Implementing one feature set in two JavaScript engines

Caio Lima & Joyee Cheung
Igalia & Bloomberg

Overview of the class features

ES6

- Public class methods (instance & static)

3 follow-up proposals

- Class instance fields: https://tc39.es/proposal-class-fields/
  - Public fields
  - Private fields
- Private methods & accessors: https://tc39.es/proposal-private-methods/
- Static class features: https://tc39.es/proposal-static-class-features/
  - Static public fields
  - Static private fields
  - Static private methods & accessors

Overview of the class features

- The class features entered Stage 3 in July 2017
- Stage 3 is when
  - TC39 settles down the design of language features
  - JavaScript engines start implementing language features, giving feedback to TC39, and shipping the implementation
  - https://tc39.es/process-document/
- Thanks Bloomberg for sponsoring Igalia's work!
  - Implementing the 3 proposals in JavaScriptCore
  - Implementing private methods (instance and static) as well as improving other class features in V8

Public fields

```
let i = 0;
function count() {
  return i++;
}

class C {
  field = count();
}

(new C).field; // returns 0
(new C).field; //returns 1
```

- Instance fields are defined during object construction.
- The initializer is executed every time a new object is instantiated.

Private fields

```
class C {
  #field = 1;
  access() {
    return this.#field;
  }
}

(new C).access(); // returns 1
(new C).access.call({}); // TypeError
```

- Private fields are not common JS properties.
  - When a private field is not present, we throw an TypeError.
  - They don’t have a property descriptor.
- They are only visible inside the scope of a class.

Private fields

```
class C {
  #field = 1;
  access() {
    return this.#field;
  }
}

(new C).access(); // returns 1
(new C).access.call({}); // TypeError

class D {
  #field = 1;
  // ...
}
```

- Private fields are not common JS properties.
  - When a private field is not present, we throw an TypeError.
  - They don’t have a property descriptor.
- They are only visible inside the scope of a class.
- Every new evaluation of a class creates a new private name.

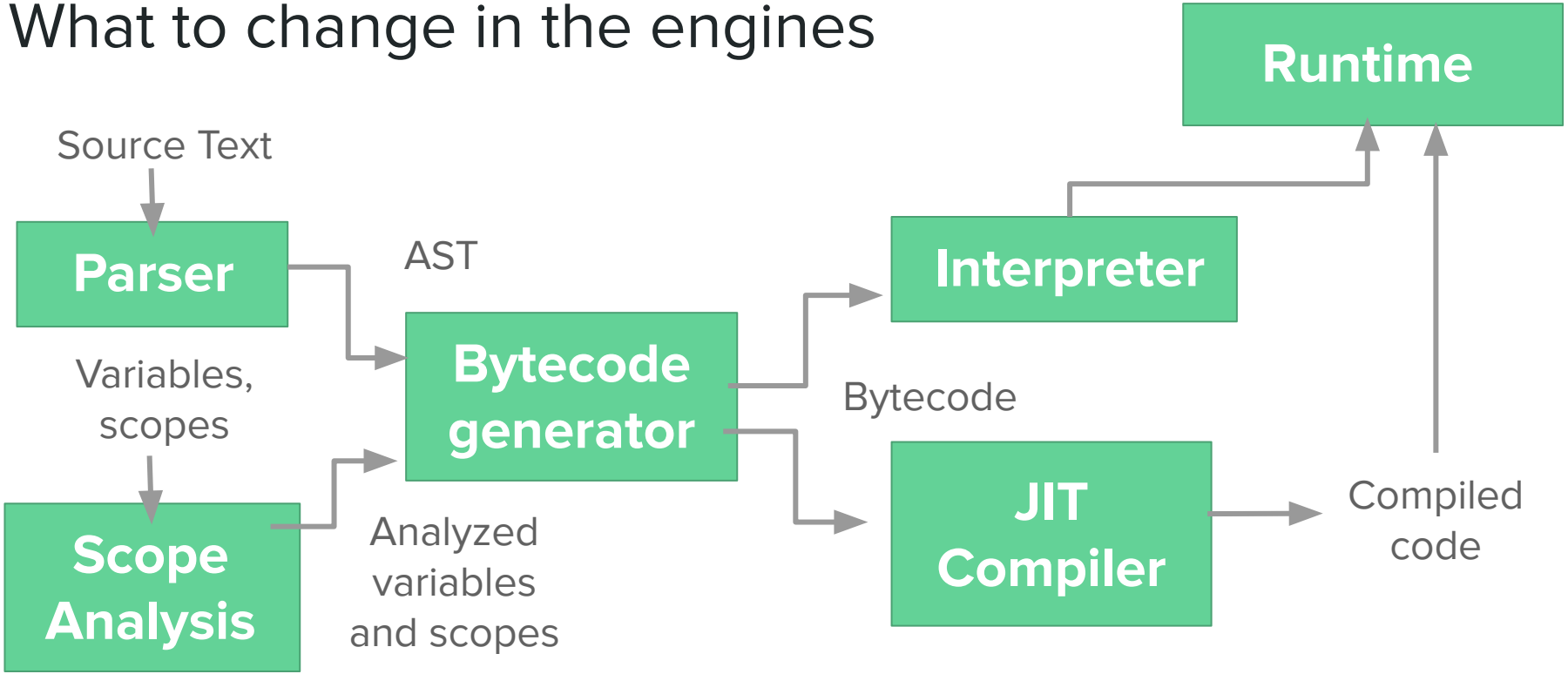
Private methods & static fields

```
class C {
  #method() {
    return “I am instance”
  };
  static #staticMethod() {
    return “I am Static”;
  }
}
```

