Hack Your Language!

CSE401 Winter 2016

Introduction to Compiler Construction

Ras Bodik
Alvin Cheung
Maaz Ahmad
Talia Ringer
Ben Tebbs

What can you do with your 401 education

Just-in-time compilation
New language design

Announcements

Final quiz tomorrow

- please attend your assigned section
- review session tonight: EEB 125, 7pm

Project presentations next Tuesday

- Enjoy spring break!

What to do with 401 skills

Managed runtimes

- Just-in-time compilation and other tricks

Language design

- rfig, rake, and memoize

Case study: v8 Internals

The V8 engine

- Latest JS engine from Google
- Used for both client side (Chrome) and server side (node.js) applications
- Includes a Just-In-Time (JIT) compiler that directly compiles to x86
  - No bytecode or intermediate code generated

C++ and JS: compute the 25,000th prime

C++

```
class Primes {
 public:
  int getPrimeCount() const { return prime_count; }
  int getPrime(int i) const { return primes[i]; }
  void addPrime(int i) { primes[prime_count++] = i; }

  bool isDivisibe(int i, int by) { return (i % by) == 0; }

  bool isPrimeDivisible(int candidate) {
    for (int i = 1; i < prime_count; ++i) {
      if (isDivisibe(candidate, primes[i])) return true;
    }
    return false;
  }

 private:
  volatile int prime_count;
  volatile int primes[25000];
};

int main() {
  Primes p;
  int c = 1;
  while (p.getPrimeCount() < 25000) {
    if (!p.isPrimeDivisible(c)) {
      p.addPrime(c);
    }
    c++;
  }
  printf("%d\n", p.getPrime(p.getPrimeCount()-1));
}
```

JAVASCRIPT

```
function Primes() {
  this.prime_count = 0;
  this.primes = new Array(25000);
  this.getPrimeCount = function() { return this.prime_count; }
  this.getPrime = function(i) { return this.primes[i]; }
  this.addPrime = function(i) {
    this.primes[this.prime_count++] = i;
  }

  this.isPrimeDivisible = function(candidate) {
    for (var i = 1; i <= this.prime_count; ++i) {
      if ((candidate % this.primes[i]) == 0) return true;
    }
    return false;
  }
};

function main() {
  p = new Primes();
  var c = 1;
  while (p.getPrimeCount() < 25000) {
    if (!p.isPrimeDivisible(c)) {
      p.addPrime(c);
    }
    c++;
  }
  print(p.getPrime(p.getPrimeCount()-1));
}

main();
```

C++ and unoptimized JS code

C++

```
% g++ primes.cc -o primes
% time ./primes
287107

real     0m2.955s
user     0m2.952s
sys      0m0.001s
```

SHELL

JavaScript

```
% time d8 primes.js
287107

real     0m15.584s
user     0m15.612s
sys      0m0.073s
```

SHELL

C++ is 5x times faster

V8 compilation

- V8 actually consists of two compilers
  - Full compiler that generates code quickly
    - No type analysis / code optimization
  - Optimizing compiler that is used to compile code on the fly
    - "Just-in-time" compiler that heavily optimized code that might use cutting-edge (read: unstable) features
    - Need to wrap code around "try/catch" blocks!
    - Example: code that utilizes platform dependent instructions / custom hardware accelerators

Just-in-time Features

Prototypes in Javascript

- JS is prototype-based
- Prototypes are cloned as new objects are created
  - Why is this costly?

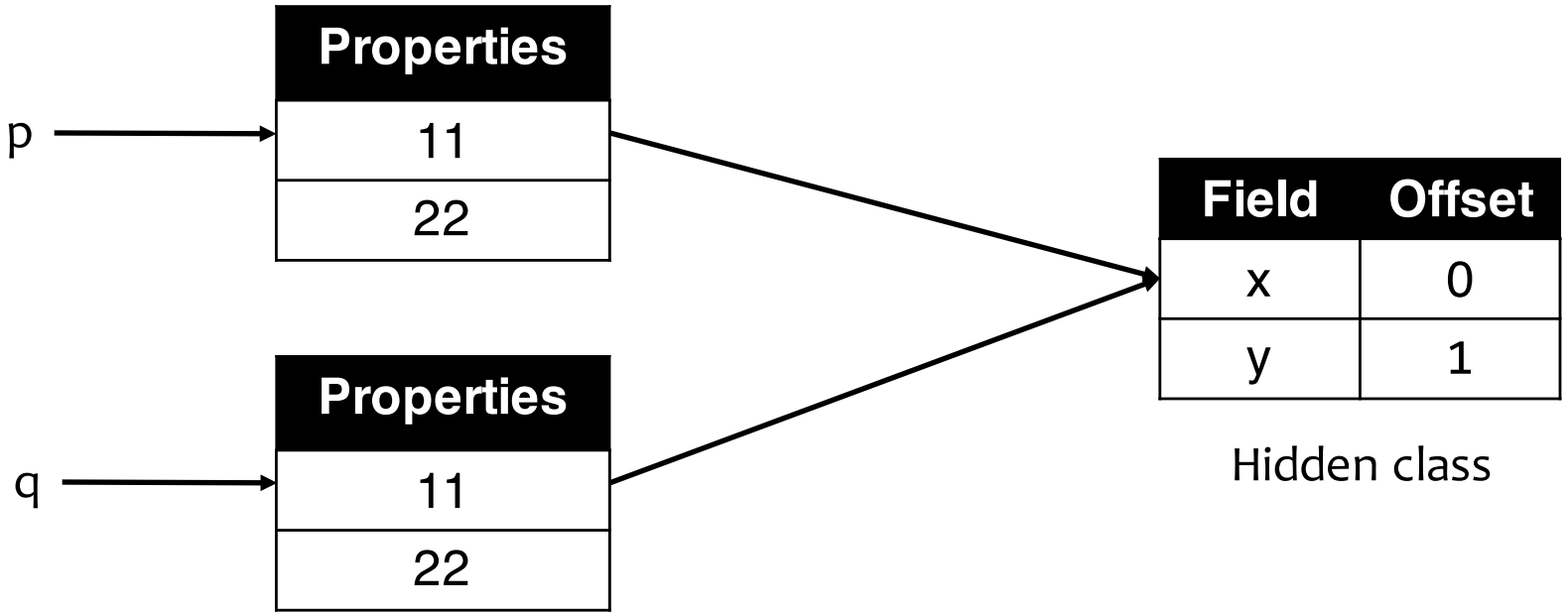
- We have seen how Lua implements objects using metatables
  - Idea: extract shared metadata into a common structure

- V8 applies similar concept as *hidden classes*

Hidden classes

```
function Point(x, y) {
  this.x = x;
  this.y = y;
}
var p = new Point(11, 22);
var q = new Point(33, 44);
```

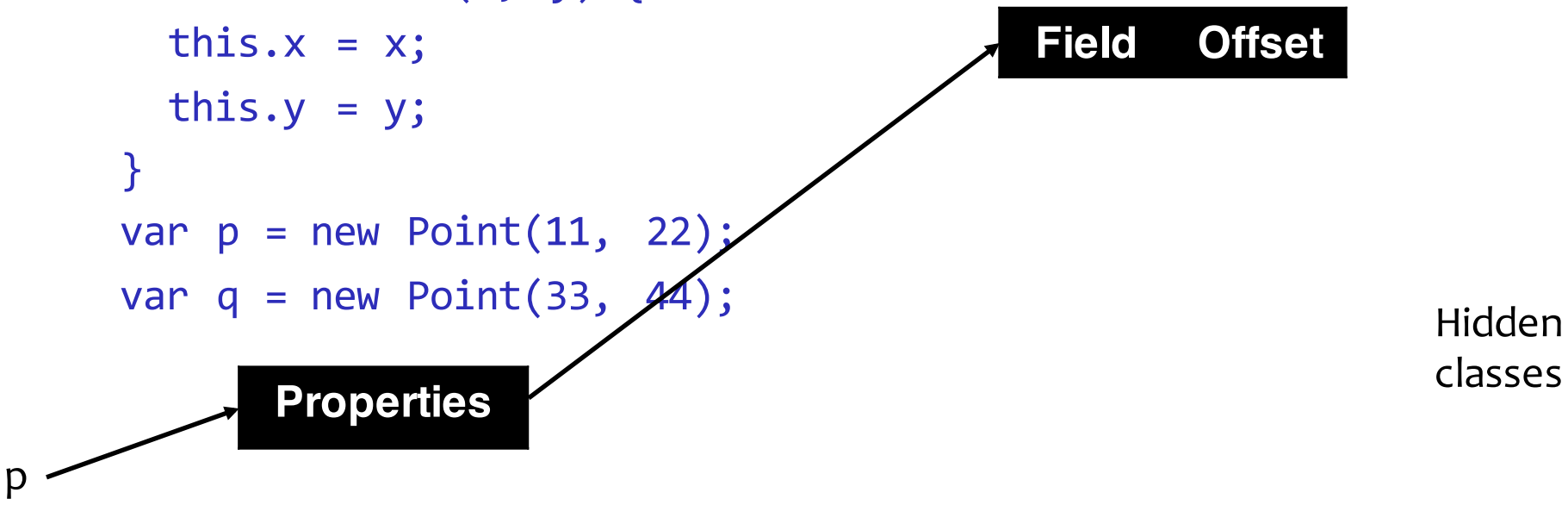
How can this be set up dynamically?



