Porting HelenOS to RISC-V

http://d3s.mff.cuni.cz

Martin Děcký

decky@d3s.mff.cuni.cz

CHARLES UNIVERSITY IN PRAGUE

Faculty of Mathematics and Physics

Department of Distributed and Dependable Systems

D3S

Introduction

Two system-level projects

- **RISC-V** is an instruction set architecture, **HelenOS** is an operating system

Introduction

Two system-level projects

- **RISC-V** is an instruction set architecture, **HelenOS** is an operating system
- Both originally started in academia
  - But with real-world motivations and ambitions
- Both still in the process of maturing
  - Some parts already fixed, other parts can be still affected

Introduction

Two system-level projects

- **RISC-V** is an instruction set architecture, **HelenOS** is an operating system
- Both originally started in academia
  - But with real-world motivations and ambitions
- Both still in the process of maturing
  - Some parts already fixed, other parts can be still affected

→ **Mutual evaluation of fitness**

Introduction

Martin Děcký

- Computer science researcher
  - Operating systems
  - Charles University in Prague
- Co-author of HelenOS (since 2004)
  - Original author of the PowerPC port


RISC-V

RISC-V in a Nutshell

Free (libre) instruction set architecture

- BSD license, in development since 2014
- **Goal:** No royalties for analyzing, designing, manufacturing and selling chips (and related software)
- http://riscv.org

RISC-V in a Nutshell (2)

Based on reduced instruction set (RISC) principles

- Design based on time-proven ideas, but avoiding mistakes and anachronisms
  - High-performance *Z-scale*, *Rocket* and *BOOM* prototype chips manufactured (at 45 nm and 28 nm)
- Scalable and extendable clean-slate design
  - From small embedded systems (low-power minimal 32-bit implementations)
  - To large computers (powerful implementations, 64-bit or even 128-bit, SIMD, VLIW, etc.)

Context of RISC-V

Comparison with previous free ISAs

- OpenRISC, OpenSPARC, LaticeMico32, MMIX, Amber, LEON
- **Goals of RISC-V**
  - BSD instead of GPL
  - Modularity and scalability
  - Supporting 32 bits and 64 bits (128 bits in the future)
  - Not just for research and education

Context of RISC-V (2)

Comparison with commercial ISAs

- IA-32, AMD64, IA-64, ARM
  - Complex or rather complex
  - Intellectual property minefield
- SPARC, POWER, MIPS
  - Still intellectual property minefield
- Old patent-free ISAs
  (ARMv2, Berkeley RISC, Stanford MIPS)
  - Obsolete

Context of RISC-V (2)

Comparison with commercial ISAs

- **Goals of RISC-V**
  - Free
  - Relevant
  - State-of-the-art
  - Practical
  - Reasonable complexity

Who Is behind RISC-V

UC Berkeley, Computer Science Division

- **Krste Asanović**
  - Principal designer
- **David Patterson**
  - Coined the term RISC and led the original Berkeley RISC project (1980)
    - Later exploited in SPARC, Alpha and ARM
  - Co-author of DLX (with John Hennessy for *Computer Architecture: A Quantitative Approach*)
- Funding from DARPA, Intel, Microsoft and others

Who Is behind RISC-V (2)

RISC-V Foundation

- Non-profit corporation
  - Rick O'Connor (Executive Director)
  - http://riscv.net
- Governing the evolution of RISC-V ISA
  - Standardization of extensions
  - Intellectual property matters (patents, logos, trademarks, etc.)
- Founding members
  - Google, Hewlett Packard Enterprise, Lattice, Oracle, lowRISC and others

Implementors of RISC-V

Indian Institute of Technology Madras

- Plan to produce six CPU designs based on RISC-V

Bluespec

- Preliminary plan to produce RISC-V based CPUs

lowRISC

- Non-profit organization (cooperating with University of Cambridge and Raspberry Pi Foundation)
- Implementing open source SoC based on 64 bit RISC-V (scheduled for 2017)
- Tagged memory

Features of RISC-V

RISC architecture

- Word size of 32 or 64 bits
  - Word size of 128 bits possible in the future
- 32 general-purpose registers (word-sized)
  - Load/store architecture
  - R0 always contains 0
  - 8 bit and 16 bit arithmetics via sign extension
  - Plain register file (no register windows, register stacks, etc.)
- Optional 32 floating point registers (IEEE 754)
- Little-endian, byte-addressable memory

Features of RISC-V (2)

RISC architecture

- 32-bit instructions
  - Orthogonal instruction set
  - Limited number of instruction templates
    - Fixed positions for specific opcode bits for fast decoding and immediate argument sign extension
  - Three-argument instructions
  - Synthetic instructions
    - R0 used to provide two-argument synthetic instructions
  - Implicit stack
  - Mandatory alignment of memory accesses

Features of RISC-V (3)

