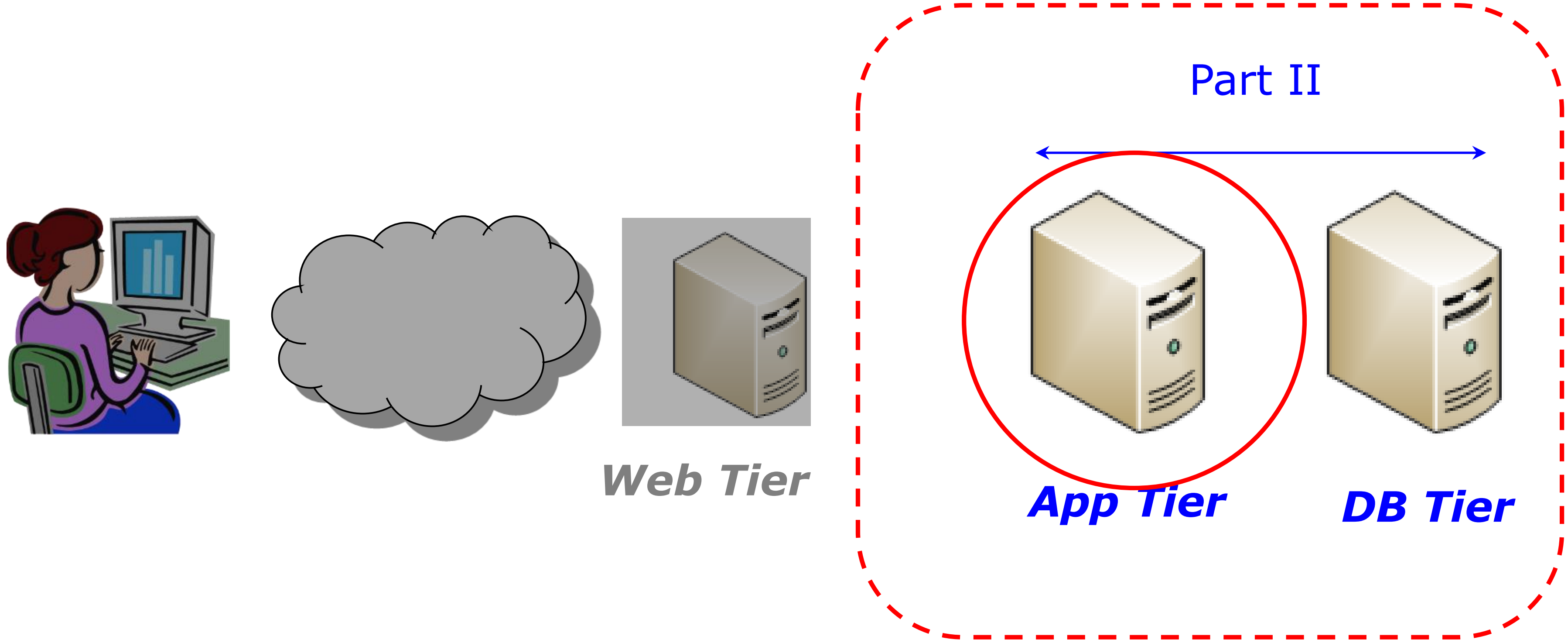
Performance Tuning Guidelines for Multi-tier Applications

