Asynchronous Dependency Injection

Cong Li sjlicong@gmail.com
Apr/2015

Overview

- Why DI?
- What’s DI?
- How does DI work?
- Asynchronous dependency?
- Proposals
- Design Principles
- What to be expected?

Why DI?

```
public class AdsFeedEngine implements FeedEngine {
    public List<Feed> compute(UserInfo userInfo) {
        FeedLog log = new AdsDatabaseLog();

        List<Feed> feeds = new ArrayList();
        try {
            // Compute code omitted...
            log.logFeedComputation(feeds);
        } catch (FeedComputationException e) {
            log.logFeedComputationException(e);
        }

        return feeds;
    }
}
```

Why DI? (cont'd)

```
public class AdsFeedEngine implements FeedEngine {
    public List<Feed> compute(UserInfo userInfo) {
        FeedLog log = new AdsDatabaseLog();

        List<Feed> feeds = new ArrayList();
        try {
            // Compute code omitted...
            log.logFeedComputation(feeds);
        } catch (FeedComputationException e) {
            log.logFeedComputationException(e);
        }

        return feeds;
    }
}
```

Q: Any problems with this?

Why DI? (cont'd)

```
public class AdsFeedEngine implements FeedEngine {
    public List<Feed> compute(UserInfo userInfo) {
        FeedLog log = new AdsDatabaseLog();

        List<Feed> feeds = new ArrayList();
        try {
            // Compute code omitted...
            log.logFeedComputation(feeds);
        } catch (FeedComputationException e) {
            log.logFeedComputationException(e);
        }

        return feeds;
    }
}
```

Q: Any problems with this?

A: Yes, DEPENDENCY!!!

Why DI? (cont'd)

```
public class AdsFeedEngine implements FeedEngine {
    public List<Feed> compute(UserInfo userInfo) {
        FeedLog log = new AdsDatabaseLog();

        List<Feed> feeds = new ArrayList();
        try {
            // Compute code omitted...
            log.logFeedComputation(feeds);
        } catch (FeedComputationException e) {
            log.logFeedComputationException(e);
        }

        return feeds;
    }
}
```

Q: Any problems with this?

A: Yes, DEPENDENCY!!!

1. What if we want to log to somewhere else for some users?

Why DI? (cont'd)

```
public class AdsFeedEngine implements FeedEngine {
    public List<Feed> compute(UserInfo userInfo) {
        FeedLog log = new AdsDatabaseLog();

        List<Feed> feeds = new ArrayList();
        try {
            // Compute code omitted...
            log.logFeedComputation(feeds);
        } catch (FeedComputationException e) {
            log.logFeedComputationException(e);
        }

        return feeds;
    }
}
```

Q: Any problems with this?

A: Yes, DEPENDENCY!!!

1. What if we want to log to somewhere else for some users?
2. How about testing?

Why DI? (cont'd)

```
public class AdsFeedEngine implements FeedEngine {
    public List<Feed> compute(UserInfo userInfo) {
        FeedLog log = new AdsDatabaseLog();

        List<Feed> feeds = new ArrayList();
        try {
            // Compute code omitted...
            log.logFeedComputation(feeds);
        } catch (FeedComputationException e) {
            log.logFeedComputationException(e);
        }

        return feeds;
    }
}
```

Q: Any problems with this?

A: Yes, DEPENDENCY!!!

1. What if we want to log to somewhere else for some users?
2. How about testing?
3. Focus as most as possible on business logic, as little as possible on others.

What’s DI?

Q: Solution to the dependency problem?

What’s DI? (cont’d)

Q: Solution to the dependency problem?

A: Dependency Injection!!!

What’s DI? (cont’d)

```
public class AdsFeedEngine implements FeedEngine {
    private final FeedLog log;

    public AdsFeedEngine(FeedLog log) {
        this.log = log;
    }

    public List<Feed> compute(UserInfo userInfo) {
        // …
        log.logFeedComputation(feeds);
        // …
        log.logFeedComputationException(e);
        // ...
    }
}
```

How does DI work?

