Svedberg Open
DISSEMINATION OF KNOWLEDGE

International Journal of Artificial Intelligence and Machine Learning

Publisher's Home Page: https://www.svedbergopen.com/



Research Paper Open Access

Integration Architecture For Cloud-Based Human Capital Management and Talent Acquisition Ecosystems: Identity Resolution, Event Consistency, and Failure Handling Across Vendor Boundaries

Hiren Nayak

Integration Solution Architect Independent Researcher, India.

Abstract
Integrating talent acquisition and human capital management systems is usually treated as a data mapping exercise. It is not. The defining constraint is that the boundary being crossed is contractual rather than technical: the counterparty is a vendor whose data model, release schedule, delivery guarantees, and rate limits are fixed by agreement rather than negotiable by engineering. A mismatch that would take an afternoon to correct inside systems under common ownership becomes a formal change request, a cost, and a wait. This paper sets out an integration architecture for that setting, organized around three problems that the vendor boundary makes structurally harder. Identity resolution must survive the applicant-to-employee transition, at which the authoritative identifier changes and the thread connecting one person's records breaks. Event consistency must be established without distributed transactions, which the architecture achieves by ordering on the effective date of the transaction rather than its creation or entry date, and by making repeated delivery safe rather than trying to prevent it. Failure handling must account for a mode that is worse than an outage, in which the boundary returns success while the payload is lost, leaving hires half-landed and invisible to both sides. The governing principle throughout is that correctness is the integrating party's responsibility and cannot be delegated across the boundary. The architecture is drawn from production integrations spanning four systems and two vendors, processing on the order of 1.2 million applications annually against a global population of approximately 130,000 employees. No performance measurements are reported; the organization's compliance policy precludes release of operational figures, and the paper is framed accordingly as an architecture contribution.

Keywords: integration architecture, human capital management, applicant tracking system, identity resolution, record linkage, event-driven architecture, idempotency, eventual consistency, reconciliation, vendor boundary, SaaS integration, HR technology.

Introduction

Note on scope and evidence

What is production experience. The architecture is drawn from integration solution design across multiple implementation projects, beginning in August 2020 and continuing through a second major programme from June 2023, spanning talent acquisition, applicant tracking, and human capital management systems: four systems across two vendors. The scale context is on the order of 1.2 million applications per year against a global population of approximately 130,000 employees.

What is not reported. No performance, quality, or failure measurement appears in this paper. Records synchronized per period, duplicate and match failure rates, event failure and retry rates, propagation latency, and reconciliation break counts were not systematically measured, and in any case the organization's compliance policy precludes their release. No such figure is estimated or implied anywhere.

What that means for how this should be read. This is an architecture and design-rationale contribution supported by production experience, not an evaluated study. Where the paper argues that one design choice is preferable to another, the argument is from failure mode and consequence, not from measurement. Section 9 states what a measurement programme would capture, so that the claims made here are at least falsifiable in principle.

Confidentiality. No employer, vendor, product, or system is named. Topologies are described by role. The incident in Section 8 is real and is recounted without identifying the organizations, systems, or region involved. No schema, match rule, identifier format, or reconciliation output is reproduced.

1. Introduction

An integration between a talent acquisition platform and a human capital management system looks, on a topology diagram, much like an integration between two systems inside one organization. Records move, events fire, fields map. The diagrams are similar enough that the work is routinely scoped as though the problems were similar.

They are not, and the difference is not primarily technical. When both systems are under common ownership, a mismatch is a bug, and a bug has an owner who can be asked to fix it on either side. The team decides which side is wrong, changes it, and moves on. When one side is a vendor, that option disappears. The data catalogue is theirs and it does not match yours, notwithstanding the existence of interchange standards for HR data [10] whose adoption across vendors is uneven. The release timeline is theirs. The guarantees are whatever the contract says and nothing more. A correction that would have taken an afternoon becomes a formal request, a cost, and weeks of waiting for resource alignment on the other side.

The consequence for architecture is specific: the integrating party cannot fix the boundary, so it must absorb it. Every design decision in this paper follows from that constraint. Where a conventional integration might assume a capability and handle its absence as an exception, an integration across a vendor boundary must assume the absence and treat the capability as a bonus.

Three problems become materially harder under that constraint, and they organize the paper.

Identity resolution. Each system mints its own identifier. A person can be tracked reliably inside recruiting, but at conversion to employee the authoritative identifier changes and the thread connecting their records breaks. Reconstructing that thread requires a composite matching key, and the key is fallible in two directions with asymmetric consequences.

Event consistency. Distributed transactions are unavailable across the boundary. A hire cannot be made atomic across four systems and two vendors. The architecture must therefore establish a consistent ordering without a coordinator, and must remain correct when events arrive twice or out of sequence, because both will happen.

Failure handling. The failure that matters most is not the outage. It is the boundary that returns success while the payload is lost, which leaves a hire half-landed, absent from both the source and the destination, and invisible to any monitoring that trusts the acknowledgement.

Section 2 positions the work. Section 3 characterizes the asymmetry of the vendor boundary and what cannot be demanded across it. Sections 4, 5, and 6 address identity, events, and failure respectively. Section 7 specifies reconciliation. Section 8 analyses a production incident that illustrates the paper's central claim. Sections 9 and 10 state the limits and conclude.