Prajakta Bhatt, Infosys

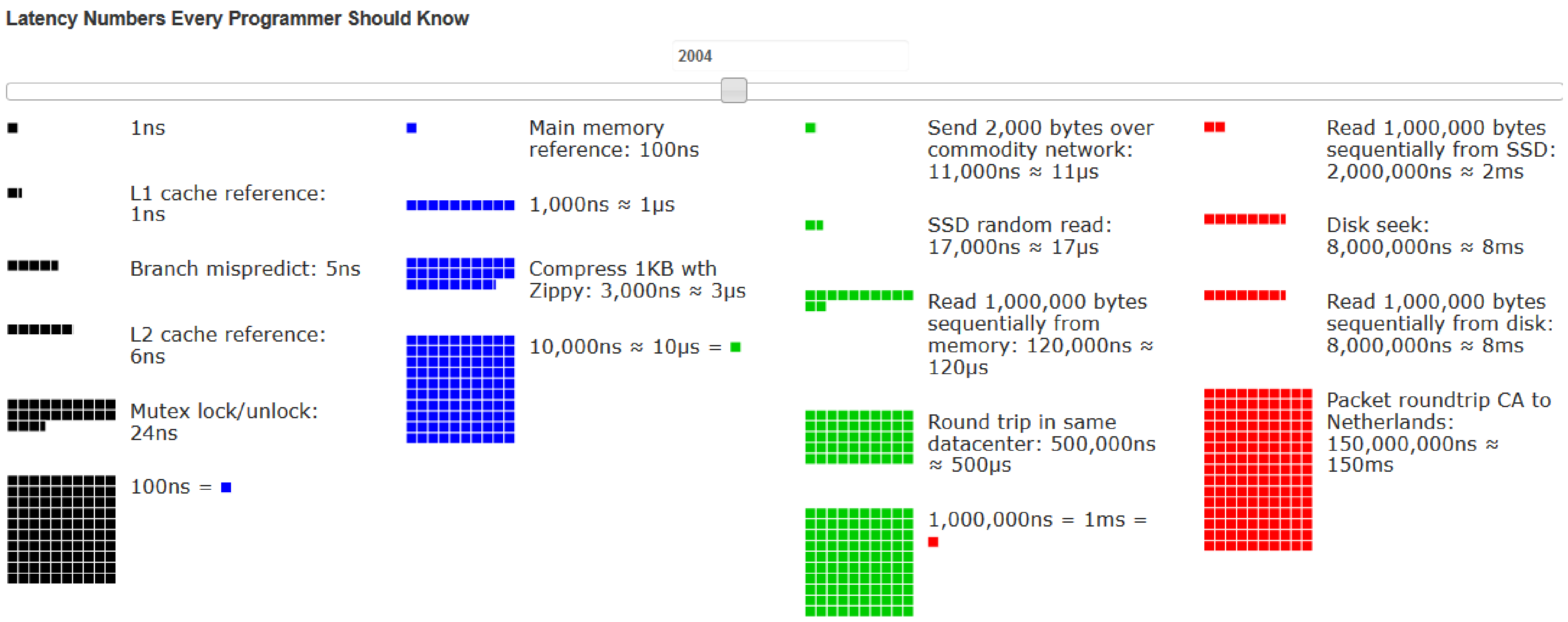
Sundarraj Kaushik, TCS

www.cmgindia.org

PART II: BACKEND TUNING: APP & DB SERVER



Numbers Every Architect Should Know - 2004



Latency Numbers Every Programmer Should Know

2004

1ns

L1 cache reference: 1ns

Branch mispredict: 5ns

L2 cache reference: 6ns

Mutex lock/unlock: 24ns

100ns = ■

Main memory reference: 100ns

1,000ns ≈ 1µs

Compress 1KB wth Zippy: 3,000ns ≈ 3µs

10,000ns ≈ 10µs = ■

Send 2,000 bytes over commodity network: 11,000ns ≈ 11µs

SSD random read: 17,000ns ≈ 17µs

Read 1,000,000 bytes sequentially from memory: 120,000ns ≈ 120µs

Round trip in same datacenter: 500,000ns ≈ 500µs

1,000,000ns = 1ms = ■

Read 1,000,000 bytes sequentially from SSD: 2,000,000ns ≈ 2ms

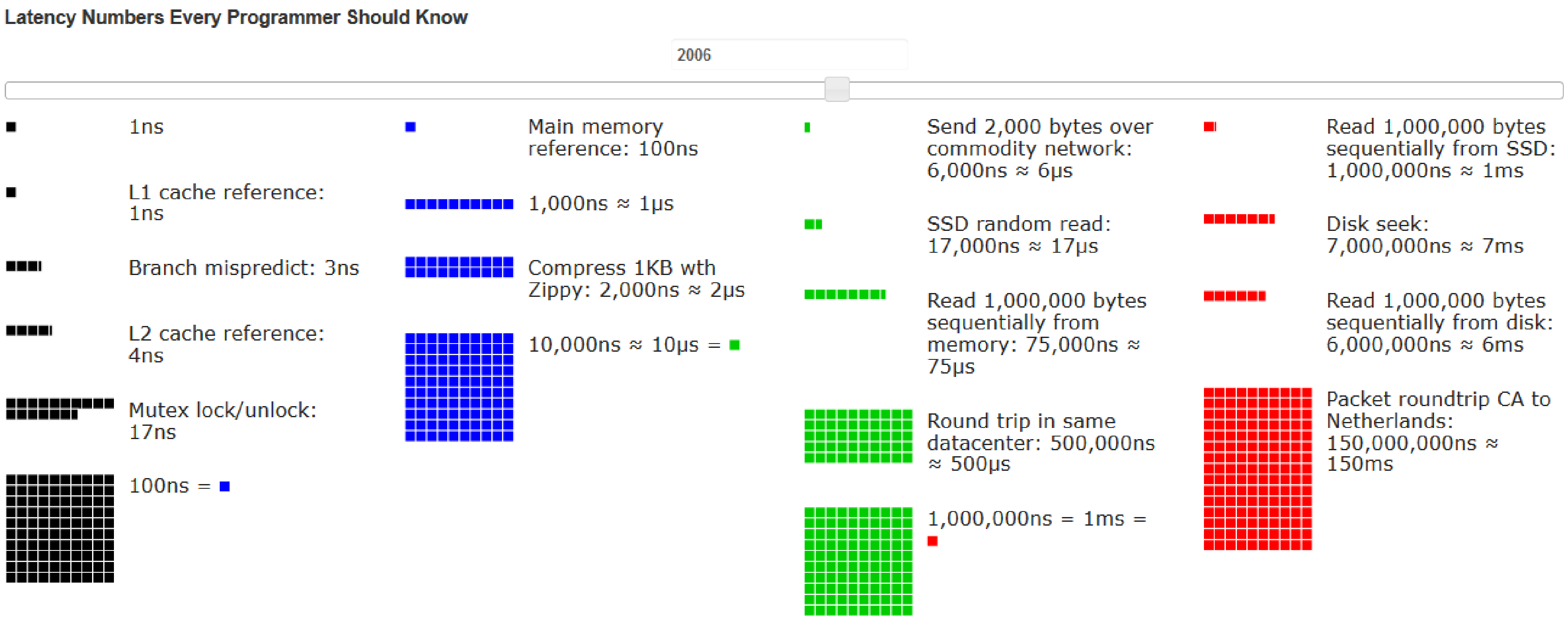
Disk seek: 8,000,000ns ≈ 8ms

Read 1,000,000 bytes sequentially from disk: 8,000,000ns ≈ 8ms

Packet roundtrip CA to Netherlands: 150,000,000ns ≈ 150ms

Source: http://www.eecs.berkeley.edu/~rcs/research/interactive_latency.html

Numbers Every Architect Should Know - 2006



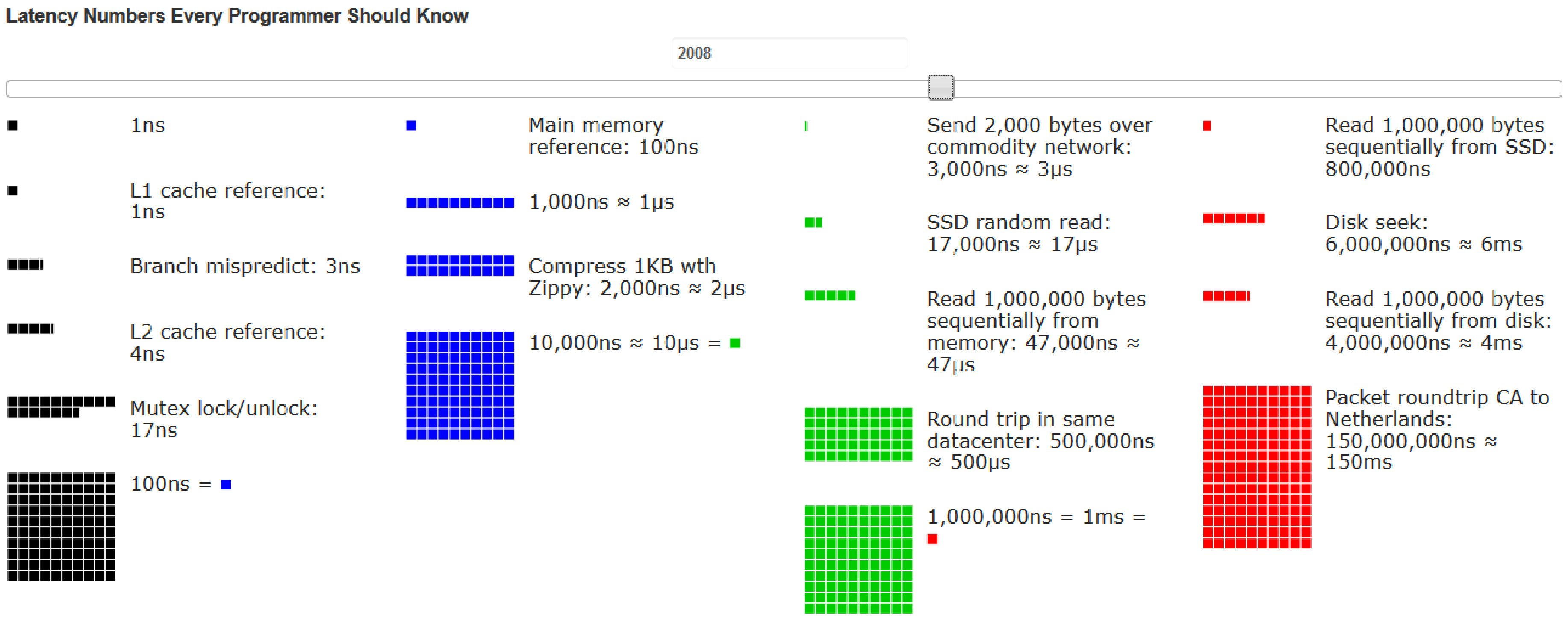
Latency Numbers Every Programmer Should Know

2006

- 1ns
- L1 cache reference: 1ns
- Branch mispredict: 3ns
- L2 cache reference: 4ns
- Mutex lock/unlock: 17ns
- 100ns = ■
- Main memory reference: 100ns
- 1,000ns ≈ 1µs
- Compress 1KB wth Zippy: 2,000ns ≈ 2µs
- 10,000ns ≈ 10µs = ■
- Send 2,000 bytes over commodity network: 6,000ns ≈ 6µs
- SSD random read: 17,000ns ≈ 17µs
- Read 1,000,000 bytes sequentially from memory: 75,000ns ≈ 75µs
- Round trip in same datacenter: 500,000ns ≈ 500µs
- 1,000,000ns = 1ms = ■
- Read 1,000,000 bytes sequentially from SSD: 1,000,000ns ≈ 1ms
- Disk seek: 7,000,000ns ≈ 7ms
- Read 1,000,000 bytes sequentially from disk: 6,000,000ns ≈ 6ms
- Packet roundtrip CA to Netherlands: 150,000,000ns ≈ 150ms

Source: https://gist.github.com/jboner/2841832
Image from: http://i.imgur.com/k0t1e.png

Numbers Every Architect Should Know - 2008



Latency Numbers Every Programmer Should Know

2008

1ns

L1 cache reference: 1ns

Branch mispredict: 3ns

L2 cache reference: 4ns

Mutex lock/unlock: 17ns

100ns = ■

Main memory reference: 100ns

1,000ns ≈ 1µs

Compress 1KB wth Zippy: 2,000ns ≈ 2µs

10,000ns ≈ 10µs = ■

Send 2,000 bytes over commodity network: 3,000ns ≈ 3µs

SSD random read: 17,000ns ≈ 17µs

Read 1,000,000 bytes sequentially from memory: 47,000ns ≈ 47µs

Round trip in same datacenter: 500,000ns ≈ 500µs

1,000,000ns = 1ms = ■

Read 1,000,000 bytes sequentially from SSD: 800,000ns

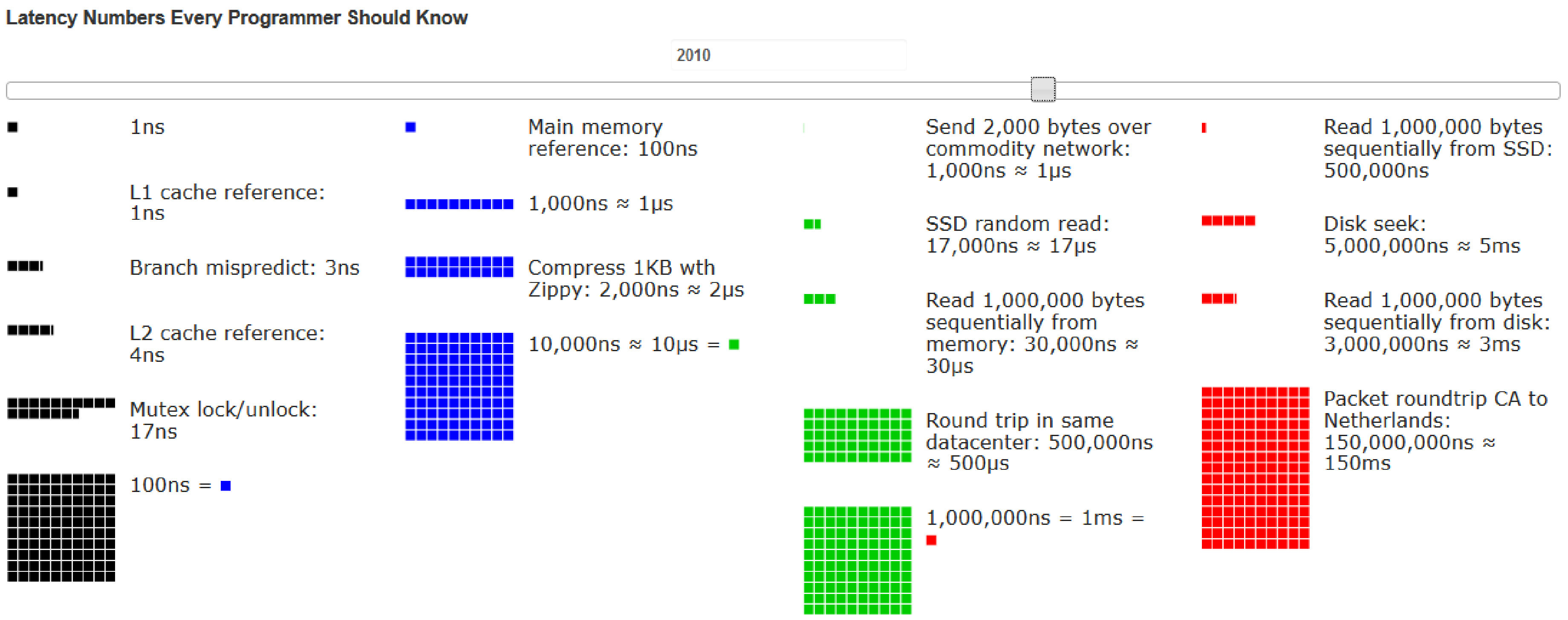
Disk seek: 6,000,000ns ≈ 6ms

Read 1,000,000 bytes sequentially from disk: 4,000,000ns ≈ 4ms

Packet roundtrip CA to Netherlands: 150,000,000ns ≈ 150ms

Source: https://gist.github.com/jboner/2841832
Image from: http://i.imgur.com/k0t1e.png

