

# Analyzing and Transforming Native Code

Applied Static Analysis 2016

**Ben Hermann**  
**@benhermann**

Dr. Michael Eichberg, Johannes Lerch, Sebastian Proksch, Karim Ali Ph.D.



# Parameter Data-Flow Analysis

naive version

- Idea: Ignore control flow, path conditions, and calling context
- Iterate over instructions and see what happens

# Parameter Data-Flow Analysis

naive version

```
%tmp = alloca i32, align 4
```

```
%tmp1 = alloca i32, align 4
```

```
store i32 %n, i32* %tmp1, align 4
```

```
%tmp2 = load i32, i32* %tmp1, align 4
```

```
%tmp3 = icmp eq i32 %tmp2, 0
```

```
br i1 %tmp3, label %bb4, label %bb5
```

```
store i32 0, i32* %tmp
```

```
br label %bb17
```

```
%tmp6 = load i32, i32* %tmp1, align 4
```

```
%n
```

```
%n
```

```
%n, %tmp1
```

```
%n, %tmp1, %tmp2
```

```
%n, %tmp1, %tmp2
```

```
%n, %tmp1, %tmp2
```

```
%n, %tmp1, %tmp2
```

```
%n, %tmp1, %tmp2
```

```
%n, %tmp1, %tmp2, %tmp6
```

# Parameter Data-Flow Analysis

exercises/ParameterFlow/

- Iterate over all arguments of a function  
`for(Argument &a : f.getArgumentList())`
- Implement an `InstVisitor` for all relevant instructions

```
struct FlowIV : public InstVisitor<FlowIV>
```

- Store tracked `Value` instances in a `DenseSet`  
`DenseSet<Value*> trackedValues;`



# Parameter Data-Flow Analysis

exercises/ParameterFlow/

- Implement handler for all instructions relevant

```
void visitBinaryOperator(BinaryOperator &I) {  
    Value *op1 = I.getOperand(0);  
    Value *op2 = I.getOperand(1);  
  
    if (isTracked(op1) || isTracked(op2)) {  
        addTrackedValue(I);  
    }  
}
```

# Parameter Data-Flow Analysis

naive version

- What are the problems of this approach?
- Is the result correct/sound?
- Is the result precise?

# Control-Flow Sensitivity

- Consider this example:

```
int doStuff(int n) {  
    int result = 0;  
    if (false) {  
        result = n;  
    }  
    return result;  
}
```

- Does `n` flow to `result`?



# Making the Analysis Control-Flow Sensitive

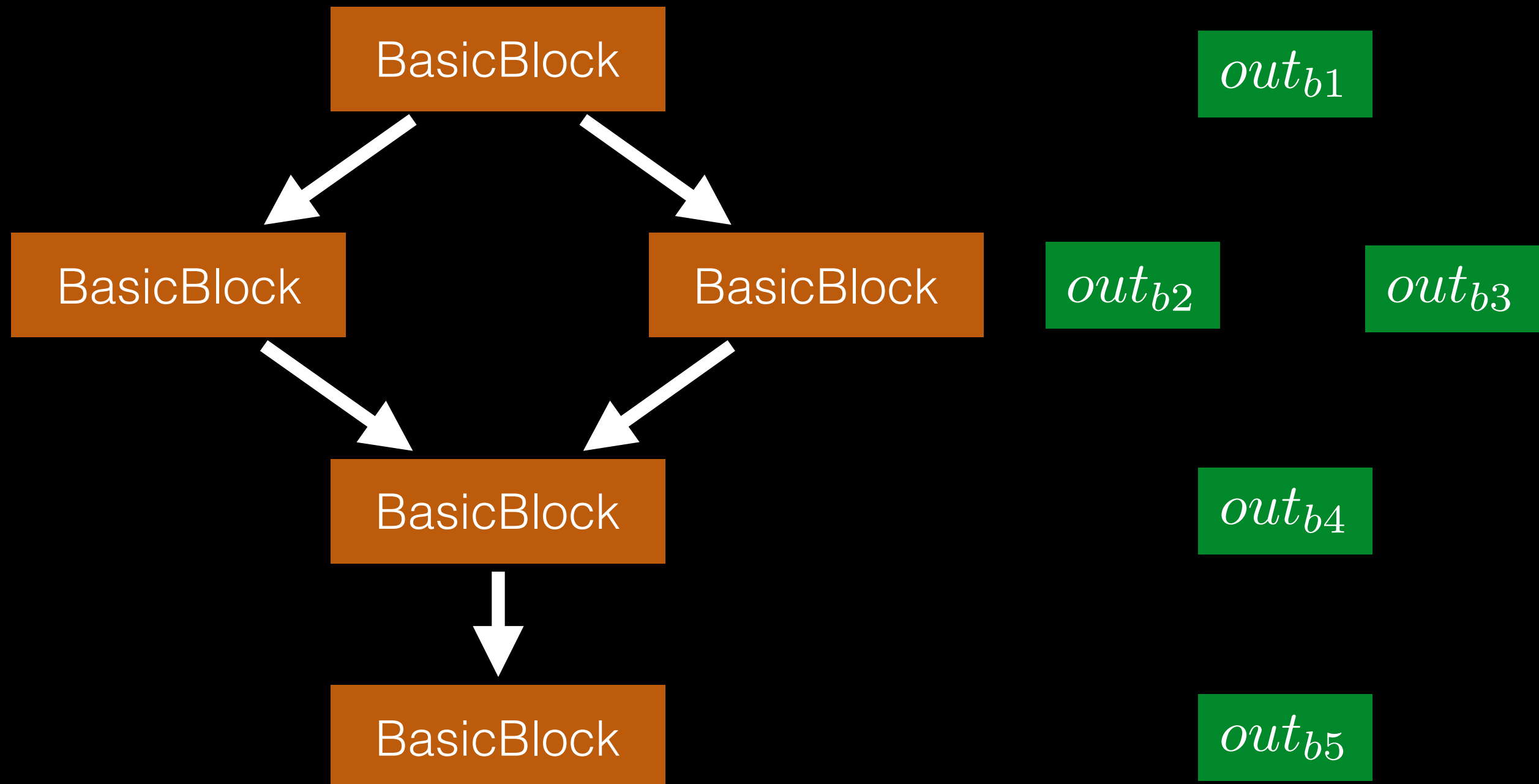
- Recall the following:

$$out_b = trans_b(in_b)$$

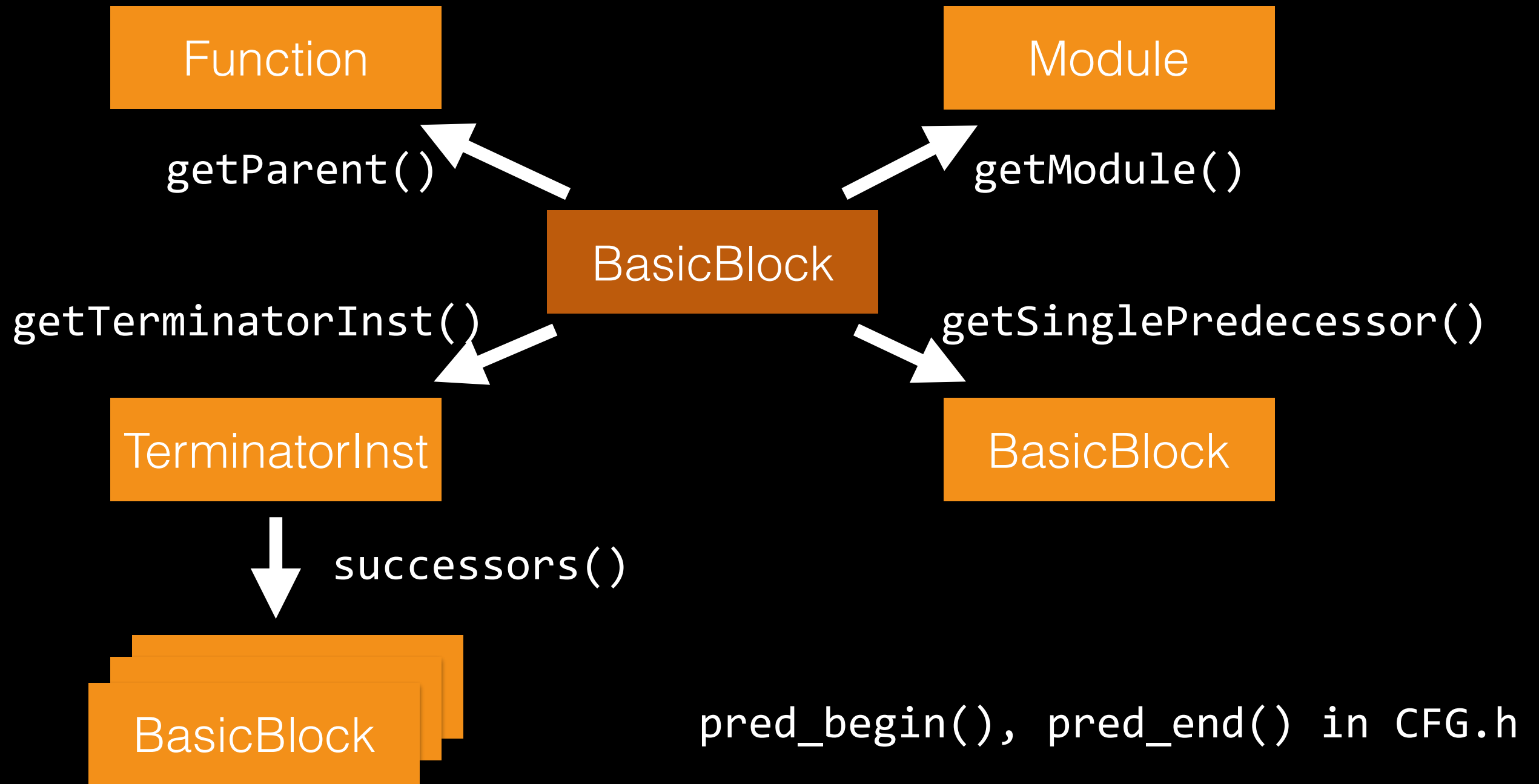
$$in_b = join_{p \in pred_b}(out_p)$$

- This is exactly what we want:
  - basic block separated
  - output is the translation of the input
  - joined output of predecessors is input the next

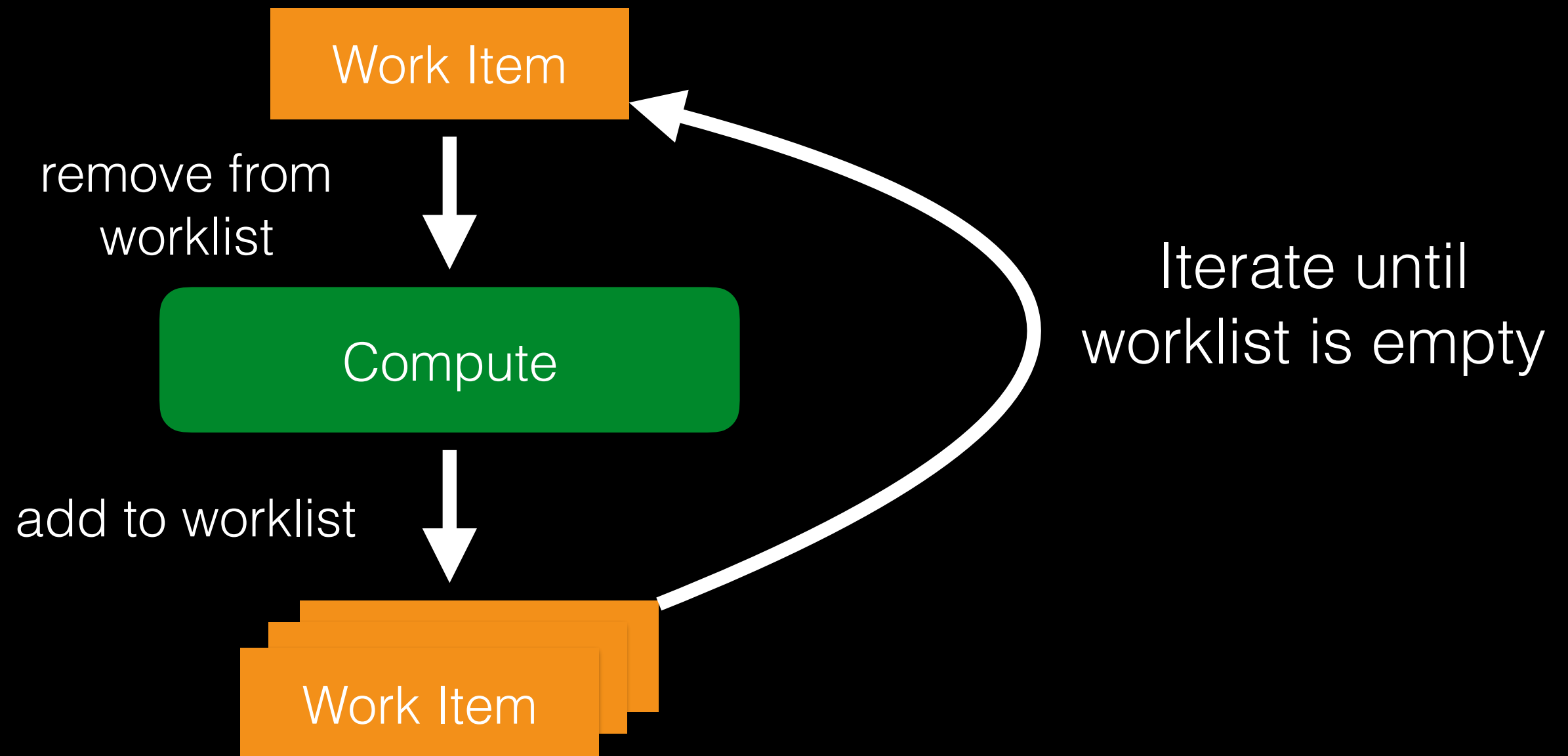
# Making the Analysis Control-Flow Sensitive



# Basic Block



# Worklist Algorithms



# Parameter Data-Flow Analysis

flow-sensitive version

```
std::vector<std::pair<BasicBlock*, DenseSet<Value*>>> worklist;
```

# Parameter Data-Flow Analysis

flow-sensitive version

```
std::vector<std::pair<BasicBlock*, DenseSet<Value*>>> worklist;
```

Standard vector data structure

# Parameter Data-Flow Analysis

flow-sensitive version

```
std::vector<std::pair<BasicBlock*, DenseSet<Value*>>> worklist;
```



# Parameter Data-Flow Analysis

flow-sensitive version

Standard pair

```
std::vector<std::pair<BasicBlock*, DenseSet<Value*>>> worklist;
```

# Parameter Data-Flow Analysis

flow-sensitive version

```
std::vector<std::pair<BasicBlock*, DenseSet<Value*>>> worklist;
```

# Parameter Data-Flow Analysis

flow-sensitive version

```
std::vector<std::pair<BasicBlock*, DenseSet<Value*>>> worklist;
```

Basic block to be processed

# Parameter Data-Flow Analysis

flow-sensitive version

```
std::vector<std::pair<BasicBlock*, DenseSet<Value*>>> worklist;
```

# Parameter Data-Flow Analysis

flow-sensitive version

Incoming Tracked Values

```
std::vector<std::pair<BasicBlock*, DenseSet<Value*>>> worklist;
```

# Parameter Data-Flow Analysis

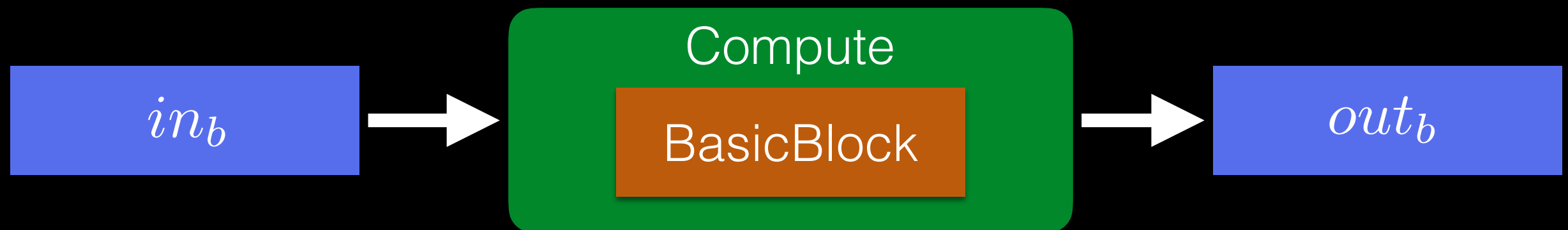
flow-sensitive version

```
std::vector<std::pair<BasicBlock*, DenseSet<Value*>>> worklist;
```

# Parameter Data-Flow Analysis

flow-sensitive version

```
std::vector<std::pair<BasicBlock*, DenseSet<Value*>>> worklist;
```





# Parameter Data-Flow Analysis

flow-sensitive version

Computing the initial seed

```
for (BasicBlock &bb : f) {
    if (!hasPredecessor(&bb)) {
        DenseSet<Value*> incomingValues;
        incomingValues.insert(&a);
        worklist.push_back(std::make_pair(&bb, incomingValues));
    }
}

// [...]
bool hasPredecessor(BasicBlock* bb) {
    if (pred_begin(bb) != pred_end(bb)) return true;
    return false;
}
```

**exercise 7.1**

# Parameter Data-Flow Analysis

flow-sensitive version

Computing the initial seed

```
for (BasicBlock &bb : f) {  
    if (!hasPredecessor(&bb)) {  
        DenseSet<Value*> incomingValues;  
        incomingValues.insert(&a);  
        worklist.push_back(std::make_pair(&bb, incomingValues));  
    }  
}
```

Get all BasicBlocks w/o predecessors

```
// [...]  
bool hasPredecessor(BasicBlock* bb) {  
    if (pred_begin(bb) != pred_end(bb)) return true;  
    return false;  
}
```

**exercise 7.1**

# Parameter Data-Flow Analysis

flow-sensitive version

Computing the initial seed

```
for (BasicBlock &bb : f) {
    if (!hasPredecessor(&bb)) {
        DenseSet<Value*> incomingValues;
        incomingValues.insert(&a);
        worklist.push_back(std::make_pair(&bb, incomingValues));
    }
}

// [...]
bool hasPredecessor(BasicBlock* bb) {
    if (pred_begin(bb) != pred_end(bb)) return true;
    return false;
}
```

**exercise 7.1**

# Parameter Data-Flow Analysis

flow-sensitive version

Computing the initial seed

```
for (BasicBlock &bb : f) {  
    if (!hasPredecessor(&bb)) {  
        DenseSet<Value*> incomingValues;  
        incomingValues.insert(&a);  
        worklist.push_back(std::make_pair(&bb, incomingValues));  
    }  
}
```

```
// [...]  
bool hasPredecessor(BasicBlock* bb) {  
    if (pred_begin(bb) != pred_end(bb)) return true;  
    return false;  
}
```

End and begin pointers differ

# Parameter Data-Flow Analysis

flow-sensitive version

Computing the initial seed

```
for (BasicBlock &bb : f) {
    if (!hasPredecessor(&bb)) {
        DenseSet<Value*> incomingValues;
        incomingValues.insert(&a);
        worklist.push_back(std::make_pair(&bb, incomingValues));
    }
}

// [...]
bool hasPredecessor(BasicBlock* bb) {
    if (pred_begin(bb) != pred_end(bb)) return true;
    return false;
}
```

**exercise 7.1**

# Parameter Data-Flow Analysis

flow-sensitive version

Computing the initial seed

```
for (BasicBlock &bb : f) {  
    if (!hasPredecessor(&bb)) {  
        DenseSet<Value*> incomingValues;  
        incomingValues.insert(&a);  
        worklist.push_back(std::make_pair(&bb, incomingValues));  
    }  
}  
  
// [...]  
bool hasPredecessor(BasicBlock* bb) {  
    if (pred_begin(bb) != pred_end(bb)) return true;  
    return false;  
}
```

Assume the argument as incoming

**exercise 7.1**

# Parameter Data-Flow Analysis

flow-sensitive version

Computing the initial seed

```
for (BasicBlock &bb : f) {
    if (!hasPredecessor(&bb)) {
        DenseSet<Value*> incomingValues;
        incomingValues.insert(&a);
        worklist.push_back(std::make_pair(&bb, incomingValues));
    }
}

// [...]
bool hasPredecessor(BasicBlock* bb) {
    if (pred_begin(bb) != pred_end(bb)) return true;
    return false;
}
```

**exercise 7.1**



# Parameter Data-Flow Analysis

flow-sensitive version

Computing the initial seed

```
for (BasicBlock &bb : f) {  
    if (!hasPredecessor(&bb)) {  
        DenseSet<Value*> incomingValues;  
        incomingValues.insert(&a);  
        worklist.push_back(std::make_pair(&bb, incomingValues));  
    }  
}
```

Add pair to work list

```
// [...]  
bool hasPredecessor(BasicBlock* bb) {  
    if (pred_begin(bb) != pred_end(bb)) return true;  
    return false;  
}
```

**exercise 7.1**

# Parameter Data-Flow Analysis

flow-sensitive version

Computing the initial seed

```
for (BasicBlock &bb : f) {
    if (!hasPredecessor(&bb)) {
        DenseSet<Value*> incomingValues;
        incomingValues.insert(&a);
        worklist.push_back(std::make_pair(&bb, incomingValues));
    }
}

// [...]
bool hasPredecessor(BasicBlock* bb) {
    if (pred_begin(bb) != pred_end(bb)) return true;
    return false;
}
```

**exercise 7.1**

# Parameter Data-Flow Analysis

flow-sensitive version

## Worklist processing

```
while(!worklist.empty()) {
    std::pair<BasicBlock*, DenseSet<Value*>> current = worklist.back();
    worklist.pop_back();

    FlowIV flowVisitor;
    for (Value *v : current.second)
        flowVisitor.addTrackedValue(v);
    flowVisitor.visit(current.first);
    DenseSet<Value*> outgoingTracked = flowVisitor.getCurrentlyTrackedValues();

    TerminatorInst *blockTi = current.first->getTerminator();
    if (blockTi) {
        for(BasicBlock *succ : blockTi->successors()) {
            worklist.push_back(std::make_pair(succ, outgoingTracked));
        }
    }
}
```

**exercise 7.1**

# Parameter Data-Flow Analysis

flow-sensitive version

## Worklist processing

```
while(!worklist.empty()) {
    std::pair<BasicBlock*, DenseSet<Value*>> current = worklist.back();
    worklist.pop_back();

    FlowIV flowVisitor;
    for (Value *v : current.second)
        flowVisitor.addTrackedValue(v);
    flowVisitor.visit(current.first);
    DenseSet<Value*> outgoingTracked = flowVisitor.getCurrentlyTrackedValues();

    TerminatorInst *blockTi = current.first->getTerminator();
    if (blockTi) {
        for(BasicBlock *succ : blockTi->successors()) {
            worklist.push_back(std::make_pair(succ, outgoingTracked));
        }
    }
}
```

