Java Update: Performance Enhancements

2022 TPF Users Group Conference
March 27-30, Dallas, TX
Application Development

—

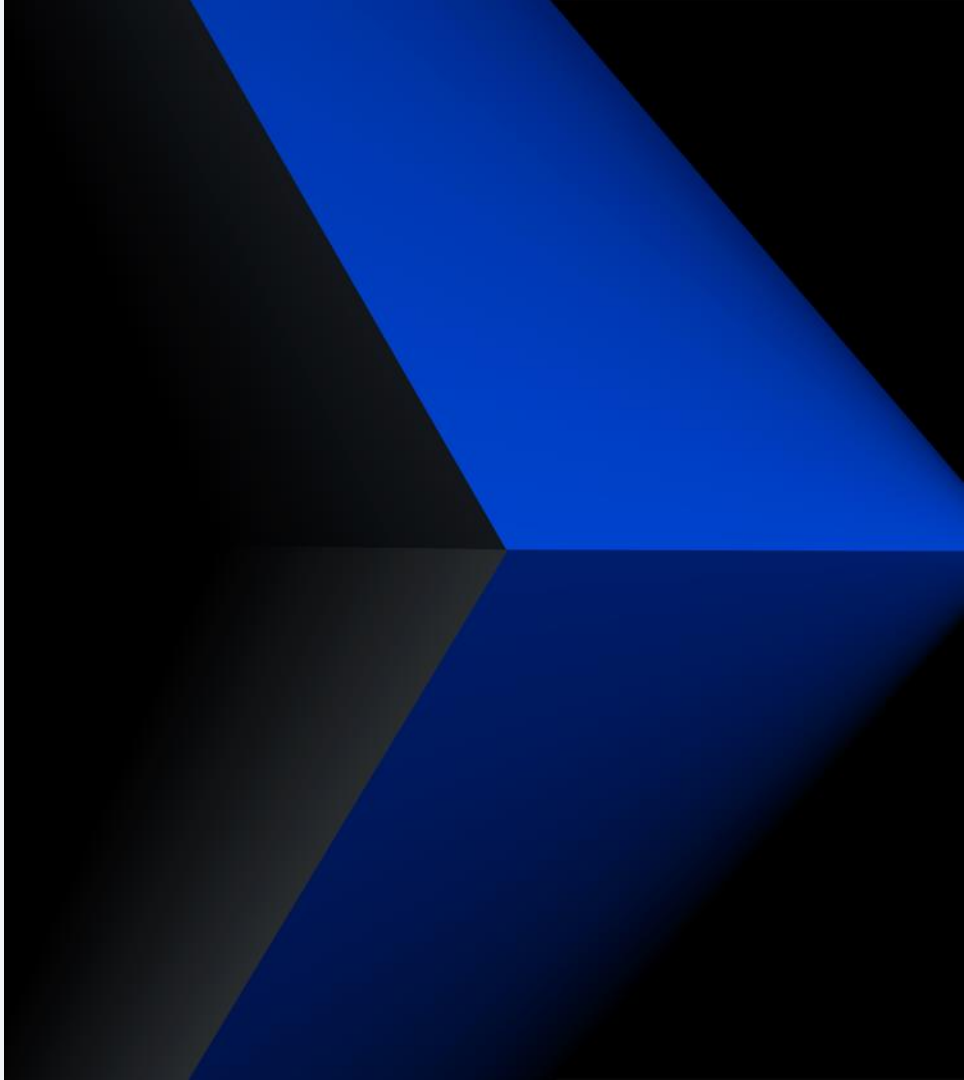
Dan Gritter

IBM Z

IBM

Disclaimer

Any reference to future plans are for planning purposes only. IBM reserves the right to change those plans at its discretion. Any reliance on such a disclosure is solely at your own risk. IBM makes no commitment to provide additional information in the future.



Java Performance Enhancements in 2021 (Recap)

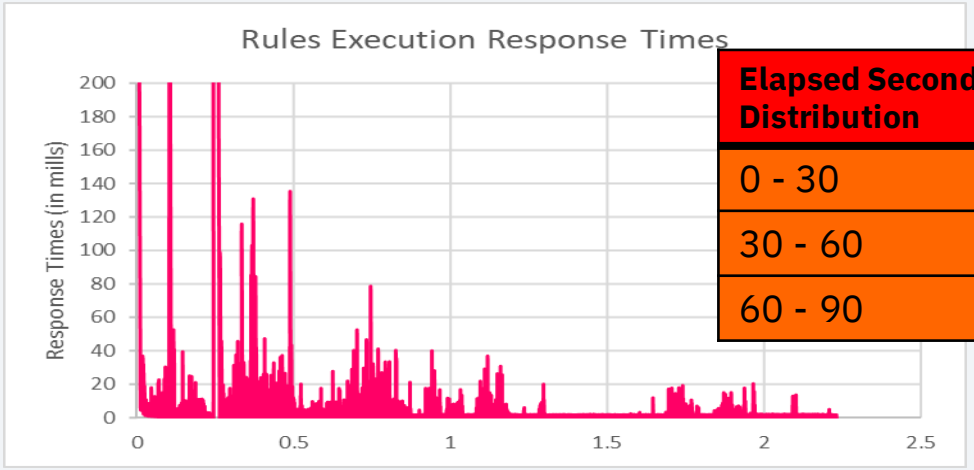
1. JIT Performance Improvements (2021 1Q Java Refresh - PJ46432)
2. Hardware-Assisted Encryption Enabled (2021 2Q Java Refresh - PJ46547)
3. Garbage Collection Performance Improvements (2021 2Q Java Refresh - PJ46547)
4. Other noteworthy items 2021

