

Apache Ignite · Reducing the dependency loop inside ignite-core from 2 476 classes to 483 using CodeLaser

`ignite-core` is one artifact of 5 632 classes, and 2 476 of them depend on each other in a loop. This is an account of reducing that loop to 483 classes with the CodeLaser refactoring engine, and of separating one Maven module out of the artifact on the way. It was measured as the work happened. It is an illustration of how the engine is used, written for anyone interested in the mechanics of the work or in the internals of Apache Ignite. It says what was done and in what order, what changed in the public packages and why, how each step was checked, and what the code now allows that it did not before.

SUBJECT

`ignite-core`, Apache Ignite
2.x, upstream commit
`a5b6b0dc44`

CONTACT

`bart.naudts@codelaser.io`

TABLE OF CONTENTS

1. The loop reduced from 2 476 classes to 483 and one module separated out
 2. The project setup
 3. How the CodeLaser engine works
 4. What a build boundary requires, and why the core cannot simply be cut
 5. The loop each step of the work went through
 6. Stepwise overview of the work
 7. The eleven steps, in more detail
 8. What changed in the public packages
 9. One separated module and five build units the code now allows
 10. How each step was checked
 11. Three things in Ignite's code found along the way
 12. Effort required to compute and perform the work
- A. What was not done, and why
 - B. How to read the figures

IN SHORT

1. The loop reduced from 2 476 classes to 483 and one module separated out

- **The goal.** `ignite-core` is one artifact. Anything that uses any part of it compiles against all of it. The goal was to find out how far the core can be separated into modules that build in a fixed order, without changing what a node does. The second goal was to do as much of it as the code allows.
- **The obstacle.** 2 476 classes in the core depend on each other in a loop. A module has to be built before whatever uses it. Inside a loop there is no first, so no boundary can be drawn anywhere in it. Every one of those classes lives in `core/main`.
- **What was done.** The loop was reduced in 65 commits, from 2 476 classes to 483. Most of them are small: a class looked up by name, a constant moved to the class that owns it, an interface declared for what a caller uses. One is large: the kernal context was split into a lower interface and an upper one, which took 27 commits and removed 985 references from the lower half of the core into the cache engine.
- **One module separated.** The management command tree, 520 classes, is now `ignite-management`, a Maven module built after the core. 299 files moved without changing package, and the core shrank from 5 632 to 5 112 classes. Step 3 gives the bill.
- **What the code now allows.** With the kernal context split, the core has no reference from its lower half into the cache engine. Measured on the whole of `core/main` when the split closed, five build units follow from that boundary, every dependency between them pointing downward. They are measured, not yet cut. Section 9.
- **What changed in the public packages.** No public type changed package. Sixteen declarations in public packages changed, all of them signatures that already named an internal type. Two are extension points that an implementation outside the repository would have to follow. Section 8.
- **How it was checked.** Every commit builds the whole reactor with Ignite's checkstyle at zero violations in all 43 modules. Every change was tested by the suites that exercise it, and most were tested a second time with the change broken on purpose, to prove the suite can tell. The whole suite was not run. Section 10 says exactly what was and was not covered.
- **How it was done.** The CodeLaser engine holds a model of the source. It answers questions about it with counts, prices a proposed change before it is made, and applies changes as operations. Claude Code works from those measurements, predicts what a step will do to the loop, and measures again after the step lands. Every step's prediction is on the record beside its measurement.

- **Where it ends up.** 483 classes in the loop, and a priced road onward: thirteen further subsystem moves that would take it to about 260, each paying on its own. Section 9.



