Salt

Combining ACID and BASE in a distributed database

Chao Xie, Chunzhi Su, Manos Kapritsos, Yang Wang, Navid Yaghmazadeh, Lorenzo Alvisi, Prince Mahajan

The University of Texas at Austin

TRANSACTIONS ARE GREAT

Four properties in a single abstraction

Atomicity | Consistency | Isolation | Durability (**ACID**)

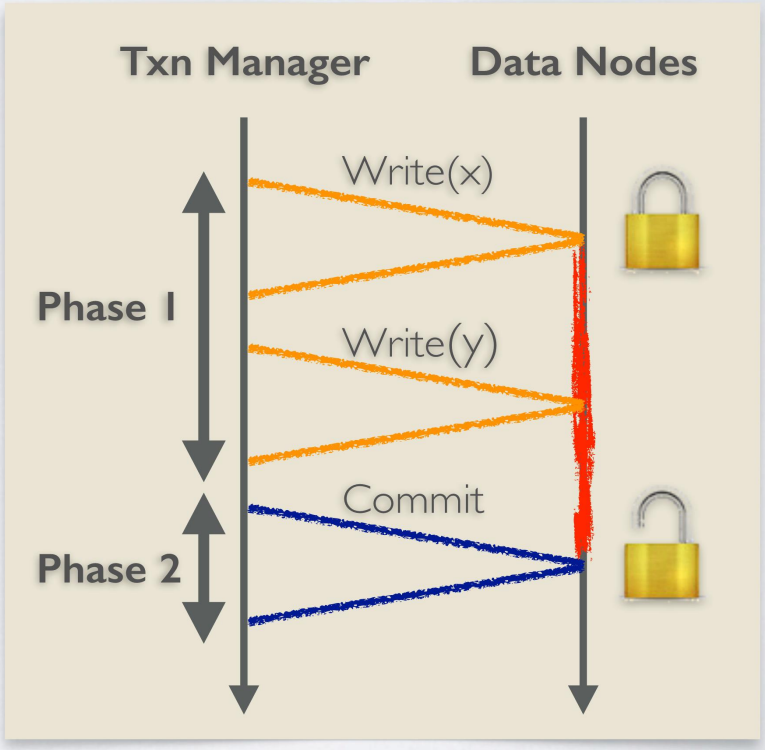
- Ease of programming
- Easy to reason about

Transaction

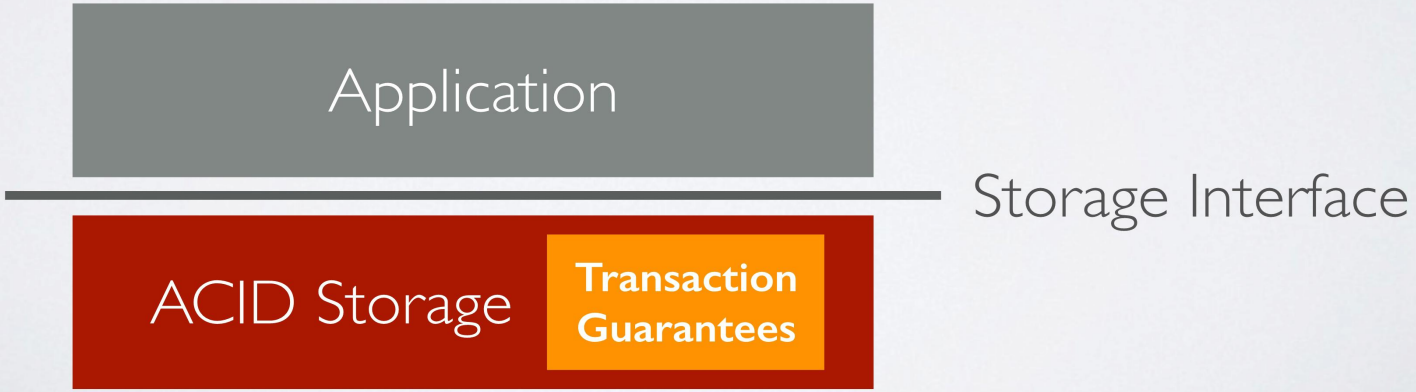
TRANSACTIONS ARE ~~GREAT~~ slow

Concurrency control limits performance

2PC protocol is costly

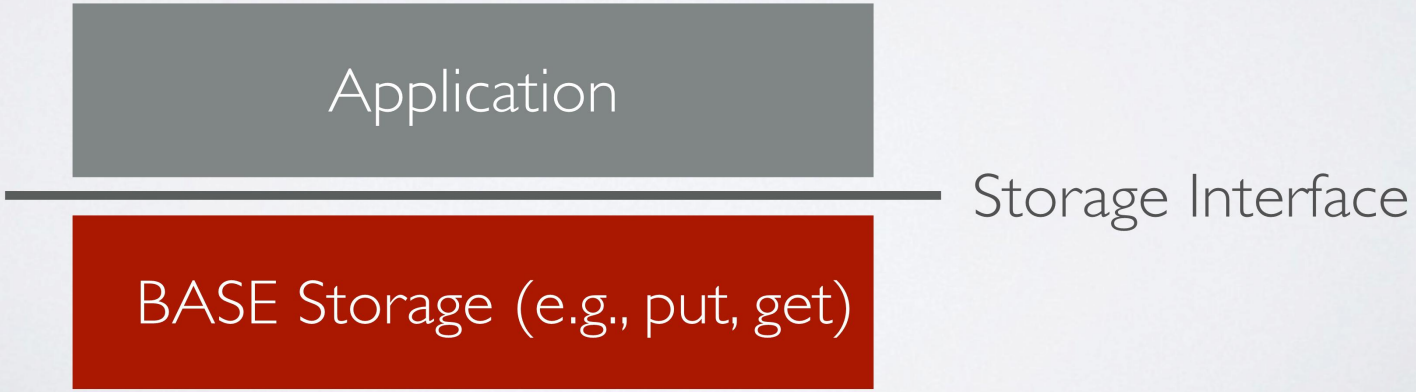


THE ALTERNATIVE: BASE



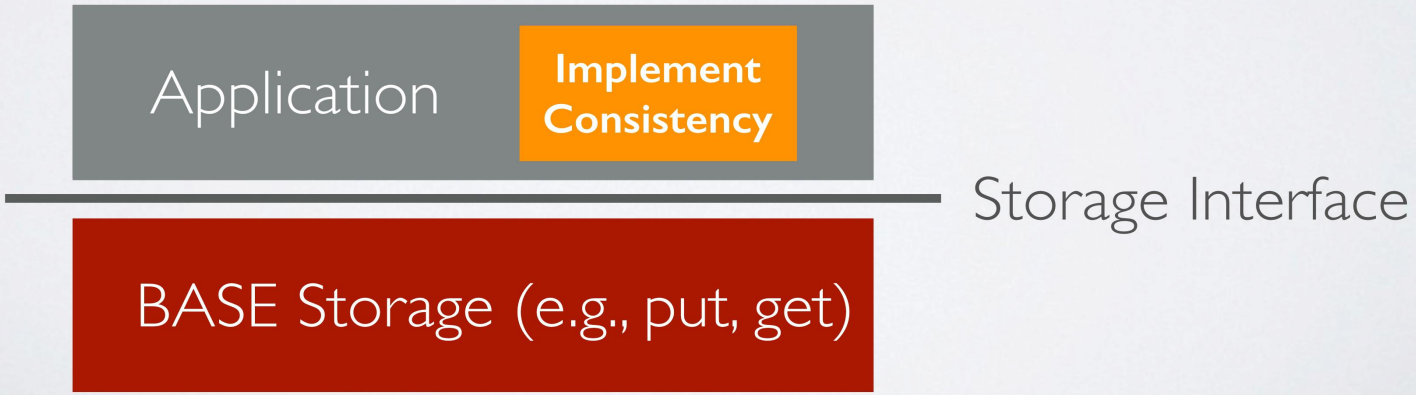
THE ALTERNATIVE: BASE

- Write custom code to get better performance



THE ALTERNATIVE: BASE

- Write custom code to get better performance
- Complexity gets out of control



A Sample Banking Application

```
1   // ACID transfer transaction
2   begin transaction
3     Select bal into @bal from accnts where id = sndr
4     if (@bal >= amt)
5       Update accnts set bal −= amt where id = sndr
6       Update accnts set bal += amt where id = rcvr
7   commit

9   // ACID total−balance transaction
10  begin transaction
11    Select sum(bal) from accnts
12  commit
```

(a) The ACID approach.

```
1   // transfer using the BASE approach
2   begin local−transaction
3     Select bal into @bal from accnts where id = sndr
4     if (@bal >= amt)
5       Update accnts set bal −= amt where id = sndr
6       // To enforce atomicity, we use queues to communicate
7       // between partitions
8       Queue message(sndr, rcvr, amt) for partition(accnts, rc
9   end local−transaction

11  // Background thread to transfer messages to other partition
12  begin transaction // distributed transaction to transfer queu
13    <transfer messages to rcvr>
14  end transaction

16  // A background thread at each partition processes
17  // the received messages
18  begin local−transaction
19    Dequeue message(sndr, rcvr, amt)
20    Select id into @id from accnts where id = rcvr
21    if (@id ≠ 0) // if rcvr's account exists in database
22      Update accnts set bal += amt where id = rcvr
23    else // rollback by sending the amt back to the original sen
24      Queue message(rcvr, sndr, amt) for partition(accnts, sr
25  end local−transaction

27  // total−balance using the BASE approach
28  // The following two lines are needed to ensure correctness o
29  // the total−balance ACID transaction
30  <notify all partitions to stop accepting new transfers>
31  <wait for existing transfers to complete>
```

```
10  begin transaction
11    Select sum(bal) from accnts
12  commit
```

