Strongly Consistent Transactions for Enterprise Applications

Using Software Transactional Memory to Improve Consistency and Performance of Read-Dominated Workloads

Sérgio Miguel Martinho Fernandes

TÉCNICO LISBOA

July 4th, 2014

inesc id lisboa

Consistency Matters

- Concurrent programming is difficult
- Strong consistency makes it easier

“To be sure of **correctness** you should always use the **serializable** isolation level.”

-- Martin Fowler (PoEAA, 2002)

"To be sure of **correctness** you should always use the **serializable** isolation level."

So... why don't we always enforce it?

-- Martin Fowler (PoEAA, 2002)

“To be sure of **correctness** you should always use the **serializable** isolation level.”

“[…] choosing serializable really messes up the liveness of a system, […] you often have to reduce serializability […] to increase throughput.”

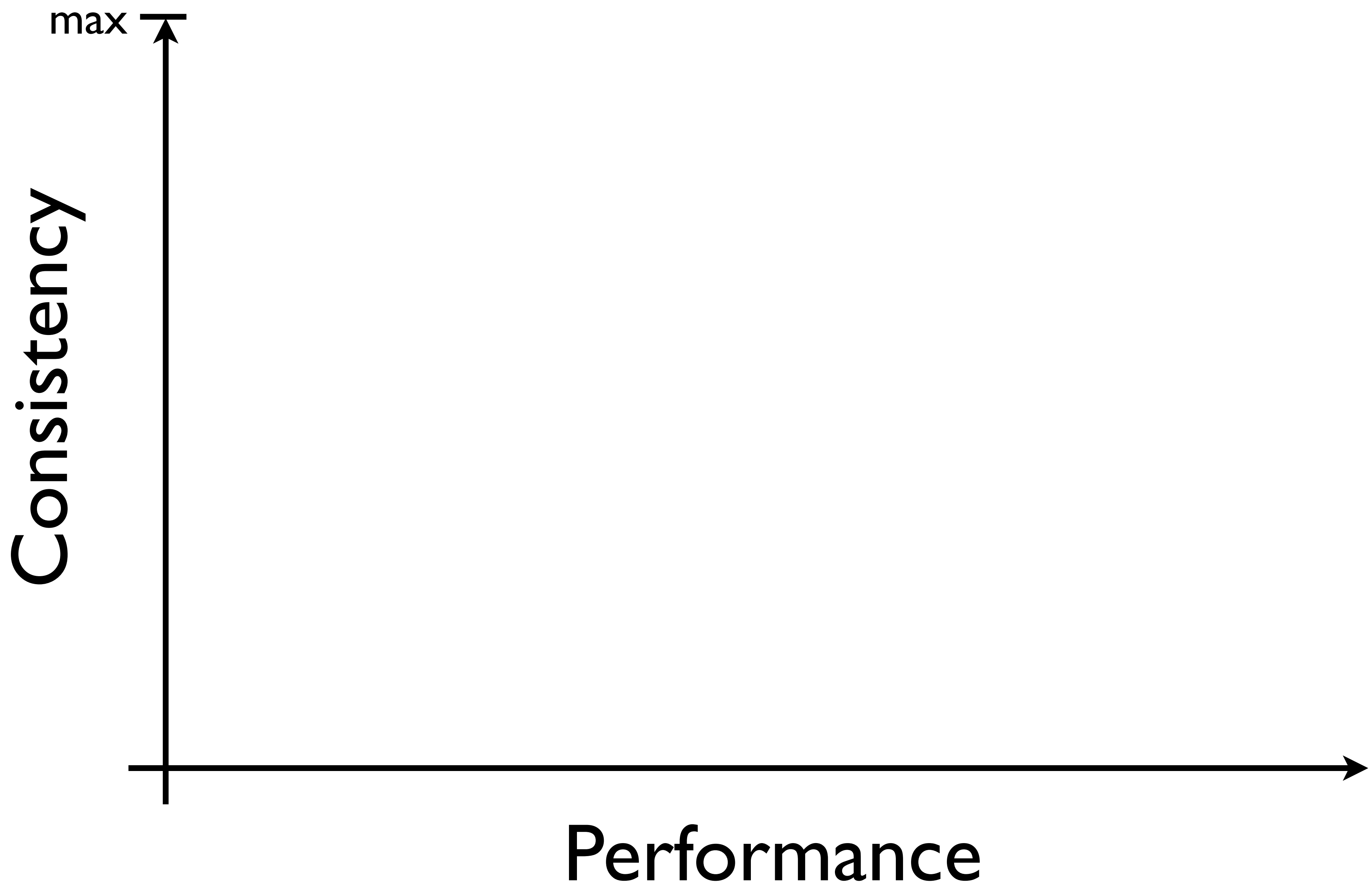
-- Martin Fowler (PoEAA, 2002)

“To be sure of **correctness** you should always use the **serializable** isolation level.”

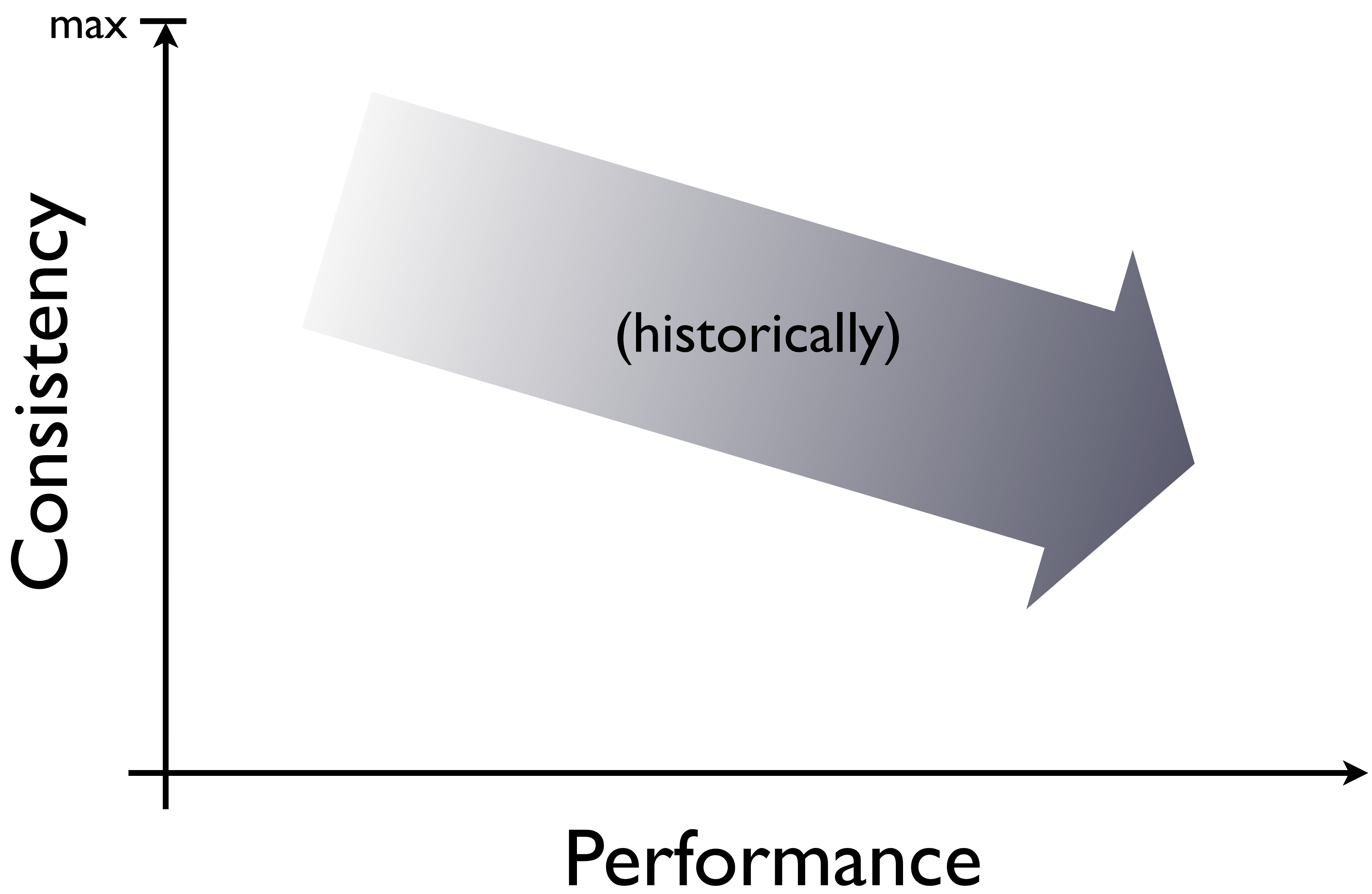
“[…] choosing serializable really messes up the liveness of a system, […] you often have to reduce serializability […] to increase throughput.”

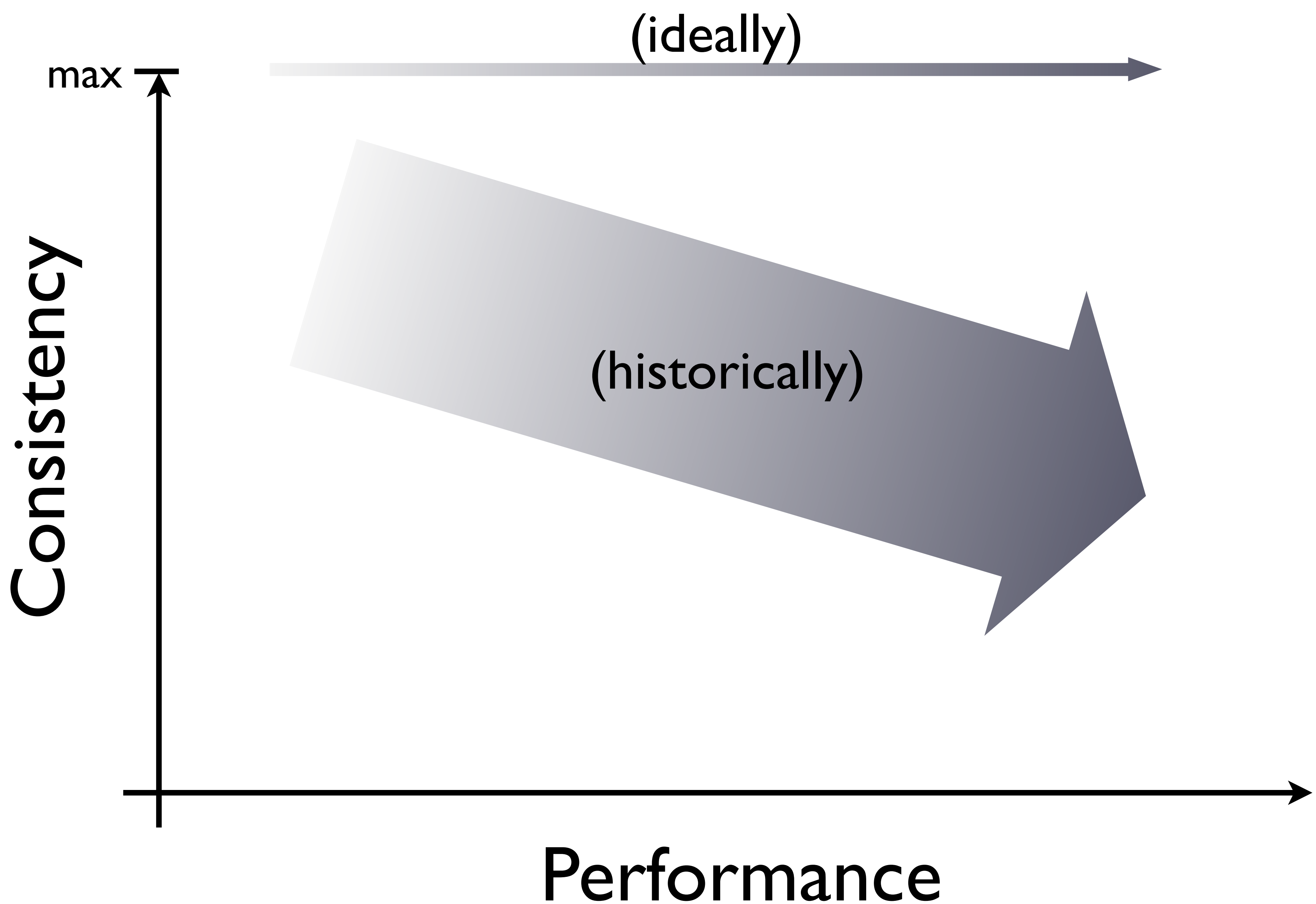
“You have to decide what risks you want take and make your own trade-off of **errors** versus **performance**.”

-- Martin Fowler (PoEAA, 2002)



max
Consistency
Performance


max
Consistency
(historically)
Performance



(ideally)
max
Consistency
(historically)
Performance

Consistency Really Matters

Consistency Really Matters

-- Google's F1 team (Shute et al, VLBD, 2013)

Consistency Really Matters

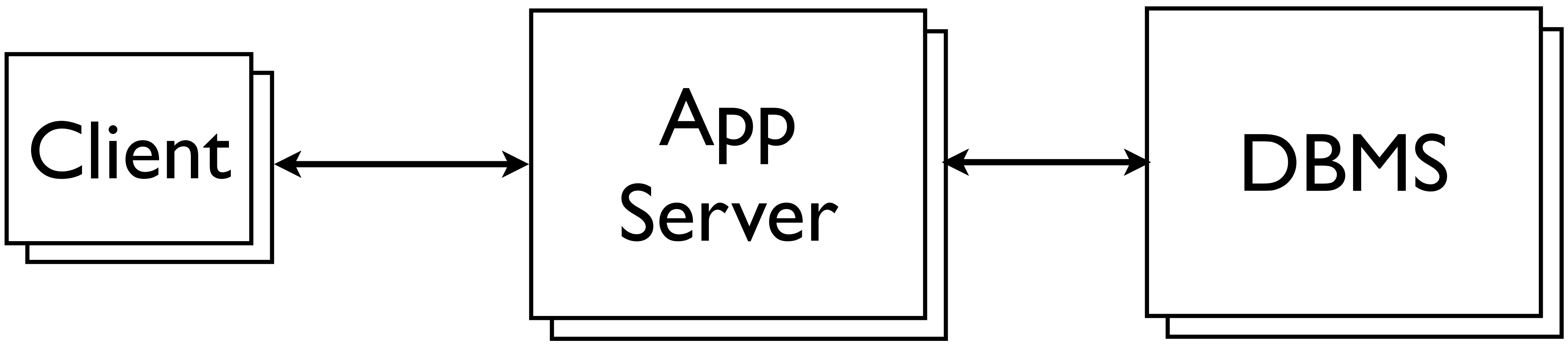
"The system […] must always present […] **consistent data**."

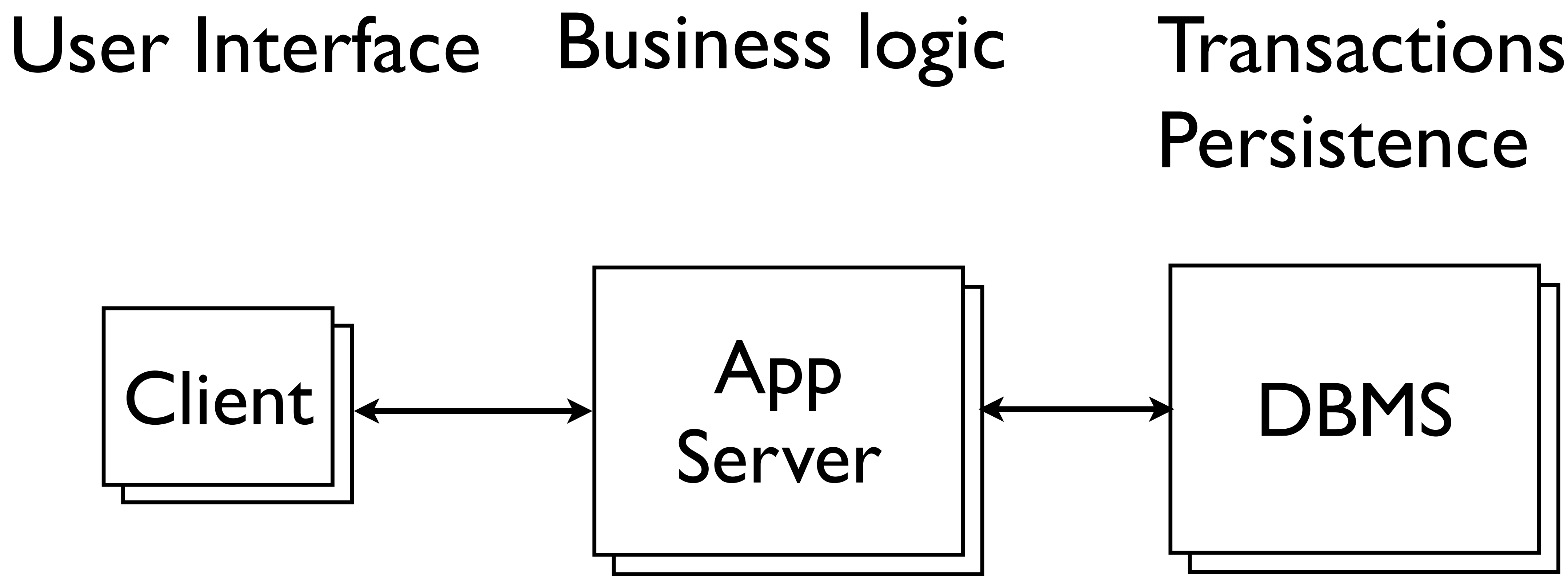
"[…] to cope with concurrency anomalies […] is very **error-prone**, **time-consuming**, and ultimately **not worth the performance gains**."

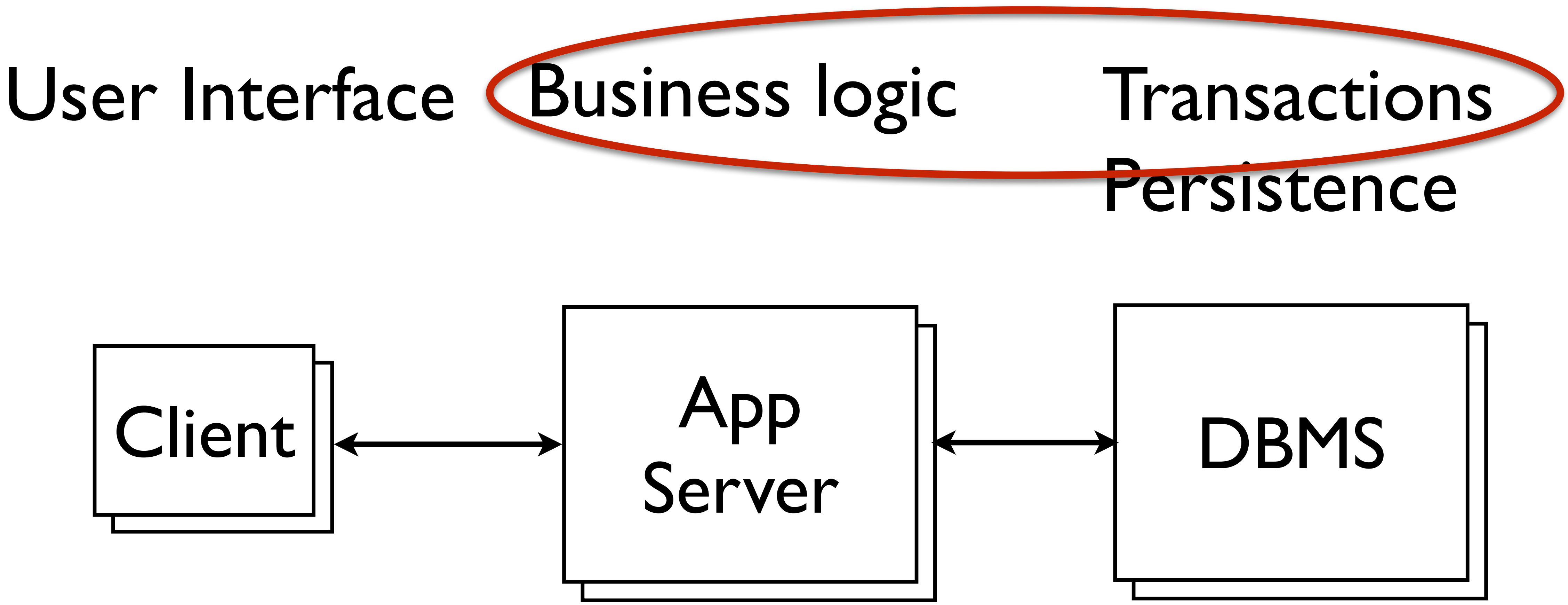
-- Google's F1 team (Shute et al, VLBD, 2013)

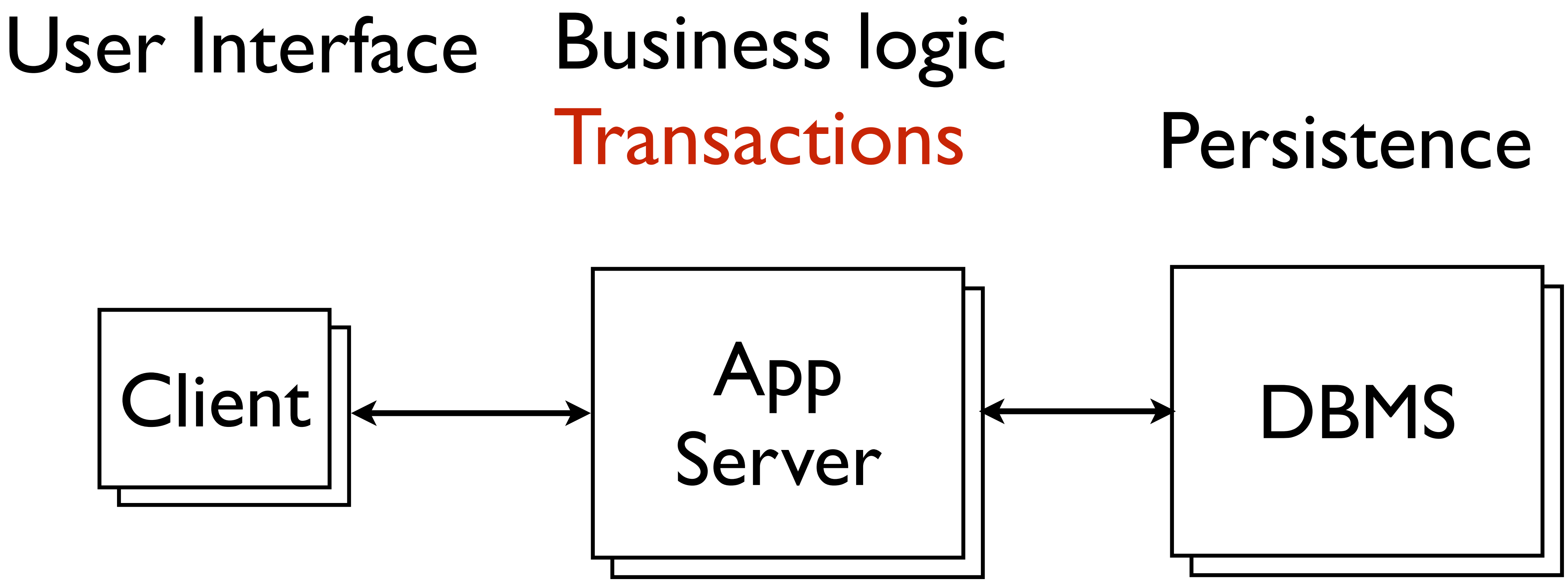
Thesis Statement

"Using an STM-based middleware, it is possible to have both **strong consistency** and **better performance**, for the typical workloads of enterprise applications."


