What is an Operating System (OS)? And does it matter anymore?

Life, the universe, and everything — one OS Researcher/Geek's perspective.

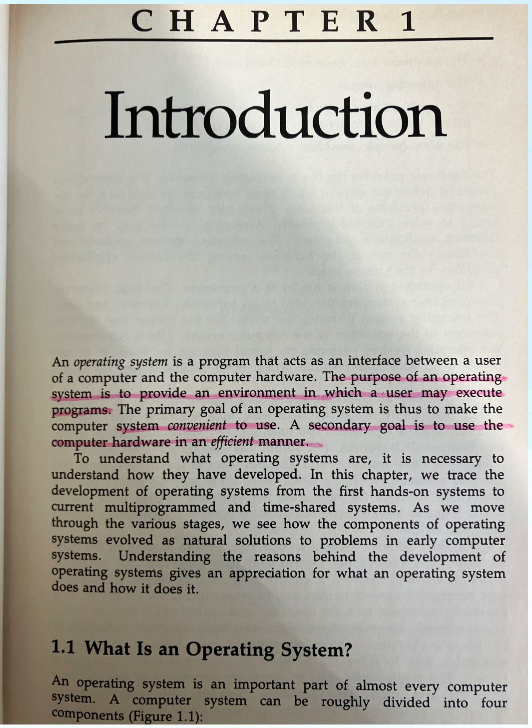
Confessions/Ramblings of an unrepentant kernel hacker

Jonathan Appavoo, Boston University and Red Hat Inc

What is an Operating System (OS)?

Textbook Answers

Silberschatz & Perterson 1988


CHAPTER 1

Introduction

An *operating system* is a program that acts as an interface between a user of a computer and the computer hardware. The purpose of an operating system is to provide an environment in which a user may execute programs. The primary goal of an operating system is thus to make the computer system *convenient* to use. A secondary goal is to use the computer hardware in an *efficient* manner.

To understand what operating systems are, it is necessary to understand how they have developed. In this chapter, we trace the development of operating systems from the first hands-on systems to current multiprogrammed and time-shared systems. As we move through the various stages, we see how the components of operating systems evolved as natural solutions to problems in early computer systems. Understanding the reasons behind the development of operating systems gives an appreciation for what an operating system does and how it does it.

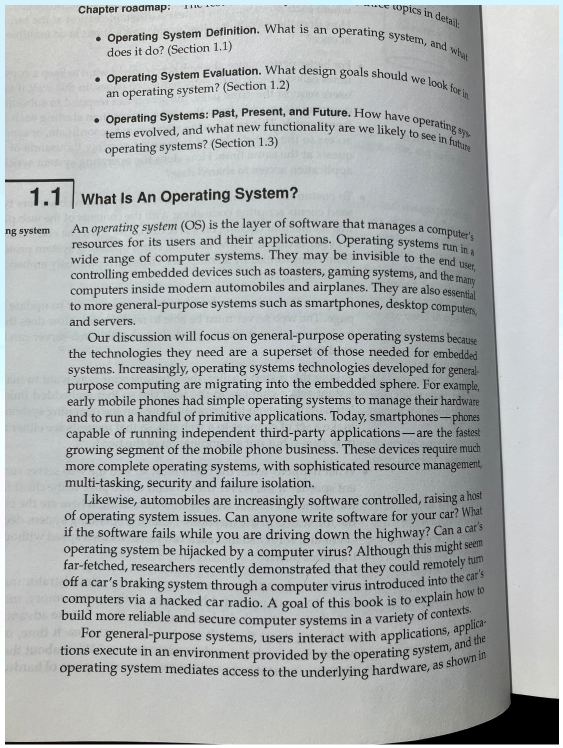
1.1 What Is an Operating System?

An operating system is an important part of almost every computer system. A computer system can be roughly divided into four components (Figure 1.1):

Goal oriented

1. "…make the computer system convenient to use."
2. "…use the computer hardware in an efficient manner."

Anderson & Dahlin 2014


Chapter roadmap:

- **Operating System Definition.** What is an operating system, and what does it do? (Section 1.1)
- **Operating System Evaluation.** What design goals should we look for in an operating system? (Section 1.2)
- **Operating Systems: Past, Present, and Future.** How have operating systems evolved, and what new functionality are we likely to see in future operating systems? (Section 1.3)

1.1 What Is An Operating System?

An *operating system* (OS) is the layer of software that manages a computer's resources for its users and their applications. Operating systems run in a wide range of computer systems. They may be invisible to the end user, controlling embedded devices such as toasters, gaming systems, and the many computers inside modern automobiles and airplanes. They are also essential to more general-purpose systems such as smartphones, desktop computers, and servers.

Our discussion will focus on general-purpose operating systems because the technologies they need are a superset of those needed for embedded systems. Increasingly, operating systems technologies developed for general-purpose computing are migrating into the embedded sphere. For example, early mobile phones had simple operating systems to manage their hardware and to run a handful of primitive applications. Today, smartphones—phones capable of running independent third-party applications—are the fastest growing segment of the mobile phone business. These devices require much more complete operating systems, with sophisticated resource management, multi-tasking, security and failure isolation.

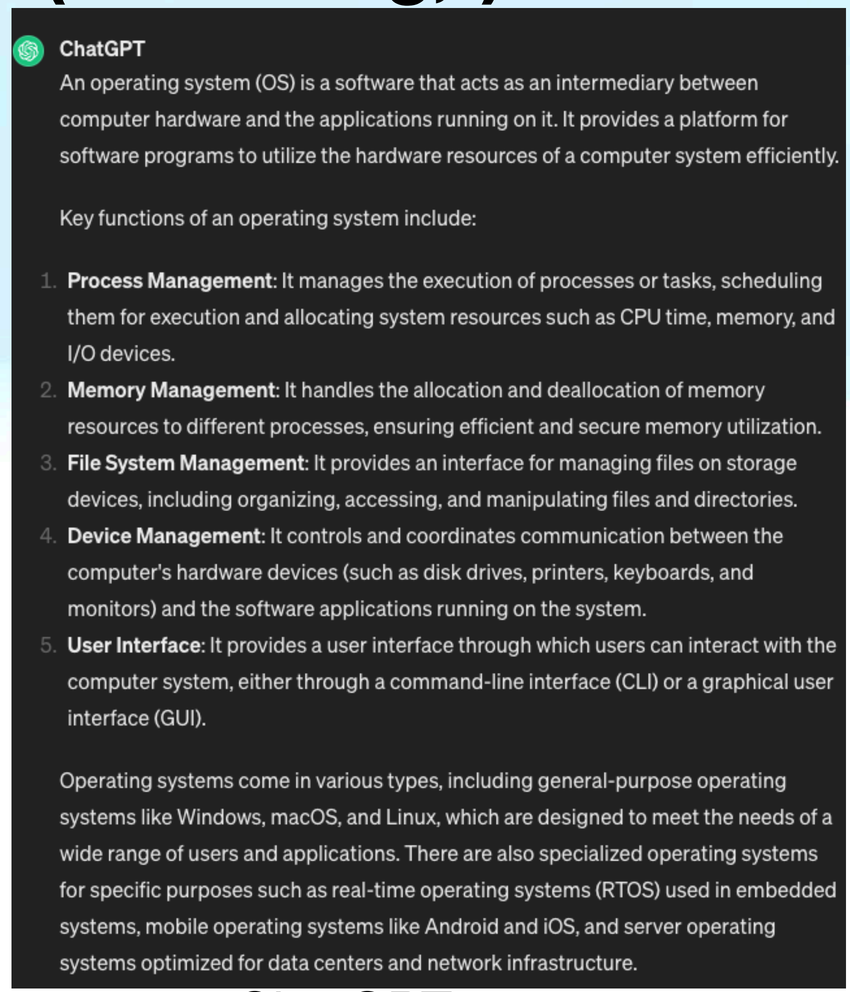
Likewise, automobiles are increasingly software controlled, raising a host of operating system issues. Can anyone write software for your car? What if the software fails while you are driving down the highway? Can a car's operating system be hijacked by a computer virus? Although this might seem far-fetched, researchers recently demonstrated that they could remotely turn off a car's braking system through a computer virus introduced into the car's computers via a hacked car radio. A goal of this book is to explain how to build more reliable and secure computer systems in a variety of contexts.

For general-purpose systems, users interact with applications, applications execute in an environment provided by the operating system, and the operating system mediates access to the underlying hardware, as shown in

Functionality oriented

"Manages a computer's resources"

The Modern textbook (marketing;-) Answer?


ChatGPT

An operating system (OS) is a software that acts as an intermediary between computer hardware and the applications running on it. It provides a platform for software programs to utilize the hardware resources of a computer system efficiently.

Key functions of an operating system include:

1. **Process Management**: It manages the execution of processes or tasks, scheduling them for execution and allocating system resources such as CPU time, memory, and I/O devices.
2. **Memory Management**: It handles the allocation and deallocation of memory resources to different processes, ensuring efficient and secure memory utilization.
3. **File System Management**: It provides an interface for managing files on storage devices, including organizing, accessing, and manipulating files and directories.
4. **Device Management**: It controls and coordinates communication between the computer's hardware devices (such as disk drives, printers, keyboards, and monitors) and the software applications running on the system.
5. **User Interface**: It provides a user interface through which users can interact with the computer system, either through a command-line interface (CLI) or a graphical user interface (GUI).

Operating systems come in various types, including general-purpose operating systems like Windows, macOS, and Linux, which are designed to meet the needs of a wide range of users and applications. There are also specialized operating systems for specific purposes such as real-time operating systems (RTOS) used in embedded systems, mobile operating systems like Android and iOS, and server operating systems optimized for data centers and network infrastructure.

ChatGPT 3.5 2024

And does it matter anymore?

Matter: verb

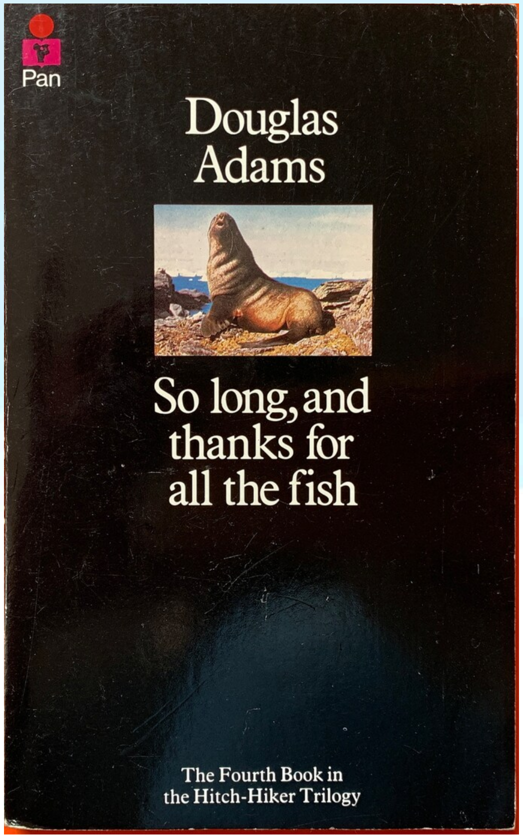
1. be of importance; have significance.

“it doesn't matter what the guests wear”

(oxford dictionary)

After all:

- Application development and deployment are stable and no longer depend on the OS interface.
  - e.g., JAVA, Python, JavaScript, WASM, etc
- AI Apps don’t need or want OS management. They want the GPUs all to themselves—direct hardware access. The **NO** OS might be the best OS.
- There is so much hardware in the cloud that maybe we should just run the apps on dedicated bare-metal nodes



So, maybe there is no real point in doing OS research. We need to recognize that the OS is a technology in maintenance mode or even in the process of becoming obsolete.

As you might have guessed, I think the OS matters, and there is room for innovation, but we need to unwind to a simpler, more progressive view of it.

A collection of "software" that makes it "**easier**" to "**use**" a "**computer.**"

Ultimately Hardware

Get stuff done with less effort, in less time, and fewer resources

Two questions to focus on

Linux, both the kernel and its massive collection of user software, does its job well!   **A familiar, well-understood environment that makes it easy to use a computer.** But what a computer is has changed and continues to.

