

Memory Management:

Garbage Collection, Memory Leaks & Analysis

GUSTAS MICKUS | g.mickus@balticamadeus.com

PUG CHALLENGE 2026 / AMSTERDAM

baltic amadeus

baltic amadeus

Congratulations!
You made to it to the end.

baltic
amadeus

Who are we?

About balticamadeus

We provide IT and strategic consulting services, including Progress OpenEdge, Omnichannel, Self-service, Marketplace, Information security, Enterprise architecture,
Cloud solutions, and more.

96% of clients recommend us

500+ local and international clients

40/60 local vs international clients

2000+ projects, including corporations, SME-level companies, and startups

38+ years of experience

300+ IT talents

baltic amadeus

Progress OpenEdge department:

International clients **70%** | Baltic **30%**



Our Team of OpenEdge
±42 & growing



Helping with:
- Support
- Development
- Modernisation
- Team enhancement

OUR VS CODE EXTENSIONS ^_^



ProBro



ProPeek



OpenEdge ABL Formatter

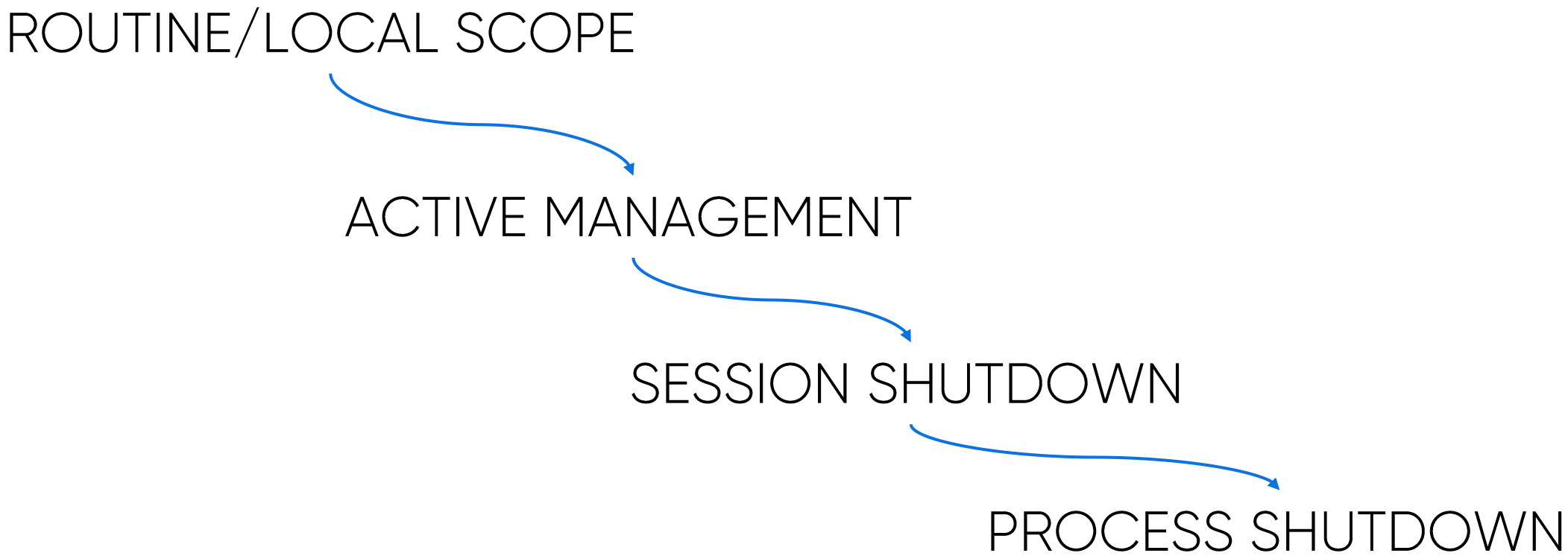
baltic amadeus

Memory Management? Why should I care? 😳

- The PASOE Multi-Session Agent.
  - Classic AppServer is dead
  - Long-lived shared processes
  - Cross-request accumulation
- Containerisation
  - Docker resource limits
  - The Linux Out-Of-Memory (OOM) Killer
  - Uncatchable SIGKILL
- Performance Degradation & Cloud Cost
  - Memory fragmentation
  - HUGE CLOUD BILLS $$$

