BLOCKS

CLOSURES IN OBJECTIVE C

iDA MediaFoundry

BLOCKS?

= CLOSURES

CLOSURES?

CLOSURE = FUNCTION POINTER?

PSEUDO EXAMPLE: FUNCTION POINTER

FUNCTION POINTER AS

```
// Somewhere in the Array class
func filter(*funcPtr()) {
    var filtered = new Array();
    for (elem in this) {
        if (funcPtr(elem)) { filtered.add(elem); }
    }
    return filtered;
}

// Somewhere in your code
func myPredicateFunction(element) = {
    return element <= 5;
}
func someFunction() {
    var array = new Array(1, 2, 3);
    array.filter(&myPredicateFunction);
}
```

FUNCTION DEFINITION

CALL PASSING FUNCTION AS POINTER

PROBLEM WITH FUNCTION POINTERS

```
// Somewhere in the Ar
func filter(*funcP
    var filtered = new       ay();
    for (elem in this) {
        if (funcPtr(elem)) { filtered.add(elem); }
    }
    return filtered;
}

// Somewhere in your code
func myPredicateFunction(element, upperBound) = {
    return element <= upperBound;
}
func someFunction() {
    var array = new Array(1, 2, 3);
    array.filter(&myPredicateFunction);
}
```

WILL NOT WORK ACCORDING TO PLAN!

ADD AN EXTRA ARGUMENT TO THE FUNCTION

CLOSURE = FUNCTION POINTER?

CLOSURE = FUNCTION POINTER ON STEROIDS!

PSEUDO EXAMPLE: CLOSURE

CLOSURE AS ARGUMENT

```
// Somewhere in the Array class
func filter(closure()) {
    var filtered = new Array();
    for (elem in this) {
        if (closure(elem)) { filtered.add(elem); }
    }
    return filtered;
}

// Somewhere in your code
func someFunction() {
    var array = new Array(1, 2, 3);
    var upperBound = 5;
    def myPredicate(element) = {
        return element <= upperBound;
    }
    array.filter(myPredicate);
}
```

CLOSURE DEFINITION

CALL PASSING THE CLOSURE

PSEUDO EXAMPLE: CLOSURE

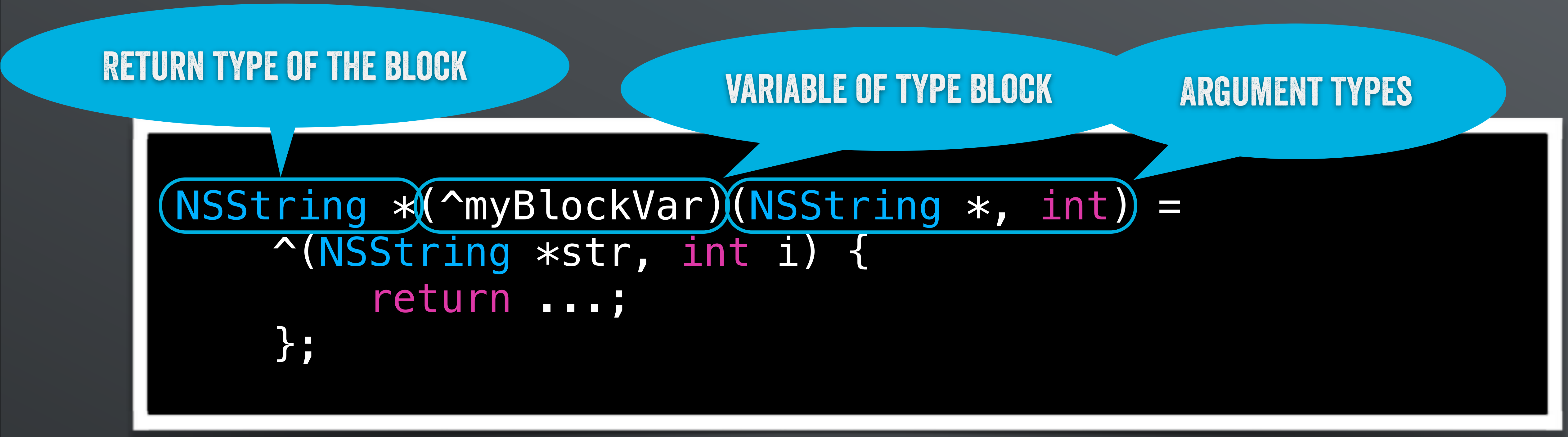
```
// Somewhere in the Array class
func filter(closure()) {
    var filtered = new Array();
    for (elem in this) {
        if (closure(elem)) { filtered.add(elem); }
    }
    return filtered;
}

// Somewhere in your cod
func someFunction() {
    var array = new Array  , 2, 3);
    var upperBound = 5;
    def myPredicate(element) = {
        return element <= upperBound;
    }
    array.filter(myPredicate);
}
```