1. How can we preserve Linux and ensure that we don't erode it by unnecessarily burdening it?

2. And yet, can we evolve the OS in light of foundational changes in hardware and usage?

Evolving the OS by studying Hardware Scale and Elasticity

1. Hurricane, Tornado and K42
2. Libra
3. Kittyhawk
4. EbbRT*

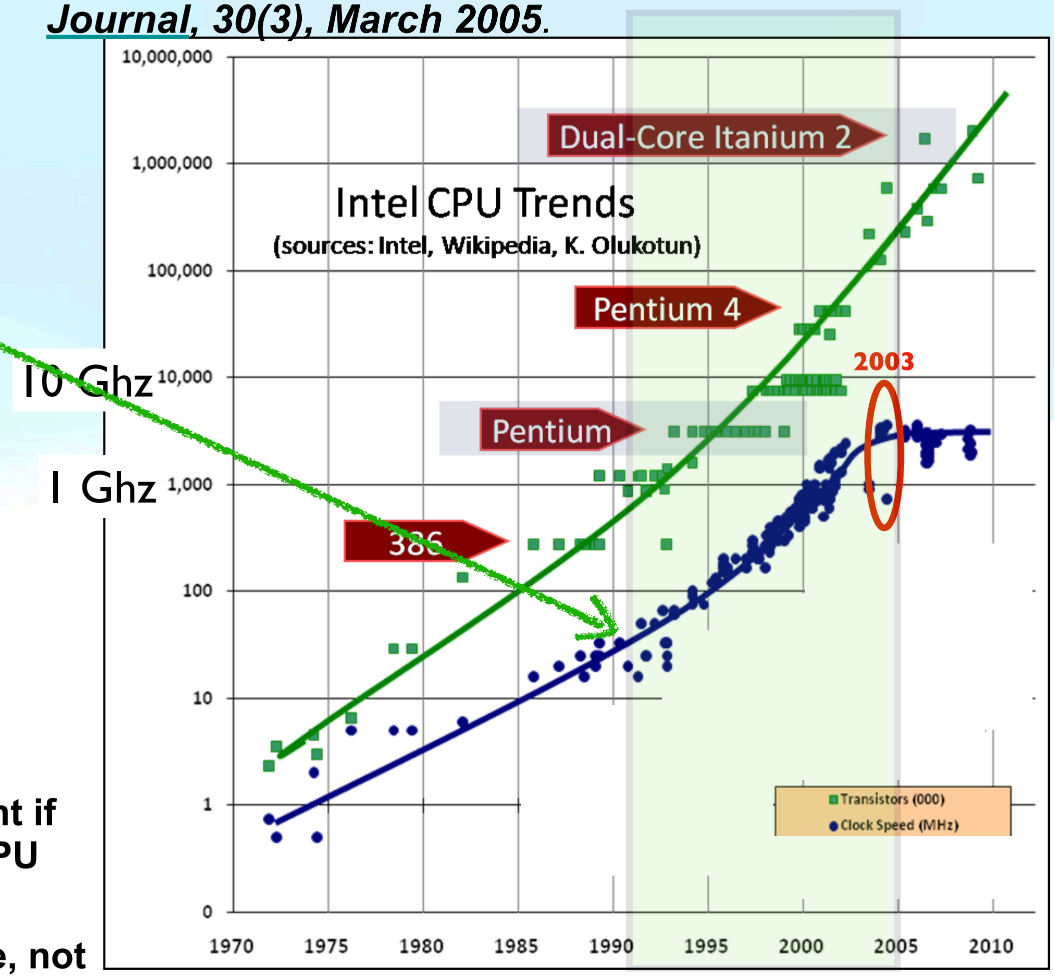
“computer.”

Hurricane, Tornado, and K42 1991 - 2005

"The Free Lunch Is Over A Fundamental Turn Toward Concurrency in Software", *Dr. Dobb's Journal, 30(3), March 2005.*

Scalable 64-bit SMMPs are coming; what should the OS look like?

(Pay attention to the dates. Notice when the research was started and conducted)



The original data was to Dec 2004 with forecasts to 2007 (updated in 2009 with forecasts to 2010)

2005 mainstream article states,

"Applications will increasingly need to be concurrent if they want to fully exploit continuing exponential CPU throughput gains

Efficiency and performance optimization will get more, not less, important"

Hurricane, Tornado, and K42

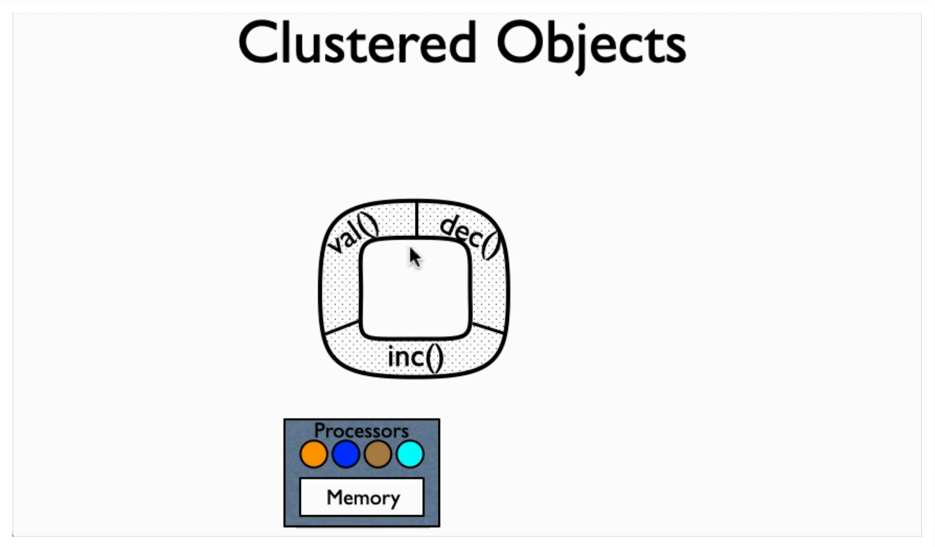
The seeds towards understanding Elasticity and its value.

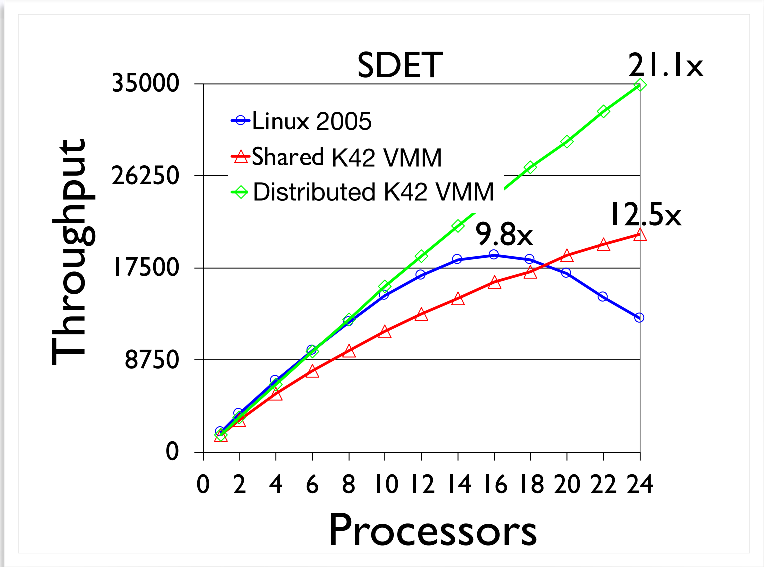
- Premises (there were others, but this is what we will focus on):
  - We know how to build modular, scalable hardware systems out of general-purpose processors:
    - NUMAchine: Shared Memory Multi-Processor incrementally scalable to 100's of processors
  - OS kernels should be designed and implemented to adapt and reflect the structure of the hardware and application demand (Eg. standard locking is insufficient)
    - One kernel binary should automatically adapt to the size of the system and the number of threads in the running application.

K42: from scratch, OS can be worth it.

Start with the right primitives.

- Build correct primitives first, then write the OS (per-core ipc, memory allocator, RCU — Elastic Object Model)
- Decomposed kernel into objects
- Made it easy to exploit per-core memory consistently
- **Reflected and programmed for Elasticity**
- **Enabled and programmed with Specialization**





Journal of Supercomputing, 1995.

Hierarchical Clustering: A Structure for Scalable Multiprocessor Operating System Design

Ron Unrau, Orran Krieger, Benjamin Gamsa, Michael Stumm
Department of Electrical and Computer Engineering
University of Toronto
Toronto, Canada M5S 1A4

Abstract

We introduce the concept of Hierarchical Clustering as a way to structure shared memory multiprocessor operating systems for scalability. As the name implies, the concept is based on clustering and hierarchical system design. Hierarchical Clustering leads to a modular system, composed of easy-to-design and efficient building blocks. The resulting structure is scalable because it *i*) maximizes locality, which is key to good performance in NUMA systems, and *ii*) provides for concurrency that increases linearly with the number of processors. At the same time, there is tight coupling within a cluster, so the system performs well for local interactions which are expected to constitute the common case. A clustered system can easily be adapted to different hardware configurations and architectures by changing the size of the clusters.

We show how this structuring technique is applied to the design of a microkernel-based operating system called HURRICANE. This prototype system is the first complete and running implementation of its kind, and demonstrates the feasibility of a hierarchically clustered system. We present performance results based on the prototype, demonstrating the characteristics and behavior of a clustered system. In particular, we show how clustering trades off the efficiencies of tight coupling for the advantages of replication, increased locality, and decreased lock contention. We describe some of the lessons we learned from our implementation efforts and close with a discussion of our future work.

1 Introduction

Considerable attention has been directed towards designing "scalable" shared-memory multiprocessor hardware, capable of accommodating a large number of processors. These efforts have been successful to the extent that an increasing number of such systems exist [1, 8, 19, 28, 30, 32]. However, scalable hardware can only be fully exploited and cost effective for general purpose use if there exists an operating system that is as scalable as the hardware.

An operating system targeting large-scale multiprocessors must consider both concurrency and locality. However, existing multiprocessor operating systems have been scaled to accommodate many processors only in an *ad hoc* manner, by repeatedly identifying and then removing the most contended bottlenecks, thus addressing concurrency issues but not locality issues. Bottlenecks are removed either by splitting existing locks, or by replacing existing data structures with more elaborate, but concurrent ones. The process can be long and tedious, and results in systems that 1) have a large number of locks that need to be held for common operations, with correspondingly large overhead, 2) exhibit little locality, and 3) are not scalable in a generic sense, but only until the next bottleneck is encountered [4, 5, 11, 12, 23]. Porting an existing system designed for networked distributed systems is also unsatisfactory, because of the large communication requirements

The following paper was originally published in the
Proceedings of the 3rd Symposium on Operating Systems Design and Implementation
New Orleans, Louisiana, February, 1999

Tornado: Maximizing Locality and Concurrency in a Shared Memory Multiprocessor Operating System

Ben Gamsa
University of Toronto
Orran Krieger
IBM T.J. Watson Research Center
Jonathan Appavoo, Michael Stumm
University of Toronto

K42: Building a Complete Operating System

Orran Krieger † Marc Auslander † Bryan Rosenburg † Robert W. Wisniewski †
Jimi Xenidis † Dilma Da Silva † Michal Ostrowski † Jonathan Appavoo †
Maria Butrico † Mark Mergen † Amos Waterland † Volkmar Uhlig †

ABSTRACT

K42 is one of the few recent research projects that is examining operating system design structure issues in the context of new whole-system design. K42 is open source and was designed from the ground up to perform well and to be scalable, customizable, and maintainable. The project was begun in 1996 by a team at IBM Research. Over the last nine years there has been a development effort on K42 from between six to twenty researchers and developers across IBM, collaborating universities, and national laboratories. K42 supports the Linux API and ABI, and is able to run unmodified Linux applications and libraries. The approach we took in K42 to achieve scalability and customizability has been successful.

The project has produced positive research results, has resulted in contributions to Linux and the Xen hypervisor on Power, and continues to be a rich platform for exploring system software technology. Today, K42, is one of the key exploratory platforms in the DOE's FAST-OS program, is being used as a prototyping vehicle in IBM's PERCS project, and is being used by universities and national labs for exploratory research . In this paper, we provide insight into building an entire system by discussing the motivation and history of K42, describing its fundamental technologies, and presenting an overview of the research directions we have been pursuing.

Categories and Subject Descriptors

D.4.0 [**Operating Systems**]: General; D.4.1 [**Operating Systems**]: Process Managment; D.4.1 [**Operating Systems**]: Process Managment—*Multiprocessing*; D.4.2 [**Operating Systems**]: Storage Management; D.4.3 [**Operating Systems**]: File Systems Management; D.4.4 [**Operating Systems**]: Communications Management; D.4.7 [**Operating Systems**]: Oragnization and Design; D.4.8 [**Operating Systems**]: Performance

General Terms

Algorithms,Performance,Design

Keywords

operating system design,scalable operating systems, customizable operating systems

1. BACKGROUND

In 1996 we began K42 to explore a new operating system design structure for scalability, customizability, and maintainability in the context of large-scope or whole-system research issues. K42's design was based on current software and hardware technology and on predictions of where those technologies were headed. In this section we describe those predictions and discuss the resulting technology decisions. At the end of the paper, we review how the predictions changed over the life of the project and the resulting changes in the technical directions of the project.

1.1 Technology predictions

Key predictions we made in 1996 were:

1. *Microsoft Windows would dominate the client space, and would increasingly dominate server systems.* By the mid 1990s, predictions made by leading consulting firms indicated Unix would disappear from all but high-end servers and Windows would dominate most markets.
2. *Multiprocessors would become increasingly important at both the high and low end.* For the high end, projects in academia [24, 1, 20] demonstrated that large scale NUMA multiprocessors are feasible and can be developed to be price/performance competitive with distributed systems. For the low end, the increasing number of transistors were yielding smaller improvements to single cores, and it seemed that the ever increasing density of transistors would instead be used for more cores and threads on a chip.
3. *The cost of maintaining and enhancing current operating systems would grow increasingly prohibitive over time.* Existing operating systems were designed as monolithic systems, with data structures and policy implementations spread across the system. Such global

†IBM T. J. Watson Research Center
This work was supported in part by a DARPA PERCS grant contract number NBCH30390004

Experience Distributing Objects in an SMMP OS

JONATHAN APPAVOO, DILMA DA SILVA, ORRAN KRIEGER, MARC AUSLANDER, MICHAL OSTROWSKI, BRYAN ROSENBURG, AMOS WATERLAND, ROBERT W. WISNIEWSKI, and JIMI XENIDIS
IBM T.J. Watson Research Center
and
MICHAEL STUMM and LIVIO SOARES
Dept. of Electrical and Computer Engineering, University of Toronto

Designing and implementing system software so that it scales well on shared-memory multiprocessors (SMMPs) has proven to be surprisingly challenging. To improve scalability, most designers to date have focused on concurrency by iteratively eliminating the need for locks and reducing lock contention. However, our experience indicates that locality is just as, if not more, important and that focusing on locality ultimately leads to a more scalable system.

