Progress®

Profiling and Monitoring Your Application in PAS for OpenEdge

Beyond The Code Series

Peter Judge

pjudge@progress.com

Available Monitoring Tools

Customer Case Studies

Establishing Patterns

Gathering Metrics via OEManager

Putting Results to Use

Available Monitoring Tools

JMX and OEManager WebApp REST API's

- Available in 11.7.4 and later
- Will be our primary focus today
- Same JSON output, just different means of access
  - JMX: Command line, queries the Tomcat instance's JAVA process
  - REST: RESTful requests, queries oemanager webapp of instance
- Some queries work at an agent level, others at a session level
  - Some data can be reported at either level (eg. all sessions for an agent)
- Can be automated for regular polling of metrics
  - Via bat/sh script (JMX) or OEHttpClient classes (REST)
  - Both options can be run any time, but may report false-positives (more later)

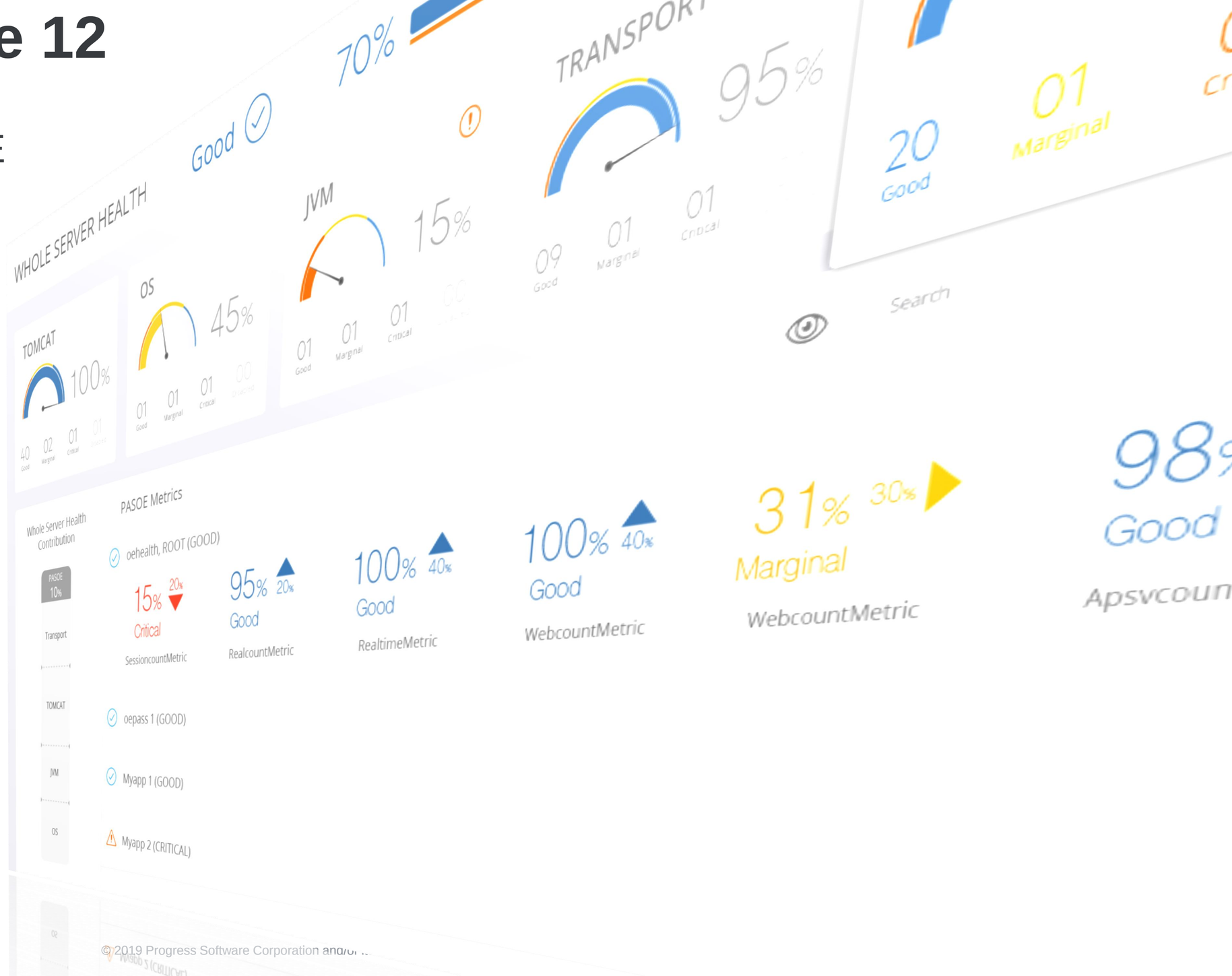
New in OpenEdge 12

HealthScanner for PASOE

- Snapshot of server health
- Uses RESTful API’s
- Non-intrusive (no code changes)
- Not meant for code diagnostics
- Useful for cloud applications

Server-Side Profiling

- Uses a JMX query to trigger data collection for X requests
- Sends data to a special WebApp on separate PAS for OE instance
- Comes with pre-defined “oediagdb” schema for data persistence

