THE JOY AND EXCITEMENT OF FORBIDDEN COMPUTING

kate temkin, !!con 2021

SO, WHO AM ‘I’?

Katherine/Kate Temkin (@ktemkin):

- easily nerd-sniped
  (by problems she shouldn’t solve)
- lead digital witch at Great Scott Gadgets
- open-source-tool-builder
- educational (reverse) engineer
- one human person, within a margin of error

PROPOSAL:
GOOD COMPUTING IS HUMAN-FOCUSED

TODAY'S EXAMPLE:
JOY FROM NON-FORBIDDEN COMPUTING

UTM

Run virtual machines on iOS

UTM: virtual machines for iOS

a speedy QEMU-based full-machine emulator for Apple devices

source: getutm.app

UTM

Run virtual machines on iOS

UTM: virtual machines for iOS

a speedy QEMU-based full-machine emulator for Apple devices running older versions of iOS

source: getutm.app

UTM
@UTMapp
Unknown

Do not update to iOS 14.4 if you wish to continue using UTM. There will not be any workarounds. Future releases of UTM for iOS will focus on jailbroken devices unless Apple's App Store policy changes. UTM for AltStore will continue to be supported for < iOS 14.4.

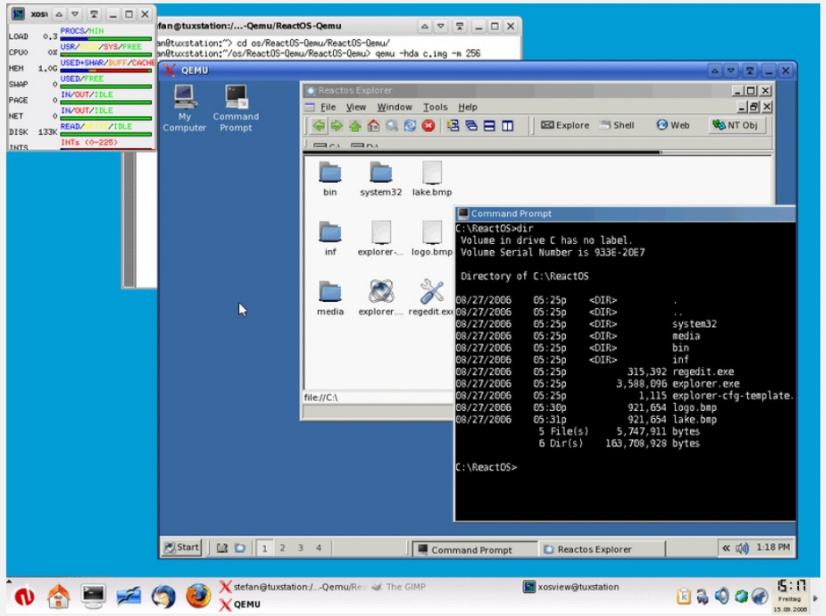
11:23 AM · Jan 26, 2021 · Tweetbot for iOS

STATUS: COMPUTATIONALLY FORBIDDEN

SO:
HOW DID THIS HAPPEN?

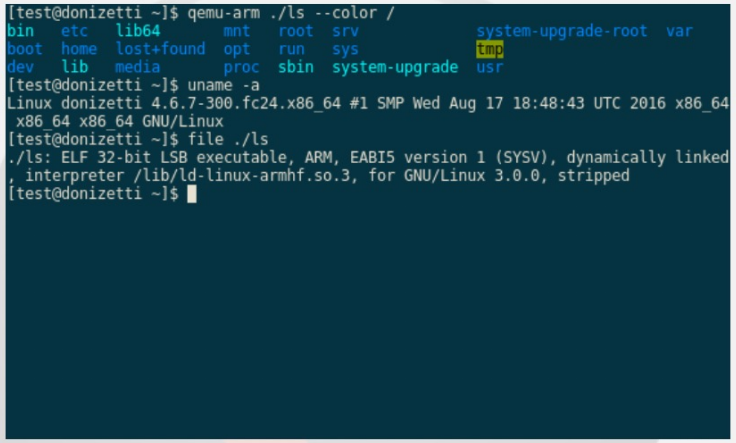
What is QEMU?

QEMU is a generic and open source machine emulator and virtualizer.