In this paper, we describe a methodology and a framework for constructing system software structured for locality, exploiting techniques similar to those used in distributed systems. Specifically, we found two techniques to be effective in improving scalability of SMMP operating systems: (*i*) an object-oriented structure that minimizes sharing by providing a natural mapping from independent requests to independent code paths and data structures, and (*ii*) the selective partitioning, distribution, and replication of object implementations in order to improve locality. We describe concrete examples of distributed objects and our experience implementing them. We demonstrate that the distributed implementations improve the scalability of operating-system-intensive parallel workloads.

Categories and Subject Descriptors: D.4.7 [**Operating Systems**]: Organization and Design

General Terms: Design

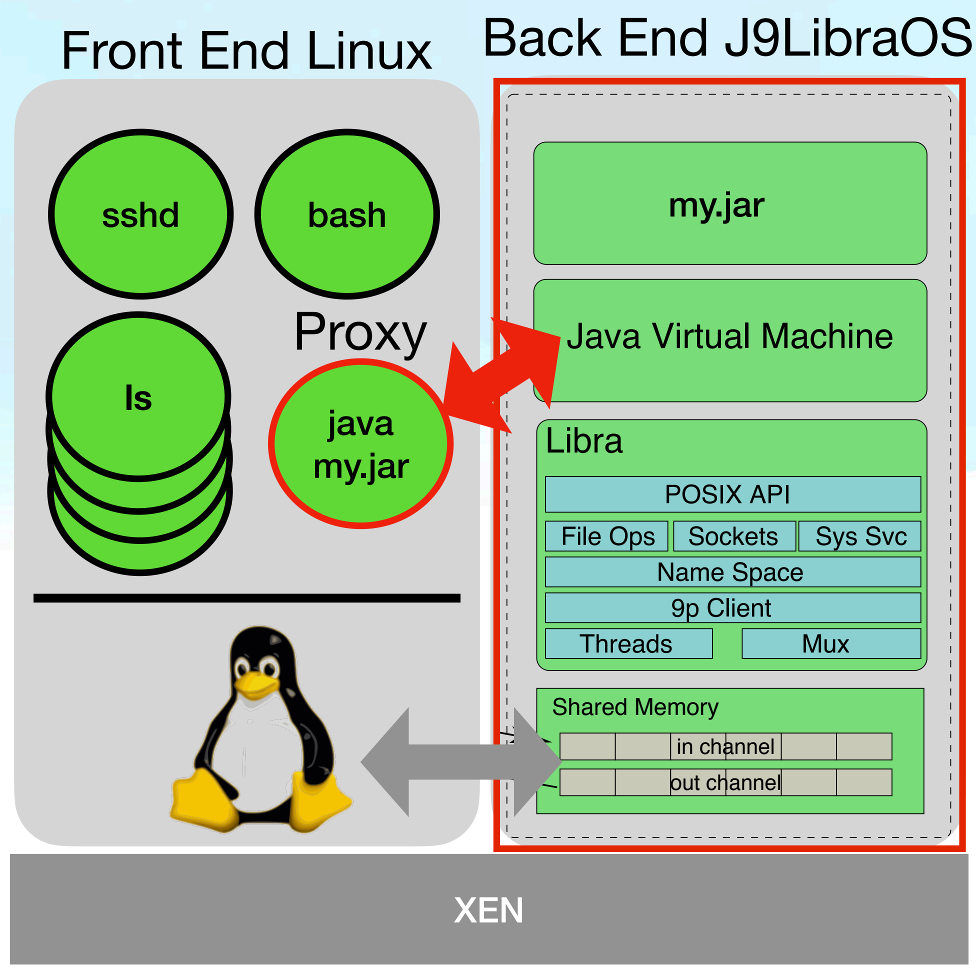
Additional Key Words and Phrases: Locality, Concurrency, Distribution, Scalability SMMP

ACM Reference Format:

Appavoo, J., da Silva, D., Krieger, O., Auslander, M., Ostrowski, M., Rosenburg, B., Waterland, A., Wisniewski, R. W., Xenidis, J., Stumm, M., and Soares L. Experience distributing objects

Authors' addresses: J. Appavoo, D. da Silva, O. Krieger, M. Auslander, M. Ostrowski, B. Rosenburg, A. Waterland, R. W. Wisniewski, J. Xenidis; e-mail: {jappavoo,dilmasilva,okrieg,Marc_Auslander, mostrows,rosnbrg,apw,bobww,jimix}@us.ibm.com; M. Stumm and L. Soares: Department of Electrical and Computer Engineering, University of Toronto; e-mail: {stumm,livio}@eecg.toronto.edu

Libra: A Library Operating System (UniKernel) for a JVM in a Virtualized Execution Environment (2005-2006 One year effort)



A key insight —
A hybrid OS relationship is compelling when leveraging an elastic pool of hardware.

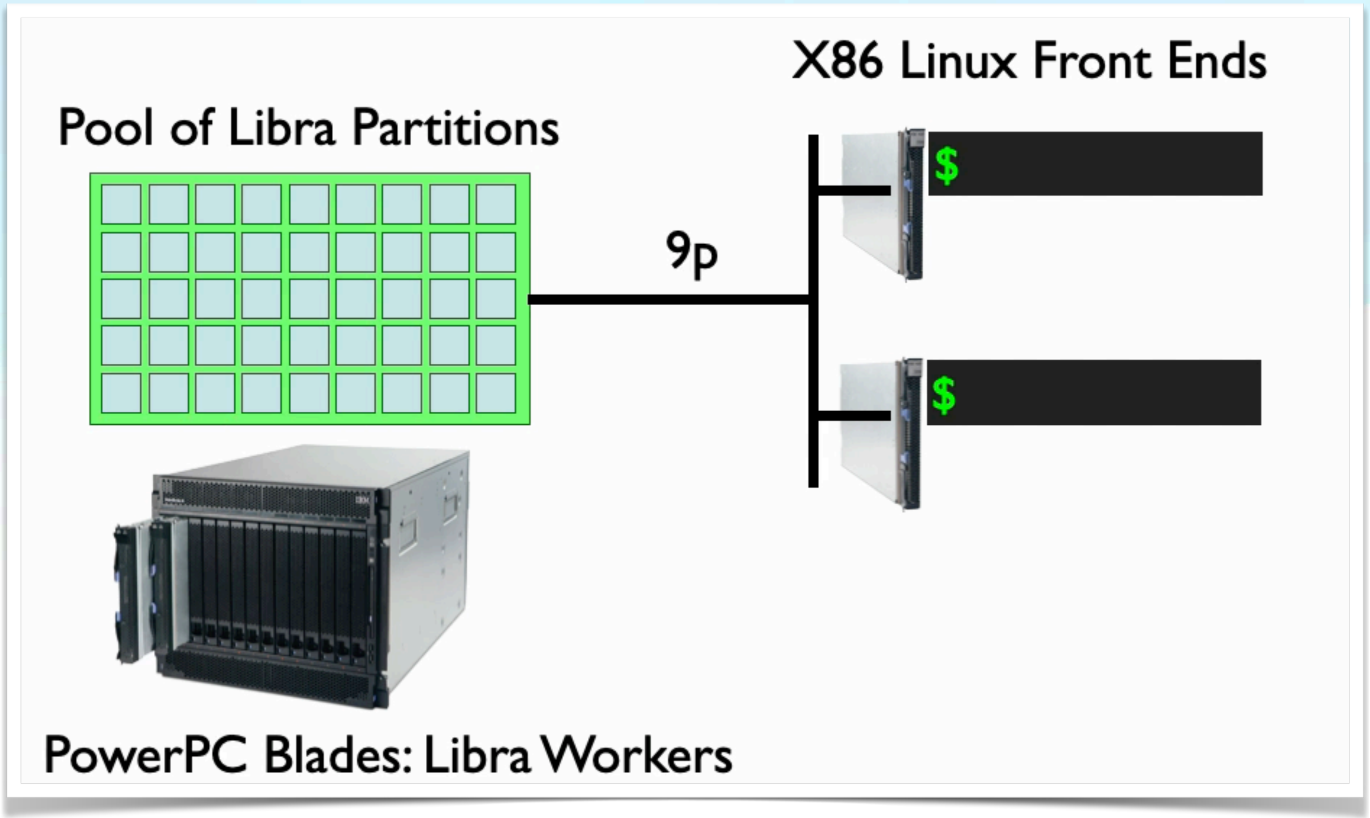
General Purpose + Special Purpose
=
Something Cool - An Accelerator Model!

Neither has to be perfect, and they can get help from each other.

Takes the pressure off Linux and eases UniKernel Functionality

More than just a UniKernel : A Platform for

- Architecture Heterogeneity
- Optimization via Specialization
- Seamless integration
- Hardware Elasticity



Work was done in 2005-2006,
Paper published in 2007

Libra: A Library Operating System for a JVM in a Virtualized Execution Environment

Glenn Ammons
IBM T.J. Watson Research Center
ammons@us.ibm.com

Jonathan Appavoo
IBM T.J. Watson Research Center
jappavoo@us.ibm.com

Maria Butrico
IBM T.J. Watson Research Center
butrico@us.ibm.com

Dilma Da Silva
IBM T.J. Watson Research Center
dilmasilva@us.ibm.com

David Grove
IBM T.J. Watson Research Center
groved@us.ibm.com

Kiyokuni Kawachiya
IBM Tokyo Research Laboratory
kawatiya@jp.ibm.com

Orran Krieger
IBM T.J. Watson Research Center
okrieg@us.ibm.com

Byran Rosenburg
IBM T.J. Watson Research Center
rosnbrg@us.ibm.com

Eric Van Hensbergen
IBM Austin Research Laboratory
ericvanhensbergen@us.ibm.com

Robert W. Wisniewski
IBM T.J. Watson Research Center
bobww@us.ibm.com

Abstract

If the operating system could be specialized for every application, many applications would run faster. For example, Java virtual machines (JVMs) provide their own threading model and memory protection, so general-purpose operating system implementations of these abstractions are redundant. However, traditional means of transforming existing systems into specialized systems are difficult to adopt because they require replacing the entire operating system.

This paper describes Libra, an execution environment specialized for IBM's J9 JVM. Libra does not replace the entire operating system. Instead, Libra and J9 form a single statically-linked image that runs in a hypervisor partition. Libra provides the services necessary to achieve good performance for the Java workloads of interest but relies on an instance of Linux in another hypervisor partition to provide a networking stack, a filesystem, and other services. The expense of remote calls is offset by the fact that Libra's services can be customized for a particular workload; for example, on the Nutch search engine, we show that two simple customizations improve application throughput by a factor of 2.7.

Categories and Subject Descriptors D.3.4 [*Processors*]: Runtime Environments; D.4.7 [*Operating Systems*]: Organization and Design; D.4.8 [*Operating Systems*]: Performance

General Terms Design, Experimentation, Performance

Keywords Virtualization, exokernels, Xen, JVM

VEE '07 June 13–15, 2007, San Diego, California, USA.

1. Introduction

This paper describes a new way to transform existing software systems into high-performance, specialized systems. Our method relies on hypervisors [12, 14], which are becoming efficient and widely available, and on the 9P distributed filesystem protocol [30, 32].

Our approach is similar to the exokernel approach [25]. An exokernel system divides the general-purpose operating system into two parts: a small, trusted kernel (called the exokernel) that securely multiplexes hardware resources such as processors and disk blocks, and a collection of unprivileged libraries (called "library operating systems" or "libOSes") that provide operating system abstractions such as filesystems and processes. Ideally, each application tailors the abstractions to its needs and pays only for what it uses. For example, the distributed search application Nutch [8, 9] needs a Java virtual machine, access to a read-only store, and a simple networking stack; Section 5.3 shows that simple implementations of these abstractions achieve good performance.

Unfortunately, exokernels are difficult to adopt, because migrating an existing application to an exokernel system requires porting the operating system on which it relies. For example, to run unmodified UNIX programs on their exokernel, Kaashoek and others wrote ExOS, a library that implements many of the BSD 4.4 abstractions [25]. Writing such a library is a significant effort. Also, because the library is a reimplementation of the operating system, the only way to take advantage of improvements to the operating system is to port them to the library.

Our system, Libra,[1] avoids these problems by casting a hypervisor (specifically, Xen [5]) in the role of the exokernel. Figure 1 depicts the overall architecture of Libra. Unlike traditional exoker-

[1] We chose the name "Libra" because our goal of providing well-balanced services aligns with the imagery associated with the constellation of the same name.

Kittyhawk Monsterous Scale and Elasticity are coming 2006-2008

When something gets so big, you have to change your thinking.