Numbers Every Architect Should Know - 2010



Latency Numbers Every Programmer Should Know

2010

1ns

L1 cache reference: 1ns

Branch mispredict: 3ns

L2 cache reference: 4ns

Mutex lock/unlock: 17ns

100ns = ■

Main memory reference: 100ns

1,000ns ≈ 1µs

Compress 1KB wth Zippy: 2,000ns ≈ 2µs

10,000ns ≈ 10µs = ■

Send 2,000 bytes over commodity network: 1,000ns ≈ 1µs

SSD random read: 17,000ns ≈ 17µs

Read 1,000,000 bytes sequentially from memory: 30,000ns ≈ 30µs

Round trip in same datacenter: 500,000ns ≈ 500µs

1,000,000ns = 1ms = ■

Read 1,000,000 bytes sequentially from SSD: 500,000ns

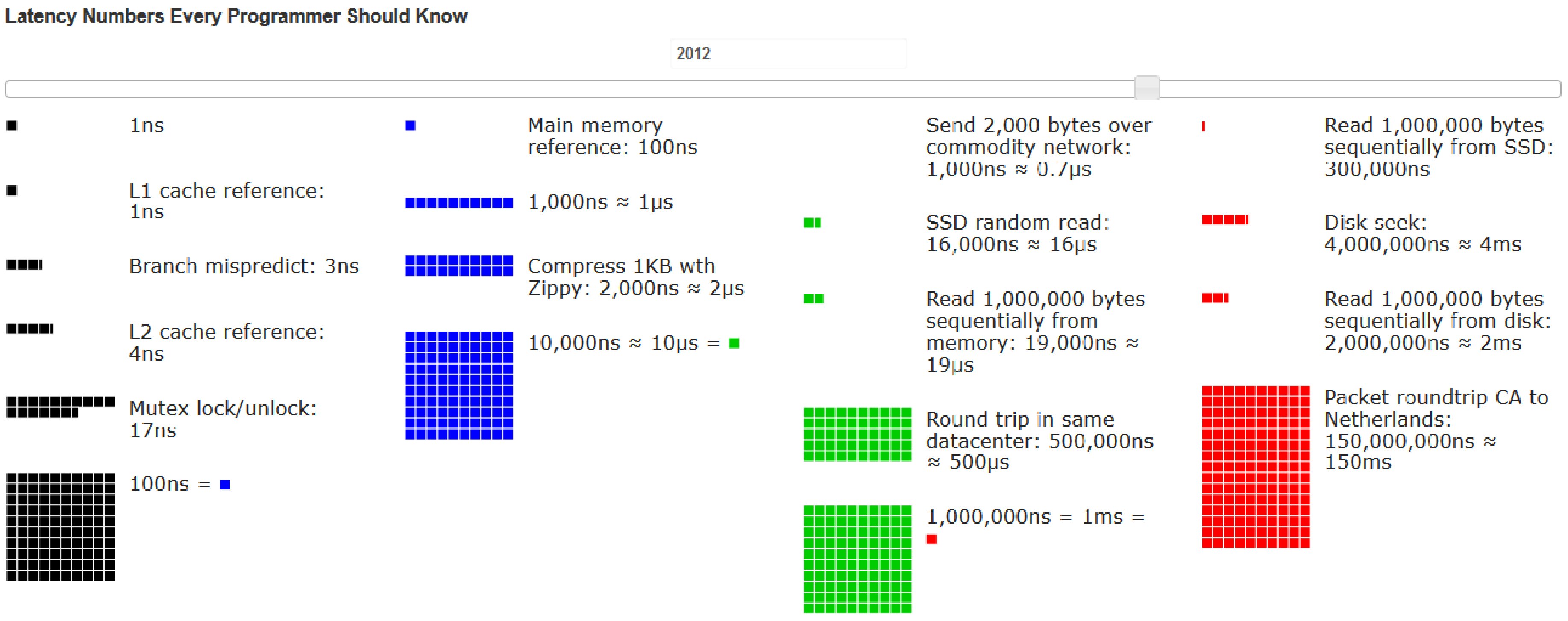
Disk seek: 5,000,000ns ≈ 5ms

Read 1,000,000 bytes sequentially from disk: 3,000,000ns ≈ 3ms

Packet roundtrip CA to Netherlands: 150,000,000ns ≈ 150ms

Source: https://gist.github.com/jboner/2841832
Image from: http://i.imgur.com/k0t1e.png

Numbers Every Architect Should Know - 2012



Latency Numbers Every Programmer Should Know

2012

1ns

L1 cache reference: 1ns

Branch mispredict: 3ns

L2 cache reference: 4ns

Mutex lock/unlock: 17ns

100ns = ■

Main memory reference: 100ns

1,000ns ≈ 1μs

Compress 1KB wth Zippy: 2,000ns ≈ 2μs

10,000ns ≈ 10μs = ■

Send 2,000 bytes over commodity network: 1,000ns ≈ 0.7μs

SSD random read: 16,000ns ≈ 16μs

Read 1,000,000 bytes sequentially from memory: 19,000ns ≈ 19μs

Round trip in same datacenter: 500,000ns ≈ 500μs

1,000,000ns = 1ms = ■

Read 1,000,000 bytes sequentially from SSD: 300,000ns

Disk seek: 4,000,000ns ≈ 4ms

Read 1,000,000 bytes sequentially from disk: 2,000,000ns ≈ 2ms

Packet roundtrip CA to Netherlands: 150,000,000ns ≈ 150ms

Source: https://gist.github.com/jboner/2841832
Image from: http://i.imgur.com/k0t1e.png

Typical Activities in the Application Layer

1. Authenticate
   a) Check with LDAP
   b) Validate SSO Token
   c) Check/Validate in session
2. Authorize Request
   a) Check ACL in LDAP
3. Read and interpret the request parameters
4. Validate the Request Parameters
   a) Read from Data Store
   b) Make a web-service call
5. Perform Business Operations
   a) Read from Data Store
6. Persist Data to Data Store
7. Send SMS/Email
8. Form and Send the HTML/JSON/XML Response

Configuring the Operating System

1. Number of open files. Determines total number of TCP Connections into the system.

2. Configure Maximum Transmission Unit (MTU)
    1. Large Media files
    2. Large Data fetches from DB as in Data Warehouses

3. Receive Send Buffer sizes

4. Configure tcp_fin_timeout
    1. If servers are expected to have multiple short lived connections

5. Configure TCP Receive Window (RWIN)
    1. Only for sites expected to serve clients over fast WAN networks

Configuring the Application Server

1. Maximum Number of Connections
2. Maximum Number of HTTP requests
3. Maximum Number of ORB threads if EJBs are being used
4. Database Connection Pooling
    1. Prepared Statement Caching
5. Message Queue Connection Pooling
6. LDAP Connection Pooling
7. HTTP Connection Pooling for Web Services
8. TCP Connection Pooling
9. Time out for all Connections
10. Set Log Level to WARN or ERROR

For implementing connection pooling consider Apache Commons Pool if it is not provided by the client library being used and if the Platform does not provide for one.

Configuring Java Application Server

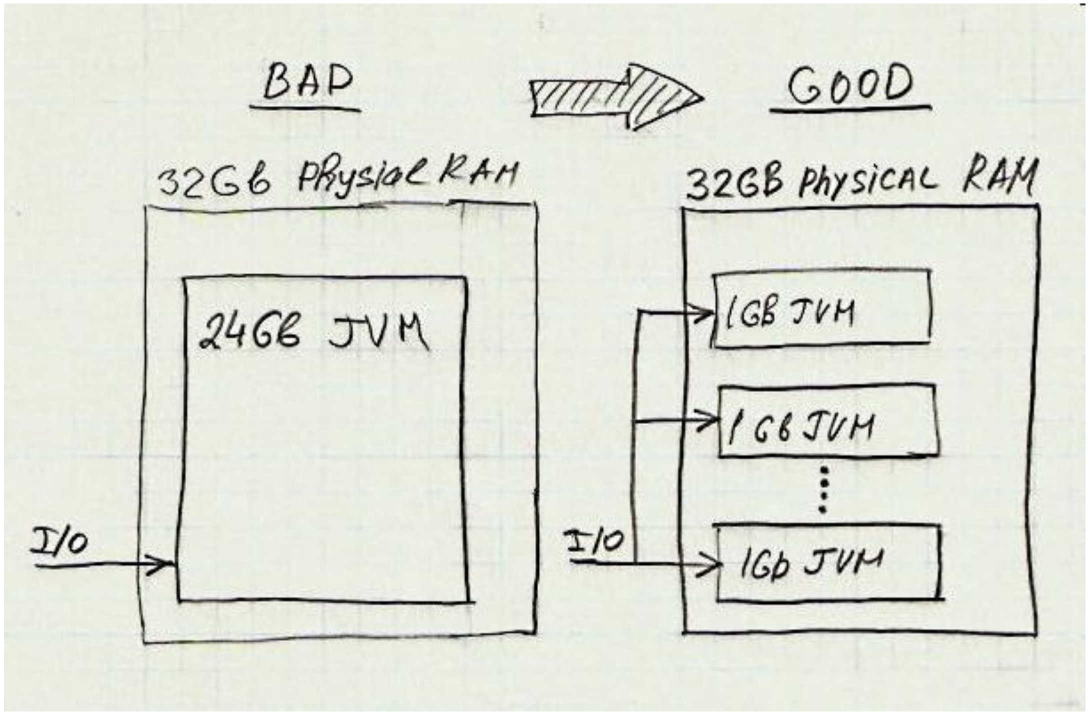
1. Maximum Heap Size

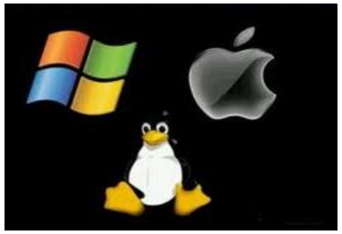
2. Maximum PermGenSize

3. Garbage Collection Algorithm
   1. Parallel
   2. Concurrent Mark Sweep
   3. G1GC

JVMs and Memory Allocations

- 64bit JVM allows us to set higher values of heap size
- But bigger the heap, longer the major GC and hence pause times.
- Multiple Smaller JVMs
- A suitable Garbage Collection Algorithm

