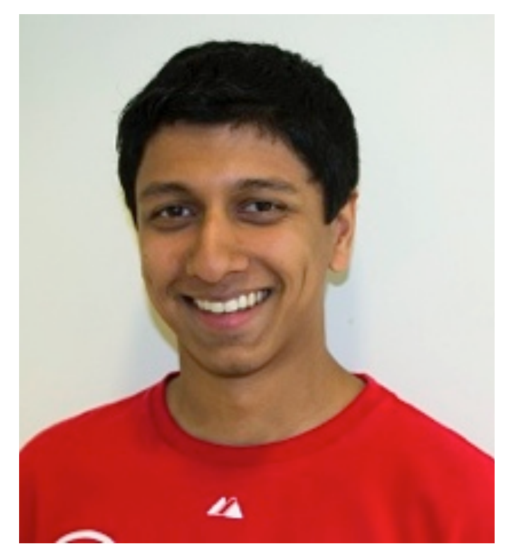
# ANGEL – JS RISC-V ISA Simulator

Sagar Karandikar

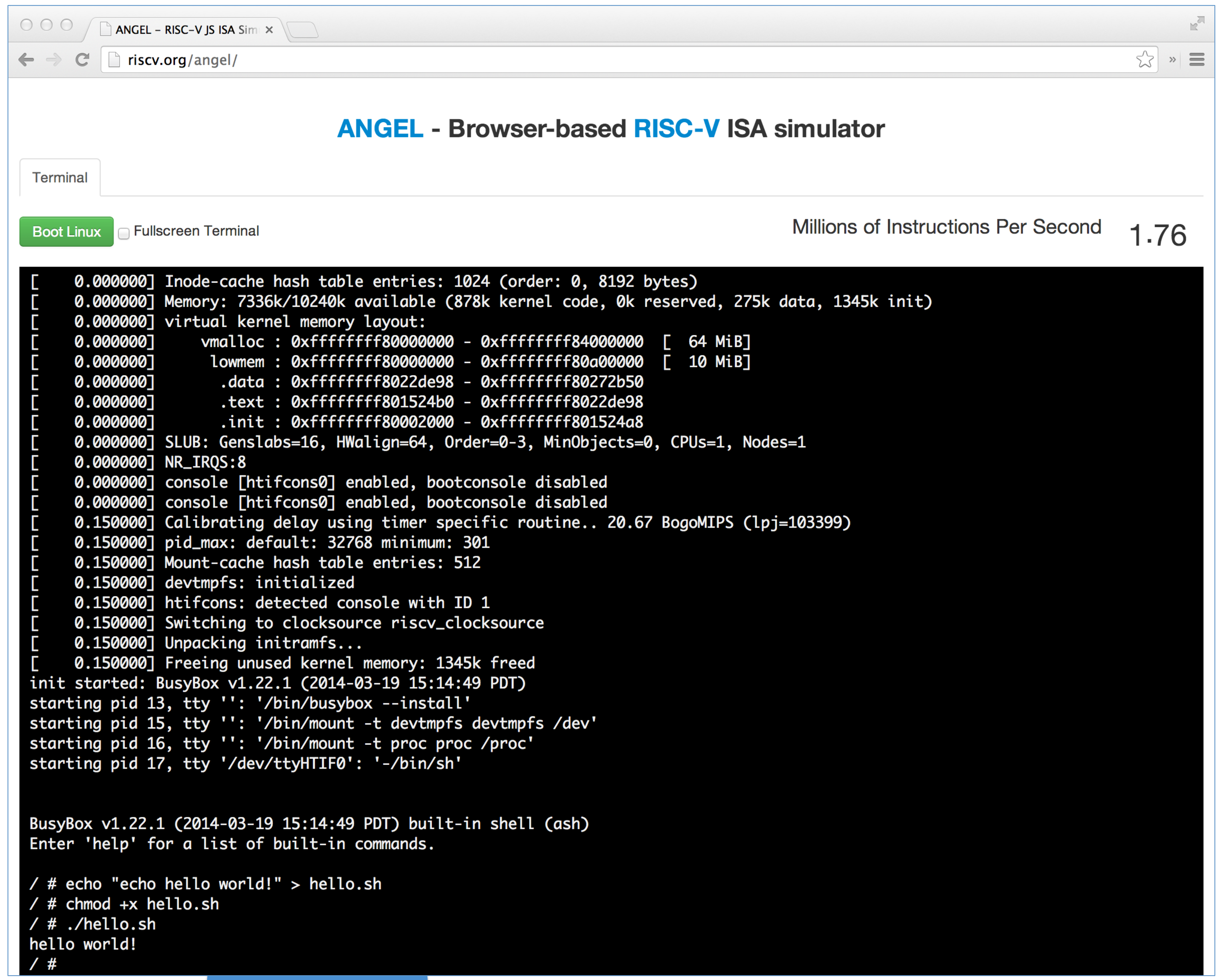


## Boot Linux in a Web Browser on a Simulated RISC-V System

Demo: http://riscv.org/angel

## Goals/System Specifications

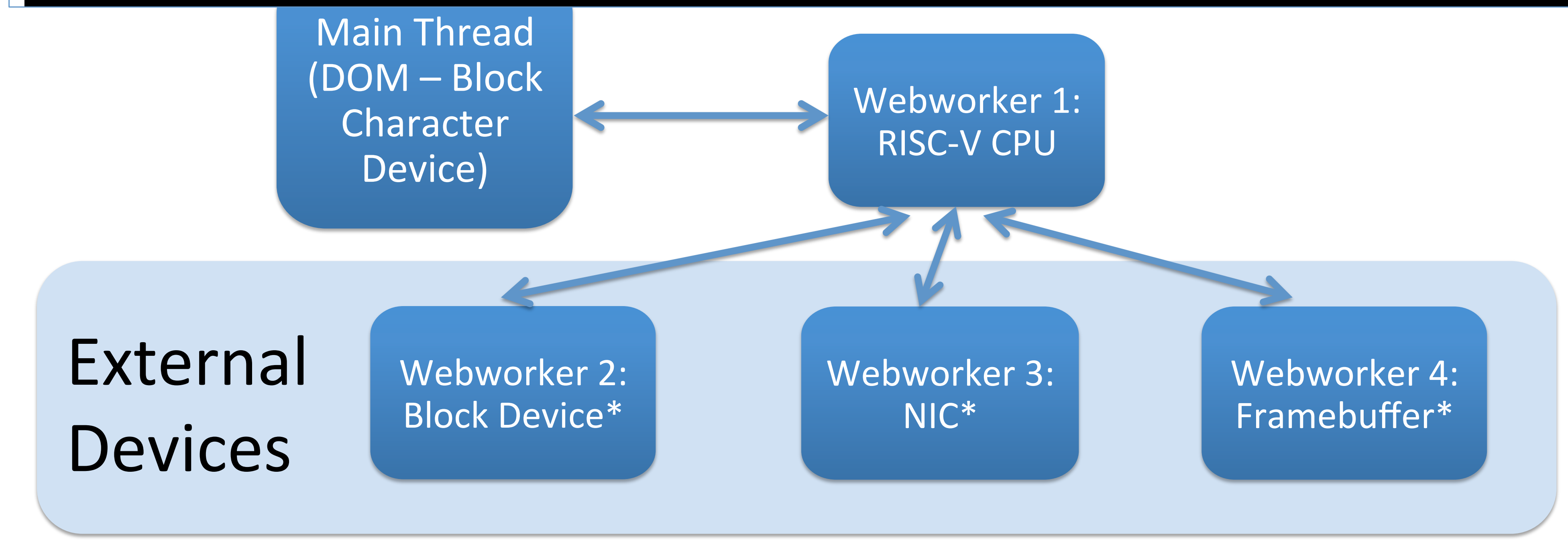
- Showcase RISC-V ISA
  - Interactive Linux session with minimal work on the part of the user (no installation of toolchains)
  - Promotion, Education
- Support RV64v43 IMA
  - (I) Want 64-bit Integer Support, but "Number" type in JS is 64-bit float – use Closure's Long.js library
  - (M) Long.js lacks support for most required multiply/divide operations
  - (A) Single core, in-order, but required for boot
  - 64-bit Address Space with Page-Based VM
- Boot unmodified riscv-linux
  - Needs to be fairly performant (JavaScript + Long.js imposes a drastic performance penalty over the built-in number type)
  - Minimum 10 MiB of simulated memory for boot to ash shell with BusyBox Toolkit in initramfs
  - Interrupts/Timer support – difficult to predict how fast the simulated system is, but knowing is crucial for Linux boot, ESC key functionality
  - User Interaction through emulated terminal (inf-loop interpreting instructions freezes DOM updates)



