Jesse Wilson from Square

Dagger

A fast dependency injector
for Android and Java.

Introduction

Motivation

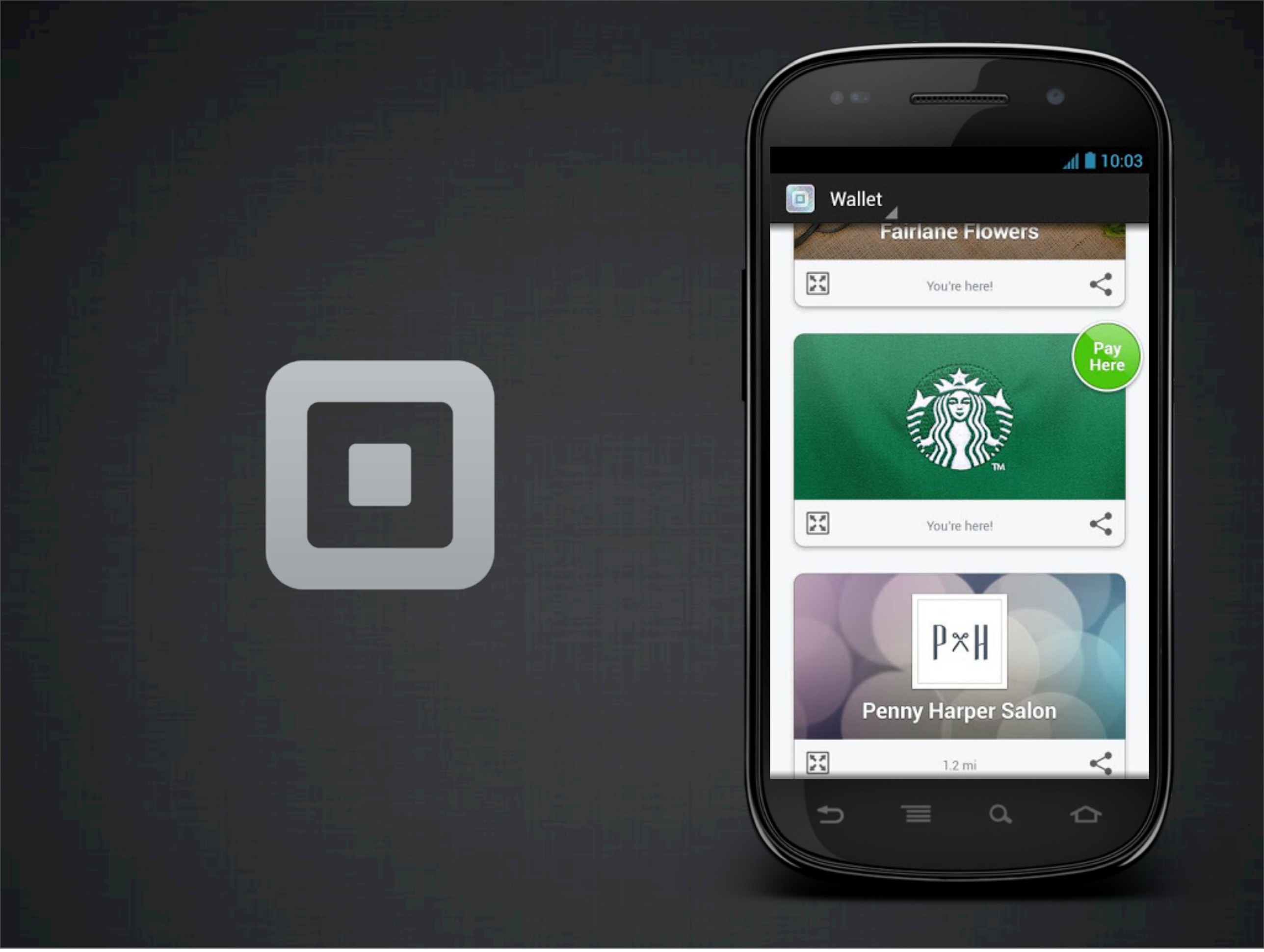
Using Dagger

Inside Dagger

Wrapping Up

Jesse

- Guice
- javax.inject
- Android Core Libraries


10:03
Wallet
Fairlane Flowers
You're here!
Pay Here
You're here!
Penny Harper Salon
1.2 mi

