dependency injection with

Dagger 2

Jake Wharton

Dependency Injection

flickr.com/photos/sunsetnoir/6984108500

Dependency Injection

- No “new”, dependencies come to you

flickr.com/photos/sunsetnoir/6984108500

Dependency Injection

- First and foremost a pattern

Dependency Injection

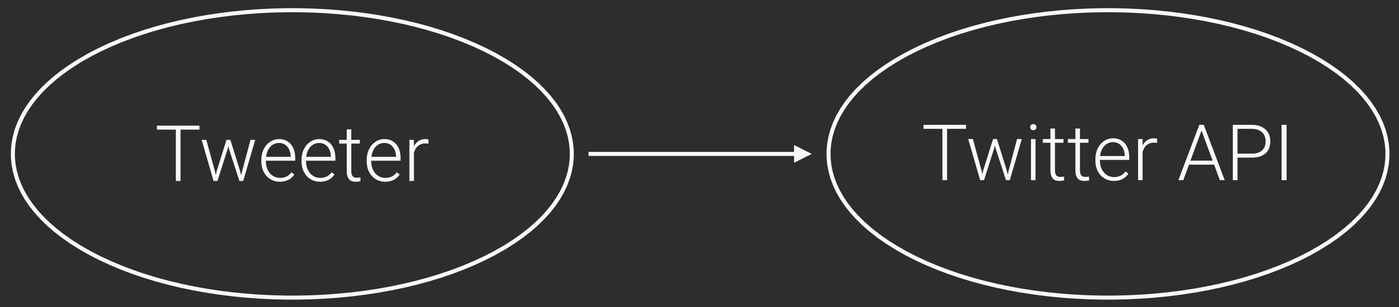
- First and foremost a pattern
- Every single app has some form of dependency injection

Dependency Injection

- First and foremost a pattern
- Every single app has some form of dependency injection

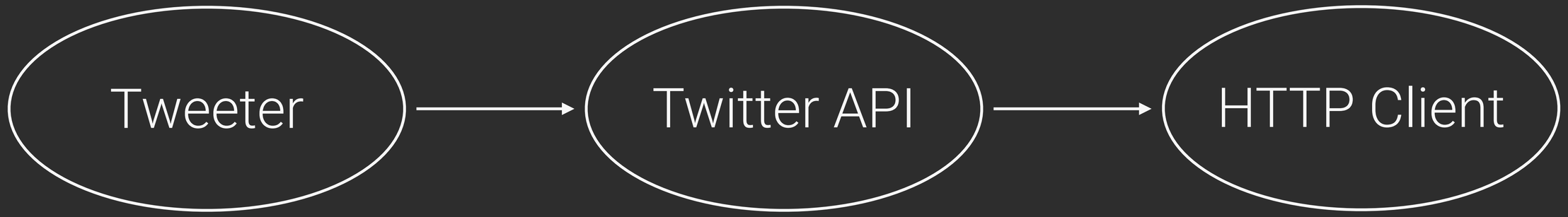
Dependency Injection

- First and foremost a pattern
- Every single app has some form of dependency injection



Dependency Injection

- First and foremost a pattern
- Every single app has some form of dependency injection



```
public class Tweeter {
  public void tweet(String tweet) {
    TwitterApi api = new TwitterApi();
    api.postTweet("JakeWharton", tweet);
  }
}
```

```
public class Tweeter {
  public void tweet(String tweet) {
    TwitterApi api = new TwitterApi();
    api.postTweet("JakeWharton", tweet);
  }
}


public class TwitterApi {
  public void postTweet(String user, String tweet) {
    OkHttpClient client = new OkHttpClient();
    Request request = // TODO build POST request...
    client.newCall(request).execute();
  }
}
```

```
public class Tweeter {
  public void tweet(String tweet) {
    TwitterApi api = new TwitterApi();
    api.postTweet("JakeWharton", tweet);
  }
}


public class TwitterApi {
  public void postTweet(String user, String tweet) {
    OkHttpClient client = new OkHttpClient();
    Request request = // TODO build POST request...
    client.newCall(request).execute();
  }
}


Tweeter tweeter = new Tweeter();
tweeter.tweet("Hello, #Devoxx 2014!");
```

```
public class TwitterApi {
  public void postTweet(String user, String tweet) {
    OkHttpClient client = new OkHttpClient();
    Request request = // TODO build POST request...
    client.newCall(request).execute();
  }
}
```

```
public class TwitterApi {
  private final OkHttpClient client = new OkHttpClient();

  public void postTweet(String user, String tweet) {
    Request request = // TODO build POST request...
    client.newCall(request).execute();
  }
}
```

```
public class TwitterApi {
  private final OkHttpClient client = new OkHttpClient();

  public void postTweet(String user, String tweet) {
    Request request = // TODO build POST request...
    client.newCall(request).execute();
  }
}
```

```
public class TwitterApi {
  private final OkHttpClient client;

  public TwitterApi(OkHttpClient client) {
    this.client = client;
  }

  public void postTweet(String user, String tweet) {
    Request request = // TODO build POST request...
    client.newCall(request).execute();
  }
}
```

```
public class TwitterApi {
  private final OkHttpClient client;

  public TwitterApi(OkHttpClient client) {
    this.client = client;
  }

  public void postTweet(String user, String tweet) {
    Request request = // TODO build POST request...
    client.newCall(request).execute();
  }
}
```

```
public class Tweeter {
  public void tweet(String tweet) {
    TwitterApi api = new TwitterApi();
    api.postTweet("JakeWharton", tweet);
  }
}
```

```
public class Tweeter {
  public void tweet(String tweet) {
    TwitterApi api = new TwitterApi(new OkHttpClient());
    api.postTweet("JakeWharton", tweet);
  }
}
```

```
public class Tweeter {
  public void tweet(String tweet) {
    TwitterApi api = new TwitterApi(new OkHttpClient());
    api.postTweet("JakeWharton", tweet);
  }
}
```

```
public class Tweeter {
  private final TwitterApi api = new TwitterApi(new OkHttpClient());

  public void tweet(String tweet) {
    api.postTweet("JakeWharton", tweet);
  }
}
```

```
public class Tweeter {
  private final TwitterApi api = new TwitterApi(new OkHttpClient());

  public void tweet(String tweet) {
    api.postTweet("JakeWharton", tweet);
  }
}
```

```
public class Tweeter {
  private final TwitterApi api = new TwitterApi(new OkHttpClient());
  private final String user;

  public Tweeter(String user) {
    this.user = user;
  }

  public void tweet(String tweet) {
    api.postTweet(user, tweet);
  }
}
```

```
public class Tweeter {
  private final TwitterApi api = new TwitterApi(new OkHttpClient());
  private final String user;

  public Tweeter(String user) {
    this.user = user;
  }

  public void tweet(String tweet) {
    api.postTweet(user, tweet);
  }
}
```

```
Tweeter tweeter = new Tweeter();
tweeter.tweet("Hello, #Devoxx 2014!");
```

```
Tweeter tweeter = new Tweeter("JakeWharton");
tweeter.tweet("Hello, #Devoxx 2014!");
```

```
Tweeter tweeter = new Tweeter("JakeWharton");
tweeter.tweet("Hello, #Devoxx 2014!");
```

```
Tweeter tweeter = new Tweeter("JakeWharton");
tweeter.tweet("Hello, #Devoxx 2014!");
tweeter.tweet("#Hungover #Dagger");
tweeter.tweet("Waffles, beer, diamonds, chocolate, and Dagger!");
tweeter.tweet("Java Posse ain’t got nothing on #Dagger");
tweeter.tweet("Are these the same slides from last year? #meta");
tweeter.tweet("I need an injection of beer, amirite? #Dagger");
```

```
Tweeter tweeter = new Tweeter("JakeWharton");
tweeter.tweet("Hello, #Devoxx 2014!");
tweeter.tweet("#Hungover #Dagger");
tweeter.tweet("Waffles, beer, diamonds, chocolate, and Dagger!");
tweeter.tweet("Java Posse ain’t got nothing on #Dagger");
tweeter.tweet("Are these the same slides from last year? #meta");
tweeter.tweet("I need an injection of beer, amirite? #Dagger");
```

```
Timeline timeline = new Timeline("JakeWharton");
timeline.loadMore(20);
for (Tweet tweet : timeline.get()) {
  System.out.println(tweet);
}
```

```
public class Timeline {
  private final List<Tweet> cache = new ArrayList<>();
  private final TwitterApi api = new TwitterApi(new OkHttpClient());
  private final String user;

  public Timeline(String user) {
    this.user = user;
  }

  public List<Tweet> get() { /* ... */ }
  public void loadMore(int amount) { /* ... */ }
}
```

```
public class Timeline {
  private final List<Tweet> cache = new ArrayList<>();
  private final TwitterApi api;
  private final String user;

  public Timeline(TwitterApi api, String user) {
    this.api = api;
    this.user = user;
  }

  public List<Tweet> get() { /* ... */ }
  public void loadMore(int amount) { /* ... */ }
}
```

```
public class Timeline {
  private final List<Tweet> cache = new ArrayList<>();
  private final TwitterApi api;
  private final String user;

  public Timeline(TwitterApi api, String user) {
    this.api = api;
    this.user = user;
  }

  public List<Tweet> get() { /* ... */ }
  public void loadMore(int amount) { /* ... */ }
}
```

```
public class Tweeter {
  private final TwitterApi api = new TwitterApi(new OkHttpClient());
  private final String user;

  public Tweeter(String user) {
    this.user = user;
  }

  public void tweet(String tweet) {
    api.postTweet(user, tweet);
  }
}
```

```
public class Tweeter {
  private final TwitterApi api;
  private final String user;

  public Tweeter(TwitterApi api, String user) {
    this.api = api;
    this.user = user;
  }

  public void tweet(String tweet) {
    api.postTweet(user, tweet);
  }
}
```

```
public class Tweeter {
  private final TwitterApi api;
  private final String user;

  public Tweeter(TwitterApi api, String user) {
    this.api = api;
    this.user = user;
  }

  public void tweet(String tweet) {
    api.postTweet(user, tweet);
  }
}
```

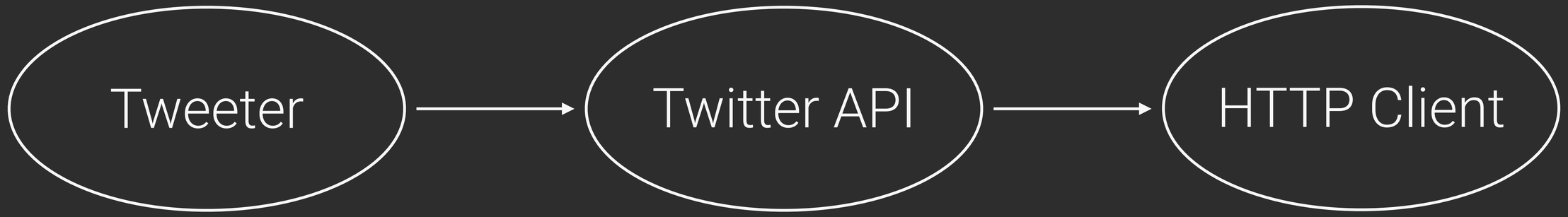
```
Tweeter tweeter = new Tweeter("JakeWharton");
tweeter.tweet("Hello, #Devoxx 2014!");

Timeline timeline = new Timeline("JakeWharton");
timeline.loadMore(20);
for (Tweet tweet : timeline.get()) {
  System.out.println(tweet);
}
```

```
OkHttpClient client = new OkHttpClient();
TwitterApi api = new TwitterApi(client);
String user = "Jake Wharton";

Tweeter tweeter = new Tweeter(api, user);
tweeter.tweet("Hello, #Devoxx 2014!");

Timeline timeline = new Timeline(api, user);
timeline.loadMore(20);
for (Tweet tweet : timeline.get()) {
  System.out.println(tweet);
}
```

```
OkHttpClient client = new OkHttpClient();
TwitterApi api = new TwitterApi(client);
String user = "Jake Wharton";

Tweeter tweeter = new Tweeter(api, user);
tweeter.tweet("Hello, #Devoxx 2014!");

Timeline timeline = new Timeline(api, user);
timeline.loadMore(20);
for (Tweet tweet : timeline.get()) {
  System.out.println(tweet);
}
```

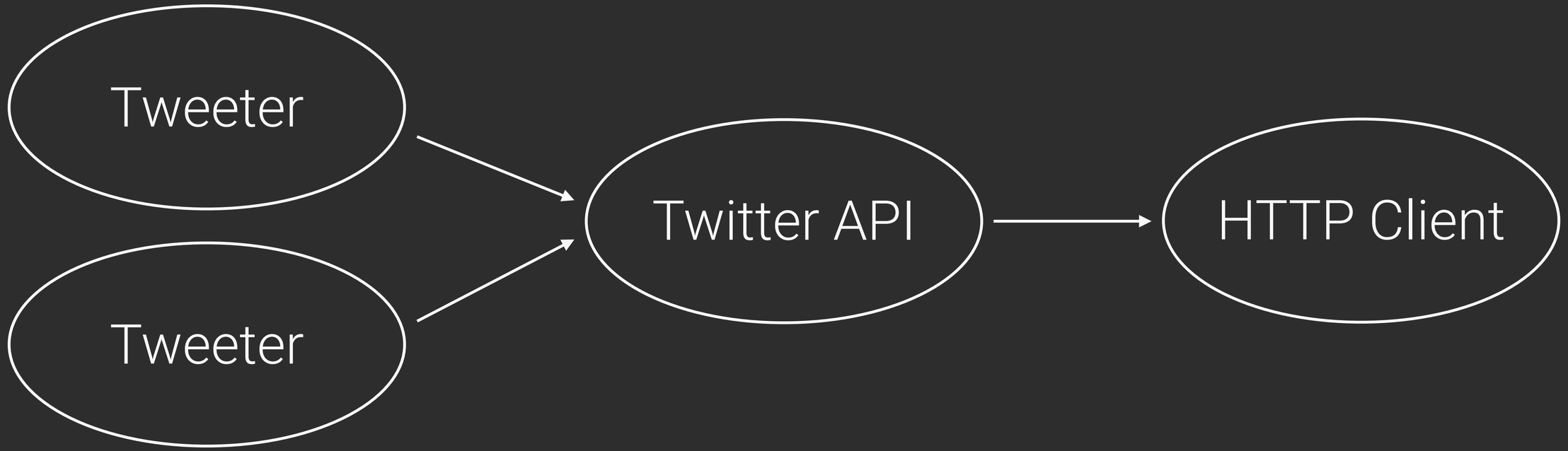
Dependency Injection

- First and foremost a pattern
- Every single app has some form of dependency injection



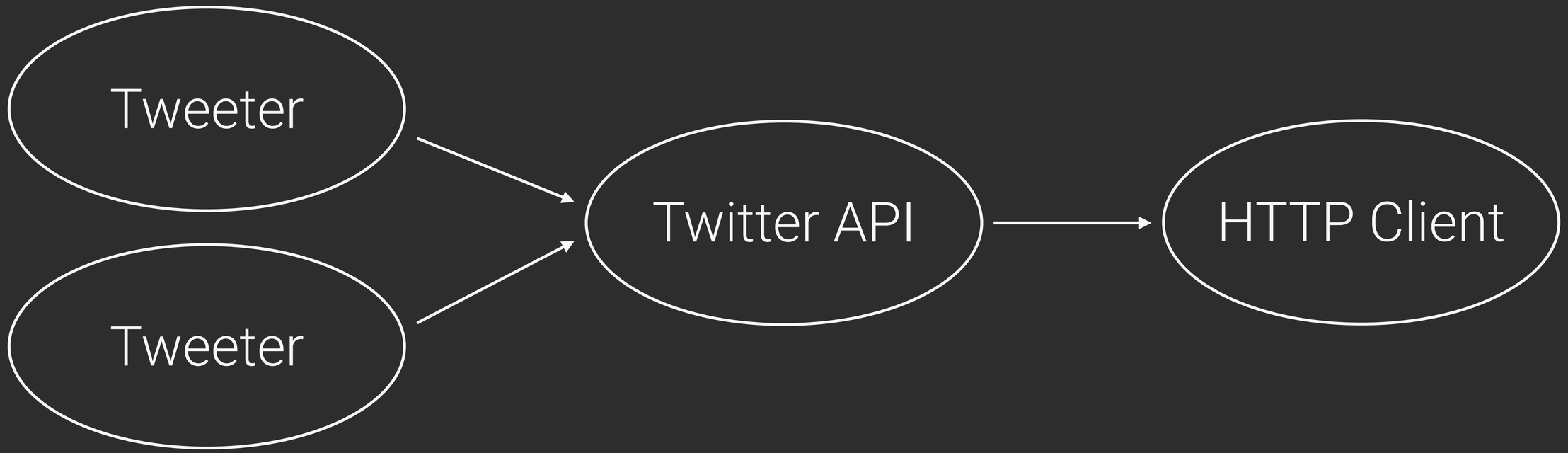
Dependency Injection

- First and foremost a pattern
- Every single app has some form of dependency injection



Dependency Injection

- How do we avoid the boilerplate that comes with the pattern?



Library Rescue

- Spring
- Guice
- Dagger (v1)
- PicoContainer
- CDI
- Others…

Guice

Dagger

(v1)

Guice

Guice

- Developed at Google by Bob Lee, later Jesse Wilson, others.

Guice

- Developed at Google by Bob Lee, later Jesse Wilson, others.
- Adopted and maintained by Java Core Libraries team.

Guice

- Developed at Google by Bob Lee, later Jesse Wilson, others.
- Adopted and maintained by Java Core Libraries team.
- Powerful, dynamic, well-tested, wide-spread, etc…

Guice

- Developed at Google by Bob Lee, later Jesse Wilson, others.
- Adopted and maintained by Java Core Libraries team.
- Powerful, dynamic, well-tested, wide-spread, etc…
- Configuration problems occur at runtime.

Guice

- Developed at Google by Bob Lee, later Jesse Wilson, others.
- Adopted and maintained by Java Core Libraries team.
- Powerful, dynamic, well-tested, wide-spread, etc…
- Configuration problems occur at runtime.
- Slow initialization, slow injection, memory concerns.

Dagger (v1)

Dagger (v1)

- Developed at Square by Jesse Wilson advised by Bob Lee.

Dagger (v1)

- Developed at Square by Jesse Wilson advised by Bob Lee.
- Initially targeted at highly resource constrained environments.

Dagger (v1)

- Developed at Square by Jesse Wilson advised by Bob Lee.
- Initially targeted at highly resource constrained environments.
- Static analysis of all dependencies and injection points.

Dagger (v1)

- Developed at Square by Jesse Wilson advised by Bob Lee.
- Initially targeted at highly resource constrained environments.
- Static analysis of all dependencies and injection points.
- Fail as early as possible (compile-time, not runtime)

Dagger (v1)

- Developed at Square by Jesse Wilson advised by Bob Lee.
- Initially targeted at highly resource constrained environments.
- Static analysis of all dependencies and injection points.
- Fail as early as possible (compile-time, not runtime)
- Eliminate reflection on methods, fields, and annotations.

Dagger (v1)

Dagger (v1)

- Wide-spread, successful use in large Android applications.

Dagger (v1)

- Wide-spread, successful use in large Android applications.
- Triggers over-eager class loading on graph creation.

Dagger (v1)

- Wide-spread, successful use in large Android applications.
- Triggers over-eager class loading on graph creation.
- FQCN string-based keys in map-like data structure.

Dagger (v1)

- Wide-spread, successful use in large Android applications.
- Triggers over-eager class loading on graph creation.
- FQCN string-based keys in map-like data structure.
- Lack of full static scope analysis and scope annotations.

Dagger (v1)

- Wide-spread, successful use in large Android applications.
- Triggers over-eager class loading on graph creation.
- FQCN string-based keys in map-like data structure.
- Lack of full static scope analysis and scope annotations.
- Still uses reflection to load generated classes.

Dagger (v2)

Dagger (v2)

- Proposed and implemented by Java Core Libraries team.

Dagger (v2)

- Proposed and implemented by Java Core Libraries team.
- Eliminate runtime library and generated code overhead.

Dagger (v2)

- Proposed and implemented by Java Core Libraries team.
- Eliminate runtime library and generated code overhead.
- Shift remaining runtime analysis to compile time.

Dagger (v2)

- Proposed and implemented by Java Core Libraries team.
- Eliminate runtime library and generated code overhead.
- Shift remaining runtime analysis to compile time.
- Scoping with annotations and associated static analysis.

Dagger API

Dagger API

- `@Module` + `@Provides`: mechanism for providing dependencies.

Dagger API

- `@Module` + `@Provides`: mechanism for providing dependencies.
- `@Inject`: mechanism for requesting dependencies.

Dagger API

- `@Module` + `@Provides`: mechanism for providing dependencies.
- `@Inject`: mechanism for requesting dependencies.
- `@Component`: bridge between modules and injections

Dagger API

- `@Module` + `@Provides`: mechanism for providing dependencies.
- `@Inject`: mechanism for requesting dependencies.
- `@Component`: bridge between modules and injections
- Plus some other sugar, magic, and conventions.

Providing Dependencies

Providing Dependencies

- Modules are classes whose methods provide dependencies.

Providing Dependencies

- Modules are classes whose methods provide dependencies.
- `@Module` on the class.

Providing Dependencies

- Modules are classes whose methods provide dependencies.
- `@Module` on the class.
- `@Provides` (and friends) on each method.

```
public class NetworkModule {

  OkHttpClient provideOkHttpClient() {
    return new OkHttpClient();
  }


  TwitterApi provideTwitterApi(OkHttpClient client) {
    return new TwitterApi(client);
  }
}
```

```java
@Module
public class NetworkModule {

  OkHttpClient provideOkHttpClient() {
    return new OkHttpClient();
  }


  TwitterApi provideTwitterApi(OkHttpClient client) {
    return new TwitterApi(client);
  }
}
```

```java
@Module
public class NetworkModule {
  @Provides
  OkHttpClient provideOkHttpClient() {
    return new OkHttpClient();
  }

  @Provides
  TwitterApi provideTwitterApi(OkHttpClient client) {
    return new TwitterApi(client);
  }
}
```

```java
@Module
public class NetworkModule {
  @Provides @Singleton
  OkHttpClient provideOkHttpClient() {
    return new OkHttpClient();
  }

  @Provides @Singleton
  TwitterApi provideTwitterApi(OkHttpClient client) {
    return new TwitterApi(client);
  }
}
```

Providing Dependencies

Providing Dependencies

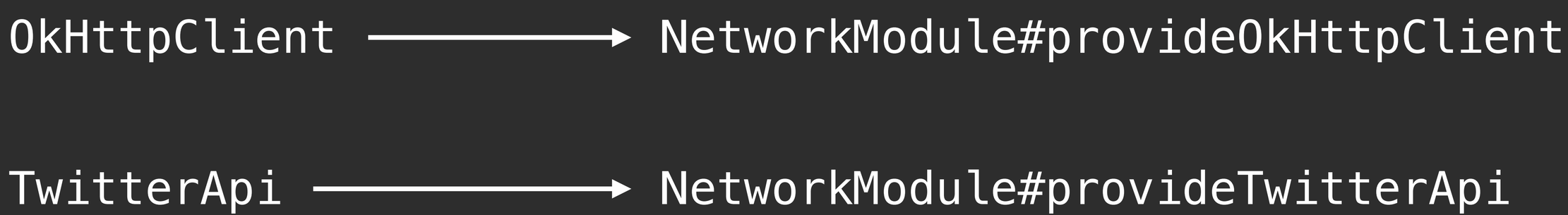
`NetworkModule#provideOkHttpClient`

`NetworkModule#provideTwitterApi`

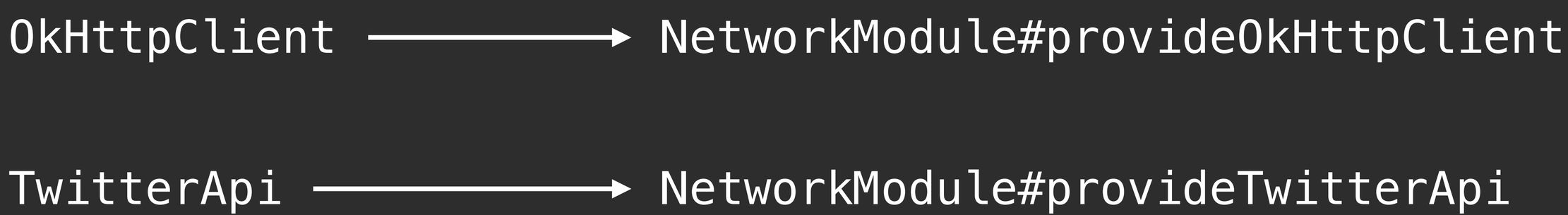
Providing Dependencies

`OkHttpClient`	`NetworkModule#provideOkHttpClient`
`TwitterApi`	`NetworkModule#provideTwitterApi`

Providing Dependencies

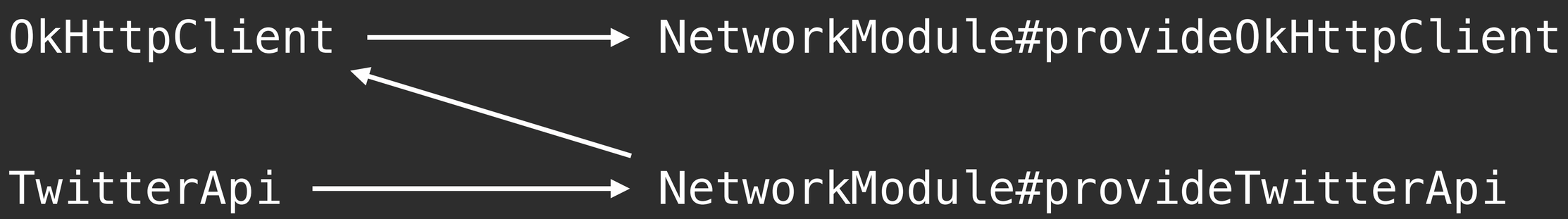


Providing Dependencies




```
TwitterApi provideTwitterApi(OkHttpClient client) {
  return new TwitterApi(client);
}
```

Providing Dependencies



Providing Dependencies

- Modules are classes whose methods provide dependencies.
- `@Module` on the class.
- `@Provides` (and friends) on each method.

