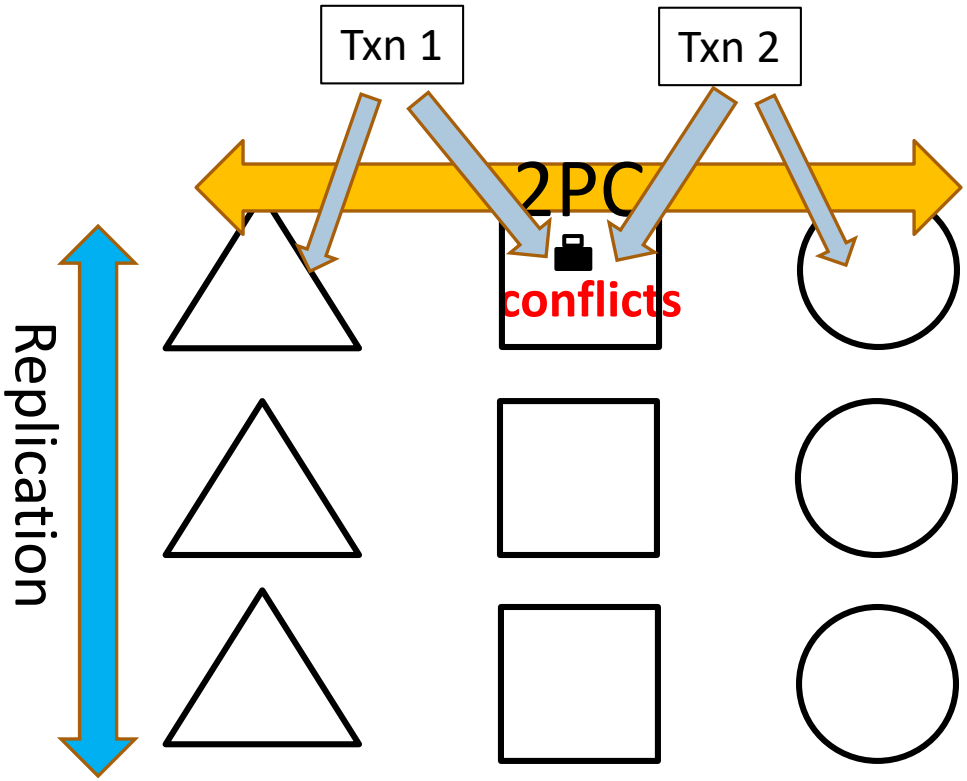
Ocean Vista: Gossip-Based Visibility Control for Speedy Geo-Distributed Transactions

Hua Fan (Alibaba Group)
guanming.fh@alibaba-inc.com

Wojciech Golab (University of Waterloo)

Transactions are hard. **Distributed** transactions are harder. Distributed transactions **over the WAN** are **final boss** hardness. *



Concurrency Control (CC) is used to provide strong consistency.

For serializable execution, CC needs at least 1RTT exclusively accessing conflicting data. (TAPIR: MV+OCC)

Geo-distributed: at least 1 WAN RTT (up to hundreds ms).

