Java 9,10,11

New Capabilities

Java has done something right for sure

- Humungous enterprise projects are today running on java and new projects continue to be built with java
- Has high market share of the programming world: https://www.tiobe.com/tiobe-index/
- Maturity of libraries, tools, community, documentation and eco system in general is very hard to beat

- JVM is hard to beat even today:
  - Garbage collection
  - Optimisations across processors
  - Reliability
  - Jython, Kotlin, Clojure, Groovy, Scala and Jruby

- Enterprises like when a corporate stands in and says - hey I have your back for next n years

- Enterprises also like when the decision making process on future path is democratic and there is no vendor lock in (Ex: Amazon Corretto, OpenJdk)

- Having said all this - frontends and mobile platforms are a different story that Java has not fit in.

Modularity

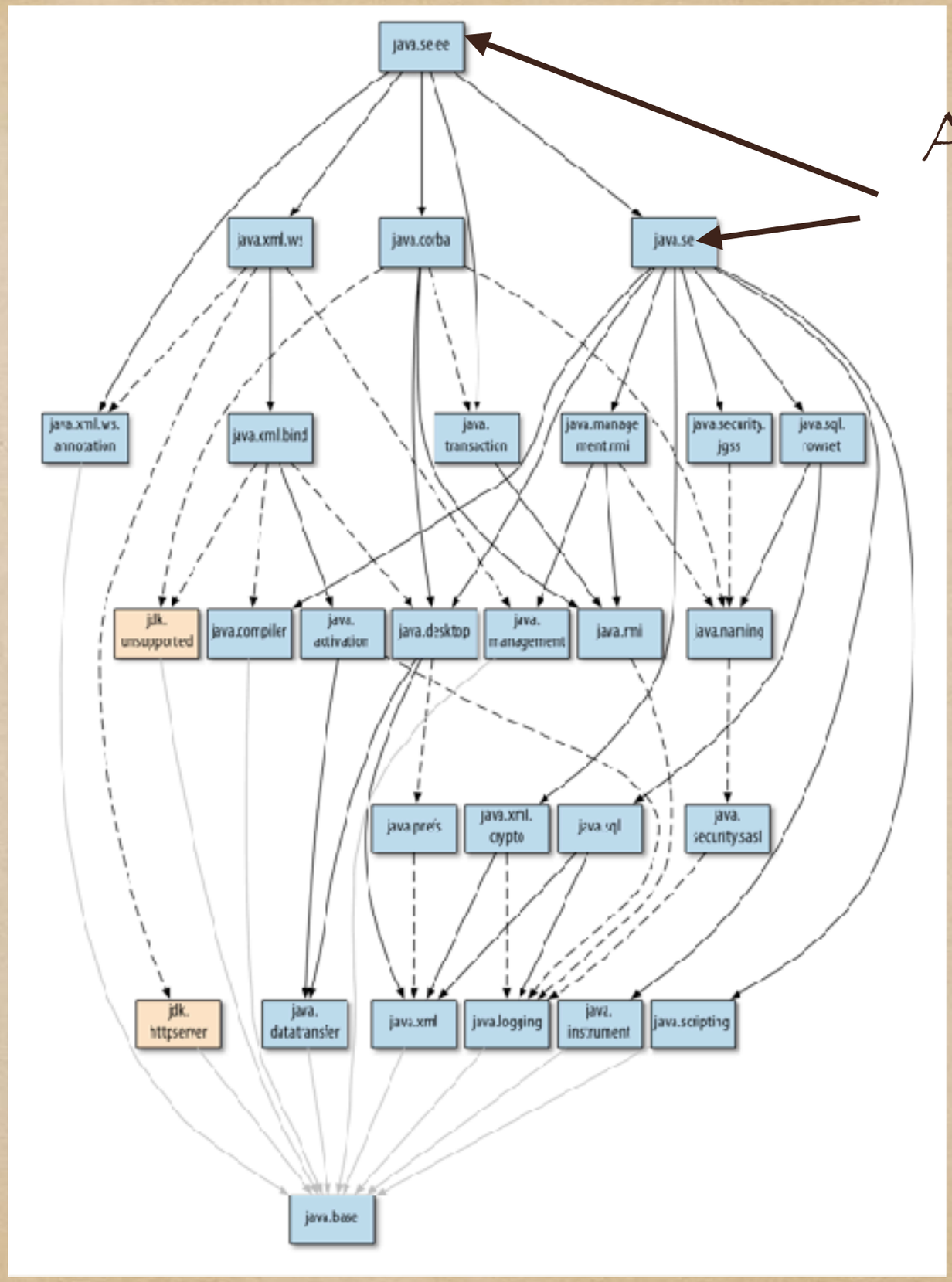
- Modularity is driven from ground up
  - Packages vs Namespaces
  - Jars
  - Maven, gradle, etc
  - Classpath hell is better than DLL hell

Platform Module System (JPMS)

