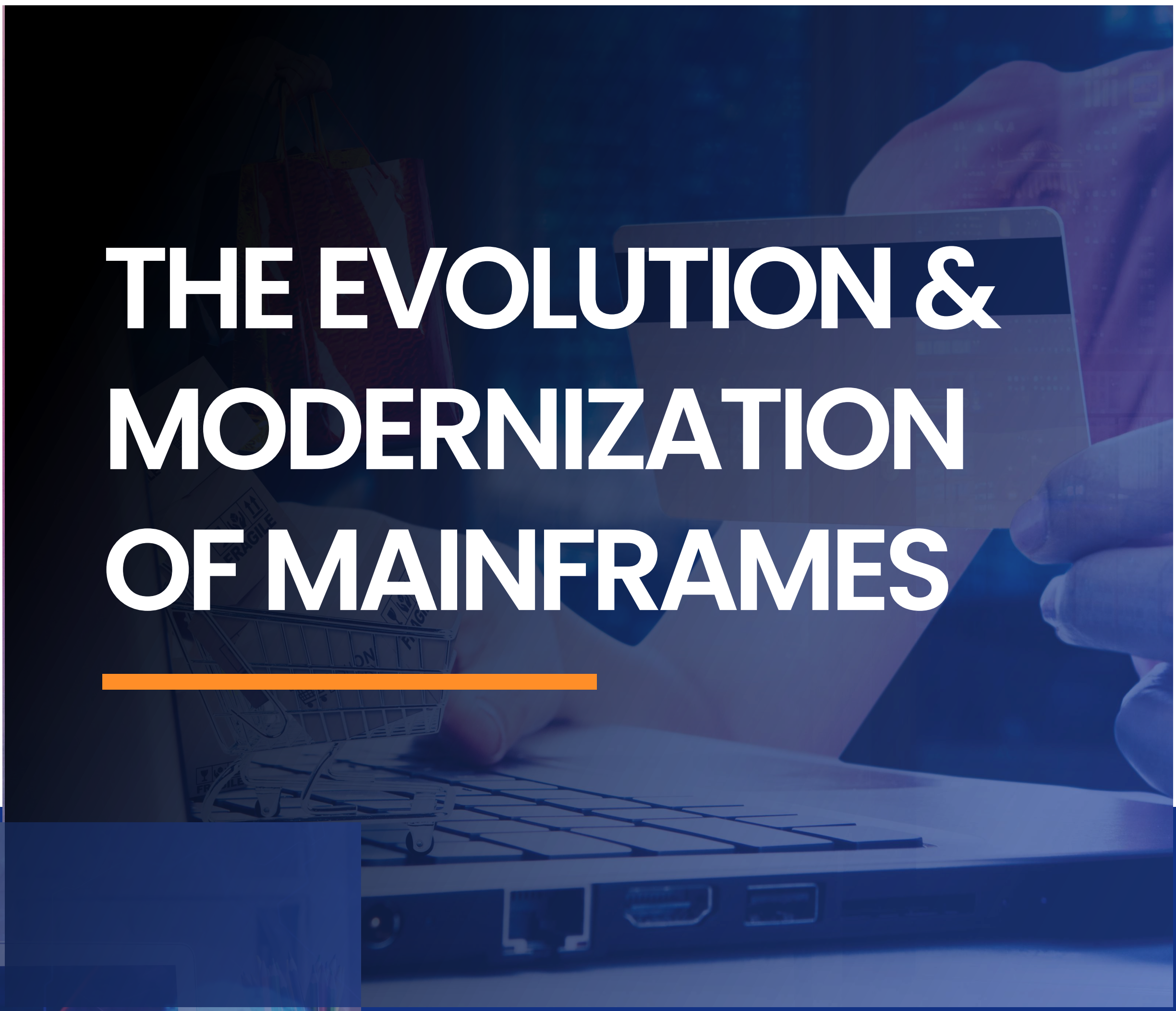
MCLAREN
STRATEGIC SOLUTIONS



THE EVOLUTION & MODERNIZATION OF MAINFRAMES

Author:

Seshadri Nathan Sundaram

Senior Partner - Technology Solutions

McLaren Strategic Solutions

whitepaper

Table of Contents

Introduction

Mainframes have long been the heartbeat of global enterprise IT. From processing airline reservations in the 1960s to managing billions of banking transactions today, their role is undeniable. Yet, for the past two decades, organizations have debated their future: modernize, migrate, or maintain? Having lived through each era — from punched cards to cloud microservices — I've witnessed their strengths and limitations firsthand.