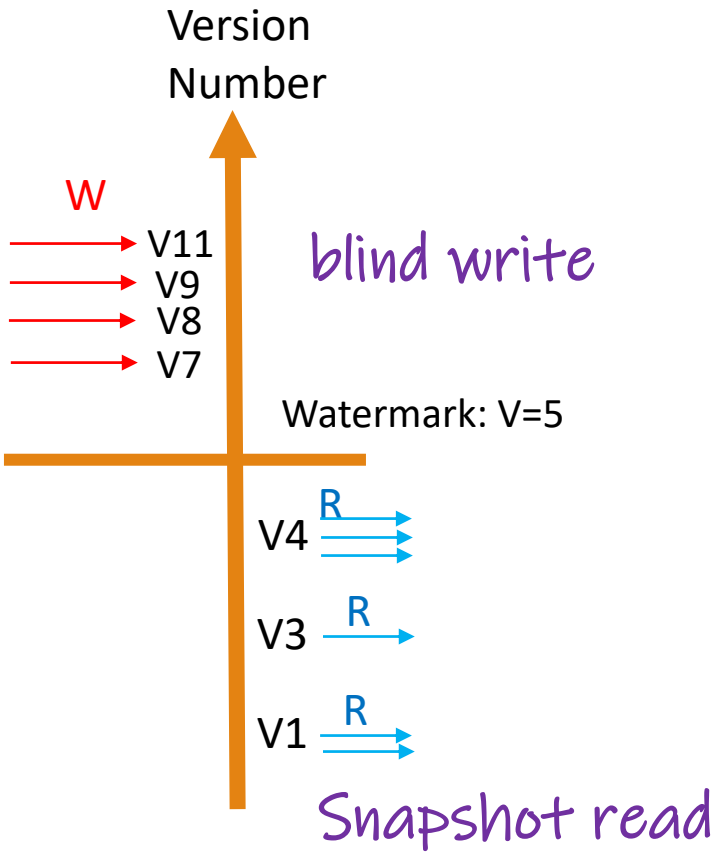
*Andy Pavlo: https://twitter.com/andy_pavlo/status/1051974710710407176

High Performance Geo-Distributed Transactions

Research Questions:

- Can we run **conflicting** transactions **in parallel** with strict serializability?
  - eliminate write-write conflicts and read-write conflicts (at least partly), and keep strict serializability
- Can **writes** complete in **1 RTT** with **quorum** acknowledgement and **read** data from **one** nearby copy?

Two scenarios that need NO concurrency control