Get the current item and pop it from the worklist

**exercise 7.1**

# Parameter Data-Flow Analysis

flow-sensitive version

## Worklist processing

```
while(!worklist.empty()) {
    std::pair<BasicBlock*, DenseSet<Value*>> current = worklist.back();
    worklist.pop_back();

    FlowIV flowVisitor;
    for (Value *v : current.second)
        flowVisitor.addTrackedValue(v);
    flowVisitor.visit(current.first);
    DenseSet<Value*> outgoingTracked = flowVisitor.getCurrentlyTrackedValues();

    TerminatorInst *blockTi = current.first->getTerminator();
    if (blockTi) {
        for(BasicBlock *succ : blockTi->successors()) {
            worklist.push_back(std::make_pair(succ, outgoingTracked));
        }
    }
}
```

**exercise 7.1**

# Parameter Data-Flow Analysis

flow-sensitive version

## Worklist processing

```
while(!worklist.empty()) {
    std::pair<BasicBlock*, DenseSet<Value*>> current = worklist.back();
    worklist.pop_back();

    FlowIV flowVisitor;
    for (Value *v : current.second)
        flowVisitor.addTrackedValue(v);
    flowVisitor.visit(current.first);
    DenseSet<Value*> outgoingTracked = flowVisitor.getCurrentlyTrackedValues();

    TerminatorInst *blockTi = current.first->getTerminator();
    if (blockTi) {
        for(BasicBlock *succ : blockTi->successors()) {
            worklist.push_back(std::make_pair(succ, outgoingTracked));
        }
    }
}
```

Reuse the FlowIV from before

**exercise 7.1**

# Parameter Data-Flow Analysis

flow-sensitive version

## Worklist processing

```
while(!worklist.empty()) {
    std::pair<BasicBlock*, DenseSet<Value*>> current = worklist.back();
    worklist.pop_back();

    FlowIV flowVisitor;
    for (Value *v : current.second)
        flowVisitor.addTrackedValue(v);
    flowVisitor.visit(current.first);
    DenseSet<Value*> outgoingTracked = flowVisitor.getCurrentlyTrackedValues();

    TerminatorInst *blockTi = current.first->getTerminator();
    if (blockTi) {
        for(BasicBlock *succ : blockTi->successors()) {
            worklist.push_back(std::make_pair(succ, outgoingTracked));
        }
    }
}
```

**exercise 7.1**



# Parameter Data-Flow Analysis

flow-sensitive version

## Worklist processing

```
while(!worklist.empty()) {
    std::pair<BasicBlock*, DenseSet<Value*>> current = worklist.back();
    worklist.pop_back();

    FlowIV flowVisitor;
    for (Value *v : current.second)
        flowVisitor.addTrackedValue(v);
    flowVisitor.visit(current.first);
    DenseSet<Value*> outgoingTracked = flowVisitor.getCurrentlyTrackedValues();

    TerminatorInst *blockTi = current.first->getTerminator();
    if (blockTi) {
        for(BasicBlock *succ : blockTi->successors()) {
            worklist.push_back(std::make_pair(succ, outgoingTracked));
        }
    }
}
```

Add initial values

**exercise 7.1**

# Parameter Data-Flow Analysis

flow-sensitive version

## Worklist processing

```
while(!worklist.empty()) {
    std::pair<BasicBlock*, DenseSet<Value*>> current = worklist.back();
    worklist.pop_back();

    FlowIV flowVisitor;
    for (Value *v : current.second)
        flowVisitor.addTrackedValue(v);
    flowVisitor.visit(current.first);
    DenseSet<Value*> outgoingTracked = flowVisitor.getCurrentlyTrackedValues();

    TerminatorInst *blockTi = current.first->getTerminator();
    if (blockTi) {
        for(BasicBlock *succ : blockTi->successors()) {
            worklist.push_back(std::make_pair(succ, outgoingTracked));
        }
    }
}
```

**exercise 7.1**

# Parameter Data-Flow Analysis

flow-sensitive version

## Worklist processing

```
while(!worklist.empty()) {
    std::pair<BasicBlock*, DenseSet<Value*>> current = worklist.back();
    worklist.pop_back();

    FlowIV flowVisitor;
    for (Value *v : current.second)
        flowVisitor.addTrackedValue(v);
    flowVisitor.visit(current.first);
    DenseSet<Value*> outgoingTracked = flowVisitor.getCurrentlyTrackedValues();

    TerminatorInst *blockTi = current.first->getTerminator();
    if (blockTi) {
        for(BasicBlock *succ : blockTi->successors()) {
            worklist.push_back(std::make_pair(succ, outgoingTracked));
        }
    }
}
```

Compute outgoing values

**exercise 7.1**

# Parameter Data-Flow Analysis

flow-sensitive version

## Worklist processing

```
while(!worklist.empty()) {
    std::pair<BasicBlock*, DenseSet<Value*>> current = worklist.back();
    worklist.pop_back();

    FlowIV flowVisitor;
    for (Value *v : current.second)
        flowVisitor.addTrackedValue(v);
    flowVisitor.visit(current.first);
    DenseSet<Value*> outgoingTracked = flowVisitor.getCurrentlyTrackedValues();

    TerminatorInst *blockTi = current.first->getTerminator();
    if (blockTi) {
        for(BasicBlock *succ : blockTi->successors()) {
            worklist.push_back(std::make_pair(succ, outgoingTracked));
        }
    }
}
```

**exercise 7.1**

# Parameter Data-Flow Analysis

flow-sensitive version

## Worklist processing

```
while(!worklist.empty()) {
    std::pair<BasicBlock*, DenseSet<Value*>> current = worklist.back();
    worklist.pop_back();

    FlowIV flowVisitor;
    for (Value *v : current.second)
        flowVisitor.addTrackedValue(v);
    flowVisitor.visit(current.first);
    DenseSet<Value*> outgoingTracked = flowVisitor.getCurrentlyTrackedValues();

    TerminatorInst *blockTi = current.first->getTerminator();
    if (blockTi) {
        for(BasicBlock *succ : blockTi->successors()) {
            worklist.push_back(std::make_pair(succ, outgoingTracked));
        }
    }
}
```

Iterate over successor block

# Parameter Data-Flow Analysis

flow-sensitive version

## Worklist processing

```
while(!worklist.empty()) {
    std::pair<BasicBlock*, DenseSet<Value*>> current = worklist.back();
    worklist.pop_back();

    FlowIV flowVisitor;
    for (Value *v : current.second)
        flowVisitor.addTrackedValue(v);
    flowVisitor.visit(current.first);
    DenseSet<Value*> outgoingTracked = flowVisitor.getCurrentlyTrackedValues();

    TerminatorInst *blockTi = current.first->getTerminator();
    if (blockTi) {
        for(BasicBlock *succ : blockTi->successors()) {
            worklist.push_back(std::make_pair(succ, outgoingTracked));
        }
    }
}
```

**exercise 7.1**

# Parameter Data-Flow Analysis

flow-sensitive version

## Worklist processing

```
while(!worklist.empty()) {
    std::pair<BasicBlock*, DenseSet<Value*>> current = worklist.back();
    worklist.pop_back();

    FlowIV flowVisitor;
    for (Value *v : current.second)
        flowVisitor.addTrackedValue(v);
    flowVisitor.visit(current.first);
    DenseSet<Value*> outgoingTracked = flowVisitor.getCurrentlyTrackedValues();

    TerminatorInst *blockTi = current.first->getTerminator();
    if (blockTi) {
        for(BasicBlock *succ : blockTi->successors()) {
            worklist.push_back(std::make_pair(succ, outgoingTracked));
        }
    }
}
```

Push successor blocks to work list

**exercise 7.1**

# Parameter Data-Flow Analysis

flow-sensitive version

## Worklist processing

```
while(!worklist.empty()) {
    std::pair<BasicBlock*, DenseSet<Value*>> current = worklist.back();
    worklist.pop_back();

    FlowIV flowVisitor;
    for (Value *v : current.second)
        flowVisitor.addTrackedValue(v);
    flowVisitor.visit(current.first);
    DenseSet<Value*> outgoingTracked = flowVisitor.getCurrentlyTrackedValues();

    TerminatorInst *blockTi = current.first->getTerminator();
    if (blockTi) {
        for(BasicBlock *succ : blockTi->successors()) {
            worklist.push_back(std::make_pair(succ, outgoingTracked));
        }
    }
}
```

**exercise 7.1**



# What's wrong in this implementation?

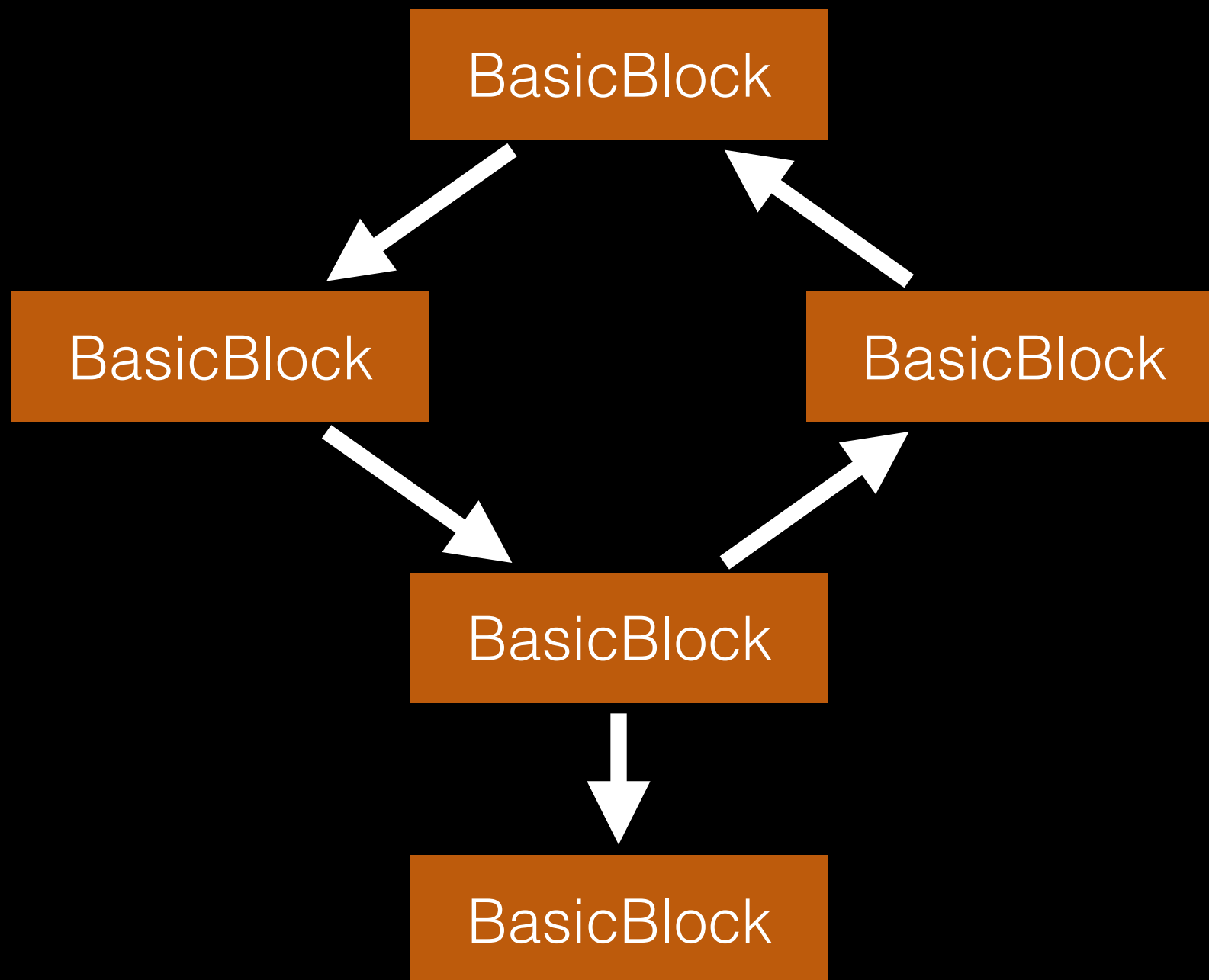
```
while(!worklist.empty()) {
    std::pair<BasicBlock*, DenseSet<Value*>> current = worklist.back();
    worklist.pop_back();

    FlowIV flowVisitor;
    for (Value *v : current.second)
        flowVisitor.addTrackedValue(v);
    flowVisitor.visit(current.first);
    DenseSet<Value*> outgoingTracked = flowVisitor.getCurrentlyTrackedValues();

    TerminatorInst *blockTi = current.first->getTerminator();
    if (blockTi) {
        for(BasicBlock *succ : blockTi->successors()) {
            worklist.push_back(std::make_pair(succ, outgoingTracked));
        }
    }
}
```

# Loops!

# Loops of Basic Blocks



# Parameter Data-Flow Analysis

flow-sensitive version

## Strategies for Handling Loops

- Only visit a BasicBlock once
- Only visit with different input flow
- Only push to worklist, if output set was not seen before

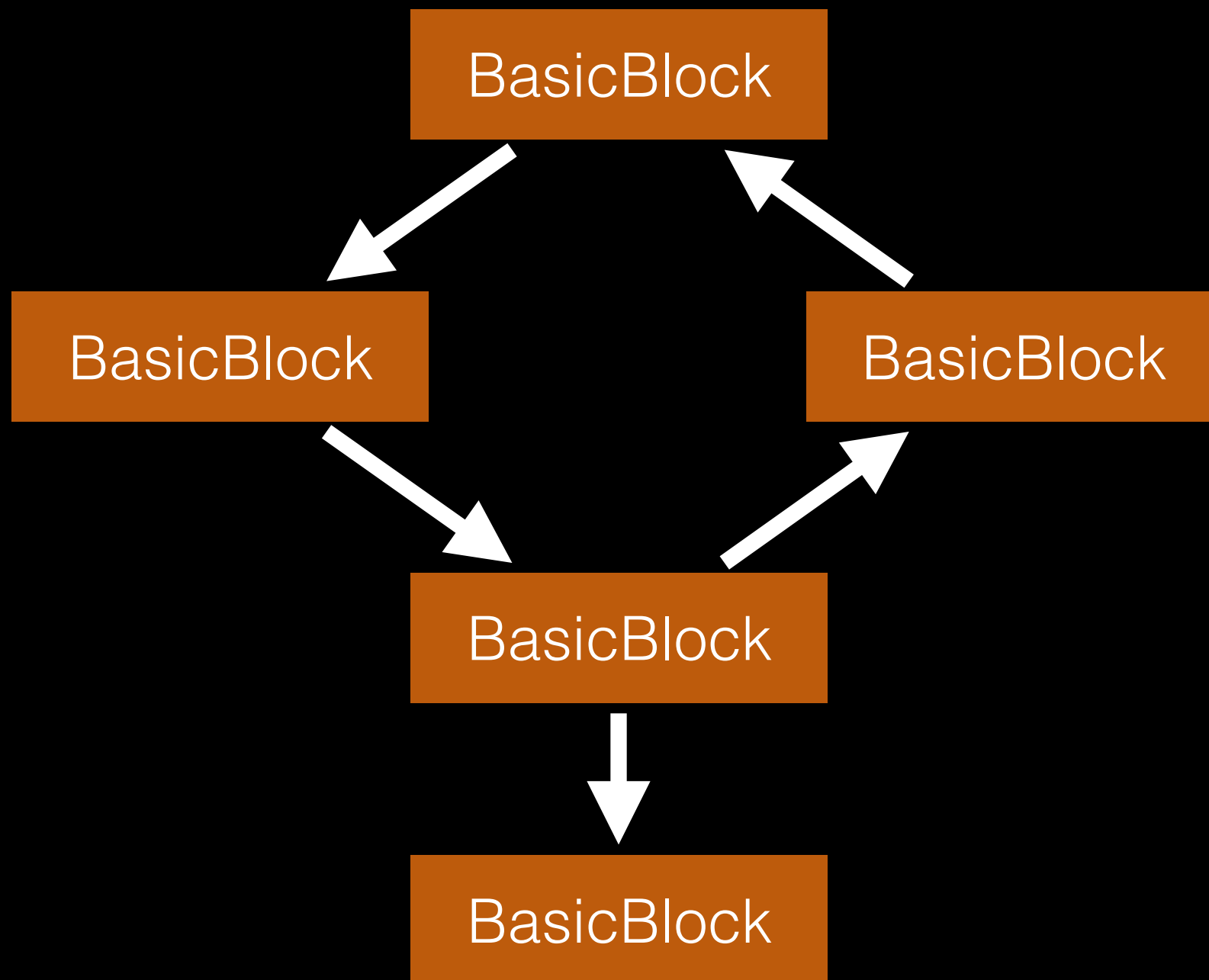
# Parameter Data-Flow Analysis

flow-sensitive version

## Strategies for Handling Loops 1. Process Blocks Once

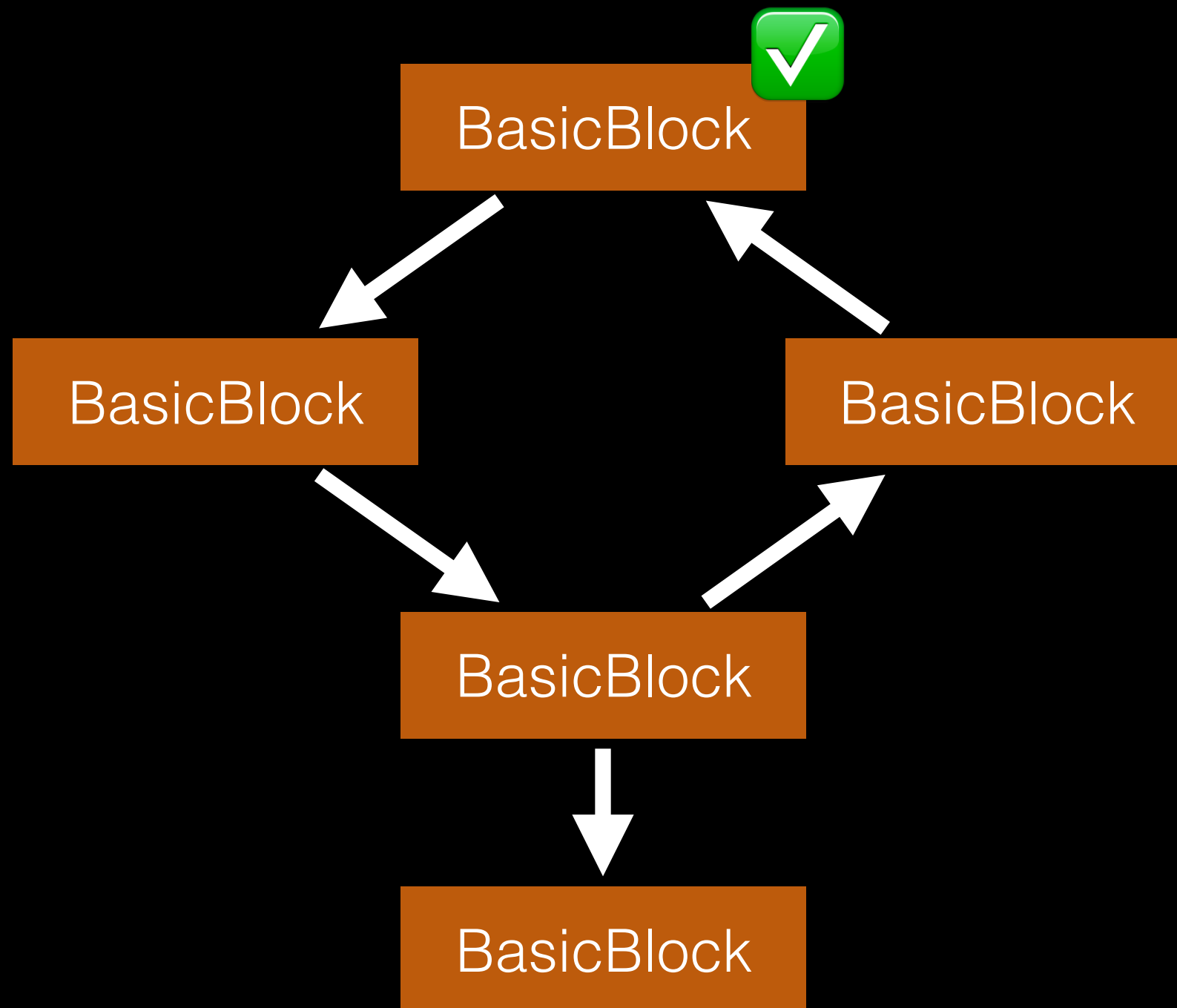
```
DenseSet<BasicBlock*> processed;  
  
// compute outgoing values  
processed.insert(current.first);  
  
if (processed.find(succ) == processed.end())  
    // add to worklist
```

