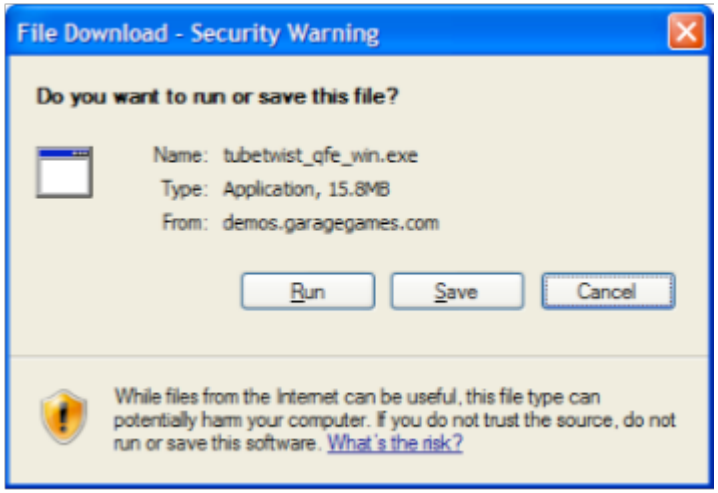
# Native Client: Safe, Portable Execution of Untrusted x86 Native Code

Brad Chen
January 2009

with Bennet Yee, David Sehr, Shiki Okasaka, Robert Muth, Greg Dardyk, Nicholas Fullagar, Neha Narula, Tavis Ormandy

# Safe Native Code for Web Apps

- Browser plugins deliver performance but sacrifice security
  - basic flaw: they require blind trust
  - exacerbated by weak OS security models




NativeClient enables native performance without sacrificing security



Google

# Native Client Goals

- Goals
  - Safe execution of untrusted x86-32 native code
  - Performance ~= native code
  - OS/browser portability
  - Trustworthy
    - Trusted code base is small and simple
    - Open source
  - Complement, not replace existing infrastructure
- Non-goals
  - execution of existing binary code (must recompile)
  - ISA portability



