A **fast** RISC-V emulator in JavaScript

How hard can it be?

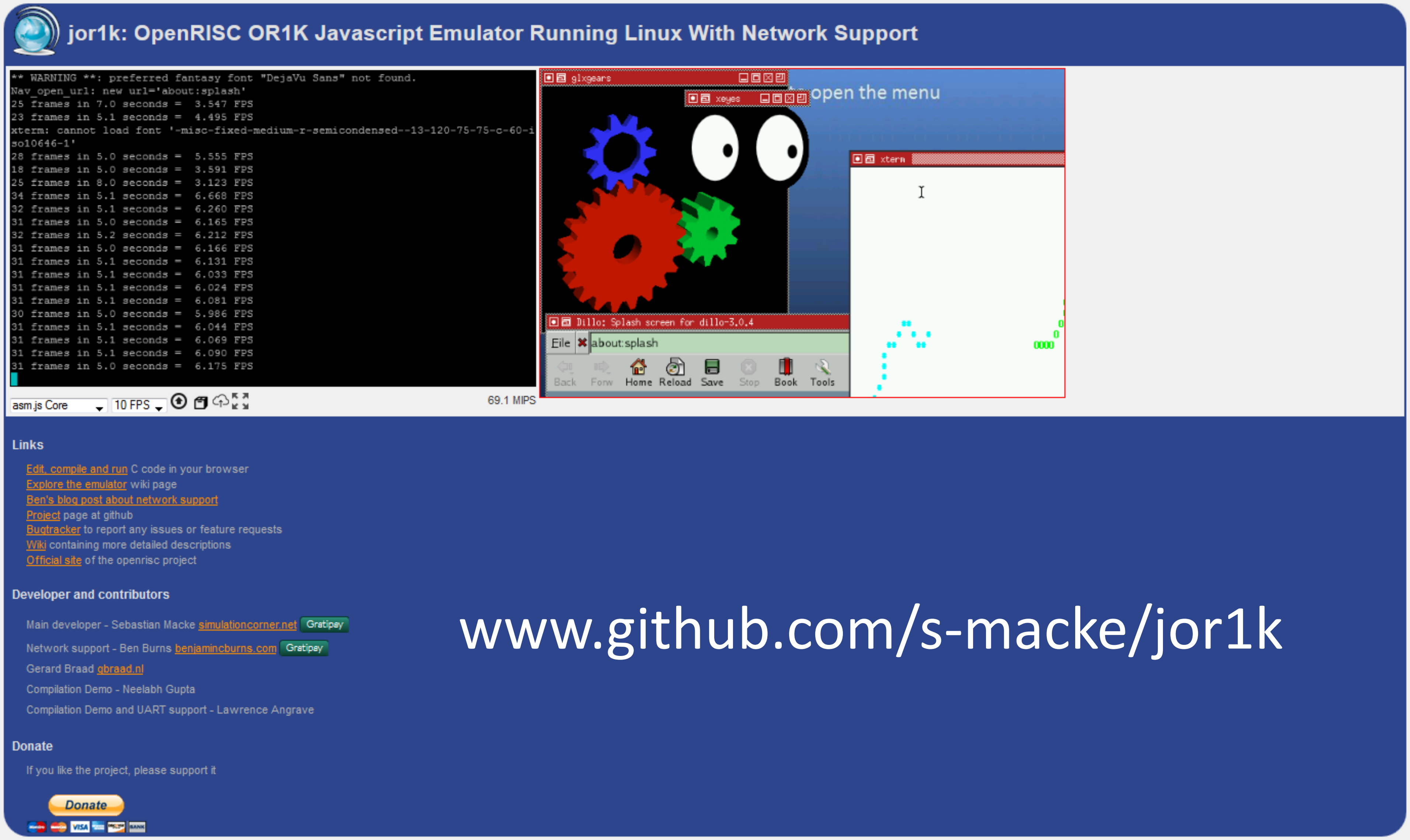
Sebastian Macke

Outline

- What is jor1k? - Overview
- The CPU implementation exemplified on the RISC-V architecture (GSoC 2015)
- How to not idle in JavaScript
- Filesystem and Network
- Outlook

What is jor1k?

The website: **jor1k.com**



jor1k: OpenRISC OR1K Javascript Emulator Running Linux With Network Support

```
** WARNING **: preferred fantasy font "DejaVu Sans" not found.
Nav_open_url: new url='about:splash'
25 frames in 7.0 seconds =  3.547 FPS
23 frames in 5.1 seconds =  4.495 FPS
xterm: cannot load font '-misc-fixed-medium-r-semicondensed--13-120-75-75-c-60-i
so10646-1'
28 frames in 5.0 seconds =  5.555 FPS
18 frames in 5.0 seconds =  3.591 FPS
25 frames in 8.0 seconds =  3.123 FPS
34 frames in 5.1 seconds =  6.668 FPS
32 frames in 5.1 seconds =  6.260 FPS
31 frames in 5.0 seconds =  6.165 FPS
32 frames in 5.2 seconds =  6.212 FPS
31 frames in 5.0 seconds =  6.166 FPS
31 frames in 5.1 seconds =  6.131 FPS
31 frames in 5.1 seconds =  6.033 FPS
31 frames in 5.1 seconds =  6.024 FPS
31 frames in 5.1 seconds =  6.081 FPS
30 frames in 5.0 seconds =  5.986 FPS
31 frames in 5.1 seconds =  6.044 FPS
31 frames in 5.1 seconds =  6.069 FPS
31 frames in 5.1 seconds =  6.090 FPS
31 frames in 5.0 seconds =  6.175 FPS
```

asm.js Core | 10 FPS | 69.1 MIPS

Links

- Edit, compile and run C code in your browser
- Explore the emulator wiki page
- Ben's blog post about network support
- Project page at github
- Bugtracker to report any issues or feature requests
- Wiki containing more detailed descriptions
- Official site of the openrisc project

