Fachhochschule Graubünden
University of Applied Sciences

AISE502: AI in Software Engineering II

Lecture 6: Pipelines, Serverless, the View Across – and Your Class (C10)

Script: Part II, Sections PF / SL + closing; Part III, Section C10

Agenda

1. **PF** – Pipes-and-filters: the throughput pattern
2. **SL** – Serverless: pay per execution, own no capacity
3. Stepping back: the quantum, partitioning beats distribution
4. The consolidated capability table – all seven, side by side
5. Outlook: agent orchestration as a composition pattern
6. Part III opens: **C10 in depth** – the class of your project
7. The C1/C2 mirror pair
8. This week's exercise: **the match**

PF – Pipes-and-filters / batch pipeline

The nightly risk run must process millions of rows, reproducibly, by 06:00: what structure is born for exactly that?

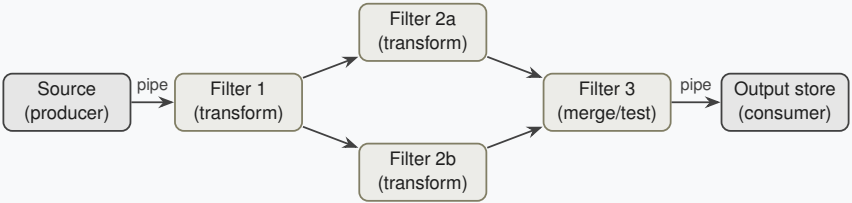
This one – the **oldest pattern in the catalogue**, and the one most precisely matched to its workload shape.

Definition: Pipes-and-filters / batch pipeline (PF)

A macro-structure of independent, composable transformation steps (*filters*) connected by unidirectional data conduits (*pipes*) into a chain or directed acyclic graph (DAG). Each filter is self-contained – ideally **stateless and idempotent** – and knows only its own data contracts, never its neighbours; the composition logic is explicit and *external* to the filters.

Historical root: the Unix pipe. Modern incarnations dominate the data world: ETL/ELT, DAG orchestration (Airflow), data-parallel engines (Spark), HPC job chains (Slurm), streaming pipelines (Flink – the bridge to EDA), and **ML and retrieval pipelines** – the current incarnation.

PF – topology and the problem it solves



every pipe is a versioned data contract; every filter is stateless, idempotent, and independently testable against golden datasets

The problem is the oldest workload in commercial computing: when the business day closes, the day's records must be collected, validated, transformed, aggregated – reliably, repeatably, *before the next day begins*. Payroll, end-of-day processing, warehouse loads, overnight risk runs: a finite body of data flows through fixed transformations, and nobody waits interactively. What matters: the run finishes **inside its window**, and the same inputs **provably produce the same outputs**.

PF – capability profile (column PF)

Dimension	Rating	Structural reason
D1 Read scalability	○	the pipeline does not serve reads; precomputation *delegates* to the output store
D2 Write scal. & elasticity	+	data-parallel frameworks are the standard operating mode
D3 Latency & predictability	−−	answers arrive in makespans, not milliseconds – by design
D4 Consistency & integrity	+	immutable inputs + idempotent stages: *reproducibility*, "as of last run"
D5 Availability & isolation	−	a failed stage stalls the run; recovery is re-execution
D6 Security & auditability	+	versioned inputs, deterministic reruns: audit trail on demand
D7 Evolvability	+	filters individually replaceable behind explicit data contracts
D8 Simplicity & TTM	++	explicit composition over self-contained filters – shipping this week
D9 Testability & deployability	+	golden datasets per filter; bit-level assertions
D10 Operating cost	++	compute in schedulable bursts, near-zero platform staff
D11 Team scaling	○	DAG/filter ownership parallelises data teams moderately
D12 AI integrability	++	ingestion, training, evals *are* pipes-and-filters; non-determinism localised
Native shape $S(p)$	scheduled batch	

