Unleashing RISC-V in Managed Runtimes

Robbin Ehn

Overview

- OpenJDK
- Just-In-Time compilation
- Trampolines
- Cross modifying code
- Extensions
- Sign extension
- Canonical NaN

OpenJDK

- HotSpot (https://openjdk.org/)
  - C++ code base
    - Inline assembly
  - Template interpreter
    - Assembly snippets
  - C1 compiler
  - C2 compiler

RISC-V Port

- Fully functional
- Optimization
- Limited testing
  - Hardware
  - Qemu

- JDK 21
- JDK 17
  - JDK 11 WIP

Just-In-Time

- Why?
  - Dynamic class hierarchy
    - First use
    - Classloading
  - Virtual
    - Default
  - Profiling

Just-In-Time

- When ?
  - Hot methods
  - C1 -> C2

- Speculative compilation
  - Profiling
  - Heavy inlining
  - Cross modifying code
  - Traps
  - Deoptimization

JIT:ed callsite

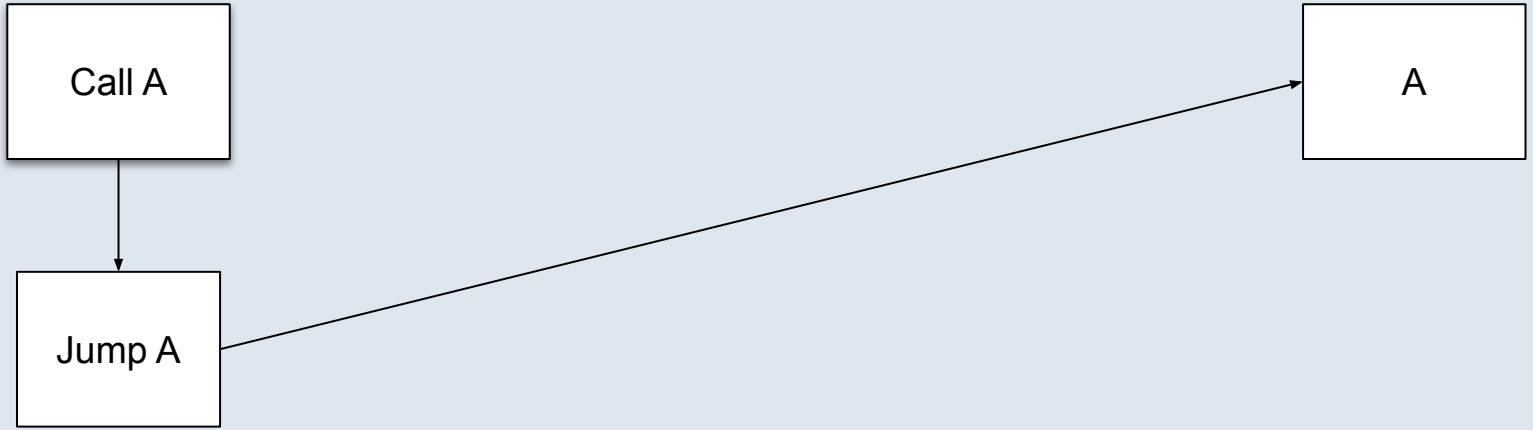
- Destination unresolved
  - Jump-And-Link
  - Jump-And-Link-Register

- Full range call
  - Address
  - ld - load 8 bytes
  - li - materialize

```
“Load Imm”
lui     a0, 357749
addiw   a0, a0, 1879
slli    a0, a0, 12
addi    a0, a0, 1397
slli    a0, a0, 1
```