2. Background and positioning

2.1 Consistency without a coordinator

The constraint that no transaction can span the boundary is a specific instance of a general result. Gilbert and Lynch's formalization of Brewer's conjecture establishes that a partition-tolerant system must trade consistency against availability [1], and a vendor boundary is a partition that is permanent by construction rather than occasional by failure. Vogels' account of eventual consistency describes the operational discipline that follows, in which convergence is achieved after the fact rather than guaranteed at the moment of write [2]. Helland states the architectural consequence most directly for this setting: beyond a certain scale, and across independently operated systems, applications must be built without distributed transactions and must instead reason explicitly about uncertainty and repair [3]. That is precisely the position an integrator occupies at a vendor boundary. The saga pattern of Garcia-Molina and Salem provides the classical mechanism, decomposing a long-running operation into steps with defined compensating actions rather than a single atomic unit [4].

Lamport's treatment of time, clocks, and event ordering supplies the reasoning behind the ordering decision in Section 5 [5]. His central point, that in a distributed system there is no single authoritative wall-clock ordering and that a logical ordering must be constructed deliberately, maps directly onto the failure this paper describes: ordering HR events by the time a record was created produces a sequence that reflects the mechanics of data entry rather than the sequence of events in the employee's actual history.

2.2 Record linkage

The identity problem in Section 4 is an instance of record linkage, whose statistical foundation is Fellegi and Sunter's decision-theoretic framework for classifying record pairs as matches, non-matches, or requiring clerical review [6]. That three-way outcome, rather than a binary match decision, is the structure the architecture adopts.

Elmagarmid, Ipeirotis, and Verykios survey duplicate record detection [7], and Christen surveys indexing techniques for scalable linkage [8]. Both establish the trade-off that Section 4.3 makes concrete in this domain: a threshold set to reduce missed matches necessarily increases false matches, and the correct setting depends on the relative cost of the two errors rather than on any intrinsic property of the data.

2.3 The gap

This literature is largely developed for settings in which the integrating party controls the systems, or at least the data, on both sides. Record linkage research generally assumes access to both record sets. Consistency and saga literature generally assumes the compensating action can be executed on both sides.

Neither assumption holds here. The counterparty's identifier assignment cannot be influenced, its internal state cannot be inspected, and its side cannot be rolled back when yours fails. What this paper contributes is the application of these established mechanisms under that additional constraint, together with the design consequences that follow from it, in a domain where the failure modes have direct consequences for a person's pay, access, and first day of employment.

3. The asymmetry of the vendor boundary

3.1 What cannot be demanded

Once a contract is in force, the integrating party receives the vendor's API, the vendor's guarantees, and the vendor's limits. Three specific capabilities are unavailable, and each has a direct architectural consequence.

Transactional guarantees. Inside systems under common control, an operation can be specified as all-or-nothing. Across the boundary each call stands alone. If a downstream step fails after a vendor call has succeeded, the vendor's side cannot be rolled back.

Webhook reliability. Vendor-emitted events are dropped, duplicated, delivered out of order, or simply not fired, and the reason is not observable from outside.

Rate headroom. The vendor sets the ceiling on call volume. Exceeding it results in throttling or rejection, and the urgency of the work has no bearing on the outcome.

Table 1. What cannot be demanded across a vendor boundary, and the architectural response to each. The pattern is consistent: the capability is not obtained, it is compensated for.

Cannot be demanded	Naive assumption it invalidates	How the architecture absorbs it
Transaction across the boundary	A hire either completes everywhere or nowhere	Track state locally, treat every call as possibly half-landed, verify after the fact rather than assuming success, and define compensating actions per step [4]

Webhook delivery, ordering, uniqueness	An event fires exactly once and arrives in order	Treat webhooks as an optimization rather than a source of truth; run a periodic sweep or poll to catch what was missed; make repeated events safe rather than trying to prevent them
Rate headroom	Requests can be issued at the pace the work requires	Queue on the integrating side and feed requests at the vendor's pace; smooth bursts; back off on throttling; accept later completion for lower-priority work

3.2 The governing principle

The three responses in Table 1 share a single principle, and it is the paper's organizing claim.

Correctness is the integrating party's responsibility and cannot be delegated across the boundary. The architecture does not trust the boundary, and it does not attempt to make the vendor behave differently. It assumes the vendor will be unreliable in defined ways and makes its own state authoritative for whether an operation actually completed.

