CSCI-GA.3033.003 Scripting Languages

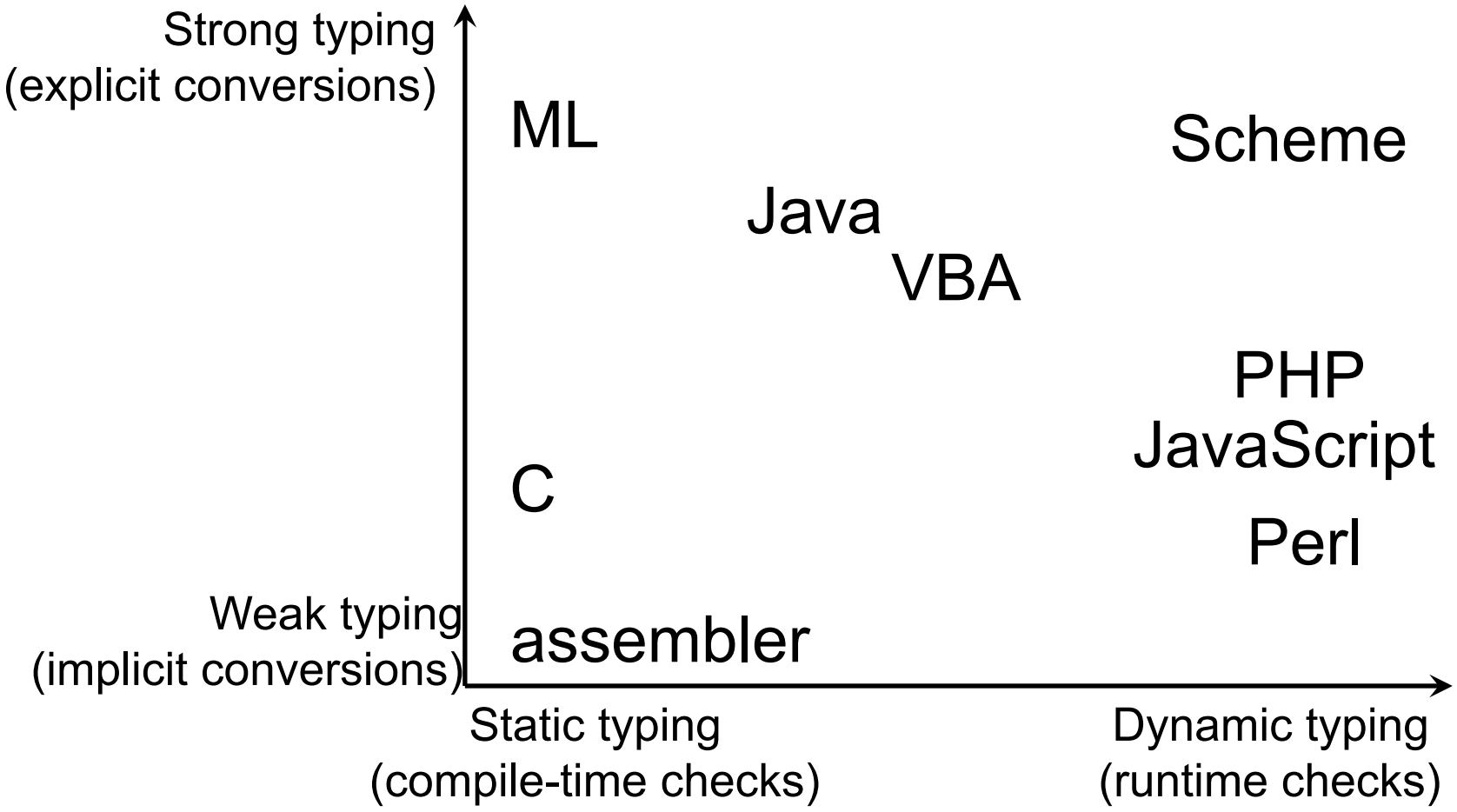
11/14/2013

TypeScript

Typing

- **Strong typing** = no implicit type conversion
- **Weak typing** = implicit type conversion
- **Static typing** = check for type errors at *compile* time
- **Dynamic typing** = check for type errors at *run time*

