goto; conference

# Web Assembly

Nick Bray

ncbray@google

# Setting the stage

# Native code on the web - today

Game engines: Unity / Unreal

Languages: repl.it

Emulators: Dosbox, JS Linux, NaCl Development Environment

PDF Viewing

Media decoding

# Web != native

Asynchronous

No threads with shared state

Inconsistent Performance

*Porting code to the web is painful!*

## … but the web is awesome

Open

Secure

Portable

Ephemeral

# In the beginning

HTML (1991)

JavaScript (1995)

Plugins

- NPAPI (1995-2015) - PDF documents
- ActiveX (1996-2015) - “document embedding”
- Shockwave / Flash (1996-??) - animations
- Java Applets (1996-??) - applications

# Let’s try again

JavaScript performance wars (2008)

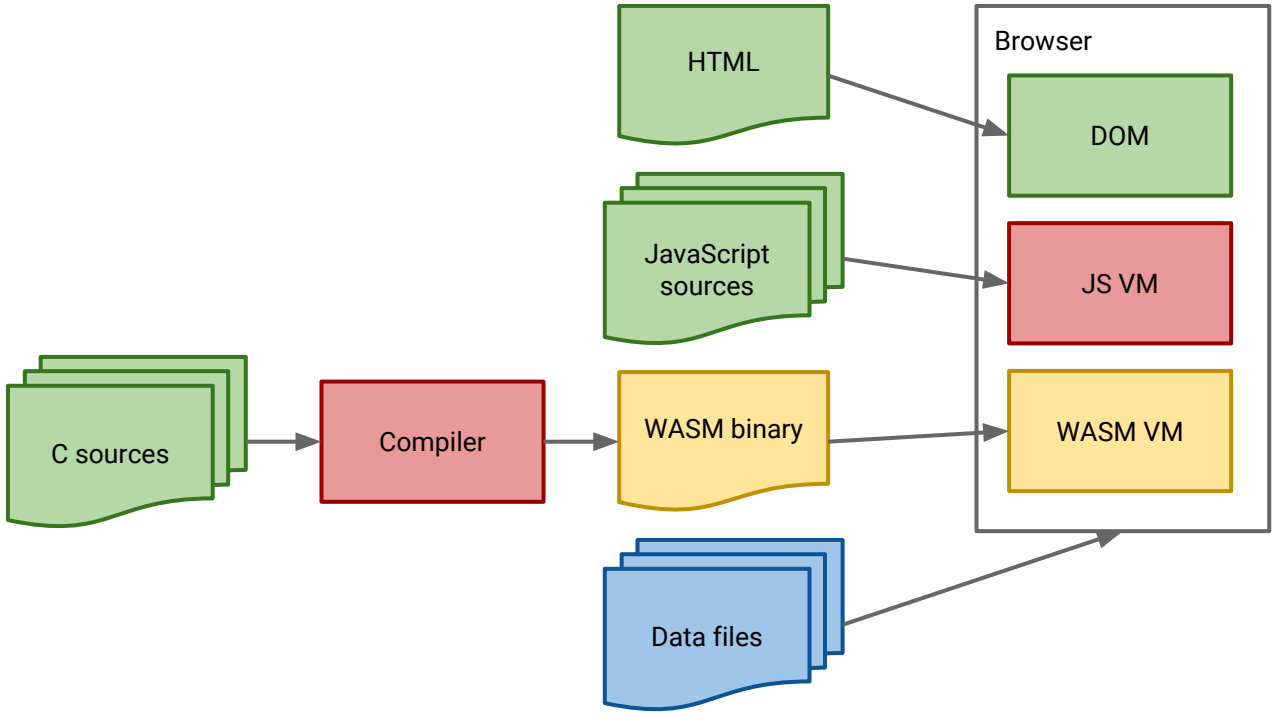
Native Client / Portable Native Client (2008 / 2013)

Emscripten / asm.js (2010 / 2013)