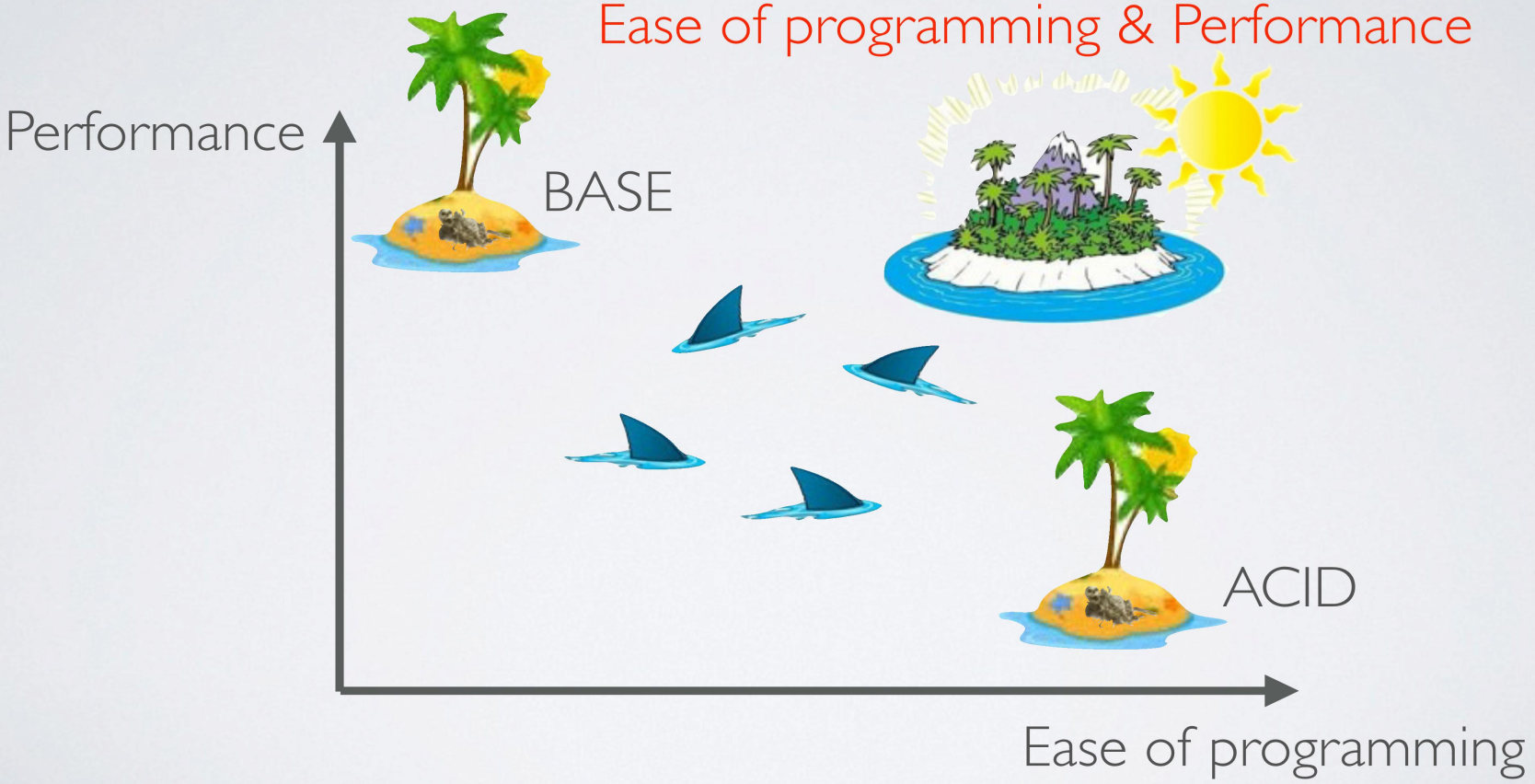
(a) The ACID approach.

```
1   // transfer using the BASE approach
2   begin local−transaction
3     Select bal into @bal from accnts where id = sndr
4     if (@bal >= amt)
5       Update accnts set bal −= amt where id = sndr
6       // To enforce atomicity, we use queues to communicate
7       // between partitions
8       Queue message(sndr, rcvr, amt) for partition(accnts, rcvr)
9   end local−transaction

11  // Background thread to transfer messages to other partitions
12  begin transaction // distributed transaction to transfer queued msgs
13    <transfer messages to rcvr>
14  end transaction

16  // A background thread at each partition processes
17  // the received messages
18  begin local−transaction
19    Dequeue message(sndr, rcvr, amt)
20    Select id into @id from accnts where id = rcvr
21    if (@id ≠ 0) // if rcvr's account exists in database
22      Update accnts set bal += amt where id = rcvr
23    else // rollback by sending the amt back to the original sender
24      Queue message(rcvr, sndr, amt) for partition(accnts, sndr)
25  end local−transaction

27  // total−balance using the BASE approach
28  // The following two lines are needed to ensure correctness of
29  // the total−balance ACID transaction
30  <notify all partitions to stop accepting new transfers>
31  <wait for existing transfers to complete>
32  begin transaction
33    Select sum(bal) from accnts
34  end transaction
35  <notify all partitions to resume accepting new transfers>
```

(b) The BASE approach.

Fig. 1: A simple banking application with two implementations: (a) ACID and (b) BASE

