Alibaba Cloud | Worldwide Cloud Services Partner

Alibaba Dragonwell JDK: Towards a Java Runtime for Cloud Computing

Xiaoming Gu
Alibaba JVM Team

Alibaba Infrastructure

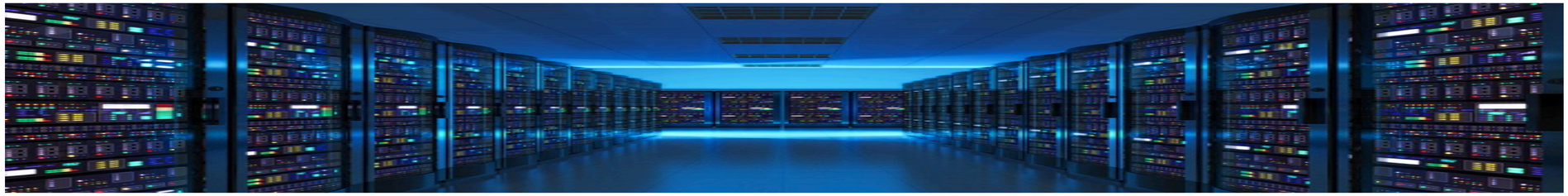
Alibaba Cloud | Worldwide Cloud Services Partner

淘宝网 Taobao.com | 天猫 TMALL.COM | CAINIAO 菜鸟 | 盒马 | ANT FINANCIAL | Alibaba Cloud

Database / Storage / Middleware / Computing Platform

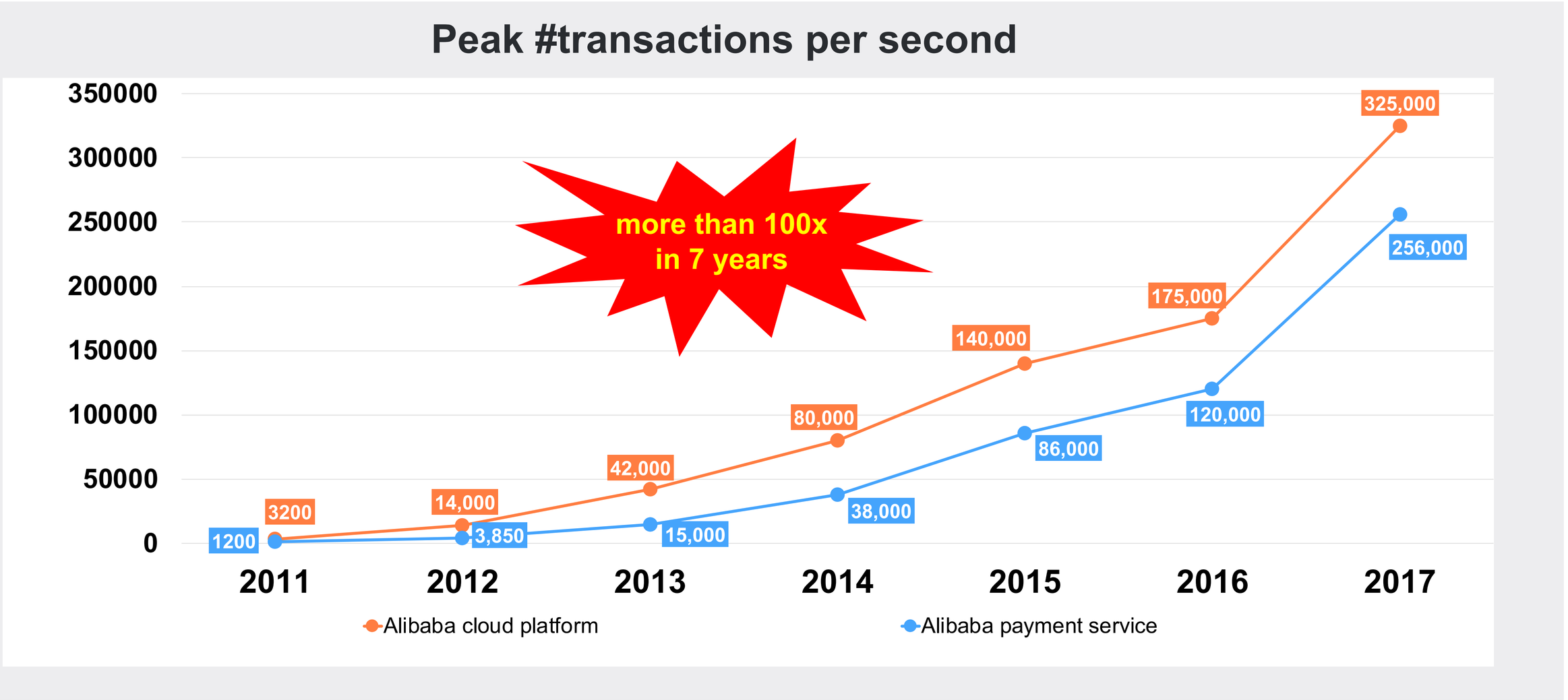
Resource Scheduling / Cluster Management / Container

