

**ORACLE®**

# JRokit Product Family

## JROCKIT MISSION CONTROL

- Complete insight into application & JVM behavior
- Zero performance overhead in production environments
- No application modification or configuration required

## JROCKIT REAL TIME

- High-performance real-time solution for standard Java
- Industry leading **Deterministic Garbage Collector**
- Millisecond response times with “five nines” guarantee
- **Improve application performance & latency** with unique tooling

## JROCKIT VIRTUAL EDITION

- Fly-weight Java container for virtualized environments
- Improve datacenter efficiency - do more with less
- Simpler and more powerful VM management
- **Scheduled for 2009**

## JROCKIT JVM

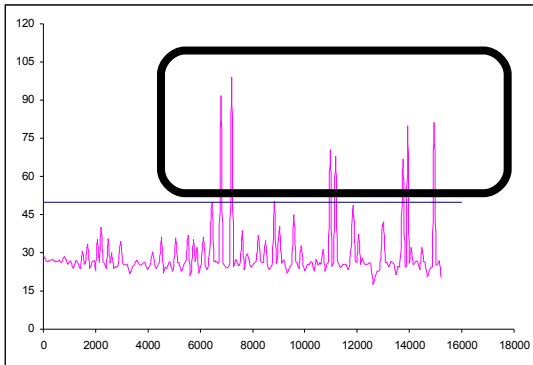
- World-class performance
- Powerful diagnostics
- Full support from Oracle

# JRokit Real Time

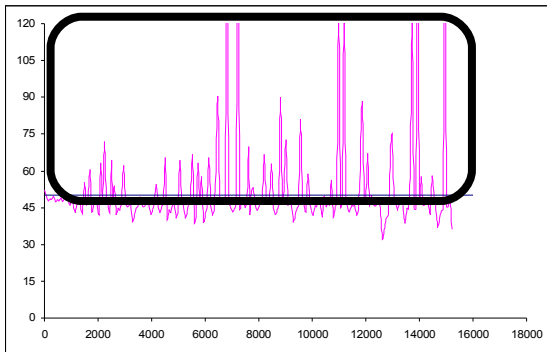
- Java SE engine with ‘soft’ real-time performance
  - Deterministic GC provides max pause time guarantees
    - “no pause should be longer than 5 ms”
    - Max latency = time to process transaction + max pause time
    - Decreases frequency and severity of latency spikes
  - Snap-in replacement for existing JVM, no code rewrite required!
- Unique RT tooling helps customer identify & remedy latency issues

