COBIA PHASE III: MARSHALING

Jasper van Baten – AmsterCHEM
Mark Stijnman – Shell Global Solutions

PRESENTATION OUTLINE

- Introduction
- Outline Phase III
- Marshaling overview
- Marshaling tasks
- Marshaling alternatives
- Marshaling proposal
- Outlook

COBIA **C**APE-**O**PEN **B**inary **I**nterop **A**rchitecture

Phase 1:

native, C++, in-process, Windows, thermo 1.1,

COM binding *(proof of concept)*

Phase 2:

extend interface set, code generation from IDL

(covers most current applications)

Phase 3:

other platforms, inter-platform interop

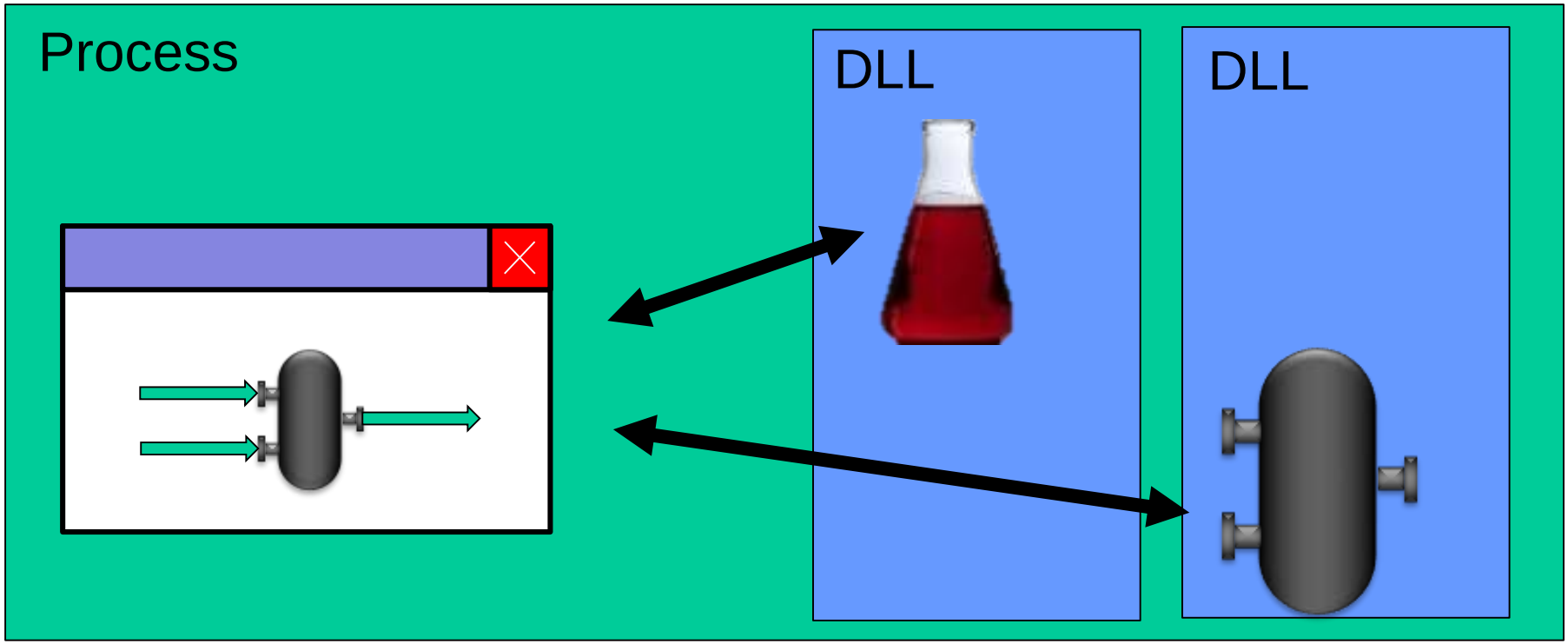
OUTLINE PHASE III

- Porting to other platforms
  - Windows: MSVC/Intel++/GCC
  - Linux: GCC/(CLang)
- Expanding Language bindings
  - Windows: .NET
  - Fortran/Java/Python: as business cases warrant
- Marshaling between different platforms

