Extreme-Scale Operating Systems

Rolf Riesen

23 August 2016

ROME

Legal Disclaimer

Introduction

Multi-million-core computing

- Pre-exascale systems will be large and complex
- Post-exascale system are already in the planning stages
- Usage patterns, applications, and programming paradigms have to adapt
- Same is true for OSes

Can we learn from the past to help us do that?

Extreme scale OS history

In 1996, ASCI Red at Sandia National Laboratories in Albuquerque, New Mexico was the first supercomputer to achieve teraflop performance.

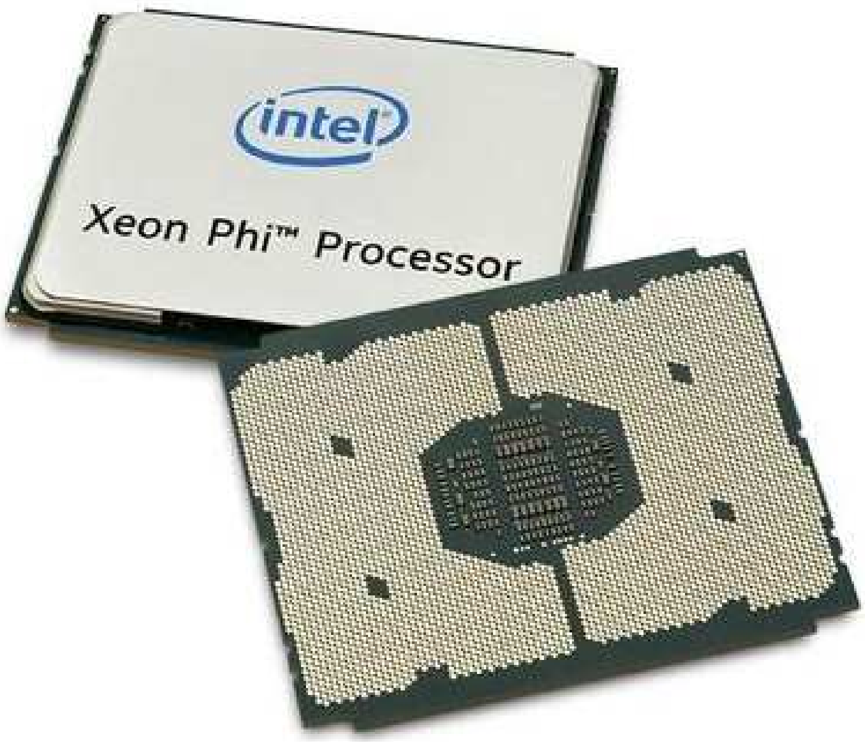
- It ran Cougar, a Lightweight kernel (LWK), on its compute nodes
- Cougar was based on SUN-MOS and Puma designed and developed by Sandia Laboratories and the University of New Mexico
- Delivering raw, scalable performance was the key goal of our efforts

Photo courtesy of Sandia National Laboratories

Present

Now we have more than 3 teraflops on a single chip!

- A modern Intel Xeon Phi processor packs the performance of former supercomputers
- Let's treat it like one!
  - Time-sharing so many resources is no longer necessary (or efficient)
  - Space share and provide resources for exclusive use by applications



An OS for extreme scale

- Future systems will be highly hierarchical
  - Systems of systems with former supercomputers as building blocks
- Machine-wide OSes to manage nodes and work flows
  - E.g., Hobbes led by Sandia National Laboratories, and Argo led by Argonne National Laboratory
- Hierarchy of OSes
- Need a highly efficient node OS (that is Linux compatible)
  - E.g., *mOS* at Intel, McKernel at RIKEN, Kitten at SNL, FFMK (L4 Linux) at TU Dresden

Why we need an LWK

LWK properties are (still) important and beneficial

- Nimbleness
  - Adapt to new, novel hardware features
  - Quickly implement new resource management strategies
  - Adapt and specialize to new programming models
- Get OS overhead out of the way; provide what hardware can do
- Simplify
  - App developers should concentrate on performance and scalability; not OS quirks and unpredictability
- Make OS research on a real system easy
  - Not a toy OS for experimentation
  - Don’t have to learn all of Linux to experiment

