Dataflow Engines for Scalable Cloud Applications: a Maslow's Hammer or Natural Outcome?

Alternative title: an ode to stateful streaming dataflows.

Asterios Katsifodimos

@kasterios

TU Delft

Table of Contents

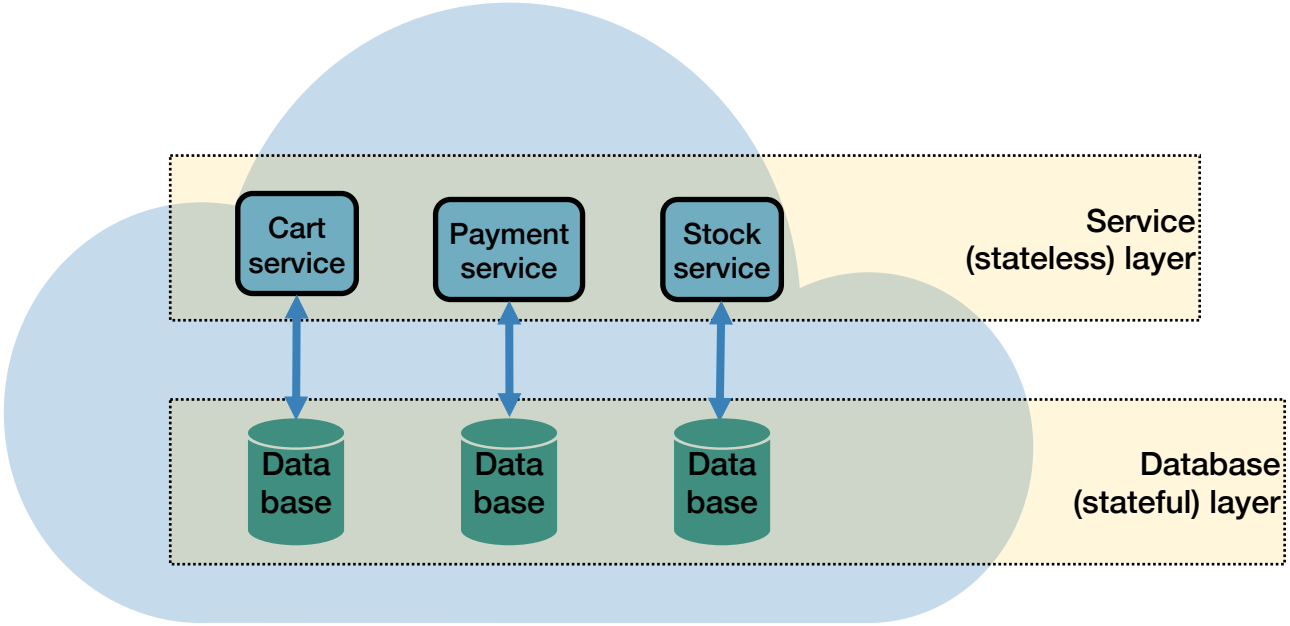
How do people build scalable cloud applications today?

What are the main pain-points?

Is it a good idea to leverage dataflow processors for cloud apps?

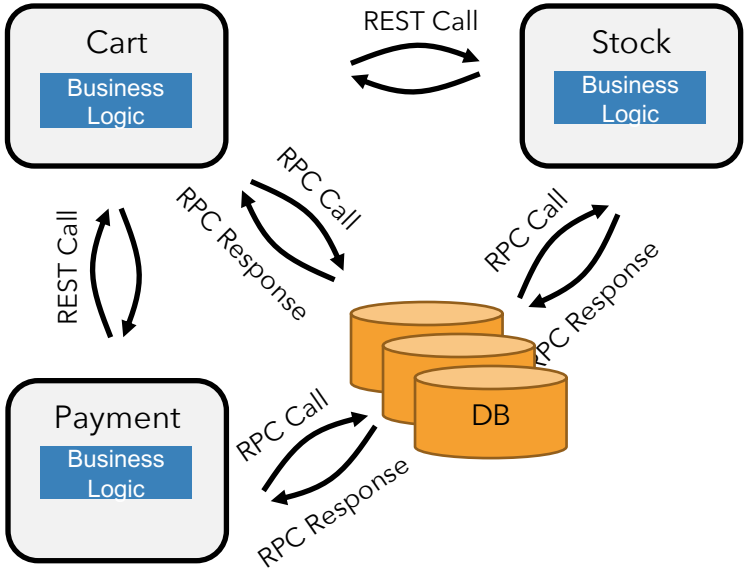
Are current dataflow processors up to the task?

A tale of three Cloud services



To checkout: check & update stock, verify payment, checkout the cart. Atomically!

Services Architecture (1): Easiest Implem.



