Optimizing Dagger on Android

Ron Shapiro
ronsh@google
@rdshapiro

Prereqs

Prereqs

- You've used Dagger before

Prereqs

- You've used Dagger before
- Optional: Understanding Dagger 2's Generated Code
  - https://goo.gl/toV6Nq

Prereqs

- You've used Dagger before
- Optional: Understanding Dagger 2's Generated Code
  - https://goo.gl/toV6Nq
- You're not scared to look at generated code

Glossary

Glossary

- Key → Type + Optional<Qualifier>

Glossary

- Key → Type + Optional<Qualifier>
- Binding → Key + ImmutableSet<DependencyRequest>

Glossary

- Key → Type + Optional<Qualifier>
- Binding → Key + ImmutableSet<DependencyRequest>

```
@Provides
Airplane airplane(
```

Glossary

- Key → Type + Optional<Qualifier>
- Binding → Key + ImmutableSet<DependencyRequest>

```
@Provides
Airplane airplane(
    Body body, @Left Wing left, @Right Wing right) { … }
```

Glossary

- Key → Type + Optional<Qualifier>
- Binding → Key + ImmutableSet<DependencyRequest>

```
@Provides
Airplane airplane(
    Body body, @Left Wing left, @Right Wing right) { … }
```

- DependencyRequest → Key + Kind

Glossary

- Key → Type + Optional<Qualifier>
- Binding → Key + ImmutableSet<DependencyRequest>

```
@Provides
Airplane airplane(
    Body body, @Left Wing left, @Right Wing right) { … }
```

- DependencyRequest → Key + Kind
  - enum Kind { INSTANCE, PROVIDER, LAZY, … }

Glossary

- Key → (normalized) Type + Optional<Qualifier>
- Binding → Key + ImmutableSet<DependencyRequest>

```
@Provides
Airplane airplane(
    Body body, @Left Wing left, @Right Wing right) { … }
```

- DependencyRequest → Key + Kind
  - enum Kind { INSTANCE, PROVIDER, LAZY, … }

Glossary

- Key → (normalized) Type + Optional<Qualifier>
- Binding → Key + ImmutableSet<DependencyRequest>

```
@Provides
Airplane airplane(
    Body body, @Left Wing left, @Right Wing right) { … }

@Provides
XWing xWing(
    Body body, @Left Wing left, @Right Wing right) { … }
```

- DependencyRequest → Key + Kind
  - enum Kind { INSTANCE, PROVIDER, LAZY, … }

Glossary

- Key → (normalized) Type + Optional<Qualifier>
- Binding → Key + ImmutableSet<DependencyRequest>

```
@Provides
Airplane airplane(
    Body body, @Left Wing left, @Right Wing right) { … }

@Provides
XWing xWing(
    Body body, @Left Provider<Wing> left, @Right Provider<Wing> right) { … }
```

- DependencyRequest → Key + Kind
  - enum Kind { INSTANCE, PROVIDER, LAZY, … }

Glossary

- Key → (normalized) Type + Optional<Qualifier>
- Binding → Key + ImmutableSet<DependencyRequest>

```
@Provides
Airplane airplane(
    Body body, @Left Wing left, @Right Wing right) { … }

@Provides
XWing xWing(
    Body body, @Left Provider<Wing> left, @Right Provider<Wing> right,
    Logo logo) { … }
```

- DependencyRequest → Key + Kind
  - enum Kind { INSTANCE, PROVIDER, LAZY, … }

Glossary

- Key → (normalized) Type + Optional<Qualifier>
- Binding → Key + ImmutableSet<DependencyRequest>

```
@Provides
Airplane airplane(
    Body body, @Left Wing left, @Right Wing right) { … }

@Provides
XWing xWing(
    Body body, @Left Provider<Wing> left, @Right Provider<Wing> right,
    Lazy<Logo> logo) { … }
```

- DependencyRequest → Key + Kind
  - enum Kind { INSTANCE, PROVIDER, LAZY, … }

Life of a DependencyRequest

java/dagger/internal/codegen/BindingExpression.java

Life of a DependencyRequest

- How many bindings have this key?

java/dagger/internal/codegen/BindingExpression.java

Life of a DependencyRequest

- How many bindings have this key?
  - 0 → missing binding error

java/dagger/internal/codegen/BindingExpression.java

Life of a DependencyRequest

- How many bindings have this key?
  - 0 → missing binding error
  - 2+ → duplicate binding error

java/dagger/internal/codegen/BindingExpression.java

Life of a DependencyRequest

- How many bindings have this key?
  - 0 → missing binding error
  - 2+ → duplicate binding error

java/dagger/internal/codegen/BindingExpression.java

Life of a DependencyRequest

- How many bindings have this key?
  - 0 → missing binding error
  - 2+ → duplicate binding error

- How can Binding X be represented as DependencyRequest.Kind Y using X's own dependencies?

java/dagger/internal/codegen/BindingExpression.java

Life of a DependencyRequest

```
@Module
class AirplaneModule {
  @Provides
  Airplane weldAndCreate() {}
}
```

java/dagger/internal/codegen/SimpleMethodBindingExpression.java

Life of a DependencyRequest

```
@Module
class AirplaneModule {
  @Provides
  Airplane weldAndCreate(Body body) {}
}
```

java/dagger/internal/codegen/SimpleMethodBindingExpression.java

Life of a DependencyRequest

```
@Module
class AirplaneModule {
  @Provides
  Airplane weldAndCreate(
      Body body, @Left Wing leftWing, @Right Wing rightWing) {}
}
```

java/dagger/internal/codegen/SimpleMethodBindingExpression.java

Life of a DependencyRequest

```
@Module
class AirplaneModule {
  @Provides
  Airplane weldAndCreate(
      Body body, @Left Wing leftWing, @Right Wing rightWing,
      Welder welder) {}
}
```

java/dagger/internal/codegen/SimpleMethodBindingExpression.java

Life of a DependencyRequest

```
@Module
class AirplaneModule {
  @Provides
  Airplane weldAndCreate(
      Body body, @Left Wing leftWing, @Right Wing rightWing,
      Welder welder, Lazy<SafetyFeature> lazySafety) {}
}
```

java/dagger/internal/codegen/SimpleMethodBindingExpression.java

Life of a DependencyRequest

```
@Module
class AirplaneModule {
  @Provides
  Airplane weldAndCreate(
      Body body, @Left Wing leftWing, @Right Wing rightWing,
      Welder welder, Lazy<SafetyFeature> lazySafety) {
    welder.weld(body, leftWing); welder.weld(body, rightWing);
    if (LAWYERS_REQUIRE_SAFETY_FEATURE) {
      lazySafety.get().protect(body);
    }
    return new Airplane(body);
  }
}
```

java/dagger/internal/codegen/SimpleMethodBindingExpression.java

Life of a DependencyRequest

```
@Component(modules = {
  AirplaneModule.class,
  WingModule.class,
})
interface AirplaneComponent {
  Airplane createAirplane();
}
```

