A *Compact* Bytecode Format for **JavaScriptCore**

Tadeu Zagallo
Apple Inc.

webkit.org

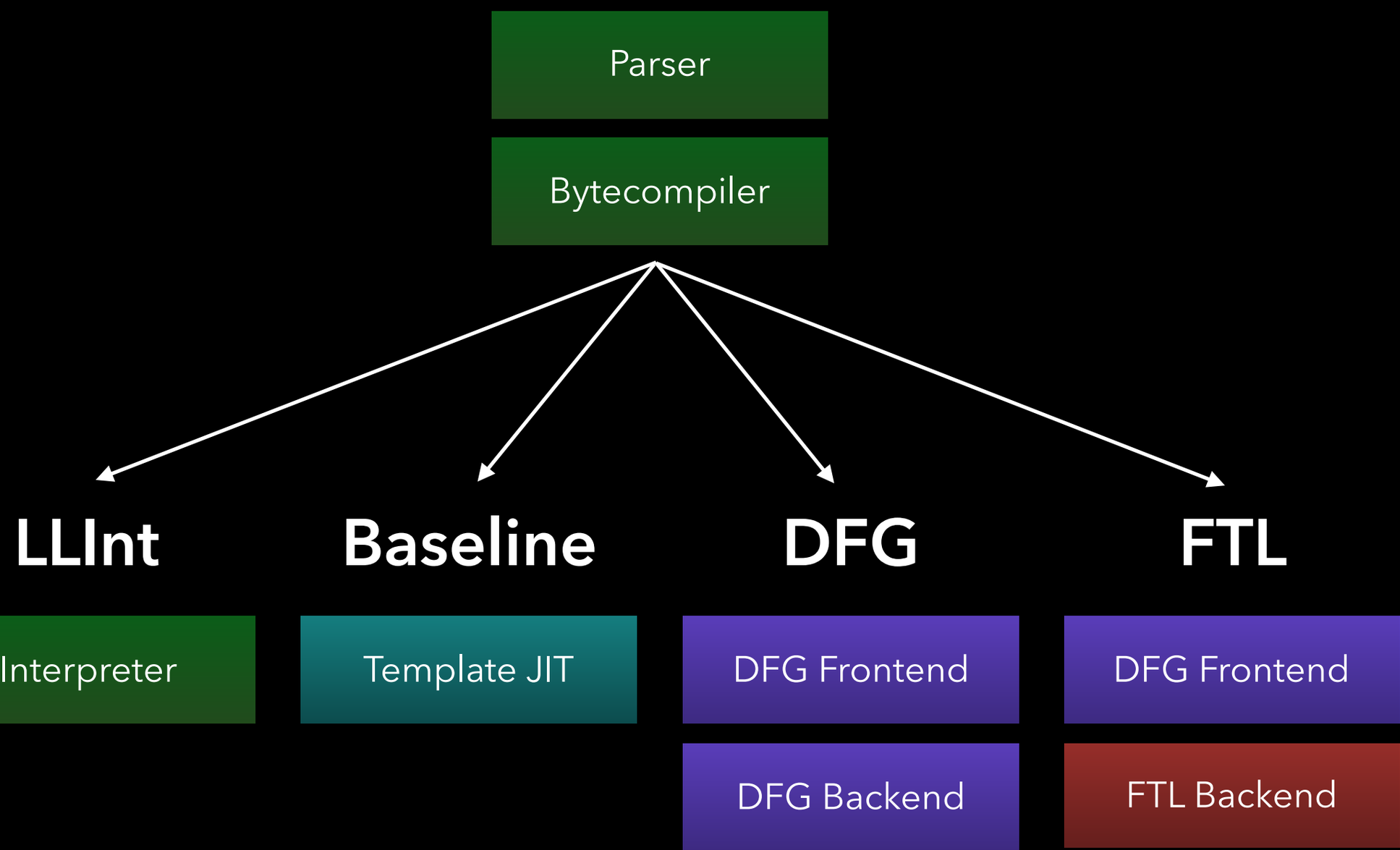
Safari

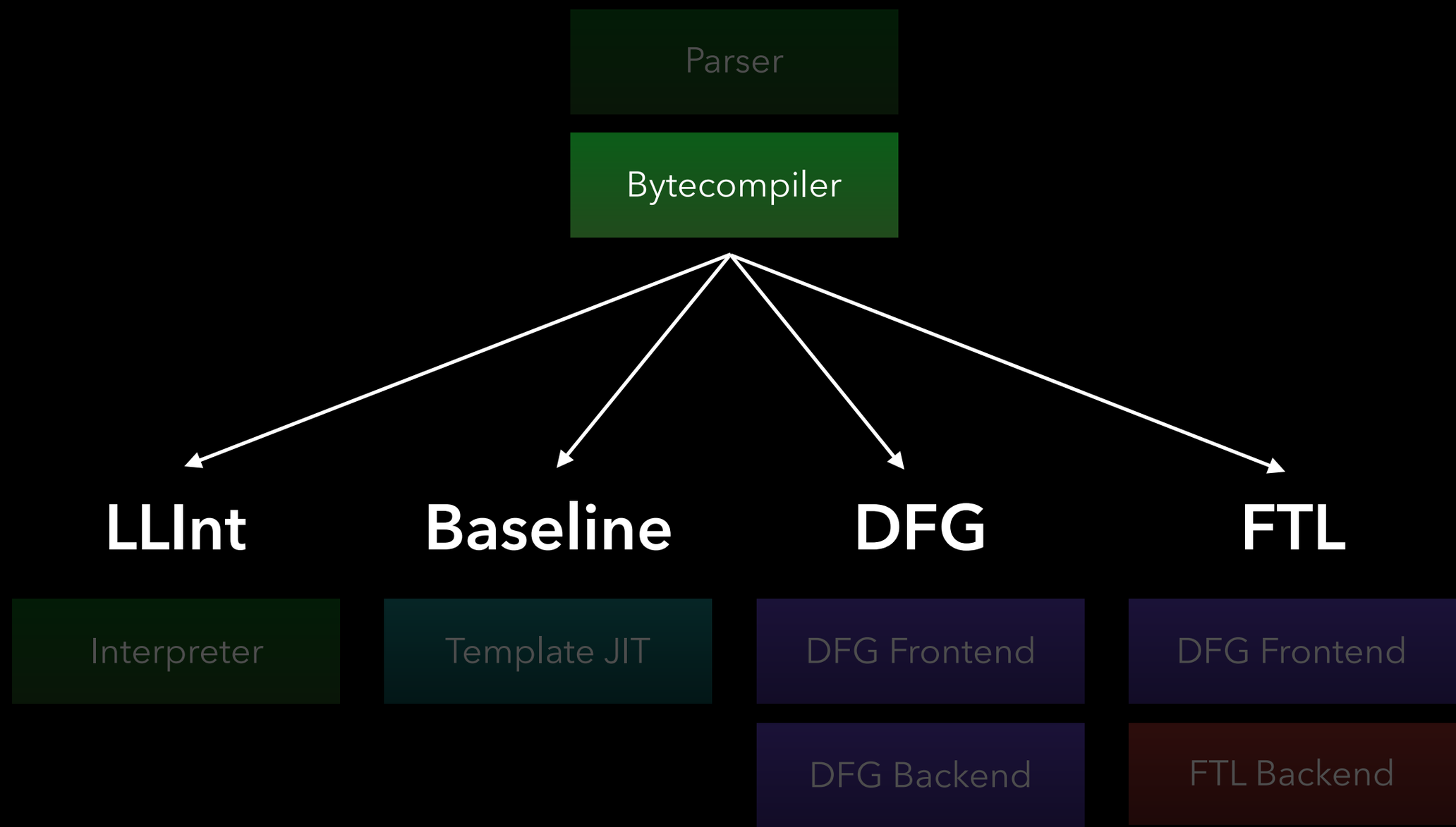
Agenda

- High level overview
- Old bytecode format
- New bytecode format
- Memory comparison
- Type safety improvements

Agenda

- High level overview
- Old bytecode format
- New bytecode format
- Memory comparison
- Type safety improvements


Parser
Bytecompiler
LLInt
Baseline
DFG
FTL
Interpreter
Template JIT
DFG Frontend
DFG Frontend
DFG Backend
FTL Backend


Parser
Bytecompiler
LLInt
Baseline
DFG
FTL
Interpreter
Template JIT
DFG Frontend
DFG Frontend
DFG Backend
FTL Backend

Bytecode Goals

- Memory efficiency
- Cacheable

Bytecode

```
// double.js
function double(a) {
    return a + a;
}
double(2);
```

```
$ jsc -d double.js
```

Bytecode

```
[ 0] enter
[ 1] get_scope     loc4
[ 3] mov           loc5, loc4
[ 6] check_traps
[ 7] add           loc7, arg1, arg1,
                     OperandTypes(126, 126)
[13] ret           loc7
```

Agenda

- High level overview
- Old bytecode format
- New bytecode format
- Memory comparison
- Type safety improvements

Old Bytecode Format

- Used too much memory
- The instruction stream was writable
- It had optimizations that were no longer beneficial

Old Bytecode Format

- Unlinked Instructions
  - Compact
  - Optimized for storage
- Linked Instructions
  - Inflated
  - Optimized for execution

Unlinked Instruction

1 byte	1 byte	1 byte	1 byte	2 bytes
