

# On Contracts, Sandboxes, and Proxies for JavaScript



*Matthias Keil*, Peter Thiemann

University of Freiburg, Germany

October 5, 2015, 18. Kolloquium Programmiersprachen und Grundlagen der Programmierung, KPS 2015  
Pörschach am Wörthersee, Österreich

- Language embedded contract system for JavaScript
- Enforced by run-time monitoring
  
- Standard abstractions for higher-order-contracts (base, function, and dependent contracts) [Findler,Felleisen'02]
  
- Systematic blame calculation
- Contract constructors generalize dependent contracts
- **Internal noninterference**

# Base Contract [Findler, Felleisen '02]

- *Base Contracts* are built from predicates
- Specified by a plain JavaScript function

## Base Contract

```
function isNumber (arg) {  
  return (typeof arg) === 'number';  
};  
var _Number_ = Contract.Base(isNumber);  
  
assert(1, _Number_); ✓  
assert('a', _Number_); ✗ blame the subject
```

# Function Contract [Findler,Felleisen'02]

```
// Number × Number → Number  
function plus (x, y) {  
  return (x + y);  
}
```

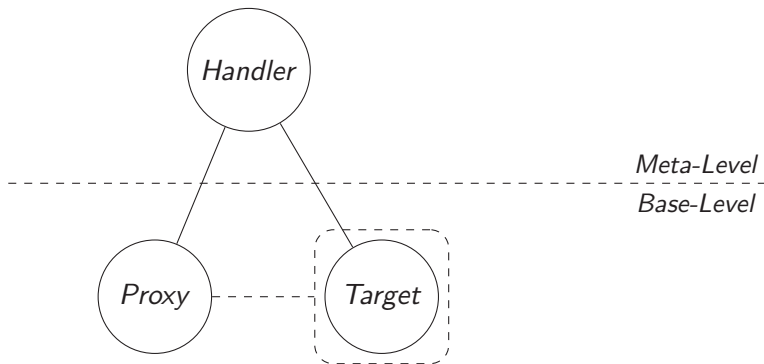
## Function Contract

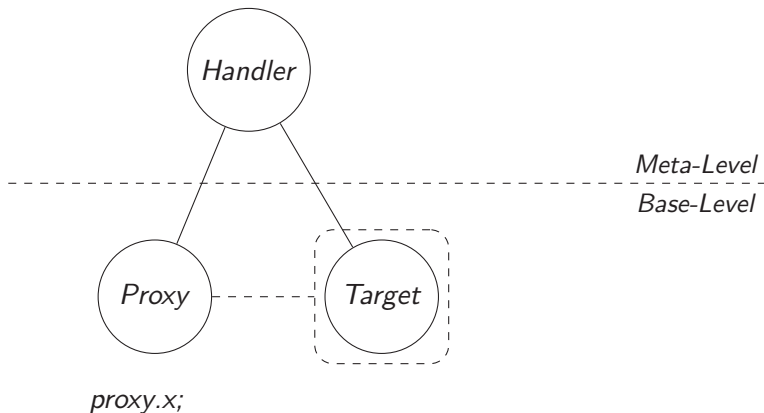
```
var _Plus_ = Contract.Function([_Number_, _Number_],  
  _Number_);
```

```
var plusNumber = assert(plus, _Plus_);
```

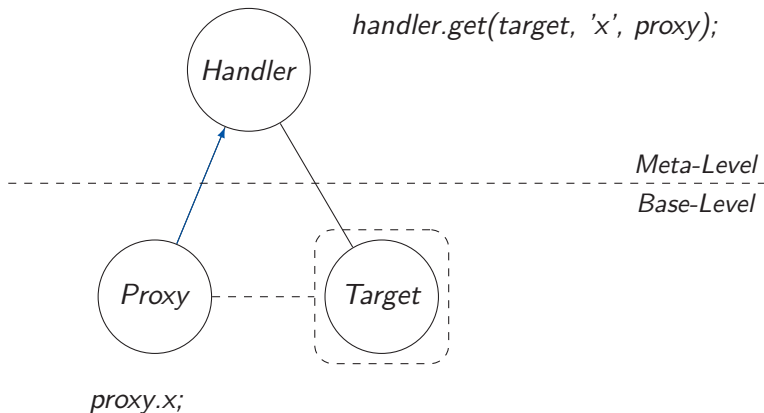
```
plusNumber(1, 1); ✓
```

```
plusNumber('a', 'a'); ✗ blame the context
```

