



Portable Native Client

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

Agenda



Motivation

Approach

Safe Translation

Bitcode as an Interchange Format

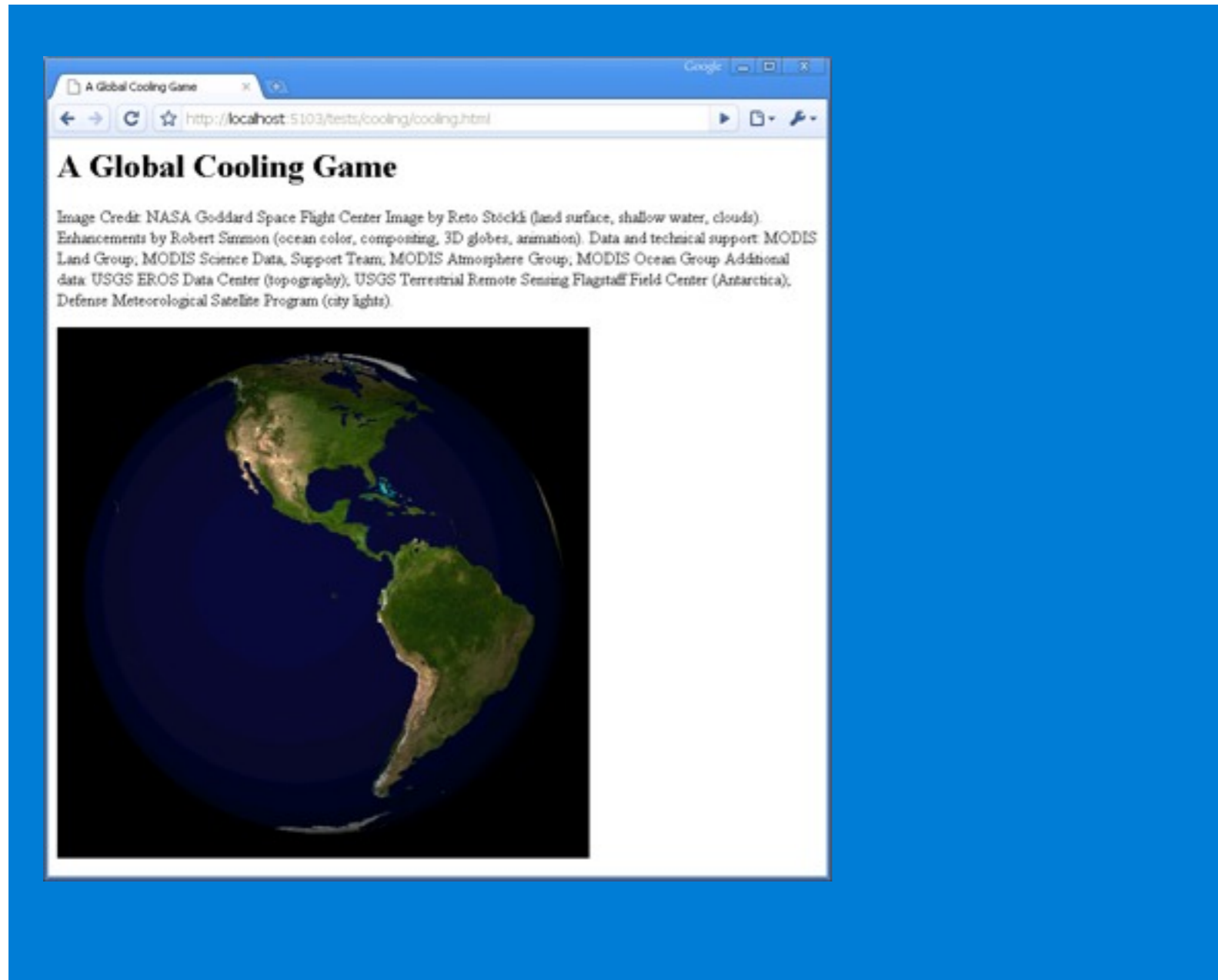
Status

Future Work

Motivation



A NaCl-Enabled Web Application



A NaCl-Enabled Web Application



The screenshot shows a web browser window with the title "A Global Cooling Game" and the URL "http://localhost:5103/tests/cooling/cooling.html". Below the title is a paragraph of text: "Image Credit: NASA Goddard Space Flight Center Image by Reto Stockli (land surface, shallow water, clouds). Enhancements by Robert Summon (ocean color, compositing, 3D globes, animation). Data and technical support: MODIS Land Group; MODIS Science Data, Support Team; MODIS Atmosphere Group; MODIS Ocean Group Additional data: USGS EROS Data Center (topography); USGS Terrestrial Remote Sensing Flagstaff Field Center (Antarctica); Defense Meteorological Satellite Program (city lights)". Below the text is a large image of the Earth from space. To the right of the globe is a diagram of an atom with a nucleus of red and blue spheres and three black electrons orbiting. The atom is enclosed in a grey rectangular frame with vertical bars, resembling a prison cell. A green box labeled "Native Client Helper" has two green arrows pointing towards the atom diagram.

Your favorite language

A NaCl-Enabled Web Application



The screenshot shows a web browser window titled "A Global Cooling Game" with the URL "http://localhost:5103/tests/cooling/cooling.html". The page content includes a title "A Global Cooling Game", a paragraph of image credits, and a large image of Earth. A green box labeled "Native Client Helper" is overlaid on the bottom right of the browser window, with a green arrow pointing from it to a stylized atomic symbol icon.

Your favorite language

Screened for malicious instructions

A NaCl-Enabled Web Application



The screenshot shows a web browser window titled "A Global Cooling Game" with the URL `http://localhost:5103/tests/cooling/cooling.html`. The page content includes a title, a paragraph of image credits, and a large image of Earth. A green box labeled "Native Client Helper" is positioned below the globe image, with a green arrow pointing from it to a stylized atomic symbol icon that is also enclosed in a green box.

Your favorite language

Screened for malicious instructions

System calls moderated by a virtualized OS

A NaCl-Enabled Web Application



The screenshot shows a web browser window titled "A Global Cooling Game" with the URL `http://localhost:5103/tests/cooling/cooling.html`. The page content includes a title, a paragraph of image credits, and a large image of Earth. To the right of the browser window is a green box labeled "Native Client Helper" with two arrows: one pointing left towards the browser and one pointing up towards a stylized atom icon inside a green square with vertical bars, representing a virtualized OS.

