| galois |

The HaLVM

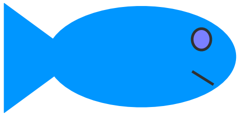
(Haskell Lightweight Virtual Machine)

Adam Wick

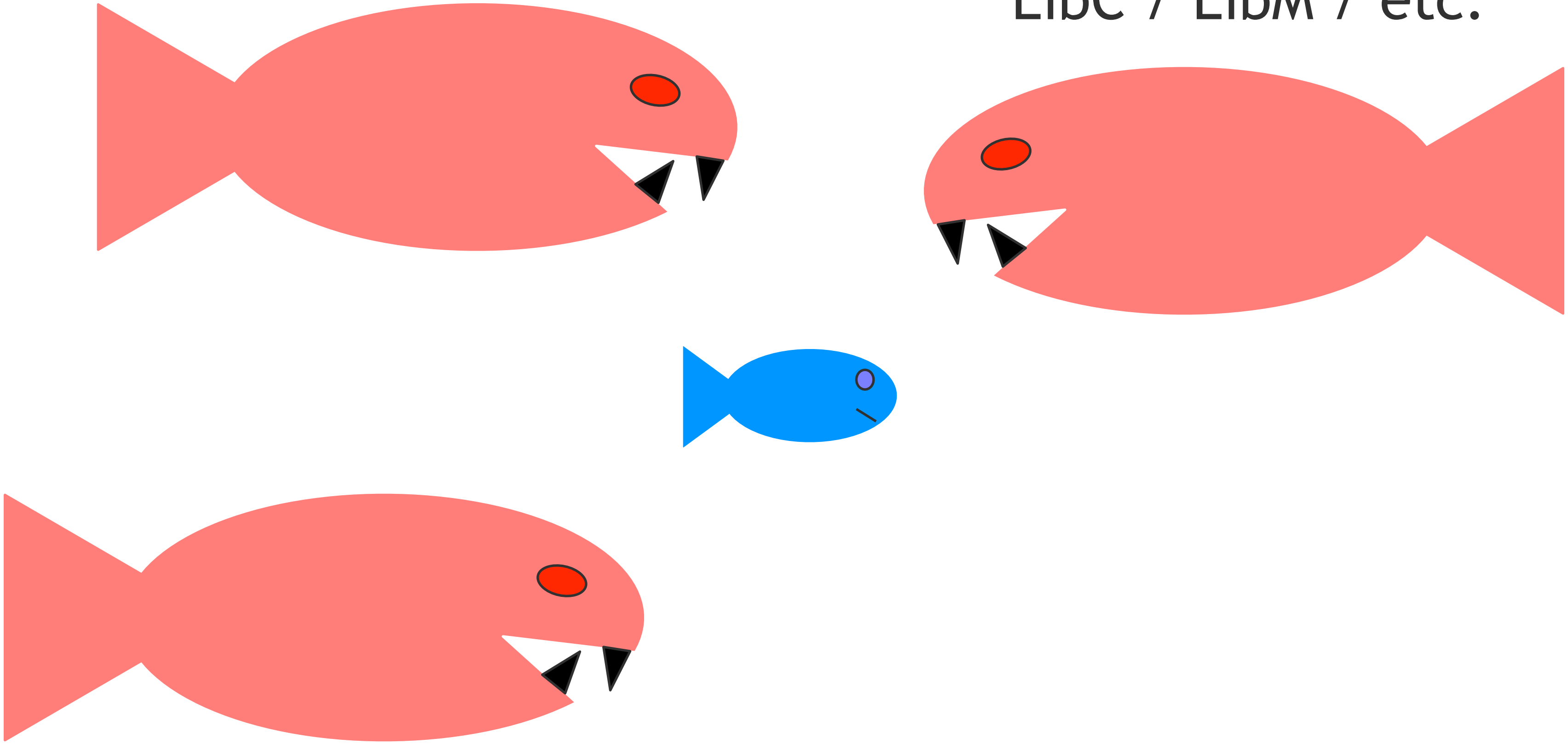
May 10th, 2007

Background

High-Assurance
Key Manager



Background



Linux Kernel

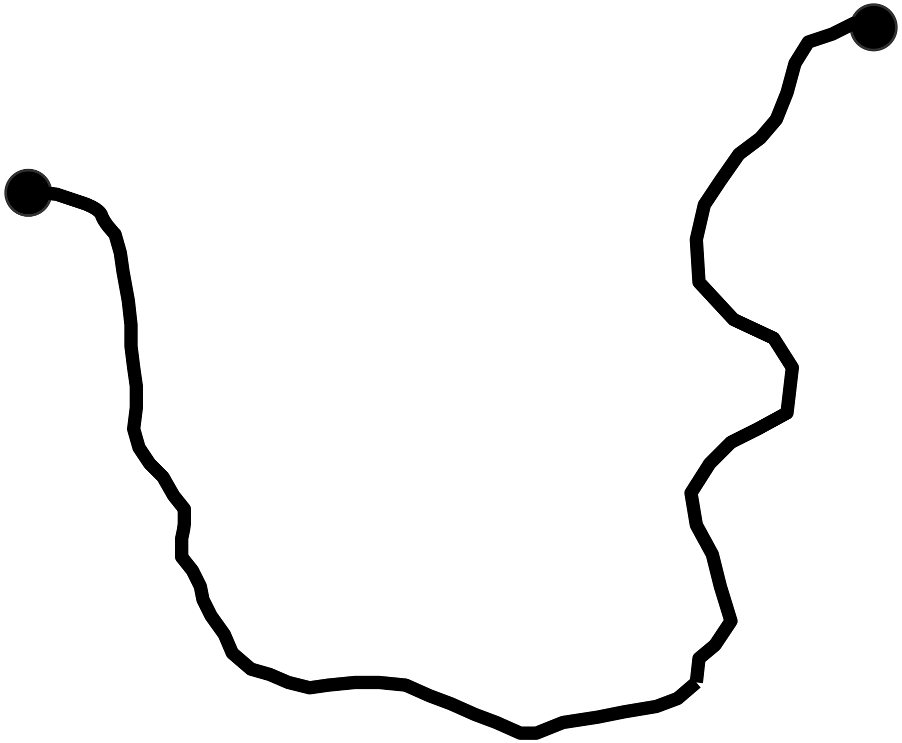
LibC / LibM / etc.

XWindows / Firefox / etc.

The Story Of The Solution

Problem:

