www.salfordsoftware.co.uk

Fortran 95 for the .NET Framework

David Bailey

David.bailey@salfordsoftware.co.uk

Why .NET?

- Backed by Microsoft – will gradually displace Win32
- Uses Just In Time technology – CPU independent computing
- Simplifies interfaces to other .NET languages and web applications
- .NET software is supposed to be more secure
- Eventual access to 64-bit address space

The .NET language model

- .NET is designed for pure object oriented languages
- C# is a typical .NET language
- Compiler uses ‘metadata’ rather than header files
- The C# compiler does not have a link phase
- The entire system library is object oriented

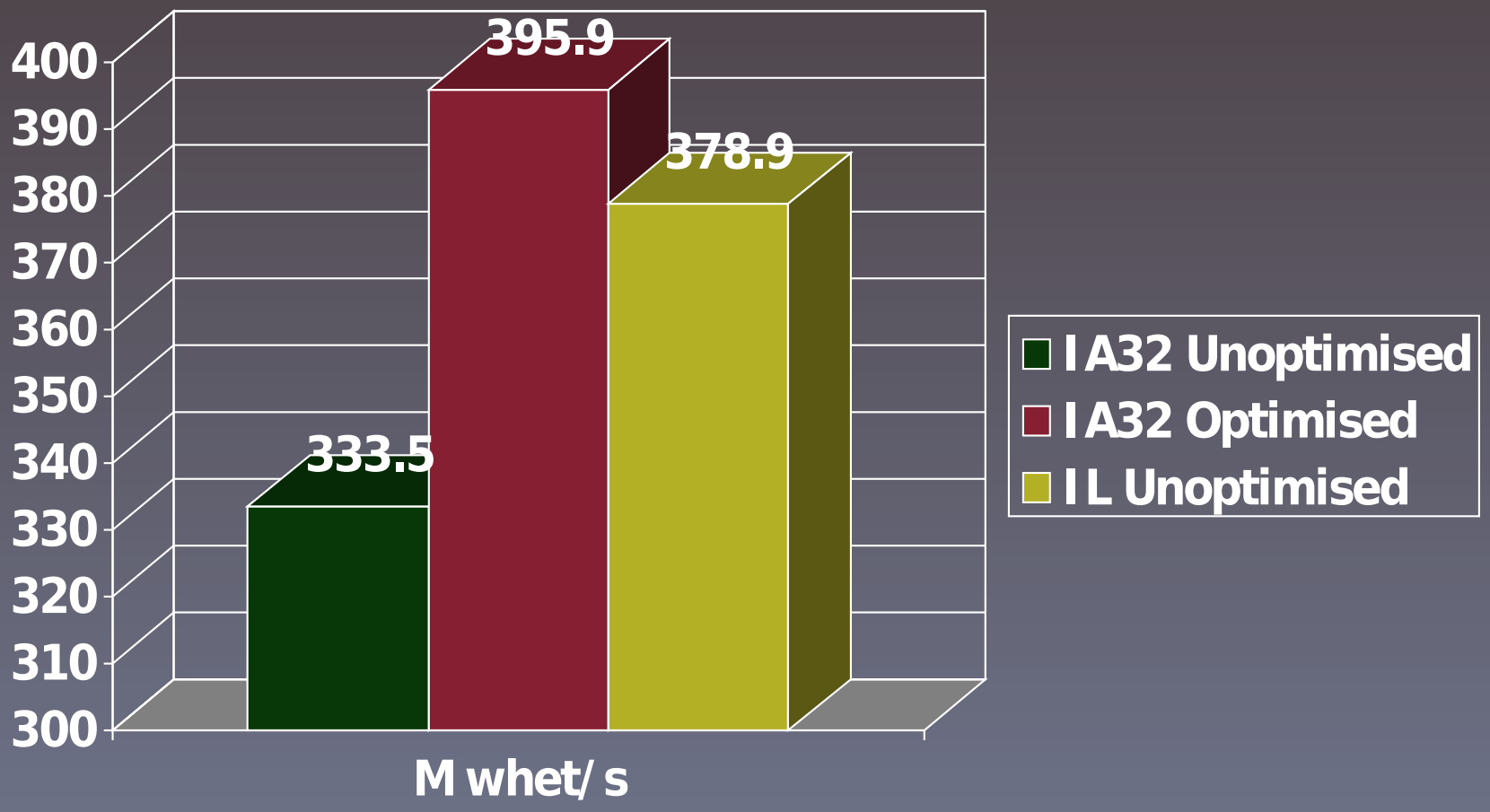
Why no link phase?

