



**So, you think you want  
real-time!**

**Eric J. Bruno**

**[eric@ericbruno.com](mailto:eric@ericbruno.com)**

**Principal Engineer**

**Sun Microsystems, inc.**



# Agenda



- Background
- Real-time Perceptions
- Financial Companies
- Proof-of-concept Woes
- Recommendations
- More Information

*“What's new in the other sectors, I've been stuck here for 200 microseconds”*

*-Ram (from the movie “Tron”)*

## My Background

- Real-time transaction systems in C++
  - > Reuters Dealing 2000 and Dealing 3000
  - > Order routing
  - > Inter-bank credit systems
- “Real-enough”-time Java development
  - > News and quotes systems
  - > Investment banking systems
- Real-time Java work with Greg Bollella at Sun
- Book: *Real-Time Java Programming*, Prentice Hall

## Real-Time Perceptions

- *Fact versus Perception*
- Industry perceptions vary:
  - > Many understand: based on a temporal condition
  - > Too many think: really, really fast
  - > Some still think: dynamic delivery of data (without considering time or latency)
- Quick lecture always helpful
- The real problem in the financial space:
  - > Latency is important (and often critical)
  - > But, not ready to give up hard-fought throughput

## Financial Companies

- Sun had over 200 simultaneous POCs at one point
- Nasdaq and CBOE have been public about it
  - > JavaOne keynotes and presentations
- They usually measure success in two dimensions:
  - > Requests (or messages) per second
  - > Maximum latency per request
- These are related but often contradictory
  - > On *average*, they meet throughput goal
  - > But, averages cover up latency issues
    - Customer complaints map back to latency (often GC related)

## Financial Companies

- All latencies being squeezed out of systems
  - > General trend to co-locate customer systems within exchange data centers
  - > Faster computers, multi-core, faster networks, faster storage...
- Exposing latency in areas not see before
- Operating systems, frameworks, and Java VM being pointed at