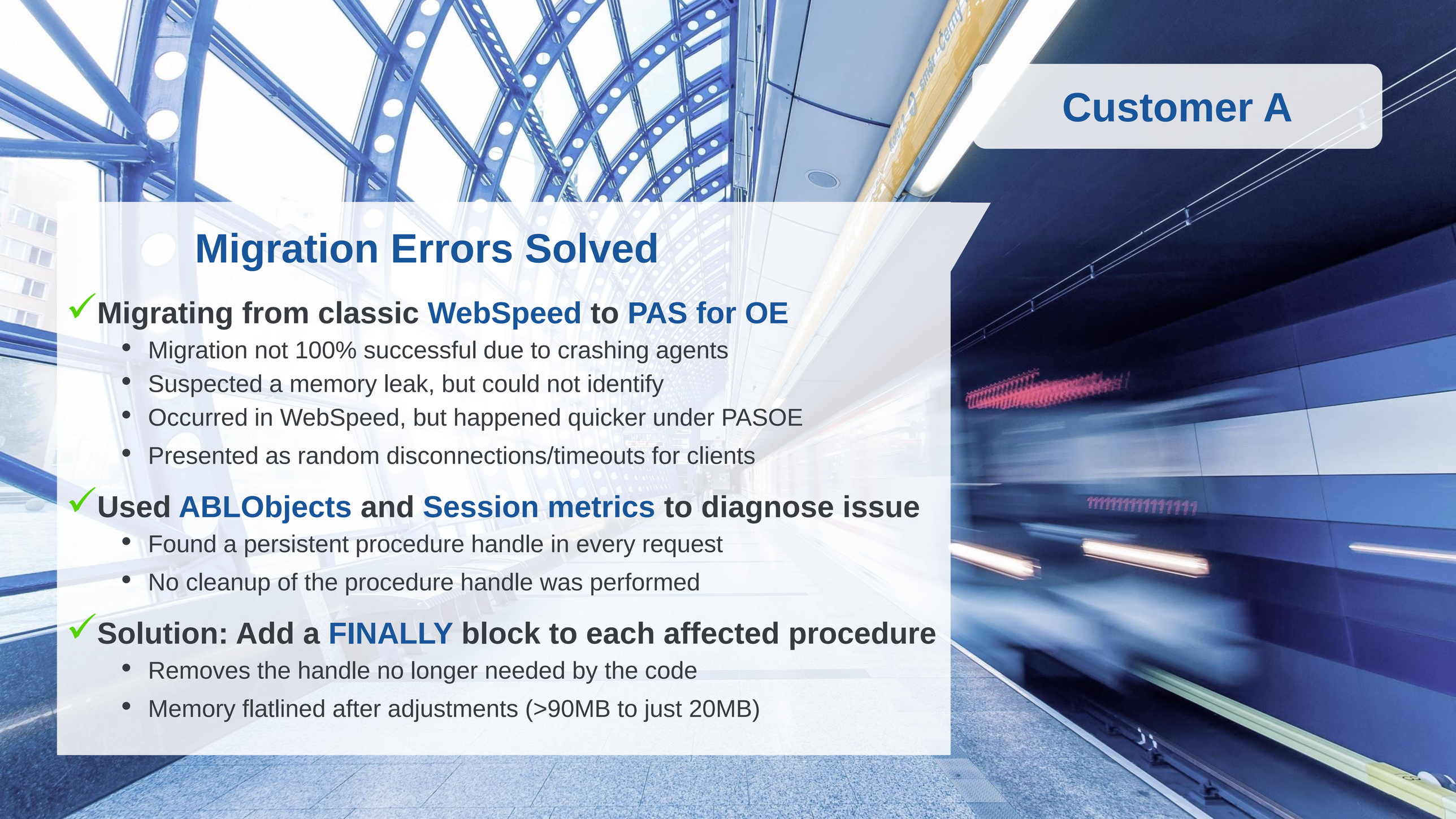


Progress®

Customer Case Studies

Overview

- Review 3 recent, successful engagements
- Names are withheld to protect the innocent
- Meant to examine honest mistakes
- Walkthrough of our processes
- Examine mitigation strategies
- Proves our troubleshooting techniques work!

