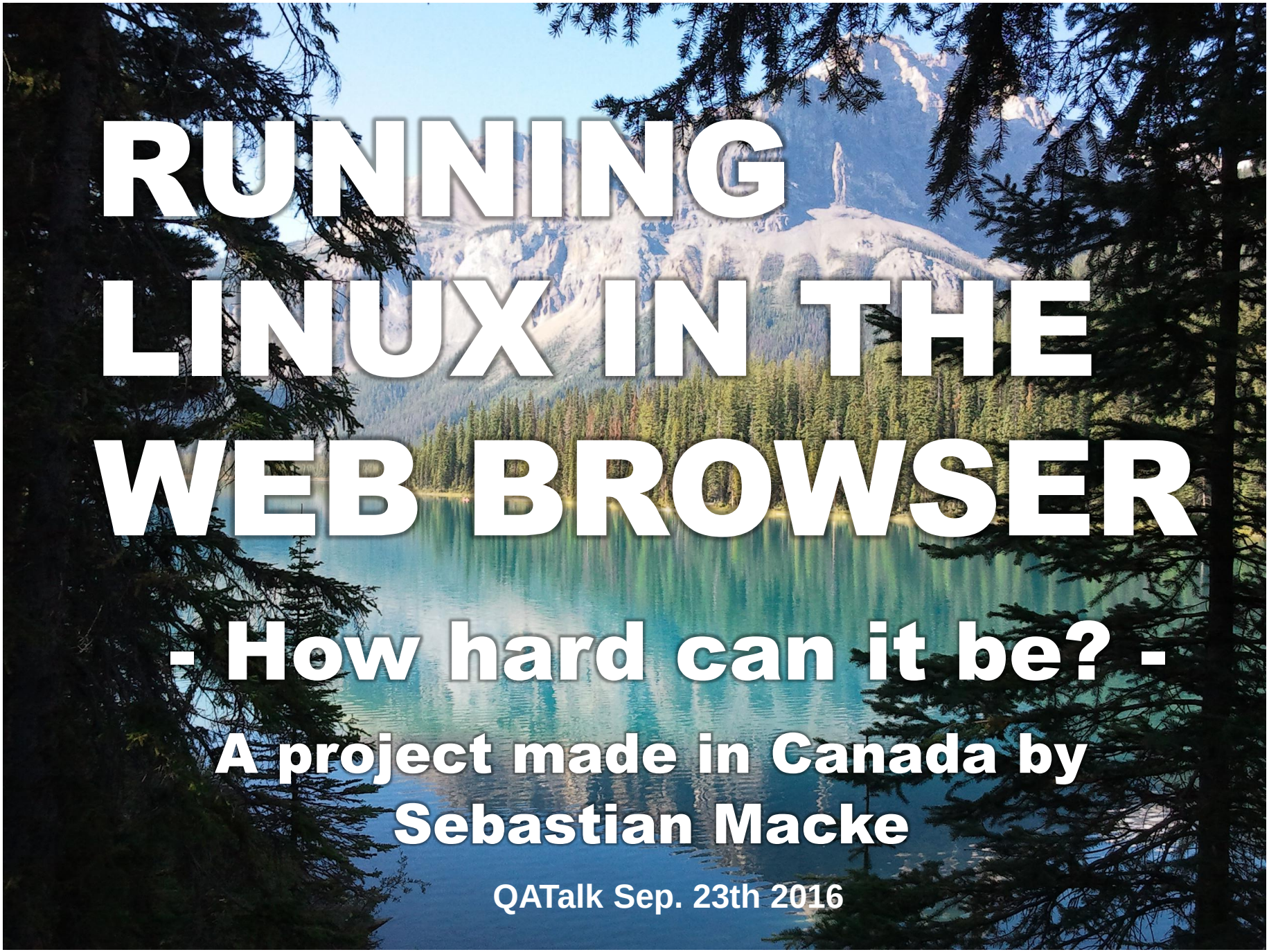

RUNNING
LINUX IN THE
WEB BROWSER
- How hard can it be? -
A project made in Canada by
Sebastian Macke
QATalk Sep. 23th 2016