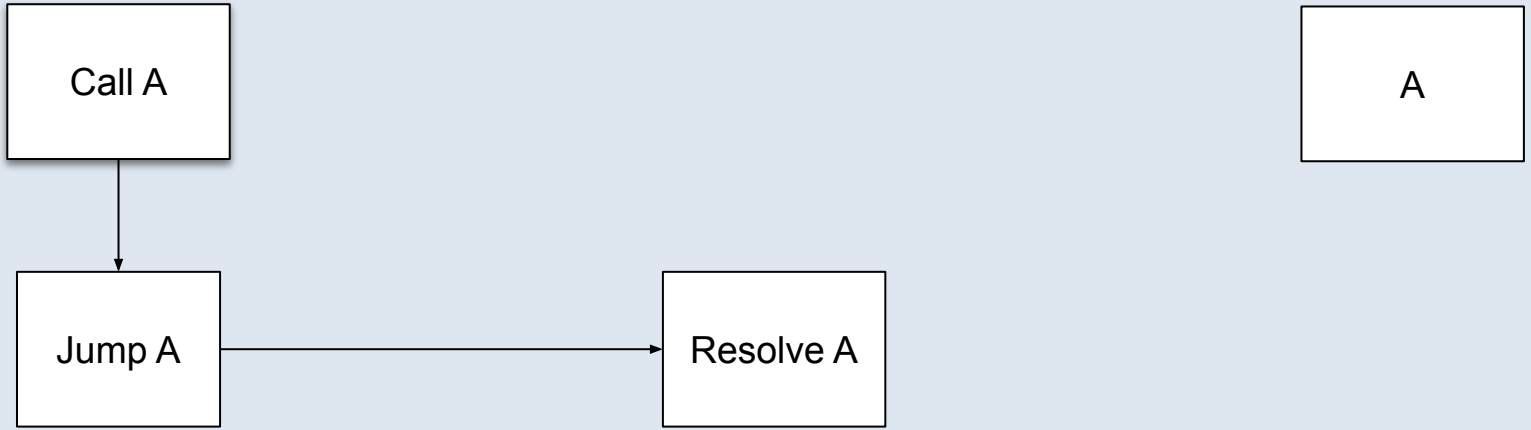
Trampoline

- Call to a jump
  - auipc + jalr(call trampoline)
  - auipc + ld + jalr(jump) + <address>



JIT

- Callsites unresolved



Cross modifying code

- Thread one
  - Modifies
- Thread two
  - Executes

- Synchronous
  - Wait
- Asynchronous
  - Executes

Synchronous Cmodx

```
Modifying Processor:
1: Store modified code;
2: Release, guard = 0;

Executing Processor:
1: WHILE (guard == 1); // wait
2: Execute new code;
```

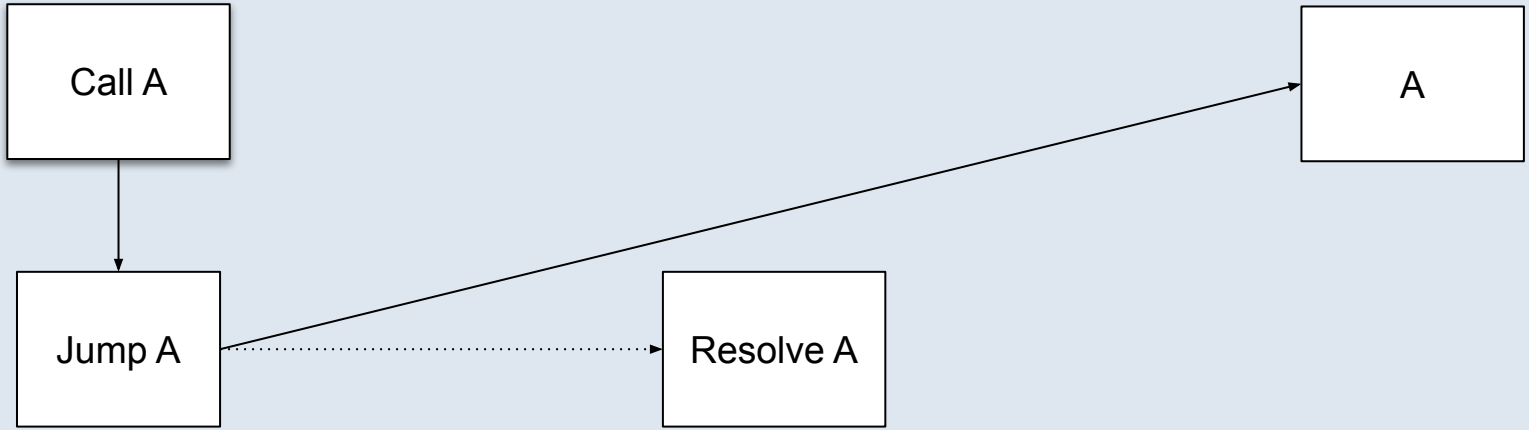
Asynchronous Cmodx

```
Modifying Processor:
1: Store modified code;

Executing Processor:
2: Execute new _OR_ old code;
```