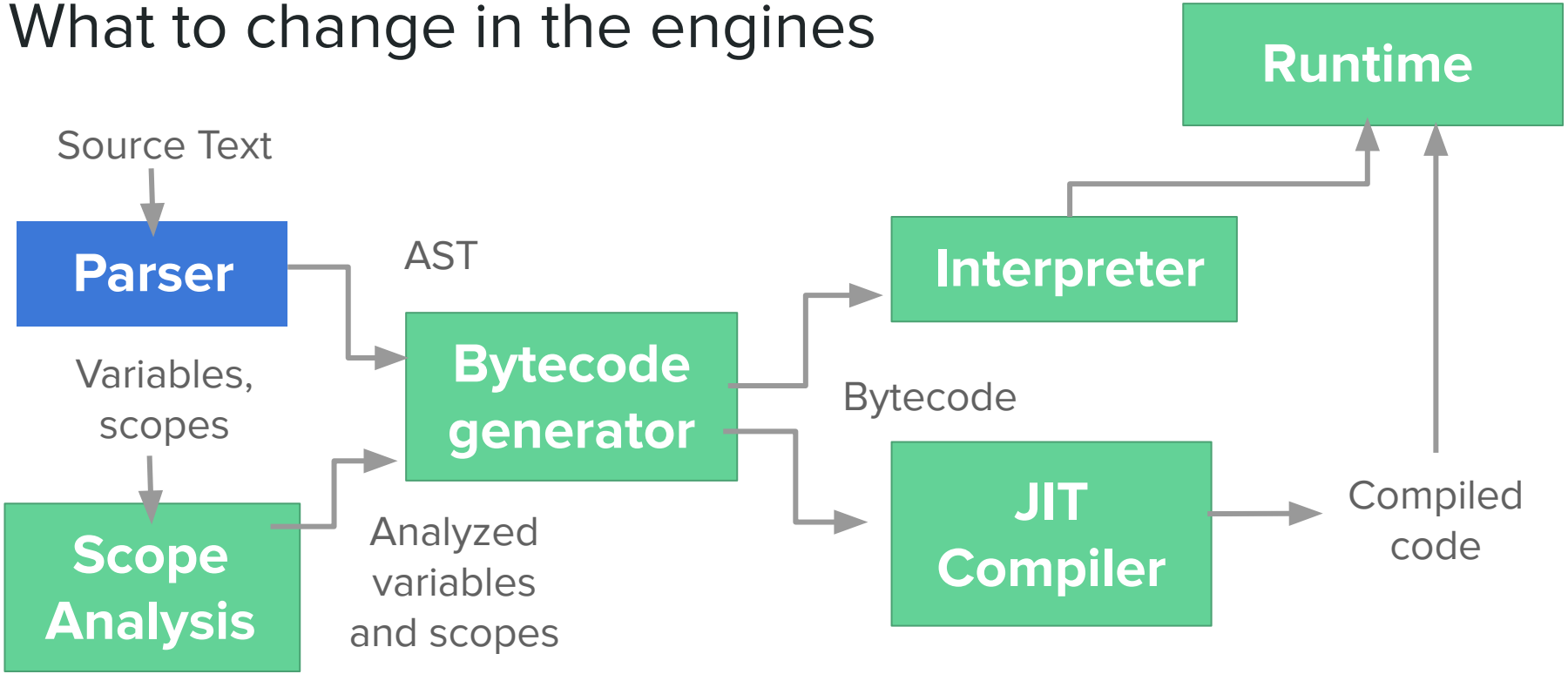
```
class C {
  static field = 1;
  static #field;
}

C.field; // returns 1
```

