Real World Garbage Collection

Filip Pizlo
Apple

JavaScriptCore

"JSC" for short

webkit.org

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

Safari

Agenda

- Introduction
- JavaScriptCore
- Efficient Mark-Sweep
    - *30 minute break*
- Concurrent GC
- bmalloc
- WTF::Lock

Introduction

Riptide

Concurrent

retreating wavefront

Guy L Steele

“Multiprocessing compactifying garbage collection”

Parallel

Generational

Sticky mark bits

Demers, Weiser, Hayes, Boehm, Bobrow, Shenker

POPL’90

Non-copying

Bump'n'pop allocation

bump'n'pop ~ reaps

Berger, Zorn, and McKinley

https://people.cs.umass.edu/~emery/pubs/berger-oopsla2002.pdf

Conservative-on-the-stack

Conservative-on-the-stack

Related things:

http://www.hboehm.info/gc/tree.html

http://www.hpl.hp.com/techreports/Compaq-DEC/WRL-88-2.pdf

Constraint-based

Efficient

Riptide

- Concurrent
- Parallel
- Generational
- Non-copying
- Bump’n’pop
- Conservative-on-the-stack
- Constraint-based
- Efficient

webkit.org/blog/7122

Agenda

- ~~Introduction~~
- JavaScriptCore
- Efficient Mark-Sweep
    - *30 minute break*
- Concurrent GC
- bmalloc
- WTF::Lock

JavaScriptCore

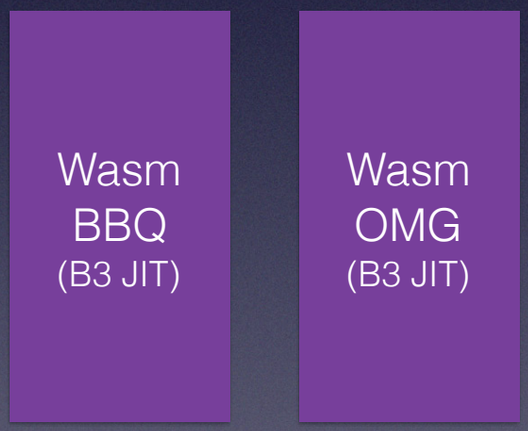
JavaScriptCore

- Supports ES2015+ and WebAssembly
- One interpreter and seven JITs
- Structures
- OSR and Watchpoints
- Designed for Embedding
- Concurrent, Parallel, and Generational GC

One Interpreter and Seven JITs

JavaScript execution engines:

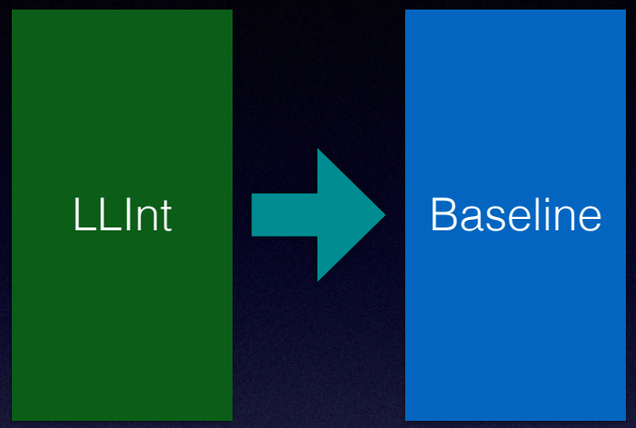
WebAssembly execution engines:

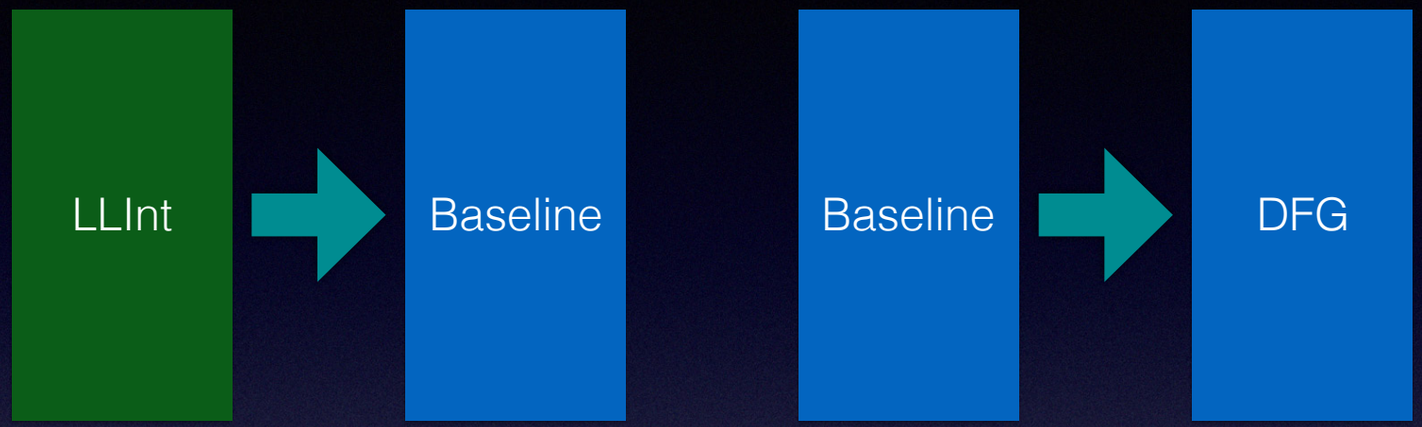


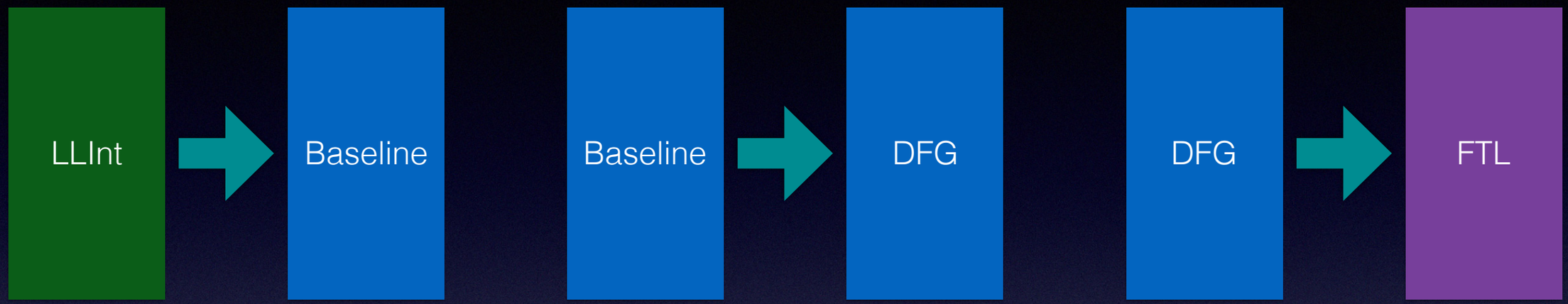
Why so many?