Google™

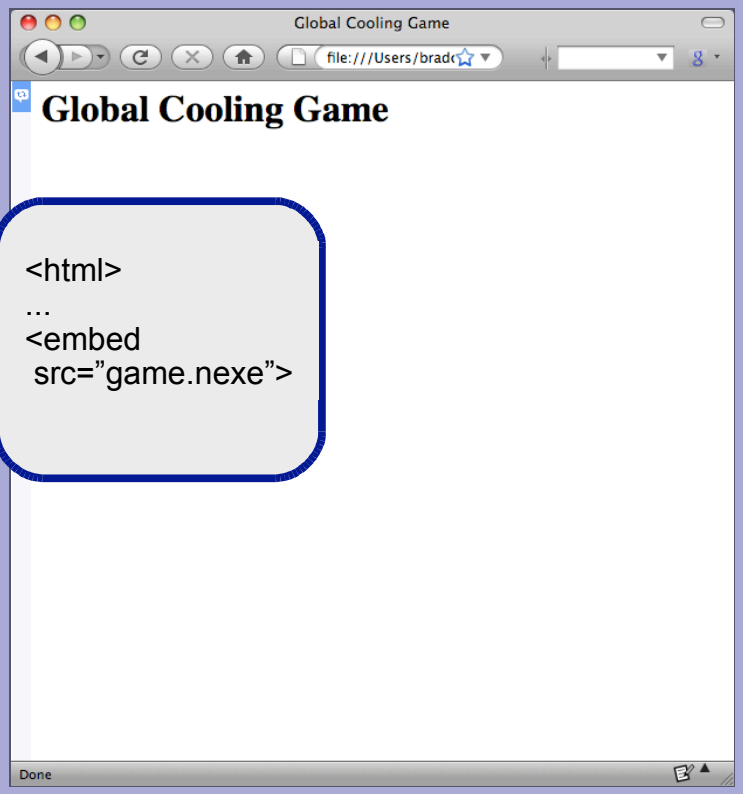
# A Quick Demonstration

# Life of a NaCl-enabled Web App




Google™

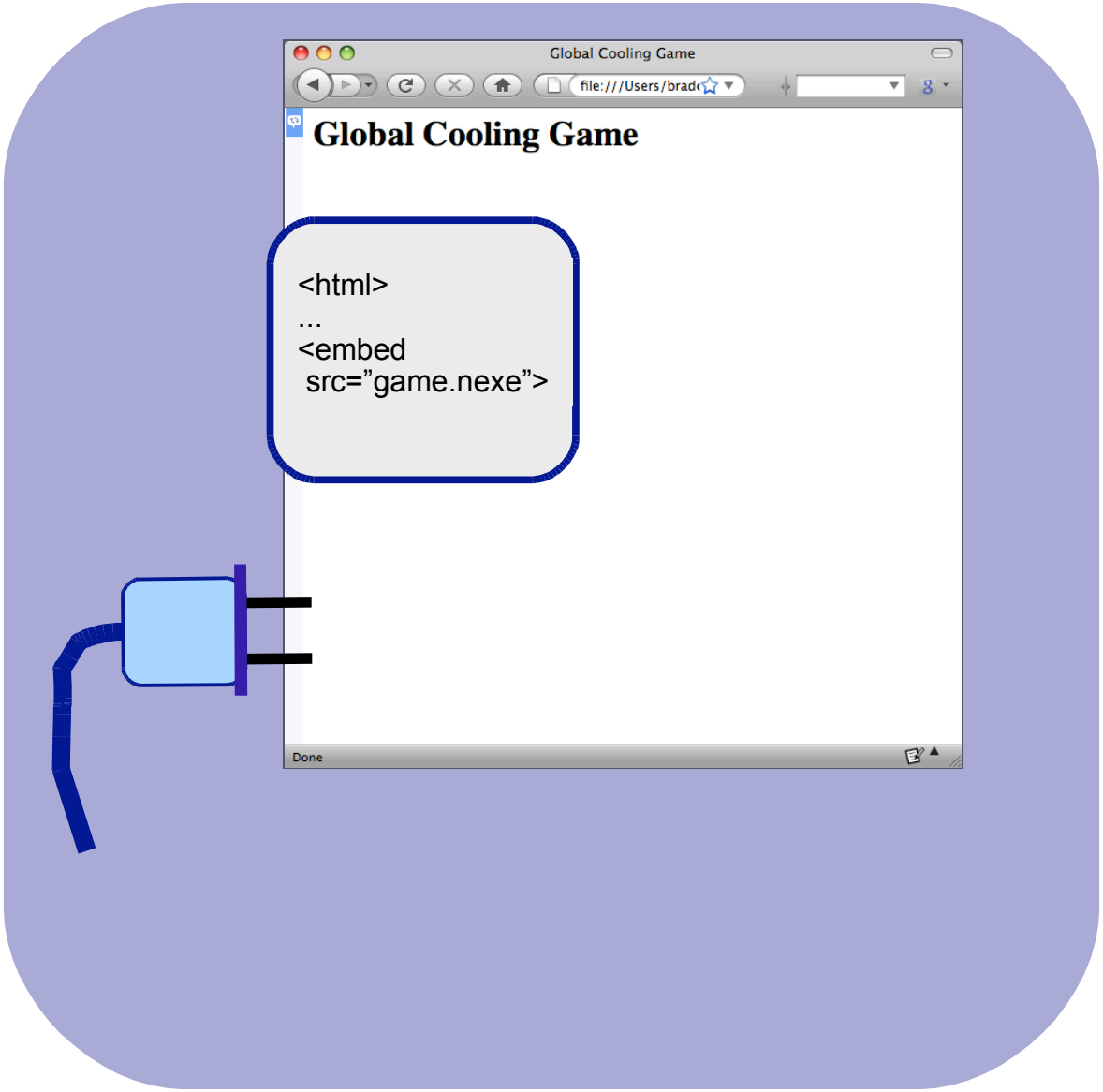
# Life of a NaCl-enabled Web App






Google™

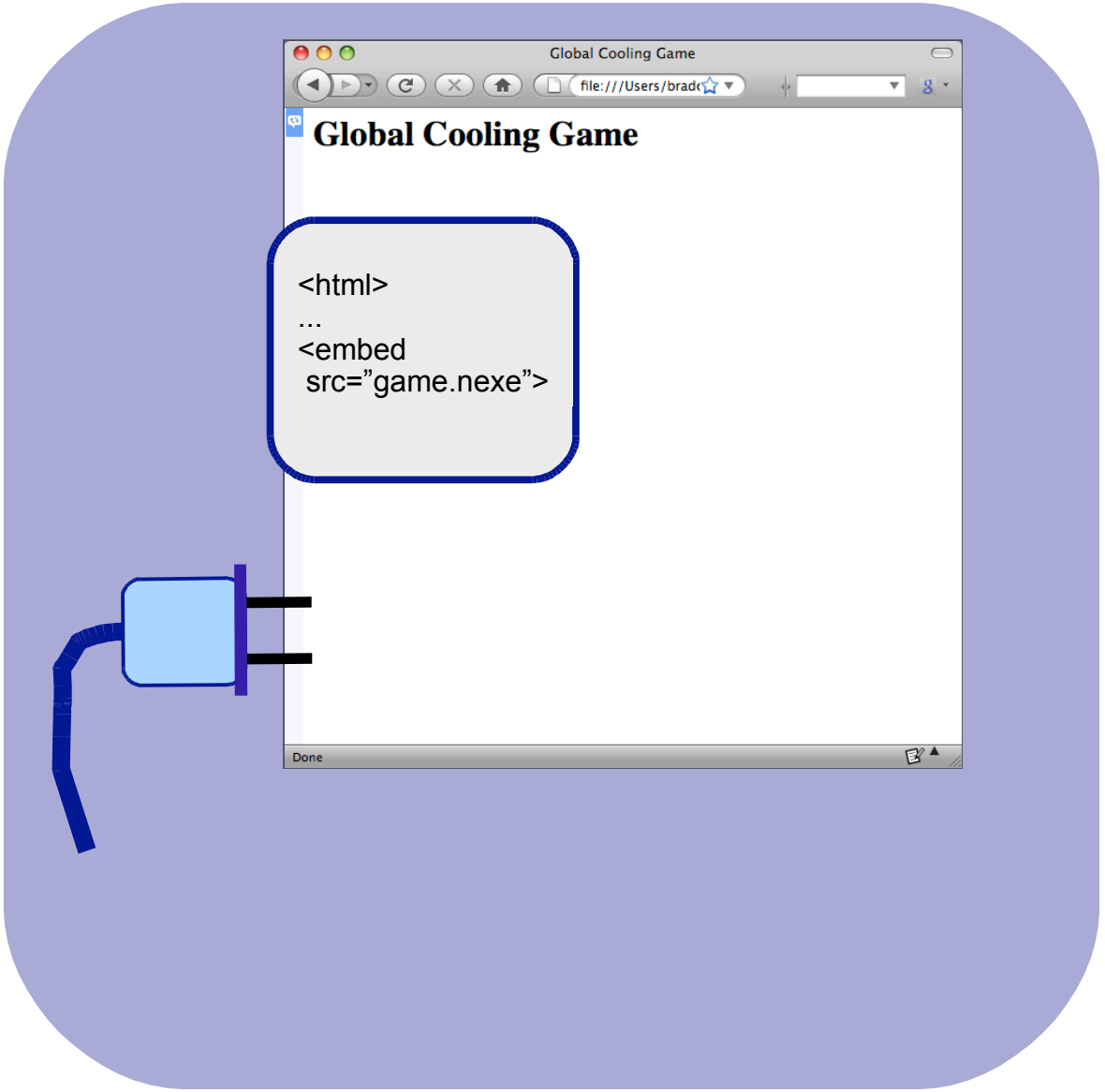
# Life of a NaCl-enabled Web App

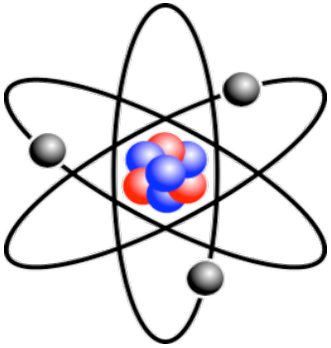





Google

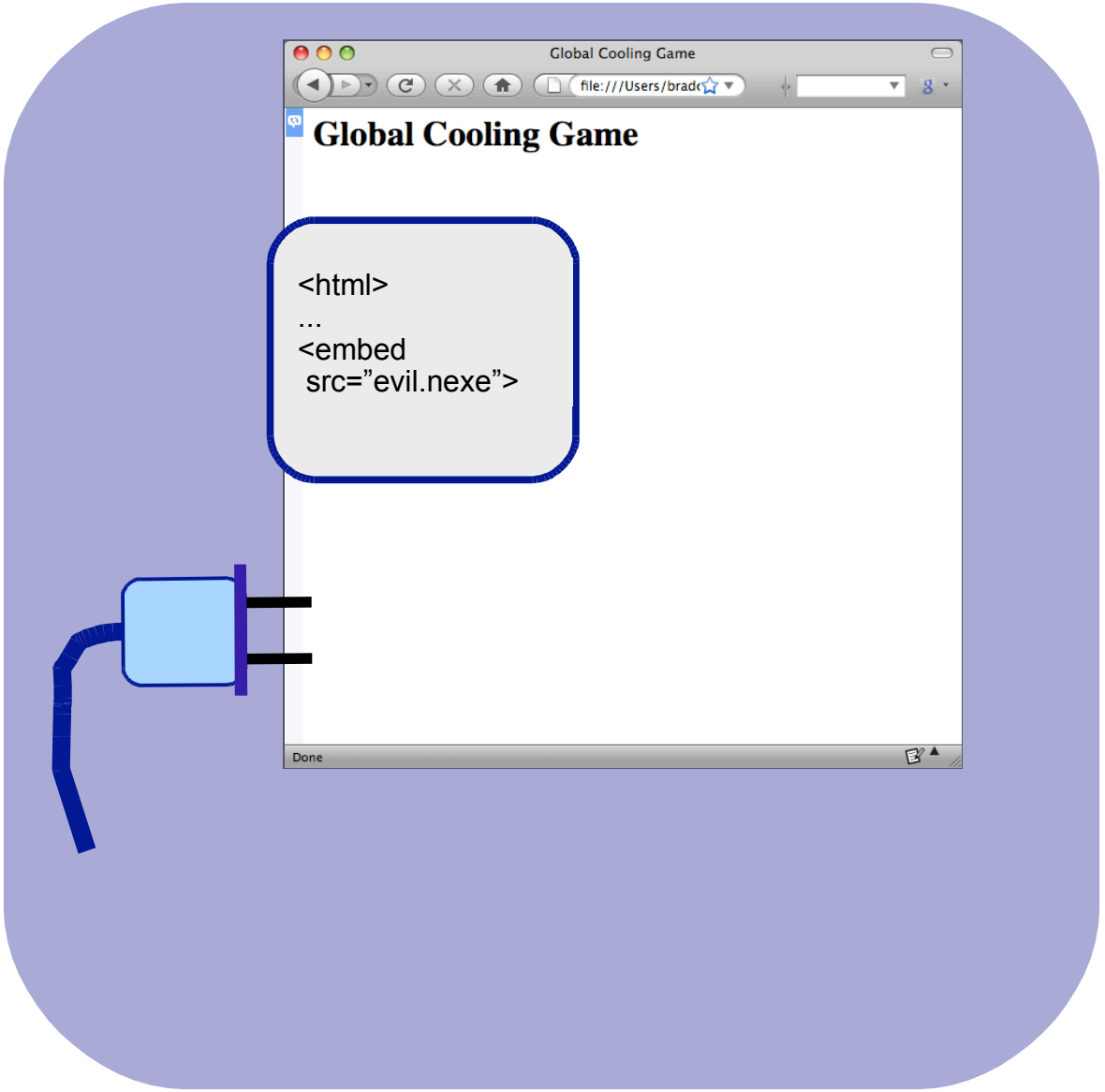
# Life of a NaCl-enabled Web App



Google™

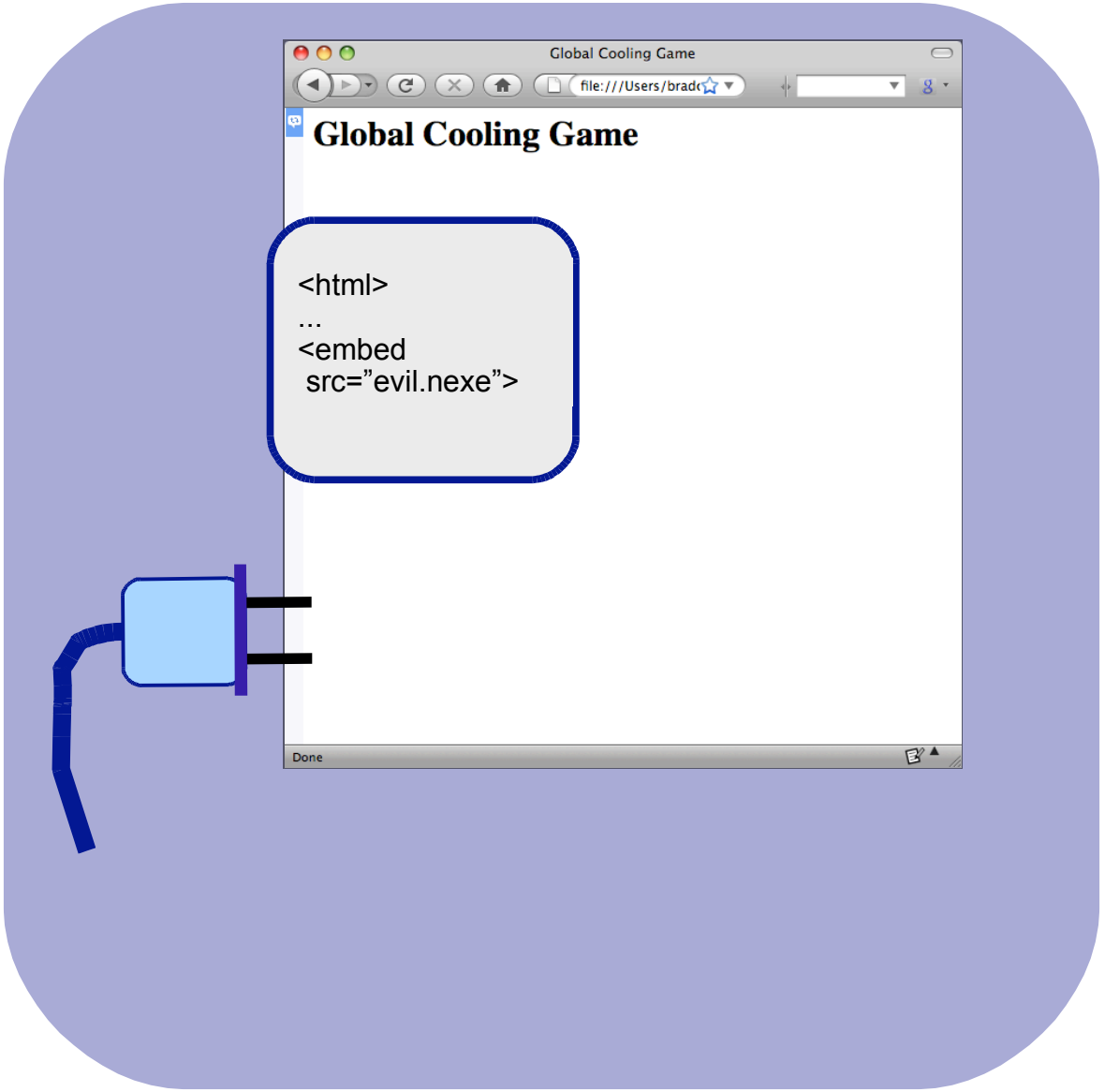
# Life of a NaCl-enabled Web App

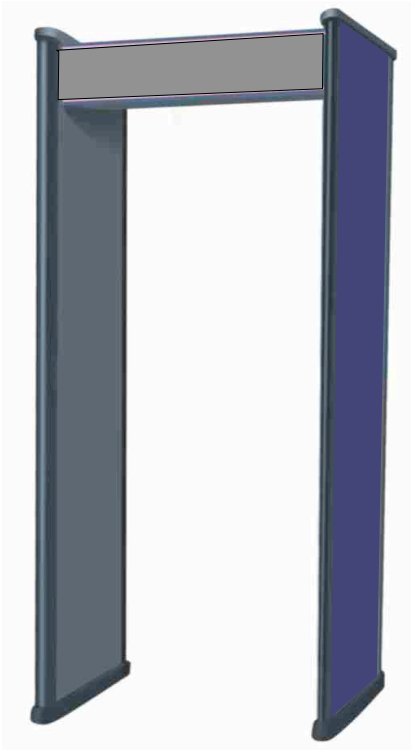





Google™

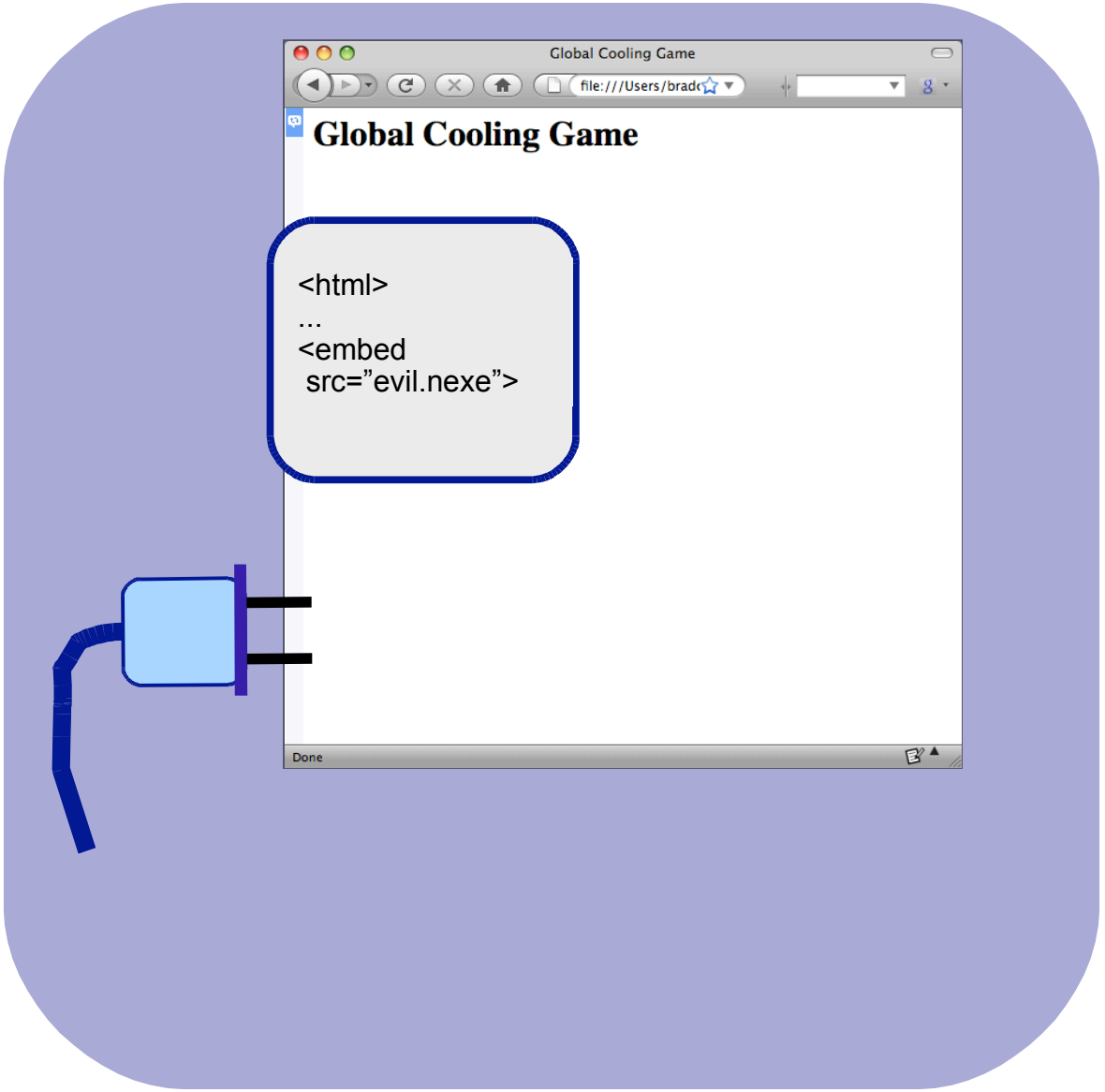
# Life of a NaCl-enabled Web App

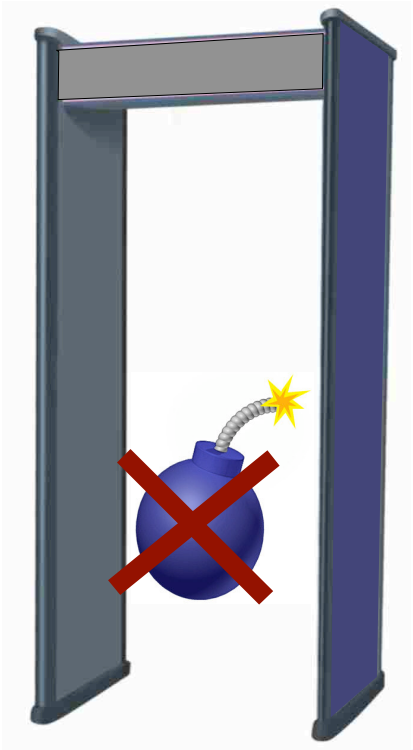


Google™

# Life of a NaCl-enabled Web App

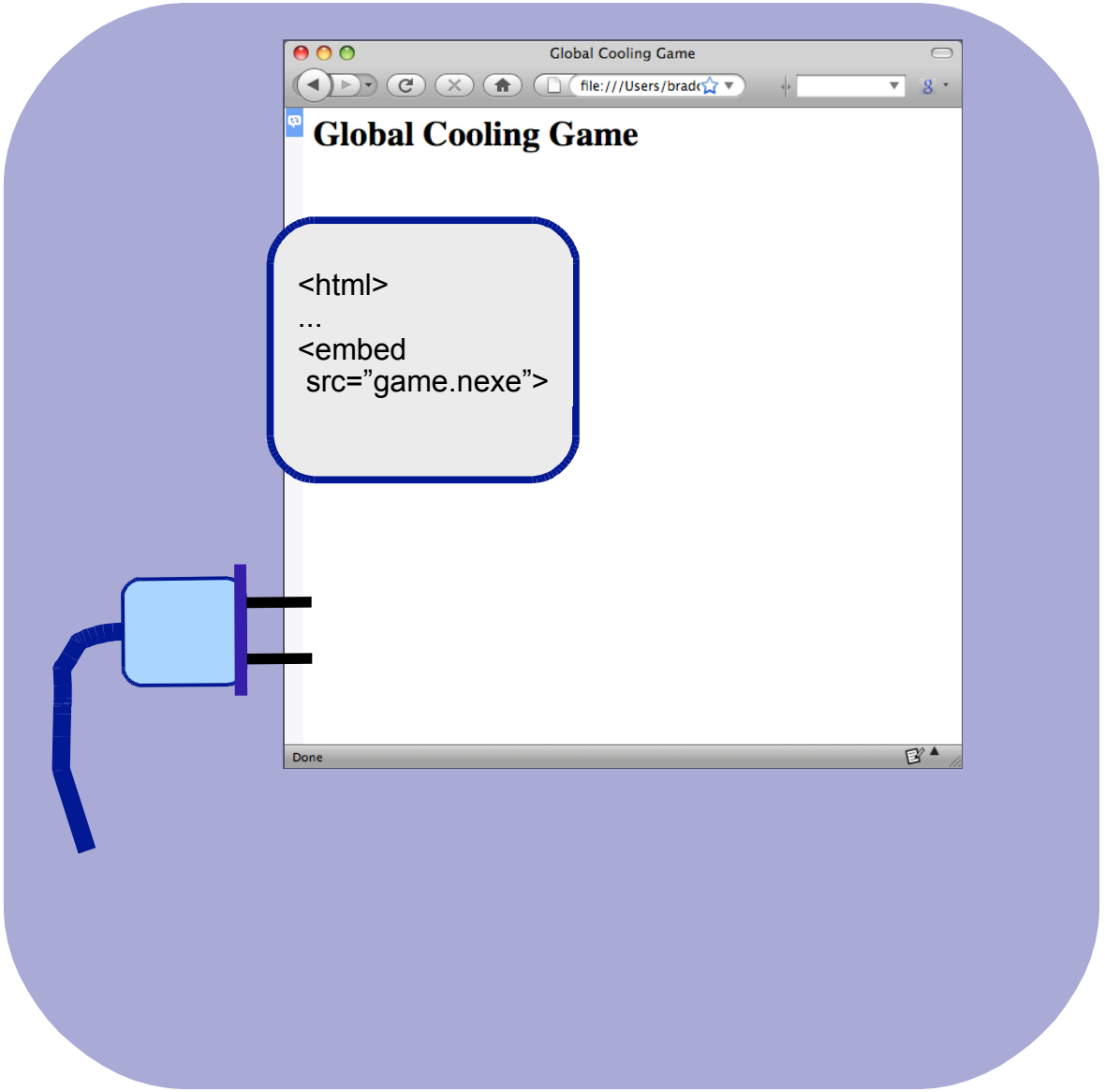