Technical proposal required

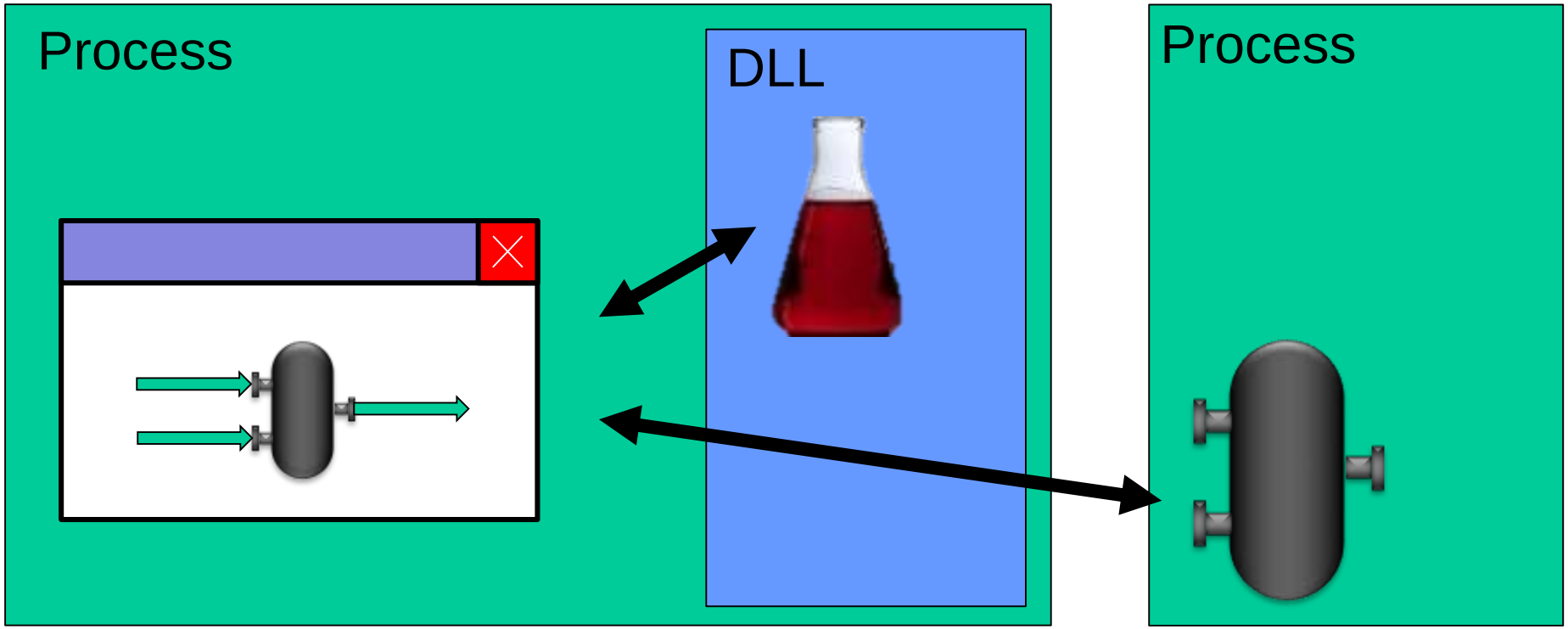
COMMON CASE: IN-PROCESS

- CAPE-OPEN defines interfaces
- COBIA defines calling convention etc…
- COBIA defines instantiation



