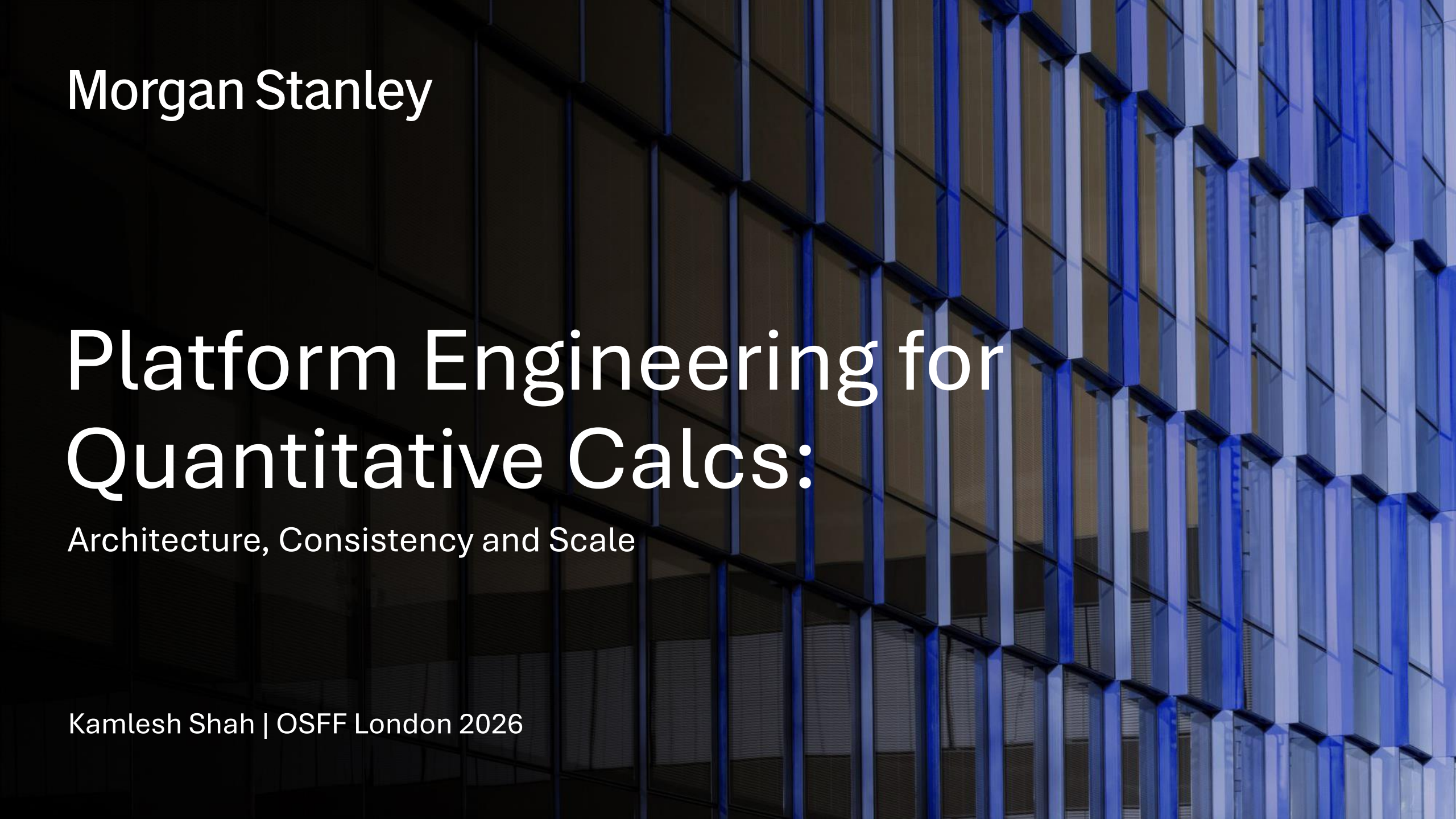