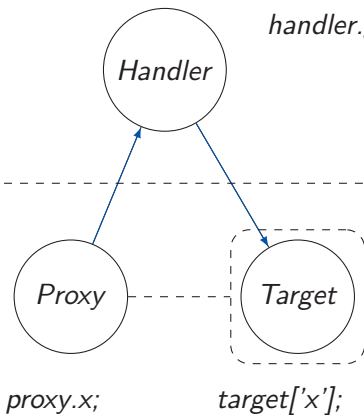




`handler.get(target, 'x', proxy);`

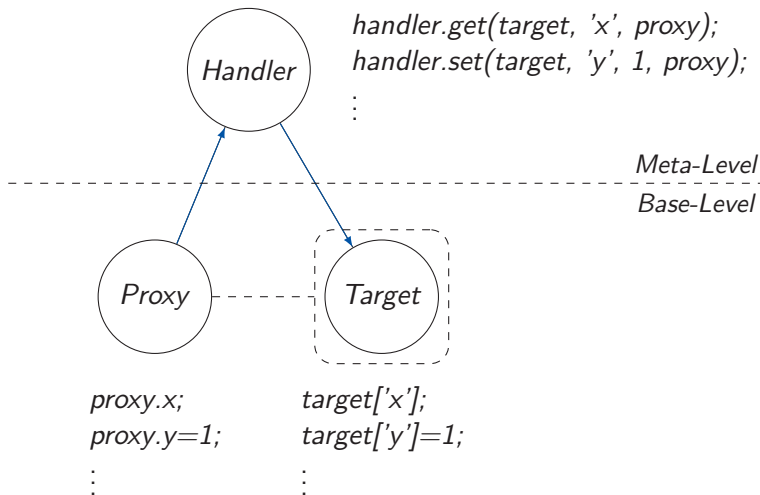


```
handler.get(target, 'x', proxy);
```



Meta-Level  
Base-Level





## Noninterference

The contract system should not interfere with the execution of application code.

- *External Noninterference* arises from the interaction of the contract system with the host program
  - 1 Exceptions
  - 2 Object equality
- *Internal Noninterference* arises from executing unrestricted code in predicates

- No syntactic restrictions on predicates
- Predicates may try to write to data that is visible to the application
- *Solution*: Predicate evaluation takes place in a sandbox

## Faulty Predicate

```
function isNumber (arg) {  
  type = (typeof arg);  
  return type === 'number';  
};
```

- No syntactic restrictions on predicates
- Predicates may try to write to data that is visible to the application
- *Solution*: Predicate evaluation takes place in a sandbox

## Faulty Predicate

```
function isNumber (arg) {  
  type = (typeof arg); X access forbidden  
  return type === 'number';  
};
```

- All contracts guarantee noninterference
- Read-only access is safe

## Predicate with Read-Access

```
function isArray (arg) {  
  return (arg instanceof Array); ✓  
}
```

- Predicate execution may violate contracts
- *Solution*: Sandbox redefines the responsibility

## Predicate with Read-Access

```
function addOne (arg) {  
  return plusNumber(arg, '1'); X blame the ?  
}
```

- Predicate execution may violate contracts
- *Solution*: Sandbox redefines the responsibility

## Predicate with Read-Access

```
function addOne (arg) {  
  return plusNumber(arg, '1'); X blame the contract  
}
```

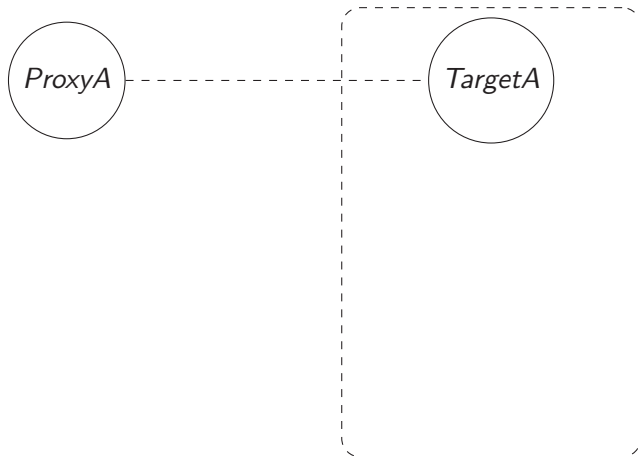
## Faulty Predicate

```
function isNumber (arg) {  
  type = (typeof arg);  
  return type === 'number';  
};
```