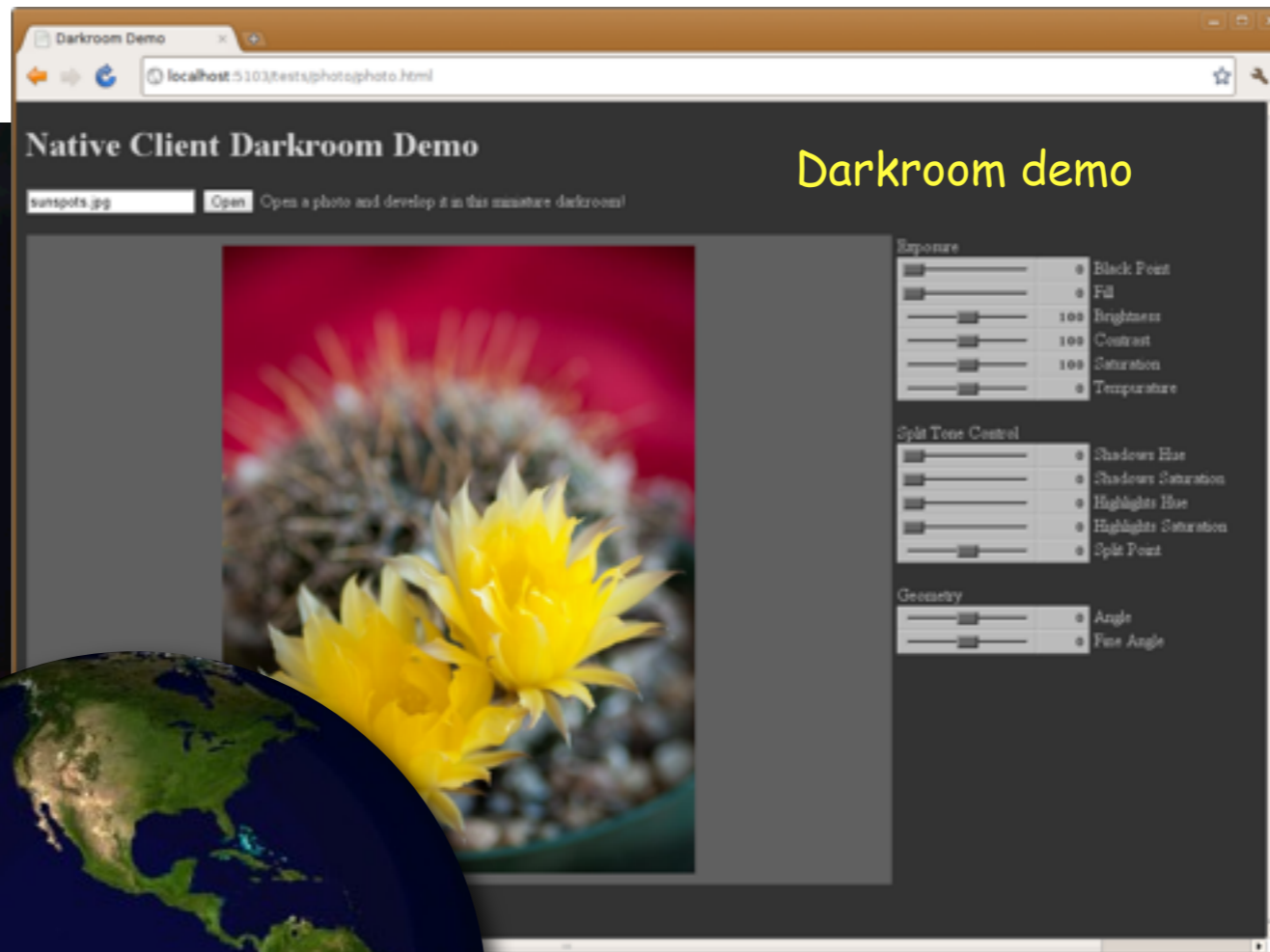
Your favorite language

Screened for malicious instructions

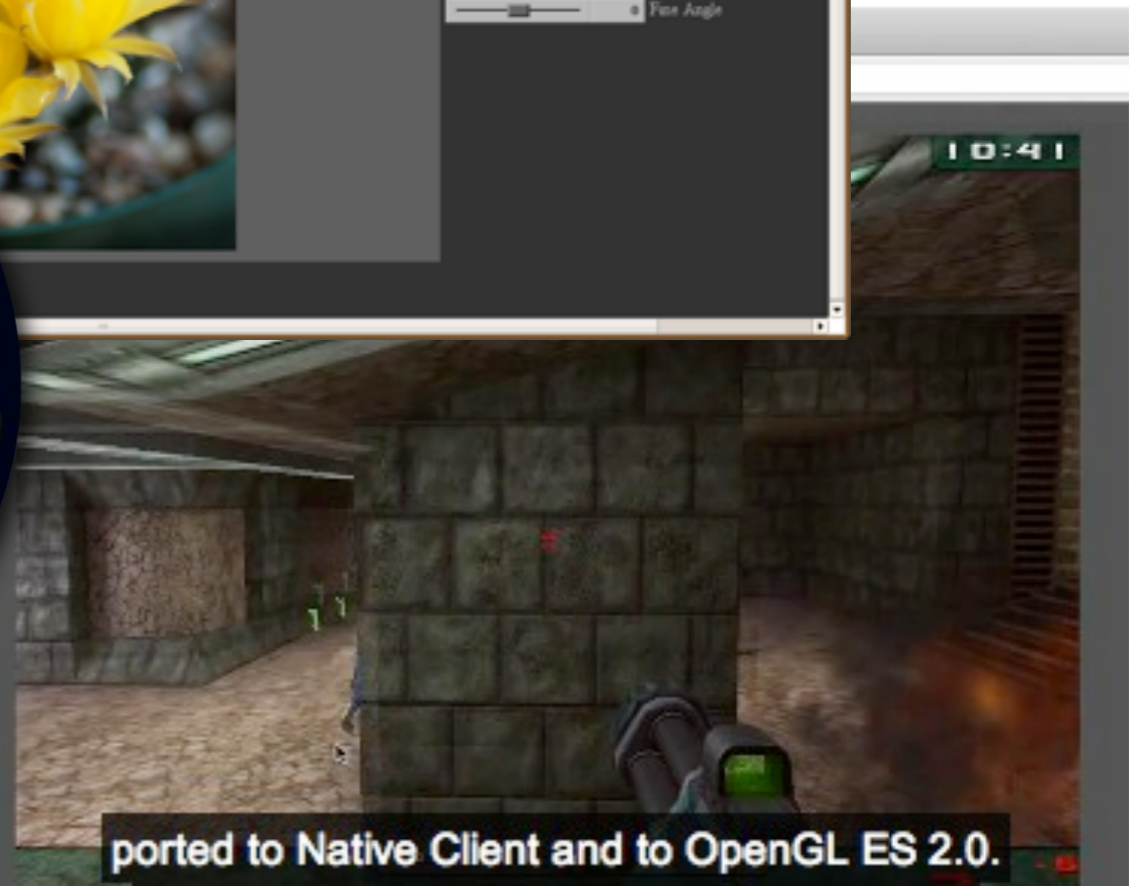
System calls moderated by a virtualized OS