Weak/Strong, Static/Dynamic Typing



Unexpected Behavior

```
$ node
> '5' + 2
'52'
> '5' - 2
3
```

```
$ node
> '' == '0'
false
> 0 == ''
true
> 0 == '0'
true
```

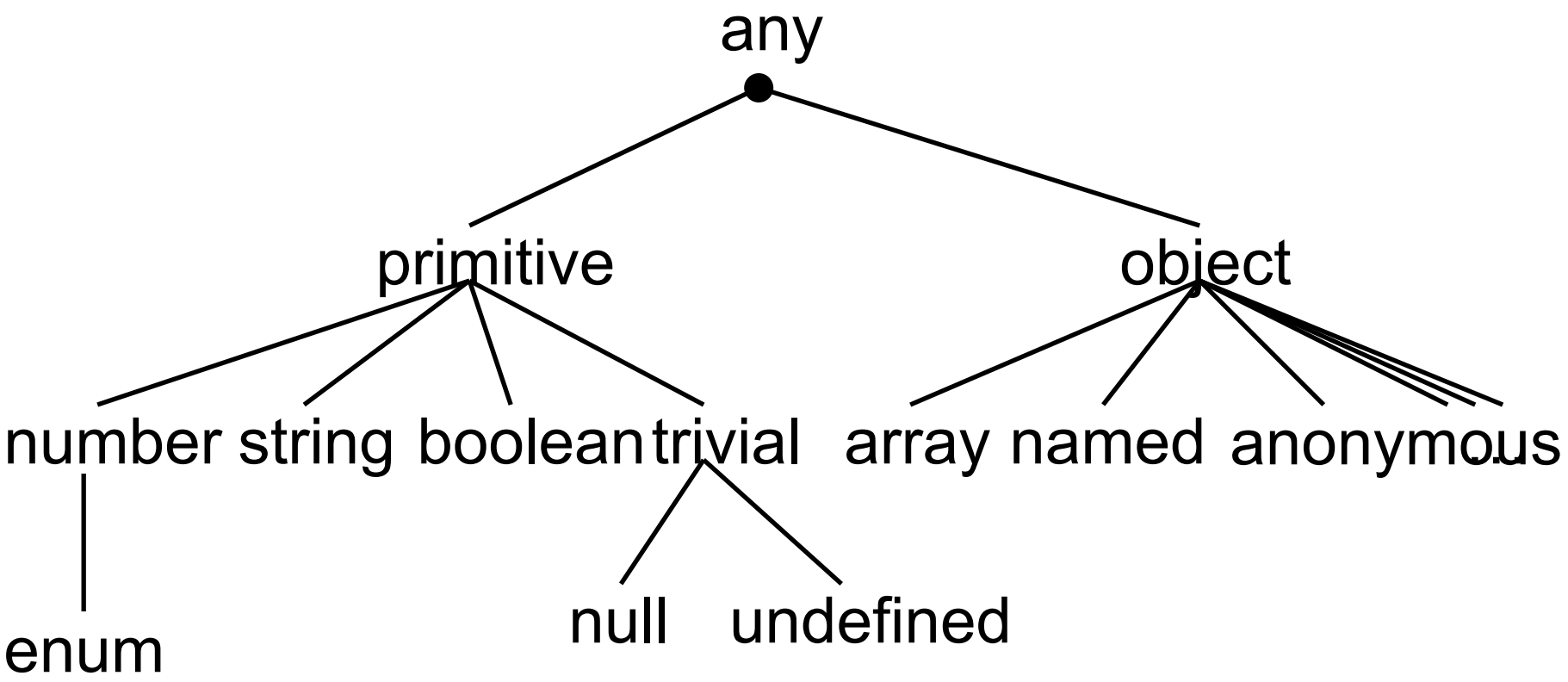
Strong Typing

- Makes intention explicit
- Makes code easier to read/maintain
- Can catch certain types of errors
- Less likely to have “unexpected” behavior