Hidden class

- Key idea: create a new hidden class every time a new property is added to an object

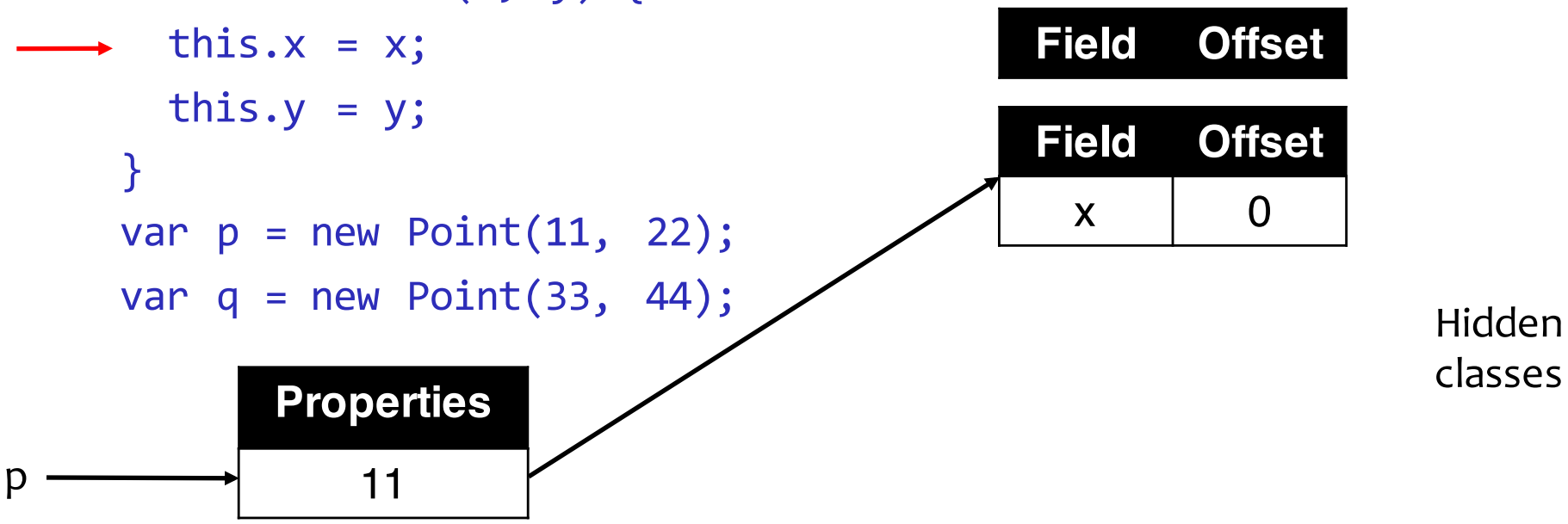
```
function Point(x, y) {
  this.x = x;
  this.y = y;
}
var p = new Point(11, 22);
var q = new Point(33, 44);
```



Hidden class

- Key idea: create a new hidden class every time a new property is added to an object

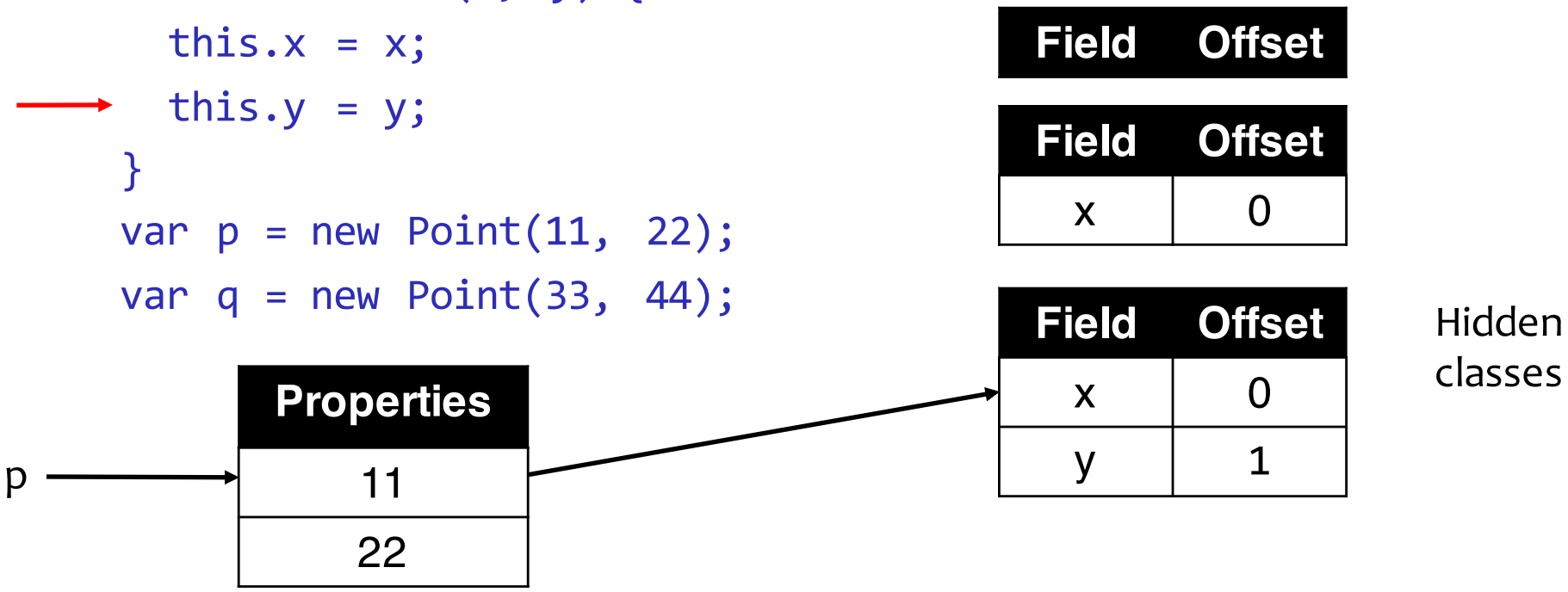
```
function Point(x, y) {
  this.x = x;
  this.y = y;
}
var p = new Point(11, 22);
var q = new Point(33, 44);
```



Hidden class

- Key idea: create a new hidden class every time a new property is added to an object