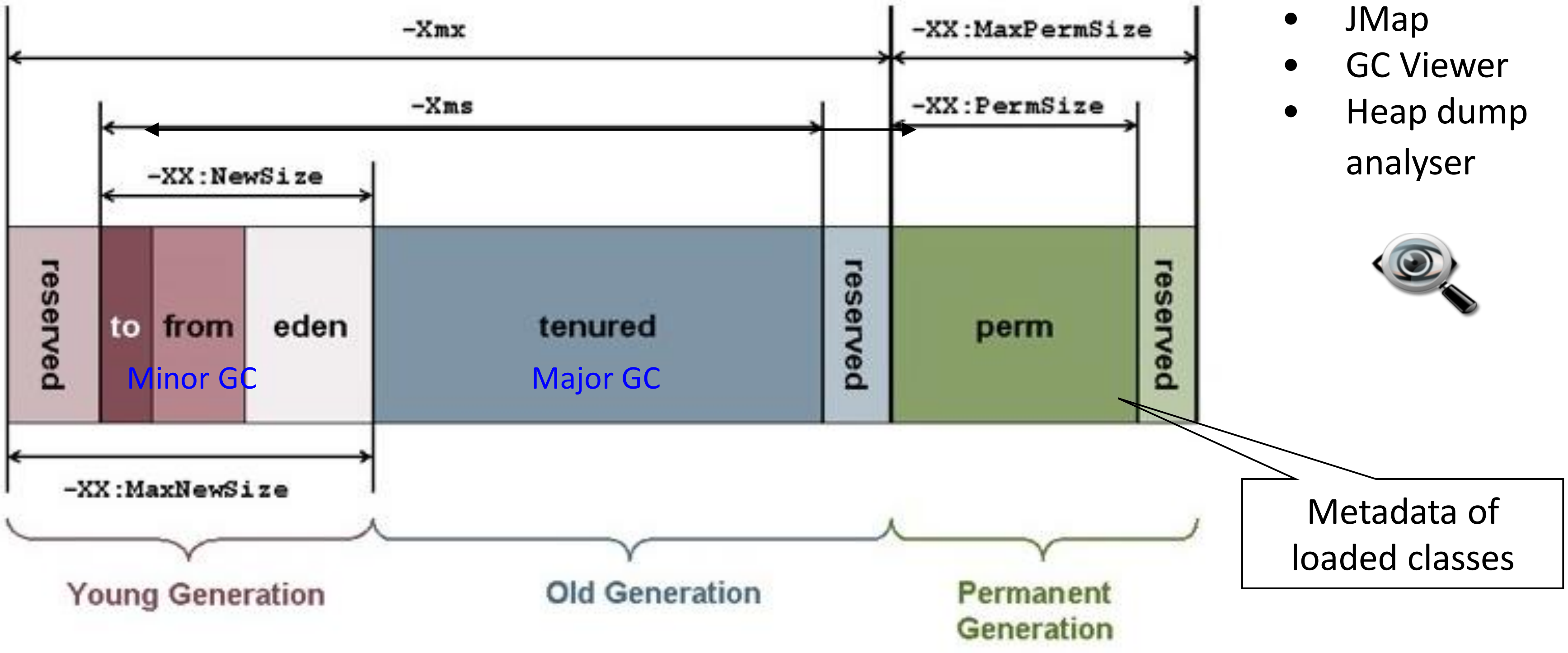




OS: Room for me?

Out of Memory

- Exception in thread “main” java.lang.OutOfMemoryError: PermGen space ?
- Exception in thread "main" java.lang.OutOfMemoryError: Java heap space ?



- Visual VM
- JMap
- GC Viewer
- Heap dump analyser

Note: Specific to SUN JVM. Internals may vary depending on choice of JVM

Designing For Performance - 1

1. Minimize I/O
    1. Cache in memory
    2. Minimize interaction with remote systems
2. Make long running processing asynchronous
    1. Sending Email
    2. Sending SMS
    3. Sending Notification to another system

    Note: Do not use Database as the Messaging System; use a proper Message Queue
3. Leverage the right platforms
    1. Use Text Search Engines like Solr, Lucene for Text search instead of Database Full Text Search
    2. Use Redis or other NoSQL for maintaining online counts and distributed key values
4. Use File Systems for storing Files
    1. Avoid using CLOBs and BLOBs for this purpose
5. Design for Horizontally Scalability; Depend not on Vertical Scalability
    1. Mind your Singletons
    2. Mind your Synchronization

Designing For Performance - 2

6. Design for proper Utilitization of Session
    1. Should be possible to replicate in a clustered environment
    2. Should not be used for storing large data
    3. Set Proper Time out parameter
    4. Plan to enable Sticky Sessions
7. Optimizing Content Management System
    1. Leveraging caching
    2. If in CMS uses database as the store, optimize the queries

Server Side Caching mechanisms

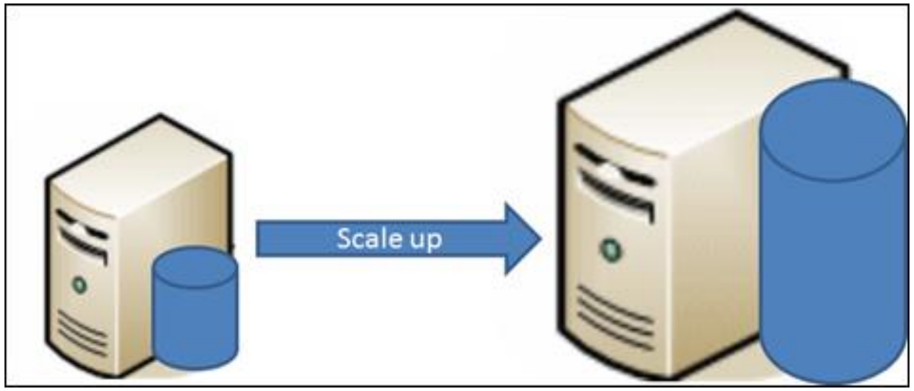
- In JVM
  - Extremely Fast
  - Occupies precious JVM heap space
  - May trigger early frequent Garbage Collection
  - Can slow down and crash system
  - Not easy to replicate changes across JVMs in a cluster
- Out of JVM
  - Relatively slower
  - Easier Invalidation/Updation
  - Will overcome all the above mentioned problems

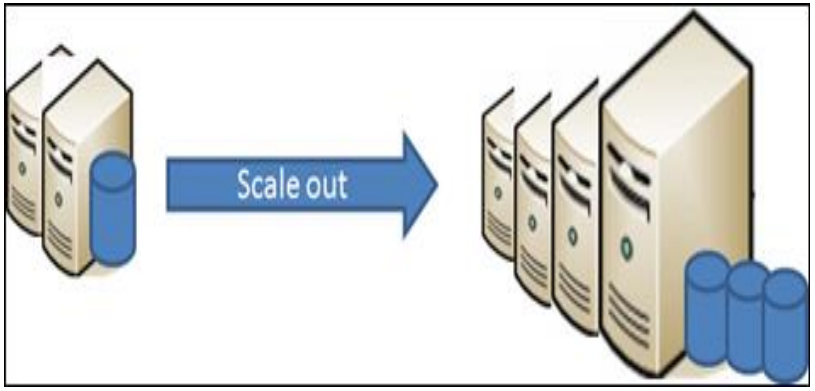
Some Distributed Caching Platforms

1. Memcached
2. CouchBase/ Membase (A fork of memcached)
3. Hazelcast
4. Redis from Google
5. Infinispan from JBoss
6. Terracotta
7. Coherence from Oracle
8. Gemfire from Gemstone
9. Xtreme Scale from IBM

Do not forget connection pooling when connecting to any of them
Access these through an Interface to ease changing the caching mechanism

Application Tier Scale Out Vs. Scale Up





Simple

Limited Scalability

Complex

Unlimited Scalability

Coding For Performance - 1

1. Minimize I/Os
    1. No writing to console
    2. Adopt right Logging practices
    3. Using StringBuilder or StringBuffer for String concatenation
2. Optimize I/O
    1. Read in chunks specific to the I/O
    2. Avoid repeated I/Os
3. Close Resources in finally
4. Follow Coding Best Practices
    1. Use PMD, CheckStyle, FindBugs to identify and remove violations
    2. Use UnitPerf if available to test the performance of granular functions
    3. Use one version of standard libraries across all applications to be codeployed
5. Optimize XML Parsing
    1. Use SAX Parsing when processing large XML
    2. Use DOM Parsing only when back and forth movement is required.

Coding For Performance - 2

6. Synchronize at the most granular level
    1. Avoid Class level and Method level synchronization
    2. Prefer variable level synchronization

Optimizing Web Services

- Avoid chatty web services
  - Use coarse grained web services
- Minimize the payload
- Avoid maintaining server states between calls
- Document/Literal encoding to RPC/SOAP encoding
  - Results in smaller and less complex messages
  - It promotes interoperability
- Minimize parsing of XMLs at the gateway
  - A Gateway may parse the XML for security or routing reasons. Use partial parsing provided by most vendors.
- Use SAX parser instead of the DOM parser
- Use as simple an XML message as possible
- ws-security adds its own overheads.

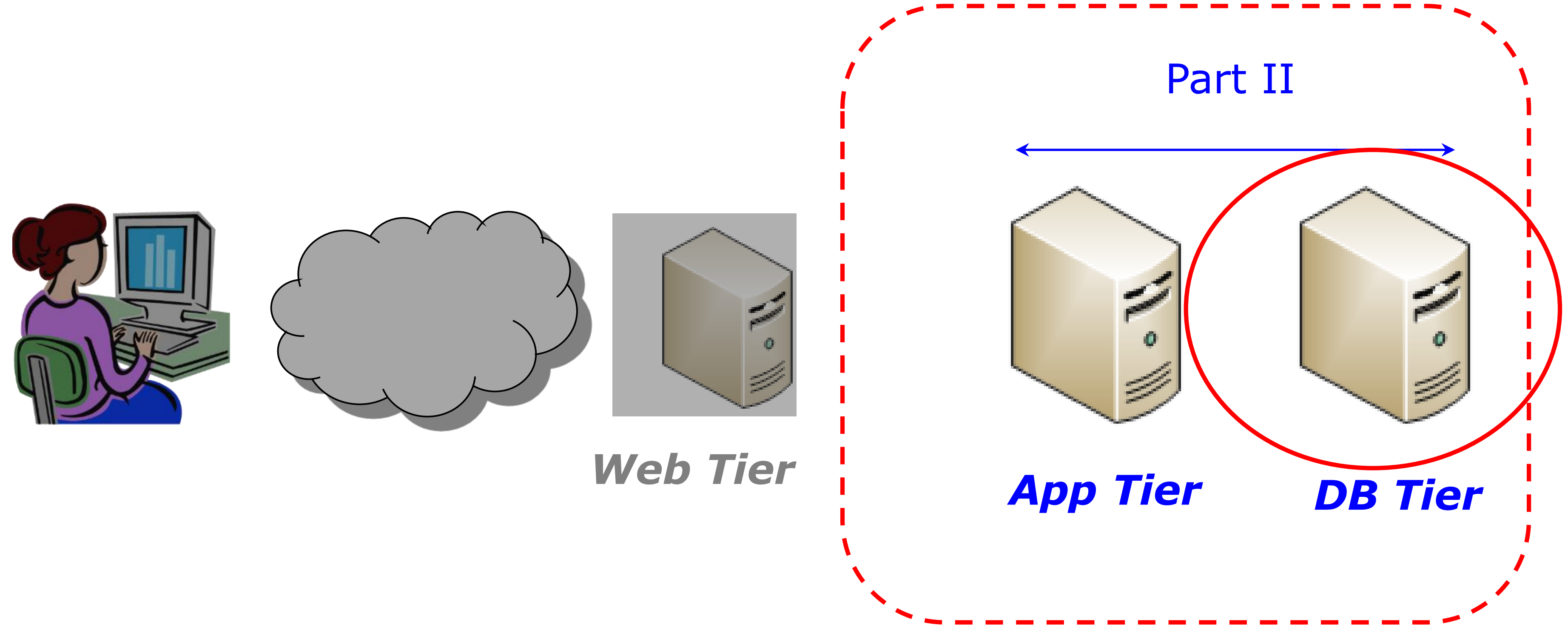
http://www.ibm.com/developerworks/library/ws-whichwsdl/
http://www.ibm.com/developerworks/library/ws-best9/
https://msdn.microsoft.com/en-us/library/ff647786.aspx

