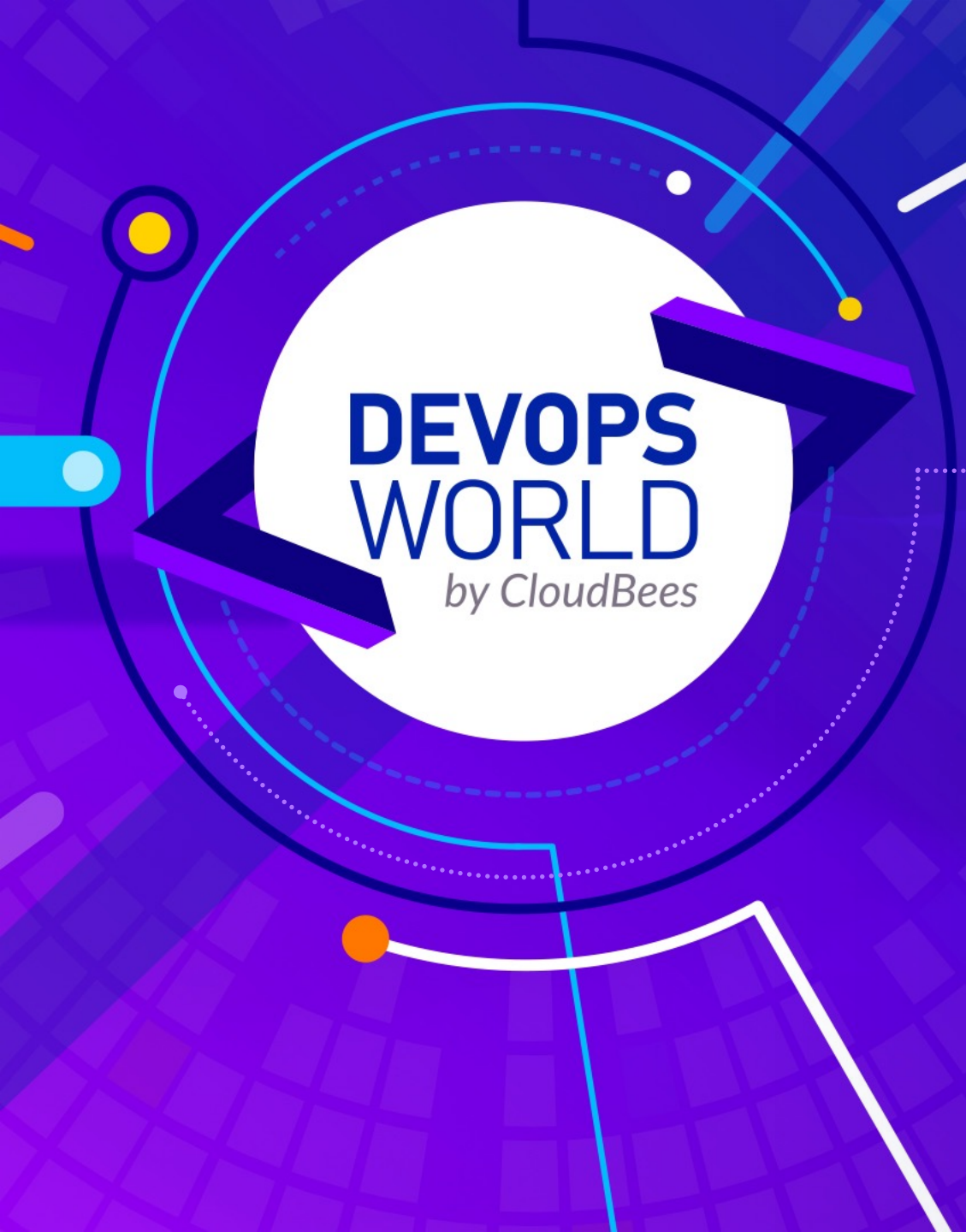


PRACTITIONER

Speeding up Maven and Jenkins with a Build Cache

JUSTIN REOCK

FIELD CTO AND CHIEF EVANGELIST

GRADLE, INC

Your Speaker

Justin Reock
Chief Evangelist and Field CTO
Gradle Enterprise

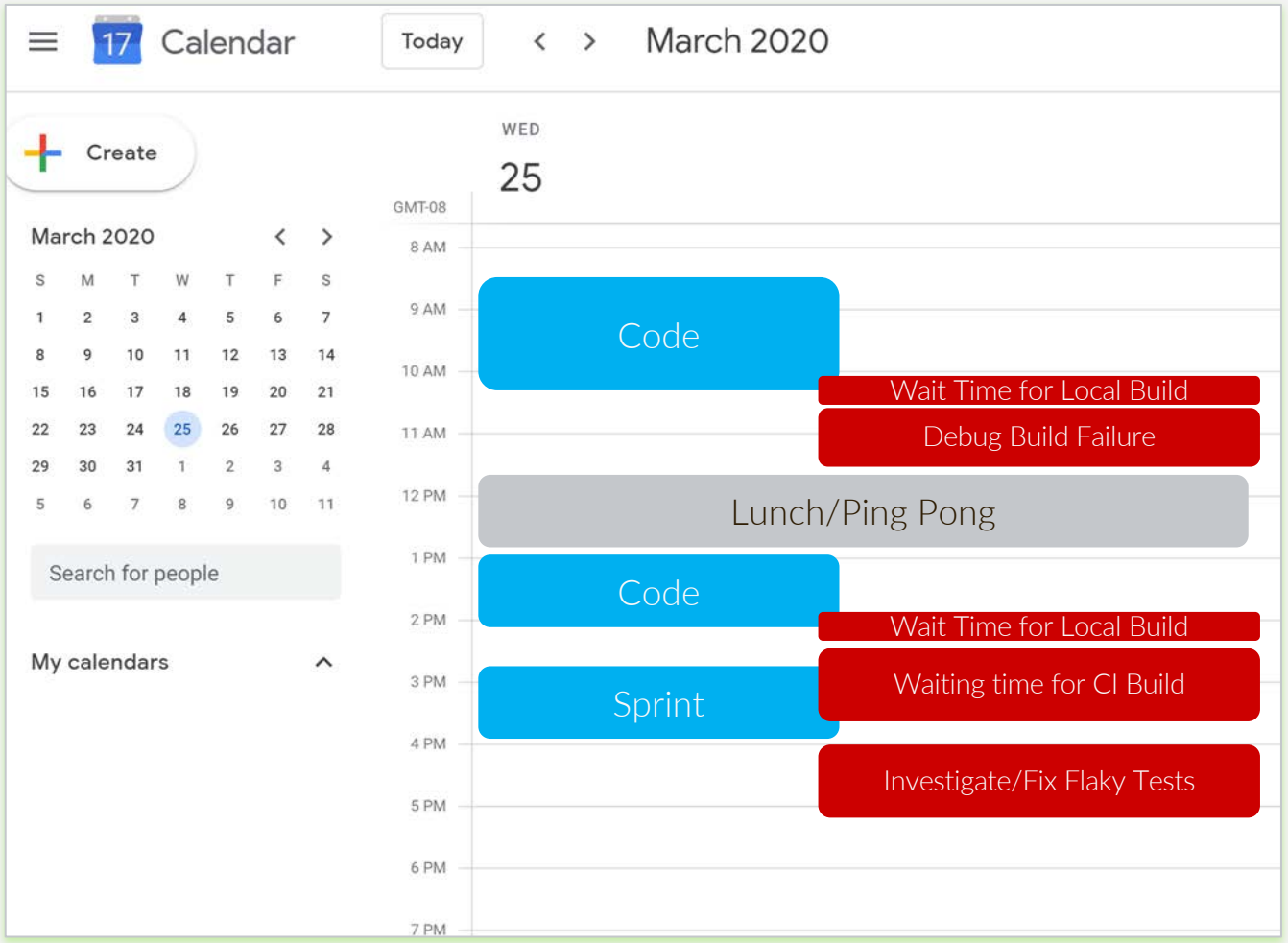
Justin has over 20 years of experience working in various software roles. He is an outspoken free software evangelist, delivering enterprise solutions, technical leadership, various publications and community education on databases, architectures, and integration projects.

@justinreock

SCAN ME

> “It’s no longer the big beating the small,
> but the fast beating the slow.”
> - Eric Pearson, CIO, International Hotels Group

Familiar?



In Search of Lost Time:

Developer Productivity in the Cloud Native Era

Respondents spend, on average, more than 15 hours every week on tasks outside of writing application code.

From maintaining internal tooling, to setting up dev environments, to debugging pipelines, to waiting for builds and test results. And **54%** of respondents identify **slow feedback loops during the development** process as a major (top 3) frustration.

