GeekyAnts

FINTECH ENGINEERING | CORE MODERNIZATION | BANKING ARCHITECTURE

The Core Banking Modernization Playbook

A Framework for Modernizing Legacy Financial Infrastructure While Banking Operations Continue

READ TIME

20 minutes

GEEKYANTS.COM

Innovate. Collaborate. Build.

CONTENTS

In This Guide

A framework for modernizing legacy financial infrastructure while banking operations continue.

Core banking modernization presents a complex engineering problem. Banks need infrastructure that supports new products, changing regulatory requirements, and evolving transaction demands. Their core platforms carry the accounting ledger, transaction processing, compliance records, and integrations that support the institution's products. Years of change requests, acquisitions, and regulatory amendments add dependencies to that foundation.

A full replacement concentrates risk into a large program. Progressive modernization takes a different path. The modern platform runs alongside the legacy core, and teams transfer individual domains or customer segments through controlled stages. The legacy platform continues to support the workloads assigned to it during the migration period.

This guide examines that approach from an engineering and program perspective. It covers the conditions a migration model must satisfy, the mechanics of the Strangler Fig Pattern, legacy-system classification, regulatory continuity, technology choices, and evidence from a production banking modernization program.

SECTION ONE

01 The Modernization Decision

Why core banking modernization requires a phased execution model.

Core banking modernization starts with a constraint: the platform under change has to support live banking operations.

A core banking system carries the accounting ledger, transaction processing, compliance records, and integration points that support products across the institution. Years of change requests, acquisitions, regulatory amendments, and custom extensions add dependencies to many platforms.

That concentration of responsibility makes a full-platform replacement a long program. EY research found that 50% of the financial institutions interviewed aim to complete their core banking modernization programs within three to five years. During the modernization period, the institution must continue to process transactions, support downstream reconciliation, maintain regulatory records, and serve products connected to the core.

The modernization decision centers on migration design.

Why Full Replacement Creates Program Risk

A full replacement places a large set of dependencies inside one transformation program. The target architecture may be sound, yet the migration path still has to account for operational continuity, data movement, compliance records, and integration behavior across the transition.

In a progressive modernization program, a modern core runs beside the legacy platform. Teams move one domain or customer segment at a time. The remaining workloads continue on the legacy system until their migration begins.

This changes the unit of execution.

The program divides the core into smaller business domains as units of migration. Each domain creates a bounded scope for architecture, traffic movement, reconciliation, validation, and decommissioning.

PROGRAM MODEL	EXECUTION SHAPE	OPERATIONAL IMPLICATION
Full-platform replacement	One large transformation program	A broad set of dependencies moves through the same program window.
Progressive modernization	Domain-by-domain migration	Each domain moves through a defined migration stage while remaining workloads stay on the legacy core.

The Questions Technology Leaders Need to Answer

For CTOs and engineering leaders, the modernization decision comes down to a set of architecture and operating questions. These questions determine whether the migration program has a workable foundation.

DECISION AREA	QUESTION
Legacy dependency mapping	Which parts of the legacy core carry load-bearing responsibilities?
Compliance continuity	How will audit and compliance records remain traceable while two systems operate during the migration period?
End-state planning	What conditions define completion and allow the legacy platform to leave production?
Production evidence	What approach has supported this type of migration on a live banking platform?

These questions move the discussion from replacement scope to execution control.

A phased program gives engineering teams a defined unit of change. Each domain can move through routing, validation, reconciliation, and production observation before the next domain enters the migration path.

> **The modernization strategy begins with the migration model. A phased program gives engineering teams a smaller unit of change, a defined production boundary, and a path toward legacy decommissioning.**

Decommissioning Defines the End State

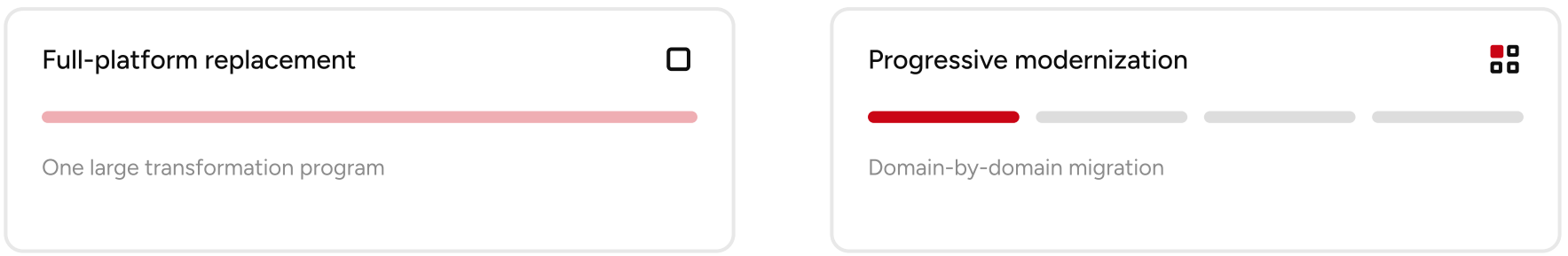
Decommissioning forms part of the modernization objective.

A program that moves workloads to a modern platform still carries the economic burden of the legacy system until the old platform leaves production. The migration plan needs a defined path from coexistence to retirement.

Progressive modernization supports that path by shrinking the role of the legacy core one domain at a time. The program reaches its intended end state when traffic and business responsibilities have moved to the new architecture and the legacy platform can be retired.

The next part of the guide examines the conditions that a migration approach must satisfy before this phased model can work.