A STARK CHOICE



Vilfredo Pareto

20% of the causes
account for
80% of the effects

NOT ALL TRANSACTIONS ARE CREATED EQUAL

20% of the causes
account for
80% of the effects

- Many transactions are not run frequently
- Many transactions are lightweight

AN OPPORTUNITY

- Identify critical transactions
- BASE-ify only critical transactions

SALT

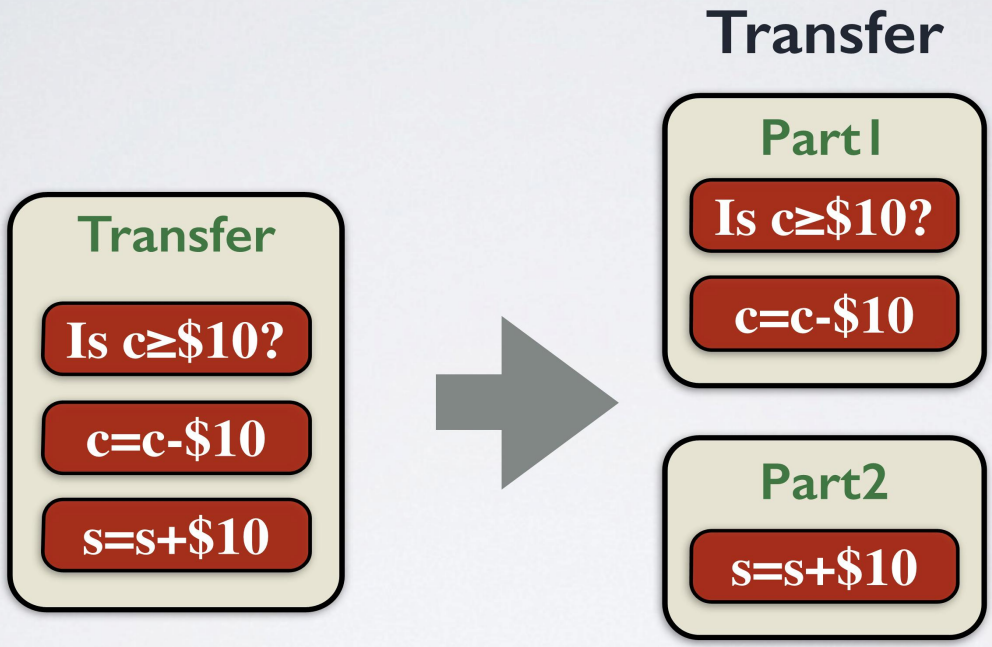
Motivation

Base Transactions & Salt Isolation

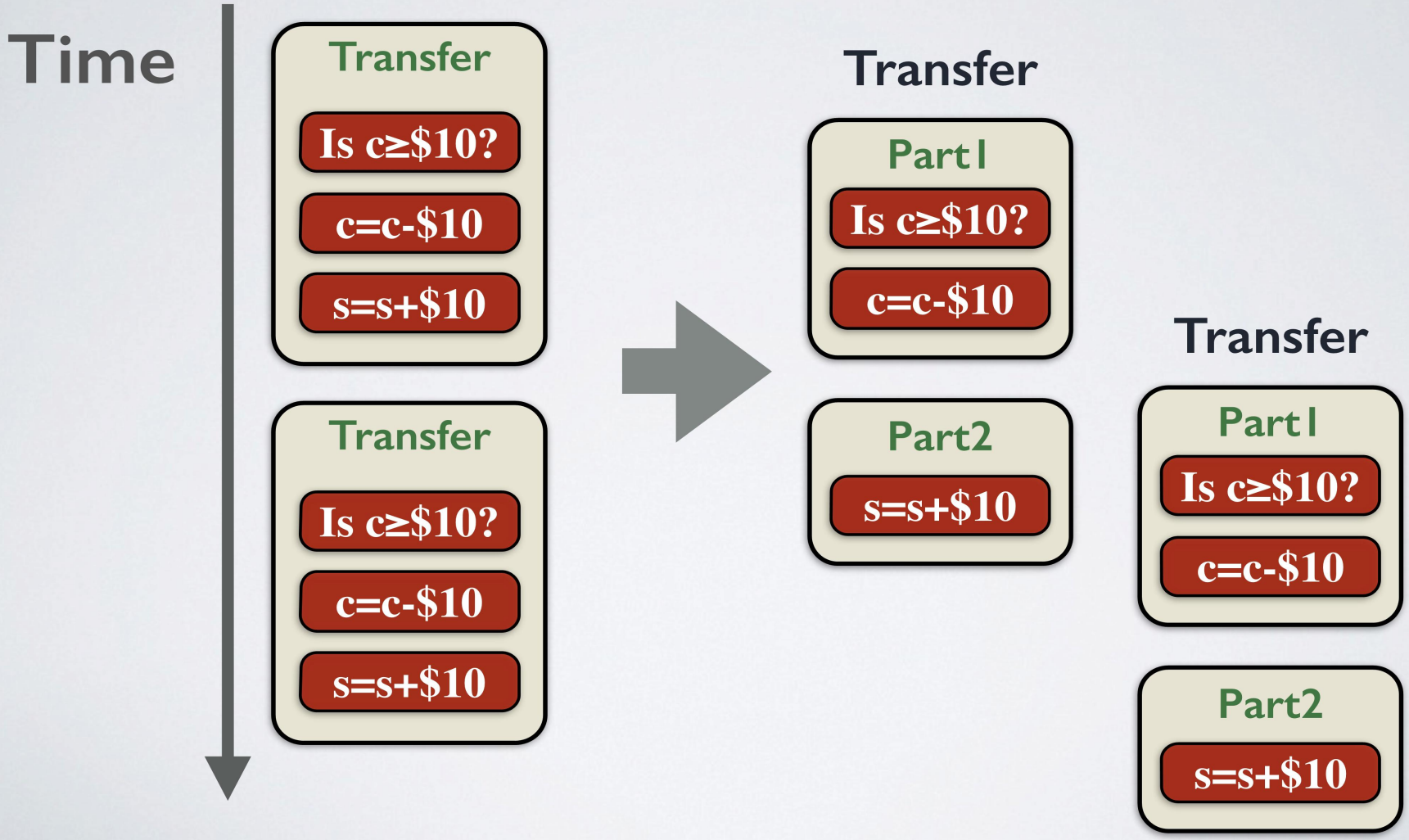
Achieving Salt Isolation

Evaluation

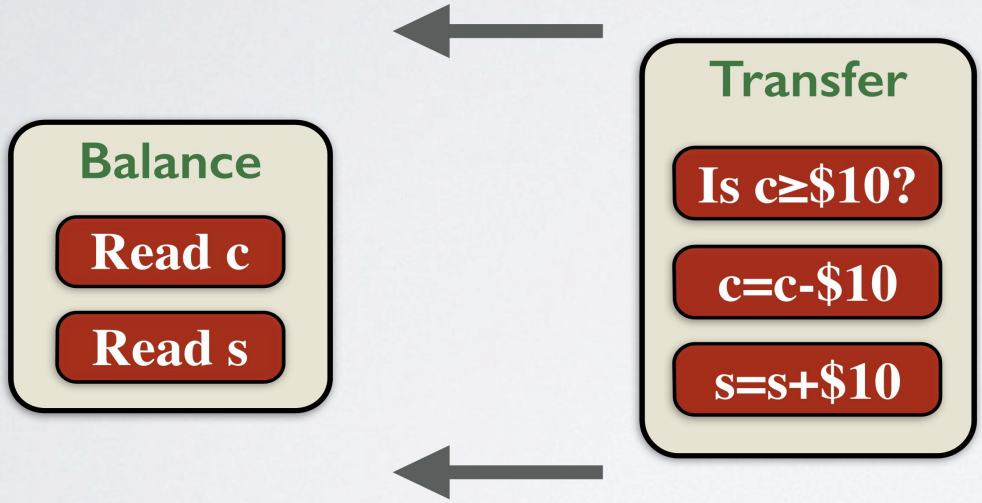
MORE CONCURRENCY !



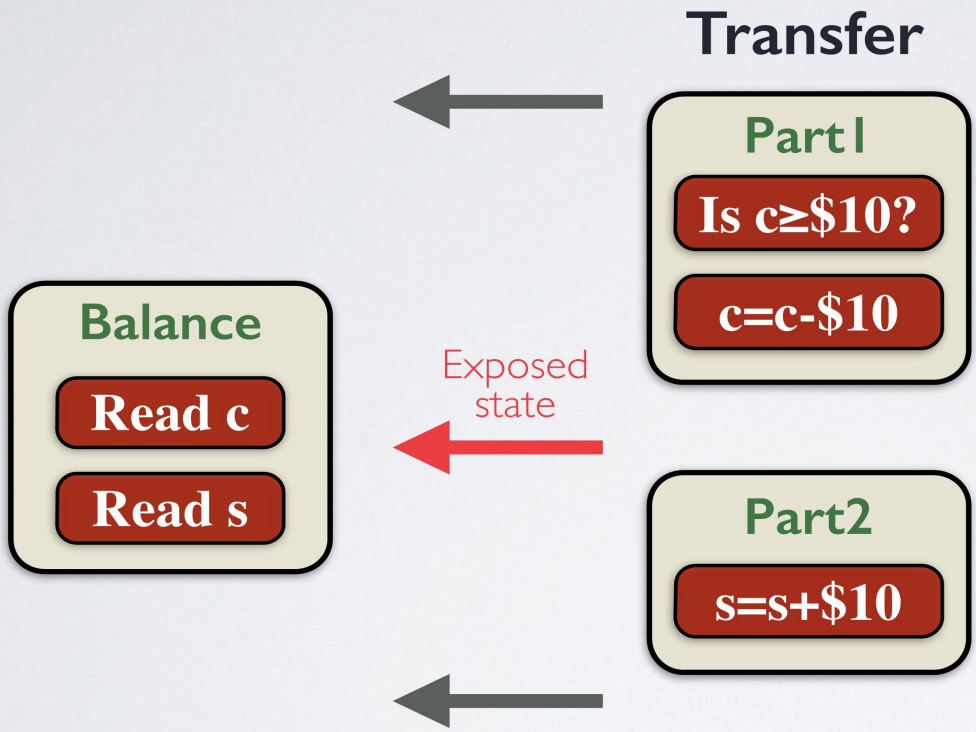
MORE CONCURRENCY !



