Portable Native Client

David Sehr, Robert Muth, Jan Voung, David Meyer,
Betul Buyukkurt, Karl Schimpf, Jason Kim, Rafael Espindola,
Alan Donovan

Agenda

Google

Motivation

Approach

Safe Translation

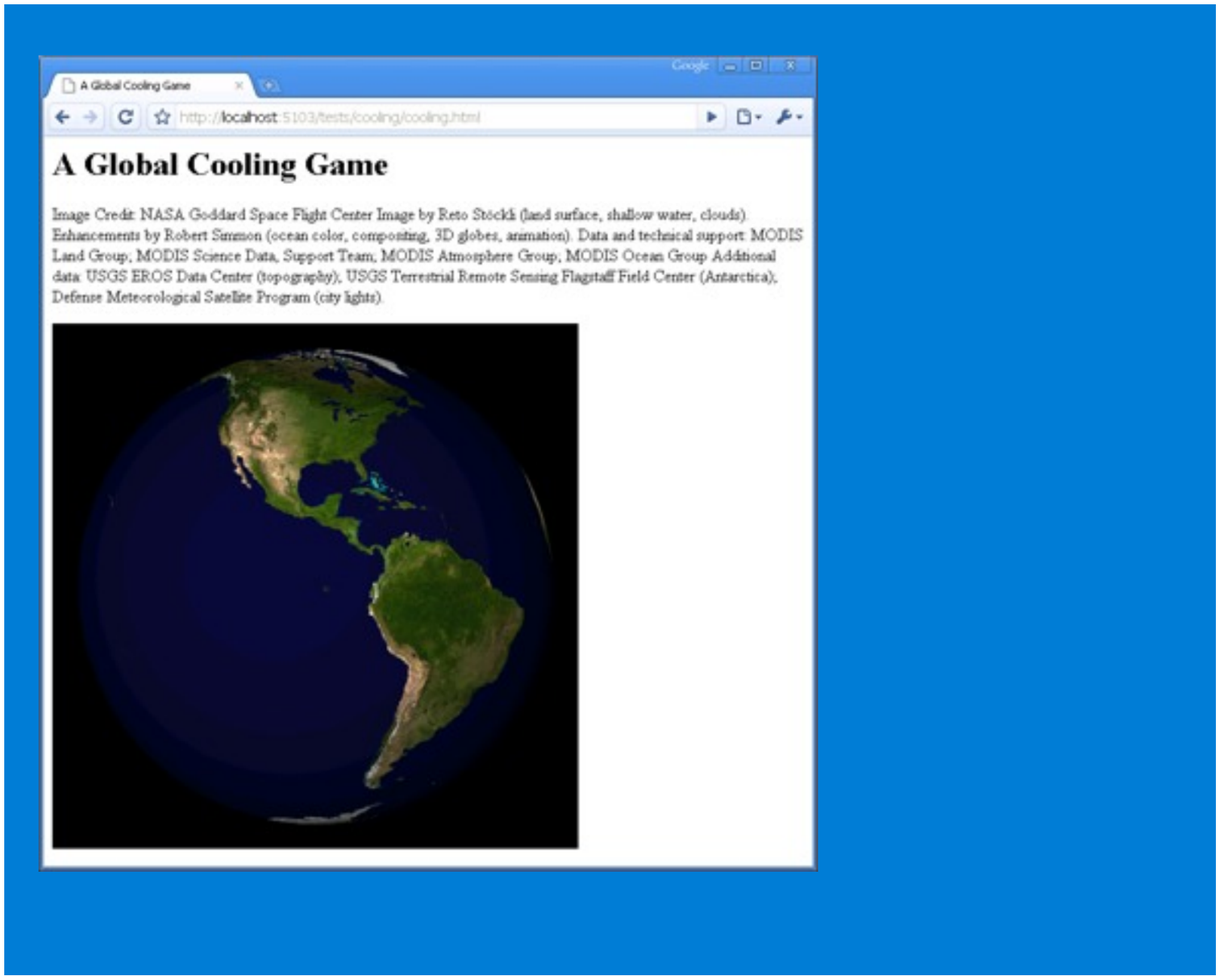
Bitcode as an Interchange Format

Status

Future Work

Motivation