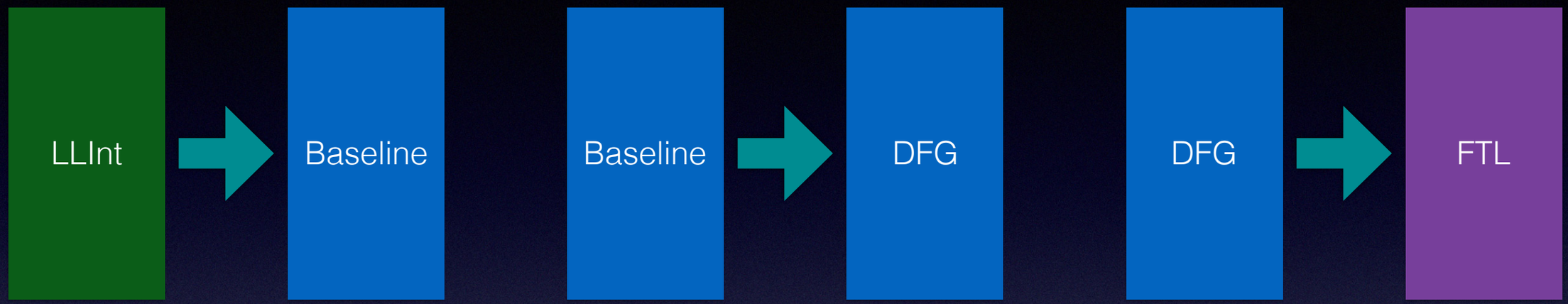
- Lots of run-once code (LLInt)
- Lots of lukewarm compute code (Baseline and DFG JITs)
- Some long-running code (DFG and FTL JITs)
- Dynamic features (PolymorphicAccess and Snippet JITs)