Services are *stateless*

Database does the heavy-lifting

High latency, costly state access

No guaranteed messaging

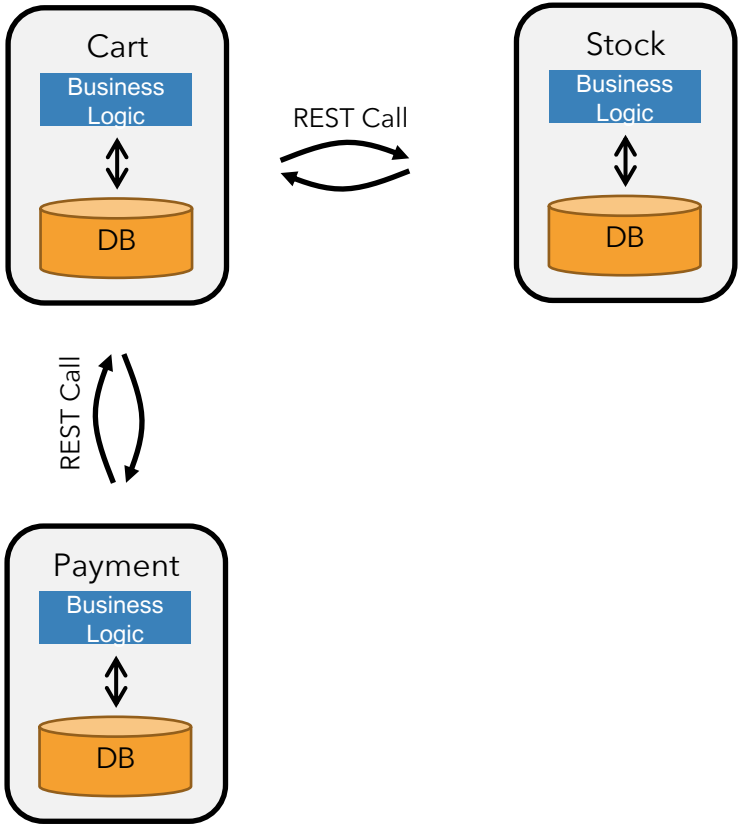
80% of code in the service layer is error checking.

Transactions:

Java XA or

SAGAs.

Services Architecture (2): Embedded State/DB



Low-latency access to local state

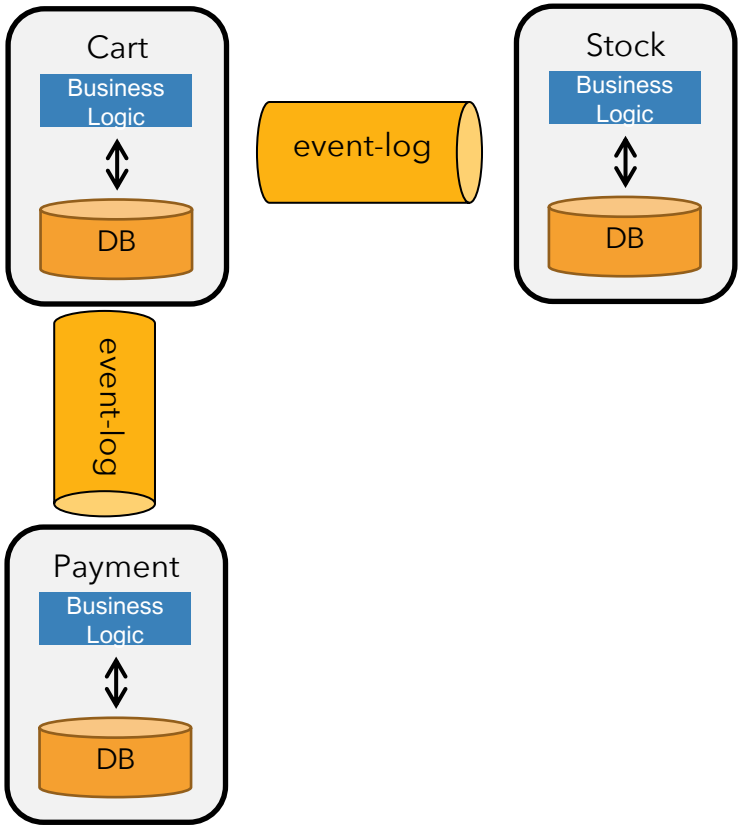
Service calls still expensive

Messaging still not guaranteed

Not obvious how to scale this out

Fault tolerance is hard!

Services Architecture (3): Event Sourcing



Message exchange through an event-log

- Guaranteed at-least once delivery!

