Using partial-evaluation to efficiently implement a language in GraalVM

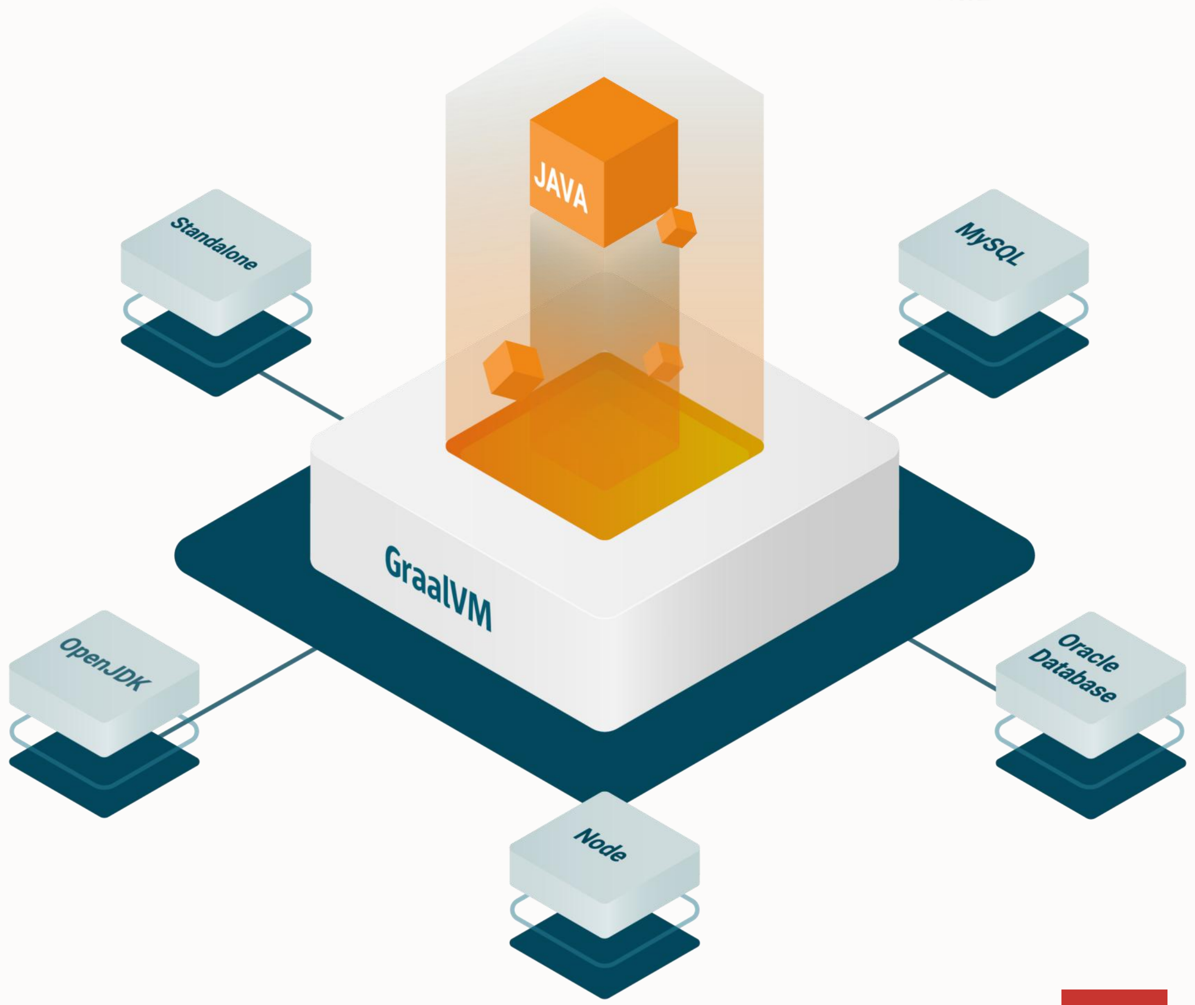
Aleksandar Prokopec

Oracle Labs

December, 2020

Implementing a language in GraalVM

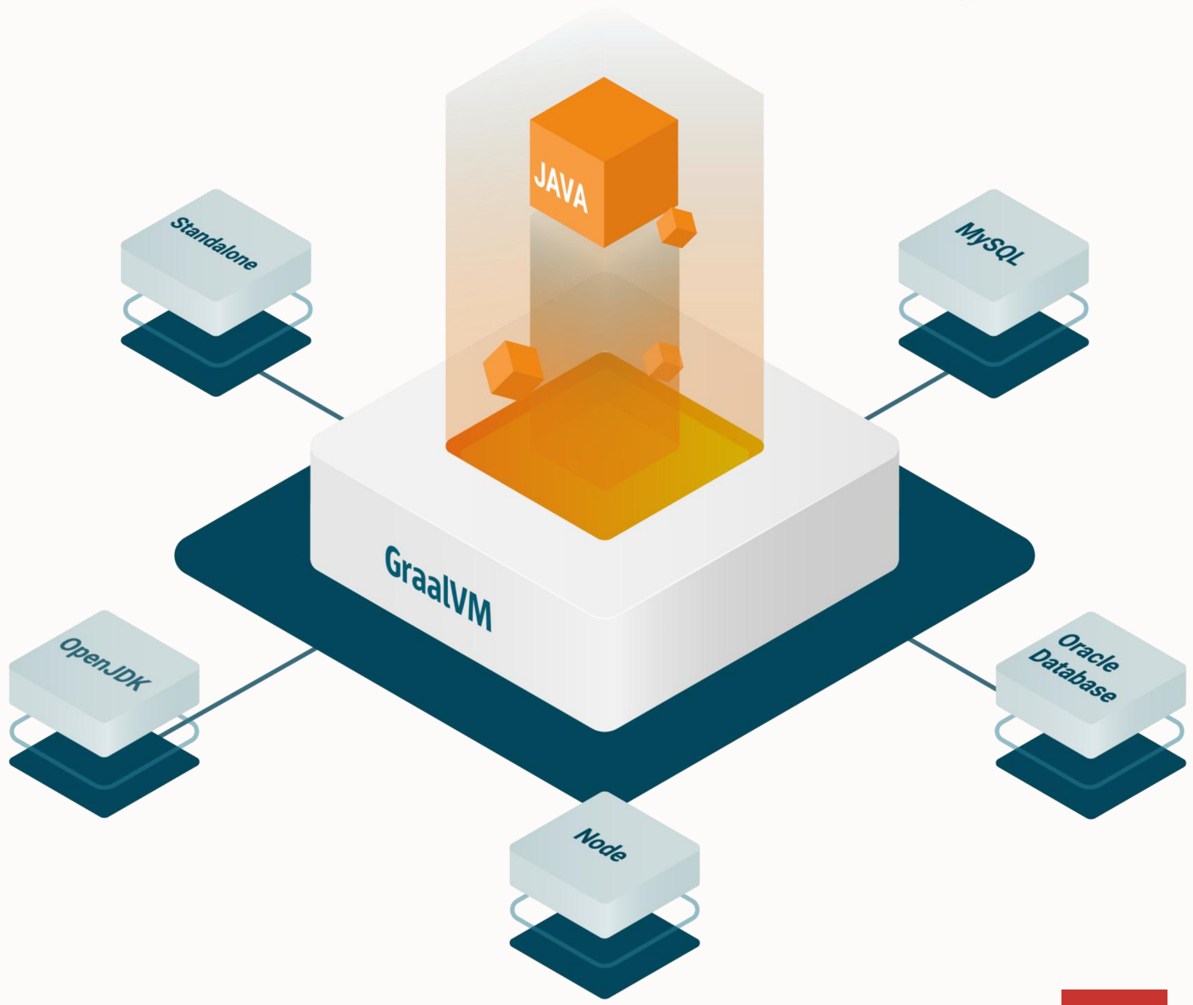
GraalVM is a shiny high-performance next-generation multi-lingual embeddable VM.



Implementing a language in GraalVM

GraalVM is a shiny high-performance next-generation multi-lingual embeddable VM.

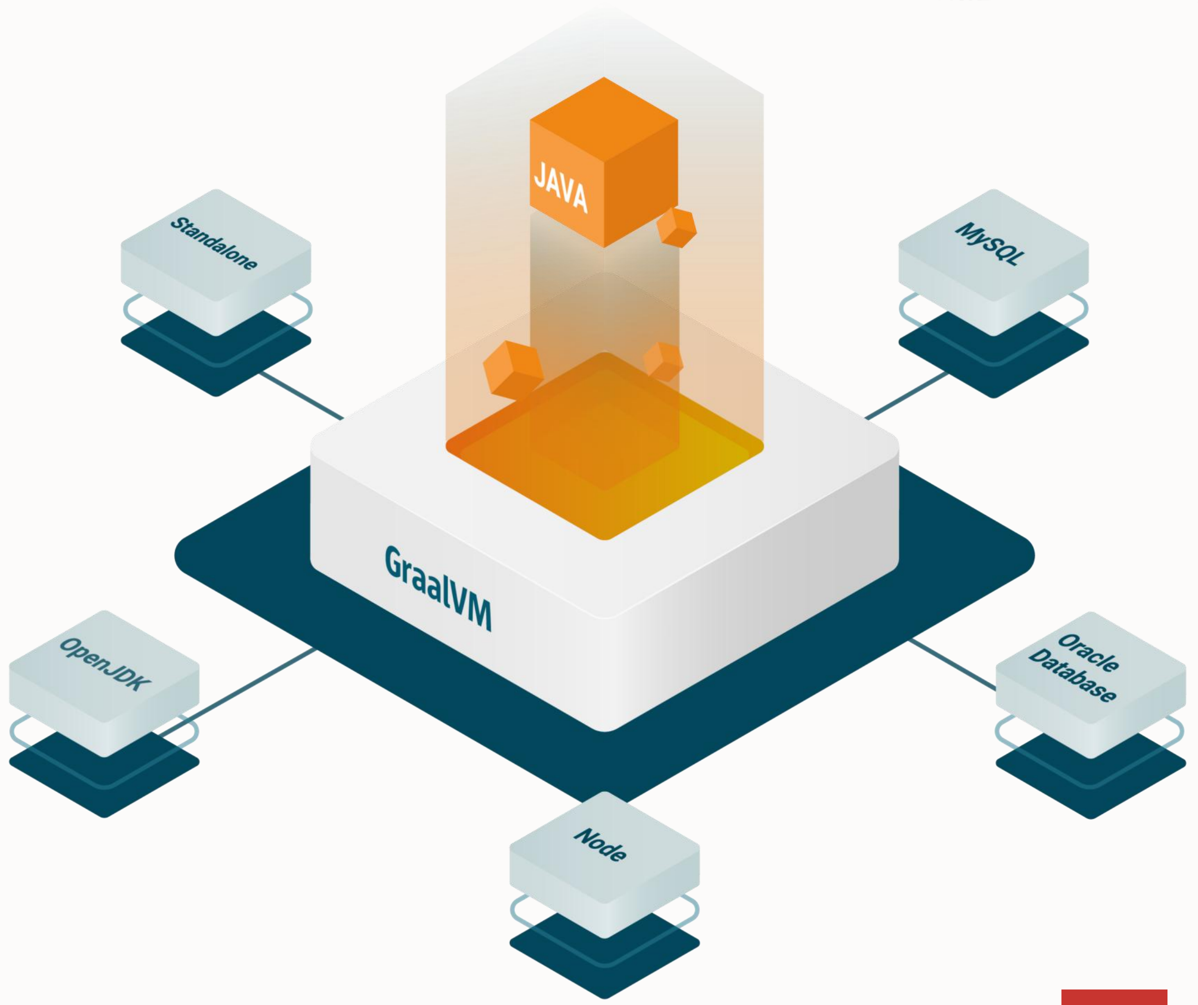
How do you implement a new language for it?



Implementing a language in GraalVM

Steps to create a language

1. Implement an executable AST.
2. Implement a parser that creates the AST for an input program.



Running example

Expression-stack languages with arithmetic and blocks.

Expression-stack language with arithmetic and blocks

```
<instr> =
    i32.const <n>
    i32.add
    i32.eq
    block
      <instr>*
    end
```

Expression-stack language with arithmetic and blocks

```
<instr> =

    i32.const <n>

    i32.add

    i32.eq

    block
      <instr>*
    end
```

Grammar

```
block
  i32.const 7
  i32.const 9
  i32.add
  block
    i32.const 11
    i32.const 5
    i32.add
  end
  i32.eq
end
```

Example program

Expression-stack language with arithmetic and blocks

Let's show the semantics on an example.

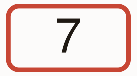
```
block
  i32.const 7
  i32.const 9
  i32.add
  block
    i32.const 11
    i32.const 5
    i32.add
  end
  i32.eq
end
```

Program state

Expression stack and program counter

Example program

Expression-stack language with arithmetic and blocks

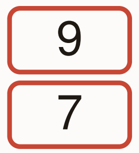


Program state

```
block
  i32.const 7
  i32.const 9
  i32.add
  block
    i32.const 11
    i32.const 5
    i32.add
  end
  i32.eq
end
```

Example program

Expression-stack language with arithmetic and blocks

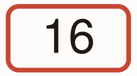


Program state

```
block
  i32.const 7
  i32.const 9
  i32.add
  block
    i32.const 11
    i32.const 5
    i32.add
  end
  i32.eq
end
```

Example program

Expression-stack language with arithmetic and blocks



Program state

```
block
  i32.const 7
  i32.const 9
  i32.add
  block
    i32.const 11
    i32.const 5
    i32.add
  end
  i32.eq
end
```

Example program

Expression-stack language with arithmetic and blocks

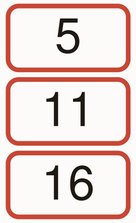
$\boxed{16}$

Program state

```
block
  i32.const 7
  i32.const 9
  i32.add
  block
    i32.const 11
    i32.const 5
    i32.add
  end
  i32.eq
end
```

Example program

Expression-stack language with arithmetic and blocks

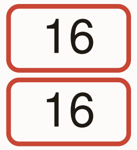


Program state

```
block
  i32.const 7
  i32.const 9
  i32.add
  block
    i32.const 11
    i32.const 5
    i32.add
  end
  i32.eq
end
```

Example program

Expression-stack language with arithmetic and blocks



Program state

```
block
  i32.const 7
  i32.const 9
  i32.add
  block
    i32.const 11
    i32.const 5
    i32.add
  end
  i32.eq
end
```

Example program

Expression-stack language with arithmetic and blocks

1

Program state

```
block
  i32.const 7
  i32.const 9
  i32.add
  block
    i32.const 11
    i32.const 5
    i32.add
  end
  i32.eq
end
```

Example program

Expression-stack language with arithmetic and blocks

Let’s capture the semantics in an interpreter.

$\boxed{1}$

Program state

```
block
  i32.const 7
  i32.const 9
  i32.add
  block
    i32.const 11
    i32.const 5
    i32.add
  end
  i32.eq
end
```

Example program

Implementing a language with an interpreter loop

Step 1: extend the Node class

```
class BlockNode extends Node {
  final int offset;
  final byte[] program;

  BlockNode(int offset, byte[] program) {
    this.offset = offset;
    this.program = program;
  }

  Object execute(Frame frame) { ... }
}
```

```
 0 : block
 1 :  i32.const 7
 2 :  i32.const 9
 3 :  i32.add
 4 :  block
 5 :    i32.const 11
 6 :    i32.const 5
 7 :    i32.add
 8 :  end
 9 :  i32.eq
10 : end
```

Implementing a language with an interpreter loop

Step 2: implement the execute method

```
Object execute(Frame frame) {

}
```

Think of it as an array of local variables

```
 0 : block
 1 :  i32.const 7
 2 :  i32.const 9
 3 :  i32.add
 4 :  block
 5 :    i32.const 11
 6 :    i32.const 5
 7 :    i32.add
 8 :  end
 9 :  i32.eq
10 : end
```

Implementing a language with an interpreter loop

Step 2: implement the execute method

```
Object execute(Frame frame) {
  int pc = offset;
  while (program[pc] != END) {







  }
  return null;
}
```

```
 0 : block
 1 :  i32.const 7
 2 :  i32.const 9
 3 :  i32.add
 4 :  block
 5 :    i32.const 11
 6 :    i32.const 5
 7 :    i32.add
 8 :  end
 9 :  i32.eq
10 : end
```

Implementing a language with an interpreter loop

Step 2: implement the execute method

```
Object execute(Frame frame) {
  int pc = offset;
  while (program[pc] != END) {
    switch (program[pc++]) {





    }
  }
  return null;
}
```

```
 0 : block
 1 :  i32.const 7
 2 :  i32.const 9
 3 :  i32.add
 4 :  block
 5 :    i32.const 11
 6 :    i32.const 5
 7 :    i32.add
 8 :  end
 9 :  i32.eq
10 : end
```

Implementing a language with an interpreter loop

Step 2: implement the execute method

```
Object execute(Frame frame) {
  int pc = offset;
  while (program[pc] != END) {
    switch (program[pc++]) {
      case I32_CONST: ...
      case I32_ADD: ...
      case I32_EQ: ...
      case BLOCK: ...
    }
  }
  return null;
}
```

```
 0 : block
 1 :  i32.const 7
 2 :  i32.const 9
 3 :  i32.add
 4 :  block
 5 :    i32.const 11
 6 :    i32.const 5
 7 :    i32.add
 8 :  end
 9 :  i32.eq
10 : end
```

Implementing a language with an interpreter loop

Step 2: implement the execute method

```
Object execute(Frame frame) {
  int pc = offset;
  while (program[pc] != END) {
    switch (program[pc++]) {
      case I32_ADD:
        int x = pop(frame);
        int y = pop(frame);
        push(frame, x + y);
        break;
    }
  }
  return null;
}
```

```
 0: block
 1:  i32.const 7
 2:  i32.const 9
 3:  i32.add
 4:  block
 5:    i32.const 11
 6:    i32.const 5
 7:    i32.add
 8:  end
 9:  i32.eq
10: end
```

Implementing a language with an interpreter loop

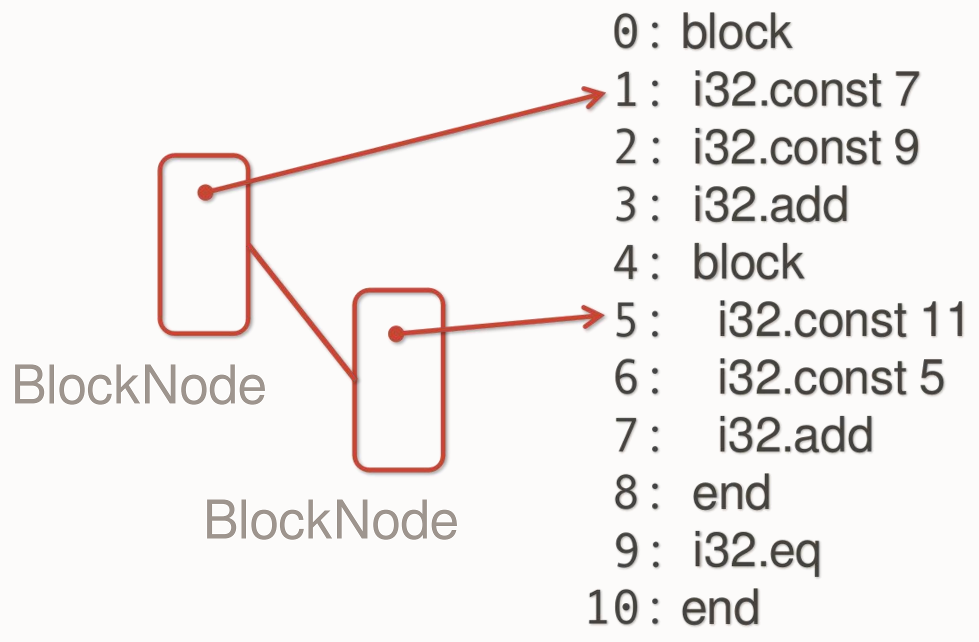
Step 2: implement the execute method

```
Object execute(Frame frame) {
  int pc = offset;
  while (program[pc] != END) {
    switch (program[pc++]) {
      case BLOCK:
        BlockNode b = children[pc];
        b.execute(frame);
        break;

    }
  }
  return null;
}
```

```
 0: block
 1:  i32.const 7
 2:  i32.const 9
 3:  i32.add
 4:  block
 5:    i32.const 11
 6:    i32.const 5
 7:    i32.add
 8:  end
 9:  i32.eq
10: end
```

Implementing a language with an interpreter loop

AST holds offsets into the instructions array



```
0 : block
1 :  i32.const 7
2 :  i32.const 9
3 :  i32.add
4 :  block
5 :    i32.const 11
6 :    i32.const 5
7 :    i32.add
8 :  end
9 :  i32.eq
10 : end
```

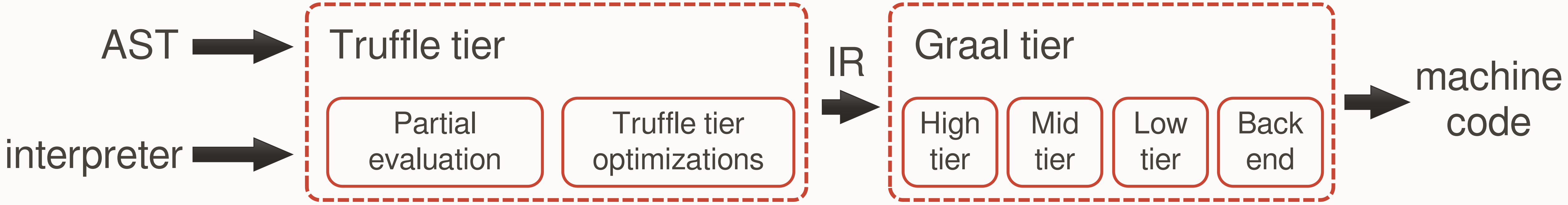
Congratulations

You're now the local WebAssembly expert.

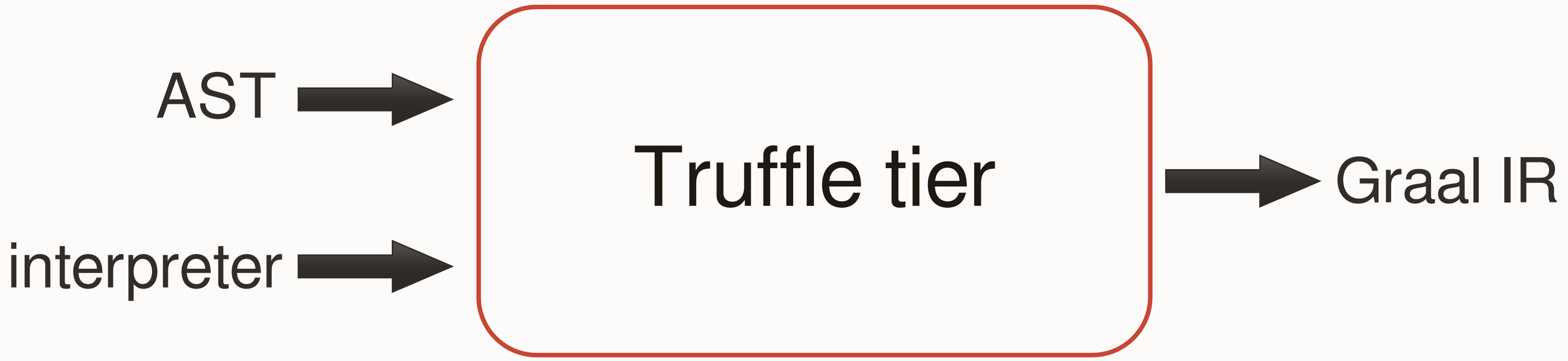
How compilation works

in Graal and Truffle

Truffle compiler pipeline

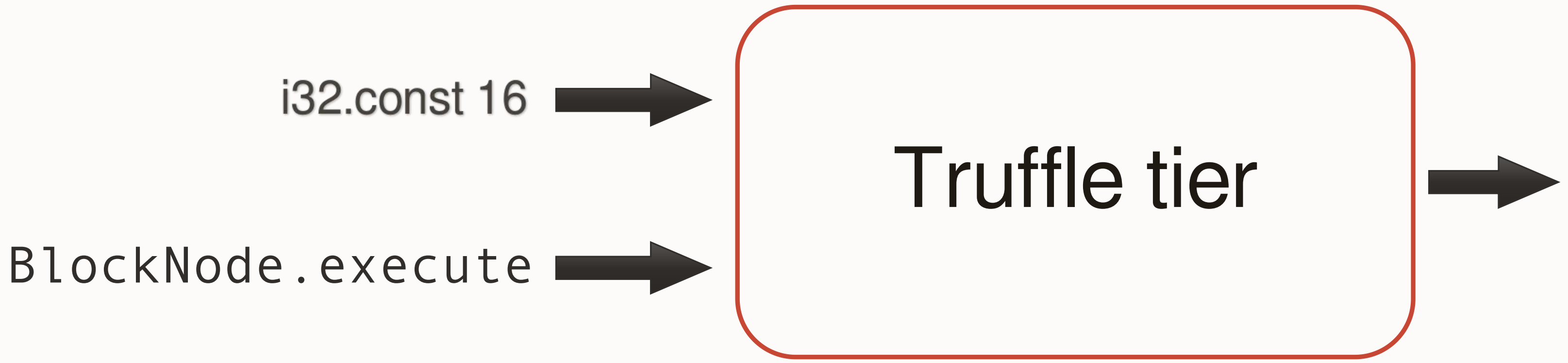


Partial evaluation

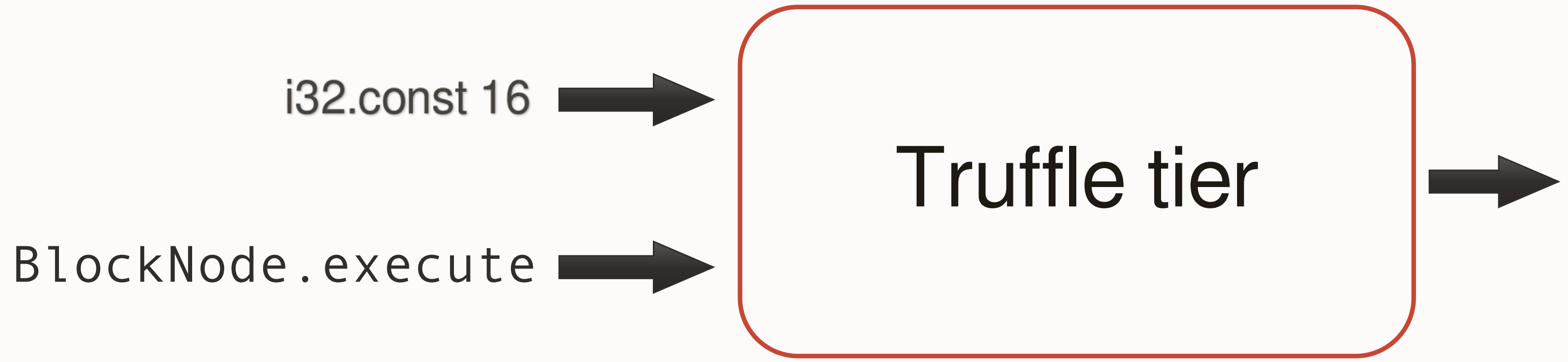


Specializes the interpreter code to a given AST.

Partial evaluation

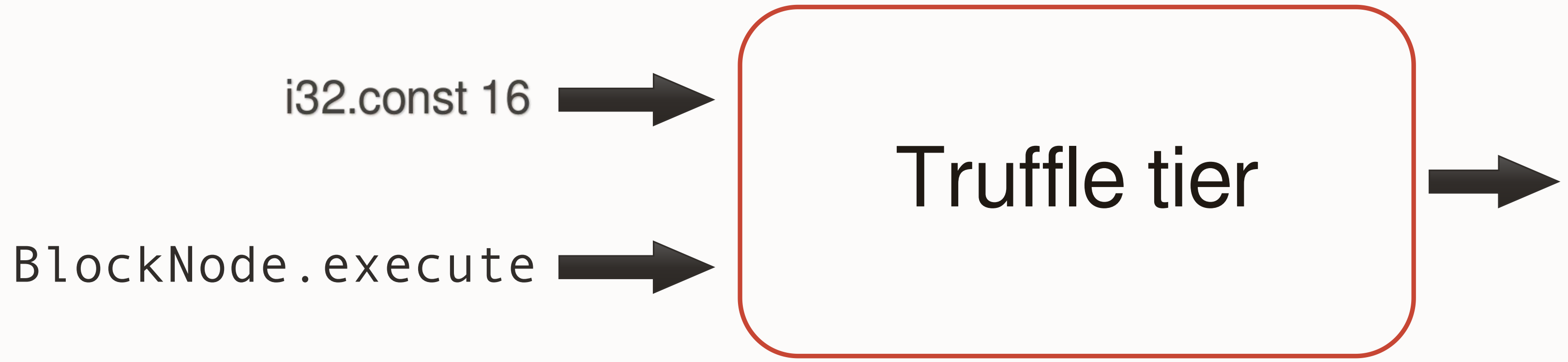


Partial evaluation



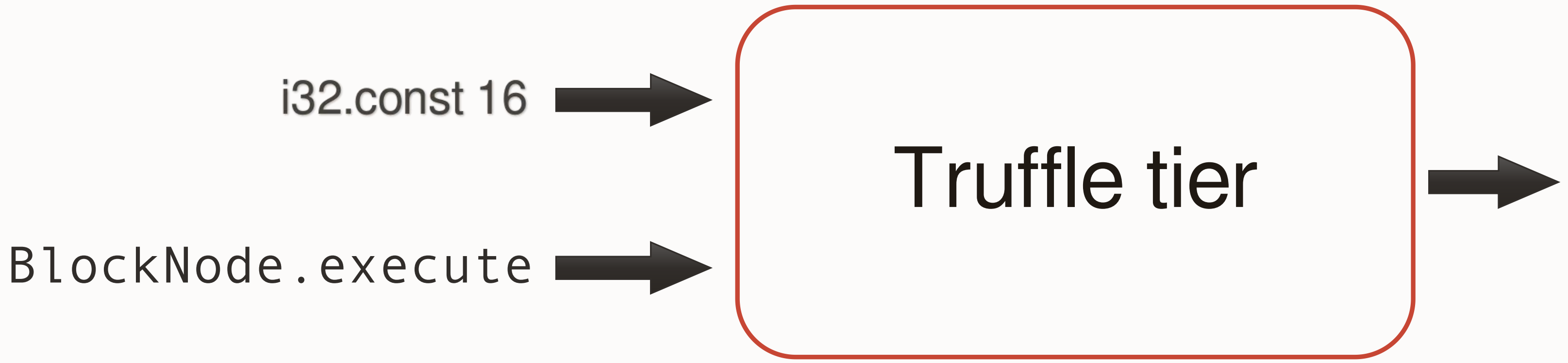
```
Object execute(Frame frame) {
  int pc = offset;
  while (program[pc] != END) {
    switch (program[pc++]) {
      case I32_ADD: ...
      case I32_CONST: ...
      ...
    }
  }
  return null;
}
```

Partial evaluation



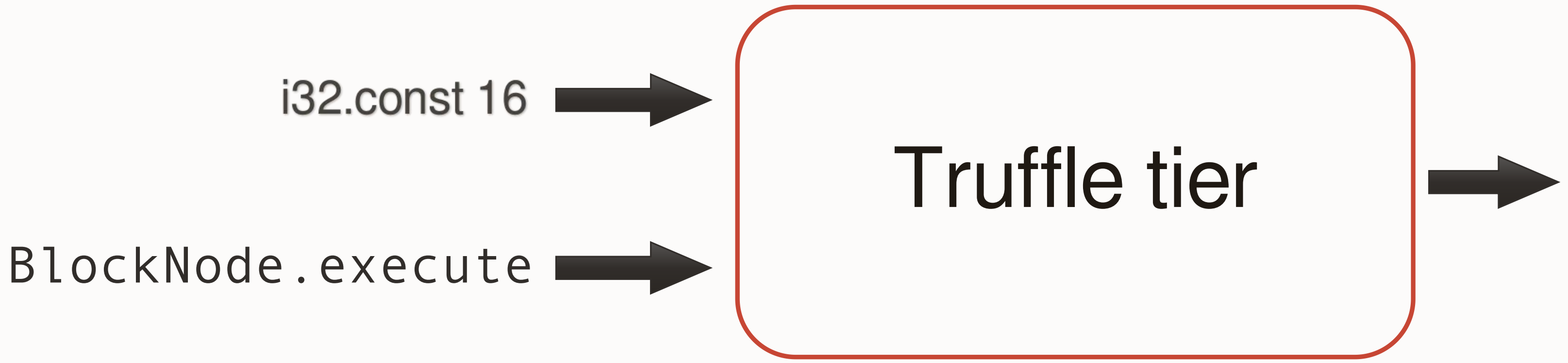
```
Object execute(Frame frame) {
  int pc = 0;
  while (program[pc] != END) {
    switch (program[pc++]) {
      case I32_ADD: ...
      case I32_CONST: ...
      ...
    }
  }
  return null;
}
```

Partial evaluation



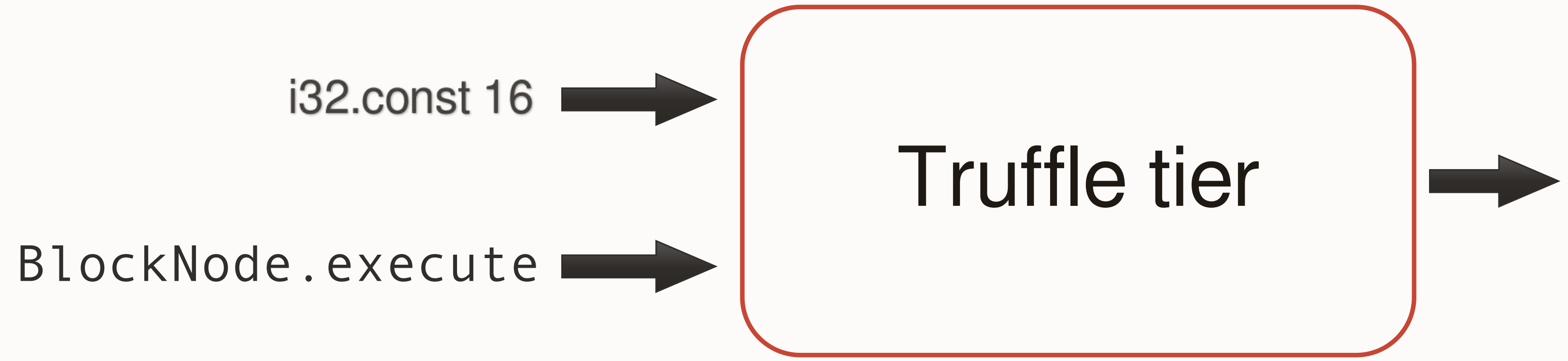
```
Object execute(Frame frame) {
  int pc = 0;
  switch (program[0]) {
    case I32_ADD: ...
    case I32_CONST: ...
    ...
  }
  return null;
}
```

Partial evaluation



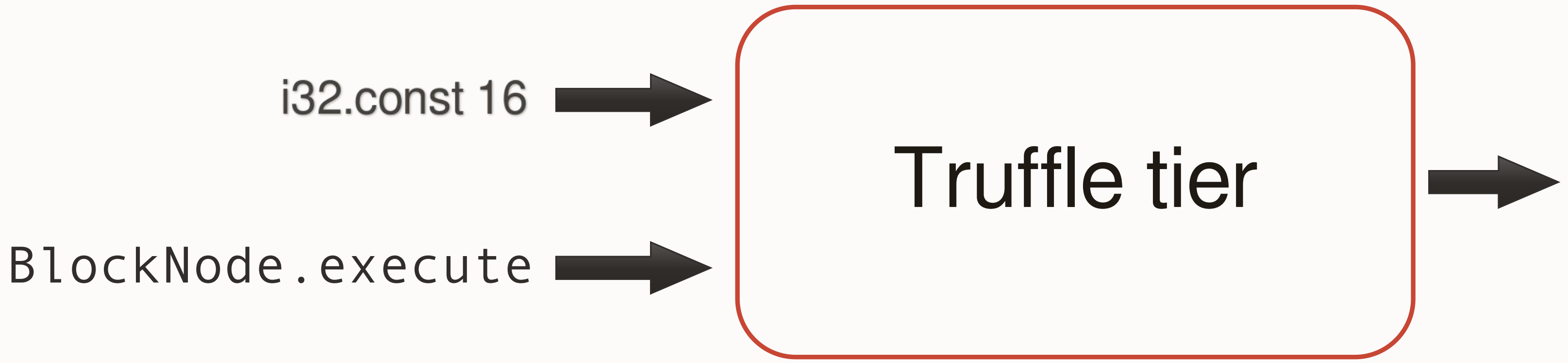
```
Object execute(Frame frame) {
  int pc = 0;
  switch (I32_CONST) {
    case I32_ADD: ...
    case I32_CONST: ...
    ...
  }
  return null;
}
```

Partial evaluation



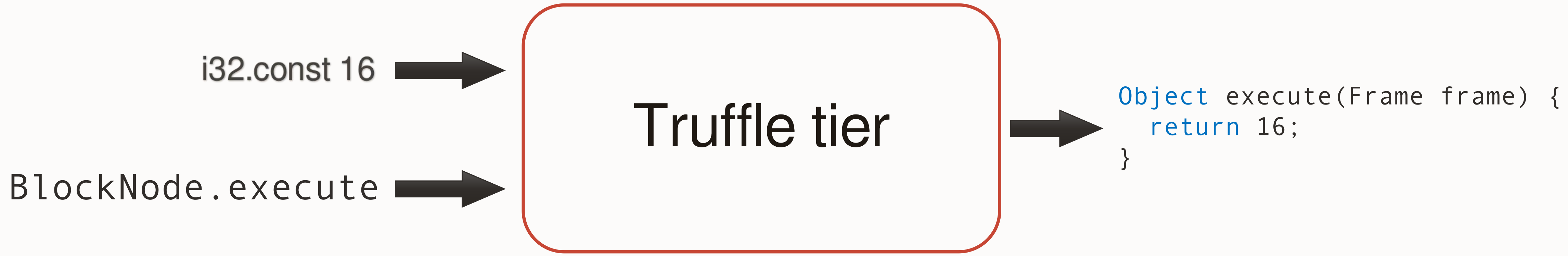
```
Object execute(Frame frame) {
  int pc = 0;

  int x = program[1];
  push(frame, x);


  return null;
}
```

Partial evaluation



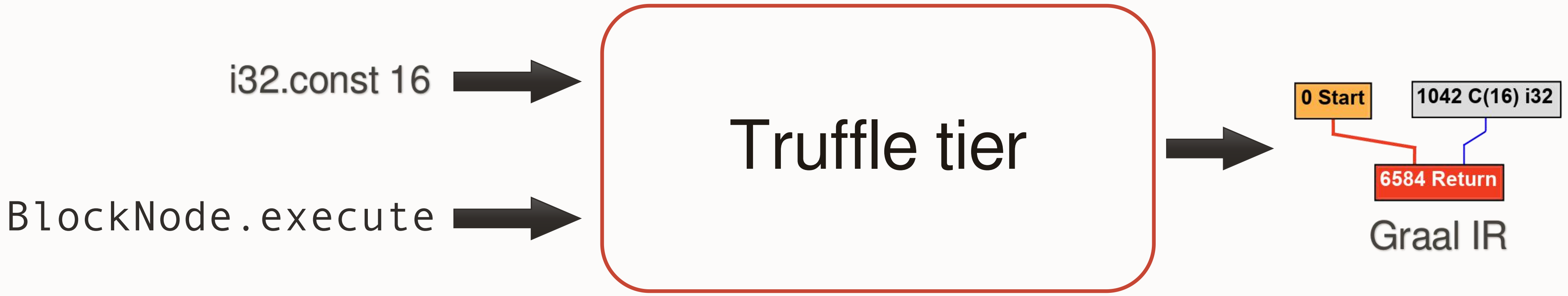
```
Object execute(Frame frame) {
  int pc = 0;

  int x = 16;
  push(frame, x);


  return null;
}
```

Partial evaluation

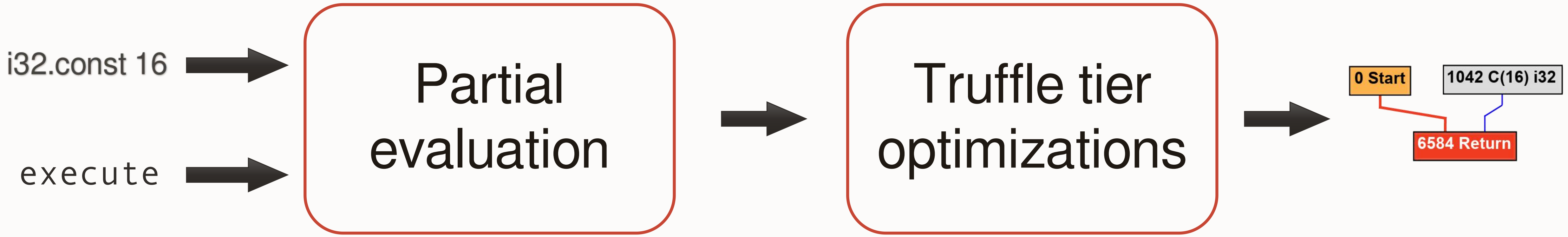


```
Object execute(Frame frame) {
  return 16;
}
```

Partial evaluation



Partial evaluation

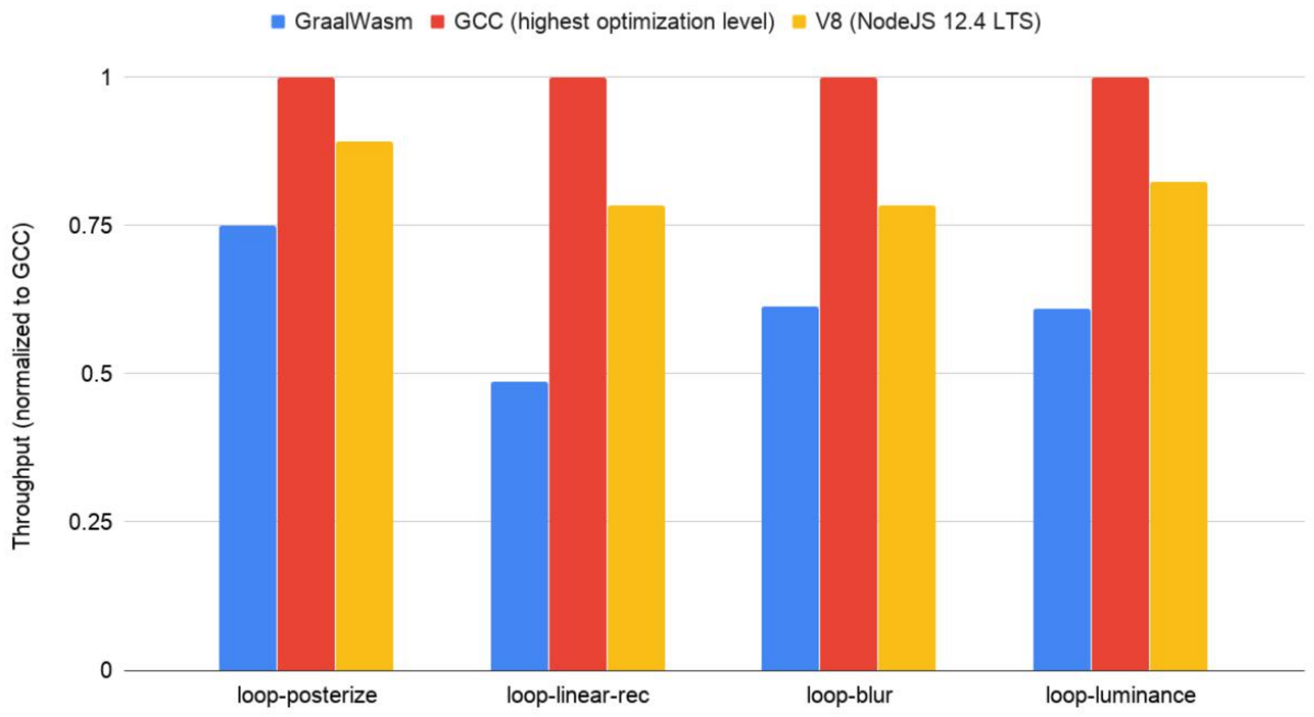


Most of the work is done in partial evaluation,
and then optimizations simplify the IR.