# Benefits of Deterministic GC

## Traditional Java

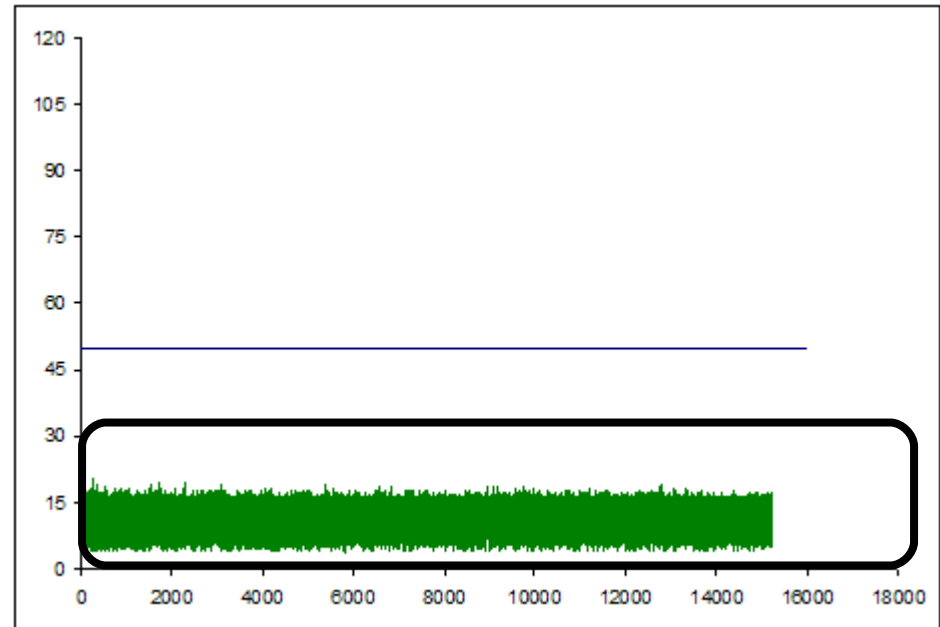


**During Low Load: GC spikes and occasional timeouts visible**



**During High Load: GC pauses can result in unacceptable response times**

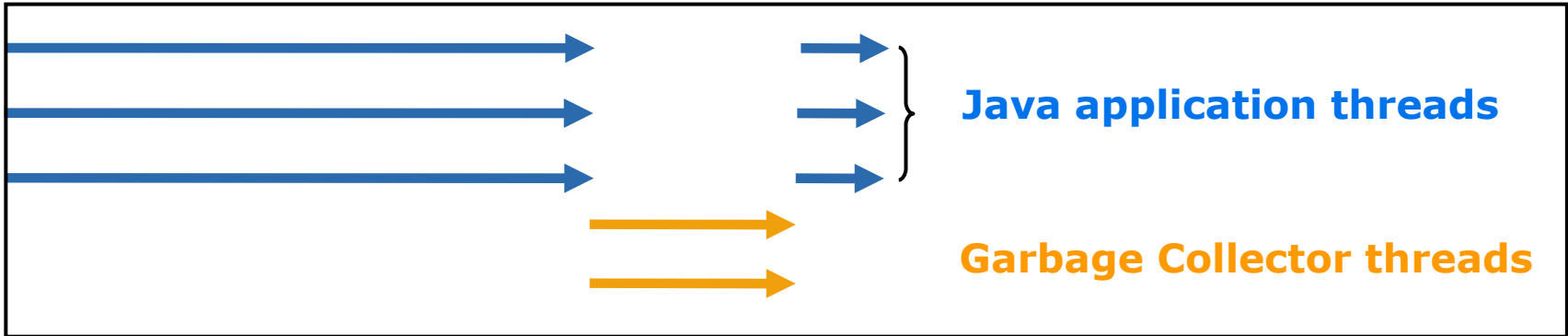
## JRockit Real Time



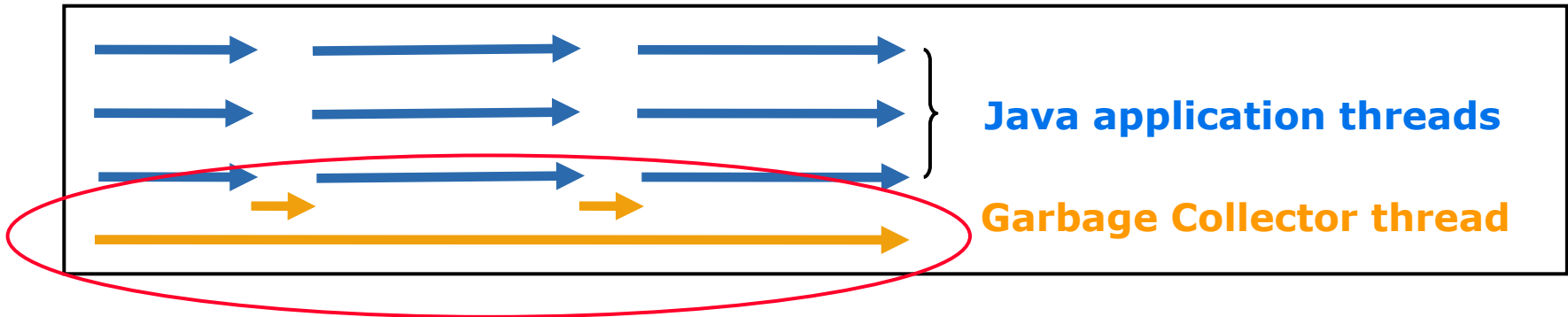
**JRRT Makes garbage collection *deterministic*.  
Allowing for the guarantee of SLAs.**

# Throughput and Latency Tradeoffs

- Parallel GC – Optimized to increase throughput



- Concurrent GC – Optimized to reduce pause time



Concurrent GC consumes additional resources

# JRokit Real Time

## Typical Opportunities

### 1. Slow response leads to lost revenue

- Trading Application: Respond too slow and you miss the deal
- Trader quote -- “Every millisecond delay means we lose money due to losing deals or increase arbitrage costs.”

