Representation choices become maintenance constraints

Key takeaways

1. A schema decides which future changes are cheap, destructive, or impossible.
2. Separate source, candidates, decisions, and views by their lifecycles.
3. Round-trip tests with unknown fields stop a rewrite from erasing data.

Common misunderstandings of data representation

Which claim would you defend right now? Choose before the evidence.

"A file-format choice is mostly syntax; the data model can evolve later."

"A filename or list position is a durable identity for a record."

"Updating one known field cannot erase unrelated data during a rewrite."

Keep your choice. Look for the mechanism or counterexample that would make it fail.

One caption slot made reruns destructive

Before: one mutable slot

```
![Figure 3](figures/fig3.png)

Caption: throughput rises to
32 threads, then plateaus.

# rerun(generator="new-model")
# replaces this text in place
```

What the rerun did

```
papers       = 204
captions     ≈ 2,100
old variants = 2 models
new variants = 1 model

# pipeline succeeded
# working tree lost old captions
# quality drop noticed 2 days later
```

The overwrite was not an implementation accident. The schema could represent only one answer, so every new answer had to destroy the old one.

source: local: slides/gen_854_case_paper_library.py; paper-library/HISTORY.md

A lossy extraction replaced the figure as source of truth

The extracted table read as precise data; the source figure shows overlapping series

Model-produced table

```
matched load    Equalizer    Junction
     ...           lower       higher
     ...           lower       higher
     ...           lower       higher

claimed result:
Equalizer is consistently 15–40% lower
```

Human inspection of Figure 8a

```
the two noisy series overlap
and interleave across the range

no clean monotonic separation
no defensible 14.7% advantage

verdict: the extracted values are false
```

The chart is the source; an extracted table is a generated candidate. Stored as plain rows with no link back to the figure, the candidate became the only version later readers could reach.

source: paper-tools/HISTORY.md, 2026-03-09; Equalizer OSDI 2026 review

One plausible table became two confident conclusions

Two agents treated the generated CSV as ground truth

stage	what happened	why it looked trustworthy
vision model	estimated overlapping chart series	returned precise rows and numbers
reviewer agent	argued that the gap opens at high load	cited the generated table
paper-QA agent	claimed a 14.7% throughput advantage	independently reused the same artifact
human check	compared claims with the source PNG	saw overlap rather than separation

Agreement downstream was correlated error, not corroboration: both agents read the same generated file, and nothing in it pointed back to the figure.

source: paper-tools/HISTORY.md, 2026-03-09

A schema defines which state transitions are possible

operation	one mutable slot	versioned collection
run another model	overwrite	append candidate
compare quality	old answer unavailable	compare side by side
accept a winner	implicit replacement	explicit status change
rollback	recover from backup/git	select prior accepted id

Choose a representation by the changes the system must survive, not by which syntax is shortest today.

Separate facts that have different lifecycles

layer	example	mutation rule
source	figure bytes + paper id	immutable; address by digest
generated candidate	model caption + request receipt	append-only
accepted artifact	candidate id + reviewer decision	new decision supersedes; never erases
derived view	caption embedded in markdown	replaceable; rebuild from records

Raw evidence, generated proposals, human acceptance, and presentation views are not the same kind of data. Putting them in one slot couples their lifetimes.

An append-only sidecar makes reruns non-destructive

Source identity, generated history, and acceptance remain separate

```
{
  "schema_version": 2,
  "figure_id": "sha256:8f4b...",
  "candidates": [
    {"id": "cap-001", "model": "gemini-2.5-flash",
     "text": "...", "created_at": "2026-02-10T18:31:00Z"},
    {"id": "cap-002", "model": "gemini-3.1-flash-lite",
     "text": "...", "created_at": "2026-04-11T09:12:00Z"}
  ],
  "acceptance": {"candidate_id": "cap-001", "decided_at": "2026-04-13T14:20:00Z"}
}
```

The markdown caption becomes a cache of the accepted record. Delete and rebuild the view; preserve the evidence that chose it.

Stable identity cannot depend on a filename or list position

Fragile identity

```
figures[2]              # third today
"Figure_1" in filename # also matches
Figure_10
paper / "fig3.png"     # breaks on rename

# real consequence:
# 92 papers had descriptions orphaned
```

Content + domain identity

```
figure = {
  "paper_id": doi,
  "figure_label": "3",
  "asset_sha256": sha256(png),
}

key = (paper_id, figure_label,
       asset_sha256)
```

Order is a view. A filename is a locator. Neither is a durable record identity.

source: local: slides/gen_854_case_paper_library.py

Missing, null, empty, and zero are different states

Defaults can erase the distinction your recovery logic needs

```
record = {
  # "caption" absent       -> producer has not run
  "caption": null,          # producer ran; no usable result
  "text": "",              # result exists and is empty
  "confidence": 0.0,       # measured value is zero
  "accepted": false        # explicit decision
}

if "caption" not in record: enqueue_generation()
elif record["caption"] is None: route_to_review()
```

Define presence semantics in the schema and test them. Do not let a deserializer silently turn “unknown” into an ordinary default.