OSR

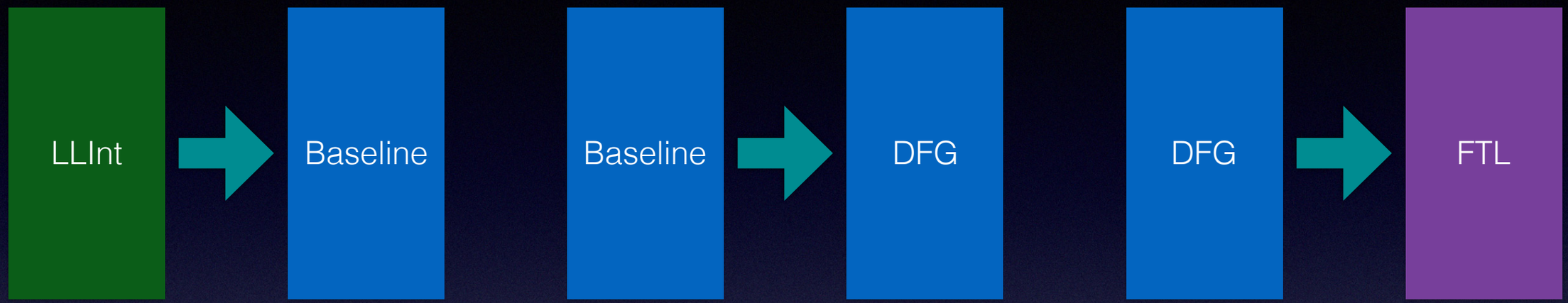

LLInt
Baseline

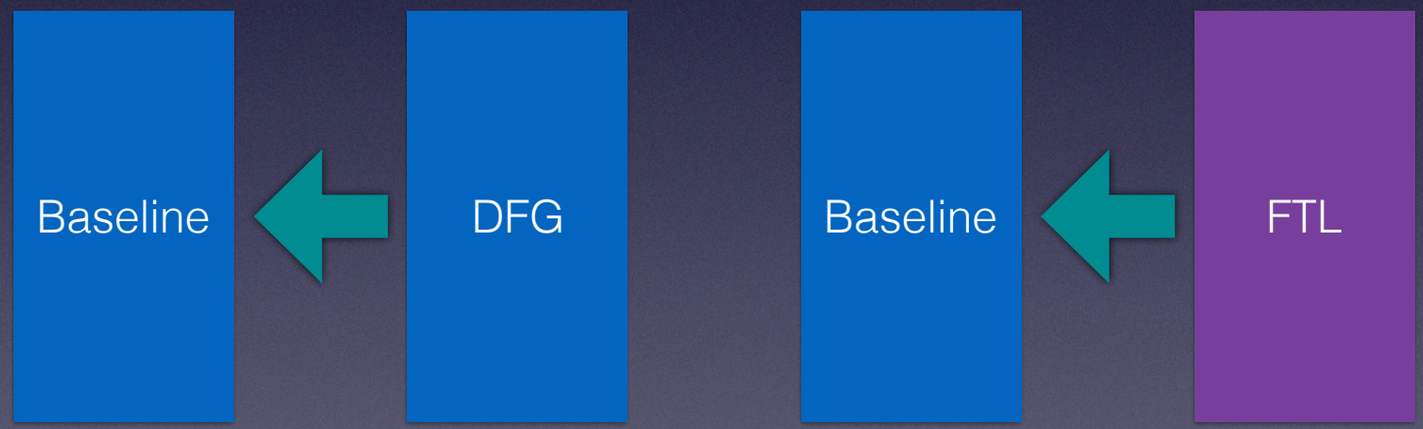

LLInt
Baseline
Baseline
DFG

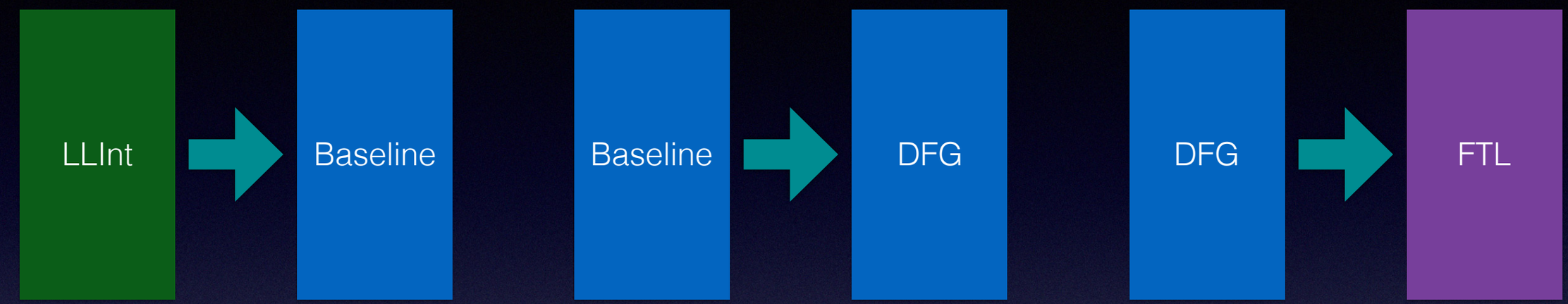

LLInt
Baseline
Baseline
DFG
DFG
FTL


LLInt
Baseline
Baseline
DFG
DFG
FTL


Baseline
DFG


LLInt
Baseline
Baseline
DFG
DFG
FTL


Baseline
DFG
Baseline
FTL


LLInt
Baseline
Baseline
DFG
DFG
FTL

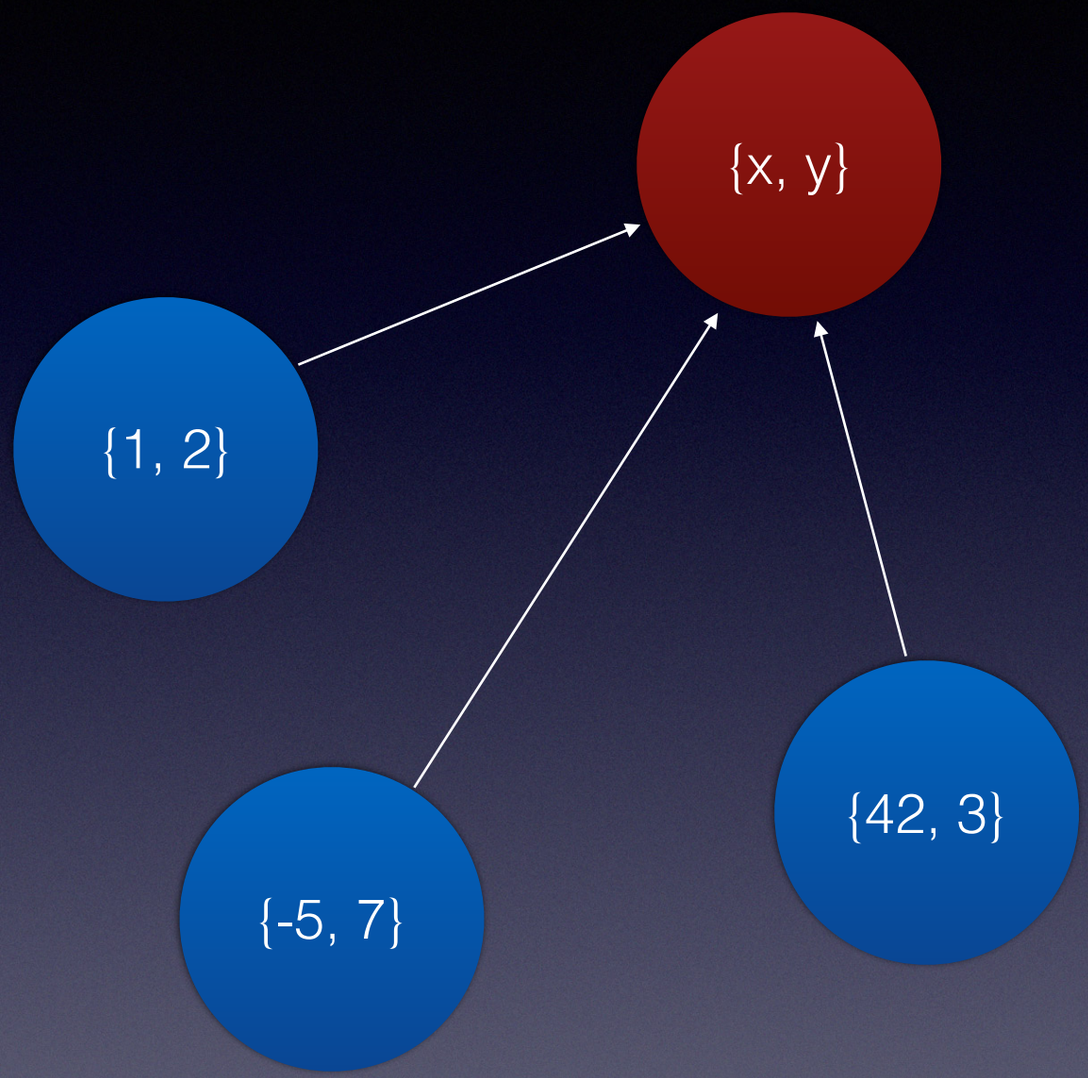

Speculation
Baseline
DFG
Baseline
FTL