Developer and contributors

Main developer - Sebastian Macke simulationcorner.net

Network support - Ben Burns benjamincburns.com

Gerard Braad gbraad.nl

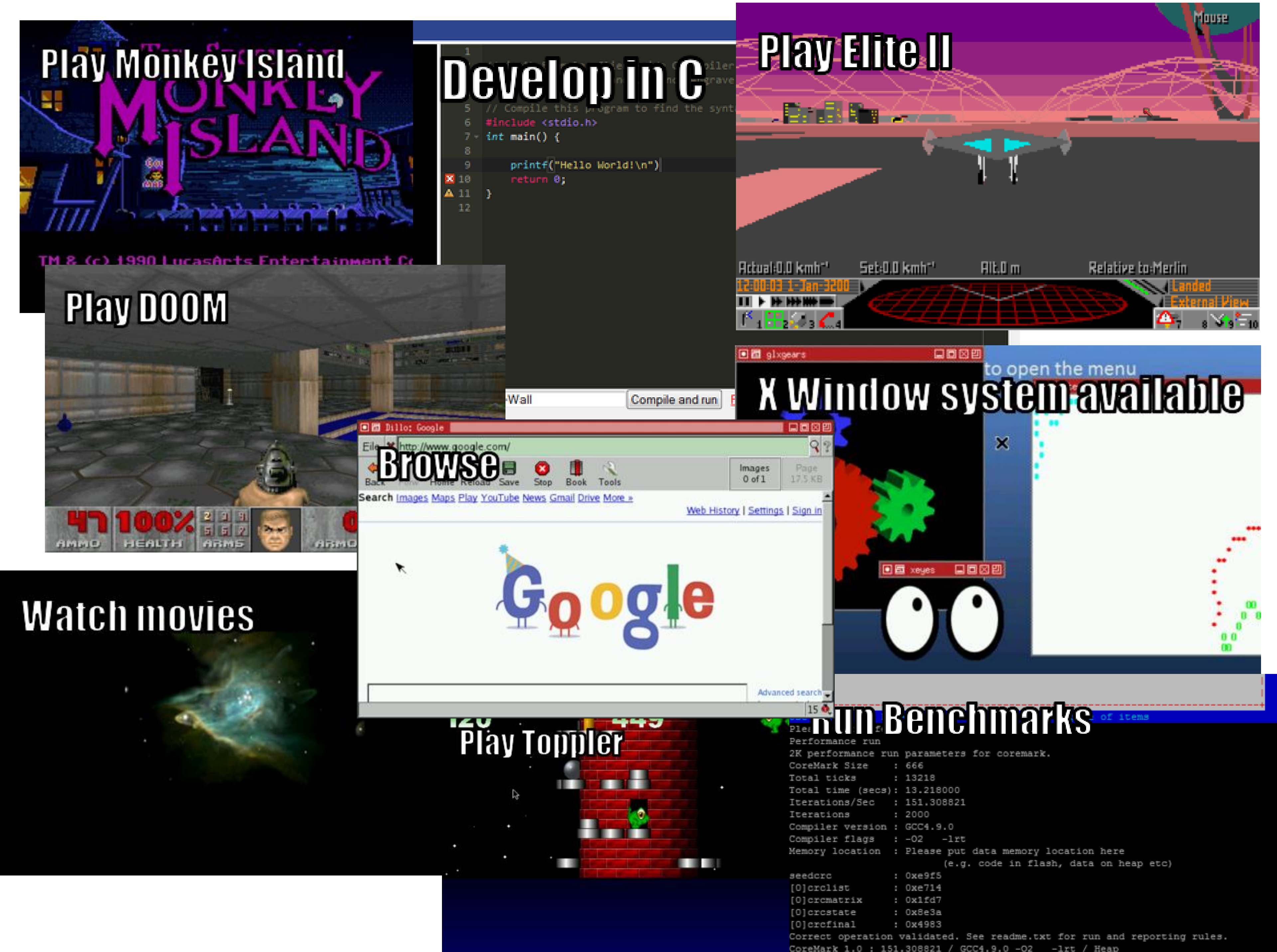
Compilation Demo - Neelabh Gupta

Compilation Demo and UART support - Lawrence Angrave

Donate

If you like the project, please support it

www.github.com/s-macke/jor1k

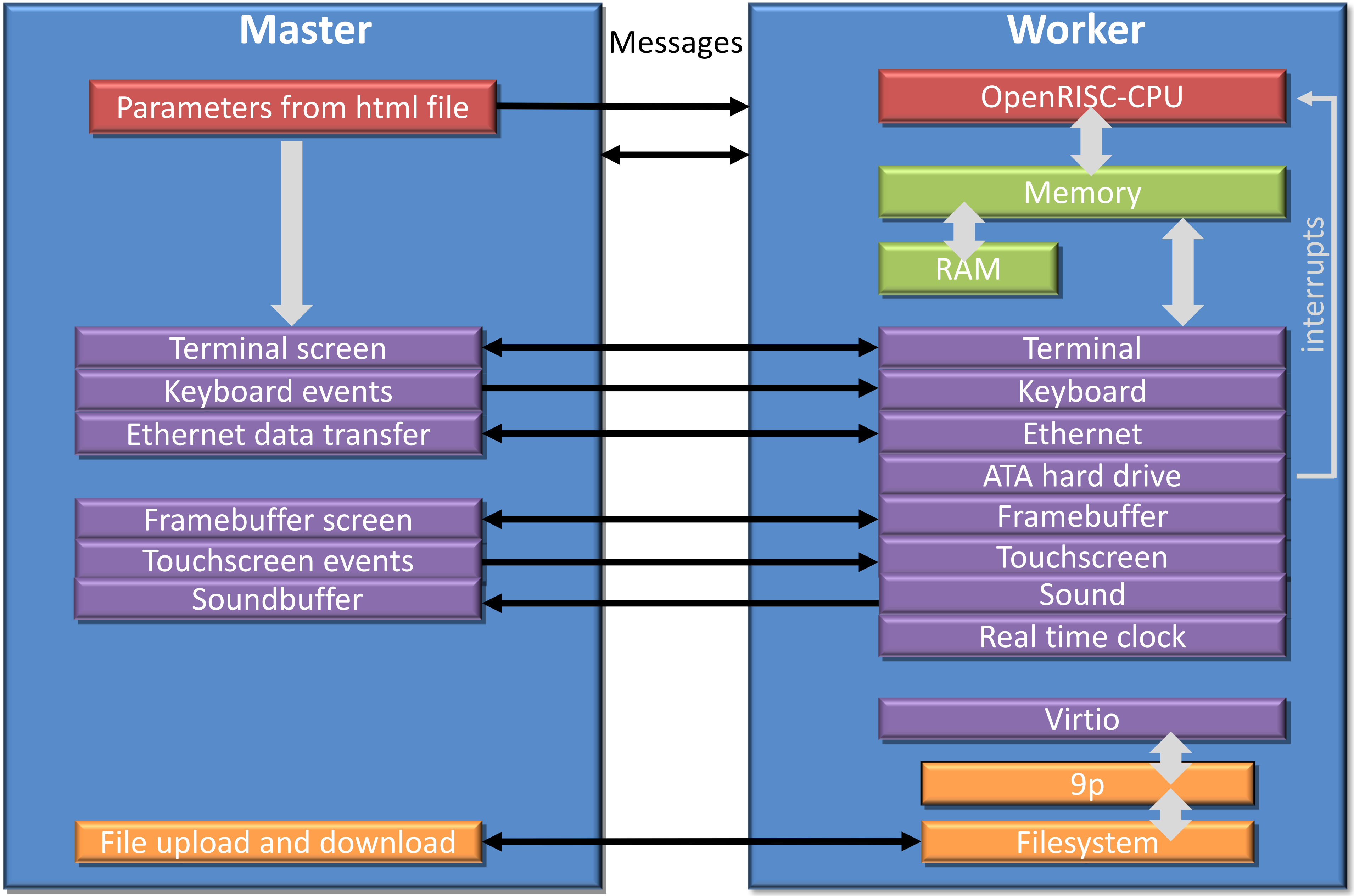


```
// Compile this program to find the synt
#include <stdio.h>
int main() {

    printf("Hello World!\n")
    return 0;
}
```

```
Performance run
2K performance run parameters for coremark.
CoreMark Size     : 666
Total ticks       : 13218
Total time (secs): 13.218000
Iterations/Sec    : 151.308821
Iterations        : 2000
Compiler version : GCC4.9.0
Compiler flags    : -O2   -lrt
Memory location   : Please put data memory location here
                        (e.g. code in flash, data on heap etc)
seedcrc           : 0xe9f5
[0]crclist        : 0xe714
[0]crcmatrix      : 0x1fd7
[0]crcstate       : 0x8e3a
[0]crcfinal       : 0x4983
Correct operation validated. See readme.txt for run and reporting rules.
CoreMark 1.0 : 151.308821 / GCC4.9.0 -O2   -lrt / Heap
```

Jor1k Modules



RISC-V

GSOC 2015 project

Student: Prannoy Pilligundla

RISC-V observations from someone who just wanted to write an emulator

- Well designed ISA
  - 32 registers , 4 byte instructions, Easy encoding
  - Modules (32-Bit, 64-Bit, 128-Bit, Integer, Floating Point, Atomic, ……)
  - No history
  - Little endian
  - Only 2 instructions for random 32-Bit jumps
  - No delayed instruction
  - No conditional instructions
  - No Flags
  - Complete implementation of mul and div
  - PC relative addressing modes („auipc“)
  - A lot of atomic instructions
  - Is even more like bytecode in comparison to OpenRISC (less side effects)