Java direction / OpenJDK migration

JIT Performance Updates - PJ46432

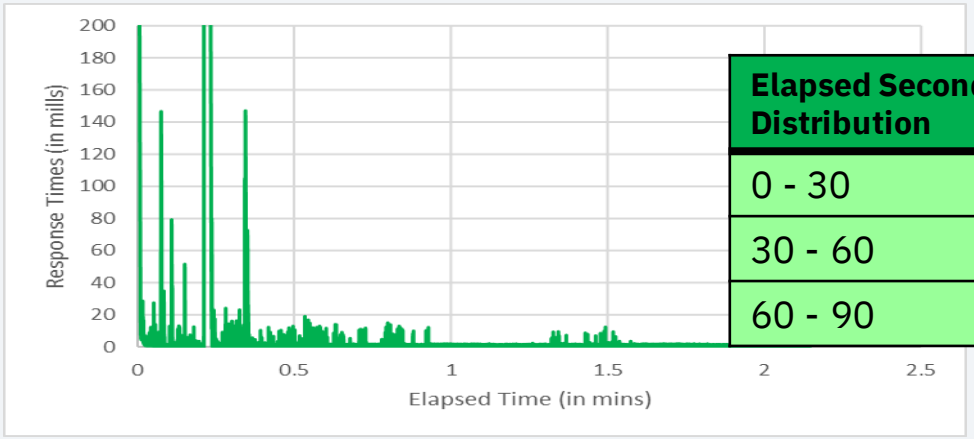
55% reduction in response time in first 30 seconds!
45% reduction in next 30 seconds

z/15, 8 I-streams
2 JVM Rules Engine JAM
Starting with Traffic



Before

Elapsed Seconds Distribution	Average Response (mills)
0 - 30	1.21
30 - 60	.45
60 - 90	.24



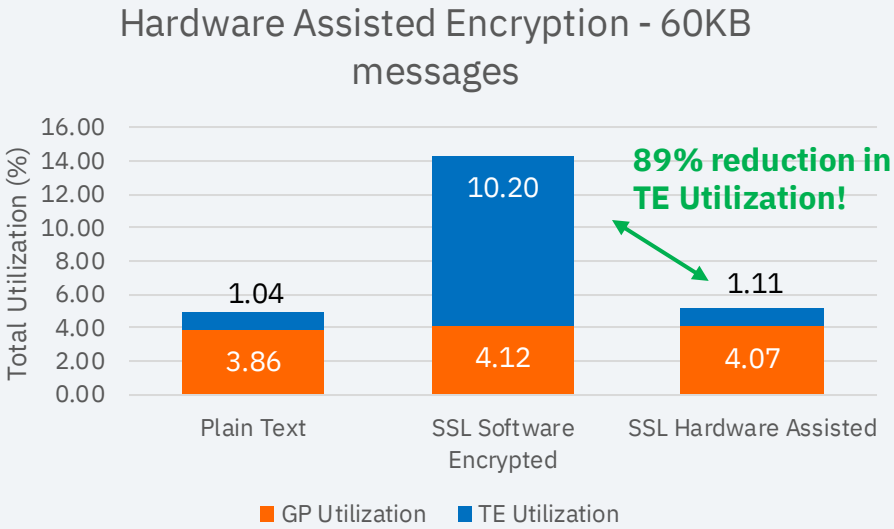
After

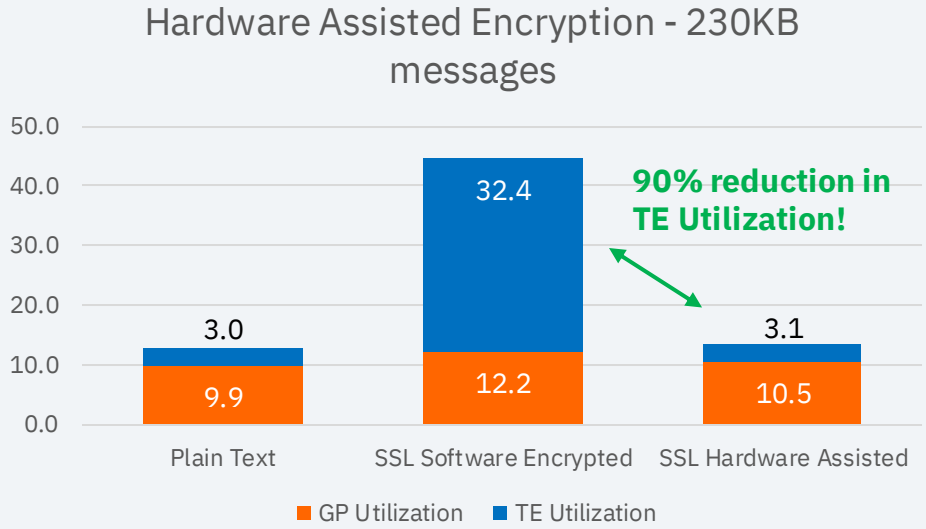
Elapsed Seconds Distribution	Average Response (mills)
0 - 30	.55
30 - 60	.25
60 - 90	.20