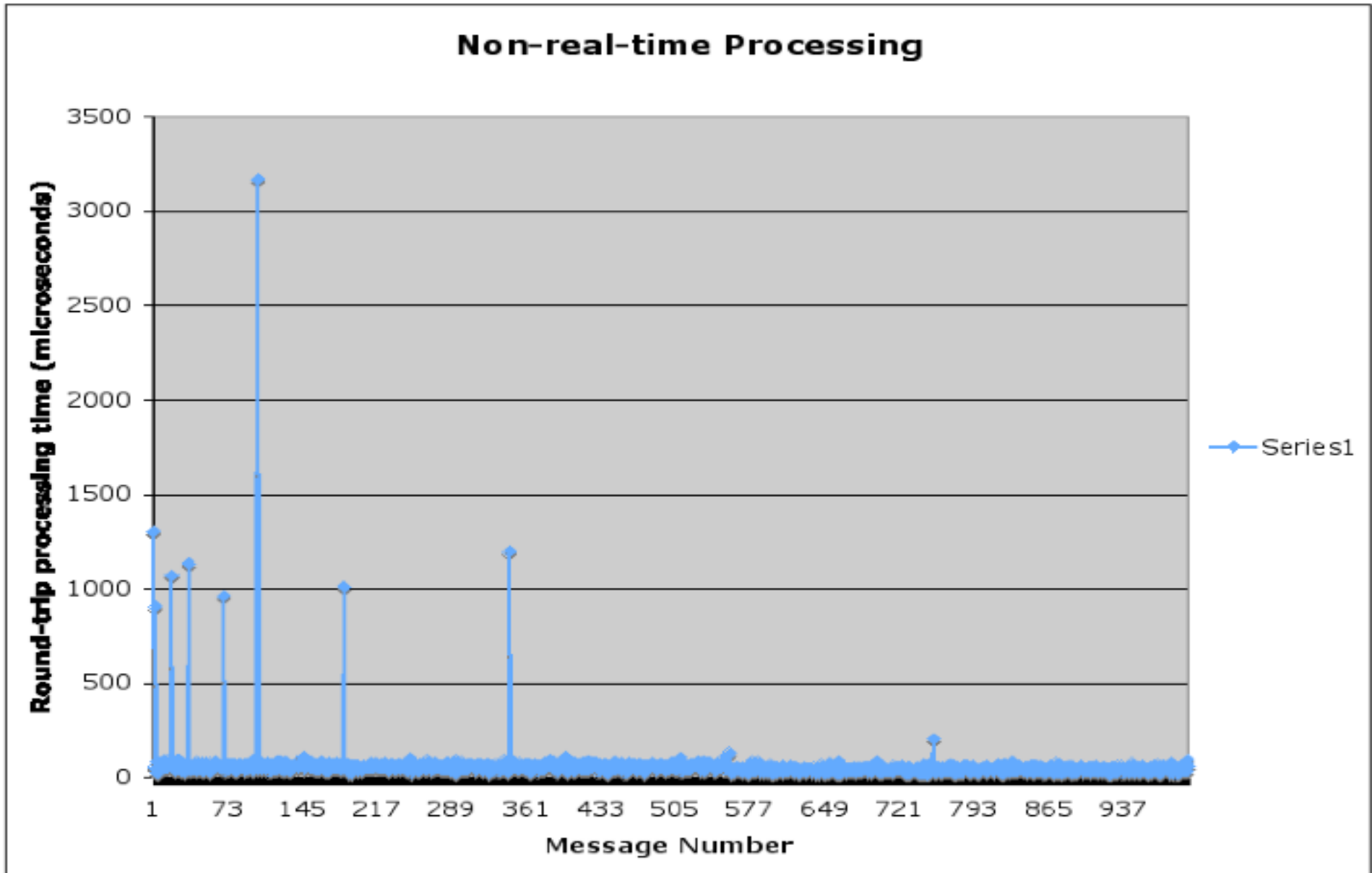
## Proof-of-concept Woes

- Many based on porting existing Java code
- Java RTS quickly equated with throughput loss
- Exposes application design issues
  - > Code tuned for general-purpose OS and Java SE
  - > i.e. locking, extreme multi-threading, fair scheduling
- Third-party libraries and frameworks
  - > Entire food-chain needs to support RTSJ
- Want results quickly - again with fast metaphor :)
- Most of RTSJ not desired, yet RTSJ requirements affect performance/throughput