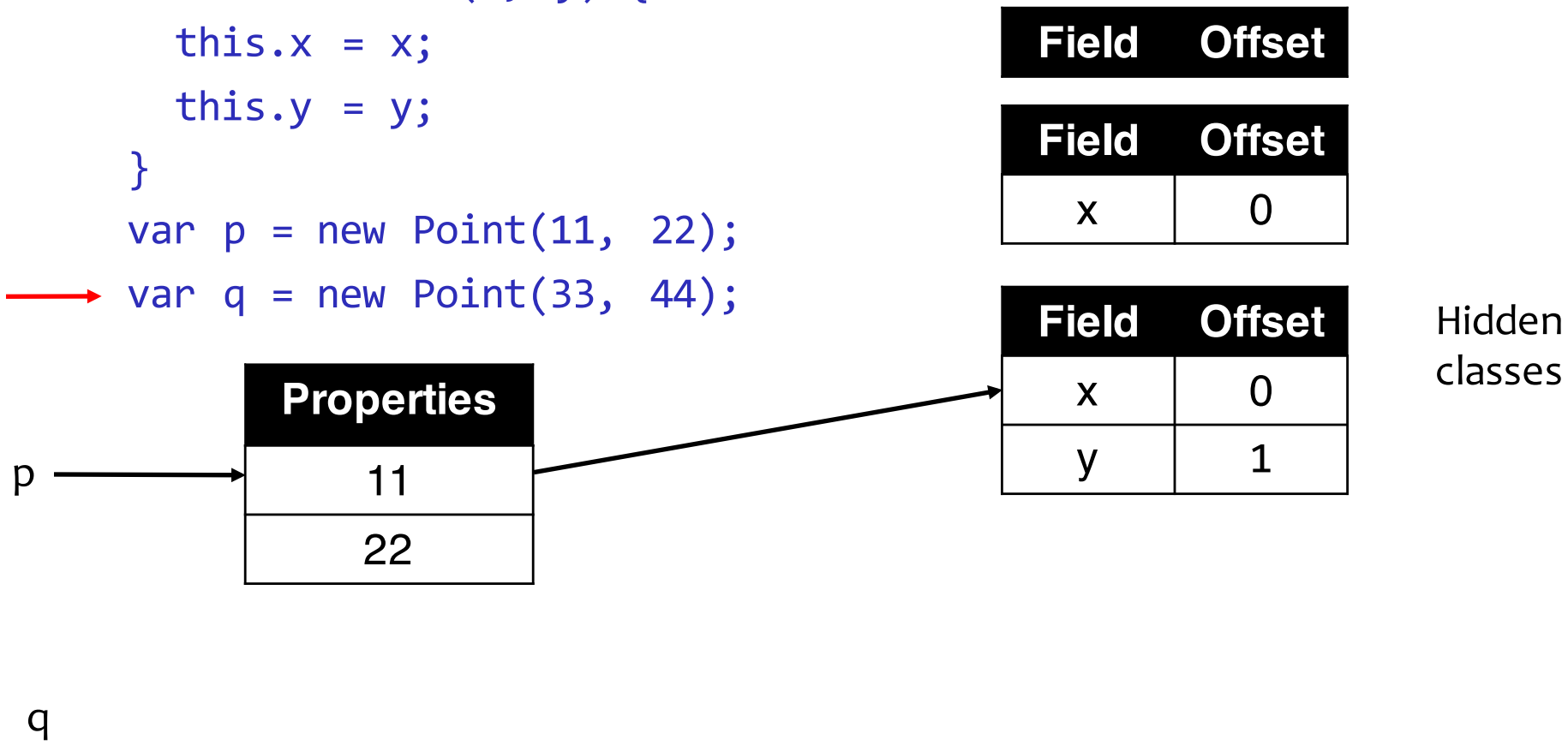
```
function Point(x, y) {
  this.x = x;
  this.y = y;
}
var p = new Point(11, 22);
var q = new Point(33, 44);
```



Hidden class

- Key idea: create a new hidden class every time a new property is added to an object

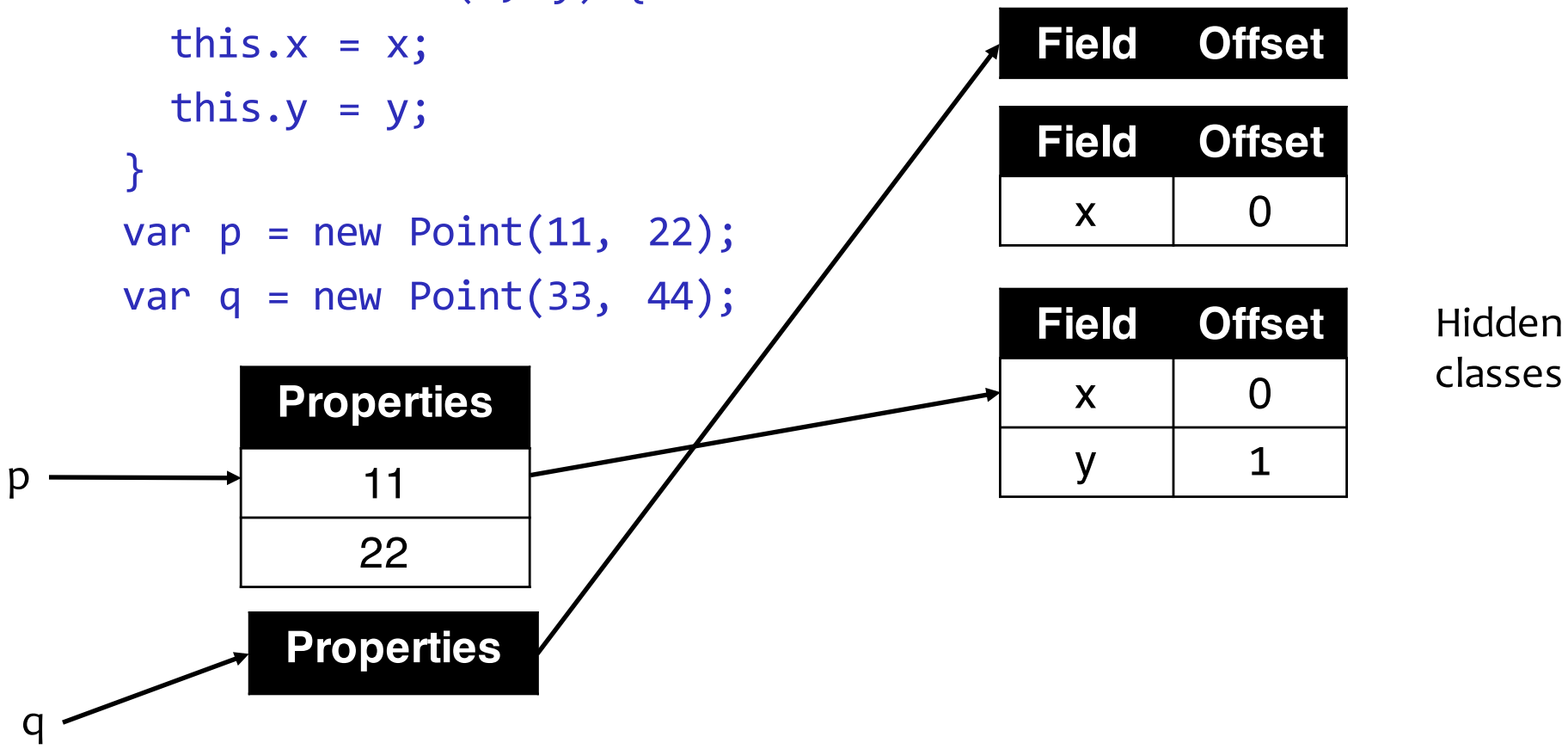
```
function Point(x, y) {
  this.x = x;
  this.y = y;
}
var p = new Point(11, 22);
var q = new Point(33, 44);
```



Hidden class

- Key idea: create a new hidden class every time a new property is added to an object

