USENIX ATC '25

July 07-09, 2025
BOSTON, MA, USA

Towards High-Performance Transactional Stateful Serverless Workflows with Affinity-Aware Leasing

Jianjun Zhao, *Haikun Liu, Shuhao Zhang, Haodi Lu, Yancan Mao, Zhuohui Duan, Xiaofei Liao, Hai Jin*

華中科技大學
HUAZHONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

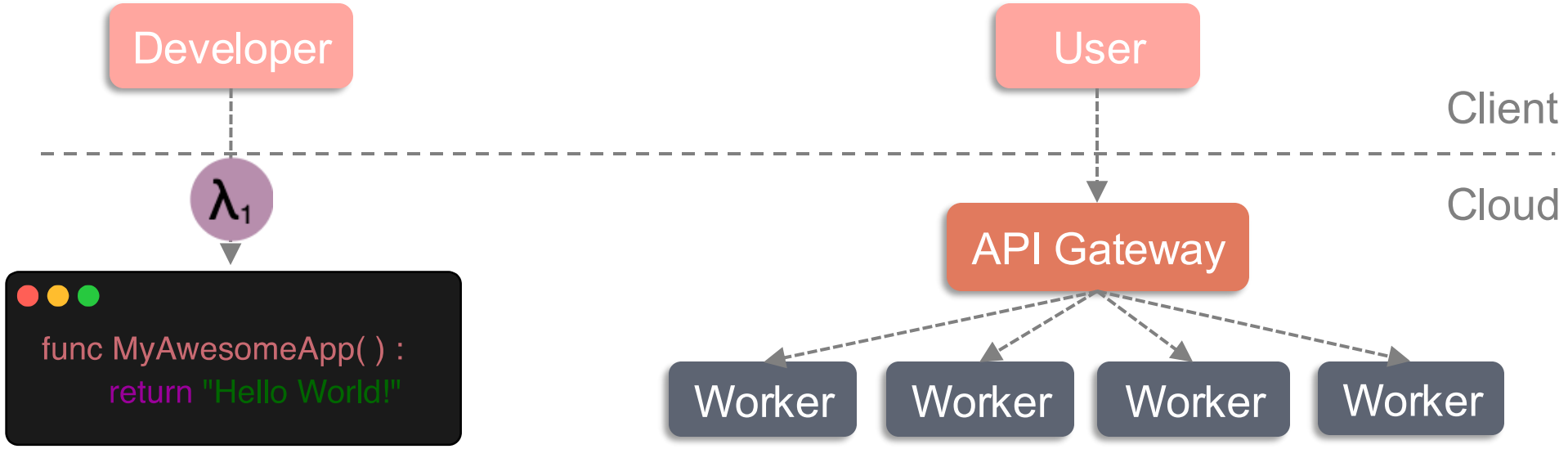
NUS National University of Singapore

Outline

- ❖Background and Motivation
- ❖RTSFaaS Design
- ❖Evaluation Results
- ❖Conclusion

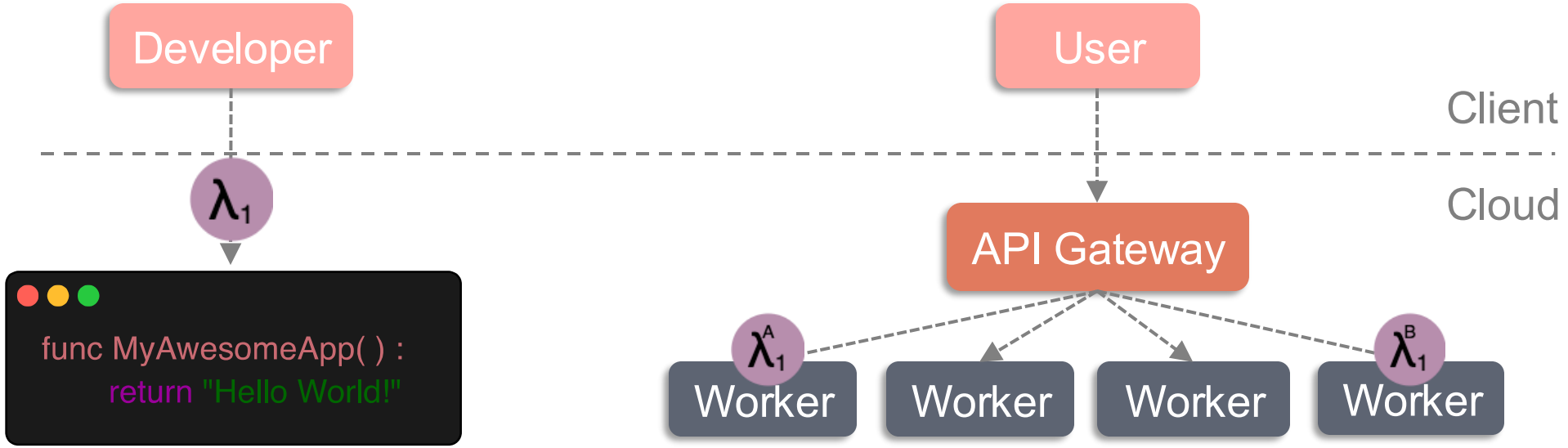
Function-as-a-Service

- What is Function-as-a-Service (FaaS)?
  - Developers write applications as ***functions*** in a high-level programming language



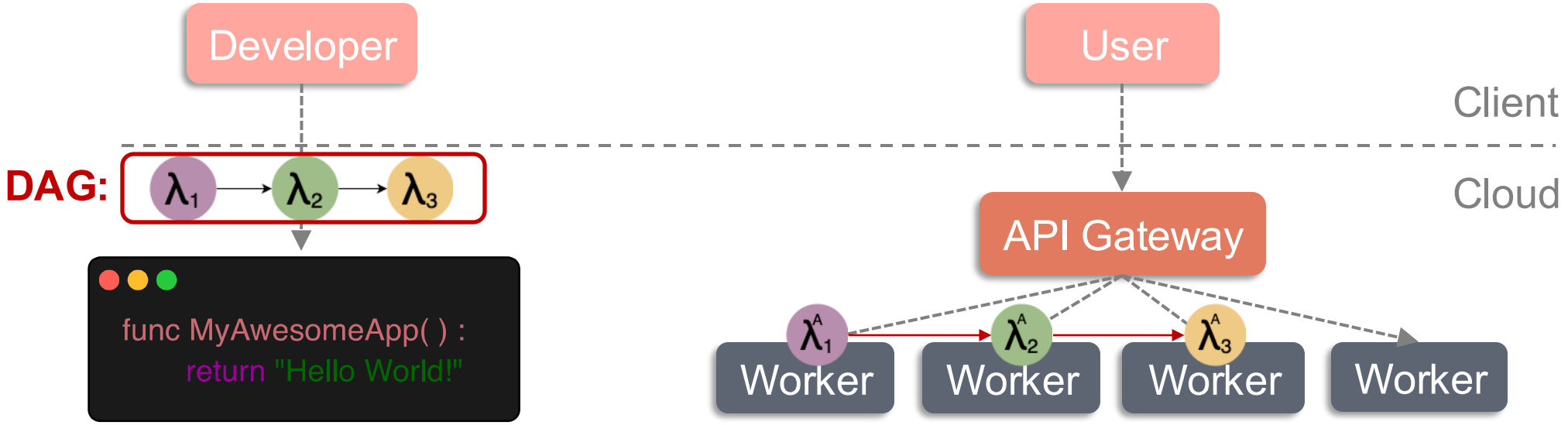
Function-as-a-Service

- What is Function-as-a-Service (FaaS)?
  - Developers write applications as ***functions*** in a high-level programming language
  - Upon invocations, cloud platforms automatically manage deployment of functions



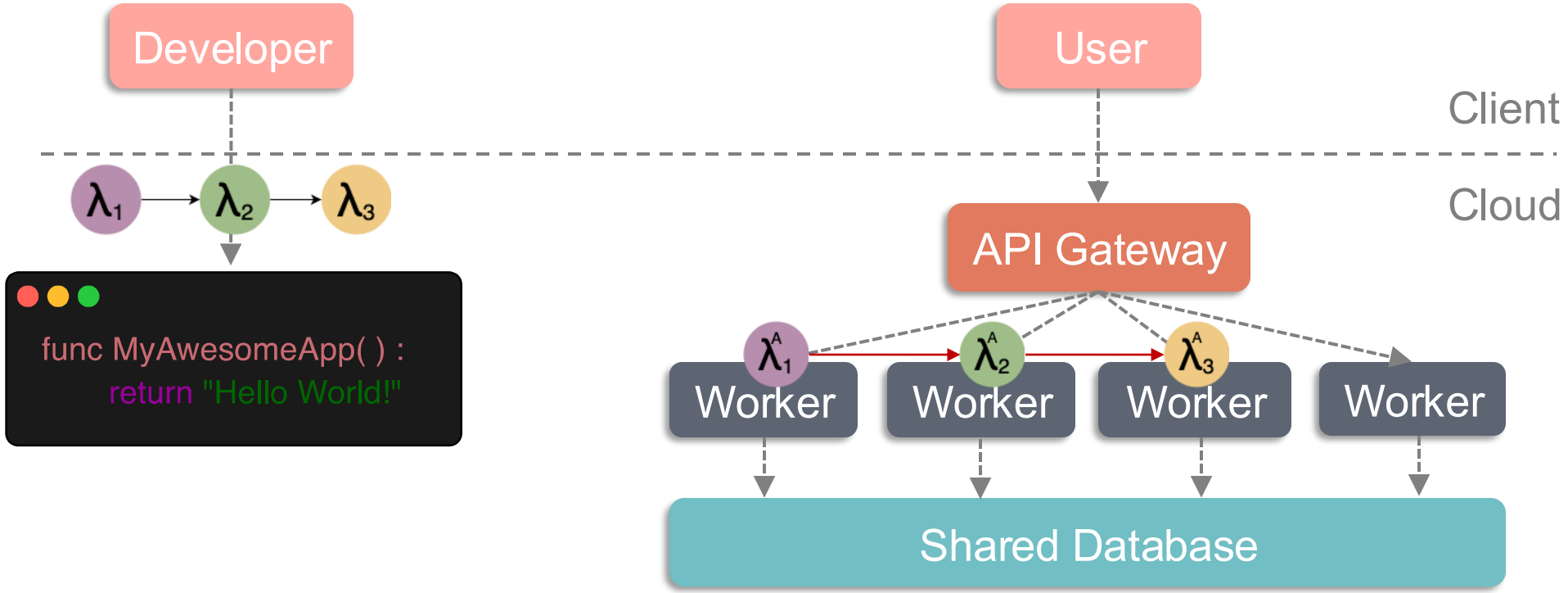
Function-as-a-Service

- What is Function-as-a-Service (FaaS)?
  - Developers write applications as ***functions*** in a high-level programming language
  - Upon invocations, cloud platforms automatically manage deployment of functions
  - Compose multiple functions into a ***workflows***



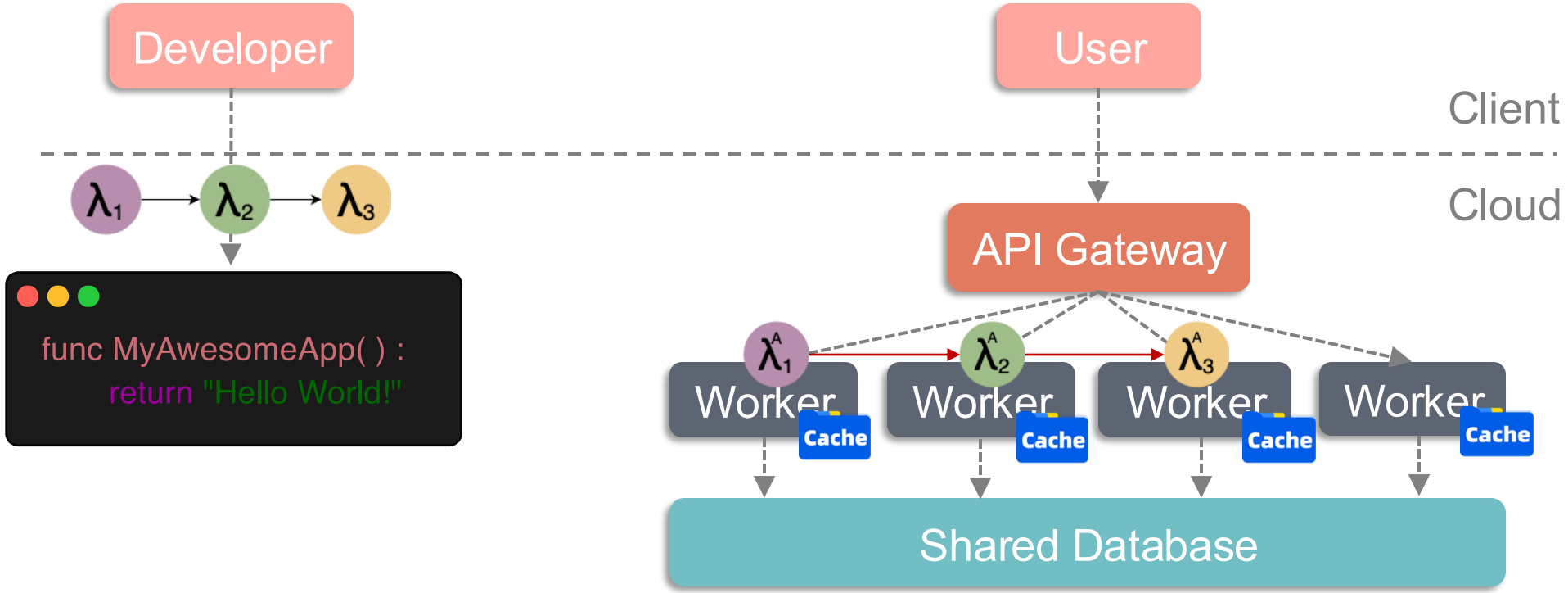
Stateful Function-as-a-Service

- Achieve flexible autoscaling by disaggregating the computation and storage
  - Built on **Remote DB**, such as Amazon DynamonDB/ S3 [HotStorage'22, SOSP'21, OSDI'20]



