Chapter 1

The Problem of Time Leakage in High-Frequency Marketplace Pricing

Formal definitions, failure modes, and diagnostics for tail-risk modeling at time-zero

Revision date: 2026-01-14

Prepared as part of the Slow21 Tail-Risk Survival Modeling Thesis

Abstract

High-frequency marketplaces emit rich observational traces (edits, scrapes, visibility, and eventual sales). The same richness makes temporal leakage the dominant failure mode in marketplace machine learning: fields that look “predictive” frequently encode information only revealed after the decision time t_0. This chapter formalizes the time-zero boundary, develops a practical taxonomy of leakage pathways, and explains why leakage is especially pernicious for long-tail outcomes (e.g., “slow-to-sell” beyond 21 days). We close with deterministic diagnostics and pipeline design constraints that allow downstream modeling—including survival models and tail classifiers—to be evaluated meaningfully.

1.1 High-frequency marketplaces as stochastic systems

Consumer-to-consumer (C2C) marketplaces behave like open stochastic systems. Supply arrives as a stream of listings; demand manifests through heterogeneous buyer arrivals; and the state of any given listing evolves through edits, visibility changes, and eventual sale or abandonment. Every listing is therefore a partially observed process whose outcome is realized in the future.

The platform’s data exhaust is temporally ordered. A single listing can produce dozens of observations (scrape snapshots, price edits, status transitions). Modeling failures often trace back to a single conceptual error: treating this panel data as if it were a static table.

Time leakage is the specific failure mode where the model sees information that would not have been available at scoring time. It is not a subtle issue: leakage can dominate model selection, hyperparameter tuning, and feature importance, producing a high-performing artifact in offline metrics that collapses under live traffic.

Definition 1.1 — Time leakage

A learning pipeline exhibits time leakage when any feature used by the model at decision time t_0 depends on events or measurements that occur at times $t > t_0$. Equivalently, the feature vector is not measurable with respect to the information available at t_0.

1.2 The time-zero boundary and admissible information

Let each listing be indexed by i, with decision time $t_{0,i}$. The sale time is a random variable T_i. In a marketplace, the goal is typically to estimate either (i) the conditional distribution of T_i given time-zero information, or (ii) tail probabilities such as $P(T_i - t_{0,i} \geq \tau \mid \text{information at } t_{0,i})$.

$$X(t_0) \text{ admissible} \iff X(t_0) \text{ is } \mathcal{F}_{t_0}\text{−measurable}$$

Equation (1.1): Admissibility—features must be measurable with respect to the information available at time-zero.

Formally, define a filtration $\{F_t\}$ capturing all data that could be observed up to time t. A feature vector $x_i(t_{0,i})$ is admissible if it is $F_{t0,i}$-measurable. Operationally, this

means $x(t_0)$ can be computed deterministically from the event log restricted to timestamps $\leq t_0$.

The Slow21 label is a thresholded survival event at horizon τ=21 days:

$$Y_\tau(t_0) = \mathbf{1}\{T - t_0 \geq \tau\}, \qquad \tau = 21 \text{ days}$$

Equation (1.2): Tail classification at horizon τ (Slow21 when τ=21 days).

Even when labels are defined correctly, leakage enters when the feature computation violates the time-zero boundary. In marketplaces, the most common culprits are lifecycle timestamps (sold_date, last_seen), mutable states (status, inactive flags), and aggregates computed "as of now" rather than "as of t_0".