Encryption Performance Updates - PJ46547

Utilization numbers normalized to 300 msg/sec

With PJ46547, Java can now take advantage of on-chip encryption capabilities, resulting in significant reduction in TE utilization for encrypted communications

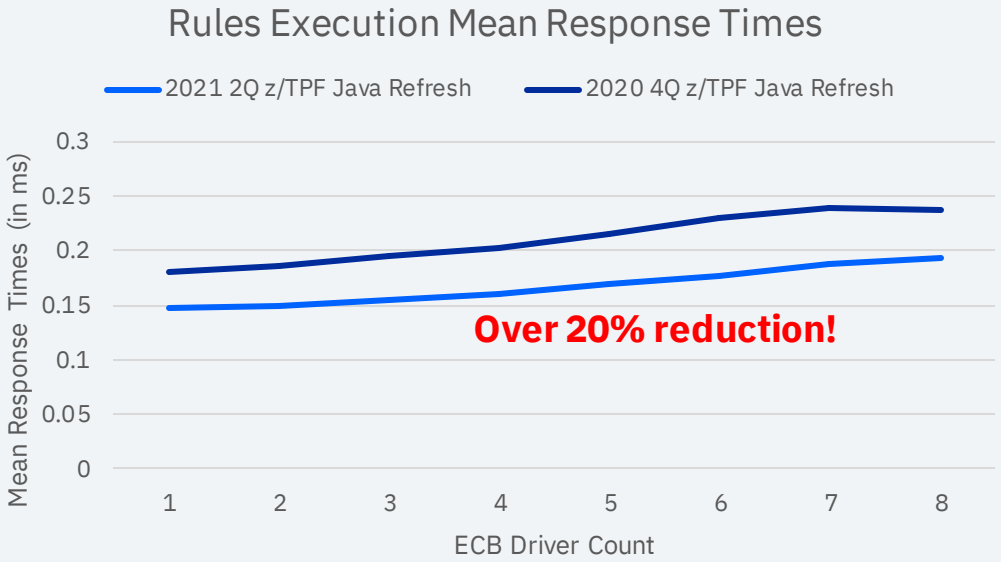




4 I-streams, z/15, 2 JVM JAM Running Kafka Using SHA256 with RSA (2048 key length)

Garbage Collection Performance Improvements - PJ46547

With PJ46547, Java garbage collection improvements reduce garbage collection overhead, resulting in better average response times for service calls and shorter pause times during collection.



10 I-streams, z/15, 2 JVM JAM Running a Rules Engine (Steady State)

Other Java Enhancements in 2021

- ✓ New JVM User exits (**PJ46432**)

  Description: UJVM user exit contains JVM startup, JVM Thread startup, and Java Dump complete functions

  Benefit: Allows for customization of JVM behavior at the system scope. For example, after a Java system dump is written out to a MFS mounted file system (for performance) then custom code could be added to FTP the dump off of z/TPF.

- ✓ Allow time-sliced ECBs to run as low priority (**PJ46547**)

  Description: Use tpf_easetc() function with TPF_EASETC_LOWPRIORITY parameter to mark an ECB as low priority

  Benefit: Allows for specifying a different service level for different JVMs. For example, marking the z/TPF Mail Service JAM as low priority as compared to a traditional transactional work.

Other Java Enhancements in 2021 (continued)

- ✓ New z/TPF Java Property allows JVM scoped MAXMMAP setting (**PJ46547**)

  Description: A new Java property can be specified on the Java Command Line for a specific JAM to control the 64-bit 1MB frame limit for the MMAP region (e.g., -Dcom.ibm.tpf.maxmmap=400)

  Benefit: Can provide savings in Java memory requirements now that a separate value can be enforced in addition to the system wide MAXMMAP setting.

  https://public.dhe.ibm.com/software/htp/tpf/tpfug/tgs21/TPFUG_2021_APPS_Java_Performance_Enhancements.pdf

- ✓ JAM User Exit extended (**PJ46590**)

  Description: UJAM now allows customization of Java Command Line options at the JVM scope.

  Benefit: Lifts restrictions for some Java applications that require JVM unique configurations when running in a JAM. Additionally allows for specifying options based on processor ID or sub-system ID. Helpful for network properties that change value based on processor (e.g., IP addresses).

