Transactional Cloud Applications: Status Quo, Challenges, and Opportunities

UNIVERSITY OF COPENHAGEN

TU Delft

Who are we?

Rodrigo Laigner
(University of Copenhagen)

George Christodoulou
(TU Delft)

Kyriakos Psarakis
(TU Delft)

Asterios Katsifodimos
(TU Delft)

Yongluan Zhou
(University of Copenhagen)

Slides, and pointers:
https://delftdata.github.io/tutorial-sigmod25/

UNIVERSITY OF COPENHAGEN

TU Delft

Disclaimer: not your typical tutorial

This is **not** your typical tutorial

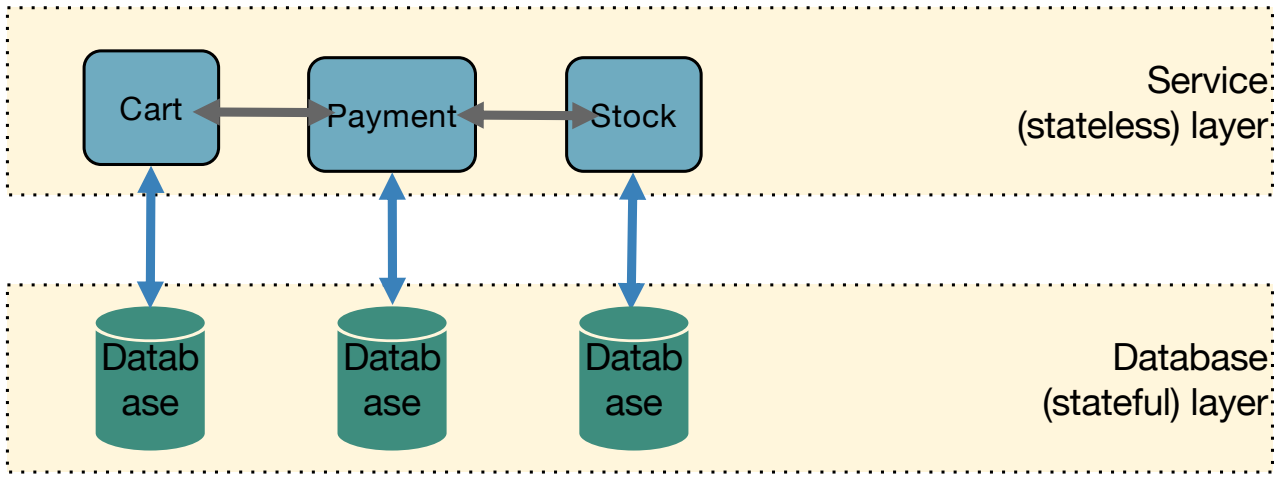
The list of presented systems is **not** exhaustive

This is **not** a survey (although it's a good start)

Presenters may be **opinionated** (at times)

Tutorial Scope

Cloud services (canonical example)



Check & update ***stock***, verify ***payment***, then checkout the shopping ***cart***.

Use Cases for Transactional Cloud Applications

Target Use-cases: *Low-latency Cloud applications requiring transactional consistency, containing complex business logic.*

Booking/reservation, systems, trading

Ad serving & bookkeeping

Fraud detection & payments

Inventory management

Target Industries

Banking, e-commerce, trading platforms, retail, etc.

ACID in the world of services

Atomicity

All three services execute, despite system/user errors

Consistency

FK constraints: shopping cart contains only products that exist in stock

Isolation

No stock updates visible without having payment cleared
No payment without stock updates reflected in stock service

