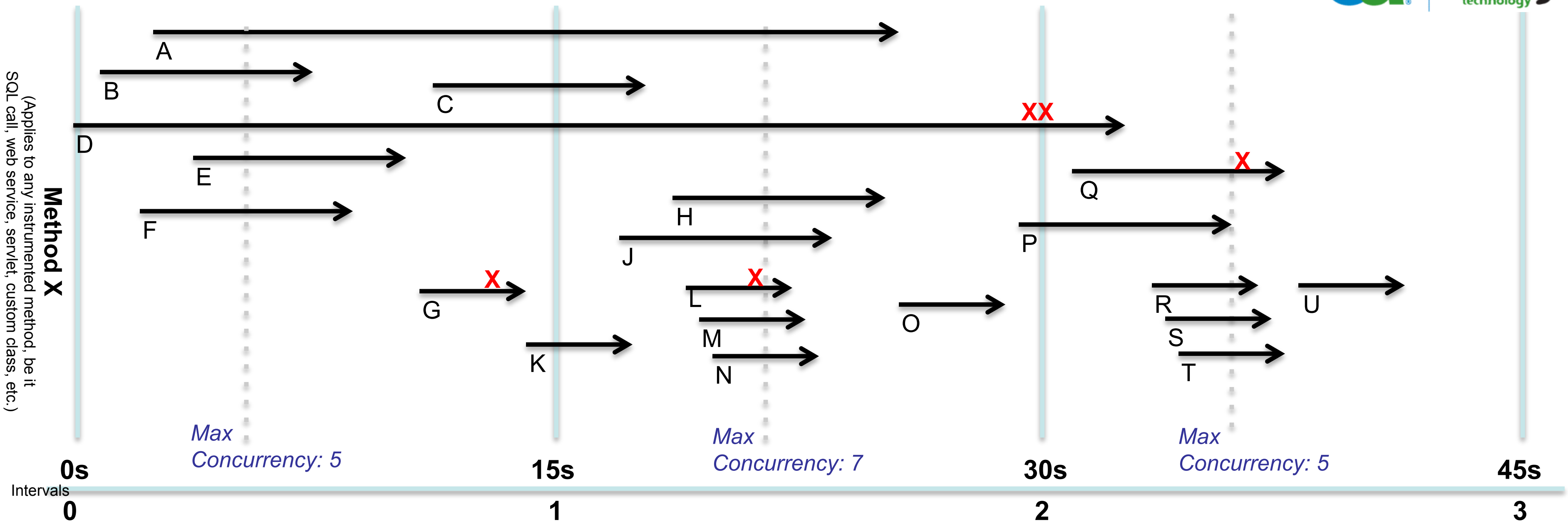
Quick Reference to Primary Introscope Metrics



Average Response Time (ms)

Interval 1: <small> B,E,F,G
Interval 2: <longer> A,C,H,J,K,L,M,N,O
Interval 3: <even longer> P,Q,R,S,T,U

Value is average of all finished invocations of a method or component. Zero usually indicates no invocations during that interval. *Count* is number of transactions finished that interval. *Min* and *Max* are fastest and slowest measurements respectively.

Errors Per Interval (X)

Interval 1: 1 – G
Interval 2: 2 – L,D
Interval 3: 1 – Q

Any exception caught in the stack will be reported and a snapshot gathered (kept 14 days).

Responses Per Interval

Interval 1: 4 - B,E,F,G
Interval 2: 9 - A,C,H,J,K,L,M,N,O
Interval 3: 7 – D,P,Q,R,S,T,U

Value reflects number of invocations finished in that interval. *Min*, *Max*, and *Count* all agree with value. When querying historical data to make comparisons, be careful to account for changes in interval units.

Count of Average Response Time is identical to Responses Per Interval.

Stall Count (XX)

Interval 2: 1 – D

When methods take too long (30 sec by default), they indicate a stuck thread, usually due to infinite loop, deadlock, or constrained resources. Snapshots are gathered (kept 14 days).

Concurrent Invocations

Interval 1: 4 - A,C,D,K (max: 5 – A,B,D,E,F)
Interval 2: 2 – D,P (max: 7 – A,D,H,J,L,M,N)
Interval 3: 0 - <none> (max: 5 – Q,P,R,S,T)

Min is the minimum number of threads in a method or component over the interval. *Max* is the peak number of threads. *Value* is the final sampling of how many threads were in the method at the end of the interval. *Count* is the total of entries and exits to the method.

