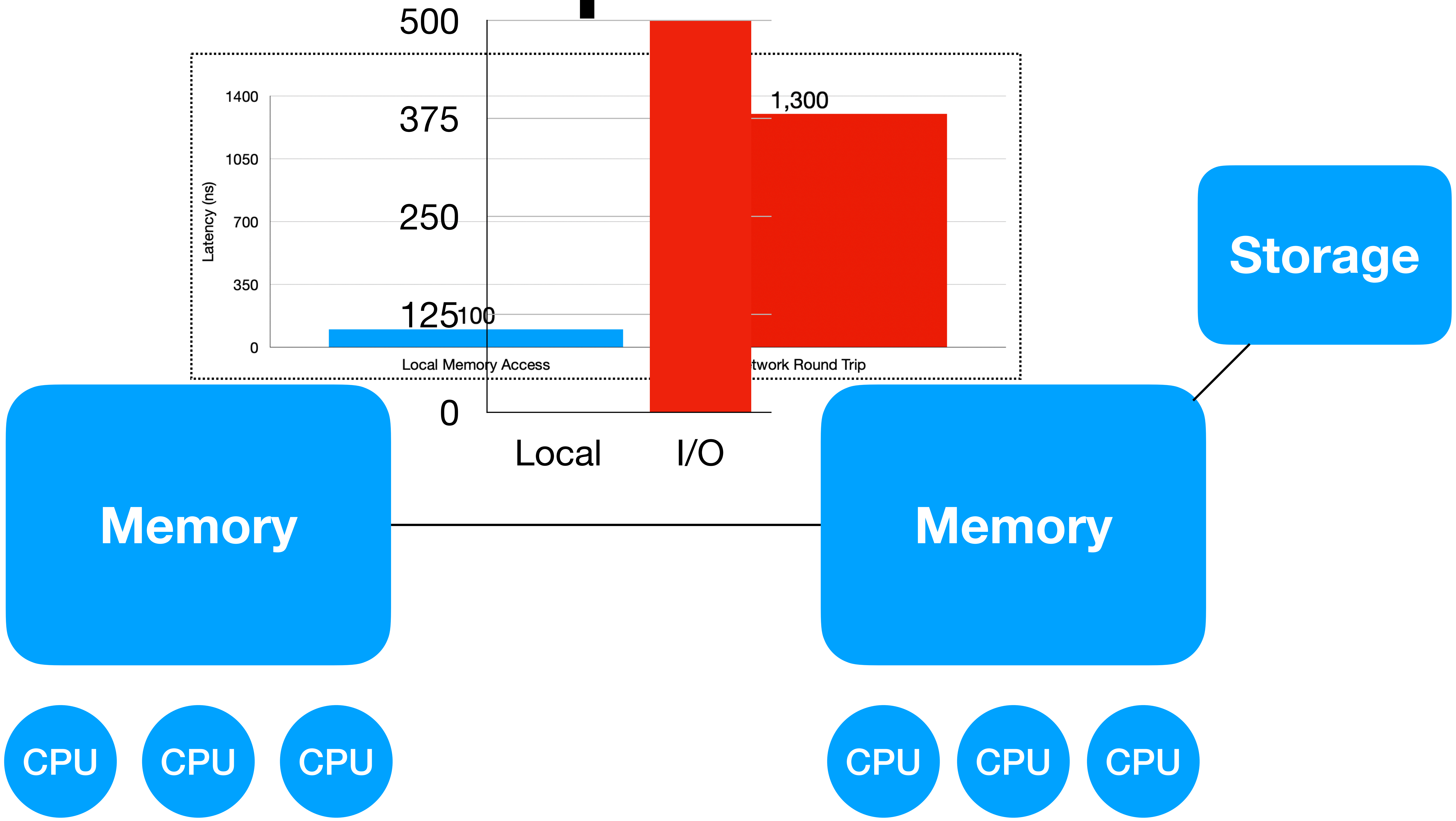
The Effects of Fast I/O on Concurrent Computing