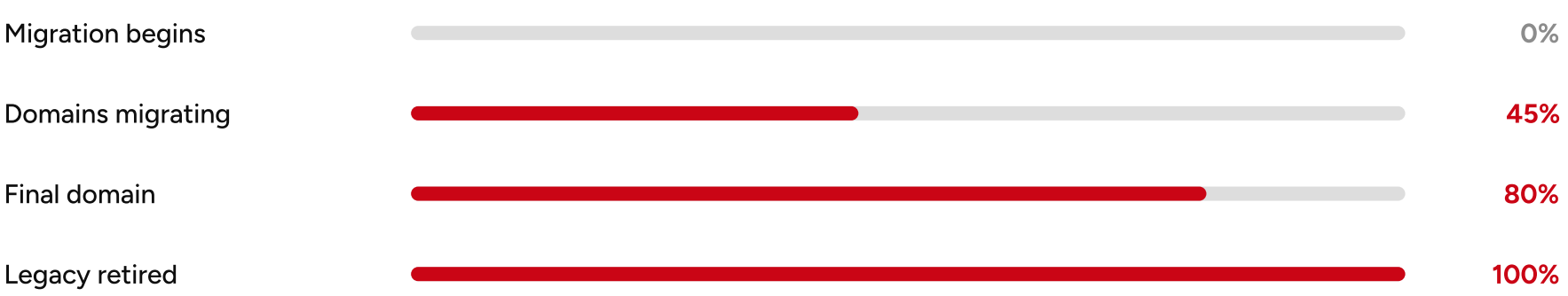
TWO EXECUTION SHAPES



Progressive modernization changes the unit of execution.

Remaining workloads stay on the legacy core.

FROM COEXISTENCE TO RETIREMENT



The legacy core shrinks one domain at a time.

The program ends when the legacy platform can be retired.

SECTION TWO

02 Four Conditions for Core Modernization

What a migration approach must satisfy before architecture decisions begin.

A phased modernization program needs clear operating conditions before architecture decisions begin. Four conditions provide the basis for evaluating an approach: service continuity, reversibility, incremental proof, and compliance continuity.

Together, these conditions provide a practical test for the migration model.

CONDITION	REQUIREMENT	WHAT IT MEANS FOR THE PROGRAM
Service continuity	The migration must preserve live banking operations.	Transaction posting, downstream reconciliation, and connected products continue to operate during migration.
Reversibility	Each migration step needs a path back.	Teams can reverse a traffic change at the level of the current migration step.
Incremental proof	Each stage needs to demonstrate results before the program expands.	A domain can establish production evidence before the next domain enters the migration path.
Compliance continuity	Audit and compliance records must remain intact while both systems operate.	The institution maintains traceability throughout the period when legacy and modern systems share responsibilities.

Service Continuity

Core banking platforms support transaction posting, downstream reconciliation, and products across the institution. This creates a continuous service requirement throughout the migration program.

A suitable migration model therefore keeps the existing platform available while workloads move. Traffic routing becomes part of the architecture because the program needs a controlled mechanism for directing transactions between legacy and modern services.

Reversibility at Each Step

A multi-year modernization program contains many individual production changes. Reversibility needs to exist at the same level.

Each migration step needs its own reversal path. A routing model supports this requirement by allowing traffic to move between the legacy and modern implementations as a domain progresses through migration.

This gives teams a defined recovery path for each stage of the program.

Incremental Proof Before Program Expansion

Progressive modernization divides the program into domains. Each domain creates an opportunity to establish production evidence before investment extends to another part of the core.

A multi-year, fixed-scope replacement can require sustained commitment before the organization sees the full result. Progressive modernization creates shorter proof cycles that demonstrate progress before the program requests further budget.

Incremental proof also creates evidence for architecture decisions. Teams can use production results from one domain when planning the next extraction.

Compliance Continuity Across Both Systems

During progressive modernization, legacy and modern systems coexist. That period creates a specific compliance requirement.

Audit posture and compliance records need to remain intact while both platforms carry responsibilities. The migration architecture therefore needs to preserve transaction traceability across the coexistence period.

The guide examines the mechanics of this requirement in the compliance section.

SHORTER PROOF CYCLES

Evidence arrives by domain, not at the end.

Production results from one domain inform the next extraction.

How the Main Approaches Compare

The four conditions create a common basis for comparing full replacement, big-bang cutover, phased rebuilds, and the Strangler Fig Pattern.

APPROACH	SERVICE CONTINUITY	REVERSIBILITY	INCREMENTAL PROOF	COMPLIANCE CONTINUITY
Full replacement	Depends on the final cutover model	Depends on the final cutover model	Limited during a long fixed-scope program	Requires separate migration design
Big-bang cutover	Requires a cutover window	Creates a complex reversal path after traffic moves	Concentrates proof around cutover	Requires reconciliation across the cutover
Phased rebuild	Supports staged delivery	Depends on routing design	Supports evidence across phases	Requires explicit design for the coexistence period
Strangler Fig Pattern	Uses traffic routing while both implementations coexist	Supports traffic movement in either direction	Allows one domain to prove the pattern before the next	Supports independent auditability while both systems coexist

The Strangler Fig Pattern aligns with all four conditions through its architecture. Traffic moves through routing rather than a single cutover. Routing can move traffic between implementations. Domains can establish production evidence in sequence. Both systems can retain auditability throughout coexistence.

> **A viable core modernization model preserves service, supports reversal at each migration step, produces evidence by domain, and maintains audit continuity throughout coexistence.**

Why the Strangler Fig Pattern Fits the Model

The Strangler Fig Pattern transfers responsibilities from an existing system to a new architecture in stages while the existing platform continues to run. Its name comes from the growth pattern of a strangler fig, which develops around a host tree and assumes its structural load over time.