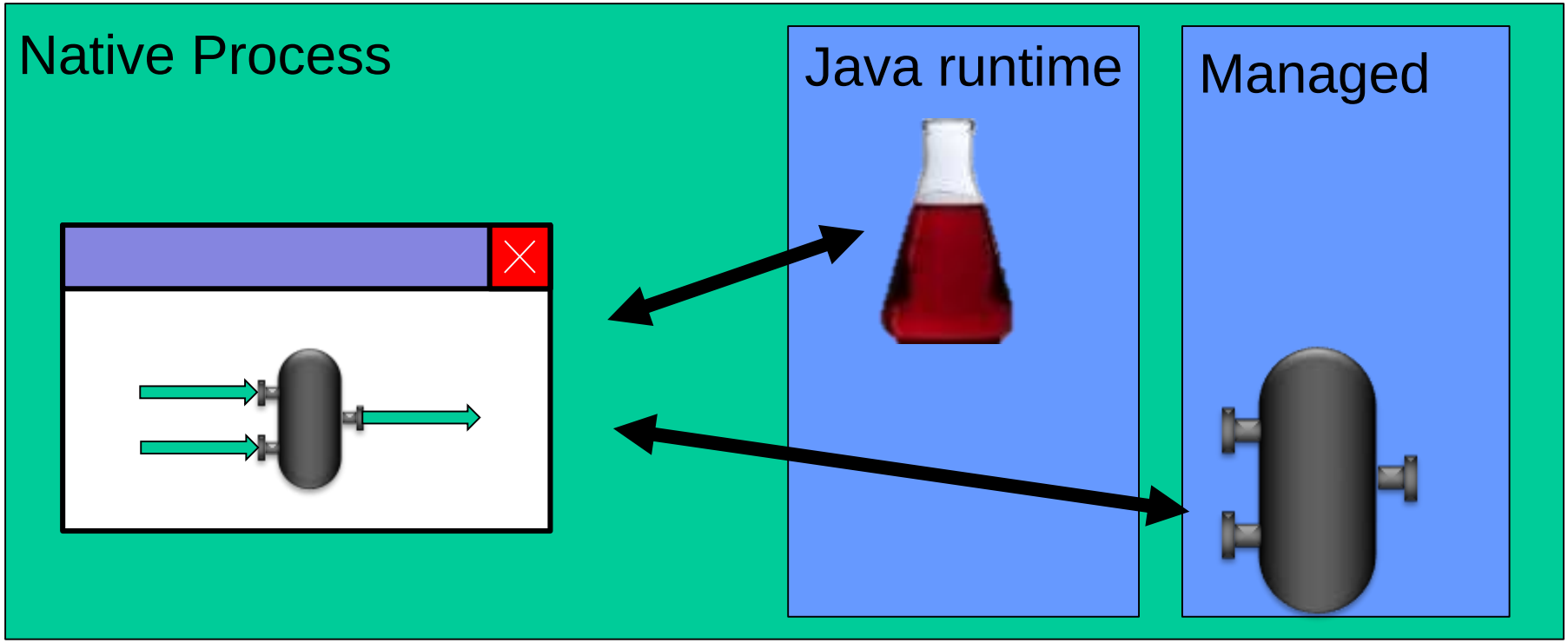
MARSHALING

- Different processes
  - 64-bit app using legacy 32-bit implementation
  - PMC hosted on different computer than PME



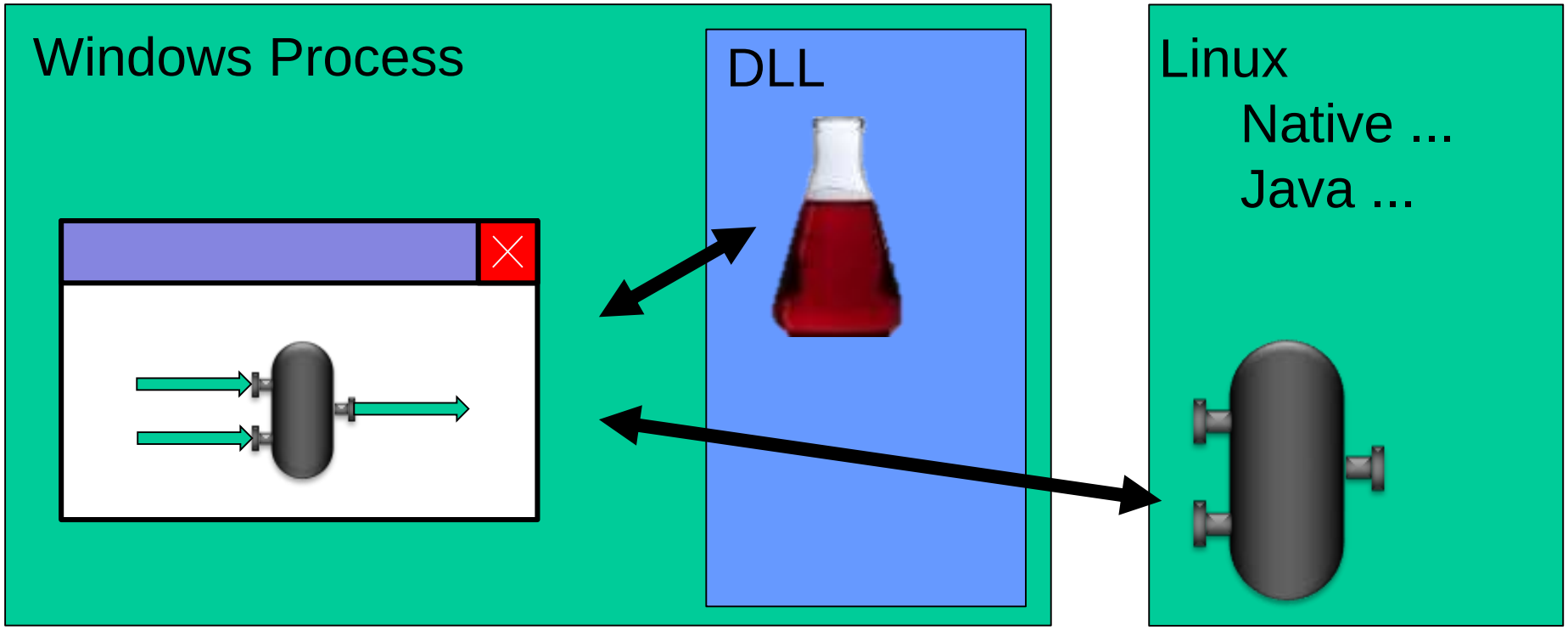
MARSHALING

- Same process, but different memory space & layout
  - Native vs .NET (managed)
  - Other VM-like environments (e.g. java, Python)