Performance within 5% of native code

Applications with NaCl



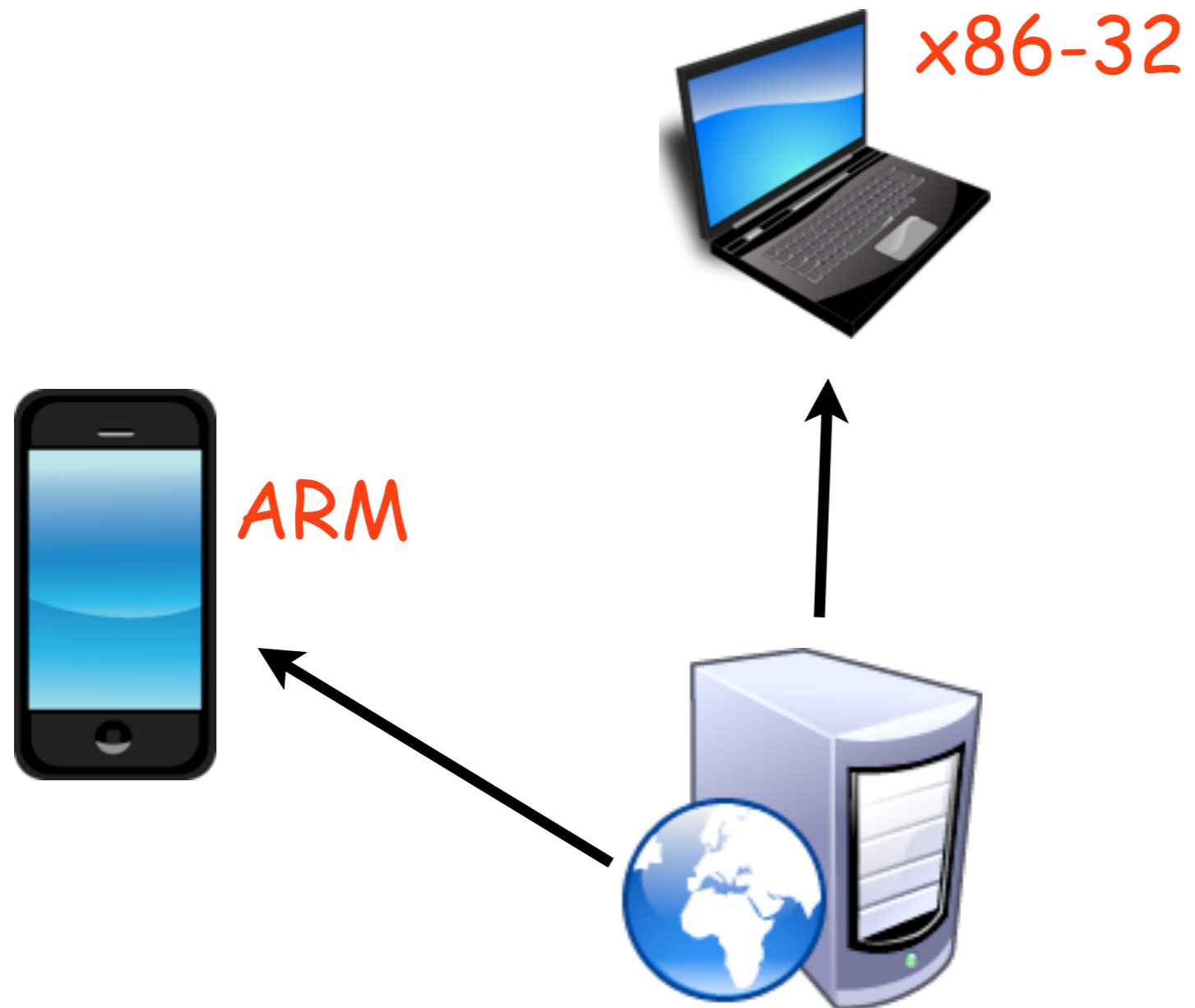
Nexuiz



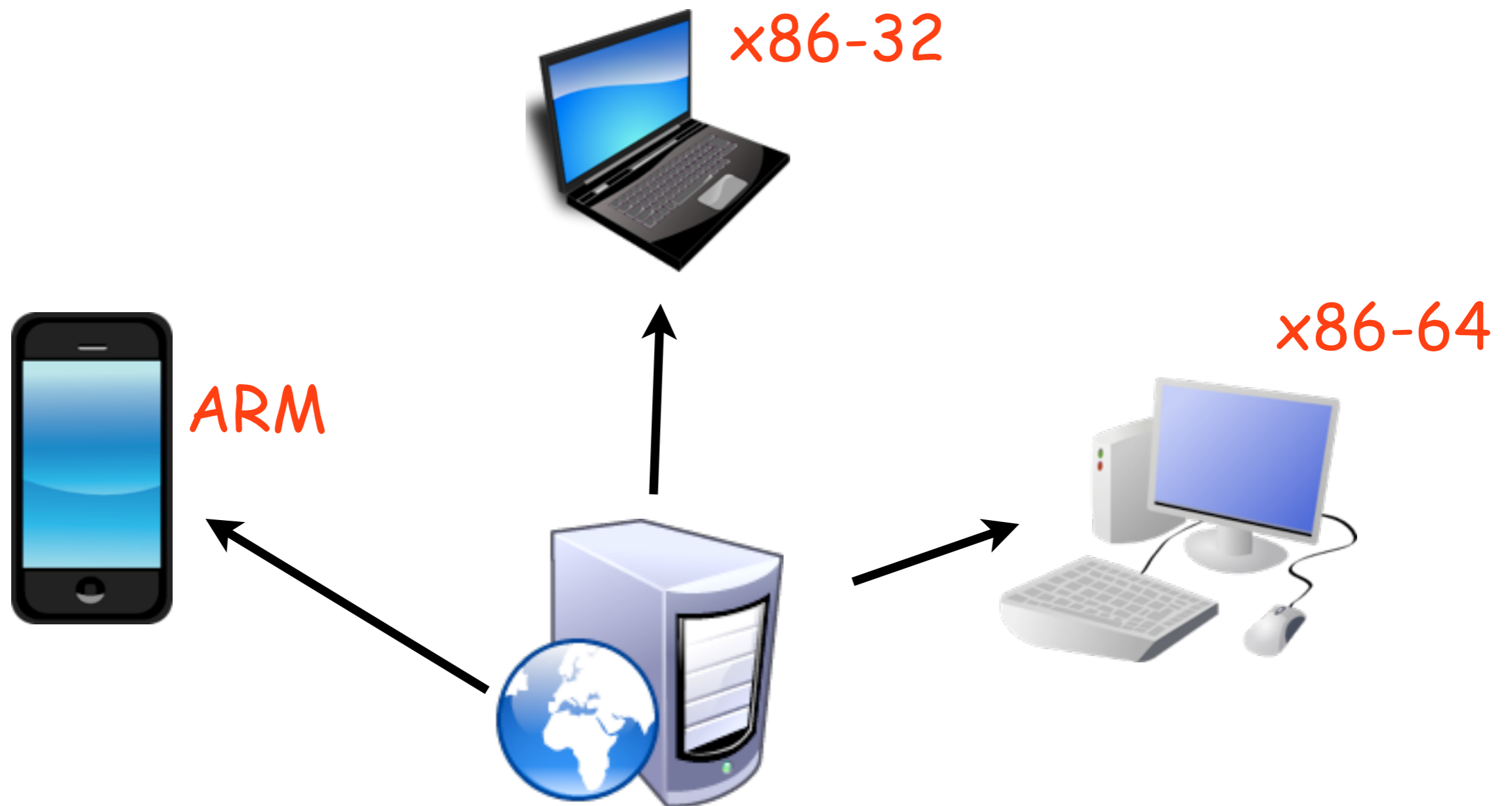
Where Native Client Started



