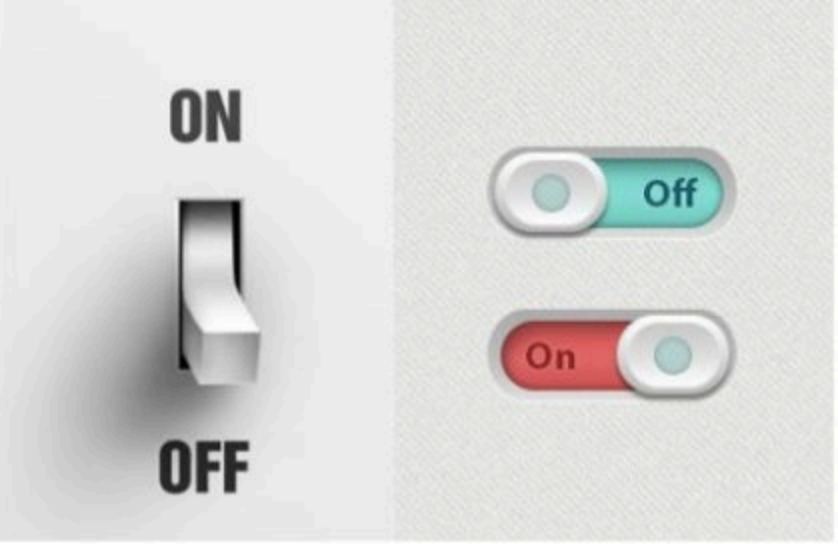
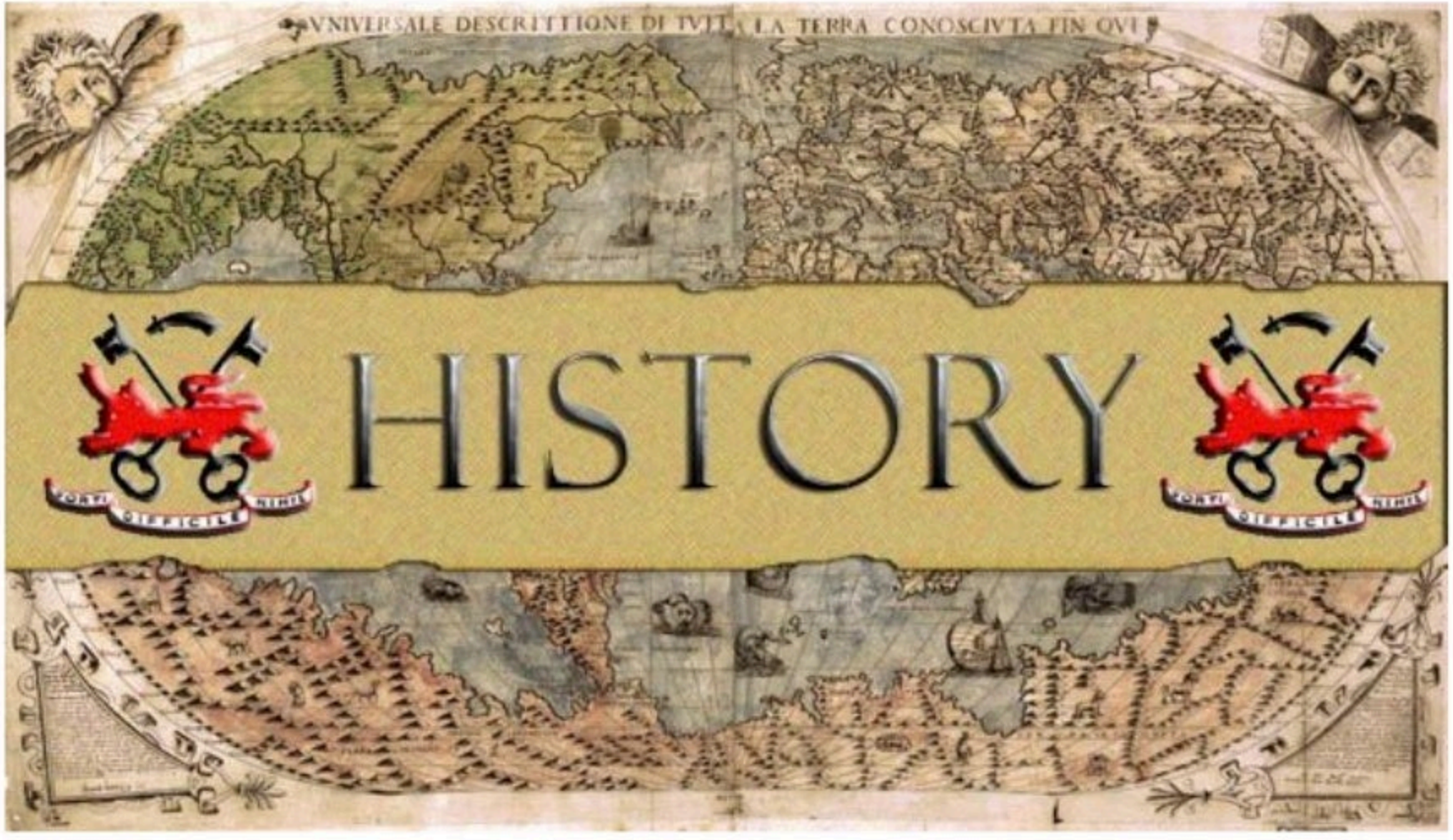
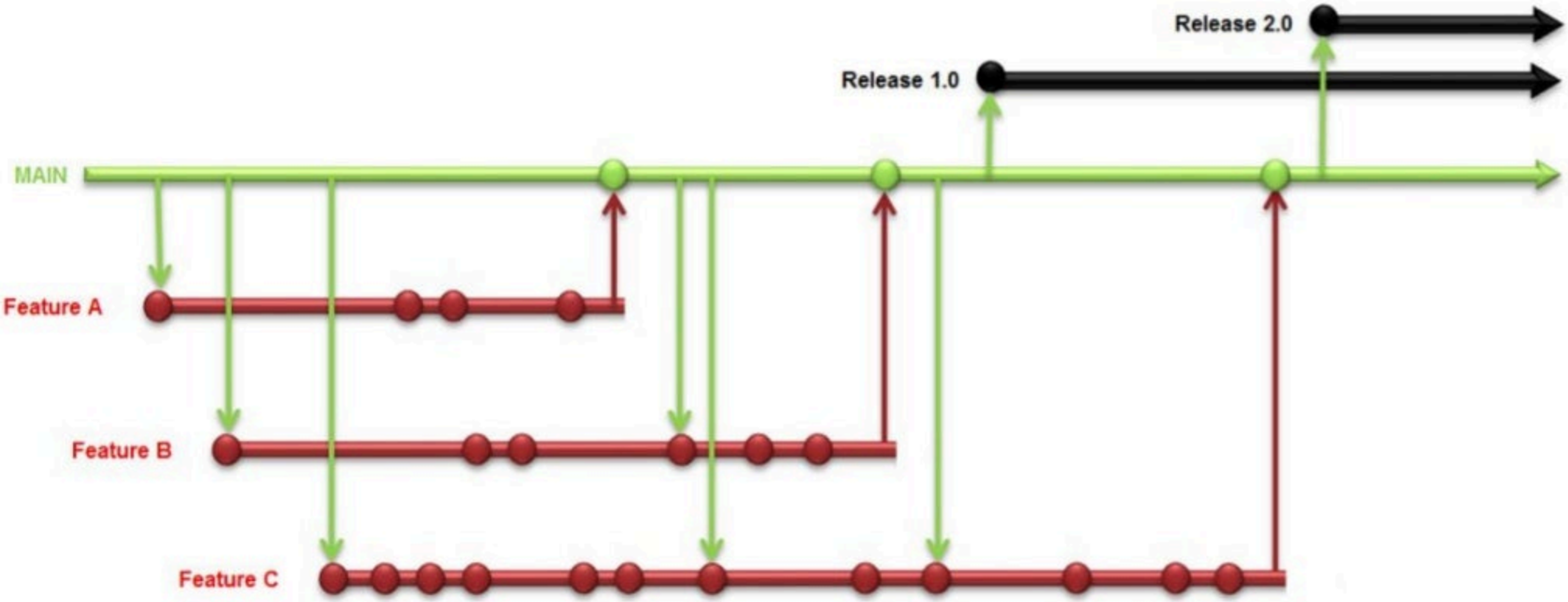


Feature Toggles On Steroids

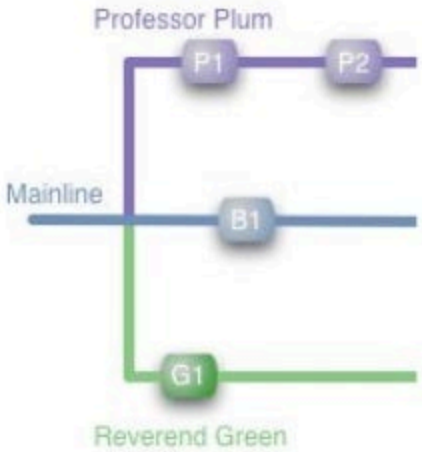

HISTORY

Feature Branches
(aka Branch Based Development)



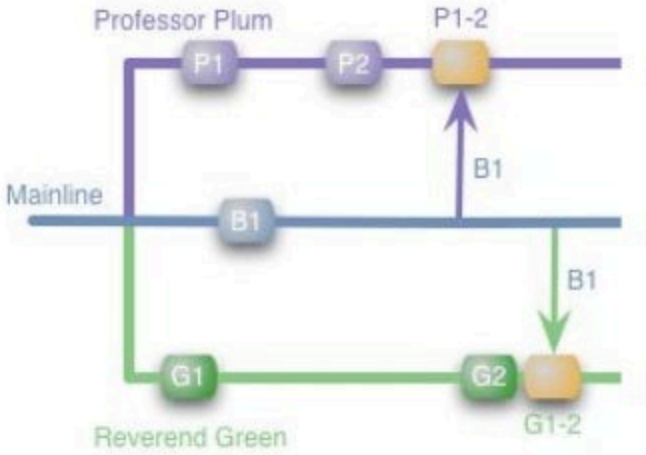
Quelle: http://geekswithblogs.net/Optikal/archive/2013/02/10/152069.aspx

Problem 1: ...



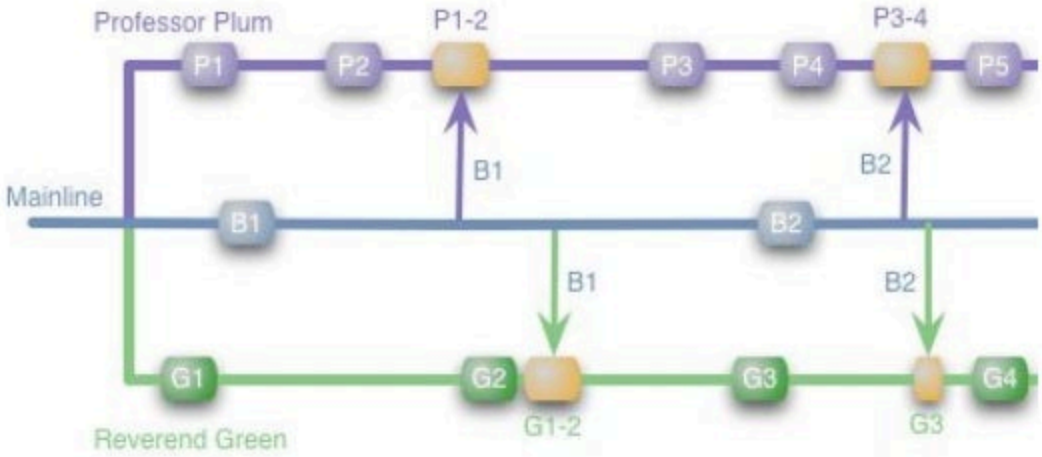
Quelle: http://martinfowler.com/bliki/FeatureBranch.html

Problem 1: ...



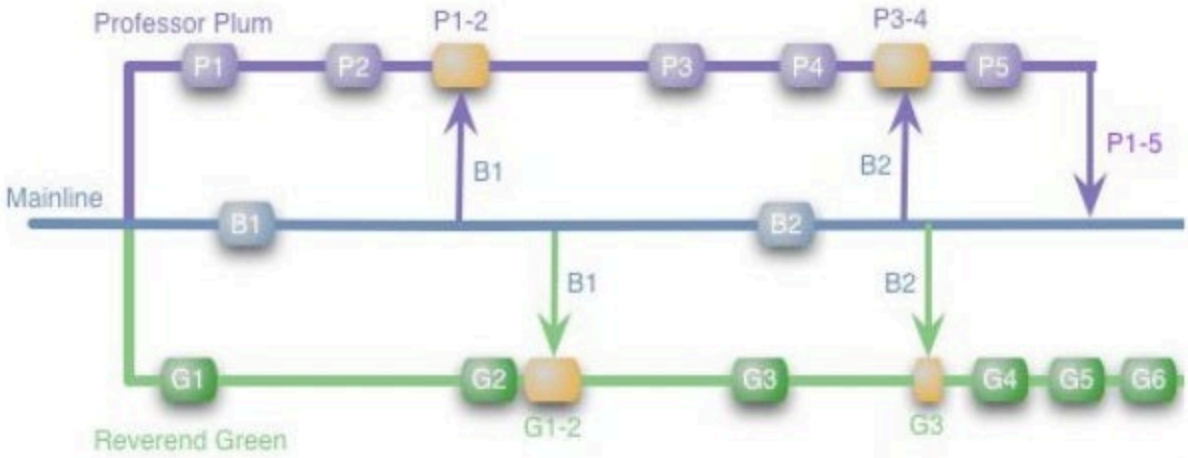
Quelle: http://martinfowler.com/bliki/FeatureBranch.html

Problem 1: ...



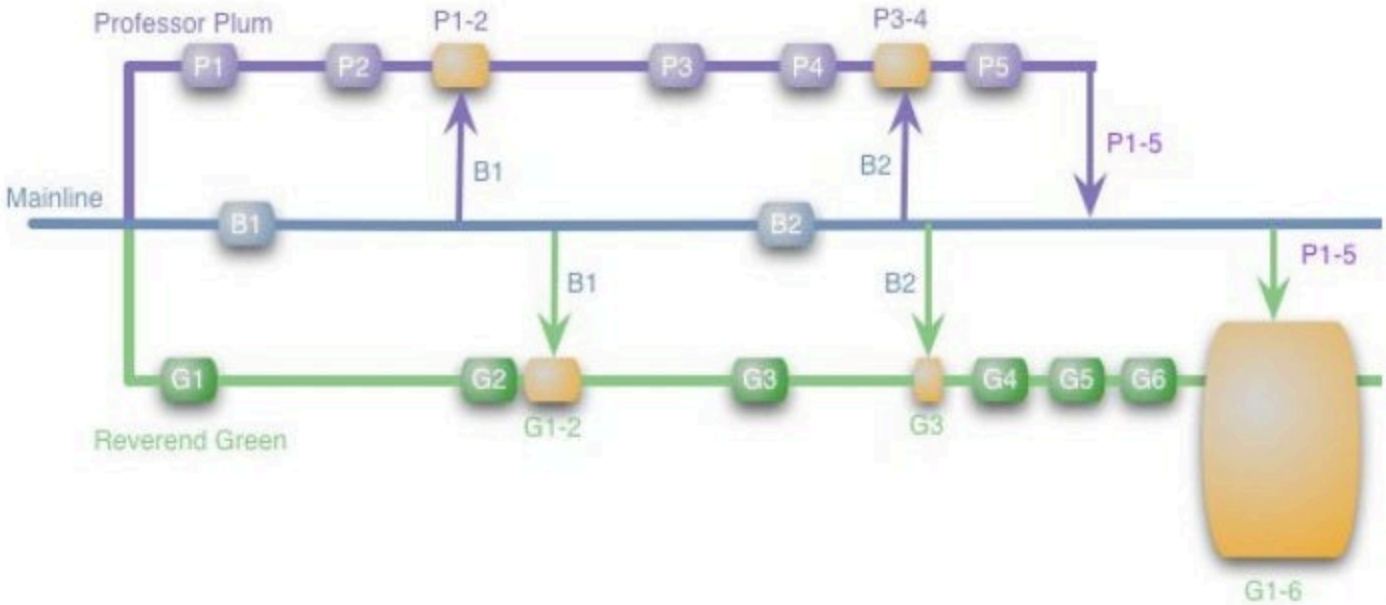
Quelle: http://martinfowler.com/bliki/FeatureBranch.html

Problem 1: ...