LOCAL TO SOME FUNCTION

USABLE WITHIN CLOSURE!

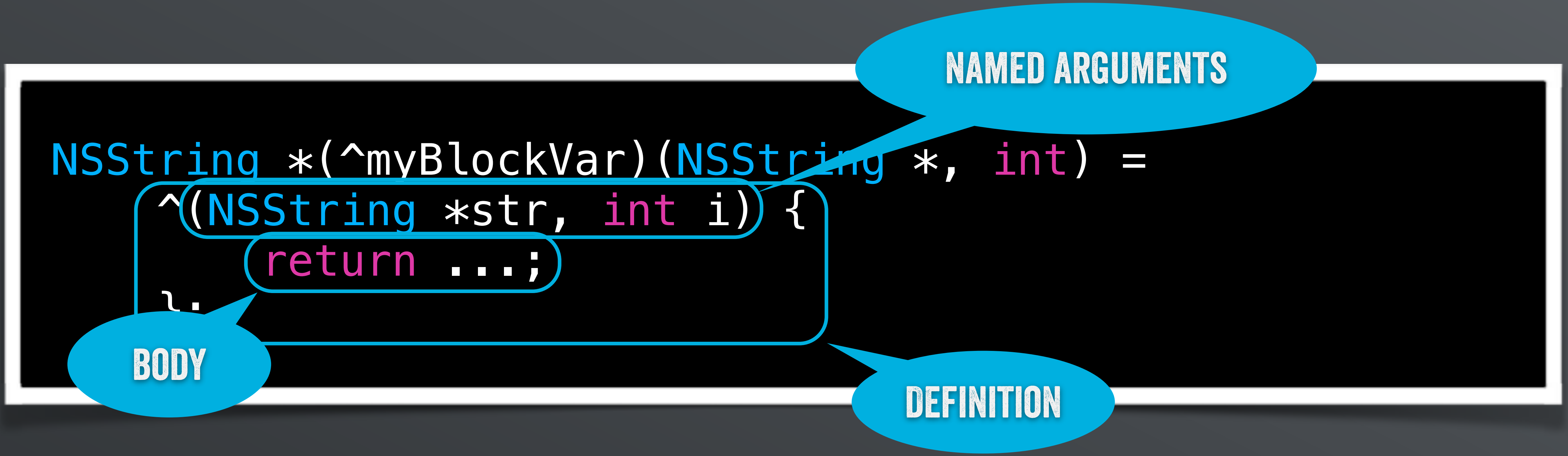
CLOSURE IN OBJECTIVE C: BLOCK

BLOCK DECLARATION & DEFINITION



```
NSString *(^myBlockVar)(NSString *, int) =
    ^(NSString *str, int i) {
        return ...;
    };
```