A NaCl-Enabled Web Application



A NaCl-Enabled Web Application

Google

Your favorite language

A NaCl-Enabled Web Application

Your favorite language

Screened for malicious instructions

A NaCl-Enabled Web Application

Your favorite language

Screened for malicious instructions

System calls moderated by a virtualized OS

A NaCl-Enabled Web Application

Your favorite language

Screened for malicious instructions

System calls moderated by a virtualized OS

Performance within 5% of native code

Applications with NaCl

Google

Where Native Client Started

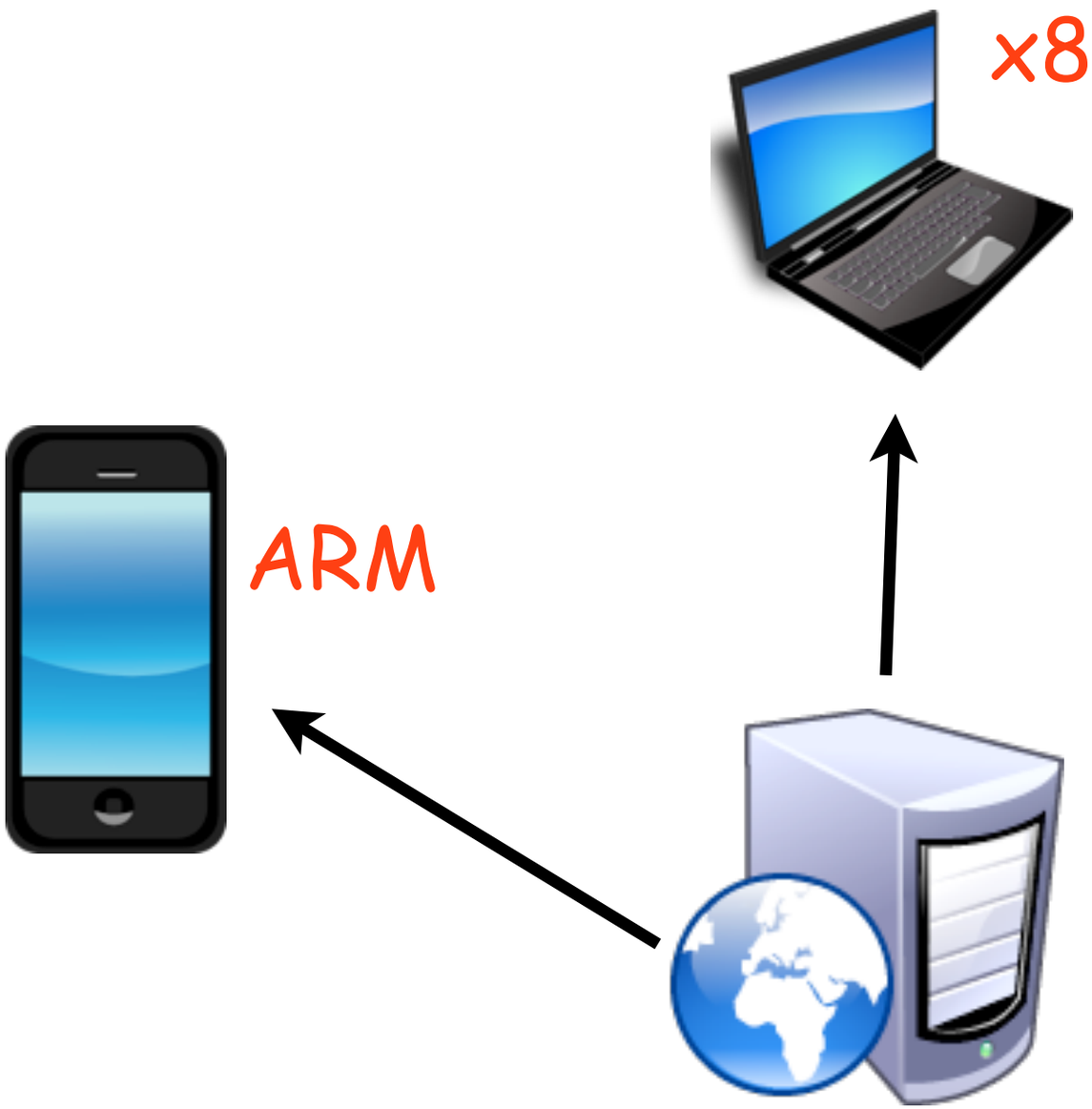
Google

OSX
Windows
Linux

x86-32