OpenJDK Migration

- Starting in 2Q2022, IBM will begin transition to the IBM Semeru Runtime Certified Edition based on OpenJDK
- Positions IBM and the z/TPF platform for long term Java support
    - Bug fixes
    - New Features
    - Performance Improvements
    - Security updates

OpenJDK Migration Strategy

- Delivering OpenJDK and IBM Java 8 simultaneously for a short period of time (1 year minimum)
- JAMs can be configured to run as Java 8 or OpenJDK
- Not required to move applications to OpenJDK immediately, allows orderly transition
- IBM lab will be updating provided applications to be compatible with Java11.

Java Performance Enhancements with OpenJDK

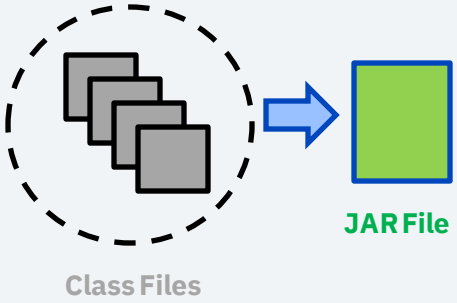
1. Recoverable JIMAGE File Format Support (OpenJDK)
2. Recoverable Share Classes Support (OpenJDK)
3. Hardware-Assisted Pause-less Support (OpenJDK)

JIMAGE File Format

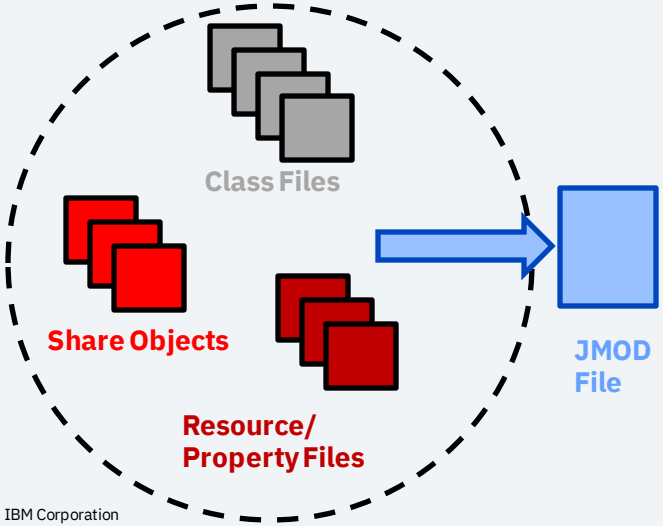
Java Platform Module System (JPMS)

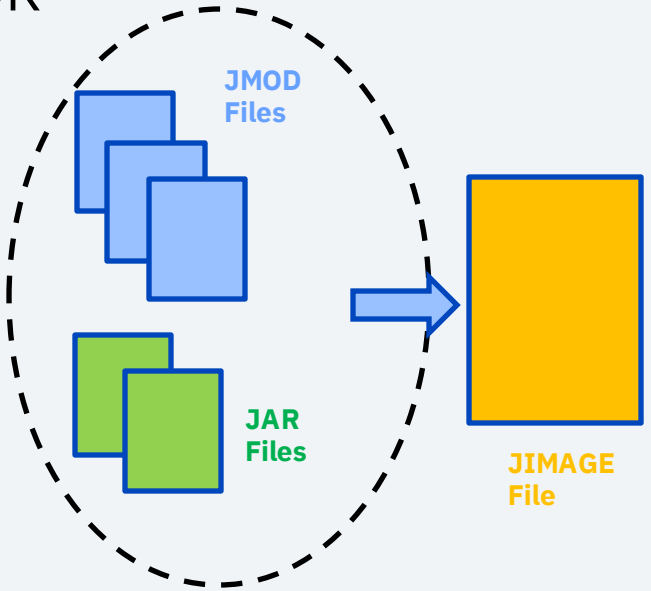
File Formats
Class
JAR
JMOD (Used for Compiles and Links Only)
JIMAGE

