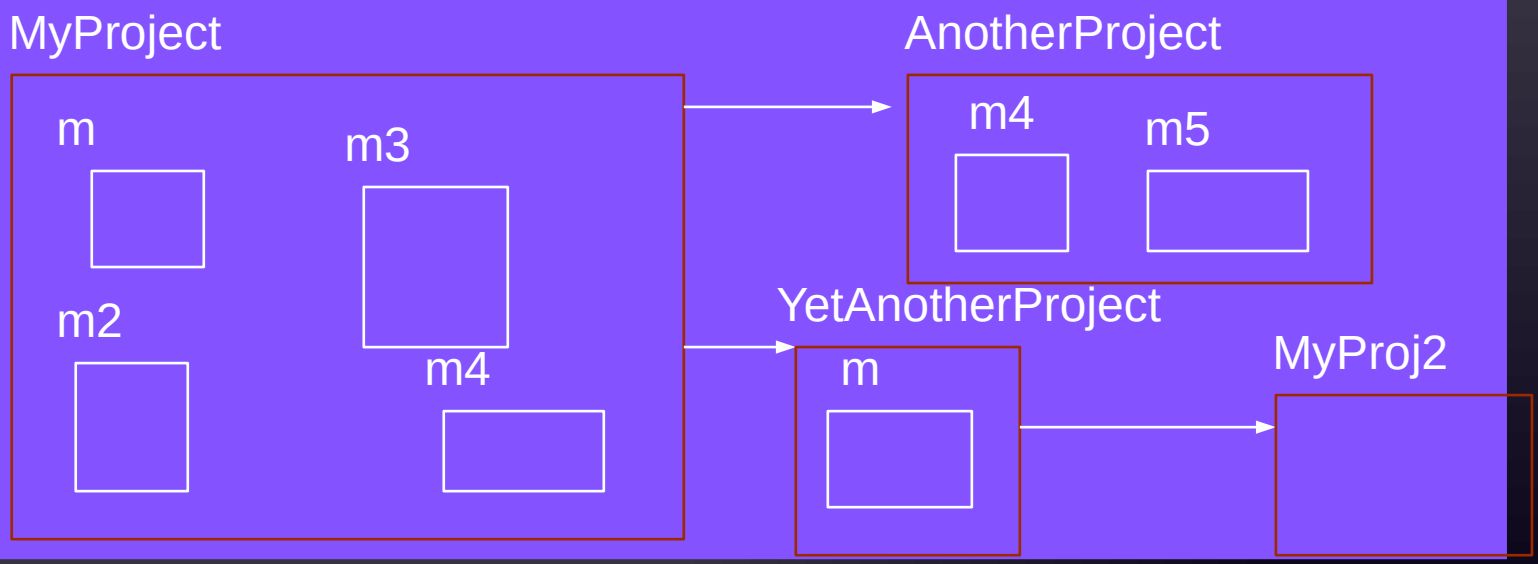
Engine

Run-time

- implemented as a library

Project Model

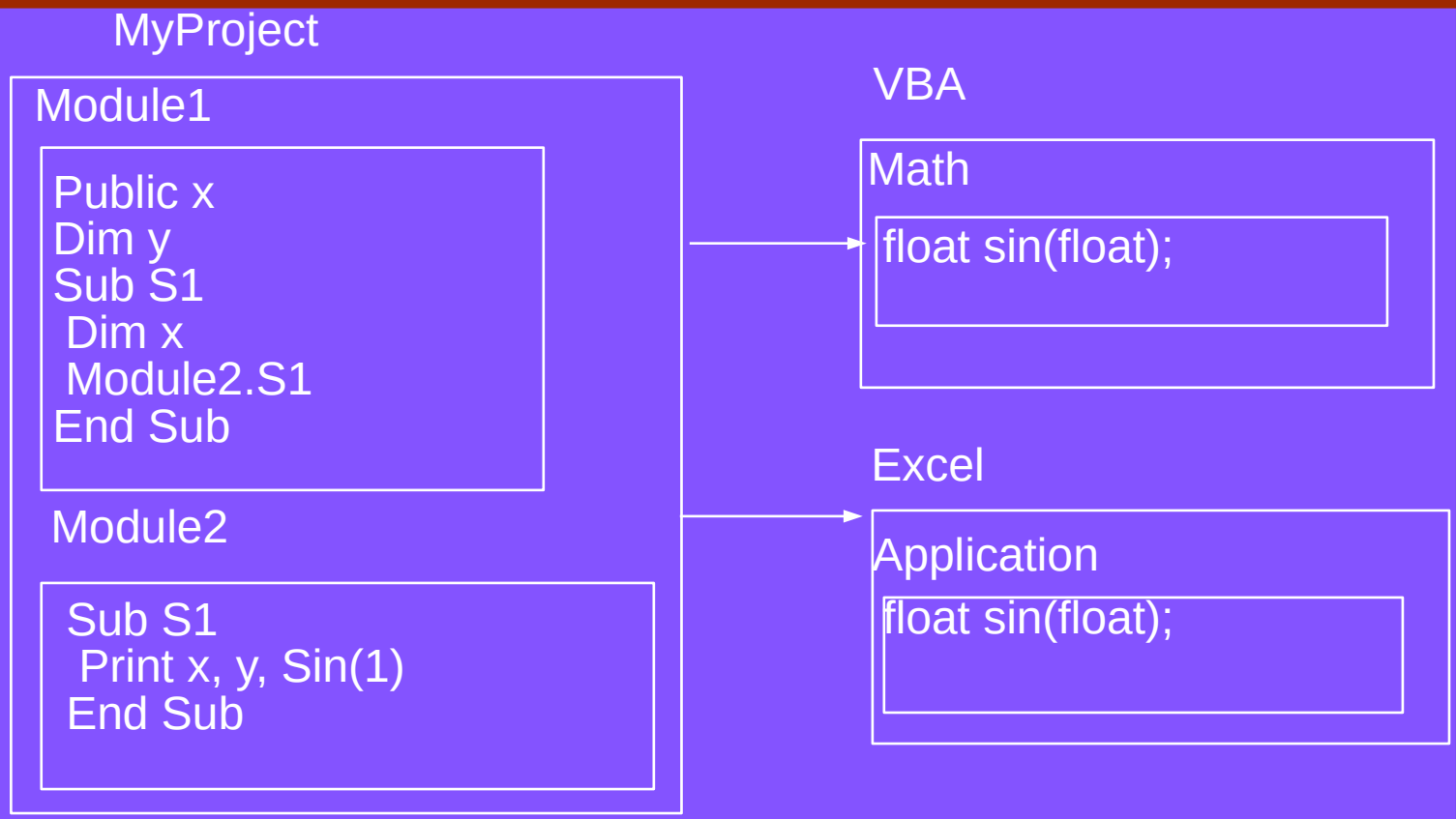
- **Project is a collection of Modules**
  - **Project is an implementation of ITypeLib**
- **Module is an implementation of ITypeInfo**
- **Project has list of referenced Projects**
  - **cycles disallowed**



Name Scoping

- Structured
  - Procedure-level
  - Module-level: public and private
- Explicit qualification
  - Call MyProj2.M2.S2
- Referenced projects
  - Names in directly referenced projects available without qualification.
  - Names in indirectly referenced projects available require qualification
- Private modules
- Application objects
  - Set myWorksheet = Worksheets.Add

Namespace Example



Beware of Implicit Variables
Look, no Declares!

Binding

- “The Earlier the Better”
- Three kinds of binding
  - Early (static) binding
    - VBA to VBA binding
    - VBA to DLL binding
  - Early (late id) binding
    - VBA to ITypeLib object method/property
  - Late (name) binding
    - VBA to OA object method/property
- Run-time binding with OA GetIDsOfNames/Invoke

