Scalable Logging Algorithm for in-Memory Database Systems

Henry Liu, Justin Kaashoek, Siye Zhu

Database Management Systems

OLTP

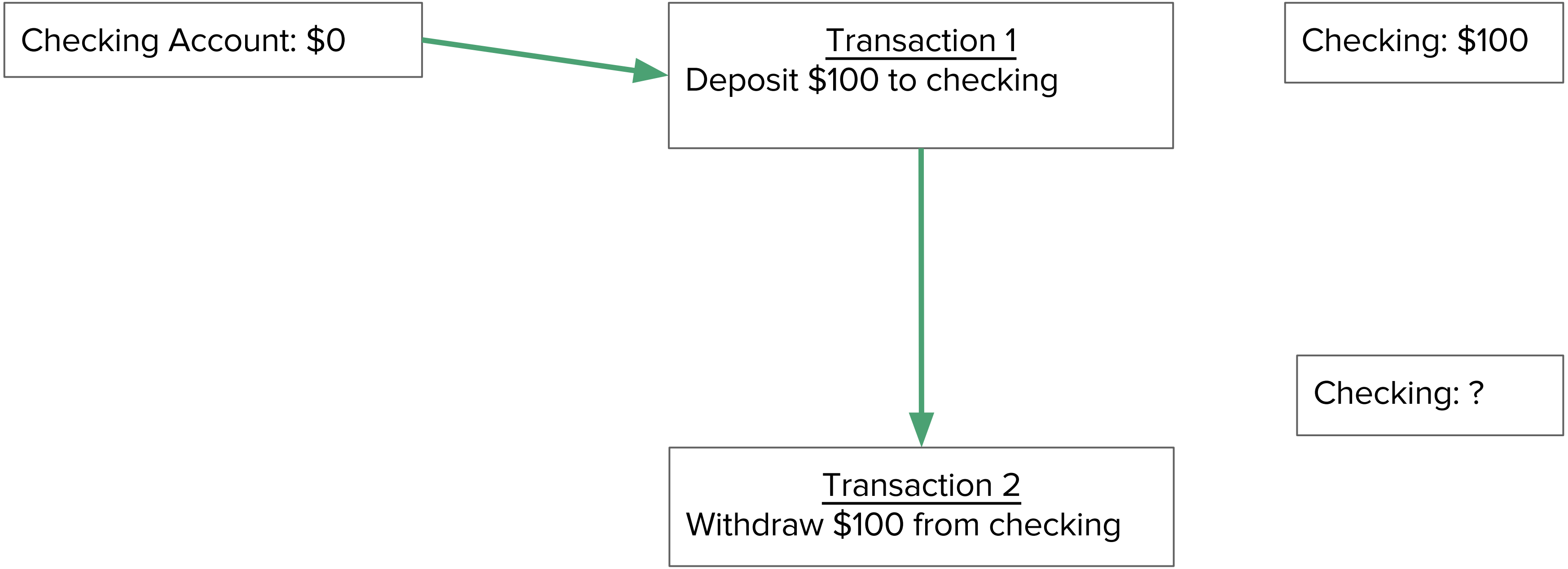
(Online Transaction Processing)

- ATM
- Online Shopping
- Retail Sales
- Financial Transaction

amazon

ebay

Why Logging is Important – Durability



Checking Account: $0

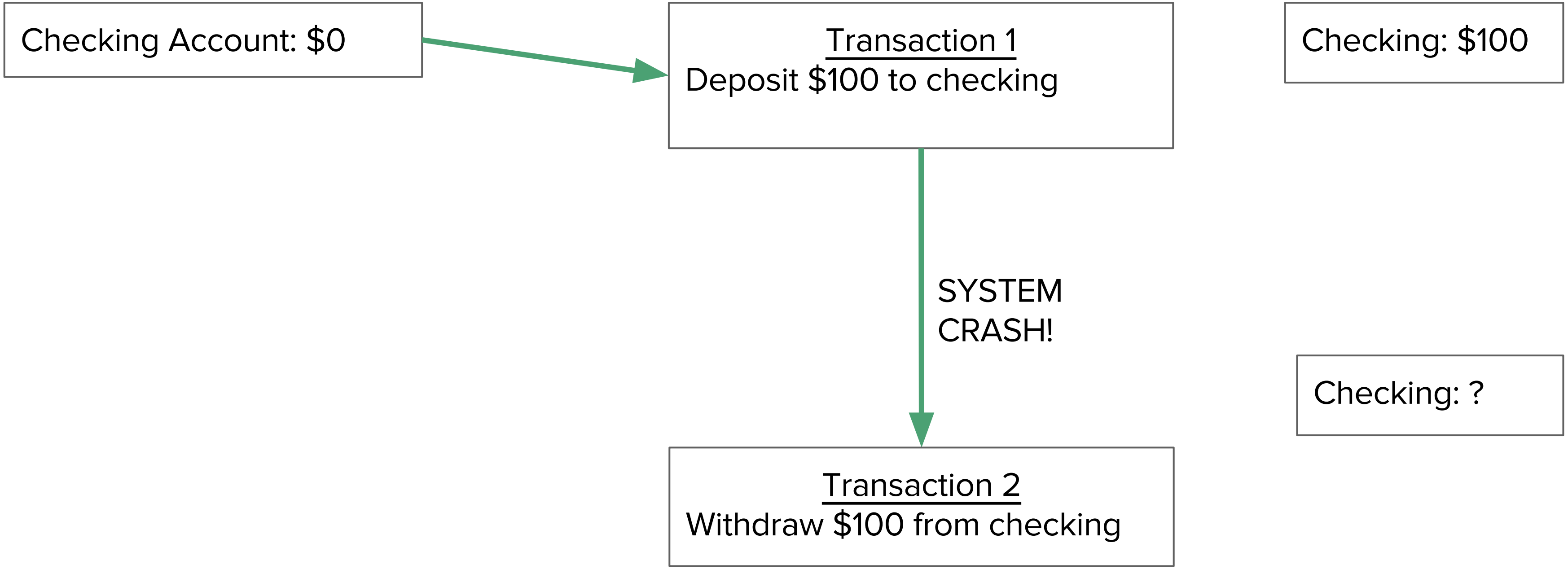
Transaction 1
Deposit $100 to checking

Checking: $100

Transaction 2
Withdraw $100 from checking

Checking: ?

Why Logging is Important – Durability



Checking Account: $0

Transaction 1
Deposit $100 to checking

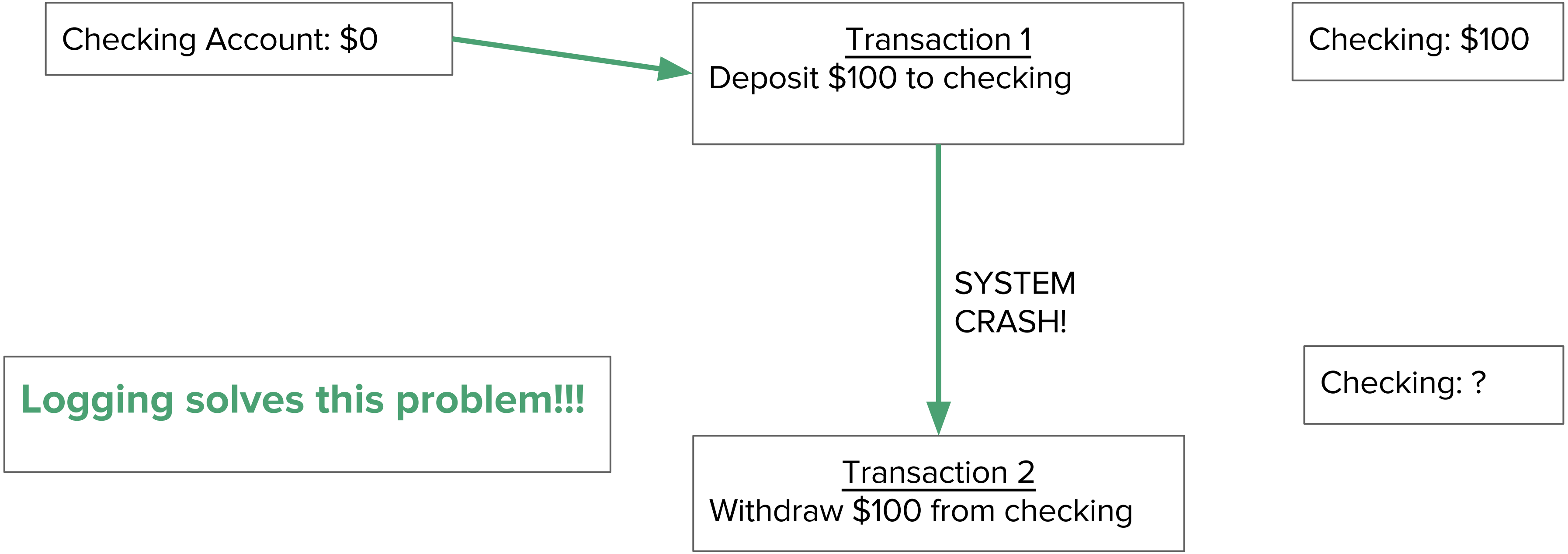
Checking: $100

SYSTEM
CRASH!

