Lazy Evaluation of Transactions in Database Systems

Jose M. Faleiro[◆], Alexander Thomson[✪], Daniel J. Abadi[◆]

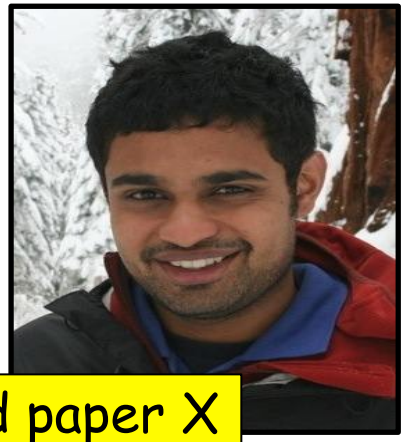
[◆]Yale University, [✪]Google

Paper “X” looks relevant, can you add it to related work?

Just read through the whole thing and added to related work. Our work is more ACIDic

In reality…

- Read only abstract to give a sensible response
- Keep “sticky” note on desk to later read paper and write related work.



Read paper X

Add to related work

Read paper X

Add to related work

Read paper Y

Add to related work

. . .

Read paper Z

Add to related work

Hey, can you check-in your additions to related work? I can't see your changes…

Read paper X

Add to related work

Read paper Y

Add to related work

. . .

Read paper Z

Add to related work

Read paper X

Add to related work

Read paper Y

Add to related work

. . .

Read paper Z

Add to related work

- Previously said I did all the work. Can't feign ignorance.
- Use information on stickies to read papers and finish writing related work.

Read paper X

Add to related work

Read paper Y

Add to related work

. . .

Read paper Z

Add to related work

Lazy Database Systems

- Promise to commit/abort while only partially executing txns
- Keep promises while maintaining ACID and serializability

Lying Considered ~~Harmful~~ Useful

- Flexibility to execute transactions when most favorable
  - Cache Locality
  - Load Balancing
  - Contention Reduction

Can’t Keep Lying Forever

Must satisfy “external” reads

Talk Outline

- **Lazy Database Design**
- Benefits of Lazy Execution
- Experimental Evaluation
- Conclusion

Customer Orders Product

Update product inventory

Compute discounted price

Update customer bill

Customer Orders Product

Update product inventory

Compute discounted price

Update customer bill

```
Read Product record

Product count -= 1

If Product count < 0:
   ABORT()
```

Customer Orders Product

```
Read Product record

Product count -= 1

If Product count < 0:
      ABORT()
```

Update product inventory

Compute discounted price

```
Read daily discount

Compute discounted price
```

Update customer bill

Customer Orders Product

Update product inventory

Compute discounted price

Update customer bill

```
Read Product record

Product count -= 1

If Product count < 0:
      ABORT()

Read daily discount

Compute discounted price
```

```
Read customer's monthly
bill record

Customer's monthly bill +=
   Discounted price

COMMIT()
```

We Ask the Question

Can we:
return commit/abort promise …

without executing to completion…

while maintaining ACID and serializability?

```
Read Product record

Product count -= 1

If Product count < 0:
  ABORT()

Read daily discount

Compute discounted price

Read customer's monthly
bill record

Customer's monthly bill +=
   Discounted price

COMMIT()
```

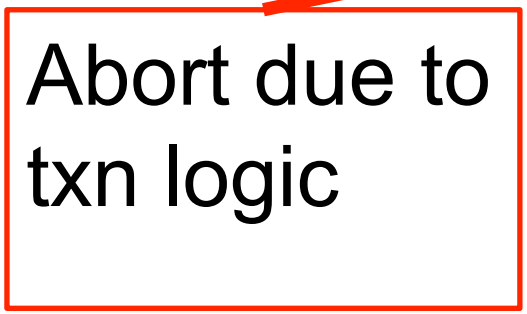
Our Solution

- Split up transactions into two parts
  - Give users the illusion that execution is atomic
- First part produces commit/abort promise
  - Executes immediately
- Second part does everything else
  - Execution is deferred

How to Split a Transaction?

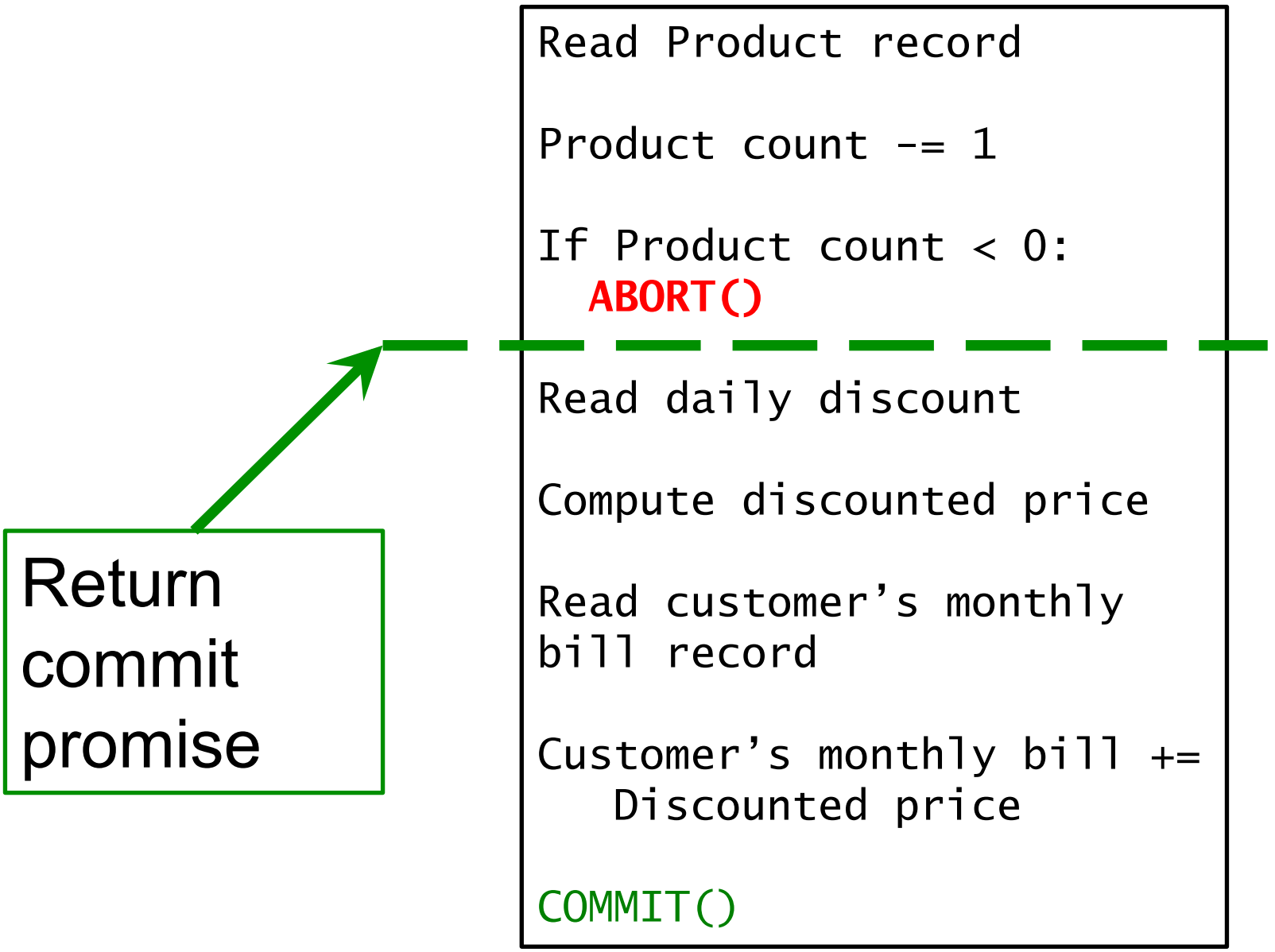
```
Read Product record

Product count -= 1

If Product count < 0:
  ABORT()

Read daily discount

Compute discounted price

Read customer's monthly
bill record

Customer's monthly bill +=
   Discounted price

COMMIT()
```

Splitting a Transaction

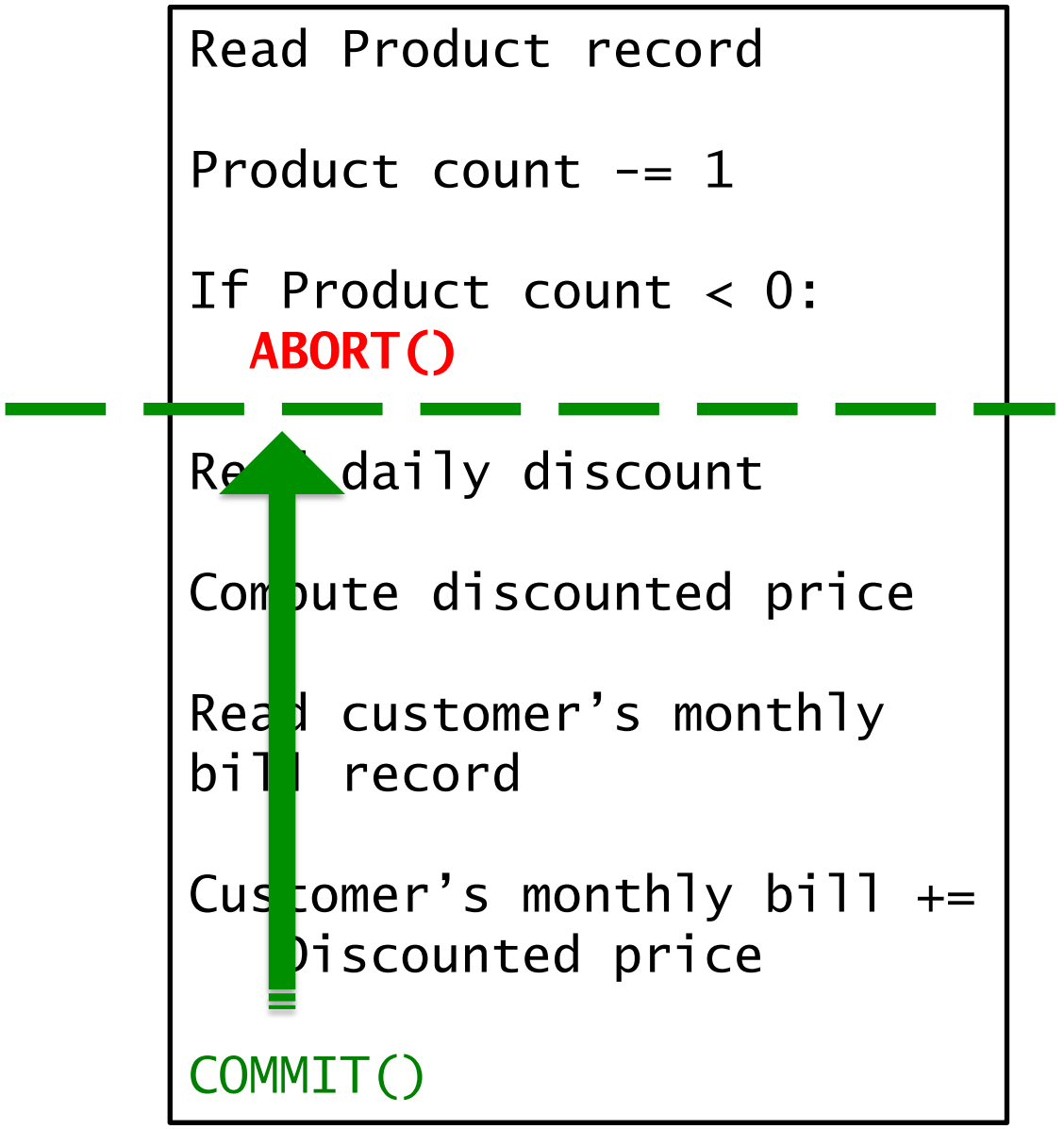



```
Read Product record

Product count -= 1

If Product count < 0:
  ABORT()

Read daily discount

Compute discounted price

Read customer's monthly
bill record

Customer's monthly bill +=
   Discounted price

COMMIT()
```

Splitting a Transaction



Splitting a Transaction



Splitting a Transaction

```
Read Product record

Product count -= 1

If Product count < 0:
  ABORT()

COMMIT_PROMISE()
```

```
Read daily discount

Compute discounted price

Read customer's monthly
bill record

Customer's monthly bill +=
   Product price
```

Splitting a Transaction

Execute immediately

```
Read Product record

Product count -= 1

If Product count < 0:
   ABORT()

COMMIT_PROMISE()
```