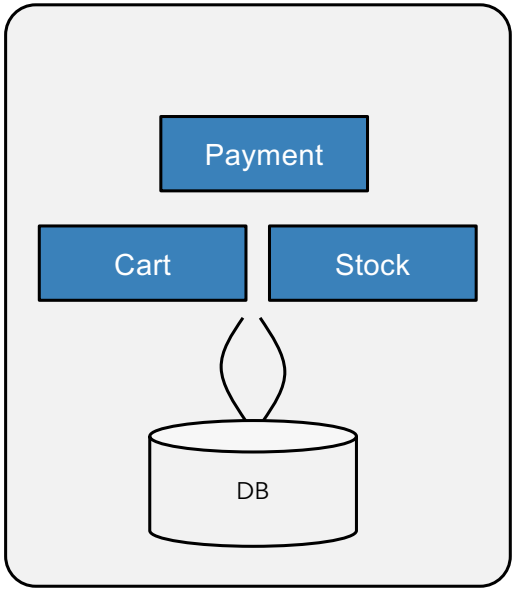
Durability

Data is safe, possibly replicated

How people develop microservices nowadays

Service Architectures Primer: Monolith

Monolith



YES,

Programmers only deal with app logic

Database ensures ACID

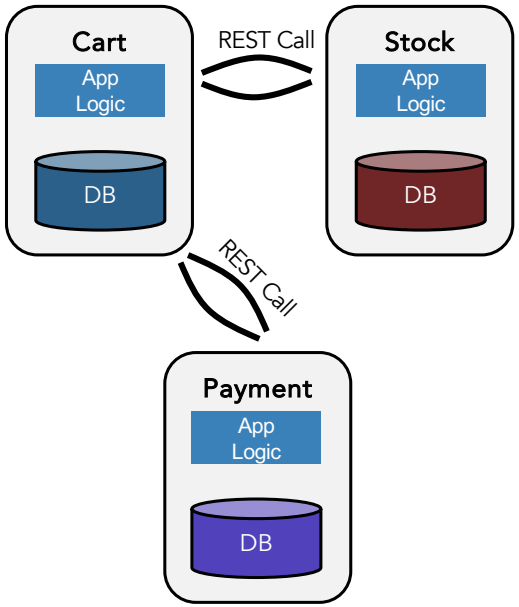
BUT

Cannot migrate easily to the Cloud

May not scale on commodity hardware

Service Architectures Primer: µService

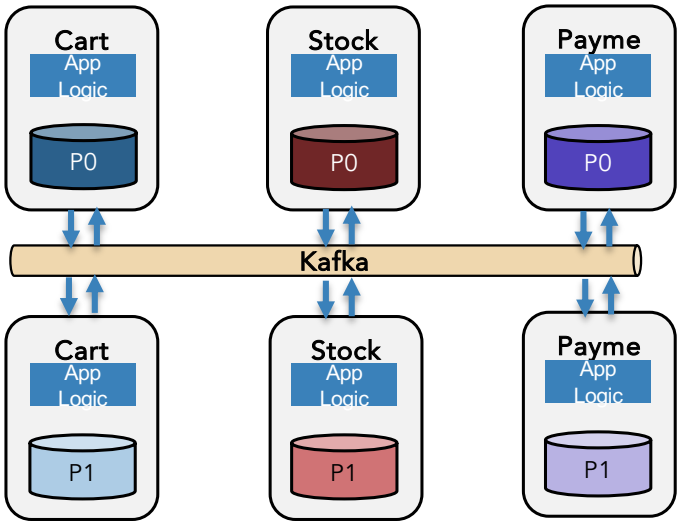
"Textbook" Microservice



YES,	BUT
Scales better	Requires orchestration
Code modularity	Retries/idempotence hard
Independent deployment/scaling	Consistency/Transactions in app code (SAGAs, 2 Phase Commit/Open XA)

Service Architectures Primer: event-driven µService

Partitioned, event-driven architecture



YES,	BUT
High Performance	Inherits µService issues
Fault-tolerant	Few people can program and keep it running
Replayable & Debuggable	Ad-hoc transactions + event-driven = trouble

Different ‘shades’ of consistency guarantees

Cloud applications largely avoid transactions through databases

- Exceptions apply, e.g., DBOS

Transactions are implemented in user code.