Stories

in performance engineering

Start by comparing with another VM



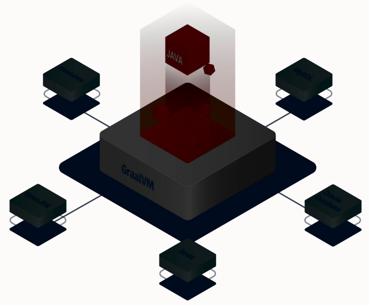
How do I improve the performance?

Look at the data

Understanding what's going on will help you solve the problem.

Look at the data

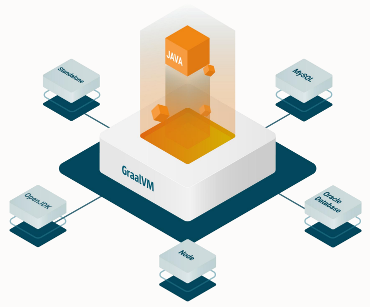
Understanding what's going on will help you solve the problem.



Your problem is not a black box.

Look at the data

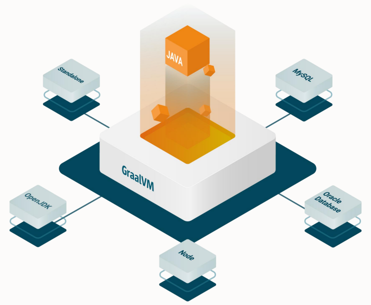
Understanding what's going on will help you solve the problem.



Your problem is not a black box.

Look at the compiler IR

Understanding what the compiler works with
will help you solve the problem.



Memory out-of-bounds access profiling

Example program: invert filter

Memory out-of-bounds access profiling

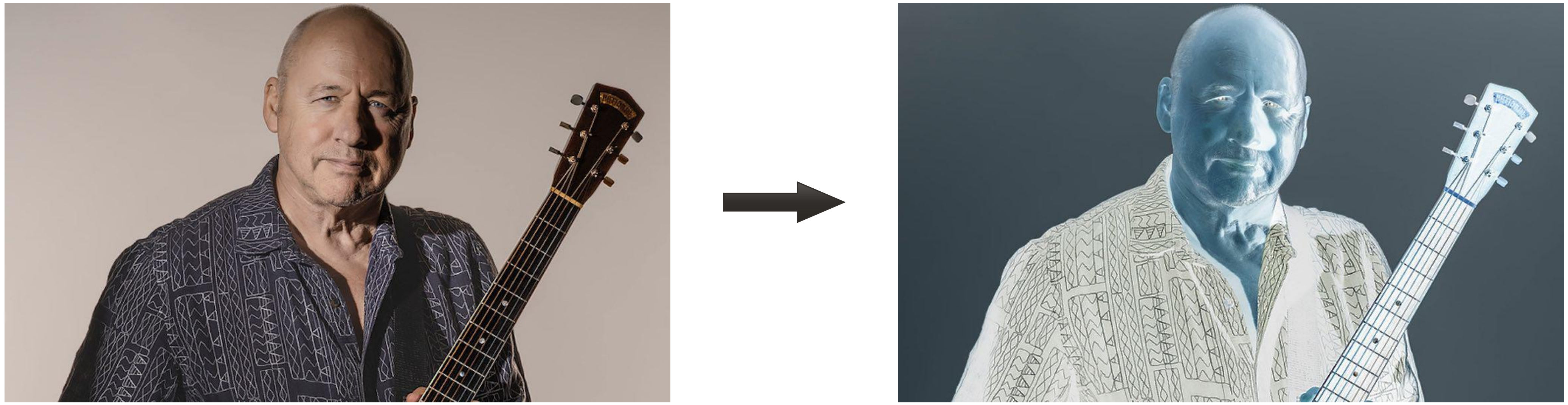
Example program: invert filter

```c
for (uint32_t pixel = 0; pixel != IMAGE_SIZE; ++pixel) {
  uint32_t color = image[pixel % IMAGE_SIZE];
  uint8_t R = 0xFF - ((color & 0xFF000000) >> 24);
  uint8_t G = 0xFF - ((color & 0x00FF0000) >> 16);
  uint8_t B = 0xFF - ((color & 0x0000FF00) >> 8);
  result[pixel % IMAGE_SIZE] = (R << 24) + (G << 16) + (B << 8) + 0xFF;
}
```

C program

Memory out-of-bounds access profiling

Example program: invert filter



Memory out-of-bounds access profiling

Example program: invert filter

```c
for (uint32_t pixel = 0; pixel != IMAGE_SIZE; ++pixel) {
  uint32_t color = image[pixel % IMAGE_SIZE];
  uint8_t R = 0xFF - ((color & 0xFF000000) >> 24);
  uint8_t G = 0xFF - ((color & 0x00FF0000) >> 16);
  uint8_t B = 0xFF - ((color & 0x0000FF00) >> 8);
  result[pixel % IMAGE_SIZE] = (R << 24) + (G << 16) + (B << 8) + 0xFF;
}
```

C program

Memory out-of-bounds access profiling

Example program: invert filter

```
for (uint32_t pixel = 0; pixel != IMAGE_SIZE; ++pixel) {
  uint32_t color = image[pixel % IMAGE_SIZE];
  uint8_t R = 0xFF - ((color & 0xFF000000) >> 24);
  uint8_t G = 0xFF - ((color & 0x00FF0000) >> 16);
  uint8_t B = 0xFF - ((color & 0x0000FF00) >> 8);
  result[pixel % IMAGE_SIZE] = (R << 24) + (G << 16) + (B << 8) + 0xFF;
}
```

Memory read

Memory write

C program

Memory out-of-bounds access profiling

Example program: invert filter

```
  loop  ;; label = @1
    local.get 2
    i32.const 2
...
    i32.const 2064
    i32.add
    i32.load
    local.tee 0
    i32.const 8
...
    i32.store
...
    i32.ne
    br_if 0 (;@1;)
  end
```

Memory read

Memory write

WebAssembly program

Memory out-of-bounds access profiling

How to implement memory reads and writes in a Truffle interpreter?

1. Allocate a region of memory for the Truffle program.

```
public UnsafeWasmMemory(int pageCount) {
  this.startAddress = unsafe.allocateMemory(pageCount * PAGE_SIZE);
}
```

Memory out-of-bounds access profiling

How to implement memory reads and writes in a Truffle interpreter?

1. Allocate a region of memory for the Truffle program.
2. Implement methods that read and write to memory.

i32.load
i32.store
and a few more…

WebAssembly

```
int load_i32(long address) {
  if (address < 0 || address + 4 > this.pageCount * PAGE_SIZE) {
    trapOutOfBounds(address, 4);
  }
  int value = unsafe.getInt(this.startAddress + address);
  return value;
}
```

GraalWasm

Memory out-of-bounds access profiling

How to implement memory reads and writes in a Truffle interpreter?

1. Allocate a region of memory for the Truffle program.
2. Implement methods that read and write to memory.
3. Call those methods when interpreting loads and stores.

```
case I32_LOAD: {
  int address = frame[--stackPointer];
  int value = memory.load_i32(address);
  frame[stackPointer] = value;
  stackPointer++;
  break;
}
```

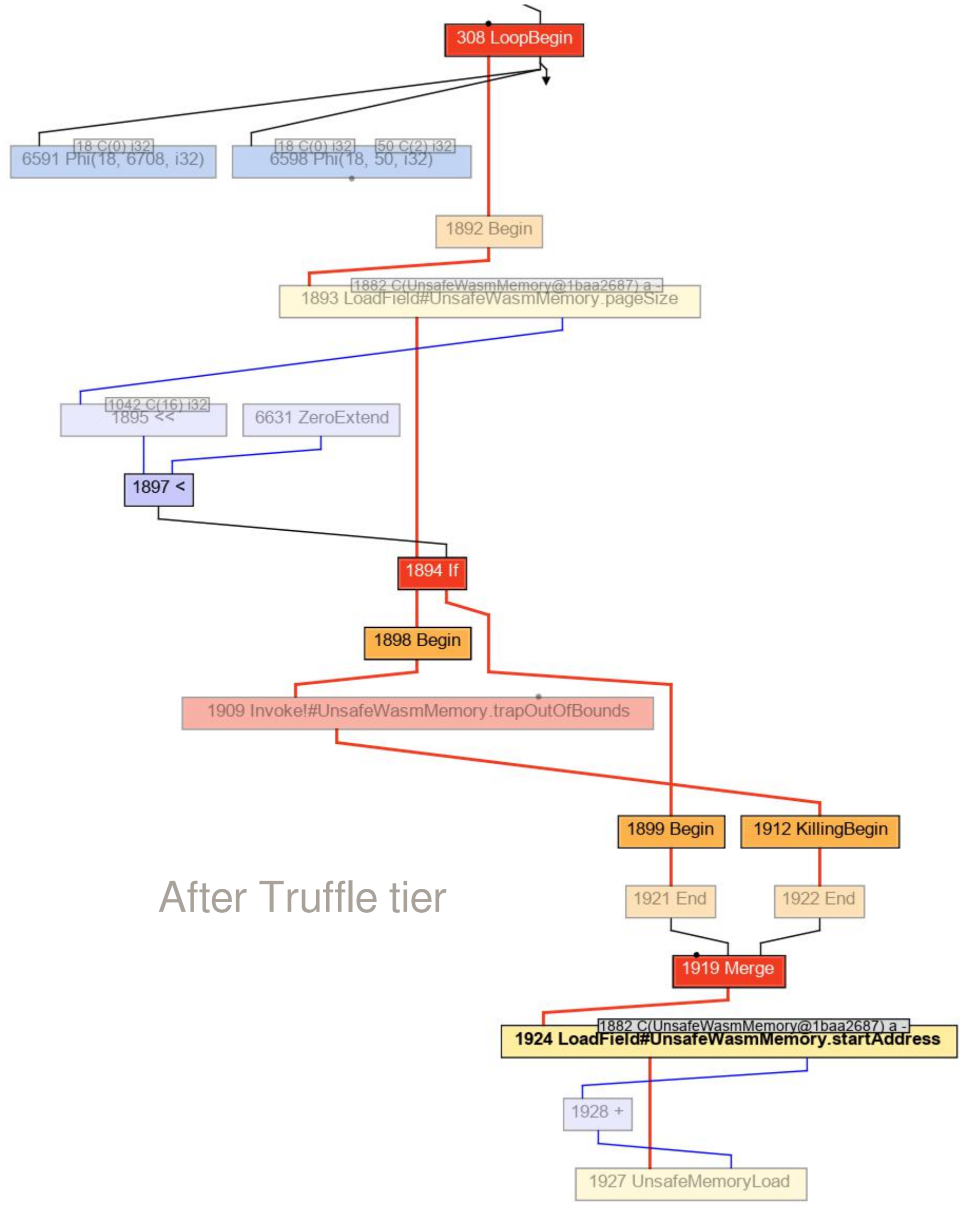
GraalWasm

Memory out-of-bounds access profiling

What does our program look like after partial evaluation completes?

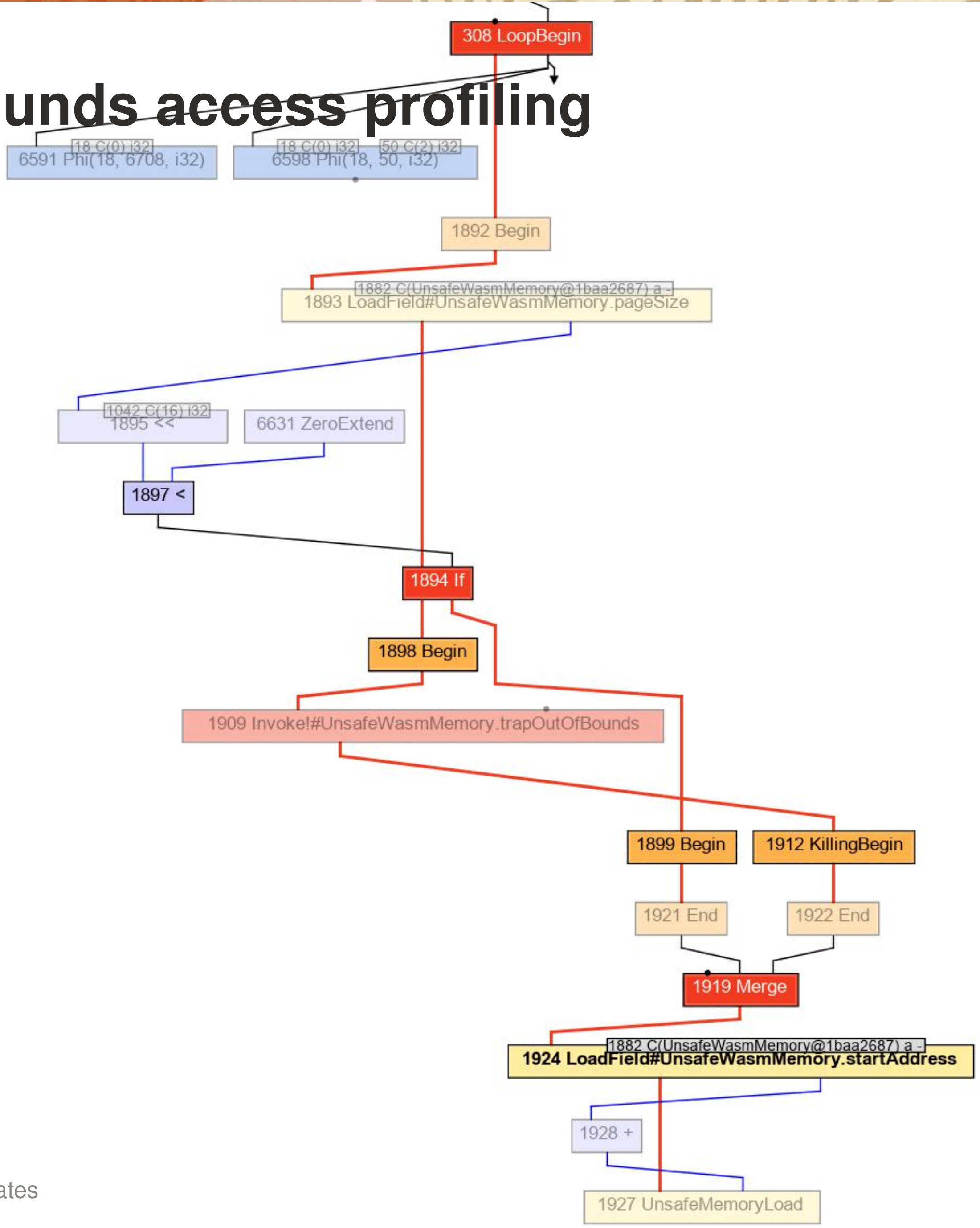
```
case I32_LOAD: {
  int address = frame[--stackPointer];
  int value = memory.load_i32(address);
  frame[stackPointer] = value;
  stackPointer++;
  break;
}
```

GraalWasm



After Truffle tier

Memory out-of-bounds access profiling

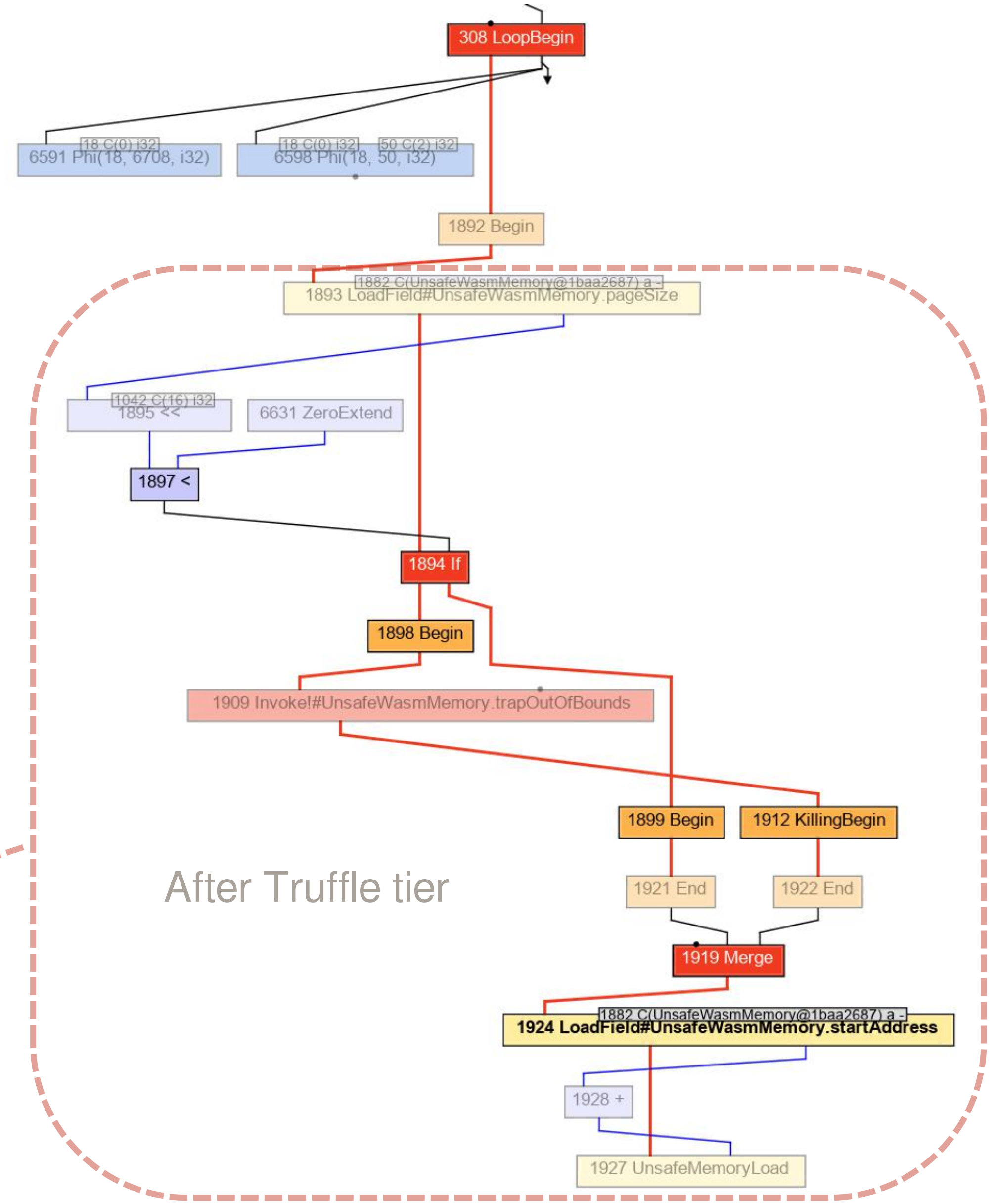


Memory out-of-bounds access profiling

The part that corresponds to the bounds check and the memory read

```
case I32_LOAD: {
  int address = frame[--stackPointer];
  int value = memory.load_i32(address);
  frame[stackPointer] = value;
  stackPointer++;
  break;
}
```

GraalWasm

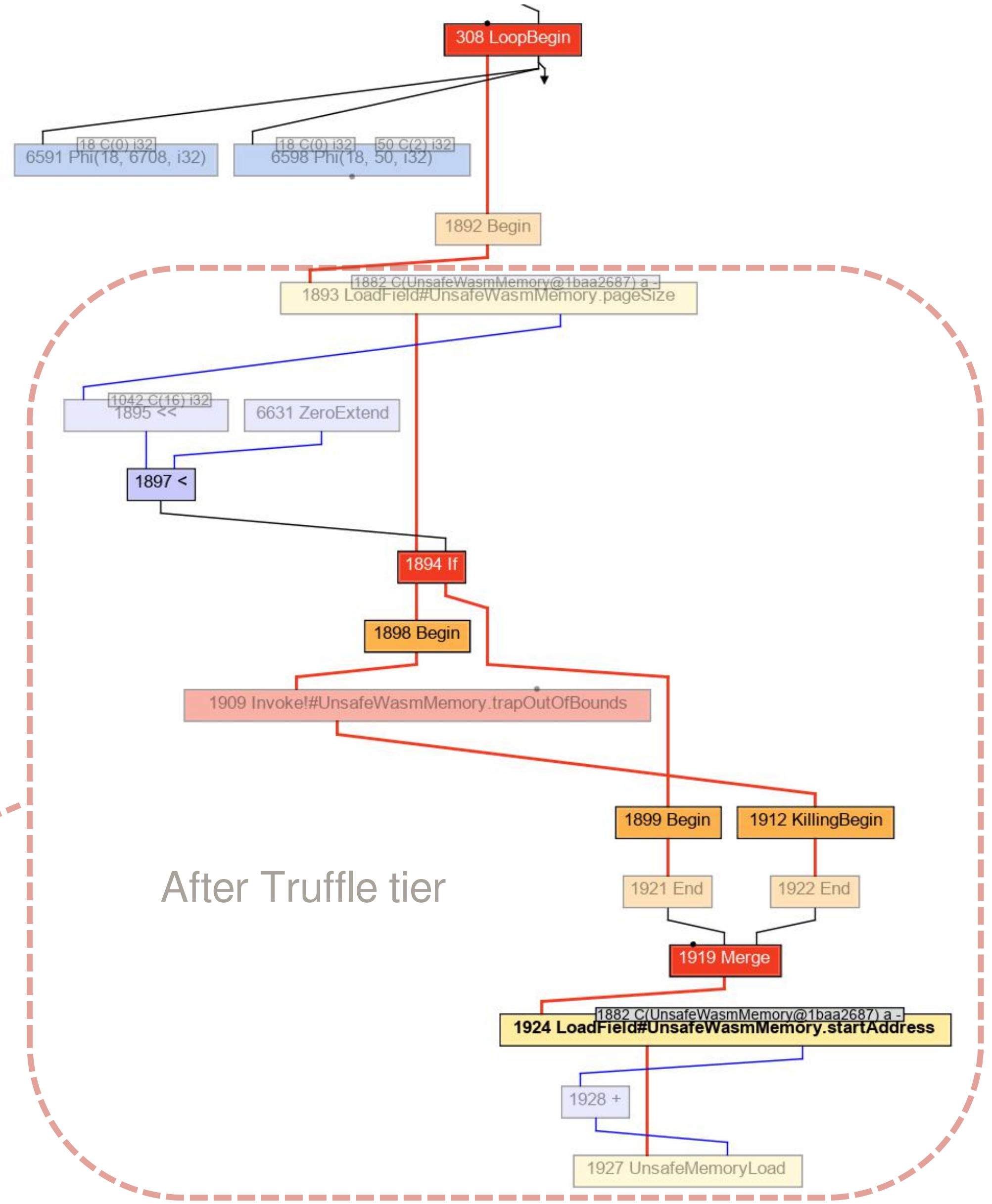


Memory out-of-bounds access profiling

The part that corresponds to the bounds check and the memory read

```
if (address + 4 > pageCount * PAGE_SIZE) {
  trapOutOfBounds(address, 4);
}
return unsafe.getInt(startAddress + address);
```

GraalWasm

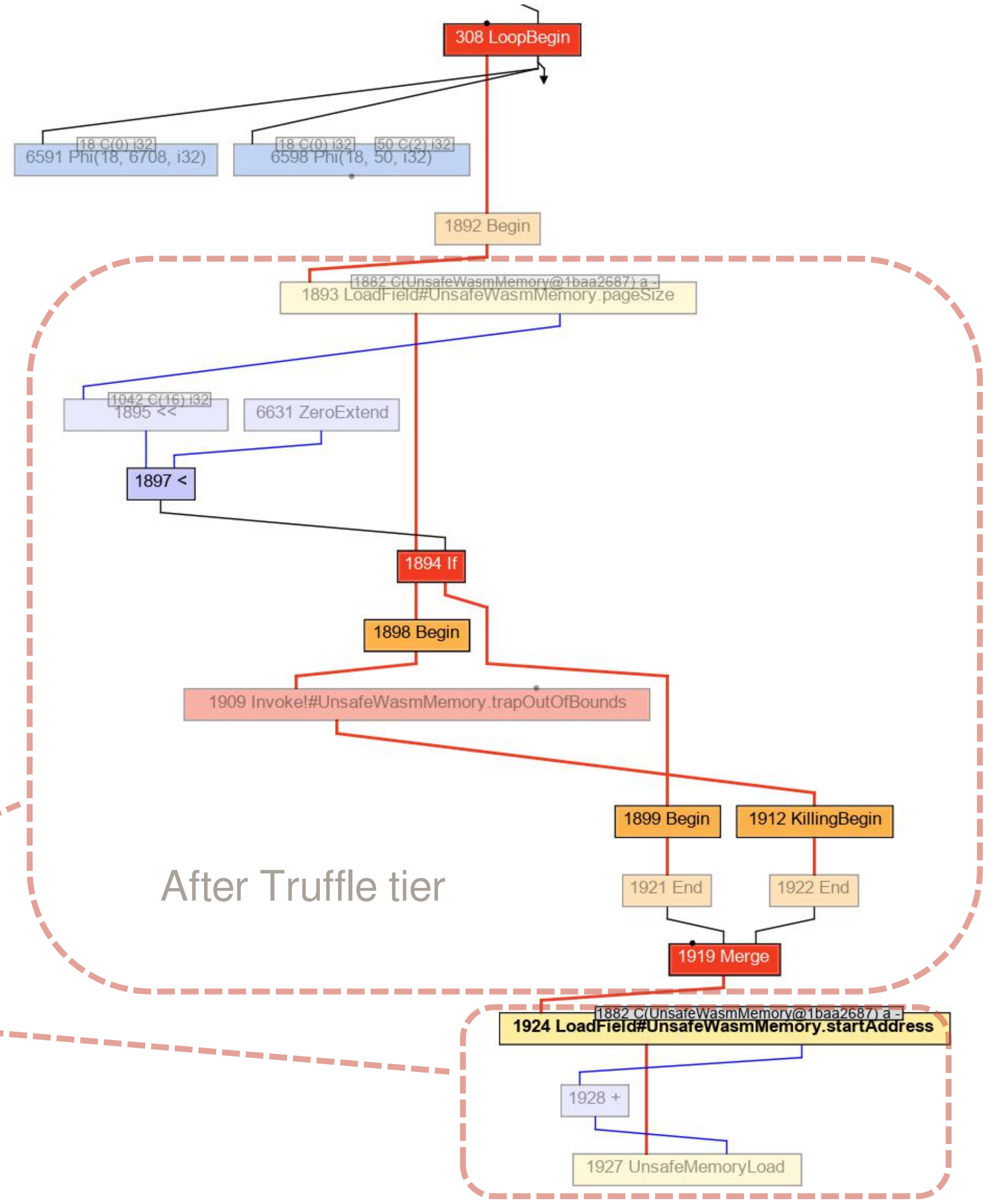


Memory out-of-bounds access profiling

The part that corresponds to the bounds check and the memory read

```
if (address + 4 > pageCount * PAGE_SIZE) {
  trapOutOfBounds(address, 4);
}
return unsafe.getInt(startAddress + address);
```

GraalWasm



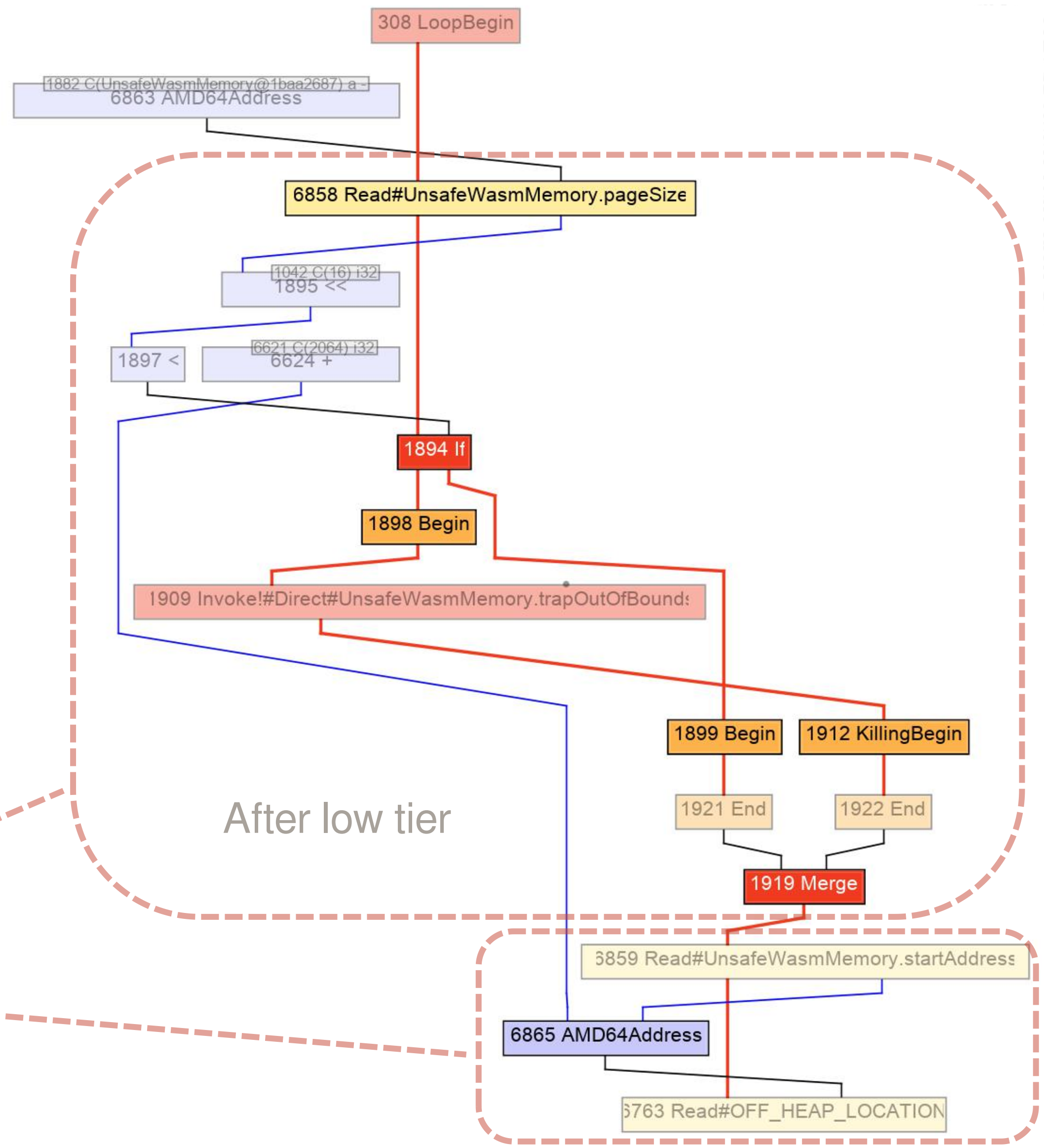
Memory out-of-bounds access profiling

It is important to take a look at the IR after the low tier as well

LoadField nodes on Java objects are converted into Read nodes, which may have been moved around the IR.

```
if (address + 4 > pageCount * PAGE_SIZE) {
  trapOutOfBounds(address, 4);
}
return unsafe.getInt(startAddress + address);
```

GraalWasm

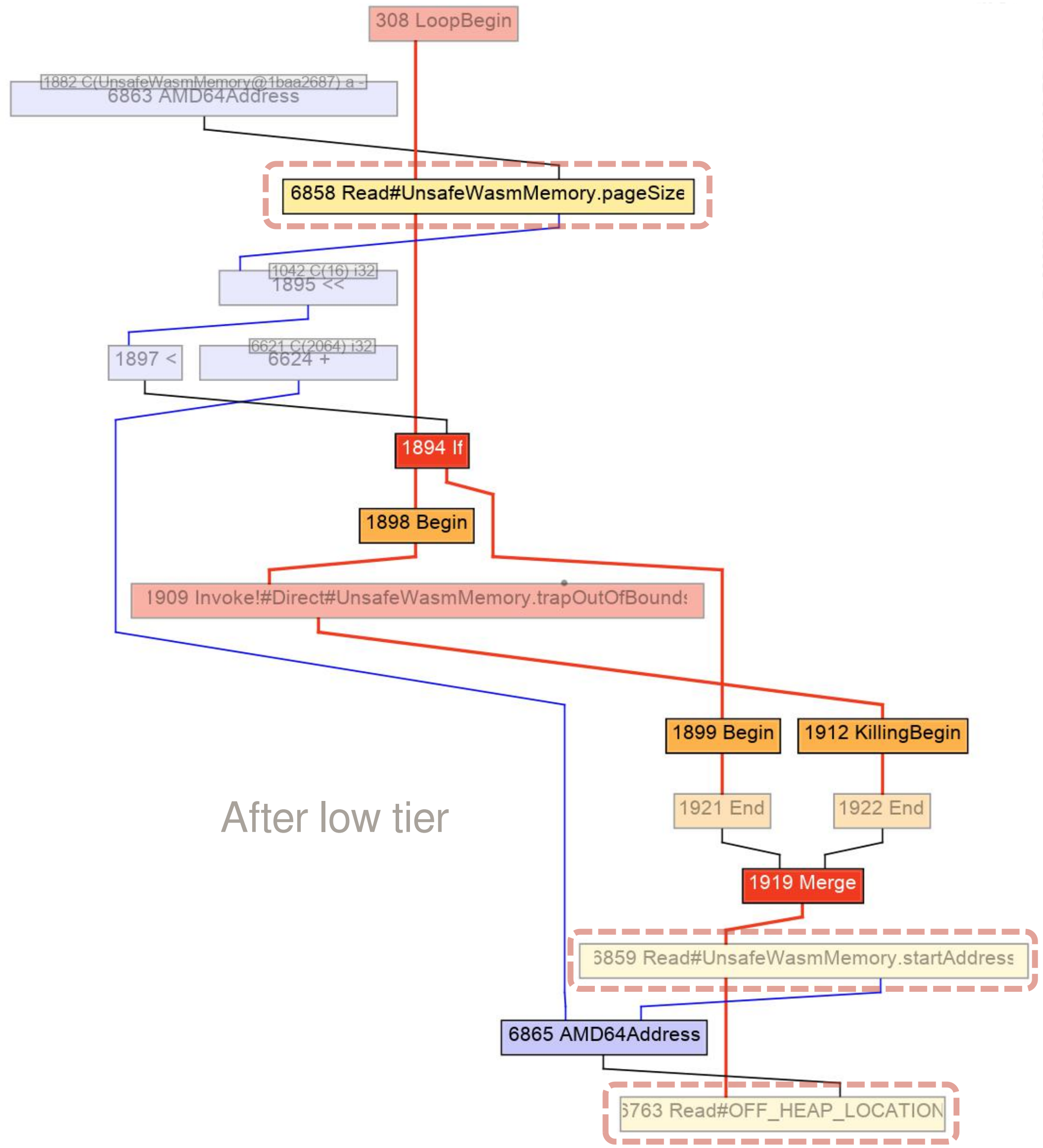


308 LoopBegin
1882 C(UnsafeWasmMemory@1baa2687) a
6863 AMD64Address
6858 Read#UnsafeWasmMemory.pageSize
1042 C(16) i32
1895 <<
1897 <
6621 C(2064) i32
6624 +
1894 If
1898 Begin
1909 Invoke!#Direct#UnsafeWasmMemory.trapOutOfBounds
1899 Begin
1912 KillingBegin
1921 End
1922 End
After low tier
1919 Merge
6859 Read#UnsafeWasmMemory.startAddress
6865 AMD64Address
6763 Read#OFF_HEAP_LOCATION

Memory out-of-bounds access profiling

We have 3 memory-read nodes for every memory read in the program

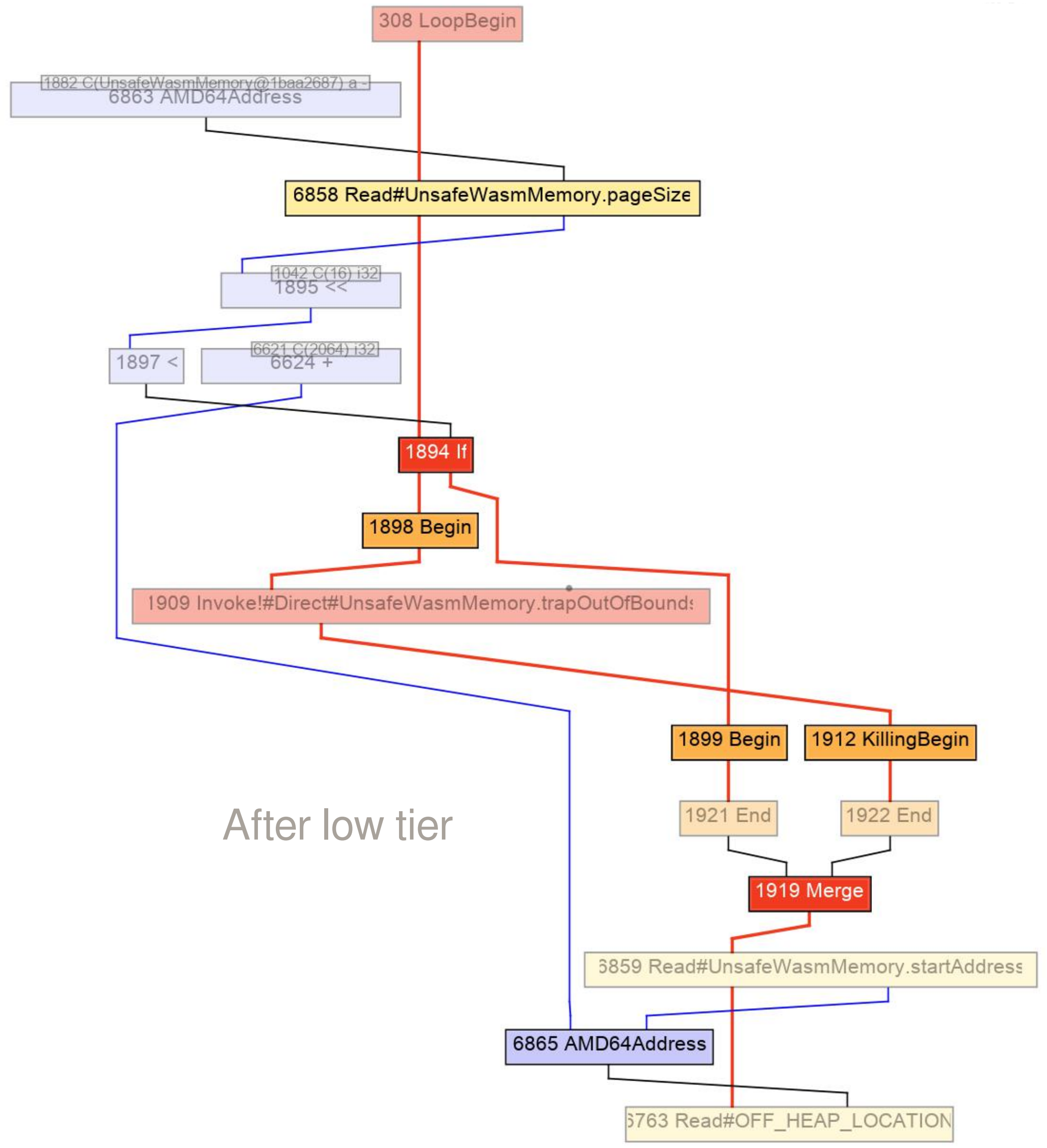
We would expect that the size of the heap and the start-address of the heap are loop-invariant.


308 LoopBegin
1882 C(UnsafeWasmMemory@1baa2687) a
6863 AMD64Address
6858 Read#UnsafeWasmMemory.pageSize
1042 C(16) i32
1895 <<
1897 <
6621 C(2064) i32
6624 +
1894 If
1898 Begin
1909 Invoke!#Direct#UnsafeWasmMemory.trapOutOfBounds
1899 Begin
1912 KillingBegin
1921 End
1922 End
1919 Merge
6859 Read#UnsafeWasmMemory.startAddress
6865 AMD64Address
6763 Read#OFF_HEAP_LOCATION

After low tier

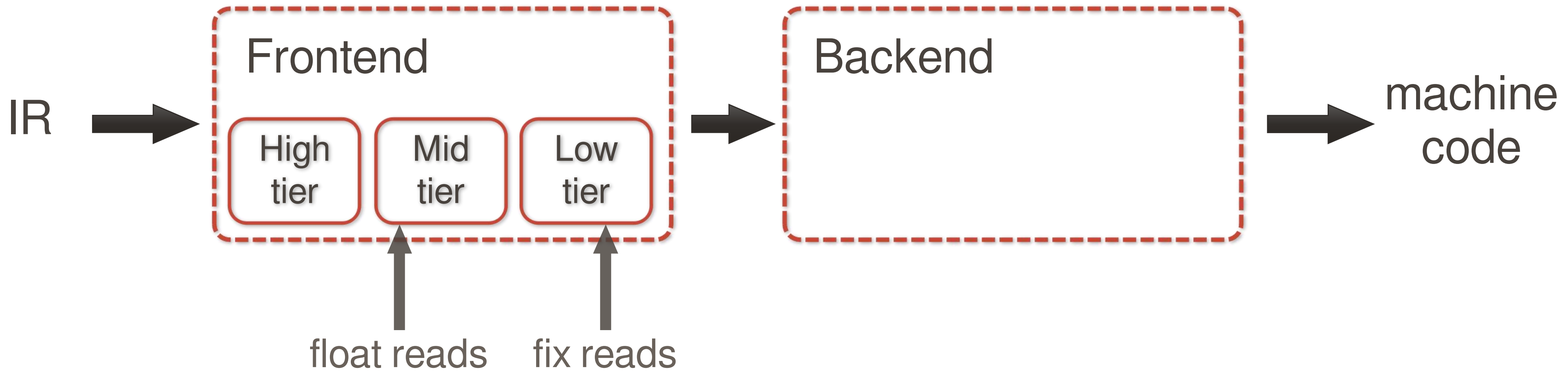
Memory out-of-bounds access profiling

The compiler decides not to float the reads out of the loop - why?