Client
App
Server
DBMS


User Interface
Business logic
Transactions
Persistence
Client
App
Server
DBMS


User Interface
Business logic
Transactions
Persistence
Client
App
Server
DBMS


User Interface
Business logic
Transactions
Persistence
Client
App
Server
DBMS

Desired Properties

Desired Properties

- Strong consistency

Desired Properties

- Strong consistency
- Performance

Desired Properties

- Strong consistency
- Performance
- DB-independent consistency

Desired Properties

- Strong consistency
- Performance
- DB-independent consistency
- Transparent persistence

Desired Properties

- Strong consistency
- Performance
- DB-independent consistency
- Transparent persistence
- Support clustering

Key Elements

Key Elements

- STM-based transactions

Key Elements

- STM-based transactions
- DB abstraction

Key Elements

- STM-based transactions
- DB abstraction
- Add persistence to STM

Key Elements

- STM-based transactions
- DB abstraction
- Add persistence to STM
- STM-aware cache

Key Elements

- STM-based transactions
- DB abstraction
- Add persistence to STM
- STM-aware cache
- Distributed synchronization protocol

TMM:
Transactional Memory Middleware

TMM:
Transactional Memory Middleware

1. STM integration

2. Data persistence

3. Clustering

TMM:
Transactional Memory Middleware

1. STM integration
   - Repository interface
   - JVSTM
2. Data persistence

3. Clustering

TMM: Transactional Memory Middleware

1. STM integration
   - Repository interface
   - JVSTM
2. Data persistence
   - Data mapping
   - Commit extension
   - Cache/Identity Map
3. Clustering

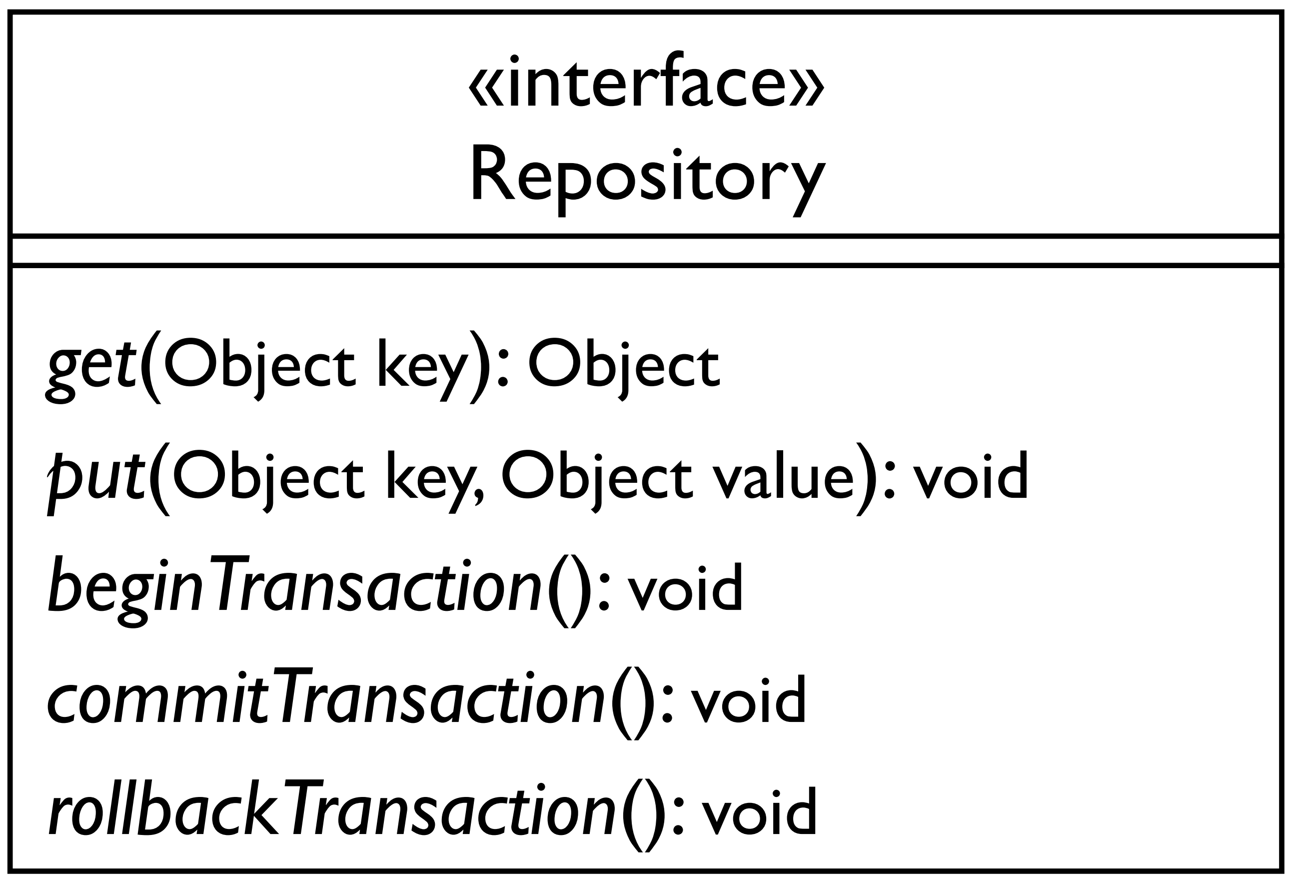
TMM:
Transactional Memory Middleware

1. STM integration
   - Repository interface
   - JVSTM
2. Data persistence
   - Data mapping
   - Commit extension
   - Cache/Identity Map
3. Clustering
   - Distributed group communication
   - Commit extension

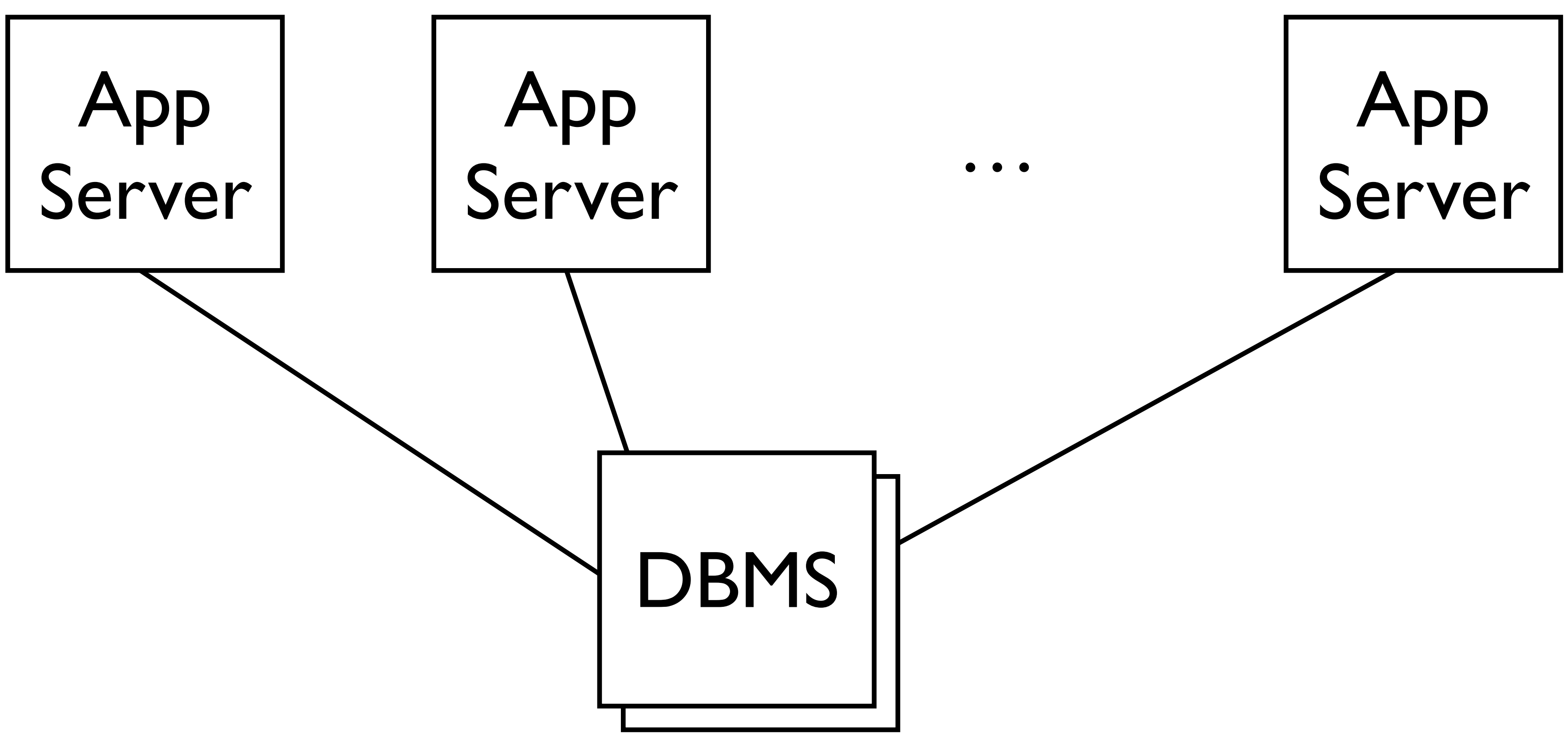
TMM:
Transactional Memory Middleware

1. STM integration
   - Repository interface
   - JVSTM
2. Data persistence
   - Data mapping
   - Commit extension
   - Cache/Identity Map
3. Clustering
   - Distributed group communication
   - Commit extension

