pretius

# Leave Oracle Forms on your terms

## Flexible Oracle Forms Migration

What your organization actually gains from migration, and three paths to get there. Chosen through analysis, not a vendor's preference.

# Executive summary

## Your Oracle Forms system still works. That is exactly why this decision keeps getting postponed, and exactly why it should not be.

For twenty years, the conversation about Oracle Forms has been framed around endings: end of support, end of patches, end of available developers. This guide argues the opposite. **The strongest case for leaving Forms has nothing to do with what ends.** It is about what your organization gains: a lower and more predictable cost base, access to mainstream talent, real-time integration, a user experience employees actually expect, a clean compliance position, and a platform that can carry analytics and AI instead of blocking them.

The core of this guide is an honest as-is to-be comparison across six dimensions that matter to the business, not just to IT. Around it, we answer the three questions every organization running a business-critical Forms system eventually faces: **should we migrate or rebuild from scratch** (migrate, in almost every case), **which of the three exit paths fits our strategy** (stay in Oracle and move to APEX; keep the Oracle database and modernize the frontend; or exit Oracle entirely), and **how do we do this safely** when the system in question runs the company.

**One thing this guide will not do is use December 2026 as a scare tactic.**
End of support for Forms 12c is a fact, and we lay the dates out soberly. But a deadline is a poor reason to run a strategic project. Value is a better one.

# What is inside

CHAPTER 1

# Your Forms system still works. That's exactly the problem

## 1. The core you built your business on

Let's start with respect for what is actually there. The Oracle Forms systems we meet are not side applications. They calculate leasing installments, price insurance policies, run warehouses, close financial periods, and register orders that keep production lines moving. They are core platforms: if they stop, the company stops.

Inside them sits something genuinely valuable: **decades of business rules, encoded in PL/SQL and refined against real edge cases.** Every discount exception, every regulatory quirk, every hard-won lesson from operations is in that code. In many organizations, this logic is among the most valuable intellectual property the company owns, even if nobody puts it on the balance sheet.

This guide is not about throwing that away. It is about protecting it. The question is not whether your Forms system deserved its place at the core of the business. It did. The question is whether the way it is delivered and maintained still serves the business that grew around it.

## 2. What “it works” actually costs

“It works, so it costs nothing” is the most expensive sentence in IT. The costs of a standing-still Forms estate are real; they are just scattered across budgets where nobody adds them up:

- **License and infrastructure spend** on a middleware layer (WebLogic, runtime licenses, the servers underneath) that exists only to keep a 1990s architecture running.
- **Integration workarounds:** file drops, nightly batches, and people re-keying data between systems because the platform cannot expose a real-time API.
- **A change queue measured in months,** because every modification to a fragile, poorly documented system is a project in itself.
- **Manual work as a habit:** processes designed around the system's limitations rather than around the business.

**This is consistent with what analysts observe across the market**
Gartner has estimated that organizations spend 60 to 80 percent of their IT budgets simply keeping existing systems running. A business-critical Forms estate sits squarely in that number, and quietly grows it every year.

## 3. Three pressures that only grow

Even if nothing in Oracle's calendar ever changed, three trends work against a Forms estate, and each compounds annually:

- **Cost.** Support fees are indexed on license stacks priced per processor, and the infrastructure underneath ages with the software. The run-rate only moves in one direction.
- **Talent.** Forms is a technology no new developer chooses to learn. The people who know your system are approaching retirement, and the people who built it may have already left. Every year, the market for this skill gets smaller and more expensive.
- **Change capacity.** The business keeps asking for things the architecture cannot deliver: mobile access, real-time integrations, analytics, AI-assisted work. The gap between what is asked and what is possible widens with every planning cycle.

None of these pressures depends on a support date. That is why the decision in front of you is not really "whether" but "when and toward what", and why it deserves to be made for business reasons rather than calendar ones.

## 4. The interface is the liability. The logic is the asset.

Here is the single most important reframe in this guide. The problem with your Forms system is not **what it does.** The rules are proven; the data is trusted; the processes work. The problem is **how it is built, delivered, and maintained:** a client-server presentation layer from the 1990s, a costly middleware stack, and a maintenance model dependent on a vanishing skill set.

Migration, done properly, replaces the liability and keeps the asset. The presentation layer, the architecture, and the maintenance model change. The business logic, the data, and the rules the company runs on stay. Everything that follows in this guide builds on this distinction.

**The interface is the liability. The logic is the asset.**

Migration replaces the liability and keeps the asset.

ASSET / KEEP IT

**What the system knows**

✓ **Business logic in PL/SQL**
– decades of proven rules

✓ **Data model**
– trusted, complete, battle-tested

✓ **Processes**
– refined against real edge cases

Flows unchanged into the new system

LIABILITY / REPLACE IT

**How the system is delivered**

✗ **1990s presentation layer**
– desktop-only client

✗ **WebLogic middleware**
– cost without capability

✗ **Aging infrastructure** – and the maintenance model on scarce skills

Replaced by the modern stack

CHAPTER 2

# End of support? A fact, not a reason.

Most vendors open with the deadline. We would rather you saw it clearly, and then set it aside.