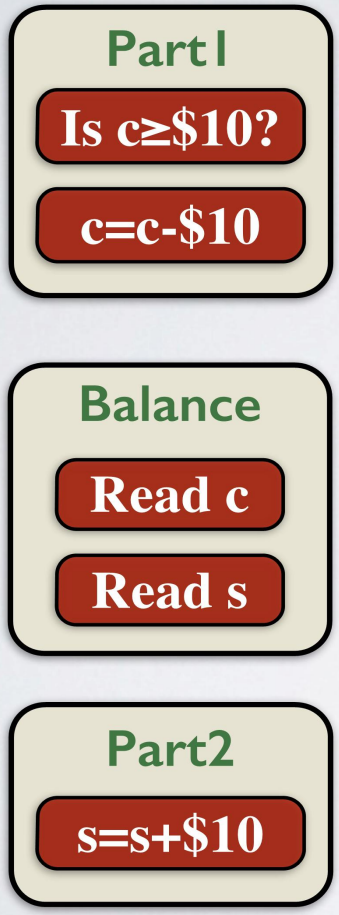
CORRECTNESS AT RISK



CORRECTNESS AT RISK

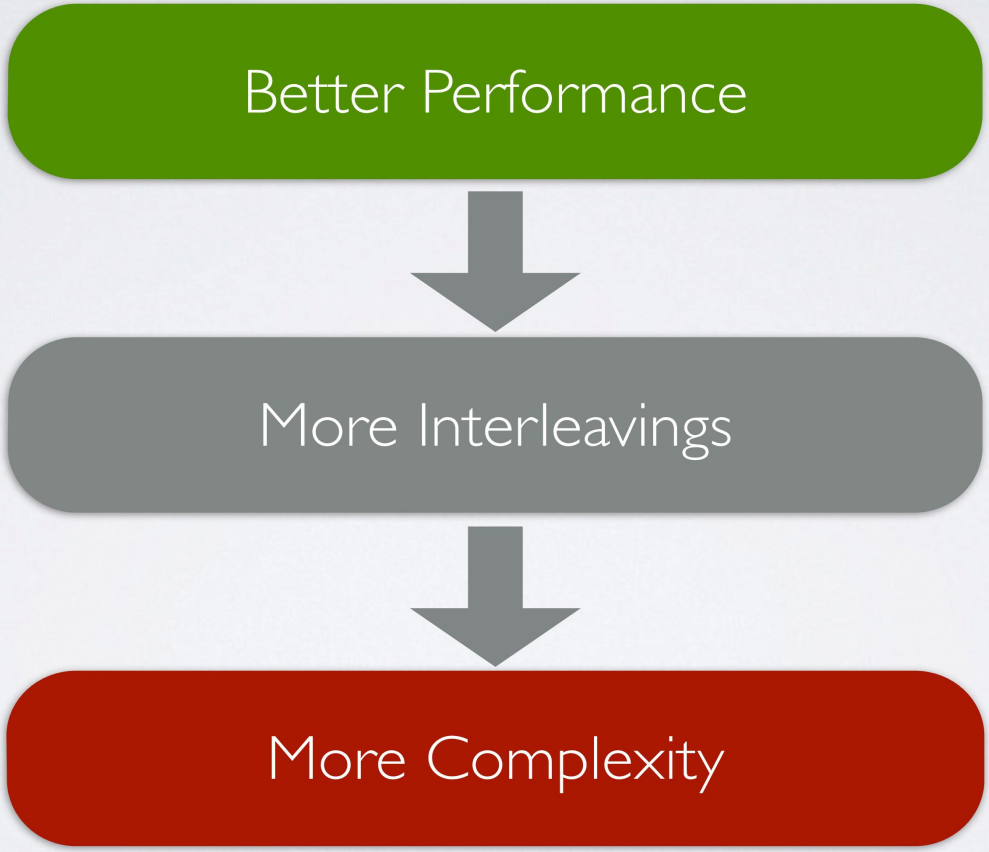


CORRECTNESS AT RISK

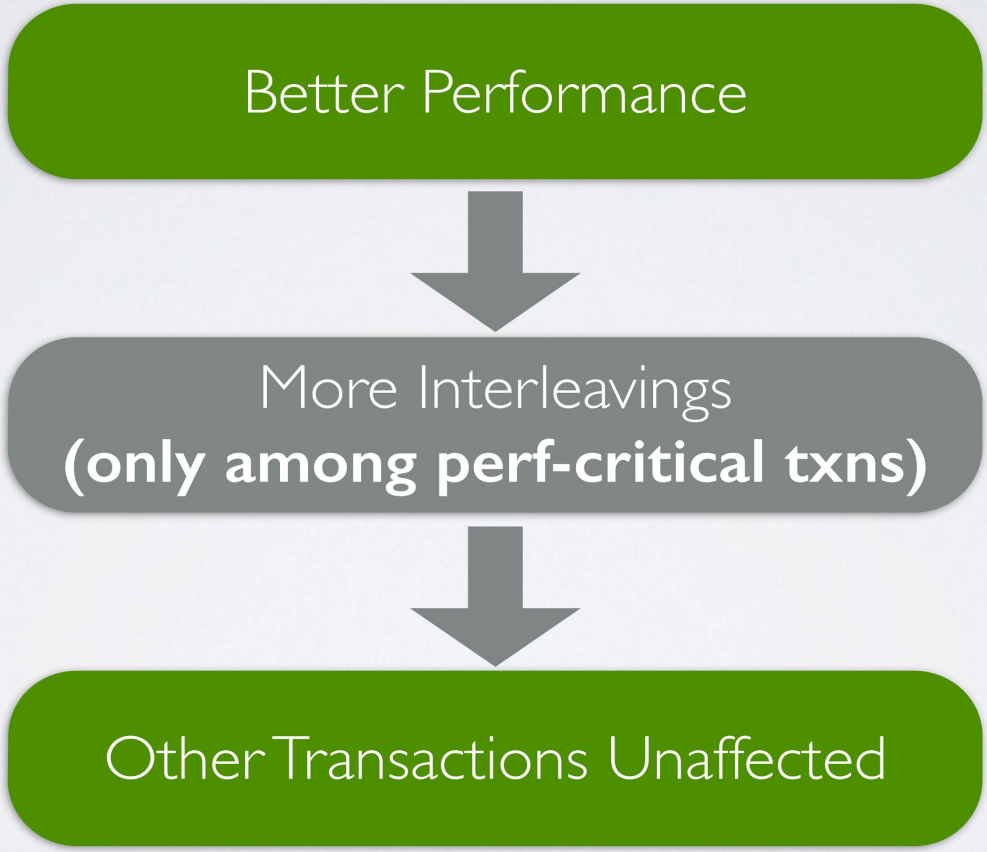


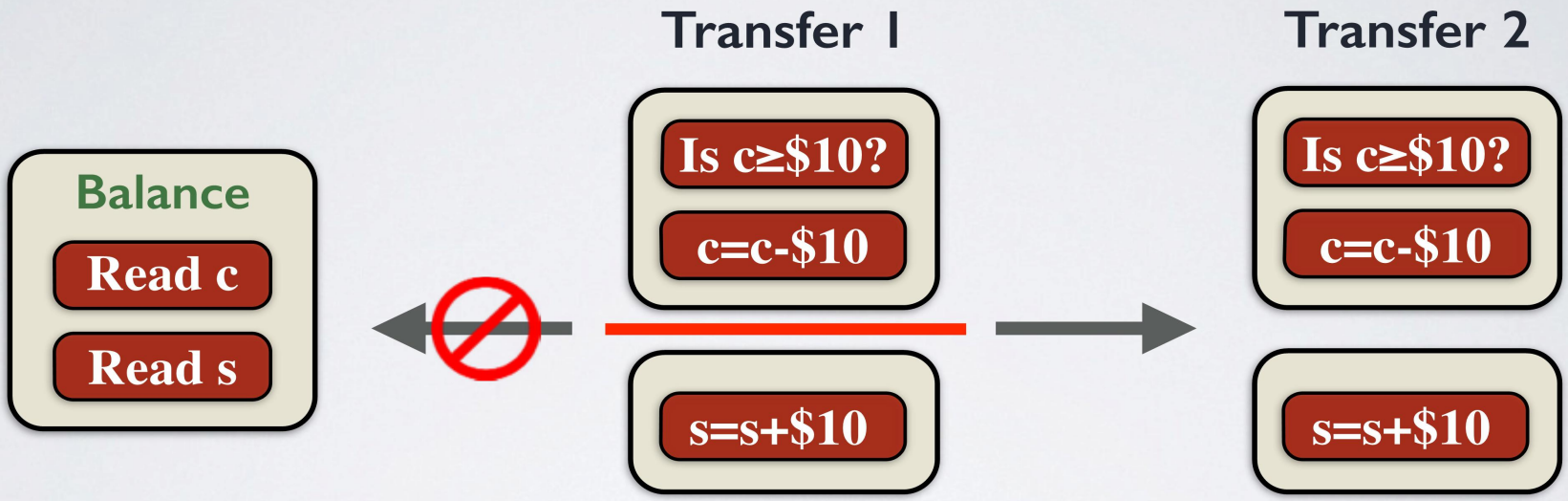
Finer Isolation for one transaction may affect all transactions!!

Performance vs Complexity

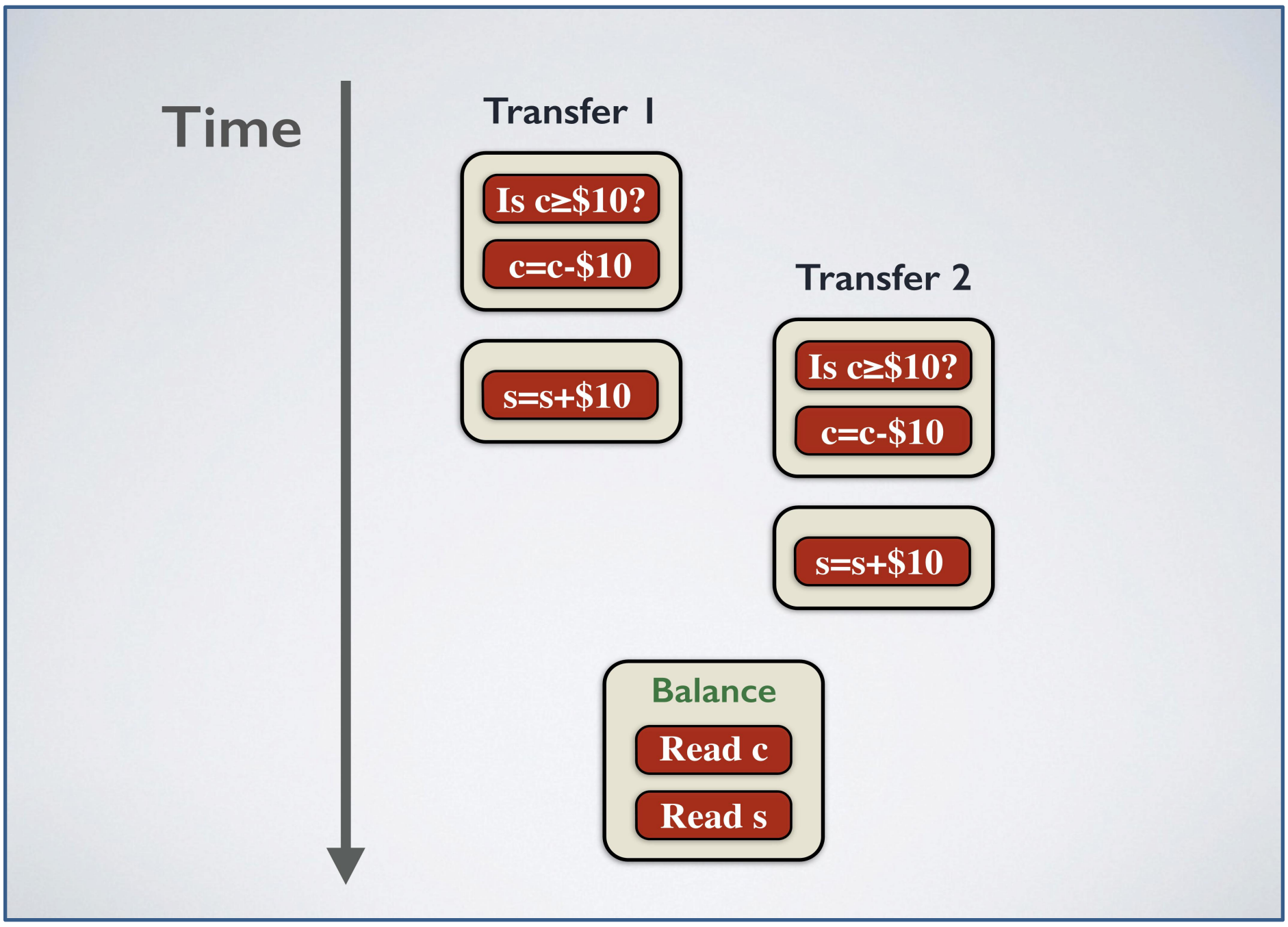


Performance vs Complexity

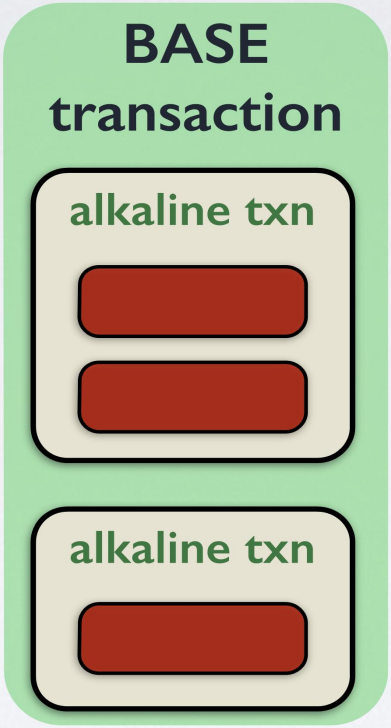




Behaves differently
when interacting with different transactions

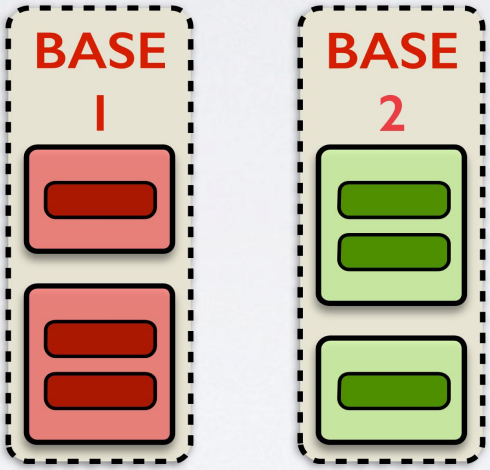

Time
Transfer 1
Is c≥$10?
c=c-$10
s=s+$10
Transfer 2
Is c≥$10?
c=c-$10
s=s+$10
Balance
Read c
Read s