Quelle: http://martinfowler.com/bliki/FeatureBranch.html

Problem 1: Merge Pain



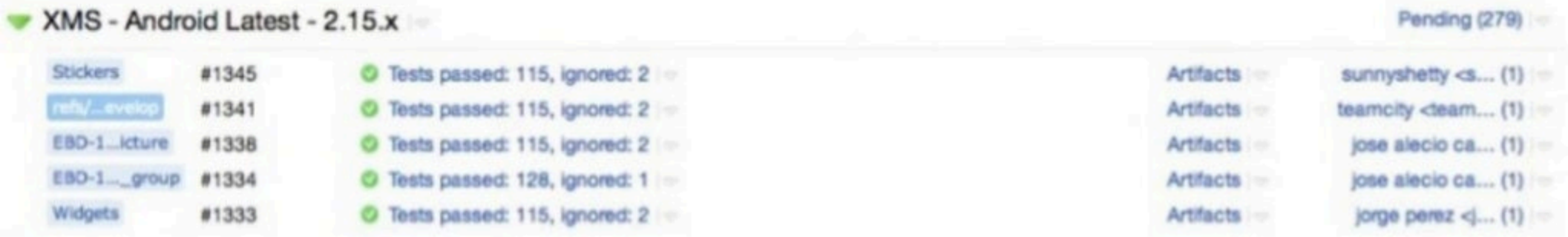
Problem 2: Communication Islands

„Branches create distance between developers and we don’t want that“

Frank Compagner

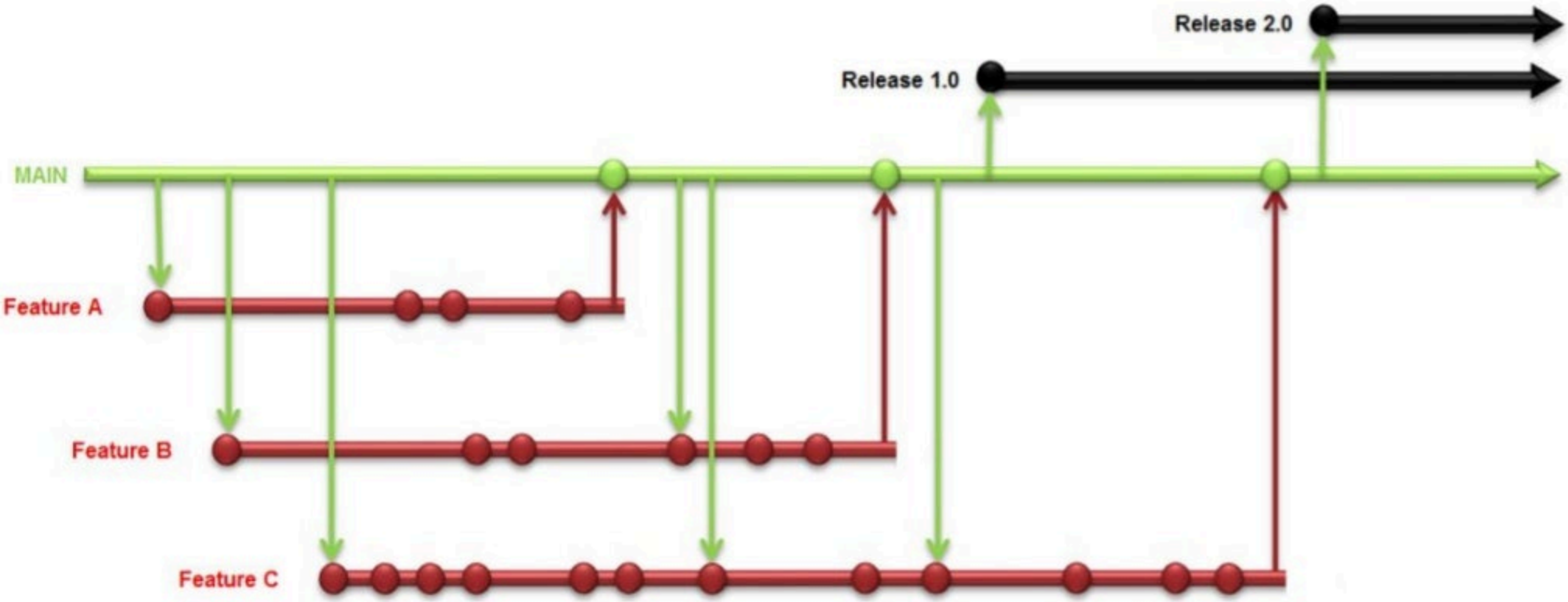
Quelle: http://paulhammant.com/2014/09/24/a-functional-branch-by-abstraction-case-study/

Problem 3: No Continuous Integration



Quelle: http://de.slideshare.net/eugenmartynov/flow-23558345

Branch Based Development



Quelle: http://geekswithblogs.net/Optikal/archive/2013/02/10/152069.aspx

Trunk Based Development

- ✓ No Merge Pain
- ✓ No Communication Islands
- ✓ Continuous Integration

Trunk Based Development

- ✓ No Merge Pain
- ✓ No Communication Islands
- ✓ Continuous Integration
- ✓ Refactoring Friendly

Trunk Based Development

There are branches.

Trunk Based Development

There are branches.

But there are no feature branches!

Feature Branches ↔ Trunk Based Dev.

Merge Conflict Resolution costs time is error-prone

Communication Islands

No Continuous Integration

No Merge Pain

No Communication Islands

Continuous Integration

Refactoring Friendly

Feature Branches ↔ Trunk Based Dev.

- ✗ Merge Conflict Resolution costs time is error-prone
- ✗ Communication Islands
- ✗ No Continuous Integration

- ✓ No Merge Pain
- ✓ No Communication Islands
- ✓ Continuous Integration
- ✓ Refactoring Friendly

- ✓ Stable trunk

Feature Branches ↔ Trunk Based Dev.

✗ Merge Conflict Resolution costs time is error-prone

✗ Communication Islands

✗ No Continuous Integration

✓ Stable trunk

✓ No Merge Pain

✓ No Communication Islands

✓ Continuous Integration

✓ Refactoring Friendly

? Unstable trunk

Development at the Speed and Scale of Google

Recorded at: QCon

by **Ashish Kumar** on Dec 13, 2010 | 4 **Discuss**

NOTICE: The next QCon is in **London Mar 2-6**, Join us!

Share | My Reading List | Read later

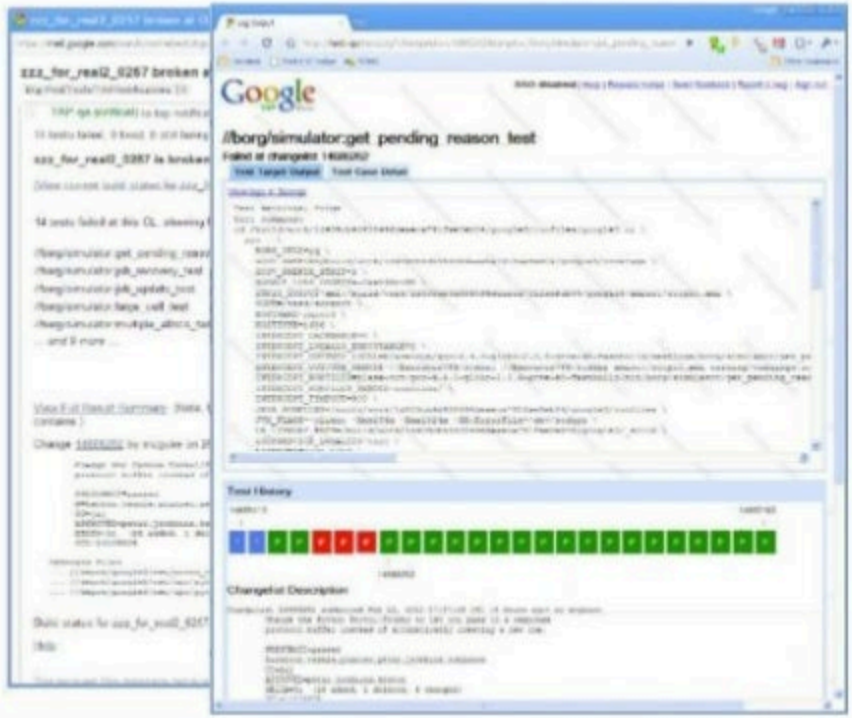
View Presentation

Download MP3 | Slides **55:17**

Summary

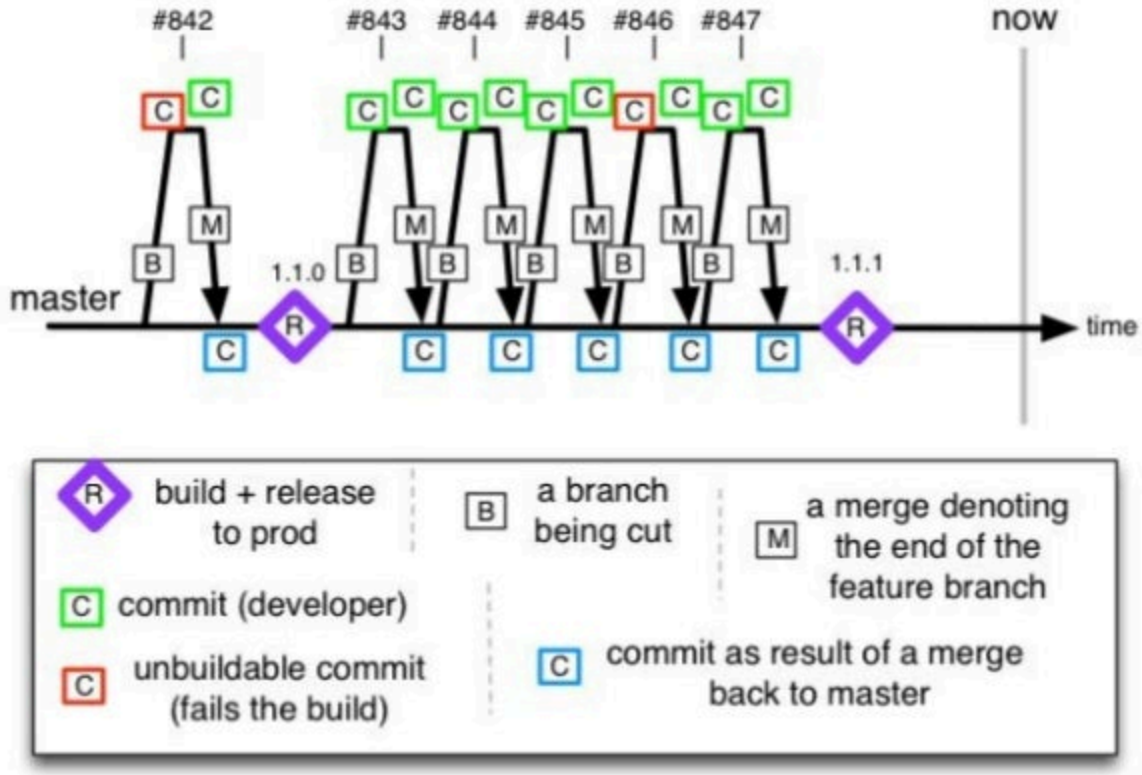
Ashish Kumar presents how Google manages to keep the source code of all its projects, over 2000, in a single code trunk containing hundreds of millions of code lines, with more than 5,000 developers accessing the same repository.

Faster time to fix



Do you have pre-commit-verification?

Use short lived *issue branches* or *personal branches* instead.



Quelle: http://paulhammant.com/2013/12/04/what_is_your_branching_model/

How do you prevent,
that incomplete features are released?

Feature Toggles

aka Feature Bits

aka Feature Flags

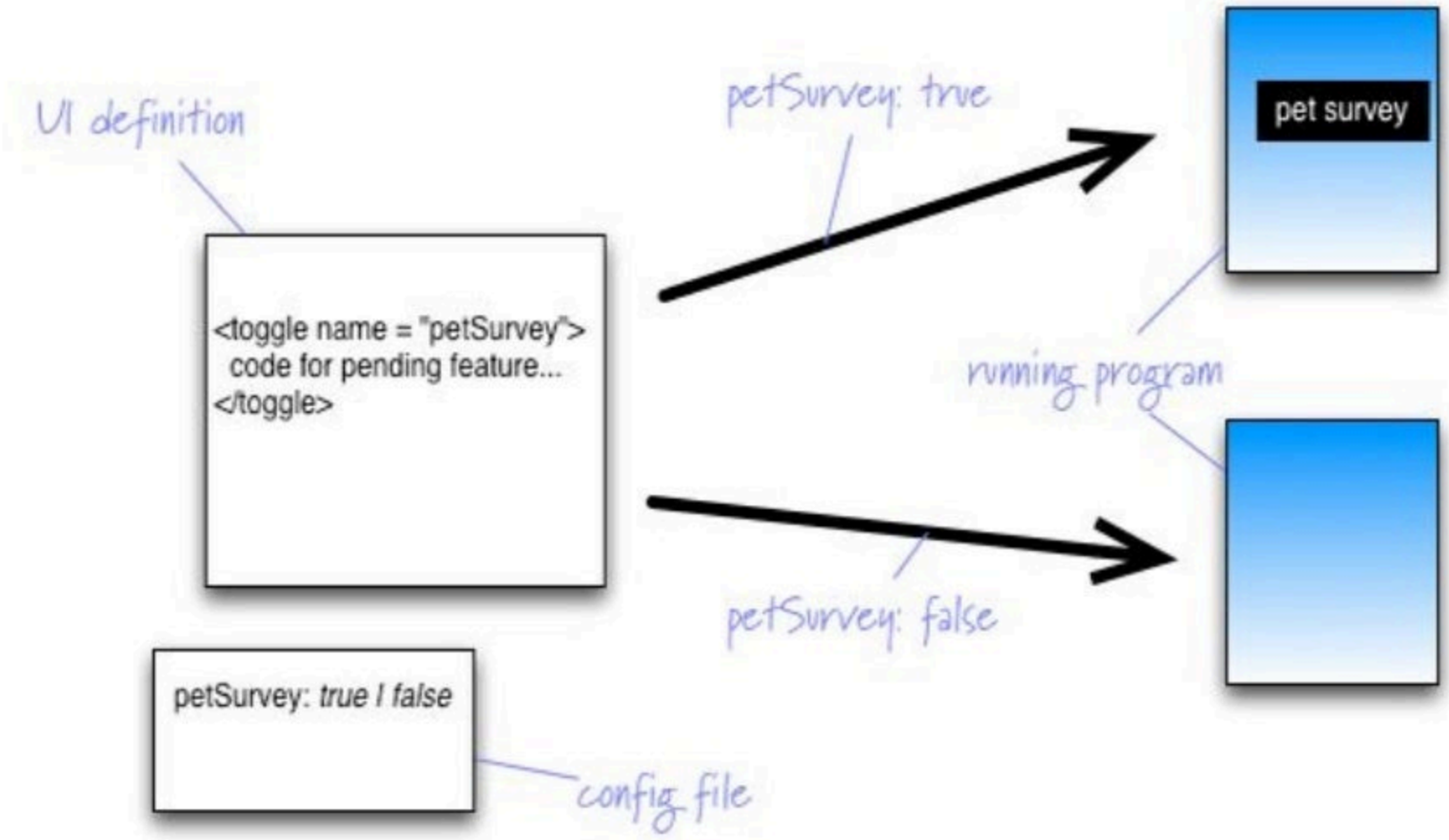
aka Feature Switches

aka Feature Flippers

aka Conditional Features

aka Latent Code Pattern

Feature Toggles



Quelle: http://martinfowler.com/bliki/FeatureToggle.html

I 

FEATURE TOGGLES

Why Feature Toggles?

Hide unfinished features from users

Why Feature Toggles?

✓ Hide unfinished features from users

“Stable trunk” for visible functionality

Why Feature Toggles?

- ✓ Hide unfinished features from users
- ✓ Separate Deployment and Release

Why Feature Toggles?

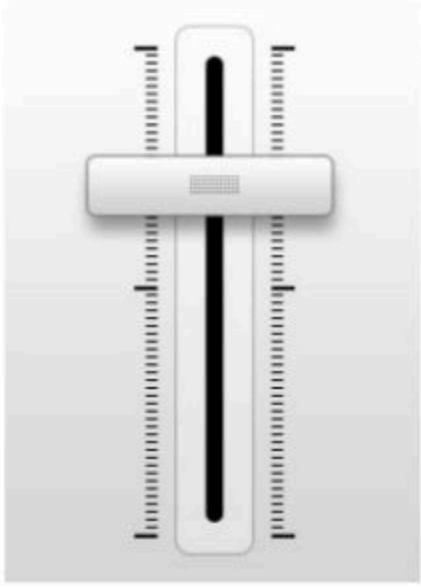
- ✓ Hide unfinished features from users
- ✓ Separate Deployment and Release
- ✓ Fearless Deployment

Why Feature Toggles?

- ✓ Hide unfinished features from users
- ✓ Separate Deployment and Release
- ✓ Fearless Deployment
- ✓ No Big Bang Releases

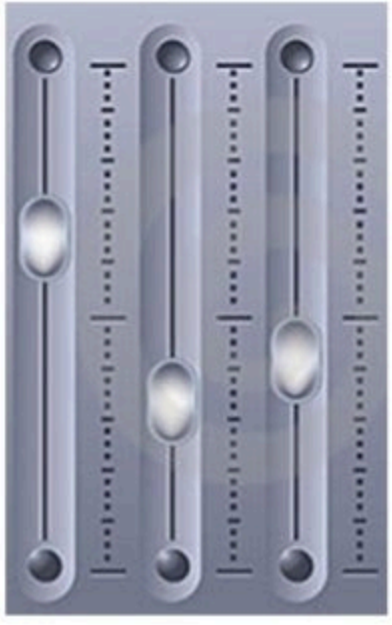
Granularity

Granularity



- per environment (local, staging, live)
- per server
- per customer/mandator
  - inherited if you have a hierarchy
- per user
  - a certain percentage of users
- per country
- per session
  - a certain percentage of sessions
- ...