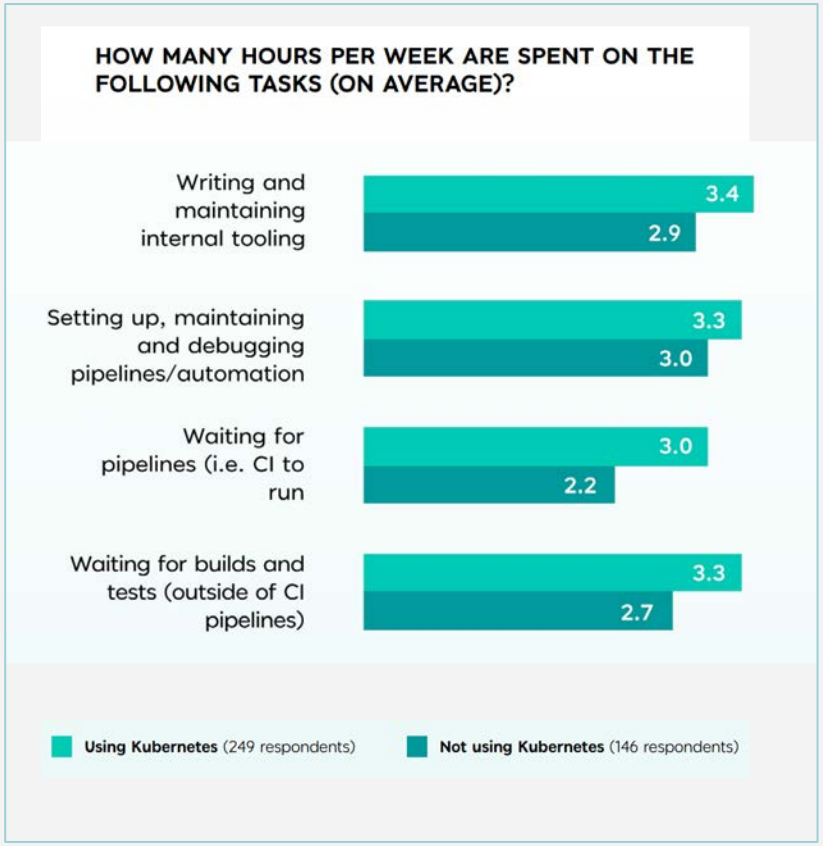
In the US alone, this time spent could be costing companies up to $61 billion per year*.

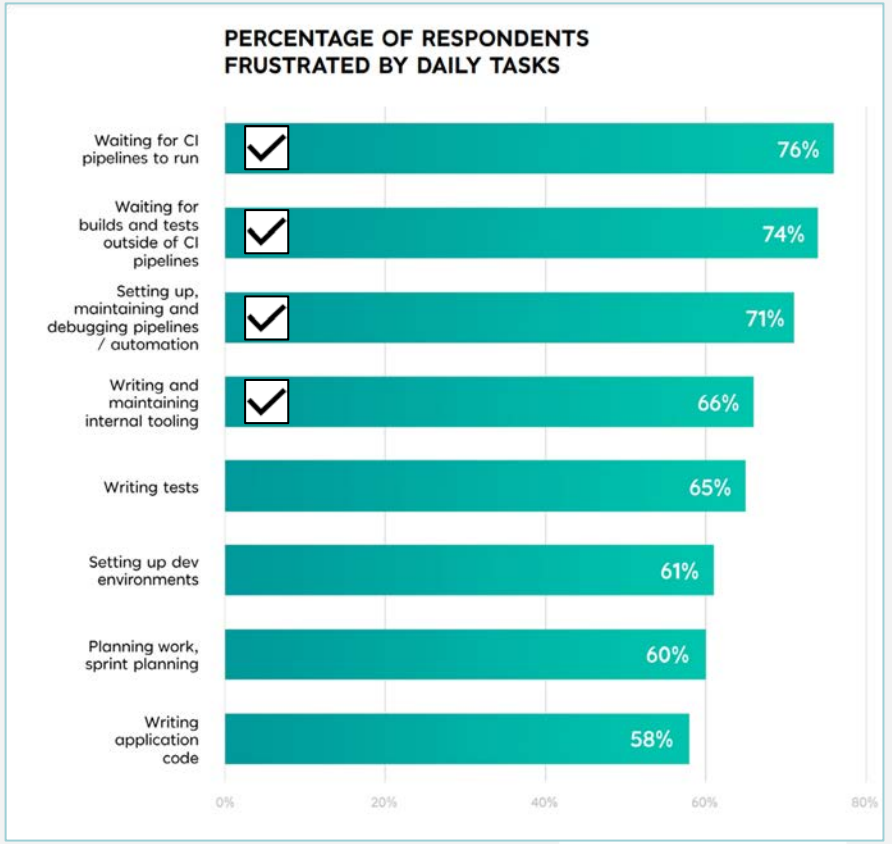
Seven in 10 respondents say the time they spend on specific tasks is time wasted and could be put to more strategic use.

49% of these respondents say they would **develop new products and services** to support the company.

46% say they would **improve speed and delivery** of existing products and services.

44% say they would **improve security for existing products** and services.