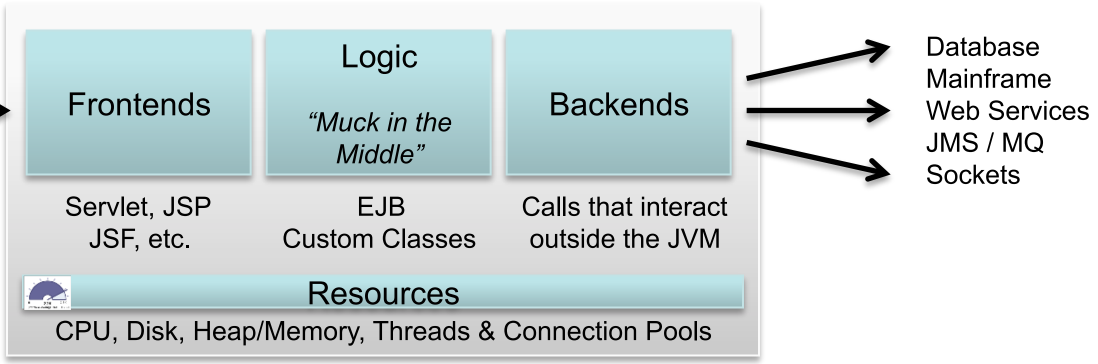
AKA Work in Process. When more work comes in than is being completed, this increases, indicating the "Pig-in-a-Python" analogy. If it spikes and then returns, this indicates a bottleneck (perhaps due to load) that was temporary.

Questions?
Chris.Kline@ca.com

Quick Reference to Java Metrics

Most modern applications use a 3-tier architecture. Requests enter a frontend, are then processed by the logic tier (AKA “the muck in the middle,” and in turn handled by calls to various backend systems.

Introscope watches ALL calls to instrumented methods and reports data every 15 seconds. Metrics are reported with 4 data points each: the min, max, count, and value of the type observed.



Frontends

Servlet, JSP
JSF, etc.

Logic

“Muck in the Middle”

EJB
Custom Classes

Backends

Calls that interact
outside the JVM

Database
Mainframe
Web Services
JMS / MQ
Sockets

Resources

CPU, Disk, Heap/Memory, Threads & Connection Pools

Triage Technique

Performance problems are effectively triaged by first looking at frontend metrics to diagnose slowness. Frontends are not normally the cause of application slowness. Then move to the backend to see if a corresponding slowdown can be located. If so, then don’t bother looking at the Logic tier, as the problem likely lies in the backend. If the backends are performing properly, however, then move to the Logic tier to triage issues. The “muck in the middle” likely has the more-challenging components; look there last.

Frontends

/url-root – metrics describing performance of all requests for that URL root.

Servlets – common application dispatcher. Good proxy for evaluating all application transactions.

JSPs - common user interface display component.

Logic and Other APIs

Struts– SQL statements used to query and update a relational database.

Session EJBs – “conversation-driven” bits of business logic.

Entity EJBs – “persistence-focused” bits of business logic

Message-driven EJBs – “message backed” bits of business logic

JNDI - Java’ s Naming and Directory Interface. Often represents a distributed lookup, and can be a source of latency.

JavaMail - Sending and receiving e-mail

XML – Often a source of significant CPU usage and latency. Data-driven parsing, analysis, and processing.

Backends

JDBC– SQL statements used to query and update a relational database.

Web Services – HTTP-wrapped calls to external services.

Sockets – Performance stats associated with access to a remote machine.

JMS - Java Messaging Service. Backed by a JMS queue, for example WebSphere MQ.

BlamePoint Metrics

Most instrumented components will show 5 “BlamePoint” metrics (SEE REVERSE): including response time, invocation rates, and error-processing. Understanding these 5 metrics unlocks the triage process.

JMX or WebSphere PMI

Metrics produced by an application server or directly by an app itself, and recorded by Wily. For exact meaning of JMX metrics, consult vendor documentation. Usually contains information about thread pools, sessions, and database connection pools.

Resources

CPU

% utilization (process) – Percentage of total available CPU in use by JVM (e.g. 25% on a 4-CPU machine equals one CPU fully in use.)
% utilization (aggregate) – Percentage of a single processor in use by any process, including but not limited to the JVM.

Memory (GC Heap)

Bytes In Use – Number of bytes currently allocated by the JVM for use by running app. Reclaimed through garbage collection at regular intervals.
Bytes Total - Max number of bytes currently available for allocation to running app by JVM.