Binding Example

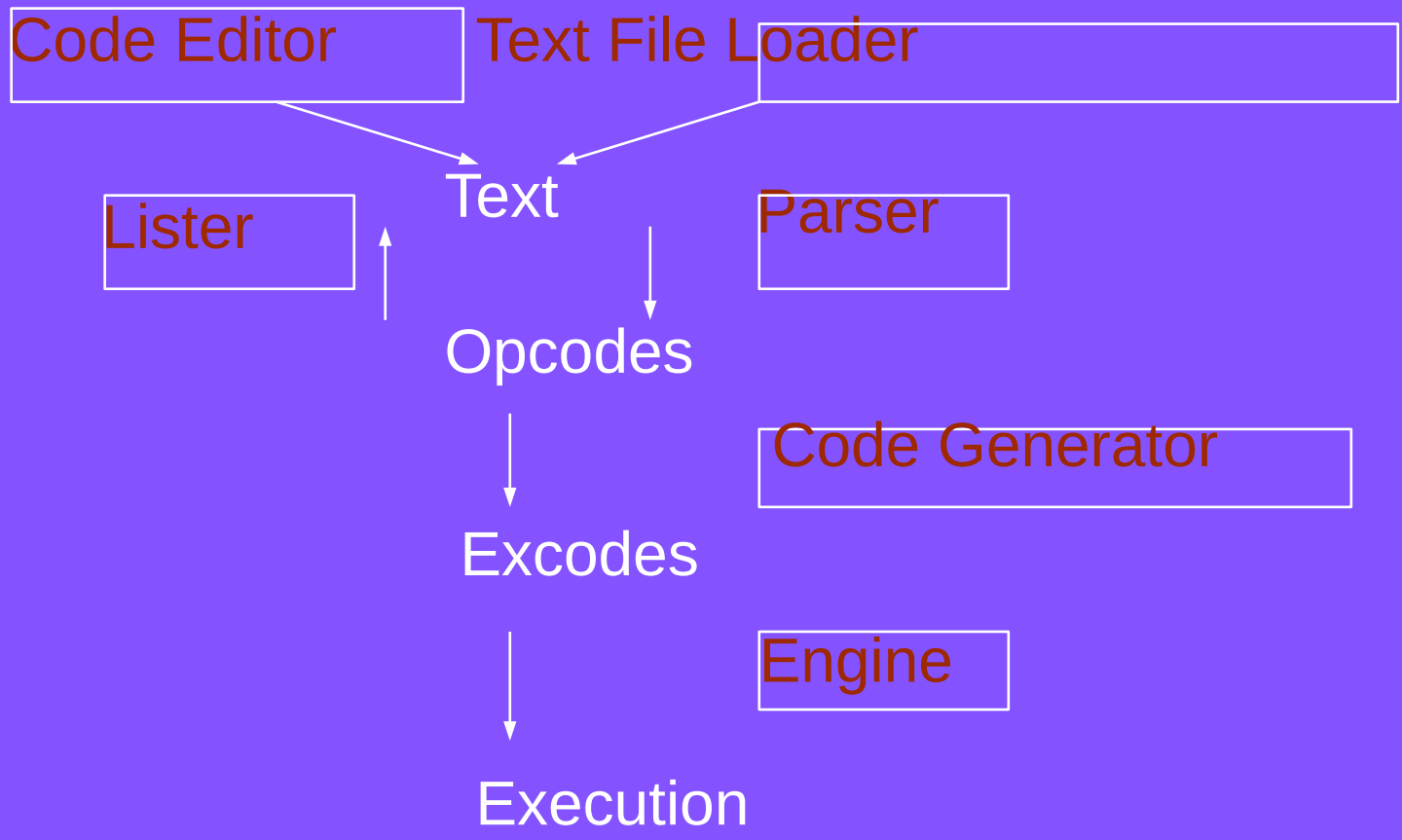
```
Dim w2 as Worksheet
Dim c as Object
Dim w as Object
Sub S2()
 S1
End Sub
Sub S1
 Set w = Workbooks.Add
 Application.Visible = true
 Set w2 = Worksheets(“sheet1”)
 Set c = ActiveCell
 c = w2.Name
 Cells(1, 2).Activate
 Set c2 = ActiveCell
 c2.Value= w.Name
End Sub
```

Early Static
Early (late id)
Late (name)

Compiler Overview

- **Dual p-code representation**
  - opcodes: produced by parser
  - excodes: produced by code-generator
- **Opcodes: postfix representation of source**
  - retains sufficient information to recreate original source text
- **Excodes: small and optimized**
  - all references are bound
  - unnecessary “decoration” is discarded: e.g. comments
  - code generator performs both semantic analysis and codegen
- **Both opcodes and/or excodes can be saved and reloaded.**