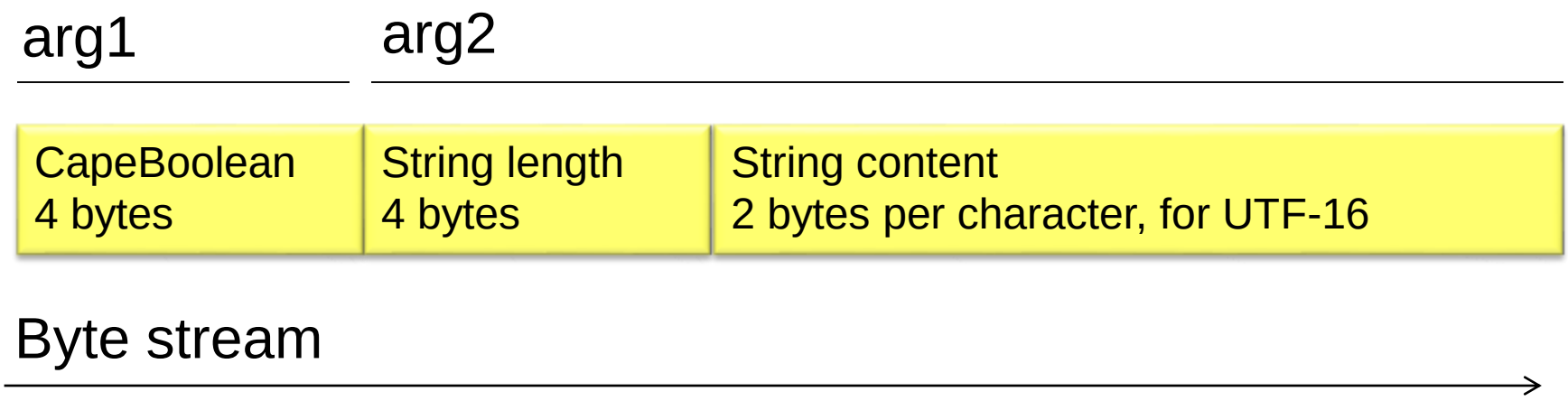
MARSHALING

- Any combination thereof
  - E.g. PME and PMC run on different OS

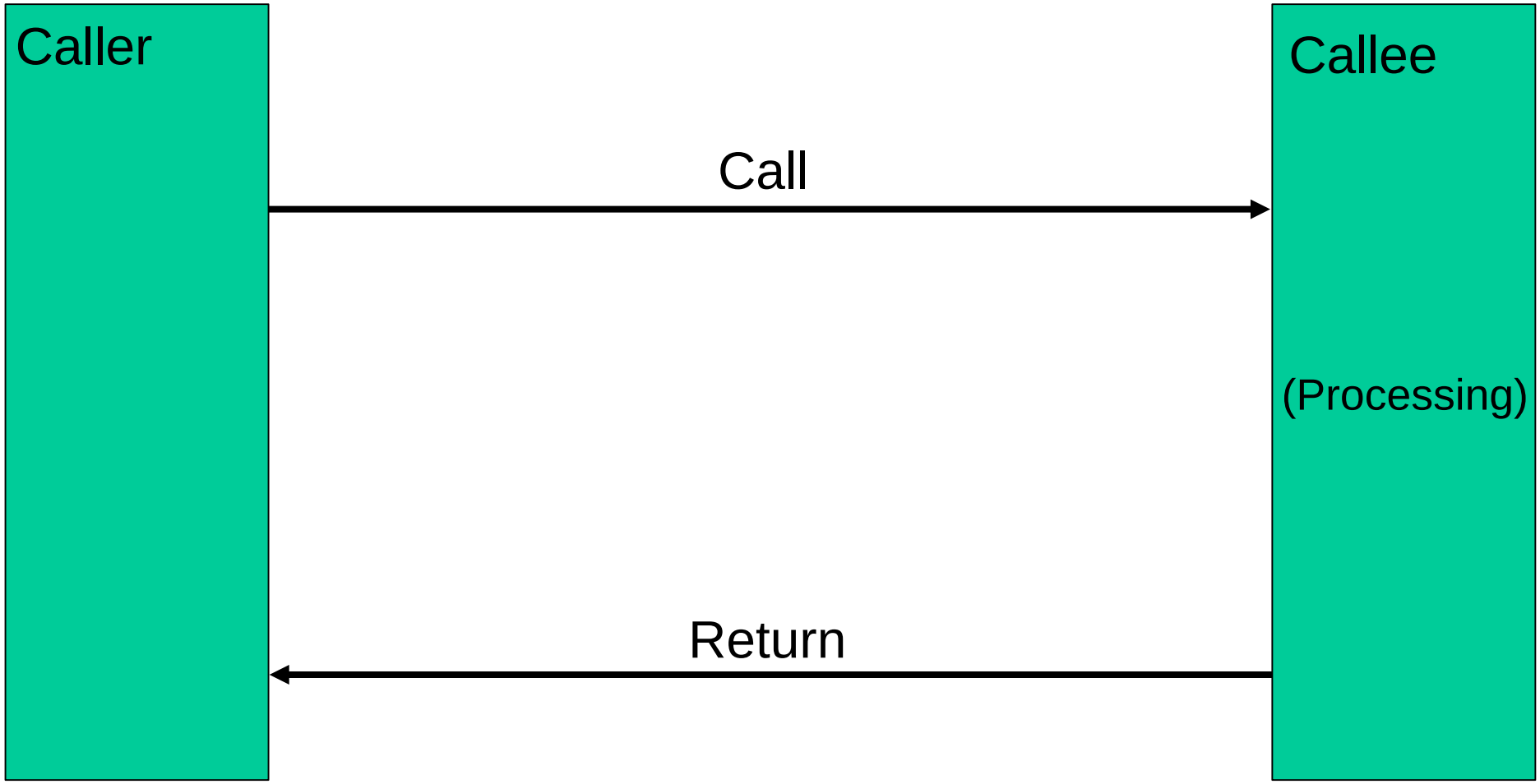