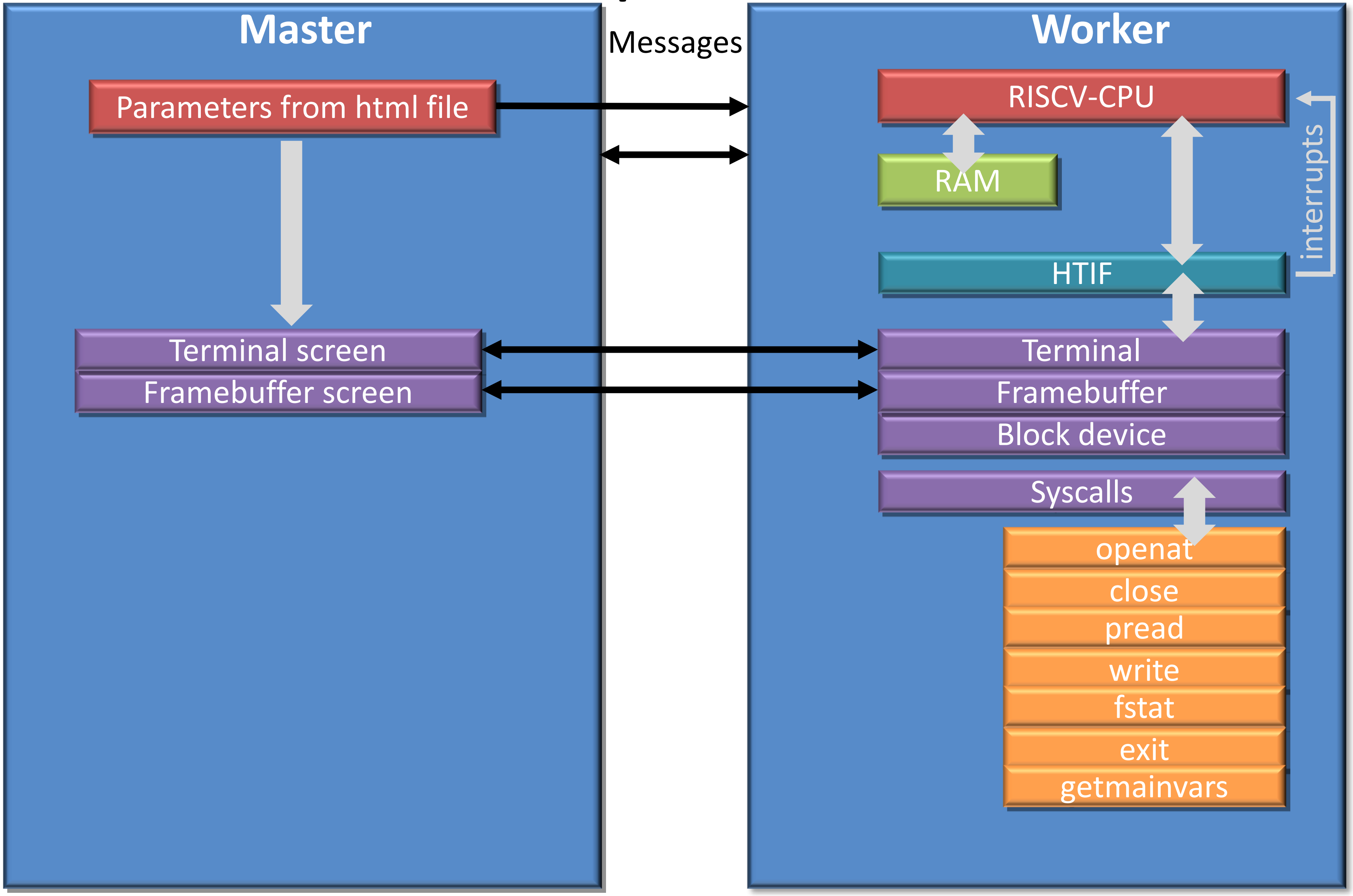
- Linux does run with the help of a proxy kernel and does not have direct access to hardware

RISC-V observations from someone who just wanted to write an emulator

- Complete/Good hardware devices support is still missing.
  - No MMIO, instead two register based HTIF interface
  - Hardware devices in 32-Bit are currently not supported

- Still lots of errors in the toolchain
  - 32-Bit is not well tested
- Documentation is still incomplete - inconsistencies with emulator
- Confusing mode switching (traps, ecall, eret, ebreak, mrts)

JavaScript Modules



Why JavaScript?

- **JavaScript is the language of the Web**

- It runs everywhere

- It is available immediately

JavaScript

- Doesn't support 64-Bit integers
- JavaScript is very creative with typing
  - `[1,2,3]+[4,5,6] =>` **`1,2,34,5,6`**
  - `{} + {} =>` **`NaN`**
  - `0 == "" =>` **`true`**
  - `x=Math.sqrt(-2); if (x==x) =>` **`false`**
  - `if (new Array() == false) =>` **`true`**
- JavaScript is considered "slow"
- At least four companies are writing optimized compilers to squeeze out the maximum performance.

All numbers are double

- There are no integers, only doubles
  - `y = 9999999999999999 => y = 10000000000000000`

    (double) (double)

  - `y = 0xFFFFFFFF+1 => y = 0x100000000`

    (internal integer) (deoptimized into double)

- But there are logical operations
  - `y = 0xFFFFFFFF|0  => y = -1`
  - `y = 0xFFFFFFFF>>> 0  => y = 0xFFFFFFFF`
- But there are typed arrays
  - `x = new Uint32Array(length)`

How to prevent deoptimizations?

- What’s wrong with the following code concerning the compiled code?

  -
    ```
    for(;;) {
      Advance_Timer();
      Check_Interr
      ppc = Transl
      ins = ram.Re
      pc += 4; //
      // decode in
      switch (ins&
             ....
      }
    }
    ```

What happens internal?

1. Add 4 to pc (integer)
2. Check for overflow
3. Deoptimize into double if overflow
4. Cascade of deoptimizations where pc is used.

How to prevent deoptimizations?

- Prevent overflow by adding a “|0”

```
– for(;;) {
   Advance_Timer();
    Check_Interrupt();
    ppc = Translate_Virtual_To_Physical(pc);
    ins = ram.Read32(
    pc = (pc + 4)|0;
    // decode instruc
    switch (ins&0x7F)
        ....
    }
 }
```

What happens internal?

1. Add 4 to pc (integer)
2. Ignore noop „|0“

How to prevent deoptimizations?

- Add more typing helpers

  -
    ```
    for(;;) {
      Advance_Timer();
       Check_Interrupt();
       ppc = Translate_Virtual_To_Physical(pc|0)|0;
       ins = ram.Read32(ppc|0)|0;
       pc = (pc + 4)|0;
       // decode instruction
       switch (ins&0x7F) {
            ....
       }
    }
    ```

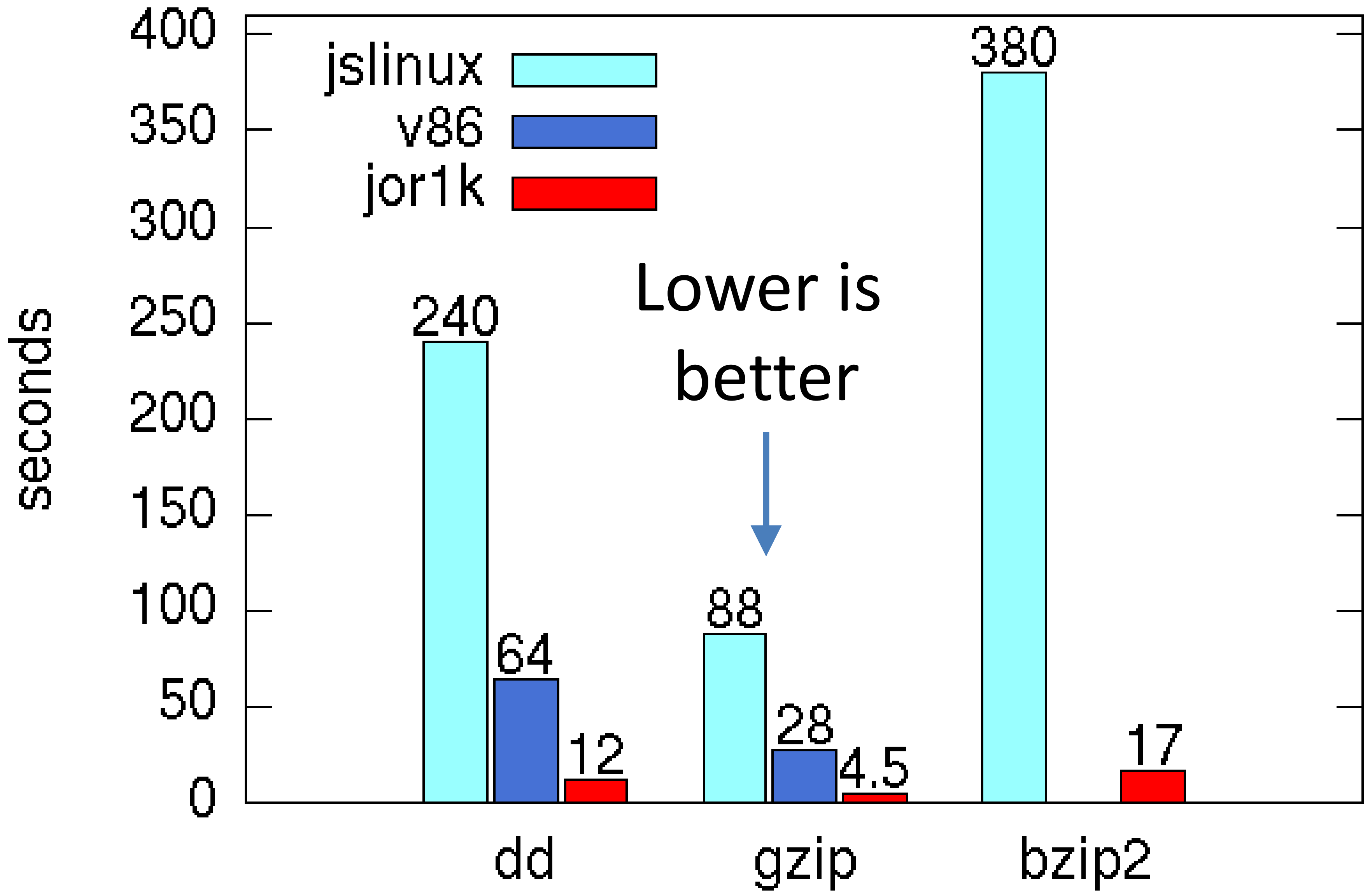
What is asm.js

- The mode **`"use strict"`**`;` adds additional error messages for accessing undefined variables.
- The mode **`"use asm"`**`;` adds additional error messages to give you a guarantee for fixed type variables that must be compiled only once.
  - Only a subset of Javascript is allowed
  - Fully compatible
- Implemented in Firefox in 2013
- Implemented in Edge in 2015