BLOCK DECLARATION & DEFINITION



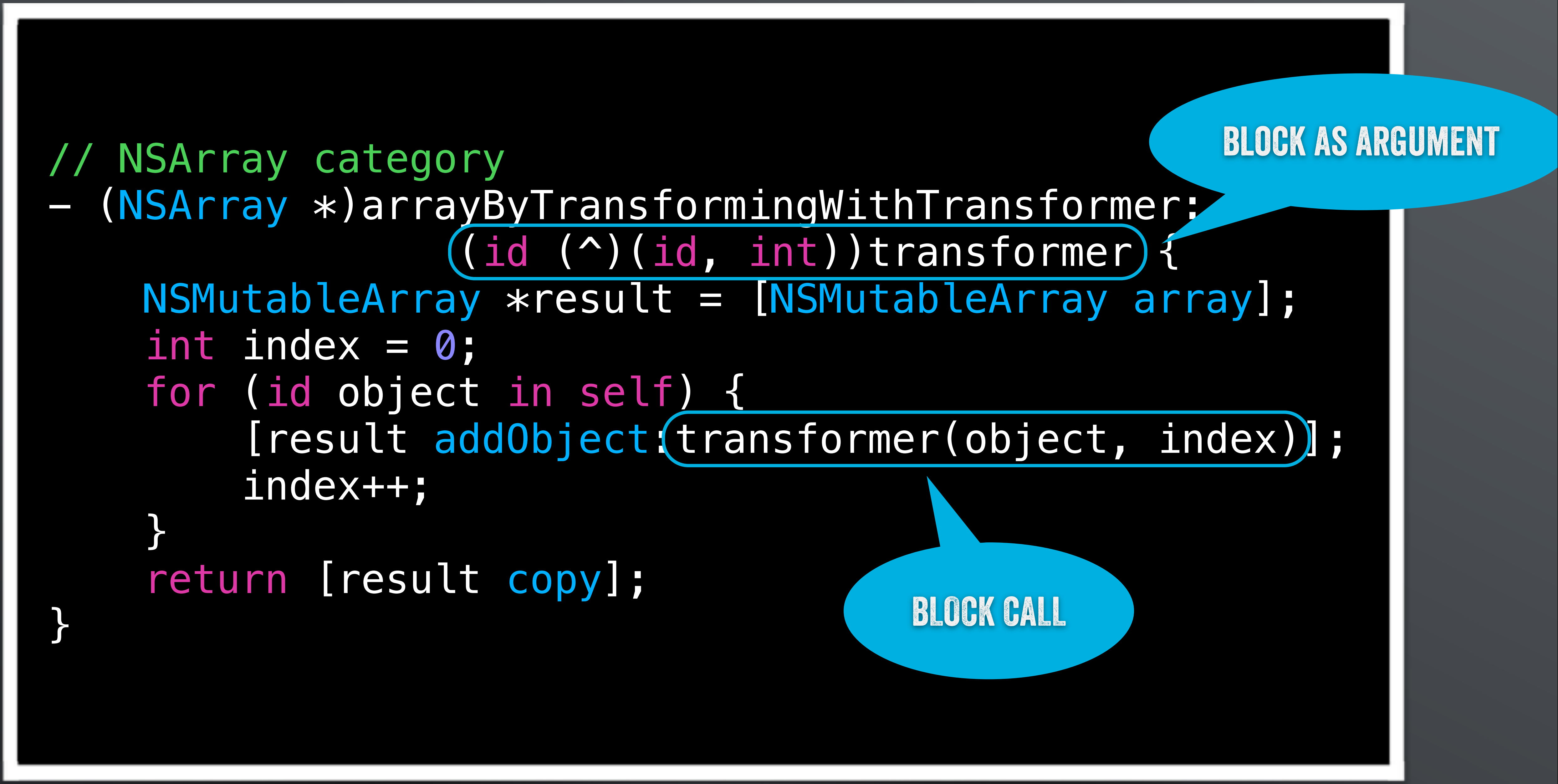
```
NSString *(^myBlockVar)(NSString *, int) =
    ^(NSString *str, int i) {
        return ...;
    };
```

USING A BLOCK DIRECTLY

```
NSString *(^myBlockVar)(NSString *, int) = ...

NSString *resultFromBlock = myBlockVar(@"argument", 10);
NSLog(@"Result: %@", resultFromBlock);
```

DIRECT BLOCK CALL

USING A BLOCK AS ARGUMENT



```
// NSArray category
- (NSArray *)arrayByTransformingWithTransformer:
                    (id (^)(id, int))transformer {
    NSMutableArray *result = [NSMutableArray array];
    int index = 0;
    for (id object in self) {
        [result addObject:transformer(object, index)];
        index++;
    }
    return [result copy];
}
```

USING A BLOCK AS ARGUMENT

```
// Somewhere in your code
NSArray *array = @[@"A", @"B", @"C"];
NSString *add = @"BC";
NSArray *transformed =
  [array arrayByTransformingWithTransformer:^id(id el, int i) {
    return [NSString stringWithFormat:@"%d. %@%@", i, el, add];
  }];
NSLog(@"Transformed: %@", transformed);


// Output
Transformed: (
    "0. ABC",
    "1. BBC",
    "2. CBC"
)
```

METHOD CALL

MUST WE BE THIS VERBOSE?