BASE TRANSACTION



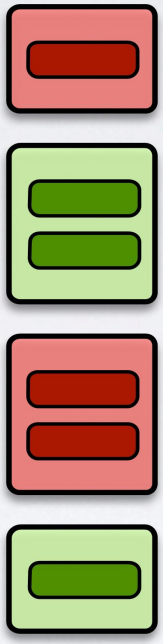
Behaves differently
when interacting with different transactions

BASE INTERACT WITH BASE



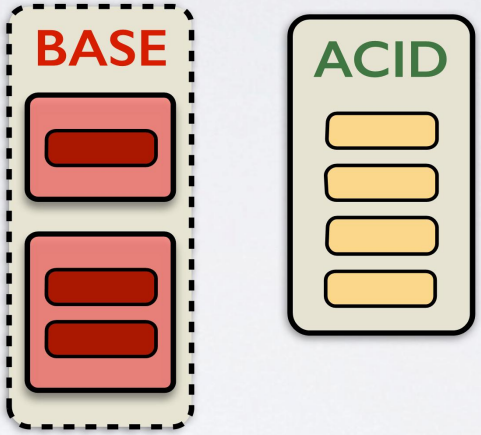
Fine Isolation granularity
between BASE transactions

BASE INTERACT WITH BASE



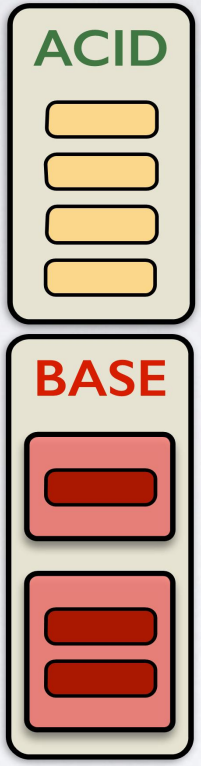
Fine Isolation granularity
between BASE transactions

BASE INTERACT WITH ACID



BASE transactions provide coarse Isolation granularity to ACID transactions

BASE INTERACT WITH ACID



BASE transactions provide coarse Isolation granularity to ACID transactions

BASE TRANSACTION

```
1   // BASE transaction: transfer
2   begin BASE transaction
3     try
4       begin alkaline−subtransaction
5         Select bal into @bal from accnts where id = sndr
6         if (@bal >= amt)
7           Update accnts set bal −= amt where id = sndr
8       end alkaline−subtransaction
9     catch (Exception e) return // do nothing
10    if (@bal < amt) return // constraint violation
11    try
12      begin alkaline−subtransaction
13        Update accnts set bal += amt where id = rcvr
14      end alkaline−subtransaction
15    catch (Exception e) //rollback if rcvr not found or timeout occurs
16      begin alkaline−subtransaction
17        Update accnts set bal += amt where id = sndr
18      end alkaline−subtransaction
19  end BASE transaction

21  // ACID transaction: total−balance (unmodified)
22  begin transaction
23    Select sum(bal) from accnts
24  commit
```