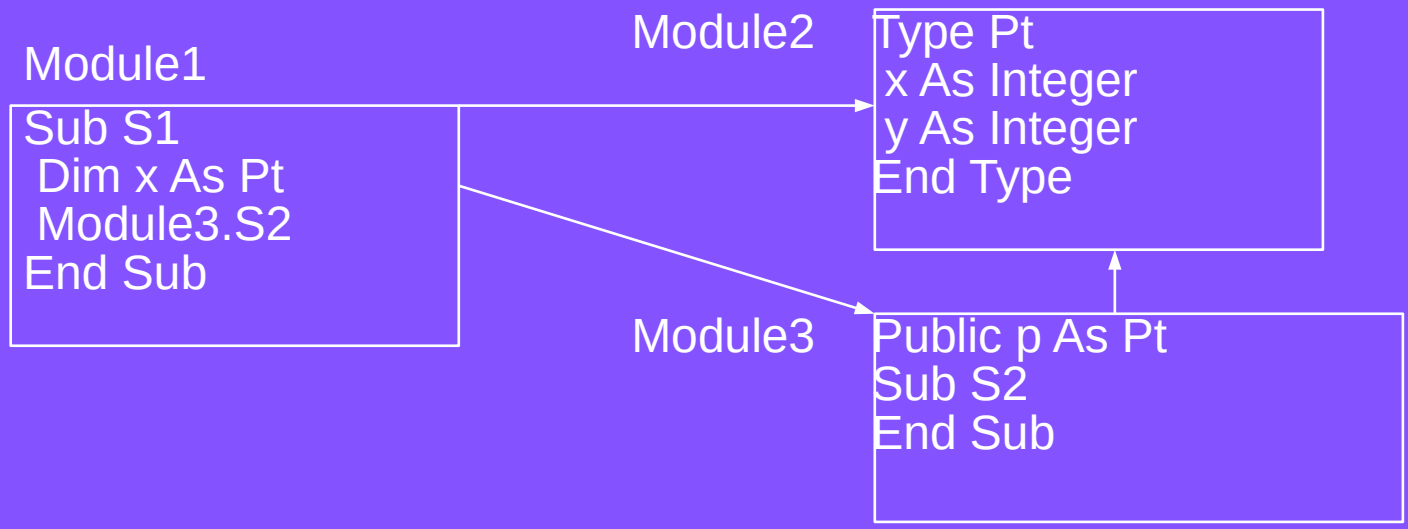
Compiler Flow



Compilation States

- VBA implements “just-in-time” (demand) compilation and decompilation.
  - Optimized for high productivity/rapid development turnaround
- Fundamental compilation unit is module.
- Multiple-states:
  - undeclared (parsed)
  - declared (laid out: shape known)
  - compiled (generated code)
  - runnable (functions can be called)
- Inter-module dependency tracking mechanism determines when and which modules are compiled and decompiled.

Dependency Example



Edits to Module1 never affect state of the other modules.

Calling Module3.S2 only requires Module2 and Module3 to be compiled

Module1 need not be compiled.

However, calling Module1 requires all three to be compiled.

Engine

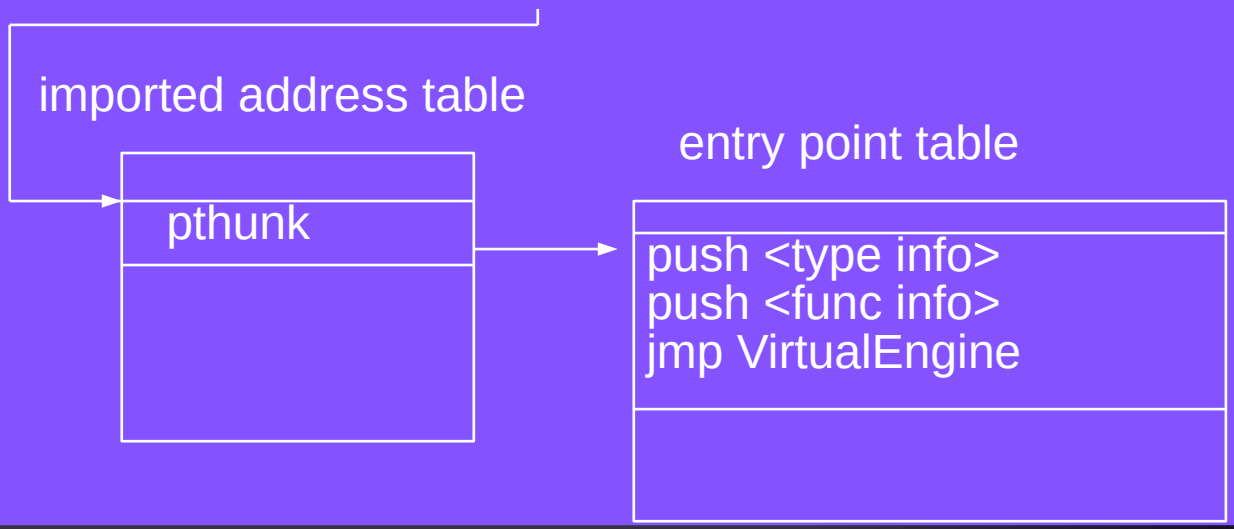
- Code generator produces code for virtual machine
  - Virtual Machine language is excodes
  - Excodes are postfix representation of program
- Stack machine
  - Single register
  - Frame pointer
  - Stack pointer
- Engine executes excodes
- Single segment (for efficiency)