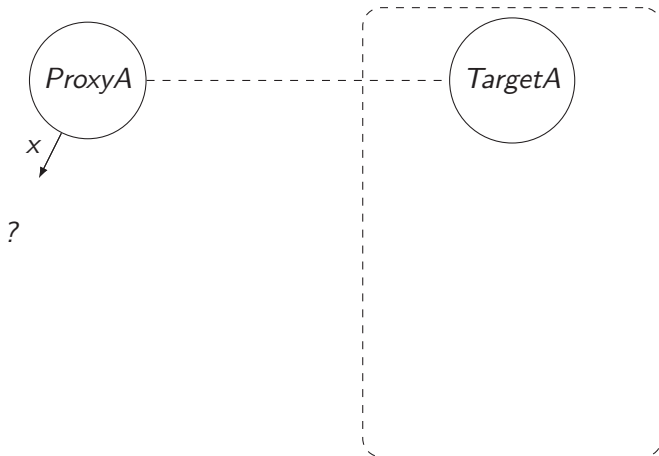
- 1 Place a write protection on objects (e.g. *this*, *arg*)
- 2 Remove external bindings of functions (e.g. *type*)



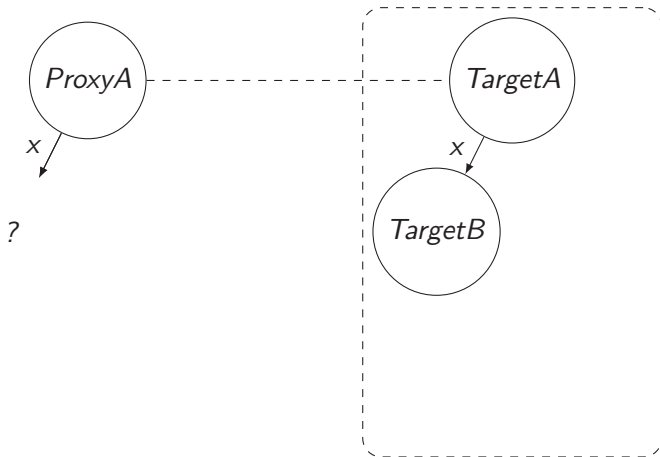
# Identity Preserving Membrane



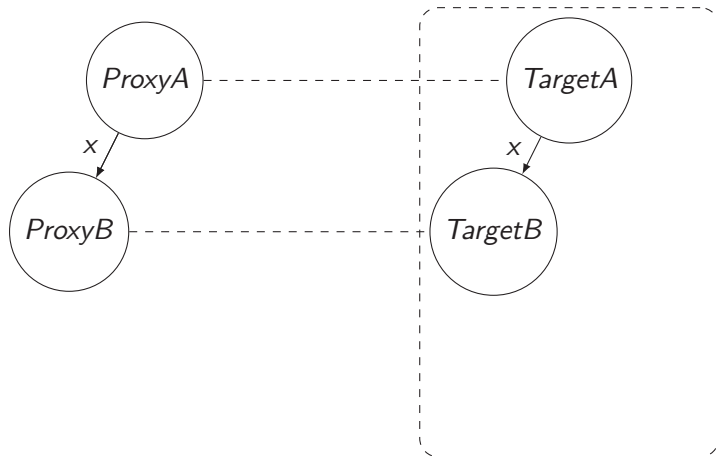
# Identity Preserving Membrane



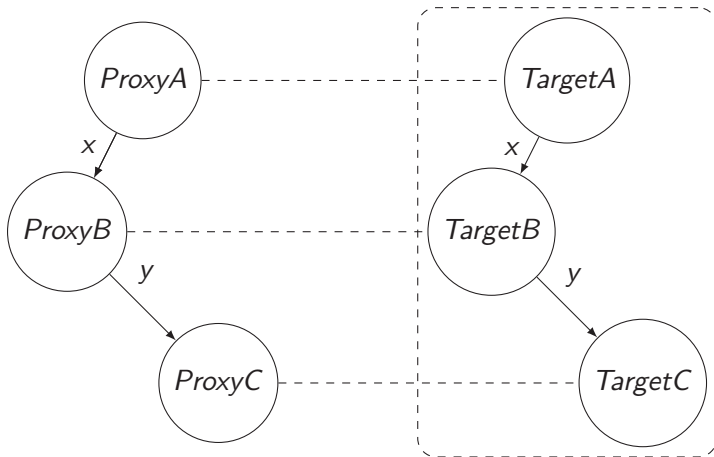
# Identity Preserving Membrane



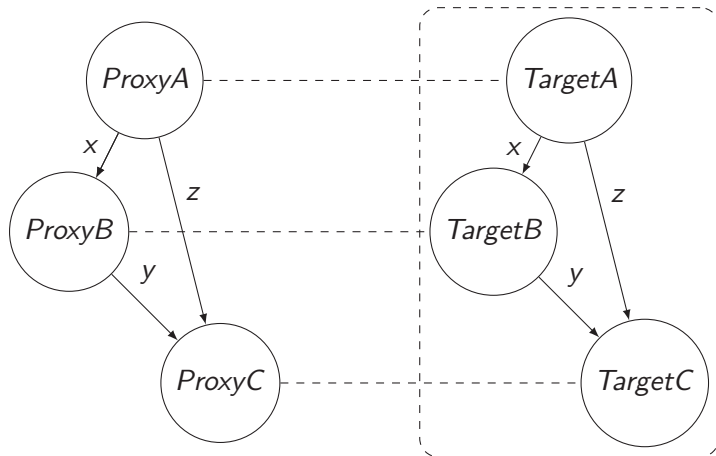
# Identity Preserving Membrane

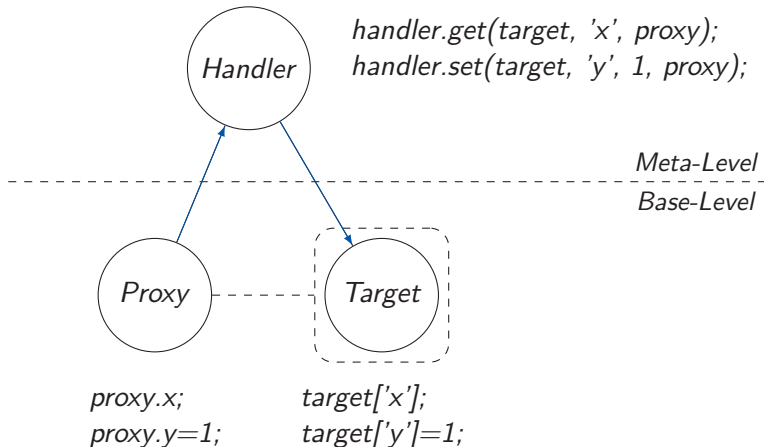