Where We Went Next



What Developers Want

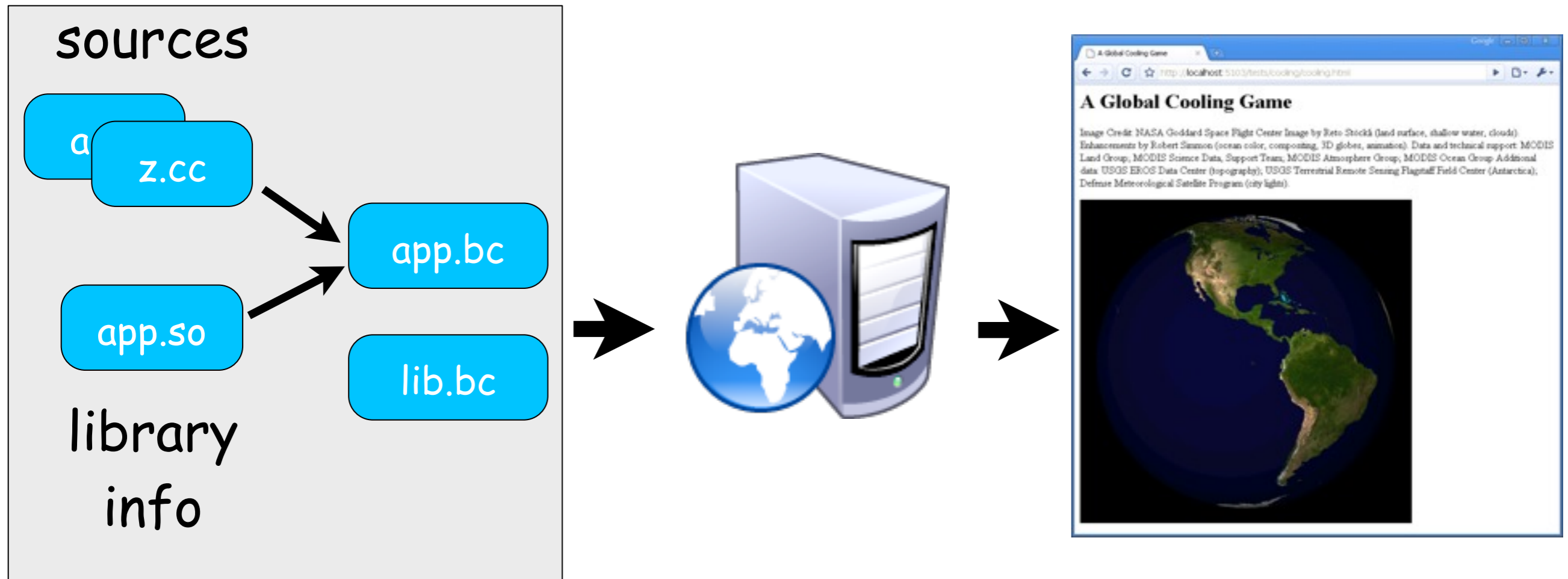


Only one porting effort

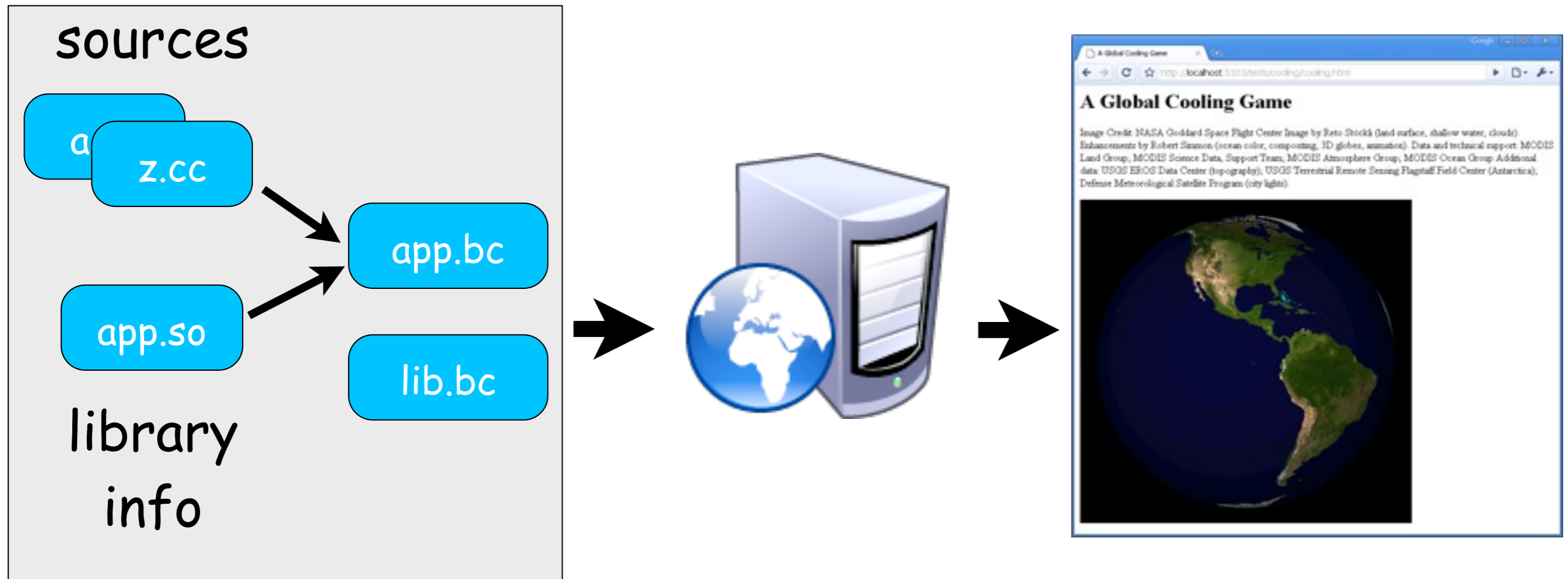
Approach

GOOGLE

Application Life Cycle



