Optimizing Builds

You should already know how to:

- Create a Google3 client using CitC, Piper, or Fig.
- Use Blaze to build and test targets with Google3.
- Edit BUILD targets and rules.

Need a review? Do build codelabs (go/build-codelab).

75K changelists a day

2.3M packages

15M builds/tests a day

4.2B lines of code

Three principles of build optimization

Build small

Design for reuse

Use automation


You work faster
Build small
Your users save time

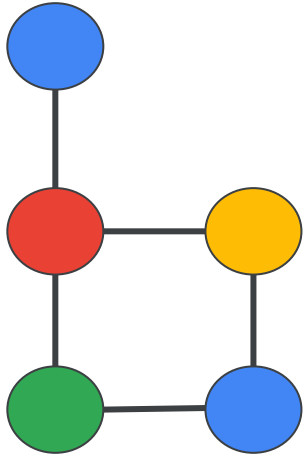
Design for efficiency

Use less resources

Do less work

Use automation

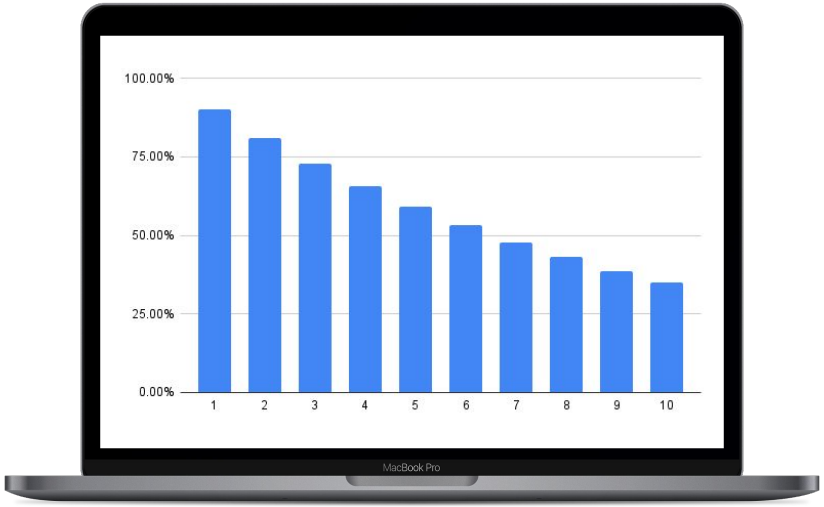
Optimizing builds will enable your team and your users to do great work efficiently.