What to change in the engines



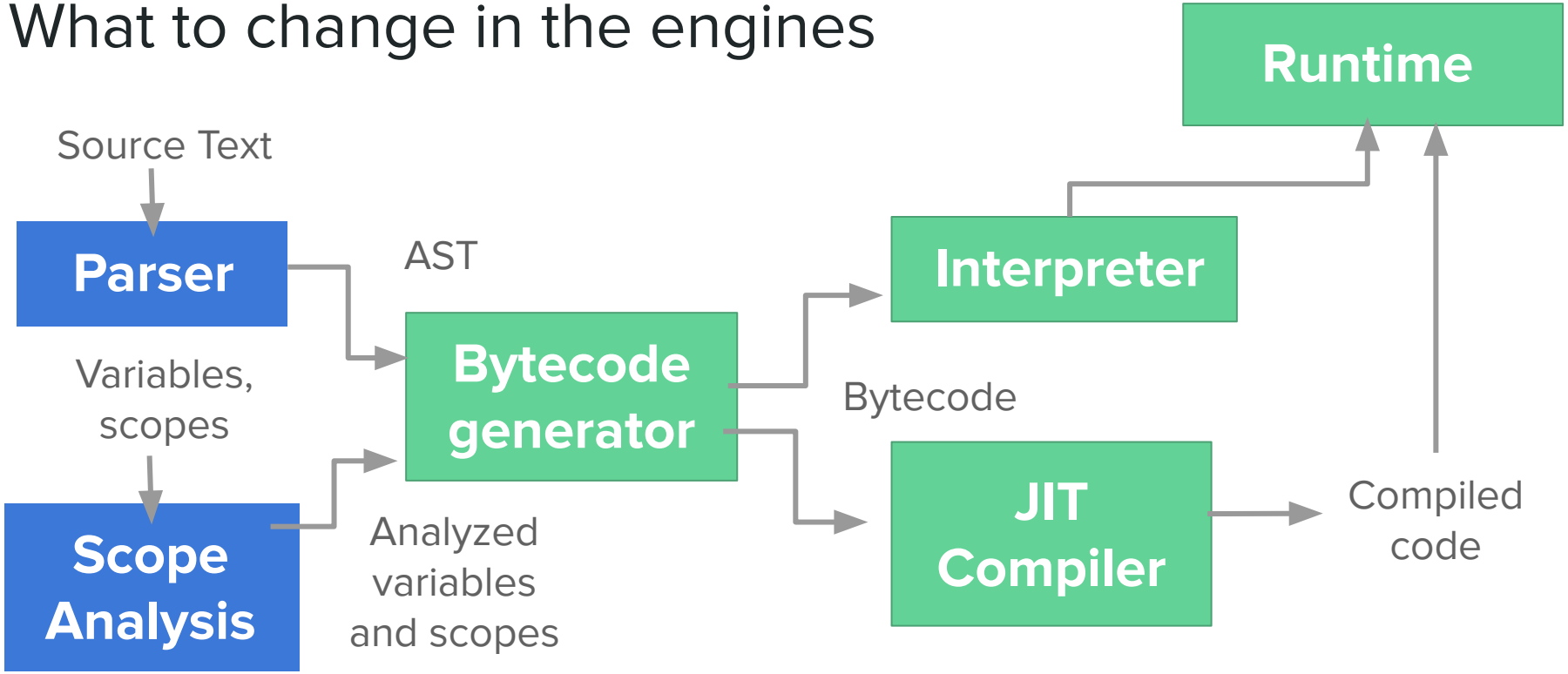
What to change in the engines



What to change in the engines: parser

- Support new production “#identifier”
- Easy: both JSC and V8 use recursive descent parsers
- Add new AST nodes for the bytecode generator to visit later

What to change in the engines



What to change in the engines: scope analysis

- Specialize the resolution of private names to identify usage of undeclared fields whenever we finish parsing a class literal
- Add additional fields to the variables to carry information about the kind of property access
- In V8: rewrote the scope analysis of class scopes