op_add 0x1A	***dst*** 0xF8	***lhs*** 0x01	***rhs*** 0x01	***operandTypes*** 0xFEFE

Linked Instruction

8 bytes	8 bytes	8 bytes	8 bytes	8 bytes
op_add 0x00000000010003240	***dst*** 0xFFFFFFFFFFFFFFF8	***lhs*** 0x0000000000000001	***rhs*** 0x0000000000000001	***arithProfile*** 0x00000000100039D8

Execution

- Direct threading
- Inline caching

Execution

- offlineasm overview
- Direct threading
- Inline caching

Execution

- offlineasm overview
- Direct threading
- Inline caching

offlineasm

```
macro load(tmp, getter)
    getter(tmp)
    loadi [tmp], tmp
end

_label:
    load(t0, macro(tmp) move 42, tmp end)
```

offlineasm

```
macro load(tmp, getter)
    getter(tmp)
    loadi [tmp], tmp
end

_label:
    load(t0, macro(tmp) move 42, tmp end)
```

Temporary registers: t0-t5

offlineasm

```
macro load(tmp, getter)
    getter(tmp)
    loadi [tmp], tmp
end

_label:
    load(t0, macro(tmp) move 42, tmp end)
```

Instruction suffixes

- b for byte
- h for 16-bit
- i for 32-bit
- q for 64-bit
- p for pointer

offlineasm

```
macro load(tmp, getter)
    getter(tmp)
    loadi [tmp], tmp
end

_label:
    load(t0, macro(tmp) move 42, tmp end)
```

Macros are lambda expressions that take zero or more arguments and return code

offlineasm

```
macro load(tmp, getter)
    getter(tmp)
    loadi [tmp], tmp
end

_label:
    load(t0, macro(tmp) move 42, tmp end)
```

Macros may be anonymous

offlineasm

```
macro load(tmp, getter)
    getter(tmp)
    loadi [tmp], tmp
end

_label:
    load(t0, macro(tmp) move 42, tmp end)
```