baltic amadeus

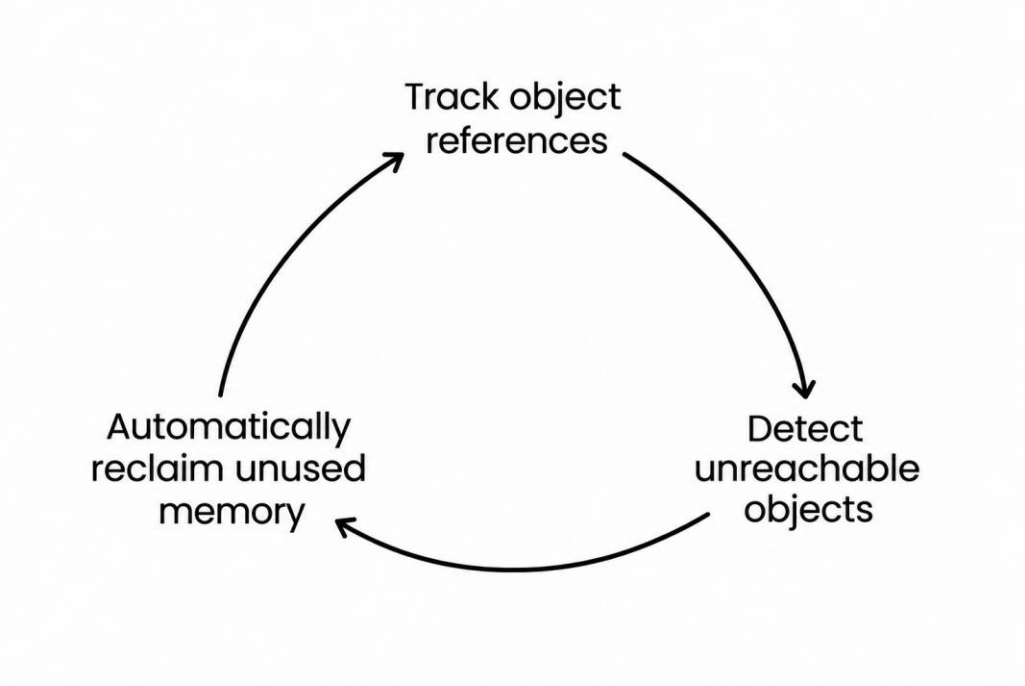
The Four Cycles Of Memory Management



baltic amadeus

Garbage Collection: What is it?

Garbage collection (GC) is a form of automatic memory management that reclaims memory.





baltic amadeus

How the AVM manages memory

```
DEFINE VARIABLE oOrderA AS CustomerOrder NO-UNDO.
DEFINE VARIABLE oOrderB AS CustomerOrder NO-UNDO.

/* Step 1: Instantiation */
/* NEW creates the instance -> RefCount = 1 */
oOrderA = NEW CustomerOrder("ORD-9021").

/* Step 2: Reference Copying */
/* --> RefCount increments to 2 */
oOrderB = oOrderA.

/* Step 3: Dereference: First variable cleared */
/* --> RefCount decrements to 1 (Still active in RAM!) */
oOrderA = ?.

/* Step 4: Final Reference Dropped */
/* --> RefCount drops to 0 */
oOrderB = ?.

/* At RefCount = 0, the AVM adds the object to the destroy list. */
/* DESTRUCTOR runs, memory freed. */
```

baltic amadeus

GC Limitations: Hybrid Environment

OpenEdge employs a hybrid memory model, balancing automated Garbage Collection with developer-led manual management.