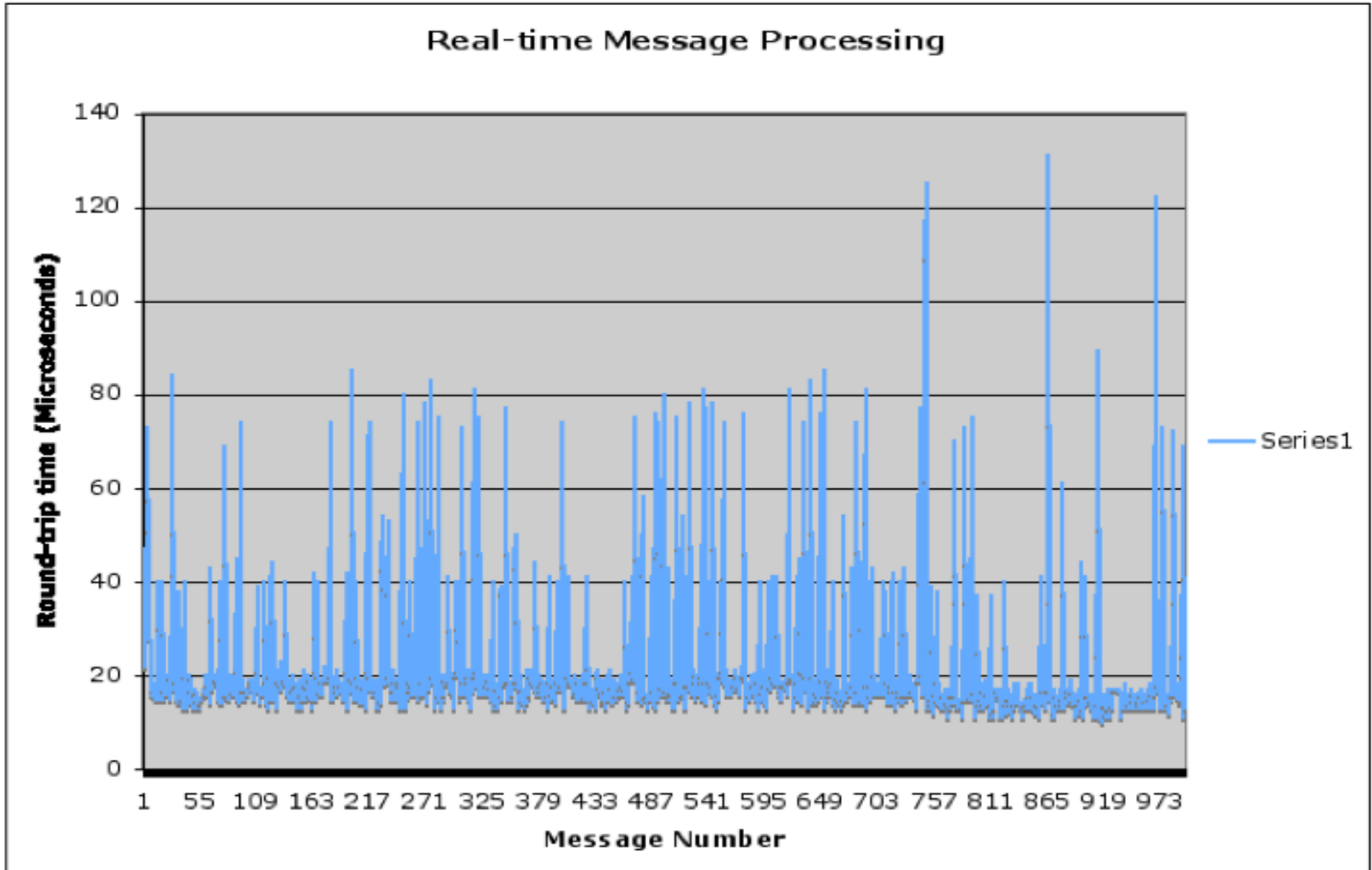
# POC Example 1

- Message processing app (customer requests)
- Results good
- However
  - > Only a prototype application
  - > Simulated their actual processing but did not use their code