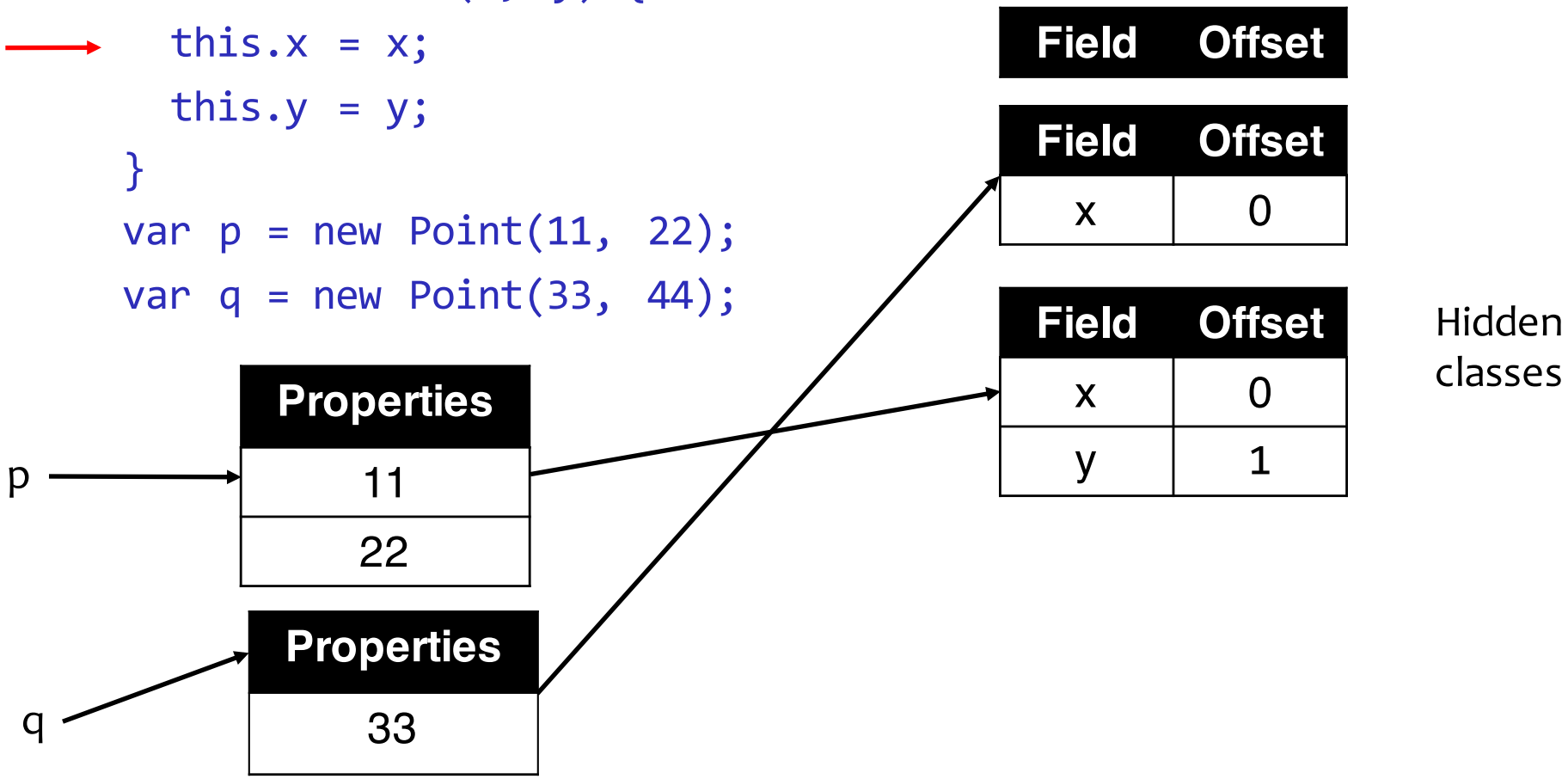
```
function Point(x, y) {
  this.x = x;
  this.y = y;
}
var p = new Point(11, 22);
var q = new Point(33, 44);
```



Hidden class

- Key idea: create a new hidden class every time a new property is added to an object

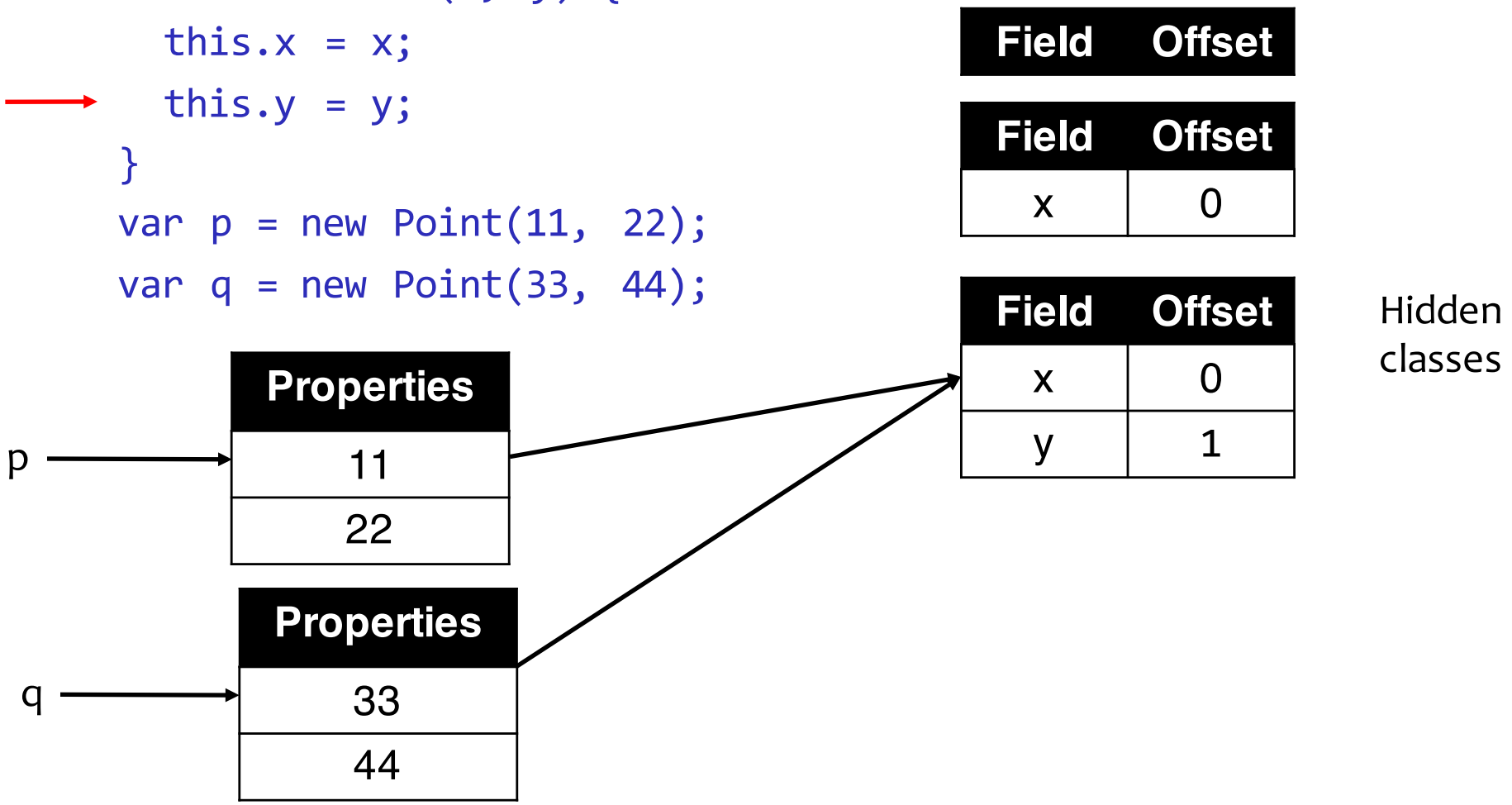
```
function Point(x, y) {
  this.x = x;
  this.y = y;
}
var p = new Point(11, 22);
var q = new Point(33, 44);
```



Hidden class

- Key idea: create a new hidden class every time a new property is added to an object