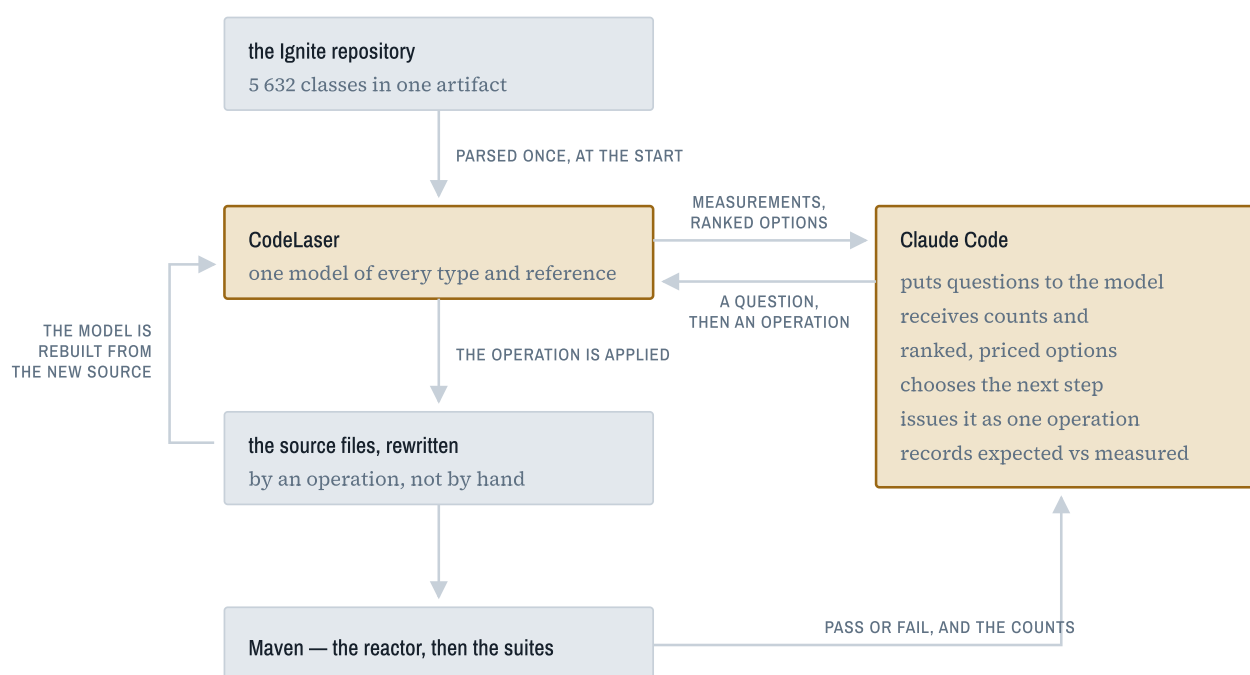
THE SETUP

2. The project setup

Four components were involved: the Ignite source, the CodeLaser engine, Claude Code, and the Maven build.

In the rest of this document, **the core** means `ignite-core`, the artifact this work started from. **The loop** means the largest set of its classes that all reach each other. **The cache engine** means the part of the core that implements caches — the cache contexts, the adapters, exchange, transactions and persistence. **The kernal** means the rest of the node: discovery, communication, jobs, tasks, services. The spelling *kernal* is Ignite's own, as in `GridKernalContext`. To **price** a step means to compute, before it is made, how many references it would change and how many classes would leave the loop. CodeLaser is always called CodeLaser.

HOW THE PIECES WERE CONNECTED



The Ignite source. A clone of `apache/ignite` at master commit `a5b6b0dc44`, version 2.19.0-SNAPSHOT, on its own branch. The upstream repository was never written to.

The CodeLaser engine. A refactoring and modernization engine. It reads the whole project once and builds a model of it — every type, every member, every reference from one to another. It answers questions from that model, and it changes code through operations.

Claude Code. Drives the work. It puts questions to the model, receives measurements and ranked options back, chooses the next step from them, and issues the change as an operation. It records what it expected each step to produce, then what was measured.

The Maven build. Ignite's own build, unchanged, on JDK 17. After each step it compiles the whole reactor with Ignite's checkstyle, and runs the tests that exercise the change.

The core has 796 generated source files — message serializers, system-view walkers and data-transfer-object serializers, produced by Ignite's own annotation processors. A build treats them as compiled output. This work treats them as source, because a generated class can hold a dependency as real as any hand-written one. The loop measured with generated code as source is 2 476 classes; without, it is 2 266. Every figure in this document uses the first, except the few marked as measured without the generated code. Appendix B says why it matters.

THE TOOL

3. How the CodeLaser engine works

This section describes what CodeLaser adds to an editor and a compiler.

It holds a model of the code, not the text. The project is parsed once: 57 source sets and 10 962 types. What CodeLaser then holds is its structure — the types, what each one declares, and how the type hierarchies fit together. It also holds every reference from one place to another: calls, constructions, field accesses, annotations, imports. The dependency graph follows from that.

So do the questions worth asking of it. "How many places create this class by name?" comes back as a number. So does "which classes leave the loop if this reference goes", and "what would this boundary cost".

It prices a change before the change is made. Given a set of classes, CodeLaser reports which references keep them in the loop and how many classes would leave it if those references were cut. Given a proposed boundary, it reports how many references cross it and in which direction. Different sets can then be compared by their numbers.

It changes code as whole operations. Moving a set of static members to another class is one instruction. CodeLaser finds every place the change reaches and rewrites each one — imports and static imports included. One instruction in this work rewrote 420 call sites across 207 files.

It refuses what it cannot do correctly. An operation that would leave the project not compiling is declined before anything is written. After every write, the whole reactor is recompiled, and a failure stops the work there.

It rebuilds its model after every change. Every result in this document was taken from a fresh parse of the source after the change.

OPERATION	WHAT IT DOES	WHERE IT WAS USED HERE
<code>graph.giantComponent</code>	reports the largest group of classes that all reach each other	measured the loop at the start, and after every step
<code>graph.cycleImpactWithout</code>	reports which classes would leave the loop if a given type were removed from it	ranked every candidate before the first change
<code>query.explainEdge</code>	lists every reference from one class to another, with its kind and position	turned "these two registries hold 278 classes" into the nine references that had to change
<code>graph.splitReadiness</code>	reports, before anything is written, whether a set of classes can leave the artifact and what stands in the way	priced the management module and listed everything else that had to change
<code>extract.moveStaticMembers</code>	moves static members to another class and rewrites every use	the JDBC protocol version table, the REST defaults
<code>extract.extractCompanion</code>	moves a chosen set of a utility class's members to a new class	63 members of <code>IgniteUtils</code> into <code>IgniteKernalUtils</code> , 420 call sites

SIX OF THE OPERATIONS USED IN THIS WORK

The first four change nothing; they answer a question. Most of the calls in this work were of that kind: 199 calls in 59 scripts, and the great majority of them measurements.

Here is one of those calls, from step 0. It asks what would happen to the loop if the root of the management command tree were taken out.

```
graph.cycleImpactWithout(  
    types=["org.apache.ignite.internal.management.IgniteCommandRegistry"])
```

A QUESTION PUT TO CODELASER IN STEP 0

<code>typesFreed</code>	154
<code>largestCycleAfter</code>	2320

THE TWO FIELDS OF THE ANSWER THAT THE WORK USED

Without that one class, 154 classes would leave the loop entirely, and the largest loop left would be 2 320 classes. Appendix B comes back to the 154.