System Software (OS / JVM / Virtualization)



Singles' Day Shopping Festival

Alibaba Cloud | Worldwide Cloud Services Partner



Alibaba Dragonwell

Alibaba Cloud | Worldwide Cloud Services Partner

- A customized downstream of OpenJDK with free LTS
- https://github.com/alibaba/dragonwell8
  - Preview now and GA soon
  - Will be the recommended JDK on Alibaba Cloud
  - Plan to update in every 3 months

Key Customizations in Alibaba Dragonwell JDK

Alibaba Cloud | Worldwide Cloud Services Partner

- Java Flight Recorder (JFR)
  - Low-overhead profiling framework
  - Backported JFR from OpenJDK 11 to Alibaba Dragonwell 8
  - In progress pushing back to OpenJDK 8
- JWarmUp
  - Reduce startup time by reusing Just-in-Time (JIT) compilation info from a previous run
  - In progress pushing back to OpenJDK

AppAOT

- A challenge in the cloud
  - CPU utilization is high during JVM startup
  - Caused by excessive JIT compilations

AppAOT

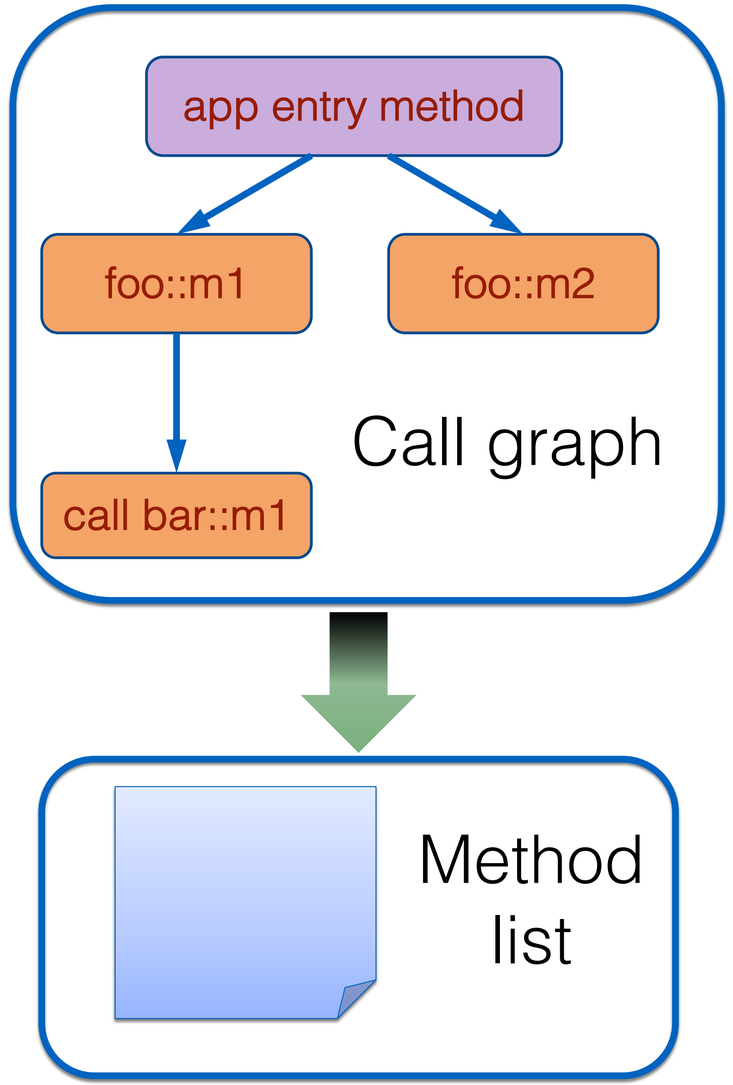
- Ahead-of-Time (AOT) compilation
  - AOT in OpenJDK
    - *jaotc --output libHelloWorld.so HelloWorld.class*
    - *java -XX:AOTLibrary=./libHelloWorld.so HelloWorld*
  - CPU utilization by JIT compilations saved
  - Limitation: AOT code loaded when JVM starts