No reads present, transactions are **write-only** and each writes a unique version against MV storage.

No writes present, transactions are **read-only** and access a historical snapshot version.

Insights

Transaction Commitment

Concurrency Control

Replication

are all about

visibility control

i.e., if a transaction is visible with respect to other transactions.

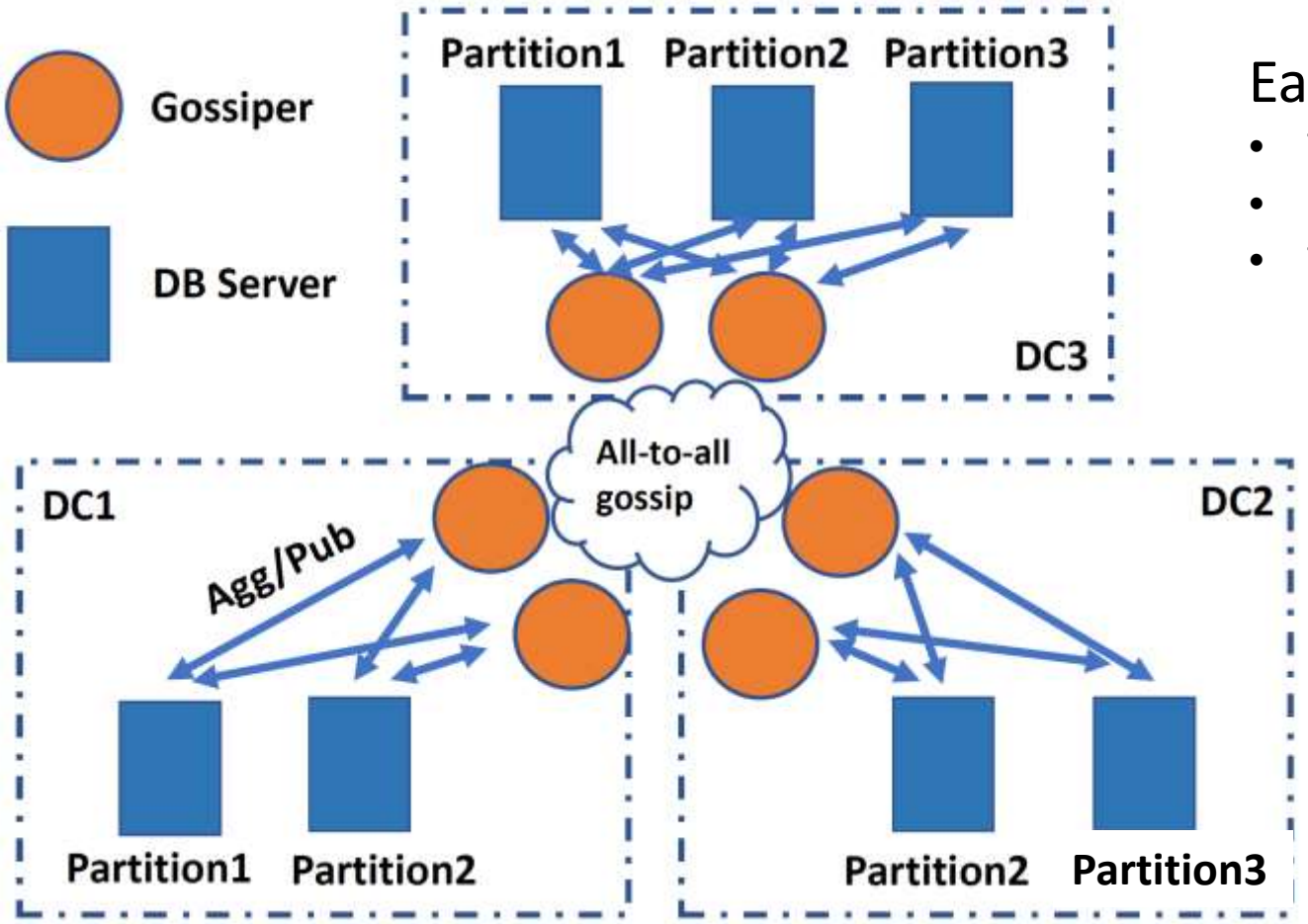
Ocean Vista (**OV**), using multi-versioning (**MV**), combines these functions into **a single protocol**, that gossips **watermarks**.