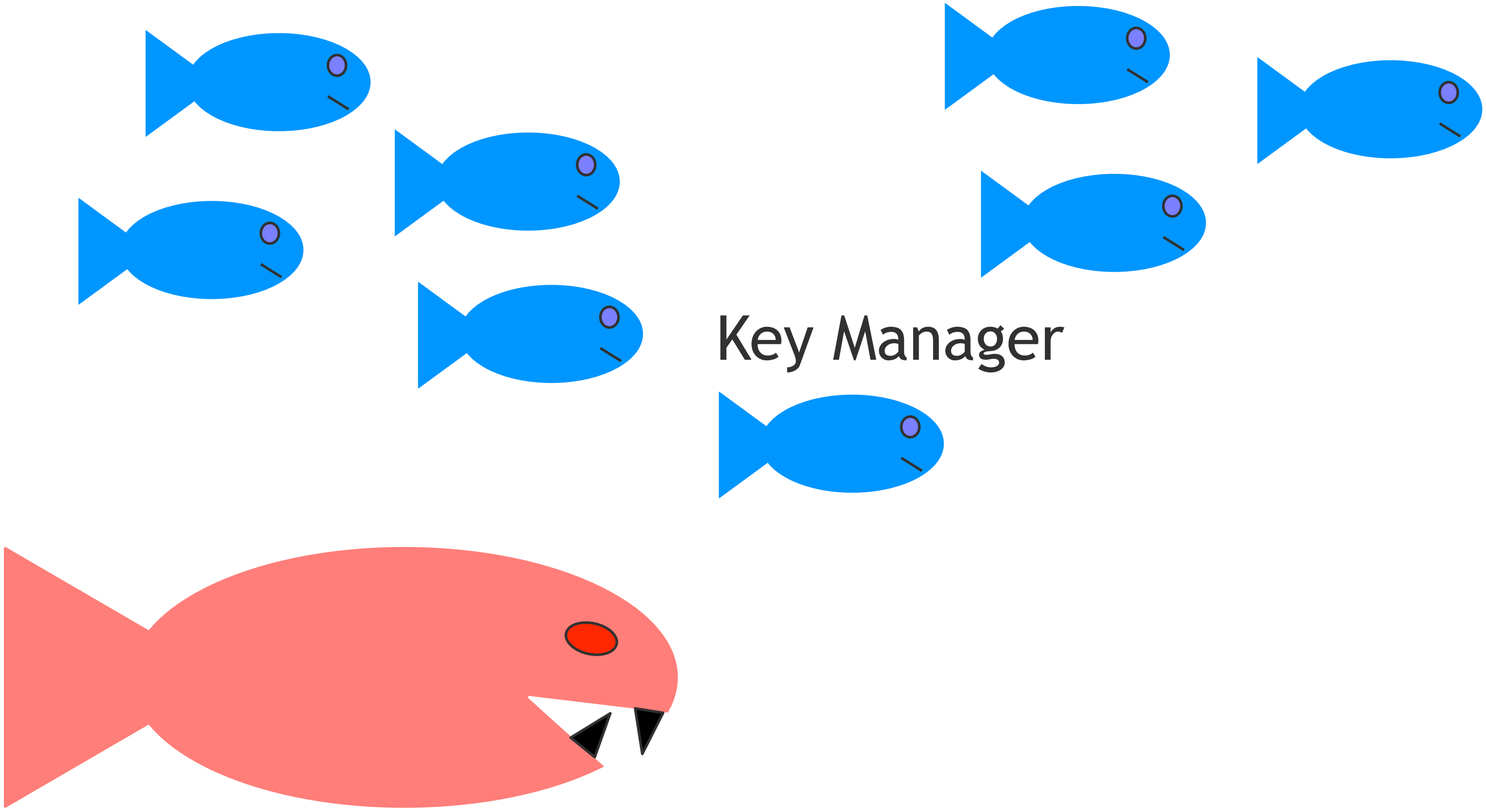
Safe harbor for high-assurance applications, coexisting with low assurance applications



Final Solution:

High-assurance operating system

Background



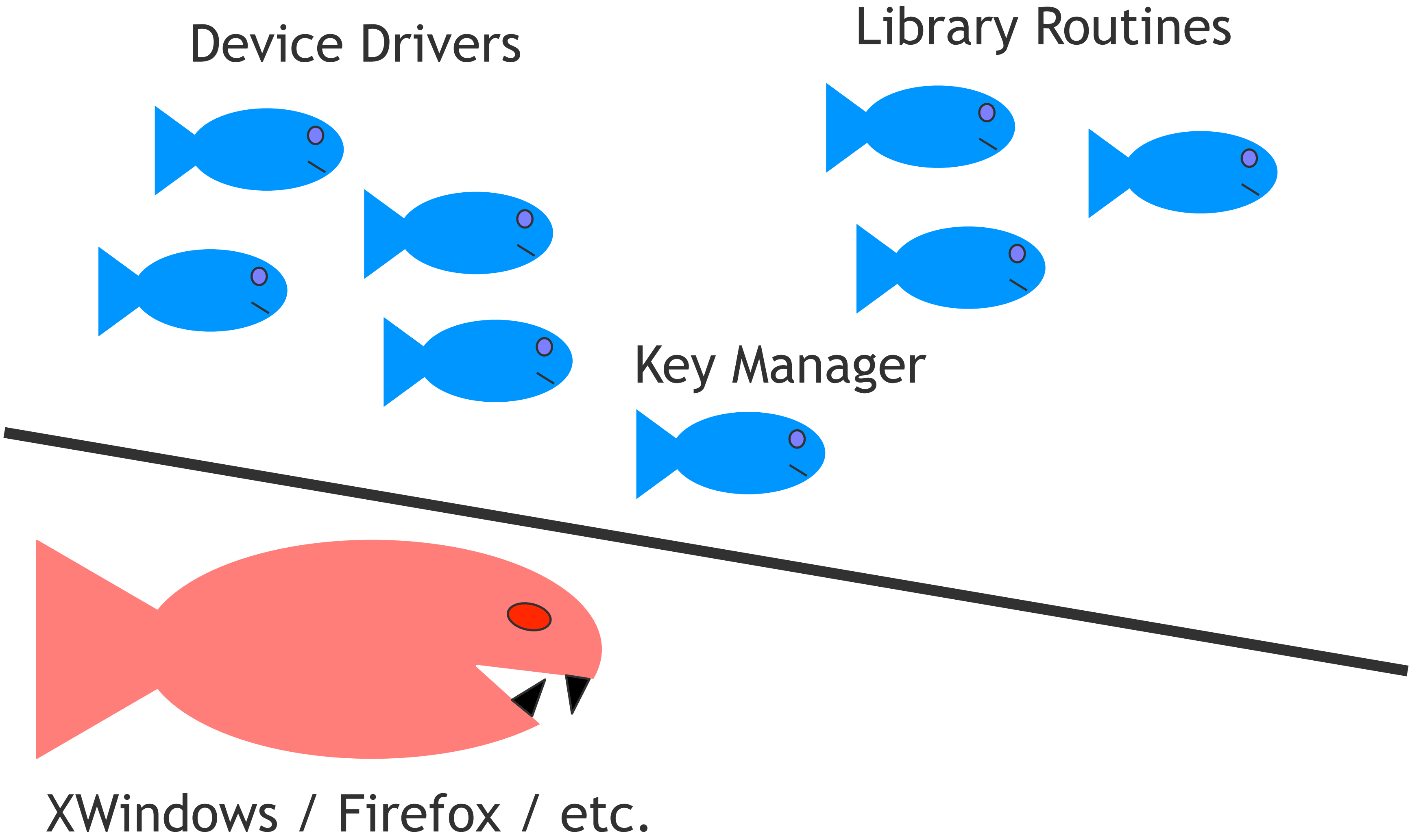
Device Drivers

Library Routines

Key Manager

XWindows / Firefox / etc.

Background


Device Drivers
Library Routines
Key Manager
XWindows / Firefox / etc.

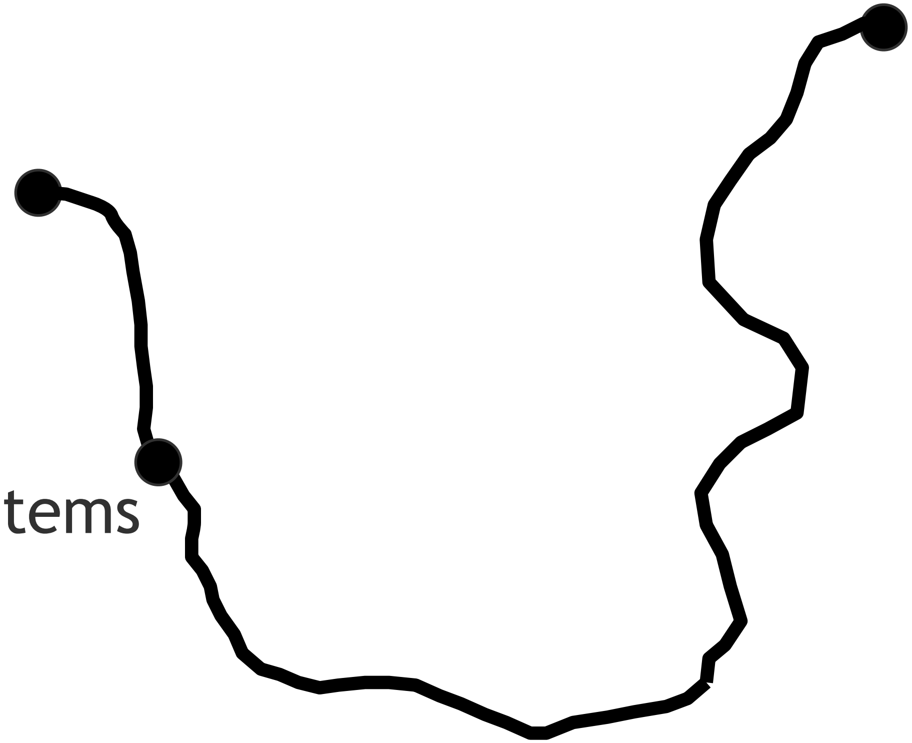
The Story Of The Solution

Problem:

Safe harbor for high-assurance applications, coexisting with low assurance applications

Final Solution:

High-assurance operating system

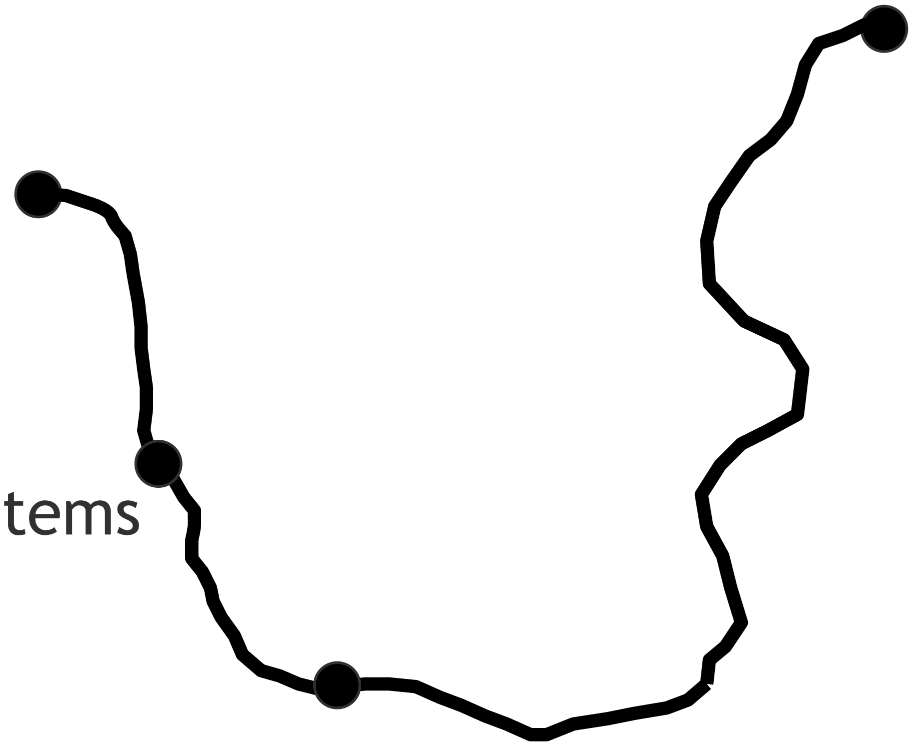


Operating systems are gigantic; decompose

The Story Of The Solution

Problem:

Safe harbor for high-assurance applications, coexisting with low assurance applications



Operating systems are gigantic; decompose

Modeling of OS components

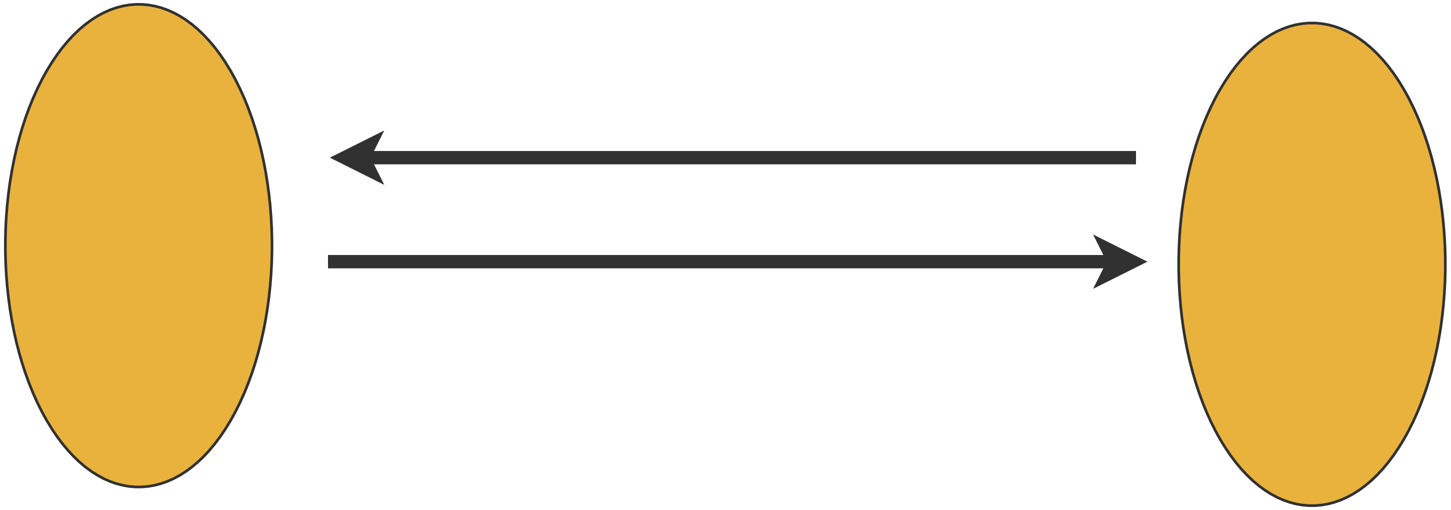
Final Solution:

High-assurance operating system

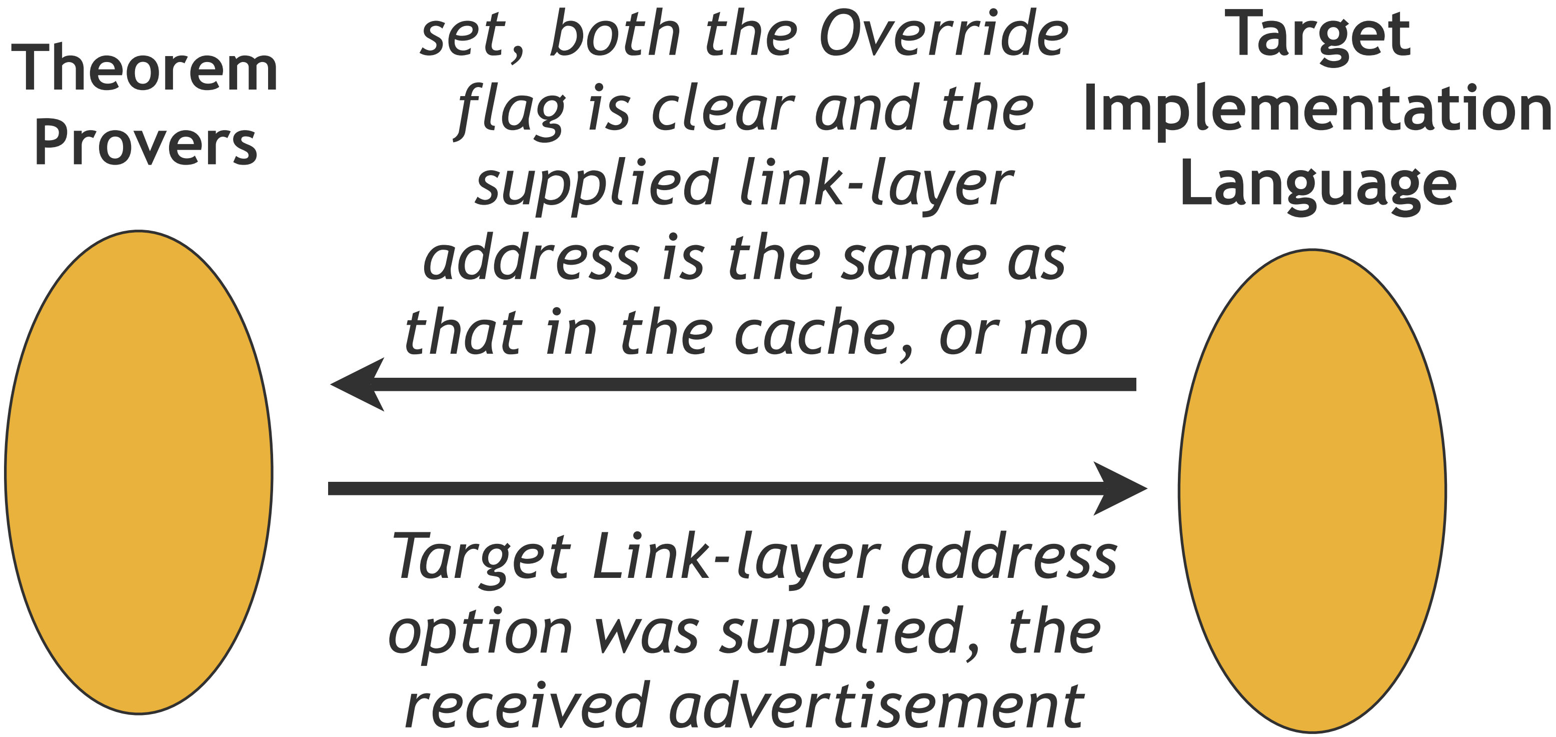
The Trouble With OS Modeling

Theorem Provers

Target Implementation Language

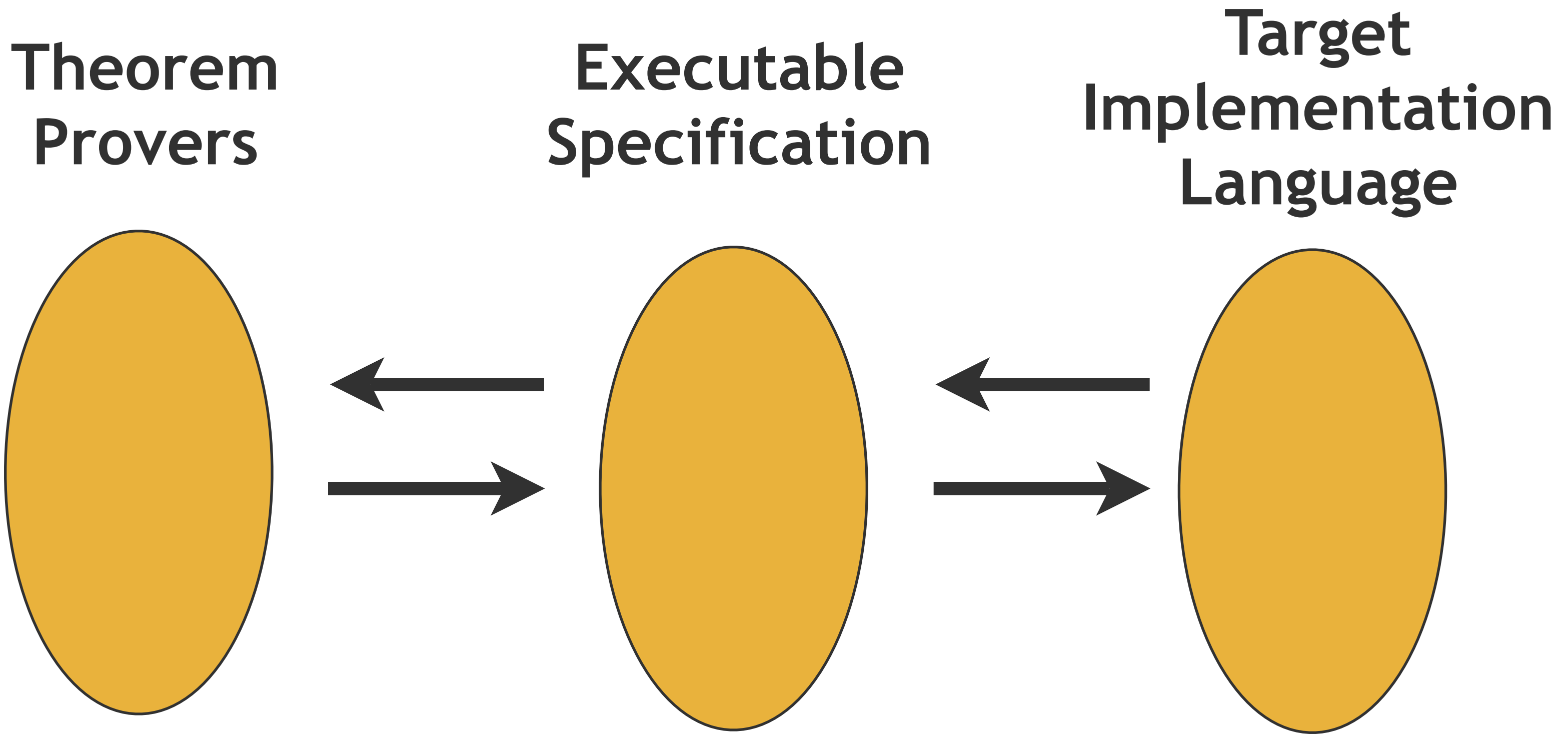


The Trouble With OS Modeling



“If the Override flag is set, both the Override flag is clear and the supplied link-layer address is the same as that in the cache, or no Target Link-layer address option was supplied, the received advertisement MUST update the Neighbor Cache entry as follows:” -- RFC 2461

The Trouble With OS Modeling



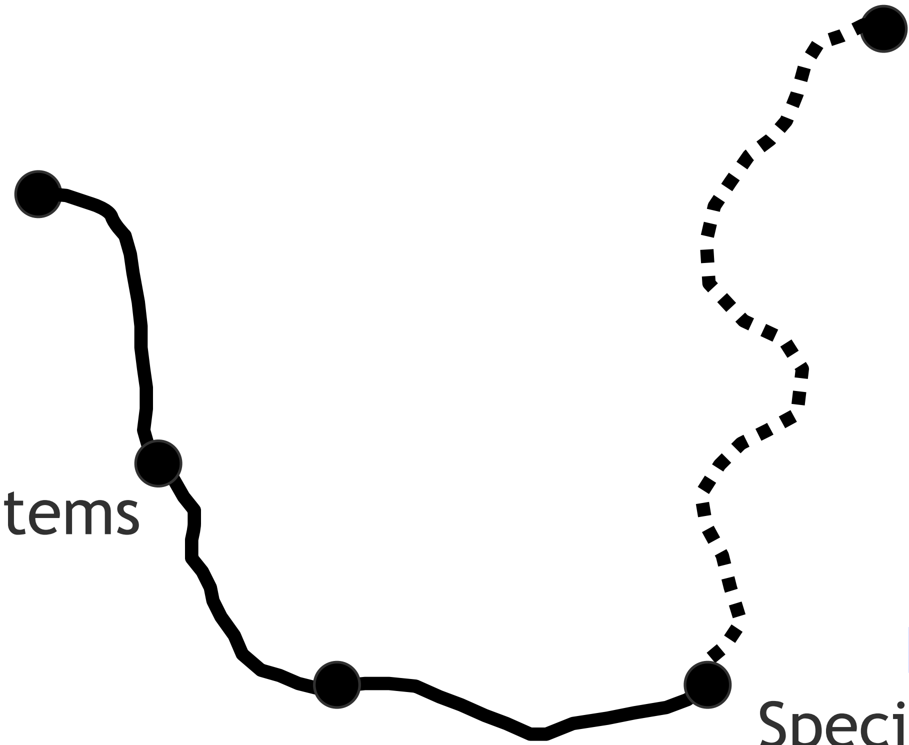
Haskell

- Haskell is a high-level, lazy, pure functional programming language.
- It is high-level enough to …
  - Serve as a good bridge to modeling languages.
  - Allow for simpler construction of complex programs, via powerful abstraction and composition mechanisms.
- It is low-level enough to …
  - Directly access memory, in order to implement device drivers.
  - Write imperative blocks of low-level code.