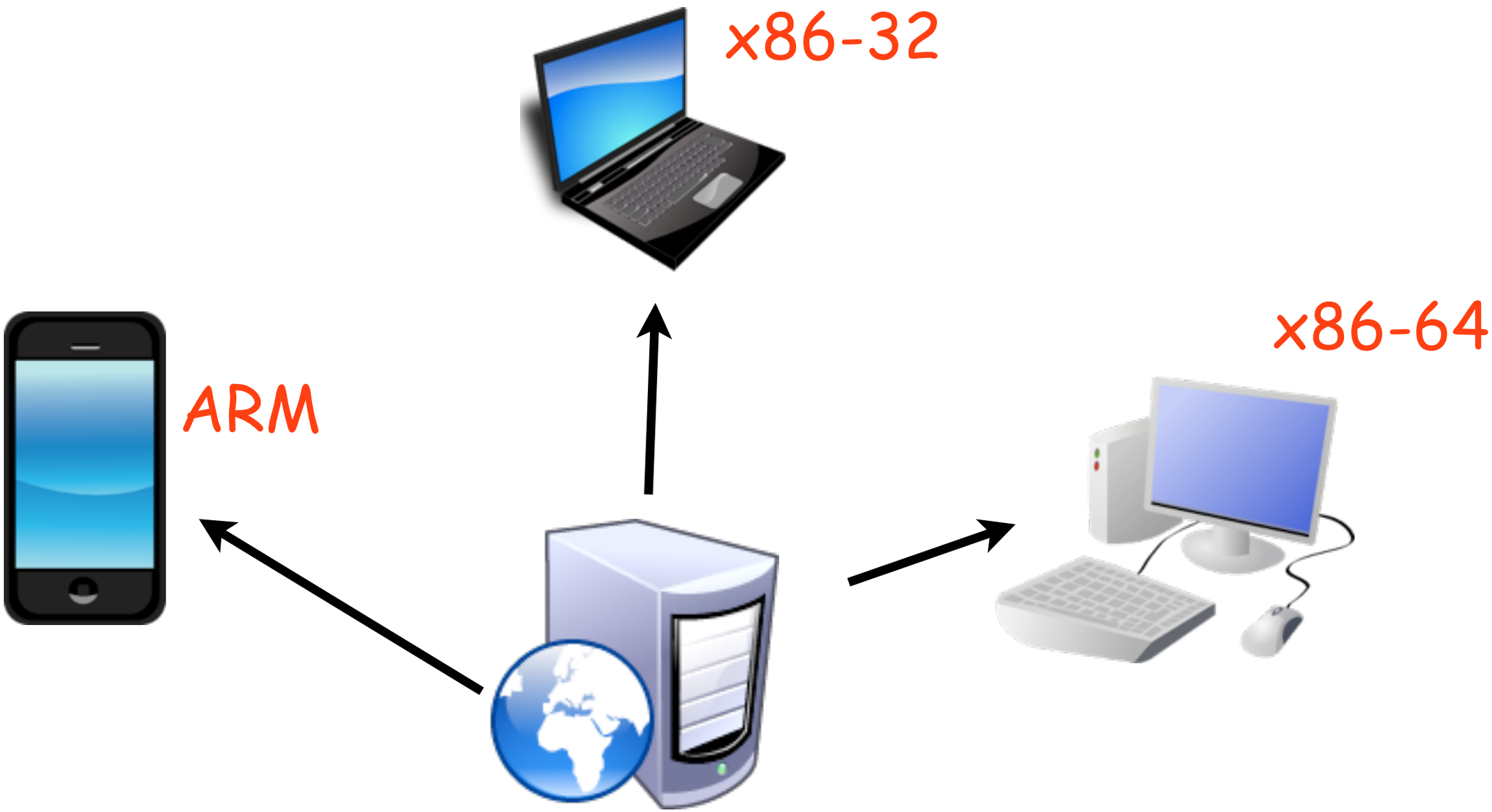
Where We Went Next

Google



What Developers Want

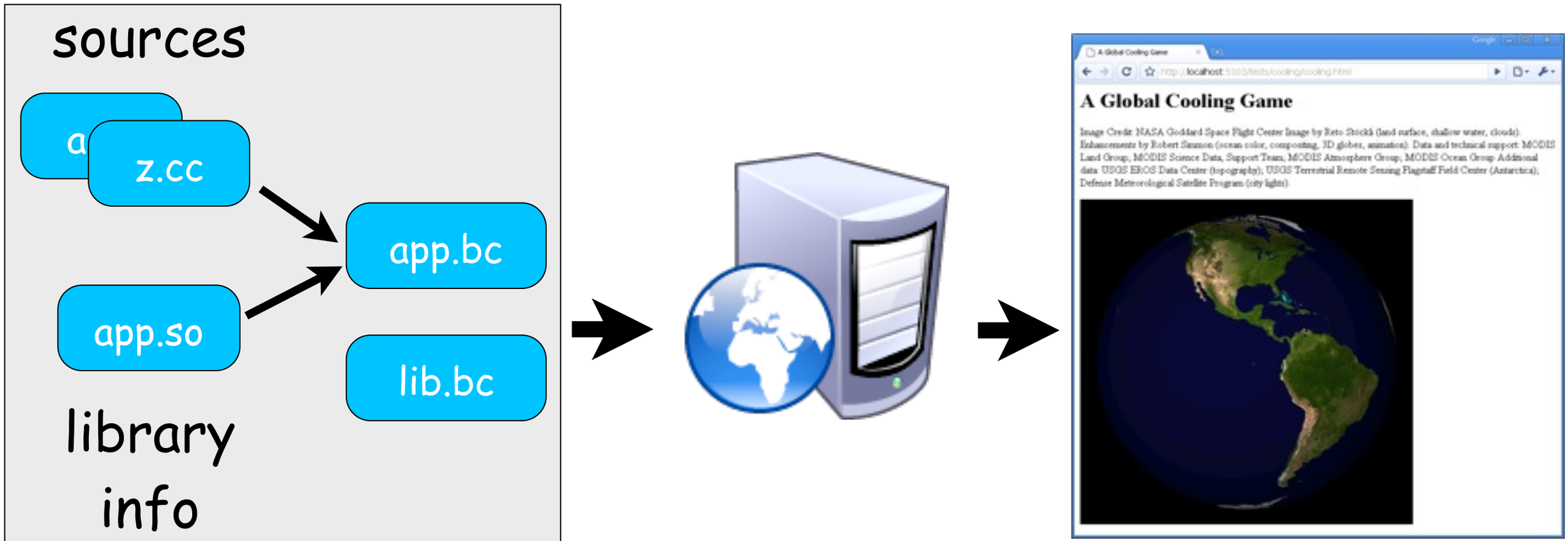
Google



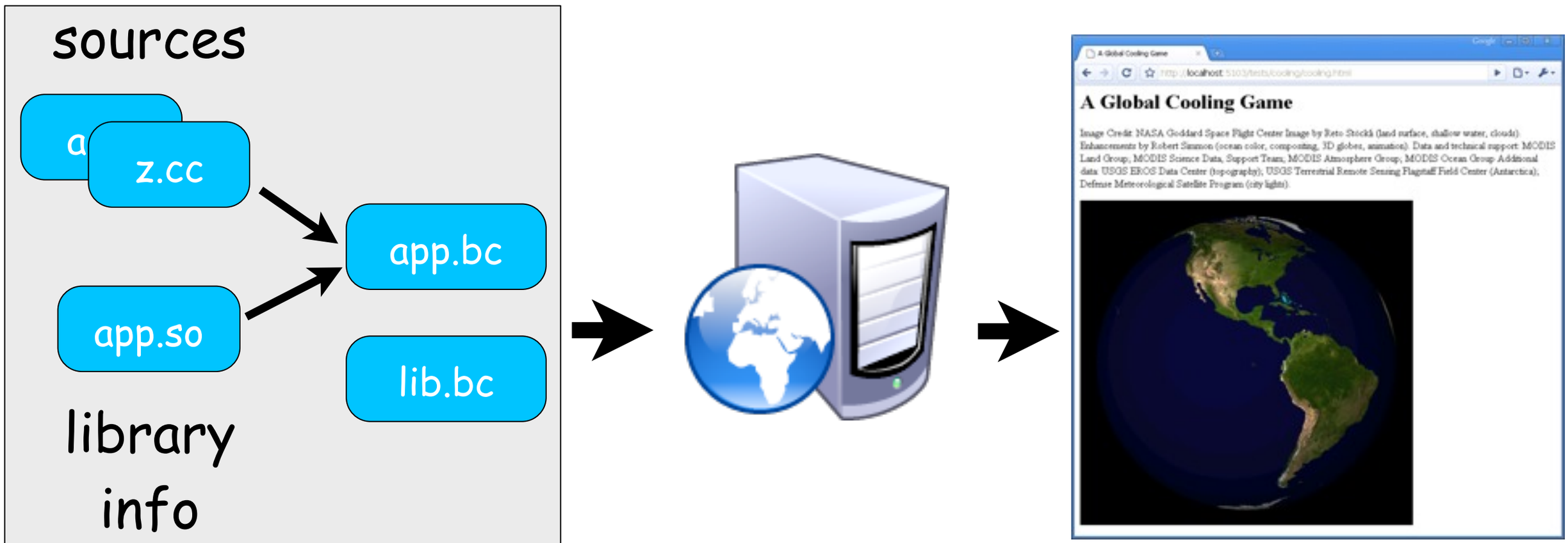
Only one porting effort

Approach

Application Life Cycle



Application Life Cycle



Bitcode is PNaCl's distribution format

Client side