Life of a DependencyRequest

```
@Override
Airplane createAirplane() {
  airplaneModule.weldAndCreate(
    /* body */,
    /* left wing */,
    /* right wing */,
    /* welder */,
    /* safety feature */);
}
```

java/dagger/internal/codegen/SimpleMethodBindingExpression.java

Life of a DependencyRequest

```
@Override
Airplane createAirplane() {
  airplaneModule.weldAndCreate(
    new Body(),
    /* left wing */,
    /* right wing */,
    /* welder */,
    /* safety feature */);
}
```

java/dagger/internal/codegen/SimpleMethodBindingExpression.java

Life of a DependencyRequest

```
@Override
Airplane createAirplane() {
  airplaneModule.weldAndCreate(
    new Body(new Steel()),
    /* left wing */,
    /* right wing */,
    /* welder */,
    /* safety feature */);
}
```

java/dagger/internal/codegen/SimpleMethodBindingExpression.java

Life of a DependencyRequest

```
@Override
Airplane createAirplane() {
  airplaneModule.weldAndCreate(
    new Body(new Steel()),
    WingModule.createLeftWing(new Steel()),
    /* right wing */,
    /* welder */,
    /* safety feature */);
}
```

java/dagger/internal/codegen/SimpleMethodBindingExpression.java

Life of a DependencyRequest

```
@Override
Airplane createAirplane() {
  airplaneModule.weldAndCreate(
    new Body(new Steel()),
    WingModule.createLeftWing(new Steel()),
    WingModule.createRightWing(new Steel(), new Logo()),
    /* welder */,
    /* safety feature */);
}
```

java/dagger/internal/codegen/SimpleMethodBindingExpression.java

Life of a DependencyRequest

```
@Override
Airplane createAirplane() {
  airplaneModule.weldAndCreate(
    new Body(new Steel()),
    WingModule.createLeftWing(new Steel()),
    WingModule.createRightWing(new Steel(), new Logo()),
    welderProvider.get(),
    /* safety feature */);
}
```

java/dagger/internal/codegen/SimpleMethodBindingExpression.java
java/dagger/internal/codegen/FrameworkType.java

Life of a DependencyRequest

```
@Override
Airplane createAirplane() {
  airplaneModule.weldAndCreate(
    new Body(new Steel()),
    WingModule.createLeftWing(new Steel()),
    WingModule.createRightWing(new Steel(), new Logo()),
    welderProvider.get() /* 😨😨😨😨 */,
    /* safety feature */);
}
```

java/dagger/internal/codegen/SimpleMethodBindingExpression.java
java/dagger/internal/codegen/FrameworkType.java

Life of a DependencyRequest

```
@Override
Airplane createAirplane() {
  airplaneModule.weldAndCreate(
    new Body(new Steel()),
    WingModule.createLeftWing(new Steel()),
    WingModule.createRightWing(new Steel(), new Logo()),
    welderProvider.get() /* 😨😨😨😨 */,
    DoubleCheck.lazy(safetyProvider);
}
```

java/dagger/internal/codegen/SimpleMethodBindingExpression.java
java/dagger/internal/codegen/FrameworkType.java

Life of a DependencyRequest

```
@Override
Airplane createAirplane() {
  airplaneModule.weldAndCreate(
    new Body(new Steel()),
    WingModule.createLeftWing(new Steel()),
    WingModule.createRightWing(new Steel(), new Logo()),
    welderProvider.get() /* 😨😨😨😨 */,
    DoubleCheck.lazy(safetyProvider /* 😨😨😨😨😨😨😨😨😨😨 */);
}
```

java/dagger/internal/codegen/SimpleMethodBindingExpression.java
java/dagger/internal/codegen/FrameworkType.java

Life of a component initialization

```
private void initialize() {
  this.welderProvider =
}
```

java/dagger/internal/codegen/FrameworkFieldInitializer.java

Life of a component initialization

```
private void initialize() {
  this.welderProvider =
    new Provider<Welder>() {
      @Override
      public Welder get() {}
    };
}
```

java/dagger/internal/codegen/FrameworkFieldInitializer.java

Life of a component initialization

```
private void initialize() {
  this.welderProvider =
    new Provider<Welder>() {
      @Override
      public Welder get() {
        return new Welder(
          new Wire(), new ProtectiveShield(),
          flamethrowerProvider.get());
      }
    };
}
```

java/dagger/internal/codegen/FrameworkFieldInitializer.java

Life of a component initialization

```
private void initialize() {
  this.welderProvider = DoubleCheck.provider(
    new Provider<Welder>() {
      @Override
      public Welder get() {
        return new Welder(
          new Wire(), new ProtectiveShield(),
          flamethrowerProvider.get());
      }
    });
}
```

java/dagger/internal/codegen/FrameworkFieldInitializer.java

Life of a component initialization

```
private void initialize() {
  this.welderProvider = DoubleCheck.provider(
    new Provider<Welder>() {
    ^ error: [MemoryLeak] this might leak your *entire* component!
      @Override
      public Welder get() {
        return new Welder(
          new Wire(), new ProtectiveShield(),
          flamethrowerProvider.get());
      }
    });
}
```