Checking: ?

Transaction 2
Withdraw $100 from checking

Why Logging is Important – Durability



Checking Account: $0

Transaction 1

Deposit $100 to checking

Checking: $100

SYSTEM CRASH!

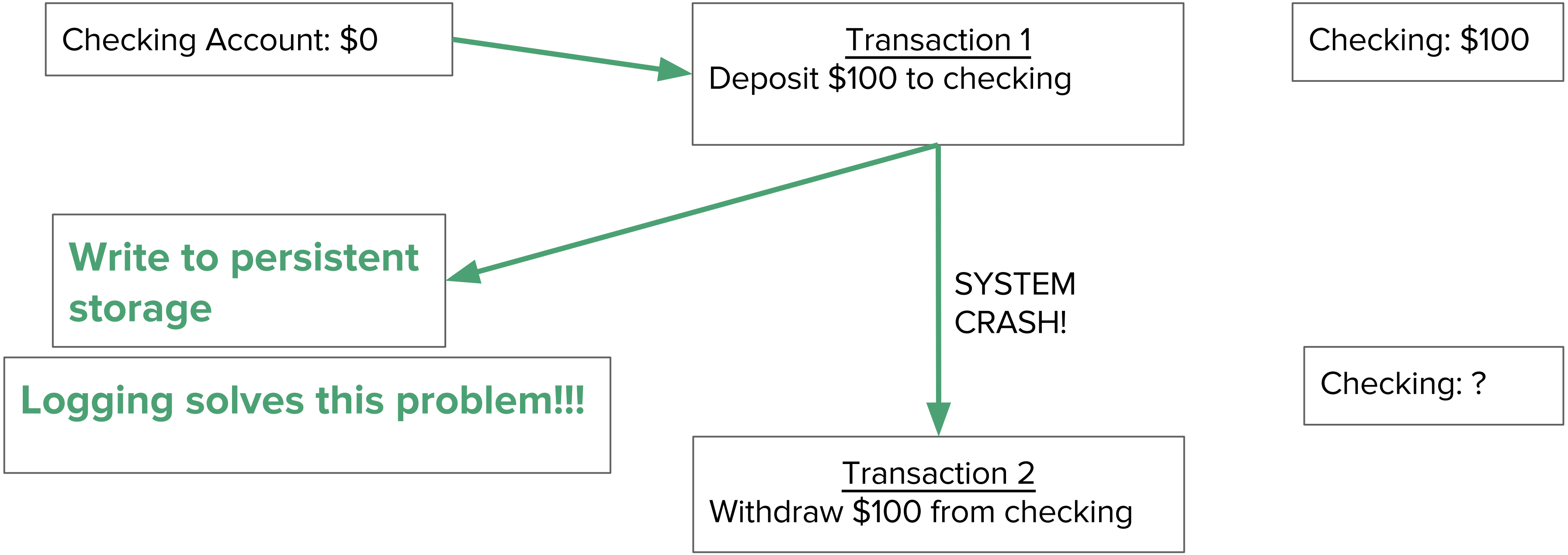
Transaction 2

Withdraw $100 from checking

Checking: ?

Logging solves this problem!!!

Why Logging is Important – Durability



Checking Account: $0

Transaction 1
Deposit $100 to checking

Checking: $100

Write to persistent storage

SYSTEM CRASH!

Logging solves this problem!!!

Checking: ?

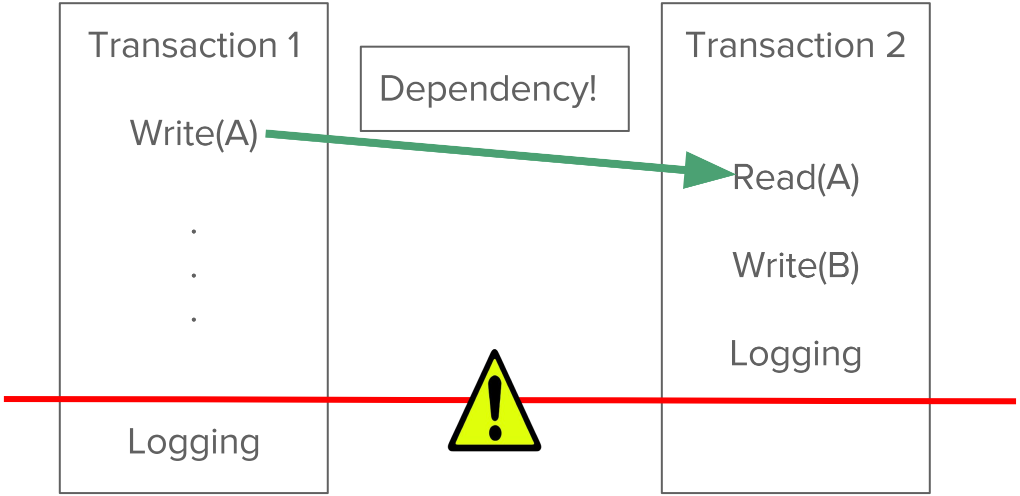
Transaction 2
Withdraw $100 from checking

How to log?

- The algorithm must be more scalable and efficient than current algorithms
- Serial logging
- Batch logging
- Parallel logging

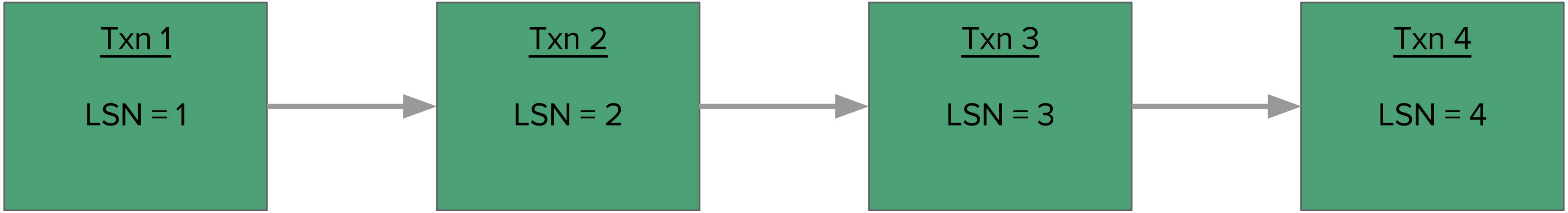
Transaction Dependency

- If a transaction is dependent on another transaction, they must be logged in order
- The logging algorithm must account for these dependencies!



Serial Logging

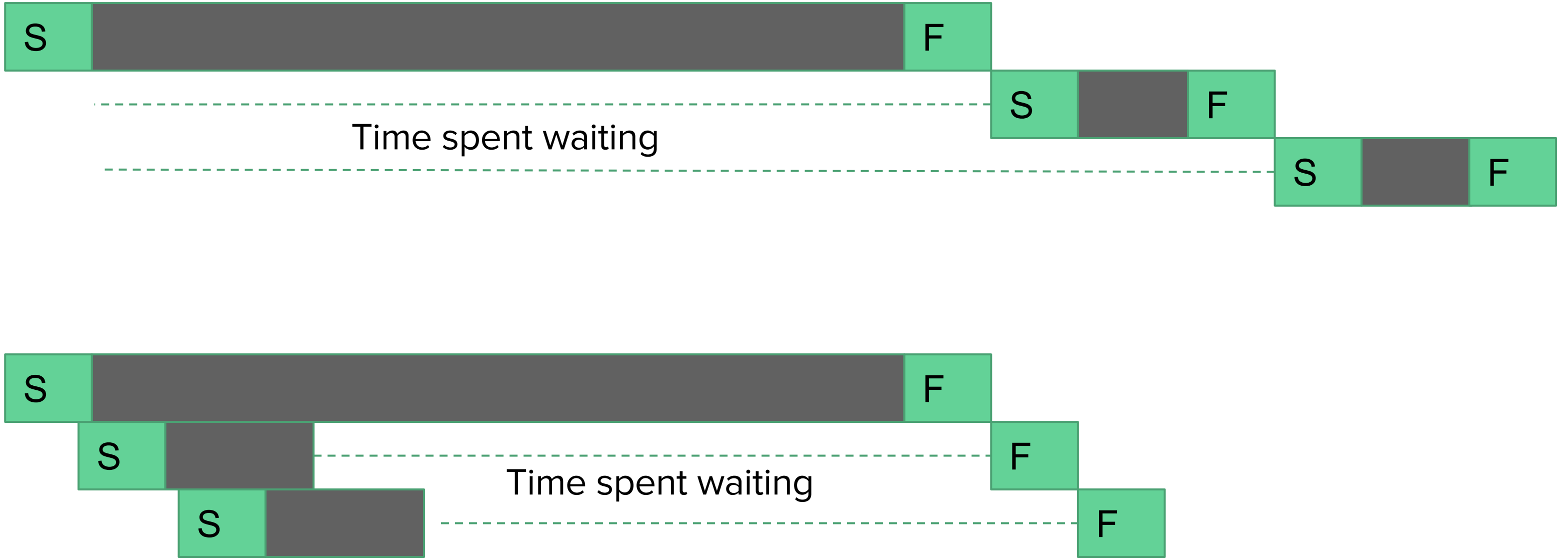
- The easiest solution to the dependency problem: log transactions in order
- Each transaction acquires a unique *Log Sequence Number* (LSN) at commit time.



