JavaScriptCore, Many Compilers Make this Engine Perform

Michael Saboff
Apple Inc.

webkit.org

https://svn.webkit.org/repository/webkit/trunk

Agenda

- High Level Overview
- Tiers
- Optimization Techniques
  - Counting Triggers
  - OSR (On Stack Replacement)
  - Profiling
  - Speculation
  - Inline Caching
  - Other Compiler Optimizations

JavaScriptCore Uses

- JS engine for Safari.
- Used by many apps, Apple and 3rd party including React Native.
- Other WebKit contributors use it for set top boxes, video game consoles, in vehicle systems and other custom JS environments.

JavaScriptCore

- Fork from KDE JS engine 17 years ago (~37K lines / 94 files).
- Written mostly in C++
  some Objective-C, Ruby, Python and Perl.
- >600K lines of code in >2500 source files.
- >4.5M lines of test code in >50,000 tests.

```
function addTo(o, v)
{
    o.sum += v;
}
```

```
function addTo(o, v)
{
    o.sum += v;
}
```

```
addTo:
[   0] enter
[   1] get_scope  loc4
[   3] mov        loc5, loc4
[   6] get_by_id  loc6, arg1, “sum”
[  11] add        loc6, loc6, arg2
[  17] put_by_id  arg1, “sum”, loc6
[  23] ret        Undefined
```

```
function addTo(o, v)
{
    o.sum += v;
}
```

```
addTo:
[   0] enter
[   1] get_scope  loc4
[   3] mov        loc5, loc4
[   6] get_by_id  loc6, arg1, “sum”
[  11] add        loc6, loc6, arg2
[  17] put_by_id  arg1, “sum”, loc6
[  23] ret        Undefined
```

```
function addTo(o, v)
{
    o.sum += v;
}
```

```
addTo:
[   0] enter
[   1] get_scope  loc4
[   3] mov        loc5, loc4
[   6] get_by_id  loc6, arg1, “sum”
[  11] add        loc6 = loc6 + arg2
[  17] put_by_id  arg1, “sum”, loc6
[  23] ret        Undefined
```

```
function addTo(o, v)
{
    o.sum += v;
}
```

```
addTo:
[   0] enter
[   1] get_scope  loc4
[   3] mov        loc5, loc4
[   6] get_by_id  loc6, arg1, “sum”
[  11] add        loc6, loc6, arg2
[  17] put_by_id  arg1, “sum”, loc6
[  23] ret        Undefined
```

Fast Path / Slow Path

Fast Path / Slow Path

```
op_add(a, b)
{
    if (isInt32(a) && isInt32(b))
        return a + b;
    return slowAdd(a, b);
}
```

Agenda

- High Level Overview
- Tiers
- Optimization Techniques
  - Counting Triggers
  - OSR (On Stack Replacement)
  - Profiling
  - Speculation
  - Inline Caching
  - Other Compiler Optimizations

~9 JIT Compilers

JavaScript execution engines:

- LLInt *interpreter*
- Baseline *template JIT*
- DFG *less optimizing JIT*
- FTL *full optimizing JIT*
- Polymorphic Access *template JIT*
- Snippet *template JIT*

WebAssembly execution engines:

- Wasm *interpreter*
- Wasm BBQ *less optimizing JIT*
- Wasm OMG *full optimizing JIT*

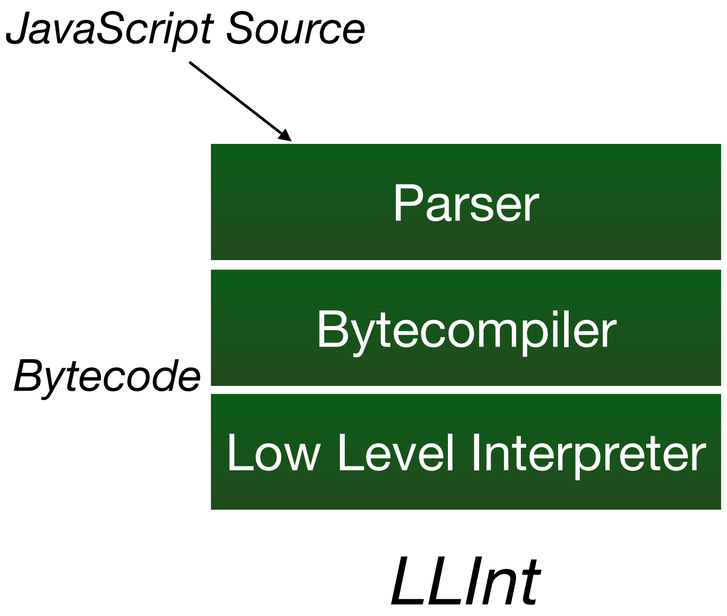
Other engines:

- YARR *regexp interpreter*
- YARR *regexp template JIT*
- CSS *template JIT*

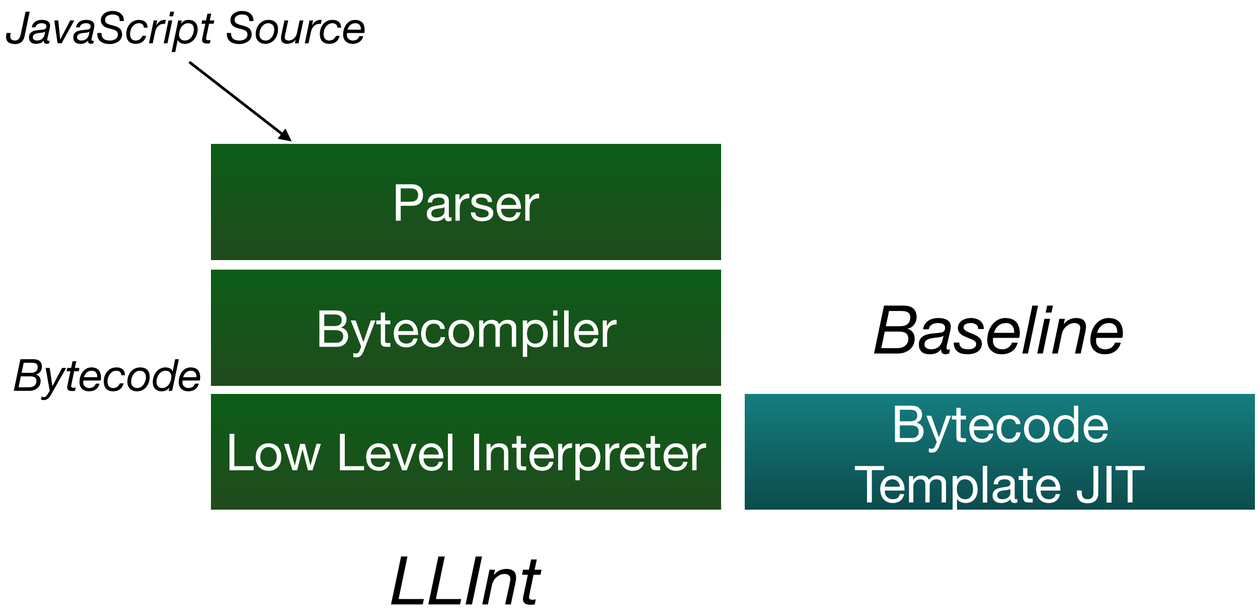
Four JS Tiers

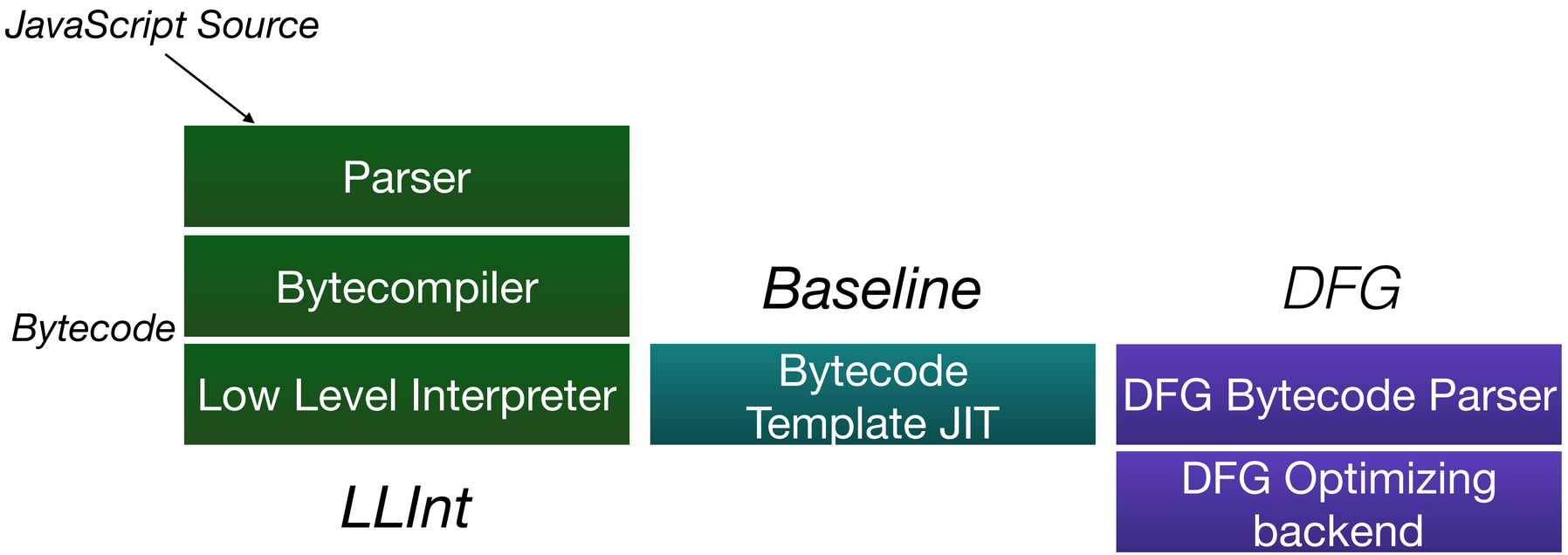
Four JS Tiers

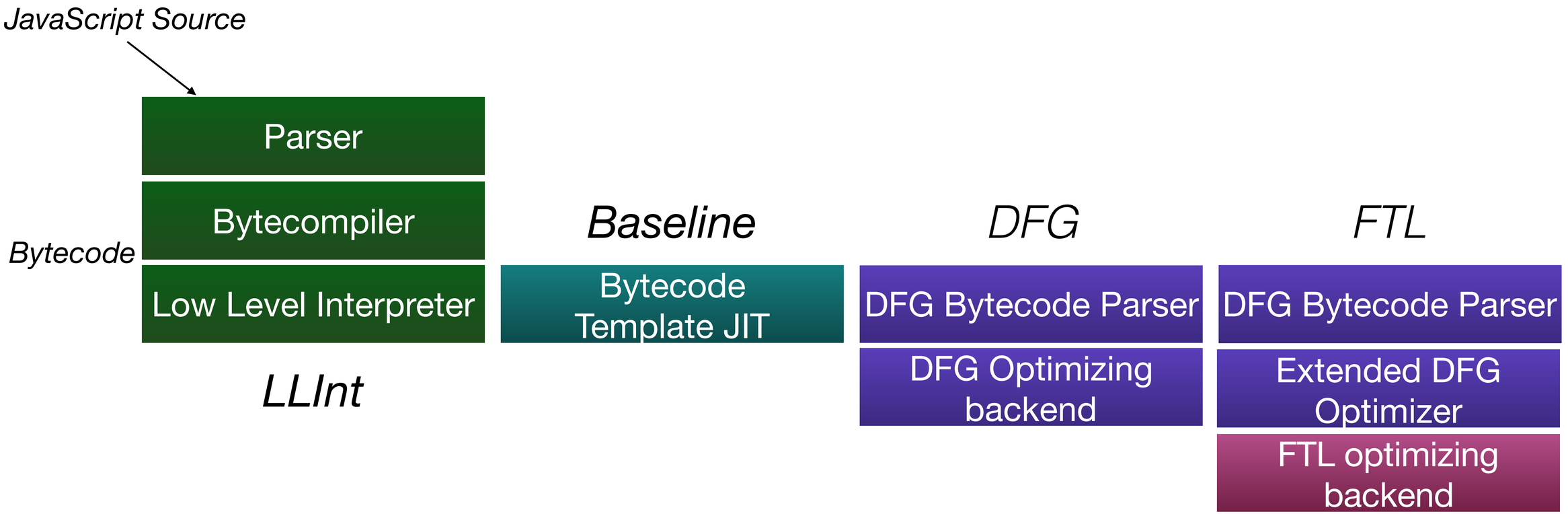
JavaScript Source


JavaScript Source
Parser
Bytecompiler
Bytecode
Low Level Interpreter

LLInt


JavaScript Source
Parser
Bytecompiler
Bytecode
Low Level Interpreter
LLInt
Baseline
Bytecode
Template JIT


JavaScript Source
Parser
Bytecompiler
Bytecode
Low Level Interpreter
LLInt
Baseline
Bytecode
Template JIT
DFG
DFG Bytecode Parser
DFG Optimizing
backend


JavaScript Source
Parser
Bytecompiler
Bytecode
Low Level Interpreter
LLInt
Baseline
Bytecode
Template JIT
DFG
DFG Bytecode Parser
DFG Optimizing
backend
FTL
DFG Bytecode Parser
Extended DFG
Optimizer
FTL optimizing
backend

Agenda

- High Level Overview
- Tiers
- Optimization Techniques
  - Counting Triggers
  - OSR (On Stack Replacement)
  - Profiling
  - Speculation
  - Inline Caching
  - Other Compiler Optimizations

Execution Count Thresholds for Tier-up

Tier-up Case	Required Count for Tier-up
LLInt → Baseline	500
Baseline → DFG	1000
DFG → FTL	100000

```
function addTo(o, v)
{
    o.sum += v;
}
```

```
addTo:
[   0] enter
[   1] get_scope  loc4
[   3] mov        loc5, loc4
[   6] get_by_id  loc6, arg1, “sum”
[  11] add        loc6, loc6, arg2
[  17] put_by_id  arg1, “sum”, loc6
[  23] ret        Undefined
```

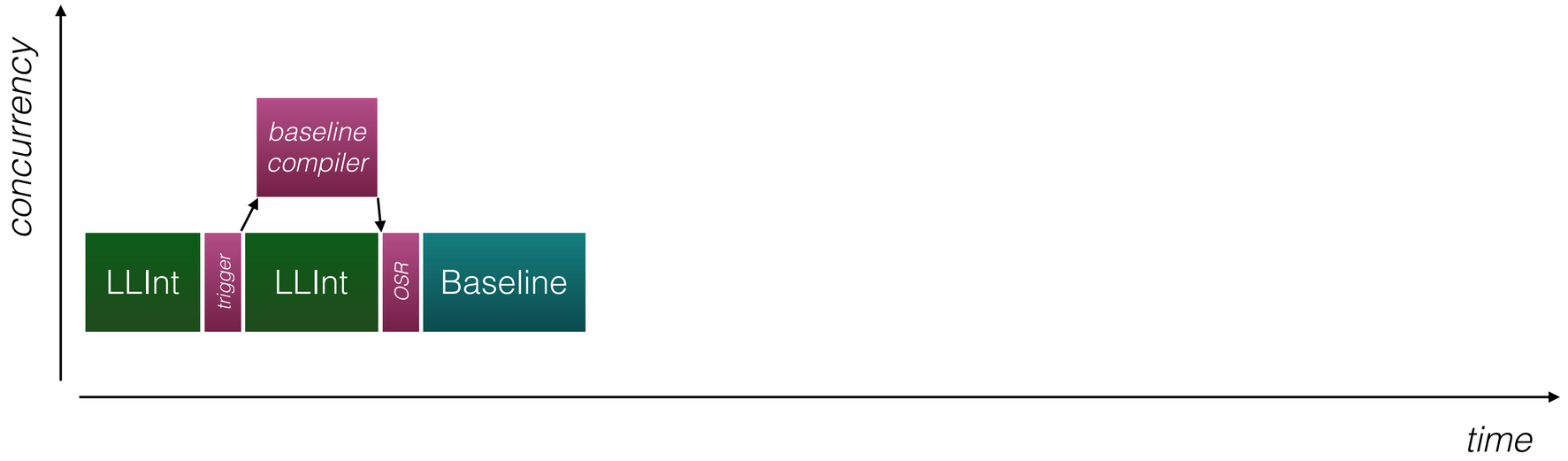
```
"use strict";

let accumulator = { sum: 0 };

for (let i = 0; i < 10000000; ++i)
    accumulator.sum += i;

print(accumulator.sum);
```

```
"use strict";

let accumulator = { sum: 0 };

for (let i = 0; i < 10000000; ++i)
    accumulator.sum += i;

print(accumulator.sum);
```