Old but cool demo videos: https://www.cs.bu.edu/~jappavoo/Resources/kittyhawk/kittyhawk/Demos.html

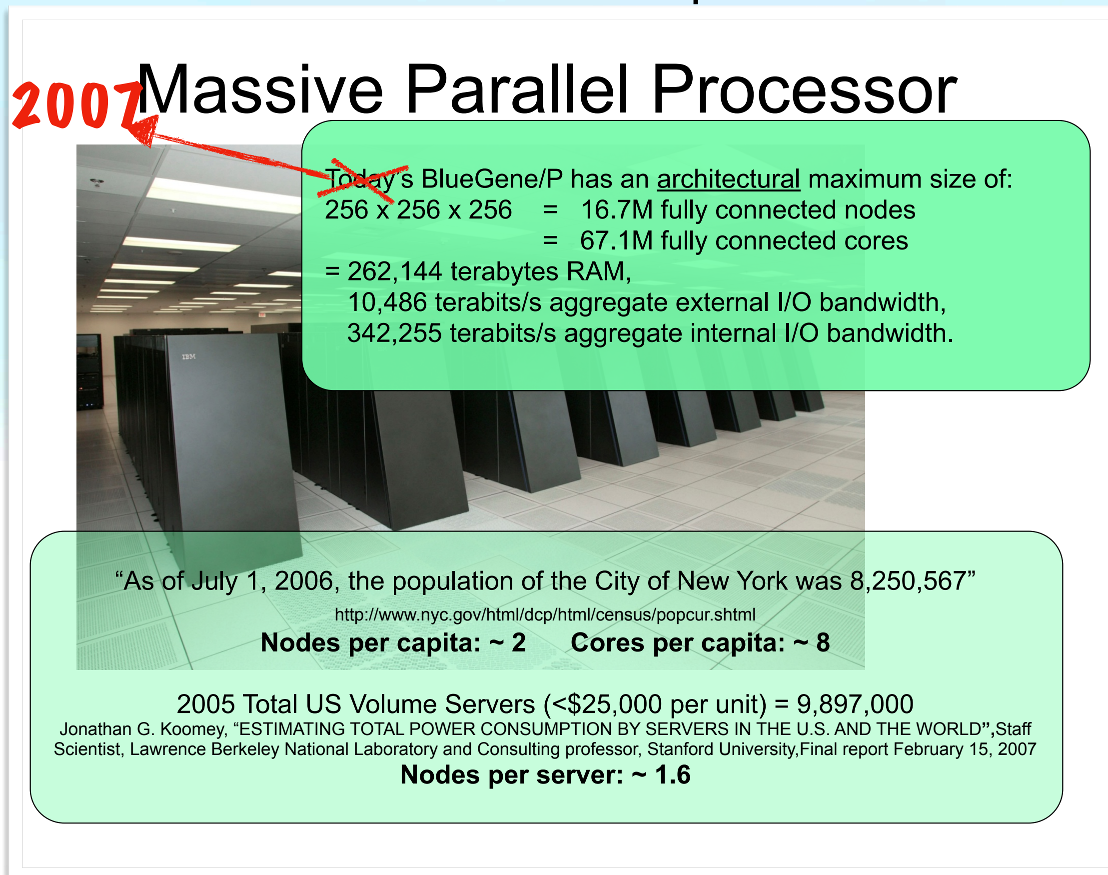
Use Chrome to view them.

Hardware like we had never seen before

Machine truly designed for the data center!
Not cobbled together with duck tape.

- Massively integrated architecture potentially MILLIONS of bootable nodes
- SOC integrates general-purpose cores, accelerators, and interconnect routers
- Supports incremental growth
- Interconnects are more like Buses than a commodity Network of the time (physical addressing and RDMA)

It turned out it was not esoteric just needed some SW to expose that fact.



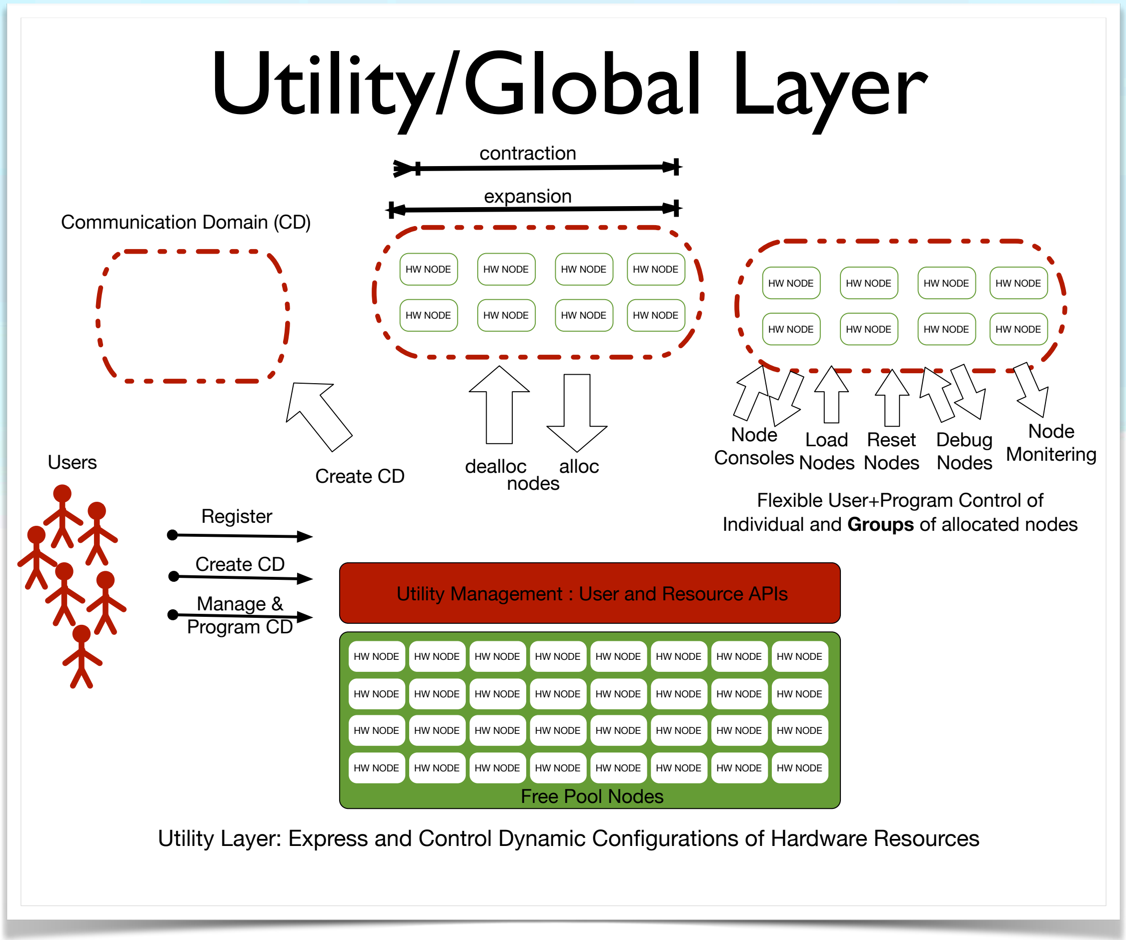
Operating systems should evolve to reflect the HW capabilities.

Decomposing the OS

“disposable” HW

- OS: expose the machine's Elasticity and Scale in a way that makes it easy to use.
- SW? when “computers” can be “malloced” and “freed” almost as quickly as threads.
- The first OS “layer”: Fast, Scalable and Programmable “utility” interface for metered units of raw HW
- Second Layer: bootloaders, virtualization, OSes, and apps, are add-ons.
- Start with **existing** software is critical - incremental development of new “elastic” stacks

Our conjecture: supercomputer-like systems would become the dominant utility (cloud) computer.



If you squint, you can almost see ESI.

Media and "users"

DAILYTECH

Home | Auto | Gadgets | Hardware | Internet | IT | Science | Software | Blogs

Suggest News

Internet

IBM Proposes One Computer to Run Entire Internet

Jason Mick (Blog) - February 8, 2008 1:44 PM

135 comment(s) - last by phxfreddy.. on Feb 27 at 3:58 PM

Dreaming big; IBM looks to host entire internet on a single modified Blue Gene supercomputer

Thomas Watson, founder of IBM, is oft misquoted as stating that the world really only would need five computers. Ironically the frequently used, erroneous quotation may come to true by the very hands of the business Watson created.

IBM launched an Epic project with an almost unfathomable goal -- to develop a single supercomputer capable of running the entire internet as a web application. The project, codenamed *Kittyhawk* (detailed in a white paper by IBM) created quite the stir in internet technology community.

While the software details descend quickly into the realm of the cerebral, one number that jumps off the page is the estimate for the number of cores and memory for the finished proposed system -- 67.1 million cores with 32PB of memory.

TD Bank Partners with IBM on Stream Computing

April 2, 2008 by

tweet | share | share | share | email

IBM today announced a partnership with TD Bank Financial Group to explore the use of IBM's stream computing software system running on an IBM Blue Gene supercomputer. Together the team will utilize a fundamentally new software architecture in order to examine thousands of real-time information sources to support financial services companies. The goal being to capitalize on on up-to-the-minute changing market conditions.

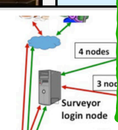
> *"Today's hardware and software is not optimized for real-time analysis of data. They came from a paradigm that paused and then queried a database of past information to give an answer while new information was still coming in that may affect the outcome," said Nagui Halim, chief scientist of the Stream Computing project in IBM Research. "We've designed and constructed a computing system from the ground up to provide continual analytics as events happen, which has very powerful applications in financial services, government, and many other scientific and business areas.*

Traditional computing paradigms and software architectures are bogged down by large volumes of data, increasing operational complexity and real-time business demands. IBM's new stream computing software architecture is pegged to break through these boundaries.

HPCwire

Since 1987 - Covering the Fastest Computers in the World and the People Who Run Them

Home | Topics | Sectors | Exascale | Specials | Resource Library | Podcast | Events | Job Bank



Researchers Implement HPC-First Cloud Approach

By Tiffany Trader

January 29, 2014

Researchers at North Carolina State University working in partnership with the Argonne Leadership Computing Facility (ALCF) have successfully demonstrated a proof-of-concept for a novel high-performance cloud computing platform by merging a cloud computing environment with a supercomputer. Project leads Patrick Dreher and Mladen Vouk discuss the evolution of the project on the ALCF website.

Project Kittyhawk: Building a Global-Scale Computer

Blue Gene/P as a Generic Computing Platform

Jonathan Appavoo, Volkmar Uhlig, Amos Waterland

IBM T.J. Watson Research Center, Yorktown Heights, NY

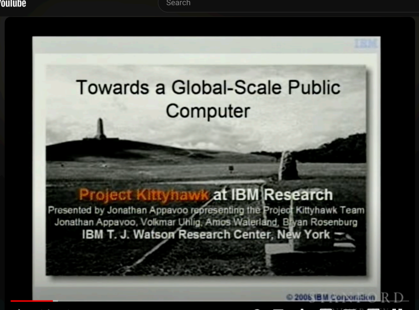
ACM SIGOPS Operating Systems Review, Volume 42, Issue 1January 2008

Kittyhawk: Enabling cooperation and competition in a global, shared computational system

J. Appavoo, V. Uhlig, A. Waterland, B. Rosenburg, D. Da Silva, J. E. Moreira

Kittyhawk represents our vision for a Web-scale computational resource that can accommodate a significant fraction of the world's computation needs and enable various parties to compete and cooperate in the provisioning of services on a consolidated platform. In this paper, we explain both the vision and the system architecture that supports it. We demonstrate these ideas by way of a prototype implementation that uses the IBM Blue Gene®/P platform. In the Kittyhawk prototype, we define a set of basic services that enable the allocation and interconnection of computing resources. By using examples, we show how higher layers of services can be built by using our basic services and standard open-source software.

YouTube



Towards a Global Scale Public Computer

Stanford

HOME › NEWS › APPLICATIONS OS › IBM DEVELOPS 'WORLD'S FASTEST FINANCIAL ANALYSIS' SYSTEM

Applications & OS

IBM Develops 'World's Fastest Financial Analysis' System

BY RICK WHITING

APRIL 10, 2009, 11:27 AM EDT

supercomputer software

Scientists at IBM Research collaborated with TD Securities during the project, which IBM said achieved a 21-times performance improvement on the volume of data consumed by today's financial trading systems.

Financial service companies such as stock brokerages rely on analysis of fixed snapshots of financial data streaming through their trading systems to make buy and sell decisions. But the volume of options and equities trading traffic has doubled every year since 2003, spurred, in part, by increased electronic trading. IBM said some systems today process up to 2 million transaction messages per second.

That means traditional approaches to capturing and analyzing trading data is increasingly inadequ...