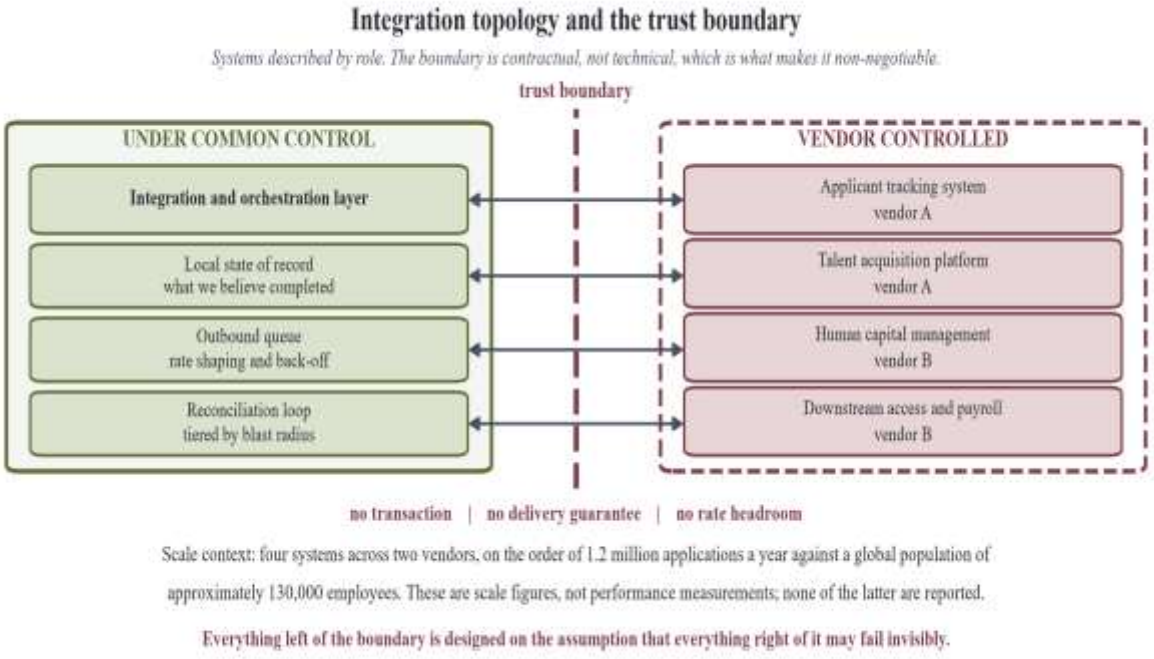


Figure 1. Integration topology and the trust boundary. Four systems across two vendors, described by role rather than by product. The dashed boundary marks where transactional guarantees, delivery guarantees, and rate control cease to be available. Everything to the left of it is designed on the assumption that everything to the right of it may fail in ways that are not observable from the left.

4. Identity resolution across the applicant-to-employee transition

4.1 Where the thread breaks

Every system mints its own identifier. The applicant tracking system assigns a candidate identifier when a person applies. The human capital management system assigns an employee identifier at hire. The two are independent, and neither system has any reason to know about the other's. Cross-domain identity provisioning standards [9] address the synchronisation of identity records between domains, but they presuppose an agreed identifier, which is precisely what is absent at this transition.

Within recruiting, a person can be tracked without difficulty. The break occurs at conversion. At the moment of hire the authoritative identifier changes, and unless something deliberately carries the prior identity forward, the person's history does not follow them across the transition.

This is not an edge case. It is the normal path, traversed by every hire.

4.2 The composite matching key

The mechanism that preserves the thread is to carry the original applicant tracking identifier forward as an attribute of the employee record, and to support it with person attributes that are independent of any system's identifier assignment: name, date of birth, and the full or partial national identifier where policy permits its use for this purpose.

The applicant tracking identifier is the primary link. The person attributes exist because it is not reliable on its own.

Table 2. Components of the composite matching key. No component is sufficient alone, which is why the decision is a weighted judgement over the combination rather than a lookup, and why the three-way outcome of Section 4.3 is necessary.

Key component	Role	Why it is insufficient alone
Originating applicant tracking identifier	Primary link carried forward across conversion	The same person can hold several, particularly on reapplication; the same identifier can occasionally be associated with the wrong person
Name	Corroboration	Not unique; changes over time; inconsistently formatted between systems
Date of birth	Corroboration	Not unique in a large population; frequently mistyped
National identifier, full or partial	High-specificity corroboration	Availability and permitted use are constrained by policy and jurisdiction; partial forms are not unique

4.3 The two failure directions

The key is fallible in two directions, and their consequences are different in kind rather than merely in degree.

Missing a match splits one person into two records. Their history does not follow them. Employee identifiers may be created more than once for the same person, which propagates into payroll as a duplicate population.

Matching too loosely merges two people into one record. Someone inherits access or pay that is not theirs.

Both are serious, but they are not symmetric. A split is a data quality problem with an administrative cost and a repair path. A false merge is a security and privacy incident, and in the pay case a financial one, and repairing it requires unpicking entitlements that have already been granted.

That asymmetry is why the architecture adopts the three-way outcome of Fellegi and Sunter [6] rather than a binary threshold. Confident matches are linked automatically. Confident non-matches create a new identity. The uncertain middle is routed for human adjudication rather than resolved by lowering the threshold until it disappears, because the error that a lowered threshold produces is the more expensive one.