  - > The real system was single threaded by design
    - This is a key point – no concurrency

# POC Example 1 – with Java SE...



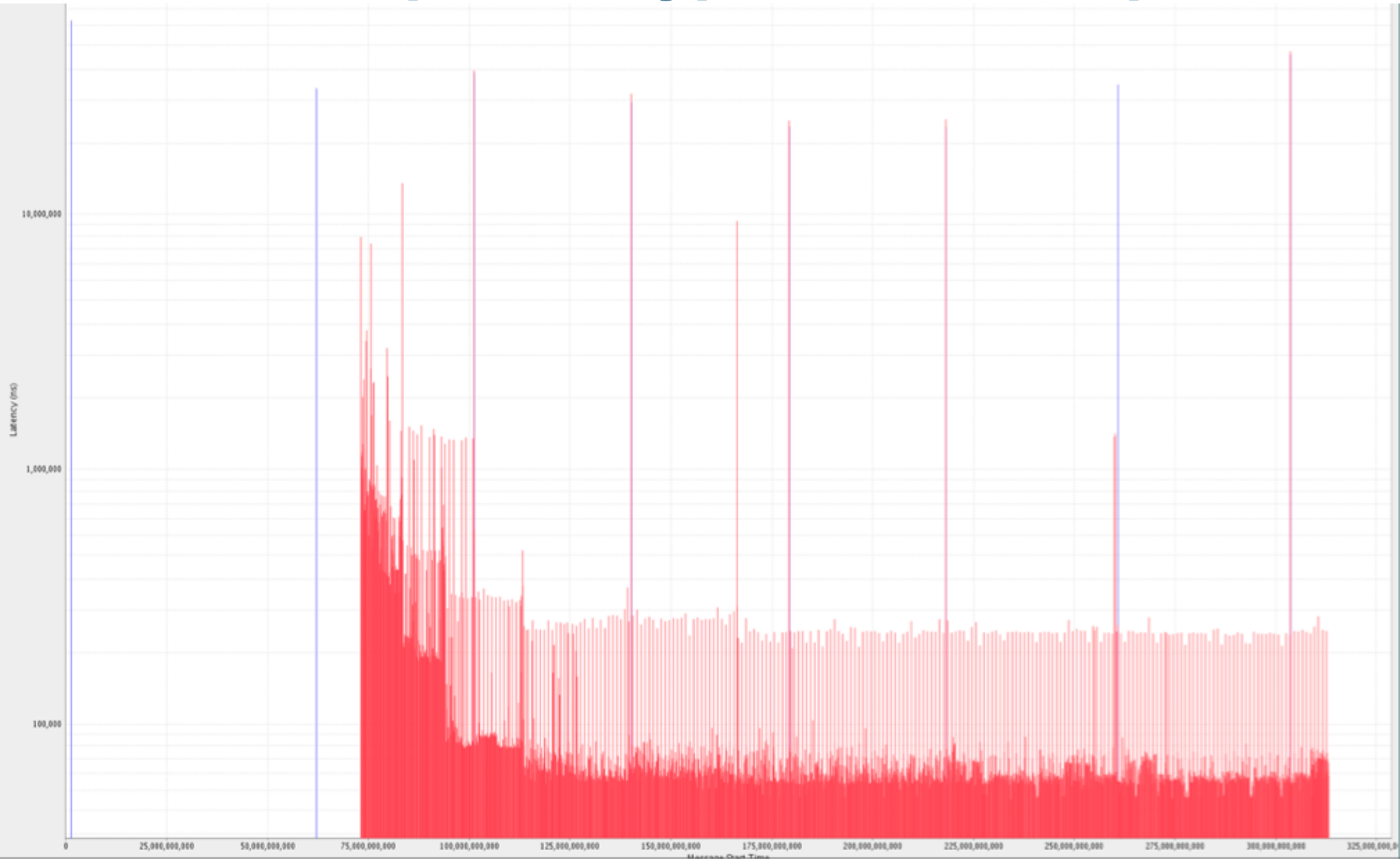
# POC Example 1 – w/JRTS (encouraging)



