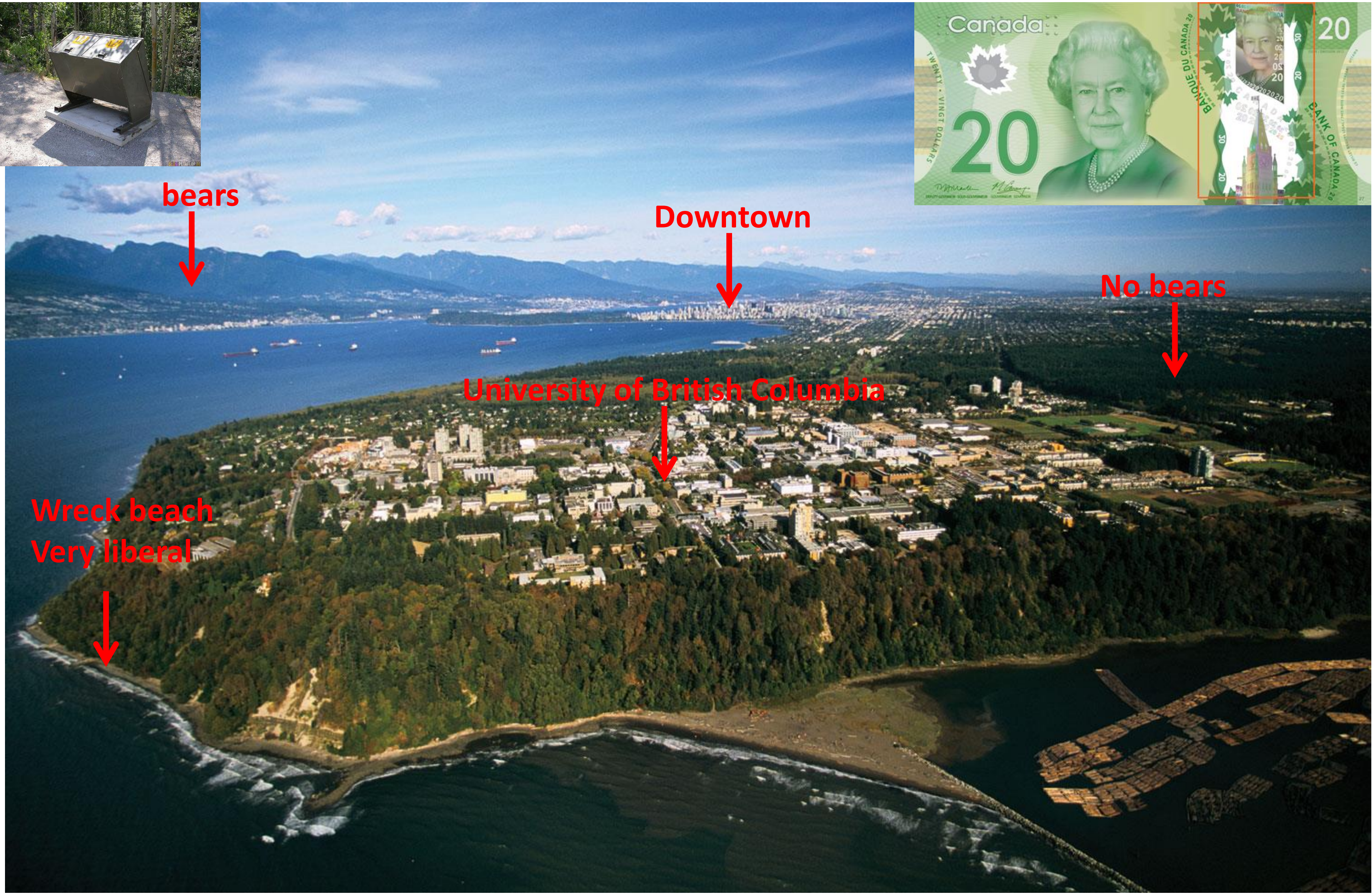
Writing a fast OpenRISC emulator in JavaScript – fun and pain