Granularity



- per environment (local, staging, live)
- per server
- per customer/mandator
  - inherited if you have a hierarchy
- per user
  - a certain percentage of users
- per country
- per session
  - a certain percentage of sessions
- ...

Granularity (example #1)

enableMediaTypeLetter

Possible values: "true", "false", "inherited". If enabled media type letter is available.

inherited — Set to default 'inherited' / remove

verifyHBaseImplementationForRecipientData

Possible values: "true", "false", "inherited" or something like "dmn-01, trk" to enable this feature only on certain servers.

gui-test, dmn — Set to default 'inherited' / remove

useHBaseImplementationForRecipientData

Possible values: "true", "false", "inherited" or something like "dmn-01, trk" to enable this feature only on certain servers.

inherited — Set to default 'inherited' / remove

useBroadmailRemoteQueryRunner2

Possible values: "true", "false", "inherited" or something like "dmn-01, trk" to enable this feature only on certain servers.

true — Set to default 'inherited' / remove

enableContentValidationInSmartCampaigns

Possible values: "true", "false", "inherited". If enabled the content of each message node in a smart campaign will be checked for velocity syntax errors.

true — Set to default 'inherited' / remove

Granularity (example #2)

MAIN

- Home
- Downloads
- Source Code
- Forums
- Issue Tracker
- stackoverflow.com
- Continuous Integration
- License

REFERENCE

- What's new?

Activation Strategies

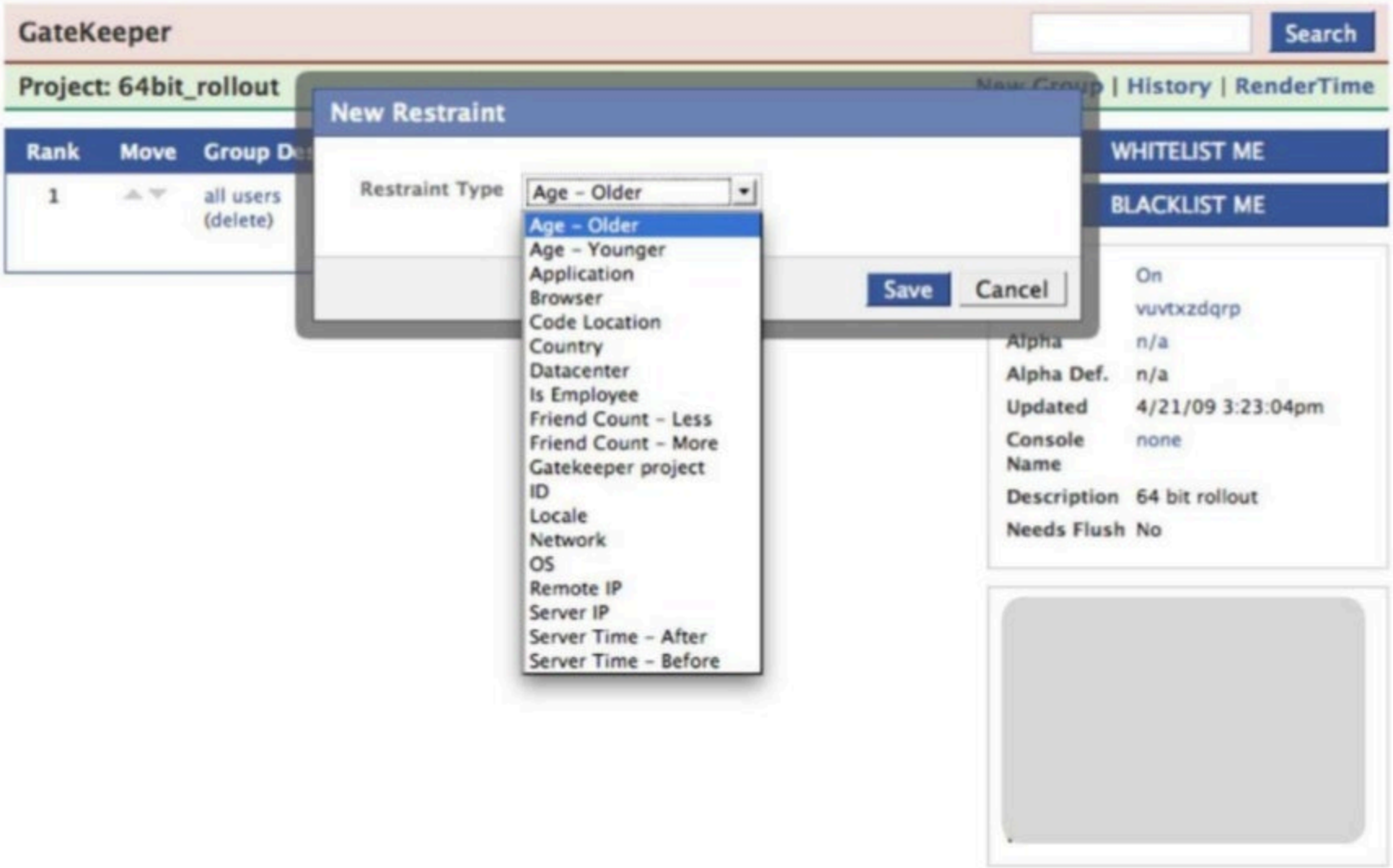
Togglz defines the concept of *activation strategies*. They are responsible to decide whether an enabled feature is active or not. Activation strategies can for example be used to activate features only for specific users, for specific client IPs or at a specified time.

Togglz ships with the following default strategies:

- Username
- Gradual rollout
- Release date
- Client IP
- Server IP
- ScriptEngine

Quelle: http://www.togglz.org/documentation/activation-strategies.html

Granularity (example #3)



Why Feature Toggles?

- ✓ Hide unfinished features from users
- ✓ Separate Deployment and Release
- ✓ Fearless Deployment
- ✓ No Big Bang Releases

Why Feature Toggles?

- ✓ Hide unfinished features from users
- ✓ Separate Deployment and Release
- ✓ Fearless Deployment
- ✓ No Big Bang Releases
- ✓ Better Software Architecture

Feature Toggle ≠ Feature Toggle

Feature Toggle ≠ Feature Toggle

long lived,
payed for by customer

Feature Toggle ≠ Feature Toggle

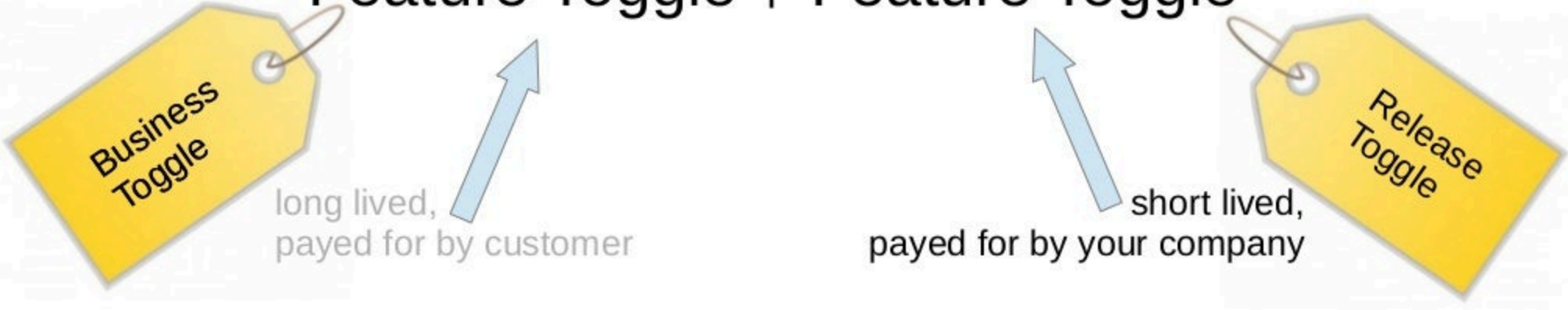
long lived,
payed for by customer

Feature Toggle ≠ Feature Toggle

long lived,
payed for by customer

short lived,
payed for by your company

Feature Toggle ≠ Feature Toggle



Release Toggle ≠ Release Toggle

Release Toggle ≠ Release Toggle

hides a **new feature**

Release Toggle ≠ Release Toggle

hides a **new feature**

e.g. new functionality
(new menu point,
new dialog, ...)

Release Toggle ≠ Release Toggle

hides a **new feature**

e.g. new functionality
(new menu point,
new dialog, ...)

hides a **new implementation**

Release Toggle ≠ Release Toggle

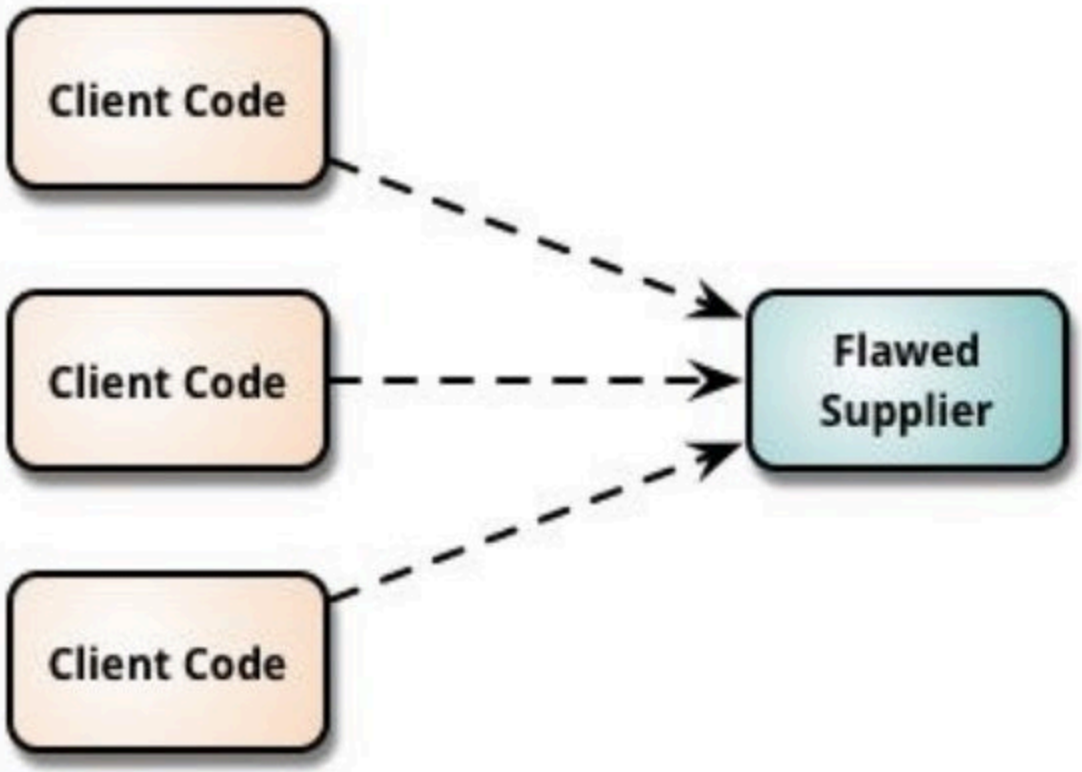
hides a **new feature**

e.g. new functionality
(new menu point,
new dialog, ...)

hides a **new implementation**

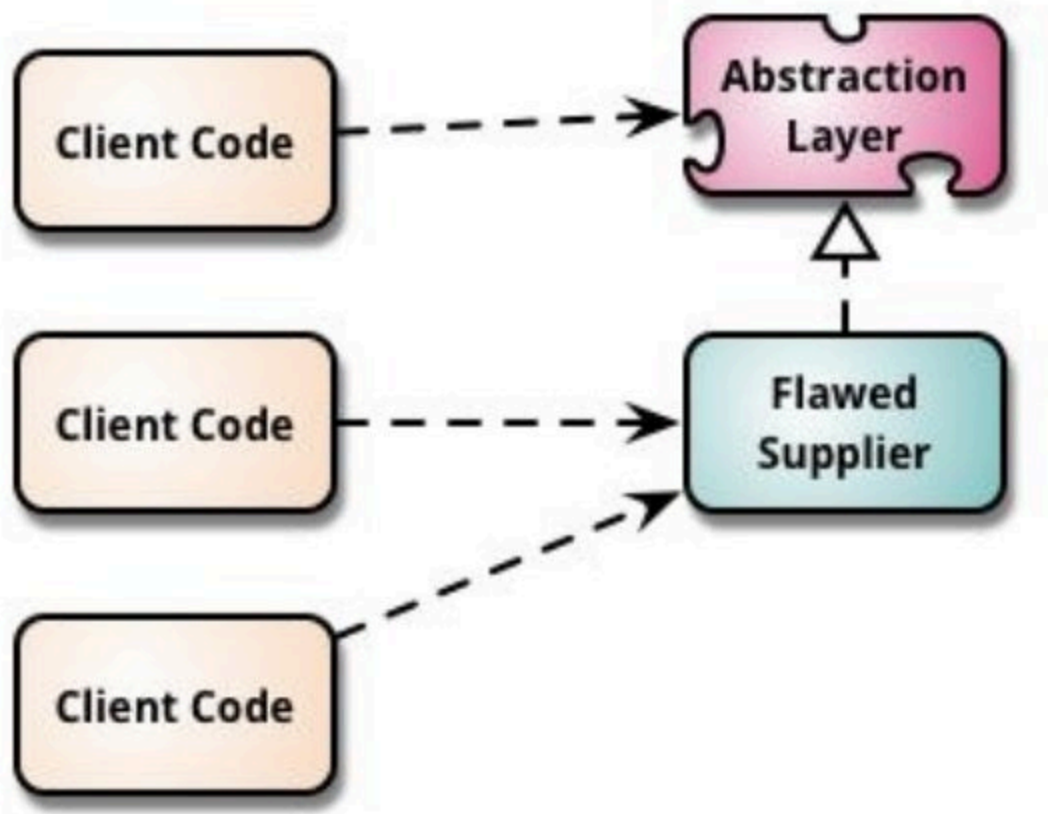
e.g. - new design
- new backend technology
(PHP → Java or
Hibernate → MongoDB)
- performance optimization
(added caching or
better algorithm)
- ...

Branch By Abstraction



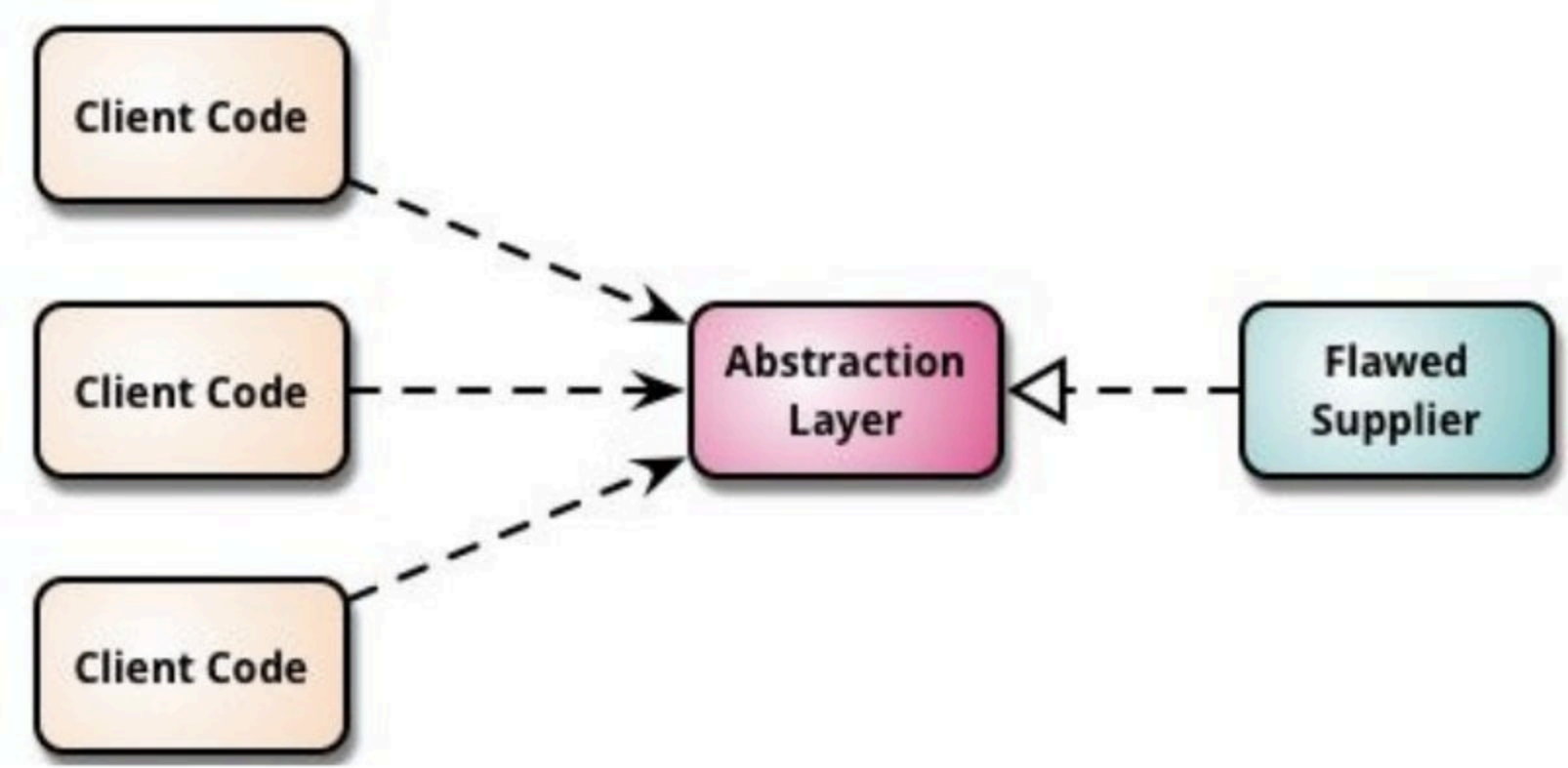
Quelle: http://martinfowler.com/bliki/BranchByAbstraction.html