VANCOUVER

bears

Downtown

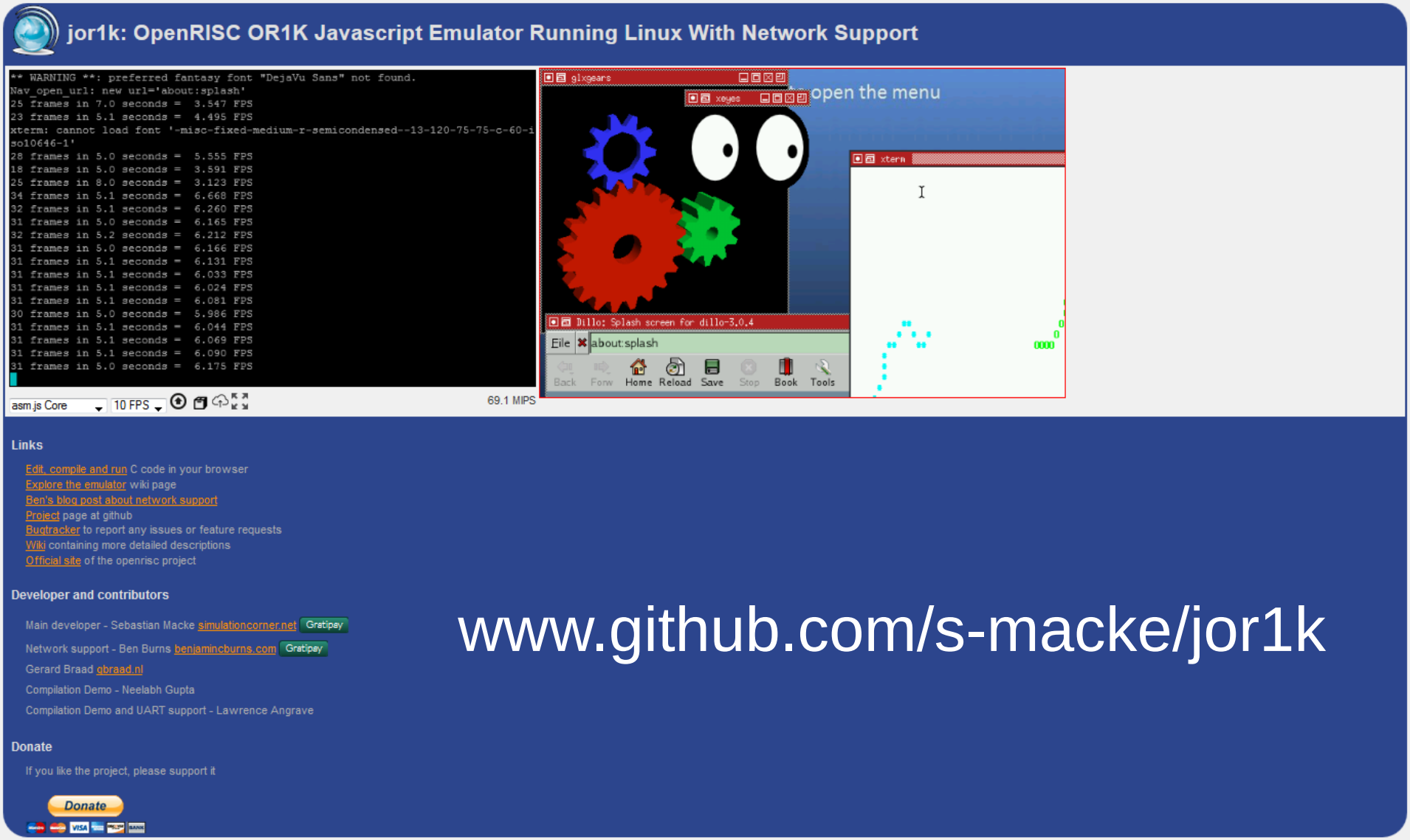
No bears

USA

University of British Columbia

Wreck beach
Very liberal

THE WEBSITE: JOR1K.COM



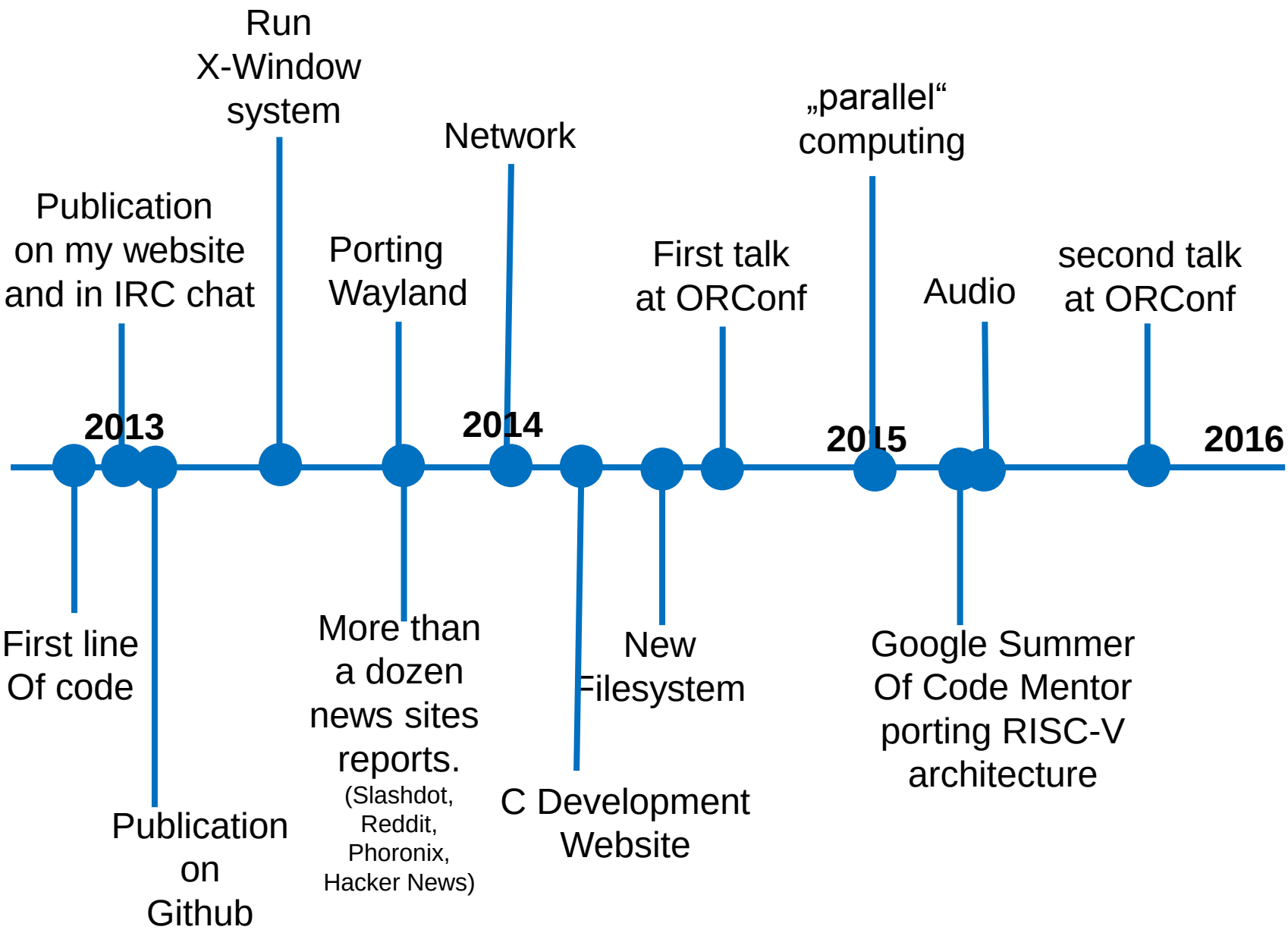
www.github.com/s-macke/jor1k

INITIAL MOTIVATION

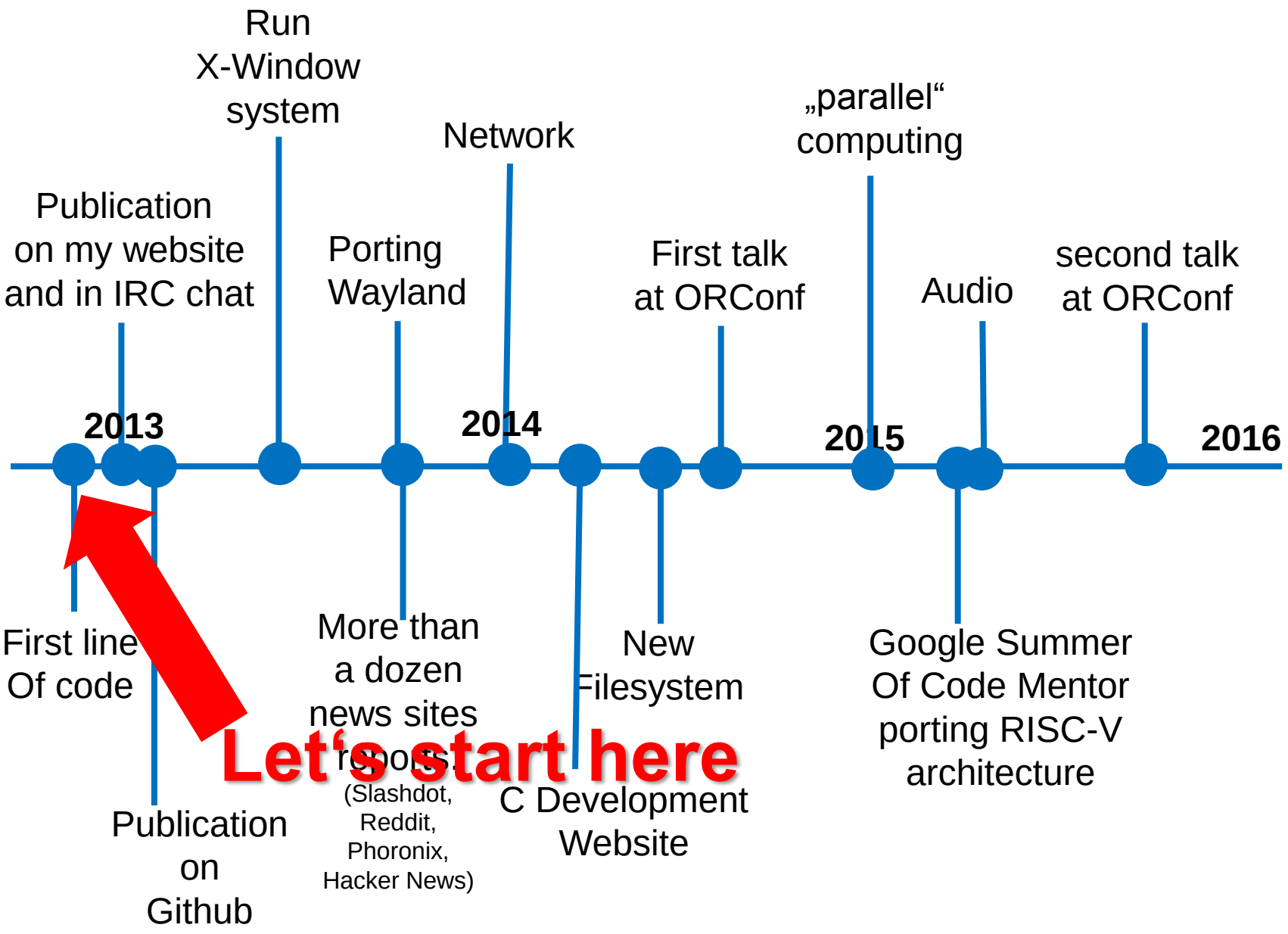
- **JavaScript is the language of the Web**
  - First true language which basically runs everywhere**.**
  - It is available immediately (one click)

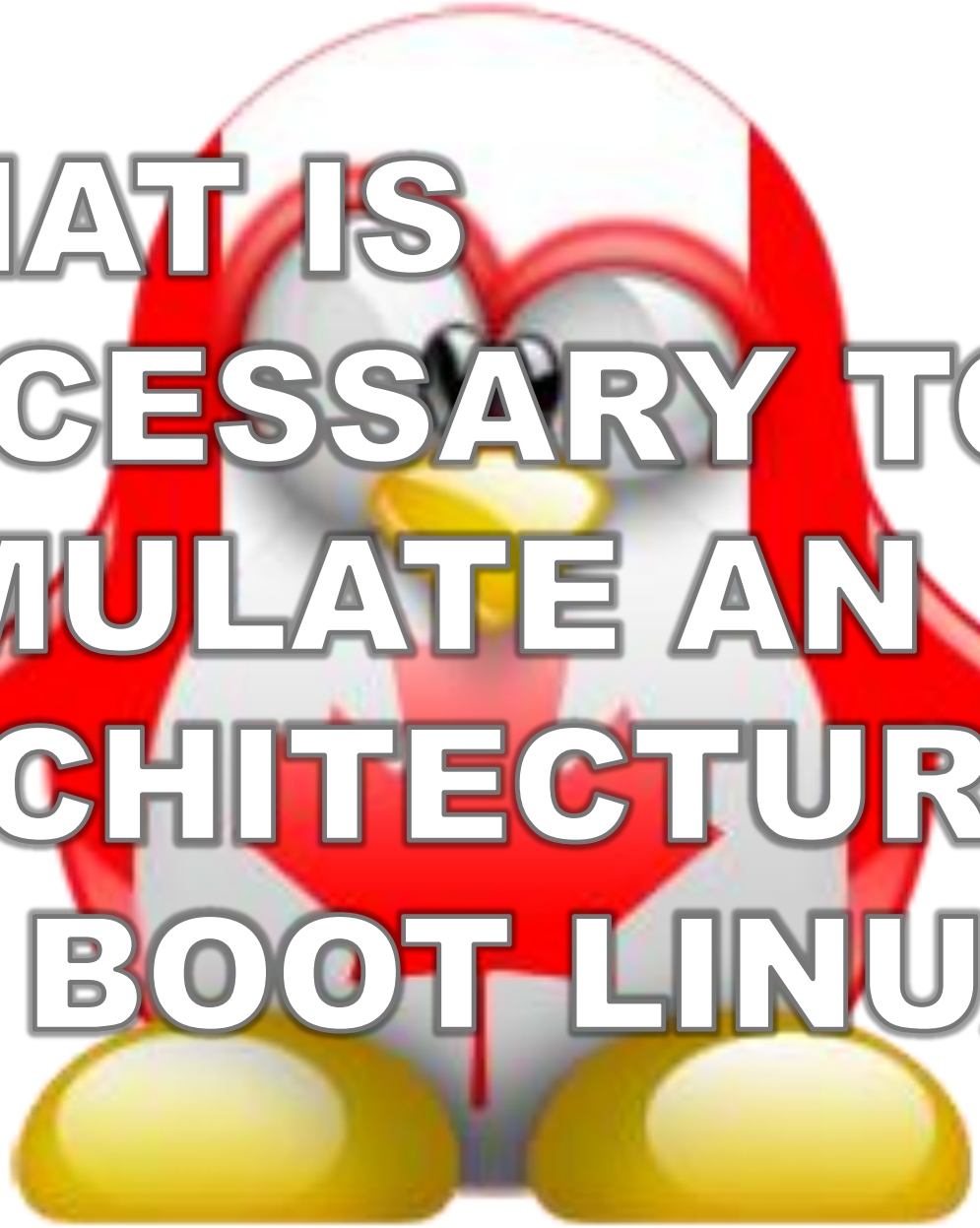
- **Writing an emulator is a fun way to learn a new language**

JOR1K TIMELINE



JOR1K TIMELINE





WHAT IS NECESSARY TO SIMULATE AN ARCHITECTURE TO BOOT LINUX?

Goal: take an easy architecture

WHICH ARCHITECTURE?

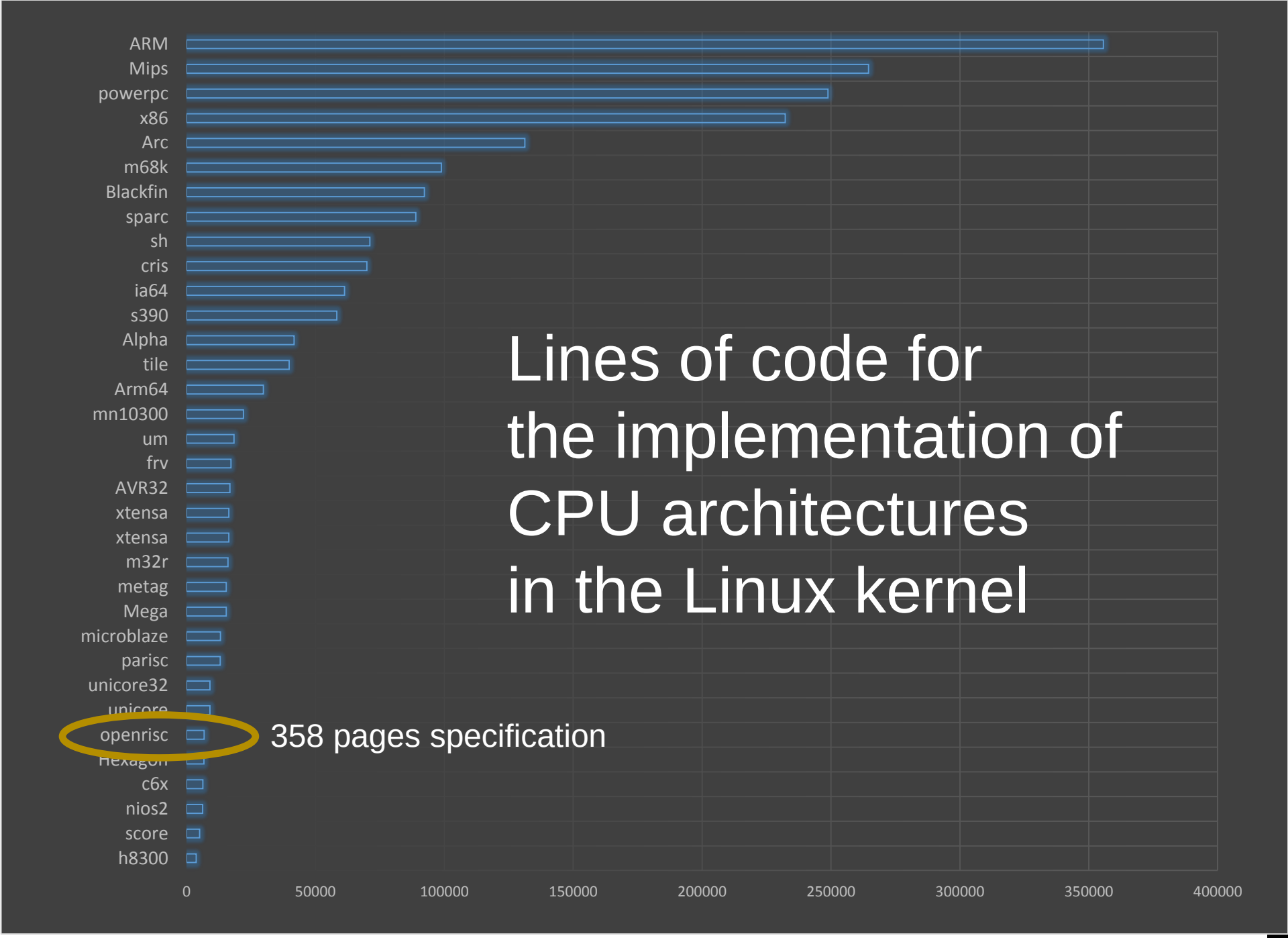
X86 architecture software developer manual