Beyond RISC

- No branch delay slots
- No condition codes, flag registers, carry bits
  - Conditions evaluated in branch instructions
- Practical design of instruction encoding
  - Opcode bits designed to reduce the number of multiplexers

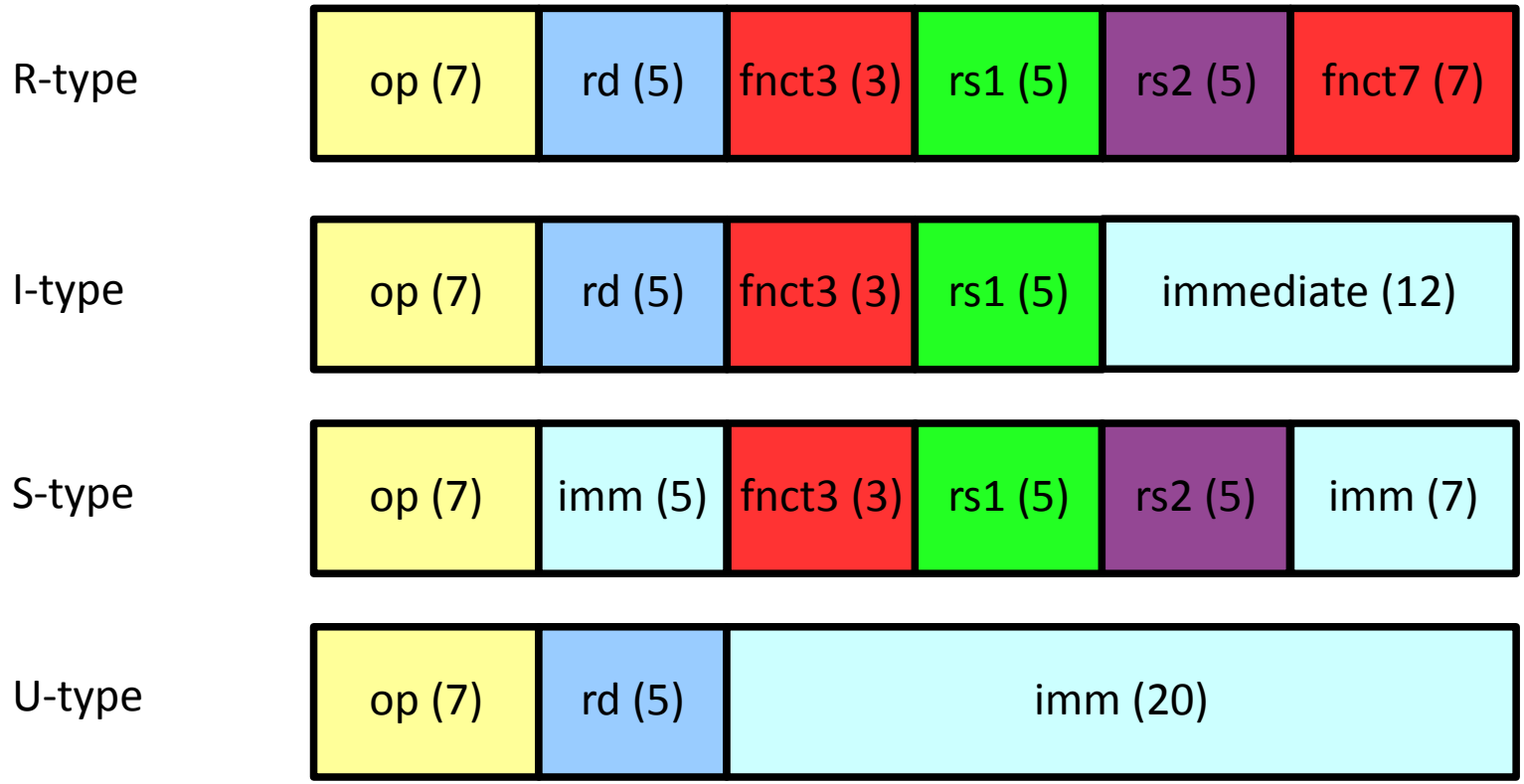
Features of RISC-V (4)

Beyond RISC

- Native support for position-independent code
  - Address calculation relative to the program counter
- Fused multiply-add by future accelerated decoding
- Instruction set extentions
  - Mandatory instruction set
  - Optional (non-conflicting) extentions
    - Green-field and brown-field allocations

RISC-V Instruction Set

RV32I (Base Integer Instruction Set)



RISC-V Instruction Set (2)

RV32I (Base Integer Instruction Set)

- Computational
  - addition, subtraction, set less than, and, or, xor, shift left logical, shift right logical, shift right arithmetic, load upper immediate, add upper immediate to PC, no-op
- Control transfer
  - unconditional jump (with link), branch (equal, not equal, less than, greater or equal)
- Load and store
  - load, store, memory fence
- System
  - system call, breakpoint, CPU cycles, retired CPU instructions, wall-clock time