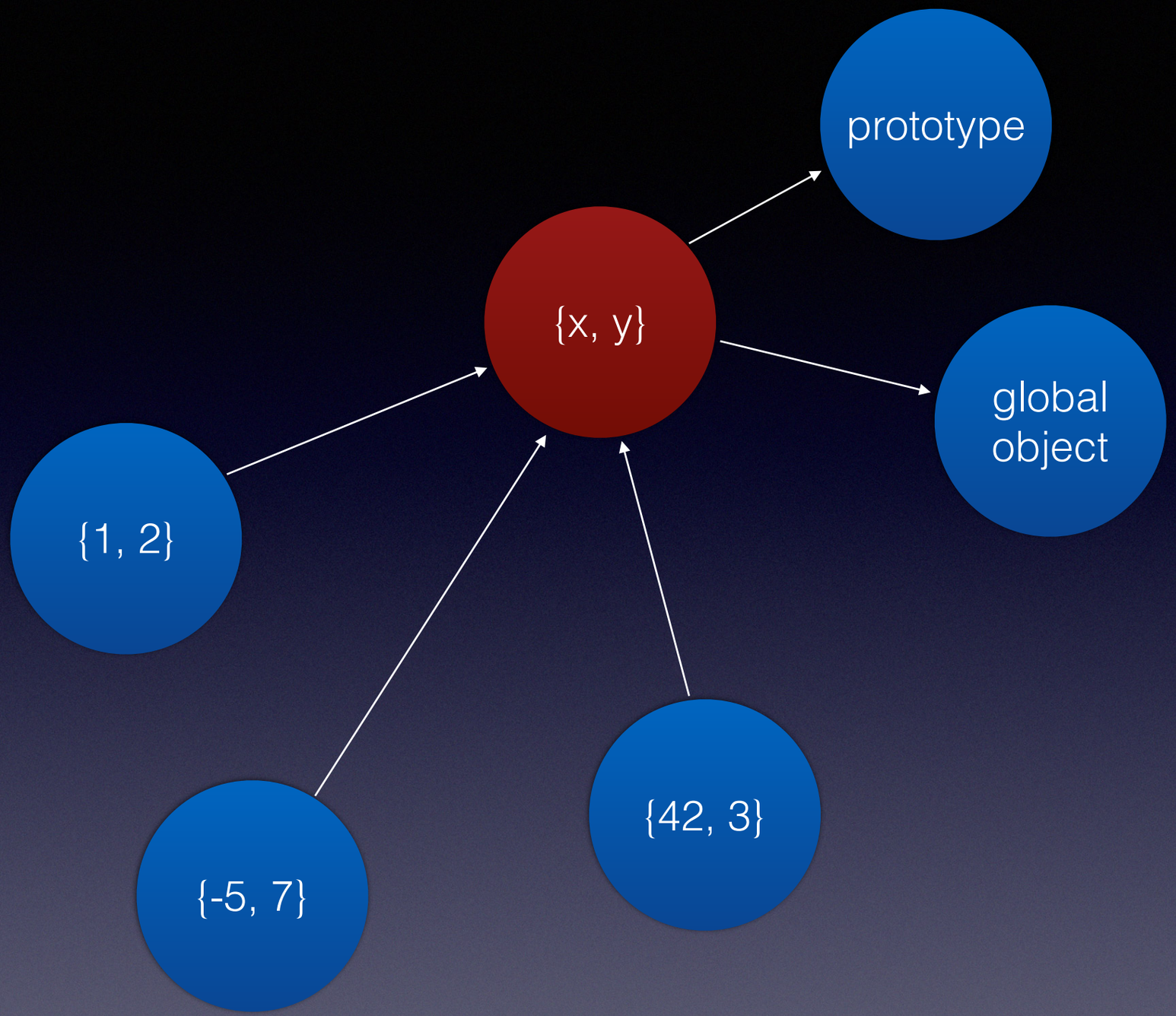
Structures

{x: 1, y: 2}

{x: -5, y: 7}

{x: 42, y: 3}


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


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

Structures Enable Optimizations

- Fast property access by specializing on structure
- Property type inference
- Immutable property inference
- Prototype optimizations

Keeping Structures Alive

- If live object O has structure S, then S is live.
- If S is used for JIT optimizations and its prototype and global object are live, then S is live.
- If an JIT optimization creates structure S based on structure R, and R is live, then S is live.

Watchpoints

```
class Foo {
    constructor()
    {
        this._bar = 42;
    }

    get bar()
    {
        return this._bar;
    }
}
```

```
var Foo = function() {
    this._bar = 42;
};

Foo.prototype.__defineGetter__(
    “bar”,
    function() {
        return this._bar;
    });
```