SERIALIZATION

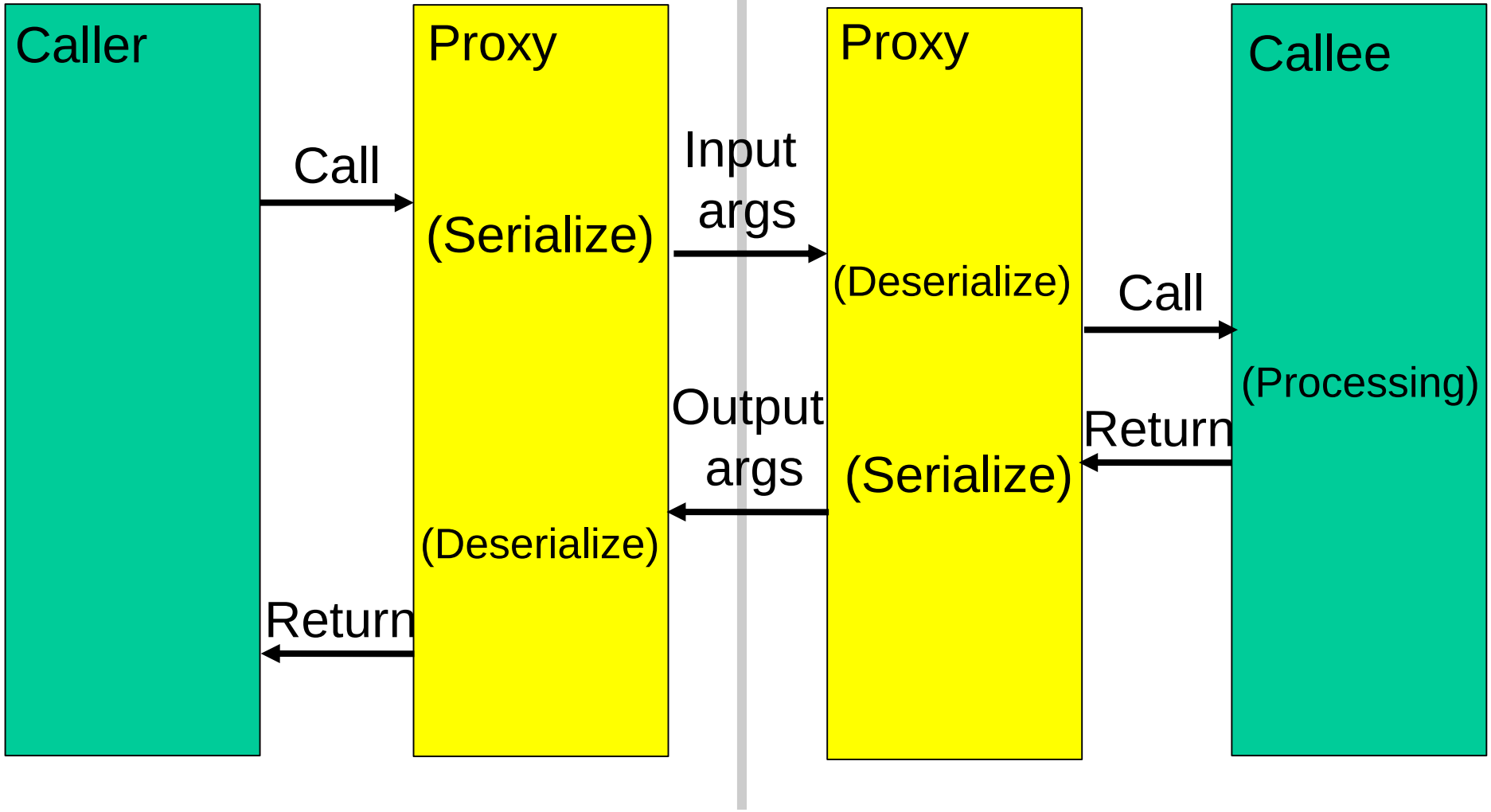
- Placing the argument in a stream
  - Example:
    SomeFunction(CapeBoolean arg1,CapeString arg2)