- Ad-hoc (with tons of hacks)
- Two-phase commit
- SAGAs

Transactions

Two-Phase Commit in Services

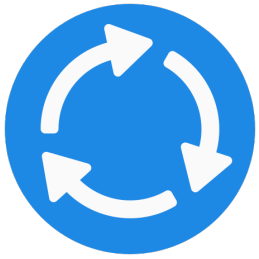
Coordination for committing or rolling back a transaction

Two phases: Prepare phase and Commit phase

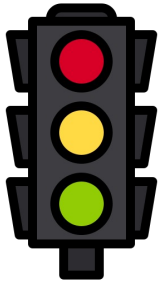
Serializable guarantees

Reduced throughput

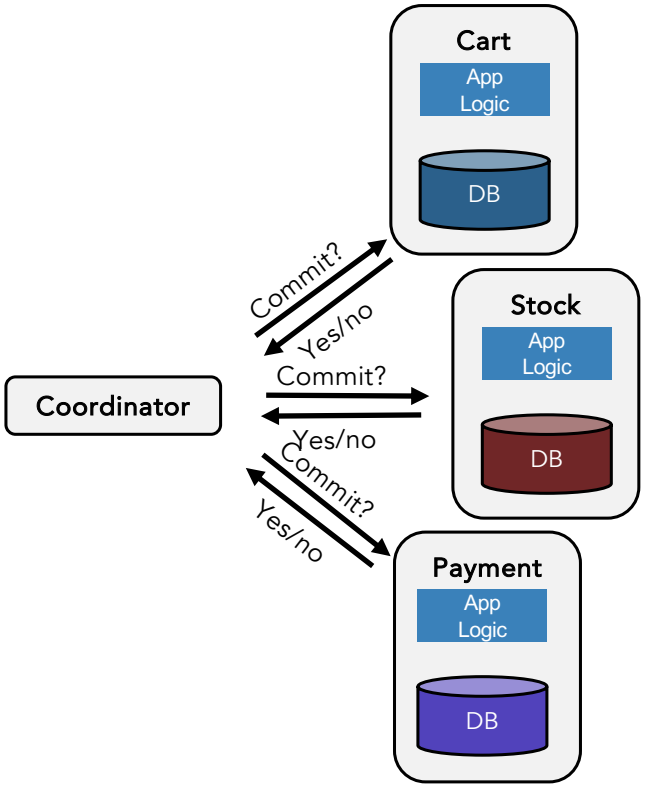
Single point of failure

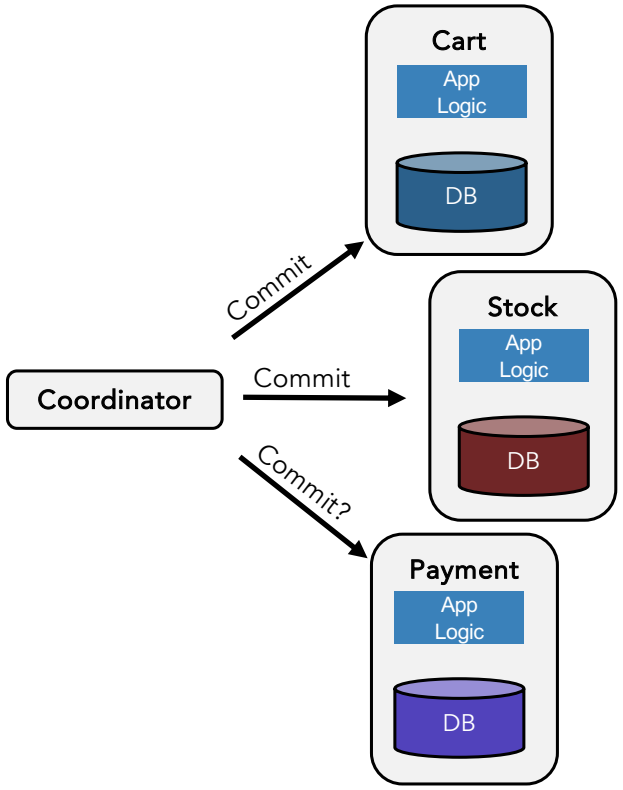


vs



Two-Phase Commit example





SAGAs in Services

Talk about the sagas, compensations, etc.