# Strategy #1



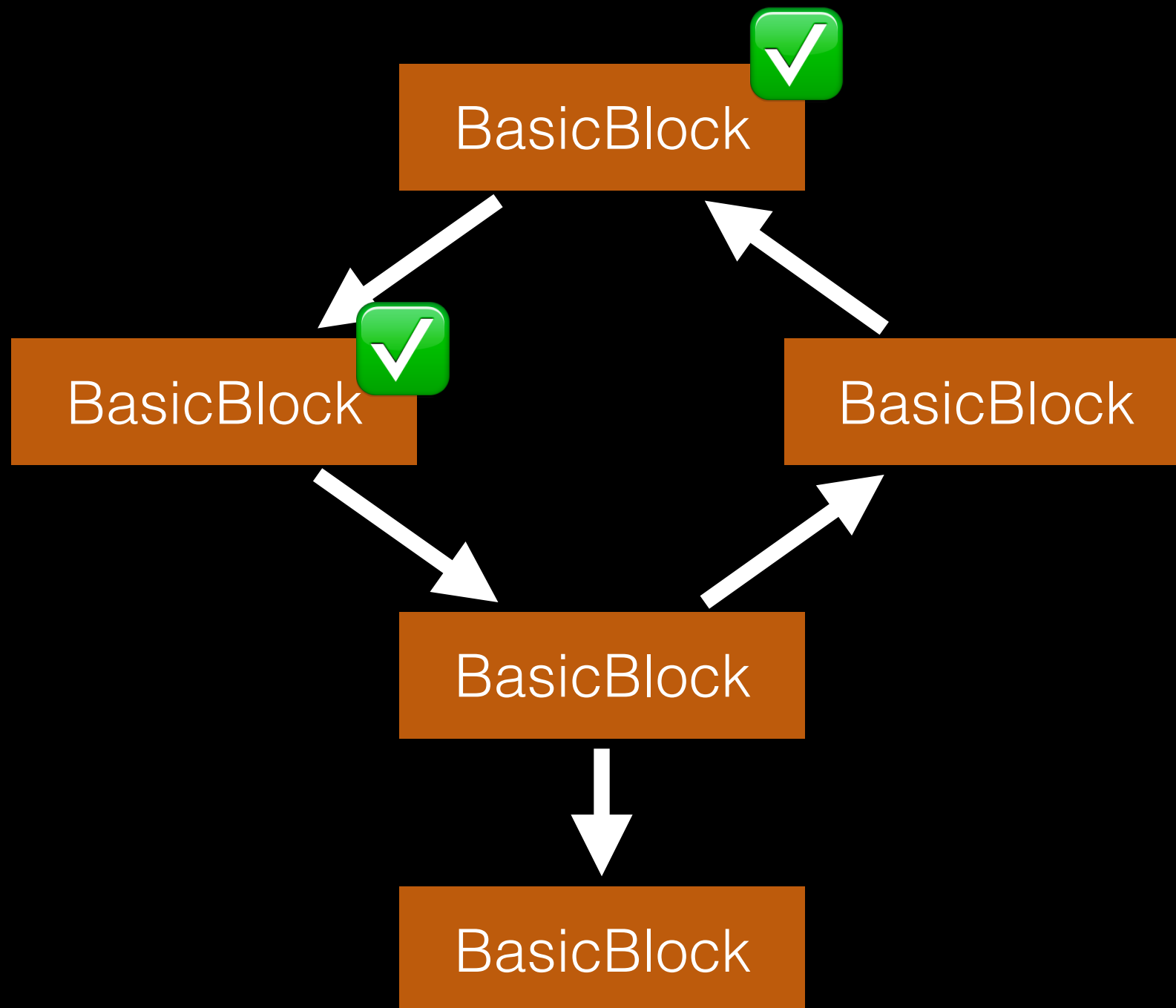
**exercise 7.2a**

# Strategy #1



**exercise 7.2a**

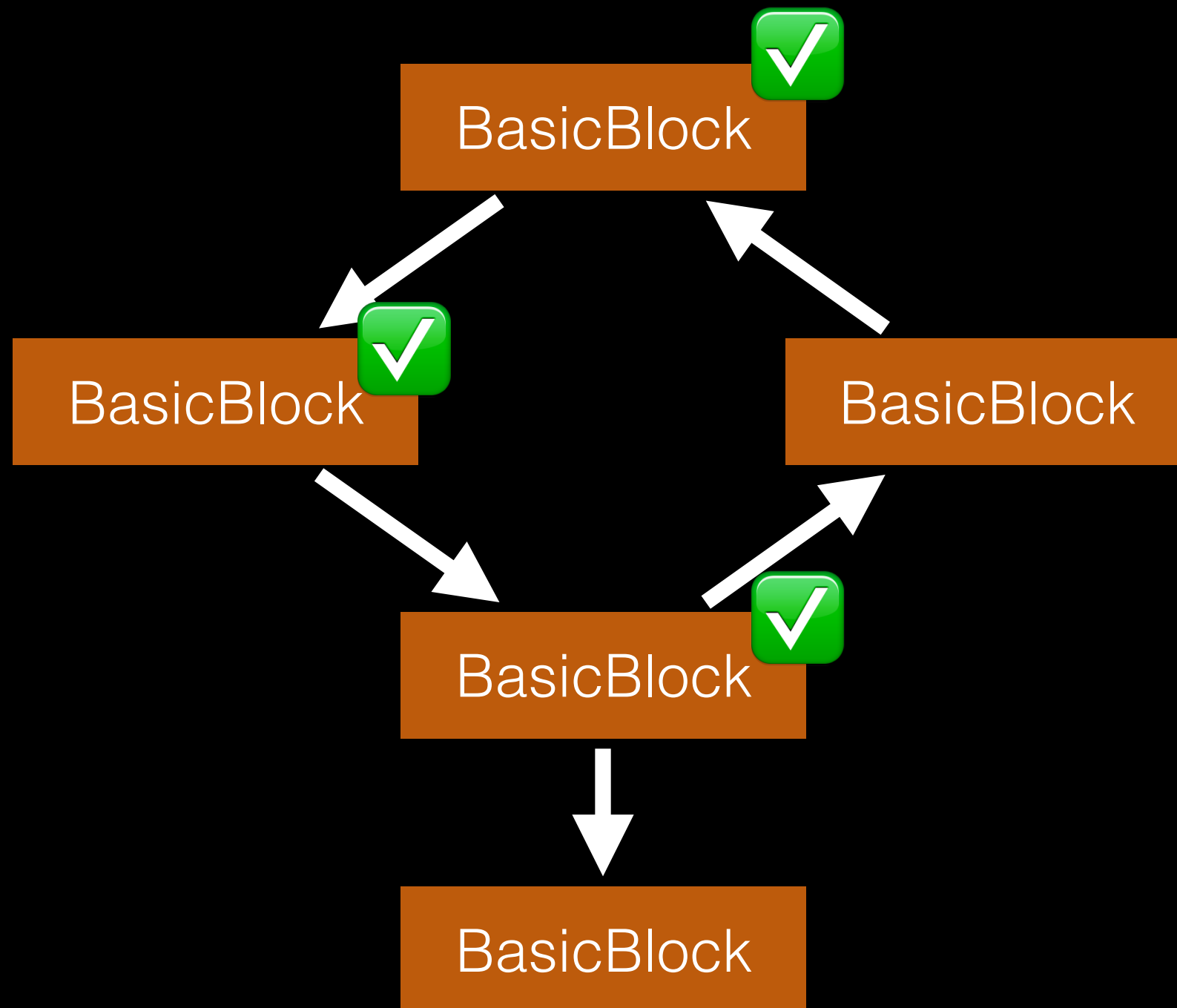
# Strategy #1



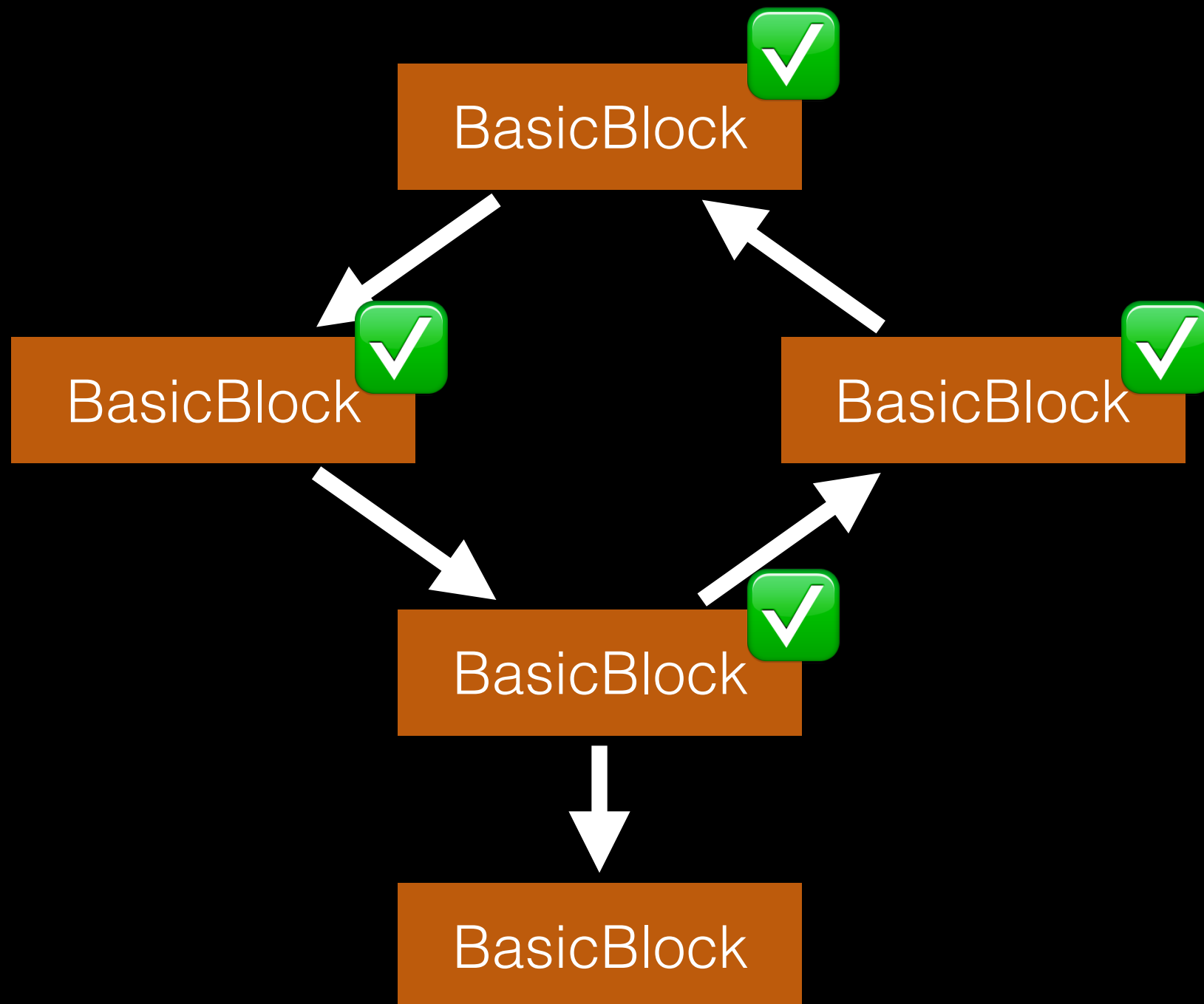
**exercise 7.2a**



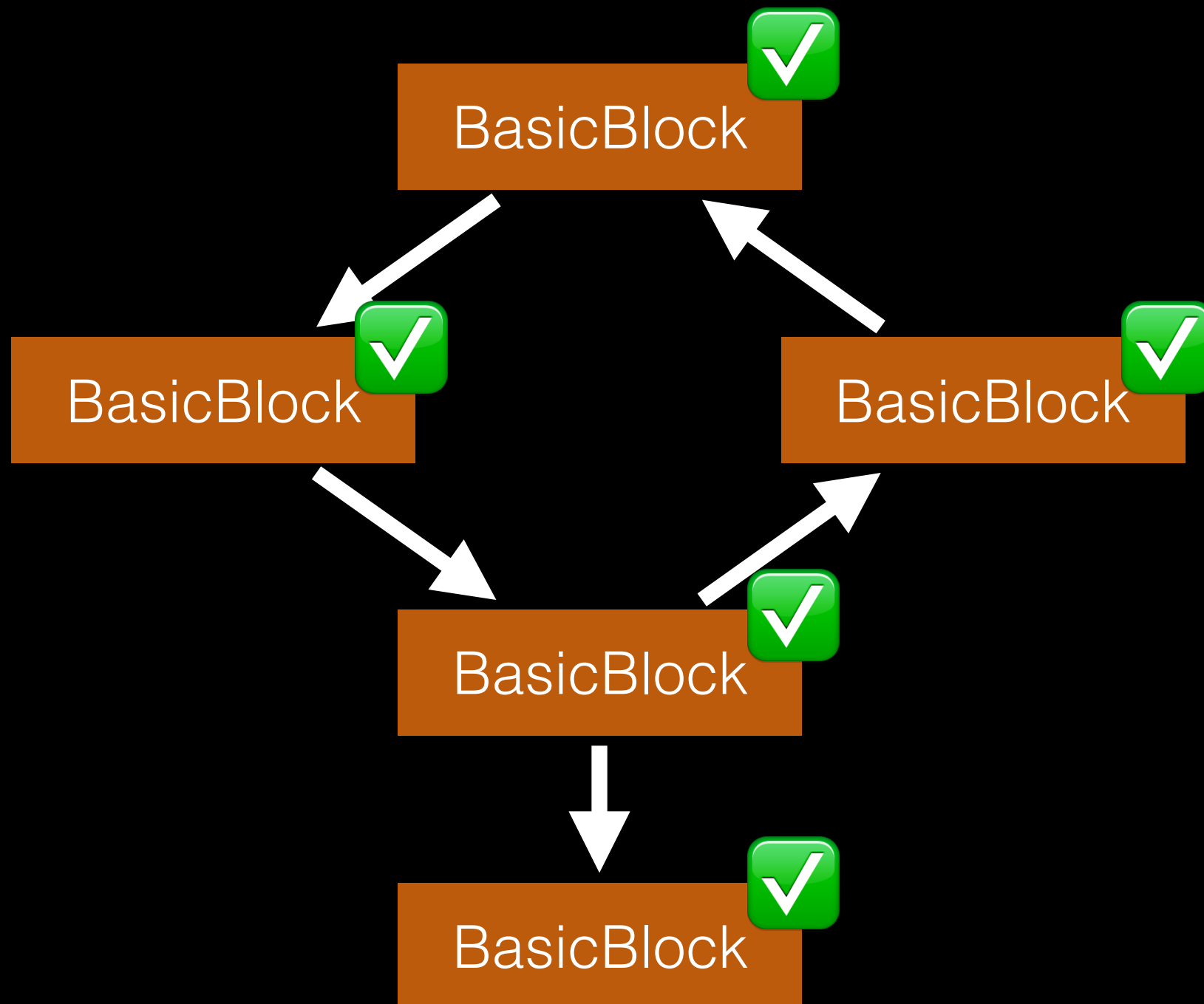
# Strategy #1



# Strategy #1



# Strategy #1



# Parameter Data-Flow Analysis

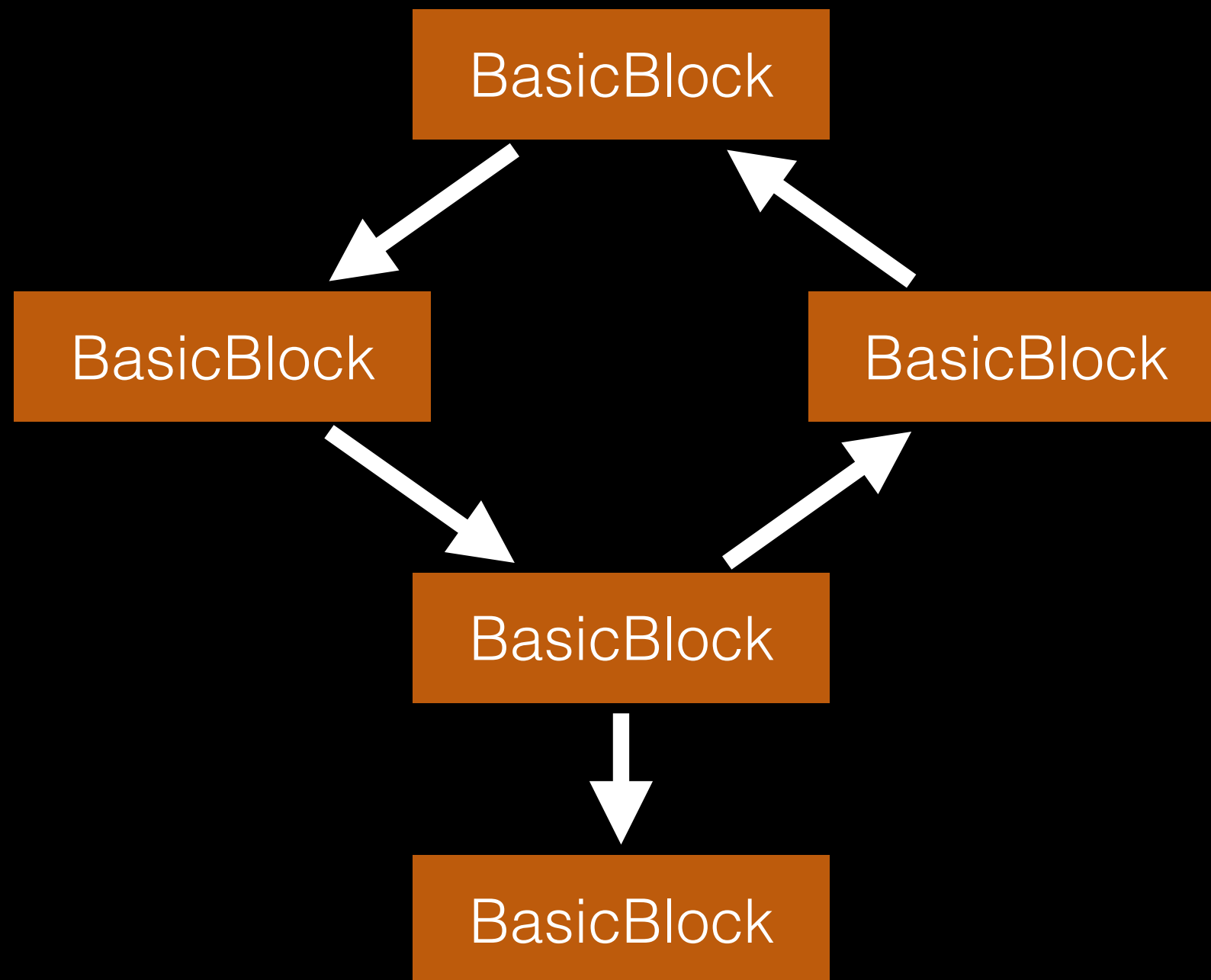
flow-sensitive version

## Strategies for Handling Loops 2. Consider Input Flows

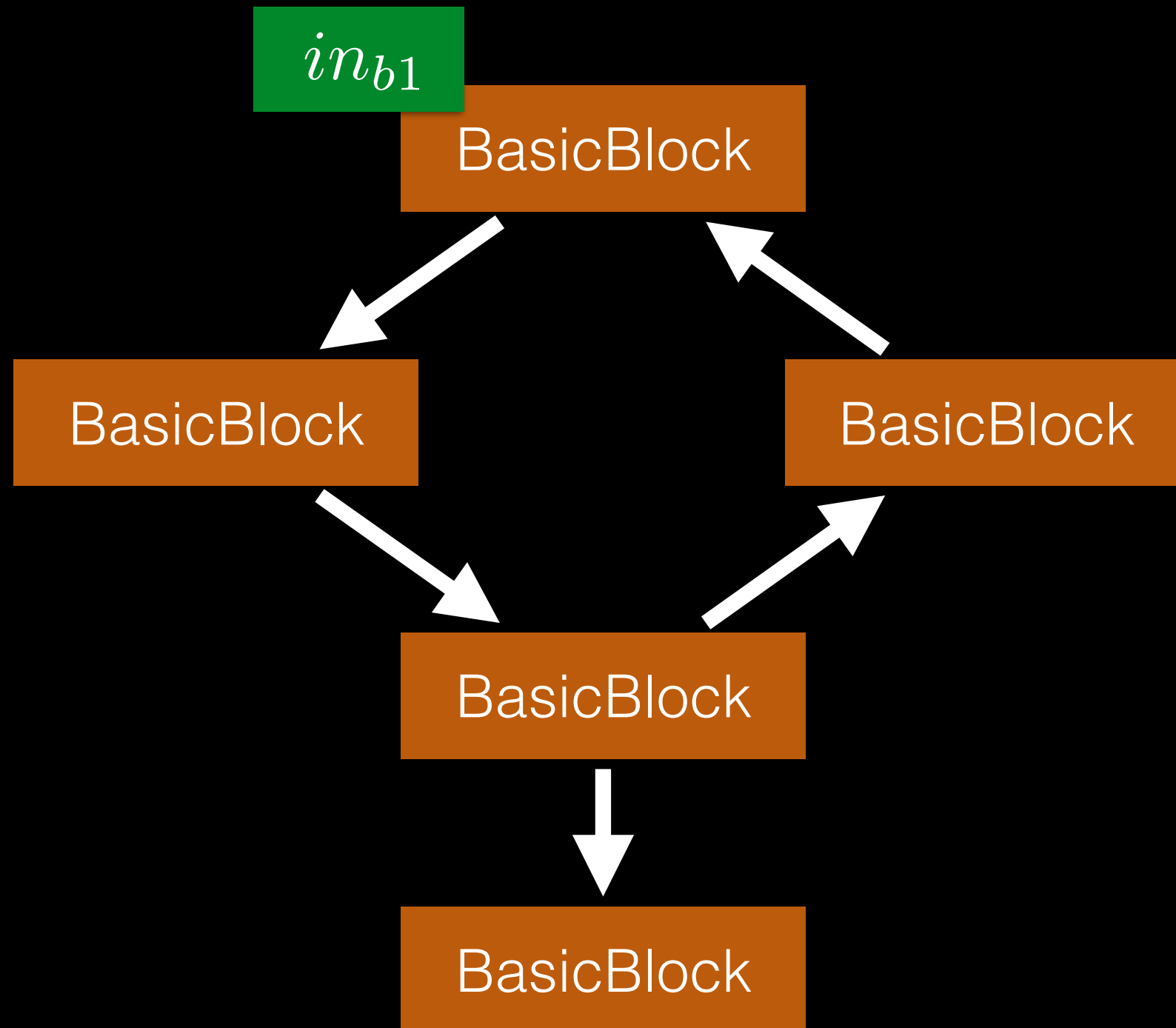
```
DenseMap<BasicBlock*, DenseSet<Value*>> knownInputs;  
  
DenseSet<Value*> processedBeforeWith = knownInputs.lookup(current.first);  
if (processedBeforeWith.size() > 0 &&  
    valueSetsAreEqual(current.second, processedBeforeWith))  
    continue;  
  
if(knownInputs.count(current.first) == 1) knownInputs.erase(current.first);  
knownInputs.insert(std::make_pair(current.first, current.second));
```