^ shameless plug for ErrorProne
github.com/google/error-prone

java/dagger/internal/codegen/FrameworkFieldInitializer.java

Life of a component initialization

java/dagger/internal/codegen/FrameworkFieldInitializer.java

Life of a component initialization

```
private void initialize() {
```

java/dagger/internal/codegen/FrameworkFieldInitializer.java

Life of a component initialization

```
private void initialize() {
  this.wireProvider = Wire_Factory.create();
```

java/dagger/internal/codegen/FrameworkFieldInitializer.java

Life of a component initialization

```
private void initialize() {
  this.wireProvider = Wire_Factory.create();
  this.protectiveShieldProvider =
    ProtectiveShield_Factory.create();
```

java/dagger/internal/codegen/FrameworkFieldInitializer.java

Life of a component initialization

```
private void initialize() {
  this.wireProvider = Wire_Factory.create();
  this.protectiveShieldProvider =
    ProtectiveShield_Factory.create();
  this.welderProvider = DoubleChecker.provider(
    Welder_Factory.create(
      wireProvider,
      protectiveShieldProvider,
      flamethrowerProvider));
```

java/dagger/internal/codegen/FrameworkFieldInitializer.java

Life of a component initialization

```
private void initialize() {
  this.wireProvider = Wire_Factory.create();
  this.protectiveShieldProvider =
    ProtectiveShield_Factory.create();
  this.welderProvider = DoubleChecker.provider(
    Welder_Factory.create(
      wireProvider,
      protectiveShieldProvider,
      flamethrowerProvider));
}
```

java/dagger/internal/codegen/FrameworkFieldInitializer.java

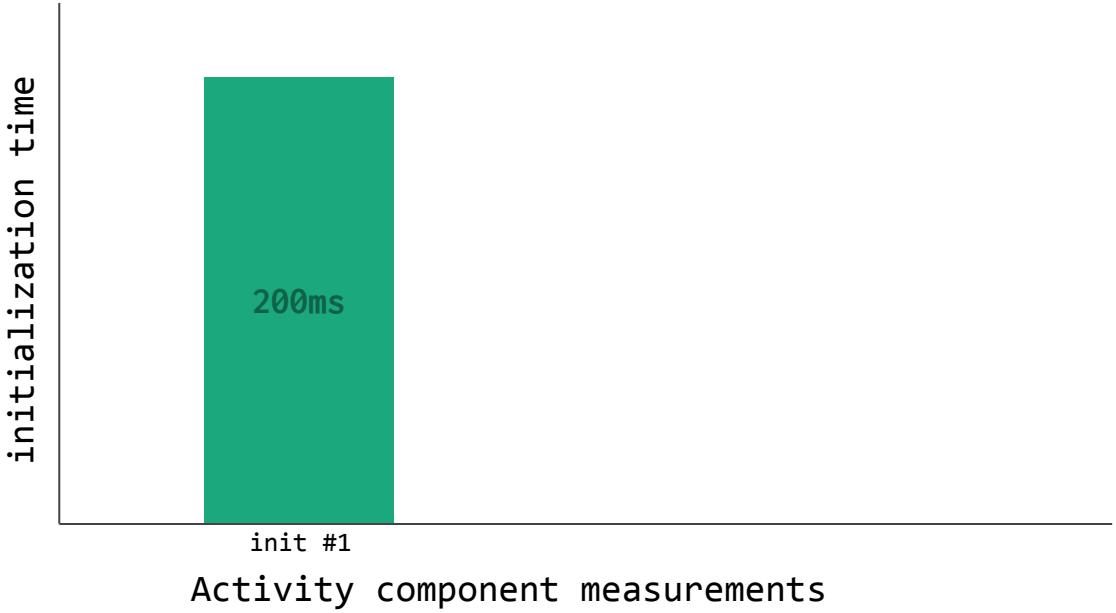
Why does this matter?

tl;dr: component initialization is

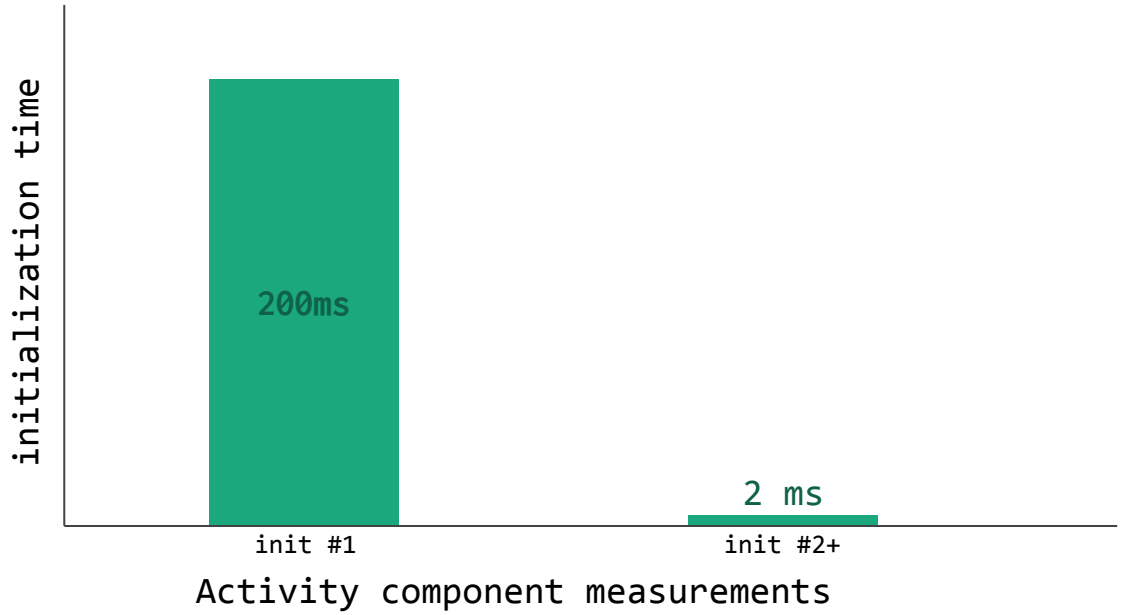
tl;dr: component initialization is *s l o w*

But just *how slow* is it?