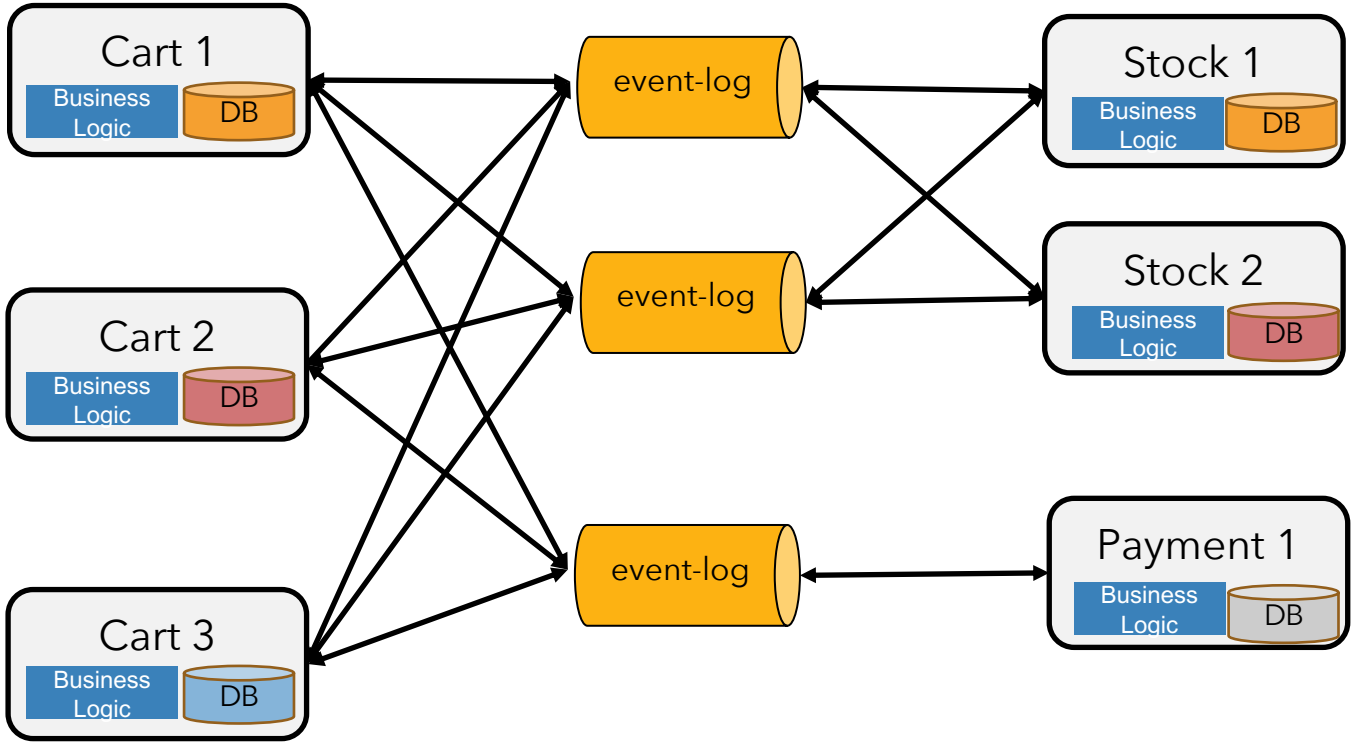
Services are asynchronous/reactive.

If we lose state, we replay the log and rebuild it.

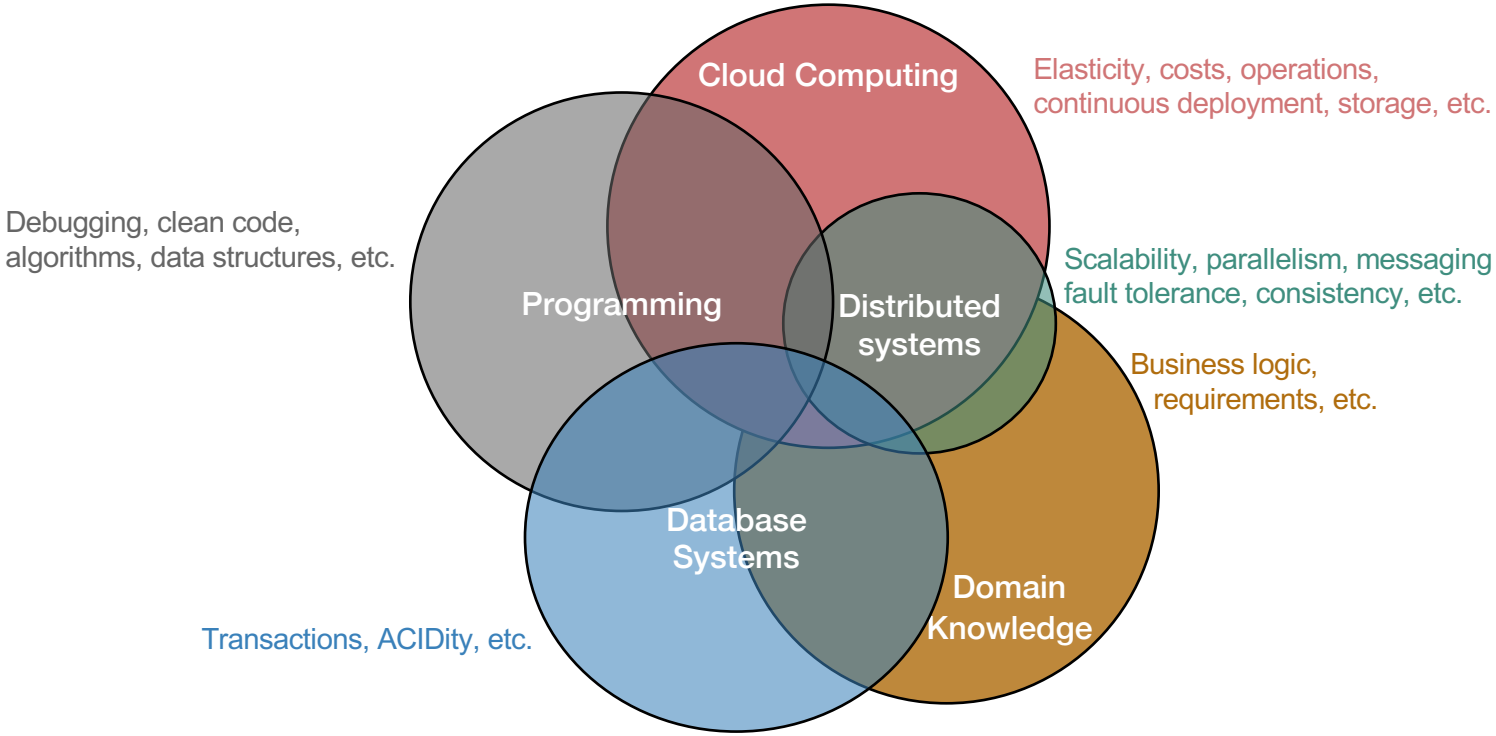
Time-travel debugging, audits, etc. are easier.