Native Client Helper

http://myurl/myapp.bc

translation engine

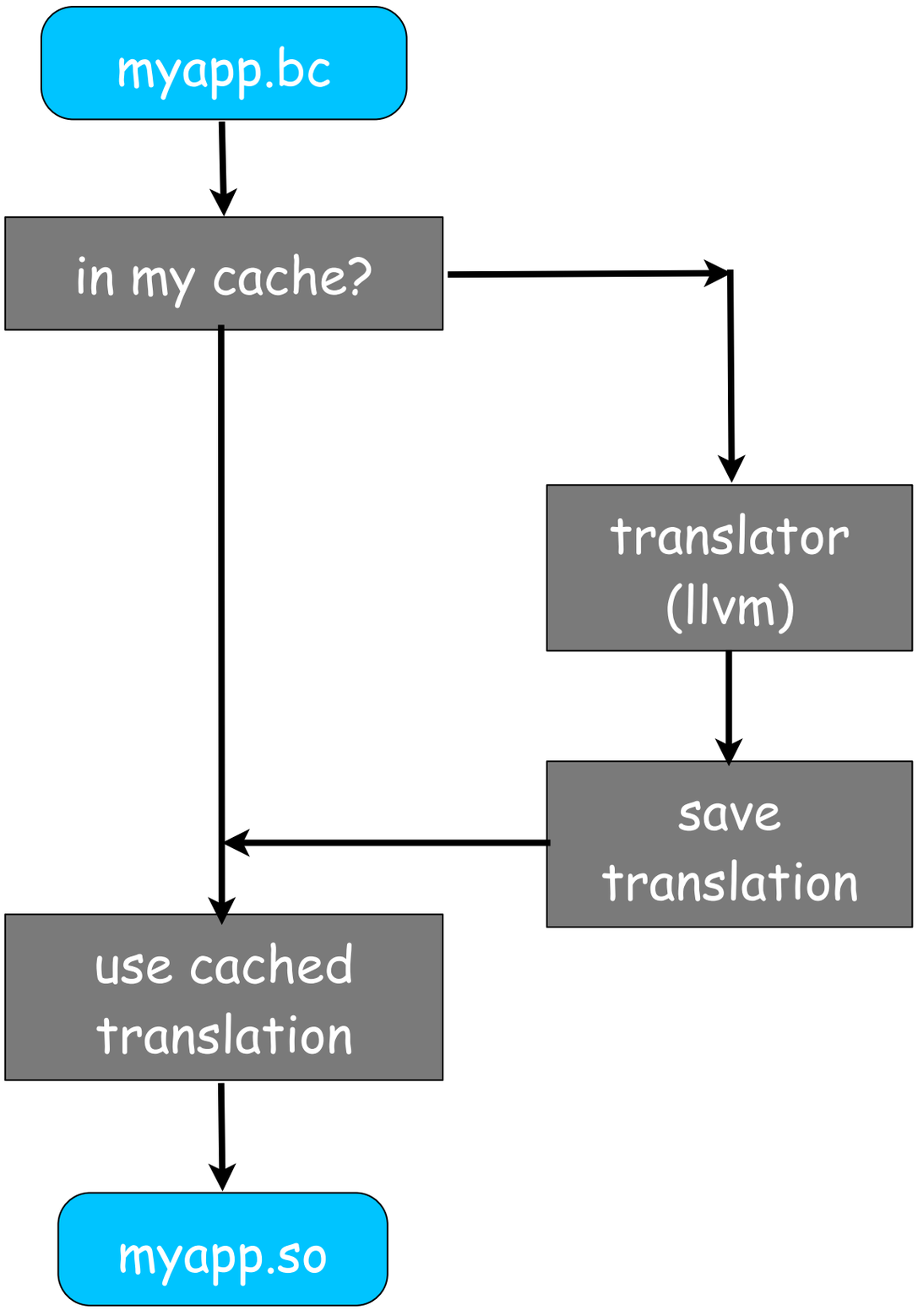
myapp.so

ELF
x86, x64, or ARM

NaCl sandbox

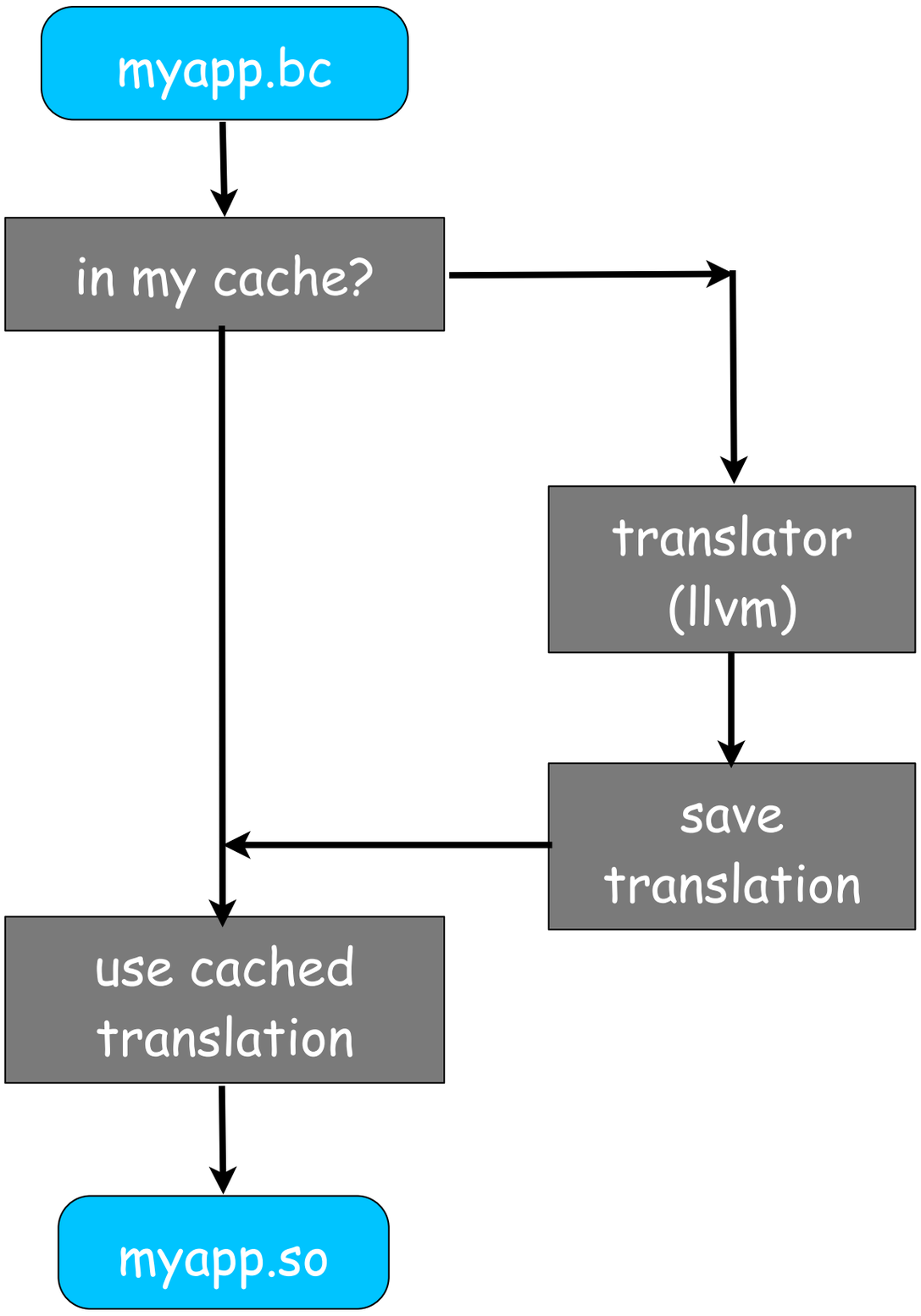
Translation Engine

Google



Translation Engine

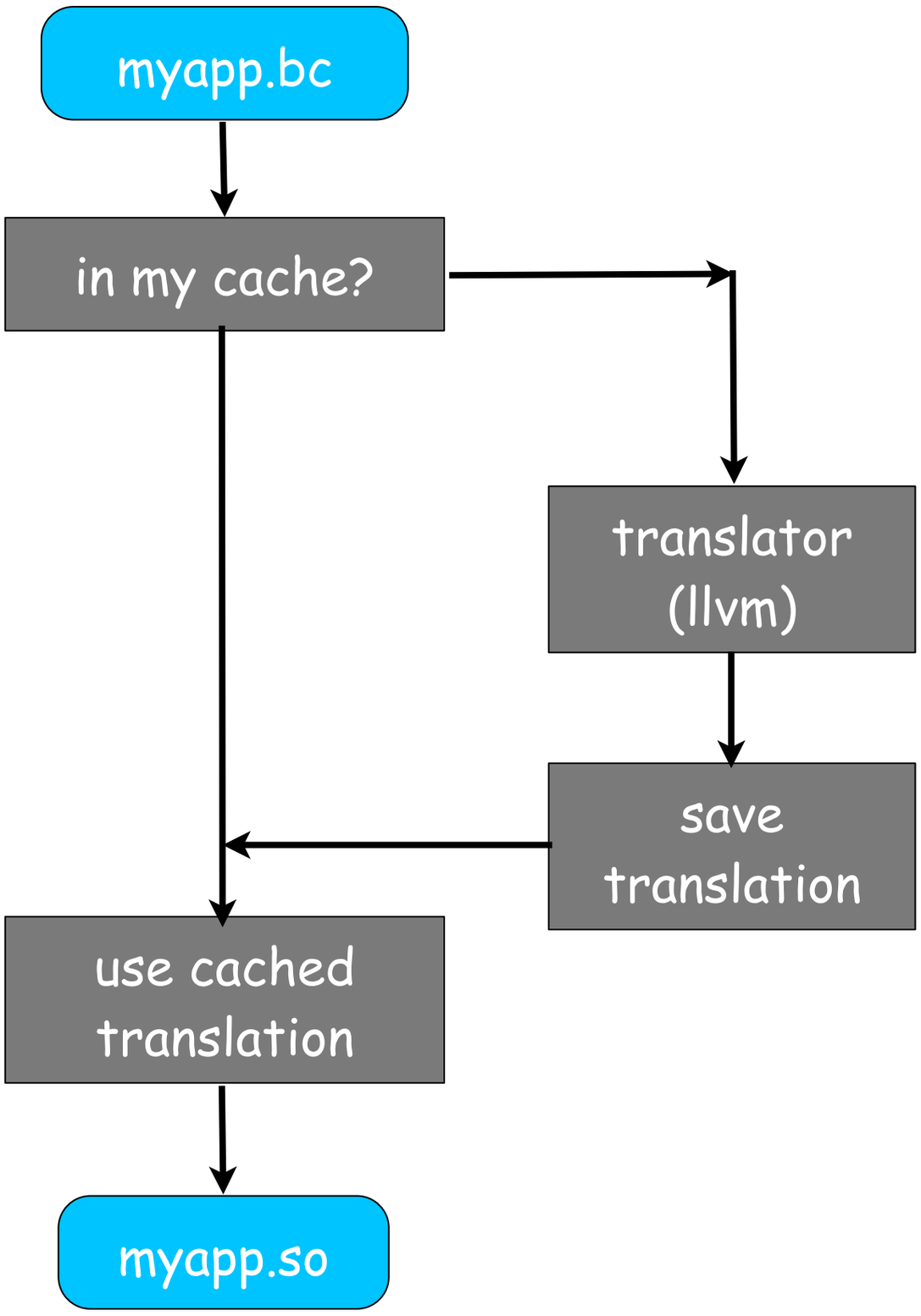
Google



Know the platform (uarch)