Providing a Cloud Network Infrastructure on a Supercomputer

Jonathan Appavoo, Volkmar Uhlig, Jan Stoess, Amos Waterland, Bryan Rosenburg, Robert Wisniewski, Dilma Da Silva, Eric Van Hensbergen, Udo Steinberg

Boston University, Aster Data Inc., Karlsruhe Institute of Technology, Germany, IBM Research, Technische Universität Dresden, Germany

A Light-Weight Virtual Machine Monitor for Blue Gene/P

Jan Stoess, Udo Steinberg, Volkmar Uhlig, Jonathan Appavoo, Amos Waterland, Jens Kehne

Karlsruhe Institute of Technology, Technische Universität Dresden, HStreaming LLC, Harvard School of Engineering and Applied Sciences, Boston University

Virtualization as an addon

ROUGH TYPE

Nicholas Carr's blog

HOME | ABOUT | SUBSCRIBE | NICHOLASCARR.COM

One computer to rule them all

IBM has launched an ambitious initiative, called Project Kittyhawk, aimed at building "a global-scale shared computer capable of hosting the entire Internet as an application." Forget Thomas Watson's apocryphal remark that the world may need only five computers. Maybe it needs just one.

This article is more than 1 year old

IBM explores 67.1m-core computer for running entire internet

Ruby on Rails on Rails on Rails on Rails

Ashlee Vance | Tue 5 Feb 2008 23:37 UTC

EXCLUSIVE We'll hand it to IBM's researchers. They think big - really big. Like holy-crap-what-have-you-done big.

Elastic Building Block Runtime — EbbRT 2012-2016

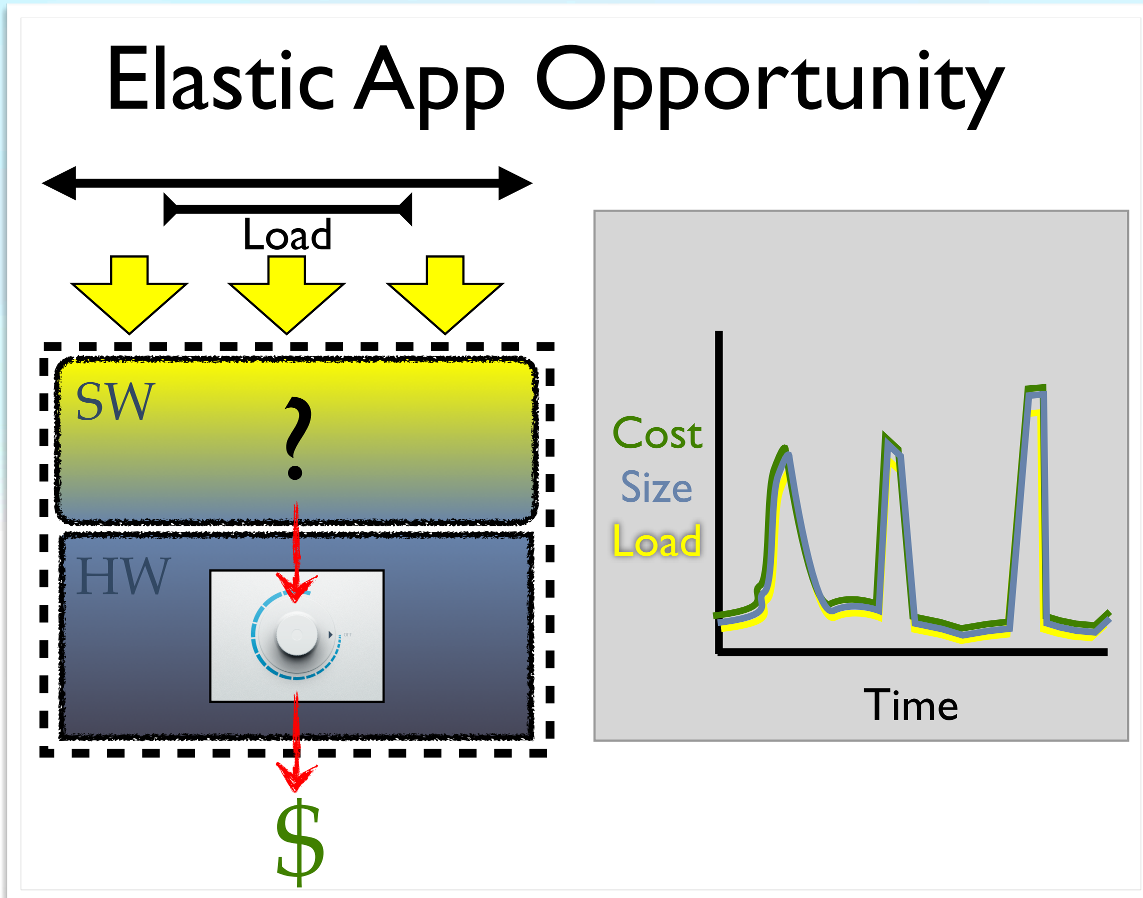
Building on the lessons we learned

Systems will eventually evolve that are massive utilities with a usage billing model (real IaaS)

What should the next layer of the OS look like to exploit this?

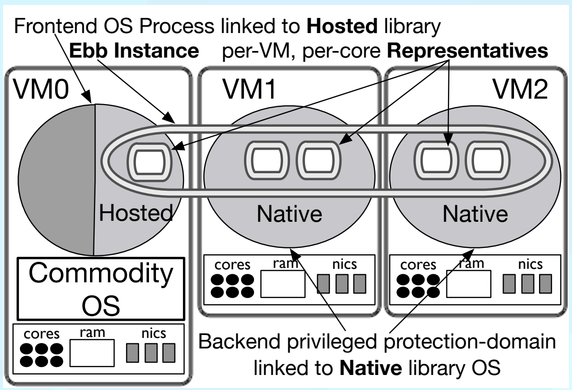
2012 - 2014 EbbLib PowerPC & X86 bare-metal prototype in C

2014 - 2016 switched directions EbbRT X86 only, Virtualized and bare-metal in C++

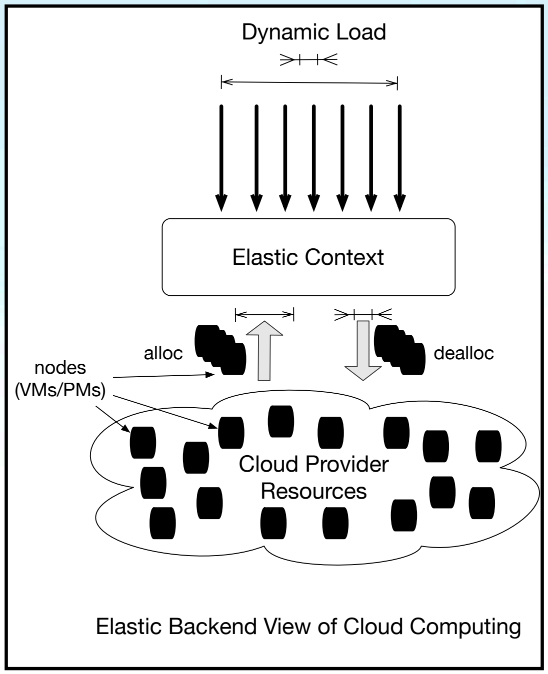


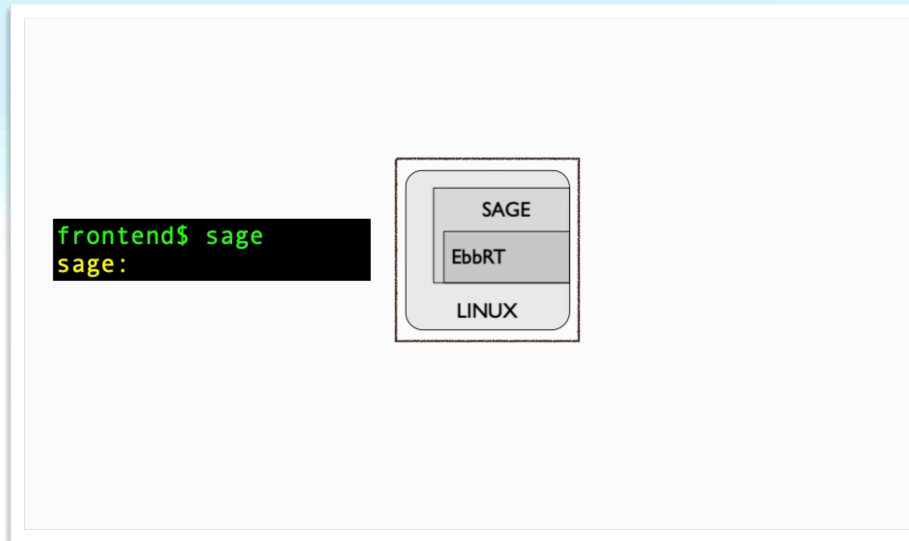
Scale, Specialiation and Elasticity go hand in hand.

EbbRT (Dan Schatzberg PhD) : Custom OS's as application Accelerator via a library on Linux



- Libra, a process accelerator
- EbbRT is a library for accelerating the whole or parts of an application via a standard Linux User Library.
- Makes it easy to integrate and exploit on-demand resources with custom OS optimizations





It becomes easy to add interactive HW Elasticity to an application.

Dan Schatzberg's PhD

Library/Unikernel OS can deliver the goods

- An application function and required system code as a library running at ring 0 in its own VM (or bare-metal node … not in these results)
- Application code running as interrupt handlers
- No protection domain switching, no arbitration, no rights checking, no multiplexing — almost feels like DOS ;-)
- Library of reusable K42-inspired elastic OS components (multicore, locality preserving and hot-swappable)

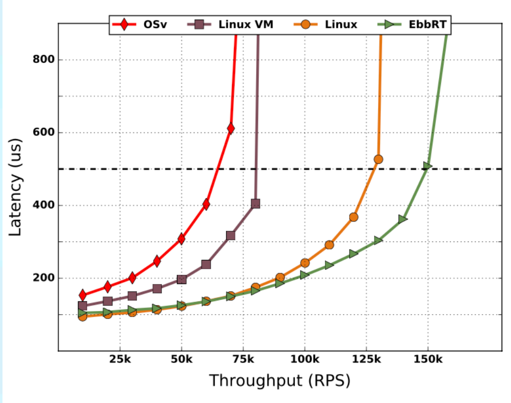


Figure 6: Memcached Single Core Performance

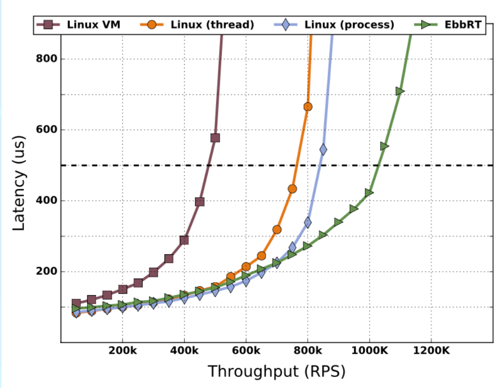


Figure 7: Memcached Multicore Performance

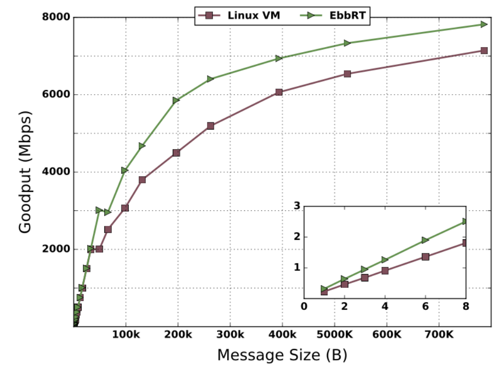


Figure 5: NetPIPE performance as a function of message size. Inset shows small message sizes.

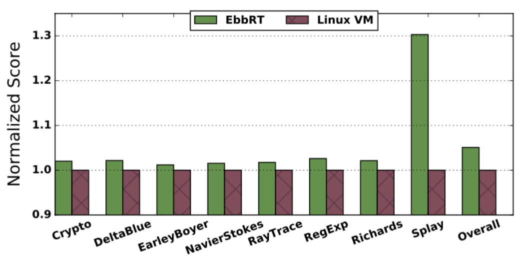


Figure 8: V8 JavaScript Benchmark


Dan Schatzberg's PhD

Scalable Elastic Systems Architecture

Dan Schatzberg
Boston University
dschatz@bu.edu

Jonathan Appavoo
Boston University
jappavoo@bu.edu

Orran Krieger
VMware
okrieger@vmware.com

Eric Van Hensbergen
IBM Austin Research Lab
ericvanhensbergen@us.ibm.com

1. Introduction

Elasticity should be treated as a first class system parameter. Particularly in large cloud environments, elastic applications would benefit if the underlying infrastructure provided primitives for elasticity and were themselves elastic. If you want to provide an elastic service and the cloud does not provide good primitives for the degree of elasticity you require, then you are forced to over-provision – acquire more resources than you instantaneously need and subsequently hoard them. Doing so hinders the cloud's ability to optimize global system utilization. Free or idle resources become hidden. If however, each cloud layer provides appropriate primitives that permit resources to be acquired and released at a scale that is equal to or better than what is required, then hoarding is less likely to occur. This permits the cloud infrastructure to collectively migrate resources to the real demand. To achieve this in a multi-layer system, demand must be transparently reflected from top to bottom. We must focus on the design and evaluation of primitives for expressing and managing elasticity at all levels, across nodes, and potentially across data centers.