We're Hiring!

squareup.com/careers

Introduction

Motivation

Using Dagger

Inside Dagger

Wrapping Up

Motivation for Dependency Injection

- Decouple concrete from concrete
- Uniformity

Dependency Injection Choices

- PicoContainer
- Spring
- Guice
- javax.inject

(cc) creative commons from flickr.com/photos/getbutterfly/6317955134/

(cc) creative commons from flickr.com/photos/wikidave/2988710537/

We still love you, Froyo

- Eager vs. lazy graph construction
- Reflection vs. codegen

“I didn’t really expect anyone to use [git] because it’s so hard to use, but that turns out to be its big appeal.

No technology can ever be too arcane or complicated for the black t-shirt crowd.”

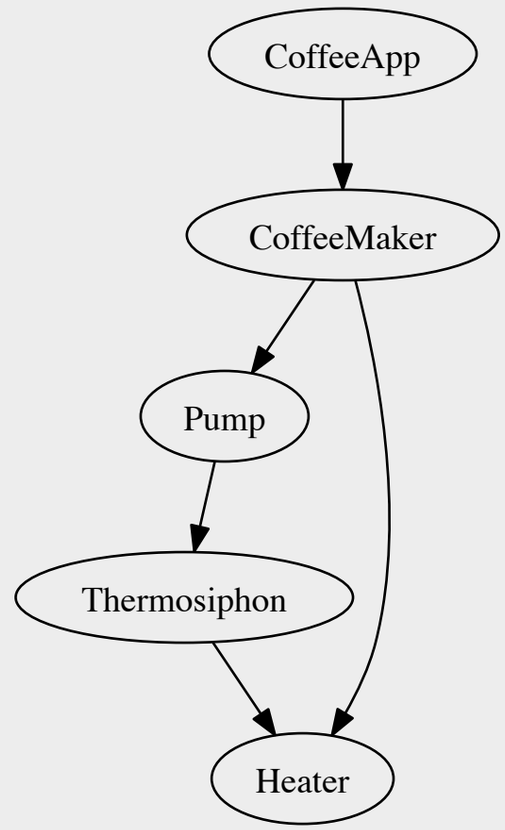
–Linus Torvalds

typicalprogrammer.com/?p=143

No black t-shirt necessary

(cc) creative commons from flickr.com/photos/mike_miley/5969110684/

- Know everything at build time.
- Easy to see how dependencies are used & satisfied



Motivation for Dagger

- It's like Guice, but with speed instead of features

also...

- Simple
- Predictable

Introduction

Motivation

Using Dagger

Inside Dagger

Wrapping Up

Dagger?

DAGger.

DAGger.

