Spark Your Legacy:

How to distribute your 8-year old monolith

Moran Tavori, Tzach Zohar // Kenshoo // June 2016

Kenshoo™

Who’s this talk for?

Who are we?

Tzach Zohar, System Architect @ Kenshoo

Moran Tavori, Lead backend developer @ Kenshoo

Working with Spark for ~2.5 years

Started with Spark version 1.0.x

Who's Kenshoo

10-year Tel Aviv-based startup

Industry Leader in Digital Marketing

500+ employees

Heavy data shop

The Problem

Legacy “batch job” in Monolith

Job performs aggregations applying complex business rules

Monolith is a Java application running hundreds of types of “jobs” (threads)

Tight coupling between jobs (same codebase, shared state)

Sharded by client

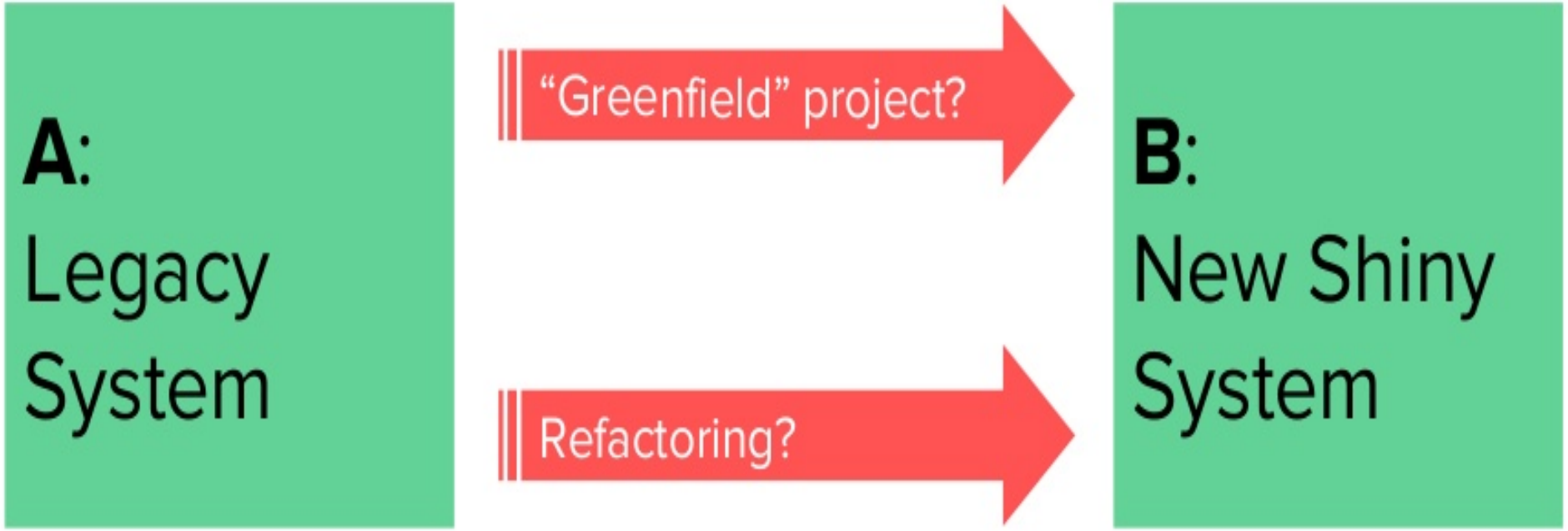
Doesn’t scale

Solution: Spark!

Spark elegantly solves the business case for that job, as proven by POC

> "*API well suited for our use case*"
>
> "*Very little boilerplate / plumbing code*"
>
> "*Testable*"
>
> - from POC conclusions

The “Greenfield” Dilemma




How do we make the “jump”?

Mitigating “Greenfield” risks

Problem #1:
Code is our only Spec

Code is our only Spec

What **exactly** should the new system do?

```
if (channel == null) { // missing channel! (shouldn't happen)
    // we'll return a best guess
    if (!profile.getSEChannels().isEmpty())  // prefer SE channels if exist
        channel = profile.getSEChannels().values().iterator().next();
    else if (!profile.getChannels().isEmpty()) // otherwise - just get the first
        channel = profile.getChannels().values().iterator().next();
    else // you leave me with no choice, mister!
        throw new RuntimeException("profile has no channels! can't process.");
}
```

Code is our only Spec

What **exactly** should the new system do?

```
if (channel == null) { // missing channel! (shouldn't happen)
    // we'll return a best guess
    if (!profile.getSEChannels().isEmpty())  // prefer SE channels if exist
        channel = profile.getSEChannels().values().iterator().next();
    else if (!profile.getChannels().isEmpty()) // otherwise - just get the first
        channel = profile.getChannels().values().iterator().next();
    else // you leave me with no choice, mister!
        throw new RuntimeException("profile has no channels! can't process.");
}
```