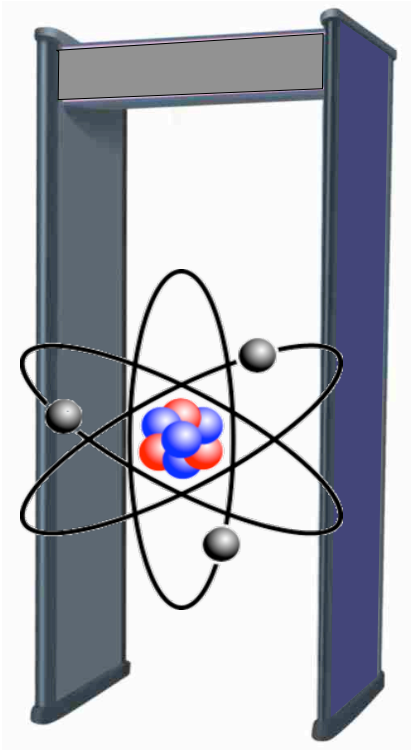





Google™

# Life of a NaCl-enabled Web App

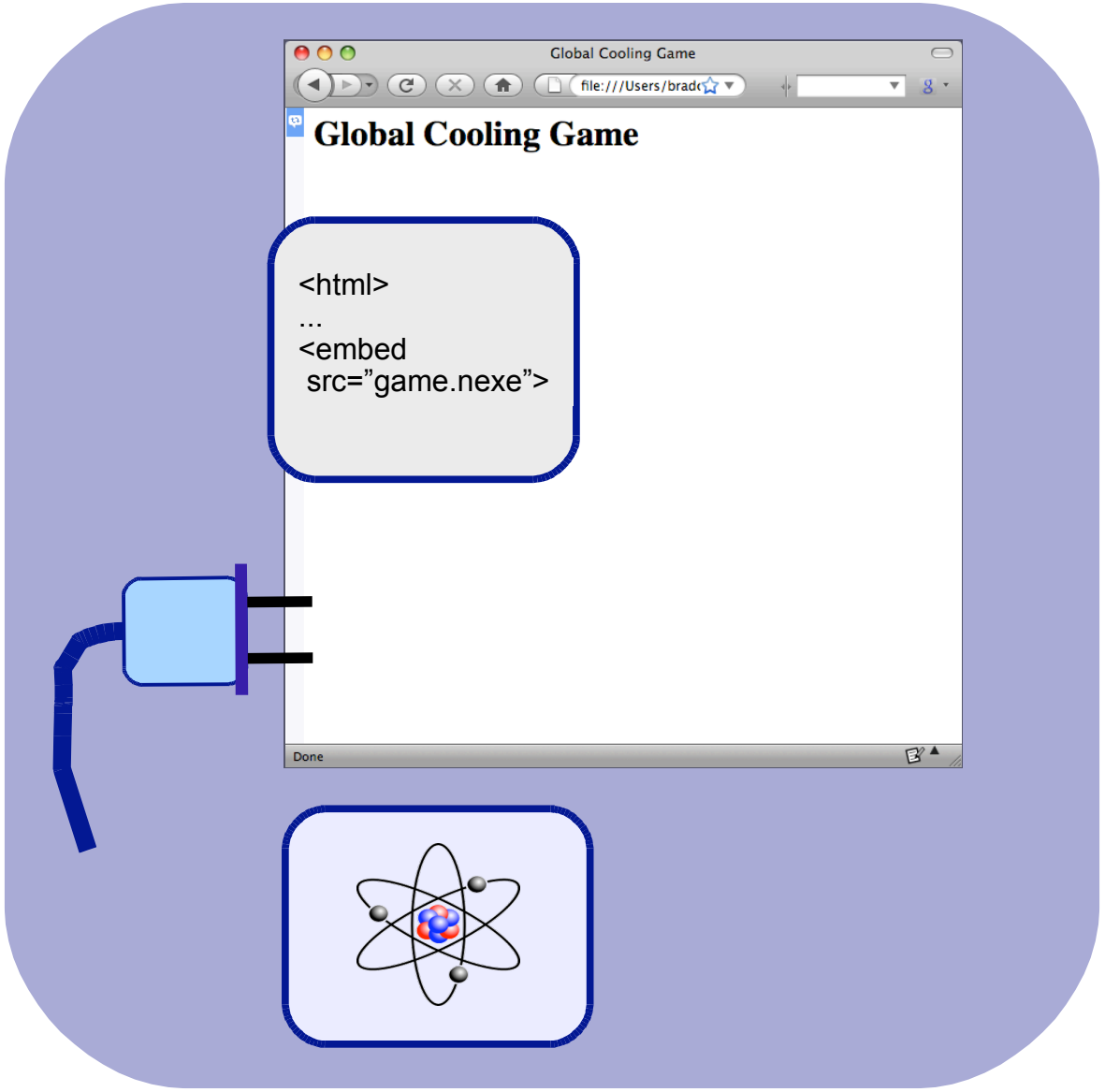




Google™

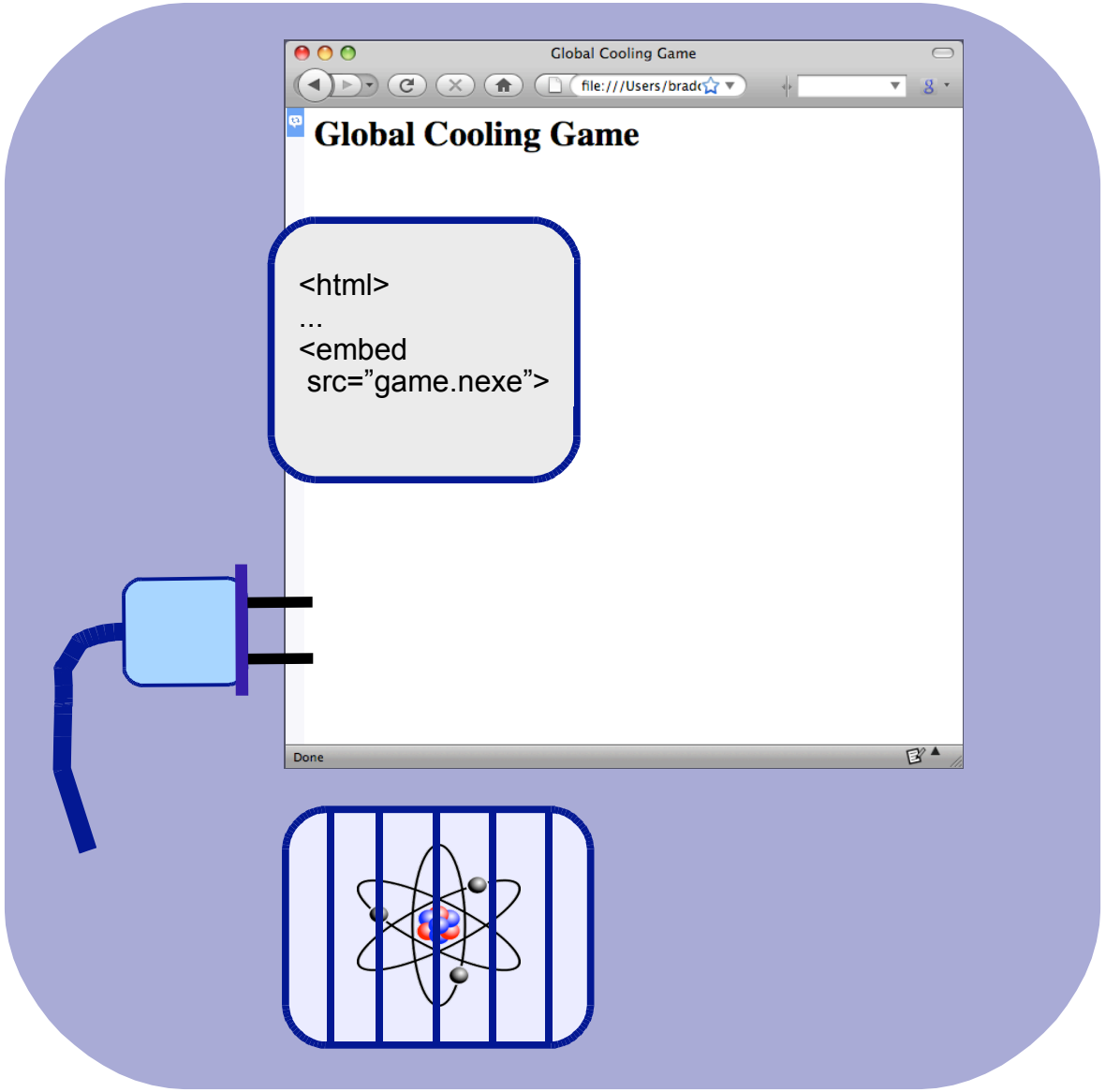
# Life of a NaCl-enabled Web App






Google™

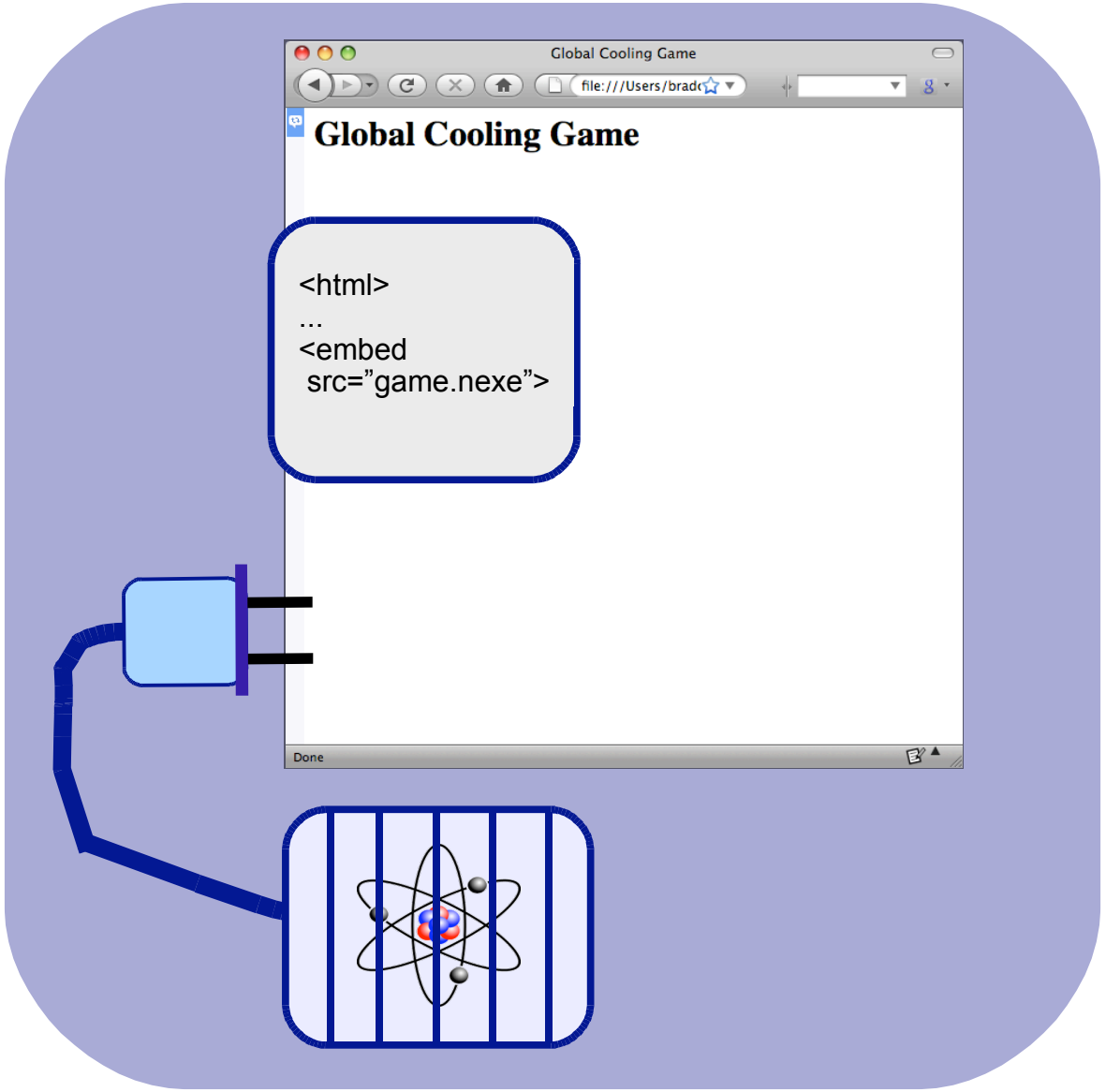
# Life of a NaCl-enabled Web App






Google™

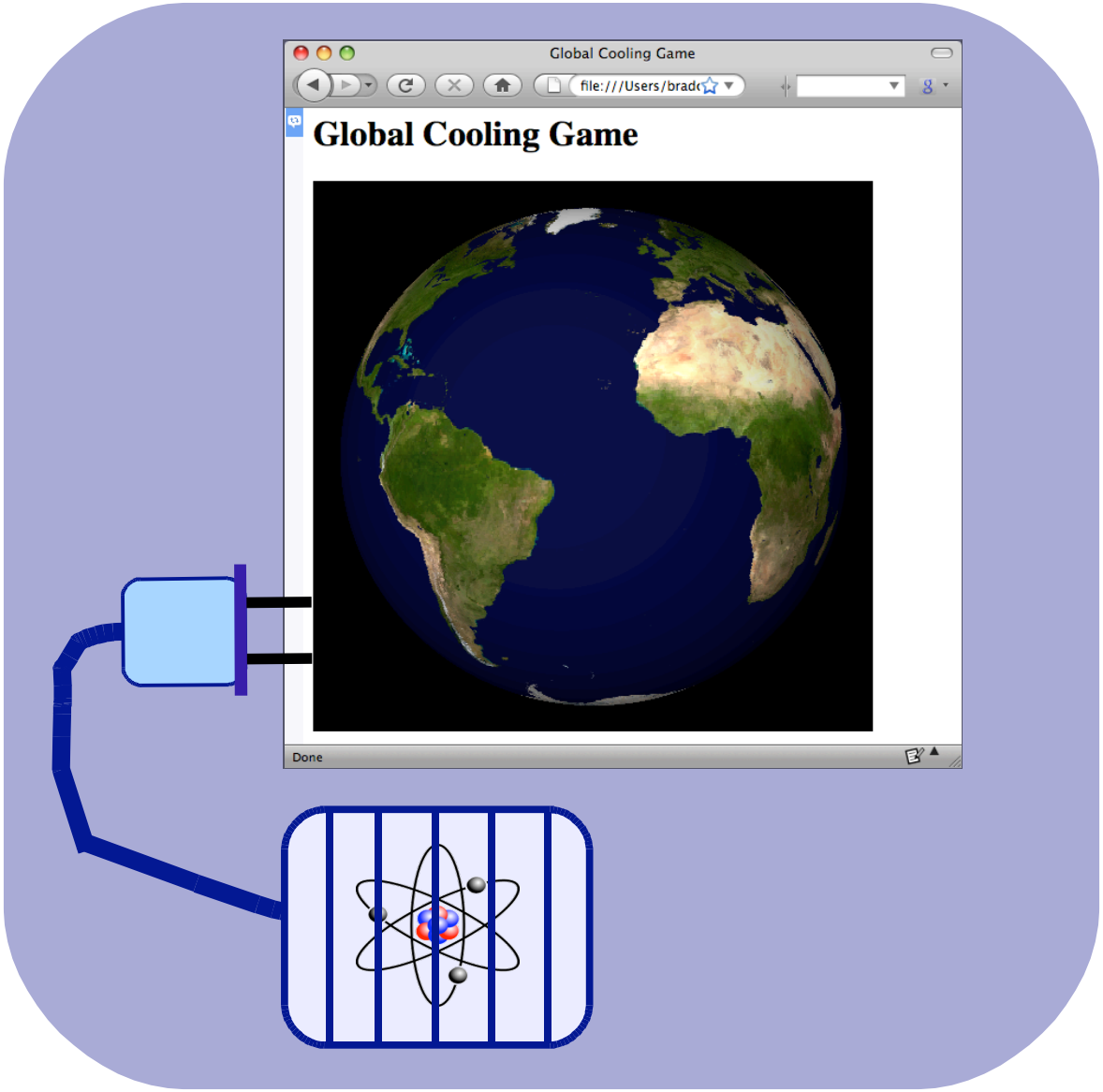
# Life of a NaCl-enabled Web App






Google™

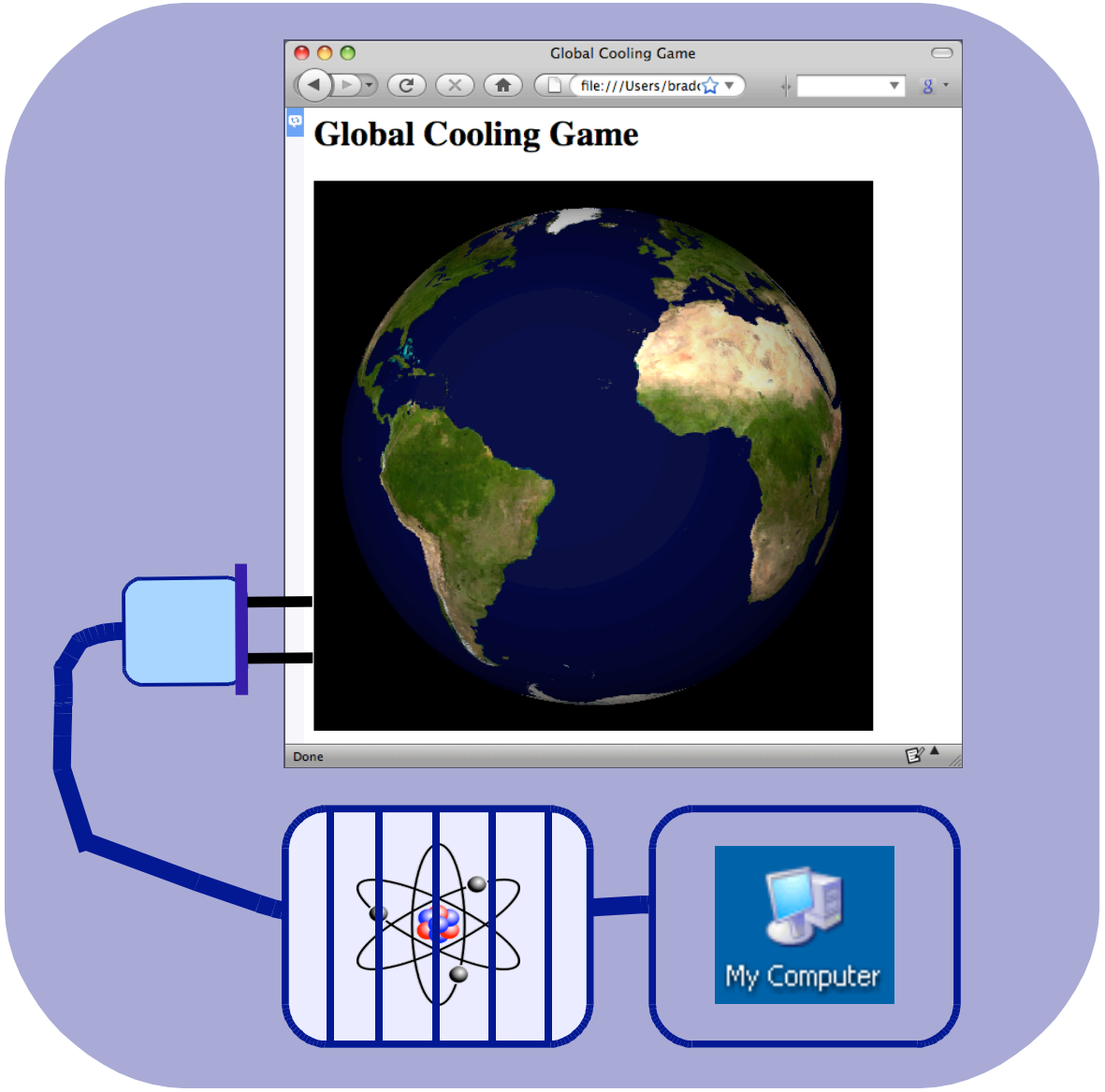
# Life of a NaCl-enabled Web App






Google™

# Life of a NaCl-enabled Web App






Google™

# Q: What does “safe” mean?

Q: What does “safe” mean?

A: No side effects except via explicit secure interfaces.

# Example 1: An Evil System Call

```
void DoEvil() {
  char *eargv[] = {"/bin/rm", "-rf", "/home/*", NULL};