Naama Ben-David

Technion
Israel Institute of Technology

I/O Speeds

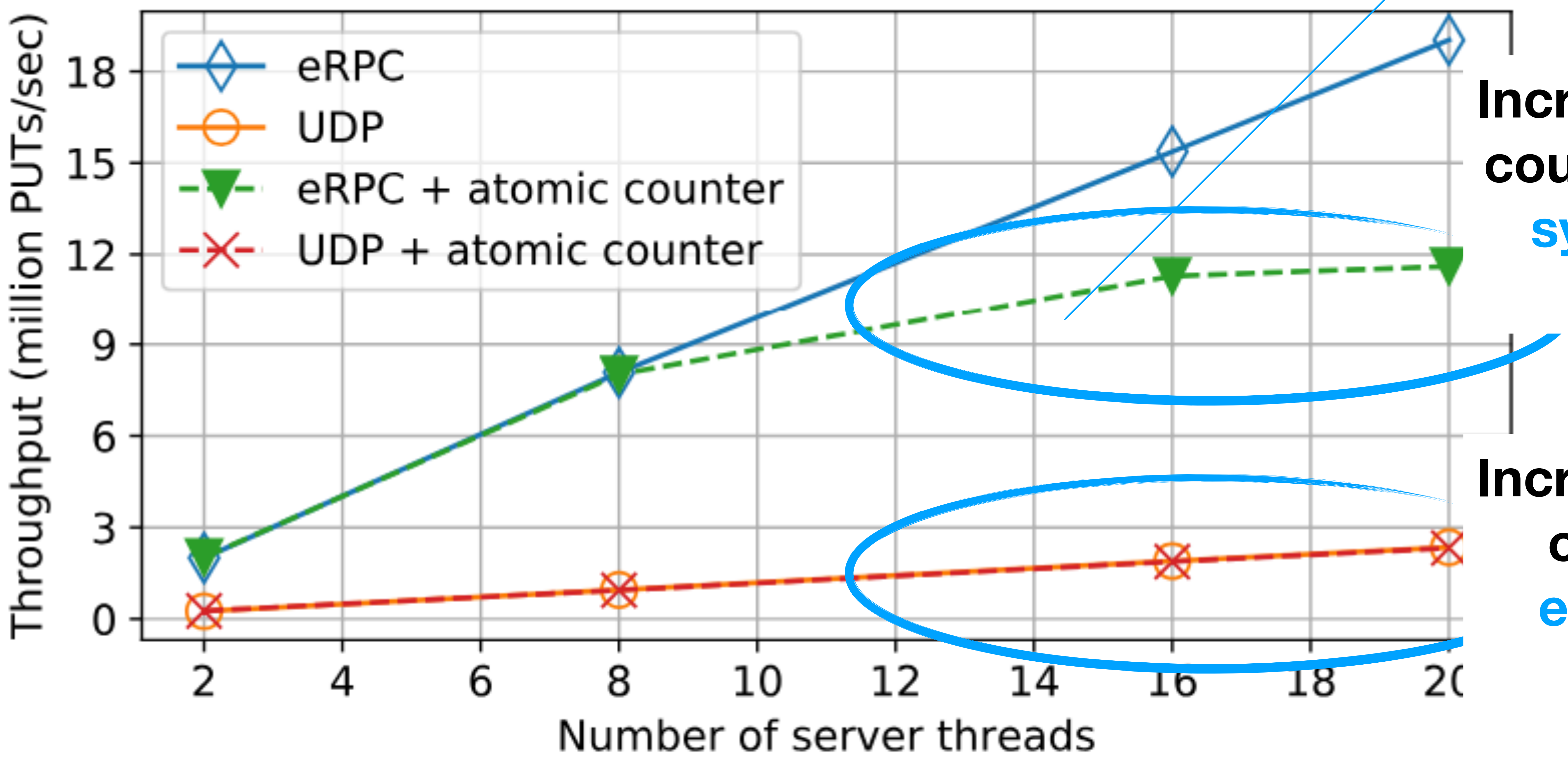


Traditionally, systems only optimize I/O

Now must optimize in memory processing and parallelism too

CPU Bottleneck

eRPC — modern fast message passing
UDP — traditional message passing



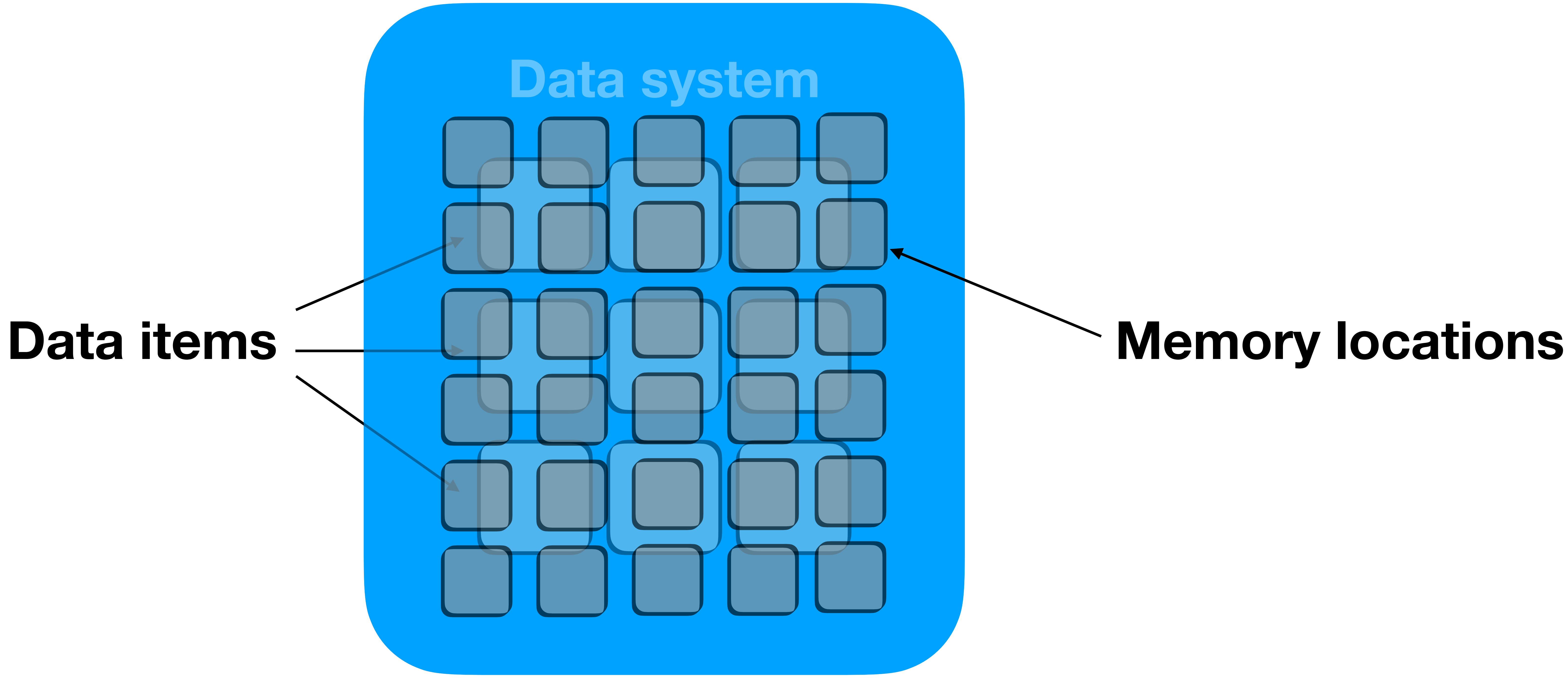
Incrementing atomic counter bottlenecks system with fast messaging

Incrementing atomic counter has no effect with slow messaging

Meerkat: Multicore-Scalable Replicated Transactions Following the Zero-Coordination Principle. Szekeres et al EuroSys'20

How can we optimize concurrency in I/O systems?