Let's scale this!

Services Architecture (4): Scalable Deployment

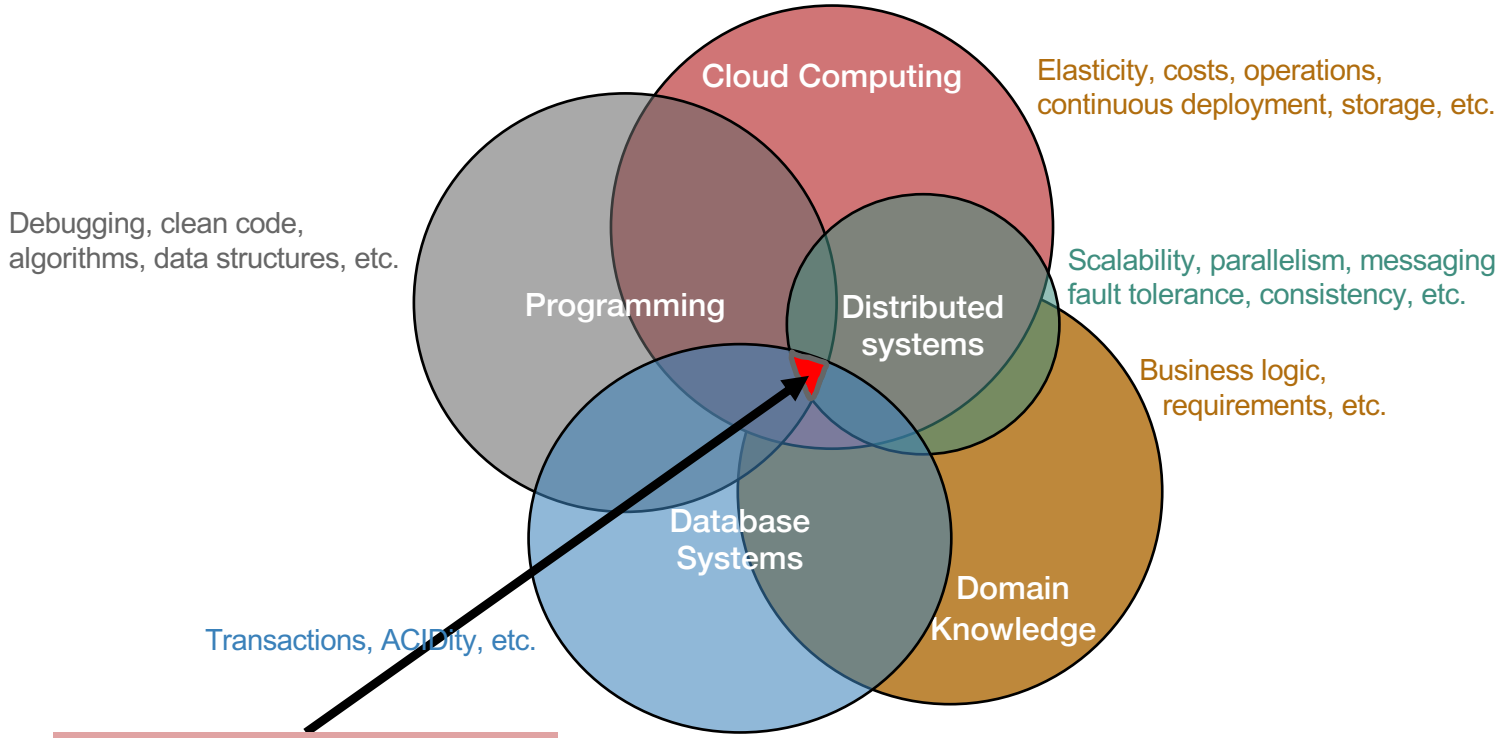


Scalable Cloud Application development is hard!



**this Venn diagram does not represent set-sizes properly.*

Scalable Cloud Application development is hard!



People who can develop scalable Cloud applications.