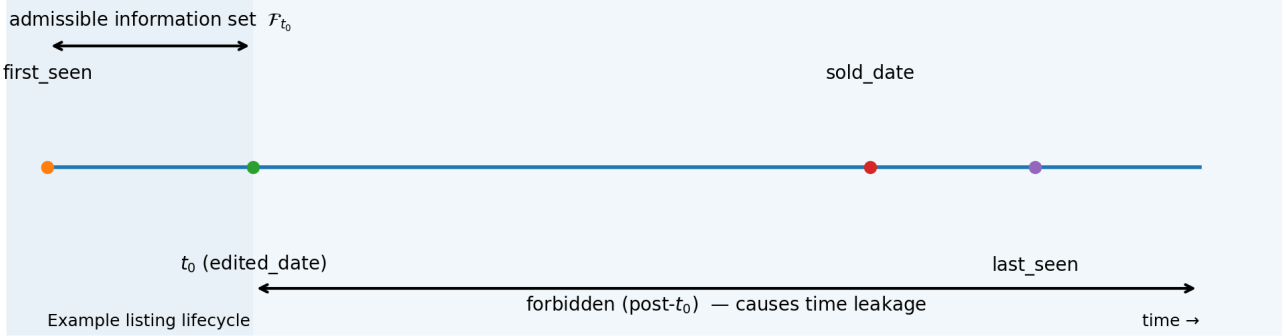

Figure 1.1: Listing lifecycle timestamps and the admissible information boundary at time-zero.

1.3 Leakage mechanisms in marketplace datasets

In high-frequency marketplaces, leakage is rarely a single bug. It is an ecosystem of failure modes that compound across sources, joins, refresh cycles, and evaluation design. Four practical categories cover most incidents:

- Direct outcome leakage. Features that are deterministic (or near-deterministic) functions of the target. Examples: sold_date, sold_price, duration_hours, "is_sold", or post-sale statuses.
- Post-t_0 lifecycle leakage. Features derived from events after time-zero: price edits after t0, visibility changes, last_seen updates, and "inactive" flags that require future observation.
- Pipeline / time-of-query leakage. Non-determinism caused by evaluating at query time (now(), current_date) rather than binding to t0, or by joining to mutable reference tables without snapshotting.
- Evaluation leakage. Leakage induced by splitting/filtering/labeling decisions that implicitly use future information (e.g., random splits over time; labeling items

whose τ-window has not elapsed).

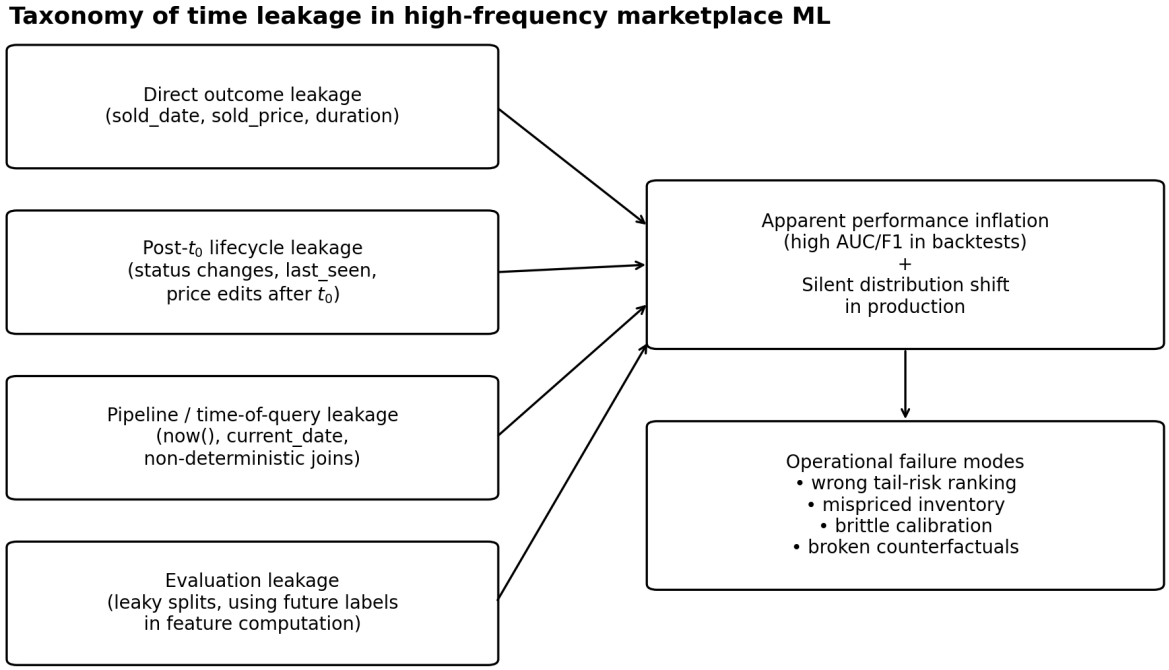

Figure 1.2: A taxonomy of leakage pathways and their operational consequences.