Providing Dependencies

- Modules are classes whose methods provide dependencies.
- `@Module` on the class.
- `@Provides` (and friends) on each method.
- Designed to be partitioned and composed together.

```
public class TwitterModule {









}
```

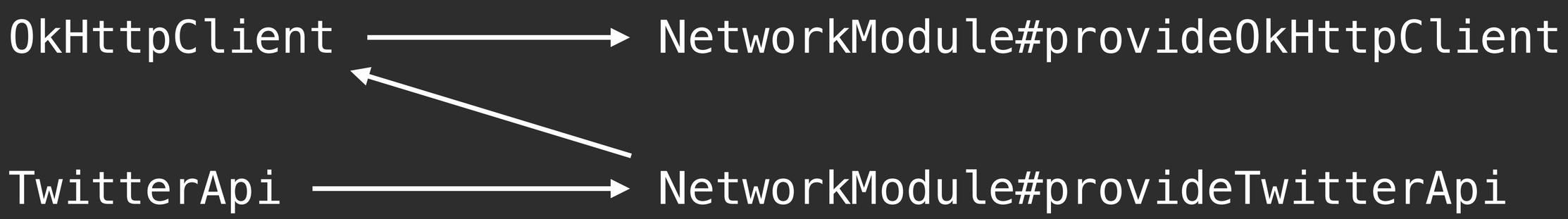
```
@Module
public class TwitterModule {









}
```

```java
@Module
public class TwitterModule {
  private final String user;

  public TwitterModule(String user) {
    this.user = user;
  }









}
```

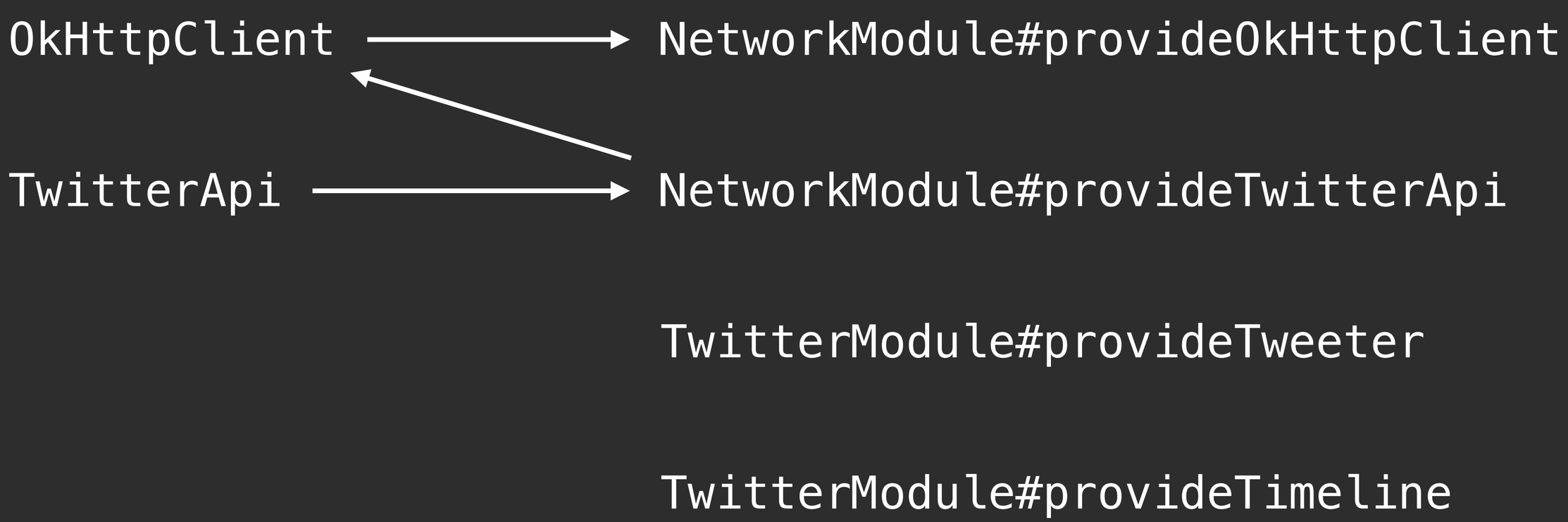
```java
@Module
public class TwitterModule {
  private final String user;

  public TwitterModule(String user) {
    this.user = user;
  }

  @Provides @Singleton
  Tweeter provideTweeter(TwitterApi api) {
    return new Tweeter(api, user);
  }



}
```

```java
@Module
public class TwitterModule {
  private final String user;

  public TwitterModule(String user) {
    this.user = user;
  }

  @Provides @Singleton
  Tweeter provideTweeter(TwitterApi api) {
    return new Tweeter(api, user);
  }

  @Provides @Singleton
  Timeline provideTimeline(TwitterApi api) {
    return new Timeline(api, user);
  }
}
```