PF – the cells with a story

- **D3 $= --$ is not a defect but the definition**: the pattern optimises *makespan* (first filter starts $\rightarrow$ last finishes) and batch-window adherence, and delegates interactive serving to the stores it fills. The streaming incarnation escapes the cell.
- **D4 $= +$ is the subtlest cell in the table**: not ACID – immutable inputs plus deterministic, *idempotent* stages give **reproducibility**, a *third* consistency semantics beside ACID and eventual ("as of last run"). For scientific and regulatory workloads, the one that matters.
- **The most deterministically testable pattern in the catalogue** – as long as no AI filter sits inside.

Example: a Monte-Carlo risk run – reproducibility by construction

Immutable market-data snapshots and versioned parameters enter; embarrassingly parallel simulation stages fan out; deterministic aggregation produces versioned risk figures. Fixed seeds $+$ versioned inputs make the run **bit-level reproducible** – not a nicety but a regulatory duty for risk models: "the same inputs produce the same books, provably, on demand."

PF – engineering, build it and study it

Engineering: the pipeline versions *three* things – code, data, schemas; **backfills** (re-running history through changed logic) are their own deployment class with their own runbook. Test with **golden datasets** per filter. The on-call page: “the 02:00 run missed its window” – standing measures: makespan trend, window adherence. Coupling risk hides in the *pipes* (implicit schemas).

Example: Build it and study it – pipes-and-filters

Build. Apache Airflow (DAG-as-code; read its shipped example DAGs first). dbt Core: each filter as one SQL model. Dagster as an active alternative.
Study. `dbt-labs/jaffle_shop_duckdb` (Apache-2.0): every `.sql` model under `models/` is a filter, `ref()` wires the pipes, `dbt build` materialises the DAG. Runs fully locally on DuckDB: *very easy*. (Status check again: the classic `jaffle_shop` was archived in 2024; its successor ships *no licence file* – use the DuckDB variant.)

Anti-patterns: *stateful filters with side effects* (alarm: non-zero **rerun-diff rate**); *pipeline sprawl* (DAGs without owners or version control); *implicit schema coupling* (alarm: downstream breakage per upstream schema change).

PF – AI lens and key concept

AI Lens: Pipelines are where AI work naturally lives

D12 $= ++$. The AI-adjacent workloads are pipes-and-filters *by construction*: retrieval ingestion (documents $\rightarrow$ chunking $\rightarrow$ embedding $\rightarrow$ index), model training and batch inference, and **the eval harness itself** – a versioned pipeline from golden set to statistical verdict. One precise change when an AI filter enters a deterministic chain: *that stage's* test regime switches from golden-dataset equality to **statistical acceptance thresholds** (pass rates, score distributions) – the rest keeps its deterministic tests. **Few structures contain AI more cheaply.**

Key Concept

PF is the most deterministically testable pattern in the catalogue and the natural home of batch, data, and ML workloads. Its $--$ latency cell is its *definition*: makespan and reproducibility, with interactive serving delegated to the stores it fills.

SL – Serverless / Function-as-a-Service

Your load is zero at night and spikes at noon: why pay for idle servers at 03:00?

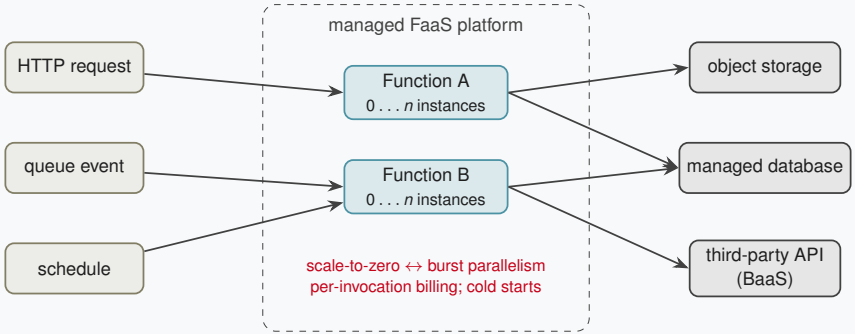
The serverless answer is radical – **pay per execution, own no capacity** – and the whole profile is the fine print of that offer.