RISC-V Instruction Set (3)

RV64I (Base Integer Instruction Set)

- Essentially the same instructions as in RV32I (some variations accommodating the 64-bit word size)

Standard extensions

- M (Integer Multiplication and Division)
- A (Atomic Instructions)
  - load-reserved, store-conditional, atomic memory operation (swap, addition, and, or, xor, max, min)
- F (Single-Precision Floating Point)
- D (Double-Precision Floating Point)

General purpose ISA: RV64IMAFD = RV64G (164 instructions)

RISC-V Instruction Set (4)

More standard extensions

- Q (Quad-Precision Floating Point)
- D (Decimal Floating Point)
- C (16-bit Compressed Instructions)
- B (Bit Manipulations)
- T (Transactional Memory)
- P (Packed-SIMD)

RV128I (Base Integer Instruction Set)

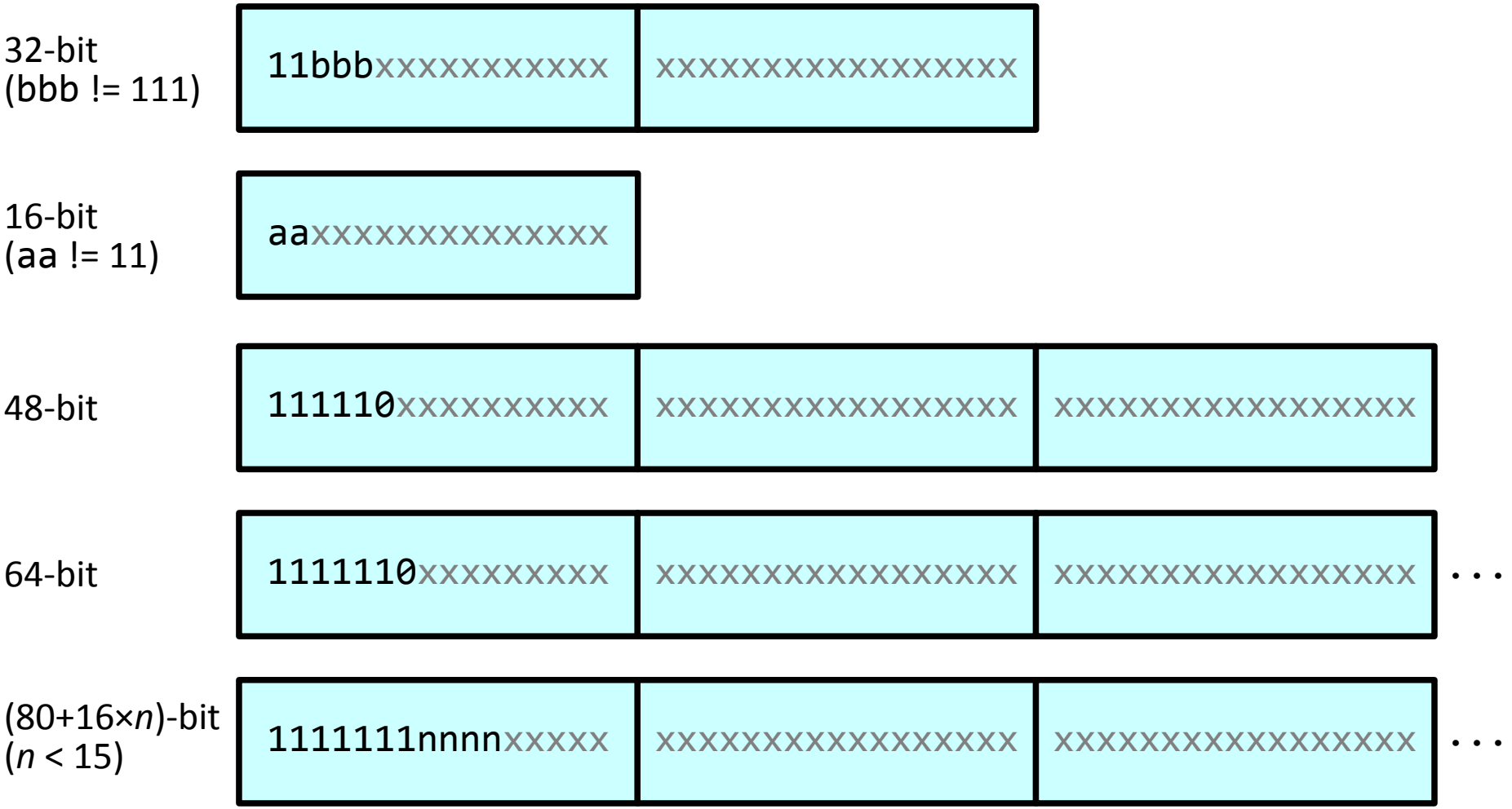
- Sketched

Features of RISC-V (5)