Translation Engine

Google



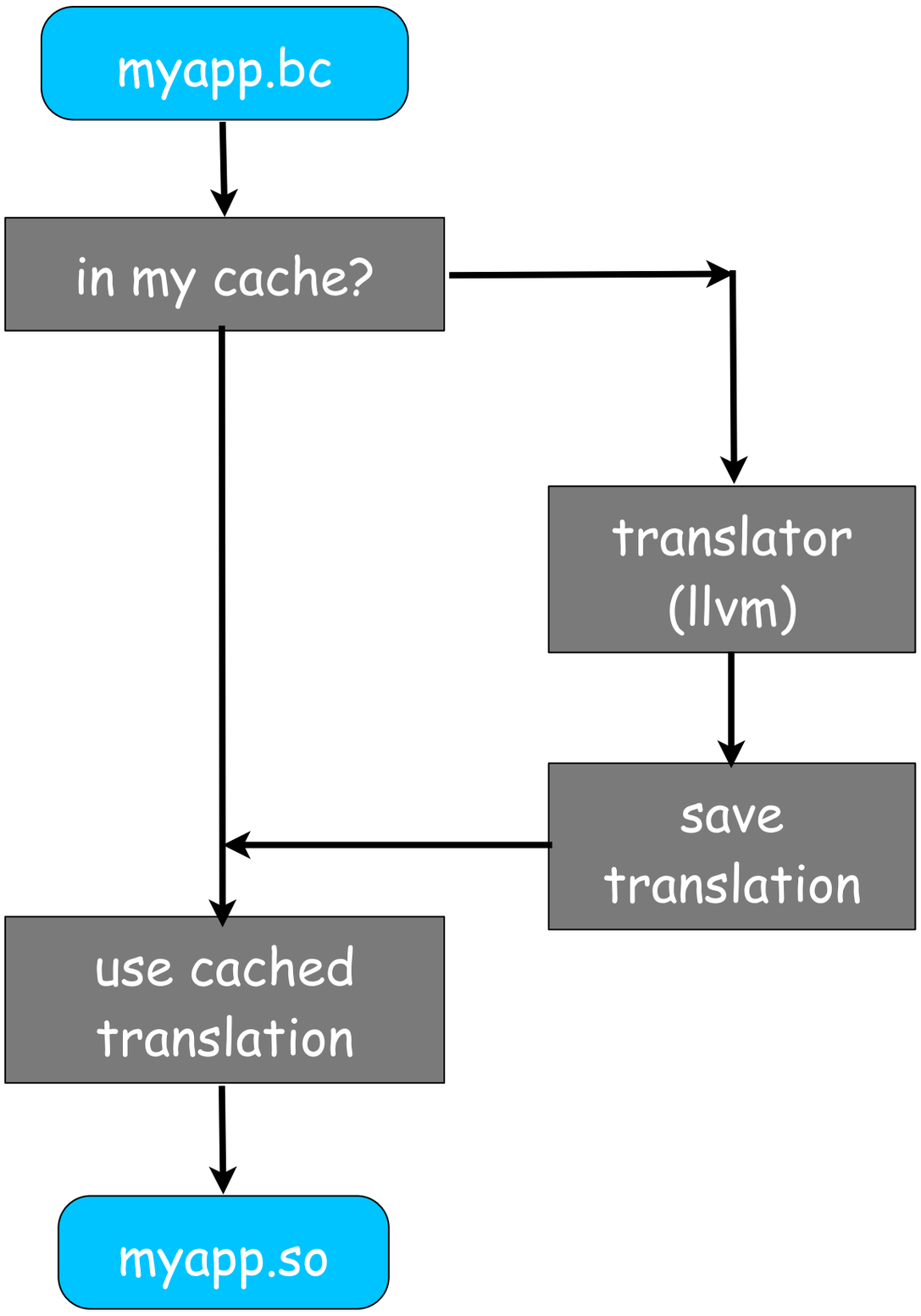
Know the platform (uarch)

Can collect/use profiling data

Webpage-specific specialization

Translation Engine

Google



Know the platform (uarch)