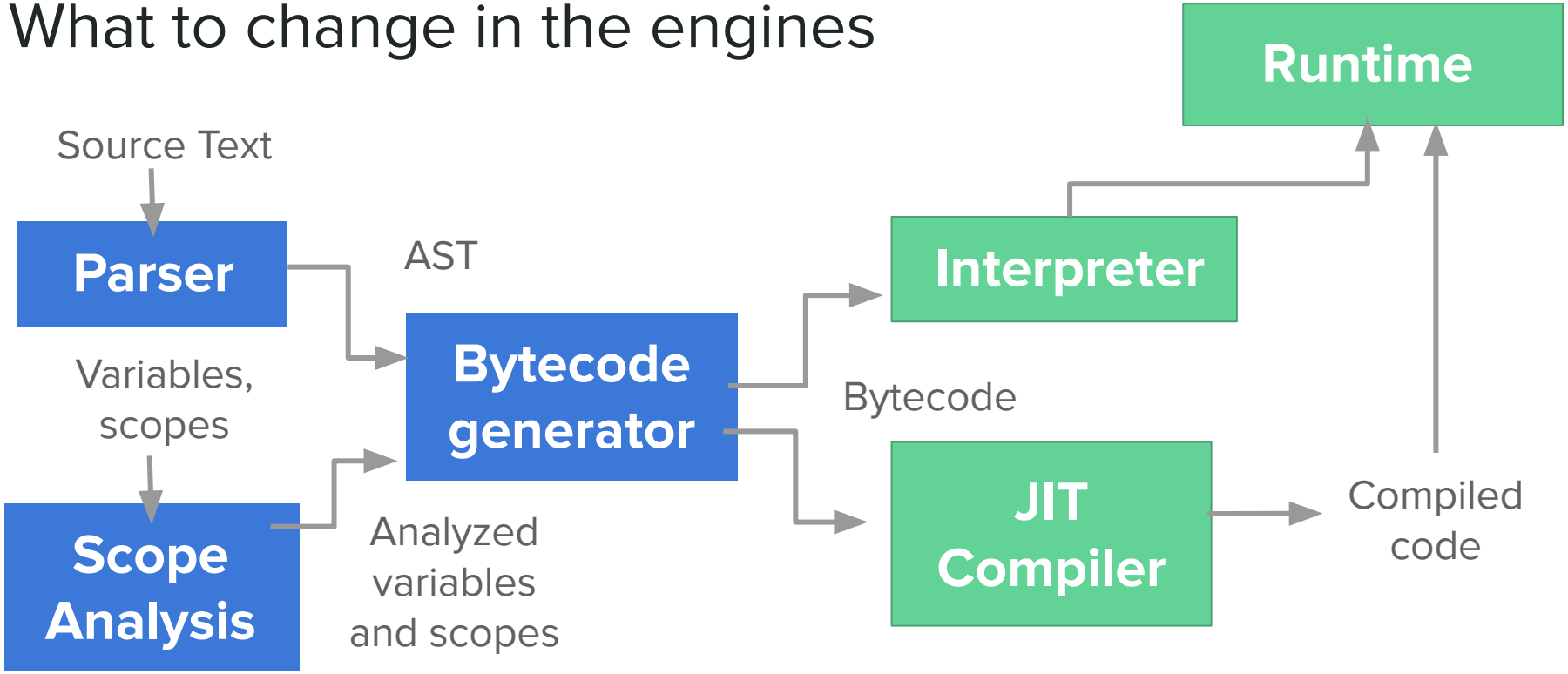
```
class C {
  #field = 1;
  method() { this.#filed = 2; }  // typo: SyntaxError
}
class C {
  #duplicateField = 1;
  #duplicateField = 2;  // SyntaxError
}
```

What to change in the engines: scope analysis

- With lazy parsing, errors are identified and the variables are serialized in the pre-parsing.
- Deserialize variables when generating the bytecode

```
class C {
  #field = 1;  // Serialized
  getField() { this.#field; /* Deserialized */ }
}
(new C).getField();  // Triggers bytecode generation of getField
```