Applied to core banking, the pattern creates a progressive path from coexistence to decommissioning. The migration program can direct traffic to an extracted domain, validate its behavior, expand its responsibility, and repeat the process across the core.

The next section examines how that pattern works at the architecture and traffic-routing level.

SECTION THREE

03 How the Strangler Fig Pattern Works

A domain-by-domain migration model.

The Strangler Fig Pattern turns core modernization into a sequence of controlled production changes. The model introduces a routing layer, extracts one domain, shifts traffic to the new service, and repeats the process until the legacy platform reaches retirement.

The routing layer sits at the center of this model. It gives the program control over which implementation receives each request throughout the migration.

THE ROUTING LAYER IN FRONT OF THE CORE

Callers keep the same interaction path.

Routing decisions occur within the migration architecture.

Step 1: Introduce the Routing Layer

The routing layer sits in front of the legacy core. Requests that once reached the legacy platform pass through this facade.

The facade can direct each request to the legacy system or the new service. Calling applications and end customers continue to use the same interaction path while routing decisions occur within the migration architecture.

This layer establishes the mechanism required for later traffic shifts.

ROUTING STATE	REQUEST PATH	PURPOSE
Before extraction	Routing layer → Legacy core	Establish the facade while preserving the existing processing path.
During migration	Routing layer → Legacy core or new service	Control traffic allocation between both implementations.
After domain migration	Routing layer → New service	Move the domain's processing responsibility to the new architecture.

Step 2: Extract One Domain

The unit of modernization is a defined domain rather than the entire core.

Examples of extraction domains include account opening, balance inquiry, and transaction processing for a specific product line. Other responsibilities remain on the legacy platform while the selected domain moves into a new service.

This creates an independent delivery boundary. Engineering teams can build and validate one domain while the rest of the core continues to support its assigned workloads.

The same pattern can then move to another domain.

Step 3: Shift Traffic in Stages

Once the new service is ready, the routing layer begins directing a share of the selected domain's traffic to it.

Traffic can start with a small allocation and increase in stages as confidence in the new service grows. The legacy route remains available as the fallback path during this period.

If the new service encounters a problem at a given traffic level, the routing layer can direct that traffic back to the legacy implementation. Reversibility therefore operates through traffic allocation.

THE UNIT OF MODERNIZATION IS A DOMAIN

One domain moves; the rest of the core keeps working. Other responsibilities remain on the legacy platform.

MIGRATION STAGE	ROUTING BEHAVIOR	ENGINEERING OBJECTIVE
Initial exposure	A small share of domain traffic reaches the new service	Observe the new service under production traffic.
Staged expansion	The new service receives a larger traffic share	Build production evidence across increasing load.
Domain transition	The new service receives the domain's traffic	Transfer processing responsibility for that domain.
Reversal	Traffic returns to the legacy implementation	Restore the prior processing path.

> **Reversibility comes from routing control.** **Each domain can move through staged production traffic while the legacy processing path remains available during migration.**

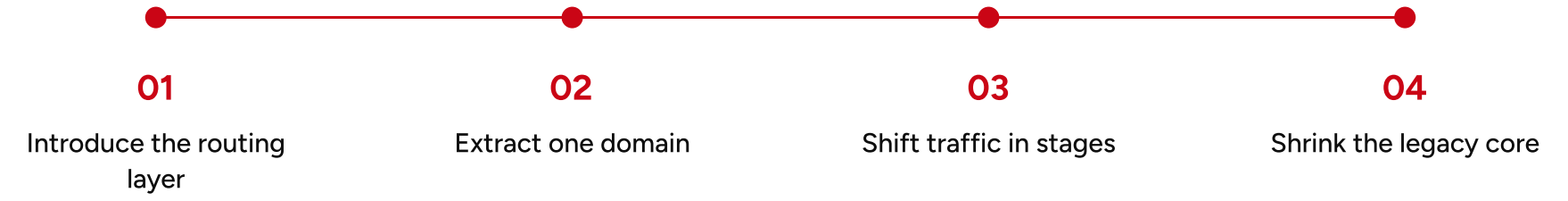
Step 4: Shrink the Legacy Core

Each completed domain reduces the set of responsibilities handled by the legacy platform.

The process repeats until traffic has moved away from the legacy core across the required domains. Decommissioning occurs after the legacy system has transferred its processing responsibilities to the new architecture.

This final step matters to the economics of modernization. Continued operation of the legacy platform creates a second system to maintain after migration. Decommissioning forms part of program completion.

FOUR STEPS, REPEATED PER DOMAIN



The cycle repeats until the legacy core reaches retirement.

Reversal stays available at every step.

The Engineering Responsibilities Created by the Pattern

The Strangler Fig Pattern introduces its own engineering responsibilities. The routing layer becomes part of the transaction path because transactions pass through it. Parallel operation also creates a period when legacy and modern systems share responsibility for a domain. That coexistence creates reconciliation and audit-trail requirements that the architecture must address.

Two areas require attention:

ENGINEERING AREA	REQUIREMENT
Routing infrastructure	The facade must support reliable traffic control and reversal throughout migration.
Reconciliation and audit	The program must preserve transaction traceability while legacy and modern systems coexist.

The shape of this migration also depends on the legacy platform itself. A packaged core, custom-built system, mainframe platform, and hybrid architecture each present a different starting point for extraction.

The next section examines those differences and the discovery work required before teams commit to a migration estimate.

STAGED TRAFFIC MOVEMENT

Reversibility operates through traffic allocation. The legacy route remains available as the fallback path.

SECTION FOUR

04 Classifying the Legacy Core

The starting architecture shapes the migration plan.