Can collect/use profiling data

- Webpage-specific specialization

Can translate at

- invocation time
- install time
- asynchronously

Safe Translation

Translating in a Sandbox

Google

The translator must run in the browser

Malicious bitcode files are a potential attack vector

Translating in a Sandbox

Google

The translator must run in the browser

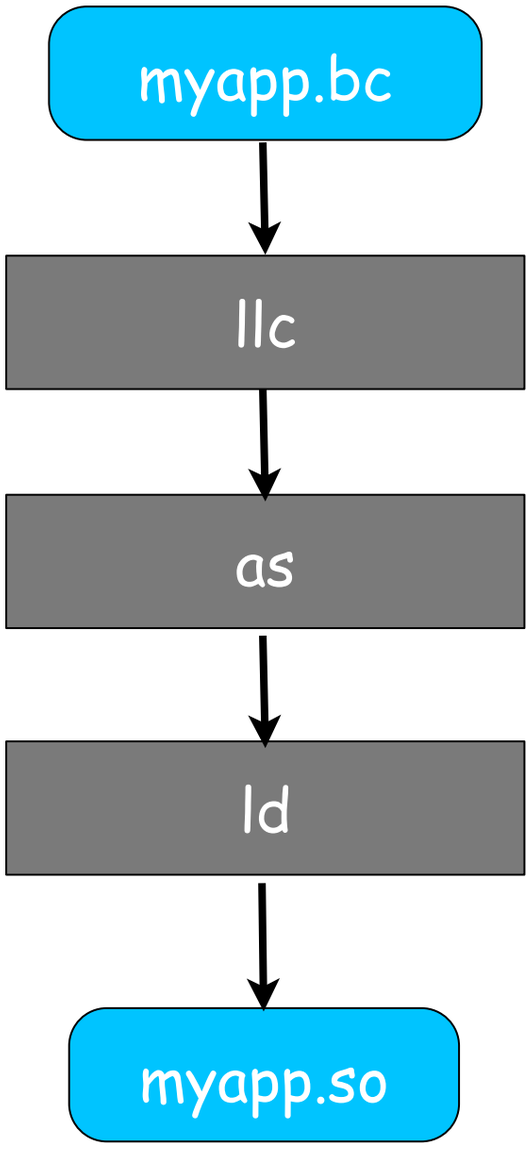
Malicious bitcode files are a potential attack vector