The result is eventually consistent, under many assumptions.

The world implements SAGAs without knowing it.

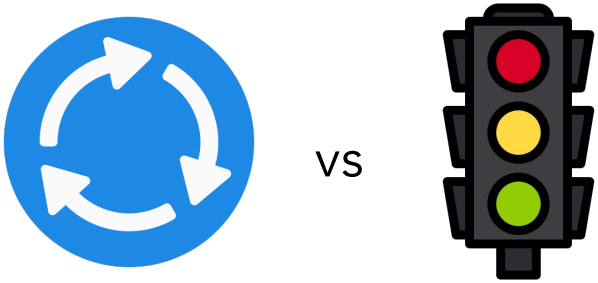
SAGAs in Services [1]

Distributed transaction as multiple single local transactions

Compensatory transactions in case of failure

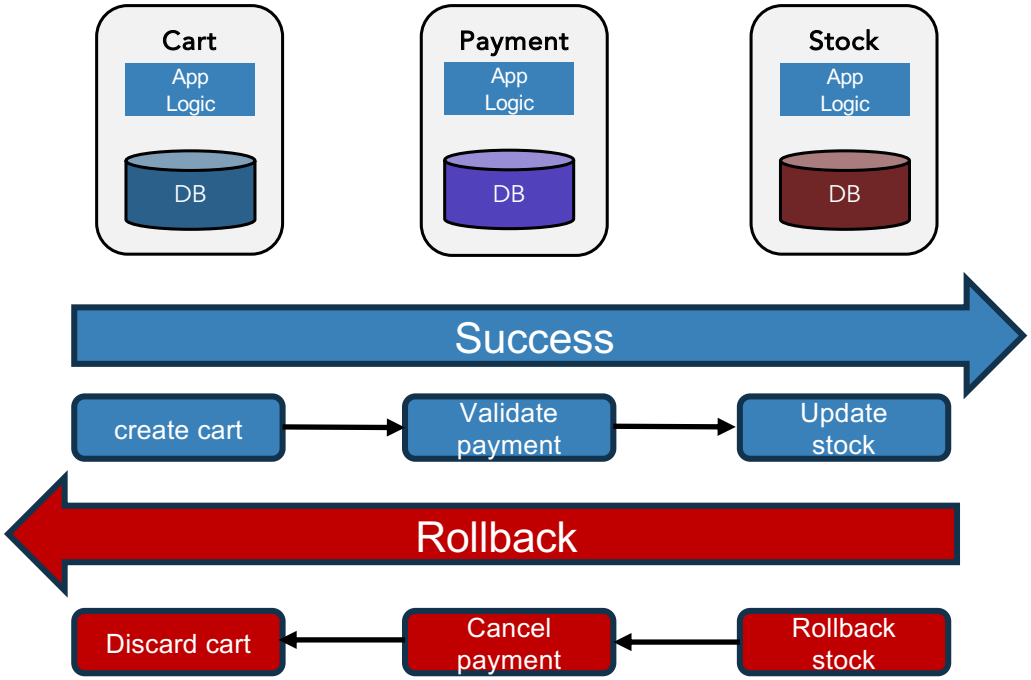
Durable and distributed logs of all messages (usually Kafka topic)

