CHICAGO
INTERNATIONAL
SOFTWARE DEVELOPMENT
CONFERENCE 2015

goto; conference

Don’t Give Up on Serializability Just Yet

Neha Narula

Don’t Give Up on Serializability Just Yet

A journey into serializable systems

Neha Narula
MIT CSAIL

GOTO Chicago May 2015

@neha

- PhD candidate at MIT
- Formerly at Google
- Research in fast transactions for multi-core databases and distributed systems

However, the most important person in my gang will be a systems programmer. A person who can debug a device driver or a distributed system is a person who can be trusted in a Hobbesian nightmare of breathtaking scope; a systems programmer has seen the terrors of the world and understood the intrinsic horror of existence.

There are some obvious people who I'll need to recruit: a locksmith (to open

A journey into serializable systems

-validateAccount
+viewAccount()
+updateAccount()
+isValidUpdate()
-serverType
-processingSpeed
-concurentSessions
+isDown()
+isTimedOut()
+grantSession()
+processRequest()
+sendResponse()
System Administrator
-login
-databaseKey
+cancelQuery()
+accessDatabase()()
Transaction interface
-transactionType
-transactionStatus
+getTransactionType()
+getExecutionStatus()
+isValidTransaction()
+getMarketInfo()
Secure Interface
-secureId
-securetyProtocol
-browserSupported
+isValidSecureId()
+isBrowsersupported()
+secureTransaction()
Account
-accountId
-accountOwner
-accountInfo
+getAccountInfo()
+updateAccountInfo()
+notifyAccountOwner()
+manageAccount()
-loginname
-email
-phonenumbe
-ssn
-password
Interface Components
Portfolio Managment
-stocks
-industryType