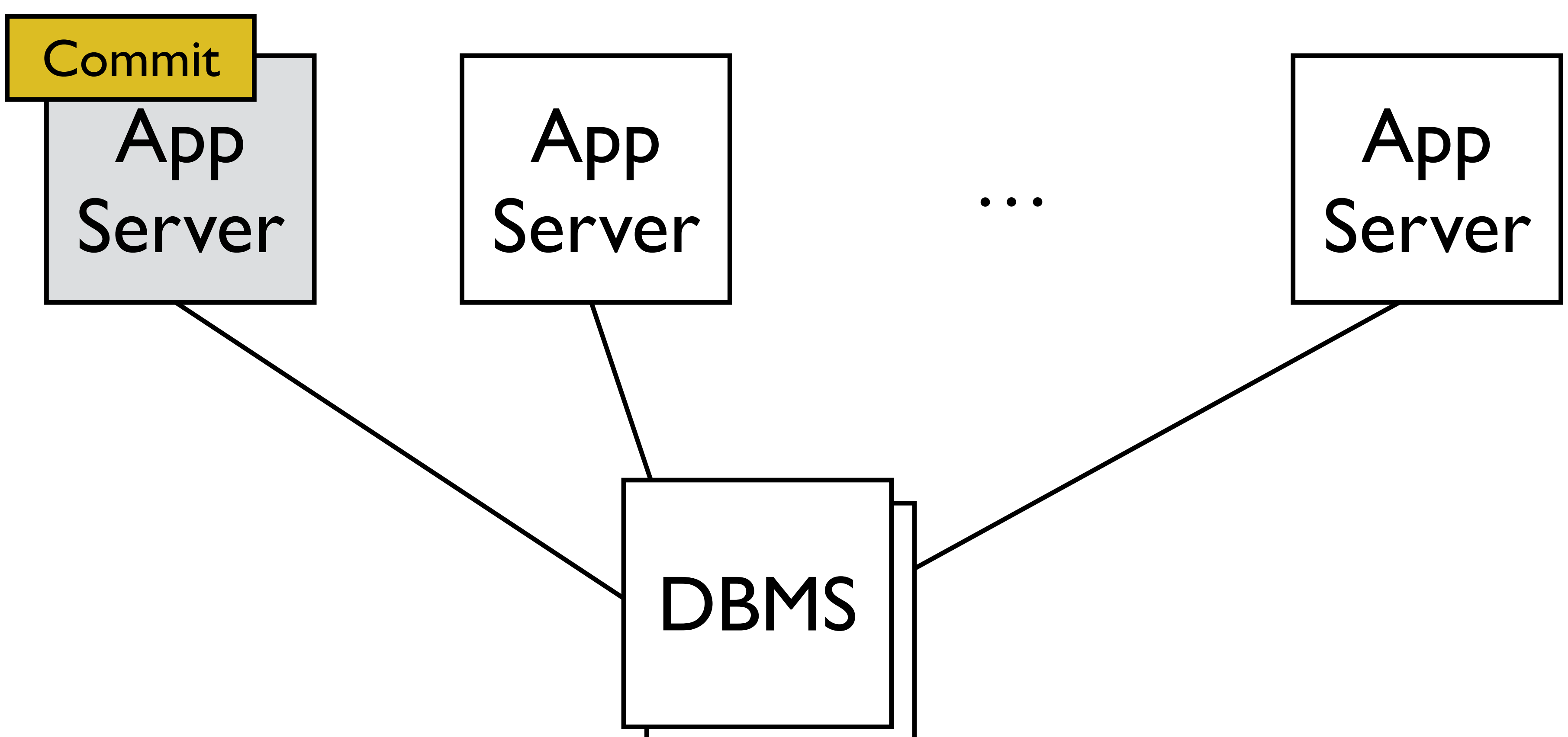
Minimal Repository Requirements



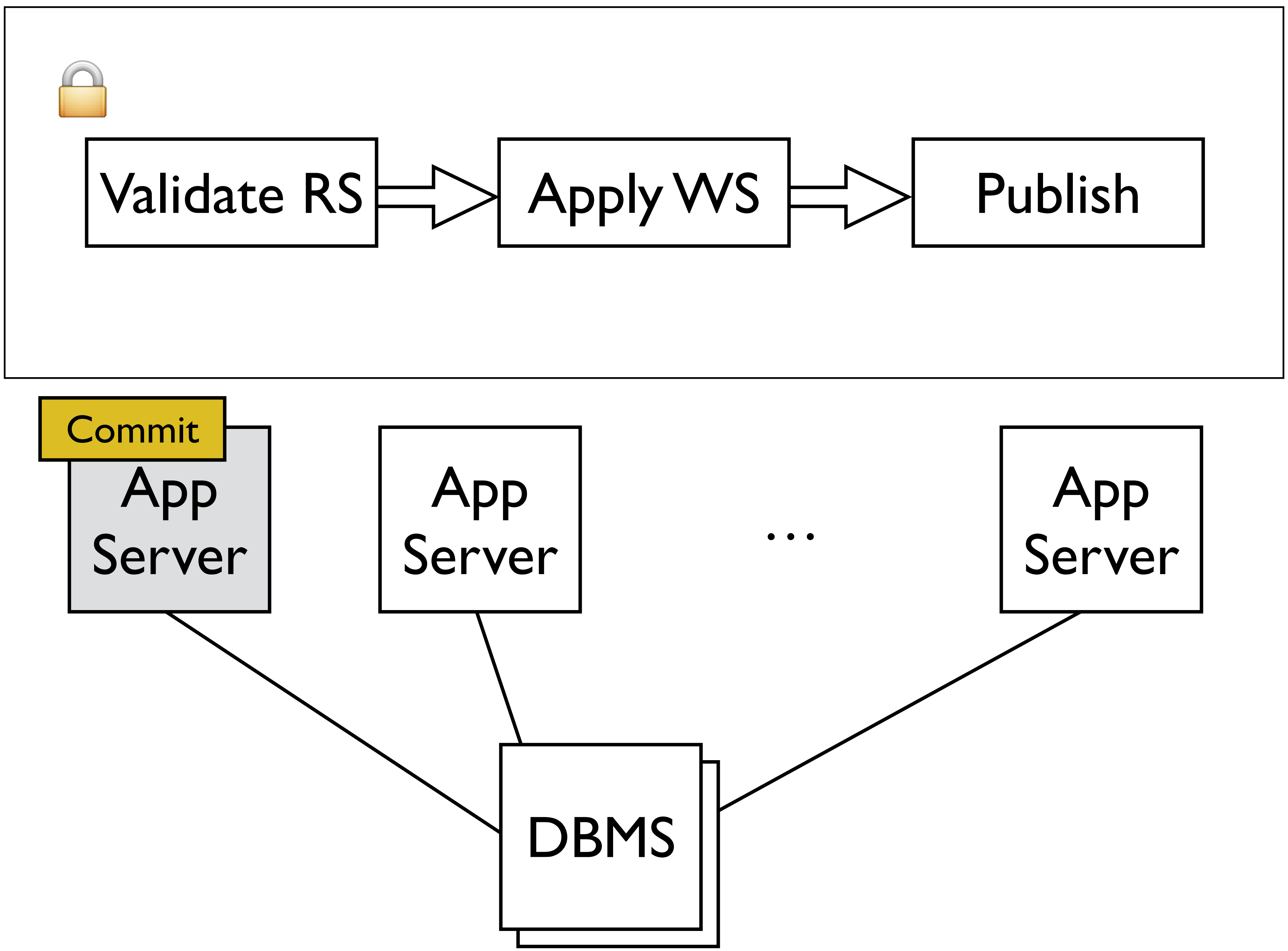
Clustering



Clustering

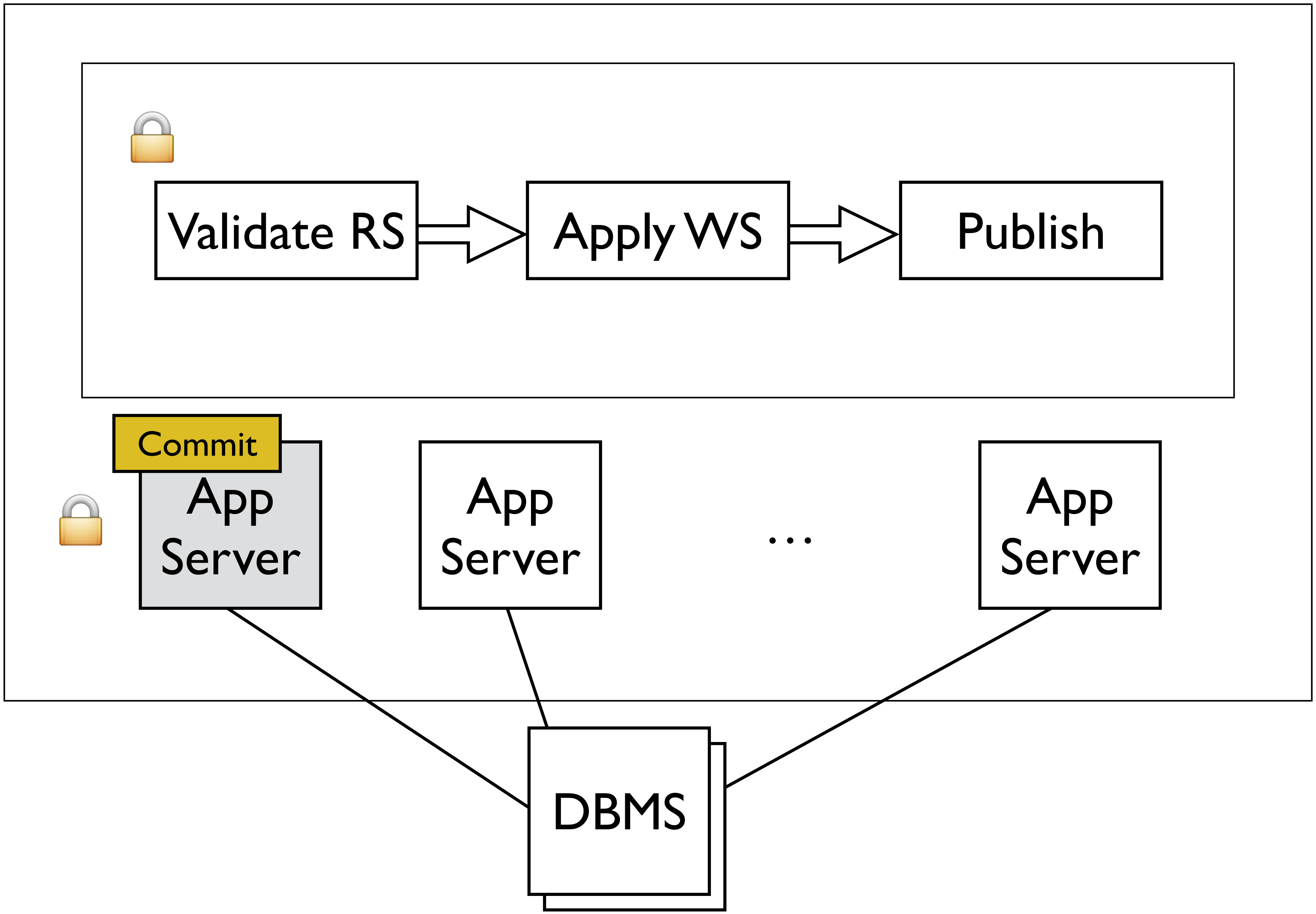


Clustering



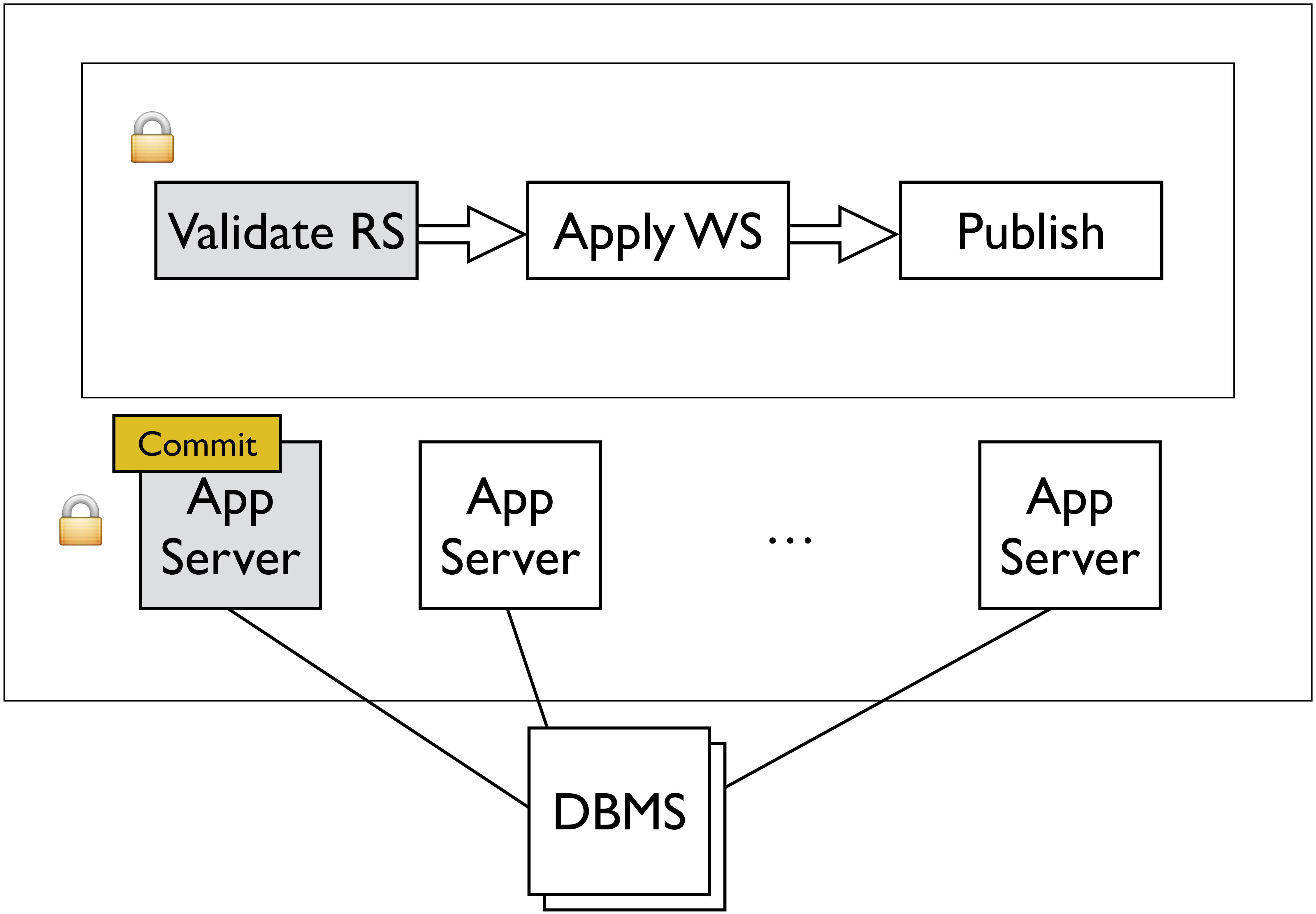
Validate RS
Apply WS
Publish
Commit
App Server
App Server
...
App Server
DBMS

Clustering



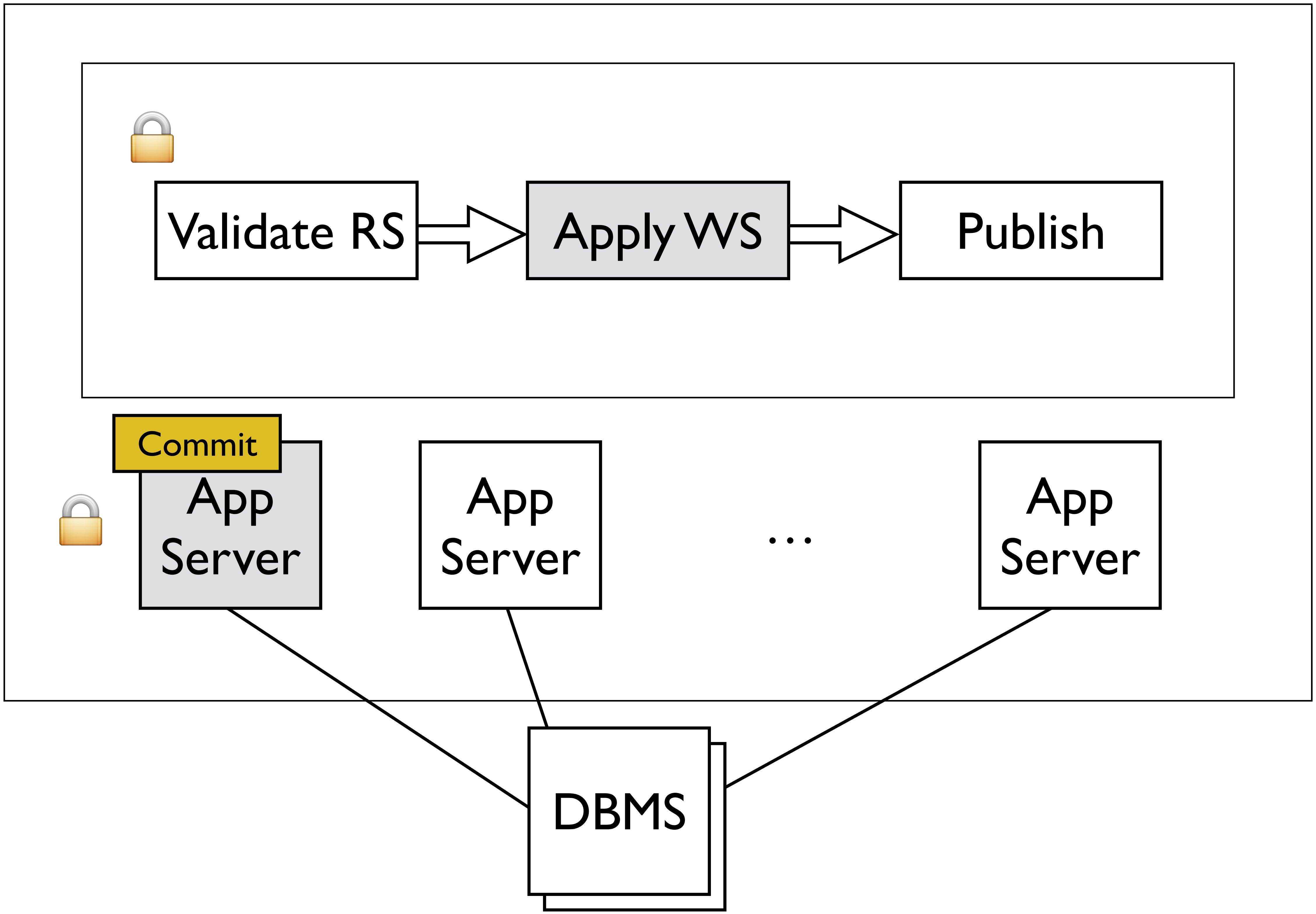
Validate RS
Apply WS
Publish
Commit
App Server
App Server
...
App Server
DBMS

