



News from VMS Software Inc. (VSI)

x86 OpenVMS Update

Clair Grant, CTO

Agenda

- Release Plan
- First Boot
 - Why was it important?
 - What was it?
- V9.0 EAK
- Getting from First Boot to V9.0
- Current Status
- Running System Examples



Release Plan

Release Plan

- Cross Tools Kit: compile / link on IA64
 - Jan / 2019 – BLISS, C, XMACRO, Linker and associated tools
 - May / 2019 – Updates plus FORTRAN
 - COBOL, BASIC, PASCAL coming soon
 - NOTE: no C++
- V9.0 EAK – very limited developer kit (12-15 participants):
 - Will start with 3 or 4 and add more well before V9.1
 - not a “kick the VMS tires”
 - less than the complete OpenVMS production system
 - compile/link on IA64, run on x86
 - VirtualBox, kvm
- V9.1 EAK – available to all customers; very close to complete system
- V9.2 – production release

First Boot

Porting Play Book (The Plan)

Chapter 1 – Executable Images

- **Definition:** Register Mapping, Calling Standard extensions
- **Creation:** Compilers, Assembler
- **Action:** LIBRARIAN, LINKER, INSTALL, Image Activator
- **Analysis:** SDA, DEBUG/XDELTA, ANALYZE IMAGE, ANALYZE OBJECT

Chapter 2 – Architecture-Specific Needs (a.k.a. “The 5%”)

- Booting
- Interrupts, Exceptions
- Memory Management: protection types, access modes, address space, etc.
- Atomic Instructions
- Floating Point
- Special needs for code in assembler (e.g. VAX QUEUE instruction emulation, stack walking)

Chapter 3 – Compiling and Linking Everything Else (a.k.a. “The 95%”)

- Large task but mostly mechanical
- Flush out any remaining ‘inter-routine linkage’ problems

First Boot – Mission Accomplished!

- Why was First Boot Important?
 - Identifiable point in the early life of the system
 - Good target for the engineers
 - Proof point for the customers
- Notable aspects of First Boot
 - Compilers and linker create executable code
 - Much of the new platform-specific code is being executed
 - Much compiled MACRO-32 code is being executed
 - Increased the size of some data structures
 - By default, code runs in 64-bit space

Q: What Was Different for First Boot?

A: The operating environment (not the code)

First Boot

1. All files were in the memory disk file
2. All files were built as execlets and therefore loaded into memory early during startup
3. System ran only in kernel mode
4. Booted memory disk over the network and started executing