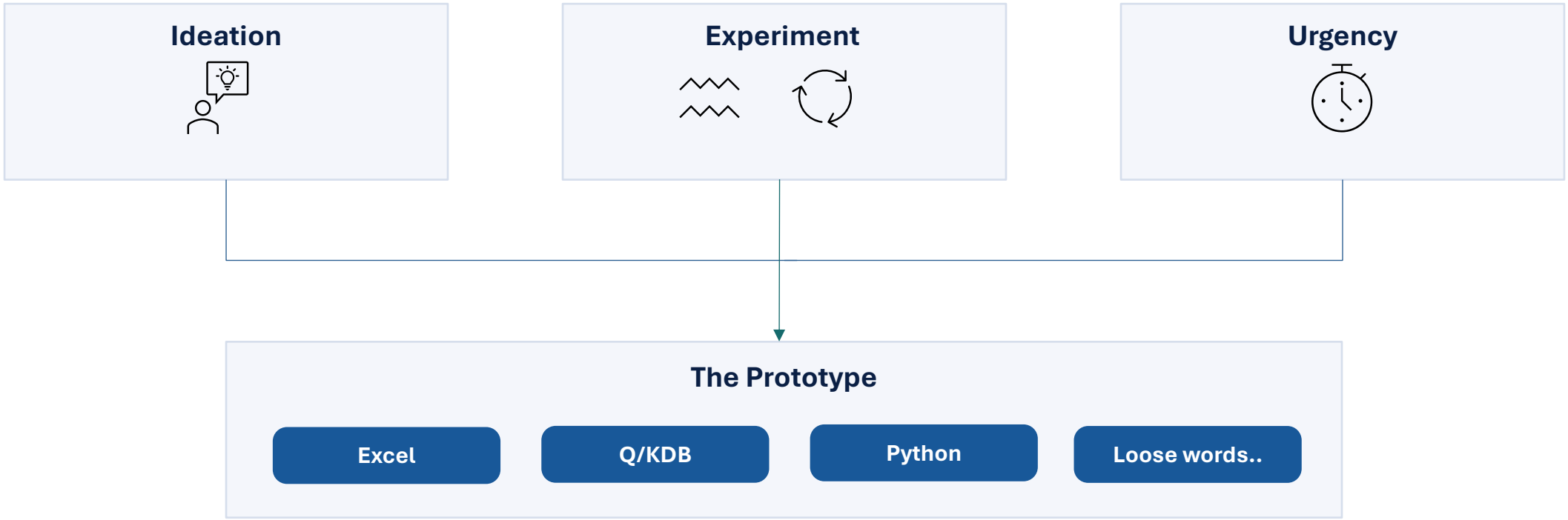