Clustering

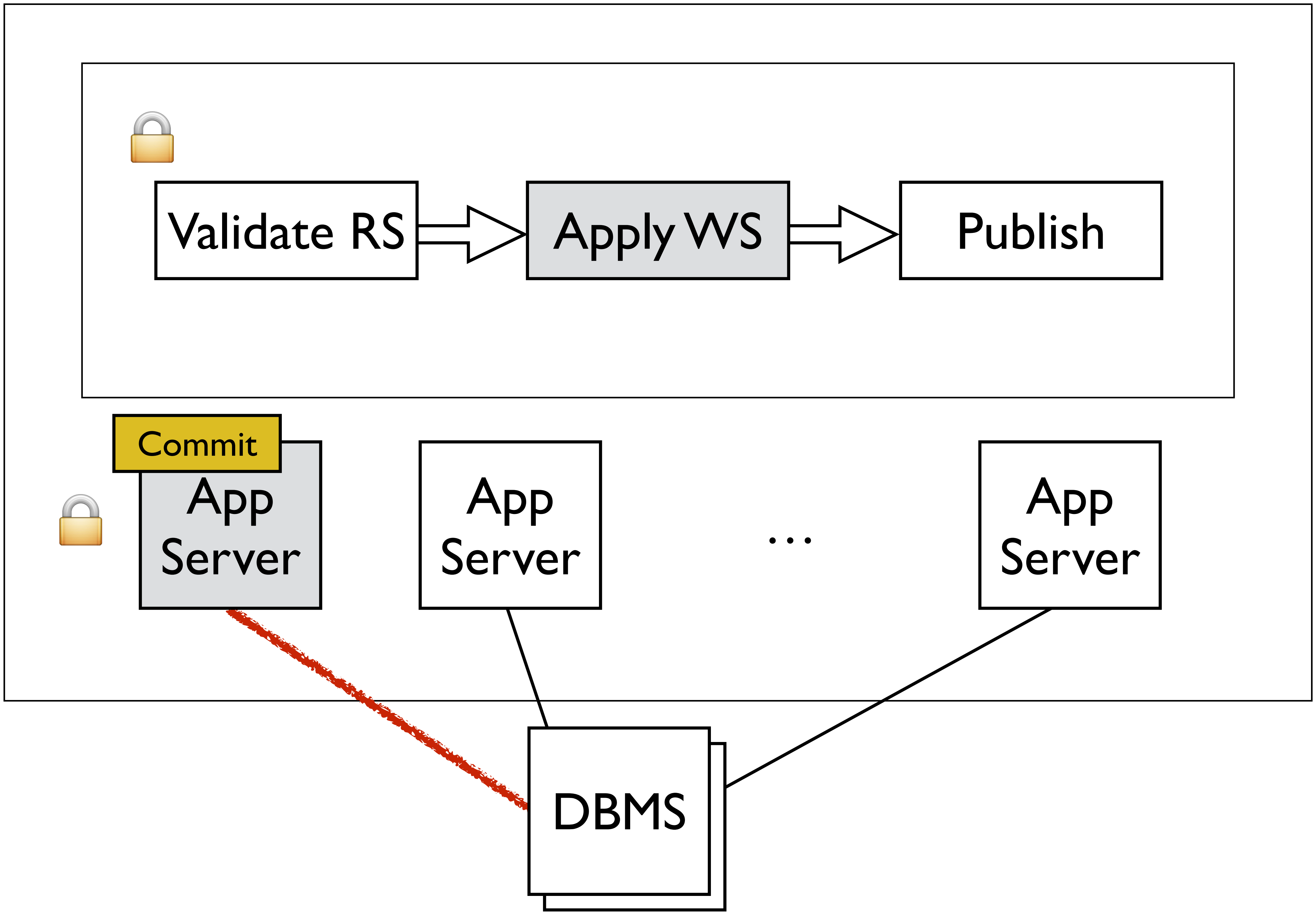


Validate RS
Apply WS
Publish
Commit
App Server
App Server
...
App Server
DBMS

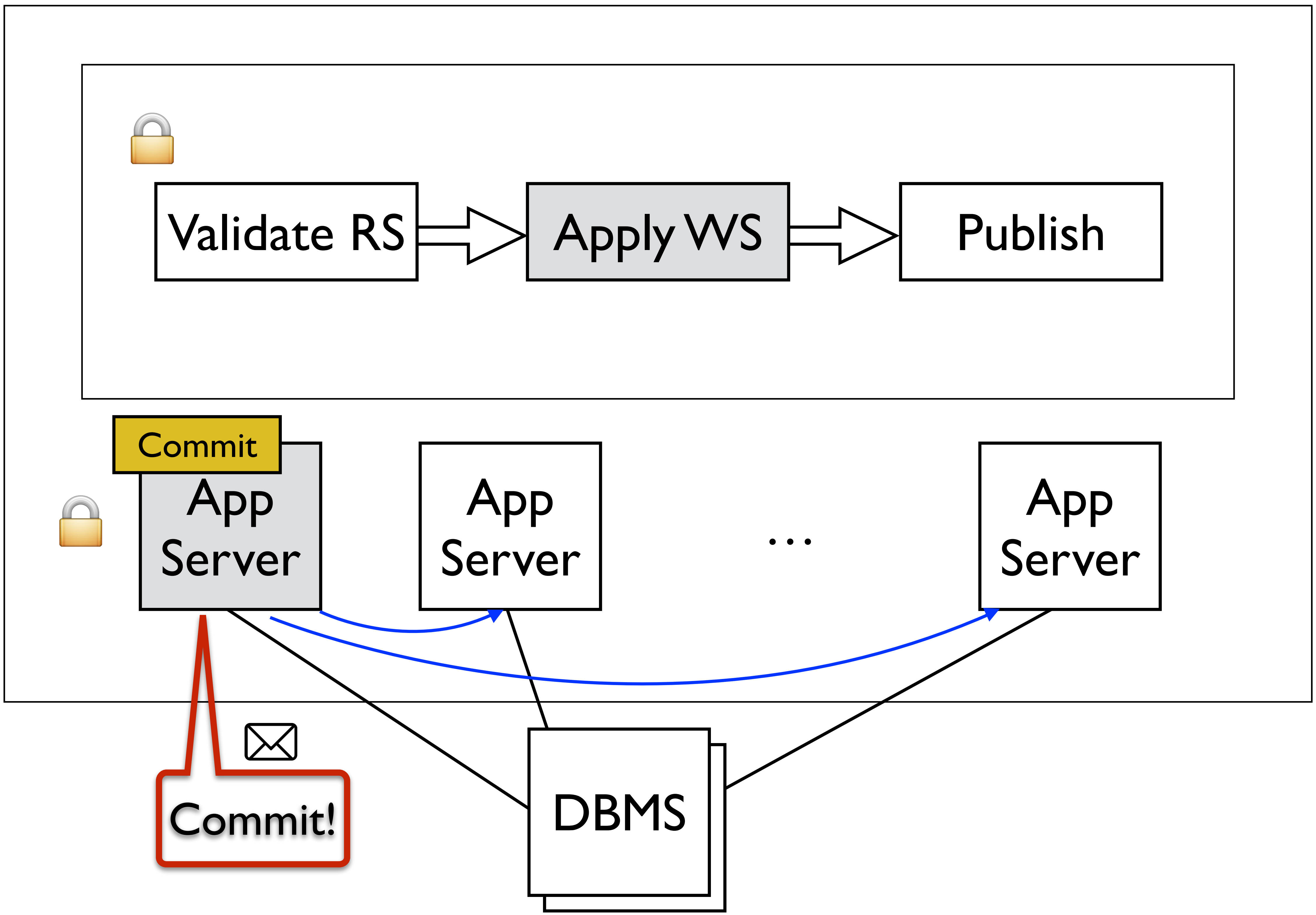
Clustering


Validate RS
Apply WS
Publish
Commit
App Server
App Server
...
App Server
DBMS

Clustering

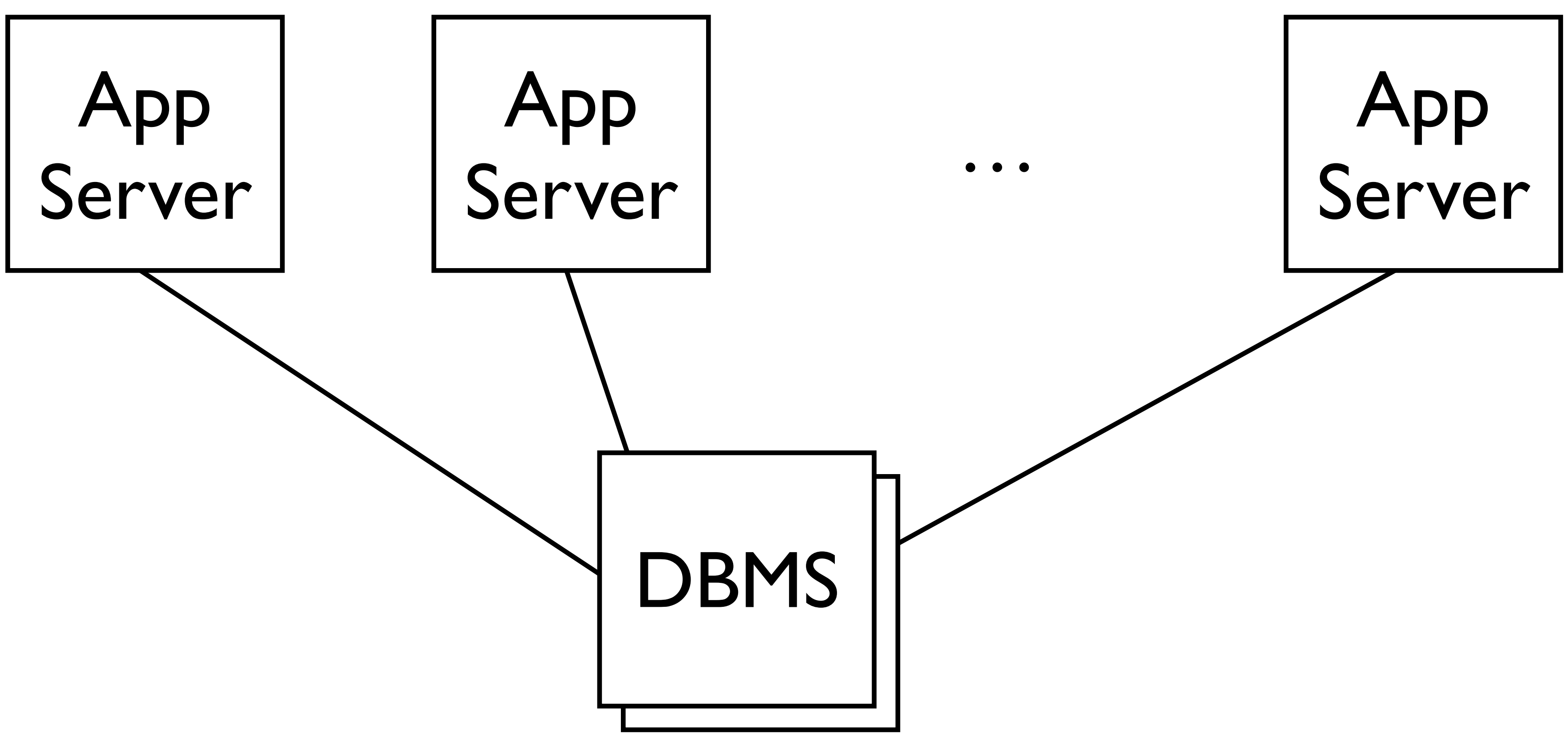

Validate RS
Apply WS
Publish
Commit
App Server
App Server
...
App Server
DBMS

Clustering

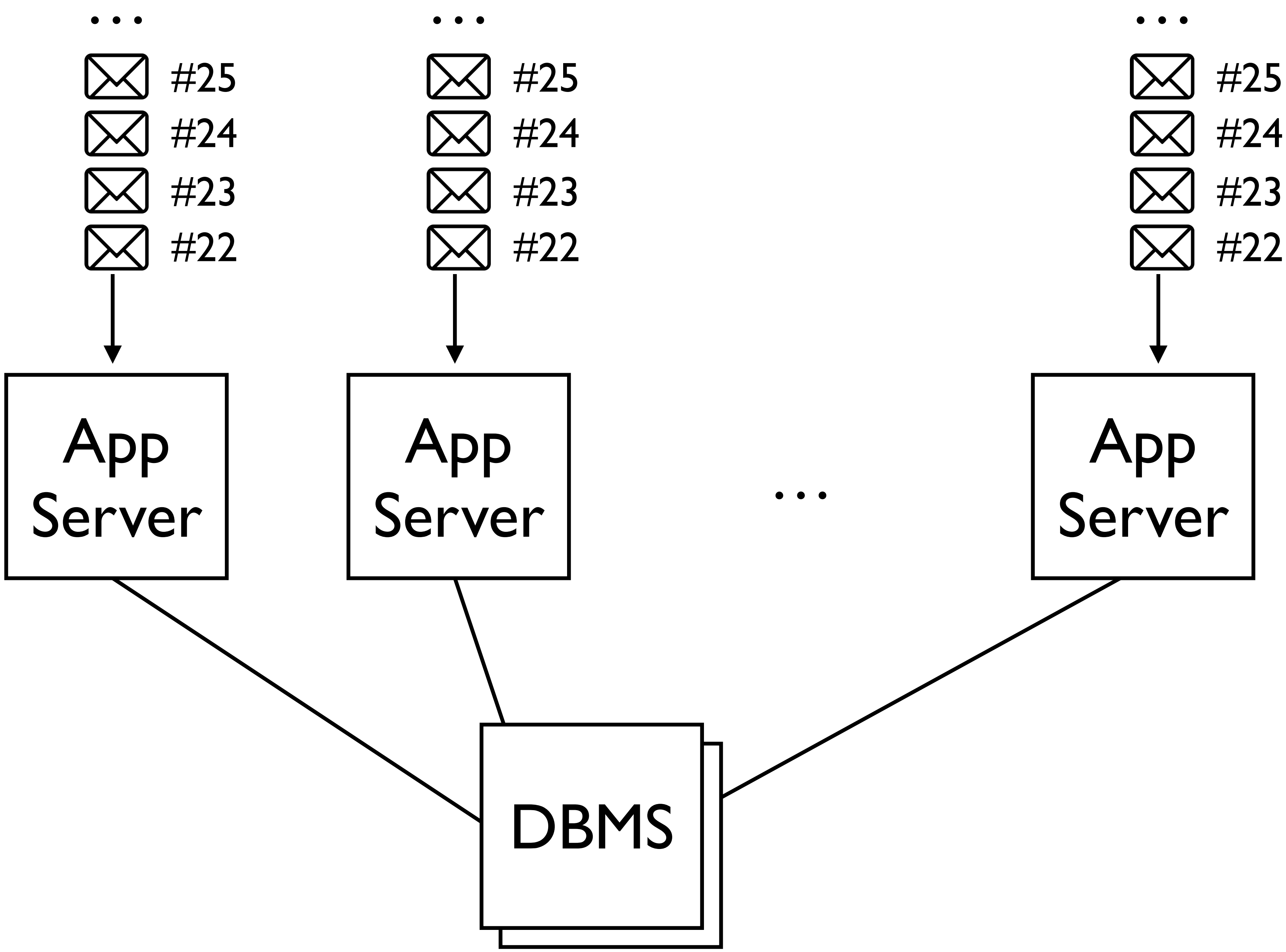


Validate RS
Apply WS
Publish
Commit
App Server
App Server
...
App Server
Commit!
DBMS

Clustering



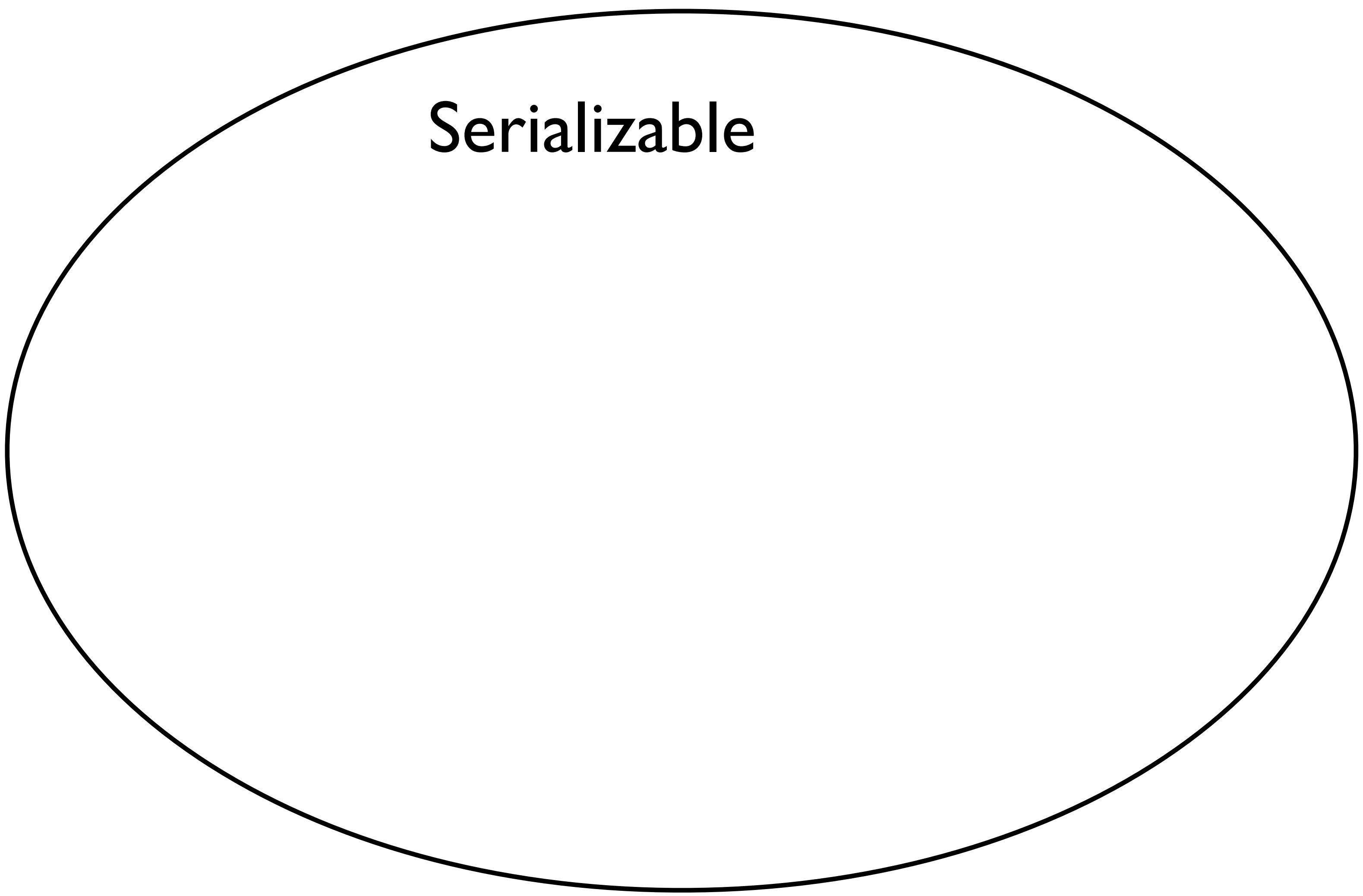
Clustering



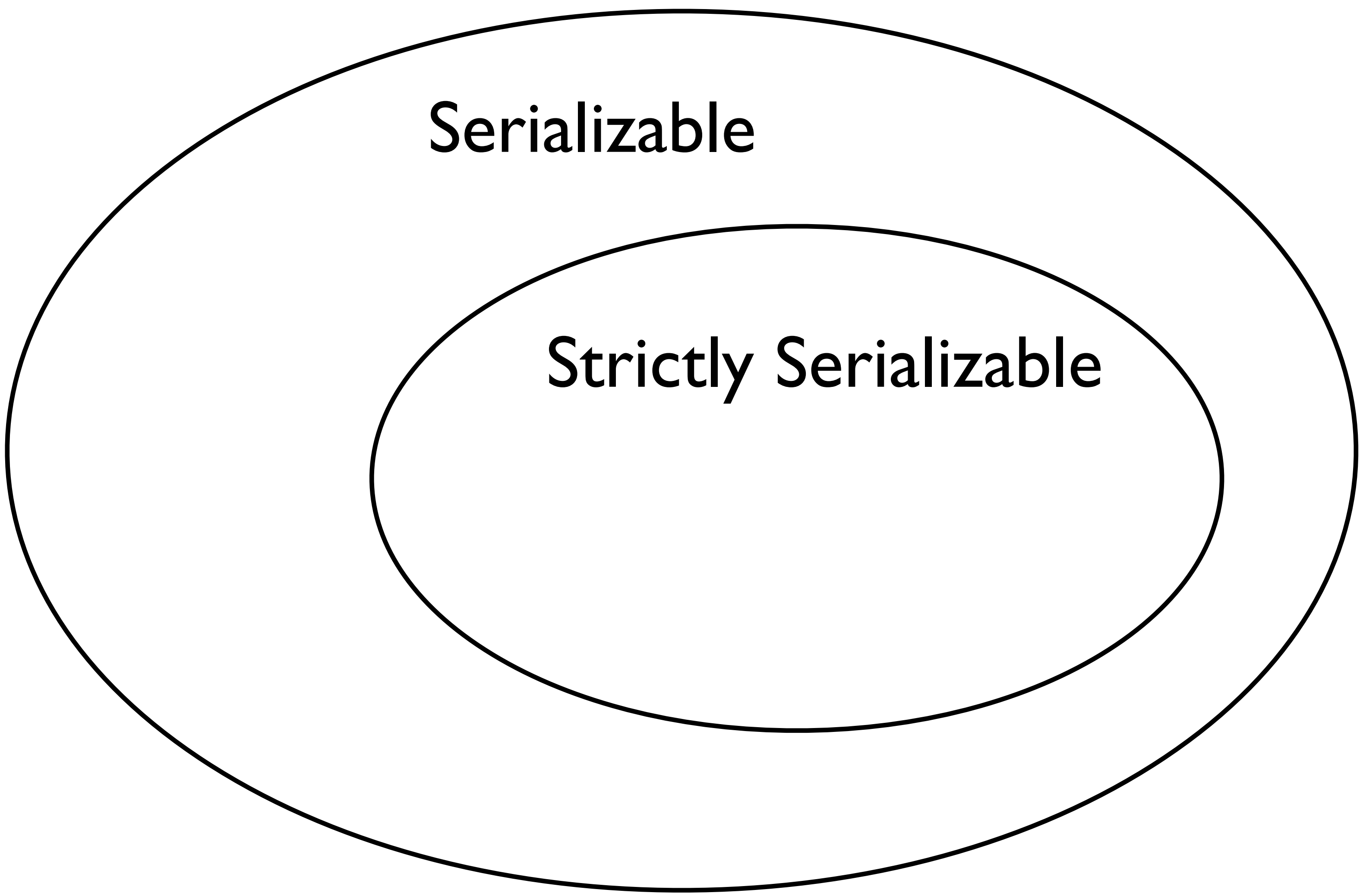
...
#25
#24
#23
#22
...
#25
#24
#23
#22
...
#25
#24
#23
#22
App Server
App Server
...
App Server
DBMS