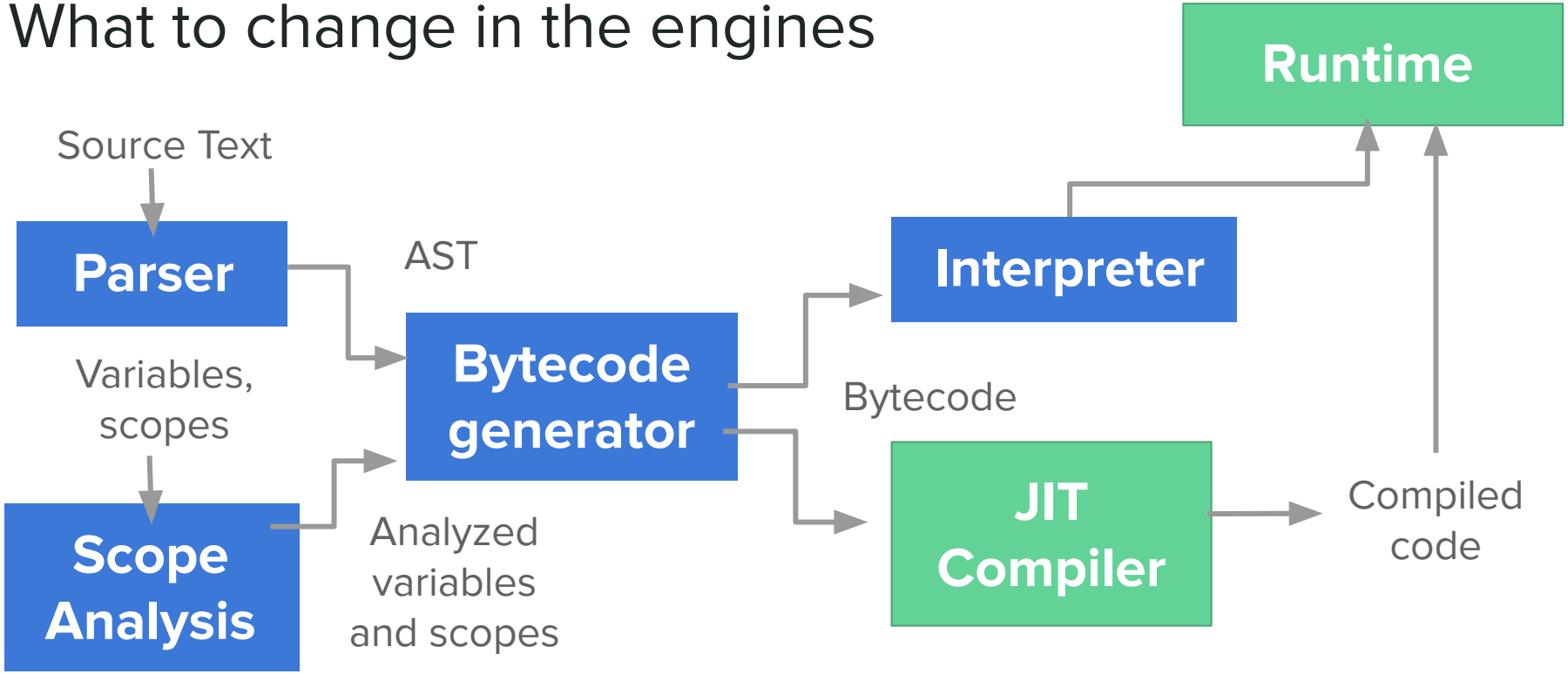
What to change in the engines



What to change in the engines: bytecode generator

- Generate bytecode for these new features
- Change the bytecode emitted for
  - Class evaluation
  - Class constructors
  - Property access