Platform Engineering for Quantitative Calcs:

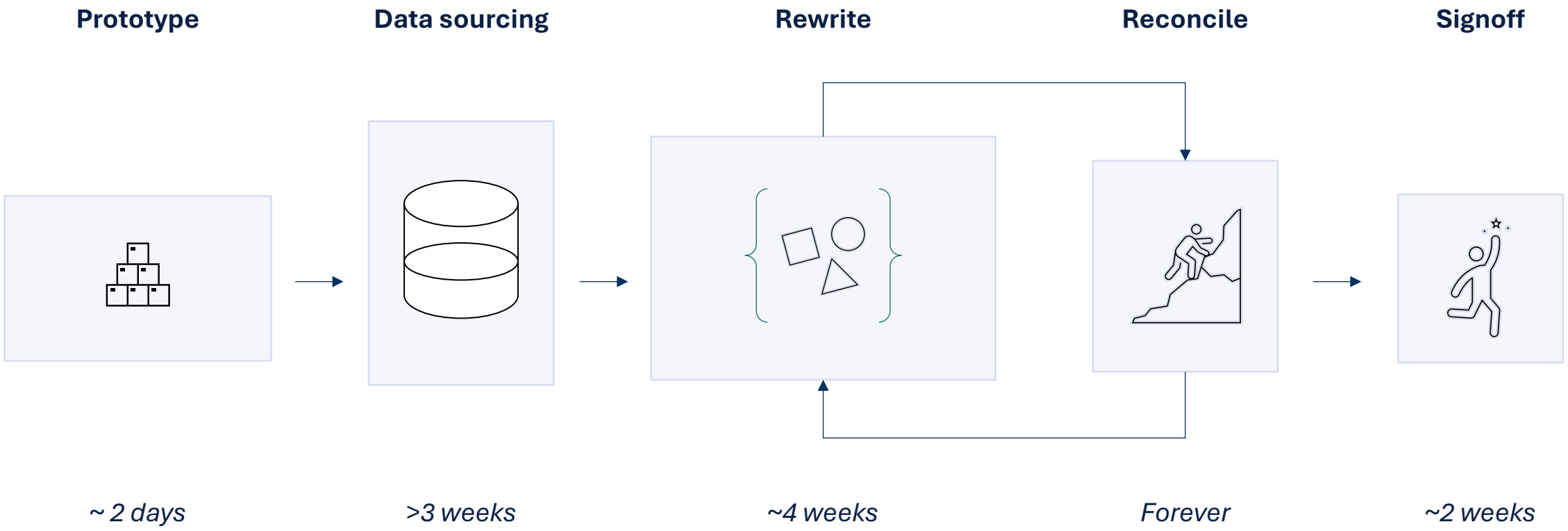
Architecture, Consistency and Scale

Kamlesh Shah | OSFF London 2026

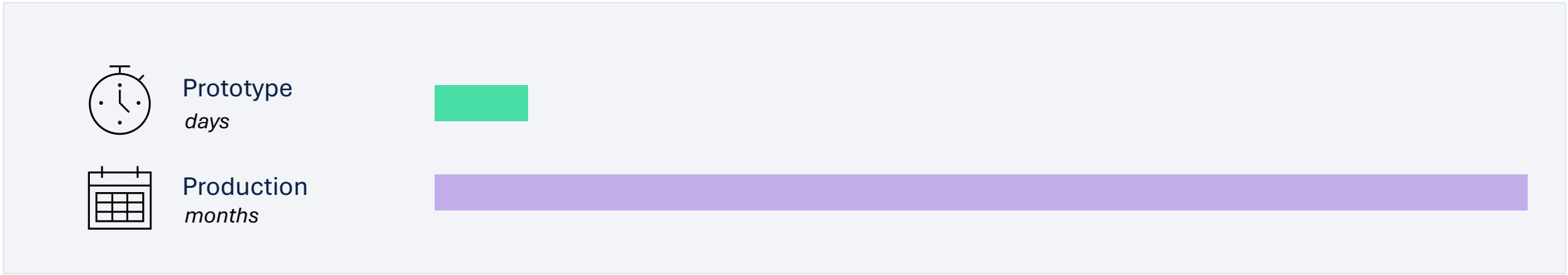
From an Idea to a ~~Product~~ Prototype



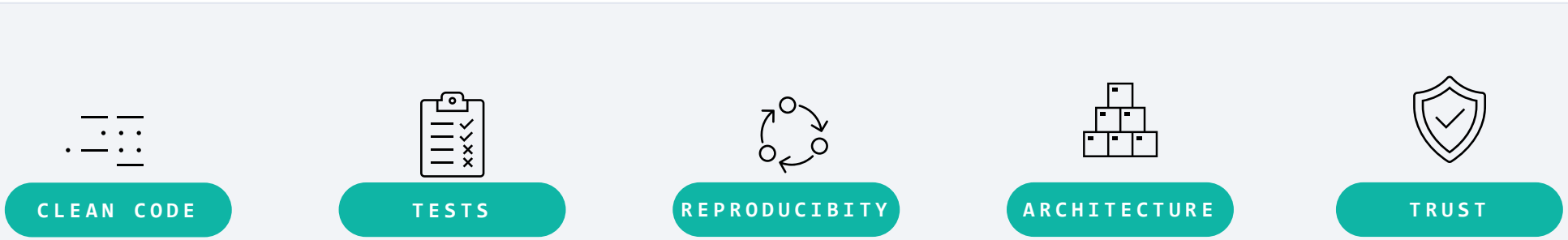
The real work starts



The damage



And what about..