THE STARTING POSITION

4. What a build boundary requires, and why the core cannot simply be cut

Ignite builds with Maven and does not use the Java module system. Two of the module system's three rules therefore do not apply: a module declares no exports, and a package may be split across two artifacts on one class path. One rule does apply, and it decides everything that follows.

A module is built before whatever uses it. Module B can use module A if A is built first. Two modules cannot depend on each other, in either direction, at any remove. So a set of classes can become a module only if every reference between the set and the rest runs one way.

That is what makes the core hard to cut. 2 476 of its classes reach each other in a loop: A uses B, B uses C, and C uses A again. No order puts any one of them first, so no boundary can be drawn between any two of them. All of the loop is in `core/main`. Measured without the generated code, it holds 76% of all the classes in the repository that sit in any loop. The next largest loop, 289 classes, is in the Calcite module and does not touch the core.

Moving files does not help. A class moved from one side of a proposed boundary to the other takes its dependencies with it, so the loop is the same size afterwards. The loop shrinks only when a dependency is removed. Every step below removes dependencies; one of them then moves files.

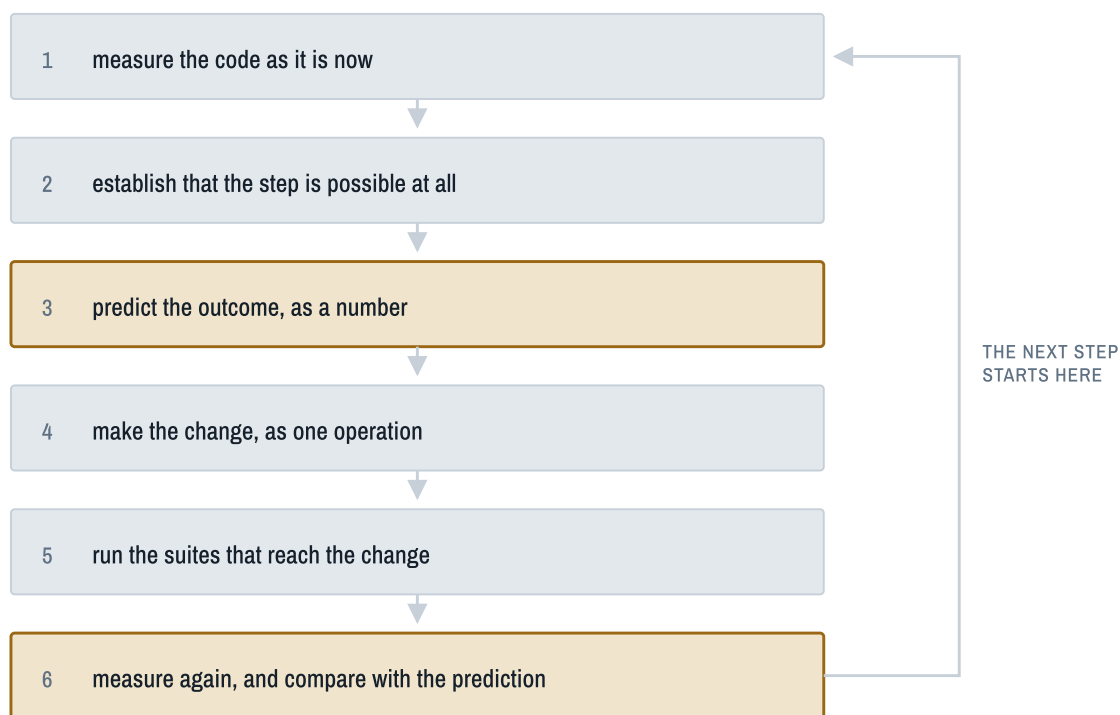
Ignite's own maintainers had already started. The modules extracted under IEP-119 — `commons`, `binary`, `thin-client`, `nio`, `unsafe` — are out of the loop, and the 2.17.0 release measured at 3 262 classes in it where master measures 2 476. Each extracted module still carries a small loop of its own, because the dependencies inside it moved with the code. This work continues in the same direction, and its two first steps are on IEP-119's path.

METHOD

5. The loop each step of the work went through

The work followed the same loop at every step: measure the code as it stands, get back a ranked set of possible moves, take one, and check the result against what was predicted.

THE LOOP EACH STEP WENT THROUGH



1. **Measure the source itself.** Every figure in this document was derived from the Ignite source, on the branch the work ran on, at the moment it was needed. Nothing was carried in from elsewhere.
2. **Establish that a step is possible before pricing it.** Before any cost is estimated, CodeLaser is asked what holds a set of classes in the loop and which references would have to change. A set that is held by an `extends`, or by a public method's return type, was not priced further.
3. **Predict the outcome as a number, then compare.** Each step was priced before it was made and scored afterwards. The prediction is the size of the loop after the step, and the classes that leave it.
4. **Make the change.** Through a CodeLaser operation where one exists for the shape, and otherwise as a scripted edit built on CodeLaser's measurements. Section 12 says which was which.
5. **Check with the tests that reach the change, and prove they can tell.** Every commit compiles the whole reactor with checkstyle. The suites that exercise the changed code are then run. For most steps they were run a second time with the change broken on purpose, a class name misspelled or a service entry deleted. A suite which stays green either way is then known not to count.
6. **Score the step by measuring again.** The loop is re-measured from a fresh parse, and its members compared with the prediction one by one.