Transactions

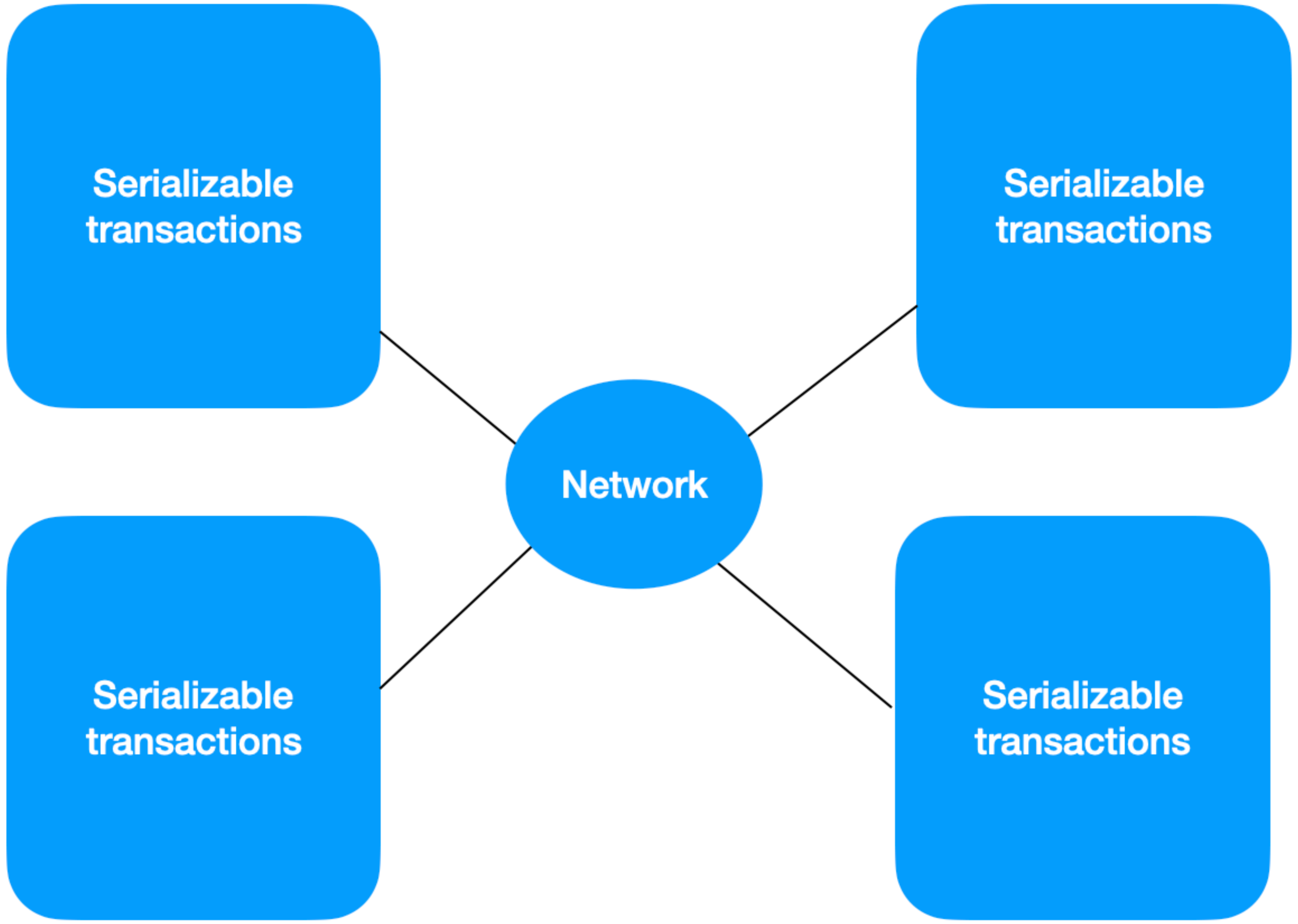


Transactions specify which data items to read and write (read and write sets)

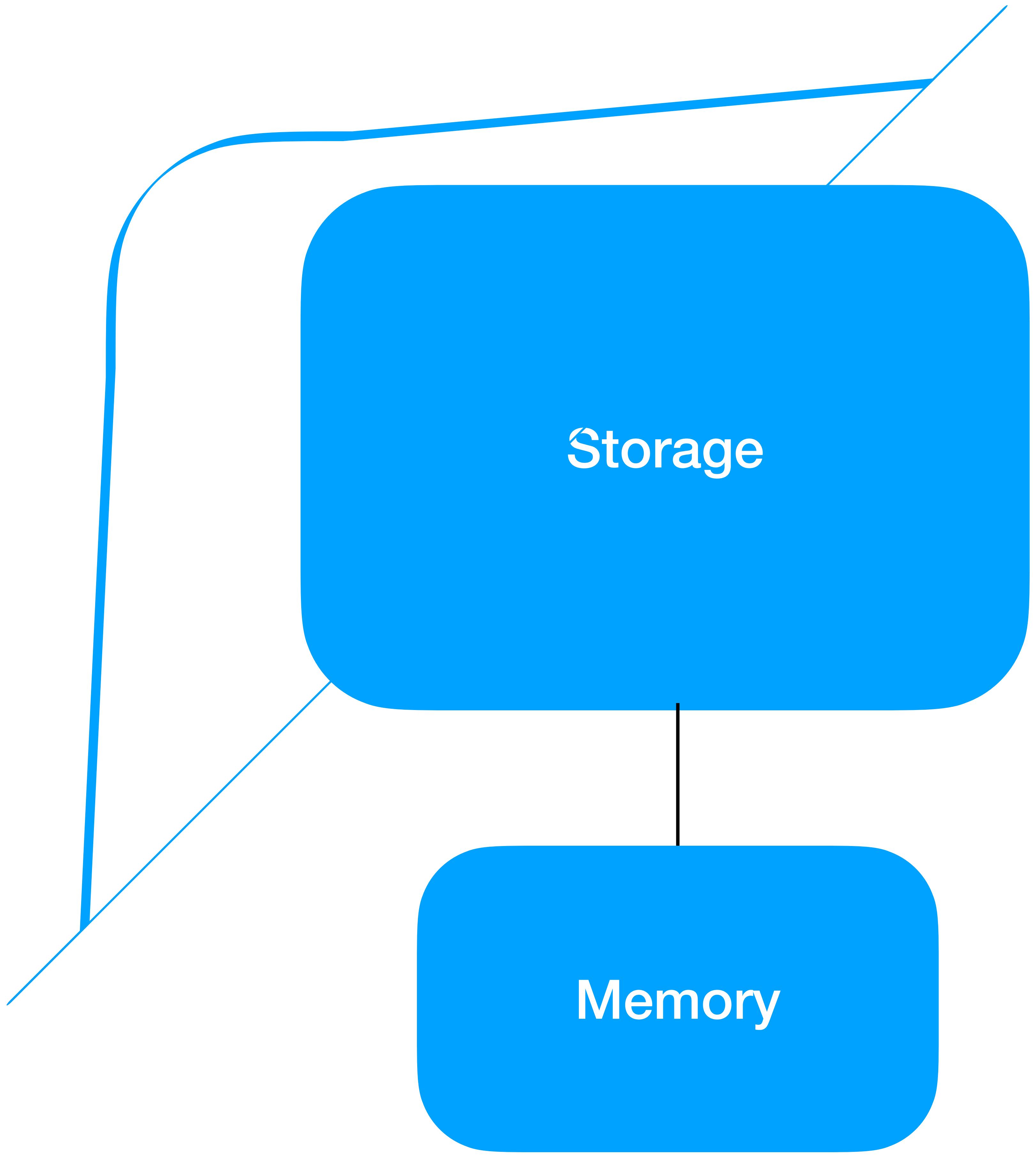
Implemented via accesses to memory locations

Serializability: transactions either *commit* atomically or *abort* with no effect

