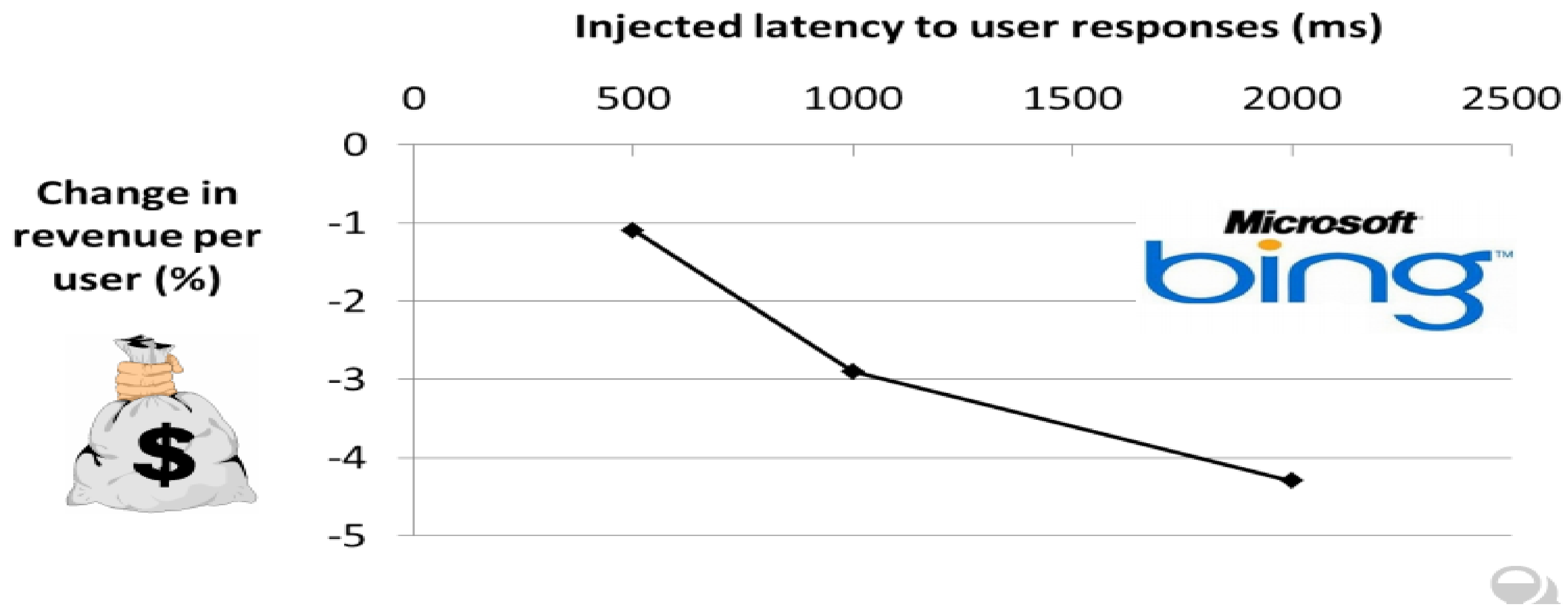
MAKING GEO-REPLICATED SYSTEMS FAST AS POSSIBLE, CONSISTENT WHEN NECESSARY

Authors : *Cheng Li, Daniel Porto, Allen Clement, Johannes Gehrke, Nuno Preguiça, and Rodrigo Rodrigues*