Branch By Abstraction



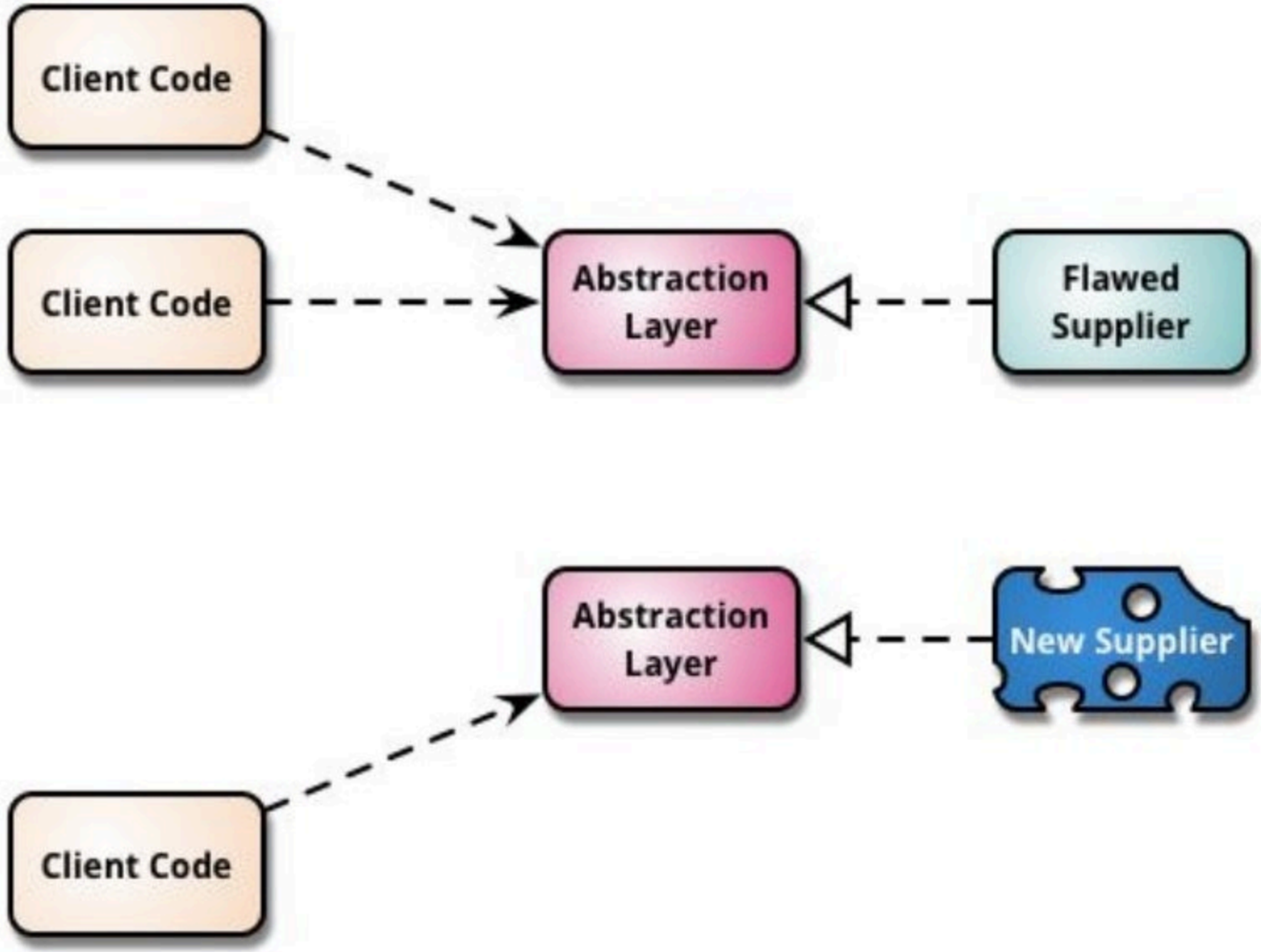
Quelle: http://martinfowler.com/bliki/BranchByAbstraction.html

Branch By Abstraction



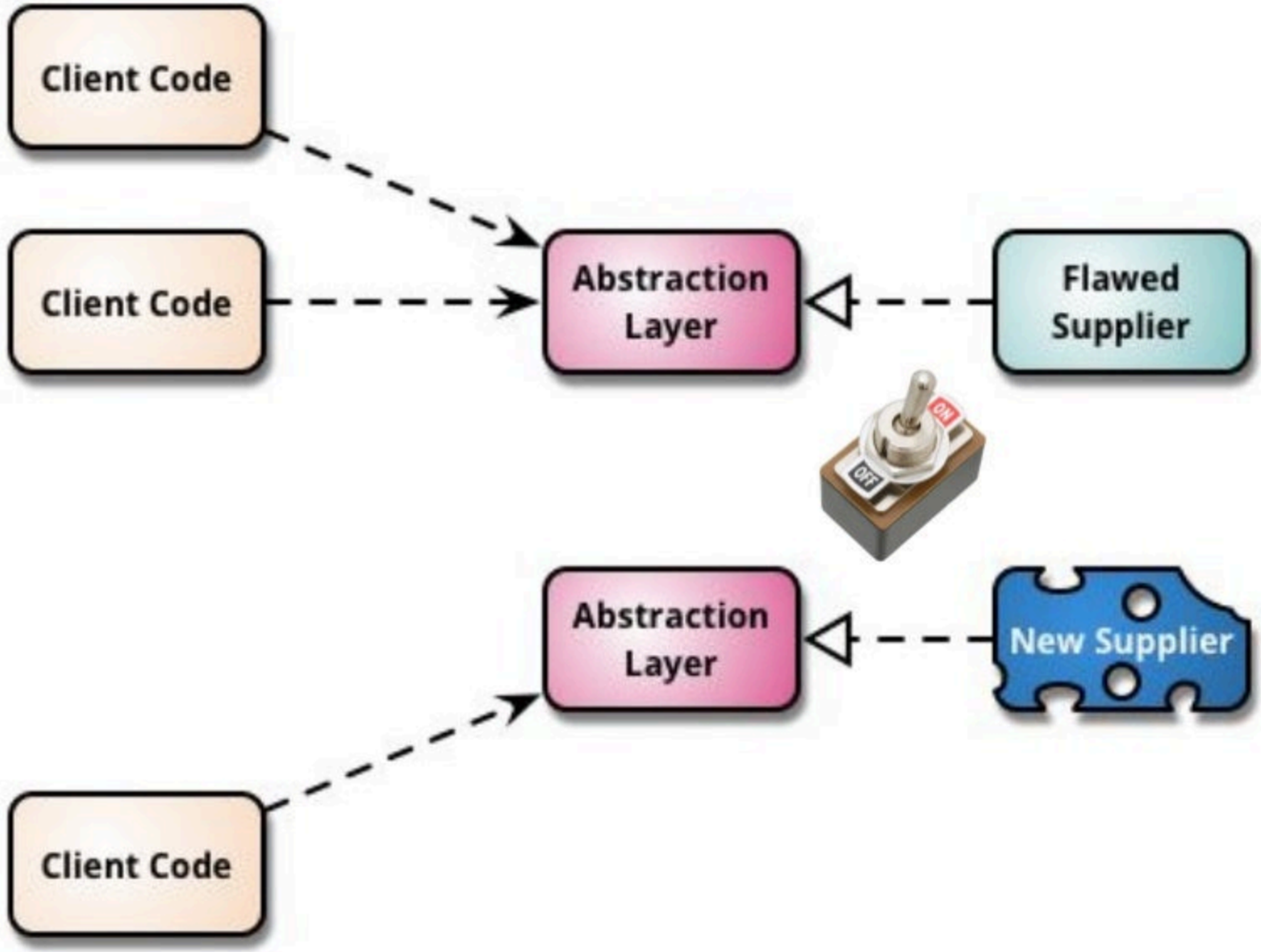
Quelle: http://martinfowler.com/bliki/BranchByAbstraction.html

Branch By Abstraction



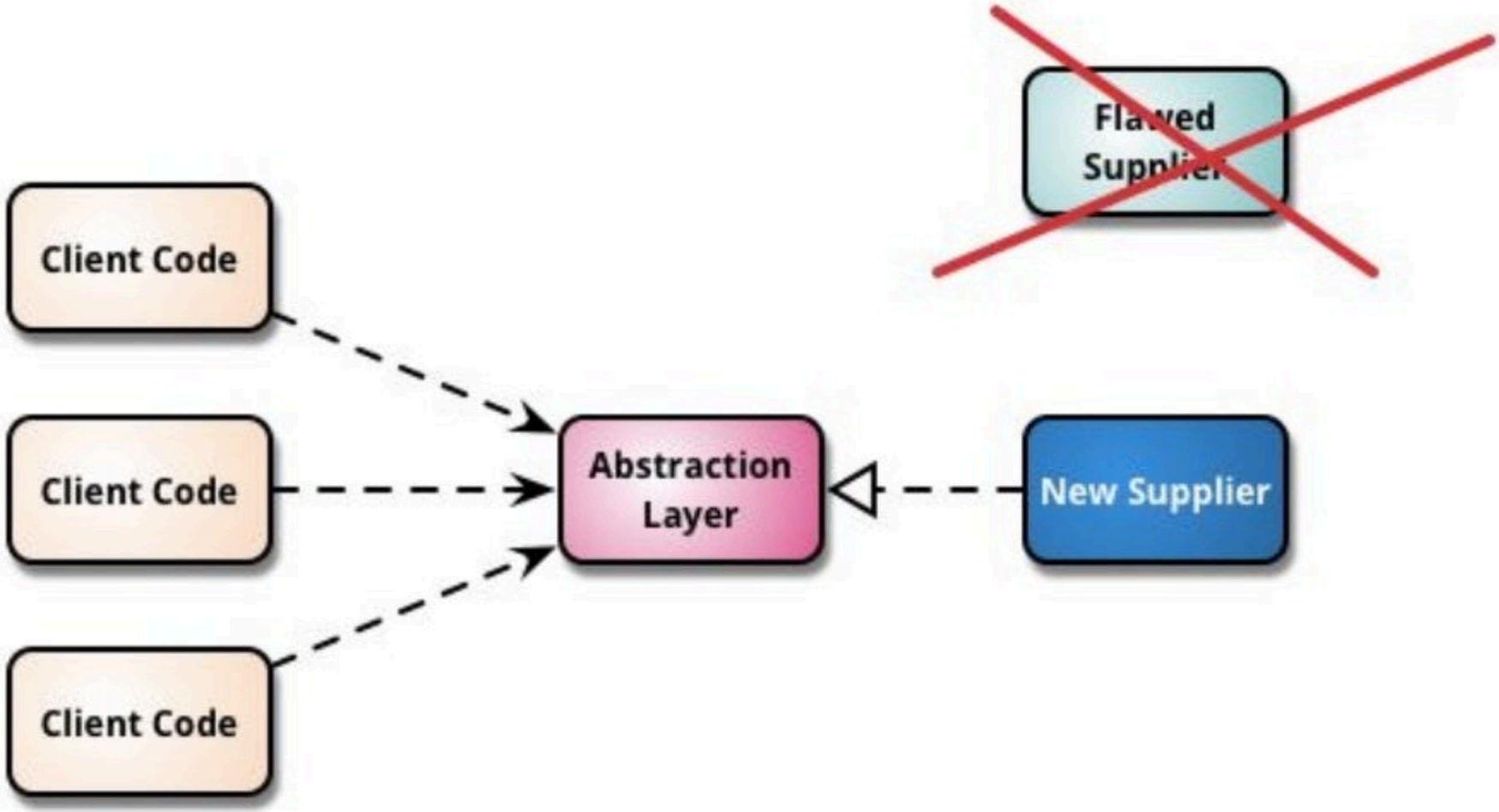
Quelle: http://martinfowler.com/bliki/BranchByAbstraction.html

Branch By Abstraction



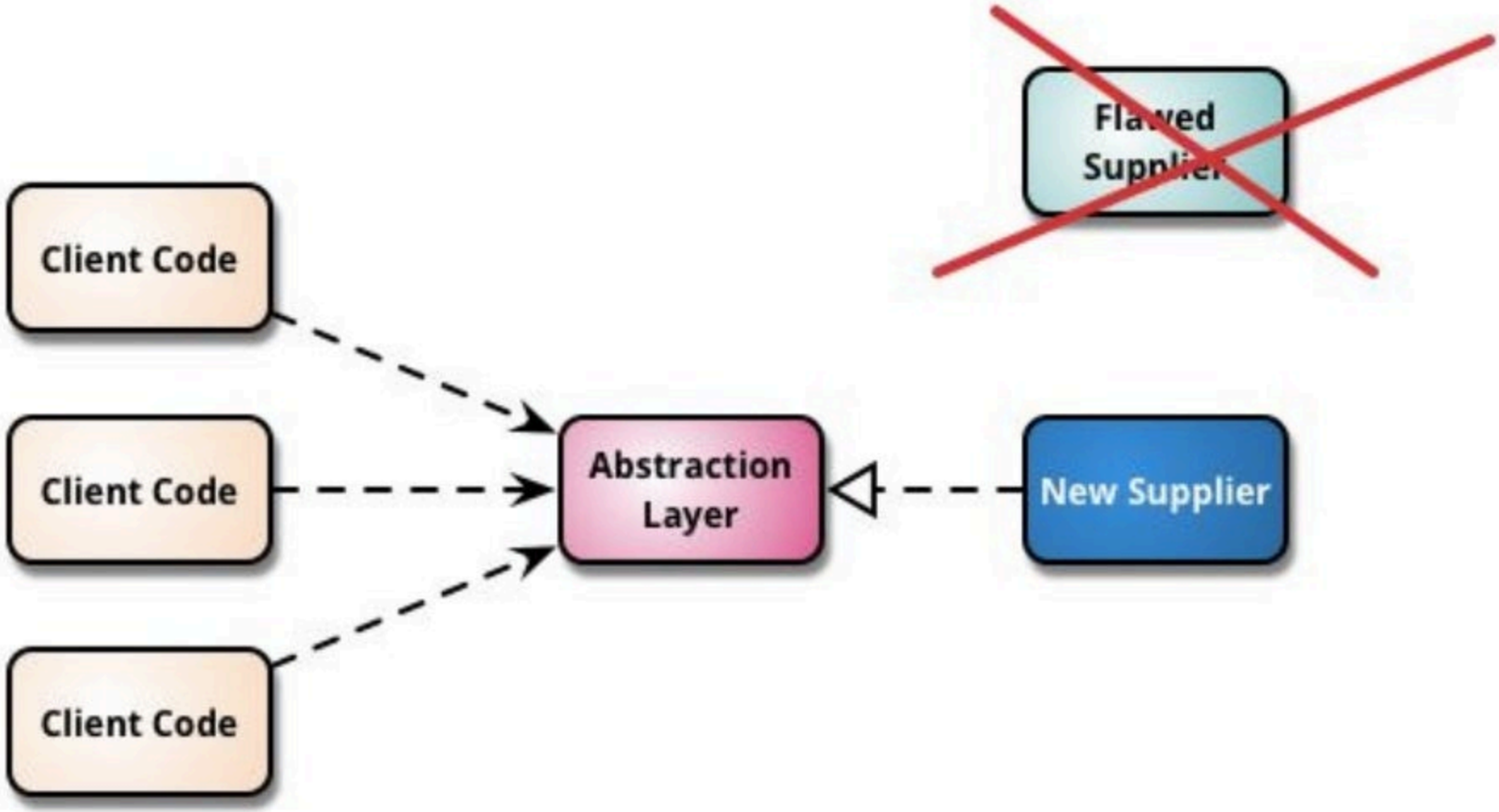
Quelle: http://martinfowler.com/bliki/BranchByAbstraction.html