If research focuses on pushing the boundaries of elasticity, new classes of applications can be developed. For example, if a cloud would permit an application to grow and shrink the use of thousands of processors between mouse clicks, then *High Performance Interactive Applications* would be viable. Consider a medical imaging and analysis application. Using a raw megapixel image with an algorithm requiring quadratic memory in the size of the input, this requires roughly 14 terabytes of memory, putting it well outside the reach of the ram capacities of desktop computers. However, a "small" supercomputer today ($1/10$ of the largest current IBM BlueGene P System), capable of approximately 10^{14} operations per second, can not only contain the data, but can perform an operation on each data value in under a second. All of a sudden, operating on the image not only becomes viable, but we can even do it at interactive speeds.

While an interactive version of this application has large value, it is not feasible today. Suppose a doctor's office had the necessary software and wanted to use Amazon's EC2 HPC offering for an 8 hour work day. To operate on the image would require 623 compute instances[1]. Given pricing at the time of writing, this translates to approximately $8000.00 per day. Due to the interactive nature of the application, the actual utilization of the instances will be a small fraction of the time that is being paid for. This is likely a cost prohibitive proposition. If, however, it was possible to acquire and release the resources at interactive time scales, then the instances could be reallocated to other EC2 users and the doctor's cost would more closely reflect the usage. Researching dramatically higher degrees of elasticity with respect to the scale of the resources and duration they are held would enable such high performance interactive applications.

If we develop effective ways of exporting the elasticity via designed and usable primitives, then we can not only ease the burden of developing elastic applications and services, but also we can foster and encourage them. We can reduce the application development burden by providing suptport for representing and reflecting dynamic demand and translating it into dynamic requests for resources. Similar to how a traditional operating system transparently manages memory via mappings and pages faults, one can explore how systems can enable primitives for elasticity.

In summary we argue that elasticity is an important area of research and hypothesize that research in this area will lead to more efficient systems with less hoarding, new applications that exploit massive cloud resources elastically, and system software and libraries that will simplify the task of developing elastic applications.

In this talk we will present our goals for a system that supports extreme elasticity. Motivated by these goals, we will present our Scalable Elastic Systems Architecture (SESA).

2. Goals

Based on our observations, we posit the following goals for a systems architecture for elasticity:

Top-Down Demand The system should enable demand on services to flow from high level layers as transparently as possible to the lowest layers of the system. Hoarding should be discouraged or at least made transparent. Event driven interfaces and services should be supported and encouraged by the system.

Bottom-Up Support We advocate that elasticity should be an explicit characteristic that should be supported in the lowest layers of a system and, if possible, all the way into the hardware. The construction of layers that are explicit about the elasticity they provide with respect to the base elasticity of the system should be encouraged via systems support.

1st ACM ASPLOS Runtime Environments/Systems, Layering, and Virtualization Environments (RESoLVE 2011), March 5, 2011.

A Way Forward: Enabling Operating System Innovation in the Cloud

Dan Schatzberg, James Cadden, Orran Krieger, Jonathan Appavoo
Boston University

1 Introduction

Cloud computing has not resulted in a fundamental change to the underlying operating systems. Rather, distributed applications are built over middleware that provides high-level abstractions to exploit the cloud's scale and elasticity. This middleware conjoins many general purpose OS instances.

Others have demonstrated that a new operating system built specifically for the cloud can achieve increased efficiency, scale and functionality [11, 14]. However, this work does not take into account the way applications are being deployed in cloud environments. In particular, entire physical or virtual machines are being dedicated to run a single application, rather than concurrently supporting many users and multiple applications.

In this paper we introduce a new model for distributed applications that embraces a reduced role of the OS in the cloud. It allows for the construction of application-driven compositions of OS functionality wherein each application can employ its own customized operating system.

2 Role of the OS

For security and auditability, Infrastructure as a Service (IaaS) providers isolate their tenants at a very low level as physical or virtual compute *nodes*. Individual tenants own and manage their compute nodes, software stack, networks and disks within an IaaS cloud.

Typically, scale-out cloud applications run across a set of compute nodes solely dedicated to that application. In such an environment, three of the major objectives that general purpose operating systems were designed to meet are relaxed or eliminated entirely.

First, the burden to support multiple users is removed from the operating system. In this environment, the isolation enforced by the IaaS provider eliminates the need for many system level security checks and accounting, and reduces the requirement for internal barriers between trusted and untrusted code.

Second, it becomes the responsibility of the IaaS provider to arbitrate and balance competitive resource usage. In a deployment where entire nodes are assigned to a single application, much of the complexity of existing operating systems (e.g., scheduling, memory management, etc.) is redundant.

Third, a symmetric structure is unnecessary in a large-scale distributed application. Many cloud applications are already composed of multiple services run across a set of compute nodes; As a result, OS functionality can be provided asymmetrically, where only some nodes need full OS functionality, while other nodes can be much simpler.

Given these observations, it is apparent that distributed cloud applications built on top of general purpose systems are comprised of unnecessary software functionality with the risk of reduced performance and added complexity.

Proceedings of the 6th USENIX Conference on Hot Topics in Cloud Computing (HotCloud'14), June 17, 2014

usenix
THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

EbbRT: A Framework for Building Per-Application Library Operating Systems

Dan Schatzberg, James Cadden, Han Dong, Orran Krieger, and Jonathan Appavoo, *Boston University*

https://www.usenix.org/conference/osdi16/technical-sessions/presentation/schatzberg

This paper is included in the Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI '16).
November 2–4, 2016 • Savannah, GA, USA
ISBN 978-1-931971-33-1

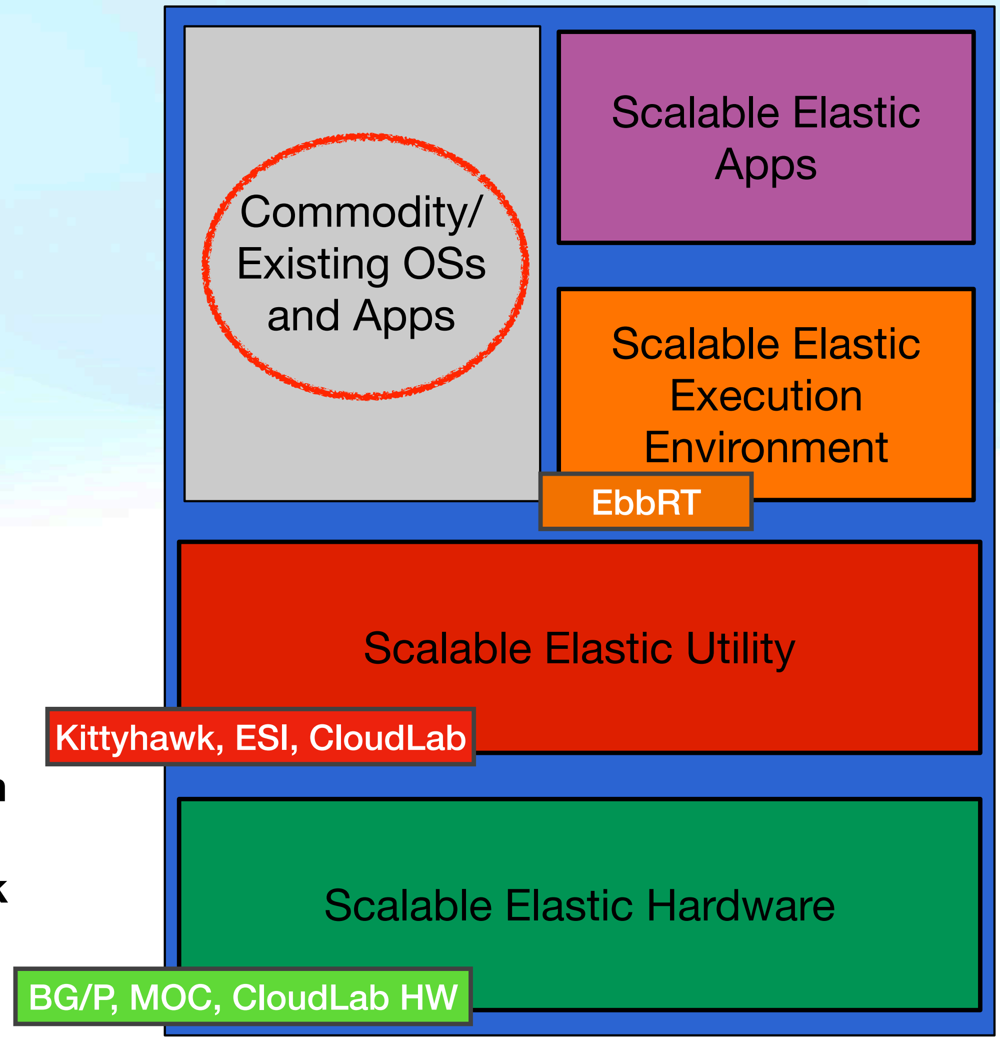
Open access to the Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation is sponsored by USENIX.

Dan Schatzberg's PhD

OS specialization is coming — starting in today's Cloud and begin ready for tomorrows

1. Are from scratch library OS the only way?
   a. UKL
   b Dynamic Privilege
2. Can specialization with energy consumption?
   a. A Data-Driven Study of Operating System Energy-Performance Trade-offs Towards System Self Optimization
   b. *Energy Efficient Stream Computing with Flink
   c. *Container Energy Efficiency
   d. *Specializing Linux for Energy Efficiency

SESA View of the World



Are from scratch library OS the only way?

While today's IaaS may not have the speed of elasticity as Kittyhawk

Today's Clouds have enough HW, VMs, and Containers **dedication** to a **single** component of a web service (key-value store, DB, FaaS executor,…) is standard. This has led to a renewed interest in library OSs/Unikernels.

Is there an a "easier" way to run a dedicated task?

"easier" to "use"

UKL: Is the LibraryOS important or are the optimizations?

Can we find ways of integrating uni-kernel techniques within Linux (and thus leverage its code base)

kernel

\+

application

\+

a little surgery

=

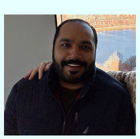
Uni-Kernel Linux

?

lean and fast

or

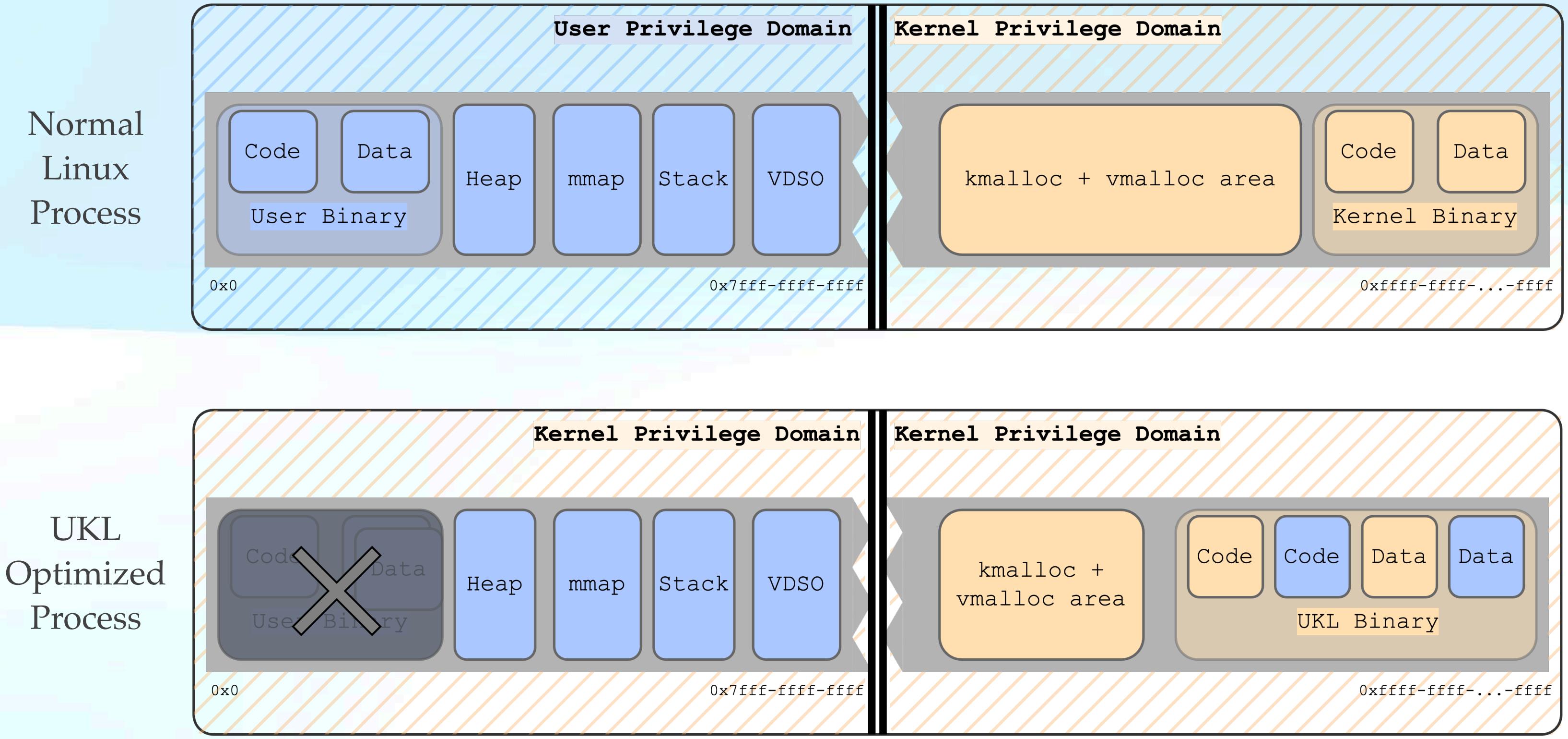
road kill ;-)