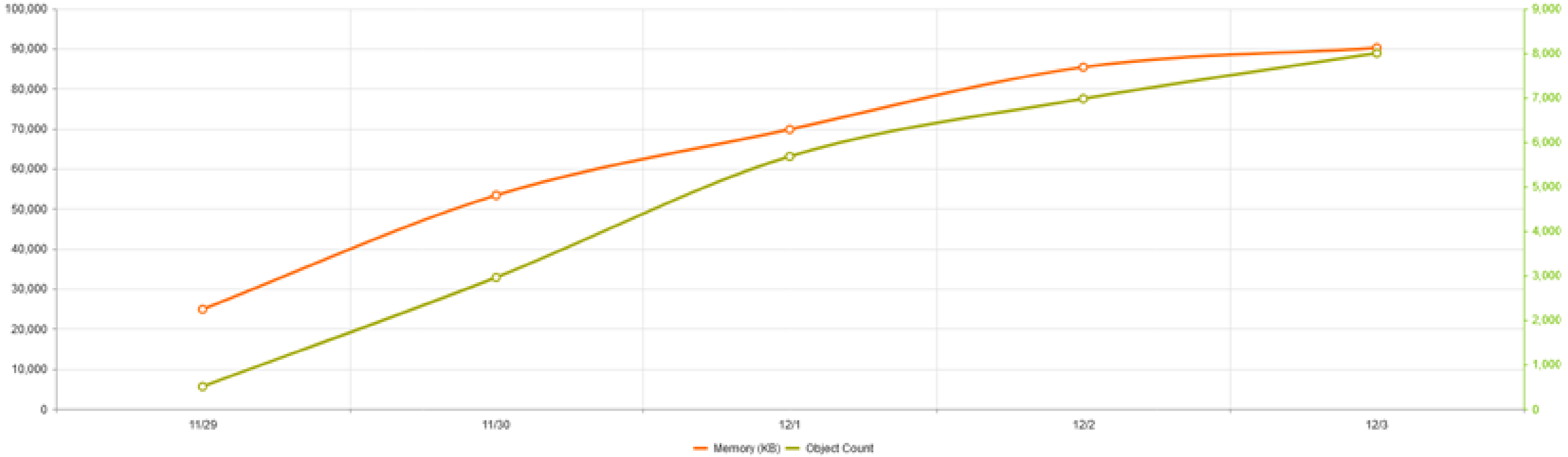


Customer A

Migration Errors Solved

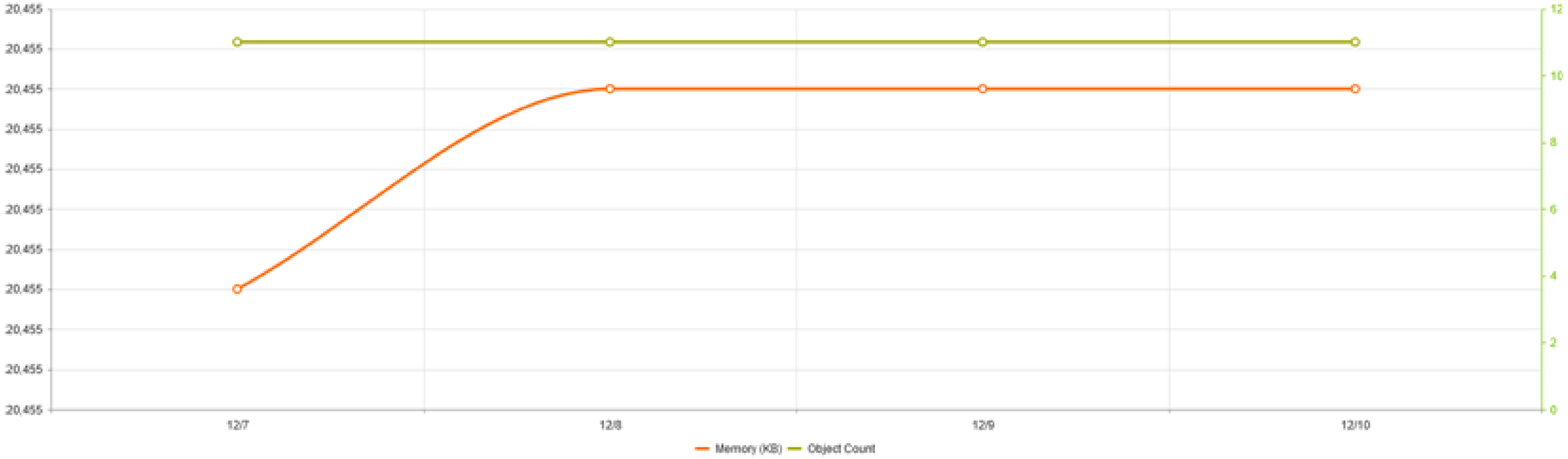
- ✓ **Migrating from classic WebSpeed to PAS for OE**
  - Migration not 100% successful due to crashing agents
  - Suspected a memory leak, but could not identify
  - Occurred in WebSpeed, but happened quicker under PASOE
  - Presented as random disconnections/timeouts for clients
- ✓ **Used ABLObjects and Session metrics to diagnose issue**
  - Found a persistent procedure handle in every request
  - No cleanup of the procedure handle was performed
- ✓ **Solution: Add a FINALLY block to each affected procedure**
  - Removes the handle no longer needed by the code
  - Memory flatlined after adjustments (>90MB to just 20MB)

Memory Usage Over 5 Days (Original Code)



Both memory and objects continued a steady climb.

Memory Usage Over 4 Days (After Code Fix)



Objects maintained at 11, memory reached a steady state.

Customer B

Memory Issues Solved

✓ **Migrated from Classic AppServer to PAS for OE**

- Reported a potential memory leak in their PAS for OE application
- Used the **ABLObjects** report to identify a growing count of artifacts