Don’t Assume.
Measure.
- Kenshoo developers, circa 2014

Solution #1: Empirical Reverse Engineering

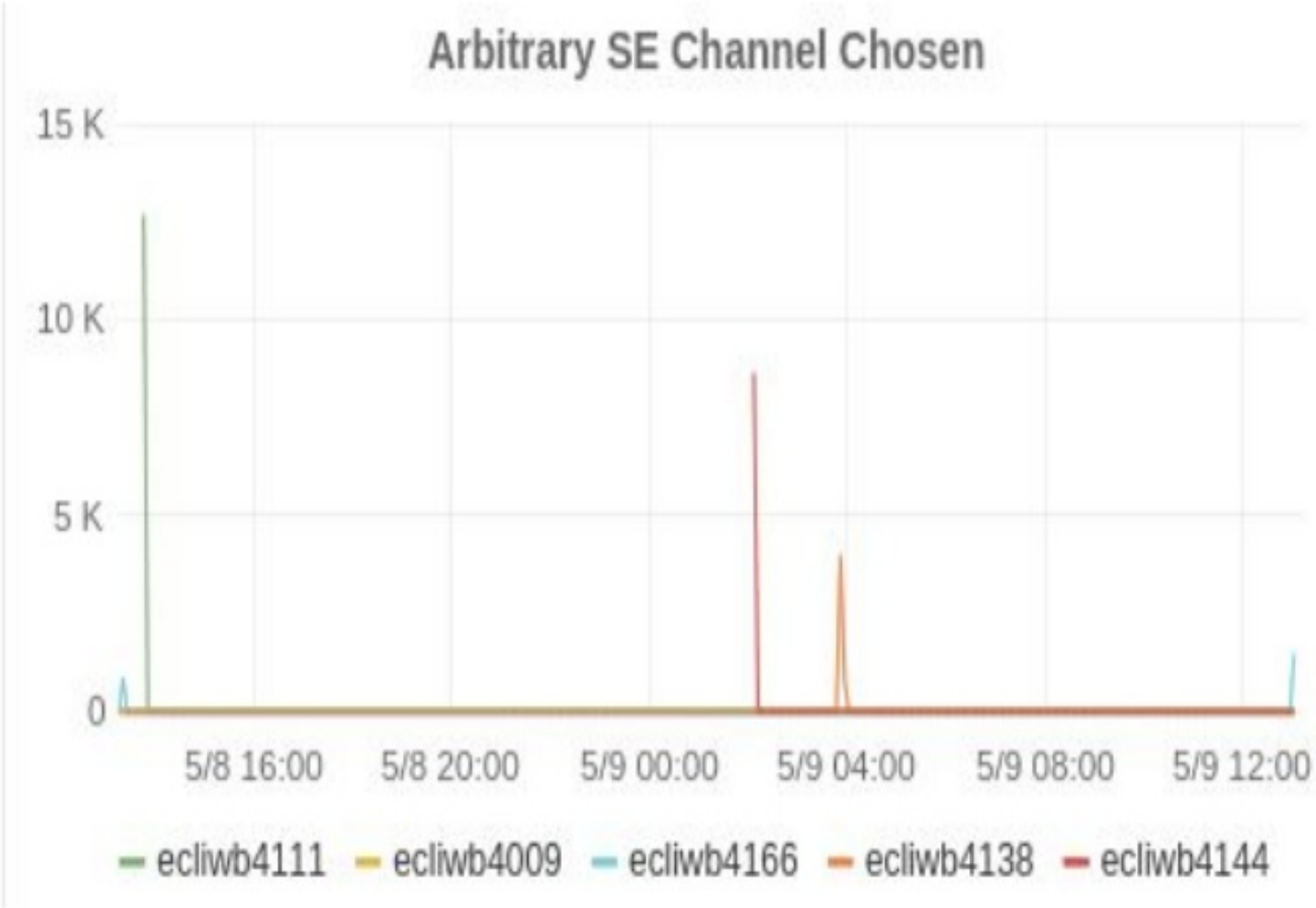
```
if (channel == null) { // missing channel! (shouldn't happen)
    // we'll return a best guess
    if (!profile.getSEChannels().isEmpty())  // prefer SE channels if exist
        channel = profile.getSEChannels().values().iterator().next();
    else if (!profile.getChannels().isEmpty()) // otherwise - just get the first
        channel = profile.getChannels().values().iterator().next();
    else // you leave me with no choice, mister!
        throw new RuntimeException("profile has no channels! can't process.");
}
```

Solution #1: Empirical Reverse Engineering

```
if (channel == null) { // missing channel! (shouldn't happen)
    // we'll return a best guess
    if (!profile.getSEChannels().isEmpty())  // prefer SE channels if exist
        channel = profile.getSEChannels().values().iterator().next();
    else if (!profile.getChannels().isEmpty()) // otherwise - just get the first
        channel = profile.getChannels().values().iterator().next();
    else // you leave me with no choice mister!
        throw new RuntimeException("profile has no channels! can't process.");
}
```

```
// try some SE channels if exist
ScoreProviderData searchEngineData = getAnySearchEngineChannelData();
if (searchEngineData != null) {
    // todo - remove if we don't see any usage in production
    logger.error("could not find observation channel (id = {}), using arbitrary search-engine channel
    Metrics.newCounter(this.getClass(), "flatten.using.arbitrary.se.channel").inc();
    return searchEngineData;
}
```

