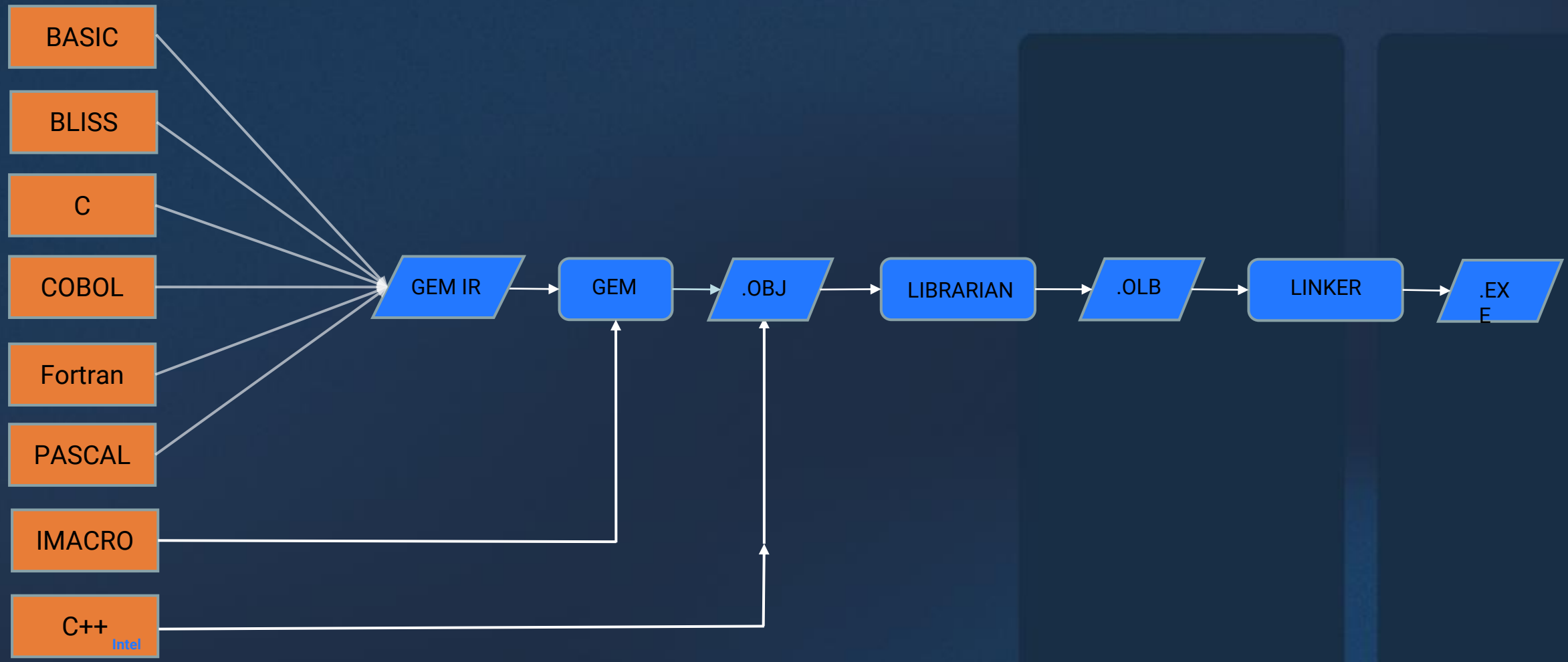
OpenVMS Compiler Update

John Reagan

April 2025

History and Design

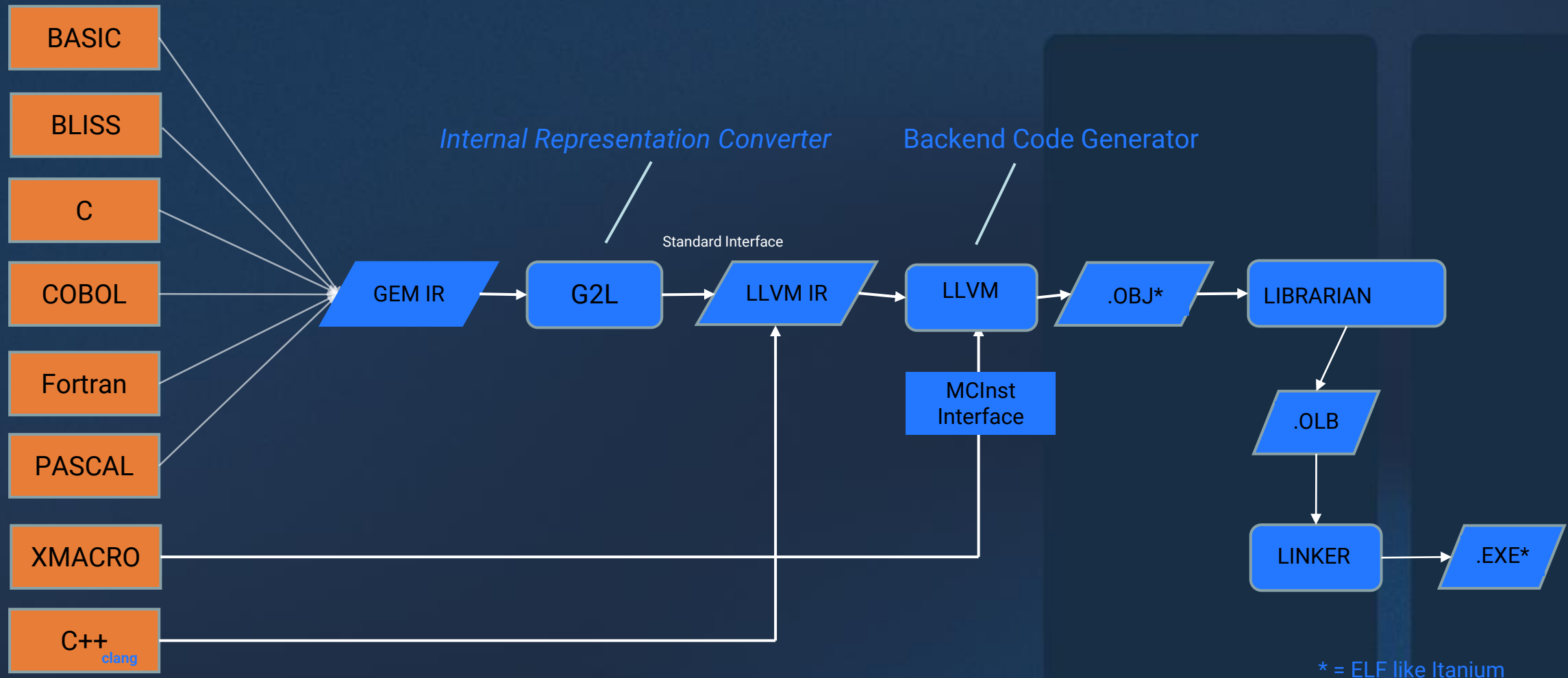
OpenVMS Itanium Compilers



OpenVMS x86-64

- HP licensed OpenVMS to VMS Software Inc in 2014 for future platform support which included porting to x86-64
- Legacy GEM backend is stale and doesn't know x86-64
- Instead of throwing money at GEM, we picked LLVM to get modern code base and not have to chase all possible chip features
- Still need to provide "recompile and go" for customers
- Leverage existing frontends which generate GEM IR/symtab
- Create a GEM IR to LLVM IR converter (G2L)
- Leverage clang as our C++ offering

OpenVMS x86-64 Compilers



GEM Meets LLVM

- Map GEM IR nodes (~275) to LLVM IR nodes
  - Needed about 240 GEM IR nodes for BLISS and C with remainder used by other frontends
  - Many GEM nodes have nice simple mappings, but some result in interesting IR sequences or converter abstractions (strings, packed decimal, complex, uplevel accesses, BASIC, etc)
  - Converter currently about 36K lines of C++ (source & headers & comments)
  - **long double** is still in progress
  - GEM's static memory initialization quite different than LLVM

LLVM Meets OpenVMS

- A handful of OpenVMS additions to LLVM
  - Mixed pointer size linker relocations
  - Memory model changes
  - **.note** section generation for module name, compilation date/time, etc
  - AMD64 ABI additions for argument count in RAX register
  - Additional DWARF language tags
  - Additional EH unwind descriptors for Macro-32 VAX register emulation
- Use the LLVM libc++ and libc++abi libraries including updating the abi library to use the lib$ Calling Standard routines
- Build various LLVM tools

Cross-compilers and Native Bootstrapping

- We had early cross-compilers (Itanium-host, x86-target) built with an older LLVM 3.4.2 code-base. No C++ and no BASIC. These are used to build the OS and included in the cross-tools kits. These provide no optimization.
- We have a Linux-hosted LLVM 10 clang with the OpenVMS object additions.
- We use the Linux compiler to compile clang/LLVM, move the OBJ files to OpenVMS, and create x86-native object libraries and the C++ compiler.
- We used the cross-compilers to build the various frontends to link with the LLVM 10 libraries to create the first generation of native compilers including BASIC
- We used those native compilers to build themselves natively.
- These compilers are built with optimized native compilers and generate optimized code. There are still several areas under investigation for optimization (routine inliner, better pointer alias analysis, etc.)