  int rc = execve(eargv[0], eargv, NULL);
}
```

NaCl modules are not allowed to directly execute system calls or other 'special' instructions.

Variants

- dynamically generated code
- code embedded in data
- overlapping instructions

# Example 1a: An Evil System Call

```
// MacOS version
int TrustMe(int returnaddr1, const char *path,
            char *const argv[], char *const envp[]) {
  int immx = 0x0000340f;     // 0f 34 is syscall inst.
  int codeaddr = 14 + (int)TrustMe;

  asm("mov   $59, %eax");    // set syscall # for execve
  asm("add   $32, %esp");    // pop local storage
  asm("mov   %esp, %ecx");   // kernel wants esp in ecx
  asm("jmp   *-20(%ecx)");   // jump to syscall inst
                             // via pointer in codeaddr
}

#define NULL 0
char *const eargv[] = {
     "/bin/rm", "-rf", "/home/*", NULL};
int main(int argc, char *argv[]) {
  TrustMe(-1, eargv[0], eargv, NULL);
}
```

# Example 1a: An Evil System Call

```
// MacOS version
int TrustMe(int returnaddr1, const char *path,
            char *const argv[], char *const envp[]) {
  int immx = 0x0000340f;    // 0f 34 is syscall inst.
  int codeaddr = 14 + (int)TrustMe;

  asm("mov   $59, %eax");   // set syscall # for execve
  asm("add   $32, %esp");   // pop local storage
  asm("mov   %esp, %ecx");  // kernel wants esp in ecx
  asm("jmp   *-20(%ecx)");  // jump to syscall inst
                            // via pointer in codeaddr
}

#define NULL 0
char *const eargv[] = {
     "/bin/rm", "-rf", "/home/*", NULL};