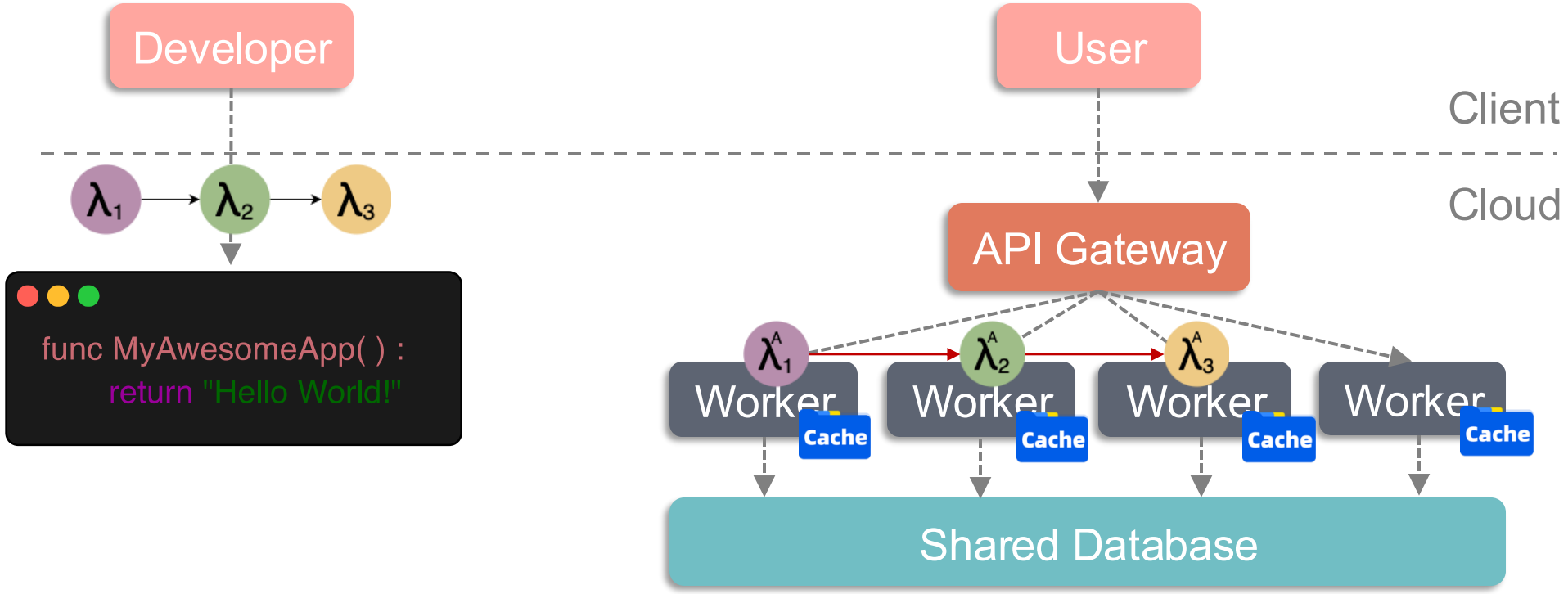
Stateful Function-as-a-Service

- Achieve flexible autoscaling by disaggregating the computation and storage
  - Built on **Remote DB**, such as Amazon DynamonDB/ S3 [HotStorage'22, SOSP'21, OSDI'20]
  - Using local data cache to improve performance[VLDB'20, SIGMOD'20, SOSP'21, VLDB'25]



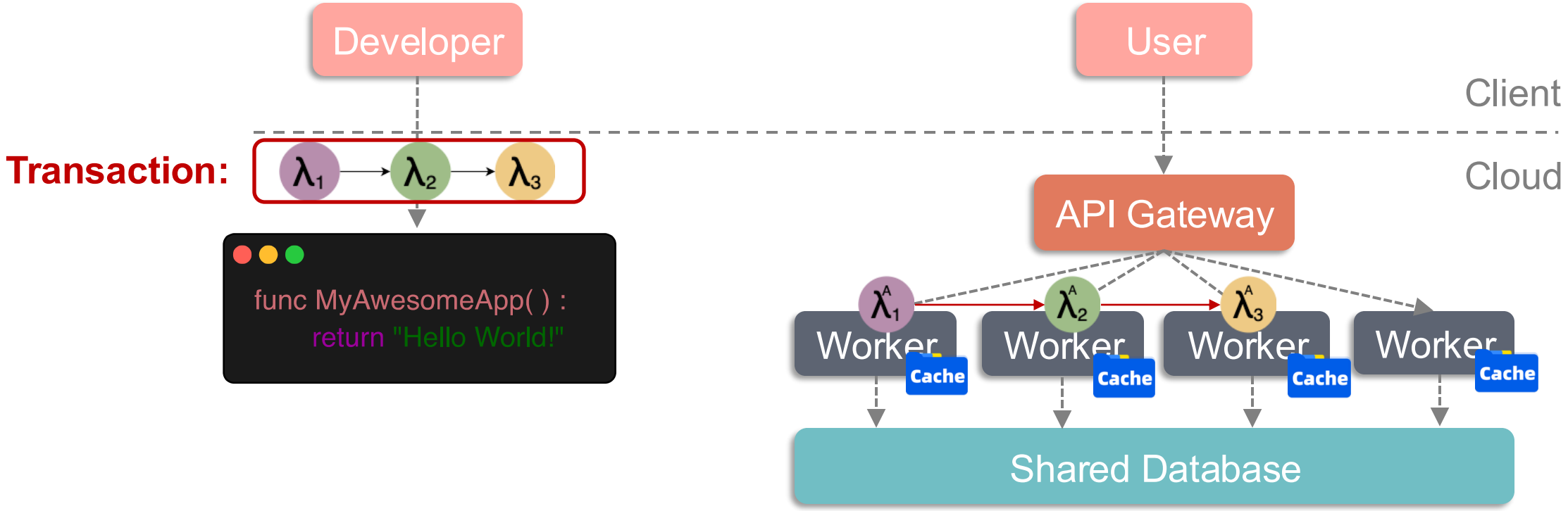
Requirement for State Consistency

- Transactional serializability consistency[SOSP'21, OSDI'20]
  - Applications require the end-to-end workflow to be processed transactionally



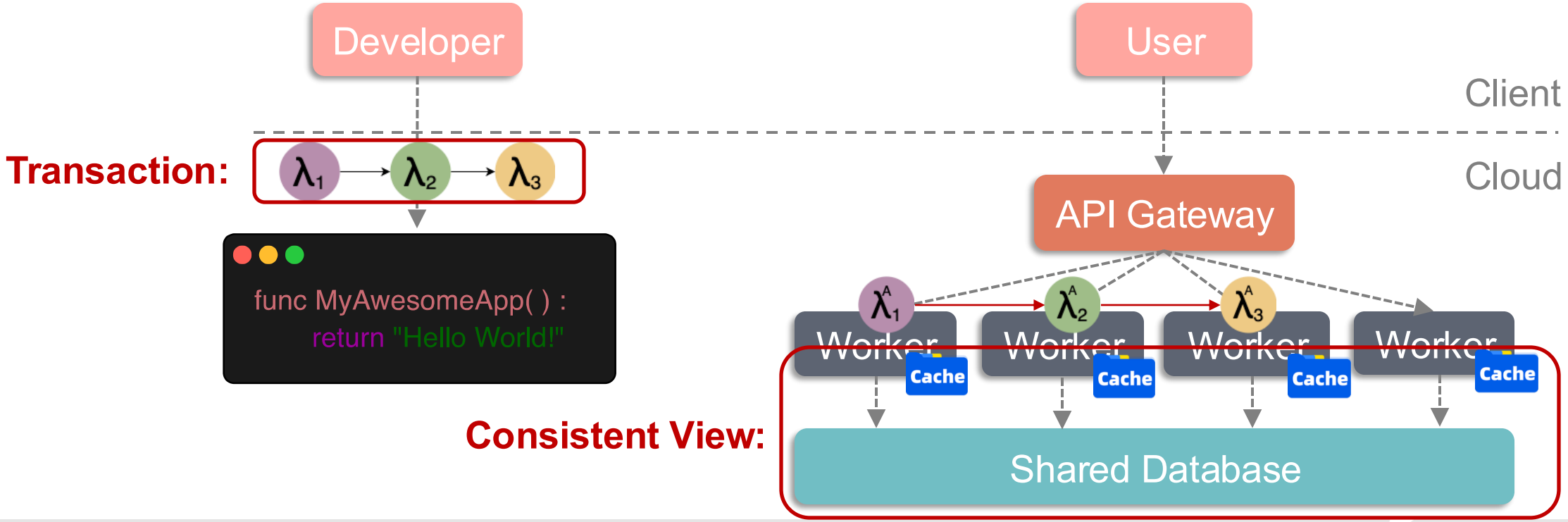
Requirement for State Consistency

- Transactional serializability consistency[SOSP'21, OSDI'20]
  - Applications require the end-to-end workflow to be processed transactionally



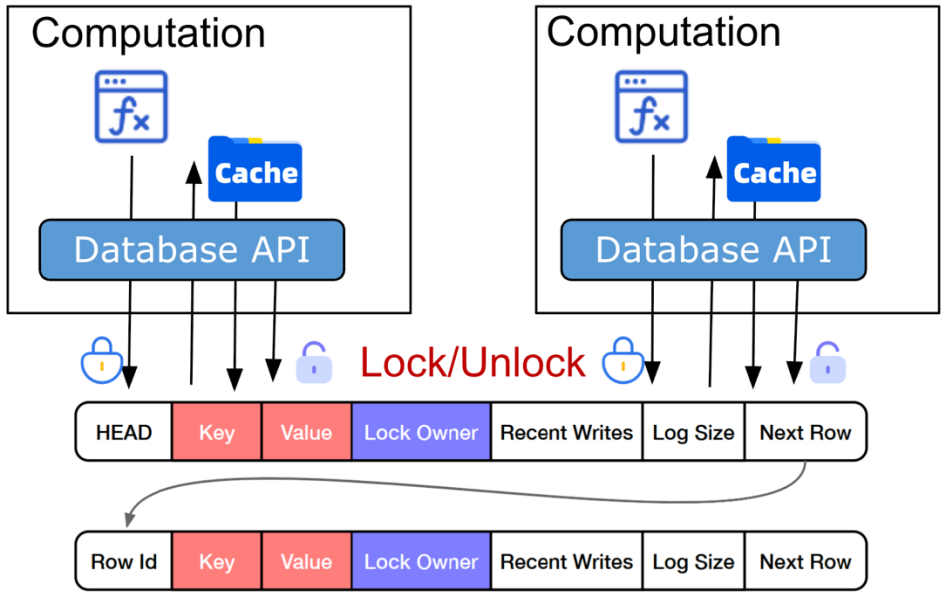
Requirement for State Consistency

- Transactional serializability consistency[SOSP'21, OSDI'20]
  - Applications require the end-to-end workflow to be processed transactionally
  - The effects of concurrent transactions are equivalent to some serial execution

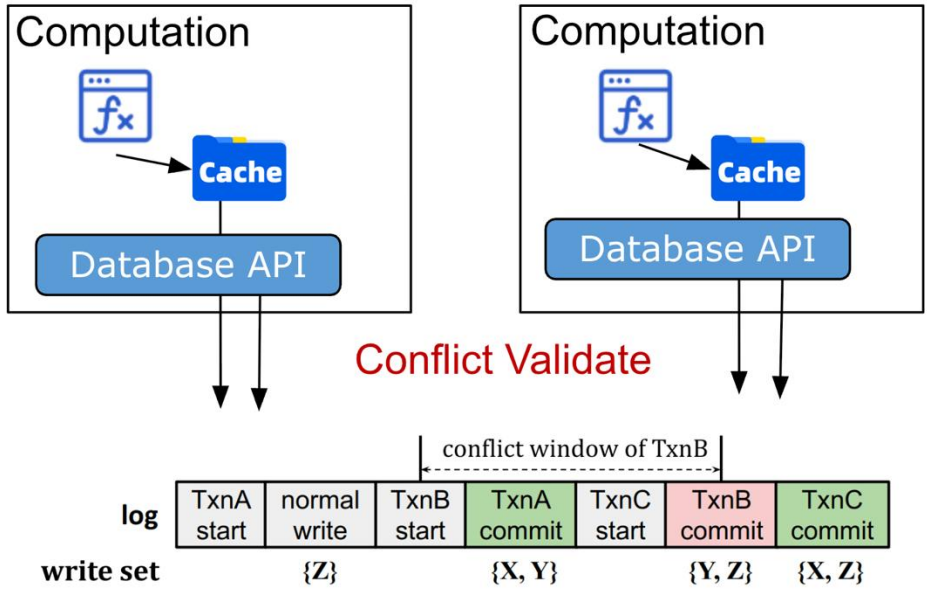


Limitations of Existing Solutions

- A variant of the two-phase locking [OSDI'20, DEBS'21]

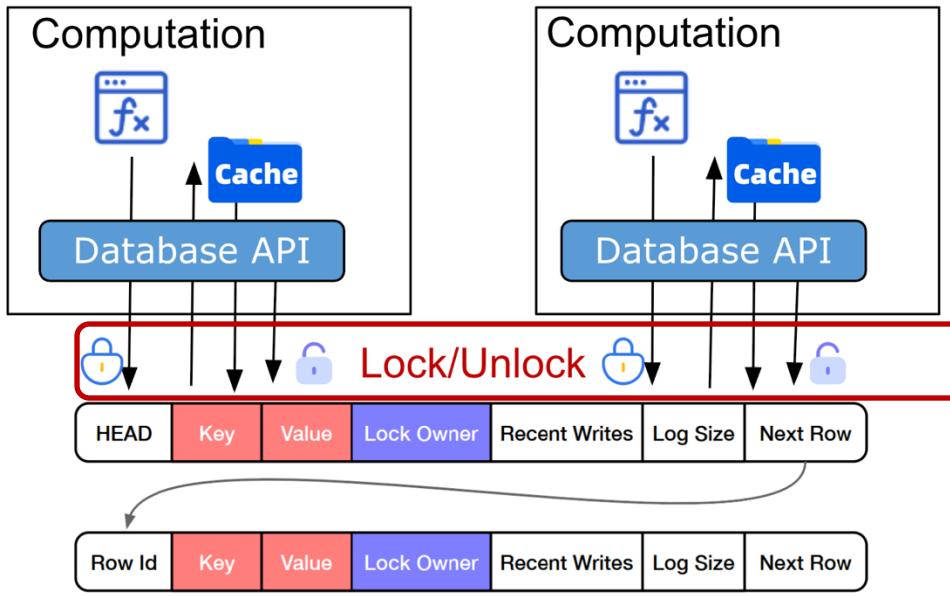


- Log-based optimistic concurrency control[SOSP'21]



