You may want to view
the original reveal.js presentation
instead of this imperfect PDF export:
https://peter.palaga.org/presentations/220328-maven-jug-luzern-and-zurich/index.html

MAVEN

MY LIFE IS SHORT!

Peter Palaga

@ppalaga

ABOUT ME

Red Hat
Integration

APACHE
Camel

PROJECTS I BUILD OFTEN

	Lines of Java Code[1]	Maven modules[2]	`mvn clean install -DskipTests`
Camel Quarkus	93K	1371	23.5 min
Camel	1609K	612	32.6 min
Quarkus	594K	1161	11.1 min

1) `cloc --include-lang=Java .`

2) `mvn clean && find . -type f -name 'pom.xml' | wc -l`

Image from imgflip

AGENDA

- Skipping mojos
- `mvnd` a.k.a. Maven Daemon
- Vertical scaling
- Incrememental builds with GIB

```
-DskipTests
```

WHY TO SKIP MOJOS?

- After pulling from team git repository
- Source tree in a clean state
- Checked by the team CI
- Tests, and various source checks can be skipped

FIND CANDIDATES FOR SKIPPING (1/3)

`maven-buildtime-profiler`

- A Maven extension
- Outputs per-mojo per-module exec. times
- Exec. times not summed up per mojo across modules - limited usefulness
- An example

FIND CANDIDATES FOR SKIPPING (2/3)