Best Practices with Libraries

<u>Do’s & don’ts</u>

1. Avoid duplicates jar files
2. Avoid multiple version of same jar files
3. Remove unwanted jars
4. Use latest stable versions of library files
5. Use Server default Library NOT Application library

PART II: BACKEND TUNING: APP & DB SERVER



Designing Database for Performance

1. Normalize for OLTP
2. Denormalize for Reporting and OLAP
3. Design to have a separate instance for Reporting
4. Design to precompute complex calculations via EOD
5. Design to avoid complex queries
6. Design to limit joins to 3 tables
7. Identify Indices to be created
8. Identify tables to partition and the right keys
9. Prepare archival policy and scripts for the tables

Configuring Database Servers for Performance

1. Configure the maximum number of expected Connections
2. Configure the maximum number of concurrent queries
3. Configure the different memory to be used
    1. Memory for Query Caching
    2. Memory for Sorting
4. Provide a Good and Fast Storage Device with the correct RAID level
    1. Use RAID 10
    2. Use RAID 5 to optimize storage usage
5. Provide separate storage for Logs, Data, Temporary space.
6. Configure the Timeouts right
7. Check Vendor specific recommendations like
    1. Setting Redo Size
    2. Setting Rollback Segment
    3. Sizing Temporary Space
    4. How to creating Table Spaces and Index Spaces
    5. How to setting Percentage Free, Free List etc.
    6. Enabling Caching of sequence generation

Coding Database Access for Performance

1. Fetch only rows required
2. Fetch only columns that are required; Avoid Select * from
3. Update only columns that have changed
4. Close ResultSet, Statements, and Connections in finally
5. Keeps Transactions Small
6. Commit in Batches
7. Use optimized query for pagination
8. Be wary of 1 + n problem of Object Relation Mapping libraries
9. Be wary of eager fetching of related tables in ORM
10. Be wary of sticking to simple, strict ORM methods
    1. Use Criteria to filter
    2. Use Projections to limit the columns fetched
    3. Use Projections to fetch standard summarizations like count, sum etc.
11. Use Prepared Statements
12. Carry out regular housekeeping
    1. Defragment Space
    2. Re Index
    3. Regenerate Statistics
    4. Shrink / Vacuum etc.

Reduce Table Joins

- Requirement – To fetch list of all products sold, along with their rate, quantity, sell date and customer to whom products were sold
- Typical Join
  - Customer Master (100000)
  - Product Master (100000)
  - Bill Header (100000)
  - Bill Details (500000) (Five items per bill)
- Instead
  - Join only Bill Header and Bill Details
  - Fetch the Customer Name, and the Product Name from a cache of Hashmaps
    - Customer Id – Customer Name
    - Product Id – Product Name

Typical reasons for slow queries

- Queries performing Table Scans and Index Scans
- Reasons why full table/index scans are performed
  - no WHERE clause
  - no index on any field in WHERE clause
  - poor selectivity on an indexed field
  - too many records meet WHERE conditions
  - using SELECT * FROM
  - function used on the indexed column in the query e.g.
    - TRUNC(ModifiedDate) > TRUNC(SYSDATE) or
    - UPPER(name) = <name>
- Queries waiting for locks or deadlocks
  - Keep transaction as small as possible
  - Identify long wait events; identify cause and eliminate

Generate Explain plans to understand how database query optimizer chooses to execute a query

Close ResultSet, Statement, Connection

```
Connection lConnection = null;
Statement lStatement = null;
ResultSet lResultSet = null;
try {
     lConnection = getConnection();
     lStatement = lConnection.createStateement();
     lResultSet = lStatement.executeQuery();
}
catch(SQLException lSQLException) {
}
finally {
     lResultSet.close();
     lStatement.close();
     lConnection.close();
}
```

Will not work properly. If exception is thrown in lResultSet.close() then the Statement and Connection will not be closed.

Close ResultSet, Statement, Connection

```
Connection lConnection = null;
Statement lStatement = null;
ResultSet lResultSet = null;
try {
     lConnection = getConnection();
     lStatement = lConnection.createStateement();
     lResultSet = lStatement.executeQuery();
}
catch(SQLException lSQLException) {
}
finally {
     try {
         lResultSet.close();
     }
     catch (SQLException lSQLException) {<Log Exception>}
     Similar for ... lStatement
     Similar for ... lConnection.close();
```

All resources will be properly closed.

Commit in Batches

```
Connection lConnection = null;
PreparedStatement lPreparedStatement = null;
List<> lRecordsToPersist;
try {
    lConnection = getConnection();
    //Begin Transaction
    lPreparedStatement = lConnection.prepareStatement(<InsertQuery>);
    for (lCount = 0; lCount < lRecordsToPersist.size(); lCount++) {
        setValues(lPreparedStatement, lRecordsToPersist.get(lCount));
        lPreparedStatement.execute();
        if (lCount % 100 == 0) {
            //Commit Transaction
            //Restart Transaction
        }
    }
    //Commit Transaction
}
finally {
    //Close all resources cleanly
}
```

ORM 1 + n Problem

- Requirement – To fetch list of all products sold, along with their rate, quantity, sell date and customer to whom products were sold
- Typical ORM statement
  - Order.fetch(<filter>) fetches List of Orders
    - For each Order Fetch Customer
    - For each Order Fetch Order Details
      - For each Order Detail Fetch Product
  - If there are 100 orders and if each order has on an average 10 items then the number of times the fetch must be done is
  - 1 (Order List) + 100 Customers + 100 Order Details + 100 * 5 Products
  - Total Fetches = 1 + 700 = 701
- To avoid this problem
  - Use a join

Stored Procedures

- Stored procedures have been the panacea of DBAs to solve performance issues
- Points to ponder before we use Stored Procedures:
  1. Gives performance gain? – YES, but not always
  2. Reduces traffic between Application and Database tier? – Depends
  3. Is it easy to write and maintain? – Maybe
  4. Is it easy to debug? – No
  5. Is it easy to test? – No
  6. Increases reusability? – Depends
  7. Does it add Security? – Not really
  8. Is it easy to refactor? – NO
  9. Is it easy to scale? – NO

Some Troubleshooting Tools for Databases

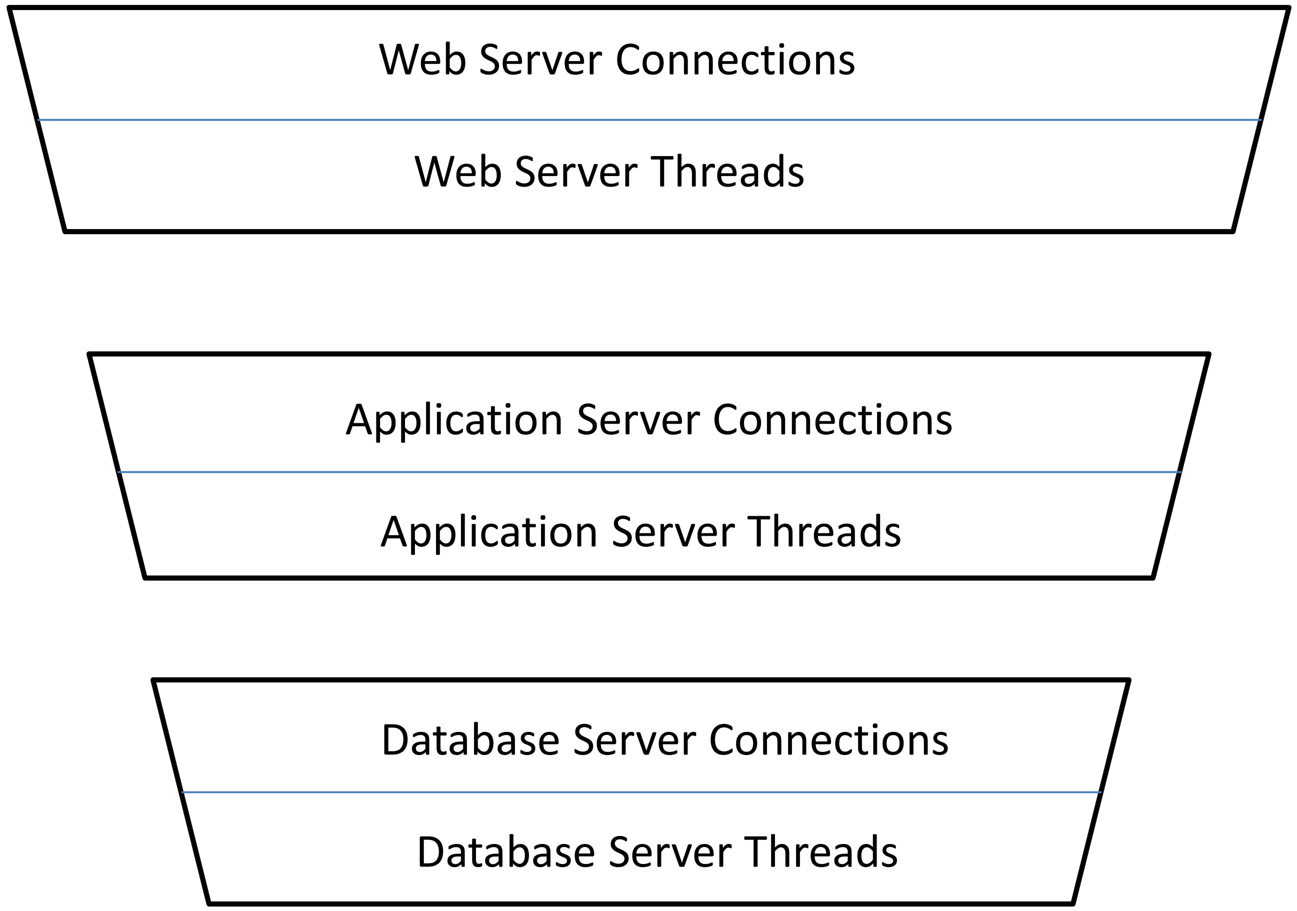
1. Oracle
    1. Oracle Enterprise Manager
    2. AWR Reports
    3. Stats Pack
2. DB2
    1. DB Snapshot
3. MSSQL
    1. SQL Trace
4. MySQL
    1. Slow Query Log
5. Postgres
    1. Slow Query Log