Q: Do you mean it’s change of below?

```
FeedEngine feedEngine =  new AdsFeedEngine(); ->

FeedLog log = new AdsDatabaseLog();
FeedEngine feedEngine =  new AdsFeedEngine(log);
```

How does DI work? (cont'd)

Q: Do you mean it's change of below?

```
FeedEngine feedEngine =  new AdsFeedEngine(); ->

FeedLog log = new AdsDatabaseLog();
FeedEngine feedEngine =  new AdsFeedEngine(log);
```

```
Sucks, it turns out I need:
C1 c1 = new C1();
C2 c2 = new C2();
// …
C999 c999 = new C999();

Engine Engine = new XEngine(c1, c2, …, c999);
```

How does DI work? (cont'd)

Q: Do you mean it's change of below?

A: No. DI should be done purely by framework, but not manually.

How does DI work? (cont'd)

One example:

```
public class AdsFeedEngine implements FeedEngine {
    private final FeedLog log;

    @Inject
    public AdsFeedEngine(FeedLog log) {
        this.log = log;
    }

    public List<Feed> compute(UserInfo userInfo) {
        // …
    }
}
```

How does DI work? (cont'd)

One example:

```
public class AdsFeedEngine implements FeedEngine {
    private final FeedLog log;

    @Inject
    public AdsFeedEngine(FeedLog log) {
        this.log = log;
    }

    public List<Feed> compute(UserInfo userInfo) {
        // …
    }
}
```

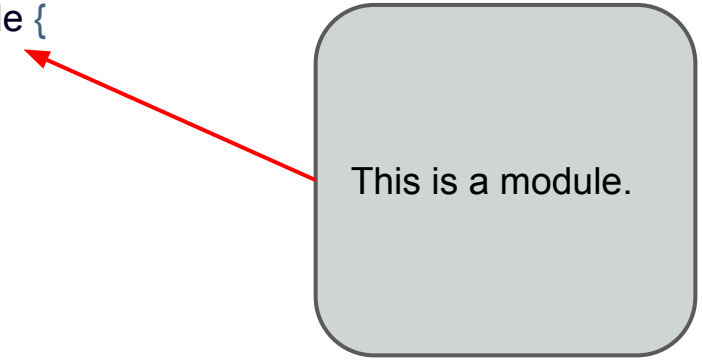
There is no "new" in your code anymore. (Except for one case).

HOORAY~

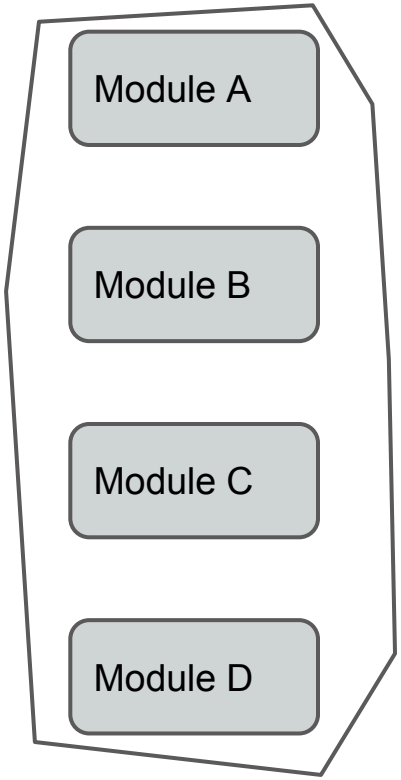
How does DI work? (cont'd)