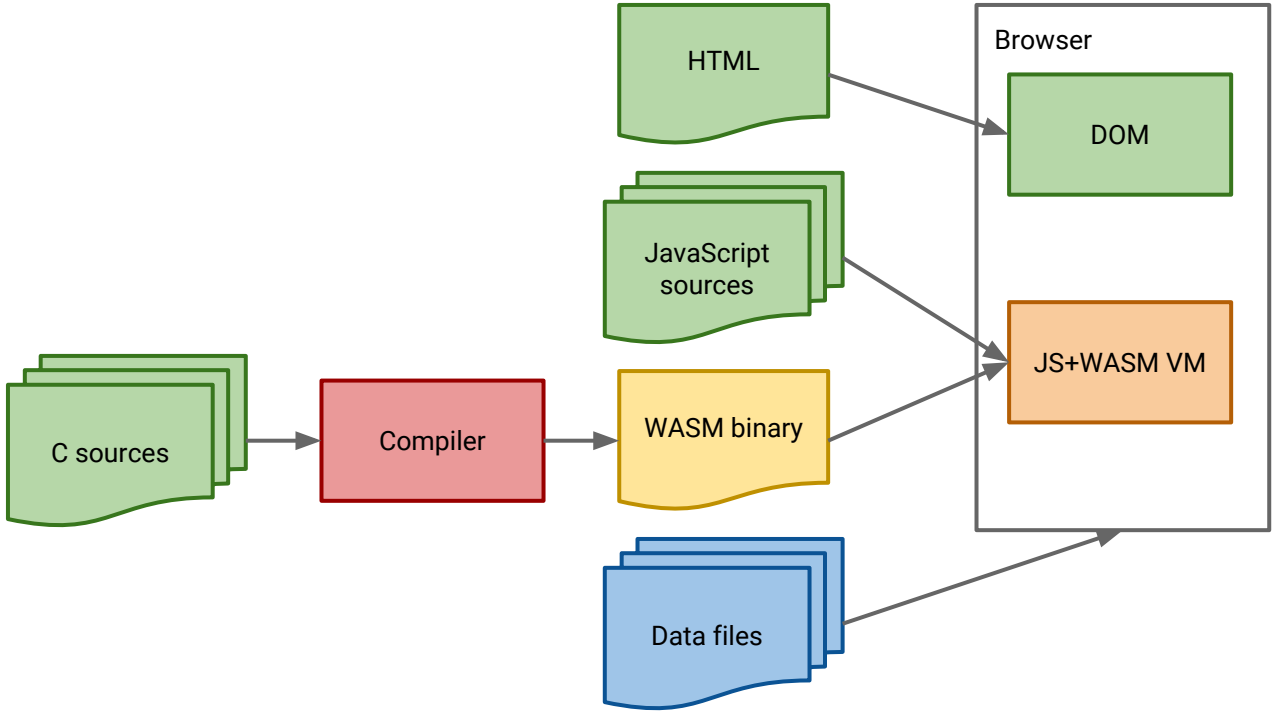
Web Assembly (2016?)

| | Year | Secure | Portable | Ephemeral | Cross Browser | Shared Memory |
|---|---|---|---|---|---|---|
| **JavaScript** | 1995 | ✓ | ✓ | ✓ | ✓ | x |
| **NPAPI** | 1995 | x | x | x | x | ✓ |
| **ActiveX** | 1996 | x | x | x | x | ✓ |
| **Flash** | 1996 | ~ | ✓ | ✓ | x | x |
| **Java Applets** | 1996 | ~ / x | ✓ | ✓ | x | ✓ |
| **Native Client** | 2008 | ✓ | ~ | ✓ | x | ✓ |
| **Emscripten** | 2010 | ✓ | ✓ | ✓ | ✓ | x |
| **asm.js** | 2013 | ✓ | ✓ | ✓ | ~ | x |
| **PNaCl** | 2013 | ✓ | ✓ | ✓ | x | ✓ |
| **Web Assembly** | 2016 ? | ✓ | ✓ | ✓ | ✓ | ✓ |