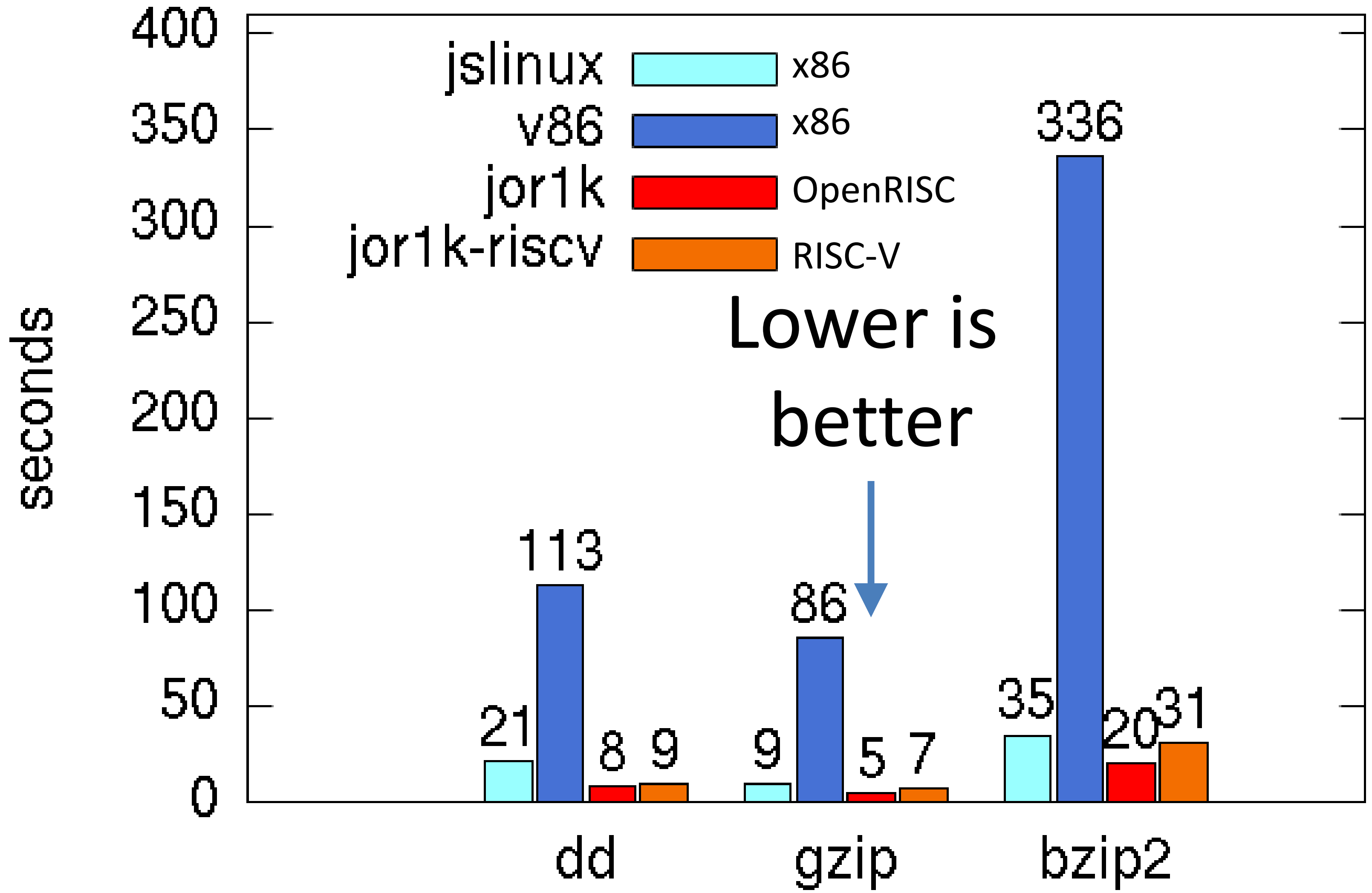
What is asm.js

- But the syntax is nasty
  - `group0[SPR_IMMUCFGR] = 0x18;`
  - `h[group0p + (SPR_IMMUCFGR<<2) >> 2] = 0x18;`
- `h` is the heap and `group0p` is the pointer to the table
- In this case the "view" of the heap is 32 Bit. Therefore the last operation for the index must be ">> 2"
- Project Emscripten allows to translate C++ to asm.js JavaScript

Benchmark from last year



Benchmark one week ago

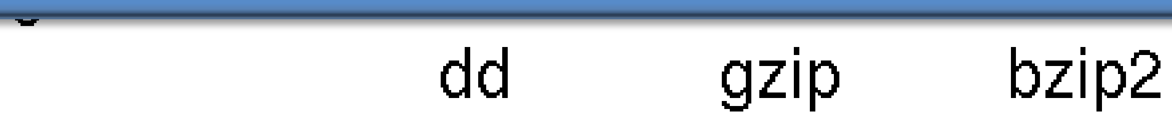


Benchmark one week ago

Firefox on Core2Duo:	120 MIPS
Firefox on Core i7 2600:	75 MIPS
Firefox on Celeron G1820:	180 MIPS
Firefox on Core i7 4770:	246 MIPS
Chrome: on Core i7 2600:	60 MIPS
IE 11 on Core i7 2600:	68 MIPS
Safari on Apple A7:	81 MIPS

Approx. speed of the CPUs of the year 1997

OpenRISC CPU implementation:	1609 lines of code
RISC-V CPU implementation :	2181 lines of code



Architecture

- OpenRISC is easy

```
switch ((ins >> 26)&0x3F) {
…
    case 0x29: // l.andi
        r[(ins>>21)&0x1F] = r[(ins>>16)&0x1F] & (ins&0xFFFF);
        continue;
…
}
```

- RISC-V is easy too, but not that easy

```
switch (ins&0x7F) {
    …
    case 0x13:
      switch((ins >> 12)&0x7)  {
        …
        case 0x7: // andi
            r[(ins>>7)&0x1F] = r[(ins>>15)&0x1F] & (ins>>20);
            continue;
```

To decode an instruction in RISC-V takes 2 switches in average and not 1 like for OpenRISC

Instruction emulation for ARM

```
void
armv5_and(){
                uint32_t icode = ICODE;
                int rn,rd;
                uint32_t cpsr=REG_CPSR;
                uint32_t Rn,op2,result;
                uint32_t S;
                if(!check_condition(icode)) {
                                return;
                }
                rd=(icode>>12)&0xf;
                rn=(icode>>16)&0xf;
                Rn=ARM9_ReadReg(rn);
                cpsr&= ~(FLAG_N | FLAG_Z | FLAG_C);
                cpsr |= get_data_processing_operand(icode);
                op2 = AM_SCRATCH1;
                result=Rn&op2;
                ARM9_WriteReg(result,rd);
                S=testbit(20,icode);
                if(S) {
                                if(!result) {
                                                cpsr|=FLAG_Z;
                                }
                                if(ISNEG(result)) {
                                                cpsr|= FLAG_N;
                                }
                                if(rd==15) {
                                                if(MODE_HAS_SPSR) {
                                                                SET_REG_CPSR(REG_SPSR);
                                                } else {
                                                                fprintf(stderr,"Mode has no spsr in line %d\n",__LINE__);
                                                }
                                } else {
                                                REG_CPSR=cpsr;
•                               }
•               }
•               dbgprintf("AND result op1 %08x,op2 %08x, result %08x\n",Rn,op2,result);
•     }
```

Fence technique

- For every instruction the program counter (pc) must be translated to the physical pc. However, the pc advances usually by 4 not leaving the current page.

- The fastpath for one instruction looks like this:

  –
  ```
  for(;;) {
    if ((physical_pc|0) != (fence|0)) {
      ins = int32ram[physical_pc >> 2]|0;
      physical_pc = physical_pc + 4|0;

      switch (ins&0x7F) {
      ....
      }
    } else {
      Advance_Timer();
      Check_Interrupt();
      ......
    }
  }
  ```

  – The idea here is that the virtual pc is computed only when needed by translating `ppc` (physical pc) back to the virtual pc address. The variable `fence` is used to break out of the fast path when `ppc` reaches a jump or the end of the current page.