In This Talk



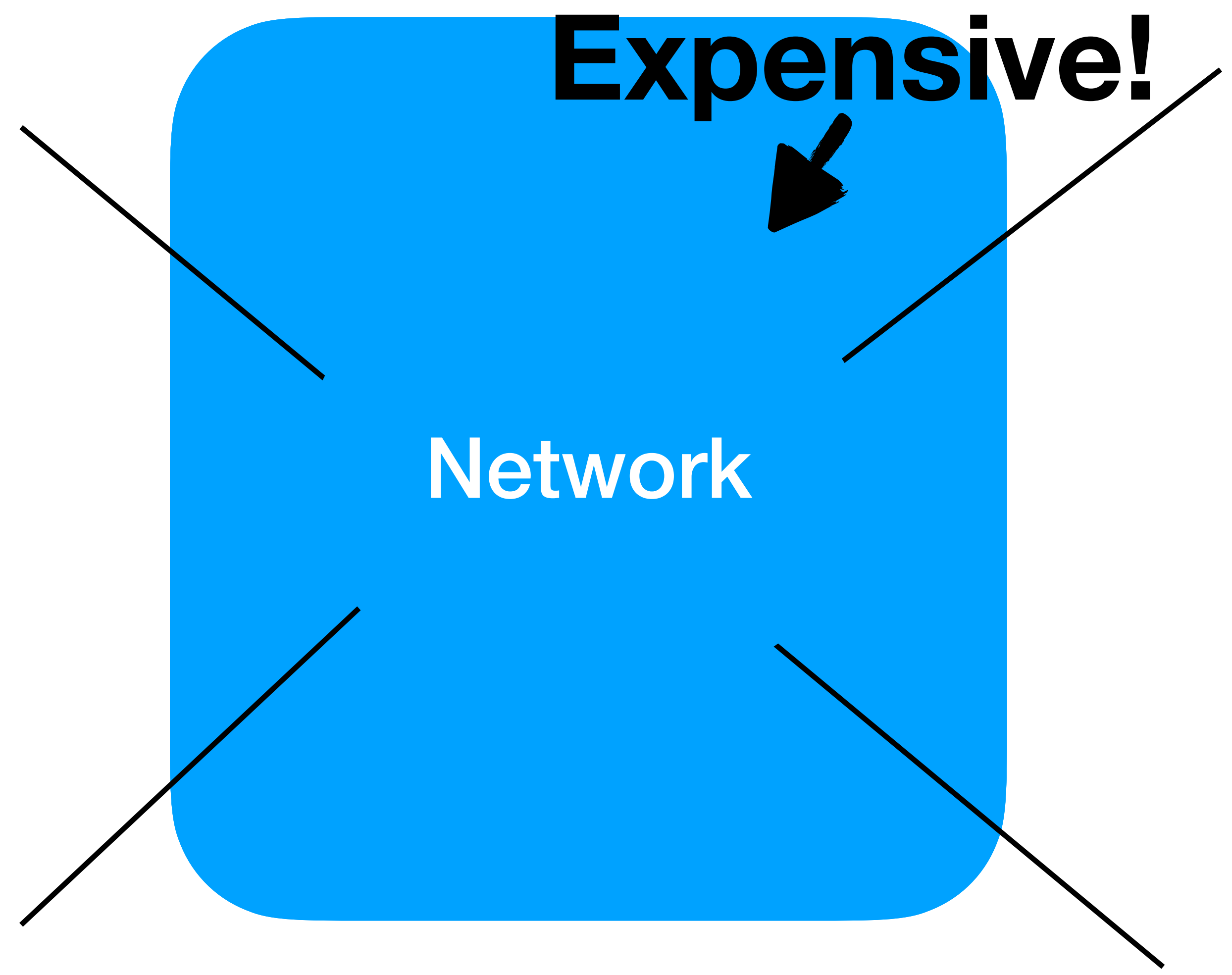
Distributed Transactional Systems



On-Disk Transactional Systems

Distributed Transactional Systems

Parallelism within each node



Clean, powerful abstraction

Goal: few round trips to commit

Distribute for:

- More data storage
- Decreased workload
- Fault tolerance
- …

Performance bottlenecks

Traditionally: **network**

Today: also **in-node**

How do we make use of parallelism in each node?

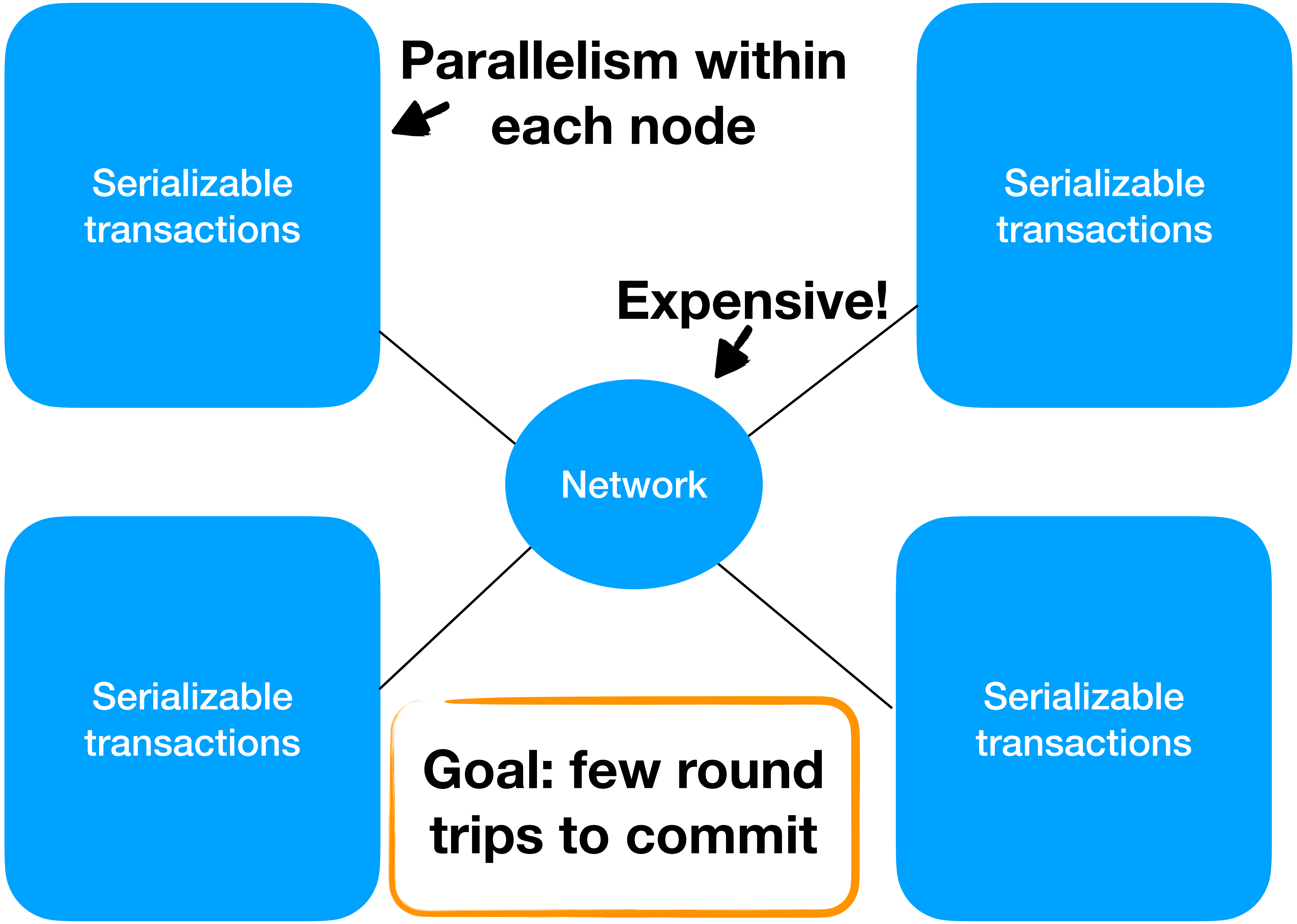
Parallel Performance

Avoid contention

- **Disjoint access parallelism:** if the data sets of two transactions do not overlap, their shared memory accesses shouldn’t overlap

- **Invisible reads:** a larger read set shouldn’t cause more shared memory modifications

Distributed Transactional Systems



Distribute for:

- More data storage
- Decreased workload
- Fault tolerance
- …

Performance bottlenecks

Traditionally: **network**

Today: also **in-node**

Distributed Performance

Fast Decision

Intuition: In good executions, transactions commit ***as fast as possible***

Challenge: Transactions need different amounts of time to find out their data set