- *Specifically used API's from 11.7.3 (some URL's changed in 11.7.4)

✓ **Found numerous items for correction**

- Passing of temp-tables without by-reference
- Didn't clean up object instances after use
- Didn't release record buffers after creation

✓ **All but 1 elusive item remained…**

- Tracked to a table handle which was no longer needed
- Fixing just that remaining item was a huge difference!

Before/After Comparison of ABLObjects

		Number of ABLObjects		
		Session1	Session2	Session3
1)	Restarted PASOE	0	0	0
2)	Hit login page	0	0	38
3)	Logged in	42	35	47
4)	Open Alert Maintenance Screen	50	42	49
5)	Run Query	50	42	51
6)	Run Query again	50	42	51
7)	Close and re-open Alert Maint	56	44	57
8)	Run Query	56	46	57
9)	Run Query Again	56	46	57

		Number of ABLObjects		
		Session1	Session2	Session3
1)	Restarted PASOE	0	0	0
2)	Hit login page	0	0	36
3)	Logged in	34	34	41
4)	Open Alert Maintenance Screen	34	34	41
5)	Run Query	34	34	41
6)	Run Query again	34	34	41
7)	Close and re-open Alert Maint	34	34	41
8)	Run Query	34	34	41
9)	Run Query Again	34	34	41

*Remaining objects were identified as integral to operation.