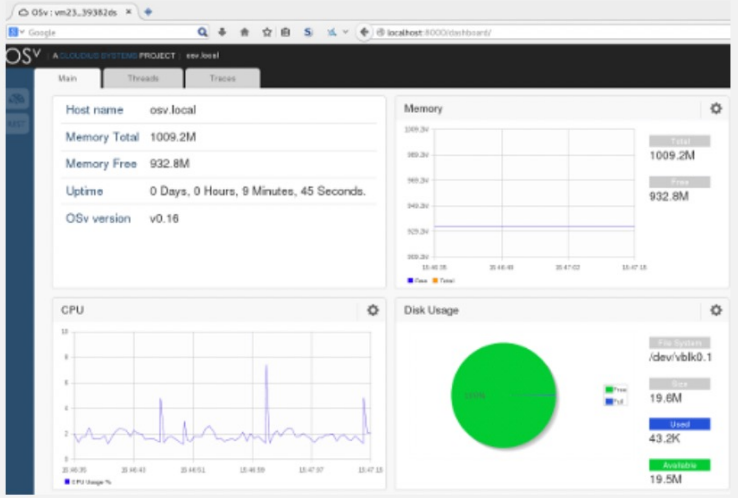
Full-system emulation

Run operating systems for any machine, on any supported architecture



User-mode emulation

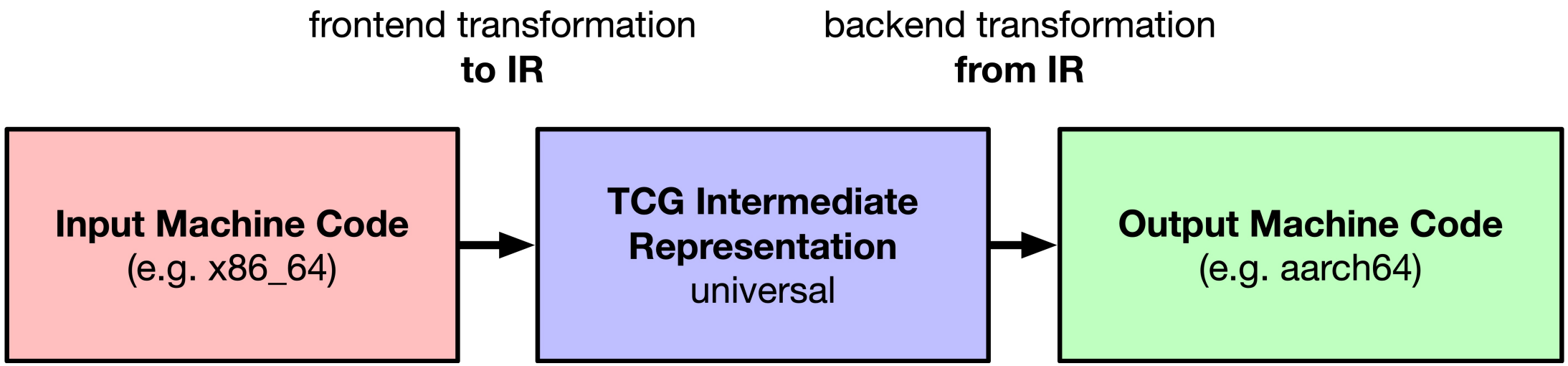
Run programs for another Linux/BSD target, on any supported architecture