Optimizing Serial Logging

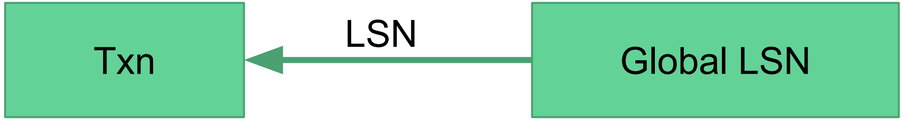
Standard, without optimization

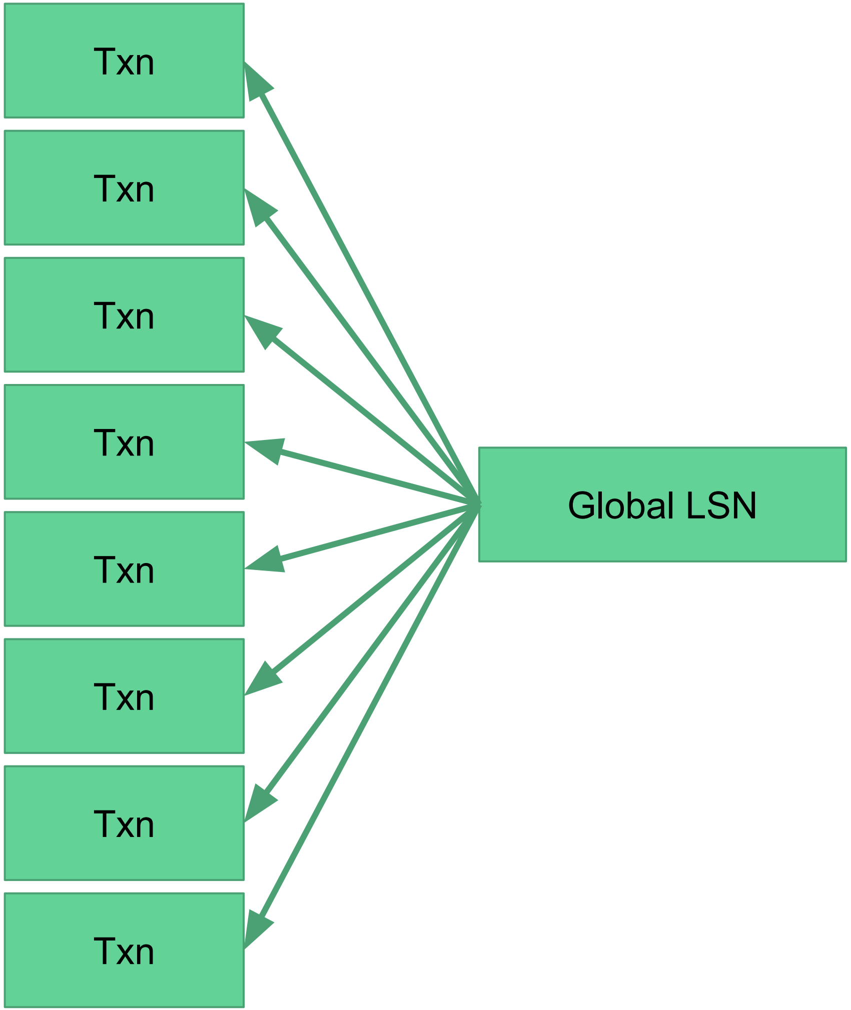
With Optimization



Batch Logging

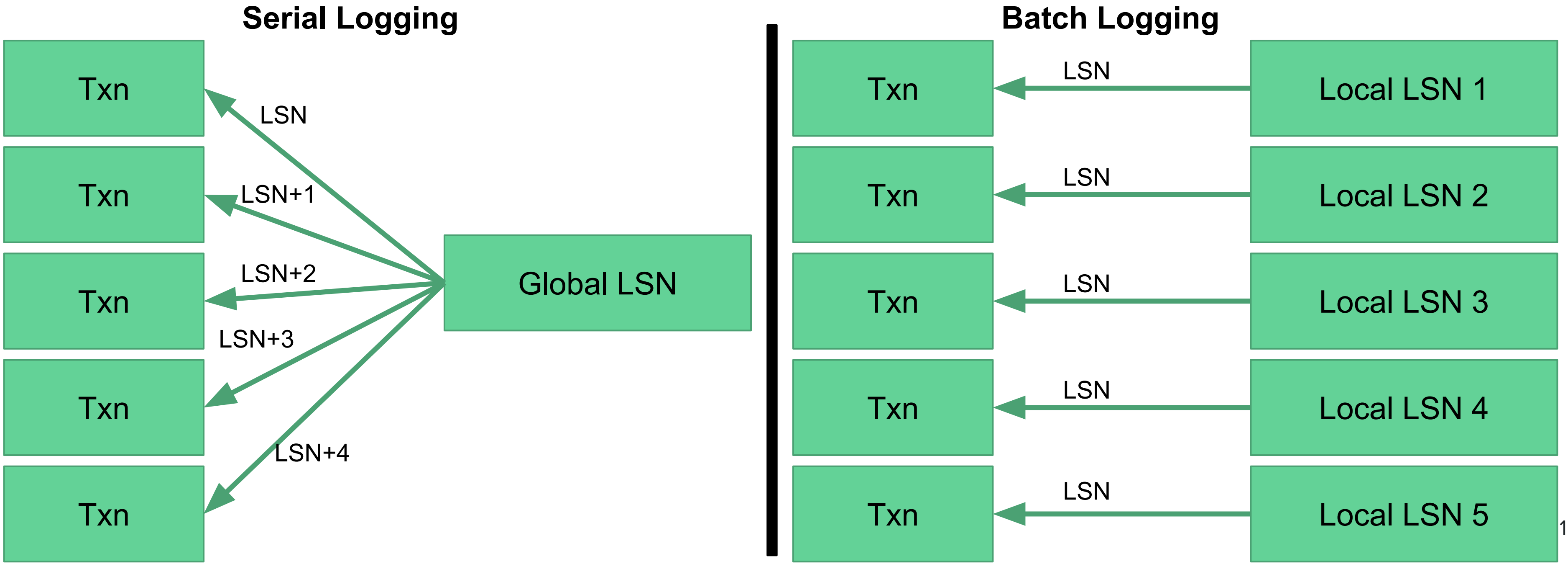
- With serial logging, each transaction needs an LSN from the Global LSN
- Quickly becomes bottleneck with large number of transactions