1.3.1 Panel data and the “latest row” trap

Most marketplace sources are not one-row-per-listing. They are append-only logs or snapshot panels. A common anti-pattern is to join a listing-level table to the “latest available row” in a panel. If “latest” means latest at query time, it almost always includes post-t0 edits.

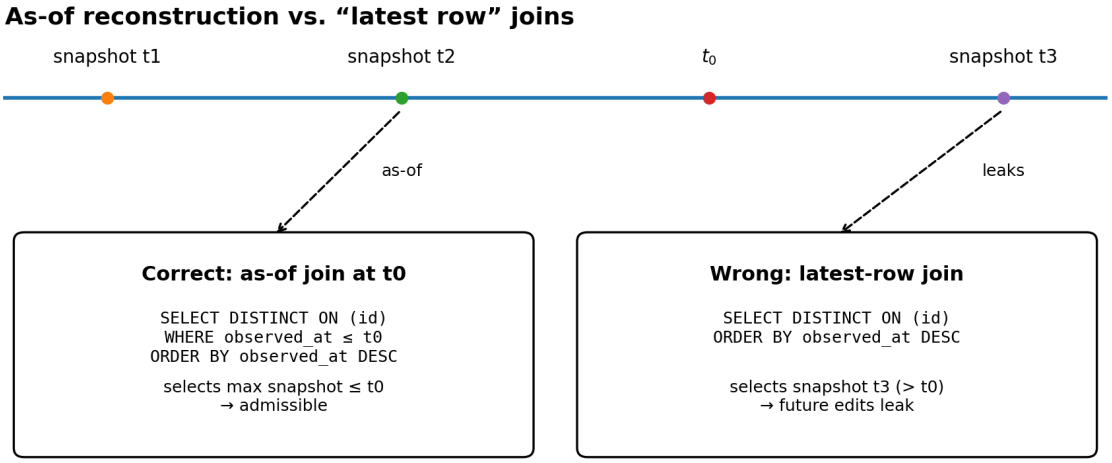

Figure 1.4: Correct as-of joins require restricting to observations with timestamp ≤ t0.

The distinction matters because leakage can be introduced even when join keys look correct. For a listing with multiple snapshots, selecting the most recent snapshot without an explicit ≤ t0 filter imports future state transitions (price edits, text edits, images added, seller profile changes) into the feature vector.

1.4 Why the tail amplifies leakage

Tail outcomes are defined by waiting. To decide whether a listing is "slow," one must observe it for at least τ days. This creates two compounding effects:

- Right-censoring and label availability. For recent listings, $Y_\tau(t0)$ is not yet observable. Any pipeline that labels them anyway must peek into the future (leakage) or distort the sample by dropping them.
- Near-deterministic proxies. Operational timestamps correlate extremely strongly with slow/fast outcomes. If any such timestamp leaks past t0, the model learns a shortcut that does not exist at scoring time.

A canonical example is a feature that checks whether the listing sold within the horizon τ. This is unavailable at t0 but trivially available later. If it enters training (even indirectly), it collapses the problem:

$$x_{\text{leak}}(t_0) = \mathbf{1}\{\text{sold\_date} \le t_0 + \tau\} \;\Rightarrow\; Y_\tau(t_0) \approx 1 - x_{\text{leak}}(t_0)$$

Equation (1.3): A leakage variable that nearly determines the Slow21 label.

Figure 1.3 shows the resulting pathology in a toy survival process: adding a single post-t0 indicator drives standard metrics toward 1.0 without improving the real decision problem.