- A C# compilation produces an executable or an assembly (effectively a DLL)
- Other assemblies on which the program depends must be physically present to supply metadata
- If assembly A refers to assembly B then B cannot refer to A (except by a trick)

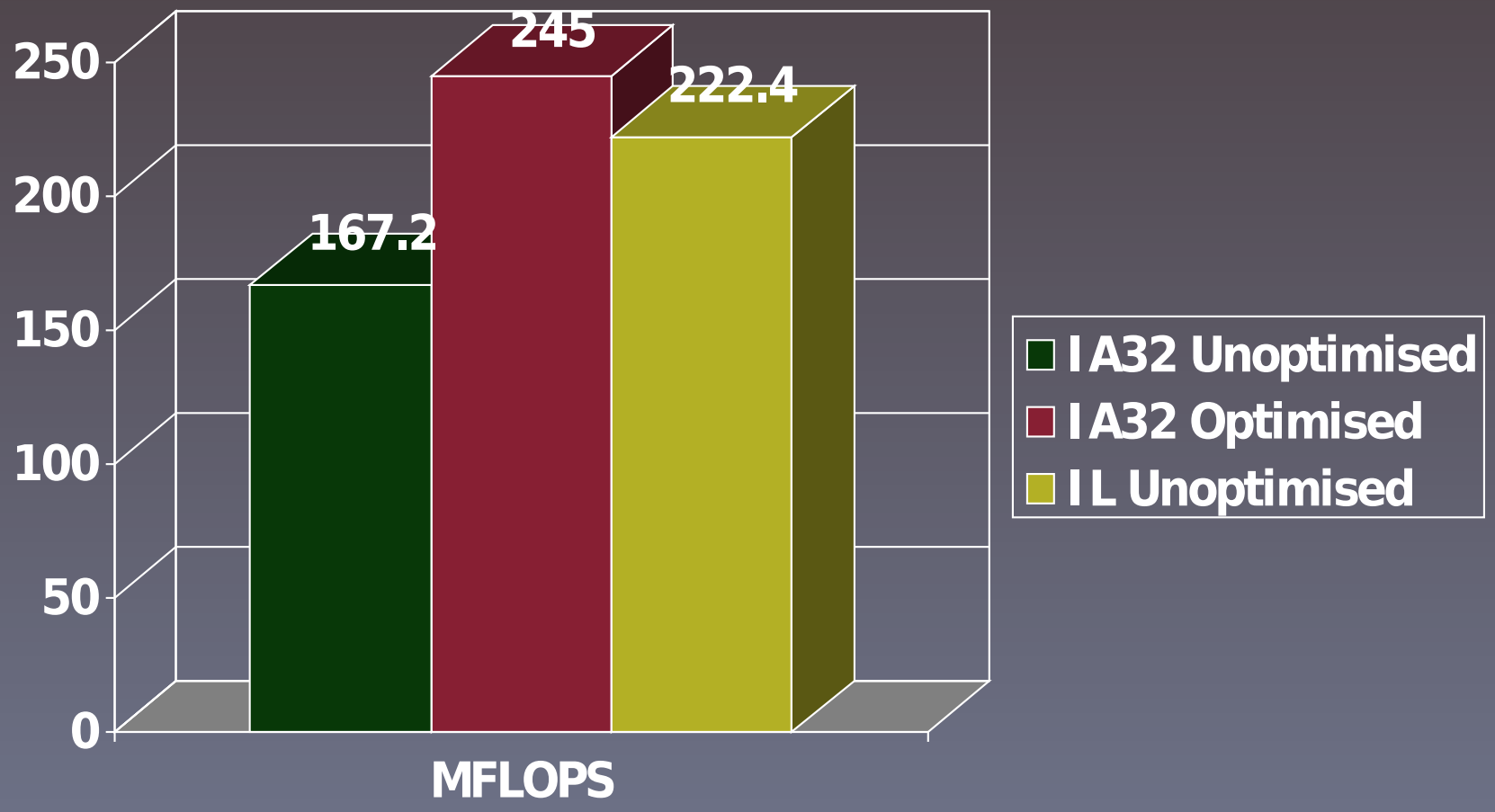
IL Performance

- Somewhere between optimised and non-optimised native code
- Difficult to find truly representative benchmark
- Integer performance relatively better than floating-point performance
- IL optimiser for FTN95 still under construction
- IL code optimised by JIT compiler