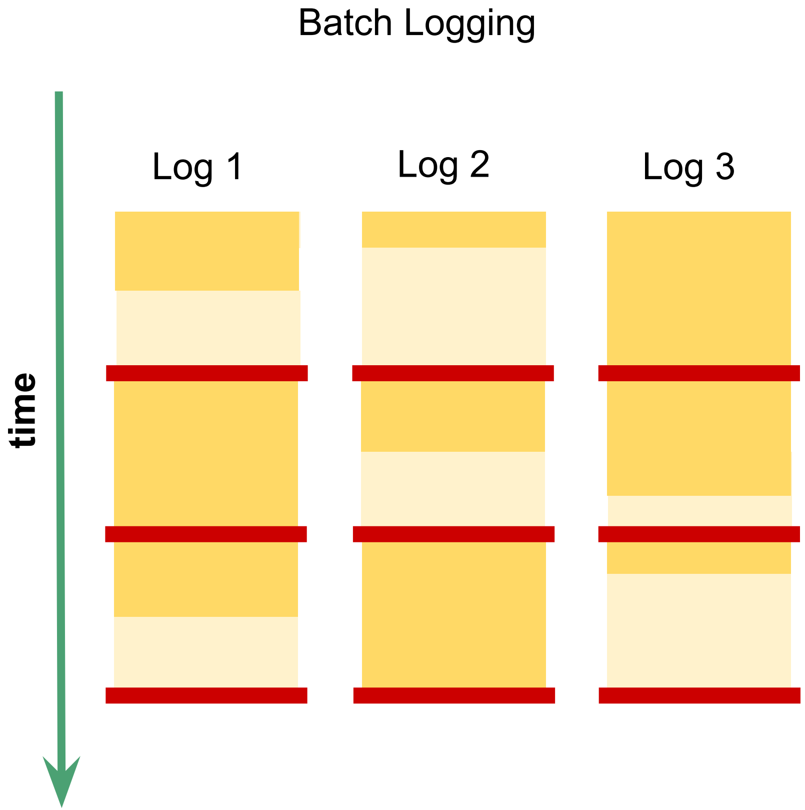
Batch Logging

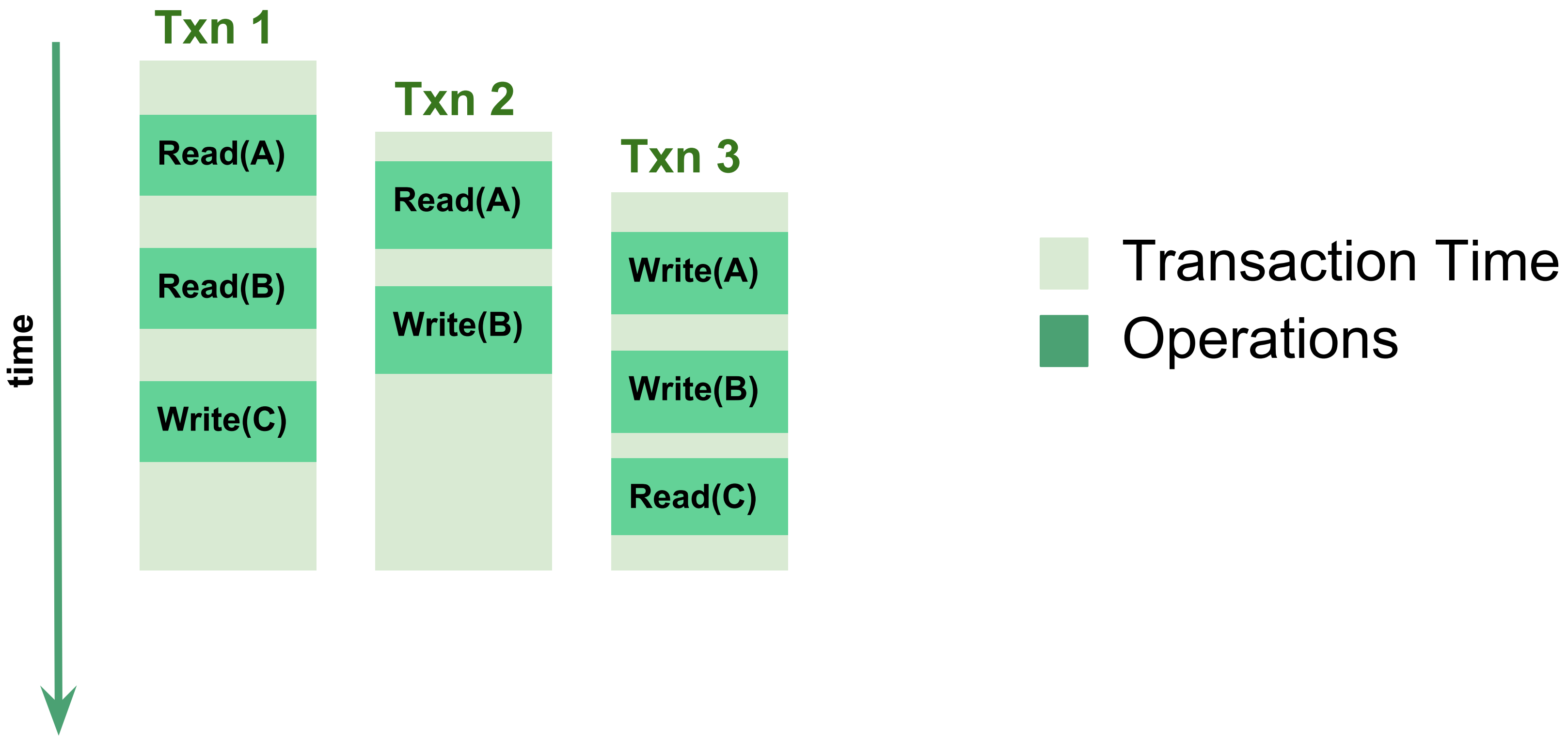
Batch logging provides one solution to this problem by having multiple loggers with multiple *local* LSNs instead of one global LSN. This removes the bottleneck



Batch Logging

- Assume dependencies between loggers
- Sync before returning to user
- Our implementation: Flush all loggers when one becomes full
- Drawback: high latency





time

Txn 1

Read(A)

Read(B)

Write(C)

Txn 2

Read(A)

Write(B)

Txn 3

Write(A)

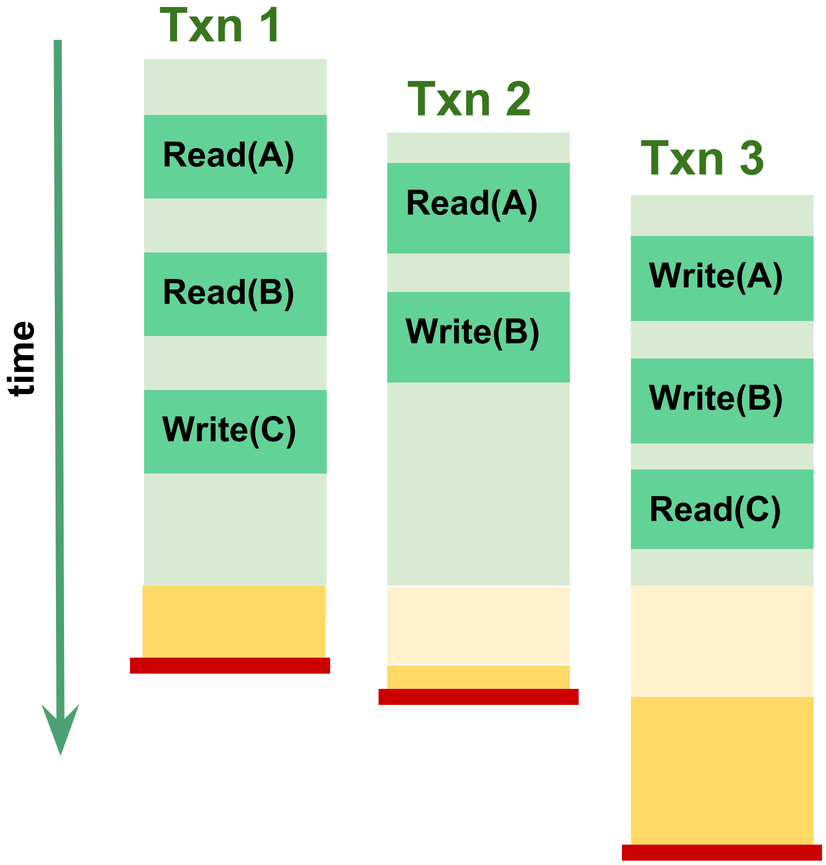
Write(B)

Read(C)