6. Explain Plan

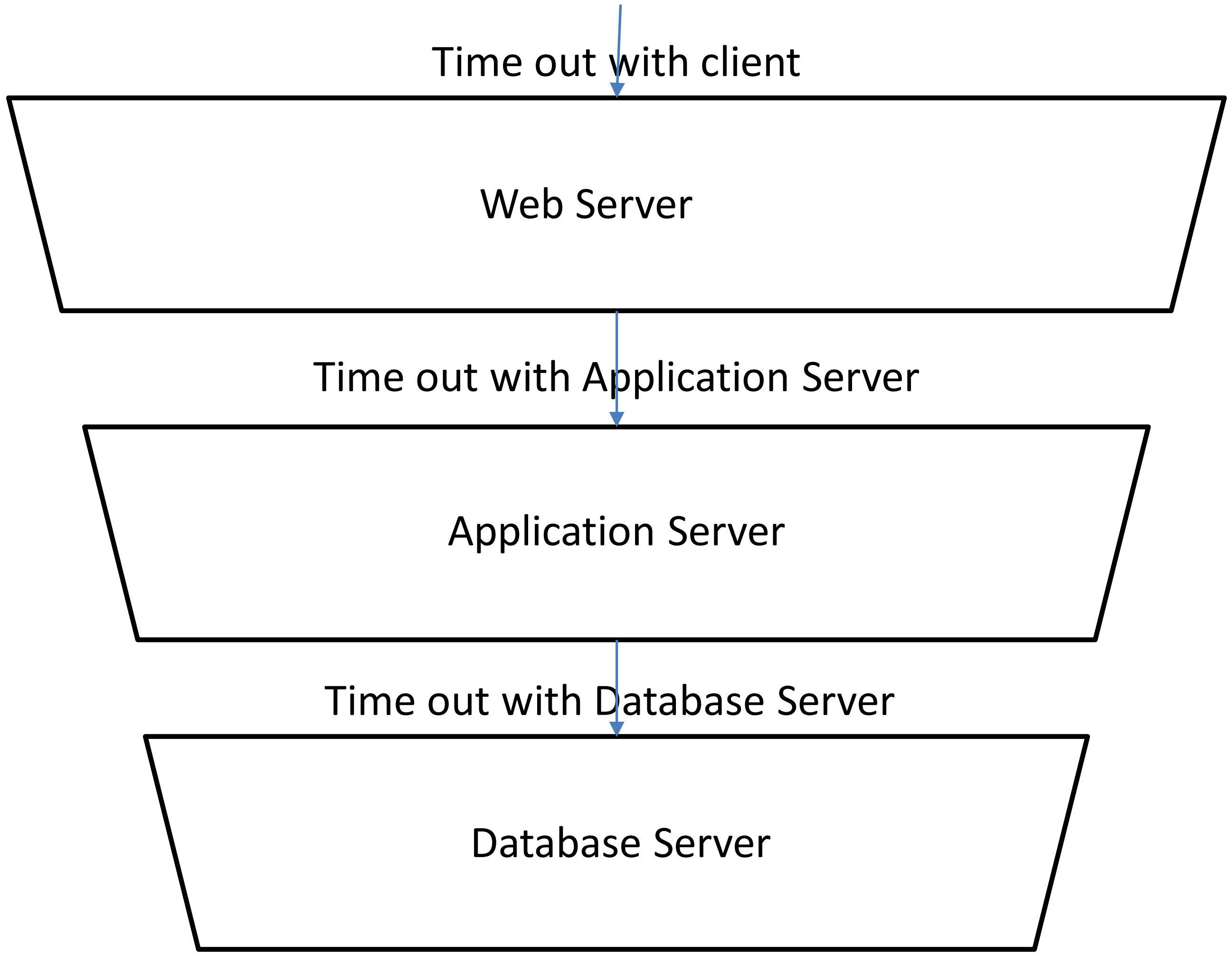
LDAP Server Optimizations

- Use Connection pooling
- Specify the right objectClass instead of a *
- Limit the number of entries to be retrieved
- Fetch only Attributes that are necessary
- Limit the time that the server should spend searching for entries
- Limit the number of entries the server needs to look into before returning back
- Design the trees to minimize traversal
- Enable and right size the cache in servers
- Set the database cache for LDAPs that depend on DB
- Create the right indexes
- Use Replication and Load Balancer to access multiple LDAP servers
- Depend on federation only when absolutely necessary
- Do not use LDAP to store frequently changing data

The Funnel Principle - Connections


Web Server Connections
Web Server Threads
Application Server Connections
Application Server Threads
Database Server Connections
Database Server Threads