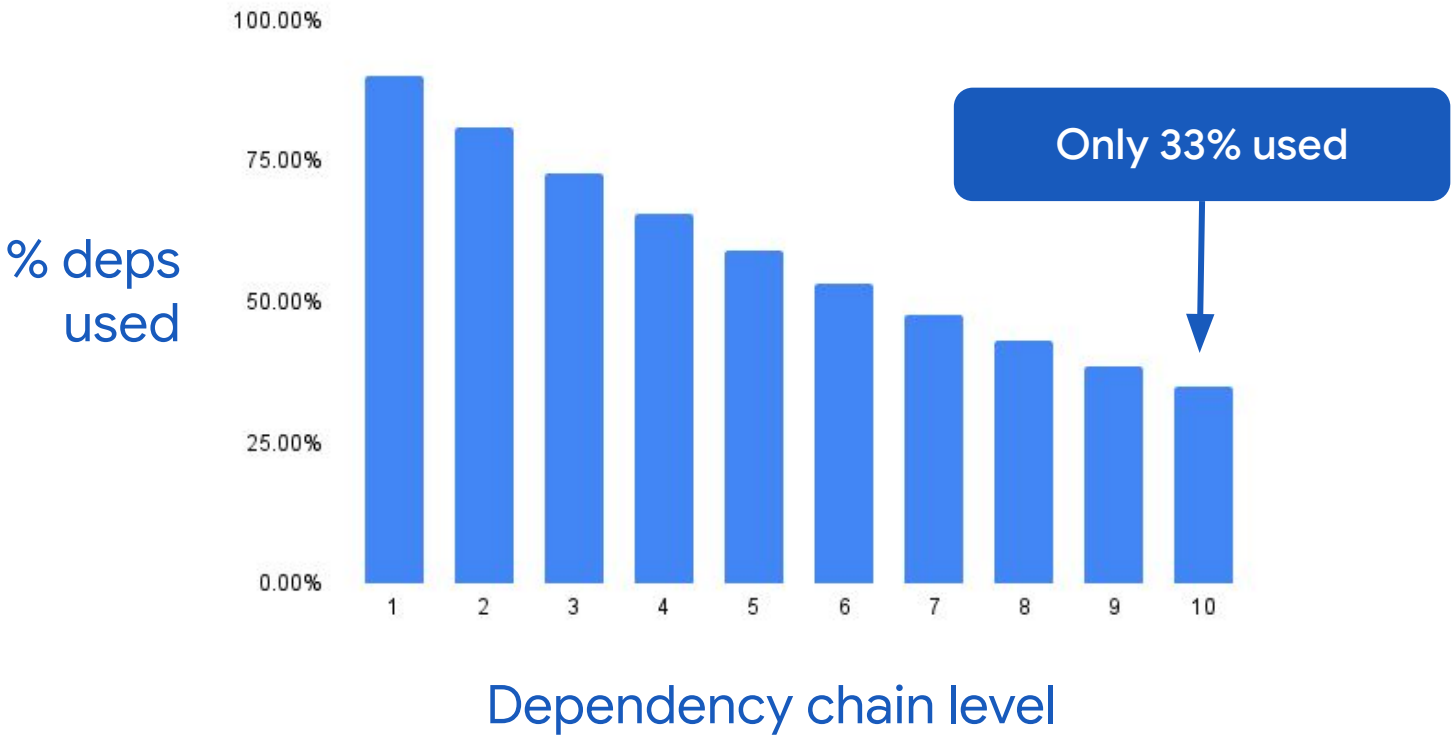
Module 1

Dependencies: Who depends on yours?

Dependency bloat is the biggest build performance issue in Google3.



Long dependency chains are wasteful



2018

Only 10% of dependencies used

Why does it matter?

If dependencies'	You and your users must
code changes	analyze and rebuild
code breaks	fix broken code
API changes	migrate
dependencies grow	see your deps grow

Even if you don't use them!

Common sources: infrastructure libraries

```
import com.google.bigtable
```

source file imports

```
java_library(
  name = "spanner",
  srcs = glob(["*.java"]),
  deps = [
    "//…jsr330_inject",
    ":api",
    ":errors",
    "//java/com/…common/collect",
    "//third_party/java/flogger",
  ],
)
```

Who do you depend on?

Who depends on you?

Identify your dependencies

```
blaze query 'deps(//java/com/google/spanner:api)'
```

Returns everything from spanner:api

go/blaze-query

Narrow your results

```
blaze query 'deps(//java/com/google/spanner:api)'
```

Returns just rules, no host config or implicit deps

```
blaze query --nohost_deps --noimplicit_deps 'kind("rule",
deps(//java/com/google/spanner:api))'
```

go/blaze-query

Find dependencies between targets

'somepath( )' finds deps for one path.