Eventual consistency



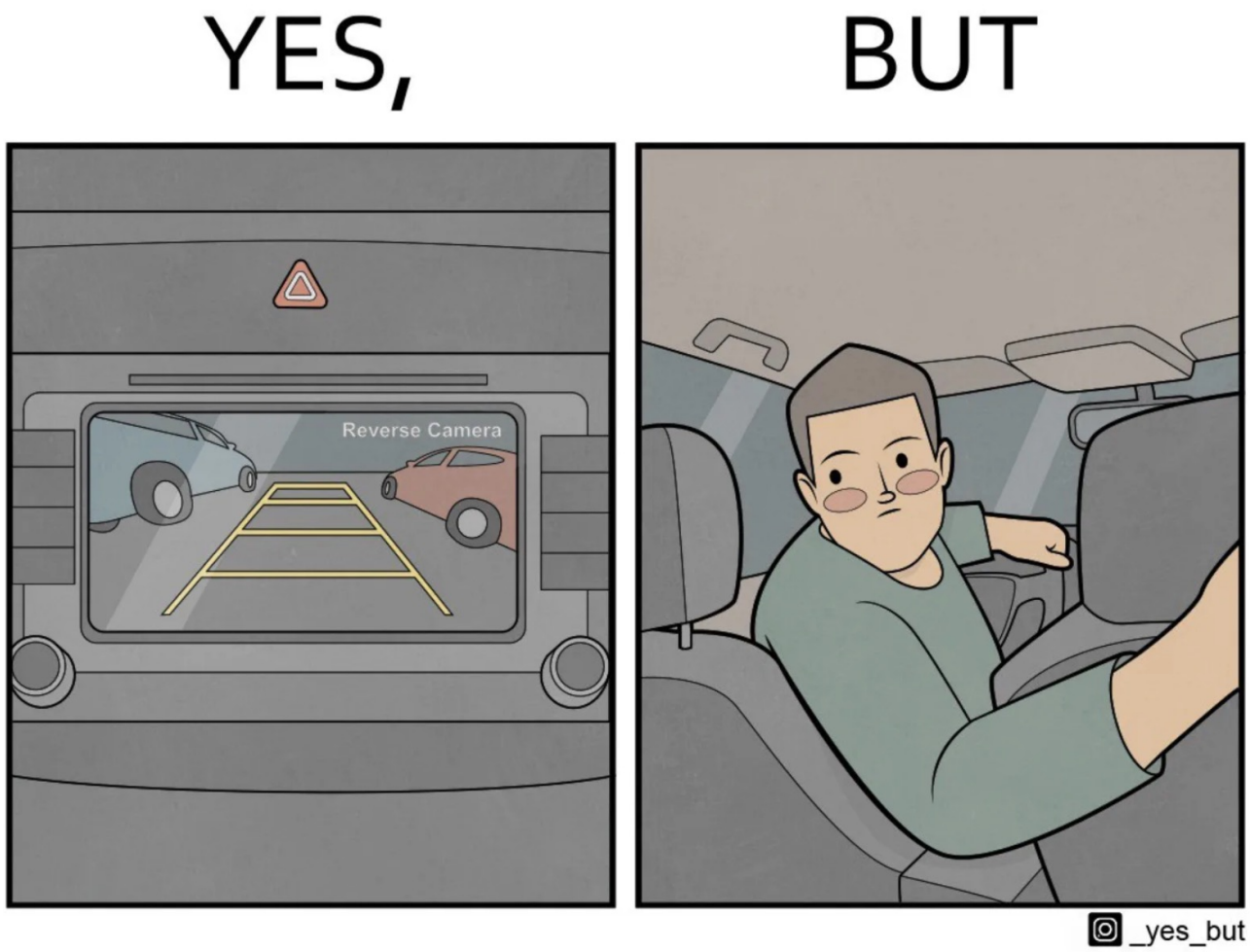
[1] H. Garcia-Molina, K. Salem, **"Sagas"**, [SIGMOD '87]

SAGAs in Services [1]



[1] H. Garcia-Molina, K. Salem, **"Sagas"**, [SIGMOD '87]