1M messages/sec

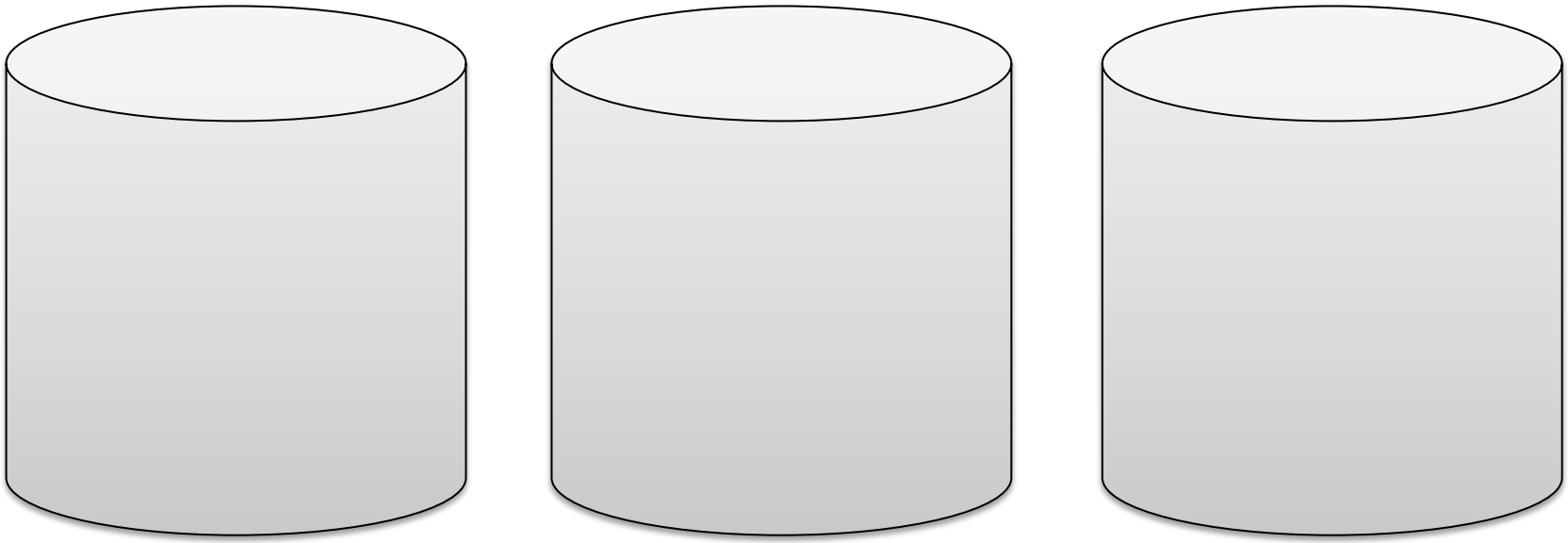
1/5 of all page views in the US

1M messages/sec from mobile devices

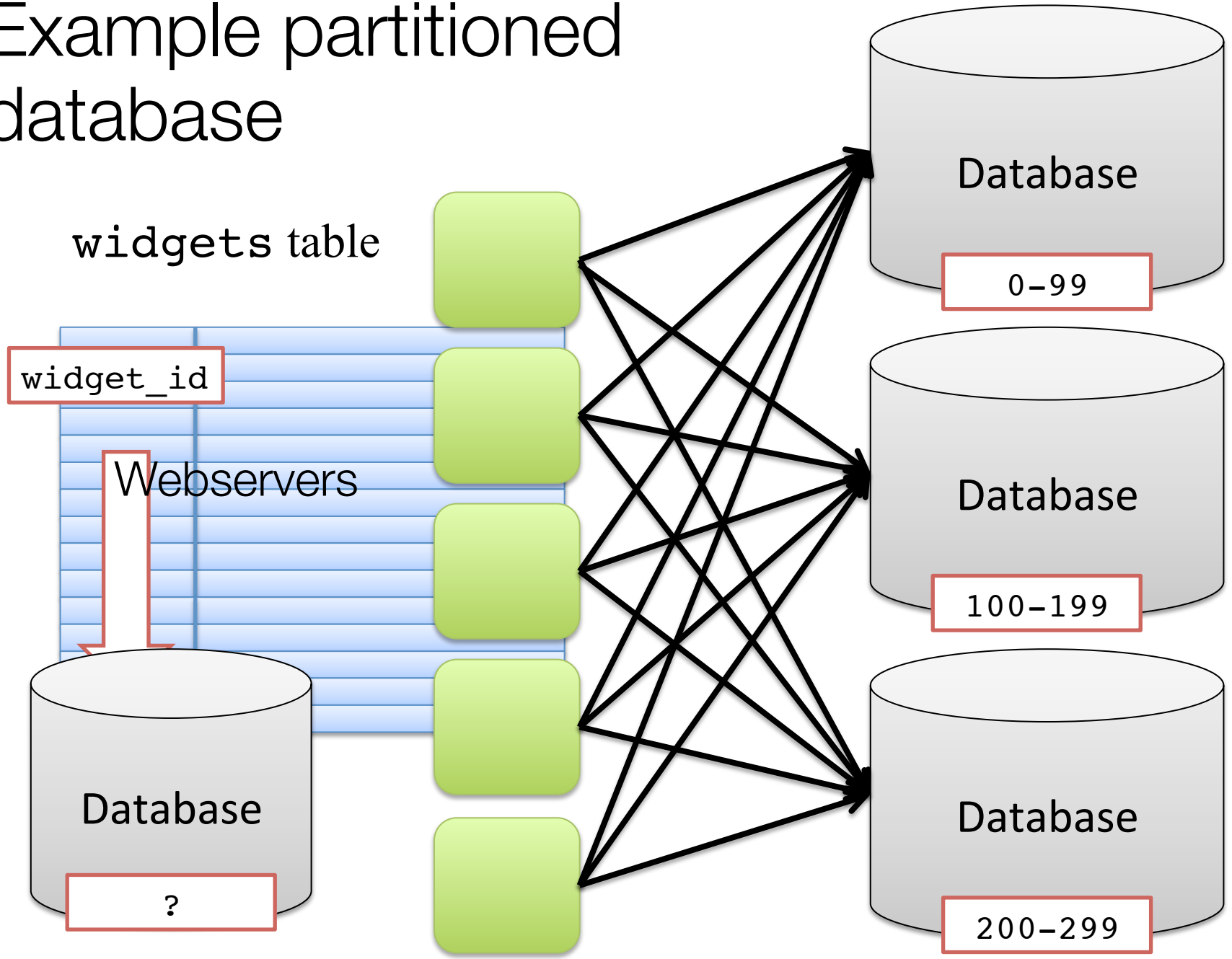
Databa… scale

Distributed databases

Partition data on multiple servers for more performance



Example partitioned database



Google

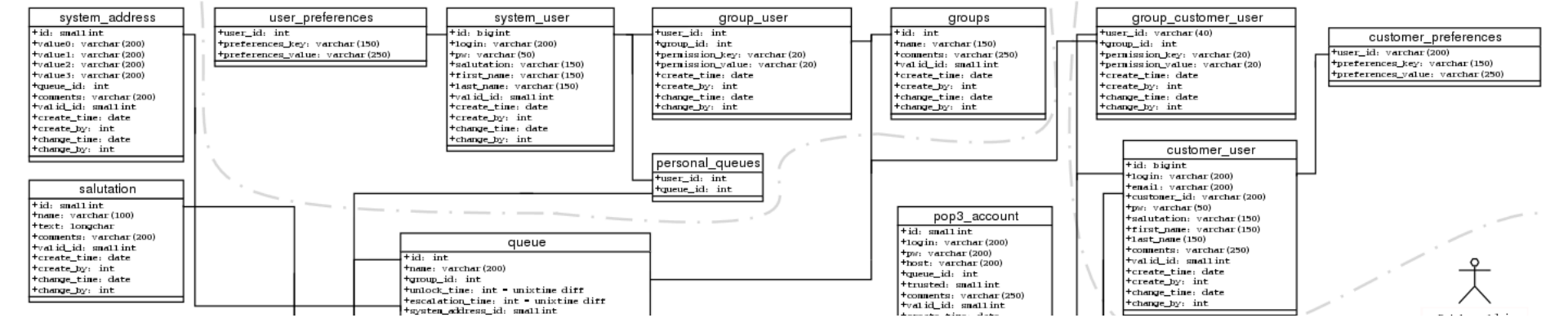
2007

- Mapreduce
- Google File System
- Bigtable

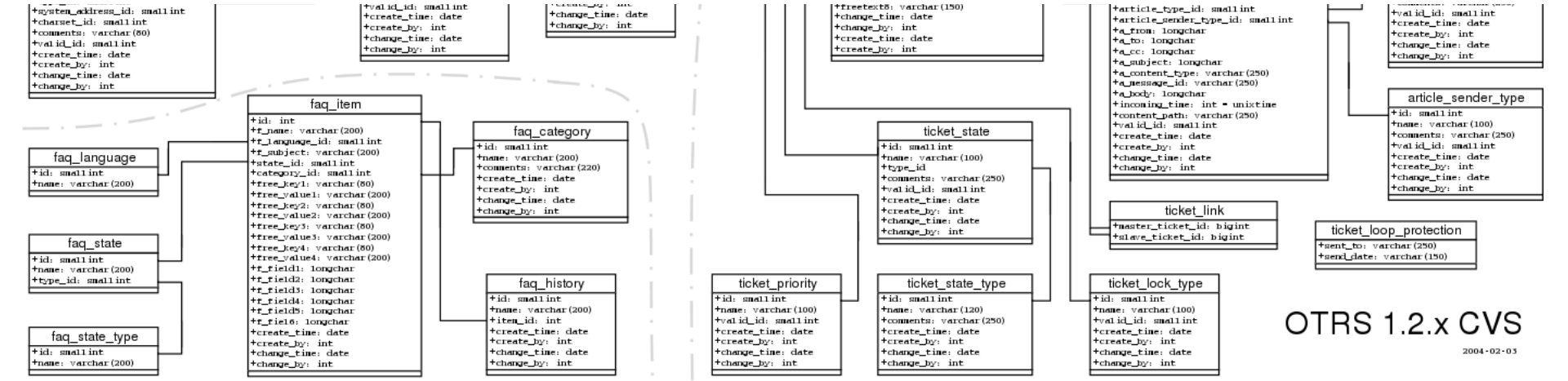
Pros/Cons

- In-memory
- HIGHLY scalable
- Transparently fault tolerant
- Geo replication

- No schema
- Require complex key/row/document design
- No query language
- No indexes
- No transactions
- No guarantees



¯_(ツ)_/¯



OTRS 1.2.x CVS

2004-02-03

Fault-tolerance

by @jrecursive

```
mysql> BEGIN TRANSACTION
UPDATE …
COMMIT
```

Problem with dropping transactions

- Difficult to reason about concurrent interleavings
- Might result in incorrect, unrecoverable state

“The hacker discovered that multiple simultaneous withdrawals are processed essentially at the same time and that the system's software doesn't check quickly enough for a negative balance”

http://arstechnica.com/security/2014/03/yet-another-exchange-hacked-poloniex-loses-around-50000-in-bitcoin/

Consistency guarantees help us reason about our code and avoid subtle bugs

Consistency

A very misused word in systems!

- C as in ACID
- C as in CAP
- C as in sequential, causal, eventual, strict consistency

ACID Transactions

Atomic — Whole thing happens or not

Consistent — Application-defined correctness

Isolated — Other transactions do not interfere

Durable — Can recover correctly from a crash

```
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE
BEGIN TRANSACTION
...
COMMIT
```

What is Serializability?