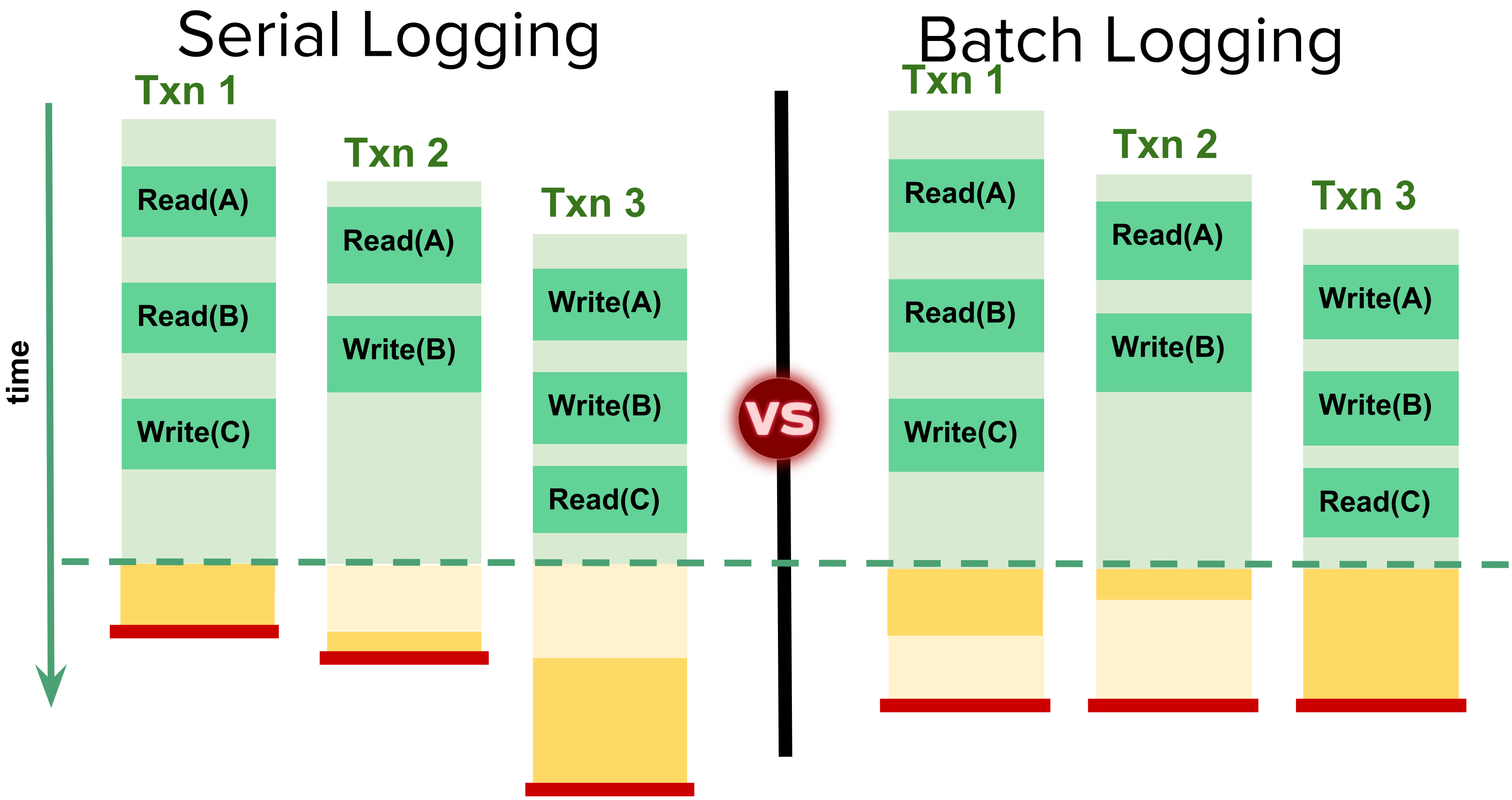
░ Transaction Time

▓ Operations

Serial Logging



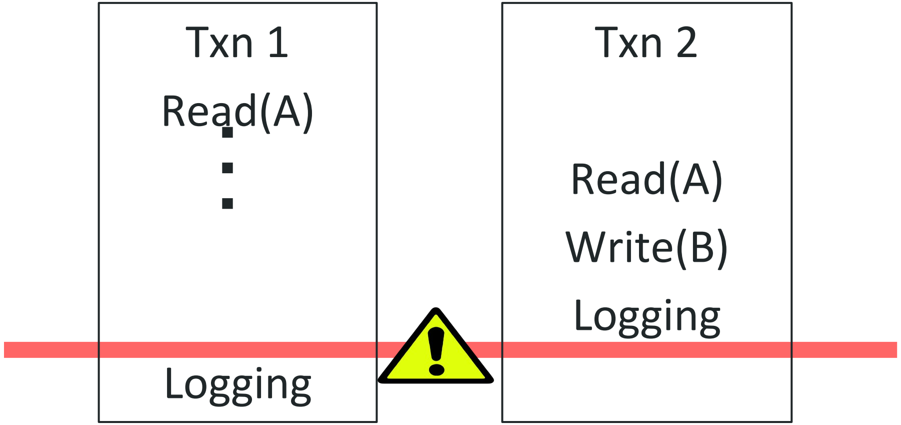
- 🟩 Transaction Time
- 🟩 Operations
- 🟨 Log Time
- 🟨 Wait Time
- 🟥 Commit


Serial Logging
Batch Logging
time
Txn 1
Read(A)
Read(B)
Write(C)
Txn 2
Read(A)
Write(B)
Txn 3
Write(A)
Write(B)
Read(C)
VS
Txn 1
Read(A)
Read(B)
Write(C)
Txn 2
Read(A)
Write(B)
Txn 3
Write(A)
Write(B)
Read(C)

Transaction Dependency

Independent Logging?

- **No dependency**
- RAW (Read After Write)
- WAW (Write After Write)
- WAR (Write After Read)



Txn 1

Read(A)

⋮

Logging

Txn 2

Read(A)

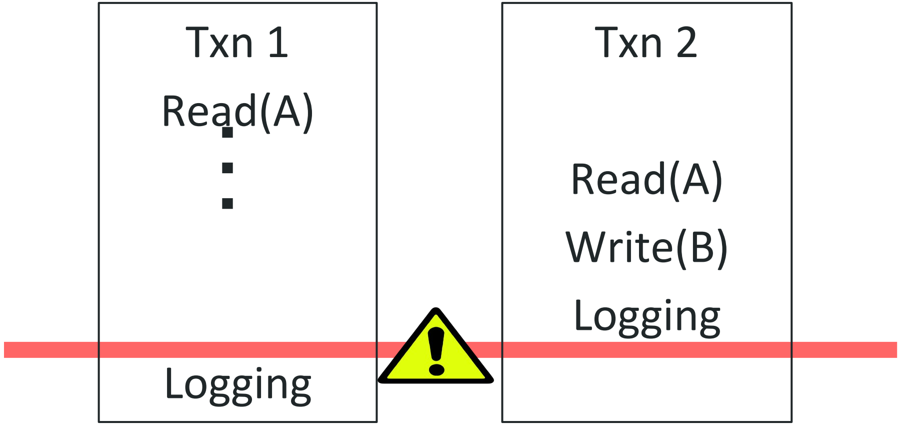
Write(B)

Logging

Transaction Dependency

- **No dependency**
- RAW (Read After Write)
- WAW (Write After Write)
- WAR (Write After Read)

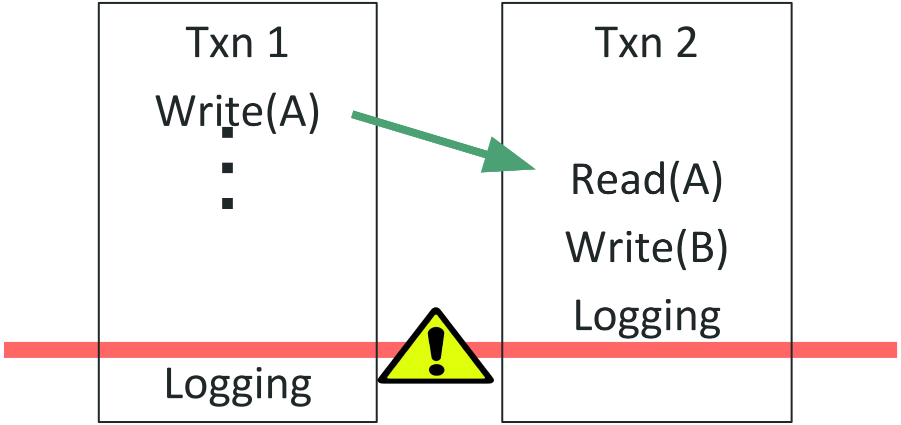
Independent Logging?
YES



Transaction Dependency

- No dependency
- **RAW (Read After Write)**
- WAW (Write After Write)
- WAR (Write After Read)

Independent Logging?
YES



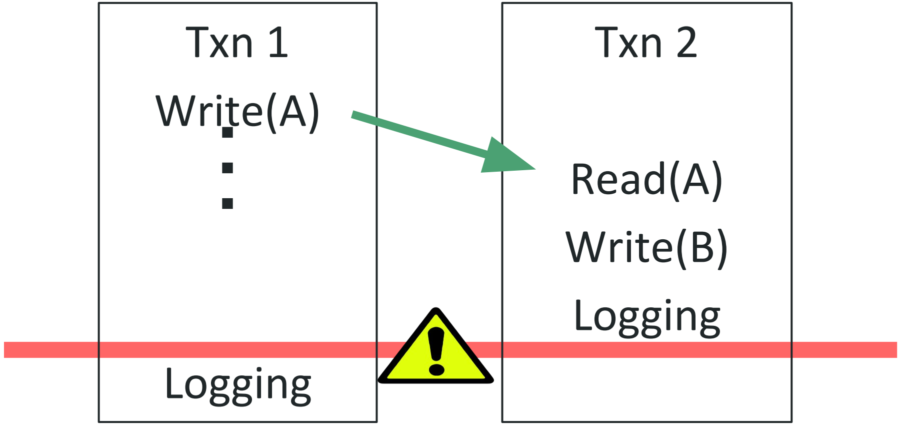
Transaction Dependency

- No dependency
- **RAW (Read After Write)**
- WAW (Write After Write)
- WAR (Write After Read)

Independent Logging?