Branch By Abstraction



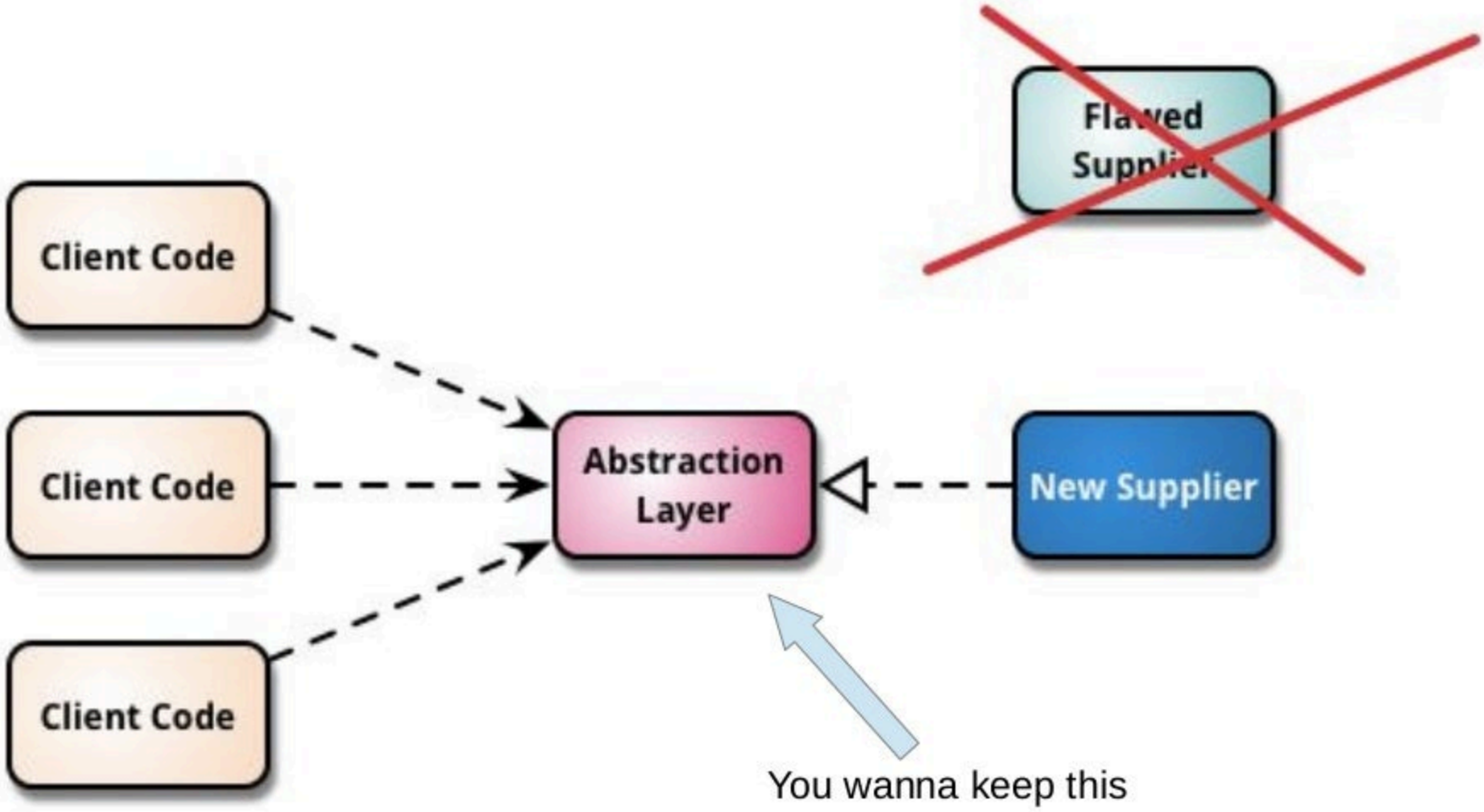
Quelle: http://martinfowler.com/bliki/BranchByAbstraction.html

Branch By Abstraction



Quelle: http://martinfowler.com/bliki/BranchByAbstraction.html

Branch By Abstraction



Quelle: http://martinfowler.com/bliki/BranchByAbstraction.html

Why Feature Toggles?

- ✓ Hide unfinished features from users
- ✓ Separate Deployment and Release
- ✓ Fearless Deployment
- ✓ No Big Bang Releases
- ✓ Better Software Architecture

Why Feature Toggles?

- ✓ Hide unfinished features from users
- ✓ Separate Deployment and Release
- ✓ Fearless Deployment
- ✓ No Big Bang Releases
- ✓ Better Software Architecture
- ✓ Testing in Live Environment with Live Data

Why Feature Toggles?

- ✓ Hide unfinished features from users
- ✓ Separate Deployment and Release
- ✓ Fearless Deployment
- ✓ No Big Bang Releases
- ✓ Better Software Architecture
- ✓ Testing in Live Environment with Live Data

Release Toggle ≠ Release Toggle

hides a **new feature**

hides a **new implementation**

Release Toggle ≠ Release Toggle

hides a **new feature**

Dark Launching Pattern

hides a **new implementation**

Dark Launching (example)

Facebook Chat

von Eugene Letuchy, Mittwoch, 14. Mai 2008 um 07:56

Ramping up:

The secret for going from zero to seventy million users overnight is to avoid doing it all in one fell swoop. We chose to simulate the impact of many real users hitting many machines by means of a "dark launch" period in which Facebook pages would make connections to the chat servers, query for presence information and simulate message sends without a single UI element drawn on the page. With the "dark launch" bugs fixed, we hope that you enjoy Facebook Chat now that the UI lights have been turned on.

Release Toggle ≠ Release Toggle

hides a **new feature**

Dark Launching Pattern

Beta-Testing
A/B-Testing

hides a **new implementation**

Release Toggle ≠ Release Toggle

hides a **new feature**

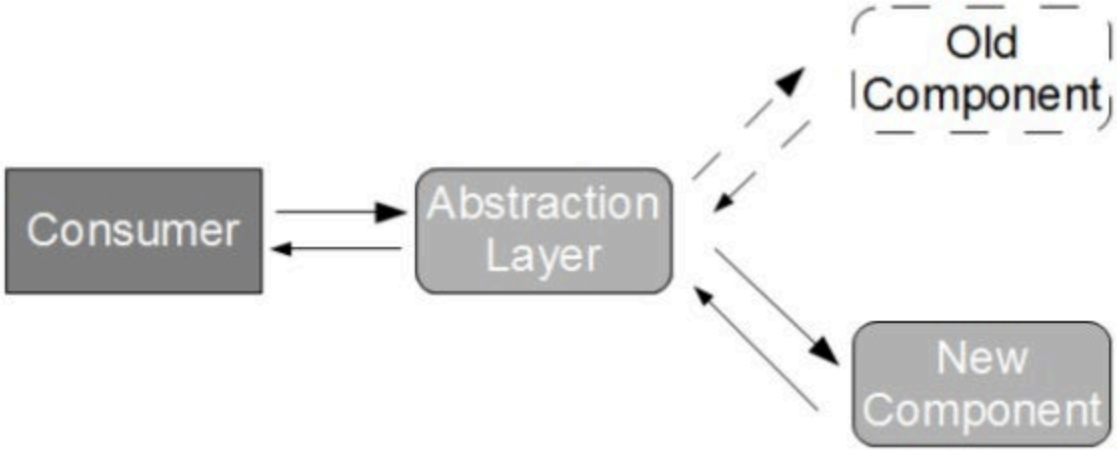
Dark Launching Pattern

Beta-Testing
A/B-Testing

hides a **new implementation**

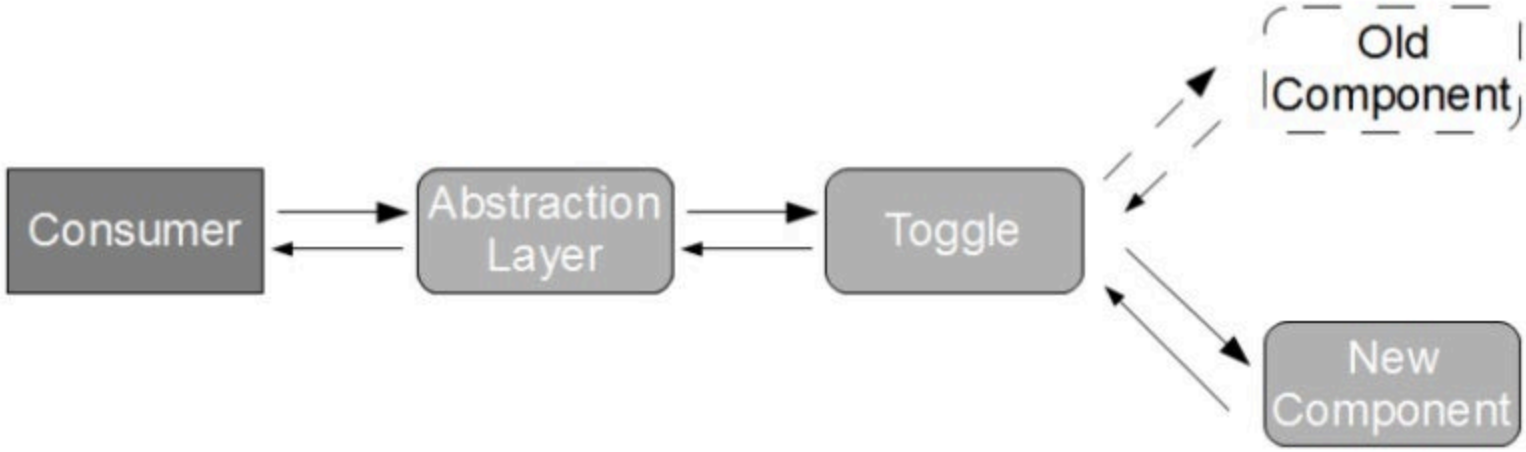
Verify Branch By Abstraction Pattern

Branch By Abstraction



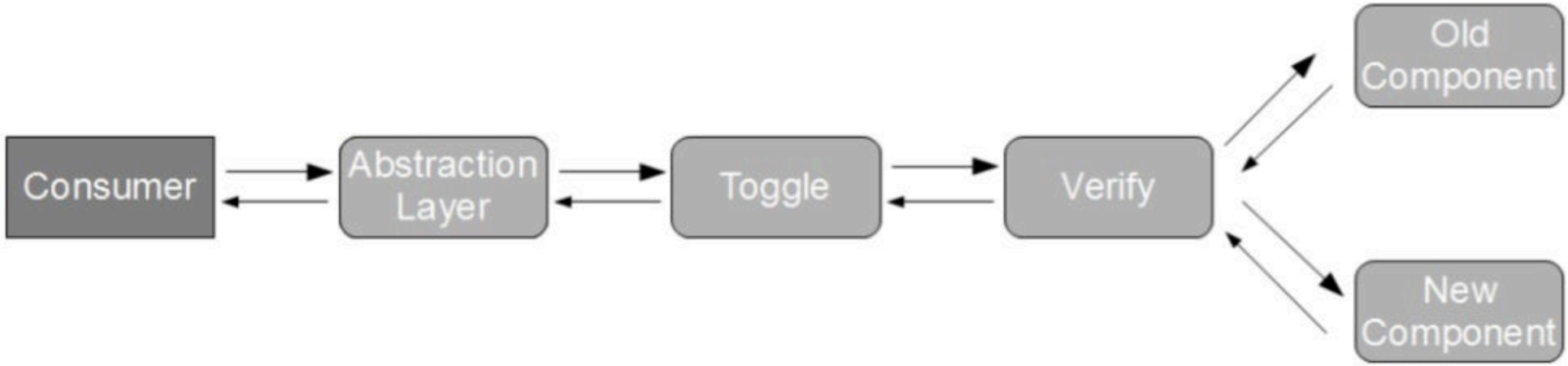
Quelle: http://www.stephen-smith.co.uk/application-pattern-verify-branch-by-abstraction/

Branch By Abstraction



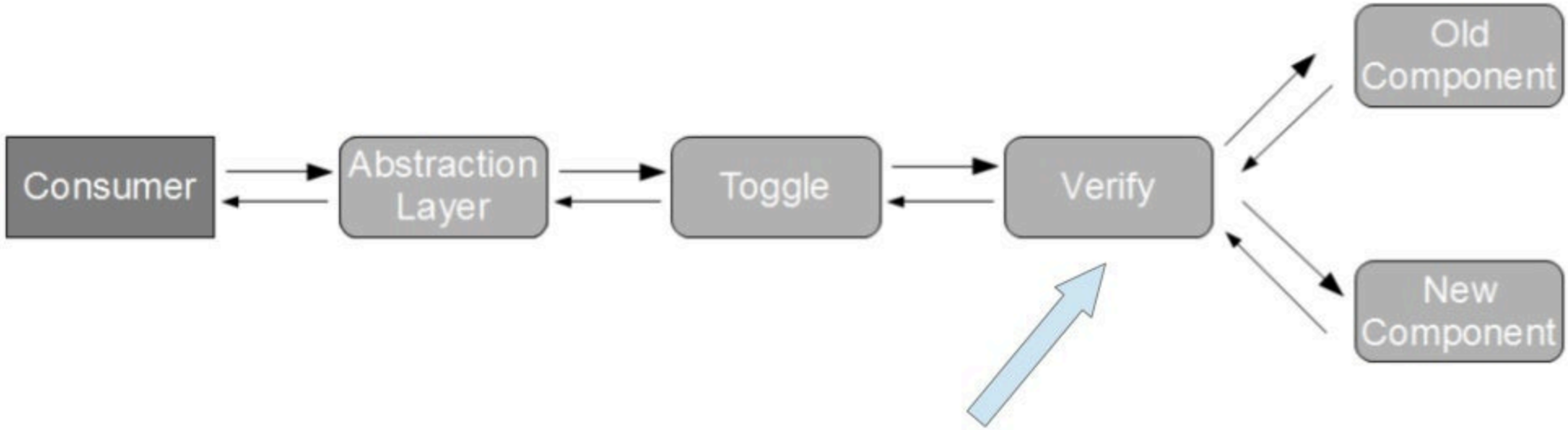
Quelle: http://www.stephen-smith.co.uk/application-pattern-verify-branch-by-abstraction/

Verify Branch By Abstraction



Quelle: http://www.stephen-smith.co.uk/application-pattern-verify-branch-by-abstraction/

Verify Branch By Abstraction



- asynchronously
  (i.e. in the background,
  invisible for users)
- check for correctness
- check for performance

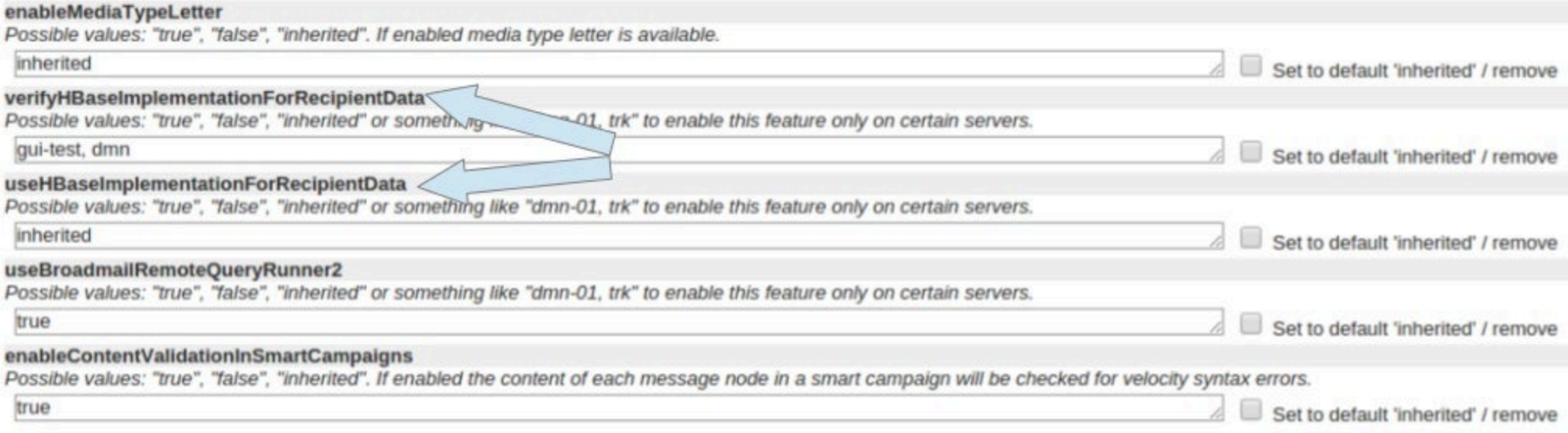
Quelle: http://www.stephen-smith.co.uk/application-pattern-verify-branch-by-abstraction/