And macros can also be passed as arguments to other macros

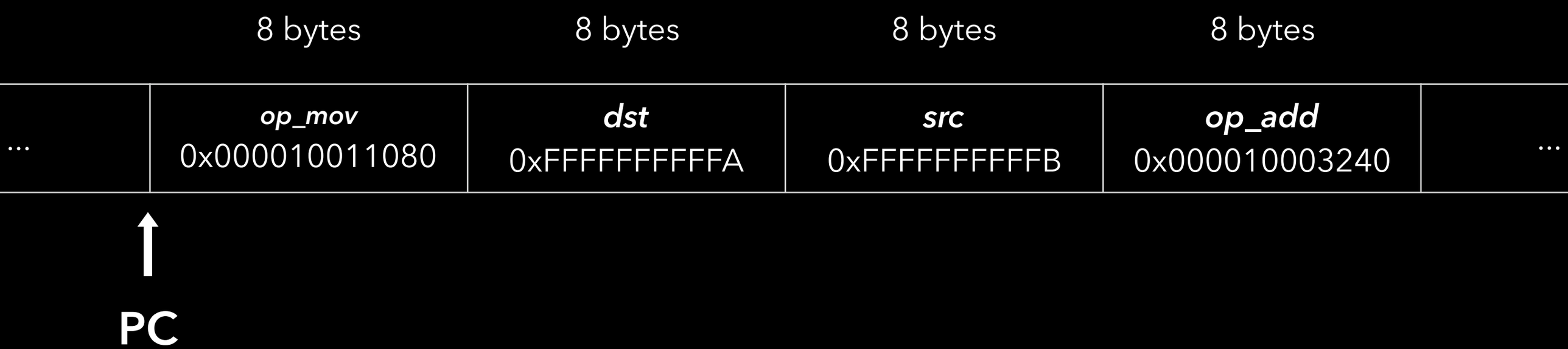
Execution

- offlineasm overview
- Direct threading
- Inline caching

Direct Threading

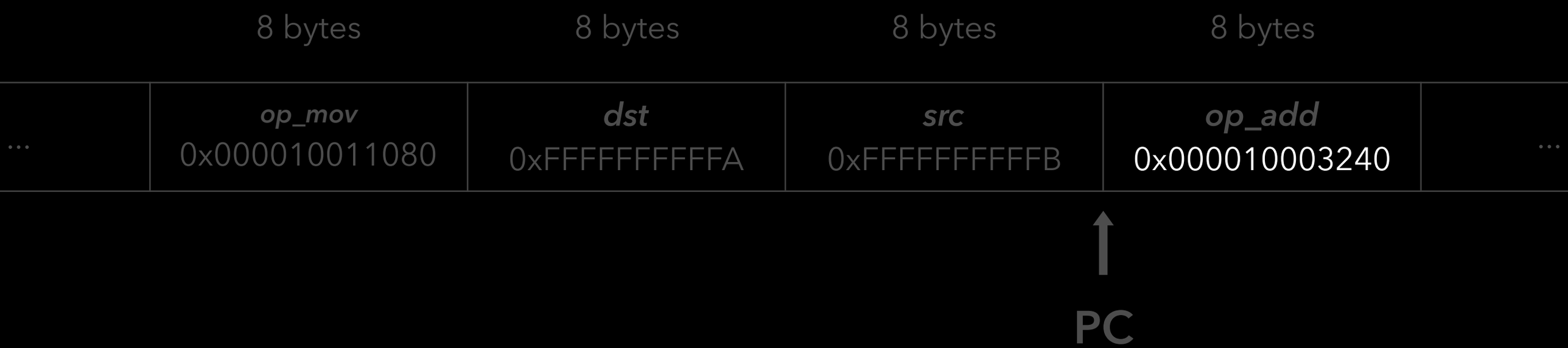

```
macro dispatch(instructionSize)
    addp instructionSize * PtrSize, PC
    jmp [PC]
end
```

Direct Threading




```
macro dispatch(instructionSize)
    addp instructionSize * PtrSize, PC
    jmp [PC]
end
```

Direct Threading




```
macro dispatch(instructionSize)
    addp instructionSize * PtrSize, PC
    jmp [PC]
end
```

Execution

- offlineasm overview
- Direct threading
- Inline caching

Inline Caching

```
object.field
```

↓

```
get_by_id object, field
```

Inline Caching

Structure #0x197	
field	0x10
x	0x20
Y	0x30

object #1	
0x10	42
0x20	“foo”
0x30	false

object #2	
0x10	[13, 42]
0x20	true
0x30	{}

Inline Caching

```
object.field
```

↓

```
get_by_id object, field, 0, 0
```

Structure ID

Offset

object #1	
0x10	42
0x20	"foo"
0x30	false

Structure #0x197	
field	0x10
x	0x20
Y	0x30

```
object.field
```

↓

```
get_by_id object, field, 0, 0
```

object #1	
0x10	42
0x20	"foo"
0x30	false

Structure #0x197	
field	0x10
x	0x20
Y	0x30

```
object.field
```

↓

```
get_by_id object, field, 0x197, 0x10
```

Agenda

- High level overview
- Old bytecode format
- New bytecode format
- Memory comparison
- Type safety improvements

New Bytecode

- Compact
  - No separate linked format
  - Multiple encoding sizes
- Cacheable
  - No runtime values
  - Read-only instruction stream

Narrow Instructions

1 byte	1 byte	1 byte	1 byte	1 byte	1 byte
op_add 0x1A	***dst*** 0xF8	***lhs*** 0x01	***rhs*** 0x01	***operandTypes*** 0xFE	***metadataID*** 0x00

Wide Instructions

(32-bit words)

1 byte	4 bytes	4 bytes	4 bytes	4 bytes	4 bytes	4 bytes
op_wide 0x01	***op_add*** 0x0000001A	***dst*** 0xFFFFFFF8	***lhs*** 0x00000001	***rhs*** 0x00000001	***operandTypes*** 0xFFFFFFFE	***metadataID*** 0x00010000

Metadata Table

op_add
op_call
…

	0	1	…
arithProfile:	ArithProfile()	ArithProfile()	…

	0	1	…
arithProfile:	ArithProfile()	ArithProfile()	…
valueProfile:	ValueProfile()	ValueProfile()	…

Metadata Table

~200 opcodes × 8 bytes × ~23k tables

=

~36MB

Metadata Table

Header			Payload			
0x0	*0x4*	…	*0x100*	*0x110*	*0x120*	…
op_add 0x100	***op_call*** 0x120	…	OpAdd::Metadata[0]	OpAdd::Metadata[1]	OpCall::Metadata[0]	…

- Allocate the whole table as a single chunk of memory
- Only allocate space for opcodes that have metadata
- Change the header from pointer to unsigned offset

Execution

- Indirect threading
- Inline caching
- Wide instruction execution

Execution

- Indirect threading
- Inline caching
- Wide instruction execution

Indirect Threading

```
macro dispatch(instructionSize)
    addp instructionSize * PtrSize, PC
    jmp [PC]
end
```

Indirect Threading

```
macro dispatch(instructionSize)
    addp instructionSize, PC
    loadb [PC], t0
    leap _g_opcodeMap, t1
    jmp [t1, t0, PtrSize]
end
```

Execution

- Indirect threading
- Inline caching
- Wide instruction execution

Inline Caching

CallFrame

↓

CodeBlock

↓

Instruction Stream

↓

MetadataTable **[** *OpcodeID* **] [** MetadataID **]**

Execution

- Indirect threading
- Inline caching
- Wide instruction execution

Wide Instruction Execution

```
macro dispatch(instructionSize)
    addp instructionSize, PC
    loadb [PC], t0
    leap _g_opcodeMap, t1
    jmp [t1, t0, PtrSize]
end
```

Wide Instruction Execution

```
macro dispatch(instructionSize)
    addp instructionSize, PC
    loadb [PC], t0
    leap _g_opcodeMap, t1
    jmp [t1, t0, PtrSize]
end

_llint_op_wide:
    loadi 1[PC], t0
    leap _g_opcodeMapWide, t1
    jmp [t1, t0, PtrSize]
```