- 4618 pages
- More than 500 instructions

ARMv8 Architecture Reference Manual:

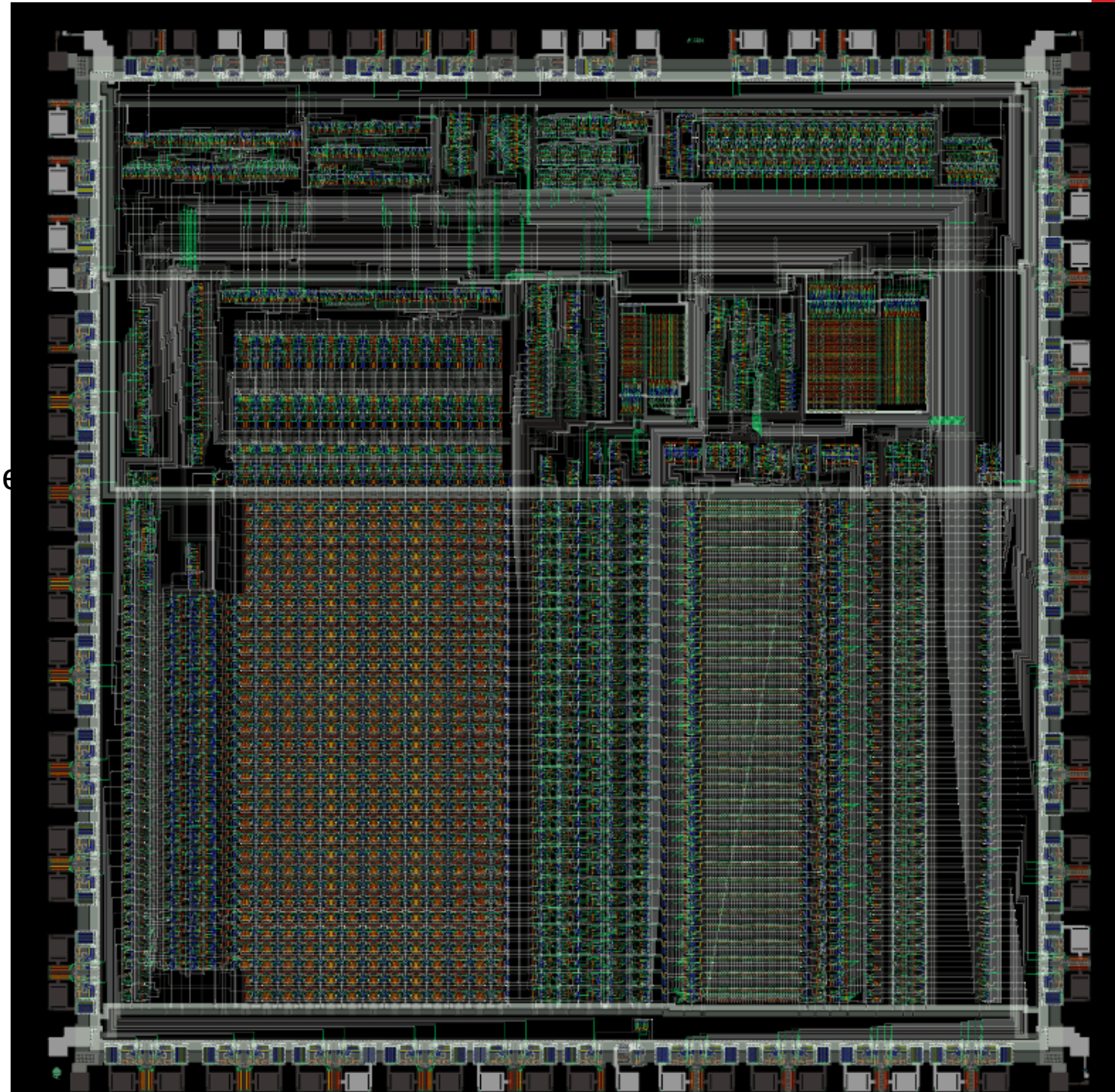
- 5740 pages
- contains 3125 times the word „unpredictable“
- contains 2290 times the word „undefined“

For comparison: MS Office file format spec ~6000 pages


ARM
Mips
powerpc
x86
Arc
m68k
Blackfin
sparc
sh
cris
ia64
s390
Alpha
tile
Arm64
mn10300
um
frv
AVR32
xtensa
xtensa
m32r
metag
Mega
microblaze
parisc
unicore32
openrisc
c6x
nios2
score
h8300
0
50000
100000
150000
200000
250000
300000
350000
400000
Lines of code for the implementation of CPU architectures in the Linux kernel
358 pages specification

SIMULATE ON THE CIRCUIT LEVEL

- ARM 1
- 25.000 transistors
- Runs at 20Hz
- 45000 lines of code



(Recorded from visual6502.org)

SIMULATE ON THE MACHINE CODE LEVEL

International Obsfucated C Code contest entry of 2013 from Adrian Cable

Intel 8086/186 CPU (29000 transistors)
1MB RAM
8072A 3.5" floppy disk controller (1.44MB/720KB)
Fixed disk controller (supports a single hard drive up to 528MB)
Hercules graphics card with 720x348 2-color graphics (64KB video RAM), and CGA 80x25 16-color text mode support
8253 programmable interval timer (PIT)
8259 programmable interrupt controller (PIC)
8042 keyboard controller with 83-key XT-style keyboard
MC146818 real-time clock
PC speaker