VBA to VBA Function Call

```
Sub S1()
 Call S2
End Sub

REM Perhaps in another module or even project
Sub S2()
End Sub
```

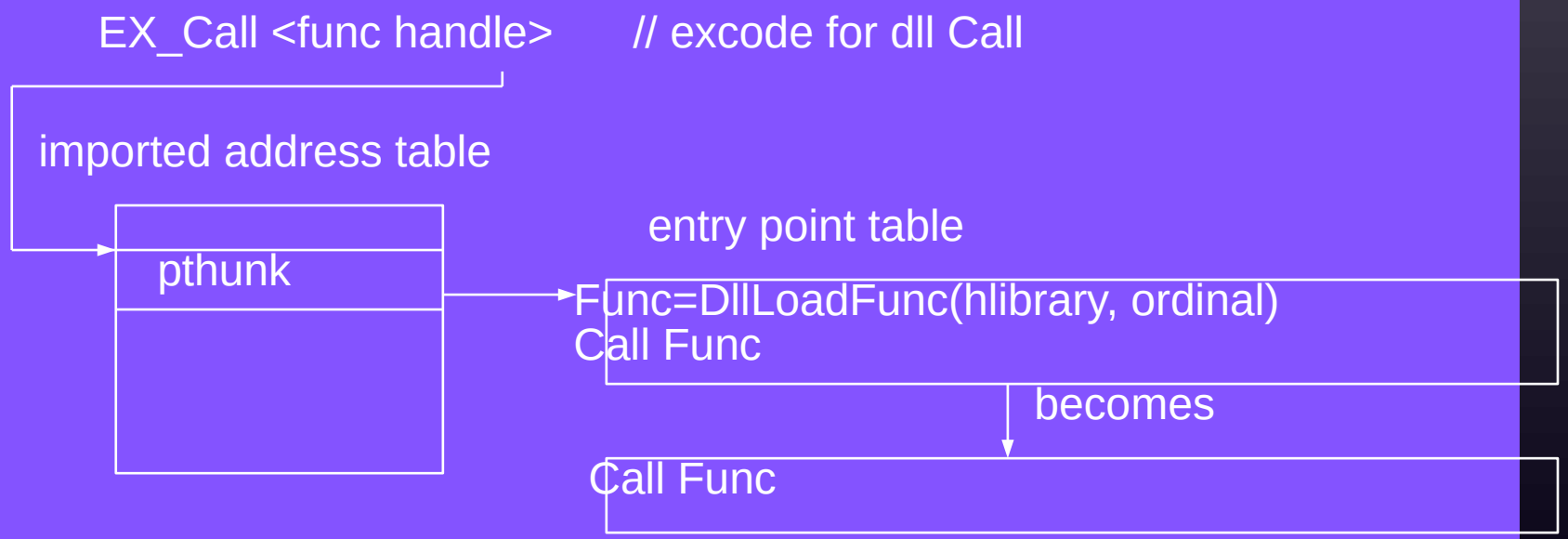
EX_Call <func handle> // excode for simple Call



VBA to DLL Function Call

```
Declare Sub S2 Lib “MyLib” ()
Sub S1()
 Call S2
End Sub
```

Same excode as VBA to VBA case
Thunk modified at run-time



Cross-process activation

- Example of VBA macro executed in Excel5
  - in-process
- Excel5 objects packaged in type library
- Example of same macro executed by sample host application manipulating Excel5
  - cross-process
    - OA provides communications layer
  - Server application activated implicitly. No need for explicit CreateObject

Questions?