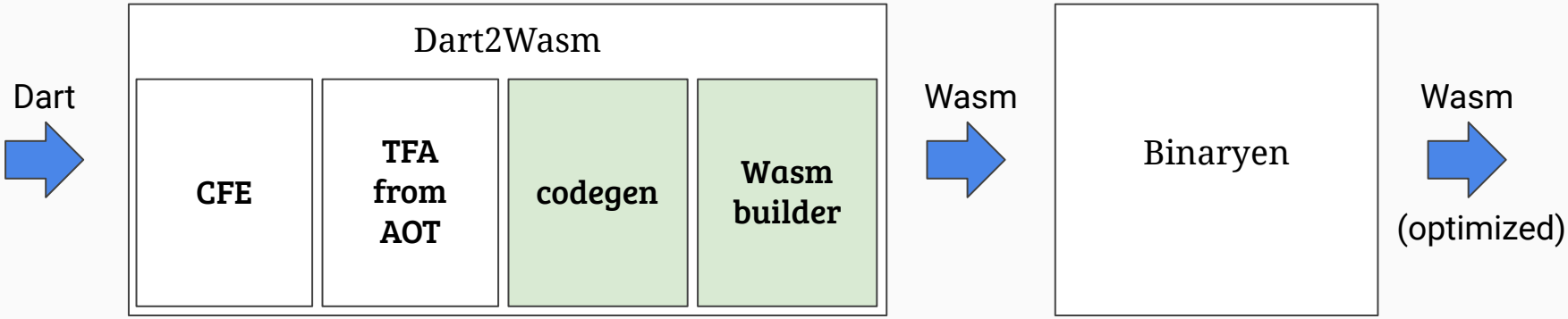
Compiling Dart to WasmGC

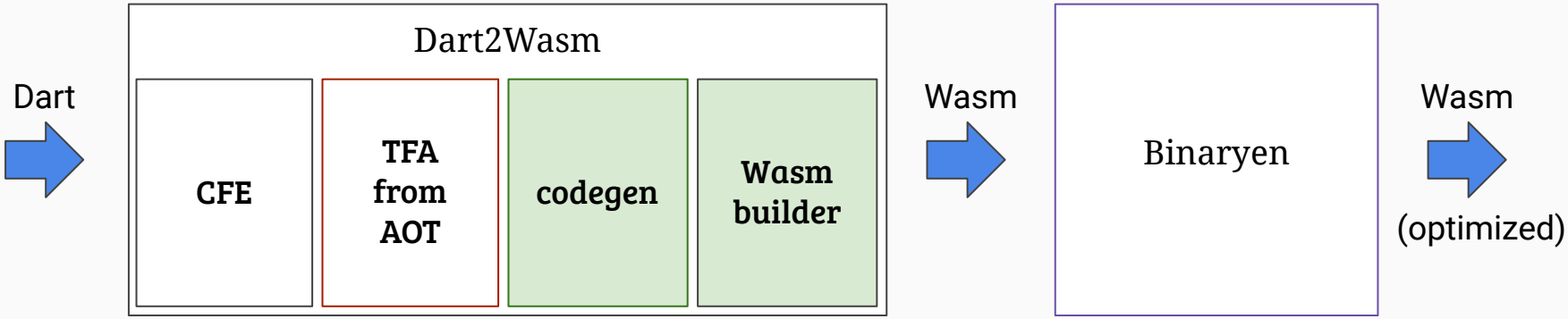
WasmGC subgroup meeting

Aske Simon Christensen, Google

Prototype architecture



Prototype architecture (optimizations)



Polyvariant type flow
Constant propagation
Nullability propagation
Tree shaking
Signature shaking
Devirtualization
Selector assignment

Type representation
Intrinsification
Direct field access
Tree shaking

Type canonicalization
Inlining
Dead code elimination
Redundant cast elimination
Local optimization
Allocation sinking
etc.

Method/getter/setter calls

if TFA identified single target
    if get or set of field
        Access field directly
    else if intrinsic available
        Emit intrinsic code
    else
        Emit direct call
else
    Emit dispatch table call

Dispatch table call:

```
Evaluate receiver
(local.tee temp)
Evaluate arguments
(local.get temp)
(struct.get #classId)
(i32.const selectorOffset)
(i32.add)
(call_indirect ...)
```

Type representation

- Each Dart class translated to a Wasm struct
  - Class ID
  - Identity hash code
  - One field per Dart field
  - One field per type parameter
- Type parameter fields of superclass reused for identity mappings
  - `class B<T> extends A<T>`
- Value types `bool`, `int`, `double`:
  - `i32`, `i64` and `f64` when non-nullable
  - Box classes when nullable (or joined with other types)

Variable types

Due to interfaces, variables can’t always be typed with their Wasm type!

- Wasm type for `C`: LUB of all classes implementing `C`
- Wasm type for type variable `T`: type for bound of `T`

Methods callable from the same call site must have the same Wasm function signature!