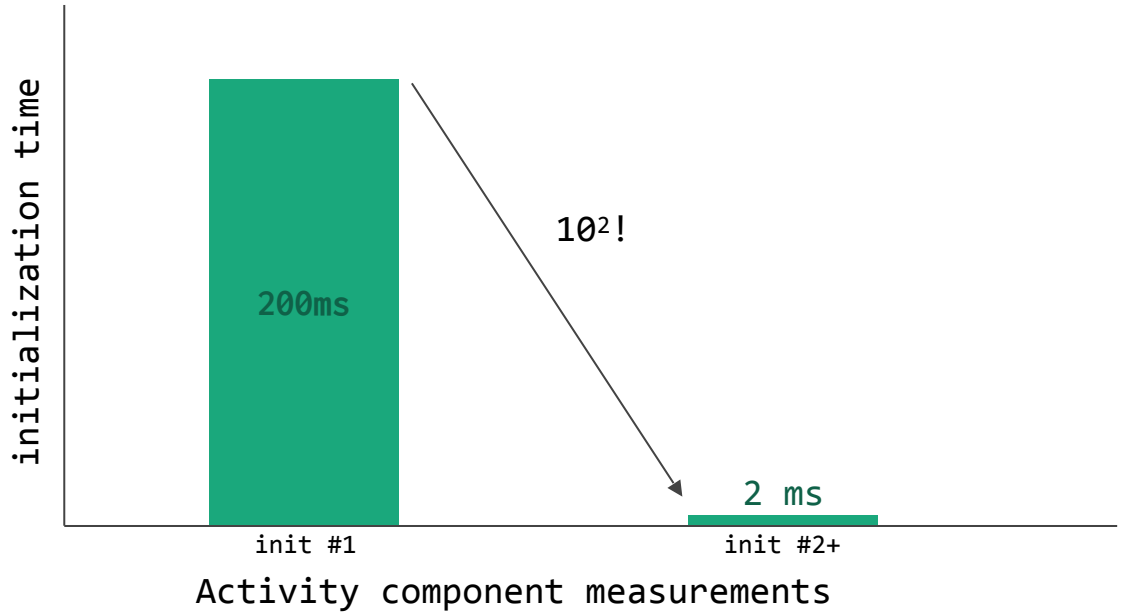
But just *how slow* is it?



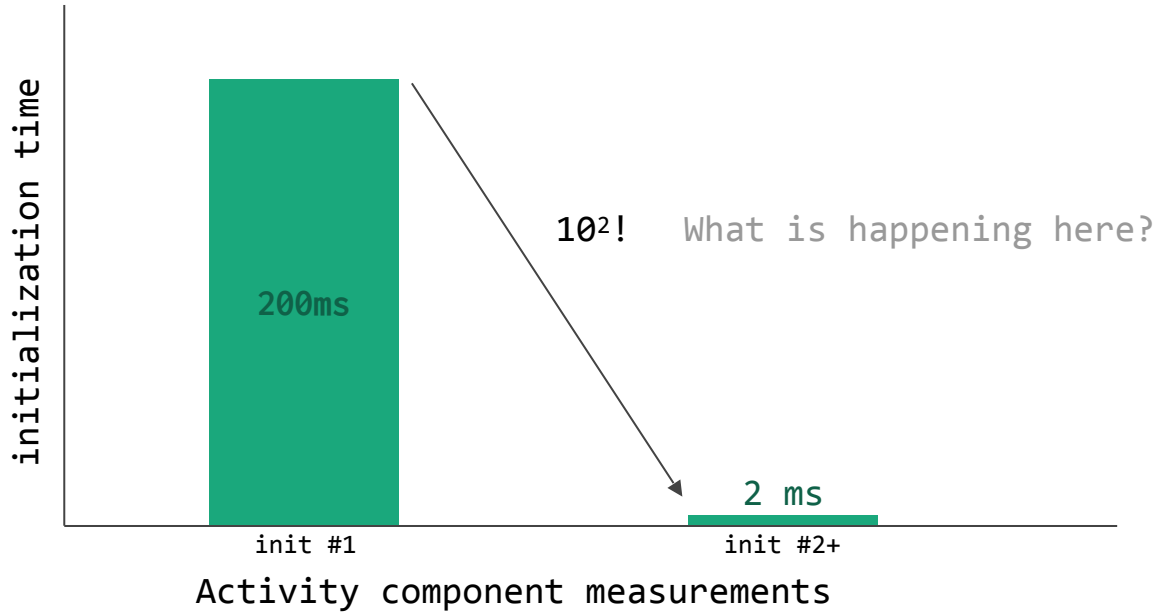
But just *how slow* is it?



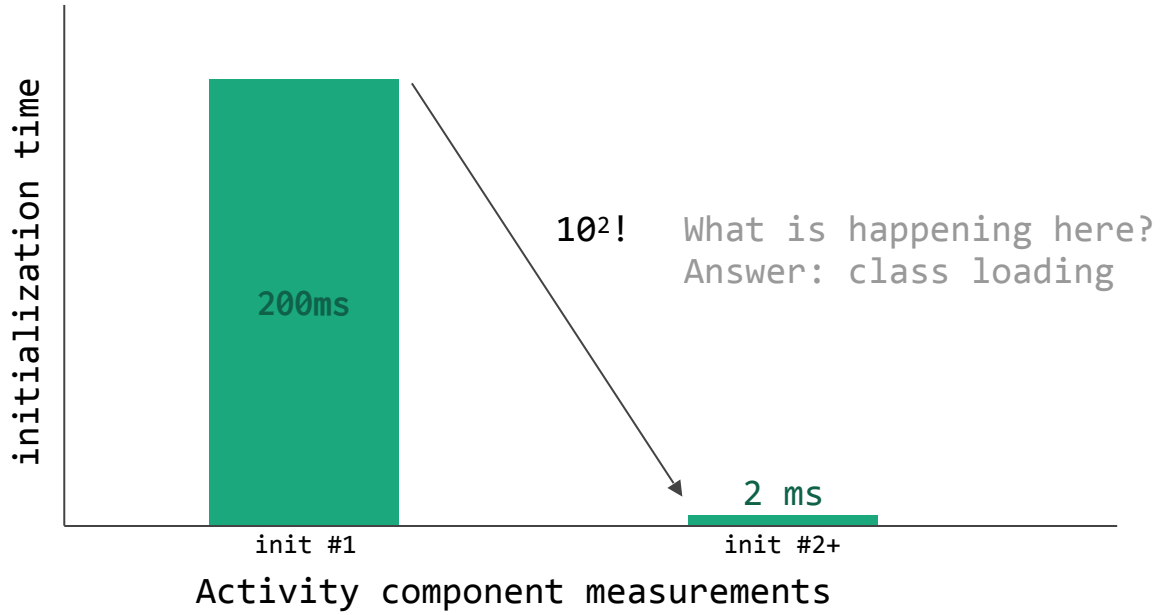
But just *how slow* is it?



