Stateful Entities: Object-oriented Cloud Applications as Distributed Dataflows


Kyriakos Psarakis
TU Delft – also here at CIDR


Wouter Zorgdrager
TU Delft


Marios Fragkoulis
Delivery Hero & TU Delft

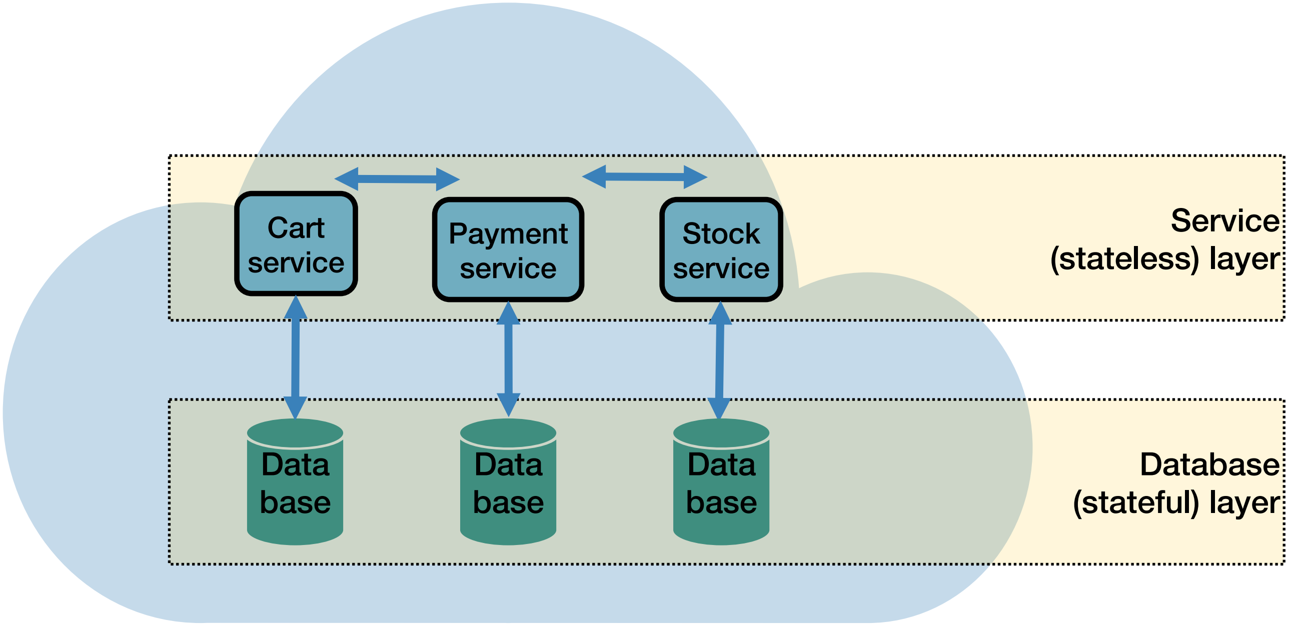

Guido Salvaneschi
University of St.Gallen


Asterios Katsifodimos
TU Delft

@kasterios

TU Delft

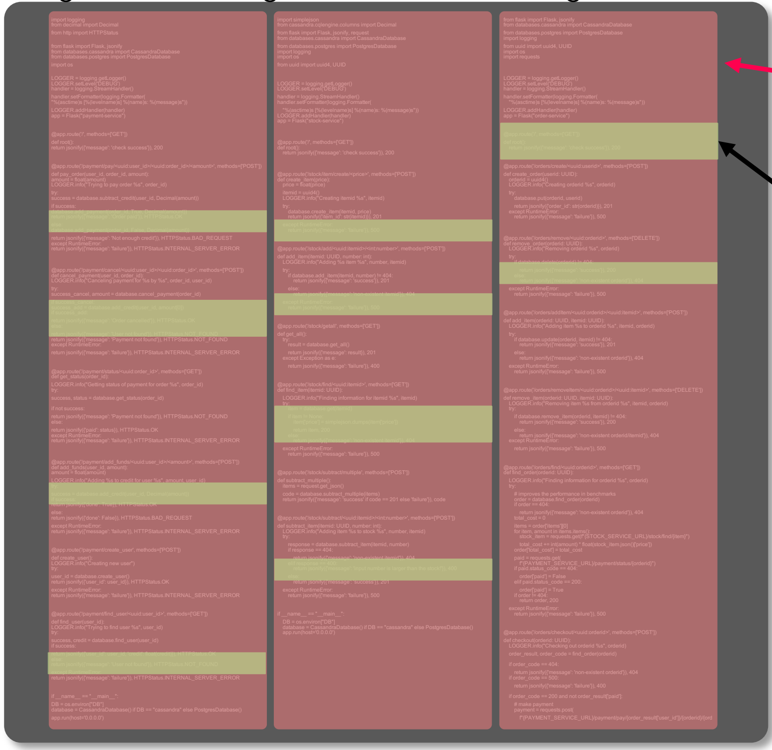
A tale of three Cloud services



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

>90% of programmers' time spent in machine/network failures (a.k.a. "plumbing")

**Actual code (shrinked) from MSc students at TU Delft using Flask and Postgres. Excludes K8s config file hell.*



Plumbing (~90%):