Sebastian Macke

Vancouver



bears

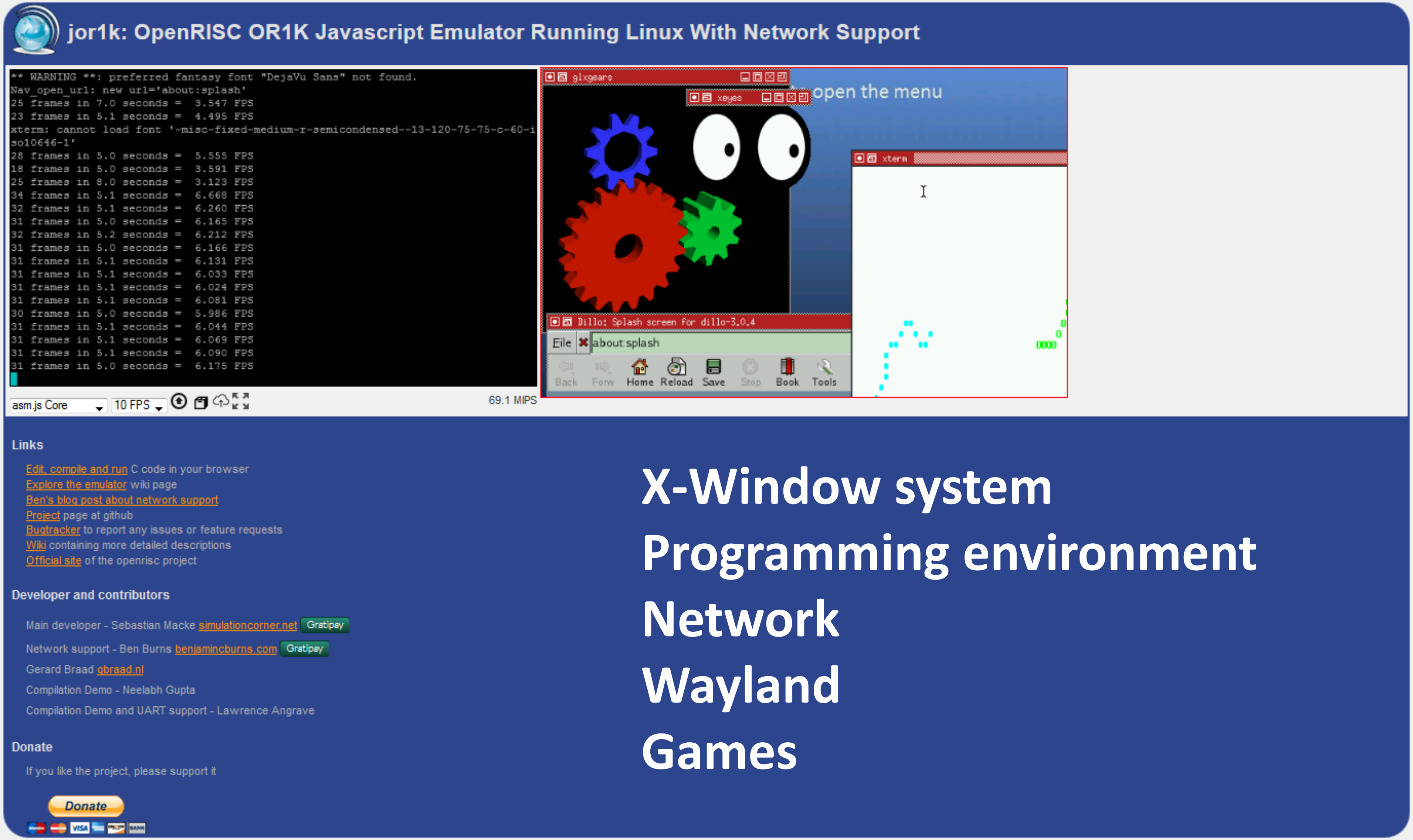
Downtown

No bears

University of British Columbia

Wreck beach
Very liberal

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

Dillo: Splash screen for dillo-3.0.4

about:splash

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 Gratipay

Network support - Ben Burns benjamincburns.com Gratipay

Gerard Braad gbraad.nl

Compilation Demo - Neelabh Gupta

Compilation Demo and UART support - Lawrence Angrave

Donate

If you like the project, please support it