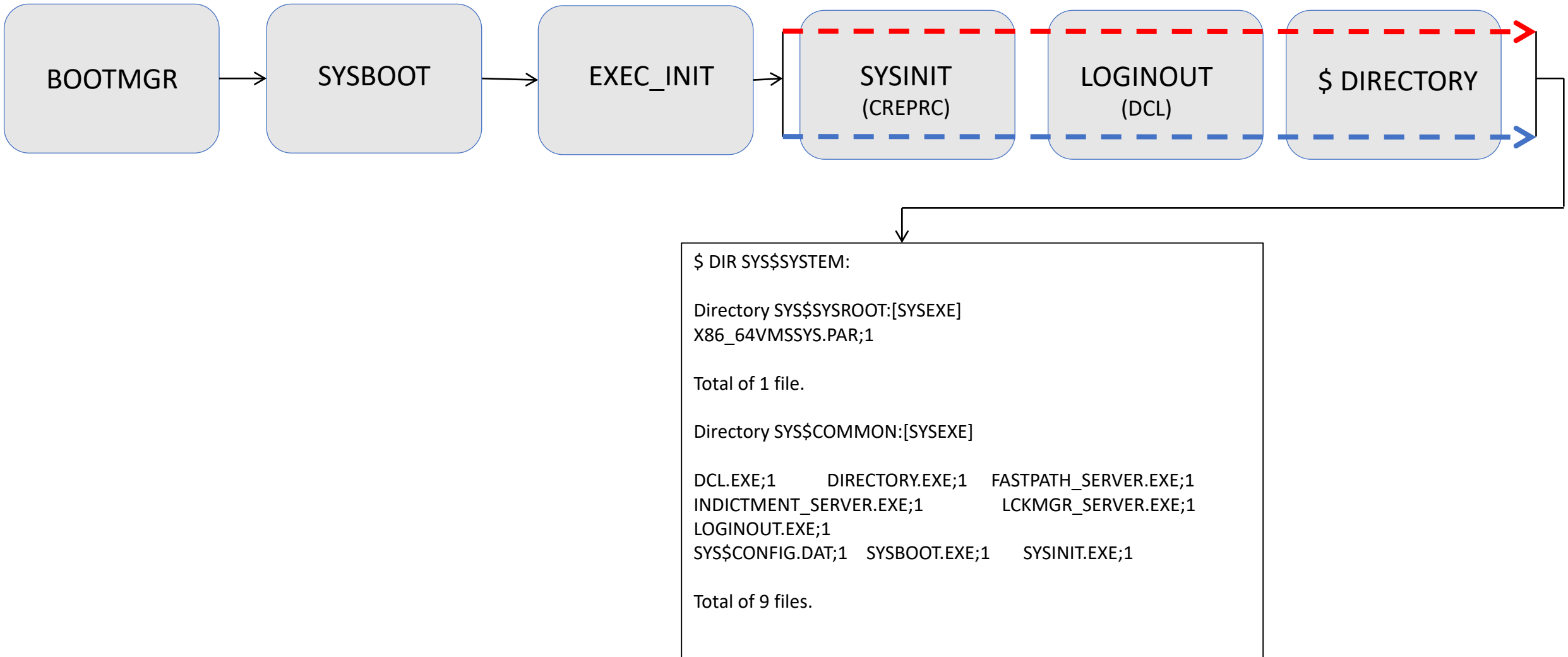
Real Boot

1. Many files are on the system disk
2. Files not in the memory disk are found and loaded when needed
3. Need to switch modes and eventually run in user mode
4. Boot from local system disk

System Startup

First Boot

Real Boot



V9.0

What is V9.0?

- VSI needs feedback from real customers doing real work.
- V9.0 will be “rough around the edges”
- Content - intersection between what a few people need to be productive and what VSI can have ready in a reasonable time
- Cross Tools Kit on IA64 for compiling and linking
- Supported on VirtualBox and kvm
- Support will be directly from the VSI engineering team
- Multiple updates prior to V9.1

V9.0 “IS NOT” (“MAYBE NOT” ?)

Would the absence of any of the following adversely affect your ability to make good use of V9.0?

- DECwindows server
- DECnet Phase IV
- DECnet Phase V (OSI)
- clusters
- volume shadowing
- reserved memory
- SMP
- XFC
- INSTALL /RESIDENT
- Support for privileged applications, for example 1) user written device drivers or 2) code that directly calls internal system routines such as those that manage page tables
- No VAX floating point support in the V9.0 cross compilers; all fp is IEEE. For V9.1 native compilers there will be VAX fp except for C++. (NOTE: It is TBD if it will ever be in C++ for x86.)

Sizing the V9.0 Proof Points

- Real Boot (L)
 - No special execlets
 - \$ DIR (output AND return to \$)
 - Boot from installed system disk
- kvm & VirtualBox booting are equivalent (M)
- Installation from 1) webserver and 2) DVD (XL)
- Crash Dumps (M), SDA (L)
- Conversational Boot (M)
- Create User Accounts (S)
- MOUNT/DISMOUNT disks (M)
- Run Batch Jobs (S)
- TCPIP: FTP, SSH (XL)
- BACKUP (M)
- User mode DEBUG (XL)
- Run a threaded (POSIX) application (XL)

S = easy, little work

M =

L =

XL = difficult, much work

Getting from First Boot to V9.0

Loaded Image List

1;L

Loaded Image List:

Seq Image Name

68	[SYS\$LDR]NT_EXTENSION	3E	ACME	1C	LOCKING
66	[SYS\$LDR]VMS_EXTENSION	3C	SYS\$MME_SERVICES	1A	PROCESS_MANAGEMENT_MON
60	SYS\$SRDRIVER	3A	SYSLDR_DYN	18	SYSDEVICE
5E	SYS\$LAN_VCITEST	38	SYS\$IPC_SERVICES	16	IO_ROUTINES_MON
5C	SYS\$LAN_CSMACD	36	MULTIPATH	14	EXCEPTION_MON
5A	SYS\$LAN	34	SYS\$UTC_SERVICES	10	SYS\$OPDRIVER
58	SYS\$EI1000X	32	SYS\$TRANSACTION_SERVICES	0E	SYSTEM_DEBUG
56	SYS\$DMDRIVER	30	SYSLICENSE	0C	SYSTEM_SYNCHRONIZATION_UNI
54	SYS\$TTDRIVER	2E	MESSAGE_ROUTINES	0A	SYSTEM_PRIMITIVES_2
52	SYS\$ISA_SUPPORT	2C	SYS\$VM	08	SYS\$ACPI
50	SYS\$PCI_SUPPORT	2A	SYSGETSYI	06	ERRORLOG
4E	<SYS\$LDR>TR\$DEBUG	28	SECURITY_MON	04	SYS\$PLATFORM_SUPPORT
4C	<SYS\$LDR>TQE\$DEBUG	26	IMAGE_MANAGEMENT	02	SYS\$BASE_IMAGE
4A	<SYS\$LDR>SYSINITX	24	RMS	00	SYS\$PUBLIC_VECTORS
48	<SYS\$LDR>SYS\$LOGINOUT	22	F11BXQP		
46	<SYS\$LDR>SYS\$DIRECTORY	20	LOGICAL_NAMES		
44	<SYS\$LDR>IO\$DEBUG	1E	SHELL8K		

New MDS Mitigation Informational

(Microarchitectural Data Sampling vulnerabilities)

Message during system startup.....

**VMS Software, Inc. OpenVMS (TM) x86_64 Operating System, XF8D-N4A
Copyright 2019 VMS Software, Inc.**

SWIS-I-MDS Mitigation active, variant haswell(HASWELL/BROADWELL)

NOTES:

- Since the mitigation will cause a performance degradation, we will provide a method for disabling the mitigation. The default will be 'enabled'.
- We will publish an estimate of the performance impact once we have a chance to do sufficient testing.

V9.0 = Real Boot and Much, Much More

- 432 individual developer tasks identified for V9.0 (approx. 40% are done)
- Eliminate “.IF DF X86_FIRST_BOOT” (and similar temporary mechanisms)
- Real Boot
 - Load all images (not just those needed for First Boot)
 - Image activation
 - Process rundown
 - Switch from Memory Disk to System Disk during startup
- Memory Management
 - Process page tables
 - Global sections
 - Adjust working sets
- Installation (via DVD or webserver)
- Exception Handling
- Run developers’ test programs
- Run Layered Products needed by participants
- Run UETP
- Run selected regression tests: I/O Hammer, etc.

Current Status

Current Status of V9.0 Proof Points

- Real Boot (L) - **DONE**
 - No special execlets
 - \$ DIR (output AND return to \$)
 - Boot from installed system disk
- kvm & VirtualBox booting are equivalent (M) - 90%
- Installation from 1) webserver and 2) DVD (XL) – **DONE**
- Crash Dumps (M), SDA (L) – 95%, 25%
- Conversational Boot (M) – **DONE**
- Create User Accounts (S) – in testing
- MOUNT/DISMOUNT disks (M) – ready to test
- Run Batch Jobs (S) - ready to test
- TCPIP: FTP, SSH (XL) – 0%
- BACKUP (M) – ready to test
- User mode DEBUG (XL) – 10%
- Run a threaded (POSIX) application (XL) – 25%

S = easy, little work

M =

L =

XL = difficult, much work

Memory Disk and System Disk

\$ directory dmm0:[sys0]

Directory DMM0:[SYS0]

SYS\$LDR.DIR;1 SYSCOMMON.DIR;1 SYSEXEC.DIR;1 SYSLIB.DIR;1

SYSMSG.DIR;1 SYSUPD.DIR;1

Total of 6 files.