- Failure management code
- Retries
- Idempotency
- Atomicity & consistency
- Recovery
- Parallelization
- (auto-) Scaling
- …

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

Wait, what about serverless / FaaS? That should work!

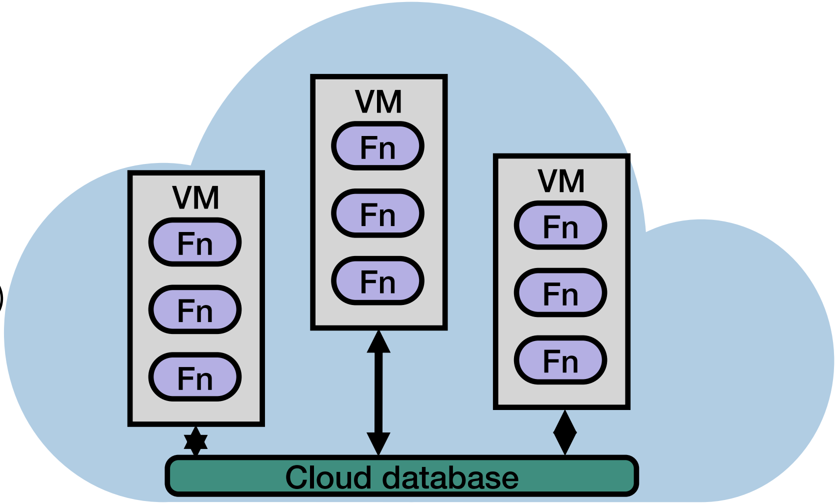
AWS Lambda

Azure Functions

Google Cloud Functions

Managed Infrastructure (autoscaling, no ops) ✓

Function-based programming model ✓

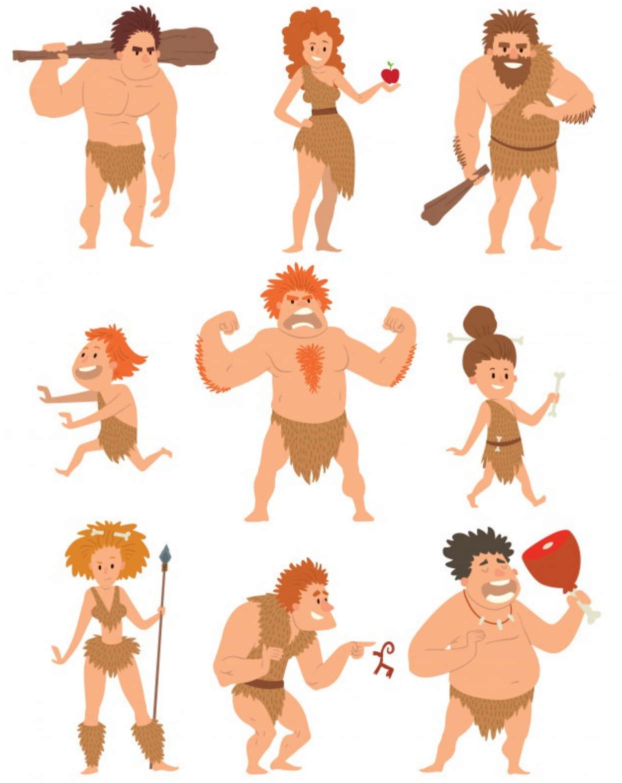


- ✗ No State
- ✗ Fn-to-fn calls
- ✗ Transactions
- ✗ No natural programming model

Central Question

Can we hide Cloud failures and scalability issues from programmers?