Wide Instruction Execution

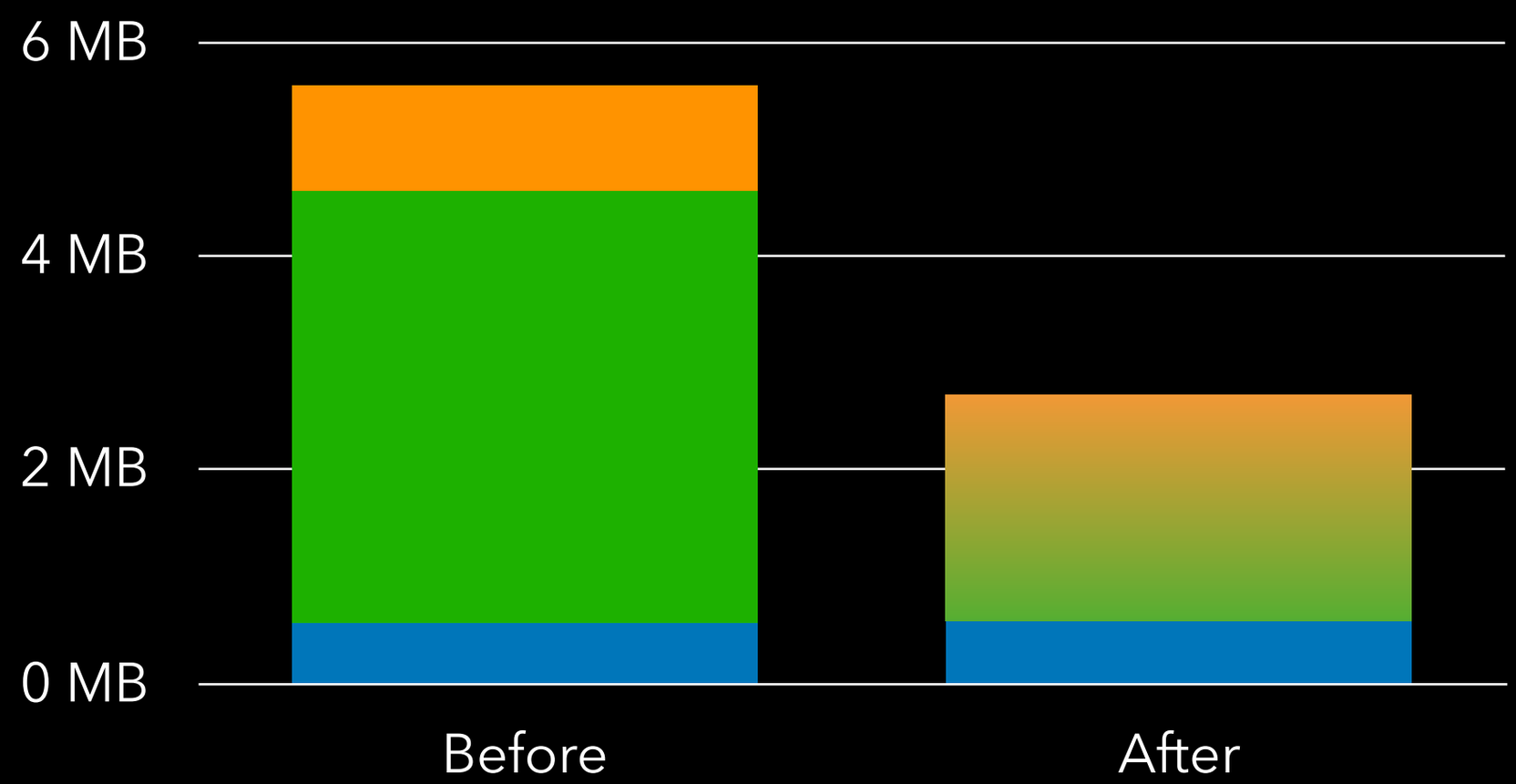
```
macro dispatch(instructionSize)
    addp instructionSize, PC
    loadb [PC], t0
    leap _g_opcodeMap, t1
    jmp [t1, t0, PtrSize]
end

_llint_op_wide:
    loadi 1[PC], t0
    leap _g_opcodeMapWide, t1
    jmp [t1, t0, PtrSize]
```

Agenda

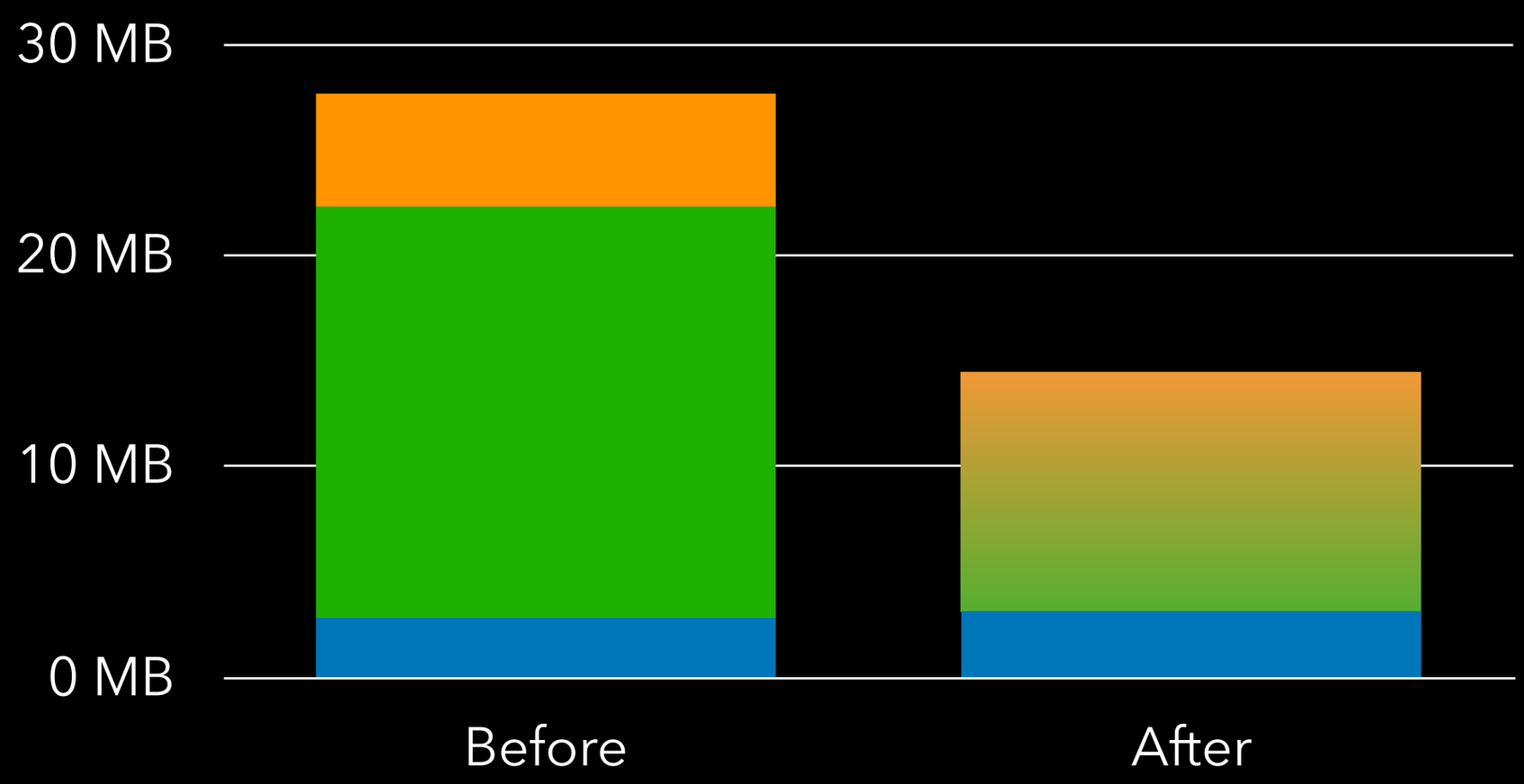
- High level overview
- Old bytecode format
- New bytecode format
- Memory comparison
- Type safety improvements

