goto; conference aarhus

Translating Dart to efficient JavaScript

Kasper Lund

Google

Translating Dart to efficient JavaScript

Kasper Lund, Google

Who am I?

Kasper Lund, software engineer at Google

Projects

- **OOVM:** Embedded Smalltalk system
- **V8:** High-performance JavaScript engine
- **Dart:** Structured programming for the web

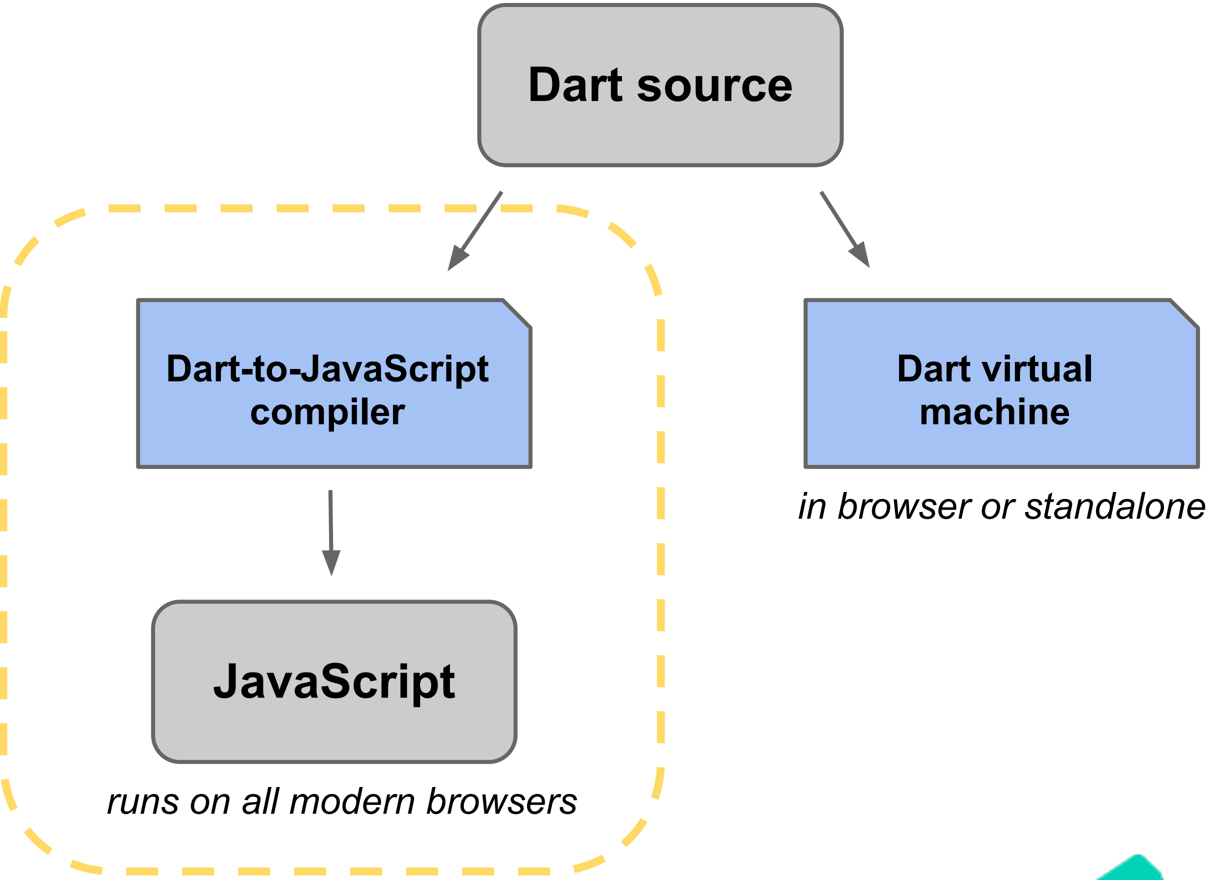
DART

What is Dart?

- Unsurprising object-oriented programming language
- Class-based single inheritance
- Familiar syntax with proper lexical scoping
- Optional static type annotations

```
main() {
  for (int i = 99; i > 0; i--) {
    print("$i bottles of beer on the wall, ....");
    print("Take one down and pass it around ...");
  }
}
```

Dart execution and deployment



Dart source

Dart-to-JavaScript compiler

JavaScript

runs on all modern browsers

Dart virtual machine

in browser or standalone

Dart-to-JavaScript compiler goals

- Support Dart apps on all modern browsers
  - Tested on Chrome, Firefox, IE, and Safari
  - Ensures that the use of the Dart VM is optional

- Generate efficient and compact JavaScript

- Implement proper Dart semantics
  - Check that the right number of arguments is passed
  - No implicit coercions to numbers or strings
  - Range checks for list access

Example: What's the point?