Load–modify–save is a migration, even when one field changes

A typed schema is also a filter: unknown fields survive only by design

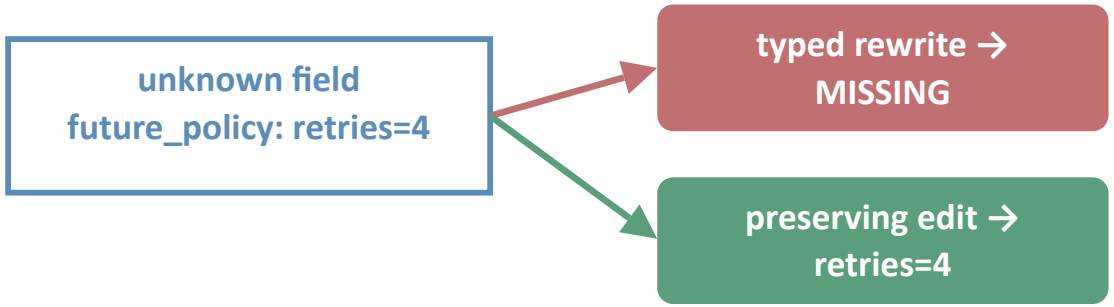
Typed rewrite drops what it cannot represent

```
@dataclass
class Job:
    name: str
    command: str

raw = yaml.safe_load(path.read_text())
job = Job(**known_fields(raw))
job.command = "new-command"
path.write_text(yaml.safe_dump(asdict(job)))

# unknown fields vanish
```

A real scheduling tool lost fields this way. If you do not own the full schema, a surgical edit may be safer.

Preserving edit

```
raw = yaml.safe_load(path.read_text())
before = deepcopy(raw)
raw["command"] = "new-command"
assert_unknown_fields_preserved(
    before, raw, changed={"command"}
)
atomic_write(path, yaml.safe_dump(raw))

# or use a schema-aware migration
```



source: local: slides/_archive/gen_856_case_scheduling.py

Test preservation as a property, not a hand-picked example

Fixture includes data the current editor does not understand

```
def test_edit_preserves_unknown_fields(tmp_path):
    original = {
        "command": "old",
        "future_policy": {"retries": 4, "jitter": true},
    }
    path = write_yaml(tmp_path, original)
    change_command(path, "new")
    after = read_yaml(path)
    assert after["command"] == "new"
    assert after["future_policy"] == original["future_policy"]
    assert set(after) == set(original)
```

Protocol Buffers preserves unknown binary fields for old and new readers, but JSON conversion can discard them. Test every representation boundary.

source: protobuf.dev/programming-guides/proto3/#unknowns

Pick a format that enforces the dominant invariant

need	good default	watch for
human-edited config	YAML/TOML + schema	implicit types, comments, unknown-field rewrites
API boundary	JSON + JSON Schema	duplicate names, number limits, absent vs null
flat interchange	CSV + explicit dialect/schema	types, embedded newlines, missing values
query + transaction	SQLite	keys, constraints, migrations, indexes
compact evolving wire	Protobuf	field-number reuse, JSON conversion, presence

Syntax does not create a contract. Pair the representation with a schema, validation, migration policy, and round-trip tests.

source: rfc-editor.org/rfc/rfc8259; yaml.org/spec/1.2.0; protobuf.dev/programming-guides/proto3

Evolve schemas additively, then migrate deliberately

Migration preserves old meaning and makes uncertainty explicit

```
def migrate_v1_to_v2(v1: dict) -> dict:
    assert v1["schema_version"] == 1
    old = v1.pop("caption", None)
    return {
        **v1,
        "schema_version": 2,
        "candidates": candidates_from_legacy(old),
        "acceptance": accepted_legacy(old),
    }
```

Keep migrations pure and repeatable. Preserve old identities; Protobuf explicitly warns never to reuse retired field numbers.

source: protobuf.dev/programming-guides/proto3/#deleting-fields

You vs. the Agent: a durable representation

Your Responsibility	Agent's Responsibility
Name identities, invariants, lifecycles, and compatibility promises	Generate schemas, validators, migrations, and test fixtures
Decide which data is append-only and which views are replaceable	Inventory readers and writers before a format change
Approve destructive migrations and rollback evidence	Run round-trip and compatibility matrices across versions
Choose the preservation tests at every representation boundary	Produce receipts that connect outputs to source and generator versions

Shared: the format stores bytes; the lifecycle model decides what may be lost

Key Takeaways

1. A representation determines which future changes are cheap, destructive, or impossible
2. Separate source evidence, generated candidates, acceptance decisions, and replaceable views by lifecycle
3. Version migrations and test round trips with unknown fields so old readers do not erase new data

Make the rerun non-destructive

10 min

- A caption pipeline keeps one record per figure and overwrites it on every run:
- figure: cache-hits.pdf · caption: “Hit rate vs. cache size” · model: model-a · run: 2026-03-14
- A rerun with model-b proposes “Cache hit rate saturates near 64 MB” — after an editor already accepted the model-a caption.
- Sketch a record alone for two minutes, then merge designs in a group of three: preserve source identity, candidates, acceptance, and provenance.
- Exchange records. The other group simulates a model-c rerun containing one unknown future field and marks anything lost; revise once.
- Deliverable: the final record, one round-trip assertion, and the destructive case the revision prevents.