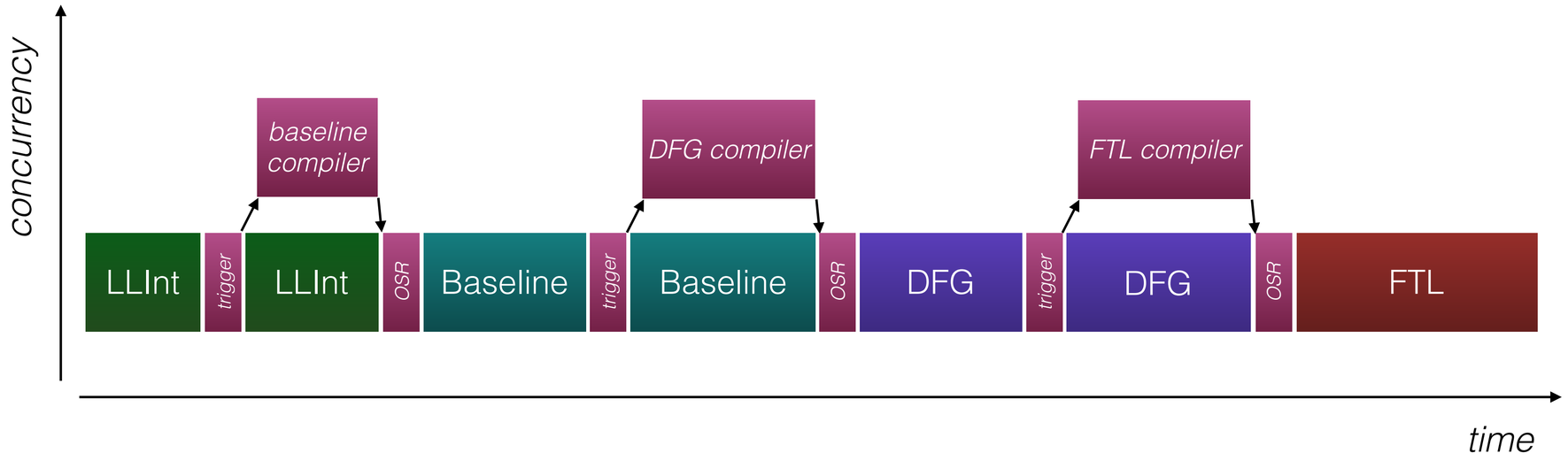
```
"use strict";

let accumulator = { sum: 0 };

for (let i = 0; i < 10000000; ++i)
    accumulator.sum += i;

print(accumulator.sum);
```

```
"use strict";

let accumulator = { sum: 0 };

for (let i = 0; i < 10000000; ++i)
    accumulator.sum += i;

print(accumulator.sum);
```

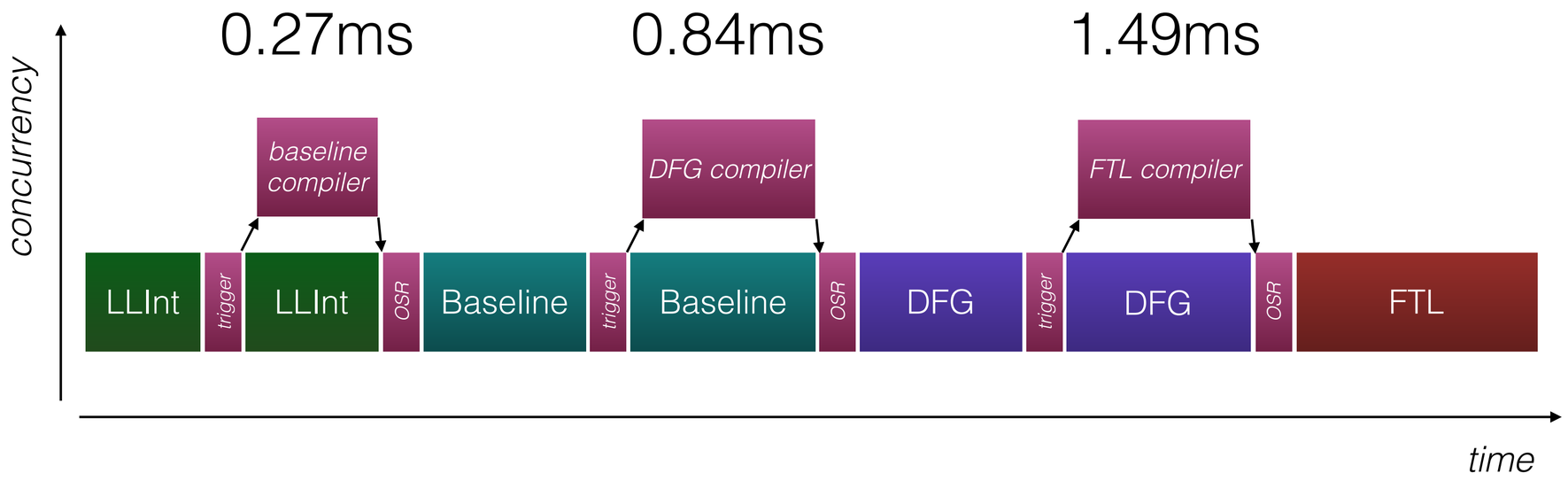
```
"use strict";

let accumulator = { sum: 0 };

for (let i = 0; i < 10000000; ++i)
    accumulator.sum += i;

print(accumulator.sum);
```



```
"use strict";

let accumulator = { sum: 0 };

for (let i = 0; i < 10000000; ++i)
    accumulator.sum += i;

print(accumulator.sum);
```



```
"use strict";

let accumulator = { sum: 0 };

for (let i = 0; i < 10000000; ++i)
    accumulator.sum += i;

print(accumulator.sum);
```



Agenda

- High Level Overview
- Tiers
- Optimization Techniques
  - Counting Triggers
  - OSR (On Stack Replacement)
  - Profiling
  - Speculation
  - Inline Caching
  - Other Compiler Optimizations

OSR

On Stack Replacement

```
function addTo(o, v)
{
    o.sum += v;
}
```

```
addTo:
[   0] enter
[   1] get_scope  loc4
[   3] mov        loc5, loc4
[   6] get_by_id  loc6, arg1, “sum”
[  11] add        loc6, loc6, arg2
[  17] put_by_id  arg1, “sum”, loc6
[  23] ret        Undefined
```

```
[  0] enter
```

```
[  0] enter
```

frame layout
matches bytecode

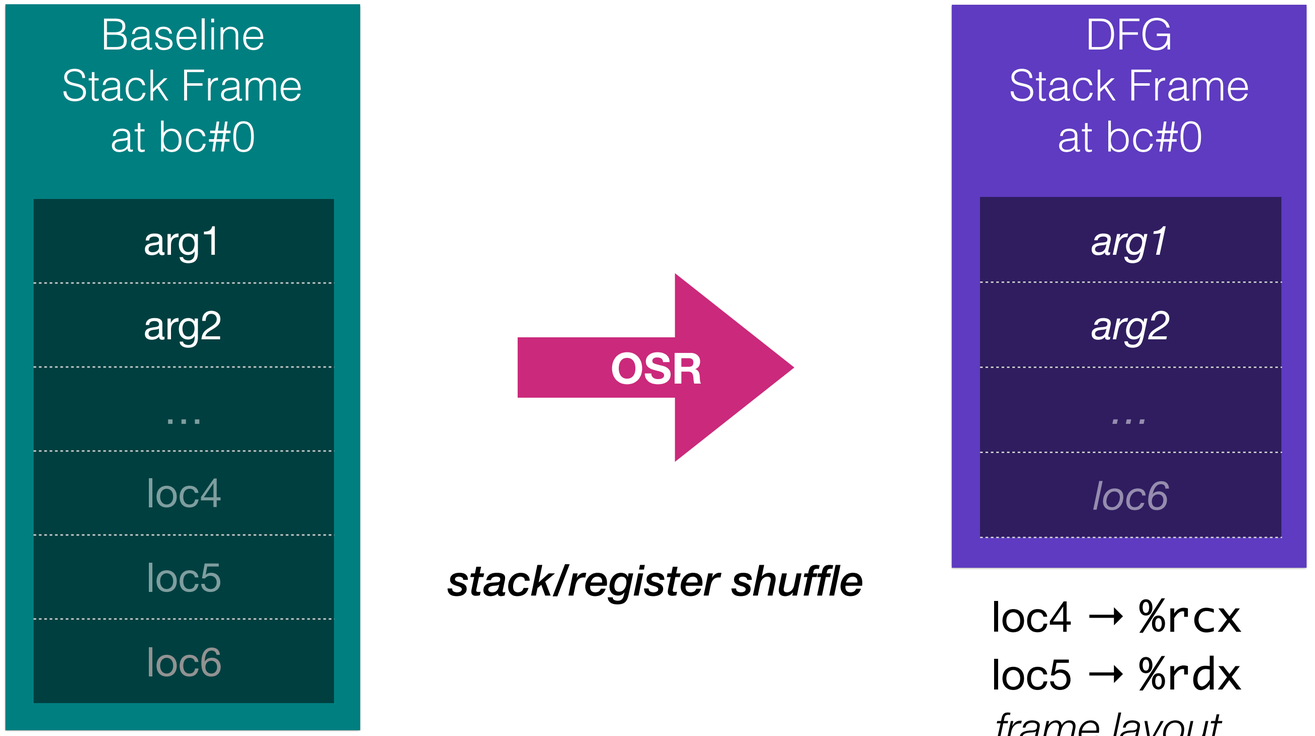
```
[  0] enter
```

frame layout matches bytecode

loc4 → %rcx
loc5 → %rdx

frame layout selected by complex process

```
[  0] enter
```



frame layout matches bytecode

frame layout selected by complex process

Fast Path / Slow Path

```
op_add(a, b)
{
    if (isInt32(a) && isInt32(b))
        return a + b;
    return slowAdd(a, b);
}
```

Fast Only

```
op_add(a, b)
{
    if (!isInt32(a) || !isInt32(b))
        exitToUnoptimizedTier();
    // Code below only needs int32 path
    return a + b;
}
```

Fast Only

```
op_addFast(a, b)
{
    return a + b;
}

if (!isInt32(a))
    exitToUnoptimizedTier();
if (!isInt32(b))
    exitToUnoptimizedTier();
…
result = op_addFast(x, 12);
…
```

```
[  11] add loc6, loc6, arg2
```

```
[  11] add loc6, loc6, arg2
```

frame layout
matches bytecode

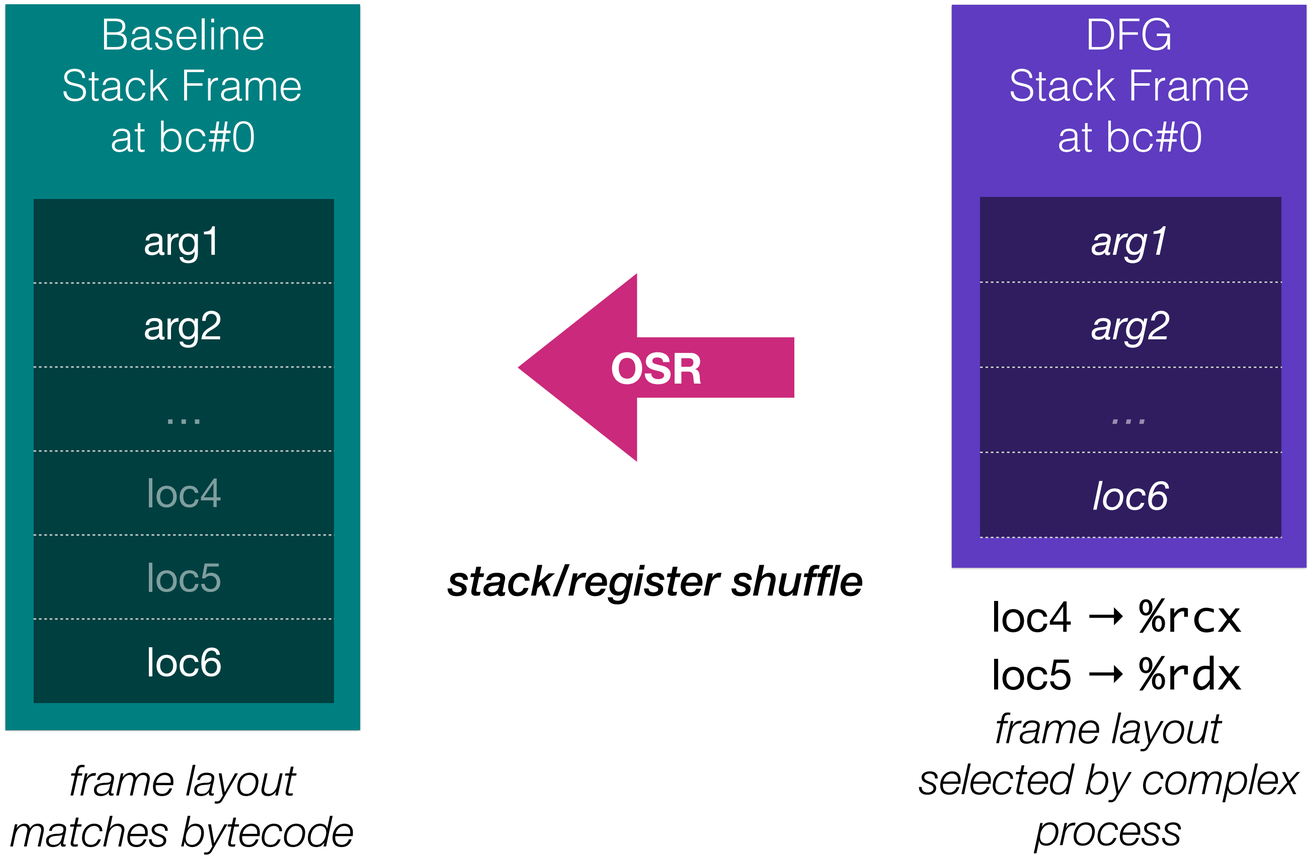
```
[  11] add loc6, loc6, arg2
```

frame layout matches bytecode

loc4 → %rcx
loc5 → %rdx

frame layout selected by complex process

```
[  11] add loc6, loc6, arg2
```



frame layout matches bytecode

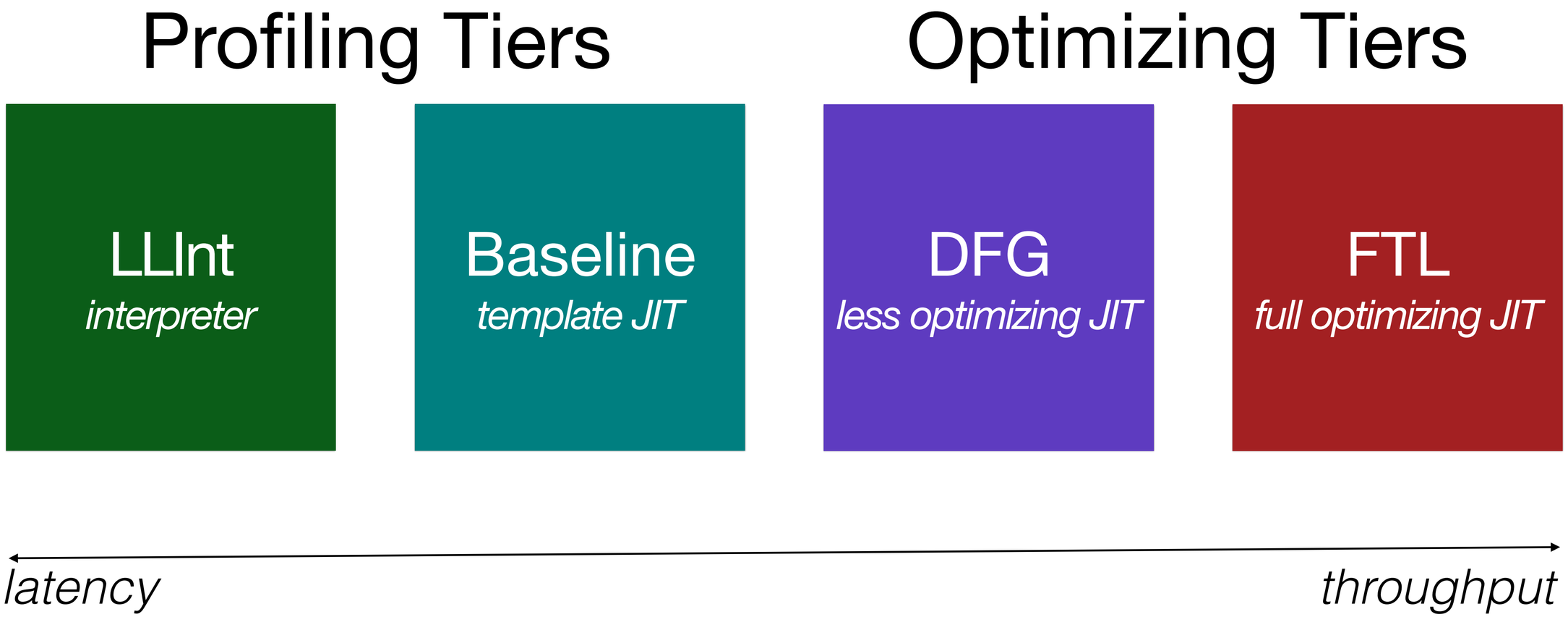
frame layout selected by complex process