Solution #1: Empirical Reverse Engineering

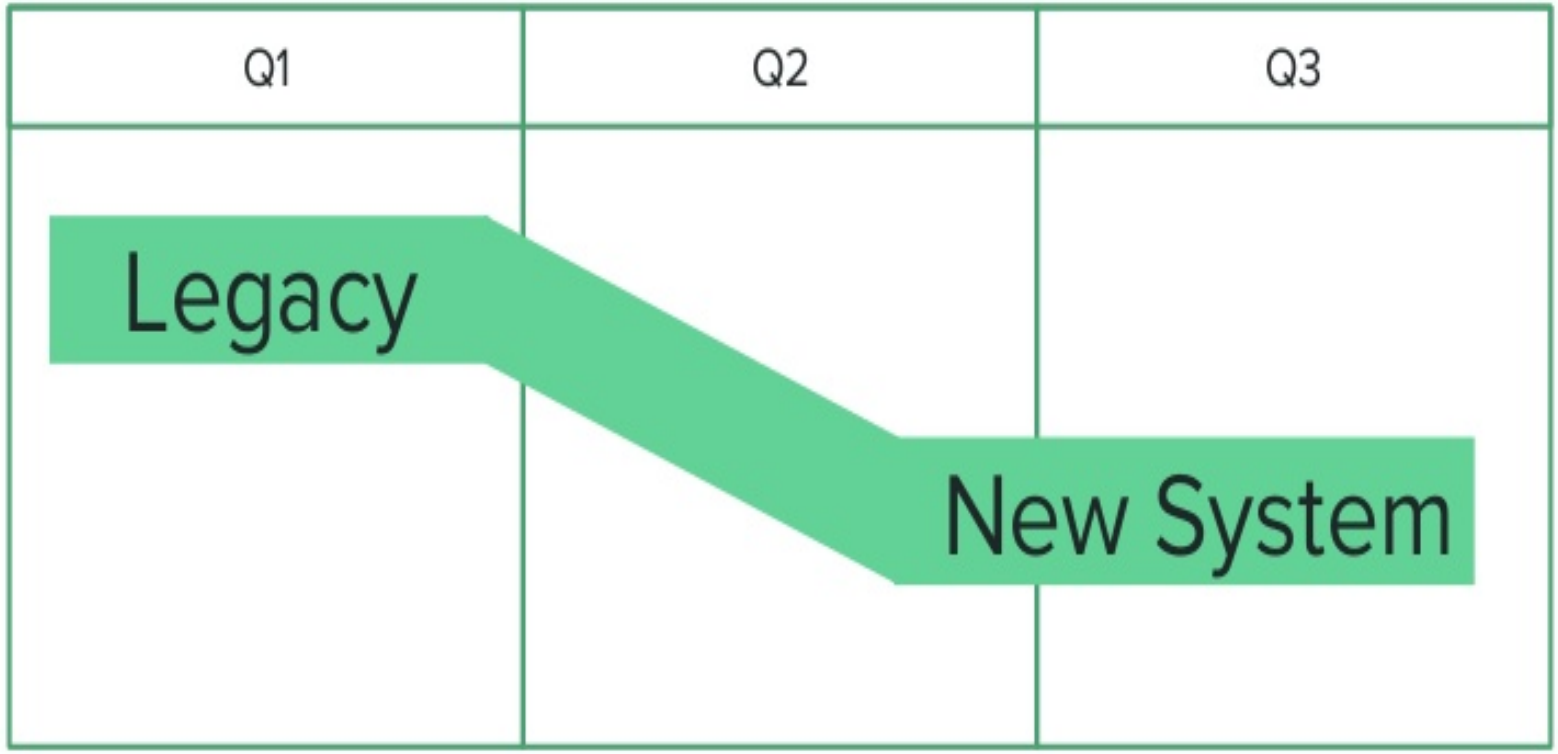


Problem #2: Moving Target

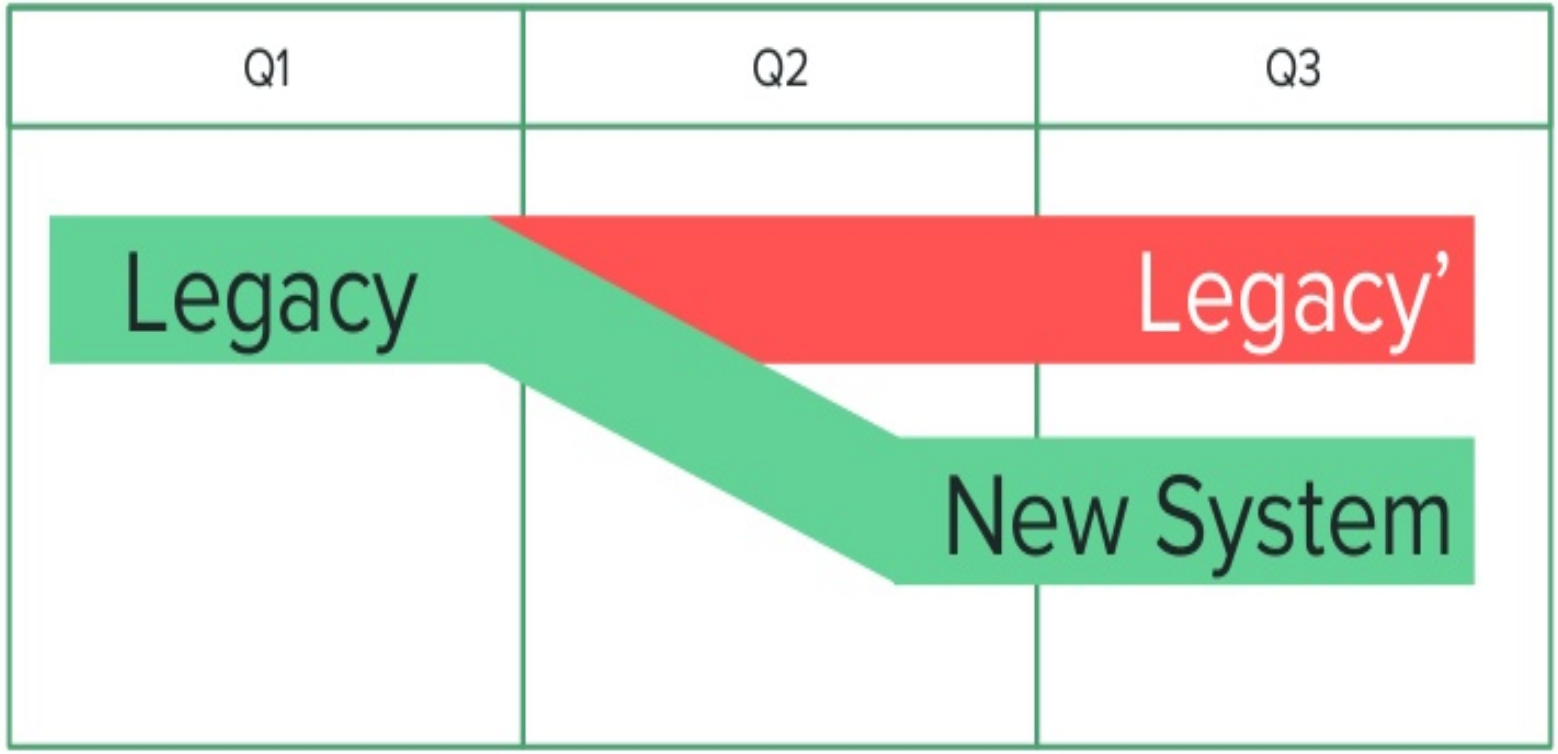
Moving Target

Q1	Q2	Q3
Legacy		

Moving Target



Moving Target

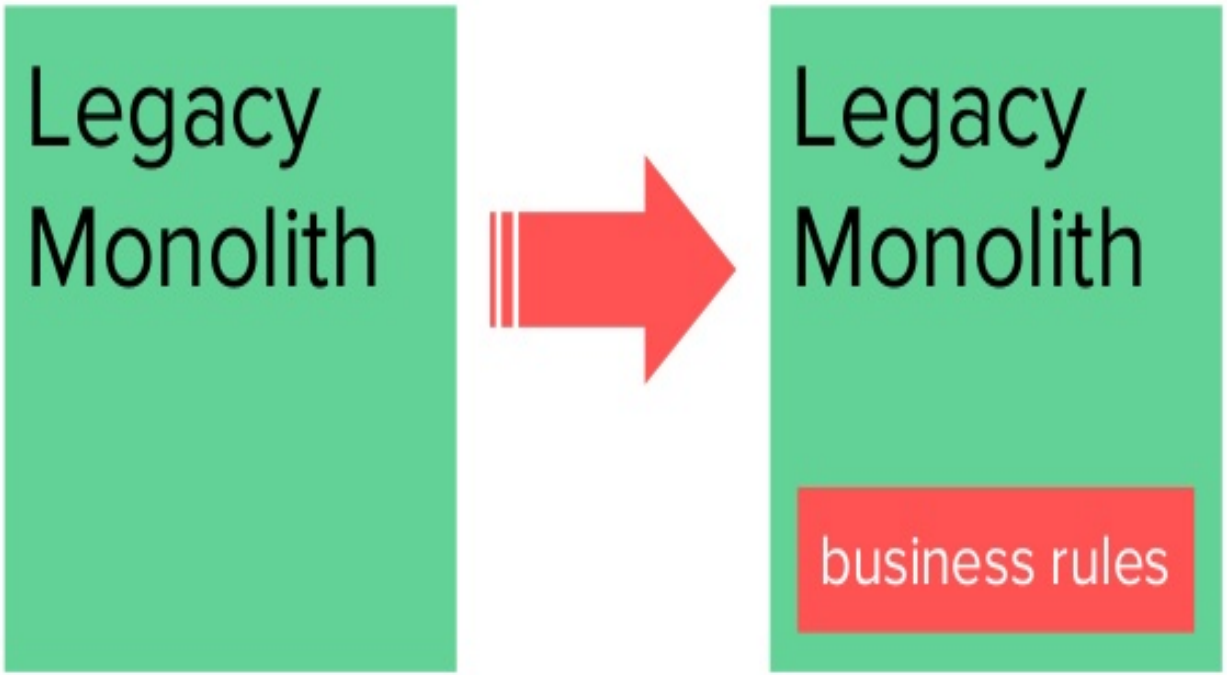


Solution #2: Share Code

1. Refactor legacy code to isolate business rules in separate **jar**

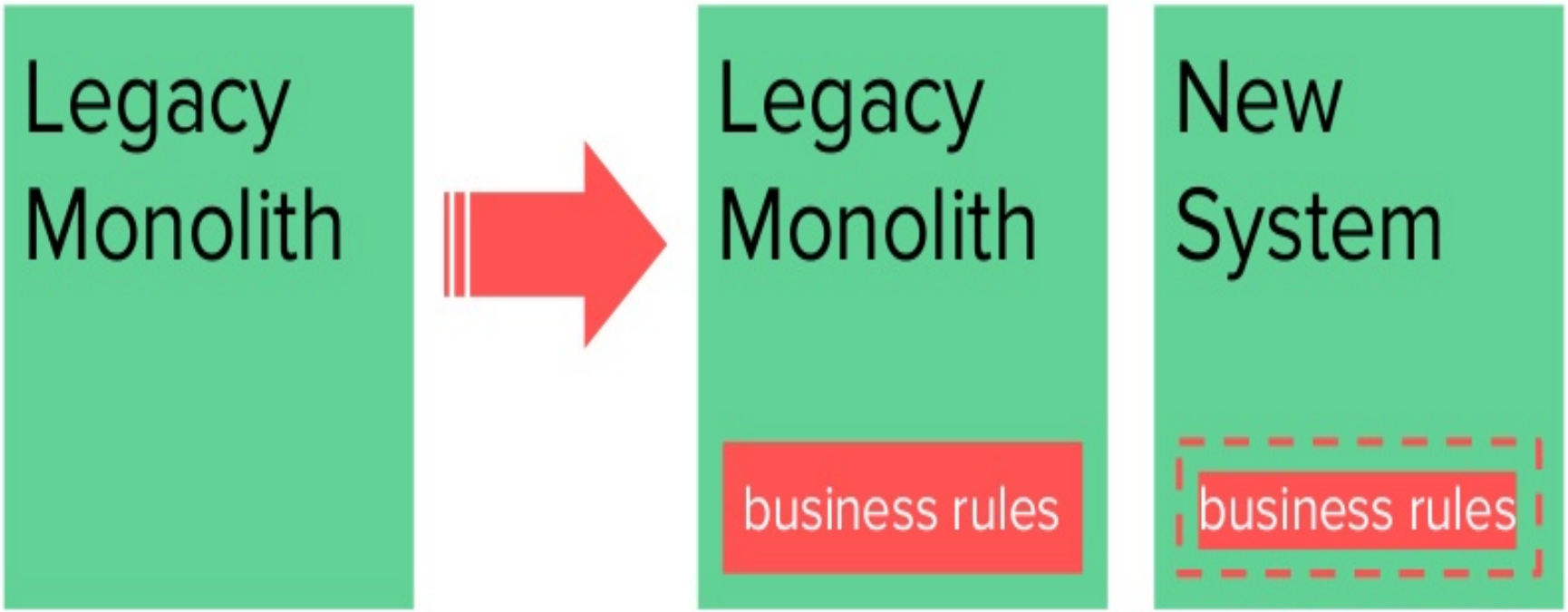
Solution #2: Share Code

1. Refactor legacy code to isolate business rules in separate **jar**



Solution #2: Share Code

1. Refactor legacy code to isolate business rules in separate **jar**
2. Build new system around this shared jar



Solution #2: Share Code

```
List<Score> filtered = new LinkedList<>();
ScoreProviderData providerData = scoreProviderDao.getByScore(scores);
for (Score s : scores) {
   if (validProviderForScore(s, providerData)) {
      ScoreSource providerSource = providerData.getSource();
      if (providerSource == s.getSource()) {
         filtered.add(s);
      }
   }
}
```

Solution #2: Share Code

```
public boolean shouldAggregateScore(ShouldAggregateKey key) { ... }


List<Score> filtered = new LinkedList<>();
for (Score s : Scores) {
   if (shouldAggregateScore(key(s)) {
      filtered.add(s);
   }
}
```

Solution #2: Share Code

```
public boolean shouldAggregateScore(ShouldAggregateKey key) { ... }