But just *how slow* is it?



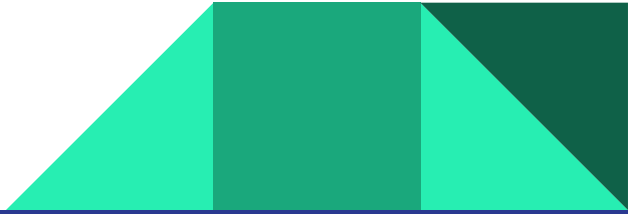
But just *how slow* is it?



How can we load fewer classes?

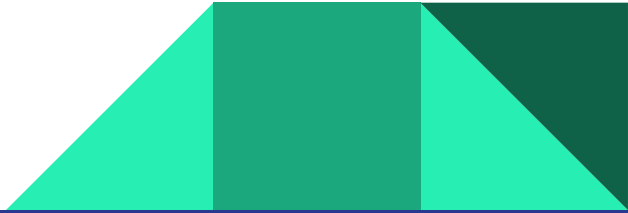
How can we load fewer classes?

- `com.google.dagger:dagger-compiler:2.12`



How can we load fewer classes?

- `com.google.dagger:dagger-compiler:2.12`
  - coming very soon!

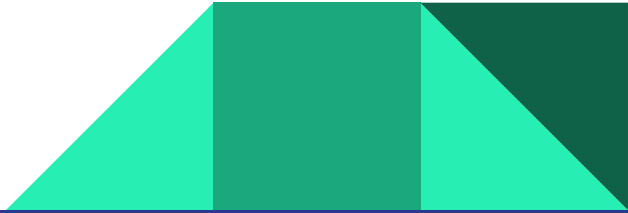


Dagger 2.12

Bindings

Provider form

Inlined form



java/dagger/internal/codegen/SetBindingExpression.java

Dagger 2.12

Bindings

```
@Provides @IntoSet T single() { … }
@Provides @ElementsIntoSet Set<T> many() { … }
```

Provider form

Inlined form

java/dagger/internal/codegen/SetBindingExpression.java

Dagger 2.12

Bindings

```
@Provides @IntoSet T single() { … }
@Provides @ElementsIntoSet Set<T> many() { … }
```

Provider form

```
SetFactory.builder()
  .add(Single_Factory.create())
  .addAll(Many_Factory.create())
  .build();
```

Inlined form

java/dagger/internal/codegen/SetBindingExpression.java

Dagger 2.12

Bindings

```
@Provides @IntoSet T single() { … }
@Provides @ElementsIntoSet Set<T> many() { … }
```

Provider form

```
SetFactory.builder()
  .add(Single_Factory.create())
  .addAll(Many_Factory.create())
  .build();
```

Inlined form

```
ImmutableSet.builder()
  .add(single())
  .addAll(many())
  .build();
```

java/dagger/internal/codegen/SetBindingExpression.java

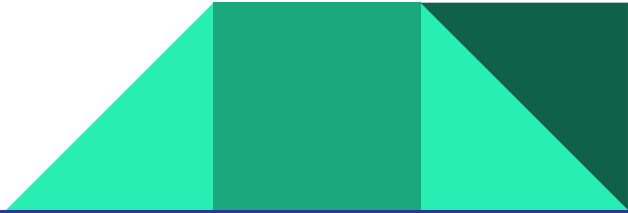
Dagger 2.12

Bindings

Provider form

Inlined form

java/dagger/internal/codegen/MapBindingExpression.java



Dagger 2.12

Bindings

```
@Provides @IntoMap @IntKey(1) V first() { … }
@Provides @IntoMap @IntKey(2) V second() { … }
```

Provider form

Inlined form

java/dagger/internal/codegen/MapBindingExpression.java

Dagger 2.12

Bindings

```
@Provides @IntoMap @IntKey(1) V first() { … }
@Provides @IntoMap @IntKey(2) V second() { … }
```

Provider form

```
MapFactory.builder()
  .put(1, First_Factory.create())
  .put(2, Second_Factory.create())
  .build();
```

Inlined form

java/dagger/internal/codegen/MapBindingExpression.java

Dagger 2.12

Bindings

```
@Provides @IntoMap @IntKey(1) V first() { … }
@Provides @IntoMap @IntKey(2) V second() { … }
```

Provider form

```
MapFactory.builder()
  .put(1, First_Factory.create())
  .put(2, Second_Factory.create())
  .build();
```

Inlined form

```
ImmutableMap.builder()
  .put(1, first())
  .put(2, second())
  .build();
```

java/dagger/internal/codegen/MapBindingExpression.java

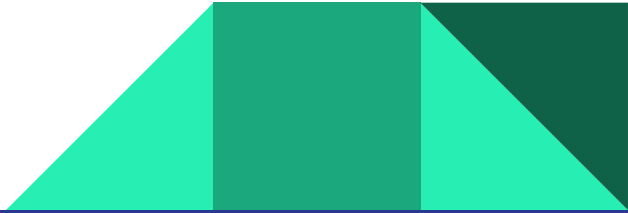
Dagger 2.12

Bindings

Provider form

Inlined form

java/dagger/internal/codegen/OptionalBindingExpression.java



Dagger 2.12

Bindings

```
@BindsOptionalOf DebugLogger sometimes();
```

Provider form

Inlined form

java/dagger/internal/codegen/OptionalBindingExpression.java

Dagger 2.12

Bindings

```
@BindsOptionalOf DebugLogger sometimes();
```

Provider form

```
new Provider<Optional<DebugLogger>>() {
  @Override
  public Optional<DebugLogger> get() {
    return Optional.of(debugLoggerProvider.get());
  }
}
```