The Story Of This Talk

Problem:

Safe harbor for high-assurance applications, coexisting with low assurance applications



Operating systems are gigantic; decompose

Modeling of OS components

HaLVM

Specifications are difficult; test them on the hardware

Final Solution:

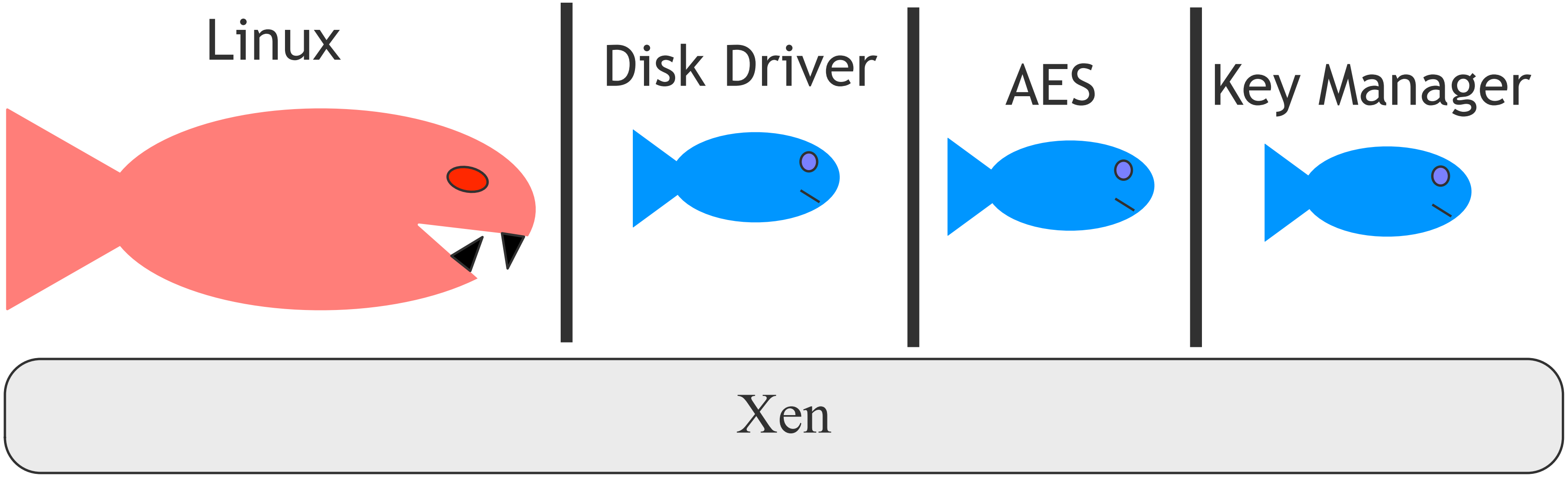
High-assurance operating system

Talk Outline

- Introduction
- Quick Technical Backgound
  - Xen
  - Haskell
  - Lightweight Virtual Machine
  - Use case
- An Overview of the HaLVM (with demos)
- Current Gaps and Future Work
- More Demos (as time permits)

galois

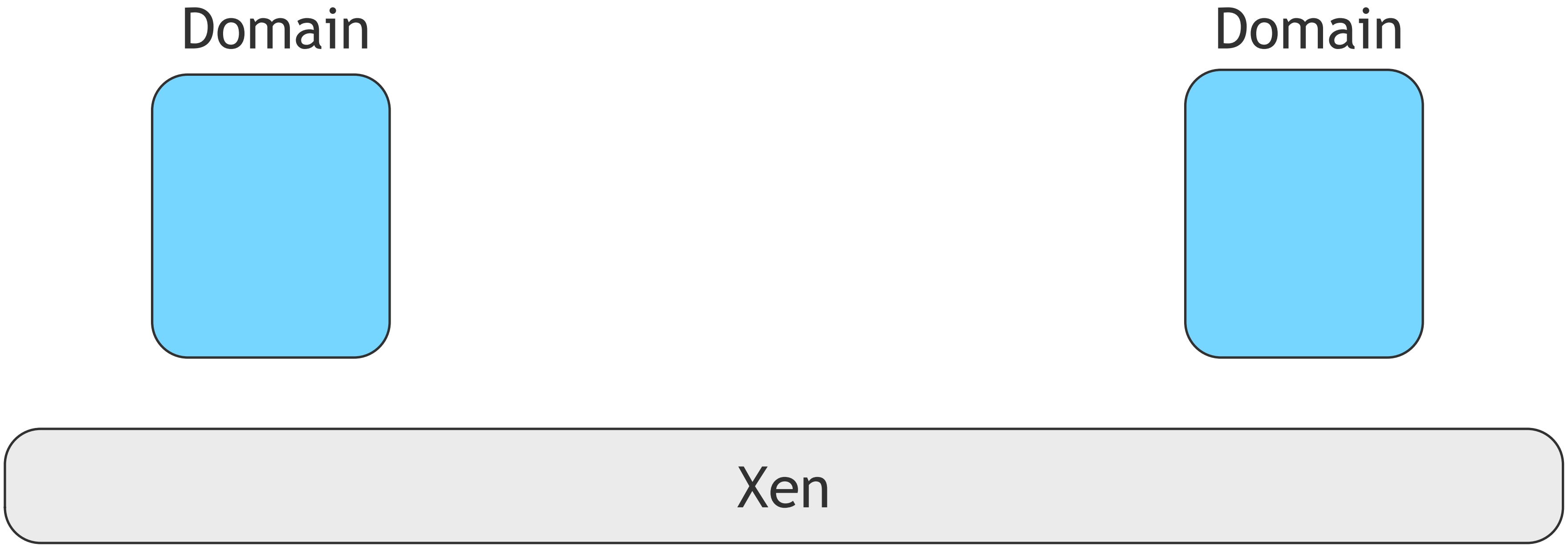
Our Vision And The HaLVM



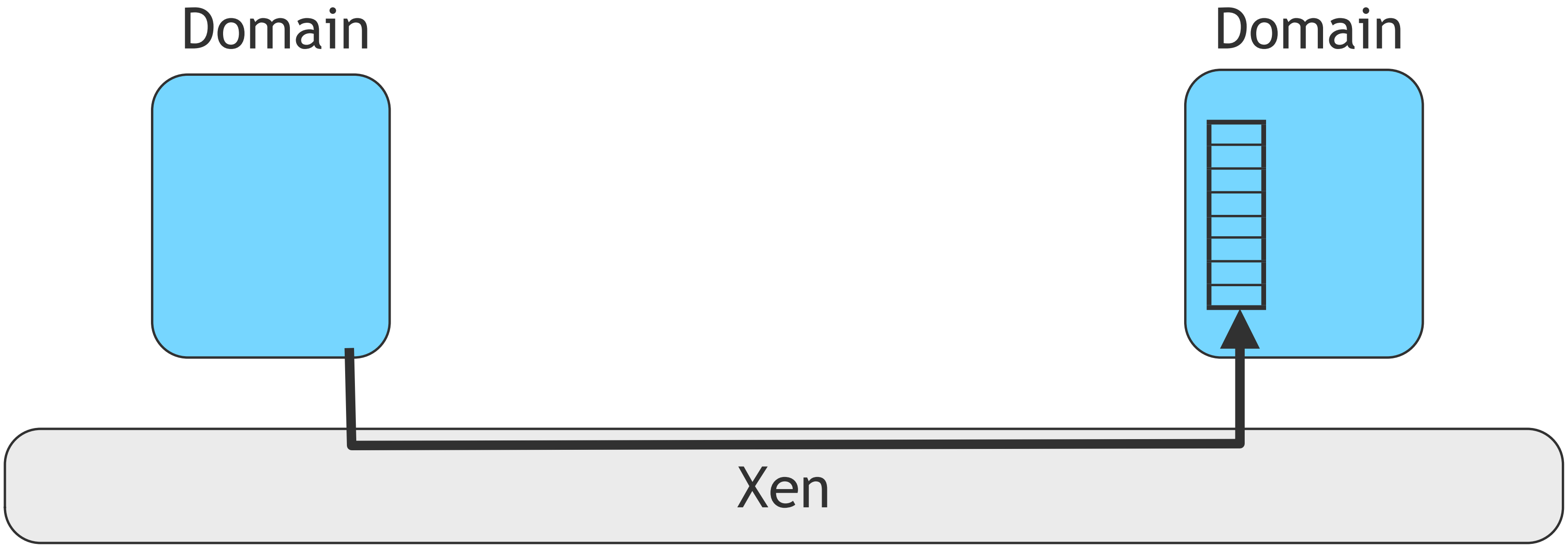
Our Vision And The HaLVM

- Hypervisors allows clean separation between concurrently running operating systems.
  - Use as a prototyping target, with “high-assurance” prototypes running concurrently with low assurance, general-purpose operating systems.
- Hypervisors also allows collaborating running components to communicate with each other.
  - Decompose the kernel drivers and library functionality into separate components with restricted channels of communication.
- We want to prototype and test these components quickly and easily.