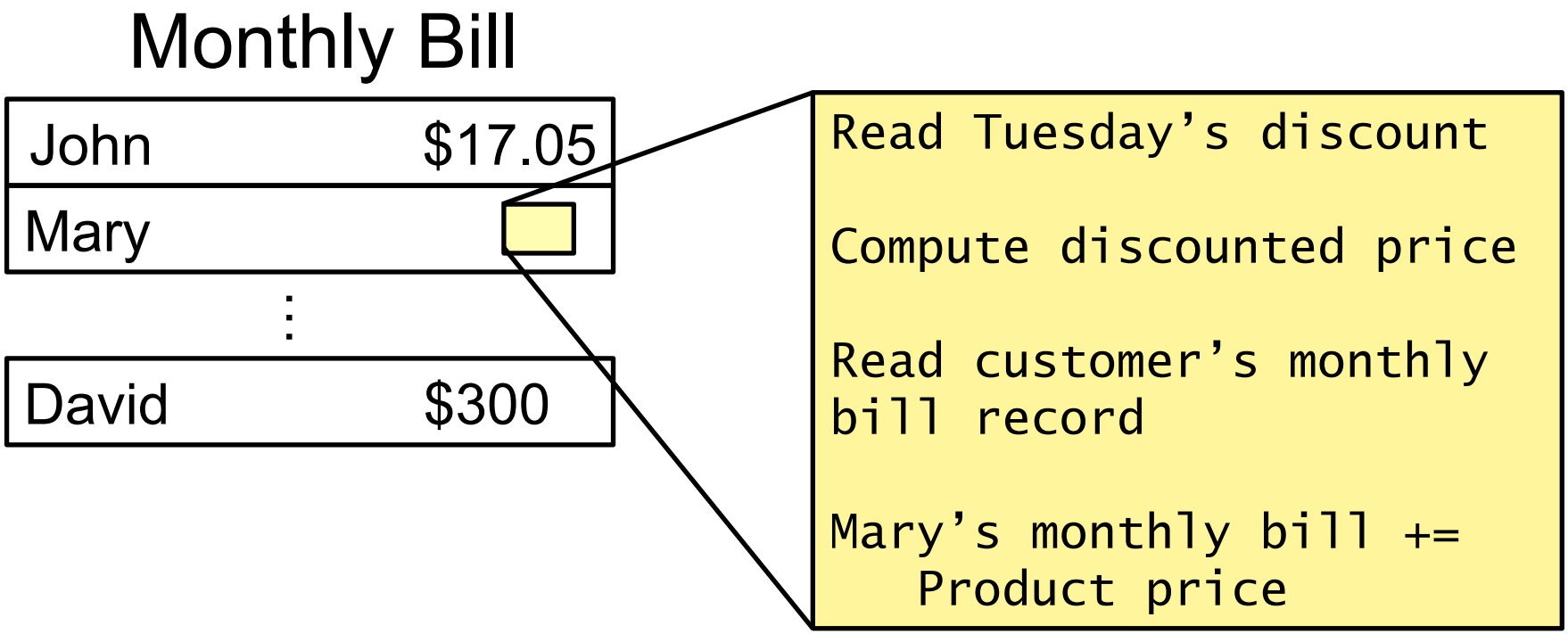
Defer execution

```
Read daily discount

Compute discounted price

Read customer's monthly
bill record

Customer's monthly bill +=
    Product price
```

Dealing with Deferred Logic

- Don’t write out actual record values
- ***Blindly*** insert placeholders corresponding to unexecuted deferred logic
  - Similar to stickies on my desk as reminders

Dealing with Deferred Logic

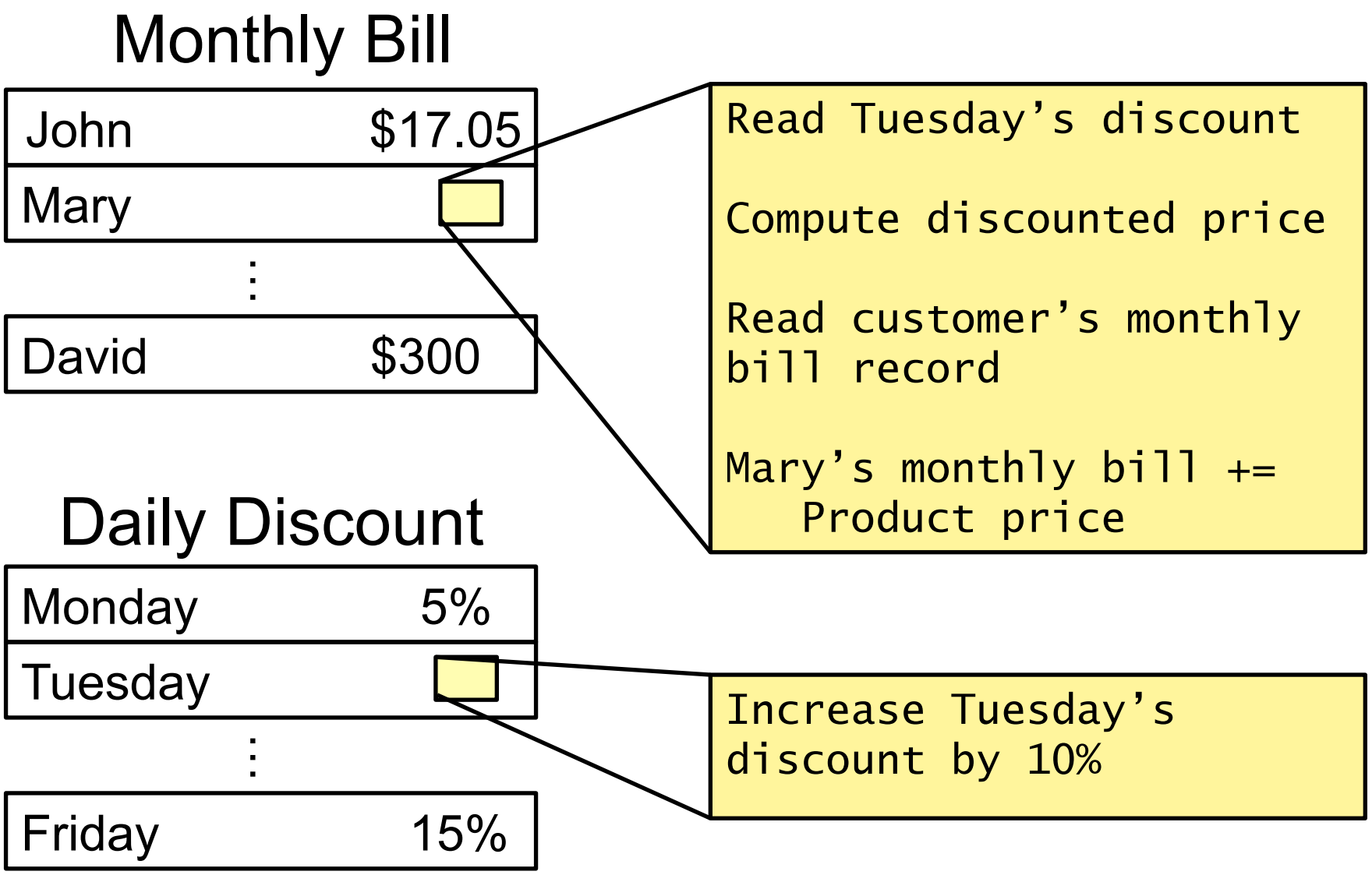


Processing External Reads

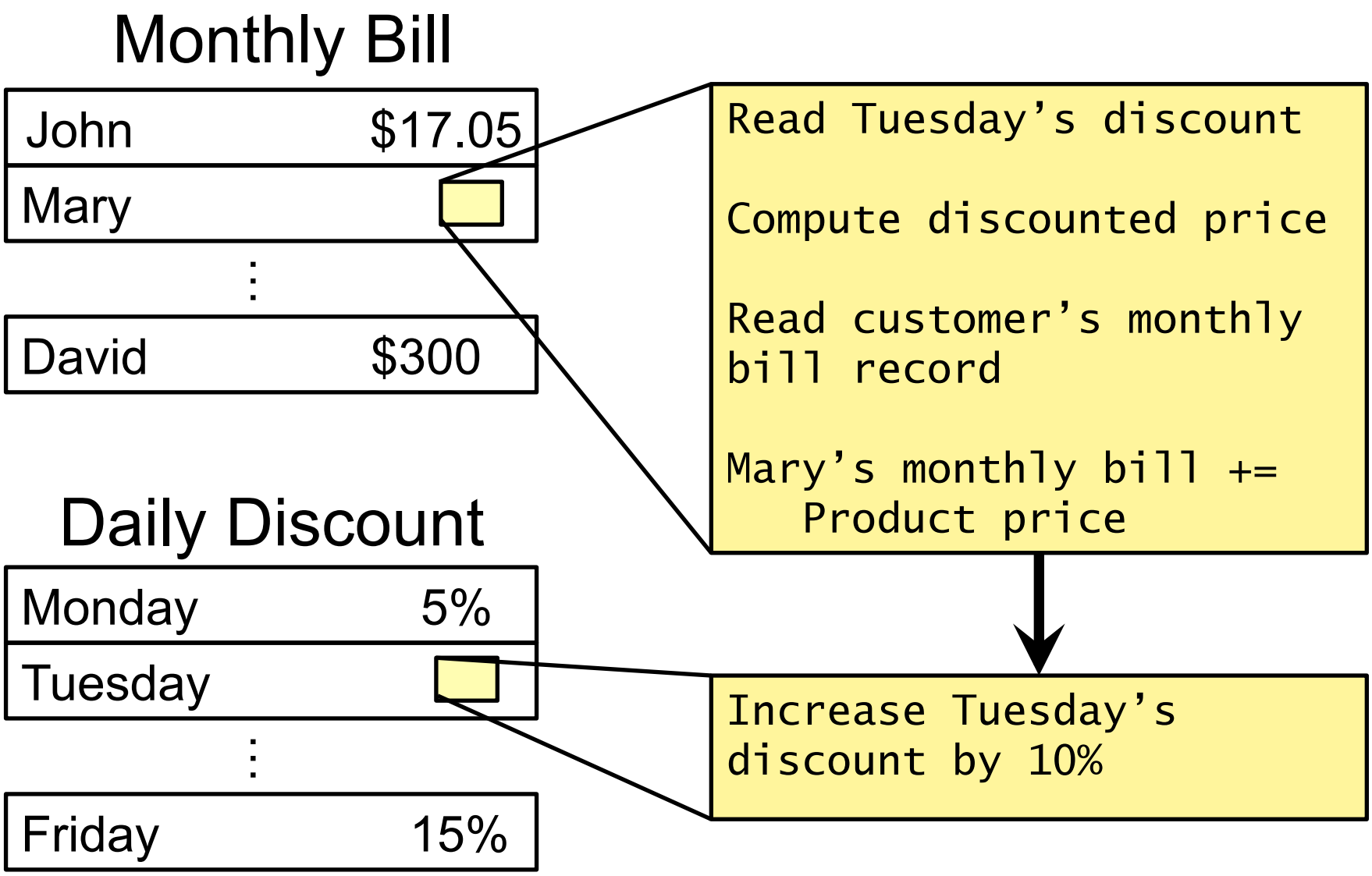
- External read may depend on a deferred transaction