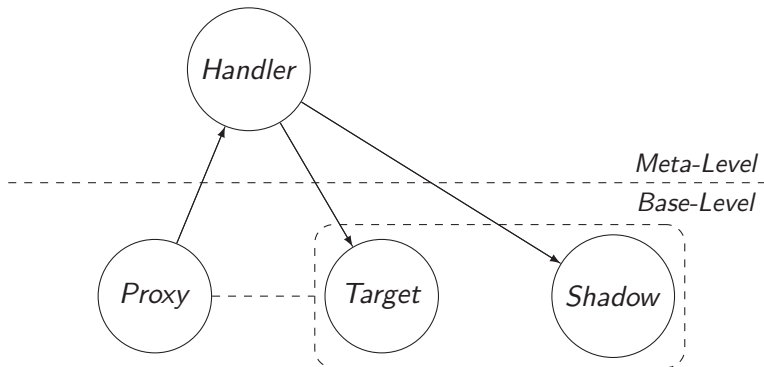
# Identity Preserving Membrane



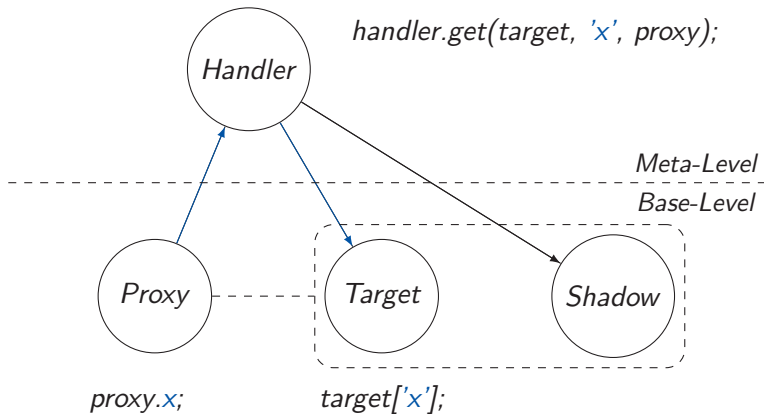
# Identity Preserving Membrane

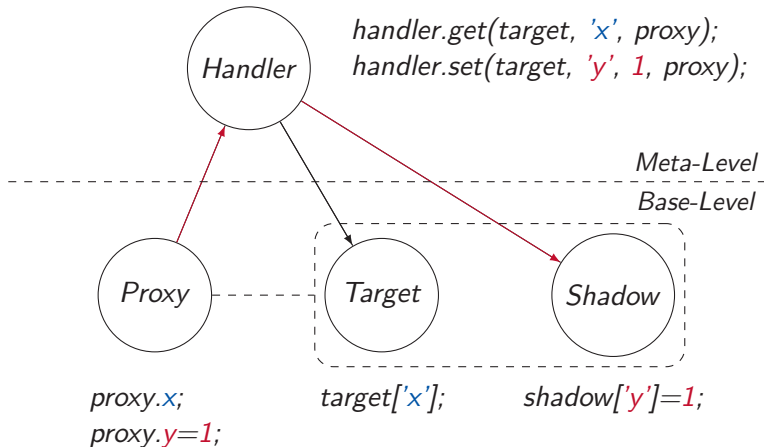




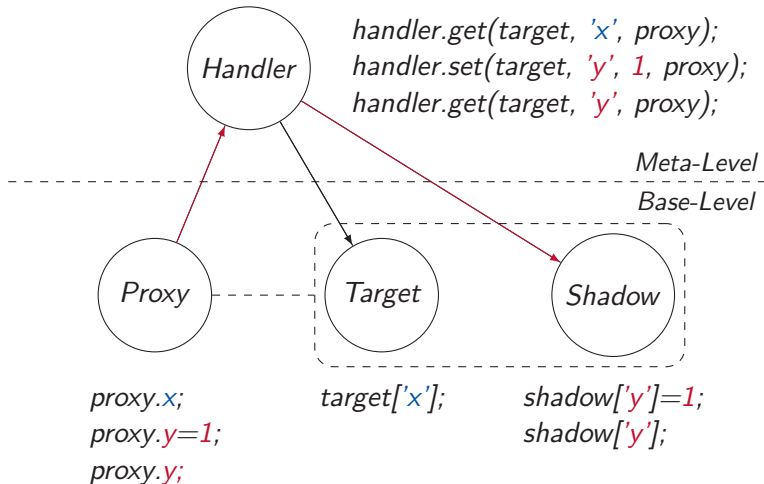








# Shadow Objects



# Function Recompilation



```
var x = 1;
```

```
function f (){
```

```
function g (y) {
```

```
var z = 1;
```

```
return x+y+z;
```

```
}
```

```
}
```

```
var x = 1;
```

```
function f (){
```

```
  "function g (y) {  
    var z = 1;  
    return x+y+z;  
  }"
```

```
}
```

```
var x = 1;
```

```
with(sbxglobal){
```

```
  eval("function g (y) {  
    var z = 1;  
    return x+y+z;  
  }");
```

```
}
```

```
var x = 1;
```

```
with(sbxglobal){
```

```
function g (y) {  
  var z = 1;  
  return x+y+z;  
}
```

```
}
```

## TreatJS:

- Language embedded, dynamic, higher-order contract system for full JavaScript
- Guarantees noninterference

## Sandbox:

- Language embedded sandbox for full JavaScript
- Runs JavaScript code in isolation isolation