In a synchronous failure-free execution with no conflicts,
a transaction must terminate within one network round trip
after some process *knows* its data set

Main Result: FIDS Theorem

$\exists n \,.\, D_n \subset D$

No **sharded** system guarantees **weak progress** and

Non-triviality; Cannot abort if no concurrent transactions

Fast decision

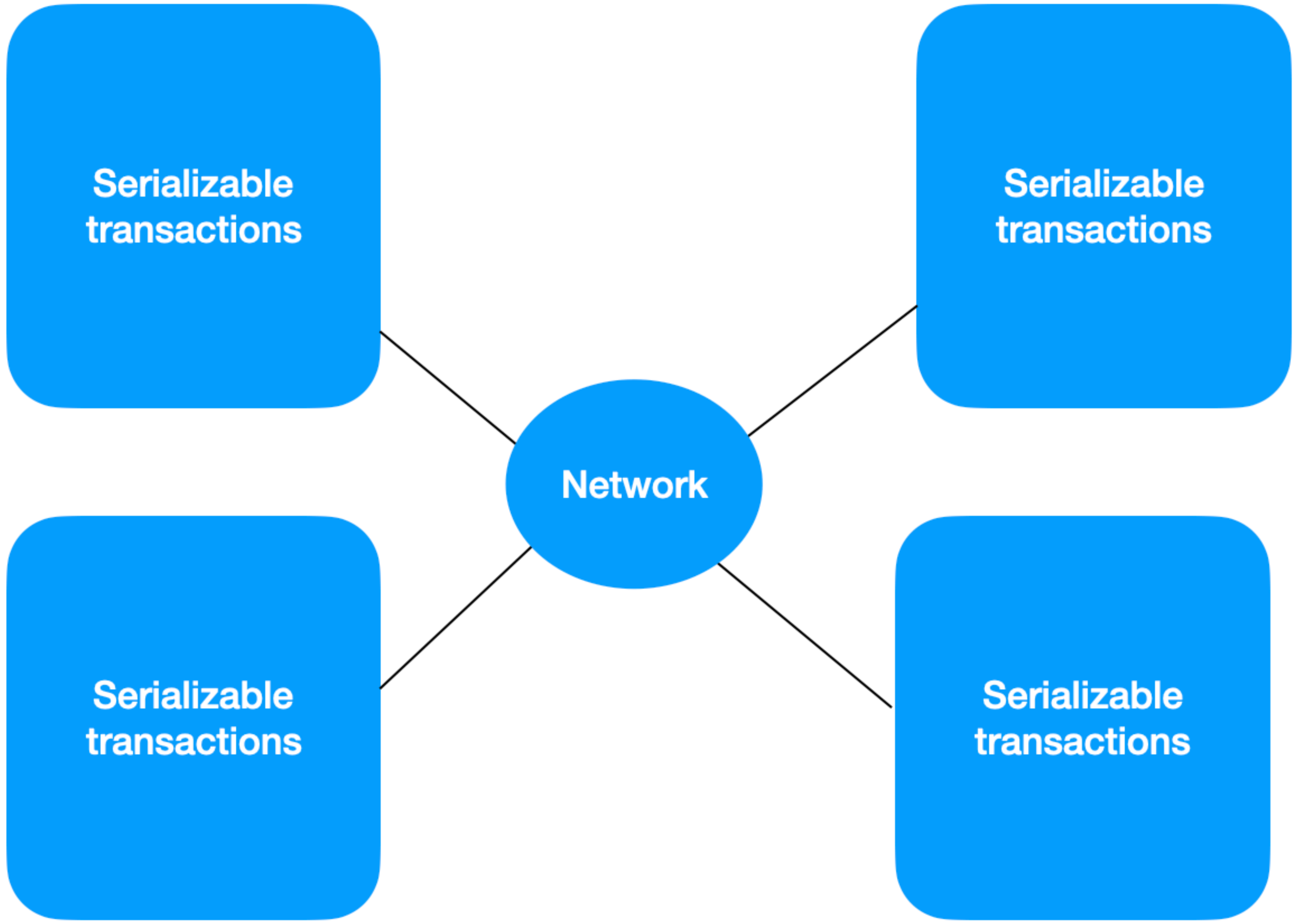
Invisible reads

Disjoint access parallelism

Serializability

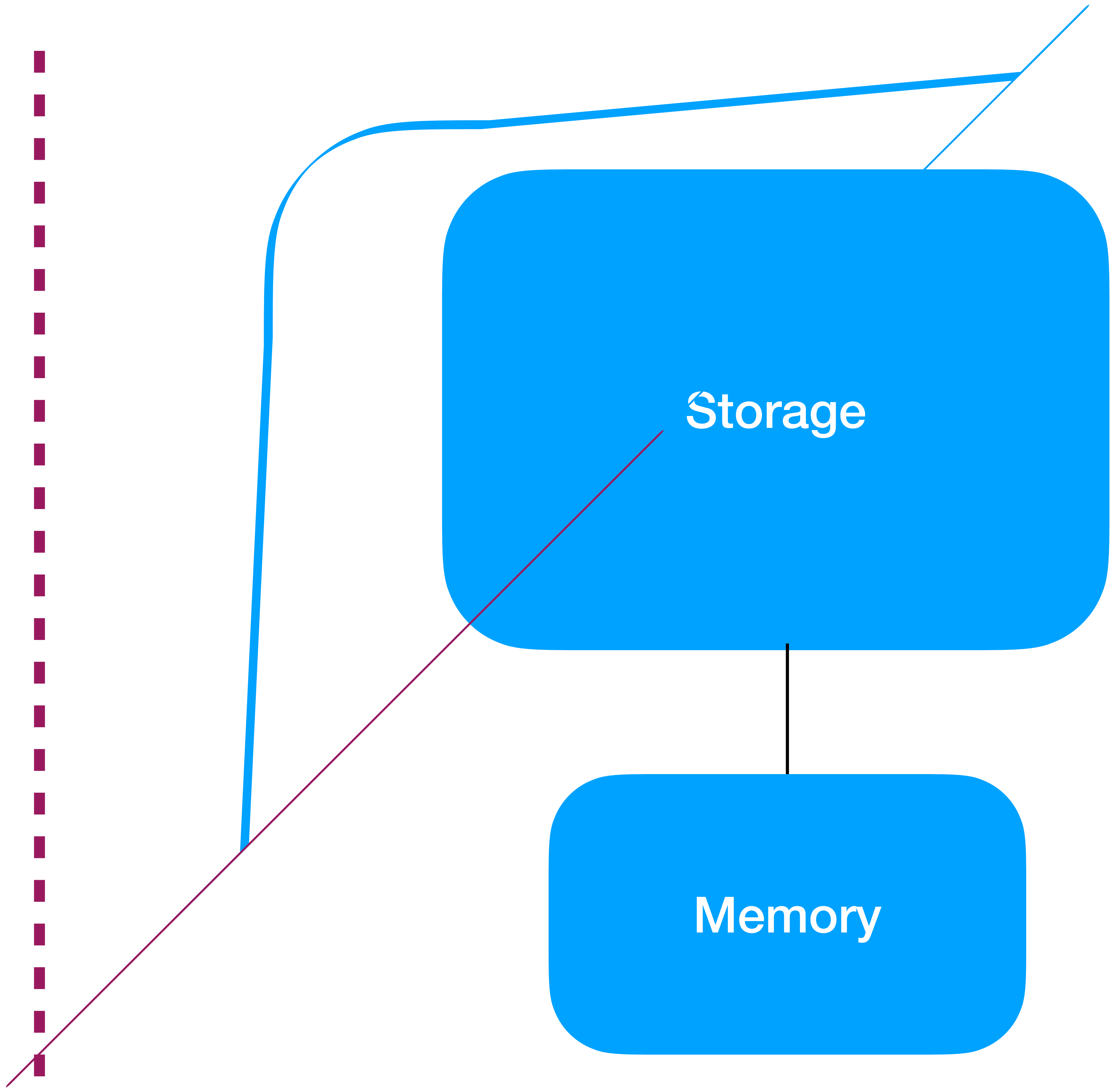
[B Sela Szekerez DISC'23]