- Java has accumulated a lot of legacy code as a platform.
- It has been a platform that works on IoT devices to large servers, swing, awt, legacy data structures, old security spec, etc. So it has everything that anybody needs in one monolith.
- JPMS was designed to solve this: module—a uniquely named, reusable group of related packages and resources

rt.jar

- its about 60 MB on jdk 8. JRE itself hits about 150MB
- Take CORBA: The classes supporting CORBA in the JDK are still present in rt.jar to this day.
- No matter whether you use CORBA or not, the classes are there.
- unnecessary use of disk space, memory, and CPU time.
- Cognitive overhead of obsolete classes showing up in IDE autocompletions and documentation during development.
- Same goes for AWS/Swing/JavaFx
    - Backward compatibility is one of the most important guiding principles for Java



Aggregator Modules

Optimisation

- Because the module system knows which modules belong together, including platform modules, no other code needs to be considered during JVM startup.
- Before modules, this was much harder, because explicit dependency information was not available and a class could reference any other class from the classpath.
- find jmods and use the jmod tool
  - `$JAVA_HOME/bin/jmod list java.net.http.jmod`

Lets create some libraries

- httplib: A library that exposes:

```
public String get(String url)
```

- Put this into an independent project

- Create another library: weather

```
public Weather getWeather(String city)
```

- weather depends on httplib. Get it running with a main() in weather project

Httplib get

```java
public String get(String url) throws Exception {
    URL obj = new URL(url);
    HttpURLConnection con = (HttpURLConnection) obj.openConnection();

    con.setRequestProperty("User-Agent", "APIClient");

    int responseCode = con.getResponseCode();
    System.out.println("\nSending 'GET' request to URL : " + url);
    System.out.println("Response Code : " + responseCode);

    BufferedReader in = new BufferedReader(new InputStreamReader(con.getInputStream()));
    String inputLine;
    StringBuffer response = new StringBuffer();

    while ((inputLine = in.readLine()) != null) {
        response.append(inputLine);
    }
    in.close();

    return response.toString();
}
```

URL for Weather:
http://api.openweathermap.org/data/2.5/weather?q=CITY&APPID=cc3d457fb9707805588d131ce51804be

For Forecast:
http://api.openweathermap.org/data/2.5/forecast?q=CITY&APPID=cc3d457fb9707805588d131ce51804be

GSON Parsing:

```java
Gson gson = new Gson();
HashMap<String, Object> dataMap = gson.fromJson(wjson, HashMap.class);
```

module-info.java

```
module Java11 {
    requires java.sql;

    exports com.mydomain;
}
```

- Readability
- Accessibility - No types from a non exported package can be used by other modules—even if types inside that package are public
- Setup module dependencies

Modules and Automatic Modules

- Setup module in Httplib

```
module httplib {
    exports com.mydomain.httplib;
}
```

- Setup module in weather. Add gson automatic module

```
module weatherlib {
    requires httplib;
    requires gson;
    requires java.sql;
}
```

- Setup classpath dependencies and run in IDE

- Run on command line

```
java --module-path Weather.jar:HttpLib.jar:gson-2.8.5.jar -m weatherlib/
com.mydomain.weatherplus.WeatherService
```

- Transitive read relationships

```
requires transitive java.logging;
```

- Requiring java.se.ee or java.se in application modules isn't a good idea. It means you're effectively replicating the pre-Java 9 behavior of having all of rt.jar accessible in your module.

- Qualified Exports

```
exports com.mydomain to com.yourdomain.util;
exports jdk.internal.jimage to jdk.jlink;
```

- Bad idea because this goes against fundamental premise of not knowing your users

Enhance with transitive

- Create another library weatherpluslib

```
public Weather getTomorrowsForecast(String city)
public Weather getWeather(String city)
```

- Uses weatherlib and httplib to make calls
- Run in IDE, run on command line by building jars

- Code compiled and loaded outside a module ends up in the unnamed module
  - The unnamed module is special: it reads all other modules
- In Java 8 and earlier, you could use unsupported internal APIs without repercussions. With Java 9, you cannot compile against encapsulated types in platform modules
- To aid migration, code compiled on earlier versions of Java using these internal APIs continues to run on the JDK 9 classpath for now

- You can still use identifiers called module, requires, etc in your other source files, because the module keyword is a restricted keyword.
  - It is treated as a keyword only inside module-info.java
- Java executable has a new —module, —show-module-resolution, —module-path options
- Using the explicit dependency information of module descriptors, module resolution ensures a working configuration of modules before running any code. (Unlike classnotfound exceptions)

Linking Modules

- An optional linking phase is introduced with Java 9, between the compilation and run-time phases

- jlink command and its options

```
/Library/Java/JavaVirtualMachines/jdk-11.0.1.jdk/Contents/Home/bin/jlink --module-path WeatherPlus.jar:Weather.jar:HttpLib.jar:gson-m-2.8.5.jar --add-modules weatherpluslib --output linkedfiles
```