308 LoopBegin
1882 C(UnsafeWasmMemory@1baa2687) a -
6863 AMD64Address
6858 Read#UnsafeWasmMemory.pageSize
1042 C(16) i32
1895 <<
1897 <
6621 C(2064) i32
6624 +
1894 If
1898 Begin
1909 Invoke!#Direct#UnsafeWasmMemory.trapOutOfBounds
1899 Begin
1912 KillingBegin
1921 End
1922 End
1919 Merge
6859 Read#UnsafeWasmMemory.startAddress
6865 AMD64Address
6763 Read#OFF_HEAP_LOCATION

After low tier

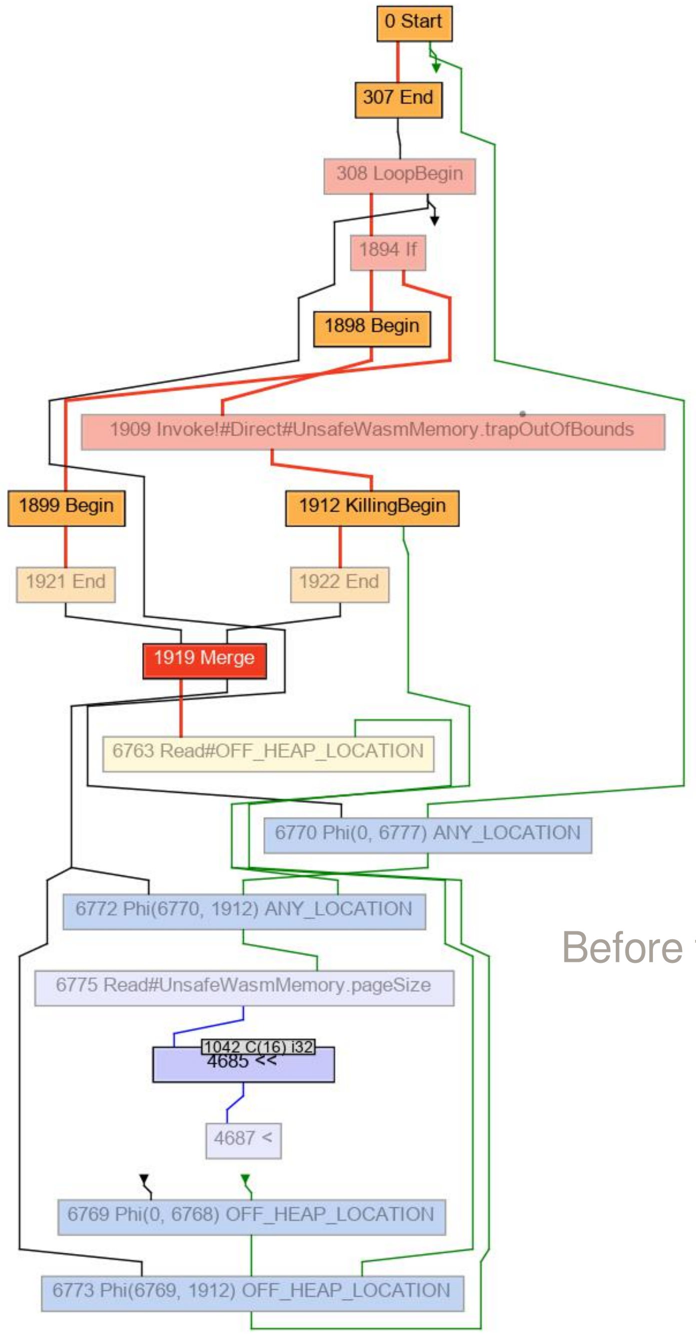
Memory out-of-bounds access profiling



Memory out-of-bounds access profiling

The compiler decides not to float the reads out of the loop - why?

To understand why, we need to inspect the memory graph that gets produced once the reads get floated.


0 Start
307 End
308 LoopBegin
1894 If
1898 Begin
1909 Invoke!#Direct#UnsafeWasmMemory.trapOutOfBounds
1899 Begin
1912 KillingBegin
1921 End
1922 End
1919 Merge
6763 Read#OFF_HEAP_LOCATION
6770 Phi(0, 6777) ANY_LOCATION
6772 Phi(6770, 1912) ANY_LOCATION
6775 Read#UnsafeWasmMemory.pageSize
1042 C(16) i32
4685 <<
4687 <
6769 Phi(0, 6768) OFF_HEAP_LOCATION
6773 Phi(6769, 1912) OFF_HEAP_LOCATION

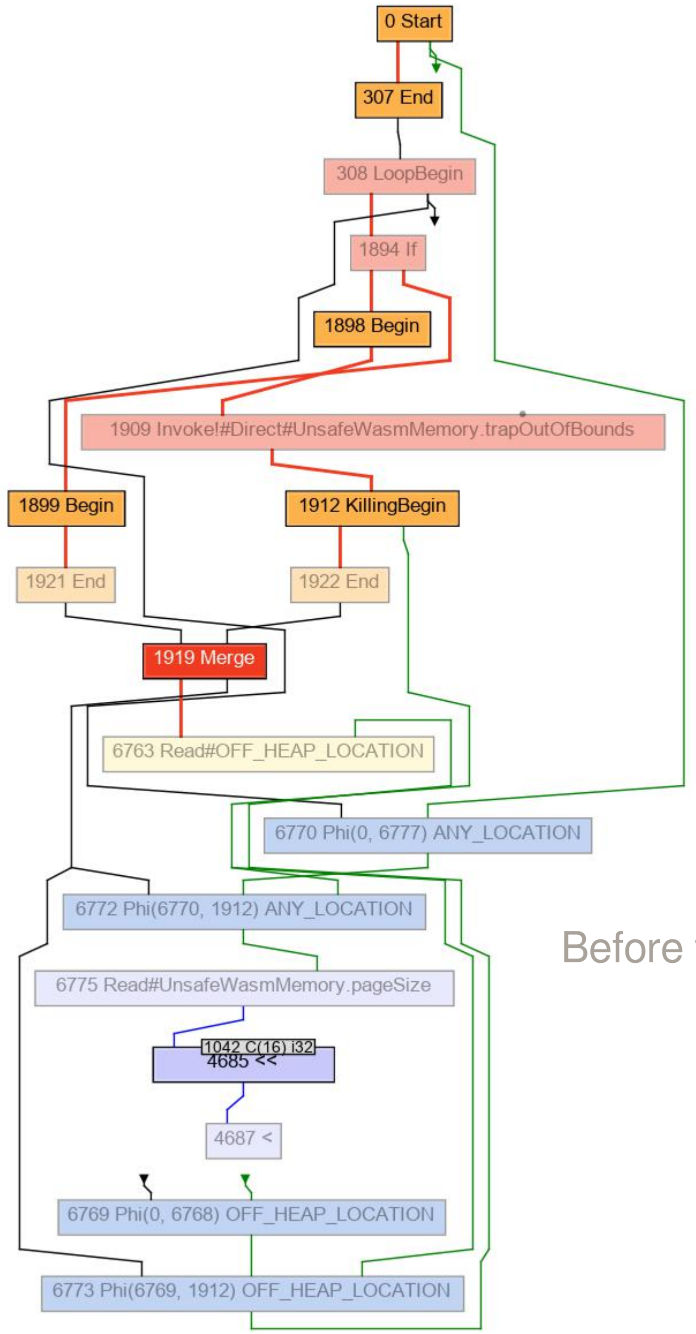
Before fix-reads

Memory out-of-bounds access profiling

The compiler decides not to float the reads out of the loop - why?

To understand why, we need to inspect the memory graph that gets produced once the reads get floated.

Heavily simplified model - a read cannot be scheduled before its (either value or memory) inputs.

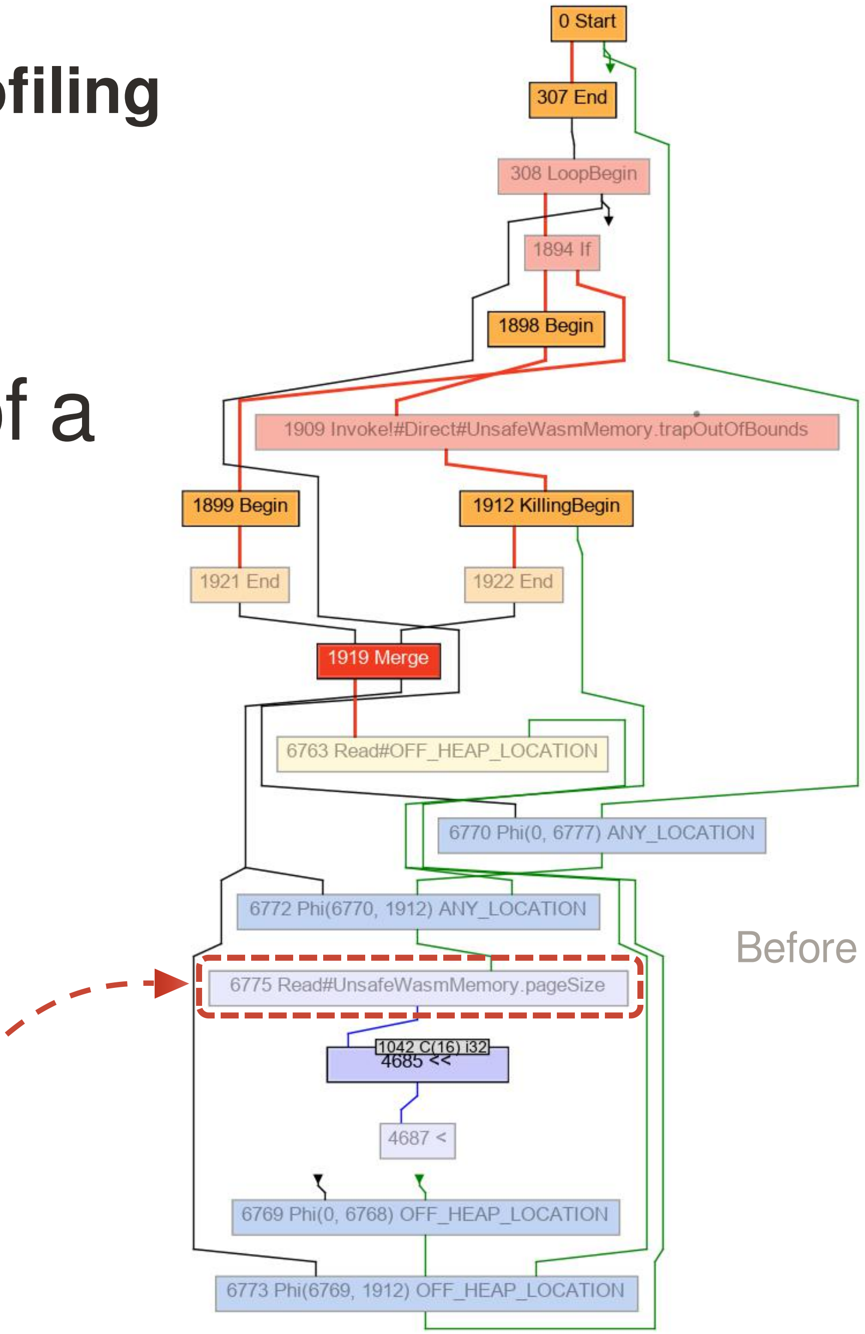

0 Start
307 End
308 LoopBegin
1894 If
1898 Begin
1909 Invoke!#Direct#UnsafeWasmMemory.trapOutOfBounds
1899 Begin
1912 KillingBegin
1921 End
1922 End
1919 Merge
6763 Read#OFF_HEAP_LOCATION
6770 Phi(0, 6777) ANY_LOCATION
6772 Phi(6770, 1912) ANY_LOCATION
6775 Read#UnsafeWasmMemory.pageSize
1042 C(16) i32
4685 <<
4687 <
6769 Phi(0, 6768) OFF_HEAP_LOCATION
6773 Phi(6769, 1912) OFF_HEAP_LOCATION

Before fix-reads

Memory out-of-bounds access profiling

The memory graph consists of a set of ordered effects

We find a read, and follow its inputs.

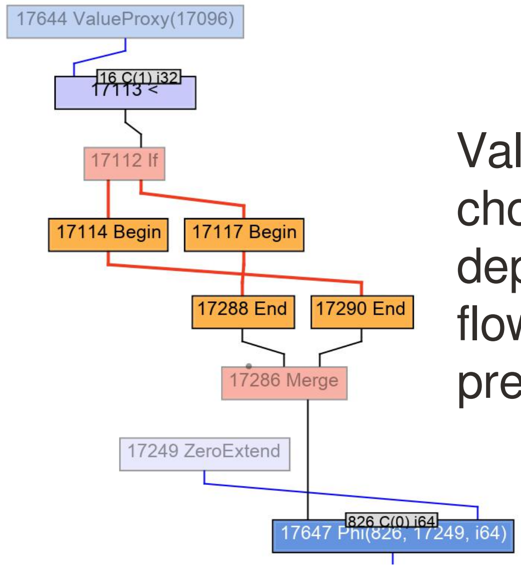

0 Start
307 End
308 LoopBegin
1894 If
1898 Begin
1909 Invoke!#Direct#UnsafeWasmMemory.trapOutOfBounds
1899 Begin
1912 KillingBegin
1921 End
1922 End
1919 Merge
6763 Read#OFF_HEAP_LOCATION
6770 Phi(0, 6777) ANY_LOCATION
6772 Phi(6770, 1912) ANY_LOCATION
6775 Read#UnsafeWasmMemory.pageSize
1042 C(16) i32
4685 <<
4687 <
6769 Phi(0, 6768) OFF_HEAP_LOCATION
6773 Phi(6769, 1912) OFF_HEAP_LOCATION

Before fix-reads

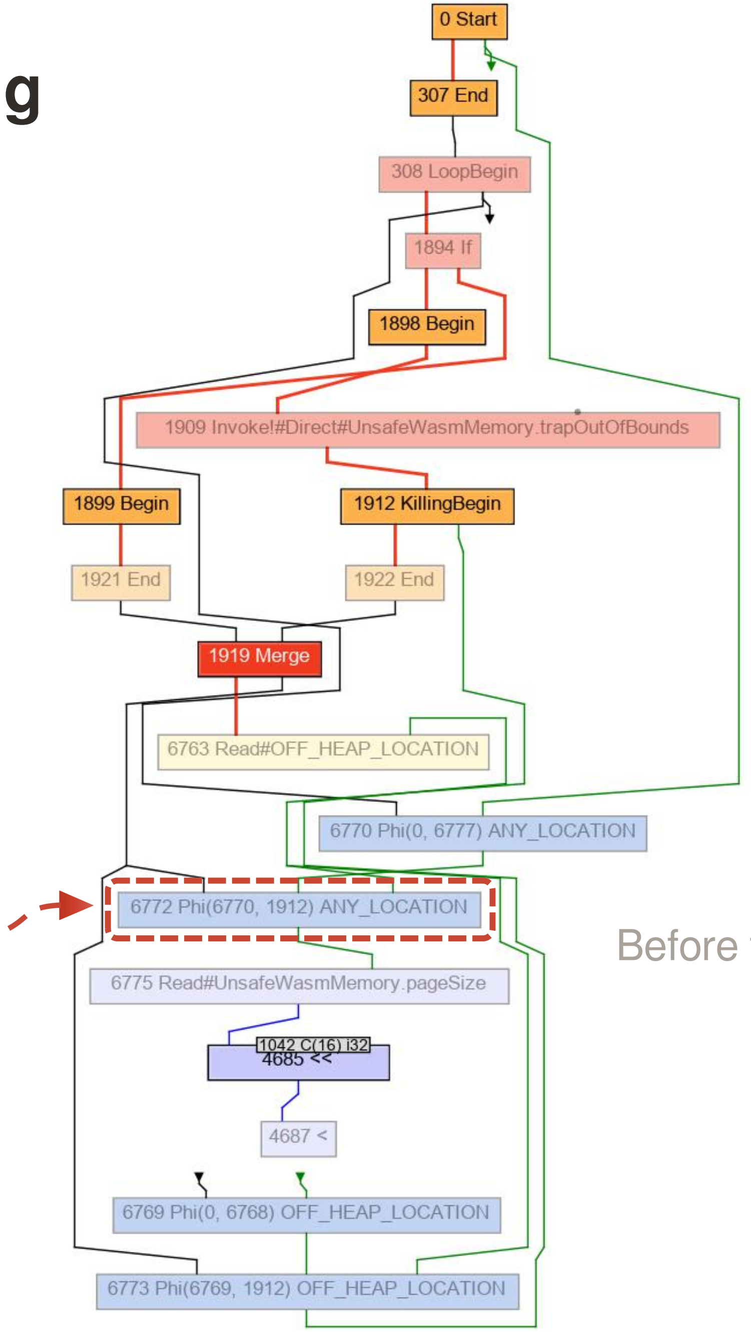
Memory out-of-bounds access profiling

The memory graph consists of a set of ordered effects

We find a read, and follow its inputs.



Value-phi nodes represent a choice between two values, depending on which control flow the program has previously traversed.


0 Start
307 End
308 LoopBegin
1894 If
1898 Begin
1909 Invoke!#Direct#UnsafeWasmMemory.trapOutOfBounds
1899 Begin
1912 KillingBegin
1921 End
1922 End
1919 Merge
6763 Read#OFF_HEAP_LOCATION
6770 Phi(0, 6777) ANY_LOCATION
6772 Phi(6770, 1912) ANY_LOCATION
6775 Read#UnsafeWasmMemory.pageSize
1042 C(16) i32
4685 <<
4687 <
6769 Phi(0, 6768) OFF_HEAP_LOCATION
6773 Phi(6769, 1912) OFF_HEAP_LOCATION

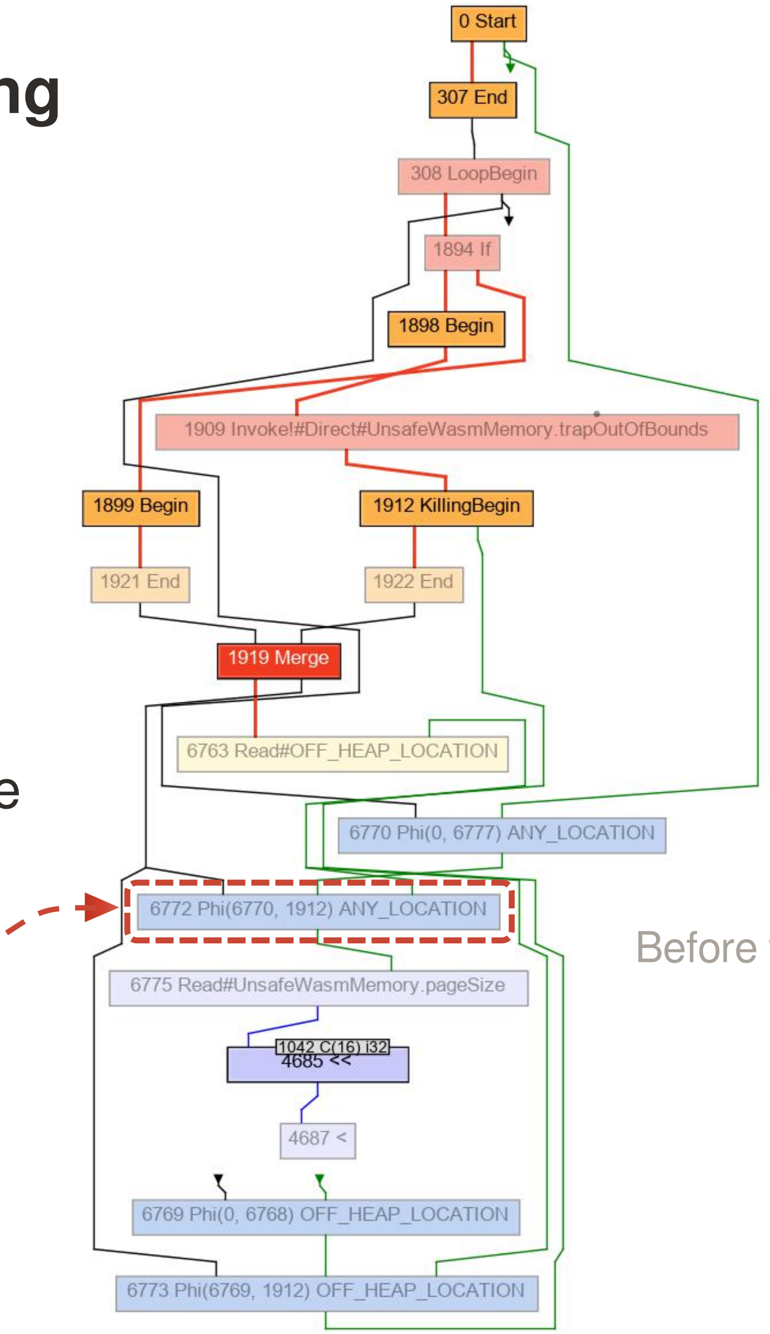
Before fix-reads

Memory out-of-bounds access profiling

The memory graph consists of a set of ordered effects

We find a read, and follow its inputs.

Memory-phi nodes (green edges) represent a choice between two *memory states*, depending on which control flow the program has previously traversed.


0 Start
307 End
308 LoopBegin
1894 If
1898 Begin
1909 Invoke!#Direct#UnsafeWasmMemory.trapOutOfBounds
1899 Begin
1912 KillingBegin
1921 End
1922 End
1919 Merge
6763 Read#OFF_HEAP_LOCATION
6770 Phi(0, 6777) ANY_LOCATION
6772 Phi(6770, 1912) ANY_LOCATION
6775 Read#UnsafeWasmMemory.pageSize
1042 C(16) i32
4685 <<
4687 <
6769 Phi(0, 6768) OFF_HEAP_LOCATION
6773 Phi(6769, 1912) OFF_HEAP_LOCATION

Before fix-reads

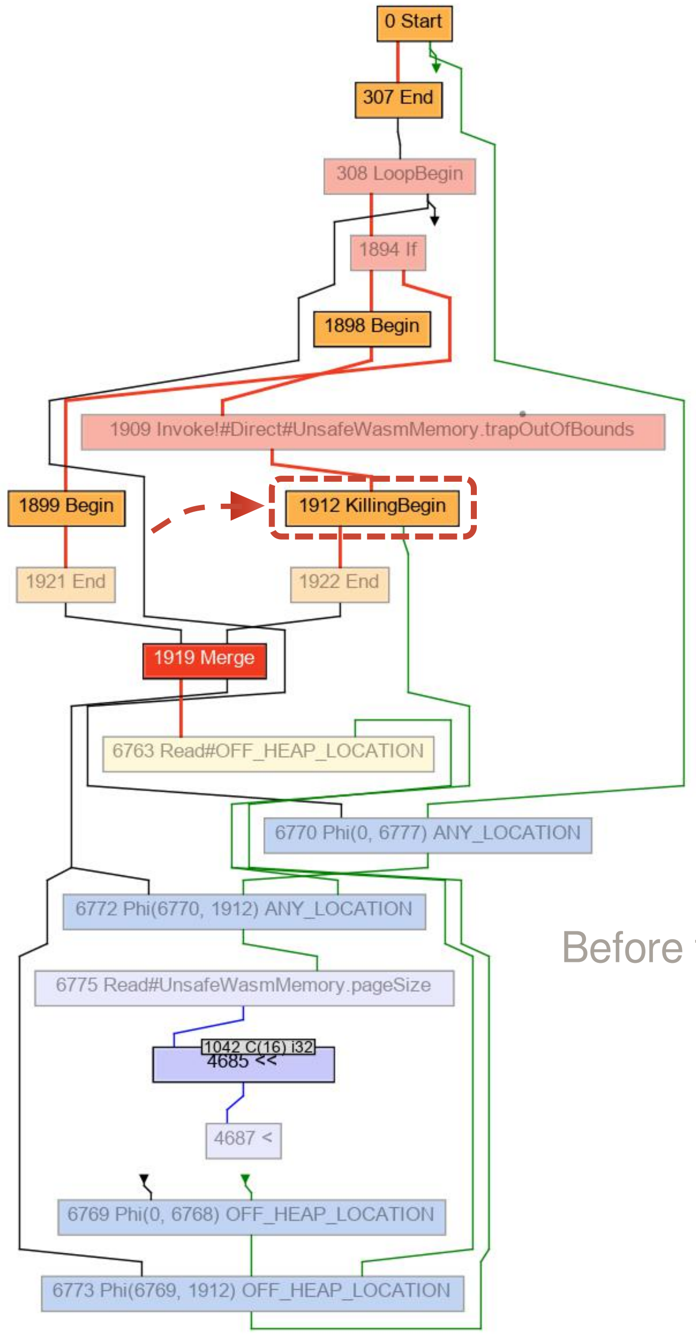
Memory out-of-bounds access profiling

The memory graph consists of a set of ordered effects

We find a read, and follow its inputs.

Memory-phi nodes (green edges) represent a choice between two *memory states*, depending on which control flow the program has previously traversed.

We reach a KillingBegin node - beginning of a basic block that could follow a write to a memory location.



0 Start
307 End
308 LoopBegin
1894 If
1898 Begin
1909 Invoke!#Direct#UnsafeWasmMemory.trapOutOfBounds
1899 Begin
1912 KillingBegin
1921 End
1922 End
1919 Merge
6763 Read#OFF_HEAP_LOCATION
6770 Phi(0, 6777) ANY_LOCATION
6772 Phi(6770, 1912) ANY_LOCATION
6775 Read#UnsafeWasmMemory.pageSize
1042 C(16) i32
4685 <<
4687 <
6769 Phi(0, 6768) OFF_HEAP_LOCATION
6773 Phi(6769, 1912) OFF_HEAP_LOCATION

Before fix-reads

Memory out-of-bounds access profiling

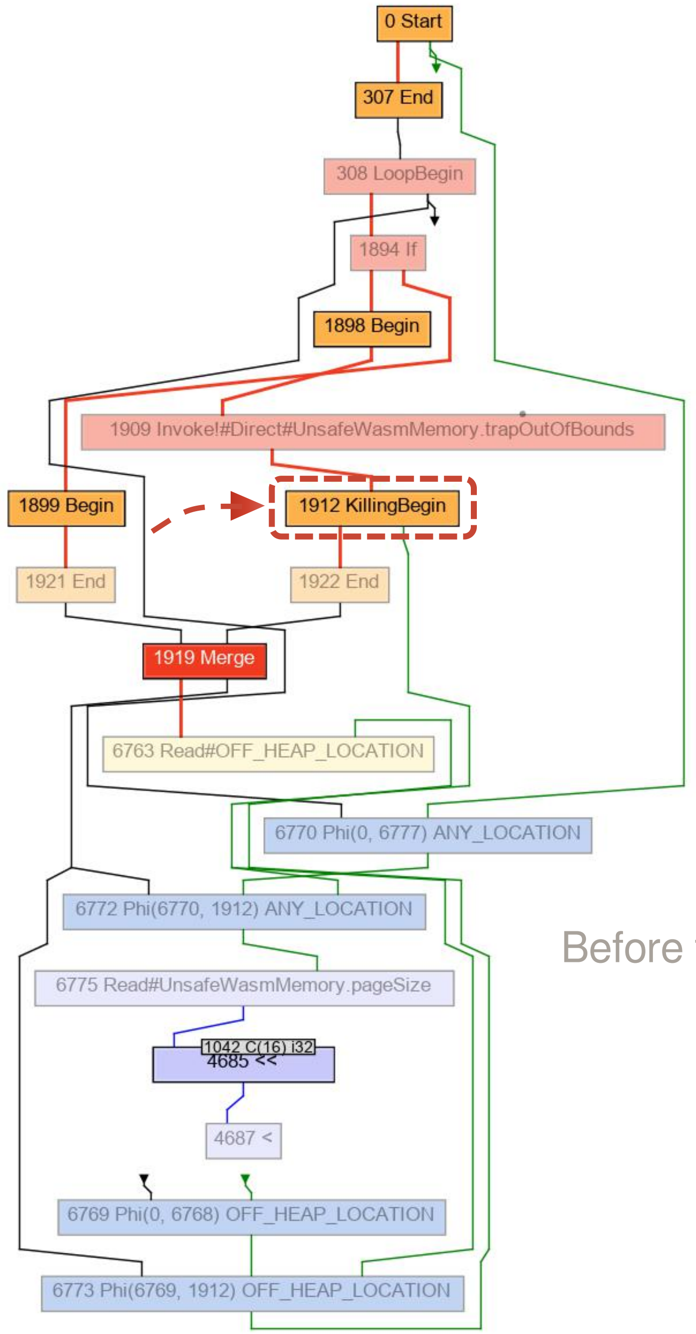
The memory graph consists of a set of ordered effects

We find a read, and follow its inputs.

Memory-phi nodes (green edges) represent a choice between two *memory states*, depending on which control flow the program has previously traversed.

We reach a KillingBegin node - beginning of a basic block that could follow a write to a memory location.

Unsurprising: preceding Invoke may write to memory.

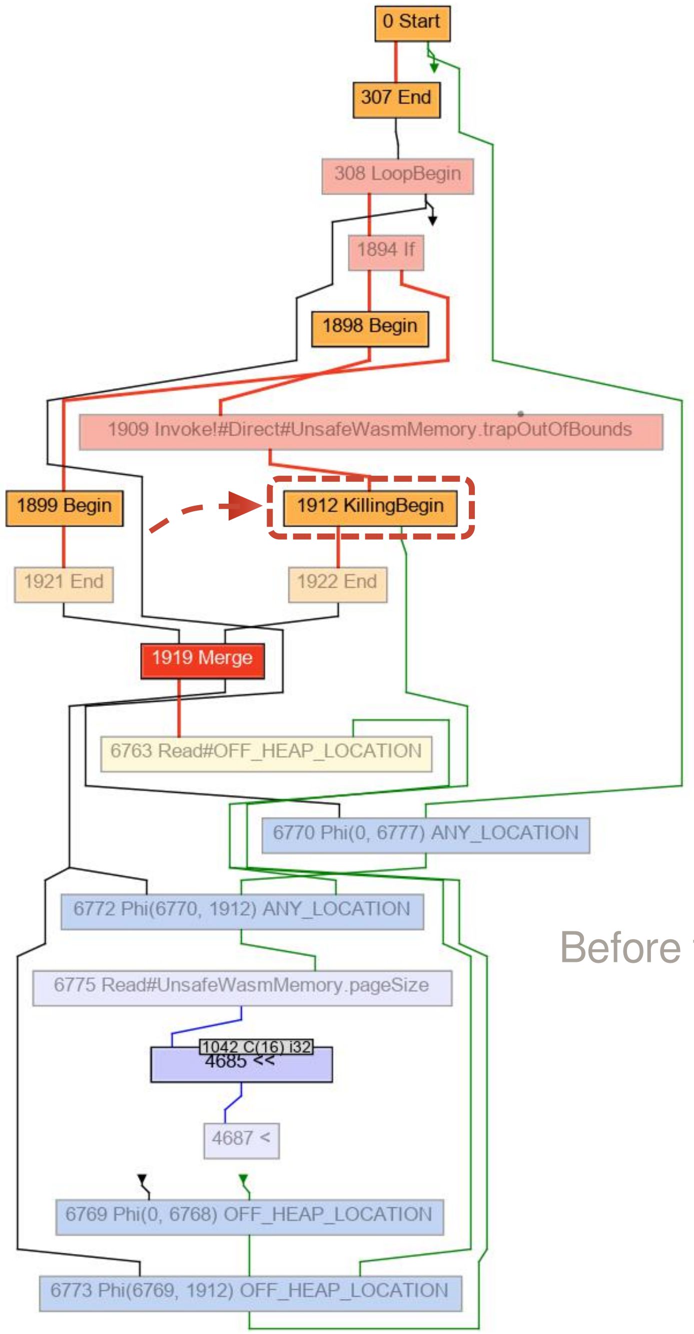

0 Start
307 End
308 LoopBegin
1894 If
1898 Begin
1909 Invoke!#Direct#UnsafeWasmMemory.trapOutOfBounds
1899 Begin
1912 KillingBegin
1921 End
1922 End
1919 Merge
6763 Read#OFF_HEAP_LOCATION
6770 Phi(0, 6777) ANY_LOCATION
6772 Phi(6770, 1912) ANY_LOCATION
6775 Read#UnsafeWasmMemory.pageSize
1042 C(16) i32
4685 <<
4687 <
6769 Phi(0, 6768) OFF_HEAP_LOCATION
6773 Phi(6769, 1912) OFF_HEAP_LOCATION

Before fix-reads

Memory out-of-bounds access profiling

Conclusion: we cannot schedule the read before a call to a method that may have written to memory

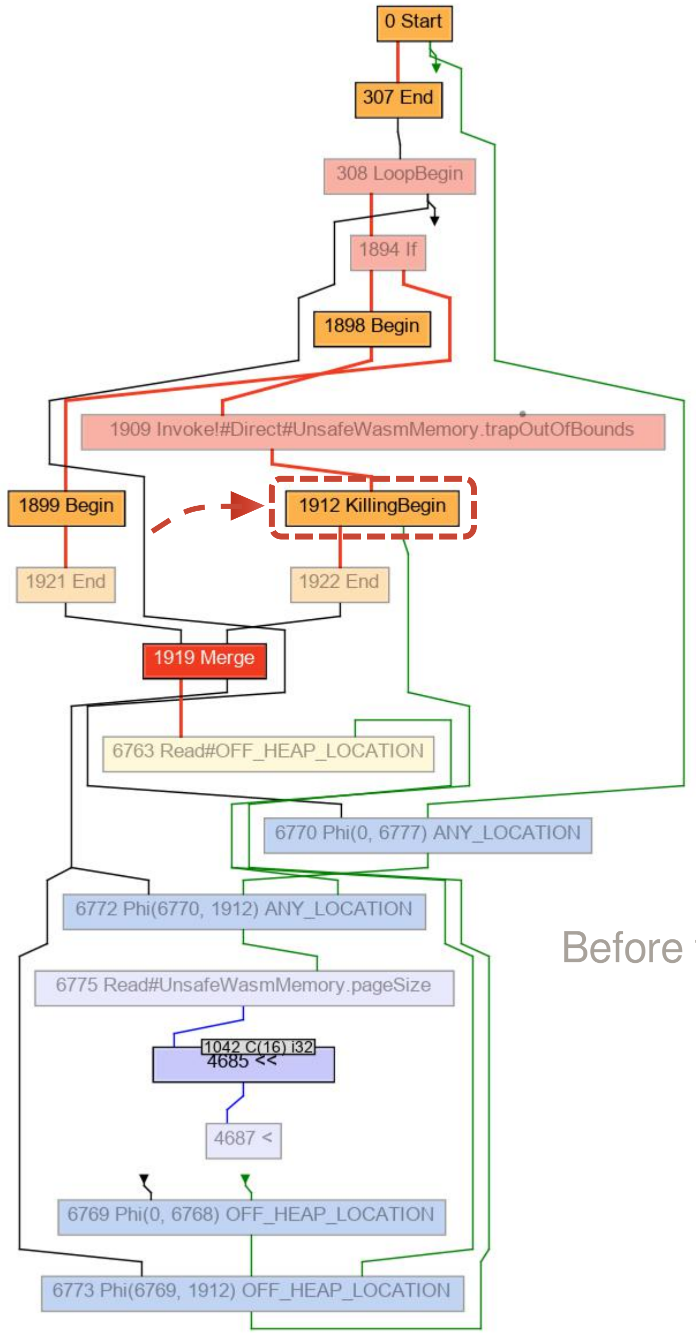
If we did schedule the read above the Invoke, then we would read a stale value.


0 Start
307 End
308 LoopBegin
1894 If
1898 Begin
1909 Invoke!#Direct#UnsafeWasmMemory.trapOutOfBounds
1899 Begin
1912 KillingBegin
1921 End
1922 End
1919 Merge
6763 Read#OFF_HEAP_LOCATION
6770 Phi(0, 6777) ANY_LOCATION
6772 Phi(6770, 1912) ANY_LOCATION
6775 Read#UnsafeWasmMemory.pageSize
1042 C(16) i32
4685 <<
4687 <
6769 Phi(0, 6768) OFF_HEAP_LOCATION
6773 Phi(6769, 1912) OFF_HEAP_LOCATION

Before fix-reads

Memory out-of-bounds access profiling

Could the compiler inline the call to trapOutOfBounds?



0 Start
307 End
308 LoopBegin
1894 If
1898 Begin
1909 Invoke!#Direct#UnsafeWasmMemory.trapOutOfBounds
1899 Begin
1912 KillingBegin
1921 End
1922 End
1919 Merge
6763 Read#OFF_HEAP_LOCATION
6770 Phi(0, 6777) ANY_LOCATION
6772 Phi(6770, 1912) ANY_LOCATION
6775 Read#UnsafeWasmMemory.pageSize
1042 C(16) i32
4685 <<
4687 <
6769 Phi(0, 6768) OFF_HEAP_LOCATION
6773 Phi(6769, 1912) OFF_HEAP_LOCATION

Before fix-reads

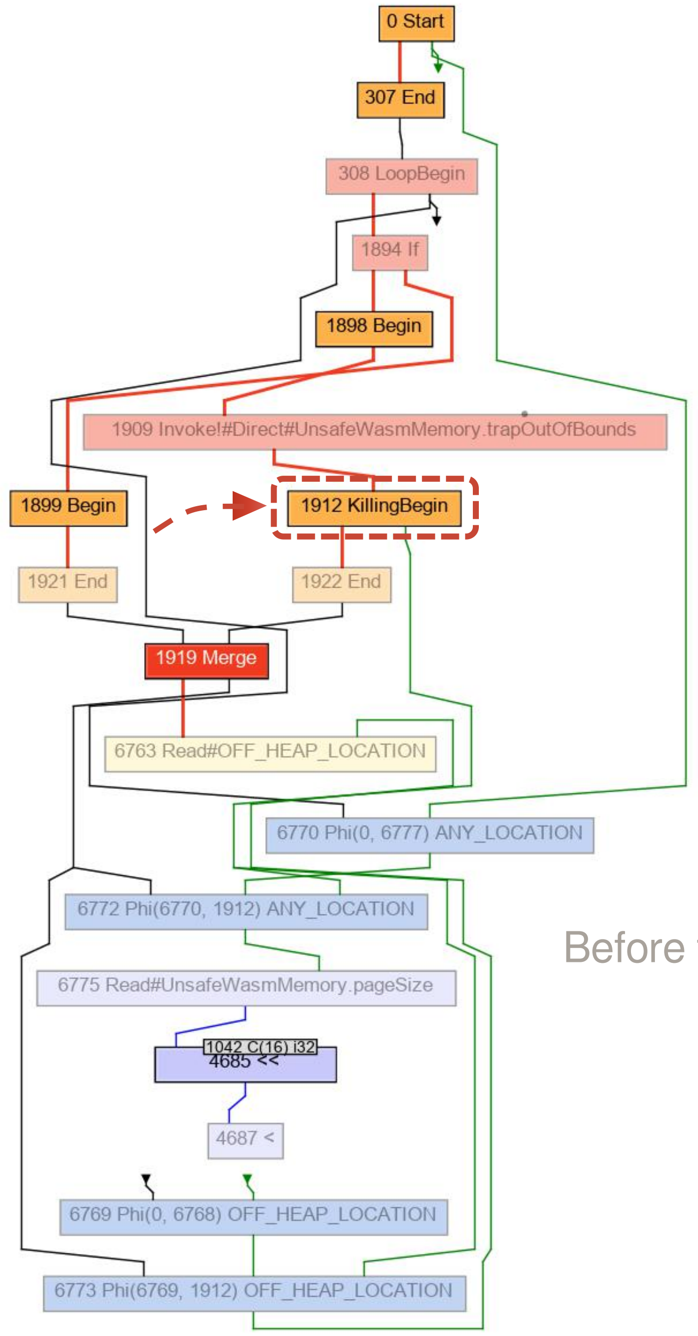
Memory out-of-bounds access profiling

Could the compiler inline the call to trapOutOfBounds?