```

```
val scores: RDD[S] = // ...

val filtered: RDD[S] = scores.filter(s => shouldAggregateScore(key(s)))
```

Problem #3:
Zero Diff Tolerance

Zero Diff Tolerance

Some downstream modules might be sensitive to any new behavior

Solution #3: Run Side-by-Side with Legacy

At the **system** level:

mysql_sum_conv	gp_sum_conv	mysql_sum_commission	gp_sum_commission
1233	1233	200297	200296.99802
1236	1236	196135.25	196135.24406
51	51	8615.00488	8615.00502
6	4	590	181.95
4	4	965.33334	181.95
4	4	148.99	181.95
0	4	16.66667	181.95
0	4	7	181.95
2	4	99.66667	181.95
2	4	199.94001	181.95
2	4	304.9	181.95
1	4	29.975	181.95
1	4	76	181.95

Solution #3: Run Side-by-Side with Legacy

... and at the **component** level:

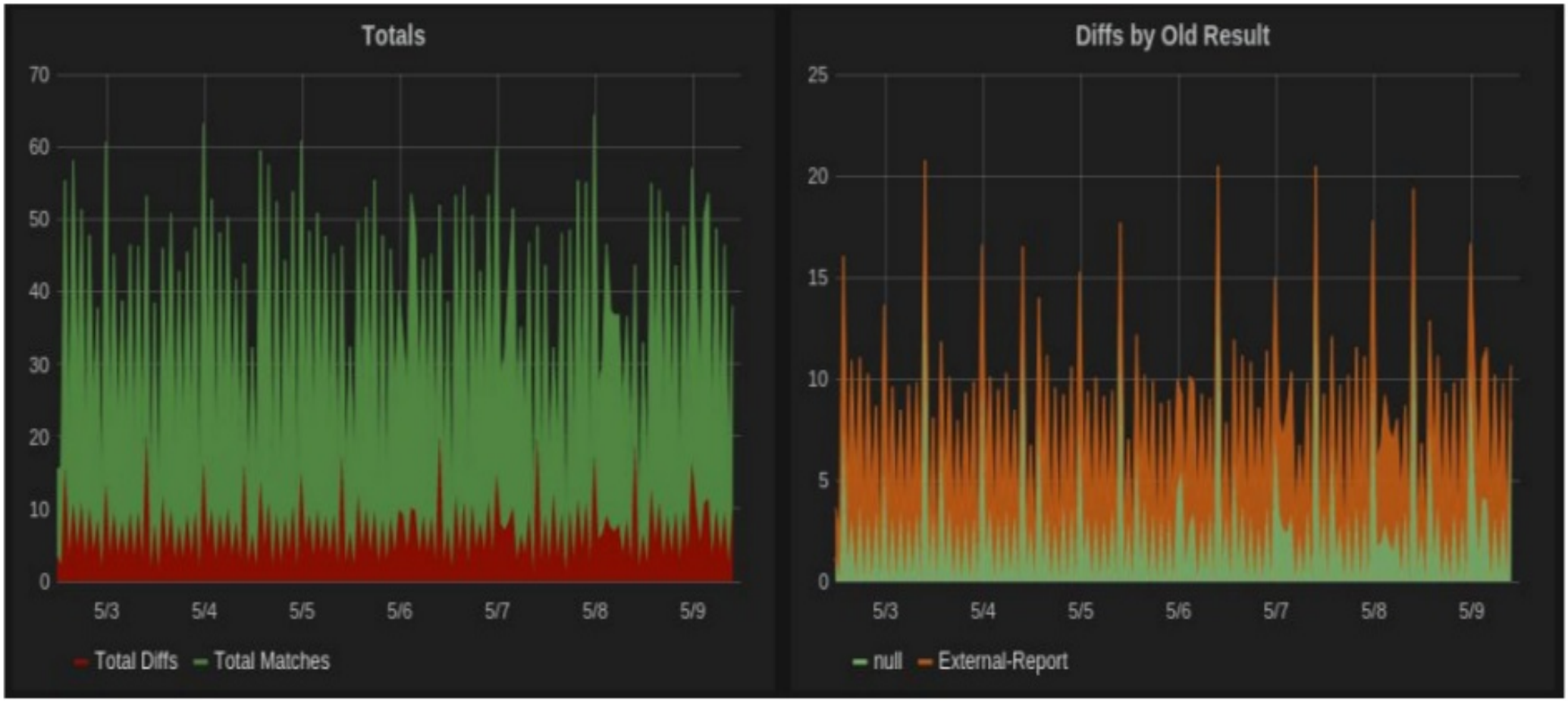
```
public final class CompositeDesiredSourceResolver implements DesiredSourceResolver {

    private final DesiredSourceResolver active;
    private final DesiredSourceResolver passive;
    private final DiffRecorder diffRecorder;

    @Override
    public ScoreSource getDesiredSource(ScoreSourcesKey key) {
        ScoreSource activeResult = active.getDesiredSource(key);
        ScoreSource passiveResult = passive.getDesiredSource(key);
        diffRecorder.recordDiff(key, activeResult, passiveResult);
        return activeResult;
    }
```