Example cont'd

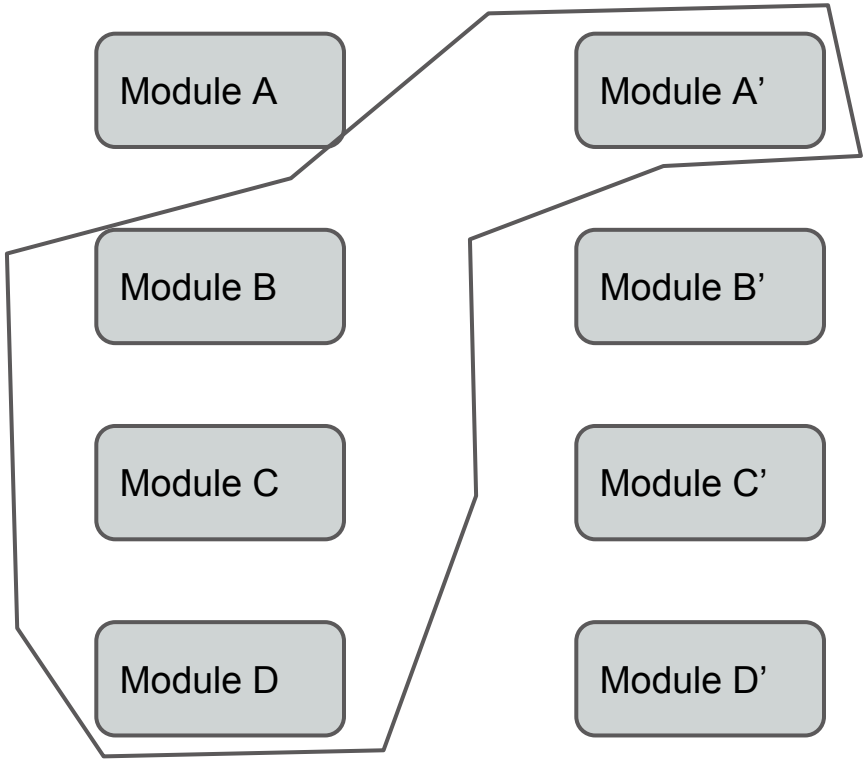
```
public class FeedEngineModule extends AbstractModule {
    @Override
    protected void configure() {
        // ...
    }

    @Provides
    FeedLog provideFeedLog() {
        FeedLog feedLog = new AdsFeedLog();
        // ...
        return feedLog;
    }
}
```



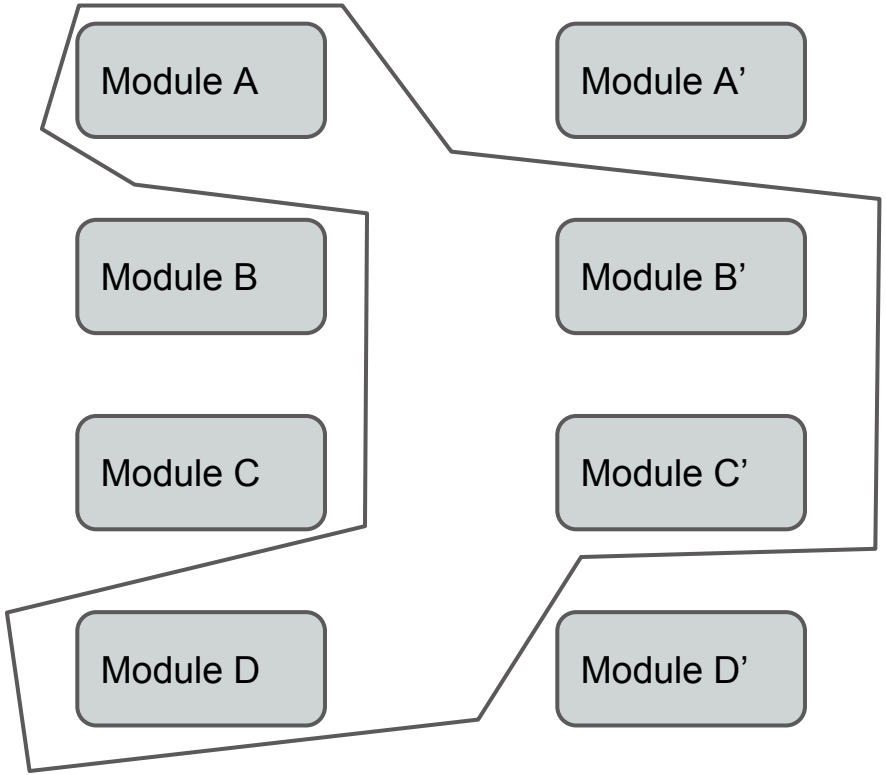
How does DI work? (cont'd)