Fig. 2: A Salt implementation of the simple banking application

SALT ISOLATION

BASE transactions: multiple granularities of Isolation

To BASE transactions:
a sequence of small
ACID transactions

To ACID transactions:
a single, monolithic
ACID transaction

Performance & Ease of Programming

SALT

Motivation

Base Transactions & Salt Isolation

Achieving Salt Isolation

Evaluation

ONE MECHANISM

LOCKS

Three flavors

ACID locks
Alkaline locks
Saline locks

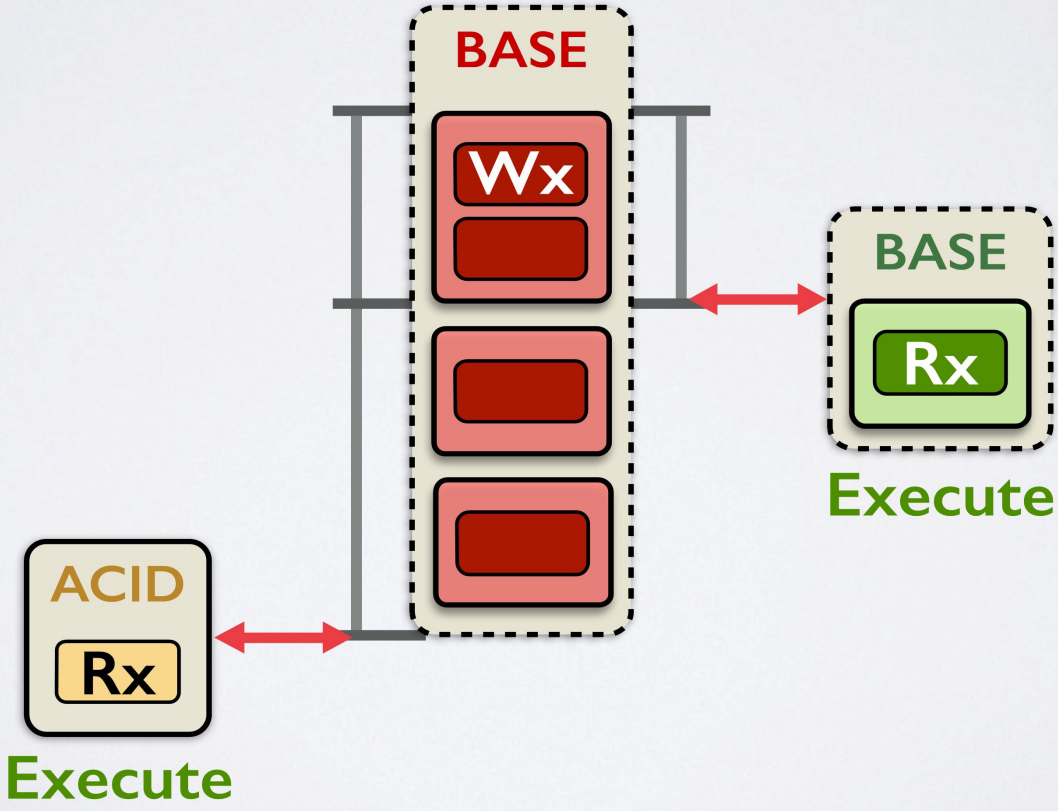
ACID LOCKS

Execute

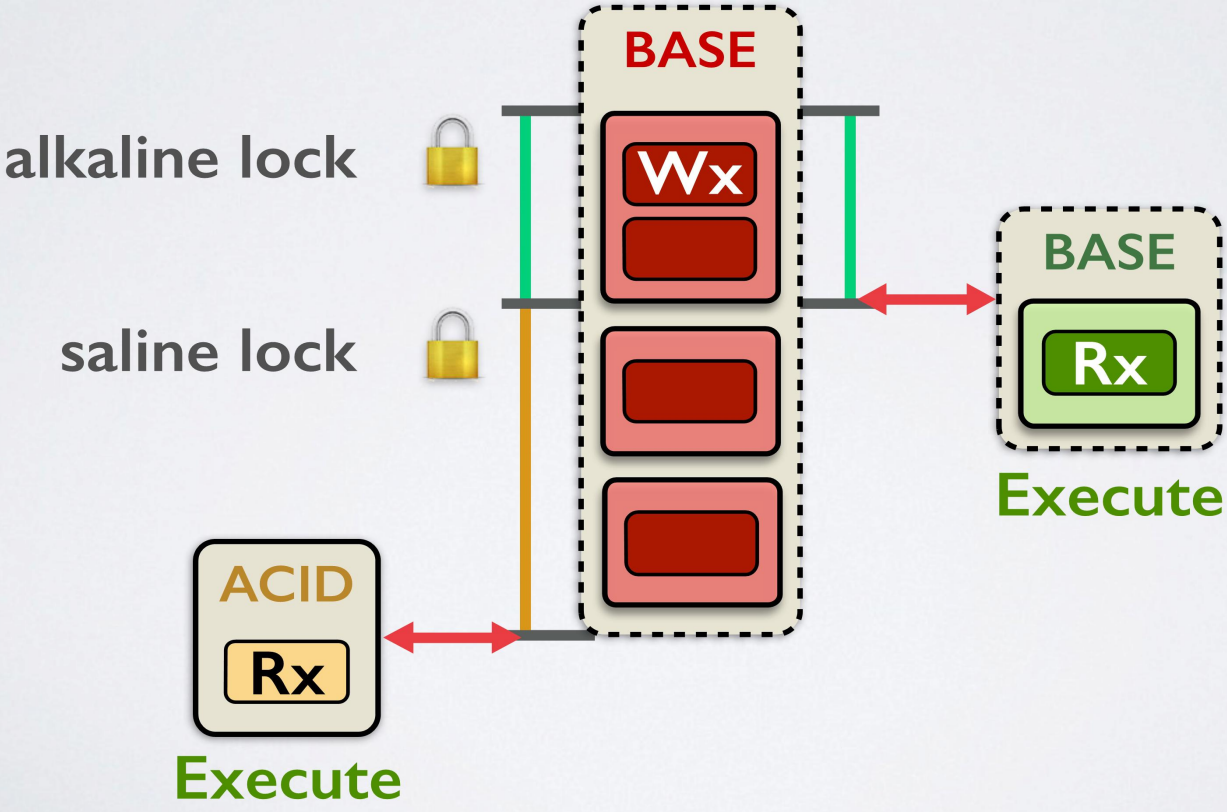
	AC-R	AC-W
AC-R	✓	X
AC-W	X	X

Lock Table

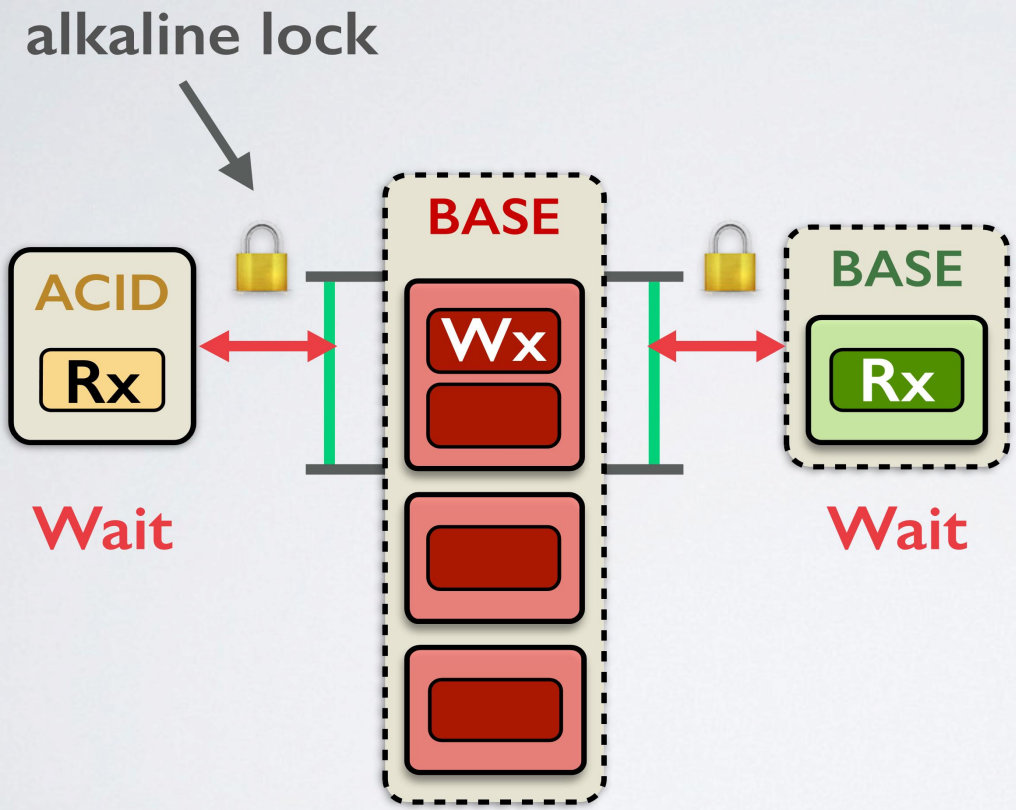
LOCKS?



LOCKS?


BASE
alkaline lock
Wx
saline lock
BASE
Rx
Execute
ACID
Rx
Execute

ALKALINE LOCKS

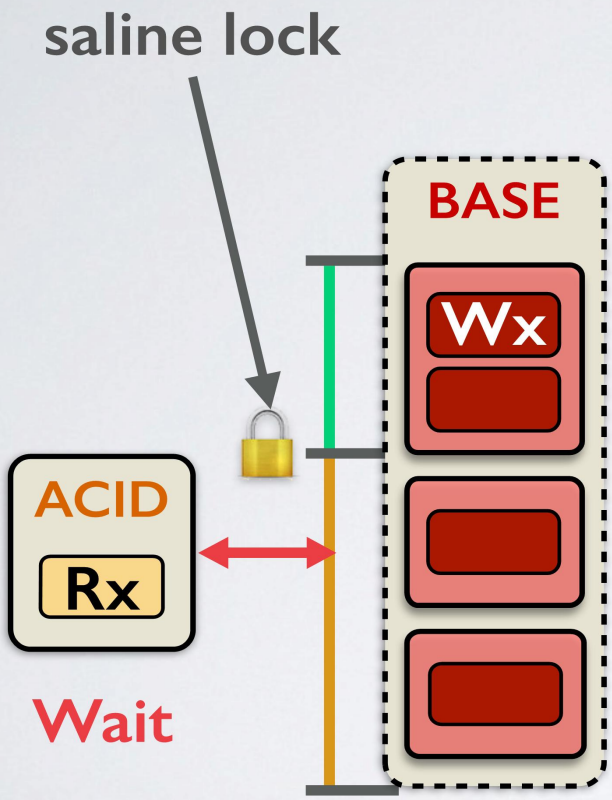


	AC-R	AC-W	alk-R	alk-W
AC-R	✓	X	✓	X
AC-W	X	X	X	X
alk-R	✓	X	✓	X
alk-W	X	X	X	X

Lock Table

Conflict with ACID & alkaline locks

SALINE LOCKS

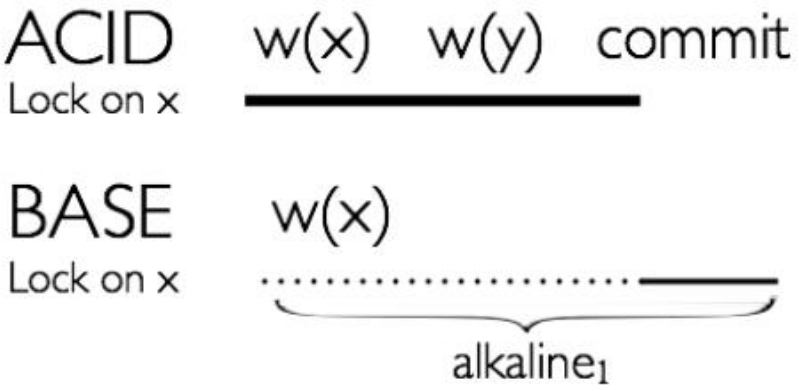


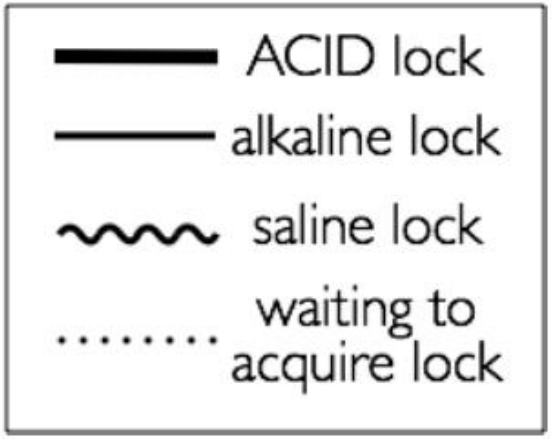
	AC-R	AC-W	alk-R	alk-W	sal-R	sal-W
AC-R	√	X	√	X	√	X
AC-W	X	X	X	X	X	X
alk-R	√	X	√	X	√	√
alk-W	X	X	X	X	√	√
sal-R	√	X	√	√	√	√
sal-W	X	X	√	√	√	√

Lock Table

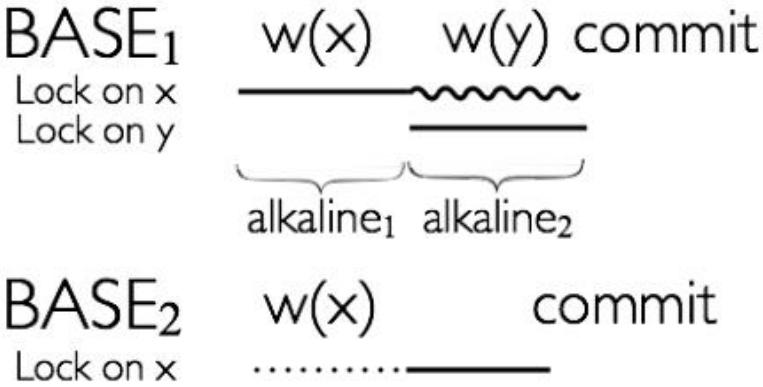
Conflict only with ACID locks

Examples

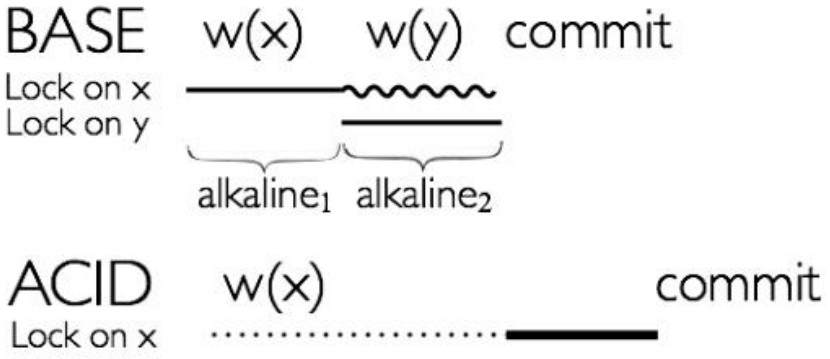




(a) *BASE* waits until *ACID* commits.

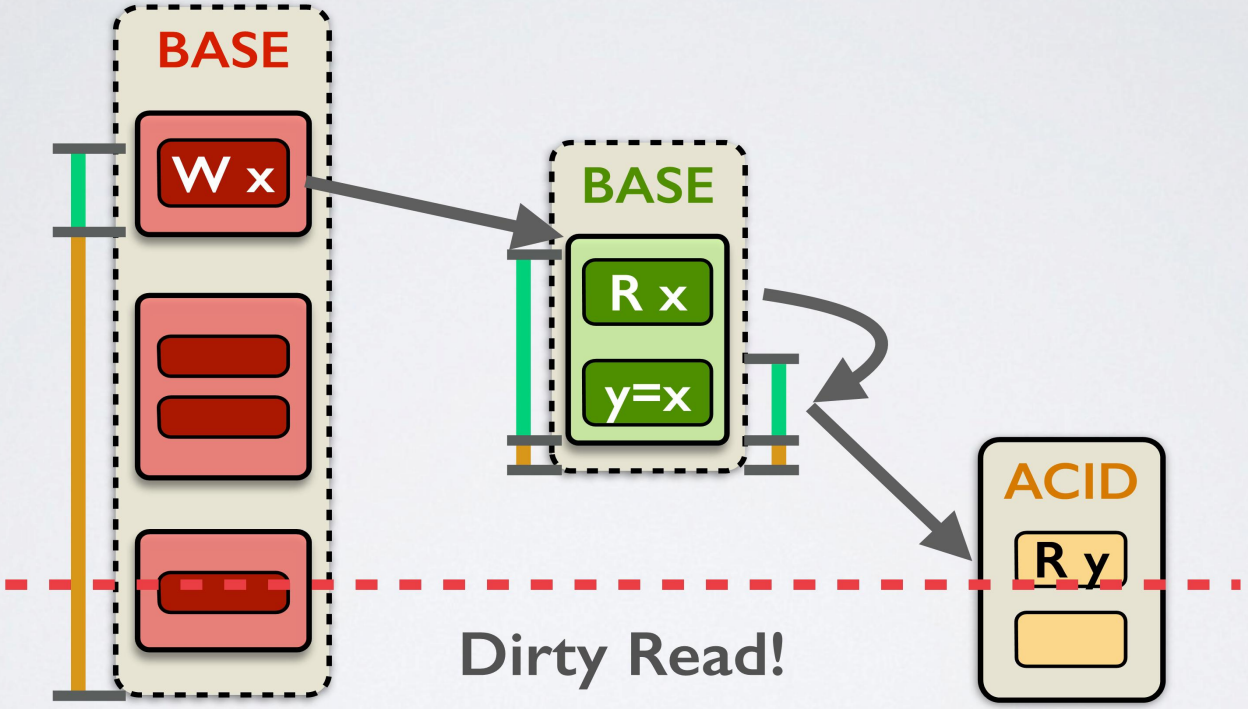


(b) $BASE_2$ waits only for $alkaline_1$...