Source code in Dart

```
main() {
  var p = new Point(2, 3);
  var q = new Point(3, 4);
  var distance = p.distanceTo(q);
  ...
}
```

Example: What's the point?

Compiled JavaScript code

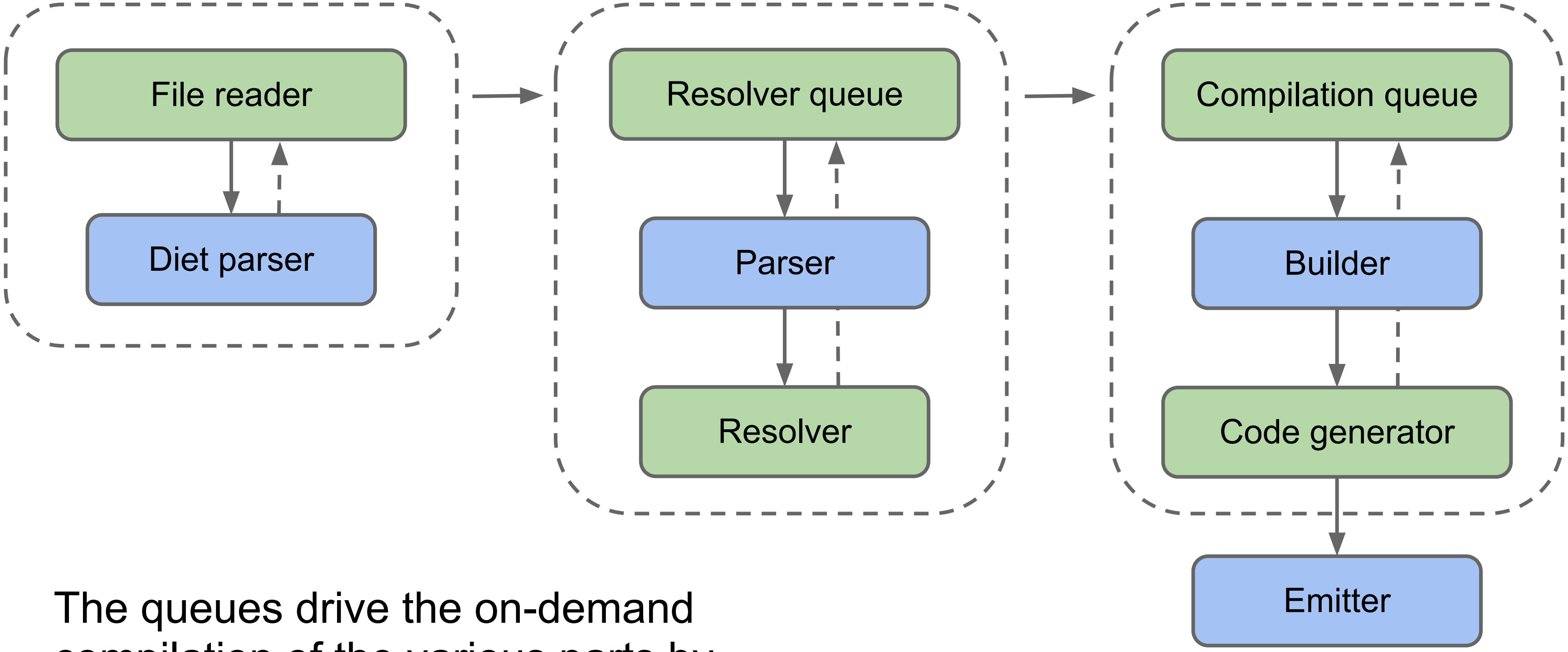
```
$.main = function() {
  var p = $.Point(2, 3);
  var q = $.Point(3, 4);
  var distance = p.distanceTo$1(q);
  ...
};
```

Example: What's the point?