The Funnel Principle - Timeouts

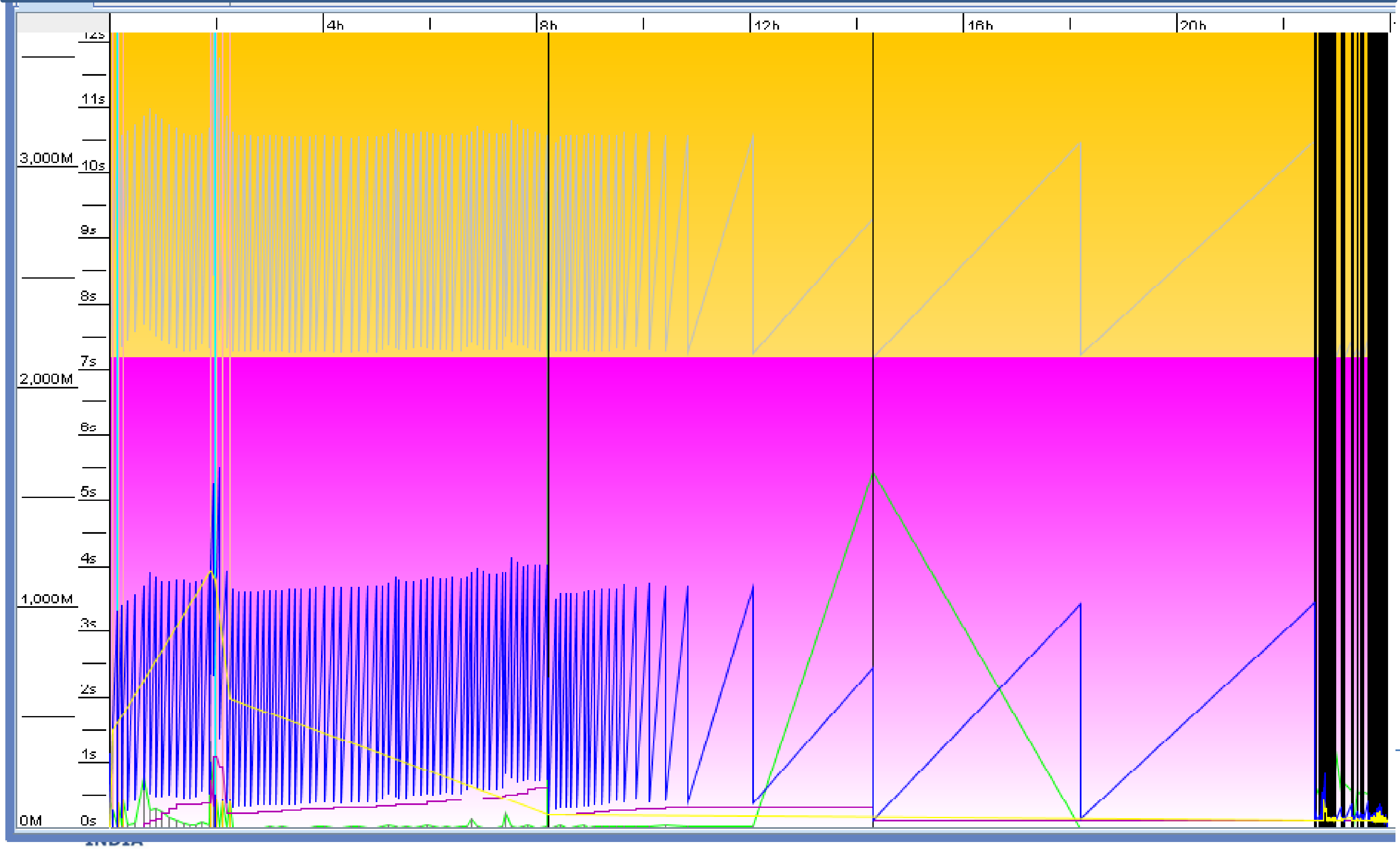


Some Troubleshooting Tools for Java

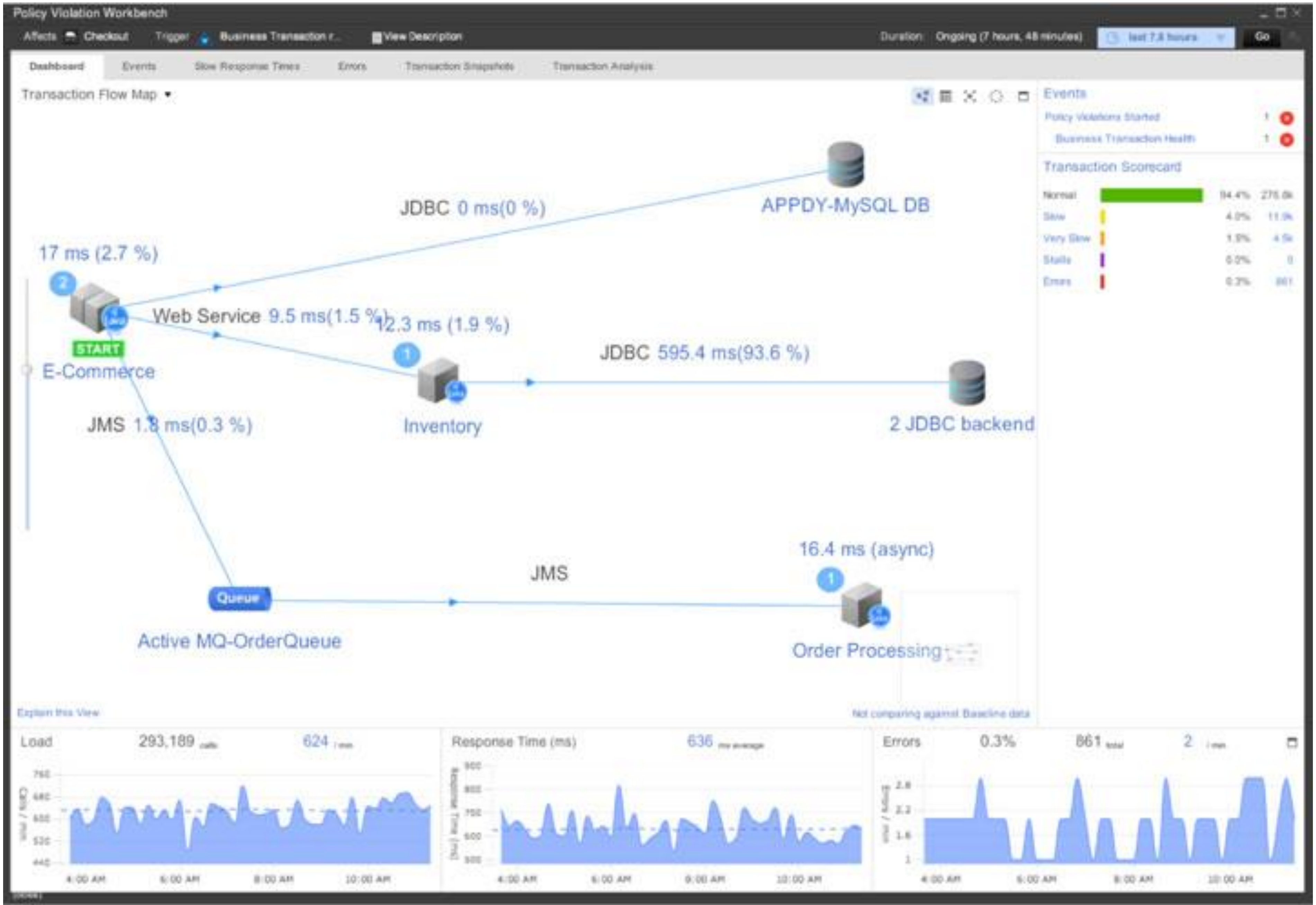
1. Visual VM or JConsole
2. JMAP: Summary – Provides summary of Memory Usage
3. JMAP: Heap Dump – Provides complete details of the Heap Memory
4. Thread Dumps – Provides details of what the Threads are doing
5. Use Profilers –
    1. Open Source or Home Grown
        1. Jensor
        2. Custom Profiler using Aspect Oriented Programming
    2. Commercial Profilers
        1. JProfiler
        2. YourKit
    3. APM Tools
        1. AppDynamics
        2. DynaTrace
        3. CA Wily Introscope

What is this sawtooth with Black Band at the end?

Garbage Collection Gone Wrong



Data flow across the infrastructure



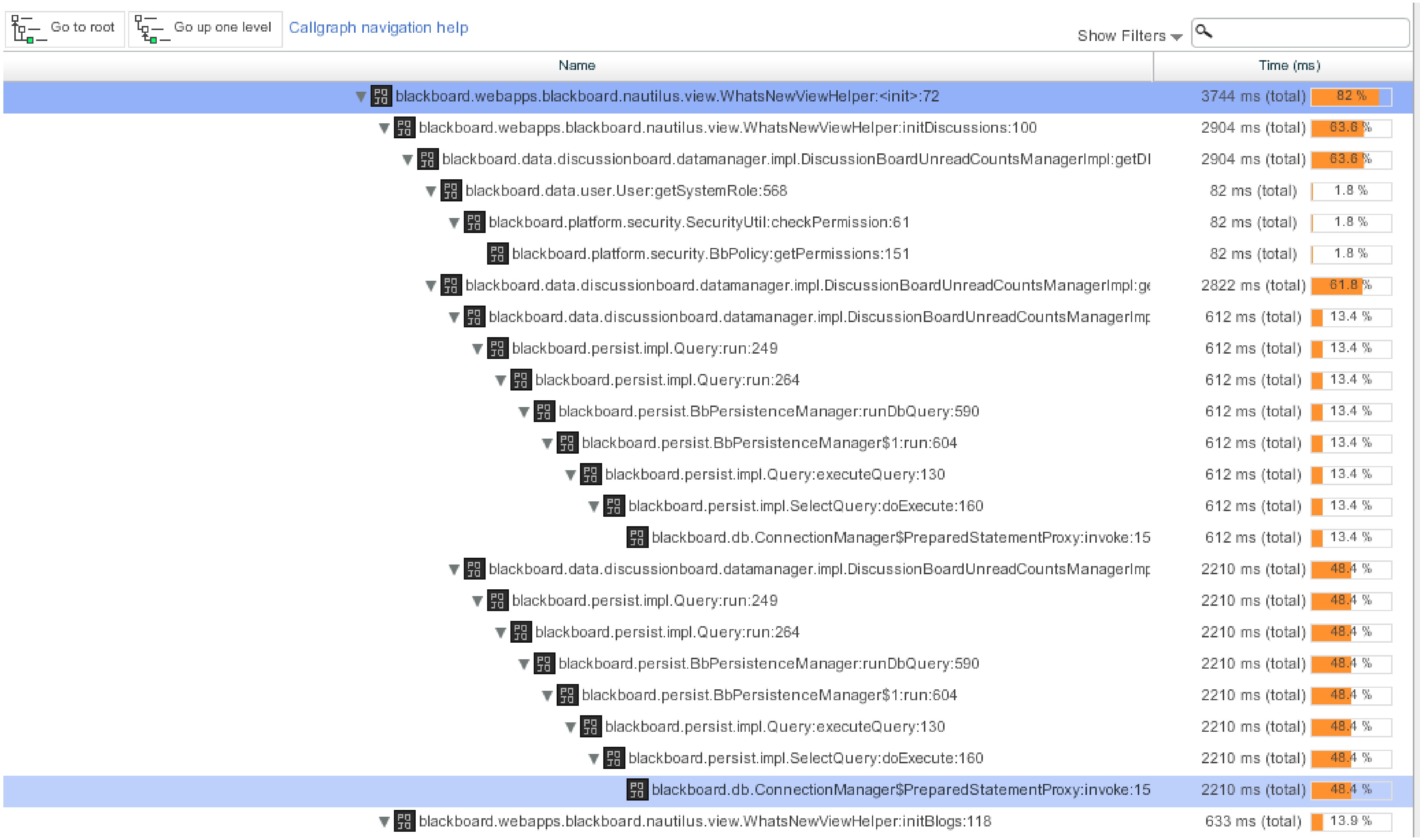
Events

Policy Violations Started	1	●
Business Transaction Health	1	●

Transaction Scorecard

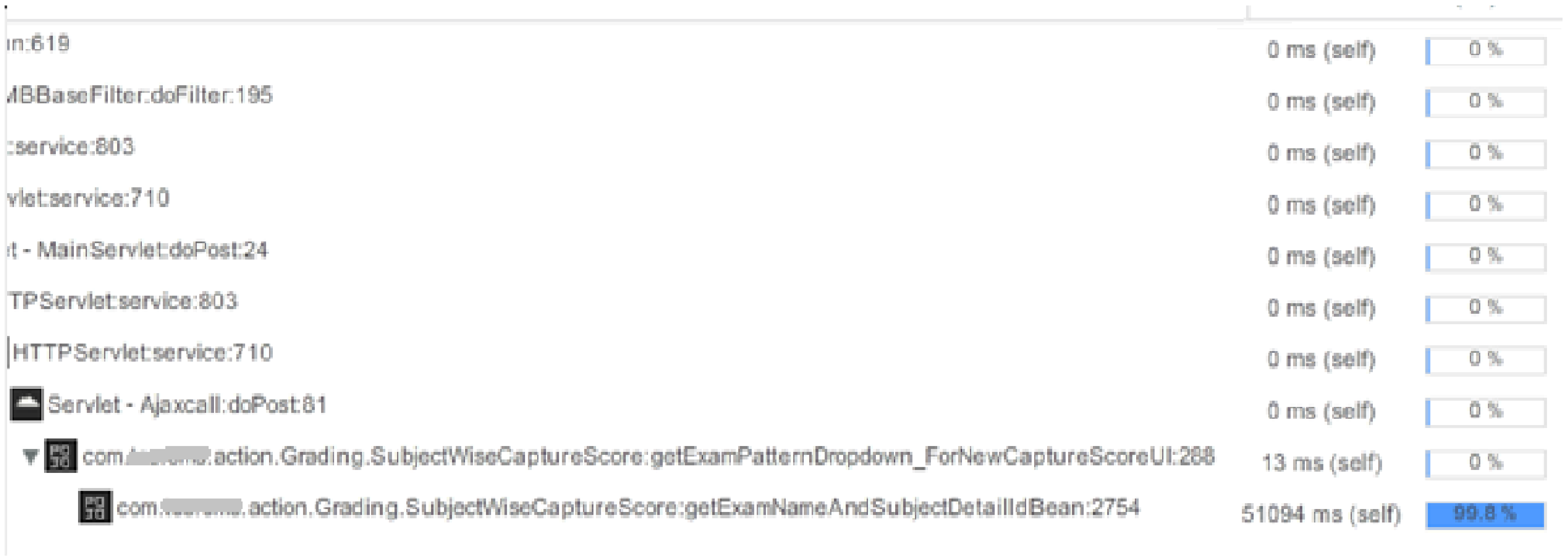
Normal	🟩	94.4%	276.0k
Slow		4.0%	11.9k
Very Slow		1.5%	4.5k
Stalls		0.0%	0
Errors		0.3%	861

All the internals opened up!! X-Ray into the JVM!