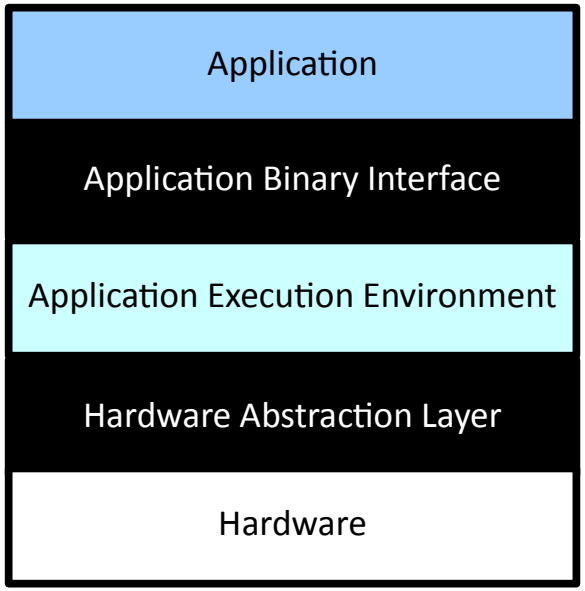
Beyond RISC

- Reserved opcodes for non-32-bit instructions
  - 64-bit instructions, variable-length instructions, even instruction bundles (VLIW)
  - Compressed instruction set
    - 16-bit instructions
    - No separate execution mode necessary (intermixed with 32-bit instructions)
    - Code on average 20 % smaller than x86, 2 % smaller than ARM Thumb-2