No, because this call is a Truffle boundary, which was added to prevent the exception creation code from getting partially evaluated.

```
@TruffleBoundary
void trapOutOfBounds(long address, long offset) {
  throw new WasmTrap(
    ”Out-of-bounds: “ + (address + offset));
}
```

GraalWasm



0 Start
307 End
308 LoopBegin
1894 If
1898 Begin
1909 Invoke!#Direct#UnsafeWasmMemory.trapOutOfBounds
1899 Begin
1912 KillingBegin
1921 End
1922 End
1919 Merge
6763 Read#OFF_HEAP_LOCATION
6770 Phi(0, 6777) ANY_LOCATION
6772 Phi(6770, 1912) ANY_LOCATION
6775 Read#UnsafeWasmMemory.pageSize
1042 C(16) i32
4685 <<
4687 <
6769 Phi(0, 6768) OFF_HEAP_LOCATION
6773 Phi(6769, 1912) OFF_HEAP_LOCATION

Before fix-reads

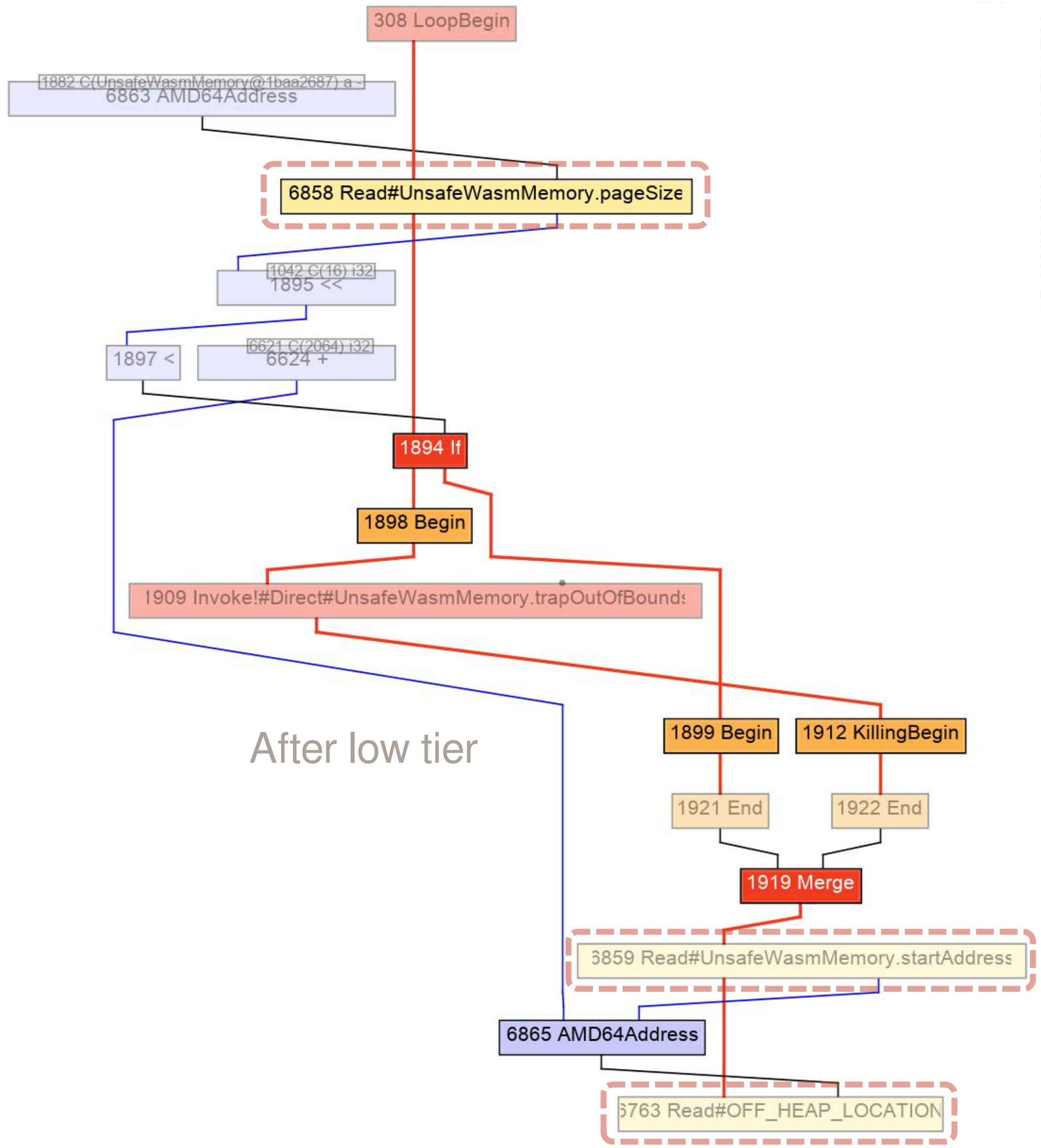
Memory out-of-bounds access profiling

We have 3 memory-read nodes for every memory read in the program

Normally, the language should annotate these values as `@CompilationFinal` values, so that partial evaluation can inline them into the code.

```
class WasmMemory {
  @CompilationFinal long startAddress;
  @CompilationFinal long pageSize;
  ...
```

GraalWasm



308 LoopBegin
1882 C(UnsafeWasmMemory@1baa2687) a
6863 AMD64Address
6858 Read#UnsafeWasmMemory.pageSize
1042 C(16) i32
1895 <<
1897 <
6621 C(2064) i32
6624 +
1894 If
1898 Begin
1909 Invoke!#Direct#UnsafeWasmMemory.trapOutOfBounds
After low tier
1899 Begin
1912 KillingBegin
1921 End
1922 End
1919 Merge
6859 Read#UnsafeWasmMemory.startAddress
6865 AMD64Address
6763 Read#OFF_HEAP_LOCATION

Memory out-of-bounds access profiling

In WebAssembly, memory can grow

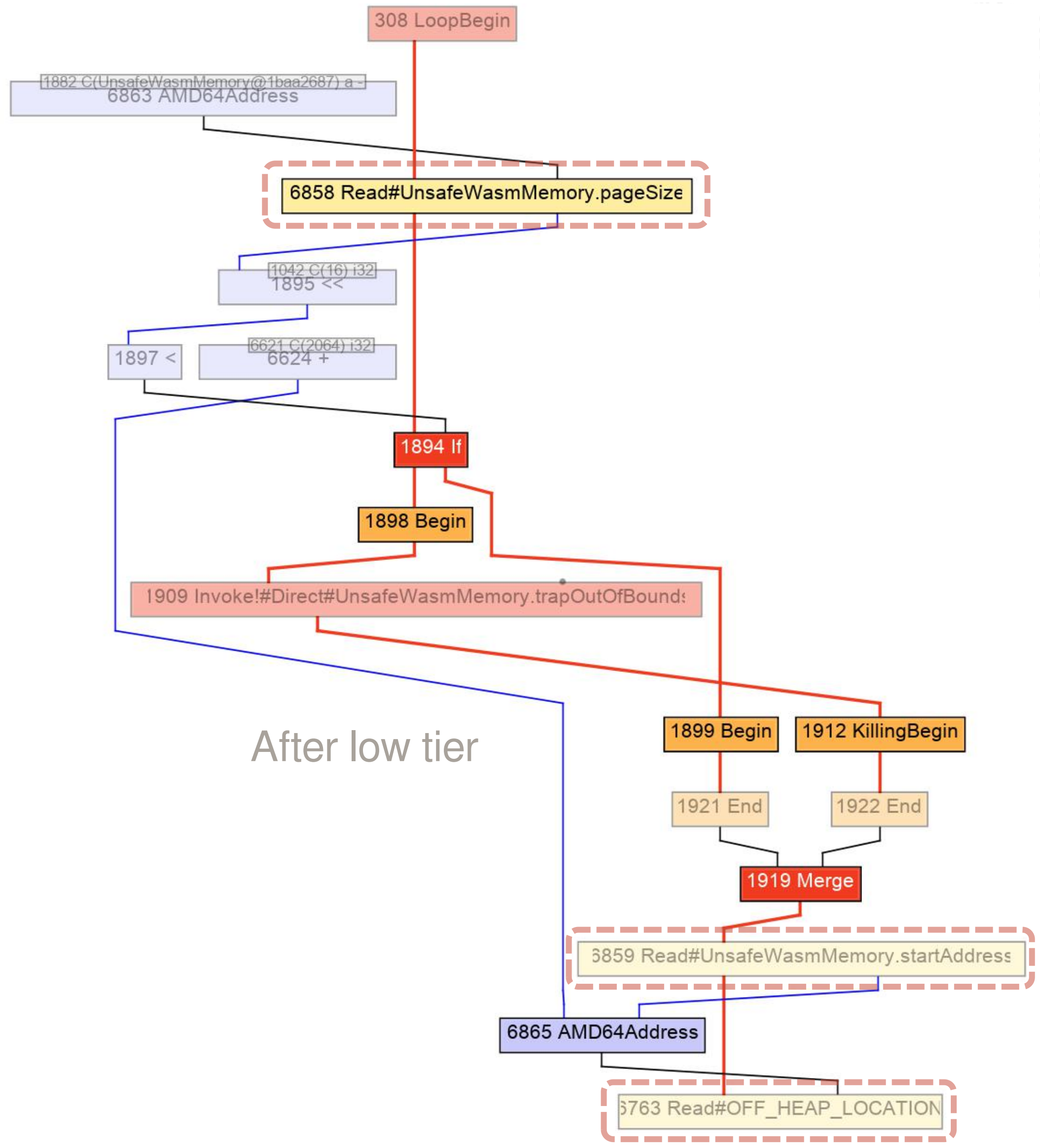
There is an instruction that can attempt to resize the memory of the program.
The `startAddress` and `pageSize` cannot be constant!

```
class WasmMemory {
  long startAddress;
  long pageSize;
  ...
```

GraalWasm

```
...
i32.load
i32.store
memory.grow
...
```

WebAssembly



308 LoopBegin
1882 C(UnsafeWasmMemory@1baa2687) a -
6863 AMD64Address
6858 Read#UnsafeWasmMemory.pageSize
1042 C(16) i32
1895 <<
1897 <
6621 C(2064) i32
6624 +
1894 If
1898 Begin
1909 Invoke!#Direct#UnsafeWasmMemory.trapOutOfBounds
1899 Begin
1912 KillingBegin
1921 End
1922 End
1919 Merge
6859 Read#UnsafeWasmMemory.startAddress
6865 AMD64Address
6763 Read#OFF_HEAP_LOCATION

After low tier

Memory out-of-bounds access profiling

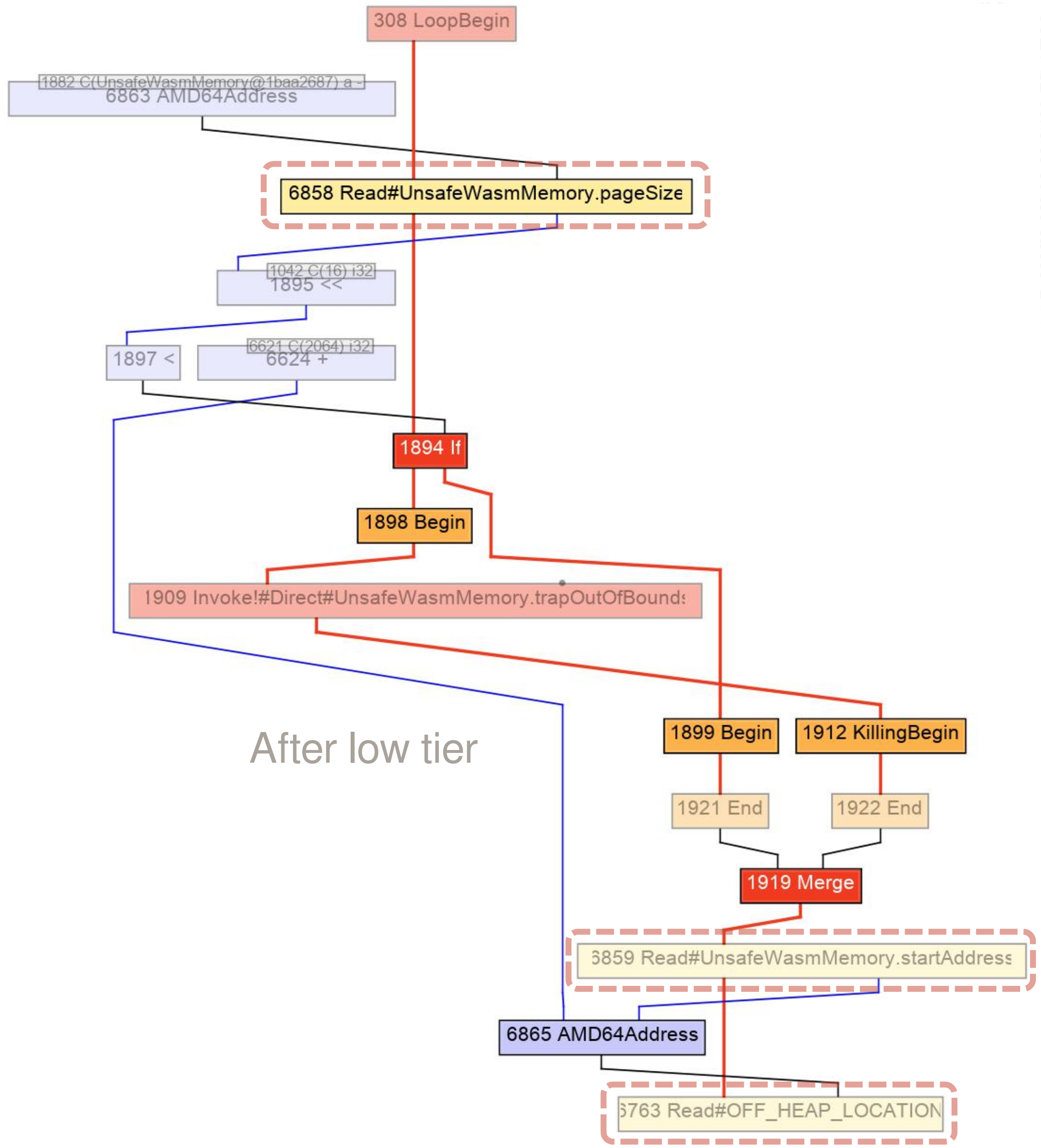
So, is there anything we can do?

```
class WasmMemory {
  long startAddress;
  long pageSize;
  ...
```

GraalWasm

...
i32.load
i32.store
memory.grow
...

WebAssembly


308 LoopBegin
1882 C(UnsafeWasmMemory@1baa2687) a -
6863 AMD64Address
6858 Read#UnsafeWasmMemory.pageSize
1042 C(16) i32
1895 <<
1897 <
6621 C(2064) i32
6624 +
1894 If
1898 Begin
1909 Invoke!#Direct#UnsafeWasmMemory.trapOutOfBounds
After low tier
1899 Begin
1912 KillingBegin
1921 End
1922 End
1919 Merge
6859 Read#UnsafeWasmMemory.startAddress
6865 AMD64Address
6763 Read#OFF_HEAP_LOCATION

Memory out-of-bounds access profiling

So, is there anything we can do?

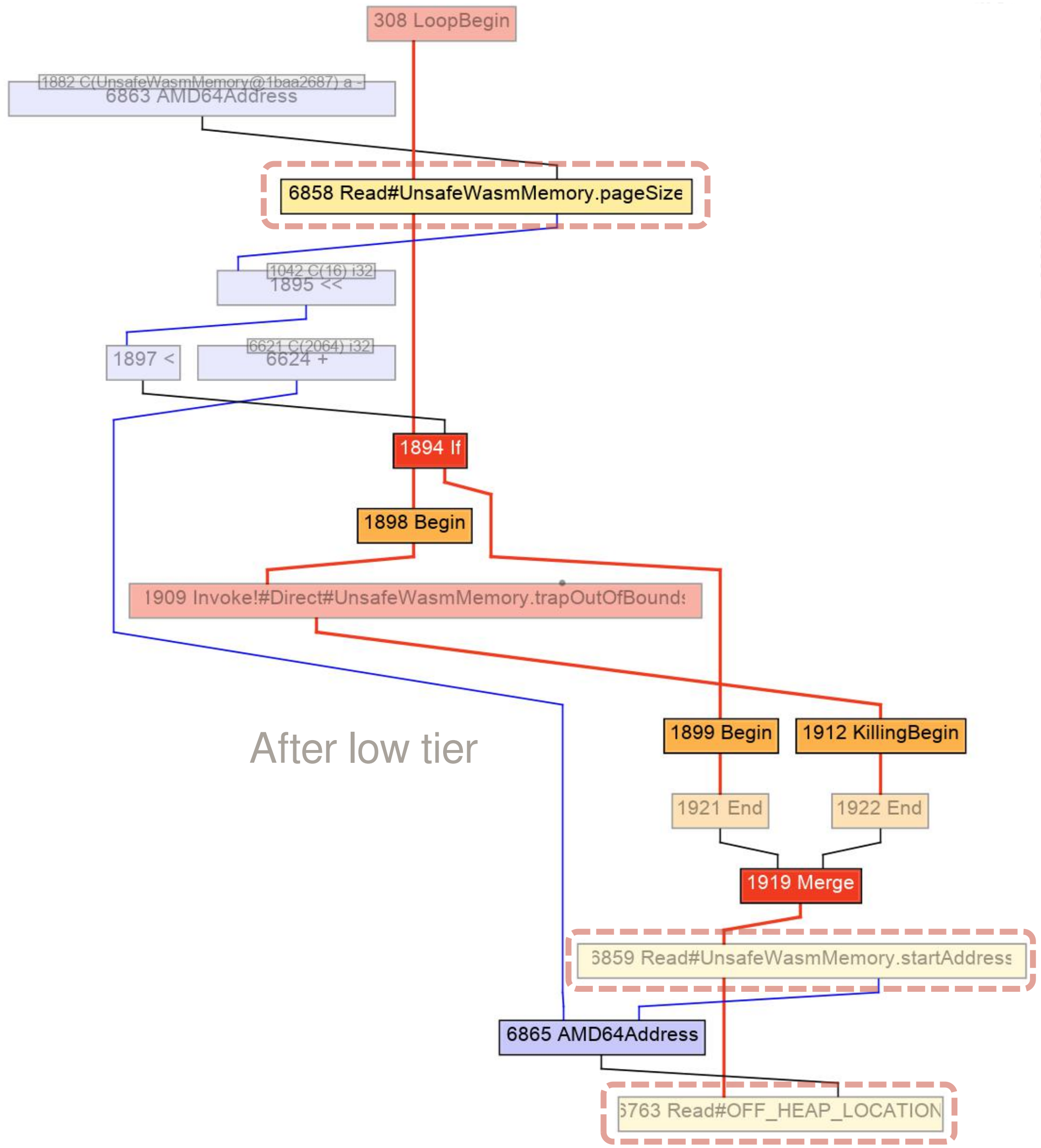
When in doubt, speculate.
How often does an out-of-bounds access error happen?

```
class WasmMemory {
  long startAddress;
  long pageSize;
  ...
```

GraalWasm

...
i32.load
i32.store
memory.grow
...

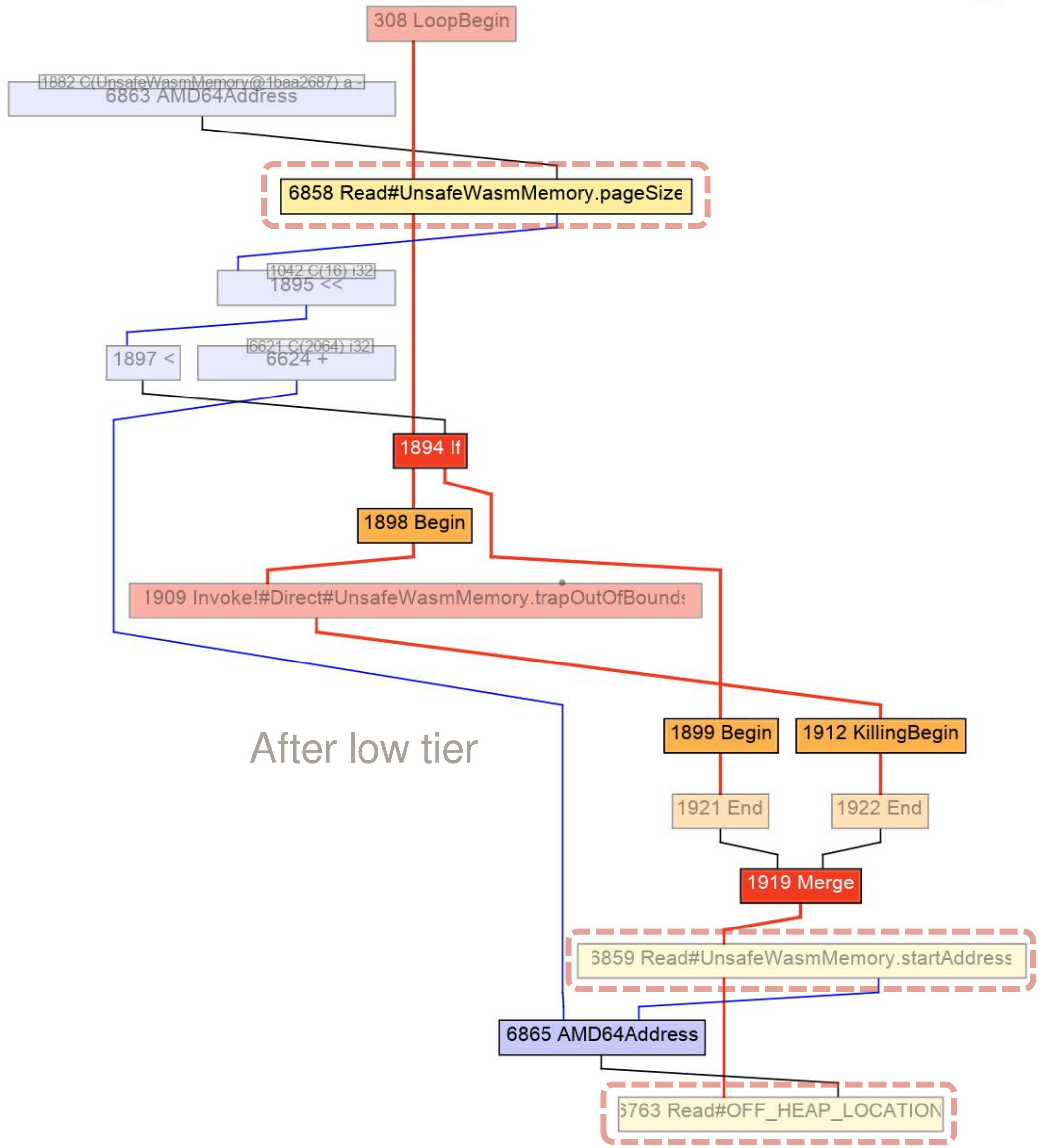
WebAssembly


308 LoopBegin
1882 C(UnsafeWasmMemory@1baa2687) a
6863 AMD64Address
6858 Read#UnsafeWasmMemory.pageSize
1042 C(16) i32
1895 <<
1897 <
6621 C(2064) i32
6624 +
1894 If
1898 Begin
1909 Invoke!#Direct#UnsafeWasmMemory.trapOutOfBounds
1899 Begin
1912 KillingBegin
1921 End
1922 End
1919 Merge
6859 Read#UnsafeWasmMemory.startAddress
6865 AMD64Address
6763 Read#OFF_HEAP_LOCATION

After low tier

Memory out-of-bounds access profiling

Out-of-bounds memory access must terminate the program


308 LoopBegin
1882 C(UnsafeWasmMemory@1baa2687) a -
6863 AMD64Address
6858 Read#UnsafeWasmMemory.pageSize
1042 C(16) i32
1895 <<
1897 <
6621 C(2064) i32
6624 +
1894 If
1898 Begin
1909 Invoke!#Direct#UnsafeWasmMemory.trapOutOfBounds
1899 Begin
1912 KillingBegin
1921 End
1922 End
1919 Merge
6859 Read#UnsafeWasmMemory.startAddress
6865 AMD64Address
6763 Read#OFF_HEAP_LOCATION

After low tier

```
int load_i32(long address) {
  if (address + 4 > pageCount * PAGE_SIZE) {
    trapOutOfBounds(address, 4);
  }
  return unsafe.getInt(startAddress + address);
}
```

GraalWasm

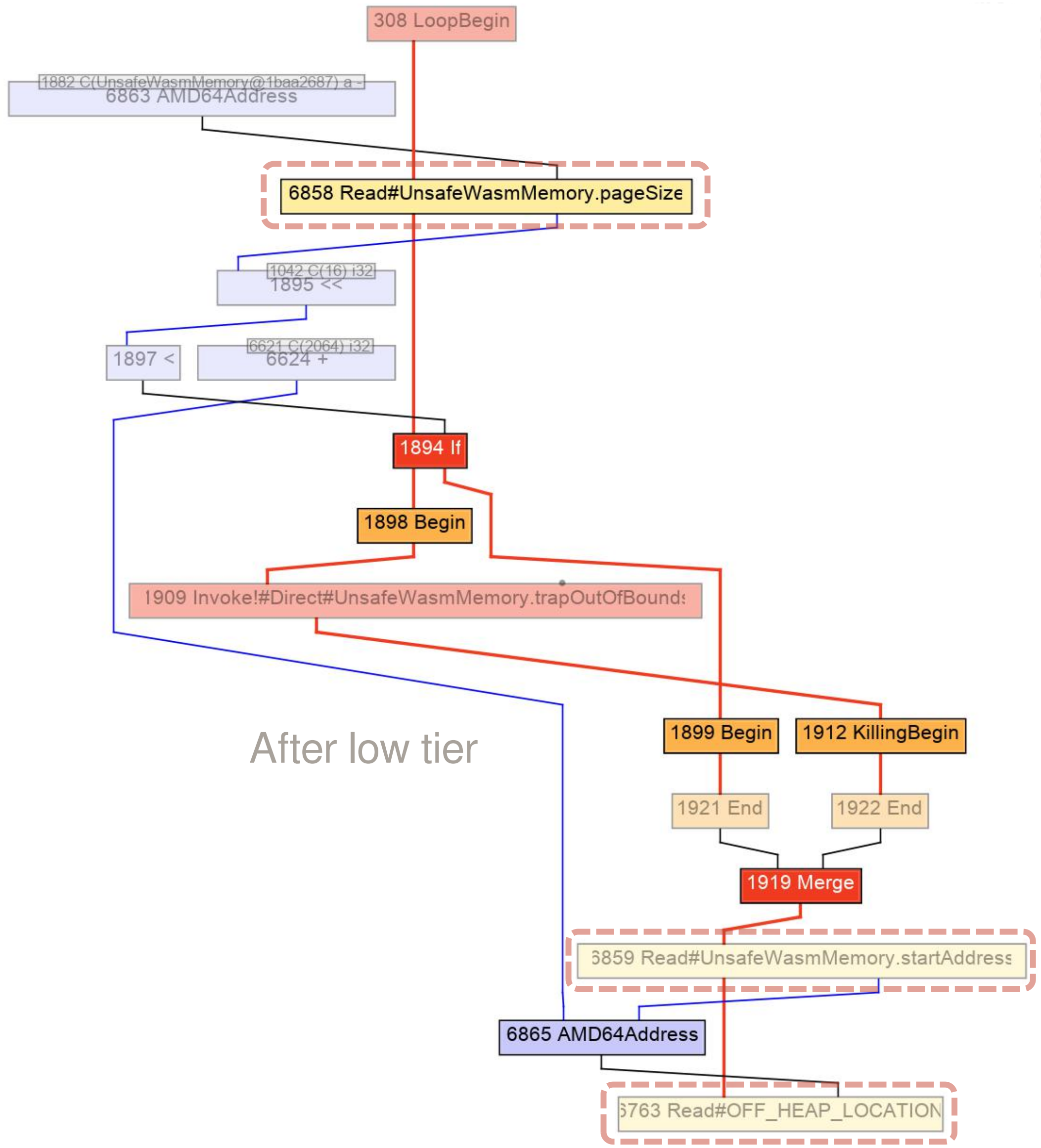
Memory out-of-bounds access profiling

Out-of-bounds memory access must terminate the program

Therefore, it will never happen while the program is running. We can therefore speculate that the branch that throws the out-of-bounds error will never happen.

```
int load_i32(long address) {
  if (address + 4 > pageCount * PAGE_SIZE) {
    trapOutOfBounds(address, 4);
  }
  return unsafe.getInt(startAddress + address);
}
```

GraalWasm


308 LoopBegin
1882 C(UnsafeWasmMemory@1baa2687) a
6863 AMD64Address
6858 Read#UnsafeWasmMemory.pageSize
1042 C(16) i32
1895 <<
1897 <
6621 C(2064) i32
6624 +
1894 If
1898 Begin
1909 Invoke!#Direct#UnsafeWasmMemory.trapOutOfBounds
1899 Begin
1912 KillingBegin
1921 End
1922 End
1919 Merge
6859 Read#UnsafeWasmMemory.startAddress
6865 AMD64Address
6763 Read#OFF_HEAP_LOCATION

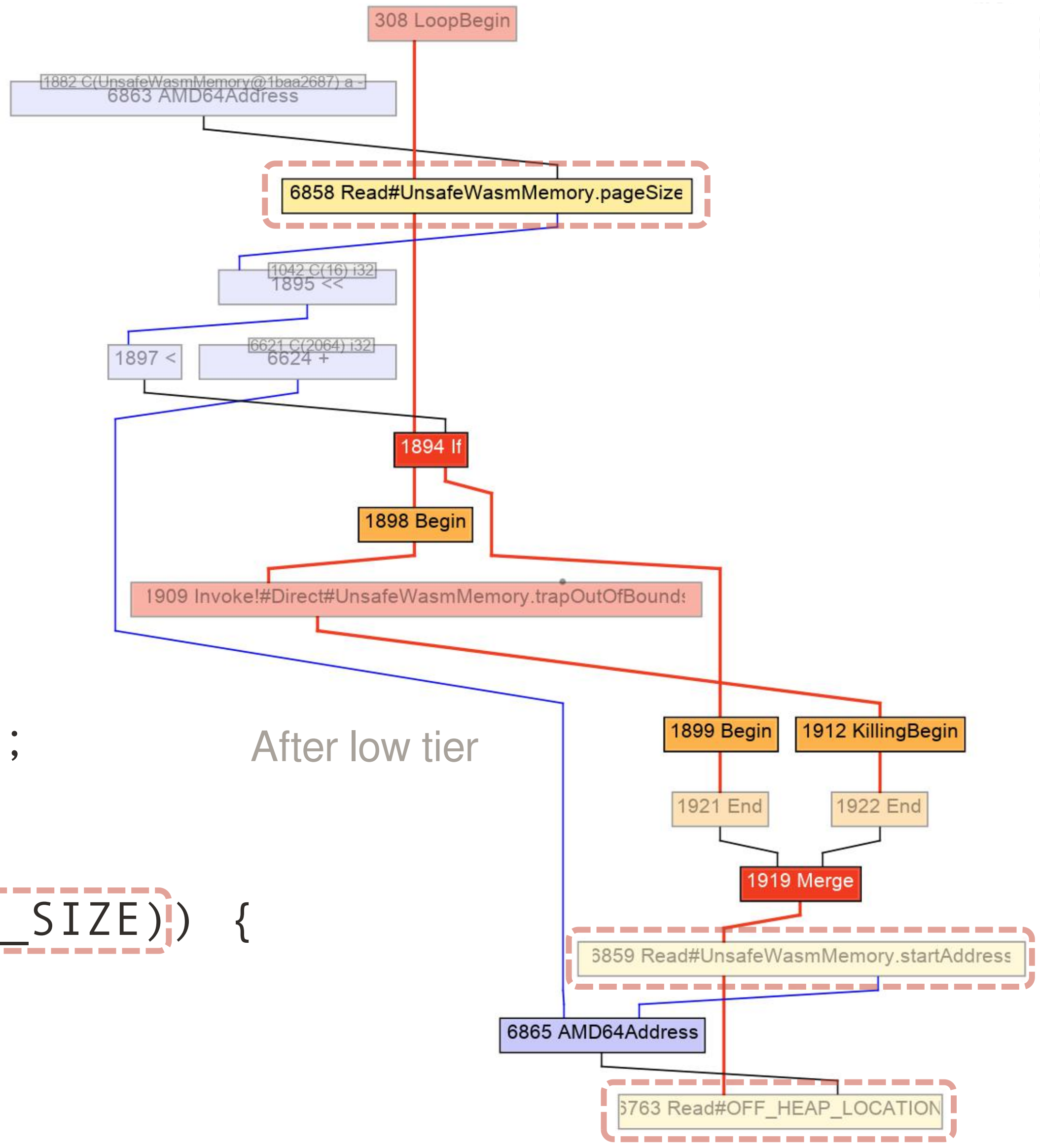
After low tier

Memory out-of-bounds access profiling

We speculate that the out-of-bounds branch with a ConditionProfile

```
ConditionProfile oob = ConditionProfile.create();

int load_i32(long address) {
  if (oob.profile(address + 4 > pageCount * PAGE_SIZE)) {
    trapOutOfBounds(address, 4);
  }
  return unsafe.getInt(startAddress + address);
}
```

GraalWasm


308 LoopBegin
1882 C(UnsafeWasmMemory@1baa2687) a -
6863 AMD64Address
6858 Read#UnsafeWasmMemory.pageSize
1042 C(16) i32
1895 <<
1897 <
6621 C(2064) i32
6624 +
1894 If
1898 Begin
1909 Invoke!#Direct#UnsafeWasmMemory.trapOutOfBounds
1899 Begin
1912 KillingBegin
1921 End
1922 End
1919 Merge
6859 Read#UnsafeWasmMemory.startAddress
6865 AMD64Address
6763 Read#OFF_HEAP_LOCATION

After low tier

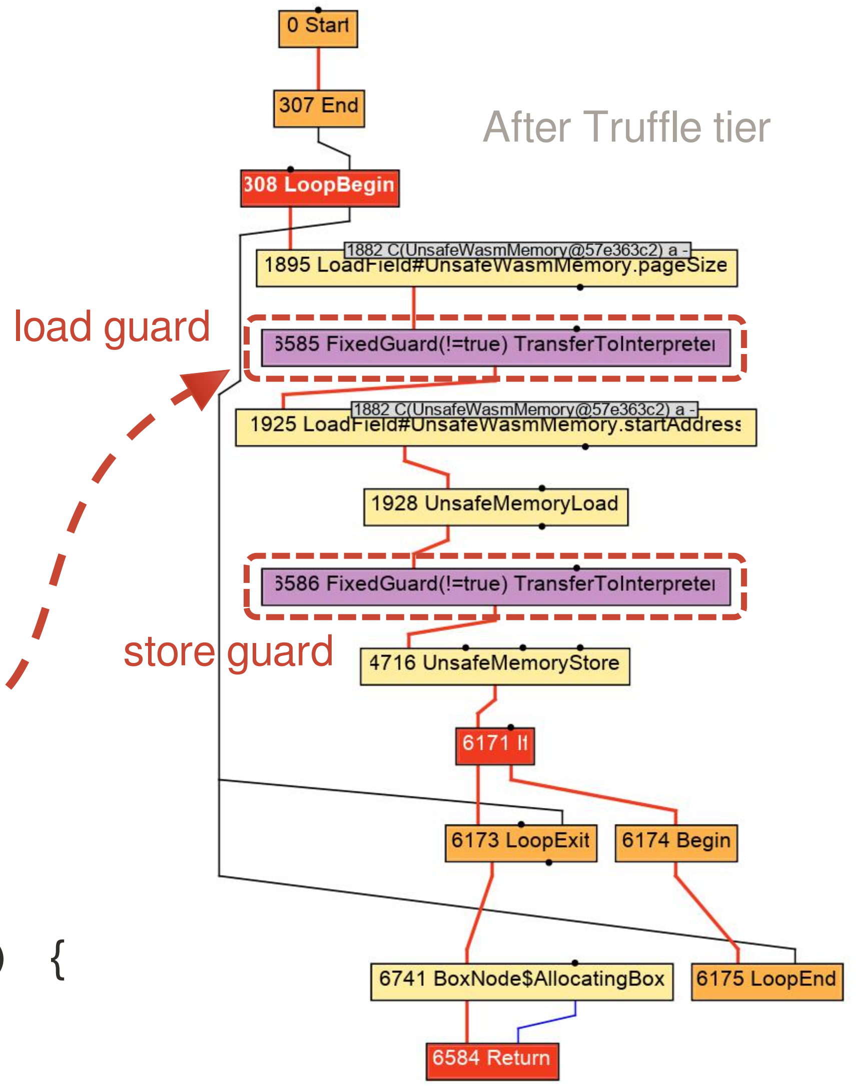
Memory out-of-bounds access profiling

A profile whose count is 0 will introduce Guard nodes into the IR

The compiler move the guard out of the loop when its condition does not depend on the values produced in that loop.

```
int load_i32(long address) {
  if (oob.profile(address + 4 > pageCount * PAGE_SIZE)) {
    trapOutOfBounds(address, 4);
  }
  return unsafe.getInt(startAddress + address);
}
```

GraalWasm


After Truffle tier
0 Start
307 End
308 LoopBegin
1882 C(UnsafeWasmMemory@57e363c2) a -
1895 LoadField#UnsafeWasmMemory.pageSize
load guard
3585 FixedGuard(!=true) TransferToInterpreter
1925 LoadField#UnsafeWasmMemory.startAddress
1928 UnsafeMemoryLoad
3586 FixedGuard(!=true) TransferToInterpreter
store guard
4716 UnsafeMemoryStore
6171 If
6173 LoopExit
6174 Begin
6741 BoxNode$AllocatingBox
6175 LoopEnd
6584 Return

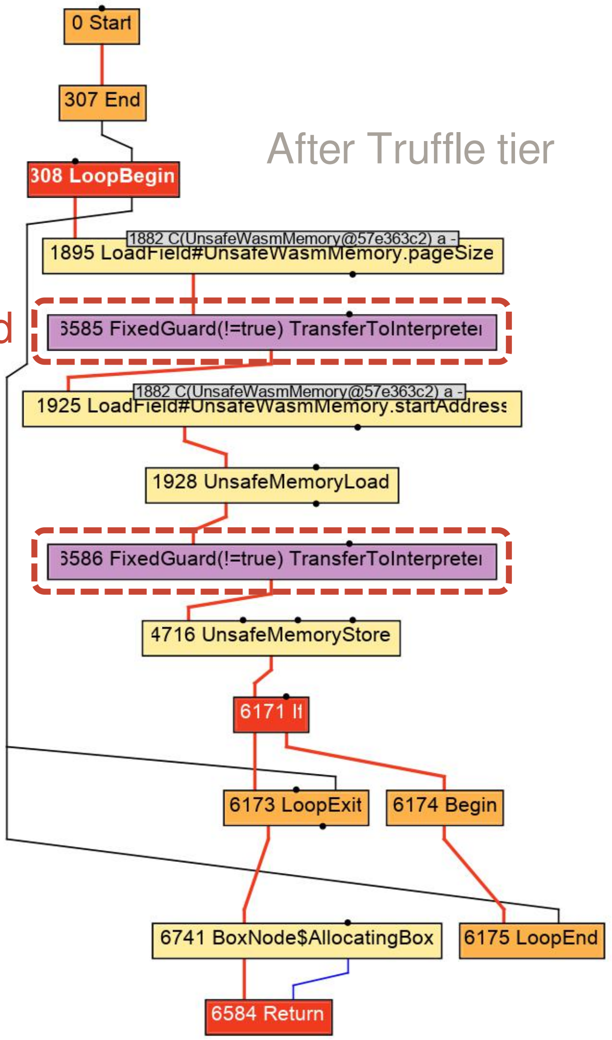
Memory out-of-bounds access profiling

Guards float out, anchored to an earlier block

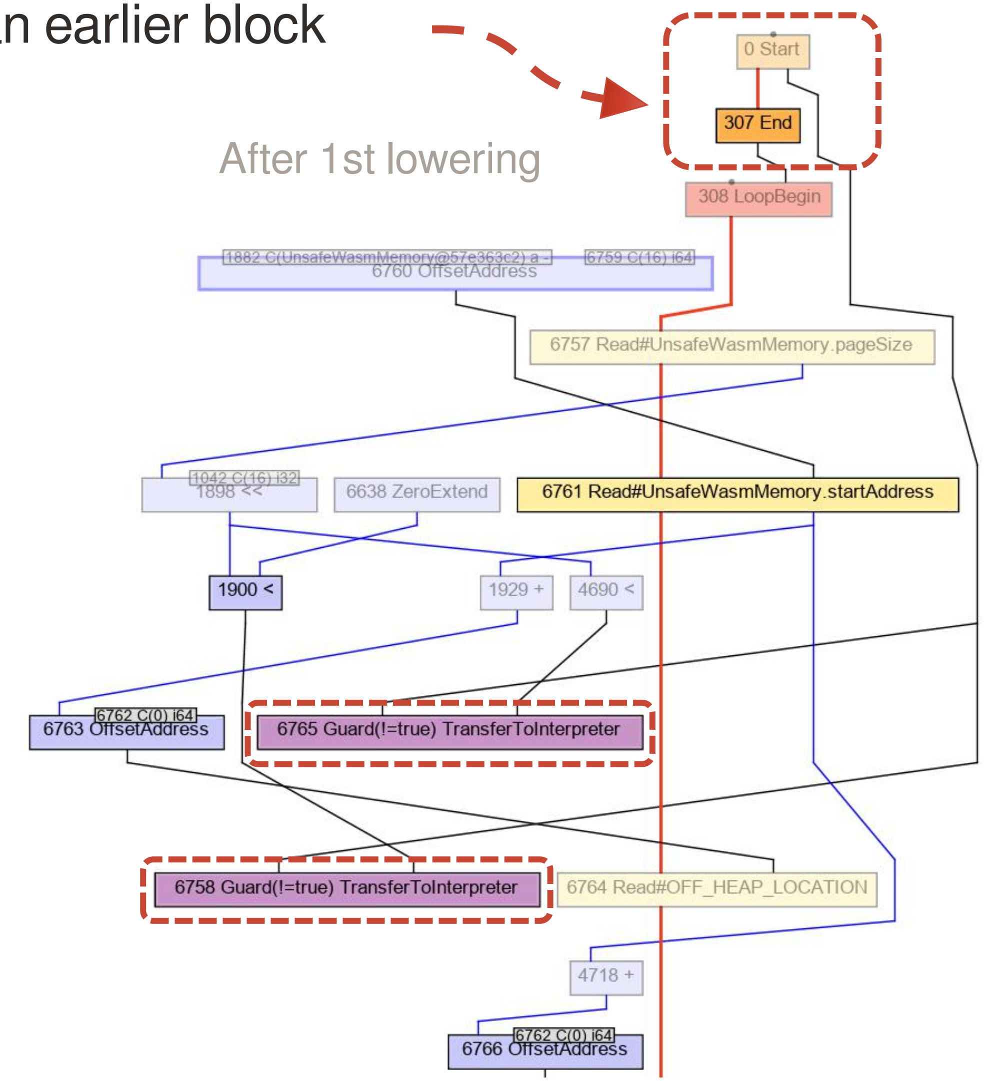
After Truffle tier

load guard

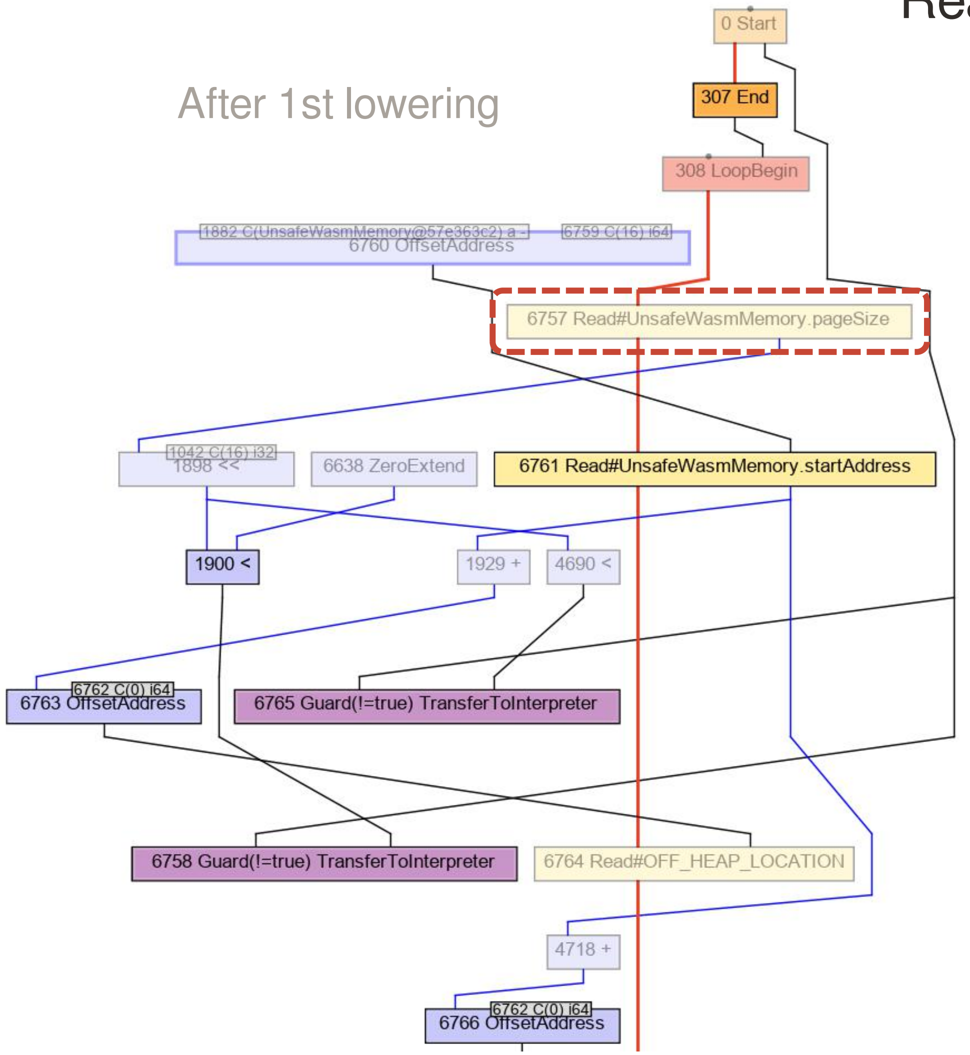
store guard


0 Start
307 End
308 LoopBegin
1882 C(UnsafeWasmMemory@57e363c2) a -
1895 LoadField#UnsafeWasmMemory.pageSize
6585 FixedGuard(!=true) TransferToInterpreter
1882 C(UnsafeWasmMemory@57e363c2) a -
1925 LoadField#UnsafeWasmMemory.startAddress
1928 UnsafeMemoryLoad
6586 FixedGuard(!=true) TransferToInterpreter
4716 UnsafeMemoryStore
6171 If
6173 LoopExit
6174 Begin
6741 BoxNode$AllocatingBox
6175 LoopEnd
6584 Return