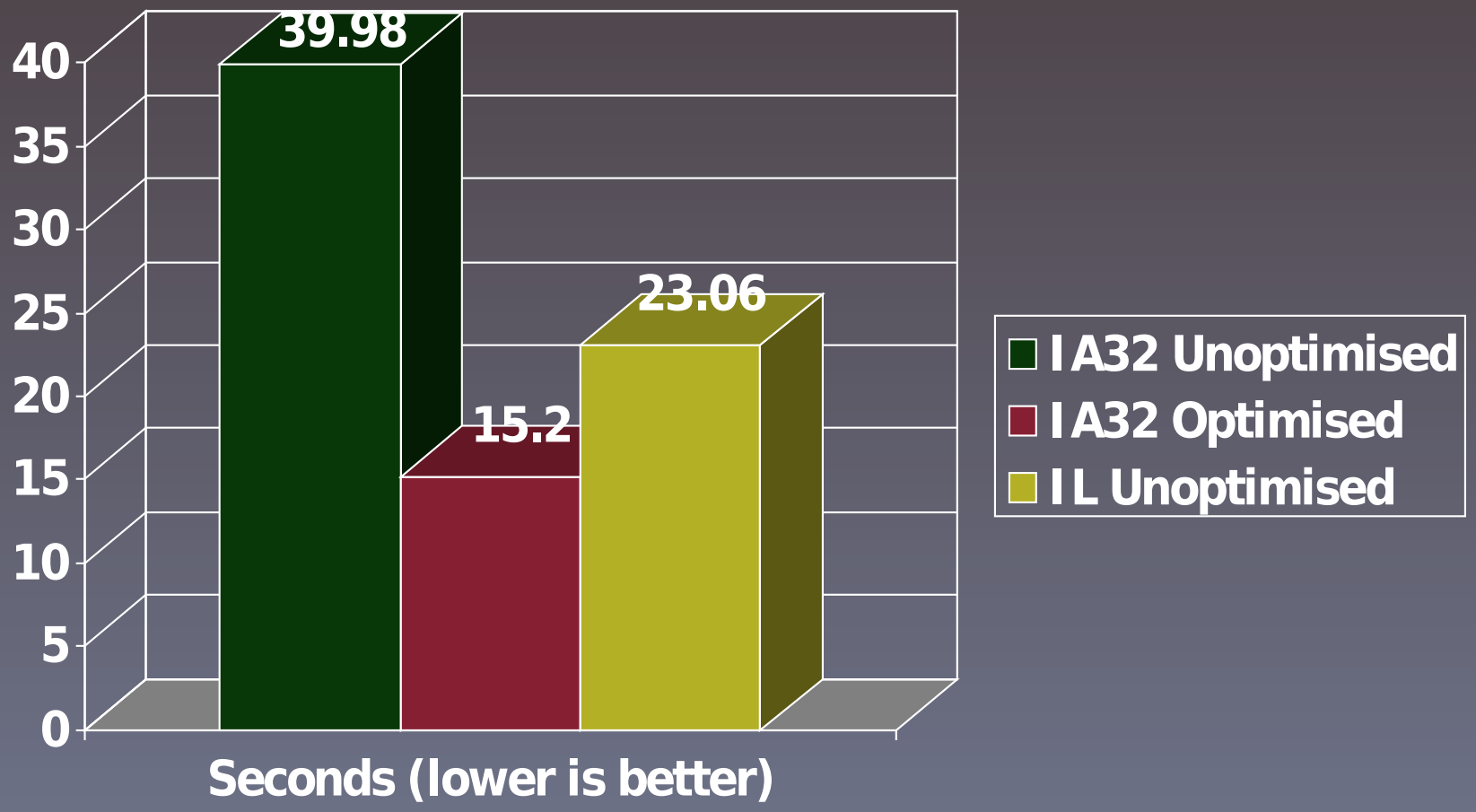
Whetstone Benchmark



LINPACK Benchmark



LARGMAT8 Benchmark



Fortran issues

- Must compile the whole of Fortran
- ENTRY, EQUIVALENCE, COMMON, contained routines
- The .NET environment assumes that routine interfaces are always present
- No (public) .NET object format
- .NET arrays are inefficient and do not work with EQUIVALENCE or COMMON

FTN95 solutions(1)

- Re-introduce object format (.DBK)
- Linker DBK_LINK creates assemblies in a Fortran aware fashion
- Link diagnostics are Fortran specific
- DBK_LINK resolves ENTRY statements and contained routines
- Matches routine calls in a Fortran specific fashion
- Every Fortran routine becomes a static method of a class. MODULE's and COMMON become static members.

FTN95 solutions(2)

- Arrays use unmanaged memory - array bound checking is added as required
- Fortran does not satisfy PEVERIFY – unsafe constructs JIT to efficient code
- Some calls to WIN32 in the short term
- Entry points share data, but not code
- EQUIVALENCE handled using structs and unmanaged memory