Xen In Two Minutes



Xen In Two Minutes



Domains can share memory via *grant references*.

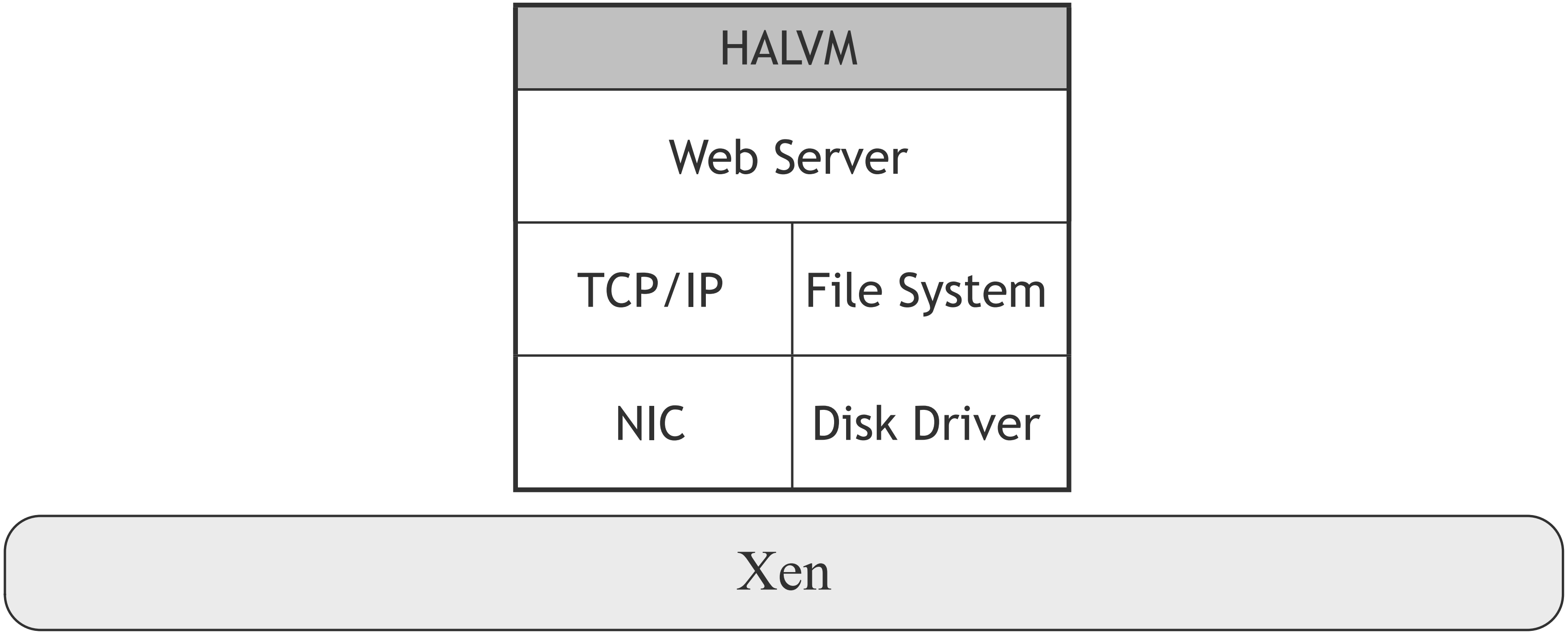
Xen In Two Minutes

- There are two kinds of virtualization:
  - *Full virtualization*: Uses hardware or software to run an *unmodified* operating system.
  - *Paravirtualization*: Requires the programmer to modify the operating system before running it.
- Xen supports both kinds of virtualization
- The HaLVM is a lightweight, paravirtualized Xen guest

Lightweight Virtual Machine

- The purpose of the HaLVM is ...
  - Exploration of the design space for a decomposed, high-assurance operating system.
  - A sandbox for experimenting with OS components.
- The HaLVM is implemented as a series of libraries built upon a core port to Xen.
  - For example, one library implements basic memory routines, the next level uses that to implement a disk driver, the next level uses the driver to implement a file system.
  - Programmers can pick and choose library routines at any level, and libraries that are not used are not linked in.
- HaLVM programs typically boot in under a second.
- HaLVM programs are typically small:
  - 1-2 MB for the complete image.
  - 3-5 MB initial memory size.