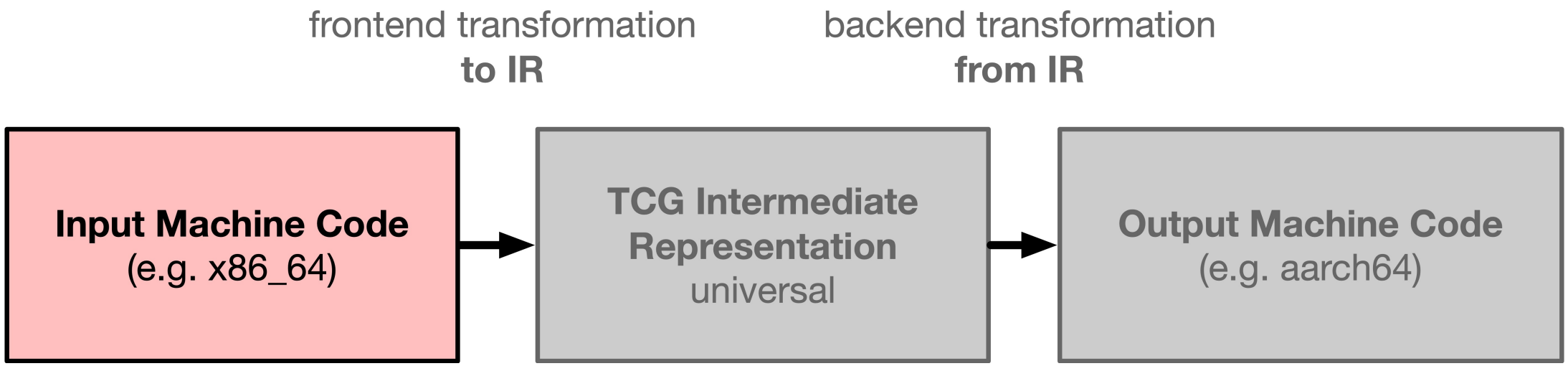


Virtualization

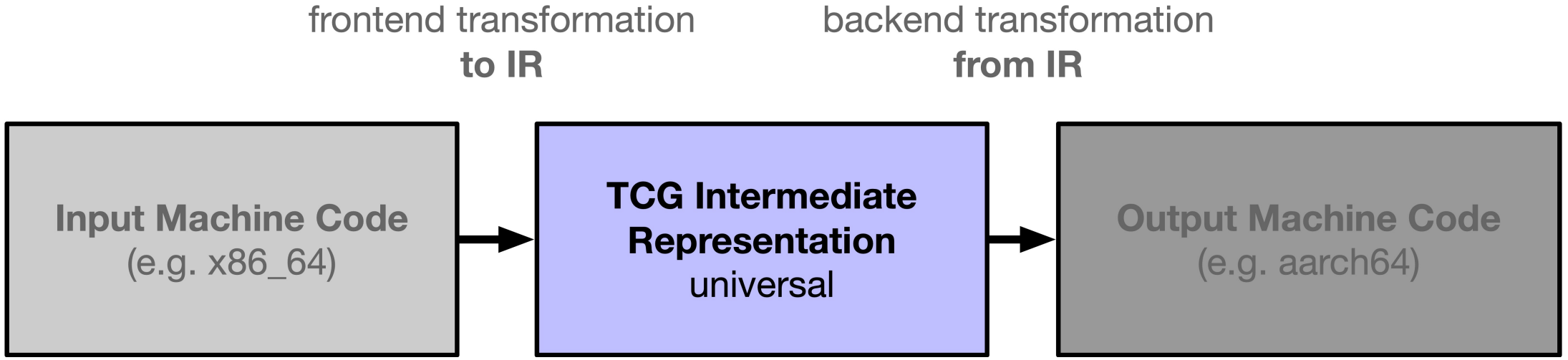
Run KVM and Xen virtual machines with near native performance

source: qemu.org


frontend transformation
to IR
backend transformation
from IR
Input Machine Code
(e.g. x86_64)
TCG Intermediate Representation
universal
Output Machine Code
(e.g. aarch64)




```
xorl    %r8d, %r8d
movl    $0x40000, %ecx
movl    $0x100, %edx
movl    $0x100, %esi
movq    $18446744071850834489, %rdi
callq   0xffffffff90256d10
```




```
xorl    %r8d, %r8d                          mov_i64 r8,$0x0
movl    $0x40000, %ecx                      mov_i64 rcx,$0x40000
movl    $0x100, %edx                        mov_i64 rdx,$0x100
movl    $0x100, %esi                        mov_i64 rsi,$0x100
movq    $18446744071850834489, %rdi         mov_i64 rdi,$0xffffffff91363a39
callq   0xffffffff90256d10                  st_i64 $0xffffffff90256d10,env,$0x80
                                            call lookup_tb_ptr,$0x6,$1,tmp21,env
                                            goto_ptr tmp21
```

frontend transformation
to IR

backend transformation
from IR


```
mov_i64 r8,$0x0
mov_i64 rcx,$0x40000
mov_i64 rdx,$0x100
mov_i64 rsi,$0x100
mov_i64 rdi,$0xffffffff91363a39
```

```
str     xzr, [x19, #0x40]
orr     w20, wzr, #0x40000
str     x20, [x19, #8]
mov     w20, #0x100
str     x20, [x19, #0x10]
mov     w20, #0x100
str     x20, [x19, #0x30]
mov     x20, #-0xc5c7
movk    x20, #0x9136, lsl #16
str     x20, [x19, #0x38]
```

frontend transformation
to IR

backend transformation
from IR


```
xorl    %r8d, %r8d
movl    $0x40000, %ecx
movl    $0x100, %edx
movl    $0x100, %esi
movq    $18446744071850834489, %rdi
```

```
str     xzr, [x19, #0x40]
orr     w20, wzr, #0x40000
str     x20, [x19, #8]
mov     w20, #0x100
str     x20, [x19, #0x10]
mov     w20, #0x100
str     x20, [x19, #0x30]
mov     x20, #-0xc5c7
movk    x20, #0x9136, lsl #16
str     x20, [x19, #0x38]
```

THE MAGIC?
JUST IN TIME GENERATION!