Assembly level optimizations

Javascript code	Block description	Generated x86 asm code
for(;;) {	.set .Llabel132981, .	

if ((fence|0) != (ppc|0)) {		`Movl (nil), %ecx`
		`Movl (nil), %eax`
		`Cmpl %eax, %ecx`
		`je .Lfrom133000`

ins = ram[ppc >> 2]|0;	MoveGroup
	BitOpI:bitand
		`movl %eax, %ecx`
		`andl $0xfffffffc, %ecx`

	AsmJSLoadHeap	`cmpl $0xfffffffc, %ecx`
		`ja .Lfrom133022`
		`movl 0x0000(%ecx), %ecx`

	MoveGroup	`movl %ecx, 0x2c(%esp)`

ppc = ppc + 4|0;	instruction AddI
	AsmJSStoreGlobalVar
		`addl $4, %eax`
		`movl %eax, (nil)`

switch(ins&0x7F) {	MoveGroup
	instruction BitOpI:bitand
		`movl 0x2c(%esp), %edx`
		`andl $0x7f, %edx`

	TableSwitch	`subl $3, %edx`
		`cmpl $0x71, %edx`
		`jae .Lfrom133081`
		`movl $0xffffffff, %ecx`
		`jmp *0x0(%ecx,%eax,4)`

Javascript code	Block description	Generated x86 asm code
for(;;) {	.set .Llabel132981, .	

if ((fence|0) != (ppc|0)) {		`Movl (nil), %ecx`
		`Movl (nil), %eax`
		`Cmpl %eax, %ecx`
		`je .Lfrom133000`

ins = ram[ppc >> 2]|0;	MoveGroup
	BitOpI:bitand
		`movl %eax, %ecx`
		`andl $0xfffffffc, %ecx`

	AsmJSLoadHeap	`cmpl $0xfffffffc, %ecx`
		`ja .Lfrom133022`
		`movl 0x0000(%ecx), %ecx`

	MoveGroup	`movl %ecx, 0x2c(%esp)`

ppc = ppc + 4|0;	instruction AddI
	AsmJSStoreGlobalVar
		`addl $4, %eax`
		`movl %eax, (nil)`

switch(ins&0x7F) {	MoveGroup
	instruction BitOpI:bitand
		`movl 0x2c(%esp), %edx`
		`andl $0x71, %edx`

	TableSwitch	`subl $3, %edx`
		`cmpl $0x71, %edx`
		`jae .Lfrom133081`
		`movl $0xffffffff, %ecx`
		`jmp *0x0(%ecx,%eax,4)`

Unnecessary load

They work on it

Javascript code	Block description	Generated x86 asm code
for(;;) {	.set .Llabel132981, .	

if ((fence|0) != (ppc|0)) {		`Movl (nil), %ecx`
		`Movl (nil), %eax`
		`Cmpl %eax, %ecx`
		`je .Lfrom133000`

ins = ram[ppc >> 2]|0;	MoveGroup
	BitOpI:bitand
		`movl %eax, %ecx`
		`andl $0xfffffffc, %ecx`

	AsmJSLoadHeap	`cmpl $0xfffffffc, %ecx`
		`ja .Lfrom133022`
		`movl 0x0000(%ecx), %ecx`

	MoveGroup	`movl %ecx, 0x2c(%esp)`

ppc = ppc + 4|0;	instruction AddI
	AsmJSStoreGlobalVar
		`addl $4, %eax`
		`movl %eax, (nil)`

switch(ins&0x7F) {	MoveGroup
	instruction BitOpI:bitand
		`movl 0x2c(%esp), %edx`
		`andl $0x7f, %edx`

	TableSwitch	`subl $3, %edx`
		`cmpl $0x71, %edx`
		`jae .Lfrom133081`
		`movl $0xffffffff, %ecx`
		`jmp *0x0(%ecx,%eax,4)`

1 sub to save 12 bytes in a table

Add dummy case 0:

Javascript code	Block description	Generated x86 asm code
for(;;) {	.set .Llabel132981, .	

if ((fence|0) != (ppc|0)) {		`Movl (nil), %ecx`
		`Movl (nil), %eax`
		`Cmpl %eax, %ecx`
		`je .Lfrom133000`

ins = ram[ppc >> 2]|0;	MoveGroup
	BitOpI:bitand
		`movl %eax, %ecx`
		`andl $0xfffffffc, %ecx`

	AsmJSLoadHeap	`cmpl $0xfffffffc, %ecx`
		`ja .Lfrom133022`
		`movl 0x0000(%ecx), %ecx`

	MoveGroup	`movl %ecx, 0x2c(%esp)`

ppc = ppc + 4|0;	instruction AddI
	AsmJSStoreGlobalVar
		`addl $4, %eax`
		`movl %eax, (nil)`

switch(ins&0x7F) {	MoveGroup
	instruction BitOpI:bitand
		`movl 0x2c(%esp), %edx`
		`andl $0x7f, %edx`

	TableSwitch	`subl $3, %edx`
		`cmpl $0x71, %edx`
		`jae .Lfrom133081`
		`movl $0xffffffff, %ecx`
		`jmp 0x0(%ecx,%eax,4)`

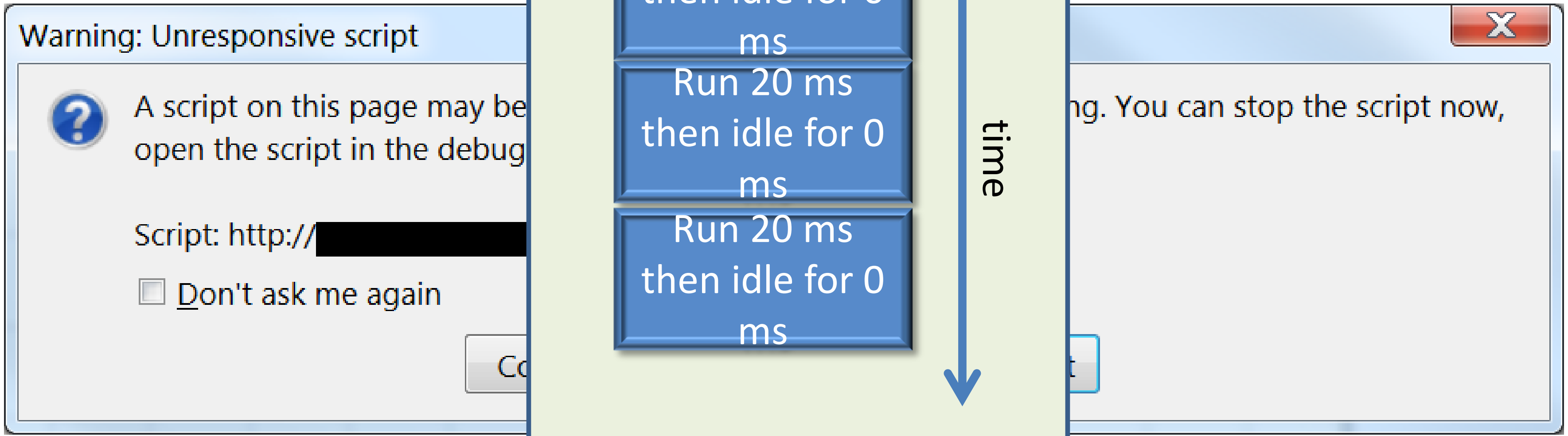
Unnecessary check

In bugzilla

How to not idle in JavaScript?

How to not idle in Javascript but stay responsive?

- Javascript fini efore continuing.