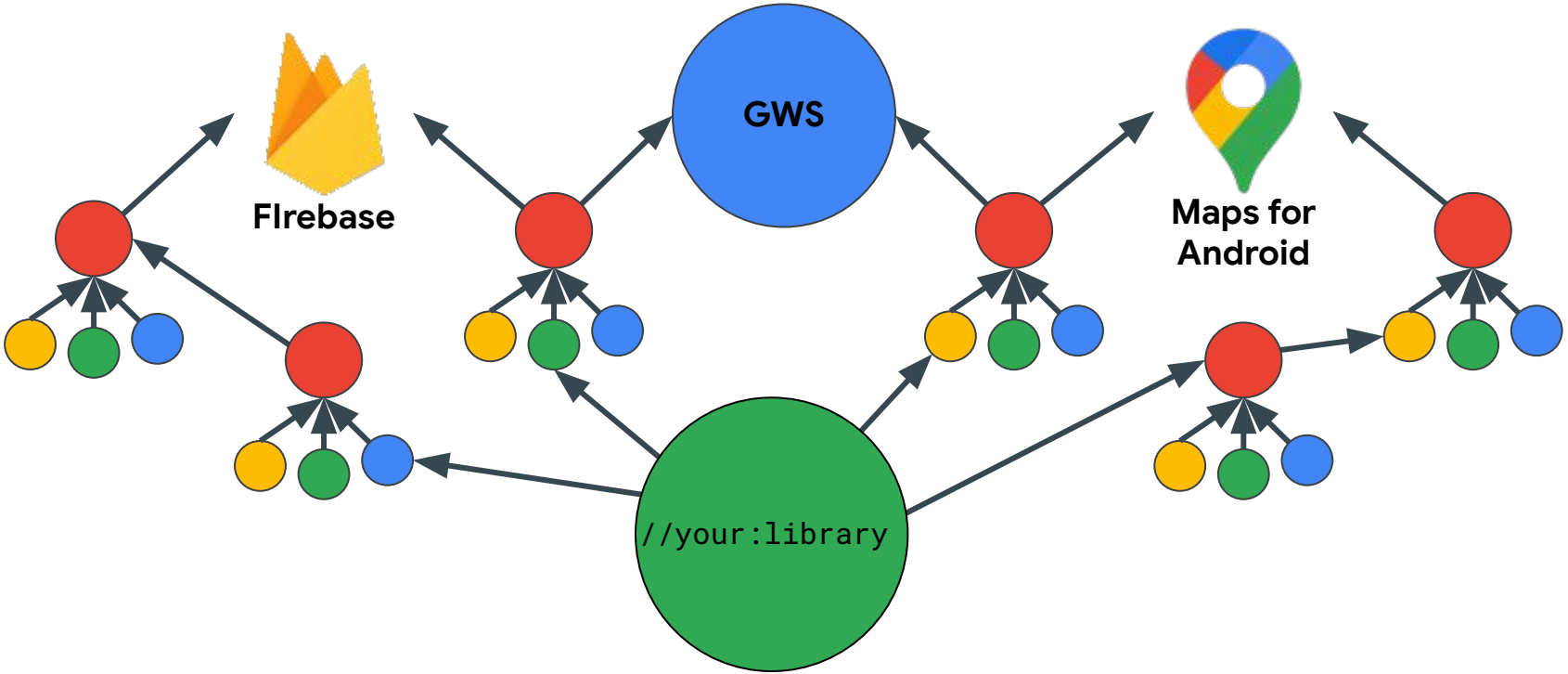
```
$ blaze query "somepath(//foo:foo, //third_party/zlib:zlibonly)"
```

```
$ blaze query "allpaths(//foo:foo, //third_party/zlib:zlibonly)"
```

'allpaths( )' find deps for all paths.

go/blaze-query

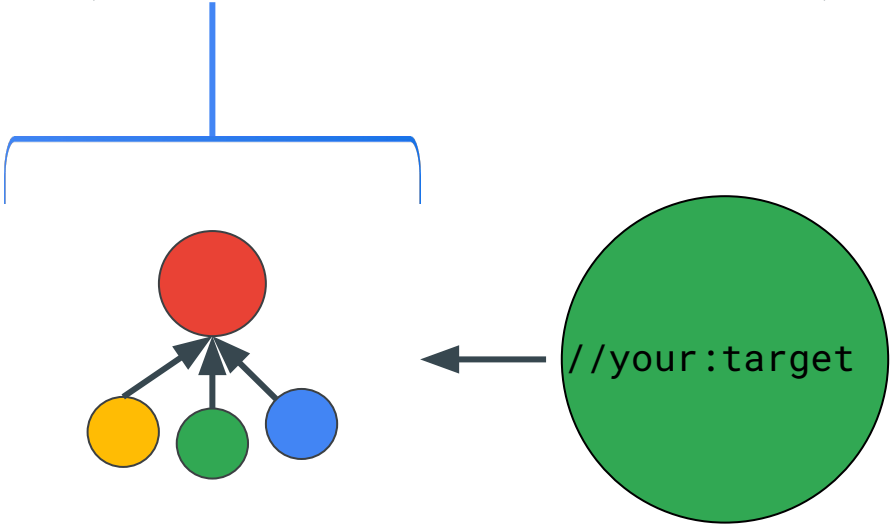
Work on a library? Who depends on you?



blaze query rdeps( )