### 2. Unpredictable response time leads to less control

- Pricing Engine: Slow response means inability to respond with a price for a securities instrument

### 3. SLA Violation leads to penalties

- Communication Service Provider – Customer SLA’s require immediate, predictable response
- LOB Owner -- “We have a stringent response time SLA to our customers. If we don’t meet it, we have to pay fines.”

### 4. New SLAs lead to Higher Revenues & Market Leadership

- Securities Trading Customers -- Ability to offer new SLAs could lead to new revenue streams
- Market-leading product, consistent offer, results in loyalty, revenues, & market leadership

### 5. Slow response invalidates use case

- RFID: Must be fast or savings by automating process are lost

### 6. Moving to Java saves money

- Port legacy C/C++ apps without worrying about pauses
- Data Center Manager quotes
  - “We have problems with long GC pauses in our Java application”
  - “We are going to build an application which requires millisecond response times, and want to do it in Java”



# Latency Analyzer Example

The screenshot displays the JRockit Mission Control 3.0.2 interface. The main window shows a 'Latency Graph' for the application 'wldf-lat-servlet.jra'. A callout box points to a significant spike in the graph, stating: **Time to process URL "/phonehome-web/dataservice" – 1.968 seconds**.

The 'Threads' panel shows the execution of various threads. A callout box points to a thread labeled '[ACTIVE] Execute...' which is highlighted in pink. A tooltip for this thread provides the following details:

- Servlet #563**
- Start: 12,656 s
- Duration: 1 s 968,08 ms
- Context URI: /phonehome-web/dataservice
- Parameters: limit=25# type=get# what=tracegroups# start=0#
- Stack Trace:**
  - at weblogic.diagnostics.instrumentation.action.LATAction.postProcess
  - at weblogic.diagnostics.instrumentation.rtsupport.InstrumentationSupport.post
  - at org.bea.phonehome.web.server.DataServlet.doPost(HttpServletReq
  - at javax.servlet.http.HttpServlet.service(HttpServletRequest, HttpSe
  - at javax.servlet.http.HttpServlet.service(HttpServletRequest, ServletRes
  - at weblogic.servlet.internal.StubSecurityHelper\$ServletServiceAction
  - at weblogic.servlet.internal.StubSecurityHelper.invokeServlet(Servlet
  - at weblogic.servlet.internal.ServletStubImpl.execute(ServletRequest,
  - at weblogic.servlet.internal.ServletStubImpl.execute(ServletRequest,
  - ...

The 'Properties' panel on the left shows a stack trace for the thread, with a callout box stating: **Looks like we're waiting for the database...** The stack trace includes the following classes and methods:

- org.postgresql.core.VisibleBufferedInputStream
- org.postgresql.core.PGStream.ReceiveChar
- org.postgresql.core.v3.QueryExecutorImpl
- org.postgresql.jdbc2.AbstractJdbc2Statement
- org.postgresql.jdbc2.AbstractJdbc2Statement
- org.postgresql.jdbc2.AbstractJdbc2Statement
- org.bea.phonehome.web.server.PostgresIn
- org.bea.phonehome.web.server.DataServle
- org.bea.phonehome.web.server.DataServle
- org.bea.phonehome.web.server.DataServle
- javax.servlet.http.HttpServlet.service(HttpS



# How do I try out?

- Download and install JRockit Real Time

1. Download from:

- [oracle.com/jrockit](http://oracle.com/jrockit) > download > Oracle JRockit Real Time 3.0

2. Install & start:

- `java -Xms1024m -Xmx1024m -Xgcprio:deterministic -Xpausetarget=10ms your.app`

# For More Information

search.oracle.com



or

**oracle.com/jrookit**

# Future Development Concepts

- JRMC - Continuous JRA (Flight recorder)
  - Provides a recording of what happened in the VM leading up to a problem
  - Compare to the black box in a plane
- “Pauseless GC” (Research)

**ORACLE®**