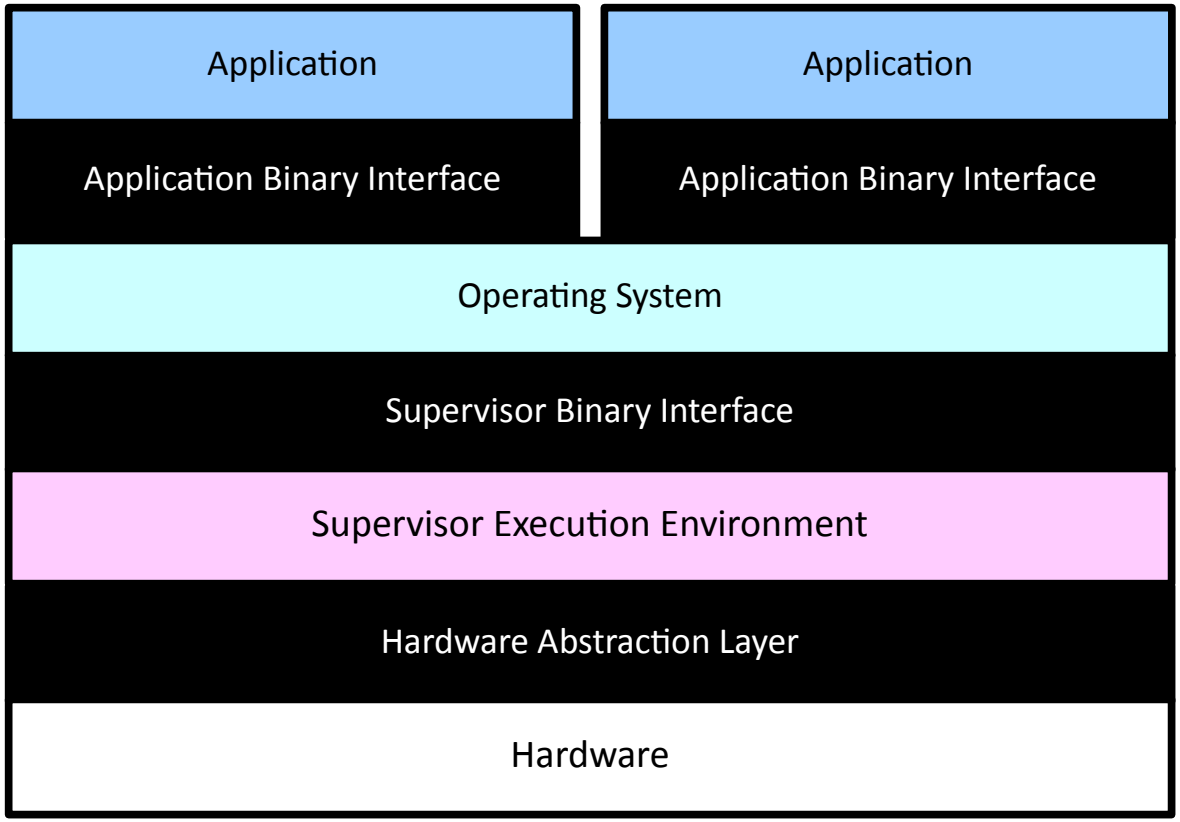
RISC-V Instruction Length



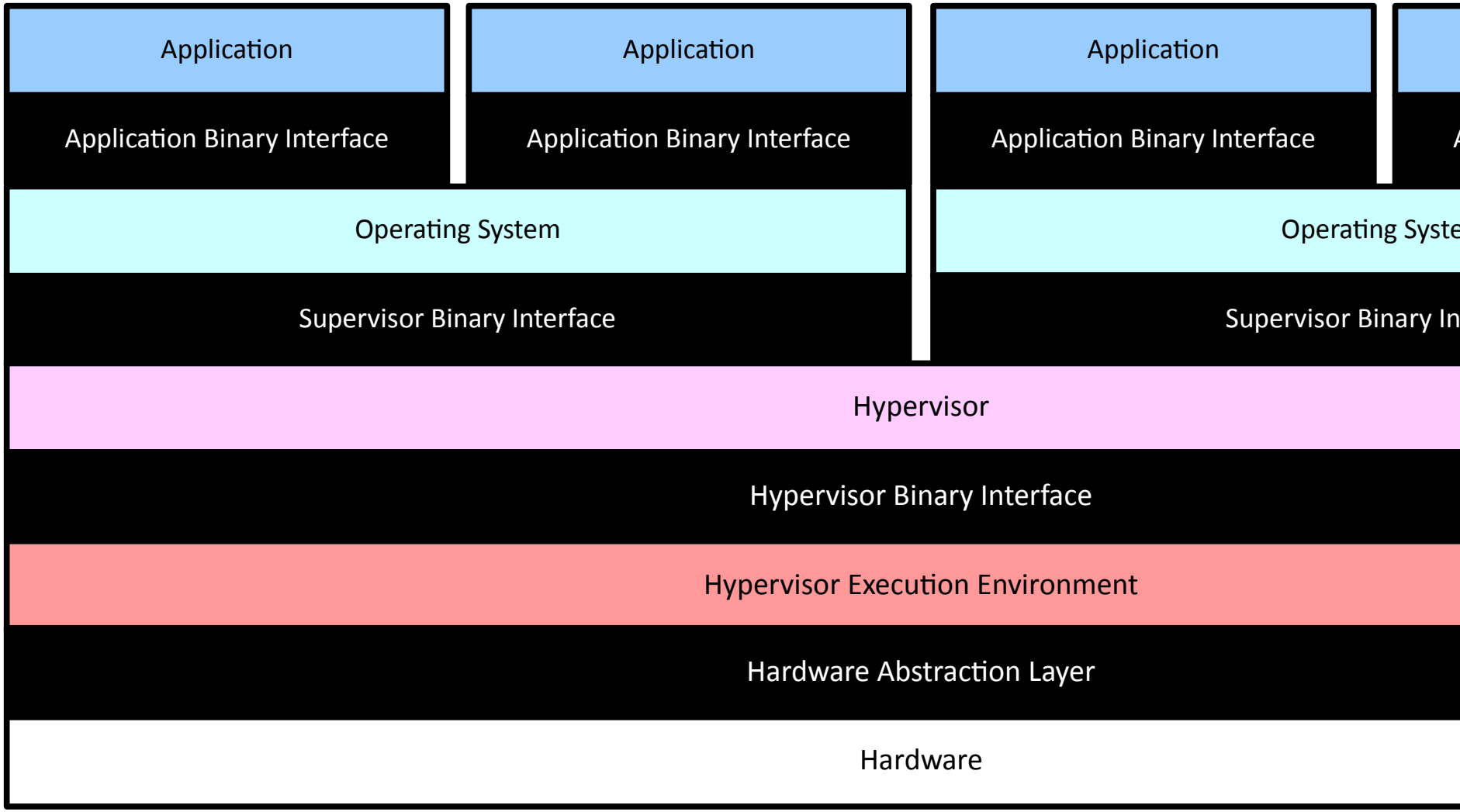
RISC-V Privileged Software Stacks



RISC-V Privileged Software Stacks



RISC-V Privileged Software Stacks



RISC-V Privileged Software Stacks

Privilege Levels

- Machine (M)
- Hypervisor (H)
- Supervisor (S)
  - Same set of privileged instructions
- User/Application (U)
  - Each level own set of Control and Status Registers (CSR)
    - 4×1024 I/O space

Supported combinations

- M
  - Embedded systems
- M+U
  - Embedded systems with protection
- M+S+U
  - Standard OS
- M+H+S+U
  - Standard OS with virtualization