Application Life Cycle



Bitcode is PNaCl's distribution format

Client side



<http://myurl/myapp.bc>

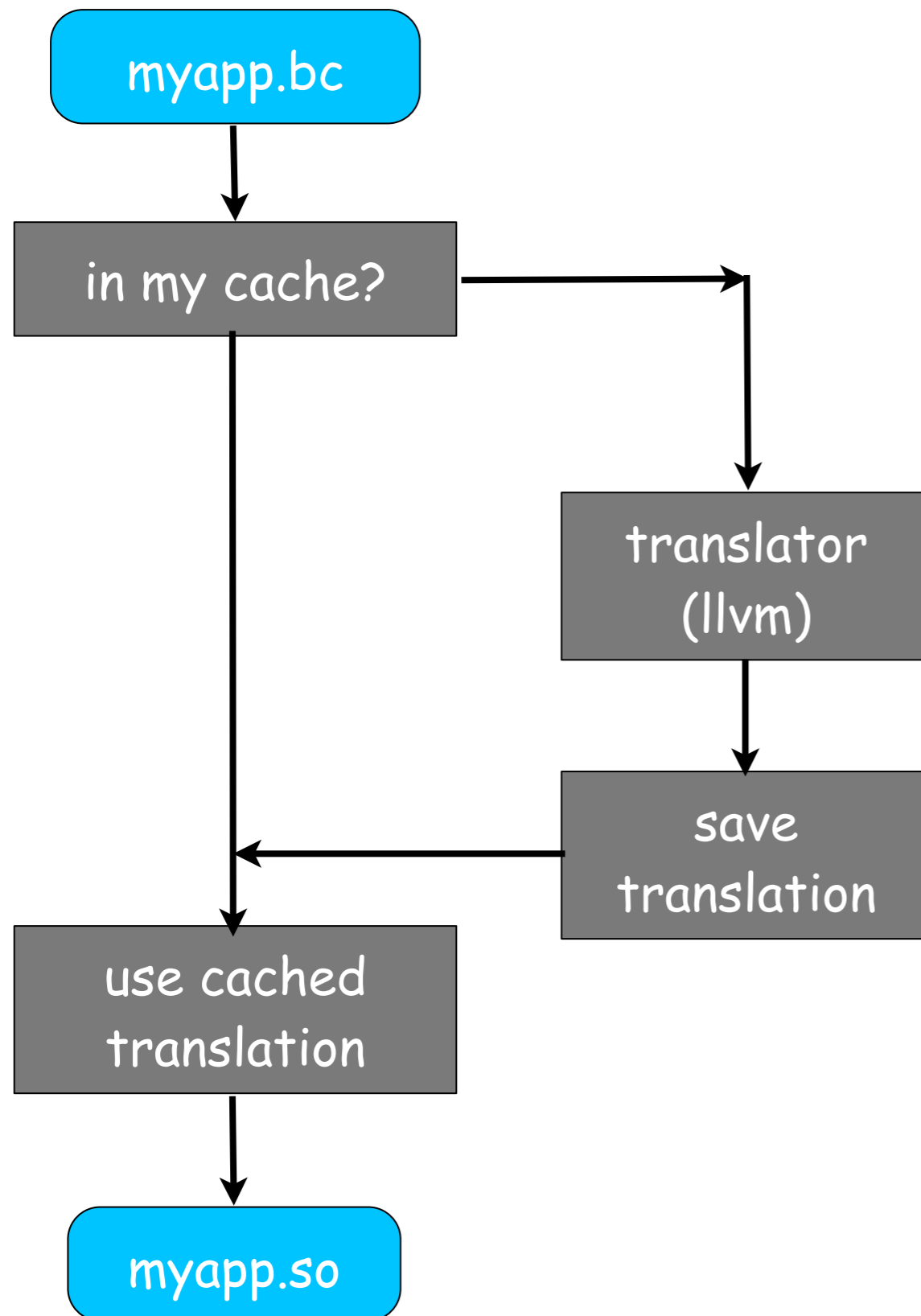
translation engine

[myapp.so](#)