AppAOT

- Enhanced from AOT: dynamically load/unload AOT code with the support of custom class loaders
- Java API
  - *AppAOTController.loadAOTLibraryForLoader(ClassLoader loader, String library)*
  - *AppAOTController.unloadAOTLibraryForLoader(ClassLoader loader)*

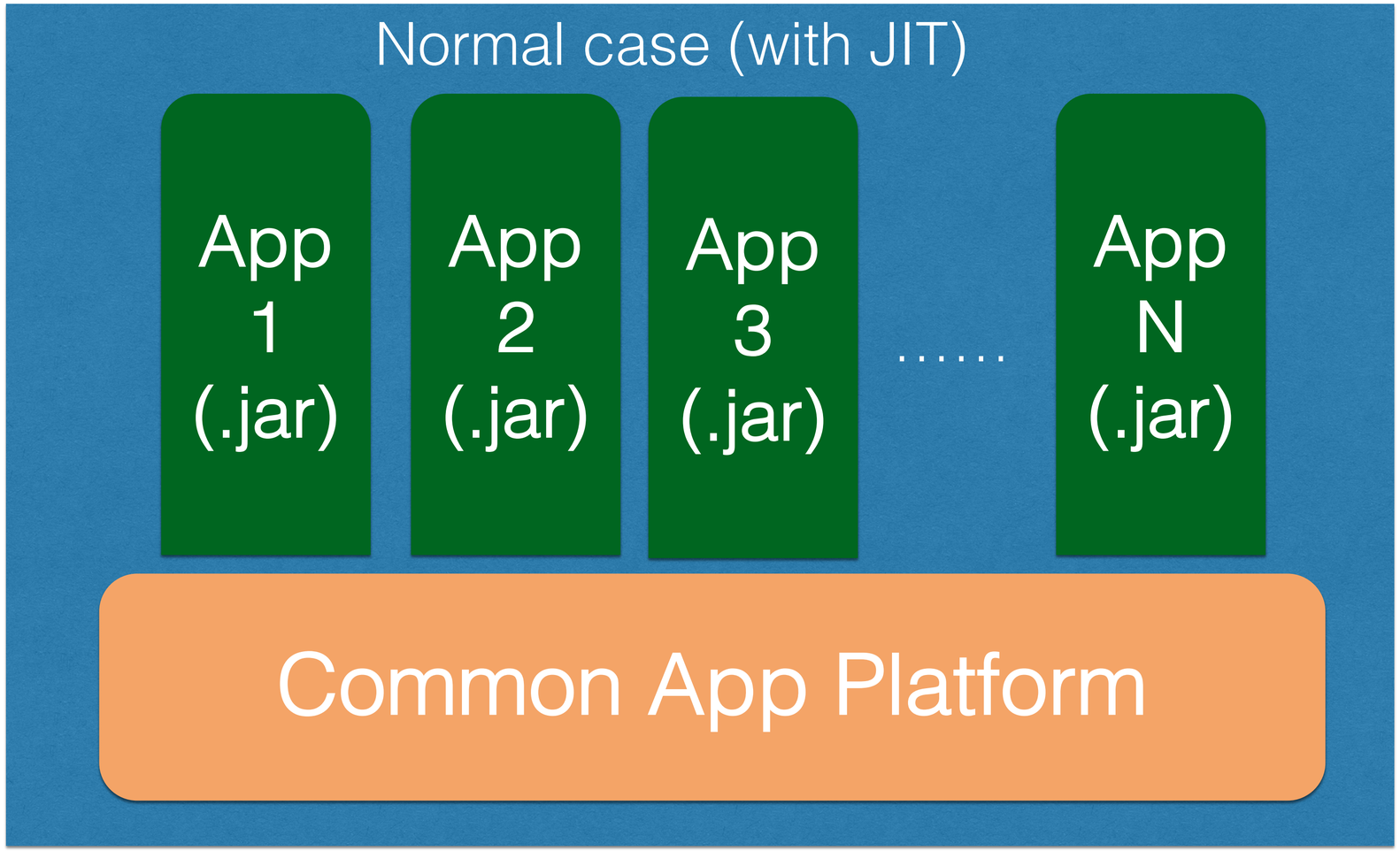
AppAOT

- Reduce AOT code size
  - Use static analysis to construct call graph
  - Generate compilation method list from call graph with unreachable methods removed
  - Do AOT compilations for methods on the compilation method list only
  - Results from an example app
    - 50% reduction on code size
    - 90% of actually executed methods covered