DEVOPSWORLD by CloudBees

garden SURVEY REPORT 2021

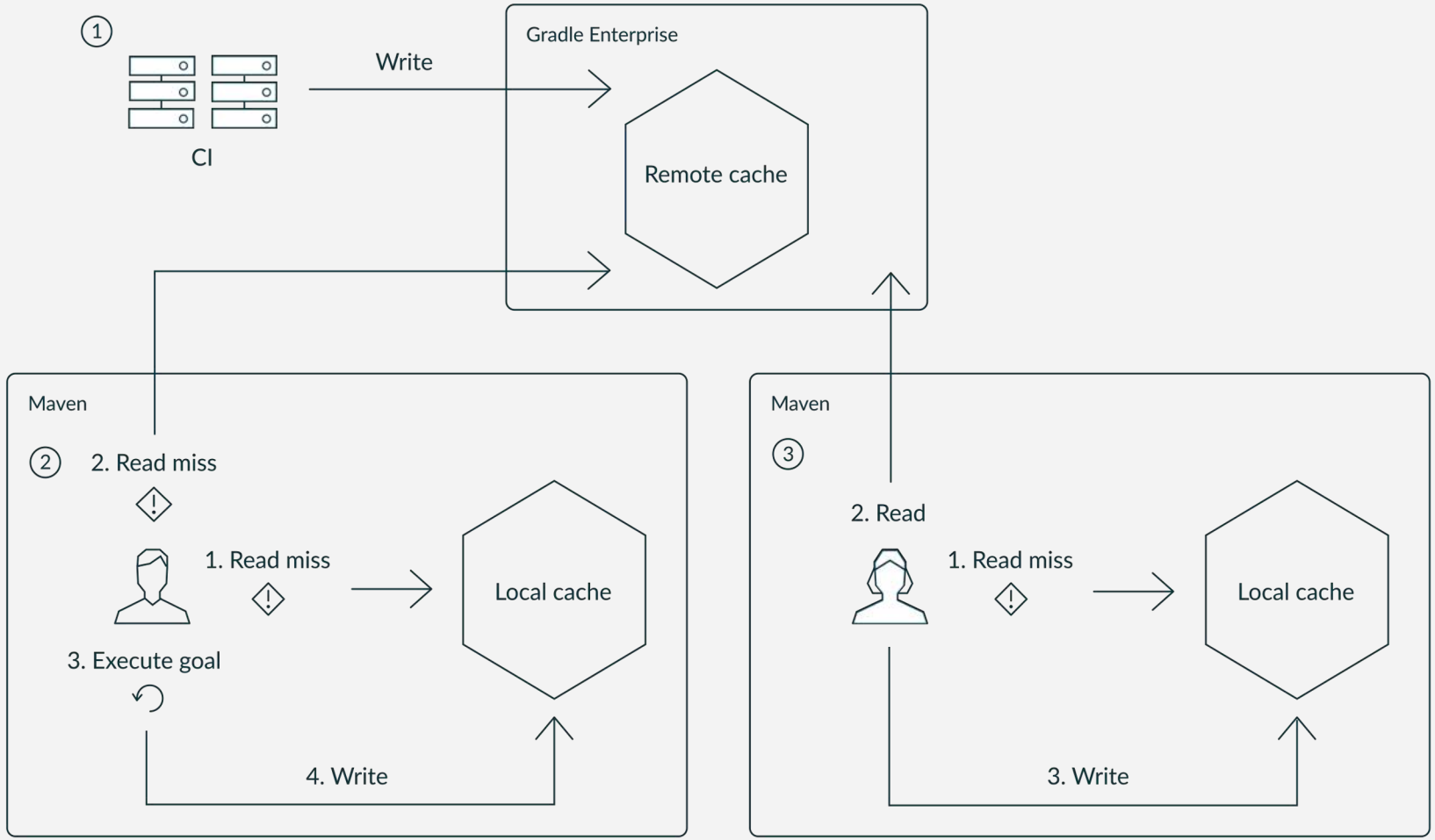
Build Caching

- Introduced to the Java world by Gradle in 2017.
- Available for Maven and Gradle, can support both user local and remote caching for distributed teams
- Build caches are complementary to dependency caches, not mutually exclusive:
  - A dependency cache caches fully compiled dependencies
  - A build cache accelerates building a single source repository
  - A build cache caches build actions (e.g. Gradle tasks or Maven goals)

What is a Build Cache?

When the inputs have not changed, the output can be reused from a previous run.