RISC-V Machine-Level ISA

CSRs

- CPU and hardware thread ID
- Machine status (privilege level, interrupts)
- Memory management status
  - No memory translation
  - Single base-and-bound
  - Separate instruction/data base-and-bound
  - 32-bit virtual addresses (2-level hierarchical page tables)
  - 39-bit virtual addresses (3-level hierarchical page tables)
  - 48-bit virtual addresses (4-level hierarchical page tables)

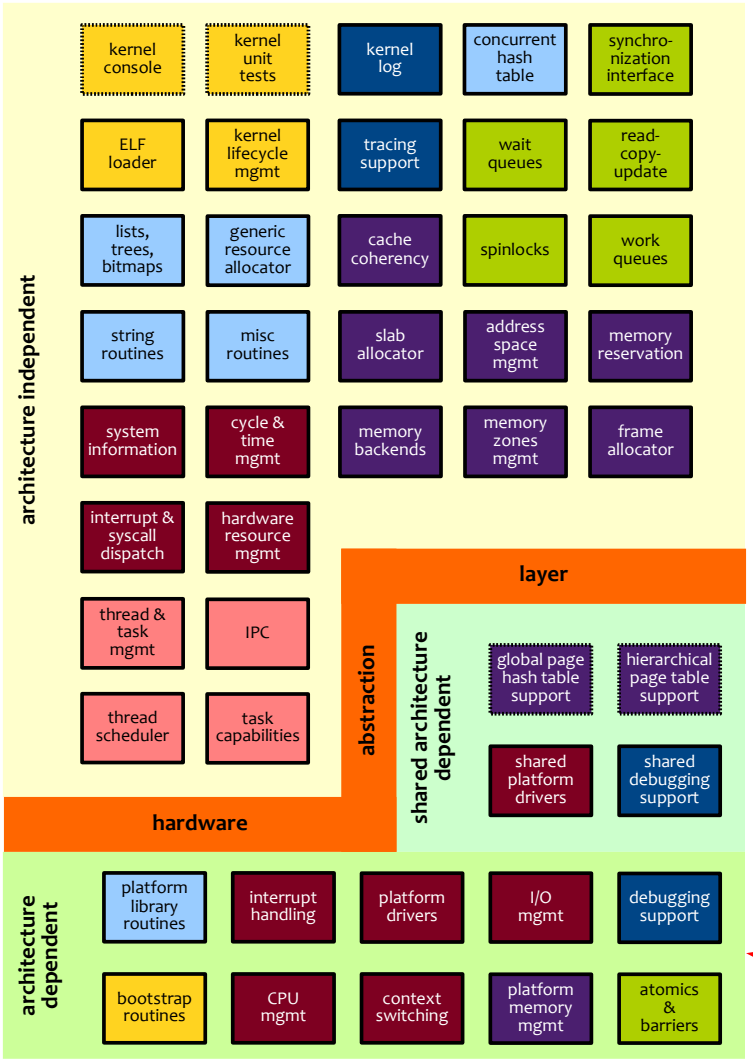
HelenOS

HelenOS in a Nutshell

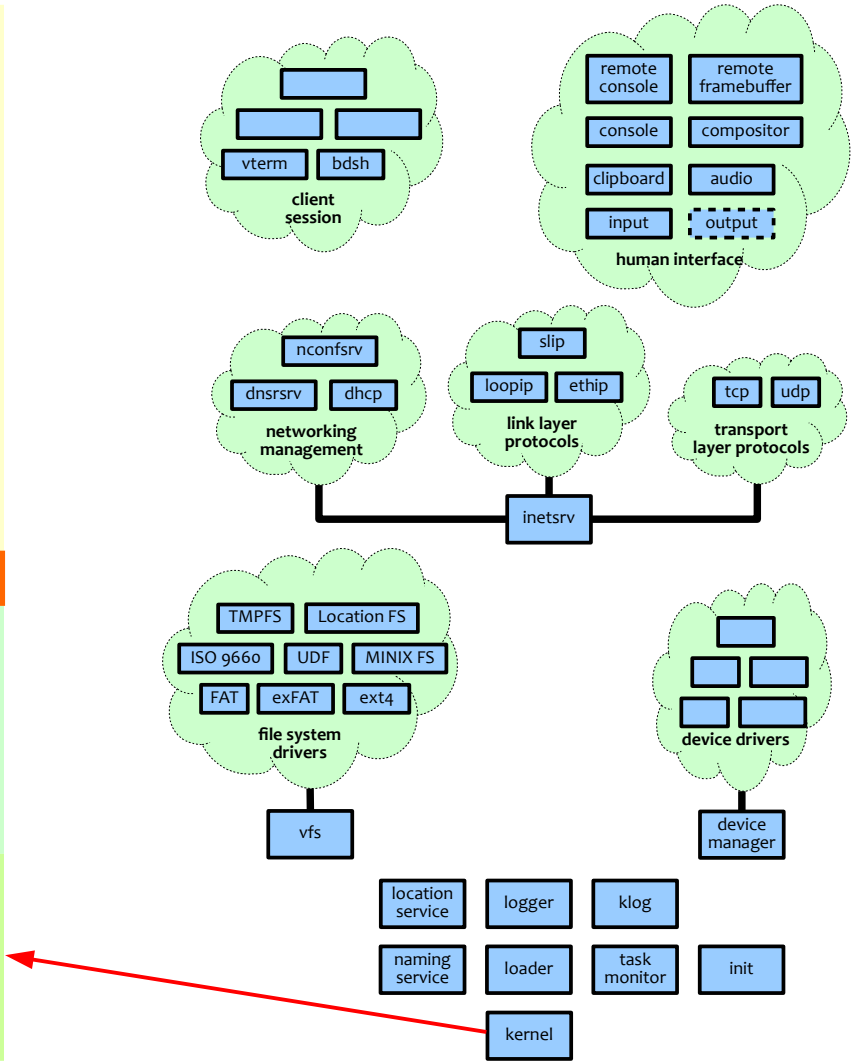
Microkernel multiserver operating system