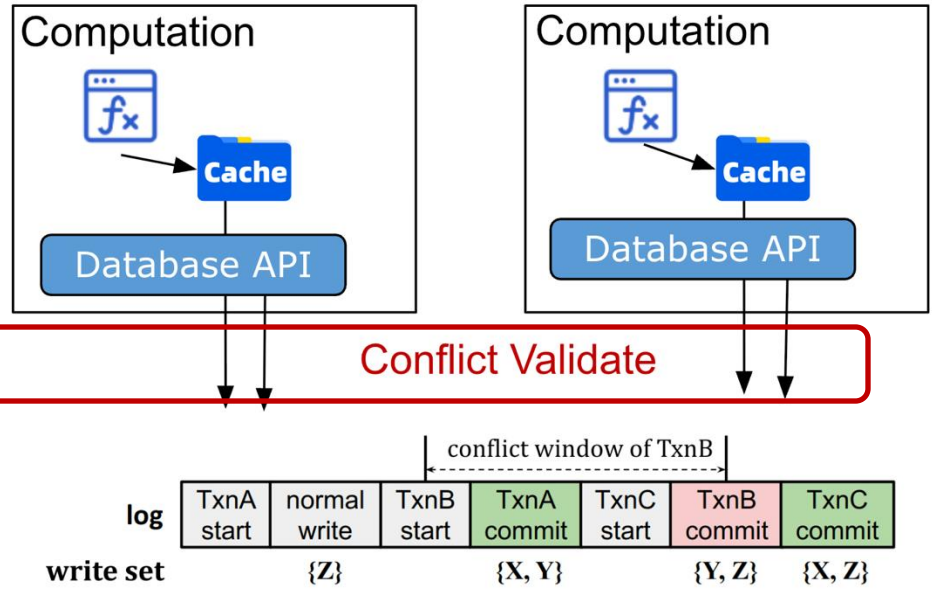
Limitations of Existing Solutions

- A variant of the two-phase locking [OSDI'20, DEBS'21]



1

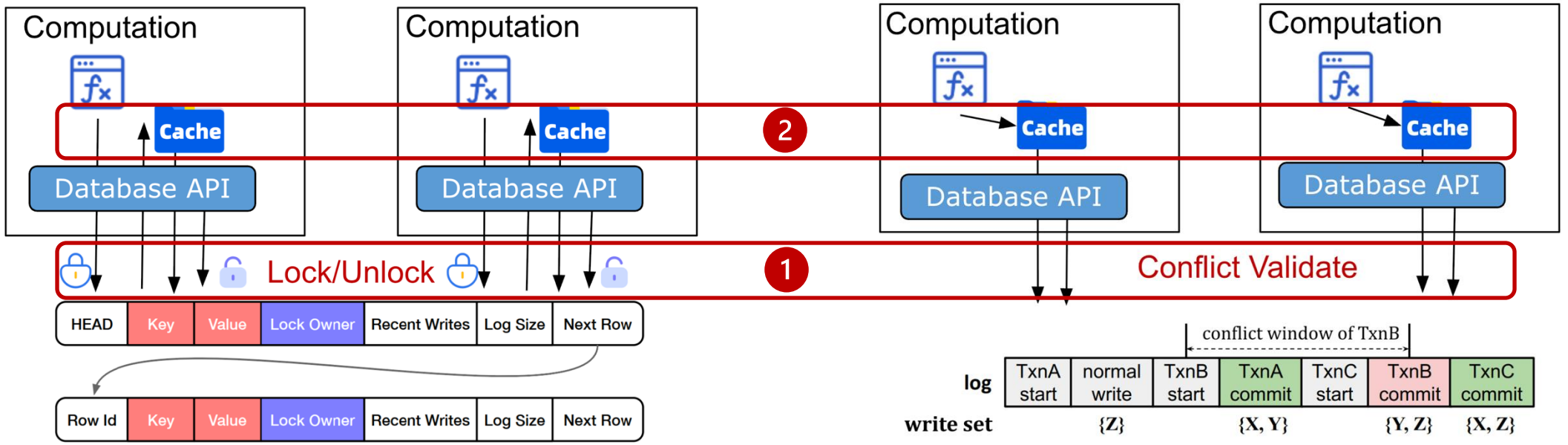
- Log-based optimistic concurrency control[SOSP'21]



1. Usually leads to high communication overhead due to frequent remote accesses to the lock status.

Limitations of Existing Solutions

- A variant of the two-phase lock [OSDI'20, DEBS'21]
- Log-based optimistic concurrency control[SOSP'21]



1. Usually leads to high communication overhead due to frequent remote accesses to the lock status.

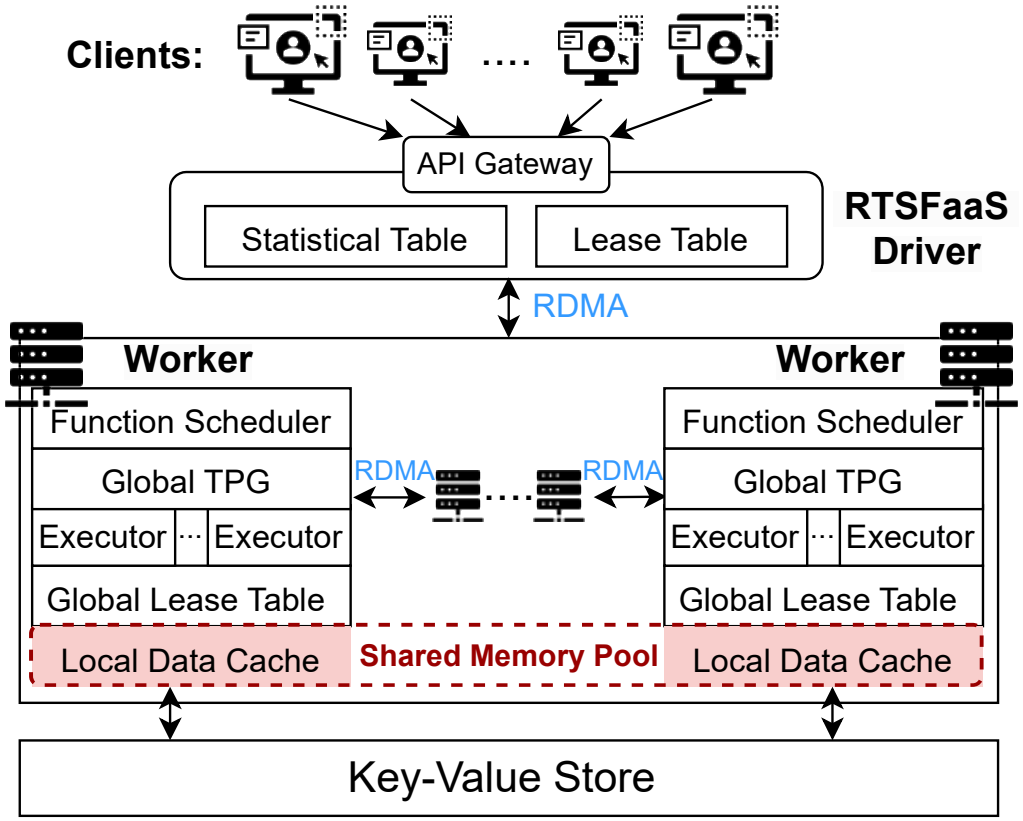
2. Often undermine the efficiency of caching in stateful FaaS platforms.

Outline

- Background and Motivation
- **RTSFaaS Design**
- Evaluation Results
- Conclusion

RTSFaaS Architecture

- **R**DMA-capable **t**ransactional **s**tateful **FaaS** framework



Driver RDMA-capable Data Center

- User requests receiving
- Function workflow routing

Worker

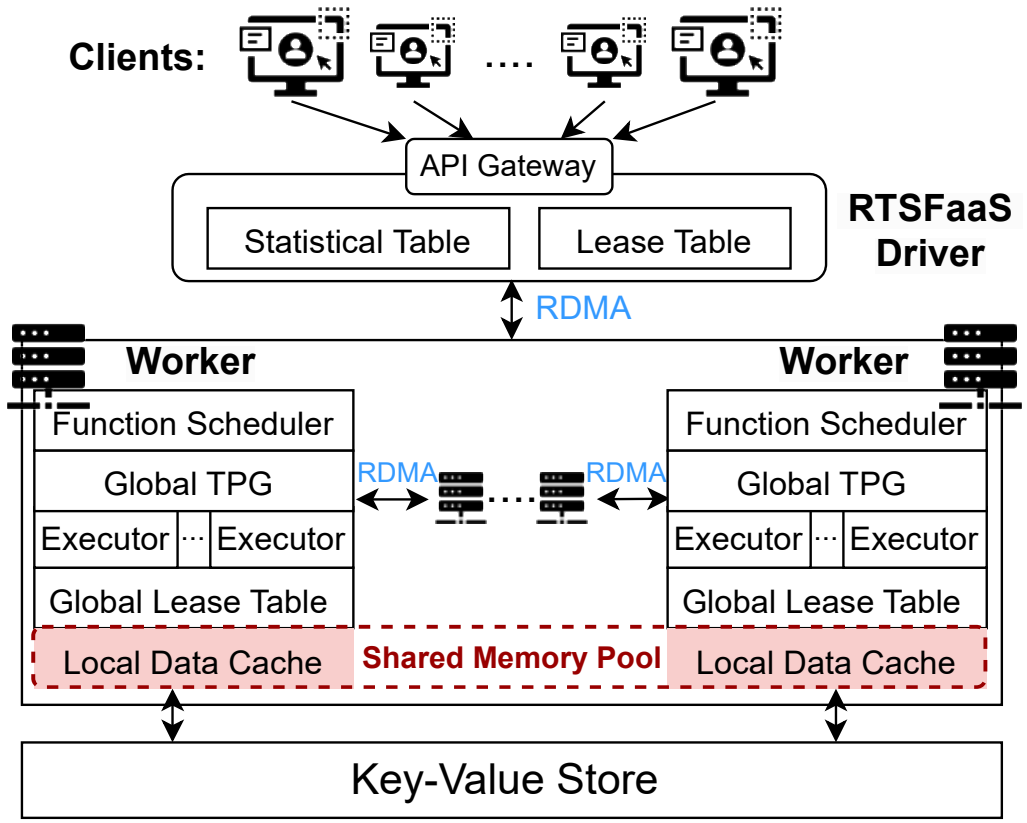
- Function invocation
- Local Cache

Shared Database High availability

- TCP/IP Key-value store (TiKV)
- Geo-distributed replication

RTSFaaS Architecture

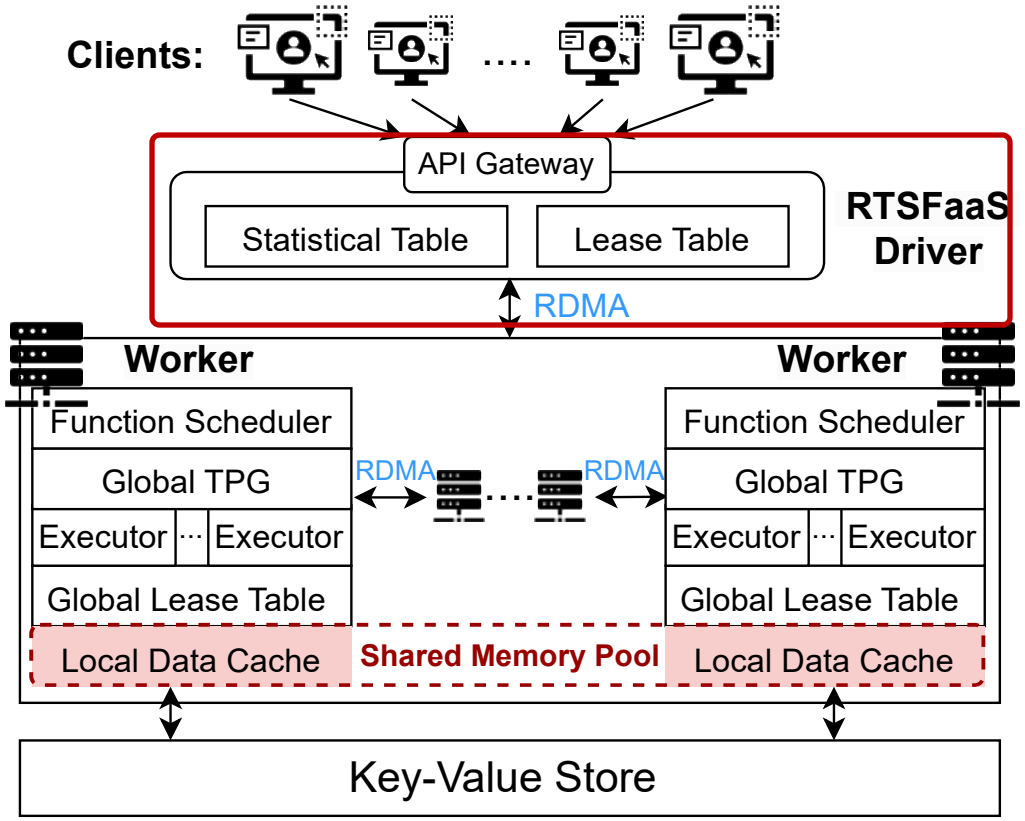
- **R**DMA-capable **t**ransactional **s**tateful **FaaS** framework