Step 5 is where this work differs from a whole-suite standard. Ignite's suite is large, and the routine here ran the suites that reach each change rather than all of them. Section 10 gives the consequence.

WHAT WAS DONE

6. Stepwise overview of the work

	STEP	LOOP	WHAT IT PRODUCED
0	Measure the starting point	2 476	57 source sets, 10 962 types, 66 loops in total (measured without the generated code), and the two registries at the top of the board
1	Two registries found through <code>ServiceLoader</code>	2 188	Nine references in twelve files. Predicted 2 187; the one extra is the new factory interface
2	The generated serializer factory removed	1 842	Predicted exactly. A cache in front of a lookup that already worked, and a latent data race with it
3	<code>ignite-management</code> separated out	1 842	520 classes above the core. The loop unchanged by design: a module boundary changes no dependency
4	<code>IgniteUtils</code> split in two	1 505	63 statics to <code>IgniteKernalUtils</code> , 420 call sites. Predicted 1 503
5	The REST processor and the statistics manager created by name	1 446	Five references. Both predicted exactly
6	Five changes to what public classes reach for	1 295	Property defaults, the JDBC version table, the thin-client handler, <code>SecuritySubject</code> , the MBeans manager. All five predicted exactly
7	Fourteen small moves of the same kinds	1 136	Components created by name, constants to their owners, the client connector no longer naming its protocols
8	Seven references a reviewer would move anyway	1 061	Sentinels, defaults and one-line delegates to the class that owns them

	STEP	LOOP	WHAT IT PRODUCED
9	The kernal context split into a lower and an upper interface	558	27 commits, 985 crossing references to zero. The loop did not move until the last three went
10	The page format stops naming its own subclasses	483	33 references, predicted exactly

THE WORK, STEP BY STEP, WITH THE LOOP AFTER EACH

Steps 1 to 6 and 10 were each predicted to the class before they were built, and measured to the class afterwards. In two cases the measurement was higher than the prediction. In step 1 the one extra class is the new interface the step introduced. In step 4 the two extra classes are held by a helper that moved, and freeing them would change how peer class loaders are found.

STEP BY STEP

7. The eleven steps, in more detail

Step 0 — measure the starting point

Goal. To know what the work was starting from, before changing anything.

In numbers. The project parsed to 57 source sets and 10 962 types. The loop is 2 476 classes, all in `core/main`. Measured without the generated code, there are 66 loops in the repository, and this one holds 76% of every class that is in any of them. The next largest, 289 classes, is in the Calcite module and does not touch the core.

CodeLaser was then asked, for each class in the loop, how many classes would leave if that class were taken out. Two stood far above the rest: `IgniteCommandRegistry`, the root of the management command tree, and `ClientMessageParser`, the root of the thin-client request tree. Each is held in the loop by a handful of references, and each holds a tree of a hundred or more classes behind it.

Step 1 — the two registries found through `ServiceLoader`

Goal. The kernal constructed the command registry directly and asked its class in a comparison. The thin-client connection constructed its parser directly. Nine references in all, and cutting them was predicted to take 289 classes out of the loop.

What changed. `IgniteEx.commandsRegistry()` returns the `CommandsRegistry` interface. The kernal finds the root registry through `ServiceLoader`, which is how the registry already found plugin commands. `CommandRegistryImpl.register` asks an overridable `root()` instead of comparing classes. The connection context receives its parser from a factory it finds the same way. Twelve files, 115 lines added and 17 removed.

In numbers. Predicted 2 187, measured 2 188. The one extra class is `ClientMessageParserFactory`, the new interface, which shares a small loop with the connection context.

Step 2 — the generated serializer factory removed

Goal. `IgniteDataTransferObject` called a generated class, `IDT0SerializerFactory`, which registered every data-transfer object with its serializer — 169 registrations, 116 of them in the command tree. That one generated class held 296 classes in the loop.

What was found. The factory was a cache in front of a lookup that already worked. On a miss, its `serializer()` fell back to `U.loadSerializer`, which finds the serializer by naming convention. All 169 registrations follow the convention. The `control-utility` tests generate a second class with the same fully-qualified name, so every core object in those tests already went through the fallback.

What changed. `IgniteDataTransferObject` caches the convention lookup in a `ClassValue`. The annotation processor generates the serializers only; 250 lines of factory generation are gone. Eight files, 24 lines added, 256 removed.

In numbers. Predicted 1 842, measured 1 842.

The generated factory mutated a plain `HashMap` on a miss, from whatever thread asked first. `ClassValue` is thread-safe by contract. The race was found by reading the code and was never seen to fail. The change removes it.

Step 3 — `ignite-management` separated out

Goal. With the registry and the factory out of the way, the management command tree had no reference from the core into it. CodeLaser was asked whether it could leave the artifact.

What CodeLaser reported. The command tree could leave with no edit inside it. Nothing left in the core names it, not even the core's tests, which Maven compiles inside the same module. Every import the moving classes carry names a core type, which is allowed for a module built after the core.

What changed. 299 files moved by rename, package unchanged, so no import and no signature changed with them. Beyond the move, 23 lines were added and 9 removed in five production files. The kernel accepts an empty root registry when none is registered. `IgniteSecurityAdapter` trusts the new jar, resolved by class name, so that management tasks stay system tasks under security. Three javadoc links became `{@code}`. One new pom, six existing poms, and three assembly descriptors, so that a released node finds the jar.