Agenda

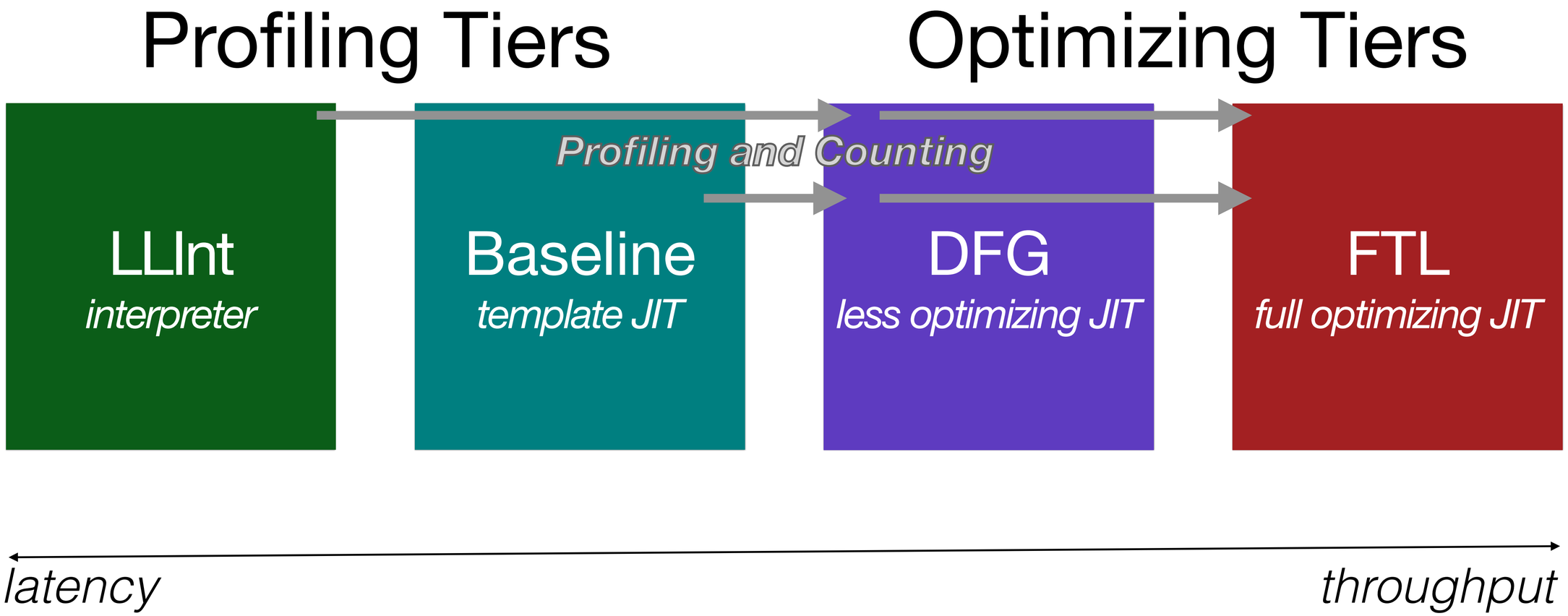
- High Level Overview
- Tiers
- Optimization Techniques
  - Counting Triggers
  - OSR (On Stack Replacement)
  - Profiling
  - Speculation
  - Inline Caching
  - Other Compiler Optimizations

Four JS Tiers

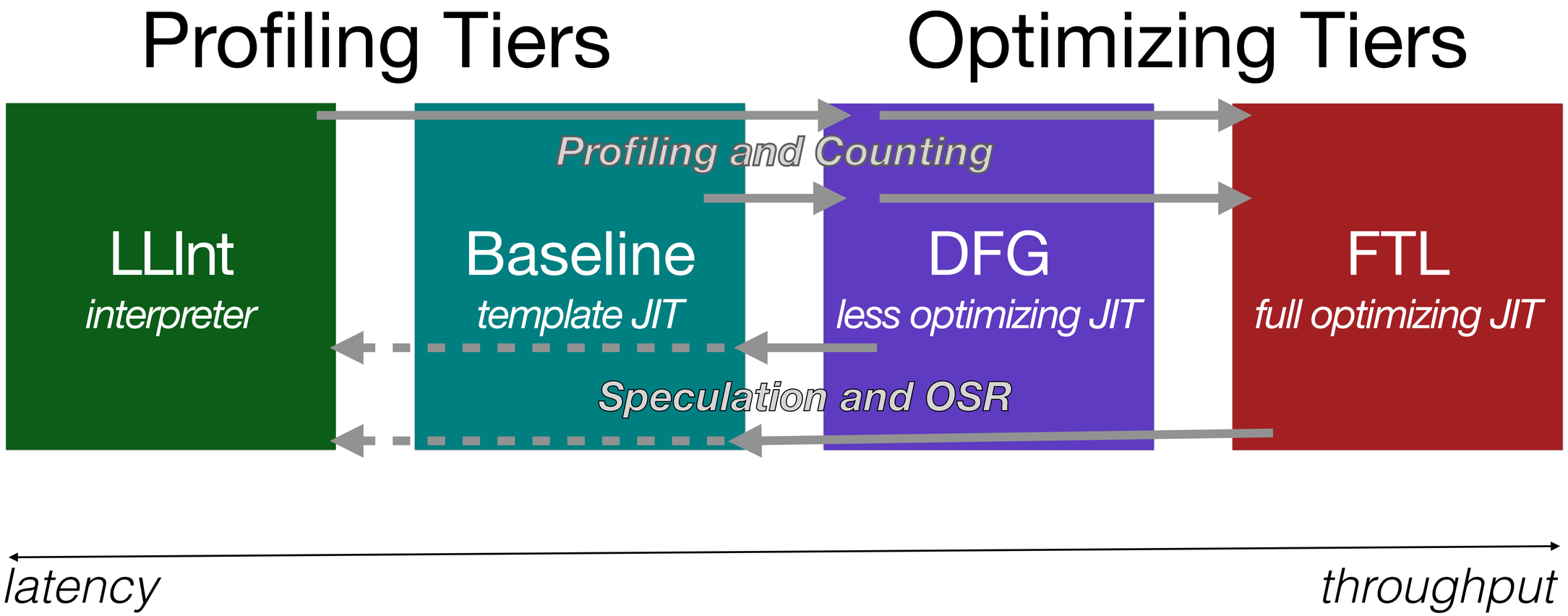
Four JS Tiers



Four JS Tiers



Four JS Tiers



Simple Profiling

- Record values, types, counts and flags
  - ✦ Low overhead
  - ✦ Provides enough detail
  - ✦ Occasionally we aggregate values and types into sets
  - ✦ Sets kept as a bit masks

Profiling Sources in JSC

- Value Profiling
- Inline Caches
- Case Flags
- Case Counts
- Watchpoints
- Exit Flags

Some Profiling Sources in JSC

- Value Profiling — *type inference of values*
- Inline Caches — *type inference of object structure*
- Case Flags — *branch speculation*
- Case Counts — *branch speculation*

Agenda

- High Level Overview
- Tiers
- Optimization Techniques
  - Counting Triggers
  - OSR (On Stack Replacement)
  - Profiling
  - Speculation
  - Inline Caching
  - Other Compiler Optimizations

Speculation Check

Speculation Check

```
speculateIsInt32(v)
{
    if (!isInt32(v))
        exitToUnoptimizedTier();
}
```

Speculation

- Speculation is making a bet
- Profiling helps us win that bet

Winning at Speculation

Winning at Speculation

- Only speculate if we believe that we will win every time.

Winning at Speculation

- Only speculate if we believe that we will win every time.
- Profiling should record counterexamples to useful speculations.

Winning at Speculation

- Only speculate if we believe that we will win every time.
- Profiling should record counterexamples to useful speculations.
- Profiling should run for a long time.

Winning at Speculation

- Only speculate if we believe that we will win every time.
- Profiling should record counterexamples to useful speculations.
- Profiling should run for a long time.
- Don’t stress when speculation fails, unless it fails in the average.

```
function addTo(o, v)
{
    o.sum += v;
}
```

Optimized JS function

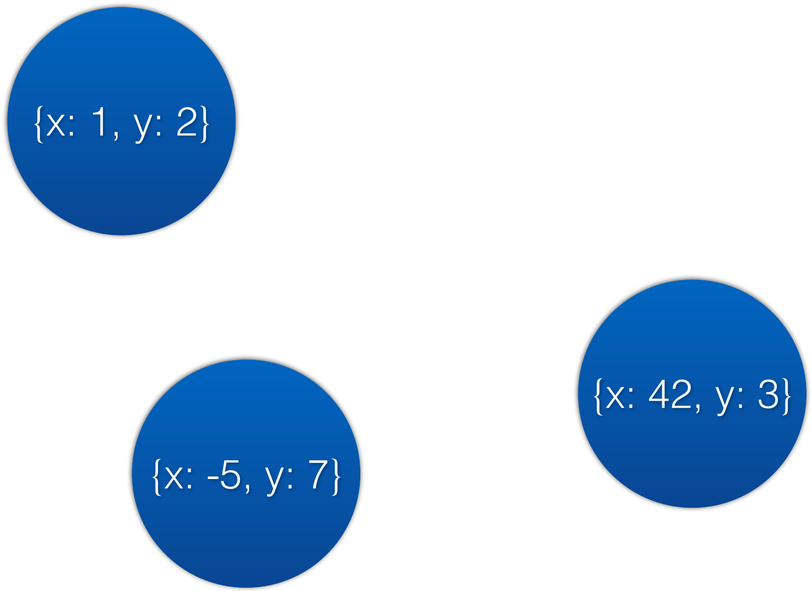
```
function addTo(o, v)
{
    speculateIsInt32(o.sum);
    speculateIsInt32(v);
    o.sum += v;
}
```

Agenda

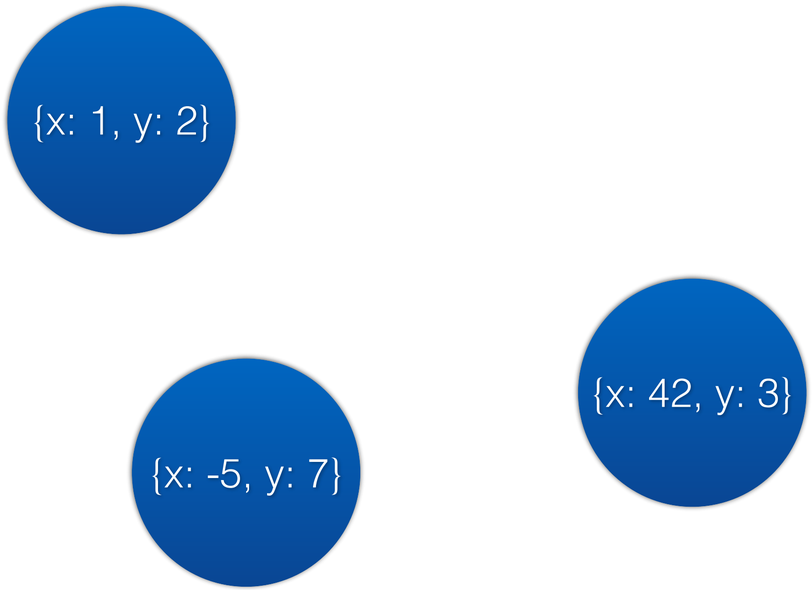
- High Level Overview
- Tiers
- Optimization Techniques
  - Counting Triggers
  - OSR (On Stack Replacement)
  - Profiling
  - Speculation
  - Inline Caching
  - Other Compiler Optimizations

```
function addTo(o, v)
{
    o.sum += v;
}
```

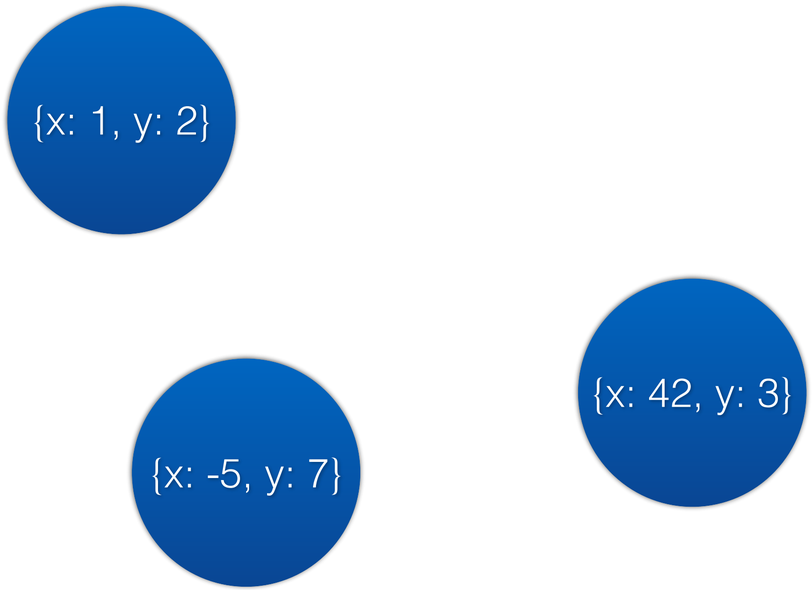
```
addTo:
[   0] enter
[   1] get_scope  loc4
[   3] mov        loc5, loc4
[   6] get_by_id  loc6, arg1, “sum”
[  11] add        loc6, loc6, arg2
[  17] put_by_id  arg1, “sum”, loc6
[  23] ret        Undefined
```