Meanwhile at the TU Delft campus…

during my MSc class, “Web-scale Data Management”

Challenge: implement Stock, Order, Payment microservices with tools of choice:

Service/business Layer: Flask/Spring, AWS/Azure Lambdas, Akka,…

Persistence Layer: Postgres, CockroachDB, Mongo, Cassandra, Redis,…

Infra Layer: Docker+Kubernetes on Amazon or Google Cloud,…

Goal: 10K per second **order.checkout()** in the Cloud!
Without losing money or stock.

Class runs 4 years (~50 5-person teams).

No team managed so far!

State management is hard,
and the current technology is primitive!

(~~or the students have learned nothing~~)

How to make stateful computations fault tolerant?

How do we (or should we) guarantee message delivery?

How do we consistently query the global state of a full system?

What abstractions should people use?

...

TL;DR

Building scalable Cloud applications is like programming assembly before high-level langauges.

Just more complicated.

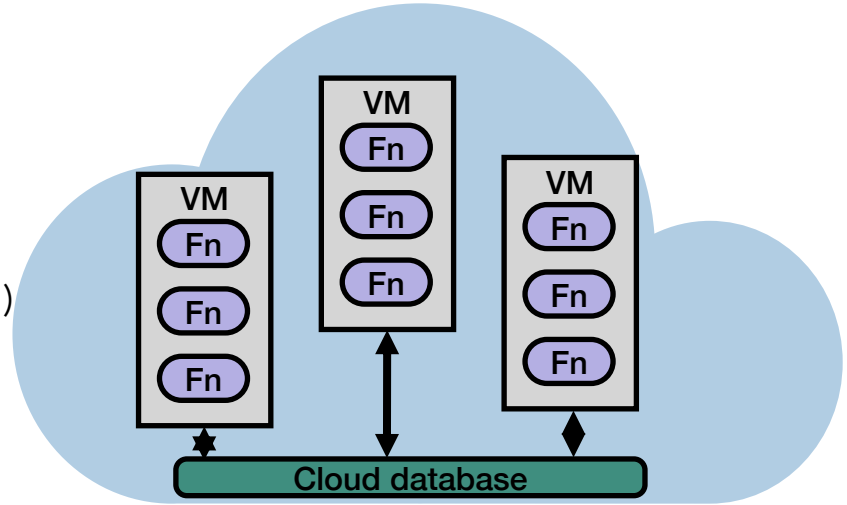
"Two-pizza" dev team in the year 2022.

Wait, what about serverless? That should work!