In This Talk



Distributed Transactional Systems

Impossibility Result

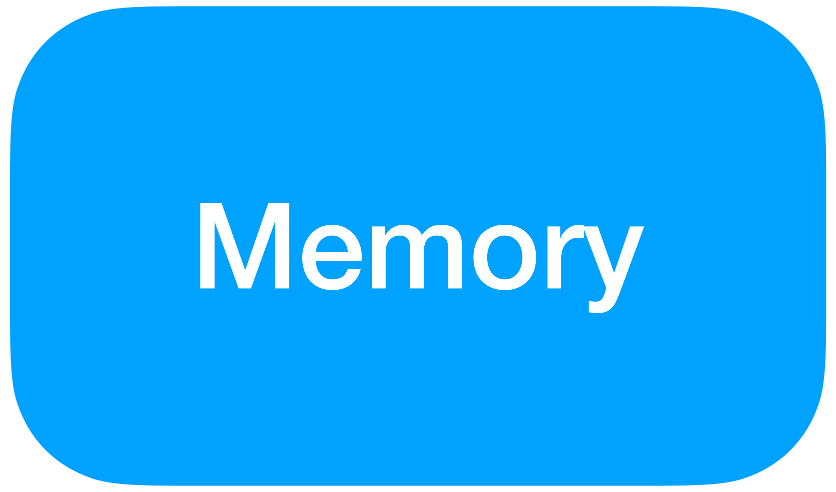


On-Disk Transactional Systems

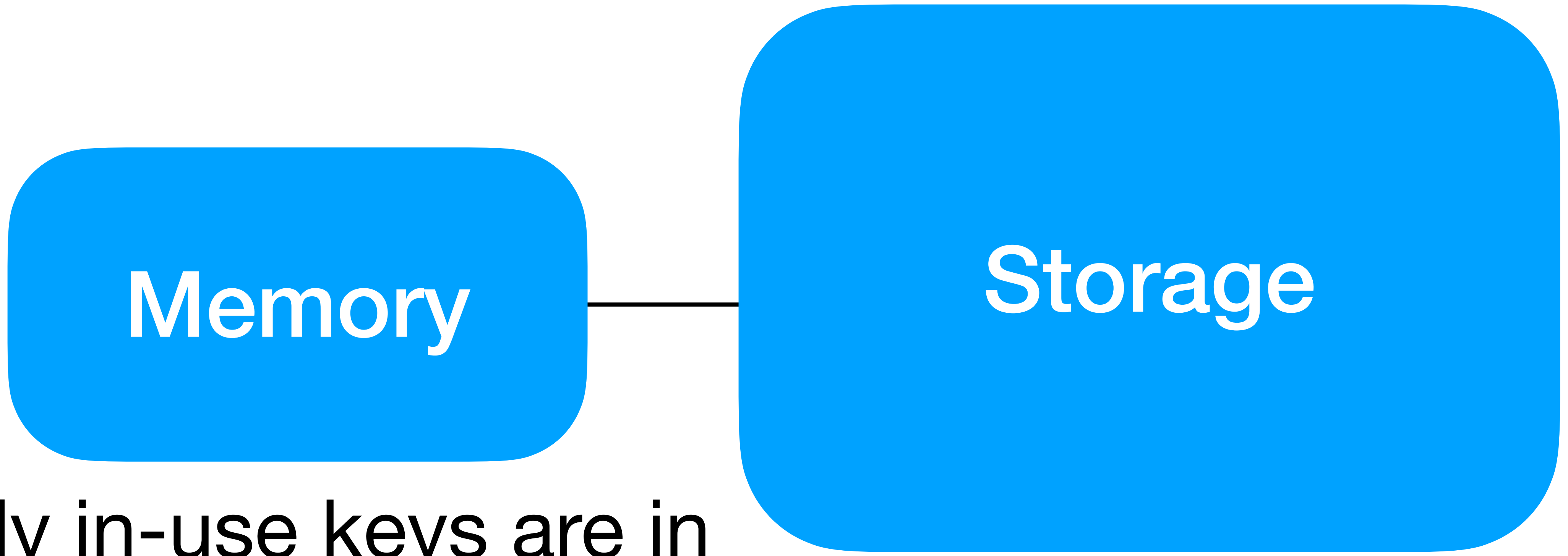
Transactional Databases

Key question: can data fit in memory?

Yes:
In-memory database



No:
On-Disk Database



Only in-use keys are in memory

Concurrency Control Mechanisms

In-Memory

- Usually, per-key timestamps

Too big for on-disk!

- Example: maintain read and write timestamps
- Fine grained concurrency control, few aborts