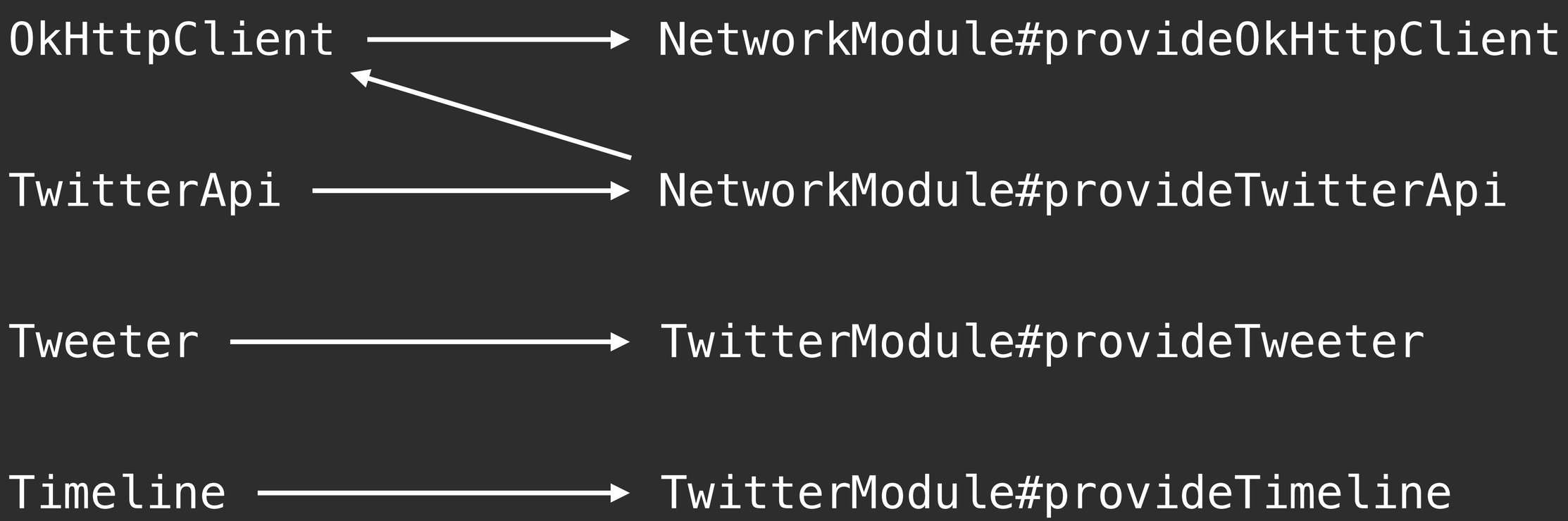
Providing Dependencies



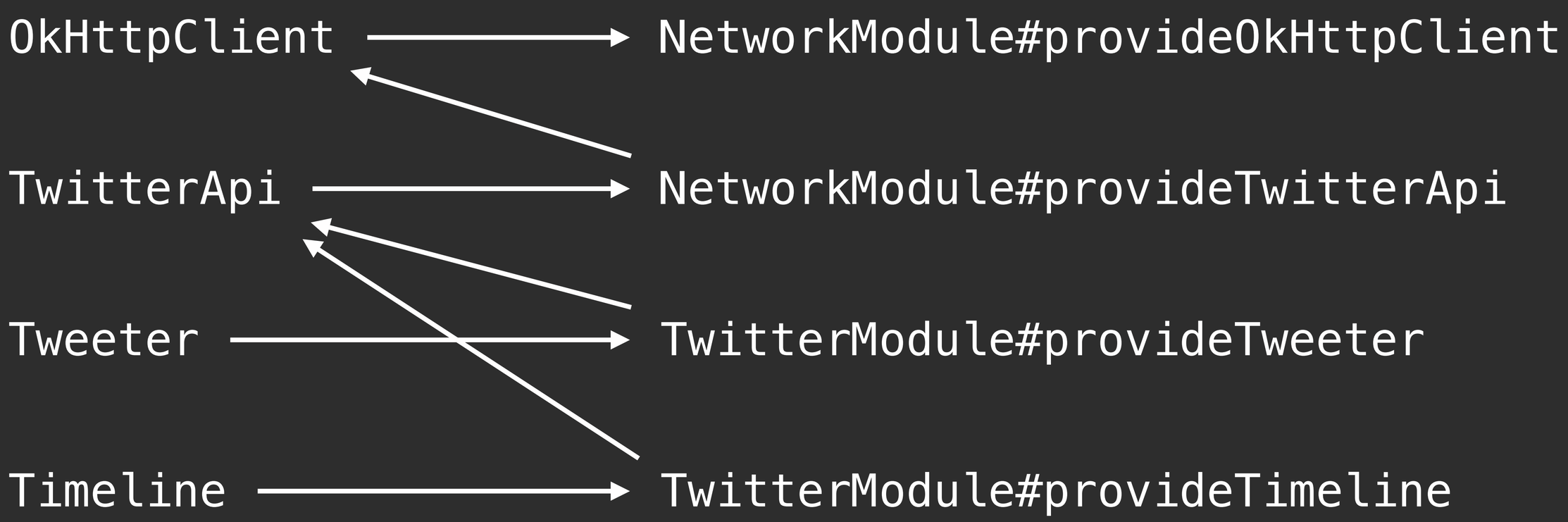
Providing Dependencies



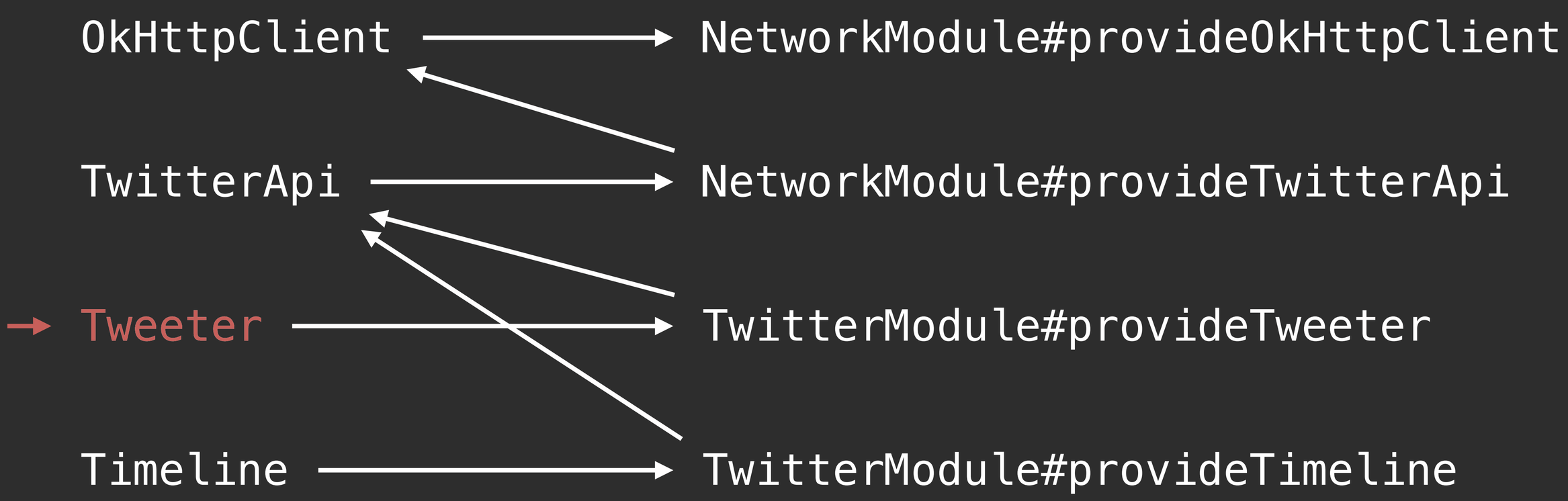
Providing Dependencies



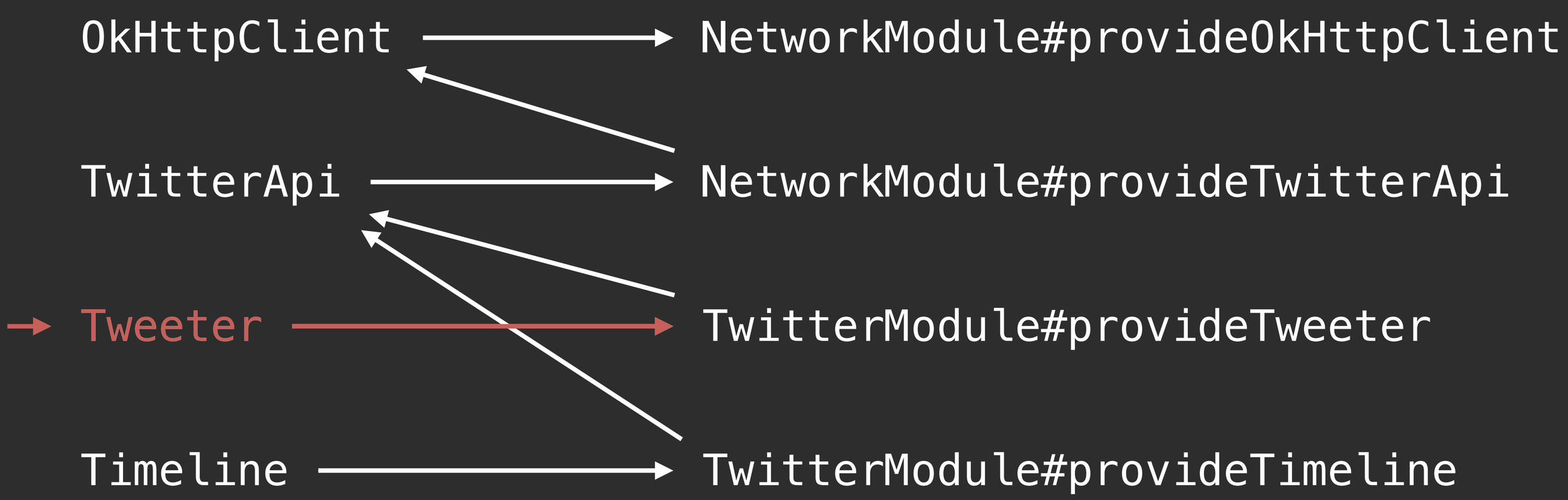
Providing Dependencies



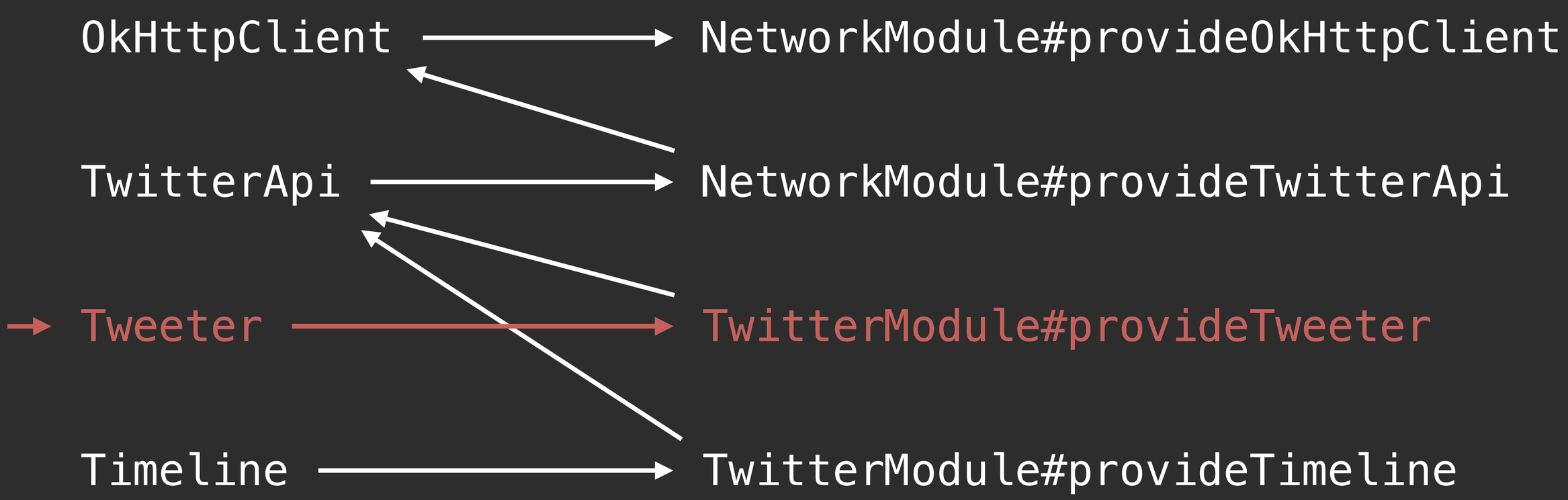
Providing Dependencies



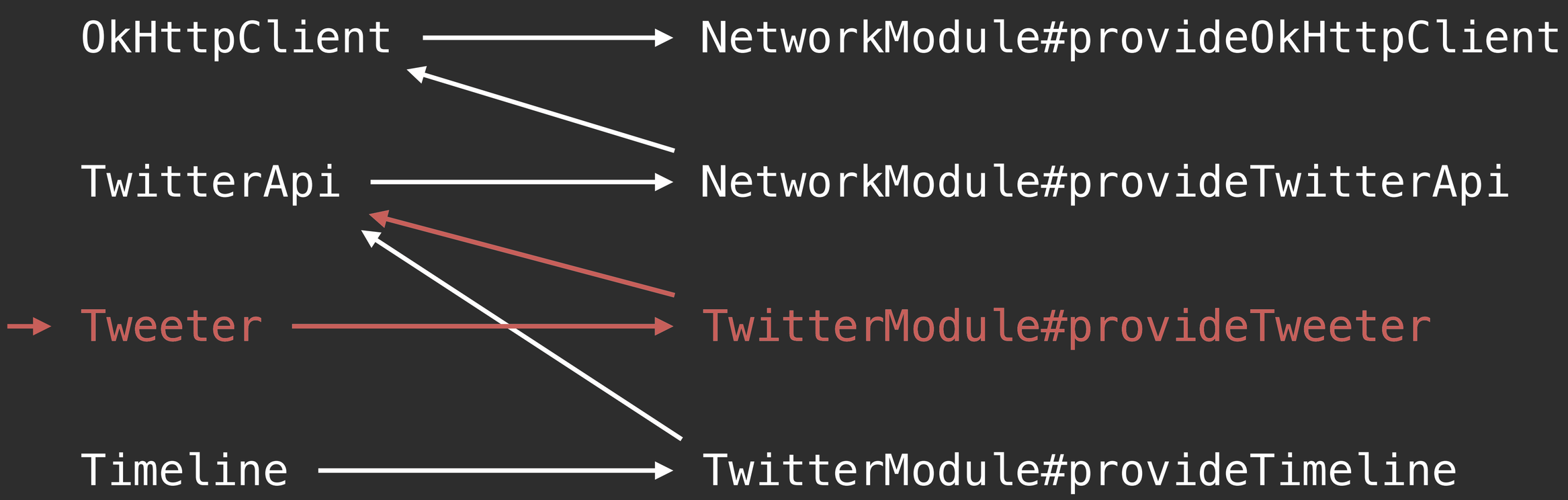
Providing Dependencies



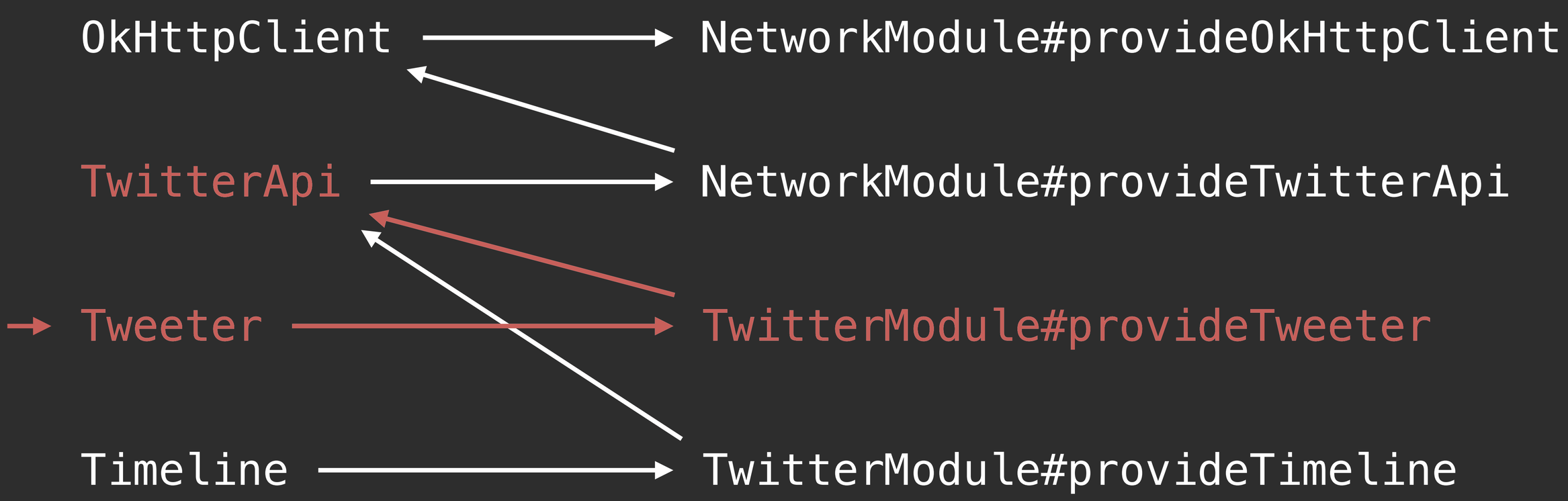
Providing Dependencies



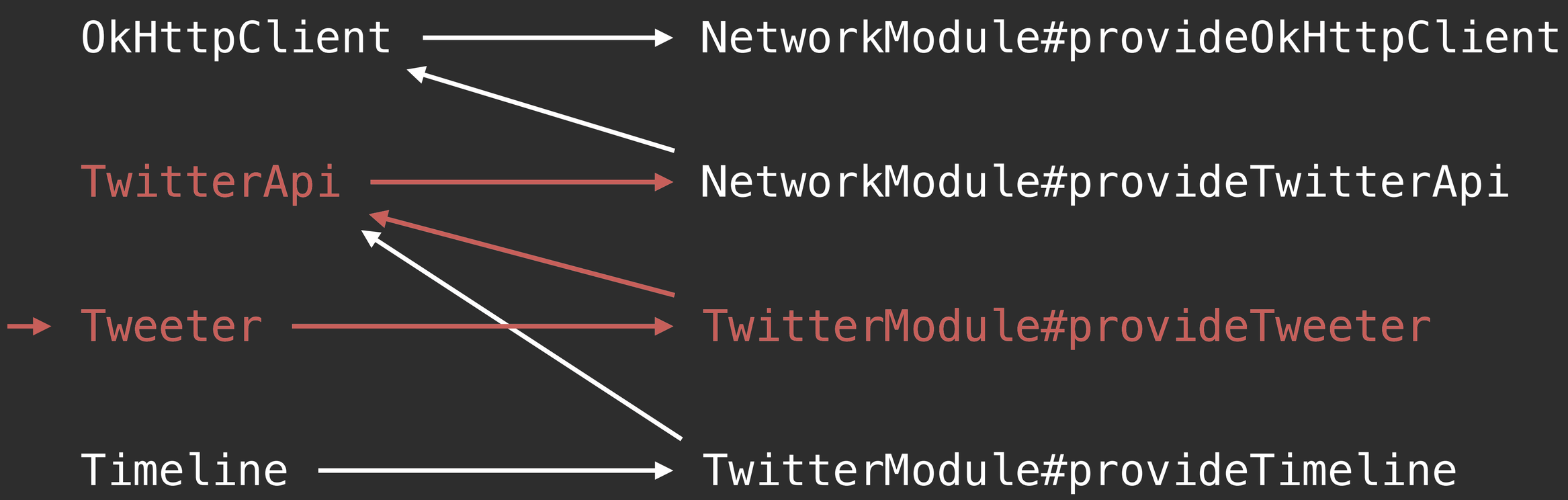
Providing Dependencies



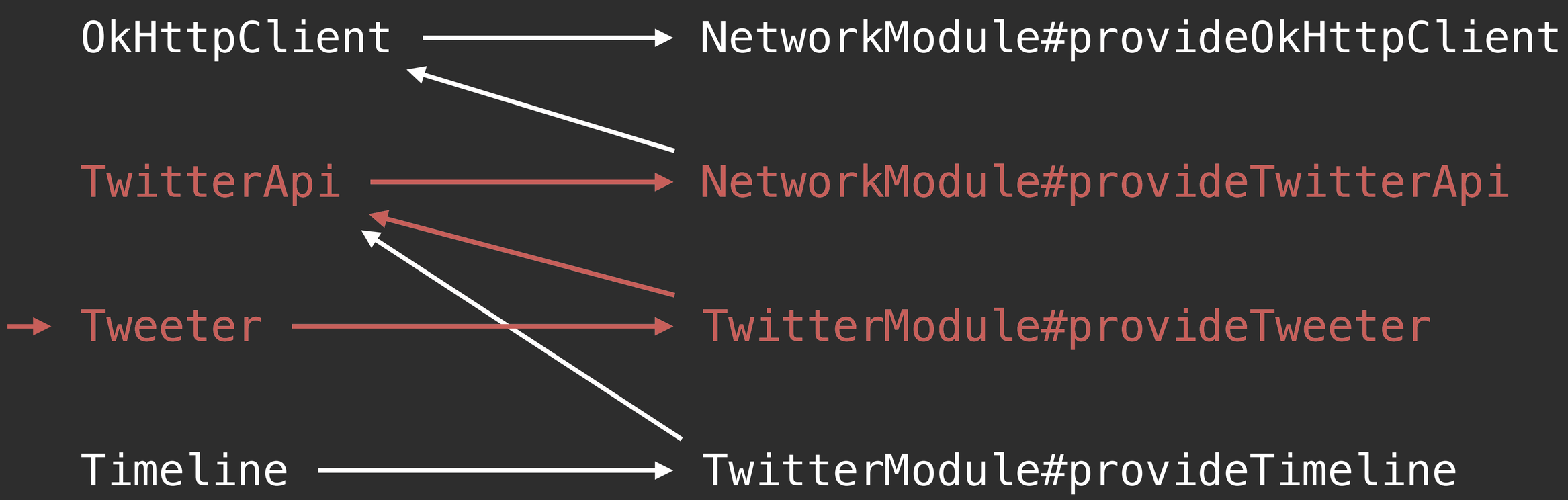
Providing Dependencies



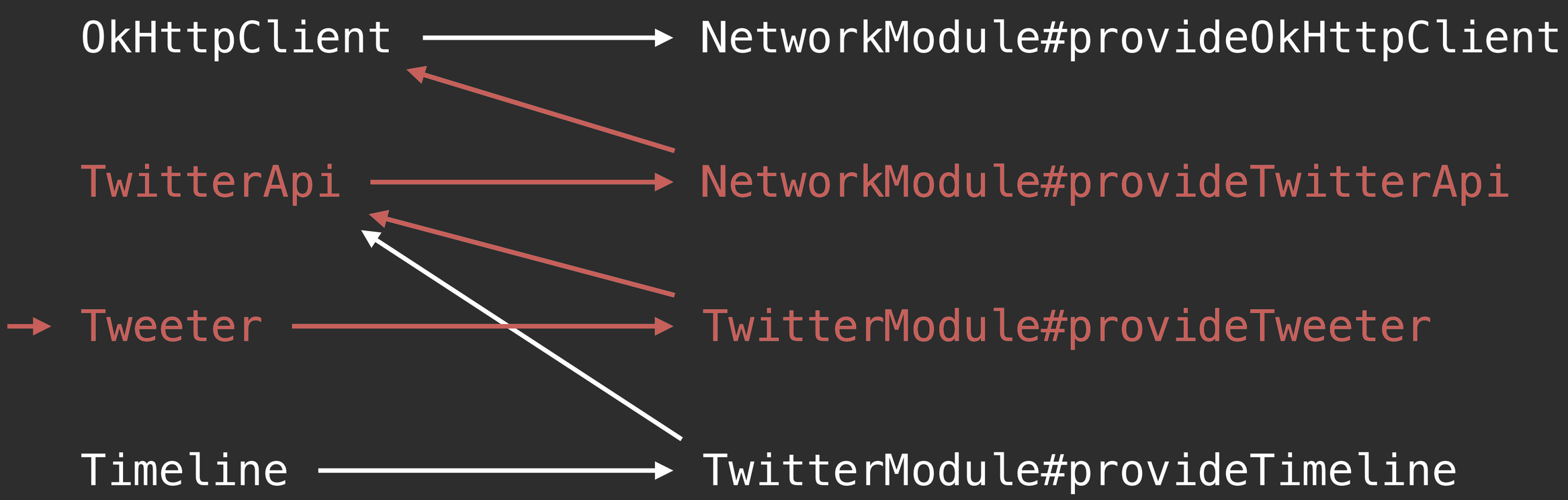
Providing Dependencies



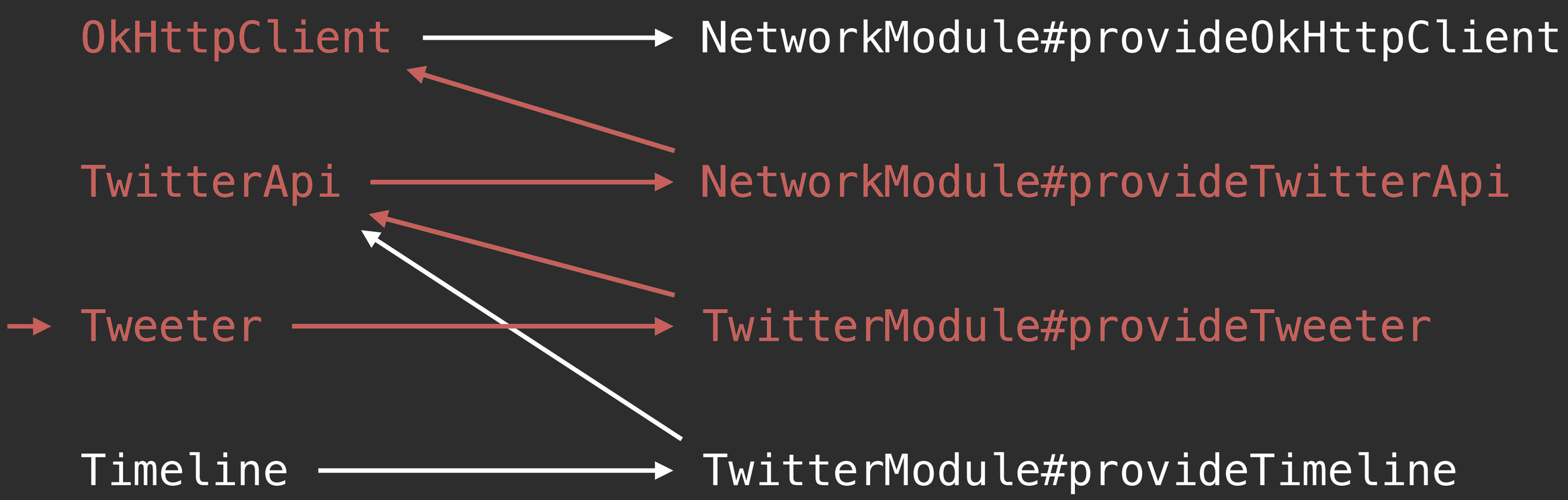
Providing Dependencies



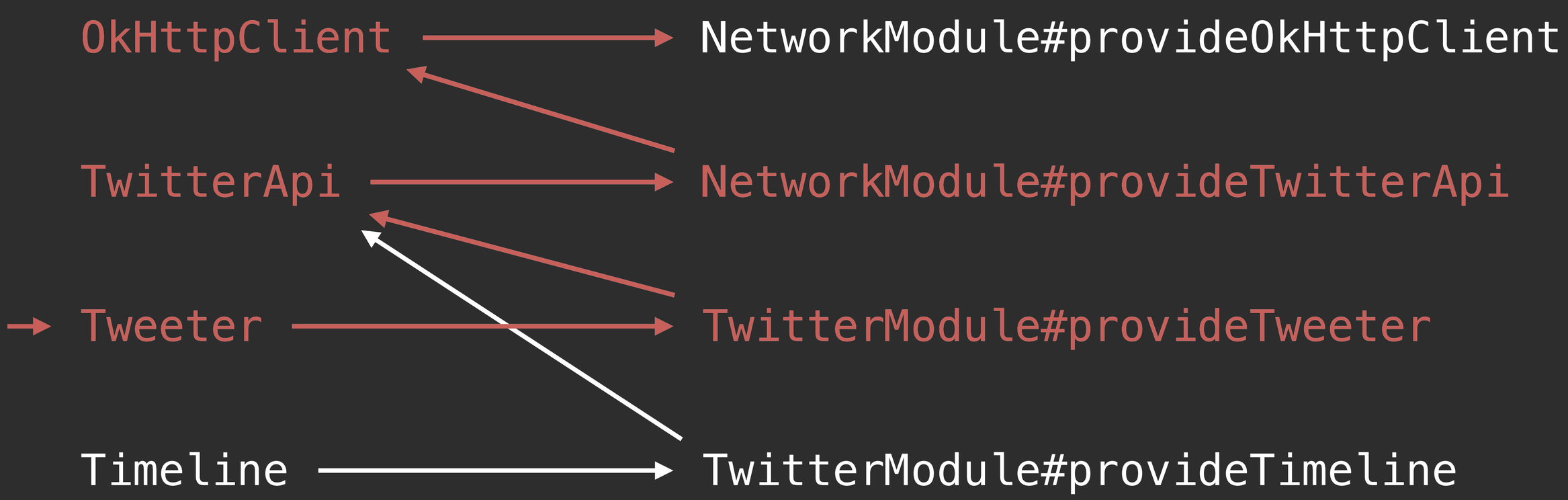
Providing Dependencies



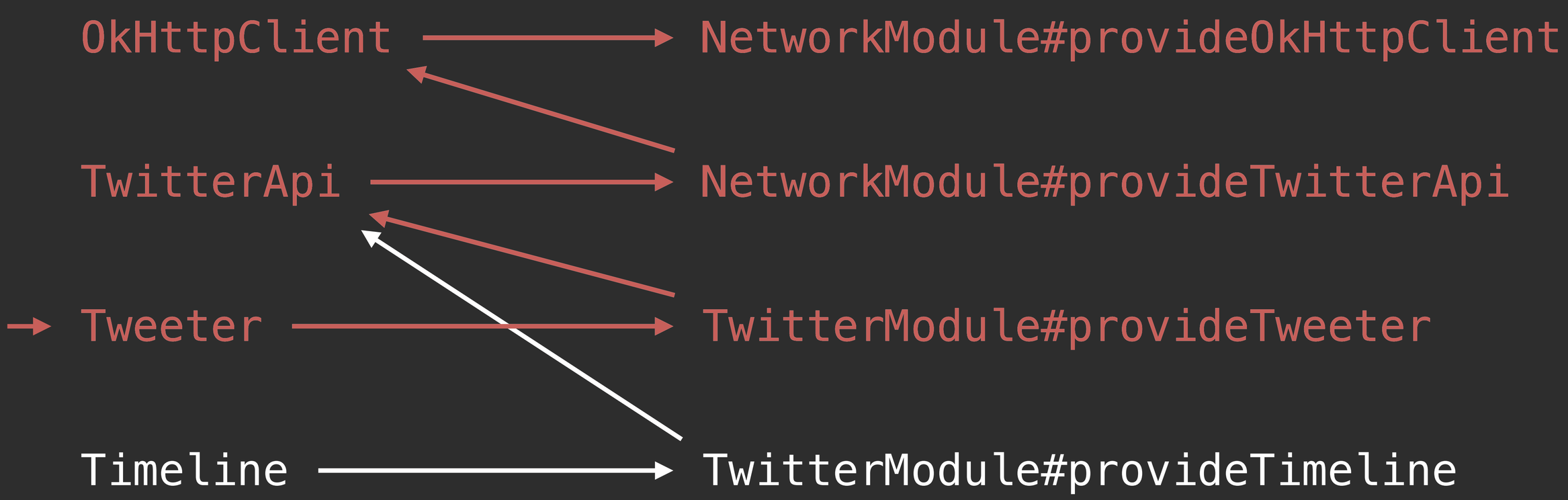
Providing Dependencies



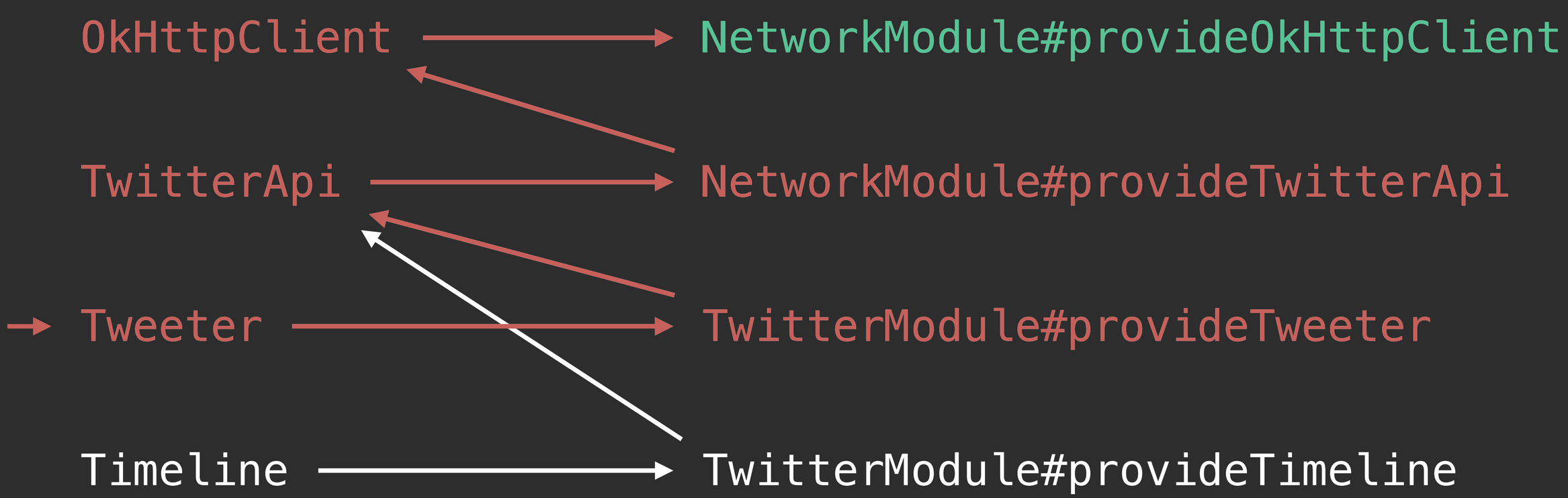
Providing Dependencies



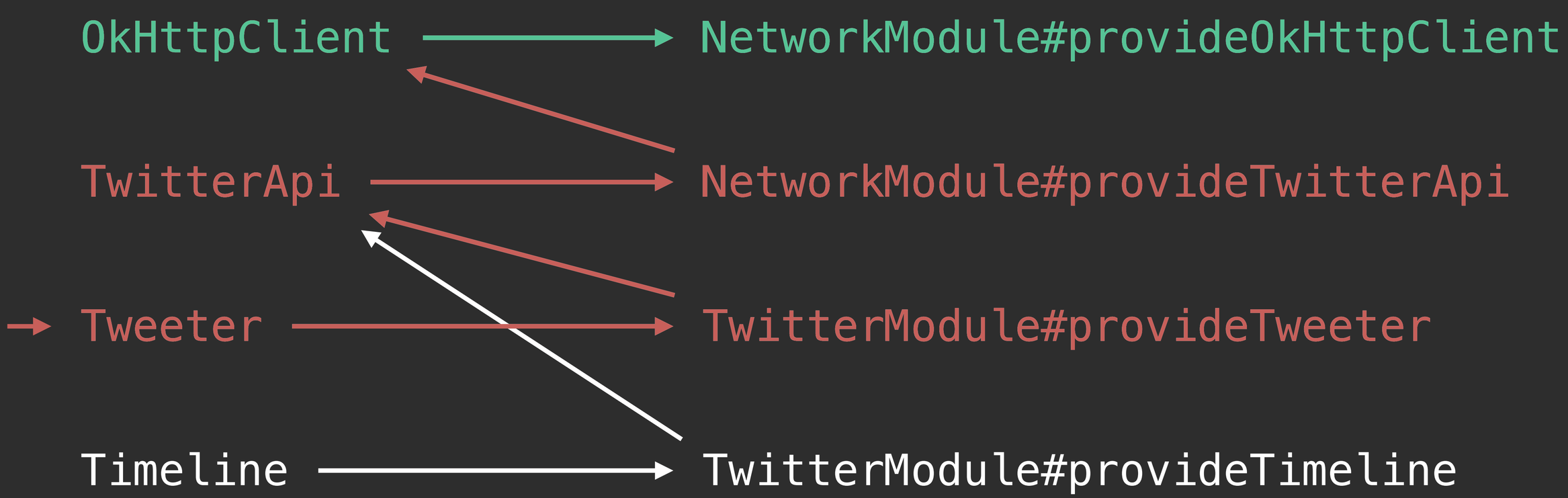
Providing Dependencies



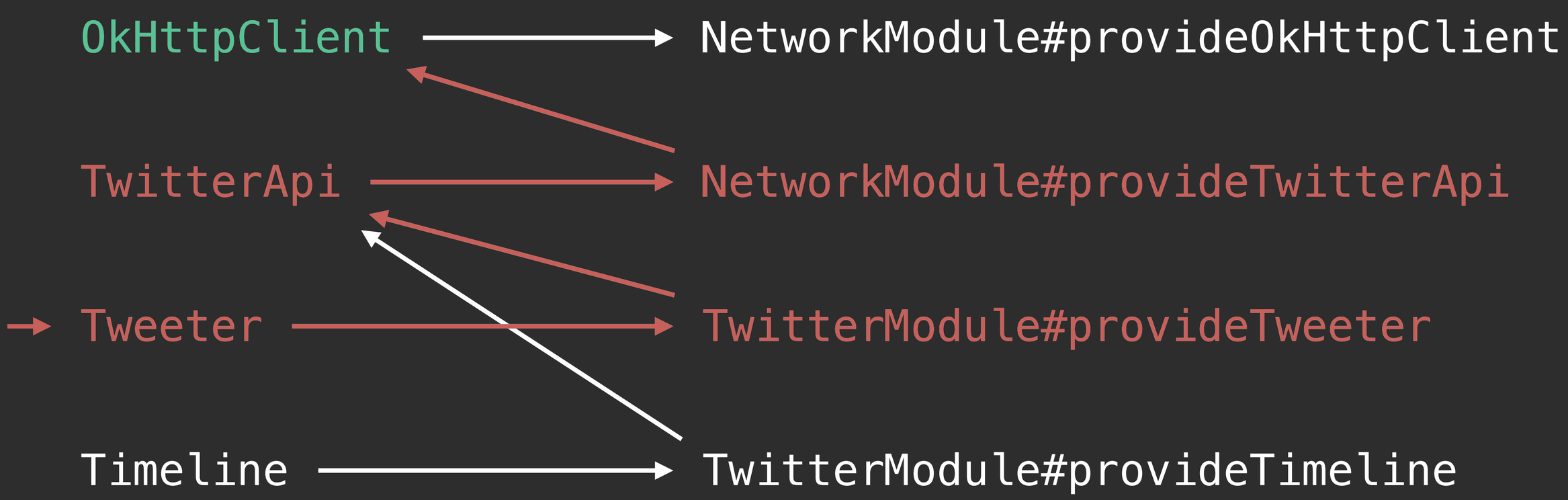
Providing Dependencies



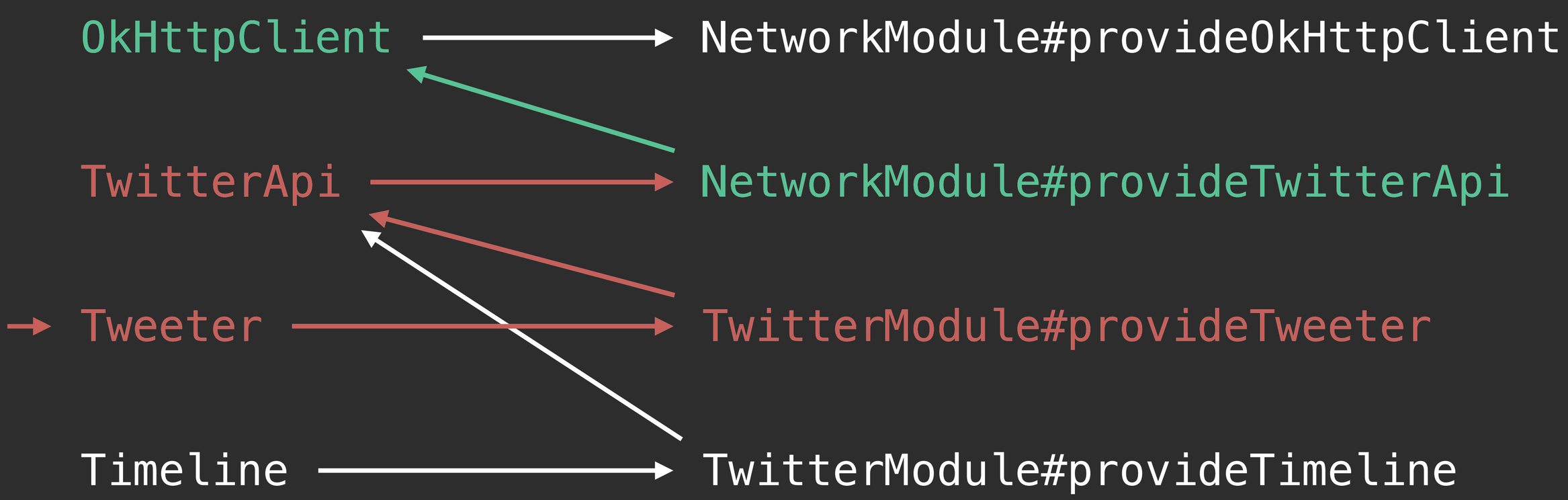
Providing Dependencies



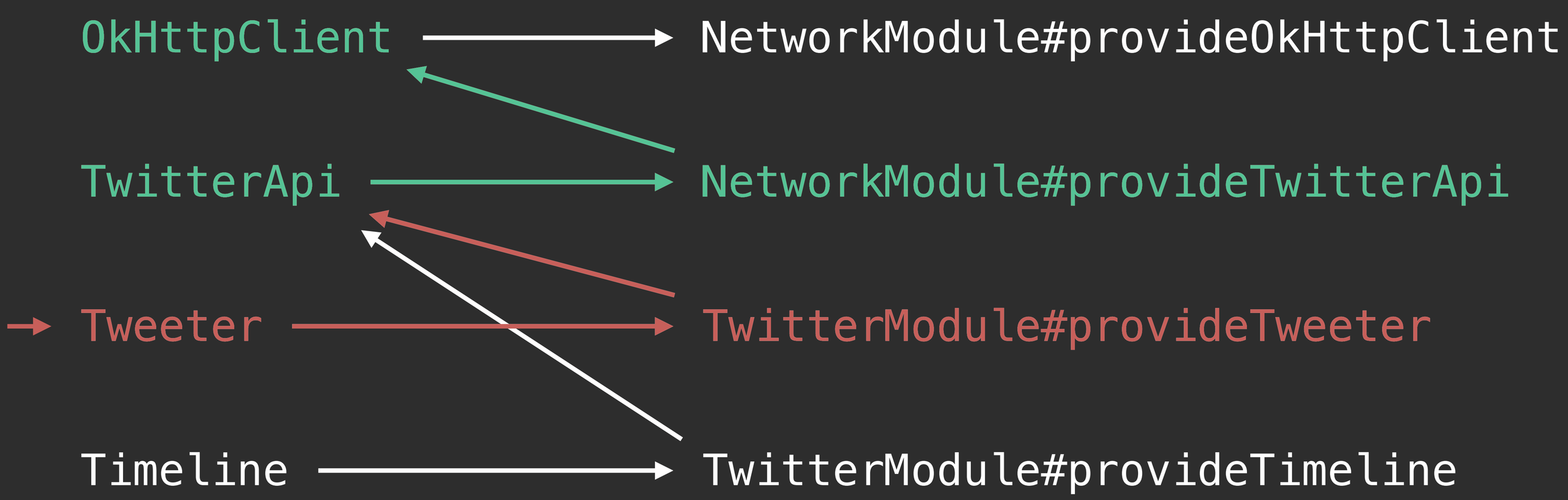
Providing Dependencies



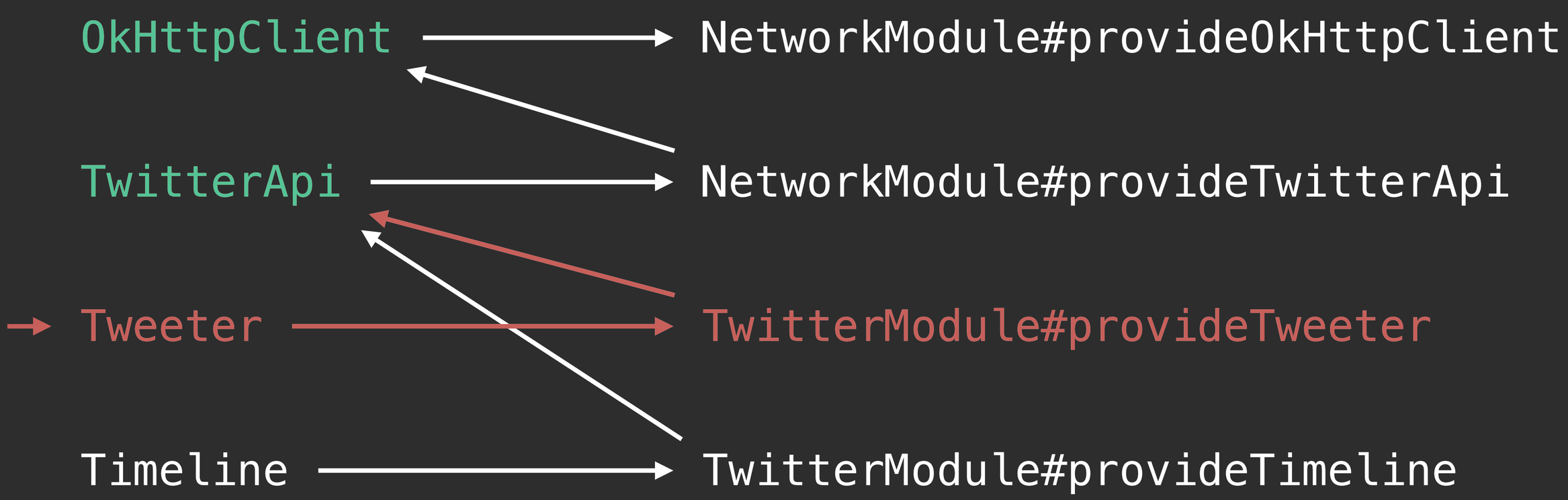
Providing Dependencies



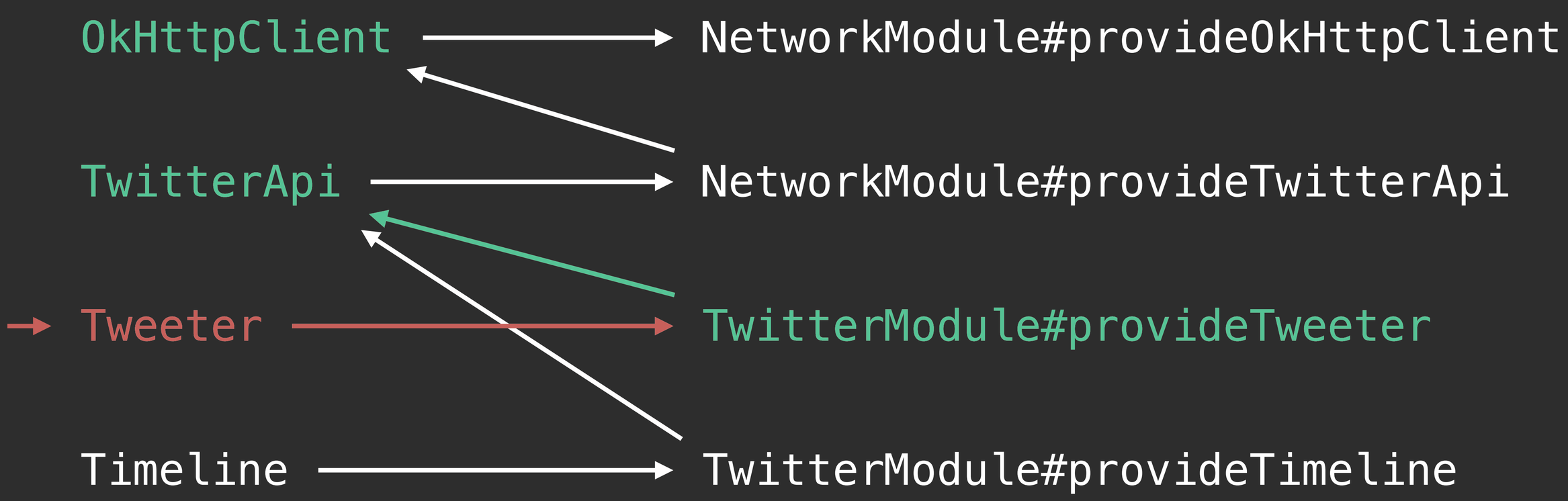
Providing Dependencies



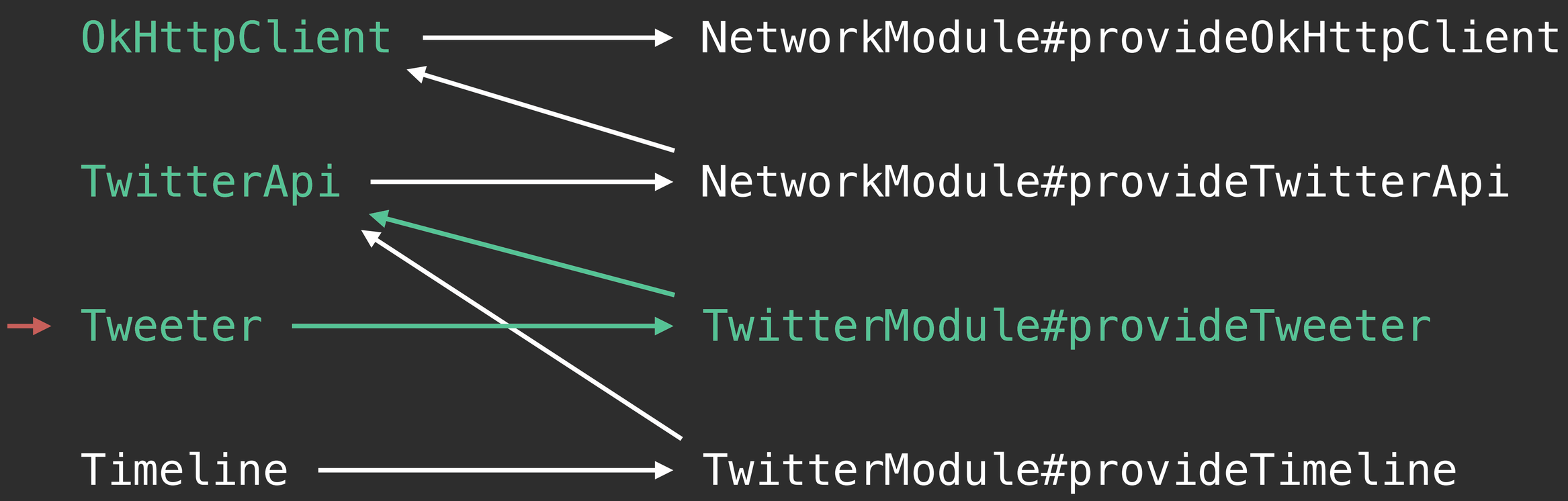
Providing Dependencies



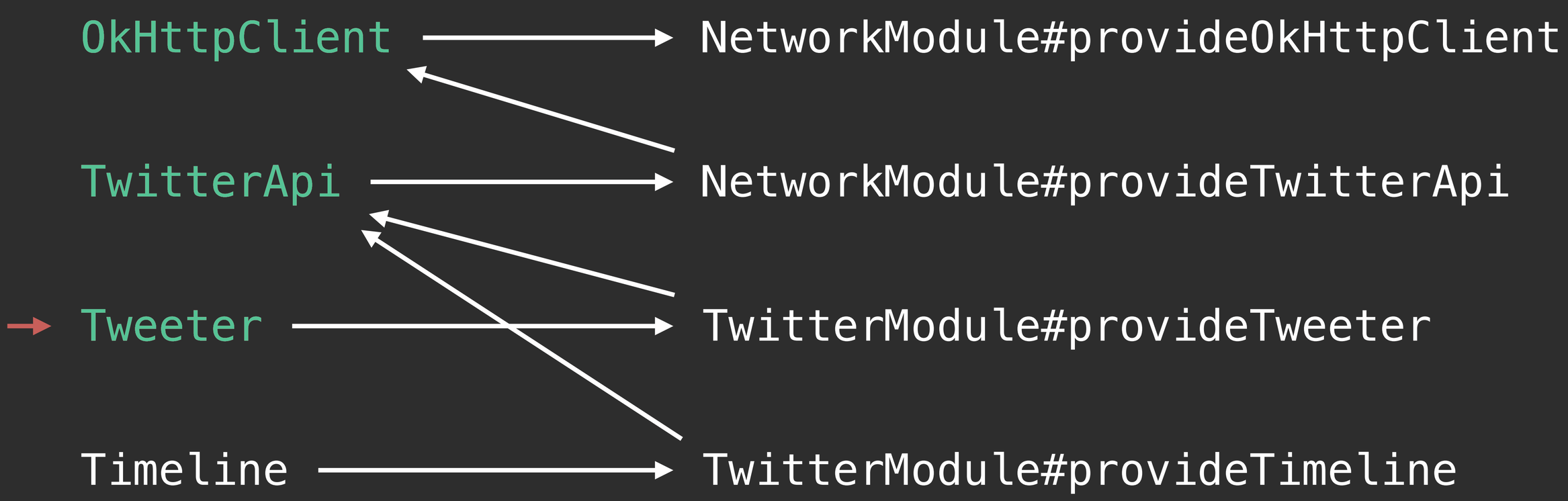
Providing Dependencies



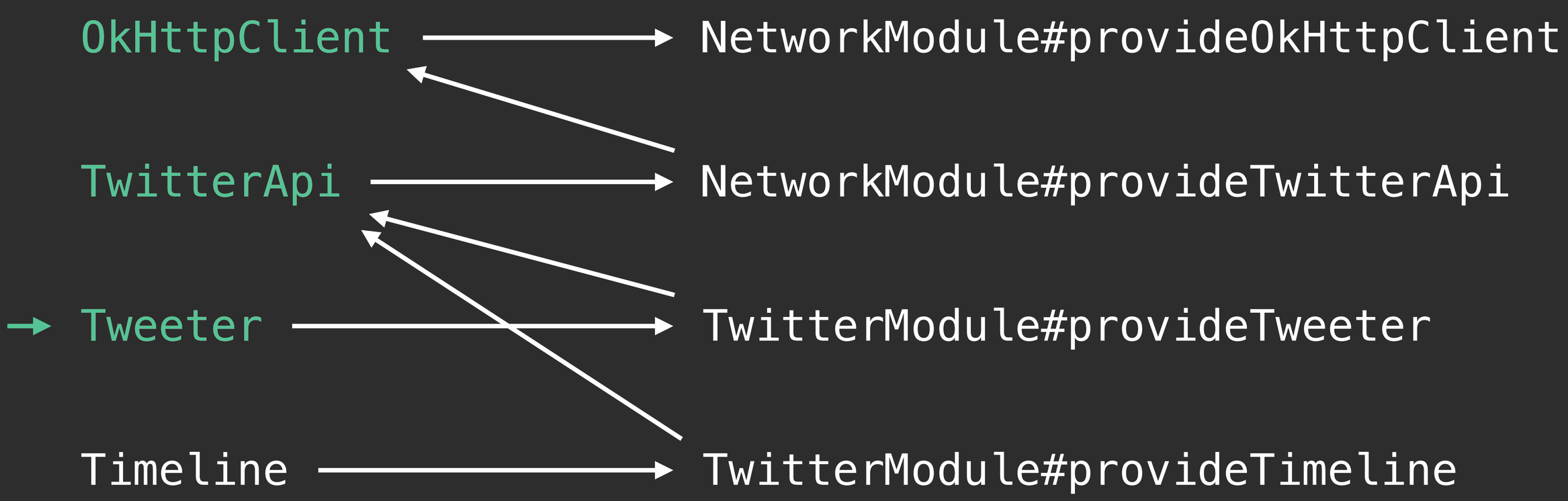
Providing Dependencies



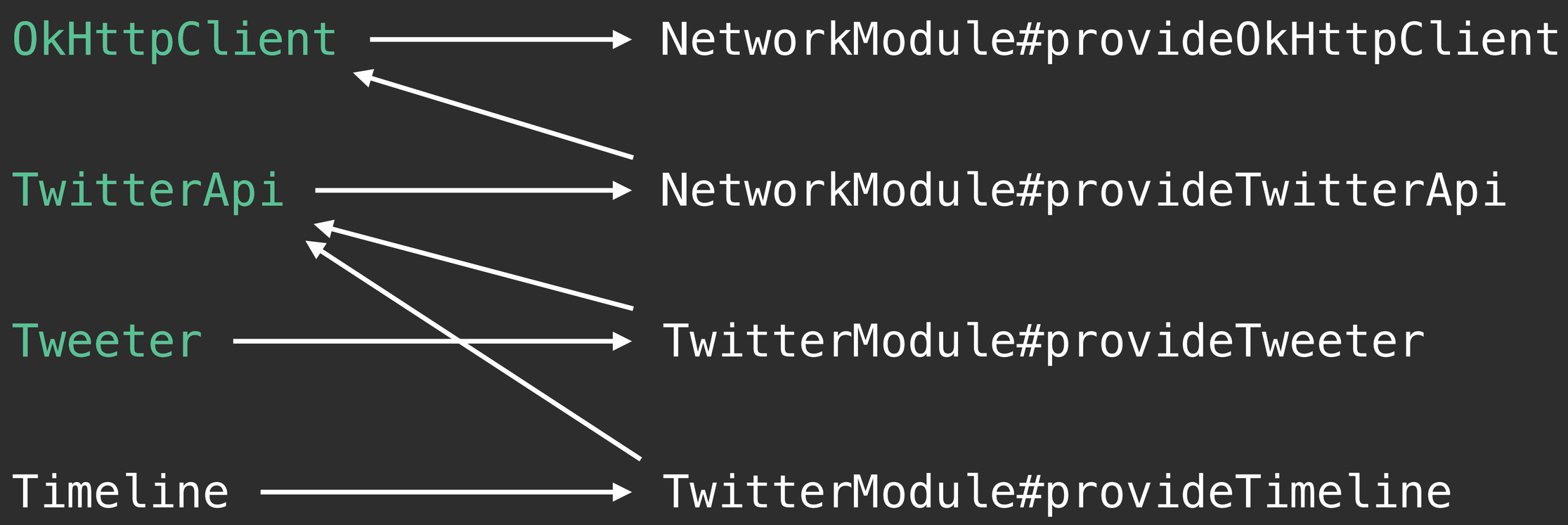
Providing Dependencies



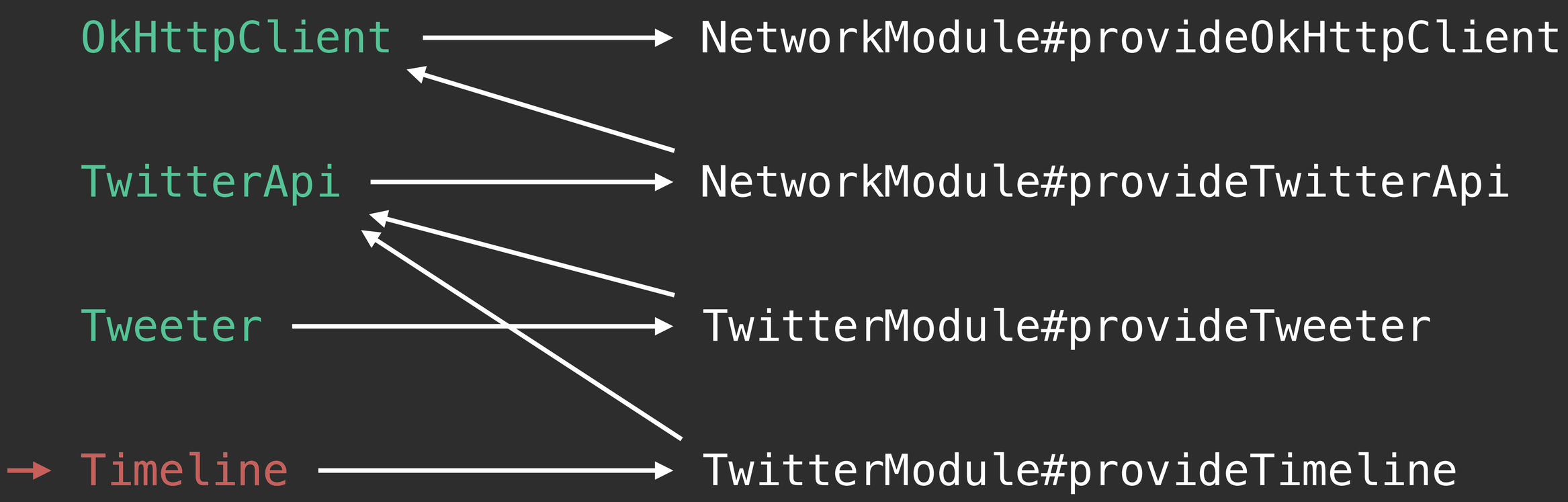
Providing Dependencies



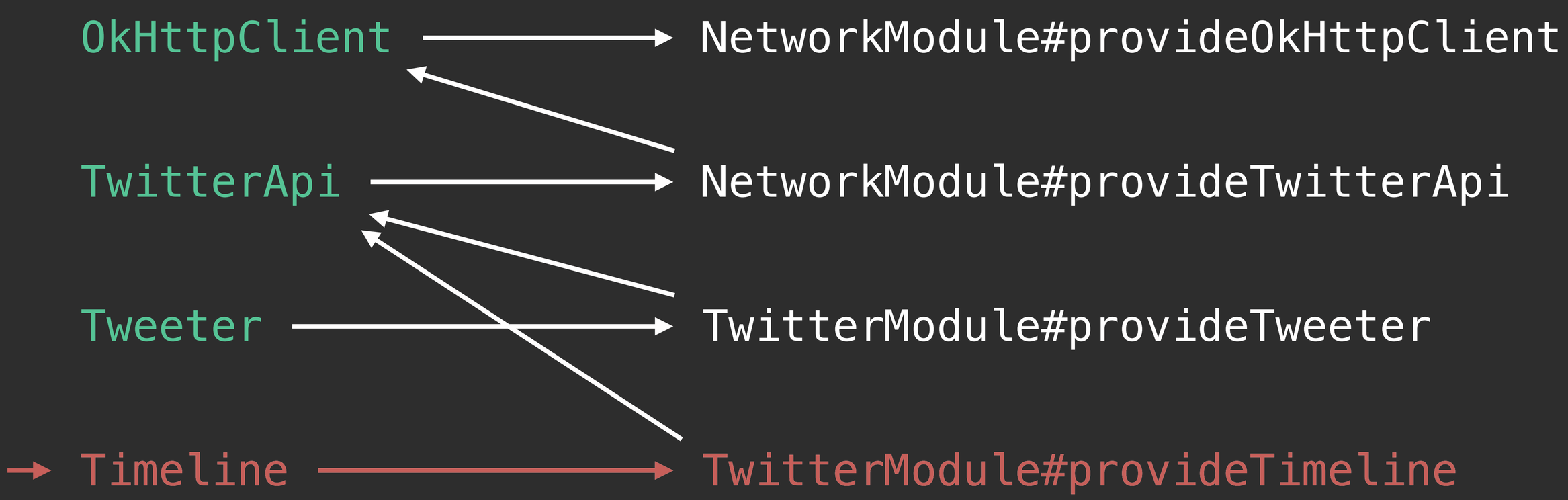
Providing Dependencies



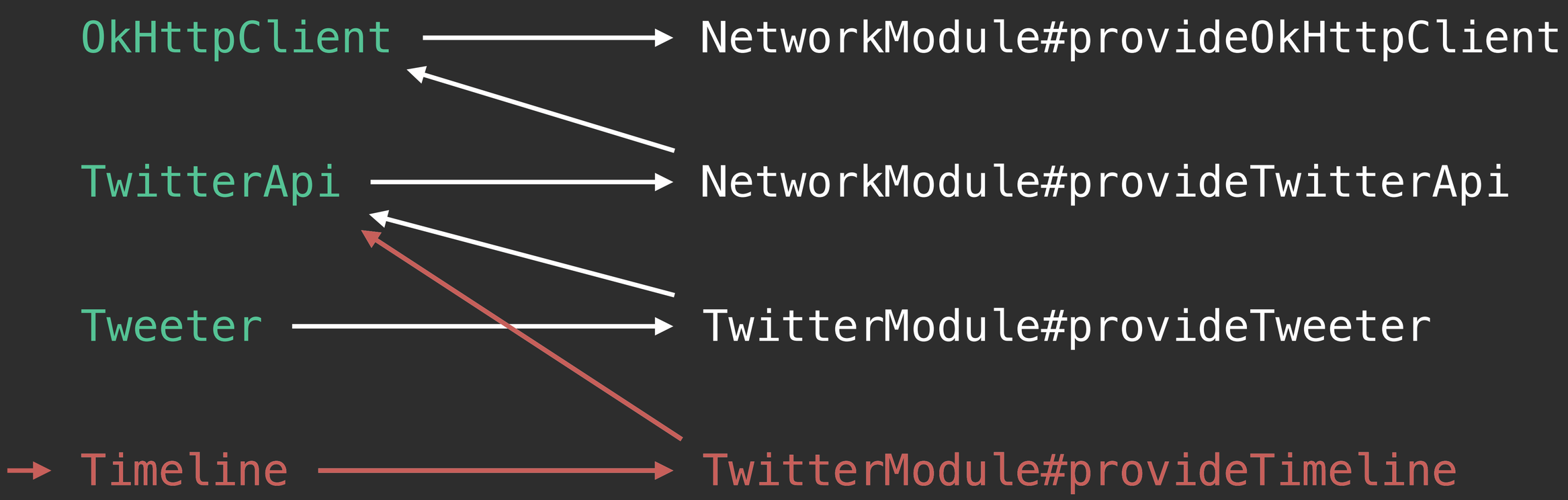
Providing Dependencies



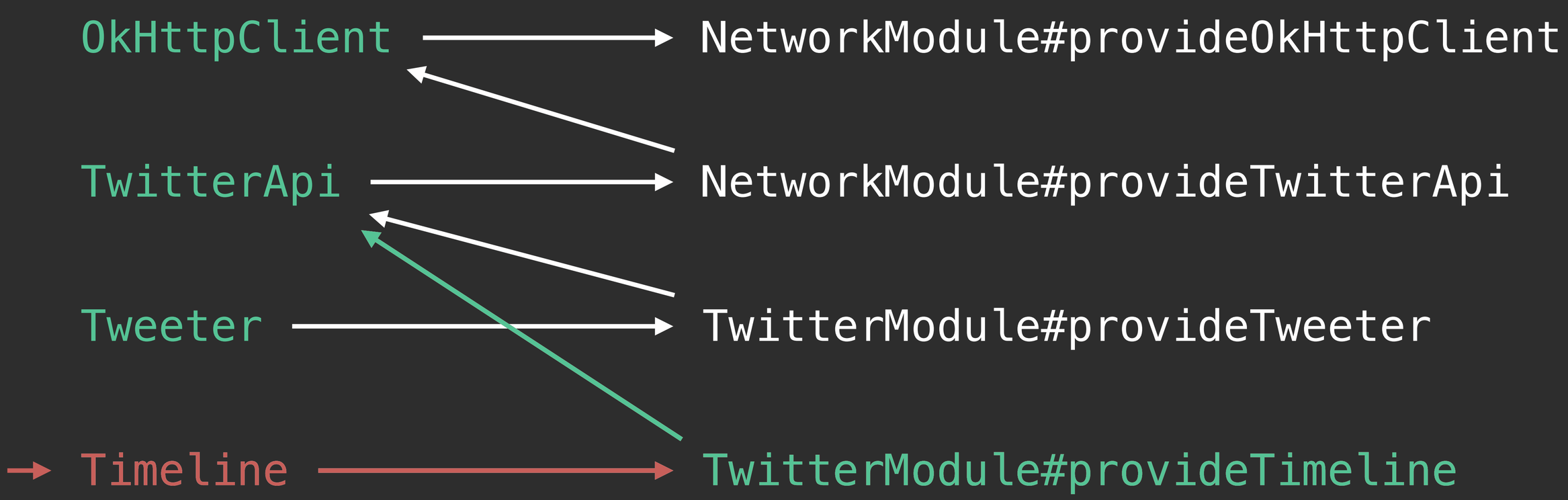
Providing Dependencies



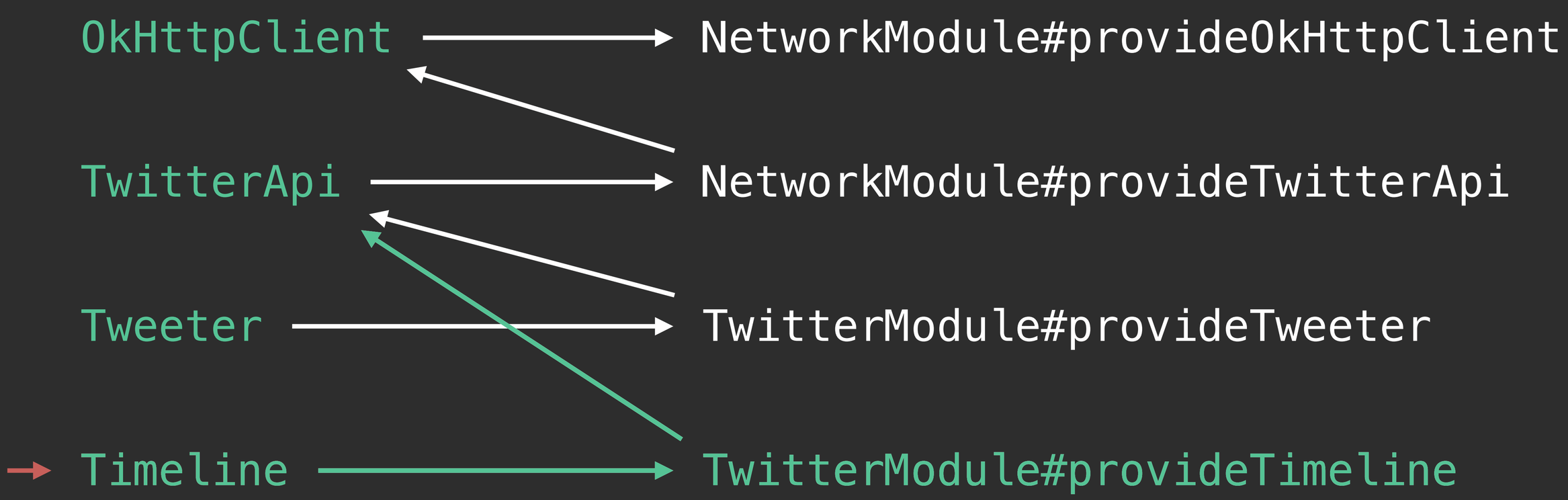
Providing Dependencies



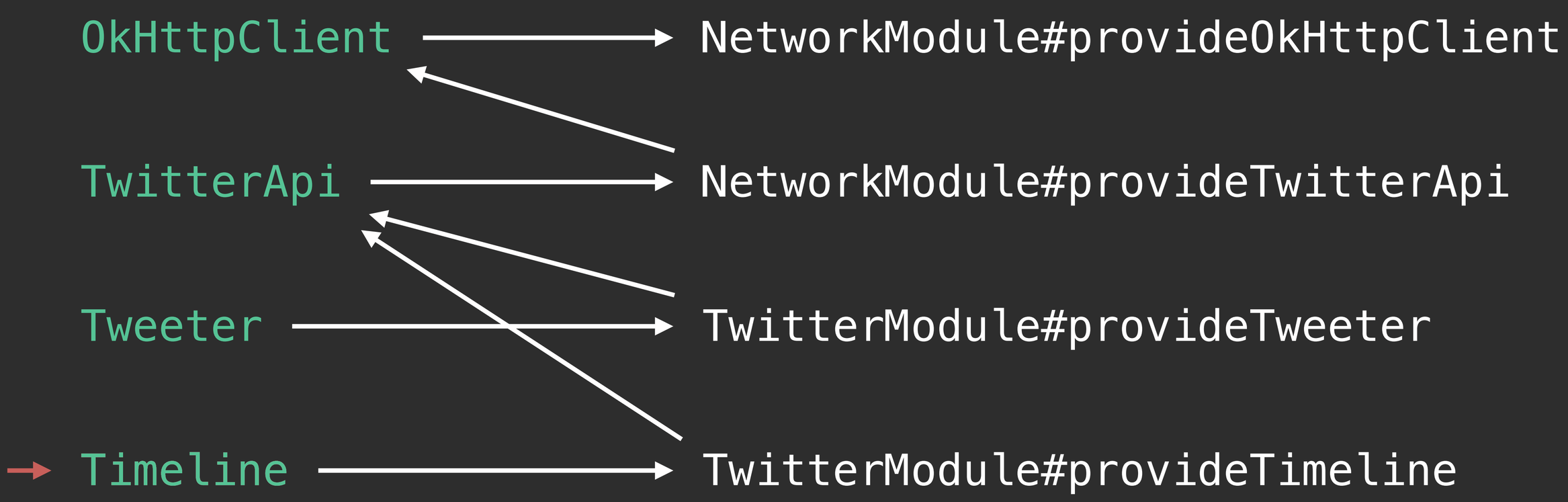
Providing Dependencies



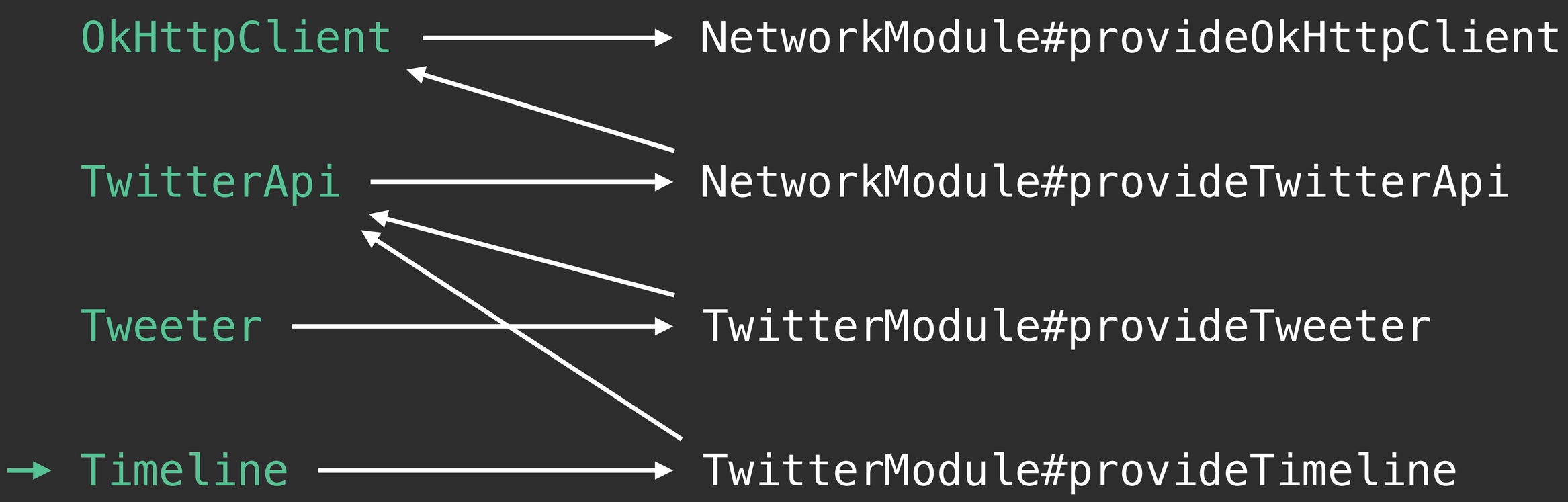
Providing Dependencies



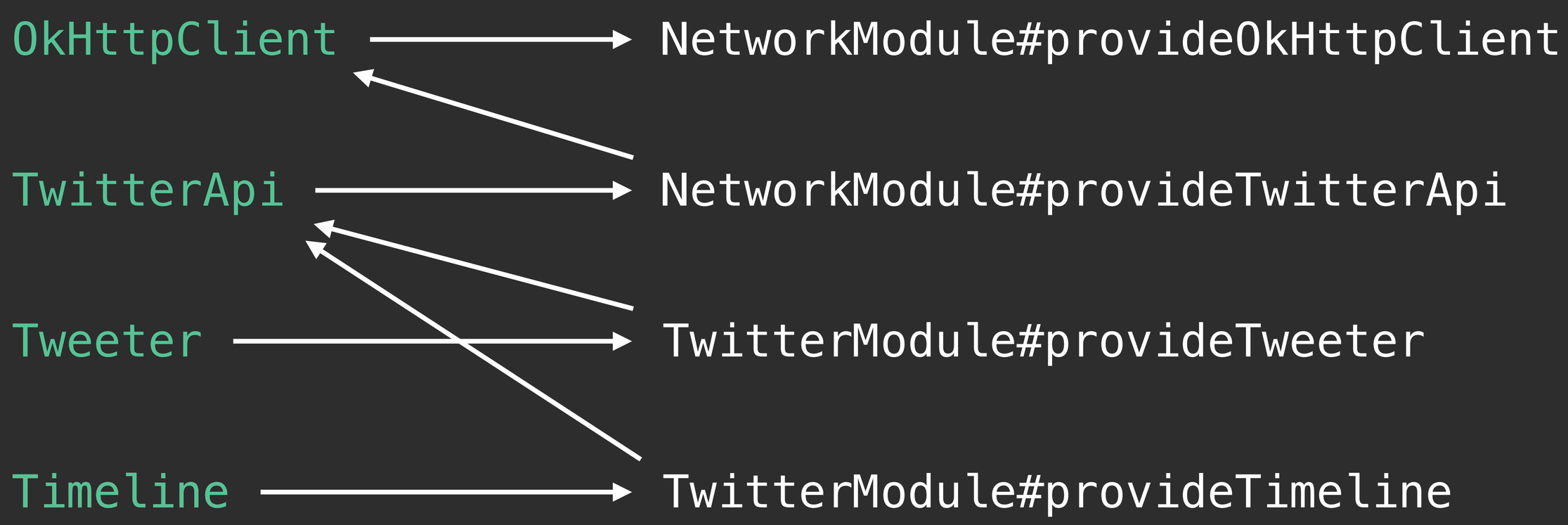
Providing Dependencies



Providing Dependencies



Providing Dependencies



Requesting Dependencies

Requesting Dependencies

- `@Inject` annotation required.

Requesting Dependencies

- `@Inject` annotation required.
- Constructor, field, and method injection.

Constructor Injection

Constructor Injection

- `@Inject` on a single constructor.

Constructor Injection

- `@Inject` on a single constructor.
- Constructor parameters are dependencies.

Constructor Injection

- `@Inject` on a single constructor.
- Constructor parameters are dependencies.
- Dependencies can be stored in private and final fields.

```
public class TwitterApplication {
  private final Tweeter tweeter;
  private final Timeline timeline;

  @Inject
  public TwitterApplication(Tweeter tweeter, Timeline timeline) {
    this.tweeter = tweeter;
    this.timeline = timeline;
  }

  // ...
}
```

Constructor Injection

- `@Inject` on a single constructor.
- Constructor parameters are dependencies.
- Dependencies can be stored in private and final fields.

Constructor Injection

- `@Inject` on a single constructor.
- Constructor parameters are dependencies.
- Dependencies can be stored in private and final fields.
- Implicitly made available for downstream injection.

```java
@Module
public class NetworkModule {
  @Provides @Singleton
  OkHttpClient provideOkHttpClient() {
    return new OkHttpClient();
  }

  @Provides @Singleton
  TwitterApi provideTwitterApi(OkHttpClient client) {
    return new TwitterApi(client);
  }
}
```

```
@Module
public class NetworkModule {
  @Provides @Singleton
  OkHttpClient provideOkHttpClient() {
    return new OkHttpClient();
  }

  @Provides @Singleton
  TwitterApi provideTwitterApi(OkHttpClient client) {
    return new TwitterApi(client);
  }
}
```

```
public class TwitterApi {
  private final OkHttpClient client;

  public TwitterApi(OkHttpClient client) {
    this.client = client;
  }

  public void postTweet(String user, String tweet) {
    Request request = // TODO build POST request ...
    client.newCall(request).execute();
  }
}
```

```
public class TwitterApi {
  private final OkHttpClient client;

  @Inject
  public TwitterApi(OkHttpClient client) {
    this.client = client;
  }

  public void postTweet(String user, String tweet) {
    Request request = // TODO build POST request ...
    client.newCall(request).execute();
  }
}
```

```java
@Singleton
public class TwitterApi {
  private final OkHttpClient client;

  @Inject
  public TwitterApi(OkHttpClient client) {
    this.client = client;
  }

  public void postTweet(String user, String tweet) {
    Request request = // TODO build POST request ...
    client.newCall(request).execute();
  }
}
```

Method Injection

Method Injection

- `@Inject` on methods.

Method Injection

- `@Inject` on methods.
- Method parameters are dependencies.

Method Injection

- `@Inject` on methods.
- Method parameters are dependencies.
- Injection happens after object is fully instantiated.

Method Injection

- `@Inject` on methods.
- Method parameters are dependencies.
- Injection happens after object is fully instantiated.
- Only one valid use case: passing ‘this’ to a dependency.

```
public class TwitterApplication {
  private final Tweeter tweeter;
  private final Timeline timeline;

  @Inject
  public TwitterApplication(Tweeter tweeter, Timeline timeline) {
    this.tweeter = tweeter;
    this.timeline = timeline;
  }




}
```

```
public class TwitterApplication {
  private final Tweeter tweeter;
  private final Timeline timeline;

  @Inject
  public TwitterApplication(Tweeter tweeter, Timeline timeline) {
    this.tweeter = tweeter;
    this.timeline = timeline;
  }

  @Inject
  public void enableStreaming(Streaming streaming) {
    streaming.register(this);
  }
}
```

```
public class TwitterApplication {
  private final Tweeter tweeter;
  private final Timeline timeline;

  @Inject