## 1. The dates, soberly

Premier Support for Oracle Forms 12c, as part of Fusion Middleware 12c, ends in December 2026, with Extended Support ending in December 2027. After those dates there are no new security patches or certifications for 12c. Those are facts, and for regulated organizations they matter (we return to this in Chapter 3.5).

**But three further facts belong in the same picture:**

- **Many Forms estates are on far older versions anyway.** 6i, 10g and 11g have been out of support for years, and those systems have not stopped working. End of support changes your risk position, not your uptime.
- **A supported path exists inside Forms itself.** Forms 14c (14.1.2) shipped in December 2024, and under Oracle's current lifetime support policy carries Premier Support to December 2030 and Extended Support to December 2033: a runway stretching well into the next decade.
- **Oracle has moved these dates before.** The 12c Premier Support date itself was pushed back multiple times over the years. Treating the current dates as immovable would be ahistorical.

**End of support is a fact. It is not an argument**


**If the deadline is your only reason, you don't have a reason.**
A deadline unlocks budget; it should not set the goal. The real reasons are on the next page.

## 2. If the deadline is your only reason, you don't have a reason

Here is the trouble with deadline-driven migrations: they optimize for the wrong goal. When the objective is "get off 12c before December", the natural outcome is a like-for-like replica of the old system on a newer runtime. The project ends, the date is met, and the business gains almost nothing: same screens, same workarounds, same change queue, on a stack that merely restarted the support clock.

A deadline is useful for one thing: **unlocking budget.** It concentrates minds and opens a planning window. But once the window is open, the question that should drive the project is not "how do we escape the date" but "what will the business gain". That question has a much better answer than most Forms-era organizations expect, and it is the subject of the next chapter.

> **Key realization:**
> You are not moving away from something. You are moving toward something.

## 3. What about simply upgrading to 14c?

For completeness, one option deserves naming out loud, because it will be raised in every board discussion: upgrade to Forms 14c, reset the support clock, and revisit the question in a few years. It is a legitimate move, and a vendor-neutral guide should say so. It keeps the compliance box ticked and buys time at a relatively modest cost.

Just be precise about what it buys and what it doesn't. An upgrade delivers **none of the gains in Chapter 3:** the cost base stays, the talent pool keeps shrinking, integrations stay batch, and the architecture remains closed to mobile and AI. You will face the same decision again in a few years, with a smaller market of Forms specialists and a deeper backlog of postponed change. Upgrading is the right call mainly when hard constraints (a pending merger, a system already scheduled for decommission, a frozen budget year) rule a real move out. Otherwise it is not a decision; it is a deferral with interest. We return to the strategic options in Chapter 5.

CHAPTER 3

# As-is → to-be: what your business actually gains

Six dimensions, each viewed from the business side. Together they are the real case for migration, and none of them mentions a support date.

## 1. Cost: from license stack to lean stack

Premier Support for Oracle Forms 12c, as part of Fusion Middleware 12c, ends in December 2026, with Extended Support ending in December 2027. After those dates there are no new security patches or certifications for 12c. Those are facts, and for regulated organizations they matter (we return to this in Chapter 3.5).

**AS-IS:**

A classic Forms estate pays for several layers at once: WebLogic Server (list-priced at 25,000 USD per processor for the Enterprise Edition on Oracle's technology price list), Forms runtime licensing on top of it, roughly 22 percent of license value in annual support fees, and the infrastructure that carries it all. In Pretius's cost modeling, a mid-size deployment can exceed 250,000 USD in licensing alone before a single line of business code is considered. Staying on this stack also means staying permanently exposed to Oracle's license audit practices, a risk every CFO in the ecosystem knows well. Much of this spend buys no capability; it keeps the delivery mechanism alive.

→

**TO-BE:**

Migration removes the middleware layer entirely. On the APEX path, the application platform is **included in the Oracle Database license you already own:** no WebLogic, no runtime licenses, no Extended Support surcharges. On the modern-frontend path, the proprietary runtime disappears in favor of open frameworks. On the full-exit path, database licensing goes too. In every scenario, the recurring cost base steps down and becomes predictable.

**What it means for the business**

**A freed, recurring budget line** that a CFO can see and redirect: from keeping the lights on to funding change. This is usually the easiest gain to quantify, and the one that anchors the business case.

## The 5-year view: how a CFO should count it

The business case is a five-year comparison, not a one-year one. On one side: the migration project. On the other: five years of licensing, support fees, aging infrastructure, and scarce-skill maintenance that disappear or shrink after the move. In Pretius's business-case modeling, organizations eliminating Forms and WebLogic typically see **five-year savings of 500,000 to 1,000,000+ USD per server environment** (Pretius figures; your licensing structure decides the exact number). Set against the migration cost ranges in Chapter 4, the payback point for most mid-size estates lands well inside the five-year window, and everything after it is margin.


Cumulative cost, illustrative mid-size estate. Pretius business-case figures. Every estate prices individually.

## 2. People: from hunting experts to hiring from the market

**AS-IS:**

Ask a simple question: if the one person who truly understands your Forms system resigned tomorrow, what would happen? In most organizations the honest answer is uncomfortable. The Forms talent pool is small, aging, and shrinking; recruiting into it is slow and expensive; and the original authors of the system are often long gone. This is key-person risk attached to a business-critical platform.