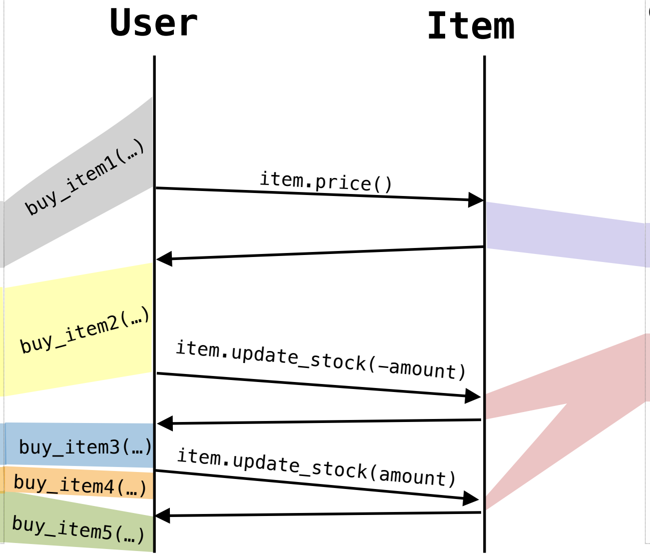
To what degree?



Cloud programmers in the year 2022.

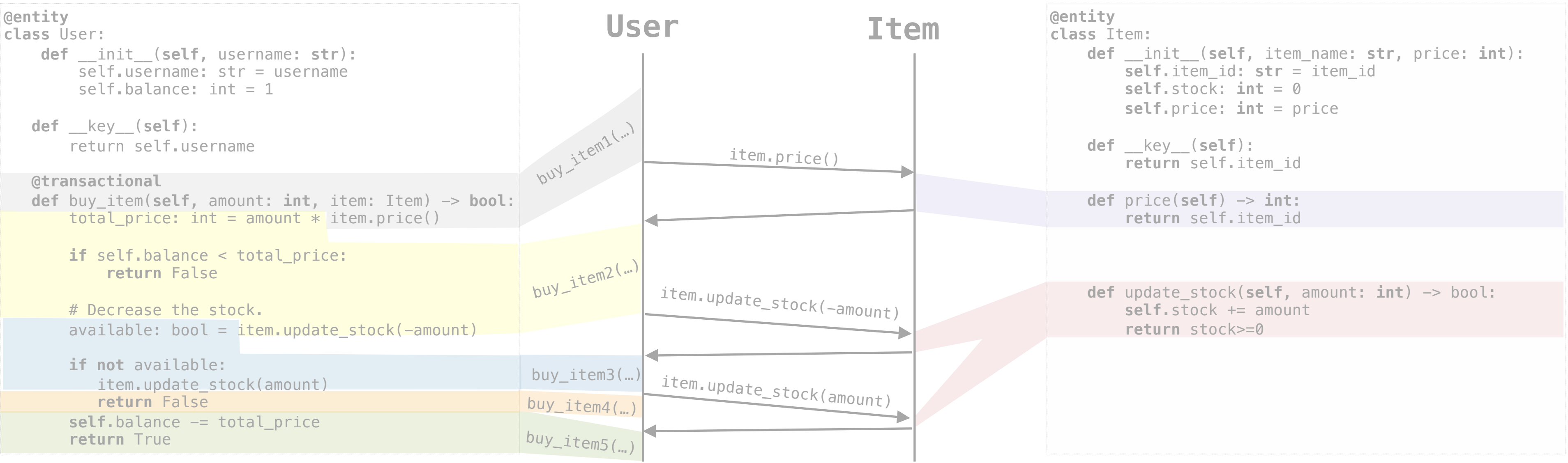
Step 1: Program analysis (using Python.ast) & Function Splitting

```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: int = amount * item.price()

        if self.balance < total_price:
            return False

        # Decrease the stock.
        available: bool = item.update_stock(-amount)

        if not available:
            item.update_stock(amount)
            return False
        self.balance -= total_price
        return True
```