Customer C

Performance Issues Solved

- ✓ **Migrated from Classic AppServer to PAS for OE**
- ✓ **Used distributed / load-balanced AWS environment**
- ✓ **Isolated an inconsistency in client response times**
- ✓ **Adjusted numerous items for improved performance**
  - Certain EC2 classes have better networking capabilities
  - Increased –Mm to maximum value: noticeable improvement
  - Found that Availability Zone (AZ) latency between DB and PAS for OE server instances impacted performance
- ✓ **Select API requests still had an unfound inconsistency**

Customer C: When the Problem isn't the Code

- Used "Requests" metric to compare ABL execution time
  - Discovered ABL execution time was consistent
  - Tomcat access logs were similarly consistent
  - Available PAS for OE connections were under-utilized
- Customer was using some ASP.Net code as middleware
  - Found overhead in IIS logs when handling requests!
- .Net Framework has a default "threads per processor core" value
  - Causing a bottleneck for request processing at the web server
  - Changing to the maximum value improved requests dramatically
  - Throughput from IIS properly saturated the PAS for OE connections

Progress®

Establishing Patterns

Repeat, Repeat, Repeat

- ✓ You need a scripted test for consistency (JMeter, SoapUI, etc.)
- ✓ Stress/use the system in a realistic way to reproduce a problem
- ✓ Process as a baseline for measuring change
- ✓ Isolate and control the variables involved
- ✓ Change, measure, evaluate, repeat

Customer Processes

- Worked with customers to craft a path through their application
  - Basic, Expert, and Admin scenarios
- Ran tests for extended periods to gather metrics
  - Some tests ran for a week to get necessary data
- Processed the metrics to visualize results
- Identified potential code for further review
- Modified, compiled, and deployed changes
- Re-ran tests using previous test pattern(s)
- You can do all this, too!

Progress®

Gathering Metrics via OEManager REST API

Useful Metrics with a Purpose

- **Agents**: Report of all agents for an ABL Application name
- **Sessions**: Provides information about each Agent session
  - Shows the session #, memory, current state, start time, and end time
- **ABLObjects**: objects, buffers, procedures, & handles in memory
  - Similar to https://knowledgebase.progress.com/articles/Article/P124514
- **Requests**: Track internal ABL requests vs. Tomcat access log
  - Reveals inconsistencies between overall request vs. code execution time
  - Shows end-to-end request chain from web to ABL runtime (OE12+)