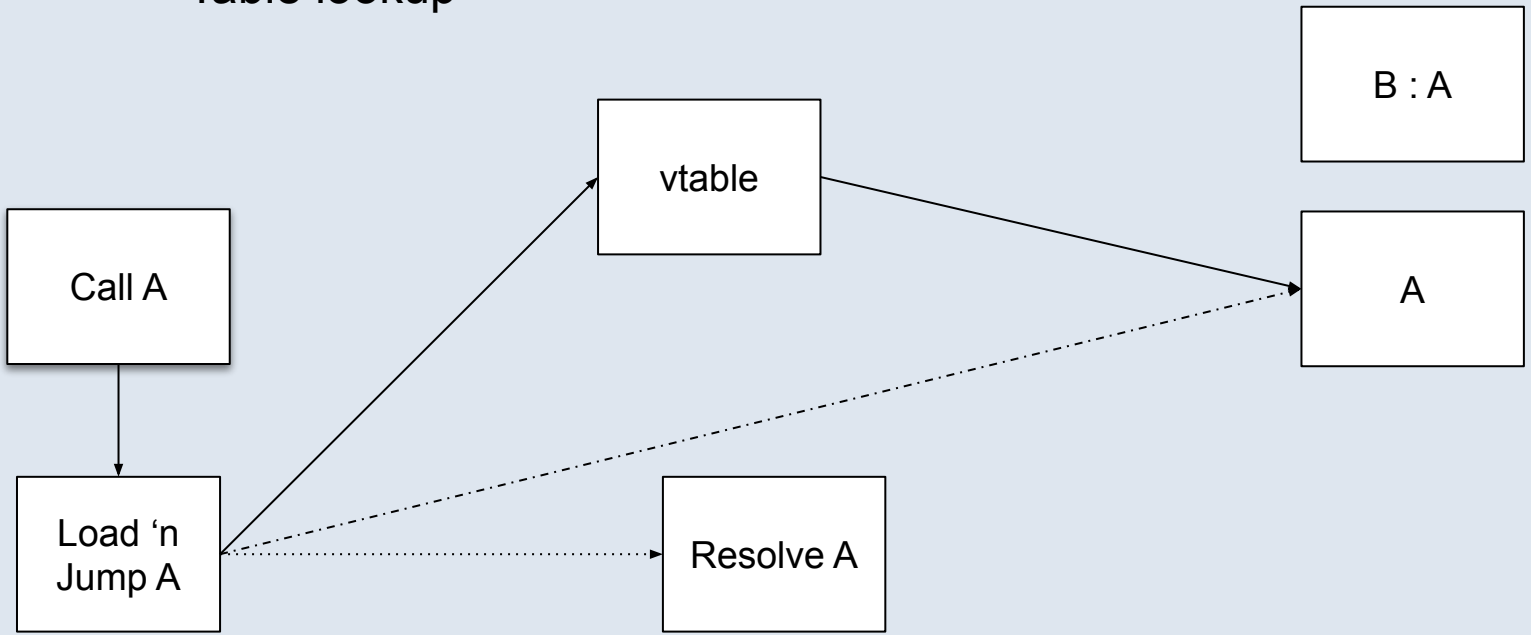
Asynchronous Cmodx

- Resolve or A ?
  - Both valid
  - Resolve => A
    - Point Of Unification



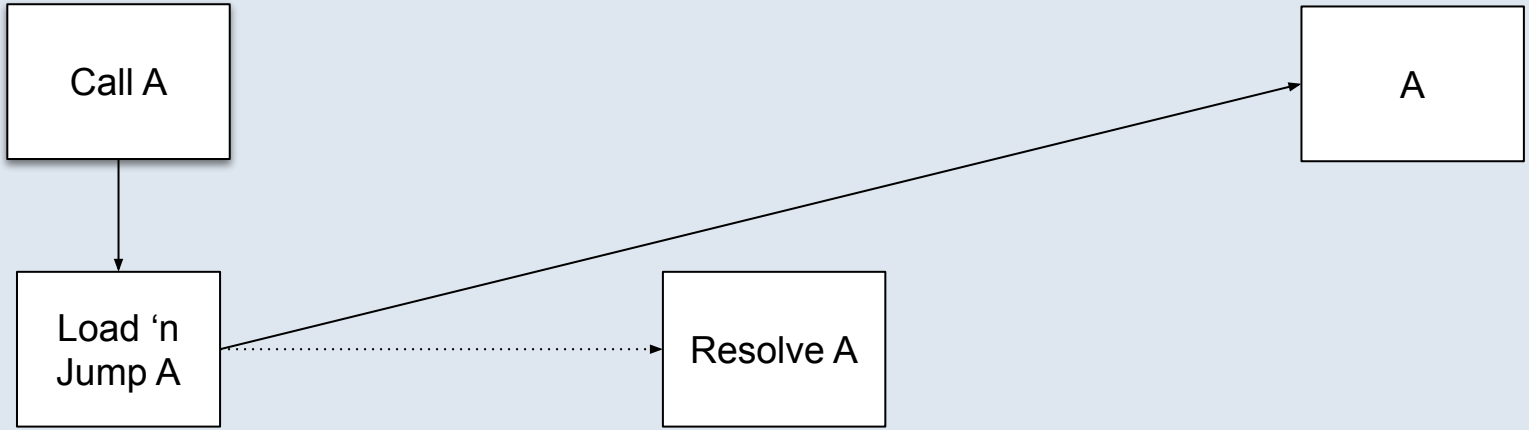
Not a one time thing

- Table lookup



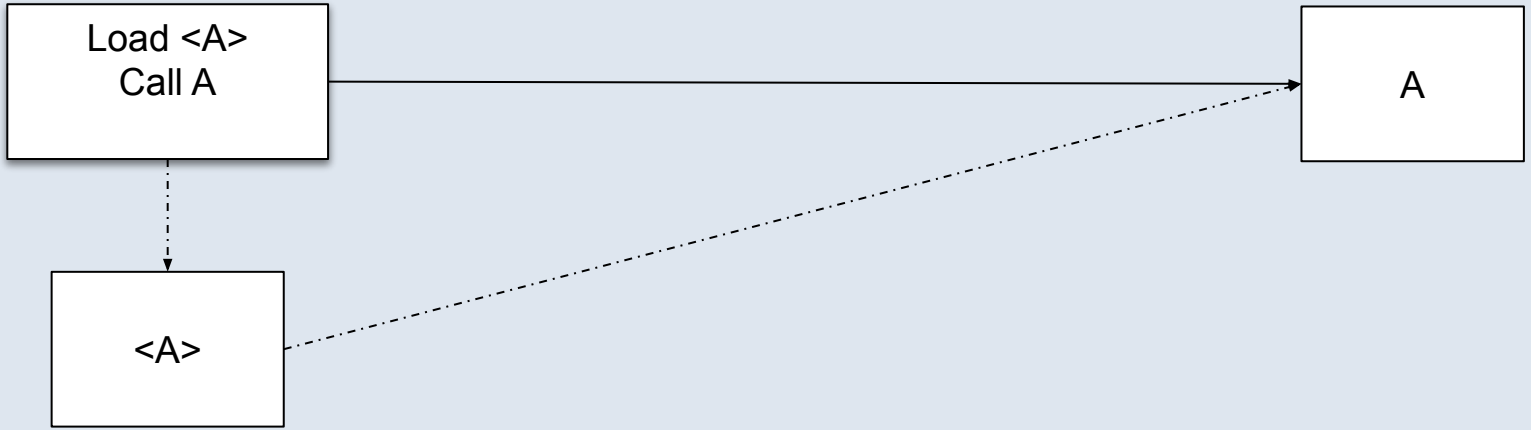
No cmodx

- Ld
  - Data
  - New or Old



No trampoline

- Full range and atomic



Allow races

- Hotpath
  - No synchronization
  - Stale instruction
- Slowpath
  - Point of Unification

- *JIT Code*
- *Deferred initialization*
- *Method entry barriers*
  - *State*
  - *Guard*
- *Method re-entry barriers*
- *Callsites*
- *Imm objects*
- *Imm gc barrier*

RISC-V PoU

- Point of Unification
  - Make stores ‘visible’
  - Invalid stale instructions

- riscv_flush_icache
  - IPI, Inter Processor Interrupt
    - On every write
  - Expensive

RISC-V PoU

- Emit fence.i
- Context switch
  - fence.i
- Ask kernel
  - Prctl
- Stores
  - Tearing

Zjid

- Extension
  - I/D Synchronization
- Import.I
- Instruction fetching
  - No tearing
- Context switches
- 2024?

Extensions

- Many!
  - ~60 ratified (rv64)
    - ~450 base ins
  - ~45 unratified (rv64)
    - ~400 new base ins

- CRC32
  - Base ISA
  - Zbc (carry-less multiplication)
  - Vector

Extensions

- Compile-time