Directed Acyclic Graph

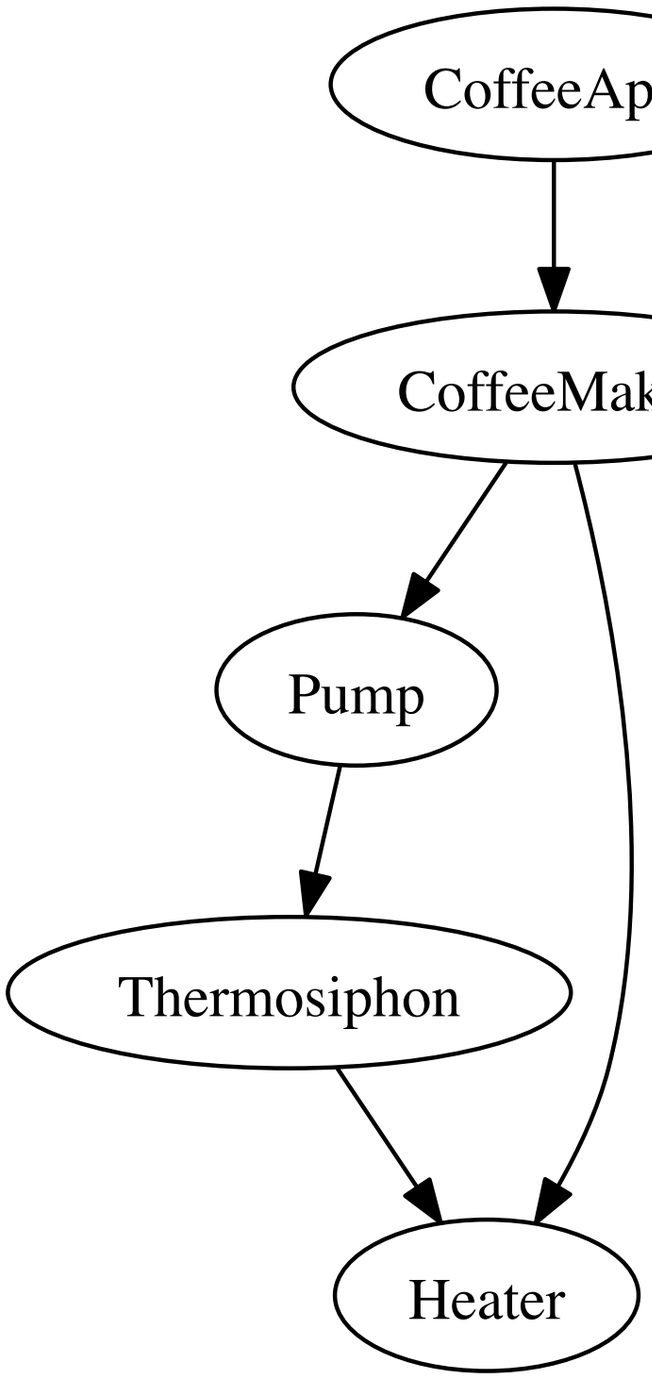
(cc) creative commons from flickr.com/photos/uscpsc/7894303566/

(cc) creative commons from flickr.com/photos/uscpsc/7894303566/

coffee maker

heater

thermosiphon pump



(cc) creative commons from flickr.com/photos/uscpsc/7894303566/

```
class Thermosiphon implements Pump {
  private final Heater heater;

  Thermosiphon(Heater heater) {
    this.heater = heater;
  }

    @Override public void pump() {
      if (heater.isHot()) {
        System.out.println("=> => pumping => =>");
      }
    }
  }
```

Declare Dependencies

```
class Thermosiphon implements Pump {
  private final Heater heater;
  @Inject
  Thermosiphon(Heater heater) {
    this.heater = heater;
  }

  ...
}
```

Declare Dependencies

```
class CoffeeMaker {
  @Inject Heater heater;
  @Inject Pump pump;

  ...
}
```

Satisfy Dependencies

```
@Module
class DripCoffeeModule {

  @Provides Heater provideHeater() {
    return new ElectricHeater();
  }

  @Provides Pump providePump(Thermosiphon pump) {
    return pump;
  }

}
```

Build the Graph

```
class CoffeeApp {

  public static void main(String[] args) {
    ObjectGraph objectGraph
        = ObjectGraph.create(new DripCoffeeModule());
    CoffeeMaker coffeeMaker
        = objectGraph.get(CoffeeMaker.class);
    coffeeMaker.brew();
  }

}
```

Neat features

- Lazy<T>
- Module overrides
- Multibindings

Lazy<T>

```
class GridingCoffeeMaker {
  @Inject Lazy<Grinder> lazyGrinder;

  public void brew() {
    while (needsGrinding()) {
      // Grinder created once and cached.
      Grinder grinder = lazyGrinder.get()
      grinder.grind();
    }
  }
}
```

Module Overrides

```
@Module(
    includes = DripCoffeeModule.class,
    entryPoints = CoffeeMakerTest.class,
    overrides = true
)
static class TestModule {
  @Provides @Singleton Heater provideHeater() {
    return Mockito.mock(Heater.class);
  }
}
```