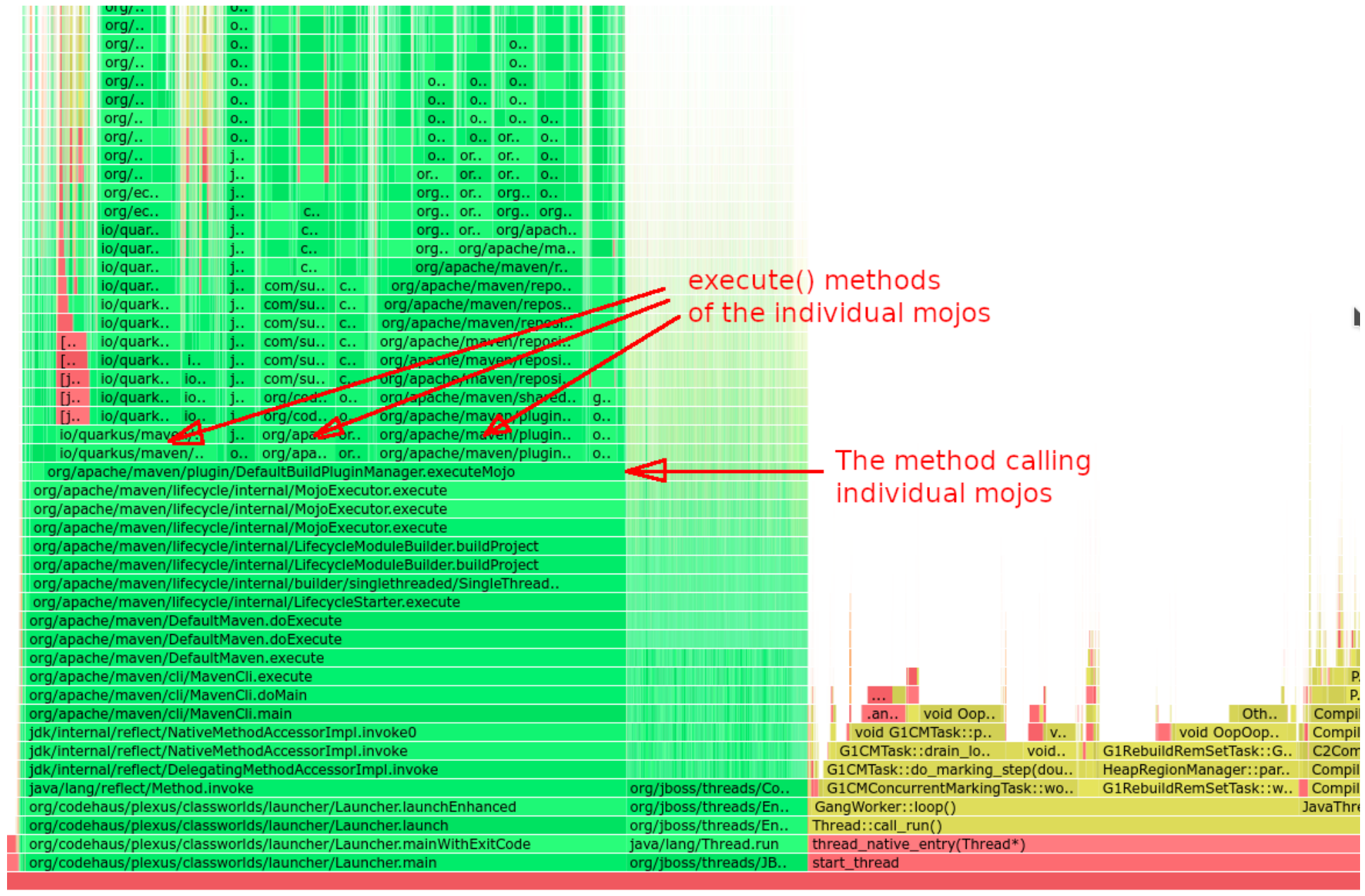
`Gradle build scans for Maven`

- A Maven extension
- Allows sorting by execution time
- An example

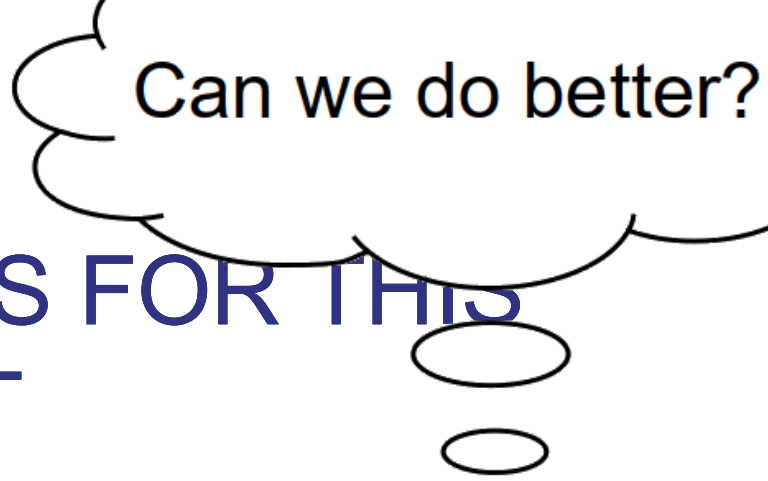
FIND CANDIDATES FOR SKIPPING (3/3)

`async-profiler` or any other Java profiler

- `export MAVEN_OPTS="-agentpath:/path/to/libasyncProfiler.so=event=cpu,file=mvn-profile.html"`
- Outputs a flame graph
- Summed mojo execution times are easy to see


execute() methods
of the individual mojos
The method calling
individual mojos
org/apache/maven/plugin/DefaultBuildPluginManager.executeMojo
org/apache/maven/lifecycle/internal/MojoExecutor.execute
org/apache/maven/lifecycle/internal/MojoExecutor.execute
org/apache/maven/lifecycle/internal/MojoExecutor.execute
org/apache/maven/lifecycle/internal/LifecycleModuleBuilder.buildProject
org/apache/maven/lifecycle/internal/LifecycleModuleBuilder.buildProject
org/apache/maven/lifecycle/internal/builder/singlethreaded/SingleThread..
org/apache/maven/lifecycle/internal/LifecycleStarter.execute
org/apache/maven/DefaultMaven.doExecute
org/apache/maven/DefaultMaven.doExecute
org/apache/maven/DefaultMaven.execute
org/apache/maven/cli/MavenCli.execute
org/apache/maven/cli/MavenCli.doMain
org/apache/maven/cli/MavenCli.main
jdk/internal/reflect/NativeMethodAccessorImpl.invoke0
jdk/internal/reflect/NativeMethodAccessorImpl.invoke
jdk/internal/reflect/DelegatingMethodAccessorImpl.invoke
java/lang/reflect/Method.invoke
org/codehaus/plexus/classworlds/launcher/Launcher.launchEnhanced
org/codehaus/plexus/classworlds/launcher/Launcher.launch
org/codehaus/plexus/classworlds/launcher/Launcher.mainWithExitCode
org/codehaus/plexus/classworlds/launcher/Launcher.main
org/jboss/threads/Co..
org/jboss/threads/En..
org/jboss/threads/En..
java/lang/Thread.run
org/jboss/threads/JB..
GangWorker::loop()
Thread::call_run()
thread_native_entry(Thread*)
start_thread

```
mvn clean install -DskipTests -Dformatter.skip -Dimpsort.skip ...
# in Camel Quarkus
```