AVM manages:

- ✓ System generated error objects
- ✓ User defined object instances
- ✓ .NET objects

```
Progress.Lang.AppError,
System.Windows.Forms
MyClass()
```

YOU 👊 manage:

- × Handle-Based Objects
- × Memory Resources
- × Other Dynamic objects

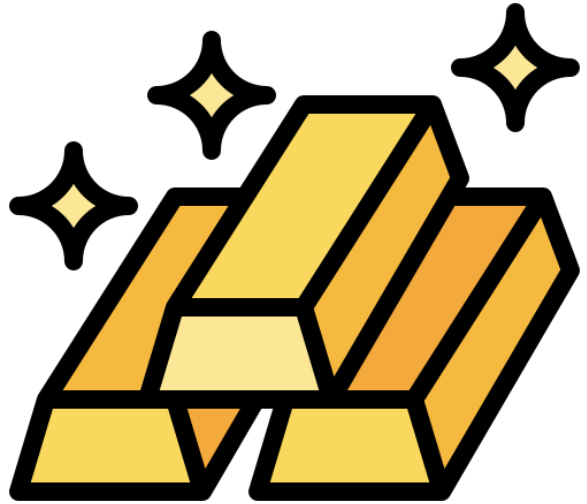
```
Memptr, Buffer, Query, Temp-Table,
Widget, Persistent Procedure.
```

baltic amadeus

The Golden Rule

The GC only manages objects derived from `Progress.Lang.Object`. It doesn't track handle-based resources.

Therefore, **classes are managed, but handles are yours.**



baltic amadeus

Turning Off the GC: -nogc

The **-nogc** startup parameter disables automatic Garbage Collection for the entire session. It shifts all the responsibility to the developer.

Why should I never need to turn off my GC?

- Debugging & isolating memory leaks.
- Object tracking.
- Codebase learning.
- Testing.

baltic amadeus

An ablation study

```
DO i = 1 TO 100000:
    oWorker = NEW WorkerClass(i).
    oWorker:DoSomething().
END.
```

Execution Time: ~49 ms

```
/* Started with -nogc startup parameter */
DO i = 1 TO 100000:
    oWorker = NEW WorkerClass(i).
    oWorker:DoSomething().
    DELETE OBJECT oWorker.
END.
```

Execution Time: ~59 ms

While **-nogc** eliminates background overhead, it often results in a **performance drag** by constantly interrupting the AVM.

baltic amadeus

Special GC Scenarios



TRAP 1: Circular References (Mutual Deadlock)

```
DEFINE VARIABLE oAttendee AS PugAttendee   NO-UNDO.
DEFINE VARIABLE oBike     AS AmsterdamBike NO-UNDO.

/* Step 1: Instantiation */
oAttendee = NEW PugAttendee().    /* Attendee RefCount = 1 */
oBike     = NEW AmsterdamBike().  /* Bike     RefCount = 1 */

/* Step 2: The Mutual Reference */
oAttendee:RentedBike = oBike.     /* Bike     RefCount = 2 */
oBike:CurrentRider   = oAttendee. /* Attendee RefCount = 2 */

/* Step 3: Session ends, attendee drops external pointers */
oAttendee = ?. /* Attendee RefCount drops to 1 (NOT 0!) */
oBike     = ?. /* Bike     RefCount drops to 1 (NOT 0!) */

/* DEADLOCK: 2 Unreachable objects stranded in Amsterdam RAM forever! */
```

baltic amadeus

Special GC Scenarios

TRAP 2: Undoable Pointer

```
/* Notice that NO-UNDO is missing on the variable! */
DEFINE VARIABLE oVipBadge AS ConferenceBadge.

DO TRANSACTION:
    oVipBadge = NEW ConferenceBadge().
    oVipBadge:AttendeeName = "Gustas @ PUG".

    /* Error occurred, discard the badge immediately! */
    oVipBadge = ?.

    /* TRAP: You expect instant Garbage Collection... BUT:                */
    /* Because oVipBadge is undoable, the AVM Transaction Undo Log holds */
    /* a hidden active pointer to restore it if the transaction aborts!  */
    /* The badge is LOCKED in memory until the entire transaction ends!  */
END.
```