- Static functions are put on the `$` object
  - Top-level functions such as `$.main`
  - Factory functions such as `$.Point`

- Method calls are translated to functions calls
  - Arity is encoded in the selector (`distanceTo$1`)
  - Supports named optional arguments

Tree shaking



File reader

Diet parser

Resolver queue

Parser

Resolver

Compilation queue

Builder

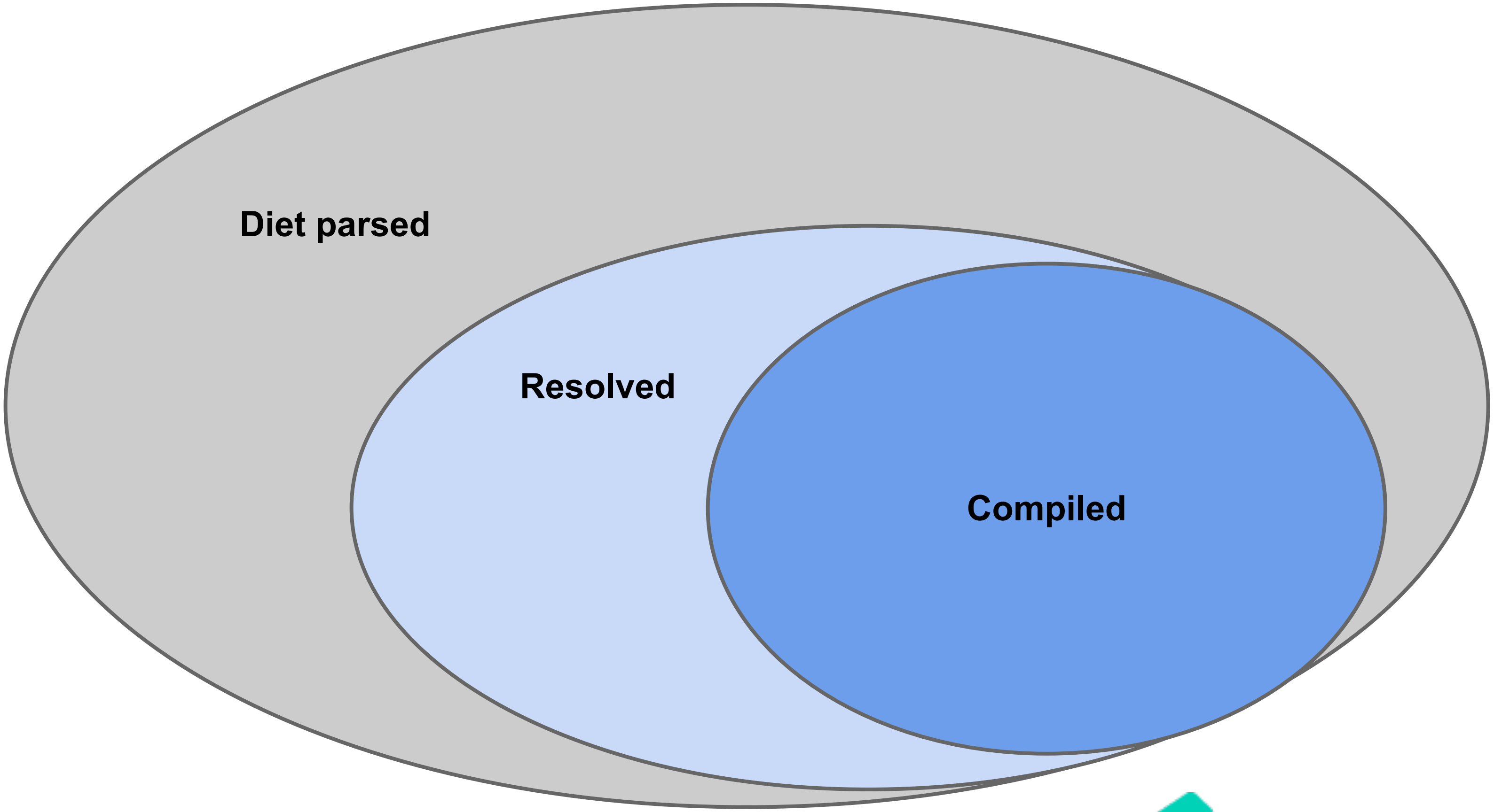
Code generator

Emitter

The queues drive the on-demand compilation of the various parts by keeping track of information about:

- Instantiated classes
- Used selectors (method names)
- Type information for receivers

Code after tree shaking



Language challenges

DART

User-definable operators

- JavaScript implicitly converts + inputs to numbers or strings

- Using method calls for all arithmetic operations is too slow

- **Solution:** Track types and use JavaScript + when it is safe to do so

```
Number.prototype.add = function(x) { return this + x; };
Number.prototype.sub = function(x) { return this - x; };
```

Range checking

- JavaScript has no notion of out of bounds access and all keys are treated as strings

- **Solution:** Insert explicit index checks unless we can prove we do not need them

Keep on truckin'

JavaScript

Example: Sum the elements of a list

Source code in Dart

```
main() {
  var list = [ 2, 3, 5, 7 ];
  var sum = 0;
  for (var i = 0; i < list.length; i++) {
    sum += list[i];
  }
  print("sum = $sum");
}
```