int main(int argc, char *argv[]) {
  TrustMe(-1, eargv[0], eargv, NULL);
}
```

# Example 2: A Wild Write

```
char* magicp = FindReturnAddrOnIEStack();
*magicp = (char *)&DoEvil;
```

- NaCl modules should not be able to damage other computations on the system
- Assumption: operating system is not cooperative
  - Basic requirement for portability
  - Must constrain writes to known safe memory

# Example 3: A Devious Read

```
char *creditcardptr = 0x38a390;
GoShoppingWithFriends(creditcardptr);
```

- NaCl modules should only be allowed to read data inside their sandbox.

# Catalog of Threats

Primary Threats

- Special instructions
- Wild control flow
- Wild writes
- Wild reads

Secondary Threats

- Infinite loops
- Memory leaks
- Corrupt output
- Covert channels

# Catalog of Threats

Primary Threats

- Special instructions
- Wild control flow
- Wild writes
- Wild reads

Secondary Threats

- Infinite loops
- Memory leaks
- Corrupt output
- Covert channels

- Code Integrity
- Control Flow Integrity
- Data Integrity

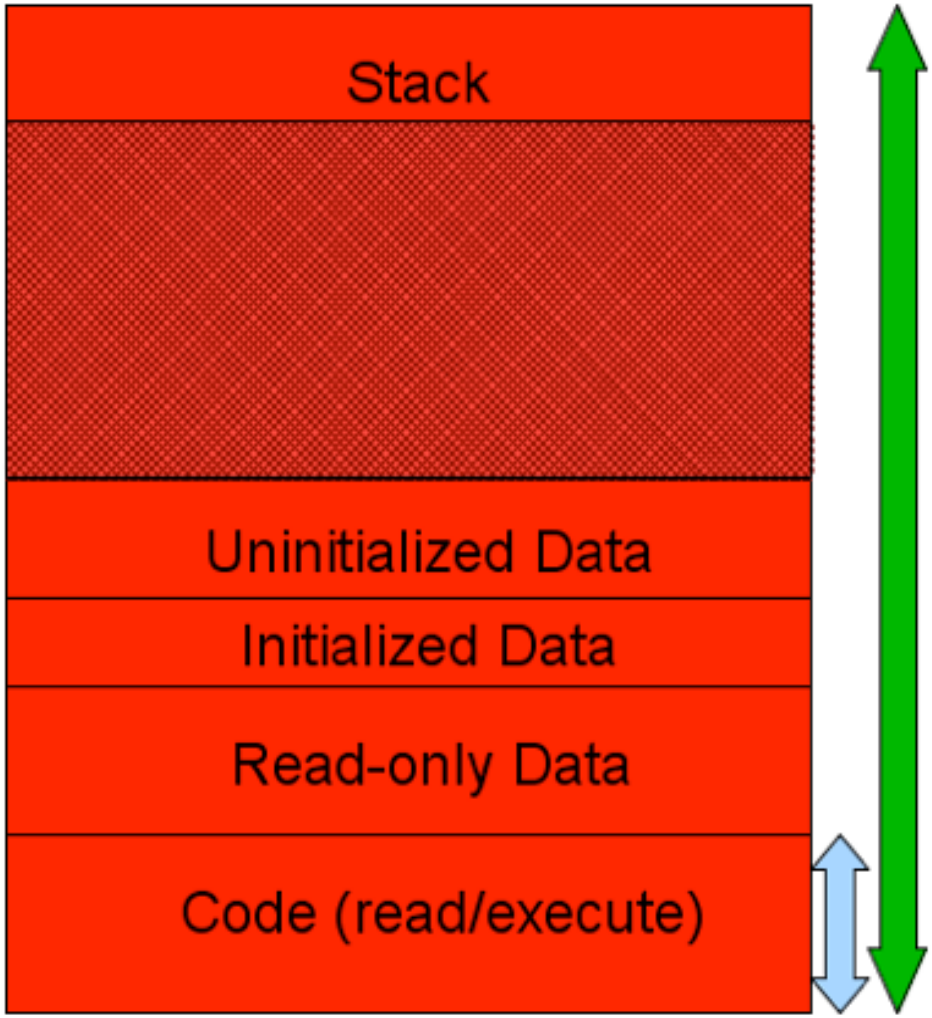
# Outline

- Introduction
- Background
- Eliminating Side Effects
  - Inner Sandbox
  - Outer Sandbox
- Allowing Side-Effects Safely
- Evaluation

# Inner Sandbox

- Software Fault Isolation + x86 segments
- No OS dependencies
  - portable
  - relatively immune to OS defects
  - caveat: LDT initialization
- Uses modified compiler, assembler and loader
  - modifications are simple and open
  - details follow
- Runtime overhead is very small