Azure Functions

Managed Infrastructure (autoscaling, no ops) ✓

Function-based programming model ✓



✗ No State

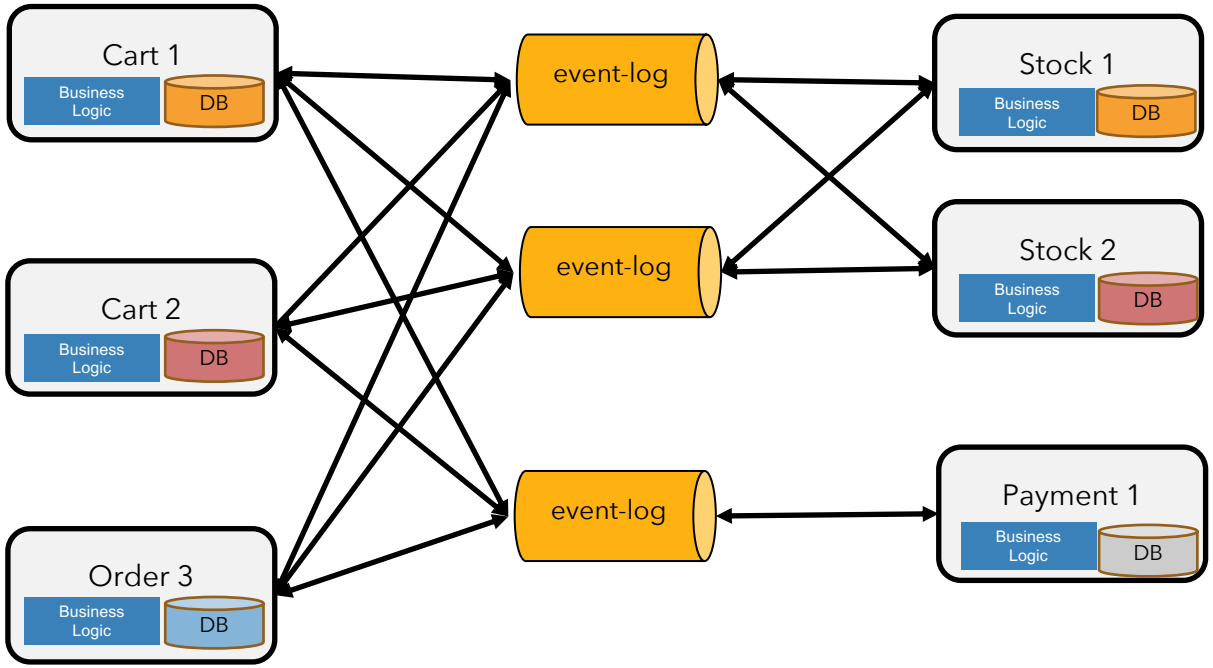
✗ Fn-to-fn calls

✗ Transactions

✗ No natural programming model

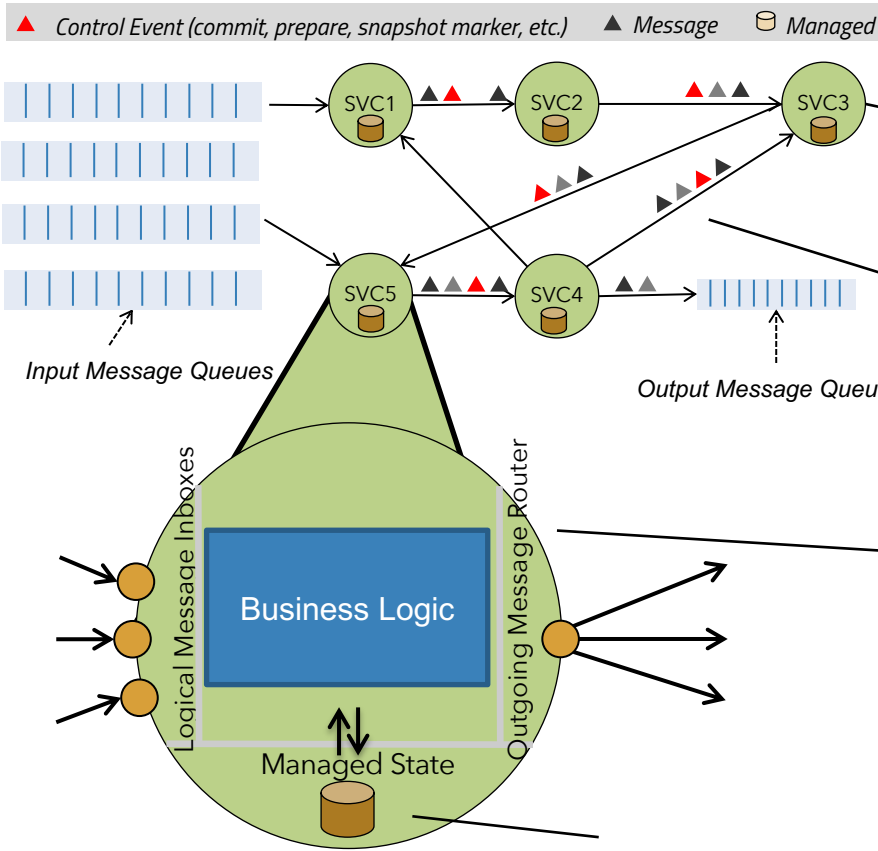
Work in progress at TU Delft

Wait a minute! I have seen this before…



Devs nowadays are implementing parallel, stateful dataflow graphs!

By hand…



Time-travel debugging using checkpoints and message broker.

Guaranteed message delivery and exactly-once execution.

Each operator executes a (group of) microservices or functions that share the same

Operator-local state partitioned or
input for scalability and fault-tolera

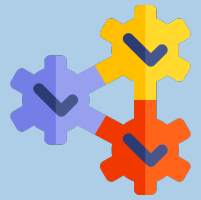
BUT data flow systems are

- **Very rigid** (e.g., redeployment of complete graphs)
- **Internal/invisible state**
- **Cumbersome programming model** for general applications

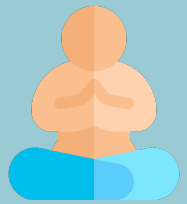
WiP @Delft: dataflow systems & abstractions for the Cloud

High-level Programming Models
[VLDB19, Arxiv, WiP]

Transactions
[ACM DEBS 21, Information Systems 22, WiP]



Global Application State Querying and Consolidation
[ICDE 22]



Zen operations: auto-scaling, migration, etc.
[WiP]