Use Case: Web Server



HALVM

Web Server

TCP/IP

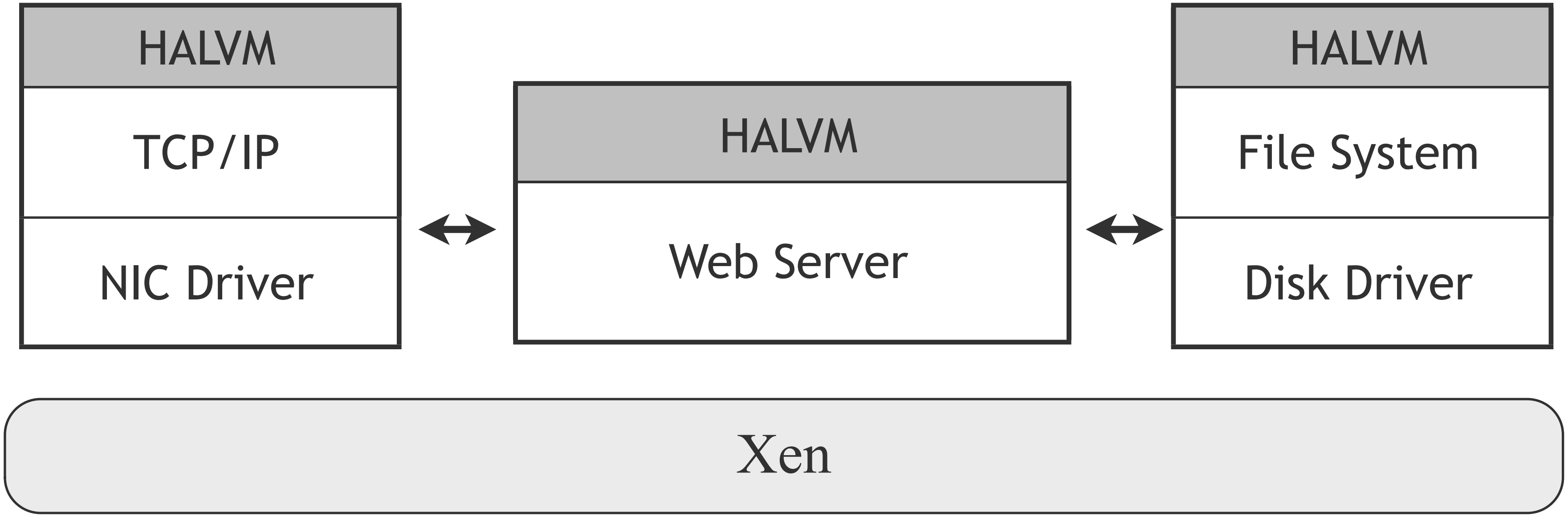
File System

NIC

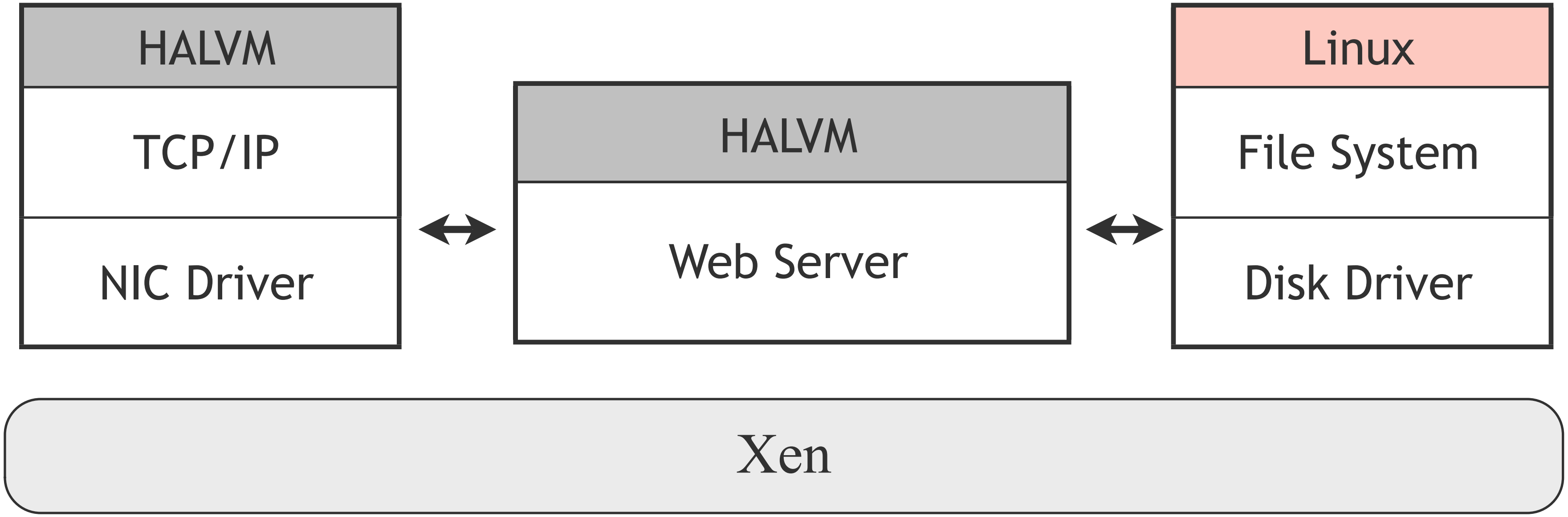
Disk Driver

Xen

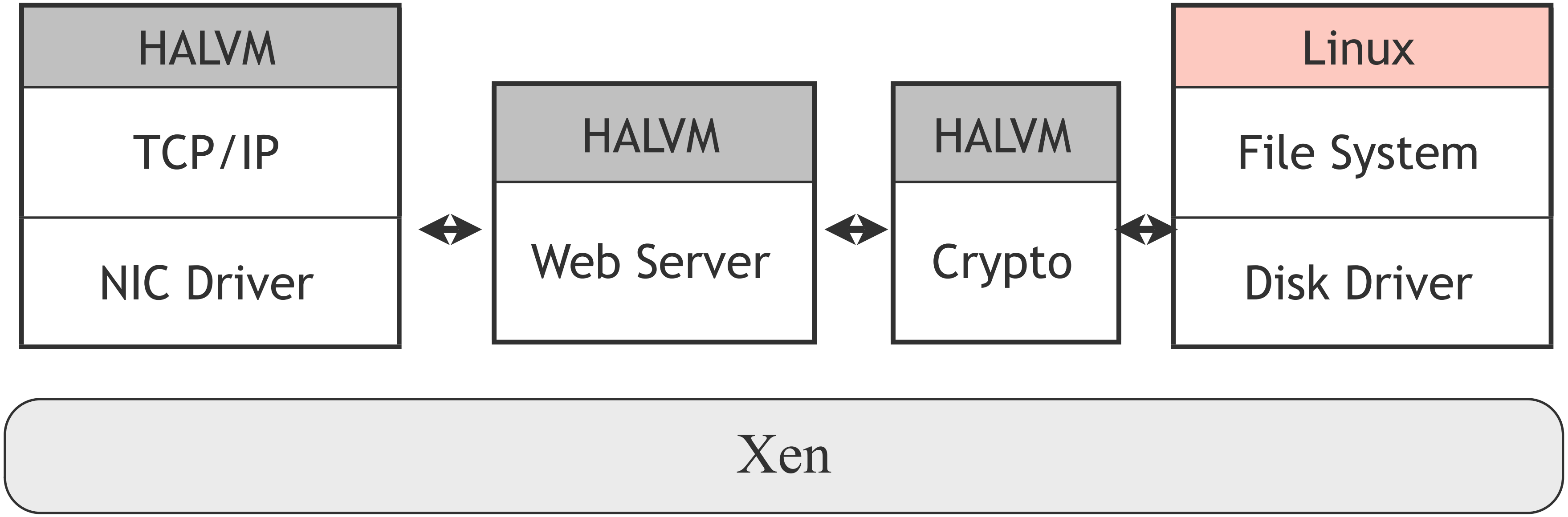
Use Case: Web Server



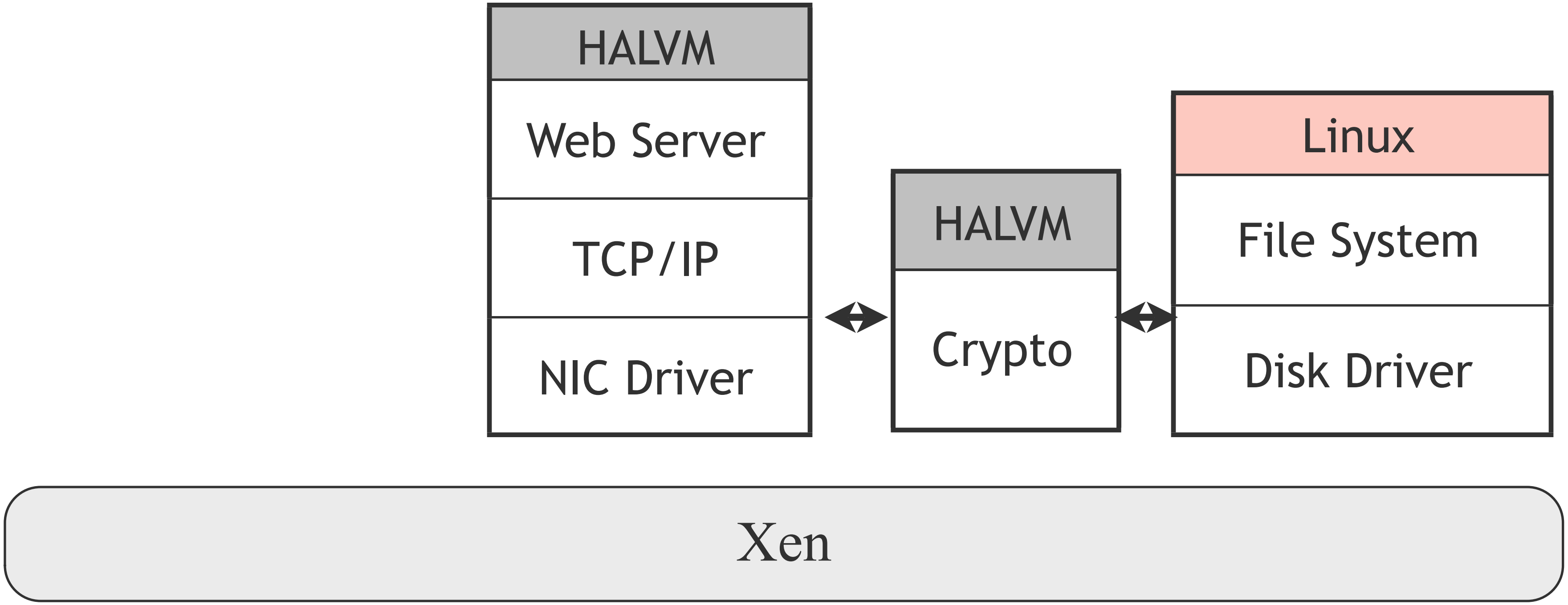
Use Case: Web Server



Use Case: Web Server



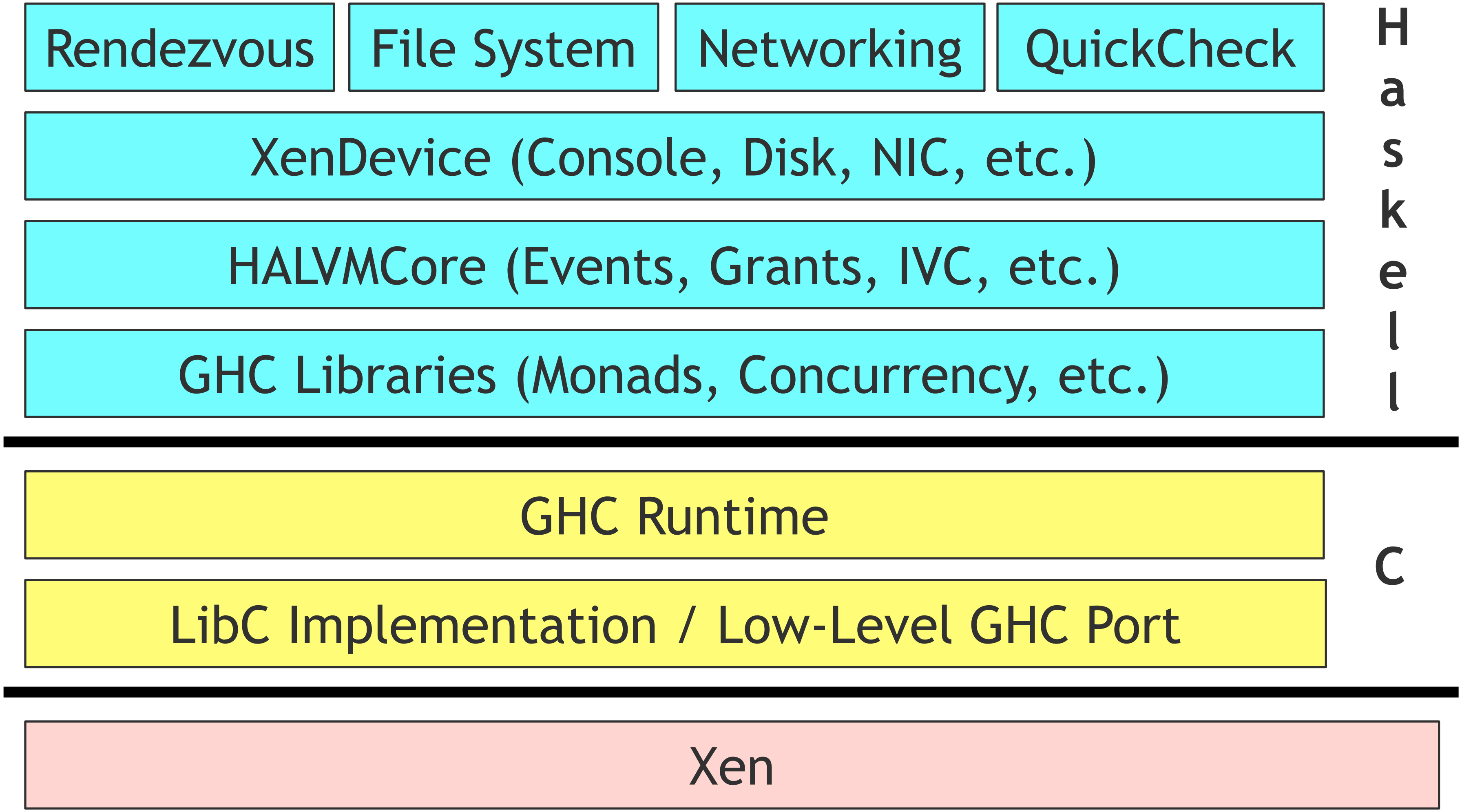
Use Case: Web Server



Talk Outline

- Introduction
- Related Projects and Background
- An Overview of the HaLVM
  - An overview of the libraries
  - Some additional features
  - ... with demos all around
- Current Gaps and Future Work
- More Demos (as time permits)

High-Level Architecture



Rendezvous

File System

Networking

QuickCheck

XenDevice (Console, Disk, NIC, etc.)

HALVMCore (Events, Grants, IVC, etc.)

GHC Libraries (Monads, Concurrency, etc.)

Haskell

GHC Runtime

LibC Implementation / Low-Level GHC Port

C

Xen

HaLVM Libraries - HALVMCore

- As suggested by the name, core libraries for implementing Xen virtual machines:
  - Event channel manipulation
    - Send and receive cross-domain events.
  - Basic Memory Management
    - Allocate and free pages, query the page tables, etc.
  - Grant table manipulation
    - Allows page sharing, transferring, and copying between domains.
  - Inter-VM Communication (IVC)
  - Useful utility types, routines, and data structures
    - Write to the debug console, manipulate MBufs, perform privileged operations, correct initialization, etc.
- First demo (XenstoreC)

HaLVM Libraries - XenDevice

- Device drivers for the basic XenDevices:
  - The virtualized console
  - The virtualized disk(s)
  - The virtualized network card(s)
  - The XenStore
- The disk and network card drivers use the same protocol for talking to the underlying "device".
  - The HaLVM exports the implementation of this protocol to programmers.
  - This allows for rapid development of standard device drivers.
- Second demo (DoubleDevice)

HaLVM Libraries - RendezvousLib

- This library allows for the simple creation of well-typed, unidirectional channels between domains.
- Declare a new channel as follows:

```
offer :: IO (OutChannel type)
accept :: IO (InChannel type)
(offer, accept) = p2pConnection “SpeedTest”
```

- Create the receiver endpoint as follows:

```
writeChan <- offer
```

- By splitting this out into a library, we got rid of a lot of boilerplate (and potential for bugs!) plus got handy typing properties.
- DoubleDevice demo, part two

HaLVM Libraries - Halfs, Hans, etc.

- **Halfs**
  - A full-featured file system, written in Haskell, ported to the HaLVM.