Yes, but



Ad-hoc Transactions; allover in Microservices

Ad Hoc Transactions: What They Are and Why We Should Care

Chuzhe Tang[1,2], Zhaoguo Wang[1,2], Xiaodong Zhang[1,2], Qianmian Yu[1,2]
Binyu Zang[1,2], Haibing Guan[3], Haibo Chen[1,2]
[1]Institute of Parallel and Distributed Systems, Shanghai Jiao Tong University
[2]Engineering Research Center for Domain-specific Operating Systems, Ministry of Education, China
[3]Shanghai Key Laboratory of Scalable Computing and Systems, Shanghai Jiao Tong University
zhaoguowang@sjtu.edu.cn

Developer's Responsibility or Database's Responsibility? Rethinking Concurrency Control in Databases

Chaoyi Cheng*†
The Ohio State University

Mingzhe Han*
The Ohio State University

Nuo Xu‡
The Ohio State University

Spyros Blanas
The Ohio State University

Michael D. Bond
The Ohio State University

Yang Wang
The Ohio State University

FINDING 1. *Every studied application uses ad hoc transactions. Among the 91 ad hoc transactions in total, 71 cases are considered critical to the web applications.*

FINDING 3. *There are 7 different lock implementations and 2 validation implementations among the 8 applications we studied. Except for Broadleaf, developers consistently use the same lock/validation implementation in individual applications.*

Ad Hoc Transactions: What They Are and Why We Should Care, Tang et.al., SIGMOD Record 2023.

C4. Due to the complex interplay between microservices' behaviors, asynchronous events are generated to trigger computations. However, avoiding anomalies related to the unintended interleaving of events across microservices is a challenging task.

C6. Developers lack viable and efficient abstractions for transactional queuing in microservice architectures. As a result, anomalies arise due to lack of isolation and ad-hoc fault-handling, leading to challenges on ensuring application correctness.

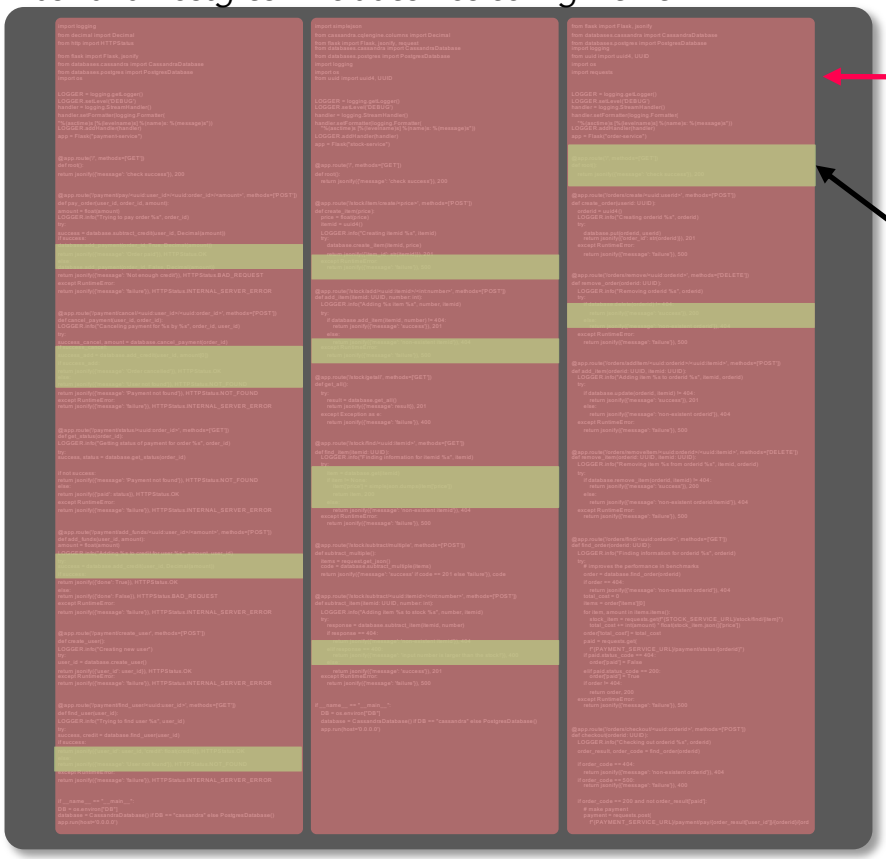
C8. Due to the distributed nature of microservice architectures, eventual consistency is often taken as the de facto consistency model by practitioners. This choice introduces a series of challenges on reasoning about distributed states and invariants.