Definition: Serverless / FaaS (SL)

A macro-structure of **event-triggered, short-lived, stateless functions** on a managed platform that provides provisioning, elastic scaling *from zero* to massive parallelism, and per-execution billing. State lives outside the functions, in managed backing services; **the function is simultaneously the unit of code, of deployment, of scaling, and of cost**.

The scientific reference point is the Berkeley view: it predicted the dominance of simplified cloud programming *while cataloguing the structural limits with unusual precision* – cold starts, enforced statelessness, communication through storage, vendor lock-in. FaaS (your code, event-triggered) vs. BaaS (API-consumed third-party services).

SL – topology



Functions are stateless: all state lives outside, and **inter-function communication runs through storage** – the documented cost trap. The problem it solves: **idle capacity** – much real work is spiky or rare (thumbnails, webhooks, reports), and provisioned machines force a bad choice between sizing for the peak and failing at it.

SL – capability profile (no star anchor: Berkeley/CNCF-derived)

Dimension	Rating	Structural reason
D1 Read scalability	$+$	wide parallelism, tempered by concurrency limits and the DB-connection bottleneck
D2 Write scal. & elasticity	$++$	scale-to-zero to mass parallelism, no capacity planning
D3 Latency & predictability	$-$	cold starts make tail latency structurally unpredictable
D4 Consistency & integrity	$-$	stateless functions push all state through external storage
D5 Availability & isolation	$+$	platform-managed redundancy; small blast radius per function
D6 Security & auditability	$\circ$	platform identity per function vs. a fragmented audit trail
D7 Evolvability	$\circ$	fine-grained deployability vs. vendor lock-in and sprawl
D8 Simplicity & TTM	$\circ$	no server management vs. a large configuration surface
D9 Testability & deployability	$\circ$	trivial unit tests vs. cloud wiring no laptop reproduces
D10 Operating cost	$++/--$	**split by load shape**: zero idle cost vs. billing $+$ storage round trips
D11 Team scaling	$+$	the smallest teams in the catalogue ship to production
D12 AI integrability	$\circ$	fine event glue; platform timeouts collide with minutes-long runs
Native shape $S(p)$	event-triggered, short-lived	

SL – the split cell, observed in production

- **D10** $= ++ / --$ **is the most workload-sensitive cell in the table**: $++$ for spiky load (zero idle cost); $--$ for sustained, data-intensive load. *It does not average to* $\circ$ *– averaging would erase precisely the information an architect needs.*
- **D3** $= -$ **has a mitigation with a sting**: provisioned concurrency removes cold starts – at the price of exactly the idle cost the pattern exists to avoid.

Example: Amazon Prime Video – a measured cost inversion

The video-monitoring service: Step Functions coordinating Lambdas, S3 buffering frames between stages. A hard scaling limit at $\sim$ 5 % of expected load, and consolidation into one ECS process cut infrastructure cost by $>$ 90 %. **The correct reading**: *one* service with a data-intensive, tightly coupled flow – PF pushed across expensive distributed boundaries – not a verdict on serverless. *(Widely reported as "Amazon abandons microservices" – which measurement, taken before the first release, would have predicted the inversion?)*

SL – engineering, build it and study it

Engineering: infrastructure-as-code becomes *part of the test subject*; integration tests run against emulators or ephemeral environments. Above all: **cost monitoring becomes an engineering discipline (FinOps)** – cost per request belongs on the same dashboards as latency.