Integration with CLS

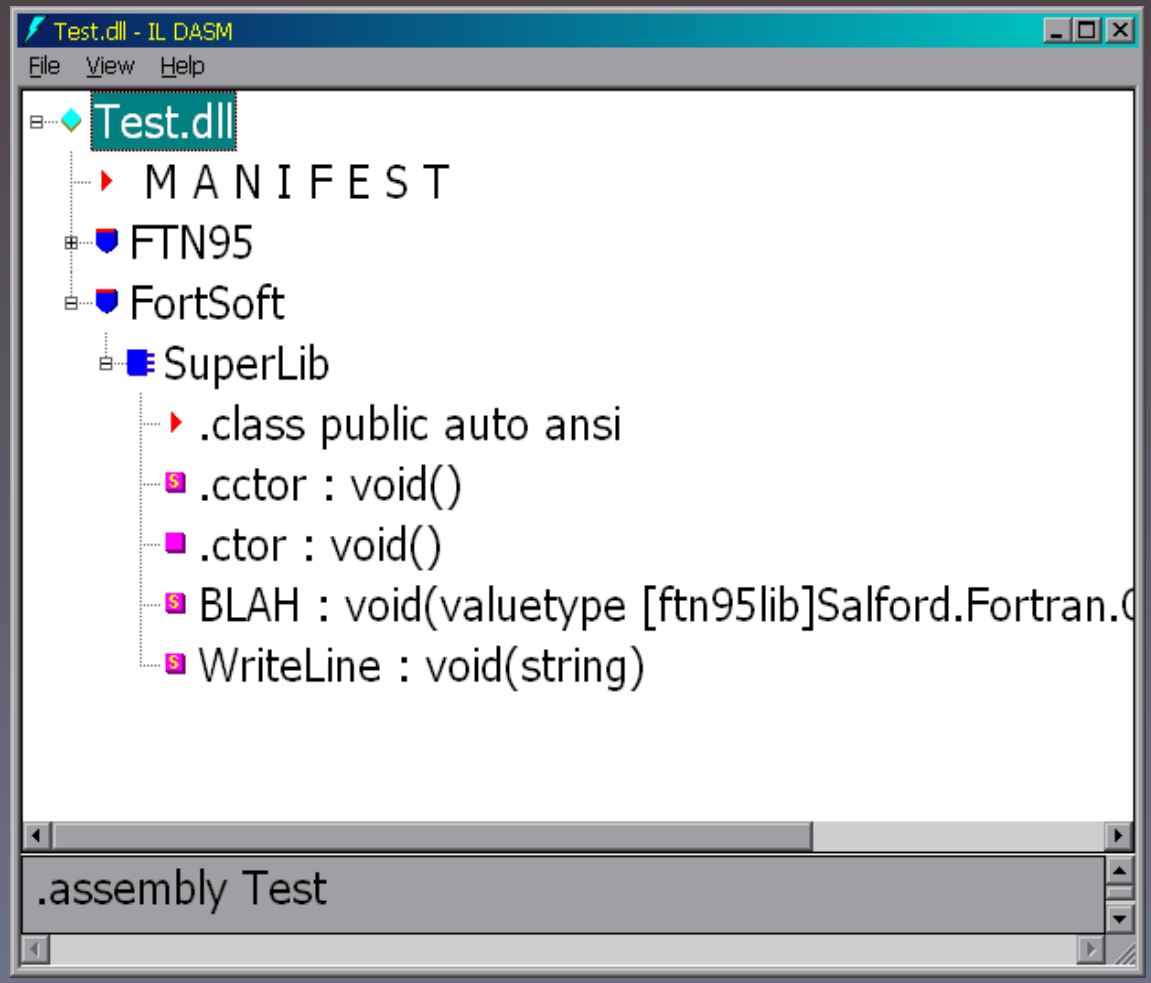
- **Call non-FTN95 methods with `assembly_external`**
- **Expose CLS compliant interface with `assembly_interface`**
- **.NET objects with `object("System.Int32")`**
- **Exception handling with `try…throw…catch…finally…end try`**