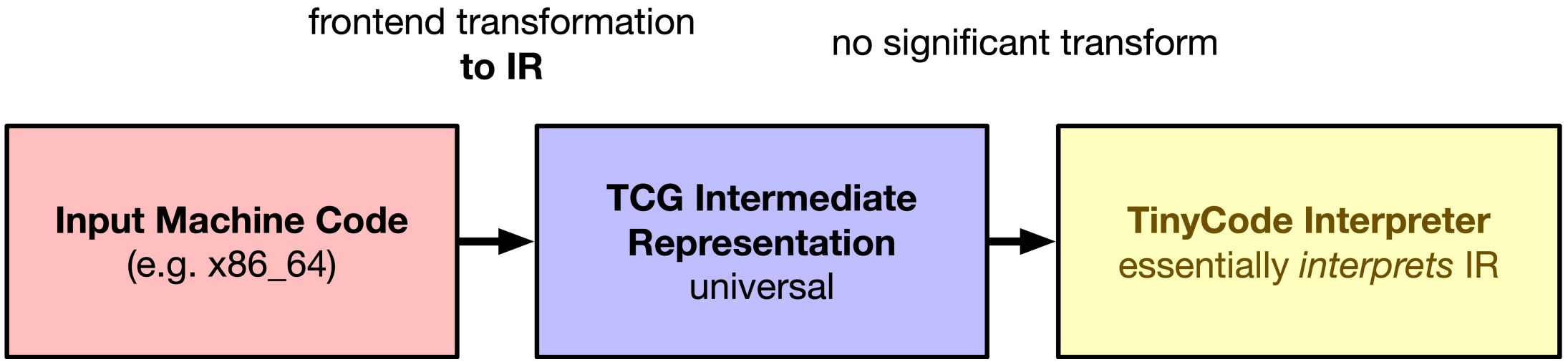
OKAY,
NOW WHAT ABOUT ON MY PHONE?

UNFORTUNATELY,
APPLE REQUIRES ALL CODE TO BE SIGNED BY THE DEVELOPER.

```
xorl    %r8d, %r8d
movl    $0x40000, %ecx
movl    $0x100, %edx
movl    $0x100, %esi
movq    $18446744071850834489, %rdi
```

```
str     xzr, [x19, #0x40]
orr     w20, wzr, #0x40000
str     x20, [x19, #8]
mov     w20, #0x100
str     x20, [x19, #0x10]
mov     w20, #0x100
str     x20, [x19, #0x30]
mov     x20, #-0xc5c7
movk    x20, #0x9136, lsl #16
str     x20, [x19, #0x38]
```

```
jit.psuedocode:0: woahthere: that's not signed!
i can't let you run this! JIT is ~forbidden~!
```




```
xorl    %r8d, %r8d
movl    $0x40000, %ecx
movl    $0x100, %edx
movl    $0x100, %esi
movq    $18446744071850834489, %rdi
```

ONE OPTION:
RUNNING A VM ON A VM

```
case INDEX_op_st_i32:
CASE_64(st32)
    tci_args_rrs(&tb_ptr, &r0, &r1, &ofs);
    ptr = (void *)(regs[r1] + ofs);
    *(uint32_t *)ptr = regs[r0];
    break;

    /* Arithmetic operations (mixed 32/64 bit). */

CASE_32_64(add)
    tci_args_rrr(&tb_ptr, &r0, &r1, &r2);
    regs[r0] = regs[r1] + regs[r2];
    break;
CASE_32_64(sub)
    tci_args_rrr(&tb_ptr, &r0, &r1, &r2);
    regs[r0] = regs[r1] - regs[r2];
    break;
CASE_32_64(mul)
    tci_args_rrr(&tb_ptr, &r0, &r1, &r2);
    regs[r0] = regs[r1] * regs[r2];
    break;
CASE_32_64(and)
```

THE ISSUE?
TCI IS SLOW


5:16

WE CAN'T RUN GENERATED CODE.
WE CAN RUN PRE-MADE CODE.

WE CAN'T RUN GENERATED CODE.

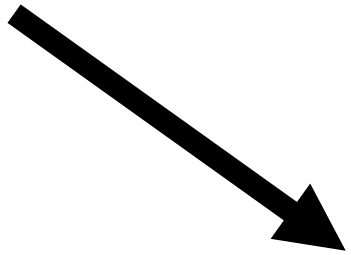
WE CAN RUN PRE-MADE CODE.

WE CAN RUN PRE-MADE CODE.

WE CAN RUN PRE-MADE CODE.