ALL POSSIBLE SKIPS FOR THIS PROJECT

```
$ mvn clean install \
    -DskipTests \
    -Dformatter.skip \
    -Dimpsort.skip \
    -Denforcer.skip \
    -Dcamel-quarkus.update-extension-doc-page.skip \
    -Dskip.installyarn \
    -Dskip.yarn \
    -Dlicense.skip \
    -Dquarkus.build.skip
    -Dgmaven.execute.skip # underway https://github.com/groovy/gmaven/pu
...
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  04:34 min  # down from 23:27 min by mvn clean inst
```

```
public class SomeMojo extends AbstractMojo {

    @Parameter(property = "some.skip", defaultValue = "false")
    boolean skip;

    @Override
    public void execute() {
        if (skip) {
            getLog().info("Skipping as requested by the user");
            return;
        }
        ...
    }
}
```

← What happens before?

- Resolve and download plugin dependencies
- Setup a classloader, load the necessary classes
- Instantiate the Mojo
- Inject fields (potentially expensive objects)
- Call `Mojo.execute()`

How can we avoid all of this?

REMOVE PLUGINS

FROM MAVEN EXECUTION PLAN

ALTOGETHER

```
<project>
    ...                         <!-- Proper skipping -->
    <build>
        <plugins>
            <!-- Only plugins essential for the fast build -->
            <plugin><artifactId>maven-compile-plugin</artifactId></plugin>
            <plugin><artifactId>maven-jar-plugin</artifactId></plugin>
            <plugin><artifactId>maven-install-plugin</artifactId></plugin>
            ...
        </plugins>
    </build>
    <profiles>
        <profile>
            <id>full</id>
            <activation>
                <property>
                    <name>!quickly</name><!-- Active by default unless -Dquickly is passed -->
                </property>
            </activation>
            <build>
                <plugins>
                    <!-- All other plugins -->
                    <plugin><artifactId>maven-enforcer-plugin</artifactId></plugin>
                    <plugin><artifactId>maven-surefire-plugin</artifactId></plugin>
                    <plugin><artifactId>maven-failsafe-plugin</artifactId></plugin>
                    <plugin><artifactId>groovy-maven-plugin</artifactId></plugin>
                    ...
                </plugins>
            </build>
        </profile>
    </profiles>
</project>
```

even better!

```
$ mvn clean install -Dquickly
...
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  04:01 min  # down from 04:34 min by
                               #
                               #   mvn clean install -DskipTests -Dformatt
                               #
```