Implementation: 2 Feature Toggles

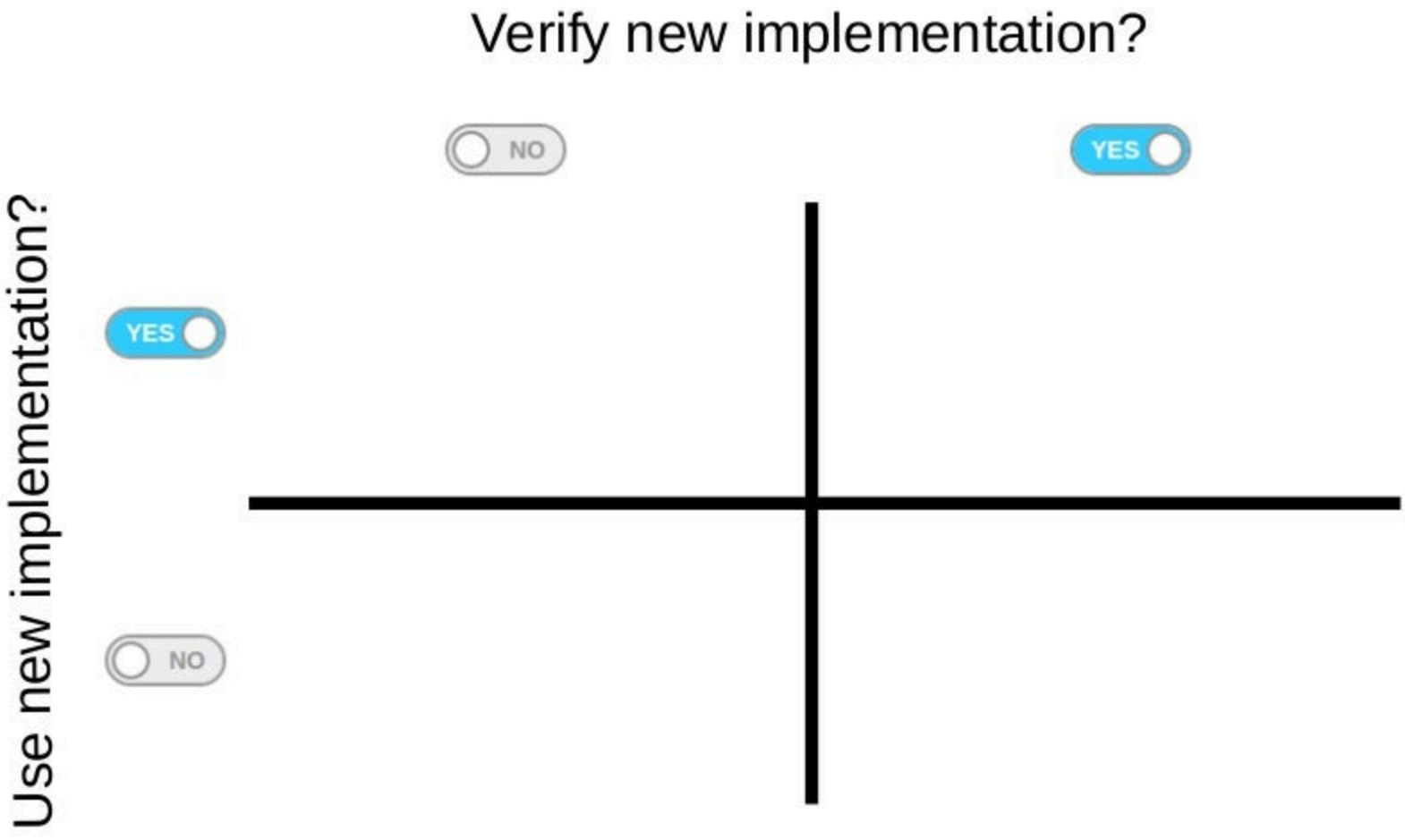
Verify new implementation?

Use new implementation?