Inlined form

java/dagger/internal/codegen/OptionalBindingExpression.java

Dagger 2.12

Bindings

```
@BindsOptionalOf DebugLogger sometimes();
```

Provider form

```
new Provider<Optional<DebugLogger>>() {
  @Override
  public Optional<DebugLogger> get() {
    return Optional.of(debugLoggerProvider.get());
  }
}
```

Inlined form

```
Optional.of(DebugModule.logger());
```

java/dagger/internal/codegen/OptionalBindingExpression.java

Dagger 2.12

Bindings

Provider form

Inlined form

java/dagger/internal/codegen/ComponentProvisionBindingExpression.java

Dagger 2.12

Bindings

```
@Component
interface DatabaseComponent {
  Database database();
}
```

```
@Component(
    dependencies = DatabaseComponent.class)
interface NetworkingComponent {
  UsesDatabase usesDatabase();
}
```

Provider form

Inlined form

java/dagger/internal/codegen/ComponentProvisionBindingExpression.java

Dagger 2.12

Bindings

```
@Component
interface DatabaseComponent {
  Database database();
}
```

```
@Component(
    dependencies = DatabaseComponent.class)
interface NetworkingComponent {
  UsesDatabase usesDatabase();
}
```

Provider form

```
new Provider<Database>() {
  @Override
  public Database get() {
    return databaseComponent.database();
  }
};
```

Inlined form

java/dagger/internal/codegen/ComponentProvisionBindingExpression.java

Dagger 2.12

Bindings

```
@Component
interface DatabaseComponent {
  Database database();
}
```

```
@Component(
    dependencies = DatabaseComponent.class)
interface NetworkingComponent {
  UsesDatabase usesDatabase();
}
```

Provider form

```
new Provider<Database>() {
  @Override
  public Database get() {
    return databaseComponent.database();
  }
};
```

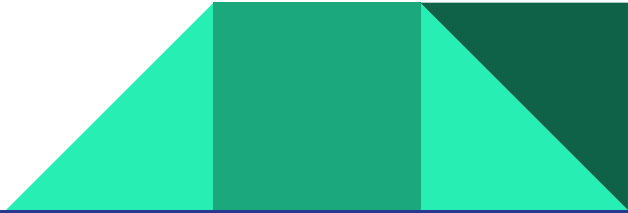
Inlined form

```
databaseComponent.database();
```

java/dagger/internal/codegen/ComponentProvisionBindingExpression.java

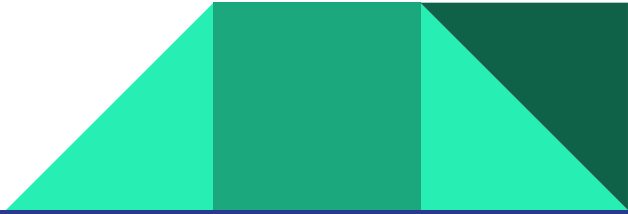
How can we load fewer classes?

- `com.google.dagger:dagger-compiler:2.12`
  - coming very soon!



How can we load fewer classes?

- com.google.dagger:dagger-compiler:2.12
  - coming very soon!
- Unnecessary Provider/Lazy usage



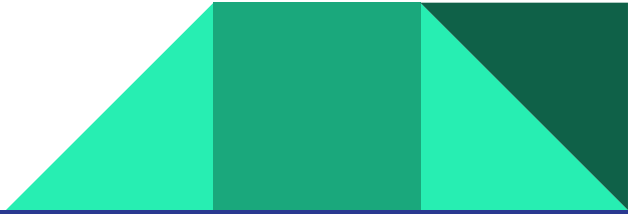
Don't force Dagger's hand

```
public class DroidconScheduleActivity extends Activity {
  @Inject Provider<Schedule> scheduleProvider;

  @Override
  public void onCreate() {
    scheduleView.setSchedule(scheduleProvider.get());
  }
}
```

Don't force Dagger's hand

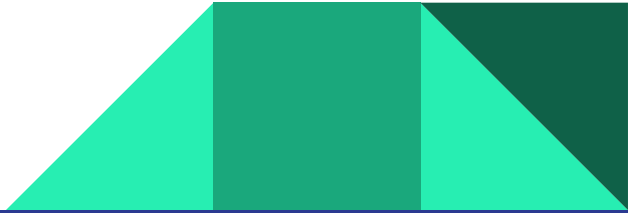
Don't force Dagger's hand

```
If you inject…
```



Don't force Dagger's hand

```
If you inject…
  || Provider<YourBinding>
```



Don't force Dagger's hand

```
If you inject…
  || Provider<YourBinding>
  || Lazy<YourBinding>
```

Don't force Dagger's hand

```
If you inject…
  || Provider<YourBinding>
  || Lazy<YourBinding>
```

Don't force Dagger's hand

```
If you inject…
  || Provider<YourBinding>
  || Lazy<YourBinding>

And
```

Don't force Dagger's hand

```
If you inject…
  || Provider<YourBinding>
  || Lazy<YourBinding>

And
  || you always call get() exactly once (especially in the constructor / onCreate())
```

Don't force Dagger's hand

```
If you inject…
  || Provider<YourBinding>
  || Lazy<YourBinding>

And
  || you always call get() exactly once (especially in the constructor / onCreate())
  || YourBinding is not expensive to create, but you aren't sure if you need it
```

Don't force Dagger's hand

```
If you inject…
  || Provider<YourBinding>
  || Lazy<YourBinding>

And
  || you always call get() exactly once (especially in the constructor / onCreate())
  || YourBinding is not expensive to create, but you aren't sure if you need it
```

Don't force Dagger's hand

```
If you inject…
  || Provider<YourBinding>
  || Lazy<YourBinding>

And
  || you always call get() exactly once (especially in the constructor / onCreate())
  || YourBinding is not expensive to create, but you aren't sure if you need it

Instead
```