Translator phases are run as NaCl modules

Translator

Google

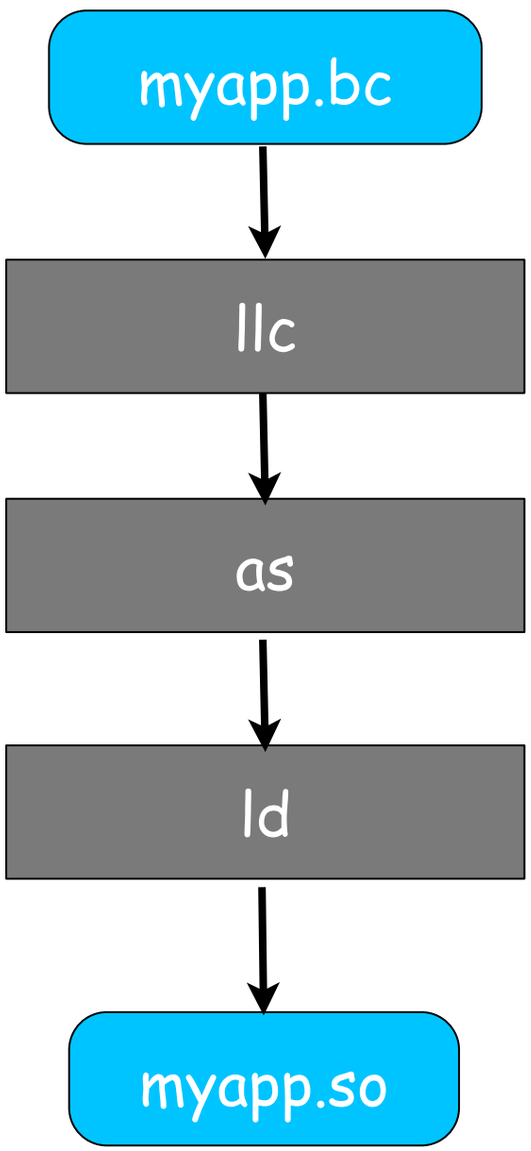
today



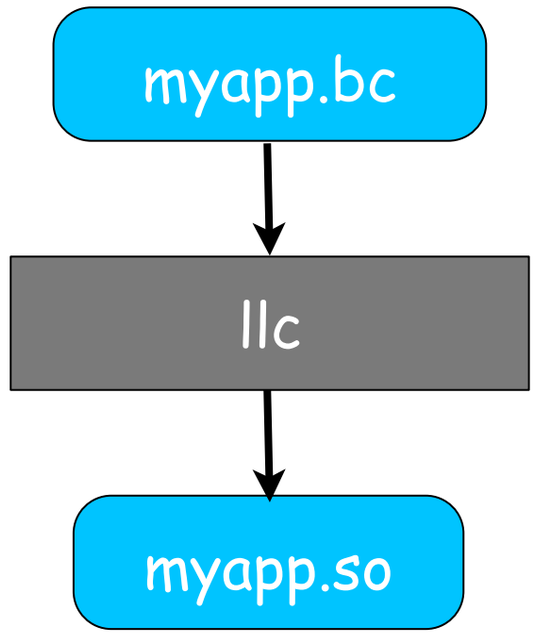
Translator

Google

today



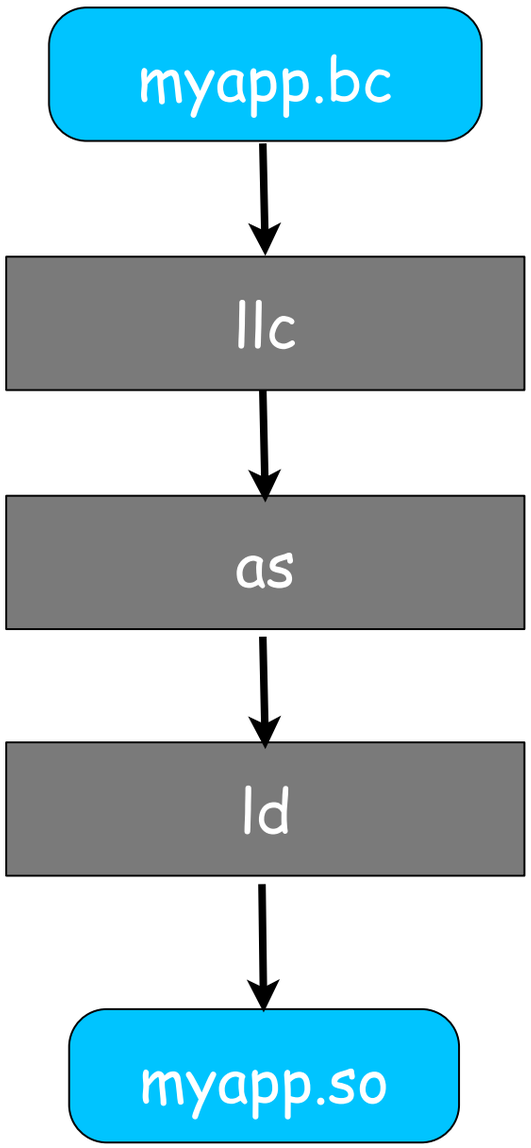
what we want



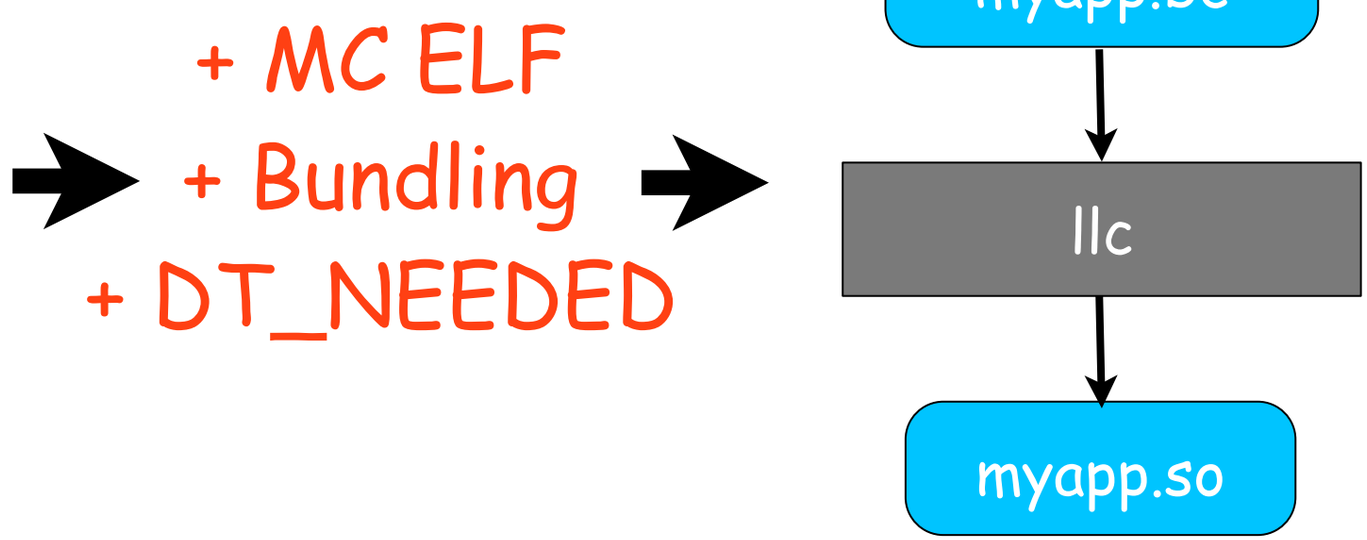
Translator

Google

today



what we want



Bitcode as an Interchange Format

Target Model

Google

- Address space / data model
  - ILP32 (sizeof(int) == sizeof(long) == sizeof(void*))
  - sizeof(va_list) == 24
  - 1GB maximum total address space
  - Stack pointer starts at the top of the address space
- Data types
  - IEEE fp
  - “natural” alignment
    - (e.g., double is aligned 0mod8)
- Byte order
  - Little Endian

Target Model

Google

C++ Exception Handling

- x86-32 Linux model

varargs

- sizeof(va_list) == 24
- Front end emits va_arg instruction

setjmp

- Consistent jmp_buf size (work in progress)

Target Model

Google

- Calling conventions
  - Bitcode file is calling convention neutral
  - Actual target convention determined by translator
- Concurrency and memory model
  - Assume a least common denominator
    - Store ordering within a thread
    - Explicit synchronization across threads
  - We expect people to use llvm atomic/barrier intrinsics where needed

Bitcode as an Interchange Format

Google

PNaCl will need bitcode stability

- Developer expects published bitcode to work forever

Download size is startup time

- .bc is ~3x bigger than .nexe, ~1.9x when .gz
- .bc is ~6x bigger than .NET

How should we handle bitcode versioning?

Bitcode as an Interchange Format

Google

PNaCl will need bitcode stability

- Developer expects published bitcode to work forever

Download size is startup time

- .bc is ~3x bigger than .nexe, ~1.9x when .gz
- .bc is ~6x bigger than .NET

How should we handle bitcode versioning?

We need your help!

Status

What's running?

Google

One bitcode file translates, validates, and runs on three architectures

- All of SPEC2000 int and the four C fp tests pass

The translator is sandboxed

- llc, as, ld runs as a NaCl module on x86-32 and 64

A few areas of portability work remain

- C++ exception handling on ARM is incomplete
- setjmp/longjmp is just coming together

CodeGen Work

Google

Control and data sandboxing on ARM

Robert, Cliff

Control and data sandboxing on x86

Robert, Alan, Jan, David

ILP32 on x86-64

Jan, David

x86-32 and x86-64 MC ELF contributions

Rafael

ARM MC ELF contributions

Jason

Front end work

ILP32 for x86-64

Jan, DavidM

Varargs

DavidM

Exception handling, setjmp

Robert

Future Work

Directly Producing .so's

- ELF MC
  - ARM support is still incomplete
- MCAssembler
  - "Bundling" support for NaCl pseudo-instructions
- .so generation
  - Simulated linking to collect symbols
  - Emission work for DT_NEEDED

Intrinsics and/or Assembly

Google

One of the promises of NaCl is access to the performance that comes from hand-tuning while not sacrificing portability or safety.

How do we get to, e.g., AES instructions, etc.?

How do we optimize for cache configuration, etc.?

Other future work

Google

Clang

Other languages that could target bitcode

- .NET/Mono, ...

JIT support

Performance

- feedback directed optimization, ...

Bitcode size

Translation time

Want to Learn More?

Google

http://www.chromium.org/nativeclient

(Follow Portable Native Client link)

http://code.google.com/p/nativeclient