On-Disk

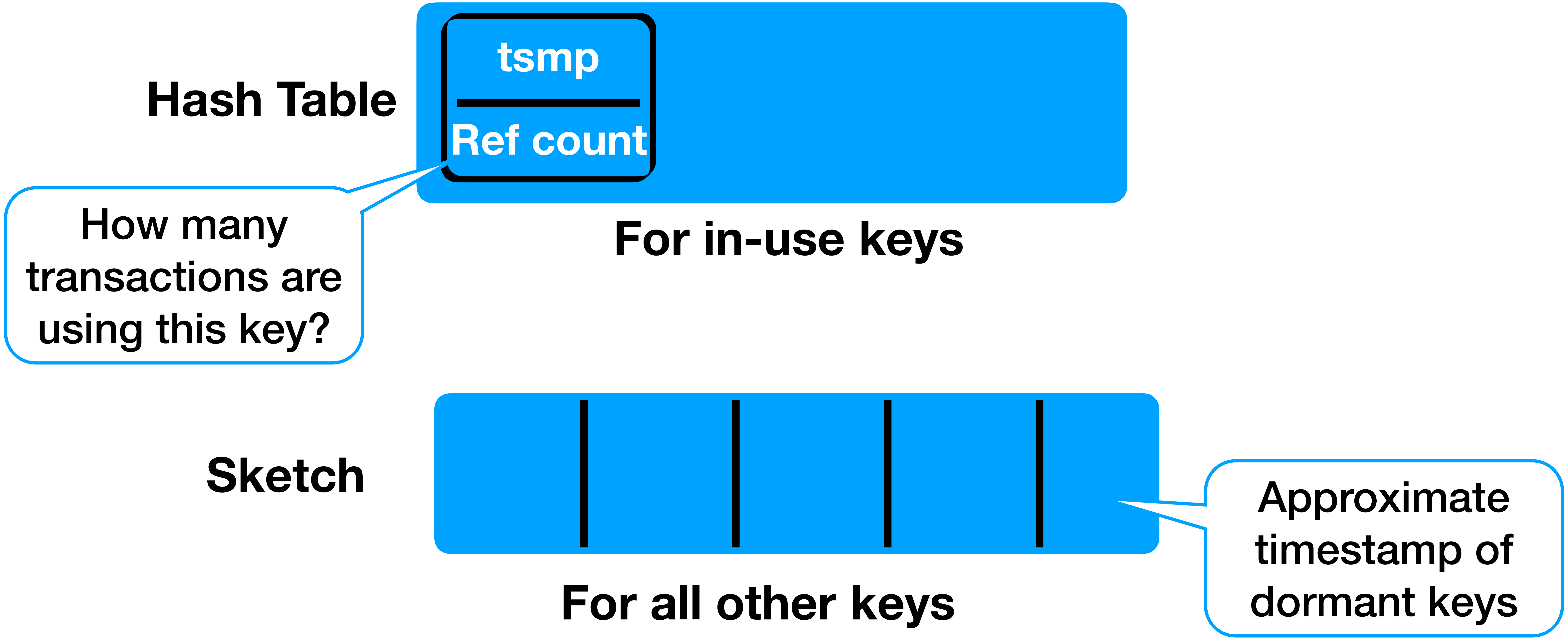
- Usually, timestamp/metadata per ongoing transaction
- Heavy-weight comparison against all ongoing transactions before commit

Too slow for fast storage!

- Higher abort rate

How can we get in-memory CC speeds with less metadata?