apple.com



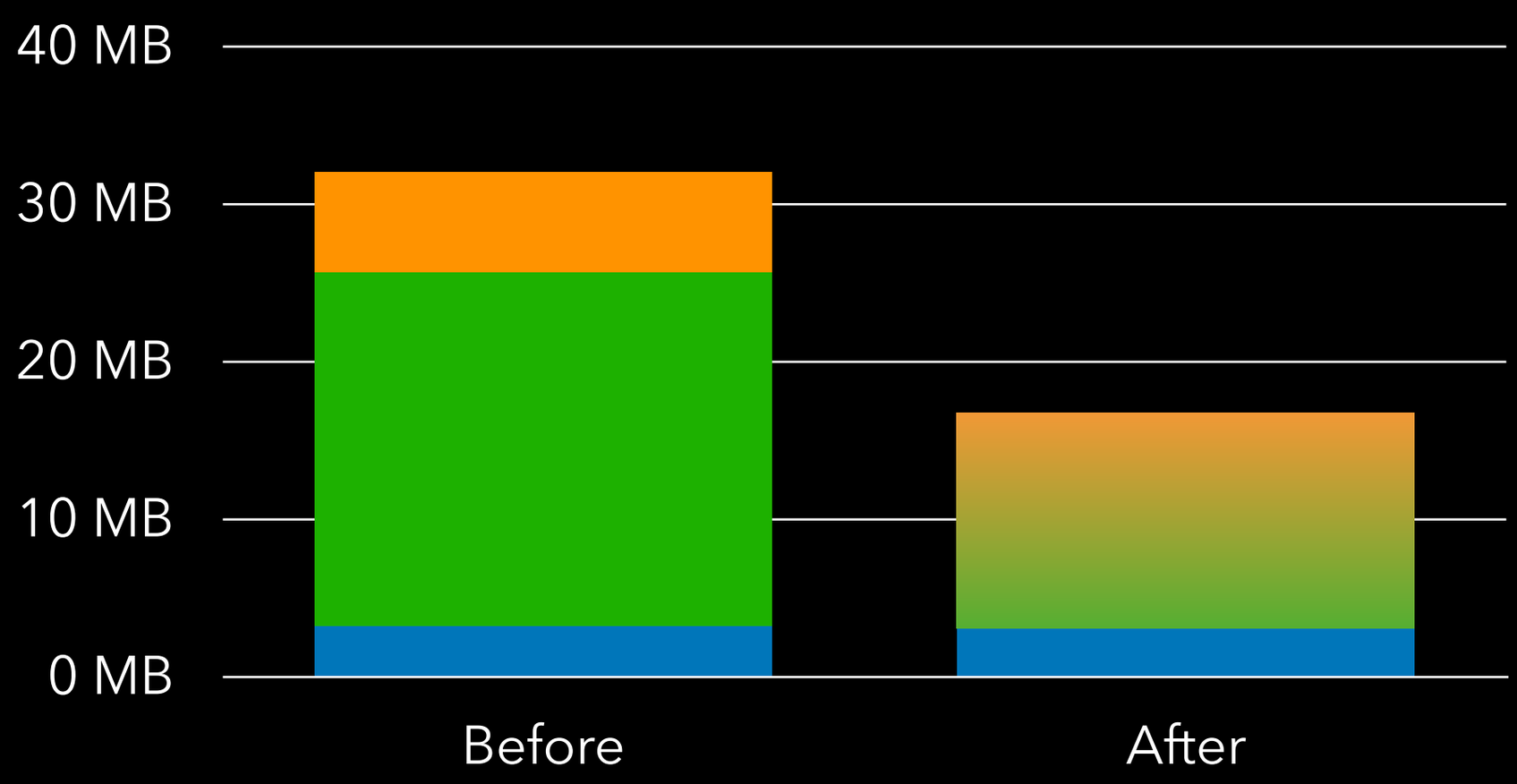
Description	Before	After	%
Unlinked	0.55 MB	0.57 MB	+4%
Linked	4.05 MB	2.14 MB	-57%
Metadata	0.99 MB		
Total	***5.60 MB***	***2.71 MB***	***-52%***