**exercise 7.2b**

# Strategy #2

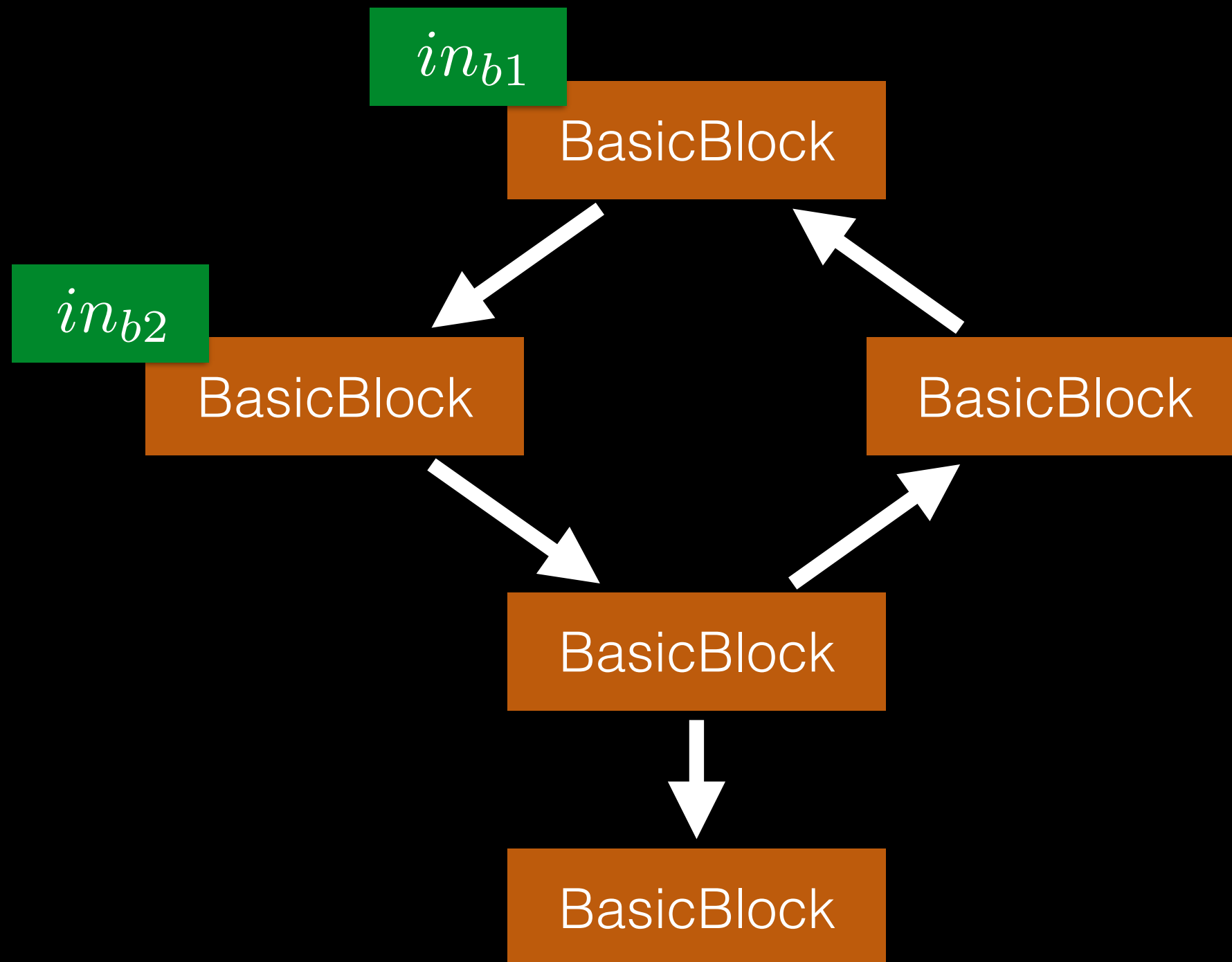


# Strategy #2



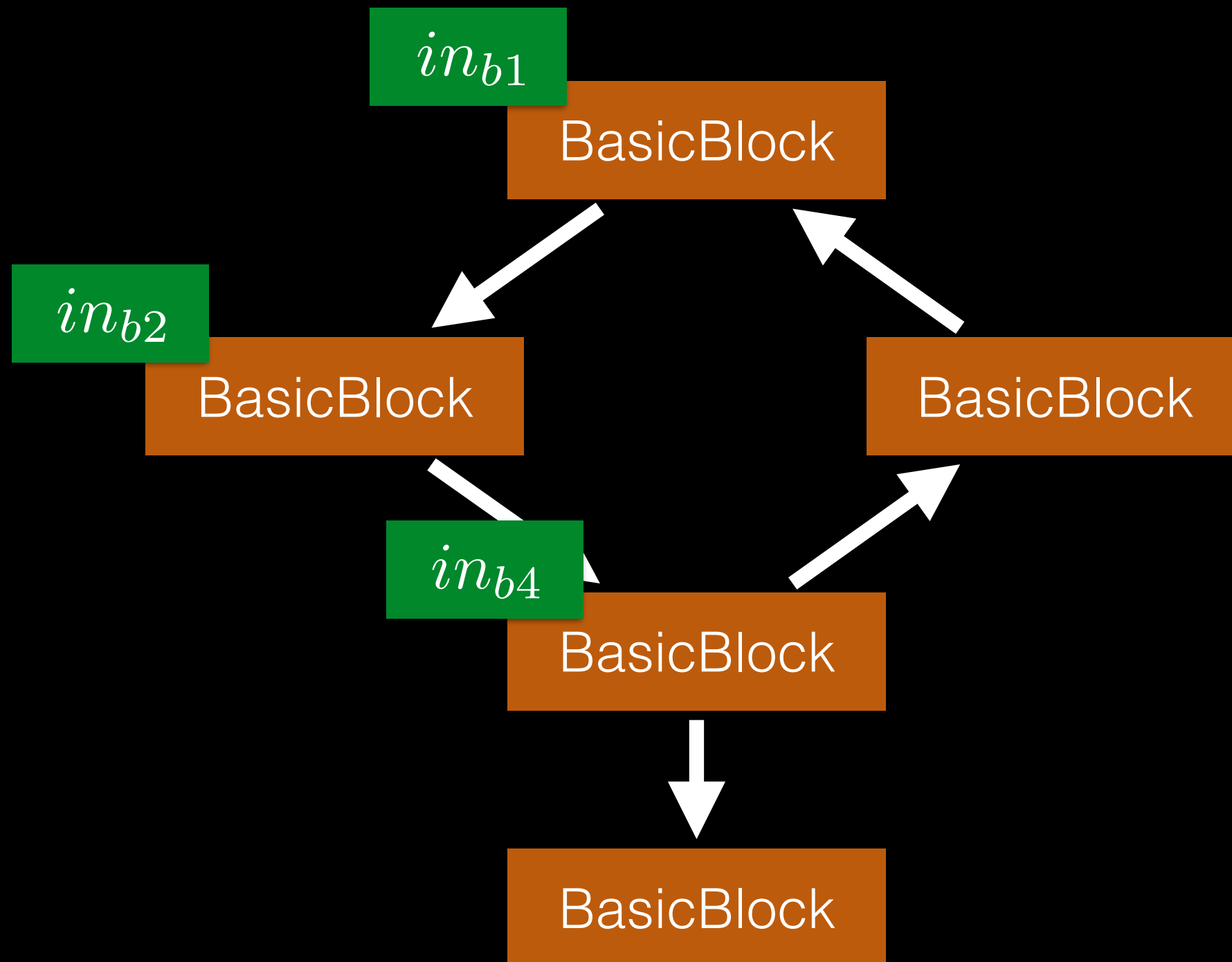
**exercise 7.2b**

# Strategy #2



**exercise 7.2b**

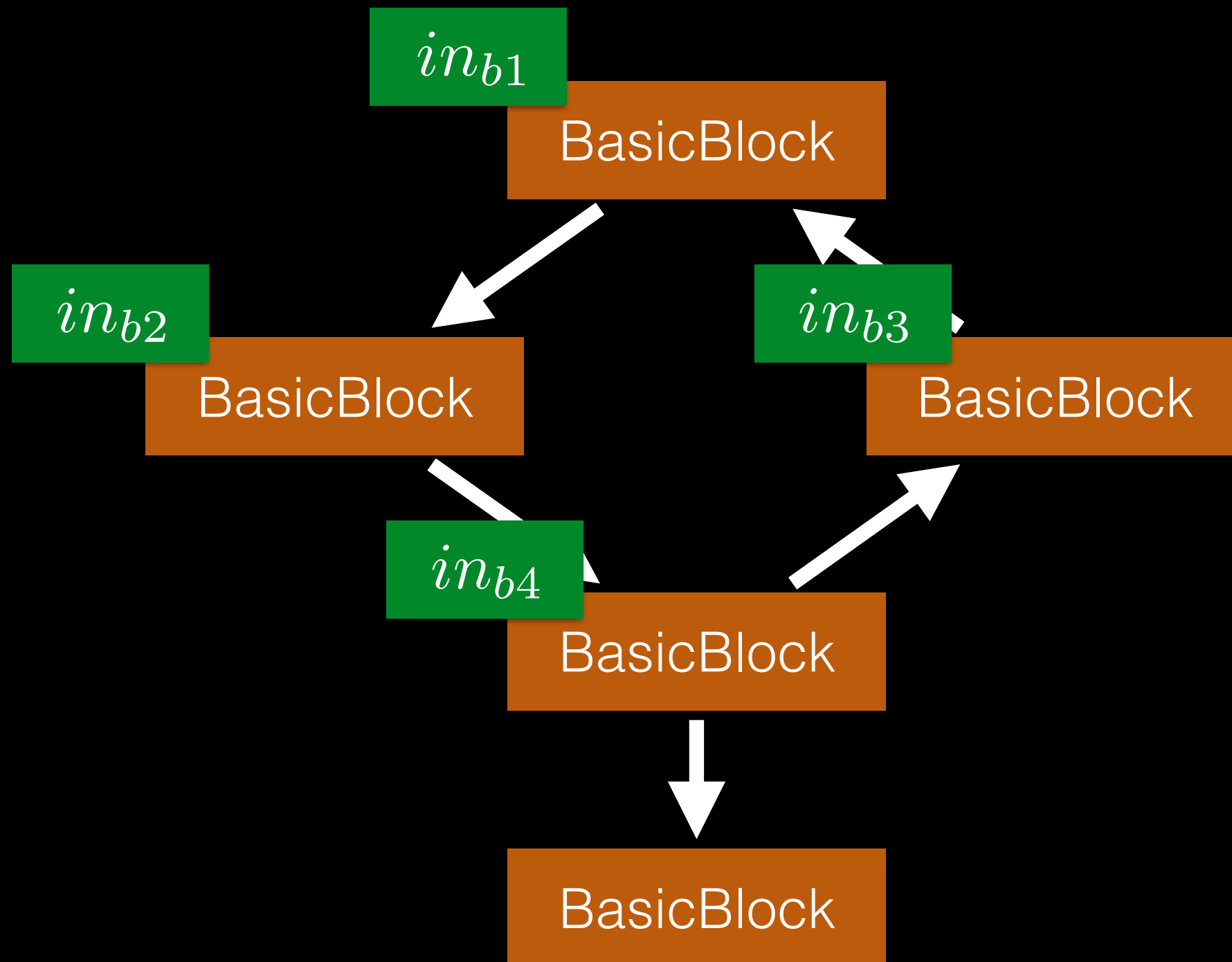
# Strategy #2



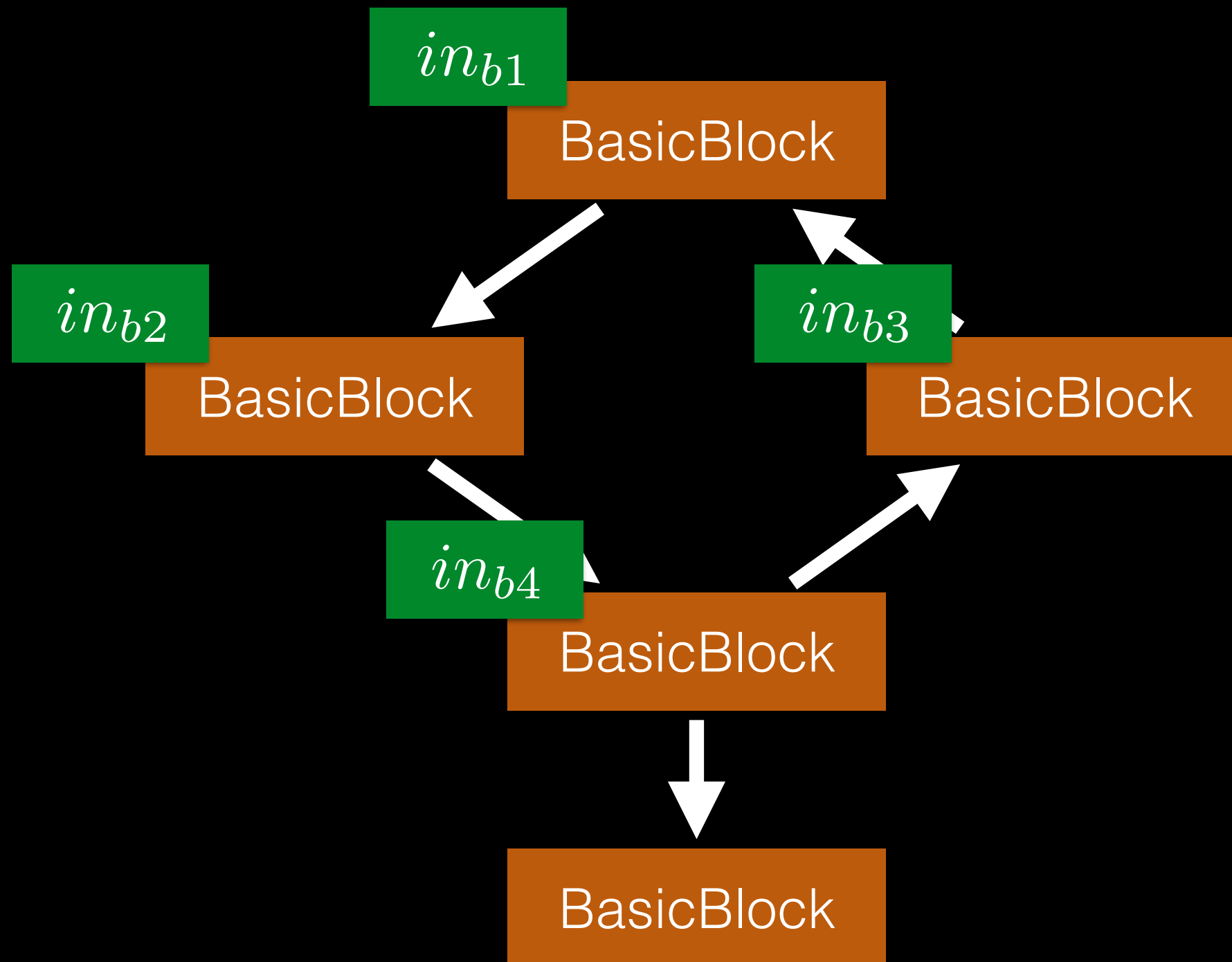
**exercise 7.2b**



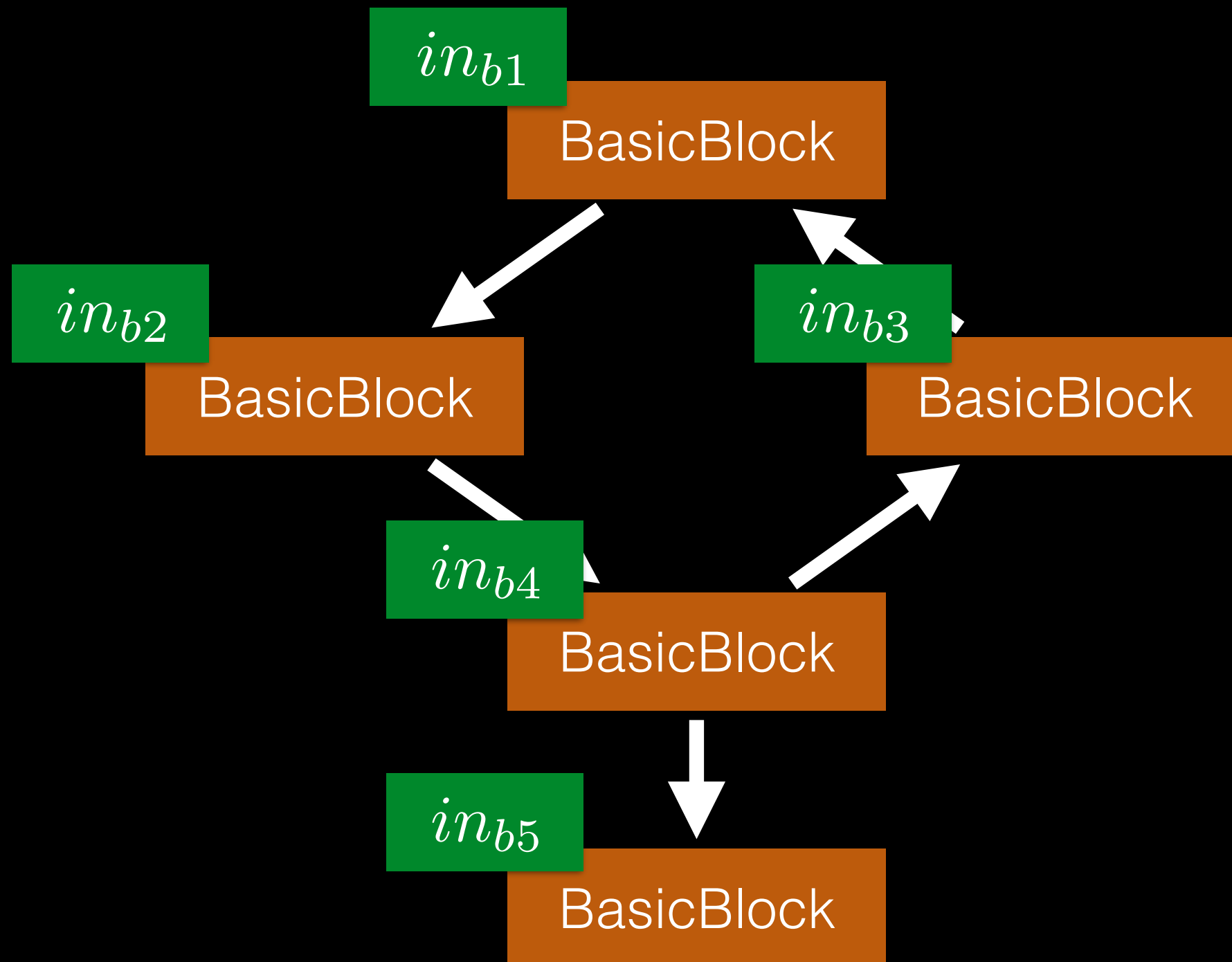
# Strategy #2



# Strategy #2



# Strategy #2



# Parameter Data-Flow Analysis

flow-sensitive version

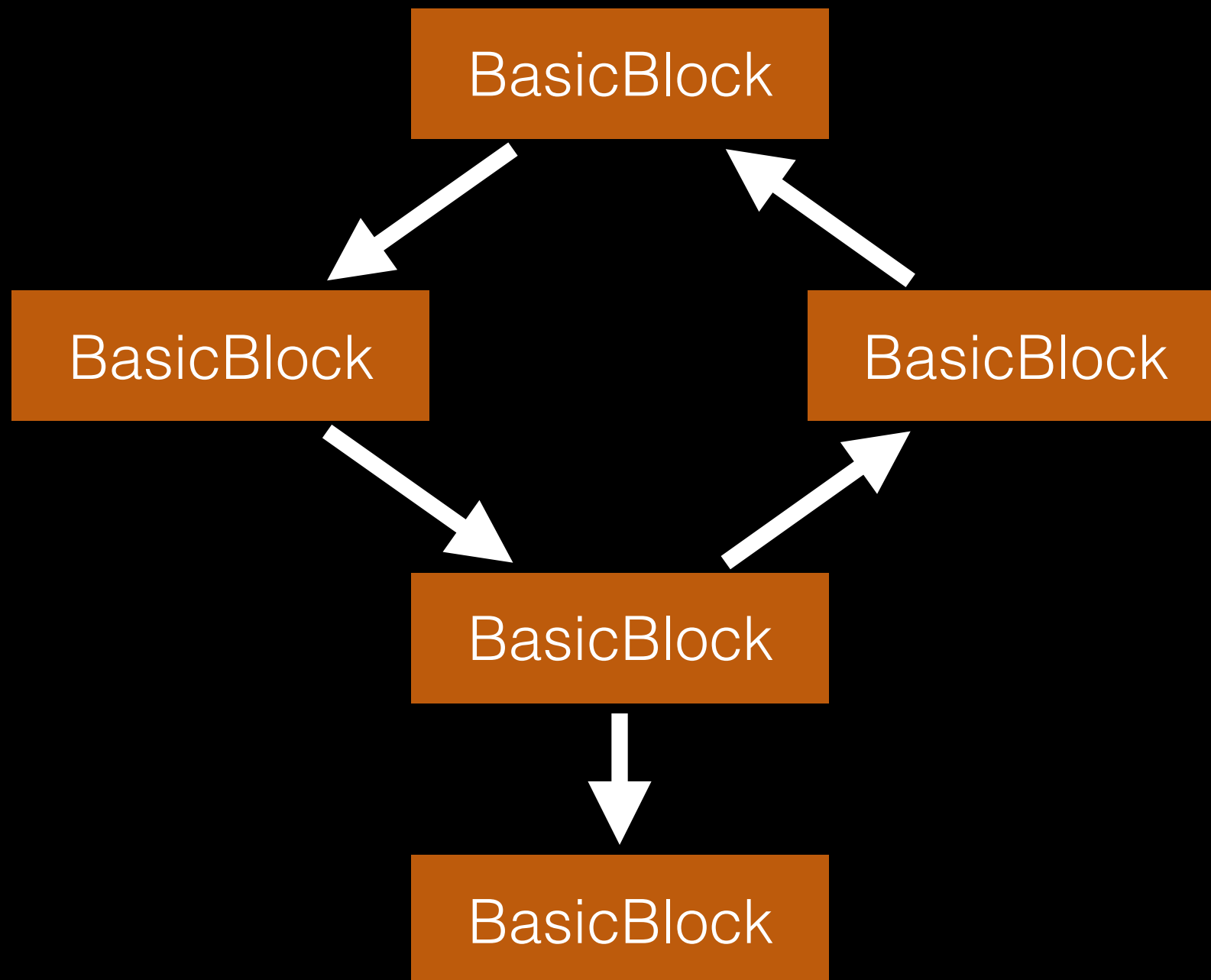
## Strategies for Handling Loops 3. Consider Output Flows

```
DenseMap<BasicBlock*, DenseSet<Value*>> previousOutputs;

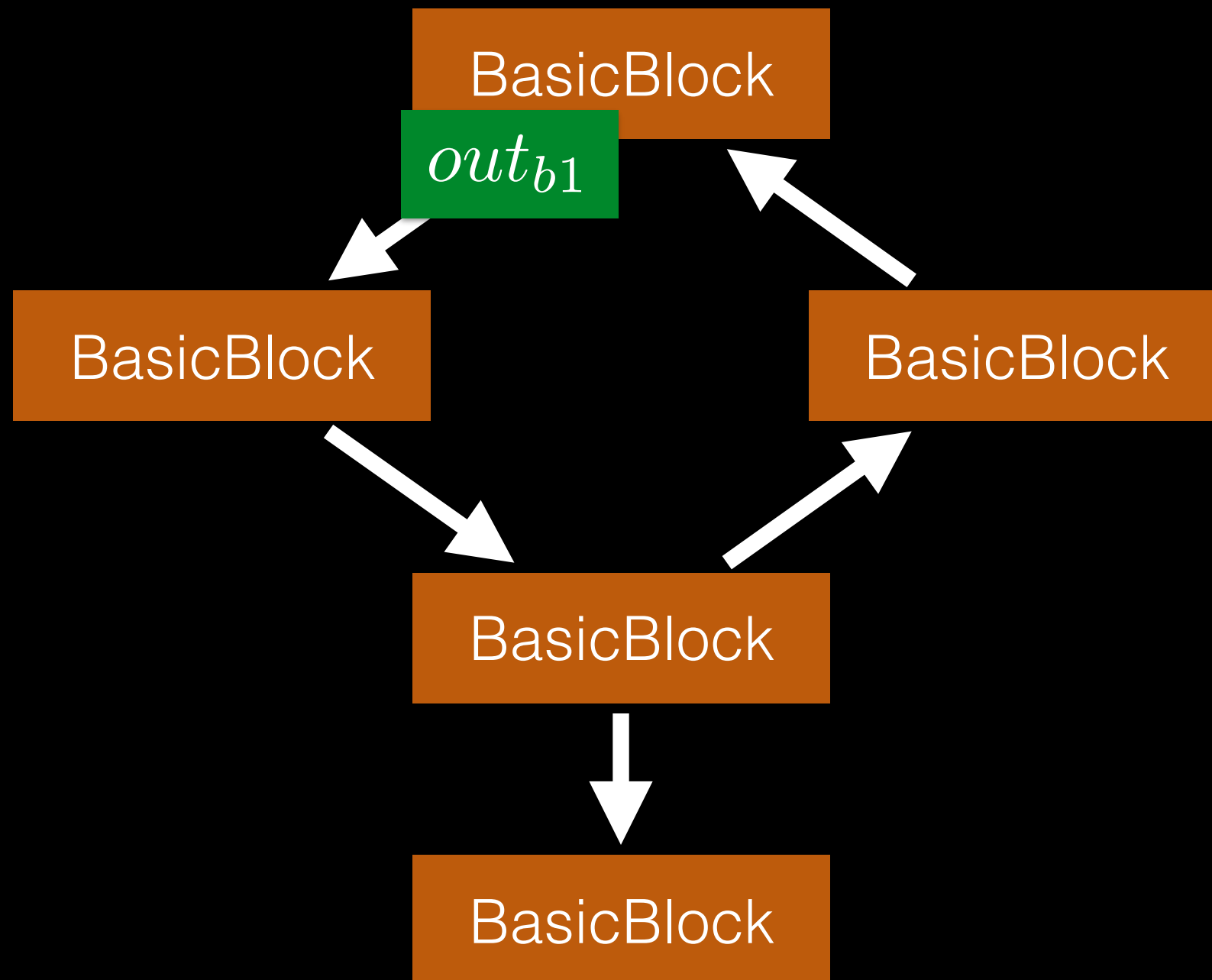
if (previousOutputs.count(current.first) > 0) {
    DenseSet<Value*> previousOutput = previousOutputs.lookup(current.first);
    if (previousOutput.size() > 0 &&
        valueSetsAreEqual(outgoingTracked, previousOutput))
        continue;
    previousOutputs.erase(current.first);
}

previousOutputs.insert(std::make_pair(current.first, outgoingTracked));
```