In numbers. `core/main` went from 5 632 to 5 112 classes. `management/main` holds 520. The loop is independent of a module boundary and stayed at 1 842 with the same members — that was the check that nothing moved which should not have.

This is a product decision as much as a code change. After the separation, `ignite-core` without the new jar still starts a node, with an empty command registry. The alternative is to make the jar required, so that every user adds it as a dependency. Which of the two is right is a choice for the maintainers. This work built the first.

Step 4 — `IgniteUtils` split in two

Goal. `IgniteUtils` is the utility class everything reaches for, and 218 references from it reach up into the kernal and the cache engine. Those references held 326 classes in the loop. After the split, what stays in `IgniteUtils` names nothing in the loop, so it is at the bottom by construction.

What changed. 63 static members — 59 methods and 4 fields — moved to a new `IgniteKernalUtils`, with their text unchanged. 420 call sites in 207 files now name the new class. Fourteen system property constants moved down to `IgniteCommonsSystemProperties` in `commons`, the direction IEP-119 already takes, and stay readable under their old names through inheritance. `loadSerializer` moved into `IgniteDataTransferObject`, its only caller. The static initializer finds `DiscoveryCustomEvent` by name instead of naming it.

In numbers. Predicted 1 503, measured 1 505. The two classes over are one type whose only reference into the loop is a moved helper, and a second that follows it. Freeing them would change how peer class loaders are found, so they stay.

Step 5 — the REST processor and the statistics manager created by name

Goal. Two subsystems, 55 classes, held in the loop by five references.

What changed. `IgniteKernal.createComponent` finds the REST processor by class name, the way the same method already finds the platform processor. Three REST defaults that `IgniteSystemProperties` documented moved to the `IgniteRestProcessor` interface. `GridQueryProcessor` creates the statistics manager through `IgniteComponentType`, the way it already creates the indexing component.

In numbers. REST alone, predicted 1 462 and measured 1 462. With statistics, predicted 1 446 and measured 1 446.

Step 6 — five changes to what public classes reach for

Goal. The public API, the configuration classes and the kernal context should not depend on the implementation above them. Five references, or families of references, did.

1. `IgniteSystemProperties` documented 88 defaults by naming the implementation constant that held each. The values now live in a class that depends on nothing, and each owner keeps its constant as an alias, so no public constant disappears and no use site changes. Every compiled value was checked identical before and after.

2. The JDBC protocol version table moved from the server's connection handler to `JdbcProtocolContext`, the class that consults it. One operation, 12 statics, 33 call sites.
3. The thin-client connection receives its request handler from the factory introduced in step 1.
4. The default `sandboxPermissions()` of `SecuritySubject` makes its privileged call itself instead of through an internal utility. Twenty-four public SPI interfaces left the loop with that one change.
5. The MBeans manager is created by class name, behind an interface with the three methods the kernel calls.

In numbers. 1 421, 1 390, 1 353, 1 329, 1 295 — each predicted exactly, and the classes that left matched the prediction one by one.

Step 7 — fourteen small moves of the same kinds

Goal. From here the loop has no single class worth removing: the best one frees 36 classes. Progress comes from subsystems, each held by a few references of a familiar kind.

What changed. The schema manager and the cache plugin manager create their default components by name. The client connector stops knowing its three protocols. The handshake codes move to the connection contract. The system view asks a connection for its type. Each protocol registers its connection-context factory as a service. Thread-pool metric names move next to the pools. `MetricUtils` owns the metric-registry names. Two public API classes call `CommonUtils` directly. `ServiceInfo` takes a class resolver instead of the kernel context. The page store owns the page format. The indexing SPI returns a filter interface. Shared configuration defaults live on the defaults interface that already existed.

In numbers. 1 281, 1 217, 1 207, 1 190, 1 166, 1 136, each measured against a prediction. The prediction held to the class each time except once, where the loop measured one class more than predicted: the new interface `ClientListenerConnectionContextFactory` joined it.

Step 8 — seven references a reviewer would move anyway

Goal. The last cheap references before the large step.

What changed. `PageMemory` hands out a single page metric. The write-ahead-log archive storage receives the "unlimited" flag from the log manager, which already reads the configuration. Entry-version sentinels live with the entry that interprets them. The Java-object key serializer is held by its own class. Table-type names belong to the table information. Two discovery types stop reaching into the cache utilities. Discovery messages receive only the one manager operation they use.

In numbers. 1 111, then 1 061. The second figure was predicted at 1 070. The nine further classes left because changing the signature of an overridden method also removes the reference from each of its six overrides. Predicted again with the overrides included, the figure is 1 061.

Step 9 — the kernal context split

Goal. From 1 061, every road to a substantially smaller loop cut one boundary: between the kernal and the cache engine. The chain that ties them is the kernal context. `GridKernalContext` has an accessor for every component of the node. The accessors that return cache-engine types are called from everywhere. `ctx.-cache()` has 1 916 call sites, and 915 of them go straight on to `.context()`, which returns the engine's shared context.

The design. A lower interface, `KernalContext`, declares the 68 accessors whose types live below the cache engine. `GridKernalContext` extends it and keeps the engine-typed accessors, covariantly. Nothing above the kernal changes: tests, other modules and engine code keep `GridKernalContext` and its concrete return types, and no cast is introduced. The kernal-level code pays, in `core/main` only. Its context declarations become `KernalContext`. Where it used an engine-typed accessor, it now goes through a narrow view declaring exactly the members that code calls, and the concrete component implements the view.