# WASM

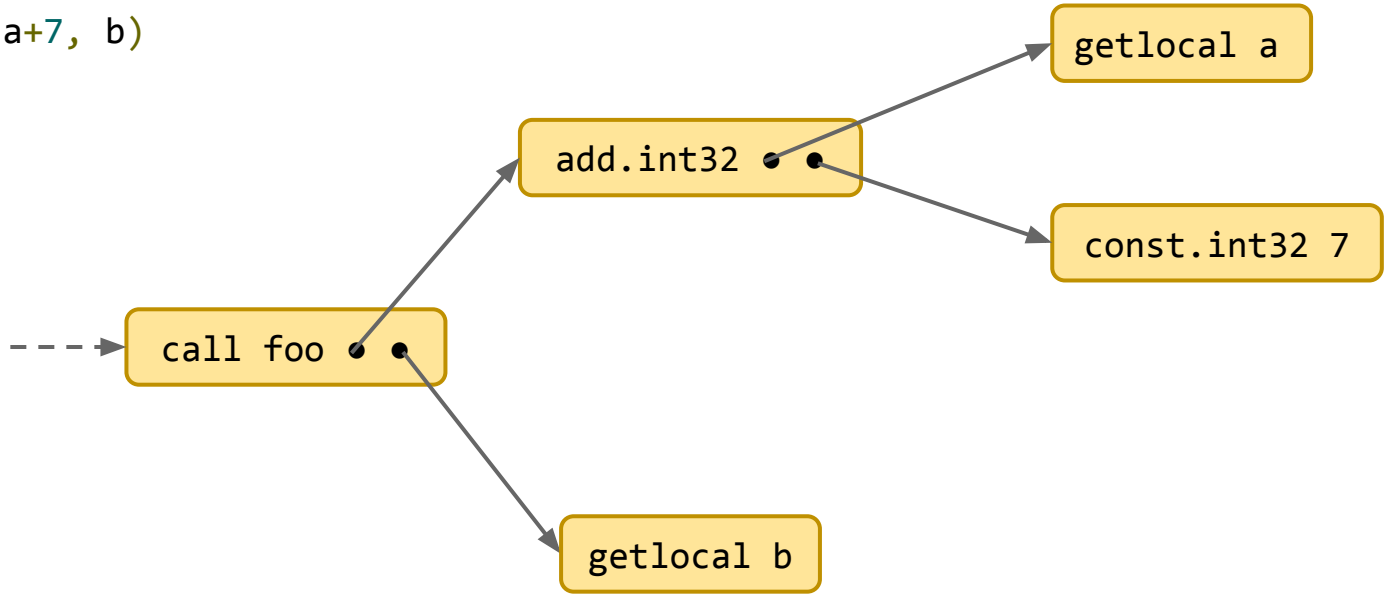

HTML
Browser
DOM
JavaScript sources
JS VM
C sources
Compiler
WASM binary
WASM VM
Data files