Solution #3: Run Side-by-Side with Legacy

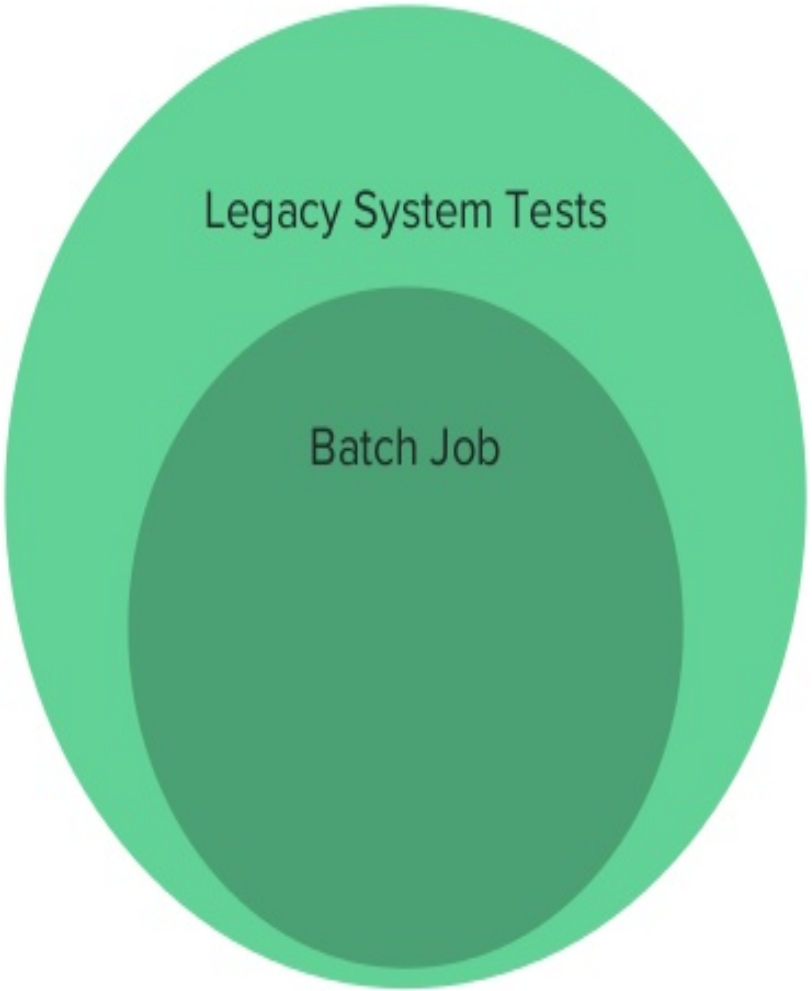
At the **component** level:



Problem #4:
Test Reuse

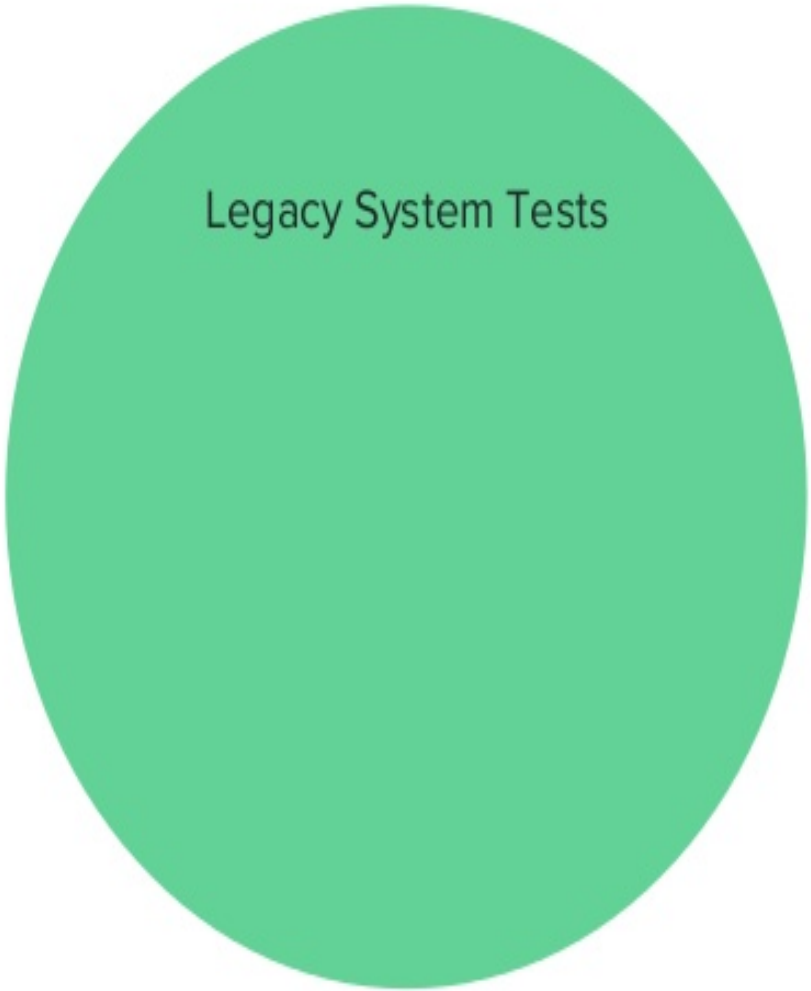
Test Reuse

Before



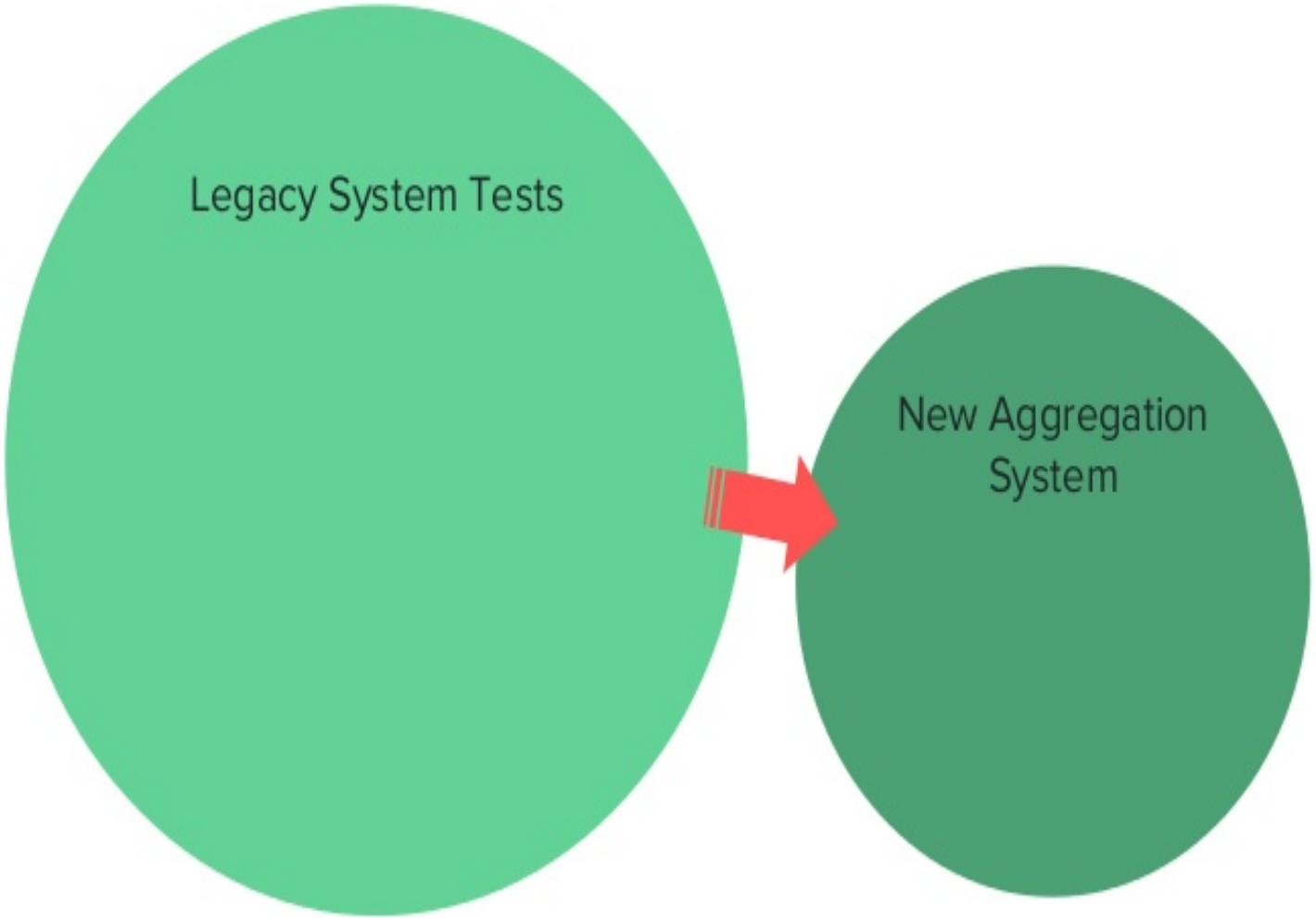
Test Reuse

After



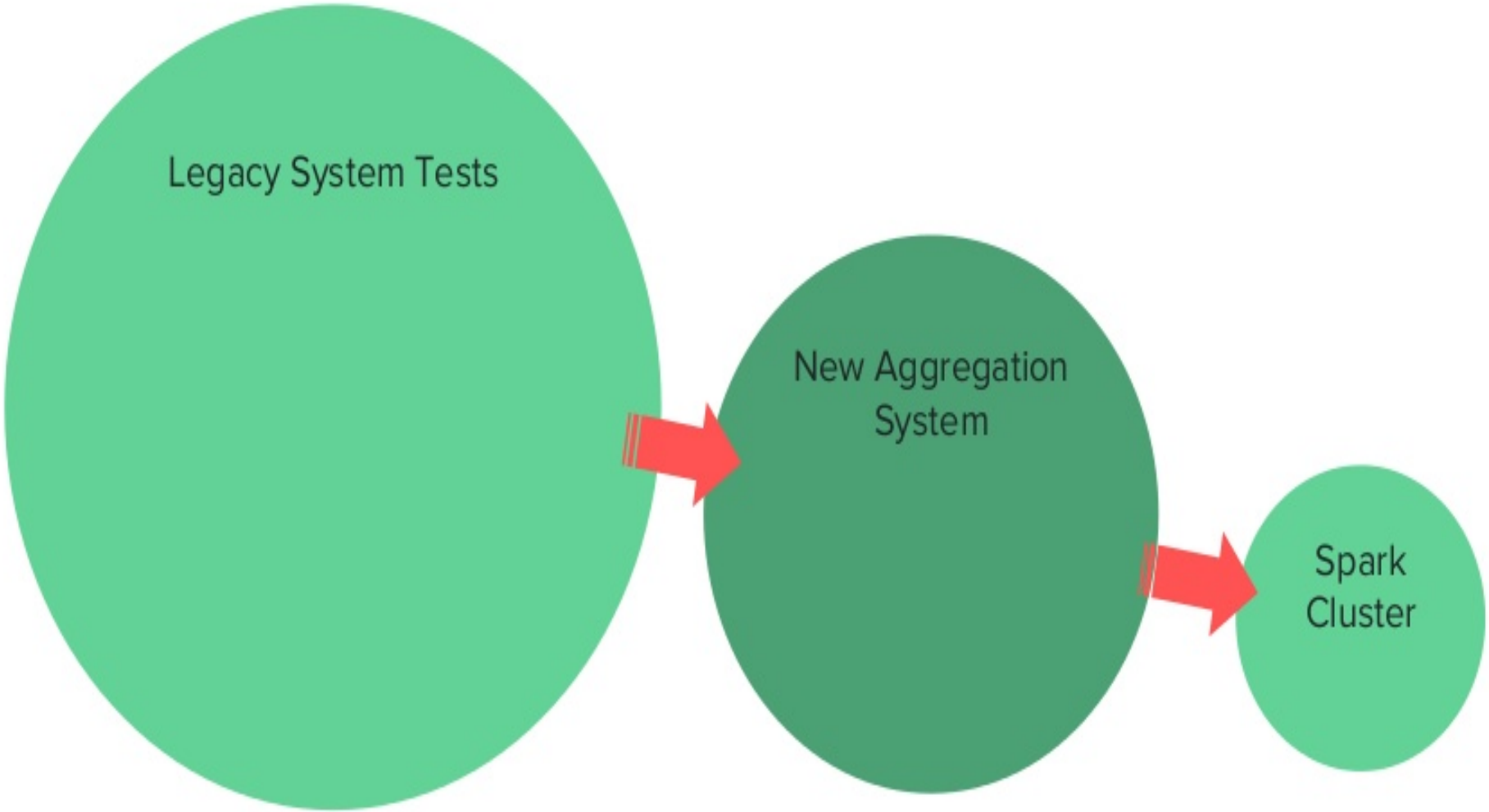
Test Reuse

After

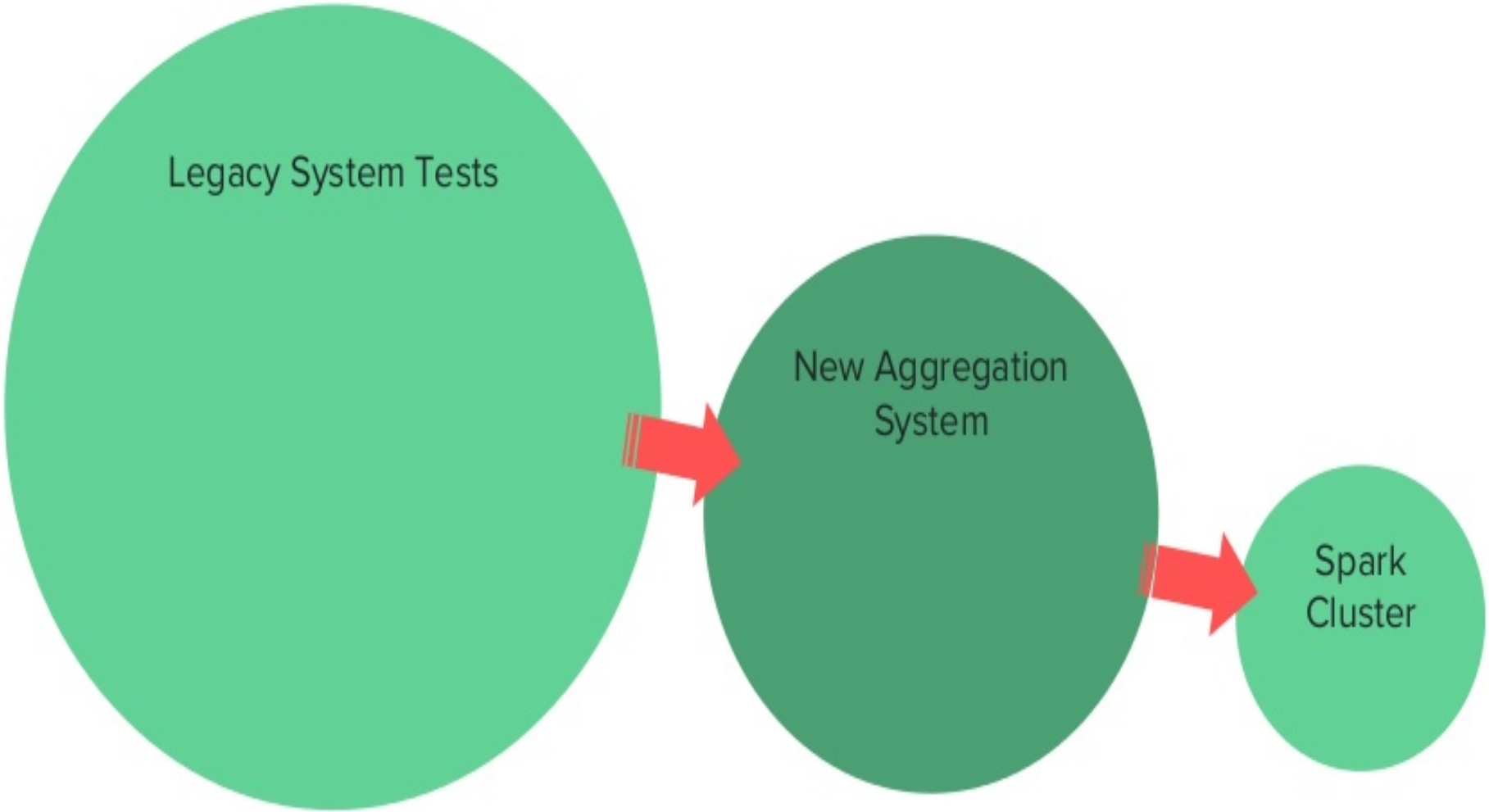


Test Reuse

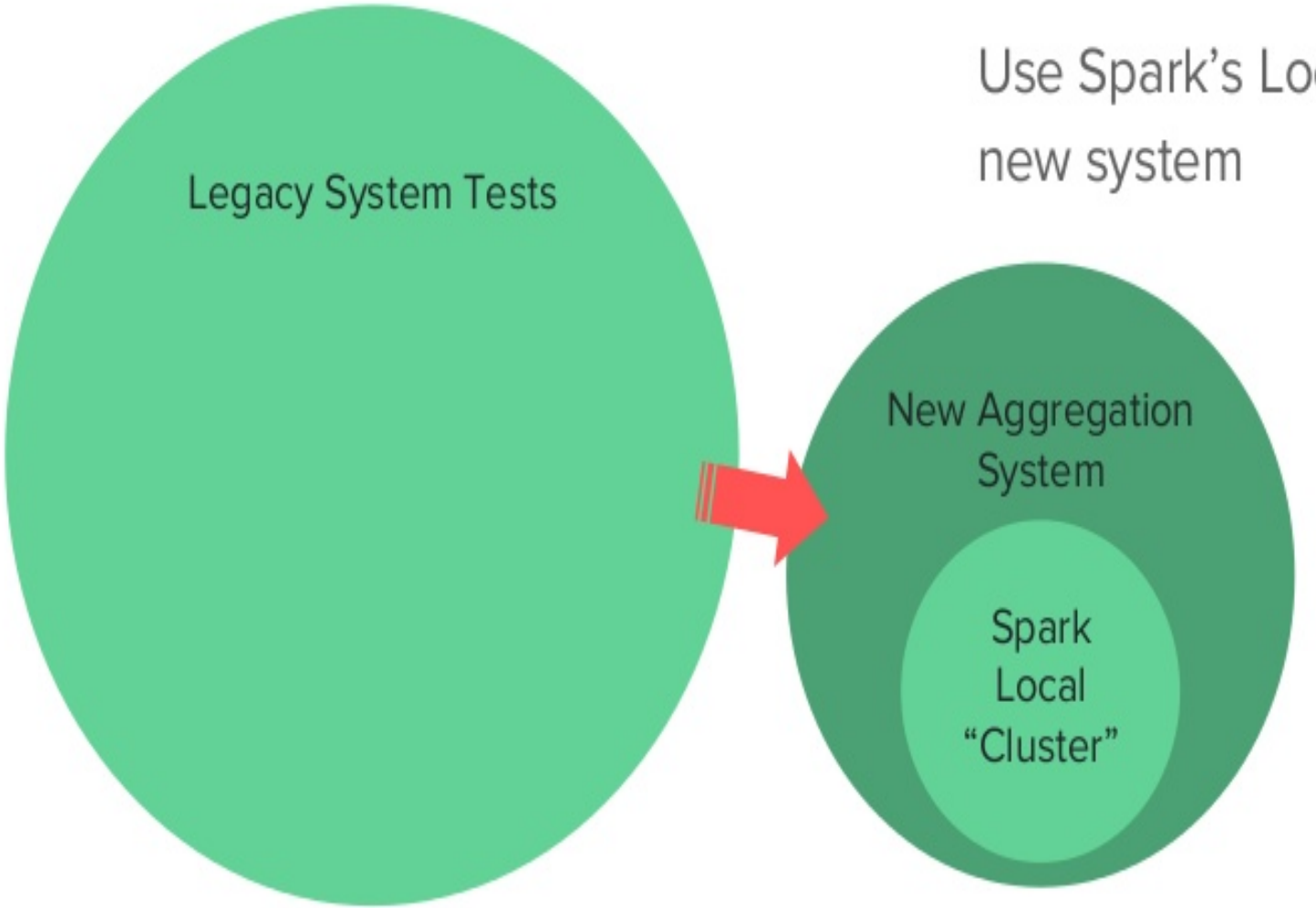
After



Solution #4: Local Mode