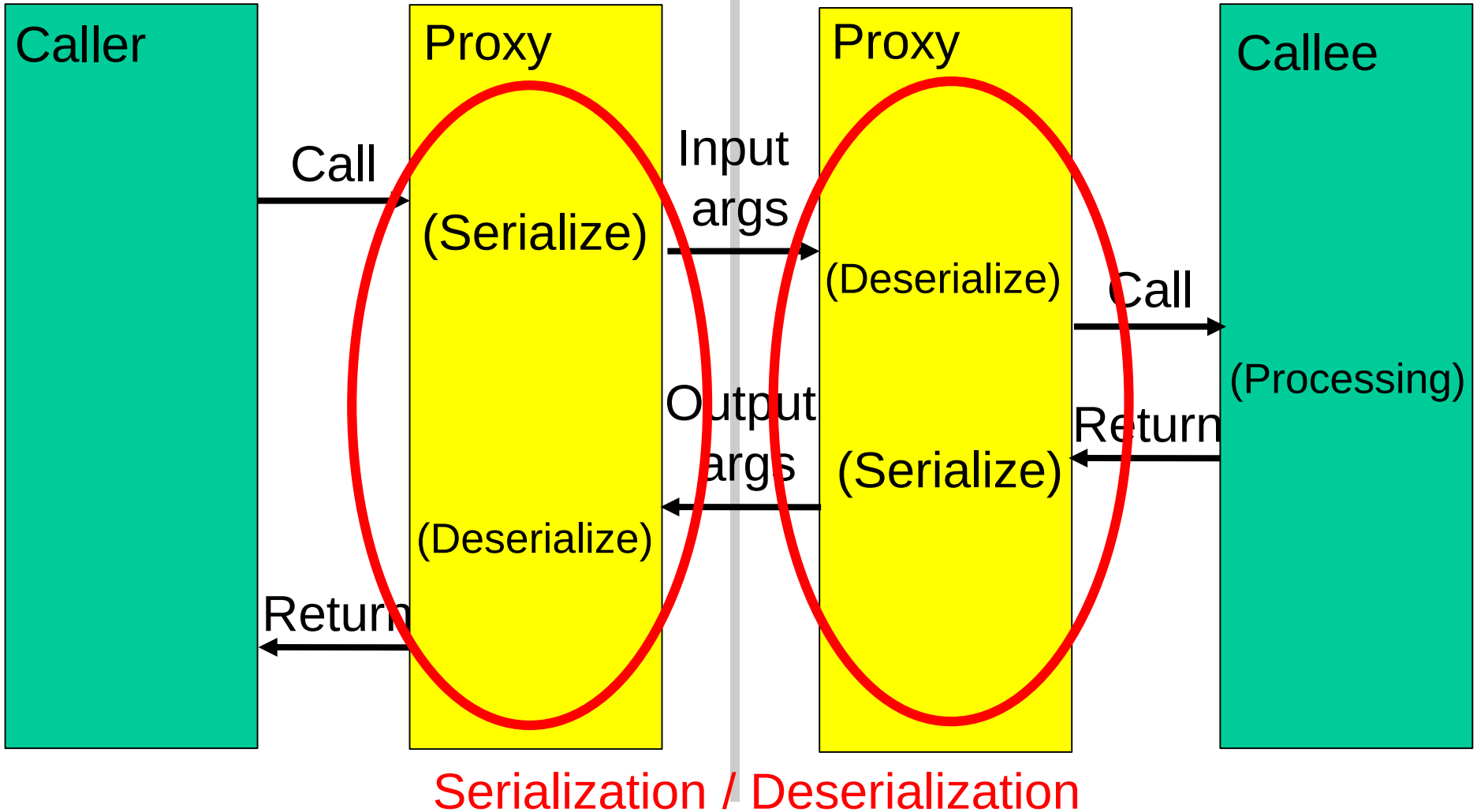
DIRECT CALL



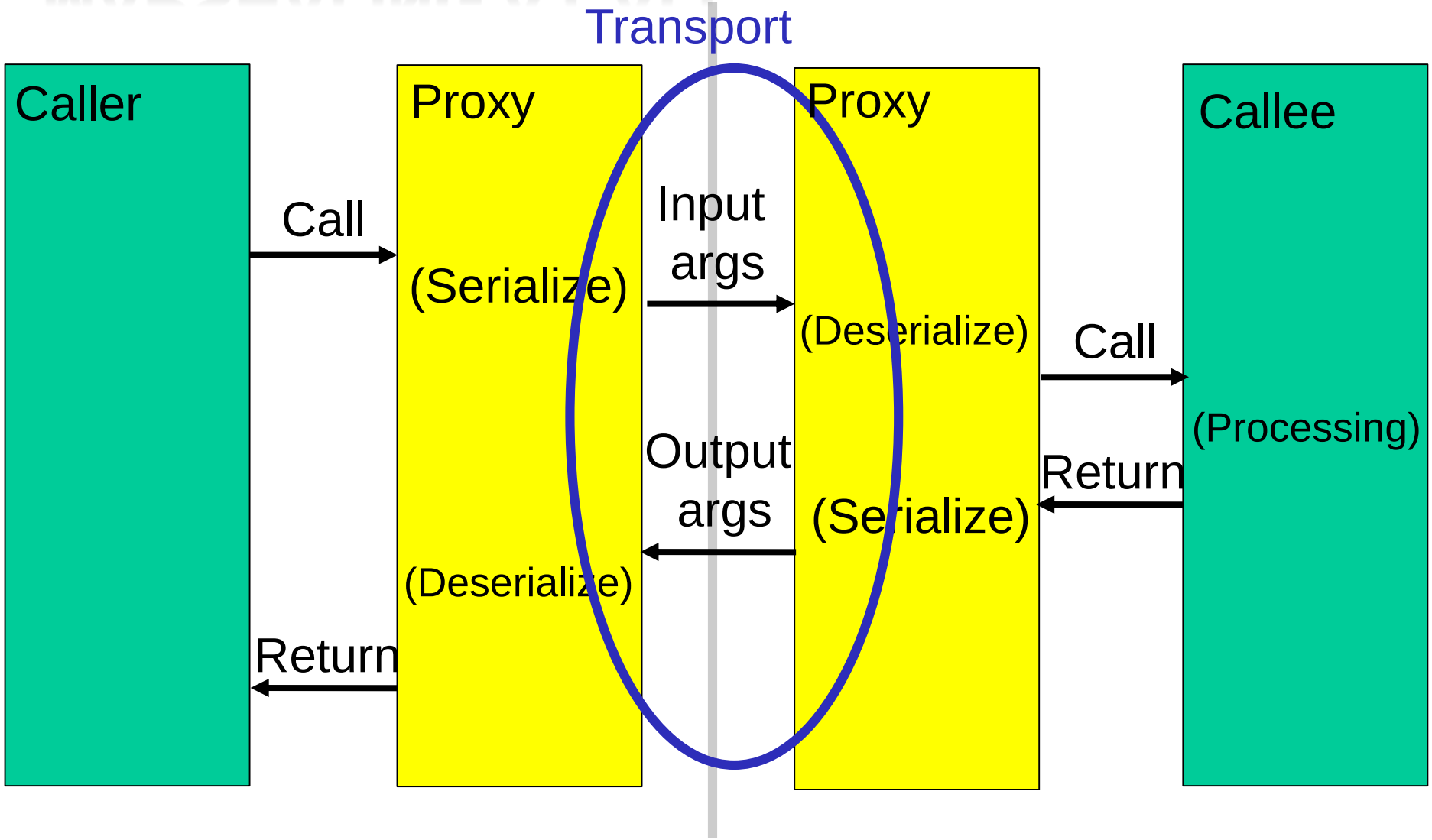
MARSHALING A CALL



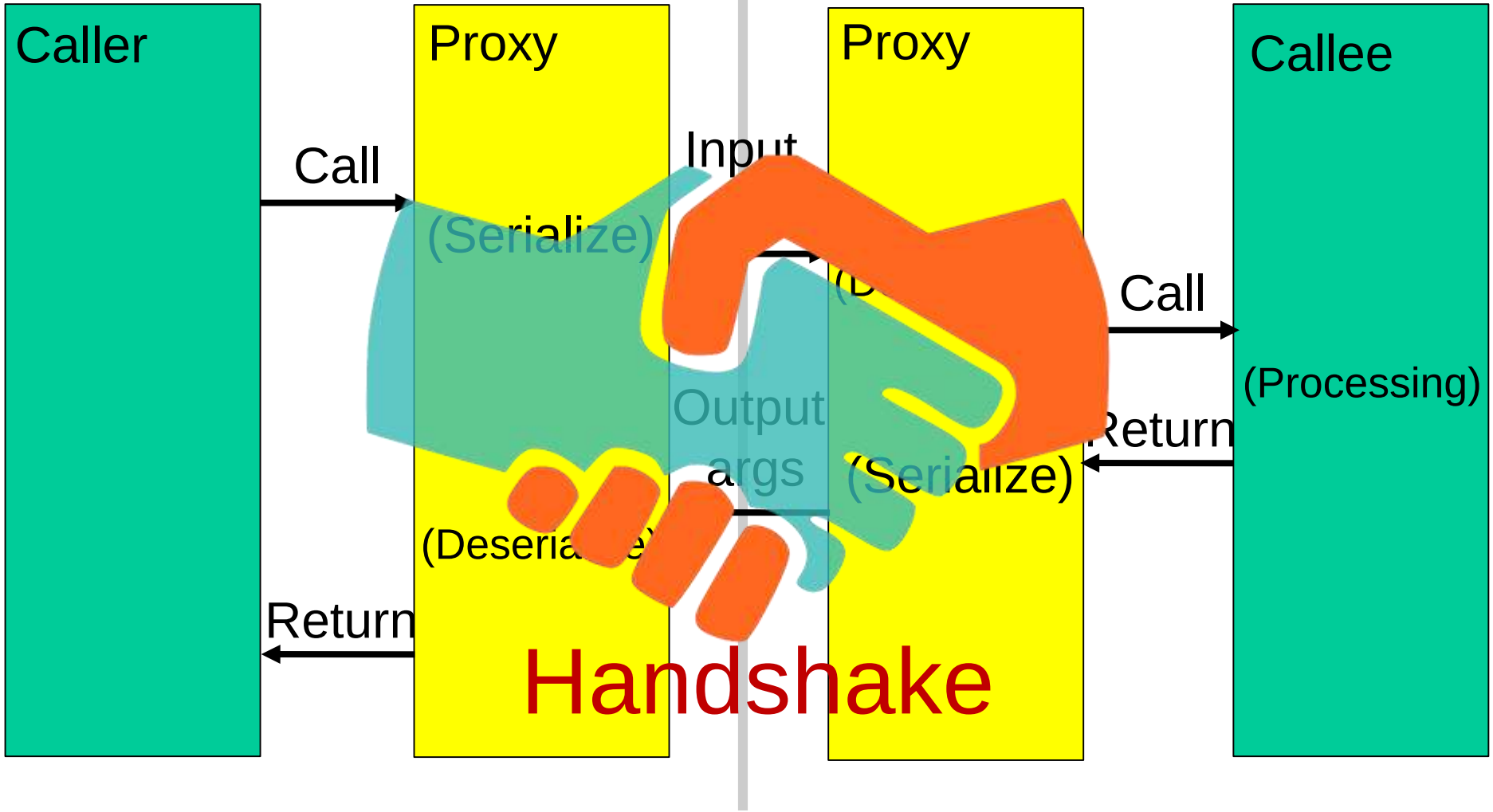
MARSHALING A CALL



MARSHALING A CALL



MARSHALING A CALL



MARSHALING A CALL

- Handshake:
  - Easy:
    COBIA defined format for agreeing on data types
- Transport
  - Easy: e.g. TCP/IP
- Serialization and Deserialization
  - Not obvious
  - Data to be (de-)serialized depends on call

A FUNCTION CALL

- Caller pushes arguments and return address
  on call stack and/or registers
- Caller transfers execution point to callee
- Callee pops arguments + return address
  off call stack and/or registers
- Callee processes data
- Callee pushes return value on call stack and/or registers
- Callee transfers execution point to return address
- Caller pops return value off call stack and/or registers

A FUNCTION CALL

Exact format depends on:

- calling convention
- data alignment (32-bit, 64-bit)
- operating system ABI
- data format (data size, string encoding)
- potentially hardware

ALTERNATIVE 1: MANIPULATE CALL STACK