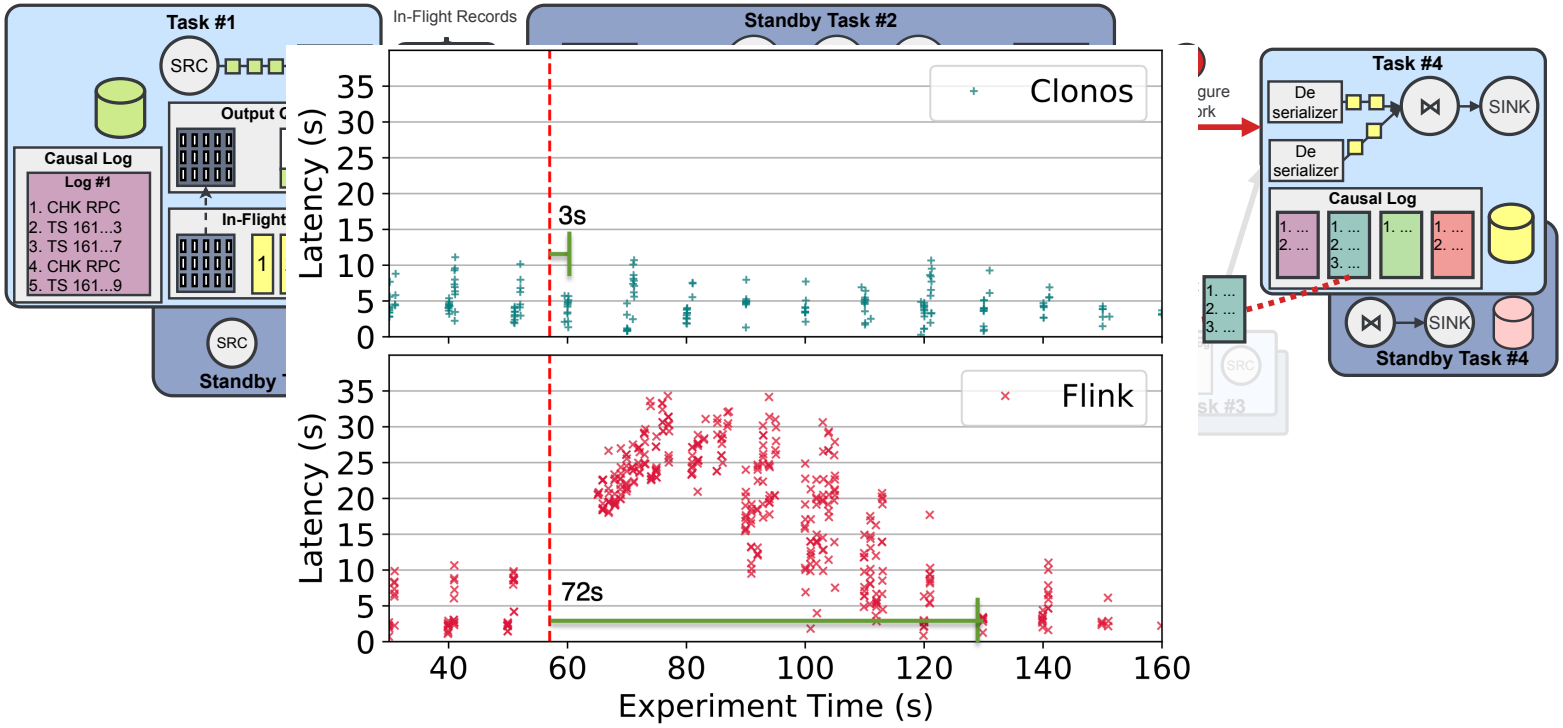
Flexible fault-tolerance with exactly-once messaging.
[SIGMOD 21]

https://github.com/delftdata

[SIGMOD 21] *Clonos: Consistent Causal Recovery for Highly-Available Streaming Dataflows*
Pedro Fortunato Silvestre, Marios Fragkoulis, Diomidis Spinellis, Asterios Katsifodimos.
ACM SIGMOD International Conference on the Management of Data 2021.

https://delftdata.github.io/clonos-web/

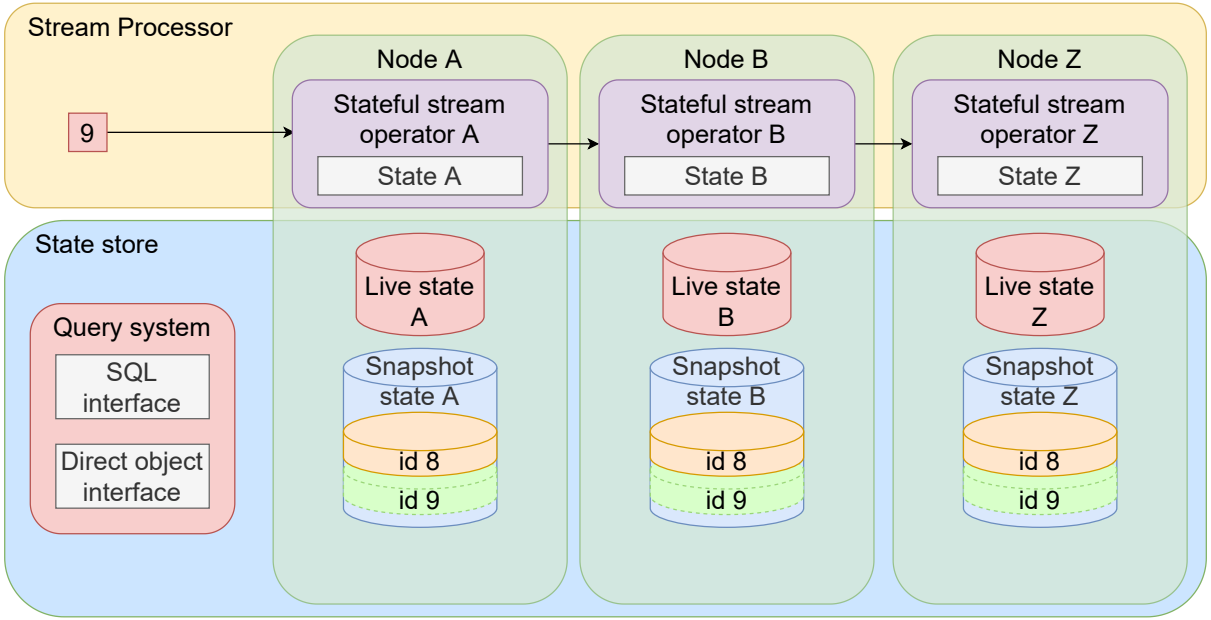
Fault Tolerance: Local-recovery with exactly-once guarantees with causal logging