- Jlink creates a package

DI with service providers

- One module can declare the API
- There can be many modules providing implementations for the API
- Module declarations can declare API and implementation. Scanning happens automatically

Providers for Pre init check

- Module that declares the API

```
module WeatherApi {
    exports com.mydomain.weather.weatherapi;
    uses WeatherApi;
}
```

- Module that implements

```
module weatherpluslib {
   requires WeatherApi;
   provides WeatherApi with WeatherServicePlus;
}
```

- Module that uses

```
module WeatherClient {
    requires WeatherApi;
}
```

- If A uses B, C provides B and requires D, A cant start till D is available in module path though it has no knowledge of C
- JLink however does not care about providers and hence does no service binding!
  - reason to not do automatic service binding in jlink is that java.base has an enormous number of uses clauses. The providers for all these service types are in various other platform modules. Binding all these services by default would result in a much larger minimum image size
  - Use —add-modules to include the providers in jlink

Unified Logging

- Common logging system for JVM components
- new command-line option -Xlog
- The syntax for configuring the logging:
  - -Xlog[:[selections][:[output][:[decorators][:output-options]]]]
  - java -Xlog:help will list all options

- tags:gc,cpu,class,thread
- filters:uptime,timemillis,pid,tid,level,tags

Try Some

- Levels: off, trace, debug, info, warning, error
- -Xlog:module*:log.txt
- -Xlog:class+load
- -Xlog:gc+phases:gcphases.txt
- -Xlog:all,verification=off
- -Xlog:all,verification=off,class*=off::uptime

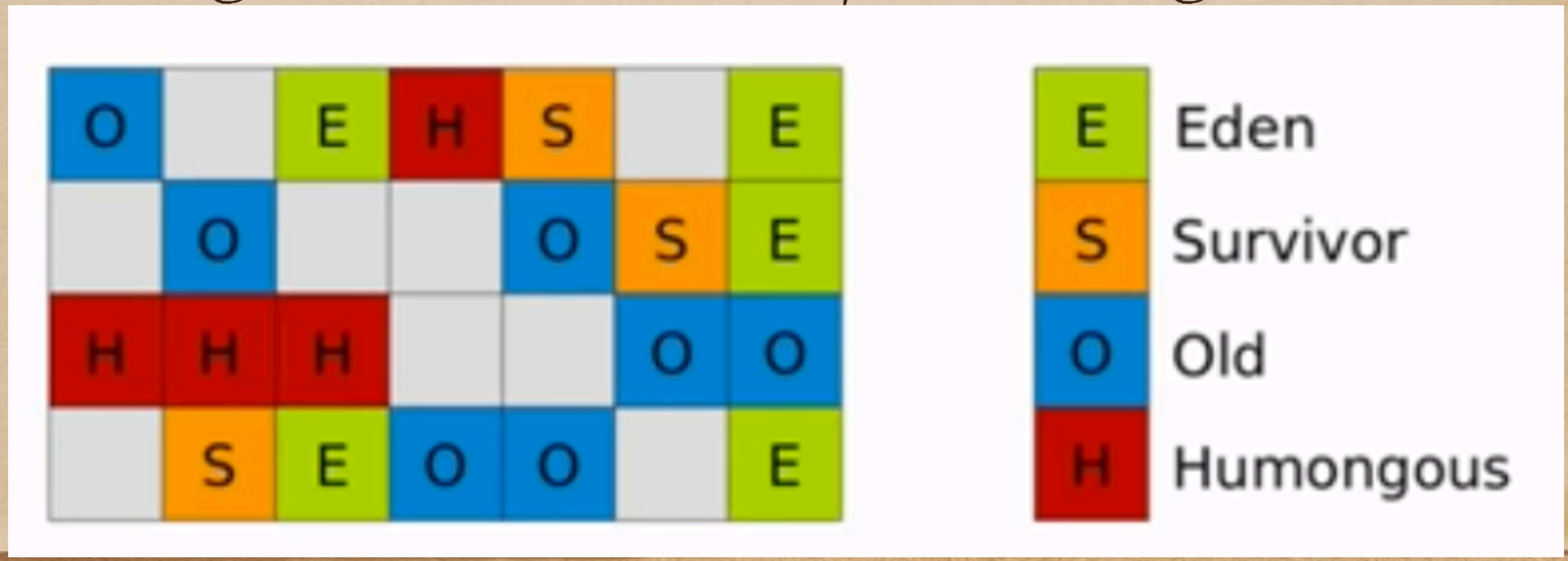
G1 Garbage Collector

- Hotspot low pause, generational collector
- Replacement for Concurrent Mark n Sweep
- Low pause over throughput unlike parallel GC
- Just simple tuning params - XX:GCTimeRatio=n, XX:MaxGCPauseMillis=n - these settings play with the heap size within the Xmx and Xms boundaires
- More things that alter the heap size dynamically
  - -XX:MinHeapFreeRatio 40
  - -XX:MaxHeapFreeRatio 70