  public TwitterApplication(Tweeter tweeter, Timeline timeline) {
    this.tweeter = tweeter;
    this.timeline = timeline;
  }

  @Inject
  public void enableStreaming(Streaming streaming) {
    streaming.register(this);
  }
}
```

Field Injection

Field Injection

- `@Inject` on fields for dependencies.

Field Injection

- `@Inject` on fields for dependencies.
- Field may not be private or final.

```
public class TwitterApplication {
  @Inject Tweeter tweeter;
  @Inject Timeline timeline;

  // ...
}
```

```
public class TwitterActivity extends Activity {
  @Inject Tweeter tweeter;
  @Inject Timeline timeline;

  // ...
}
```

```
public class TwitterProcessor extends AbstractProcessor {
  @Inject Tweeter tweeter;
  @Inject Timeline timeline;

  // ...
}
```

Field Injection

- `@Inject` on fields for dependencies.
- Field may not be private or final.

Field Injection

- `@Inject` on fields for dependencies.
- Field may not be private or final.
- Injection happens after the object is fully instantiated.

Field Injection

- `@Inject` on fields for dependencies.
- Field may not be private or final.
- Injection happens after the object is fully instantiated.
- Object is usually responsible for or aware of injection.

Components

Components

- Bridge between modules and injection.

Components

- Bridge between modules and injection.
- The injector.

```
@Component
public interface TwitterComponent {
}
```

```
@Component(modules = {
    NetworkModule.class,
    TwitterModule.class,
})
public interface TwitterComponent {
}
```

```
@Singleton
@Component(modules = {
    NetworkModule.class,
    TwitterModule.class,
})
public interface TwitterComponent {
}
```

```
@Singleton
@Component(modules = {
    NetworkModule.class,
    TwitterModule.class,
})
public interface TwitterComponent {
  Tweeter tweeter();
  Timeline timeline();
}
```

```
TwitterComponent component = Dagger_TwitterComponent.builder()
    .networkModule(new NetworkModule())
    .twitterModule(new TwitterModule("JakeWharton"))
    .build();
```

```
TwitterComponent component = Dagger_TwitterComponent.builder()
    .networkModule(new NetworkModule())
    .twitterModule(new TwitterModule("JakeWharton"))
    .build();

Tweeter tweeter = component.tweeter();
Timeline timeline = component.timeline();
```

```
TwitterComponent component = Dagger_TwitterComponent.builder()
    .networkModule(new NetworkModule())
    .twitterModule(new TwitterModule("JakeWharton"))
    .build();

Tweeter tweeter = component.tweeter();
Timeline timeline = component.timeline();
```

```
TwitterComponent component = Dagger_TwitterComponent.builder()
    .twitterModule(new TwitterModule("JakeWharton"))
    .build();

Tweeter tweeter = component.tweeter();
Timeline timeline = component.timeline();
```

```
TwitterComponent component = Dagger_TwitterComponent.builder()
    .twitterModule(new TwitterModule("JakeWharton"))
    .build();

Tweeter tweeter = component.tweeter();
tweeter.tweet("Hello, #Devoxx 2014!");


Timeline timeline = component.timeline();
timeline.loadMore(20);
for (Tweet tweet : timeline.get()) {
  System.out.println(tweet);
}
```

```java
public class TwitterApplication implements Runnable {
  private final Tweeter tweeter;
  private final Timeline timeline;

  @Inject
  public TwitterApplication(Tweeter tweeter, Timeline timeline) {
    this.tweeter = tweeter;
    this.timeline = timeline;
  }

  @Override public void run() {
    tweeter.tweet("Hello #Devoxx 2014!");

    timeline.loadMore(20);
    for (Tweet tweet : timeline.get()) {
      System.out.println(tweet);
    }
  }
}
```

```
@Singleton
@Component(modules = {
    NetworkModule.class,
    TwitterModule.class,
})
interface TwitterComponent {
  TwitterApplication app();
}
```

```java
@Singleton
@Component(modules = {
    NetworkModule.class,
    TwitterModule.class,
})
interface TwitterComponent {
  TwitterApplication app();
}


TwitterComponent component = Dagger_TwitterComponent.builder()
    .twitterModule(new TwitterModule("JakeWharton"))
    .build();
component.app().run();
```

```java
public class TwitterApplication implements Runnable {
  @Inject Tweeter tweeter;
  @Inject Timeline timeline;

  @Override public void run() {
    tweeter.tweet("Hello #Devoxx 2014!");

    timeline.loadMore(20);
    for (Tweet tweet : timeline.get()) {
      System.out.println(tweet);
    }
  }
}
```

```
@Singleton
@Component(modules = {
    NetworkModule.class,
    TwitterModule.class,
})
interface TwitterComponent {
  void injectApp(TwitterApplication app);
}
```

```java
@Singleton
@Component(modules = {
    NetworkModule.class,
    TwitterModule.class,
})
interface TwitterComponent {
  void injectApp(TwitterApplication app);
}


TwitterComponent component = Dagger_TwitterComponent.builder()
    .twitterModule(new TwitterModule("JakeWharton"))
    .build();
```

```
@Singleton
@Component(modules = {
    NetworkModule.class,
    TwitterModule.class,
})
interface TwitterComponent {
  void injectApp(TwitterApplication app);
}


TwitterComponent component = Dagger_TwitterComponent.builder()
    .twitterModule(new TwitterModule("JakeWharton"))
    .build();


TwitterApplication app = new TwitterApplication();
component.injectApp(app);
app.run();
```

```
@Singleton
@Component(modules = {
    NetworkModule.class,
    TwitterModule.class,
})
interface TwitterComponent {
  void injectActivity(TwitterActivity activity);
}
```

```
TwitterComponent component = Dagger_TwitterComponent.builder()
    .twitterModule(new TwitterModule("JakeWharton"))
    .build();
```

```
TwitterActivity activity = // Android creates instance...
component.injectActivity(activity);
```

```java
@Singleton
@Component(modules = {
    NetworkModule.class,
    TwitterModule.class,
})
interface TwitterComponent {
  void injectActivity(TwitterActivity activity);
}
```

```java
TwitterComponent component = Dagger_TwitterComponent.builder()
    .twitterModule(new TwitterModule("JakeWharton"))
    .build();
```

```java
TwitterActivity activity = // Android creates instance...