What changed, in five kinds. The lower interface itself. Views by widening: an interface declares what a helper uses, its parameters widen to it, and every caller keeps compiling. Generic accessors, the idiom of `Ignite.plugin()`, so that assignments and arguments need no edit. Members moved to their own home — constants, helpers and holders that a lower type read off an upper class. Two seams: `IgnitionLifecycle` for the node lifecycle statics that lower components call, and a marker interface for management tasks.

In numbers. 27 commits. The references crossing from the lower half into the engine went 985, 928, 777, 560, 449, 391, 299, 266, 207, 179, 105, 68, 54, 50, 21, 3, 0. The loop did not move for seventeen of those commits — 1 061 to 1 058 — and fell to 558 on the commit that removed the last three. Removing most of the references did not shrink the loop. It shrank only when none were left.

What the boundary allows. This was measured on the whole of `core/main`, 5 151 classes, by giving every class to the side it transitively requires. `core/main` has no reference from the lower side into the cache engine or anything above it. Section 9 draws the build units that follow.

Step 10 — the page format stops naming its own subclasses

Goal. Inside the 558, CodeLaser was asked, subsystem by subsystem, what each would cost to unwire from the cache engine and what it would free. The page-format package — `PageIO`, its versions, the B+ tree page types — has 264 references pointing into it and 20 pointing out. Nineteen of the 33 references in those 20 are one `switch` in `PageIO` over its own concrete subclasses: the registry shape steps 1 and 2 had already removed twice.

In numbers. Predicted 483, measured 483.

THE TRADE-OFF

8. What changed in the public packages

No type in a public package changed package or name. Sixteen declarations in public packages changed their signature, and every one of them already named an internal type — a kernel context, an implementation class, an internal manager — in a public position. They fall into three groups.

DECLARATION	WHAT CHANGED	WHAT IT COSTS
<code>MessageMarshaller</code> <code>marshal</code> and <code>unmarshal</code> in <code>plugin.extensions</code> <code>.communication</code>	the context parameter widens from <code>GridKernalContext</code> to <code>KernalContext</code>	an implementation outside the repository that overrides them must change its signature, or it silently stops overriding. Ignite's own generated marshallers were regenerated
<code>IgniteSpiAdapter</code> <code>ignite</code> <code>TcpDiscoveryMulticastIpFinder</code> <code>setIgnite</code> in <code>spi</code>	the field and the parameter change type from <code>IgniteEx</code> to <code>IgniteContextView</code> , which is <code>Ignite</code> plus the lower context	an SPI outside the repository that reached the cache engine through this field no longer can. Ignite's own SPIs used only what the view declares. A subclass that overrides <code>setIgnite</code> must change its signature, or it silently stops overriding
<code>IndexingQueryCacheFilter</code> in <code>spi.indexing</code>	a class becomes the interface it was always used as; the affinity implementation moves to <code>IndexingQueryFilterImpl</code>	code that constructed the class directly must construct the implementation
eleven system view constructors and type arguments in <code>spi.systemview.view</code>	they name the public data- structure APIs or a narrow interface instead of the internal implementation	none for a reader of the views; only the core constructs them

CHANGED DECLARATIONS IN PUBLIC PACKAGES

Two further changes in public classes are not signature changes. The machine-derived defaults of `IgniteConfiguration` and the constants of `IgniteSystemProperties` moved to interfaces the two classes already implement, and stay readable under their old names. `Ignition` defines its exit codes from the new `IgnitionLifecycle`, and they remain compile-time constants.

⚠ **Two of these changes can fail without a compile error.** Widening a parameter is source-compatible for every caller and for every implementation inside the repository, which the compiler checks. An imple-

mentation outside the repository is the one case the compiler cannot see. A plugin that implements `MessageMarshaller` by hand would compile and would stop being called. So would a subclass outside the repository that overrides `setIgnite` on the multicast IP finder with its old parameter type.

THE RESULT

9. One separated module and five build units the code now allows

The separated module

`ignite-management` is a Maven module above the core: 520 classes, 299 files moved by rename, built and tested. Its runtime couplings each have a test that fails without the line that handles it. The security trust line is the clearest: without it, five of seven permission tests fail, because the commands' internal tasks arrive from an untrusted jar and are refused.

The thin-client server side could leave the same way: 130 classes, priced at three references. It stays in the core for now. Seventy-nine core test files open thin-client connections, and a core test cannot see a module built after the core. Separating it means reorganising those tests first, which is the maintainers' to schedule.