The Strangler Fig Pattern depends on a legacy platform that teams can wrap, observe, and extract in stages. The amount of discovery work changes with the structure of that platform.

Legacy cores fall into four categories: packaged, custom-built, mainframe-based, and hybrid systems. Each category creates a different starting point for routing, extraction, and data migration.

Four Legacy Platform Types

LEGACY PLATFORM TYPE	WHAT DEFINES IT	WHAT IT MEANS FOR MIGRATION
Packaged core	Vendor-maintained platform with documented APIs and known upgrade paths. Examples include Oracle FlexCube, Temenos, and Finacle.	Existing integration points can support the routing facade. Vendor documentation can reduce discovery work.
Custom-built proprietary core	In-house platform developed across years, with business logic held in code and limited documentation.	Discovery must identify business rules before extraction begins. Teams may need to reconstruct those rules from the running system.
Mainframe-based system	Platform built on technologies such as COBOL, PL/I, or hierarchical databases such as IMS.	The routing layer may need to connect modern APIs with batch and transaction protocols. Data migration becomes a major program concern.
Hybrid or previously modernized system	Platform that contains components from an earlier modernization effort alongside legacy architecture.	Discovery must establish which responsibilities belong to each layer before extraction begins.

Packaged cores provide a favorable starting point because documented integration points give the routing layer a defined place to connect. Custom and mainframe systems require more discovery because teams need to identify business logic, protocols, or data structures before extraction begins. Hybrid systems create another challenge because the visible platform label may describe only part of the architecture.

The Hybrid-System Misdiagnosis

A platform can begin as a packaged core and change shape across years of operation.

Custom modules may appear around local regulatory requirements. Teams may add reconciliation layers, patches, and services outside the original product. A previous modernization program may leave newer services connected to the legacy platform. The result can carry a packaged-core vendor name while operating as a hybrid architecture.

That distinction affects planning.

Discovery estimates, migration timelines, and budgets depend on the architecture teams expect to encounter. A program priced as a contained upgrade or re-platforming effort can significantly underestimate the work when discovery reveals a hybrid system that requires progressive modernization, parallel operation, extensive integration, and eventual legacy decommissioning. In this situation, early estimates can reach 2–3x their original level.

> **Legacy classification should come from the running system and its change history. The platform label alone provides an incomplete basis for migration planning.**

WHEN DISCOVERY REVEALS A HYBRID SYSTEM

Hybrid systems are misdiagnosed as contained upgrades. Early estimates can reach 2–3x their original level.

A Discovery Audit Before Estimation

Teams should conduct a short discovery audit before committing to extraction estimates. The audit focuses on the current production system and its change history.

DISCOVERY CHECK	WHAT THE TEAM EXAMINES	WHAT IT REVEALS
Review recent change requests	The last two to three years of changes, patches, and modifications	The gap between the original platform and its current production form.
Trace one transaction	One transaction path across services, batch jobs, and manual interventions	The difference between documented architecture and production behavior.
Test business-rule knowledge	Whether a business rule can be explained from documentation or institutional knowledge	The amount of logic that exists within the running code.
Look for prior modernization artifacts	Orphaned services, unexplained feature flags, and different logging formats	Evidence that the platform contains elements from an earlier modernization effort.

The transaction trace gives the team a production-level view of the system. Teams can choose one non-trivial transaction and follow it through every service, batch job, and manual intervention involved in its production path.

Business-rule discovery adds another signal. When teams depend on source code to explain a rule, the migration program needs to account for business-logic reconstruction as part of discovery.

Evidence from a previous modernization attempt also changes the classification. Orphaned services, unexplained feature flags, and different logging formats can indicate a hybrid architecture.

TRACE ONE TRANSACTION

Follow one transaction through its full production path.

Documented architecture vs production behavior.

Classification Changes the Program

Legacy classification influences discovery, compliance planning, and technology selection.

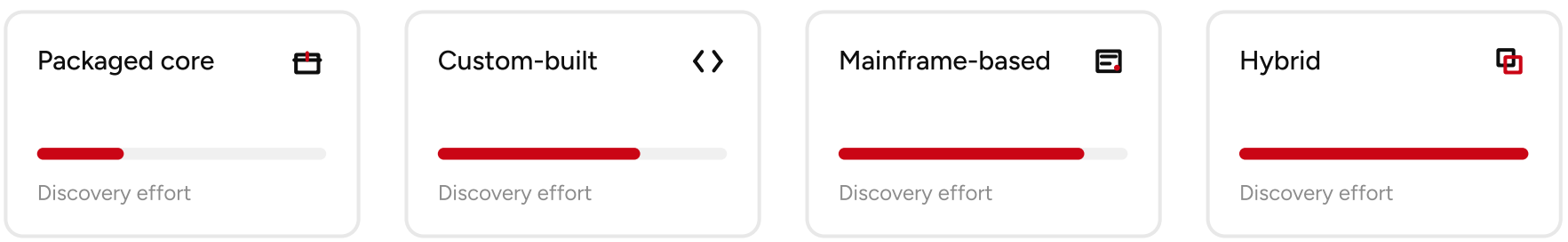
First, it shapes the amount of discovery required before the routing layer can be built. Second, it affects the reconciliation and audit requirements created during coexistence. Third, it constrains the technology choices available for routing, data movement, and integration.

A packaged core may offer documented interfaces for extraction. A custom system may require business-rule reconstruction. A mainframe system may require integration across different protocol generations. A hybrid system may require teams to map multiple architectural layers before defining the first extraction boundary.

Classification therefore belongs before timeline, budget, and stack commitments.