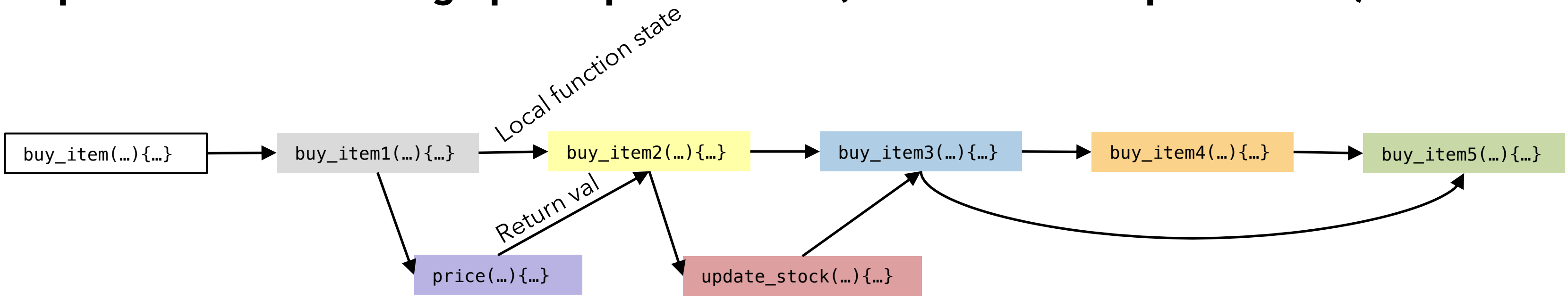



```python
@entity
class Item:
    def __init__(self, item_name: str, price: int):
        self.item_id: str = item_id
        self.stock: int = 0
        self.price: int = price

    def __key__(self):
        return self.item_id

    def price(self) -> int:
        return self.item_id

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

Step 1: Program analysis (using Python.ast) & Function Splitting

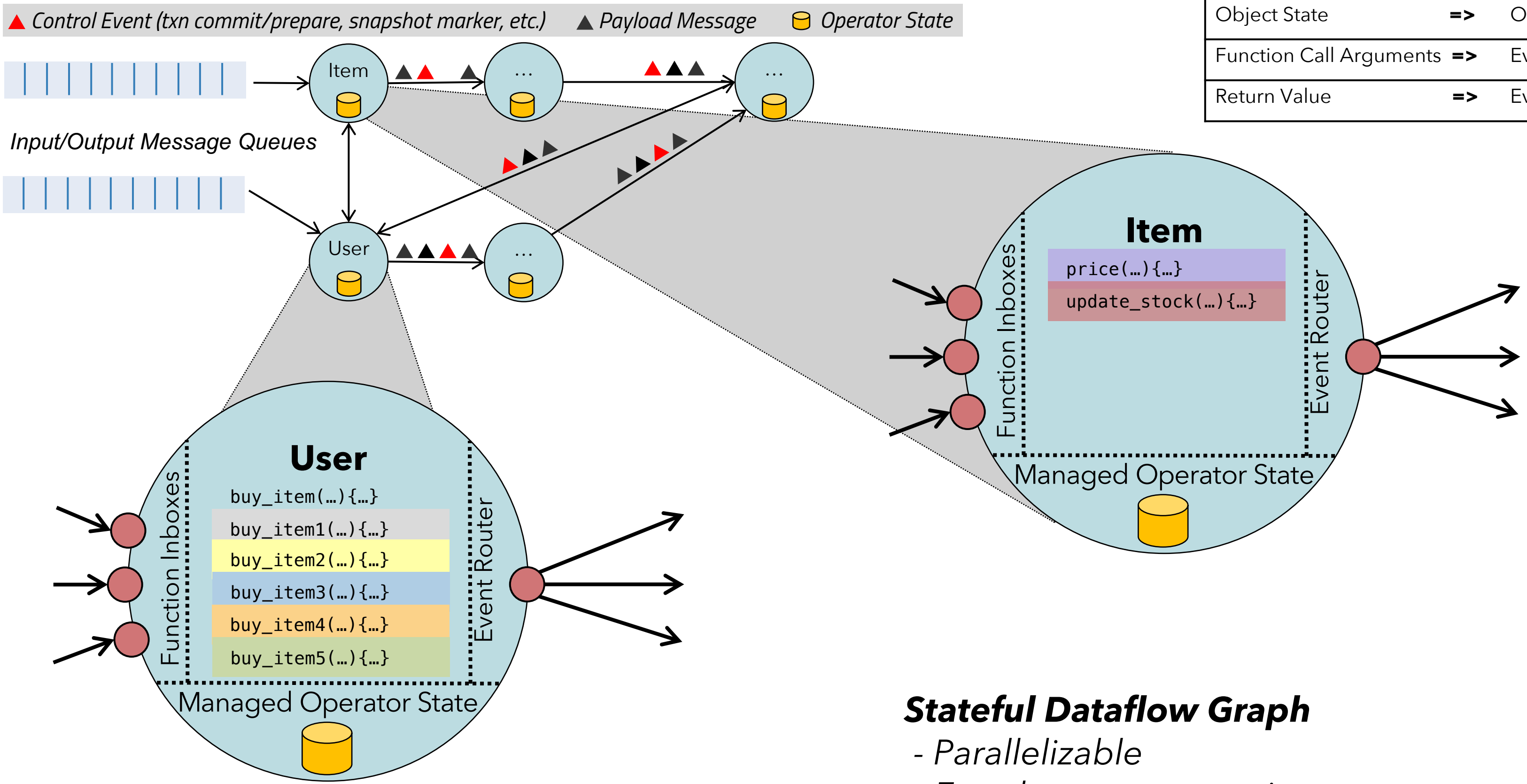


Step 2: stateful dataflow graph of split functions (+ state machines per function)



Step 3: Deployment to a Streaming Dataflow Engine

Python		Dataflow