Integration with CLS

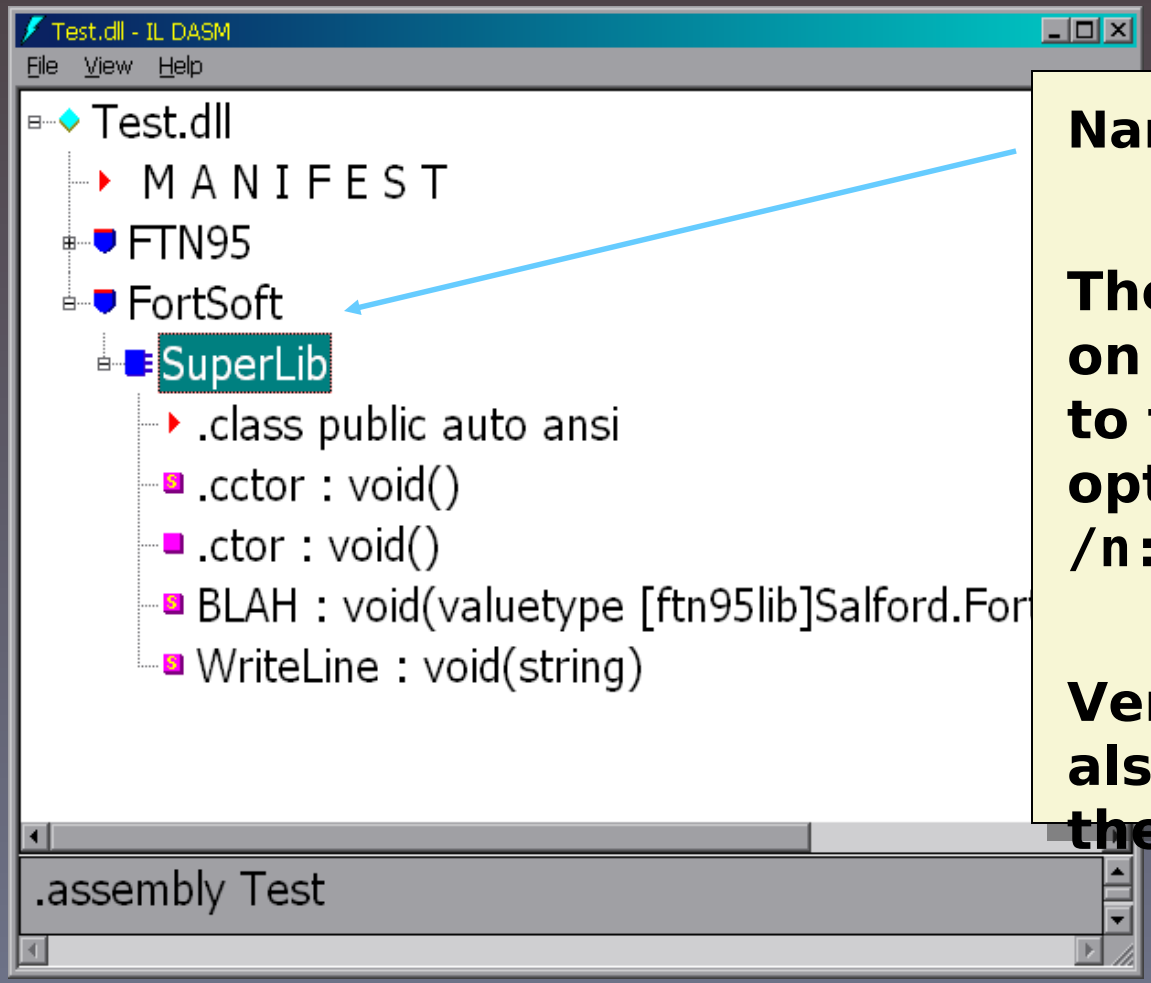
Create or call a method in a class :

```
subroutine blah(s)
   character(len=*), intent(in) :: s
   assembly_interface(name="WriteLine")
   assembly_external(name="System.Console.WriteLine") foo
   call foo("{0}, world.", s)
end subroutine
```