# … but actually…



# WASM Ops

```
foo(a+7, b)
```




Note: can statically infer all types

# WASM prototype vs Minified JS

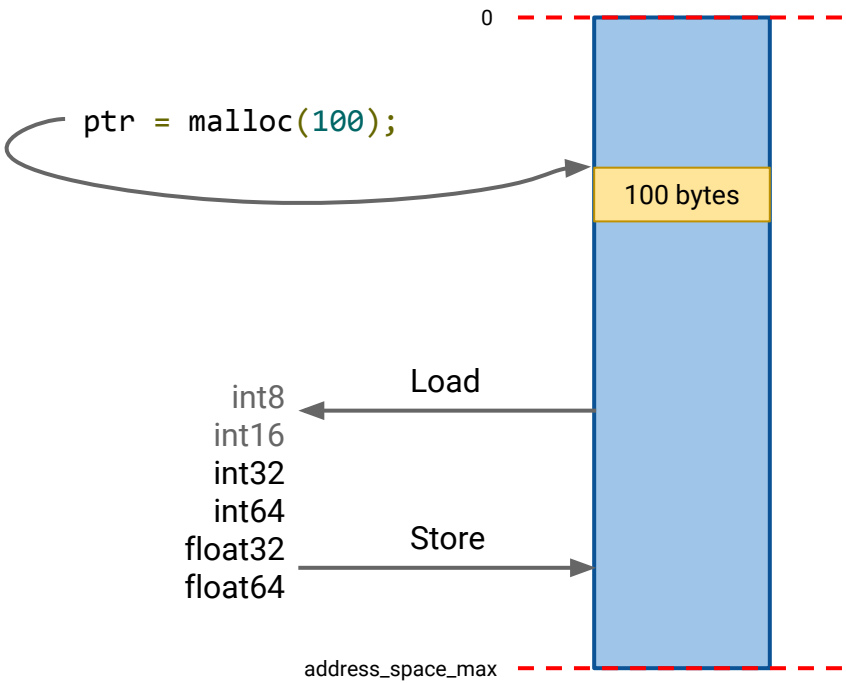
66% smaller uncompressed

26% smaller compressed

23x faster to parse

Early estimates from Mozilla

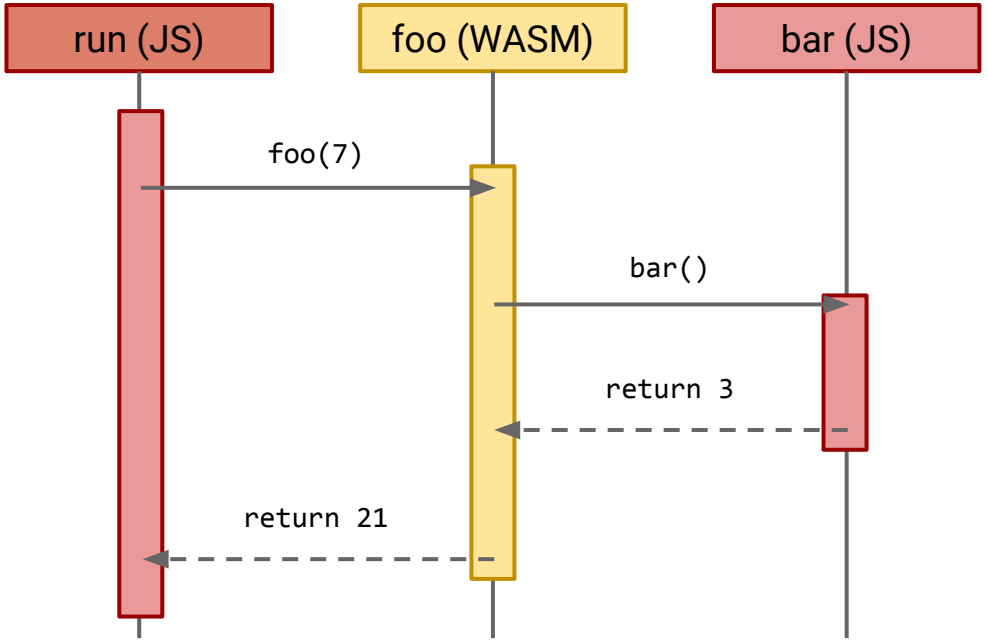
# WASM Memory


0
ptr = malloc(100);
100 bytes
Load
int8
int16
int32
int64
float32
float64
Store
address_space_max