The next section examines the compliance requirements created when legacy and modern systems operate during the same migration period.

DISCOVERY EFFORT BY PLATFORM TYPE



Classification belongs before timeline, budget, and stack commitments.

The label is not the architecture.

CLASSIFICATION FEEDS THREE DECISIONS

- **Discovery scope**
  How much discovery is required before the routing layer can be built.
- **Reconciliation and audit**
  The requirements created while legacy and modern systems coexist.
- **Technology constraints**
  The choices available for routing, data movement, and integration.

Classification feeds three downstream decisions.

Discovery, compliance planning, and technology selection.

SECTION FIVE

05 Maintaining Regulatory Compliance During Migration

Audit continuity across two live systems.

Progressive modernization creates a period when legacy and modern systems share responsibility for transaction processing. That period changes the compliance model.

A regulator needs a clear record of which system processed a transaction, which customer the transaction affected, and which system held authority at that point in the migration. Reconstructability provides that record across the transition.

The architecture therefore needs to preserve an auditable transaction path across both systems during coexistence.

The Two-System-of-Record Problem

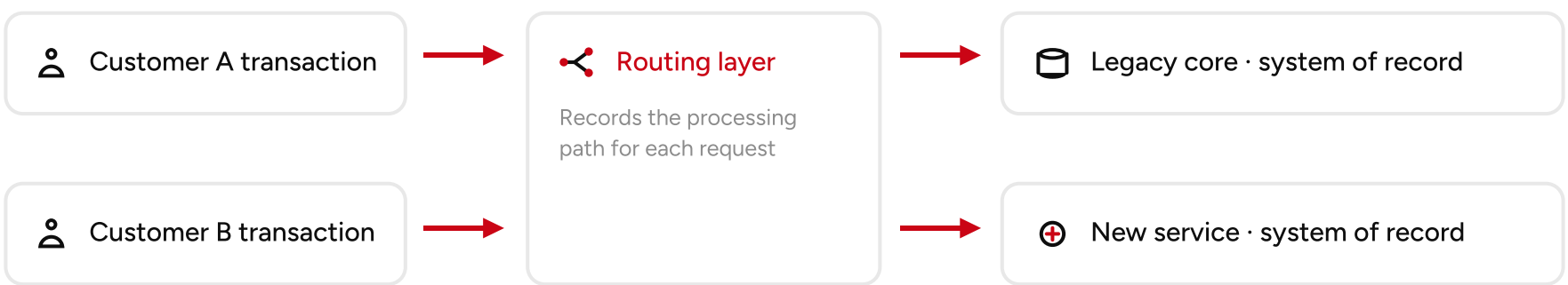
A phased migration can place one domain across two processing environments. One customer's transaction may pass through the legacy system while another customer's transaction for the same domain reaches the new service.

This creates a system-of-record question for every transaction.

The routing layer becomes part of the audit model because it records which processing path received each request. That record needs to support regulatory reconstruction during the migration period.

The compliance design therefore needs four capabilities.

ONE DOMAIN, TWO SYSTEMS OF RECORD



The routing layer records which path received each request. That record supports regulatory reconstruction.

COMPLIANCE REQUIREMENT	WHAT THE PROGRAM NEEDS
Dual audit trails	Each transaction remains traceable in the system that processed it, with a reconciliation process across both records.
Authority record	The program maintains a record of which system held authority for each transaction.
Regulatory reporting continuity	Reports continue to produce the required data across legacy, modern, and coexistence states.
Regulator communication	The institution shares the migration model, reversal mechanics, and reconciliation approach with the regulator before migration begins.

Dual Audit Trails and Reconciliation

Each transaction needs a trace from processing through audit.

During coexistence, the program maintains records across both systems and reconciles them. The reconciliation process identifies discrepancies across the migration period.

This requirement connects architecture with compliance operations. Transaction identifiers, processing records, routing decisions, and reconciliation outputs need to support a common reconstruction path.

A Record of Transaction Authority

The migration program needs a clear answer to one question for every transaction: which system held authority when the transaction occurred?

The routing layer provides part of that record by identifying the processing destination for each request. Routing history, transaction records, and reconciliation outputs contribute to the audit trail used to reconstruct transaction processing during migration.

That history connects migration mechanics with regulatory evidence.

TRANSACTION STATE	PROCESSING RECORD	AUDIT REQUIREMENT
Legacy route	Transaction processed by the legacy core	Preserve the legacy transaction record and routing decision.
Modern route	Transaction processed by the extracted service	Preserve the modern transaction record and routing decision.
Migration-period reconciliation	Records reviewed across both environments	Identify and resolve discrepancies between processing records.

Regulatory Reporting During Coexistence

Regulatory reporting still needs complete transaction data while domains move between systems.

A report may need information stored in the legacy platform, the modern service, or both environments during a migration stage. The reporting design therefore needs a consistent path to the required data across each processing state.

This makes reporting part of migration architecture rather than an end-state activity.

Regulator Communication Before Migration

The migration plan should include regulator communication before transaction traffic begins to move.

That communication covers the phased migration model, reversal mechanics, and reconciliation process before transaction traffic moves.

Regulatory requirements vary across jurisdictions, including frameworks associated with RBI, NPCI, FinCEN, BaFin, and MENA markets. Transaction reconstructability remains the governing requirement throughout the migration.

> **Compliance continuity depends on reconstructability.** **Every transaction needs a traceable processing path, an authority record, and a reconciled audit trail during the migration period.**

Where Compliance Gaps Can Emerge

Compliance gaps can emerge at the edges of the migration architecture.

A batch job may depend on the legacy platform as the authoritative source. A regulatory report may require an update when a domain moves. A reversal may change application state while the audit record follows a different path. These dependencies need coverage within the migration design.