Level of the reverse dep chain.

```
blaze query 'rdeps(<universe>, <your target>, depth)'
```



rdeps() example

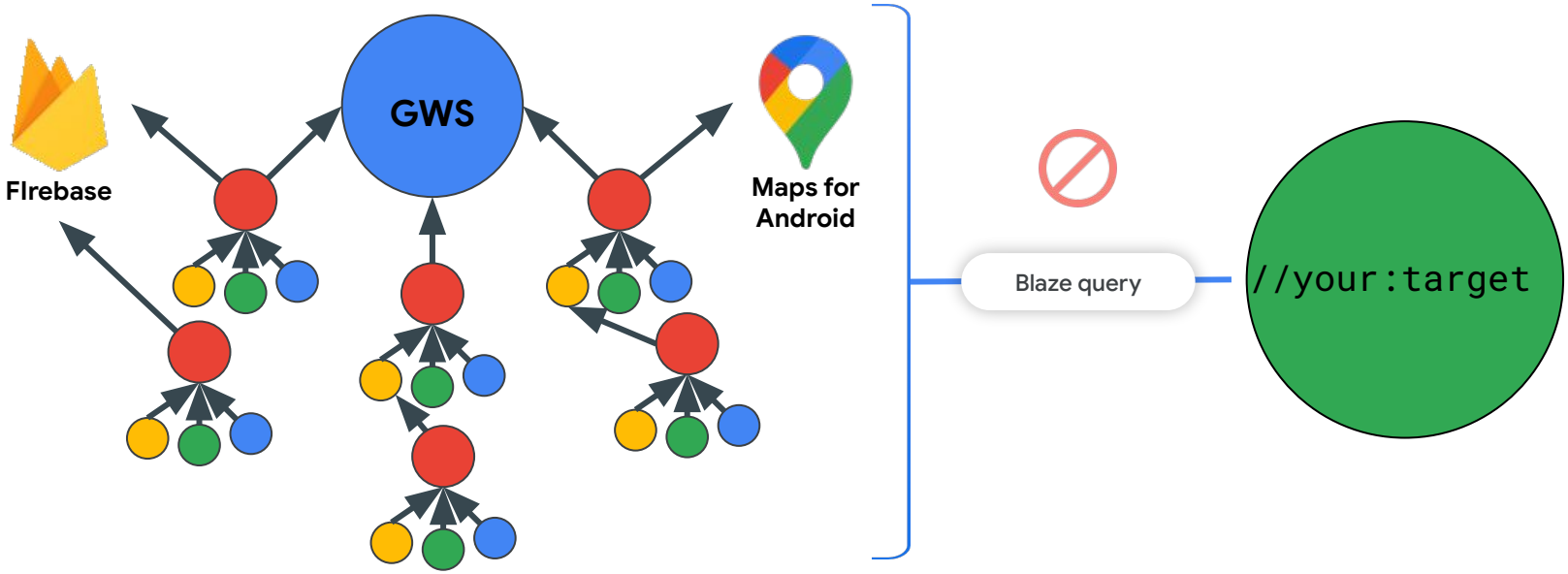
Evaluate all the deps in this target...

...that depend on this target.