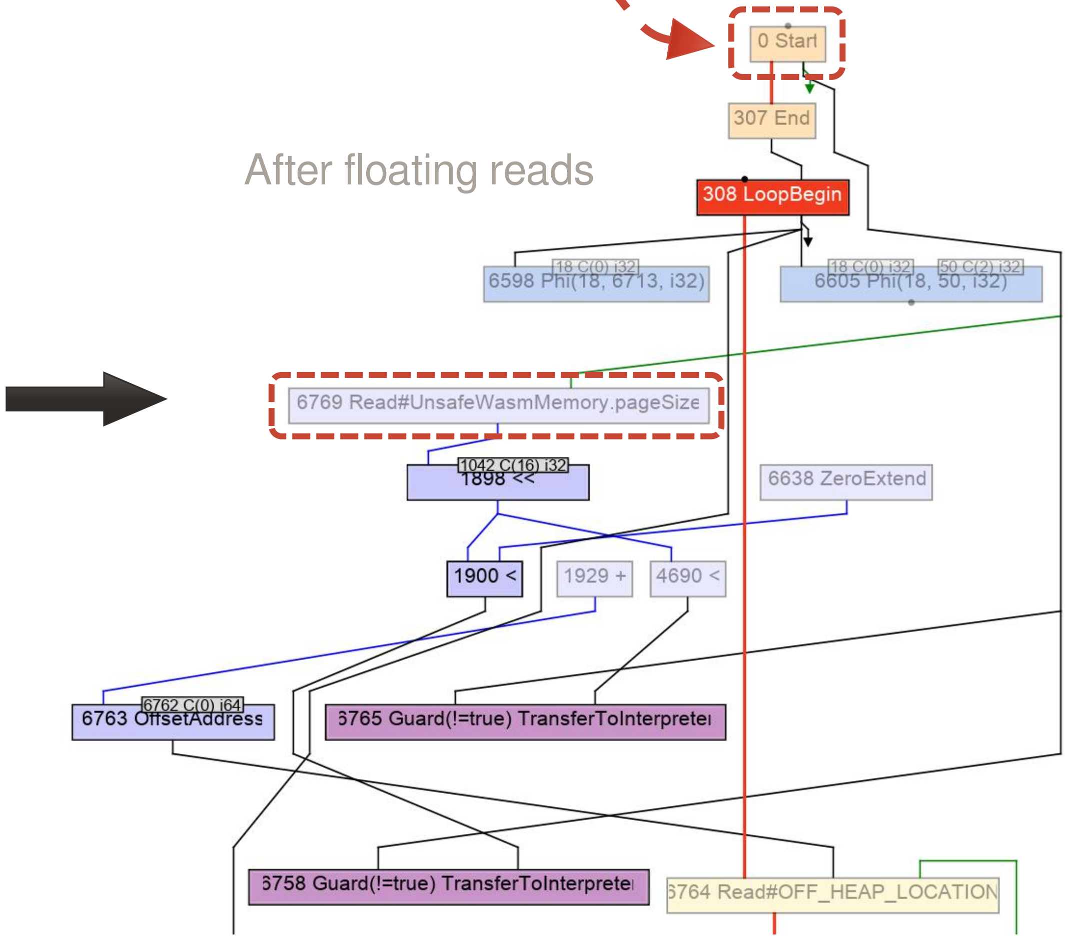
After 1st lowering


0 Start
307 End
308 LoopBegin
1882 C(UnsafeWasmMemory@57e363c2) a -
6759 C(16) i64
6760 OffsetAddress
6757 Read#UnsafeWasmMemory.pageSize
1042 C(16) i32
1898 <<
6638 ZeroExtend
6761 Read#UnsafeWasmMemory.startAddress
1900 <
1929 +
4690 <
6762 C(0) i64
6763 OffsetAddress
6765 Guard(!=true) TransferToInterpreter
6758 Guard(!=true) TransferToInterpreter
6764 Read#OFF_HEAP_LOCATION
4718 +
6762 C(0) i64
6766 OffsetAddress

Memory out-of-bounds access profiling


After 1st lowering
0 Start
307 End
308 LoopBegin
1882 C(UnsafeWasmMemory@57e363c2) a
6759 C(16) i64
6760 OffsetAddress
6757 Read#UnsafeWasmMemory.pageSize
1042 C(16) i32
1898 <<
6638 ZeroExtend
6761 Read#UnsafeWasmMemory.startAddress
1900 <
1929 +
4690 <
6762 C(0) i64
6763 OffsetAddress
6765 Guard(!=true) TransferToInterpreter
6758 Guard(!=true) TransferToInterpreter
6764 Read#OFF_HEAP_LOCATION
4718 +
6762 C(0) i64
6766 OffsetAddress

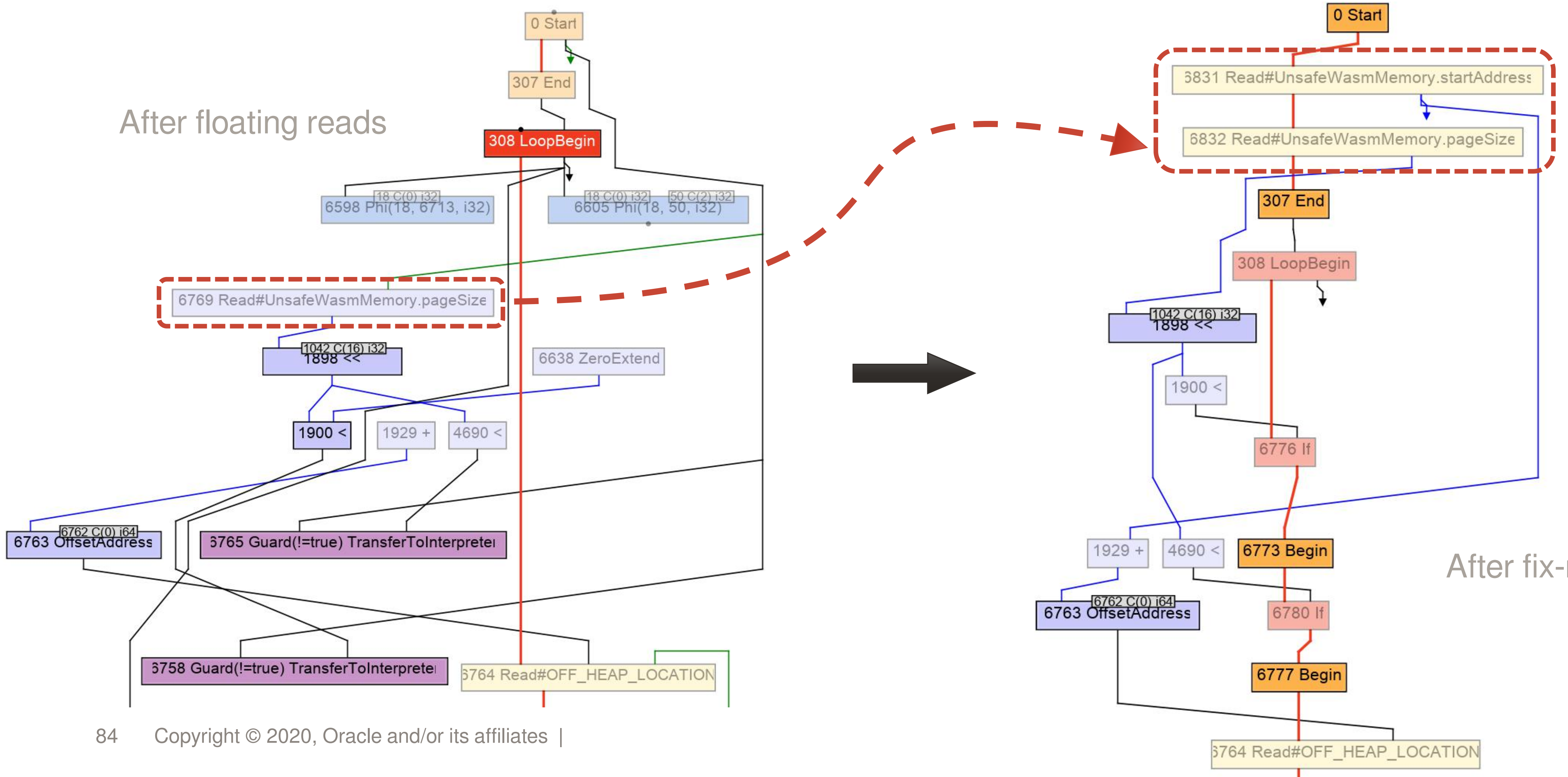
Reads float, and take the previous memory effect as input



Memory out-of-bounds access profiling

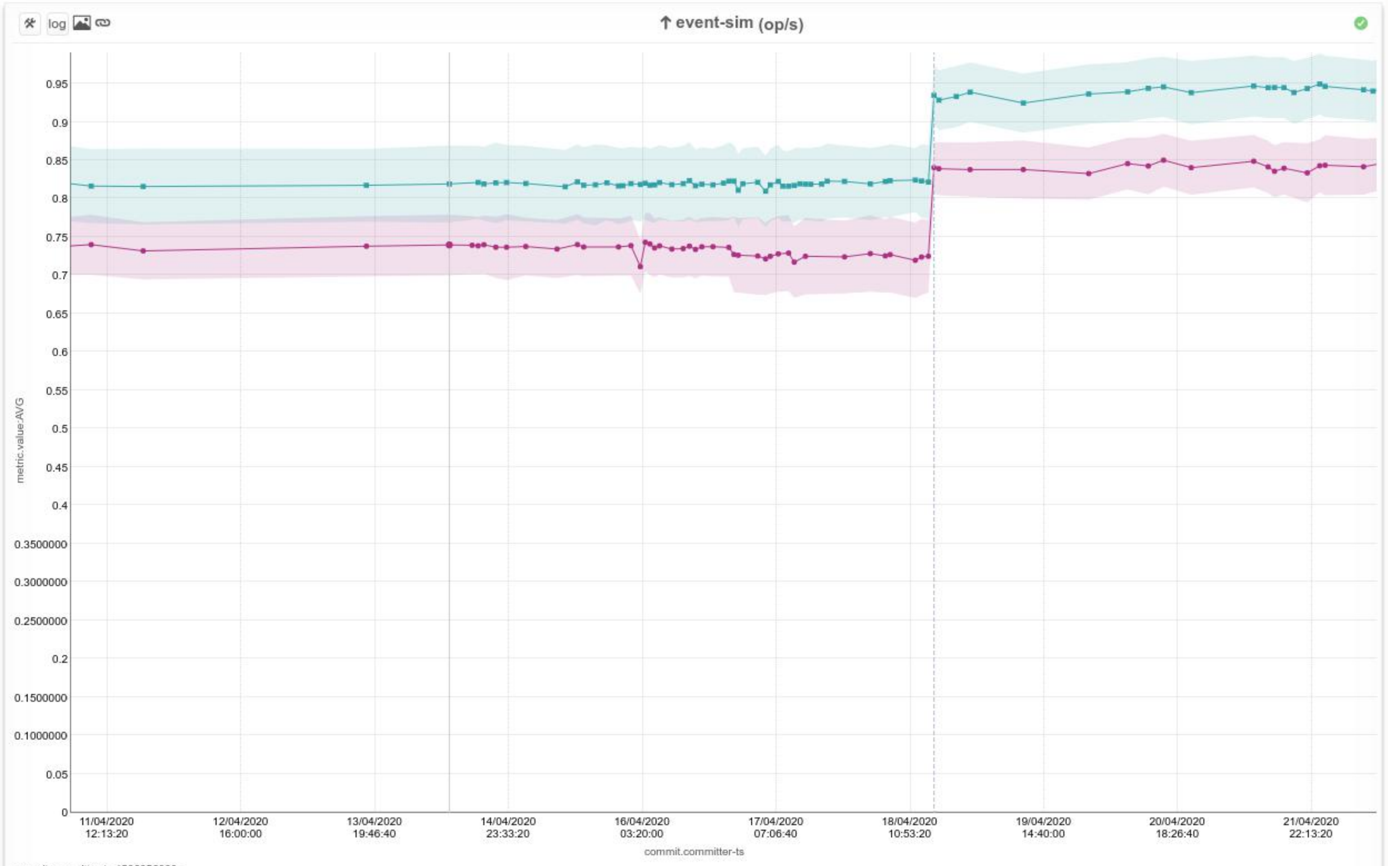
Reads are then scheduled into basic blocks according to their inputs

After floating reads



After fix-reads

Memory out-of-bounds access profiling



commit.committer-ts 1586856080:
●server,graal-core,none,8: ● 0.7394953743659519

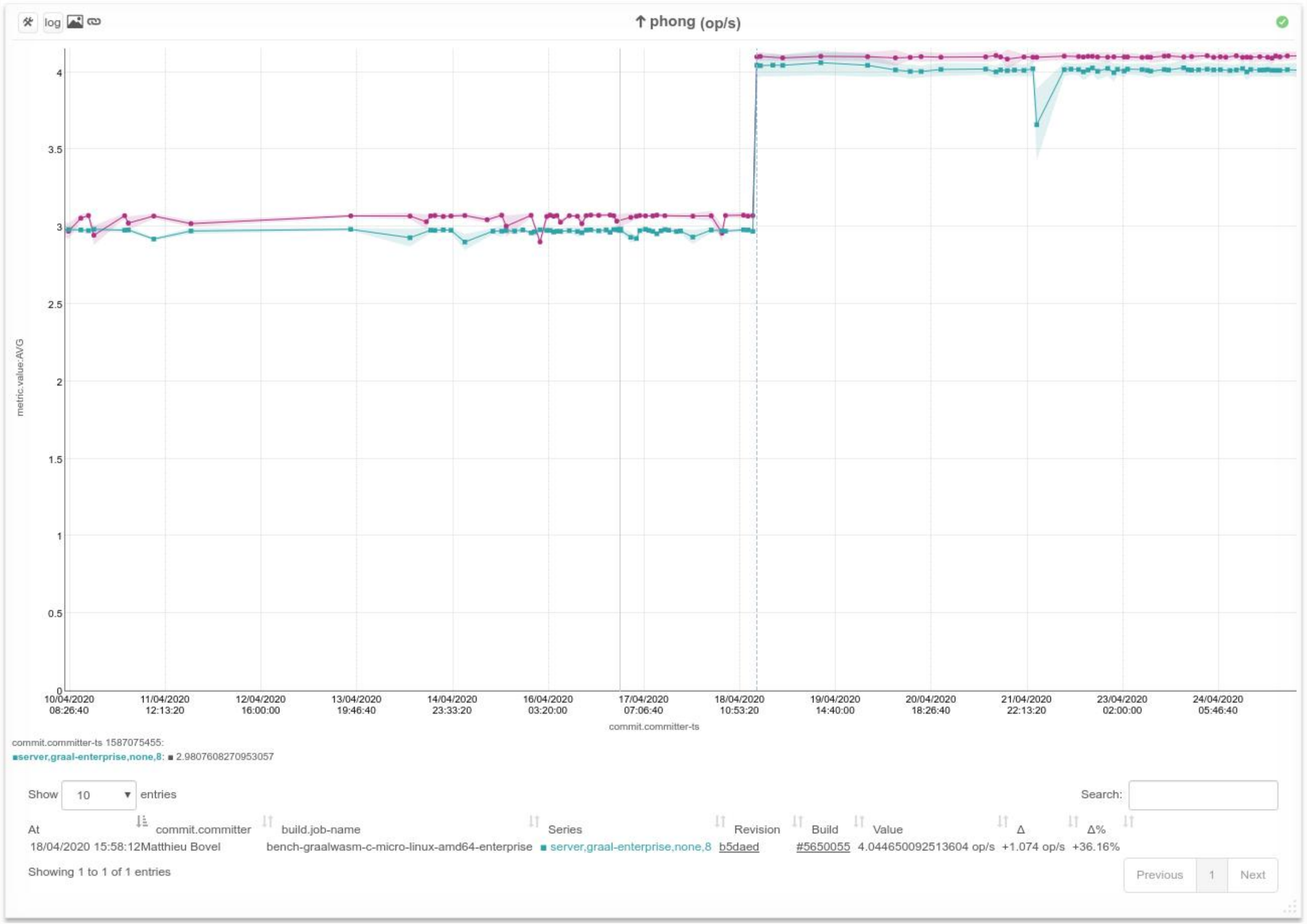
Show 10 ▾ entries   Search:

At	commit.committer	build.job-name	Series	Revision	Build	Value	Δ	Δ%
18/04/2020 15:58:12	Matthieu Bovel	bench-graalwasm-c-micro-linux-amd64-enterprise	■ server,graal-enterprise,none,8	b5daed	#5650055	0.9345965804296438 op/s	+0.113 op/s	+13.76%

Showing 1 to 1 of 1 entries   Previous 1 Next

- 14% improvement on a discrete-event simulation benchmark (*event-sim*)

Memory out-of-bounds access profiling



commit.committer-ts 1587075455:
■server,graal-enterprise,none,8: ■ 2.9807608270953057

Show 10 entries Search:

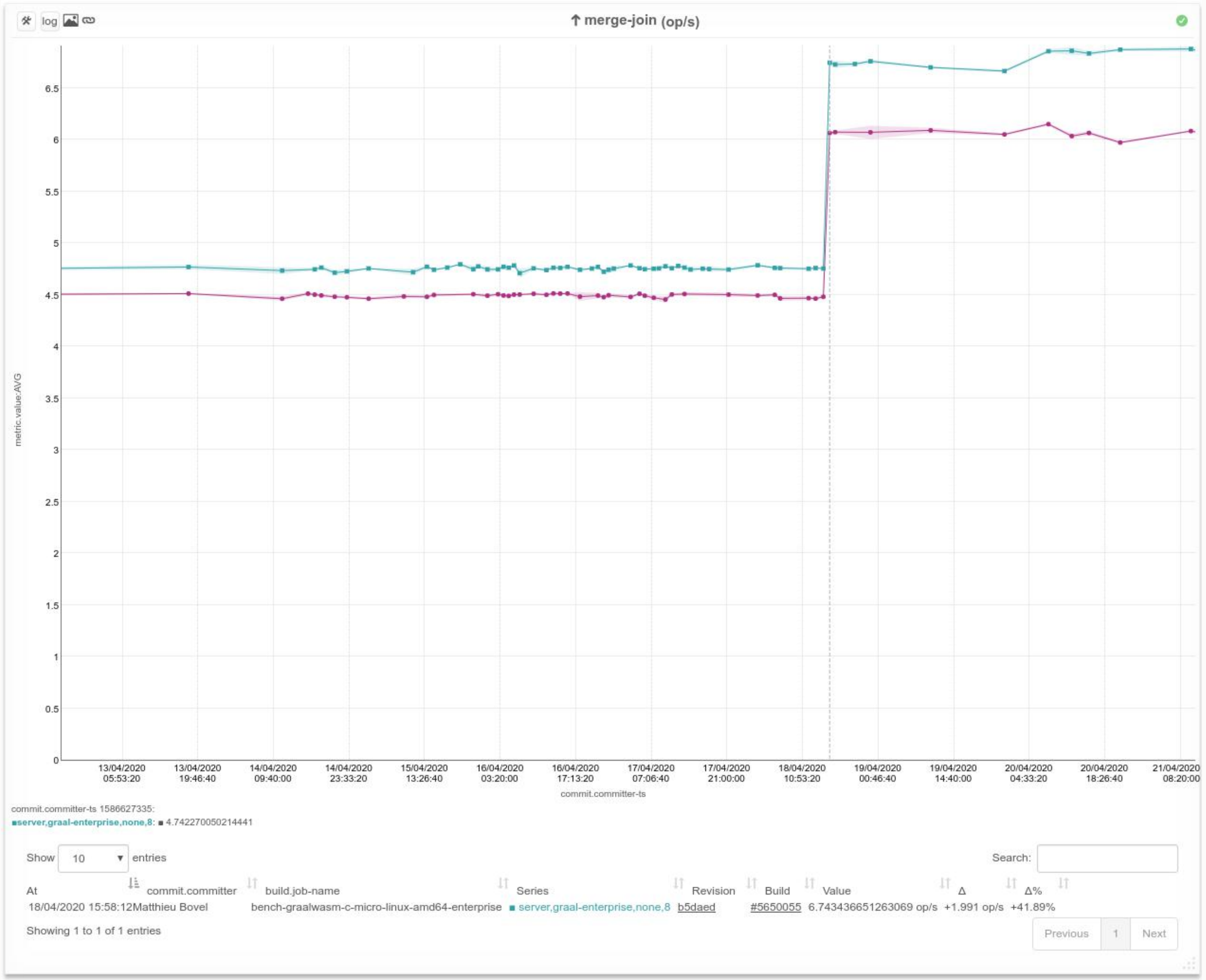
At	commit.committer	build.job-name	Series	Revision	Build	Value	Δ	Δ%
18/04/2020 15:58:12	Matthieu Bovel	bench-graalwasm-c-micro-linux-amd64-enterprise	■ server,graal-enterprise,none,8	b5daed	#5650055	4.044650092513604 op/s	+1.074 op/s	+36.16%

Showing 1 to 1 of 1 entries

Previous 1 Next

- 14% improvement on a discrete-event simulation benchmark (*event-sim*)
- 39% improvement on a 3d-renderer benchmark (*phong*)

Memory out-of-bounds access profiling



commit.committer-ts 1586627335:
■server,graal-enterprise,none,8: ■ 4.742270050214441

Show 10 entries Search:

At	commit.committer	build.job-name	Series	Revision	Build	Value	Δ	Δ%
18/04/2020 15:58:12	Matthieu Bovel	bench-graalwasm-c-micro-linux-amd64-enterprise	■ server,graal-enterprise,none,8	b5daed	#5650055	6.743436651263069 op/s	+1.991 op/s	+41.89%

Showing 1 to 1 of 1 entries

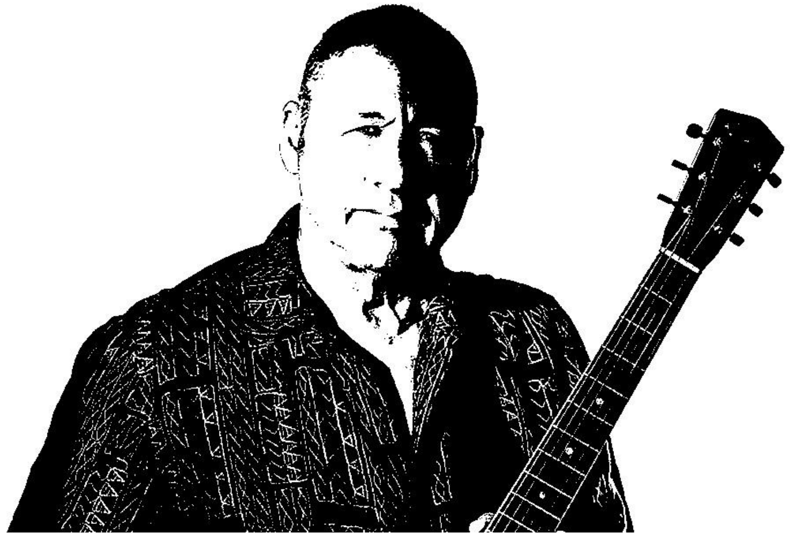
Previous 1 Next

- 14% improvement on a discrete-event simulation benchmark (*event-sim*)
- 39% improvement on a 3d-renderer benchmark (*phong*)
- 42% improvement on a merge-join benchmark (*merge-join*)
- improvements on other benchmarks in the 5%-60% range

Memory out-of-bounds access profiling

Example program: posterize filter

$\rightarrow$



Resetting Truffle frame-slot values

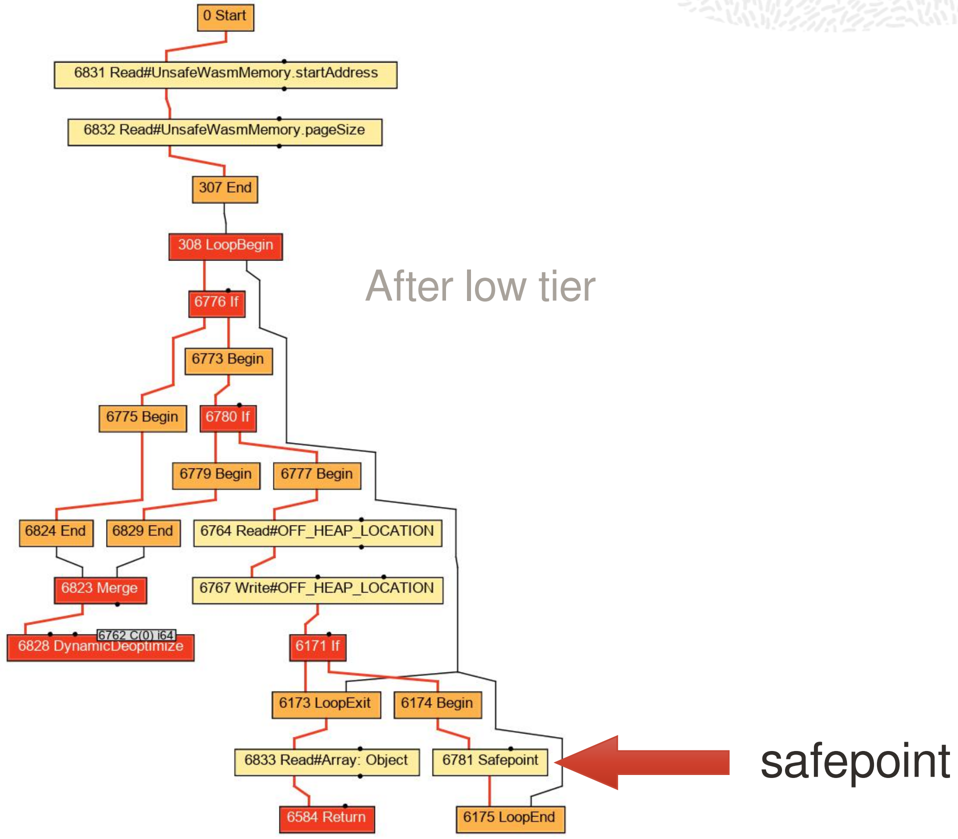
Example program: posterize filter

```
for (uint32_t pixel = 0; pixel != IMAGE_SIZE; ++pixel) {
  uint32_t color = image[pixel % IMAGE_SIZE];
  uint8_t R = (color & 0xFF000000) >> 24;
  uint8_t G = (color & 0x00FF0000) >> 16;
  uint8_t B = (color & 0x0000FF00) >> 8;
  double luminance = (0.2126 * R + 0.7152 * G + 0.0722 * B);
  result[pixel % IMAGE_SIZE] = luminance > 127 ? UINT32_MAX : 0xFF;
}
```

Truffle frame-slot values

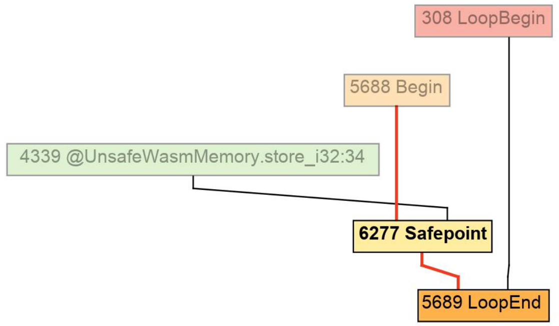
In GraalVM, a program can deoptimize at safepoints

When this happens, the values of the local variables must be copied back to the corresponding frame slots of the interpreter.



Truffle frame-slot values

Frame-state nodes encode local variable state of the Java program

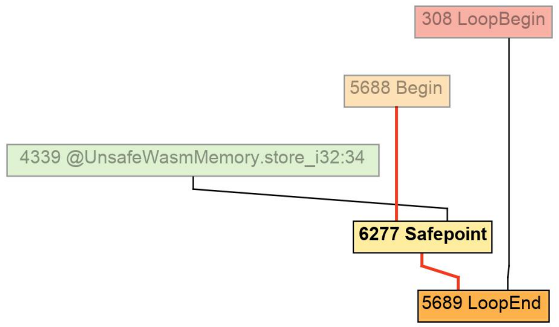


After low tier

Truffle frame-slot values

Frame-state nodes encode local variable state of the Java program

A Truffle interpreter is a Java program, so the values of its local variables must be linked to the frame state node.



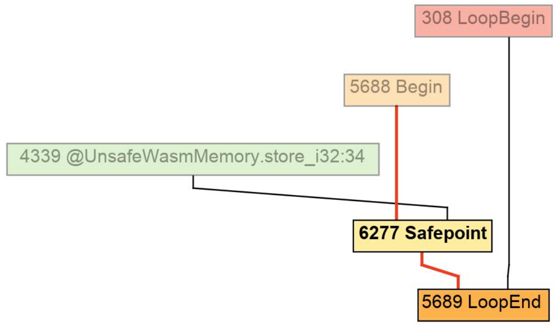
After low tier

Truffle frame-slot values

Frame-state nodes encode local variable state of the Java program

A Truffle interpreter is a Java program, so the values of its local variables must be linked to the frame state node.

```
int execute(VirtualFrame frame) {
  int offset = initialOffset;
  ...
```



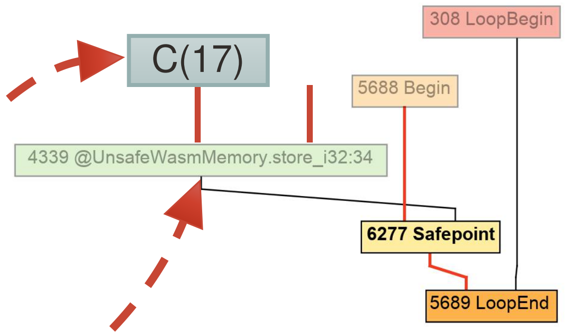
After low tier

Truffle frame-slot values

Frame-state nodes encode local variable state of the Java program

A Truffle interpreter is a Java program, so the values of its local variables must be linked to the frame state node.

```
int execute(VirtualFrame frame) {
  int offset = initialOffset;
  ...
```



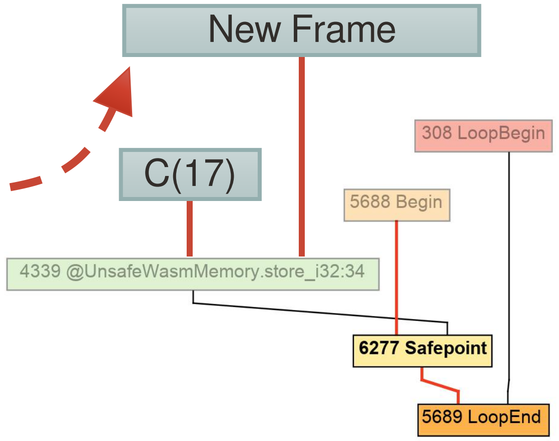
After low tier

Truffle frame-slot values

Truffle Frames contain the values of the local variables of the guest program

One of the local variables in the interpreter is the Truffle frame, which represents the local variables of the guest program. The frame is represented as an allocation of the Frame object.

```
int execute(VirtualFrame frame) {
   int offset = initialOffset;
   ...
```



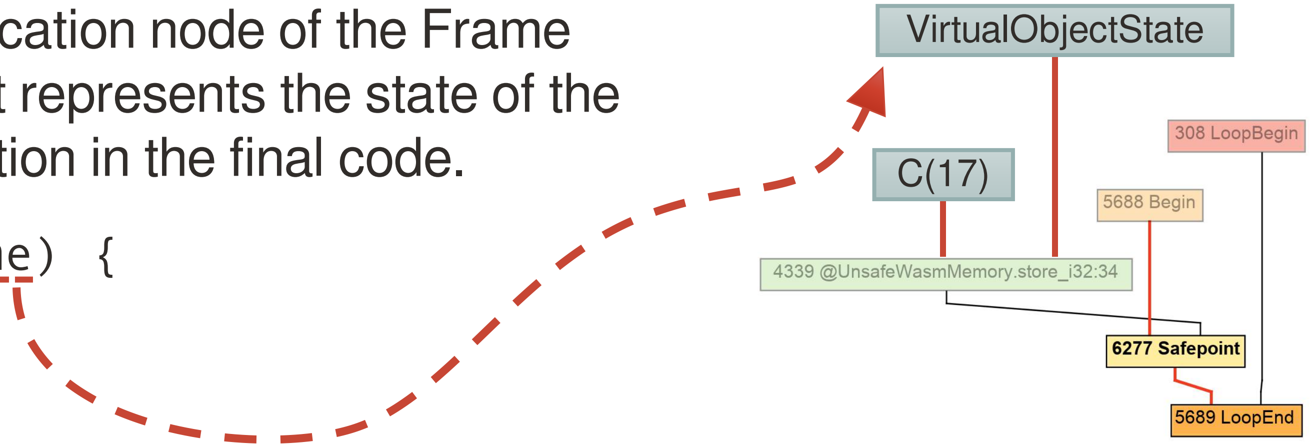
“After PE”

Truffle frame-slot values

Truffle Frames contain the values of the local variables of the guest program

The escape analysis replaces the allocation node of the Frame with a virtual object state - a node that represents the state of the Frame, but does not require an allocation in the final code.

```
int execute(VirtualFrame frame) {
  int offset = initialOffset;
  ...
```



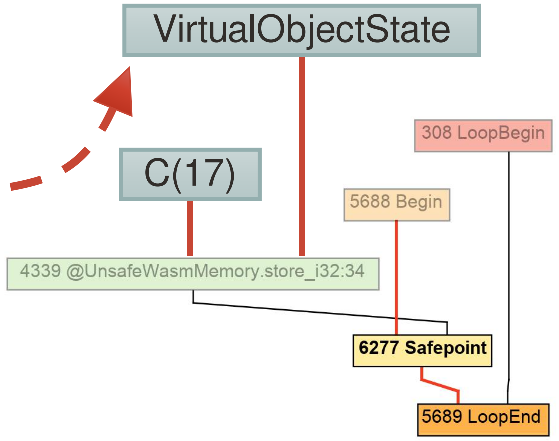
“After escape analysis”

Resetting Truffle frame-slot values

Truffle Frames contain the values of the local variables of the guest program

If the compiled code of the guest program gets deoptimized, then the Truffle Frame object **must be allocated** (this is the same as with any other object in a Java program).

```
int execute(VirtualFrame frame) {
  int offset = initialOffset;
  ...
```



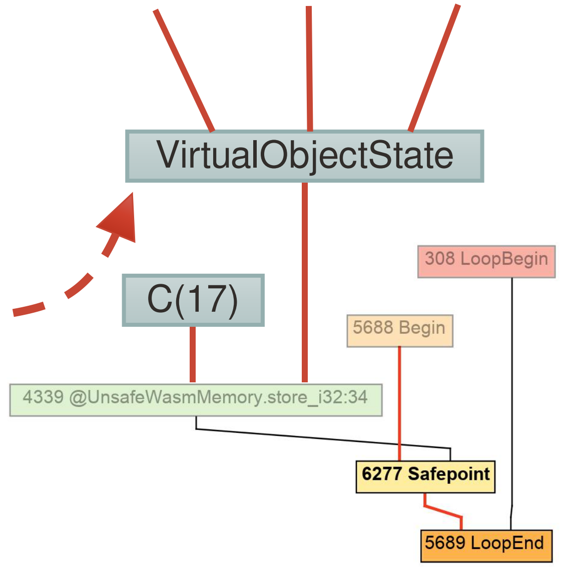
After low tier

Truffle frame-slot values

Truffle Frames contain the values of the local variables of the guest program

By looking at the inputs of the VirtualObjectState node that corresponds to the Truffle Frame, we can see which values in the guest program were “saved”.

```
  int execute(VirtualFrame frame) {
    int offset = initialOffset;
    ...
```

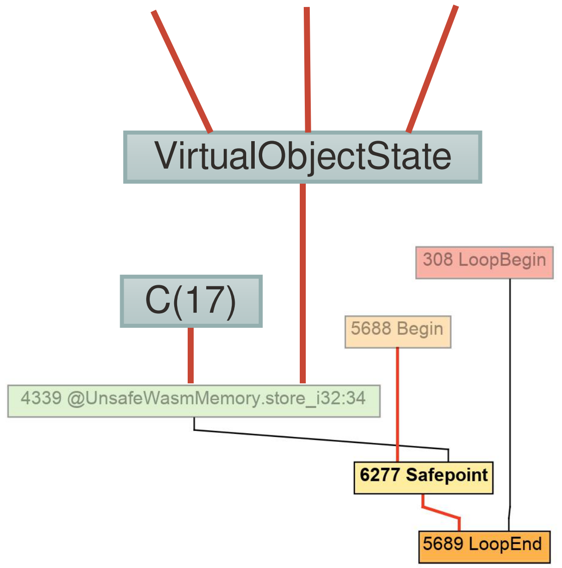


After low tier

Truffle frame-slot values

The state of the Frame must be tracked in the compiled code

The corresponding values have to stay live at the point where the frame state is used. This means that they must be stored either in a register or on the program stack.