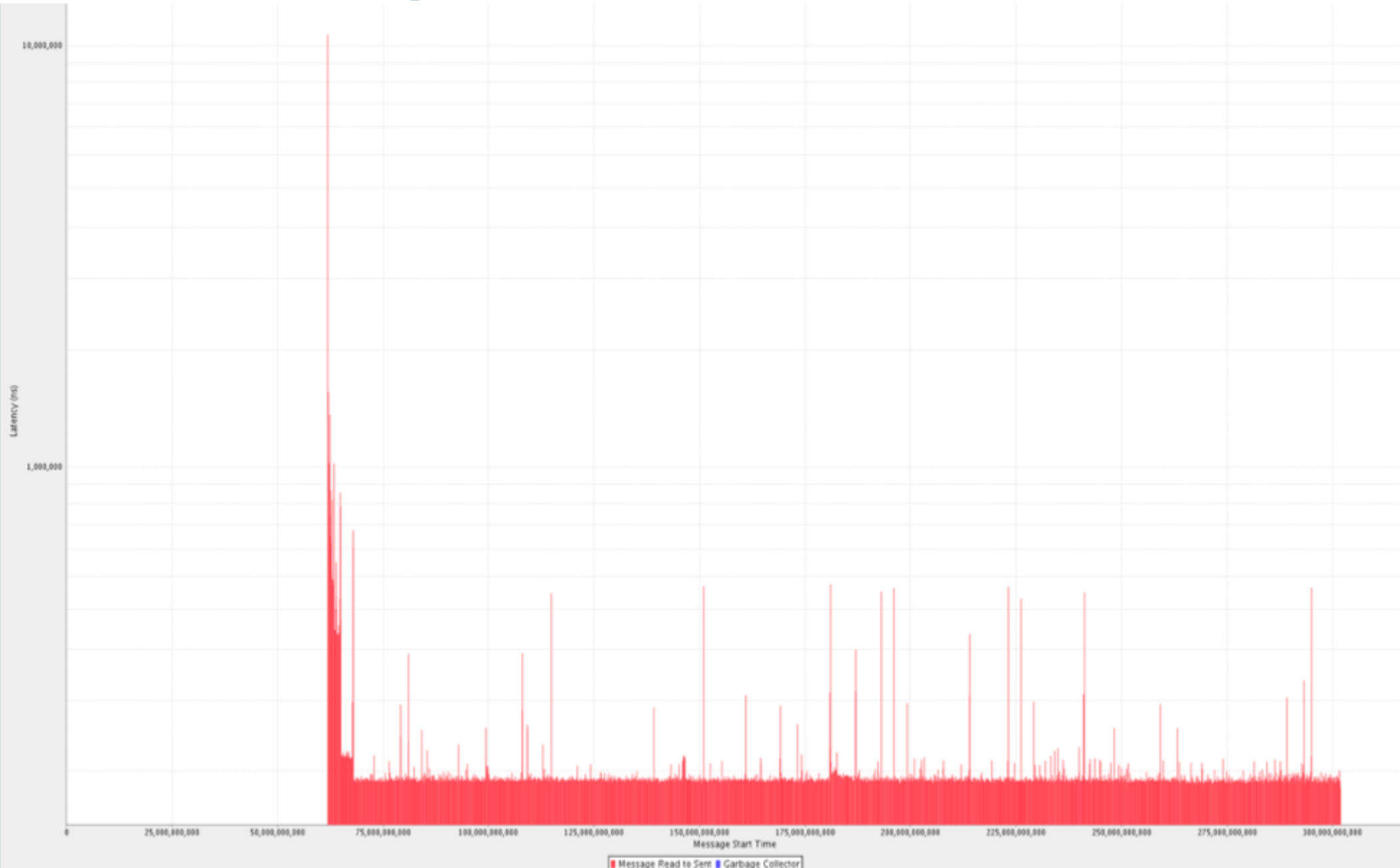
## POC Example 2

- Message processing application (customer requests)
- Throughput *and* latency very important
- Lots of threads with some contention
- Lots of IO
- GC-related latency was the issue
  - > i.e. GC pauses to 20 milliseconds

# POC Example 2 – Typical Java SE profile



# POC Example 2 – common JRTS results



# POC Example 2 – Increased Averages

- Average latency with Java SE ~100 microseconds
  - > Spikes to 20 milliseconds
- Average latency with Java RTS ~200 microseconds
  - > Spikes to 400 microseconds

# Exaple 2 – The Numbers (different test)

- With Java SE:

- With Java RTS:

**Average Latency: 228**

Median Latency: 196

Standard Deviation: 704

Minimum Latency: 131

Histogram of Latencies (Range values in microseconds):

[ Range ]	Count	Parts Per Million
bin[ -MAX: <=-1 ] = 0	0	0
bin[ >-1: <=0 ] = 0	0	0
bin[ >0: <=100 ] = 0	0	0
bin[ >100: <=200 ] = 47	47	435
bin[ >200: <=500 ] = 107757	107757	997750
bin[ >500: <=1000 ] = 152	152	1407
bin[ >1000: <=2000 ] = 12	12	111
bin[ >2000: <=5000 ] = 10	10	92
bin[ >5000: <=10000 ] = 3	3	27
bin[ >10000: <=20000 ] = 12	12	111
bin[ >20000: <=50000 ] = 7	7	64
bin[ >50000: <=100000 ] = 0	0	0
bin[ >100000: <=200000 ] = 0	0	0
bin[ >200000: <=500000 ] = 0	0	0
bin[ >500000: +MAX ] = 0	0	0