Donate

X-Window system

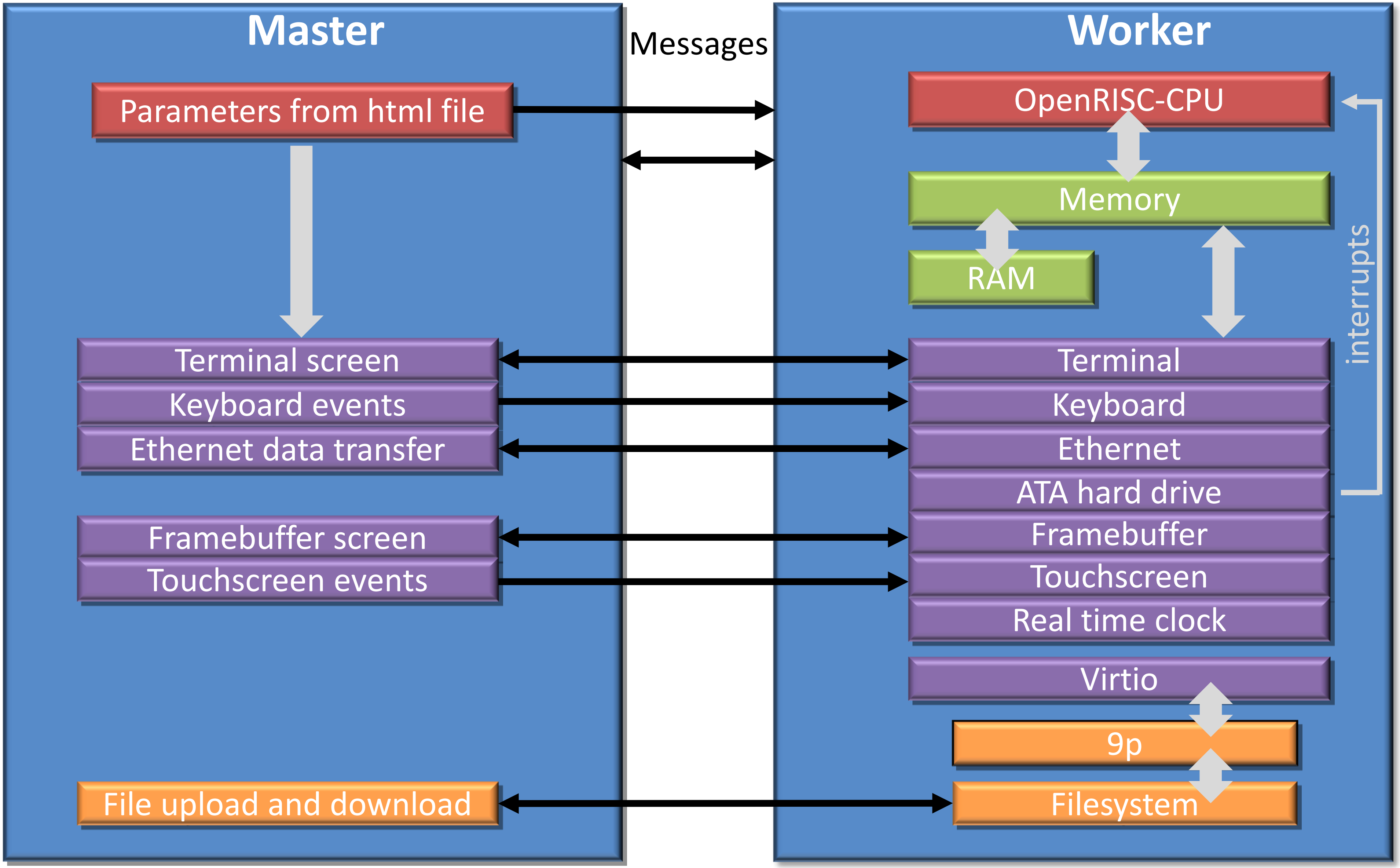
Programming environment

Network

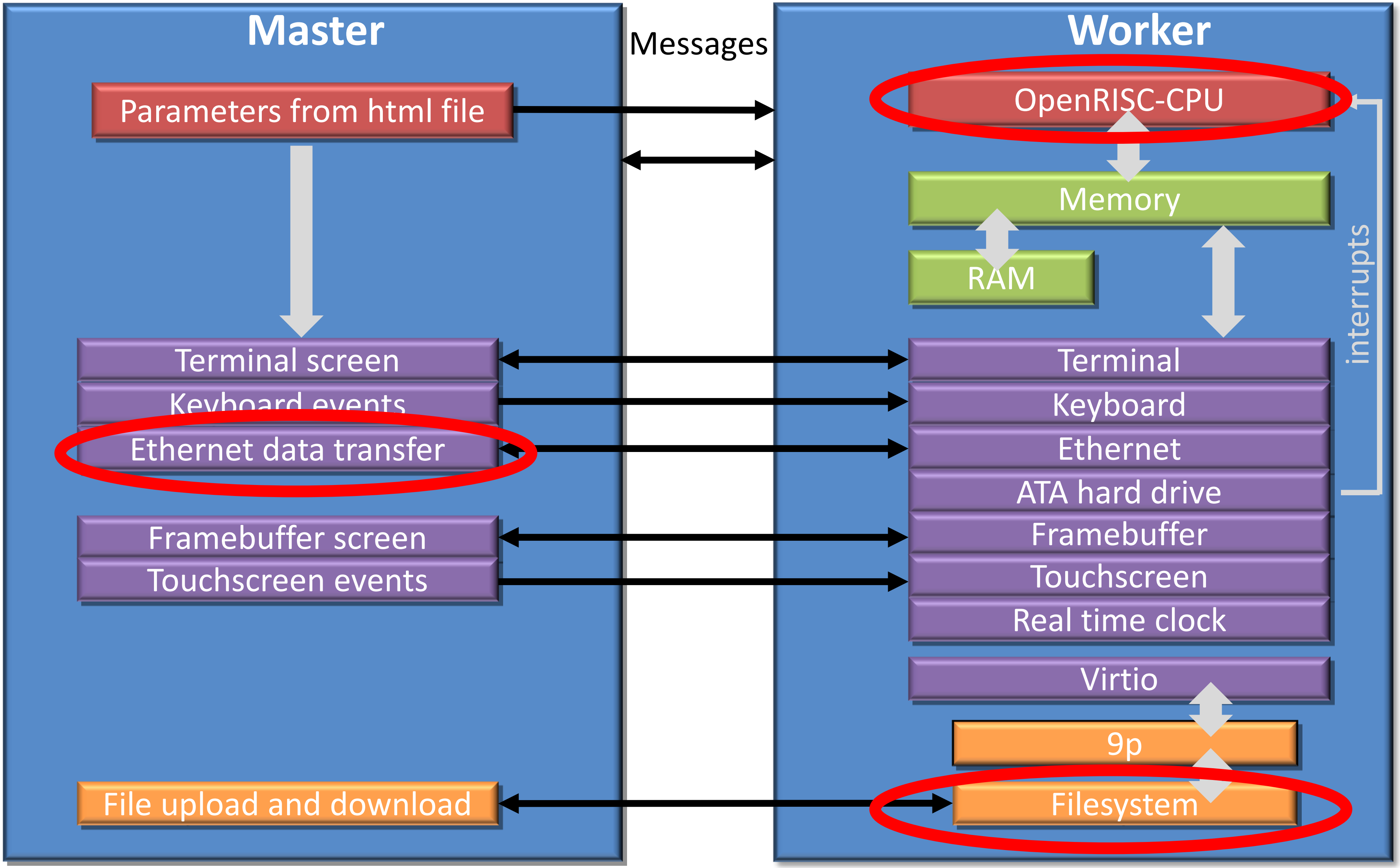
Wayland

Games

JavaScript Modules



JavaScript Modules



Master

Parameters from html file

Terminal screen

Keyboard events

Ethernet data transfer

Framebuffer screen

Touchscreen events

File upload and download

Messages

Worker

OpenRISC-CPU

Memory

RAM

Terminal

Keyboard

Ethernet

ATA hard drive

Framebuffer

Touchscreen

Real time clock

Virtio

9p

Filesystem

interrupts

Why JavaScript?