[ICDE 22] *S-Query: Opening the Black Box of Internal Stream Processor State*
Jim Verheijde, Vassilis Karakoidas, Marios Fragkoulis, Asterios Katsifodimos.
In the Proceedings of the 2022 IEEE 38th International Conference on Data Engineering (ICDE).

https://github.com/delftdata/s-query

Querying Internal Operator State on the Fly



Ability to query live or snapshotted partitioned state with SQL.

Different isolation guarantees depending on setting.

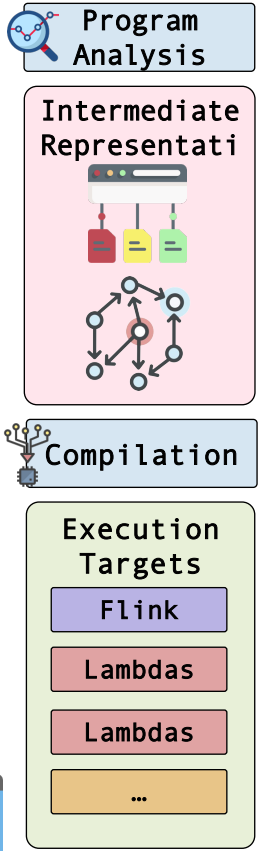
[***Arxiv***] *"Stateful Entities: Object-oriented Cloud Applications as Distributed Dataflows"*
Wouter Zorgdrager, Kyriakos Psarakis, Marios Fragkoulis, Eelco Visser, Asterios Katsifodimos
https://arxiv.org/abs/2112.00710

https://github.com/delftdata/stateflow

WiP: From pure Python OO to Flink, Beam, Lambdas

```python
@entity
class User:
    def __init__(self, username: str):
        self.username: str = username
        self.balance: int = 1

    def __key__(self):
        return self.username

    @transactional
    def buy_item(self, amount: int, item: Item) -> bool:
        total_price = amount * item.price

        if self.balance < total_price:
            return False

        # Decrease the stock.
        available_stock = item.update_stock(-amount)

        if not available_stock:
            item.update_stock(amount)
            return False

        self.balance -= total_price
```

```python
@entity
class Item:
    def __init__(self, item_name: str, price: int):
        self.item_id: str = item_id
        self.stock: int = 0
        self.price: int = price

    def __key__(self):
        return self.item_id

    def update_stock(self, amount: int) -> bool:
        self.stock += amount
        return stock>=0
```

Python	Dataflow
Class	Operator
Object State	Operator State
Function Call Arguments	Event (header)
Return Value	Event (payload)

Exactly-once guarantees form underlying runtime can hide all failures from the application.

Programmers code *business logic only.*

TL;DR

Dataflow engines can be an excellent execution engine for scalable and consistent, cloud-native applications.

We still need to make them less rigid, auto-scaling and Cloud-friendly.
And especially: programmable by normal folks.

*hiring postdocs!

Further Reading

[ICDE 22] *S-Query: Opening the Black Box of Internal Stream Processor State*
Jim Verheijde, Vassilis Karakoidas, Marios Fragkoulis, Asterios Katsifodimos.
In the Proceedings of the 2022 IEEE 38th International Conference on Data Engineering (ICDE).

[SIGMOD 21] *Clonos: Consistent Causal Recovery for Highly-Available Streaming Dataflows*
Pedro Fortunato Silvestre, Marios Fragkoulis, Diomidis Spinellis, Asterios Katsifodimos.
ACM SIGMOD International Conference on the Management of Data 2021.

[DEBS 21] *Distributed Transactions on Serverless Stateful Functions*
Martijn De Heus, Kyriakos Psarakis, Marios Fragkoulis, Asterios Katsifodimos.
ACM International Conference on Distributed and Event-based Systems (DEBS) 2021.

[SIGMOD 20] *Beyond Analytics: The Evolution of Stream Processing Systems*
Paris Carbone, Marios Fragkoulis, Vasiliki Kalavri, Asterios Katsifodimos.
ACM SIGMOD International Conference on Management of Data 2020 (tutorial).

[VLDB 19] *Stateful Functions as a Service in Action*
Adil Akhter, Marios Fragkoulis, Asterios Katsifodimos.
International Conference on Very Large Data Bases (VLDB) 2019 (demo).

[EDBT 19] *Operational Stream Processing: Towards Scalable and Consistent Event-Driven Applications*
Asterios Katsifodimos, Marios Fragkoulis.
International Conference on Extending Database Technology (EDBT) 2019.