Serializability != Serial

What is Serializability?

The result of executing a set of transactions is the same as if those transactions had executed one at a time, in some serial order.

If each transaction preserves correctness, the DB will be in a correct state.

We can pretend like there's no concurrency!

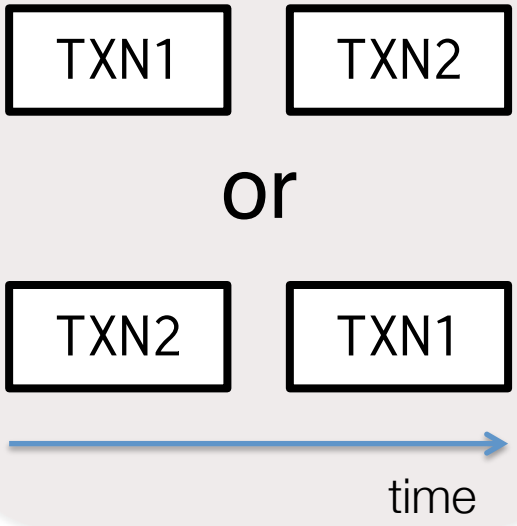
Database transactions should be serializable

```
TXN1(k, j Key)→(Value, Value) {
    a := GET(k)
    b := GET(j)
    return a, b
}
```

```
TXN2(k, j Key) {
    ADD(k,1)
    ADD(j,1)
}
```

k=0,j=0

To the programmer:



Valid return values for TX1: (0,0) or (1,1)

Benefits of Serializability

- Do not have to reason about interleavings
- Do not have to express invariants separately from the code!

Serializability Costs

- On a multi-core database, serialization and cache line transfers
- On a distributed database, serialization and network calls

Concurrency control: Locking and coordination

Eventual consistency

If no new updates are made to the object, eventually all accesses will return the last updated value.

Eventual consistency

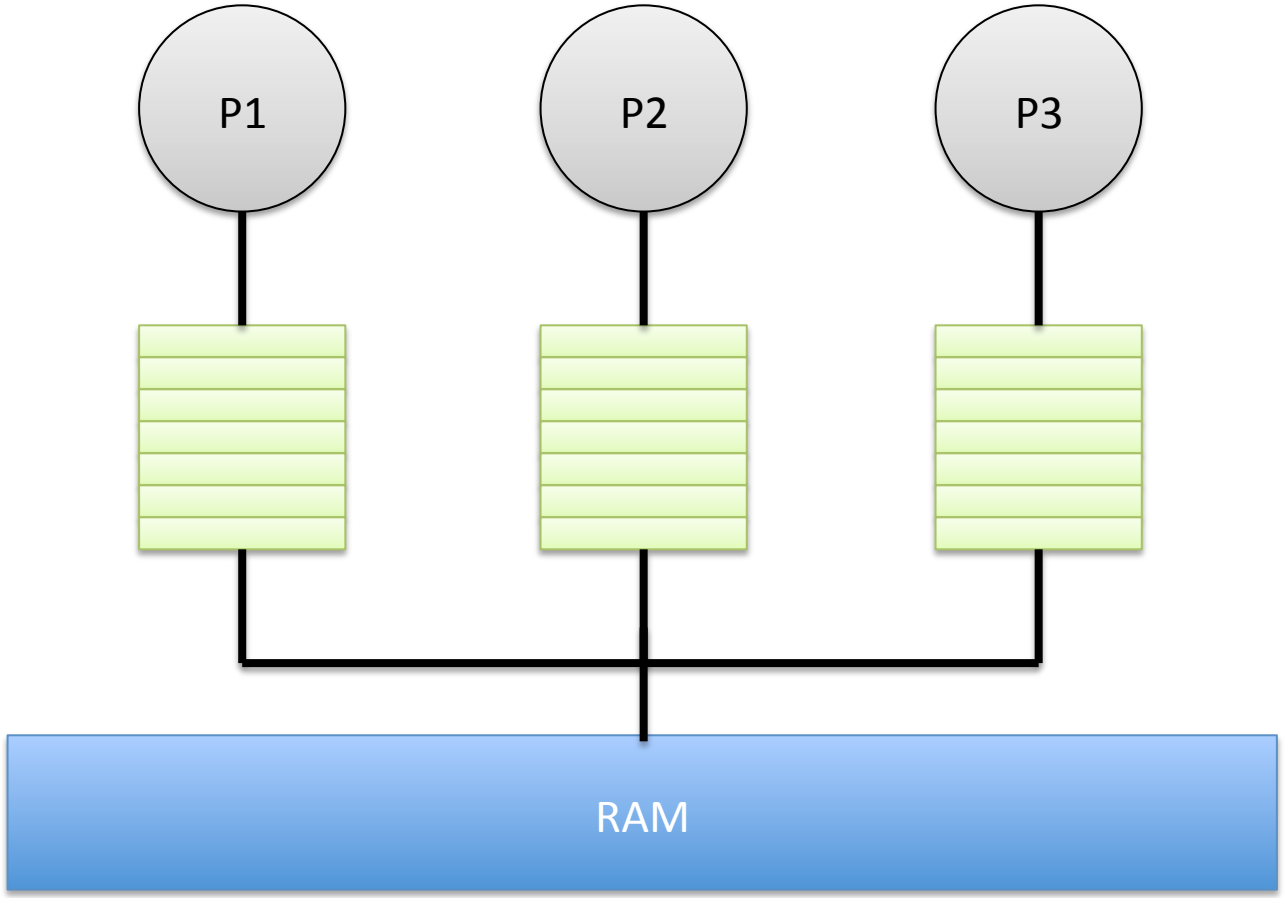
If no new updates are made to the object, eventually all accesses will return ~~the last updated value~~ the same value.