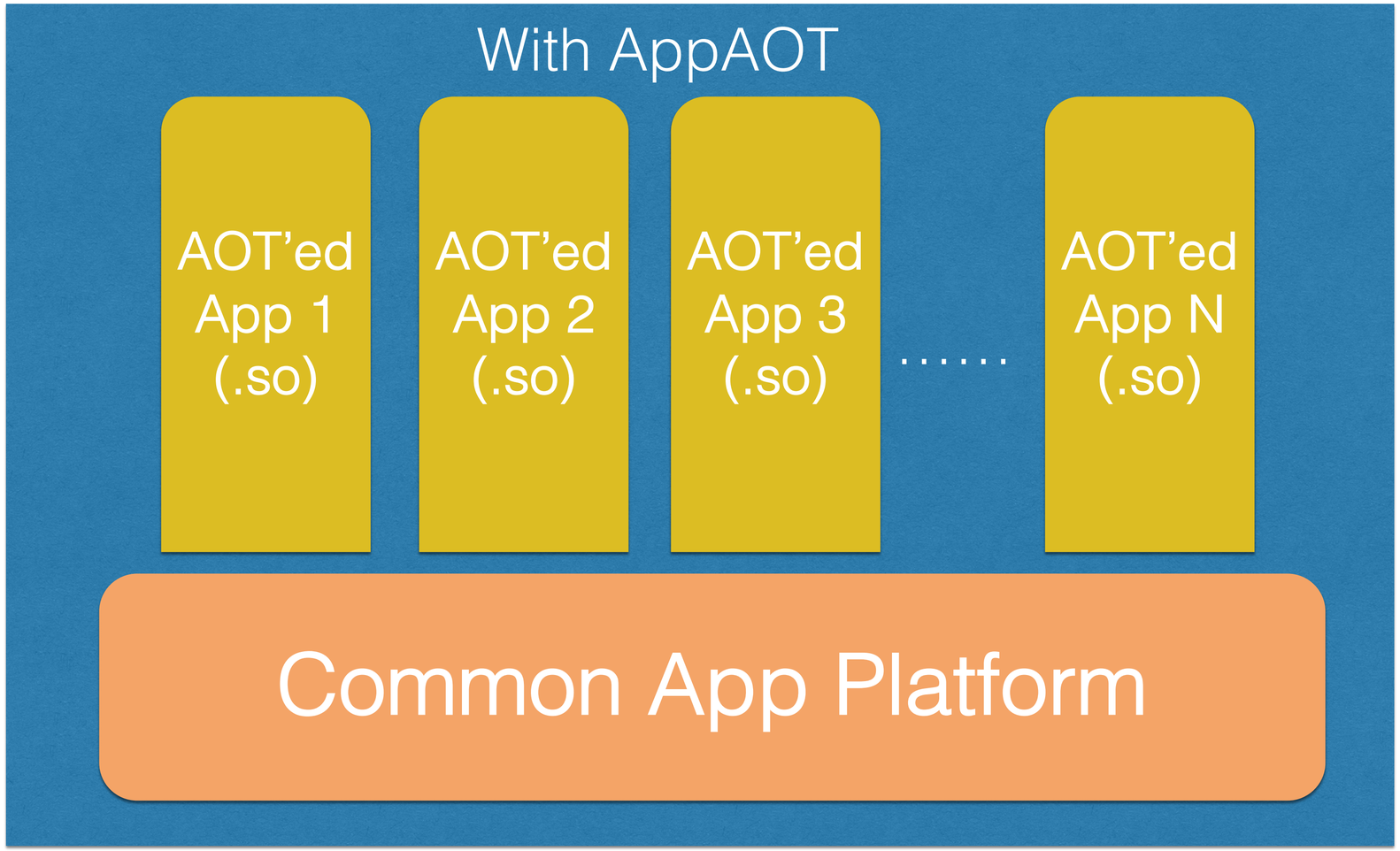
AppAOT

- Use case



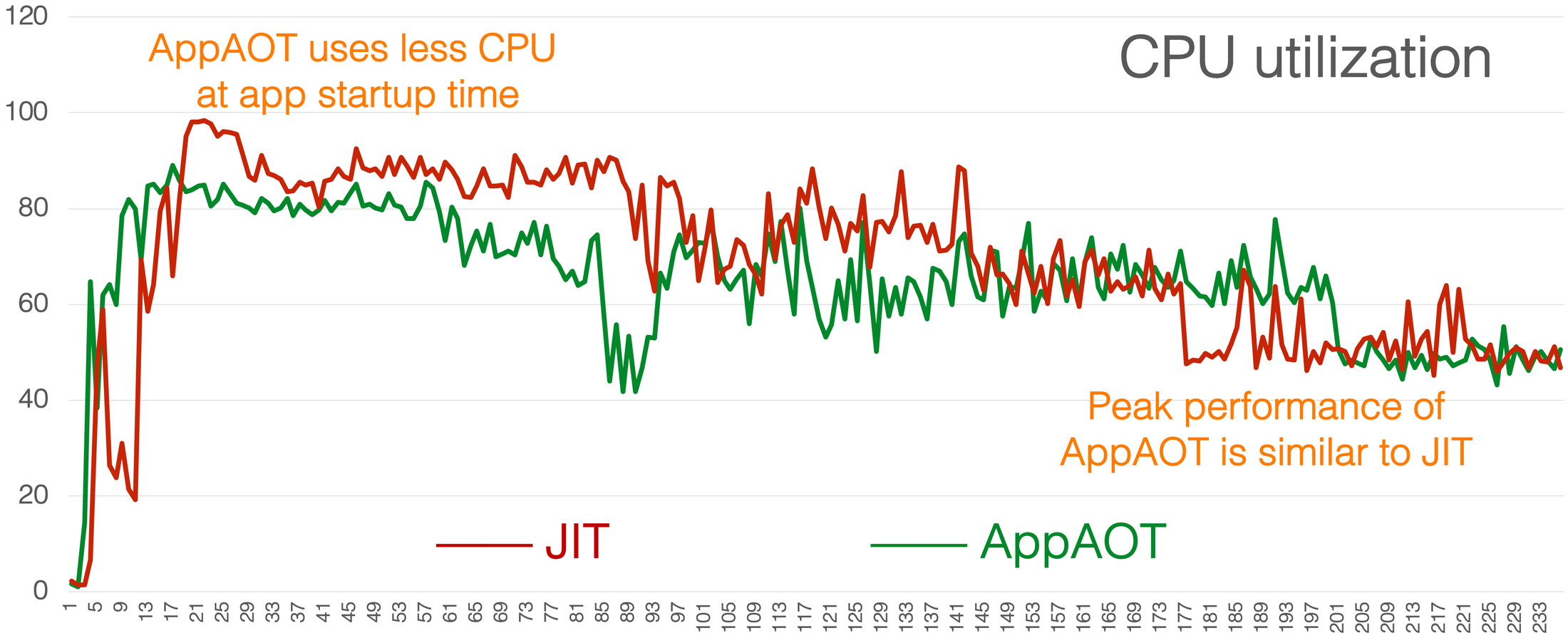
AppAOT

- Use case