Multibindings

```
@Module
class TwitterModule {
  @Provides(type=SET) SocialApi provideApi() {
    ...
  }
}

@Module
class GooglePlusModule {
  @Provides(type=SET) SocialApi provideApi() {
    ...
  }
}

...

@Inject Set<SocialApi>
```

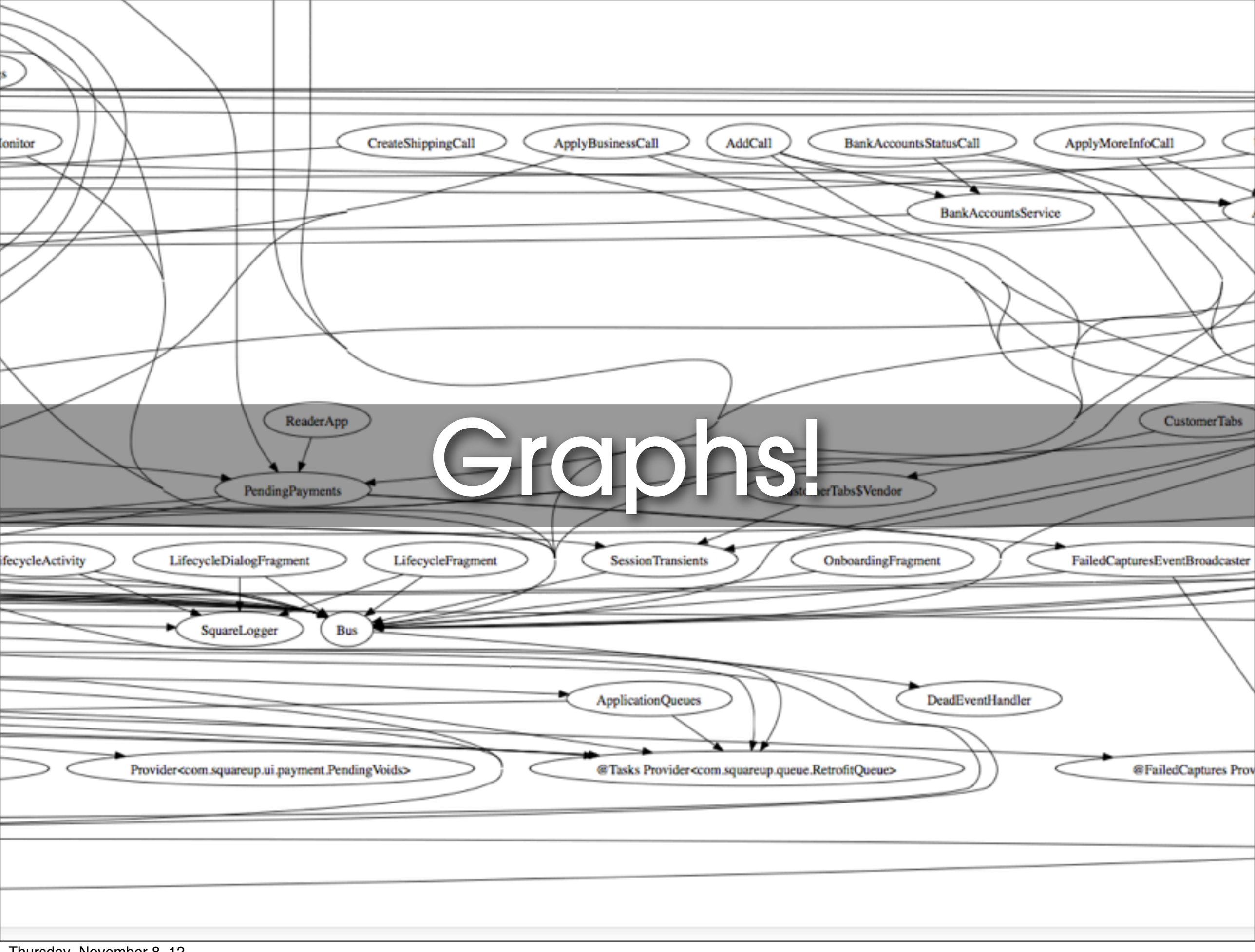
Introduction

Motivation

Using Dagger

Inside Dagger

Wrapping Up


CreateShippingCall
ApplyBusinessCall
AddCall
BankAccountsStatusCall
ApplyMoreInfoCall
BankAccountsService
ReaderApp
CustomerTabs
PendingPayments
LifecycleActivity
LifecycleDialogFragment
LifecycleFragment
SessionTransients
OnboardingFragment
FailedCapturesEventBroadcaster
SquareLogger
Bus
ApplicationQueues
DeadEventHandler
Provider<com.squareup.ui.payment.PendingVoids>
@Tasks Provider<com.squareup.queue.RetrofitQueue>

Graphs!