# Strategy #3

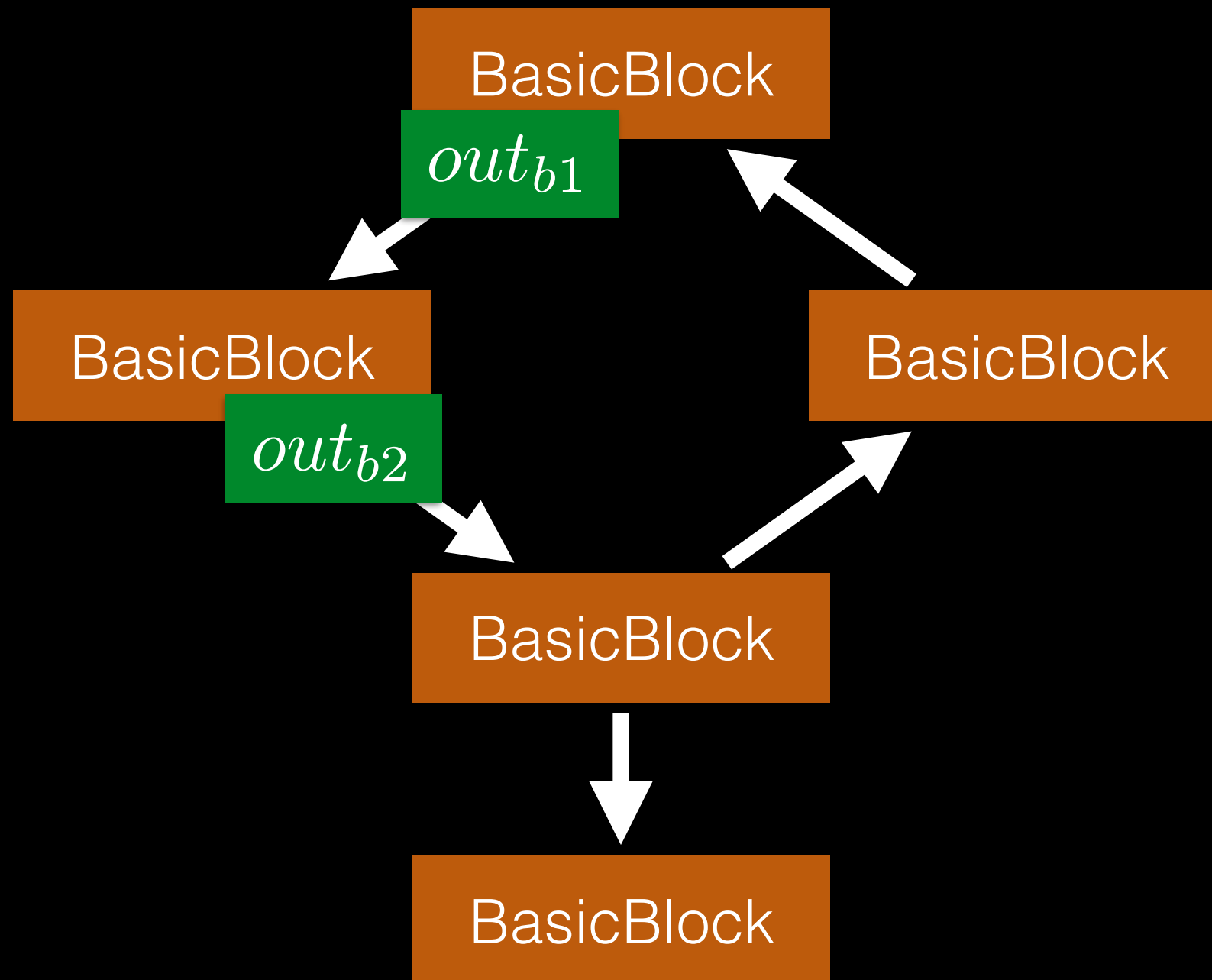


# Strategy #3

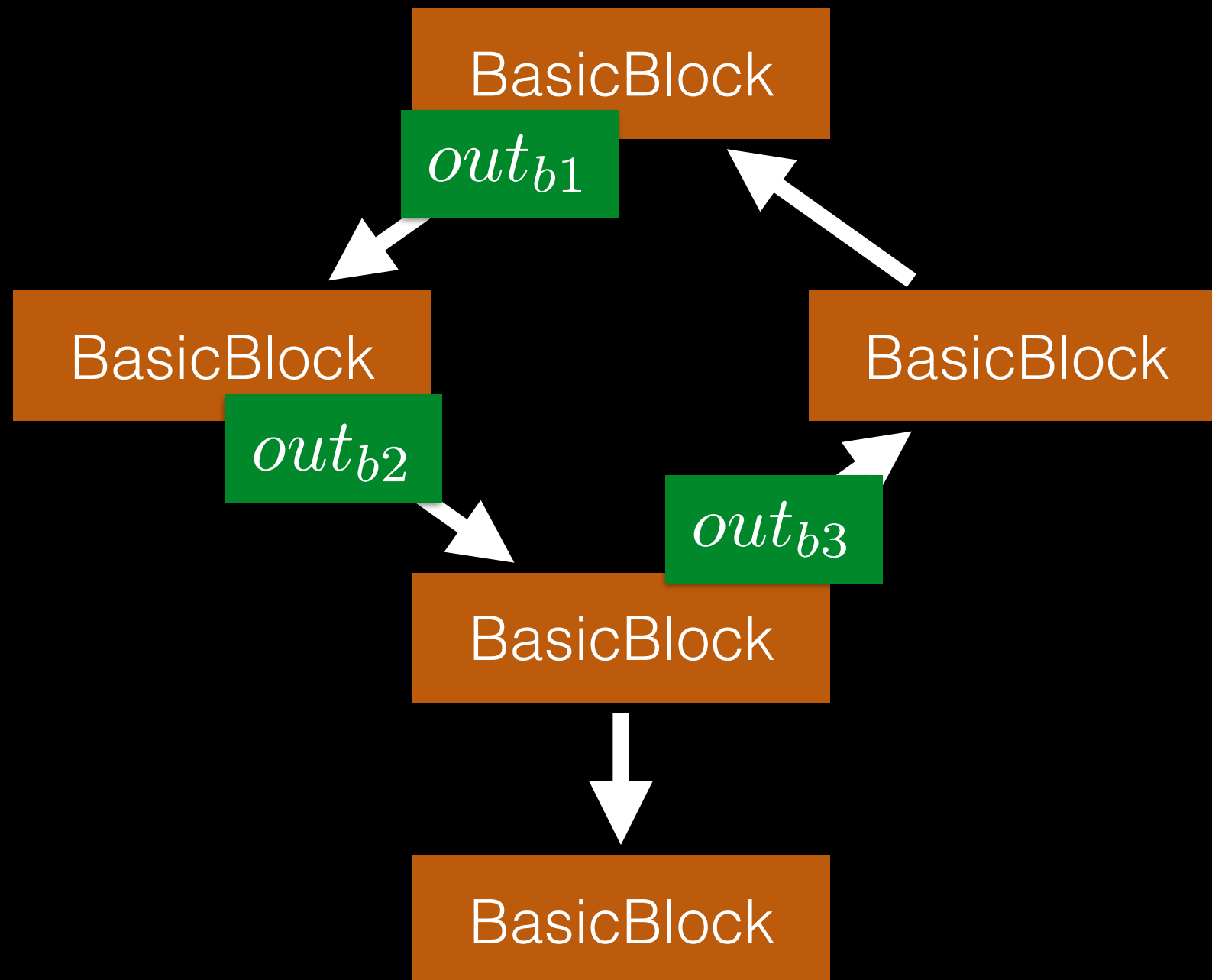


**exercise 7.2c**

# Strategy #3

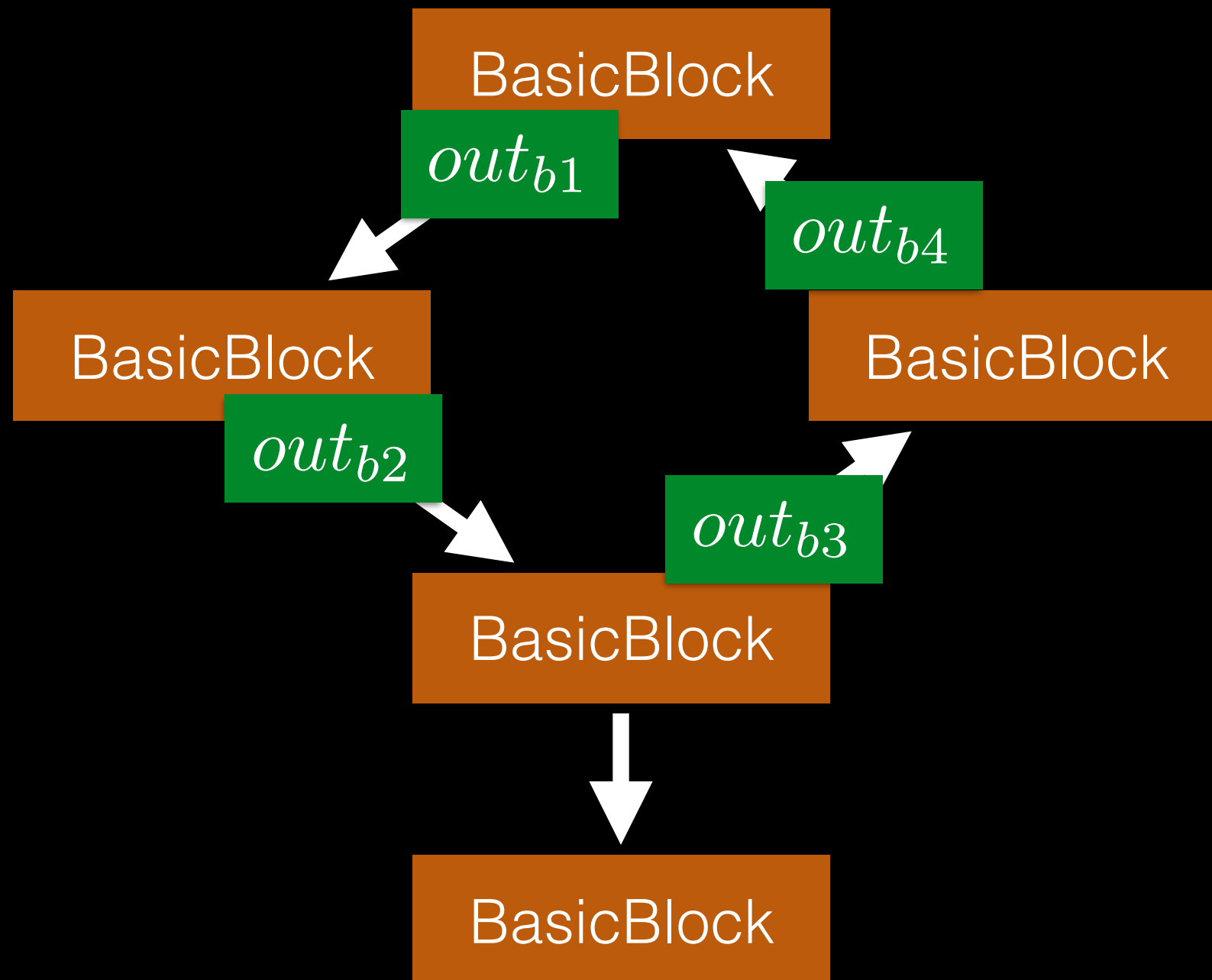


# Strategy #3

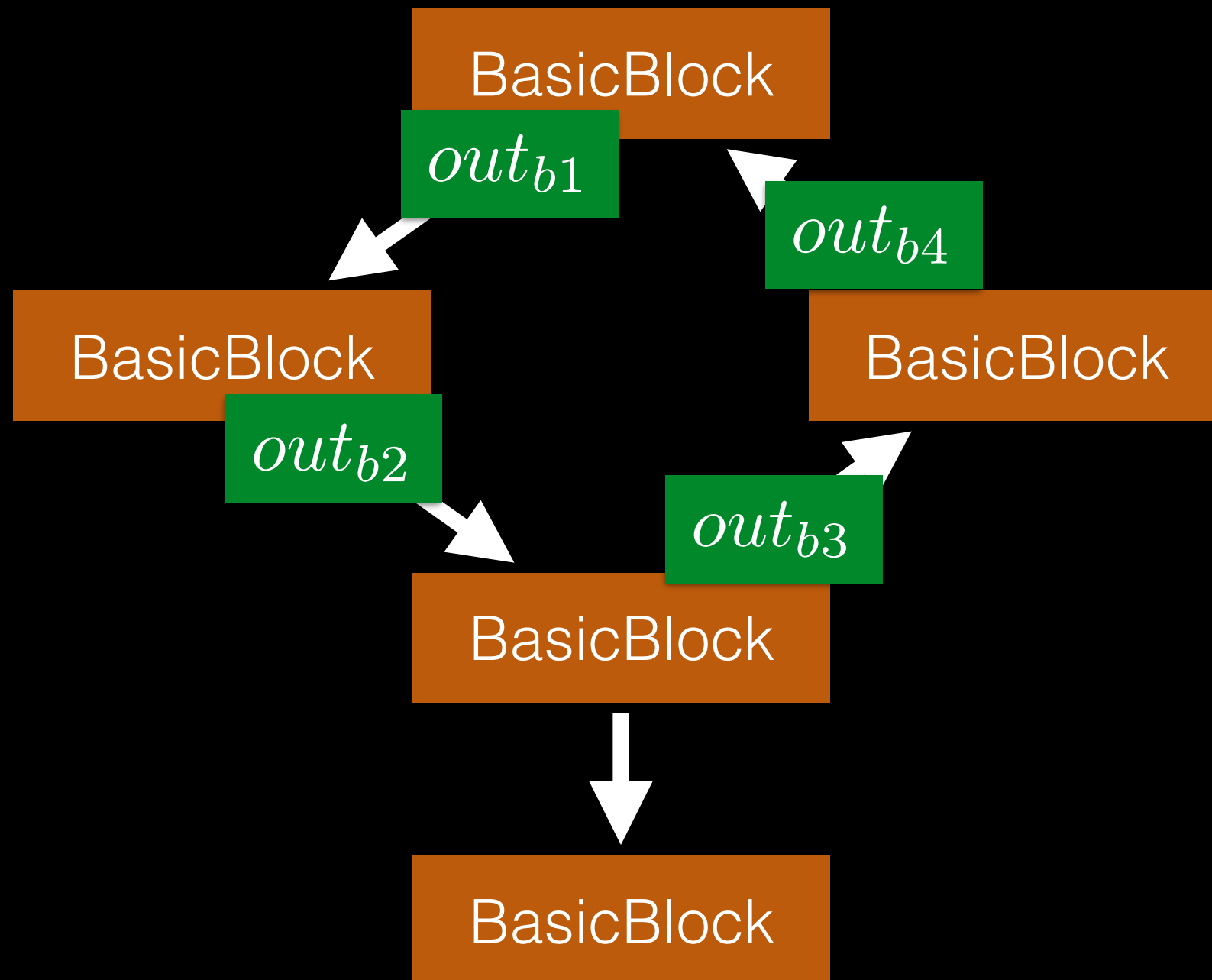




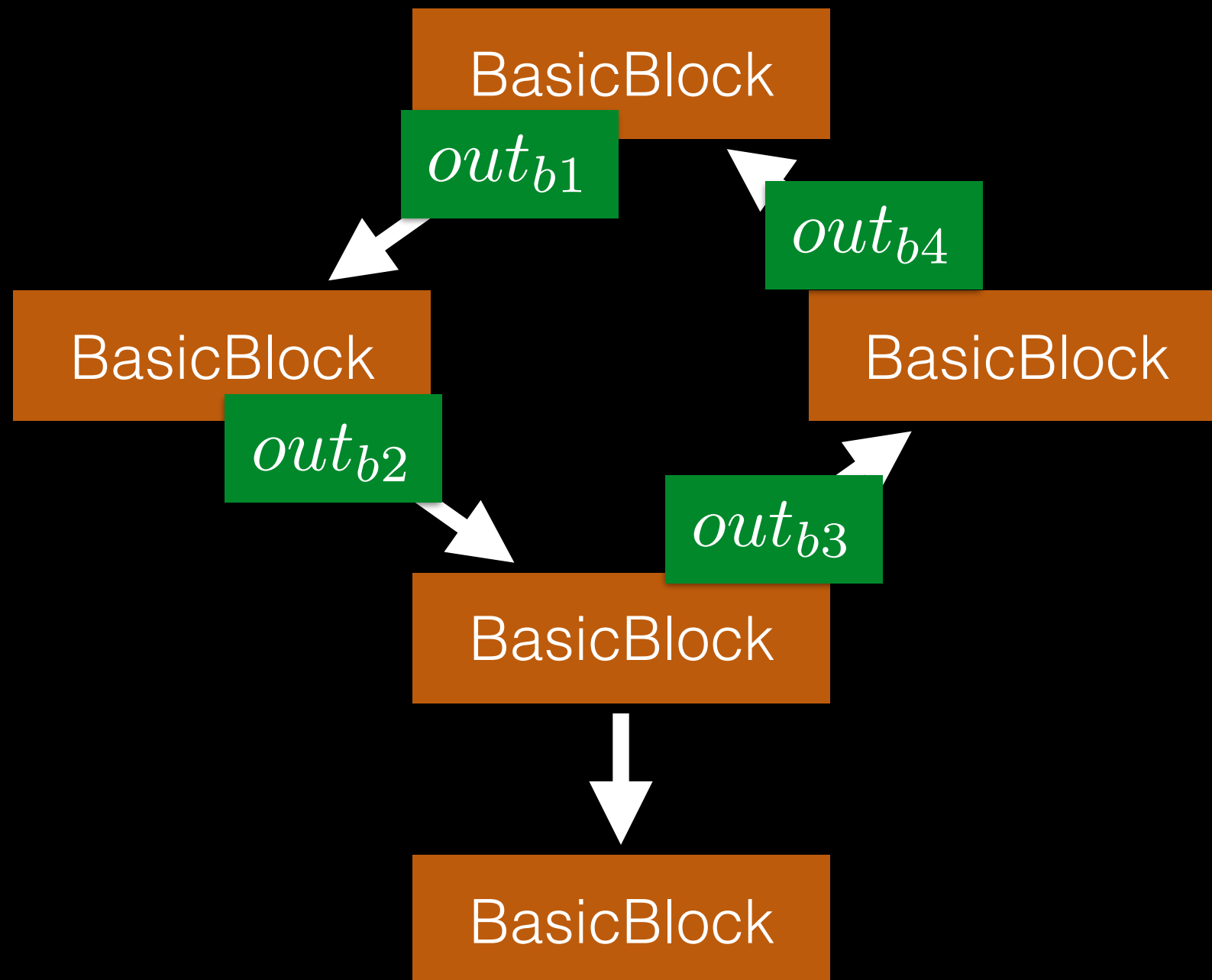
# Strategy #3



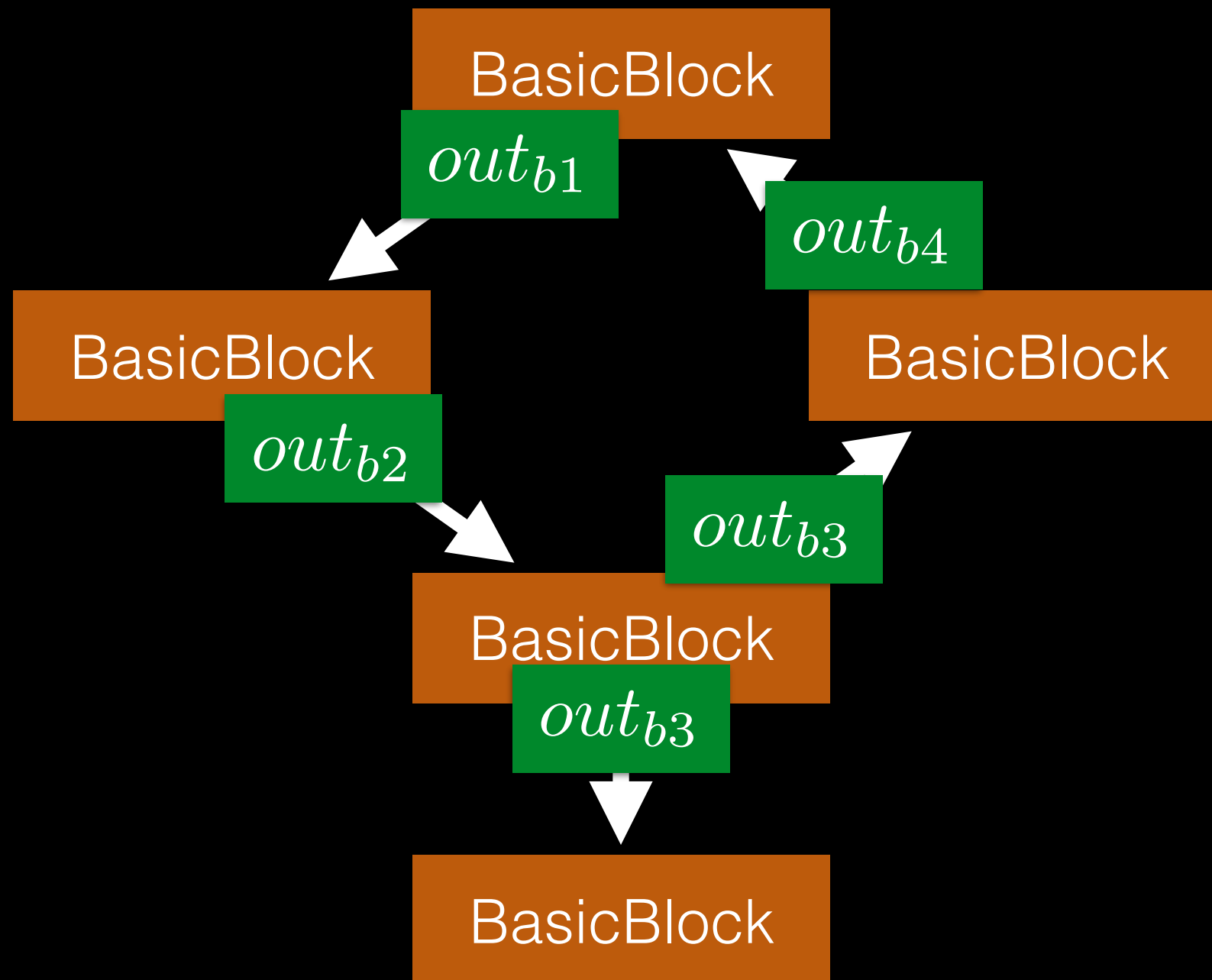
# Strategy #3



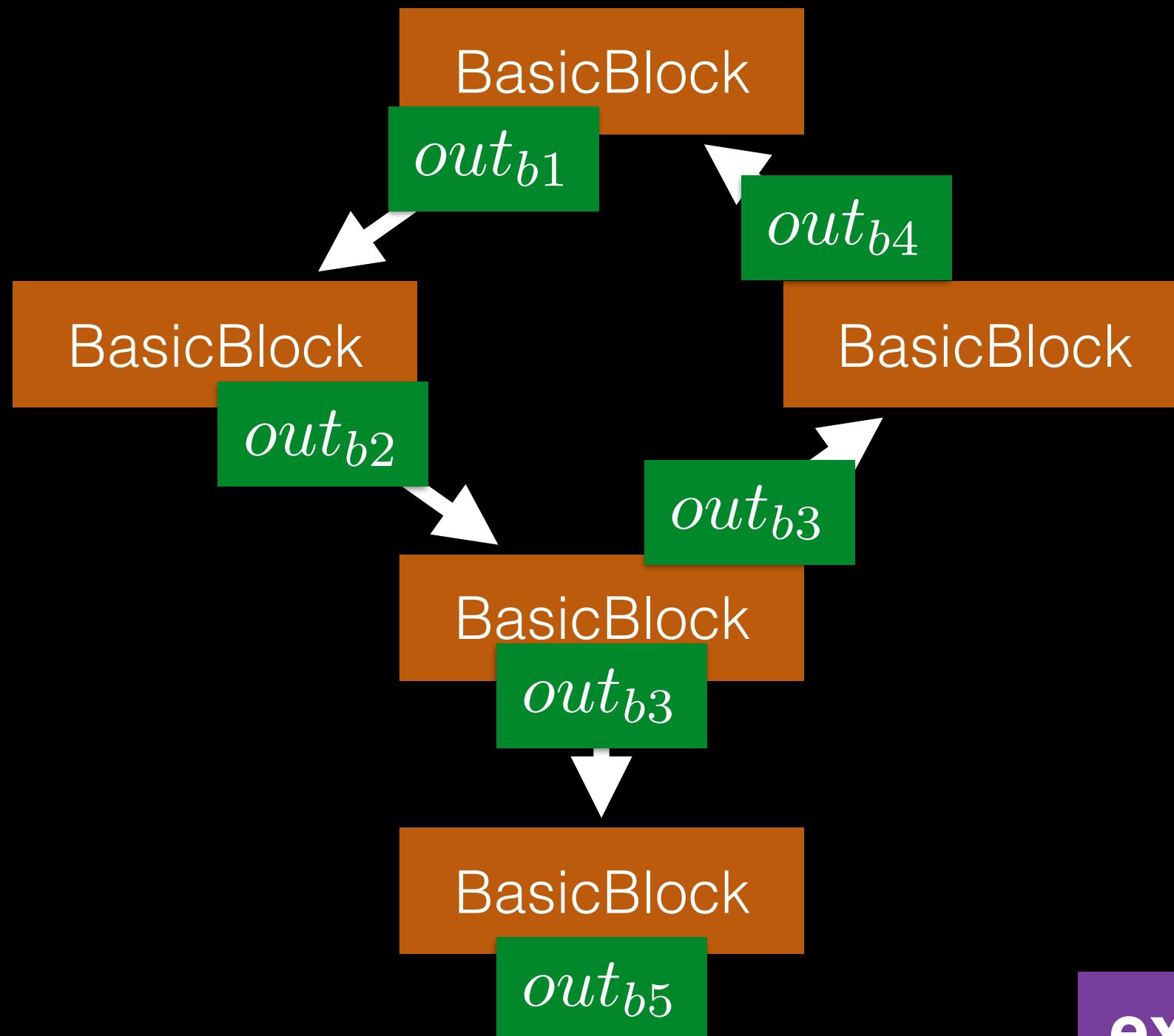
# Strategy #3



# Strategy #3



# Strategy #3



**exercise 7.2c**

# What's left to do?

- **Path-sensitivity** — Reconstruct conditions
- **Context-sensitivity** — Implement more clever call handling
- This exercise is just for your own curiosity, I won't present a solution in the next lecture

# Writing Transformers

# Why transform code?

- Finding problems in code is nice
- Solving them automatically is much, much nicer
- Can be part of your usual compile pipeline and helps you maintain quality
- Make runtime observations w/o cluttering your code



# How to Write a Transformer?

`exercises/FirstTransform/`

- A transformer is a normal pass
- But it returns `true` to signal code changes

```
for(BasicBlock &bb : F) {  
    Instruction *firstInst = bb.getFirstNonPHI()  
    IRBuilder<> builder(firstInst);  
    Instruction *newInst =  
        builder.CreateAlloca(Type::getInt32Ty(bb.getContext()));  
}
```

# How to Write a Transformer?

`exercises/FirstTransform/`

- A transformer is a normal pass
- But it returns `true` to signal code changes

Get the first instructions

```
for(BasicBlock &bb : F) {  
    Instruction *firstInst = bb.getFirstNonPHI()  
    IRBuilder<> builder(firstInst);  
    Instruction *newInst =  
        builder.CreateAlloca(Type::getInt32Ty(bb.getContext()));  
}
```

# How to Write a Transformer?

`exercises/FirstTransform/`

- A transformer is a normal pass
- But it returns `true` to signal code changes


```
for(BasicBlock &bb : F) {  
    Instruction *firstInst = bb.getFirstNonPHI()  
    IRBuilder<> builder(firstInst);  
    Instruction *newInst =  
        builder.CreateAlloca(Type::getInt32Ty(bb.getContext()));  
}
```

# How to Write a Transformer?

`exercises/FirstTransform/`

- A transformer is a normal pass
- But it returns `true` to signal code changes

```
for(BasicBlock &bb : F) {  
    Instruction *firstInst = bb.getFirstNonPHI()  
    IRBuilder<> builder(firstInst);  
    Instruction *newInst =  
        builder.CreateAlloca(Type::getInt32Ty(bb.getContext()));  
}
```



Construct a IRBuilder

# How to Write a Transformer?

`exercises/FirstTransform/`

- A transformer is a normal pass
- But it returns `true` to signal code changes

```
for(BasicBlock &bb : F) {  
    Instruction *firstInst = bb.getFirstNonPHI()  
    IRBuilder<> builder(firstInst);  
    Instruction *newInst =  
        builder.CreateAlloca(Type::getInt32Ty(bb.getContext()));  
}
```

# How to Write a Transformer?

`exercises/FirstTransform/`

- A transformer is a normal pass
- But it returns `true` to signal code changes

```
for(BasicBlock &bb : F) {  
    Instruction *firstInst = bb.getFirstNonPHI()  
    IRBuilder<> builder(firstInst);  
    Instruction *newInst =  
        builder.CreateAlloca(Type::getInt32Ty(bb.getContext()));  
}
```

Create an `alloc` instruction

**exercise 7.4**

# How to Write a Transformer?

`exercises/FirstTransform/`

- A transformer is a normal pass
- But it returns `true` to signal code changes

```
for(BasicBlock &bb : F) {  
    Instruction *firstInst = bb.getFirstNonPHI()  
    IRBuilder<> builder(firstInst);  
    Instruction *newInst =  
        builder.CreateAlloca(Type::getInt32Ty(bb.getContext()));  
}
```

# Running a Transformer

- Much like an analysis pass

```
opt-3.8 -load FirstTransform/libFirstTransform.so  
        -FirstTransform  
        < factorial.bc  
        > factorial-transformed.bc
```

```
opt-3.8 -load FirstTransform/libFirstTransform.so  
        -FirstTransform  
        --debug-pass=Structure  
        < factorial.bc  
        > /dev/null
```



# Running a Transformer

- Much like an analysis pass

```
opt-3.8 -load FirstTransform/libFirstTransform.so  
        -FirstTransform  
        < factorial.bc  
        > factorial-transformed.bc
```

```
opt-3.8 -load FirstTransform/libFirstTransform.so  
        -FirstTransform  
        --debug-pass=Structure  
        < factorial.bc  
        > /dev/null
```

Show me the structure of the pipeline

# Running a Transformer

- Much like an analysis pass

```
opt-3.8 -load FirstTransform/libFirstTransform.so  
        -FirstTransform  
        < factorial.bc  
        > factorial-transformed.bc
```

```
opt-3.8 -load FirstTransform/libFirstTransform.so  
        -FirstTransform  
        --debug-pass=Structure  
        < factorial.bc  
        > /dev/null
```

# Pass Structure

Pass Arguments: -targetlibinfo -tti -FirstTransform -verify

Target Library Information

Target Transform Information

ModulePass Manager