- Receiver type: LUB of all classes declaring the method
- Parameter/return types: LUB of all types for parameter/return
- Optional parameters implemented via caller-side defaults (for now)

Internal type conversions

- Same Dart type can have different Wasm representations!
  - Due to overrides, promotion, generics, constructor calls, etc.

```
void wrap(Expression node, w.ValueType expectedType) {
  w.ValueType resultType = node.accept1(this, expectedType);
  convertType(resultType, expectedType);
}
```

- Boxing / unboxing
- Null check
- Downcast
- Drop result when expecting void

Uses of expected type

- Type of `null` constant
- Only preserve value of assignment when needed
- Use of `this` (explicit or implicit):

- When receiver type is not current class:
  - Cast receiver to current class and store in separate local
- For each (explicit or implicit) use of `this`:
  - If original receiver local is precise enough, use that
  - Otherwise, use precise version
- Precise receiver gets optimized out when not used

Constants

- Simple: `bool`, `int`, `double`, `String`, `Symbol`, function
- Compound: instance, `List`, `Map`, `Set`, `Type`

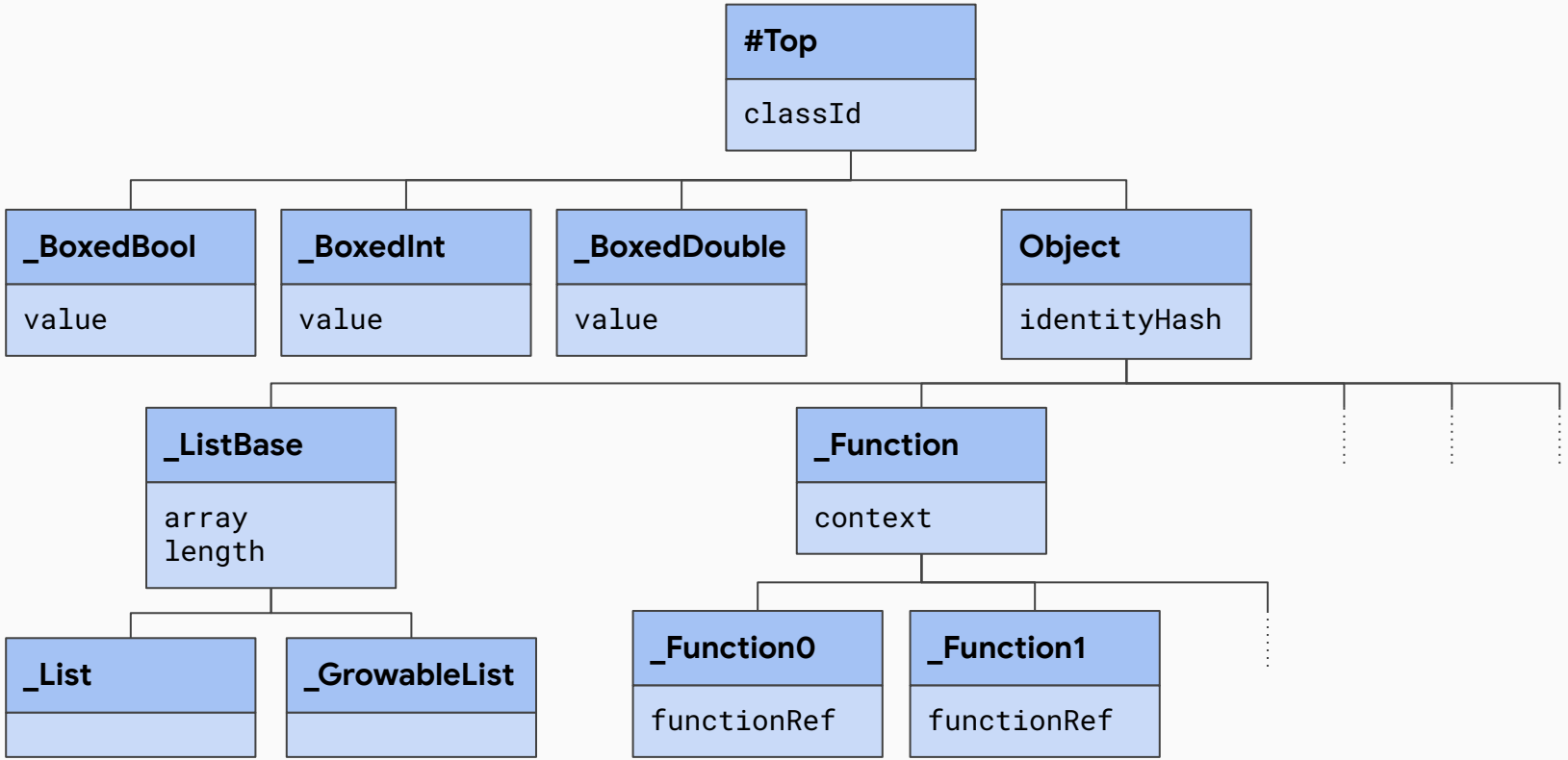
Tried two different strategies:

- Lazy: Read global, if null, call function to initialize
- Eager: Constants fully in global initializers
  - Using `struct.new`, `array.init`

Eager wins:

- 17% smaller, 20% faster early execution, 5% faster late execution
- Marginally slower startup (a few ms out of about 60 ms)

Dart2Wasm type hierarchy



Unsolved challenges

- Dynamic calls
  - No dynamic call instruction
- Async/await
  - No arbitrary goto
- Optional parameters
  - With arbitrary assignment of function objects
- Complex type tests
  - Interfaces, type arguments, function types, runtime types
- Using `i31ref`
  - Forces `num`, `Object`, `Comparable` to `eqref`
- Weak maps

?