Your calc is doing too much.

```python
def calculate_var(book, scenario="BASE", conf=0.99):
    conn = cx_Oracle.connect(os.environ["RISK_DSN"])
    px   = pd.read_sql(f"select * ..{book}' and dt='{date.today()}'", conn)
    cfg = yaml.safe_load(open("//nas/risk/scenarios.yml"))
    cov = pickle.load(open("cov_2021.pkl", "rb"))
    px["notional"] = px["notional"].astype(float)
    if   scenario == "STRESS": px["vol"] *= cfg["mult"]
    elif scenario == "CCAR08": px = apply_2008(px)
    pnl = monte_carlo(px, cov, draws=10_000)
    _CACHE[book] = pnl
    return np.percentile(pnl, (1-conf)*100) * (3 if book in LIMITS else 1)

def calculate_var_em(book, scenario="BASE", conf=0.99, ccy="USD"):
    conn = cx_Oracle.connect(os.environ["RISK_DSN"])
    ...
    fx = pd.read_sql(f"select rate from fx where ccy='{ccy}'", conn).iloc[0]
    _CACHE[book] = pnl
    return np.percentile(pnl, (1-conf)*100) * 3

def backtest(book):
    hist = pd.read_csv("marks_2015_2024.csv")
    pool = ThreadPoolExecutor(max_workers=8)
    for dt, px in hist.groupby("dt"):
          ...
    return _CACHE
```

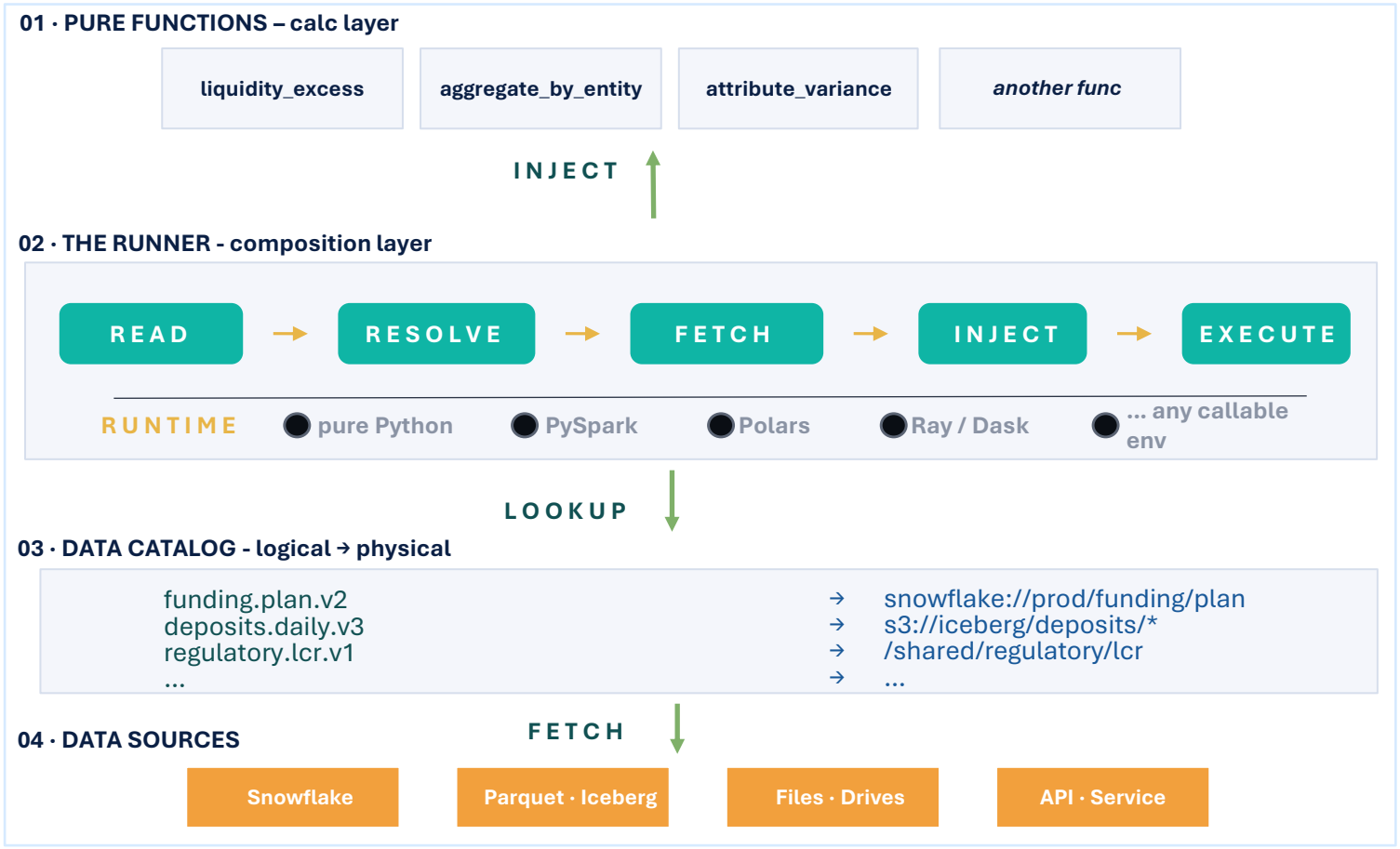
- Mixed responsibilities
- Duplicate VaR paths
- Hidden global state
- Unsafe data access
- Fragile concurrency model
- Hardcoded infrastructure coupling
- Weak resource lifecycle
- Poor reproducibility