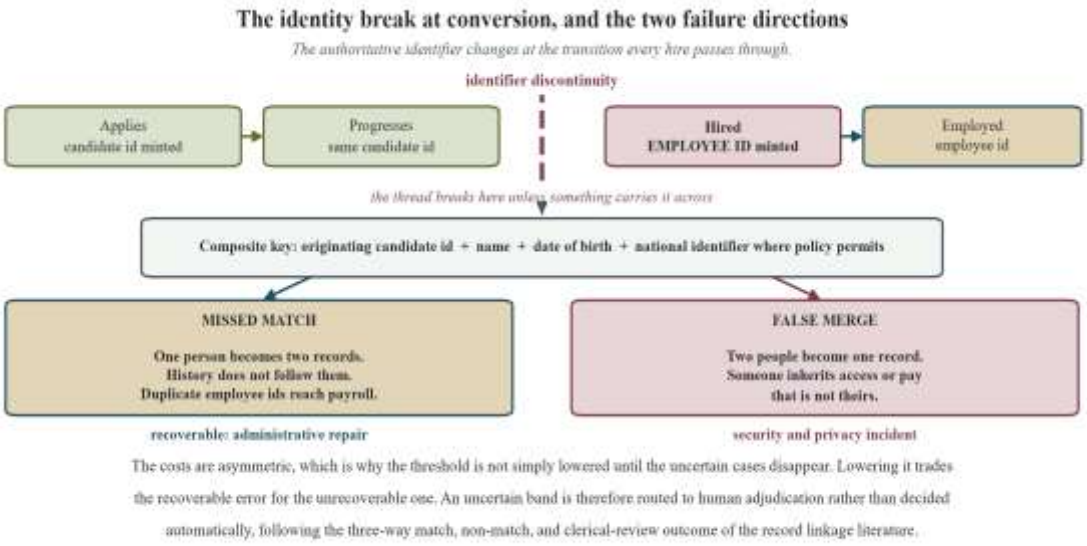


Figure 2. The identity break at conversion, and the two failure directions. The upper path shows the identifier discontinuity that the composite key exists to bridge. The lower panel shows the asymmetry that governs threshold placement: a missed match is recoverable, a false merge grants access or pay that is not the person's.

4.4 The rehire and internal applicant cycle

The hardest case is not the first hire. It is the person who has been through the cycle before.

An internal employee whose profile is terminated may later apply again using a different email address. The applicant tracking system, keyed on that address, has no reason to associate the new application with the prior identity, and mints a new candidate identifier. The composite key must now bridge not one transition but a chain of them, and the person attributes carry more of the weight because the identifier link is broken at the point where a new application record was created.

This case recurs continuously in an organization of any size, and it is the principal reason the person attributes are part of the key rather than a fallback.

5. Event consistency across the boundary

5.1 What must cross

Hire-related events must cross the boundary in both the initial and the amended case: new hire detail, start date, position data, hire confirmation, any subsequent change to those, and withdrawal.

The amended case is the one that determines the design. A start date that changes after the original hire event has propagated is not an exception; it is routine, and every downstream system must converge on the corrected value rather than on whichever version arrived last.

5.2 Ordering on effective date, not creation date

The central design decision is that events are ordered and applied by the effective date of the transaction, not by the date the record was created or entered.

Effective dating is not newly derived here. It is a first-class primitive within established HRIS and payroll platforms, and its general form is standard in temporal data management, where SQL:2011 added application-time period tables for precisely this purpose [12]. The contribution here is narrower and specific to the boundary: recognizing that the effective date is also the correct cross-vendor ordering key, and that adopting it is what converts the unattainable requirement of exactly-once delivery into the attainable one of making at-least-once delivery safe.

The reasoning follows from Lamport's observation that a distributed system has no single authoritative ordering and that the ordering used must be constructed deliberately [5]. Creation date orders events by the

mechanics of data entry. Effective date orders them by the sequence in the person's actual history. These diverge constantly and unavoidably: a retroactive correction is entered after the event it corrects, and a future-dated change is entered long before it takes effect.

An integration ordered on creation date will apply a retroactive correction after a later-entered but earlier-effective change, and downstream systems will settle on a state that never occurred. An integration ordered on effective date converges on the same state regardless of the sequence in which the messages arrived.