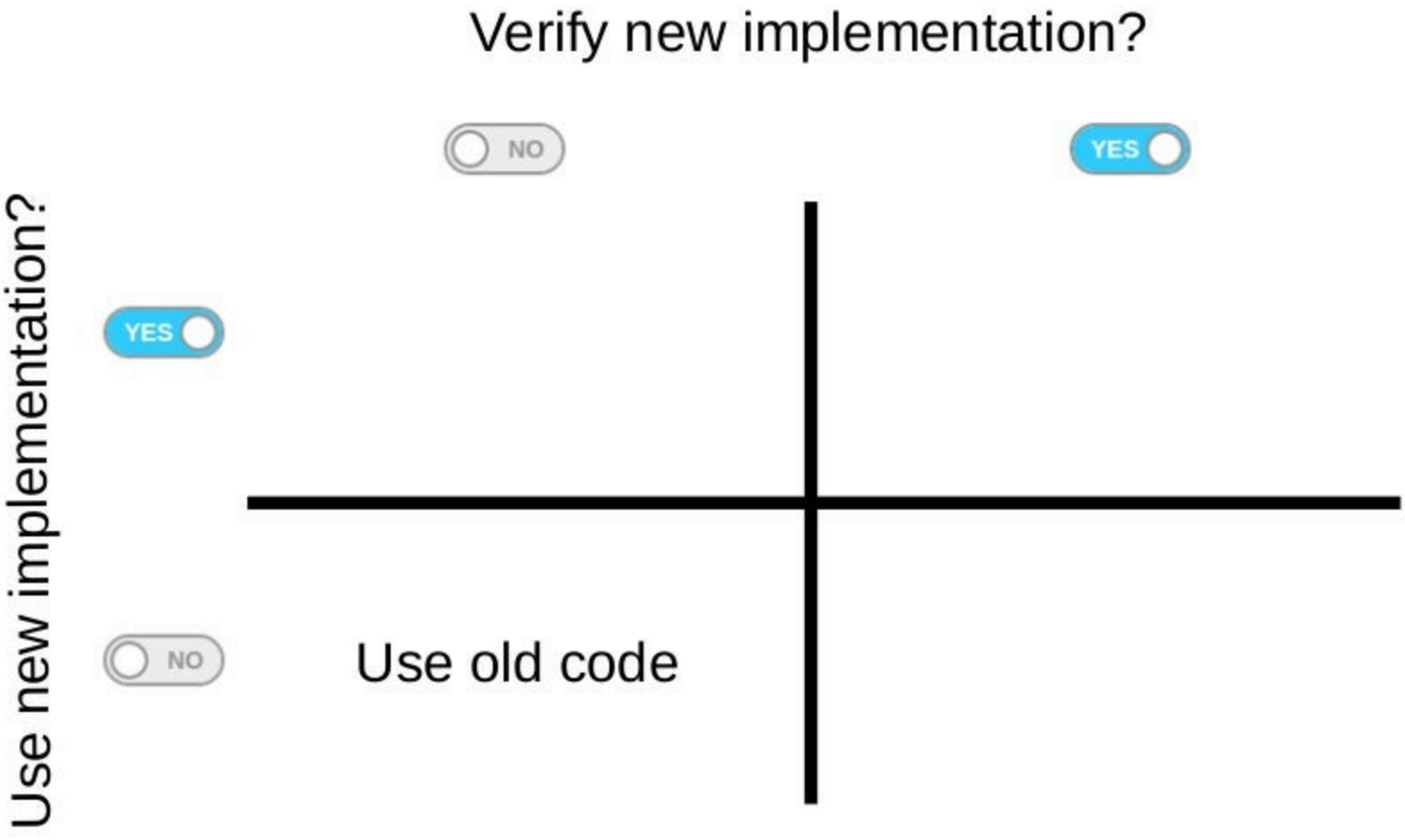
Implementation: 2 Feature Toggles



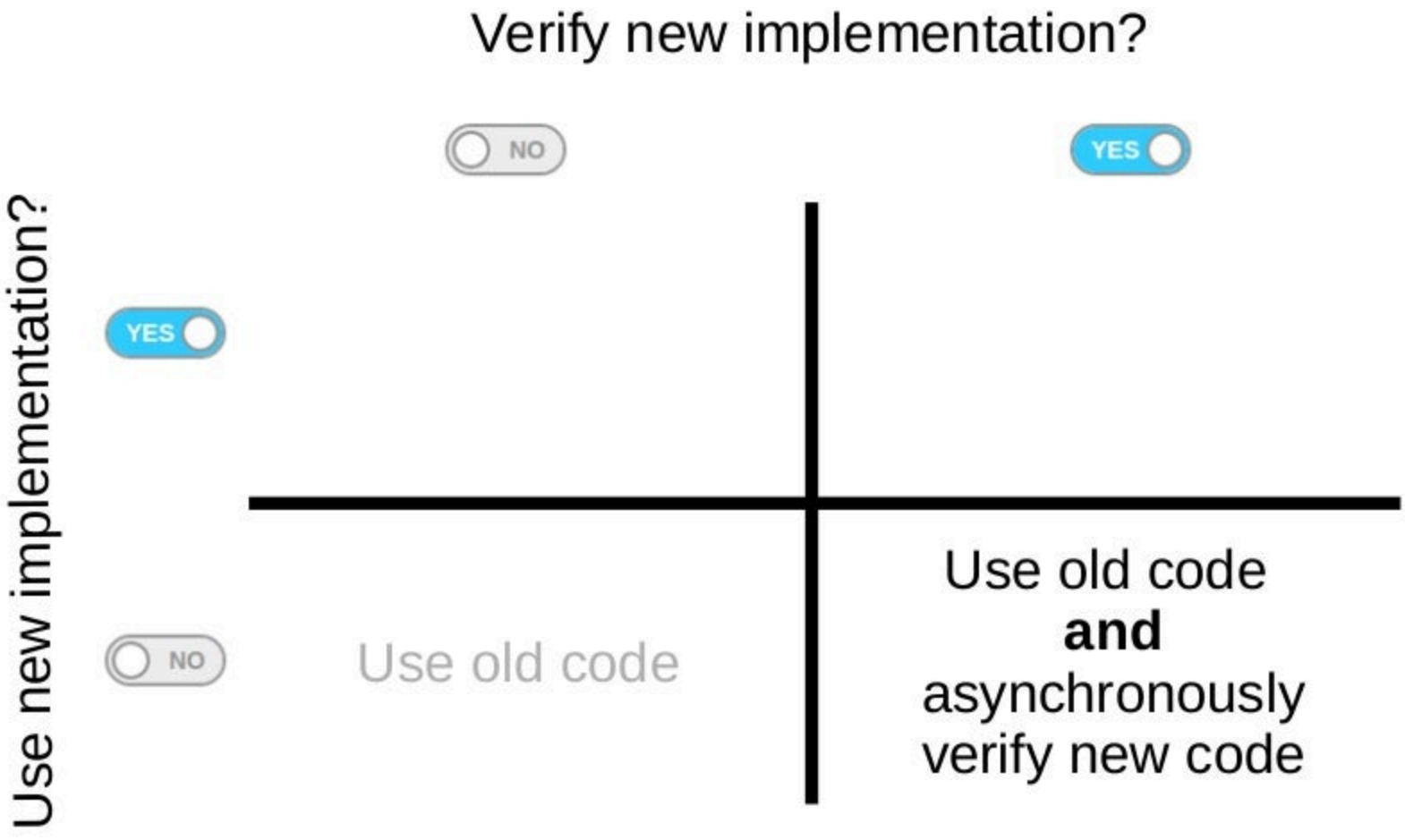
Implementation: 2 Feature Toggles



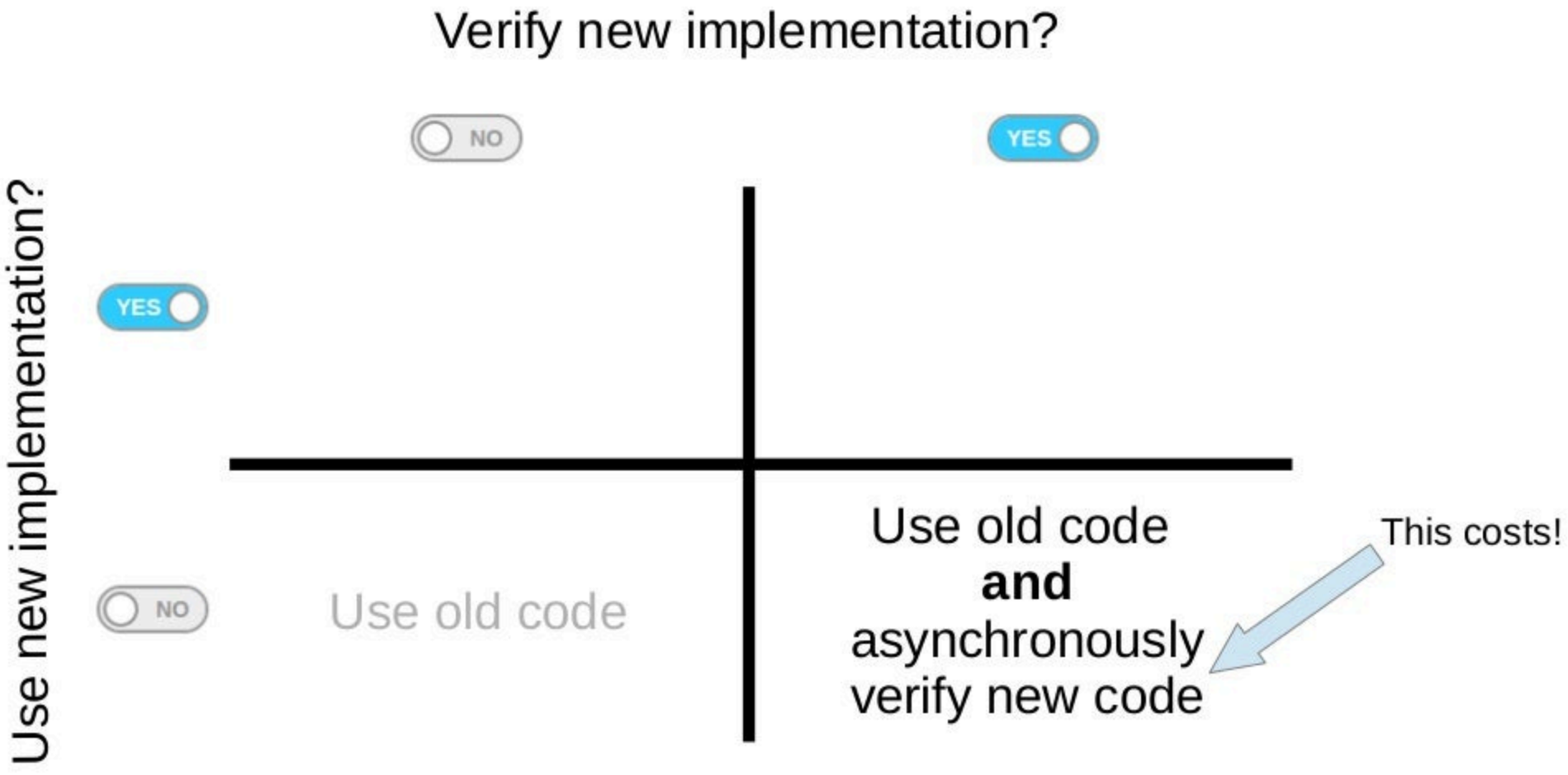
Implementation: 2 Feature Toggles



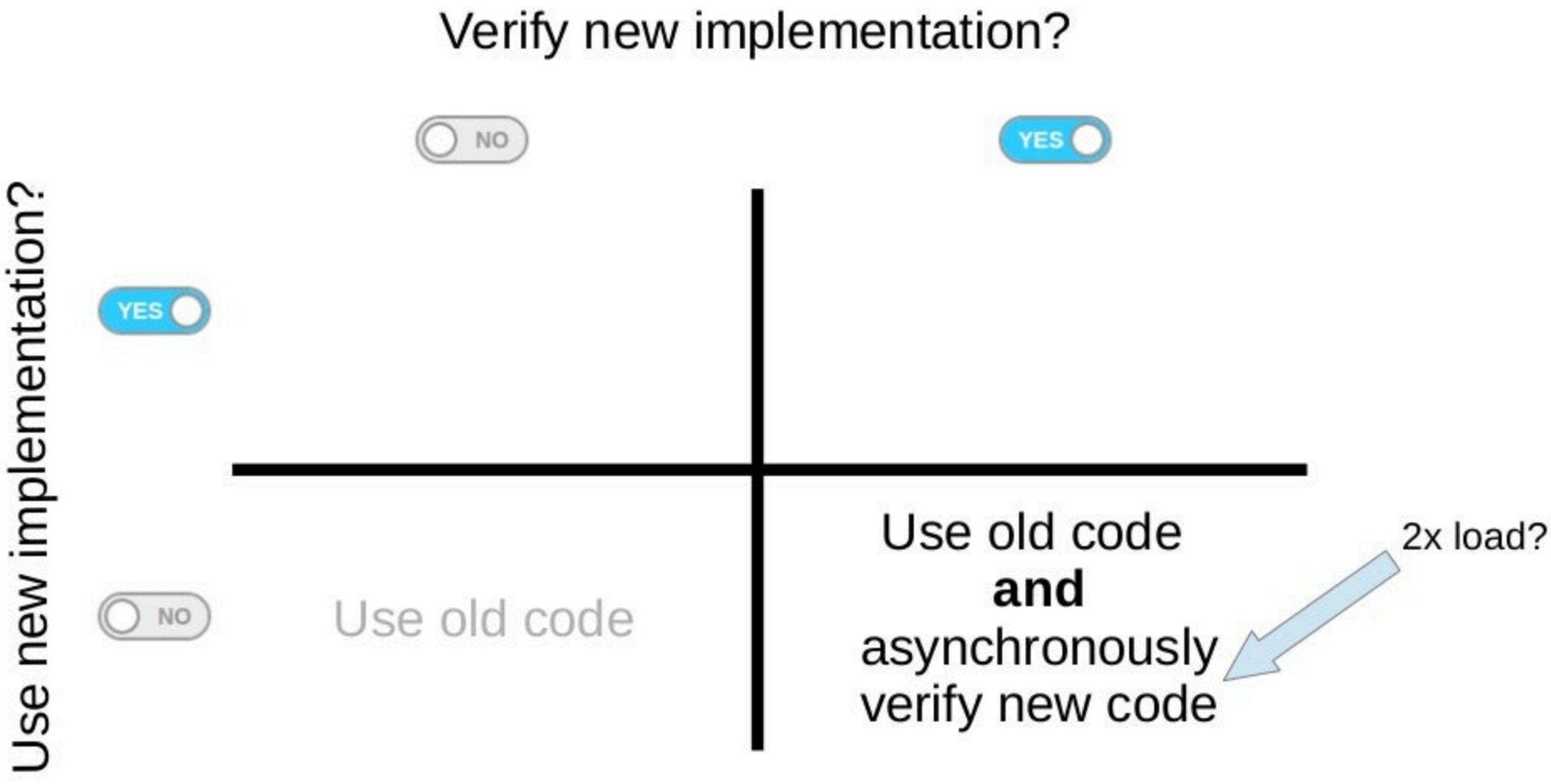
Implementation: 2 Feature Toggles



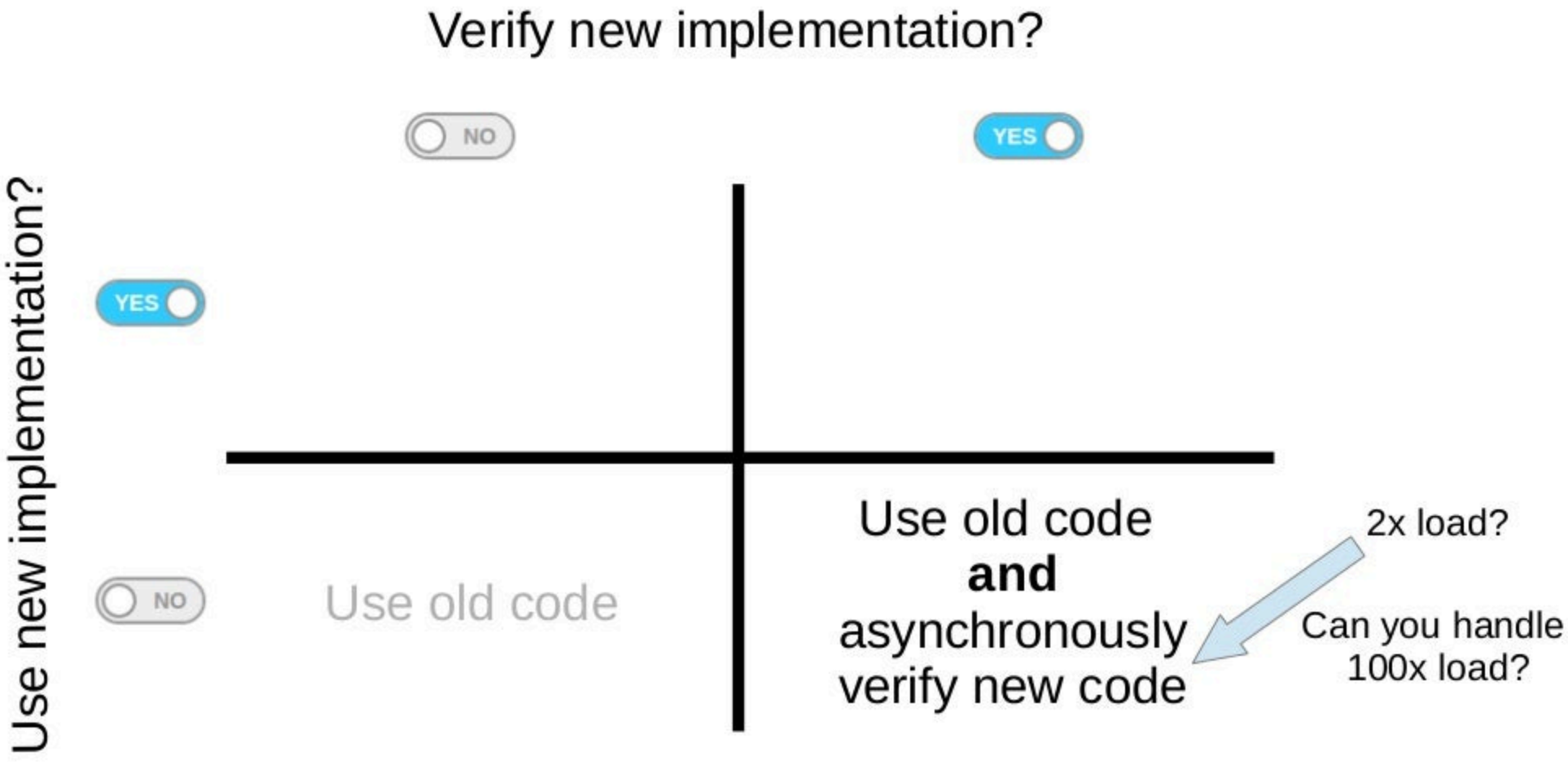
Implementation: 2 Feature Toggles



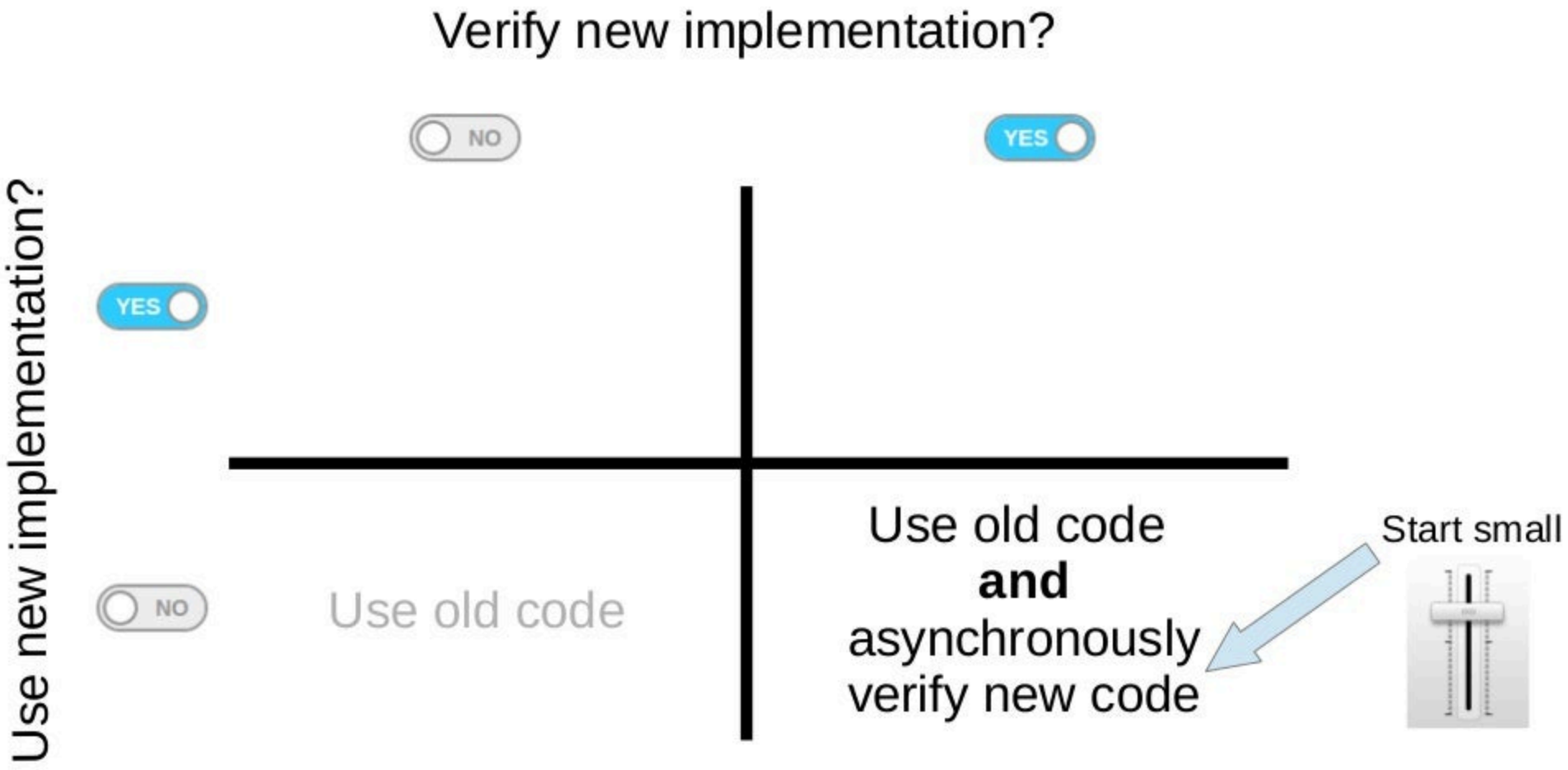
Implementation: 2 Feature Toggles



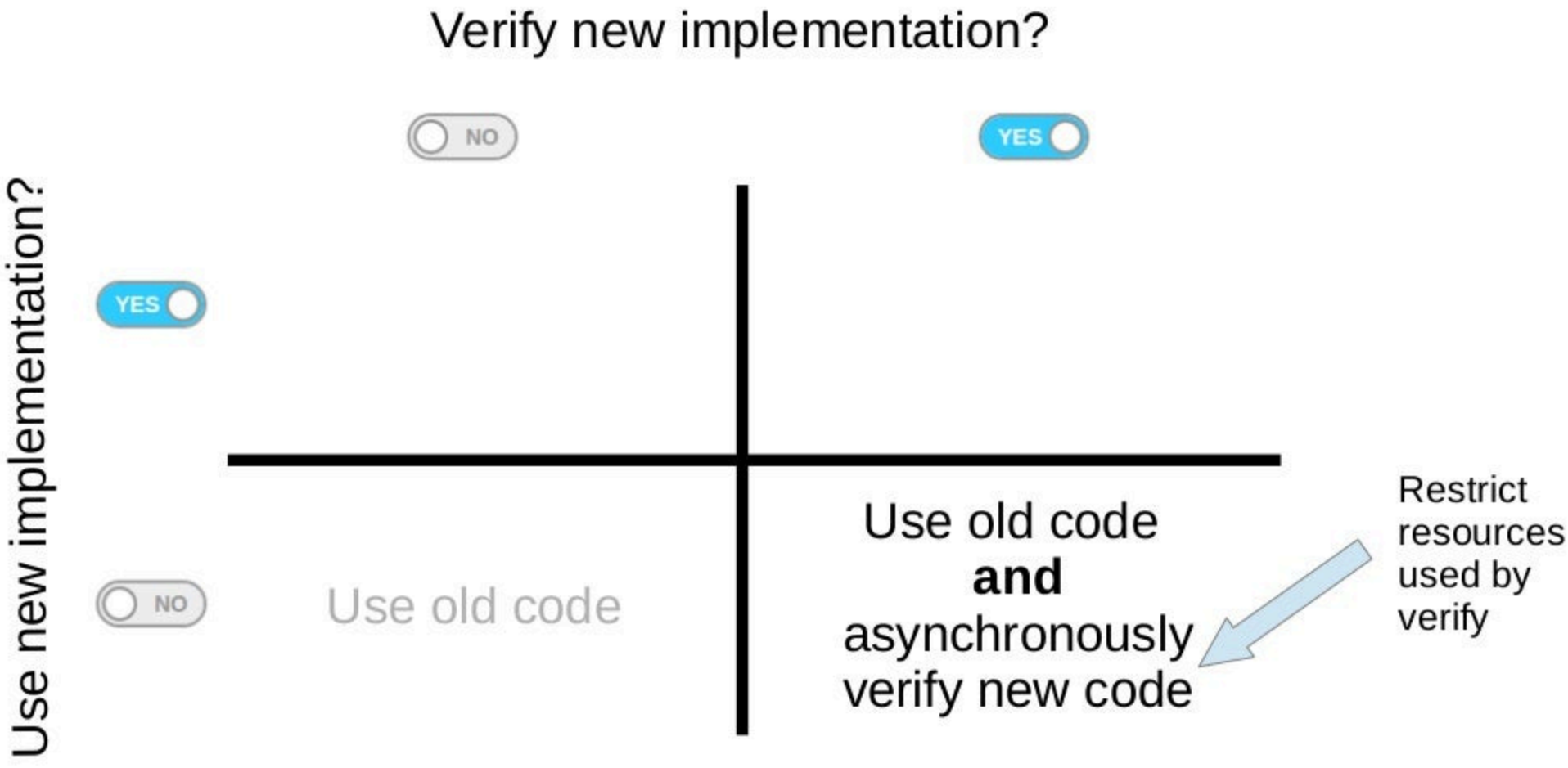
Implementation: 2 Feature Toggles



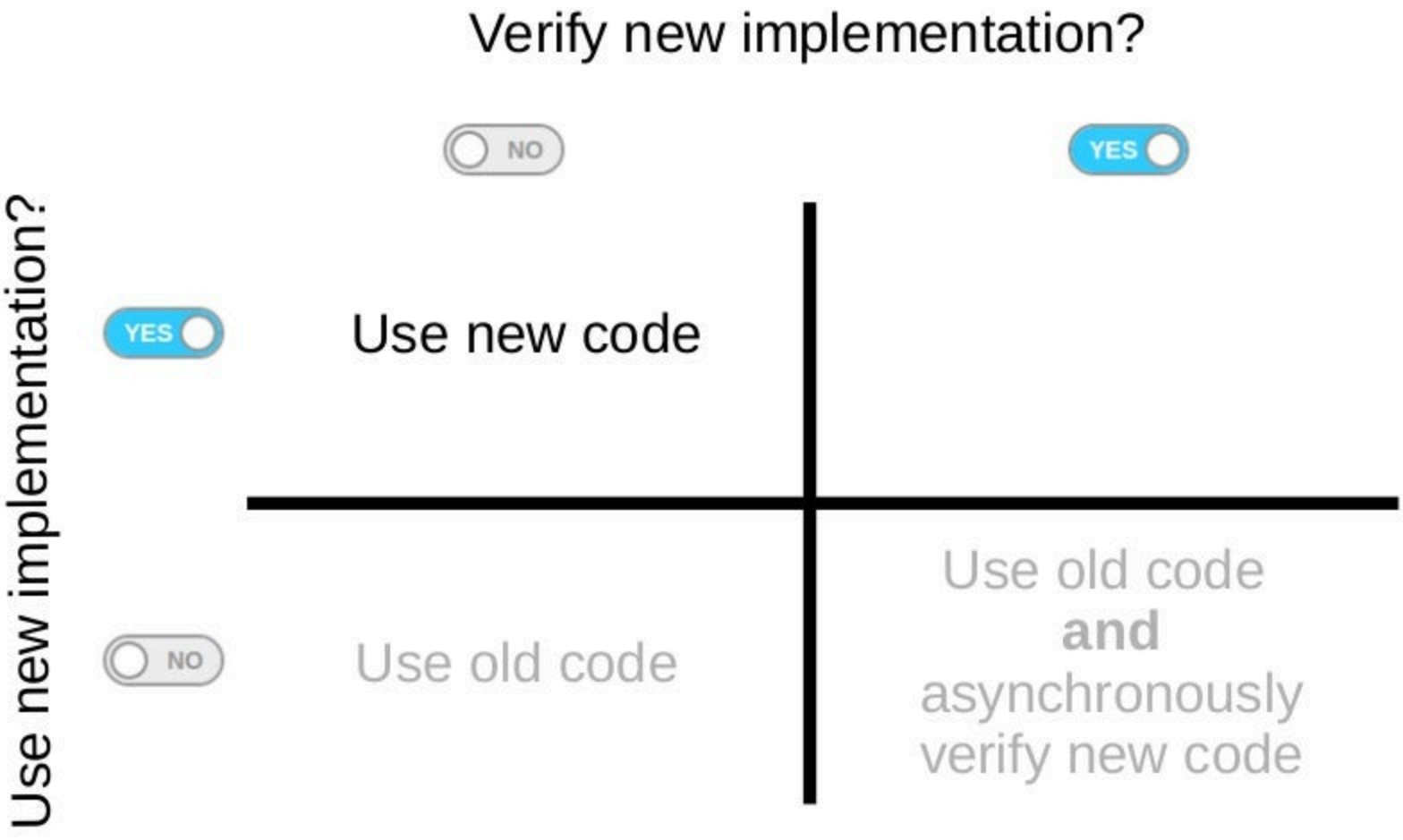
Implementation: 2 Feature Toggles



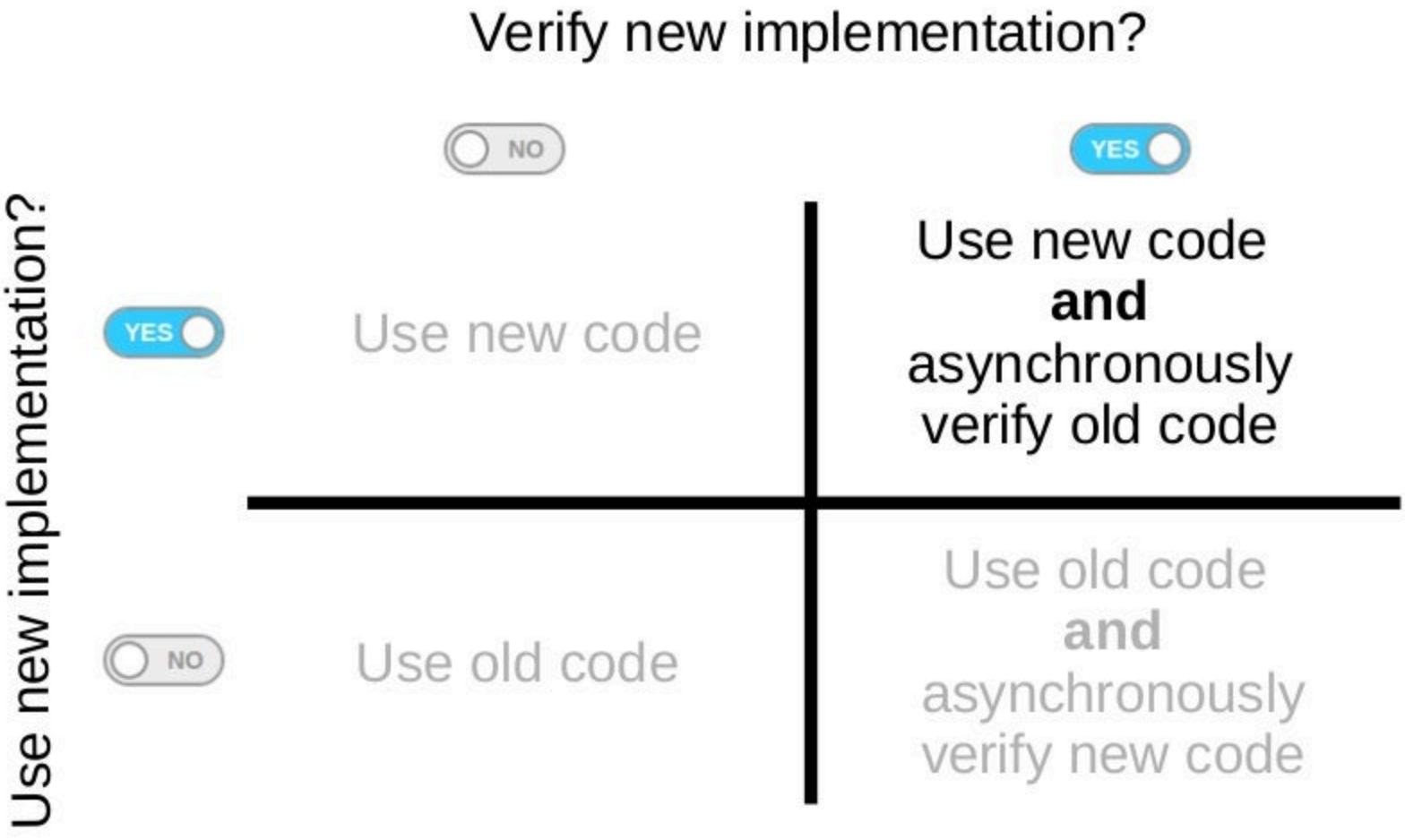
Implementation: 2 Feature Toggles



Implementation: 2 Feature Toggles



Implementation: 2 Feature Toggles



Verify Pattern (example)

```
public class OpenFactory {

    private static final String OPEN_CACHE_ID = "OpenFactory.open";
    private static final String HAS_OPENED_CACHE_ID = "OpenFactory.hasOpened";

    private static final Cache<Long, Open> CACHE = CacheFactory.createAndRegister(OPEN_
    private static final Cache<Long, Boolean> HAS_OPENED_CACHE = CacheFactory.createAnd
    private static final OpenDao DAO = new OpenDao();
    private static final OpenRestClient REST_CLIENT = new OpenRestClient();

    public static boolean hasOpened(long mailingToUserId, long mailingId) {
        Boolean hasOpened = HAS_OPENED_CACHE.get(mailingToUserId);
        if (hasOpened == null) {
            hasOpened = DAO.hasOpened(mailingToUserId);
            if (hasOpened) {
                HAS_OPENED_CACHE.put(mailingToUserId, true);
            } else {
                hasOpened = REST_CLIENT.hasOpened(mailingToUserId, mailingId);
                HAS_OPENED_CACHE.put(mailingToUserId, hasOpened);
            }
        }
        return hasOpened;
    }
```

Verify Pattern (example)

```
public class OpenFactory {

    private static OpenFactoryOldImpl c_oldImpl;
    private static OpenFactoryHBaseImpl c_newImpl;

    public static class Init {
        @Autowired
        public Init(OpenFactoryOldImpl oldImpl, OpenFactoryHBaseImpl newImpl) {
            c_oldImpl = oldImpl;
            c_newImpl = newImpl;
        }
    }

    public static boolean hasOpened(final long mailingToUserId, final long mailingId)
        return getVerifyStrategyForMailingId(mailingId).run(
            new TwoImplementations<Boolean>(OpenFactory.class, "hasOpened", mailingTo
                @Override public Boolean oldImpl() { return c_oldImpl.hasOpened(maili
                @Override public Boolean newImpl() { return c_newImpl.hasOpened(maili
            }
        );
    }
```

Verify Pattern (example)

- VerifyTwoImplementationsStrategy *(optivo.core.verify)*
  - UseNewImplVerifyOldImpl *(optivo.core.verify)*
  - UseOldImpl *(optivo.core.verify)*
  - UseNewImpl *(optivo.core.verify)*
  - UseOldImplVerifyNewImpl *(optivo.core.verify)*

```
public static boolean hasOpened(final long mailingToUserId, final long mailingId)
    return getVerifyStrategyForMailingId(mailingId).run(
        new TwoImplementations<Boolean>(OpenFactory.class, "hasOpened", mailingTo
            @Override public Boolean oldImpl() { return c_oldImpl.hasOpened(maili
            @Override public Boolean newImpl() { return c_newImpl.hasOpened(maili
        }
    );
}
```

Verify Pattern (example)

```
public abstract class TwoImplementations<T> {

    private final Object _this;
    private final Class _class;
    private final String _method;
    private final Object[] _methodParameters;

    /**...*/
    protected TwoImplementations(Object thiz, String method, Object... methodParameters) {...}

    public abstract T oldImpl();

    public abstract T newImpl();
```

```
public static boolean hasOpened(final long mailingToUserId, final long mailingId)
    return getVerifyStrategyForMailingId(mailingId).run(
        new TwoImplementations<Boolean>(OpenFactory.class, "hasOpened", mailingTo
            @Override public Boolean oldImpl() { return c_oldImpl.hasOpened(maili
            @Override public Boolean newImpl() { return c_newImpl.hasOpened(maili
        }
    );
}
```

Verify Pattern (example)

```
public class OpenFactory {

    private static OpenFactoryHBaseImpl c_newImpl;

    public static class Init {
        @Autowired
        public Init(OpenFactoryHBaseImpl newImpl) {
            c_newImpl = newImpl;
        }
    }

    public static boolean hasOpened(long mailingToUserId, long mailingId) {
        return c_newImpl.hasOpened(mailingToUserId);
    }
```

Why Feature Toggles?

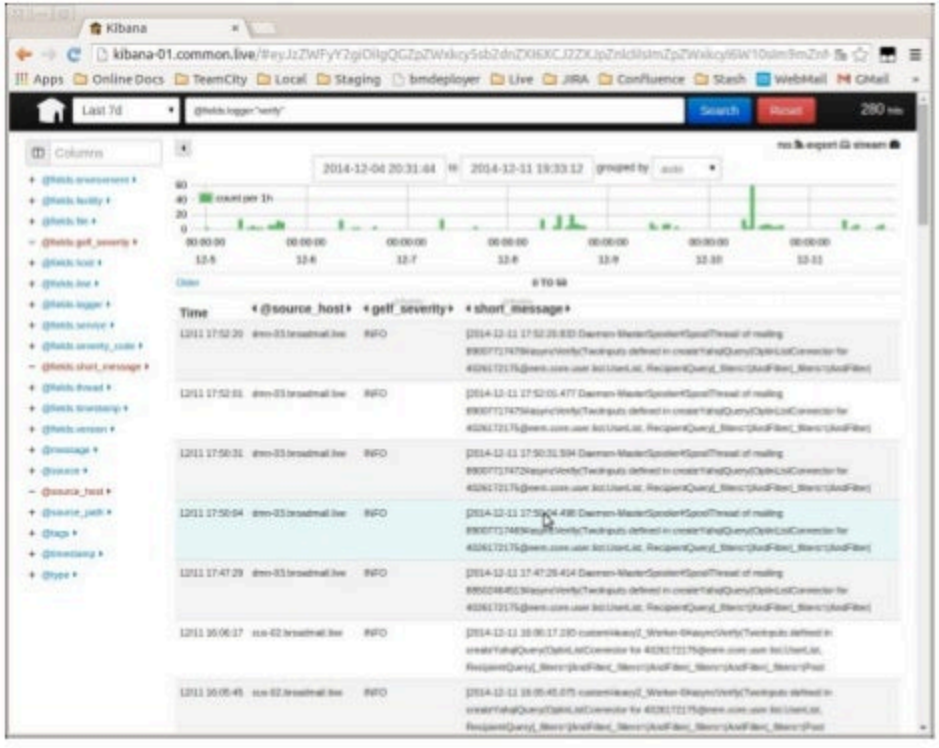
- ✓ Hide unfinished features from users
- ✓ Separate Deployment and Release
- ✓ Fearless Deployment
- ✓ No Big Bang Releases
- ✓ Better Software Architecture
- ✓ Testing in Live Environment with Live Data

Some Questions Remain

- What should the system do
  if an asynchronous verify fails?
- Which library should you use
  for Feature Toggles?
  (Should you use a library at all?)
- When do you delete
  old Feature Toggles?
- How do you do testing with
  Feature Toggles?
- ...

What should the system do if an asynchronous verify fails?