- Deferred transaction itself may have dependencies

Dealing with Deferred Logic

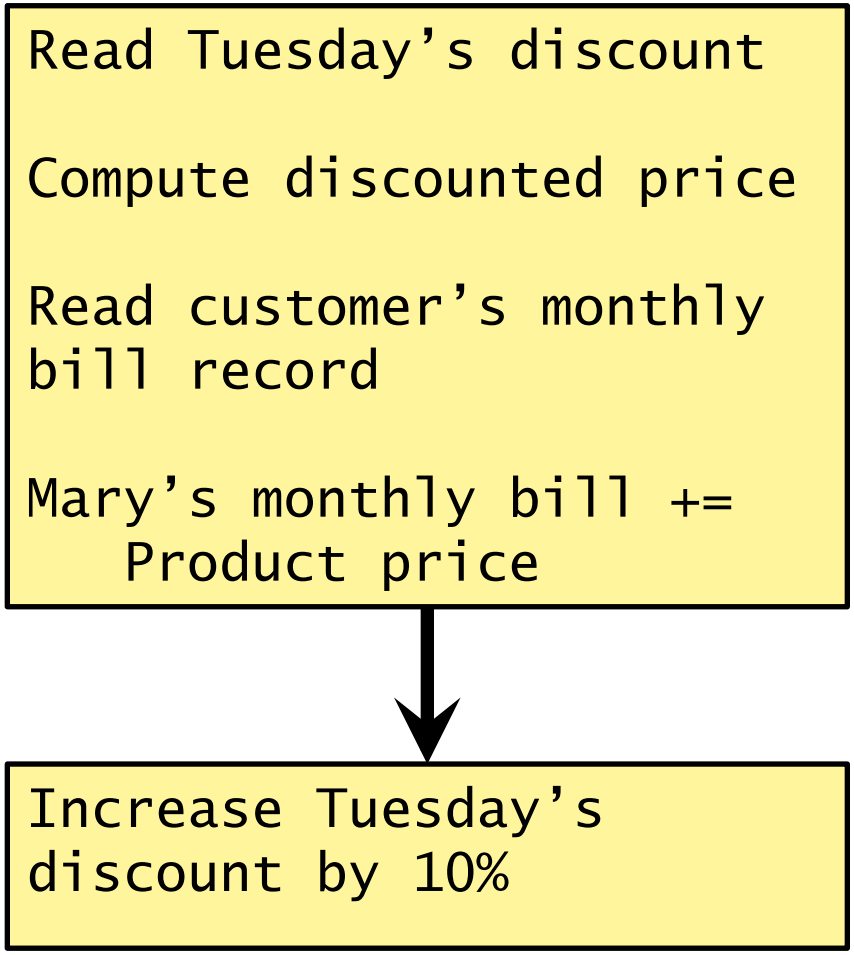


Dealing with Deferred Logic



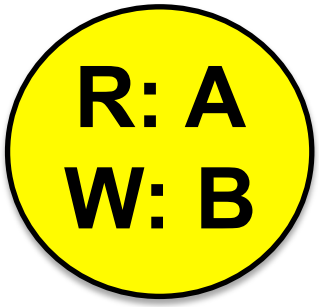
Dealing with Deferred Logic

Deferred pieces of logic are implicitly ordered

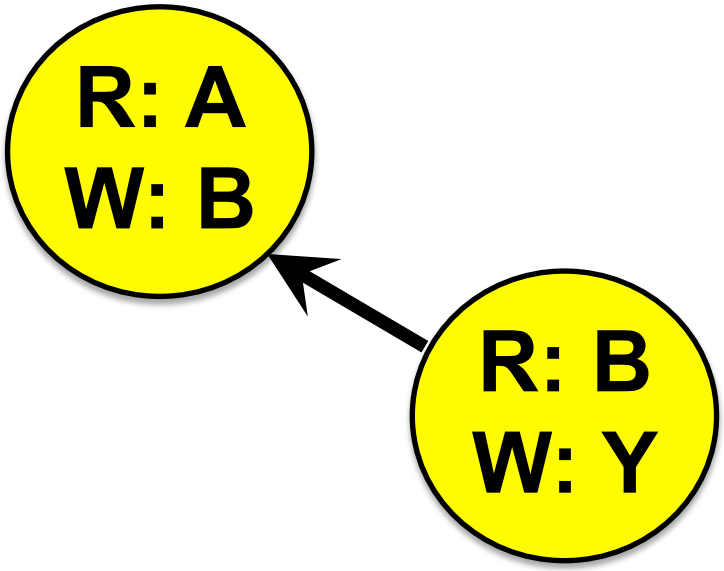


Dependency Graph of Transactions

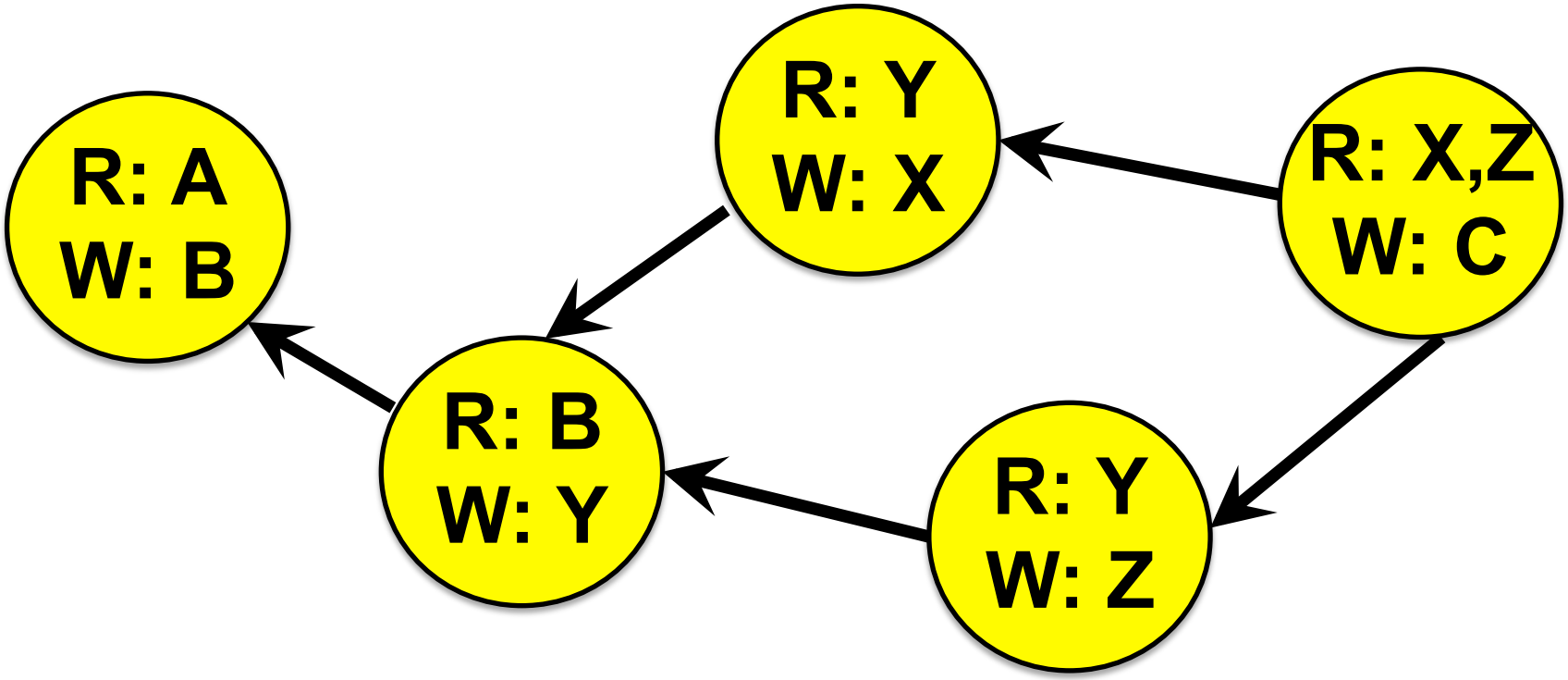
Dependency Graph of Transactions



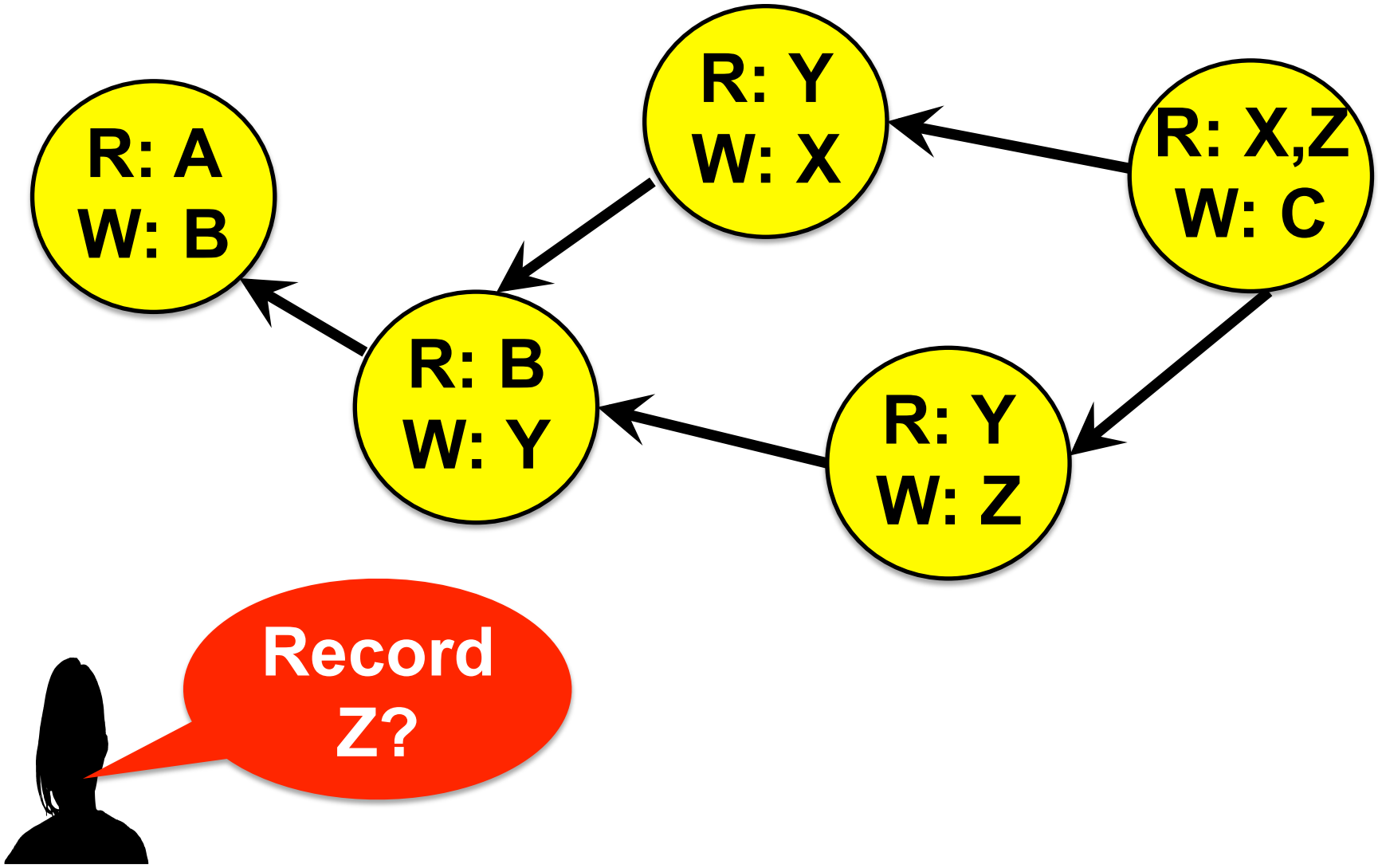
Dependency Graph of Transactions



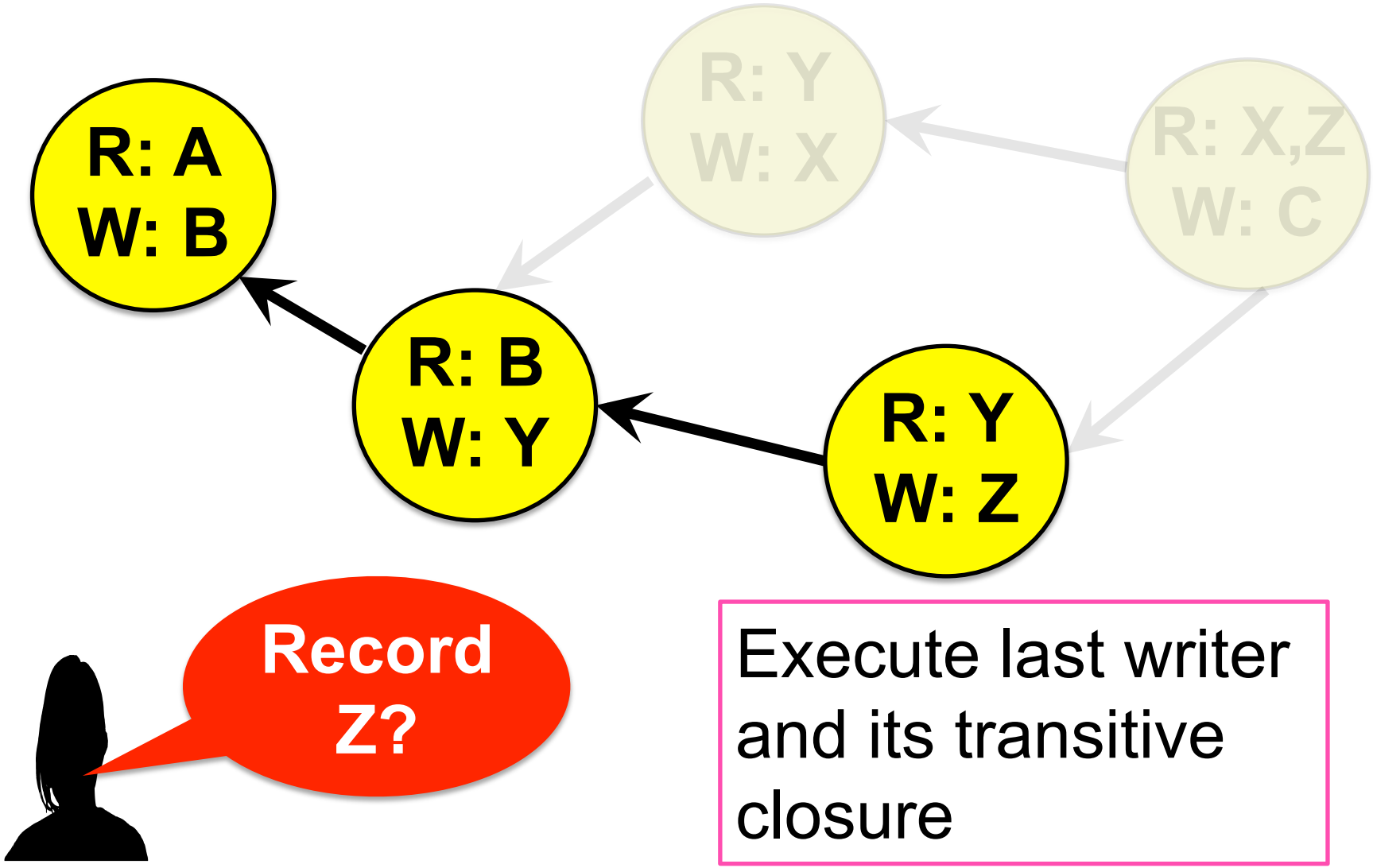
Dependency Graph of Transactions



Dependency Graph of Transactions



Dependency Graph of Transactions



Execute last writer and its transitive closure

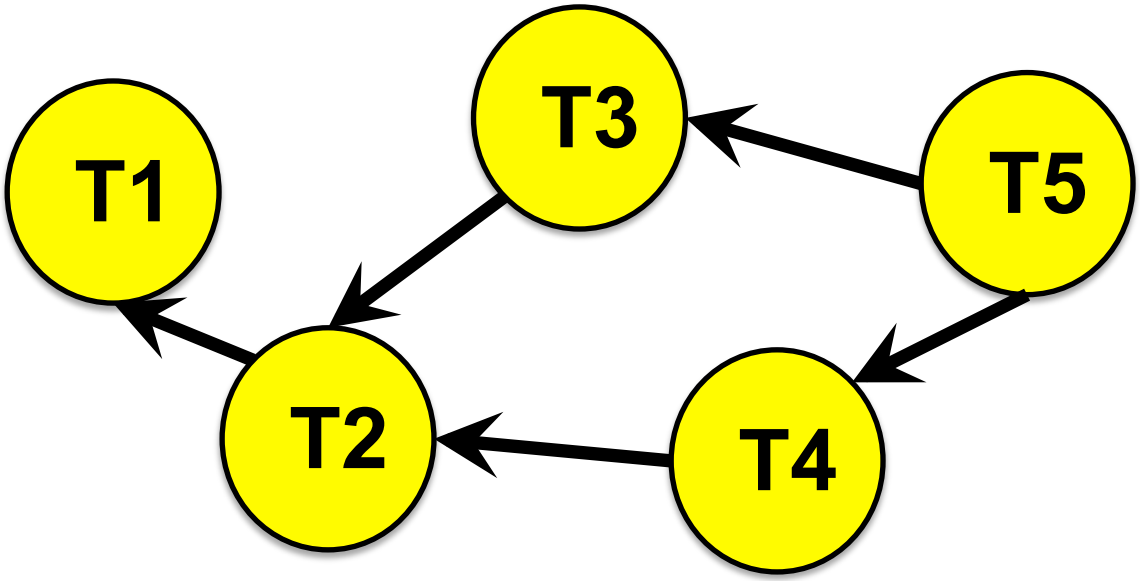
Large Transitive Closures

- The subset of the graph a record depends on may grow very large
  - High external read latency