Transactions below the (visible) watermark are visible.

Contents

- Introduction
- Asynchronous Concurrency Control (ACC)
- Replication Protocol
- Experiments
- Summary

Architecture



Each DB Server:

- Txn Coordination
- MV storage
- Txn Execution

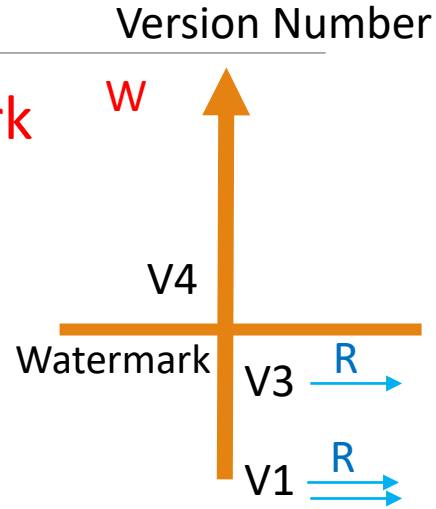
Gossipers:

- Redundant in DC
- Independent

Asynchronous Concurrency Control (ACC)

Sync. Txn Processing
(1)Read all keys
(2)Compute
(3)Write all keys

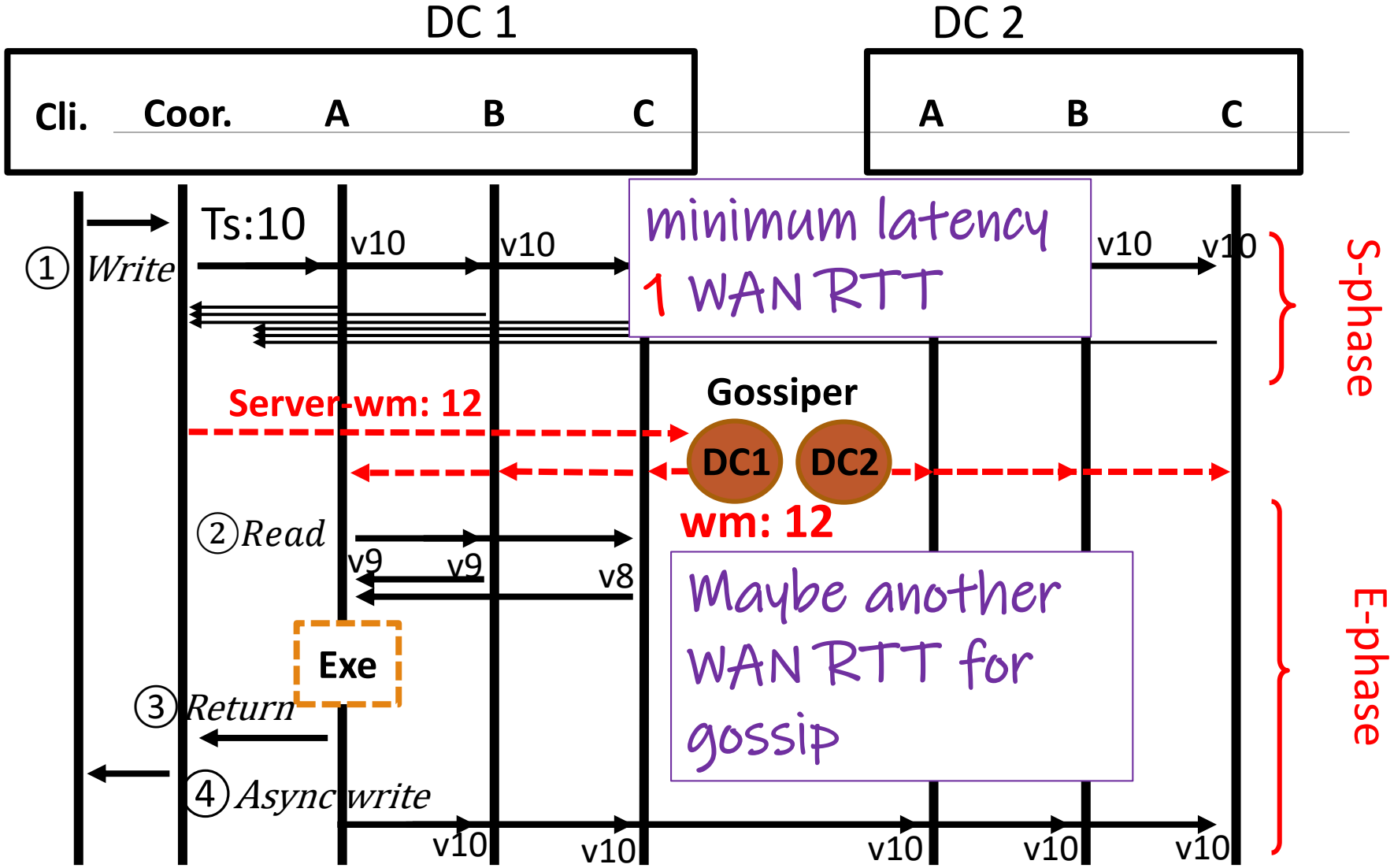
ACC
a) Write functors above watermark
b) Read & Compute
c) Async. Write below watermark