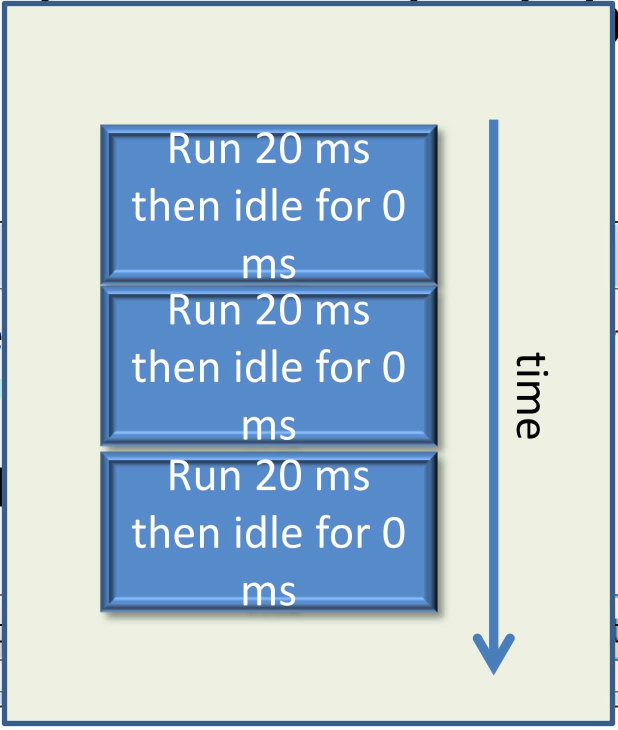


Warning: Unresponsive script

A script on this page may be ng. You can stop the script now, open the script in the debug

Script: http://[redacted]

Don't ask me again

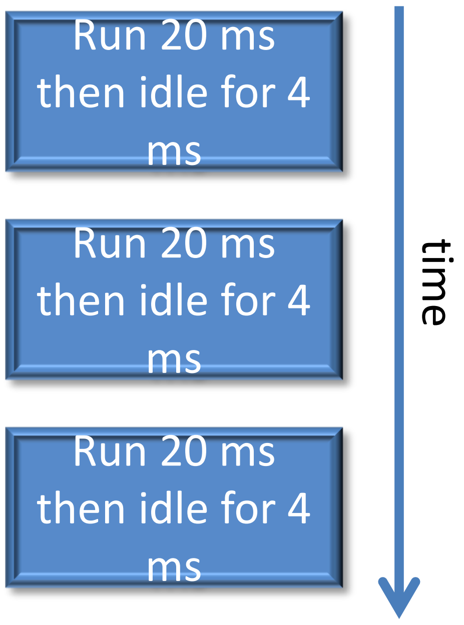


Run 20 ms then idle for 0 ms
Run 20 ms then idle for 0 ms
Run 20 ms then idle for 0 ms
time

- Solution: *setTimout(function(){…}, 0);*

What about the worker thread?

- *setTimout(function(){…}, 0);* doesn’t work in worker thread. Message queue is never processed.
- But *setTimout(function(){…}, 4);* works (4ms waiting time)


Run 20 ms then idle for 4 ms
Run 20 ms then idle for 4 ms
Run 20 ms then idle for 4 ms
time

What about the worker thread?

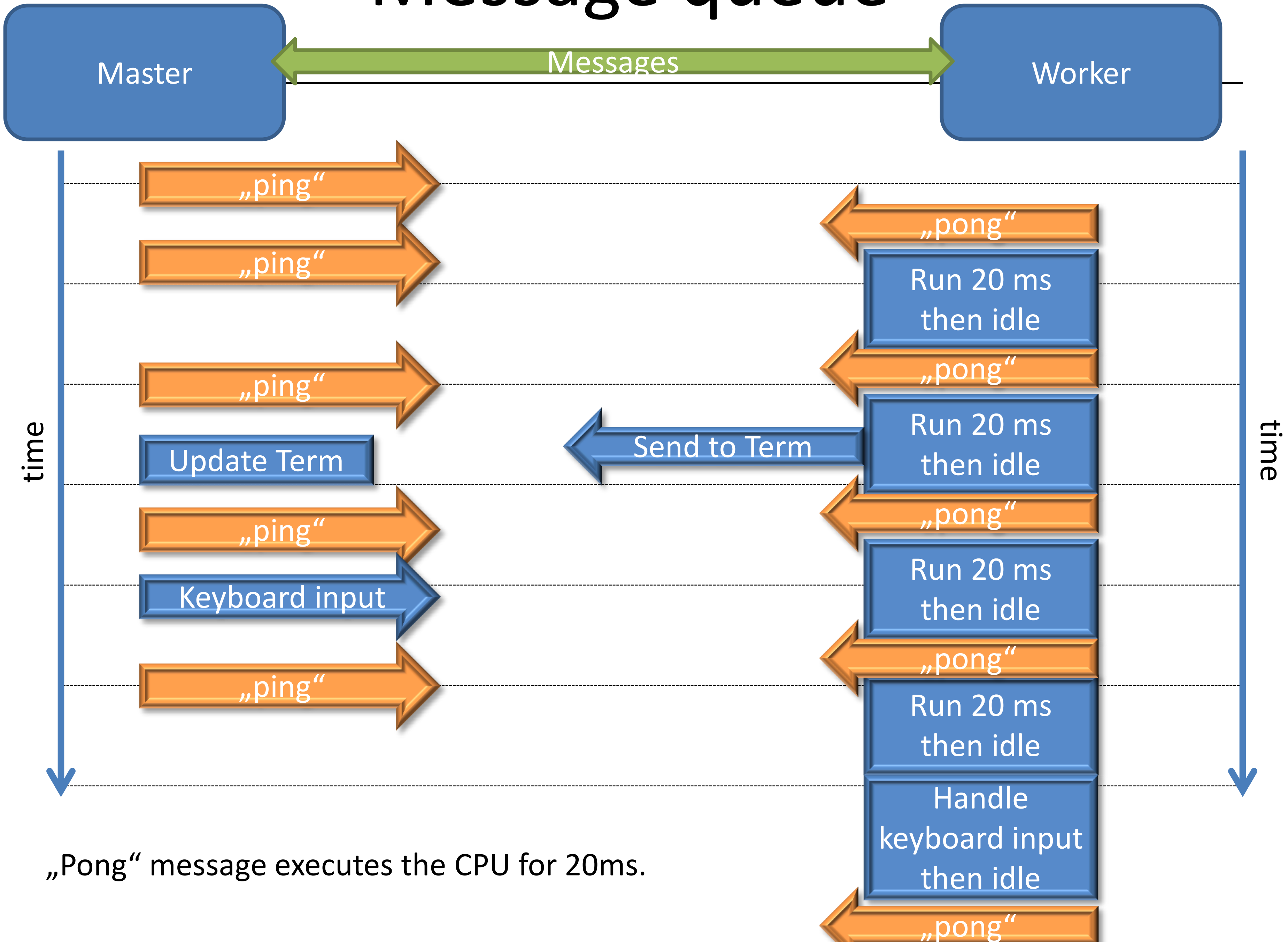
- Can we get the number of messages in the queue?

NOPE!

Hack

- **Play message ping pong, so that at least one message is always in the queue.**

Message queue



„Pong“ message executes the CPU for 20ms.

This was harmless!

Next year:
The horror of timing in JavaScript

How to implement a (streaming) audio device into JavaScript?

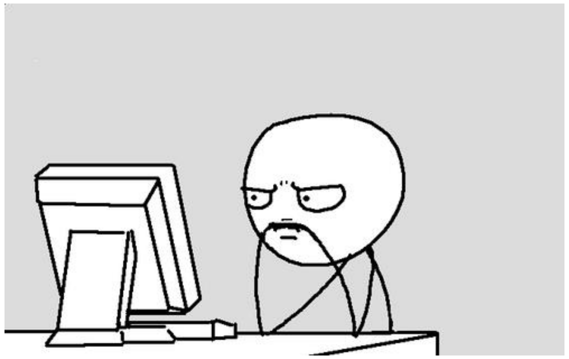
Unreliable speed of Javascript
Only millisecond time resolution
Interrupts exists only, when you are idle
Message queue between worker and master

The Filesystem

How to implement an efficient filesystem with a size of 200MB and 5000 files that runs over the internet?

The Filesystem

- How long does a "`du /`" take over the internet?
  - NFS
  - Samba
  - Sshfs
  - On demand block device



Problem is mainly latency, not throughput

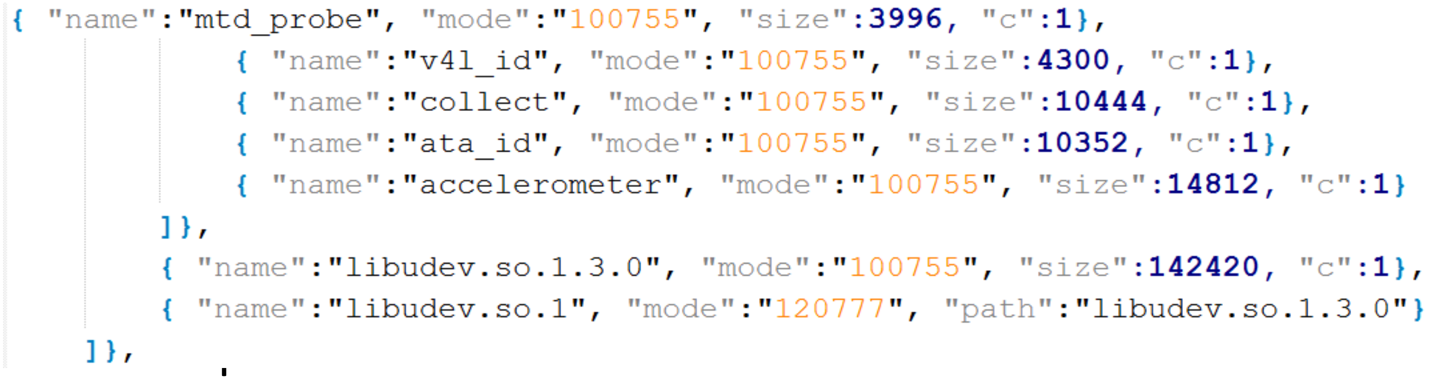
Advantages of our filesystem:

- Read only filesystem on server

The Filesystem

- Implement filesystem outside of the emulator to have full control
  - tmpfs like. Use virtio/9p to exchange commands with Linux

- Load the filesystem layout and metadata during the Linux boot process.