**TO-BE:**

A modern stack (APEX, Java, React, PostgreSQL, depending on your path) is **hireable from the mainstream market.** Universities teach it, developers choose it, and multiple vendors can support it. Knowledge about the system moves from a few heads into documentation, tests, and code that ordinary teams can read.

### What it means for the business

**Operational continuity.** The bus factor on a core platform stops being a board-level risk. Hiring timelines shorten, salary premiums for scarce skills disappear, and you regain bargaining power with vendors because you are no longer captive to whoever still knows Forms.

## 3. Integration: from workarounds to real-time APIs

**AS-IS:**

Forms was designed before the integrated enterprise. So today's estate typically talks to the rest of the company through nightly batch jobs, file exchanges, database links, and people re-typing data from one screen into another. Each workaround is a small tax on process speed and data quality, paid daily.

**TO-BE:**

A migrated system exposes **real-time REST APIs** and consumes them. It becomes a first-class citizen of your architecture: connected to ERP, e-commerce, BI, and partner systems as events happen, not as files arrive overnight.

### What it means for the business

**Faster end-to-end processes and fewer errors.** Orders, prices and statuses flow between systems in seconds. Beyond efficiency, real-time integration is what makes new digital services possible at all: customer portals, partner APIs, live dashboards. None of it can be built on a nightly batch.

## 4. Experience: from terminal habits to modern work

| AS-IS: | TO-BE: |
| --- | --- |
| The Forms client experience is familiar to veterans and hostile to everyone else. It requires a desktop, often a legacy Java runtime, and weeks of onboarding to learn screen sequences that exist nowhere else. There is no mobile access: nothing for the warehouse floor, the field technician, or the manager approving on the move. | A browser-native, responsive application: usable from any device, consistent with the software people already know from the rest of their lives. Done well, migration **keeps what power users love** (keyboard-first speed, dense data entry) while removing what everyone else struggles with. |

### What it means for the business

**Productivity and retention.** New employees are effective in days instead of weeks. Work happens where the work is, not only at desks. And the daily frustration of fighting a 1990s interface, a real if unmeasured driver of turnover in operational roles, goes away.

## 5. Security & compliance: from audit finding to audit-ready

| AS-IS: | TO-BE: |
| --- | --- |
| An unsupported, unpatched application at the core of the business is no longer a technical detail; it is an audit finding. For financial-sector organizations, DORA (in force since January 2025) makes the management of ICT risk, including legacy systems, an explicit board-level obligation, and national regulators ask the same questions. Customer security questionnaires and cyber-insurance underwriters increasingly ask them too. “We run our core on software past its support window” is a sentence nobody wants to read aloud in an audit committee. | A supported, patched, auditable stack with modern identity and access management (SSO, MFA), current encryption, and clean audit logging. Security posture becomes something you demonstrate, not something you explain. |

### What it means for the business

**Regulatory and reputational risk comes off the board's table.** Audits get shorter, security reviews stop escalating, insurance conversations get easier, and the compliance argument for the migration budget writes itself in regulated industries.

## 6. Agility & AI: from frozen backlog to platform for growth

**AS-IS:**

On a fragile legacy core, every business request becomes a project, so most requests are simply never made. The backlog freezes, and IT is quietly recast as the department of "not possible". Meanwhile, the technologies reshaping competitors' operations (analytics on live data, workflow automation, AI-assisted work) are architecturally out of reach: a closed 1990s client with no APIs and no event stream has nothing for an AI to plug into.

**TO-BE:**

A modern platform where changes ship in iterations, data is accessible for analytics, and AI capabilities (assisted data entry, anomaly detection, conversational access to the system) have real integration points. Every path also opens the cloud option: the new stack can run on-premises or in the cloud, on your schedule rather than the architecture's. You do not have to deploy AI on day one; you gain the **option** to, which the old architecture denies you.

### What it means for the business

**IT stops being the bottleneck of strategy.** The cost of trying new things collapses, and the organization can respond to change in weeks. In a decade where competitiveness will be shaped by how well companies operationalize data and AI, the core platform either carries that or blocks it.

## 7. The as-is → to-be table

The table below compresses this chapter into a single view. It is designed to be lifted out of this guide and shown to a management board as-is.

| Dimension | As-is: Forms estate | To-be: after migration | Business outcome |
|---|---|---|---|
| **Cost** | WebLogic + runtime licenses + 22% support + aging infra | Middleware eliminated; APEX included in existing DB license | Recurring budget freed and predictable |
| **People** | Shrinking Forms talent pool; key-person risk | Mainstream, hireable skills; knowledge in code and tests | Operational continuity; no bus factor |
| **Integration** | Nightly batches, file drops, re-keying | Real-time REST APIs, event-driven | Faster processes; new digital services possible |
| **Experience** | Desktop-only, weeks of onboarding, no mobile | Browser-native, responsive, mobile-ready | Productivity; onboarding in days |
| **Security & compliance** | Unpatched core = audit finding (DORA, regulators, insurers) | Supported, patched, auditable; SSO/MFA | Regulatory risk off the board's table |
| **Agility & AI** | Every change a project; AI out of reach | Iterative delivery; real AI integration points | IT enables strategy instead of blocking it |

CHAPTER 4

# Migrate, don't rebuild

Once the decision to move is made, the next trap opens: “if we're doing this, let's rewrite everything from scratch.” It sounds ambitious. It is usually the most expensive mistake available.