Local & Remote Caching



Examples of cacheable tasks/goal executions

Gradle Compile/Maven Compile	Gradle Test/Maven Surefire
• Source Files • Compile Classpath • Java version • Java compiler configuration • …	• Test Source Files • Test classpath • Test JVM configuration • …

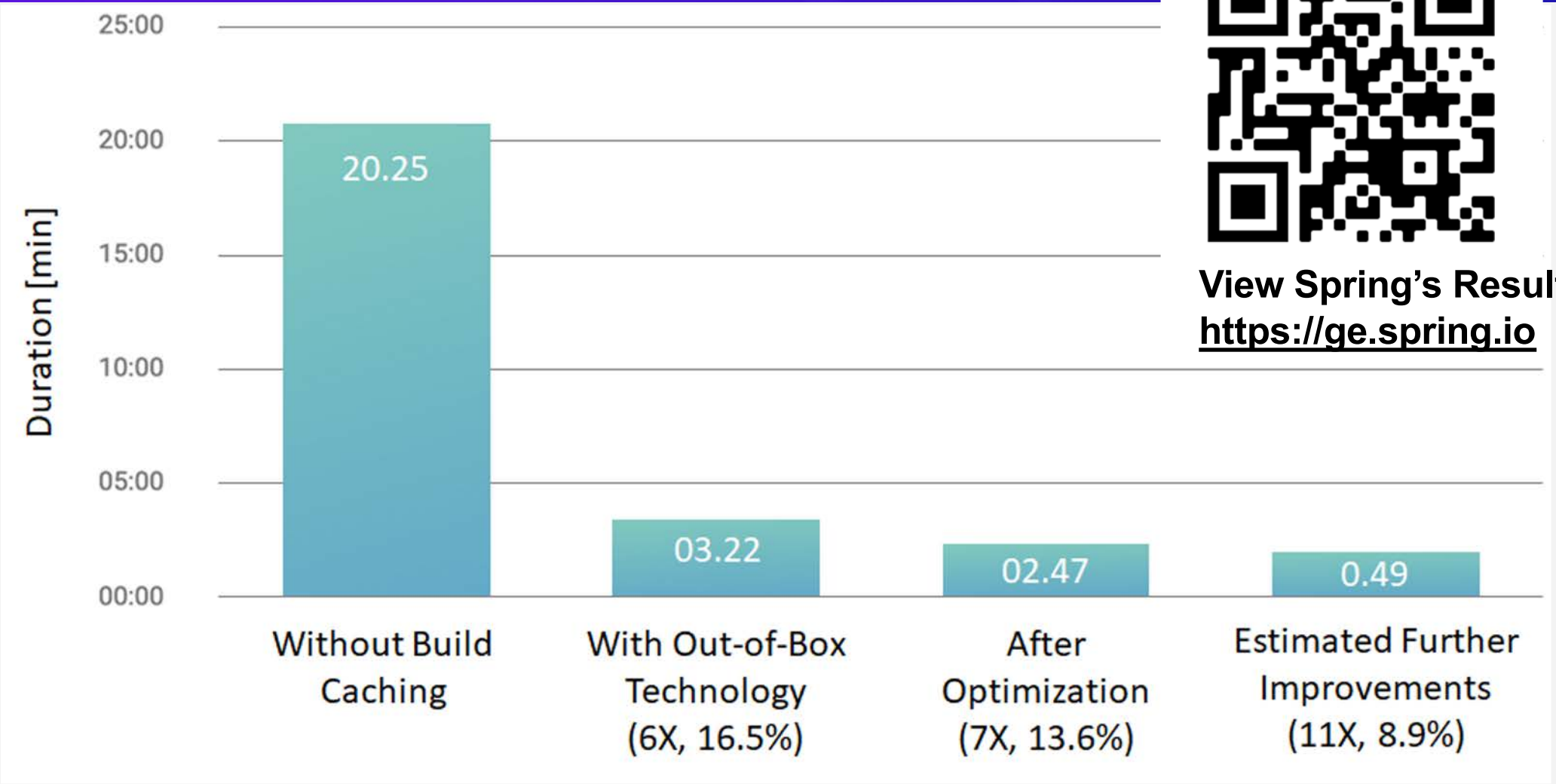
- Caching is a generic feature and applies to all task/goals
  - Compile, Test, Source code generation, Javadoc, Checkstyle, Pmd, Custom tasks/plugins
- For some tasks/goals caching has no benefits, e.g. clean, copy

Measurements with OSS Projects

When running `mvn clean verify`

Project	**uncached**	**fully cached**
Apache Commons IO	01:23 min	00:04 min
Google Guava	07:19 min	01:00 min
Spring Boot	21:31 min	06:53 min
Apache TomEE	1h 27min	20 min

Spring Boot Build Time for Compile & Unit Tests



View Spring's Results:
https://ge.spring.io

Setting Up a Local Cache

- For our sample project, we'll use the Camel Spring Boot Maven archetype from here:

  https://camel.apache.org/manual/latest/camel-maven-archetypes.html

```
mvn archetype:generate
   -DarchetypeGroupId=org.apache.camel.archetypes
   -DarchetypeArtifactId=camel-archetype-spring-boot
```