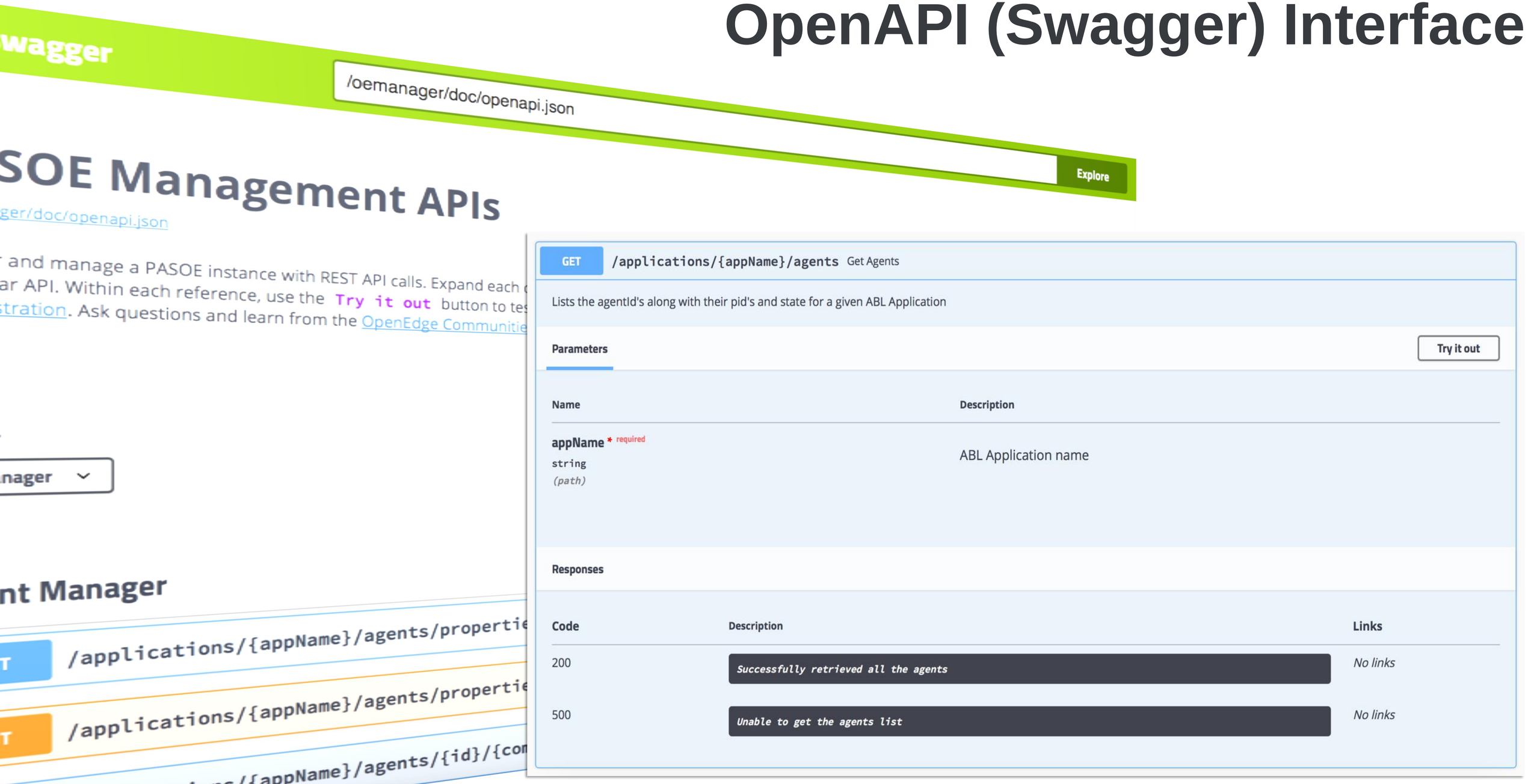
Why a Preference for REST API?

- No command-line or administrator access necessary
  - JMX only available when PASOE is started via “tcman”
- Must first create a query file on disk with the correct parameter values
  - {"O":"PASOE:type=OEManager, name=AgentManager","M":["getAgents", "**SportsPASOE**"]}
  - Compare to **GET /oemanager/applications/SportsPASOE/agents**
- No need for OS-COMMAND() calls + reading of output files
  - Output already returned as JSON format for parsing
  - **oejmx.bat -R -Q jmxqueries/agents.qry** → JSON File
- Can be run at request boundaries within an application
  - Translation: allows you to gather metrics after any FINALLY blocks
  - For ABLObjects, avoids false-positives due to legitimate items in-flight

Additional REST API Benefits

- Easily accessed programmatically via ABL code (OEHttpClient)
  - During session startup/shutdown or activate/deactivate event pairs
- Typically present in a development PAS for OE instance
  - Included with “-f ” option to “pasman create”
  - For Production: tcman deploy $DLC/servers/pasoe/extras/oemanager.war
- 11.7.4+ offers an OpenAPI (Swagger) interface for easy integration
  - Evaluate code during your PAS for OE migration, before moving to OE12
  - Can run at any time (minding the caveat about request boundaries)
  - Disabled by default for security, but we’ll walk through the process