AppAOT

- Results



Elastic Heap

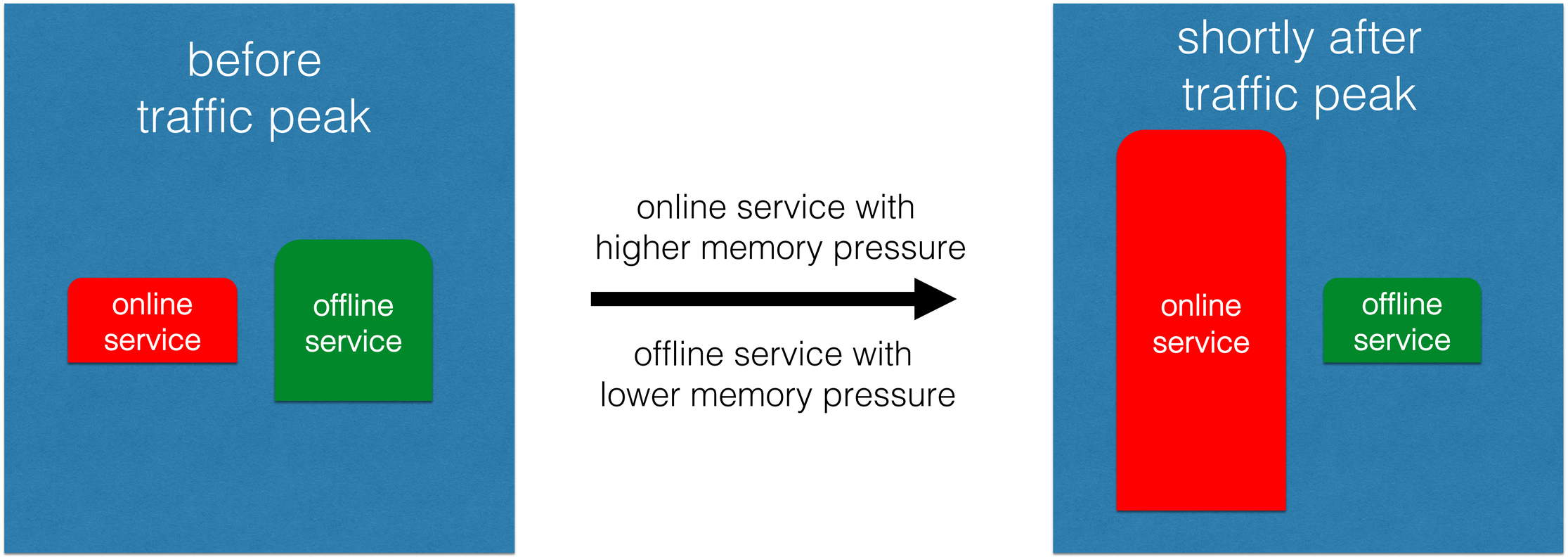
- Multiple Java applications running on the same host together
  - Memory is a shared resource
  - Memory consumption changes along running
  - Dynamically resize heap on demand