# Data/Memory Integrity




NaCl's address space is bounded by x86 segments:

- `%cs` allows execution only in code segment
- `%ds`, `%es`, `%ss` restrict data address space
- `%gs` used for thread descriptor



Google

# NaCl Control Flow Integrity Rules

Making x86 disassembly reliable:

- All valid instruction boundaries are determined by a top-to-bottom disassembly of text segment
- All aligned memory addresses must be a valid instruction boundary
- Computed jumps use sandboxing to force alignment
- Branch targets are within code segment (dynamically enforced via segmentation)
- Forbidden: mixed text/data, overlapping instructions, dynamic code generation



Google

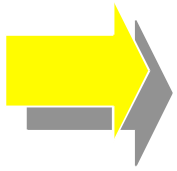
# Control Flow Integrity

- Static validation of most calls and jumps

```
call   8ef40
```

- Sandboxed returns and indirect calls/jumps
  - NaCl text must be aligned to *n*-byte blocks
  - All indirect branch targets must be aligned

```
naclcall   3830(%eax)
```



```
lea   %eax, 3830(%eax)
and   %eax, 0xffffffe0
call %eax
```



Google

# NaCl ISA/ABI Restrictions

Given reliable disassembly, NaCl can enforce:

- Omitted instructions: **`ret`**, **`int`**, **`syscall`** ... most x86-64 omissions
- Restricted prefix usage
  - At most one prefix byte per instructions
  - Segment, addr16 prefixes disallowed
  - Prefix usage restricted to known useful prefixes
- However, ISA extensions generally supported
  - unsupported instructions overwritten HLT

# The Most Recent Validator Bug

```
...
83 e2 e0    and 0xffffffe0,%eax
0f e2       jmp (%eax)     ;; okay
...
83 e2 e0    and 0xffffffe0,%eax
0f 12       jmp (*%eax)    ;; not okay
```

# Outer Sandbox

# Outer Sandbox

- System call filter for untrusted modules
- Redundant: “defense-in-depth”
- OS-dependent
- Implementation
  - Linux: ptrace
  - MacOS: sandbox.h
  - Windows: system call ACLs

# Outline

- Introduction
- Background
- Eliminating Side Effects (SFI)
- Allowing Side-Effects Safely
- Evaluation

# NaCl Runtime Environment

- IMC
  - reliable datagram service
  - handles: shared memory, descriptors
- POSIX subset
  - operations on descriptors
  - memory management
- pthreads, synchronization, TLS
- multimedia: video, audio, events

# NaCl Runtime: Interesting features

- Shared memory for communication into/out of untrusted modules
- Calls out to trusted code via 'trampoline' and 'springboard' mechanism
- Uses a separate trusted stack
- All hardware exceptions are fatal
  - C++ catch/throw exceptions are okay
- Calls into untrusted code via simple RPC system, with constrained data types:
  - int float string array
  - opaque “handle”; no pointers

# Attack Surface

- inner sandbox
  - binary validation
  - hardware errata
- outer sandbox: OS system-call interception
- binary module loader
- trampoline interfaces
- IMC communications interface
- NPAPI interface
  - will likely be further constrained

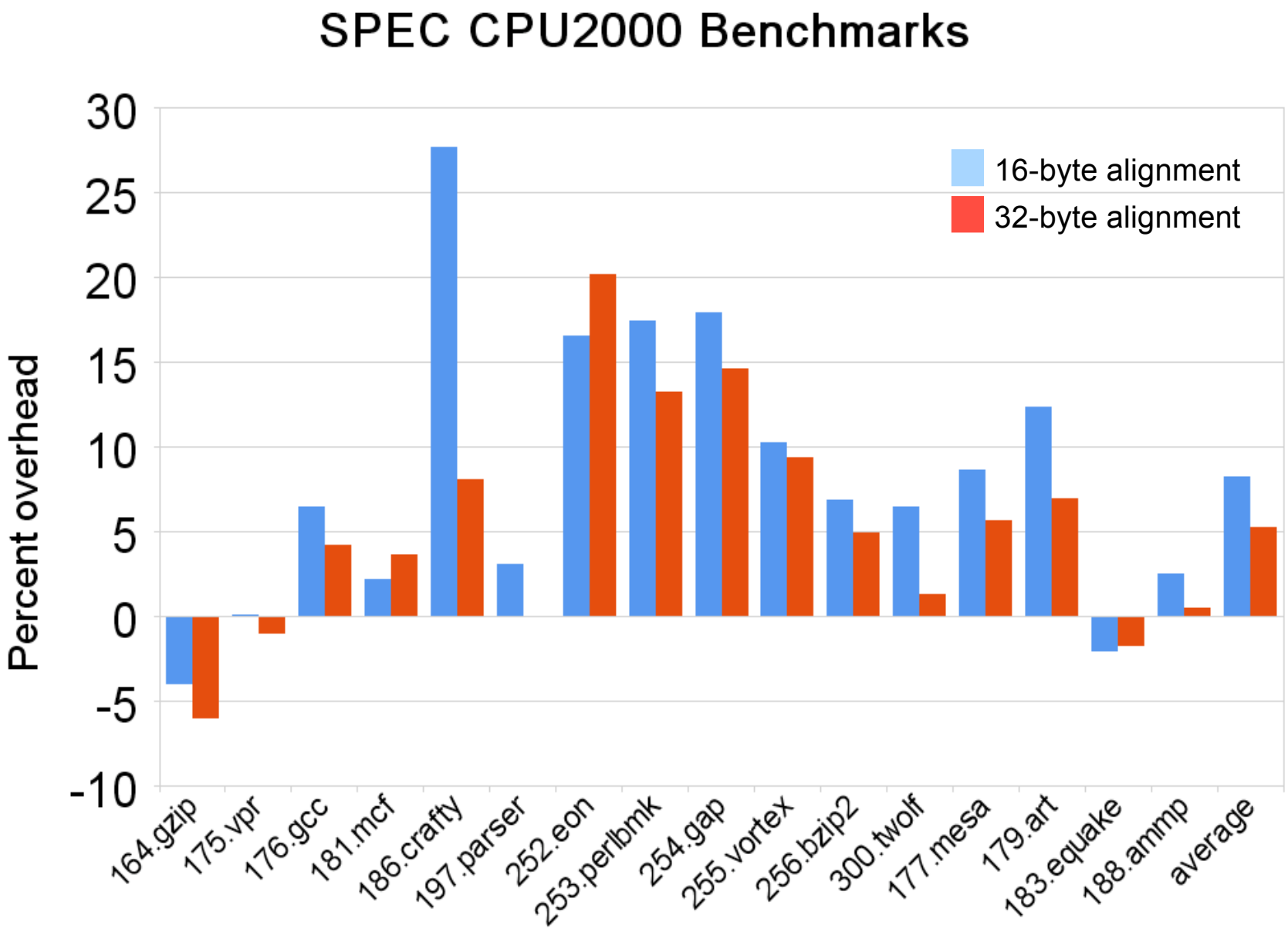


Google

# Outline

- Introduction
- Background
- Eliminating Side Effects (SFI)
- Allowing Side-Effects Safely (Architecture)
- Evaluation

# Performance



# What have we done with NaCl?

- Xaos
- Quake
- Lua interpreter
- H.264 decoder
- zlib (compression)
- cairo (2D rendering)
- bullet (physics)
- Life
- Voronoi diagram
- Spinning globe
- Mandelbrot
- ...

# Related Work

There are many relevant comparisons:

- SFI
  - Wahbe, Lucco, Anderson, Graham
  - VINO (SFI), SPIN (type safety), Nooks (page)
  - MiSFIT, PittSFIeld
- DTrace, Systemtap, XFI, Xax
- VMs: VMWare, Java
- JavaScript, Flash/AIR, Silverlight
- SSL, SSH, PGP

# Questions?

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



Google™

# Getting Involved

- Find a bug
- Port something
  - a compiler
  - an interpreted language
  - a game
  - an interesting library
- Build an application
- Build a multithreaded browser
- Add dynamic loading to NaCl
- Add support for JITted languages to NaCl



Google