baltic amadeus

TRAP 3: The Infinite Stroopwafel Stand

```
CLASS StroopwafelStand:
    /* Internal instance variable pointing to itself */
    DEFINE PRIVATE VARIABLE oSelf AS StroopwafelStand NO-UNDO.

    CONSTRUCTOR PUBLIC StroopwafelStand():
        /* Self-Reference 🤯 */
        oSelf = THIS-OBJECT.
    END CONSTRUCTOR.
END CLASS.
```

```
DEFINE VARIABLE oStand AS StroopwafelStand NO-UNDO.

/* Step 1: Instantiation */
/* RefCount = 2 (1 from outer variable oStand + 1 from internal oSelf) */
oStand = NEW StroopwafelStand().

oStand = ?.

/* RESULT: RefCount drops from 2 to 1... but NEVER hits 0! 😱 */
/* We have 1 unreachable StroopwafelStand permanently pinned in session memory! */
```

baltic amadeus

Checking For Unreachable Objects

```
DEFINE VARIABLE oObject AS Progress.Lang.Object NO-UNDO.
DEFINE VARIABLE iCount  AS INTEGER              NO-UNDO.

/* Code where you suspect unreachable objects */

oObject = SESSION:FIRST-OBJECT.

DO WHILE VALID-OBJECT(oObject):
    iCount  = iCount + 1.
    oObject = oObject:NEXT-SIBLING.
END.

MESSAGE iCount "Leaked Object(s) Found in Session!".

/* BTW: The exact same pattern works for dynamic handles */
hBuffer = SESSION:FIRST-BUFFER. /* (Traverse via hBuffer:NEXT-SIBLING) */
```

baltic amadeus

Memory Leaks

A memory leak occurs when a program allocates memory (via handles, buffers, or unmanaged references) but fails to release it when no longer needed.

Unlike syntax errors or runtime exceptions that crash instantly, a memory leak is **invisible** at first.





baltic amadeus

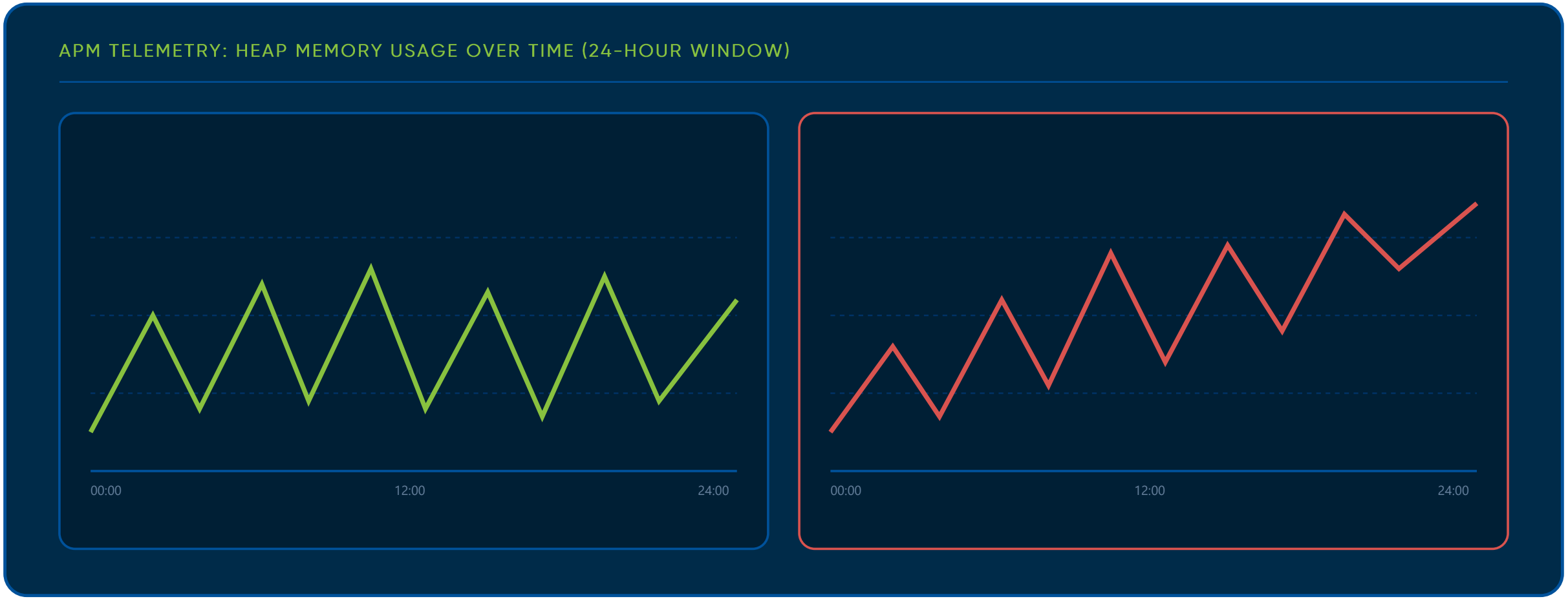
Identifying Memory Leak Symptoms

APM TELEMETRY: HEAP MEMORY USAGE OVER TIME (24-HOUR WINDOW)