Elastic Heap

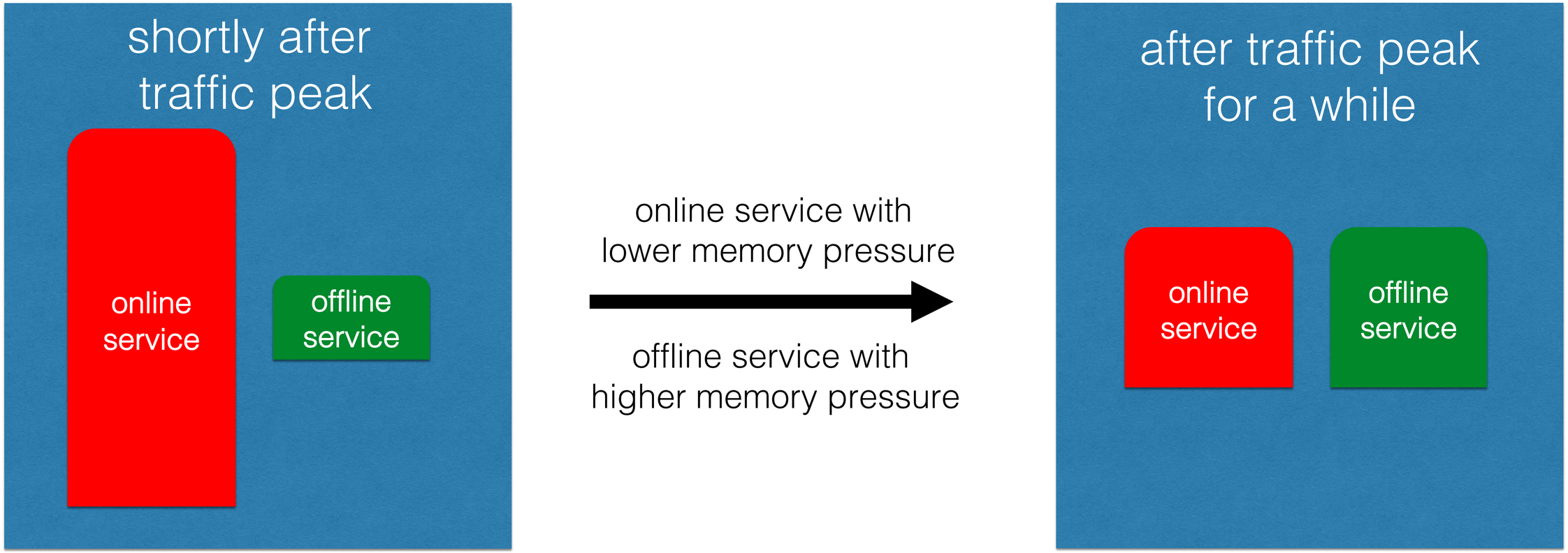
- Garbage collection (GC)
  - Automatic memory management on heap
  - Reclaim the space occupied by dead objects
- Intuition
  - Increase heap size when GC happens more
  - Decrease heap size with GC happens less