## 1. What a rebuild really means

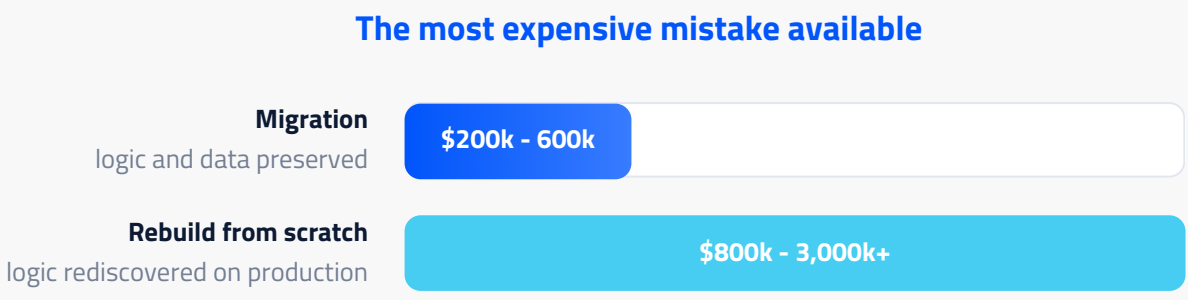
Rebuilding from scratch means abandoning the proven business logic described in Chapter 1 (often decades of rules embedded in PL/SQL) and attempting to rediscover it: from outdated documents, from interviews with people who half-remember, and finally from production incidents when the new system gets an edge case wrong that the old one had handled silently for fifteen years.

It is the most expensive and riskiest route. In Pretius's cost modeling, a from-scratch custom rebuild of a substantial Forms system typically lands at **800,000 to 3,000,000+ USD,** against an indicative **200,000 to 600,000 USD** for migrating a comparable mid-complexity system of around 200 screens (Pretius estimates; every system prices individually). A rebuild is justified in essentially one situation: when the old system's logic no longer matches the business and would have to be scrapped anyway.

**And the quiet third option: doing nothing.** It never appears on a slide, yet it is the scenario every business case is really measured against. Doing nothing has a price too: the run-rate from Chapter 3.1 paid every year, a talent pool that shrinks every year, a backlog of postponed change that compounds every year, and the same decision waiting at the end, to be made on worse terms. Treat “stay as we are” as a costed scenario next to migrate and rebuild, not as a neutral baseline, and the comparison becomes honest.

## 2. What migration preserves, and what it replaces

Migration draws the line exactly where Chapter 1.4 drew it. **Preserved:** the business logic, the data, the rules the company runs on: the asset. **Replaced:** the presentation layer, the architecture, and the maintenance model: the liability. In practice this means lower risk and a shorter timeline for a comparable end result, because the hardest part of any core-system project (getting the rules right) is already done and verified by years of production use.




Pretius estimates. Mid-complexity system, ~200 screens. Every system prices individually.

## 3. Replicate or improve? The scope question

"Preserve the logic" does not mean "copy everything blindly". Decades of growth leave sediment: screens nobody has opened in years, dead code, reports that outlived their readers. A good migration triages the estate: **transfer 1:1** what is proven and used, **simplify** what is obviously overgrown, and **drop** what analysis shows to be dead. It is routine to find that a meaningful share of screens in an old estate is no longer used at all.

The discipline matters in both directions. Cutting dead weight makes the project smaller and the new system cleaner; resisting the urge to redesign everything keeps the project from swelling into the rebuild you decided against. The result should be a system that is **better, not just newer**, without scope creep.

## 4. The real choice is strategic, not technological

Which brings us to the decision that actually determines the project. It is rarely "which technology". Far more often it is three strategic questions:

- Is the value in **owning your own platform,** or in getting off any proprietary platform at all?
- Do you want to **maintain the system yourselves,** or would you rather outsource it?
- Do you ultimately want to **stay in Oracle, or leave it?**

These questions belong to company strategy, not to the IT department alone, and their answers decide the migration path: not cost alone, and certainly not a vendor's comfort zone. The next chapter maps the three possible answers.

CHAPTER 5

# Three exit paths. One right answer for your situation.

Most vendors sell you a destination first, then justify it. The honest sequence is the reverse: analysis first, then the recommendation. The right path depends on your strategy, not on which platform a partner happens to know best.

## Path #1: Stay in Oracle, move to APEX

### WHO IT IS FOR

Oracle-centric organizations with a deep, healthy PL/SQL investment and no strategic intent to leave the Oracle ecosystem. If your database team is strong and the logic is the asset you want to protect, this is the shortest distance between as-is and to-be.

### WHAT YOU GAIN

- **Zero new licensing.** APEX is included in the Oracle Database license you already own; WebLogic and runtime licenses are eliminated entirely.
- **The shortest timeline and lowest risk of the three paths:** the database, the logic, and the team's core skills all stay in place while the presentation layer is rebuilt.
- **Development speed.** Low-code delivery on top of your existing PL/SQL makes new features a matter of days or weeks, not the multi-month efforts Forms required.
- **Every gain from Chapter 3,** from cost through compliance to AI-readiness, arrives on the earliest possible schedule.