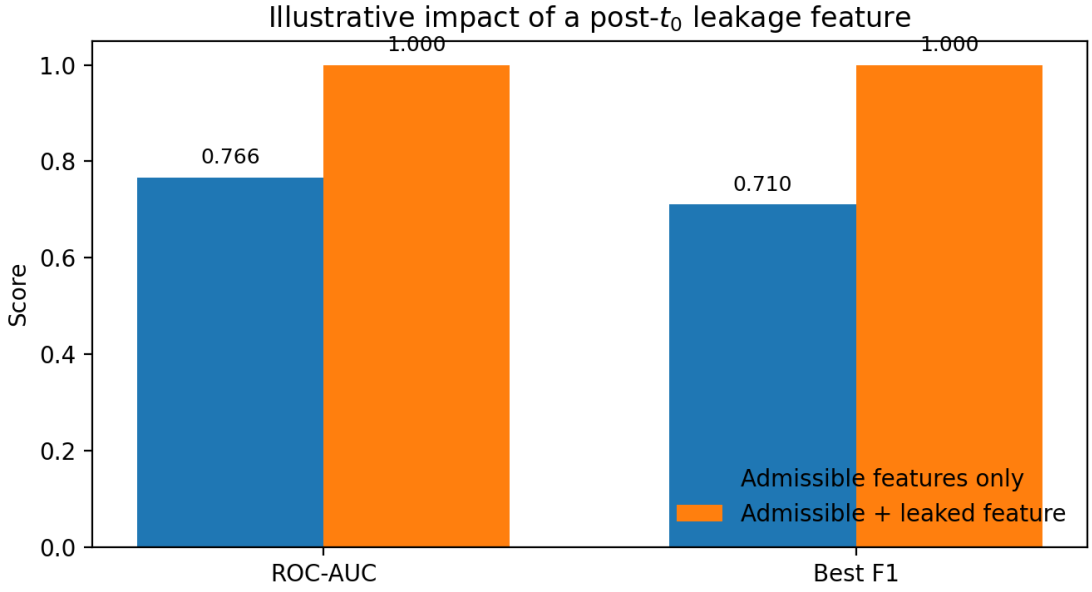

Figure 1.3: Illustrative simulation—post-t0 leakage produces unrealistically high backtest scores.

1.4.1 The generalization gap created by leakage

Leakage is not a small perturbation; it changes the learning problem. The model selected by minimizing backtest loss with leaked features is generally not the model that minimizes deployment loss.

$$R_{\text{deploy}}(f) = \mathbb{E}[\,\ell(Y, f(X_{\text{adm}}))\,] \;\neq\; \mathbb{E}[\,\ell(Y, f(X_{\text{adm}}, X_{\text{leak}}))\,] = R_{\text{backtest}}(f)$$

Equation (1.4): With leakage, the objective optimized offline is not the objective faced online.

Cross-validation does not automatically protect against this failure. If leakage is present in the feature table, it is present in both train and validation folds, so the leakage signal does not break under resampling. Consequently, feature selection, Optuna tuning, and SHAP interpretation can all converge to a leakage-dominated solution.

1.4.2 The “not-yet-slow” region and censoring bias

At any fixed dataset extraction time t_max, labels for a τ-horizon outcome are only observable for listings with t0 ≤ t_max−τ. The remaining listings are right-censored: we do not yet know whether they will exceed the horizon.

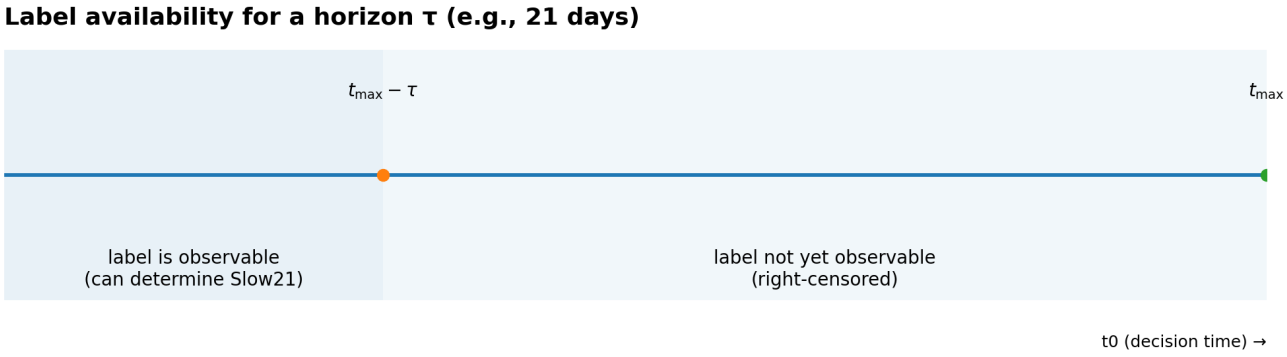

Figure 1.5: Label availability—near the dataset end, Slow21 cannot be observed without peeking into the future.