Java 8

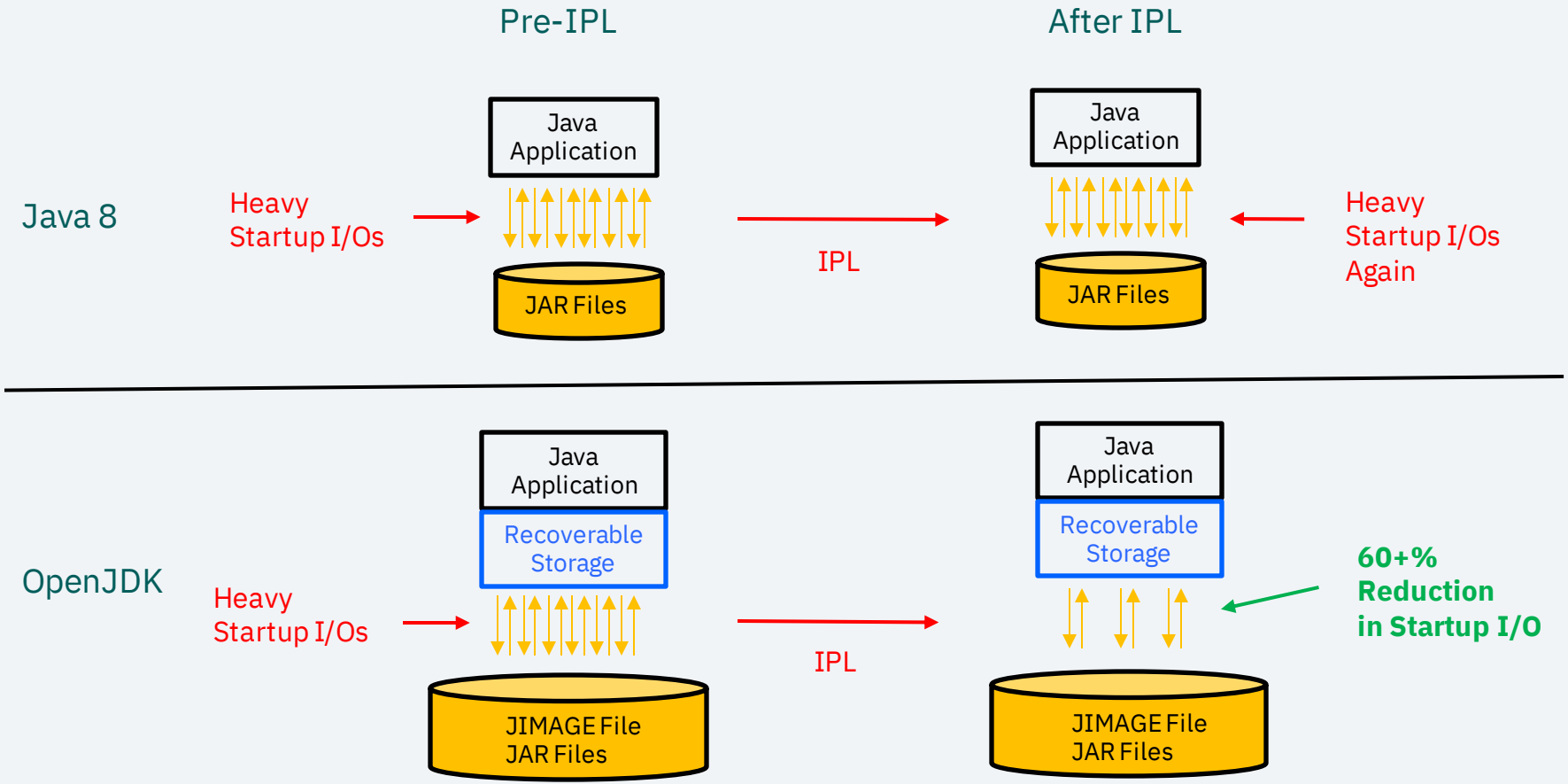


OpenJDK





JIMAGE File Performance Improvements



Shared Class Cache – Sharing Data between Java Applications

- Provides the capability of subsequent JVMs to startup faster
- Cache holds loaded classes, AOT compiled code, JAR file indexes for quick lookup, other data
- AOT compiled code is copied into local memory from the cache.
- JIT compiled code resides in local memory and is **not** shared

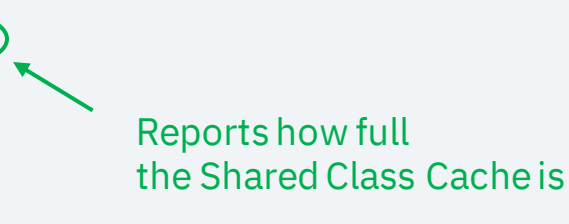
For more information see:

https://blog.openj9.org/2018/10/10/intro-to-ahead-of-time-compilation/

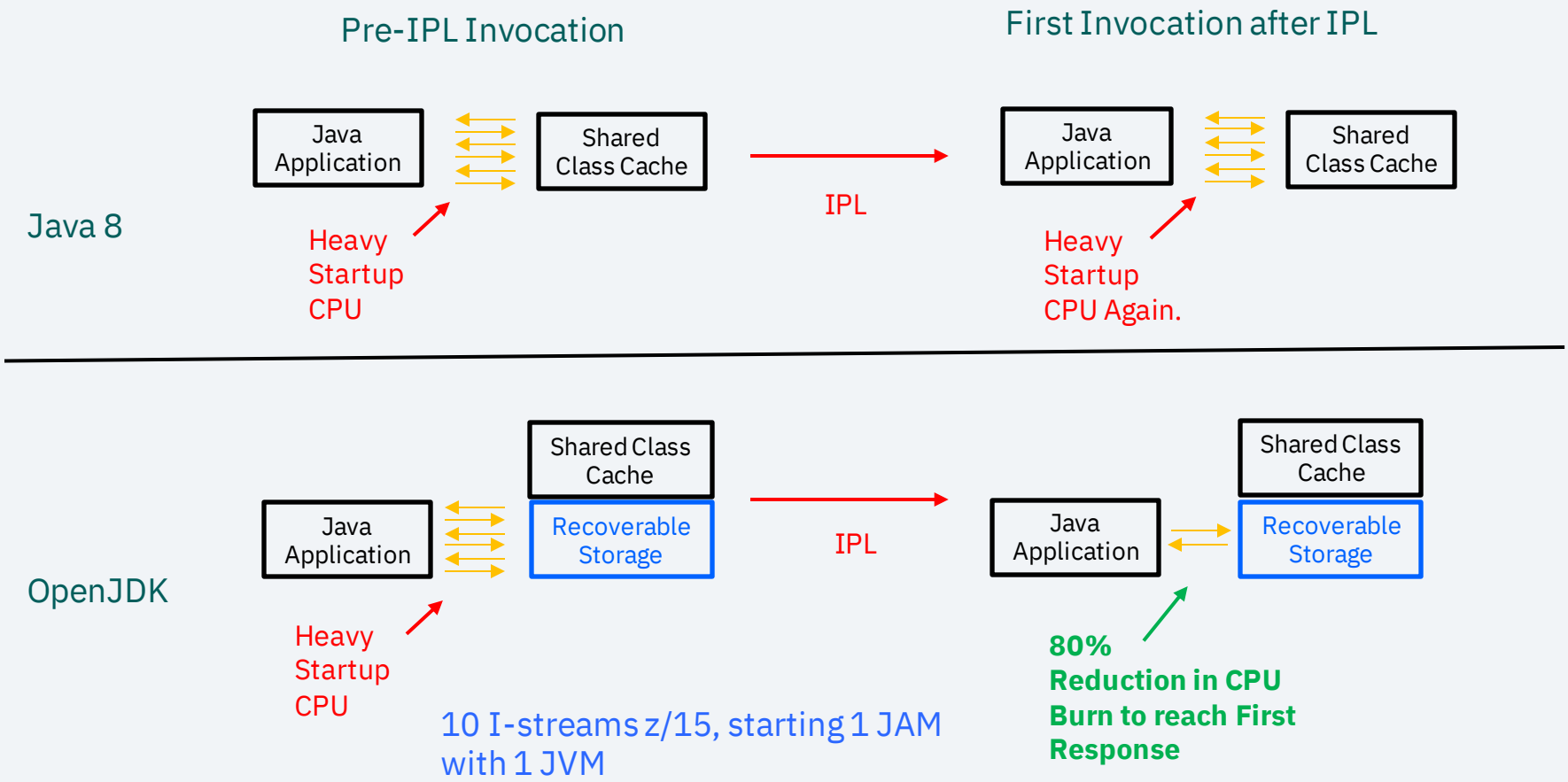
https://www.eclipse.org/openj9/docs/shrc/

Shared Class Cache – How to use Share Class Cache

- Applications create or attach to the SCC with a Java command line option:
  - *–Xshareclasses:name=MyCache*
- Administrative Commands are available to manage all the Java SCCs.
  - *–Xshareclasses:listAllCaches*
  - *–Xshareclasses:name=MyCache,printStats*
  - *–Xshareclasses:name=MyCache,destroy*
- SCC is non-persistent for z/TPF with Java 8