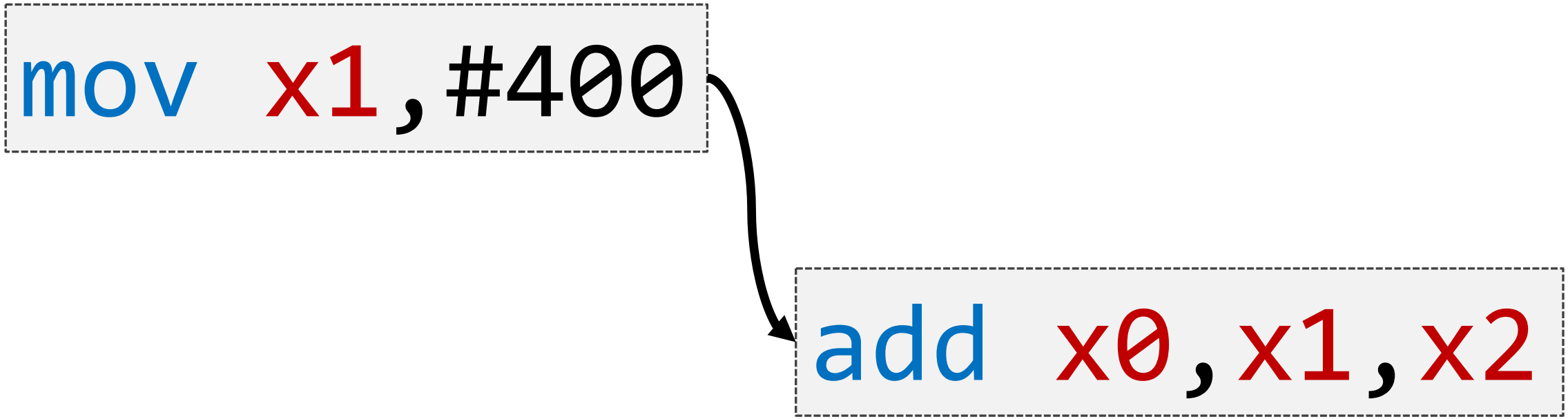
...SO, WHAT IF WE PRE-GENERATE
ALL THE CODE WE MIGHT NEED?

```
mov_i64 r0,r1
```



```
mov x0,x1
```

```
mov_i64 x1,#400
add_i64 x0, x1, x2
```



“program list”
of gadgets

```
0xffaa0000
0xffab0024
```

0xffaa0000

```
mov x0,x1
```

0xffaa0024

```
add x0,x1,x2
```

“program list”
of gadgets

```
0xffaa0000
0xffab0024
```

0xffaa0000

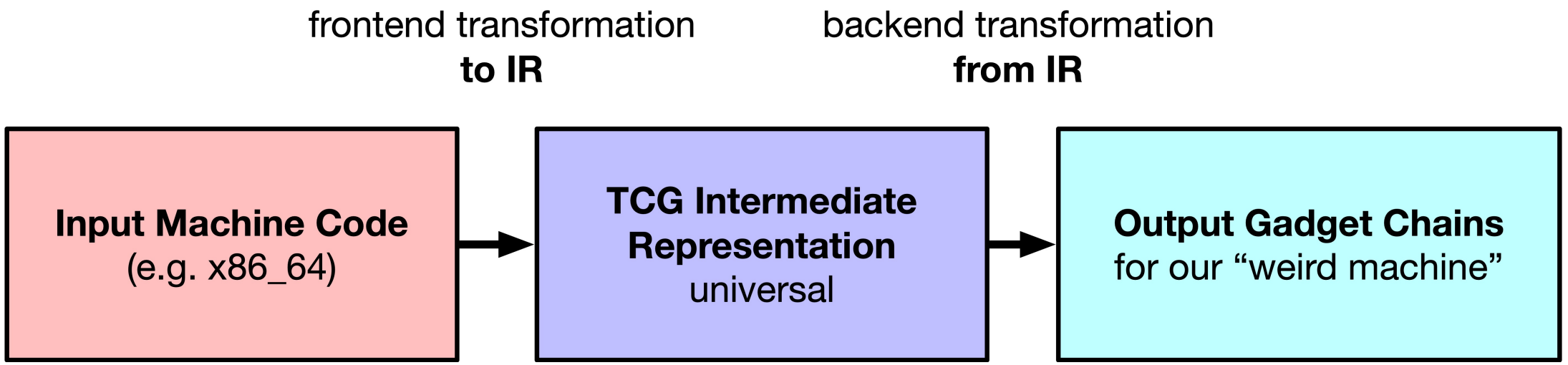
```
mov x0,x1
ldr x27,[x28],#8
br x27
```

0xffaa0024

```
add x0,x1,x2
ldr x27,[x28],#8
br x27
```

```
# This optimization helps costly reads from memories for simple operations.
with_d_immediate("movi_imm_i32", "mov Wd, #Ii", immediate_range=range(64))
with_d_immediate("movi_imm_i64", "mov Xd, #Ii", immediate_range=range(64))

START_COLLECTION("load_unsigned")

# LOAD variants.
# TODO: should the signed variants have X variants for _i64?
ldst_dn("ld8u",      "ldrb  Wd, [Xn, x27]")
ldst_dn("ld16u",     "ldrh  Wd, [Xn, x27]")
ldst_dn("ld32u",     "ldr   Wd, [Xn, x27]")
ldst_dn("ld_i64",    "ldr   Xd, [Xn, x27]")

START_COLLECTION("load_signed")

ldst_dn("ld8s_i32",  "ldrsb Wd, [Xn, x27]")
```

```
1310
1311    static __attribute__((naked)) void gadget_add_i32_arg0_arg9_arg1(void)
1312    {
1313        asm(
1314            "add w0, w9, w1 \n"
1315            "ldr x27, [x28], #8 \n"
1316            "br x27 \n"
1317        );
1318    }
1319
1320    static __attribute__((naked)) void gadget_add_i32_arg0_arg9_arg2(void)
1321    {
1322        asm(
1323            "add w0, w9, w2 \n"
1324            "ldr x27, [x28], #8 \n"
1325            "br x27 \n"
1326        );
```


