Towards Semantics-Aware Resilience in Composable Data Systems

Frédéric Loulergue and **Jolan Philippe**

Université d'Orléans, INSA Centre Val de Loire, LIFO UR 4022

September 30^{th}, 2026

Running example

Example

A service computes **SUM(sales) by region** every five minutes for operational monitoring.

After an incident, it must run on a smaller secondary site with fewer machines and no GPU accelerators.

When a data system degrades, can it still answer *with a known meaning*?

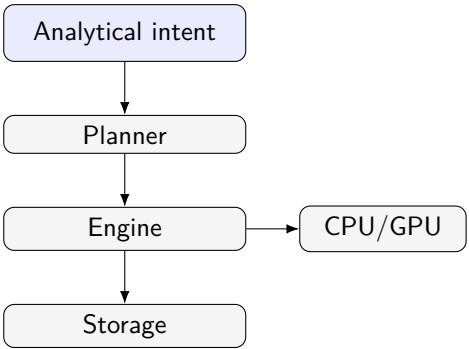
1. Why composability creates an opportunity.
2. Why naive fallback risks semantic drift.
3. How semantic contracts control degradation.
4. What research challenges remain.

The starting point: data systems are no longer monoliths

Modern data platforms are assembled from reusable parts:

- query planners,
- execution engines,
- storage backends,
- hardware targets such as CPU or GPU,
- external services.

This composability is usually used for **performance** and **extensibility**.



Running example: the sales-monitoring pipeline may switch planner rules, operators, engines, or hardware.

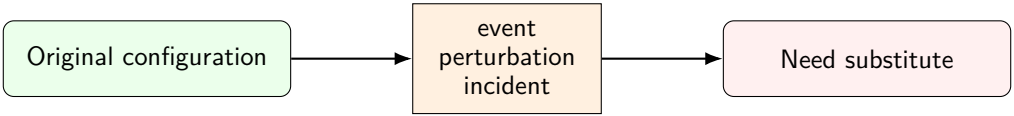
But the operating context can collapse

Normal mode

- Primary site available.
- GPU accelerators available.
- Refresh every 5 minutes.
- Exact result preferred.

Degraded mode

- Secondary site only.
- Fewer machines.
- No GPU accelerators.
- Same monitoring need.



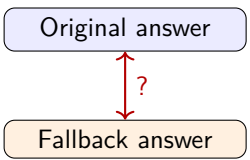
Can we run something cheaper? what meaning does the new answer have?

Fallback can change the meaning of the computation

- CPU instead of GPU may preserve exactness.
- Sampling may preserve speed but introduce error.
- Dropping late data may impact completeness.
- Changing ordering assumptions may change results.
- Using fewer tools may reduce coverage or cross-checking.

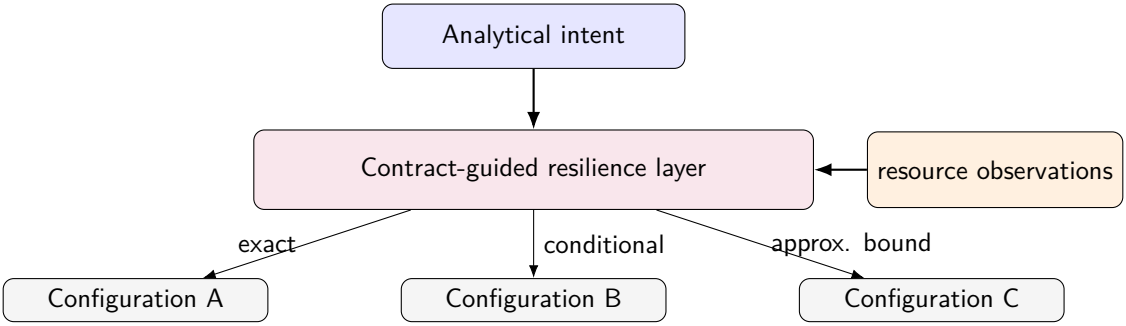
Semantic drift

A substitute looks operationally valid, but no one knows whether it preserves the analytical intent.



Running example: if the system reports SUM(sales), users need to know whether it is exact, conditional, or approximate.

Main idea: put contracts between intent and execution



Key idea

Adaptation becomes an explicit semantic choice, not an emergency change from the initial configuration.

Three classes of semantic guarantees

Class	Meaning	In the running example
Exact	Same intended output.	CPU implementation of the same SUM after losing GPU.
Conditional	Same behavior only if assumptions hold.	Equivalent aggregation only if duplicates are absent, ordering is irrelevant, or schema assumptions hold.
Approximate	Quality is deliberately relaxed with an explicit bound.	Sampling or sketches give a fast estimate of SUM(sales) with a stated error bound.

Resilience = strongest guarantee still feasible under the current constraints.

In the example

Option 1: exact CPU execution

- ✓ Preserves meaning.
- ▲ May miss the 5-minute refresh target.
- ▲ Higher latency under fewer machines.

Option 2: approximate execution

- ✓ Keeps the refresh rhythm.
- ▲ Introduces controlled error.
- ✓ Contract records the bound.

What the controller must decide

Which substitute preserves the most valuable semantics under the new environments ? with constraints ?

Three scopes for reconfiguration

Vertical

Change components or parameters inside one stack.

- CPU/GPU reassignment
- exact vs sampled aggregation

Horizontal

Coordinate several tools or services for the same task.

- reduced tool set
- less coverage
- weaker cross-checking

Hybrid

Combine both dimensions.

- approximate first result
- refine later if resources recover

Running example: the sales monitor is primarily vertical, but the same idea scales to multi-tool exploratory analytics.

Next steps: Research agenda

1. **Contract design**
   What exactly does a contract describe: operator, pipeline components, or task?
2. **Decision-making**
   How to balance latency, energy, quality, and semantic status?
3. **Benchmarks**
   How to measure controlled degradation under crisis scenarios?
4. **Mechanized reasoning**
   Which substitutions can be checked by proof rather than trust?

Proposed first formal target: Apache Spark

Apache Spark A single ecosystem with many ways to express and execute data analytics.

Many analytical paradigms

- Spark SQL / DataFrames: relational analytics
- GraphX: graph analytics
- MLlib: machine learning
- Structured Streaming: real-time analytics

Many programming interfaces

- Scala
- Java
- Python
- R
- SQL

Proposed first formal target: Apache Spark

Apache Spark A single ecosystem with many ways to express and execute data analytics.

Many analytical paradigms

- ...

Many data sources

- Files: Parquet, ORC, CSV, Avro, etc.
- Databases through JDBC
- Hive and external storage systems

Many programming interfaces

- ...

Many execution choices

- alternative transformations and plans
- partitioning and data placement
- batch vs. incremental execution
- different operator implementations

Formalization target

A core of Spark transformations, with mechanically checkable lemmas for **exact substitutions**.

Conclusion

Composable data systems can degrade gracefully only if they know **what meaning survives** each reconfiguration.

Composability	**Contracts**	**Resilience**
multiple ways to execute	known semantic relation	controlled degradation

Claim

Exact, conditional, or approximate substitutions are not new. The purpose is to unify them in a **resilience-oriented contract layer** for composable data systems. With fallback choices to be semantic, explicit, justified, and auditable.

Thank you