Example: Build it and study it – serverless

Build. Knative (Go, Apache-2.0): clean, self-hostable FaaS on Kubernetes, runs on a local kind cluster. Moto (Python, Apache-2.0) mocks AWS APIs in local tests. Deliberately *off* this list: LocalStack (archived into a closed model, 2026) and Serverless Framework v4 (proprietary) – **the licence check, twice over**.
Study. The Knative Bookstore sample: independent Knative Services wired by Brokers and Triggers in YAML – the most involved setup of the seven boxes, but self-hostable end to end.

Anti-patterns: *Lambda pinball* (alarm: function hops per request); *cost inversion under load growth* (alarm: cost-per-request trend vs. load trend, break-even utilisation as a standing fitness function); *cold-start denial* (alarm: cold-start rate on p99 routes).

SL – AI lens and key concept

AI Lens: Serverless and AI: excellent glue – conditional runtime

D12 = ○, and the split mirrors the D10 cell. Event glue around *batch* AI APIs fits beautifully: a function that submits, polls, and stores an asynchronous AI job is serverless at its best. But **platform timeout ceilings collide with minutes-long LLM and solver runs** – a hard constraint, not a tuning issue – and per-call cost stacking across functions *plus* tokens is opaque without a single gateway measurement point. **A fine chauffeur for AI jobs, and a poor place for them to live.**

Key Concept

The serverless profile is dominated by one variable more than any other pattern's: **load shape**. Unmatched elasticity and scale-to-zero economics for spiky workloads – inverting into the catalogue's worst cost cell under sustained, data-intensive load. The split D10 rating is the table being honest where an average would lie.

Stepping back: the architecture quantum

Seven times the same movement: problem → topology → profile → engineering → runnable code. What no single section could deliver is the view *across* – and it starts with the unit you met in every topology figure: **the dashed boundary**.

Definition: Architecture quantum

An *independently deployable* unit that can be deployed, scaled, and can fail independently of the rest – the joint unit of deployment, scaling, and failure. L, MM, and PF form exactly **one** quantum; EDA one or more; MS **many**; SL many small, short-lived ones.

Key Concept

Maxim 3. One quantum → cheap, simple, consistent, rigid. Many quanta → expensive, complex, eventually consistent, elastic. *The quantum count explains most of the capability table.*

Partitioning beats distribution

The second axis is orthogonal: *how* the units are cut. **Technical partitioning** groups by technical role (layers, filter stages); **domain partitioning** by business capability (modules, services). Change requests arrive in the *domain's* vocabulary – so a feature cuts across every technical unit, but lands *inside one* domain unit. (Parnas's criterion, six decades on.)

You watched the axis operate **in isolation**:

- **L** $\rightarrow$ **MM**: quantum count constant, only the cut flips – D7/D9 rise from $-$ to $+$
- **MM** $\rightarrow$ **MS**: domain cut constant, quanta multiply – evolvability stays, the bill arrives on D8/D10

Key Concept

Maxim 4. The partitioning axis beats the distribution axis: *domain-oriented partitioning, not the number of deployment units, is the strongest single predictor of evolvability.*

The 2×2 logic: L, PF $=$ technical/single quantum $\cdot$ MM $=$ domain/single $\cdot$ MS $=$ domain/many $\cdot$ EDA, SL multiply quanta along technical seams $\cdot$ HX orthogonal to both.

The evidence base – and its honest gaps

The most systematic public rating: the star scheme of Richards & Ford – eleven characteristics, one to five stars, calibrated here via the fixed mapping ($5\star \to ++ \ldots 1\star \to --$). Its own headline result proves a matching problem exists: **no style dominates** – microservices lead the aggregate yet one star on cost and simplicity; layered is the exact mirror.

Important Note

Four caveats before comparative use: (i) star ratings are structured **expert judgement**, not measurements – transcribed from the first edition, to be reconciled against the second before print; (ii) **HX carries no star profile at all** – correctly, as a dependency-organisation pattern; its column is a flagged delta; (iii) **SL** likewise – its column derives from Berkeley + CNCF; (iv) **MM** is a rated style only since the second edition. Compensation: *triangulation* – expert ratings $\times$ documented production cases $\times$ defined response measures – plus the measurement contract, which converts every adopted claim into a testable one.