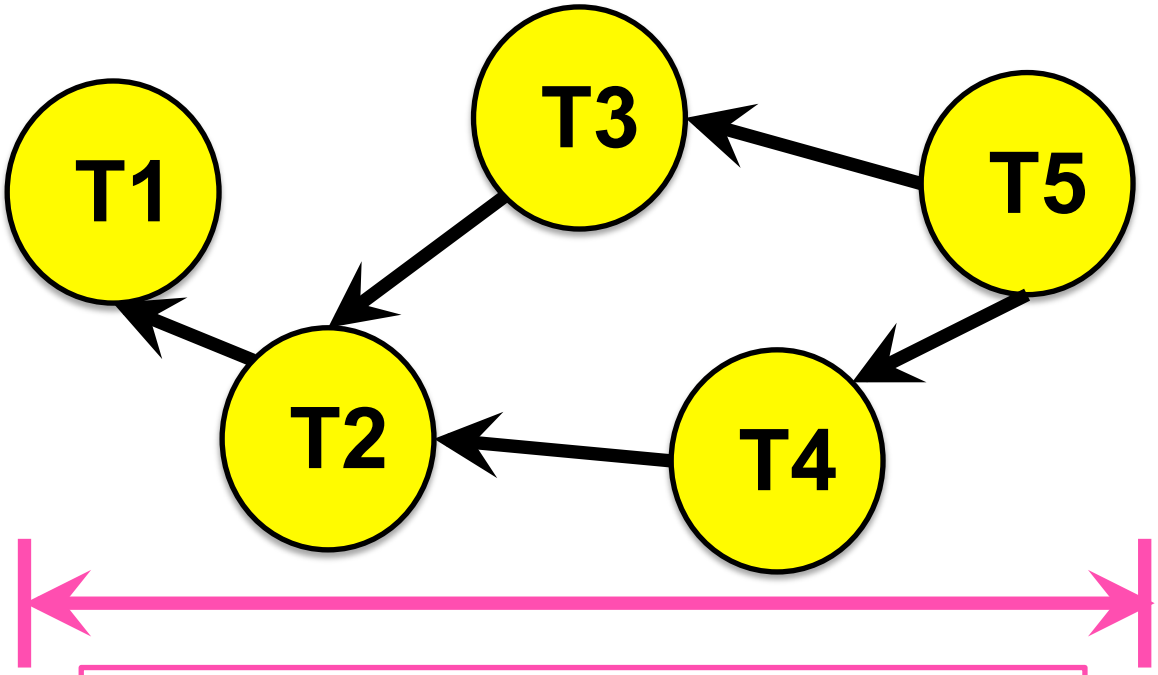
Large Transitive Closures

- The subset of the graph a record depends on may grow very large
  - High external read latency
  - **Incrementally process transactions in the background**

Incremental Transaction Processing

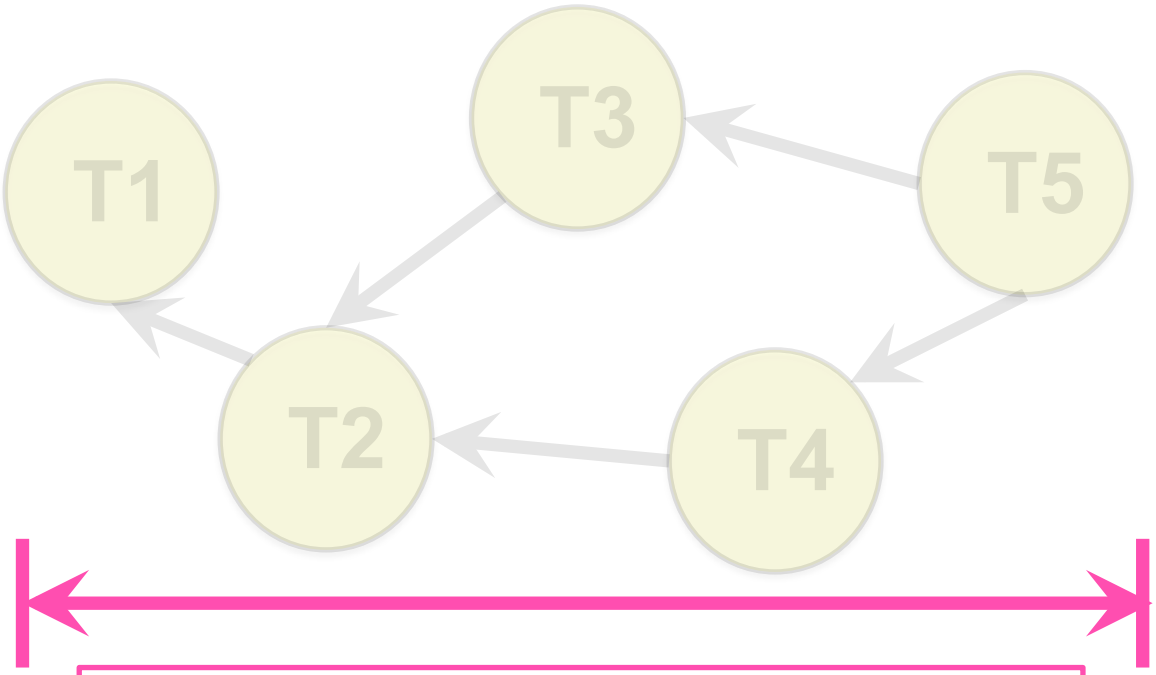


Incremental Transaction Processing



Bound the maximum depth of the dependency graph

Incremental Transaction Processing



Bound the maximum depth of the dependency graph

Incremental Transaction Processing

Tradeoff:

- Higher bound means larger batches (good for cache locality)

- Larger batches mean increased external read latency

Talk Outline

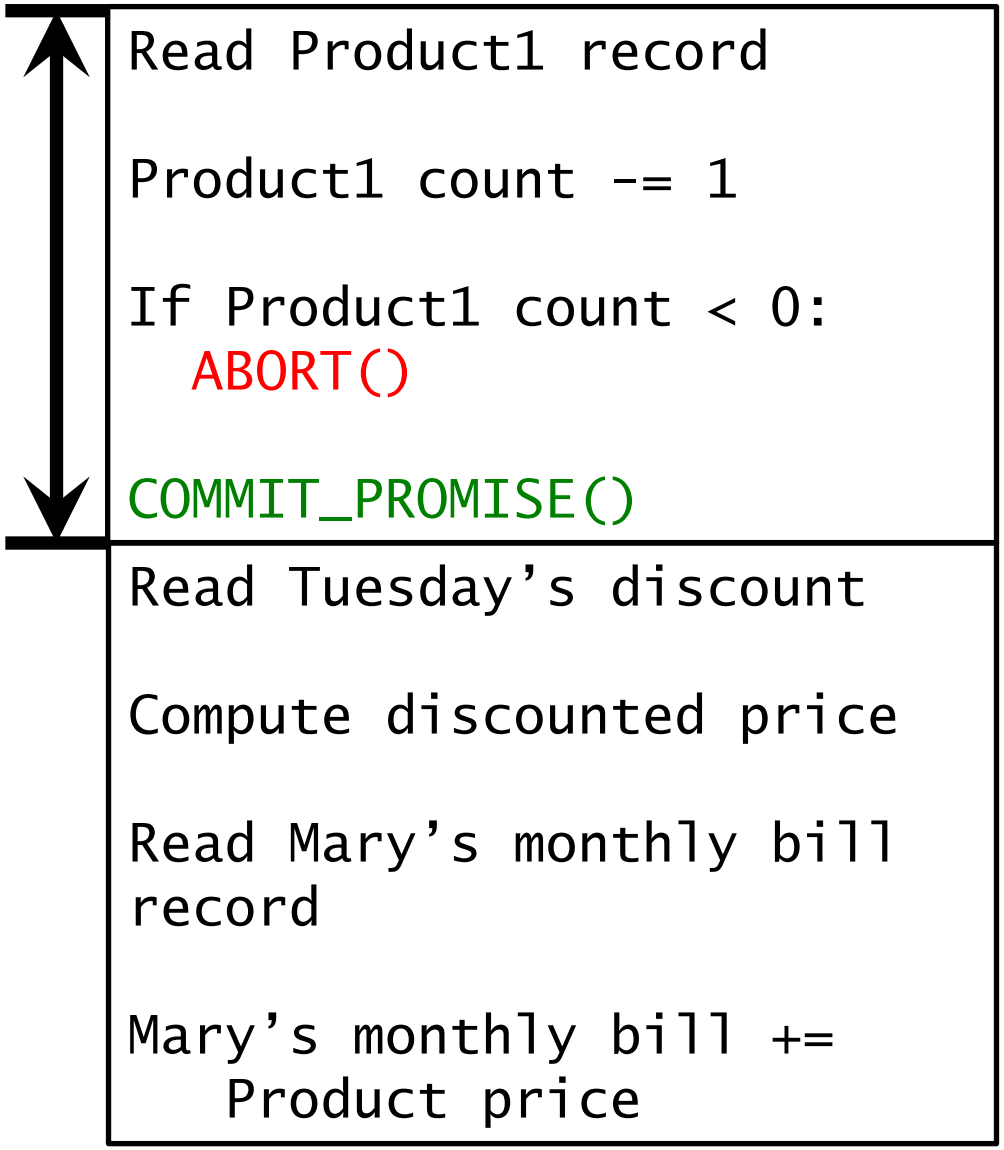
- Lazy Database Design
- **Benefits of Lazy Execution**
- Experimental Evaluation
- Conclusion

Benefits of Lazy Transactions

- Data Cache/Buffer Pool Locality
- Temporal Load Balancing
- Reduced Contention

Data Cache/Buffer Pool Locality

Execute immediately



Data Cache/Buffer Pool Locality

```
Read Tuesday's discount

Compute discounted price

Read Mary's monthly bill
record

Mary's monthly bill +=
   Product price
```

Deferred

Data Cache/Buffer Pool Locality

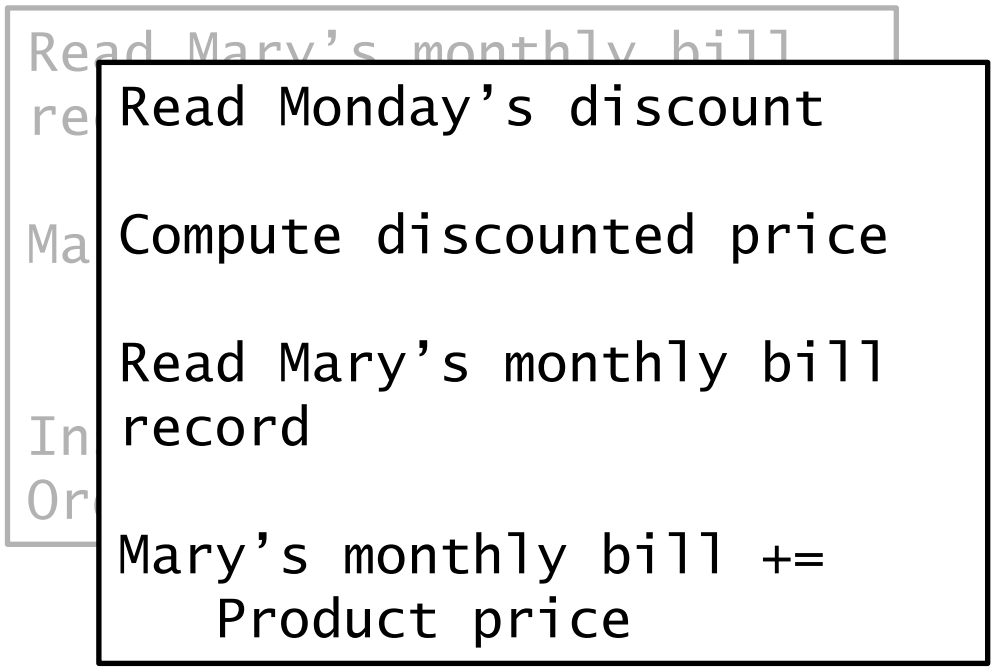
```
Read Mary's monthly bill
record

Mary's monthly bill +=
   Product1 price

Insert a record into
Orders table
```

```
Read Product2 record

Product2 count -= 1

If Product2 count < 0:
  ABORT()

COMMIT_PROMISE()
```

```
Read Monday's discount

Compute discounted price

Read Mary's monthly bill
record

Mary's monthly bill +=
   Product price
```