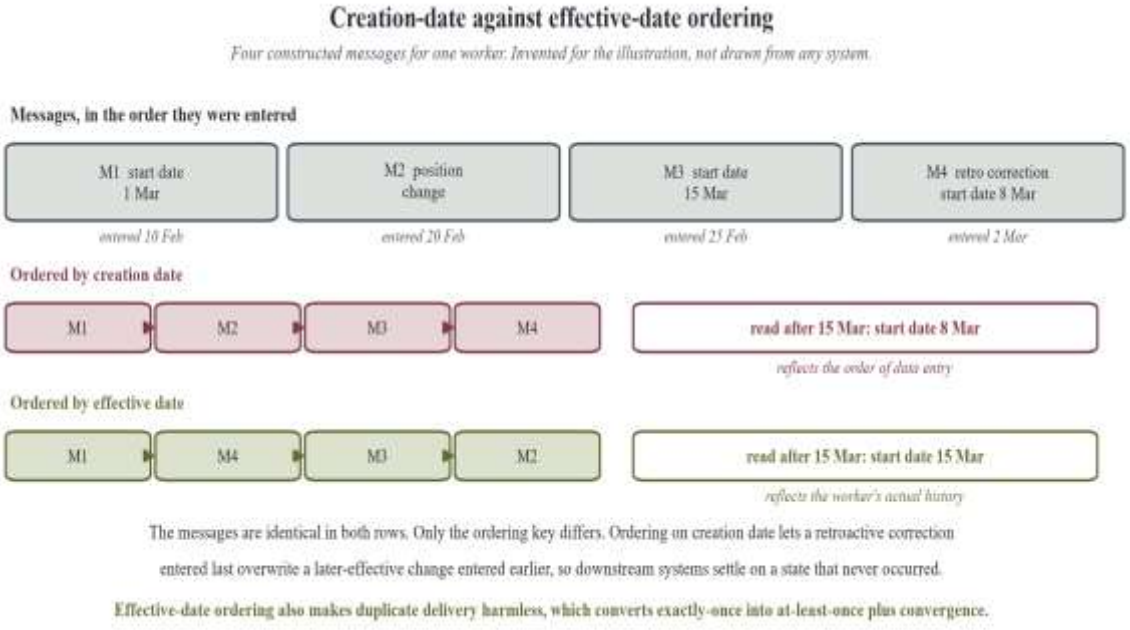


Figure 3. Creation-date against effective-date ordering, for one worker whose start date is corrected retroactively after a later-entered change. Ordering on creation date produces a record that reflects the order of data entry. Ordering on effective date produces the value the employee's record should return when it is read after the last effective date. The messages are identical in both cases; only the ordering key differs.

Reading Figure 3. The four messages are constructed for the illustration. M1 sets a start date of 1 March and is entered on 10 February. M2 is a position change entered on 20 February, taking effect after mid-March. M3 moves the start date to 15 March and is entered on 25 February. M4 is entered last, on 2 March, and asserts a start date of 8 March. Entry date and effective date are separate axes, and M4 is the difficult case in the sense given above: it is entered after the message it corrects, while taking effect earlier in the worker's timeline.

Ordered by creation date, the messages are applied in entry sequence and each overwrites the one before it, so M4 is applied last and the record holds 8 March. Ordered by effective date, M4 sits at 8 March and M3 at 15 March, so M3 is the assertion in force from 15 March onward and a read taken after that date returns 15 March. M2 changes a different field and so never competes with the start date; it moves between the two rows only because its effective date falls last.

What makes the second answer the correct one is that an effective-dated record is a timeline rather than a single cell. Each message asserts what is true from its effective date onward, and reading the record on a given date returns the assertion with the latest effective date at or before that date. Read on 10 March, both orderings return 8 March and agree. Read on 20 March, effective-date ordering returns 15 March while creation-date ordering still returns 8 March, because collapsing the timeline into one cell has destroyed the later-effective fact. The error is not that the wrong message arrived last. It is that the ordering key discarded information the record was supposed to hold, and no downstream system can recover it.

5.3 Idempotency as a consequence, not an addition

A precondition makes this work: events carry absolute state, not deltas. An event asserts that salary is Y with effect from date D, never that salary increases by X. A delta is not safe to replay, because applied twice it doubles, whereas an absolute assertion applied twice yields the same state as applying it once. The architecture requires state-based events for exactly this reason, and the idempotency that follows is a property of that choice rather than of the transport.

Once events are ordered and applied by effective date, duplicate delivery ceases to be a correctness problem. Applying the same effective-dated fact twice produces the same state as applying it once, so a message delivered more than once is redundant rather than harmful.

This inverts the usual framing. Rather than attempting to guarantee exactly-once delivery across a boundary that cannot provide it, the architecture makes at-least-once delivery safe. Given the webhook behaviour described in Section 3.1, the second is achievable and the first is not.

Table 3. Events crossing the boundary. The amendable column determines which events require effective-date convergence rather than one-time application, and the consequence column determines reconciliation priority in Section 7.2.

Event	Direction	Amendable	Consequence of loss
New hire detail	Talent acquisition to HCM	Yes	Person hired but absent downstream; no account, access, or pay record
Start date	Talent acquisition to HCM, then onward	Yes, frequently	Onboarding, provisioning, and equipment sequenced against the wrong date
Position data	Talent acquisition to HCM	Yes	Incorrect reporting line, entitlements, and cost allocation
Hire confirmation	HCM to downstream	No	Downstream systems never begin provisioning
Withdrawal	Talent acquisition to HCM	No	Provisioning proceeds for a person who is not joining
Termination and transfer	HCM to downstream	Yes	Access persists after it should have been removed, per Section 7.3

6. Failure handling

6.1 Three failure modes, of which one is genuinely dangerous

Slow produces backlog. It is visible, it is measurable from the integrating side, and it degrades rather than breaks.

Down produces an explicit delivery error. It is disruptive but honest: the failure is known at the moment it occurs and can be retried.

Silent rejection is the dangerous mode. The boundary returns a success response, commonly a 200, while the payload is lost in transit or discarded on the far side. One side records successful delivery. The other has no record of having received anything. Nothing raises an error, because from the perspective of every component involved nothing failed.

The operational consequence is disproportionate. Hires end in limbo, onboarding is delayed, equipment orders do not trigger, and the discrepancy typically surfaces close to the start date, when the remaining

options are all expensive and manual. A person's first day should not be the point at which the integration is discovered to have failed weeks earlier.