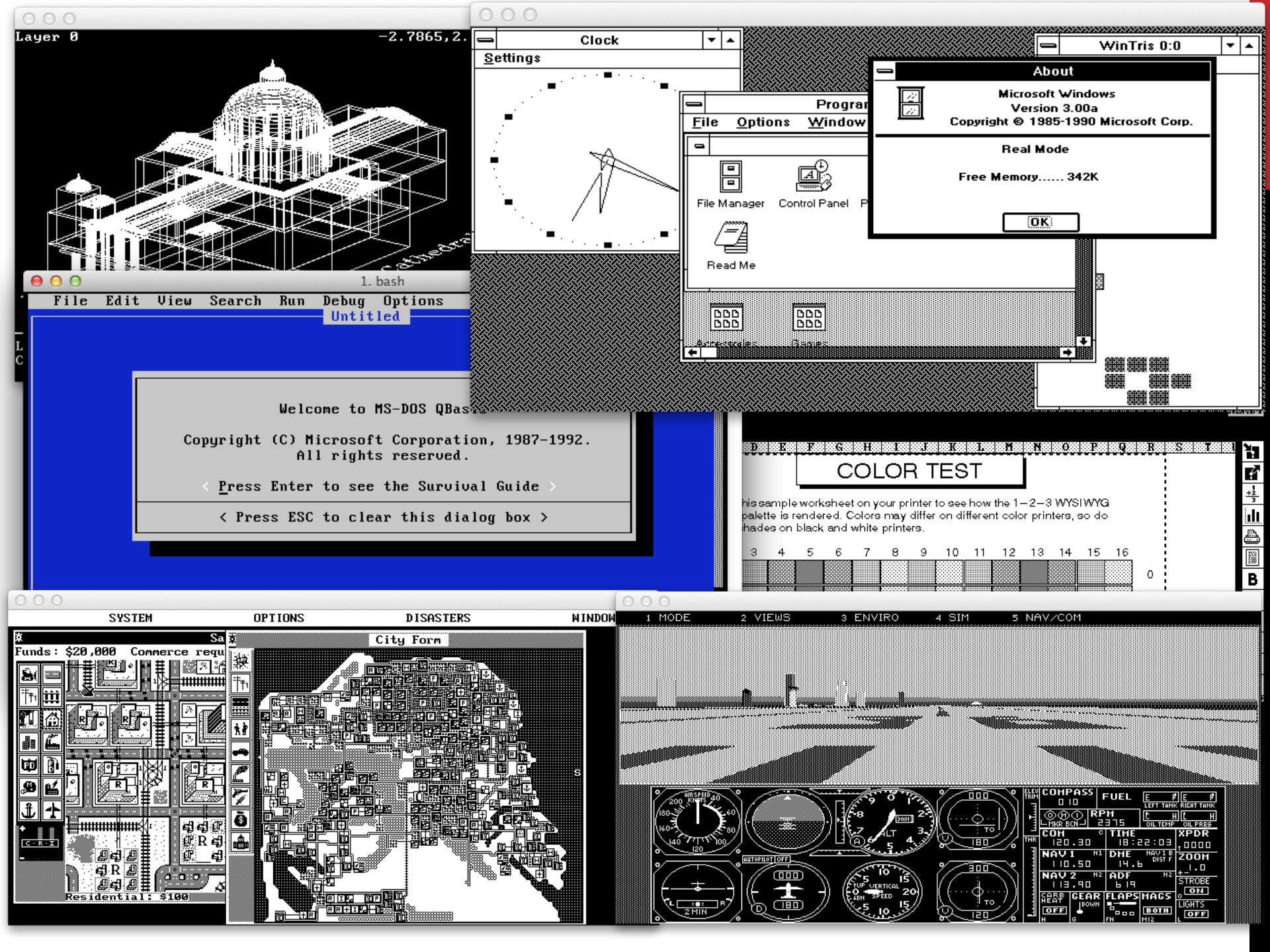
```
#include "SDL.h"

#define $ for(O=9
#define CX M+=(T%3+2*!(!T*t-6))
#define x ,A=4*!T,O=t,W=h=T<3?u(Q?p:D(A+3),D(A),D(A+1)[i]+D(A+2)*g+):K(t),U=V=K(a),o?U=h,W=V:V,
#define C 8*-~L
#define Z short
#define y a(Z)Y[++O]
#define B ),a--||(
#define _ ),e--||(

#define V(I,D,E)(O=a(I)h[r])&&!(A=(D)(V=(l[E+L]<<16)+*i)/O,A-(I)A)?l[E+L]=V-O*(*E=A):H(O)
#define i(B,M)B(o){return M;}
#define R(O,M,_)(S=L?a(I Z)O:O,N=L?a(I Z)O M(f=a(I Z)_):(O M(f=a(I n)_)))
#define T(_)R(r[u(10,L=4,--)],=,_)
#define u(a,r,T)16*i[a]+(I Z)(T i[r])
#define a(_)*(_*)&
#define L(_)M(W,_,U)

#define M(S,F,T)R(r[S],F,r[T])
#define A(_)(i[L=4]+=2,R(_,=,r[u(10,4,-2+)]))
#define c(R,T)(l[u=19,L+T]=(N=a(R)h[r]*(R)*T)>>16,*i=N,G(F(N-(R)N)))
#define h(_)(l&(L?a(Z)_:_)>>C-1)
#define I unsigned
#define n char
#define e(_)v(F(40[L(_##=40[E]+),E]&N==S|_ N<_(int)S))

I n t,e,l[80186],*E,m,u,L,a,T,o,r[1<<21],X,*Y,b,Q,R;I Z*i,M,p,q=3;I*localtime(),f,S,kb,h,W,U,c,g,d,V,A;N,O,P=983040,j[5];SDL_Surface*k;i(F,40[E]=!!o)i(
z,42[E]=!!o)i(D,r[a(I)E[259+4*o]+0])i(w,i[o]+=~(-2*47[E])*~L)i(v,G(N-S&&l&(40[z((f^=S^N)&16),E]^f>>C-1)))J(){V=61442;$;O--;)V+=40[E+0]<<D(25);}i(H,(46[
u=76,J(),T(V),T(9[i]),T(M),M(P+18,=,4*o+2),R(M,=,r[4*o]),E]=0))s(o){$;O--;)40[E+0]=l&&l<<D(25)&o;}i(BP,(*i+=262*o*z(F((*E&15)>9|42[E])),*E&=15))i(SP,(w
(7),R&&--l[i]&&o?R++,Q&&Q++,M--:O))DX(){$,O*=27840;O--;)O[(I*)k->pixels]=-!!(1<<7-O%8&r[O/2880*90+O%720/8+(88+952[l]/128*4+O/720%4<<13)]);SDL_Flip(k);}
main(BX,nE)n**nE;{9[i=E=r+P]=P>>4;$;q;)j[--q]=*++nE?open(*nE,32898):0;read(2[a(I)*i=*j?lseek(*j,0,2)>>9:0,j],E+(M=256),P);$;Y=r+16*9[i]+M,Y-r;Q|R||kb&
46[E]&&KB)--64[T=1[O=32[L=(X=*Y&7)&1,o=X/2&1,l]=0,t=(c=y)&7,a=c/8&7,Y]>>6,g=~-T?y:(n)y,d=BX=y,l],!T*t-6&&T-2?T-1?d=g:0:(d=y),Q&&Q--,R&&R--x(O=*Y,O=u=D(
51),e=D(8),m=D(14)_ O=*Y/2&7,M+=(n)c*(L^(D(m)[E]|D(22)[E]|D(23)[E]^D(24)[E]))_ L=*Y&8,R(K(X)[r],=,c)_ L=e+=3,o=0,a=X x a=m _ T(X[i])_ A(X[i])_ a<2?M(U
,+=1-2*a+,P+24),v(f=1),G(S+1-a==1<<C-1),u=u&4?19:57:a-6?CX+2,a-3||T(9[i]),a&2&&T(M),a&1&&M(P+18,=,U+2),R(M,=,U[r]),u=67:T(h[r])_(W=U B u=m,M-=~L,R(W[r
],&,d)B 0 B L(=~)B L(=-),S=0,u=22,F(N>S)B L?c(I Z,i):c(I n,E)B/**/L?c(Z,i):c(n,E)B L?V(I Z,I,i):V(I n,I Z,E)B L?V(Z,int,i):V(n,Z,E))_++e,h=P,d=c,T=3,a=m
,M--_++e,13[W=h,i]=(o|=!L)?(n)d:d,U=P+26,M-=~!o,u=17+(m=a)_(a=m B L(+=),F(N<S)B L(|=)B e(+)B e(-)B L(&=)B L(-=),F(N>S)B L(^=)B L(-),F(N>S)B L(=))_!L?L=
a+=8 x L(=):!o?Q=1,R(r[p=m x V],=,h):A(h[r])_ T=a=0,t=6,g=c x M(U,=,W)_(A=h(h[r]),V=m?++M,(n)g:o?31&2[E]:1)&&(a<4?V%=a/2+C,R(A,=,h[r]):0,a&1?R(h[r
],>>=,V):R(h[r],<<=,V),a>3?u=19:0,a<5?O:F(S>>V-1&1)B R(h[r],+=,A>>C-V),G(h(N)^F(N&1))B A&=(1<<V)-1,R(h[r],+=,A<<C-V),G(h(N*2)^F(h(N)))B R(h[r],+=(40[E
]<<V-1)+,A>>1+C-V),G(h(N)^F(A&1<<C-V))B R(h[r],+=(40[E]<<C-V)+,A<<1+C-V),F(A&1<<V-1),G(h(N)^h(N*2))B G(h(N)^F(h(S<<V-1)))B G(h(S))B 0 B V<C||F(A),G(0),
R(h[r],+=,A*=~((1<<C)-1>>V)))_(V=!!--1[a=X,i]B V&=!m[E]B V&=m[E]B 0 B V=!++1[i]),M+=V*(n)c _ M+=3-o,L?O:o?9[M=0,i]=BX:T(M),M+=o*L?(n)c:c _ M(U,&,W)_ L=
e+=8,W=P,U=K(X)_!R||1[i]?M(m<2?u(8,7,):P,=,m&1?P:u(Q?p:11,6,)),m&1||w(6),m&2||SP(1):0 _!R||1[i]?M(m?P:u(Q?p:11,6,),-,u(8,7,)),43[u=92,E]=!N,F(N>S),m||w
(6),SP(!N==b):0 _ o=L,A(M),m&&A(9[i]),m&2?s(A(V)):o||(4[i]+=c)_ R(U[r],=,d)_ 986[l]^=9,R(*E,=,l[m?2[i]:(n)c])_ R(l[m?2[i]:(n)c],=,*E)_ R=2,b=L,Q&&Q++_
W-U?L(^=),M(U,^=,W),L(^=):0 _ T(m[i])_ A(m[i])_ Q=2,p=m,R&&R++_ L=0,O=*E,F(D(m+=3*42[E]+6*40[E])),z(D(1+m)),N=*E=D(m-1)_ N=BP(m-1)_ 1[E]=-h(*E)_ 2[i]=-
h(*i)_ 9[T(9[i]),T(M+5),i]=BX,M=c _ J(),T(V)_ s(A(V))_ J(),s((V&~m)+1[E])_ J(),1[E]=V _ L=o=1 x L(=),M(P+m,=,h+2)_++M,H(3)_ M+=2,H(c&m)_++M,m[E]&&H(4)_
(c&=m)?1[E]=*E/c,N=*E%=c:H(0)_*i=N=m&E[L=0]+c*1[E]_*E=-m[E]_*E=r[u(Q?p:m,3,*E+)]_ m[E]^=1 _ E[m/2]=m&1 _ R(*E,&,c)_(a=c B write(1,E,1)B time(j+3),
memcpy(r+u(8,3,),localtime(j+3),m)),a<2?*E=~lseek(O=4[E][j],a(I)5[i]<<9,0)?((I(*)())(a?write:read))(O,r+u(8,3,),*i):0:0),O=u,D(16)?v(0):D(17)&&G(F(0)),
CX*D(20)+D(18)-D(19)*~!!L,D(15)?O=m=N,41[43[44[E]=h(N),E]=!N,E]=D(50):0,!++q?kb=1,*l?SDL_PumpEvents(),k=k?k:SDL_SetVideoMode(720,348,32,0),DX():k?
SDL_Quit(),k=0:0:0;}i(G,48[E]=o)i(K,P+(L?2*o:2*o+o/4&7))
```

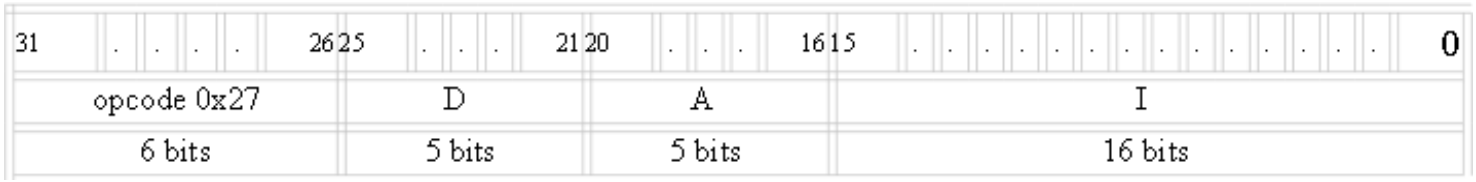

Layer 0
-2.7865,2.
Clock
Settings
WinTris 0:0
About
Microsoft Windows
Version 3.00a
Copyright © 1985-1990 Microsoft Corp.
Real Mode
Free Memory..... 342K
OK
File Manager
Control Panel
Read Me
1. bash
File Edit View Search Run Debug Options
Untitled
Welcome to MS-DOS QBas
Copyright (C) Microsoft Corporation, 1987-1992.
All rights reserved.
Press Enter to see the Survival Guide
Press ESC to clear this dialog box
COLOR TEST
his sample worksheet on your printer to see how the 1-2-3 WYSIWYG
palette is rendered. Colors may differ on different color printers, so do
hades on black and white printers.
SYSTEM
OPTIONS
DISASTERS
Funds: $20,000
City Form
Residential: $100
1 MODE
2 VIEWS
3 ENVIRO
4 SIM
5 NAV/COM

1. CPU (OPENRISC)

- **32-Bit address and data bus**
- **32 registers**
- **Instruction size of 4 byte (reduced instruction set)**
- **arithmetic and logical operations**
  - add, sub, mul, div, or, and, xor, shift left, shift right
  - three operand instructions for signed and unsigned integers
- **Load-Store (word, half-word, byte, signed and unsigned load)**
- **Conditional and unconditional branching (absolute and relative)**
- **Comparison (signed and unsigned)**
- **Two modes of operation: supervisor and user**
- **Exceptions (timer interrupts, switch of modes, bus error, reset, div by 0)**
- **Special purpose registers (Status and Control, Timer, PIC, MMU, Debug, Power Management)**
- **optional floating point**