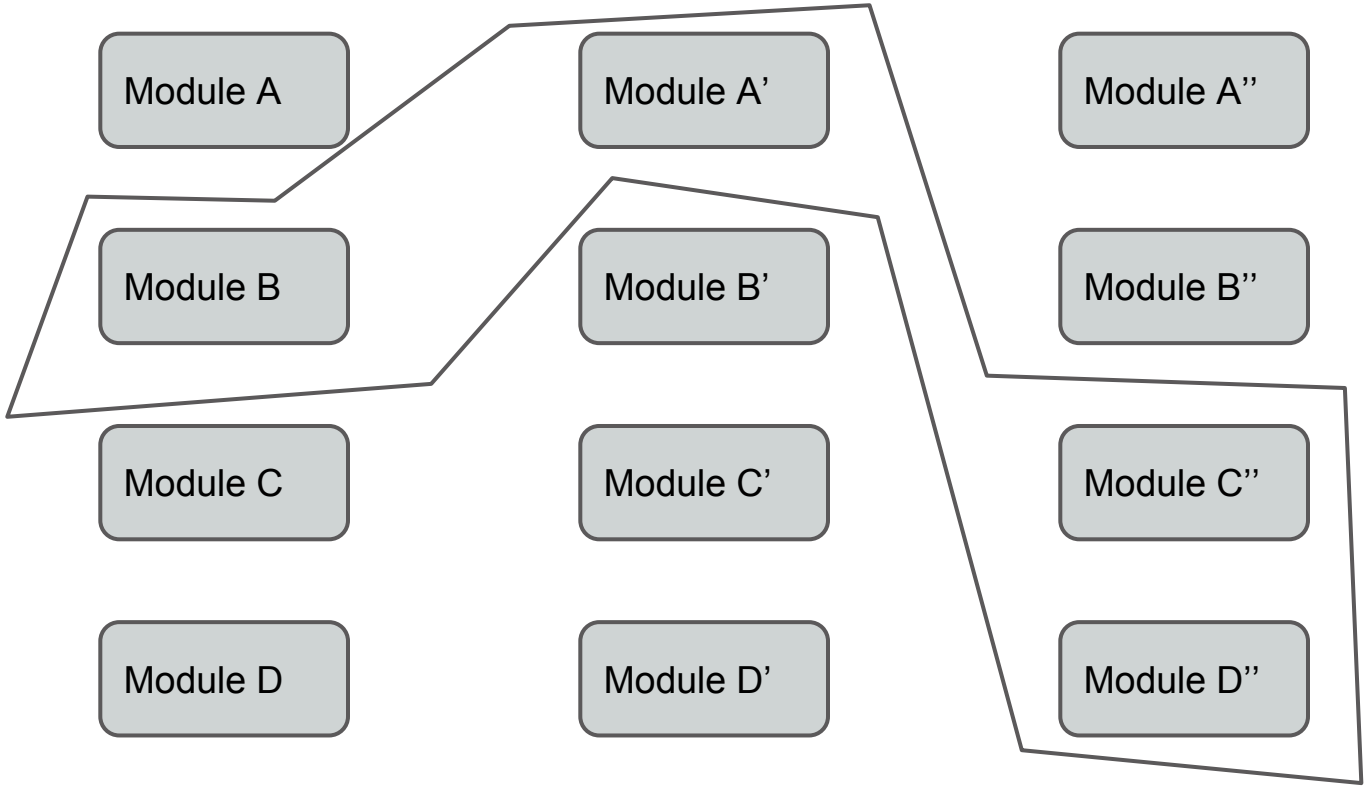
How does DI work? (cont'd)



How does DI work? (cont'd)



How does DI work? (cont'd)



How does DI work? (cont'd)

Frameworks (Java example):

Spring, Guice, Java EE6 CDI, Deduplication etc.

Types (Java example):

Construction, Setter, Interface etc.

Asynchronous Dependency?

```
public class MainStream implements Stream {
    private final UserInfo userInfo;
    private final AdsFeed adsFeed;

    @Inject
    public MainStream(UserInfo userInfo, AdsFeed adsFeed) {
        this.userInfo = userInfo;
        this.adsFeed = adsFeed;
    }

    public List<Item> getStream() {
        adsFeed.compute(userInfo);

        // …
    }
}
```

Asynchronous Dependency? (cont'd)

```
public class MainStream implements Stream {
    private final UserInfo userInfo;
    private final AdsFeed adsFeed;

    @Inject
    public MainStream(UserInfo userInfo, AdsFeed adsFeed) {
        this.userInfo = userInfo;
        this.adsFeed = adsFeed;
    }

    public List<Item> getStream() {
        adsFeed.compute(userInfo);

        // …
    }
}
```

Any problem?