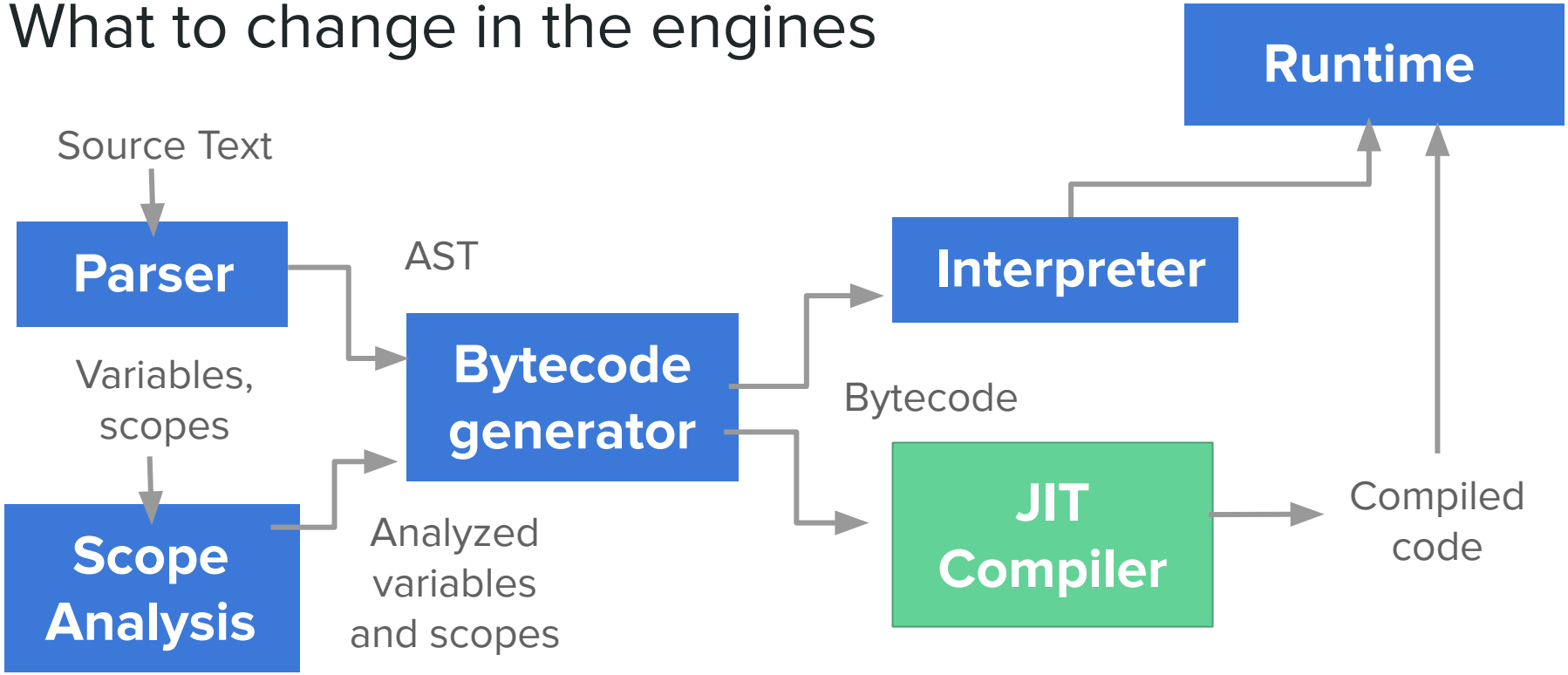
What to change in the engines



What to change in the engines: interpreter

- Add new handlers for new operations added for the features, if any.
- JSC added a new `get_by_val_direct` instruction.

What to change in the engines



What to change in the engines: runtime

- Runtime: property lookup is desugared to static lexical lookups
  - Special path to lookup private symbols (different semantics)
  - Methods and accessors need to validate if receiver has the correct brand
  - Static: validate the receivers and change where things are installed

Bytecode generated for instance private fields

During class evaluation

```
class C {
  #instanceField = 1;
}
```

V8

```
CallRuntime [CreatePrivateNameSymbol]  // #instanceField
StaCurrentContextSlot [4]  // known index to a fixed array
...
CreateClosure  // instance_members_initializer
// Store instance_members_initializer in the class
StaNamedProperty  <class_constructor>  ...
```

Bytecode generated for instance private fields

During class evaluation

JSC

```
create_lexical_environment loc4, ...
...
call                loc12, “@createPrivateSymbol”
put_to_scope        loc4, “#instanceField”, loc12
new_fuc_exp         loc13, ...
put_by_id           <C>, “@instanceFieldInitializer”, loc13
...
```

```
class C {
  #instanceField = 1;
}
```

Bytecode generated for instance private fields

In the constructor

```
class C {
  #instanceField = 1;
}
```

V8

```
// In the class C constructor
LdaNamedProperty  // load instance_members_initializer
CallProperty0 // run instance_members_initializer

// In instance_members_initializer
LdaCurrentContextSlot [4] // Load the #instanceField symbol from the context
Star r1
LdaSmi [1]
Star r2
Mov <this>, r0
CallRuntime [AddPrivateField], r0-r2  // Define this.#instanceField as 1
```

Bytecode generated for instance private fields

In the constructor

```
class C {
  #instanceField = 1;
}
```

JSC

```
// In the C constructor
get_by_id_direct   loc7, callee, “@instanceFieldInitializer”
mov                loc8, this
call               loc9, loc7, 1
ret                this

//In the “@instanceFieldInitializer”
mov                loc6, Int32: 1
resolve_scope      loc7, loc4, “#inscanteField”
get_from_scope     loc8, loc7, “#inscanteField”
put_by_val_direct  this, loc8, loc6, PrivateName|ThrowIfExists
ret                Undefined(const1)
```

Bytecode generated for instance private fields

When evaluating `getInstanceField()`

V8

```
class C {
  #instanceField = 1;
  getInstanceField() { return this.#instanceField; }
}
```

```
LdaCurrentContextSlot [4]  // Load the private symbol
LdaKeyedProperty <this>, [0]  // Error in the IC if the field does not exist
```

Bytecode generated for instance private fields

When evaluating `getInstanceField()`

```
class C {
  #instanceField = 1;
  getInstanceField() { return this.#instanceField; }
}
```

JSC

```
resolve_scope      loc7, loc4, “#instanceField”
get_from_scope     loc8, loc7, “#instanceField” // get PrivateSymbol
get_by_val_direct  loc6, this, loc8
ret                loc6
```

Other class features

- Private methods are shared among the instances, the validation of the receiver is guarded by a per-class special symbol property (the “brand”).
- Static features are implemented similarly to instance features, but handled during class evaluation time.