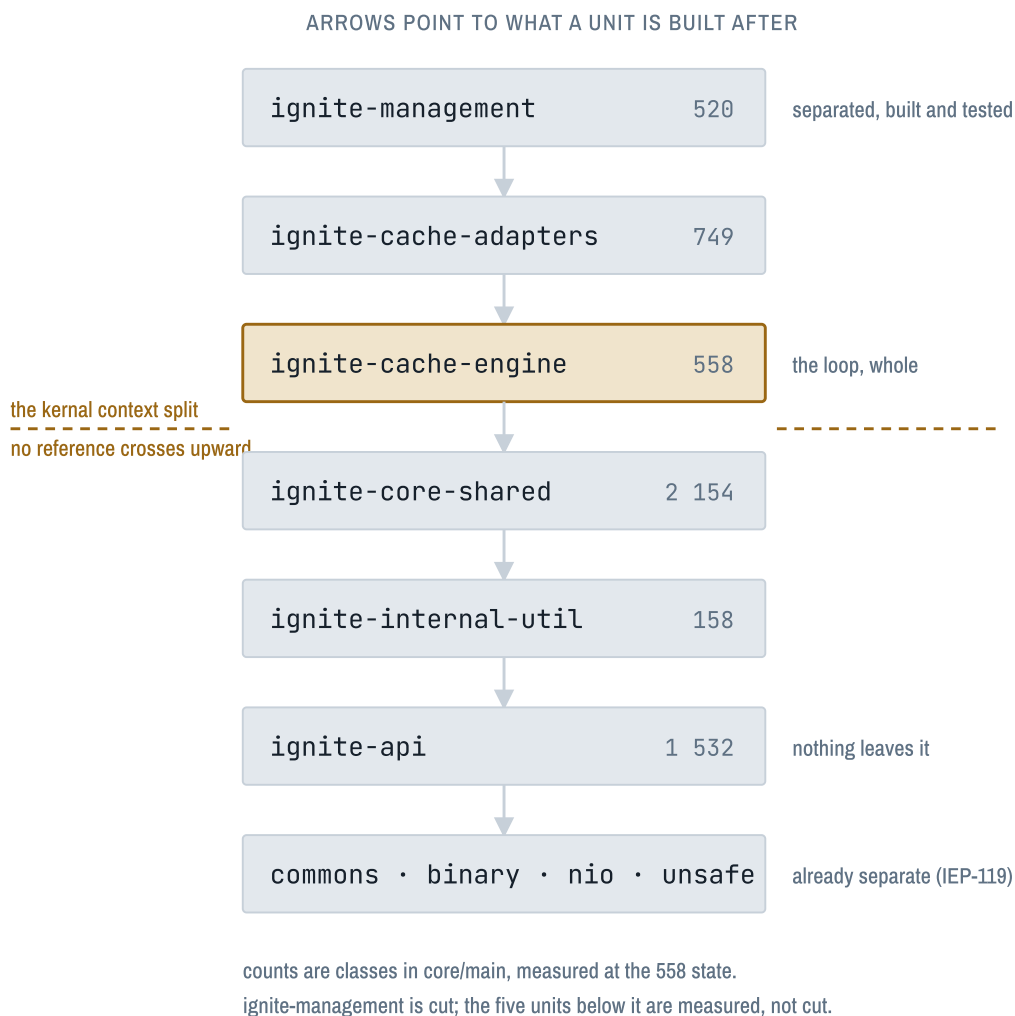
The build units the boundary allows

With the kernal context split, every reference in `core/main` between the following five sets runs downward, and none runs up. This is measured on all 5 151 classes of `core/main`, not on the loop alone. It is not yet cut: the poms, the test sources, the service files, the security trust list and the assembly descriptors are the same bill step 3 paid for one module, five times over. The names in the table are labels used in this document. None of these modules exists.

UNIT	CLASSES	WHAT IT HOLDS
<code>ignite-api</code>	1 532	the public API and the internal helpers it reaches. Nothing leaves it: 1 095 references come in, none go out
<code>ignite-internal-util</code>	158	<code>internal.util</code> and its closure
<code>ignite-core-shared</code>	2 154	messages, data-transfer objects, records, protocol and view types
<code>ignite-cache-engine</code>	558	the whole loop as it stood when the split closed
<code>ignite-cache-adapters</code>	749	thin-client and REST cache requests, atomic update futures, tree pages

THE FIVE BUILD UNITS THAT FOLLOW FROM THE SPLIT, MEASURED ON CORE/MAIN AT THE 558 STATE, BEFORE STEP 10

THE KERNAL CONTEXT SPLIT, AND THE FIVE BUILD UNITS THE BOUNDARY ALLOWS



`ignite-api` is the artifact a user would compile against. It is 1 532 classes, and 608 of them are the non-internal types. The other 924 are the internal helpers those types reach. That is a property of Ignite's public API as it stands today.

What the next steps would cost

Inside the 558, CodeLaser was asked what each subsystem would cost to unwire from the cache engine. Fourteen moves take the loop from 558 to about 260 by changing 840 references. Each of them makes the loop smaller by itself; the split made it smaller only once it was complete. The first, the page format, is step 10. The next are the query and index subsystems, then persistence in five parts, then data structures, the cache tree and atomic updates.

What would remain at 260 is the cache engine proper: the distributed cache's entry, context, topology, exchange and transaction managers. Those refer to each other because together they make up one distributed cache. About 260 is a plausible floor for this design.

ASSURANCE

10. How each step was checked

Three checks were applied at every commit, and one comparison at the largest step.

The whole reactor compiles, with checkstyle. Every commit builds all 43 modules with Ignite's checkstyle profile at zero violations. A change that does not compile never landed.

The suites that reach the change pass. For each step, the suites that exercise the changed code were run. That means a node started with the changed component, a command sent over the wire, a thin-client handshake, a page written and read back. Section 7 names the behaviour each step touches. The record names each suite.

The suite can tell. For most steps the same suites were run a second time with the change broken on purpose: the class name misspelled, the service entry deleted, the wrong page format answered. A suite that stayed green with the change broken did not count, and another suite was used in its place.

The largest step was compared against its own starting point. For the kernal context split, 65 suites across six modules — 882 tests — were run on the finished split and on the commit before it, built the same way. Five tests fail on both. They are a system view that lists a directory in an unsorted order, a discovery test bound to the machine's network interface, an upstream test that fails on purpose pending IGNITE-4706, and two page-compression tests that need Linux. Nothing fails on one side and not the other.