- **It runs everywhere**
- Everything which takes more than one click to show does not get much attention.
- It is considered as a bad language, which is sadly true.
  - `[1,2,3]+[4,5,6] => 1,2,34,5,6`
  - `0 == “” => true`
- But the language is better than its reputation.
- At least four companies are writing optimized compilers to squeeze out the maximum performance.

Javascript is not typed

- There are no integers, only doubles, but the compilers try to optimize it
  - `y = 9999999999999999 => y = 10000000000000000`
    (double) (double)
  - `y = 0xFFFFFFFF+1 => y = 0x100000000`
    (integer) (deoptimized into double)
  - `y = 0x7FFFFFFF+1 => y = 0x80000000`
    (Integer) (Int? 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)`
- Ways to optimize:
  - Write like it would be a typed language
  - Take care, that deoptimizations to doubles never occur
    - `y = (0xFFFFFFFF+1)|0 => y= 0x0`
      (integer) (integer)

What is asm.js

- The mode **`“use strict”;`** adds restrictions to JavaScript like additional error messages for accessing undefined variables.
- The mode **`“use asm”;`** adds additional error messages to give you a guarantee for typed variables that must be compiled only once.
  - Only a subset of Javascript is allowed
  - Fully compatible
- Implemented in Firefox in 2013

What is asm.js

Why just error messages?

Firefox with asm:	75.5 MIPS
Firefox without asm:	58.1 MIPS
Chrome (no support for asm):	60.7 MIPS
IE 11 (no support for asm):	68.3 MIPS
Safari on iPAD:	81.0 MIPS