α Alpha *a modular execution platform*

CALM MANIFEST

```
"nodes": [
 {"unique-id":"calc-layer",
  "node-type":"service"},
 {"unique-id":"runner",
  "node-type":"service"},
 {"unique-id":"data-catalog",
  "node-type":"service"},
 {"unique-id":"data-sources",
  "node-type":"database"} ],

"relationships": [
 {"unique-id":"runner-injects-calcs",
  "relationship-type":{"interacts":{
    "actor":"runner",
    "nodes":["calc-layer"]}}},
 {"unique-id":"runner-resolves-catalog",
  "relationship-type":{"interacts":{
    "actor":"runner",
    "nodes":["data-catalog"]}}},
 {"unique-id":"catalog-fetches-sources",
  "relationship-type":{"connects":{
    "source":{"node":"data-catalog"},
    "destination":{"node":"data-sources"}
```



To trust the outcome, you need to trust the architecture.
With CALM, you have a spec that you can version, validate and enforce.

Pure Calcs

THE CALCULATIONS ARE PURE FUNCTIONS
with Injected Data

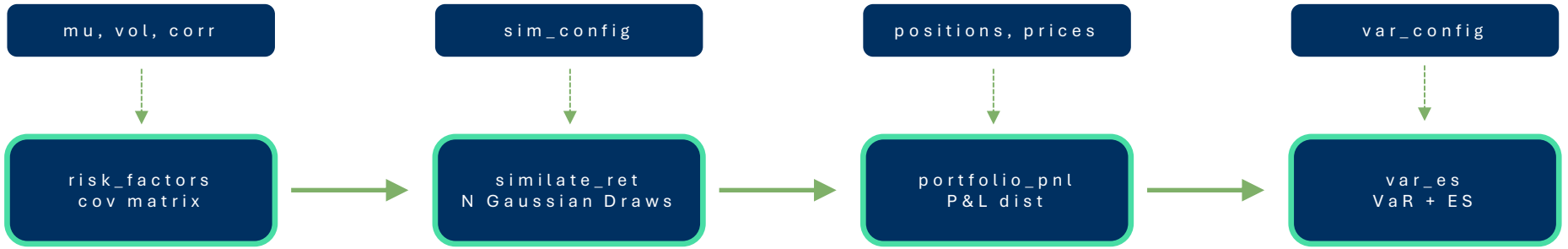
```
class MonteCarloVarCalc:
    def risk_factors(self, mu, vol, correlation):
        ...
    def simulate_returns(self, risk_factors, sim_config):
        rng = random.Random(int(sim_config["seed"]))
        ...                                          # correlated Gaussian draws (Cholesky)
    def portfolio_pnl(self, positions, prices, simulated_returns): ...
    def var_es(self, pnl_distribution, var_config):    # VaR + Expected Shortfall
        ...
```

TESTABLE
with literal python

```
def test_var_es_from_known_distribution():
    out = MonteCarloVarCalc().var_es([-1..-100], {"confidence": 0.99})
    assert out["var"] == pytest.approx(100.0)
```