Creating Graphs

- Compile time
  - annotation processor
- Runtime
  - generated code loader
  - reflection

Using Graphs

- Injection
- Validation
- Graphviz!

But how?

- Bindings have names like
  `“com.squareup.geo.LocationMonitor”`

- Bindings know the names of their dependencies, like
  `“com.squareup.otto.Bus”`

```
final class CoffeeMaker$InjectAdapter extends Binding<CoffeeMaker> {

  private Binding<Heater> f0;
  private Binding<Pump> f1;

  public CoffeeMaker$InjectAdapter() {
    super("coffee.CoffeeMaker", ...);
  }

  public void attach(Linker linker) {
    f0 = linker.requestBinding("coffee.Heater", coffee.CoffeeMaker.class);
    f1 = linker.requestBinding("coffee.Pump", coffee.CoffeeMaker.class);
  }

  public CoffeeMaker get() {
    coffee.CoffeeMaker result = new coffee.CoffeeMaker();
    injectMembers(result);
    return result;
  }

  public void injectMembers(CoffeeMaker object) {
    object.heater = f0.get();
    object.pump = f1.get();
  }

  public void getDependencies(Set<Binding<?>> bindings) {
    bindings.add(f0);
    bindings.add(f1);
  }
}
```

Validation

- Eager at build time
- Lazy at runtime

dagger-compiler

(cc) creative commons from flickr.com/photos/discover-central-california/8010906617

javax.annotation.processing

Built into `javac`

Foolishly easy to use. Just put `dagger-compiler` on your classpath!

It's a hassle (for us)

- Coping with prebuilt .jar files
- Versioning
- Testing

... and it's limited (for you)

- No private or final field access
- Incremental builds are imperfect
- ProGuard

... but it's smoking fast! (for your users)

- Startup time for Square Wallet on one device improved from ~5 seconds to ~2 seconds

Different Platforms are Different

- HotSpot
- Android
- GWT

HotSpot: Java on the Server

- The JVM is fast
- Code bases are huge

Android

- Battery powered
- Garbage collection causes jank
- Slow reflection, especially on older devices
- Managed lifecycle

GWT

- Code size really matters
- Compiler does full-app optimizations
- No reflection.

github.com/tbroyer/sheath

API Design in the GitHub age

- Forking makes it easy to stay small & focused

javax.inject

```
public @interface Inject {}

public @interface Named {
  String value() default "";
}

public interface Provider {
  T get();
}

public @interface Qualifier {}

public @interface Scope {}

public @interface Singleton {}
```

dagger

```
public interface Lazy<T> {
  T get();
}

public interface MembersInjector<T> {
  void injectMembers(T instance);
}

public final class ObjectGraph {
  public static ObjectGraph create(Object... modules);
  public ObjectGraph plus(Object... modules);
  public void validate();
  public void injectStatics();
  public <T> T get(Class type);
  public <T> T inject(T instance);
}
```

```
public @interface Module {
  Class[] entryPoints() default { };
  Class[] staticInjections() default { };
  Class[] includes() default { };
  Class addsTo() default Void.class;
  boolean overrides() default false;
  boolean complete() default true;
}

public @interface Provides {
  enum Type { UNIQUE, SET }
  Type type() default Type.UNIQUE;
}
```