Java is fast *

* When warmed up

JAVA WARMUP COSTS

- JVM boot
- JIT (Just in Time) compilation

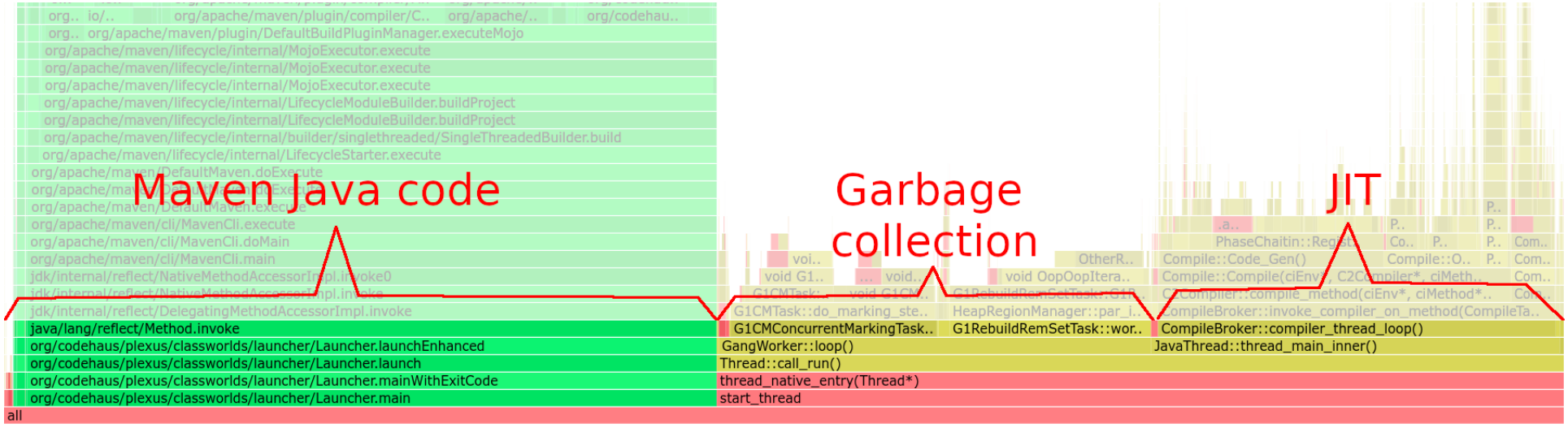
COST OF STARTING MAVEN

Measurable through a simple experiment:

```
package org.apache.maven;
public class DefaultMaven implements Maven {
    ...
    private MavenExecutionResult doExecute( MavenExecutionRequest request ) {
        System.out.println("End of Maven init: " + System.currentTimeMillis());
        request.setStartTime( new Date() ); // Maven measures from here
    }
    ...
}
```

```
$ echo $(($(date +%s%N)/1000000)) && mvn clean install -Dquickly
1634117024986
...
End of Maven init: 1634117025765
$ echo "$((1634117025765-1634117024986))"
779 # Maven start time in milliseconds
```

COST OF JIT COMPILATION




```
mvn clean install -DskipTests -Dformatter.skip -Dimpsort.skip ...
# in Camel Quarkus
```

ELIMINATE JVM WARM UP

- Have a long living Java process, a.k.a. daemon
- Pass build requests through a socket
- Like Gradle daemon, but for Maven

mvnd - the Maven Daemon

https://github.com/apache/maven-mvnd

- Started by Guillaume Nodet in 2019
- 24 releases since then
- Donated to the ASF Maven Project in December 2021

Follow @mvndaemon

`mvnd` - how to install

- ZIP file
- SDKMAN!
- Homebrew
- MacPorts
- Chocolatey
- ASDF

MAVEN DAEMON OVERVIEW

`mvnd` - the client ⇄ Daemon

- GraalVM native executable
- Looks up a running daemon
  - Or starts a new one
- Sends a build request via socket
- Receives events from the daemon
- Displays the progress

- Long running Java process
- Embeds a specific Maven version
  - Does not use any local Maven installation
- Receives build requests
- Caches plugin class loaders
- Exits after a configurable period of time

mvnd SPEED GAINS

A build with

Many modules

(relatively independent)

```
cd camel-quarkus && \
   mvn[d] clean install -Dquickly
```

Single module

```
cd camel-quarkus/extensions-core/core/dep
   mvn[d] clean install \
   -DskipTests              -Dtest=CamelBea
```

	Many modules	Single module	
`mvn` baseline	4:01 min	10.11 sec	12.79 sec
`mvnd` 1st	1:19 (3x)	9.15 (1.1x)	11.82 (1.1x)
`mvnd` 2nd	1:04 (3.8x)	2.37 (4.3x)	5.26 (2.4x)
`mvnd` 3rd	1:01 (4x)	1.19 (5.3x)	4.86 (2.6x)

Gains through

- ✓ Class loader caching
  - ✓ No repeated JIT
  - ✓ Faster JIT-compiled code
- ✓ Parallel execution

- ✓ No JVM boot cost
- ✓ Class loader caching

`mvnd` - PARALLEL BUILDS

- Default number of threads:

```
Runtime.getRuntime().availableProcessors() - 1
```