The compliance work therefore extends beyond reconciliation logic. Teams need to identify every processing, reporting, and audit path connected to the migrated domain.

This requirement shapes the next set of architecture decisions. The technology stack needs to support routing control, data synchronization, reconciliation, audit evidence, and the integration constraints of the legacy platform.

EDGES OF THE MIGRATION ARCHITECTURE

— Batch dependency

A batch job may depend on the legacy platform as the authoritative source.

— Report scope

A regulatory report may require an update when a domain moves.

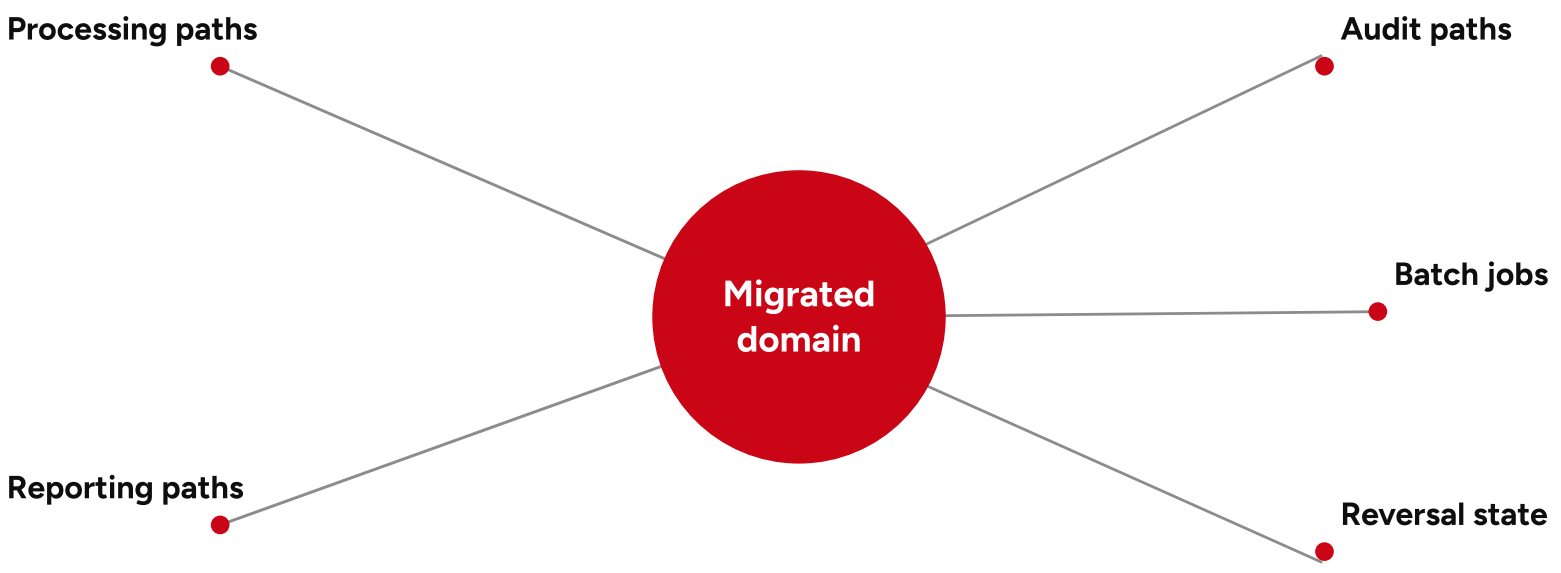
— Reversal state

A reversal may change application state while the audit record follows a different path.

These dependencies need coverage within the migration design.

Compliance work extends beyond reconciliation logic.

EVERY PATH CONNECTED TO THE DOMAIN



Identify every path before the domain moves.

Processing, reporting, and audit paths all need coverage.

SECTION SIX

06 Making the Technology Stack Decision

Architecture choices for routing, data, and parallel operations.

The technology stack has a defined role in a phased modernization program. It must support routing reliability, reconciliation, audit requirements, and the integration limits of the legacy platform.

The decision therefore begins with migration requirements rather than a general preference for newer technologies.

Four Questions for Technology Evaluation

Stack evaluation covers four areas: routing reversibility, data synchronization, parallel-run cost, and compliance tooling.

EVALUATION AREA	QUESTION	WHAT THE STACK MUST SUPPORT
Routing	Can the routing layer support traffic shifts and reversal?	Controlled traffic allocation between legacy and modern services.
Data	Can the data layer support dual-write or change-data-capture patterns?	Observation and reconciliation across both systems.
Infrastructure cost	What does parallel operation cost across the migration period?	Cost visibility across legacy and modern infrastructure.
Compliance tooling	Which audit and reconciliation capabilities already exist, and which require development?	Transaction reconstruction and audit evidence.

Routing and Reversibility

The routing layer needs to support percentage-based traffic movement and reversal.

This capability allows a domain to move through production stages while the legacy processing path remains available during migration. The gateway or service-mesh layer must support percentage-based traffic routing and reversal.

The technology choice therefore needs to match the routing model established earlier in the guide.

Data Synchronization and Reconciliation

Parallel operation creates a data-observation requirement across both systems.

Dual-write and change-data-capture patterns can support reconciliation between legacy and modern environments. The available choice depends on what the legacy platform can expose.

That constraint connects the technology decision to legacy-system classification.