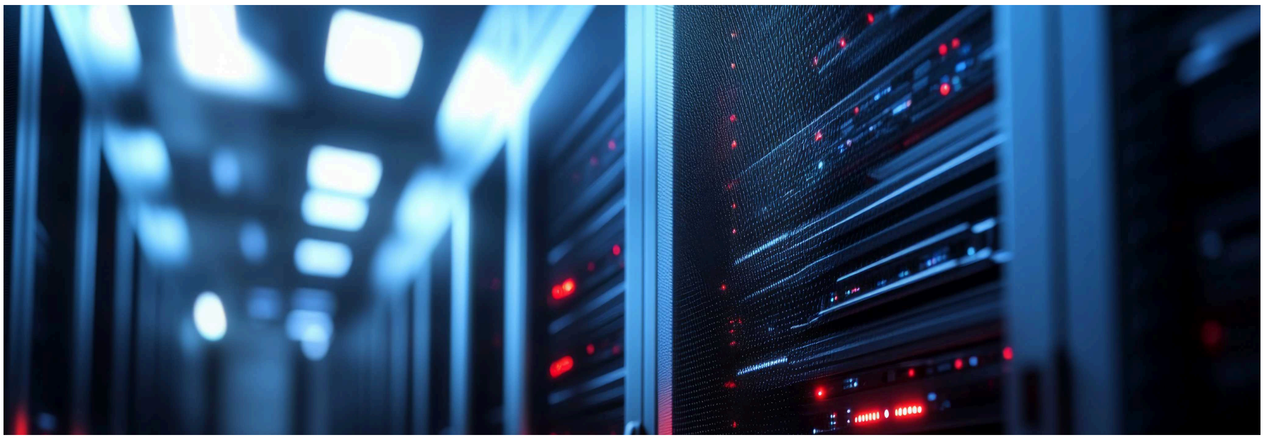
This paper presents a deeply personal yet technically grounded view on how mainframes have evolved, their continuing importance, and how modern technologies — particularly Generative AI — offer a safe and strategic pathway to modernization.

The Birth of Mainframes (1950s–1970s)

Mainframes were born in an era of scientific exploration and industrial automation:

- UNIVAC I (1951): First commercial mainframe by Remington Rand.
- IBM 701 (1952): Designed for scientific use.
- IBM System/360 (1964): Groundbreaking because of its architecture compatibility across models.

These machines used vacuum tubes, magnetic drums, and punch cards. Programming was done in machine



Use Cases

- **Military logistics** and **space mission control**.
- **Payroll and general ledger systems** for large businesses.
- **Batch processing** was the dominant workload type.

Challenges

- Enormous physical size and power requirements.
- Limited I/O and minimal interactivity.
- High cost made them exclusive to governments and large corporations.

Notable Metrics

Metric	Value (1960s - 70)
CPU Speed	0.1 - 1 MIPS
Memory	64KB - 1MB
Storage	Magnetic tapes and drums
Batch Job Duration	30 minutes - several hours
Cost	$2M - $5M USD per installation

Maturity and Dominance (1980s - 1990s)

Widespread Adoption

By the 1980s, mainframes evolved in processing power and architecture:

- **IBM System/370, System/390**
- Introduction of **virtual memory, spooling,** and **JES2/3** job control.
- **TP Monitors** like CICS allowed transaction-level control.

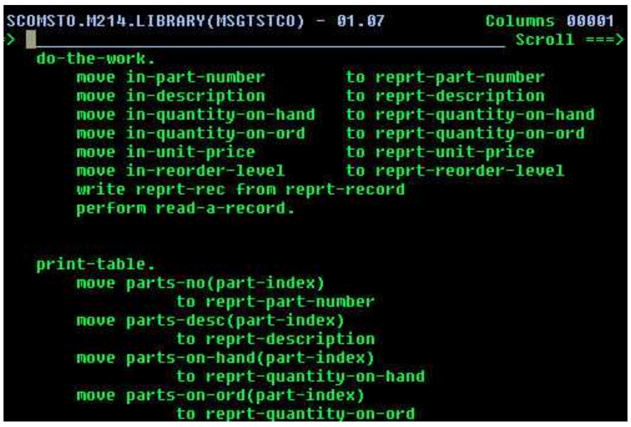
Enterprise Standard

Mainframes became standard for:

- **Core banking** systems (deposits, loans, reconciliation)
- **Stock exchanges** for real-time settlement.
- **Airline reservations** with sub-second booking.
- **Manufacturing and ERP** using IDMS, IMS DB/DC.