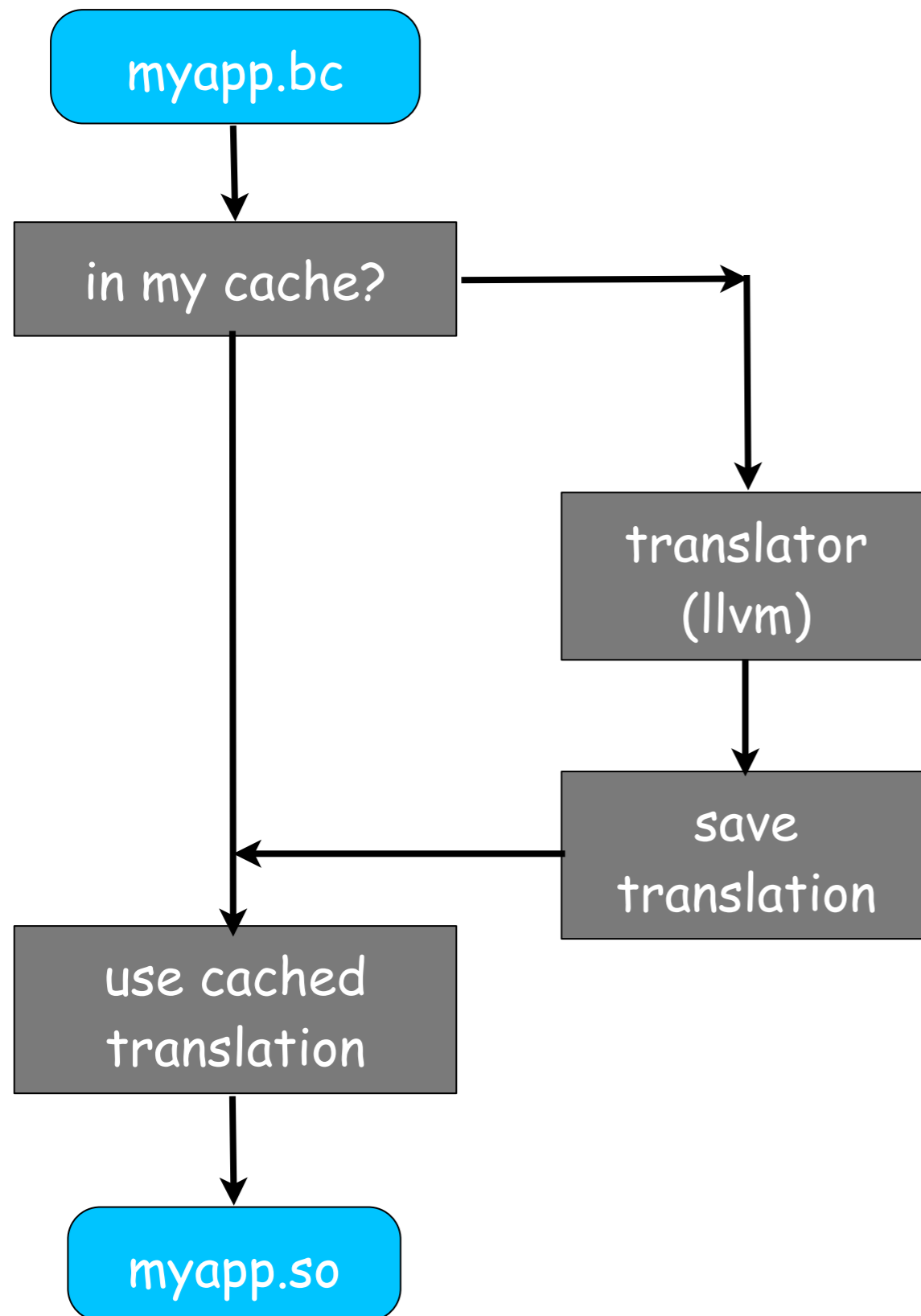
NaCl sandbox

ELF
x86, x64, or ARM

Translation Engine

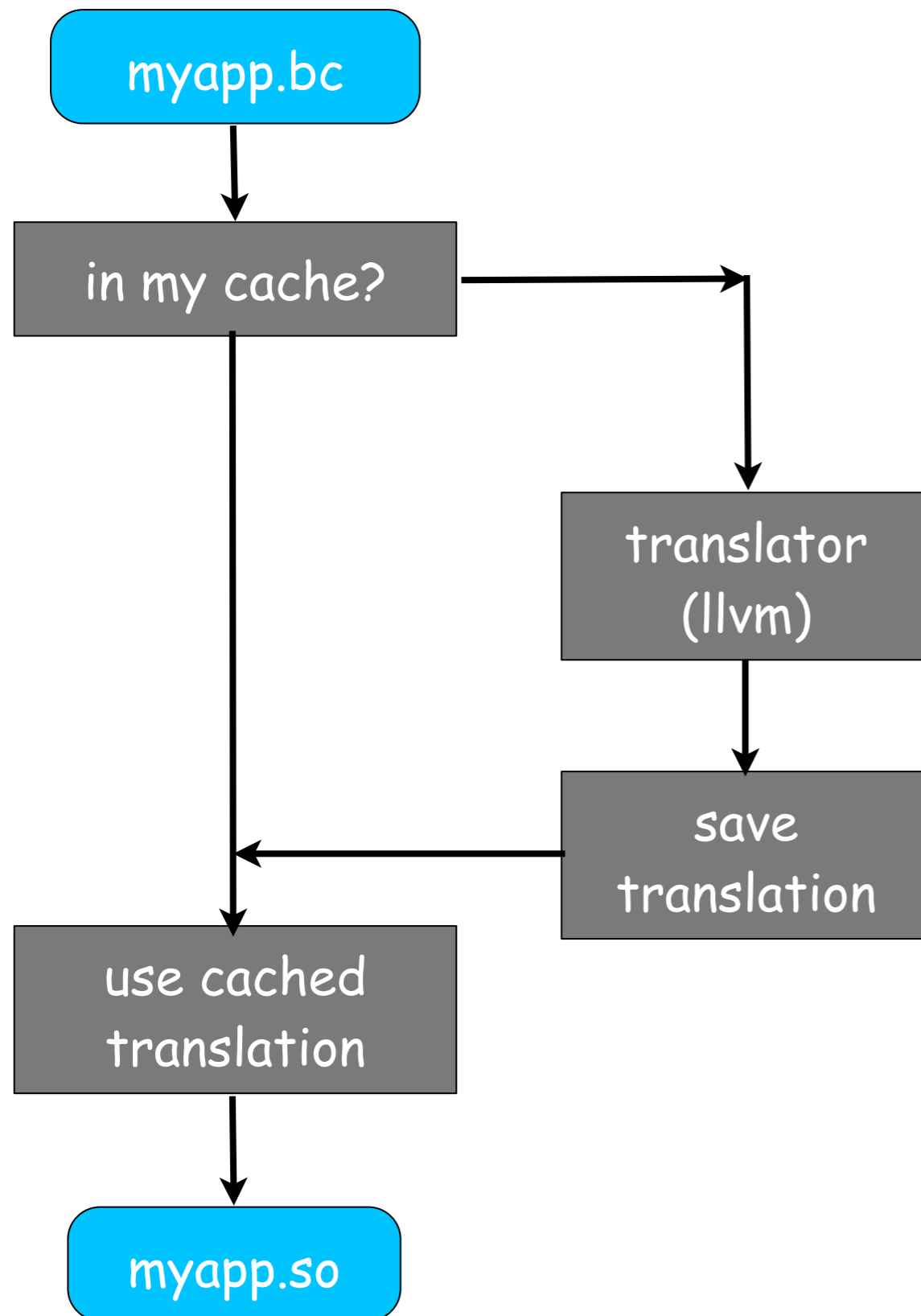


Translation Engine



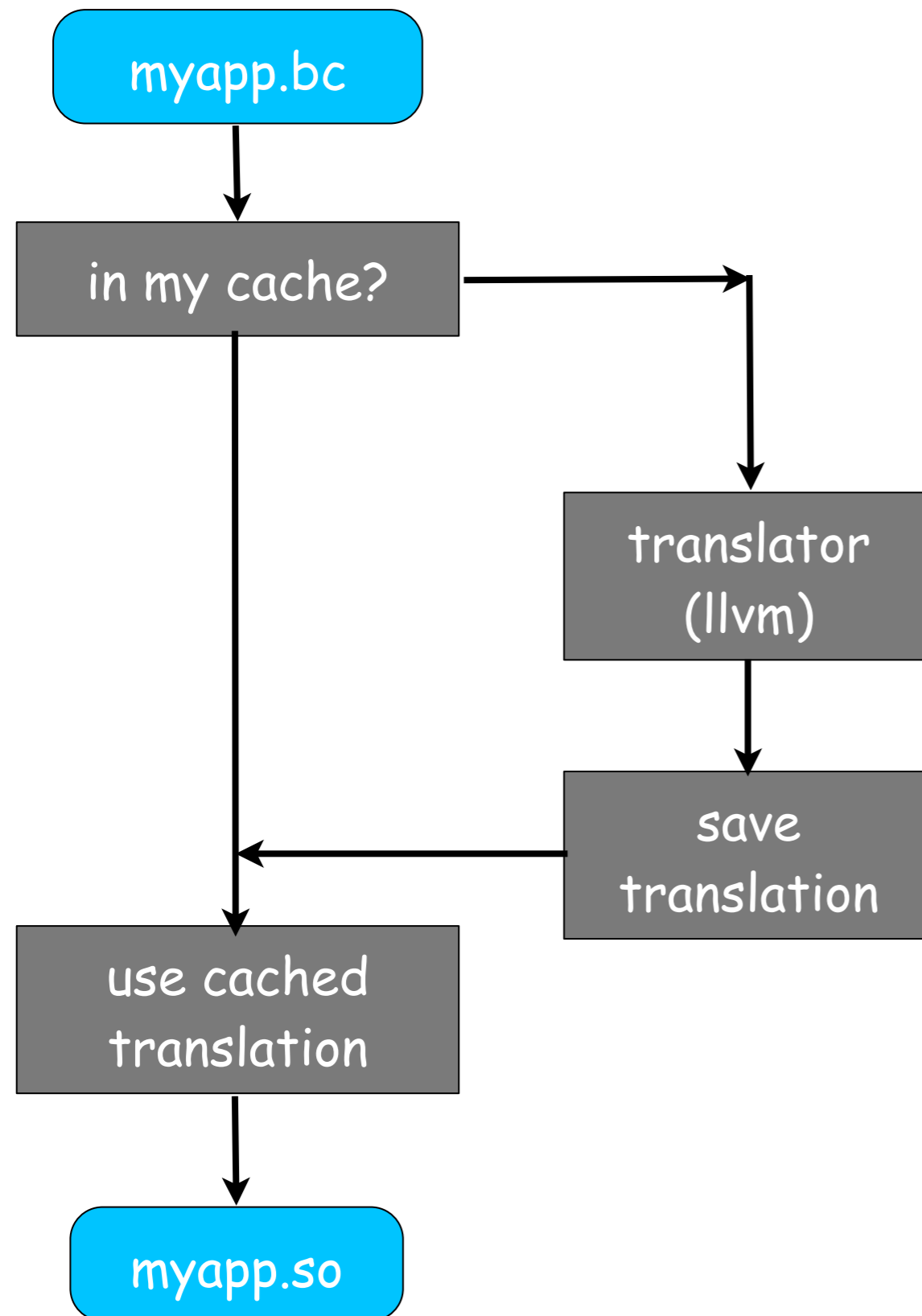
Know the platform (uarch)

Translation Engine



Know the platform (uarch)
Can collect/use profiling data
Webpage-specific
specialization

Translation Engine



Know the platform (uarch)
Can collect/use profiling data
Webpage-specific specialization
Can translate at invocation time
install time
asynchronously