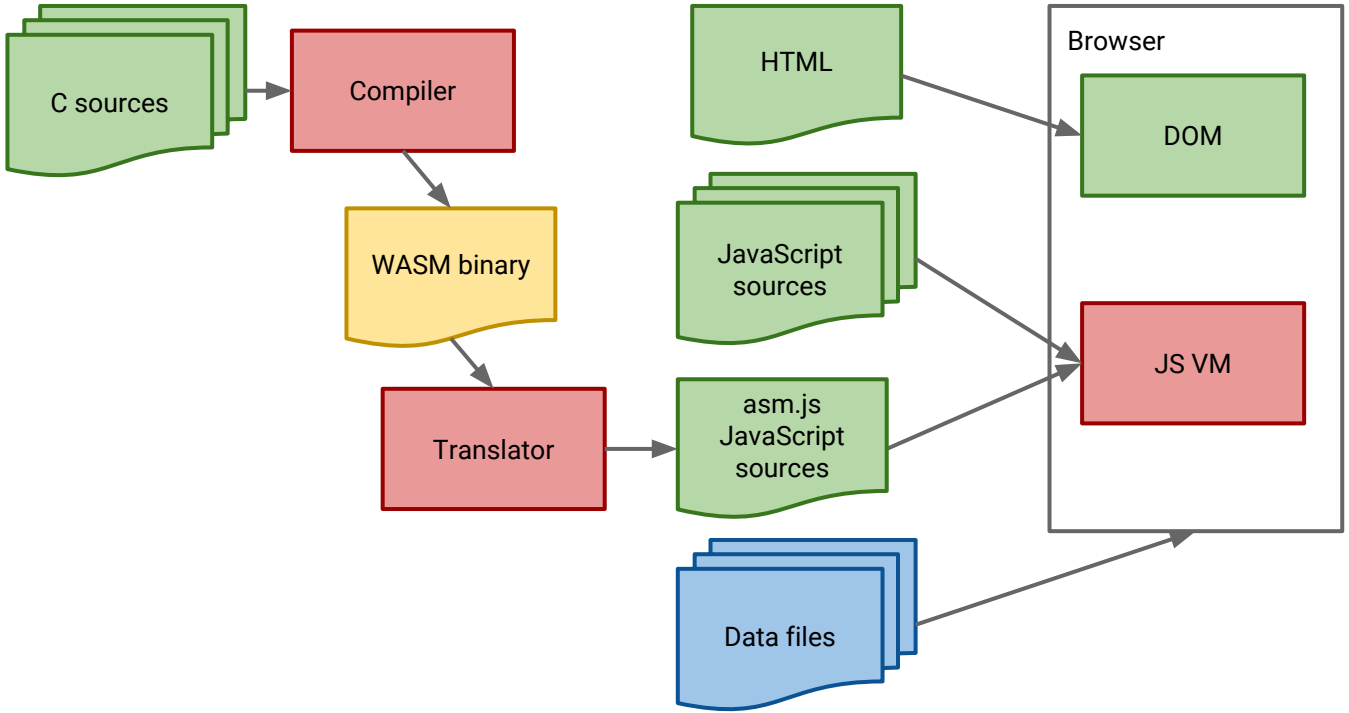

Objects

# WASM FFI

```
// FFI Interface
export int foo(int);
import int bar();

// C compiled into WASM
int foo(num int) {
  return num * bar(); // Call into JS
}


// JavaScript
function bar() {
  return 3;
}

function run(module) {
  var instance = WASM.createInstance(
    module,
    {bar: bar}
  );
  return instance.foo(7); // Call into WASM
}
```



# WASM Polyfill



# WASM Pollyfill

```
int add(int *a, int *b) {
  return *a + *b;
}
```

```
int32 add(int32 a, int32 b) {
  return(add.int32(load.i32(getlocal(a)), load.i32(getlocal(b))))
}
```

```
function _add($a, $b) {
  $a = $a | 0;
  $b = $b | 0;
  return (HEAP32[$b >> 2] | 0) + (HEAP32[$a >> 2] | 0) | 0;
}
```

# Demo

# Roadmap

v1.0 (2016?)

Single thread w/ event loop.  Loads fast, runs fast.

v 1.1 (2016?)

Threads! Blocking!

v1.2+ (2017?)