Development Ecosystem

- Programming languages: COBOL, PL/I, Assembly, C
- Tools: TSO, ISPF, JCL, VSAM, DB2, IDMS
- Introduction of **RACF** for security



```
SCOMSTO.M214.LIBRARY(MSGTSTCO) - 01.07            Columns 00001
>                                                    Scroll ===>
   do-the-work.
       move in-part-number         to reprt-part-number
       move in-description         to reprt-description
       move in-quantity-on-hand    to reprt-quantity-on-hand
       move in-quantity-on-ord     to reprt-quantity-on-ord
       move in-unit-price          to reprt-unit-price
       move in-reorder-level       to reprt-reorder-level
       write reprt-rec from reprt-record
       perform read-a-record.


   print-table.
       move parts-no(part-index)
                 to reprt-part-number
       move parts-desc(part-index)
                 to reprt-description
       move parts-on-hand(part-index)
                 to reprt-quantity-on-hand
       move parts-on-ord(part-index)
                 to reprt-quantity-on-ord
```

Quantitative Highlights

Metric	Value
MIPS per system	10 - 200
Downtime	< 5 hours/year
Concurrent Sessions Supported	> 10000
TPS (Transactions/sec)	10,000+
Uptime SLA	99.999%

The Client-Server and Internet Era (2000s)

Decentralization Begins

Client-server models began to replace mainframe-exclusive workloads:

- **Java and .NET** enabled GUI-based desktop and web apps.
- Two-tier and three-tier systems became common.

Despite this, mainframes remained:

- **The system of record** (SOR)
- Handling **nightly batch processing, settlements** and **reconciliations.**

Integration Patterns

To coexist with modern systems, mainframes adapted:

- **DCE RPC**, CORBA, and COM/DCOM.
- **Message-Oriented Middleware** like IBM MQ.
- **ETL Pipelines** to sync data with data warehouses.

Hybrid Workloads

- Real-time web interfaces on JAVA/.NET
- Backend processing and validations on mainframes.
- File-based and message-based integrations.

The Cloud Era (2010s - 2020s)

The Rise of Cloud-Native Architectures

While mainframes and client-server models coexisted, the 2010s brought a shift toward cloud-native computing:

- Public cloud providers like AWS, Azure, and GCP gained traction.
- Enterprises began moving workloads to IaaS, PaaS, and SaaS models.
- Containerization (Docker), orchestration (Kubernetes), and serverless computing introduced new deployment paradigms.

Impact on Legacy Systems

- **Decoupling strategy:** APIs and microservices were used to expose core functions.
- **Hybrid environments:** Mainframes continued to coexist with cloud applications.
- **DevOps culture:** CI/CD pipelines improved release velocity, requiring integration with legacy platforms.

Cloud Integration Challenges

- Latency and data gravity concerns
- Securing hybrid traffic.
- Maintaining consistency across on-prem and cloud systems.
- Licensing and compliance complexity.

Yet, cloud computing set a strong precedent for agility and scalability — inspiring legacy modernization initiatives to align with its principles.

Mainframes in the Modern IT Ecosystem

Rebranding and Reinvention

Mainframes didn't just survive—they evolved:

- IBM Z Systems offered encryption-at-scale, container support, and hybrid cloud integration.
- z/OS Connect enabled RESTful APIs.

Key Industries Still Relying

- **Banking:** >90% of ATM and credit card transactions.
- **Insurance:** Claim processing, policy lifecycle management.
- **Government:** Social security, taxation, citizen ID systems.
- **Retail:** POS and supply chain logistics.

Metrics (IBM Z14 and Z15 Series)

Metric	Value
Transactions/day	>30 billion
Encryption Throughput	19 billions REST calls/day
Mean Time Between Failure	>40 years
Power usage Efficiency	~1.5x modern data centers
CPU Utilization	~90-95% sustained

Why Mainframes Persist

Mainframes continue to be the backbone of critical enterprise systems due to their unmatched reliability, performance, and security.

Unmatched Reliability

- Designed for **24x7 uptime** with failover.
- Hardware redundancy and fault-tolerance.

Performance and Scalability

- Sub second response even under **high-concurrency.**
- Linear scaling with LPARs and workload manager.

Security

- EAL5+ certified hardware.
- Built-in encryption and auditing.

Total Cost of Ownership

While upfront costs are high, the **TCO per transaction** is low due to:

- High throughput.
- Long lifecycle (often 20+ years)

The Modernisation Dilemma

Challenges of Staying

Challenge	Details
Skill Shortage	COBOL developers are ageing
High Licensing Costs	Annual MIPS-based pricing
Change Management	Rigid SDLC, limited CI/CD
Knowledge Silos	Minimal documentation, tribal knowledge

Challenges of Leaving

Risk	Details
Business Continuity	Cannot afford downtime
Testing Complexity	Billions of test cases required
Performance Risk	Equivalent performance hard to replicate
Data Consistency	50+ year old data models

The Role of GenAI in Legacy Transformation

The Game Changer

With the emergence of Gen AI and Agentic AI, a safe, incremental modernization path is feasible.