### WHAT TO WEIGH

You remain within the Oracle ecosystem, by choice. If board-level strategy calls for vendor independence, this path does not deliver it (though it does not block a later move either: the logic remains portable).

→ **Profile:** lowest risk / shortest timeline / lowest cost / Oracle-bound

---

## Path #2: Keep the database, modernize the frontend

### WHO IT IS FOR

Organizations that trust their Oracle database and proven PL/SQL, but want frontend freedom: a contemporary UX built in the framework that fits their team and stack (Java, React, or a low-code platform such as Mendix), rather than the one any single vendor prefers.

**WHAT YOU GAIN**

- **A trusted core, kept.** The Oracle database and the business logic stay exactly where they are; only the presentation layer changes.
- **Freedom of technology.** The frontend matches your existing skills, standards and enterprise architecture, which matters in mixed-stack organizations.
- **Full modern UX and integration:** responsive, mobile-ready applications and real-time APIs, free of the legacy client constraints that frustrate users today.
- **A wide talent market on the frontend side:** modern web skills are the easiest engineering skills to hire for.

**WHAT TO WEIGH**

Medium risk and timeline: two layers (database and new frontend) must be developed and operated in concert, and API design between them becomes a first-class concern. Runtime licensing for Forms and WebLogic disappears; database licensing stays.

→ **Profile:** medium risk / medium timeline / DB stays, frontend new / partial independence

---

## Path #3: Full exit from Oracle

**WHO IT IS FOR**

Organizations whose strategy is independence from any single vendor, end to end. Both the frontend and the database migrate (typically Oracle to PostgreSQL) onto a fully open-source stack.

**WHAT YOU GAIN**

- **No licensing costs at all:** neither database nor middleware. This is the largest possible reduction in platform cost.
- **No vendor lock-in.** No dependence on a single vendor's roadmap, pricing, or audit practices.
- **A mainstream destination.** PostgreSQL is now the most-used database in the industry's largest developer survey (55.6 percent of 49,000+ respondents, Stack Overflow 2025): this is not an exotic bet, it is the market's default.
- **A modern, integrable stack throughout:** open standards, real-time APIs, contemporary frontend technology.