Exceptions, SIMD, dynamic linking, debugging, APIs

COPENHAGEN
INTERNATIONAL
SOFTWARE DEVELOPMENT
CONFERENCE 2015

goto; conference

# Questions?

Nick Bray

ncbray@google

https://github.com/WebAssembly

Google

# Backup Slides

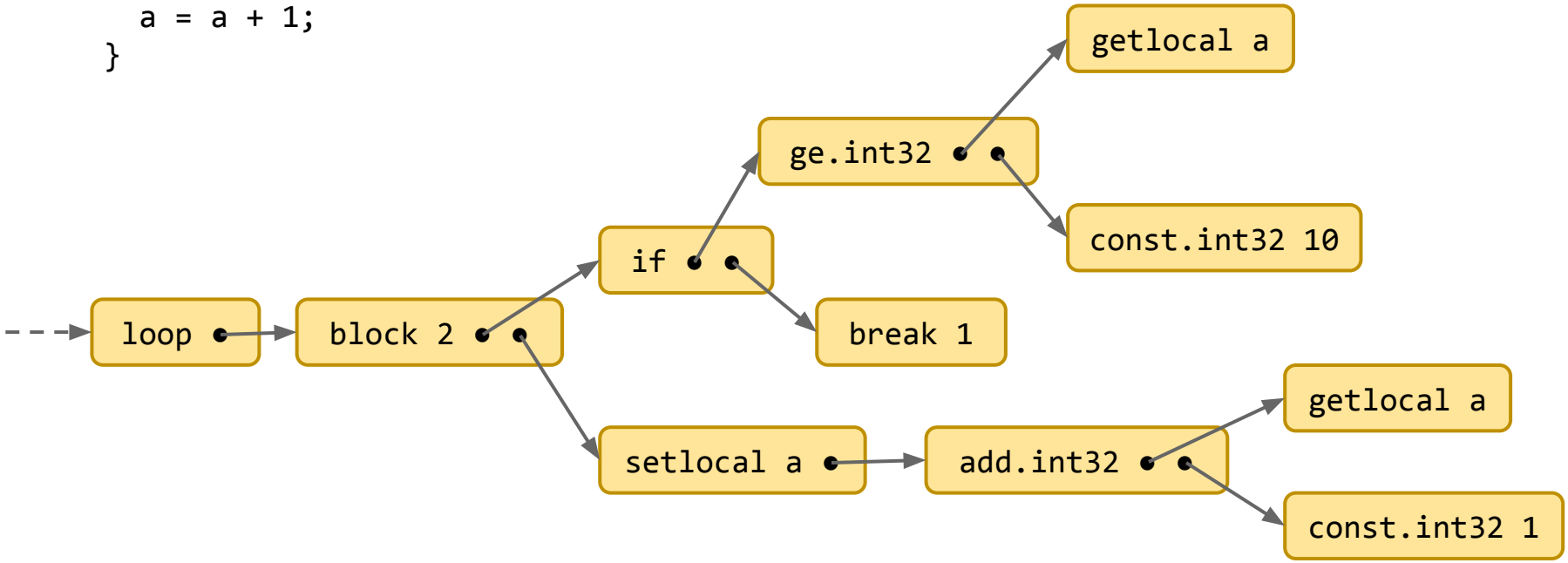
# … as opposed the the status quo.

```
foo(a+7, b)

  ~ is simplified ~

t0 = 7
t1 = a+t0
foo(t1, c)
```

```
const.int32 7 t0
add.int32 a t0 t1
call foo t1 c _
```

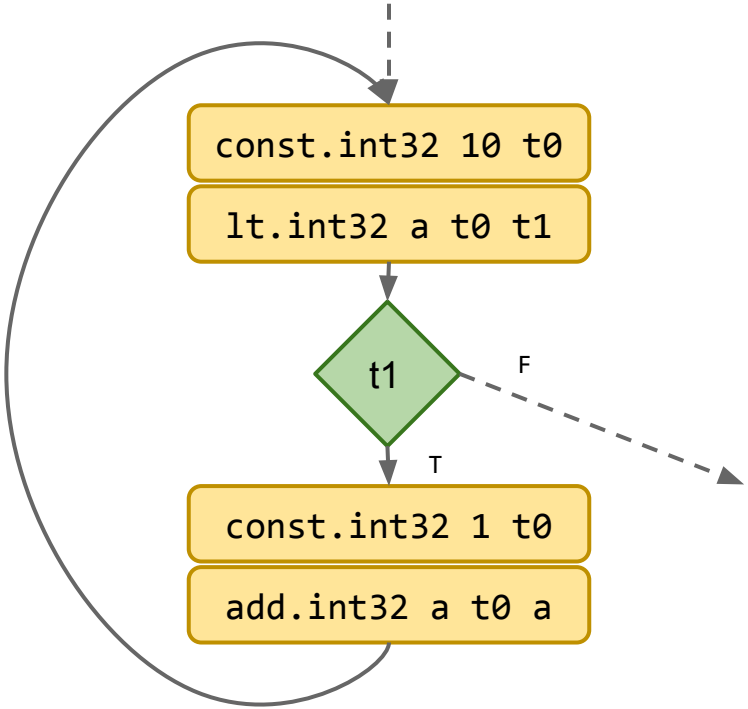
# … but this is really about control flow.

```
while (a < 10) {
  a = a + 1;
}
```



# … as opposed the the status quo.

```
while (a < 10) {
  a = a + 1;
}
```



# Competing for Address Space