frontend transformation
to IR
backend transformation
from IR
Input Machine Code
(e.g. x86_64)
TCG Intermediate Representation
universal
Output Gadget Chains
for our “weird machine”

```
case INDEX_op_bswap16_i32:  /* Optional (TCG_TARGET_HAS_bswap16_i32). */
case INDEX_op_bswap16_i64:  /* Optional (TCG_TARGET_HAS_bswap16_i64). */
    tcg_out_binary_gadget(s, gadget_bswap16, args[0], args[1]);
    break;

case INDEX_op_bswap32_i32:  /* Optional (TCG_TARGET_HAS_bswap32_i32). */
case INDEX_op_bswap32_i64:  /* Optional (TCG_TARGET_HAS_bswap32_i64). */
    tcg_out_binary_gadget(s, gadget_bswap32, args[0], args[1]);
    break;

case INDEX_op_bswap64_i64:  /* Optional (TCG_TARGET_HAS_bswap64_i64). */
    tcg_out_binary_gadget(s, gadget_bswap64, args[0], args[1]);
    break;

case INDEX_op_not_i64:      /* Optional (TCG_TARGET_HAS_not_i64). */
    tcg_out_binary_gadget(s, gadget_not_i64, args[0], args[1]);
    break;
```

frontend transformation
to IR

backend transformation
from IR

`mov_i32 cc_op,$0x10`	`gadget_movi_imm_i32_arg9_arg16`
	`gadget_st_i32_sh8_imm_arg9_arg14_arg21`
	`gadget_movi_imm_i64_arg9_arg0`

`brcond_i64 cc_dst,$0x0,ne,$L1`	`gadget_brcond_i64_ne_arg8_arg9` `00000002800003d0`
`goto_tb $0x0`	`gadget_br` `0000000000000000`

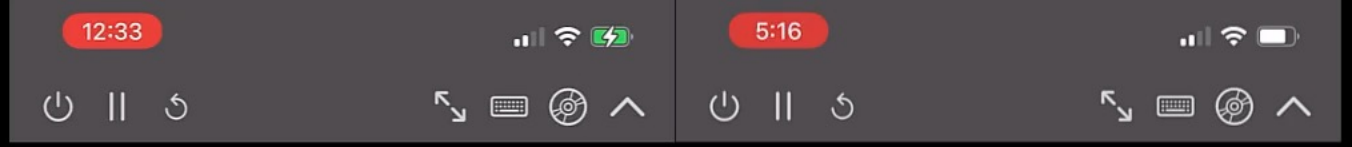
`st_i64 $0xe066,env,$0x80`	`gadget_movi_i64_arg8`
	`000000000000e066`
	`gadget_st_i64_sh8_imm_arg8_arg14_arg16`