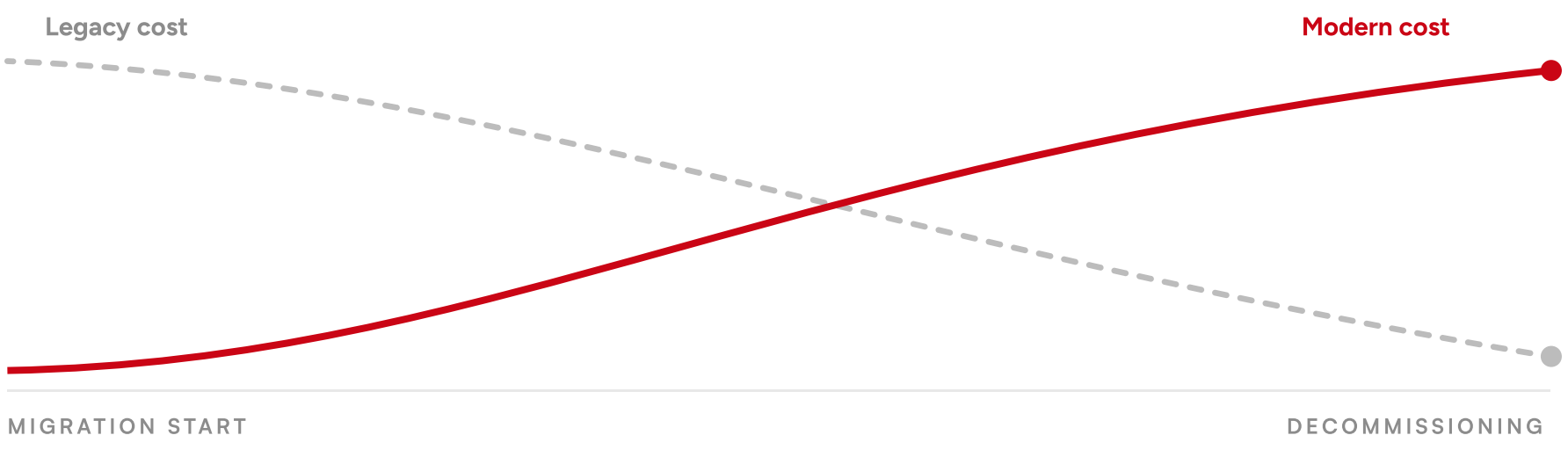
A packaged platform may expose documented interfaces. A custom or mainframe system may require a different data-access path. The data architecture needs to support the form of observation available in the source system.

Modeling the Cost of Parallel Operations

Progressive modernization creates a period when legacy and modern infrastructure run at the same time.

That coexistence creates an infrastructure cost curve across the migration period. Teams should model this cost before the program begins and track it while both environments remain active.

THE COST CURVE OF PARALLEL OPERATIONS



Both environments carry cost until decommissioning.

Track the curve while both environments run.

COST AREA	PLANNING QUESTION
Legacy infrastructure	What cost continues while domains remain on the existing platform?
Modern infrastructure	What cost appears as new services enter production?
Parallel-run period	How long will both environments carry operating cost for the same program?
Decommissioning	When will legacy infrastructure leave the cost base?

Compliance Tooling

The stack decision also covers audit-trail and reconciliation tooling.

The program needs to determine which capabilities can come from existing systems and which need engineering work. These capabilities must support full transaction reconstruction during migration.

This evaluation belongs alongside routing and data decisions because all three contribute to the evidence required during coexistence.

> **The right technology stack is the one that supports routing control, data reconciliation, audit evidence, and the integration limits of the legacy platform.**

Technology Decisions Can Change by Domain

Stack selection should remain a progressive decision throughout the migration.

A technology choice that suits the first extracted domain may fit a later domain differently. Complexity, data access, transaction behavior, and integration constraints can change as the migration moves across the core.

The architecture process therefore needs a review point at each extraction stage.

Teams can assess the next domain against the same four questions: routing, data, cost, and compliance tooling. This keeps the stack aligned with the migration path rather than locking every future domain into one early architecture choice.

The next section moves from framework to production evidence and examines how these decisions appeared in a live core banking modernization program.

SECTION SEVEN

07 Progressive Modernization in Practice

A production core banking modernization program.

The framework becomes concrete when applied to a live banking platform. One production engagement began with a packaged core and used progressive modernization across a multi-year program.

The engagement covers the starting architecture, transaction-layer changes, compliance continuity, and confirmed outcomes.

Starting Architecture

The production engagement involved a packaged core built on Oracle and FlexCube through the Oracle Banking Platform, or OBP.

This classification shaped the discovery phase. Vendor documentation provided information about the existing platform and its integration points. That documentation gave the team a foundation for extraction work with less business-rule reconstruction than custom and mainframe environments require.

STARTING POINT	ENGAGEMENT DETAIL
Legacy classification	Packaged core
Core technologies	Oracle and FlexCube
Platform	Oracle Banking Platform
Initial advantage	Existing vendor documentation supported discovery

A Multi-Year Modernization Program

The modernization engagement ran from March 2023 through September 2025. During this period, the program followed a progressive model in which responsibilities moved from the legacy platform as modernization continued. Domain-level migration formed part of this progression. Program duration depends on the condition of the existing platform, the modernization scope, and the requirements identified during discovery.

Transaction-Layer Architecture

One documented technical change involved transaction-layer communication.

The program included a move in transaction-layer communication from SOAP to gRPC and the introduction of multi-tier caching. These changes formed part of the transaction-layer modernization implemented during the program.

ARCHITECTURE AREA	DOCUMENTED CHANGE
Transaction communication	SOAP to gRPC
Performance layer	Multi-tier caching
Migration context	Transaction load within the progressive modernization program

These choices addressed the communication and performance requirements of the transaction layer.

Compliance Continuity

RBI audit-trail continuity remained in place across the migration period. Regulatory reportability continued across each phase of the program.