component.injectActivity(activity);
```

```
@Singleton
@Component(modules = {
    NetworkModule.class,
    TwitterModule.class,
})
interface TwitterComponent {
  void injectProcessor(TwitterProcessor processor);
}
```

```
TwitterComponent component = Dagger_TwitterComponent.builder()
    .twitterModule(new TwitterModule("JakeWharton"))
    .build();
```

```
TwitterProcessor processor = // ServiceLoader creates instance...
component.injectProcessor(processor);
```

```
@Singleton
@Component(modules = {
    NetworkModule.class,
    TwitterModule.class,
})
interface TwitterComponent {
  void injectProcessor(TwitterProcessor processor);
}


TwitterComponent component = Dagger_TwitterComponent.builder()
    .twitterModule(new TwitterModule("JakeWharton"))
    .build();


TwitterProcessor processor = // ServiceLoader creates instance...
component.injectProcessor(processor);
```

```
@Singleton
@Component(modules = {
    NetworkModule.class,
    TwitterModule.class,
})
interface TwitterComponent {
  TwitterApplication injectApp(TwitterApplication app);
}


TwitterComponent component = Dagger_TwitterComponent.builder()
    .twitterModule(new TwitterModule("JakeWharton"))
    .build();


TwitterApplication app = new TwitterApplication();
component.injectApp(app).run();
```

```
@Singleton
@Component(modules = {
    NetworkModule.class,
    TwitterModule.class,
})
interface TwitterComponent {
  TwitterApplication injectApp(TwitterApplication app);
}


TwitterComponent component = Dagger_TwitterComponent.builder()
    .twitterModule(new TwitterModule("JakeWharton"))
    .build();


TwitterApplication app = new TwitterApplication();
component.injectApp(app).run();
```

Components

- Bridge between modules and injection.
- The injector.

Components

- Bridge between modules and injection.
- The injector.
- Implementation of scopes.

```
@Singleton
@Component(modules = NetworkModule.class)
public interface ApiComponent {
}
```

```
@Singleton
@Component(modules = NetworkModule.class)
public interface ApiComponent {
}
```

```
@Component(
    modules = TwitterModule.class
)
public interface TwitterComponent {
  TwitterApplication app();
}
```

```java
@Singleton
@Component(modules = NetworkModule.class)
public interface ApiComponent {
}
```

```java
@Component(
    dependencies = ApiComponent.class,
    modules = TwitterModule.class
)
public interface TwitterComponent {
  TwitterApplication app();
}
```

```
@Singleton
@Component(modules = NetworkModule.class)
public interface ApiComponent {
  TwitterApi api();
}
```

```
@Component(
    dependencies = ApiComponent.class,
    modules = TwitterModule.class
)
public interface TwitterComponent {
  TwitterApplication app();
}
```

```
@Singleton
@Component(modules = NetworkModule.class)
public interface ApiComponent {
  TwitterApi api();
}
```

```
@Singleton
@Component(
    dependencies = ApiComponent.class,
    modules = TwitterModule.class
)
public interface TwitterComponent {
  TwitterApplication app();
}
```

```java
@Singleton
@Component(modules = NetworkModule.class)
public interface ApiComponent {
  TwitterApi api();
}
```

```java
@Singleton
@Component(
    dependencies = ApiComponent.class,
    modules = TwitterModule.class
)
public interface TwitterComponent {
  TwitterApplication app();
}
```

```java
@Singleton
@Component(modules = NetworkModule.class)
public interface ApiComponent {
  TwitterApi api();
}
```

```java
@Component(
    dependencies = ApiComponent.class,
    modules = TwitterModule.class
)
public interface TwitterComponent {
  TwitterApplication app();
}
```

```java
@Module
public class TwitterModule {
  private final String user;

  public TwitterModule(String user) {
    this.user = user;
  }

  @Provides @Singleton
  Tweeter provideTweeter(TwitterApi api) {
    return new Tweeter(api, user);
  }

  @Provides @Singleton
  Timeline provideTimeline(TwitterApi api) {
    return new Timeline(api, user);
  }
}
```

```java
@Module
public class TwitterModule {
  private final String user;

  public TwitterModule(String user) {
    this.user = user;
  }

  @Provides @Singleton
  Tweeter provideTweeter(TwitterApi api) {
    return new Tweeter(api, user);
  }

  @Provides @Singleton
  Timeline provideTimeline(TwitterApi api) {
    return new Timeline(api, user);
  }
}
```

```java
@Module
public class TwitterModule {
  private final String user;

  public TwitterModule(String user) {
    this.user = user;
  }

  @Provides
  Tweeter provideTweeter(TwitterApi api) {
    return new Tweeter(api, user);
  }

  @Provides
  Timeline provideTimeline(TwitterApi api) {
    return new Timeline(api, user);
  }
}
```

```
ApiComponent apiComponent = Dagger_ApiComponent.builder().build();
```

```
ApiComponent apiComponent = Dagger_ApiComponent.create();
```

```
ApiComponent apiComponent = Dagger_ApiComponent.create();
```

```
ApiComponent apiComponent = Dagger_ApiComponent.create();



TwitterComponent twitterComponent = Dagger_TwitterComponent.builder()
    .apiComponent(apiComponent)
    .twitterModule(new TwitterModule("Jake Wharton"))
    .build();
```

```
ApiComponent apiComponent = Dagger_ApiComponent.create();



TwitterComponent twitterComponent = Dagger_TwitterComponent.builder()
    .apiComponent(apiComponent)
    .twitterModule(new TwitterModule("Jake Wharton"))
    .build();



component.app().run();
```

Scope Annotations

Scope Annotations

- Only create a single instance.

Scope Annotations

- Only create a single instance.
- `@Singleton` is the “largest” scope.

Scope Annotations

- Only create a single instance.
- `@Singleton` is the “largest” scope.
- Custom annotations for semantic clarity, shorter lifetime.

Scope Annotations

- Only create a single instance.
- `@Singleton` is the “largest” scope.
- Custom annotations for semantic clarity, shorter lifetime.

```
public @interface User {
}
```

Scope Annotations

- Only create a single instance.
- `@Singleton` is the “largest” scope.
- Custom annotations for semantic clarity, shorter lifetime.