Safe Translation

GOOGLE

Translating in a Sandbox



The translator must run in the browser

Malicious bitcode files are a potential attack vector

Translating in a Sandbox



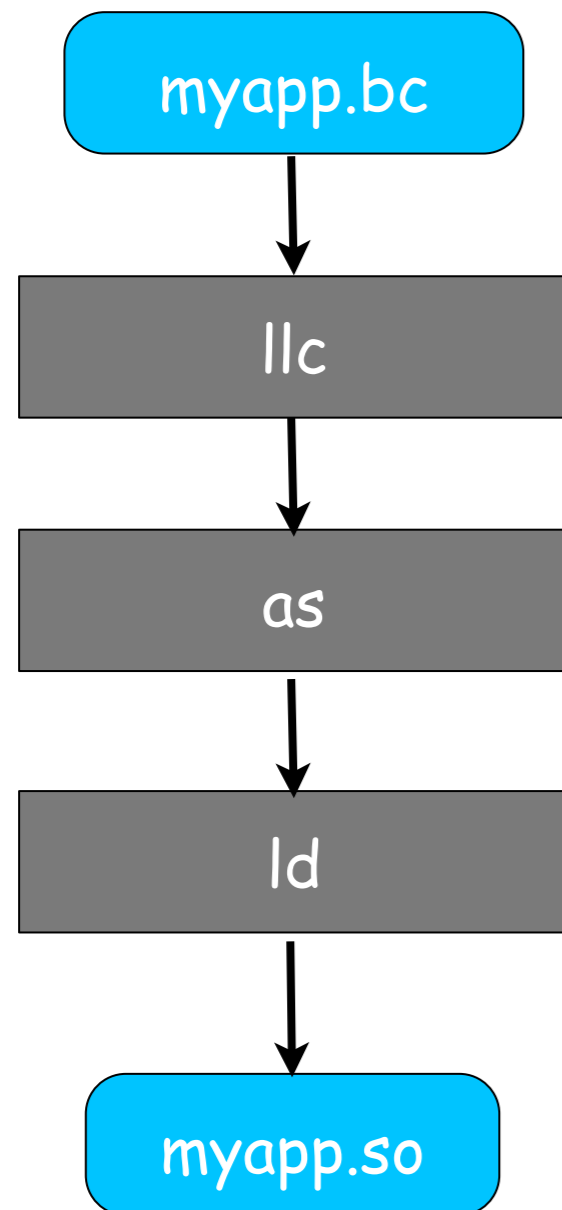
The translator must run in the browser

Malicious bitcode files are a potential attack vector

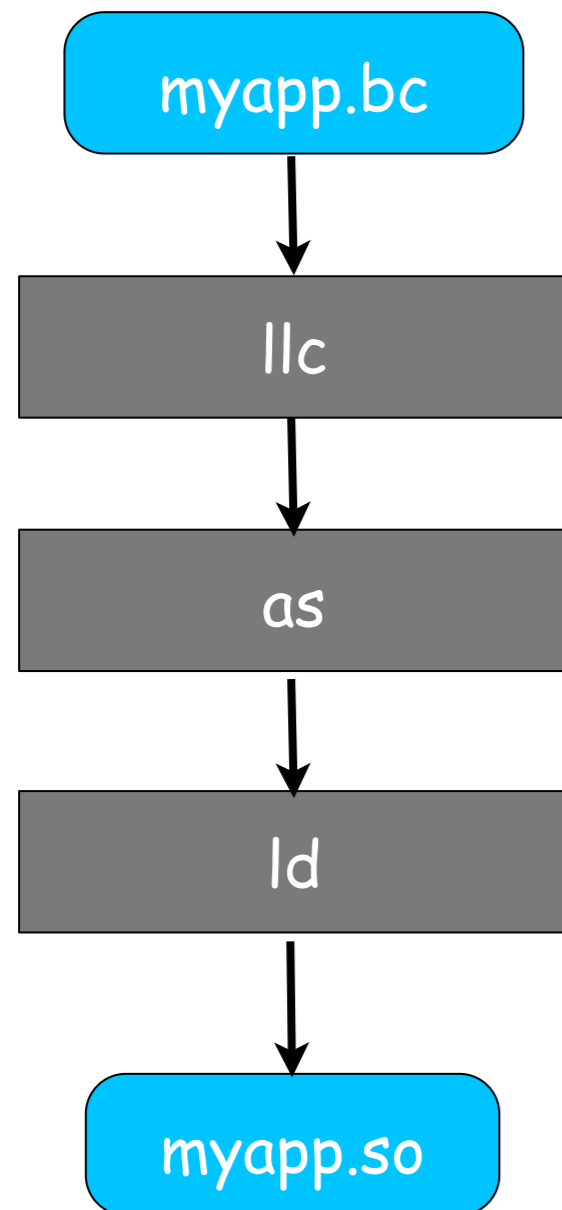
Translator phases are run as
NaCl modules

Translator

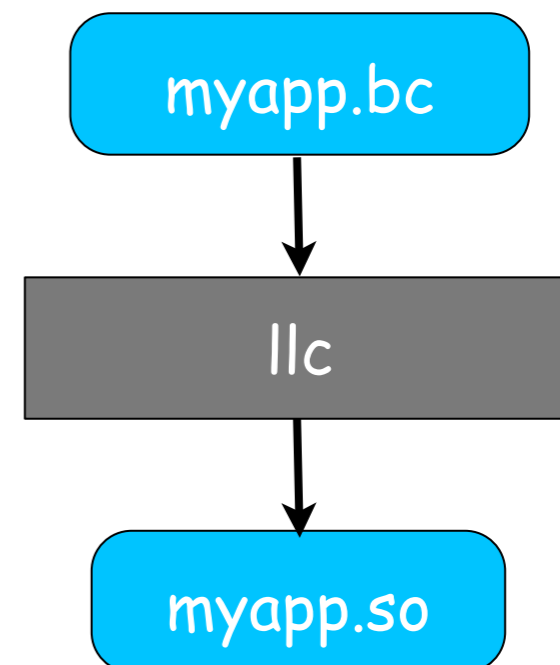
today



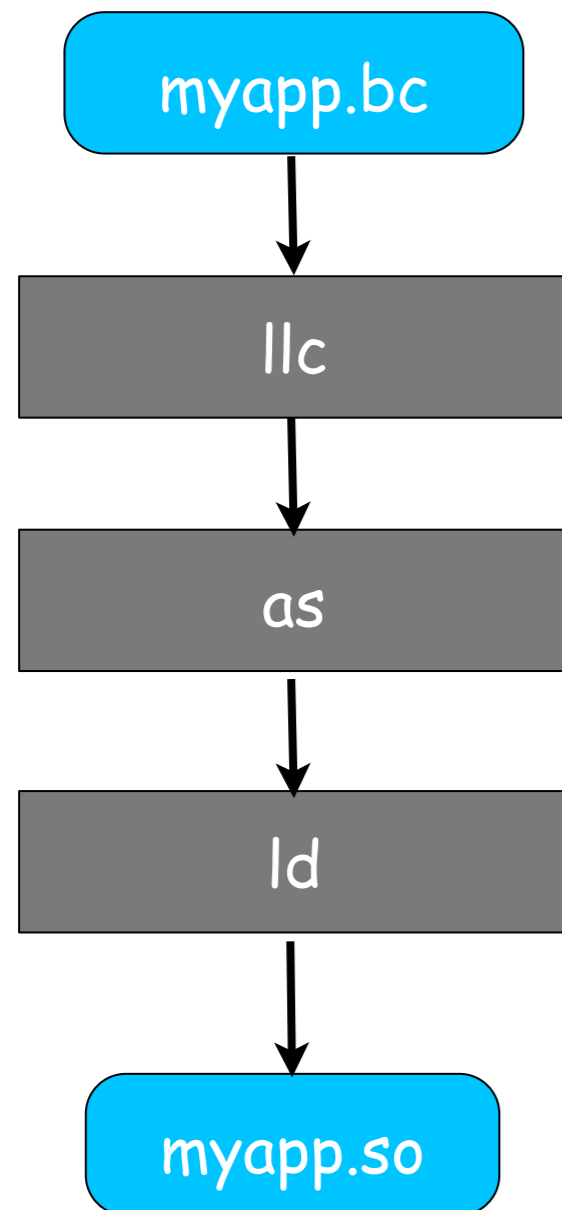
today



what we want

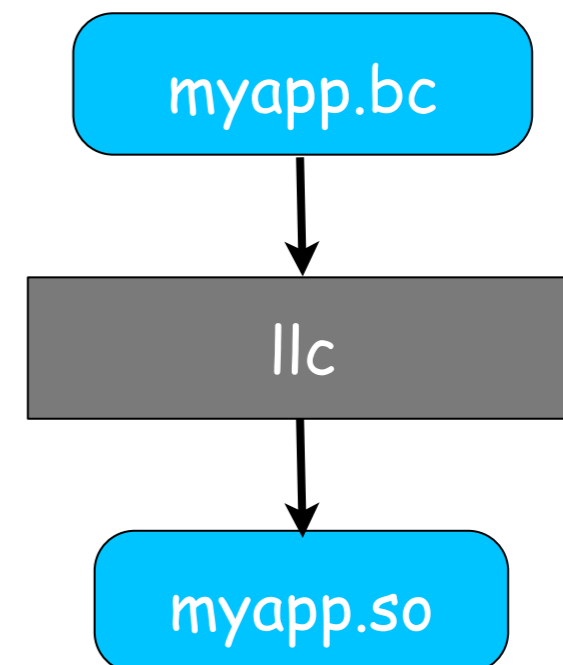


today



what we want

+ MC ELF
+ Bundling
+ DT_NEEDED



Bitcode as an Interchange Format



Target Model



Address space / data model

ILP32 ($\text{sizeof}(\text{int}) == \text{sizeof}(\text{long}) == \text{sizeof}(\text{void}^*)$)

$\text{sizeof}(\text{va_list}) == 24$

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 $0 \bmod 8$)

Byte order

Little Endian

Target Model



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)

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

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?

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?



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

lrc, 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

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

GOOGLE

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



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



Clang

Other languages that could target bitcode

.NET/Mono, ...

JIT support

Performance

feedback directed optimization, ...

Bitcode size

Translation time

Want to Learn More?



<http://www.chromium.org/nativeclient>

(Follow Portable Native Client link)

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