YES

No



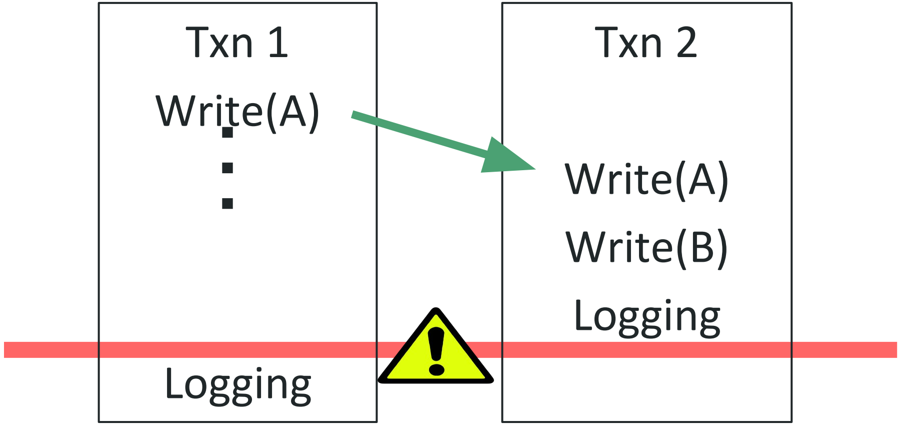
Transaction Dependency

- No dependency
- RAW (Read After Write)
- **WAW (Write After Write)**
- WAR (Write After Read)

Independent Logging?

YES

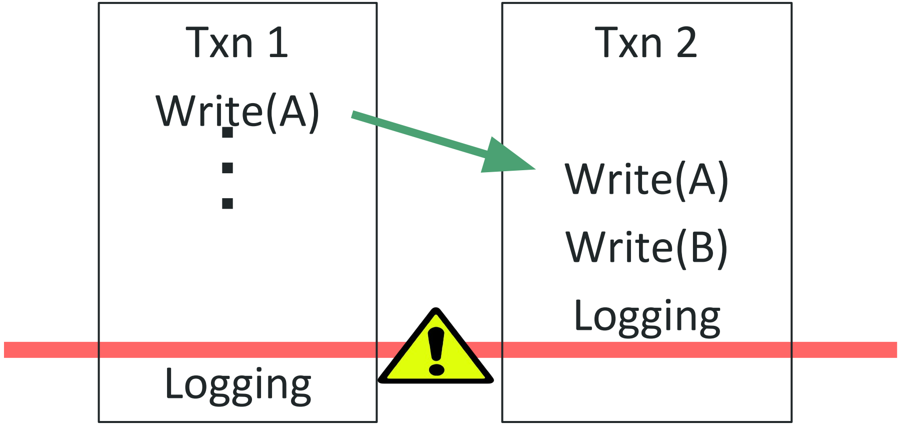
No



Transaction Dependency

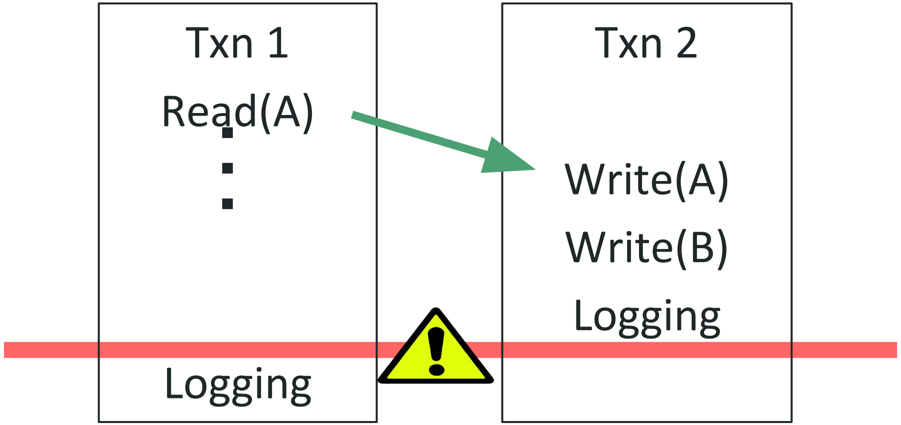
Independent Logging?

- No dependency YES
- RAW (Read After Write) No
- **WAW (Write After Write)** No
- WAR (Write After Read)



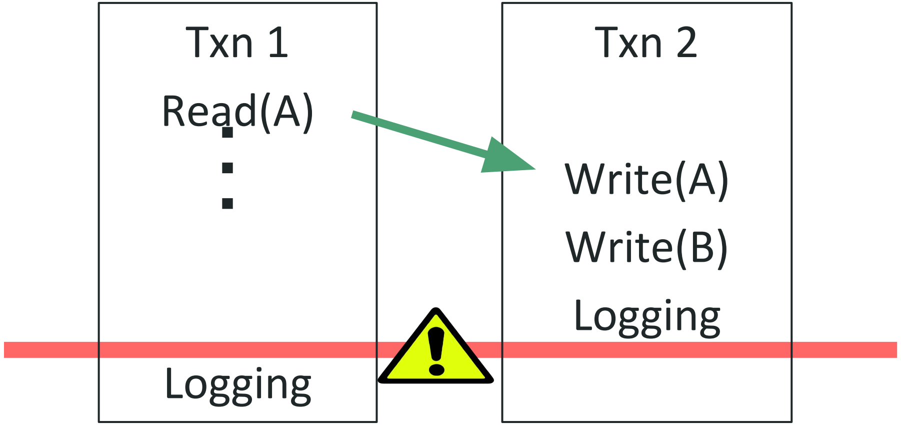
Transaction Dependency

	Independent Logging?
No dependency	YES
RAW (Read After Write)	No
WAW (Write After Write)	No
WAR (Write After Read)	



Transaction Dependency

	Independent Logging?
No dependency	YES
RAW (Read After Write)	No
WAW (Write After Write)	No
WAR (Write After Read)	**YES**



Txn 1

Read(A)

⋮

Logging

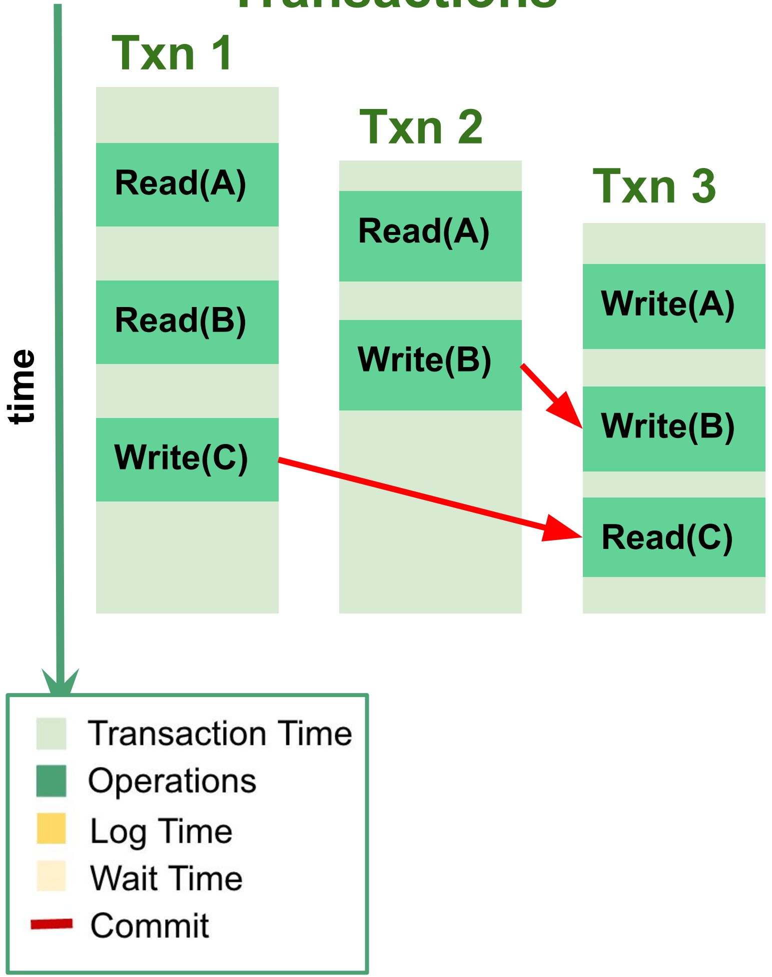
Txn 2

Write(A)