Consistency Guarantees

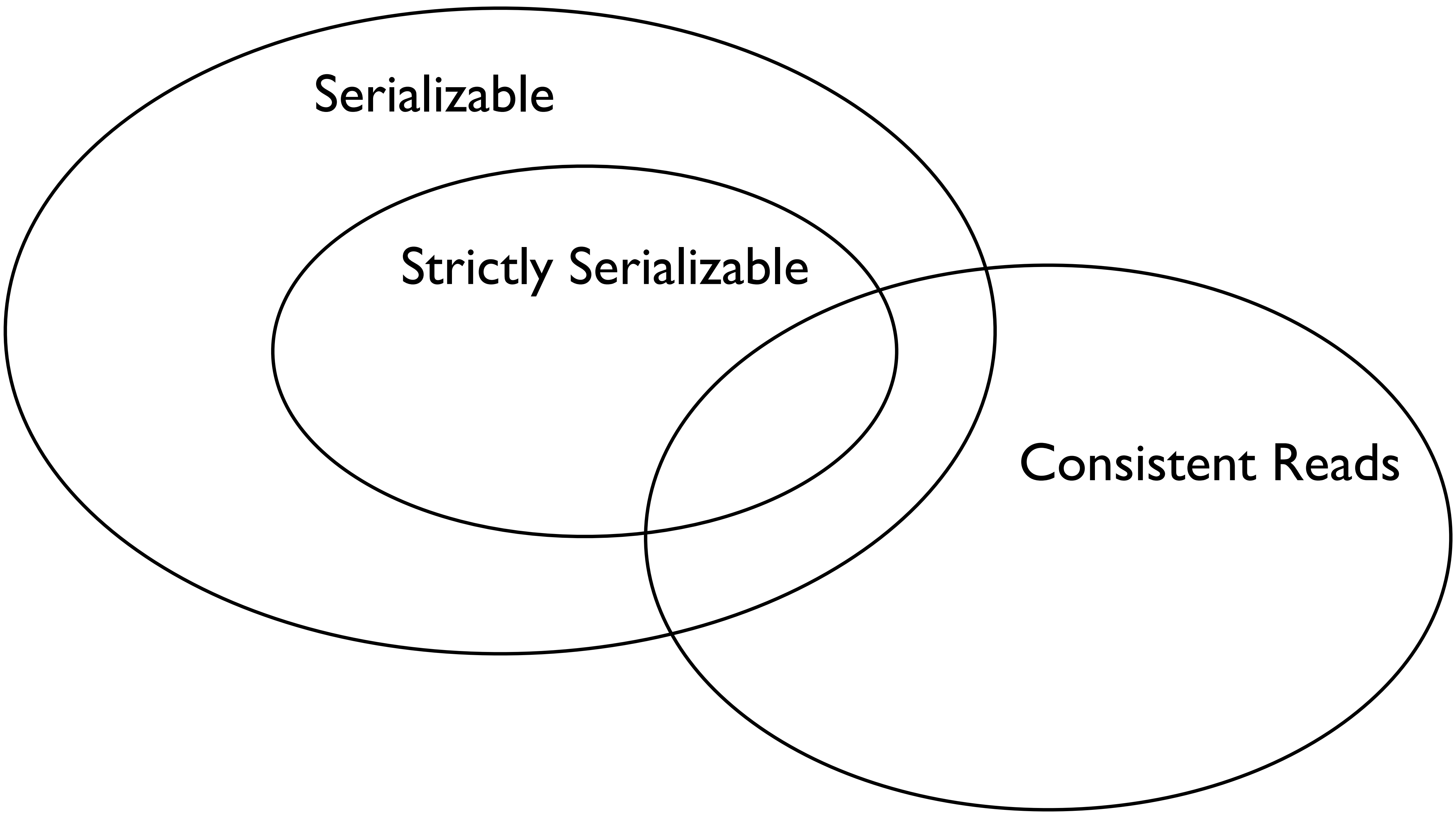
Consistency Guarantees



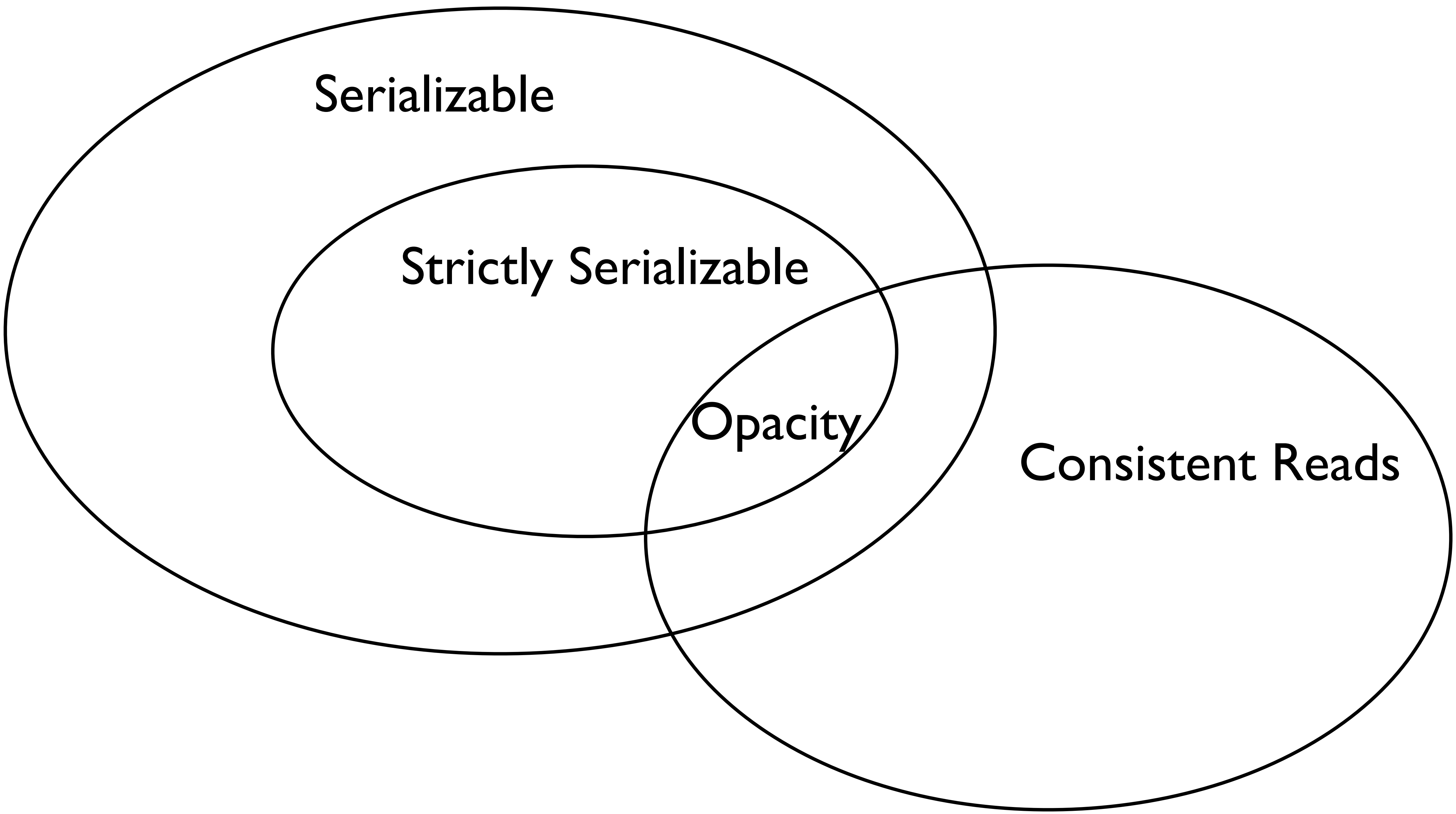
Consistency Guarantees



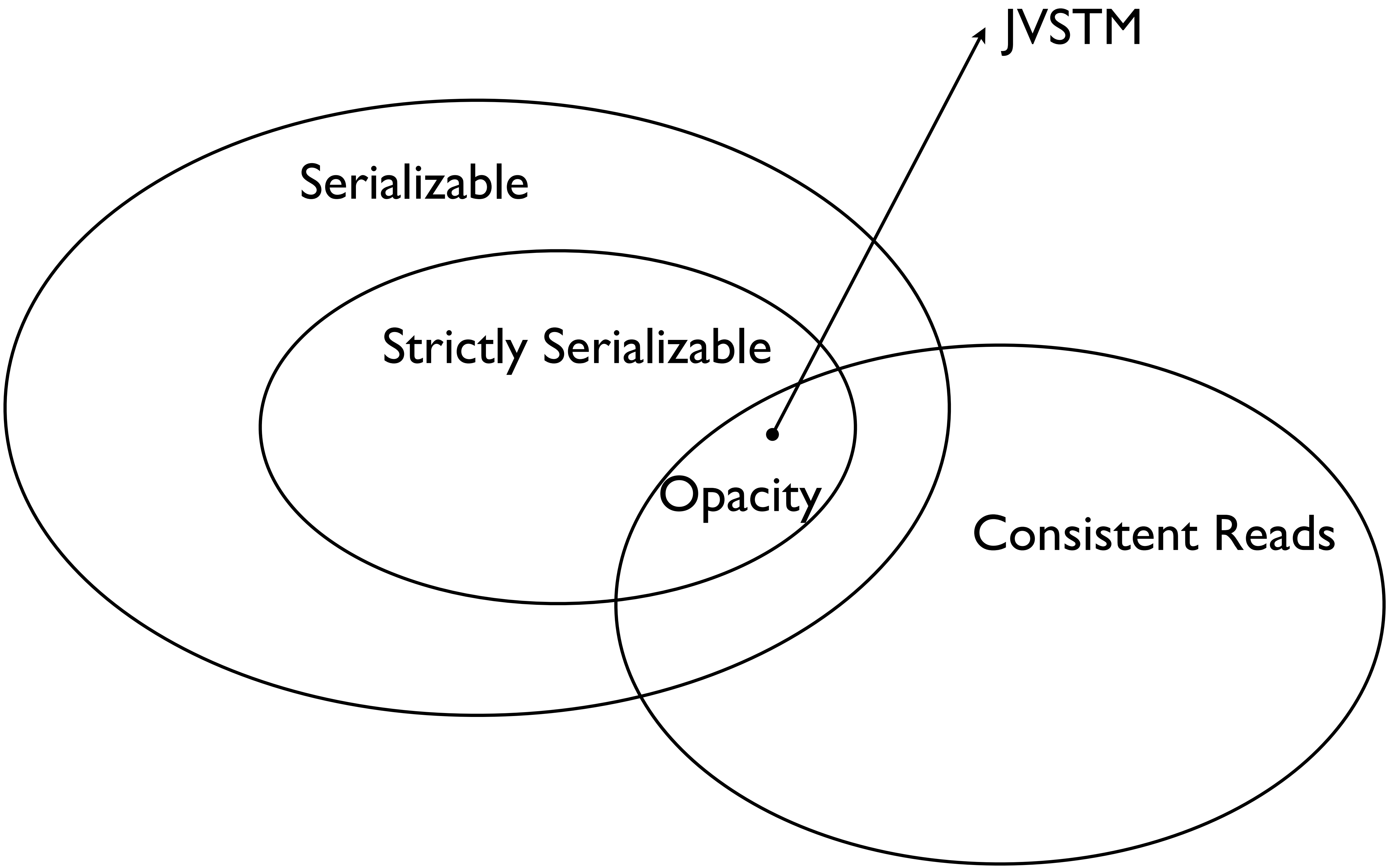
Consistency Guarantees



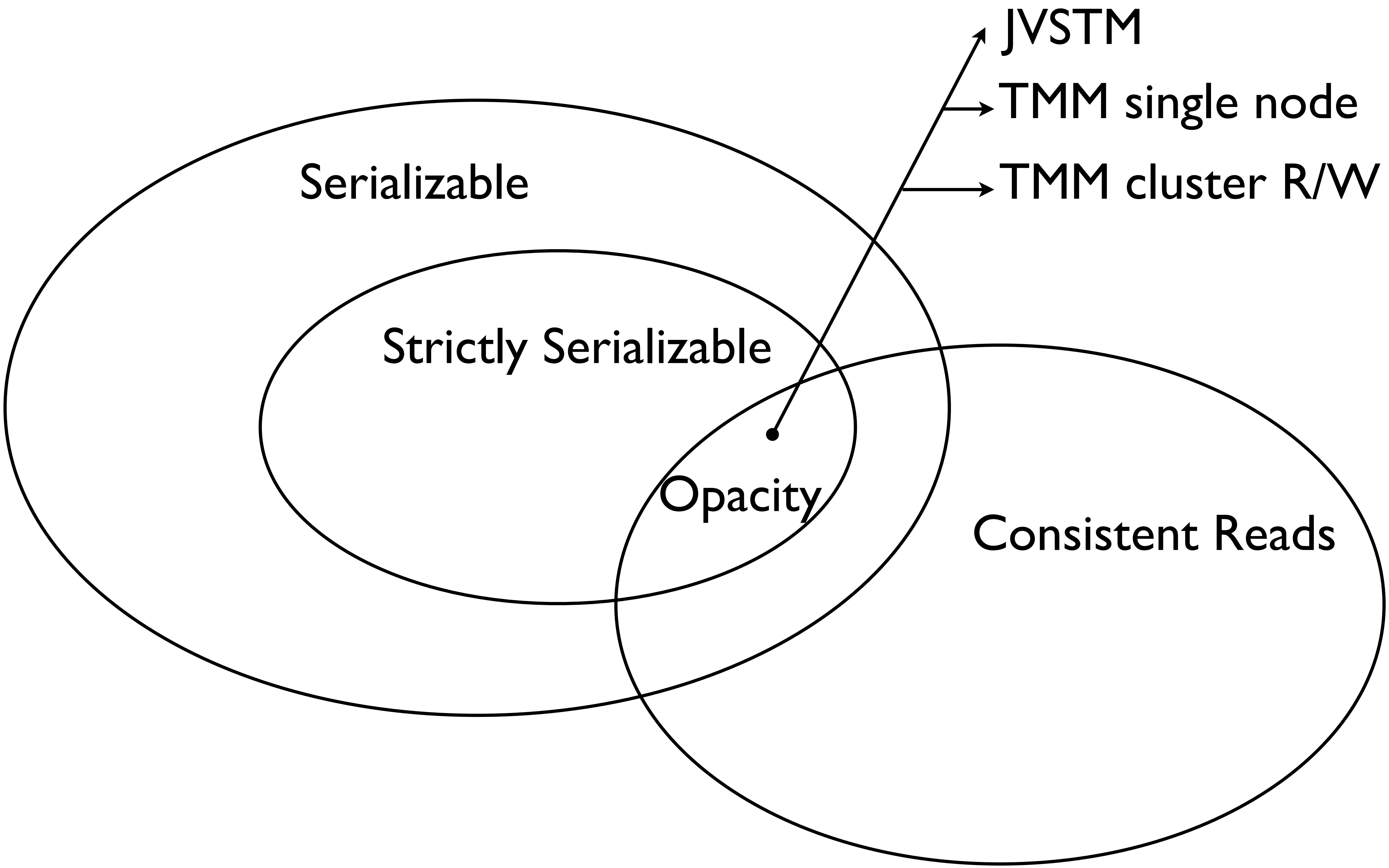
Consistency Guarantees



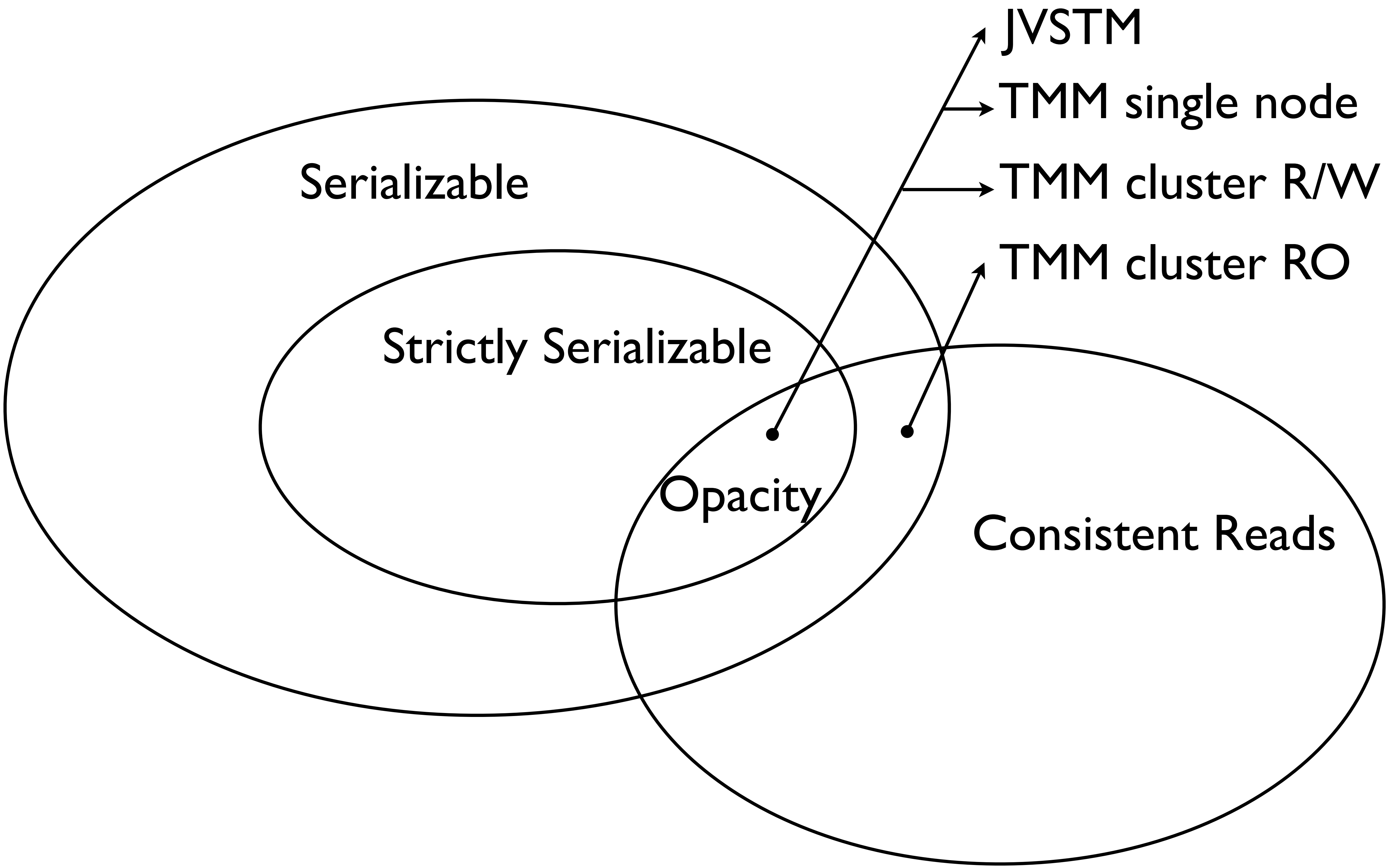
Consistency Guarantees



Consistency Guarantees



Consistency Guarantees



Evaluation

Evaluation

- RadarGun

Evaluation

- RadarGun
- Infinispan, Hazelcast, EHCache

Evaluation

- RadarGun
- Infinispan, Hazelcast, EHCache
- Workloads: 1% to 20% write transactions

Evaluation

- RadarGun
- Infinispan, Hazelcast, EHCache
- Workloads: 1% to 20% write transactions
- Single node

Evaluation

- RadarGun
- Infinispan, Hazelcast, EHCache
- Workloads: 1% to 20% write transactions
- Single node
- Cluster: 2 to 12 nodes

Results

Results

- Outstanding single-node performance

    From 2x to 100x faster

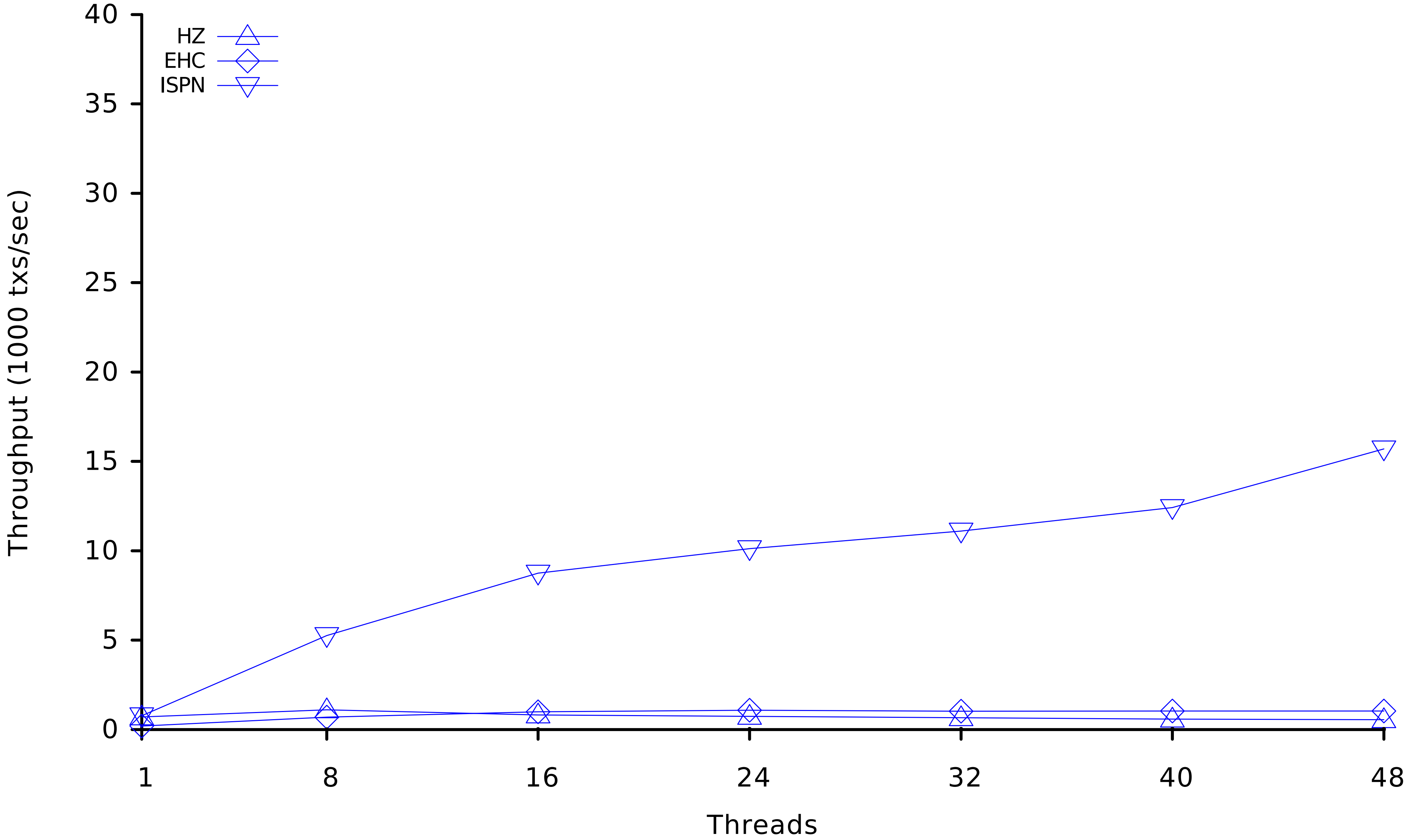
Results

- Outstanding single-node performance

  From 2x to 100x faster

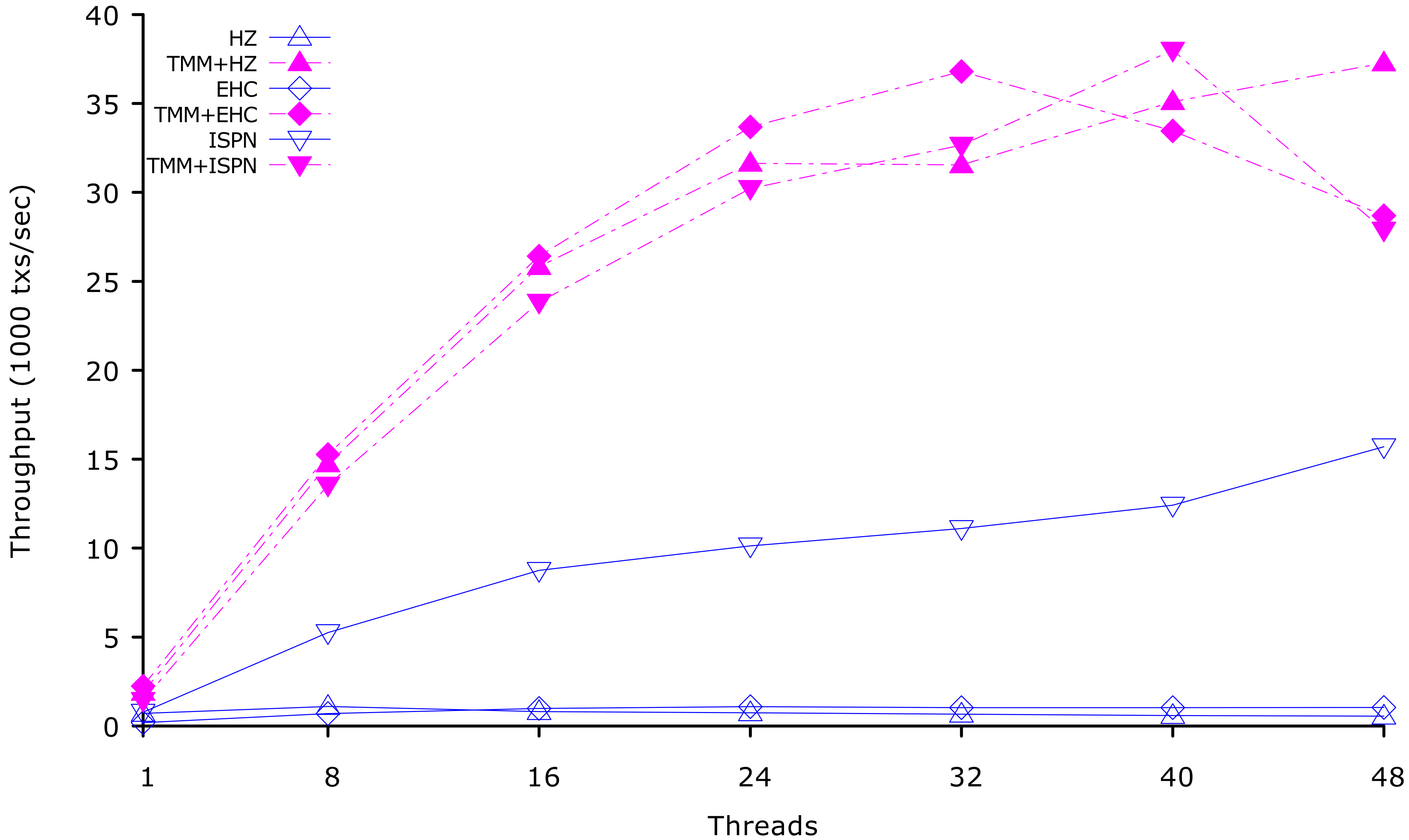
- Poor scalability in cluster

Results (Single node)

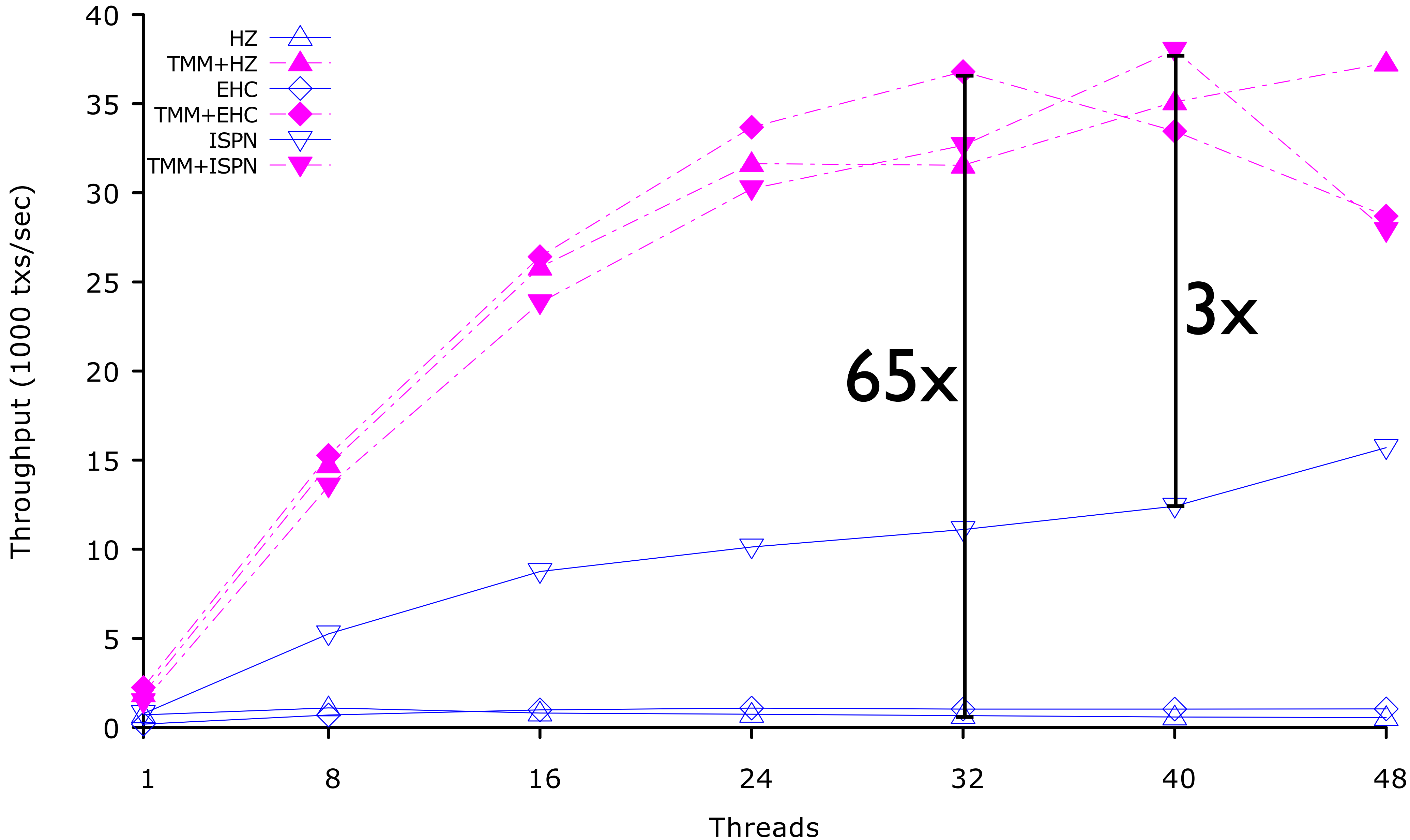


Results (Single node)