- Function call on-the-fly by COBIA
- Arguments are determined from IDL / Registry
- Call stack and registers are manipulated programmatically
- This is what Windows / COM does
- Advantage: all that is needed is IDL / type info

- Disadvantage:
  - Requires assembly (C++ cannot do this)
  - Complex, and platform dependent

ALTERNATIVE 2: A LOCAL COMPILER

- All of these tasks is exactly what a compiler does when compiling a function call
- Proxy code could be compiled on the fly

- Advantage: all that is required is type info
- Disadvantage:
  COBIA needs to distribute with compilers

ALTERNATIVE 3: PRE-COMPILED SERIALIZERS

- All of these tasks is exactly what a compiler does when compiling a function call
- Code for proxy could be compiled on by the manufacturer

- Advantage:
  Proxy code to be completely generated

- Disadvantage:
  Proxy binary needs to be shipped

PROPOSAL 1/2

- Proxies for interfaces are shared components
- Proxy for all known CAPE-OPEN interfaces are provided by COBIA itself
- Software vendors can provide proxies for custom interfaces
- Code for the proxies is generated by COBIA
- Proxy software components are part of vendor installer as shared component
- Vendor provides proxies for all platforms on which vendor software may run

PROPOSAL 2/2

- Upon installation, shared component for serialization is registered using COBIA registrar function
- On a single platform, multiple serializers may be required
  - E.g. Windows:
    - x64 native
    - x86 native
    - managed
- COBIA takes are of handshake and transport
- COBIA data types are not marshaled, but serialized

A WORD ABOUT LOGGING

- COM does not have built-in facilities
  - COLTT (CO-LaN)
  - OATS / COULIS (COCO Simulator)
- For COBIA we can make a logging ‘fence’
  - All traffic automatically logged
  - No need to implement a logger specific to any interface
  - Part of COBIA distribution

CONCLUSIONS

- Marshaling is complex
- Serialization/deserialization and function calls are the difficult part
- Proposal: vendors deliver precompiled serialization/deserialization code generated by COBIA from IDL
- COBIA is pretty cool!