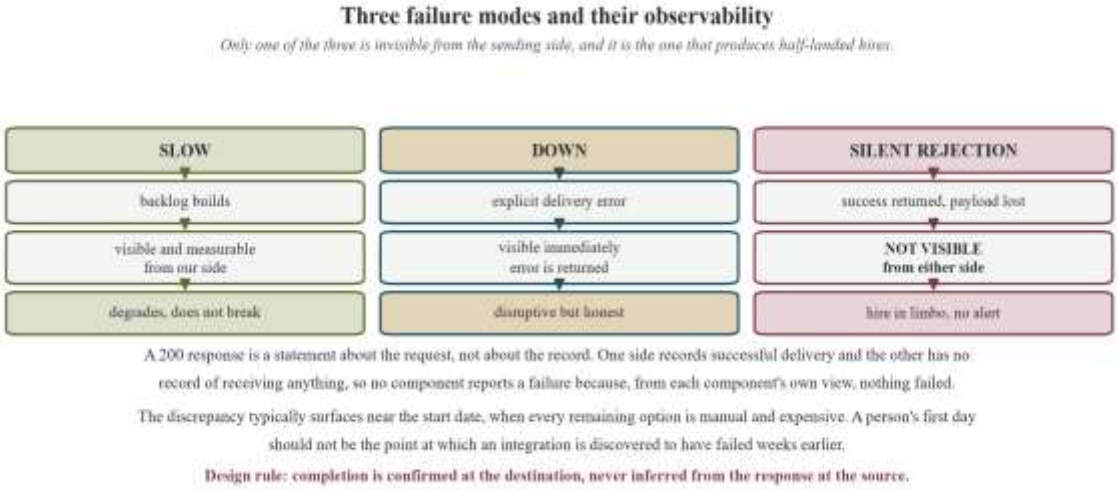


Figure 4. The three failure modes and their observability. Slow and down are visible from the integrating side. Silent rejection is not, which is why acknowledgement cannot be treated as evidence of arrival, and why the architecture requires positive confirmation at the destination rather than a success response at the source.

6.2 What follows: acknowledgement is not arrival

The design rule that answers silent rejection is that a successful send is not evidence of a successful receipt. Completion is established by confirming the record's presence at the destination, not by the response code returned at the source.

This is the same principle Helland arrives at from the transactional direction [3]: in the absence of a distributed transaction, the integrating party must assume a call may have partly landed and verify afterwards rather than trusting the acknowledgement.

6.3 Retries, timeouts, and escalation

Retries preserve order, since an out-of-order retry can reintroduce the sequencing problem that effective-date ordering exists to solve.

Timeouts are actionable rather than merely logged. A timeout that increments a counter no one reads is equivalent to no timeout at all.

Escalation is proactive. When retries are exhausted, or a timeout persists, the technical team is engaged before the business consequence becomes visible, rather than after someone notices a missing hire. The distinction between a defect caught by monitoring and one caught by a manager on a start date is largely the distinction between engaging early and engaging late.

A fallback path exists for the case where automated delivery cannot be completed before a start-date deadline. A half-finished hire is resolved through a defined manual route rather than left to the automated retry cycle, on the reasoning that the deadline is externally fixed and the retry cycle is not guaranteed to beat it.

7. Reconciliation

7.1 Detection by comparison

Divergence is detected by pulling state from both sides and comparing: who is active, what access is held, what each record asserts. Anything that does not agree is a divergence, assessed against agreed quality expectations for the exchanged data [11]. A person active in one system and not the other. A field asserting one value upstream and another downstream. A record present on one side and absent on the other.

This is the operational expression of eventual consistency [2]. Convergence is not guaranteed by the write path, so it is established by a repair loop that runs independently of it.

7.2 Cadence by blast radius

Reconciliation runs at two tiers, and the assignment rule is explicit: the higher the blast radius, the more often it is checked.

Table 4. Reconciliation tiers. The tiering exists because uniform frequency is either too expensive at full scope or too slow for the highest-consequence fields. Cadence follows consequence rather than convenience.

Tier	Scope	Cadence	Rationale
Lightweight sweep	Access entitlements and termination status	Hourly to several times daily	Highest consequence of drift; a security exposure that grows with time undetected
Full reconciliation	All synchronized records and fields	Nightly, during low traffic	Complete coverage where cost and contention permit it

7.3 The mismatch that recurs

The most common divergence is stale access following a status change. Someone is terminated, moved, or transferred in the human capital management system, and remains active or retains prior entitlements in a downstream application, because that system never received the update, or received it and dropped it silently in the manner of Section 6.1.

It is both the most frequent mismatch and the most consequential, because it is a security exposure rather than a data quality irritation. Duplicates arising from a poor identity match are the second most common, per Section 4.3.

The architecture's reconciliation loop is built around the stale access case specifically. It is the reason the lightweight sweep exists as a separate tier rather than being folded into the nightly run.