**Average Latency: 383**

Median Latency: 374

Standard Deviation: 70

Minimum Latency: 315

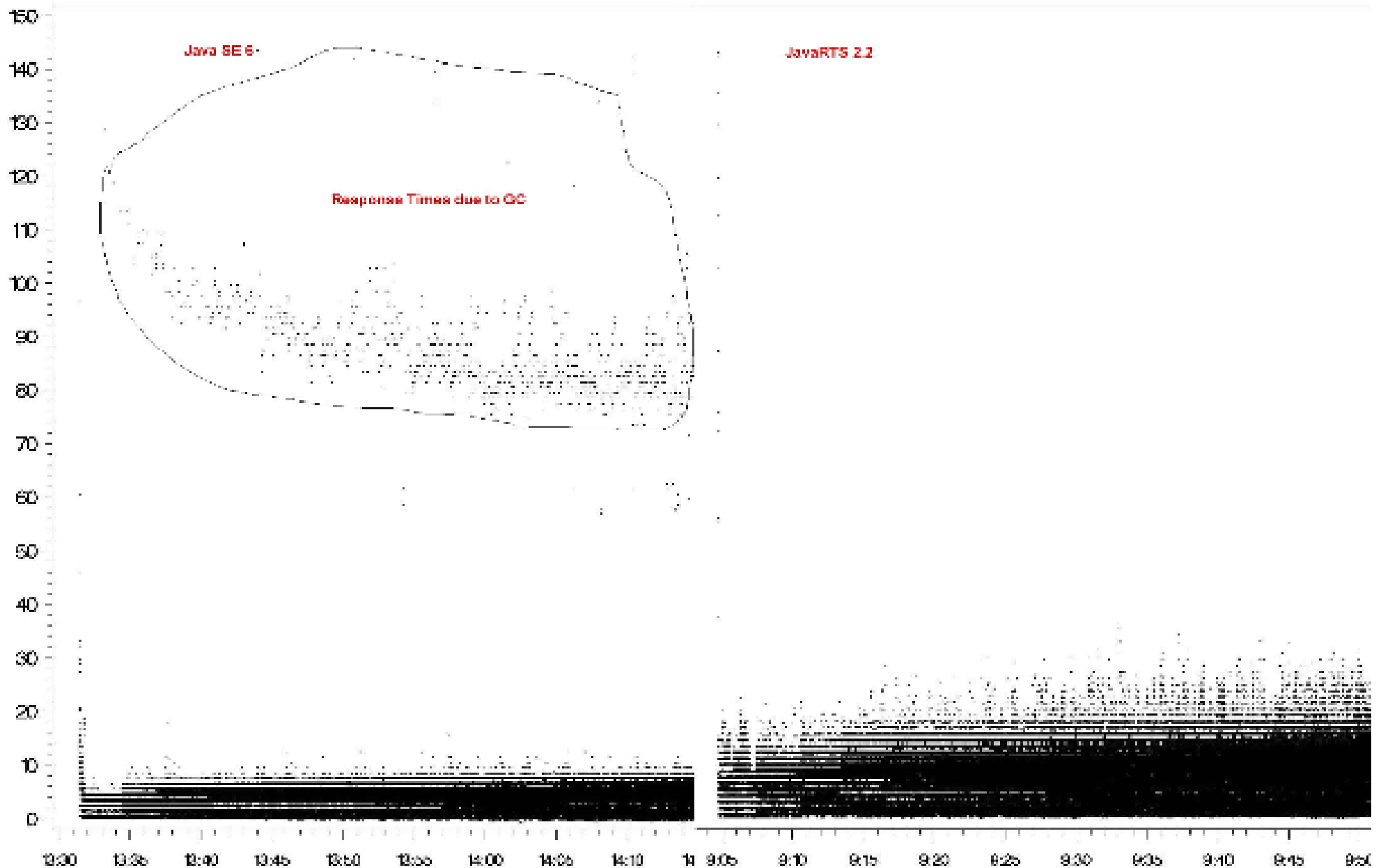
Histogram of Latencies (Range values in microseconds):

[ Range ]	Count	Parts Per Million
bin[ -MAX: <=-1 ] = 0	0	0
bin[ >-1: <=0 ] = 0	0	0
bin[ >0: <=100 ] = 0	0	0
bin[ >100: <=200 ] = 0	0	0
bin[ >200: <=500 ] = 296312	296312	987706
bin[ >500: <=1000 ] = 2533	2533	8443
bin[ >1000: <=2000 ] = 1126	1126	3753
bin[ >2000: <=5000 ] = 29	29	96
bin[ >5000: <=10000 ] = 0	0	0
bin[ >10000: <=20000 ] = 0	0	0
bin[ >20000: <=50000 ] = 0	0	0
bin[ >50000: <=100000 ] = 0	0	0
bin[ >100000: <=200000 ] = 0	0	0
bin[ >200000: <=500000 ] = 0	0	0
bin[ >500000: +MAX ] = 0	0	0

## POC Example 3

- Another message processing app (customer requests)
- Extreme concurrency (thousands of threads)
- Lots of locking and contention
- Tuned for years for Java SE
- GC-related latency and pauses are the issue
  - > 90 millisecond pauses
  - > Java RTS removed this, but also removed throughput
    - Cut by 50%

# POC Example 3 – Scatter Plot



# Conclusion

- Java RTS not ideal for all solutions
  - > Admittedly, Java RTS can probably be made more efficient
- RTSJ implementation adds overhead
  - > With good reason in the hard real-time space
- For throughput sensitive applications, a subset may be ideal. Examples:
  - > Remove Scoped Memory and IM, and hence memory access checks
  - > Remove priority inversion avoidance, and hence the penalty paid in locking and with “synchronized”

## More Information



- Available now
- Book information page:
  - > <http://www.informit.com/store/product.aspx?isbn=0137142986>
- Chapters online:
  - > <http://safari.informit.com/9780137153619>
- Also: [www.ericbruno.com/realtime](http://www.ericbruno.com/realtime)

## More Information

- Email contact information:
  - > [eric@ericbruno.com](mailto:eric@ericbruno.com)