More on G1

- G1 is a generational, incremental, parallel, mostly concurrent, stop-the-world, and evacuating garbage collector which monitors pause-time goals in each of the stop-the-world pauses.
- G1 reclaims space mostly by using evacuation

- G1 divides heap into 2048 regions by default (tuneable)
- Each region is tagged as Eden, Survivor, Old, Humungous (Contains one obj > 50% of region)



- G1 prepares Eden region when JVM starts
- All allocation goes into Eden
- When Edens are full, young generation collection happens (Stop the world)
- G1 tracks pointers between regions (old to eden for example)

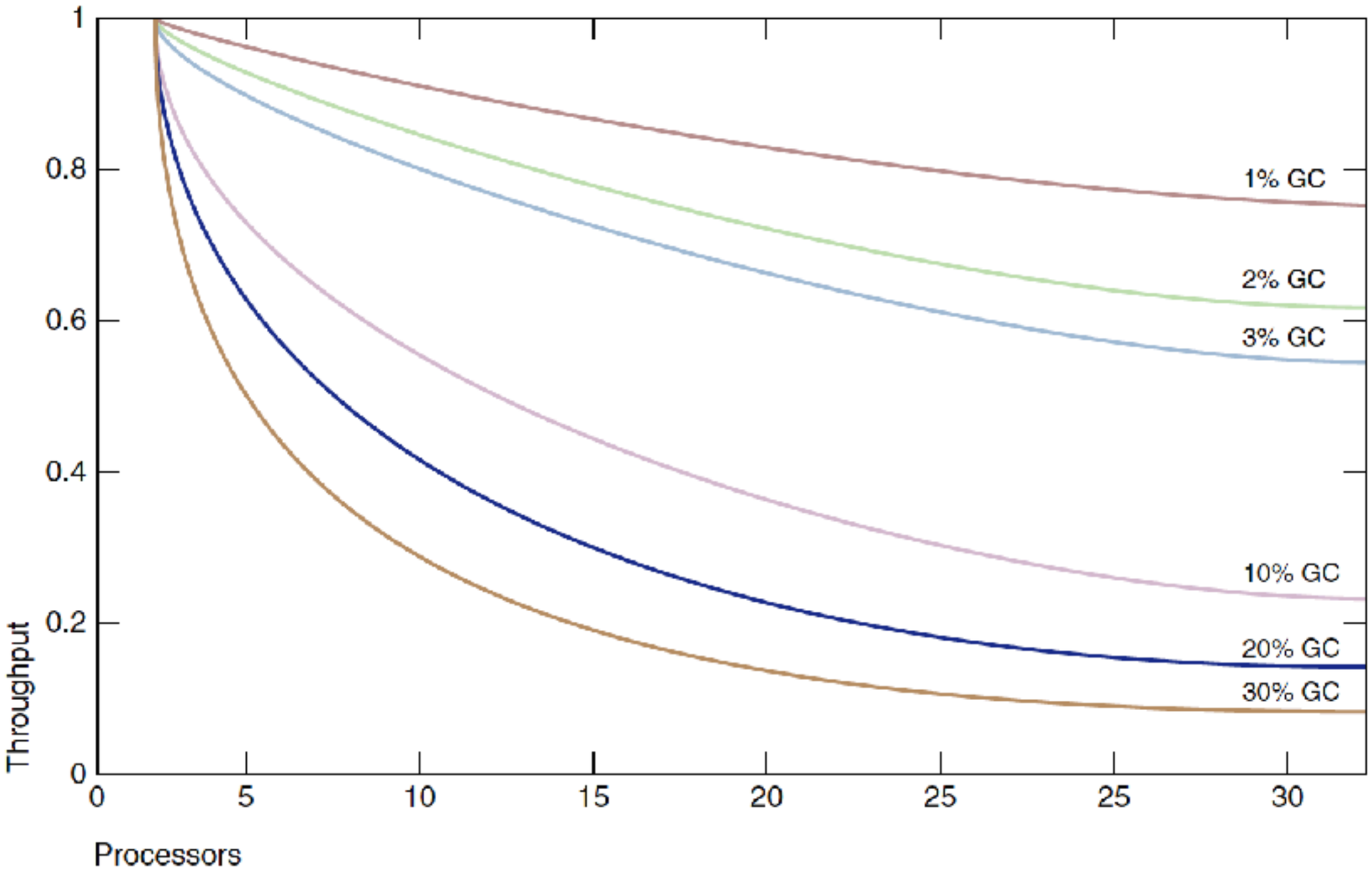
Young Generation Size

- The proportion of the heap dedicated to the *young* generation affects performance.
- The bigger the *young* generation, the less often minor collections occur.
- For a bounded heap size, a larger *young* generation implies a smaller old generation
  - This increase the frequency of major collections.
- The optimal choice depends on the lifetime distribution of the objects allocated by the application.