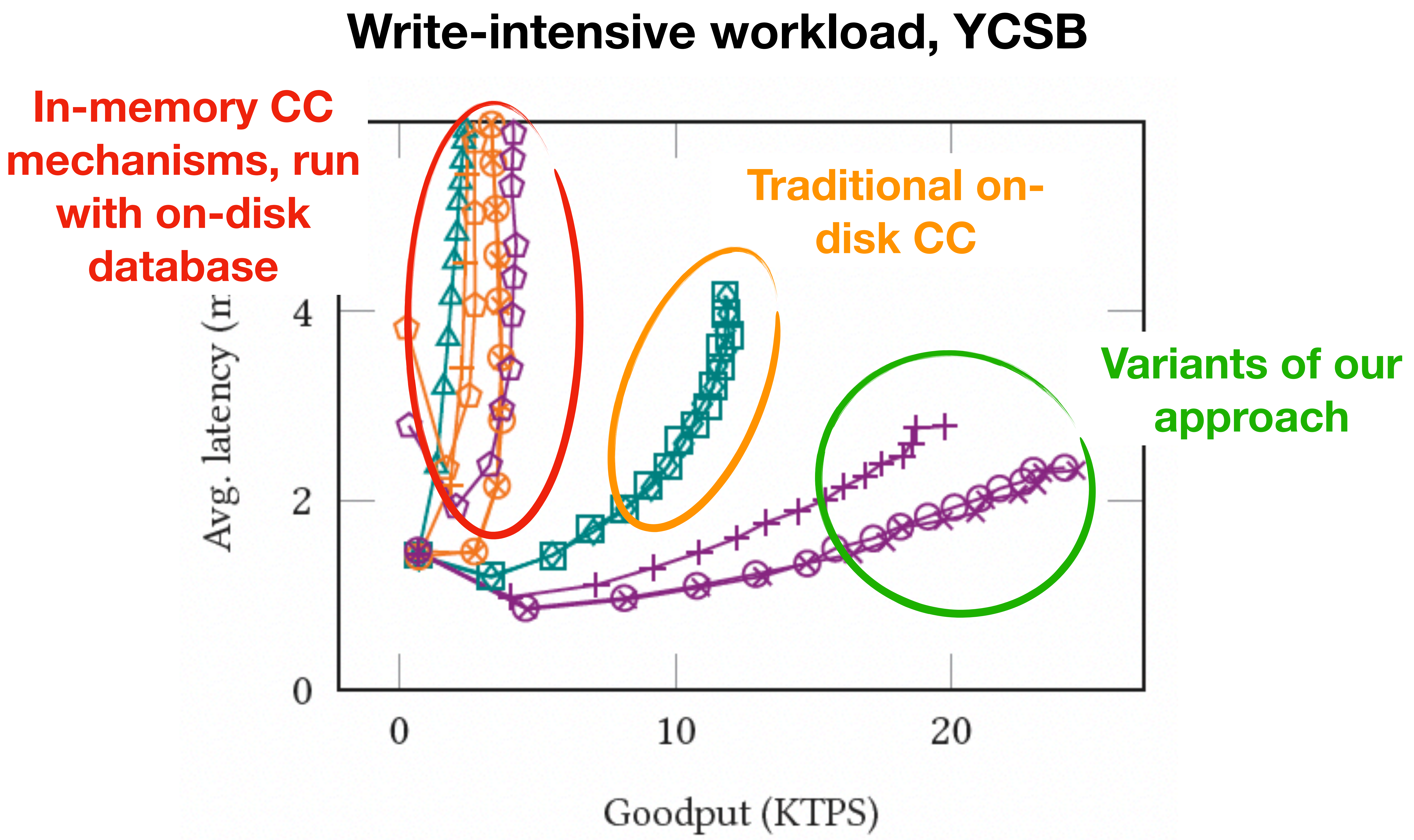
Our Solution: Approximate Timestamping



When ref count hits 0, key moves back to sketch, min/max with current entry

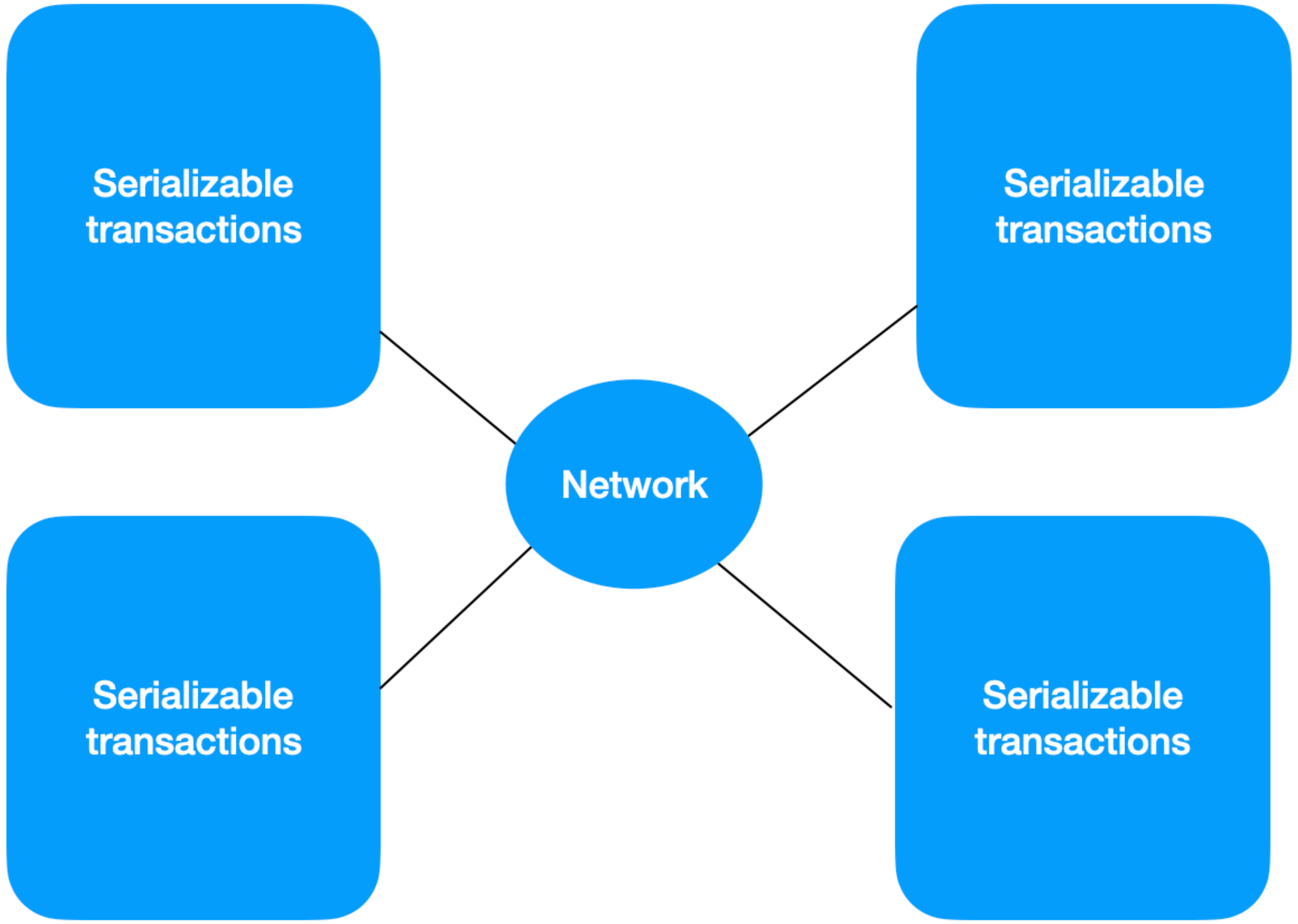
[Hwang B Conway Garcia-Alvarado Johnson Szekerez Yuan]

Experimental Results



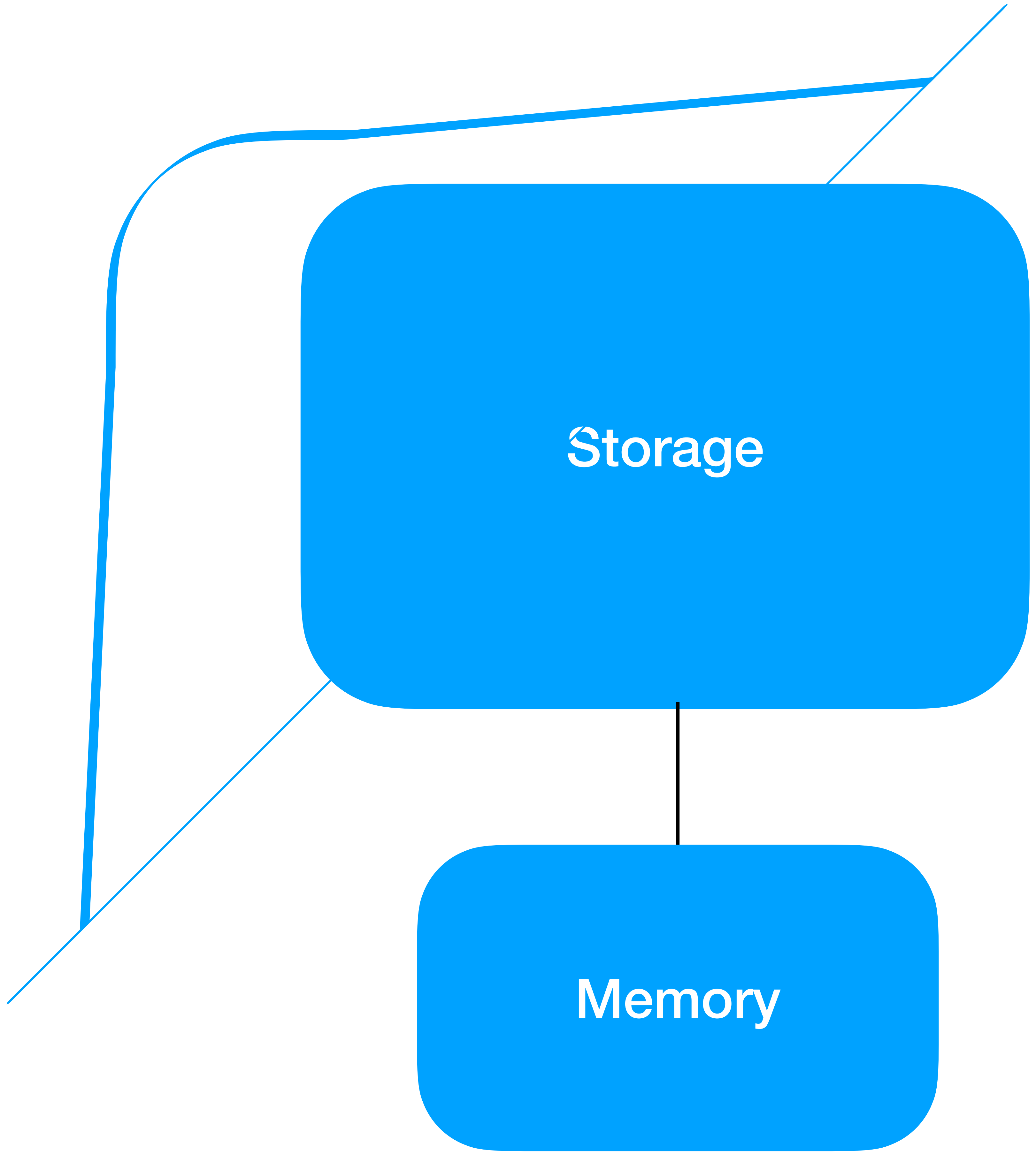
[Hwang B Conway Garcia-Alvarado Johnson Szekerez Yuan]

In This Talk



Distributed Transactional Systems

Impossibility Result



On-Disk Transactional Systems

Approximate Timestamping

Concluding Thoughts

- Faster I/Os are changing how concurrency should be used in large systems
- Showed impossibility in distributed concurrent transactions;
    - **What are good algorithms that optimize both parallelism and network communication as much as possible?**
- Only considered transactional systems;
    - **How do fast I/Os affect other problems in concurrent computing?**

Thank you!