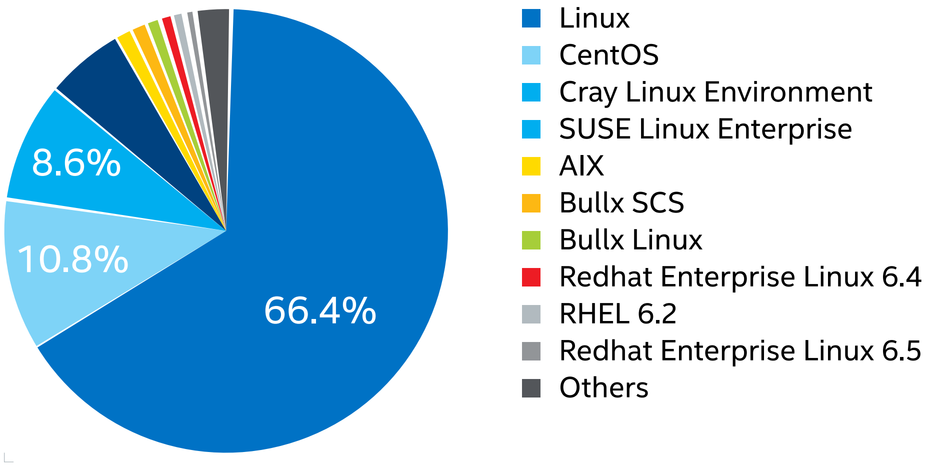
Extreme simplification

- Process management
  - Cooperative, non-preemptive task scheduling
  - Single, or few, task per logical CPU
- Memory management
  - Limited paging, no swap, “pinned” memory
  - Large pages
- Omit functionality; rely on Full Weight Kernel (FWK)
- Space sharing
  - Use massive hardware parallelism, not time sharing
- Code and binary size
  - One person can understand and remember the entire LWK

Linux dominates Top 500 list

Modern systems cannot live with an LWK alone. We also need Linux.

- Familiar to developers from their laptops and desktops
- Has the features requested by users and tool makers
  - New runtime systems and tools target Linux
  - Not just Linux system calls, also `/proc`, `/sys`, ...



OS diversity on the Nov. 2015 Top 500 list (% of all systems)

Different design goals

	LWK	FWK
Target	massively parallel systems	laptops, desktops, servers
Support	scalable applications	everything under the sun
Development environment for	parallel applications	business, games, commerce, etc.
Emphasis	efficiency	functionality
Resources	maximize use	fair sharing, QoS
Time to completion	minimal	when needed

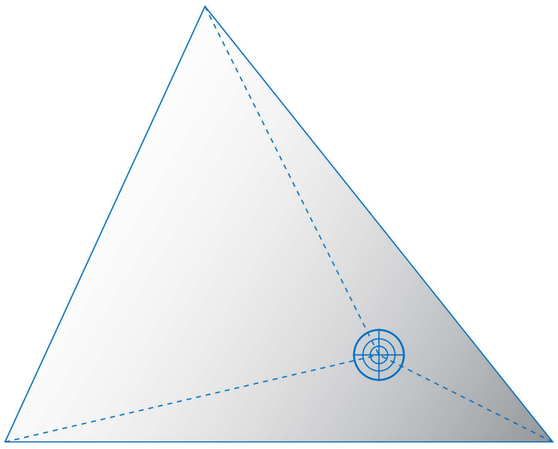
mOS Architecture and Design

mOS overview

- *mOS* (Multi-OS) is a research project at Intel
- Aimed to be the node OS for high-end HPC machines
  - Extreme scale systems: ten to hundred millions of threads
- Goal is to provide a solution beyond exa-scale
- Also, an OS that can be easily adapted to new types of hardware
  - Try out hardware ideas and quickly support them in *mOS*
- An OS that lets us provide support for new runtimes quickly
  - Future runtimes may want more control of the hardware