reddit.com



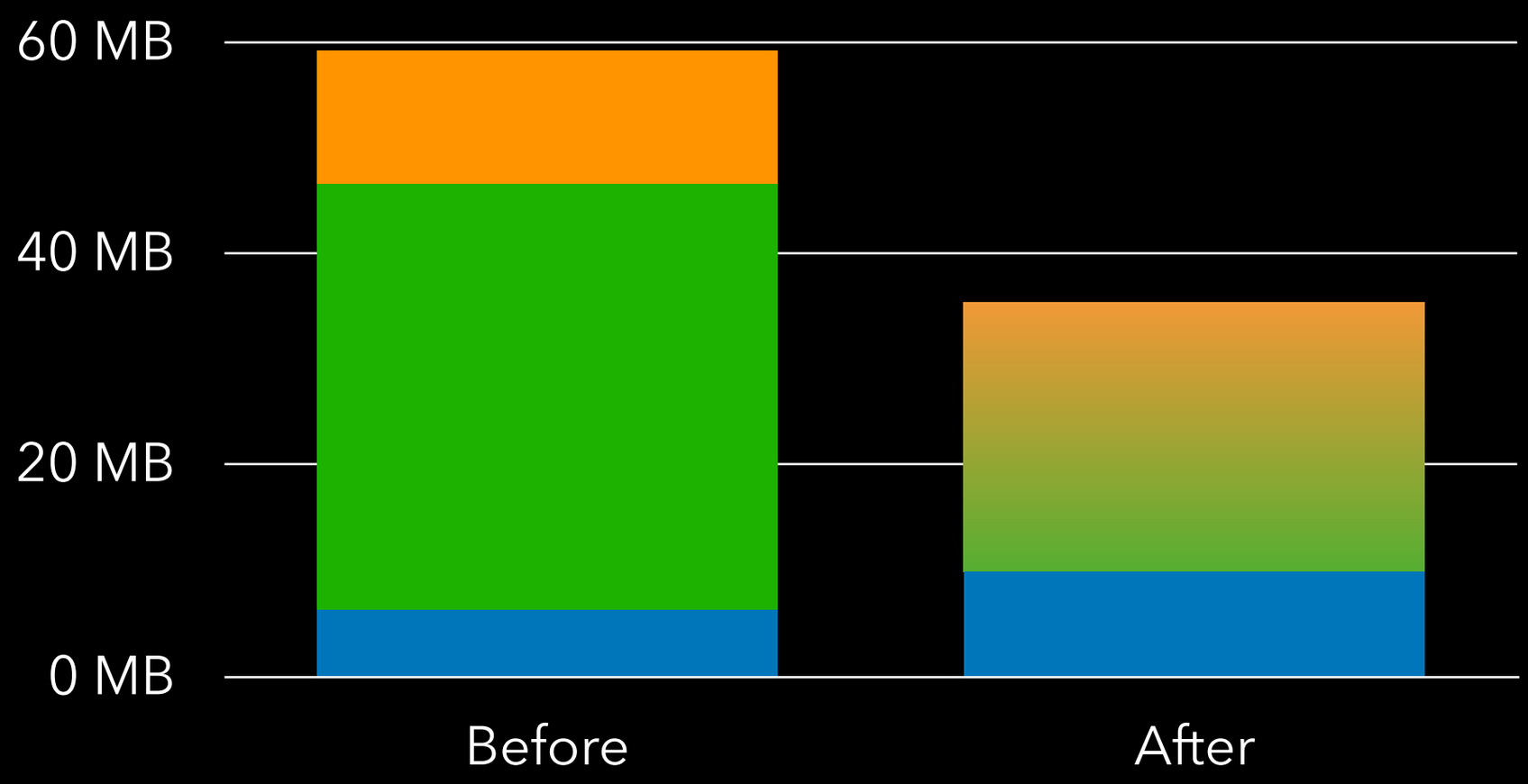
Description	Before	After	%
Unlinked	2.76 MB	3.08 MB	+12%
Linked	19.51 MB	11.37 MB	-54%
Metadata	5.34 MB		
Total	***27.61 MB***	***14.45 MB***	***-48%***

facebook.com



Description	Before	After	%
Unlinked	3.11 MB	2.99 MB	-4%
Linked	22.43 MB	13.66 MB	-52%
Metadata	6.51 MB		
Total	***32.04 MB***	***16.65 MB***	***-48%***

gmail.com



Description	Before	After	%
Unlinked	6.17 MB	9.89 MB	+60%
Linked	40.28 MB	25.51 MB	-52%
Metadata	12.75 MB		
Total	***59.21 MB***	***35.40 MB***	***-40%***

gmail.com

- More than 12k code blocks
- More than 830k instructions
- 270k wide instructions (33%)

Wide Instructions

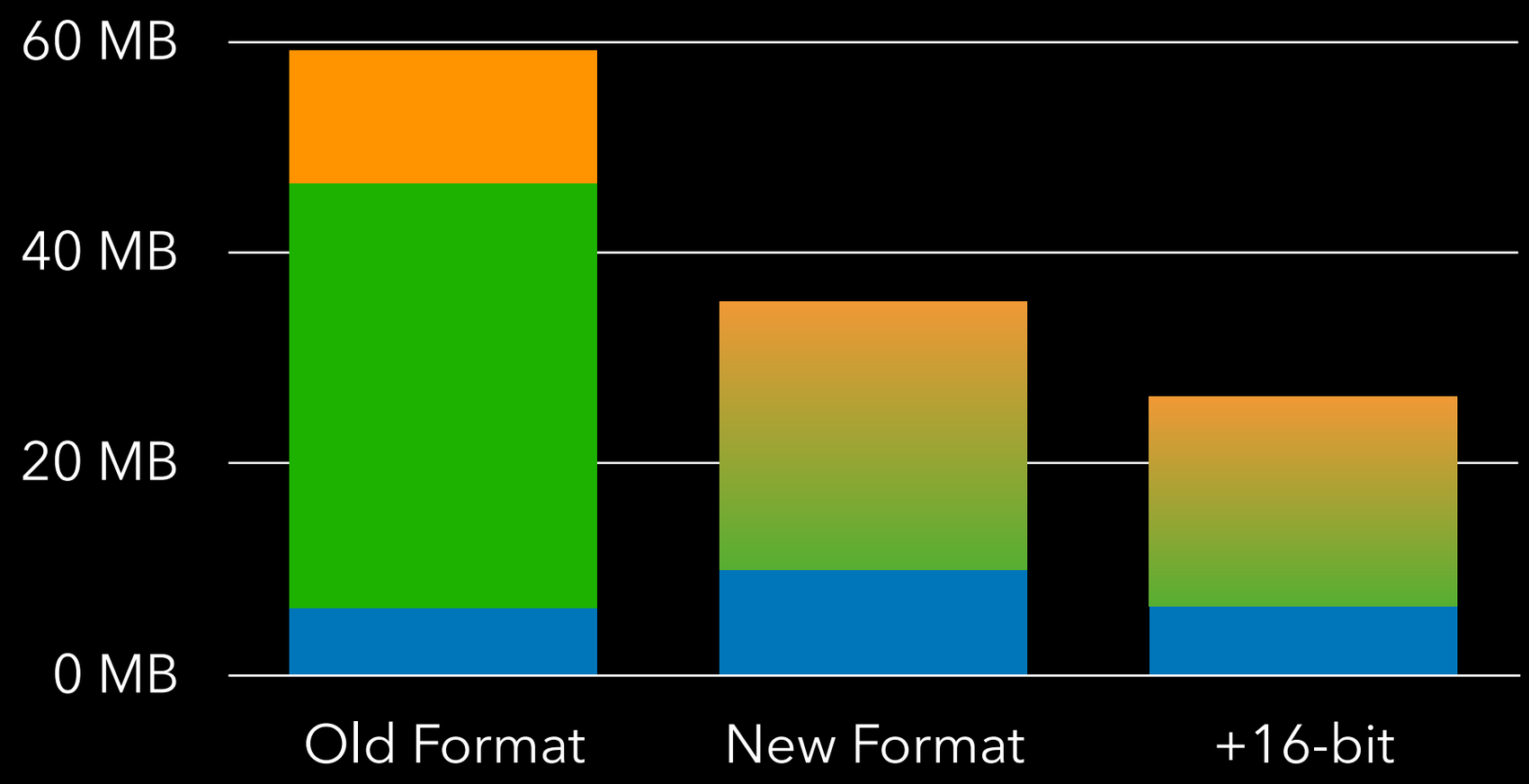
(16-bit words)