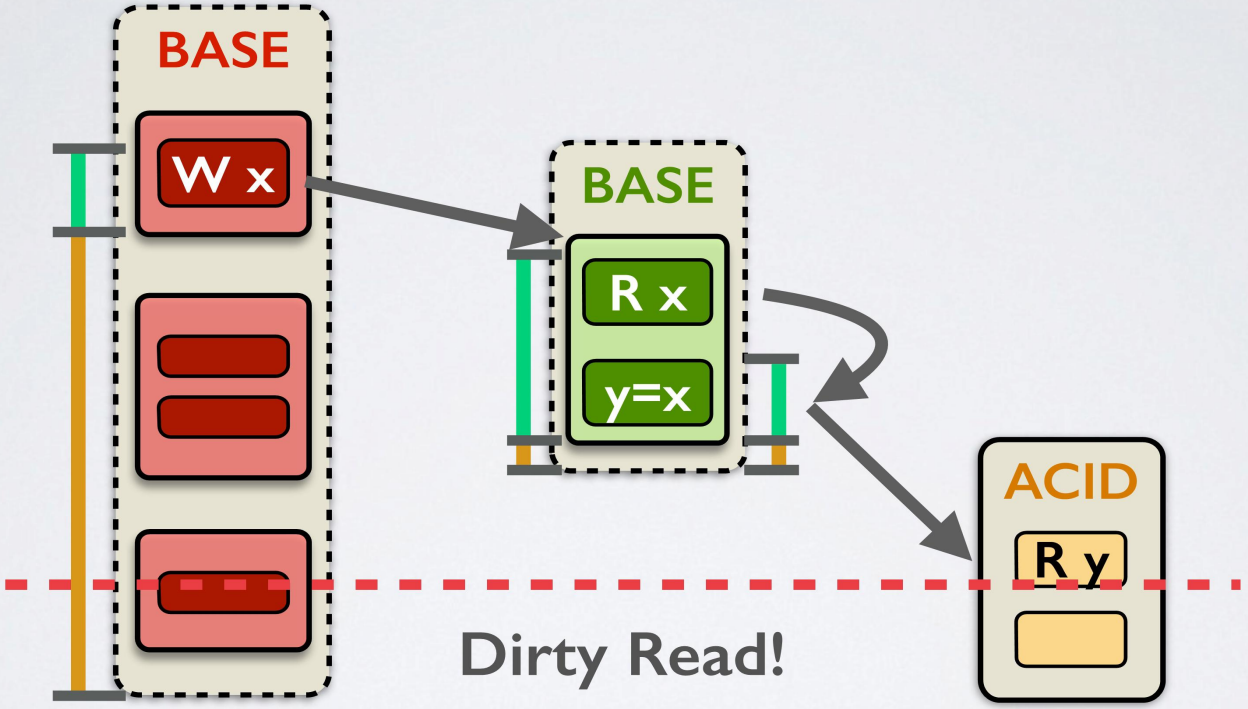
(c) ... but *ACID* must wait all of *BASE* out.

A SUBTLE PROBLEM



ACID reads uncommitted value of x!

A SUBTLE PROBLEM



For the solution, please read our paper

SOLUTION

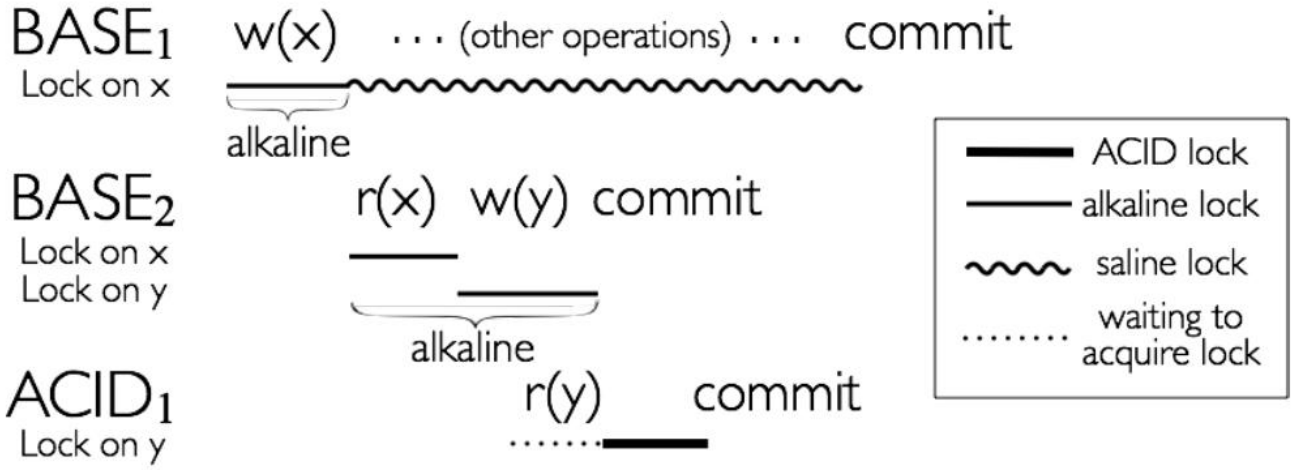


Fig. 4: $ACID_1$ indirectly reads the uncommitted value of x.

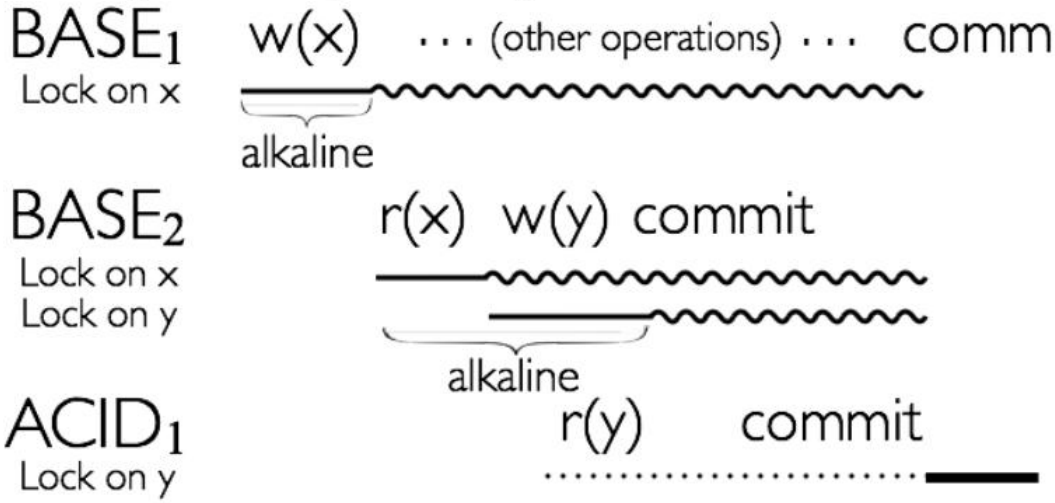


Fig. 5: How Salt prevents indirect dirty reads.

THE BOTTOM LINE

Guarantee

Salt prevents all ACID transactions from being affected by BASE transactions either directly or indirectly.

SALT

Motivation

Base Transactions & Salt Isolation

Achieving Salt Isolation

Evaluation

QUESTIONS TO ANSWER

What is the performance gain of Salt compared to ACID?

Can we get most performance gain compared to the BASE approach?

EXPERIMENTAL SETUP

Configuration

- Emulab Cluster (Dell Power Edge R710)
- 10 shards, 3-way replicated

Workloads

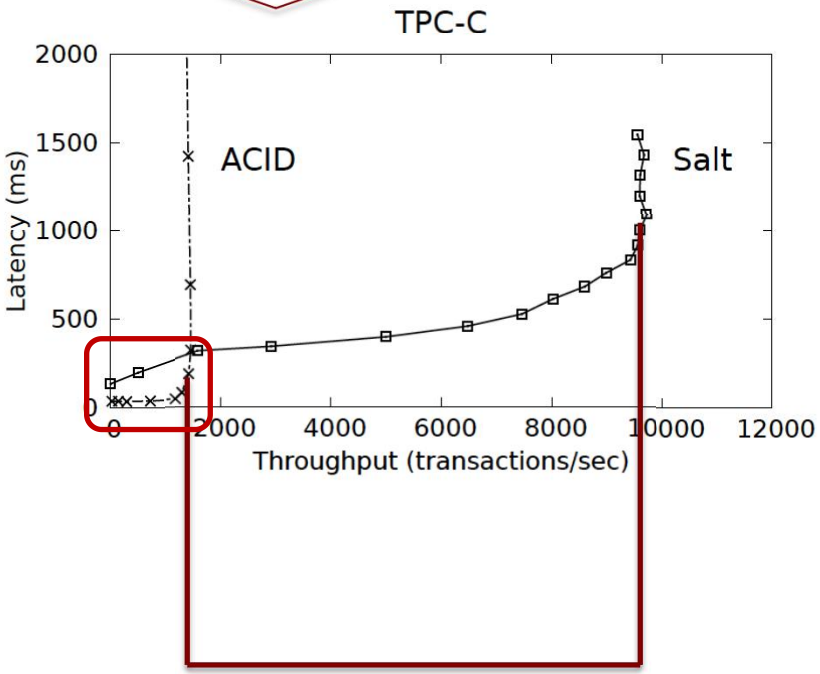
- TPC-C
- Fusion Ticket
- Microbenchmarks