Asynchronous Dependency? (cont'd)

```
public class MainStream implements Stream {
    private final UserInfo userInfo;
    private final AdsFeed adsFeed;

    @Inject
    public MainStream(UserInfo userInfo, AdsFeed adsFeed) {
        this.userInfo = userInfo;
        this.adsFeed = adsFeed;
    }

    public List<Item> getStream() {
        adsFeed.compute(userInfo);

        // …
    }
}
```

What's if UserInfo needs a RPC computation?

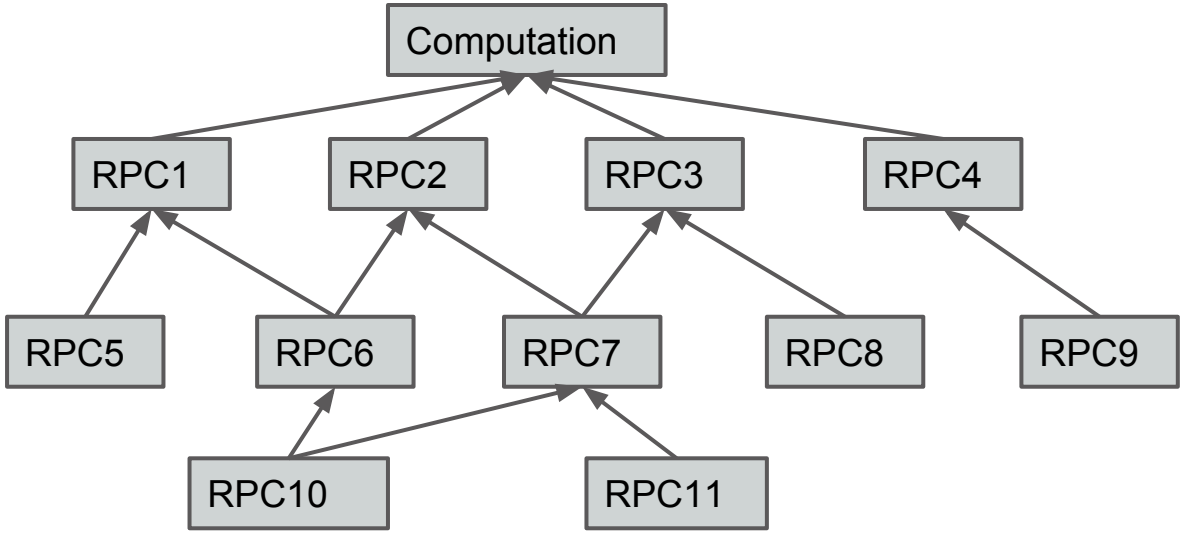
Asynchronous Dependency? (cont'd)

Thread is blocked!!!

Asynchronous Dependency? (cont'd)

Thread is blocked!!!

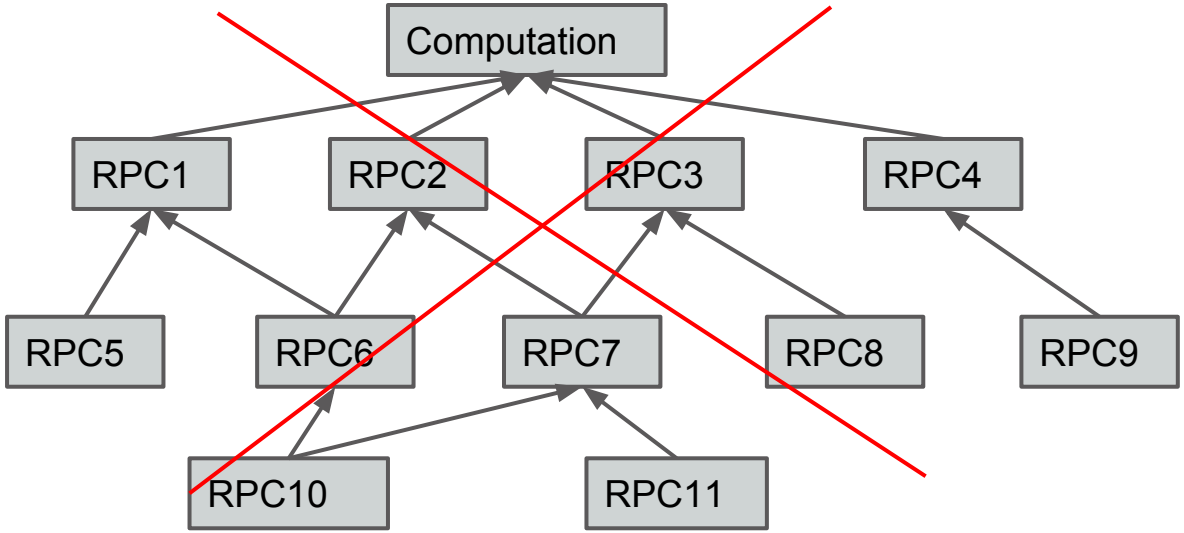
E.g. 11 RPCs with 500ms each may take 5.5 seconds to finish, or even more.