ANGEL - Browser-based RISC-V ISA simulator

Millions of Instructions Per Second 1.76

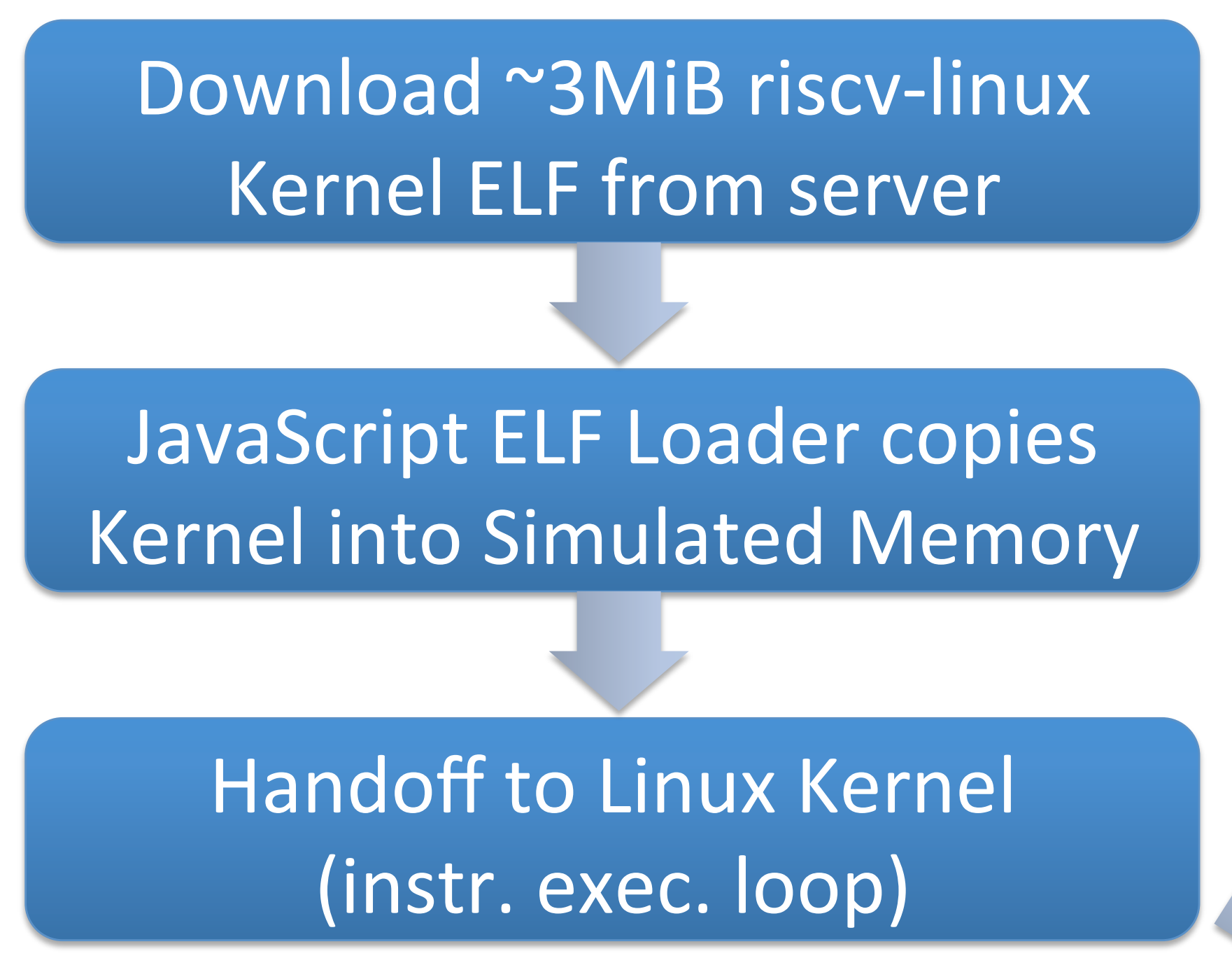
```
[    0.000000] Inode-cache hash table entries: 1024 (order: 0, 8192 bytes)
[    0.000000] Memory: 7336k/10240k available (878k kernel code, 0k reserved, 275k data, 1345k init)
[    0.000000] virtual kernel memory layout:
[    0.000000]     vmalloc : 0xffffffff80000000 - 0xffffffff84000000  [  64 MiB]
[    0.000000]      lowmem : 0xffffffff80000000 - 0xffffffff80a00000  [  10 MiB]
[    0.000000]       .data : 0xffffffff8022de98 - 0xffffffff80272b50
[    0.000000]       .text : 0xffffffff801524b0 - 0xffffffff8022de98
[    0.000000]       .init : 0xffffffff80002000 - 0xffffffff801524a8
[    0.000000] SLUB: Genslabs=16, HWalign=64, Order=0-3, MinObjects=0, CPUs=1, Nodes=1
[    0.000000] NR_IRQS:8
[    0.000000] console [htifcons0] enabled, bootconsole disabled
[    0.000000] console [htifcons0] enabled, bootconsole disabled
[    0.150000] Calibrating delay using timer specific routine.. 20.67 BogoMIPS (lpj=103399)
[    0.150000] pid_max: default: 32768 minimum: 301
[    0.150000] Mount-cache hash table entries: 512
[    0.150000] devtmpfs: initialized
[    0.150000] htifcons: detected console with ID 1
[    0.150000] Switching to clocksource riscv_clocksource
[    0.150000] Unpacking initramfs...
[    0.150000] Freeing unused kernel memory: 1345k freed
init started: BusyBox v1.22.1 (2014-03-19 15:14:49 PDT)
starting pid 13, tty '': '/bin/busybox --install'
starting pid 15, tty '': '/bin/mount -t devtmpfs devtmpfs /dev'
starting pid 16, tty '': '/bin/mount -t proc proc /proc'
starting pid 17, tty '/dev/ttyHTIF0': '-/bin/sh'


BusyBox v1.22.1 (2014-03-19 15:14:49 PDT) built-in shell (ash)
Enter 'help' for a list of built-in commands.

/ # echo "echo hello world!" > hello.sh
/ # chmod +x hello.sh
/ # ./hello.sh
hello world!
/ #
```