The consolidated capability table

Dimension	L	MM	HX	MS	EDA	PF	SL
D1 Read scalability	○	○	◇	++	++	○	+
D2 Write scalability & elasticity	−−	−	◇	++	++	+	++
D3 Latency & predictability	+	+	◇	−	+	−−	−
D4 Consistency & integrity	++	++	◇	−−	−−	+	−
D5 Availability & fault isolation	−	−	◇	++	++	−	+
D6 Security & auditability	+	+	+	○	○	+	○
D7 Evolvability & maintainability	−	+	++	++	++	+	○
D8 Simplicity & time-to-market	++	+	−	−−	−−	++	○
D9 Testability & deployability	−	+	++	+	−	+	○
D10 Operating cost efficiency	++	++	◇	−−	○	++	++/−−
D11 Team scaling (Conway)	−	○	◇	++	+	○	+
D12 AI integrability	○	+	++	○	++	++	○
Native shape $S(p)$	interact.	interact.	(host's)	interact.	stream	batch	event-trig.

The 21 table notes (every deviation, with evidence) are in the script.

Four reading rules – and the table's status

1. **HX is a delta pattern**: ◇ inherits the host; its own cells are what the cut adds
2. **Every deviation is footnoted** – an unexplained deviation would violate the theory's own rigour standard
3. **One cell is split**: SL's D10 genuinely inverts with load shape – a standing sensitivity point, not a ○
4. **The last row feeds the shape gate**: $S(p)$ vs. $S(a)$ in stage-1 knock-out screening

Key Concept

Ordinal reading only: rankings and exclusions, never weighted sums. Every cell is a **default hypothesis** – replaced by measurement once the system exists. Two cells documented to invert with context: SL cost (Prime Video), L read scalability (Stack Overflow). *The matrix predicts the default, not the exception.*

Reading the catalogue as a whole

- **No column dominates – so a matching problem exists.** A2 made visible in data; two variables – quantum count (Maxim 3) and partitioning axis (Maxim 4) – explain most of 84 cells.
- **The cost curves cross.** Single quanta: low fixed cost, superlinear maintenance growth unless governance holds. Many quanta: high fixed cost (platform staffing) or load-proportional cost. The five case studies are where fit and misfit were *measured in money*.
- **Conway is a decision filter, not a footnote.** Every column presupposes a team topology: L one team · MM 3–5 coordinated · MS stream-aligned + platform. Loosely coupled architectures *and teams* predict continuous delivery.

Discussion

Stack Overflow: $\sim$ 1.3bn page views/month from *one* quantum. Monzo: a licensed bank on $\sim$ 2,800. What exactly does this pair falsify about “microservices scale better than monoliths” – and what does it *not* falsify?

Outlook: agent orchestration – an emergent composition pattern

One candidate for an “eighth pattern”, filed correctly: agent orchestration is **not a new style** – it is a *composition pattern* for non-deterministic components that **reuses this catalogue’s topologies**:

Agent construct	Classical topology
prompt chaining	pipes-and-filters
routing	dispatch layer
parallelisation with voting	broker-style fan-out
orchestrator–workers	mediator EDA
evaluator–optimizer loop	feedback control loop
multi-agent systems	broker fan-out of autonomous quanta

Load-bearing distinction: **workflows** (predefined code paths) vs. **agents** (the model steers its own process). The rigour case for restraint is quantified: a multi-agent research system beat a single agent by 90.2 % – at roughly **fifteen times** the token consumption, with token use alone explaining 80 % of the variance. The module’s default rule: *workflows before agents*; every escalation is an ADR with a measurement contract.

Part III opens: C10 – AI-native advisory platforms

The component your product is built around is non-deterministic, priced per call, and deprecated within months: what structure contains it?