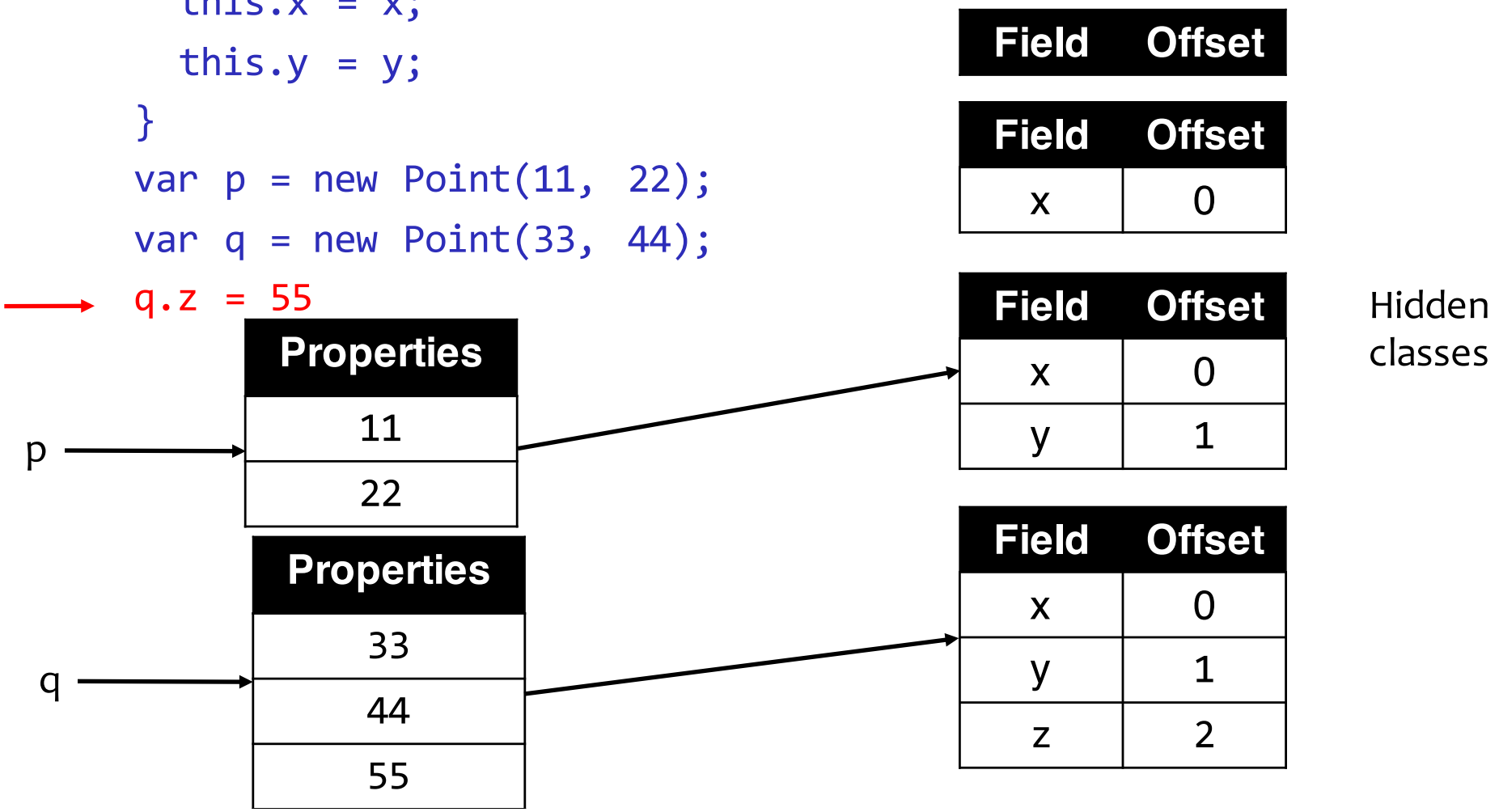
```
function Point(x, y) {
  this.x = x;
  this.y = y;
}
var p = new Point(11, 22);
var q = new Point(33, 44);
```



Hidden class

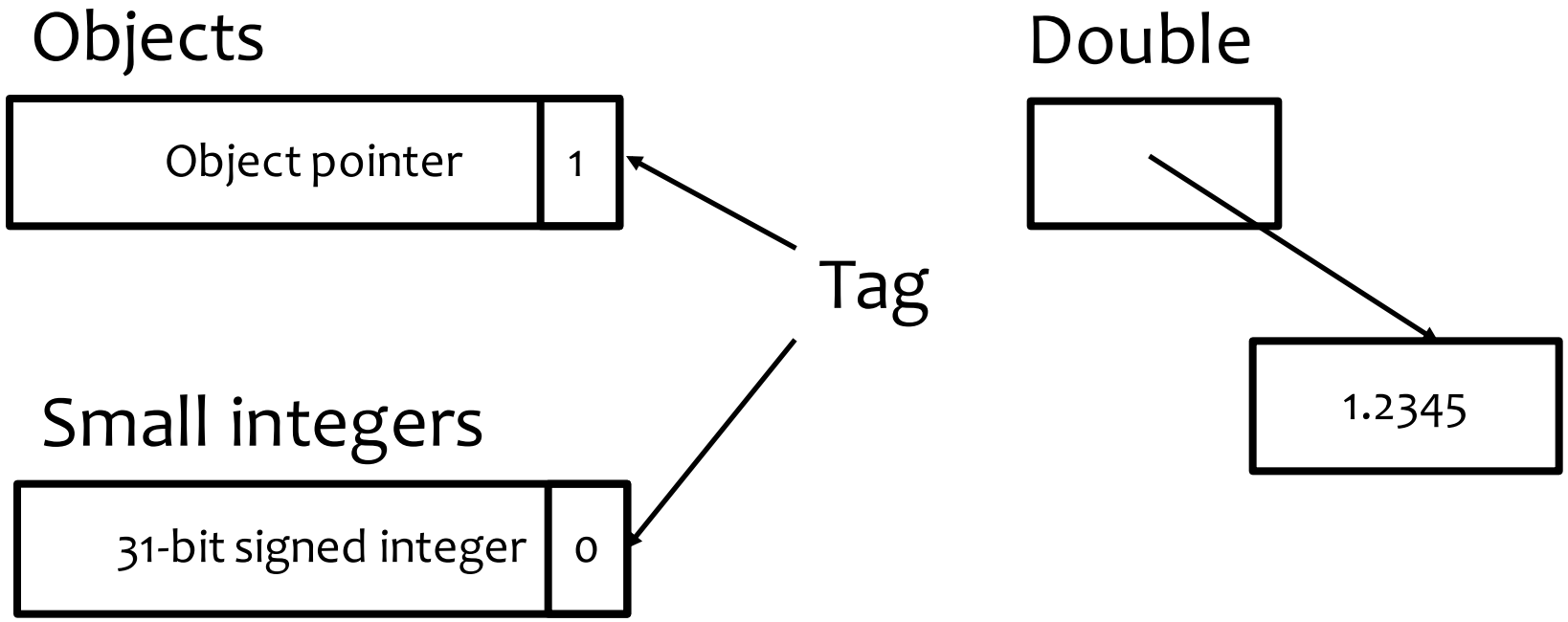
- Key idea: create a new hidden class every time a new property is added to an object

```
function Point(x, y) {
  this.x = x;
  this.y = y;
}
var p = new Point(11, 22);
var q = new Point(33, 44);
q.z = 55
```



Representing values

- Interoperate between objects and small ints



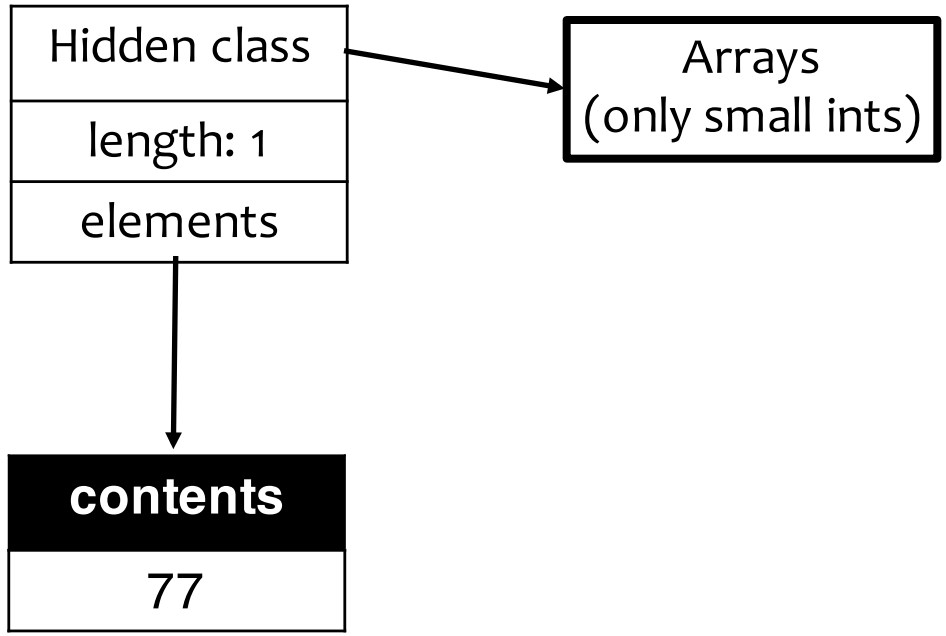
Representing arrays

- Simple: use dictionaries
  - What might be a performance issue?
- Better: specialize based on keys
  - If keys are consecutive, use pre-allocated linear array
  - If keys are sparse and non-consecutive, use hashtable
- Special case: array of doubles
  - Simple: store array of object pointers
  - Better: store raw double values instead

Back to hidden classes

```
var a = new Array();

a[0] = 77;
a[1] = 88;
a[2] = 0.2;
a[3] = true;
```