Applications used for the experiments

- TPC-C benchmark : database performance standard
  - 5 transactions (new-order, payment, stock-level, order-status and delivery)
  - New-order and payment are responsible for 43.5% of total number of transactions
- Fusion Ticket: open source ticketing application in PHP and MySQL
  - Includes several transactions
  - Create-order and payment most frequent and performance critical
- Micro benchmark:
  - Contention ratio
  - Contending transaction position

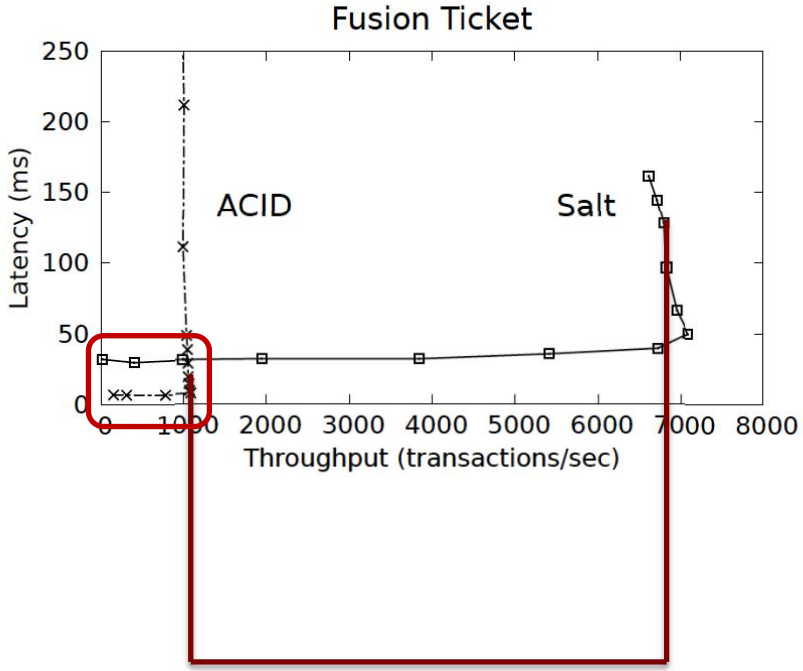
Did Salt win the award? (Yes…)

Base-ify: new-order and payment



6.6x higher

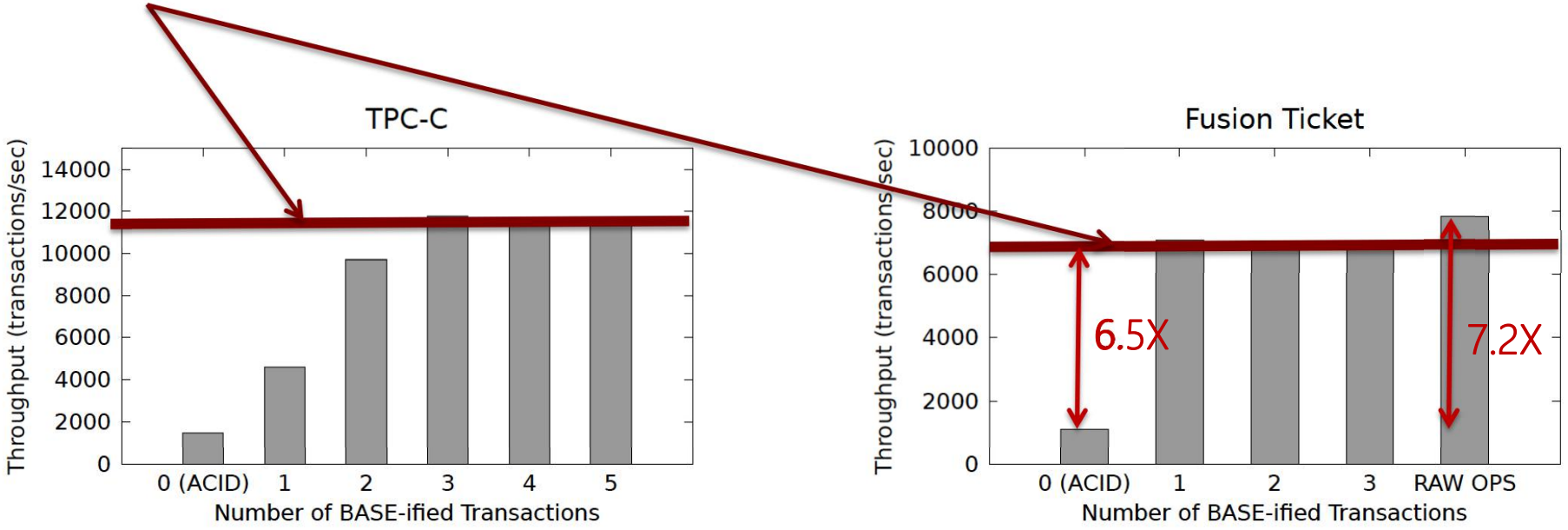
Base-ify: Create-order and payment



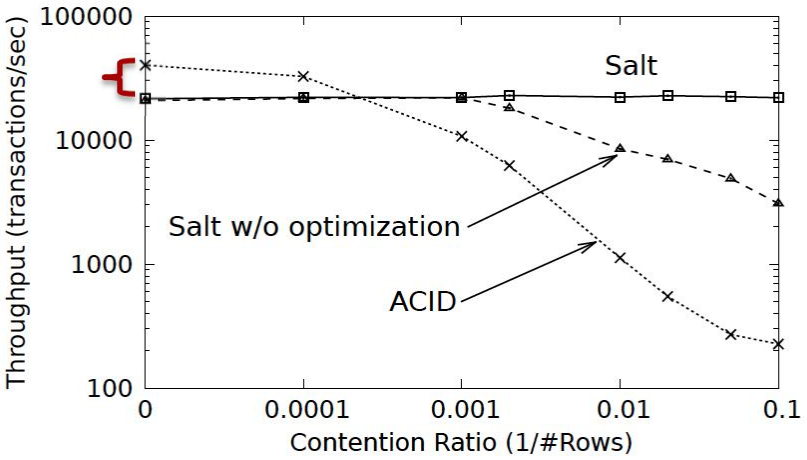
6.5x higher

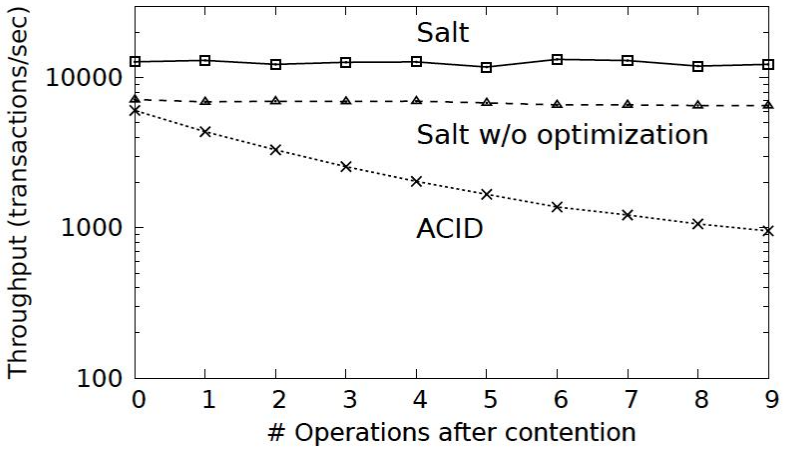
Did Salt win the award? (Contd…)

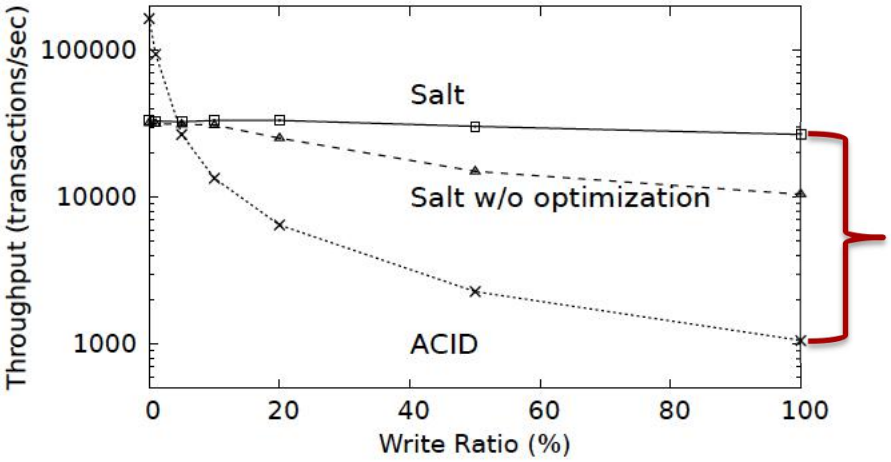
Performance gain becomes stagnant



Did Salt win the award? (Contd…)







Discussion & Q-A

- Pros:
  - Performance gain is drastic
  - Control in the hands of the programmer

- Cons:
  - Cassandra allows batching of commands into groups to form transactions called atomic batches
  - Leverage the power of MySQL clusters - MySQL does not allow nested transactions and MySQL now has several NoSQL/NewSQL like features
  - Programming complexity - the difficulty of identifying the transactions that need to be converted and verifying the correctness is a daunting task

RELATE WORK

Optimizing ACID Performance!

- H-Store, Granola, F1, Sagas, Transaction Chain, Calvin …

BASE with enhanced semantics (e.g., partition local transactions)

- ElasTras, G-Store, Megastore …

Making Geo-Replicated Systems Fast as Possible, Consistent when Necessary (OSDI'12,Cheng Li)

Coordination Avoidance in Database Systems (VLDB'15,Peter Baills)

Thank You!