Don't force Dagger's hand

```
If you inject…
  || Provider<YourBinding>
  || Lazy<YourBinding>

And
  || you always call get() exactly once (especially in the constructor / onCreate())
  || YourBinding is not expensive to create, but you aren't sure if you need it

Instead
  &&  Simply inject YourBinding with no wrapper type
```

Don't force Dagger's hand

```
If you inject…
  || Provider<YourBinding>
  || Lazy<YourBinding>

And
  || you always call get() exactly once (especially in the constructor / onCreate())
  || YourBinding is not expensive to create, but you aren't sure if you need it

Instead
  &&  Simply inject YourBinding with no wrapper type
  &&  Profile your code!
```

Don't force Dagger's hand

```
public class DroidconScheduleActivity extends Activity {
  @Inject Lazy<Schedule> lazySchedule;

  @Override
  public void onCreate() {
    scheduleView.setSchedule(lazySchedule.get());
  }
}
```

Don't force Dagger's hand

```
public class DroidconScheduleActivity extends Activity {
  @Inject Lazy<Schedule> lazySchedule;
  @Inject Lazy<Schedule> lazySchedule2;

  @Override
  public void onCreate() {
    scheduleView.setSchedule(lazySchedule.get());
  }
}
```

Don't force Dagger's hand

```
public class DroidconScheduleActivity extends Activity {
  @Inject Lazy<Schedule> lazySchedule;
  @Inject Lazy<Schedule> lazySchedule2;

  @Override
  public void onCreate() {
    assertThat(lazySchedule.get());
  }
}
```

Don't force Dagger's hand

```
public class DroidconScheduleActivity extends Activity {
  @Inject Lazy<Schedule> lazySchedule;
  @Inject Lazy<Schedule> lazySchedule2;

  @Override
  public void onCreate() {
    assertThat(lazySchedule.get())
      .isNotSameAs(lazySchedule2.get());
  }
}
```

Don't force Dagger's hand

```
@Inject Provider<YourBinding> myProvider;

YourBinding[] array = new YourBinding[4];

for (i = 0; i < 4; i++) {
  array[i] = myProvider.get();
}
```

Don't force Dagger's hand

```
@Inject Provider<YourBinding> myProvider;
@Inject YourBinding yb0;
@Inject YourBinding yb1;
@Inject YourBinding yb2;
@Inject YourBinding yb3;

YourBinding[] array = new YourBinding[4];

for (i = 0; i < 4; i++) {
  array[i] = myProvider.get();
}
```

Don't force Dagger's hand

```
@Inject Provider<YourBinding> myProvider;
@Inject YourBinding yb0;
@Inject YourBinding yb1;
@Inject YourBinding yb2;
@Inject YourBinding yb3;

YourBinding[] array = {yb0, yb1, yb2, yb3};
```

Don't force Dagger's hand

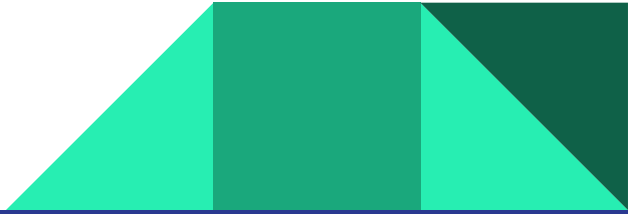
```
@Inject YourBinding yb0;
@Inject YourBinding yb1;
@Inject YourBinding yb2;
@Inject YourBinding yb3;

YourBinding[] array = {yb0, yb1, yb2, yb3};
```

How can we load fewer classes?

- com.google.dagger:dagger-compiler:2.12
  - coming very soon!
- Unnecessary Provider/Lazy usage

How can we load fewer classes?

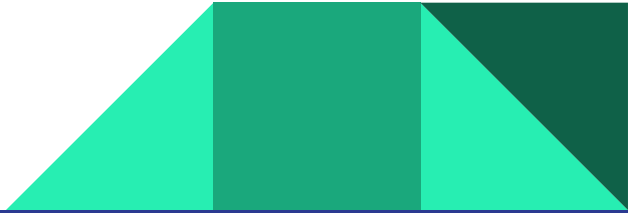
- com.google.dagger:dagger-compiler:2.12
  - coming very soon!
- Unnecessary Provider/Lazy usage
- Reconsider scoping decisions



Reconsider scoping

Reconsider scoping

```
@AnalyticsScope
class AnalyticsTracker {
```



Reconsider scoping

```
@AnalyticsScope
class AnalyticsTracker {
  @Inject AnalyticsTracker(
    UnsyncedAnalyticsQueue queue,
    String flowName,
    ViewToAnalyticsIdTranslator viewTranslator) { … }
```

Reconsider scoping

```
@AnalyticsScope
class AnalyticsTracker {
  @Inject AnalyticsTracker(
    UnsyncedAnalyticsQueue queue,
    String flowName,
    ViewToAnalyticsIdTranslator viewTranslator) { … }

  void trackClick(View view) {
    queue.add(flowName, viewTranslator.translate(view)); }
```

Reconsider scoping

```
@AnalyticsScope
class AnalyticsTracker {
  @Inject AnalyticsTracker(
    UnsyncedAnalyticsQueue queue,
    String flowName,
    ViewToAnalyticsIdTranslator viewTranslator) { … }

  void trackClick(View view) {
    queue.add(flowName, viewTranslator.translate(view)); }

  void onDetach() { queue.sendAnalyticsToServer(); }
}
```

Reconsider scoping

```
@AnalyticsScope
class AnalyticsTracker {
  @Inject AnalyticsTracker(
    UnsyncedAnalyticsQueue queue,
    String flowName,
    ViewToAnalyticsIdTranslator viewTranslator) { … }
}
```

Reconsider scoping

```
@AnalyticsScope
class AnalyticsTracker {
  @Inject AnalyticsTracker(
    UnsyncedAnalyticsQueue queue,
    String flowName,
    ViewToAnalyticsIdTranslator viewTranslator) { … }
}


class UnsyncedAnalyticsQueue {}
```