- Setting up caching is simple; we just need to enable the Gradle Enterprise plugin for Maven by creating or modifying **.mvn/gradle-enterprise.xml** and **.mvn/extensions.xml**
- Note that the .mvn directory can be local to your project, i.e. created in the project root, or it can be global to your user settings, and stored in ~/.mvn

Setting Up a Local Cache

.mvn/extensions.xml

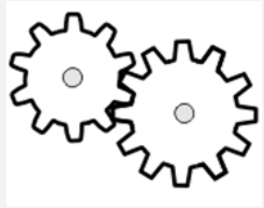
```
<extensions>
    <extension>
        <groupId>com.gradle</groupId>
        <artifactId>gradle-enterprise-maven-extension</artifactId>
        <version>1.10.2</version>
    </extension>
</extensions>
```

.mvn/gradle-enterprise.xml

```
<gradleEnterprise>
    <server>
        <url>https://gradle-enterprise.server/</url>
    </server>
</gradleEnterprise>
```

Demo: Local Maven Caching

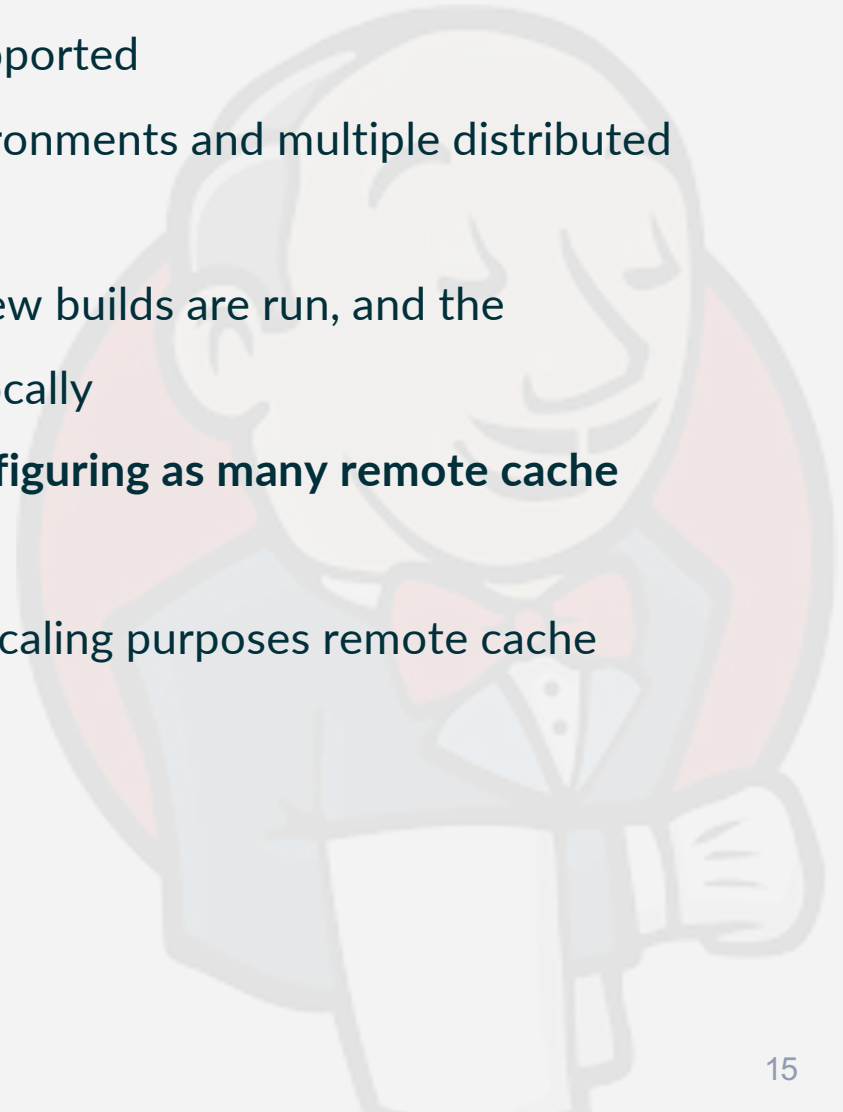
- Generate the camel-spring-boot Maven archetype
- Run the build without a build cache
- Enable the Gradle Enterprise plugin and run the build, populating the cache
- Run the build a second time, and observe the caching behavior