Integration with CLS



Integration with CLS

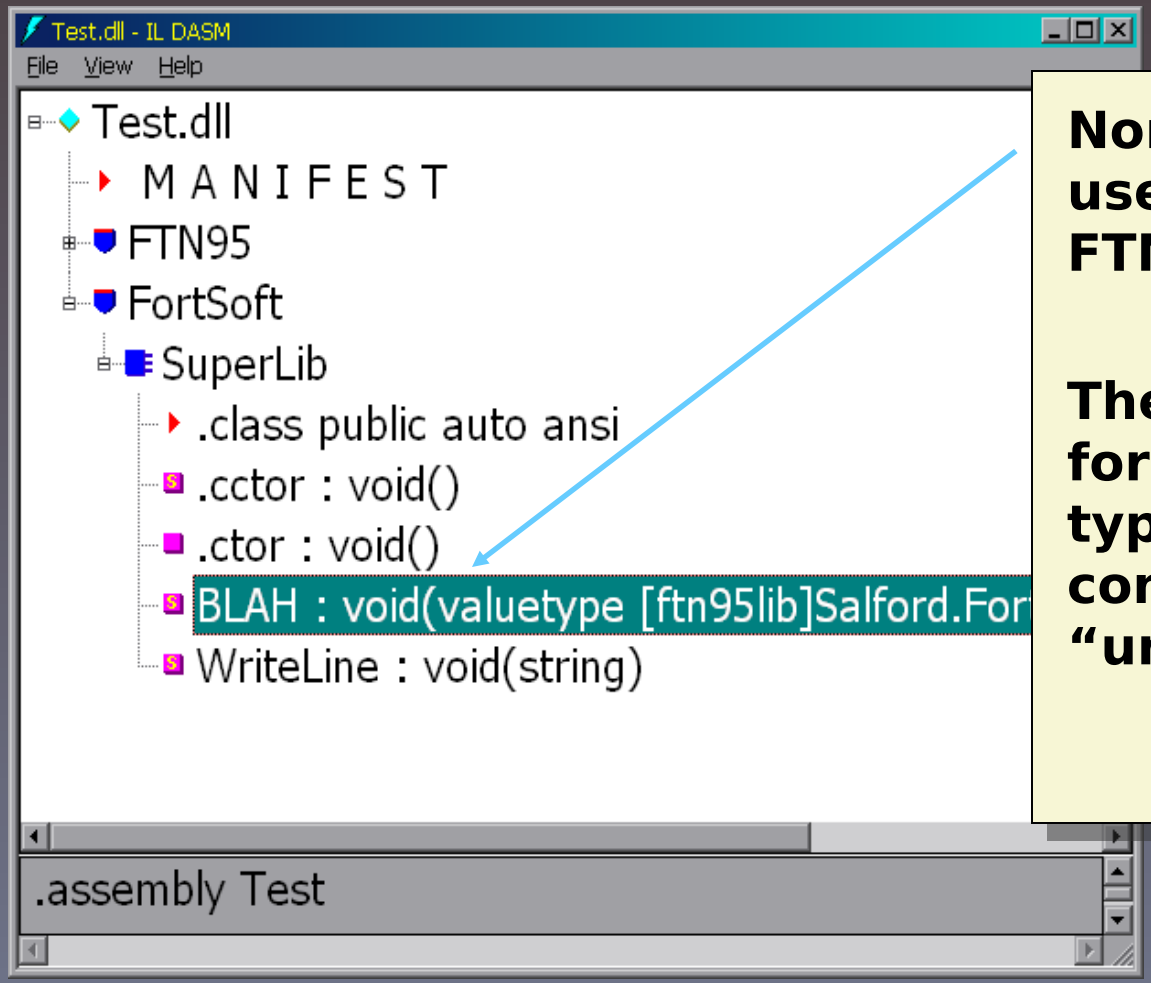


Namespace and class.

These were specified on the command-line to the linker with the option:
`/n:FortSoft.SuperLib`

Versioning etc. can also be specified to the linker.

Integration with CLS



Non-compliant method used by FTN95 → FTN95 calls.

The CLR types used for some Fortran types are not CLS compliant or are "unsafe".

Integration with CLS

Trapping an exception:

```
try
    call do_something
    catch(exception)
    call recover
    finally
    call cleanup_regardless
end try
```

Integration with CLS

Using .NET objects :

```
object("System.String")str,str1
Object(“System.Object”)obj
character*10 fred
fred="r"
str=new@("System.String",fred)
obj=cast@(str,"System.Object")
str1=cast@(obj,"System.String")
call wr(str1)
end
```

CheckMate

Advanced Run Time Checking

- **Undefined variable access**
- **Overwriting of DO-loop index, constants and INTENT(IN) variables**
- **Dangling POINTER references**
- **Argument type/length mismatch**
- **Array bounds checking, even for `integer :: array(*)`**

A wider perspective

- JIT technology is well suited to fast CPU's with plenty of memory
- CPU independent computing is already useful (JAVA) and may dominate in the years ahead
- Should set INTEL, AMD, and others head to head
- Theoretically JIT technology should out-perform traditional techniques

Salford FTN95/.NET

- Full integration with Microsoft Visual Studio .NET
- Easy to use from command line
- Good managed run-time performance
- Full access to CLR
- Advanced debugging options
- Old code runs as IL assembly without requiring changes to source code

Contacting Us

Salford Software Ltd
Adelphi House
Adelphi Street
Salford
UK
M3 6EN

web: www.salfordsoftware.co.uk
e-mail: sales@salfordsoftware.co.uk
tel: +44 161 906 1002
fax: +44 161 906 1003