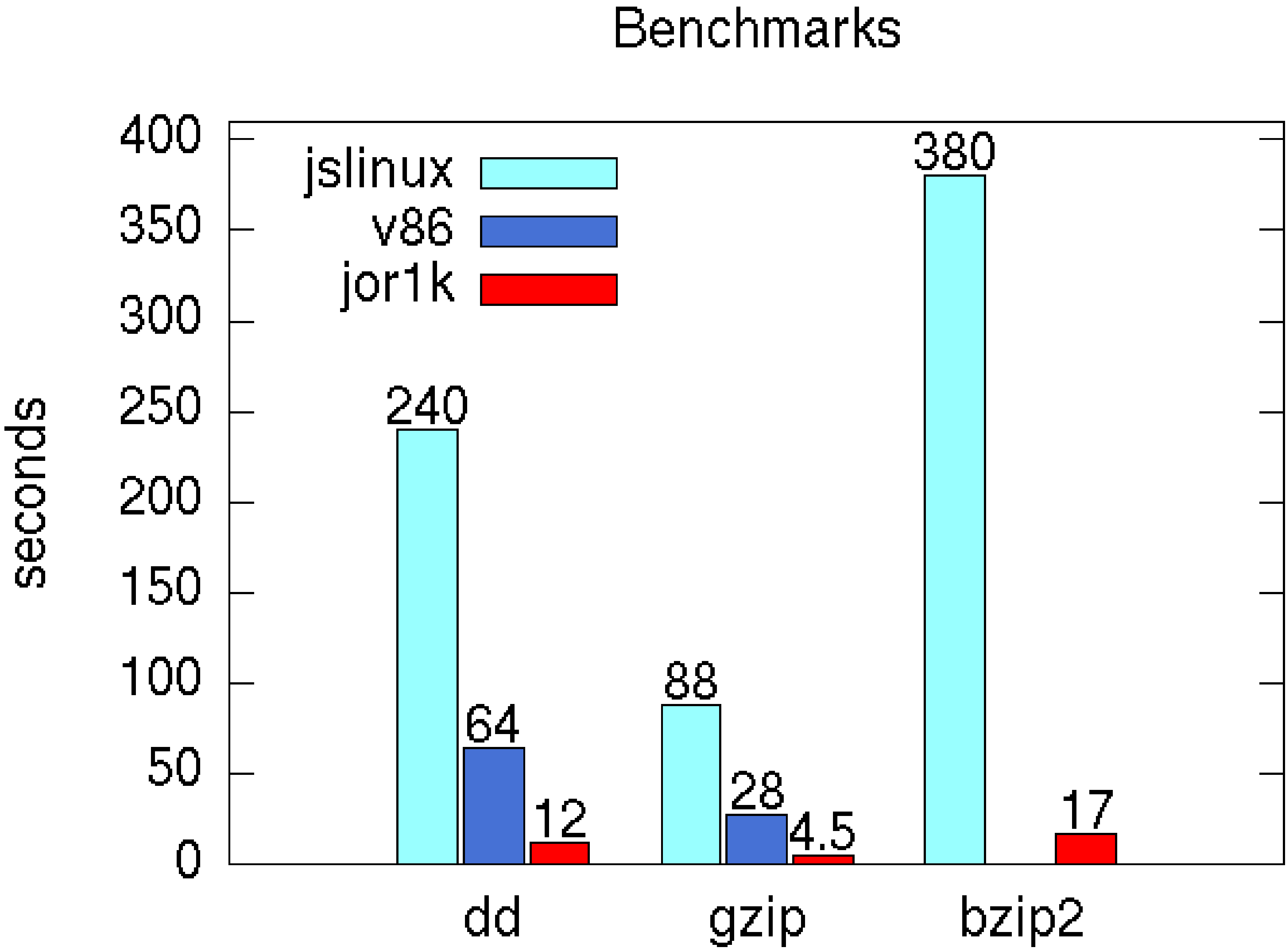
  - Fully compatible

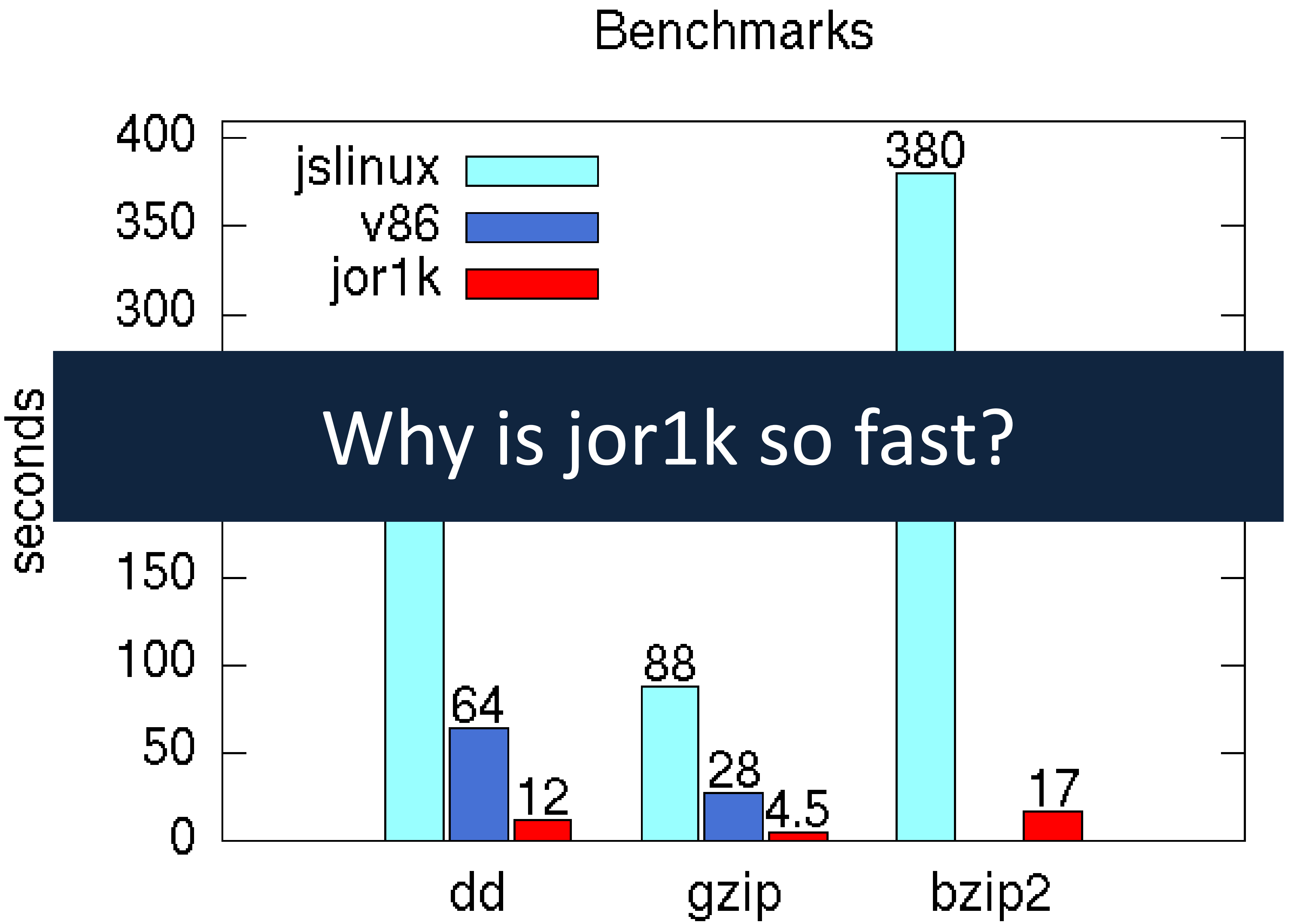
- Implemented in Firefox in 2013

What is asm.js

- But the syntax is nasty
  - `group0[SPR_IMMUCFGR] = 0x18;`
  - `h[group0p + (SPR_IMMUCFGR<<2) >> 2] = 0x18|0;`
- `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
  - Switch-case is used instead of goto

The CPU


Benchmarks
seconds
400
350
300
250
200
150
100
50
0
jslinux
v86
jor1k
240
64
12
88
28
4.5
380
17
dd
gzip
bzip2



Why is jor1k so fast?

Architecture

- OpenRISC is easy
  - No history
  - Almost no side effects

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

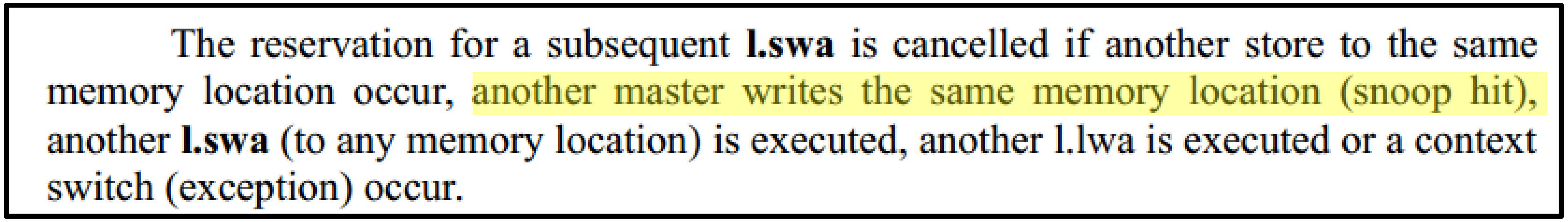
=> The instruction set is more like bytecode.