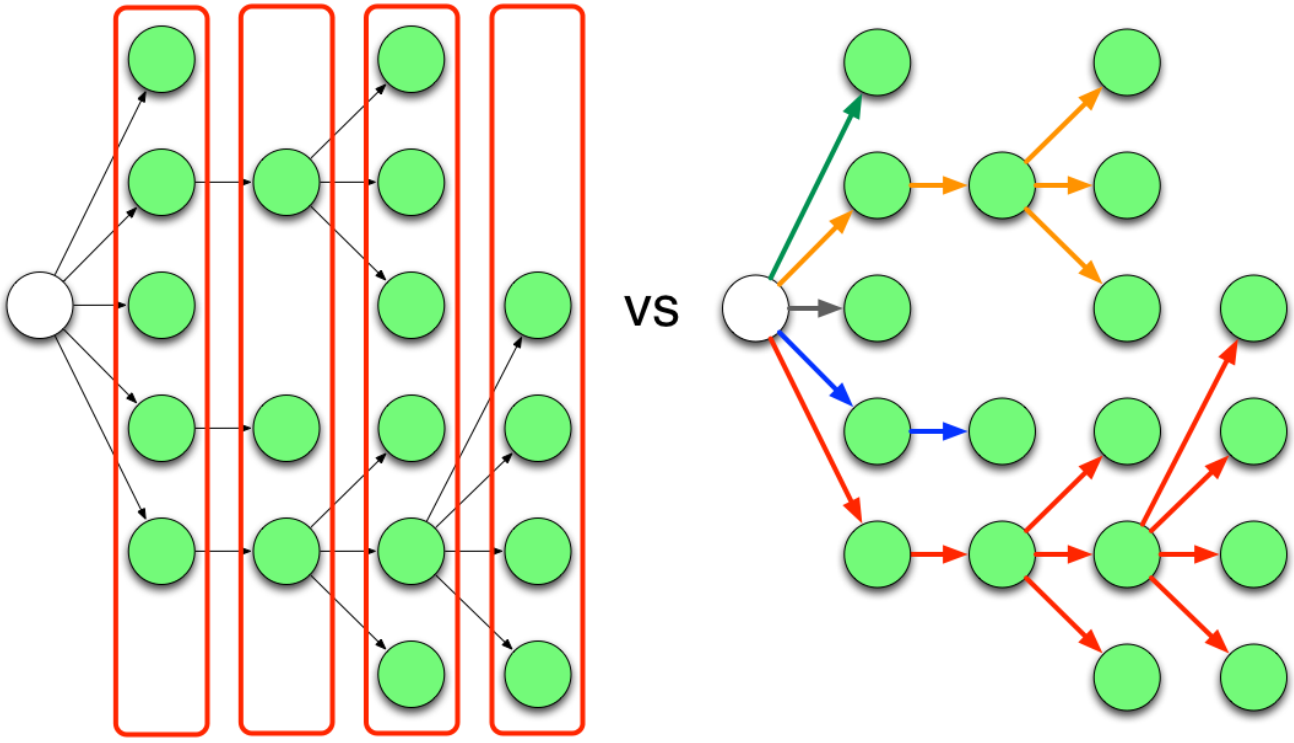
- `-T<n>` or `-T<n>C` supported like with stock Maven

MAVEN `builder`

- Pluggable `builder` since Maven 3.2.1
- A strategy for scheduling and building modules
- `singlethreaded` (default)
- `multithreaded` (with -T)

mvnd - SMART BUILDER

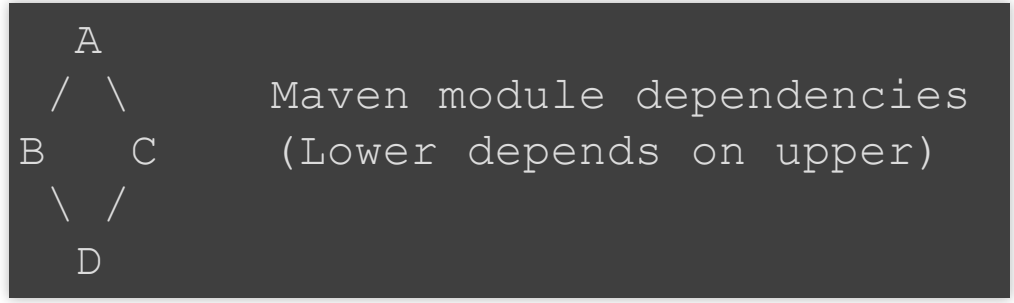
- Custom `builder` provided by `mvnd`
- Based on Takari Smart Builder