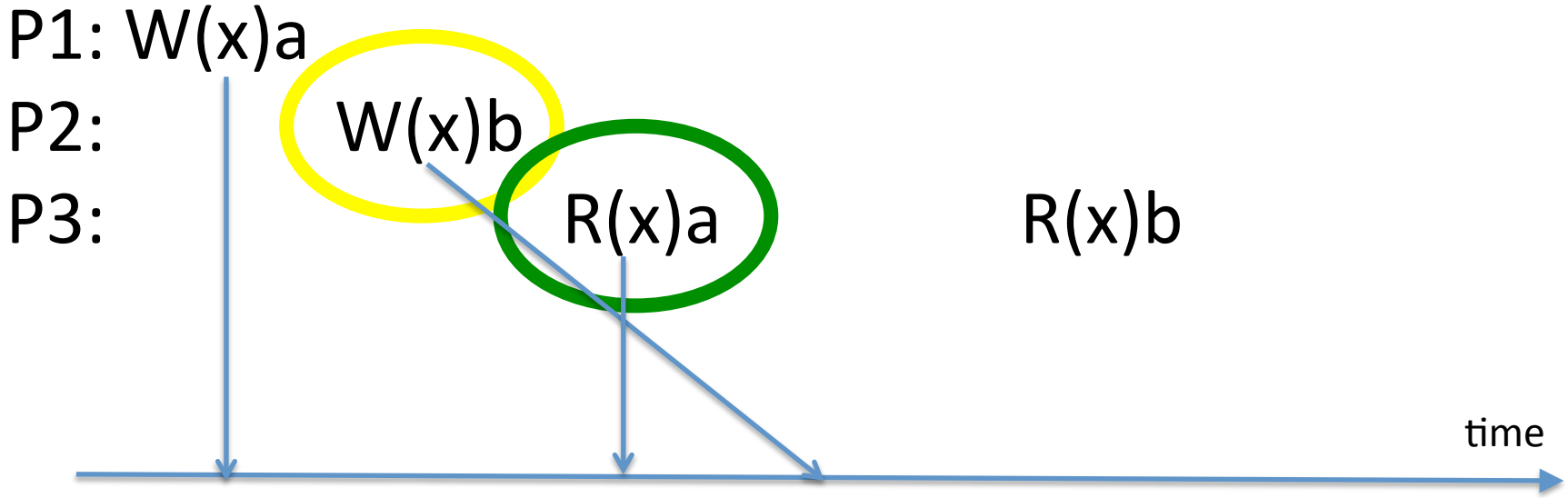
(What is last, really?)