1. A CPU (OPENRISC)

Example: **add** with immediate Half Word $(r_D = r_A + I)$




```
switch(ins >>> 26)
{
....
   case 0x27: // addi instruction
      r[(ins>>>21)&0x1F] = r[(ins>>>16)&0x1F] + (ins  << 16  >> 16);
      break;
....
}
```

Sign extension from 16-Bit to 32-Bit

Instruction emulation implementation: ~ 600 lines of code

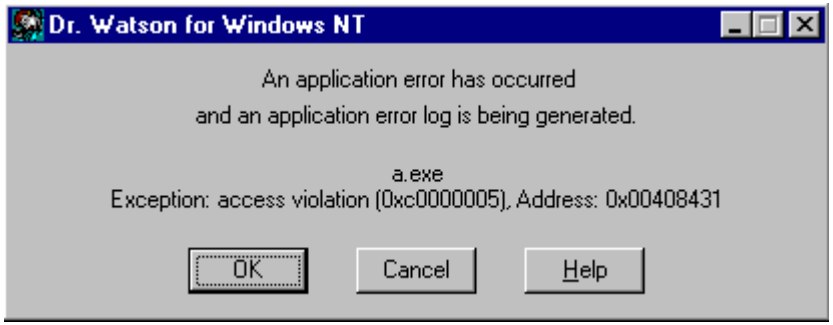
COMPARISON: “AND” INSTRUCTION EMULATION FOR ARM

```
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);
}
```

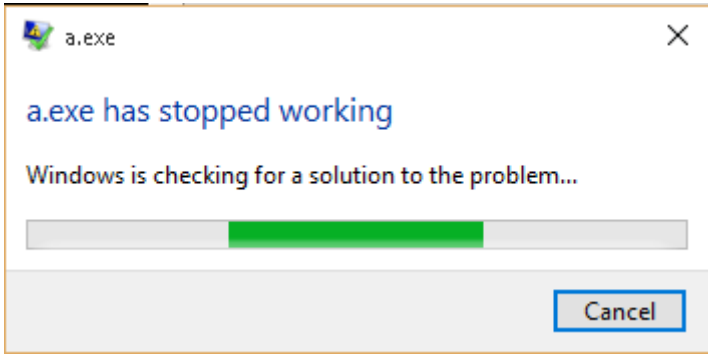
2. MEMORY PROTECTION WITH AN MMU - PHENOMENOLOGY

This happens when you execute `*(int*)NULL = 0xDEADBEEF;`

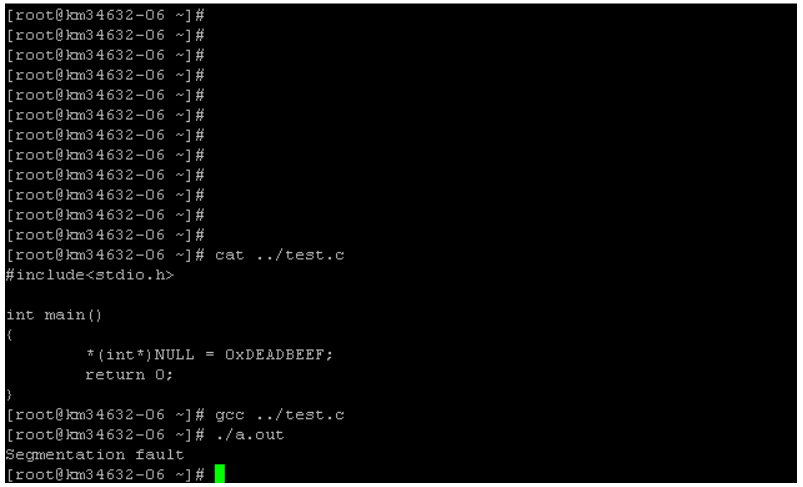
Windows 95 (bluescreen or freeze or something random)



Windows NT 4.0



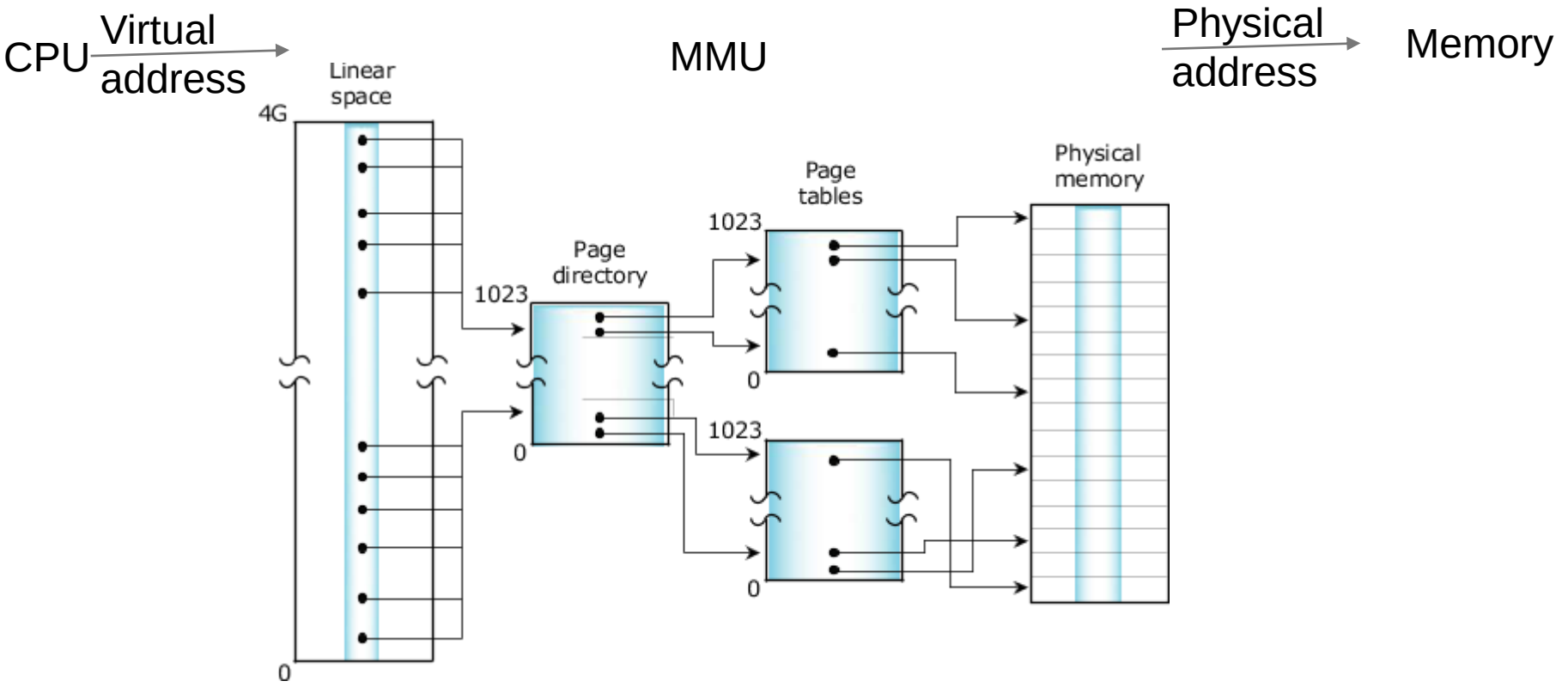
Windows 10



Linux

2. THE MEMORY MANAGEMENT UNIT

Run each memory access through a table



- Page size of 8kB. 4 byte per page
  - Total page table size: 2MB to address full 4GB
- *Read, Write*, *Execute* flags for each page
- In Linux each process gets its own page directory and page table

3. TIMER

32-Bit	Tick timer count register (TTCR)
4 bit control, 28-Bit comparison	Tick timer mode register (TTMR)

```
CPU.prototype.AdvanceTimer = function(cycles) {
    if ((TTMR>>>30) == 0) return; // check if timer is enabled
    delta = getDelta(TTCR, TTMR); // distance between timer and alarm
    TTCR += cycles; //advance timer

    if (delta < cycles) {
        if (TTMR&(1<<29)) // if interrupt enabled
            TTMR |= (1<<28); // set pending timer interrupt
    }
}

CPU.prototype.CheckTimer = function() {
    if ((SR_TEE) && (TTMR & (1<<28)))
        Exception(EXCEPT_TICK);
}
```

4. PROGRAMMABLE INTERRUPT CONTROLLER

32-Bit — PIC Status register, each bit enables one interrupt line

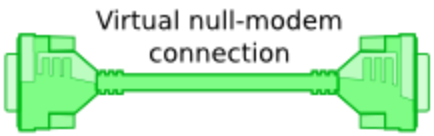
32-Bit — PIC Mask Register, each bit enables one interrupt line

```
CPU.prototype.RaiseInterrupt = function(line) {
    PICSR |= 1 << line;
    CheckForInterrupt();
}

CPU.prototype.ClearInterrupt = function(line) {
    PICSR &= ~(1 << line);
}

CPU.prototype.CheckInterrupt = function() {
    if (!SR_IEE) return; // check CPU interrupt enable flag
    if (PICMR & PICSR)  {  // compare interrupt mask and interrupt set
register
        Exception(EXCEPT_INT);
    }
}
```

5. TERMINAL SUPPORT AND SERIAL INTERFACE



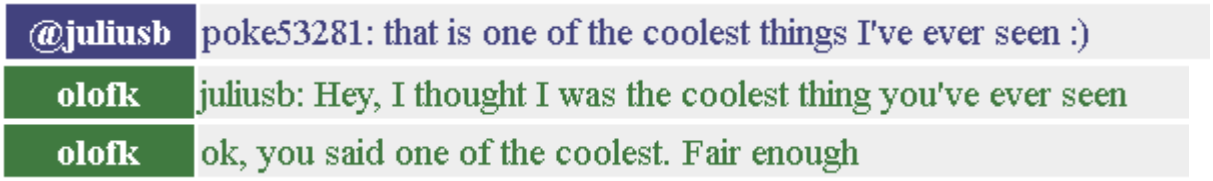
- Terminal output: 360 lines of code
- Terminal input: 90 lines code
- Serial controller:
  - Connected to an interrupt line
  - Memory mapped I/O
  - 220 lines code

WHAT YOU NEED ON THE SOFTWARE SIDE?

Archictecture dependent lines of code

- *binutils* 14207
  - assembler, disassembler, linker, elf-file format, application binary interface
- *GNU C Compiler* *6411*
- *Linux kernel* 6183
- *C library (e. g. musl)* *2108*
  - Interface between user space program and kernel and basic functionality for C programs
- *Busybox* 0
  - stripped down unix tools in a single executable including shell

The first version of jor1k had around 2000 lines of code!