Some Tuneable Params

- -XX:NewRatio=n : ratio between the young and old generation
- -XX:NewSize and -XX:MaxNewSize to fine tune the young generation sizes

Significance of Pauses



Guidelines

- First decide on the maximum heap size that you can afford to give the JVM
- Plot your performance metric against the young generation sizes
- If the total heap size is fixed, then increasing the young generation size requires reducing the old generation size. Keep the old generation large enough to hold all the live data used by the application at any given time, plus some amount of slack
- Subject to the previously stated constraint on the old generation:
  - Grant plenty of memory to the young generation.
  - Increase the young generation size as you increase the number of processors because allocation can be parallelised.

GC Options

- Serial GC: -XX:+UseSerialGC
- ParallelGC -XX:+UseParallelGC
- G1 Collector: -XX:+UseG1GC

Lets try

- Single thread huge young alloc: Parallel /G1 gc, low heap (256m), high heap(4g)
- Single thread huge old alloc: Parallel /G1 gc, high heap(4g)
- Same things with the multi threaded version
- Mix of old and young alloc

REPL with JShell

- jshell ⏎ /help
- jshell> System.out.println(...)
- jshell> 2 + 3
- jshell> var k = 4 + 9
- jshell> 10>k || 4<k
- jshell> /set feedback ⏎
- jshell> /set feedback verbose ⏎

REPL with Jshell

- Define a method
- public void printStatus() {
- ...> System.out.println(k);
- ...> System.out.println(accStatus);
- ...> }
- Call the method (You can use tab for auto completion)

REPL with Jshell

- /list
- /edit 4 ⏎ And edit the contents
- call a method display(str) instead of S.o.p and try to call the method
- String and press tab - view documentation
- /var or /methods

REPL with jshell

- Create a class
- Create a reference to the class and instantiate
- Create a reference to a non existing class

- /types
- List names = new ArrayList()
- /import
- Connection c;
  import java.sql.*;

REPL with Jshell

- /open <text file>
- /save <text file name>
- jshell test.txt
  - put a /exit as last line in the txt file

Some good uses

- Count letters in a string in with jshell
  - System.getProperty to fetch parameters
    - -R-Dparamname=value to pass params
- If we want it to be part of a shell script:
- echo "System.out.println(\"$JAVA_HOME\".length());" | jshell -s | sed -n '2p'

- Print all env variables starting with J and ending with E

```
env > tmp.txt
echo "List<String> lines = Files.readAllLines(Paths.get( \"tmp.txt\")); \
lines.stream().filter(l -> l.matches(\"^J.*\")).forEach(l -> System.out.println(l));" \
                | jshell -q | sed -n '2p'
```

Multi Release Jars

- How often we have to keep our core libraries at pre java 6 versions to support multiple clients
- Java 8 cannot be helped but from Java 9 onwards we can take advantage of new features of language with same jar

Multi Release Jars

- A multi-release JAR file is a JAR file whose MANIFEST.MF file includes the entry Multi-Release: true in its main section.
- META-INF contains a versions subdirectory whose integer-named subdirectories -- starting with 9 -- store version-specific class and resource files

Jar Structure

```
JAR content root
  A.class
  B.class
  C.class
  D.class
META-INF
   MANIFEST.MF
   versions
      9
         A.class
         B.class
      10
         A.class
```