Example: Sum the elements of a list

Compiled JavaScript code

```
$.main = function() {
  var list = [1, 2, 3, 4];
  for (var t1 = list.length, sum = 0, i = 0; i < t1; ++i) {
    // Check that the index is within range before
    // reading from the list.
    if (i < 0 || i >= t1) throw $.ioore(i);
    var t2 = list[i];
    // Check that the element read from the list is
    // a number so it is safe to use + on it.
    if (typeof t2 !== 'number') throw $.iae(t2);
    sum += t2;
  }
  $.print('sum = ' + $.S(sum));
};
```

Compact class definitions

- Lots of classes means lots of boilerplate for creating instances and accessing fields

- **Solution:** Use a helper for defining classes and use dynamic code generation to cut down on the boilerplate

Compact class definitions

Compiled JavaScript code

```
$.Point = {"": ["x", "y"],
  "super": "Object",
  distanceTo$1: function(other) {
    var dx = this.x - other.x;
    var dy = this.y - other.y;
    return $.sqrt(dx * dx + dy * dy);
  }
};
```

Compact class definitions

Compiled JavaScript code

Essentially, we turn the field list `["x","y"]` into the following code using `new Function(...)` at runtime:

```
function Point(x, y) {
  this.x = x;
  this.y = y;
}

Point.prototype.get$x = function() { return this.x; };
Point.prototype.get$y = function() { return this.y; };
```

We also support field lists like `["x=",...]` which automatically introduces a setter too.

Closures

- Closures support named arguments and we must check the number of arguments

- Allocating small JavaScript objects is fast!
  - New JavaScript closure ~ new object with six fields

- **Solution:** Treat closures as class instances
  - Use instance fields for captured (boxed) variables
  - Use methods for implementing calling conventions

DART

Example: Closures

Source code in Dart

```
main() {
  var list = [ 1, 2, 3 ];
  print(list.map((each) => list.indexOf(each)));
}
```

Example: Closures

Compiled JavaScript code

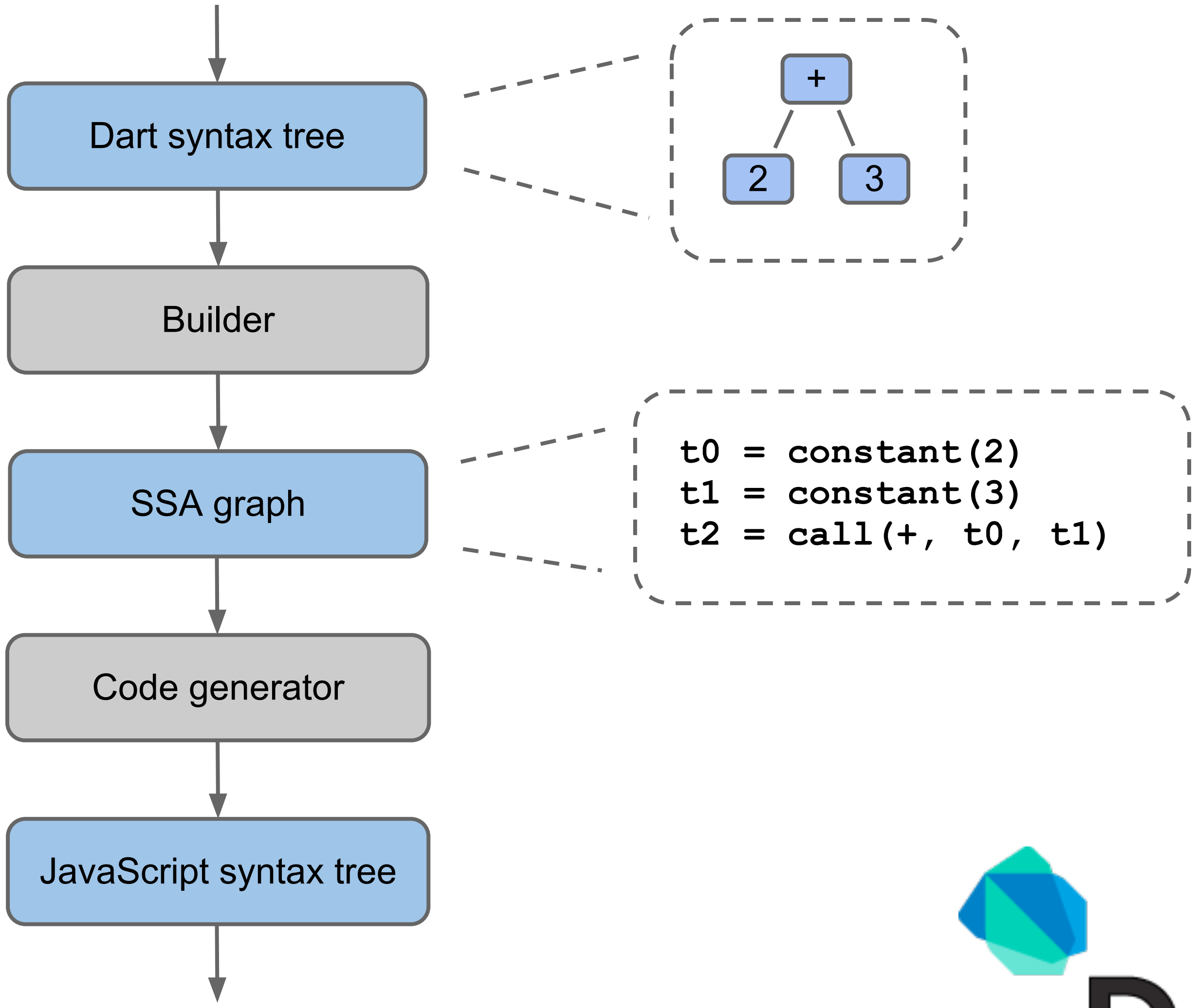
```
$.main = function() {
  var list = [1, 2, 3];
  $.print($.map(list, new $.main$closure(list)));
};

$.main$closure = {"": ["list"],
  call$1: function(each) {
    return $.indexOf$1(this.list, each);
  }
};
```