Issues of parallel builds

HIDDEN DEPENDENCIES

- No issues with serial builder
- Possible issues with a parallel builder:
  - `C` could be reading a file in `B`'s `target` folder
  - `C`'s test could dynamically read an artifact produced by `B` from the local Maven repository



Better remedy:

Make the dependency explicit

MAKE HIDDEN DEPENDENCIES EXPLICIT

```
<dependency>                                    <!-- Add this in C -->
    <groupid>org.my-group</groupid>
    <artifactid>B</artifactid>
    <version>${project.version}</version>
    <type>pom</type>
    <scope>test</scope>
    <exclusions>
        <exclusion>
            <groupid>*</groupid>
            <artifactid>*</artifactid>
        </exclusion>
    </exclusions>
</dependency>
```

This won't add any real dependency to C but it will guarantee that B is fully built before C

Issues of parallel builds

LOCAL REPOSITORY ACCESS

- Two modules may download the same artifact concurrently
- Solved in `mvnd` 0.7.1

Issues of parallel builds

BROKEN PLUGINS

Plugins may do nasty things

- Mutable global state
- Race conditions

`-1`/`--serial` may help

Better: Report/fix the issue in the given plugin

mvnd DRAWBACKS

- Blocks a few gigs of RAM (configurable)
- Better not on the CI
- Windows issues and testing catching up gradualy
- Class loader caching may have issues in some situations

mvnd --help

```
$ mvnd --help

usage: mvnd [options] [<goal(s)>] [<phase(s)>]

Options:
 -am,--also-make                          If project list is specified, also build projects
                                          required by the list
 -amd,--also-make-dependents              If project list is specified, also build projects
                                          that depend on projects on the list
 -B,--batch-mode                          Run in non-interactive (batch) mode (disables output
                                          color)
 -b,--builder <arg>                       The id of the build strategy to use
 -C,--strict-checksums                    Fail the build if checksums don't match
 -c,--lax-checksums                       Warn if checksums don't match
    --color <arg>                         Defines the color mode of the output. Supported are
                                          'auto', 'always', 'never'.
 -cpu,--check-plugin-updates              Ineffective, only kept for backward compatibility
 -D,--define <arg>                        Define a system property
 -e,--errors                              Produce execution error messages
 -emp,--encrypt-master-password <arg>     Encrypt master security password
 -ep,--encrypt-password <arg>             Encrypt server password
 -f,--file <arg>                          Force the use of an alternate POM file (or directory
                                          with pom.xml)
 -fae,--fail-at-end                       Only fail the build afterwards; allow all
                                          non-impacted builds to continue
 -ff,--fail-fast                          Stop at first failure in reactorized builds
```