- BSD license, in development since 2004
- **Goal:** General-purpose usability, not limited by any specific use case or hardware platform, component-based design and implementation
- http://helenos.org

HelenOS in a Nutshell (2)



Kernel subsystems



System components

Features of HelenOS

Design based on explicit design principles

- Non-fundamentalistic metaprinciple
- General-purpose principle
- Microkernel principle
- Full-fledged principle
- Multiserver principle
- Split of mechanism and policy principle
- Encapsulation principle
- **Portability principle**
- Modularity principle

Portability of HelenOS

Supporting 8 hardware platforms

- IA-32, AMD64, IA-64, ARM, PowerPC, MIPS, SPARC V8, SPARC V9
- Portability case studies
  - Port to ARM in 53 days by 3 developers
  - Port to SPARC V8 in 13 weeks by 1 developer
- Porting efforts improve portability
- Portability simplifies future porting efforts

Portability of HelenOS (2)

Portability design principle

- "Do not be biased by any single hardware platform"
  - Reusable abstract algorithms
    - 4-level page tables
    - ASID LRU management
    - interrupt routing
  - Hardware Abstraction Layer defined by a virtual `abs32le` port

HelenOS abs32le

```
NO_TRACE static inline void atomic_inc(atomic_t *val)
{
#ifdef CONFIG_SMP
    asm volatile (
        "lock incq %[count]\n"
        : [count] "+m" (val->count)
    );
#else
    asm volatile (
        "incq %[count]\n"
        : [count] "+m" (val->count)
    );
#endif /* CONFIG_SMP */
}
```

Atomic increment on IA-32

HelenOS abs32le

semantic annotations

```
NO_TRACE ATOMIC static inline void atomic_inc(atomic_t *val)
    WRITES(&val->count)
    REQUIRES_EXTENT_MUTABLE(val)
    REQUIRES(val->count < ATOMIC_COUNT_MAX)
{
    /*
     * On real hardware the increment has to be done
     * as an atomic action.
     */

    val->count++;
}
```

plain C behavior summary

Atomic increment behavior summary on `abs32le`

HelenOS RISC-V Port

Reasons

- Future combined verification of HW/SW correctness

Current status

- Started in **January 2016**
- **Finished:** Boot loader, initial memory management setup, kernel hand-off
  - Everything compiles
  - Memory management data structures in place
  - 18 hours (net)

Demo

HelenOS RISC-V Port (2)

General approach

- Cloning the `abs32le` virtual port
  - Changing names, endianity, word width, primitive types, linker script and other basic definitions
- Adding the new platform to the build system
  - Adding basic options to `HelenOS.config`
  - Adding the compiler toolchain to `tools/autotool.py`
- Checking that everything compiles (mostly trivial)
- Gradually adding actual working implementation

HelenOS RISC-V Port (3)

```
#define EM_RISCV  243  /* RISC-V */
```

abi/include/abi/elf.h

```
.org DEFAULT_MTVEC + TRAP_VECTOR_RESET
start:
    /* Set up stack, create stack frame */
    la sp, boot_stack + BOOT_STACK_SIZE
    addi sp, sp, -16

    j bootstrap
```

boot/arch/riscv64/src/asm.S

HelenOS RISC-V Port (3)

```
#define ADDRESS_SPACE_HOLE_START  UINT64_C(0x0008000000000000)
#define ADDRESS_SPACE_HOLE_END    UINT64_C(0xffff7fffffffffff)

#define KERNEL_ADDRESS_SPACE_SHADOWED_ARCH  0

#define KERNEL_ADDRESS_SPACE_START_ARCH  UINT64_C(0xffff800000000000)
#define KERNEL_ADDRESS_SPACE_END_ARCH    UINT64_C(0xffffffffffffffff)
#define USER_ADDRESS_SPACE_START_ARCH    UINT64_C(0x0000000000000000)
#define USER_ADDRESS_SPACE_END_ARCH      UINT64_C(0x00007fffffffffff)
```