Generating code

DART

Intermediate representations



Dart syntax tree

Builder

SSA graph

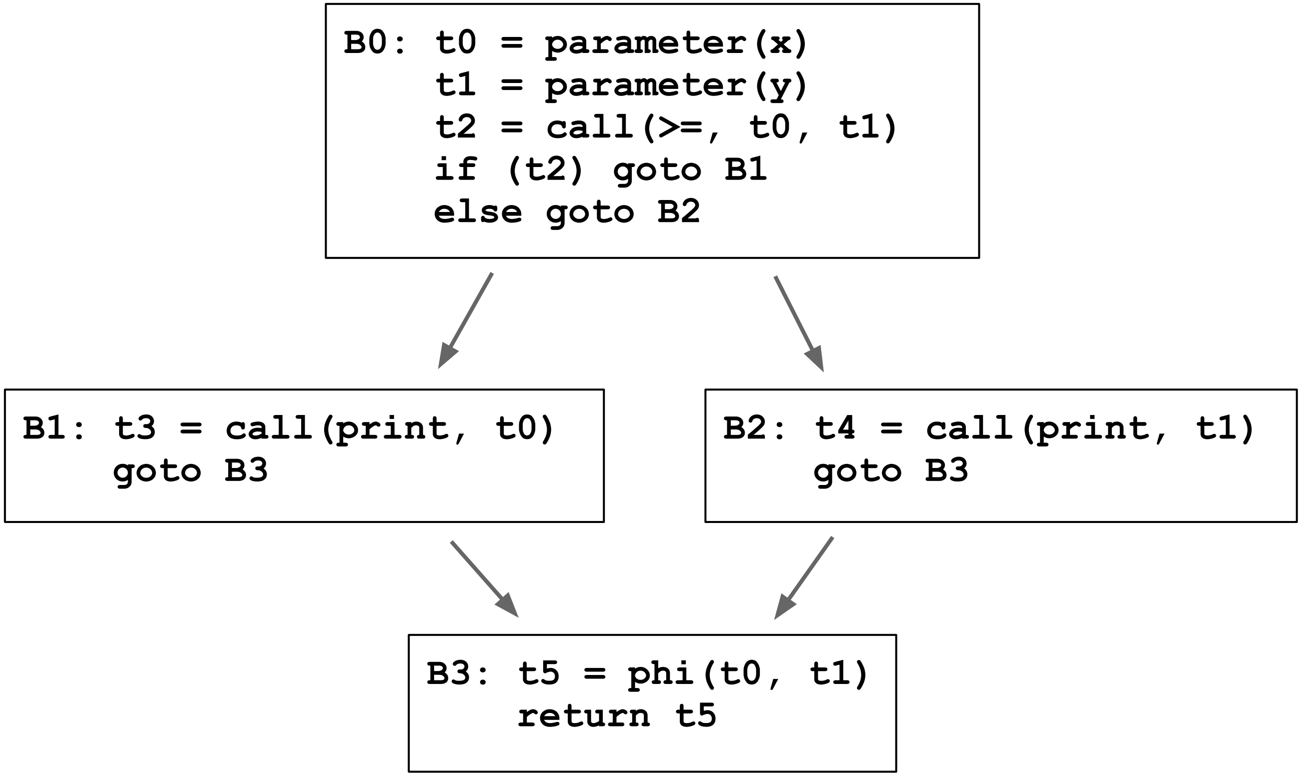
Code generator

JavaScript syntax tree

```
t0 = constant(2)
t1 = constant(3)
t2 = call(+, t0, t1)
```