About TypeScript

- Superset of JavaScript
  - Static typing
  - Class-based object-oriented programming
- Developed by Microsoft
- Designed to make it easier to build large JavaScript applications
- Related languages:
  - CoffeeScript
  - Dart
  - Mascara

Types


any
primitive
object
number
string
boolean
trivial
array
named
anonymous
enum
null
undefined

How to Write + Run Code

Install as a node.js package:

$ npm install -g typescript

Run the compiler:

$ tsc hello.ts

Output is JavaScript:

$ ls

hello.js hello.ts

Type Declarations

hello.ts

```
function printit(msg : string) {
    console.log(msg);
}

var msg = "Hello World!";
printit(msg);
```

hello.js

```
function printit(msg) {
    console.log(msg);
}

var msg = "Hello World!";
printit(msg);
```

Type Declarations

hello.ts

```
function printit(msg : string) {
    console.log(msg);
}

var msg = "Hello World!";
printit(msg);
```

hello.js

```
function printit(msg) {
    console.log(msg);
}

var msg = "Hello World!";
printit(msg);
```

Type Checking

hello.js	output

function printit(msg : string) {
  console.log(msg);
}
 	
var msg = 5;
printit(msg);
	$ node hello.js
	5

Type Checking

hello.js	output

function printit(msg : string) {
  console.log(msg);
}
 	
var msg = 5;
printit(msg);
	$ node hello.js
	5

Type Checking

hello.ts	output

function printit(msg : string) {
  console.log(msg);
}
 	
var msg = 5;
printit(msg);
	$ tsc hello.ts
	hello.ts(6,1): error TS2081: Supplied parameters do not match any signature of call target.

Object Types

fruit.ts

```
interface Fruit {
    weight: number;
    color: string;
    seed?: boolean;
}

function pluck(f : Fruit) {
    console.log(f.color + " fruit");
}

var x = {weight: 10, color: "red"};
pluck(x);
```

- An *interface* gives a name to an object type
- Purely compile-time construct
- Checks *structural* equality
- Fields with a ? Are optional.

Object Types

fruit.ts

```
interface Fruit {
    weight: number;
    color: string;
    seed?: boolean;
}

function pluck(f : Fruit) {
    console.log(f.color + " fruit");
}

var x = {weight: 10, color: "red"};
pluck(x);
```

- An *interface* gives a name to an object type
- Purely compile-time construct
- Checks *structural* equality
- Fields with a ? Are optional.

Classes

```
class Apple {
    constructor(public weight,
                public color) { }
    public pluck() {
        return this.color + " apple";
    }
}
```

- Introduces a named type and a member
- A declaration includes a type declaration and a constructor

Class Type

```
class Apple {
    constructor(public weight : number,
                public color : string) { }
    public pluck() {
        return this.color + " apple";
    }
}
```

```
Interface Apple {
    weight : number;
    color : string;
    pluck() : string;
}
```

Class Member

```
var Apple: {
    new(weight : number,
        color : string) : Apple;
}
```

- This is the JavaScript constructor function

Class Inheritance

```
class A {
  a: number;
  public f() {console.log("a"); }
  public g() {console.log("a"); }
}

class B extends A {
  b: string;
  public g() {console.log("b"); }
}

var b = new B();
b.f(); // a
```

- A derived class inherits all members from its base class if it does not override them
- Members with the same name and type are overridden

this

```
class A {
  a: number;
  public f() {console.log("a"); }
  public g() {console.log(a"); }
}

class B extends A {
  b: string;
  public g() {console.log("b"); }
}

var b = new B();
b.f(); // a
```

- A derived class inherits all members from its base class if it does not override them
- Members with the same name and type are overridden

Modules

```
module M {
 export function f() { return "f"; }
 function g() { return "g"; }
}
M.f();
M.g(); // Error, g is not exported
```

- Introduces namespaces
- Provide encapsulation
- Implemented using the same “pattern” as node modules

this

- In a constructor, member function:
  this is the class instance
- In a static function:
  this is constructor function
- In a function declaration: this is Any
- In the global module: this Any

Last Slide

- Next week: Security!