Often undermine the efficiency of caching

RTSFaaS Architecture

- **R**DMA-capable **t**ransactional **s**tateful **FaaS** framework

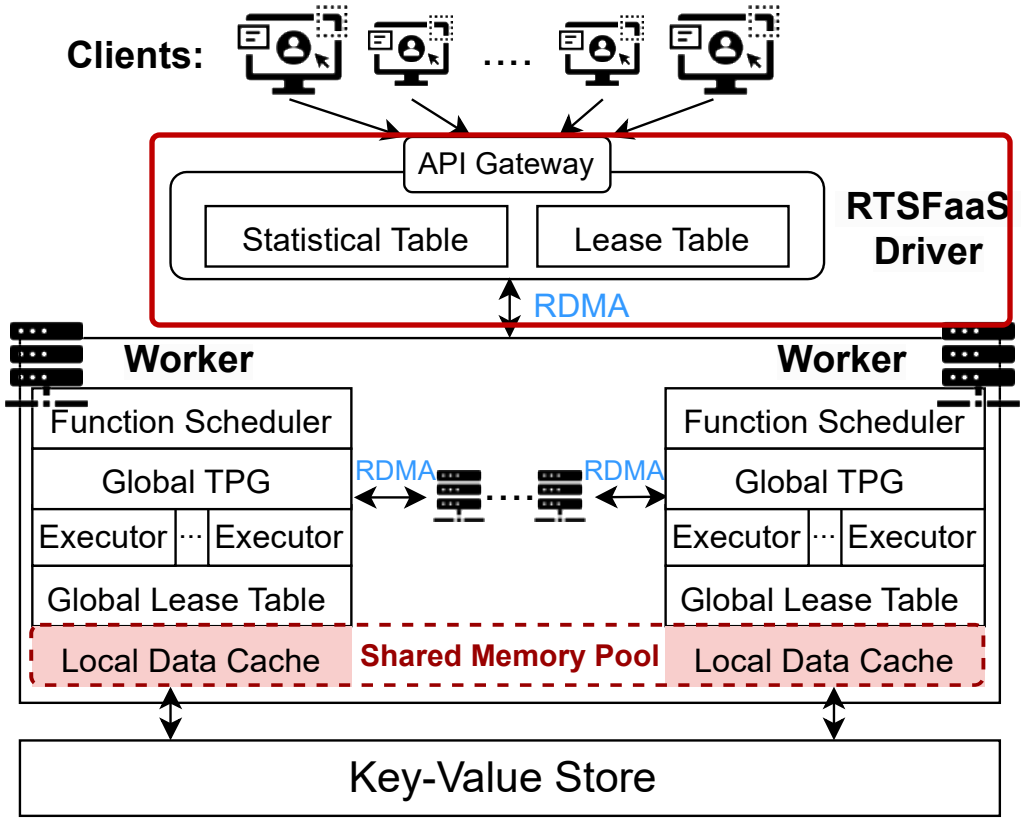


Often undermine the efficiency of caching

Affinity-aware lease assignment

RTSFaaS Architecture

- **R**DMA-capable **t**ransactional **s**tateful **FaaS** framework



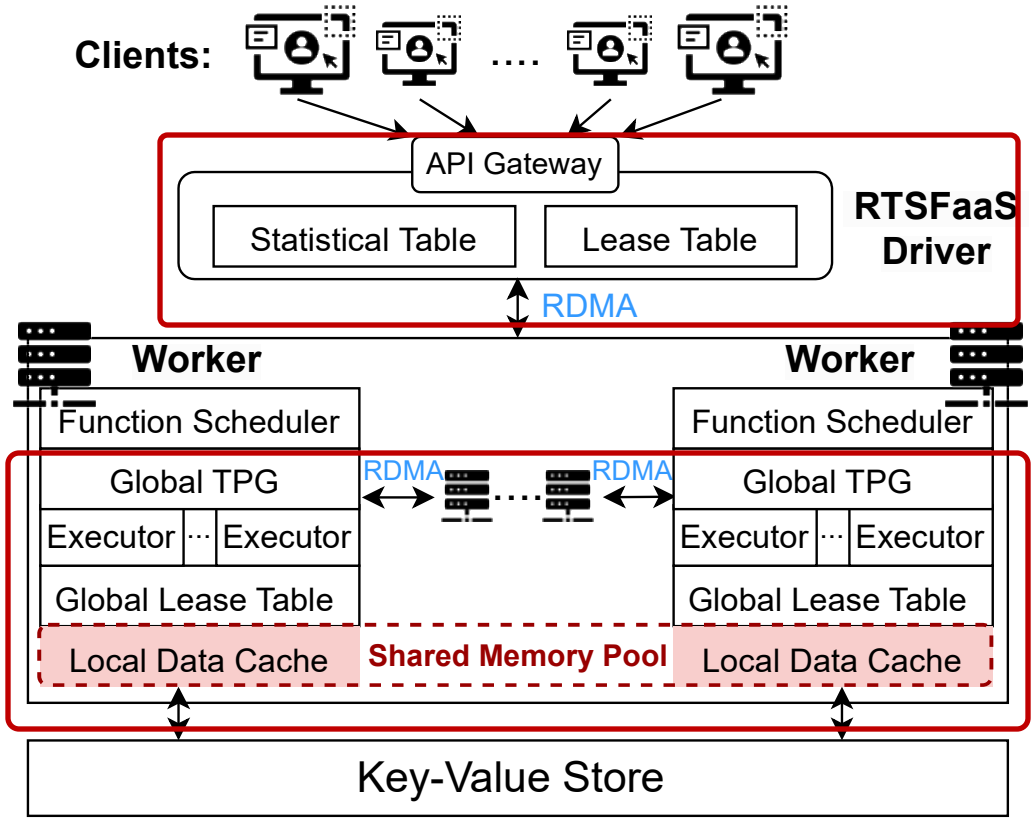
Often undermine the efficiency of caching

Affinity-aware lease assignment

Remote communication is costly

RTSFaaS Architecture

- **R**DMA-capable **t**ransactional **s**tateful **FaaS** framework



Often undermine the efficiency of caching

Affinity-aware lease assignment

Remote communication is costly

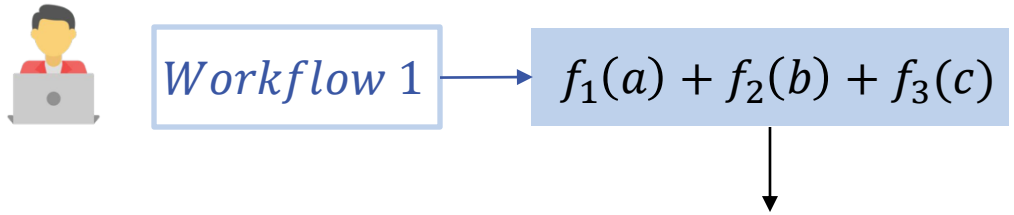
RDMA-capable dynamic lease transferring

Affinity-aware Lease Assignment

- Driver assigns an incoming request to a worker so that they have the strongest data-function affinity.

Affinity-aware Lease Assignment

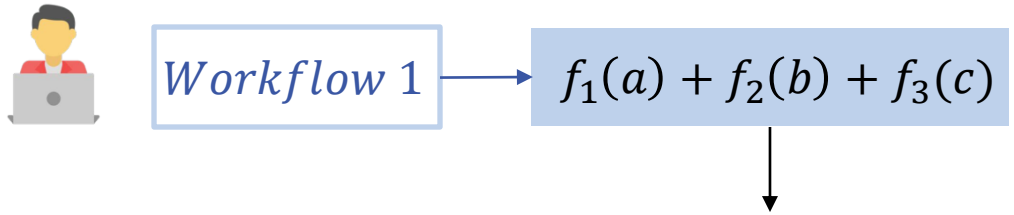
- Driver assigns an incoming request to a worker so that they have the strongest data-function affinity.



Driver: ❶ Identifies its read-write sets

Affinity-aware Lease Assignment

- Driver assigns an incoming request to a worker so that they have the strongest data-function affinity.



Driver:

1. Identifies its read-write sets
2. Calculates affinity score A_i for each worker i

Affinity-aware Lease Assignment

- Driver assigns an incoming request to a worker so that they have the strongest data-function affinity.

$Workflow\ 1$ → $f_1(a) + f_2(b) + f_3(c)$

<a, b, c>

Driver:

1. Identifies its read-write sets
2. Calculates affinity score A_i for each worker i

k_j	A	B	C
$N_1(j)$	2	4	5

Statistics Table

Affinity-aware Lease Assignment

- Driver assigns an incoming request to a worker so that they have the strongest data-function affinity.

$Workflow\ 1$ → $f_1(a) + f_2(b) + f_3(c)$

<a, b, c>

Driver:

1. Identifies its read-write sets
2. Calculates affinity score A_i for each worker i

$A_1 = 2 + 4 + 5 = 11$

k_j	A	B	C
$N_1(j)$	2	4	5

Statistics Table

Affinity-aware Lease Assignment

- Driver assigns an incoming request to a worker so that they have the strongest data-function affinity.

$Workflow\ 1 \rightarrow f_1(a) + f_2(b) + f_3(c)$

<a, b, c>

Driver:

1. Identifies its read-write sets
2. Calculates affinity score A_i for each worker i — $A_1 = 11$ $A_2 = 3$ $A_3 = 3$
3. Calculates load balance N_i for each worker i — $N_1 = 20$ $N_2 = 26$ $N_3 = 29$

Affinity-aware Lease Assignment

- Driver assigns an incoming request to a worker so that they have the strongest data-function affinity.

$Workflow\ 1 \rightarrow f_1(a) + f_2(b) + f_3(c)$

<a, b, c>

Driver:

1. Identifies its read-write sets
2. Calculates affinity score A_i for each worker i — $A_1 = 11$ $A_2 = 3$ $A_3 = 3$
3. Calculates load balance N_i for each worker i — $N_1 = 20$ $N_2 = 26$ $N_3 = 29$
4. Normalizes the two values and combine them to compute the final score — $\mathrm{F}_1 = 2$ $\mathrm{F}_2 = 0.3$ $N_3 = 0$

Affinity-aware Lease Assignment

- Driver assigns an incoming request to a worker so that they have the strongest data-function affinity.

Data-function affinity and Load balancing

$Workflow\ 1 \rightarrow f_1(a) + f_2(b) + f_3(c)$

<a, b, c>

Driver:

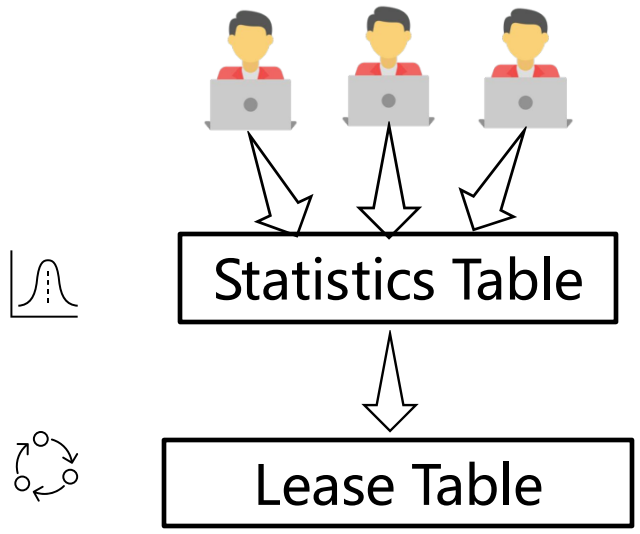
1. Identifies its read-write sets
2. Calculates affinity score A_i for each worker i — $A_1 = 11$ $A_2 = 3$ $A_3 = 3$
3. Calculates load balance N_i for each worker i — $N_1 = 20$ $N_2 = 26$ $N_3 = 29$
4. Normalizes the two values and combine them to compute the final score — $\mathrm{F}_1 = 2$ $\mathrm{F}_2 = 0.3$ $N_3 = 0$
5. Forwards the request to the worker with the highest score and update the statistics table

k_j	A	B	C
$N_1(j)$	2+1	4+1	5+1

Statistics Table

Affinity-aware Lease Assignment

- Driver assigns an incoming request to a worker so that they have the strongest data-function affinity.
- If a worker has the highest access frequency to an object, it becomes the leaseholder of the object.



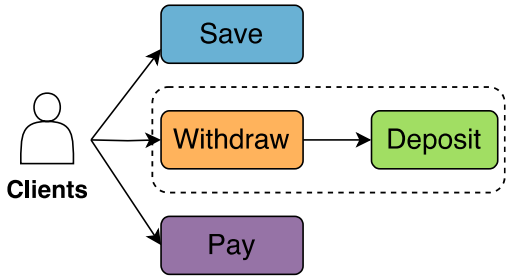
Affinity-aware Lease Assignment

Each KV object is exclusively cached by only leaseholder

Allow more function to execute with the local cache

RDMA-capable dynamic lease transferring

- Dependency Definition



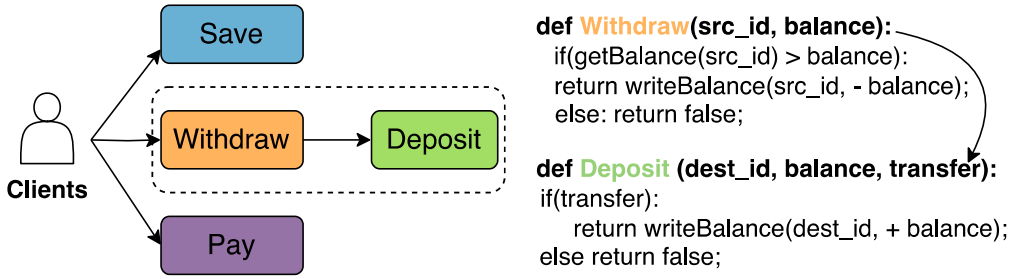

```
def Withdraw(src_id, balance):
  if(getBalance(src_id) > balance):
  return writeBalance(src_id, - balance);
  else: return false;

def Deposit (dest_id, balance, transfer):
 if(transfer):
     return writeBalance(dest_id, + balance);
else return false;
```

An example of the banking service workflow

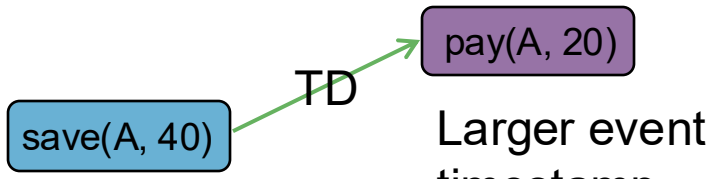
RDMA-capable dynamic lease transferring

- Dependency Definition
  - Temporal Dependency (TD)