{x: 1, y: 2}
{x: -5, y: 7}
{x: 42, y: 3}

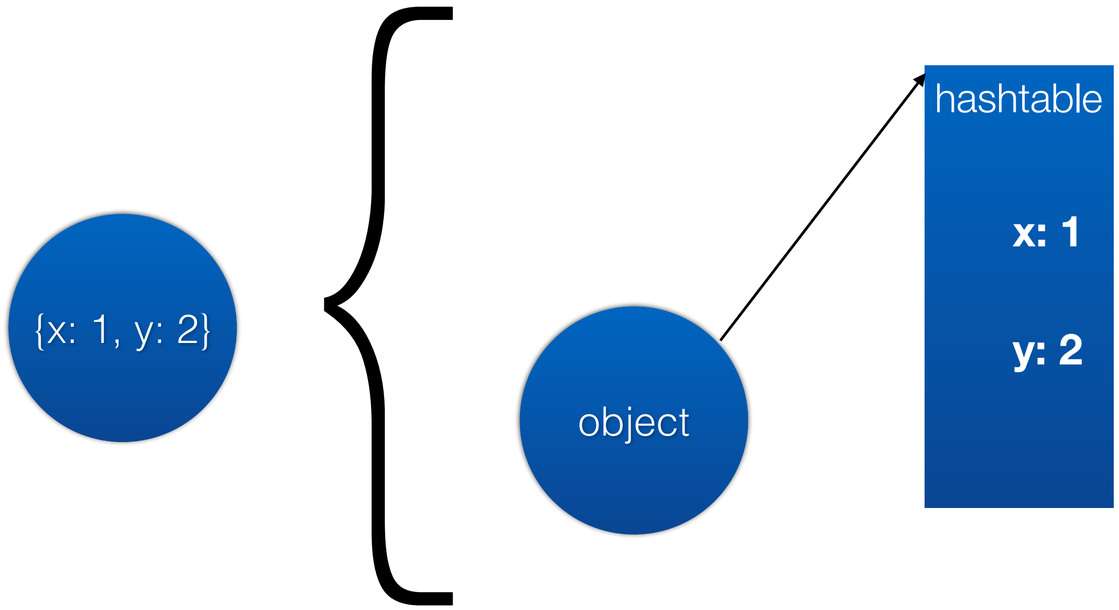
```
var x = o.x;
```



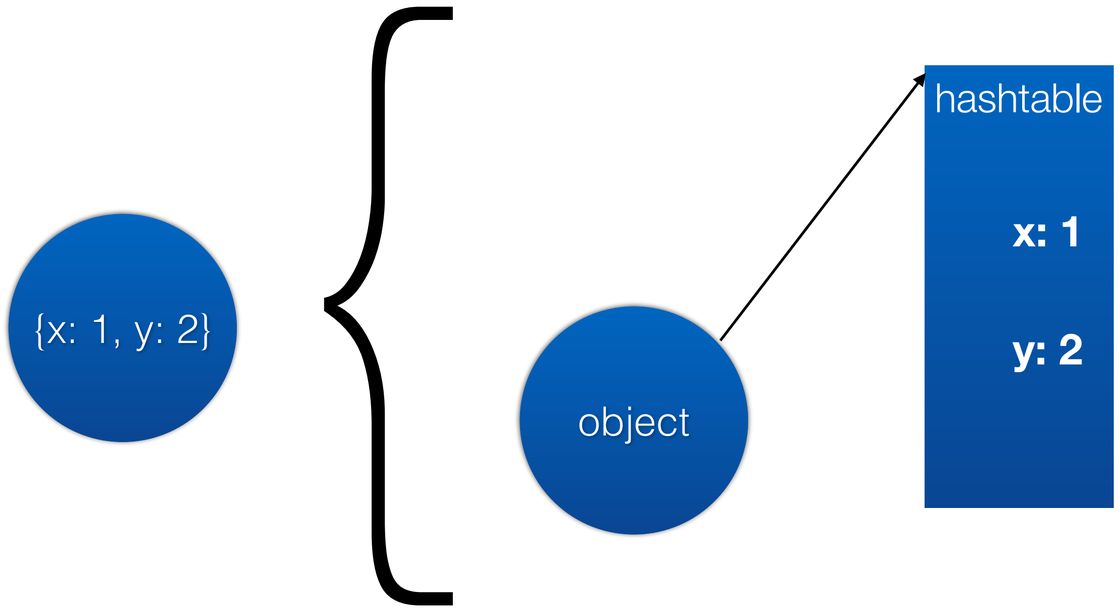
```
o.x = x;
```



Hashtable

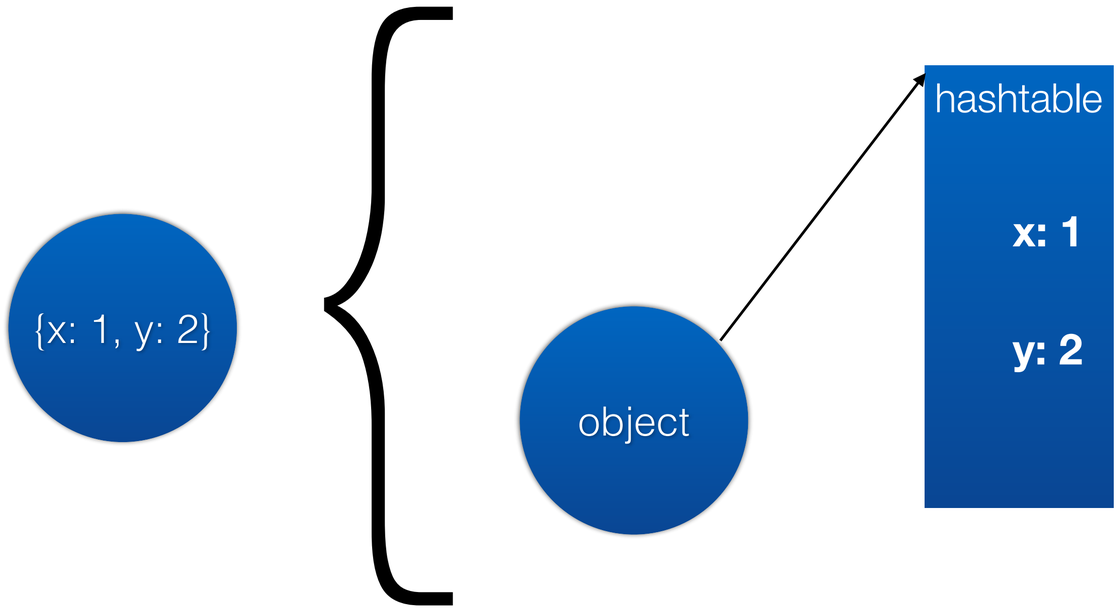


Hashtable



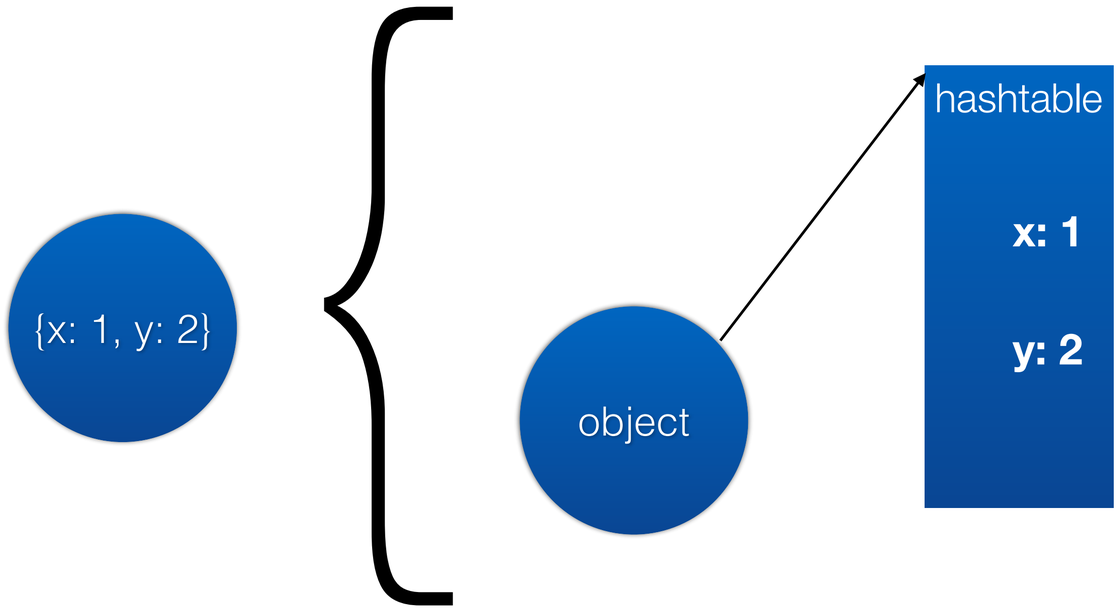
- Pointer chasing is slow

Hashtable



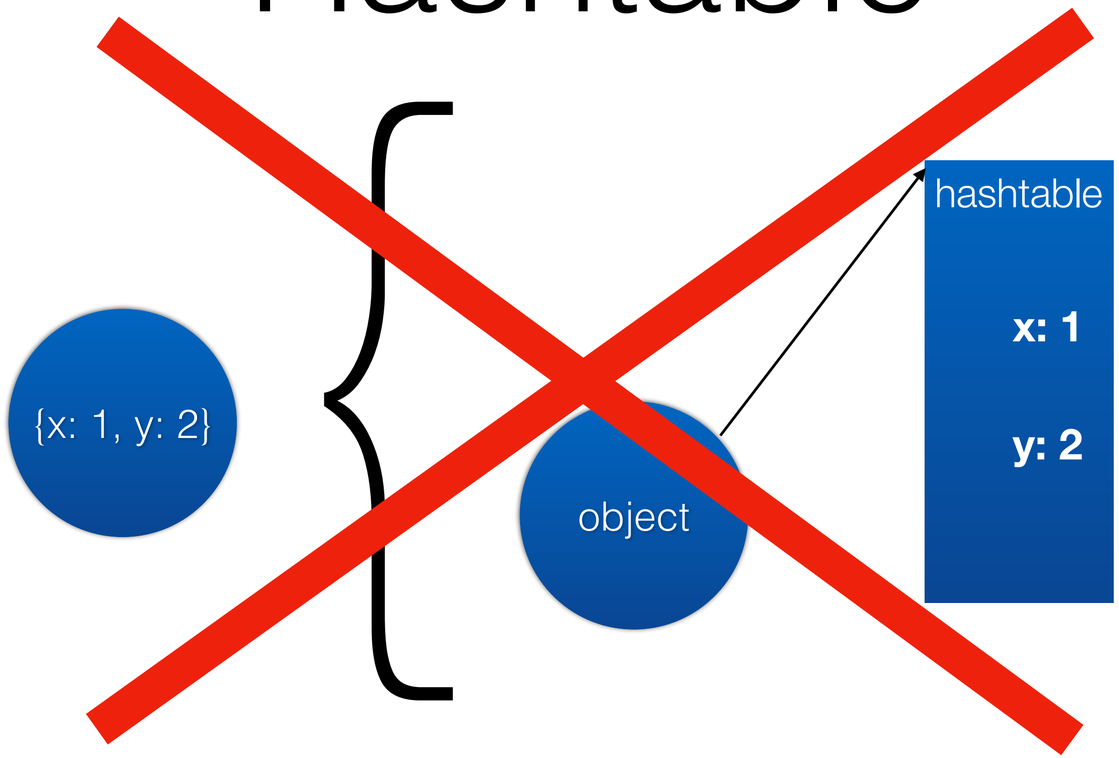
- Pointer chasing is slow
- Hash codes take time to compute

Hashtable

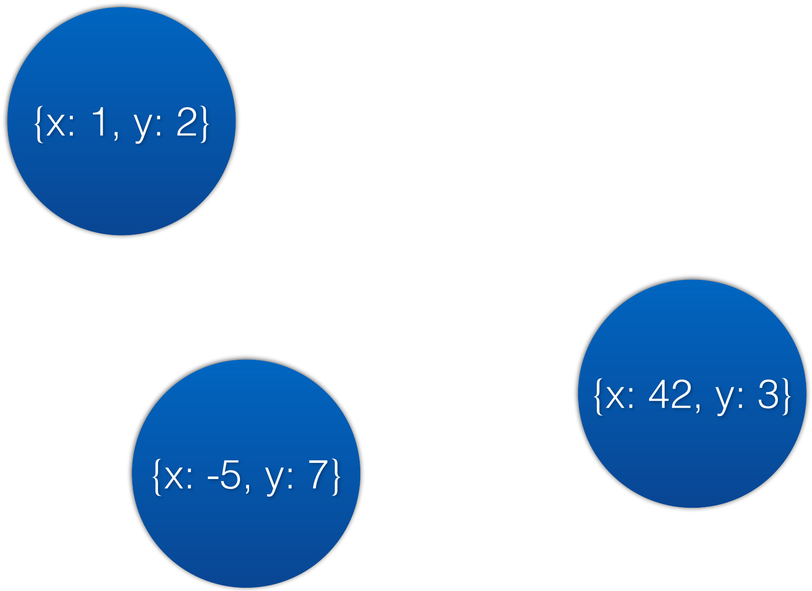


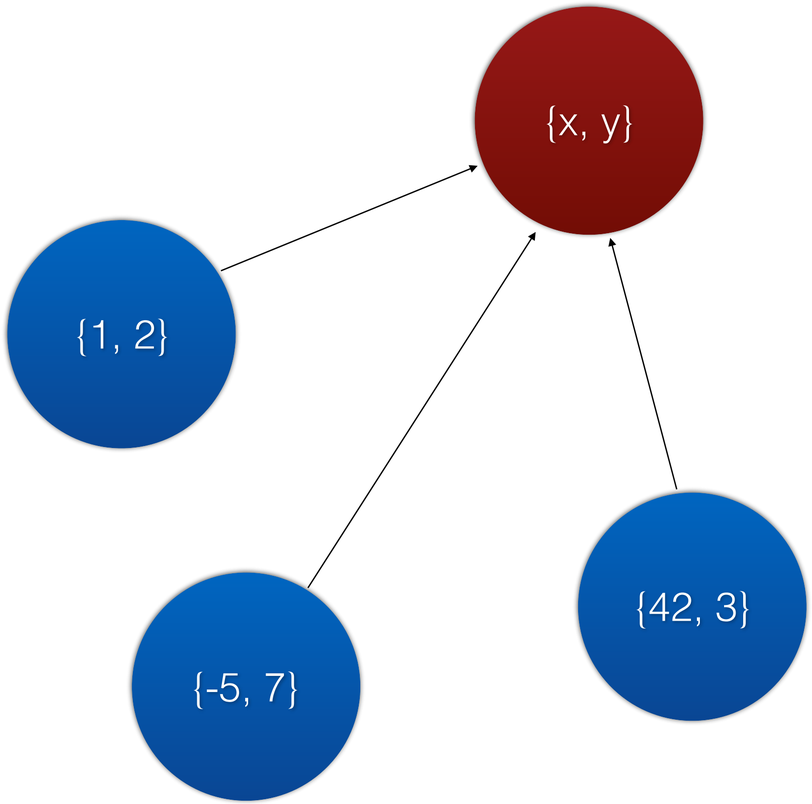
- Pointer chasing is slow
- Hash codes take time to compute
- Lots of instructions, hard to inline

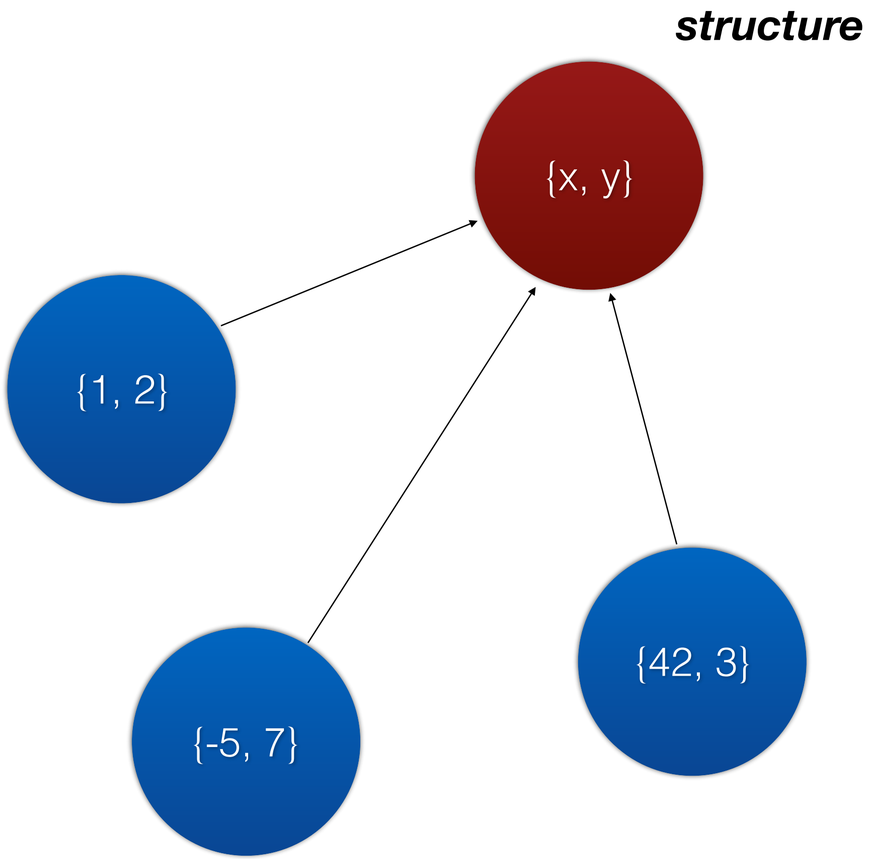
Hashtable



- Pointer chasing is slow
- Hash codes take time to compute
- Lots of instructions, hard to inline