Asynchronous Dependency? (cont'd)

Thread is blocked!!!

E.g. 11 RPCs with 500ms each may take 5.5 seconds to finish, or even more.



You will also need to manage multiple dependencies (scheduling).

Using Future everywhere is a nightmare.

Proposals?

Proposals? (cont'd)

How about asynchronous DI?

Proposals? (cont'd)

How about asynchronous DI?

I.E. I want some framework to do everything above

Proposals? (cont'd)

How about asynchronous DI?

I.E. I want some framework to do everything above that causes me HEADACHES.

Proposals? (cont'd)

Let's recall:

```
public class FeedEngineModule extends AbstractModule {
    @Override
    protected void configure() {
        // ...
    }

    @Provides
    FeedLog provideFeedLog() {
        FeedLog feedLog = new AdsFeedLog();
        // ...
        return feedLog;
    }
}
```

Proposals? (cont'd)

How about:

```
public class FeedEngineModule extends AbstractAsyncModule {
    @Override
    protected void configure() {
        // ...
    }

    @AsyncProvides
    UserInfo provideUserInfo(CookieId cookieId) {
        // ...
        return UserStore.fetchUserInfo(cookieId);
    }
}
```

Design Principles

- All @AsyncProvides must be run in background threads - That’s all we meant.

Design Principles (cont'd)

- All @AsyncProvides must be run in background threads - That's all we meant.
- Thread safety must be considered.

Design Principles (cont'd)

- All @AsyncProvides must be run in background threads - That's all we meant.
- Thread safety must be considered.
- Cache is also very important.

Design Principles (cont'd)

- All @AsyncProvides must be run in background threads - That's all we meant.
- Thread safety must be considered.
- Cache is also very important.
- Non-Cache is equally important.

Design Principles (cont'd)

- All @AsyncProvides must be run in background threads - That's all we meant.
- Thread safety must be considered.
- Cache is also very important.
- Non-Cache is equally important.
- Exception handling must be supported.

Design Principles (cont'd)

- All @AsyncProvides must be run in background threads - That's all we meant.
- Thread safety must be considered.
- Cache is also very important.
- Non-Cache is equally important.
- Exception handling must be supported.
- Visibility.

Design Principles (cont'd)

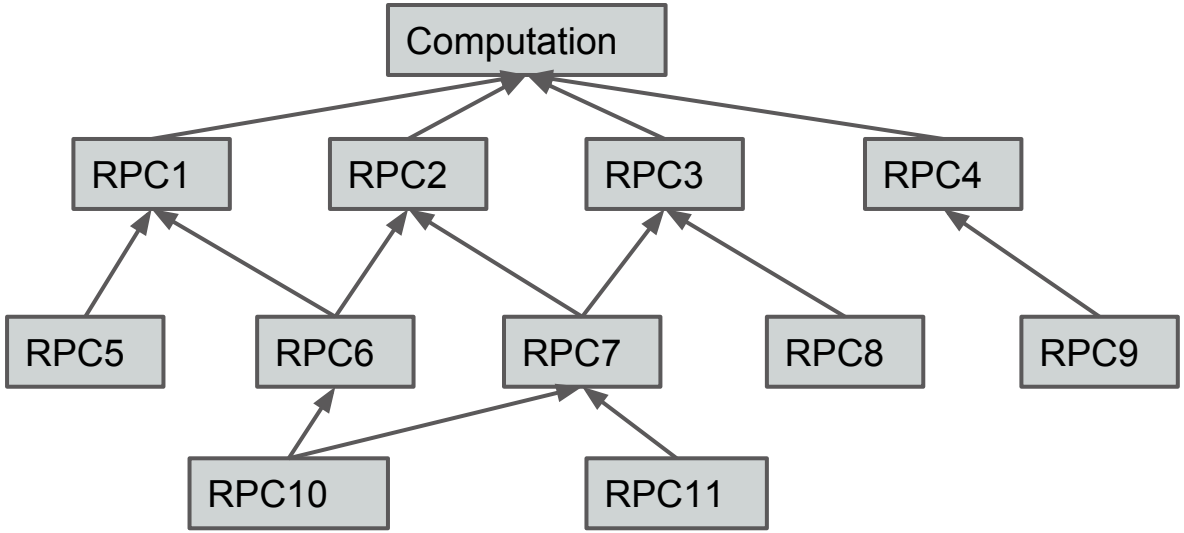
- All @AsyncProvides must be run in background threads - That's all we meant.
- Thread safety must be considered.
- Cache is also very important.
- Non-Cache is equally important.
- Exception handling must be supported.
- Visibility.
- Fire and forget.