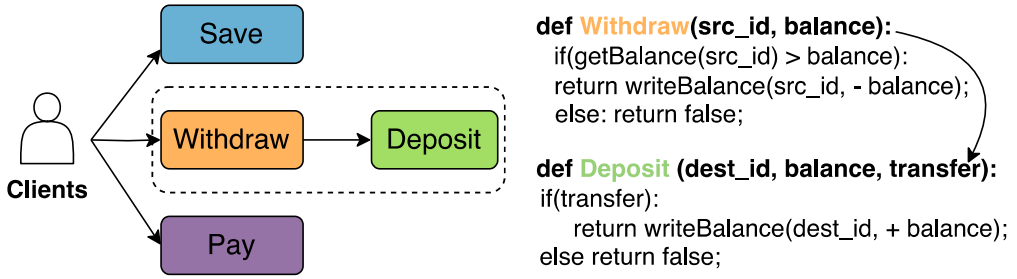
An example of the banking service workflow

Functions access the same state need to executed following the event timestamp sequence



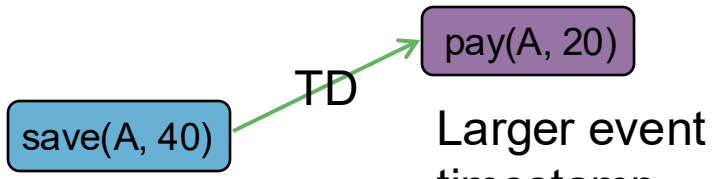
RDMA-capable dynamic lease transferring

- Dependency Definition
  - Temporal Dependency (TD)
  - Parametric Dependency (PD)

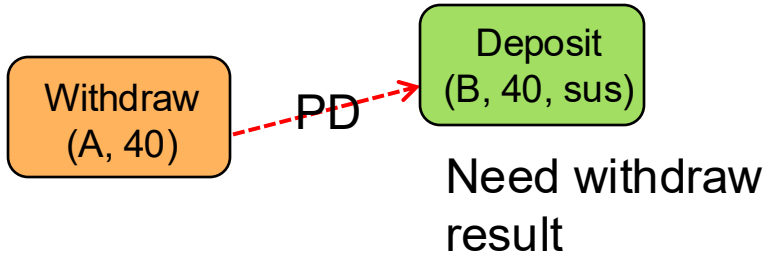


An example of the banking service workflow

Functions access the same state need to executed following the event timestamp sequence



A function parametrically depends on another when it consumes the output of a preceding function within the same workflow.

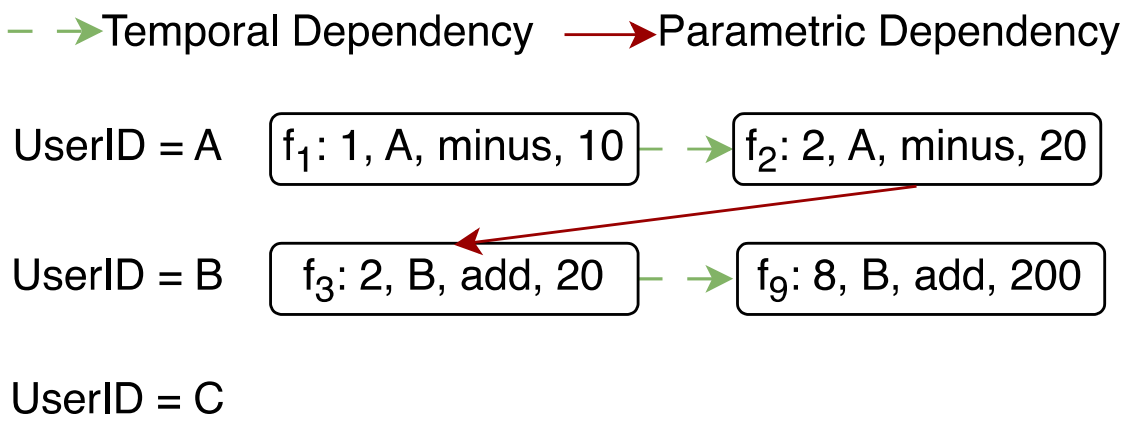


RDMA-capable dynamic lease transferring

➢ Task Precedence Graph(TPG) Construction - Local TPG Construction

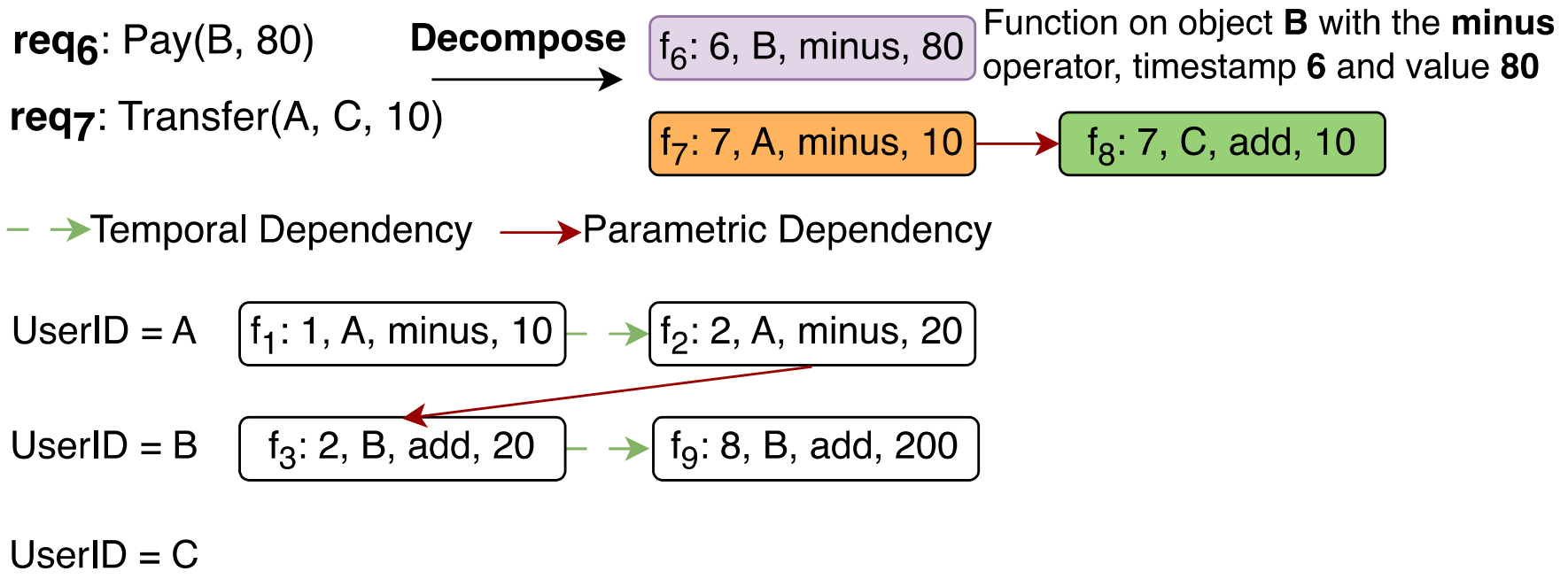
req_6: Pay(B, 80)

req_7: Transfer(A, C, 10)