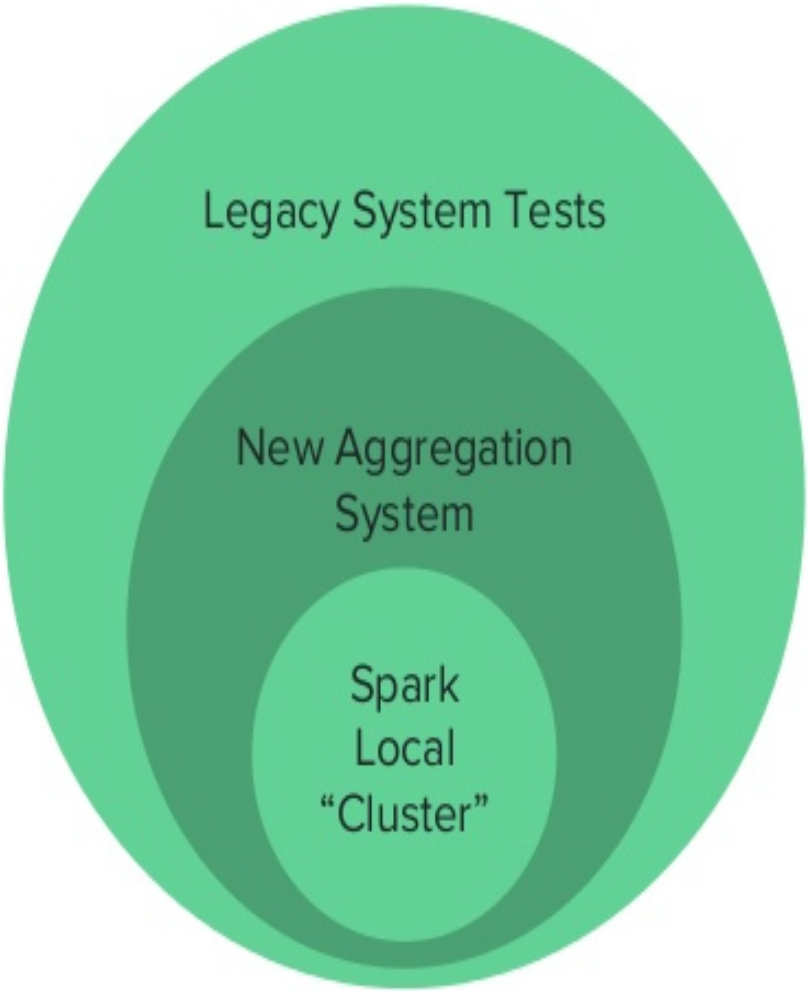


Solution #4: Local Mode



Use Spark's Local Mode to embed it in new system

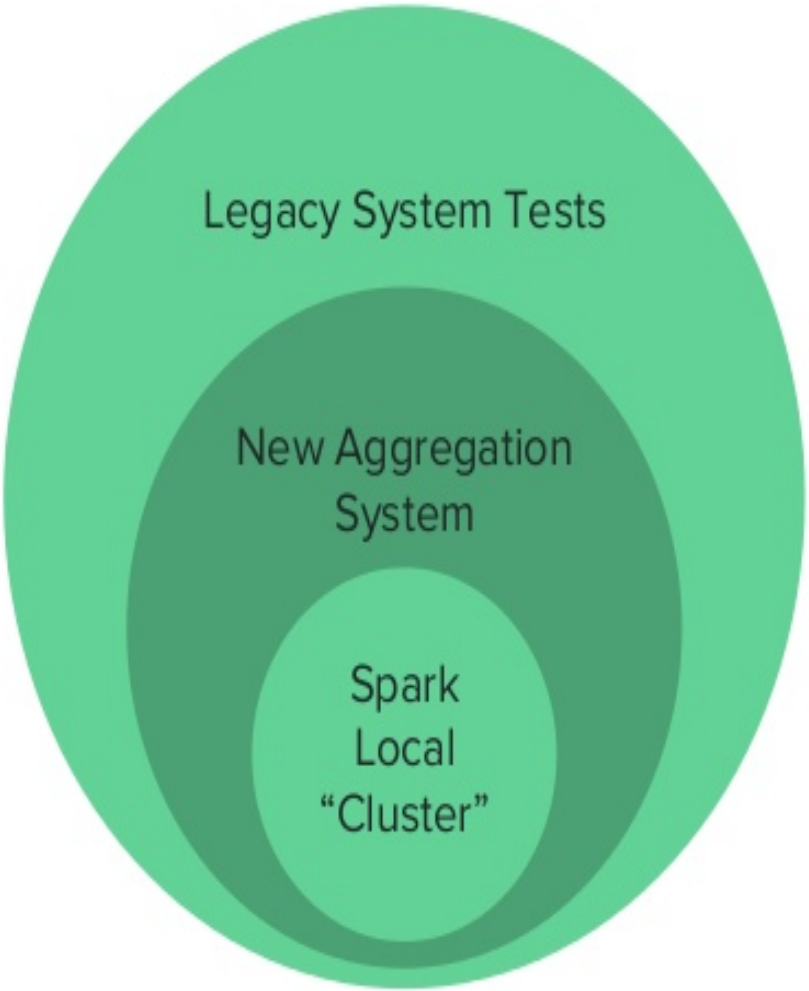
Solution #4: Local Mode



Use Spark's Local Mode to embed it in new system

Use new system's "Local Mode" to embed it in legacy system

Solution #4: Local Mode



Use Spark's Local Mode to embed it in new system

Use new system's "Local Mode" to embed it in legacy system

Ta Da! No test setup

In Conclusion

Spark's fluent APIs made it possible to share code with the old system

Spark's local mode made testing easier

Common agile practices gave us control over the results before our system was client facing


Make your Greenfield special.

Thank You

Questions?