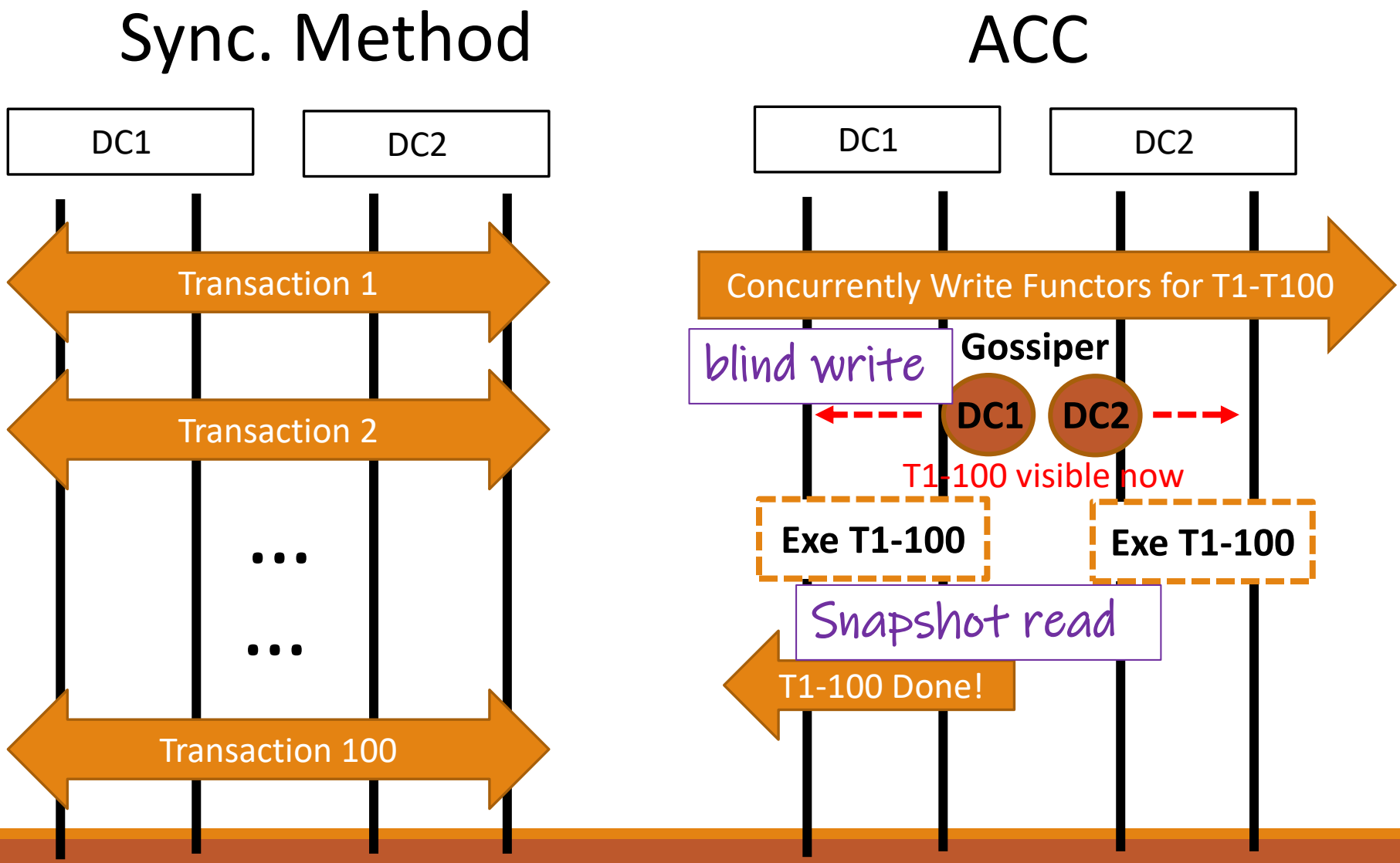
Transaction life cycle:

- a) ACC **writes functors (Fan&Golab ICDCS2018)** as data version placeholders & function of txn processing.
- b) Below watermark, transaction order is fixed; functors can read a **consistent snapshot version,** compute the final values.
- c) Async write, replaces the functors with the final values.

Example


DC 1
DC 2
Cli.
Coor.
A
B
C
A
B
C
Ts:10
① Write
v10
v10
v10
v10
minimum latency
1 WAN RTT
S-phase
Server-wm: 12
Gossiper
DC1
DC2
wm: 12
② Read
v9
v9
v8
Maybe another
WAN RTT for
gossip
Exe
③ Return
④ Async write
v10
v10
v10
v10
v10
E-phase

100 Transactions under Conflicts



Conflict Matrix: Parallelism when Keys Overlap

Sync. CC e.g., Spanner (2PL), TAPIR (OCC)