Write(B)

Logging

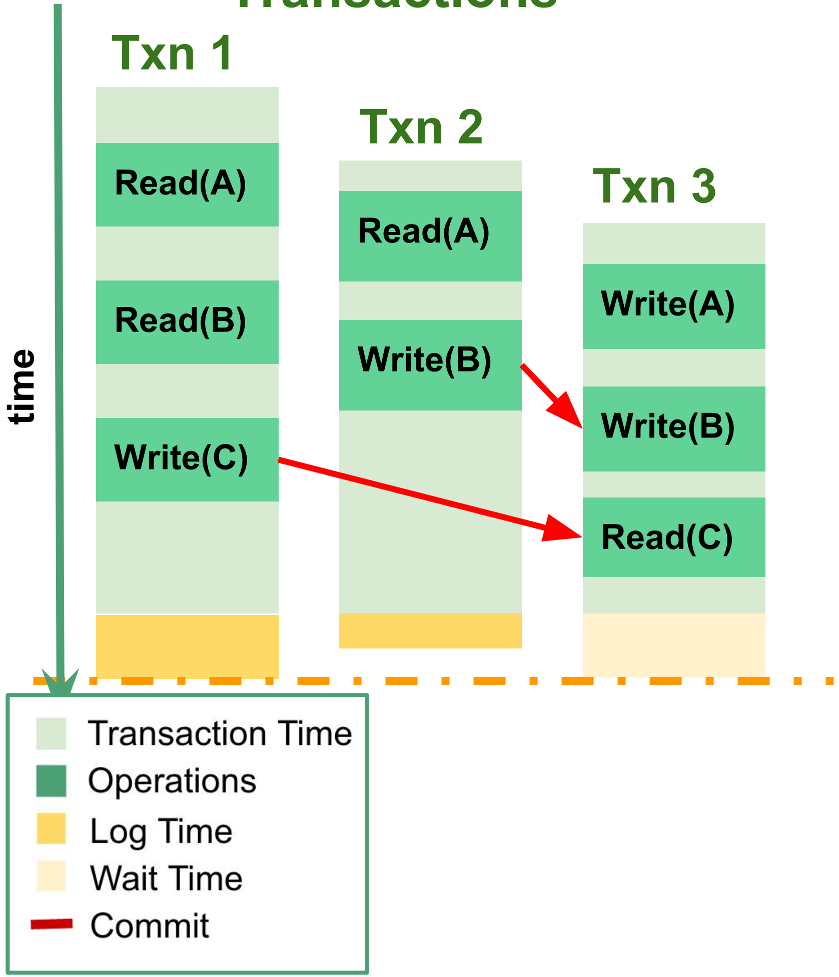
Transactions



Parallel Logger

LSN	Txn ID	Data Tuple	Dependency info
Logger 1			
Wait Buffer 1			
Logger 2			
Wait Buffer 2			

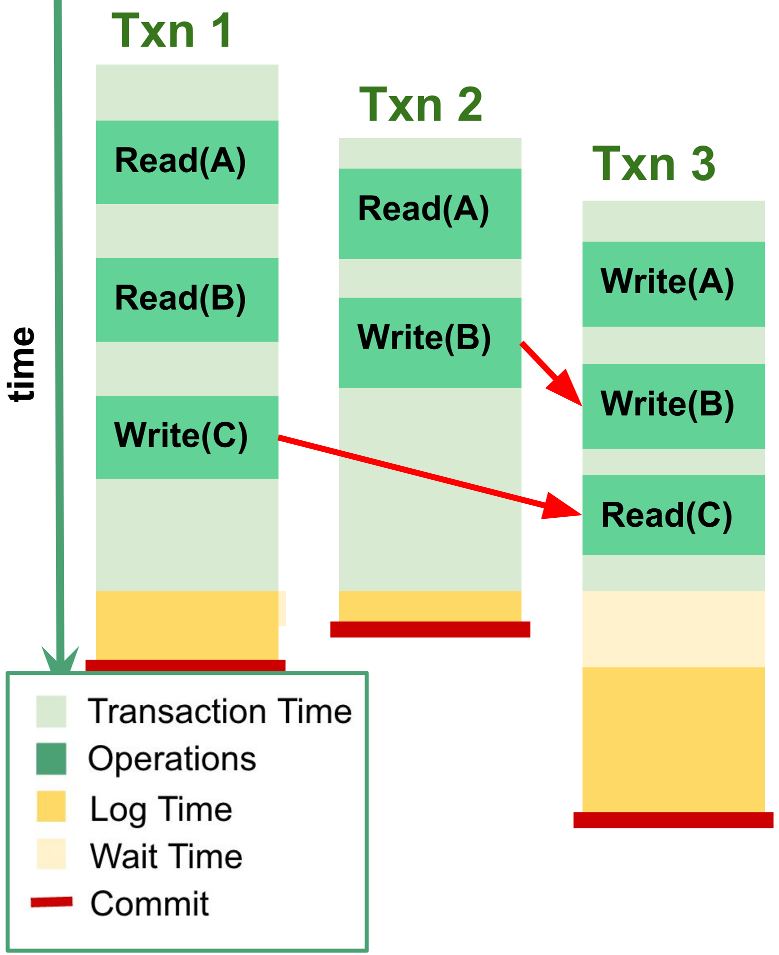
Transactions



Parallel Logger

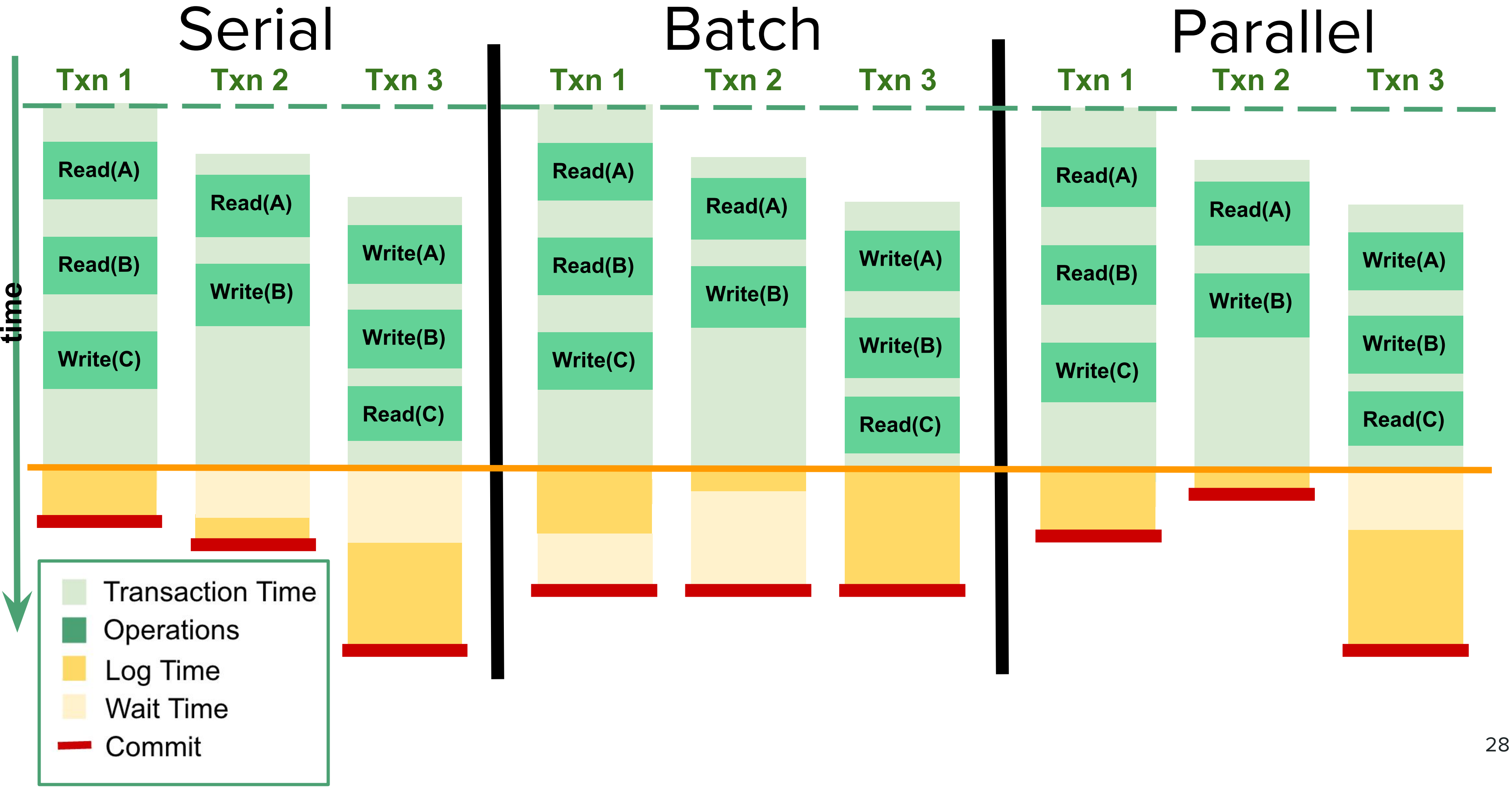
LSN	Txn ID	Data Tuple	Dependency info
Logger 1			
1	1	C	[0, 0]
Wait Buffer 1			
11	3	A	[1,1]
		B	
Logger 2			
1	2	B	[0,0]
Wait Buffer 2			