Reconsider scoping

```
class AnalyticsTracker {
  @Inject AnalyticsTracker(
    UnsyncedAnalyticsQueue queue,
    String flowName,
    ViewToAnalyticsIdTranslator viewTranslator) { … }
}

@AnalyticsScope
class UnsyncedAnalyticsQueue {}
```

Reconsider scoping

```
@Singleton
@Component(modules = ExecutorModule.class)
interface NetworkingComponent { … }

@Module
abstract class ExecutorModule {
  @Provides
  @Singleton
  @Background Executor provideBackgroundExecutor() {
    return Executors.newCachedThreadPool(4);
  }
}
```

Reconsider scoping

```
@Singleton
@Component(modules = ExecutorModule.class)
interface NetworkingComponent {
  // …
}
```

Reconsider scoping

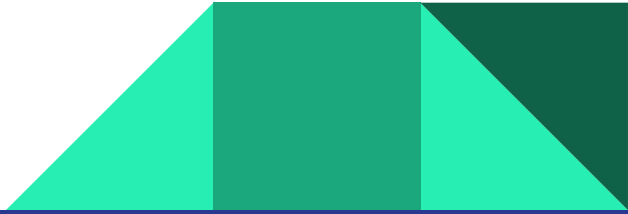
```
@Singleton
@Component
interface NetworkingComponent {
  // …




}
```

Reconsider scoping

```
@Singleton
@Component
interface NetworkingComponent {
  // …

  @Component.Builder
  interface Builder {
    @BindsInstance Builder executor(Executor e);
    NetworkingComponent build();
  }
}
```

How can we load fewer classes?

- com.google.dagger:dagger-compiler:2.12
  - coming very soon!
- Unnecessary Provider/Lazy usage
- Reconsider scoping decisions



How can we load fewer classes?

- com.google.dagger:dagger-compiler:2.12
  - coming very soon!
- Unnecessary Provider/Lazy usage
- Reconsider scoping decisions
- Suggest better codegen strategies
  - github.com/google/dagger/issues

Dagger on Android

Stop making me think about injection

Dagger on Android

```
public class LoginActivity extends Activity {
  @Inject OAuthClient oauth;
  @Inject PasswordEncrypter encrypter;

  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    ((DroidconApp) getApplication())
        .getComponent()
        .inject(this);
  }
}
```

Dagger on Android

```
public class DroidconApp extends Application {
  private DroidconComponent component;

  public DroidconComponent getComponent() {
    if (component == null) {
      component = DaggerDroidconComponent.create();
    }
    return component;
  }
}
```

dagger.android

```
public class LoginActivity extends DaggerActivity {
  @Inject OAuthClient oauth;
  @Inject PasswordEncrypter encrypter;
}

public class DroidconApp extends DaggerApplication {}

@Module
interface LoginModule {
  @ContributesAndroidInjector
  LoginActivity configureLoginActivityInjection();
}
```

dagger.android

- DaggerActivity
- DaggerFragment
- DaggerService + DaggerIntentService
- DaggerBroadcastReceiver
- DaggerApplication

- DaggerAppCompatActivity
- dagger.android.support.DaggerFragment

- @ContributesAndroidInjector

dagger.android under the hood

```
@ContributesAndroidInjector(modules = LoginSpecificModule.class)
LoginActivity loginActivity();


@Subcomponent(modules = LoginSpecificModule.class)
interface LoginActivitySubcomponent extends AndroidInjector<LoginActivity> {
  @Subcomponent.Builder
  abstract class Builder extends AndroidInjector.Builder<LoginActivity> {}
}
```

Faster builds w/ Dagger

Compile as little as possible

- When you have a hammer, everything looks like a @Subcomponent
  - but subcomponents require compilation at the root!

- Break up compilation where possible
  - gradle users: make lots of modules
  - bazel users: split up java_librarys

Compile as little as possible

Compile as little as possible

```
@Component(modules = NetworkingModule.class)
interface NetworkingComponent {
  OkHttpClient httpClient();
}
```

Compile as little as possible

```
@Component(modules = NetworkingModule.class)
interface NetworkingComponent {
  OkHttpClient httpClient();
}
```

Compile as little as possible

```
@Component(modules = NetworkingModule.class)
interface NetworkingComponent {
  OkHttpClient httpClient();
}


@Component(dependencies = NetworkingComponent.class)
interface AppComponent {
  ApiService apiService();
}
```

Compile as little as possible

```
@Component(modules = NetworkingModule.class)
interface NetworkingComponent {
  OkHttpClient httpClient();
}


@Component(dependencies = NetworkingComponent.class)
interface AppComponent {
  ApiService apiService();
}
```

Compile as little as possible

```
@Component(modules = NetworkingModule.class)
interface NetworkingComponent {
  OkHttpClient httpClient();
}


@Component(dependencies = NetworkingComponent.class)
interface AppComponent {
  ApiService apiService();
}

class ApiService {
  @Inject ApiService(OkHttpClient httpClient) { … }
}
```

Compile as little as possible

```
@Component(modules = NetworkingModule.class)
interface NetworkingComponent {
  OkHttpClient httpClient();
}

--------------------------------------------------------------------------------

@Component(dependencies = NetworkingComponent.class)
interface AppComponent {
  ApiService apiService();
}

class ApiService {
  @Inject ApiService(OkHttpClient httpClient) { … }
}
```

Compile as little as possible

```
@Component(modules = NetworkingModule.class)
interface NetworkingComponent {
  OkHttpClient httpClient();
}

--------------------------------------------------------------------------------

@Component(dependencies = NetworkingComponent.class)
interface AppComponent {
  ApiService apiService();
}

class ApiService {
  @Inject ApiService(OkHttpClient httpClient) { … }
}
```

separate compilations!

Compile as little as possible

```
public final class DaggerAppComponent implements AppComponent {
  private NetworkingComponent networkComponentDependency;

  @Override
  public void ApiService apiService() {
    return new ApiService(networkComponentDependency.httpClient());
  }
}
```

Summarize:
Read the generated code

Questions?

ronsh@google
@rdshapiro