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);
•     }
```

Neglecting unused features

- CPU flags are not used
- Unaligned memory accesses are not checked
- Snoop hit never happens



The reservation for a subsequent **l.swa** is cancelled if another store to the same memory location occur, another master writes the same memory location (snoop hit), another **l.swa** (to any memory location) is executed, another l.lwa is executed or a context switch (exception) occur.

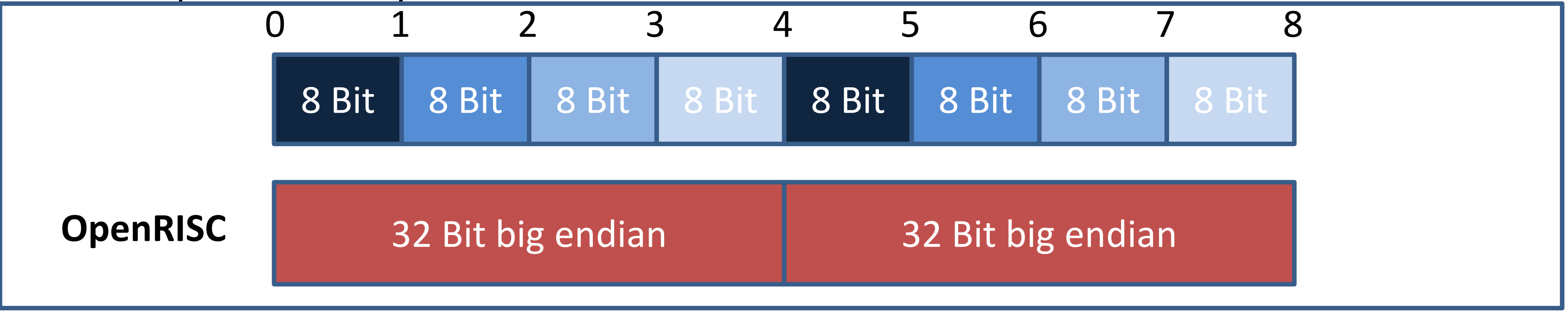
This would add an additional check to the load and store instructions.

Are there any downsides of the architecture to write a performant emulator?

BIG endian on little endian machines

JavaScript allows different views of typed arrays:

JavaScript runs mainly on little endian machines.



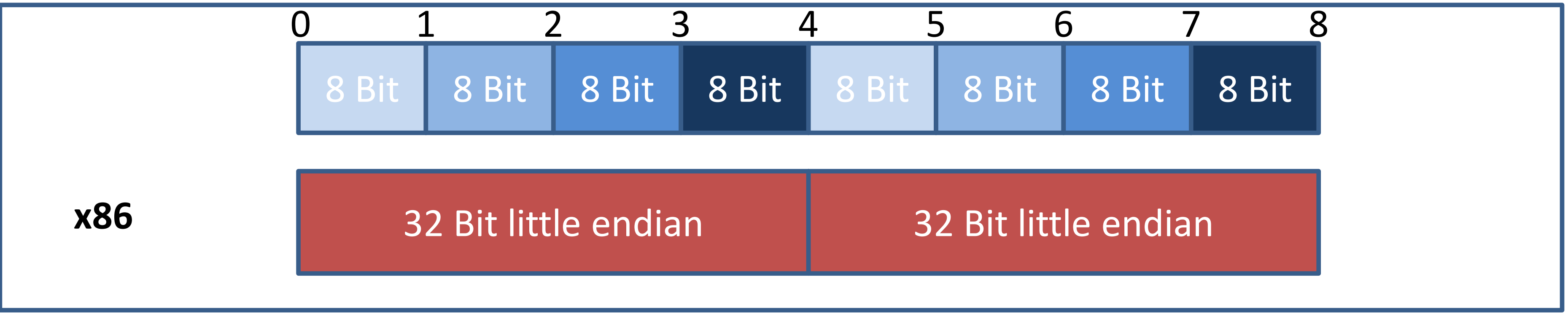
0 1 2 3 4 5 6 7 8

8 Bit 8 Bit 8 Bit 8 Bit 8 Bit 8 Bit 8 Bit 8 Bit

OpenRISC

32 Bit big endian 32 Bit big endian

But all memory accesses in the emulator are aligned and 32 Bit. So flip every 32-Bit word.



0 1 2 3 4 5 6 7 8

8 Bit 8 Bit 8 Bit 8 Bit 8 Bit 8 Bit 8 Bit 8 Bit

x86

32 Bit little endian 32 Bit little endian

Correction table
For little endian
machines

Memory access bits	Access
32	`w[addr]`
16	`h[addr^2]`
8	`b[addr^3]`

w[]: 32-Bit view of RAM

h[]: 16-Bit view of RAM

b[]: 8-Bit view of RAM

The MMU: Software TLB lockup

- Usually implemented in two stages
  1. Full translation table in memory
  2. Small translation lookaside buffer (TLB) in the CPU
     - Usually filled in software => But code part translated to JavaScript
- Add third stage
  3. TLB variables (tlb buffer with one entry)

Translation fastpath of virtual to physical addresses:

```
if ((tlb_check ^ virtual_addr) >> 13)
{
    ...
    tlb_check = ...
    tlb_trans = ...
}
physical_addr = tlb_trans ^ virtual_addr;
```

Overhead of the delayed instruction

- Usual instruction pointer increment command line: “`pc += 4;`”
- With delayed instruction support this would turn into

  -
    ```
    pc = nextpc;
    nextpc += 4;
    ```

- But currently the fastpath for one instruction looks like this:

  -
    ```
    for(;;) {
      if (ppc == fence) {
        ....
      }
      ins = int32ram[ppc >> 2];
      ppc = ppc + 4;

      switch ((ins >> 26)&0x3F) {
        ....
      }
    }
    ```
  - 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.
  - The delayed instruction gives not additional overhead

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

Grab a cup of coffee

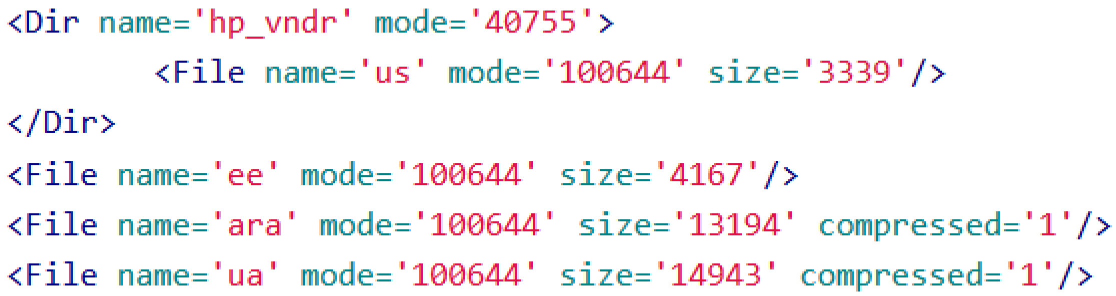
Problem is mainly latency, not throughput

Advantages of our filesystem:

- Read only filesystem on server
- Only one user

The Filesystem

- Implement filesystem outside of the emulator
  - tmpfs like. Use virtio/9p to exchange commands with Linux
- Load the filesystem layout and metadata during the Linux boot process.

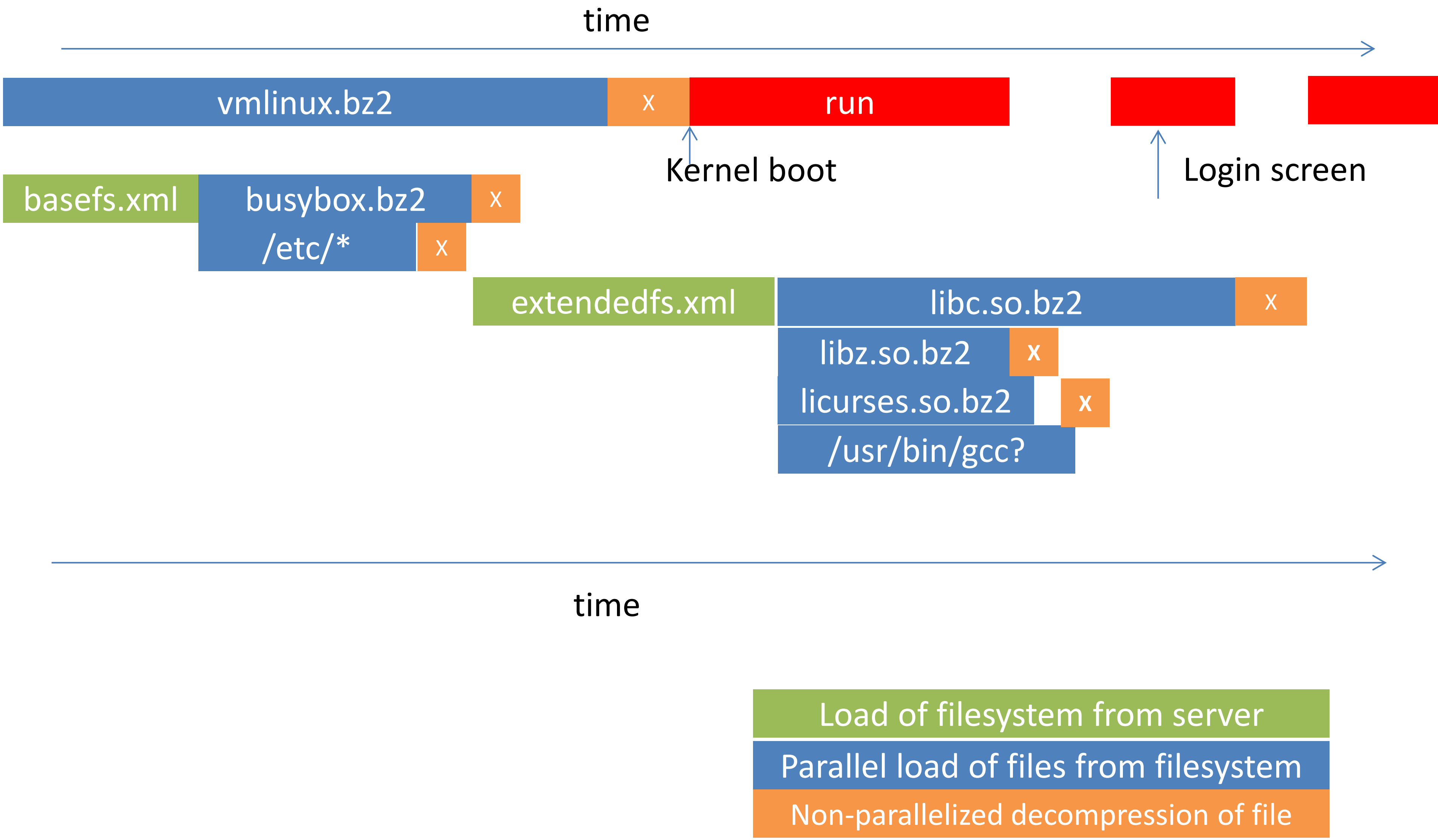