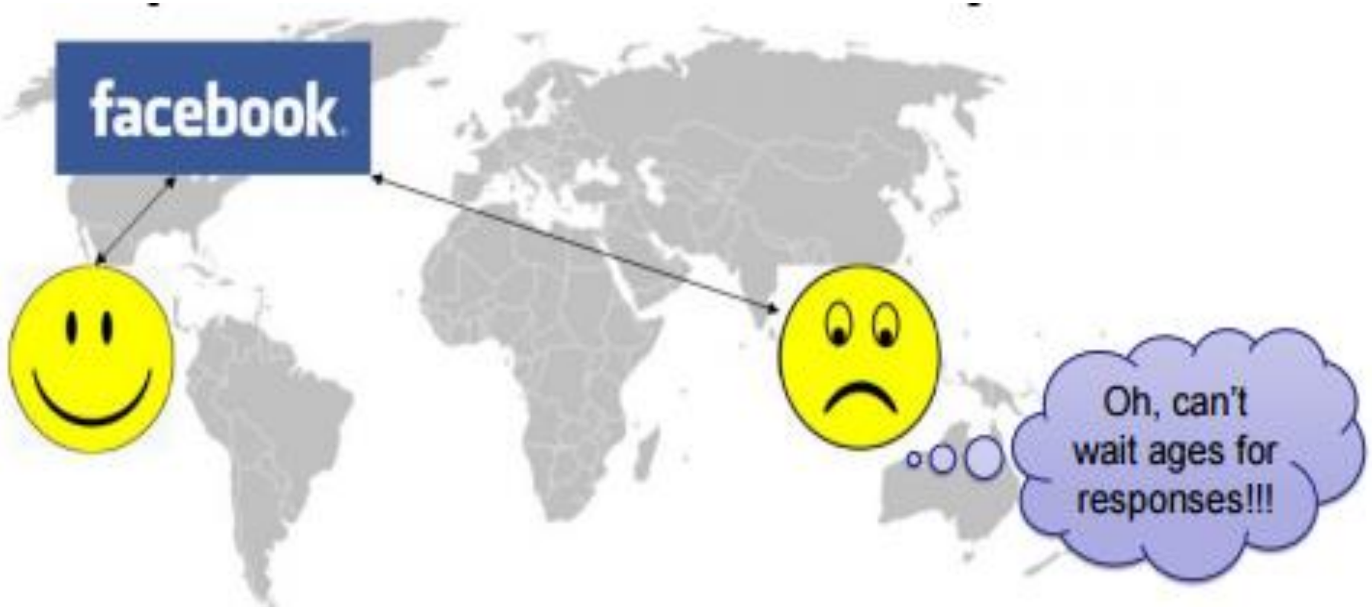
Presenter: Devesh Kumar Singh

GEO-REPLICATION



- Internet users globally distributed
- Higher Latency => poor user experience => loss in revenue
- Replicate data across geographically diverse sites
- Serve users from closest/least loaded site
- Need to decide on consistency models

STRONG CONSISTENCY



- Single server behavior with natural semantics like linearizability
- Coordination overhead between replicas, amplified in geo-replication
- High Latency for remote clients
- e.g Yahoo PNUTS

EVENTUAL CONSISTENCY

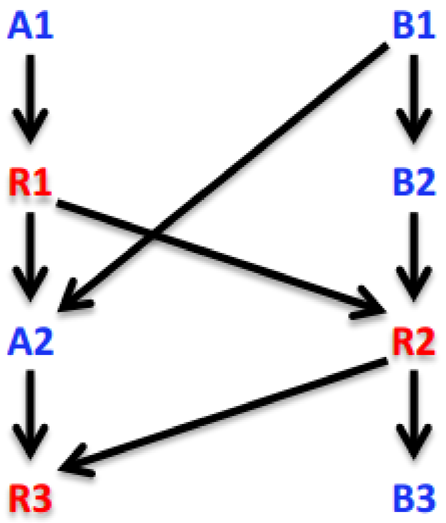
- Multi server behavior with short term state divergence
- Conflict resolution by last writer wins etc.
- Low latency for remote users, might cause undesirable behavior
- e.g Amazon Dynamo

RED-BLUE CONSISTENCY

Strong consistency

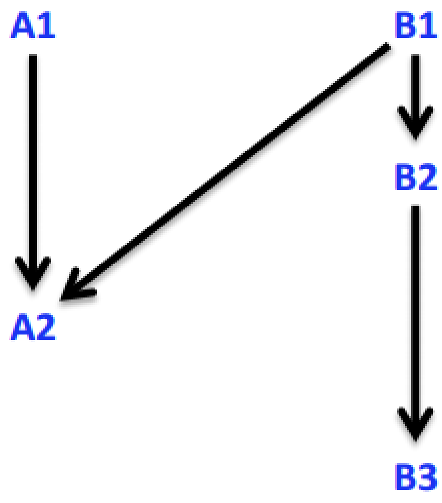
Totally ordered Red ops

Red-Blue consistency



Low latency blue ops when possible, Coordination for Red ops only when necessary

Eventual Consistency



Partially ordered Blue ops

RED-BLUE CONSISTENCY

- RedBlue order
  - Red operations must be totally ordered
  - Blue operations order can vary from site to site