Implementation status

- Class Fields
  - Chrome: Shipped full implementation in 74 (23 April 2019).
  - WebKit: In progress (link).
- Private Methods & accessors
  - Chrome: Fully implemented behind --harmony-private-methods on master (link).
  - WebKit: In progress (methods and accessors).
- Static features
  - Chrome: Static class fields shipped in 74, static private methods are fully implemented behind --harmony-private-methods on master (link).
  - WebKit: In progress (link).

Implementation status

Spec issues discovered during implementation

- https://github.com/tc39/proposal-class-fields/issues/263
- https://github.com/tc39/proposal-private-methods/issues/69

Test262 status

- Already complete (last PR from 30 August 2019).
- Total of 6325 new tests.

Questions?

Desugaring public fields (incorrect, just conceptual)

```
class C {
  field = 1;
}

class C {
  constructor() {
    Object.defineProperty(this, ‘field’, {value: 1});
  }
}
```

Desugaring private fields (incorrect, just conceptual)

```
class C {
  #field = 1;
  getField() { return this.#field; }
}

class C {
  // Imagine it's possible to declare a fieldSymbol here.
  constructor() {
    Object.defineProperty(this, fieldSymbol, {value: 1});
  }
  getField() { return this[fieldSymbol];  }
}
```

Desugaring private methods (incorrect, just conceptual)

```
class C {
  #method() { }
  runMethod() { this.#method(); }
}

class C {
  // Imagine it's possible to declare a brandSymbol and a <method>() here.
  constructor() { Object.defineProperty(this, brandSymbol, {value: /*?*/}); }
  runMethod() {
    if (!(brandSymbol in this)) { throw TypeError('...'); }
    <method>.call(this);
  }
}
```

Desugaring static methods and fields (incorrect)

```
class C {
  static #method() { }
  static #field = 1;
  static runMethod() { this.#method(); }
}

// Imagine it's possible to declare a brandSymbol and a <method>() here.
Object.defineProperty(C, brandSymbol, {value: /*?*/});
Object.defineProperty(C, fieldSymbol, {value: 1});
C.runMethod = function() {
  if (!(brandSymbol in this)) { throw TypeError('...'); }
  <method>.call(this);
}
```

Bytecode generated for instance private methods

During class evaluation

```
class C {
  #instanceMethod() {}
}
```

V8

```
CallRuntime [CreatePrivateNameSymbol]  // brand symbol
StaCurrentContextSlot [5]
......
...
// Create the private method
CreateClosure
StaCurrentContextSlot [4]
```

Bytecode generated for instance private methods

During class evaluation

JSC

```
create_lexical_environment loc9, loc4
...
call                loc13, “@createPrivateSymbol”
put_to_scope        loc4, “@privateBrand”, loc13
new_func_exp        loc12, loc4, ...
put_by_id           loc12, “@homeObject”, loc11
put_to_scope        loc4, “#inscanceMethod”, loc12
```

```
class C {
  #instanceMethod() {}
}
```

Bytecode generated for instance private methods

In the constructor

V8

```
LdaCurrentContextSlot [5]  // brand symbol
Star r1
Mov <this>, r0
CallRuntime [AddPrivateBrand], r0-r1
```

```
class C {
  #instanceMethod() {}
}
```

Bytecode generated for instance private methods

In the constructor

JSC

```
resolve_scope      loc7, loc4, “@privateBrand”
get_from_scope     loc8, loc7, “@privateBrand”
put_by_val_direct  this, loc8, loc8, PrivateName|ThrowIfExists
ret                this
```

```
class C {
  #instanceMethod() {}
}
```

Bytecode generated for instance private methods

When evaluating `runInstanceMethod()`

```
class C {
  #instanceMethod() {};
  runInstanceMethod() { this.#instanceMethod(); }
}
```

V8

```
LdaCurrentContextSlot [5]  // Load the brand symbol
LdaKeyedProperty <this>, [0]  // brand check - errors if it does not exist
LdaCurrentContextSlot [4]  // Load the method
Star r0
CallAnyReceiver r0, <this>-<this>, [2]
```