OpenAPI (Swagger) Interface



Enabling OpenAPI (Swagger)

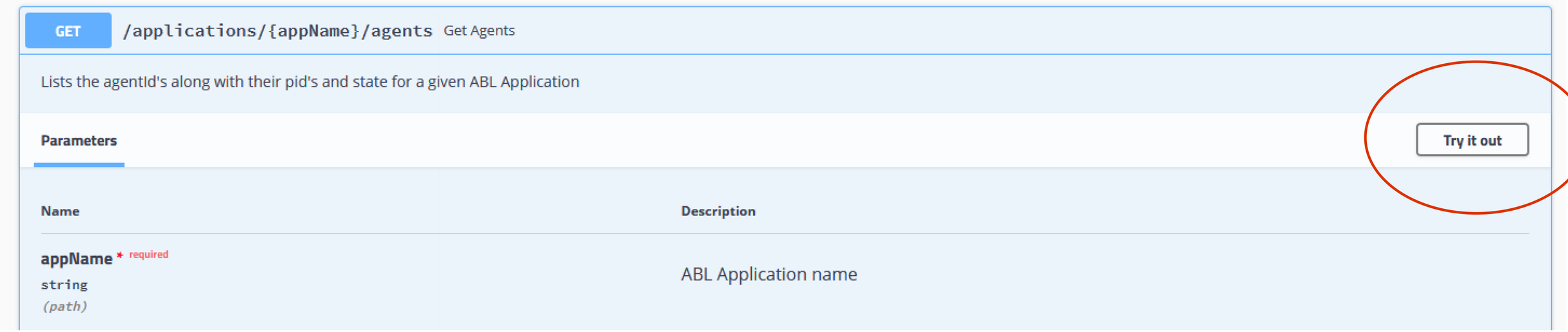
- Navigate to **CATALINA_BASE/webapps/oemanager/WEB-INF/**
- Open **oemgrSecurity-container.xml** in a text editor
- Edit the following section as described in the comments:

```
<!-- Access to SwaggerUI. Disabled by default, user has to uncomment the below line to enable it -->
<intercept-url pattern="/doc/**" method="GET" access="hasAnyRole('ROLE_PSCAdmin','ROLE_PSCOper','ROLE_PSCUser')"/>
```

- Save and restart your PASOE instance (since we altered security)
- Visit http[s]://<hostname>:<port>/oemanager/ (note trailing slash)
  - Default username/password is tomcat/tomcat
  - The “container” security inherits from Tomcat itself

Using the OpenAPI (Swagger) Interface

- Click on an item to expand it, then click the “Try it out” button
- Fill in the parameter fields available, click ”Execute”
  - Note: the default ABL Application name is the PASOE instance name
- View the “Responses” area for output (next slide)



Obtaining Agent Information

- Most API's require an **AgentID** available from the following URL:
  - GET /oemanager/application/<abl_app_name>/agents
- Response should contain "**agents**" array with "**agentId**" values

Server response

Code	Details
200	**Response body**


```
{
  "result": {
    "agents": [
      {
        "agentId": "tHEgoLy6SqyyrupC5Gxfdg",
        "pid": "6512",
        "state": "AVAILABLE"
      }
    ]
  },
  "versionStr": "v11.7.4 ( 2018-10-10 )",
  "versionNo": 1,
  "errmsg": "",
  "outcome": "SUCCESS",
  "operation": "GET AGENTS"
}
```


Download

Obtaining Session Information

- Session information requires an **ABL App Name** and **AgentID**
  - GET /oemanager/applications/<abl_app_name>/agents/<agent_id>/sessions
- Results include a “**SessionMemory**” property, in bytes


```
{
  "result": {
    "AgentSession": [
      {
        "SessionId": 4,
        "SessionState": "IDLE",
        "StartTime": "2019-04-12T11:08:26.007",
        "EndTime": null,
        "ThreadId": -1,
        "ConnectionId": null,
        "SessionExternalState": 0,
        "SessionMemory": 54724492
      },
      {
        "SessionId": 6,
        "SessionState": "IDLE",
        "StartTime": "2019-04-12T11:08:26.007",
        "EndTime": null,
        "ThreadId": -1,
        "ConnectionId": null,
        "SessionExternalState": 0,
        "SessionMemory": 54724492
      }
    ]
  },
```