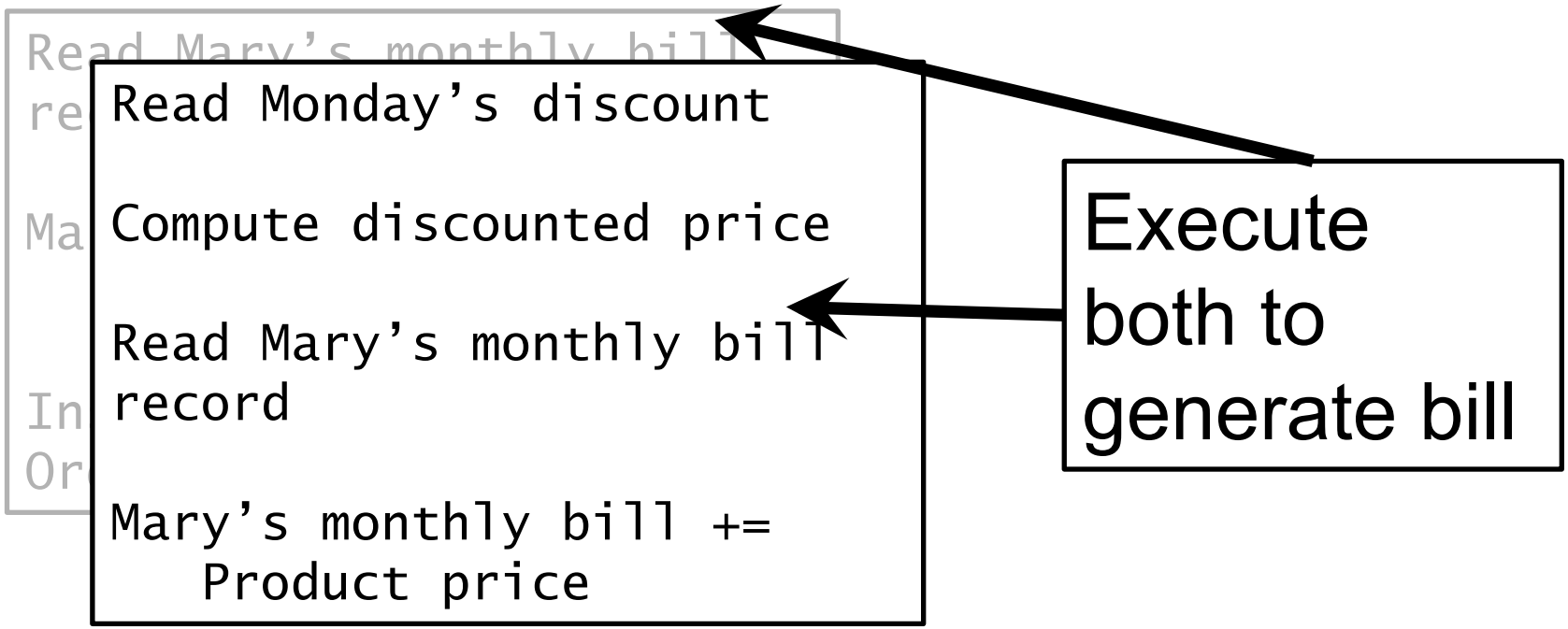
Deferred

Data Cache/Buffer Pool Locality



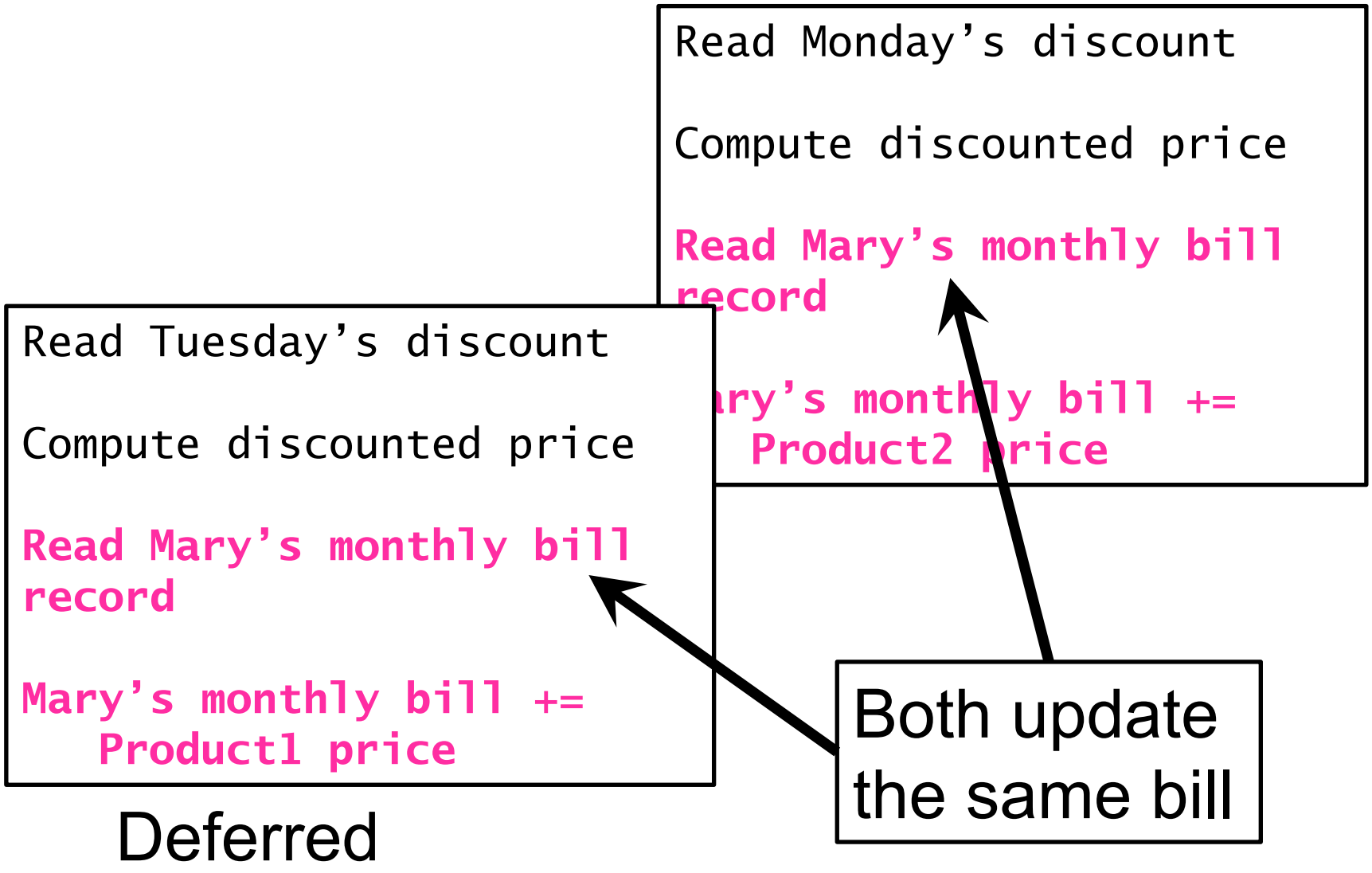
Deferred

Data Cache/Buffer Pool Locality

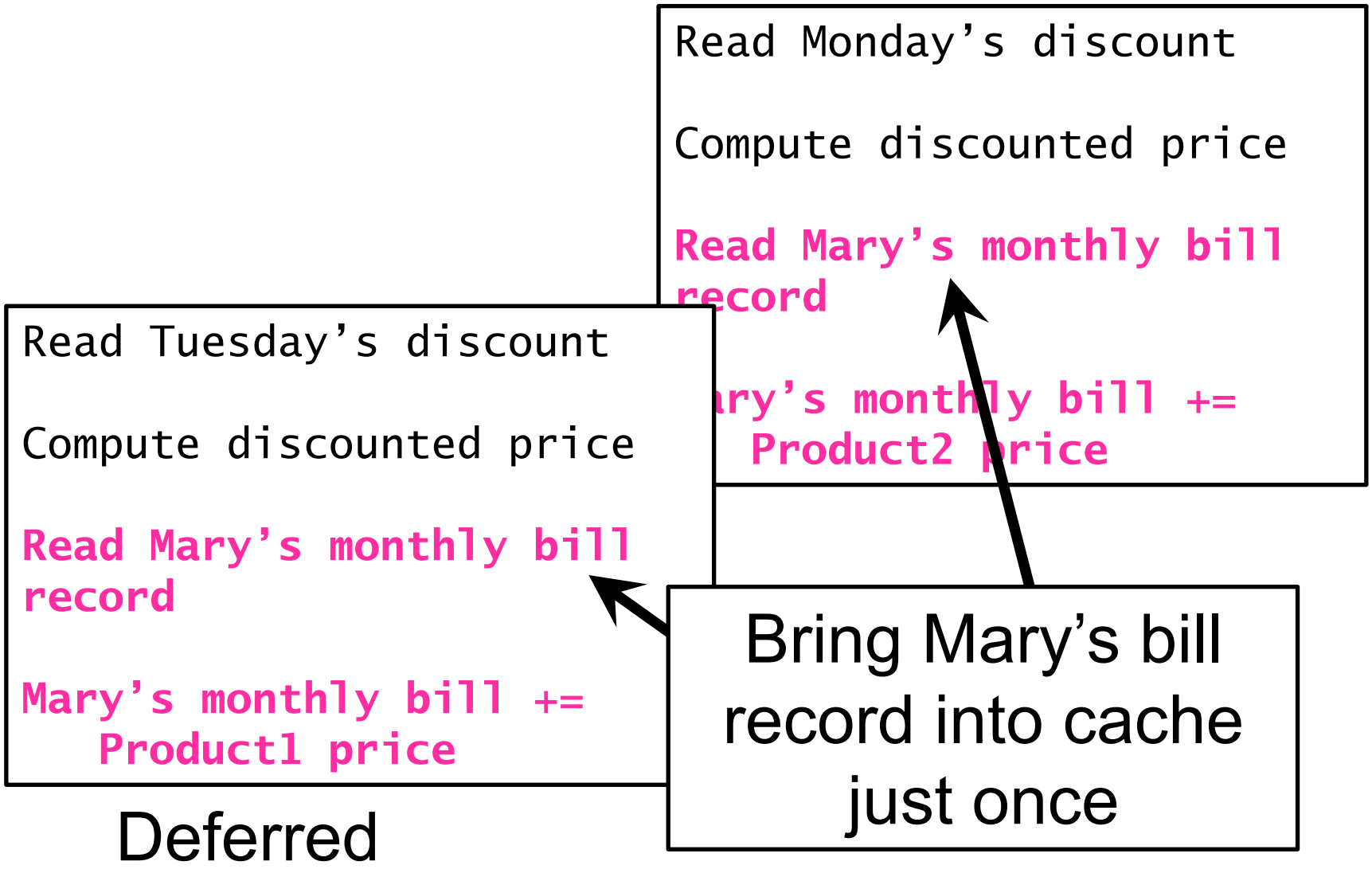


Deferred

Data Cache/Buffer Pool Locality



Data Cache/Buffer Pool Locality



Temporal Load Balancing

```
Read Product record

Product count -= 1

If Product count < 0:
  ABORT()

Read daily discount

Compute discounted price

Read Customer's monthly
bill record

Customer's monthly bill +=
   Product price

COMMIT()
```

```
Read Product record

Product count -= 1

If Product count < 0:
  ABORT()

COMMIT_PROMISE()
```

```
Read daily discount

Compute discounted price

Read Customer's monthly
bill record

Customer's monthly bill +=
   Product price
```

Temporal Load Balancing

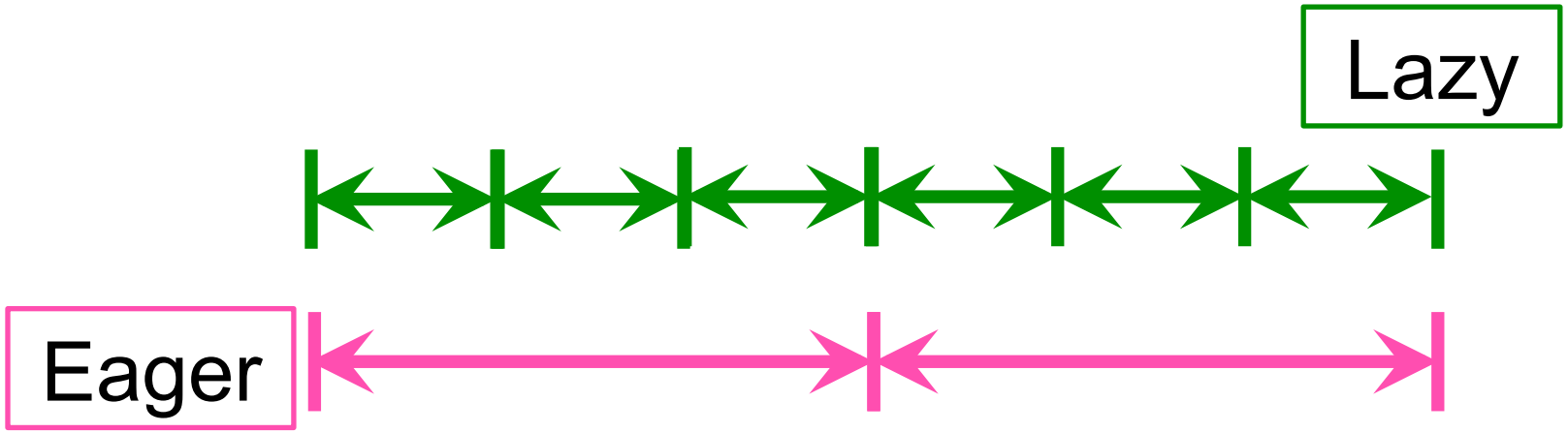
```
Read Product record

Product count -= 1

If Product count < 0:
  ABORT()

Read daily discount

Compute discounted price

Read Customer's monthly
bill record

Customer's monthly bill +=
   Product price

COMMIT()
```

```
Read Product record

Product count -= 1

If Product count < 0:
  ABORT()

COMMIT_PROMISE()
```