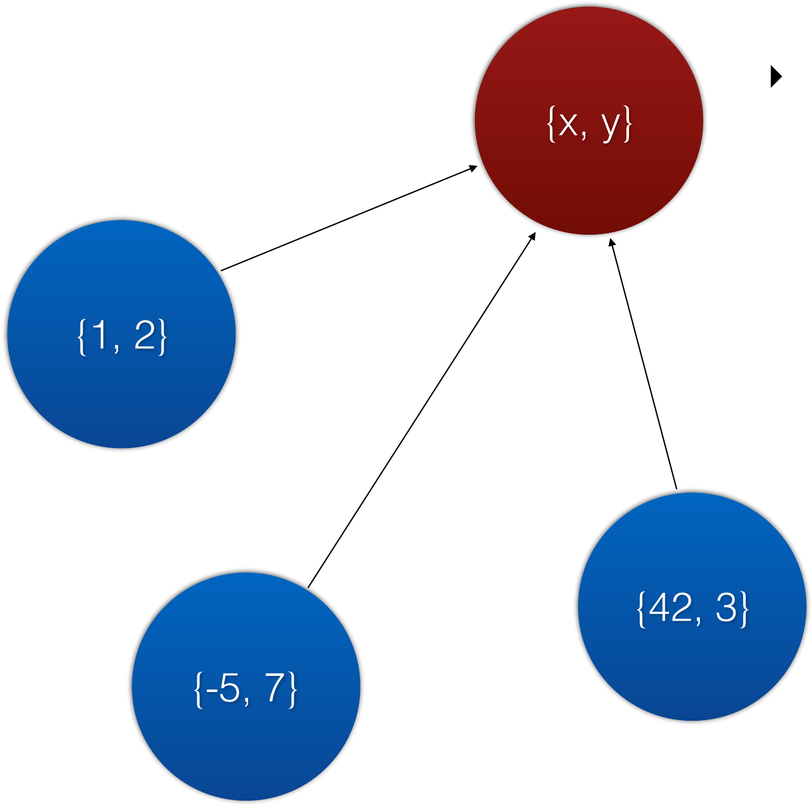

{x: 1, y: 2}
{x: -5, y: 7}
{x: 42, y: 3}

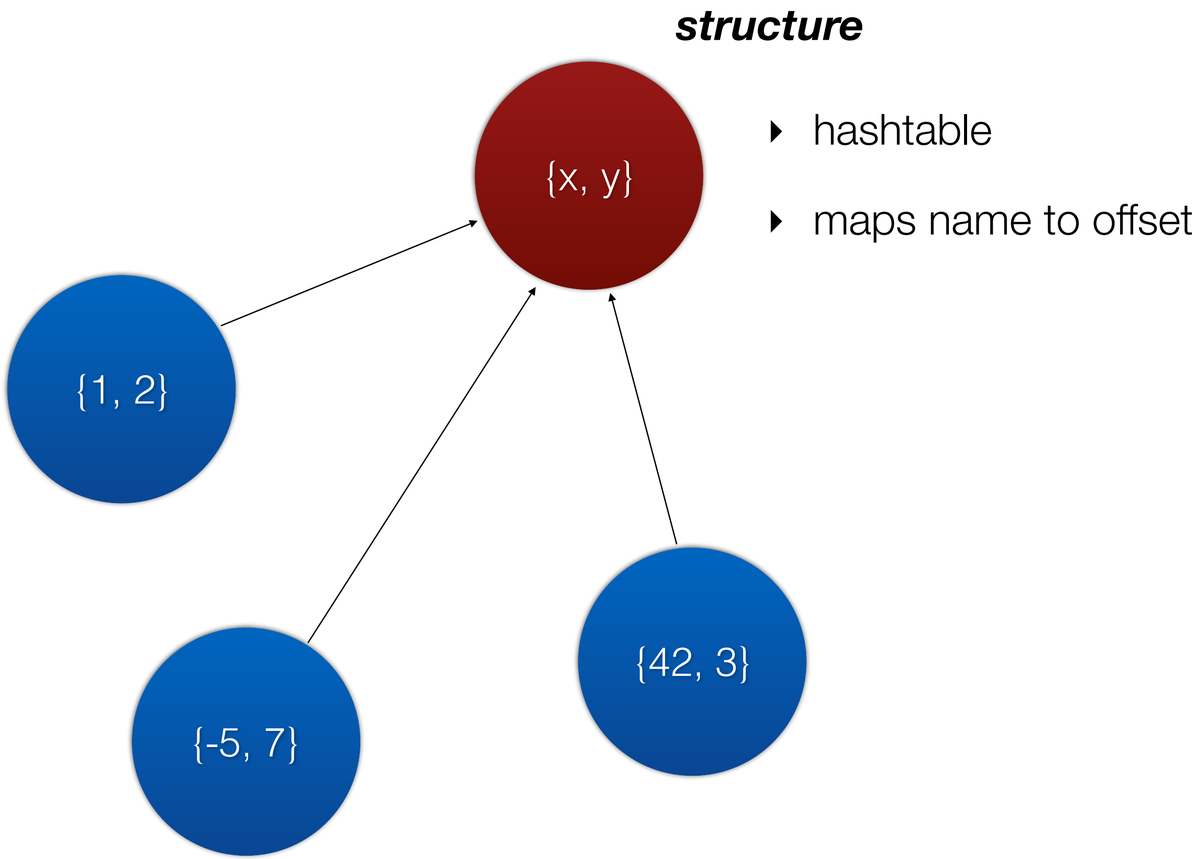

{x, y}
{1, 2}
{-5, 7}
{42, 3}


structure
{x, y}
{1, 2}
{-5, 7}
{42, 3}

structure

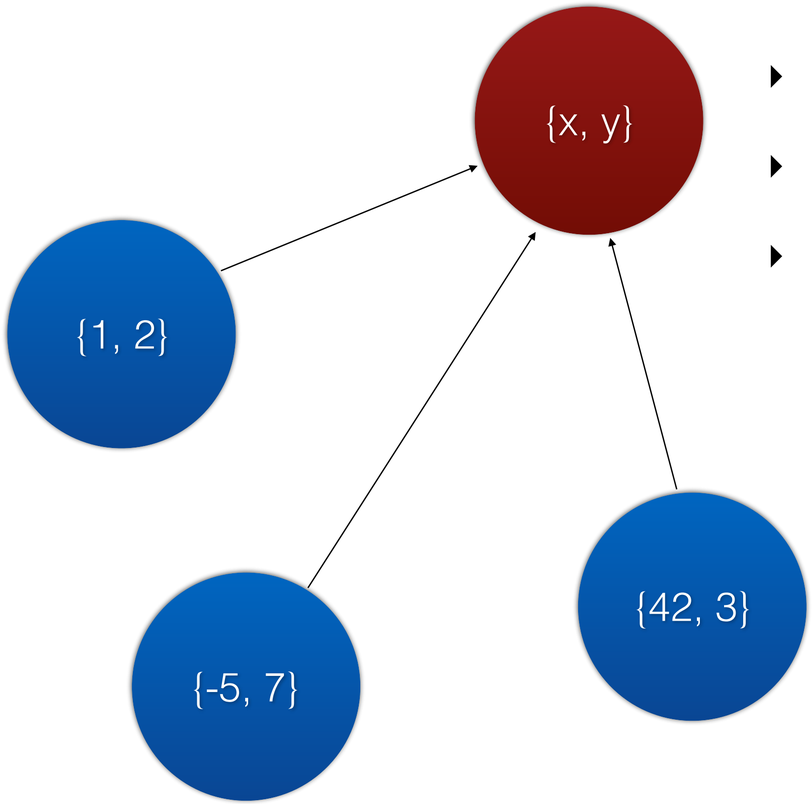
- hashtable





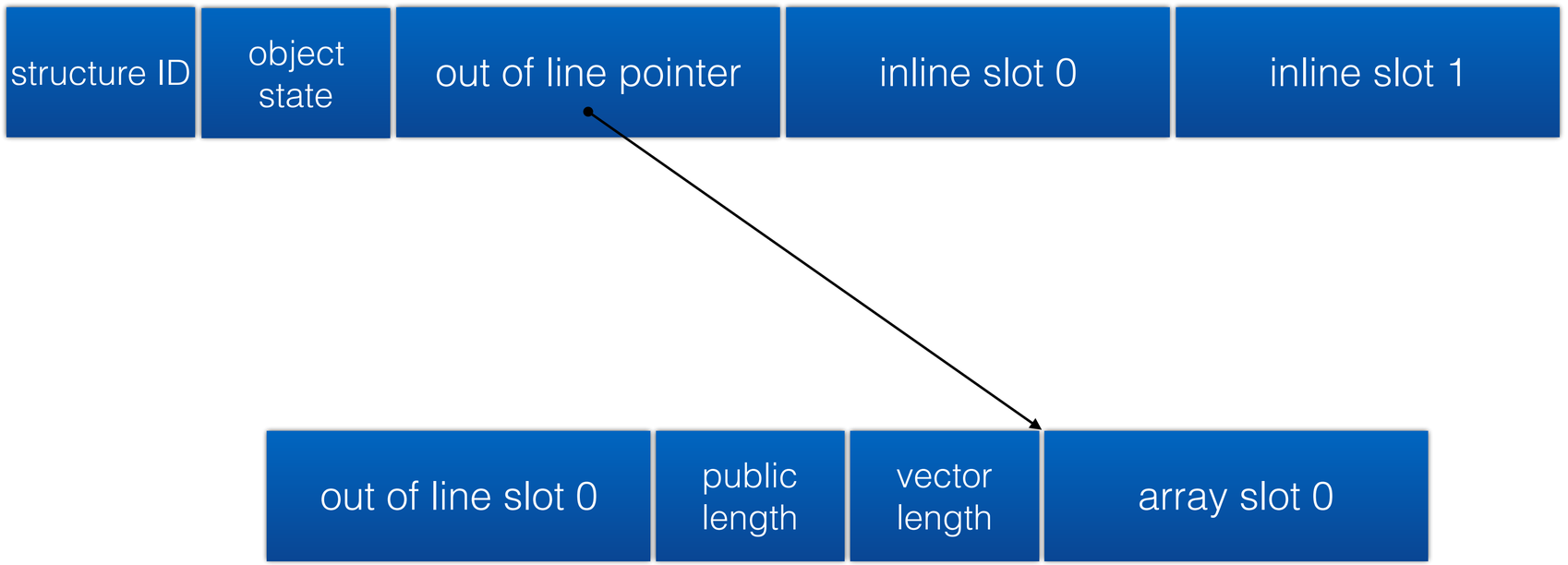
- hashtable
- maps name to offset

structure

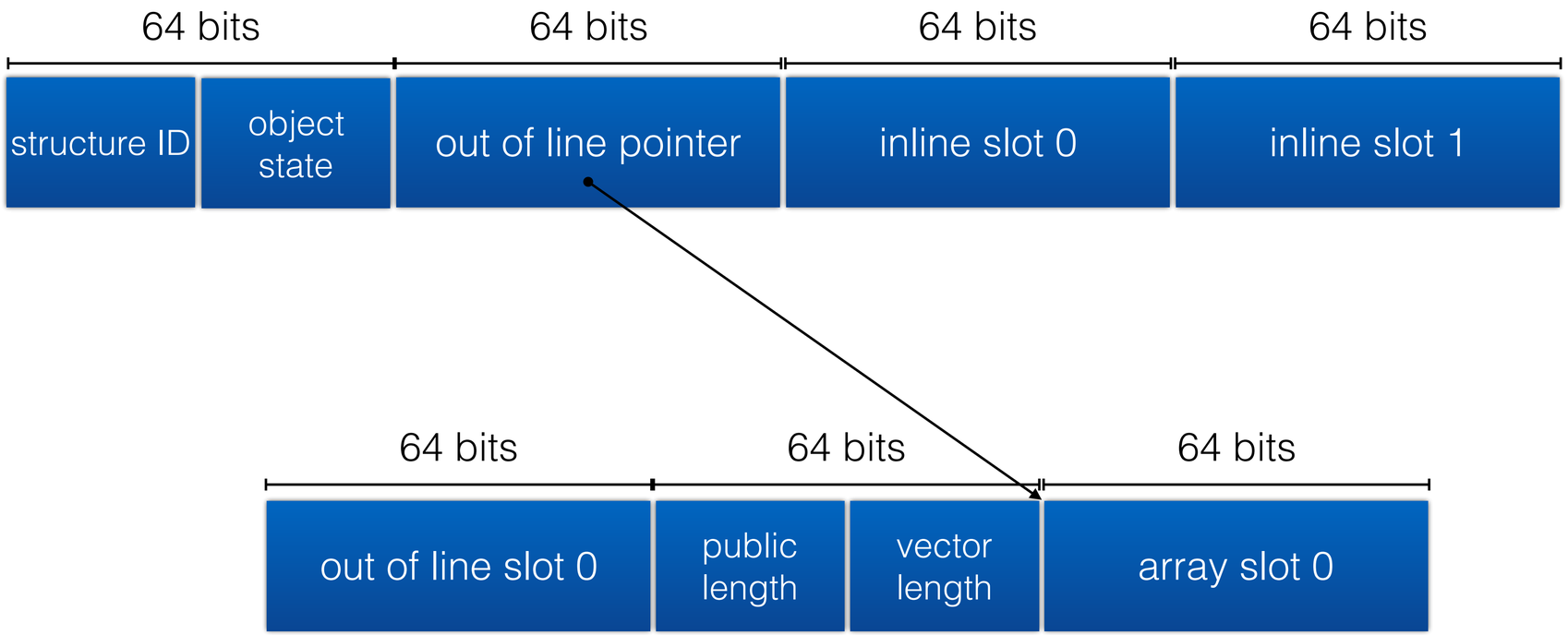


- hashtable
- maps name to offset
- hash-consed

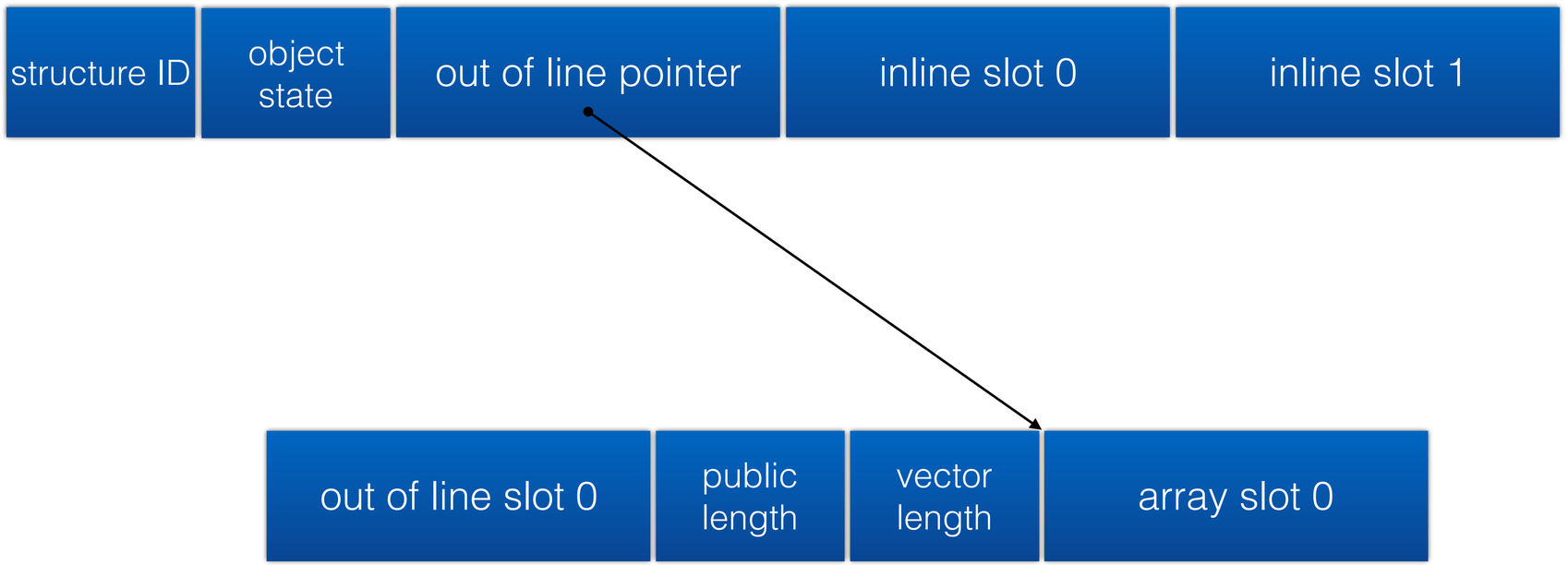
JSC Object Model



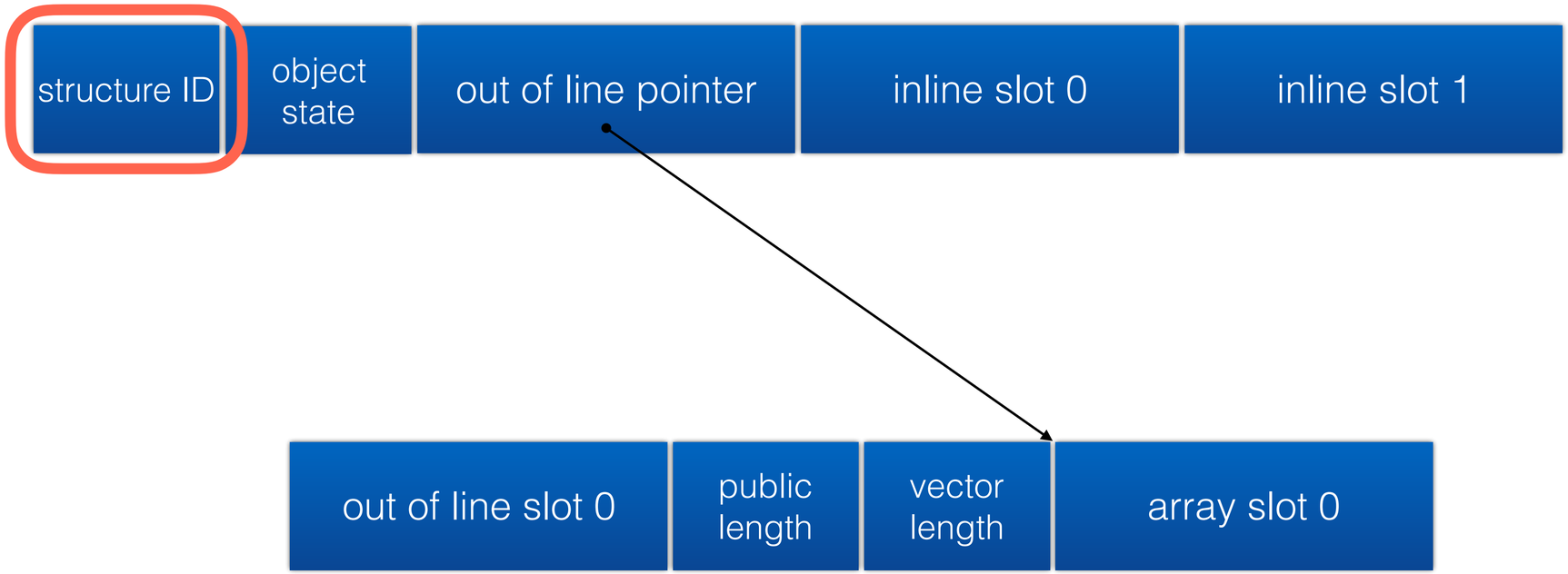