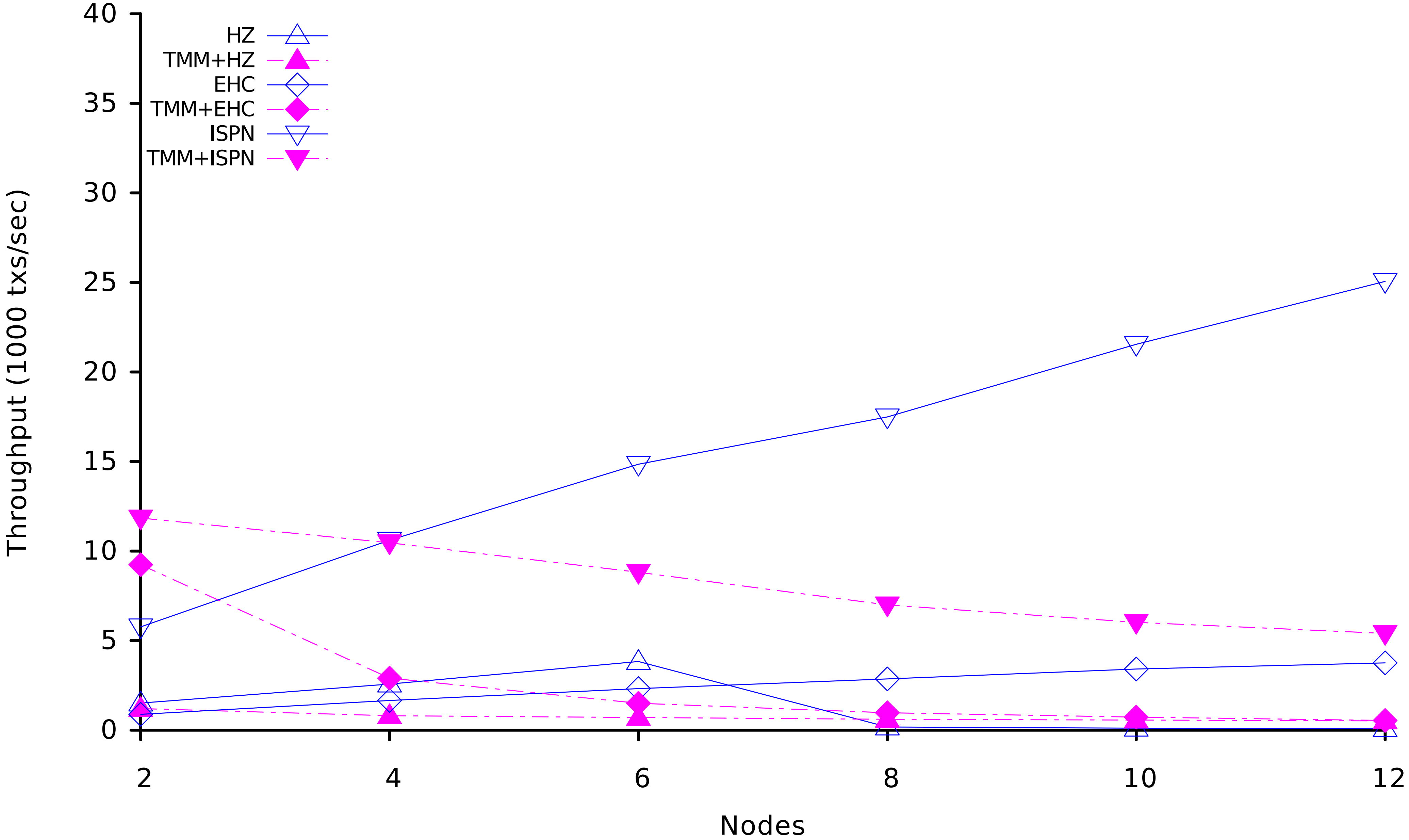
TMM vs. TDGs



Results (Single node)
TMM vs. TDGs



Results (Clustered)