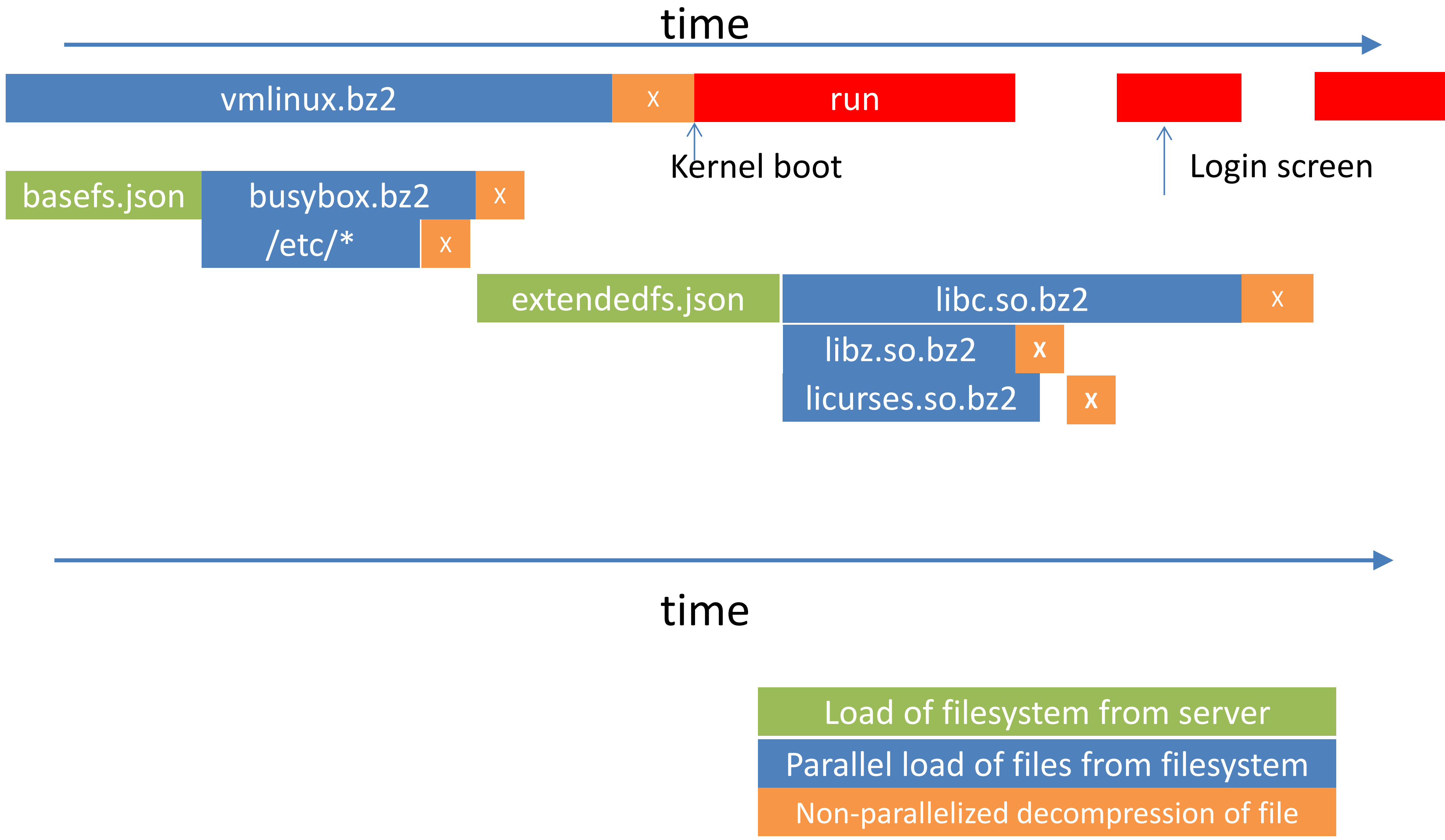


```
{ "name":"mtd_probe", "mode":"100755", "size":3996, "c":1},
            { "name":"v4l_id", "mode":"100755", "size":4300, "c":1},
            { "name":"collect", "mode":"100755", "size":10444, "c":1},
            { "name":"ata_id", "mode":"100755", "size":10352, "c":1},
            { "name":"accelerometer", "mode":"100755", "size":14812, "c":1}
        ]},
        { "name":"libudev.so.1.3.0", "mode":"100755", "size":142420, "c":1},
        { "name":"libudev.so.1", "mode":"120777", "path":"libudev.so.1.3.0"}
    ]},
```

-
  - OpenRISC binaries compress really well
  - .bz2 currently, in future .xz
  - Ordinary web server needed

- Future: dependencies between files, packages
  - http 2.0 will help here

Boot process timeline



Additional features of the filesystem

- Atomic file operations
- Full control from the outside
- Watching Files
- Upload files into home folder
- Download home folder (as .tar)
- My own cloud: Sync with server
  - Unique user id (`http://s-macke.github.io/jor1k/?`**`user=cdqKKPxjfa`**)
  - Currently 1MB quota
  - server only needs upload.php

Network

- *Yo dawg, I heard you like browsing the web, so I put a browser in your browser so you can browse while you browse!* (Twitter user Scott Elcomb)

Network

- Server in the USA
  - connected via websockets
  - Sending and receiving ethernet frames connected to a Linux TAP device
- Full working intranet
  - Start jor1k in two windows and open a ssh session between them.
- Major network applications available
  - wget, curl, nc, ping, traceroute, telnet, ssh, nmap
  - Openssl with certificates
  - Web browsers: lynx, links, dillo

Is the emulator useful?

- Well, sort of ...
  - Technology demonstration, Advertisement
  - Online interactive tutorials
  - Easy way to port and present terminal software
  - Teaching programming languages
  - Rogue like Network access
  - Fast testing environment for binaries
  - JavaScript benchmark
  - You can play games like Doom, Monkey Island, Elite II and Toppler