Dec. 04th 2012: OpenRISC IRC Chat


SPEEDY
JAVASCRIPT

JAVASCRIPT

- **is very creative with type coercion**
  - `0 == "" => true`
  - `"" – "" => 0`
  - `if (new Array() == false) => true`
  - `{} + {} => NaN`
  - `{} + [] => 0`
  - `{} + {} + [] => "NaN"`
  - `("NaN") => "NaN"`
  - `({} + {} + []) => ("[object Object][object Object]")`
  - `0 > null => false`
  - `0 >= null => true`
  - `0 == null => false`
  - `[1,2,3]+[4,5,6] => “1,2,34,5,6“`

- **is considered “slow”**
  - At least four companies are writing optimized compilers to squeeze out the maximum performance.

ALL NUMBERS ARE DOUBLE

JavaScript doesn't know about integers, only doubles

- `y = 9999999999999999 => y = 10000000000000000`
  (double) (double)

But there are logical operations which act on 32-Bit

- `y = 1.1234|2  => y = 3`
- `y = 0xFFFFFFFF|0  => y = -1`
- `y = -1 >>> 0  => y = 0xFFFFFFFF`

The JavaScript-engines optimize for accuracy

- `y = 0x7FFFFFFF` (treated internal as integer)

There are also 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_Interrupt()
    ppc = Translate_V
    ins = ram.Read32(
    vpc += 4;  // adva
    // decode instruct
    switch (ins&0x7F)
        ....
    }
}
```

What happens internal?

1. Add 4 to vpc (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
    ins = ram.Read32(ppc);
    vpc = (vpc + 4)|0;
    // decode instruction
    switch (ins&0x7F) {
        ....
    }
}
```

What happens internal?

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

HOW TO PREVENT DEOPTIMIZATIONS?

Add more typing helpers

```
for(;;) {
  Advance_Timer();
    Check_Interrupt();
    ppc = Translate_Virtual_To_Physical(vpc|0)|0;
    ins = ram.Read32(ppc|0)|0;
    vpc = (vpc + 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. Hence the last operation for the index must be “>> `2`”

Project Emscripten allows to translate C++ to asm.js JavaScript

"asm.js requires a style of coding only compilers can output. *A person writing actual asm.js code by hand would need to be insane* as asm.js code required style of coding is horribly disorganized"[1]

"Unfortunately asm.js requires one giant array to put things on. *No one in their right mind codes like that by hand.*"[2]

From Grant Galitz (JavaScript GameBoy Advance Emulator)

▲ binarymax 1093 days ago [-]

Very impressive work. I saw Brendan Eich speak At jquery uk in April and he said asm.js was meant for compilers and people should not be programming it directly...I couldn't help but smile and think 'oh yeah?'

On Hacker News Sep 17th 2013 regarding jor1k

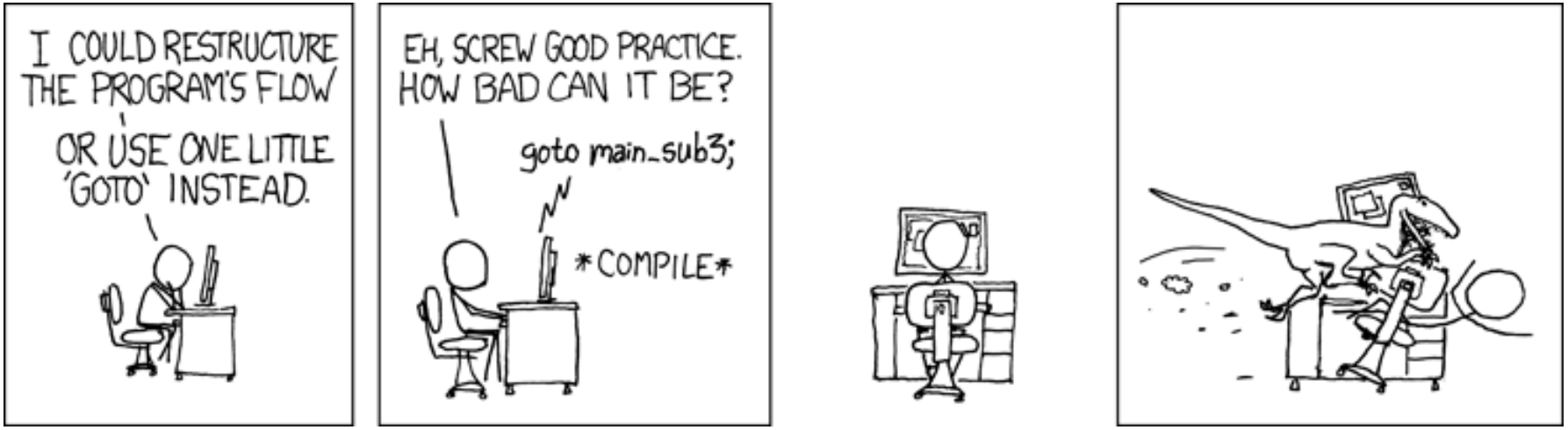
▲ Sephr 493 days ago [-]

That's quite an impressive implementation! I didn't realize hand-written asm.js could be that readable.

On Hacker News May 10th 2015 regarding jor1k

FURTHER OPTIMIZATIONS

Reducing the code per EMULATED instruction which has to be executed



THE MMU: SOFTWARE TLB LOCKUP

Implemented in two stages

1. Full translation table in memory

2. TLB variables (tlb buffer with one entry)

Translation fastpath of virtual to physical addresses:

```
if ((tlb_check ^ virtual_addr) >> 12)
{
    ...
    tlb_check = ...
    tlb_trans = ...
}
physical_addr = tlb_trans ^ virtual_addr;
```

FENCE TECHNIQUE

For every instruction the program counter (pc) must be translated to the physical pc. However, the pc advances usually by 4 and is not leaving the current page. This property can be used by the fence technique

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();
      ......
    }
  }
```

Jump unlikely

- 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
OPTIMIZATION

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
		jae .Lfrom133002
		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 load

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`
		`cmpl %ecx, 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`

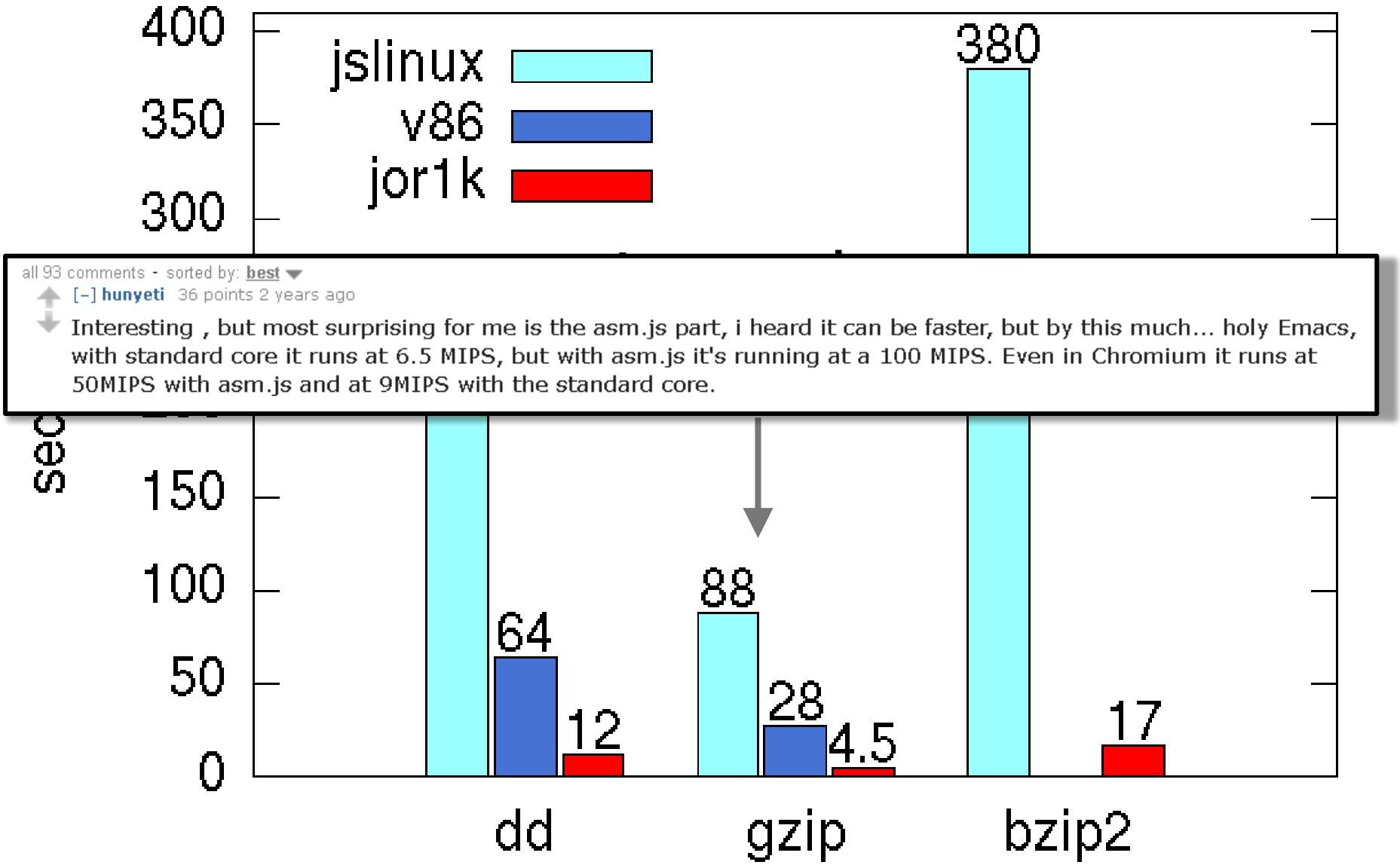
	AsmJSLoadHeap	`cmpl $0xfffffffc, %ecx`
		`ja .Lfrom133022`
		`movl 0x0000(%ecx), %ecx`

	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

BENCHMARK IN 2014



all 93 comments - sorted by: **best**

[–] **hunyeti** 36 points 2 years ago

Interesting , but most surprising for me is the asm.js part, i heard it can be faster, but by this much... holy Emacs, with standard core it runs at 6.5 MIPS, but with asm.js it's running at a 100 MIPS. Even in Chromium it runs at 50MIPS with asm.js and at 9MIPS with the standard core.