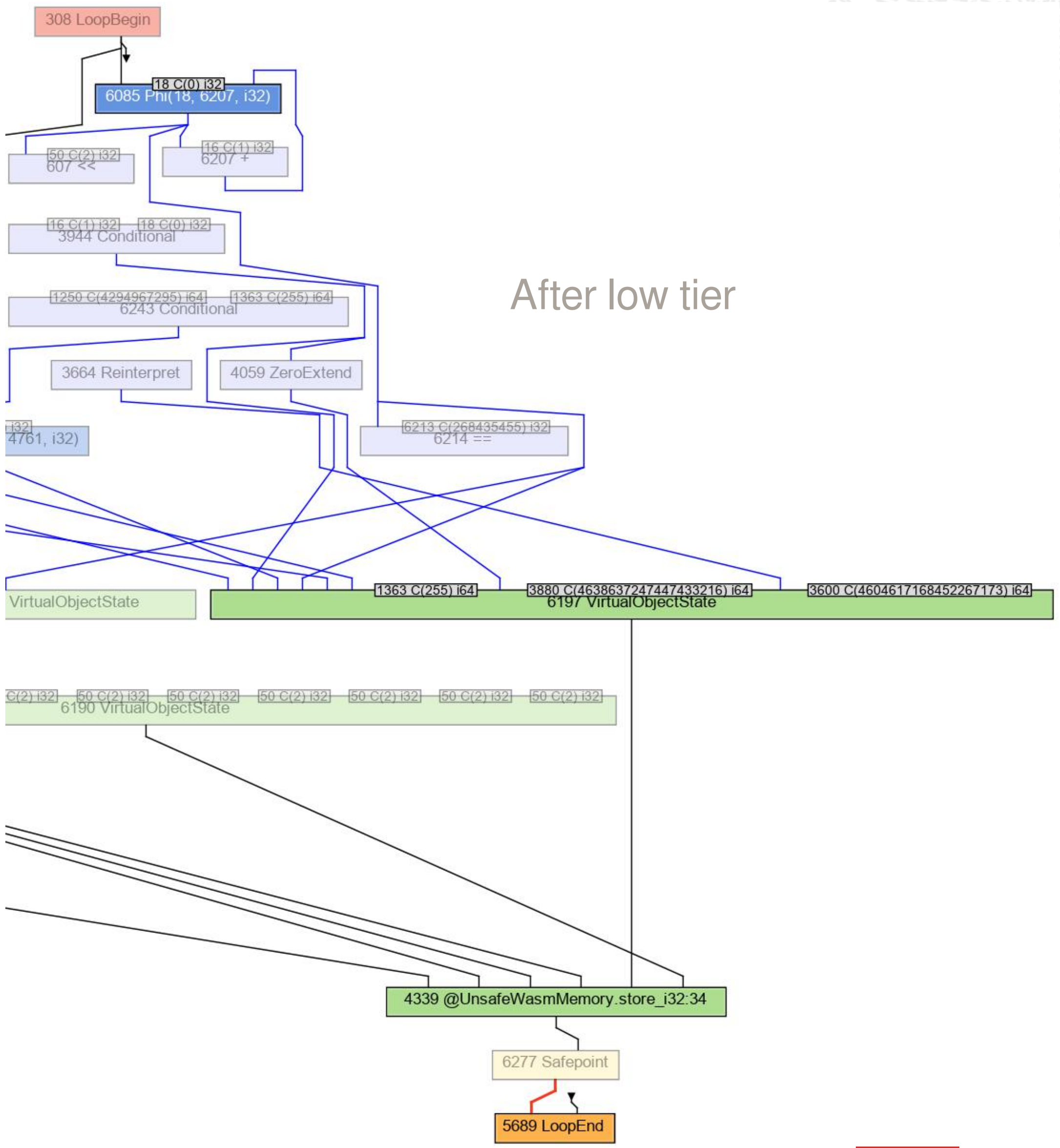
After low tier

Truffle frame-slot values

By inspecting the state of the frame, we can see what's tracked

A lot of values tracked. Recall, the original C program.

```
for (uint32_t pixel = 0; pixel != IMAGE_SIZE; ++pixel) {
  uint32_t color = image[pixel % IMAGE_SIZE];
  uint8_t R = (color & 0xFF000000) >> 24;
  uint8_t G = (color & 0x00FF0000) >> 16;
  uint8_t B = (color & 0x0000FF00) >> 8;
  double luminance =
    (0.2126 * R + 0.7152 * G + 0.0722 * B);
  result[pixel % IMAGE_SIZE] =
    luminance > 127 ? UINT32_MAX : 0xFF;
  black_pixels += luminance > 127 ? 0 : 1;
}
```



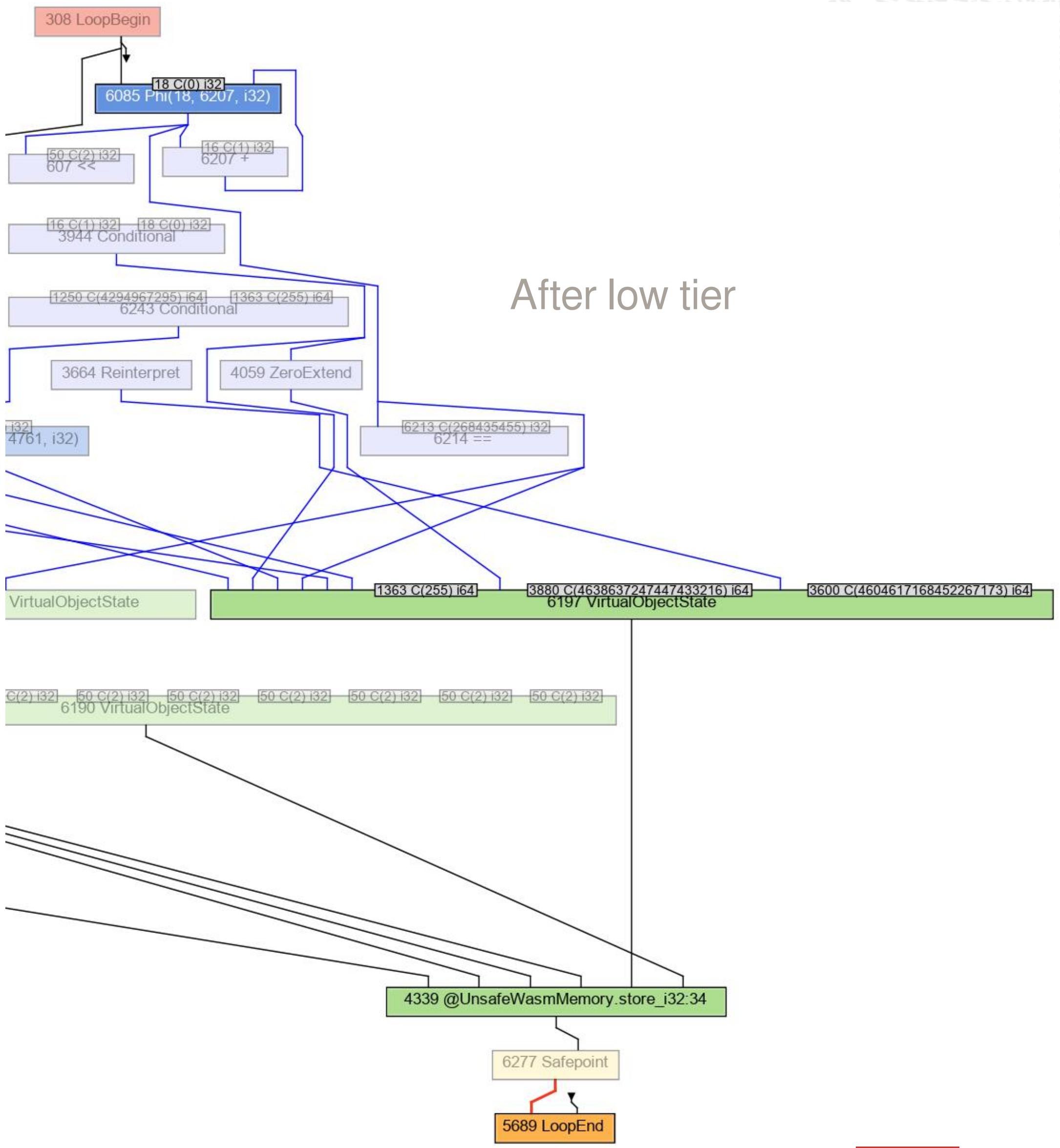
After low tier

Truffle frame-slot values

By inspecting the state of the frame, we can see what's tracked

We need to at least track these 2 values at loop end.

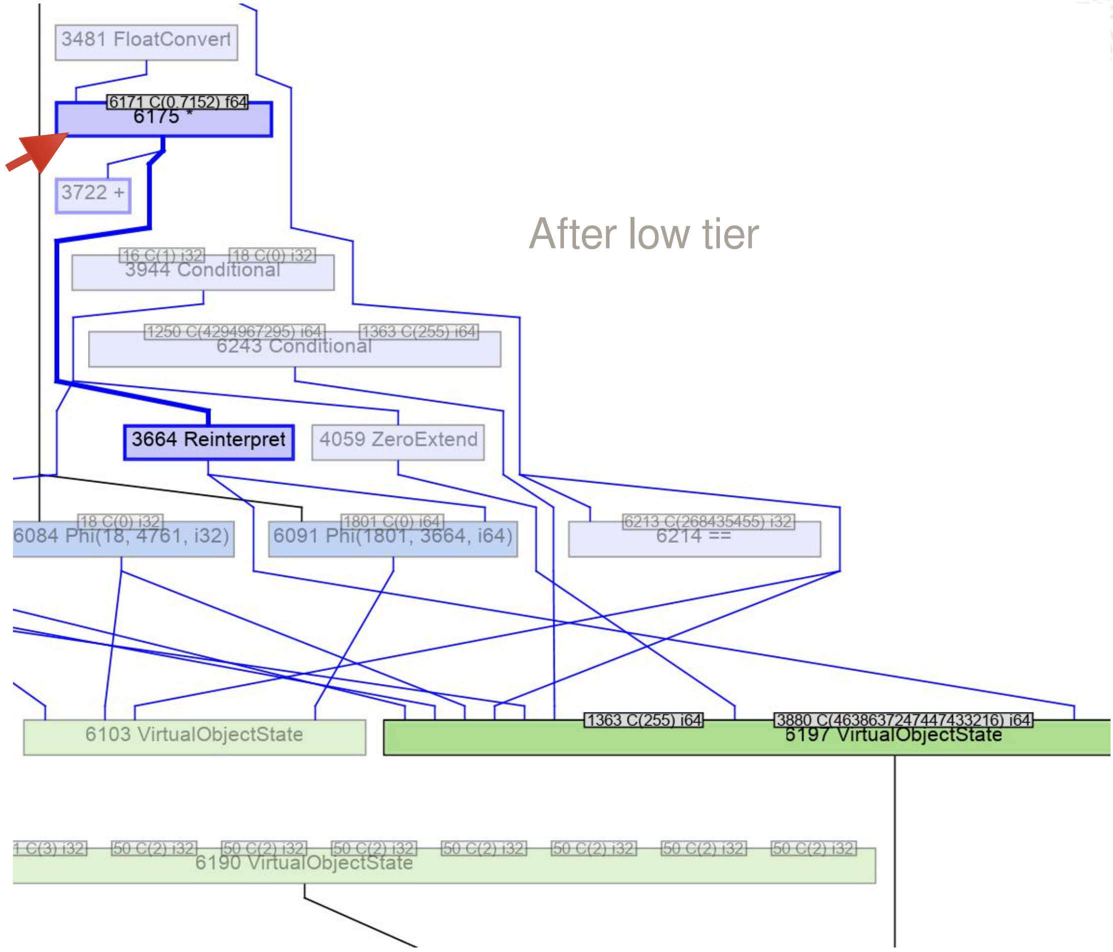
```
for (uint32_t pixel = 0; pixel != IMAGE_SIZE; ++pixel) {
  uint32_t color = image[pixel % IMAGE_SIZE];
  uint8_t R = (color & 0xFF000000) >> 24;
  uint8_t G = (color & 0x00FF0000) >> 16;
  uint8_t B = (color & 0x0000FF00) >> 8;
  double luminance =
    (0.2126 * R + 0.7152 * G + 0.0722 * B);
  result[pixel % IMAGE_SIZE] =
    luminance > 127 ? UINT32_MAX : 0xFF;
  black_pixels += luminance > 127 ? 0 : 1;
}
```



Truffle frame-slot values

There are more values stored in the frame than what's necessary

What's this?



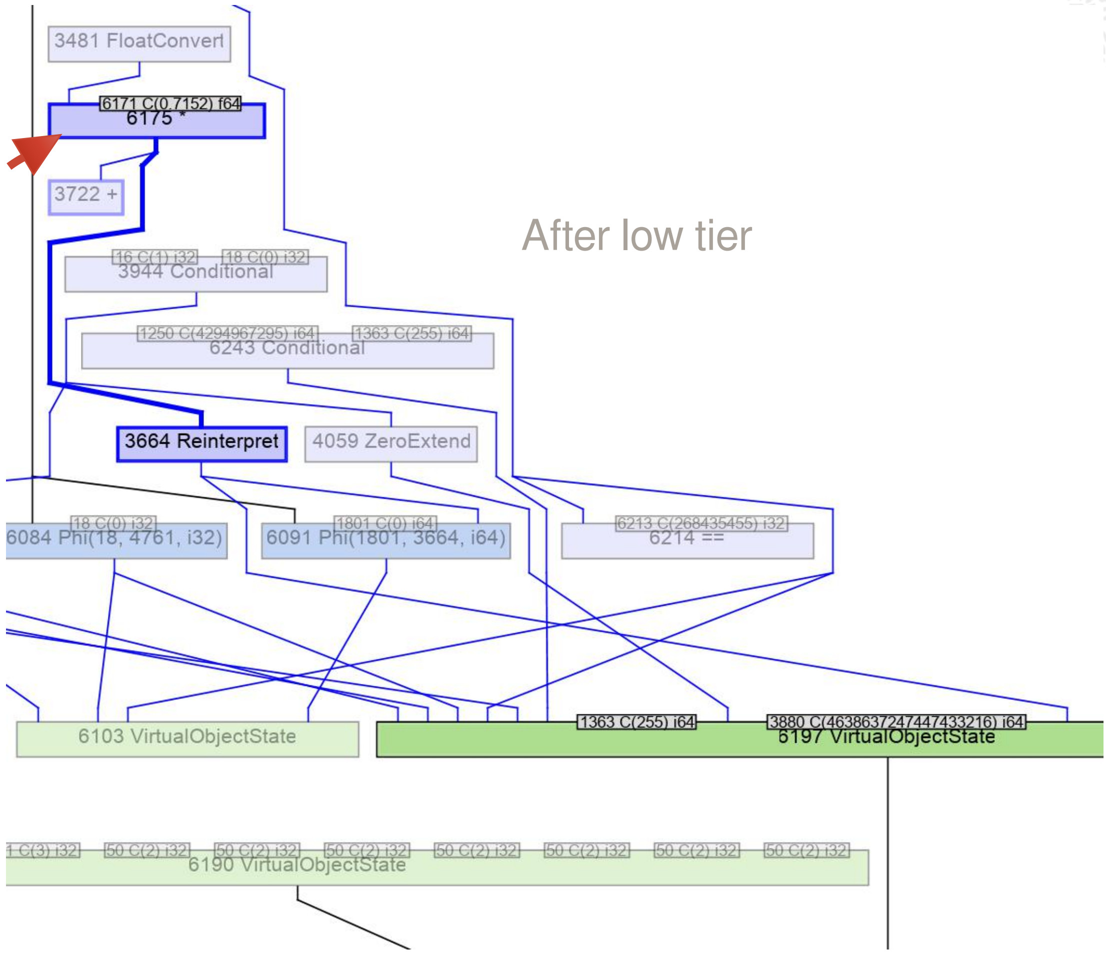
```
for (uint32_t pixel = 0; pixel != IMAGE_SIZE; ++pixel) {
  uint32_t color = image[pixel % IMAGE_SIZE];
  uint8_t R = (color & 0xFF000000) >> 24;
  uint8_t G = (color & 0x00FF0000) >> 16;
  uint8_t B = (color & 0x0000FF00) >> 8;
  double luminance =
    (0.2126 * R + 0.7152 * G + 0.0722 * B);
  result[pixel % IMAGE_SIZE] =
    luminance > 127 ? UINT32_MAX : 0xFF;
  black_pixels += luminance > 127 ? 0 : 1;
}
```

Truffle frame-slot values

There are more values stored in the frame than what's necessary

No need to store the intermediate expression result.

```
for (uint32_t pixel = 0; pixel != IMAGE_SIZE; ++pixel) {
  uint32_t color = image[pixel % IMAGE_SIZE];
  uint8_t R = (color & 0xFF000000) >> 24;
  uint8_t G = (color & 0x00FF0000) >> 16;
  uint8_t B = (color & 0x0000FF00) >> 8;
  double luminance =
    (0.2126 * R + 0.7152 * G + 0.0722 * B);
  result[pixel % IMAGE_SIZE] =
    luminance > 127 ? UINT32_MAX : 0xFF;
  black_pixels += luminance > 127 ? 0 : 1;
}
```



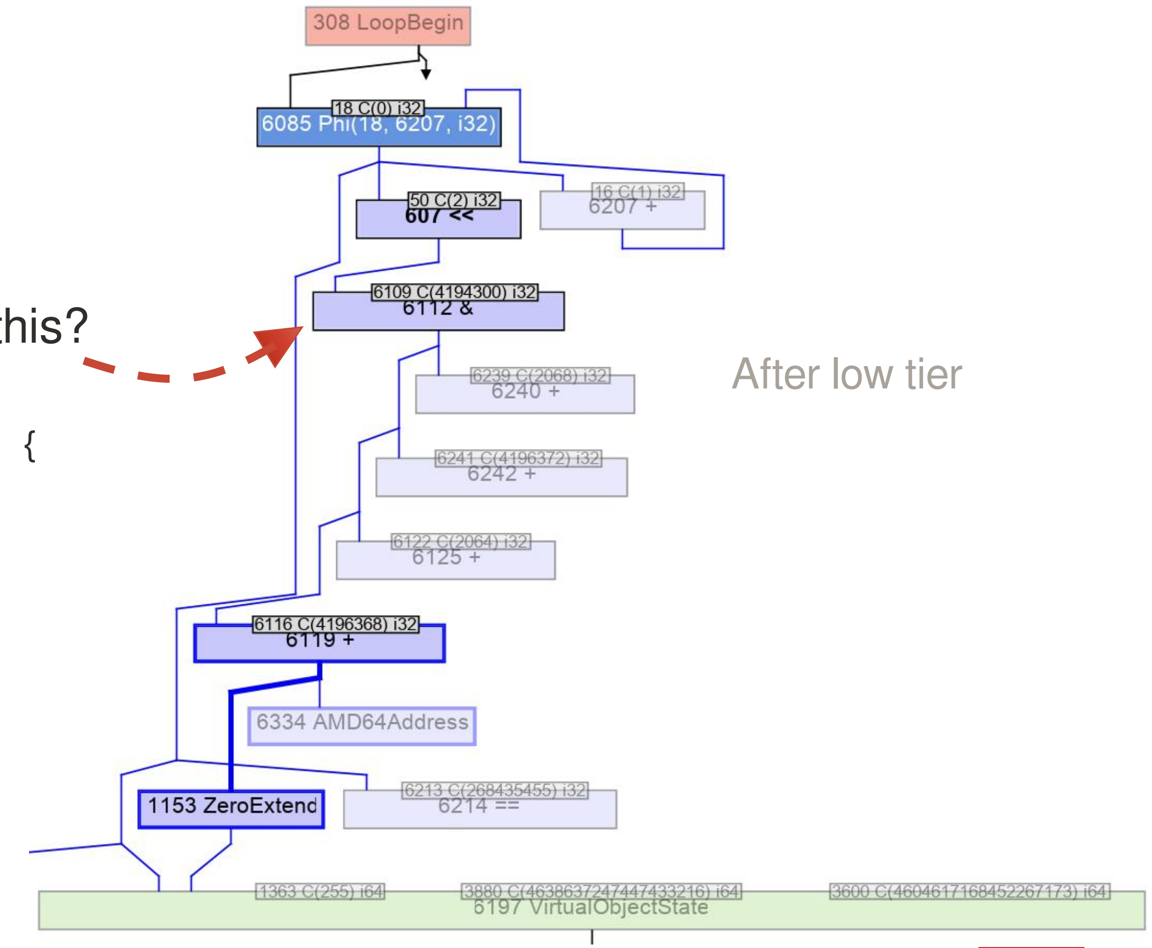
After low tier

Resetting Truffle frame-slot values

There are more values stored in the frame than what's necessary

What's this?

```
for (uint32_t pixel = 0; pixel != IMAGE_SIZE; ++pixel) {
  uint32_t color = image[pixel % IMAGE_SIZE];
  uint8_t R = (color & 0xFF000000) >> 24;
  uint8_t G = (color & 0x00FF0000) >> 16;
  uint8_t B = (color & 0x0000FF00) >> 8;
  double luminance =
    (0.2126 * R + 0.7152 * G + 0.0722 * B);
  result[pixel % IMAGE_SIZE] =
    luminance > 127 ? UINT32_MAX : 0xFF;
  black_pixels += luminance > 127 ? 0 : 1;
}
```



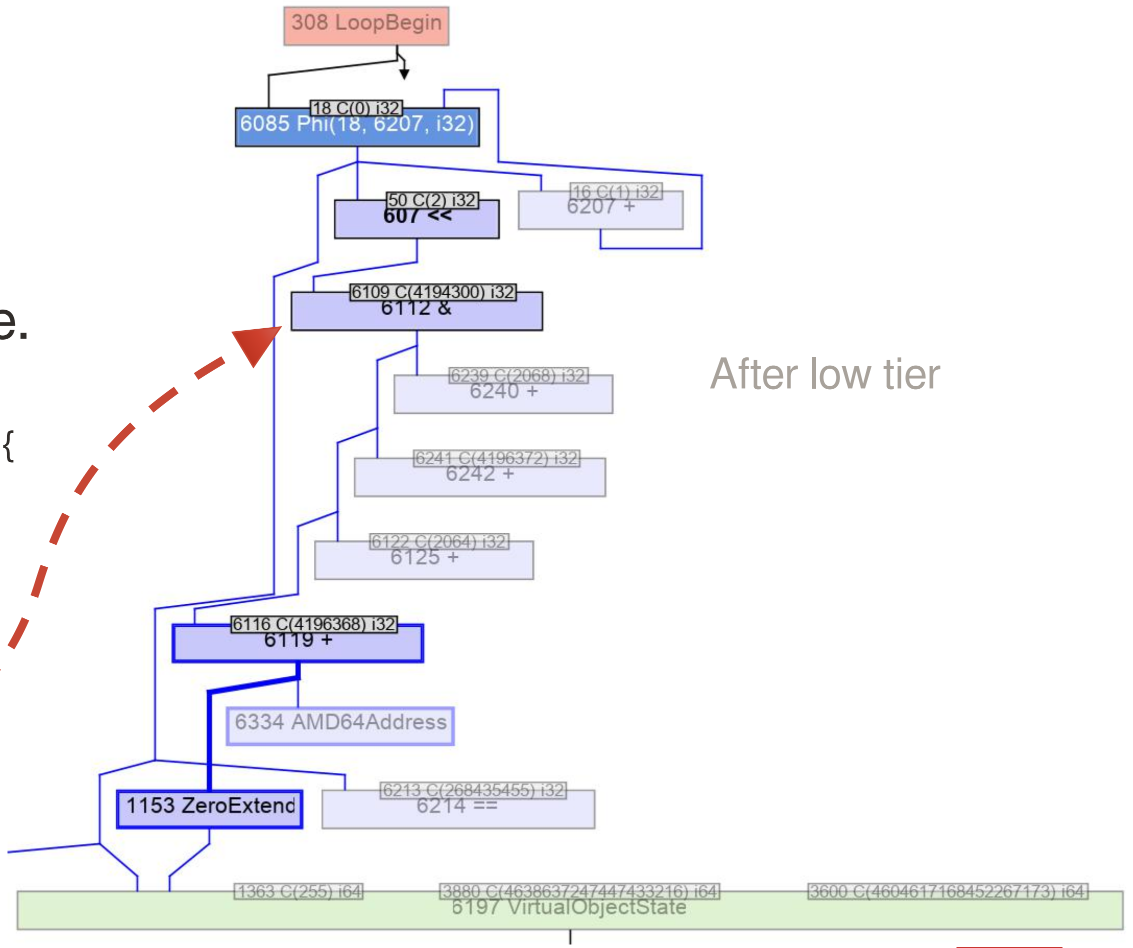
After low tier

Resetting Truffle frame-slot values

There are more values stored in the frame than what's necessary

No need to store the array offset after the write is done.

```
for (uint32_t pixel = 0; pixel != IMAGE_SIZE; ++pixel) {
  uint32_t color = image[pixel % IMAGE_SIZE];
  uint8_t R = (color & 0xFF000000) >> 24;
  uint8_t G = (color & 0x00FF0000) >> 16;
  uint8_t B = (color & 0x0000FF00) >> 8;
  double luminance =
    (0.2126 * R + 0.7152 * G + 0.0722 * B);
  result[pixel % IMAGE_SIZE] =
    luminance > 127 ? UINT32_MAX : 0xFF;
  black_pixels += luminance > 127 ? 0 : 1;
}
```



After low tier

Resetting Truffle frame-slot values

Why do these extra values persist?

To understand why, we need to look at the WebAssembly code that the C program was compiled to.

```
loop  ;; label = @1
  local.get 2
  i32.const 2
  i32.shl
  i32.const 4194300
  i32.and
  local.tee 0
  i32.const 4196368
  i32.add
  i32.const -1
  i32.const 255
  local.get 0
  i32.const 2064
  i32.add
  i32.load
  local.tee 0
  i32.const 8
  i32.shr_u
  i32.const 255
  i32.and
  f64.convert_i32_s
  f64.const 0x1.27bb2fec56d5dp-4 (;=0.0722;)
  f64.mul
  local.get 0
  i32.const 24
  i32.shr_u
  f64.convert_i32_s
  f64.const 0x1.b367a0f9096bcp-3 (;=0.2126;)
  f64.mul
  local.get 0
  i32.const 16
  i32.shr_u
  i32.const 255
  i32.and
  f64.convert_i32_s
  f64.const 0x1.6e2eb1c432ca5p-1 (;=0.7152;)
  f64.mul
  f64.add
  f64.add
  f64.const 0x1.fcp+6 (;=127;)
  f64.gt
  local.tee 0
  select
  i32.store
  local.get 1
  local.get 0
  i32.const 1
  i32.xor
  i32.add
  local.set 1
  local.get 2
  i32.const 1
  i32.add
  local.tee 2
  i32.const 268435456
  i32.ne
  br_if 0 (;@1;)
end
```

Resetting Truffle frame-slot values

Part of the guest language Frame is used as an expression stack

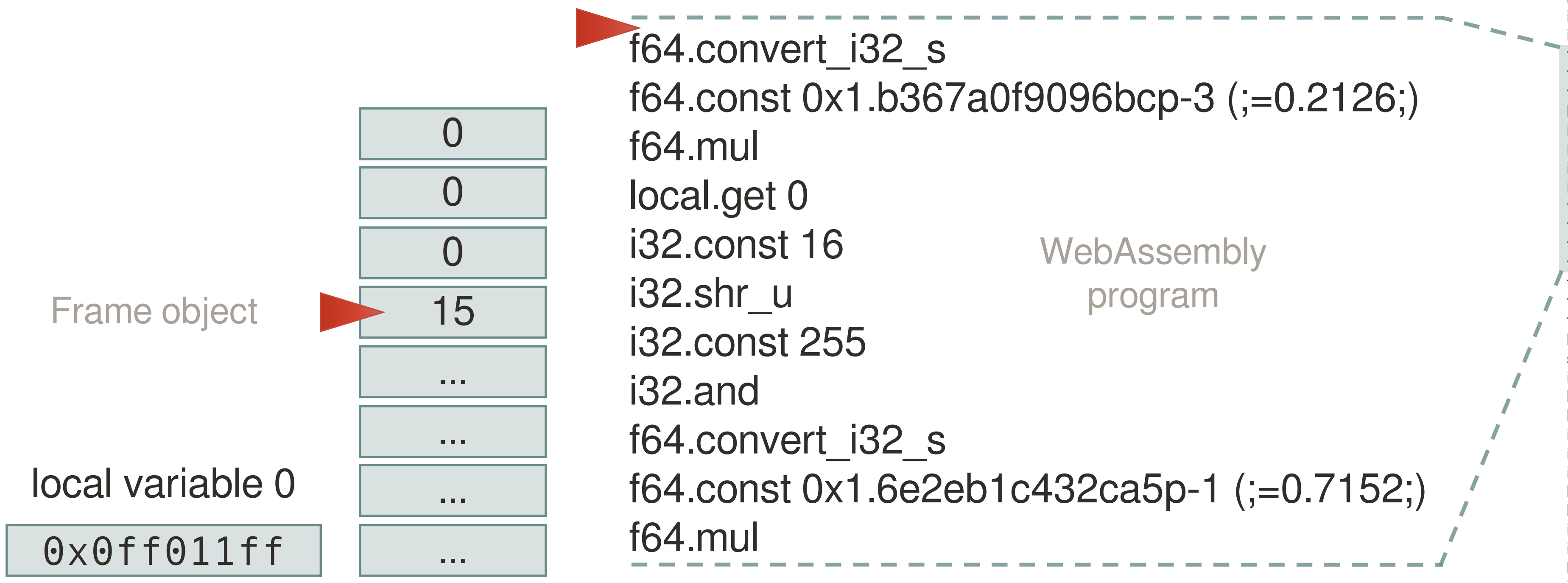
```
f64.convert_i32_s
f64.const 0x1.b367a0f9096bcp-3 (;=0.2126;)
f64.mul
local.get 0
i32.const 16
i32.shr_u
i32.const 255
i32.and
f64.convert_i32_s
f64.const 0x1.6e2eb1c432ca5p-1 (;=0.7152;)
f64.mul
```

WebAssembly program

```
loop  ;; label = @1
 local.get 2
 i32.const 2
 i32.shl
 i32.const 4194300
 i32.and
 local.tee 0
 i32.const 4196368
 i32.add
 i32.const -1
 i32.const 255
 local.get 0
 i32.const 2064
 i32.add
 i32.load
 local.tee 0
 i32.const 8
 i32.shr_u
 i32.const 255
 i32.and
 f64.convert_i32_s
 f64.const 0x1.27bb2fec56d5dp-4 (;=0.0722;)
 f64.mul
 local.get 0
 i32.const 24
 i32.shr_u
 f64.convert_i32_s
 f64.const 0x1.b367a0f9096bcp-3 (;=0.2126;)
 f64.mul
 local.get 0
 i32.const 16
 i32.shr_u
 i32.const 255
 i32.and
 f64.convert_i32_s
 f64.const 0x1.6e2eb1c432ca5p-1 (;=0.7152;)
 f64.mul
 f64.add
 f64.add
 f64.const 0x1.fcp+6 (;=127;)
 f64.gt
 local.tee 0
 select
 i32.store
 local.get 1
 local.get 0
 i32.const 1
 i32.xor
 i32.add
 local.set 1
 local.get 2
 i32.const 1
 i32.add
 local.tee 2
 i32.const 268435456
 i32.ne
 br_if 0 (;@1;)
end
```

Resetting Truffle frame-slot values

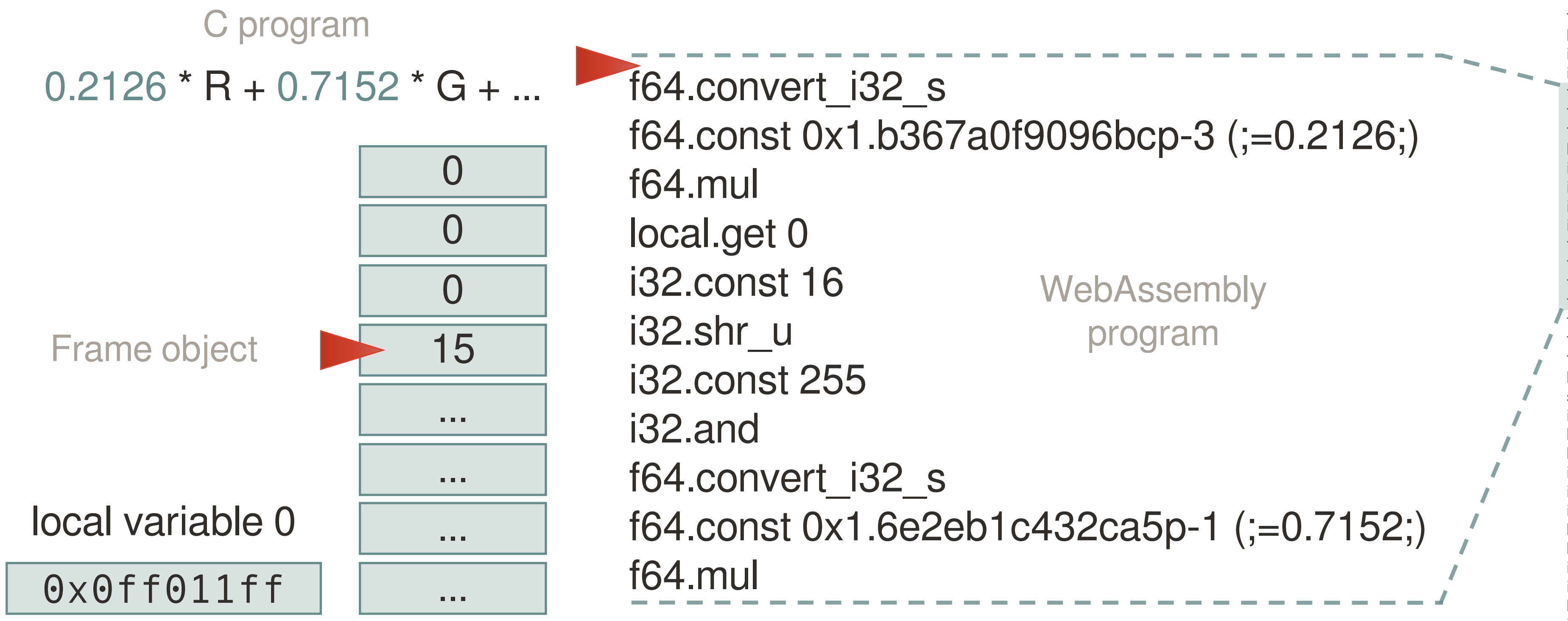
Part of the guest language Frame is used as an expression stack



```
f64.convert_i32_s
f64.const 0x1.b367a0f9096bcp-3 (;=0.2126;)
f64.mul
local.get 0
i32.const 16
i32.shr_u
i32.const 255
i32.and
f64.convert_i32_s
f64.const 0x1.6e2eb1c432ca5p-1 (;=0.7152;)
f64.mul
```

```
loop  ;; label = @1
 local.get 2
 i32.const 2
 i32.shl
 i32.const 4194300
 i32.and
 local.tee 0
 i32.const 4196368
 i32.add
 i32.const -1
 i32.const 255
 local.get 0
 i32.const 2064
 i32.add
 i32.load
 local.tee 0
 i32.const 8
 i32.shr_u
 i32.const 255
 i32.and
 f64.convert_i32_s
 f64.const 0x1.27bb2fec56d5dp-4 (;=0.0722;)
 f64.mul
 local.get 0
 i32.const 24
 i32.shr_u
 f64.convert_i32_s
 f64.const 0x1.b367a0f9096bcp-3 (;=0.2126;)
 f64.mul
 local.get 0
 i32.const 16
 i32.shr_u
 i32.const 255
 i32.and
 f64.convert_i32_s
 f64.const 0x1.6e2eb1c432ca5p-1 (;=0.7152;)
 f64.mul
 f64.add
 f64.add
 f64.const 0x1.fcp+6 (;=127;)
 f64.gt
 local.tee 0
 select
 i32.store
 local.get 1
 local.get 0
 i32.const 1
 i32.xor
 i32.add
 local.set 1
 local.get 2
 i32.const 1
 i32.add
 local.tee 2
 i32.const 268435456
 i32.ne
 br_if 0 (;@1;)
end
```

Resetting Truffle frame-slot values



```
f64.convert_i32_s
f64.const 0x1.b367a0f9096bcp-3 (;=0.2126;)
f64.mul
local.get 0
i32.const 16
i32.shr_u
i32.const 255
i32.and
f64.convert_i32_s
f64.const 0x1.6e2eb1c432ca5p-1 (;=0.7152;)
f64.mul
```

```
loop  ;; label = @1
 local.get 2
 i32.const 2
 i32.shl
 i32.const 4194300
 i32.and
 local.tee 0
 i32.const 4196368
 i32.add
 i32.const -1
 i32.const 255
 local.get 0
 i32.const 2064
 i32.add
 i32.load
 local.tee 0
 i32.const 8
 i32.shr_u
 i32.const 255
 i32.and
 f64.convert_i32_s
 f64.const 0x1.27bb2fec56d5dp-4 (;=0.0722;)
 f64.mul
 local.get 0
 i32.const 24
 i32.shr_u
 f64.convert_i32_s
 f64.const 0x1.b367a0f9096bcp-3 (;=0.2126;)
 f64.mul
 local.get 0
 i32.const 16
 i32.shr_u
 i32.const 255
 i32.and
 f64.convert_i32_s
 f64.const 0x1.6e2eb1c432ca5p-1 (;=0.7152;)
 f64.mul
 f64.add
 f64.add
 f64.const 0x1.fcp+6 (;=127;)
 f64.gt
 local.tee 0
 select
 i32.store
 local.get 1
 local.get 0
 i32.const 1
 i32.xor
 i32.add
 local.set 1
 local.get 2
 i32.const 1
 i32.add
 local.tee 2
 i32.const 268435456
 i32.ne
 br_if 0 (;@1;)
end
```

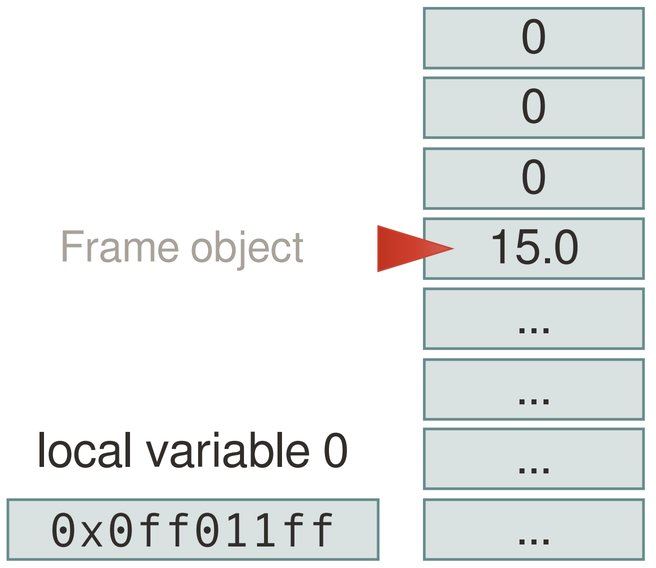
Resetting Truffle frame-slot values

GraalWasm

```java
case F64_CONVERT_I64_S:
case F64_CONVERT_I64_U:
  long x = pop(frame, stackPointer);
  double result = (double) x;
  pushDouble(frame, stackPointer, result);
```

C program

0.2126 * R + 0.7152 * G + ...