Top-level requirements

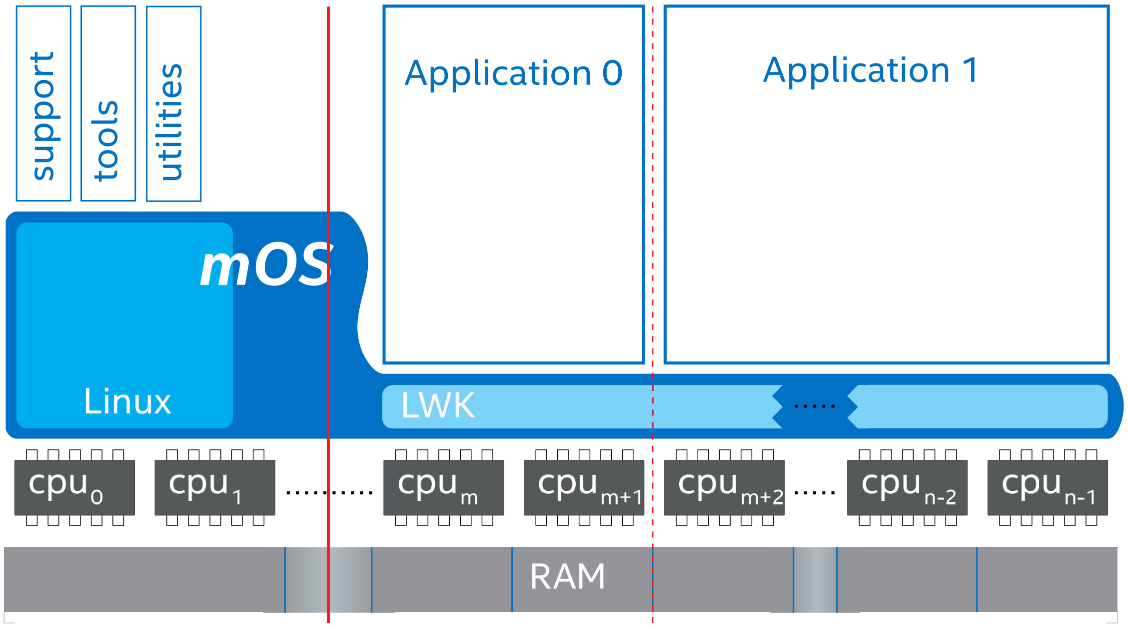
1. Foremost is for *mOS* to scale and deliver the parallel performance needed in an extreme-scale system.
2. *mOS* cannot exist unless we can implement and maintain it.
3. Linux compatibility is also important, but comes in after the performance and scalability goals have been met.



High-level architecture

To get the best of both worlds, run both OSes!

- Dedicate a few cores in a many-core system to Linux
- The remaining cores run compute intensive processes on LWK
- Service and compute partitions of ASCI Red in the past are now on one chip

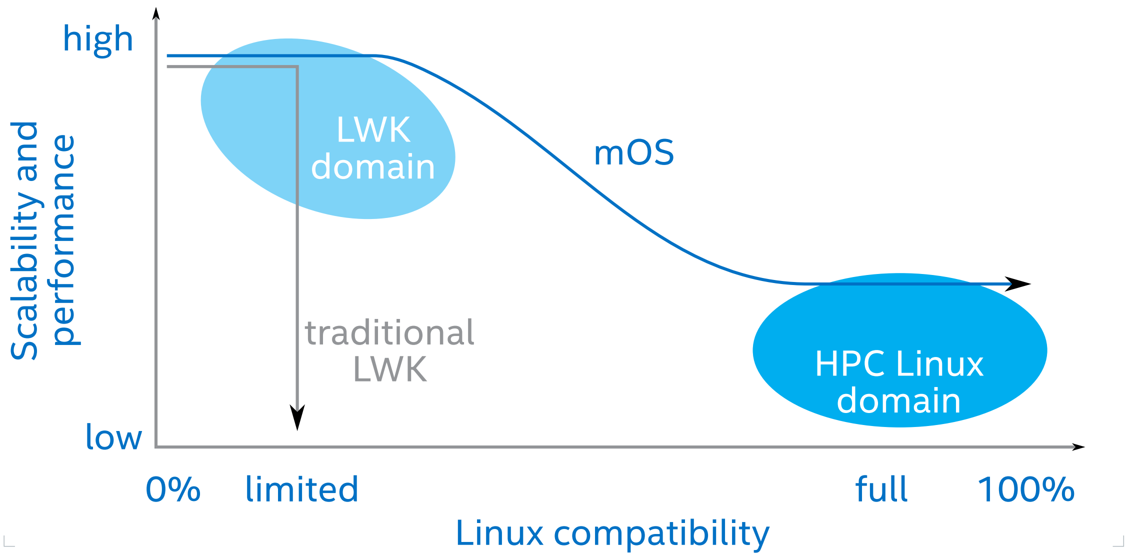


Lightweight kernels and Linux

In the past it was possible to achieve performance and scalability. Or, one could run Linux. But not both.