Email from Sep 2015 from Fabrice Bellard (QEMU maintainer)

Von Fabrice Bellard <[redacted]>

Betreff **jslinux benchmark**

An Mich <sebastian@macke.de>

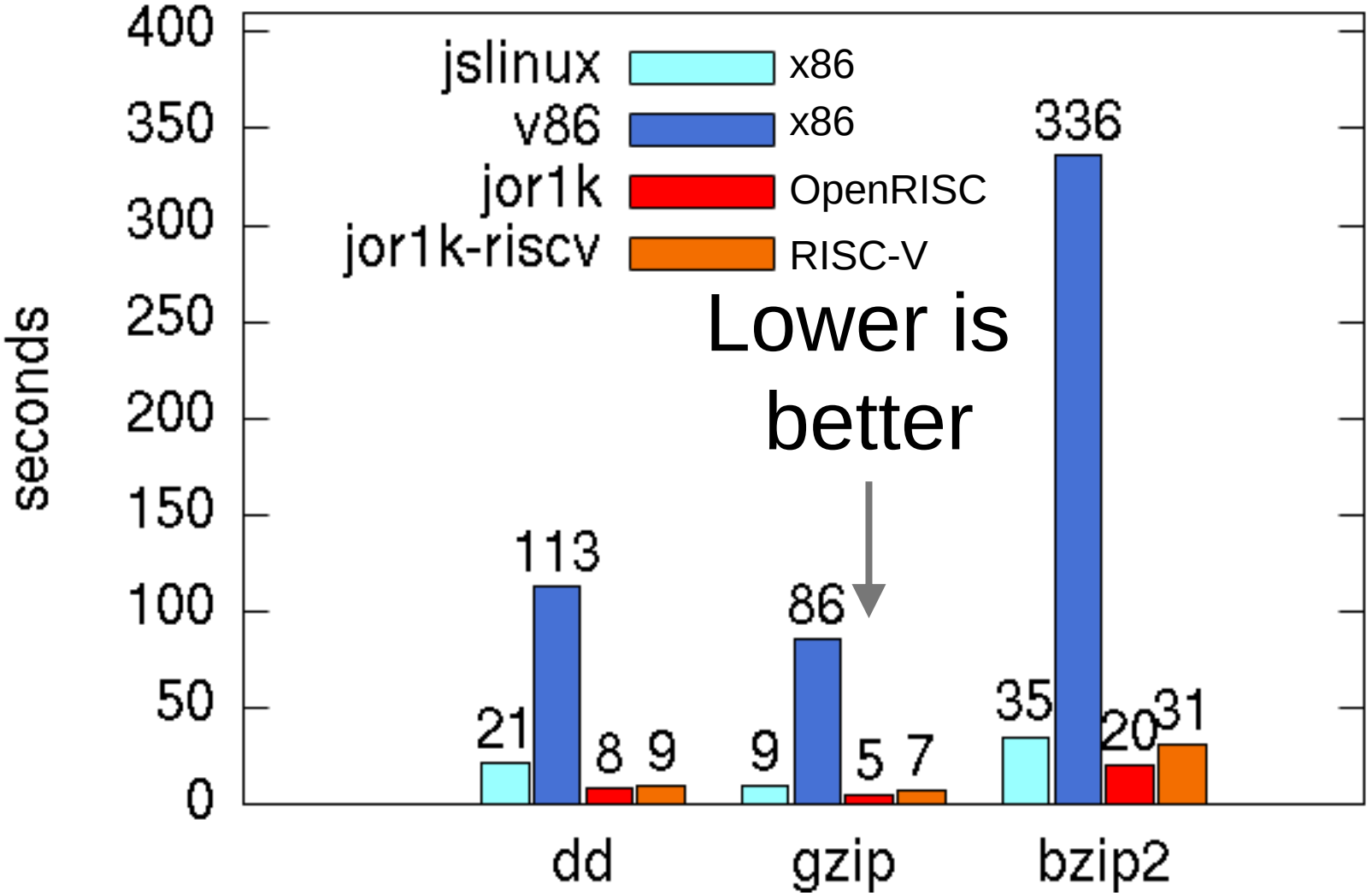
Hi Sebastian,

I spent some time converting jslinux to asm.js. It is still slower than your excellent jor1k but your benchmark at https://github.com/s-macke/jor1k/wiki/Benchmark-with-other-emulators needs to be revised 🙂

Best regards,

Fabrice.

BENCHMARK FROM SEP 2015



BENCHMARK FROM SEP 2015

- 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

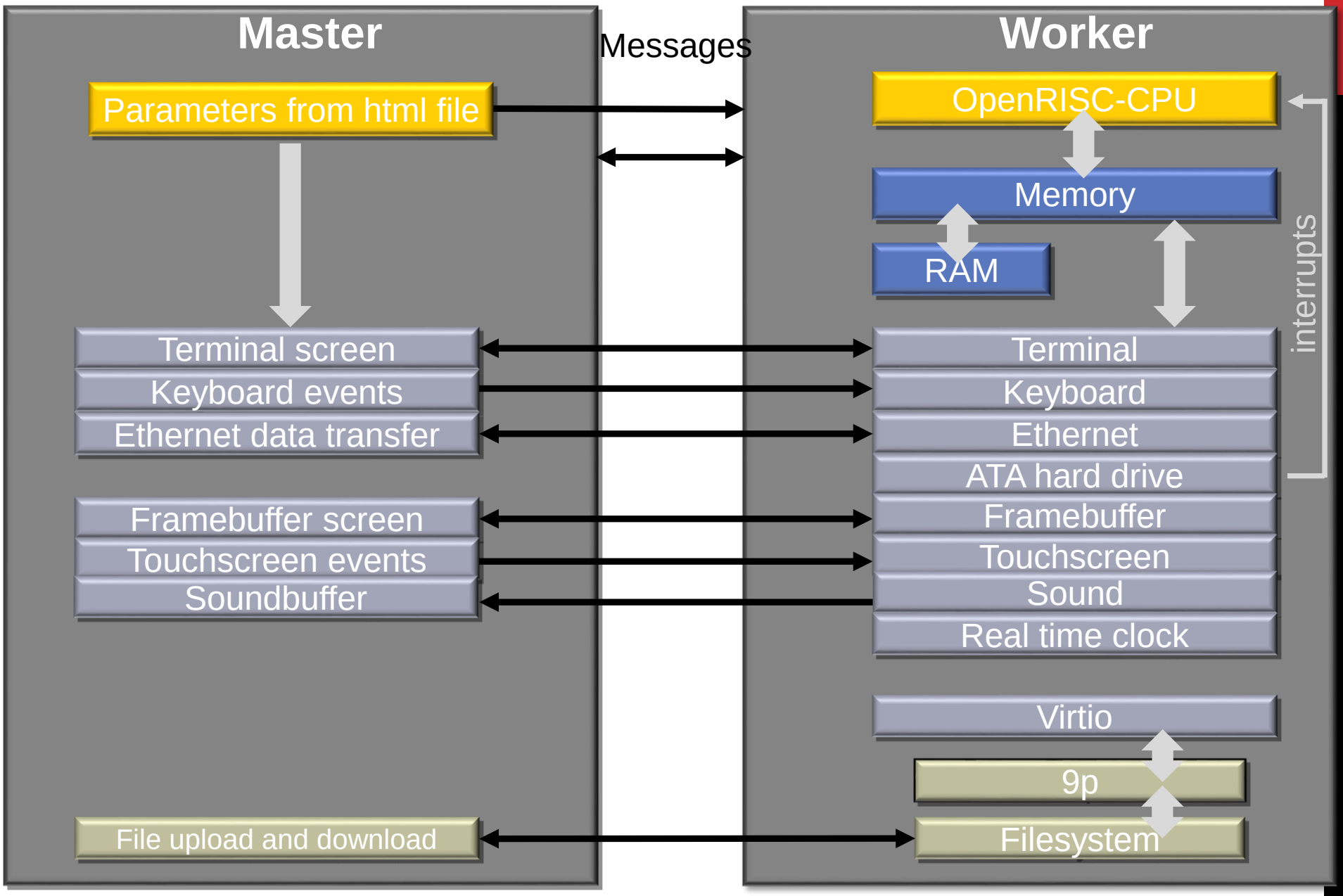
Approx. speed of the CPUs of the year 1997

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

dd gzip bzip2

MODULARIZATION AND PARALLELIZATION

SEPARATE INTO TWO THREADS



MULTIPLE CORES

You need additionally:

- **Core ID (consecutive number)**
- **Software interrupt: one core can send interrupt to another core**
- **One global timer, but separate alarms for each core.**
- **New synchronization instructions**
  - Load-Link/Store Conditional (read-modify write operation, atomic)

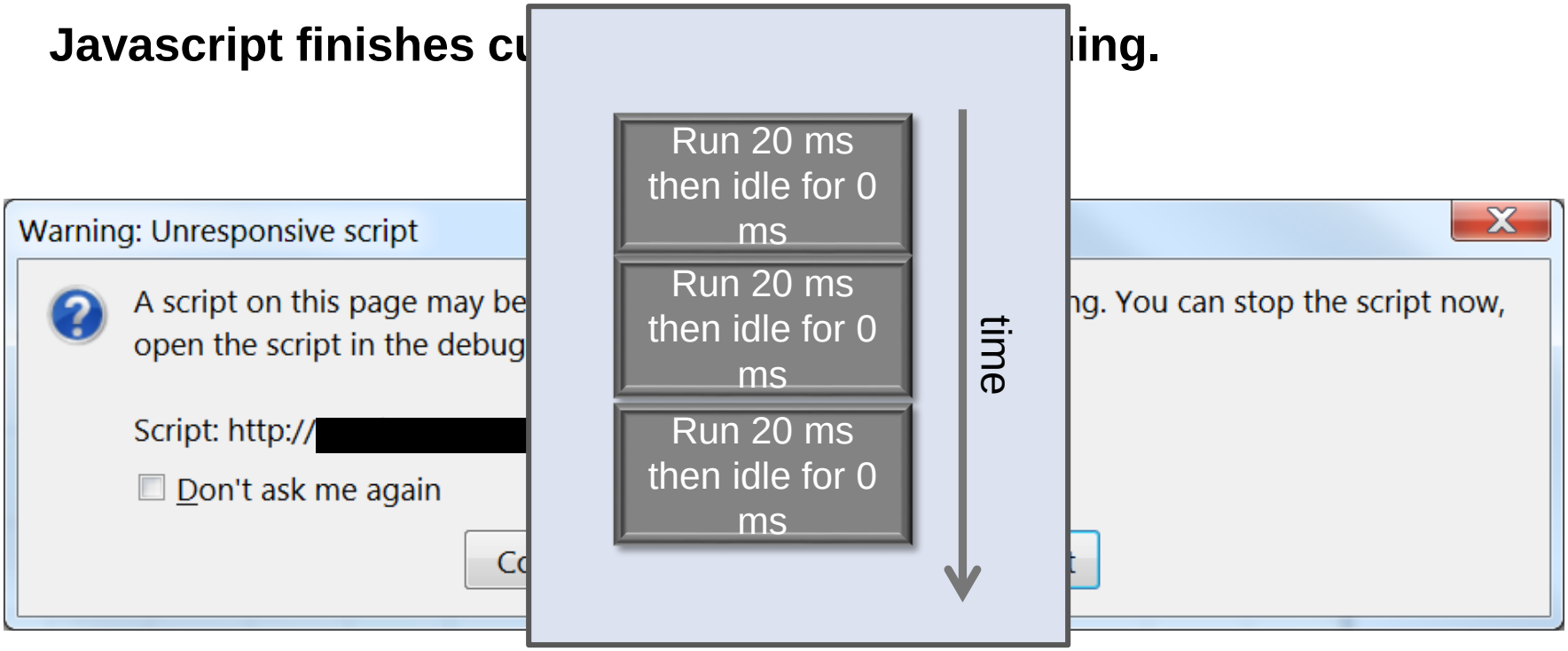
Currently implemented (buggy) in one thread with time shaping

Mozilla announced SharedArrayBuffer and atomics for Javascript to allow real multithreading

HOW TO NOT IDLE IN JAVASCRIPT?