Declared Inputs/Output & Runner

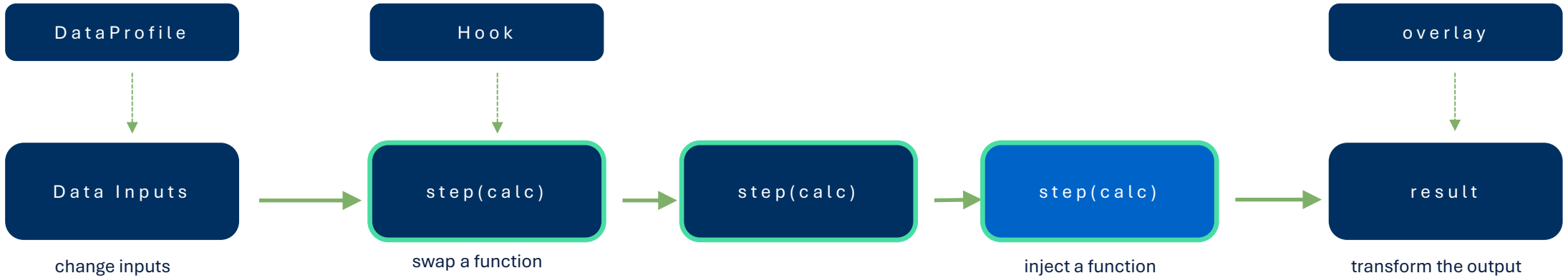
```
GenericRunner(model_id="monte_carlo_var_v1", steps=[
  StepSpec(step_id="risk_factors",     func=calc.risk_factors,
           args={"mu":DataSpec("mu"), "vol":DataSpec("vol"), ...}, result_key="risk_factors"),
  StepSpec(step_id="simulate_returns", func=calc.simulate_returns,
           args={"risk_factors":DataSpec("risk_factors"), ...}),
  StepSpec(step_id="portfolio_pnl", ...),
  StepSpec(step_id="var_es", ...) ], outputs=["var_es", "pnl_distribution"])
      ...
```




```
{ "unique-id": "flow-mc-var-primary", "name": "Monte Carlo VaR – Primary",
  "transitions": [
   { "relationship-unique-id": "runner-resolves-catalog",  "sequence-number": 1 },
   { "relationship-unique-id": "catalog-fetches-sources",  "sequence-number": 2 },
   { "relationship-unique-id": "runner-injects-calcs",     "sequence-number": 3 },
   { "relationship-unique-id": "runner-risk-factors",      "sequence-number": 4 },
   { "relationship-unique-id": "runner-simulate-returns",  "sequence-number": 5 },
   { "relationship-unique-id": "runner-portfolio-pnl",     "sequence-number": 6 },
   { "relationship-unique-id": "runner-var-es",            "sequence-number": 7 } ] }
```


- ✓ **Calc is a set of StepSpec declarations**
- ✓ **Runner builds DAG and executes them**
- ✓ **CALM Flow is the architecture contract**

Points of Control



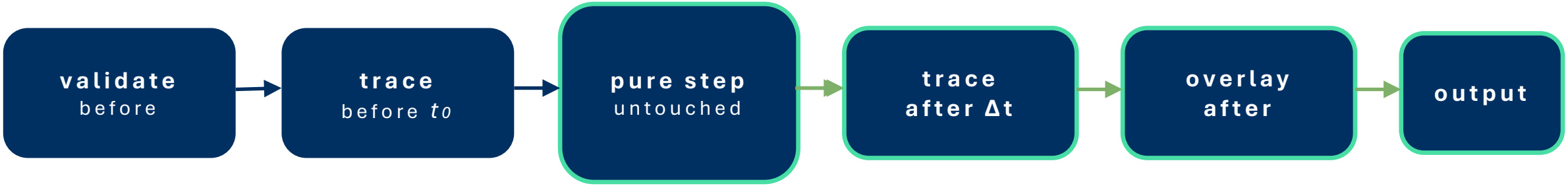
LEVERS	MECHANISM	CHANGES
Data-driven	DataProfile BASE/STRESS/WHAT-IF	The Inputs
Overlay	OverlayInterceptor	Post Calc Results
Hook	StepOverrides	Entire Step Function
New Logic hook	Interceptor	Business Logic