A naive workaround—dropping the censored region—changes the time distribution of training data and can bias the learned tail rate. A principled alternative is to use survival modeling (explicit censoring) or to enforce a strict “older than τ” gate when constructing classification labels.

1.5 Leakage as a systems problem

Leakage is not only a feature engineering issue; it is a systems issue. Common engineering practices in data warehouses can inadvertently violate the F_{t0} boundary:

- Materialized views refreshed out of order. A downstream view may incorporate post-t0 updates if it is refreshed after upstream tables advance.
- Non-idempotent ETL. If a transformation depends on clock time or unpinned reference tables, reruns change historical rows (“dataset drift by recomputation”).
- Asymmetric joins. Joining a t0-frozen table to a non-frozen one (e.g., “latest listing row”) silently injects future information.
- Implicit filters. Filtering on “status='live'” at query time answers “is live now?”, not “was live at t0?”.

1.5.1 Inventory features: when “live now” is not “live at t0”

Inventory (stock) features are useful because they proxy local competition: how many comparable listings were simultaneously visible to buyers. However, inventory is one of the easiest places to leak. A query like WHERE status='live' uses the current state, not the historical state.

A leak-resistant definition uses listing live intervals: for each listing define live_end = min(last_seen, sold_date) (when sold_date exists) and treat the listing as live at t0 if first_seen ≤ t0 ≤ live_end. Stock counts at t0 are then groupwise counts over live intervals containing t0.

Leak-resistant stock features can then be computed as:

• stock_n_sm4_gen: # live listings in the same (super_metro_v4_geo, generation) segment at t0
• stock_n_sm4_gen_sbucket: # live listings in the same (super_metro_v4_geo, generation, sbucket) segment at t0
• stock_share_sbucket: stock_n_sm4_gen_sbucket / NULLIF(stock_n_sm4_gen, 0)

These definitions are admissible because they require only events timestamped ≤ t0 (first_seen) and an upper bound on visibility (live_end) constructed from observed last_seen and sold_date. Crucially, the computation is expressed as an as-of containment query, not a present-state filter.

1.6 Practical diagnostics: detecting leakage before modeling

Before any modeling, leakage must be treated as a falsifiable hypothesis. The following diagnostics are deterministic and can be automated as unit tests in the data pipeline.

Diagnostic	What it catches	Typical implementation
Time token scan	now(), current_timestamp, current_date in view closure	Static scan over view definitions
As-of invariance	Historical rows change when recomputed later	Dataset hash drift checks across refreshes
Leakage canaries	sold_date / duration / last_seen usage	Strict allow-list / deny-list gates
Temporal correlation	Feature acts as proxy for time index	corr(feature, epoch(t0)) + per-day summaries
Split sanity	Random splits over time	Forward-chaining splits anchored on t0

Table 1.1: Deterministic leakage diagnostics suitable for automated pipeline checks.

When these checks are institutionalized, leakage becomes a tractable engineering constraint rather than an endless debugging exercise. The remainder of this thesis builds on that principle: every feature store is treated as a time-indexed function evaluated at t_0, and the pipeline enforces that contract mechanically.

1.7 Summary

Time leakage is the central obstacle in high-frequency marketplace modeling. The combination of panel data, mutable listing states, and delayed outcomes makes it easy to time-travel accidentally. In the tail, the problem is amplified: the definition of “slow” depends on future observation, so post-t0 timestamps become near-perfect shortcuts.

A production-grade system therefore treats time-zero admissibility as a first-class constraint. It reconstructs as-of states, models censoring explicitly, and subjects feature pipelines to deterministic certification tests. With this foundation in place, subsequent chapters can focus on model structure (Slow21 gating, AFT survival models) rather than on artifact-driven backtest inflation.