Applications

- Code Understanding: Generate comments, UML, C4, and lineage.
- Transformation: COBOL → Java/C# via LI-AST or DSL.
- Test Automation: AI-generated test cases ensure behavioral parity.
- Documentation: Auto-generate business rules and function specs.
- Validation: AI compares transformed code with original outcomes.

Leveraging Gen AI in Legacy Modernization

Feature	Details
Static Analyzer	Extracts C4, lineage, data dictionary
DTO Generator	Creates POJOs from COBOL data structures
Code Generator	Java/C# modules from Procedure Division AST
Test Case Generator	Generates condition-path based test cases
Validator Agent	Ensures transformed logic maintains intent.

Transformation Potential and Projected Metrics

While full-scale modernization is yet to be executed, industry benchmarks, pilot experiments, and early prototyping suggest promising outcomes for organizations planning Gen AI-led legacy transformation.

Expected Impact: Core Banking Modernization (COBOL → JAVA)

Metric	Current(typical)	Projected with AI-Led Modernization
Code Size (LOC)	10 million	~2.5 million (post modularization and refactoring)
Test Coverage	~35% (manual effort)	>90% using AI-generated test cases
Bug Fix Turnaround	7-10 days	< 1-2 days with AI based code insights and lineage tracing
Developer Onboarding	6-8 months	< 1 month via AI-generated documentations and visualizations
Migration Effort	100%(manual baseline)	40-50% savings using Gen AI tooling and automation pipelines

These are projected metrics based on internal pilot runs, benchmarking studies, and industry reports. Actual results will vary based on system complexity, coverage goals, and transformation approach.

Strategic Vision: Financial Sector Transformation Blueprint

- Legacy stack includes 500+ COBOL programs, spanning over 40 years.
- Expected transformation approach:
    - **Phase 1**: Static analysis + documentation of 20,000+ business rules using GenAI agents.
    - **Phase 2**: Modular modernization of non-core read-only/ reporting modules.
    - **Phase 3**: Parallel deployment of modernized modules with AI-led shadow validation.
- Transformation to occur incrementally with zero planned downtime and ongoing system integrity validation.

Outcomes expected:

- Measurable reduction in maintenance overhead.
- Seamless handover between retiring new-generation engineers.
- Business agility through API-first, microservice-enables components

Conclusion and Future Outlook

Recap

Mainframes were once monoliths, but they became the world's most resilient and powerful transaction engines. As modern architectures demand agility, the need to transform — safely and incrementally — becomes critical.

The Path Forward

- **Don’t rip and replace.**
- **Modernize in phases**, starting with read-only or reference modules.
- **Leverage Gen AI** for reverse engineering, transformation and validation

Final Words

Having worked across every wave of computing, I see Gen AI as a historic inflection point. It doesn’t just accelerate transformation — it democratizes it. With careful planning, mainframe-dependent organizations can move toward the future with **confidence, control, and continuity.**

> *“The mainframe era is not ending. It's evolving into something safer, smarter, and stronger.”*

Personal Commentary

As someone who has spent over four decades in the IT industry, I’ve had the rare privilege of witnessing—and contributing to—the evolution of computing from the era of punch cards and green screens to the age of intelligent Gen AI systems. My journey has taken me through diverse roles and technologies: from COBOL on mainframes and C on Unix systems, to Java and C# in multi-tier architectures, and now to Gen AI platforms transforming how we build, test, and modernize software. I've worked extensively across mainframes, client- server systems, web and mobile application platforms, and more recently, on strategies and architectures to migrate systems to modern cloud-native platforms—always driven by a deep desire to simplify complexity and improve system resilience.

Throughout my career—especially in financial services—I’ve seen organizations struggle with the operational constraints of mainframes: the growing scarcity of COBOL skills, the cost burden of MIPS, and the limitations of batch-oriented processing in an era of real-time customer expectations. For a long time, I believed that mainframes would remain an immovable legacy due to the sheer risk and complexity involved in replacing them. That’s why it feels personally rewarding, even surreal, to be part of the wave that’s enabling safe, incremental mainframe transformation using Gen AI. To architect and guide this shift at this stage in my career is not just an achievement—it’s a milestone I never thought I’d witness, let alone shape.