FunctionPass Manager

First transform

Module Verifier

Bitcode Writer

# Pass Structure

Pass Arguments: -targetlibinfo -tti -FirstTransform -verify

Target Library Information

Target Transform Information

ModulePass Manager

FunctionPass Manager

First transform

Module Verifier

Bitcode Writer



Pass managers try to optimize the execution of passes

# Pass Structure

Pass Arguments: -targetlibinfo -tti -FirstTransform -verify

Target Library Information

Target Transform Information

ModulePass Manager

FunctionPass Manager


First transform

Module Verifier

Bitcode Writer

# Pass Structure

Pass Arguments: -targetlibinfo -tti -FirstTransform -verify  
Target Library Information  
Target Transform Information  
ModulePass Manager  
FunctionPass Manager  
First transform  
Module Verifier  
Bitcode Writer



Verifies the resulting bitcode

# Pass Structure

Pass Arguments: -targetlibinfo -tti -FirstTransform -verify

Target Library Information

Target Transform Information

ModulePass Manager

FunctionPass Manager

First transform

Module Verifier

Bitcode Writer

# Pass Structure

Pass Arguments: -targetlibinfo -tti -FirstTransform -verify

Target Library Information

Target Transform Information

ModulePass Manager

FunctionPass Manager

First transform

Module Verifier

Bitcode Writer



Emits bitcode



# Pass Structure

Pass Arguments: -targetlibinfo -tti -FirstTransform -verify

Target Library Information

Target Transform Information

ModulePass Manager

FunctionPass Manager

First transform

Module Verifier

Bitcode Writer

# PassManager

- Its job:
  - Pipeline the execution of passes on the program
  - Share the results of analyses
- There is no need for you to call it directly
- However, you can help the PassManager

# PassManager

- Help the PassManager do its job

```
virtual void getAnalysisUsage(AnalysisUsage &Info) const;
```

# PassManager

- Help the PassManager do its job

```
virtual void getAnalysisUsage(AnalysisUsage &Info) const;
```

```
// This example modifies the program, but does not modify the CFG
void LICM::getAnalysisUsage(AnalysisUsage &AU) const {
    AU.setPreservesCFG();
    AU.addRequired<LoopInfoWrapperPass>();
}
```

# PassManager

- Help the PassManager do its job

```
virtual void getAnalysisUsage(AnalysisUsage &Info) const;
```

```
// This example modifies the program, but does not modify the CFG  
void LICM::getAnalysisUsage(AnalysisUsage &AU) const {  
    AU.setPreservesCFG();  
    AU.addRequired<LoopInfoWrapperPass>();  
}
```

Provide information on your pass

# PassManager

- Help the PassManager do its job

```
virtual void getAnalysisUsage(AnalysisUsage &Info) const;
```

```
// This example modifies the program, but does not modify the CFG
void LICM::getAnalysisUsage(AnalysisUsage &AU) const {
    AU.setPreservesCFG();
    AU.addRequired<LoopInfoWrapperPass>();
}
```

# PassManager

- Help the PassManager do its job

```
virtual void getAnalysisUsage(AnalysisUsage &Info) const;
```

```
// This example modifies the program, but does not modify the CFG  
void LICM::getAnalysisUsage(AnalysisUsage &AU) const {  
    AU.setPreservesCFG();  
    AU.addRequired<LoopInfoWrapperPass>();  
}
```



Require other passes

# PassManager

- Help the PassManager do its job

```
virtual void getAnalysisUsage(AnalysisUsage &Info) const;
```

```
// This example modifies the program, but does not modify the CFG
void LICM::getAnalysisUsage(AnalysisUsage &AU) const {
    AU.setPreservesCFG();
    AU.addRequired<LoopInfoWrapperPass>();
}
```



# PassManager

- Get modularity in return

```
bool runOnFunction(Function &F) override {  
    LoopInfo &LI = getAnalysis<LoopInfoWrapperPass>().getLoopInfo();  
    //...  
}
```

# PassManager

- Get modularity in return

```
bool runOnFunction(Function &F) override {  
    LoopInfo &LI = getAnalysis<LoopInfoWrapperPass>().getLoopInfo();  
    //...  
}
```

Get a reference to the required  
other pass

# PassManager

- Get modularity in return

```
bool runOnFunction(Function &F) override {  
    LoopInfo &LI = getAnalysis<LoopInfoWrapperPass>().getLoopInfo();  
    //...  
}
```

# PassManager

- Get modularity in return

```
bool runOnFunction(Function &F) override {  
    LoopInfo &LI = getAnalysis<LoopInfoWrapperPass>().getLoopInfo();  
    //...  
}
```

Retrieve information from other  
pass

# PassManager

- Get modularity in return

```
bool runOnFunction(Function &F) override {  
    LoopInfo &LI = getAnalysis<LoopInfoWrapperPass>().getLoopInfo();  
    //...  
}
```

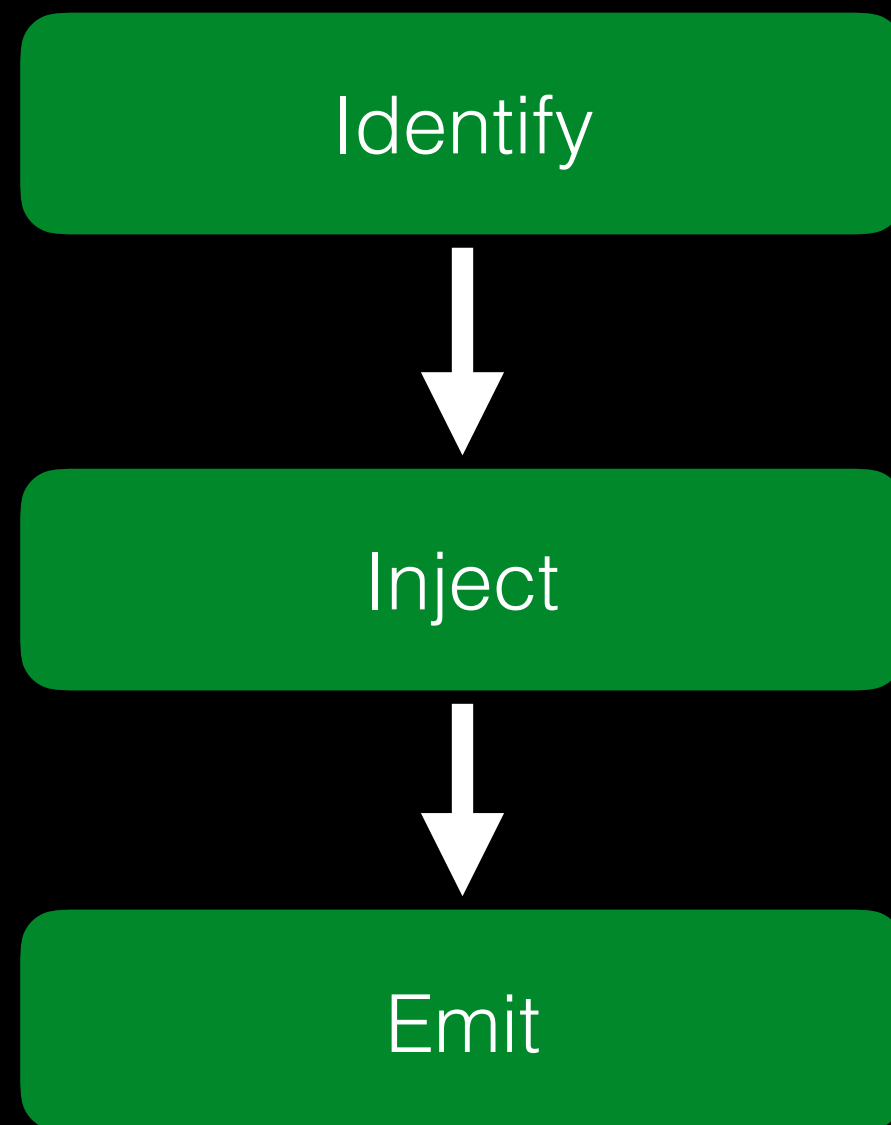
# PassManager

- Get modularity in return

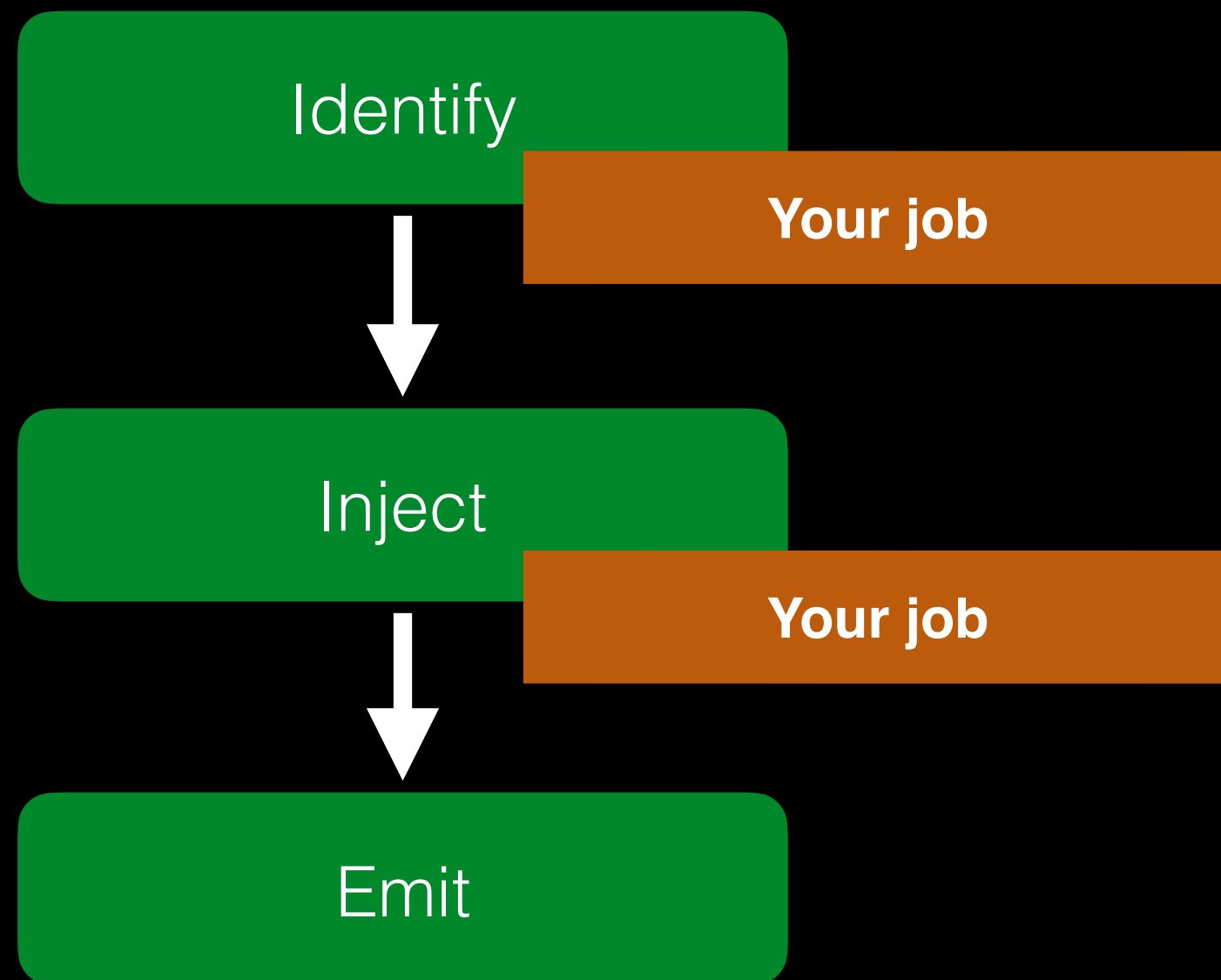
```
bool runOnFunction(Function &F) override {  
    LoopInfo &LI = getAnalysis<LoopInfoWrapperPass>().getLoopInfo();  
    //...  
}
```

```
if (DominatorSet *DS = getAnalysisIfAvailable<DominatorSet>()) {  
    // A DominatorSet is active. This code will update it.  
}
```