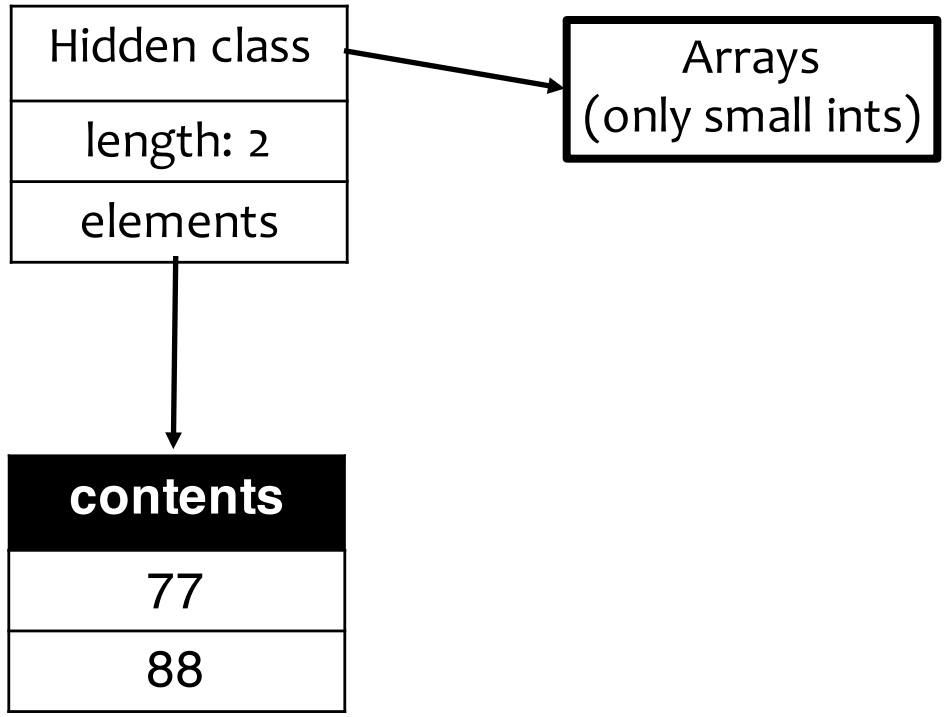
How many hidden classes are created?

Back to hidden classes

```
var a = new Array();

a[0] = 77;
a[1] = 88;
a[2] = 0.2;
a[3] = true;
```

How many hidden classes are created?

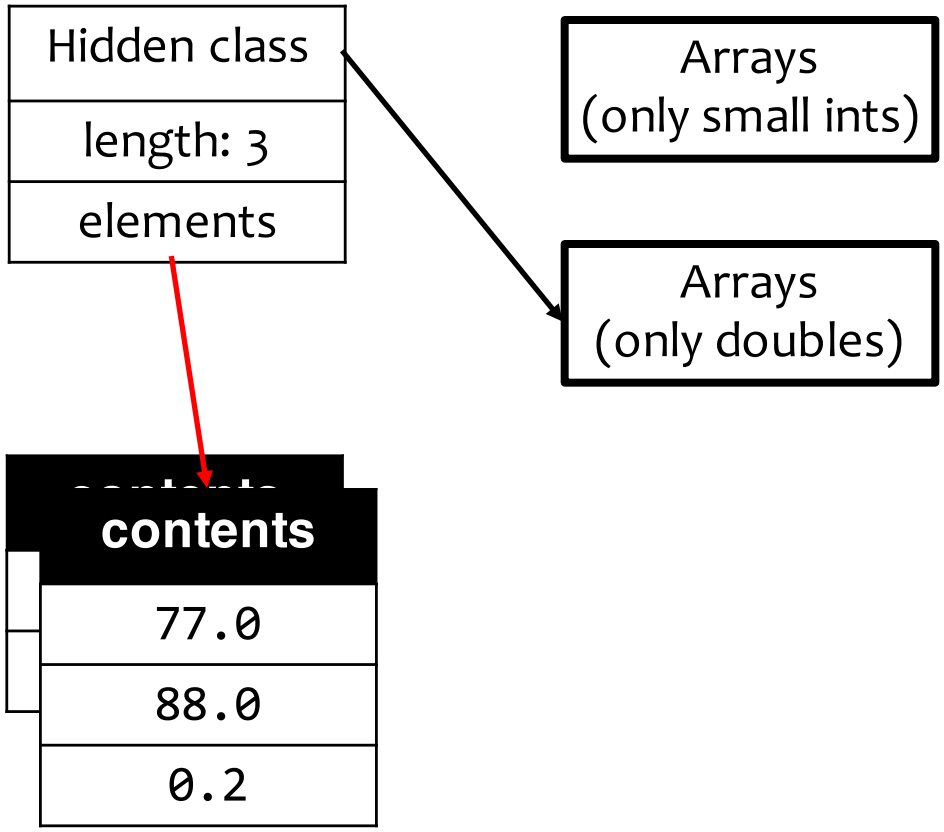


Back to hidden classes

```
var a = new Array();

a[0] = 77;
a[1] = 88;
a[2] = 0.2;
a[3] = true;
```

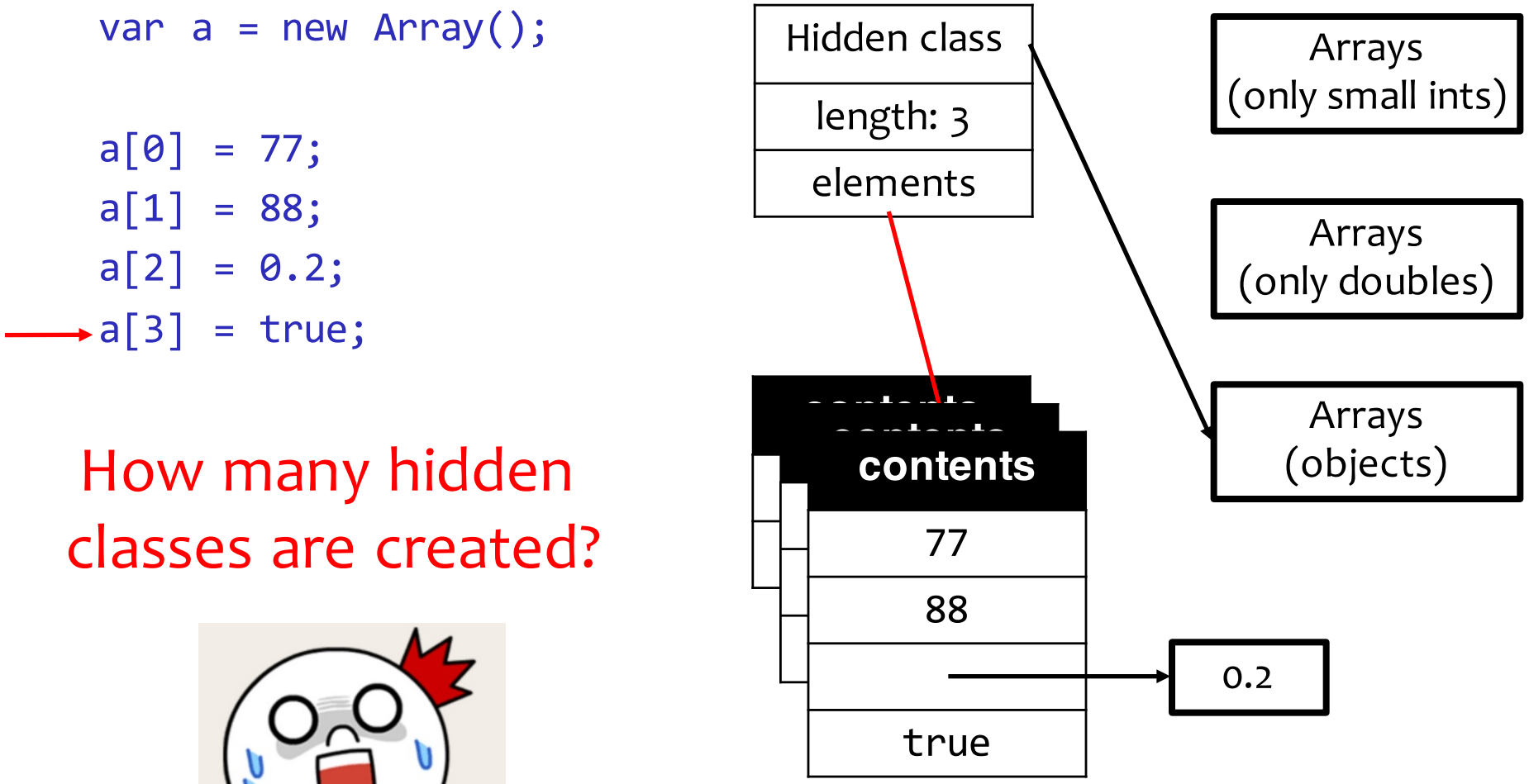
How many hidden classes are created?