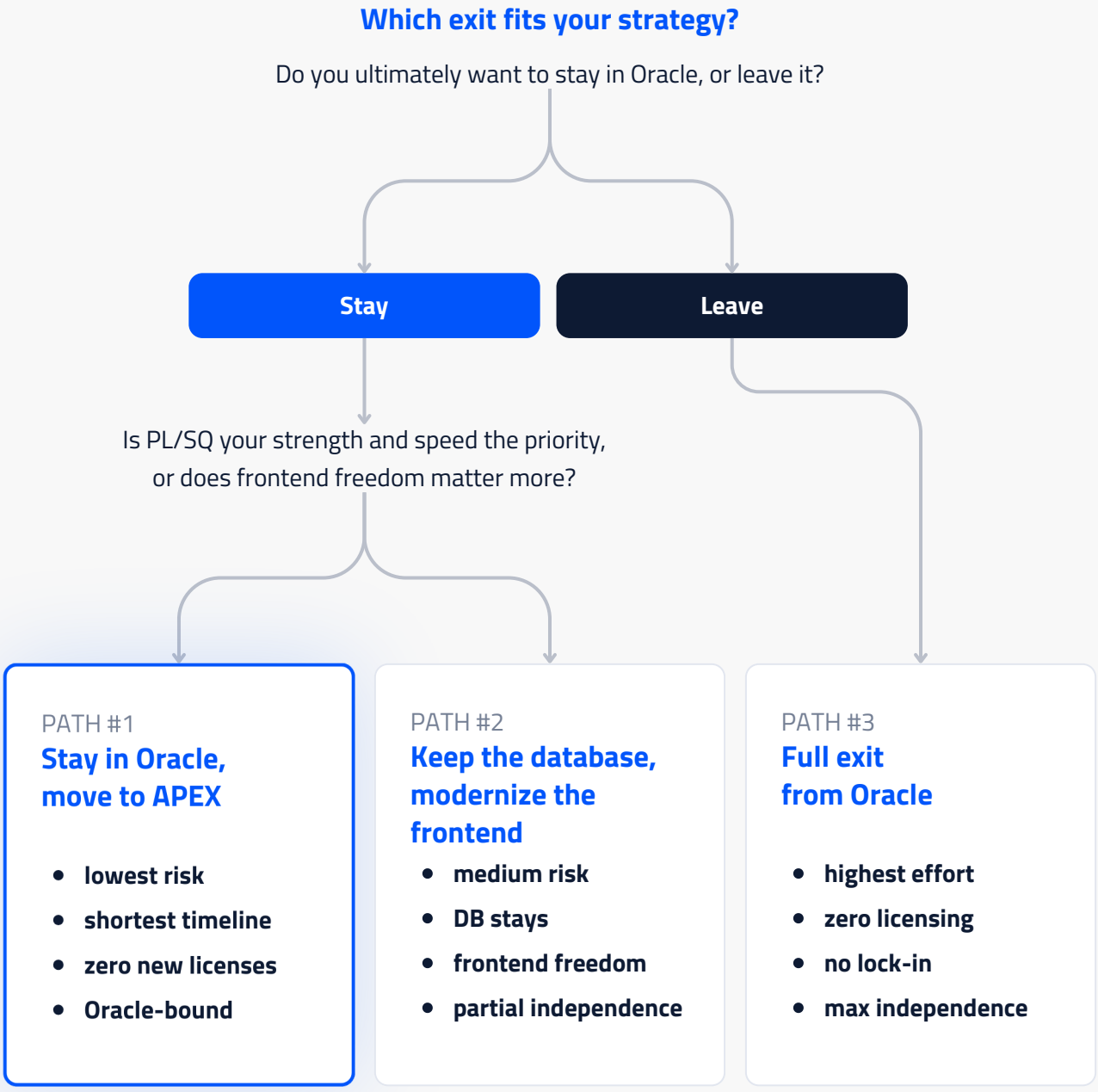
**WHAT TO WEIGH**

Honestly: this is the highest-risk, highest-cost path. Decades of PL/SQL must be converted and re-verified, and at large scale the undertaking requires careful thought and staging. It makes sense when leaving the platform is the strategy, not merely a preference.

→ **Profile:** highest risk and cost / longest timeline / maximum independence / zero licensing

## How to choose: a decision framework

The three strategic questions from Chapter 4.4 map almost directly onto the paths. The table below is a starting hypothesis, not a verdict; the final call belongs to analysis of your specific estate.




Large estates commonly end hybrid: different systems, different paths.
The final call belongs to analysis, not to this page.

CHAPTER 6

# Why companies wait, and how to de-risk the move

If the benefits are this clear, why do most companies still wait? It is rarely the technology. It is uncertainty and risk: three fears that keep a working-but-aging system exactly where it is.

## 1. Three fears that freeze the decision

- **"We have no clear view of the options."** APEX? A new frontend? Leaving Oracle entirely? Each path has very different cost, risk and long-term consequences, and it is hard to know which fits without independent, vendor-neutral analysis.
- **"It works. Better not to touch it."** These are the systems the business runs on. Add the expected cost, the disruption risk, and the effort demanded of your own team, and standing still feels safer than moving.
- **"Nobody knows what's really inside."** Built and patched over decades, often by people who have left, the system is a black box. Without knowing which screens are used, what the logic does, and what is dead code, nobody can scope the migration: so it never starts. In our experience, this is the key blocker.

Each fear is legitimate. Each also has a specific, structural answer, which is what the rest of this chapter describes.

## 2. Analysis first: turn the black box into a map

Before anything is migrated, the existing application should be turned into a single, structured source of truth. The raw material already exists: the Forms definitions themselves, the PL/SQL logic and schemas in the database, whatever documentation survives, and (most tellingly) how users actually work with the system day to day.

Analyzed together, these sources yield a map: which screens exist and **which are actually used, what logic sits behind them, where the decision points are, and what is dead.** From the map follow the things a decision needs: a defensible scope, an effort estimate, a timeline, and a fact-based recommendation of the path.

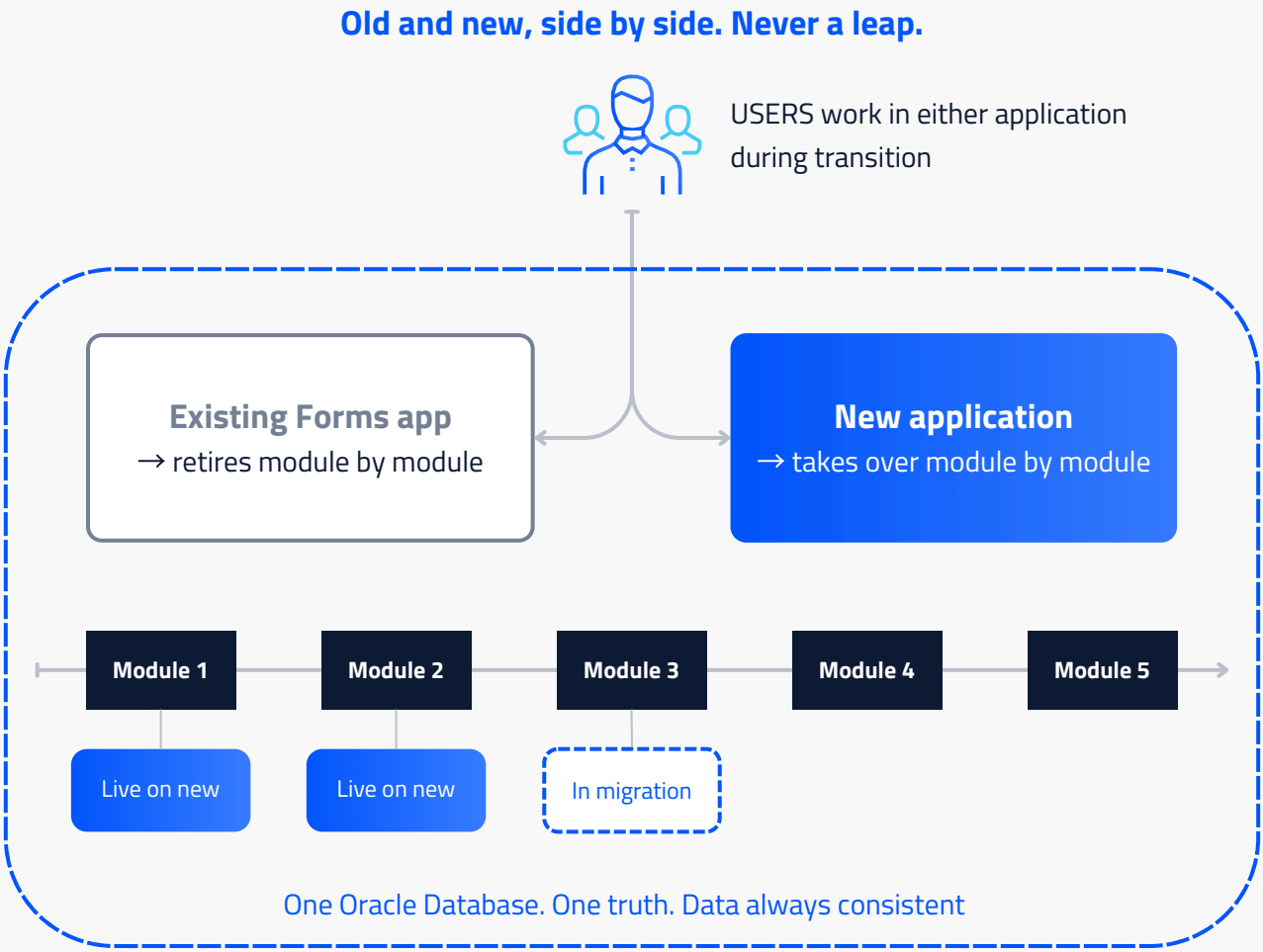
Two properties make this step valuable regardless of what happens next. First, it is **path-neutral:** the same map serves an APEX migration, a frontend rebuild, or a full exit. Second, the knowledge **stays with you:** even if the migration waited a year, the organization would understand its own core system for the first time in a decade. This directly answers fears #1 and #3.