WebAssembly program

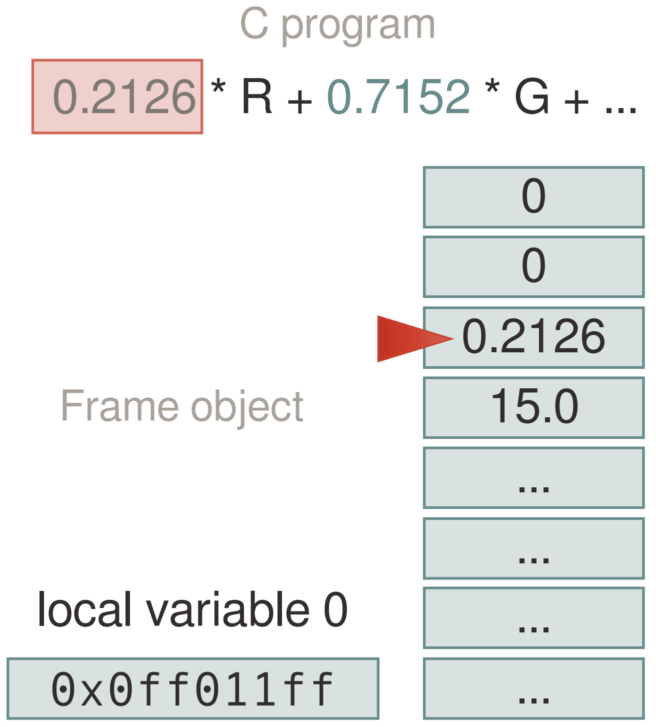
```
f64.convert_i32_s
f64.const 0x1.b367a0f9096bcp-3 (;=0.2126;)
f64.mul
local.get 0
i32.const 16
i32.shr_u
i32.const 255
i32.and
f64.convert_i32_s
f64.const 0x1.6e2eb1c432ca5p-1 (;=0.7152;)
f64.mul
```

```
loop  ;; label = @1
 local.get 2
 i32.const 2
 i32.shl
 i32.const 4194300
 i32.and
 local.tee 0
 i32.const 4196368
 i32.add
 i32.const -1
 i32.const 255
 local.get 0
 i32.const 2064
 i32.add
 i32.load
 local.tee 0
 i32.const 8
 i32.shr_u
 i32.const 255
 i32.and
 f64.convert_i32_s
 f64.const 0x1.27bb2fec56d5dp-4 (;=0.0722;)
 f64.mul
 local.get 0
 i32.const 24
 i32.shr_u
 f64.convert_i32_s
 f64.const 0x1.b367a0f9096bcp-3 (;=0.2126;)
 f64.mul
 local.get 0
 i32.const 16
 i32.shr_u
 i32.const 255
 i32.and
 f64.convert_i32_s
 f64.const 0x1.6e2eb1c432ca5p-1 (;=0.7152;)
 f64.mul
 f64.add
 f64.add
 f64.const 0x1.fcp+6 (;=127;)
 f64.gt
 local.tee 0
 select
 i32.store
 local.get 1
 local.get 0
 i32.const 1
 i32.xor
 i32.add
 local.set 1
 local.get 2
 i32.const 1
 i32.add
 local.tee 2
 i32.const 268435456
 i32.ne
 br_if 0 (;@1;)
end
```

Resetting Truffle frame-slot values

```
case F64_CONST:
  long value = peek8(instructions, offset);
  offset += 8;
  push(frame, stackPointer, value);
```

GraalWasm



```
f64.convert_i32_s
f64.const 0x1.b367a0f9096bcp-3 (;=0.2126;)
f64.mul
local.get 0
i32.const 16
i32.shr_u
i32.const 255
i32.and
f64.convert_i32_s
f64.const 0x1.6e2eb1c432ca5p-1 (;=0.7152;)
f64.mul
```

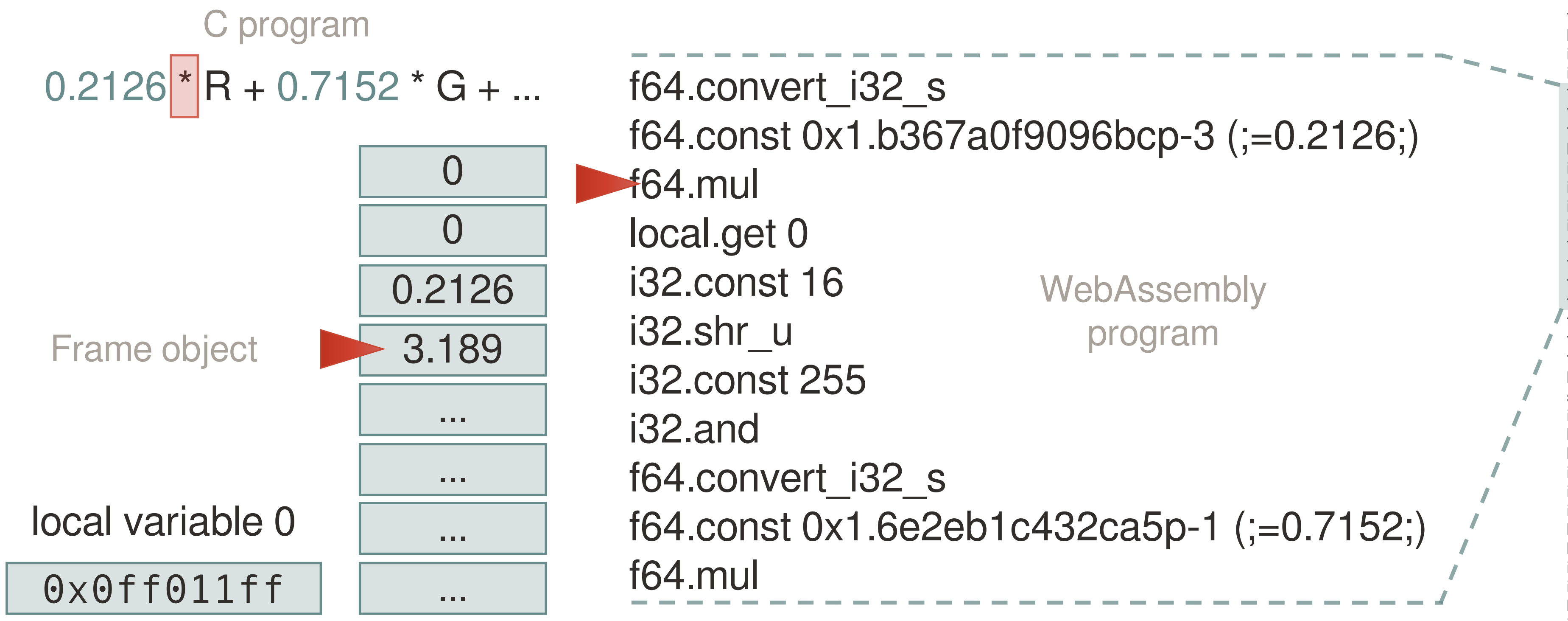
WebAssembly program

```
loop  ;; label = @1
 local.get 2
 i32.const 2
 i32.shl
 i32.const 4194300
 i32.and
 local.tee 0
 i32.const 4196368
 i32.add
 i32.const -1
 i32.const 255
 local.get 0
 i32.const 2064
 i32.add
 i32.load
 local.tee 0
 i32.const 8
 i32.shr_u
 i32.const 255
 i32.and
 f64.convert_i32_s
 f64.const 0x1.27bb2fec56d5dp-4 (;=0.0722;)
 f64.mul
 local.get 0
 i32.const 24
 i32.shr_u
 f64.convert_i32_s
 f64.const 0x1.b367a0f9096bcp-3 (;=0.2126;)
 f64.mul
 local.get 0
 i32.const 16
 i32.shr_u
 i32.const 255
 i32.and
 f64.convert_i32_s
 f64.const 0x1.6e2eb1c432ca5p-1 (;=0.7152;)
 f64.mul
 f64.add
 f64.add
 f64.const 0x1.fcp+6 (;=127;)
 f64.gt
 local.tee 0
 select
 i32.store
 local.get 1
 local.get 0
 i32.const 1
 i32.xor
 i32.add
 local.set 1
 local.get 2
 i32.const 1
 i32.add
 local.tee 2
 i32.const 268435456
 i32.ne
 br_if 0 (;@1;)
end
```

Resetting Truffle frame-slot values

GraalWasm

```
case F64_MUL:
  double x = popDouble(frame, stackPointer);
  double y = popDouble(frame, stackPointer);
  double result = y * x;
  pushDouble(frame, stackPointer, result);
```



```
f64.convert_i32_s
f64.const 0x1.b367a0f9096bcp-3 (;=0.2126;)
f64.mul
local.get 0
i32.const 16
i32.shr_u
i32.const 255
i32.and
f64.convert_i32_s
f64.const 0x1.6e2eb1c432ca5p-1 (;=0.7152;)
f64.mul
```

```
loop  ;; label = @1
 local.get 2
 i32.const 2
 i32.shl
 i32.const 4194300
 i32.and
 local.tee 0
 i32.const 4196368
 i32.add
 i32.const -1
 i32.const 255
 local.get 0
 i32.const 2064
 i32.add
 i32.load
 local.tee 0
 i32.const 8
 i32.shr_u
 i32.const 255
 i32.and
 f64.convert_i32_s
 f64.const 0x1.27bb2fec56d5dp-4 (;=0.0722;)
 f64.mul
 local.get 0
 i32.const 24
 i32.shr_u
 f64.convert_i32_s
 f64.const 0x1.b367a0f9096bcp-3 (;=0.2126;)
 f64.mul
 local.get 0
 i32.const 16
 i32.shr_u
 i32.const 255
 i32.and
 f64.convert_i32_s
 f64.const 0x1.6e2eb1c432ca5p-1 (;=0.7152;)
 f64.mul
 f64.add
 f64.add
 f64.const 0x1.fcp+6 (;=127;)
 f64.gt
 local.tee 0
 select
 i32.store
 local.get 1
 local.get 0
 i32.const 1
 i32.xor
 i32.add
 local.set 1
 local.get 2
 i32.const 1
 i32.add
 local.tee 2
 i32.const 268435456
 i32.ne
 br_if 0 (;@1;)
end
```

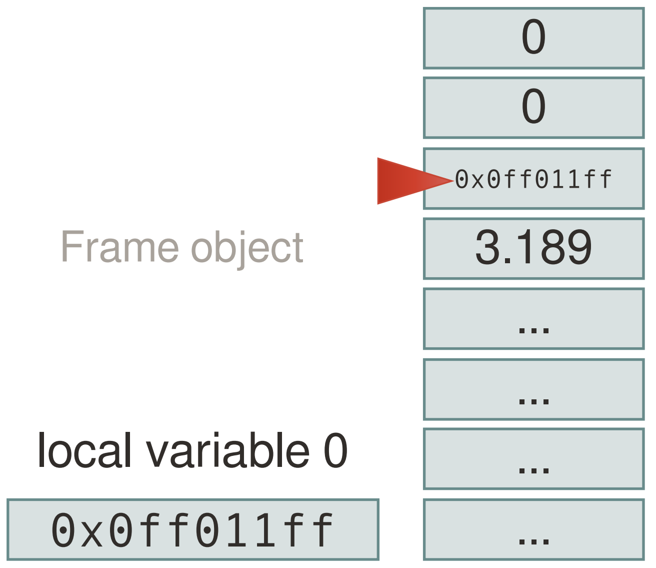
Resetting Truffle frame-slot values

GraalWasm

```
case LOCAL_GET:
  int value = getInt(frame, index);
  pushInt(frame, stackPointer, value);
```

C program

0.2126 * R + 0.7152 * G + ...



```
f64.convert_i32_s
f64.const 0x1.b367a0f9096bcp-3 (;=0.2126;)
f64.mul
local.get 0
i32.const 16
i32.shr_u
i32.const 255
i32.and
f64.convert_i32_s
f64.const 0x1.6e2eb1c432ca5p-1 (;=0.7152;)
f64.mul
```

WebAssembly program

```
loop  ;; label = @1
 local.get 2
 i32.const 2
 i32.shl
 i32.const 4194300
 i32.and
 local.tee 0
 i32.const 4196368
 i32.add
 i32.const -1
 i32.const 255
 local.get 0
 i32.const 2064
 i32.add
 i32.load
 local.tee 0
 i32.const 8
 i32.shr_u
 i32.const 255
 i32.and
 f64.convert_i32_s
 f64.const 0x1.27bb2fec56d5dp-4 (;=0.0722;)
 f64.mul
 local.get 0
 i32.const 24
 i32.shr_u
 f64.convert_i32_s
 f64.const 0x1.b367a0f9096bcp-3 (;=0.2126;)
 f64.mul
 local.get 0
 i32.const 16
 i32.shr_u
 i32.const 255
 i32.and
 f64.convert_i32_s
 f64.const 0x1.6e2eb1c432ca5p-1 (;=0.7152;)
 f64.mul
 f64.add
 f64.add
 f64.const 0x1.fcp+6 (;=127;)
 f64.gt
 local.tee 0
 select
 i32.store
 local.get 1
 local.get 0
 i32.const 1
 i32.xor
 i32.add
 local.set 1
 local.get 2
 i32.const 1
 i32.add
 local.tee 2
 i32.const 268435456
 i32.ne
 br_if 0 (;@1;)
end
```

Resetting Truffle frame-slot values

```
case I32_CONST:
  int value = peek4(instructions, offset);
  offset += 4;
  push(frame, stackPointer, value);
```

GraalWasm

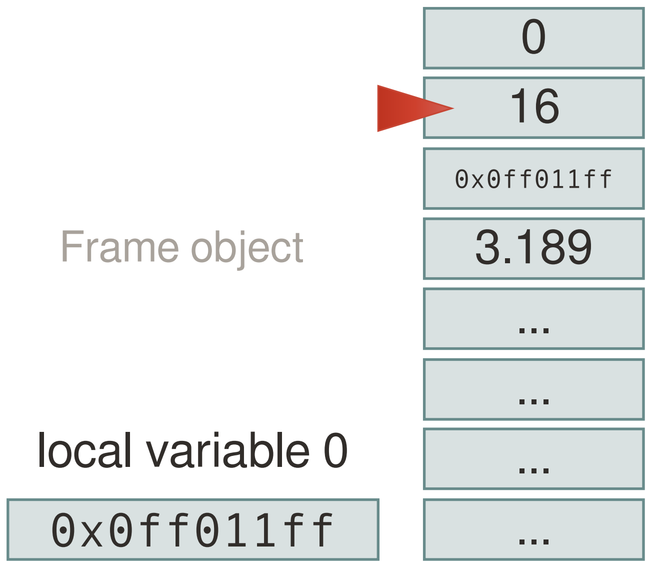
C program

```
G = (color & 0x00FF0000) >> 16;
```

0.2126 * R + 0.7152 * G + ...

```
f64.convert_i32_s
f64.const 0x1.b367a0f9096bcp-3 (;=0.2126;)
f64.mul
local.get 0
i32.const 16
i32.shr_u
i32.const 255
i32.and
f64.convert_i32_s
f64.const 0x1.6e2eb1c432ca5p-1 (;=0.7152;)
f64.mul
```

WebAssembly program



```
loop ;; label = @1
 local.get 2
 i32.const 2
 i32.shl
 i32.const 4194300
 i32.and
 local.tee 0
 i32.const 4196368
 i32.add
 i32.const -1
 i32.const 255
 local.get 0
 i32.const 2064
 i32.add
 i32.load
 local.tee 0
 i32.const 8
 i32.shr_u
 i32.const 255
 i32.and
 f64.convert_i32_s
 f64.const 0x1.27bb2fec56d5dp-4 (;=0.0722;)
 f64.mul
 local.get 0
 i32.const 24
 i32.shr_u
 f64.convert_i32_s
 f64.const 0x1.b367a0f9096bcp-3 (;=0.2126;)
 f64.mul
 local.get 0
 i32.const 16
 i32.shr_u
 i32.const 255
 i32.and
 f64.convert_i32_s
 f64.const 0x1.6e2eb1c432ca5p-1 (;=0.7152;)
 f64.mul
 f64.add
 f64.add
 f64.const 0x1.fcp+6 (;=127;)
 f64.gt
 local.tee 0
 select
 i32.store
 local.get 1
 local.get 0
 i32.const 1
 i32.xor
 i32.add
 local.set 1
 local.get 2
 i32.const 1
 i32.add
 local.tee 2
 i32.const 268435456
 i32.ne
 br_if 0 (;@1;)
end
```

Resetting Truffle frame-slot values

GraalWasm

```
case I32_SHR_U:
  int x = popInt(frame, stackPointer);
  int y = popInt(frame, stackPointer);
  int result = y >>> x;
  pushInt(frame, stackPointer, result);
```

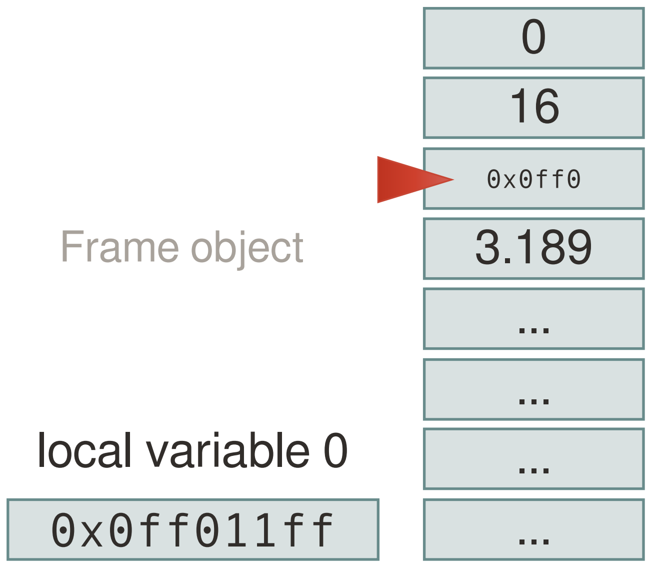
C program

```
G = (color & 0x00FF0000) >> 16;
```

0.2126 * R + 0.7152 * G + ...

WebAssembly program

```
f64.convert_i32_s
f64.const 0x1.b367a0f9096bcp-3 (;=0.2126;)
f64.mul
local.get 0
i32.const 16
i32.shr_u
i32.const 255
i32.and
f64.convert_i32_s
f64.const 0x1.6e2eb1c432ca5p-1 (;=0.7152;)
f64.mul
```



```
loop  ;; label = @1
 local.get 2
 i32.const 2
 i32.shl
 i32.const 4194300
 i32.and
 local.tee 0
 i32.const 4196368
 i32.add
 i32.const -1
 i32.const 255
 local.get 0
 i32.const 2064
 i32.add
 i32.load
 local.tee 0
 i32.const 8
 i32.shr_u
 i32.const 255
 i32.and
 f64.convert_i32_s
 f64.const 0x1.27bb2fec56d5dp-4 (;=0.0722;)
 f64.mul
 local.get 0
 i32.const 24
 i32.shr_u
 f64.convert_i32_s
 f64.const 0x1.b367a0f9096bcp-3 (;=0.2126;)
 f64.mul
 local.get 0
 i32.const 16
 i32.shr_u
 i32.const 255
 i32.and
 f64.convert_i32_s
 f64.const 0x1.6e2eb1c432ca5p-1 (;=0.7152;)
 f64.mul
 f64.add
 f64.add
 f64.const 0x1.fcp+6 (;=127;)
 f64.gt
 local.tee 0
 select
 i32.store
 local.get 1
 local.get 0
 i32.const 1
 i32.xor
 i32.add
 local.set 1
 local.get 2
 i32.const 1
 i32.add
 local.tee 2
 i32.const 268435456
 i32.ne
 br_if 0 (;@1;)
end
```

Resetting Truffle frame-slot values

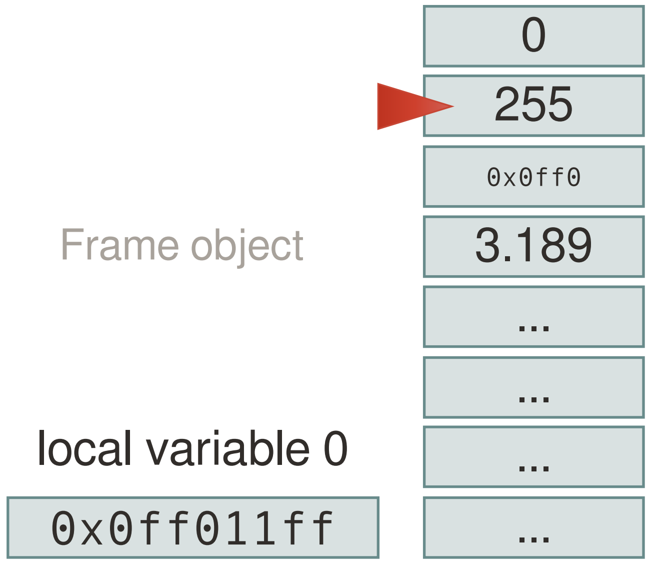
```
case I32_CONST:
  int value = peek4(instructions, offset);
  offset += 4;
  push(frame, stackPointer, value);
```

GraalWasm

C program

```
G = (color & 0x00FF0000) >> 16;
```

0.2126 * R + 0.7152 * G + ...



```
f64.convert_i32_s
f64.const 0x1.b367a0f9096bcp-3 (;=0.2126;)
f64.mul
local.get 0
i32.const 16
i32.shr_u
i32.const 255
i32.and
f64.convert_i32_s
f64.const 0x1.6e2eb1c432ca5p-1 (;=0.7152;)
f64.mul
```

WebAssembly program

```
loop  ;; label = @1
  local.get 2
  i32.const 2
  i32.shl
  i32.const 4194300
  i32.and
  local.tee 0
  i32.const 4196368
  i32.add
  i32.const -1
  i32.const 255
  local.get 0
  i32.const 2064
  i32.add
  i32.load
  local.tee 0
  i32.const 8
  i32.shr_u
  i32.const 255
  i32.and
  f64.convert_i32_s
  f64.const 0x1.27bb2fec56d5dp-4 (;=0.0722;)
  f64.mul
  local.get 0
  i32.const 24
  i32.shr_u
  f64.convert_i32_s
  f64.const 0x1.b367a0f9096bcp-3 (;=0.2126;)
  f64.mul
  local.get 0
  i32.const 16
  i32.shr_u
  i32.const 255
  i32.and
  f64.convert_i32_s
  f64.const 0x1.6e2eb1c432ca5p-1 (;=0.7152;)
  f64.mul
  f64.add
  f64.add
  f64.const 0x1.fcp+6 (;=127;)
  f64.gt
  local.tee 0
  select
  i32.store
  local.get 1
  local.get 0
  i32.const 1
  i32.xor
  i32.add
  local.set 1
  local.get 2
  i32.const 1
  i32.add
  local.tee 2
  i32.const 268435456
  i32.ne
  br_if 0 (;@1;)
end
```

Resetting Truffle frame-slot values

```
case I32_AND:
  int x = popInt(frame, stackPointer);
  int y = popInt(frame, stackPointer);
  int result = y & x;
  pushInt(frame, stackPointer, result);
```

GraalWasm

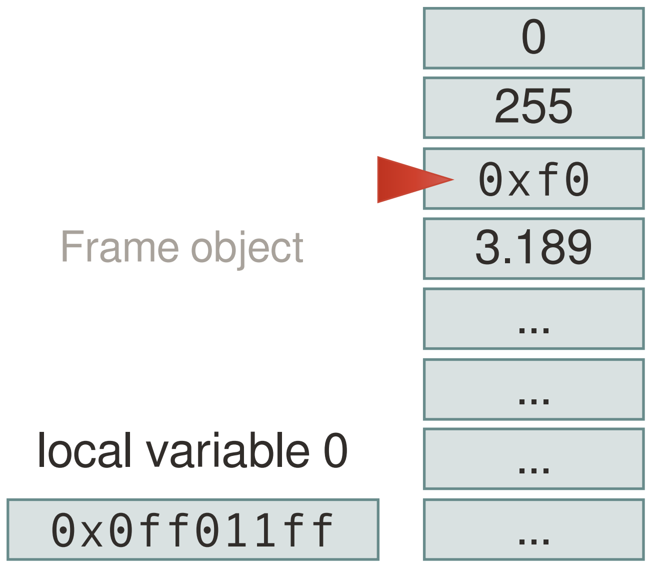
C program

```
G = (color & 0x00FF0000) >> 16;
```

0.2126 * R + 0.7152 * G + ...

```
f64.convert_i32_s
f64.const 0x1.b367a0f9096bcp-3 (;=0.2126;)
f64.mul
local.get 0
i32.const 16
i32.shr_u
i32.const 255
i32.and
f64.convert_i32_s
f64.const 0x1.6e2eb1c432ca5p-1 (;=0.7152;)
f64.mul
```

WebAssembly program

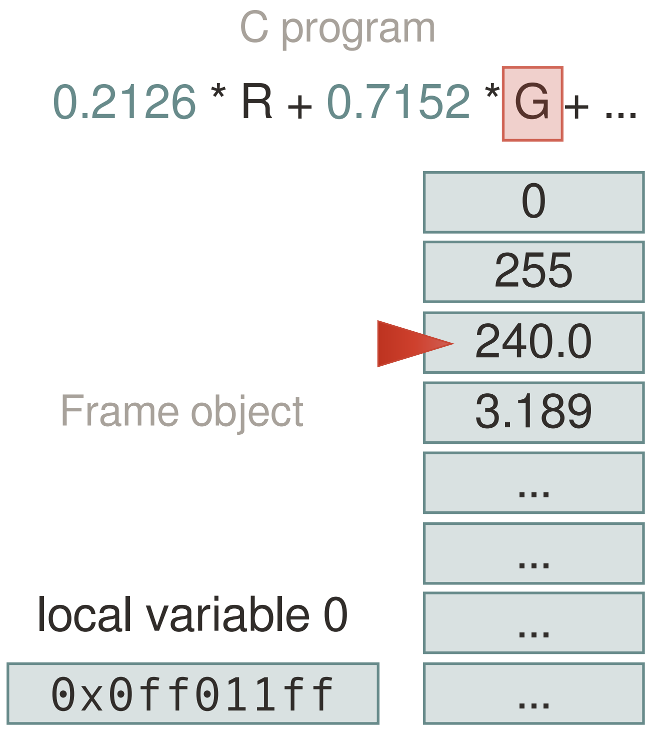


```
loop  ;; label = @1
 local.get 2
 i32.const 2
 i32.shl
 i32.const 4194300
 i32.and
 local.tee 0
 i32.const 4196368
 i32.add
 i32.const -1
 i32.const 255
 local.get 0
 i32.const 2064
 i32.add
 i32.load
 local.tee 0
 i32.const 8
 i32.shr_u
 i32.const 255
 i32.and
 f64.convert_i32_s
 f64.const 0x1.27bb2fec56d5dp-4 (;=0.0722;)
 f64.mul
 local.get 0
 i32.const 24
 i32.shr_u
 f64.convert_i32_s
 f64.const 0x1.b367a0f9096bcp-3 (;=0.2126;)
 f64.mul
 local.get 0
 i32.const 16
 i32.shr_u
 i32.const 255
 i32.and
 f64.convert_i32_s
 f64.const 0x1.6e2eb1c432ca5p-1 (;=0.7152;)
 f64.mul
 f64.add
 f64.add
 f64.const 0x1.fcp+6 (;=127;)
 f64.gt
 local.tee 0
 select
 i32.store
 local.get 1
 local.get 0
 i32.const 1
 i32.xor
 i32.add
 local.set 1
 local.get 2
 i32.const 1
 i32.add
 local.tee 2
 i32.const 268435456
 i32.ne
 br_if 0 (;@1;)
end
```

Resetting Truffle frame-slot values

GraalWasm

```
case F64_CONVERT_I64_S:
case F64_CONVERT_I64_U:
  long x = pop(frame, stackPointer);
  double result = x;
  pushDouble(frame, stackPointer, result);
```



WebAssembly program

```
f64.convert_i32_s
f64.const 0x1.b367a0f9096bcp-3 (;=0.2126;)
f64.mul
local.get 0
i32.const 16
i32.shr_u
i32.const 255
i32.and
f64.convert_i32_s
f64.const 0x1.6e2eb1c432ca5p-1 (;=0.7152;)
f64.mul
```

```
loop  ;; label = @1
 local.get 2
 i32.const 2
 i32.shl
 i32.const 4194300
 i32.and
 local.tee 0
 i32.const 4196368
 i32.add
 i32.const -1
 i32.const 255
 local.get 0
 i32.const 2064
 i32.add
 i32.load
 local.tee 0
 i32.const 8
 i32.shr_u
 i32.const 255
 i32.and
 f64.convert_i32_s
 f64.const 0x1.27bb2fec56d5dp-4 (;=0.0722;)
 f64.mul
 local.get 0
 i32.const 24
 i32.shr_u
 f64.convert_i32_s
 f64.const 0x1.b367a0f9096bcp-3 (;=0.2126;)
 f64.mul
 local.get 0
 i32.const 16
 i32.shr_u
 i32.const 255
 i32.and
 f64.convert_i32_s
 f64.const 0x1.6e2eb1c432ca5p-1 (;=0.7152;)
 f64.mul
 f64.add
 f64.add
 f64.const 0x1.fcp+6 (;=127;)
 f64.gt
 local.tee 0
 select
 i32.store
 local.get 1
 local.get 0
 i32.const 1
 i32.xor
 i32.add
 local.set 1
 local.get 2
 i32.const 1
 i32.add
 local.tee 2
 i32.const 268435456
 i32.ne
 br_if 0 (;@1;)
end
```

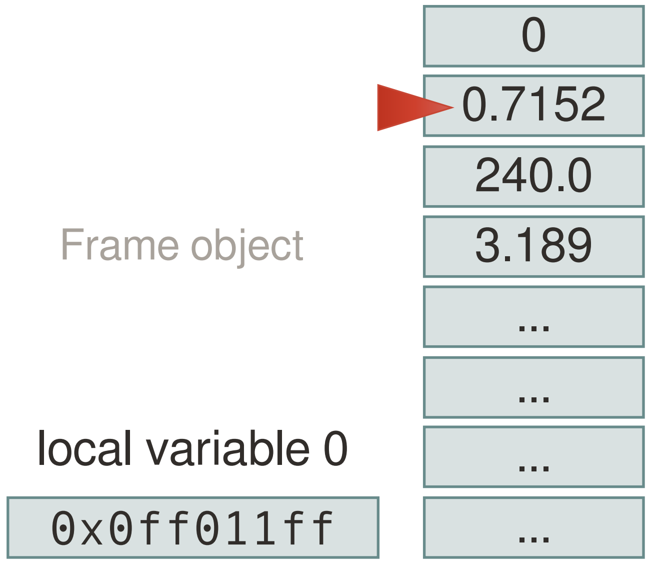
Resetting Truffle frame-slot values

```
case F64_CONST:
  long value = peek8(instructions, offset);
  offset += 8;
  push(frame, stackPointer, value);
```

GraalWasm

C program

0.2126 * R + 0.7152 * G + ...



```
f64.convert_i32_s
f64.const 0x1.b367a0f9096bcp-3 (;=0.2126;)
f64.mul
local.get 0
i32.const 16
i32.shr_u
i32.const 255
i32.and
f64.convert_i32_s
f64.const 0x1.6e2eb1c432ca5p-1 (;=0.7152;)
f64.mul
```

WebAssembly program

```
loop  ;; label = @1
 local.get 2
 i32.const 2
 i32.shl
 i32.const 4194300
 i32.and
 local.tee 0
 i32.const 4196368
 i32.add
 i32.const -1
 i32.const 255
 local.get 0
 i32.const 2064
 i32.add
 i32.load
 local.tee 0
 i32.const 8
 i32.shr_u
 i32.const 255
 i32.and
 f64.convert_i32_s
 f64.const 0x1.27bb2fec56d5dp-4 (;=0.0722;)
 f64.mul
 local.get 0
 i32.const 24
 i32.shr_u
 f64.convert_i32_s
 f64.const 0x1.b367a0f9096bcp-3 (;=0.2126;)
 f64.mul
 local.get 0
 i32.const 16
 i32.shr_u
 i32.const 255
 i32.and
 f64.convert_i32_s
 f64.const 0x1.6e2eb1c432ca5p-1 (;=0.7152;)
 f64.mul
 f64.add
 f64.add
 f64.const 0x1.fcp+6 (;=127;)
 f64.gt
 local.tee 0
 select
 i32.store
 local.get 1
 local.get 0
 i32.const 1
 i32.xor
 i32.add
 local.set 1
 local.get 2
 i32.const 1
 i32.add
 local.tee 2
 i32.const 268435456
 i32.ne
 br_if 0 (;@1;)
end
```

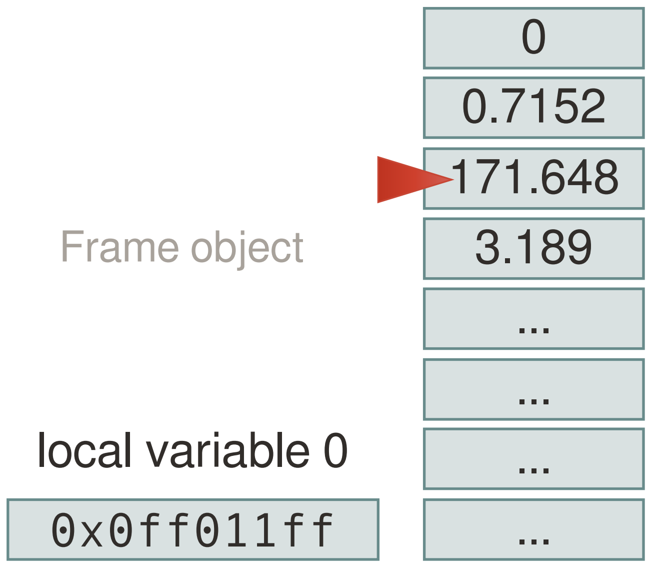
Resetting Truffle frame-slot values

GraalWasm

```
case F64_MUL:
  double x = popDouble(frame, stackPointer);
  double y = popDouble(frame, stackPointer);
  double result = y * x;
  pushDouble(frame, stackPointer, result);
```

C program

0.2126 * R + 0.7152 * G + ...



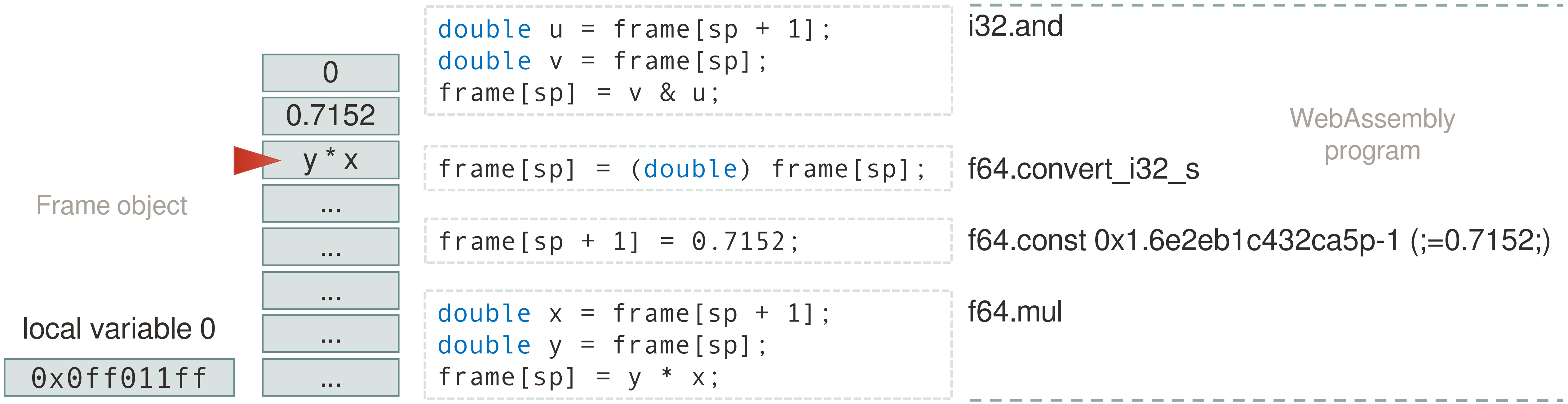
```
f64.convert_i32_s
f64.const 0x1.b367a0f9096bcp-3 (;=0.2126;)
f64.mul
local.get 0
i32.const 16
i32.shr_u
i32.const 255
i32.and
f64.convert_i32_s
f64.const 0x1.6e2eb1c432ca5p-1 (;=0.7152;)
f64.mul
f64.add
```

WebAssembly program

```
loop  ;; label = @1
  local.get 2
  i32.const 2
  i32.shl
  i32.const 4194300
  i32.and
  local.tee 0
  i32.const 4196368
  i32.add
  i32.const -1
  i32.const 255
  local.get 0
  i32.const 2064
  i32.add
  i32.load
  local.tee 0
  i32.const 8
  i32.shr_u
  i32.const 255
  i32.and
  f64.convert_i32_s
  f64.const 0x1.27bb2fec56d5dp-4 (;=0.0722;)
  f64.mul
  local.get 0
  i32.const 24
  i32.shr_u
  f64.convert_i32_s
  f64.const 0x1.b367a0f9096bcp-3 (;=0.2126;)
  f64.mul
  local.get 0
  i32.const 16
  i32.shr_u
  i32.const 255
  i32.and
  f64.convert_i32_s
  f64.const 0x1.6e2eb1c432ca5p-1 (;=0.7152;)
  f64.mul
  f64.add
  f64.add
  f64.const 0x1.fcp+6 (;=127;)
  f64.gt
  local.tee 0
  select
  i32.store
  local.get 1
  local.get 0
  i32.const 1
  i32.xor
  i32.add
  local.set 1
  local.get 2
  i32.const 1
  i32.add
  local.tee 2
  i32.const 268435456
  i32.ne
  br_if 0 (;@1;)
end
```

Resetting Truffle frame-slot values

The partial evaluator stitches these interpreter snippets together.



```
double u = frame[sp + 1];
double v = frame[sp];
frame[sp] = v & u;
```

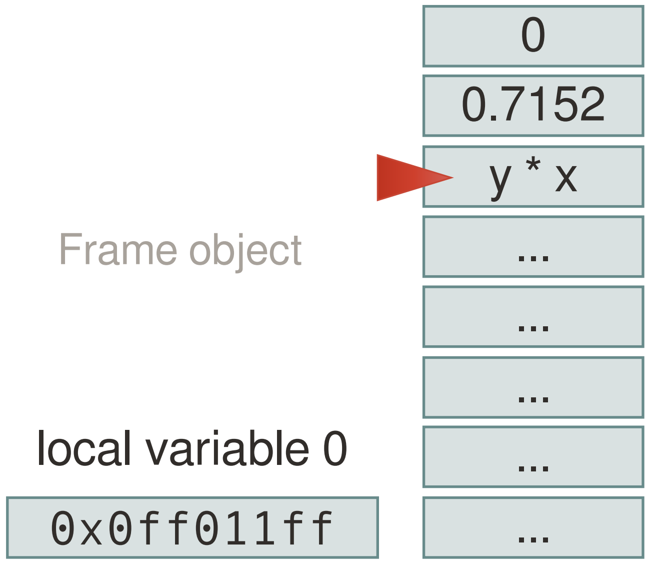
```
frame[sp] = (double) frame[sp];
```

```
frame[sp + 1] = 0.7152;
```

```
double x = frame[sp + 1];
double y = frame[sp];
frame[sp] = y * x;
```

Resetting Truffle frame-slot values

Reads and writes on the frame slots are eliminated during escape analysis.



```
double u = frame[sp + 1];
double v = frame[sp];
frame[sp] = v & u;
```

```
frame[sp] = (double) frame[sp];
```

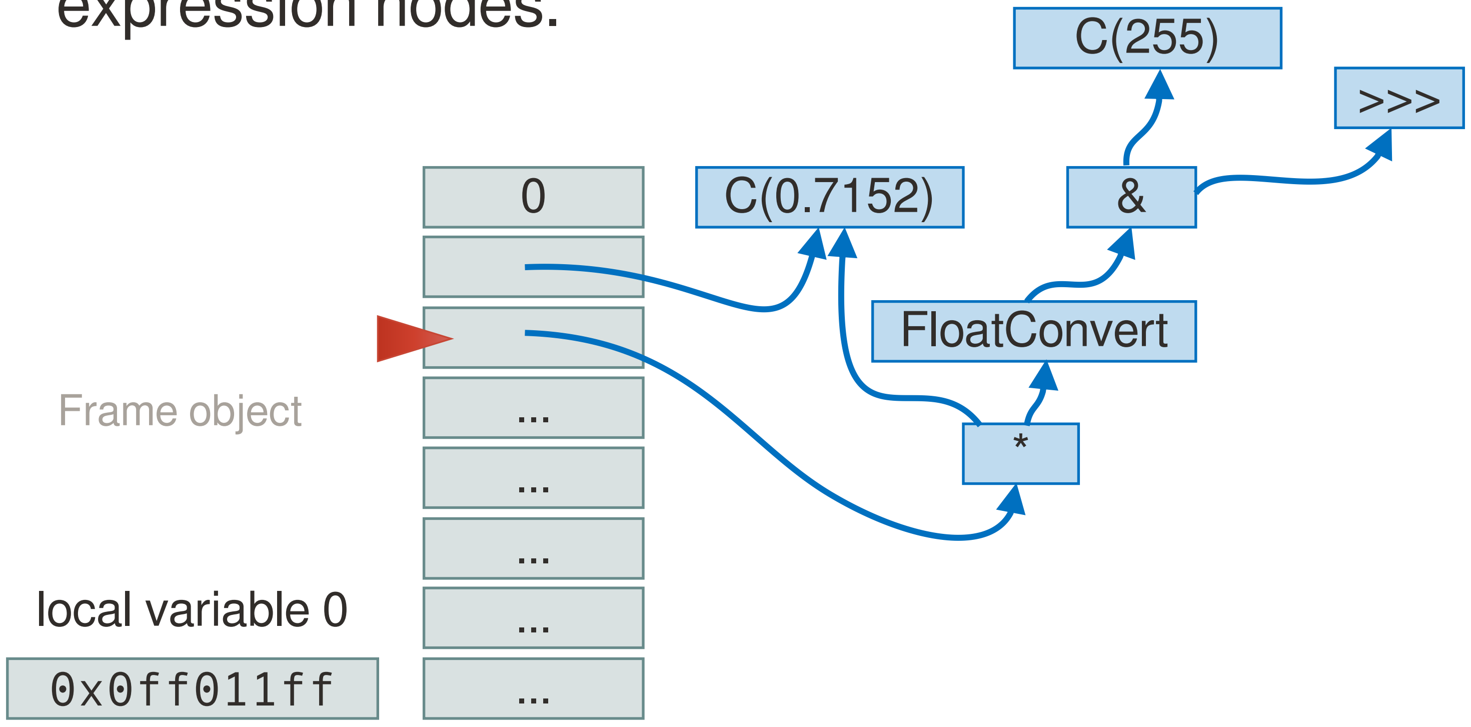
```
frame[sp + 1] = 0.7152;
```

```
double x = 0.7152;
double y = (double) (v & u);
frame[sp] = y * x;
```

Values are propagated through the reads and the writes.

Resetting Truffle frame-slot values

From the compiler's standpoint, the contents of the Frame object are various expression nodes.



```
double u = frame[sp + 1];
double v = frame[sp];
frame[sp] = v & u;
```

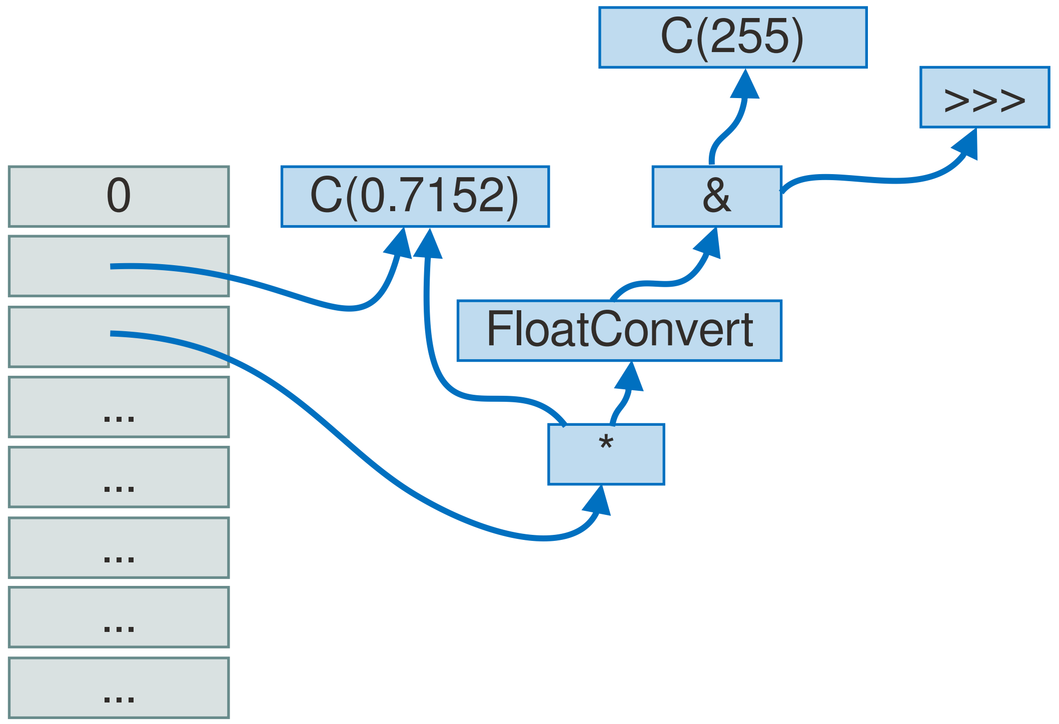
```
frame[sp] = (double) frame[sp];
```

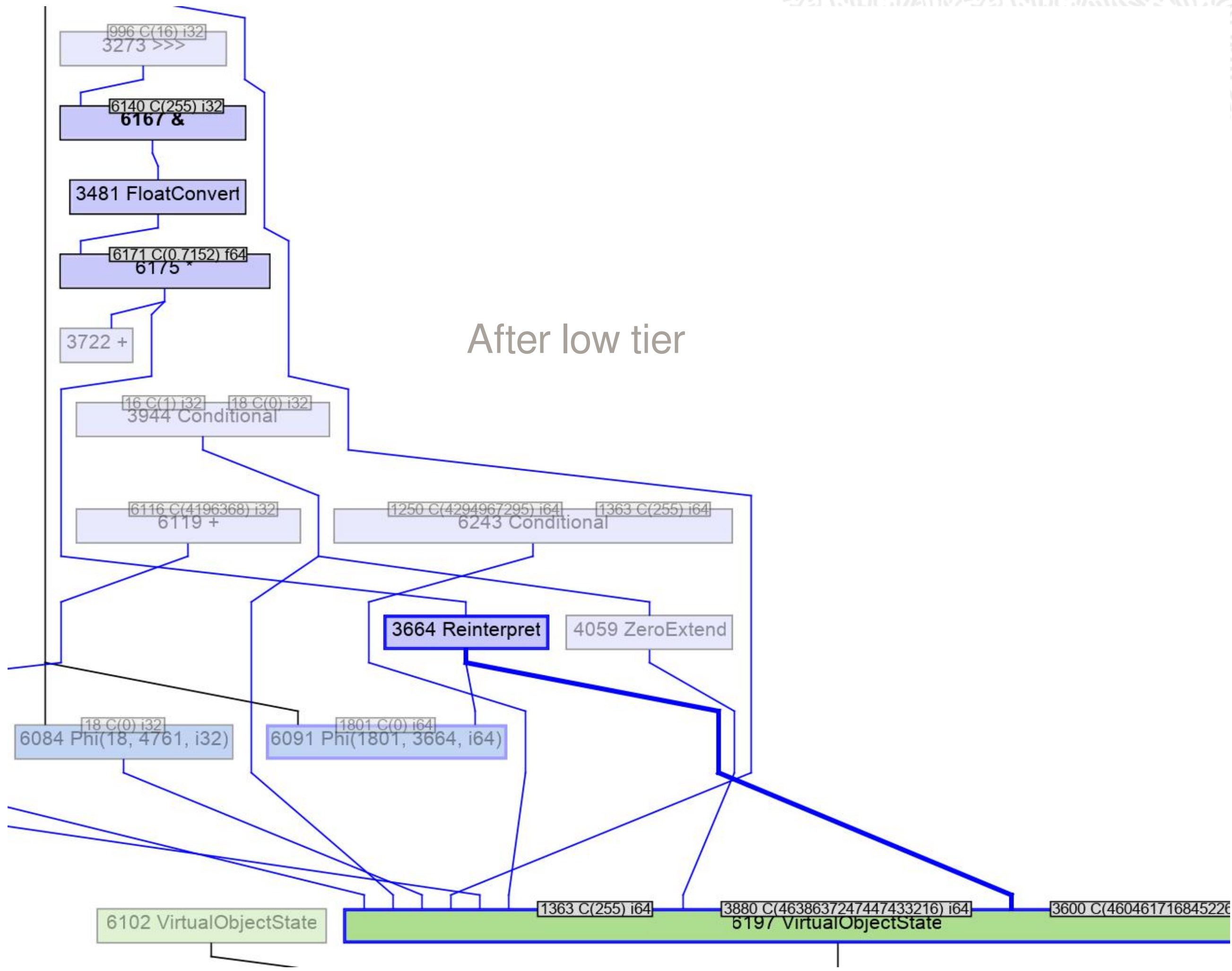
```
frame[sp + 1] = 0.7152;
```

```
double x = 0.7152;
double y = (double) (v & u);
frame[sp] = y * x;
```

Resetting Truffle frame-slot values

This is exactly what we see in the IR.