Logging



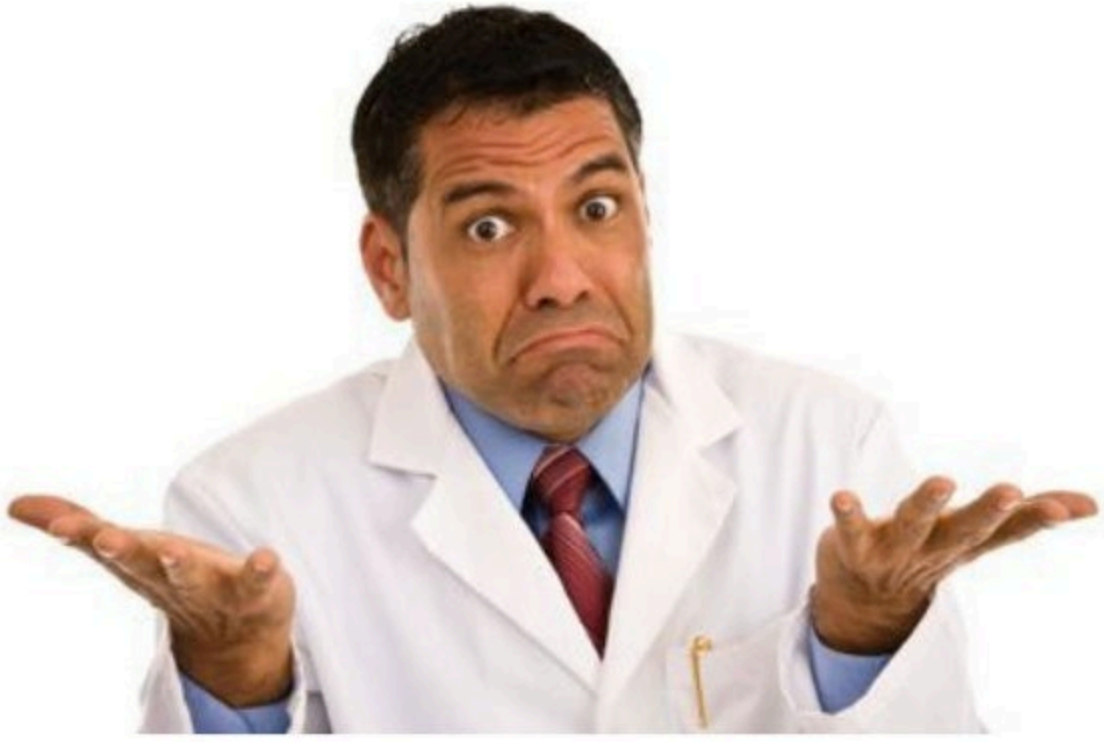
Metriken

What should the system do if an asynchronous verify fails?

Logging

Metriken

Which library should you use for Feature Toggles?



But here is a tip ...

Don't use Strings, Use an Enum.

 Prevents typing errors

 Code completion

 You can easily add documentation

 Makes it easy to remove a Feature Toggle

When do you delete old Feature Toggles?

When you are confident

When do you delete old Feature Toggles?

When you are confident

(probably serveral weeks after you activated the Feature Toggle globally)

Testing with Feature Toggles – Combinatorial Explosion?

Testing with Feature Toggles (example #1)

```
public class InnerJoinIntegrationTest {

    @Rule
    public YahqlIntegrationTestFixture _testFixture = new YahqlIntegrationTestFixture();

    @Test
    @RunTwiceToggleFeature(YahqlFeature.USE_NEW_TUPLE_LIST_IMPL)
    public void test() throws Exception {
```

Testing with Feature Toggles (example #1)

```
public class YahqlIntegrationTestFixture extends HadoopTestFixture {

    private static final Logger LOG = Logger.getLogger(YahqlIntegrationTestFixture.class);

    private final Map<YahqlFeature, Boolean> _features = new HashMap<YahqlFeature, Boolean>();
    private boolean _createJobConfCalled;

    @Override
    public Statement apply(Statement base, FrameworkMethod testMethod, Object testInstance) {
        final RunTwiceToggleFeature runTwiceToggleFeature = testMethod.getAnnotation(RunTwiceToggleFeature.class);
        if (runTwiceToggleFeature == null) {
            return super.apply(base, testMethod, testInstance);
        } else {
            final Statement runTest = super.apply(base, testMethod, testInstance);
            return new Statement() { @Override public void evaluate() throws Throwable {
                YahqlFeature feature = runTwiceToggleFeature.value();
                _features.put(feature, false);
                LOG.info("Testing with disabled YahqlFeature." + feature + " ...");
                runTest.evaluate();
                if (!_createJobConfCalled) {
                    throw new AssertionError("YahqlIntegrationTestFixture.createJobConf() was not called -- @RunTwiceToggleFeature annotation has no effect");
                }
                _features.put(feature, true);
                LOG.info("Testing with enabled YahqlFeature." + feature + " ...");
                runTest.evaluate();
            }};
        }
    }
```

Testing with Feature Toggles (example #2)

GitHub Gist Search... All Gists

MichaelTamm / **ToggleFeaturesRunner.java**
Created 14 minutes ago

JUnit 4 Runner to run a test with all possible feature toggle combinations

ToggleFeaturesRunner.java

```
public enum Feature {
    coolFeature1, awesomeFeature2, greatFeature3
}
```

Then you could write the following test class:

```
@RunWith(ToggleFeaturesRunner.class)
public class FooTest {
    @Test
    @ToggleFeatures({ coolFeature1, awesomeFeature2 })
    public void test() {
        // Setup test environment using ToggleFeaturesRunner.getCurrentFeatureCombination()
        // ...
    }
}
```

and the test method would be called 4 times for all possible combinations for the 2 specified features.

Quelle: https://gist.github.com/MichaelTamm/d5a5a67460d7a2db08a4

Testing with Feature Toggles (example #3)

```
@RunWith(Suite.class)
@SuiteClasses({
    ReceivedMailingFilterIntegrationTest.class,
    OpenedWithinLastXDaysFilterIntegrationTest.class,
    ClickedCategoryMailingFilterIntegrationTest.class,
    IsInCampaignFilterIntegrationTest.class
})
public class RecipientQueryToYahqlQueryConverter3RegressionTest {

    private static final AfterCreateHook AFTER_CREATE_HOOK = new AfterCreateHook() { @Override public void afterCreate(Object o) thr
        // Make sure the useRecipientQueryToYahqlQueryConverter3 feature switch is enabled for every created mailing group ...
        if (o instanceof MailingGroup) {
            MailingGroupConfig config = ((MailingGroup) o).getConfig();
            config.getCustomConfig().setFeatureSwitch(Feature.useRecipientQueryToYahqlQueryConverter3, true);
            try { config.save(); } catch (Exception e) { throw Throwables.propagate(e); }
        }
    }};

    @BeforeClass
    public static void setUp() {
        SharedBroadmailIntegrationTestFixture.setSpringConfigLocationFactory(new Function<Class<?>, String>() { @Override public Str
            return "broadmail/recipientdatareplicator/beansRegressionTesting.xml";
        }});
        TestDataCreators.addAfterCreateHook(AFTER_CREATE_HOOK);
    }

    @AfterClass
    public static void tearDown() {
        SharedBroadmailIntegrationTestFixture.resetSpringConfigLocationFactory();
        TestDataCreators.removeAfterCreateHook(AFTER_CREATE_HOOK);
    }
```

Main Takeaways

- Use short lived issue or personal branches
- Use fine-grained Feature Toggles
- Verify new implementations
  - asynchronously in the background on live systems
  - in the background, but blocking on test systems
- Choose names for Feature Toggles with care
- Use Enums for Feature Toggles