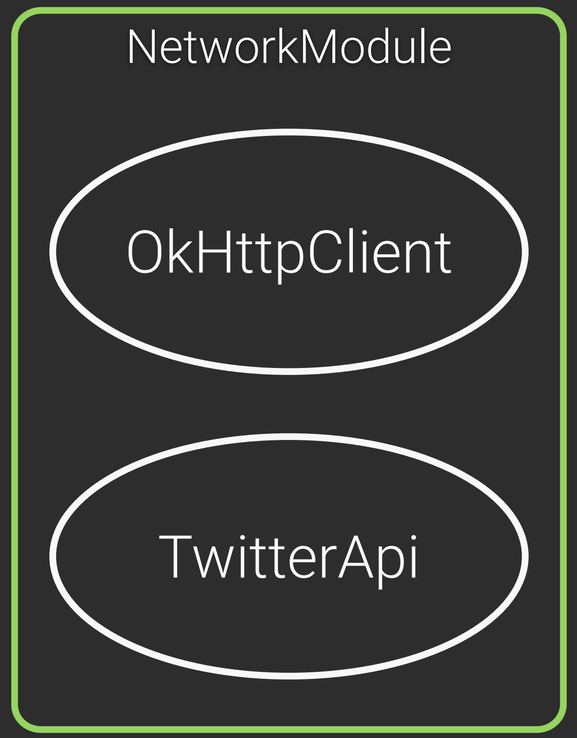
```
@Scope
public @interface User {
}
```

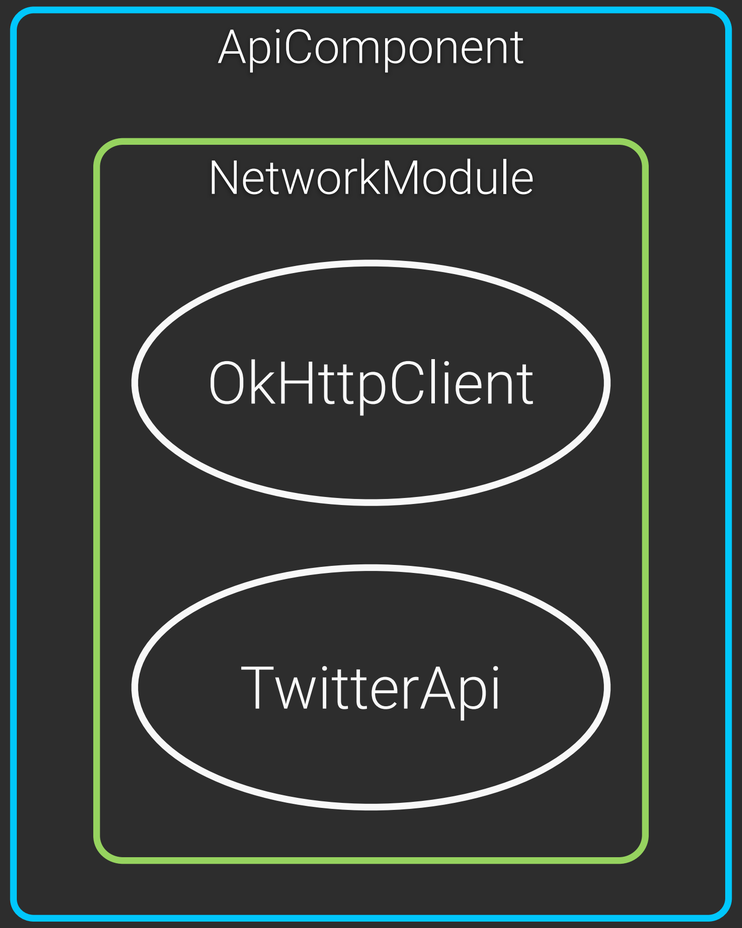
```java
@Module
public class TwitterModule {
  private final String user;

  public TwitterModule(String user) {
    this.user = user;
  }

  @Provides
  Tweeter provideTweeter(TwitterApi api) {
    return new Tweeter(api, user);
  }

  @Provides
  Timeline provideTimeline(TwitterApi api) {
    return new Timeline(api, user);
  }
}
```

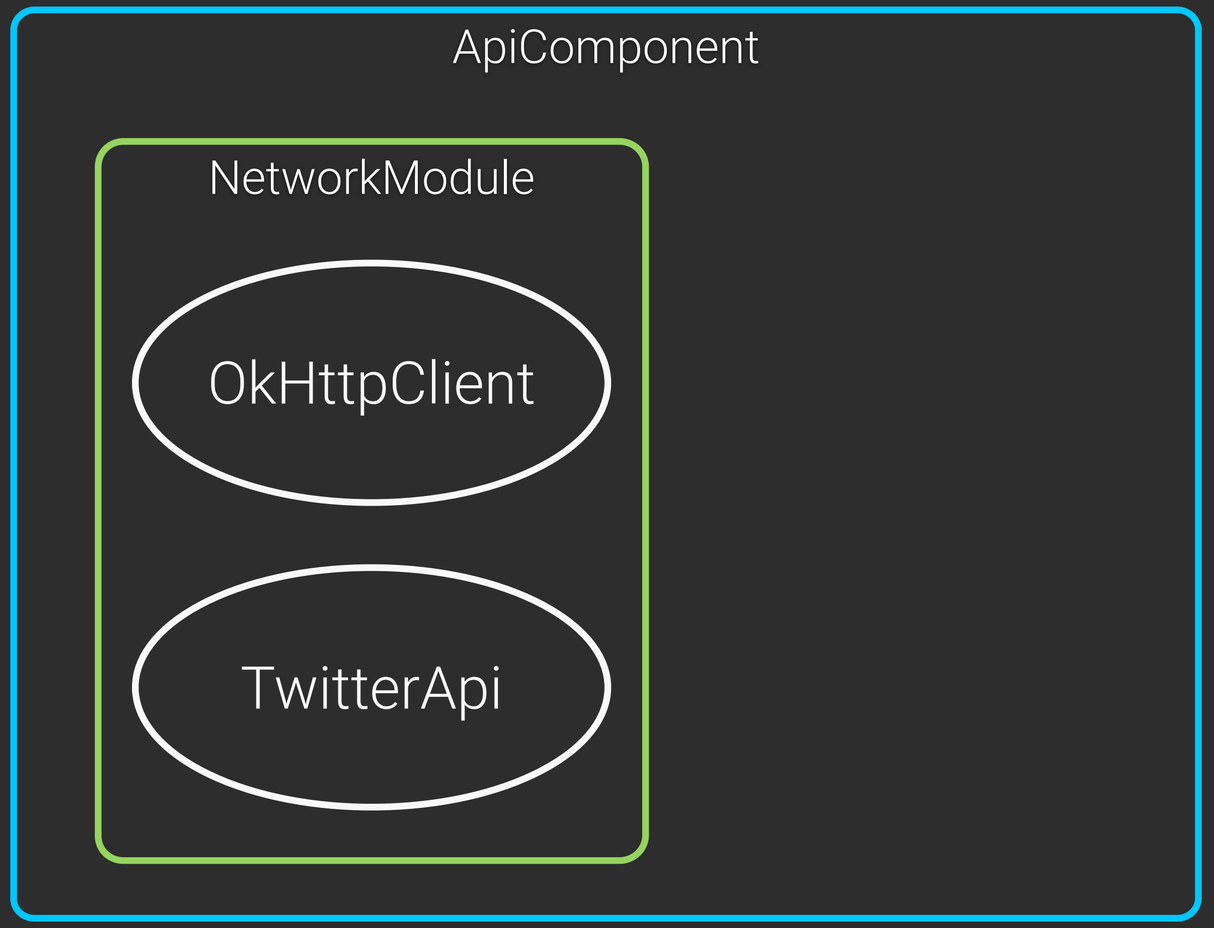
```java
@Module
public class TwitterModule {
  private final String user;

  public TwitterModule(String user) {
    this.user = user;
  }

  @Provides @User
  Tweeter provideTweeter(TwitterApi api) {
    return new Tweeter(api, user);
  }

  @Provides @User
  Timeline provideTimeline(TwitterApi api) {
    return new Timeline(api, user);
  }
}
```

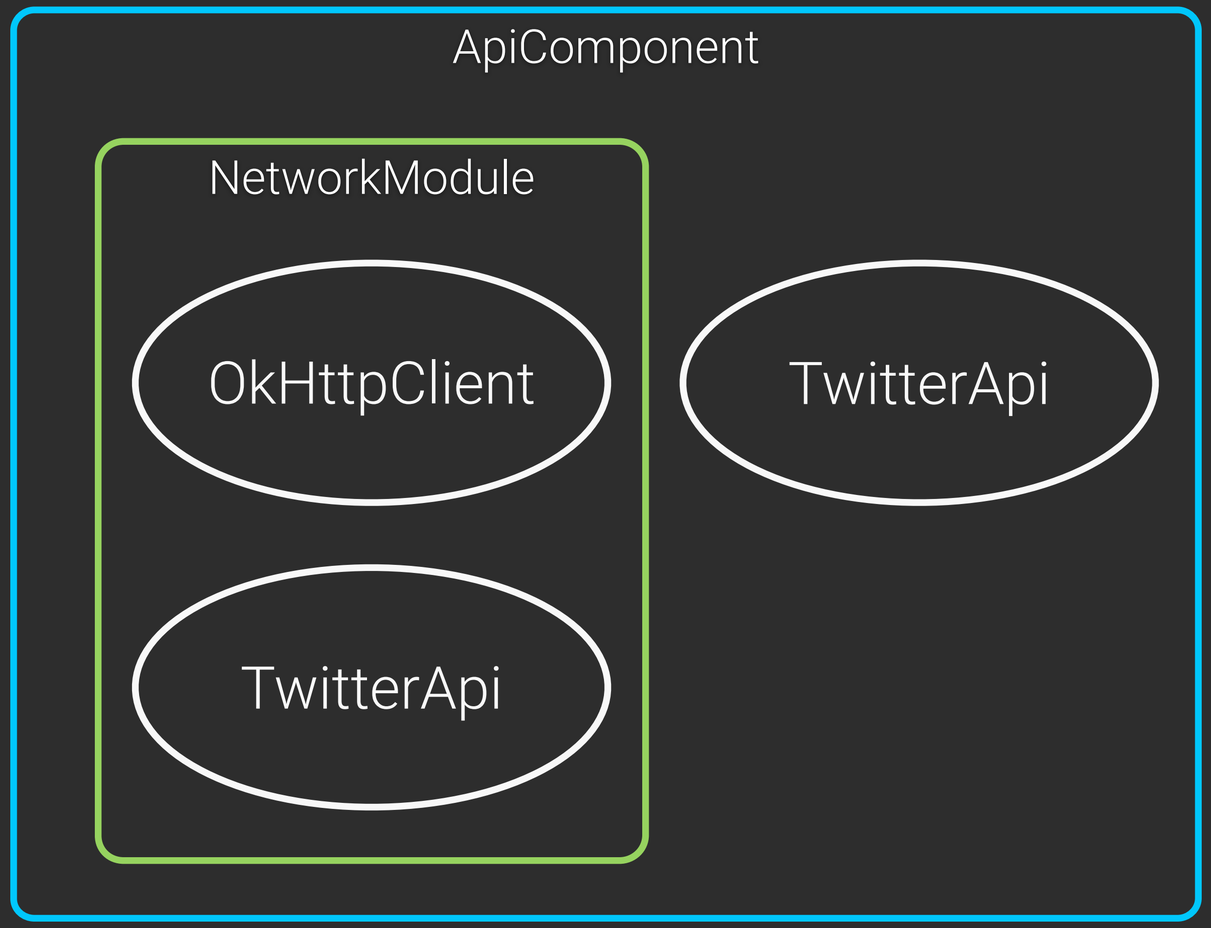
```
@Component(
    dependencies = ApiComponent.class,
    modules = TwitterModule.class
)
public interface TwitterComponent {
  TwitterApplication app();
}
```

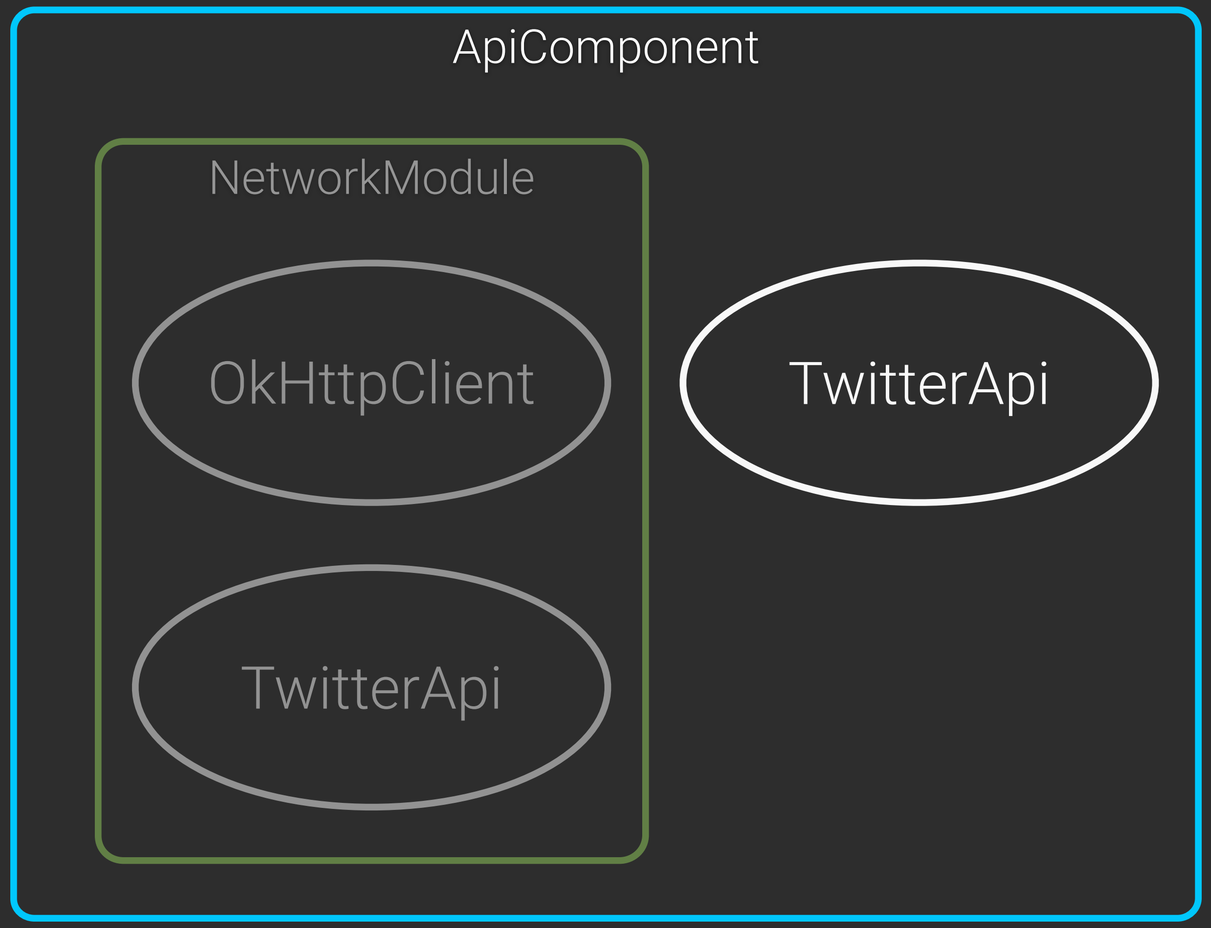
```
@User
@Component(
    dependencies = ApiComponent.class,
    modules = TwitterModule.class
)
public interface TwitterComponent {
  TwitterApplication app();
}
```