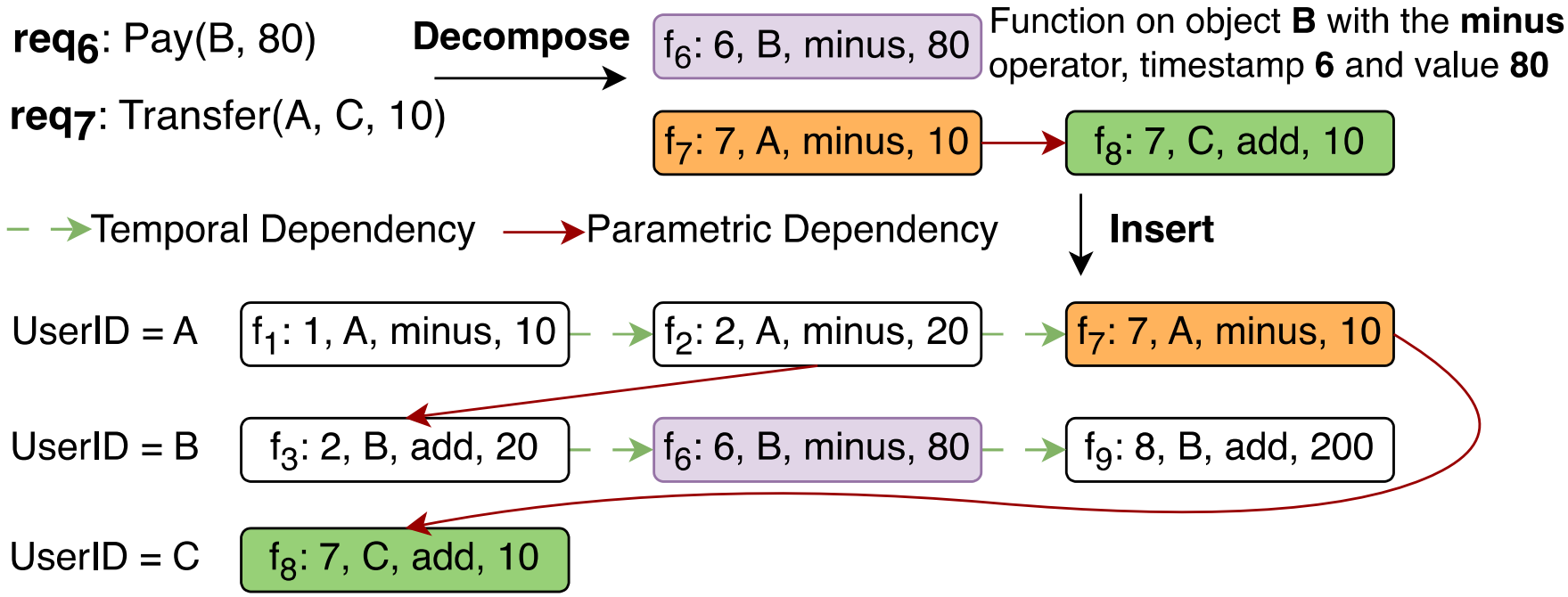
RDMA-capable dynamic lease transferring

➢ Task Precedence Graph(TPG) Construction - Local TPG Construction



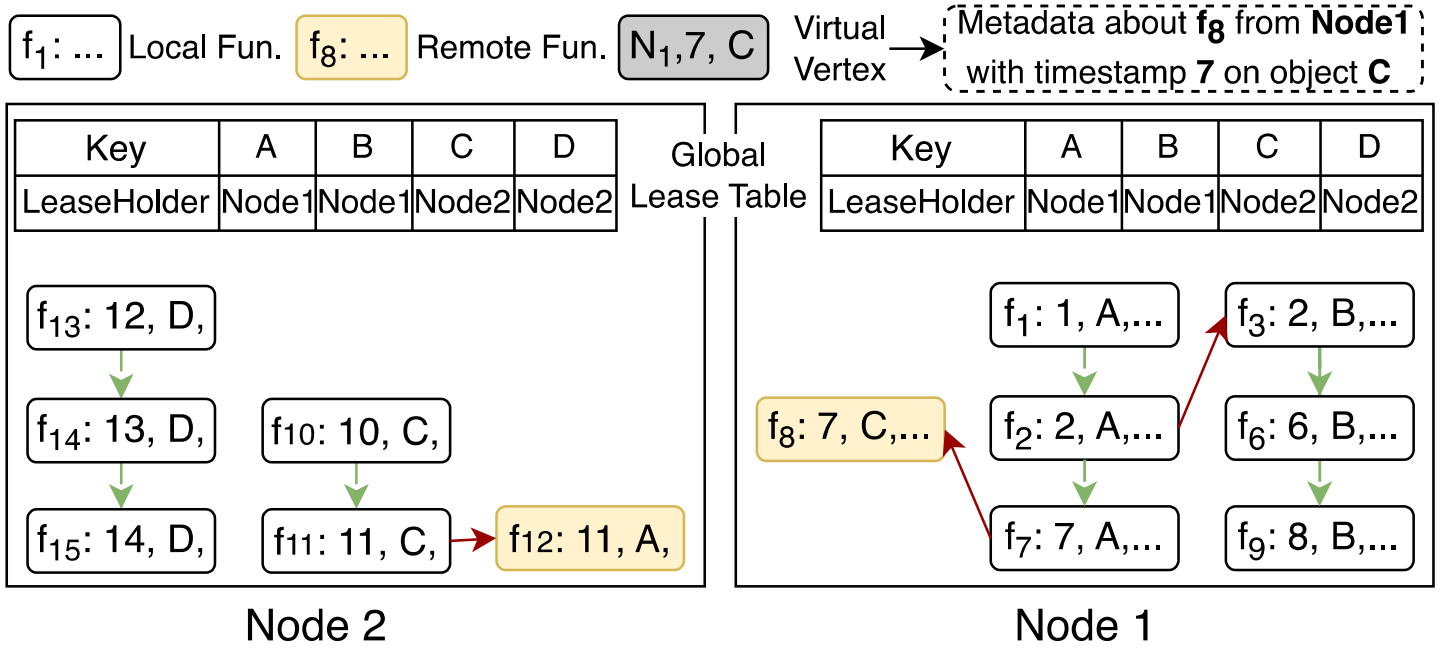
RDMA-capable dynamic lease transferring

➢ Task Precedence Graph(TPG) Construction - Local TPG Construction



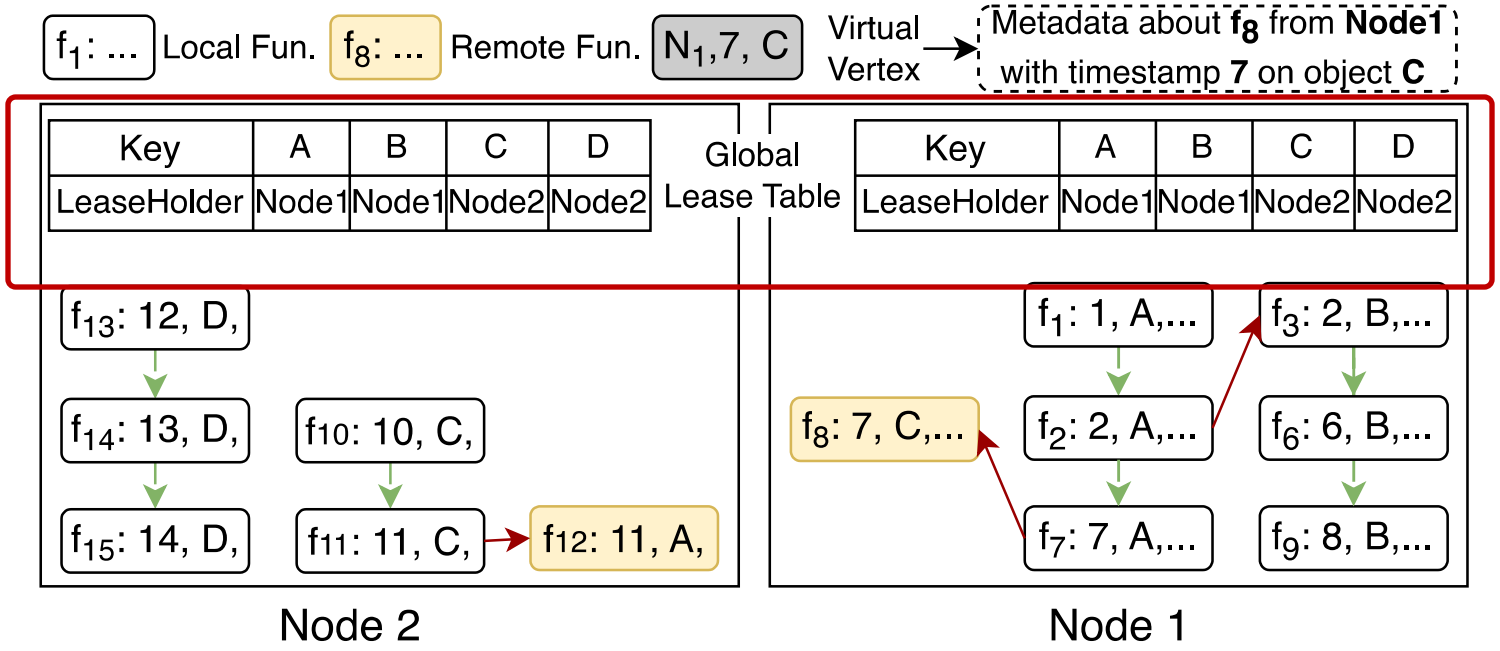
RDMA-capable dynamic lease transferring

- Task Precedence Graph(TPG) Construction - Global TPG Construction



RDMA-capable dynamic lease transferring

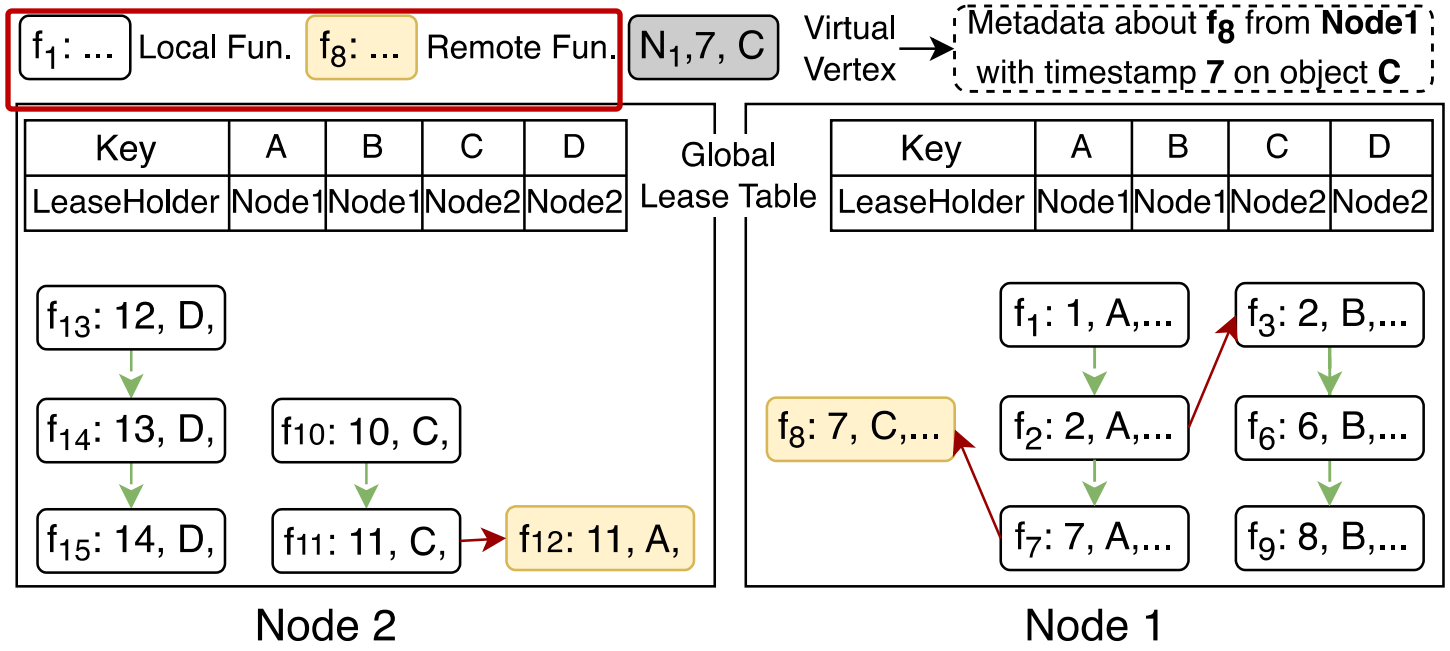
- Task Precedence Graph(TPG) Construction - Global TPG Construction



Exclusively cached by only leaseholder

RDMA-capable dynamic lease transferring

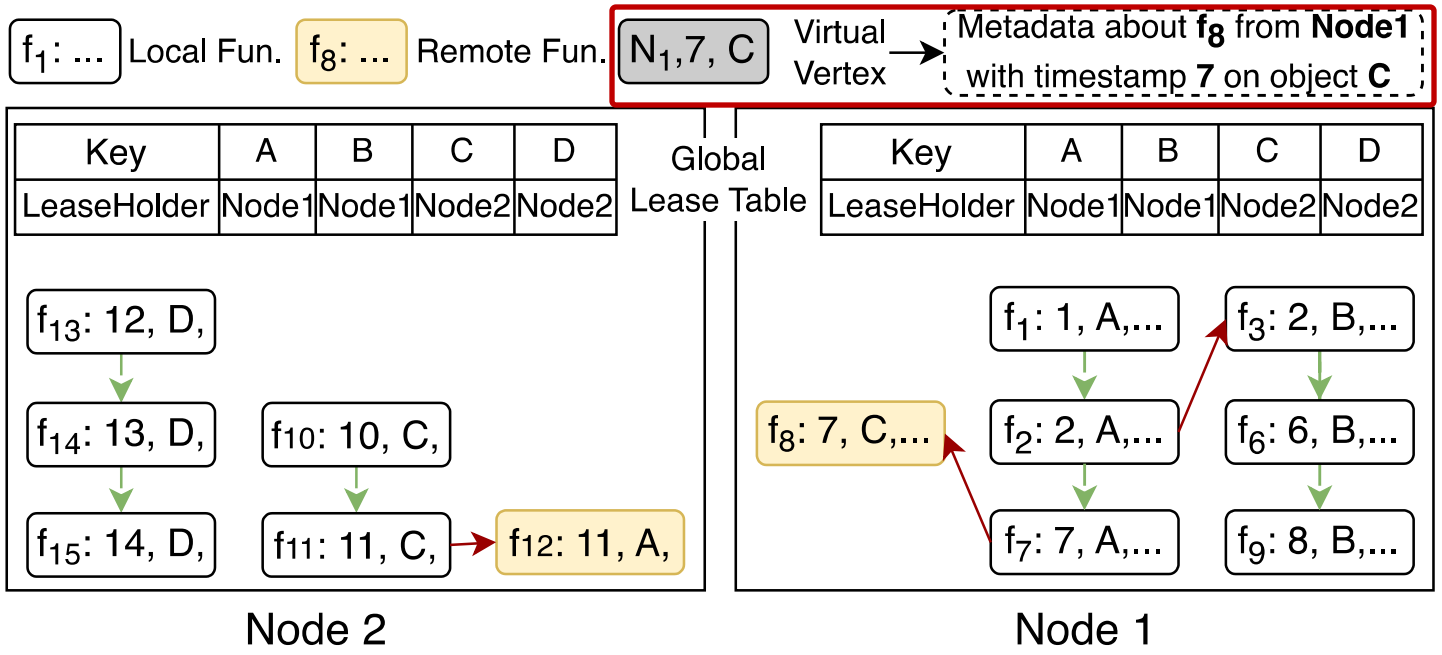
- Task Precedence Graph(TPG) Construction - Global TPG Construction



Classify local function and remote function

RDMA-capable dynamic lease transferring

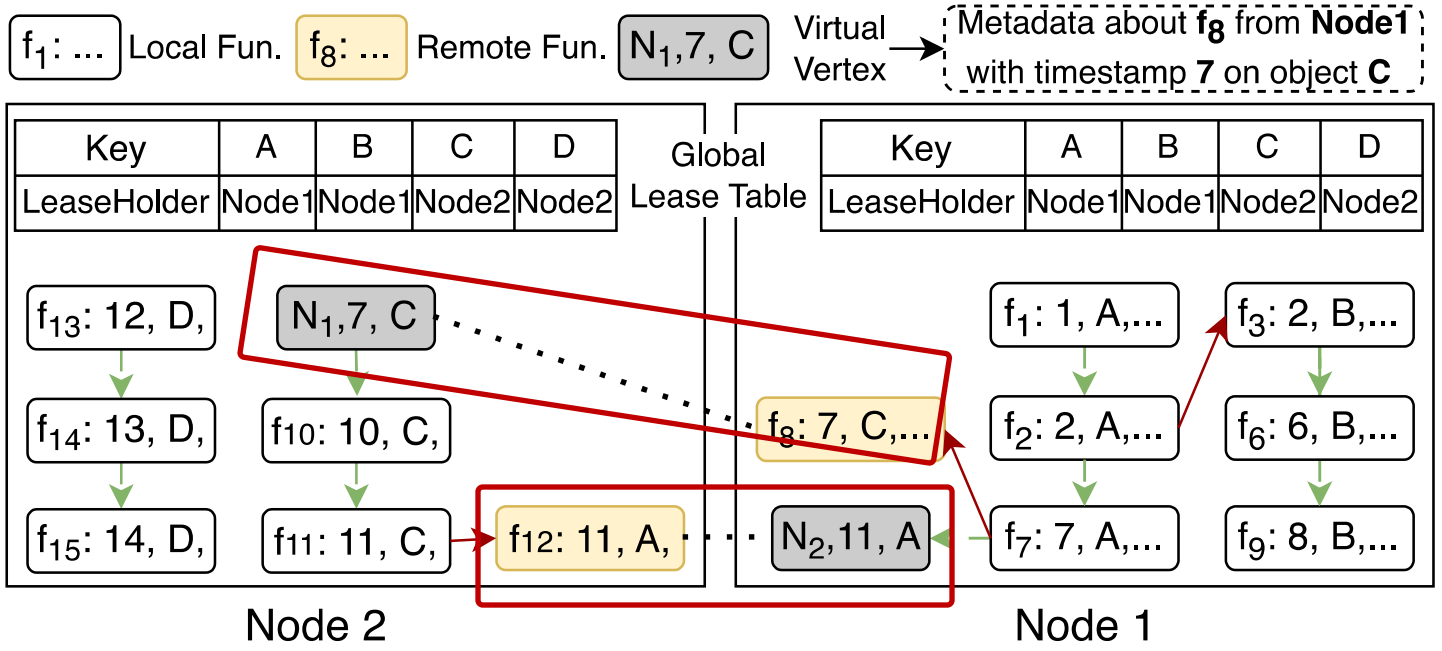
- Task Precedence Graph(TPG) Construction - Global TPG Construction



Virtual node to include the metadata about remote function

RDMA-capable dynamic lease transferring

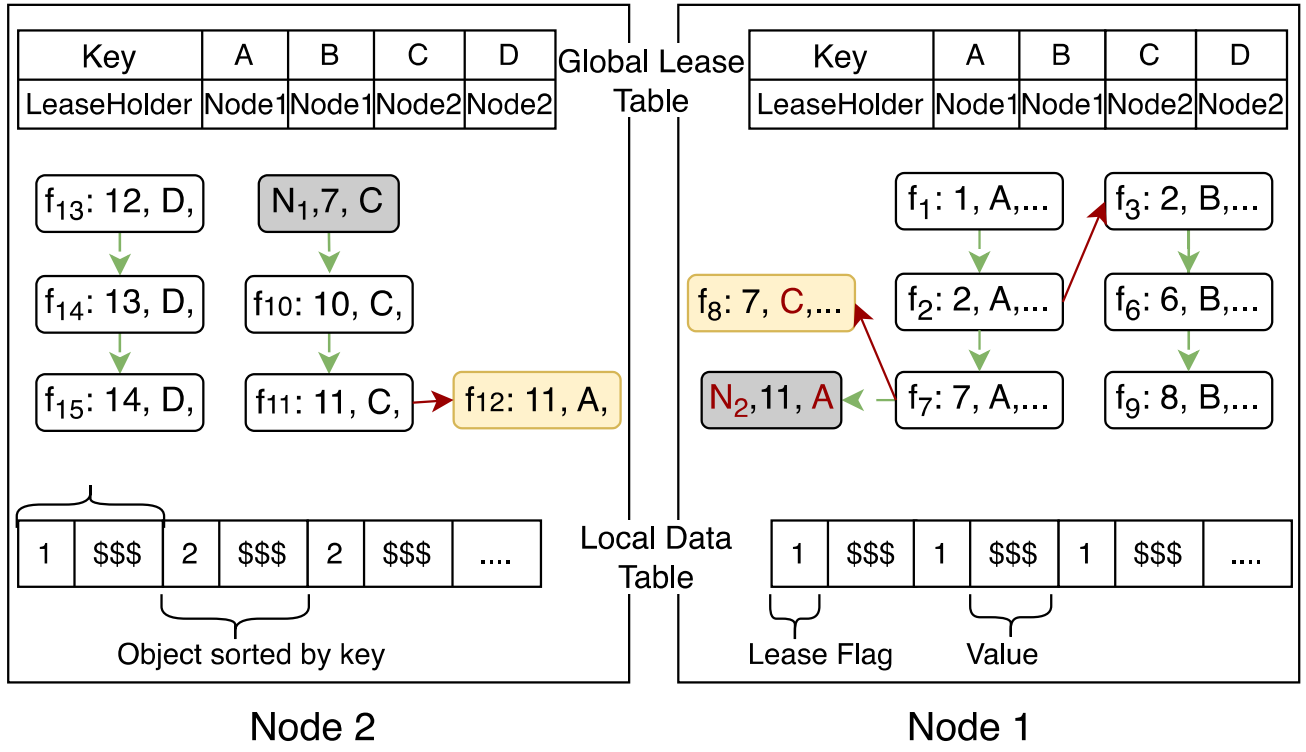
- Task Precedence Graph(TPG) Construction - Global TPG Construction