Transactions

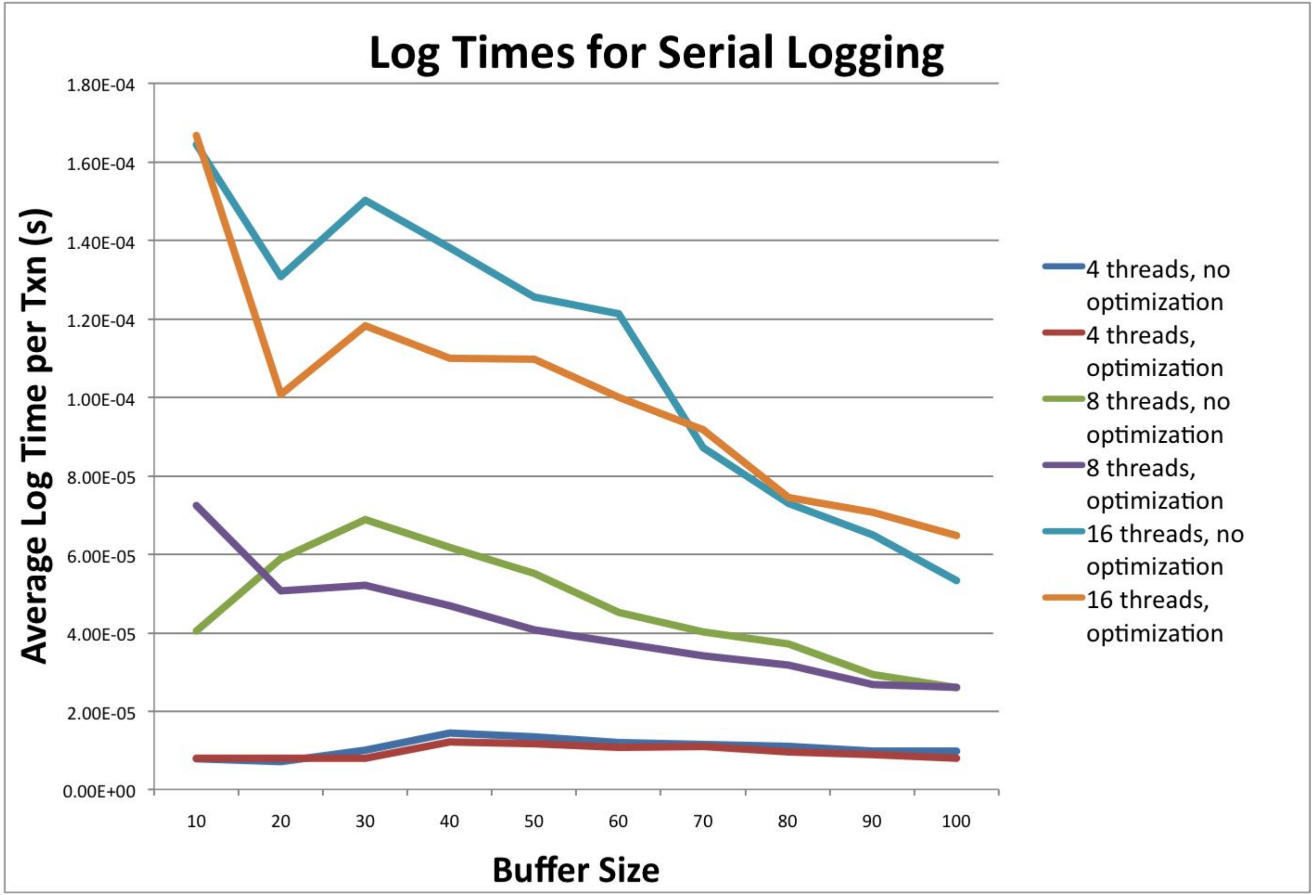


Parallel Logger

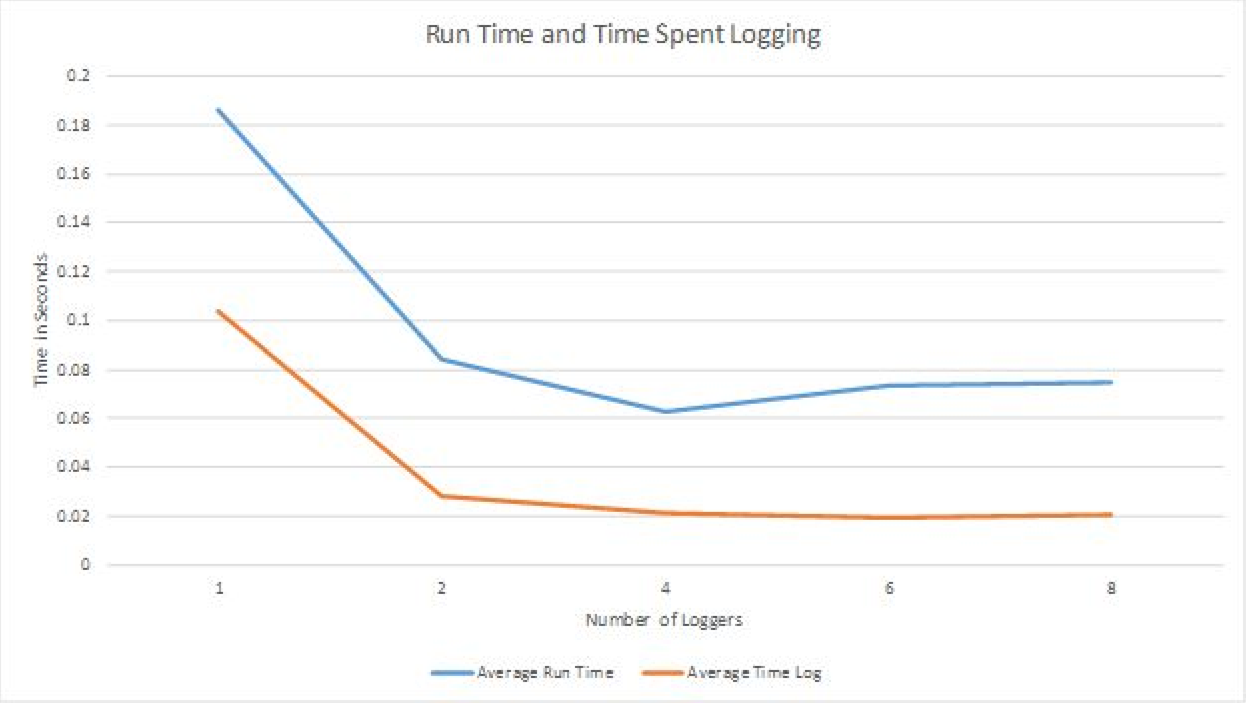
LSN	Txn ID	Data Tuple	Dependency info
Logger 1			
1	1	C	[0, 0]
11	**3**	**A**	**[1,1]**
		B	
Wait Buffer 1			
Logger 2			
1	2	B	[0,0]
Wait Buffer 2			


Serial
Batch
Parallel
time
Txn 1
Txn 2
Txn 3
Read(A)
Read(B)
Write(C)
Read(A)
Write(B)
Write(A)
Write(B)
Read(C)
Txn 1
Txn 2
Txn 3
Read(A)
Read(B)
Write(C)
Read(A)
Write(B)
Write(A)
Write(B)
Read(C)
Txn 1
Txn 2
Txn 3
Read(A)
Read(B)
Write(C)
Read(A)
Write(B)
Write(A)
Write(B)
Read(C)
Transaction Time
Operations
Log Time
Wait Time
Commit

Scalability and Results



Batch Logging Results



Conclusion and Future Work

- Accomplishments
  - Implemented serial, batch, and parallel logging
  - Determined areas of improvement
  - Tested scalability and efficiency
- Future Goals
  - Gather results for parallel logging
  - Other optimizations for serial logging
  - Log recovery
  - Publish paper

Special thanks to...

- Our mentor, Xiangyao Yu
- Prof. Srini Devadas for his help and guidance
- The PRIMES program

Thank you!