Scalability Bottleneck

Coarse Locks Do Not Scale

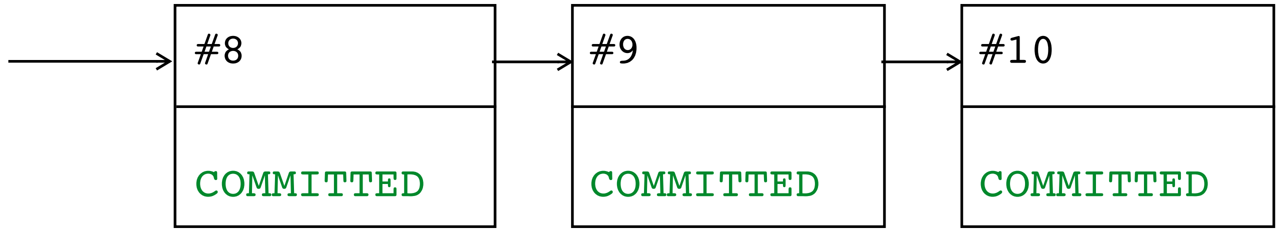
Lock-free Commit Design

Lock-free Commit Design

- Preserve original stages

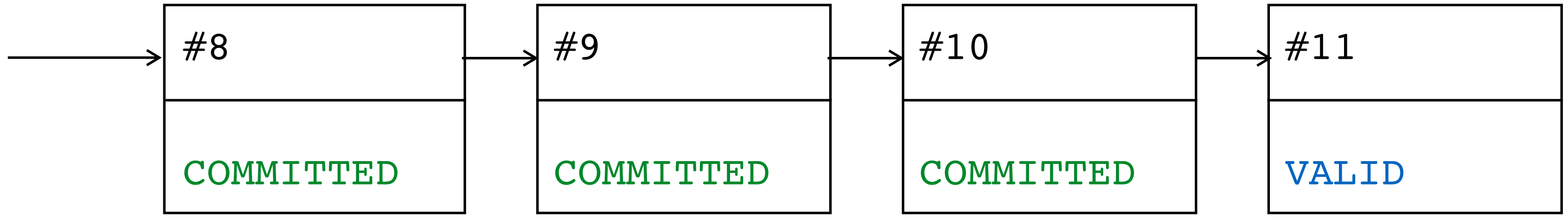
Lock-free Commit Design

- Concurrent validation



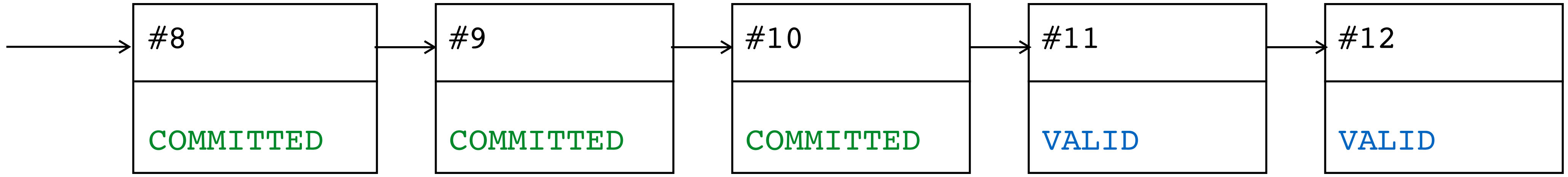
Lock-free Commit Design

- Concurrent validation



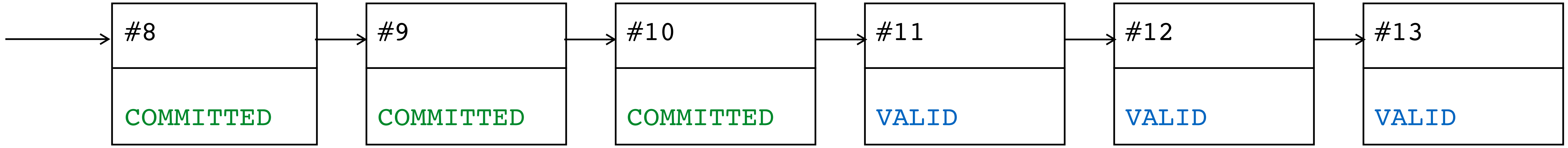
Lock-free Commit Design

- Concurrent validation

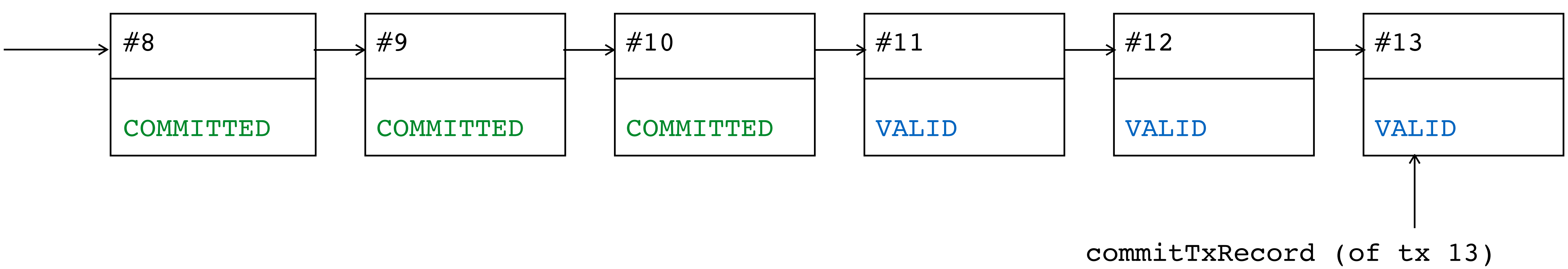


Lock-free Commit Design

- Concurrent validation

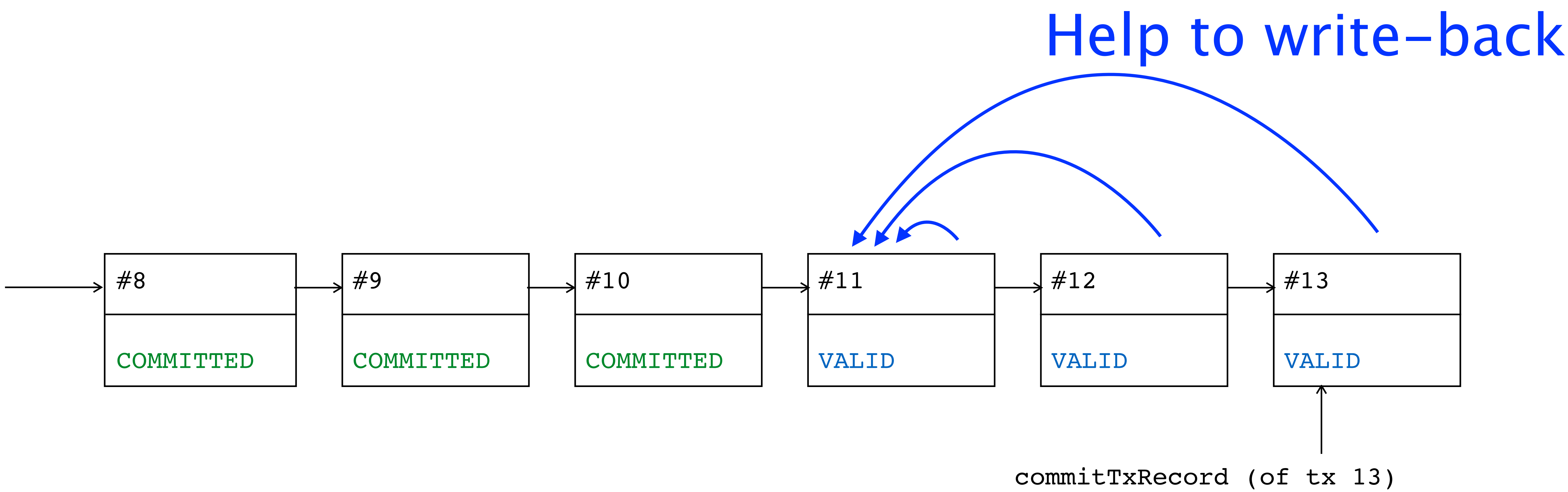


Lock-free Commit Design



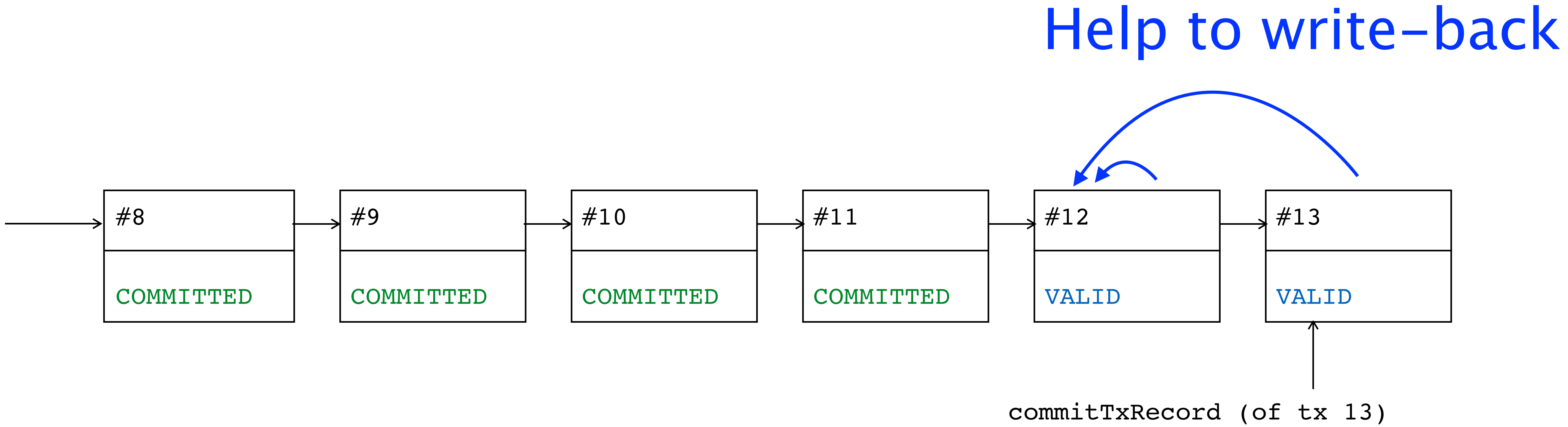
Lock-free Commit Design

- Thread helping



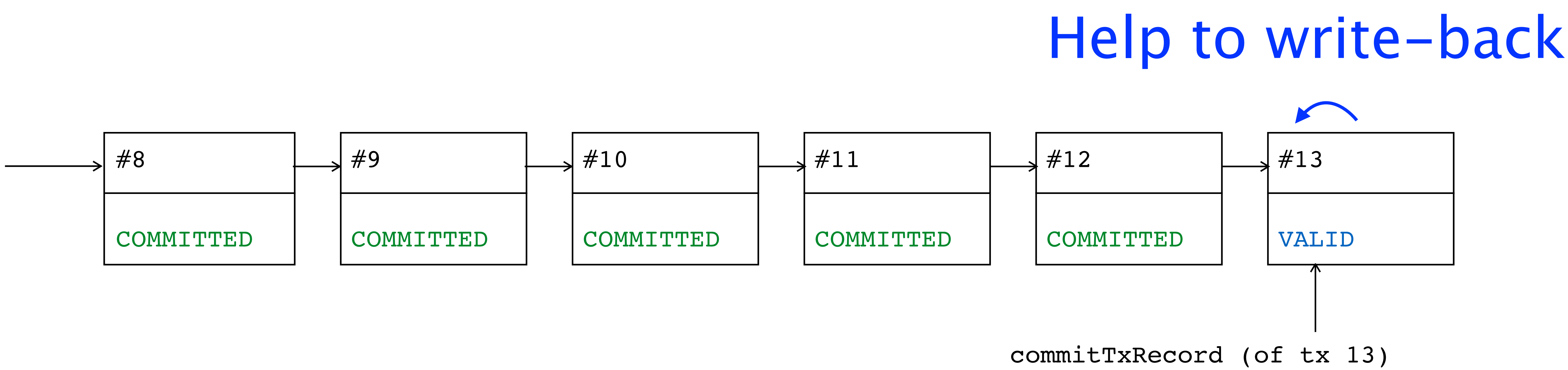
Lock-free Commit Design

- Thread helping



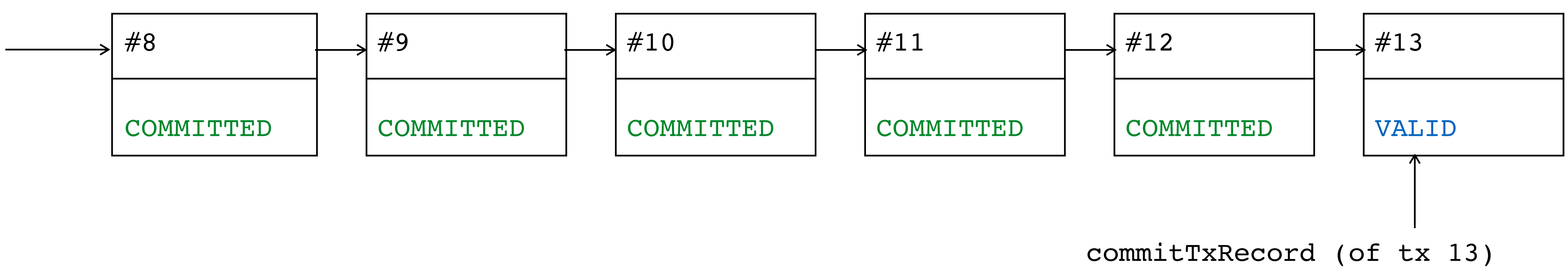
Lock-free Commit Design

- Thread helping



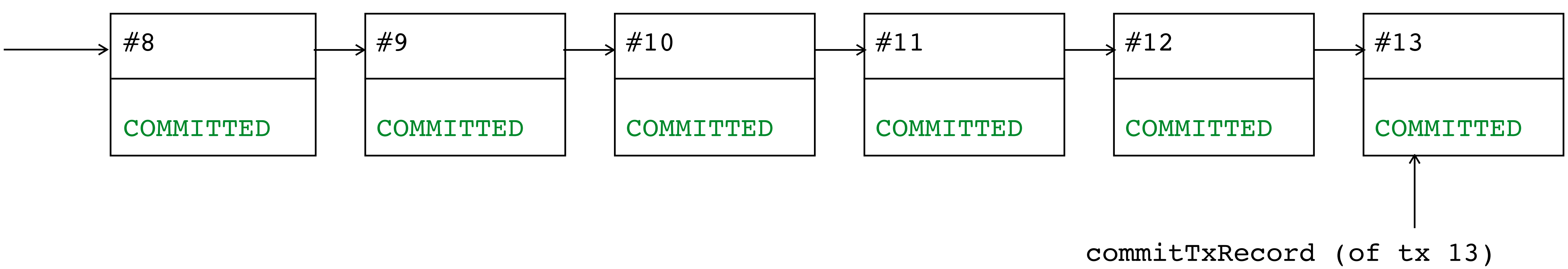
Lock-free Commit Design

- Thread helping



Lock-free Commit Design

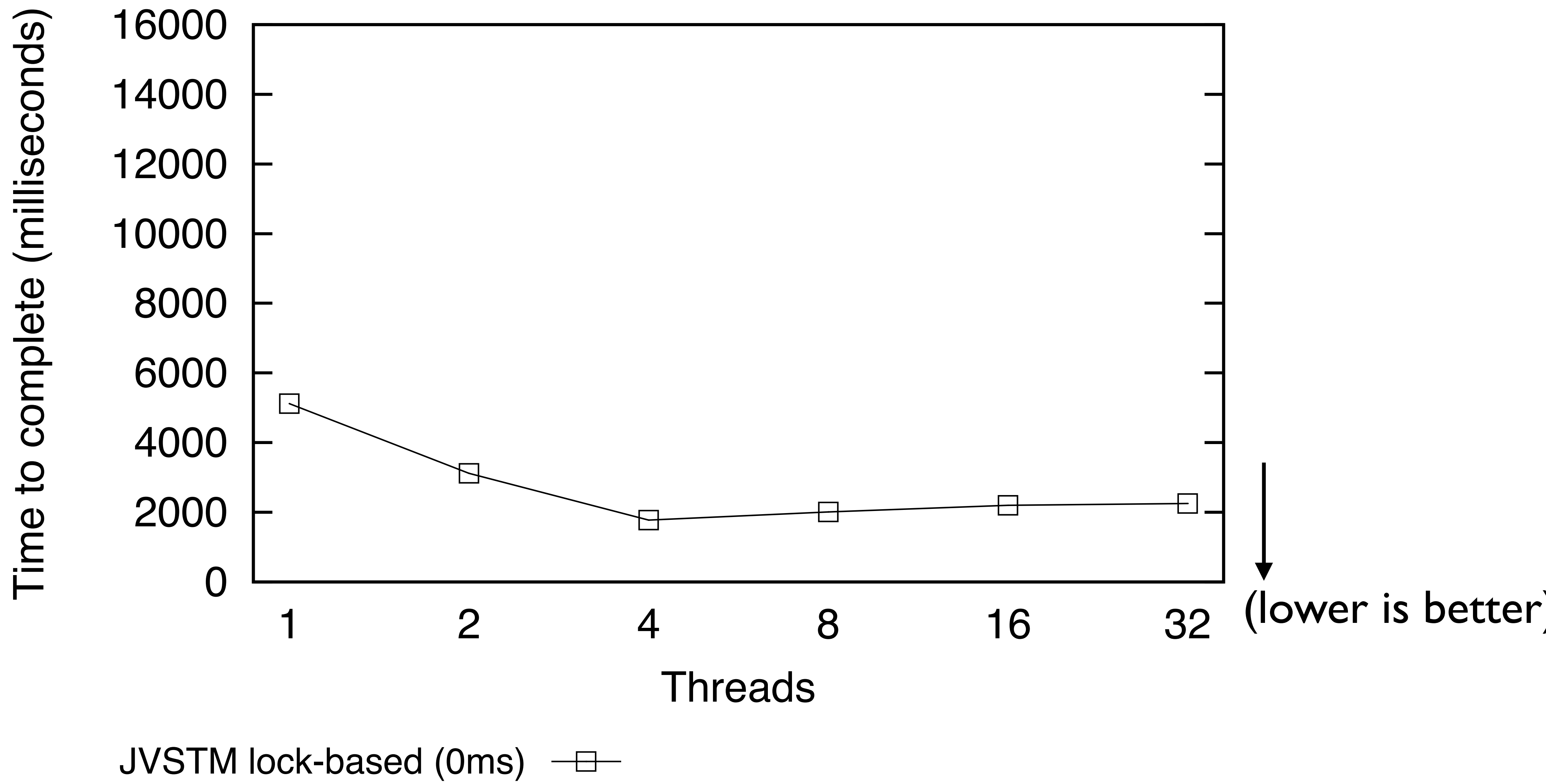
- Linearization point



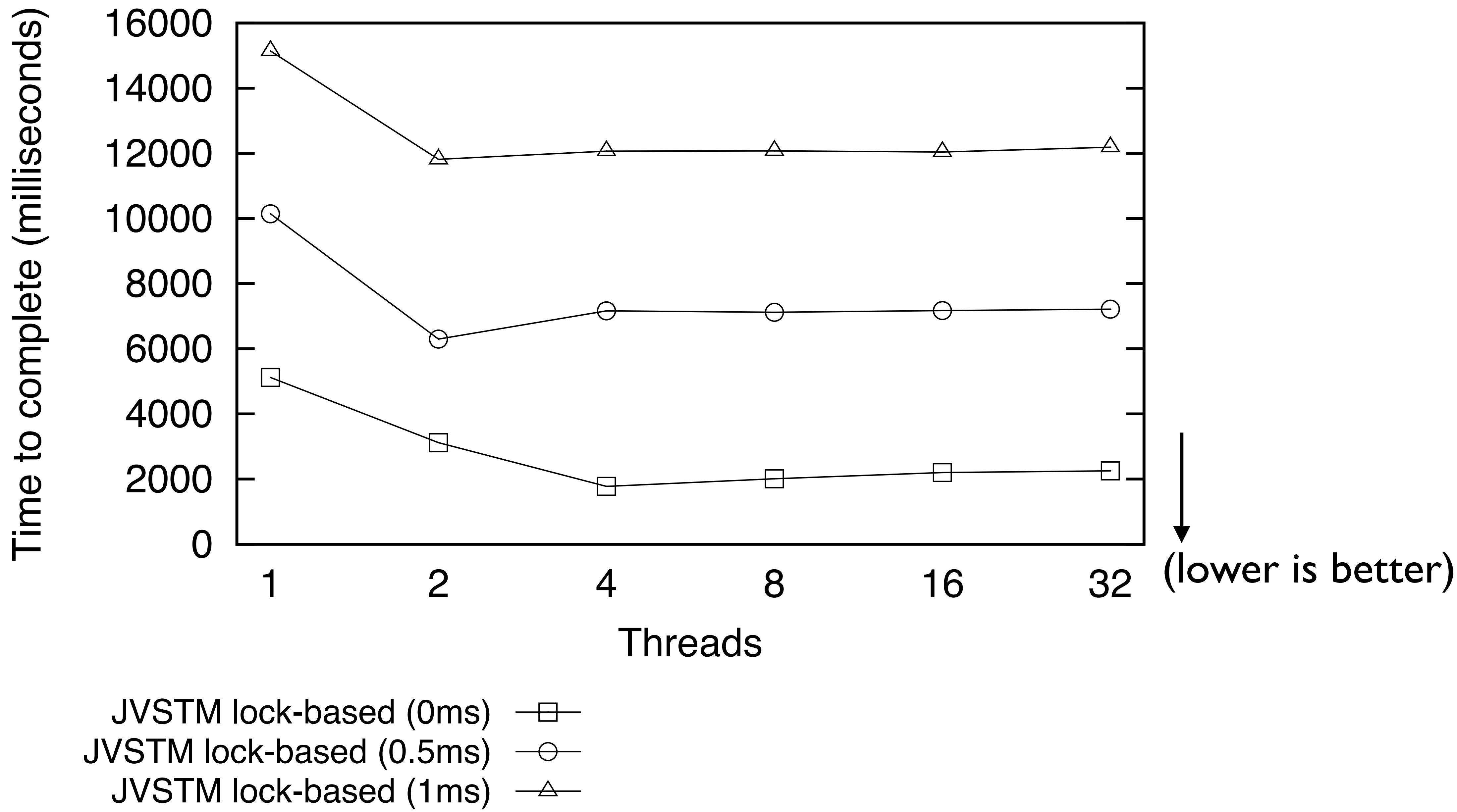
Results

- Full lock-free STM
- Reads unaffected
- More complex algorithm
- Scalable design
- Comparable with top-performing STMs

Results

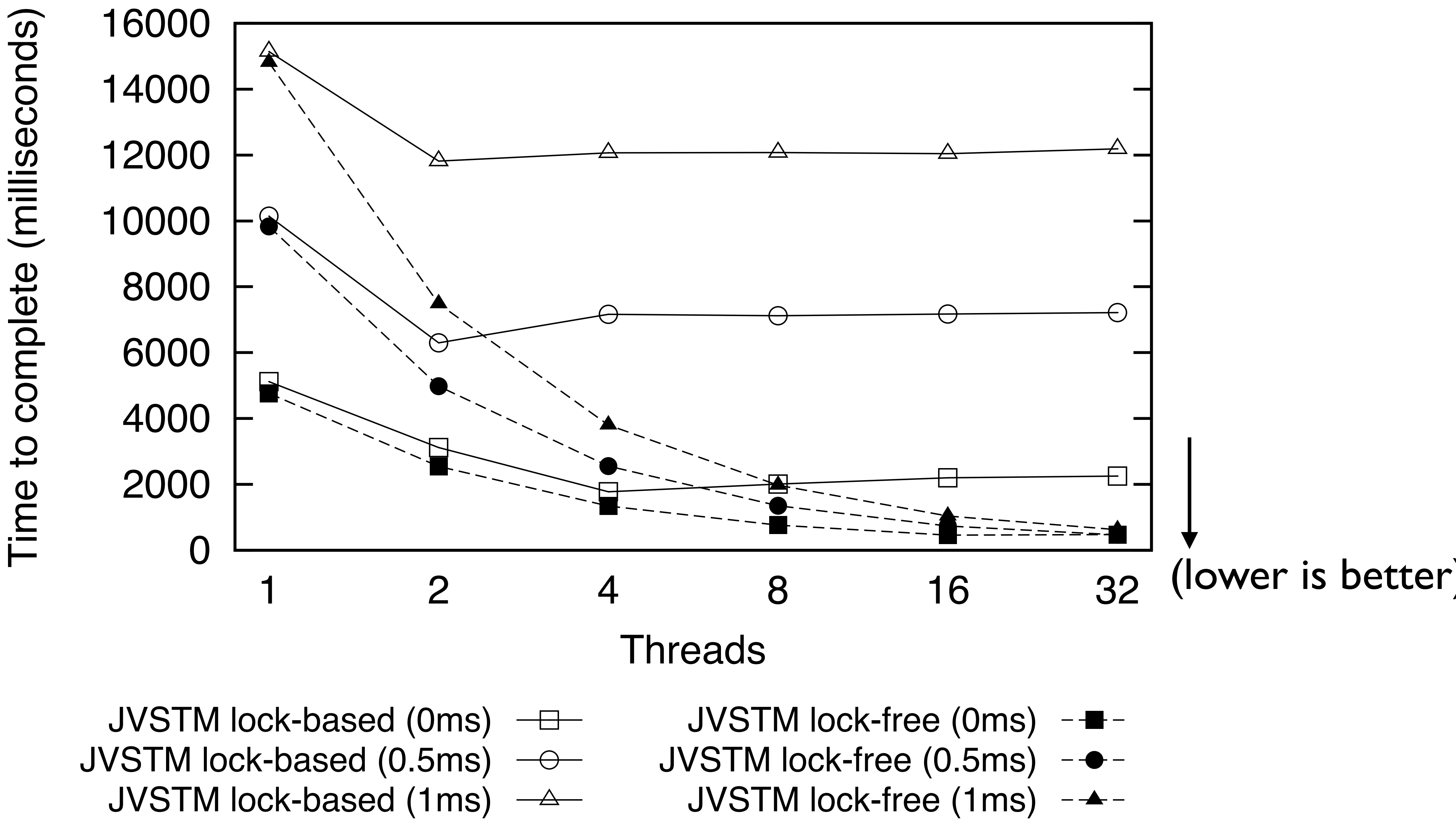

Time to complete (milliseconds)
16000
14000
12000
10000
8000
6000
4000
2000
0
1
2
4
8
16
32
Threads
(lower is better)
JVSTM lock-based (0ms)

Results



Time to complete (milliseconds)
16000
14000
12000
10000
8000
6000
4000
2000
0
1
2
4
8
16
32
Threads
(lower is better)
JVSTM lock-based (0ms)
JVSTM lock-based (0.5ms)
JVSTM lock-based (1ms)

Results


16000
14000
12000
10000
8000
6000
4000
2000
0
Time to complete (milliseconds)
1
2
4
8
16
32
Threads
(lower is better)
JVSTM lock-based (0ms)
JVSTM lock-based (0.5ms)
JVSTM lock-based (1ms)
JVSTM lock-free (0ms)
JVSTM lock-free (0.5ms)
JVSTM lock-free (1ms)

Nonblocking TMM

= TMM + LF JVSTM

NbTMM: Goals

NbTMM: Goals

- Improve clustered performance

NbTMM: Goals

- Improve clustered performance
- Preserve TMM's properties

NbTMM: Goals

- Improve clustered performance
- Preserve TMM’s properties
- Nonblocking algorithms only

NbTMM: Goals

- Improve clustered performance
- Preserve TMM's properties
- Nonblocking algorithms only
- Minimize communication costs

NbTMM: Design highlights

NbTMM: Design highlights

- Read set not shared

NbTMM: Design highlights

- Read set not shared
- Only persist write set once

NbTMM: Design highlights

- Read set not shared
- Only persist write set once
- Unique write set keys

NbTMM: Design highlights

- Read set not shared
- Only persist write set once
- Unique write set keys
- Immutable data mappings

NbTMM: Design highlights

- Read set not shared
- Only persist write set once
- Unique write set keys
- Immutable data mappings
- Broadcast commit intentions

NbTMM: Design highlights

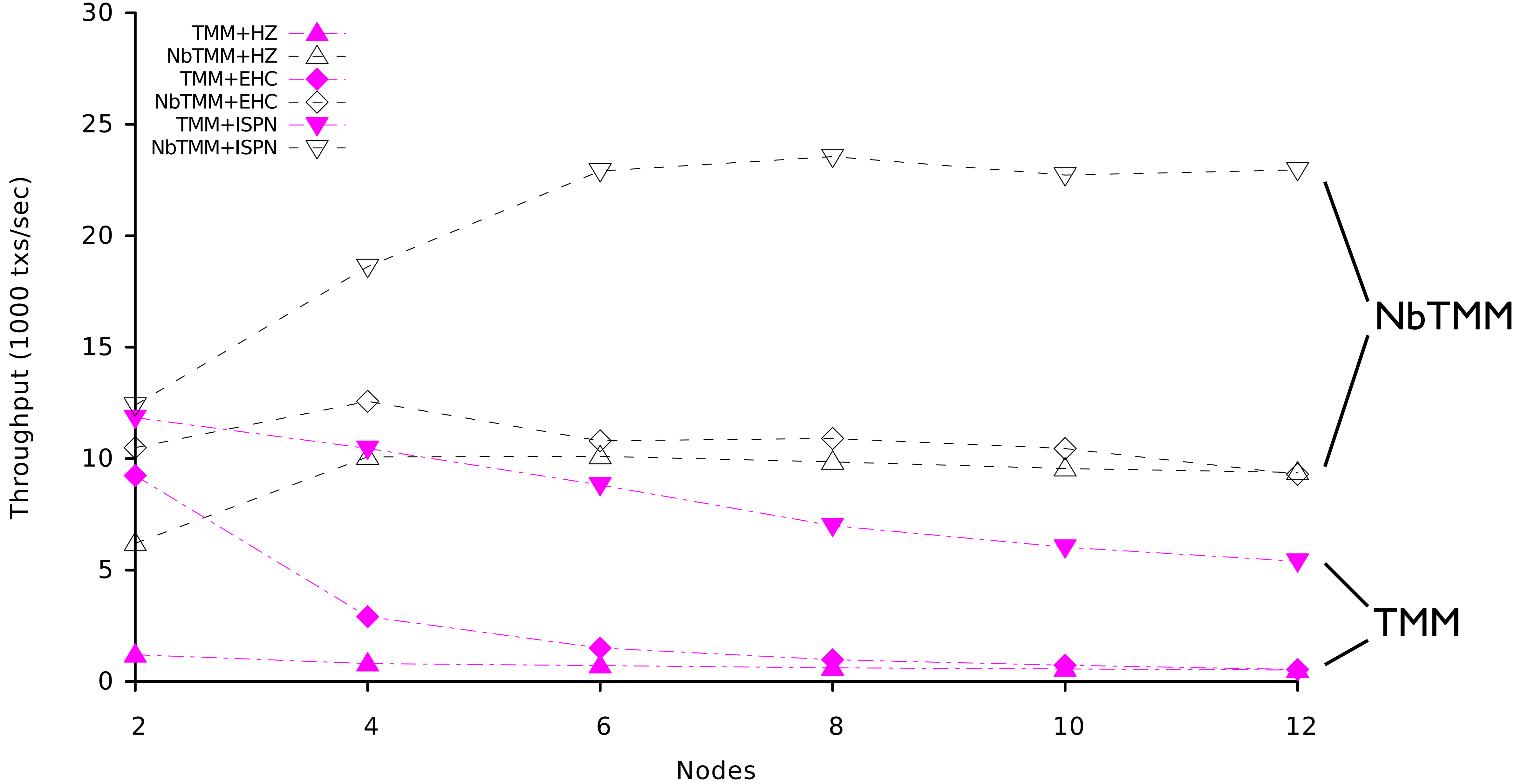
- Read set not shared
- Only persist write set once
- Unique write set keys
- Immutable data mappings
- Broadcast commit intentions
- Deterministic, independent commit decision

Results (Clustered)

TMM vs. NbTMM

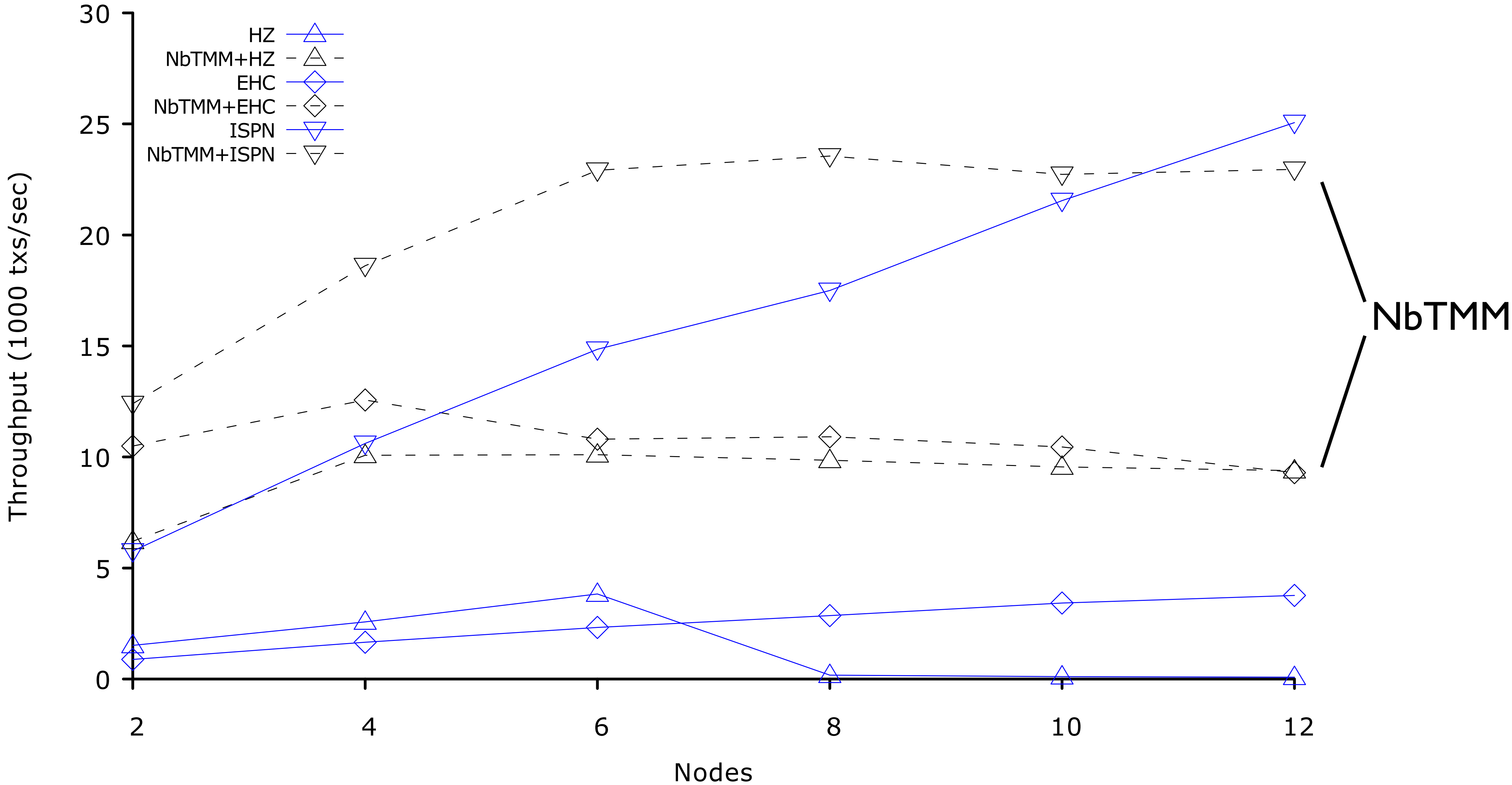
Results (Clustered)

TMM vs. NbTMM



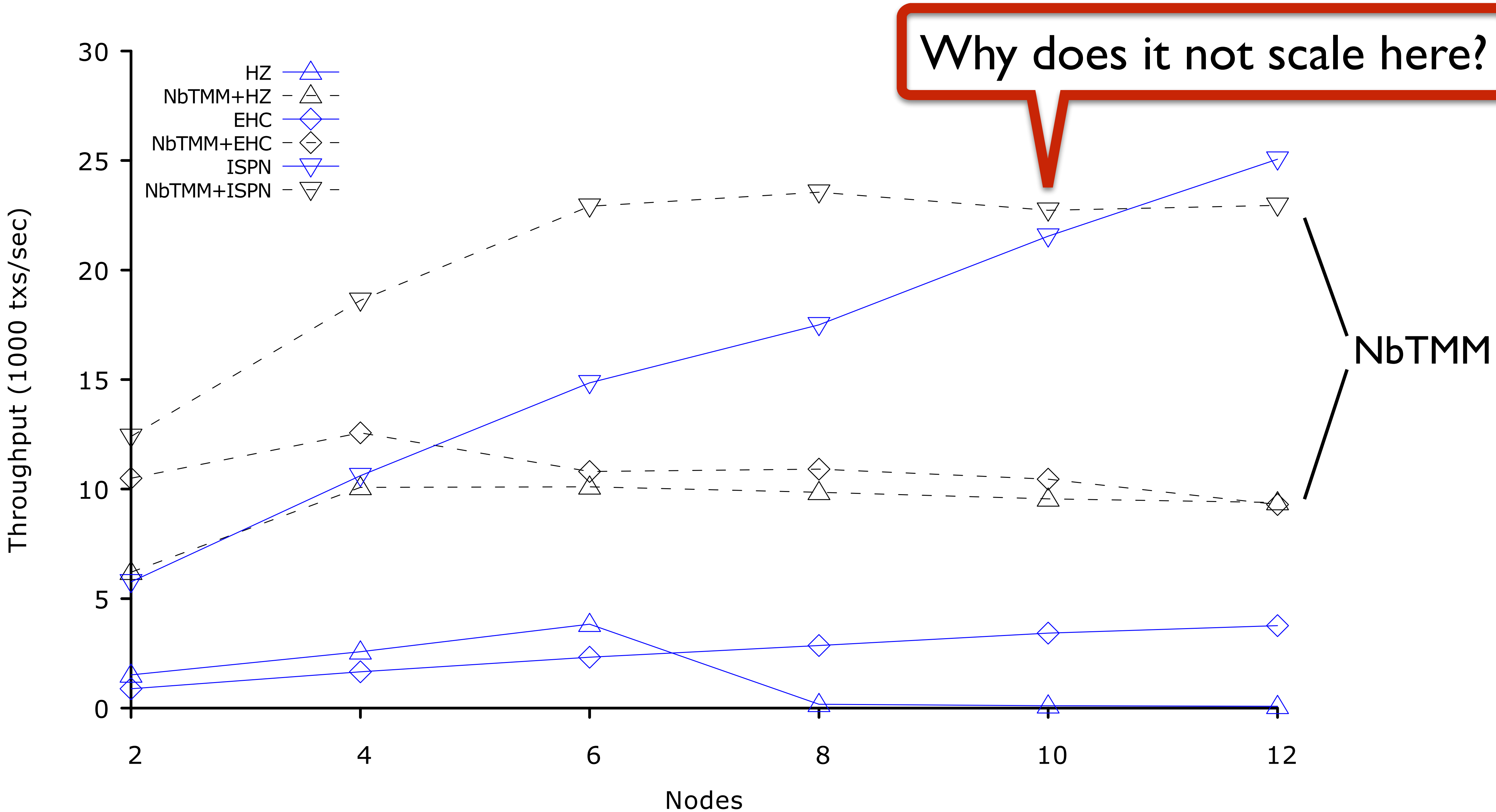
Results (Clustered)