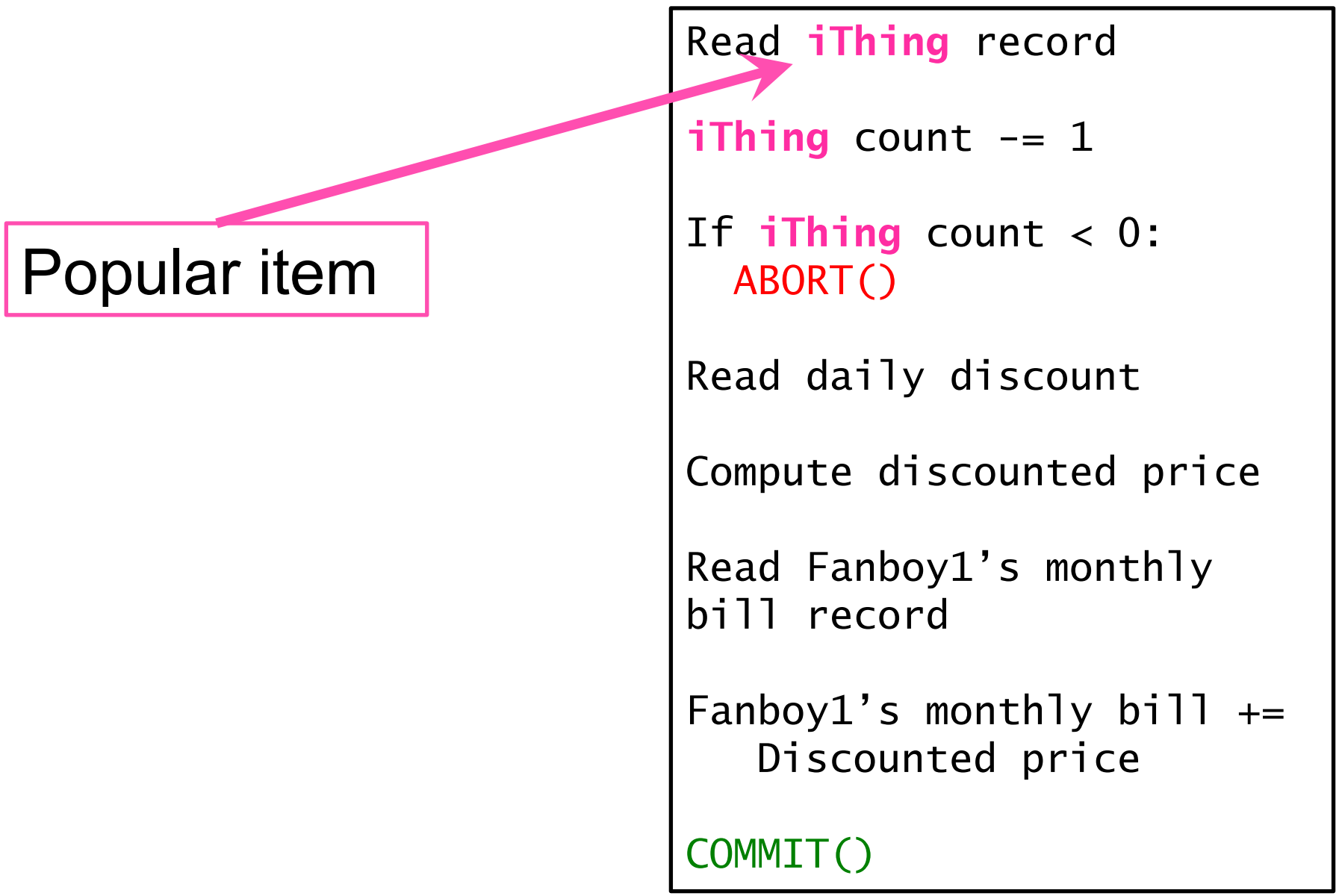
Arrows represent time required to immediately process transactions

Temporal Load Balancing

More commit decisions per unit time

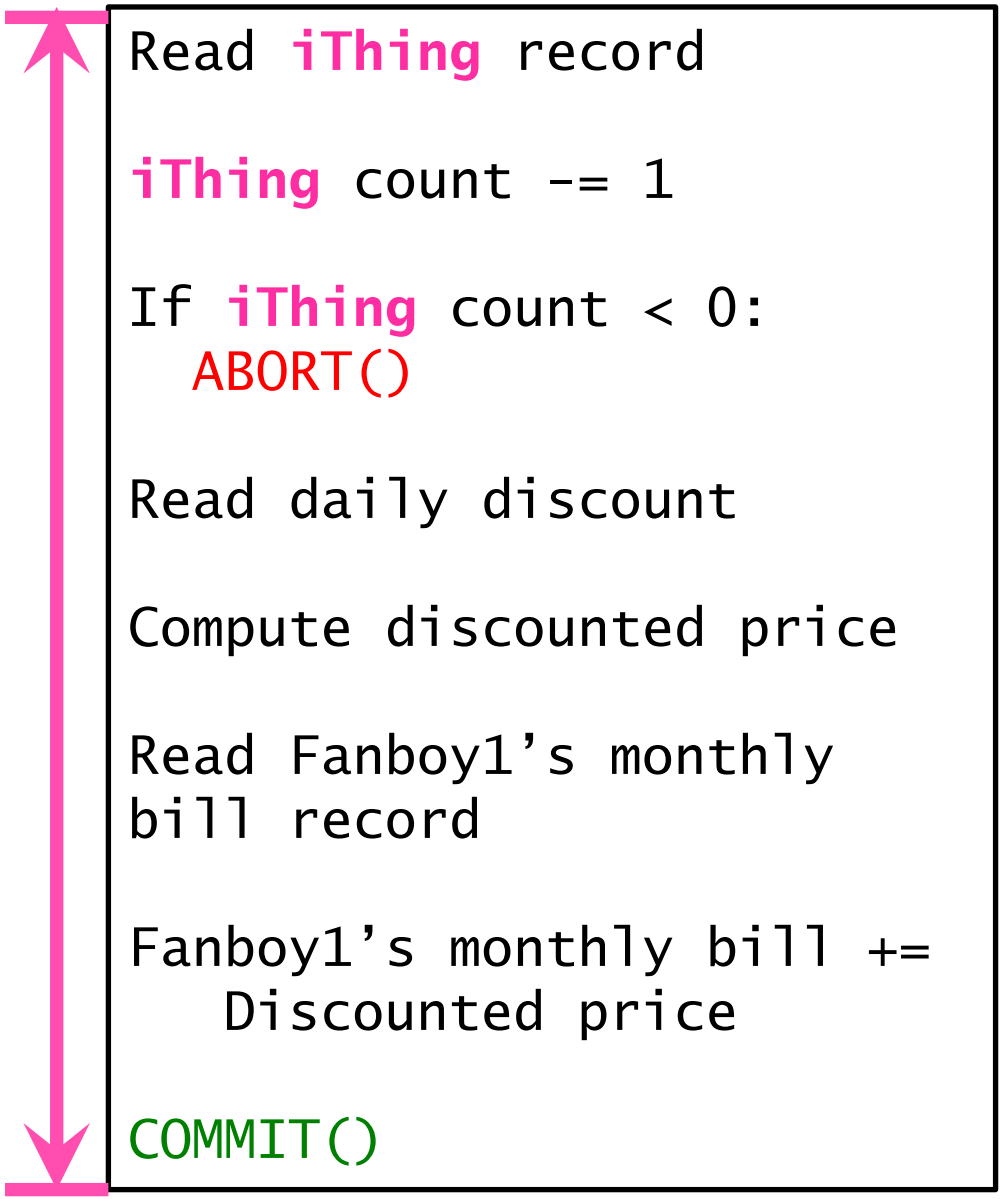


Reduced Contention



Reduced Contention

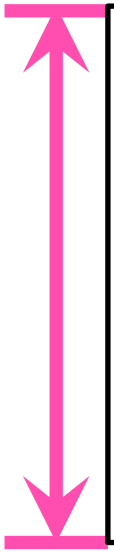
Lock must be held for this period



```
Read iThing record

iThing count -= 1

If iThing count < 0:
  ABORT()

Read daily discount

Compute discounted price

Read Fanboy1's monthly
bill record

Fanboy1's monthly bill +=
   Discounted price

COMMIT()
```

Reduced Contention

Hold contended lock for less time



```
Read iThing record

iThing count -= 1

If iThing count < 0:
  ABORT()

COMMIT_PROMISE()
```

```
Read daily discount

Compute discounted price

Read Fanboy1's monthly
bill record

Fanboy1's monthly bill +=
   Discounted price
```

Talk Outline

- Lazy Database Design
- Benefits of Lazy Execution
- Experimental Evaluation
- Conclusion

Experimental Setup

- All data in memory
- Single machine
- 8 cores
  - Lazy: 1 commit/abort + 7 deferred logic
  - Eager: 8 cores