`exit_tb $0x280000200`	`gadget_exit_tb` `0000000280000200`

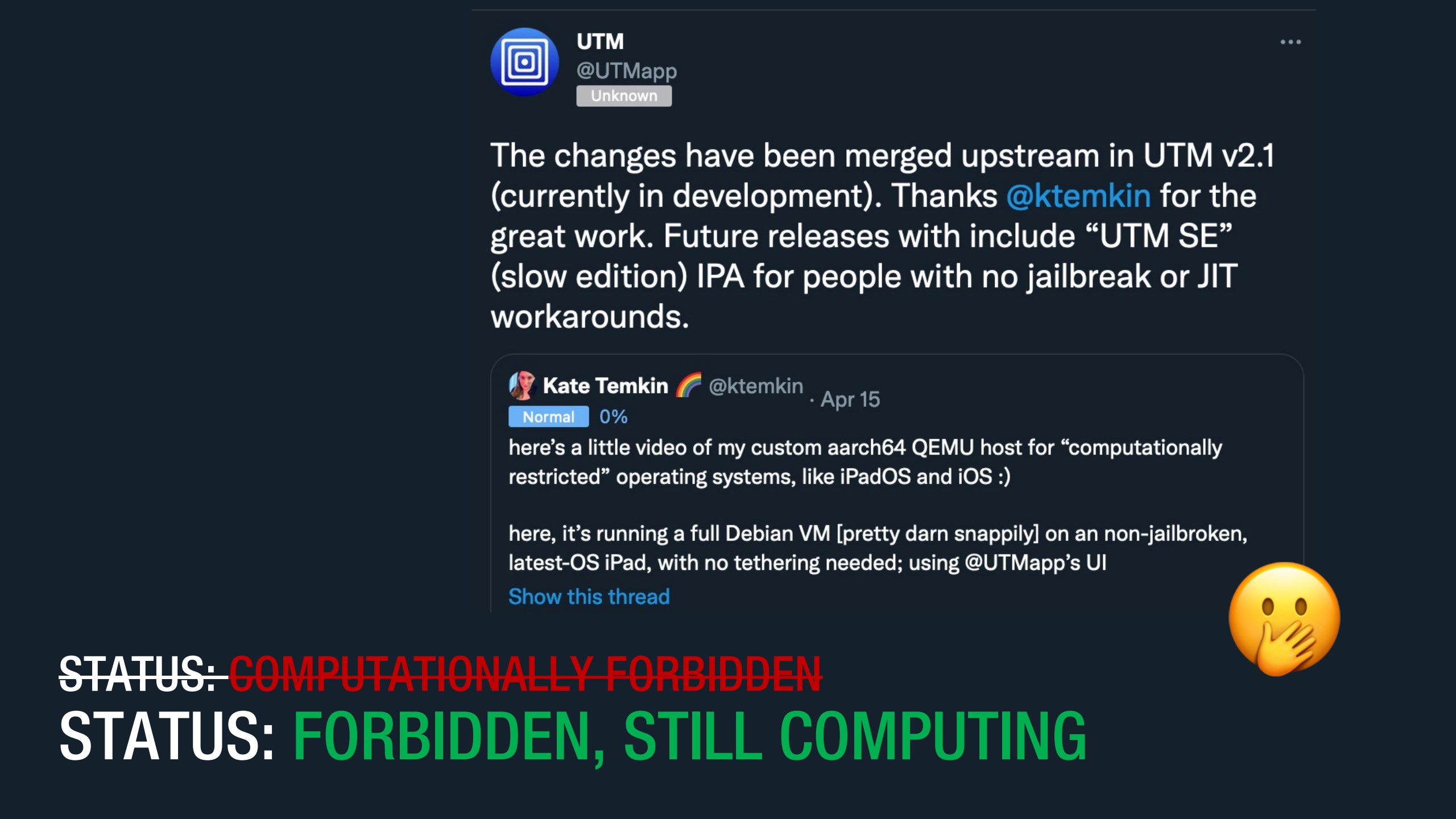
frontend transformation
to IR

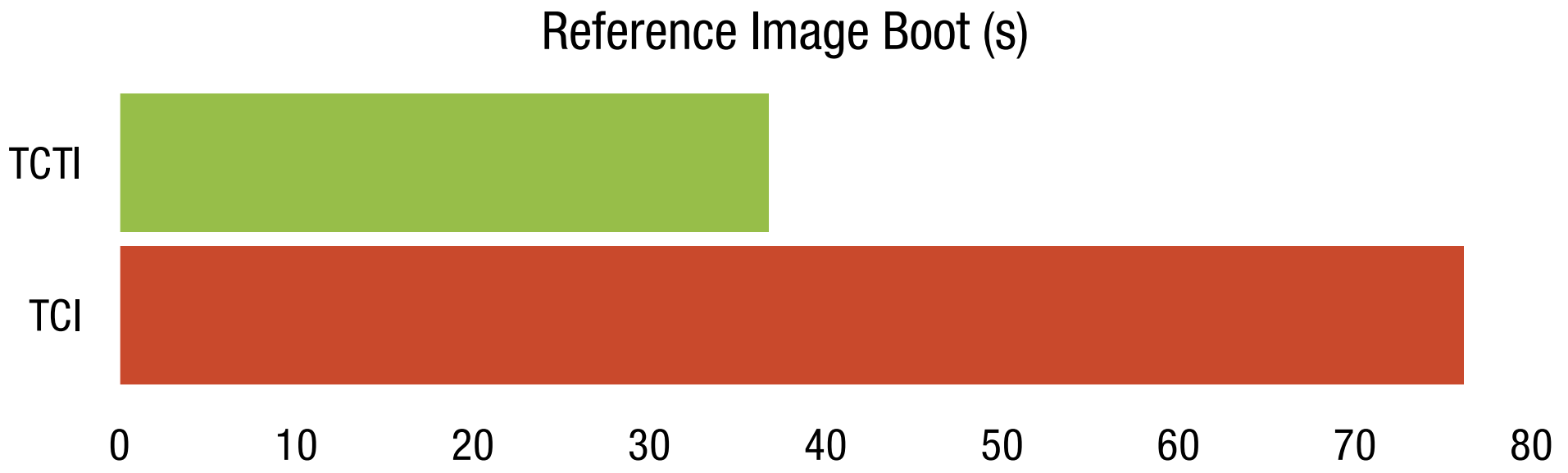
backend transformation
from IR

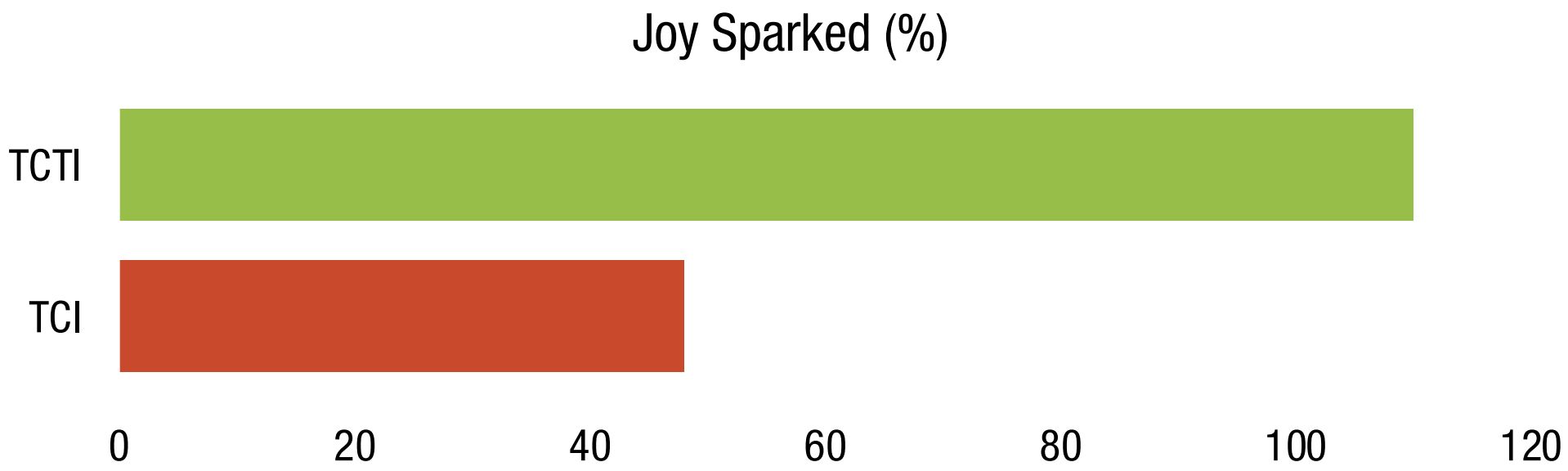

```
gadget_movi_imm_i32_arg9_arg16                  mov w9, #16
gadget_st_i32_sh8_imm_arg9_arg14_arg21          st w9, [x14, #21]
gadget_movi_imm_i64_arg9_arg0                   mov x9, #0
-------------------------------------------------------------------
gadget_brcond_i64_ne_arg8_arg9                  cmp x8, w9
0000002800003d0                                 bne 0x02800003d0
-------------------------------------------------------------------
gadget_br                                       b <target address after relocation*>
0000000000000000
-------------------------------------------------------------------
gadget_movi_i64_arg8                            ldr x8, =#0xe066
000000000000e066
gadget_st_i64_sh8_imm_arg8_arg14_arg16          st x8 [x14, #16]
-------------------------------------------------------------------
gadget_exit_tb                                  ldr x0, =0x280000200
0000002800000200                                ret
```



TCI TCTI


UTM
@UTMapp
Unknown
The changes have been merged upstream in UTM v2.1 (currently in development). Thanks @ktemkin for the great work. Future releases with include “UTM SE” (slow edition) IPA for people with no jailbreak or JIT workarounds.
Kate Temkin @ktemkin · Apr 15
Normal 0%
here’s a little video of my custom aarch64 QEMU host for “computationally restricted” operating systems, like iPadOS and iOS :)
here, it’s running a full Debian VM [pretty darn snappily] on an non-jailbroken, latest-OS iPad, with no tethering needed; using @UTMapp’s UI
Show this thread
~~STATUS: COMPUTATIONALLY FORBIDDEN~~
STATUS: FORBIDDEN, STILL COMPUTING


Reference Image Boot (s)
TCTI
TCI
0
10
20
30
40
50
60
70
80


Joy Sparked (%)
TCTI
TCI
0
20
40
60
80
100
120

LESSONS LEARNED:
WHEN LIFE FORBIDS YOU FROM COMPUTING
SPARK JOY WITH A NEWER, WEIRDER COMPUTER

[QUESTIONS&ANSWERS]