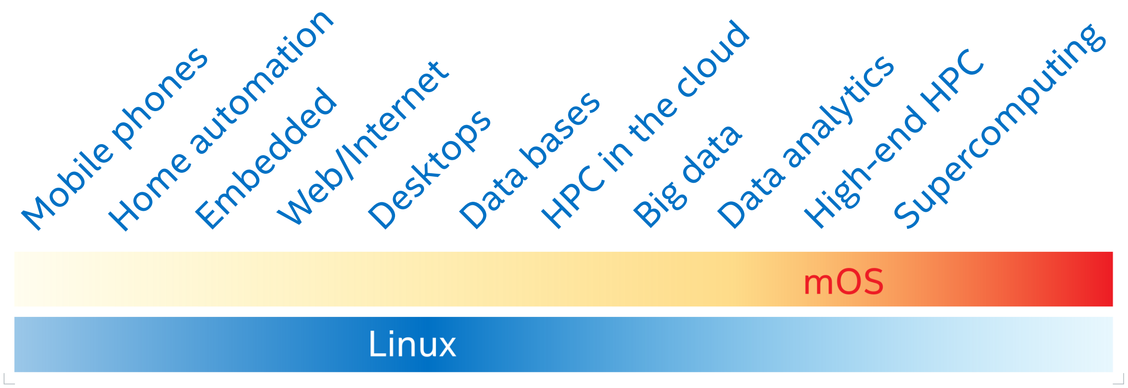
With an architecture like *mOS*, it is possible to have a more gradual path from the upper left LWK corner to the lower right FWK corner.

An application's choice of which features it wants to use, influences the overall performance and scalability.



mOS specialization

- *mOS* is not trying to be a better Linux than Linux
  - Seven of us and six months versus > 2,000 of the best developers in the world and 20 years
- *mOS* specializes for a small segment with unique requirements
- If Linux did that, it could not cover its whole spectrum



Resource management

Three stages in *mOS* resource management:

1. Resources **designated** at boot time
2. Resources **reserved** at launch time
3. Resources **allocated** at run time

Designated resources

Isolate (take away) resources from Linux

- When: Early during boot
- Why:
  - Ensure resources are available exclusively for LWK
- Example: `lwkmem=96G mos_syscall_cpus=1.2-27:29.30-55`

Reserved resources

Partition LWK designated resources among one or more HPC programs

- When: At program launch time with `yod`
- Why:
  - Prevent first program to reserve all designated resources
  - Clear indication to program what resources are available; e.g., how much memory
  - No over-commit
  - Uncertainty under Linux forces use of 80% of memory since that much is always available
- Example: `yod -M 0.5 -C 0.5 program`