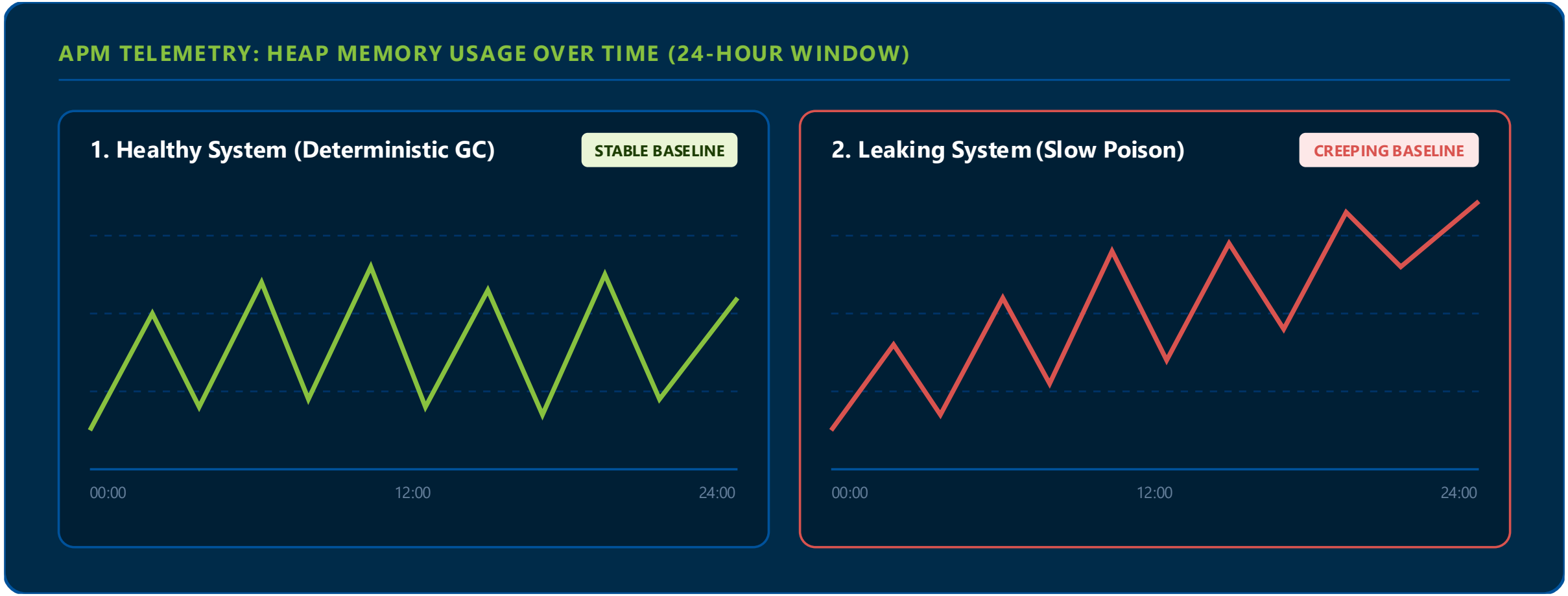
baltic amadeus

Identifying Memory Leak Symptoms



baltic amadeus

Identifying Memory Leak Symptoms



baltic amadeus

The Anti-Pattern: Orphaned Handles

```
PROCEDURE DataCleanup:
    DEFINE VARIABLE hBuffer AS HANDLE NO-UNDO.

    /* Dynamic buffer allocated in session handle table */
    CREATE BUFFER hBuffer FOR TABLE "Order".

    /* ... business logic executes ... */

    /* TRAP: Procedure ends. Variable hBuffer goes out of scope, */
    /* but the buffer object remains permanently pinned in RAM. */
END PROCEDURE.
```