Introduction

Motivation

Using Dagger

Inside Dagger

Wrapping Up

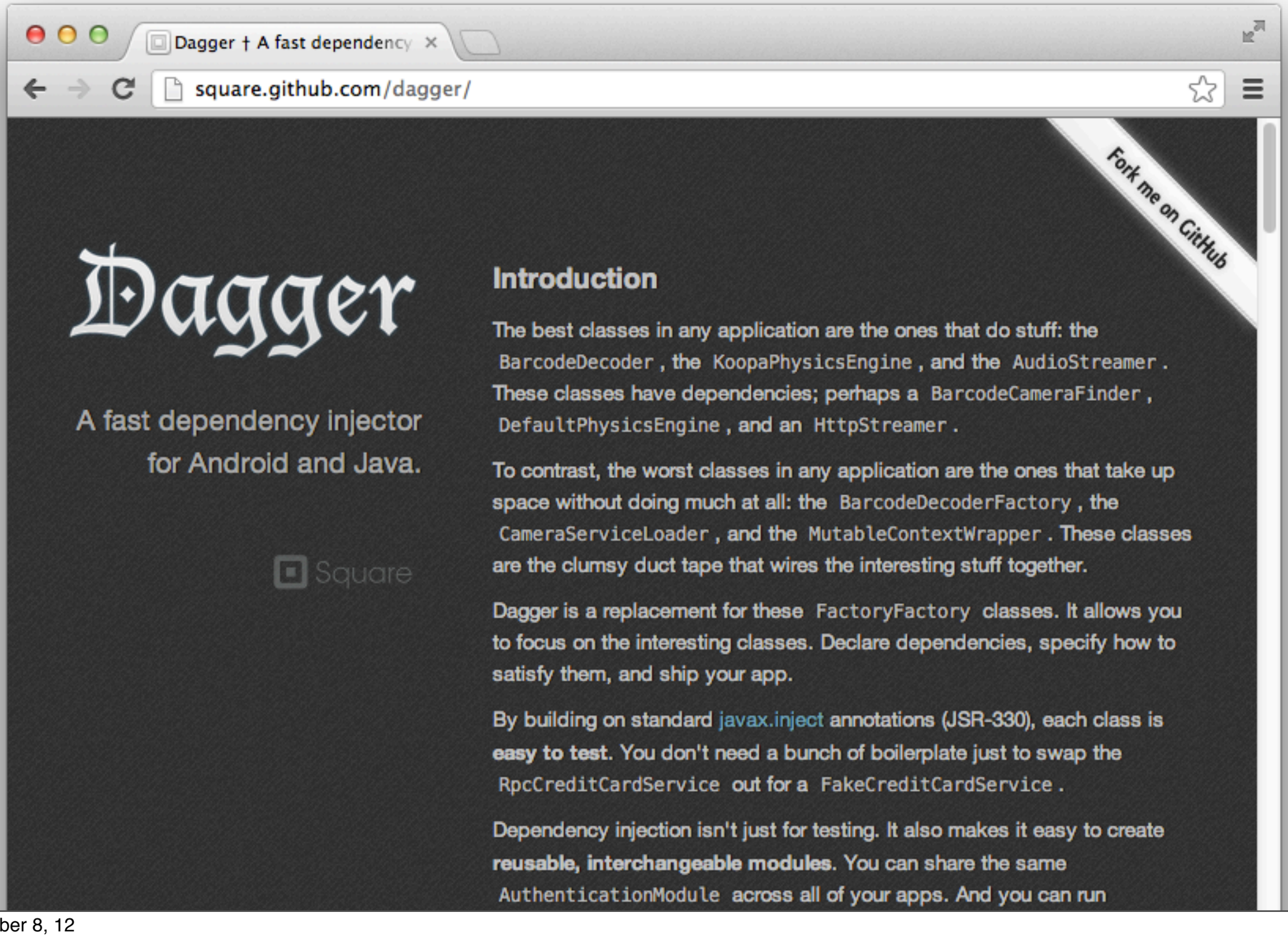
Guice

- Still the best choice for many apps
- May soon be able to mix & match with Dagger

Dagger

- Dagger 0.9 today!
- Dagger 1.0 in 2012ish
- Friendly open source.

square.github.com/dagger



Questions?

squareup.com/careers
corner.squareup.com

swank.ca
jwilson@squareup.com