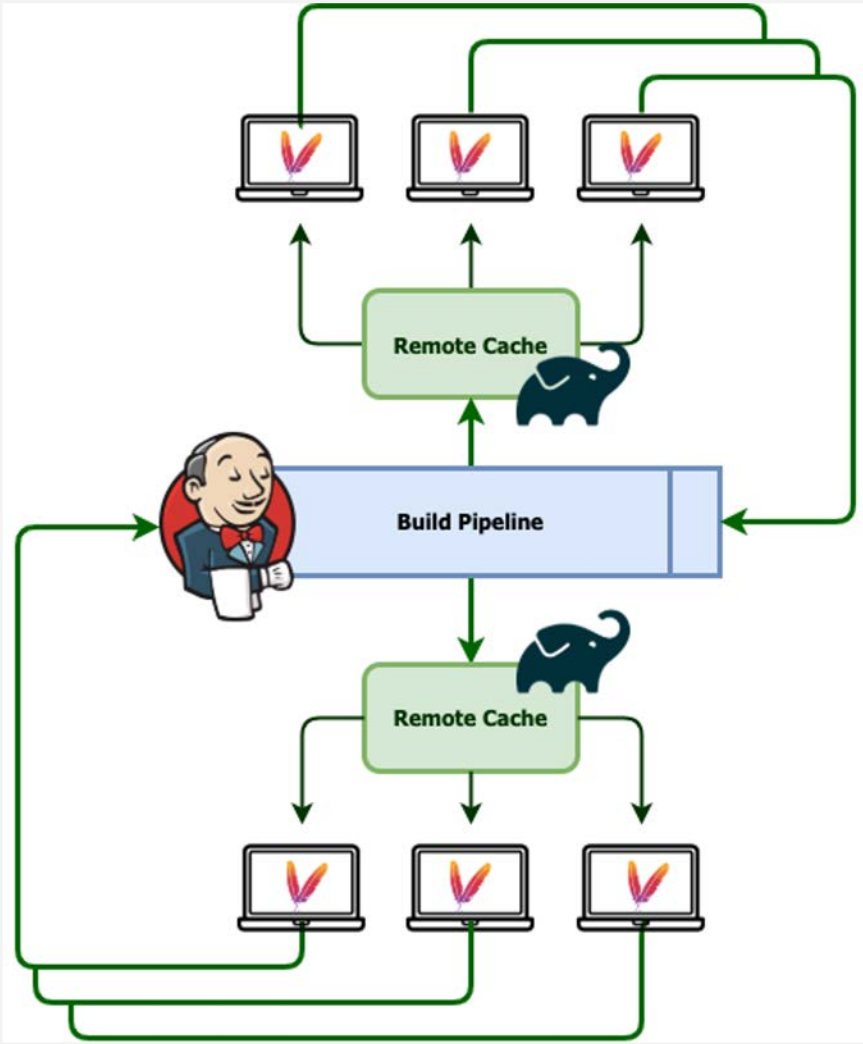
```
[INFO] --- maven-jar-plugin:2.4:jar (default-jar) @ spring-boot-camel ---
[INFO] Building jar: /Users/jreock/Development/ApacheTech/Camel/spring-boot-camel/target/spring-boot-camel-1.0-SNAPSHOT.jar
[INFO]
[INFO] --- spring-boot-maven-plugin:2.4.5:repackage (default) @ spring-boot-camel ---
[INFO] Replacing main artifact with repackaged archive
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  1.885 s
[INFO] Finished at: 2021-06-14T17:55:13-04:00
[INFO] ------------------------------------------------------------------------
[INFO] 8 goals, 5 executed, 3 from cache, saving at least 2s
[INFO]
[INFO] Publishing build scan...
[INFO] https://enterprise-training.gradle.com/s/hc6prhhznzzxm
[INFO]
jreock@Justins-MacBook-Air spring-boot-camel %
```

Integrating with CI

- Integrating with a **CI pipeline such as Jenkins** is also supported
- In a case like this, to support things like ephemeral environments and multiple distributed users, **remote caching is preferred**
- The **CI pipeline can populate** the remote cache when new builds are run, and the **developers can read** from that cache when they build locally
- Speeding up CI Maven builds then is just as easy as **configuring as many remote cache nodes** are necessary to support the users
- **Gradle Enterprise comes with a built-in cache**, but for scaling purposes remote cache nodes are recommended



Integrating with CI



Configuring a Remote Cache Node

- Remote cache nodes can be **deployed in a few ways:**
  - **Fat Standalone Jar Instances**

    https://docs.gradle.com/build-cache-node/#jar_downloads
  - **Docker** (public or private/airgapped install)

    ```
    docker pull gradle/build-cache-node:9.9
    ```

    (open source, can be saved/loaded to airgapped environment)
  - **Kubernetes**

    https://docs.gradle.com/build-cache-node/#kubernetes

Configuring Maven for Caching

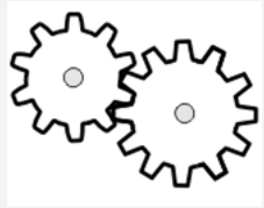
- The best practice is for
  - **Developers to read** from the remote cache
  - **CI environment to write** to those nodes