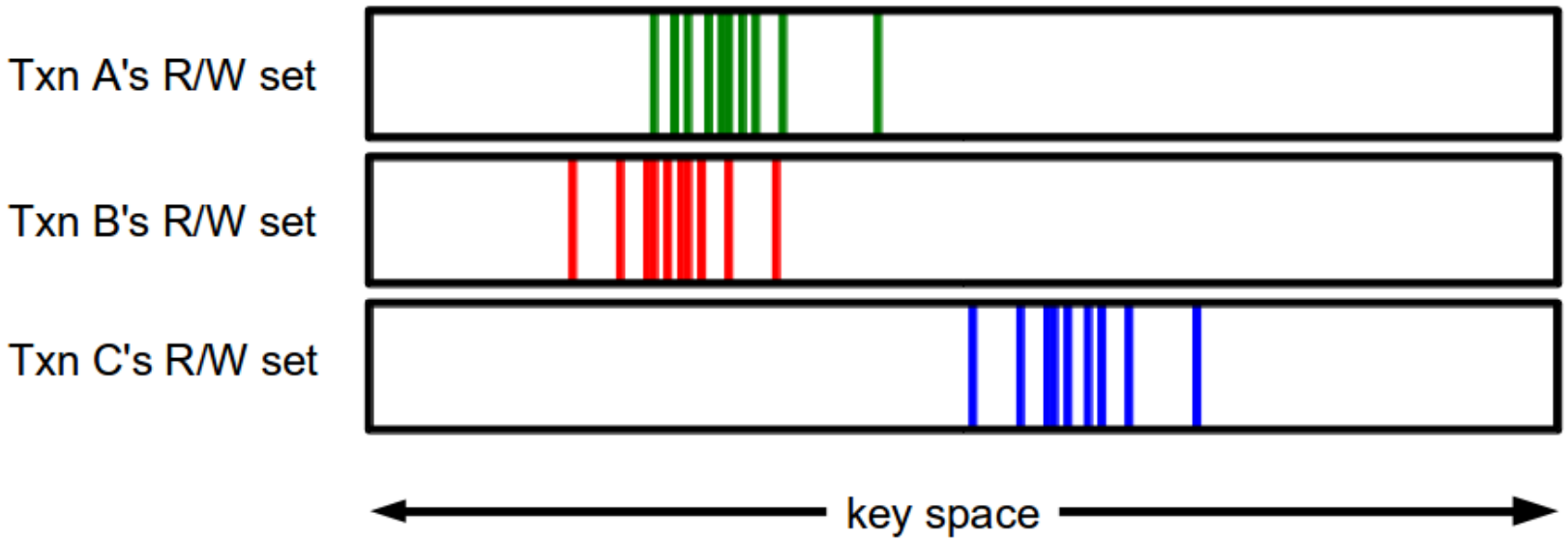
Experimental Evaluation

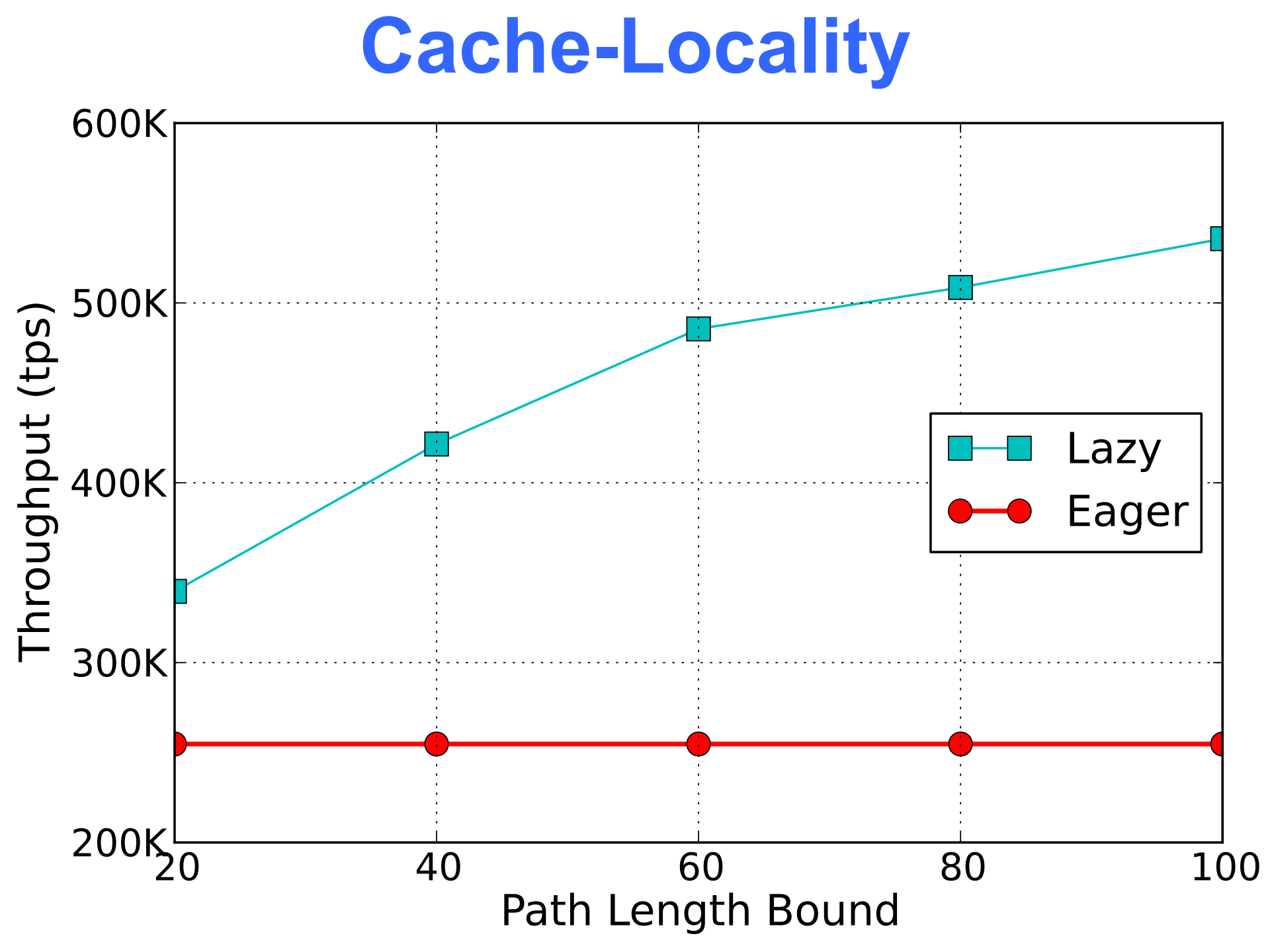
- Cache Locality
- External Read Latency
- Temporal Load Balancing
- Contention Reduction

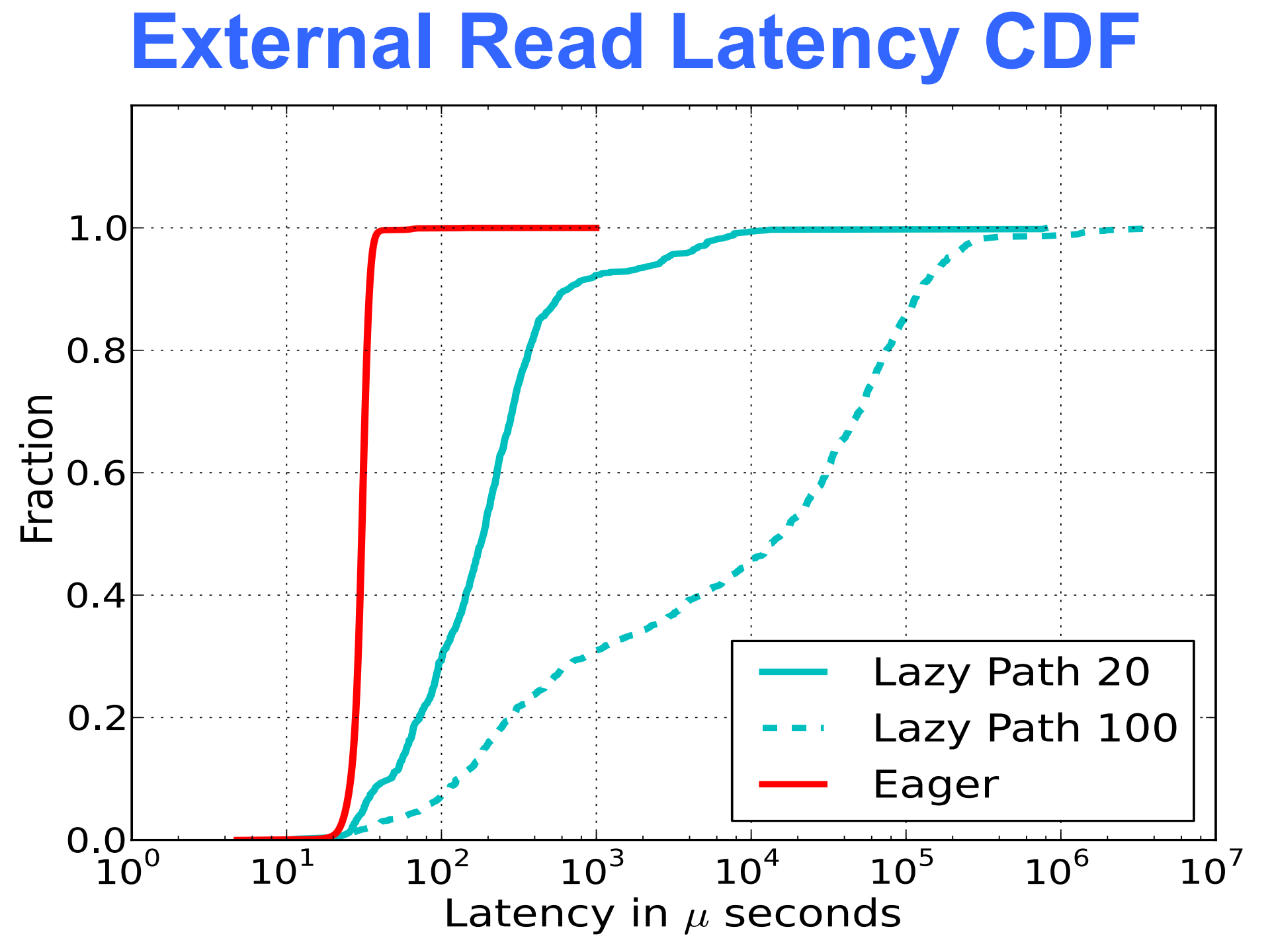
Cache-Locality

- “Cliquey” workload

- Conflicting transactions conflict on several records



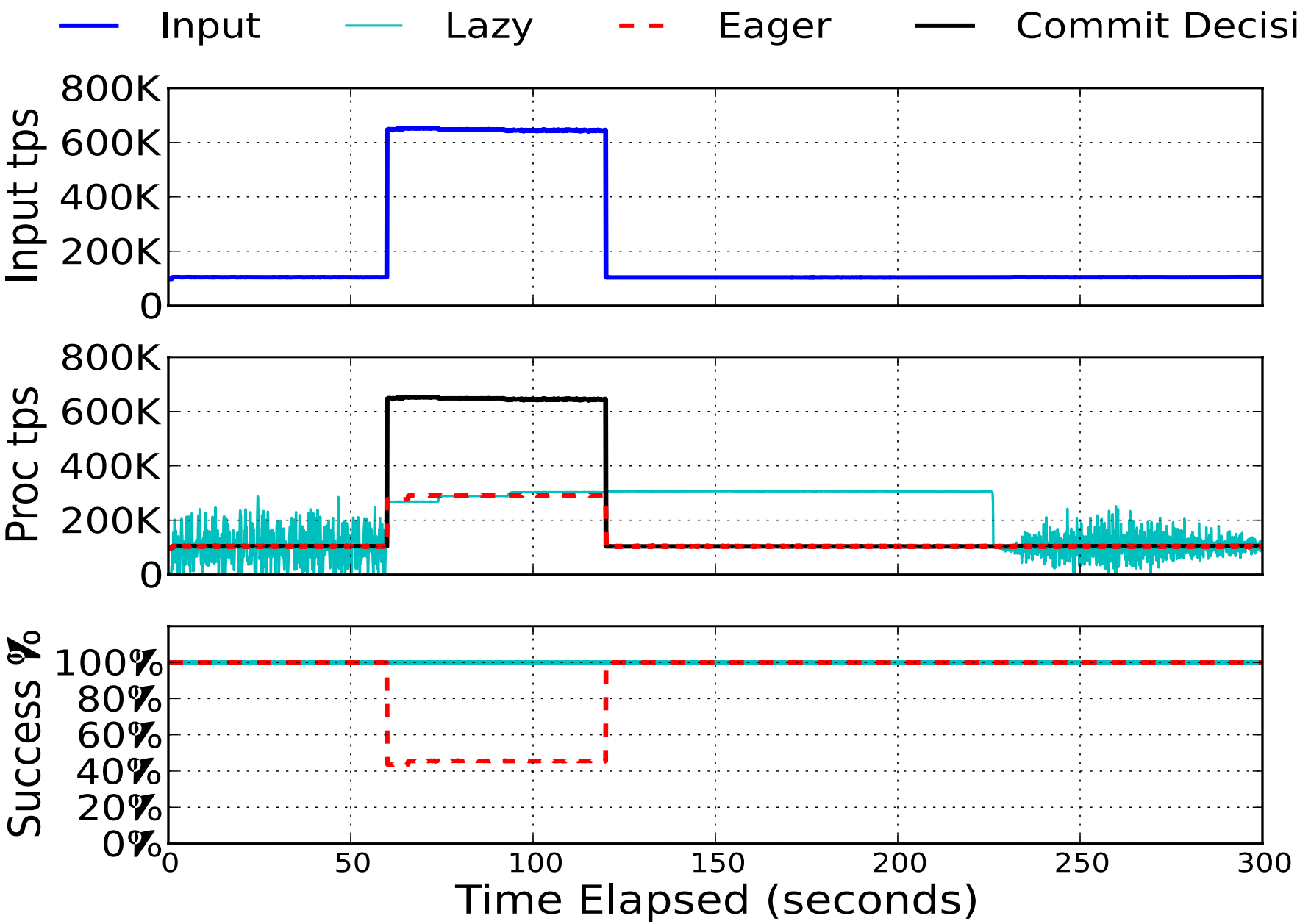

Cache-Locality
600K
500K
400K
300K
200K
Throughput (tps)
20
40
60
80
100
Path Length Bound
Lazy
Eager


External Read Latency CDF
Fraction
1.0
0.8
0.6
0.4
0.2
0.0
10^0
10^1
10^2
10^3
10^4
10^5
10^6
10^7
Latency in μ seconds
Lazy Path 20
Lazy Path 100
Eager