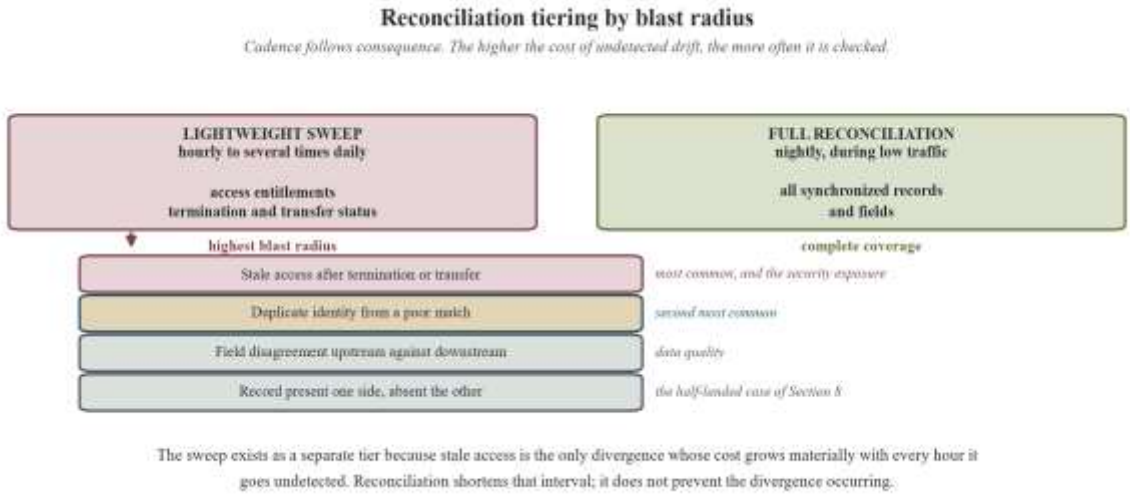


Figure 5. Reconciliation tiering by blast radius, and the divergence classes each tier is designed to catch. The stale-access path is drawn separately because it is the case the loop is built around: it is the most common divergence and the only one whose cost increases materially with every hour it goes undetected.

8. Incident analysis: a timing collision with a vendor maintenance window

8.1 What happened

New hire data originating from a global business unit was to flow into a staging layer before onward delivery to the human capital management system.
The unit transmitted hires during a window in which the vendor was running scheduled maintenance on the regional server serving that path. The sends were issued. The far end was not available to receive them. Rather than failing cleanly, most of the hires stalled midway: they left the source and never fully landed in staging.

8.2 Why it was not detected

The records were no longer pending in the origin system, because from its perspective they had been sent. They had not arrived in the destination. They occupied an intermediate state that no system considered its responsibility to report, and nothing surfaced them.
The affected people were, in the organization's own systems, hired and simultaneously invisible. No accounts, no access, no payroll record, and no alert indicating that anything was missing. The gaps were discovered manually, one at a time.

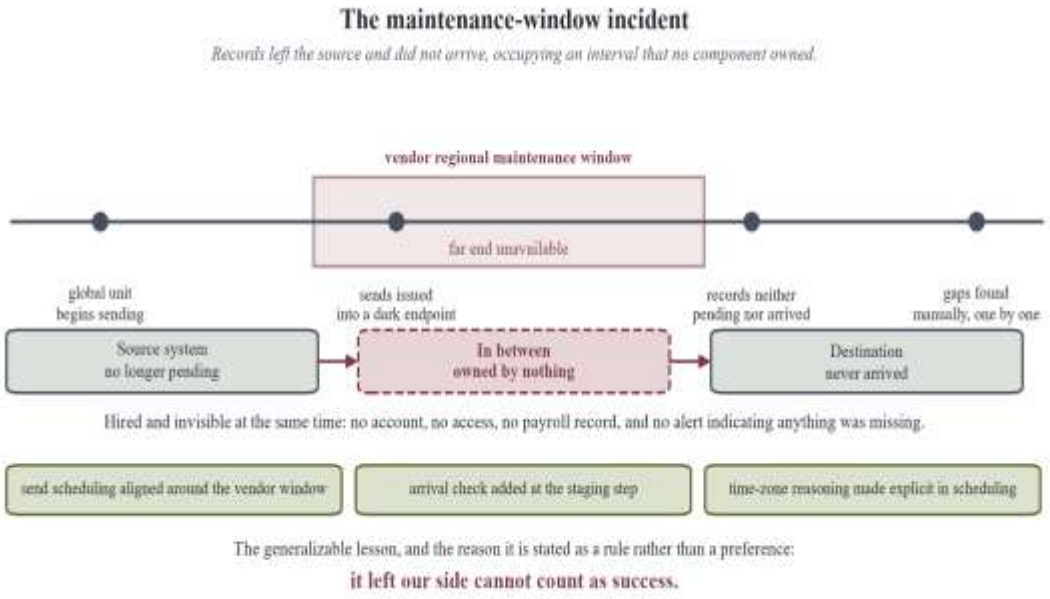


Figure 6. The incident. Time-zone separation placed the sending window inside the vendor's regional maintenance window. Records left the source and did not arrive, occupying an intermediate state that neither side monitored. The absence of an alert was not evidence of success; it was the consequence of no component owning the interval between departure and arrival.

8.3 What changed, and the generalizable lesson

Three changes followed.
The vendor's maintenance window stopped being treated as the vendor's concern. Send scheduling was aligned around it, so data was not pushed into a system known to be unavailable.
A check was added at the staging step, so that a hire which left the source and did not land was flagged rather than disappearing quietly.
Time-zone reasoning was made explicit in scheduling, since the sending unit and the receiving infrastructure were in different regions and the receiving end could be dark while the sending end was mid-business-day.
The generalizable lesson is the design rule of Section 6.2, and this incident is the reason it is stated as a rule rather than as a preference: it left our side cannot count as success. Arrival must be confirmed at the destination. Every element of the architecture in this paper, the reconciliation loop, the positive-confirmation requirement, and the effective-date convergence that makes re-delivery safe, is downstream of that single observation.