SSA: Basic block graph

```
max(x, y) {
  var result;
  if (x >= y) {
    print(x);
    result = x;
  } else {
    print(y);
    result = y;
  }
  return result;
}
```



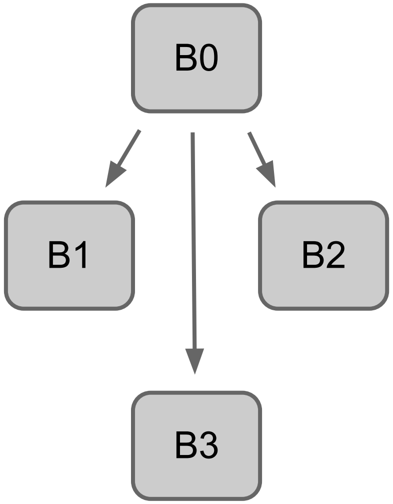
```
B0: t0 = parameter(x)
    t1 = parameter(y)
    t2 = call(>=, t0, t1)
    if (t2) goto B1
    else goto B2
```

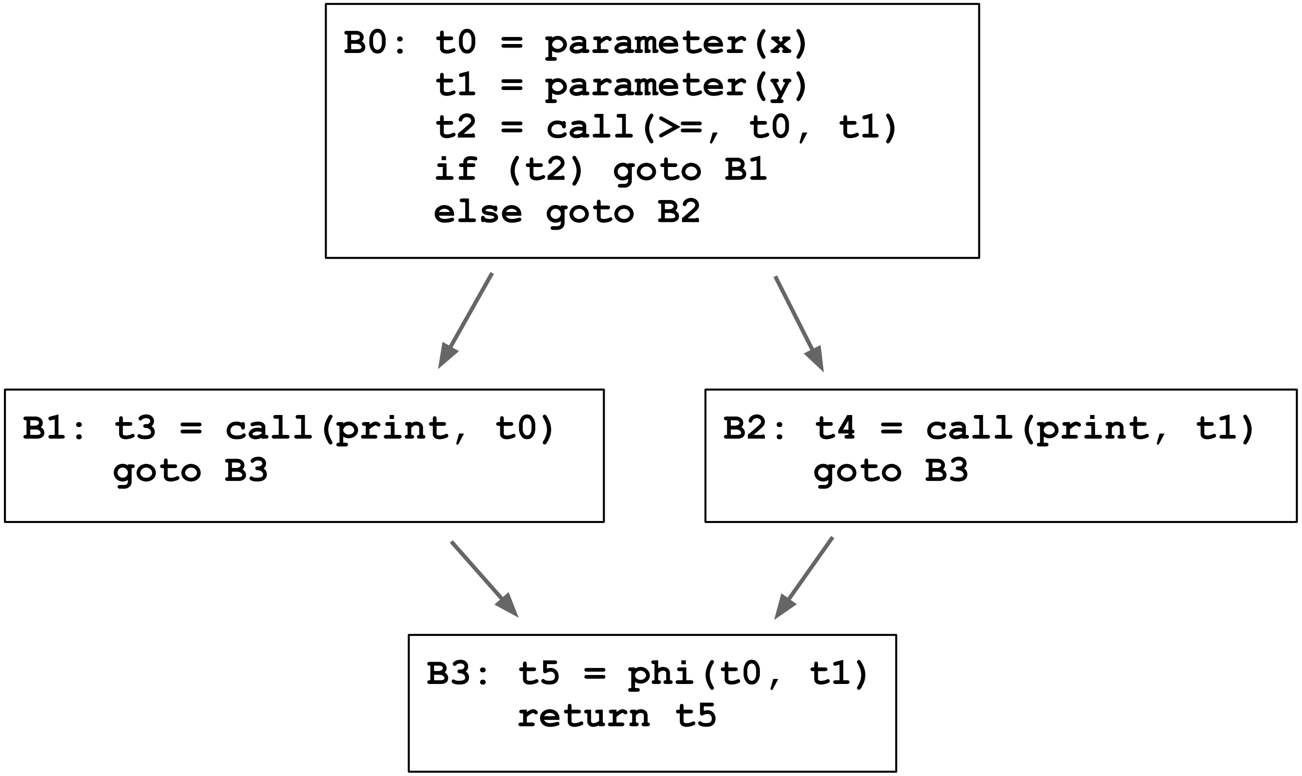
```
B1: t3 = call(print, t0)
    goto B3
```

```
B2: t4 = call(print, t1)
    goto B3
```

```
B3: t5 = phi(t0, t1)
    return t5
```