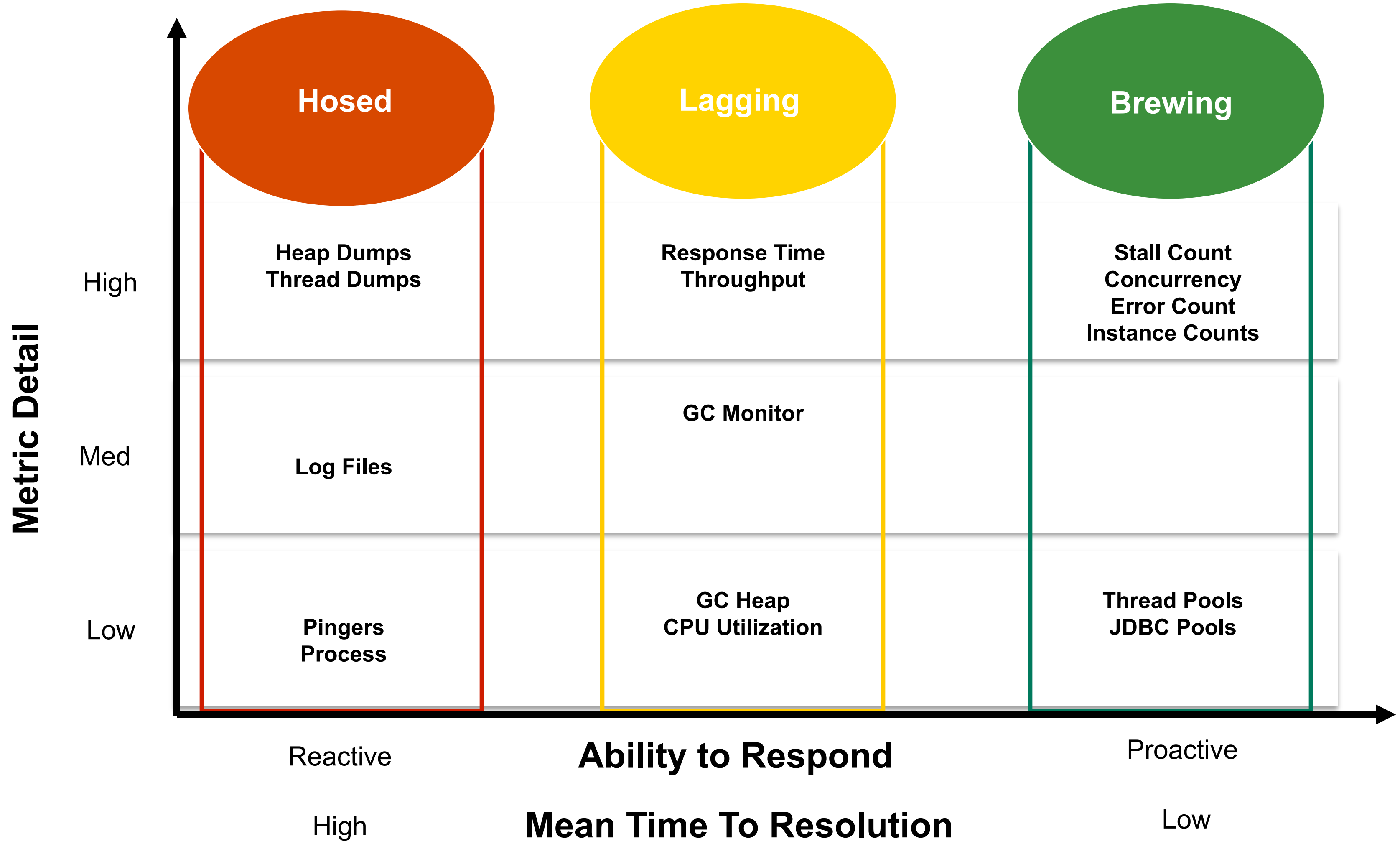
Input / Output

File – bytes per second read or written to a file by the running app. Count reflects total number of bytes written during that interval.
Socket – incoming requests accepted, opened, and closed during that interval. Threads concurrently reading or writing to the socket during that interval.

Threads

Active Threads – number of threads currently live in the JVM. Threads may not be active.

Interpreting Performance Metrics (Production)



Metric Detail	Hosed	Lagging	Brewing
High	Heap Dumps Thread Dumps	Response Time Throughput	Stall Count Concurrency Error Count Instance Counts
Med	Log Files	GC Monitor	
Low	Pingers Process	GC Heap CPU Utilization	Thread Pools JDBC Pools