What to be expected?

What to be expected? (cont'd)

Recall that -

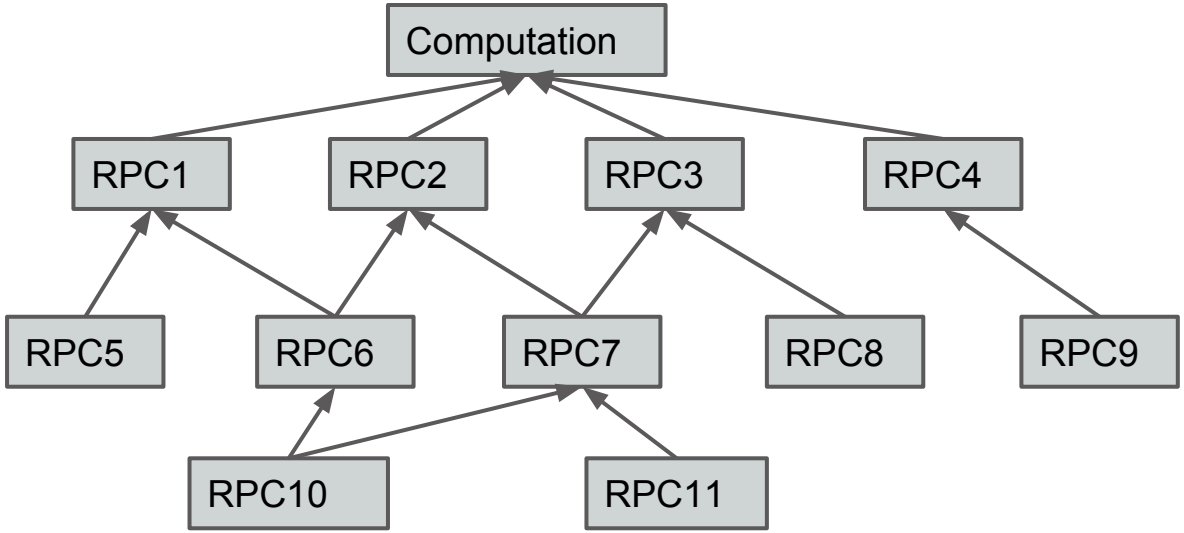
11 RPCs with 500ms each may take 5.5 seconds to finish, or even more.



What to be expected? (cont'd)

Now -

It takes merely 1.5 seconds.



What to be expected? (cont'd)

- Multi threads
- No more handling Future manually

What to be expected? (cont'd)

- Multi threads
- No more handling Future manually
- No more due RPCs

What to be expected? (cont'd)

- Multi threads
- No more handling Future manually
- No more due RPCs
- No more complicated dependency manipulation

What to be expected? (cont'd)

- Multi threads
- No more handling Future manually
- No more due RPCs
- No more complicated dependency manipulation
- Elegant code

What to be expected? (cont'd)

```
public class MainStream implements Stream {
    private final UserInfo userInfo;
    private final AdsFeed adsFeed;

    @Inject
    public MainStream(UserInfo userInfo, AdsFeed adsFeed) {
        this.userInfo = userInfo;
        this.adsFeed = adsFeed;
    }

    public List<Item> getStream() {
        adsFeed.compute(userInfo);

        // …
    }
}
```

THAT'S IT!


THANK
YOU