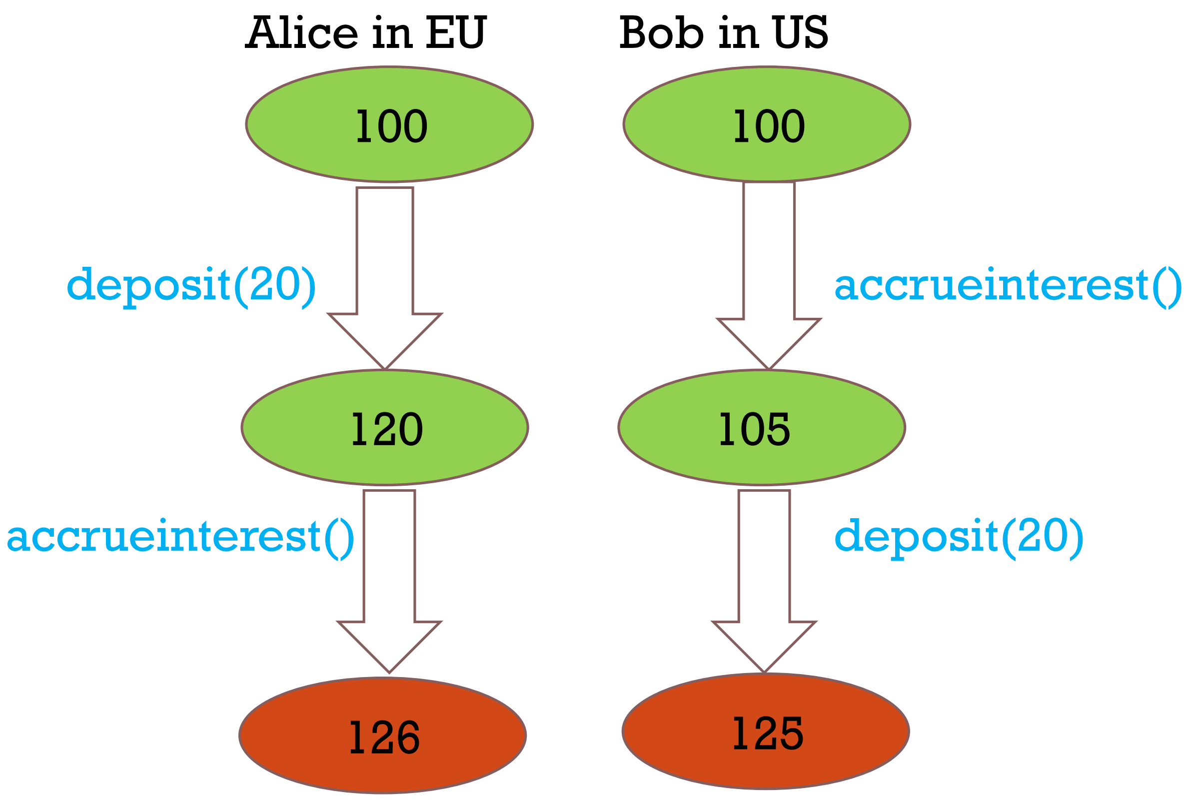
Site A: **A1 B1 R1 B2 A2 R2 R3 B3**

Site B: **B1 B2 A1 R1 R2 A2 B3 R3**

- Causal serialization
  - A site has a causal serialization of the RedBlue order if the ordering is a linear extension of the RedBlue order
- State convergence
  - All causal serializations of the RedBlue order reach the same state
  - All Blue orders must be globally commutative
- Red Blue Consistency
  - Each site applies operations according to the causal serialization of the RedBlue order

RED-BLUE CONSISTENT BANK SYSTEM

Initial: balance = 100, interest = 0.05




```
deposit(float money) {
  balance = balance + money;
}

withdraw(float money) {
 if ( balance - money >= 0 )
   balance = balance - money;
 else
   print "failure";
}

accrueinterest() {
  float delta = balance ×
interest;
  balance = balance + delta;
}
```