SSA: Dominator tree





```
B0: t0 = parameter(x)
    t1 = parameter(y)
    t2 = call(>=, t0, t1)
    if (t2) goto B1
    else goto B2
```

```
B1: t3 = call(print, t0)
    goto B3
```

```
B2: t4 = call(print, t1)
    goto B3
```

```
B3: t5 = phi(t0, t1)
    return t5
```

Optimizations

- Type propagation
- Function inlining
- Global value numbering
- Loop-invariant code motion

Global value numbering

- Two instructions are equal if they perform the same operation on the same inputs

- Executing an instruction can have or be affected by side-effects

- **Optimization:** Replace instructions with equal ones from dominators if no side-effects can affect the outcome

Global value numbering (1)

```
wat(x) => (x + 1) + (x + 1);
```

```
t0 = parameter(x, type = num)
t1 = constant(1)
t2 = call(+, t0, t1)
t3 = constant(1)
t4 = call(+, t0, t3)
t5 = call(+, t2, t4)
return t5
```

Global value numbering (2)

```
wat(x) => (x + 1) + (x + 1);
```

```
t0 = parameter(x, type = num)
t1 = constant(1)
t2 = call(+, t0, t1)
t3 = constant(1)
t4 = call(+, t0, t1)
t5 = call(+, t2, t4)
return t5
```

Global value numbering (3)

```
wat(x) => (x + 1) + (x + 1);
```

```
t0 = parameter(x, type = num)
t1 = constant(1)
t2 = call(+, t0, t1)

t4 = call(+, t0, t1)
t5 = call(+, t2, t2)
return t5
```

Global value numbering (4)

```
wat(x) => (x + 1) + (x + 1);
```

```
t0 = parameter(x, type = num)
t1 = constant(1)
t2 = call(+, t0, t1)
t5 = call(+, t2, t2)
return t5
```

Global value numbering (5)

```
wat(x) => (x + 1) + (x + 1);
```

```
$.wat = function(x) {
  var t2 = x + 1;
  return t2 + t2;
};
```

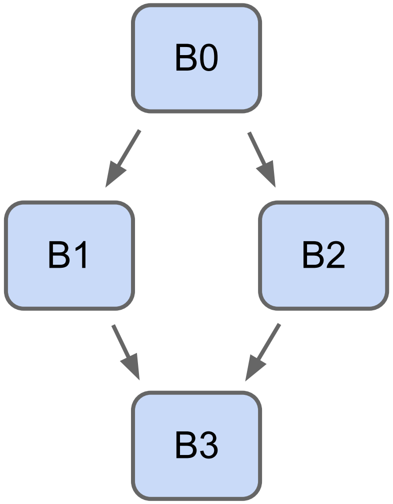
Global value numbering algorithm

- Walk the dominator tree while keeping a hash set of **live** values
  - Replace instructions with equal instructions from set
  - Add instructions that are not replaced to the set
  - Copy the set before visiting dominated children

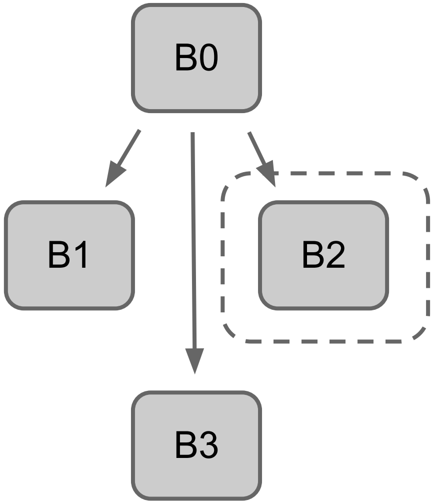
- When visiting an instruction that has side effects, **kill** all values in the set that are affected by those side effects

Global value numbering algorithm

Control flow graph



Dominator tree



Side-effects in B2 may kill values in the initial live set for B3 because B2 is on a control flow path from B0 to B3

Speculative optimizations

- Even after type propagation we may have instructions with unknown types
  - Cannot safely use primitive JavaScript operations
  - Don't know if the instructions have side-effects