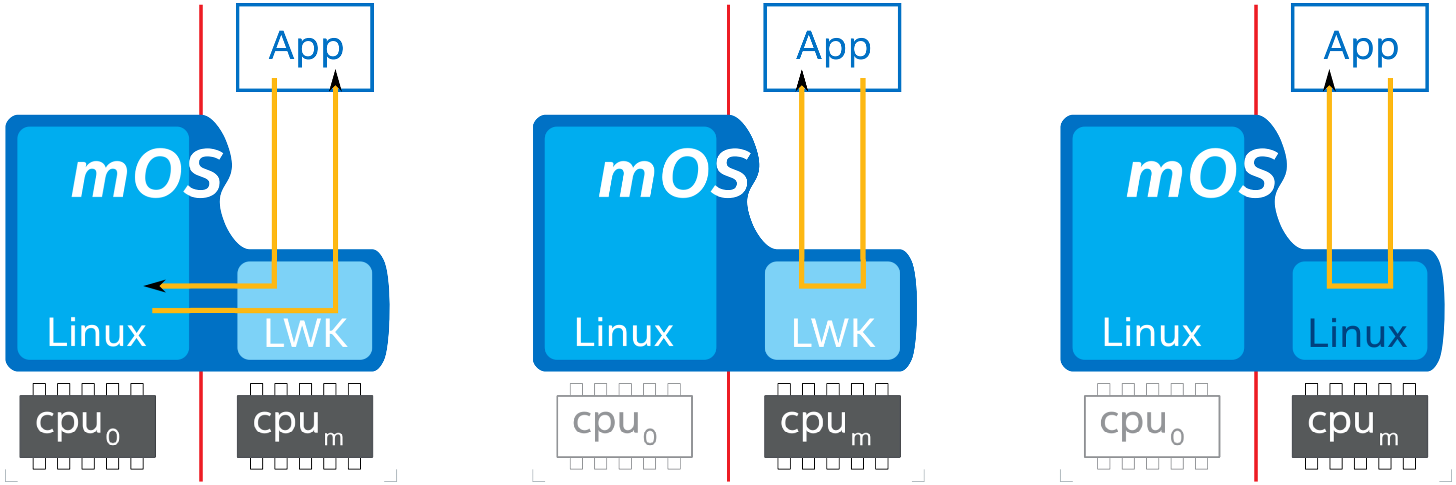
Allocated resources

Make resources available for use

- When: Allocation request; e.g., `mmap()`
- Why:
  - On return from `mmap()`, memory is already present and pinned
  - Tasks run on specific CPUs; no migration unless requested

System call locality

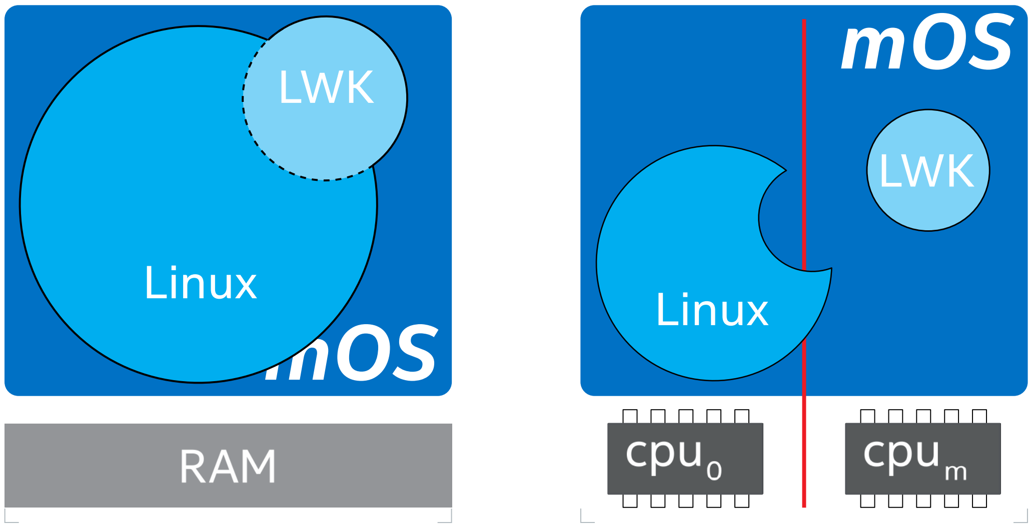
- System calls can execute locally or “remote”
- Can use Linux or LWK code



Call Linux remotely   Call LWK directly   Call Linux directly

An embedded LWK

- We're neither trimming Linux to an LWK
- Nor are we adding Linux functionality to an LWK
- We are compiling our LWK into the Linux kernel
- Then, for each logical CPU, decide which kernel has control



Code in memory

Code running

Advantages

A lot of things just work!

- Linux knows about KNL → *mOS* knows about KNL
  - At least enough to boot and run
- LWK processes are visible on the Linux side
  - Tools using `ptrace()` and `prctl()` work
  - Although we may break some things in the future as we tighten LWK side
- Linux loads binary and deals with dynamic libraries
- Machine check code; e.g., floating point exception exists and works
  - No need to port Linux code into a stand-alone LWK and maintain it
  - That is true for a lot of other code the LWK needs and Linux already has

Disadvantages

- Need more discipline to keep track of Linux
  - Continuous integration
  - Similar to keeping a device driver current
- It may be harder to support future hardware that Linux cannot deal with
  - We have not found a (reasonable) example of something like that
  - Are planning an experiment soon

Conclusion

Summary

Extreme-scale systems and their usage are sufficiently different and complex that a new look at the operating systems that orchestrate their resources is waranted.

Learning from past experiences at the leading edge teaches us that simple approaches, rather than adding complexity, lead to success.

mOS is a project at Intel that combines our experience with lightweight kernels and the need for full-weight kernel functionality.

People involved with mOS

Architecture, design, implementation, and testing:

- John Attinella
- David van Dresser
- Tom Musta
- Evan Powers
- Rolf Riesen
- Andrew Tauferner

Vision, management, and guidance:

- Todd Inglett
- Pardo Keppel
- Lance Shuler
- Robert W. Wisniewski

Collaboration with RIKEN, Japan

- Balazs Gerofi
- Yutaka Ishikawa

intel®