Elastic Heap

- Use case

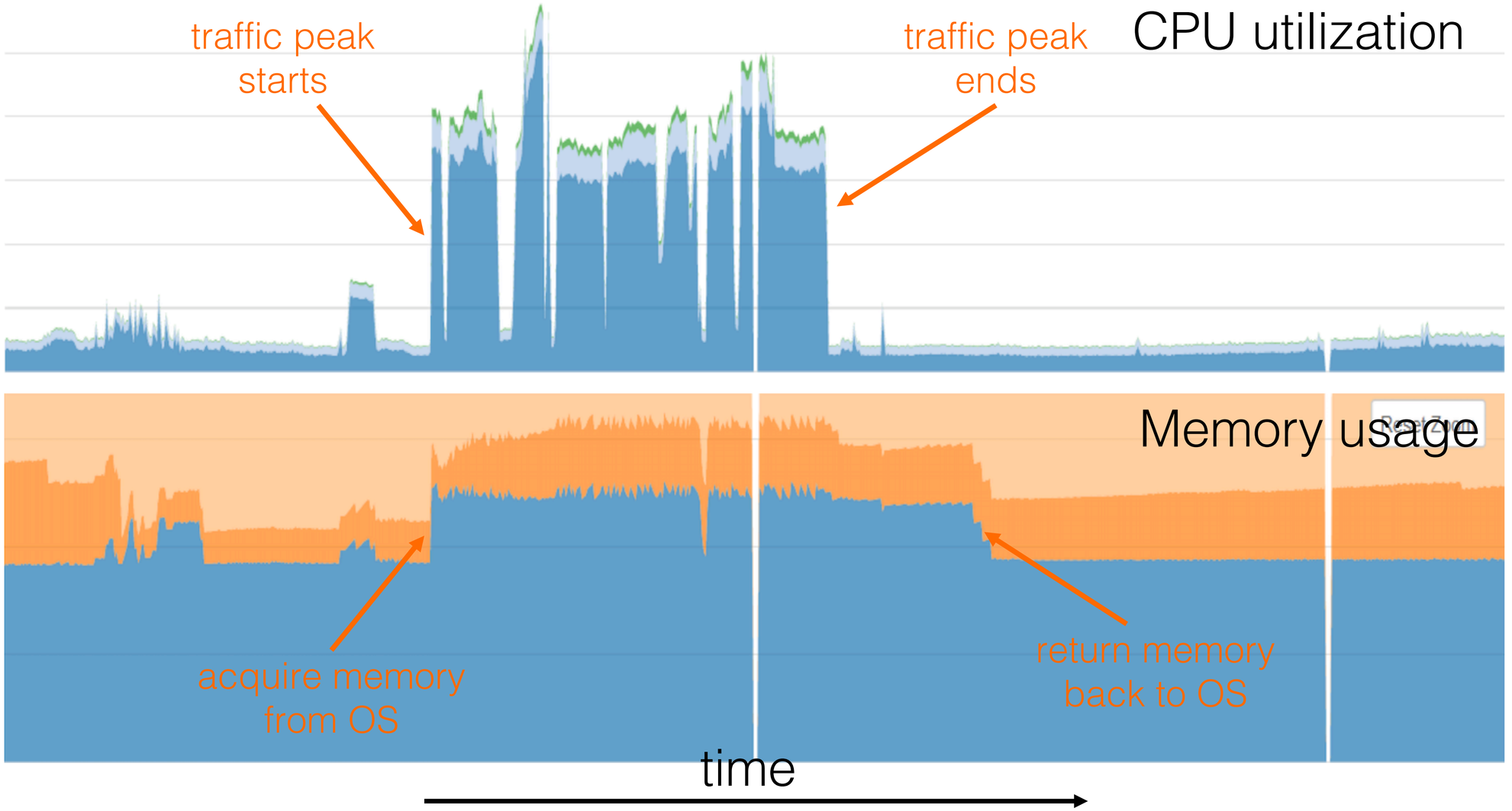


Elastic Heap



Elastic Heap

- Results from an online service



Elastic Heap

- Both OpenJDK and our solution are based on Garbage-First (G1) GC
- The solution in OpenJDK is for full GC and concurrent cycle only
- Young GC is covered in our solution
  - Heap resizing is more prompt because Young GC happens more frequently
  - Heap resizing is concurrent

Summary

- Challenges in the cloud
  - Excessive CPU utilization caused by JIT compilations
    - AppAOT
  - Better memory sharing across JVM instances on the same host
    - Elastic heap

Alibaba Cloud | Worldwide Cloud Services Partner

Q & A