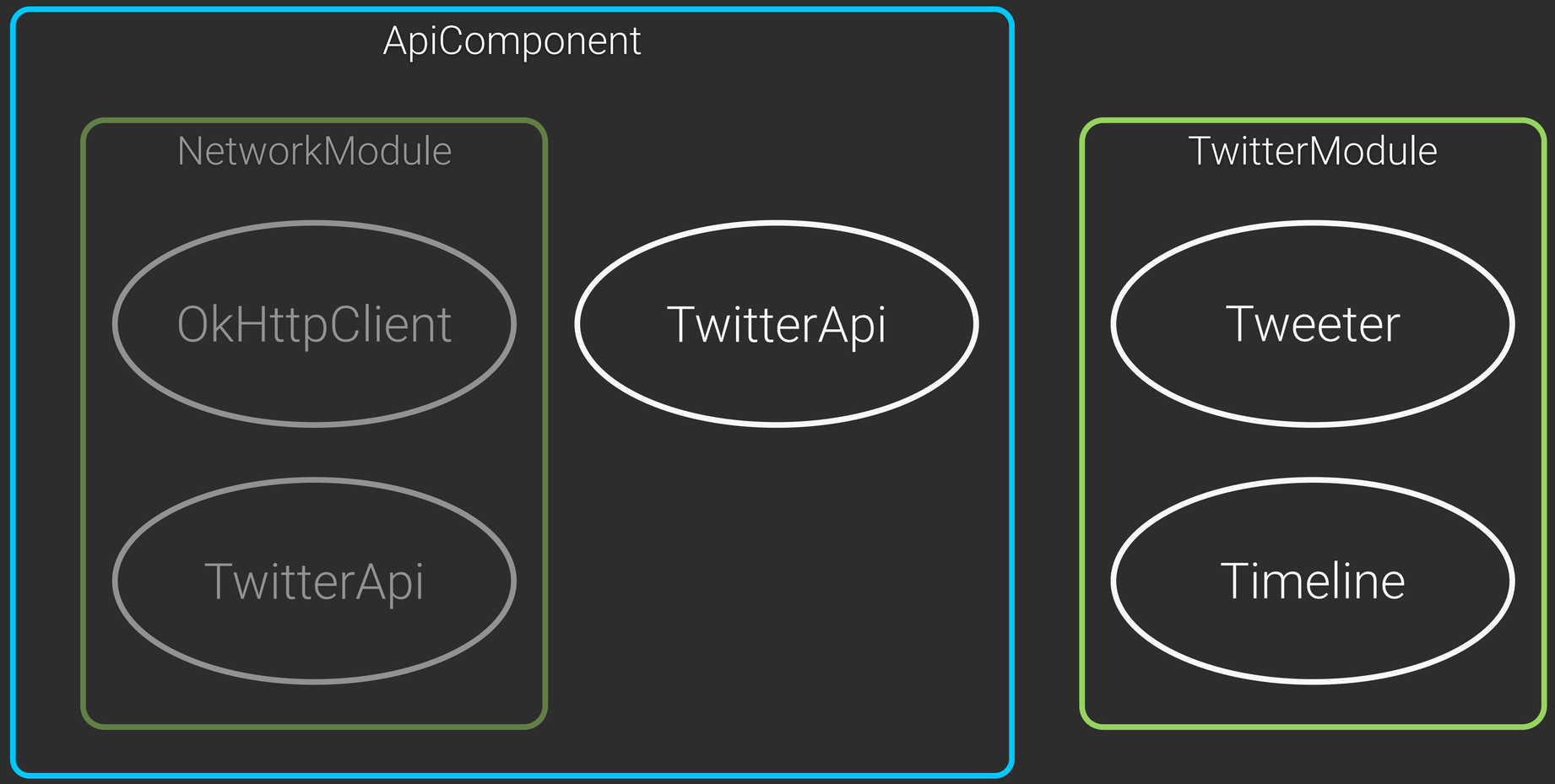

NetworkModule
OkHttpClient
TwitterApi

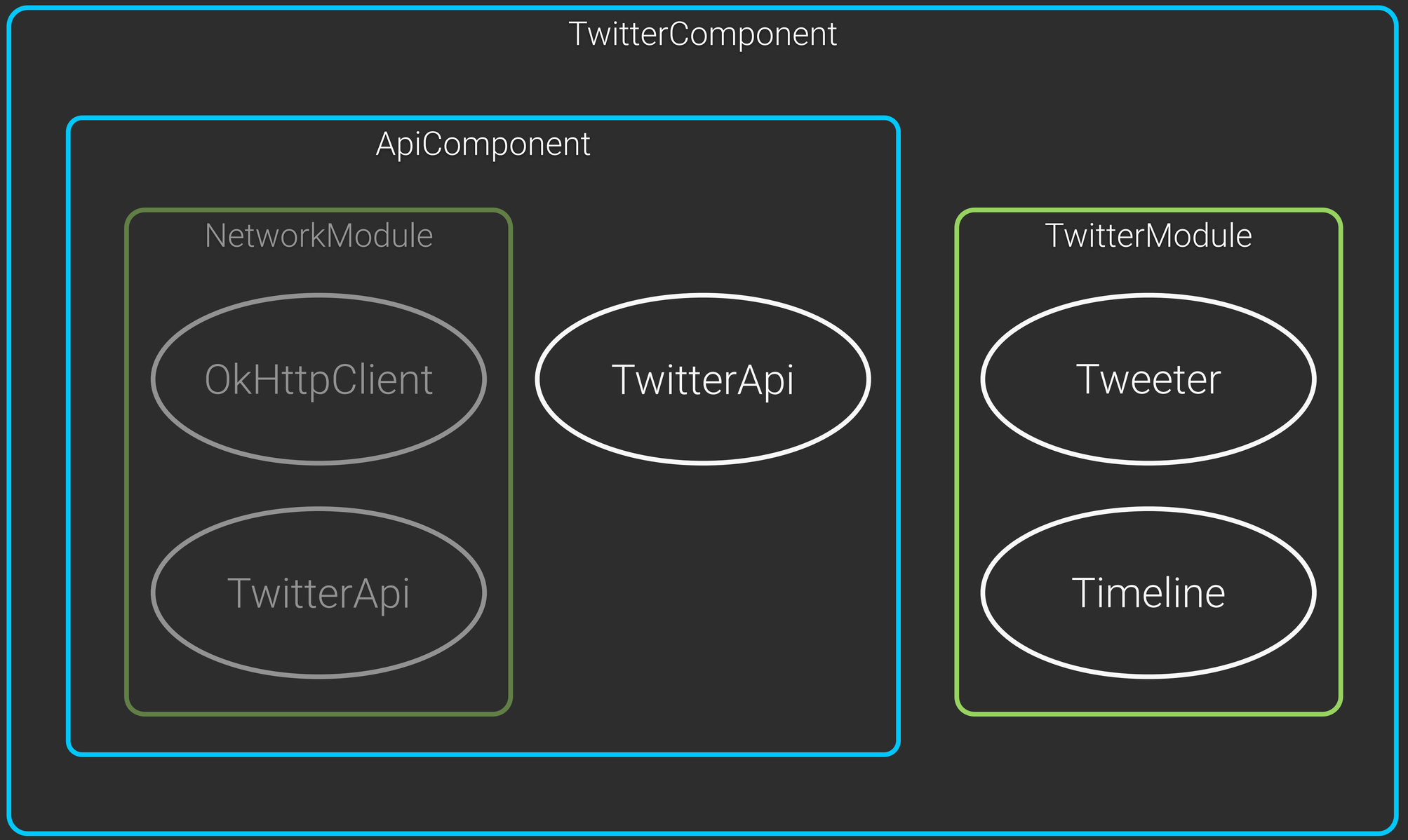

ApiComponent
NetworkModule
OkHttpClient
TwitterApi

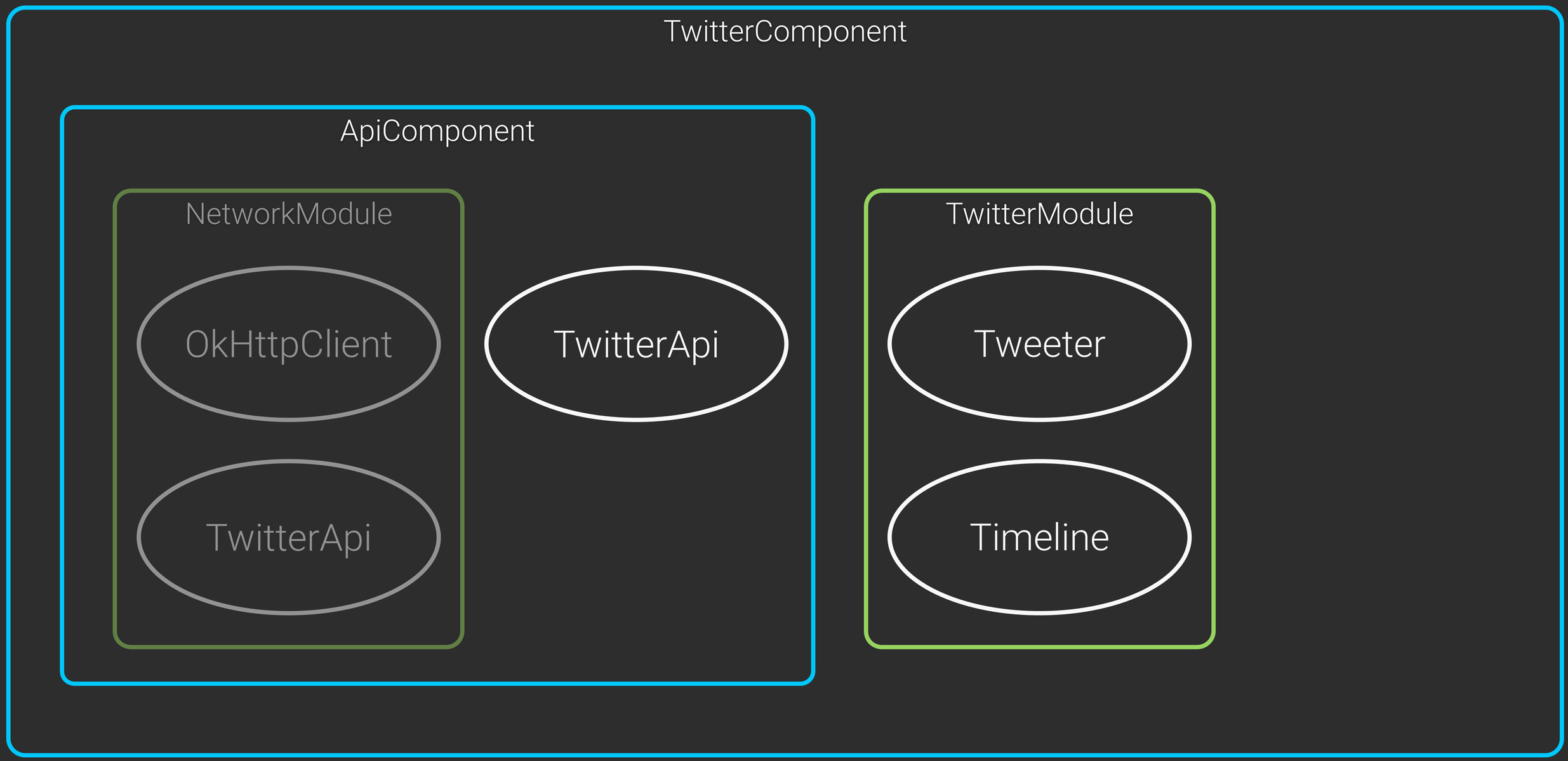

ApiComponent
NetworkModule
OkHttpClient
TwitterApi

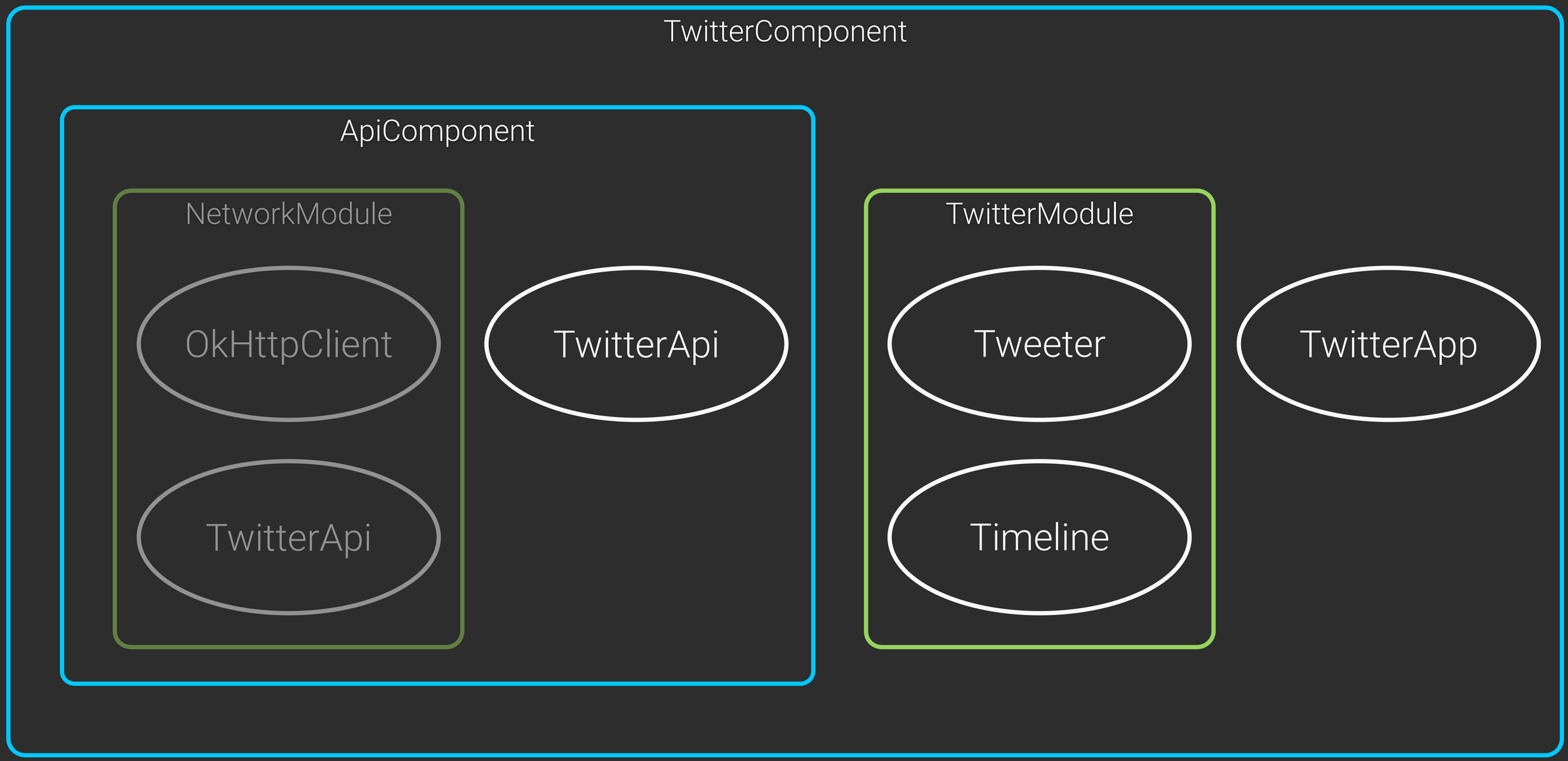

ApiComponent
NetworkModule
OkHttpClient
TwitterApi
TwitterApi

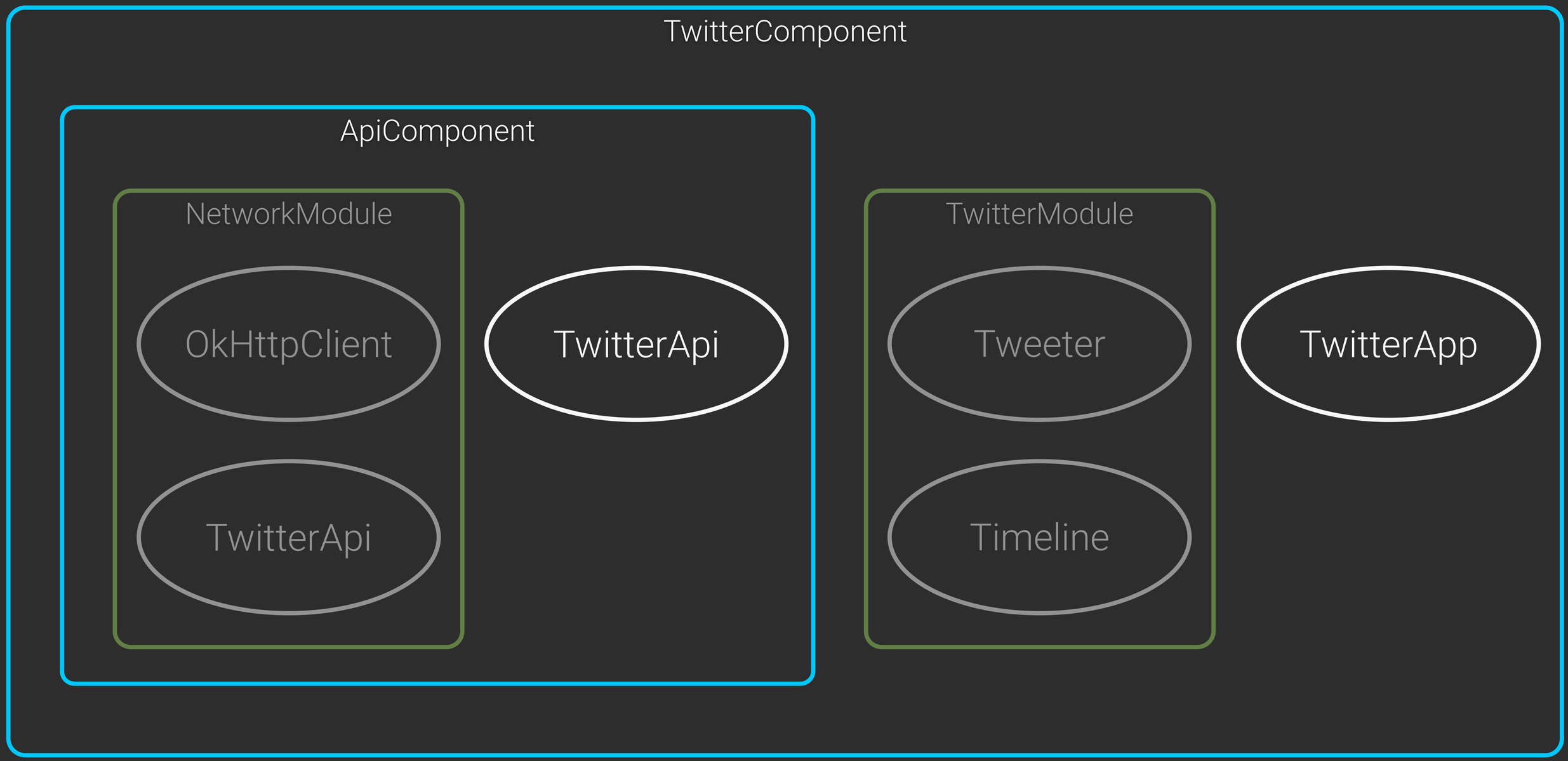

ApiComponent
NetworkModule
OkHttpClient
TwitterApi
TwitterApi

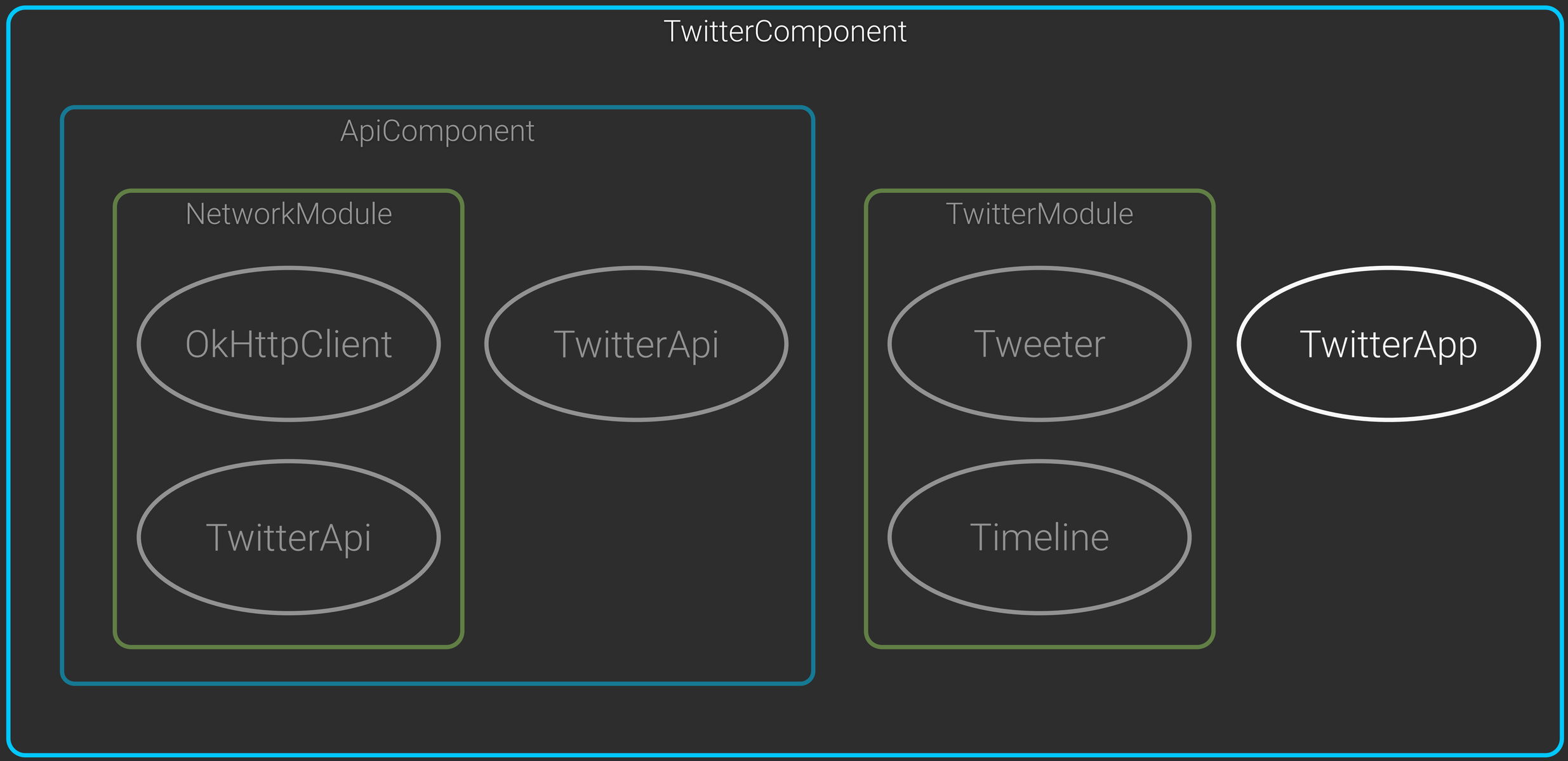

ApiComponent
NetworkModule
OkHttpClient
TwitterApi
TwitterApi
TwitterModule
Tweeter
Timeline

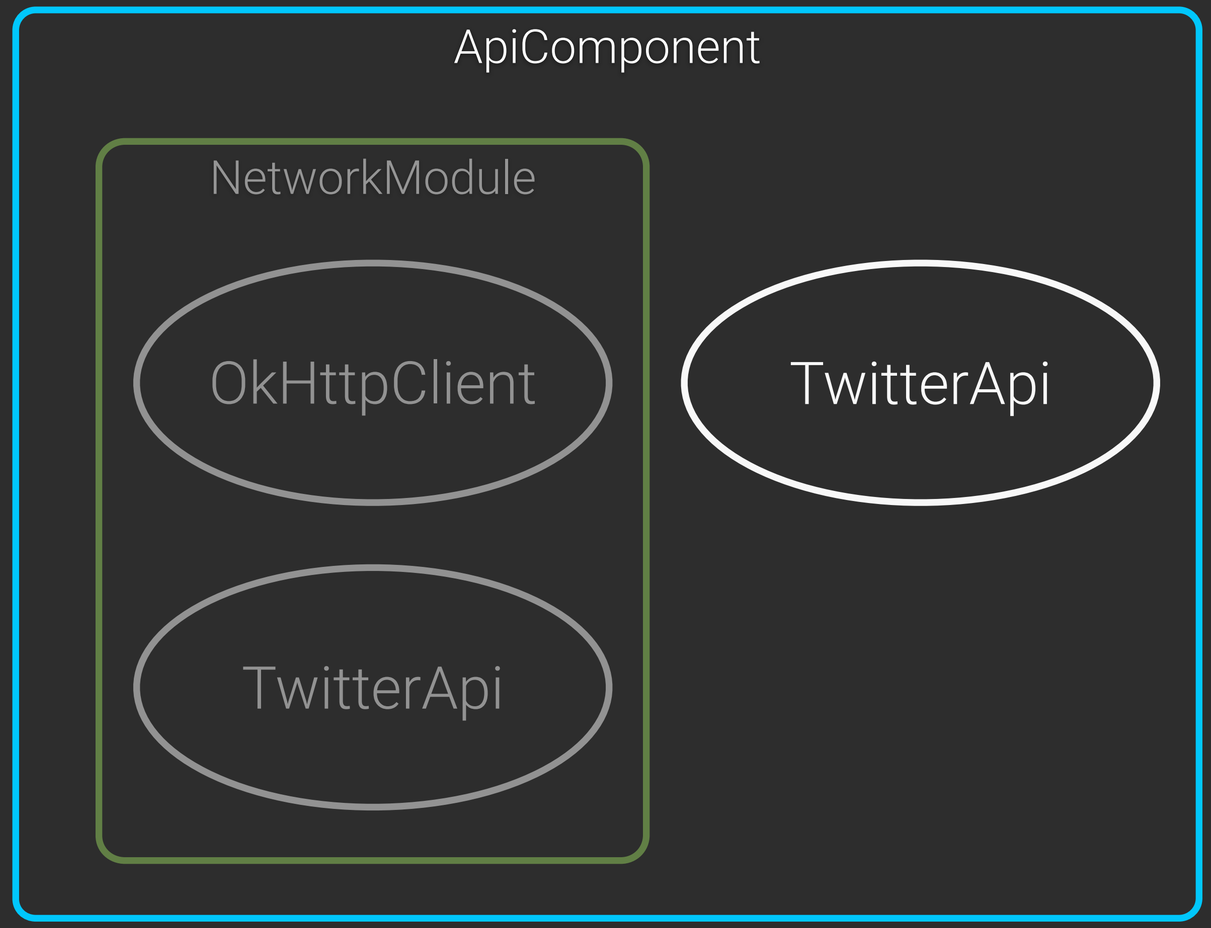

TwitterComponent
ApiComponent
NetworkModule
OkHttpClient
TwitterApi
TwitterApi
TwitterModule
Tweeter
Timeline

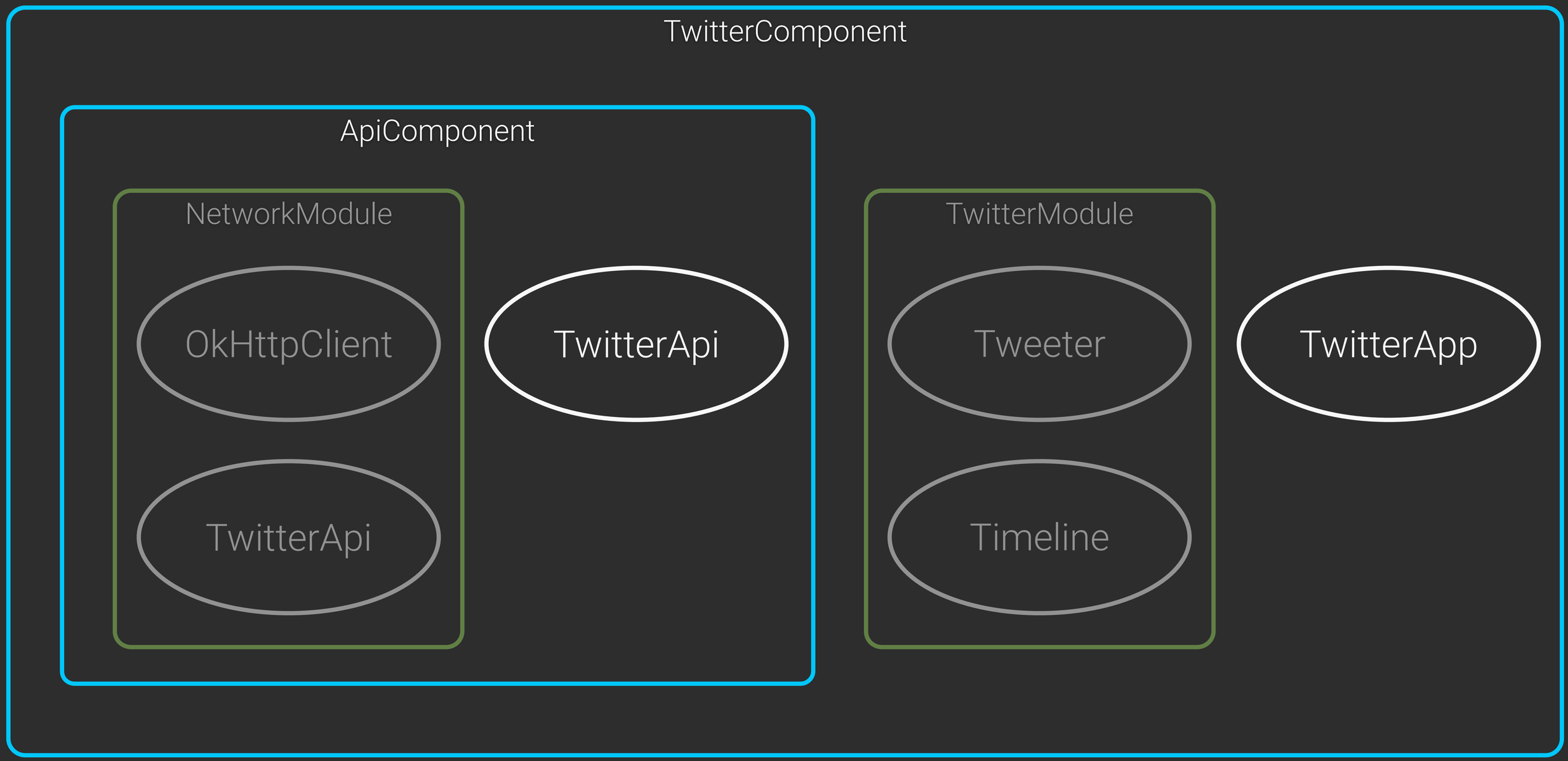

TwitterComponent
ApiComponent
NetworkModule
OkHttpClient
TwitterApi
TwitterApi
TwitterModule
Tweeter
Timeline


TwitterComponent
ApiComponent
NetworkModule
OkHttpClient
TwitterApi
TwitterApi
TwitterModule
Tweeter
Timeline
TwitterApp


TwitterComponent
ApiComponent
NetworkModule
OkHttpClient
TwitterApi
TwitterApi
TwitterModule
Tweeter
Timeline
TwitterApp

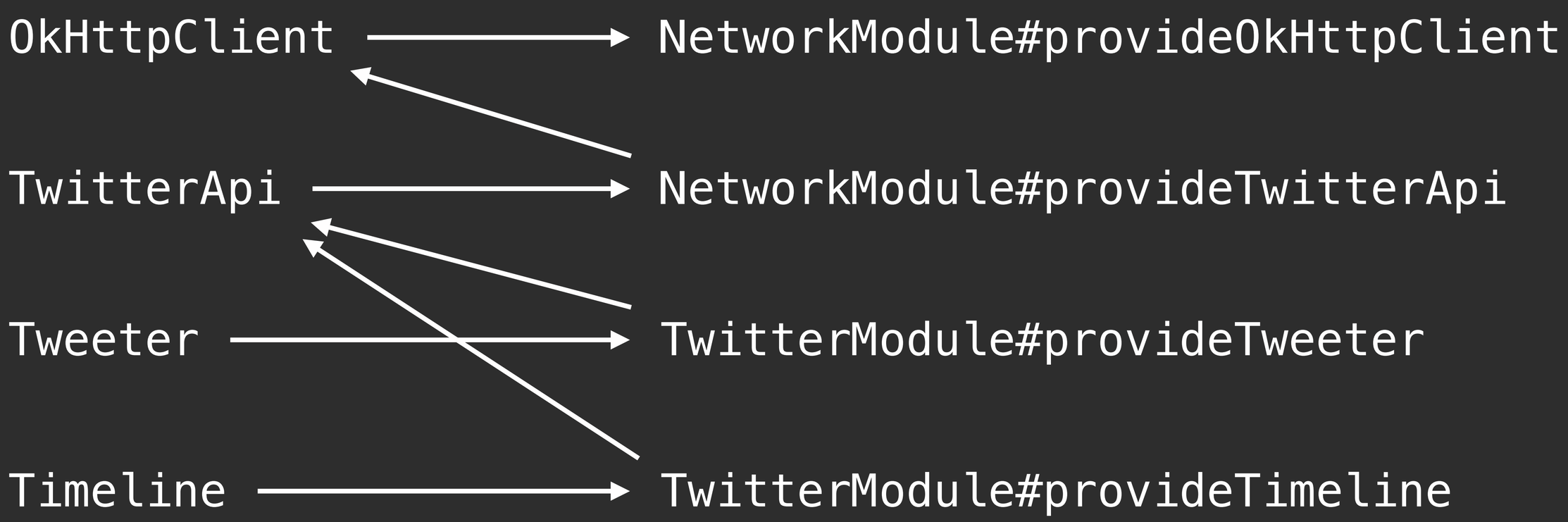

TwitterComponent
ApiComponent
NetworkModule
OkHttpClient
TwitterApi
TwitterApi
TwitterModule
Tweeter
Timeline
TwitterApp


ApiComponent
NetworkModule
OkHttpClient
TwitterApi
TwitterApi


TwitterComponent
ApiComponent
NetworkModule
OkHttpClient
TwitterApi
TwitterApi
TwitterModule
Tweeter
Timeline
TwitterApp