Compiler Status

Compiler	Current Version	Field Tests
BASIC	V1.10	Soon – Bugfixes
BLISS	V1.14	X1.15 – Bugfixes
C	V7.6	X7.7 – Bugfixes
C++ <more slides below>	V10.1-2 (new)	A10.1-3
COBOL	V3.3	X3.4 – Bugfixes
Fortran	V8.6	X8.7 – Bugfixes
Macro	X6.0-111 (V9.2 thru V9.2-2) V6.0-115 (V9.2-3)	V6.0-117 (V9.2-3U1) - Improved debug support - Bugfixes
Pascal	V6.4	Soon – Bugfixes
X86ASM (native assembler)	V10.0	A10.1-3 – Bugfixes

Debug

Debug

- Itanium GEM generates DWARF 2+3
- Debugger only processes the GEM DWARF, not full DWARF
- LLVM generates DWARF 4 but doesn’t know legacy compiler info
- Teaching debugger about new C++ tags
- Teaching LLVM about legacy compiler tags
- Better debugger in V9.2-3
- Even better debugger in V9.2-3 update 1
- Some fixes require better compilers

Macro

Macro-32

- Unlike Alpha and Itanium, there are not enough hardware registers to map R0-R31
- Use memory locations for these Alpha pseudo registers managed by the system
- Operations like ADDL3 R2,R3,R4 are two memory reads, the addition, and a memory write
- RET instructions put the results both into R0 and %rax
- BLISS LINKAGE, C pragma linkage, DEBUG, and EH unwinding code also know about the pseudo registers
- Porting Macro-32 should not require any changes

Legacy (non-C++) Compilers

Legacy Compilers

- Well-written programs port with little modification needed
- Early compiler bugs are with things that are difficult to describe to LLVM such as static data initialization and COMMON blocks
- Programs that use target-specific knowledge need to be updated
- Most common program mistakes include
  - Alignment holes added by GEM protect code with buffer overruns and 32/64 bit assignment mistakes. LLVM provides no such alignment for x86.
  - Any assumption about location of code in 32-bit space vs 64-bit space.

C++

C++ Update

- Itanium only at C++03 standard but many open source applications now demand a higher level
- Itanium compiler is EDG/Intel-based with license/support issues
- Itanium STL is old with a non-portable license
- Need to update to a modern C++ for OpenVMS x86-64
- Obvious choice is the clang frontend from LLVM

Clang Meets OpenVMS

- Differences from Itanium
  - Size of long, size_t, nullptr_t, ptrdiff_t == 64 with no option to change
  - Pointer size is 64 unless changed to 32
  - Names is “as-is” unless changed
  - Names is “no length limit” unless changed
  - Message names for pragma are different
  - Current pointer-size affects new operator
  - No global new/delete
  - No VAX floating
  - Two compilers: one with DCL interface; one with Linux interface

C++ V10.1-2

- Several new features including
  - Listing file support similar to the Itanium compiler
  - New[ ] now looks at current pointer size to allocate in 32-bit vs 64-bit heap
- A10.1-3 field test includes
  - Assorted bugfixes for 32-bit strings; listing files; MMS dependencies
- LIBCXX/LIBCXXABI RTLs bundled with V9.2-3
- CXXFIXUP kit included in kit to help with RTL transition for V9.2-2 systems

Compiler Futures

- Complete long double support
- Improved optimization
- Continued work on debugger support
  - Legacy compilers require OpenVMS-unique DWARF
  - Debugger requires better C++ knowledge
- Missing /MACHINE_CODE listing
  - Choice #1 – add hooks into LLVM for code and static data
  - Choice #2 – add metadata into OBJ for ANAL/OBJ/DISA
- Refresh LLVM
  - Currently using 10.0.1. Current version is 20.1.2.
- Provide buffer overflow detection from LLVM
- More work for libcxx for C++17 and beyond
- Investigate TLS (thread local symbols)
- Investigate various LLVM sanitizers

Thanks!