Class	=>	Operator
Object State	=>	Operator State
Function Call Arguments	=>	Event
Return Value	=>	Event



Stateful Dataflow Graph

- *Parallelizable*
- *Exactly-once processing guarantees*
- *100s thousands of events-per-second per core*

samza Flink kafka

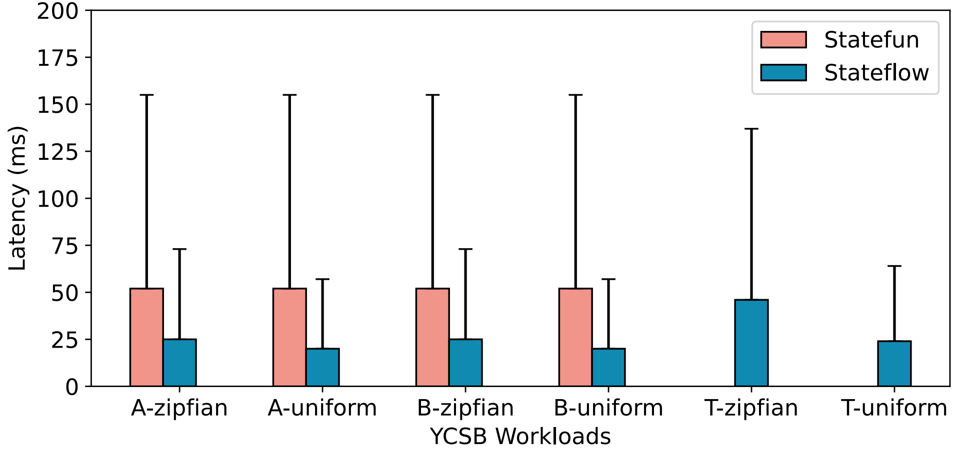
Low-latency & high-throughput “for free”

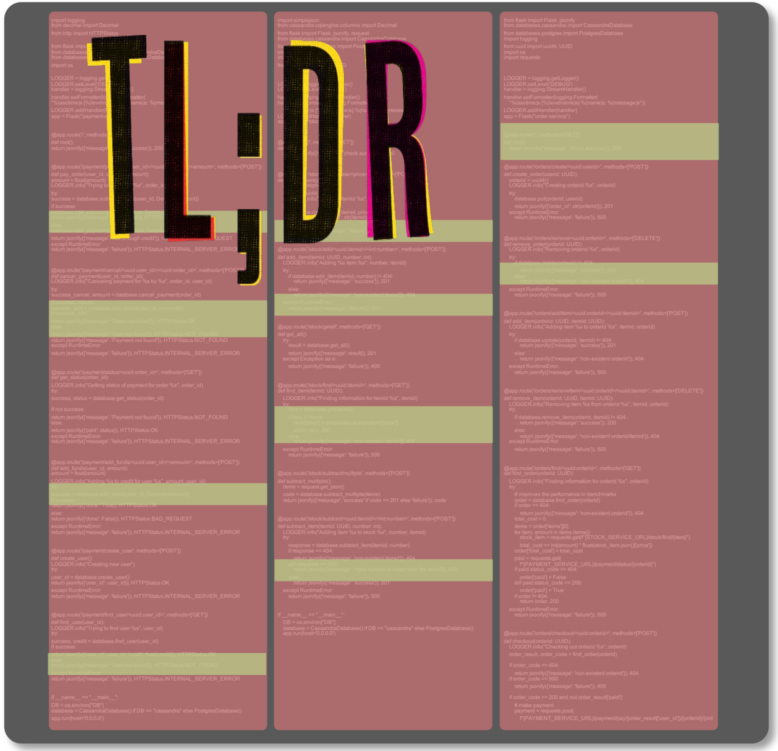
YCSB Workload (zipfian vs. uniform distributions)

Program analysis in Python ASTs, spits out dataflow graphs

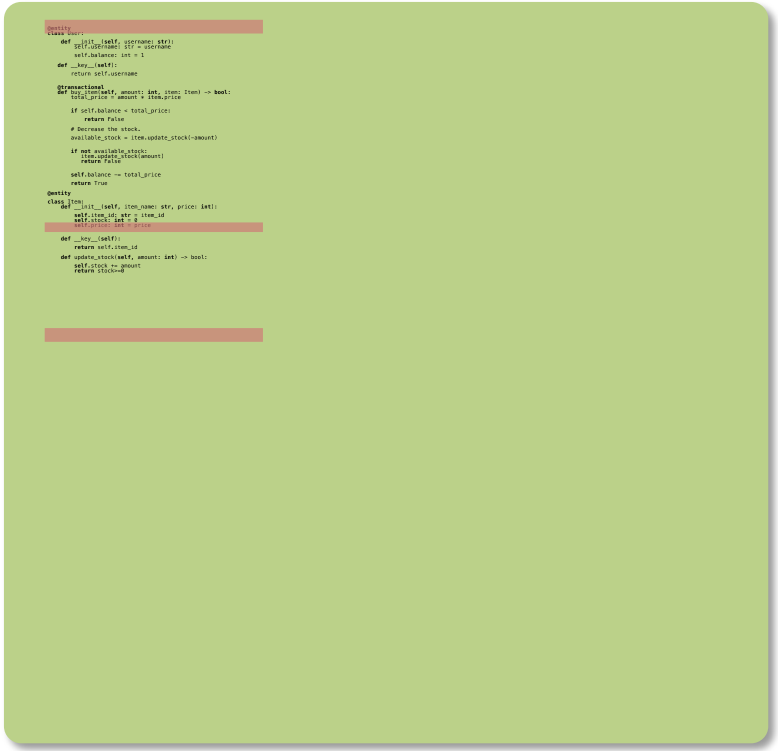
Compiled into:

- Apache Flink (Statefun)
- Home-made Dataflow system (Stateflow)





vs.



https://github.com/delftdata/stateflow

Hiring PhD students & postdocs

- Dataflows, programming languages & transactions (Asterios)
- DB4ML + Data Integration (Rihan)

Backup

TL;DR

*Dataflow engines **can** be the universal execution engines for scalable and consistent, cloud-native applications (batch, stream, ML, transactional workloads).*

*We still need to make them **less rigid**, **auto-scaling**, **transactional**, and **Cloud-friendly**.*

And programmable by normal folks.

StateFlow

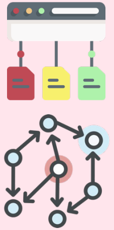
A "holistic" approach of a **programming model** and **dataflow execution engine** for **Cloud applications.**