BLOCK TYPEDEF

TYPE NAME

```
// At the top of some .h file
typedef id(^Transformer)(id, int);

// In an @interface definition
- (NSArray *) arrayByTransformingWithTransformer:
                (Transformer) transformer;
```

USED AS
ARGUMENT TYPE

MORE ABOUT (LEXICAL) SCOPE

TYPES OF VARIABLES

```
static int globalVar = 1;

- (void) someMethod {
    int localVar = 2;
    __block int blockLocalVar = 3;
    void(^varAccessor)(int) = ^(int blockArg) {
        globalVar = 10;
        localVar = 20;
        blockLocalVar = 30;
        blockArg = 40;
    };
    globalVar = 11;
    localVar = 22;
    blockLocalVar = 33;
    varAccessor(4);
    globalVar = 111;
    localVar = 222;
    blockLocalVar = 333;
}
```

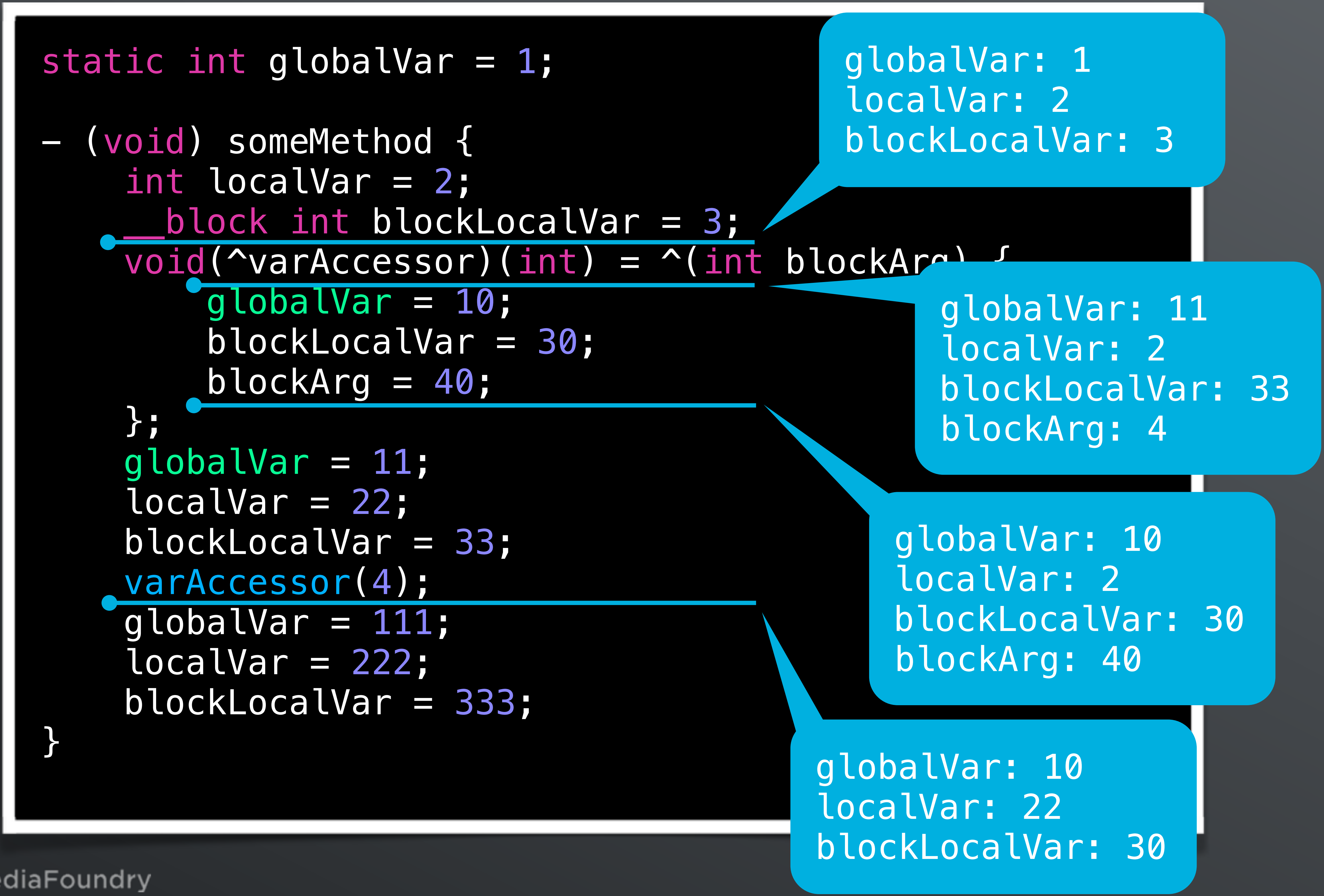
VARIABLE ACCESS

	READ	WRITE	CHANGE VISIBLE	VALUE
GLOBAL	✔	✔	✔	call time
LOCAL	✔	✗	/	creation time
__BLOCK	✔	✔	✔	call time
ARGUMENT	✔	✔	✗	/

TYPES OF VARIABLES

```
static int globalVar = 1;

- (void) someMethod {
    int localVar = 2;
    __block int blockLocalVar = 3;
    void(^varAccessor)(int) = ^(int blockArg) {
        globalVar = 10;
        ~~localVar = 20;~~
        blockLocalVar = 30;
        blockArg = 40;
    };
    globalVar = 11;
    localVar = 22;
    blockLocalVar = 33;
    varAccessor(4);
    globalVar = 111;
    localVar = 222;
    blockLocalVar = 333;
}
```

TYPES OF VARIABLES



```
static int globalVar = 1;

- (void) someMethod {
    int localVar = 2;
    __block int blockLocalVar = 3;
    void(^varAccessor)(int) = ^(int blockArg) {
        globalVar = 10;
        blockLocalVar = 30;
        blockArg = 40;
    };
    globalVar = 11;
    localVar = 22;
    blockLocalVar = 33;
