Towards Holistic Systems

Bryan Cantrill
CTO, Oxide Computer Company

Connect. Collaborate. Accelerate.

Holistic systems

- In the beginning, computing systems were tautologically **holistic**, consisting of co-designed hardware *and* software
- System software was delivered *with* the computer, and often had to be newly developed for a new machine
- Requiring new software for new hardware created schedule delays, viz. OS/360 and *The Mythical Man Month*
- With the advent of Unix, system software became portable – it could be ported rather than developed *de novo* for each new computer…

The PC era

- With the rise of the PC, however, the hardware became irreconcilably **divorced** from the system software
- Hardware enabling-software was driven into the BIOS
- The BIOS interface became what system software bound to – it became the definition of “compatibility”
- Worse, the software on both sides of the BIOS/OS divide were nearly exclusively proprietary, serving to **harden** the boundary

It gets worse: SMM

- The BIOS was not content to be inert: in order to be able implement system software functionality delivered by the hardware, *system management mode* was invented
- SMM allows effectively arbitrary code execution at arbitrary time without even allowing system software awareness
- This is the **opposite** of a holistic system: it is one that has been deliberately and perniciously divided

EFI/UEFI

- All of this might have been fine had x86 remained relegated to personal computing…
- …but Intel and AMD out-executed the RISC vendors in the 2000s, forcing PC constructs into the server space
- Starting with (ill-fated) Itanium, Intel introduced EFI in an attempt to modernize…

UEFI: What might have been

Short History of EFI

The Extensible Firmware interface (EFI) project was developed by Intel, with the initial specification released in 1999. At the time, it was designed as the means by which to boot Itanium-based systems. The original proposal for booting Itanium was the SAL (System Architectural Layer) SAL_PROC interface, with an encapsulation of the PC/AT BIOS registers as the arguments and parameters. Specifically, the means to access the disk in the SAL_PROC proposal was "SAL_PROC (0x13, 0x2, ...)", which is aligned with the PC/AT conventional BIOS call of "int13h."

Given the opportunity to clean up the boot interface, various proposals were provided. These included but were not limited to Open Firmware and Advanced RISC Computing (ARC). Ultimately, though, EFI prevailed and its architecture-neutral interface was adopted.

UEFI: What happened instead

- While its goals were laudable, UEFI was overconstrained
- In particular, the need for legacy and Windows compatibility required UEFI to support **all** past abstractions
- UEFI has become the worst of all worlds: complicated, proprietary software that remains at once isolated from – yet also still entirely entangled with! – system software
- UEFI is unquestionably **not** the way you would do it if you could do it from scratch – especially in a post-cloud and post-open source world!

Oxide's approach

- At Oxide, we are taking a from-scratch, **rack-scale** approach to server-side computing
- We have AMD Milan-based compute sleds of our own design
- We do not have a traditional BMC, but rather a fit-to-purpose service processor running our own (Rust-based, open source) operating system
- Our approach is integrated, yet open
- **Could we develop a truly holistic system on x86?**

Aside: AMD Details

- On AMD, the Platform Security Processor (PSP) is a non-architectural core that executes proprietary software to perform system initialization – including DRAM training
- System management controller (SP in our case) puts the PSP payload into SPI flash and brings the CPU out of reset
- The PSP will perform its initialization and eventually vector into host software executing on the bootstrap core (BSC)
- Historically, post-PSP initialization done by AMD's AGESA firmware – which makes a holistic system **impossible**

Challenge #1: Initialization

- To implement holistic boot, system software must perform the activities historically done by AGESA
- Modern CPUs are very complicated! Post-PSP initialization includes configuring I/O interconnects, core complexes, etc.
- For AMD Milan, this specifically includes DXIO engine configuration, NBIO PCIe strapping, hotplug configuration
- The software that has implemented this level of initialization has historically been done by the CPU vendor; these interfaces are not always documented thoroughly – if at all!

Challenge #2: Boot Phasing

- Payload that boots from PSP is size-constrained to ~13MB
- **Stage-based** approaches (e.g., oreboot + LinuxBoot) use Linux drivers to load (and execute) a production kernel
- This necessitates a pseudo-reset of the system – as well as the creation or emulation of an interface (e.g., ACPI) to pass system state to later stages
- Instead must adopt a **phase-based** approach whereby part of the system is loaded from SPI NOR and is able to load the remainder from SSDs – but the system is never discarded

Holistic booting!

- Helios is our illumos derivative that includes the Oxide bhyve-based hypervisor
- We have holistic Helios booting on our compute sleds!
- Importantly, this includes **all necessary functionality** for platform initialization (I/O, SMP, etc.)
- Currently working on phasing – but this is much lower risk than platform enablement
- Helios – along with all Oxide-authored software – will be open source when we ship our first racks later this year!

Towards Holistic Systems

- Holistic systems have clear advantages in terms of reliability, security, observability, manageability, sustainability, etc.
- Based on our experience to date, holistic systems are challenging to implement but **emphatically attainable**
- Documentation from microprocessor vendors is **essential**; they have much to gain by encouraging more enablement software on their platforms!
- Oxide may represent the first open, holistic server-side system in the post-PC x86 era – but unlikely to be the last!