```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
       return True
```

```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
```

Program Analysis

Intermediate Representati



Compilation

Execution Targets

- Flink
- Lambdas
- Styx*
- …

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

MSc students at TU Delft enjoy the ride

during my MSc class, "Web-scale Data Management" (2018 - today)

Challenge: implement three idependent Cloud services: Stock, Order, Payment

Goal: 10K/second concurrent checkouts, without losing money or stock

Using any tech/DB *(Lambdas, Flask, Spring, Cockroach, Dynamo, K8s, …)*

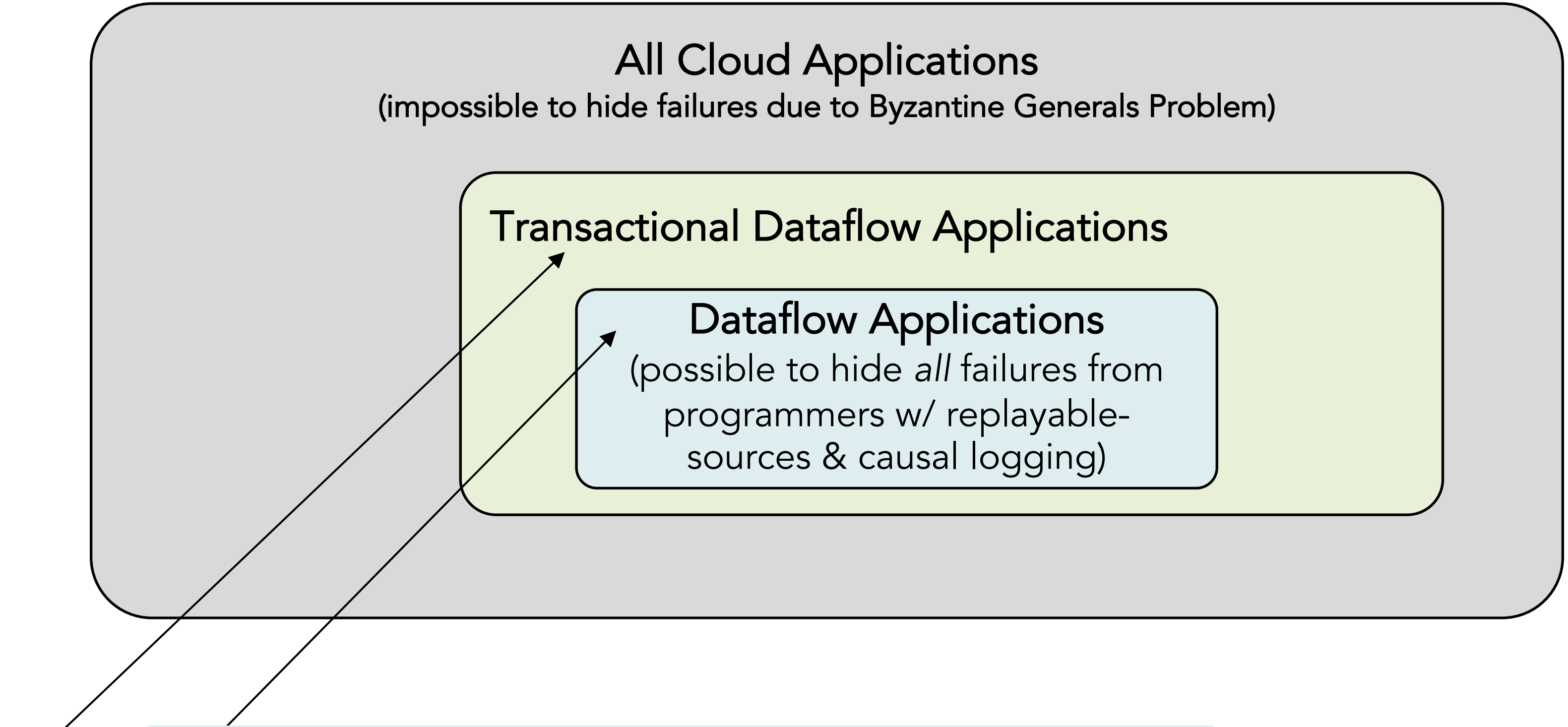
+ ***Cloud resources.***

Class runs 4 years (~50x5-person teams).

No team managed 10K consistent transactions/s.

The current technology is primitive!

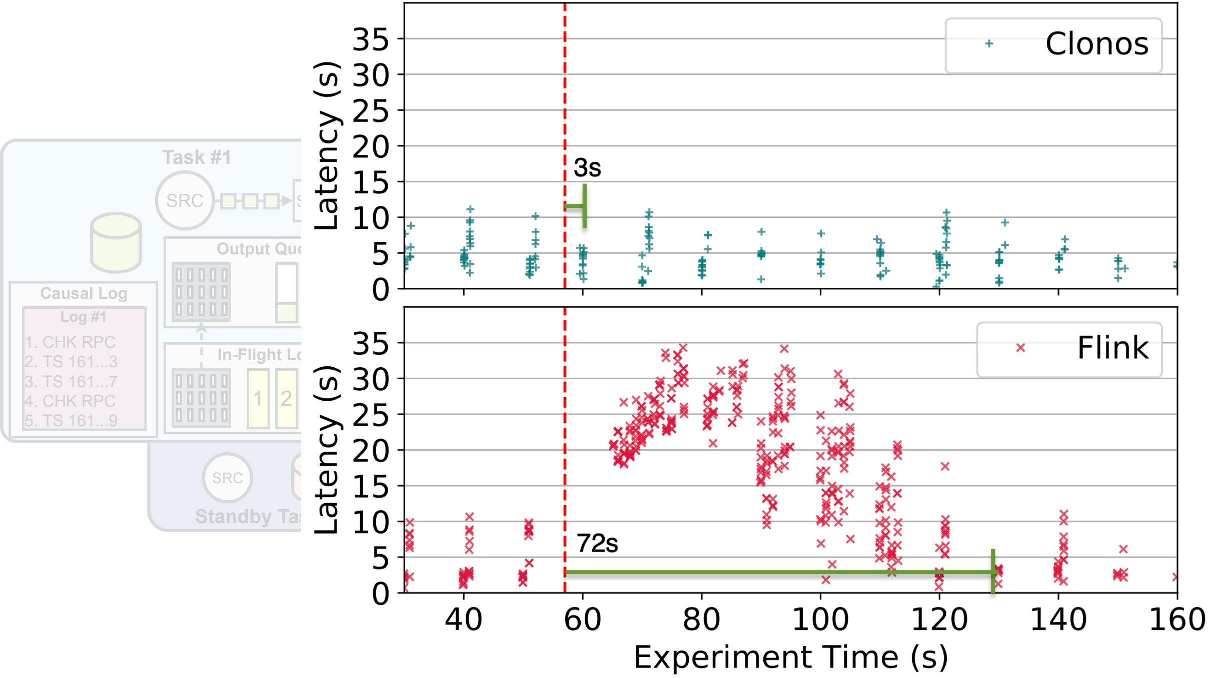
Beyond the Byzantine Generals problem



e.g., Database queries, stream processing, Big Data analytics, ML pipelines, …

WiP: shopping cart applications, complex transactional workflows, fraud detection systems, etc.

Local-recovery & exactly-once guarantees for dataflows



[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/

Publications Connected to this project

[CIDR23] *Stateful Entities: Object-oriented Cloud Applications as Distributed Dataflows*
Kyriakos Psarakis, Wouter Zorgdrager, Marios Fragkoulis, Guido C Salvaneschi, Asterios Katsifodimos

[Information Systems 22] *Transactions across serverless functions leveraging stateful dataflows*
Martijn De Heus, Kyriakos Psarakis, Marios Fragkoulis, Asterios Katsifodimos.
Elsevier Information Systems Journal, 2022

[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.

[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.