Bytecode generated for instance private methods

When evaluating `runInstanceMethod()`

```
class C {
  #instanceMethod() {};
  runInstanceMethod() { this.#instanceMethod(); }
}
```

JSC

```
mov                 loc8, this
resolve_scope       loc9, loc4, “#instanceMethod”
get_from_scope      loc10, loc9, “@privateBrand”
get_by_val_direct   loc10, loc8, loc10 // Brand Check
get_from_scope      loc6, loc9, “#instanceMethod”
call                loc6, loc6, 1
...
```

Brand checking saves memory

```
class C {
  constructor() { Object.defineProperty(this, brandSymbol, {value: /*?*/}); }

  runMethod() {
    if (!(brandSymbol in this)) { throw TypeError('...'); }
    <methodA>.call(this);
    <methodB>.call(this);
  }
}
```

Brand checking saves memory

```
class C {
  constructor() {
    Object.defineProperty(this, methodASymbol, {value: <methodA>, ...});
    Object.defineProperty(this, methodBSymbol, {value: <methodB>, ...});
    // More symbols and references in proportion to the number of private methods
  }

  runMethod() {
    this[methodASymbol]();
    this[methodBSymbol]();
  }
}
```

Bytecode generated for static private methods

During class evaluation

V8

```
// During class evaluation
CreateClosure
StaCurrentContextSlot [4]
```

```
class C {
  static #staticMethod() {}
  static runStaticMethod() { this.#staticMethod(); }
}
```

Bytecode generated for static private methods

When evaluating runStaticMethod()

```
class C {
  static #staticMethod() {}
  static runStaticMethod() { this.#staticMethod(); }
}
```

V8

```
LdaCurrentContextSlot [5]  // Load the class that declares the static method
TestReferenceEqual <this>  // Make sure the receiver is the class
Mov <this>, r1
JumpIfTrue
LdaSmi.Wide
Star r2
LdaConstant [0]
Star r3
CallRuntime [NewTypeError], r2-r3
Throw
LdaCurrentContextSlot [4]  // Where JumpIfTrue jumps to: load the static method
Star r0
CallAnyReceiver r0, r1-r1, [0]
```

Bytecode generated for static private methods

JSC

```
class C {
  static #staticMethod() {}
  static runStaticMethod() { this.#staticMethod(); }
}
```

```
create_lexical_environment loc9, loc4
call                loc13, “@createPrivateSymbol”
put_to_scope        loc4, “@privateStaticBrand”, loc13
new_func_exp        loc12, loc4, 1
put_by_id           loc12, “@homeObject”, loc11,
put_to_scope        loc4, “#staticMethod”, loc12
...
resolve_scope       loc7, loc4, “@privateStaticBrand”
get_from_scope      loc8, loc7, “@privateStaticBrand”
put_by_val_direct   <C>, loc8, loc8, PrivateName|ThrowIfExists
```

Bytecode generated for private accessors

- Similar to private methods, guarded by brand checks
- Complementary accessors are stored in AccessorPairs in V8, but separately in JSC
- Generate TypeError statically for incorrect usage to read-only or write-only private accessors

```
class C {
  get #value() { }
  set #value(val) { }
  inc() { return this.#value++; }
}
```

Bytecode generated for static public fields

- Defined in static field initializers in V8 and accessed as usual
- Inlined in the class evaluation in JSC

```
class C {
  static publicField = 1;
  static publicMethod() { return this.publicField; }
}

C.publicMethod();
```

Bytecode generated for static private fields

V8

```
class C {
  static #staticField = 1;
  static getStaticField() { return this.#staticField; }
}
```

```
// During class evaluation
// Create the #staticField symbol
CallRuntime [CreatePrivateNameSymbol]
StaCurrentContextSlot [4]
...
CreateClosure  // static_fields_initializer
CallProperty0  // calls static_fields_initializer on the class

// In the static_fields_initializer
LdaCurrentContextSlot [4] // Load the #staticField symbol
Star r1
LdaSmi [1]
Star r2
Mov <this>, r0
CallRuntime [AddPrivateField], r0-r2  // Define C.#staticField as 1
```

Bytecode generated for static private fields

JSC

```
class C {
  static #staticField = 1;
  static getStaticField() { return
this.#staticField; }
}
```

```
create_lexical_environment loc4, ...
...
call                loc12, “@createPrivateSymbol”
put_to_scope        loc4, “#instanceField”, loc12
...
mov                 loc6, Int32: 1
resolve_scope       loc7, loc4, “#inscanteField”
get_from_scope      loc8, loc7, “#inscanteField”
put_by_val_direct   <C>, loc8, loc6, ...
```