Obtaining the ABLObjects Report

- Must first enable tracking before running code, obtaining report:
  - PUT {“enable”:”true”} to /oemanager/applications/<abl_app_name/agents/<agent_id>/ABLObjects/status
  - GET /oemanager/applications/<abl_app_name/agents/<agent_id>/ABLObjects
- Regardless of URL, the JSON content should be consistent
  - JSON Path: result.ABLOutput.ABLObjects[{obj1} , … , {objN}]
  - Each child object contains an “AgentSessionId” (number) and “Objects” array

```
{
 "result": {
  "ABLOutput": {
   "ABLObjects": [ {
     "AgentSessionId": <session_id>,
     "Objects": [ … ]
   } …
```

Obtaining Request Information

- Must run the endpoint API first to enable, then again to report
  - GET /oemanager/applications/<abl_app_name>/agents/<agent_id>/requests
- Returns the last 1000 requests to an ABL Application name
  - Currently limit imposed for performance which may or may not be changed
- Result contains “**AgentRequest**” array of JSON objects containing:
  - Request Procedure Name (could also be Class Method)
  - Session ID
  - Start Time and End Time
  - Sequential Request Number
  - For OE12: Client ID matching the enhanced Tomcat access log output

Progress®

Putting Results to Use

Regarding the ABL Objects

- Review output and investigate any often-repeated items
  - Items not properly removed should noticeably compound over time
- Add FINALLY blocks to help with clean up
  - Remember: These also run in the event of an error!
- Improve your code by assisting the Garbage Collection process
  - If you define it, delete it
  - If you create it, release it
  - If you open it, close it
  - Pass by-reference when/if possible
  - “Try it until it doesn’t work”

Regarding Session Memory

- Determine if session memory keeps rising or reaches a plateau
- Identify an average high-water mark for your sessions
  - This is very application-specific and will not be a one-size value
- OOM Check: grep -i 'killed process' /var/log/messages
- Size your application appropriately for OS resources
  - Calculate the **Agent** memory (Avg. Session Bytes x Connections)
  - Calculate total **ABL Application** memory (**Agent** memory x Max Agents)
  - Get total **PASOE Instance** memory from sum of all ABL Applications
  - Calculate total expected memory from sum of all **PASOE Instances**
  - Add expected OS and operational memory, plus growth overhead

Regarding Request Information

- Much easier to parse in OE12 thanks to enhanced logging
  - Tomcat access log already contains timestamp with milliseconds
  - Tomcat access log already contains a unique “OE Request ID”
  - Request metric contains a ClientID (same as OE Request ID)
  - Provides a 1:1 match between the web server and ABL runtime
- Compare to Tomcat access
  - May need to add a %D token to the log pattern (elapsed time in ms)
- Obtain elapsed ABL execution from End Time – Start Time (ms)
- Compare elapsed times for unexpected overhead
  - Overhead (ms) = Tomcat elapsed time - ABL elapsed time

Lessons Learned

- Small Leaks + Time = Big Problems
- Memory consumption matters for PASOE stability
- Beware: OOM Killer (Linux) and Swap Disk (Windows)
- Understand trends to predict future growth
- Prevent overcommitting of system resources
- Isolate application layers and their true behavior
- Tests best done in non-production environments

What's Next?

- What about automating metrics collection?
- See the automation guide here:
  - "Automation with Spark Diagnostics" at https://bit.ly/2IxrScN
- Once collection is automated, just let it go!
  - Run on a schedule (explicit start/end dates and times)
  - Report at defined interval (eg. every hour or every 2 days)
- Parse results easily by reading output files on disk
  - Create visualization (charts/graphs) from data points
  - Review content for suspicious data

Questions?

pjudge@progress.com


Progress®