- Any instance of Foo could override bar
- Foo.prototype.bar could be removed or reconfigured, affecting all current and future instances of Foo
- Foo.prototype could be reassigned, affecting all future instances of Foo
- Foo.prototype.__proto__ could reassigned, affecting all current and future instances of Foo
- Any instance of Foo could remove or reconfigure _bar
- Foo.prototype._bar could be defined to be a setter
- Foo could be reassigned
- …

```
class Foo {
    constructor()
    {
        this._bar = 42;
    }

    get bar()
    {
        return this._bar;
    }
}
```

```
var o = new Foo();

print(o.bar);
```

```
class Foo {
    constructor()
    {
        this._bar = 42;
    }

    get bar()
    {
        return this._bar;
    }
}
```

```
var o = new Foo();

print(o.bar);
```

```
class Foo {
    constructor()
    {
        this._bar = 42;
    }

    get bar()
    {
        return this._bar;
    }
}
```

```
var o = new Foo();

Object.defineProperty(o, “bar”, …);

print(o.bar);
```

```
class Foo {
    constructor()
    {
        this._bar = 42;
    }

    get bar()
    {
        return this._bar;
    }
}
```

```
var o = new Foo();

print(o.bar);
```

```
class Foo {
    constructor()
    {
        this._bar = 42;
    }

    get bar()
    {
        return this._bar;
    }
}
```

```
var o = new Foo();

delete Foo.prototype.bar;

print(o.bar);
```

```
class Foo {
    constructor()
    {
        this._bar = 42;
    }

    get bar()
    {
        return this._bar;
    }
}
```

```
var o = new Foo();

print(o.bar);
```

```
class Foo {
    constructor()
    {
        this._bar = 42;
    }

    get bar()
    {
        return this._bar;
    }
}


Foo.prototype = {};
var o = new Foo();


print(o.bar);
```

```
class Foo {
    constructor()
    {
        this._bar = 42;
    }

    get bar()
    {
        return this._bar;
    }
}
```

```
var o = new Foo();

print(o.bar);
```

```
class Foo {
    constructor()
    {
        this._bar = 42;
    }

    get bar()
    {
        return this._bar;
    }
}
```

```
Foo.prototype.__proto__ = {set _bar(value) { print(“hi!”); } };
var o = new Foo();

print(o.bar);
```

```
class Foo {
    constructor()
    {
        this._bar = 42;
    }

    get bar()
    {
        return this._bar;
    }
}
```

```
var o = new Foo();

print(o.bar);
```

```
class Foo {
    constructor()
    {
        this._bar = 42;
    }

    get bar()
    {
        return this._bar;
    }
}

var o = new Foo();

delete o._bar;

print(o.bar);
```

```
class Foo {
    constructor()
    {
        this._bar = 42;
    }

    get bar()
    {
        return this._bar;
    }
}
```

```
var o = new Foo();

print(o.bar);
```

```
class Foo {
    constructor()
    {
        this._bar = 42;
    }

    get bar()
    {
        return this._bar;
    }
}

Foo.prototype.__defineSetter__(“_bar”, value => print(“hi!”));
var o = new Foo();

print(o.bar);
```

```
class Foo {
    constructor()
    {
        this._bar = 42;
    }

    get bar()
    {
        return this._bar;
    }
}
```

```
var o = new Foo();

print(o.bar);
```

```
class Foo {
    constructor()
    {
        this._bar = 42;
    }

    get bar()
    {
        return this._bar;
    }
}

Foo = 42;
var o = new Foo();

print(o.bar);
```

- Any instance of Foo could override bar
- Foo.prototype.bar could be removed or reconfigured, affecting all current and future instances of Foo
- Foo.prototype could be reassigned, affecting all future instances of Foo
- Foo.prototype.__proto__ could reassigned, affecting all current and future instances of Foo
- Any instance of Foo could remove or reconfigure _bar
- Foo.prototype._bar could be defined to be a setter
- Foo could be reassigned
- …

OSR + Watchpoints

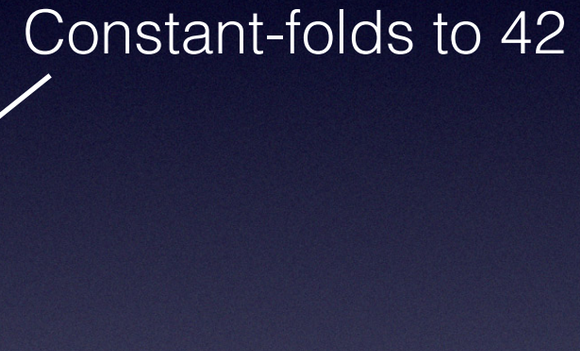
Things you can watch

- The value of a property
- The presence and configuration of a property
- The absence of a property
- The state of __proto__
- …