Lets setup and build

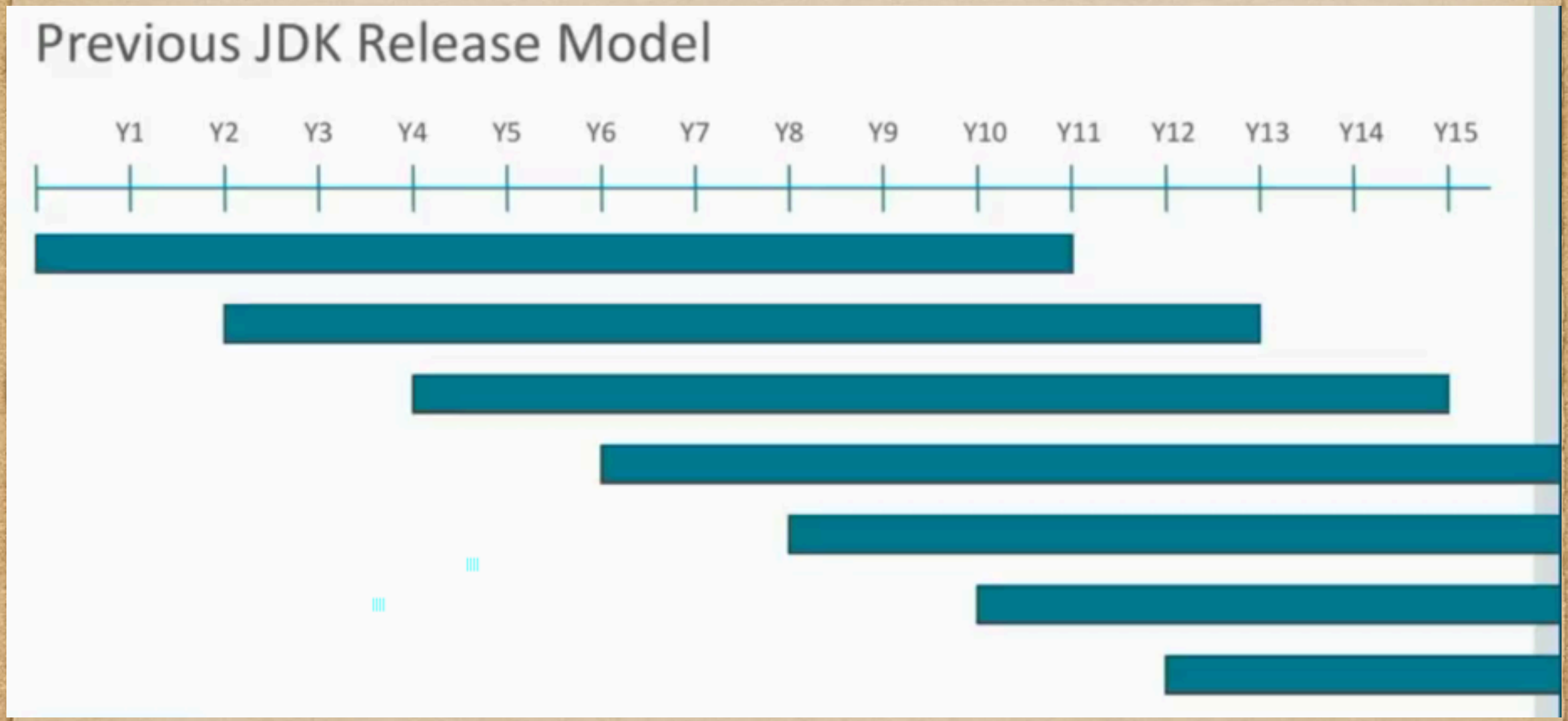
- Java *8*

```
public static long getPid() {
    System.out.println("Running java 8 process id util");
    // name is in the format <pid>@<hostname>. (Usually)
    final String jvmName = ManagementFactory.getRuntimeMXBean().getName();

    final int index = jvmName.indexOf('@');
    if (index < 1) {
        return 0; // No PID.
    }

    try {
        return Long.parseLong(jvmName.substring(0, index));
    } catch (NumberFormatException nfe) {
        return 0;
    }
}
```

- Java *9*

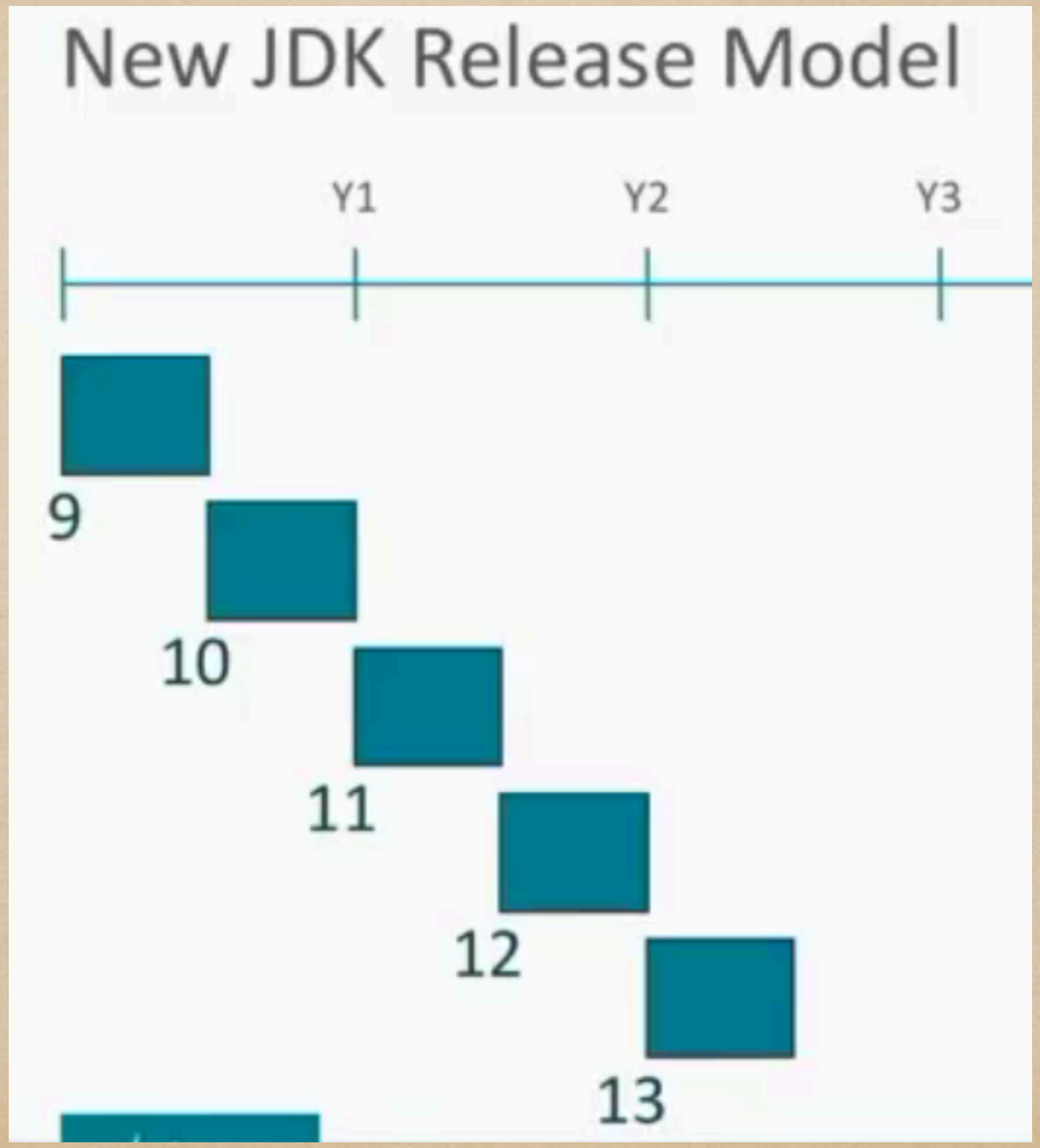
```
public static long getPid() {
    System.out.println("Running java 9 process id util");
    return ProcessHandle.current().pid();
}
```