RED-BLUE CONSISTENT BANK SYSTEM

- Problem: Different execution order lead to divergent state
- Cause: accrueinterest doesn't commute with deposit
- Solution: Mark all as Red for convergence, but Red is slow
- Better Solution: Split non-commutative operations into two
  - Compute the amount of interest accrued
  - Treat computed value as deposit

accrueinterest(): delta = balance * interest balance = balance + delta	→	accrueinterest_gen(): delta = balance * interest accrueinterest(delta): balance = balance + delta

GENERATOR/SHADOW OPERATIONS

- Generator Operation
  - Only executed at the primary site against a system state
  - Produces no side effects
  - Determines state transitions that would occur
  - Produces shadow operations
- Shadow Operation
  - Applies the state transitions to all the sites including the primary site
  - Must produce the same effects as the original operation given the original state for the Generator operation

BANKING SYSTEM REVISITED

Original/Generator operation		Shadow operation

deposit(**float** *m*){
 balance = balance + m;
 }
	produces	deposit'(**float** *m*){
		 balance = balance + m;
		 }

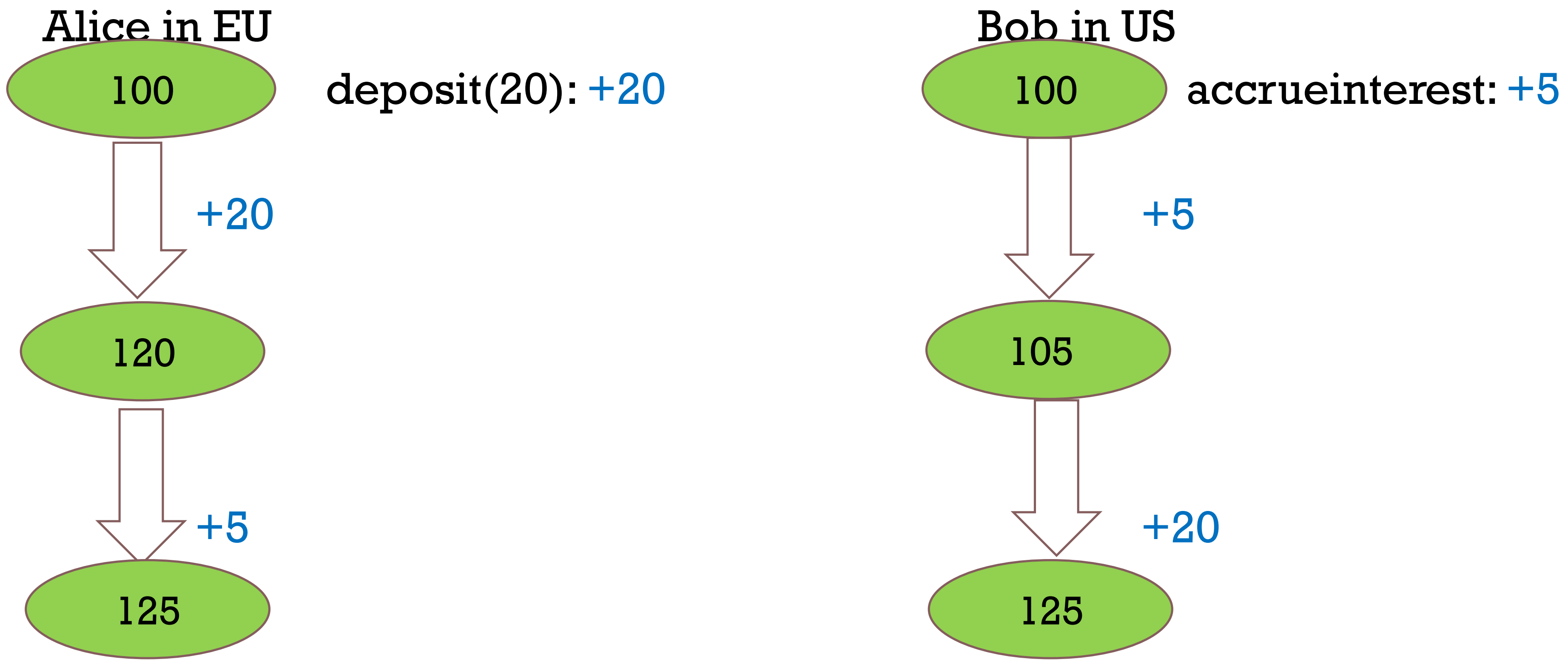
accrueinterest(){
 float *delta=balance × interest;*
 balance=balance + delta;
 }
	produces	accrueinterest'(**float** *delta*){
		 balance=balance + delta;
		 }