  - Via the FUSE architecture, can be mounted in domain 0, as well.
- **Hans** (in progress)
  - A network stack, written in Haskell, ported to the HaLVM.
  - Will include TCP, IP4, IP6, DNS, DHCP, and other acronyms.
- **QuickCheck**
  - An automated testing library, standard in most Haskell implementations, ported to the HaLVM.
  - Useful for running tests that require the libraries to be tested in their final environment.

Useful Details - halvm_kernel

- Automated handling of device initialization order, which is sometimes non-intuitive.
  - The programmer need not know that the disk driver requires the XenStore driver, let alone that the XenStore driver must be initialized first.
  - Additional driver libraries can be plugged into this infrastructure, as well; they simply implement a data structure describing how to initialize the device, how to shut it down, and what its dependencies are.
- Also handles exceptions that escape the main program, and performs safe shutdown if one is detected.
- Some more of what I didn’t show you before …

Useful Details - BitFiddler

- As suggested by the name, a library for very low-level data layout issues.
- Defines a type class for performing host and network order conversions.
- Defines a Template Haskell macro for defining and manipulating structures with bit fields.
  - No more computing offsets via scratch paper and RFCs.
  - No more endless lists of poke/peek helper functions.
  - Generates getters and setters, performs network/host order conversions on request, computes the total size of the structure in bytes.
- Another demo (ARPSniffer)

Current Gaps

- The HaLVM is a work-in-progress, and we have often traded breadth in the underlying system for depth in the support libraries. For example:
  - The HaLVM is x86 only.
  - The HaLVM is 32-bit only (no PAE, no 64-bit).
  - The HaLVM is uniprocessor only.
- The HaLVM is a prototyping system, not a final execution environment.
- As of Xen version 3.0.4, using inter-virtual machine communication requires a minor patch and recompilation of Xen.
- Debugging operating system components is hard; there is lots of opportunity to improve support for debugging.

Current Activities

- At this point, we have mostly stopped adding features and are working on finishing support we've already begun:
  - Hans (the network stack)
  - A testing framework for regression testing
  - vTPM driver support
- Plus standardizing a few of our interfaces, documentation clean-ups, and so forth.
- We intend to open source the HALVM.
  - Come talk to us if you're interested in playing around with it!

As There Is Time

More Demos!