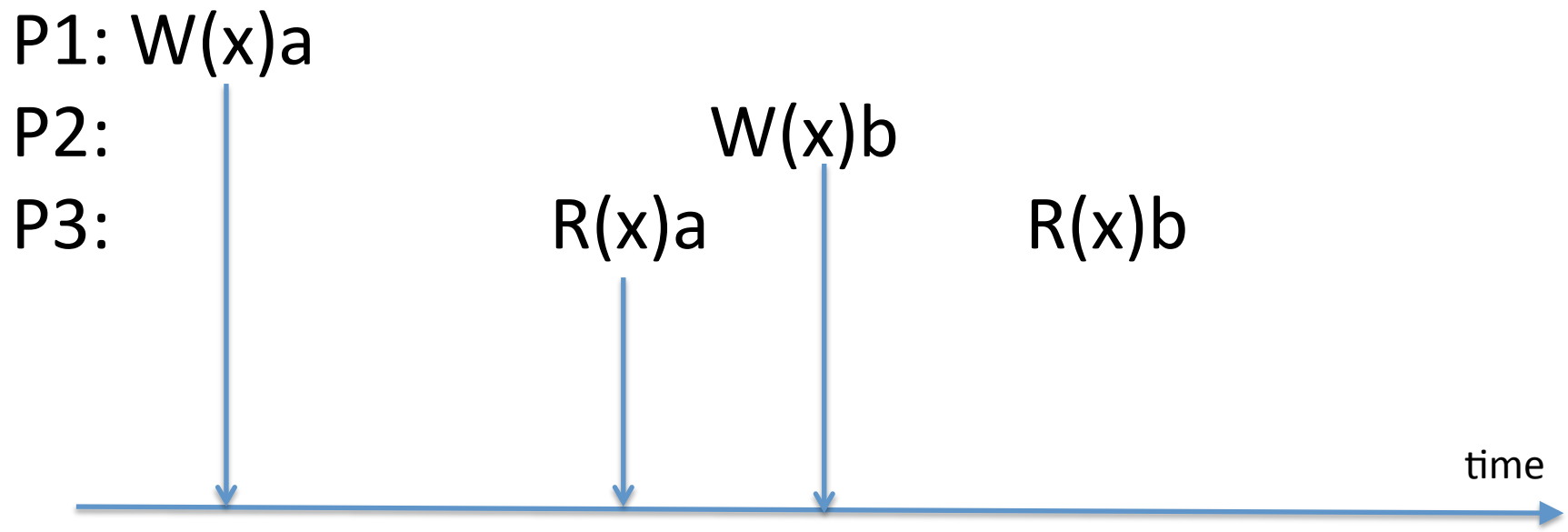
(And when do we stop writing?)

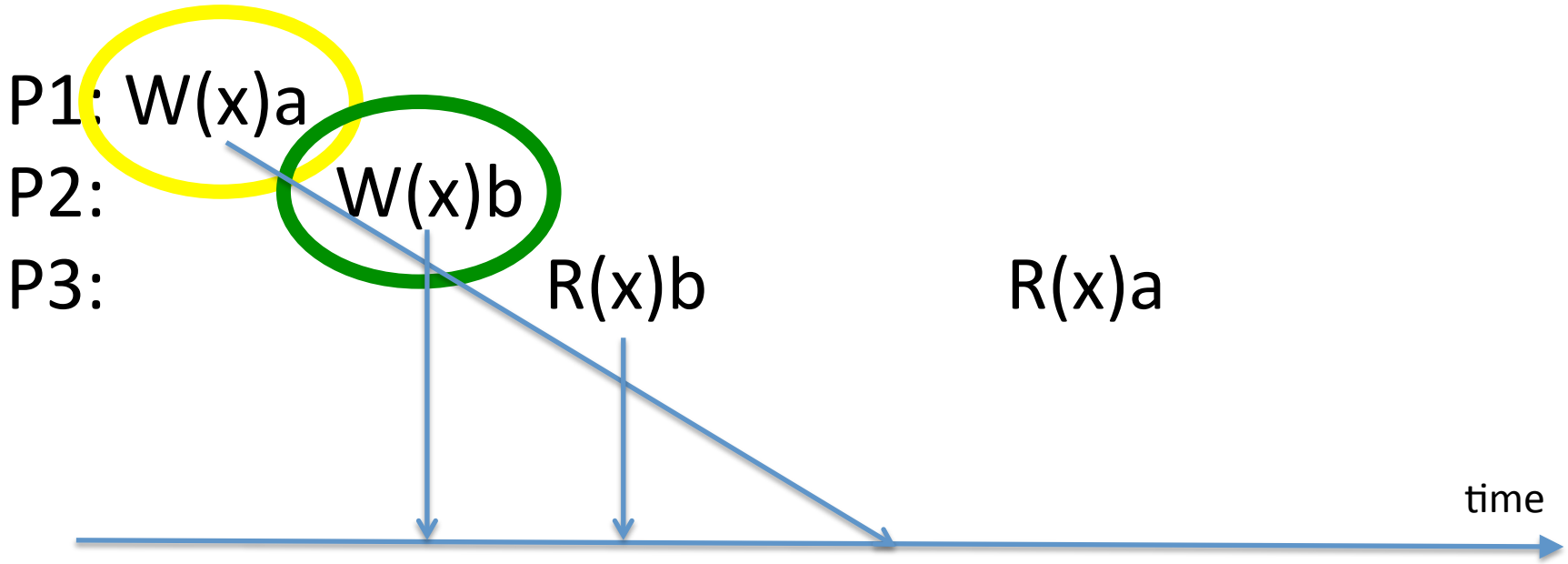
(And what about multi-key transactions?)