Remote functions and their metadata

RDMA-capable dynamic lease transferring

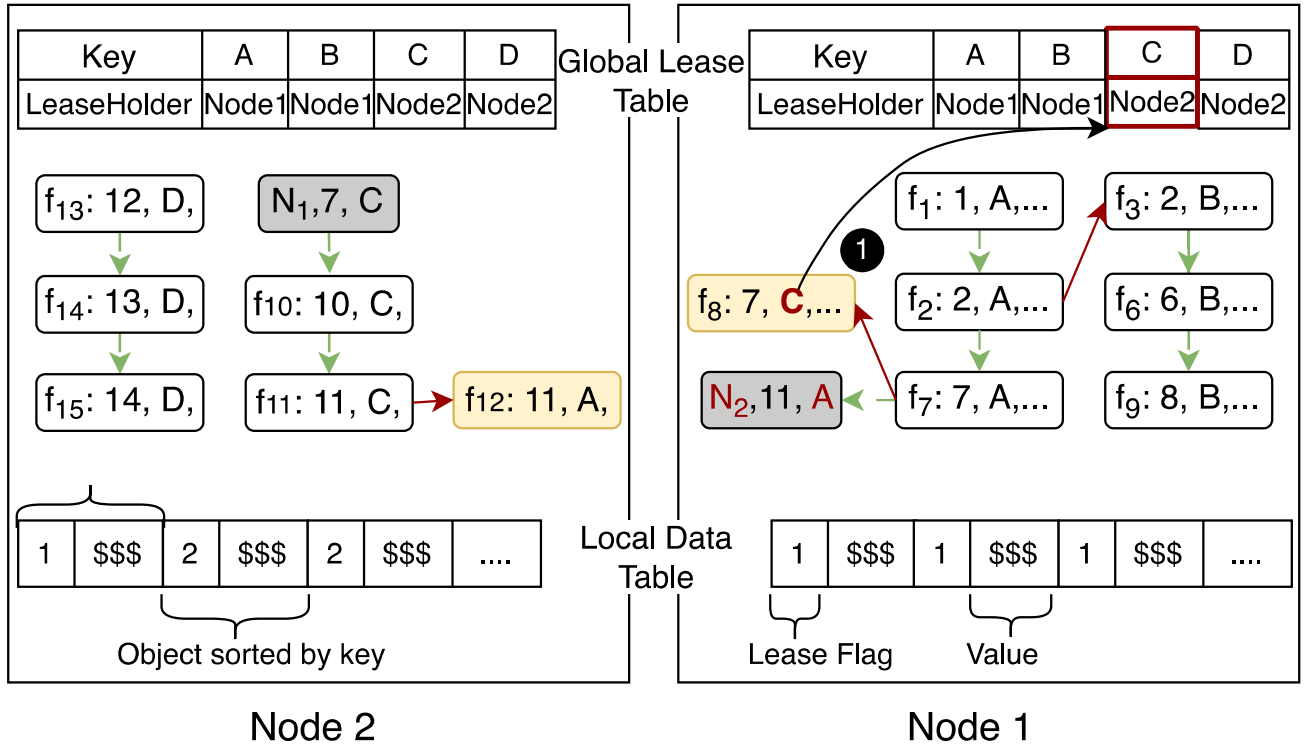
- Workers schedule functions based on the global TPG



Scheduling functions based on the global TPG

RDMA-capable dynamic lease transferring

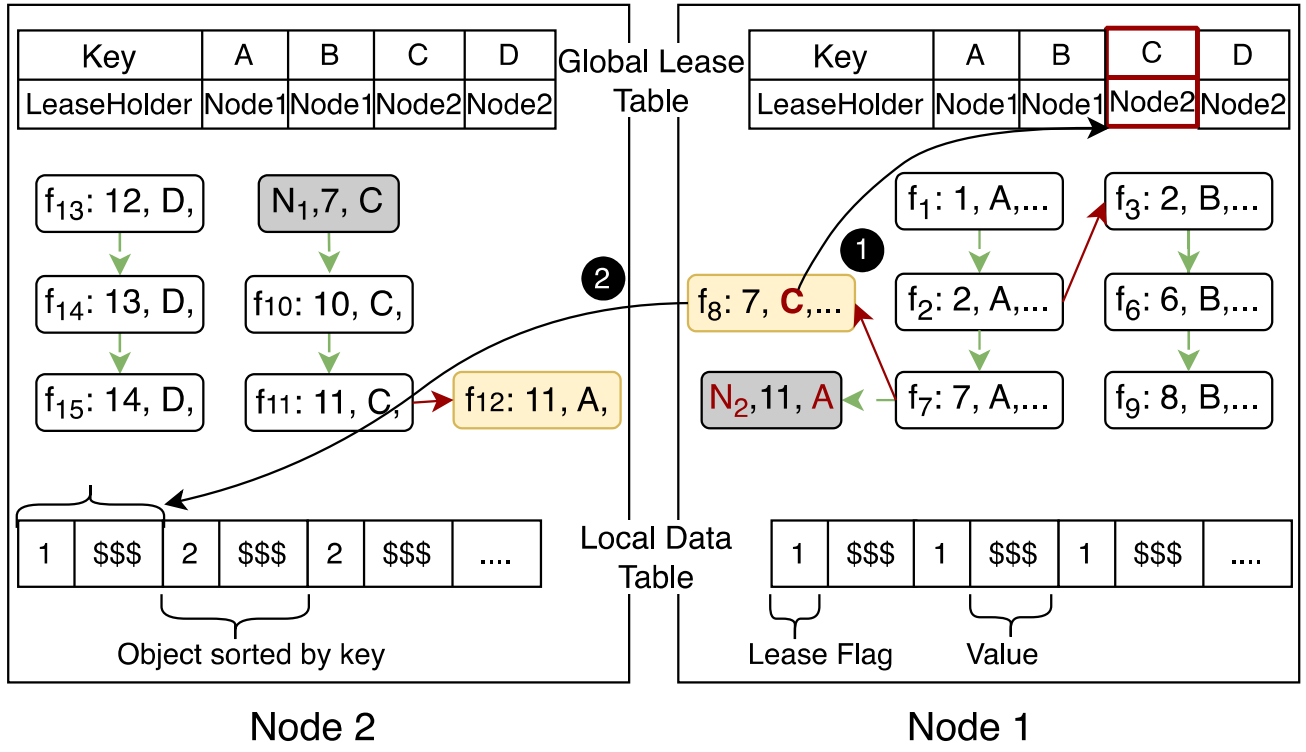
- Access remote shared caches via efficiently leasehold check



Locate where is the object using the global lease table

RDMA-capable dynamic lease transferring

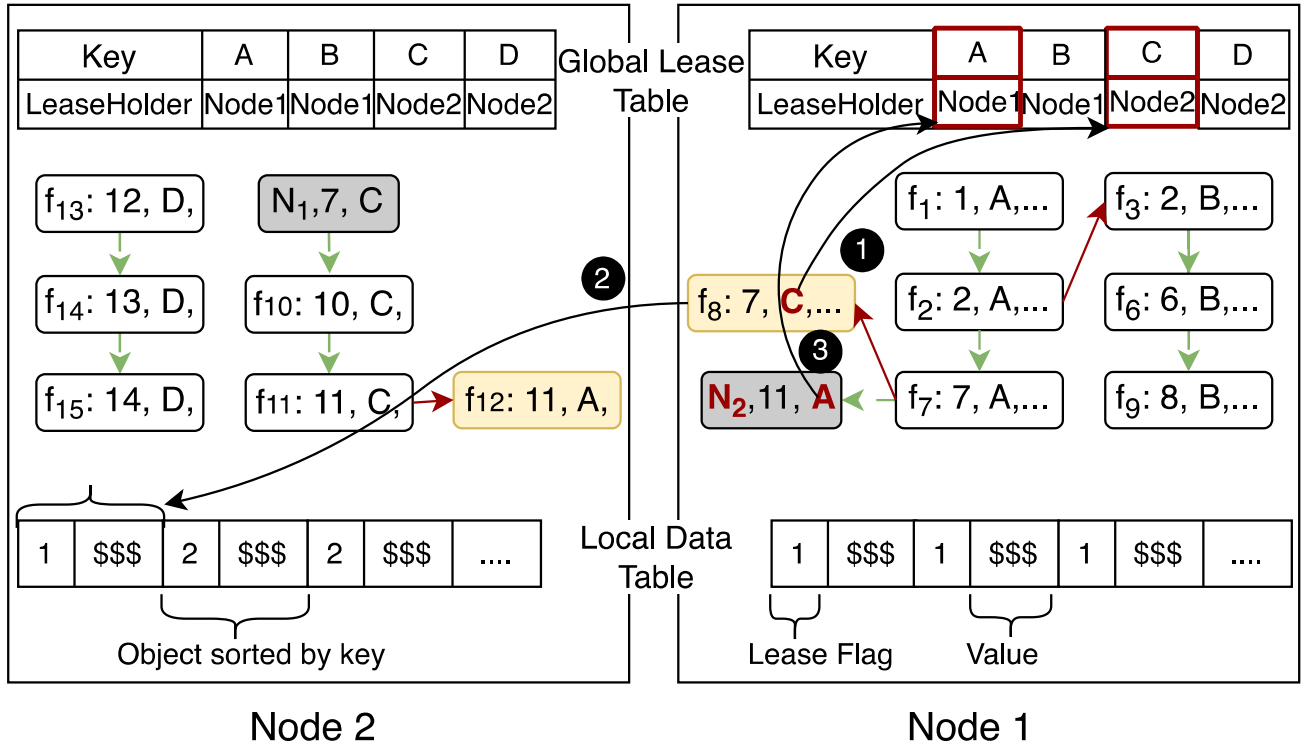
- Access remote shared caches via efficiently leasehold check



Fetch and check the lease flag of the remote object

RDMA-capable dynamic lease transferring

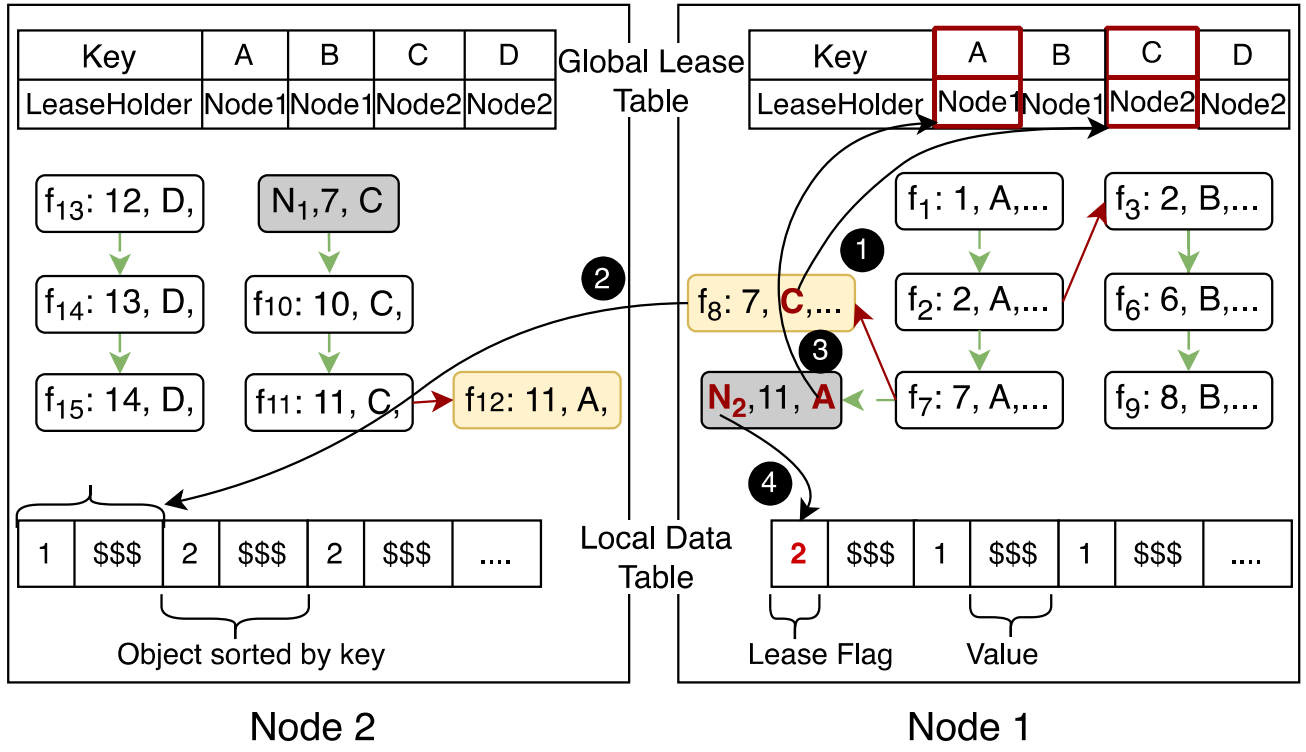
- Ensure sequential data accesses via dynamic lease transferring