Shared Class Cache Performance Improvements

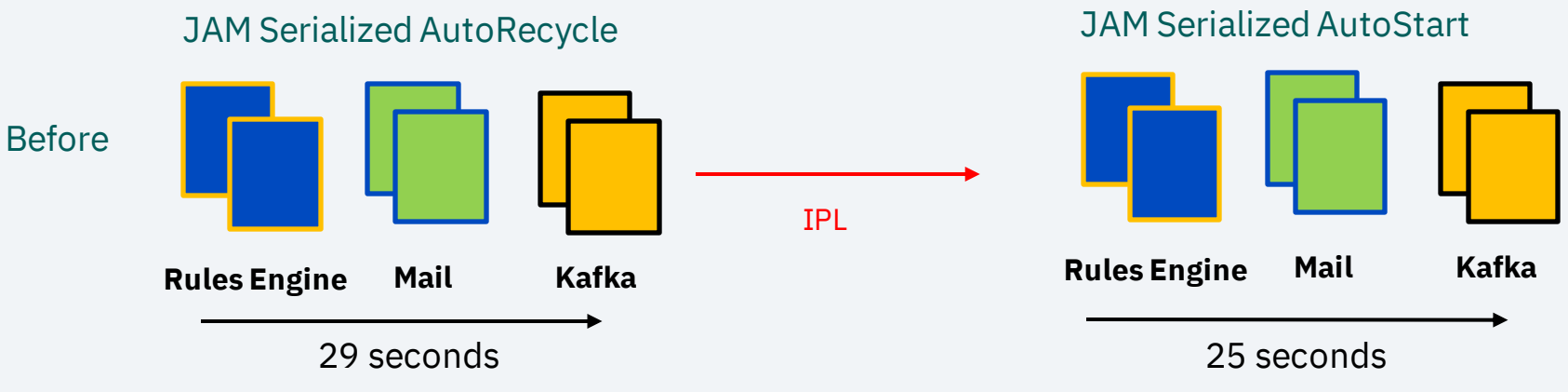


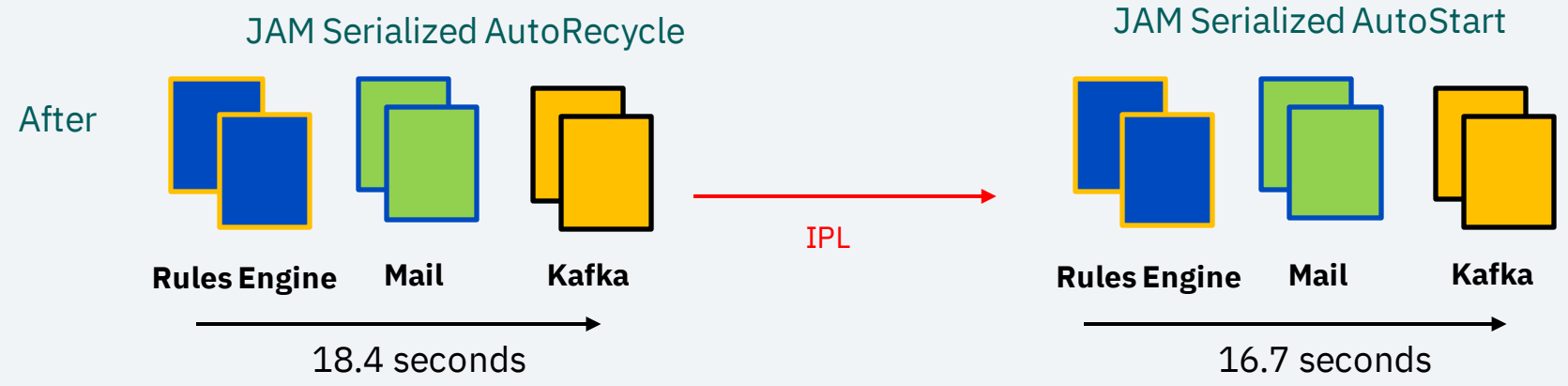
JAM Class Cache Strategy Change

- With initial release, JAM support allocated a separate class cache for each activation for each JAM
- Starting with OpenJDK support, this behavior will change
- OpenJDK adds support for “layering” shared class caches where each JAM can share a base cache while also having specialized caches per JAM

JAM Class Cache Strategy Change

10 I-streams z/15, starting 3 JAMs with 2 JVMs each

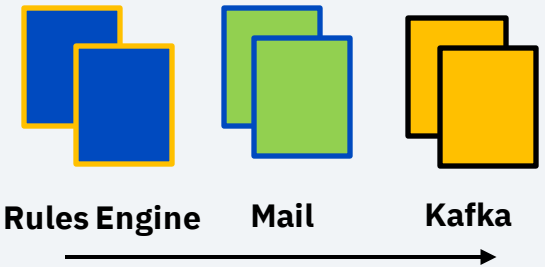




Future OpenJDK Concurrent Start

10 I-streams z/15, starting 3 JAMs with 2 JVMs each

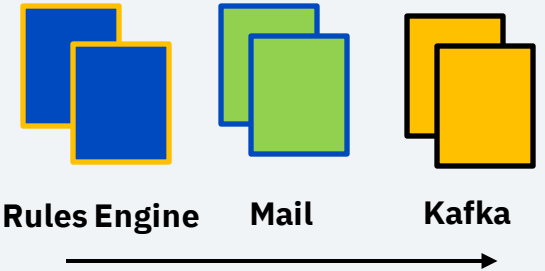
JAM Serialized AutoStart (no cache)



18.4 seconds

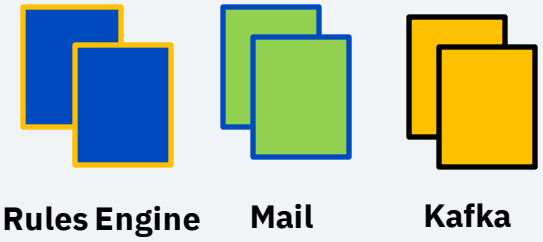
IPL

JAM Serialized AutoStart with successfully recovered system heap



16.7 seconds

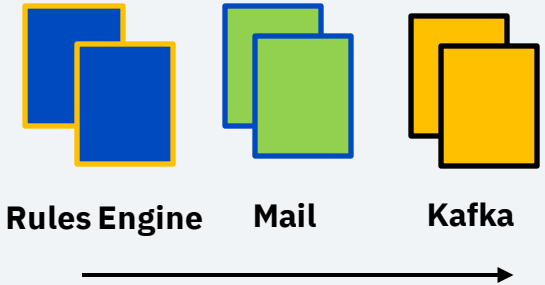
JAM Concurrent AutoStart (no cache)