Outlook

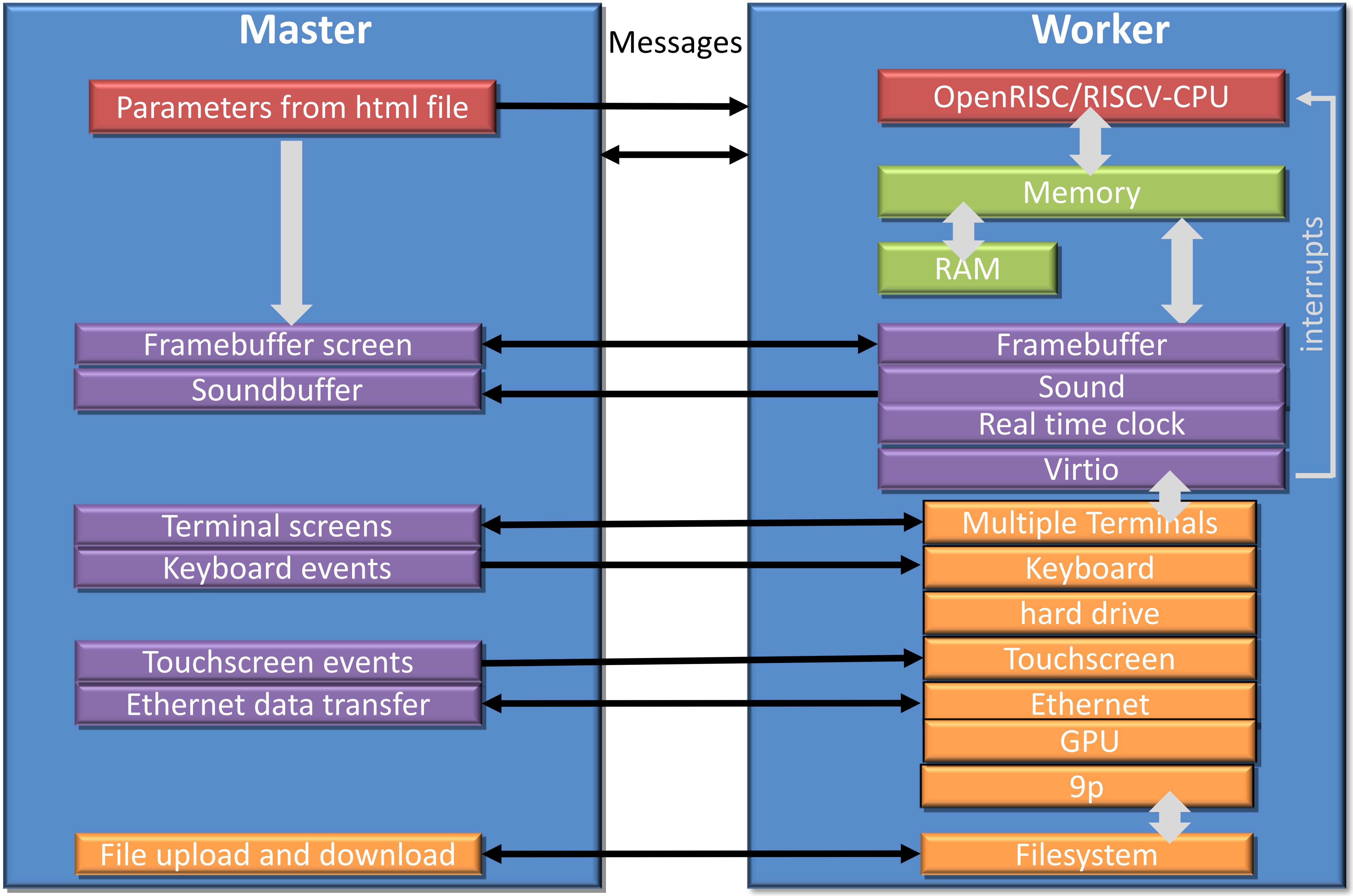
„Future“ slide from last year

- Sound ✓
- SMP ✓
- Run Debian (just one bug left) **X**
- Run Firefox **X** **(compiles but crashes and too big)**
- Status, statistics and debug screen **X**
- Download already booted Linux (state file) **X**
- More terminals, better user interface, direct **X** access to the filesystem tree
- Dynamic recompiler with the eval function? ✓ **(slow)**

Future 2.0

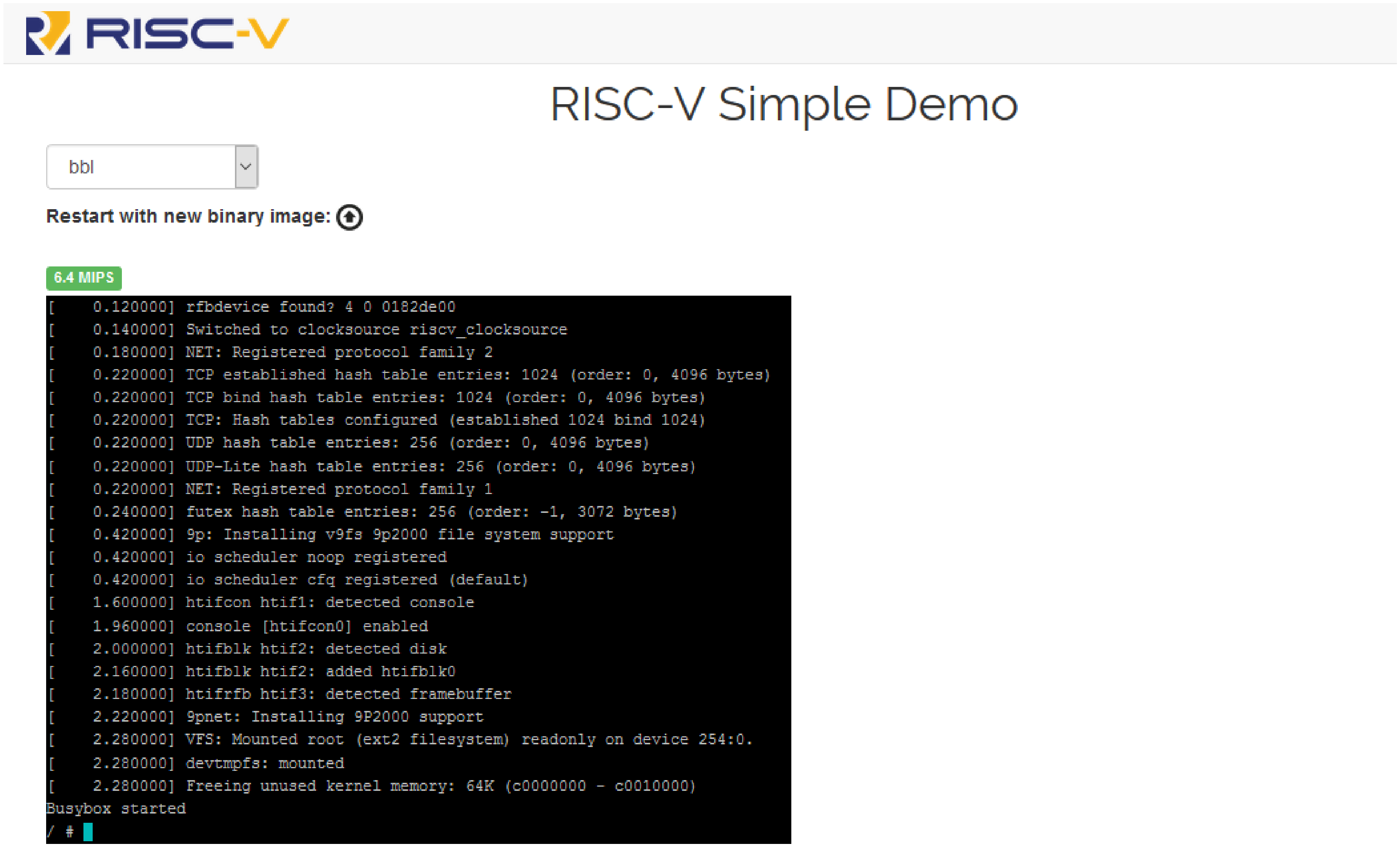
- Full virtio driver support (done)
- Virtio-GPU and full-screen X-Window system
- Implementation of virtio for RiSC-V
- More terminals, better user interface
- Download already booted Linux (state file)
- Status, statistics and debug screen
- 64-Bit RISC-V
- Run Firefox

Modules in the near future



Thanks

- **Stefan Kristiansson** for the toolchain and infinite help in the chat.
- **Ben Burns** for implementing the network and providing the relay server
- **Prannoy Pilligundla** for implementing RISC-V
- **Lawrence Angrave and Neelabh Gupta** for the C-development website
- **Jonas Bonn** for the Linux kernel support
- **Christian Svensson** for the OpenRISC Debian distribution



RISC-V Simple Demo

```
[    0.120000] rfbdevice found? 4 0 0182de00
[    0.140000] Switched to clocksource riscv_clocksource
[    0.180000] NET: Registered protocol family 2
[    0.220000] TCP established hash table entries: 1024 (order: 0, 4096 bytes)
[    0.220000] TCP bind hash table entries: 1024 (order: 0, 4096 bytes)
[    0.220000] TCP: Hash tables configured (established 1024 bind 1024)
[    0.220000] UDP hash table entries: 256 (order: 0, 4096 bytes)
[    0.220000] UDP-Lite hash table entries: 256 (order: 0, 4096 bytes)
[    0.220000] NET: Registered protocol family 1
[    0.240000] futex hash table entries: 256 (order: -1, 3072 bytes)
[    0.420000] 9p: Installing v9fs 9p2000 file system support
[    0.420000] io scheduler noop registered
[    0.420000] io scheduler cfq registered (default)
[    1.600000] htifcon htif1: detected console
[    1.960000] console [htifcon0] enabled
[    2.000000] htifblk htif2: detected disk
[    2.160000] htifblk htif2: added htifblk0
[    2.180000] htifrfb htif3: detected framebuffer
[    2.220000] 9pnet: Installing 9P2000 support
[    2.280000] VFS: Mounted root (ext2 filesystem) readonly on device 254:0.
[    2.280000] devtmpfs: mounted
[    2.280000] Freeing unused kernel memory: 64K (c0000000 - c0010000)
Busybox started
/ #
```

Idea for a good name?

- FEW - the Fastest Emulator of the Web.
- RISE - RISCV Instruction Set Emulator
- ORE - Online RISCV Emulator
- RETRO - RISCV Emulator That ROcks
- RETRO - RISCV Emulator That Runs Online
- ERIN - Emulation of RISC-V in a nutshell
- OVER - Online Version of RISC-V
- OVER - Online Versatile Emulator of RISC-V
- FEAR – Fastest emulator around the R????

Thanks for your attention!