kernel/arch/riscv64/include/arch/mm/as.h

```
/** Page Table Entry. */
typedef struct {
    unsigned long valid : 1;       /**< Valid bit. */
    unsigned long type : 4;        /**< Entry type. */
    unsigned long referenced : 1;  /**< Refenced bit. */
    unsigned long dirty : 1;       /**< Dirty bit. */
    unsigned long reserved : 3;    /**< Reserved bits. */
    unsigned long pfn : 54;        /**< Physical frame number. */
} pte_t;
```

kernel/arch/riscv64/include/arch/mm/page.h

HelenOS RISC-V Port (4)

Near future

- Basic kernel functionality
  - Interrupt/exception handling
  - Context switching, atomics
  - **ETA:** 18 – 24 hours (net)
- Basic user space functionality
  - Thread-local storage
  - User space context switching
  - I/O
  - **ETA:** 18 – 24 hours (net)

Current State of RISC-V

User-level ISA

- Version 2.0, frozen since May 6th 2014

Compressed ISA

- Version 1.9, draft, to be frozen soon

Privileged ISA

- Version 1.7, draft, expected to be frozen in mid-2016 or later

Vector ISA

- Only sketched so far

Current State of RISC-V (2)

Holes in the specification

- Memory consistency model
- Application Binary Interface (ABI)
- Performance counters
- Hypervisor support
- Formal specification (for verification)

Current State of RISC-V (3)

Holes in the specification

- Reference platform
  - Standard I/O locations (standard memory map)
  - Debugging support (hardware breakpoints, JTAG)
  - Interrupt controller, timer, RTC, reset mechanism, DMA, IOMMU
  - Power management
  - Standard firmware (standard device tree)
  - Standard boot loader

Current State of RISC-V (4)

Available tooling

- Development tools
  - GNU Binutils, GCC, GDB, LLVM, clang, Go, libffi
- Enviroments
  - Newlib, glibc
- Simulators / Emulators
  - Spike (MSIM-like), QEMU (80 % of privileged ISA), ANGEL (JavaScript), *Simics*, *trace32*

Not upstreamed yet

Current State of RISC-V (5)

Software stack

- Firmware
  - coreboot (on Spike, upstreamed)
  - UEFI (a hack of EDKII on QEMU with PC peripherals)
- Operating systems
  - Linux (with busybox), Yocto/OpenEmbedded
  - FreeBSD, NetBSD (to be upstreamed soon)
  - seL4
  - HelenOS (work-in-progress)

Preliminary Porting Observations

RISC-V is like a mixture of MIPS and AMD64

- HelenOS generic 4-level page tables suitable for RISC-V
  - RISC-V compressed access permission field in page tables is more cumbersome than access bits
- No forced platform-independent code change expected
- HelenOS and RISC-V evaluate as good match for each other

Preliminary Porting Observations

RISC-V underspecification causes only minor issues

- Generally, the CPU specs are fine
- "Reference platform" documented only in the code of the Spike simulator
  - E.g. *host-target interface* implementing basic I/O devices using CSRs

Conclusion

RISC-V

- Interesting and solid research/development effort
- Great potential for both academia and industry

HelenOS

- RISC-V port underway, no roadblock in sight
- History of HelenOS portability improves future portability

Q&A

www.riscv.org

www.helenos.org

Backup slides

Context of RISC-V

Not the first free ISA, but the most degrees of freedom

- OpenRISC
  - 2000, LGPL, based on DLX, fixed ISA (not extendable)
- OpenSPARC
  - 2005, GPL, based on UltraSPARC T1 (SPARC V9), fixed configuration (i.e. number of cores, etc.)
- LatticeMico32
  - 2006, custom open source license (a mixture of several licenses), 32-bit

Context of RISC-V (2)

Not the first free ISA, but the most degrees of freedom

- MMIX
  - 2009, a non-free open source license, Donald Knuth, John Hennessy and Richard Sites (inspired by Knuth's MIX), 64-bit, for educational purposes
- Amber
  - 2010, LGPL, ARMv2 compatible (32-bit)
- LEON
  - 2000-2010, LGPL + dual licensing, SPARC V8 compatible (32-bit)

RISC-V in Academia

Highlights of research projects based on RISC-V

- Formal specification & verification
  - Chisel open source hardware construction language [UC Berkeley]
  - Novel bespoke weak memory consistency models [MIT]
- Novel manycore configurations [Gray Research]
- First experimental photonic CPU [UC Berkeley]
- Experimental ISA extensions [ETH Zürich]
  - Hardware loops
- Faster ISA simulators using JIT [Cornell]