Why we have 3 major versions in under 2 years!



Previous JDK Release Model

New JDK Release Model



How is back support handled

Oracle Licensing

- Oracle moves towards paid subscription model
- Also open sources commercial features so it can be available in OpenJdk
  - Java Mission Control
  - Java Flight Recorder
  - Application Class Data Sharing

Open JDK GPLv2

it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

var

- var can be used to define a reference as long as the type can be inferred from its right hand side.

```
var k = 0;
var m = "123";
k = m;
k = new Object()
```

- var does not make java a weakly typed language
  - Decompile a method that uses var

var

- var is allowed only as local variables
  - Cannot be used on class fields
  - Cannot be used in method signatures
- What happens in below case?

```
var k = {1, 2, 3};
int[] m = {1, 2, 3};
```

So does it hurt readability?

```
public Weather getTomorrowForecast(String city) throws Exception {
    HttpLib lib = new HttpLib();
    String wjson = lib.get(WEATHER_FORECAST_URL.replaceAll("CITY", city));
    Gson gson = new Gson();
    HashMap<String, Object> dataMap = gson.fromJson(wjson, HashMap.class);
    Weather retVal = new Weather();
    retVal.lowTemp = ((Double) ((Map) ((Map) ((ArrayList)
        dataMap.get("list")).get(2)).get("main")).get("temp_min"))
           .floatValue() - ZERO_KELVIN;
    retVal.highTemp =
           ((Double) ((Map) ((Map) ((ArrayList)
        dataMap.get("list")).get(2)).get("main")).get("temp_max"))
                   .floatValue() - ZERO_KELVIN;

    return retVal;
}

public Weather getTomorrowForecast2(String city) throws Exception {
    var lib = new HttpLib();
    var wjson = lib.get(WEATHER_FORECAST_URL.replaceAll("CITY", city));
    var gson = new Gson();
    var dataMap = gson.fromJson(wjson, HashMap.class);
    var retVal = new Weather();
    retVal.lowTemp = ((Double) ((Map) ((Map) ((ArrayList)
        dataMap.get("list")).get(2)).get("main")).get("temp_min"))
           .floatValue() - ZERO_KELVIN;
    retVal.highTemp =
           ((Double) ((Map) ((Map) ((ArrayList)
        dataMap.get("list")).get(2)).get("main")).get("temp_max"))
                   .floatValue() - ZERO_KELVIN;

    return retVal;
}
```

Generics and Wildcards

```
List<String> names = x;
Map<String, List<Integer>> subjectScores = y;
```

```
var names = x;
var subjectScoresMap = y;
```