JSC Object Model



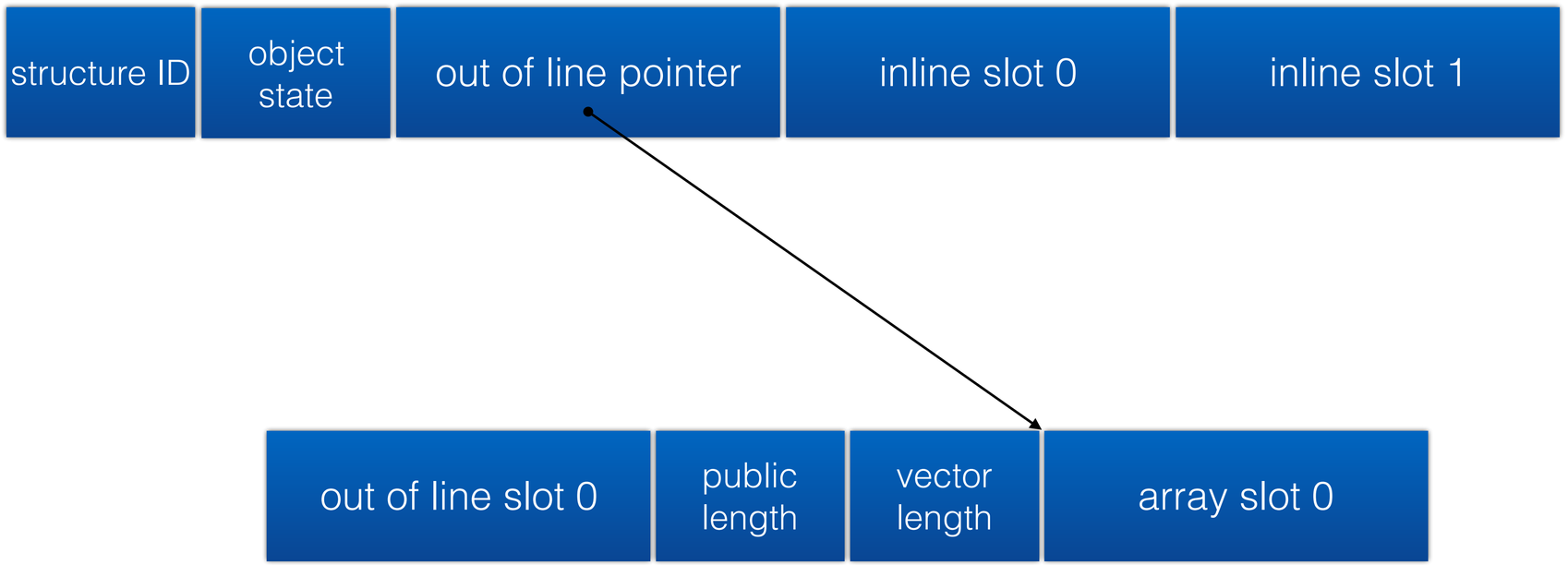
JSC Object Model



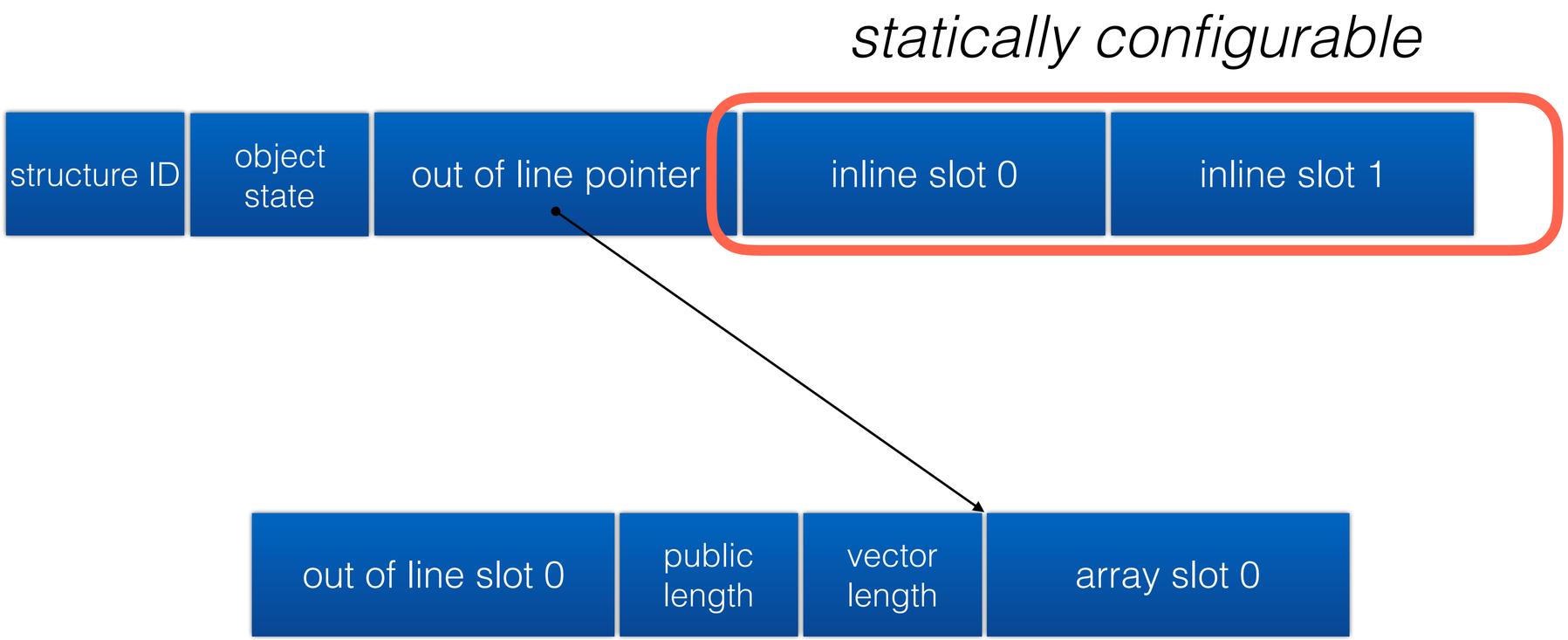
JSC Object Model



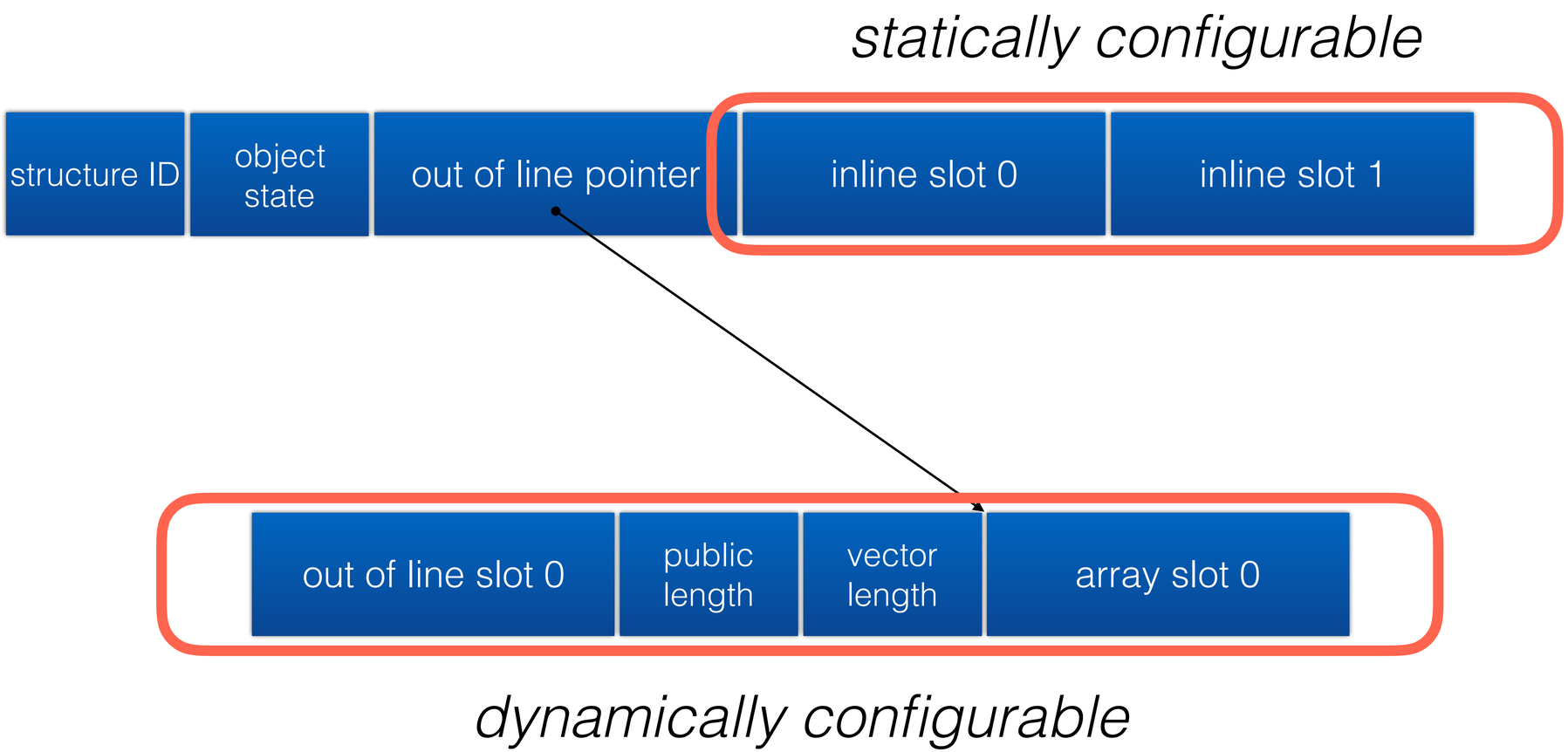
JSC Object Model



JSC Object Model



JSC Object Model



Fast JSObject

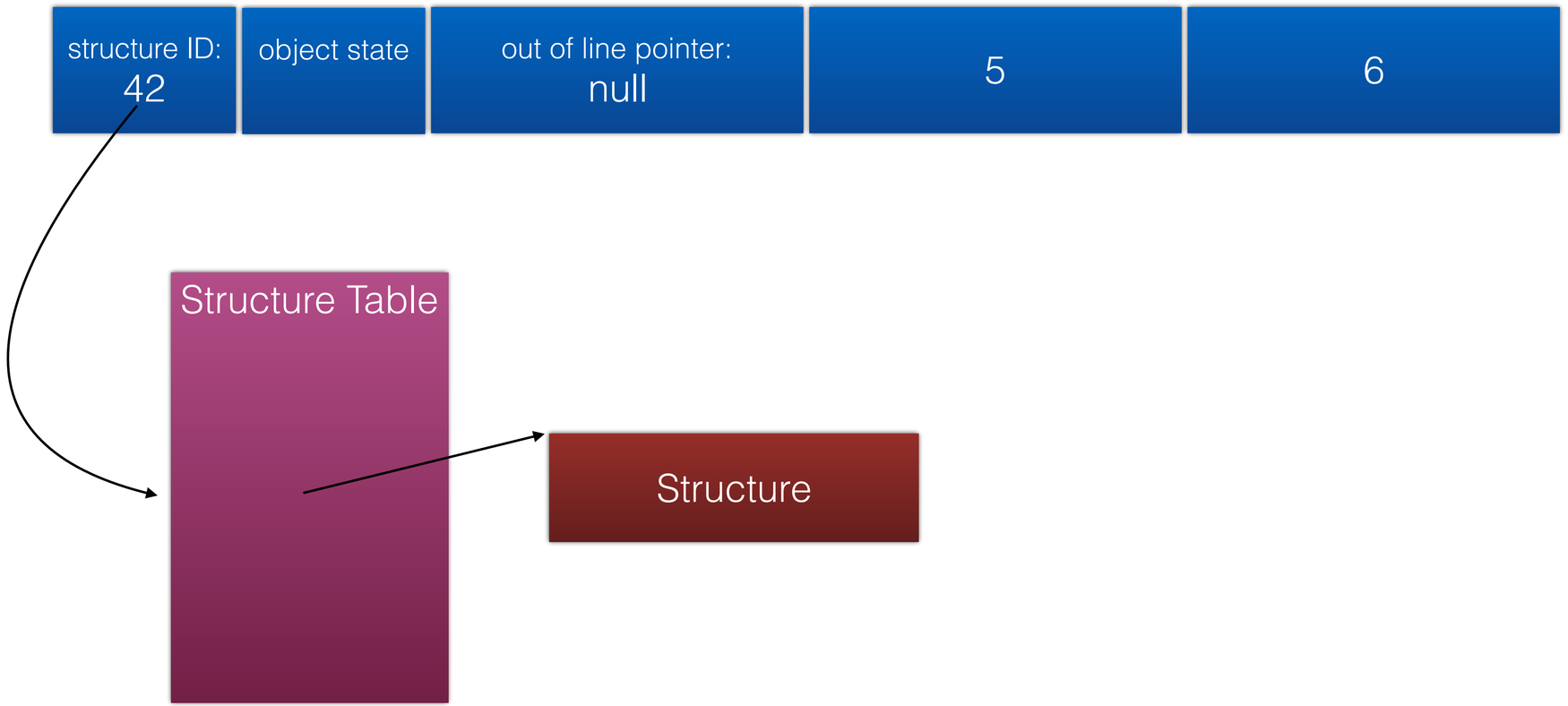
var o = {x: 5, y: 6};

Fast JSObject

var o = {x: 5, y: 6};

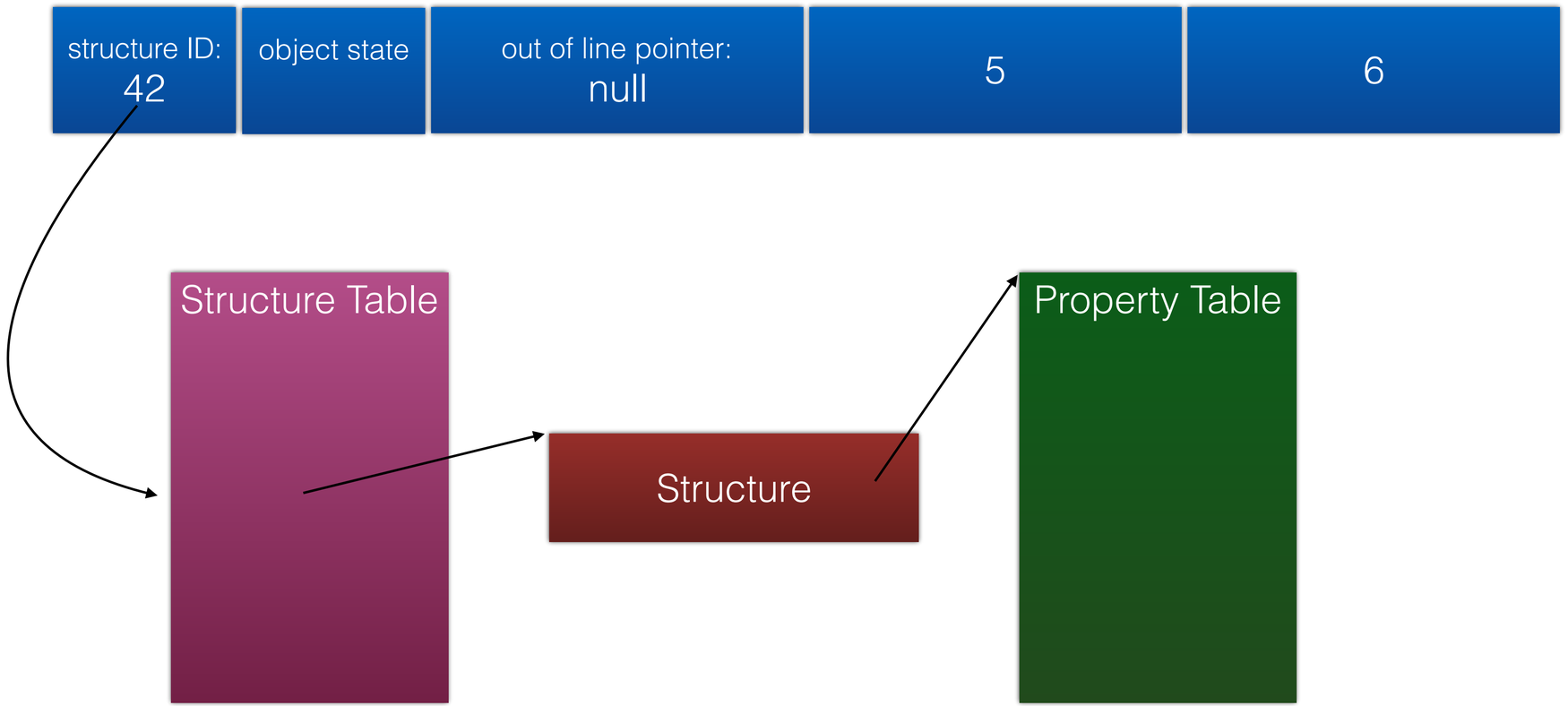
Fast JSObject

var o = {x: 5, y: 6};



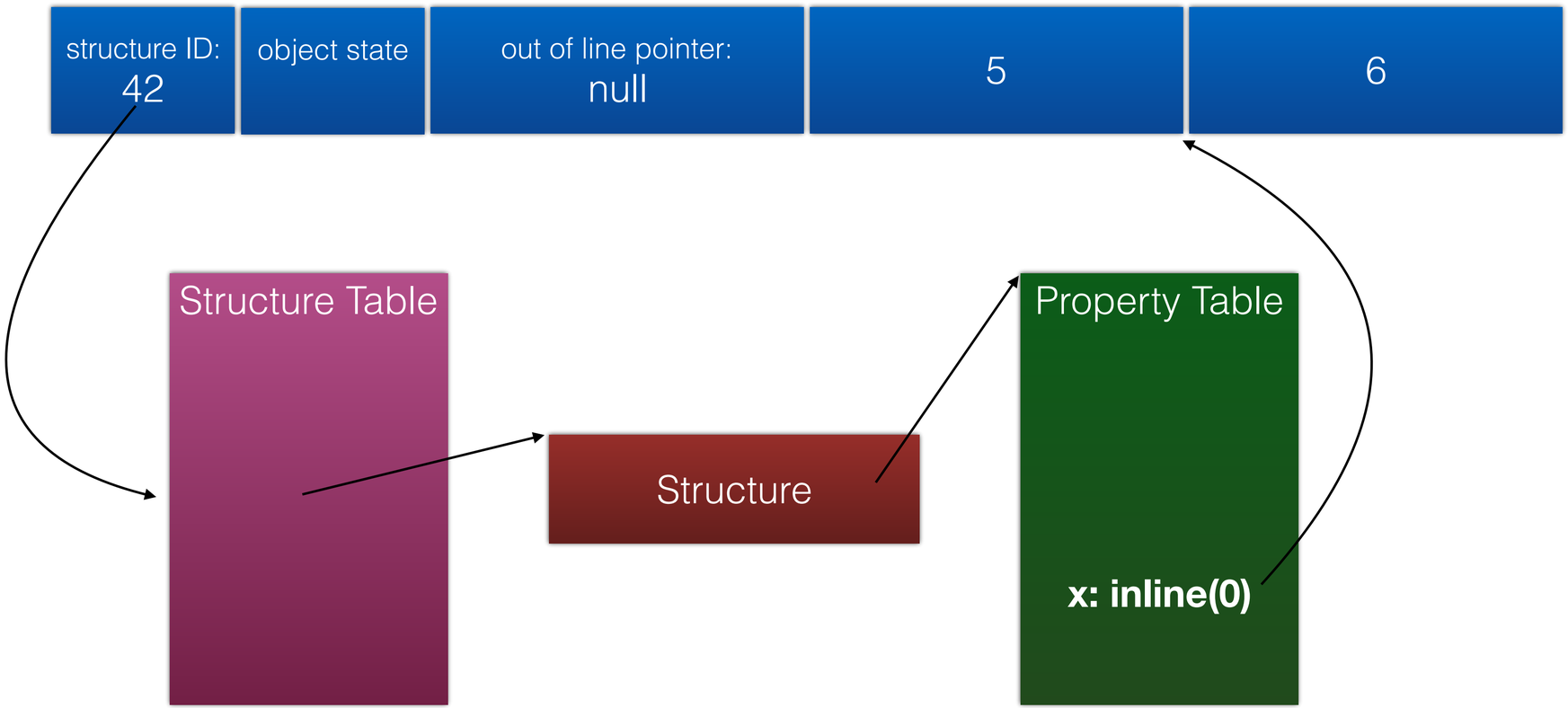
Fast JSObject

var o = {x: 5, y: 6};