Back to hidden classes

```
var a = new Array();

a[0] = 77;
a[1] = 88;
a[2] = 0.2;
a[3] = true;
```



How many hidden classes are created?

Why does this generate better code?

```
var a = [77, 88, 0.2, true];
```

Inline caches

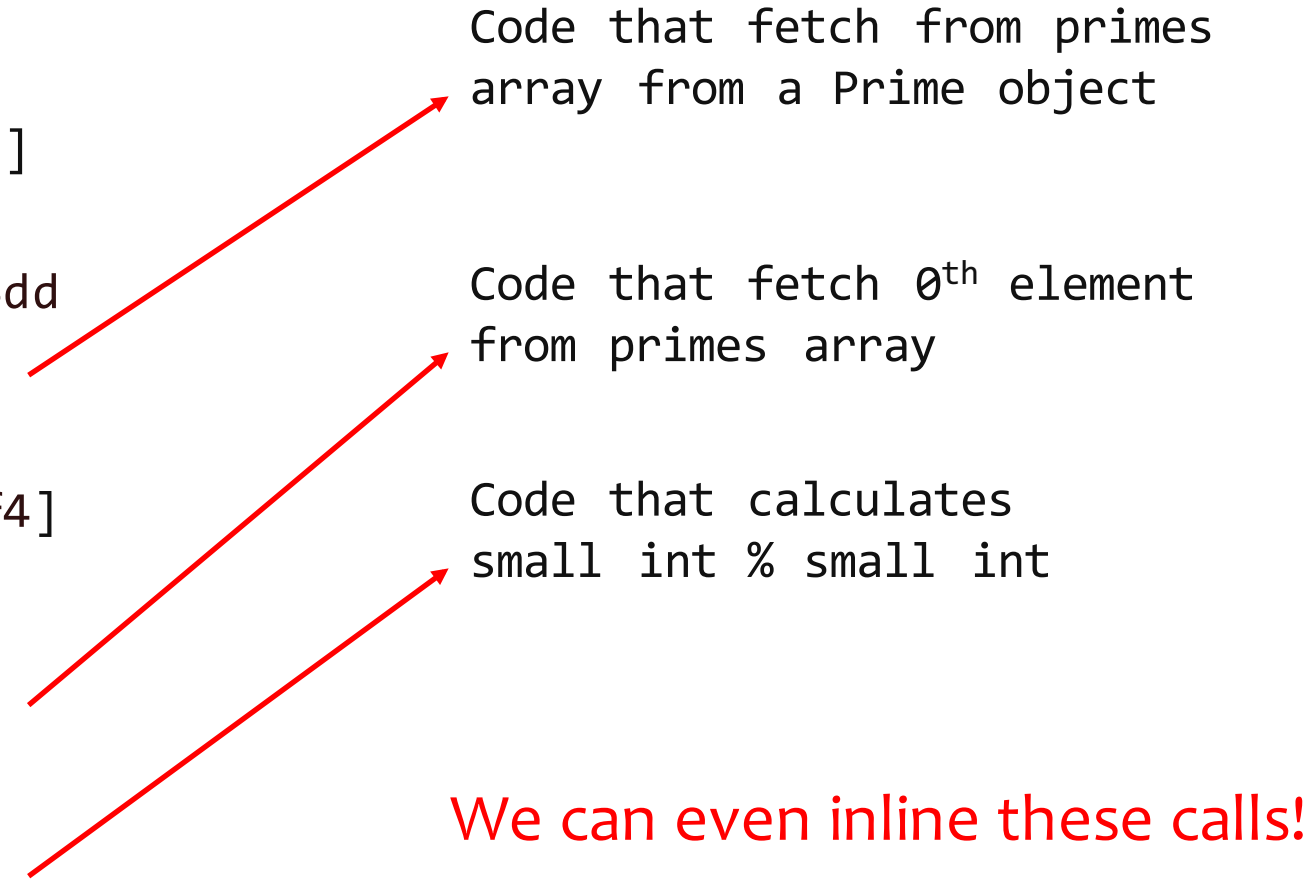
Unoptimized code for `candidate % this.primes[i]`

```
...
push [ebp+0x8]
mov eax,[ebp+0xc]
mov edx,eax
mov ecx,0x50b155dd
call LoadIC_Initialize          ;; this.primes
push eax
mov eax,[ebp+0xf4]
pop edx
mov ecx,eax
call KeyedLoadIC_Initialize     ;; this.primes[i]
pop edx
call BinaryOpIC_Initialize Mod ;; candidate % this.primes[i]
```

Inline caches

Key idea: skip type checking if we know the type of variables

```
...
push [ebp+0x8]
mov eax,[ebp+0xc]
mov edx,eax
mov ecx,0x50b155dd
call 0x311286e0
push eax
mov eax,[ebp+0xf4]
pop edx
mov ecx,eax
call 0x31129ae0
pop edx
call 0x3112ade0
```



We can even inline these calls!

Function inlining

- Non-polymorphic functions can be inlined entirely

```
function add (x, y) {
  return x + y;
}

add(1, 2);       // + is non-polymorphic
add("a", "b");   // + is now polymorphic
```

- Polymorphic functions requires generating call instructions
  - Need to check type of object that calls the function

After all these…

C++

```
% g++ primes.cc -o primes
% time ./primes
287107

real    0m2.955s
user    0m2.952s
sys     0m0.001s
```

SHELL

JavaScript

```
% time d8 primes-2.js
287107

real    0m1.829s
user    0m1.827s
sys     0m0.010s
```

SHELL

JS is 60% faster than C++!!

Don’t be too happy yet

C++

```
% g++ primes.cc -o primes -O3
% time ./primes
287107

real    0m1.564s
user    0m1.560s
sys     0m0.002s
```

SHELL

JavaScript

```
% time d8 primes-2.js
287107

real    0m1.829s
user    0m1.827s
sys     0m0.010s
```

SHELL

JS is still 17% slower than C++ -O3…

Lessons

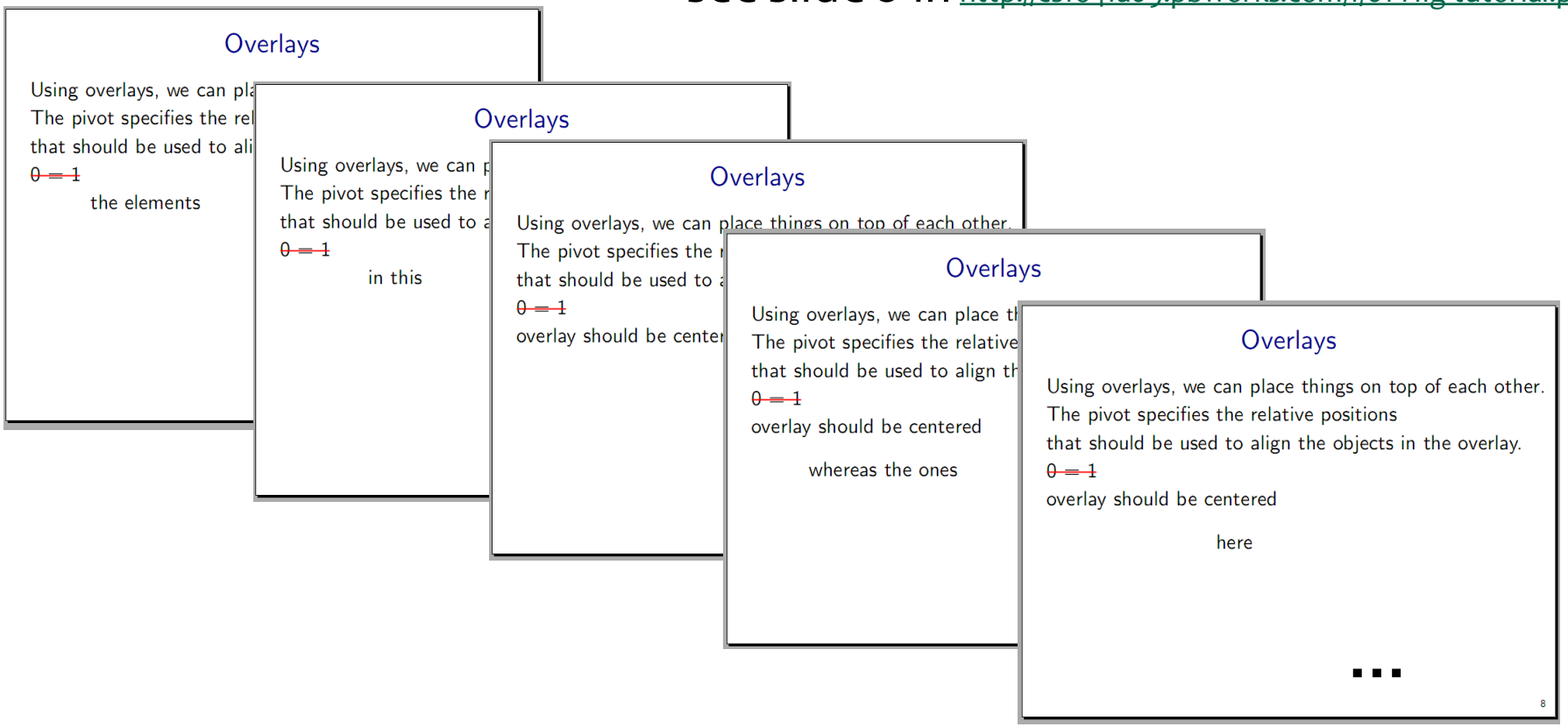
- Static-typing is a good thing ☺
- Opportunities to apply implementation techniques from statically-typed to dynamic-typed languages
- Techniques that you learned in this class are directly translatable to building real-world compilers!!