New Collection API

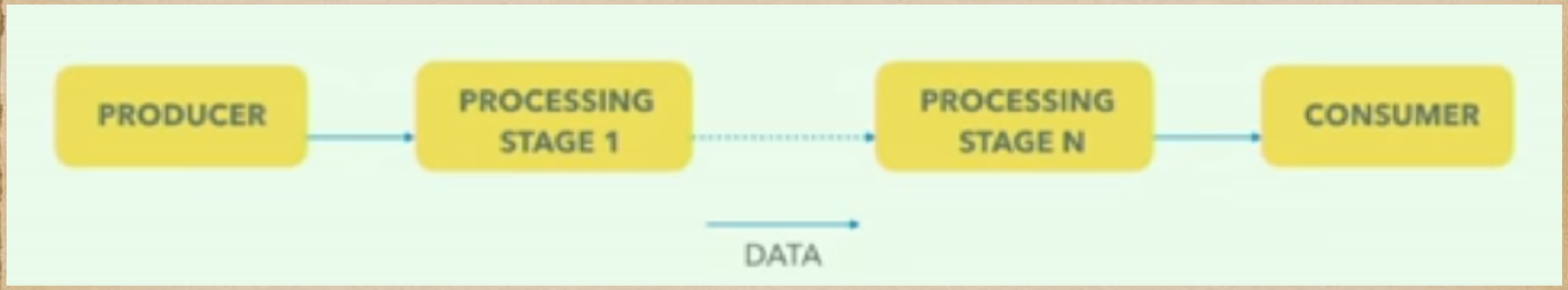
- `List<String> list = List.of("Hello", "World", "from", "Java");`
- `Set<String> set = Set.of("Hello", "World", "from", "Java");`
- `Map<String, Integer> cities = Map.of("Brussels", 1_139000, "Cardiff", 341_000);`
- `Map<String, Integer> cities = Map.ofEntries(entry("Brussels", 1139000), entry("Cardiff", 341000));`
- The new Set/Map collections change their iteration order from run to run, hopefully flushing out order dependencies earlier in test or development.

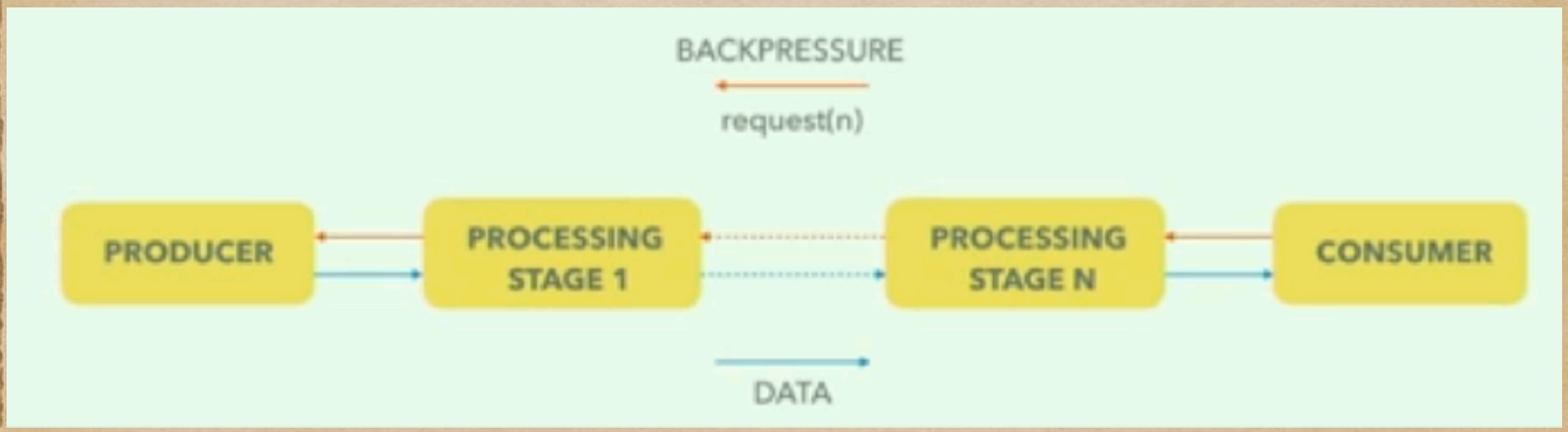
Spin Waits

- Tells JVM to cut CPU cycles to a thread/loop
- Measure CPU in both cases

```
while(true) {
    while(!newConnectionMade()) {

    }
    processConnection();
}

while(true) {
    while(!newConnectionMade()) {
        Thread.onSpinWait();
    }
    processConnection();
}
```

Reactive Streams





Flow.Publisher

- Produces items of type T to be consumed by subscribers
- Multiple subscribers receive events in the same order
- subscribe() method is used to register with publisher

Subscriber

- Subscribes to a publisher
- onSubscribe event is called to confirm and hand over a "Subscription" object
- items are delivered via onNext(T)
- completion signal via onComplete(T)

Subscription

- Subscriber can issue back pressure via request(n) method
- Subscribers cancel subscription using the cancel() method

Create a bare bones implementation

- Support only one subscriber
  - request one event at a time after processing event
  - Publish and receive strings - just print the consumed strings

True Reactive

- To be truly reactive everything has to run asynchronous
- One simple implementation available with JDK is SubmissionPublisher