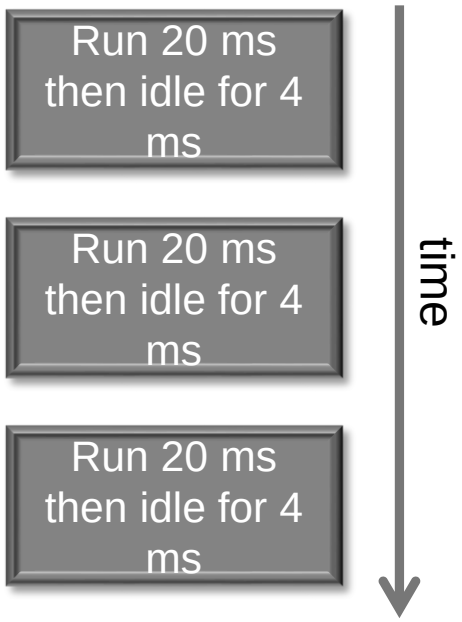
HOW TO STAY 100‘% BUSY IN JAVASCRIPT BUT REMAIN RESPONSIVE?

Javascript finishes cu ing.



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)**



WHAT ABOUT THE WORKER THREAD?

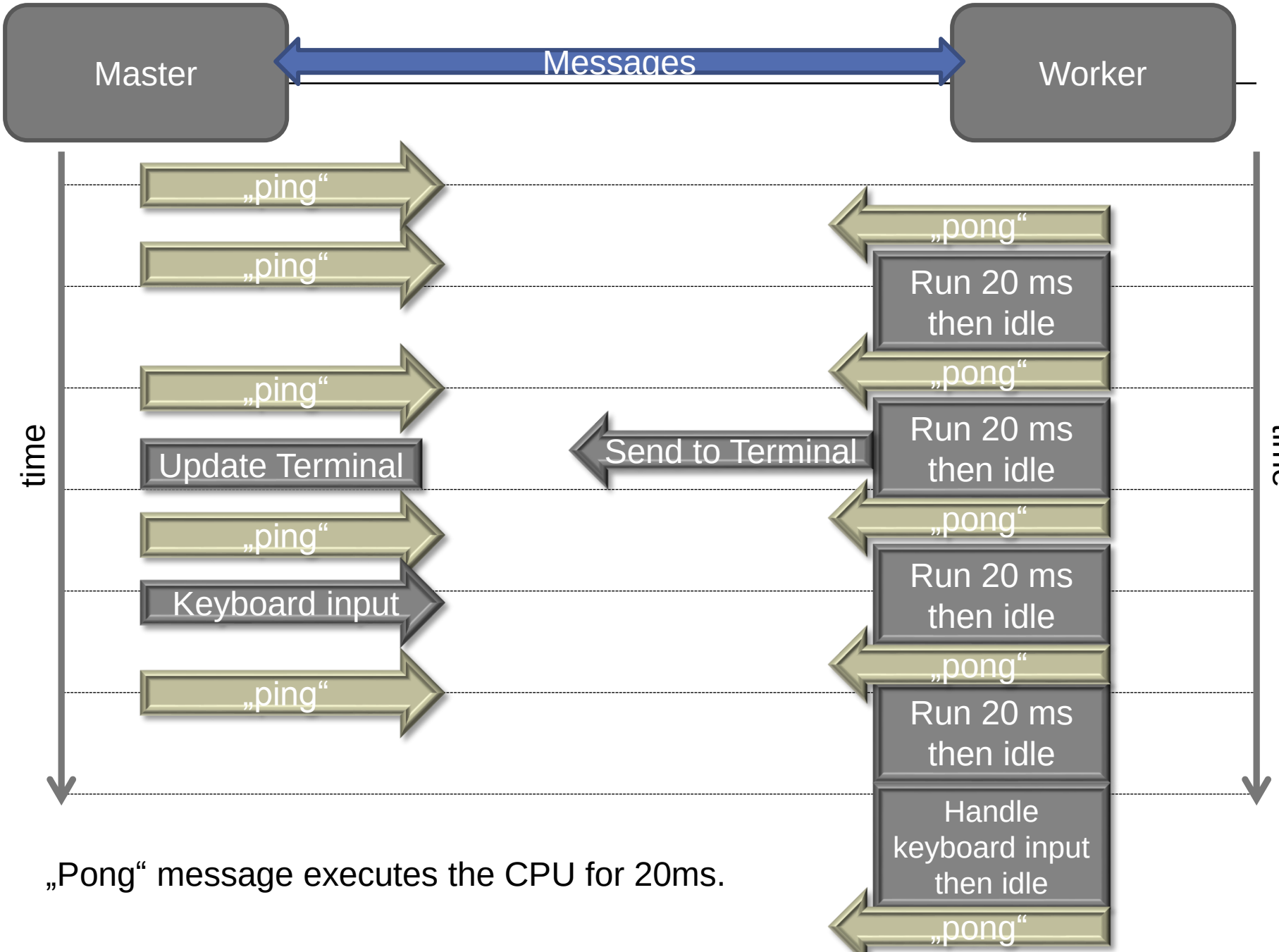
Can we get the number of messages in the queue or have non-blocking access?

NOPE!

SOLUTION

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 time:
The horror of exact timing in JavaScript

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

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

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

The Filesystem

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

THE FILESYSTEM

How long does it take to get the size of a directory structure with 10000 files 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 as “file system as hard ware device” 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"}
    ]},
```

Load compressed files on demand.

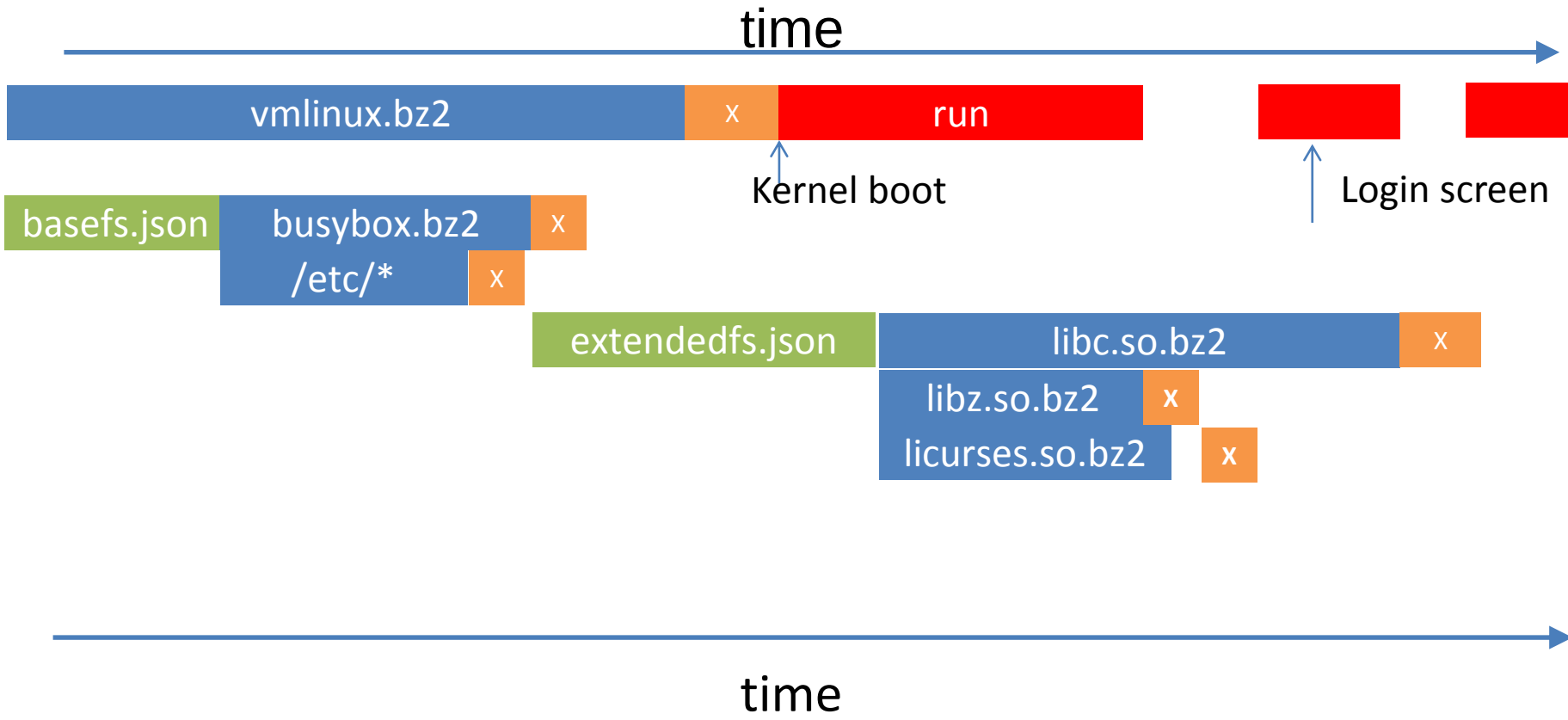
- OpenRISC binaries compress really well
- .bz2 currently, in future .xz
- Ordinary web server needed

Future: dependencies between files, packages

- http 2.0 will help heres

reddit on jor1k

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

IS THE EMULATOR USEFUL?

Well, sort of ...

- Technology demonstration, Advertisement
- Interactive online 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

FUTURE

- Full virtio driver support (done)
- Virtio-GPU and full-screen X-Window system
- More terminals, better user interface
- Download already booted Linux (state file)
- Status, statistics and debug screen
- Run Java
- Run Firefox

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**


Play Monkey Island
MONKEY ISLAND


Develop in C
// Compile this program to find the syntax
#include <stdio.h>
int main() {
printf("Hello World!\n")
return 0;
}
Compile and run


Play Elite II
Mouse
Actual:0.0 kmhr⁻¹
Set:0.0 kmhr⁻¹
Alt:0 m
Relative to:Merlin
12:00:03 1-Jan-3200
Landed
External View


Play DOOM
47
AMMO
100%
HEALTH
ARMS


glxgears
to open the menu
X Window system available
xeyes


Dillo: Google
http://www.google.com/
Back
Home
Reload
Save
Stop
Book
Tools
Images 0 of 1
Page 17.5 KB
Search Images Maps Play YouTube News Gmail Drive More »
Web History | Settings | Sign in
Google
Advanced search
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