  - -march=rv64i2p1_m2p0_a2p1_f2p2_d2p2_c2p0_v1p0_zi…
  - Profiles
    - RVA23
- JIT
  - Options
    - -XX:+UseZbb
- Hwprobe
  - ifunc/templates

Extensions

- Testing
  - Misaligned access
  - Vector
  - Memory model
  - Combinations !
- Assembly

Compressed

- Option
- Auto-magically exchanged
  - Registers (x8-x15)
  - Heapbase x27
- Fixed size code snippets
- 5-10% code size reduction

Memory model

- Hardware
  - RVWMO - weak
  - RVTSO - strong

- openJDK
  - Hotspot MM (~x86)
  - Java MM
  - C++ MM

Memory model

- 6 mapping
  - Extensions
    - Zacas (CAS instead of LR/SC)
- Cross ABI
- Testing costly

Sign extension

- Enlarge
  - Word -> Double Word
  - Replicates the most significant bit
    - Preserving sign
- Full register instructions
  - branch/and/or

Sign extension

- Assembly
  - Type-less (register) passing
  - Templates + inline assembly
  - Type aliasing
- Natural to avoid
- Un-natural
  - Different

Canonical NaN

- Not-A-Number
  - Sign bit

- Canonical NaN (signed bit not set)
  - fcvt.d.s, fadd, fmul, …
  - No sign propagating

- std::signbit()
- Float.floatToRawIntBits()

Baseline ISA

- RVA23 + zjid + ?

- One more instruction
  - Imm64
  - Ld
    - Cache misses
    - Memory bandwidth

Thank you