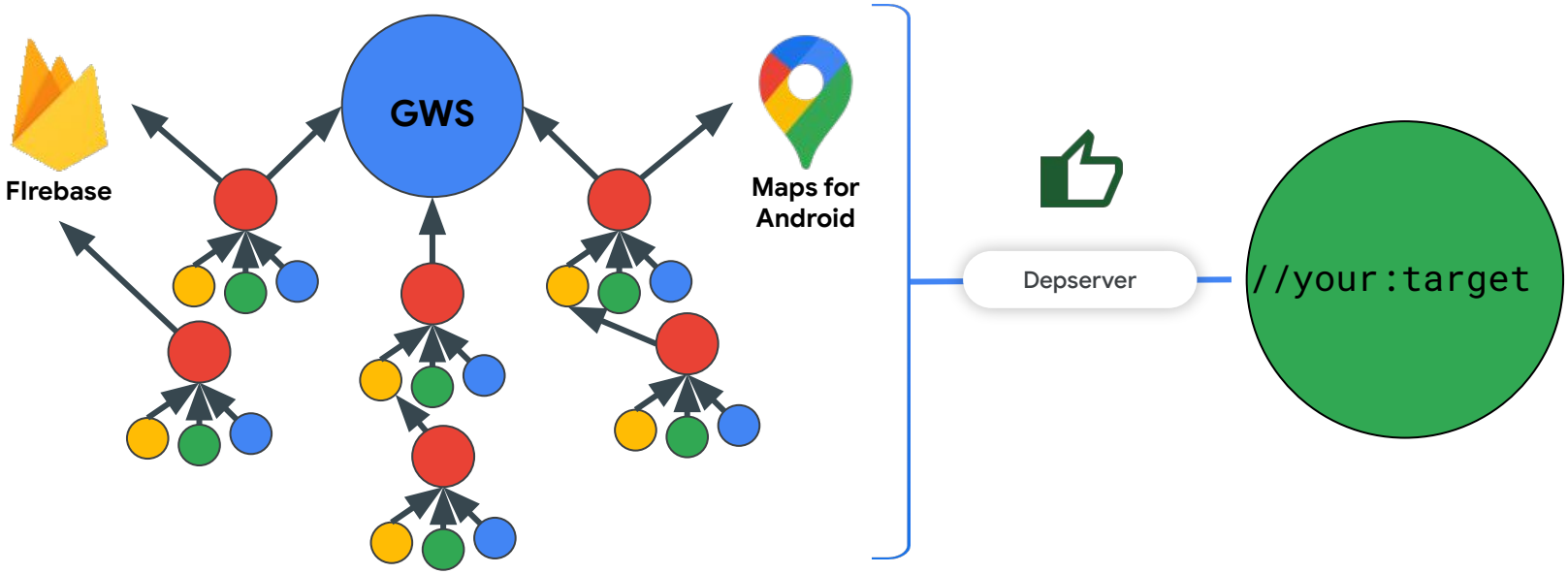
```
blaze query --nohost_deps
  'rdeps(//google/android/apps/play/movies, //wireless/android/tv/common, 3)'
```

go/blaze-query

Blaze query can't do it all



Depserver can query all google3



Depserver for evaluating it all

Prints all targets that depend on your library

```
blaze run -c opt //devtools/deps/depserver/query:depends_on.sh
    //your:library
```

go/depserver-overview

Diagnose and fix a bloated dependency

Target

```
//java/com/google/spanner:api
```

Library for generating and parsing HTML

depends on its dependency

```
//java/com/google/common/html:html
```

Why do we depend on that?

```
blaze query --noimplicit_deps --nohost_deps
    'somepath(//java/com/google/spanner:api,
     //java/com/google/common/html:html)'
```

Finds one route between targets...

```
//java/com/google/spanner:api
//java/com/google/net/rpc3:rpc3
//java/com/google/net/rpc/contrib/rpcinjectz2:client
//java/com/google/net/rpc3:rpc3_noloas_internal
//java/com/google/common/html:html
```

...and returns this dependency chain

Why do we depend on that?

```
blaze query --noimplicit_deps --nohost_deps
    'somepath(//java/com/google/spanner:api,
     //java/com/google/common/html:html)'
```

```
//java/com/google/spanner:api
//java/com/google/net/rpc3:rpc3
//java/com/google/net/rpc/contrib/rpcinjectz2:client
//java/com/google/net/rpc3:rpc3_noloas_internal
//java/com/google/common/html:html
```

Dependency comes from rpc3

Take a look at the rule

```
java_library(
    name = "rpc3_noloas_internal",
    srcs = glob(
        [
            "*.java",
            "client/*.java",
            "client/loadbalancer/*.java",
            "client/util/*.java",
            "server/*.java",
            "stream/*.java",
            "impl/*.java",
            "impl/client/*.java",
            "impl/server/*.java",
            "impl/server/plugin/*.java",
            … more…
        ],
```

Globbing a lot of sources

"

Find what deps are being used

```
rpc3$ grep -r com.google.common.html *.java

BUILD:        "//java/com/google/common/html",
examples/HttpServerSupport.java:import static
com.google.common.html.HtmlEscapers.htmlEscaper;



"
```

Just this?

Narrow the dependency

```
# Most users should depend on :html.
java_library(name="html", ...)

# Targets for HtmlEscapers only, to avoid i18n identifiers and ICU4J.
java_library(name = "htmlescapers, ...)
```

Use `html:htmlescapers` instead of `html:html`

`java/com/google/common/html/BUILD`

Learn more about build health

Things you can do:

- Use blaze query to identify dependencies.
- Identify targets that depend on your target with rdeps().
- Identify reverse dependency depot-wide.
- **Fix a bloated dependency by finding underused deps.**