△ **The whole test suite was not run.** Ignite's core test sources hold about 3 400 test classes. The routine here ran the suites that reach each change. That is a weaker standard than running the whole suite. Every changed behaviour has a test that passes and that was shown to fail when the behaviour is broken. No claim is made about behaviour no step touched.

ASIDE

11. Three things in Ignite's code found along the way

None of them affects a result above. Each is worth knowing.

`EventType` and `IgniteUtils` initialise each other through `Unsafe`. `EventType.EVTS_ALL` is computed by `IgniteUtils`, whose static block reflects over `EventType` and, when the field is still null, writes it through `putObjectVolatile`. The two classes know about each other on purpose, and `Unsafe` repairs the initialisation order. A cleaner design has `EventType` compute its own list.

Components found by name are found by constructor signature. `IgniteKernal`, `IgniteComponentType` and `CachePluginManager` look a component up with `getConstructor(GridKernalContext.class)`. A component whose constructor now takes the lower `KernalContext` keeps a constructor the lookup finds. Nothing at compile time checks this; only a node start does. It is written down because the next person to change a component's constructor will not be told by the compiler.

The test harness shares one JVM across modules. The parent pom sets `forkCount` to 0, so the tests of every module in one Maven invocation share a JVM. Ignite’s serial filter is JVM-global, so the second module’s grid tests fail at class initialisation. Run one module per invocation and everything is green. One test, `GridManagerMxBeanIllegalArgumentHandleTest`, installs a failing mock in a static field that every later suite in the same JVM then inherits.

EFFORT

12. Effort required to compute and perform the work

Included because it shows what repeating the exercise would involve.

	VALUE
commits on the branch	65
files moved by rename	299
files changed otherwise	794, 4 445 lines added and 6 815 removed
new files	51
CodeLaser calls in the scripts	199, in 59 scripts
predictions scored in the first thirteen sessions	217 — 168 held, 49 refuted

THE WORK

The work removed more lines than it added. The two largest deletions are the generated factory in step 2 and the 68 declarations that left `GridKernalContext` in step 9.

How the changes were made. Steps 1 to 6 were made through CodeLaser operations or through scripted edits built on CodeLaser’s measurements, with every write recompiled by CodeLaser before the next. Steps 7 to 10, including the kernal context split, were made through CodeLaser’s operations.

The predictions. In the first thirteen sessions every claim was written down before the run that would test it: 217 predictions, 168 held. Almost all of the refuted ones estimated the cost of a step: how many call sites or files it would touch. The size of the loop after every built step was predicted to within two classes. From the fourteenth session on, each step carried its prediction of the loop and its measured result without the formal tally.

△ **Time is not reported.** Most of the elapsed time was the build running, which depends on the machine it runs on.

APPENDIX

A. What was not done, and why

- **The thin-client server side stays in the core.** Section 9 gives the reason.
- **The first target, a loop of 200 to 400, was set aside.** When the loop stood at 1 190 classes, every arrangement the search found below about 500 had to cut one of three things. The first is the calls of `Ignition` into `IgnitionEx`, which is the facade's whole purpose. The second is a built-in failure handler's call to `IgnitionEx.stop`, which its javadoc names as its contract. The third is a configuration bean's field holding its own sub-configuration. None of those is a change a reviewer should accept, so the target became about 555, the smallest loop reachable without them. The kernel context split reached 558, and step 10 then reached 483 without touching any of the three.
- **Reflection was not introduced for the graph's sake.** Several one-reference changes were priced and refused: making `PdsConsistentIdProcessor` or a default failure resolver reflective would have freed a few classes each and served no design purpose. The by-name constructions that were made each follow a precedent in Ignite's own code — `IgniteComponentType`, the platform processor, the thin-client service factory.
- **Two changes were made that a maintainer may want to revisit.** The statistics manager created by name in step 5 treats query statistics as a pluggable component. It is a separate commit so it can be dropped, and the REST change stands without it. The Java-object key serializer in step 8 moved a mutable global from one class to another rather than redesigning it. The commit says so.

APPENDIX

B. How to read the figures

Two loops were measured at the start, and this document uses the larger. With the 796 generated sources treated as a compiled library, the loop is 2 266 classes. Treated as source, it is 2 476. The first is what a build sees; the second is what the dependencies are. The difference matters because one generated class held 296 classes in the loop, and a model that cannot see it prices the separation of the command tree alone at 278 classes for its six references, where the true figure is 154. Every figure here is from the second model, except the few marked as measured without the generated code.

A prediction is the size of the loop after a step, and which classes leave it. Where a measurement differed from its prediction the difference is stated in the step: one class more in steps 1 and 7, each a new interface the step introduced; two more in step 4; nine fewer in step 8, from overrides the prediction had not been given.

The build units in section 9 have not been cut. They follow from giving every class in `core/main` to the side it transitively requires, and from checking that no reference then runs upward. The test sources, which Maven compiles inside the core, would have to be divided the same way, and the bill step 3 paid for one module is owed for each of them.

The road onward in section 9 is an estimate. Each of the fourteen moves was priced from the graph at the 558 state. The first has been built and measured exactly. The rest have not been built. The graph gives the number of references each would change. It does not say whether the change is a good design.

What the figures do not cover. The whole test suite was not run; section 10 gives the standard that was applied instead. Runtime behaviour that no test reaches is not claimed. Where this document says a change "stays readable under its old name", that was checked by compiling and, for the compiled constants of step 6, by comparing every value before and after.