Ali Raza’s PhD

Linux & UKL Address Spaces

A new approach to General Purpose - Custom UniKernels

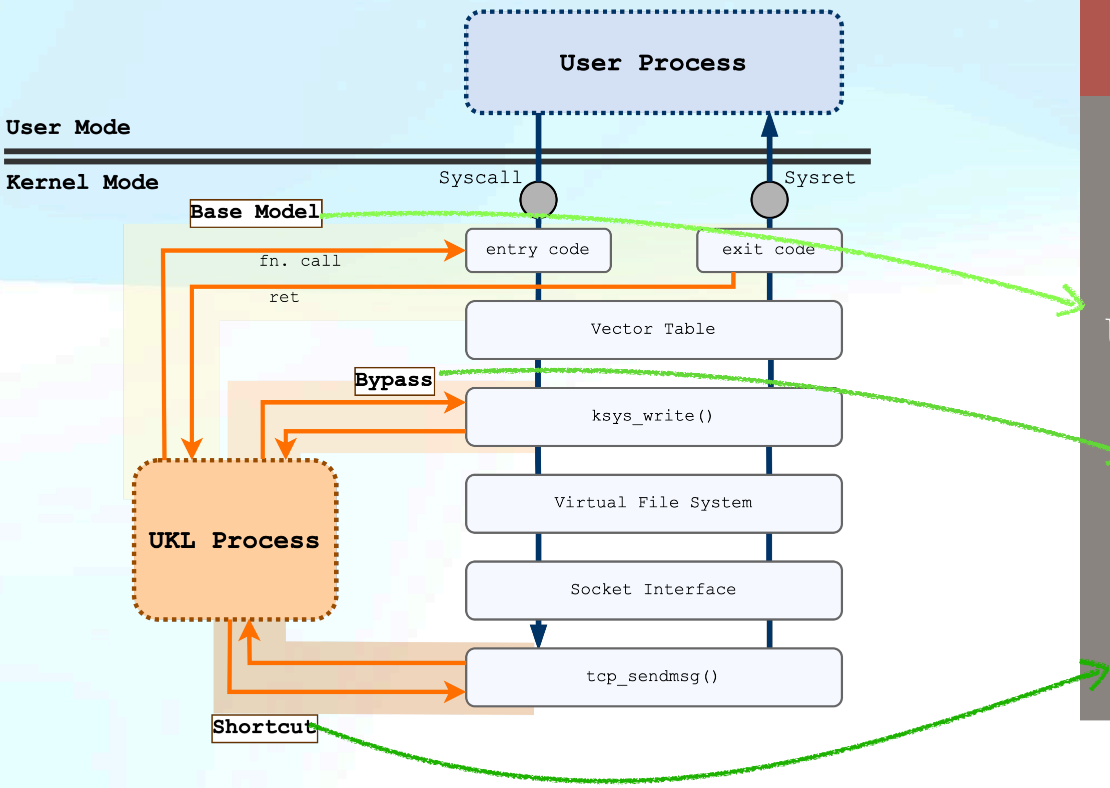


- Binaries statically linked
- Kernel Privilege


Ali Raza's PhD

Redis - Baremetal

Incremental specialization - may often be enough



	X-put %higher	99-Tail %faster
Linux	-	-
UKL Base	1.2%	0.3%
UKL Bypass	12%	11%
UKL Shortcut	26%	22%

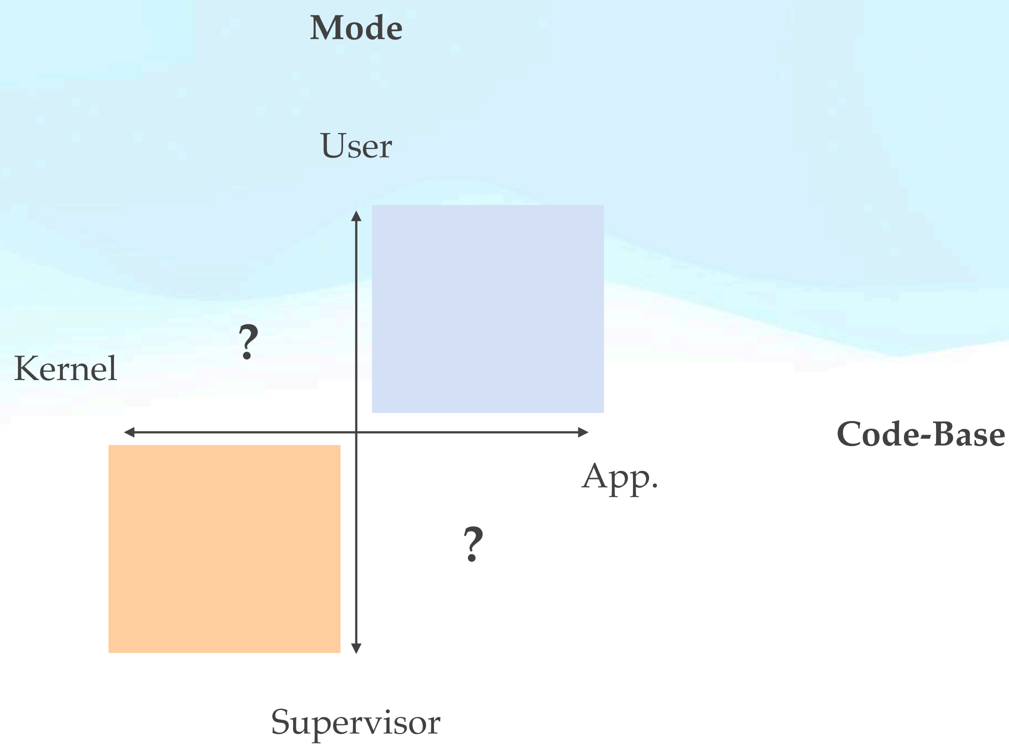
Ali Raza's PhD

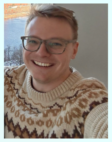
Dynamic Privilege

There may be an even more fundamental OS change we can make.

Let's question the very ground we have stood on.

/sys/libkernel.so ;-)





Tommy Unger's PhD

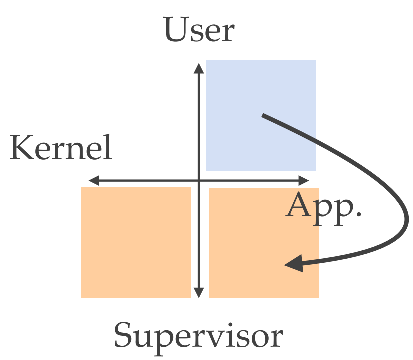
Privilege as a runtime switch

Enables a new kind of "application"

- Bracketing privilege access in time
- Accessing a privileged register
    - x86_64 Control Register 3
- Executing privileged instructions
    - mov cr3, %rax

```
priv = loadMod(priv_lib_path)
kele = loadMod(kele_lib_path)
kele.kElevate()
cr3 = priv.getcr3()
kele.kLower()
print(f"CR3 = 0x{cr3:x}")
```

```
CR3 = 0x2e868002
```



The entire kElevate patch for Linux (x86_64) is 173 lines of new kernel code

Tommy Unger's PhD

Dynamic Privilege from Anywhere

- We've mainly prototyped in C, C++, and Python, but have written prototypes to demonstrate kElevate can be used in others
  - Java
  - Node
  - Rust
  - Go
  - Assembly

```
In [10]:
ret = com.run_cmd("taskset -c 0 idt_tool -g")
addr_old_idt = ret.stdout.splitlines()[0]
# Prepend 0x if not there.
addr_old_idt = hex(int(addr_old_idt, 16))
# Print the old IDT is located at this address

print("Old IDT is located at: ", addr_old_idt)

Old IDT is located at:  0xffffc9000880f000

In [11]:
# Allocate a kernel page, copy the old idt onto it, and return the address of this page
ret = com.run_cmd("taskset -c 0 idt_tool -c")
addr_new_idt = ret.stdout.splitlines()[0]
addr_new_idt = hex(int(addr_new_idt, 16))

print("New IDT is located at: ", addr_new_idt)

New IDT is located at:  0xffffc90003f57000

In [13]:
# Install the new IDT
com.run_cmd("taskset -c 0 idt_tool -i -a " + addr_new_idt)

# Get the currently loaded IDT ptr
ret = com.run_cmd("taskset -c 0 idt_tool -g")
addr_current_idt = ret.stdout.splitlines()[0]
addr_current_idt = hex(int(addr_current_idt, 16))

print("Current IDT is located at: ", addr_current_idt)

Current IDT is located at:  0xffffc90003f57000
```

Who says you can't hack the kernel from a Jupyter Notebook

Shortcutting: Macrobenchmarks

Short Cuts are now a user tool!

- Refer to dissertation
  - Redis:
    - Shallow shortcuts: 9-11% xput improvement
    - Deep shortcuts: 20-22% xput improvement
  - Memcached
    - Shallow shortcuts: 9-11% xput improvement

```
$ ./sc.sh -s write->__x64_sys_write \
    --- ./wrLoop $SZ $ITER $PATH
```

```
$ ./sc.sh -s \
    write->tcp_sendmsg:0x42ed60 \
    --- ./wrLoop $SZ $ITER $PATH
```

With Dynamic Privilege, an app can remake the world (kernel) in its own image at both compile and runtime!

Can specialization help with energy consumption?

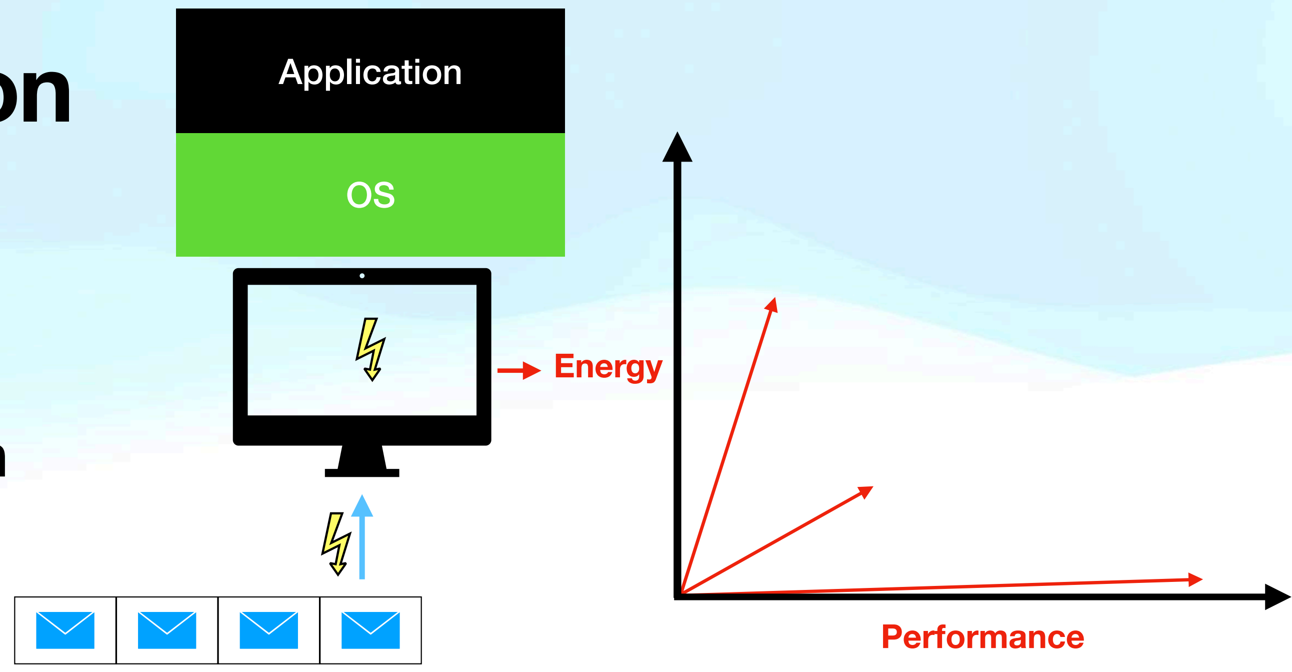
Using a computer efficiently should be easy!

“easier” to “use”

Can specialization help us use our resources more efficiently?