1 byte	2 bytes	2 bytes	2 bytes	2 bytes	2 bytes	2 bytes
op_wide16 0x00	***op_add*** 0x001A	***dst*** 0xFFF8	***lhs*** 0x0001	***rhs*** 0x0001	***operandTypes*** 0xFEFE	***metadataID*** 0x0100

Metadata Table

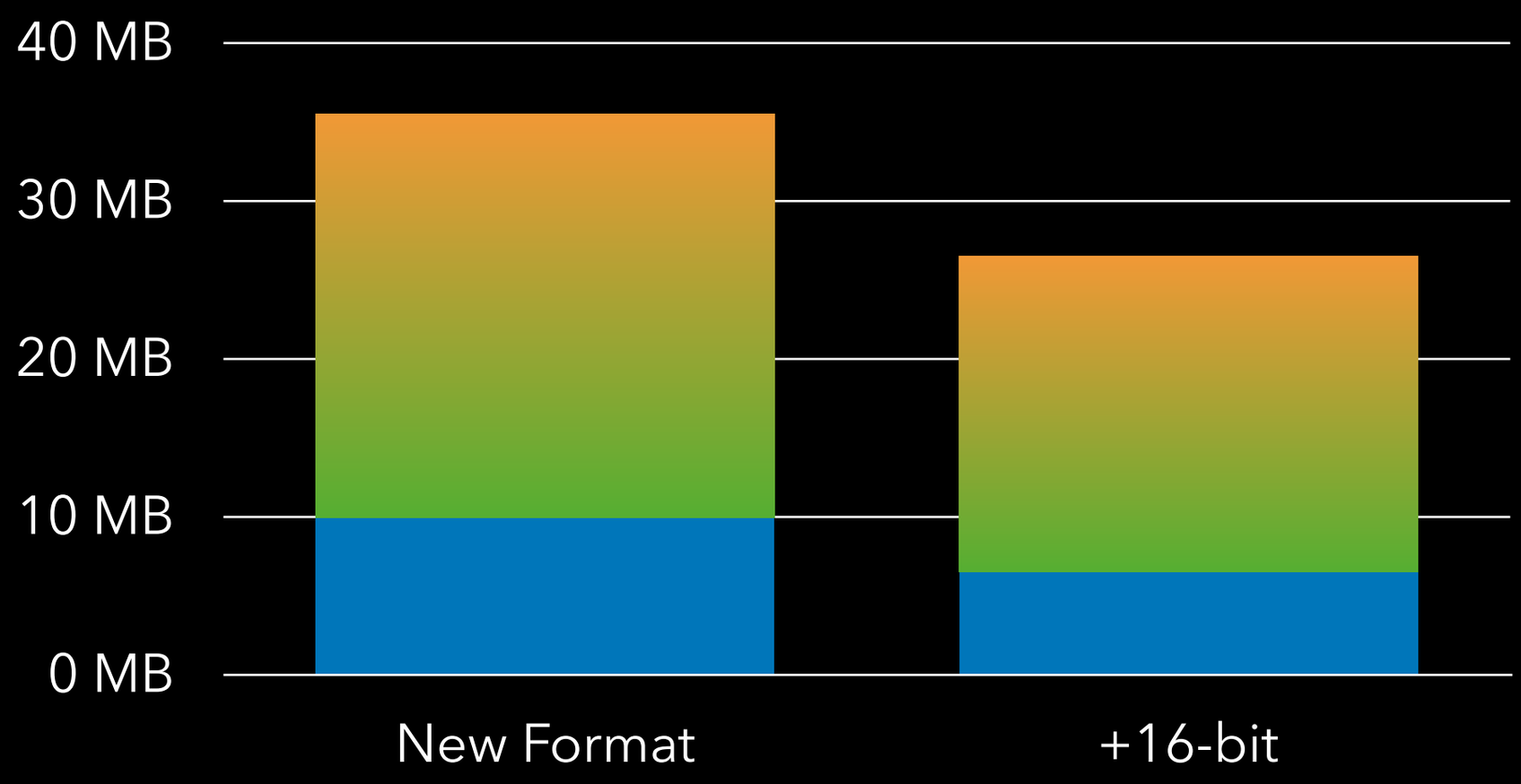
Header			Payload			
0x0	*0x2*	…	*0x80*	*0x90*	*0xA0*	…
op_add 0x80	***op_call*** 0xA0	…	OpAdd::Metadata[0]	OpAdd::Metadata[1]	OpCall::Metadata[0]	…