Note: This applies to all local scopes, including Methods, Functions, Constructors, and Triggers.

baltic amadeus

The Best Practice: Secured via FINALLY



```
PROCEDURE DataCleanup:
    DEFINE VARIABLE hBuffer AS HANDLE NO-UNDO.

    CREATE BUFFER hBuffer FOR TABLE "Order".

    /* ... business logic executes ... */

    FINALLY:
        /* GUARANTEED EXECUTION: Even on RETURN, ERROR, or STOP */
        IF VALID-HANDLE(hBuffer) THEN
            DELETE OBJECT hBuffer.
    END FINALLY.
END PROCEDURE.
```

baltic amadeus

The Anti-Pattern: Cascading Leaks

```
/* main.p — The Entry Point */
DEFINE VARIABLE hParent AS HANDLE NO-UNDO.
RUN Parent.p PERSISTENT SET hParent.

/* TRAP: Deleting the parent ONLY kills Parent.p! */
/* It does NOT trigger cleanup for anything Parent.p created. */
DELETE PROCEDURE hParent.

/* Parent.p */
DEFINE VARIABLE hChild AS HANDLE NO-UNDO.
RUN Child.p PERSISTENT SET hChild. /* Survives as an orphan! */

/* Child.p */
DEFINE VARIABLE hBuffer AS HANDLE NO-UNDO.
CREATE BUFFER hBuffer FOR TABLE "Customer". /* Permanently leaked! */
```

We cannot assume that child resources vanish automatically.

baltic amadeus

The Best Practice: Teardown Architecture

```
/* In Parent.p */
PROCEDURE shutdown:
    /* 1. Command the child lineage to purge itself first */
    IF VALID-HANDLE(hChild) THEN
        RUN shutdown IN hChild.

    /* 2. Now safely delete the parent instance */
    DELETE PROCEDURE THIS-PROCEDURE.
END PROCEDURE.

/* In Child.p */
PROCEDURE shutdown:
    /* 3. Clean up local dynamic objects, then self-destruct */
    IF VALID-HANDLE(hBuffer) THEN DELETE OBJECT hBuffer.
    DELETE PROCEDURE THIS-PROCEDURE.
END PROCEDURE.
```

baltic amadeus

The Anti-Pattern: Passing a TABLE-HANDLE by value

```
/* CALLER */
RUN process_data.p (INPUT TABLE-HANDLE hMyTable).

/* CALLEE (process_data.p) */
DEFINE INPUT PARAMETER TABLE-HANDLE hIncoming.

/* TRAP: Because BY-REFERENCE was omitted, the AVM allocates */
/* a brand-new dynamic temp-table and copies every single record! */
/* The callee receives a new dynamic object—permanently leaked if unhandled. */
```

baltic amadeus

The Best Practice: Using BY-REFERENCE

```
/* CALLER: Explicitly passing by reference */
RUN process_data.p (INPUT TABLE-HANDLE hMyTable BY-REFERENCE).

/* CALLEE (process_data.p) */
DEFINE INPUT PARAMETER TABLE-HANDLE hIncoming.

/* RESULT: hIncoming points directly to the caller's original RAM. */
/* CRITICAL RULE: Never call DELETE OBJECT hIncoming in the callee, */
/* as it will destroy the caller's live data! */
```

baltic amadeus

Memory Leak Analysis

- AVM log files & Leak Checker Class.
- OpenEdge Memory Profiler.



baltic amadeus



ProPeek

- Open-source profiler viewer that transforms raw, unreadable .prof files into clear, interactive visual call trees.

https://www.balticamadeus.com/blogs/abl-profiling-made-easier-with-openedge-profiler-and-propeek

baltic amadeus

Conclusions

- **Classes are managed, but handles are yours.**
- **Explicit Cleanup is Mandatory.** Use FINALLY blocks for local routines and DESTRUCTORS for classes.
- Use **BY-REFERENCE** when passing table handles.
- **Trust the Modern Engine:** Never use -nogc as a production optimisation.
- **Instead of guessing, analyse.**