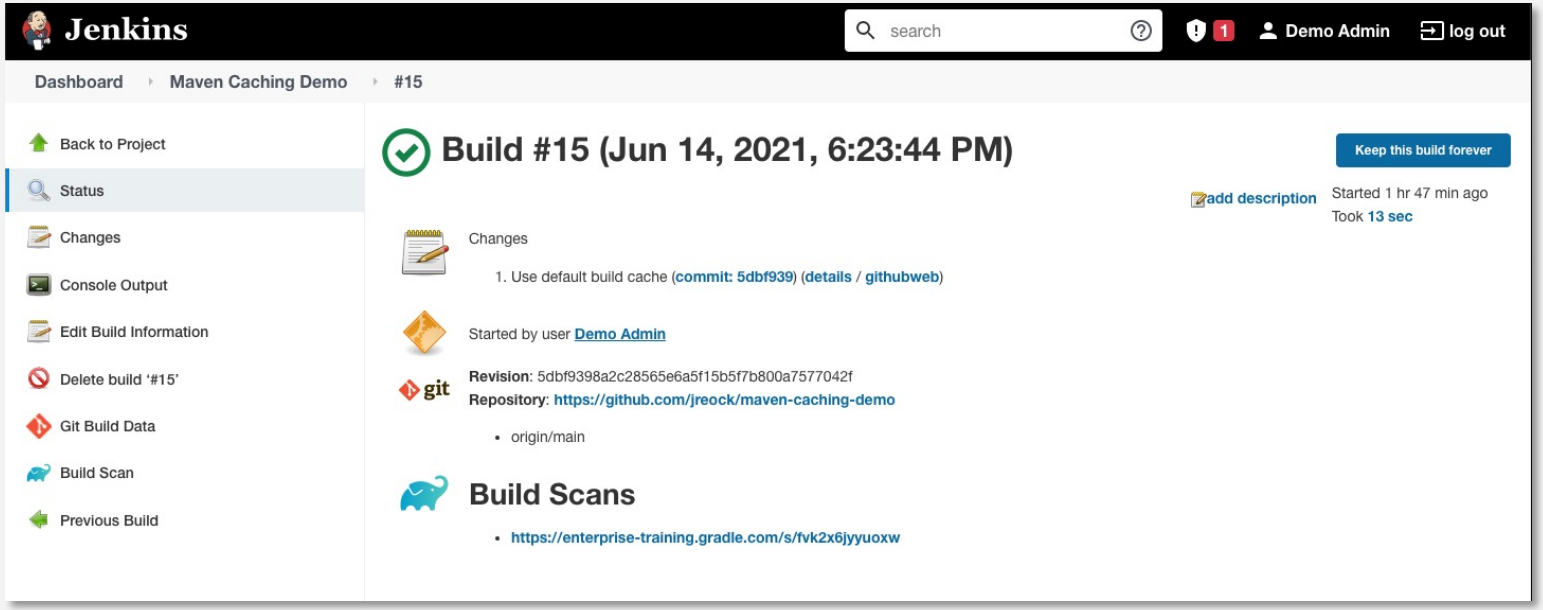
  This **maximizes the efficiency** of the build system and **reduces developer contention** over the cache
- While remote **caching is enabled** by default, remote **storage is not**
- Builds in the **CI environment** should be configured as shown, and **further access restrictions** should be configured on the Gradle Enterprise server

```
<gradleEnterprise>
    <server>
        <url>http://gradle.server/</url>
    </server>
    <buildCache>
        <remote>
            <storeEnabled>true</storeEnabled>
            <server>
                <url>http://remote-cache:5071/</url>
                <credentials>
                    <username>username</username>
                    <password>password</password>
                </credentials>
            </server>
        </remote>
    </buildCache>
</gradleEnterprise>
```

Demo: Integrating with Jenkins

- Stand up a local remote build cache node
- Update the configuration to use a remote build cache
- Deploy the code into Jenkins
- Trigger a build, and note the use of the remote build cache





Q & A

Next Steps:

- Try a free Maven Build Scan! (Hint: Just enable the Maven extension)
- Learn More about the Maven build extension: *https://docs.gradle.com/enterprise/maven-build-cache/*
- Read about Build Scans and Build Comparisons: *https://gradle.com/gradle-enterprise-solution- overview/build-scan-root-cause-analysis-data/*

Thank you!

JUSTIN REOCK – JREOCK@GRADLE.COM

FIELD CTO AND CHIEF EVANGELIST