Scheduling the virtual node to transfer the object lease

RDMA-capable dynamic lease transferring

- Ensure sequential data accesses via dynamic lease transferring



Updating the lease flag to allow other nodes to access the local cache

Outline

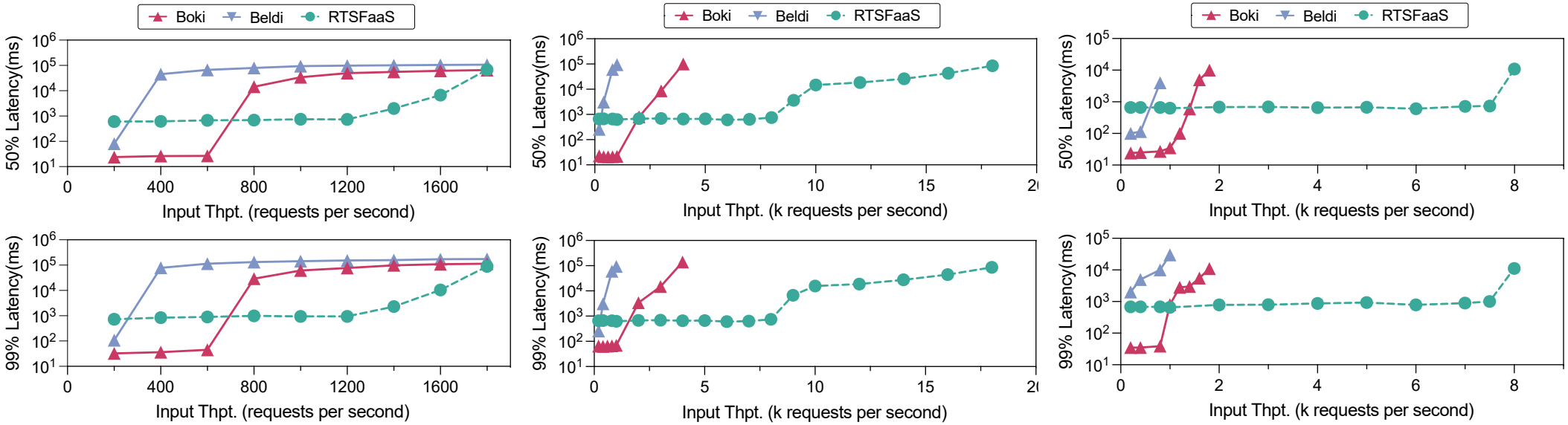
- Background and Motivation
- SmartANNS Design
- **Results**
- Conclusion

Experimental Setup

- Hardware Platform
  - A cluster with five physical machines
  - Interconnected via Mellanox ConnectX-3 40/56 GbE network control
  - Using TiKV for database management
- Comparisons:
  - Transactional FaaS Systems
    - Beldi[OSDI'20]
    - Boki[SOSP'21]
  - RDMA-enable CC Mechanisms
    - Remote Lock
    - Remote OCC + Local Cache
- Benchmarks[ASPLOS'19,OSDI'20,SOSP'21]
  - Movie Review
  - Travel Reservation
  - Banking Service

Comparing to Transactional FaaS Systems

❖ Under different dataset

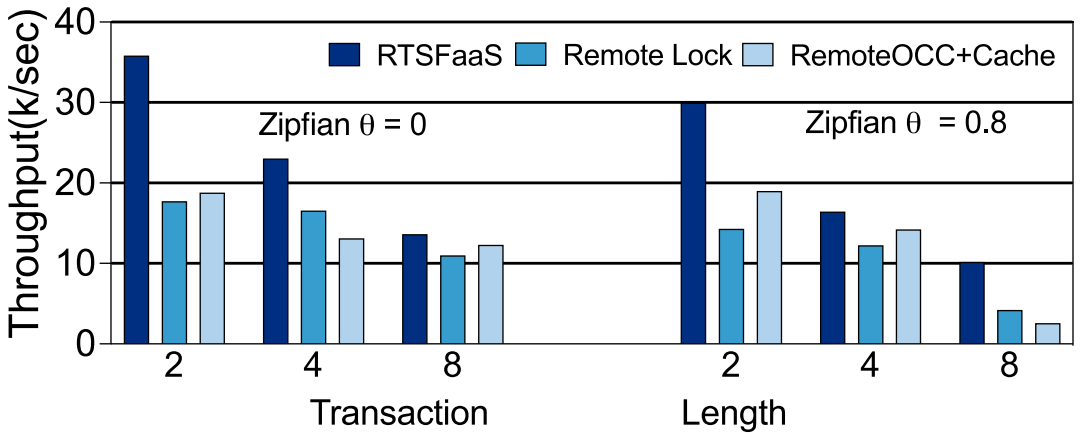


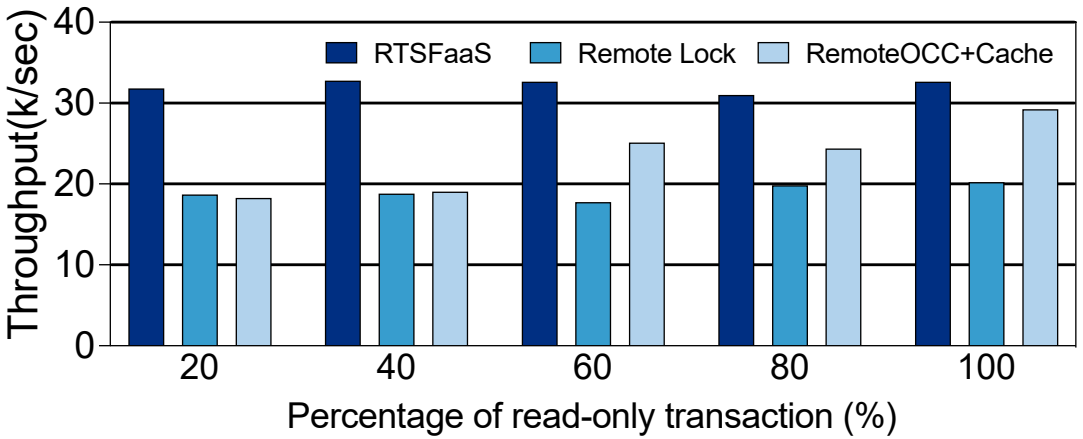
(a) Movie Reviews (b) Travel Reservations (c) Banking Services

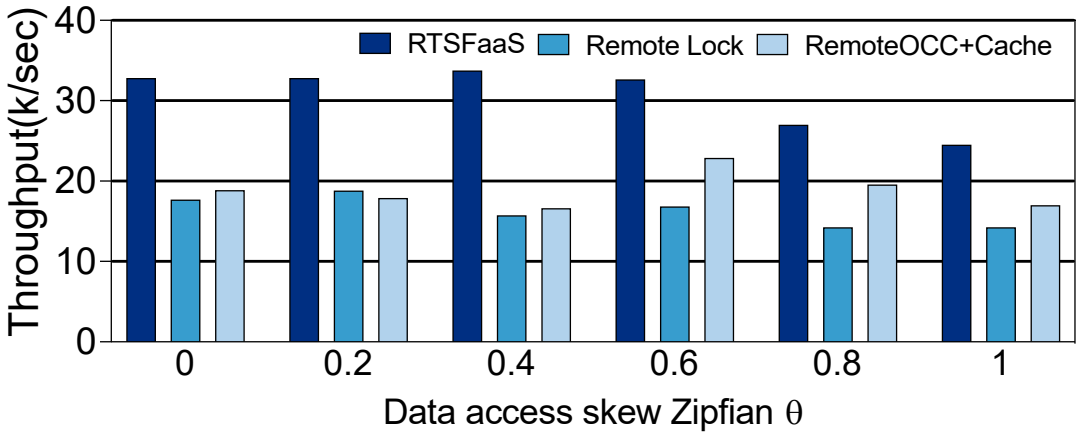
RTSFaaS demonstrates superior performance compared to Boki and Beldi by maintaining consistently lower latency across increasing input throughput.

Comparing to RDMA-enable CC Mechanisms

❖ Under different workload characteristics



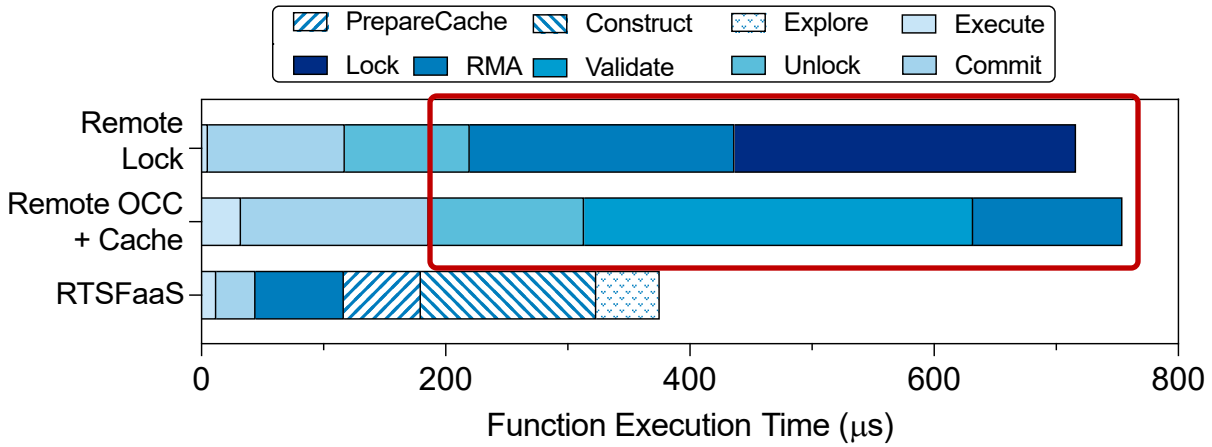




Skew-tolerant (Zipfian)
Read-heavy and long transactions handled
Less aborts, higher locality

Performance Influencing Factors

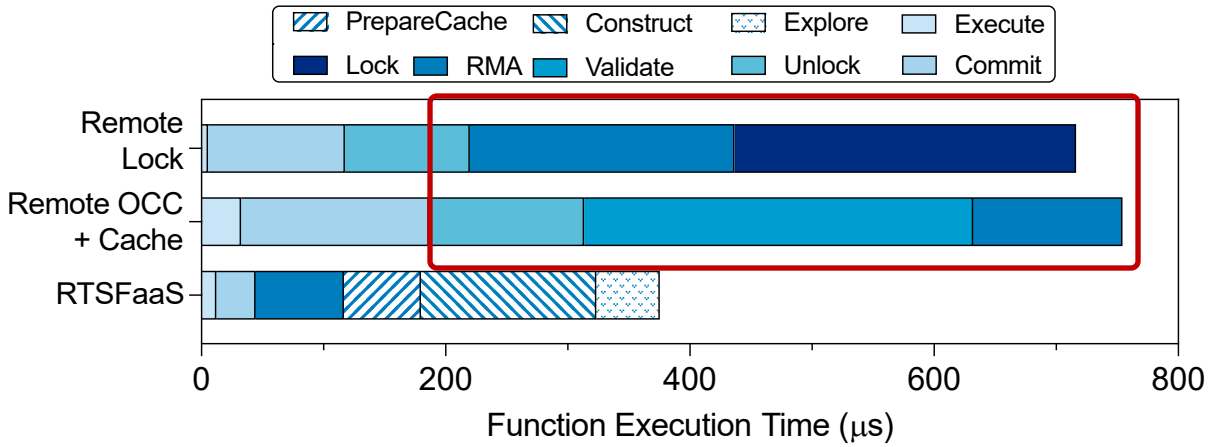
❖ Factor Analysis



This result highlights that merely adding RDMA on top of existing protocols designed for transactional consistency does not effectively alleviate the overhead of remote communication.

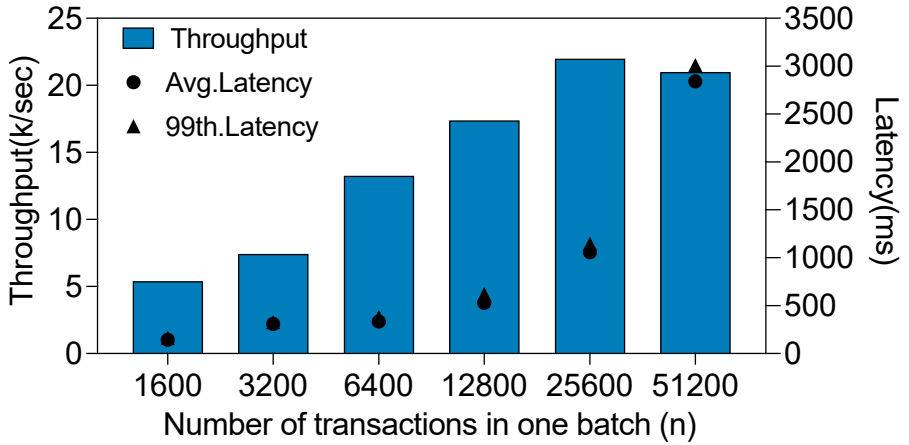
Performance Influencing Factors

❖ Factor Analysis



This result highlights that merely adding RDMA on top of existing protocols designed for transactional consistency does not effectively alleviate the overhead of remote communication.

❖ Impact of Batch Size



The results indicate that throughput initially increases with the batch size, reaching a plateau when the computational resources on each machine are fully utilized.

Outline

- Background and Motivation
- SmartANNS Design
- Results
- **Conclusion**

Conclusion

- **The Problem:** Traditional FaaS platforms failed to support transactional stateful workflows due to high communication overhead when ensuring strong consistency across shared application state.

- **The Solution:** We propose **RTSFaaS**, an RDMA-capable FaaS framework that ensures transactional consistency via a lease-based concurrency control protocol, featuring affinity-aware lease assignment and RDMA-capable lease transfer.

- **The Results:** RTSFaaS achieves up to **5× and 20× speedup** over state-of-the-art systems (Boki and Beldi), and up to **1.7× and 2.1× improvement** when integrating RDMA-based concurrency protocols.

Thanks & QA

Contact Information: curry_zhao@hust.edu.cn