Under The Hood

Under The Hood



```java
@Module
public class NetworkModule {
  @Provides @Singleton
  OkHttpClient provideOkHttpClient() {
    return new OkHttpClient();
  }

  @Provides @Singleton
  TwitterApi provideTwitterApi(OkHttpClient client) {
    return new TwitterApi(client);
  }
}
```

```
public class NetworkModule {
  OkHttpClient provideOkHttpClient() {}
  TwitterApi provideTwitterApi(OkHttpClient) {}
}
```

```
public final class NetworkModule_ProvideOkHttpClientFactory
    implements Factory<OkHttpClient> {

  private final NetworkModule module;

  public NetworkModule_ProvideOkHttpClientFactory(NetworkModule module) {
    this.module = module;
  }

  @Override
  public OkHttpClient get() {
    return module.provideOkHttpClient();
  }
}
```

```
public class NetworkModule {
  OkHttpClient provideOkHttpClient() {}
  TwitterApi provideTwitterApi(OkHttpClient) {}
}
```

```
public final class NetworkModule_ProvideOkHttpClientFactory
    implements Factory<OkHttpClient> {

  private final NetworkModule module;

  public NetworkModule_ProvideOkHttpClientFactory(NetworkModule module) {
    this.module = module;
  }

  @Override
  public OkHttpClient get() {
    return module.provideOkHttpClient();
  }
}
```

```
public class NetworkModule {
  OkHttpClient provideOkHttpClient() {}
  TwitterApi provideTwitterApi(OkHttpClient) {}
}
```

```
public final class NetworkModule_ProvideOkHttpClientFactory
    implements Factory<OkHttpClient> {

  private final NetworkModule module;

  public NetworkModule_ProvideOkHttpClientFactory(NetworkModule module) {
    this.module = module;
  }

  @Override
  public OkHttpClient get() {
    return module.provideOkHttpClient();
  }
}
```

```
public class NetworkModule {
  OkHttpClient provideOkHttpClient() {}
  TwitterApi provideTwitterApi(OkHttpClient) {}
}
```

```
public final class NetworkModule_ProvideOkHttpClientFactory
    implements Factory<OkHttpClient> {

  private final NetworkModule module;

  public NetworkModule_ProvideOkHttpClientFactory(NetworkModule module) {
    this.module = module;
  }

  @Override
  public OkHttpClient get() {
    return module.provideOkHttpClient();
  }
}
```

```
public class NetworkModule {
  OkHttpClient provideOkHttpClient() {}
  TwitterApi provideTwitterApi(OkHttpClient) {}
}
```

```
public final class NetworkModule_ProvideOkHttpClientFactory {
  public OkHttpClient get() {
    return module.provideOkHttpClient();
  }
}
```

```
public final class NetworkModule_ProvideTwitterApiFactory
    implements Factory<TwitterApi> {

  private final NetworkModule module;
  private final Provider<OkHttpClient> clientProvider;

  public NetworkModule_ProvideTwitterApiFactory(
      NetworkModule module, Provider<OkHttpClient> clientProvider) {
    this.module = module;
    this.clientProvider = clientProvider;
  }

  @Override
  public TwitterApi get() {
    return module.provideTwitterApi(clientProvider.get());
  }
}
```

```
public class NetworkModule {
  OkHttpClient provideOkHttpClient() {}
  TwitterApi provideTwitterApi(OkHttpClient) {}
}
```

```
public final class NetworkModule_ProvideOkHttpClientFactory {
  public OkHttpClient get() {
    return module.provideOkHttpClient();
  }
}
```

```
public final class NetworkModule_ProvideTwitterApiFactory
    implements Factory<TwitterApi> {

  private final NetworkModule module;
  private final Provider<OkHttpClient> clientProvider;

  public NetworkModule_ProvideTwitterApiFactory(
      NetworkModule module, Provider<OkHttpClient> clientProvider) {
    this.module = module;
    this.clientProvider = clientProvider;
  }

  @Override
  public TwitterApi get() {
    return module.provideTwitterApi(clientProvider.get());
  }
}
```

```
public class NetworkModule {
  OkHttpClient provideOkHttpClient() {}
  TwitterApi provideTwitterApi(OkHttpClient) {}
}
```

```
public final class NetworkModule_ProvideOkHttpClientFactory {
  public OkHttpClient get() {
    return module.provideOkHttpClient();
  }
}
```

```
public final class NetworkModule_ProvideTwitterApiFactory
    implements Factory<TwitterApi> {

  private final NetworkModule module;
  private final Provider<OkHttpClient> clientProvider;

  public NetworkModule_ProvideTwitterApiFactory(
      NetworkModule module, Provider<OkHttpClient> clientProvider) {
    this.module = module;
    this.clientProvider = clientProvider;
  }

  @Override
  public TwitterApi get() {
    return module.provideTwitterApi(clientProvider.get());
  }
}
```

```
@Singleton
@Component(modules = NetworkModule.class)
interface ApiComponent {
  TwitterApi api();
}
```

```
interface ApiComponent {
  TwitterApi api();
}
```

```
public final class Dagger_ApiComponent implements ApiComponent {
  // ...

  public static final class Builder {
    private NetworkModule networkModule;

    public ApiComponent build() {
      if (networkModule == null) networkModule = new NetworkModule();
      return new Dagger_ApiComponent(this);
    }

    public Builder networkModule(NetworkModule networkModule) {
      if (networkModule == null) throw new NullPointerException();
      this.networkModule = networkModule;
      return this;
    }
  }
}
```

```
interface ApiComponent {
  TwitterApi api();
}
```

```
public final class Dagger_ApiComponent implements ApiComponent {
  // ...

  public static final class Builder {
    private NetworkModule networkModule;

    public ApiComponent build() {
      if (networkModule == null) networkModule = new NetworkModule();
      return new Dagger_ApiComponent(this);
    }

    public Builder networkModule(NetworkModule networkModule) {
      if (networkModule == null) throw new NullPointerException();
      this.networkModule = networkModule;
      return this;
    }
  }
}
```

```
interface ApiComponent {
  TwitterApi api();
}
```

```
public final class Dagger_ApiComponent implements ApiComponent {









}
```

```
interface ApiComponent {
  TwitterApi api();
}
```

```
public final class Dagger_ApiComponent implements ApiComponent {
  private Provider<OkHttpClient> provideOkHttpClientProvider;
  private Provider<TwitterApi> provideTwitterApiProvider;

  private Dagger_ApiComponent(Builder builder) {
    NetworkModule networkModule = builder.networkModule;
    provideOkHttpClientProvider = ScopedProvider.create(
        new NetworkModule_ProvideOkHttpClientFactory(networkModule));
    provideTwitterApiProvider = ScopedProvider.create(
        new NetworkModule_ProvideTwitterApiFactory(networkModule,
          provideOkHttpClientProvider));
  }

  @Override
  public TwitterApi api() {
    return provideTwitterApiProvider.get();
  }
```

```
interface ApiComponent {
  TwitterApi api();
}
```

```
public final class Dagger_ApiComponent implements ApiComponent {
  private Provider<OkHttpClient> provideOkHttpClientProvider;
  private Provider<TwitterApi> provideTwitterApiProvider;

  private Dagger_ApiComponent(Builder builder) {
    NetworkModule networkModule = builder.networkModule;
    provideOkHttpClientProvider = ScopedProvider.create(
        new NetworkModule_ProvideOkHttpClientFactory(networkModule));
    provideTwitterApiProvider = ScopedProvider.create(
        new NetworkModule_ProvideTwitterApiFactory(networkModule,
          provideOkHttpClientProvider));
  }

  @Override
  public TwitterApi api() {
    return provideTwitterApiProvider.get();
  }
```

```
interface ApiComponent {
  TwitterApi api();
}
```

```
public final class Dagger_ApiComponent implements ApiComponent {
  private Provider<OkHttpClient> provideOkHttpClientProvider;
  private Provider<TwitterApi> provideTwitterApiProvider;

  private Dagger_ApiComponent(Builder builder) {
    NetworkModule networkModule = builder.networkModule;
    provideOkHttpClientProvider = ScopedProvider.create(
        new NetworkModule_ProvideOkHttpClientFactory(networkModule));
    provideTwitterApiProvider = ScopedProvider.create(
        new NetworkModule_ProvideTwitterApiFactory(networkModule,
          provideOkHttpClientProvider));
  }

  @Override
  public TwitterApi api() {
    return provideTwitterApiProvider.get();
  }
```

```
interface ApiComponent {
  TwitterApi api();
}
```

```
public final class Dagger_ApiComponent implements ApiComponent {
  private Provider<OkHttpClient> provideOkHttpClientProvider;
  private Provider<TwitterApi> provideTwitterApiProvider;

  private Dagger_ApiComponent(Builder builder) {
    NetworkModule networkModule = builder.networkModule;
    provideOkHttpClientProvider = ScopedProvider.create(
        new NetworkModule_ProvideOkHttpClientFactory(networkModule));
    provideTwitterApiProvider = ScopedProvider.create(
        new NetworkModule_ProvideTwitterApiFactory(networkModule,
          provideOkHttpClientProvider));
  }

  @Override
  public TwitterApi api() {
    return provideTwitterApiProvider.get();
  }
```

```
interface ApiComponent {
  TwitterApi api();
}
```

```
public final class Dagger_ApiComponent implements ApiComponent {
  private Provider<OkHttpClient> provideOkHttpClientProvider;
  private Provider<TwitterApi> provideTwitterApiProvider;

  private Dagger_ApiComponent(Builder builder) {
    NetworkModule networkModule = builder.networkModule;
    provideOkHttpClientProvider = ScopedProvider.create(
        new NetworkModule_ProvideOkHttpClientFactory(networkModule));
    provideTwitterApiProvider = ScopedProvider.create(
        new NetworkModule_ProvideTwitterApiFactory(networkModule,
          provideOkHttpClientProvider));
  }

  @Override
  public TwitterApi api() {
    return provideTwitterApiProvider.get();
  }
```

```
interface ApiComponent {
  TwitterApi api();
}
```

```
public final class Dagger_ApiComponent implements ApiComponent {
  private Provider<OkHttpClient> provideOkHttpClientProvider;
  private Provider<TwitterApi> provideTwitterApiProvider;

  private Dagger_ApiComponent(Builder builder) {
    NetworkModule networkModule = builder.networkModule;
    provideOkHttpClientProvider = ScopedProvider.create(
        new NetworkModule_ProvideOkHttpClientFactory(networkModule));
    provideTwitterApiProvider = ScopedProvider.create(
        new NetworkModule_ProvideTwitterApiFactory(networkModule,
          provideOkHttpClientProvider));
  }

  @Override
  public TwitterApi api() {
    return provideTwitterApiProvider.get();
  }
```

```java
@Module
public class TwitterModule {
  private final String user;

  public TwitterModule(String user) {
    this.user = user;
  }

  @Provides
  Tweeter provideTweeter(TwitterApi api) {
    return new Tweeter(api, user);
  }

  @Provides
  Timeline provideTimeline(TwitterApi api) {
    return new Timeline(api, user);
  }
}
```

```java
@Component(
    dependencies = ApiComponent.class,
    modules = TwitterModule.class
)
interface TwitterComponent {
  TwitterApplication app();
}
```

```
public final class TwitterModule_ProvideTweeterFactory
    implements Factory<Tweeter> {

  private final TwitterModule module;
  private final Provider<TwitterApi> apiProvider;

  public TwitterModule_ProvideTweeterFactory(
      TwitterModule module, Provider<TwitterApi> apiProvider) {
    this.module = module;
    this.apiProvider = apiProvider;
  }

  @Override
  public Tweeter get() {
    return module.provideTweeter(apiProvider.get());
  }
}
```

```
public final class TwitterModule_ProvideTimelineFactory
    implements Factory<Timeline> {

  private final TwitterModule module;
  private final Provider<TwitterApi> apiProvider;

  public TwitterModule_ProvideTimelineFactory(
      TwitterModule module, Provider<TwitterApi> apiProvider) {
    this.module = module;
    this.apiProvider = apiProvider;
  }

  @Override
  public Timeline get() {
    return module.provideTimeline(apiProvider.get());
  }
}
```

```
public final class TwitterApplication_Factory
    implements Factory<TwitterApplication> {

  private final Provider<Tweeter> tweeterProvider;
  private final Provider<Timeline> timelineProvider;

  public TwitterApplication_Factory(
      Provider<Tweeter> tweeterProvider,
      Provider<Timeline> timelineProvider) {
    this.tweeterProvider = tweeterProvider;
    this.timelineProvider = timelineProvider;
  }

  @Override
  public TwitterApplication get() {
    return new TwitterApplication(tweeterProvider.get(),
        timelineProvider.get());
  }
}
```

```
public static final class Builder {
  private TwitterModule twitterModule;
  private ApiComponent apiComponent;

  public TwitterComponent build() {
    if (twitterModule == null) throw new IllegalStateException("...");
    if (apiComponent == null) throw new IllegalStateException("...");
    return new Dagger_TwitterComponent(this);
  }

  public Builder twitterModule(TwitterModule twitterModule) {
    if (twitterModule == null) throw new NullPointerException("twitterModule");
    this.twitterModule = twitterModule;
    return this;
  }

  public Builder apiComponent(ApiComponent apiComponent) {
    if (apiComponent == null) throw new NullPointerException("apiComponent");
    this.apiComponent = apiComponent;
    return this;
  }
}
```

```
public static final class Builder {
  private TwitterModule twitterModule;
  private ApiComponent apiComponent;

  public TwitterComponent build() {
    if (twitterModule == null) throw new IllegalStateException("...");
    if (apiComponent == null) throw new IllegalStateException("...");
    return new Dagger_TwitterComponent(this);
  }

  public Builder twitterModule(TwitterModule twitterModule) {
    if (twitterModule == null) throw new NullPointerException("twitterModule");
    this.twitterModule = twitterModule;
    return this;
  }

  public Builder apiComponent(ApiComponent apiComponent) {
    if (apiComponent == null) throw new NullPointerException("apiComponent");
    this.apiComponent = apiComponent;
    return this;
  }
}
```

```
public static final class Builder {
  private TwitterModule twitterModule;
  private ApiComponent apiComponent;

  public TwitterComponent build() {
    if (twitterModule == null) throw new IllegalStateException("...");
    if (apiComponent == null) throw new IllegalStateException("...");
    return new Dagger_TwitterComponent(this);
  }

  public Builder twitterModule(TwitterModule twitterModule) {
    if (twitterModule == null) throw new NullPointerException("twitterModule");
    this.twitterModule = twitterModule;
    return this;
  }

  public Builder apiComponent(ApiComponent apiComponent) {
    if (apiComponent == null) throw new NullPointerException("apiComponent");
    this.apiComponent = apiComponent;
    return this;
  }
}
```

```
public final class Dagger_TwitterComponent implements TwitterComponent {
  // ...

  private Dagger_TwitterComponent(Builder builder) {
    twitterModule = builder.twitterModule;
    apiComponent = builder.apiComponent;

    apiProvider = new Factory<TwitterApi>() {
      @Override public TwitterApi get() {
        return apiComponent.api();
      }
    };

    provideTweeterProvider =
        new TwitterModule_ProvideTweeterFactory(twitterModule, apiProvider);
    provideTimelineProvider =
        new TwitterModule_ProvideTimelineFactory(twitterModule, apiProvider);
    twitterApplicationProvider = new TwitterApplication_Factory(
        provideTweeterProvider, provideTimelineProvider);
  }
```

```
public final class Dagger_TwitterComponent implements TwitterComponent {
  // ...

  private Dagger_TwitterComponent(Builder builder) {
    twitterModule = builder.twitterModule;
    apiComponent = builder.apiComponent;

    apiProvider = new Factory<TwitterApi>() {
      @Override public TwitterApi get() {
        return apiComponent.api();
      }
    };

    provideTweeterProvider =
        new TwitterModule_ProvideTweeterFactory(twitterModule, apiProvider);
    provideTimelineProvider =
        new TwitterModule_ProvideTimelineFactory(twitterModule, apiProvider);
    twitterApplicationProvider = new TwitterApplication_Factory(
        provideTweeterProvider, provideTimelineProvider);
  }
```

```
Dagger_ApiComponent.create()
```

```
Dagger_ApiComponent.create()
   new NetworkModule
```

```
Dagger_ApiComponent.create()
  new NetworkModule
  new NetworkModule_ProvideOkHttpClientFactory
  new NetworkModule_ProvideTwitterApiFactory
```

```
Dagger_ApiComponent.create()
  new NetworkModule
  new NetworkModule_ProvideOkHttpClientFactory
  new NetworkModule_ProvideTwitterApiFactory

new TwitterModule("Jake Wharton")
```

```
Dagger_ApiComponent.create()
  new NetworkModule
  new NetworkModule_ProvideOkHttpClientFactory
  new NetworkModule_ProvideTwitterApiFactory

new TwitterModule("Jake Wharton")

Dagger_TwitterComponent.builder()...build()
```

```
Dagger_ApiComponent.create()
  new NetworkModule
  new NetworkModule_ProvideOkHttpClientFactory
  new NetworkModule_ProvideTwitterApiFactory

new TwitterModule("Jake Wharton")

Dagger_TwitterComponent.builder()...build()
  new Factory<TwitterApi>
```

```
Dagger_ApiComponent.create()
  new NetworkModule
  new NetworkModule_ProvideOkHttpClientFactory
  new NetworkModule_ProvideTwitterApiFactory

new TwitterModule("Jake Wharton")

Dagger_TwitterComponent.builder()...build()
  new Factory<TwitterApi>
  new TwitterModule_ProvideTweeterFactory
  new TwitterModule_ProvideTimelineFactory
  new TwitterApplication_Factory
```

```
twitterComponent.app()
```

```
twitterComponent.app()
  twitterApplicationProvider.get()
```

```
twitterComponent.app()
  twitterApplicationProvider.get()
    tweeterProvider.get()
      apiProvider.get()
        apiComponent.api()
```

```
twitterComponent.app()
  twitterApplicationProvider.get()
    tweeterProvider.get()
      apiProvider.get()
        apiComponent.api()
          provideTwitterApiProvider.get()
            clientProvider.get()
              networkModule.provideOkHttpClient()
```

```
twitterComponent.app()
  twitterApplicationProvider.get()
    tweeterProvider.get()
      apiProvider.get()
        apiComponent.api()
          provideTwitterApiProvider.get()
            clientProvider.get()
              networkModule.provideOkHttpClient()    new OkHttpClient()
```

```
twitterComponent.app()
  twitterApplicationProvider.get()
    tweeterProvider.get()
      apiProvider.get()
        apiComponent.api()
          provideTwitterApiProvider.get()
            clientProvider.get()
              networkModule.provideOkHttpClient()    new OkHttpClient()
            networkModule.provideTwitterApi(..)
```

```
twitterComponent.app()
  twitterApplicationProvider.get()
    tweeterProvider.get()
      apiProvider.get()
        apiComponent.api()
          provideTwitterApiProvider.get()
            clientProvider.get()
              networkModule.provideOkHttpClient()    new OkHttpClient()
            networkModule.provideTwitterApi(..)      new TwitterApi(..)
```

```
twitterComponent.app()
  twitterApplicationProvider.get()
    tweeterProvider.get()
      apiProvider.get()
        apiComponent.api()
          provideTwitterApiProvider.get()
            clientProvider.get()
              networkModule.provideOkHttpClient()    new OkHttpClient()
            networkModule.provideTwitterApi(..)      new TwitterApi(..)
    twitterModule.provideTweeter(..)
```

```
twitterComponent.app()
  twitterApplicationProvider.get()
    tweeterProvider.get()
      apiProvider.get()
        apiComponent.api()
          provideTwitterApiProvider.get()
            clientProvider.get()
              networkModule.provideOkHttpClient()    new OkHttpClient()
            networkModule.provideTwitterApi(..)      new TwitterApi(..)
    twitterModule.provideTweeter(..)                 new Tweeter(..)
```

```
twitterComponent.app()
  twitterApplicationProvider.get()
    tweeterProvider.get()
      apiProvider.get()
        apiComponent.api()
          provideTwitterApiProvider.get()
            clientProvider.get()
              networkModule.provideOkHttpClient()    new OkHttpClient()
            networkModule.provideTwitterApi(..)      new TwitterApi(..)
    twitterModule.provideTweeter(..)                 new Tweeter(..)
    timelineProvider.get()
      apiProvider.get()
        apiComponent.api()
```

```
twitterComponent.app()
  twitterApplicationProvider.get()
    tweeterProvider.get()
      apiProvider.get()
        apiComponent.api()
          provideTwitterApiProvider.get()
            clientProvider.get()
              networkModule.provideOkHttpClient()    new OkHttpClient()
            networkModule.provideTwitterApi(..)      new TwitterApi(..)
    twitterModule.provideTweeter(..)                 new Tweeter(..)
    timelineProvider.get()
      apiProvider.get()
        apiComponent.api()
          provideTwitterApiProvider.get()
```

```
twitterComponent.app()
  twitterApplicationProvider.get()
    tweeterProvider.get()
      apiProvider.get()
        apiComponent.api()
          provideTwitterApiProvider.get()
            clientProvider.get()
              networkModule.provideOkHttpClient()    new OkHttpClient()
            networkModule.provideTwitterApi(..)      new TwitterApi(..)
    twitterModule.provideTweeter(..)                 new Tweeter(..)
    timelineProvider.get()
      apiProvider.get()
        apiComponent.api()
          provideTwitterApiProvider.get()
    twitterModule.provideTimeline(..)
```

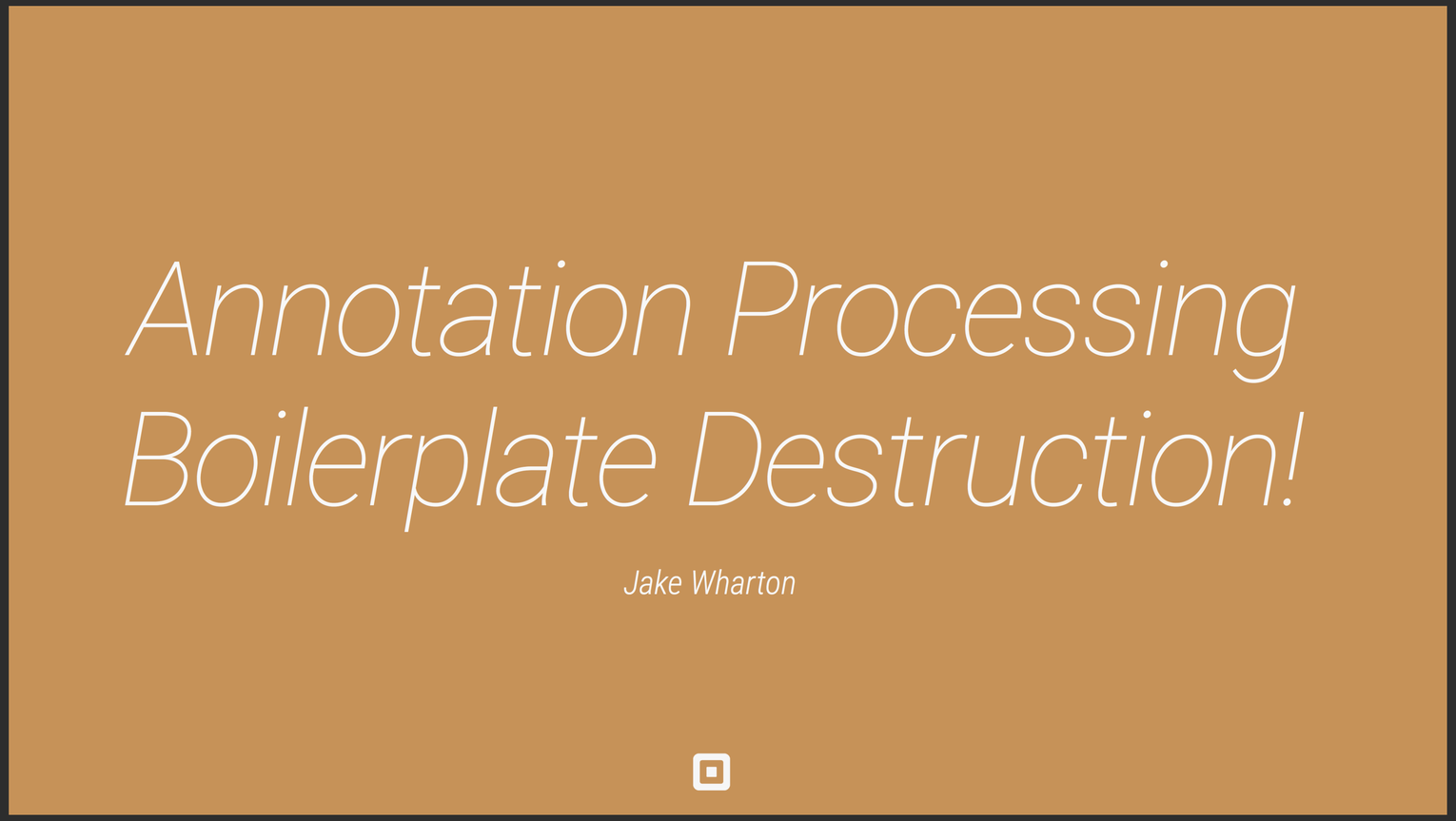
```
twitterComponent.app()
  twitterApplicationProvider.get()
    tweeterProvider.get()
      apiProvider.get()
        apiComponent.api()
          provideTwitterApiProvider.get()
            clientProvider.get()
              networkModule.provideOkHttpClient()    new OkHttpClient()
            networkModule.provideTwitterApi(..)      new TwitterApi(..)
    twitterModule.provideTweeter(..)                 new Tweeter(..)
    timelineProvider.get()
      apiProvider.get()
        apiComponent.api()
          provideTwitterApiProvider.get()
    twitterModule.provideTimeline(..)                new Timeline(..)
```

```
twitterComponent.app()
  twitterApplicationProvider.get()
    tweeterProvider.get()
      apiProvider.get()
        apiComponent.api()
          provideTwitterApiProvider.get()
            clientProvider.get()
              networkModule.provideOkHttpClient()    new OkHttpClient()
            networkModule.provideTwitterApi(..)      new TwitterApi(..)
    twitterModule.provideTweeter(..)                 new Tweeter(..)
    timelineProvider.get()
      apiProvider.get()
        apiComponent.api()
          provideTwitterApiProvider.get()
    twitterModule.provideTimeline(..)                new Timeline(..)
  new TwitterApplication(..)
```

Annotation Processing

Annotation Processing

bit.ly/apt-bd



JSR 330

JSR 330

```
public @interface Inject {}
```

```
public @interface Named {
  String value();
}
```

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

```
public @interface Qualifier {}
```

```
public @interface Scope {}
```

```
public @interface Singleton {}
```

Dagger 1

Dagger 1

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

public @interface Module {
  Class<?>[] injects() default { };
  Class<?>[] staticInjections() default { };
  Class<?>[] includes() default { };
  Class<?> addsTo() default Void.class;
  boolean overrides() default false;
  boolean complete() default true;
  boolean library() default true;
}

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

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

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

Dagger 1

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

public @interface Module {
  Class<?>[] injects() default { };
  Class<?>[] staticInjections() default { };
  Class<?>[] includes() default { };
  Class<?> addsTo() default Void.class;
  boolean overrides() default false;
  boolean complete() default true;
  boolean library() default true;
}

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

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

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

Dagger 2

Dagger 2

```
public @interface Component {
  Class<?>[] modules() default {};
  Class<?>[] dependencies() default {};
}
```

```
public @interface Module {
  Class<?>[] includes() default { };
}
```

```
public @interface Provides {
}
```

```
public @interface MapKey {
  boolean unwrapValue();
}
```

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

Dagger 2

```
public @interface Component {
  Class<?>[] modules() default {};
  Class<?>[] dependencies() default {};
}
```

```
public @interface Module {
  Class<?>[] includes() default { };
}
```

```
public @interface Provides {
}
```

```
public @interface MapKey {
  boolean unwrapValue();
}
```

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

More Concepts

More Concepts

- Set and map bindings.

More Concepts

- Set and map bindings.
- Asynchronous producers.

More Concepts

- Set and map bindings.
- Asynchronous producers.
- Testing and module overrides.

More Concepts

- Set and map bindings.
- Asynchronous producers.
- Testing and module overrides.
- AutoFactory for assisted injection.

github.com/google/dagger

Questions?

Questions?

twitter.com/ jakewharton

google.com/+ jakewharton

jakewharton .com

DEPENDENCY INJECTION WILL RETURN