The youngest class in the catalogue, **the profile of the course project**, and the reason this course exists in its present form. The architecture must *contain* the AI component: deterministic services for everything deterministic, LLM calls only where determinism cannot reach, every generated statement grounded in retrievable sources.

Three challenges define the class:

1. **Accountability for probabilistic output** – provenance per claim, a log per agent step: observability is *domain functionality*, with EU AI Act force behind it (potentially high-risk classification)
2. **A genuinely new cost model** – requests rare but heavy: cost *per request* (tokens, GPU), not per user; batch APIs $\sim$ 50 % cheaper; model cascades up to 98 % cost reduction; learned routers halve cost
3. **Churn at the core's edge** – models, prompts, frameworks turn over in months: the most extreme change rate in the catalogue, and the strongest argument for ports and adapters

C10 – the binding scenarios

- **S1 (grounded answer).** A client asks for a recommendation; **every factual claim carries a resolvable provenance reference**, failing answers are blocked and escalated – **detection rate** $\geq 99\,\%$ **at** $< 2\,$**s added latency**. (The D6 scenario.)
- **S2 (cost per request).** A session triggers a multi-step agent workflow; it completes **within a per-request token-cost budget (e.g. CHF 0.40 at p95) and a p95 latency budget (e.g. 20 s)** – both CI-gated fitness functions. (The D10 scenario.)
- **S3 (model migration).** The provider deprecates the production model; the platform migrates with **eval-harness pass rate** $\geq$ **threshold on the golden set before rollout, rollback available**. (The D7/D9 scenario.)

Knock-out reading of K**:** an architecture in which agent steps are not loggable, tool privileges not boundable, or provenance not reconstructable is **excluded before scoring** – C1's ACID veto logic, transposed to accountability. Prompt injection cannot be fully solved in the model $\rightarrow$ system-level defence in depth is *constitutive*.

C10 – requirements profile (the only column with an H on D12)

Dimension	Weight	Why
D1 Read scalability	M	retrieval reads; user concurrency modest
D2 Write scal. & elasticity	L	requests rare; batch pipelines scheduled, not elastic
D3 Latency	M	users accept seconds-to-minutes for advisory answers
D4 Consistency & integrity	M	hybrid: knowledge index eventual, audit path ACID
D5 Availability & isolation	M	degraded answers beat no answers
D6 Security & auditability	**H**	provenance per claim, a log per agent step – AI Act force (S1)
D7 Evolvability	**H**	models, prompts, frameworks turn over in months (S3)
D8 Simplicity & TTM	M	start simplest – but never simpler than the audit path
D9 Testability & deployability	**H**	**evals are the operative meaning of testability** (S3)
D10 Operating cost	**H**	cost per *request* – a run-cost class no classical profile contains (S2)
D11 Team scaling	L	small product teams; the platform premium is unaffordable
D12 AI integrability	**H**	definitional: the class exists to contain the probabilistic component
Shape $S(a)$	hybrid: interactive + batch/async	
Constraints $K(a)$	EU AI Act 2024/1689 (logging, oversight; potentially high-risk); GDPR	

C10 – what real systems chose, and why

Three documented building blocks define the reference shape:

- **Retrieval-augmented generation**: ingestion → vector index → retrieval → context → generation with citations – structurally a **PF pipeline plus a serving layer**
- **Agent orchestration**: *workflows before agents* – the restraint case is quantified (90.2 % better at 15× the tokens)
- **Compound AI systems**: results come from systems of retrievers, models, tools, verifiers – *the system architecture becomes the differentiator*

The capability tables explain the host choice:

- **MM** hosts the deterministic majority in one ACID quantum *and* gives the AI subsystem a hard, CI-verifiable boundary
- **HX** answers the two hardest Highs: the LLM as a swappable adapter behind a port (D7); the port is where the **eval harness and cost gateway dock** (D9, D10)
- where A2 bites: synchronous chains multiply LLM latency – what keeps **MS at** ○; the monolith's weak cells are mitigated by **asynchronous edges**