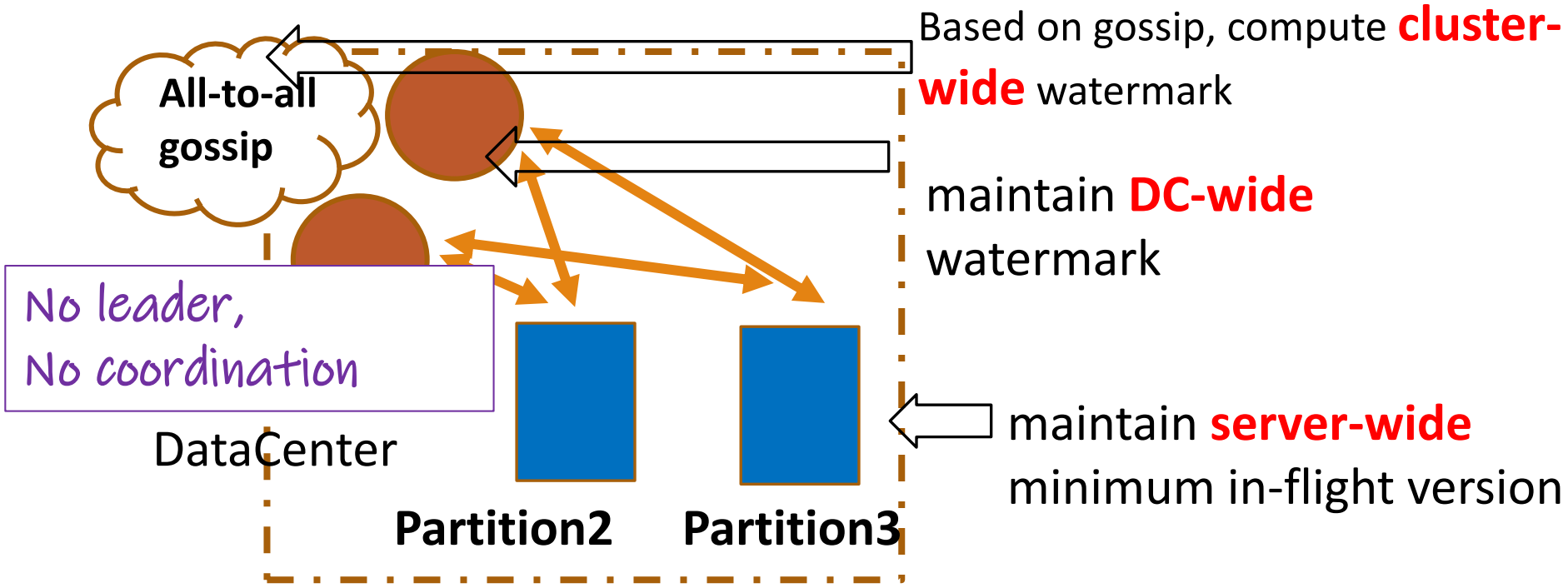
	ReadSet	WriteSet
ReadSet	✓	✗
WriteSet	✗	✗

Async. CC e.g., OV

	WriteOnly	ReadOnly	AsyncWrite
WriteOnly	✓	✓	✓
ReadOnly	✓	✓	partially
AsyncWrite	✓	partially	✓

Gossip of watermarks

- Txn versions (globally unique) are assigned by loosely synchronized clocks; version number is server-wide **monotonically increasing**. [**No central component**]
- Watermarks are all **monotonically increasing**.



Contents

- Introduction
- Asynchronous Concurrency Control (ACC)
- Replication Protocol
- Experiments
- Summary

Replication Protocol

Write-All Read-One

Must wait for stragglers or failed nodes

Write-Quorum
Read-Quorum

Pay cost on read: read leader (bottleneck) or read quorum (more work)

Write-One Read-All

Lost data on failure

OV

- Write-Quorum Read-One (common case)
- Maintain **fully-replicated watermark**, below it Read-One.
- Write success in 1 RTT in **fast path** or 2 RTT in **slow path** (**NO conflicts on write**, only with more failures).

Fault Tolerance

- DB Server Failure
- Gossiper Failure
- DC Failure

Detailed in the paper.