\$

\$

\$ directory dkc300:[sys0]

Directory DKC300:[SYS0]

DIA\$TOOLS.DIR;1 MOM\$SYSTEM.DIR;1 SYS\$I18N.DIR;1 SYS\$LDR.DIR;1

SYS\$STARTUP.DIR;1 SYSCBI.DIR;1 SYSCOMMON.DIR;1 SYSERR.DIR;1

SYSEXEC.DIR;1 SYSHLP.DIR;1 SYSLIB.DIR;1 SYSMOINT.DIR;1

SYSMGR.DIR;1 SYSMSG.DIR;1 SYSTEST.DIR;1 SYSUPD.DIR;1

TNT.DIR;1

Total of 17 files.

\$

Crash Dump

\$

<^P here>

VSI VMS X86 XDELTA Debugger [SYSTEM_DEBUG], XFCD, Oct 23 2019 13:47:07

Brk 0 at FFFF8300.07C02660

FFFF8300.07C02660!retq (New IPL = 21) ;C <<<<<<<<< force a crash

...About to enter the Dump Kernel...

VSI VMS X86 XDELTA Debugger [SYSTEM_DEBUG], XFCD, Oct 23 2019 13:47:07

Brk 0 at FFFF8300.07C02660

FFFF8300.07C02660!retq (New IPL = 31) ;P

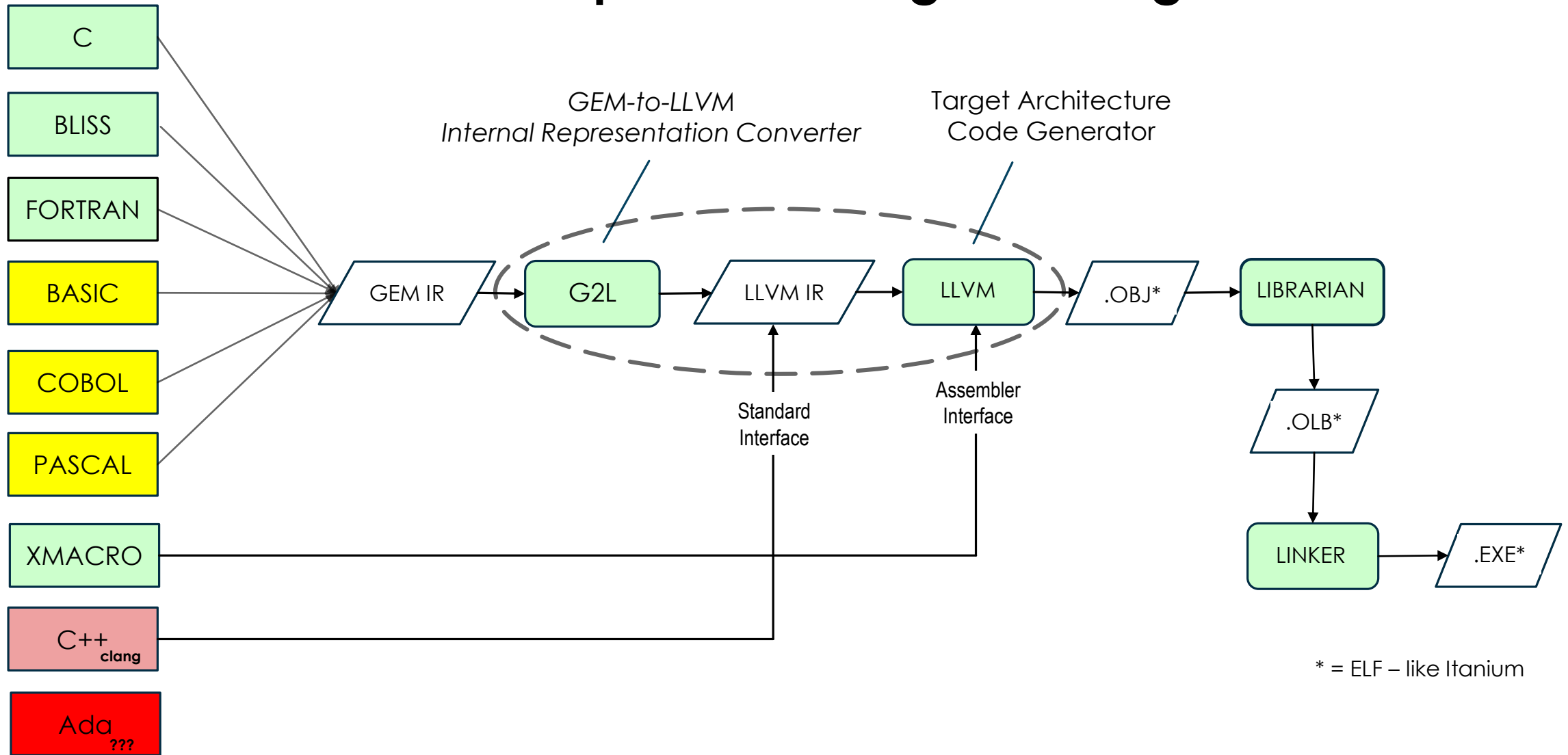
VSI Dump Kernel SYSBOOT Oct 24 2019 13:36:49