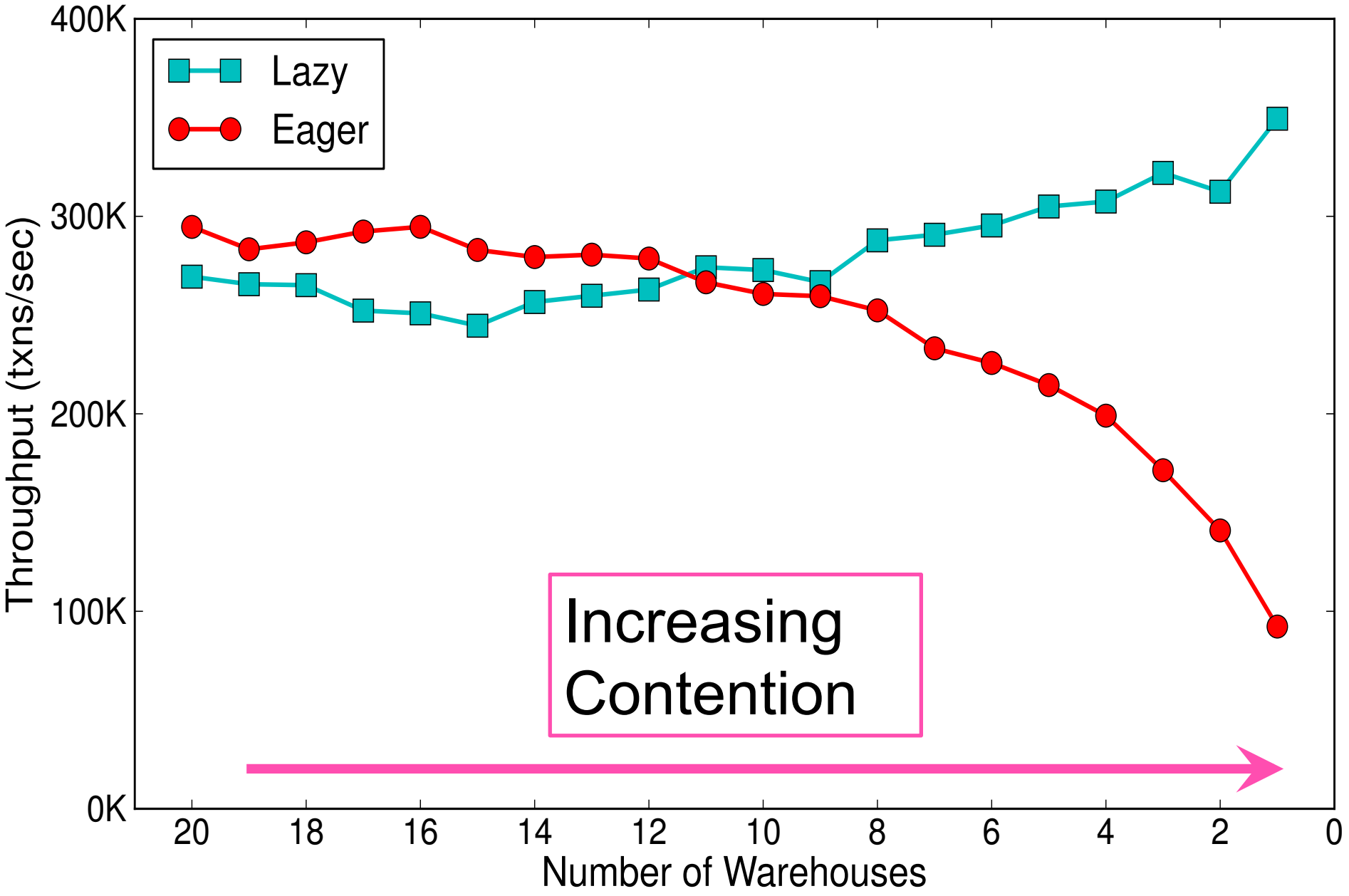
Temporal Load Balancing

- Lazy and eager systems have comparable throughput

Temporal Load Balancing



TPC-C Throughput

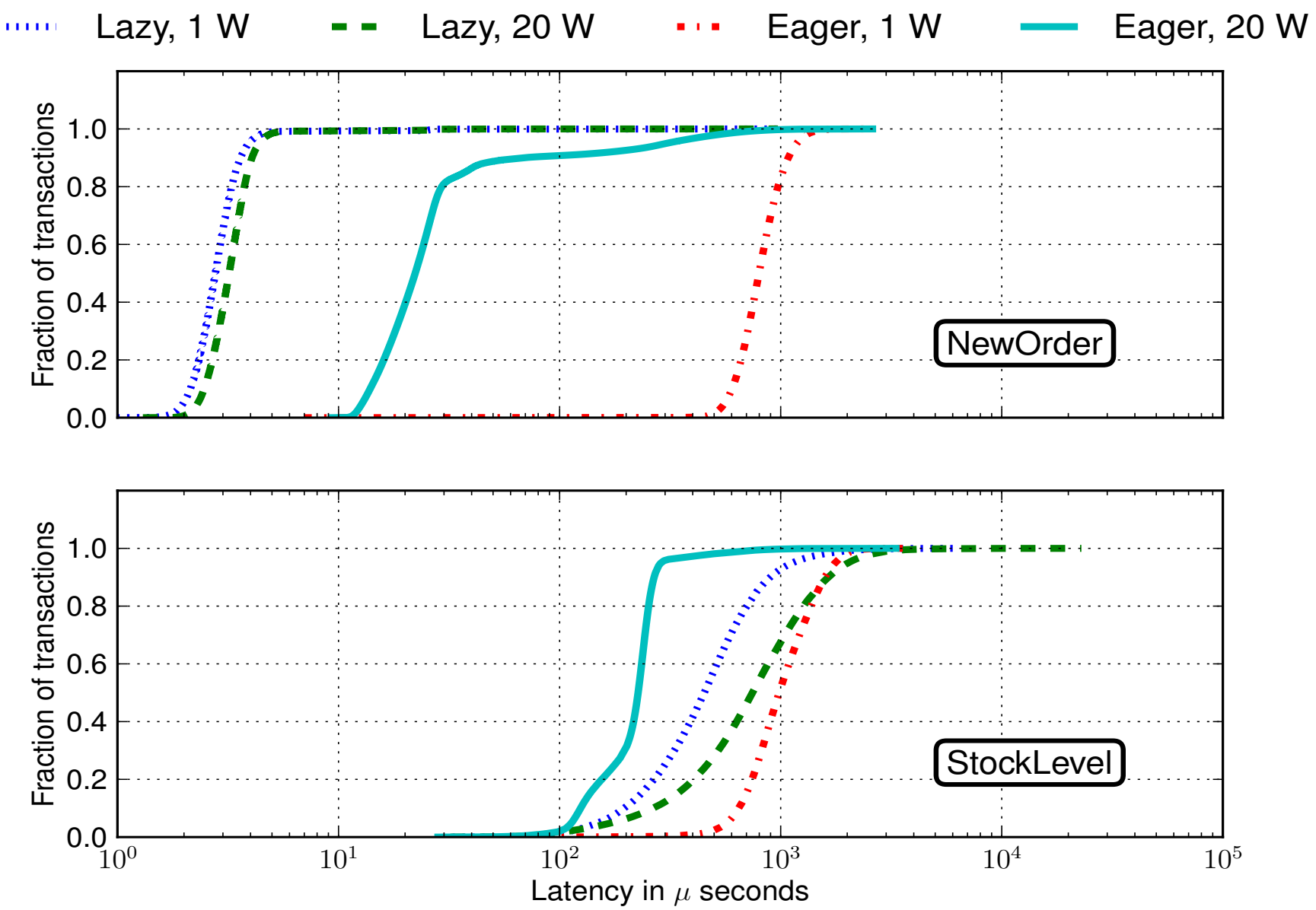


Conclusions

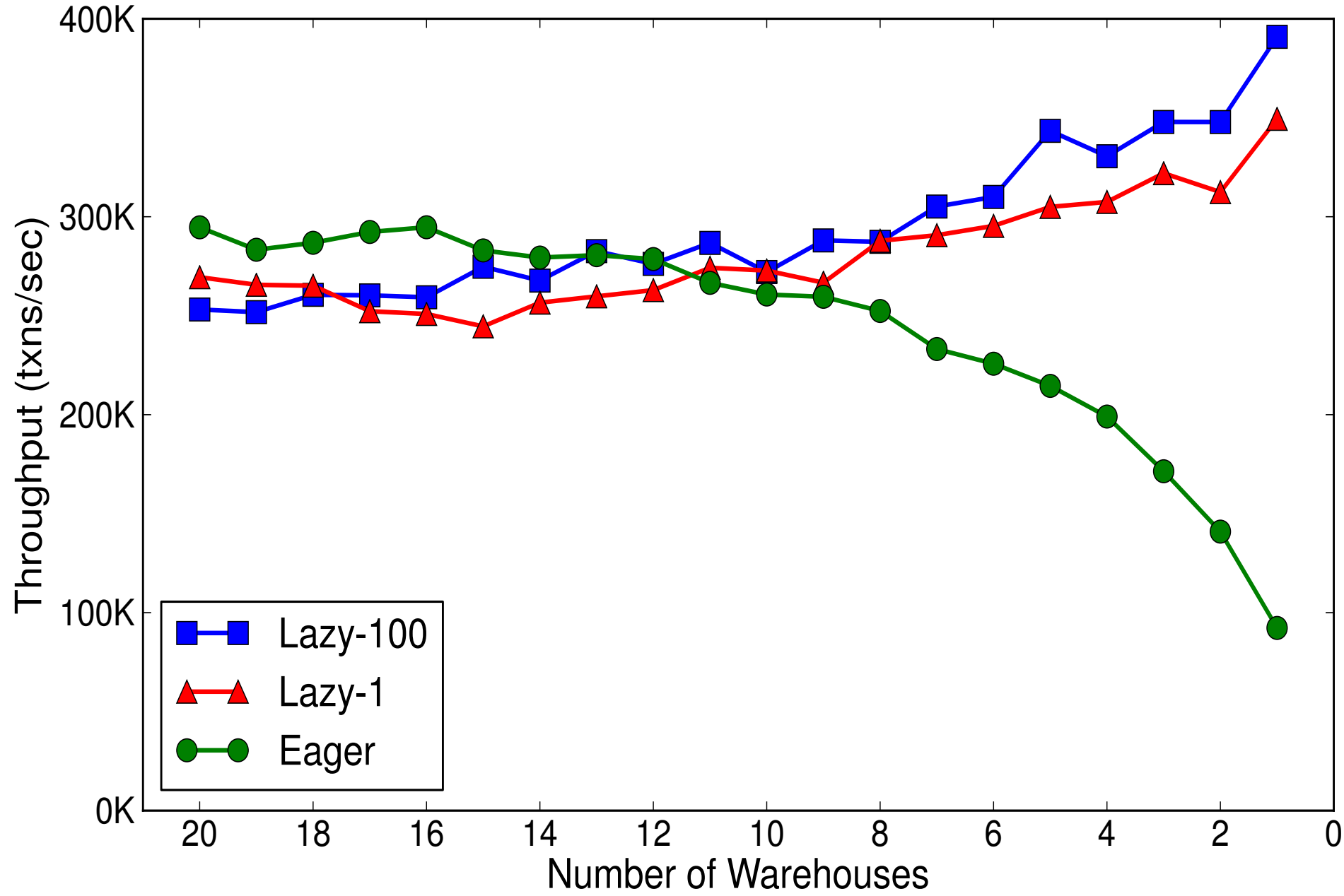
- Lazily evaluating txns has its benefits
  - Cache Locality
  - Temporal Load Balancing
  - Contention Reduction

To see the rest of our conclusions, read the paper

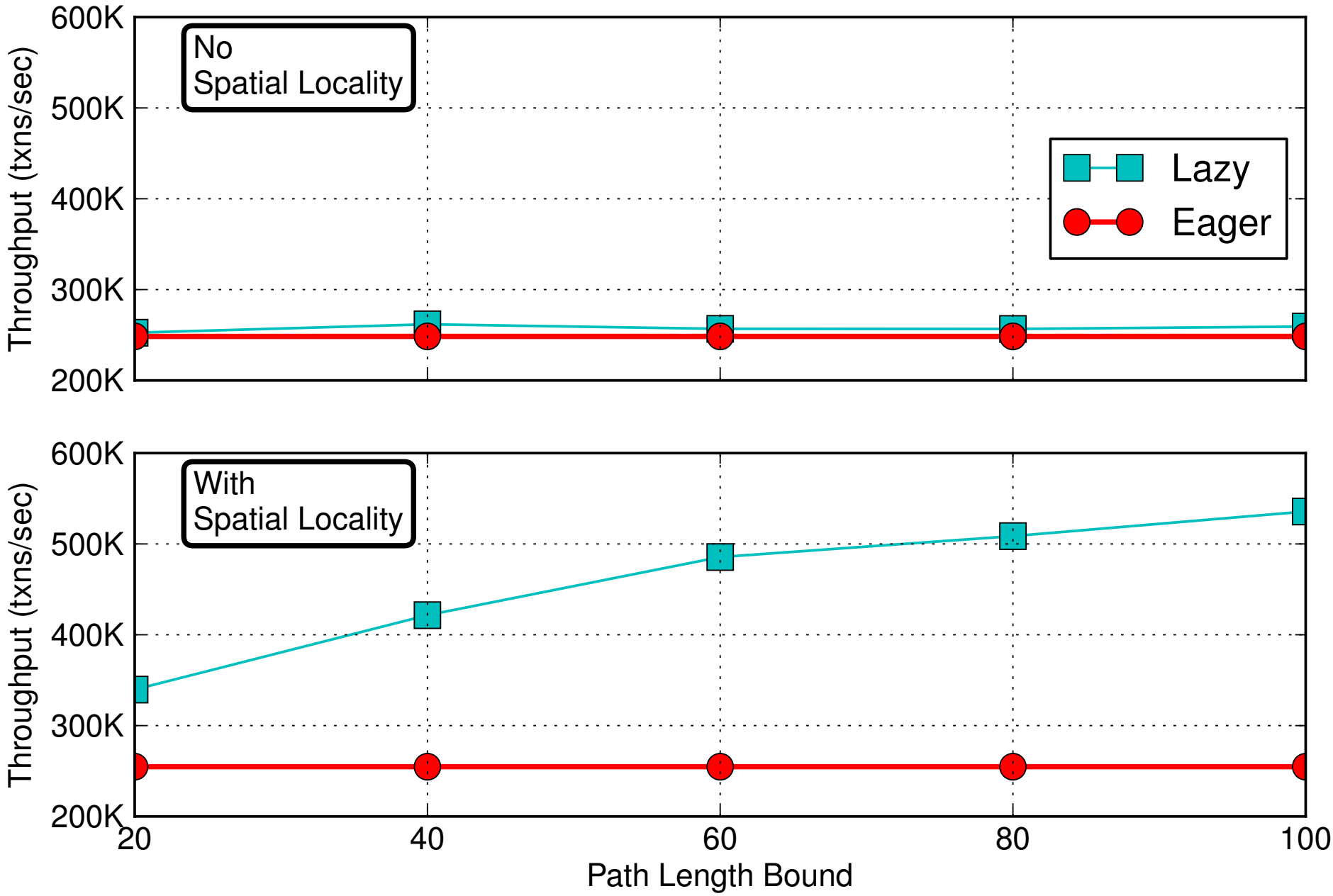
TPC-C Latency



TPC-C Throughput



Microbenchmark Throughput



Microbenchmark Latency