## 3. Module by module, never big bang

A business-critical system is never switched over in one weekend. The safe pattern is phased: start with a **representative module as a proof of concept** (in Pretius practice, typically four to eight weeks), validate the approach on it, then proceed module by module, with subsequent modules measured in months rather than years (Pretius practice figures; analysis sets the exact cadence for your estate). Throughout the transition, **the old and the new application run in parallel on the same database:** users can work in either, data stays consistent, and the organization is never asked to jump into the deep end.

This architecture also answers a question boards rarely ask out loud: **does the business have to freeze while we migrate?** No. Because both applications share one database, the system keeps living during the project: urgent changes remain possible, they simply need scope coordination so they land in the right place (old, new, or both). A migration run this way is a lane change, not a road closure.

The cutover, when it comes, is an administrative step rather than a leap of faith: the new system has already been running the business alongside the old one. This is how fear #2 is answered in practice: not with assurances, but with an architecture in which nothing critical is ever bet on a single day.




The cutover, when it comes, is an administrative step rather than a leap of faith: the new system has already been running the business alongside the old one.

## 4. Prove it's the same system: 1-to-1 verification

"Trust us, it works the same" is not evidence. Verification should be built on tests **derived from the actual behavior of the old system:** the same inputs must produce the same outputs, field by field, calculation by calculation. Where the migration deliberately improves something (Chapter 4.3), additional tests cover the new scope, so improvements are also proven rather than assumed.

The result is an artifact most modernization projects never produce: **documented, repeatable proof that the new system matches the old,** ready for the management board, the auditors, and the regulator. For a core platform, this proof is not a nice-to-have; it is the difference between a controlled change and an act of faith.

## 5. What the project asks of your team

Honesty requires saying this part out loud: no migration of a core system is zero-effort for the client. You will need to make **key users available** for interviews and validation, make **scope decisions** (what transfers 1:1, what improves, what dies), and staff **acceptance testing** windows. A good, analysis-driven process minimizes this burden (facts from the system replace weeks of workshops, and generated tests shrink manual UAT), but it cannot remove it.

Plan for it, and the project runs on schedule. Ignore it, and even the best partner will end up blocked on decisions only you can make. Realistic expectations at the start are themselves a de-risking measure.

One more thing worth planning for, because it decides more projects than any technology: **your Forms team.** The people who keep the current system alive can read a migration as a threat to their roles, and quietly become its most effective opponents. The honest message, and the true one, is the opposite: their PL/SQL knowledge is the asset the whole project is built to preserve, and on the APEX and modern-frontend paths it becomes **more valuable,** not less. Framed as upskilling rather than replacement (with training and, if you choose it, co-creation), the team that knows the system best becomes the migration's strongest engine. It has been done exactly this way before: after one Forms 6i migration, the client's in-house team was trained to maintain and evolve the new system entirely on their own.

CHAPTER 7

# Before you decide: honest questions

The right approach starts with strategy, not technology. These are the questions worth answering internally, before talking to any vendor: including us.