12 seconds

IPL

JAM Concurrent AutoStart with successfully recovered system heap

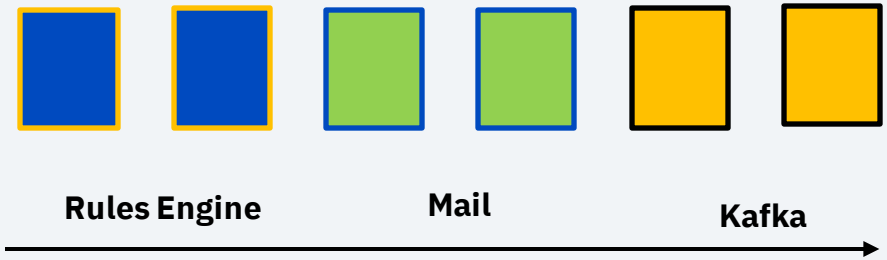


6.2 seconds

Future OpenJDK Concurrent AutoStart

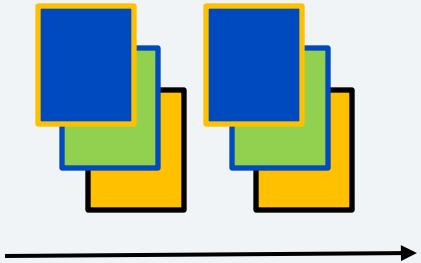
10 I-streams z/15, starting 3 JAMs with 2 JVMs each

JAM Current AutoStart



12 seconds until work complete, but access to “later” JAMS dependent on start of ALL preceding JAMs

JAM Concurrent AutoStart with successfully recovered system heap



6.2 seconds until work complete, but “faster” JAMs now available sooner without dependency on order

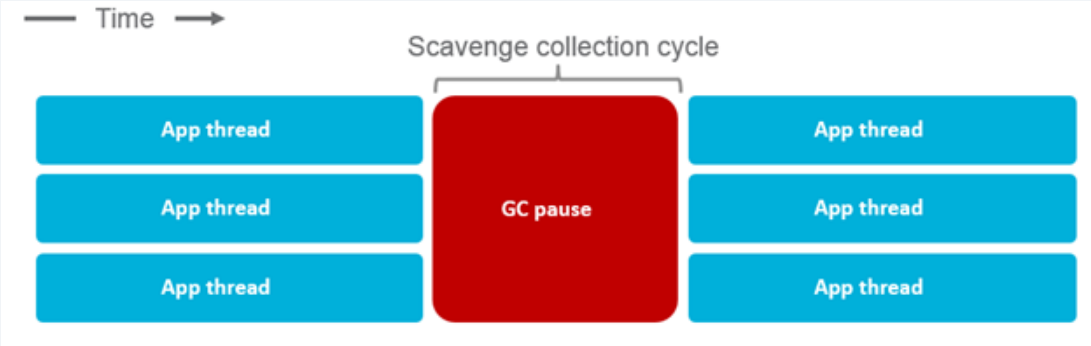
Hardware accelerated pauseless GC

- Potential future inclusion for OpenJDK
- Takes advantage of hardware assist (Guarded Storage facility) for memory references during garbage collection
- Reduces impact of garbage collection on application scalability by reducing the scavenge lock time

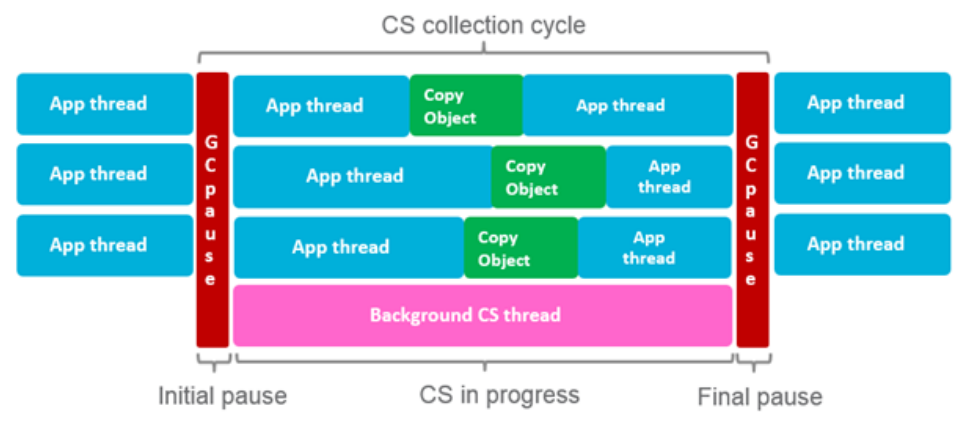
Hardware accelerated pauseless GC

https://blog.openj9.org/2019/03/25/concurrent-scavenge-garbage-collection-policy/

Before



After



Thank you

- Questions, comments, or issues related to Java please e-mail dgritter@us.ibm.com