# General Structure of a Transformer

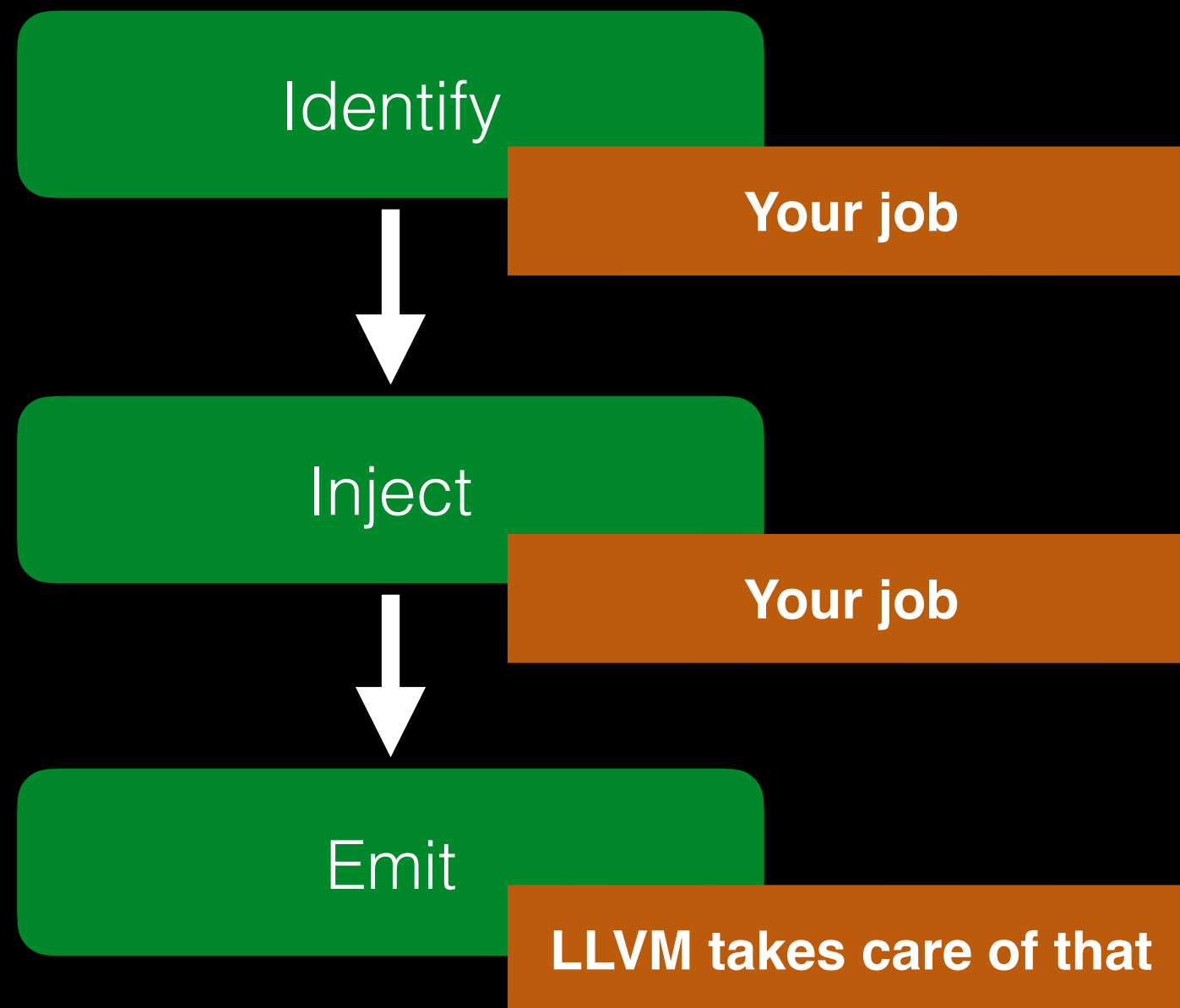


# General Structure of a Transformer



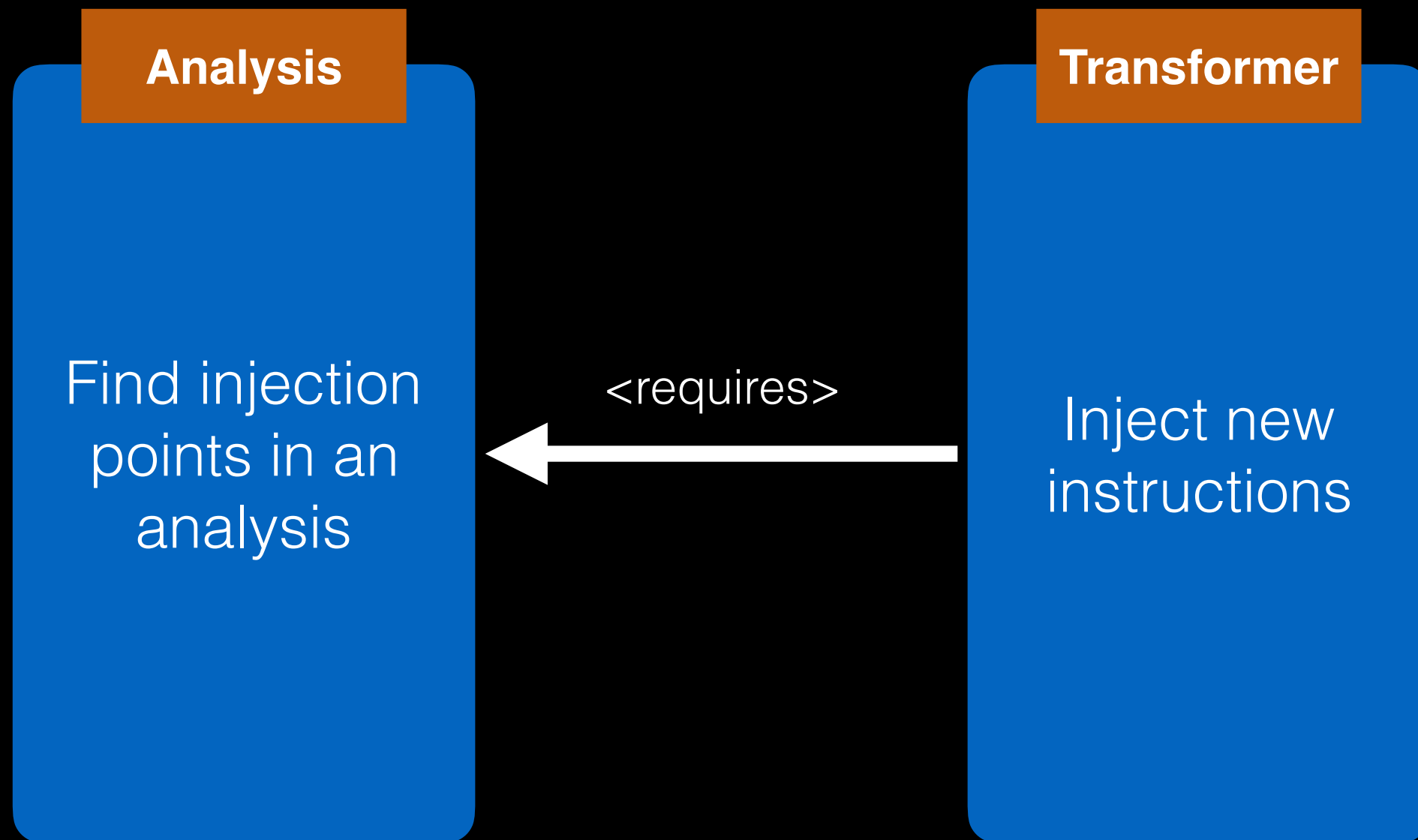


# General Structure of a Transformer



# Chain your Passes

*just a matter of style*



# Exercise: Observe pointer instructions during runtime

- Pointer operations can lead to serious issues during runtime
- Moving a pointer to a different region of memory can lead unexpected results
- Reasoning about pointers statically is pretty hard
- Goal: Observe pointer instructions during runtime

# Identification Phase

```
DenseSet<Instruction*> pointerInstructions;
for(BasicBlock &bb : f) {
    for(Instruction &i : bb) {
        if (GetElementPtrInst* gep = dyn_cast<GetElementPtrInst>(&i)) {
            pointerInstructions.insert(gep);
        }
    }
}
```

# Injection Phase

```
for (Instruction *i : pointerInstructions) {  
    errs() << "Injecting print operation\n";  
    IRBuilder<> builder(i);  
    Function *printfFunction = getPrintfPrototype(i->getContext(),  
                                                    i->getModule());  
  
    builder.CreateCall(printfFunction,  
                       geti8StrVal(*i->getModule(),  
                                   "Pointer instruction\n",  
                                   "name"));  
}
```

# Function Prototypes

```
Function *getPrintfPrototype(LLVMContext &ctx, Module *mod) {  
    FunctionType *printf_type =  
        TypeBuilder<int(char*, ...), false>::get(getGlobalContext());  
    Function *func = cast<Function>(  
        mod->getOrInsertFunction("printf", printf_type,  
        AttributeSet().addAttribute(mod->getContext(), 1U,  
        Attribute::NoAlias)));  
    return func;  
}
```

# String Constant

```
Constant* geti8StrVal(Module& M, char const* str, Twine const& name) {
    LLVMContext& ctx = getGlobalContext();
    Constant* strConstant = ConstantDataArray::getString(ctx, str);
    GlobalVariable* GVStr =
        new GlobalVariable(M, strConstant-&gtgetType(), true,
                           GlobalValue::InternalLinkage, strConstant,
                           name);
    Constant* zero = Constant::getNullValue(IntegerType::getInt32Ty(ctx));
    Constant* indices[] = {zero, zero};
    Constant* strVal = ConstantExpr::getGetElementPtr(Type::getInt8PtrTy(ctx),
                                                         GVStr, indices, true);
    return strVal;
}
```

# String Constant

```
Constant* geti8StrVal(Module& M, char const* str, Twine const& name) {  
    LLVMContext& ctx = getGlobalContext();  
    Constant* strConstant = ConstantDataArray::getString(ctx, str);  
    GlobalVariable* GVStr =  
        new GlobalVariable(M, strConstant-&gtgetType(), true,  
                           GlobalValue::InternalLinkage, strConstant,  
                           name);  
    Constant* zero = Constant::getNullValue(IntegerType::getInt32Ty(ctx));  
    Constant* indices[] = {zero, zero};  
    Constant* strVal = ConstantExpr::getGetElementPtr(Type::getInt8PtrTy(ctx),  
                                                         GVStr, indices, true);  
    return strVal;  
}
```

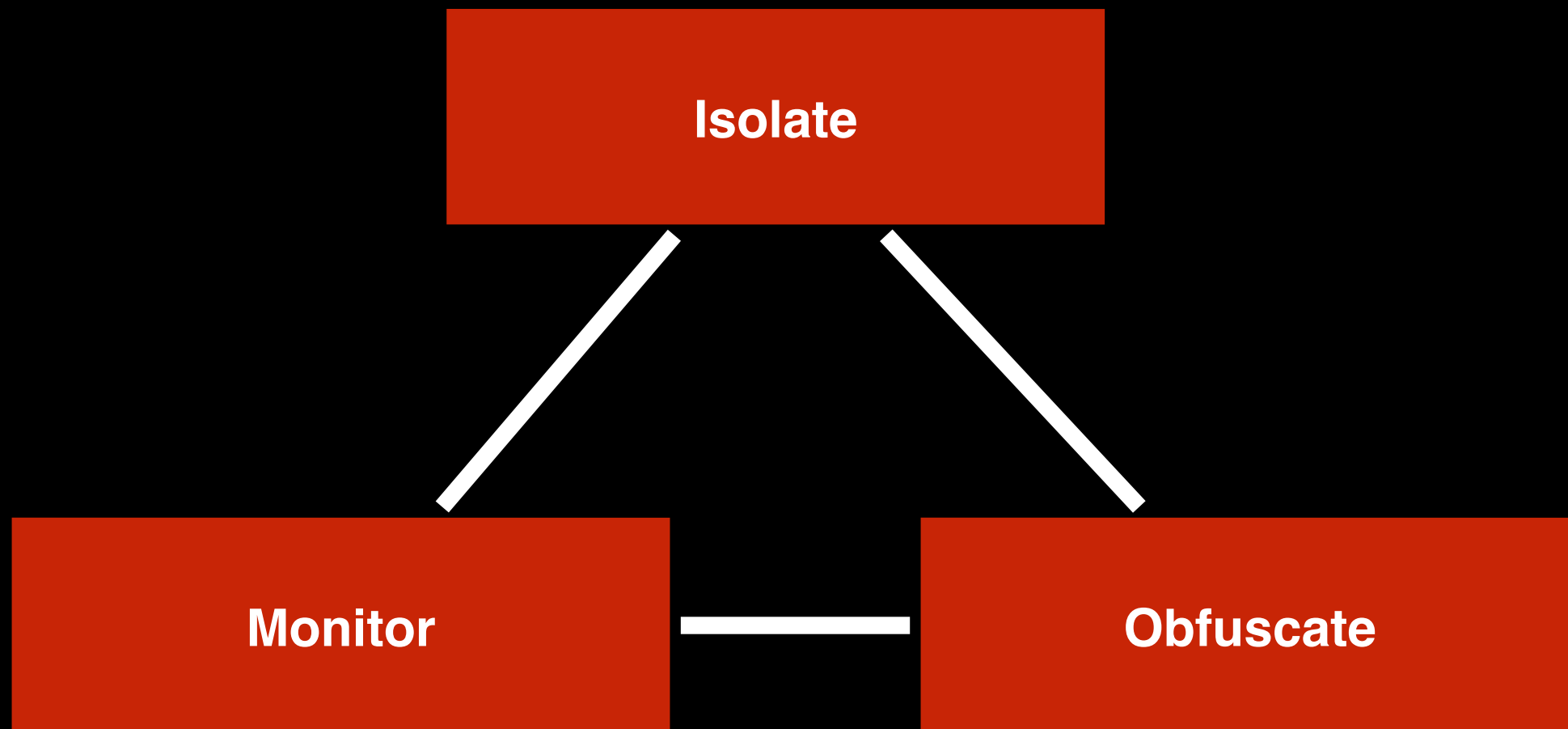
There is an error here... Can you provide the correct type?



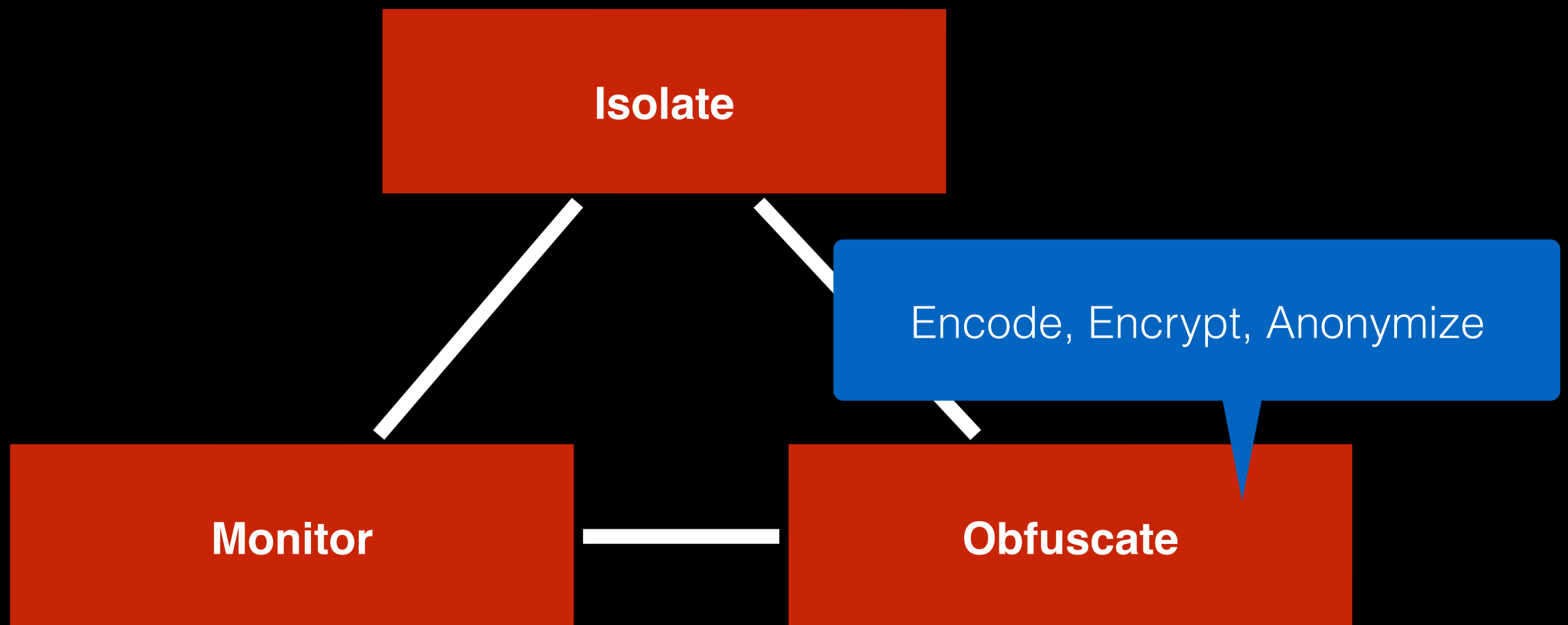
# String Constant

```
Constant* geti8StrVal(Module& M, char const* str, Twine const& name) {
    LLVMContext& ctx = getGlobalContext();
    Constant* strConstant = ConstantDataArray::getString(ctx, str);
    GlobalVariable* GVStr =
        new GlobalVariable(M, strConstant-&gtgetType(), true,
                           GlobalValue::InternalLinkage, strConstant,
                           name);
    Constant* zero = Constant::getNullValue(IntegerType::getInt32Ty(ctx));
    Constant* indices[] = {zero, zero};
    Constant* strVal = ConstantExpr::getGetElementPtr(Type::getInt8PtrTy(ctx),
                                                         GVStr, indices, true);
    return strVal;
}
```