gmail.com



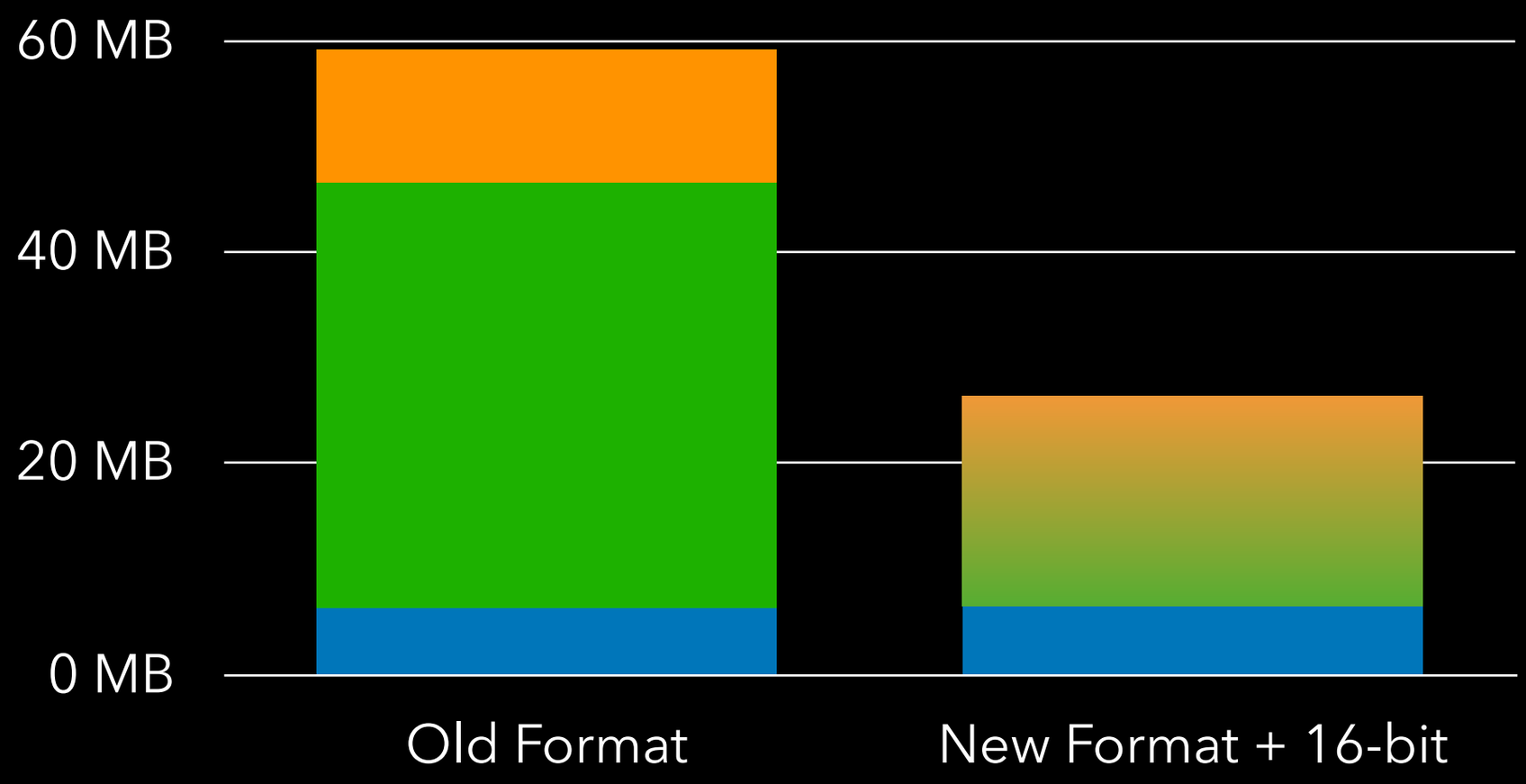
Description	Old Format	New Format	+ 16-bit
Unlinked	6.17 MB	9.89 MB	6.40 MB
Linked	40.28 MB	25.51 MB	20.03 MB
Metadata	12.75 MB		
Total	***59.21 MB***	***35.40 MB***	***26.42 MB***

gmail.com



Description	New Format	+ 16-bit	%
Unlinked	9.89 MB	6.40 MB	-35%
Linked	25.51 MB	20.03 MB	-21%
Metadata			
Total	***35.40 MB***	***26.42 MB***	***-26%***

gmail.com



Description	Before	16-bit	%
Unlinked	6.17 MB	6.40 MB	+4%
Linked	40.28 MB	20.03 MB	-62%
Metadata	12.75 MB		
Total	***59.21 MB***	***26.42 MB***	***-55%***

Agenda

- High level overview
- Old bytecode format
- New bytecode format
- Memory comparison
- Type safety improvements

Old Instruction Definition

```
{ "name": "op_add", "length": 5 }
```

Old Instruction Access

```
SLOW_PATH_DECL(slow_path_add)
{
    JSValue lhs = OP_C(2).jsValue();
    JSValue rhs = OP_C(3).jsValue();
    ...
}
```

Old Instruction Access

```
SLOW_PATH_DECL(slow_path_add)
{
    JSValue lhs = exec->r(pc[2].u.operand).jsValue();
    JSValue rhs = exec->r(pc[3].u.operand).jsValue();
    …
}
```

Old Instruction Access

```
SLOW_PATH_DECL(slow_path_add)
{
    JSValue lhs = exec->r(pc[2].u.operand).jsValue();
    JSValue rhs = exec->r(pc[3].u.operand).jsValue();
    …
}
```

Old Instruction Access

```
union {
    void* pointer;
    Opcode opcode;
    int operand;
    unsigned unsignedValue;
    WriteBarrierBase<Structure> structure;
    StructureID structureID;
    WriteBarrierBase<SymbolTable> symbolTable;
    WriteBarrierBase<StructureChain> structureChain;
    WriteBarrierBase<JSCell> jsCell;
    WriteBarrier<Unknown>* variablePointer;
    Special::Pointer specialPointer;
    PropertySlot::GetValueFunc getterFunc;
    LLIntCallLinkInfo* callLinkInfo;
    UniquedStringImpl* uid;
```

New Instruction Definition

```
op :add,
    args: {
        dst: VirtualRegister,
        lhs: VirtualRegister,
        rhs: VirtualRegister,
        operandTypes: OperandTypes,
    },
    metadata: {
        arithProfile: ArithProfile,
    }
```

Opcode Struct

```
struct OpAdd : public Instruction {
    static constexpr OpcodeID opcodeID = op_add;

    VirtualRegister m_dst;
    VirtualRegister m_lhs;
    VirtualRegister m_rhs;
    OperandTypes m_operandTypes;
    unsigned m_metadataID;
};
```

Metadata Struct

```
struct OpAdd::Metadata {
    WTF_MAKE_NONCOPYABLE(Metadata);

public:
    Metadata(const OpAdd& __op)
        : m_arithProfile(__op.m_operandTypes)
    { }

    ArithProfile m_arithProfile;
};
```

Autogenerate all the things!

- Instruction fitting
- Instruction decoding (narrow vs wide)
- Pretty printing
- Constants for offlineasm
- Opcode IDs
- ...

New Instruction Access

```
SLOW_PATH_DECL(slow_path_add)
{
    OpAdd bytecode = pc->as<OpAdd>();
    JSValue lhs = GET_C(bytecode.m_lhs);
    JSValue rhs = GET_C(bytecode.m_rhs);
    ...
}
```

New Instruction Access

```
SLOW_PATH_DECL(slow_path_add)
{
    OpAdd bytecode = pc->as<OpAdd>();
    JSValue lhs = exec->r(bytecode.m_lhs.offset());
    JSValue rhs = exec->r(bytecode.m_rhs.offset());
    ...
}
```

New Instruction Access

```
SLOW_PATH_DECL(slow_path_add)
{
    OpAdd bytecode = pc->as<OpAdd>();
    JSValue lhs = exec->r(bytecode.m_lhs.offset());
    JSValue rhs = exec->r(bytecode.m_rhs.offset());
    ...
}
```

Thank you!

@tadeuzagallo