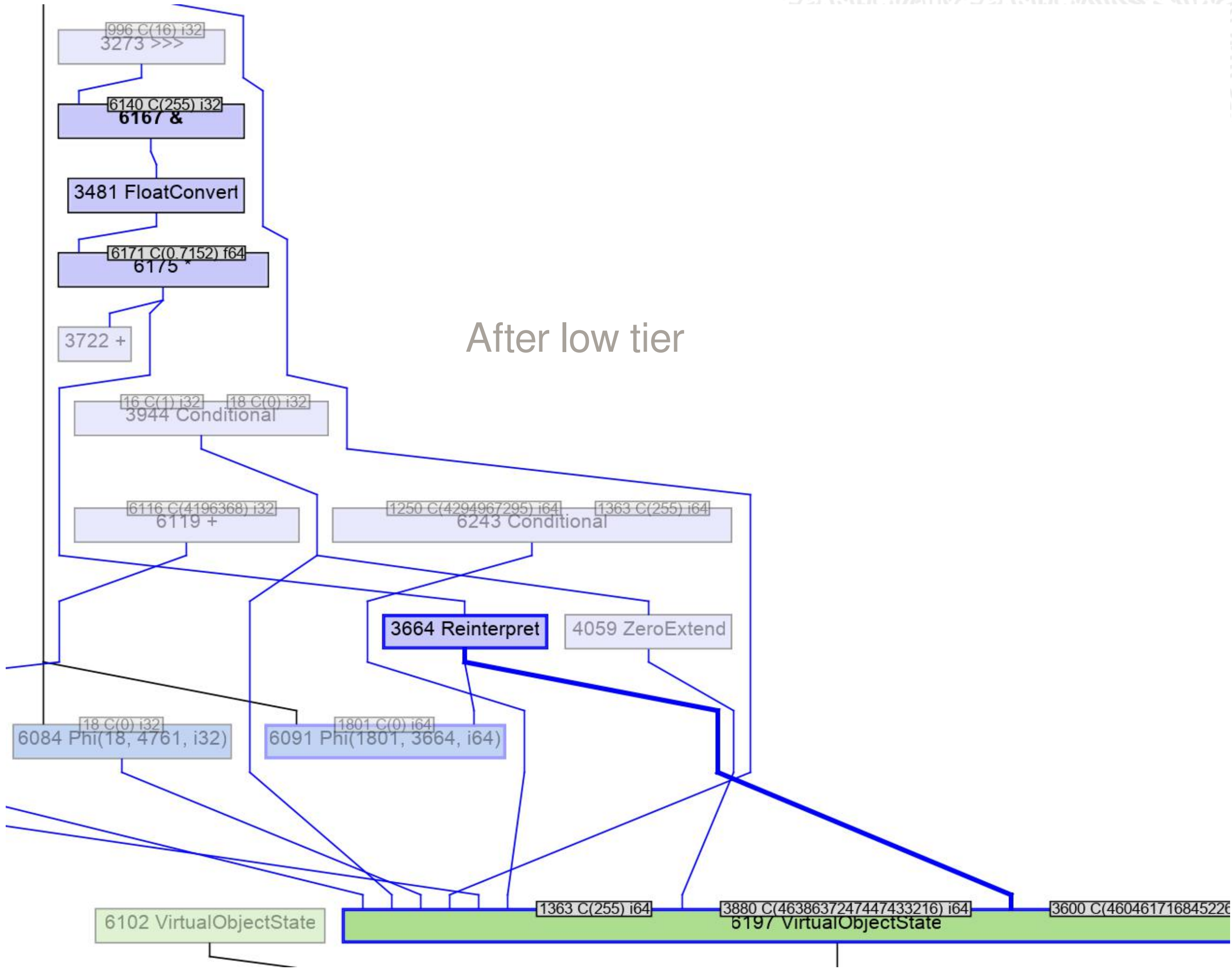
After low tier

Resetting Truffle frame-slot values

How do we prevent this?

Set the Frame slots to their default values after the value is no longer needed, because the virtual object state does not need to track default values.

```
long pop(VirtualFrame frame, int slot) {
  long result = frame.getLong(slot);
  return result;
}
```

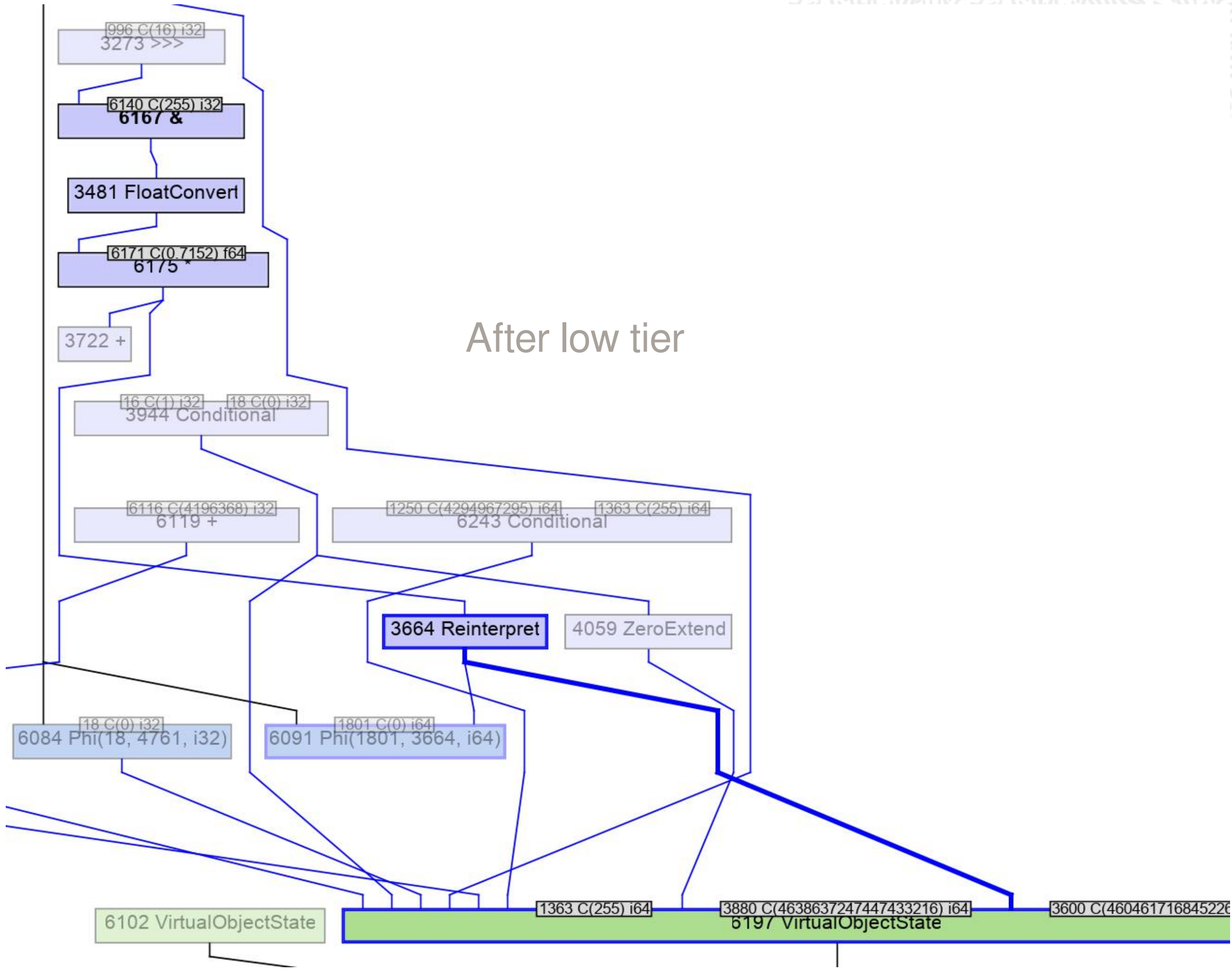


Resetting Truffle frame-slot values

How do we prevent this?

Set the Frame slots to their default values after the value is no longer needed, because the virtual object state does not need to track default values.

```
long pop(VirtualFrame frame, int slot) {
  long result = frame.getLong(slot);
  frame.setLong(slot, 0L);
  return result;
}
```

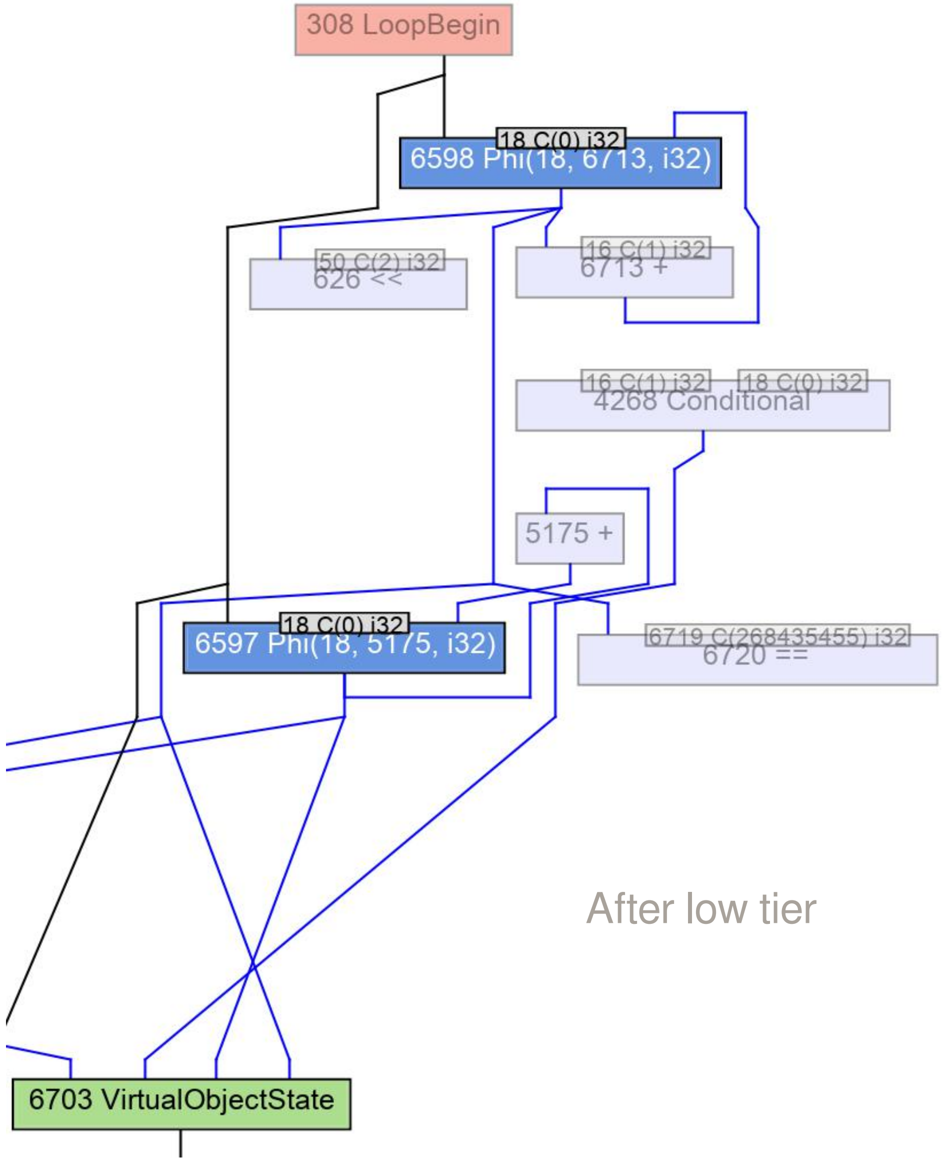


Resetting Truffle frame-slot values

How do we prevent this?

Set the Frame slots to their default values after the value is no longer needed, because the virtual object state does not need to track default values.

```
long pop(VirtualFrame frame, int slot) {
  long result = frame.getLong(slot);
  frame.setLong(slot, 0L);
  return result;
}
```



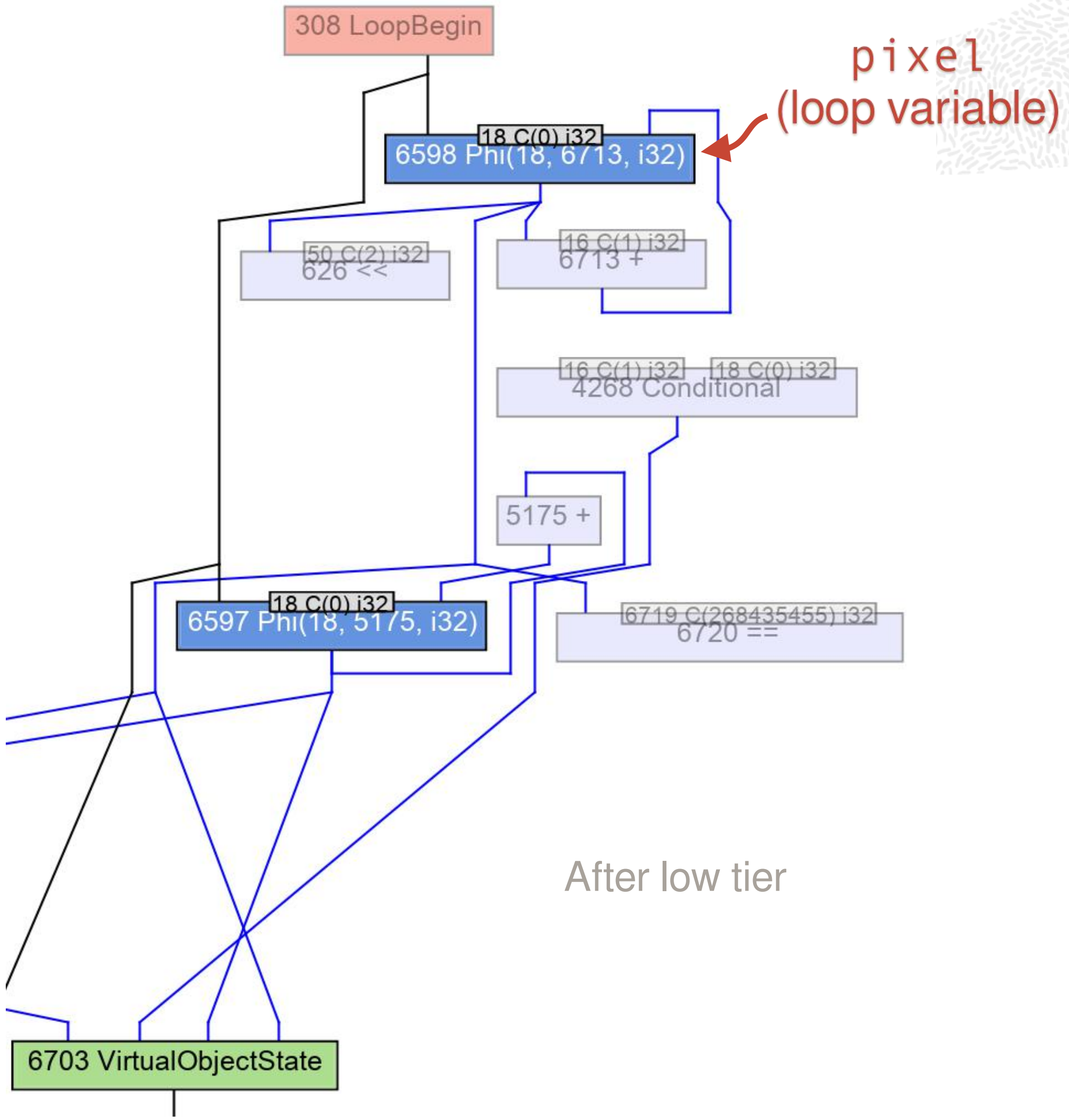
After low tier

Resetting Truffle frame-slot values

How do we prevent this?

Set the Frame slots to their default values after the value is no longer needed, because the virtual object state does not need to track default values.

```
long pop(VirtualFrame frame, int slot) {
  long result = frame.getLong(slot);
  frame.setLong(slot, 0L);
  return result;
}
```



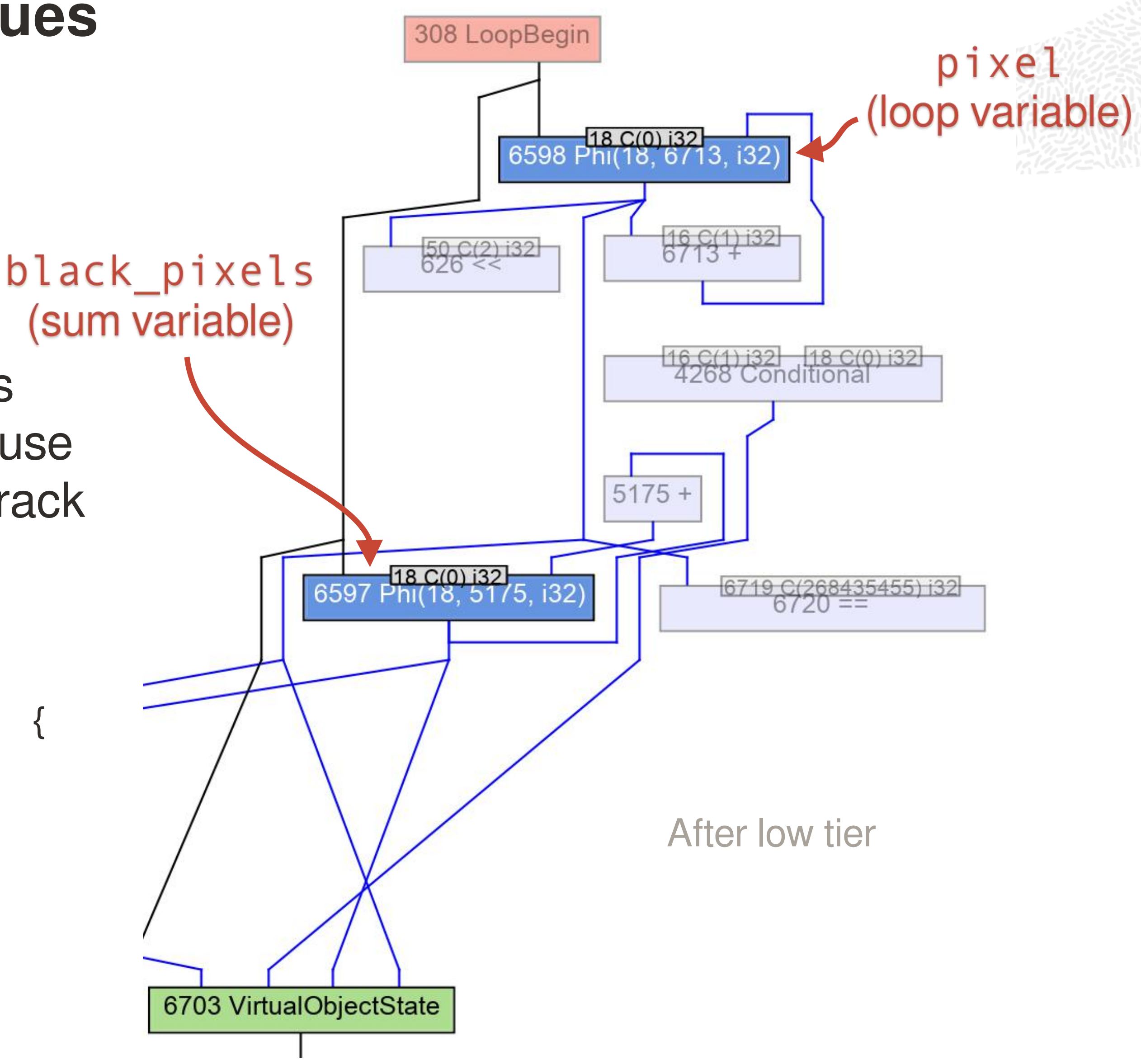
After low tier

Resetting Truffle frame-slot values

How do we prevent this?

Set the Frame slots to their default values after the value is no longer needed, because the virtual object state does not need to track default values.

```
long pop(VirtualFrame frame, int slot) {
  long result = frame.getLong(slot);
  frame.setLong(slot, 0L);
  return result;
}
```

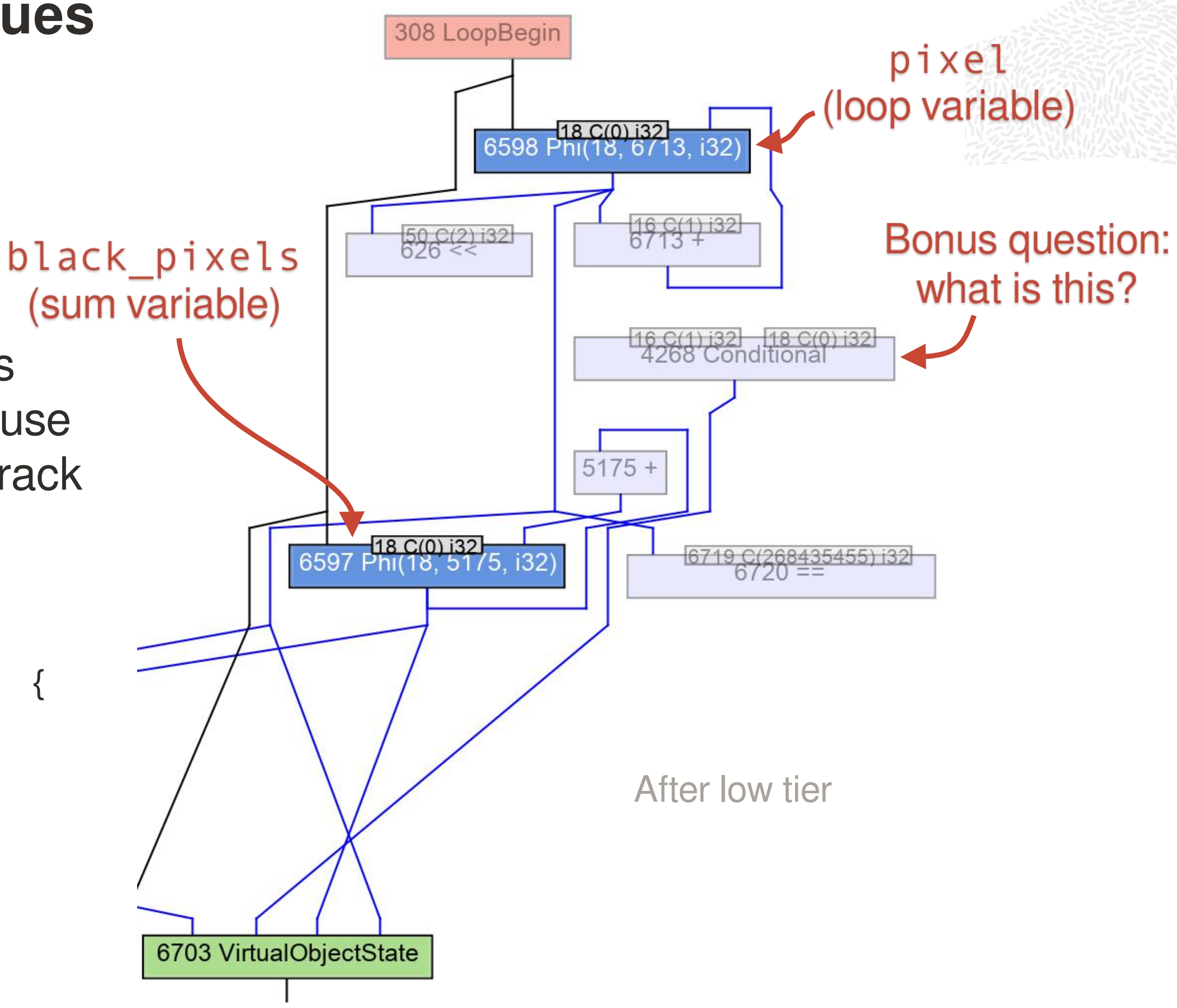


Resetting Truffle frame-slot values

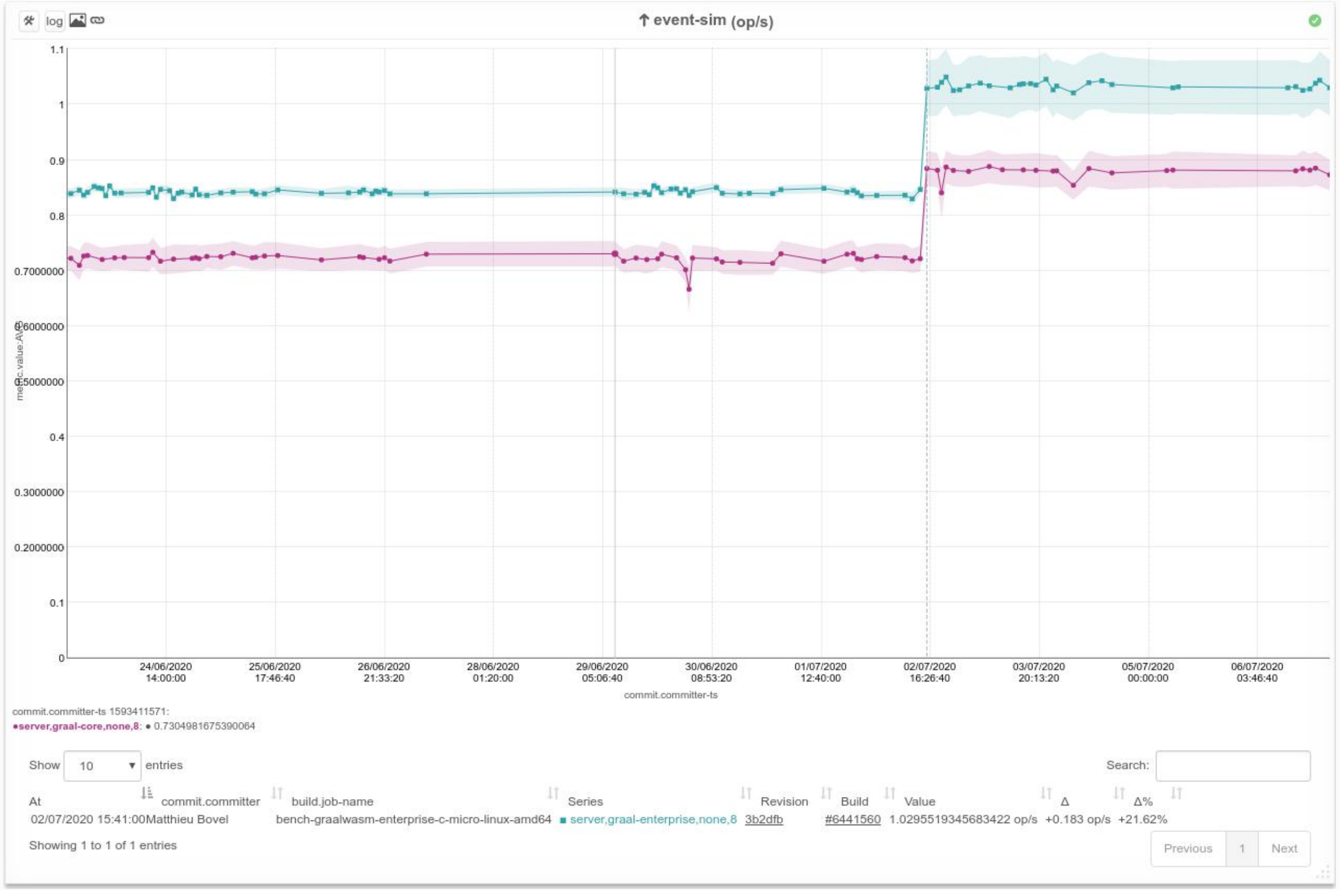
How do we prevent this?

Set the Frame slots to their default values after the value is no longer needed, because the virtual object state does not need to track default values.

```
long pop(VirtualFrame frame, int slot) {
  long result = frame.getLong(slot);
  frame.setLong(slot, 0L);
  return result;
}
```



Resetting Truffle frame-slot values



commit.committer-ts 1593411571:
●server,graal-core,none,8: ● 0.7304981675390064

Show 10 entries Search:

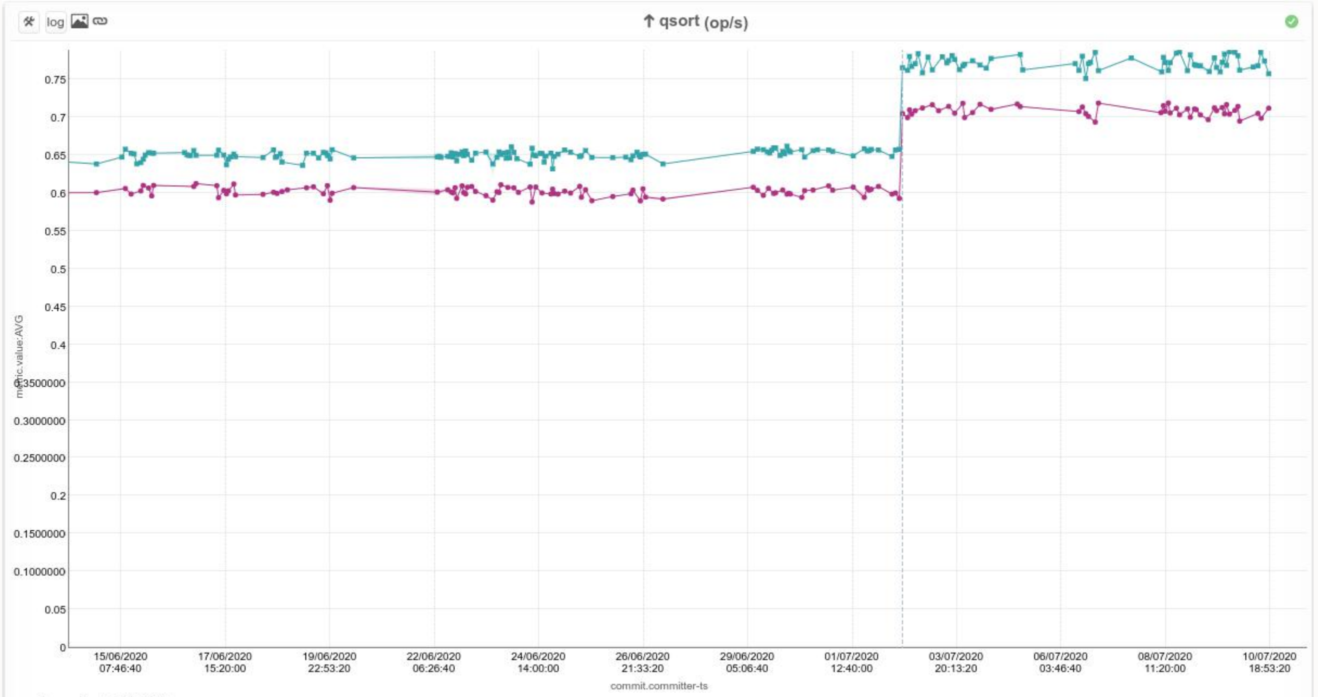
At	commit.committer	build.job-name	Series	Revision	Build	Value	Δ	Δ%
02/07/2020 15:41:00	Matthieu Bovel	bench-graalwasm-enterprise-c-micro-linux-amd64	■ server,graal-enterprise,none,8	3b2dfb	#6441560	1.0295519345683422 op/s	+0.183 op/s	+21.62%

Showing 1 to 1 of 1 entries

Previous 1 Next

- 21% improvement on a discrete-event simulation benchmark (*event-sim*)

Resetting Truffle frame-slot values



commit.committer-ts 1592011674:
■server,graal-enterprise,none,8: ■ 0.6449945761351136

Show 10 ▾ entries Search:

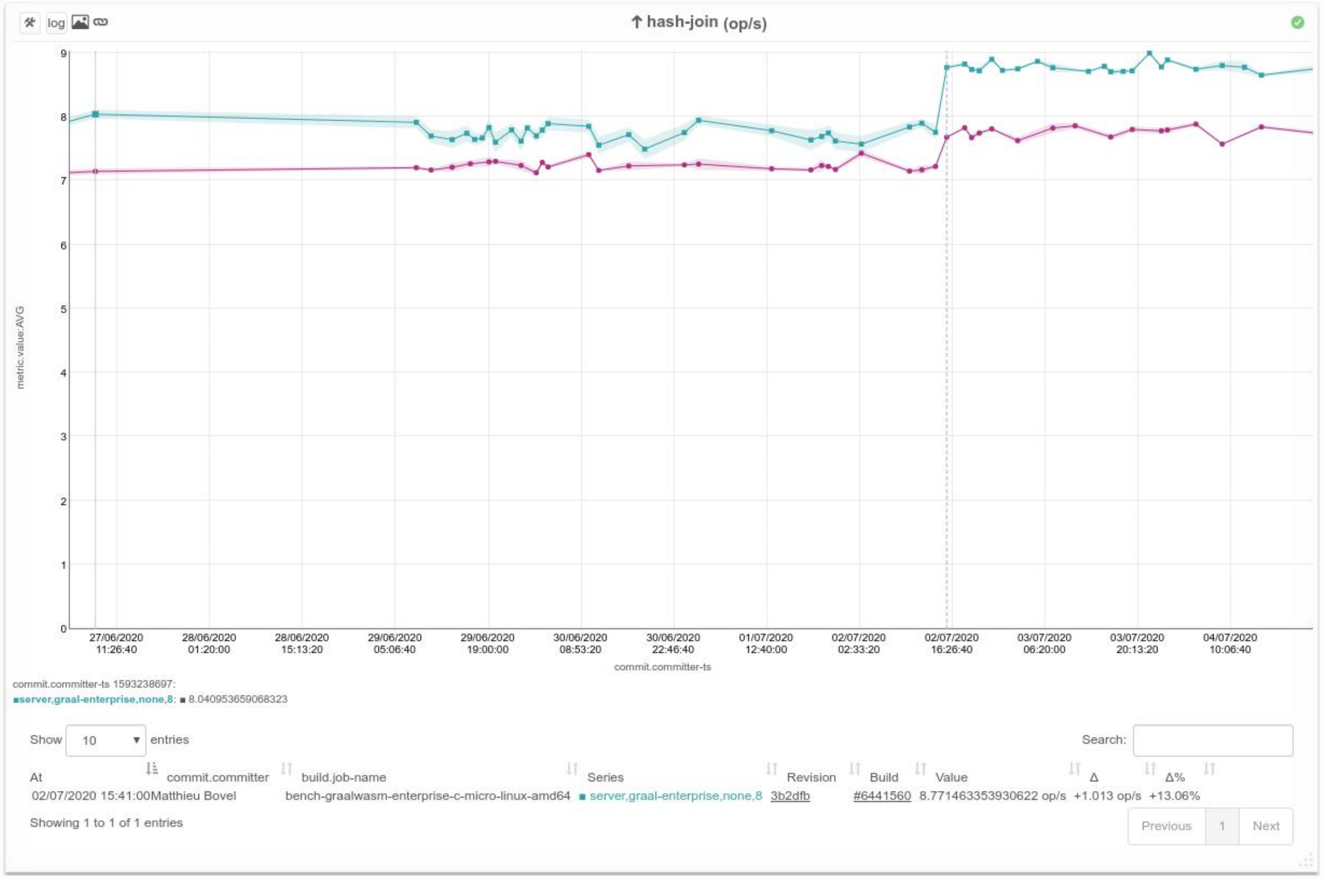
At	commit.committer	build.job-name	Series	Revision	Build	Value	Δ	Δ%
02/07/2020 15:41:00	Matthieu Bovel	bench-graalwasm-enterprise-c-micro-linux-amd64	■ server,graal-enterprise,none,8	3b2dfb	#6441560	0.7654390170905648 op/s	+0.108 op/s	+16.42%

Showing 1 to 1 of 1 entries

Previous 1 Next

- 21% improvement on a discrete-event simulation benchmark (*event-sim*)
- 16% improvement on the quicksort benchmark (*qsort*)

Resetting Truffle frame-slot values



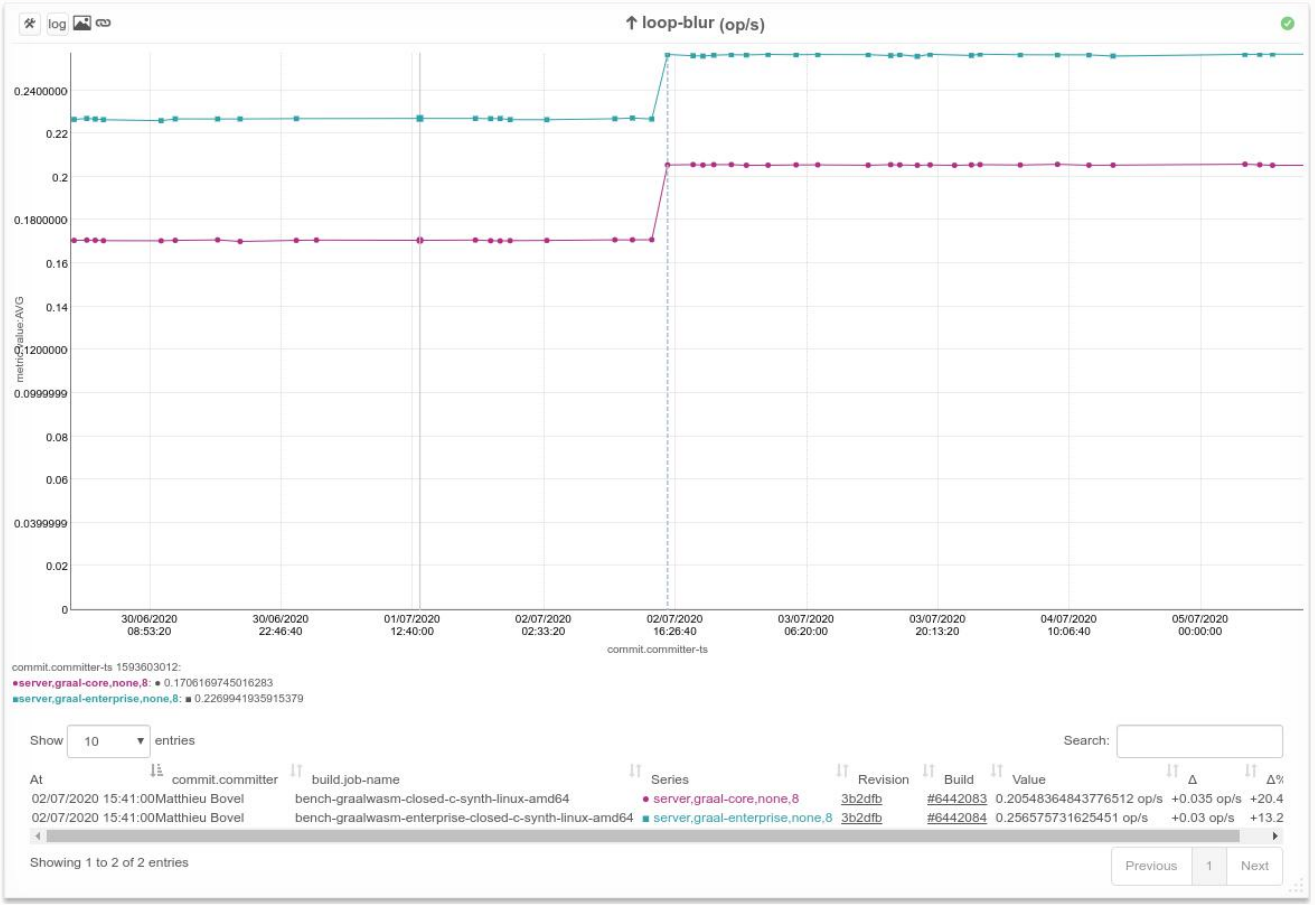
commit.committer-ts 1593238697:
■server,graal-enterprise,none,8: ■ 8.040953659068323

At	commit.committer	build.job-name	Series	Revision	Build	Value	Δ	Δ%
02/07/2020 15:41:00	Matthieu Bovel	bench-graalwasm-enterprise-c-micro-linux-amd64	■ server,graal-enterprise,none,8	3b2dfb	#6441560	8.771463353930622 op/s	+1.013 op/s	+13.06%

Showing 1 to 1 of 1 entries

- 21% improvement on a discrete-event simulation benchmark (*event-sim*)
- 16% improvement on the quicksort benchmark (*qsort*)
- 13% improvement on a hash-join benchmark (*hash-join*)

Resetting Truffle frame-slot values



commit.committer-ts 1593603012:
●server,graal-core,none,8: ● 0.1706169745016283
■server,graal-enterprise,none,8: ■ 0.2269941935915379

Show 10 entries Search:

At	commit.committer	build.job-name	Series	Revision	Build	Value	Δ	Δ%
02/07/2020 15:41:00	Matthieu Bovel	bench-graalwasm-closed-c-synth-linux-amd64	● server,graal-core,none,8	3b2dfb	#6442083	0.20548364843776512 op/s	+0.035 op/s	+20.4
02/07/2020 15:41:00	Matthieu Bovel	bench-graalwasm-enterprise-closed-c-synth-linux-amd64	■ server,graal-enterprise,none,8	3b2dfb	#6442084	0.256575731625451 op/s	+0.03 op/s	+13.2

Showing 1 to 2 of 2 entries

Previous 1 Next

- 21% improvement on a discrete-event simulation benchmark (*event-sim*)
- 16% improvement on the quicksort benchmark (*qsort*)
- 13% improvement on a hash-join benchmark (*hash-join*)
- 20% improvement on an image-processing microbenchmark (*loop-blur*)

We’ve only scratched the surface.

Internships

in all areas of GraalVM

Internship Program

Our Research Center Locations

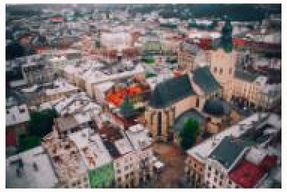
Zurich, Switzerland

Linz, Austria

California, USA

Prague, Czech Republic

Brno, Czech Republic


Lviv, Ukraine

Casablanca, Morocco

Belgrade, Serbia

https://www.graalvm.org/community/internship/

Thank you!

Questions?