rfig

Rfig: A slide presentation language in Ruby

You need to give talks but get tired of PowerPoint.
Or you realize you are not a WYSIWYG person.
You embed a domain-specific language (DSL) into Ruby.

see slide 8 in http://cs164fa09.pbworks.com/f/01-rfig-tutorial.pdf



The animation in rfig, a Ruby-based language

```
slide!('Overlays',
  'Using overlays, we can place things on top of each other.',
  'The pivot specifies the relative positions',
  'that should be used to align the objects in the overlay.',


  overlay('0 = 1', hedge.color(red).thickness(2)).pivot(0, 0),

  staggeredOverlay(true, # True means that old objects disappear
    'the elements', 'in this', 'overlay should be centered', nil).pivot(0, 0),

  cr, pause,              # pivot(x, y): -1 = left, 0 = center, +1 = right

  staggeredOverlay(true,
    'whereas the ones', 'here', 'should be right justified', nil).pivot(1, 0),

  nil) { |slide| slide.label('overlay').signature(8) }
```

rfig was developed by Percy Liang, a Berkeley student

rake

rake

rake: an internal DSL, embedded in Ruby

Author: Jim Weirich

functionality similar to make

- has nice extensions, and flexibility, since it's embedded
- ie can use any ruby commands

even the syntax is close (perhaps better):

- embedded in Ruby, so all syntax is legal Ruby

http://martinfowler.com/articles/rake.html

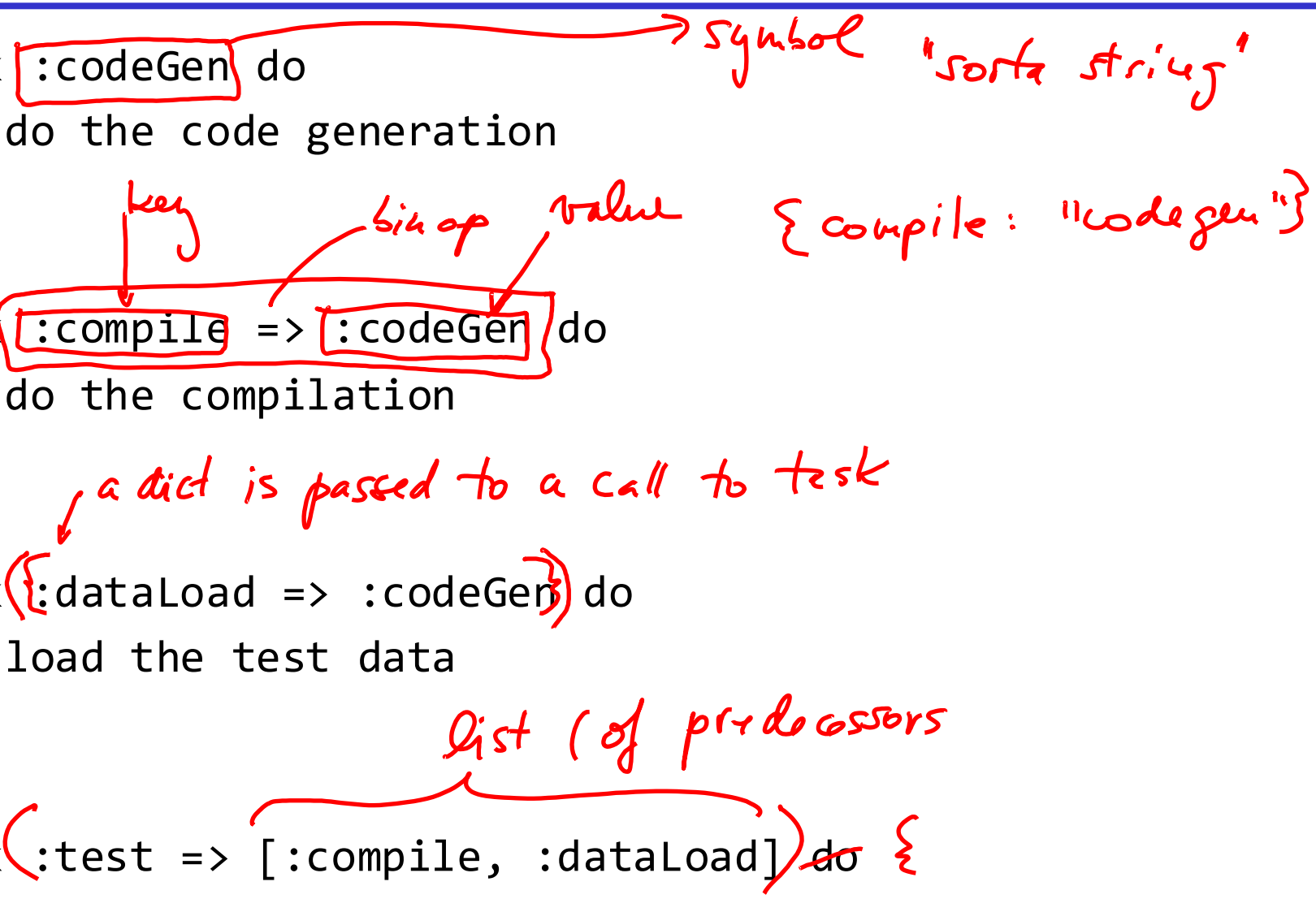
Example rake file

```
task :codeGen do
  # do the code generation
end

task :compile => :codeGen do
  # do the compilation
end

task({:dataLoad => :codeGen}) do
  # load the test data
end

task(:test => [:compile, :dataLoad]) do
  # run the tests
end
```



memoize

memoize

Memoize: a replacement for make.

Author: Bill McCloskey, Berkeley

Allows writing build scripts in "common" languages

- eg in Python or the shell
- rather than forcing you to rely on make's hopelessly recondite makefile language.

http://www.cs.berkeley.edu/~billm/memoize.html

Example: a shell script calling memoize

```
#!/bin/sh
memoize.py gcc -c file1.c
memoize.py gcc -c file2.c
memoize.py gcc -o program file1.o file2.o
```

Example: a python script calling memoize

```
#!/usr/bin/env python
import sys
from memoize import memoize
def run(cmd):
   status = memoize(cmd)
   if status: sys.exit(status)
run('ocamllex x86lex.mll')
run('ocamlyacc x86parse.mly')
run('ocamlc -c x86parse.mli')
run('ocamlc -c x86parse.ml')
run('ocamlc -c x86lex.ml')
run('ocamlc -c main.ml')
run('ocamlc -o program x86parse.cmi x86parse.cmo
   x86lex.cmo main.cmo')
```

How memoize works

Key idea: determine if a command needs to run

Assumptions: a command is a pure function

- its output depends only on its input files
- common for compilers and other build tools

Computing Dependences (what cmd depends on):

- uses strace to intercept system calls, like open
- r = os.system('strace -f -o %s -e trace=%s /bin/sh -c "%s"' % (outfile, calls, ecmd))

Computing file modification times:

- Alternative 1: use system file modification time
- Alternative 2: compute MD5 hash value for a value

Keep dependences and times in a file