mvnd UI

`+/-` - reveal/hide rolling log lines for the individual builder threads

`CTRL+B` - toggle between threaded and rolling views

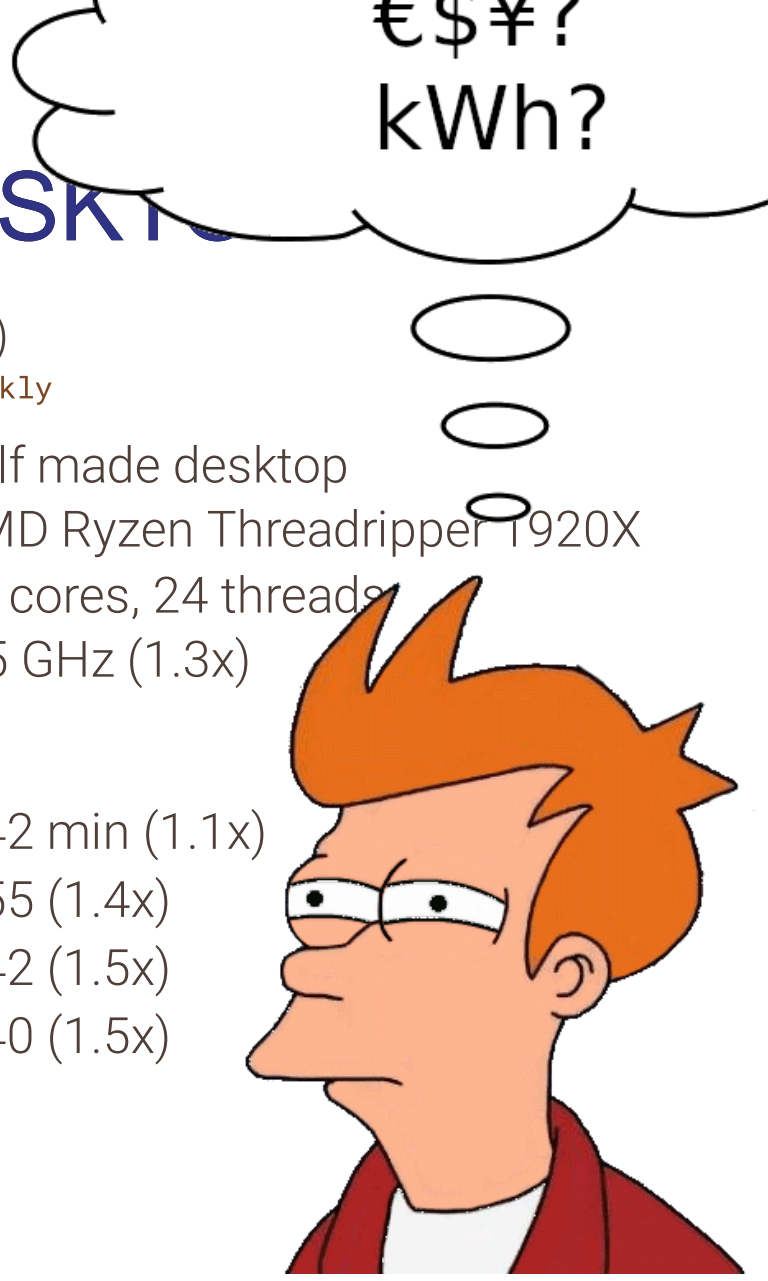
VERTICAL SCALING

LAPTOP VS. DESKTOP

A build with **Many modules** (relatively independent)

```
cd camel-quarkus && mvn[d] clean install -Dquickly
```

Machine	Lenovo ThinkPad P1 Gen 3 Core i7 10850H 12 cores 2.7 GHz (baseline)	Self made desktop AMD Ryzen Threadripper 1920X 12 cores, 24 threads 3.5 GHz (1.3x)
`mvn`	4:01 min	3:42 min (1.1x)
`mvnd` 1st	1:19	0:55 (1.4x)
`mvnd` 2nd	1:04	0:42 (1.5x)
`mvnd` 3rd	1:01	0:40 (1.5x)



COSTS OF VERTICAL SCALING

Machine	Lenovo ThinkPad P1 Gen 3 Core i7 10850H 12 cores 2.7 GHz	Self made desktop AMD Ryzen Threadripper 1920X 12 cores, 24 threads 3.5 GHz (1.3x)
Purchase price	2630,- € (Oct. 2021)	~1300 € (Feb. 2020, w/out 🖥️ ⌨️ 🖱️)
Power suply unit	135 W	650 W
Idle	~18 W*	~60 W*
Max	129 W*	278 W*
2h work session (builds, web browsing)	0.06 kWh*	0.21 KWh*
€/year (0.3 €/kWh)	23 €	80 €

*) Measured with a budget power meter from a hobby market

Incremental builds with

GITFLOW INCREMENTAL BUILDER

(GIB)

WHAT IS GIB?

https://github.com/gitflow-incremental-builder/gitflow-incremental-builder

- A maven extension
- Compares the current topic branch with the reference branch
  - typically `origin/main` or `origin/master`
- Build/test only the modules impacted by the change

GIB IN QUARKUS

https://github.com/quarkusio/quarkus/blob/main/.github/workflows/ci-actions-incremental.yml

Multiple steps to be able run tests on multiple nodes in parallel:

- Build the whole tree w/out tests and identify impacted modules
- Calculate test jobs
- Run test jobs in parallel

Compare the number of tests and execution times per pull request

WRAP UP

Techniques to speed up Maven builds

- ✓ Skip unessential mojos
- ✓ Maven daemon to keep the building JVM warm
- ✓ Larger machine
- ✓ Incremental builds with GIB

LINKS

- ✓ https://github.com/mvndaemon/mvnd
- ✓ https://peter.palaga.org/blog.html
- ✓ https://github.com/gitflow-incremental-builder/gitflow-incremental-builder