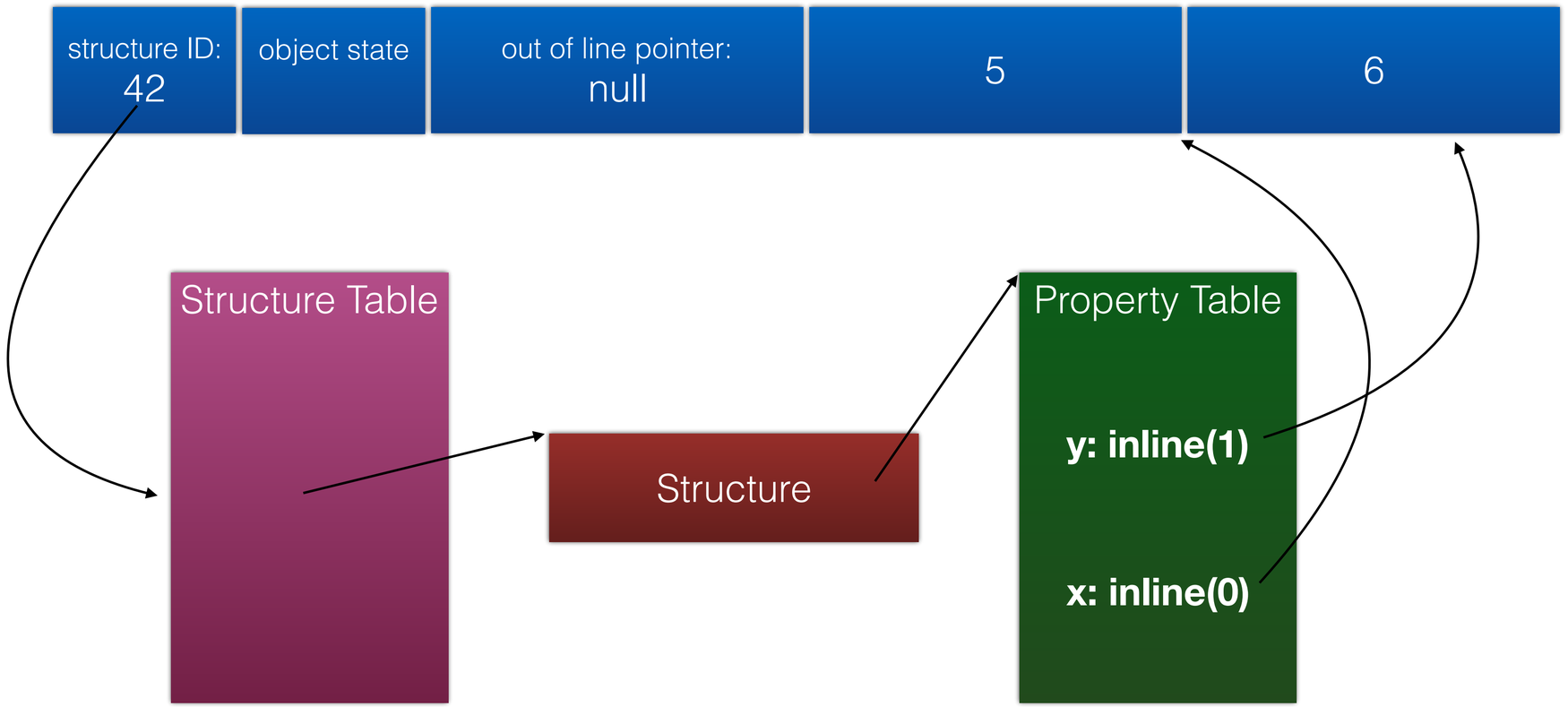
Fast JSObject

var o = {x: 5, y: 6};

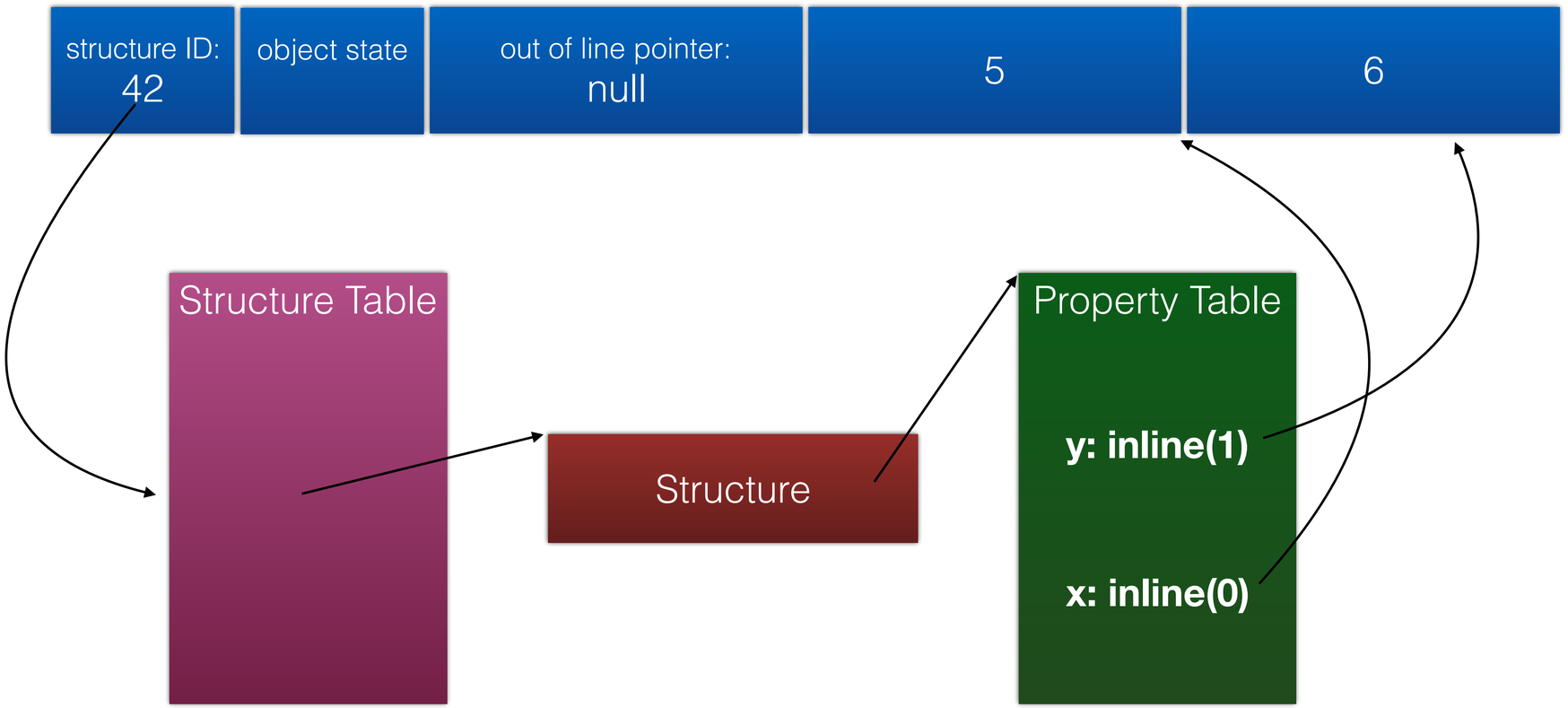


Fast JSObject

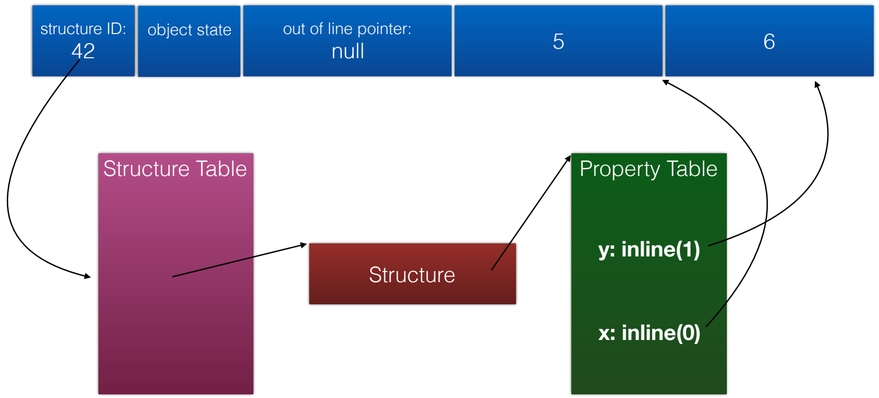
var o = {x: 5, y: 6};



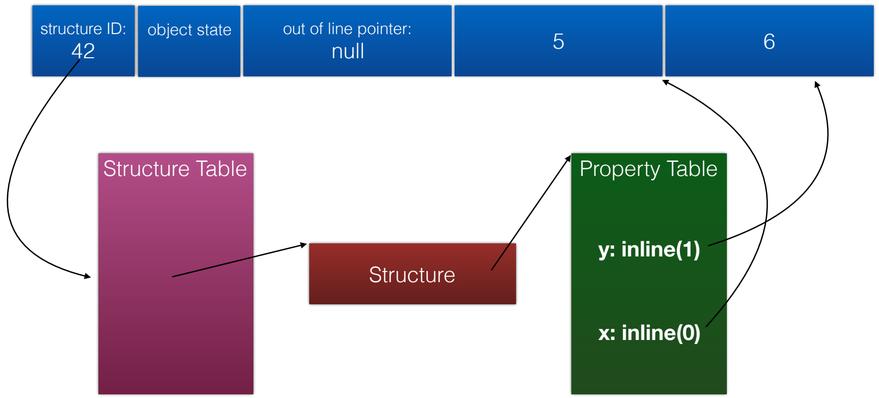
var o = {x: 5, y: 6};


structure ID:
42
object state
out of line pointer:
null
5
6
Structure Table
Structure
Property Table
y: inline(1)
x: inline(0)

var o = {x: 5, y: 6};

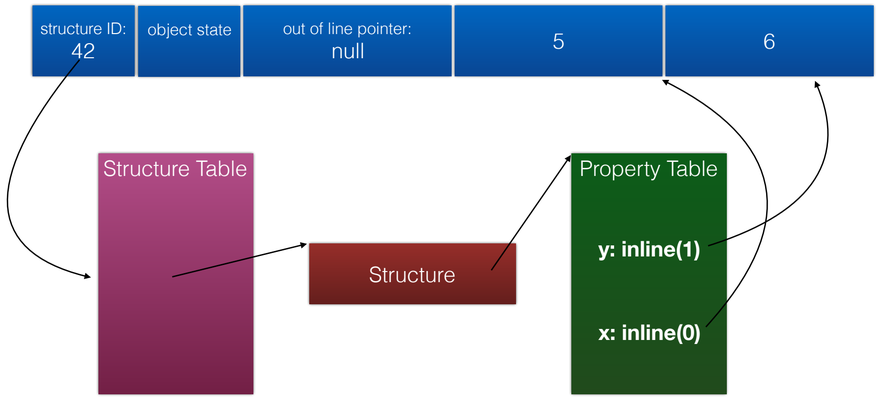

structure ID:
42
object state
out of line pointer:
null
5
6
Structure Table
Structure
Property Table
y: inline(1)
x: inline(0)

var o = {x: 5, y: 6};




```
var v = o.x;
```

var o = {x: 5, y: 6};

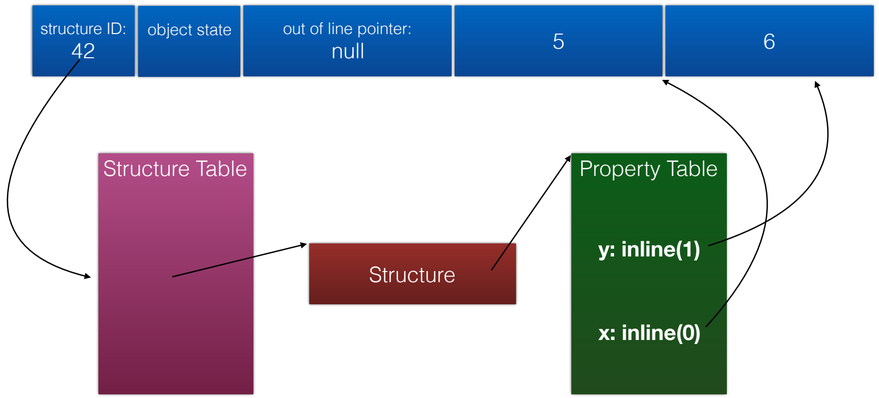



```
var v = o.x;
```

```
if (o->structureID == 42)
    v = o->inlineStorage[0]
else
    v = slowGet(o, “x”)
```

“Inline Cache”

var o = {x: 5, y: 6};




```
var v = o.x;
```

```
if (o->structureID == 42)
    v = o->inlineStorage[0]
else
    v = slowGet(o, “x”)
```

Interpreter Inline Cache

```
get_by_id <result>, <base>, <properyName>
```

Interpreter Inline Cache

```
get_by_id <result>, <base>, <properyName>,
          <cachedStructureID>, <cachedOffset>
```

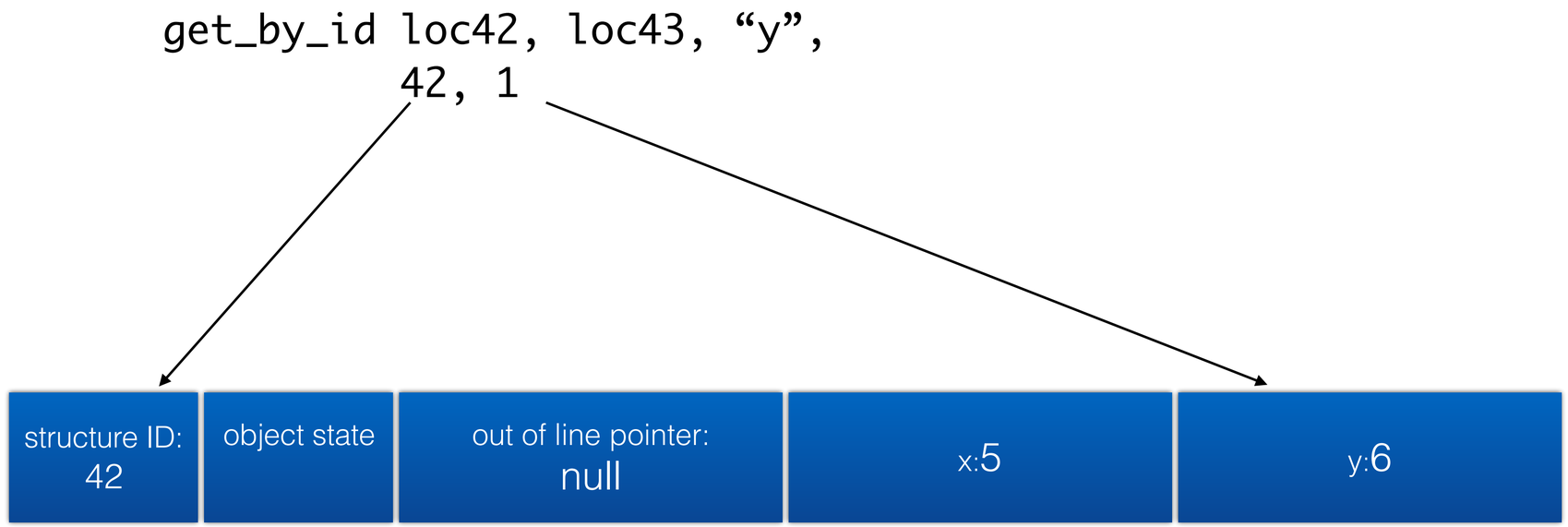
Interpreter Inline Cache

```
get_by_id loc42, loc43, “y”,
          0, 0
```

Interpreter Inline Cache

```
get_by_id loc42, loc43, “y”,
          0, 0
```

Interpreter Inline Cache



JIT Inline Cache

```
0x46f8c30b9b0: mov 0x30(%rbp), %rax
0x46f8c30b9b4: test %rax, %r15
0x46f8c30b9b7: jnz 0x46f8c30ba2c
0x46f8c30b9bd: jmp 0x46f8c30ba2c
0x46f8c30b9c2: o16 nop %cs:0x200(%rax,%rax)
0x46f8c30b9d1: nop (%rax)
0x46f8c30b9d4: mov %rax, -0x38(%rbp)
```

JIT Inline Cache

```
0x46f8c30b9b0: mov 0x30(%rbp), %rax
0x46f8c30b9b4: test %rax, %r15
0x46f8c30b9b7: jnz 0x46f8c30ba2c
0x46f8c30b9bd: jmp 0x46f8c30ba2c
0x46f8c30b9c2: o16 nop %cs:0x200(%rax,%rax)
0x46f8c30b9d1: nop (%rax)
0x46f8c30b9d4: mov %rax, -0x38(%rbp)
```

JIT Inline Cache

```
0x46f8c30b9b0: mov 0x30(%rbp), %rax
0x46f8c30b9b4: test %rax, %r15
0x46f8c30b9b7: jnz 0x46f8c30ba2c
0x46f8c30b9bd: jmp 0x46f8c30ba2c
0x46f8c30b9c2: o16 nop %cs:0x200(%rax,%rax)
0x46f8c30b9d1: nop (%rax)
0x46f8c30b9d4: mov %rax, -0x38(%rbp)
```

JIT Inline Cache

```
0x46f8c30b9b0: mov 0x30(%rbp), %rax
0x46f8c30b9b4: test %rax, %r15
0x46f8c30b9b7: jnz 0x46f8c30ba2c
0x46f8c30b9bd: cmp $0x2a, (%rax)
0x46f8c30b9c3: jnz 0x46f8c30ba2c
0x46f8c30b9c9: mov 0x18(%rax), %rax
0x46f8c30b9cd: nop 0x200(%rax)
0x46f8c30b9d4: mov %rax, -0x38(%rbp)
```