    varAccessor(4);
    globalVar = 111;
    localVar = 222;
    blockLocalVar = 333;
}
```

```
globalVar: 1
localVar: 2
blockLocalVar: 3
```

```
globalVar: 11
localVar: 2
blockLocalVar: 33
blockArg: 4
```

```
globalVar: 10
localVar: 2
blockLocalVar: 30
blockArg: 40
```

```
globalVar: 10
localVar: 22
blockLocalVar: 30
```

BLOCKS IN IOS

UIKIT

```
//  UIView

+ (void)animateWithDuration:(NSTimeInterval)duration
          animations:(void (^)(void))animations
          completion:(void (^)(BOOL))completion;
```

FOUNDATION

```
//  NSArray / NSSet
- (void)enumerateObjectsUsingBlock:(void (^)(id, BOOL*))block;

//  NSDictionary
- (void)enumerateKeysAndObjectsUsingBlock:
        (void (^)(id, id, BOOL*))block;

//  NSURLConnection
+ (void)sendAsynchronousRequest:(NSURLRequest *)request
                          queue:(NSOperationQueue*)queue
             completionHandler:
        (void (^)(NSURLResponse*, NSData*, NSError*))handler;
```

GRAND CENTRAL DISPATCH

```
dispatch_async(dispatch_get_global_queue(
    DISPATCH_QUEUE_PRIORITY_DEFAULT, 0ul), ^{
        // Code will be executed concurrently

        // At the end, go back to the main queue
        dispatch_async(dispatch_get_main_queue(), ^{
            // Code will be scheduled on the main queue
        });
    });
```

NSOPERATION & NSOPERATIONQUEUE

```
NSBlockOperation *operation =
    [NSBlockOperation blockOperationWithBlock:^{
        // Some code here
    }];

NSOperationQueue *queue =
    [[NSOperationQueue alloc] init];
[queue addOperation:operation];
```

Q & A

CONTACT US

iDA MediaFoundry

info@ida-mediafoundry.be
@iDAMediaFoundry

michael.seghers@ida-mediafoundry.be
@mikeseghers