Main Thread (DOM – Block Character Device)
Webworker 1: RISC-V CPU
External Devices
Webworker 2: Block Device*
Webworker 3: NIC*
Webworker 4: Framebuffer*

## Performance Optimizations

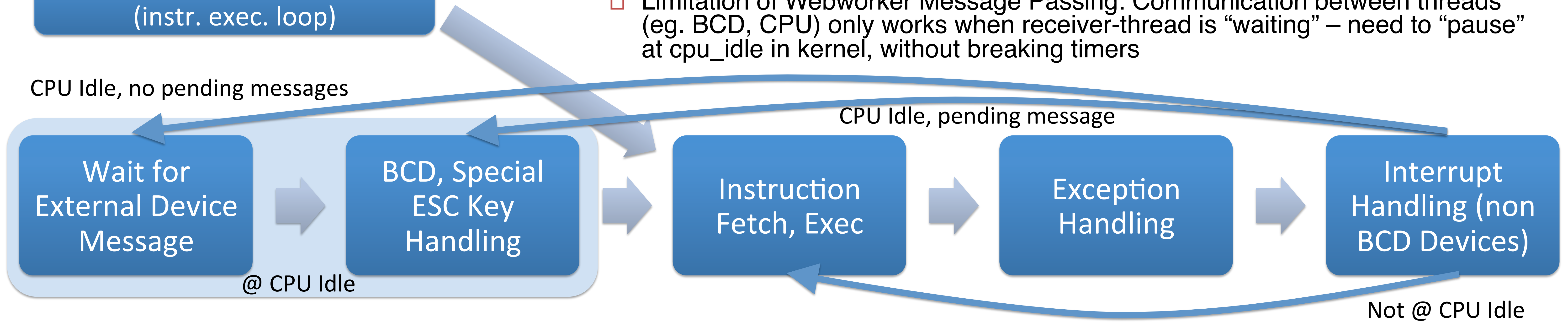
- Initially, 10 seconds to boot minimal proxy-kernel and run basic C program
- Webworkers (new in HTML5)
  - "Standard" JS implementation is one-thread, cannot support simultaneous UI updates with a loop continuously executing instructions
  - Fix: use asynchronous setTimeout – but a minimum timeout of 4ms makes this infeasible (250 Insts/sec)
  - Instead, UI (block character device) gets main thread, CPU + other devices each get a webworker thread
  - Inter-thread comm. with message passing
  - *Reduced boot time from days to hours*
- 32-bit Program Counter
  - 64-bit PC operations drastically increase seconds/cycle
  - Table of most significant 32 bits, indexed on most significant 12 bits of PC
  - *Reduced boot time from hours to ~10 mins*
- TLB
  - Translation process requires ~25 operations on 64-bit quantities per instruction fetch
  - "Infinite" sized TLB to avoid translation whenever possible
  - *Reduced boot time from ~10 minutes to ~20 seconds*
- Measured Specs on Chrome 35.0 on OSX
  - Execs 1.75 Million Instructions Per Second
  - 15 seconds to boot to ash shell prompt

## Boot Process


Download ~3MiB riscv-linux Kernel ELF from server
JavaScript ELF Loader copies Kernel into Simulated Memory
Handoff to Linux Kernel (instr. exec. loop)

## Execution Loop

- Limitation of Webworker Message Passing: Communication between threads (eg. BCD, CPU) only works when receiver-thread is "waiting" – need to "pause" at cpu_idle in kernel, without breaking timers


CPU Idle, no pending messages
CPU Idle, pending message
Wait for External Device Message
BCD, Special ESC Key Handling
@ CPU Idle
Instruction Fetch, Exec
Exception Handling
Interrupt Handling (non BCD Devices)
Not @ CPU Idle

## Future Work

- Implement Additional Devices*
  - Block Device: Programs like gcc won't fit on initramfs
  - Framebuffer: boot a GUI
- Higher Performance
  - Ideally, JS 64-Bit Integer support
  - Support for 64-bit Integers in ASM.js
- Education
  - Web-based IDE for RISC-V Assembly, like MARS for MIPS