Contents

- Introduction
- Asynchronous Concurrency Control (ACC)
- Replication Protocol
- Experiments
- Summary

Experiments

Questions to answer:

- How much transaction processing parallelism is there under conflicts?
- What is the latency overhead of gossip?

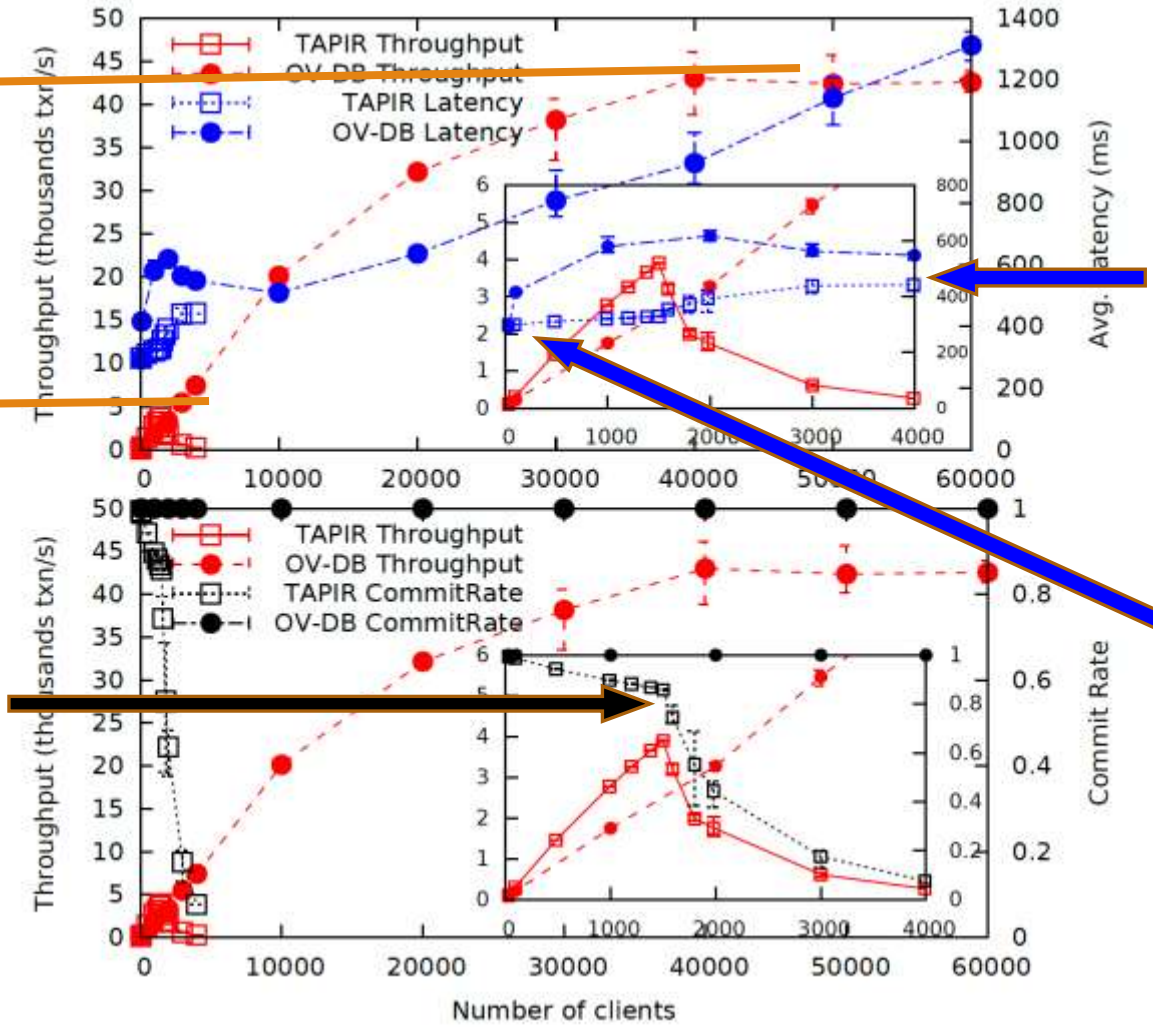
Settings:

- 3 shards and 3 replicas in Asia,EU,US, max WAN RTT 253ms
- YCSB+T benchmark, each txn read-modify-writes 4 keys
- Distribution 1: Zipf coefficient 0.5
- Distribution 2: contention index (CI), 1 hot key and 3 cold keys
- Compare with TAPIR*

*I. Zhang, etc. Building consistent transactions with inconsistent replication.ACM Trans. Comput. Syst., 35(4):12:1-12:37, Dec.2018

Throughput, Latency and Commit Rate

10X on peak throughput



OV pays cost at higher latency, around **1 WAN RTT**.

Both have best latency at **1 WAN RTT**.

Commit Rate:

TAPIR drops as pair-wise conflicts increase;

OV never aborts a txn due to conflicts.

No abort workload

- Keyspace has only 1000 hot keys [CI-0.001]. TAPIR uses 1000 clients, each accesses unique keys. [CI-fix]
  - **Max throughput TAPIR can achieve, probably.**
  - **No conflicts.**
- OV uses the same key distribution but more clients.
  - **Has conflicts.**

	OV-DB	TAPIR	Speedup
CI-fix	39247	2781	14x
CI	29580	465	64x
Speedup	1.3x	6x	

Comparable to:
14 conflicting transactions running in parallel

Summary

- Distributed transaction protocol is all about **visibility control.**
- Async. CC can run **conflicting transactions in parallel.**

	WriteOnly	ReadOnly	AsyncWrite
WriteOnly	✓	✓	✓
ReadOnly	✓	✓	partially
AsyncWrite	✓	partially	✓

- Watermarks enable simple and efficient replication.

Write-All Read-One
Write-Qrm.Read-Qrm
Write-One Read-All

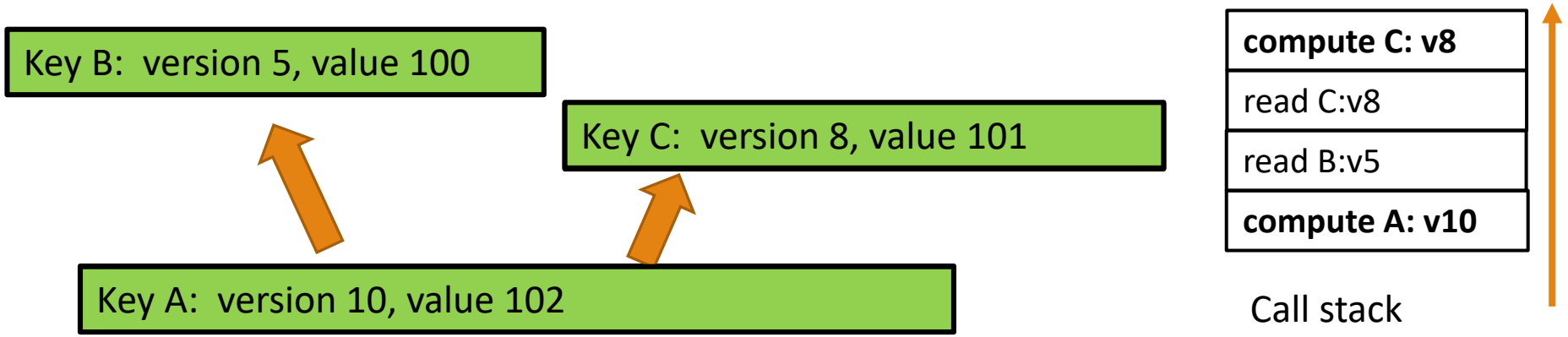
Write-Quorum Read-One

Thank you!

Hua Fan
guanming.fh@alibaba-inc.com

Backup Slides

Compute functors recursively



Computing a functor requires reading the latest snapshot version of the keys in its read set.

The snapshot version functor will be computed recursively in the read procedure if it is not already a final value.

Recursive execution resembles a rescheduling of the functor computing order.