✓ **CALM Pattern enforces:**
- ✓ Every calc must have a TraceInterceptor
- ✓ Hooks can only reference declared relationships
- ✓ No structural changes without a manifest update

calm docify generates:
- ✓ Which controls sit on each calc
- ✓ What each lever impacts
- ✓ A full audit trail

Auditable & Reproducible

WRAP GOVERNANCE AS A CHAIN OF INTERCEPTORS



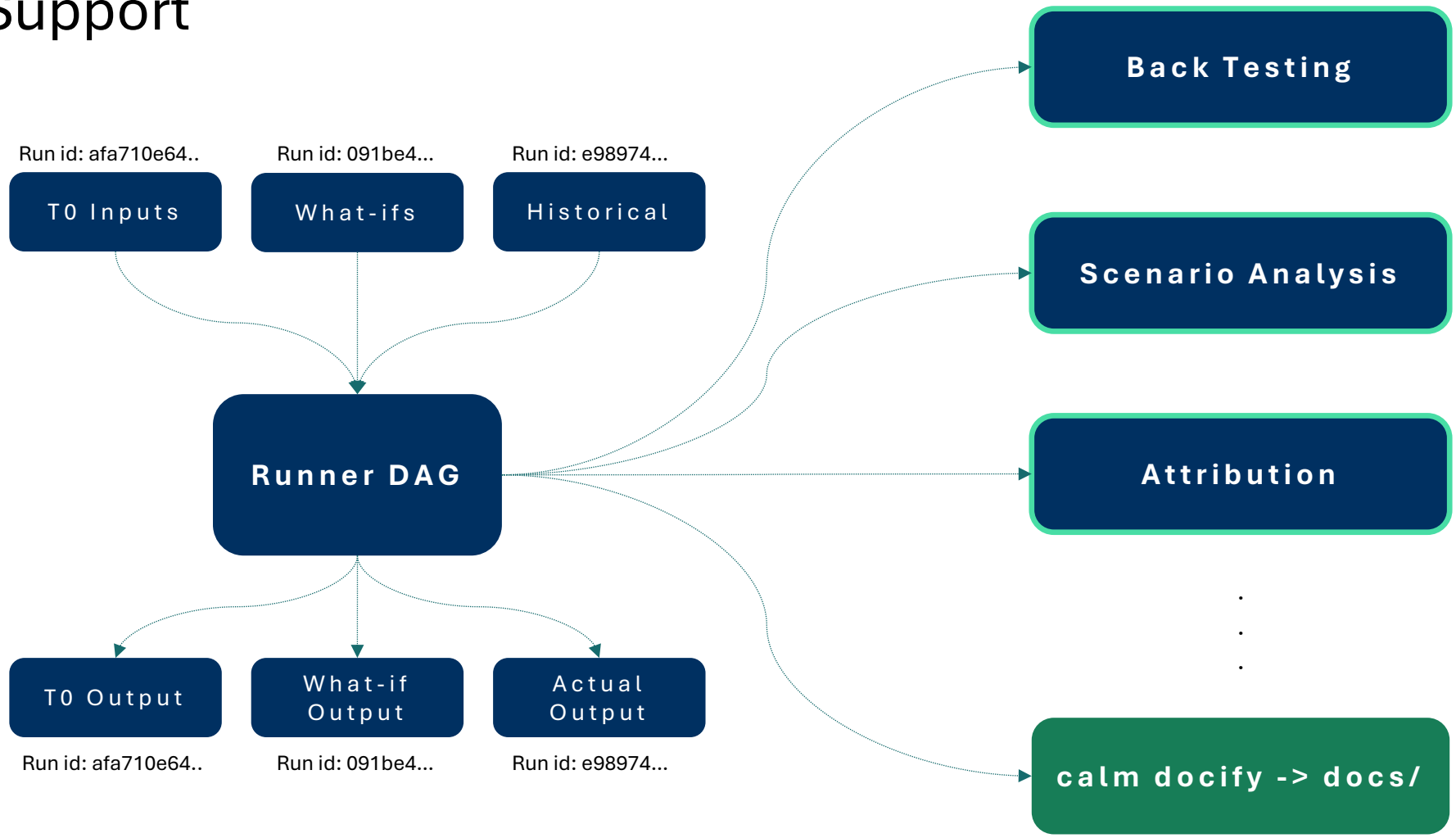
*run_id = SHA-256(profile · date · workspace) → BASE id => **afa71091be3d8e64**. Tag every moving part and data → re-run reproduces exact same results*

TraceIntercetop (Reusable construct) → logs after before and after every function call.