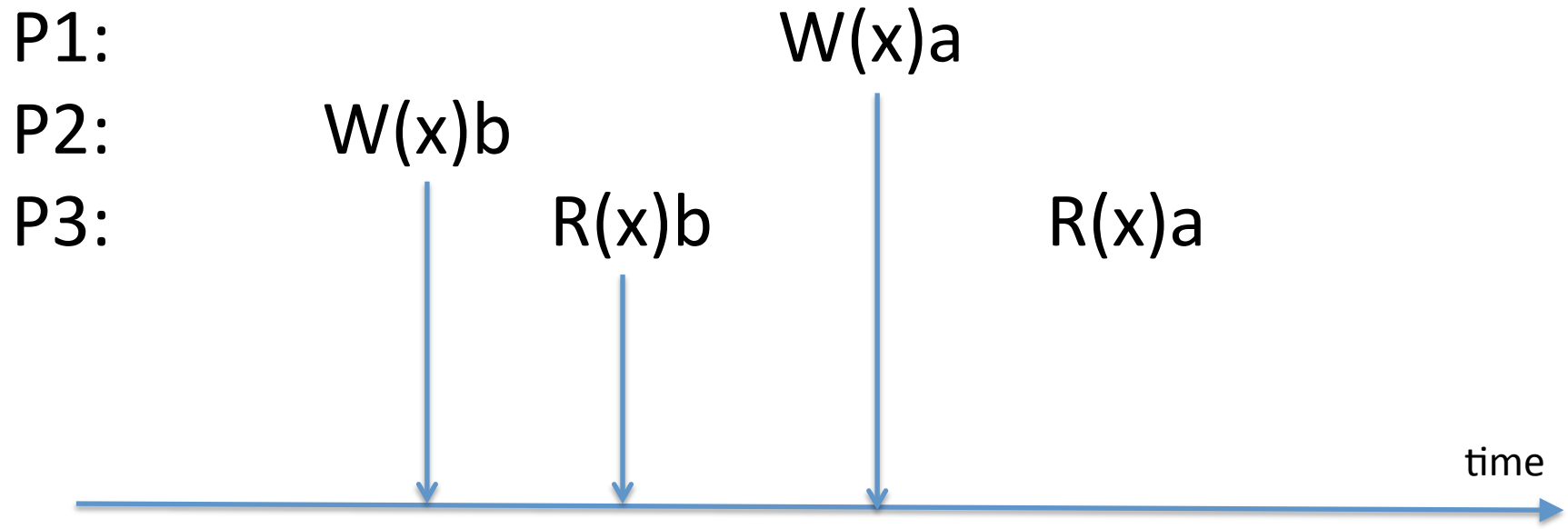
Sequential consistency: cache coherence




P1: W(x)a
P2:
W(x)b
P3:
R(x)a
R(x)b
time


P1: W(x)a
P2:
W(x)b
P3:
R(x)a
R(x)b
time


P1: W(x)a
P2: W(x)b
P3: R(x)b R(x)a
time


P1: W(x)a
P2: W(x)b
P3: R(x)b R(x)a
time

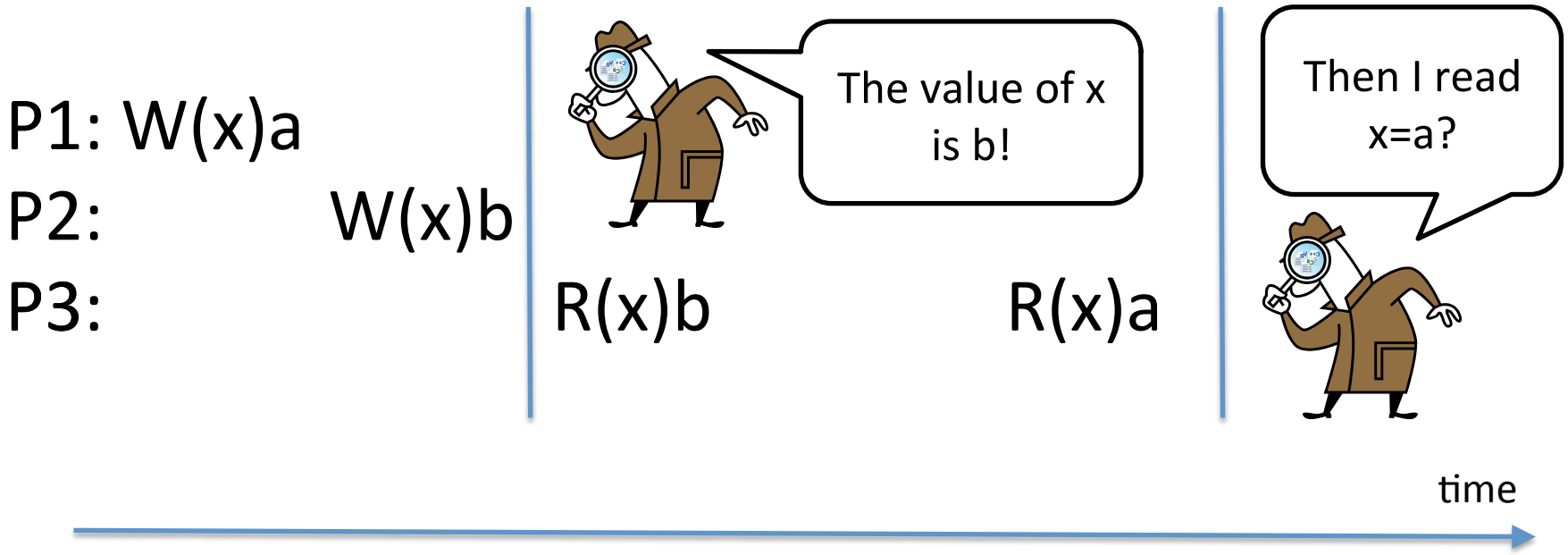
External Consistency

Everything that sequential consistency has

Except results actually match *time.*

An external observer

Not Externally Consistent



CAP Theorem

- Brewer's PODC talk: "Consistency, Availability, Partition-tolerance: choose two" in 2000
  - Partition-tolerance is a failure model
  - Choice: can you process reads and writes during a partition or not?

- FLP result – "Impossibility of Distributed Consensus with One Faulty Process" in 1985
  - Asynchronous model; cannot tell the difference between message delay and failure

What does this mean?

It’s impossible to decide anything on the internet?

NP-hard

What does CAP mean?

It's impossible to 100% of the time decide everything on the internet if we can't rely on synchronous messaging

We can 100% of the time decide everything if partitions heal (we know the upper bound on message delays)

We can still play Candy Crush

~~CAP~~
Consistency vs. Performance

Consistency (like serializability) requires communication and blocking

How do we reduce these costs while:

- Producing a correct ordering of reads and writes and
- Handling failures and (eventually) making progress?

Improving Serializability Performance

Technique	Systems
Atomic clocks to bound time skew	Spanner
Transaction chopping	Lynx, ROCOCO