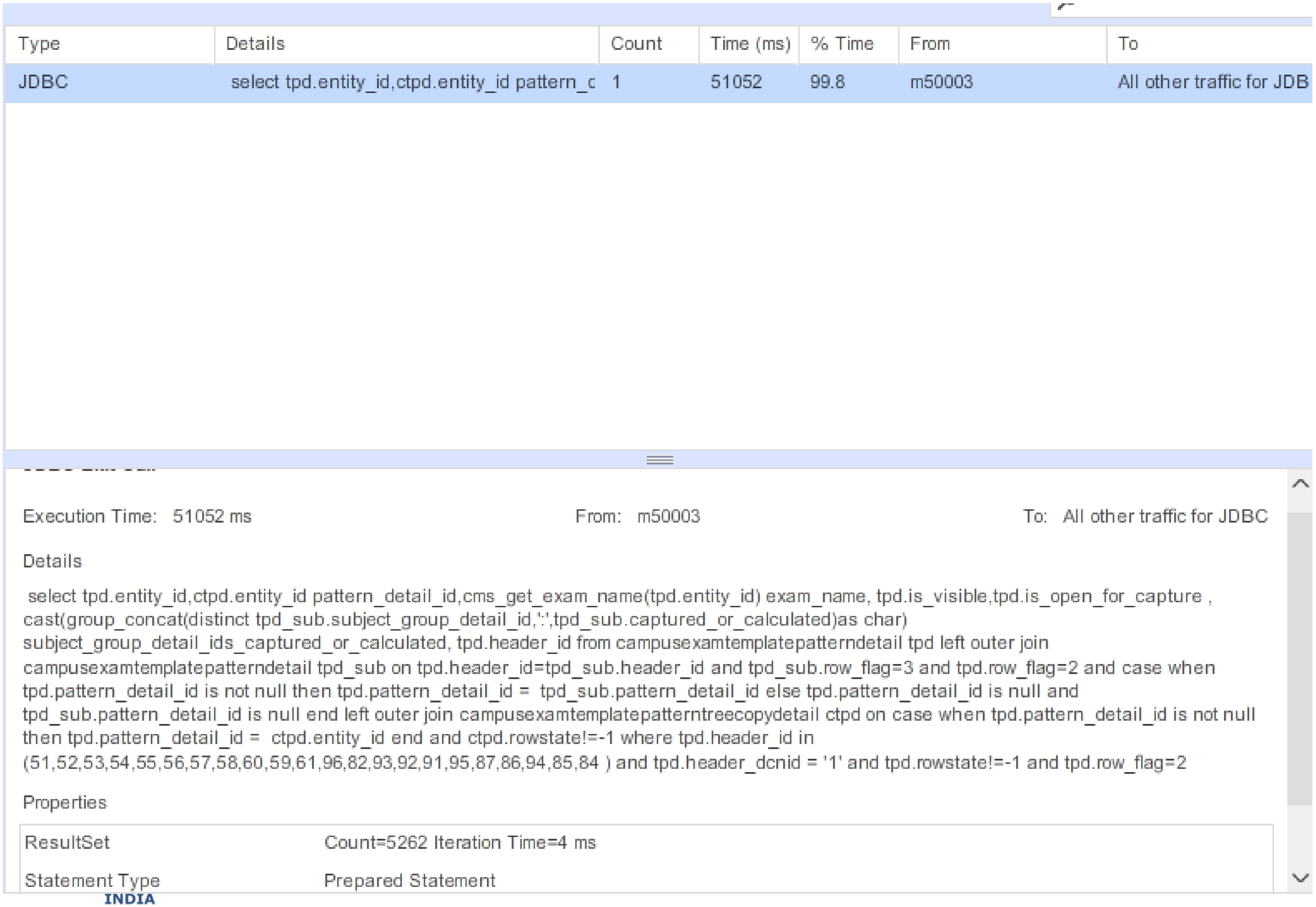
Name	Time (ms)	
blackboard.webapps.blackboard.nautilus.view.WhatsNewViewHelper:<init>:72	3744 ms (total)	82 %
blackboard.webapps.blackboard.nautilus.view.WhatsNewViewHelper:initDiscussions:100	2904 ms (total)	63.6 %
blackboard.data.discussionboard.datamanager.impl.DiscussionBoardUnreadCountsManagerImpl:getDI	2904 ms (total)	63.6 %
blackboard.data.user.User:getSystemRole:568	82 ms (total)	1.8 %
blackboard.platform.security.SecurityUtil:checkPermission:61	82 ms (total)	1.8 %
blackboard.platform.security.BbPolicy:getPermissions:151	82 ms (total)	1.8 %
blackboard.data.discussionboard.datamanager.impl.DiscussionBoardUnreadCountsManagerImpl:ge	2822 ms (total)	61.8 %
blackboard.data.discussionboard.datamanager.impl.DiscussionBoardUnreadCountsManagerImp	612 ms (total)	13.4 %
blackboard.persist.impl.Query:run:249	612 ms (total)	13.4 %
blackboard.persist.impl.Query:run:264	612 ms (total)	13.4 %
blackboard.persist.BbPersistenceManager:runDbQuery:590	612 ms (total)	13.4 %
blackboard.persist.BbPersistenceManager$1:run:604	612 ms (total)	13.4 %
blackboard.persist.impl.Query:executeQuery:130	612 ms (total)	13.4 %
blackboard.persist.impl.SelectQuery:doExecute:160	612 ms (total)	13.4 %
blackboard.db.ConnectionManager$PreparedStatementProxy:invoke:15	612 ms (total)	13.4 %
blackboard.data.discussionboard.datamanager.impl.DiscussionBoardUnreadCountsManagerImp	2210 ms (total)	48.4 %
blackboard.persist.impl.Query:run:249	2210 ms (total)	48.4 %
blackboard.persist.impl.Query:run:264	2210 ms (total)	48.4 %
blackboard.persist.BbPersistenceManager:runDbQuery:590	2210 ms (total)	48.4 %
blackboard.persist.BbPersistenceManager$1:run:604	2210 ms (total)	48.4 %
blackboard.persist.impl.Query:executeQuery:130	2210 ms (total)	48.4 %
blackboard.persist.impl.SelectQuery:doExecute:160	2210 ms (total)	48.4 %
blackboard.db.ConnectionManager$PreparedStatementProxy:invoke:15	2210 ms (total)	48.4 %
blackboard.webapps.blackboard.nautilus.view.WhatsNewViewHelper:initBlogs:118	633 ms (total)	13.9 %

Drill down to the problem



```
ın:619                                                                                     0 ms (self)     0 %
ИBBaseFilter:doFilter:195                                                                  0 ms (self)     0 %
:service:803                                                                               0 ms (self)     0 %
vlet:service:710                                                                           0 ms (self)     0 %
ıt - MainServlet:doPost:24                                                                 0 ms (self)     0 %
TPServlet:service:803                                                                      0 ms (self)     0 %
|HTTPServlet:service:710                                                                   0 ms (self)     0 %
 Servlet - Ajaxcall:doPost:81                                                              0 ms (self)     0 %
 ▼ com.[redacted].action.Grading.SubjectWiseCaptureScore:getExamPatternDropdown_ForNewCaptureScoreUI:288     13 ms (self)     0 %
     com.[redacted].action.Grading.SubjectWiseCaptureScore:getExamNameAndSubjectDetailIdBean:2754      51094 ms (self)   99.8 %
```

See the actual problem



Type	Details	Count	Time (ms)	% Time	From	To
JDBC	select tpd.entity_id,ctpd.entity_id pattern_c	1	51052	99.8	m50003	All other traffic for JDB

Execution Time: 51052 ms   From: m50003   To: All other traffic for JDBC

Details

select tpd.entity_id,ctpd.entity_id pattern_detail_id,cms_get_exam_name(tpd.entity_id) exam_name, tpd.is_visible,tpd.is_open_for_capture , cast(group_concat(distinct tpd_sub.subject_group_detail_id,':',tpd_sub.captured_or_calculated)as char) subject_group_detail_ids_captured_or_calculated, tpd.header_id from campusexamtemplatepatterndetail tpd left outer join campusexamtemplatepatterndetail tpd_sub on tpd.header_id=tpd_sub.header_id and tpd_sub.row_flag=3 and tpd.row_flag=2 and case when tpd.pattern_detail_id is not null then tpd.pattern_detail_id = tpd_sub.pattern_detail_id else tpd.pattern_detail_id is null and tpd_sub.pattern_detail_id is null end left outer join campusexamtemplatepatterntreecopydetail ctpd on case when tpd.pattern_detail_id is not null then tpd.pattern_detail_id = ctpd.entity_id end and ctpd.rowstate!=-1 where tpd.header_id in (51,52,53,54,55,56,57,58,60,59,61,96,82,93,92,91,95,87,86,94,85,84) and tpd.header_dcnid = '1' and tpd.rowstate!=-1 and tpd.row_flag=2

Properties

ResultSet	Count=5262 Iteration Time=4 ms
Statement Type	Prepared Statement

INDIA

Do we really need it?

- YAGNI
  - You Aren’t Going to Need It
- YAGNI
  - You Are Going to Need It
- Question: Do we really need it?
- Answer: DEPENDS ☺
- The quintessential answer of Architects

Be Pragmatic not Pedantic when deciding

References

- https://gist.github.com/jboner/2841832
- http://www.oracle.com/technetwork/java/javase/gc-tuning-6-140523.html
- http://publib.boulder.ibm.com/infocenter/javasdk/tools/index.jsp?topic=%2Fcom.ibm.java.doc.igaa%2F_1vg000139b8b453-11951f1e7ff-8000_1001.html
- http://architects.dzone.com/articles/5-tips-proper-java-heap-size
- Sundarraj Kaushik: 3rd CMGIndia Mumbai Event, April 2014, http://www.cmgindia.org/wp-content/uploads/2014/04/SundarrajKaushik-SaaS-Performance-Management.ppt
- http://www.eecs.berkeley.edu/~rcs/research/interactive_latency.html

Numbers Every Programmer Should Know

Activity	Time in ns	L1 Ratio	Ratio with Previous
L1 cache reference	0.5	1	-
Branch mispredict	5.0	10	10.00
L2 cache reference	7.0	14	1.40
Mutex lock/unlock	25.0	50	3.57
Main memory reference	100.0	200	4.00
Compress 1K bytes with Zippy	3000.0	6,000	30.00
Send 1K bytes over 1 Gbps network	10000.0	20,000	3.33
Read 4K randomly from SSD*	150000.0	300,000	15.00
Read 1 MB sequentially from memory	250000.0	500,000	1.67
Round trip within same datacenter	500000.0	1,000,000	2.00
Read 1 MB sequentially from SSD*	1000000.0	2,000,000	2.00
Disk seek	10000000.0	20,000,000	10.00
Read 1 MB sequentially from disk	20000000.0	40,000,000	2.00
Send packet CA->Netherlands->CA	150000000.0	300,000,000	7.50

Source: https://gist.github.com/jboner/2841832

QUESTIONS?