NbTMM vs. TDGs

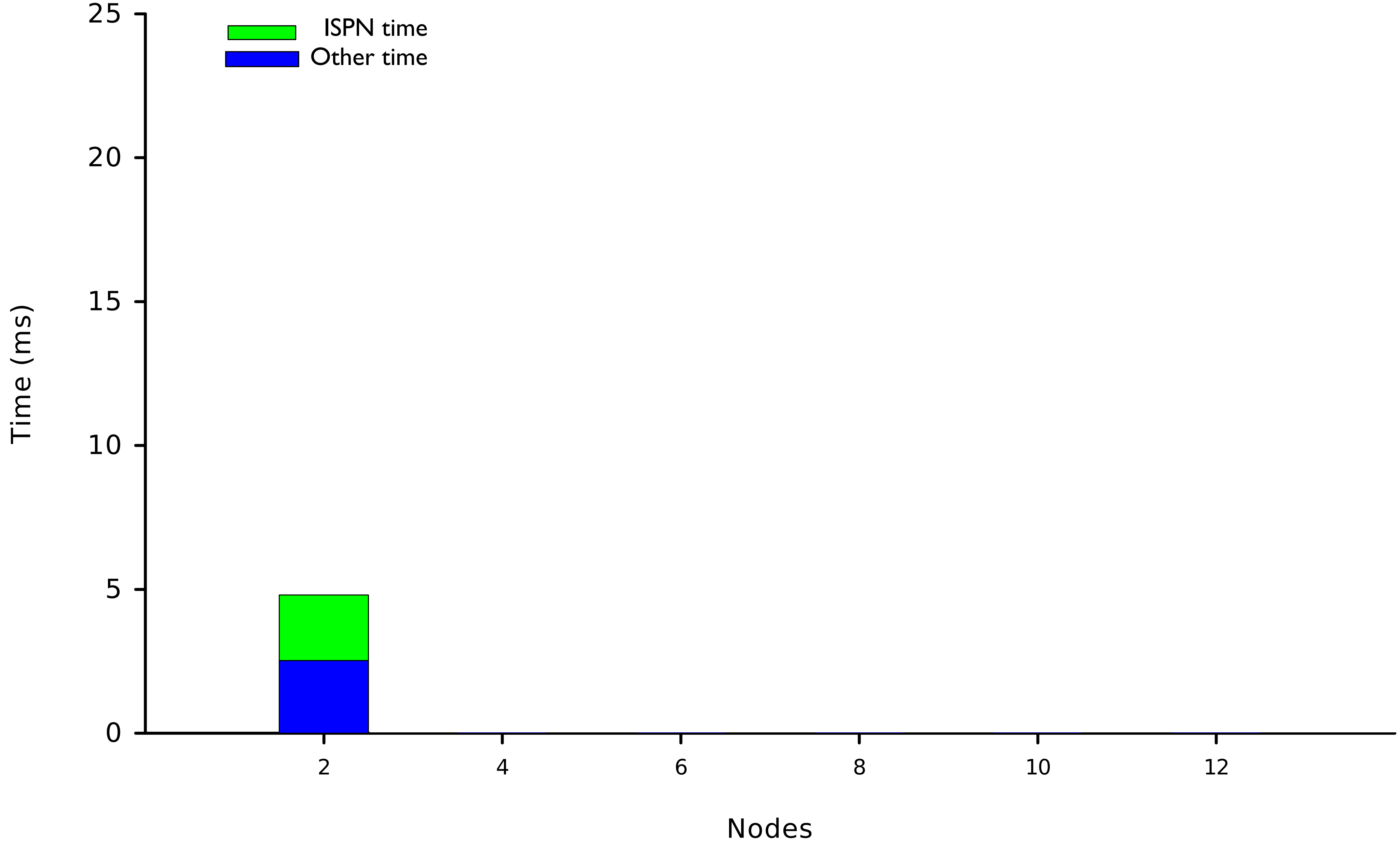


Results (Clustered)

NbTMM vs. TDGs

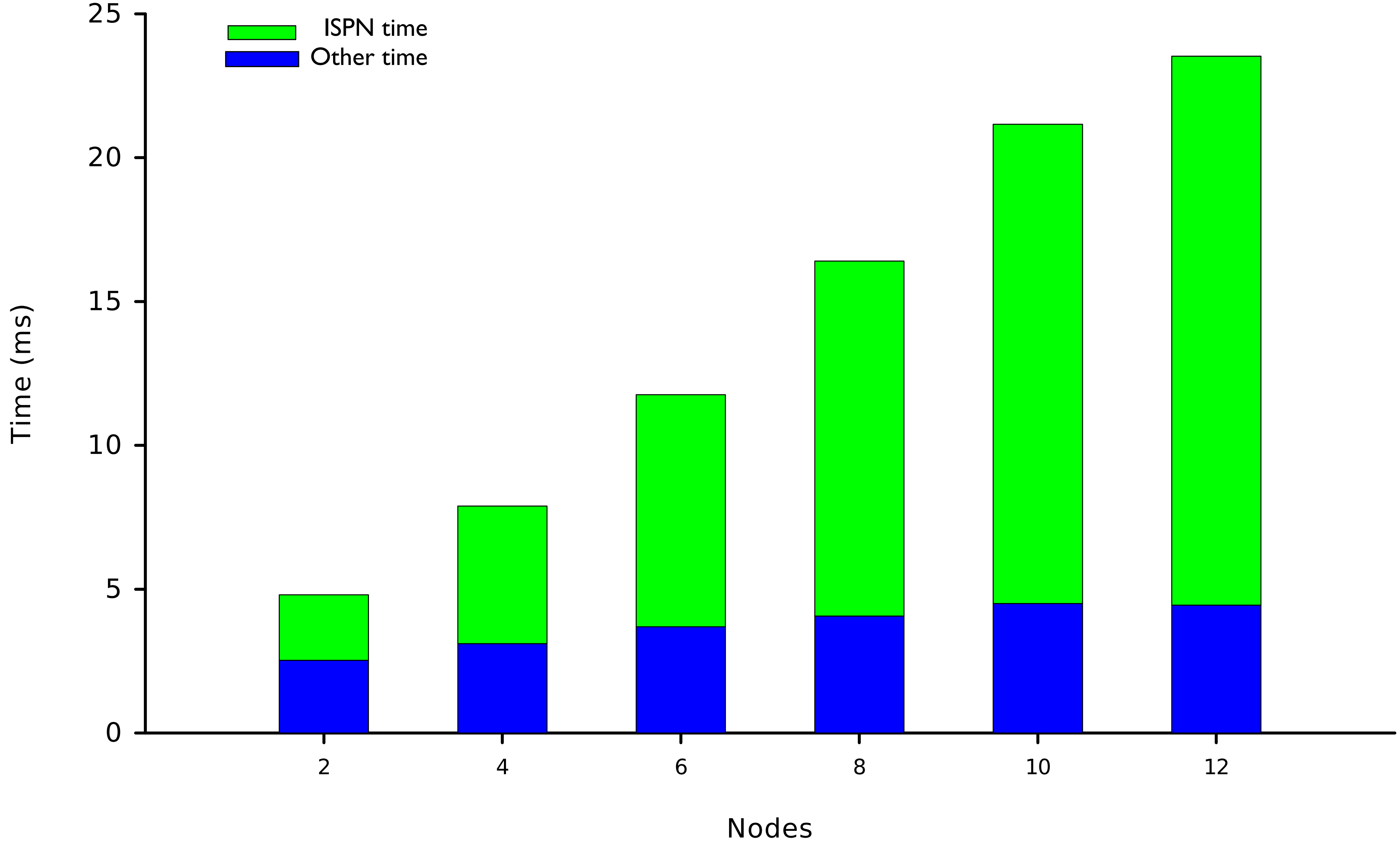


Results (Clustered)
Avg. Commit Time

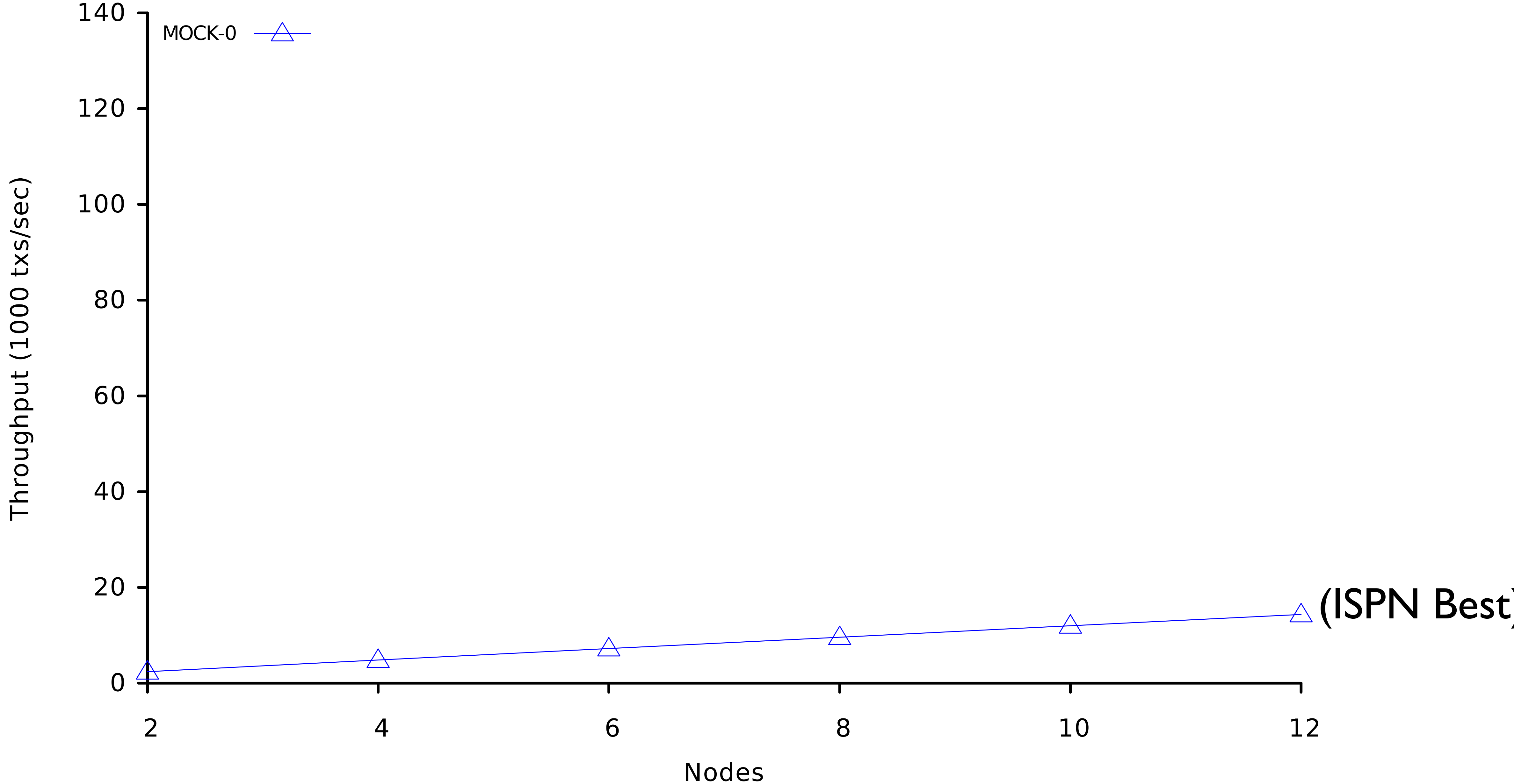


Results (Clustered)

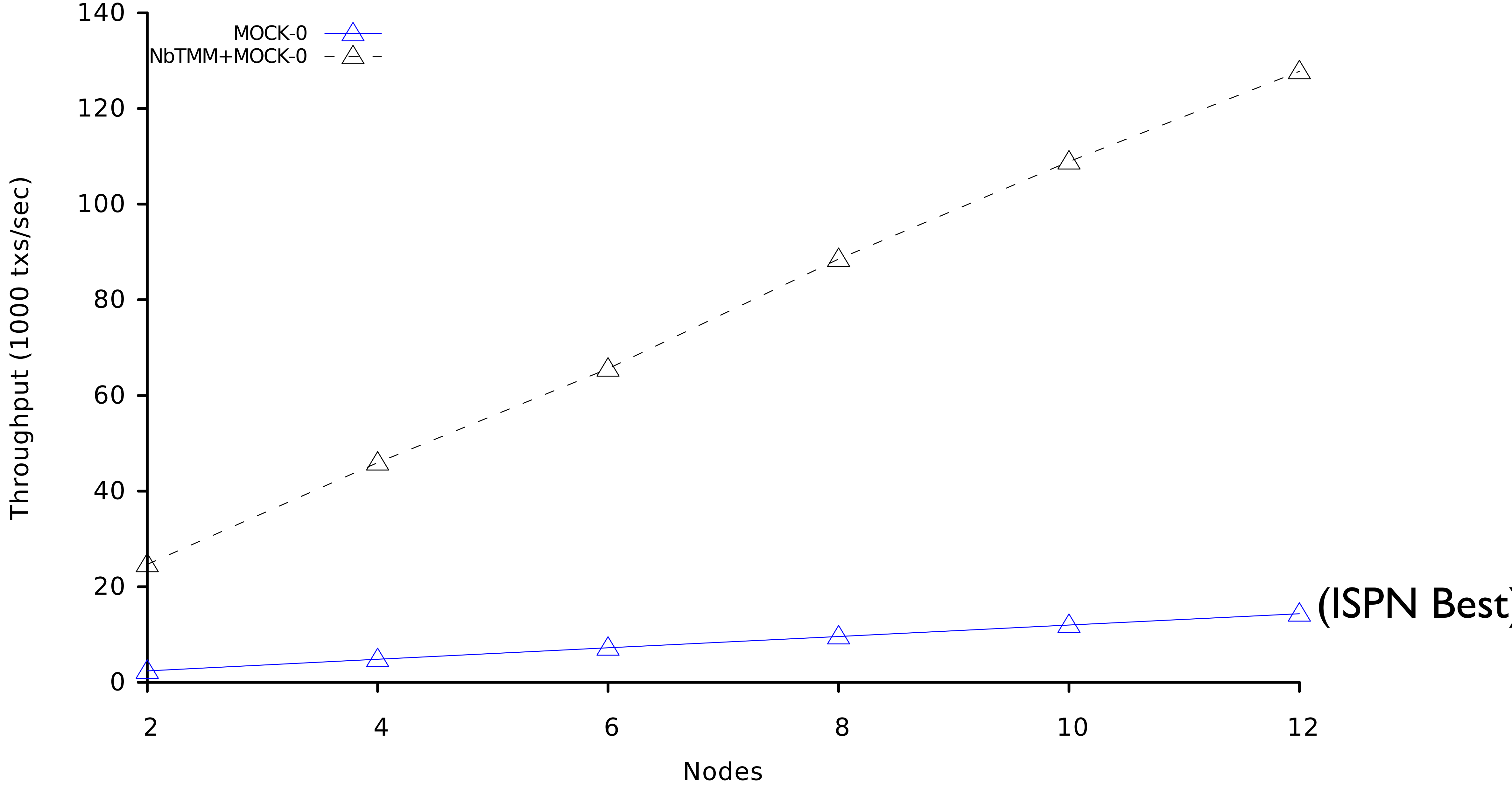
Avg. Commit Time



Results (Clustered)
Simulated ISPN

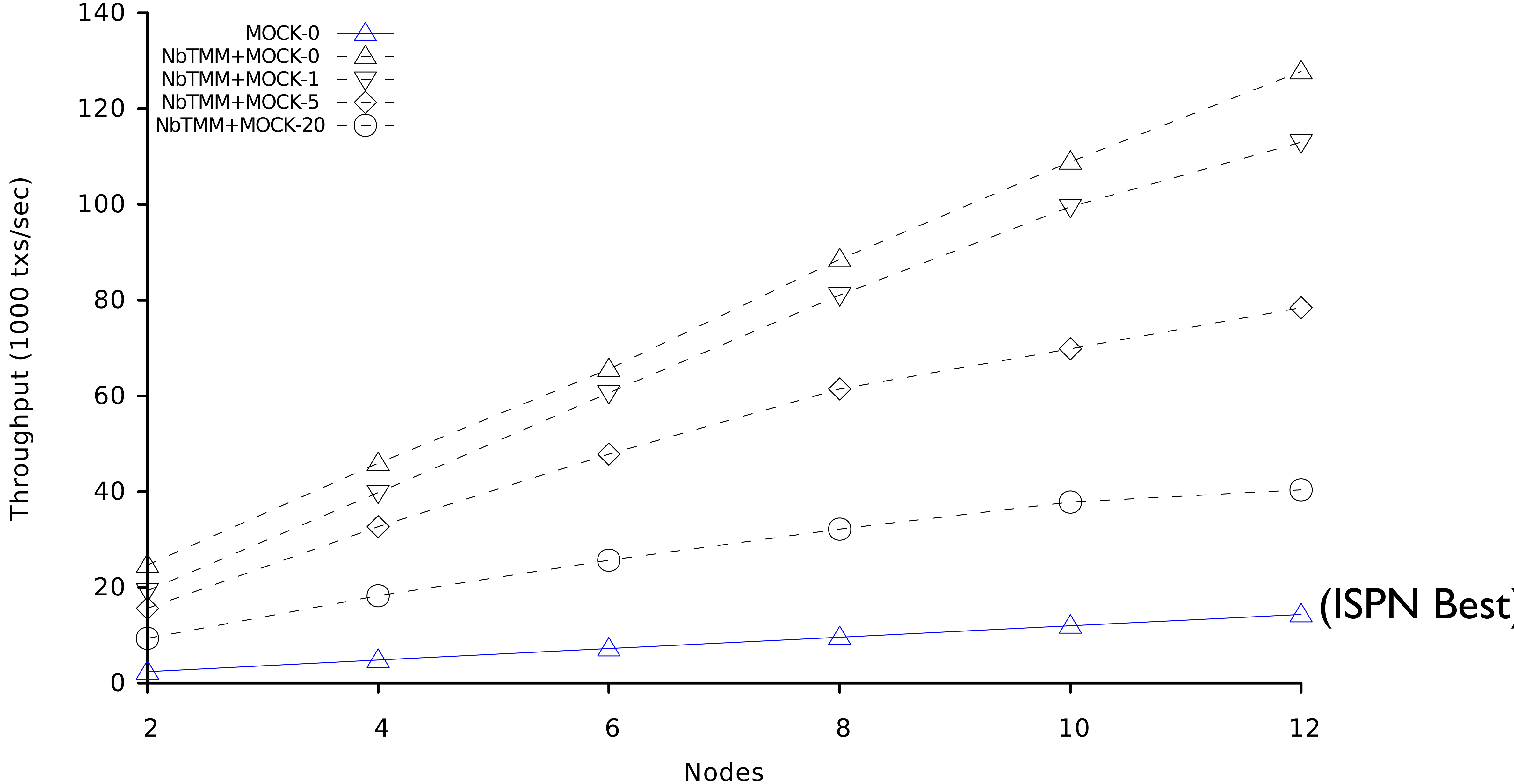


Results (Clustered)
Simulated ISPN



Results (Clustered)

Simulated ISPN



Main Contributions

Design and implementation of:

Main Contributions

Design and implementation of:

- Transactional middleware for enterprise applications based on STM

Main Contributions

Design and implementation of:

- Transactional middleware for enterprise applications based on STM
- Efficient lock-free multi-version STM

Main Contributions

Design and implementation of:

- Transactional middleware for enterprise applications based on STM
- Efficient lock-free multi-version STM
- NbTMM alternative to TMM using nonblocking algorithms

Publications

- **<u>Sérgio Fernandes</u>** and João Cachopo. *A scalable and efficient commit algorithm for the JVSTM*. 5th ACM SIGPLAN Workshop on Transactional Computing, April 2010.
- **<u>Sérgio Fernandes</u>** and João Cachopo. *Lock-free and scalable multi-version Software Transactional Memory*. 16th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming, 179-188, February 2011.
- **<u>Sérgio Fernandes</u>** and João Cachopo. *Strict Serializability is Harmless: A New Architecture for Enterprise Applications*. SPLASH Wavefront 2011, Portland, Oregon, USA, October 2011.
- Jorge Martins, João Pereira, **<u>Sérgio Fernandes</u>** and João Cachopo. *Towards a simple programming model in Cloud Computing platforms*. IEEE First Symposium on Network Cloud Computing and Applications , Toulouse, France, 83-90, November 2011.
- Nuno Diegues, **<u>Sérgio Fernandes</u>** and João Cachopo. *Parallel nesting in a lock-free multi-version Software Transactional Memory*. 7th ACM SIGPLAN Workshop on Transactional Computing, February 2012.

Strongly Consistent Transactions for Enterprise Applications

Using Software Transactional Memory to Improve Consistency and Performance of Read-Dominated Workloads

Sérgio Miguel Martinho Fernandes

Thank You!