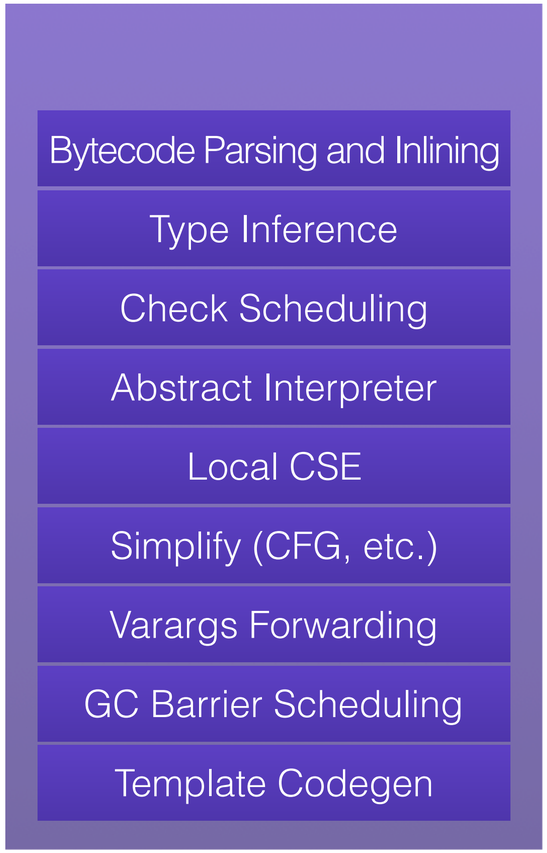
Agenda

- High Level Overview
- Tiers
- Optimization Techniques
  - Counting Triggers
  - OSR (On Stack Replacement)
  - Profiling
  - Speculation
  - Inline Caching
  - Other Compiler Optimizations

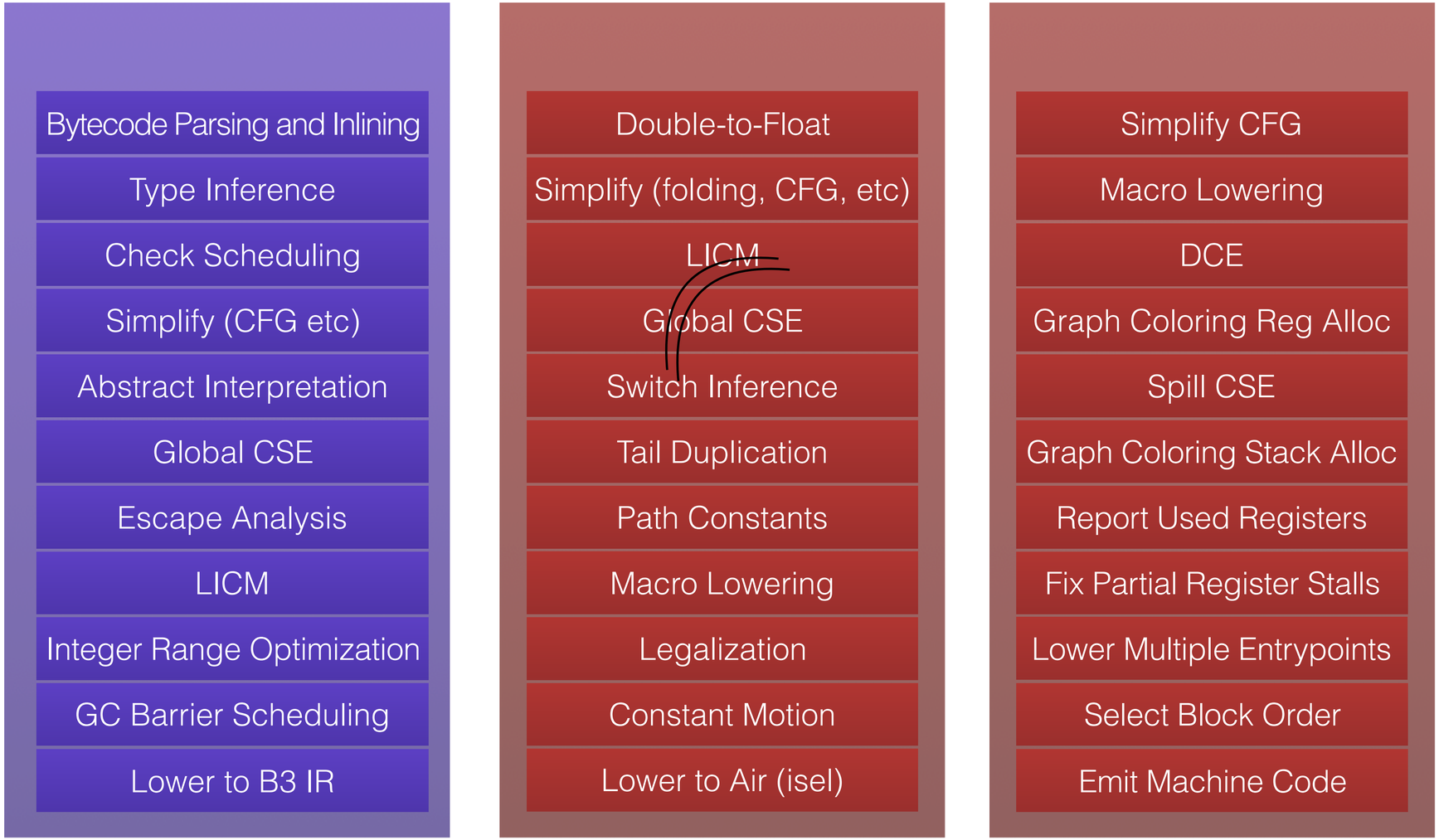
Optimizations

- Generatorification
- Inlining
- Strength Reduction
- CSE (local and global)
- LICM
- Type/Bounds/Overflow Check Removal
- Object Allocation Sinking
- Arguments/Varargs Elimination
- Sparse Conditional Constant Propagation
- Barrier Placement
- Strength Reduction
- Tail Duplication
- Switch Inference
- Float Inference
- DCE
- Register Allocation
  - Linear Scan
  - Briggs
  - Iterated Register Coalescing
- Stack Allocation

DFG optimization pipeline



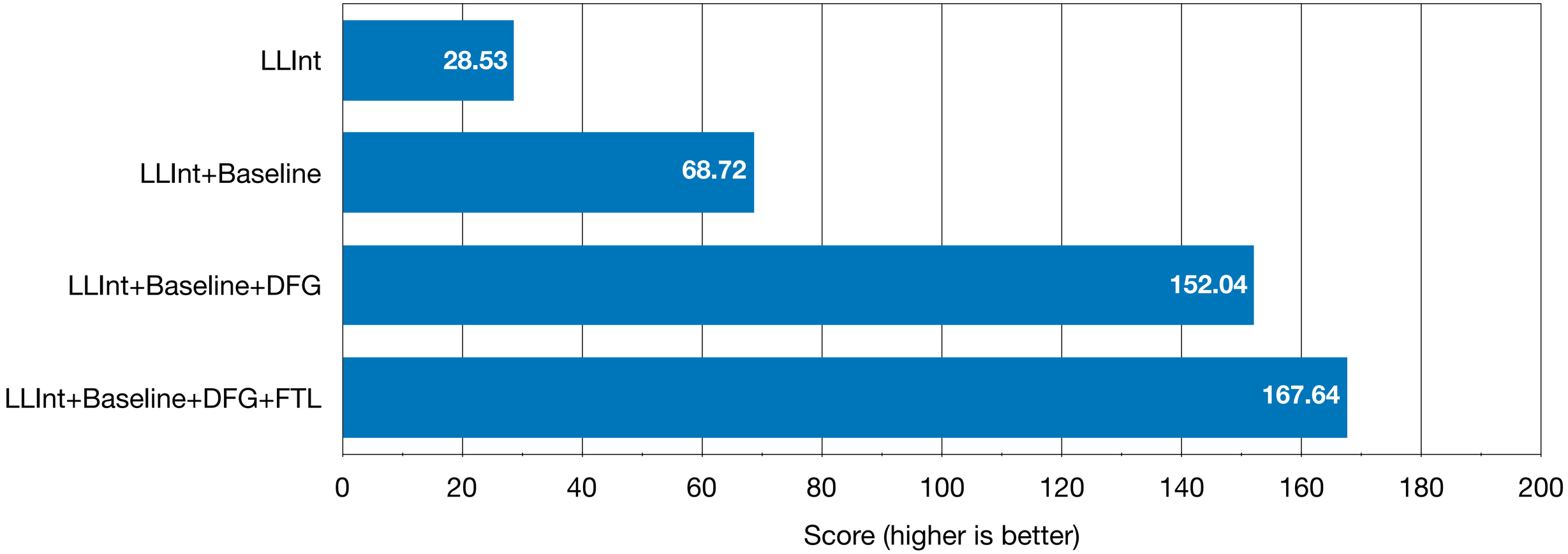
FTL optimization pipeline



Results

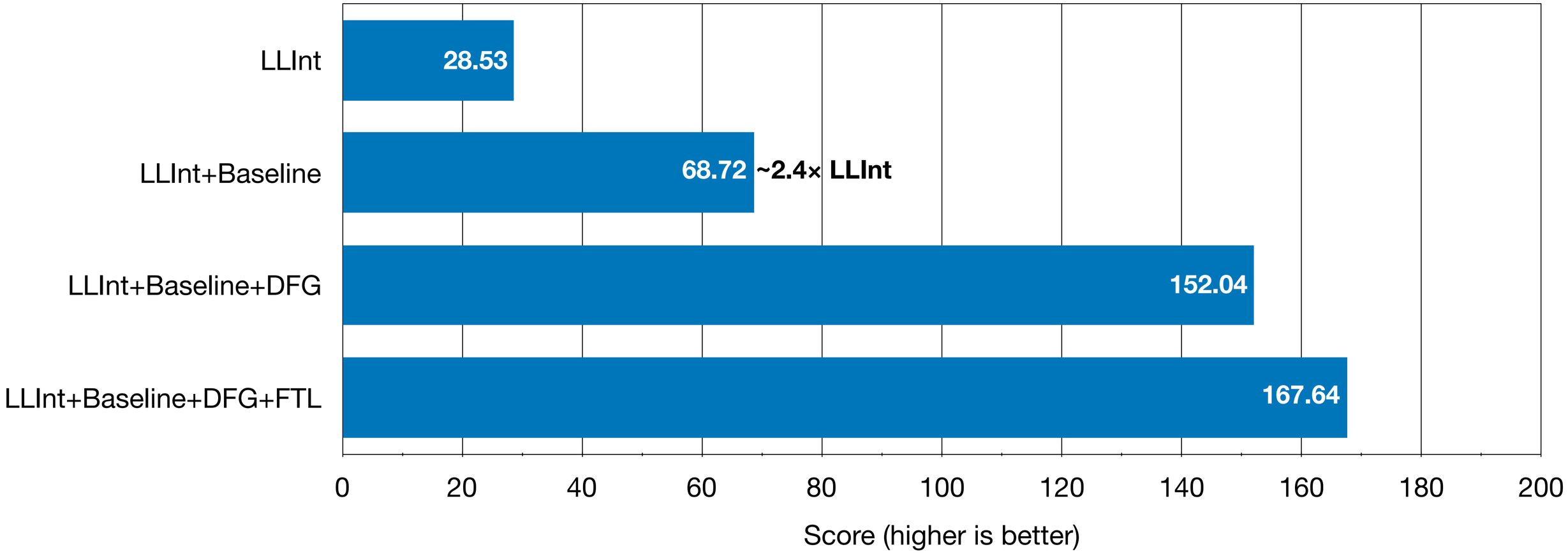
JetStream 2 Score

on my computer one day



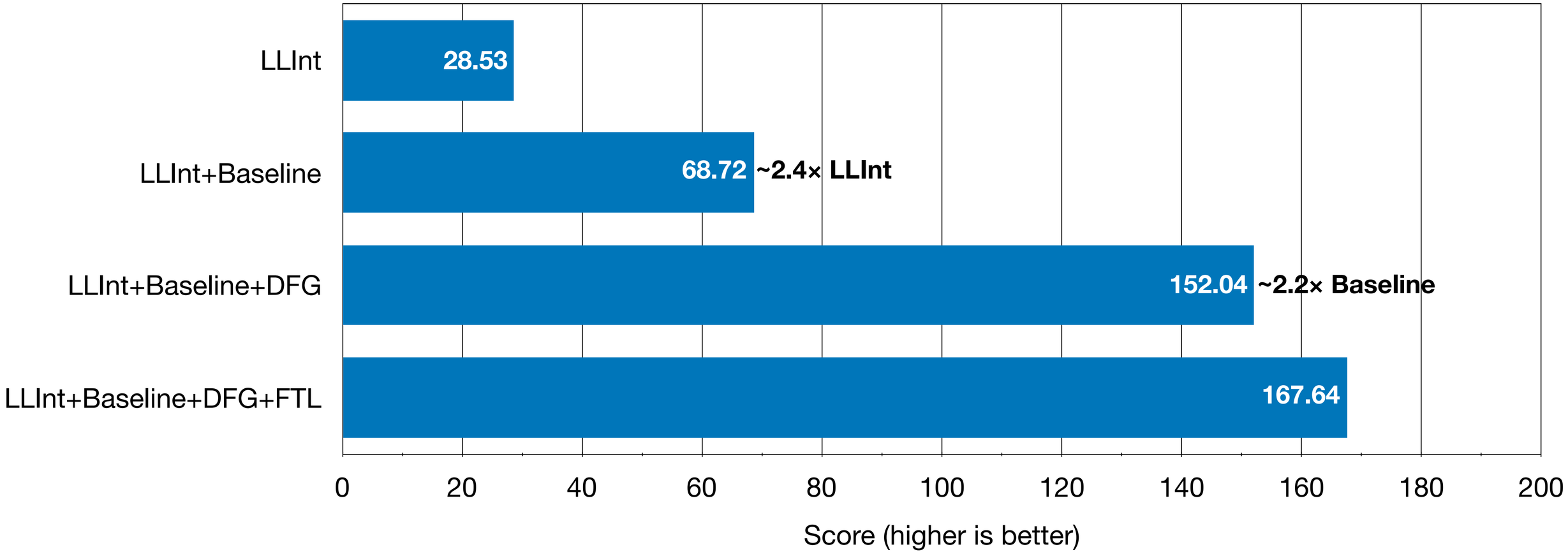
JetStream 2 Score

on my computer one day



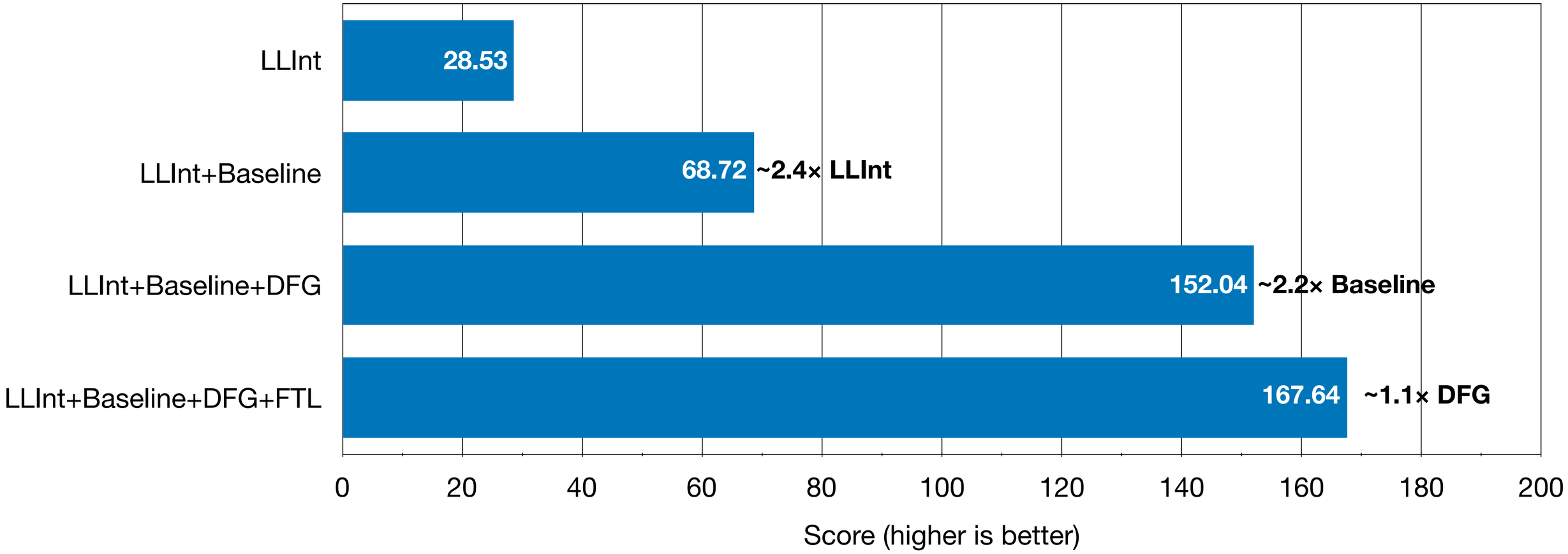
JetStream 2 Score

on my computer one day



JetStream 2 Score

on my computer one day



Questions?

JavaScriptCore, Many Compilers Make this Engine Perform

Michael Saboff
Apple Inc.