- **Optimization:** Try to guess the type of an instruction based on its inputs and uses

Speculative optimizations (1)

It would be great if `x` was a JavaScript array

```
sum(x) {
  var result = 0;
  for (var i = 0; i < x.length; i++) {
    result += x[i];
  }
  return result;
}
```

Speculative optimizations (2)

We really hope `x` is a JavaScript array

```
$.sum = function(x) {
  if (!$.isJsArray(x)) return $.sum$bailout(1, x);
  var result = 0;
  for (var t1 = x.length, i = 0; i < t1; ++i) {
    if (i < 0 || i >= t1) throw $.ioore(i);
    var t2 = x[i];
    if (typeof t2 !== 'number') throw $.iae(t2);
    result += t2;
  }
  return result;
};
```

Speculative optimizations (3)

What if it turns out `x` is not a JavaScript array?

```
$.sum$bailout = function(state, x) {
  var result = 0;
  for (var i = 0; $.ltB(i, $.get$length(x)); ++i) {
    var t1 = $.index(x, i);
    if (typeof t1 !== 'number') throw $.iae(t1);
    result += t1;
  }
  return result;
};
```

Heuristics for speculating

- To avoid generating too much code we need to control the speculative optimizations

- Hard to strike the right balance between optimizing too little and too much

- **Current solution:** Only speculate about types for values that are used from within loops

Profile guided optimizations

What if we aggressively speculated about types and used profiling to figure out if it was helpful?

1. Use speculative optimizations everywhere!
2. Profile the resulting code
3. Re-compile with less speculation

Don't keep optimized methods that are rarely used or always bail out

Dealing with control flow

- It is hard to translate generic SSA graph to JavaScript (no arbitrary jumps)

- **Solution:** Try to keep track of the Dart code's structure and compile back to it

- Use a generic, but less efficient way when this is not possible

Dealing with control flow (1)

Is that an index bounds check in your condition?

```
sum(x) {
  var result = 0;
  for (var i = 0; x[i] != null; i++) {
    result += x[i];
  }
  return result;
}
```

Dealing with control flow (2)

Bounds check turns the condition into a statement

```
$.sum = function(x) {
  ...
  var t1 = x.length;
  var i = 0;
  while (true) {
    if (i < 0 || i >= t1) throw $.ioore(i);
    if (x[i] == null) break;
    ...
  }
  ...
};
```

Status

Code size

- Size of the generated code has improved since our first release!
- If your app translates to sizeable chunks of JavaScript it could be because of imports
- Work on supporting minification is in progress (use --minify option)

Performance

Goal chart	Scores			Relative to v8	
Benchmark	v8	dart	dart2js	dart	dart2js
DeltaBlue	279.72	368.50	190.31	131.74%	68.04%
Richards	400.30	566.22	281.36	141.45%	70.29%
NBody	15944.00	17513.50	10876.00	109.84%	68.21%
BinaryTrees	9.01	9.35	8.24	103.79%	91.47%
Mandelbrot	169.33	167.92	138.29	99.16%	81.67%
Fannkuch	3465.00	4325.50	3142.00	124.83%	90.68%
Meteor	6.69	5.60	2.19	83.75%	32.81%
BubbleSort	25237.50	26449.00	18222.00	104.80%	72.20%
Fibonacci	9198.50	13534.00	9405.50	147.13%	102.25%
Loop	34889.50	35319.00	35469.00	101.23%	101.66%
Permute	11082.00	16535.00	7519.50	149.21%	67.85%
Queens	117959.99	181779.51	98879.00	154.10%	83.82%
QuickSort	17107.50	15312.50	9403.50	89.51%	54.97%
Recurse	14019.50	20194.00	14424.00	144.04%	102.89%
Sieve	102290.50	114639.50	102462.50	112.07%	100.17%
Sum	74423.50	59832.00	75394.00	80.39%	101.30%
Tak	3064.00	4763.50	2490.00	155.47%	81.27%
Takl	8910.00	16699.50	8431.00	187.42%	94.62%
Towers	4919.50	5611.00	3107.00	114.06%	63.16%
TreeSort	7041.00	7933.00	5427.00	112.67%	77.08%
Geo. mean	4009.77	4785.62	3120.74	119.35%	77.83%

Conclusions

- You should write your web apps in Dart
  - Be more productive with a better toolchain
  - Deploy to all modern browsers through JavaScript
  - Let us worry about the low-level optimizations

- We want to improve the web platform!
  - Better support for programming in the large
  - Faster application startup in particular on mobile
  - More predictable and better runtime performance

Questions?