This result connects the production engagement with the reconstructability requirement established earlier in the guide. Transactions needed an auditable record while modernization progressed across the live platform.

Production Scale Across Banking Modules

The broader banking engagement included production modules across money transfers, bill payments, UPI, loans, cheques, and card services.

The money-transfer module reached more than 5 million users following a 1.5-year delivery period and supported transactions worth hundreds of crores each day. Its scope included NEFT, IMPS, RTGS, foreign remittance, self-transfers, credit-card transfers, ECMS, beneficiary management, and transaction history.

The BillPay module supported payments across more than 1,000 billers through credit cards, debit cards, and bank accounts.

Engineering a Production UPI Module

The UPI engagement began with team onboarding in September 2023, followed by implementation work from late October. The documented development period covered around 16 months.

The implementation addressed NPCI SDK version mismatches, race conditions between the UPI library and the main application, secure token storage, and edge cases within the Send Money flow.

The engineering approach used React Native Keychain for token storage. The application stored and checked the NPCI SDK version and creation date before transactions and refreshed token data when a version mismatch occurred. Common methods standardized data received from the NPCI SDK before it entered application flows.

Verified Outcomes

The engagement produced five confirmed outcomes.

OUTCOME	RECORDED RESULT
Transaction latency	56% reduction, from 500 ms to 220 ms
Throughput	10x improvement
Service continuity	Zero downtime throughout the modernization program
Regulatory continuity	Full regulatory audit trail maintained throughout the migration
Platform scale	More than 5 million active users served by the migrated platform

The program recorded improvements in transaction latency and throughput alongside the transaction-layer modernization.

Service continuity and audit-trail preservation show how the migration operated against a live financial platform while responsibilities moved across the architecture.

> **The production program combined progressive migration with transaction-layer modernization, maintained regulatory audit continuity, and recorded gains in latency and throughput while serving more than 5 million active users.**

SECTION EIGHT

08 Core Banking Modernization Readiness Assessment

Is your organization ready for a phased modernization program?

A modernization program requires preparation across architecture, compliance, infrastructure economics, and organizational commitment. This assessment brings those requirements into one decision framework.

Use the seven questions below to review the current state of your modernization planning and identify areas that may benefit from further preparation.

Assess Your Core Banking Modernization Readiness

01	**Domain Sequencing** Can you define the first three domains in the extraction sequence?	☐ Yes	☐ No
02	**Legacy Classification** Have you classified your legacy system as packaged, custom-built, mainframe-based, or hybrid?	☐ Yes	☐ No
03	**Compliance Continuity** Does your organization have a plan for dual audit trails during the migration period?	☐ Yes	☐ No
04	**Regulator Communication** Has your regulator been brought into the migration plan before implementation begins?	☐ Yes	☐ No
05	**Infrastructure Economics** Has your organization modeled the cost of running legacy and modern infrastructure in parallel?	☐ Yes	☐ No
06	**Legacy Decommissioning** Is there a funded plan with a defined date for decommissioning the legacy system?	☐ Yes	☐ No
07	**Incremental Proof** Can your organization prove one domain in production before committing budget to the next?	☐ Yes	☐ No

Evaluate Your Readiness

Review the pattern across your seven responses and identify the profile that best reflects the current state of your modernization planning.

High Readiness

Response pattern: Most responses are Yes.

The modernization plan covers the major foundations for a phased program. The assessment provides a checkpoint for validating these decisions against the planned migration scope.

Developing Readiness

Response pattern: The responses include a mix of Yes and No.

The assessment highlights specific areas that can benefit from further planning before implementation begins. Each response provides a starting point for the corresponding planning area.

Foundation Stage

Response pattern: Most responses are No.

The assessment provides a starting point for planning across legacy classification, compliance, migration sequencing, infrastructure economics, and decommissioning.

> **Modernization readiness comes from defined extraction boundaries, an accurate view of the legacy platform, compliance continuity, infrastructure planning, and a funded path to decommissioning.**

Turn Readiness Gaps Into Program Decisions

The value of the assessment comes from the action that follows it.

An unclear extraction sequence points to domain-planning work. Uncertain legacy classification calls for a discovery audit. Gaps in audit trails or regulator communication require compliance planning. An undefined parallel-run cost profile calls for infrastructure modeling. A missing decommissioning plan leaves the program without a defined economic endpoint.

The assessment gives technology leaders a practical starting point for converting modernization intent into a phased program plan.

FROM GAP TO PROGRAM DECISION

— Unclear extraction sequence	/ Domain-planning work
— Uncertain legacy classification	/ Discovery audit
— Audit or regulator gaps	/ Compliance planning
— Undefined parallel-run cost	/ Infrastructure modeling
— No decommissioning plan	/ A funded end date

Each gap points to a defined piece of program work.

The assessment converts intent into a phased plan.

Get in touch for a core modernization consultation

Schedule a Call

geekyants.com

Let’s talk.

Modernizing a core banking platform? We would like to hear about it.

MAIL
hello@geekyants.com

WEB
www.geekyants.com

GeekyAnts

I N N O V A T E . C O L L A B O R A T E . B U I L D

🇮🇳 GeekyAnts India Pvt Ltd

No. 18, 2nd Cross Road, N S Palya,
2nd Stage, BTM Layout, Bangalore,
560076, Karnataka, India

🇬🇧 GeekyAnts UK Ltd

SPACES Finsbury Park, 17 City
North Place, London N4 3FU,
England, UK

🇺🇸 GeekyAnts Inc

315 Montgomery Street, 9th &
10th Floors, San Francisco,
CA, 94104, USA