Challenges with microservices & the like

>90% of programmers' time spent in machine/network failures

(a.k.a. "plumbing")

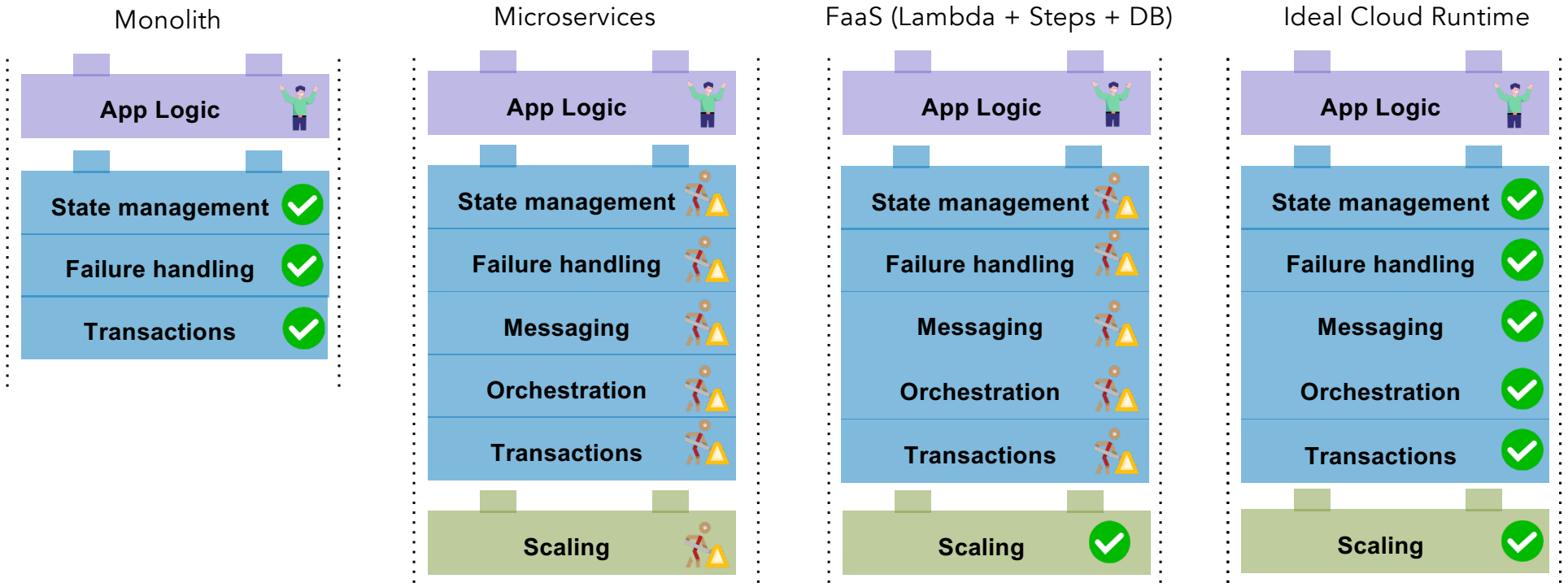
**Actual code (shrinked) of scalable shopping cart, using Flask and Postgres. Excludes K8s config file hell.*