## 1. The self-assessment checklist

Circulate this page to the people who own the system, the budget, and the strategy, and compare answers. Disagreement here is cheaper now than mid-project.

| # | Question | Why it matters |
|---|---|---|
| **1** | Why do we want to move off Forms: what is really driving it? | "The support date" and "the business needs X" lead to very different projects. |
| **2** | Are there objective deadlines or events forcing the timing? | Regulatory dates, hardware end-of-life or contract ends set real windows; vendor calendars do not. |
| **3** | Is this a chance to improve the system, not just replicate it? | Defines scope discipline: what transfers 1:1, what gets fixed, what dies. |
| **4** | Is it an opportunity to unify or change our technology stack? | A migration can consolidate the estate, or quietly add one more stack to maintain. |
| **5** | Do we want to maintain the new system ourselves, or outsource it? | Determines both the target technology and the delivery model (co-creation vs. full service). |
| **6** | Do we ultimately want to stay in Oracle, or leave it? | The single biggest fork: it separates Path #1 from Paths #2 and #3. |
| **7** | What resources do we actually have: stack, licenses, in-house skills? | Existing licenses and skills change the economics of each path materially. |
| **8** | If our key system expert left tomorrow, what would happen? | Measures the real urgency better than any support date. |

## 2. Reading your answers

A few patterns to look for. If question 6 comes back "we are staying in Oracle" and question 7 shows a strong PL/SQL bench, Path #1 is your working hypothesis. If the frontend and UX ambitions dominate questions 1 and 3 while trust in the database is intact, look at Path #2. If question 6 comes back "independence" at board level, Path #3 is on the table, and the honest follow-up is whether the appetite survives its cost and timeline.

And if the answers say "not yet"? Then say so explicitly, and price it: waiting is also a decision, and Chapters 1 and 3 describe what it costs per year. The one option this guide argues against is the silent default: no decision, made annually, by nobody.

CHAPTER 8

# Why Pretius

Everything to this point holds true whoever you work with. This last chapter is about why organizations choose to do it with us.

## 1. No preferred destination

The advice in Chapter 5 is only as trustworthy as the incentives behind it. A partner who only builds APEX will recommend APEX; a partner who only builds Java will recommend Java. Pretius maintains deep teams across **Oracle (APEX and PL/SQL), Java, React, and PostgreSQL** simultaneously: there is no structural reason for us to push any single path. That is what lets us start with analysis and follow the facts: the recommendation is an output of your situation, not of our staffing.

## 2. You own everything

- **Full code ownership.** The code belongs entirely to you, built to engineering best practices with CI/CD (Liquibase, SQLcl) and Oracle ACE-level standards.
- **No Pretius lock-in.** No Pretius licenses, ever. You are never dependent on us to maintain or evolve your own system.
- **Your choice of operating model.** If you want to grow an in-house team, our co-creation model puts your people side by side with ours until they can take the system over. If you would rather outsource the whole thing and not carry the maintenance risk, we support that too. It is an option, not a default.
- **Security you can verify.** We work in line with information-security procedures; our ISO 27001 compliance process is currently being renewed.

## 3. Proof, not promises

Pretius has been delivering software for enterprises for nearly twenty years: **300+ specialists, 70+ clients on four continents, 93 percent of clients becoming long-term partners, and a longest single partnership of 19 years.** Our Oracle investment runs deep: **7 Oracle ACEs on board, the largest ACE representation in Poland,** alongside teams where half the engineers are Java experts. Organizations that trust us with critical systems include BNP Paribas, T-Mobile, Munich Re, Canal+, Play, Philip Morris International, mBank, Sweco, and Deloitte.

| | | |
|---|---|---|
| **300+** | **70+** | **93%** |
| **specialists** | **clients on 4 continents** | **long-term partners** |
| **19 yrs** | **7** | **ISO 27001** |
| **longest partnership** | **Oracle ACEs** | **certified** |

VAN group

## E-VAN, a custom freight management system on Oracle APEX

Freight calculations automated with machine learning for instant reactions on transport marketplaces; route planning automated; one source of truth across the operation; licensing costs (Forms support, WebLogic) cut by a significant margin.

**Industry:** Transportation

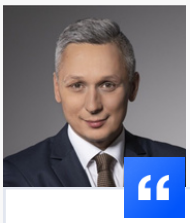
**System:** TMS & CRM

**Technology:** Oracle Database, Oracle APEX

**600+**
**employees using the e-VAN system**

**40**
**locations across Europe**

**6 yrs**
**of partnership**
**1 BA + 4 developers**



**Adam Kasperowicz**
CIO at VAN group

> The new e-VAN system has become an innovative tool and work environment.

Full testimonial: pretius.com/case-study/van-group

# How to start: a low-risk first step

The entry path is designed to validate both the migration direction and Pretius as a partner, before any commitment:

**For selected cases, the estimate and the Proof of Concept can be free of charge**
You verify the direction; we get to know your system and recommend the best approach, risks, and next steps. Either way, you leave the conversation knowing more about your own system than you did before it.

## Let's plan your Forms migration

Start with analysis. Choose the right path. Own the result.


**Przemysław Staniszewski**
**CEO | Oracle ACE** ♠
pstaniszewski@pretius.com
+48 600 800 881



**Bartłomiej Jajkowski**
**Head of Presales**
bjajkowski@pretius.com
+48 793 006 987


TRUSTED BY MARKET LEADERS INCLUDING

BNP PARIBAS · PLAY · CANAL+ · Munich RE · Deloitte. · PHILIP MORRIS INTERNATIONAL · SWECO · ORACLE Partner