Example

```
"use strict";

class Foo {
    constructor()
    {
        this._bar = 42;
    }

    get bar()
    {
        return this._bar;
    }
}

function baz()
{
    return new Foo().bar;
}

noInline(baz);

for (var i = 0; i < 10000; ++i)
    print(baz());
```

```
"use strict";

class Foo {
    constructor()
    {
        this._bar = 42;
    }

    get bar()
    {
        return this._bar;
    }
}

function baz()
{
    return new Foo().bar;
}

noInline(baz);

for (var i = 0; i < 10000; ++i)
    print(baz());
```



```
"use strict";

class Foo {
    constructor()
    {
        this._bar = 42;
    }

    get bar()
    {
        return this._bar;
    }
}

function baz()
{
    return new Foo().bar;
}

noInline(baz);

for (var i = 0; i < 10000; ++i)
    print(baz());

delete Foo.prototype.bar;
Foo.prototype.bar = 666;

print(baz());
```

```
"use strict";

class Foo {
    constructor()
    {
        this._bar = 42;
    }

    get bar()
    {
        return this._bar;
    }
}

function baz()
{
    return new Foo().bar;
}

noInline(baz);

for (var i = 0; i < 10000; ++i)
    print(baz());

delete Foo.prototype.bar;
Foo.prototype.bar = 666;

print(baz());
```

Constant-folds to 42

```
"use strict";

class Foo {
    constructor()
    {
        this._bar = 42;
    }

    get bar()
    {
        return this._bar;
    }
}

function baz()
{
    return new Foo().bar;
}

noInline(baz);

for (var i = 0; i < 10000; ++i)
    print(baz());

delete Foo.prototype.bar;
Foo.prototype.bar = 666;

print(baz());
```

Constant-folds to 42

Bunch of watchpoints fire

```
"use strict";

class Foo {
    constructor()
    {
        this._bar = 42;
    }

    get bar()
    {
        return this._bar;
    }
}

function baz()
{
    return new Foo().bar;
}

noInline(baz);

for (var i = 0; i < 10000; ++i)
    print(baz());

delete Foo.prototype.bar;
Foo.prototype.bar = 666;

print(baz());
```

Constant-folds to 42

FTL-compiled baz() no longer exists; this runs in Baseline

Bunch of watchpoints fire

```
"use strict";

class Foo {
    constructor()
    {
        this._bar = 42;
    }

    get bar()
    {
        return this._bar;
    }
}

let o = new Foo();
let p = {bar: 42};

function baz(o)
{
    return o.bar;
}

noInline(baz);

for (var i = 0; i < 10000; ++i) {
    baz(o);
    baz(p);
}
```

```
"use strict";

class Foo {
    constructor()
    {
        this._bar = 42;
    }

    get bar()
    {
        return this._bar;
    }
}
```

Compiled by PolymorphicAccess JIT

```
let o = new Foo();
let p = {bar: 42};

function baz(o)
{
    return o.bar;
}

noInline(baz);

for (var i = 0; i < 10000; ++i) {
    baz(o);
    baz(p);
}
```

Embedding

- We go *all in* for embedding.

Embedding

- WebCore
- C API
- Objective-C API

Embedding

- WebCore
- C API *experimenting with marking constraints API*
- Objective-C API

JavaScriptCore

- Supports ES2015+ and WebAssembly
- One interpreter and seven JITs
- OSR and Watchpoints
- Structures
- Designed for Embedding
- *Concurrent, Parallel, and Generational GC*

Agenda

- ~~Introduction~~
- ~~JavaScriptCore~~
- Efficient Mark-Sweep
  - *30 minute break*
- Concurrent GC
- bmalloc
- WTF::Lock