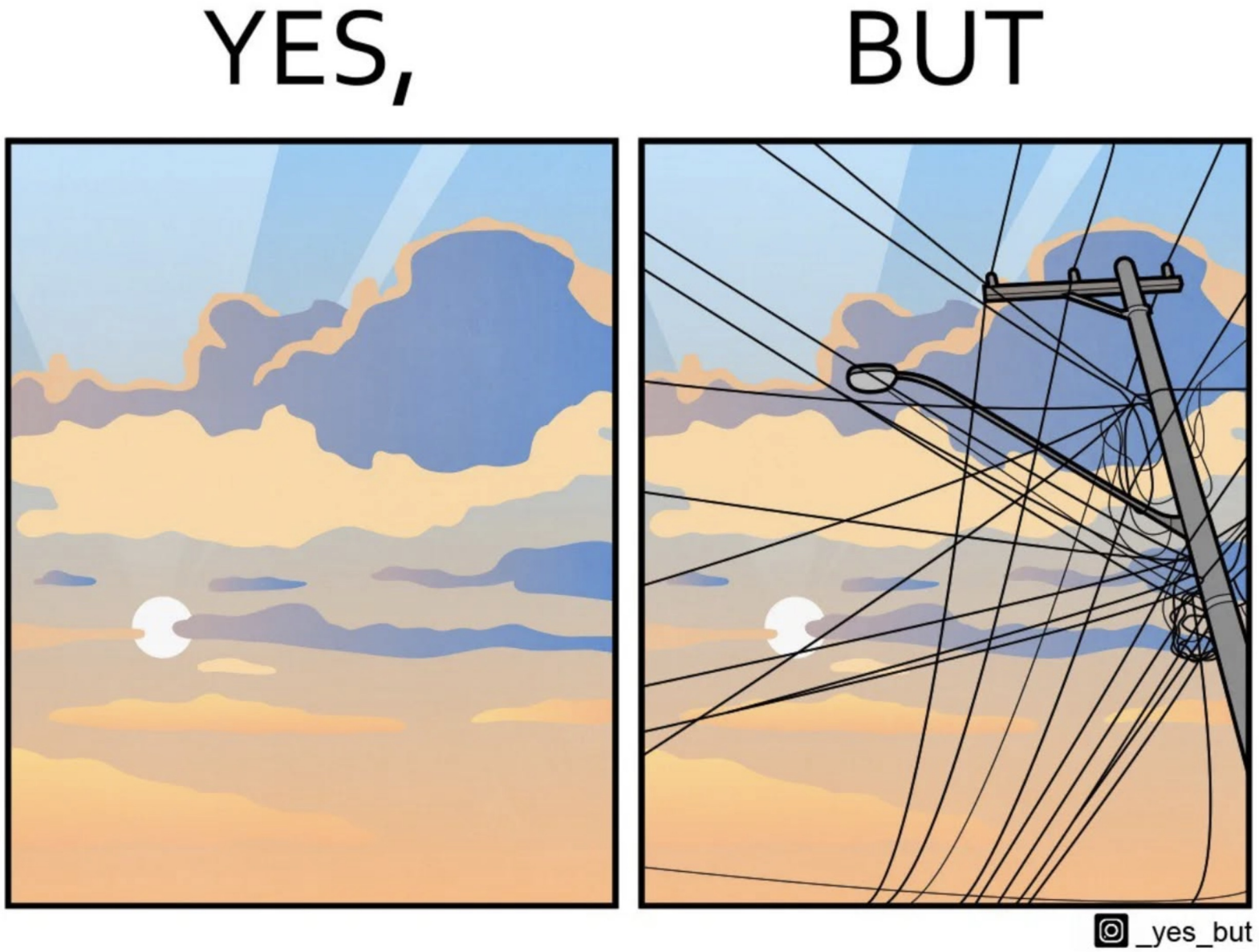


- Retries/Atomicity/Idempotency
- Consistency
- Recovery
- Partitioning
- Scaling
- Maintaining DBs, proxies, containers
- …

"Useful" application-logic code percentage: 5-10%.

Requirements for transactional Cloud apps




YES,
BUT
_yes_but

The rest of this tutorial

Right after
- Programming Models

After the break
- Runtimes
- Benchmarks
- Open Problems