```
<Dir name='hp_vndr' mode='40755'>
        <File name='us' mode='100644' size='3339'/>
</Dir>
<File name='ee' mode='100644' size='4167'/>
<File name='ara' mode='100644' size='13194' compressed='1'/>
<File name='ua' mode='100644' size='14943' compressed='1'/>
```

- Load compressed files on demand.
  - 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

Booting process timeline



🟩 Load of filesystem from server

🟦 Parallel load of files from filesystem

🟧 Non-parallelized decompression of file

Additional features of the filesystem

- Upload files into home folder
- Download home folder (as .tar)
- 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

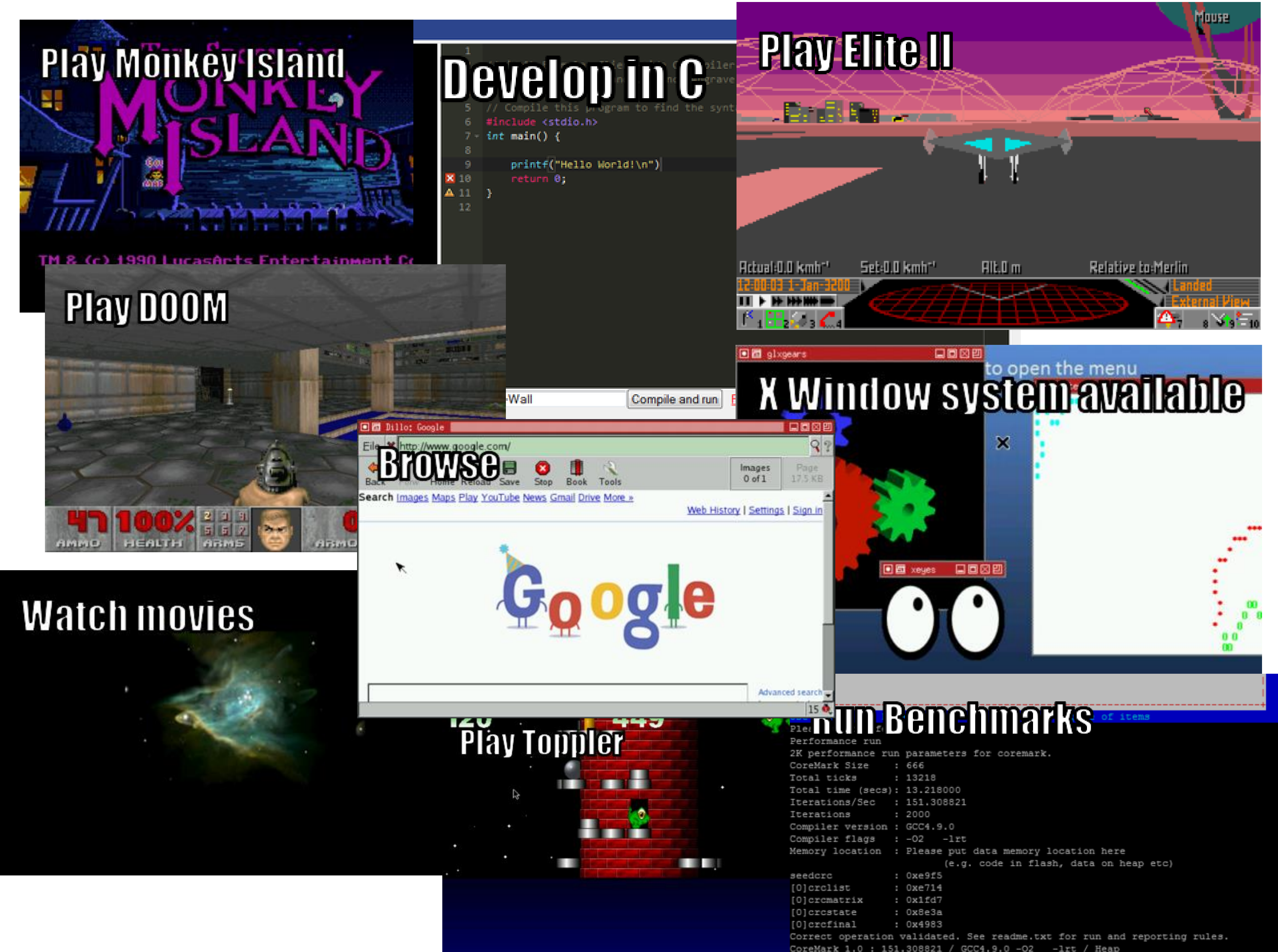
Future

- Sound (implemented but not activated)
- SMP
- Run Debian (just one bug left)
- Run Firefox (70% compiles)
- Status, statistics and debug screen
- Download already booted Linux (state file)
- More terminals, better user interface, direct access to the filesystem tree.
- Dynamic recompiler with the `eval` function?

Suggestions welcome

Thanks

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



Play Monkey Island

Develop in C

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

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

Play Elite II

Play DOOM

X Window system available

Browse

Watch movies

Play Toppler

Run Benchmarks

```
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
```