The C1/C2 mirror pair – weights, not dimensions, define a class

C1 core banking

- a lost or double-posted booking *creates or destroys money*: D4/D5/D6/D7/D9 High
- verdict: **MM+HX core**, EDA edges, PF batch runs; MS only at the Monzo condition (D11 forced High)
- LMAX vs. Monzo: *same profile, opposite structures* – $R(a)$ defines the feasible set, $K(a)$ decides within it

C2 social/content platforms

- $\sim$ 50:1 read/write ratio; a *stale* feed is invisible, an *unavailable* feed is the defect: D1/D3/D5/D7/D9/D11 High, **D4 Low**
- verdict: **EDA+MS hybrid** at organisational scale; MM secondary until that scale is *measured* (Instagram: a Django monolith at 100 deployments/day)

Key Concept

C1 and C2 are **mirror images** across the consistency/availability trade: same twelve dimensions, inverted weights on D1 and D4. *Weights, not dimensions, define a class.*

C10 – your project as an inheritance diagram

Project Link: Portfolio Intelligence Platform

The Portfolio Intelligence Platform *is* a C10 instance. **Inherited:** ingestion and eval pipelines are C6 (reproducible batch); analytics are C7 (refresh contracts, lineage); the deterministic services are C3 (ACID, boring on purpose). **New:** the cost model – *expensive per request*: a token-cost budget, enforced in CI. **Also new:** D12 = H, and D9 = H in its eval reading – a versioned golden set gates every prompt change and model migration. *Spend your design budget on the new elements – the inherited disciplines are solved problems.*

Key Concept

C10 **stress-tests** the method rather than overthrowing it: one new High dimension, one new cost semantics, one new test-artefact class – otherwise *inherited* (A6 made concrete). In one sentence: a *hexagonal modular monolith* – AI adapters at ports, an LLM gateway as the single measurement point – plus PF for ingestion and evals, EDA for the job and audit spine.

This week's exercise: the match

Project Link: Portfolio Intelligence Platform

Run the **full three-stage procedure** for your platform, against your A1 requirements profile:

1. **Knock-out**: constraints and the workload-shape gate – which candidates cannot carry the core?
2. **Veto**: hold every High weight against the capability columns – which vetoes fire, which have *documented* mitigations?
3. **Ordinal reading**: rank the survivors on the High set; run the sensitivity check

Take the decision – and begin the ADR (MADR: drivers, options, consequences, confirmation).

Next week's lecture shows the same procedure formally – you will recognise every step.

Summary

1. **PF**: makespan and reproducibility – a *third* consistency semantics (“as of last run”); the most deterministically testable pattern; the natural home of ingestion, training, and evals
2. **SL**: the profile is dominated by load shape; the split D10 cell is the table being honest – Prime Video is note 18 observed in production, with a price tag
3. **Maxims 3 and 4**: quantum count and partitioning axis explain most of the 84 cells
4. The **consolidated table** supports ordinal reading only – every cell a default hypothesis, two cells documented to invert with context
5. **Agent orchestration** reuses this catalogue’s topologies – workflows before agents (15$\times$ token finding)
6. **C10**: Highs on D6/D7/D9/D10/D12 – accountability, churn, evals, cost per request, containment; inherits C6/C7/C3, adds the eval harness and the token budget
7. **C1**/**C2**: mirror images across the consistency/availability axis – weights, not dimensions, define a class

Next week

Lecture 7 – Part IV: the match, formally

- three cases, three stages
- the procedure in general; reading the 7×10 matrix
- the measurement contract, introduced

Reading

- this week: Part II close; Part III, C10
- ahead: Part IV, sections 1–3

Exercise / deliverable

- the match; decision; ADR begun
- **A2 + design gate: next week**

Thank you!

Dr. Florian Herzog

Fachhochschule Graubünden, Chur

AISE502 – AI in Software Engineering II