Two views of specialization

1. Special purpose OS
2. An ML-Tuning OS for the OS



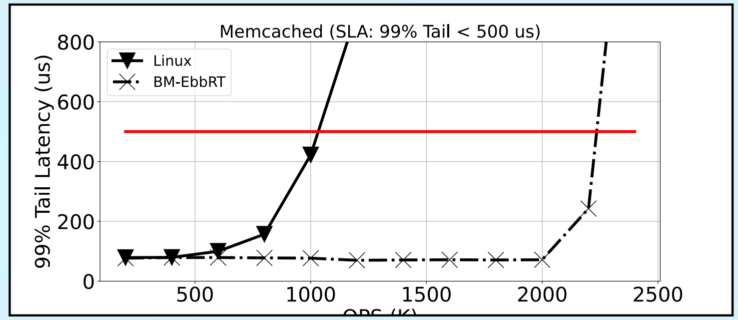
Han Dong's PhD

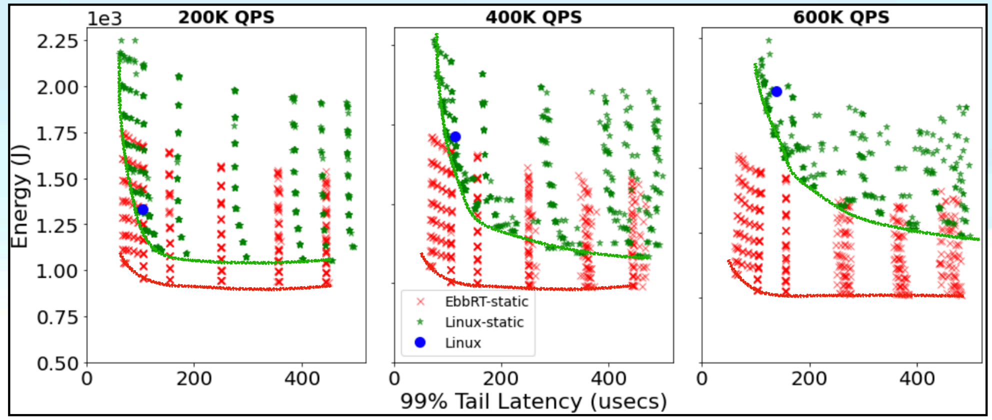
A Data-Driven Study of Operating System Energy-Performance Trade-offs Towards System Self Optimization

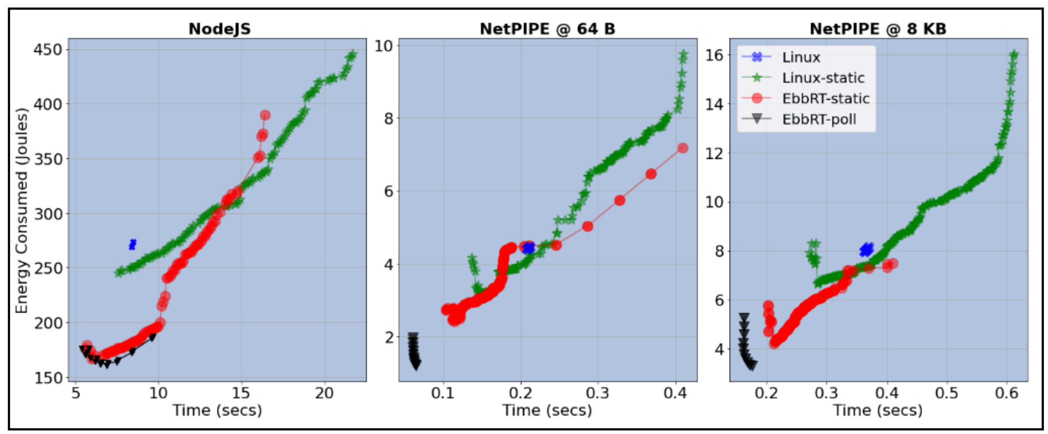
YES IT DOES!

1. Special purpose OS

- Wrote a 10Ge NIC driver for EbbRT
    - First bare-metal head-to-head comparison — performance win is even more dramatic than we thought!
- All workloads tested at all loads show a clear separation between specialized OS versus Linux





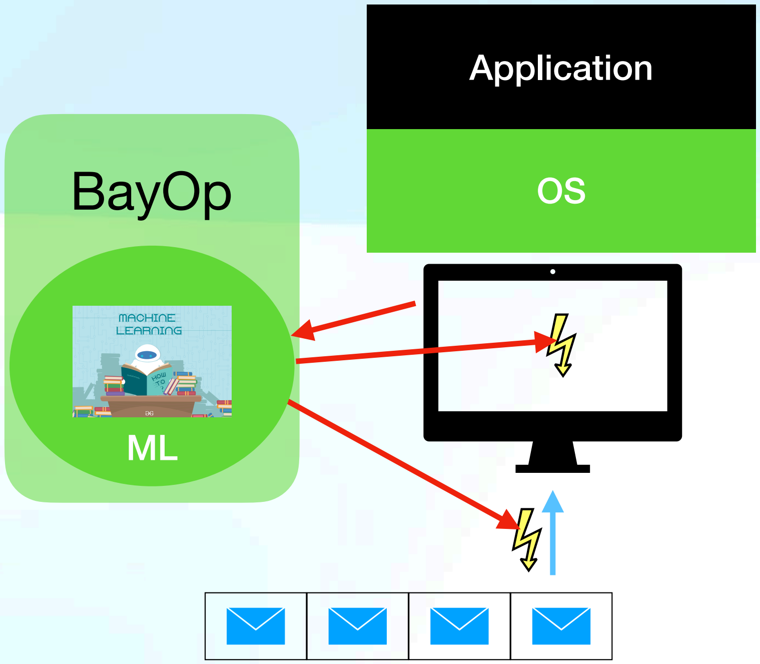


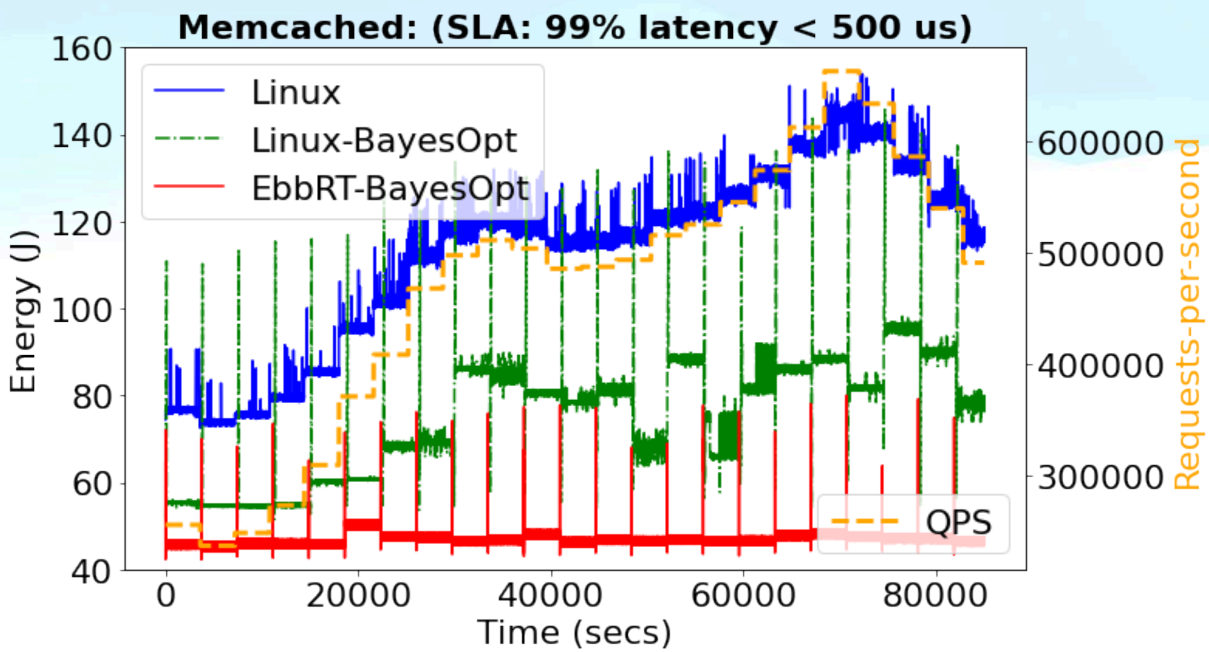

Han Dong's PhD

Another view of Specialization

2. An ML-Tuning OS for the OS

- Specialization can be a dynamic problem that sometimes we can out source
- A new view of the OS
  - expose controls and defer tuning to an external agent. When appropriate





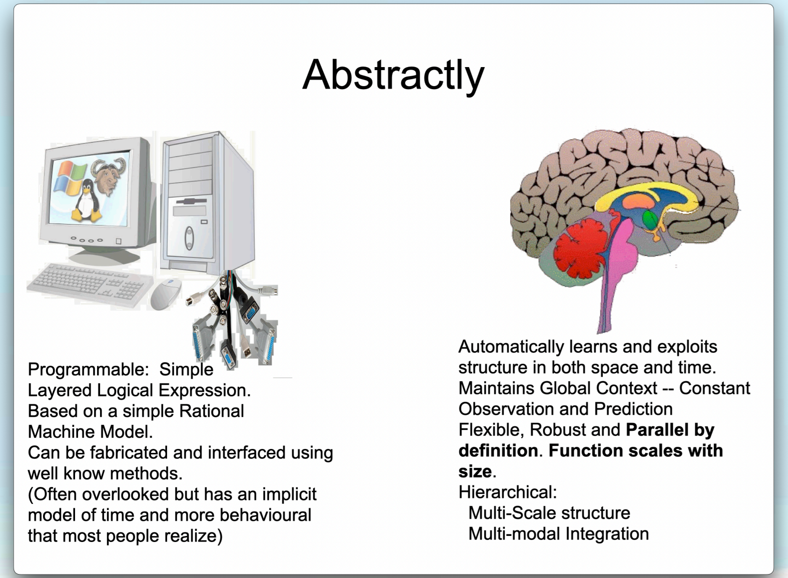
Han Dong's PhD

Openned the door

Projects currently being pursued

- **Energy Efficient Stream Computing with Flink**
  - Can we extend these results to a multi-node scenario?
  - Can we extend these results to a more complex software stack?
- **Container Energy Efficiency** (joint work with PEAKS)
  - What impact does containerization have on consumption
  - What impact does containerization have on control
- **The Shepherd OS Framework**
  - Generalize BayOp into a distributed OS with pluggable ML modules
- **Specializing Linux for Energy Efficiency**
  - Can we use Dynamic Privilege or UKL to obtain the same efficiency demonstrated by EbbRT?

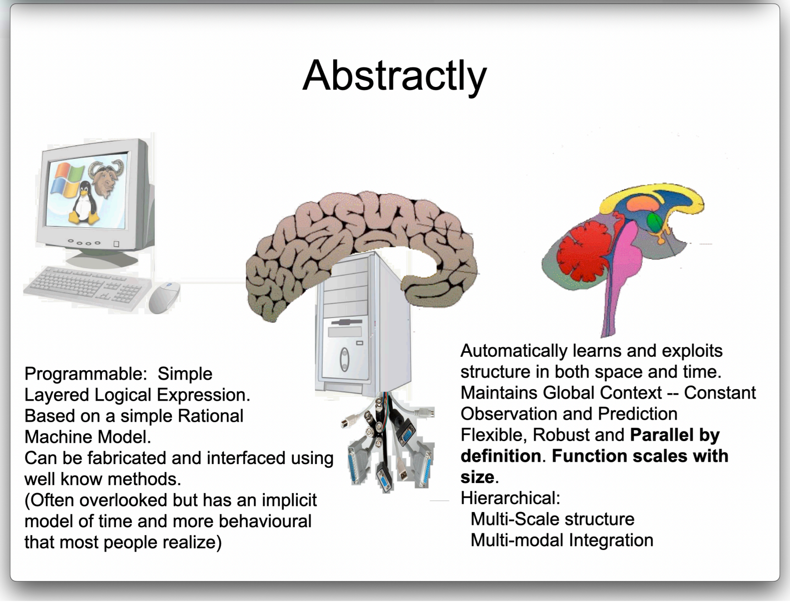
2008 PSML
DARPA Talk



One more Thing
2004 - Now

We have been exploring a radically different approach to scalability — a globally distributed "Computational Cache" that can exploit future sub-threshold (neuromorphic) devices without sacrificing programmability.

PSML, ASC, DANA, SUESS*, OM and ForkU*



OS: A collection of "software" that makes it "easier" to "use" a "computer."

Get stuff done with less effort, in less time, and fewer resources

Ultimately Hardware

- The evolution of hardware has not stopped yet — remember, we are both consumers and influencers (Stone and Dennings).
- The value of an Open foundation of Linux software is hard to deny
  - Perhaps the best way to preserve it is not to over-burden it
  - But this does not mean there is only one way to think about the OS
  - The cloud finally lets us redefine the OS, and as further elasticity and resources become available, there will be additional opportunities.
  - Efficient use of resources matters — help people be frugal
  - There are novel hybrid OS approaches — it need not be an all-or-nothing proposition
  - OS Researchers must be willing to look into the future and take risks even if the mainstream does not see the value — be willing to redefine, conjecture, and experiment

Even if systems programming gets outsourced to AI, creativity and science are not dead

Thank you to all our funders who have been willing to take leaps of faith and fund work that was not “obviously useful.”

We live in interesting times

We must be willing to change and adapt — being a brilliant systems programmers is not enough.

- Physics, Chemistry, and material sciences, if anything we are told are about to explode
- Every year, we are getting closer and closer to understanding the relationship between neurobiology, ML, and computation, more broadly.
- We are reaching the limits of what is sustainable, and we need to become more efficient.
- The impact of computer systems is genuinely global— both for good and bad
- Computation is power — this puts us and what we do at the forefront of socio-economics
- Our creativity, commitment to truth, and integrity matter, not just how much complex code we can write.