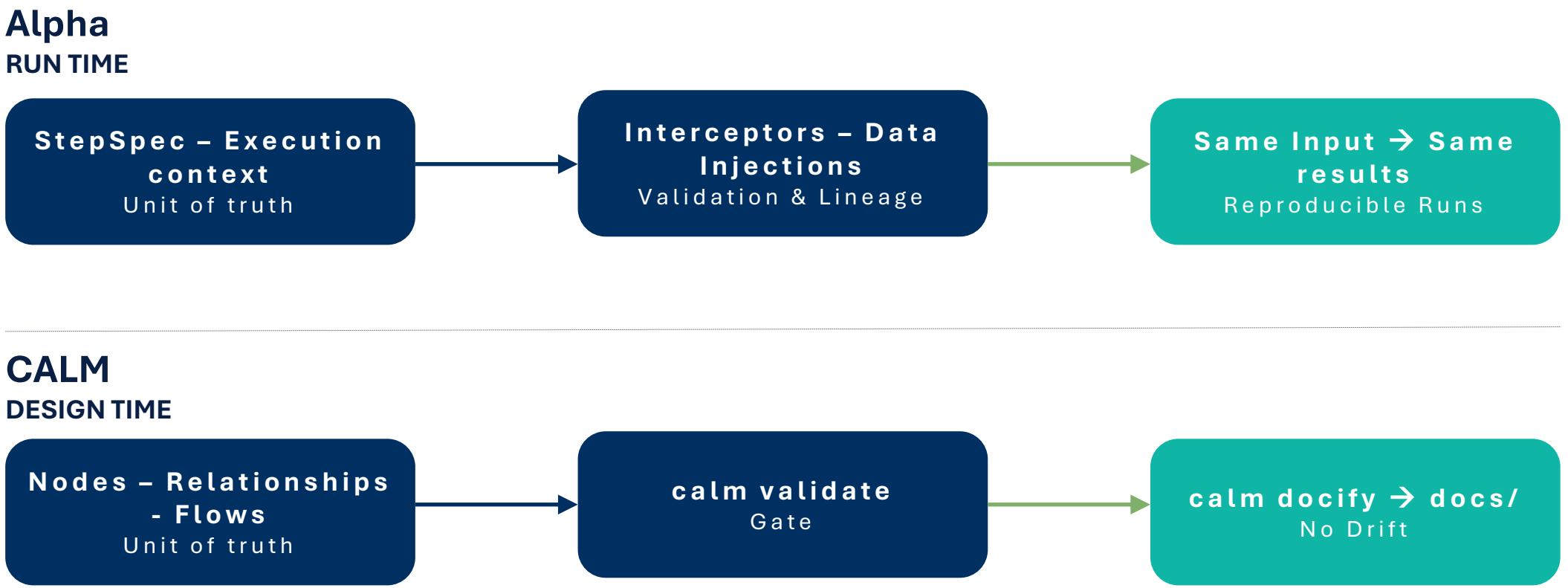
Run ID: **afa71091be3d8e64** Data Profile: base

Step ID	Function	Event	Duration (ms)
risk_factors	MonteCarloVarCalc.risk_factors	after	0.00
simulate_returns	MonteCarloVarCalc.simulate_returns	after	161.95
portfolio_pnl	MonteCarloVarCalc.portfolio_pnl	after	27.05
var_es	MonteCarloVarCalc.var_es	after	7.01

Pillars of Support



Deterministic



Alpha enforces determinism at runtime. CALM eliminates architectural drift.

Thank you

Re-engineer the calc lifecycle around a shared platform

→ the rewrite vanishes

→ discipline scales

→ architecture is the contract