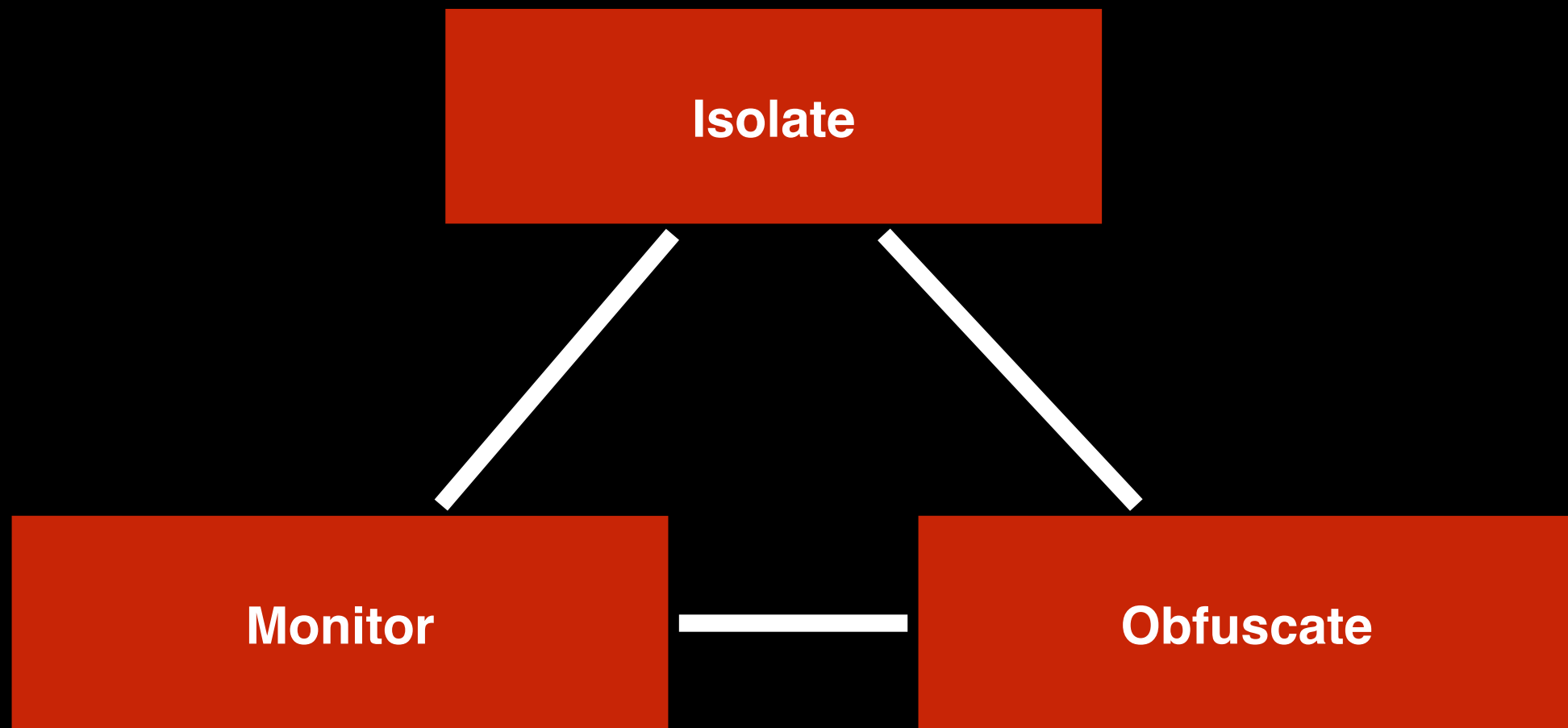
# Dimensions of Security



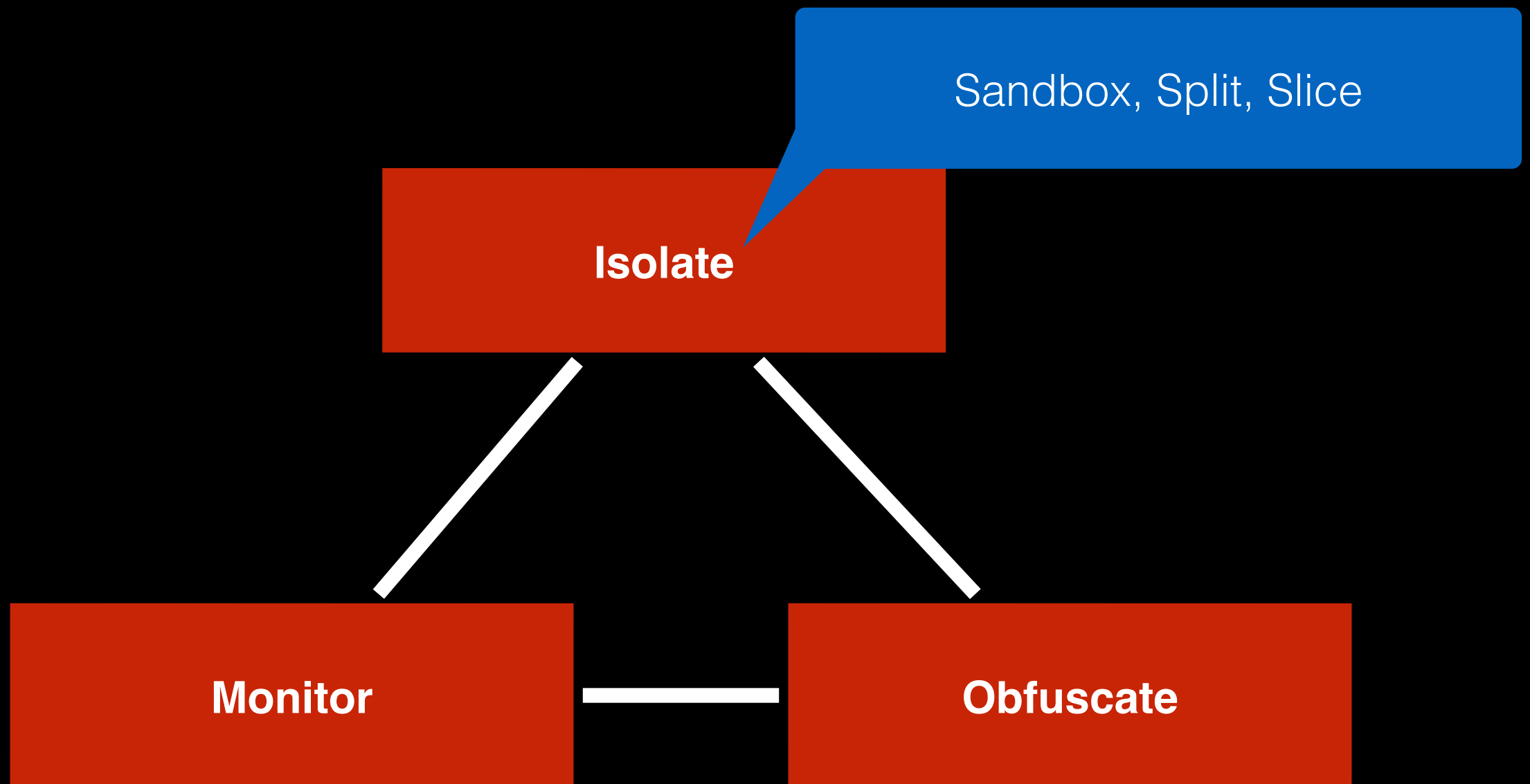
# Dimensions of Security



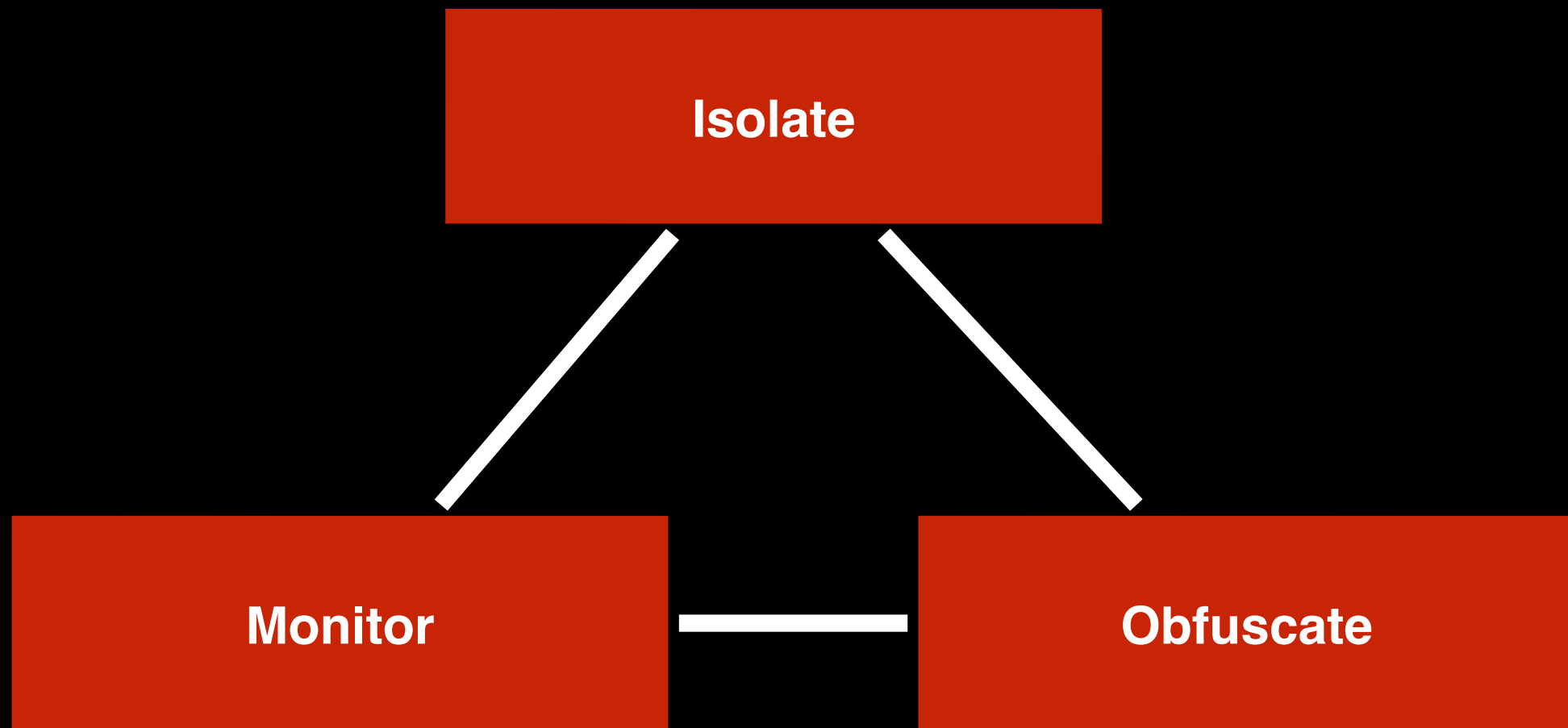
# Dimensions of Security



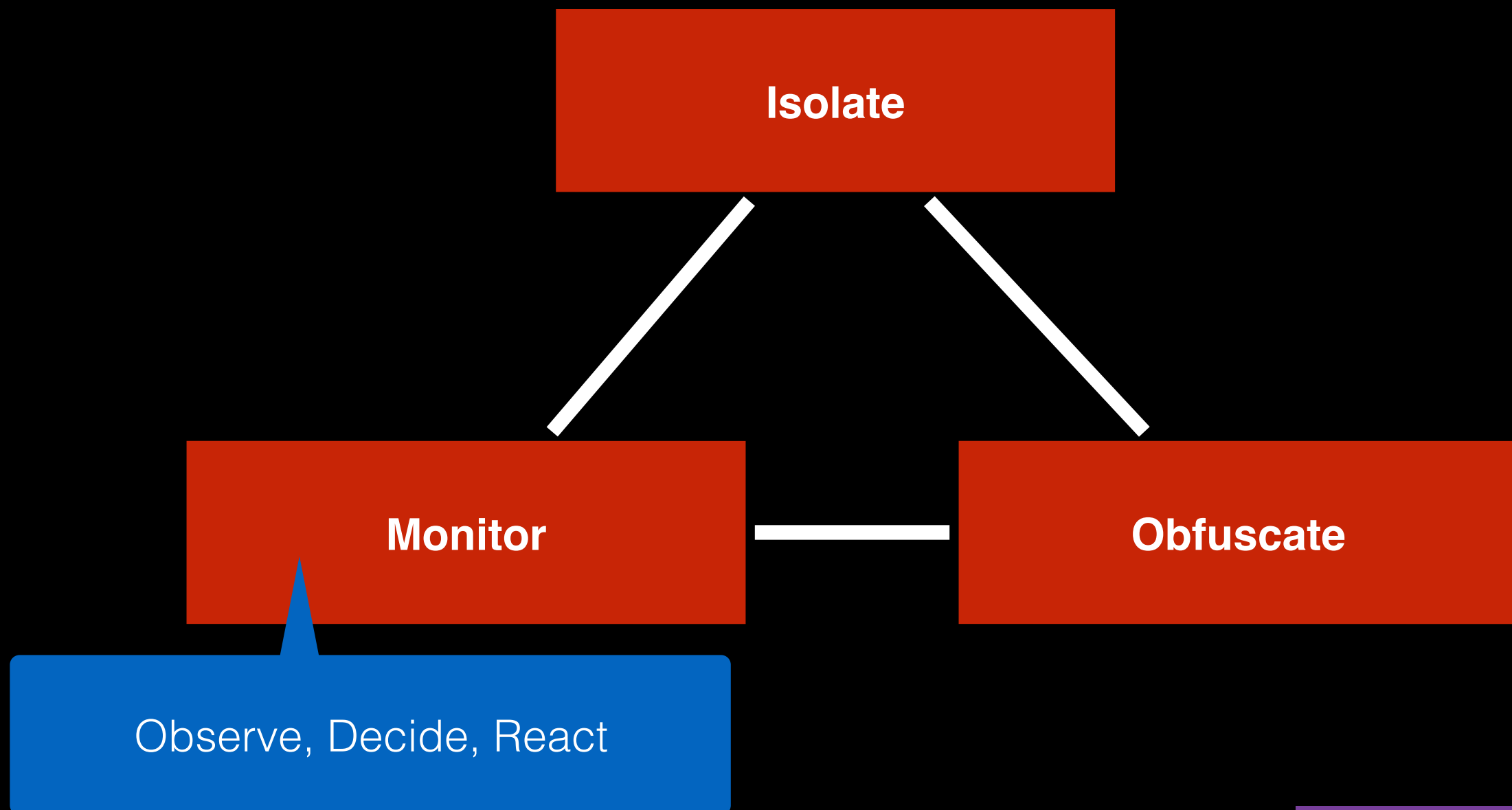
# Dimensions of Security



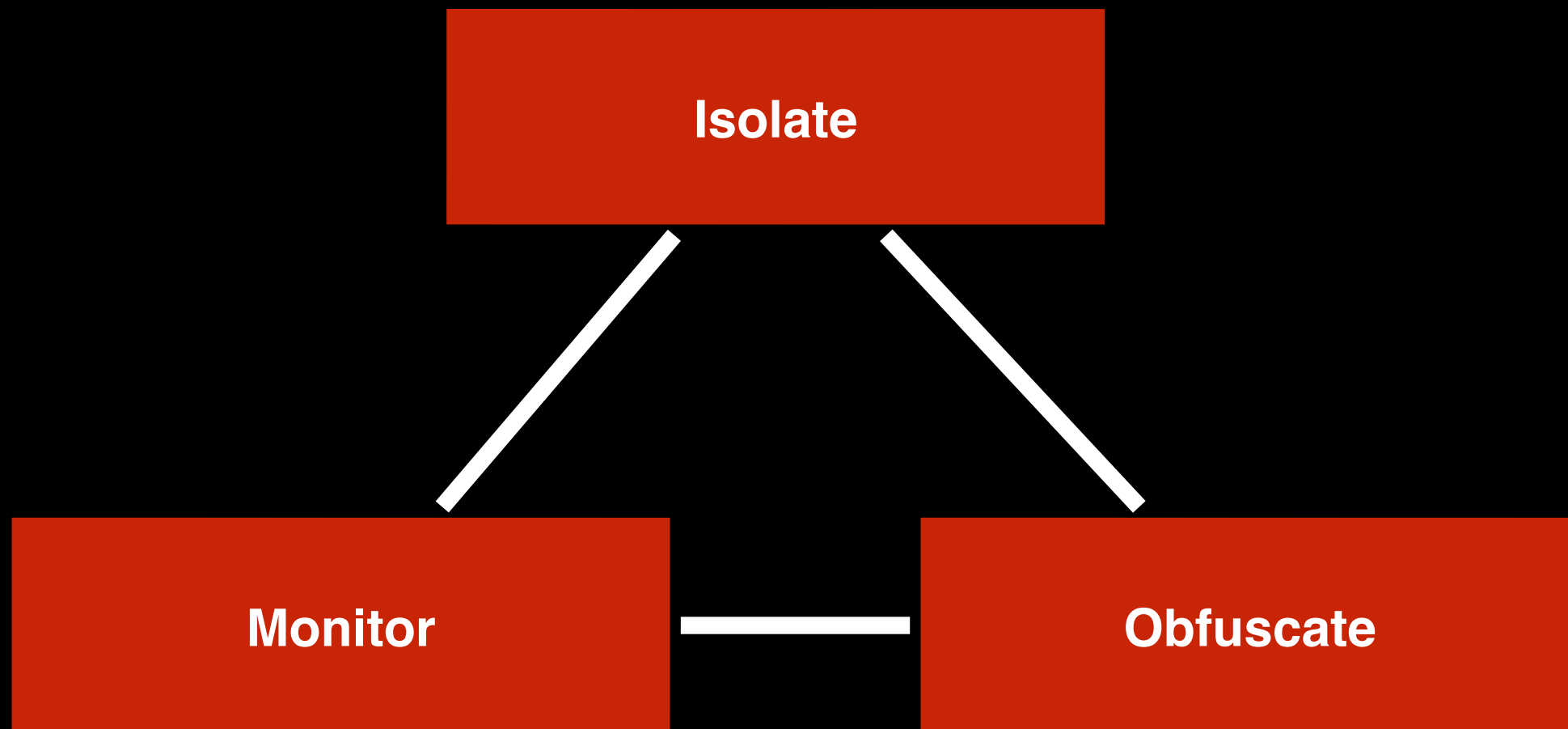
# Dimensions of Security



# Dimensions of Security



# Dimensions of Security

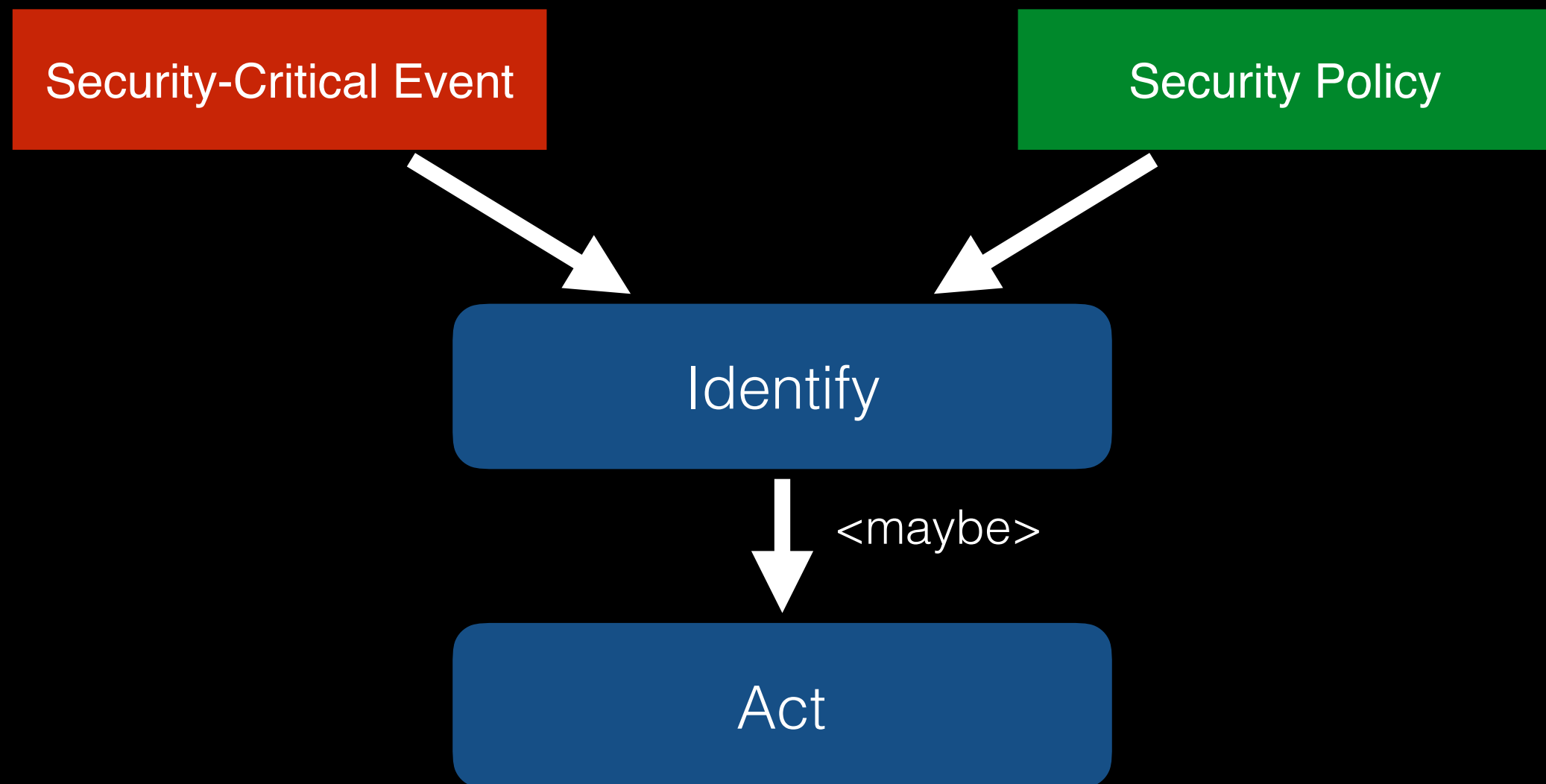




# Exercise: Inline Reference Monitor

- A reference monitor observes the execution of a program
- It halt or pauses the execution if something “bad” happens
- What “bad” ist defines a security policy
- It makes sense to observe only security critical events

# Exercise: Inline Reference Monitor



# Exercise: Inline Reference Monitor

- Goal: Write a transformer that injects a reference monitor according to the following specification.

Security-Critical Event

Any function calls

Security Policy

Prevent function calls when the first argument is a number of value 42

- If you want to make that very elegant you decompose it into different passes and make it configurable

# Shameless Advertisement

- Yes, I do some of my research using LLVM
- There are topics for Master Theses
- There are topics for Hands-On Courses
- It's all security related
- Please talk to me if you liked the things we did here

# Exercises in this Block

**exercise 6.6**

Parameter Data-Flow Analysis



**exercise 7.1**

Control-Flow Sensitivity



**exercise 7.2**

Handling Loops



**exercise 7.3**

Path- & Context-Sensitivity



**exercise 7.4**

Writing Transformers



**exercise 7.5**

Observe Pointer Operations

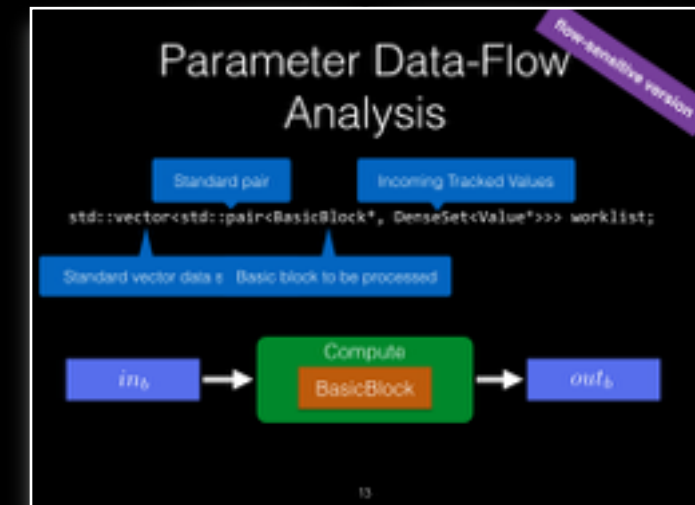
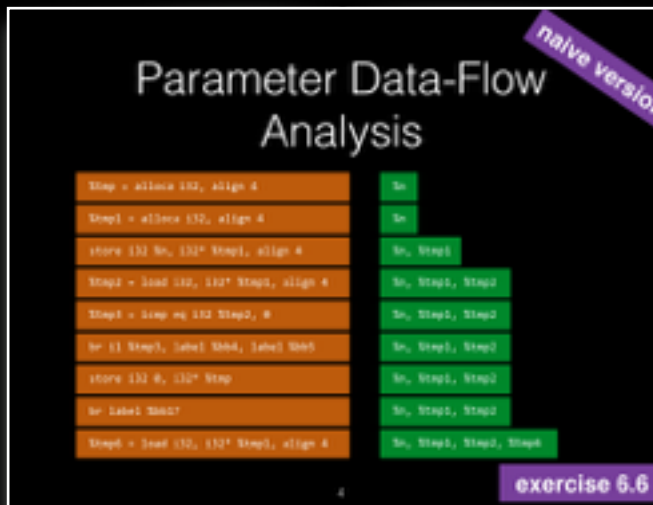


**exercise 7.6**

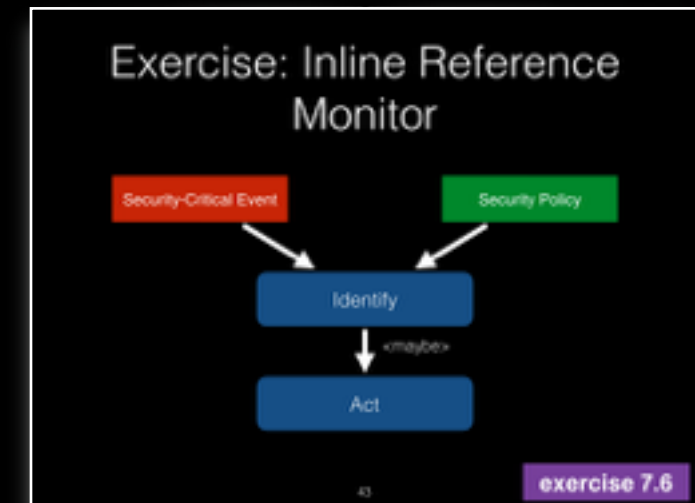
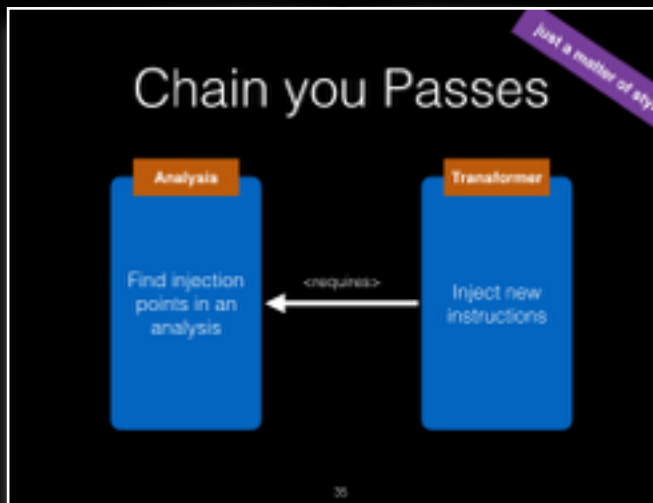
Inline Reference Monitor



# Summary



Writing Transformers



# IFDS-Exercise Set-Up

Compiling OPAL may take some time,  
therefore start with the set up now, if  
not already done.

```
git clone https://bitbucket.org/delors/opal.git
```

```
git clone https://github.com/Sable/heros.git
```

```
git clone https://github.com/stg-tud/apsa.git
```

```
cd opal
```

```
git checkout develop
```

```
sbt publishLocal
```

```
cd ../heros
```

```
cp ant.settings.template ant.settings
```

```
mkdir javadoc
```

```
ant publish-Local
```

```
cd ../apsa/2016/ifds/ifds-exercise
```

```
sbt eclipse
```

Import projects IFDS-exercise and IFDS-testcases in Eclipse

Verify set-up: should compile without errors, some tests should succeed

From within Eclipse select Run As →  
Ant Build... on the build.xml file