Commutative locking	Escrow transactions, abstract data types, Doppel
Deterministic ordering	Granola, Calvin

Goal: parallel performance

- Different concurrency control schemes for popular, contended data
- Commutative locking
- Abstract datatypes
- Per-core (or per-server) data and constraints

Operation Model

Developers write transactions as stored procedures which are composed of *operations* on keys and values:

Traditional key/value operations	`value GET(k)`	**Replicate for reads**
	`void PUT(k,v)`	**Save last write**
Operations on numeric values which modify the existing value	`void INCR(k,n)`	**Replicate for commutative operations**
	`void MAX(k,n)`	
	`void MULT(k,n)`	
Ordered `PUT`, insert to an ordered list, user-defined functions	`void OPUT(k,v,o)`	
	`void TOPK_INSERT(k,v,o)`	
	`void UDF(k,v,a)`	**Log operations**

Spanner/F1

“We believe it is better to have application programmers deal with performance problems due to overuse of transactions as bottlenecks arise, rather than always coding around the lack of transactions.”

Takeaways

- Use well-tested, long-lived database systems
- Use `SERIALIZABLE` until it becomes a performance problem
- Think about what is changing when you move to systems with different models

Thanks!

narula@mit.edu
http://nehanaru.la
@neha

CSAIL

The Stata Center via emax: http://hip.cat/emax/

CHICAGO
INTERNATIONAL
SOFTWARE DEVELOPMENT
CONFERENCE 2015

goto; conference

Questions?

Please remember to evaluate via the GOTO Guide App

follow us @gotochgo

Conference: May 11-12 / Workshops: 13-14