Dump Kernel started at 24-OCT-2019 13:37:43.84

**** OpenVMS x86_64 Operating System XFCD-N4A - BUGCHECK ****

** Bugcheck code = 00000964: DEBUGCRASH, Debugger forced system crash

x86-64 OpenVMS Image Building



It Should Just Work!

Frequently heard:

XYZ has no code changes for x86 so **it should just work!**

Considerations:

- Well over 3000 modules in the base OS build have conditionals. Each needs to produce the correct results for execution in V9.0 on Alpha, IA64 and x86.
- ALL executing code on x86 is being produced by the new-to-VMS backend code generator LLVM.
- All compilers except XMACRO use the new VSI GEM-to-LLVM (G2L) converter for input to LLVM.
- The LINKER has new code.
- The default is to execute in 64-bit space. (There is a switch to override this.)
- All build tools, analysis tools, and debuggers must know about the x86 Calling Standard plus a few additions.

Bootstrapping LLVM to x86 for V9.1

- Current LLVM based on V3.4.2 (June 2014 release)
- The plan
 - Apply OpenVMS specific changes to LLVM 8.0.0 sources on a Linux system
 - Build it and move objects to OpenVMS IA64
 - Use cross-linker to create new LLVM for OpenVMS x86
 - Build libraries, like libcxx, and move objects to OpenVMS Itanium
 - Cross build everything and move to x86 for native builds
- Will upgrade to newer LLVM version prior to V9.1.

Factoids

NOTE: These numbers do *not* include compilers, new TCPIP, DECnet V, layered products

- Current IA64 build – 886 images
- Current x86 build – 684 images (347 - 18 mos. ago)
- Need 92 more for V9.0
- Approx. 3000 individual module replacements, so far
 - New modules
 - Revised source modules
 - Upgraded build procedures
 - Verifying/updating conditionals (/* Verified for x86 port – John Smith */)
- First V9.0 build was 9 January 2017
- 26 modules in native assembler – most consist of a few short routines
- 1366 QTV test hours on V9.0 IA64 and Alpha last week (approx. 30% of new/rewritten x86 memory management work is common code for all platforms)

Thank You

To learn more please contact us:

vmssoftware.com

info@vmssoftware.com

[+1.978.451.0110](tel:+19784510110)