9. Limitations

No measurement. The paper reports no operational figures, for the reasons given in the scope note. Every comparative claim is argued from failure mode and consequence rather than demonstrated by data, and should be read on those terms.

Single organizational context. The architecture derives from integrations across four systems and two vendors in one organizational setting. The vendor boundary constraints of Section 3 are general to commercial SaaS integration, but the specific balance struck between automated matching and human adjudication reflects one organization's tolerance for the two error directions.

The matching key is policy-dependent. The composite key's strongest component is the national identifier, whose availability and permitted use vary by jurisdiction and by internal policy. Where it is unavailable the key is materially weaker, and the uncertain band requiring adjudication widens.

Reconciliation detects, it does not prevent. The loop of Section 7 shortens the interval during which a divergence persists. It does not stop the divergence occurring, and a divergence introduced and repaired between two sweeps is invisible to it.

What a measurement programme would capture. So that the claims here are falsifiable in principle: match rate and adjudication volume, with the split between the two error directions; event replay and duplicate rates; propagation latency from effective date to downstream application; reconciliation break counts by class and by tier, with time-to-detection; and the count of deliveries acknowledged at source but unconfirmed at destination, which is the quantity that would have surfaced the Section 8 incident on the day it occurred.

10. Conclusion

The recurring error in scoping HCM and talent acquisition integration is to treat the vendor boundary as an implementation detail rather than as the defining constraint. Once the counterparty is a vendor, the data model, the release cadence, the delivery guarantees, and the rate ceiling all move outside the integrating party's control, and no amount of engineering on the near side changes them. What the architecture can do is stop depending on them.

Three design positions follow, and each is a response to a capability that cannot be obtained.

Identity is reconstructed rather than assumed, because the authoritative identifier changes at exactly the transition that matters most, and the composite key that bridges it is fallible in two directions whose costs differ. A missed match splits a person's history. A false merge hands someone another person's access or pay. Threshold placement follows that asymmetry, which is why an uncertain band is adjudicated rather than resolved by relaxing the rule until it disappears.

Ordering is established on the effective date of the transaction rather than the date of entry, because there is no authoritative wall-clock sequence across the boundary and the ordering must therefore be chosen deliberately. Choosing effective date makes duplicate delivery harmless as a by-product, which converts the unattainable requirement of exactly-once delivery into the attainable one of making at-least-once safe.

Failure handling treats acknowledgement and arrival as different facts. This is the position the paper argues most strongly, because it is the one the incident in Section 8 was caused by not holding. A boundary that returns success while losing the payload defeats every monitor that trusts the response code, and the people affected were hired and invisible at the same time, in the organization's own systems, until someone found them by hand.

None of this is measured, and the paper says so throughout rather than at the end. What is offered is a design rationale grounded in production experience and in one instructive failure, together with a statement of what would have to be measured to test it. The single sentence that carries the architecture is the one the incident produced: it left our side cannot count as success.

References

[1] S. Gilbert and N. Lynch, "Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services," ACM SIGACT News, vol. 33, no. 2, pp. 51-59, 2002. doi: 10.1145/564585.564601

[2] W. Vogels, "Eventually consistent," Communications of the ACM, vol. 52, no. 1, pp. 40-44, 2009. doi: 10.1145/1435417.1435432

[3] P. Helland, "Life Beyond Distributed Transactions," ACM Queue, vol. 14, no. 5, pp. 69-98, 2016. doi: 10.1145/3012426.3025012

[4] H. Garcia-Molina and K. Salem, "Sagas," in Proc. 1987 ACM SIGMOD International Conference on Management of Data, 1987, pp. 249-259. doi: 10.1145/38713.38742

[5] L. Lamport, "Time, clocks, and the ordering of events in a distributed system," Communications of the ACM, vol. 21, no. 7, pp. 558-565, 1978. doi: 10.1145/359545.359563
[6] I. P. Fellegi and A. B. Sunter, "A Theory for Record Linkage," Journal of the American Statistical Association, vol. 64, no. 328, pp. 1183-1210, 1969. doi: 10.1080/01621459.1969.10501049
[7] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios, "Duplicate Record Detection: A Survey," IEEE Transactions on Knowledge and Data Engineering, vol. 19, no. 1, pp. 1-16, 2007. doi: 10.1109/TKDE.2007.250581
[8] P. Christen, "A Survey of Indexing Techniques for Scalable Record Linkage and Deduplication," IEEE Transactions on Knowledge and Data Engineering, vol. 24, no. 9, pp. 1537-1555, 2012. doi: 10.1109/TKDE.2011.127
[9] P. Hunt, K. Grundy, and E. Wahlstroem, "System for Cross-domain Identity Management: Protocol," RFC 7644, Internet Engineering Task Force, 2015. doi: 10.17487/RFC7644
[10] HR Open Standards Consortium, "HR Open Standards Specification," HR Open Standards.
[11] International Organization for Standardization, "ISO 8000, Data quality," ISO, Geneva.
[12] K. Kulkarni and J.-E. Michels, "Temporal features in SQL:2011," ACM SIGMOD Record, vol. 41, no. 3, pp. 34-43, 2012. doi: 10.1145/2380776.2380786