withdraw(**float** *m*){ **if**(*balance-m>=0*) *balance=balance - m;*	*produces*	withdrawAck'(**float** *m*) { *balance=balance - m;* }

else
 print "Error"
 }
	produces	withdrawFail'(){
		 }

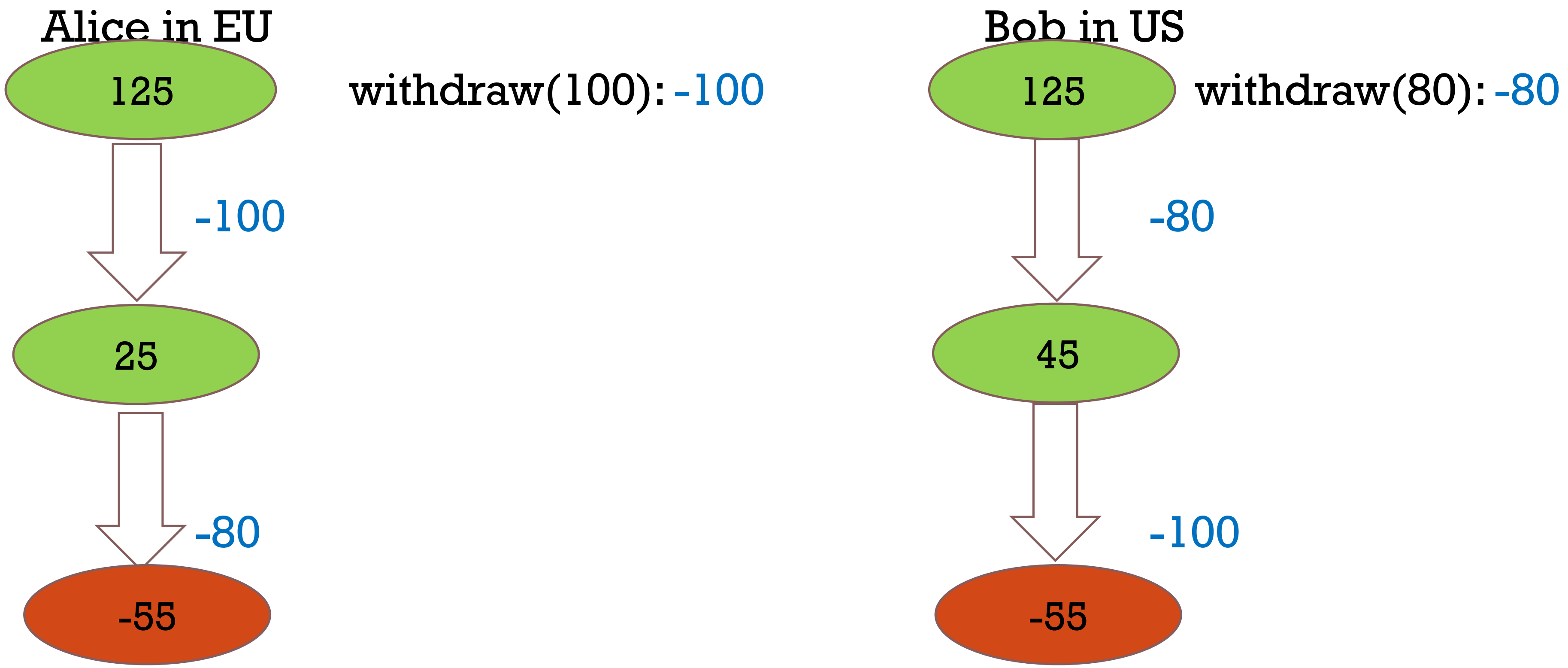
FAST AND CONSISTENT BANK

Initial: balance = 100, interest = 0.05



ANOTHER ISSUE

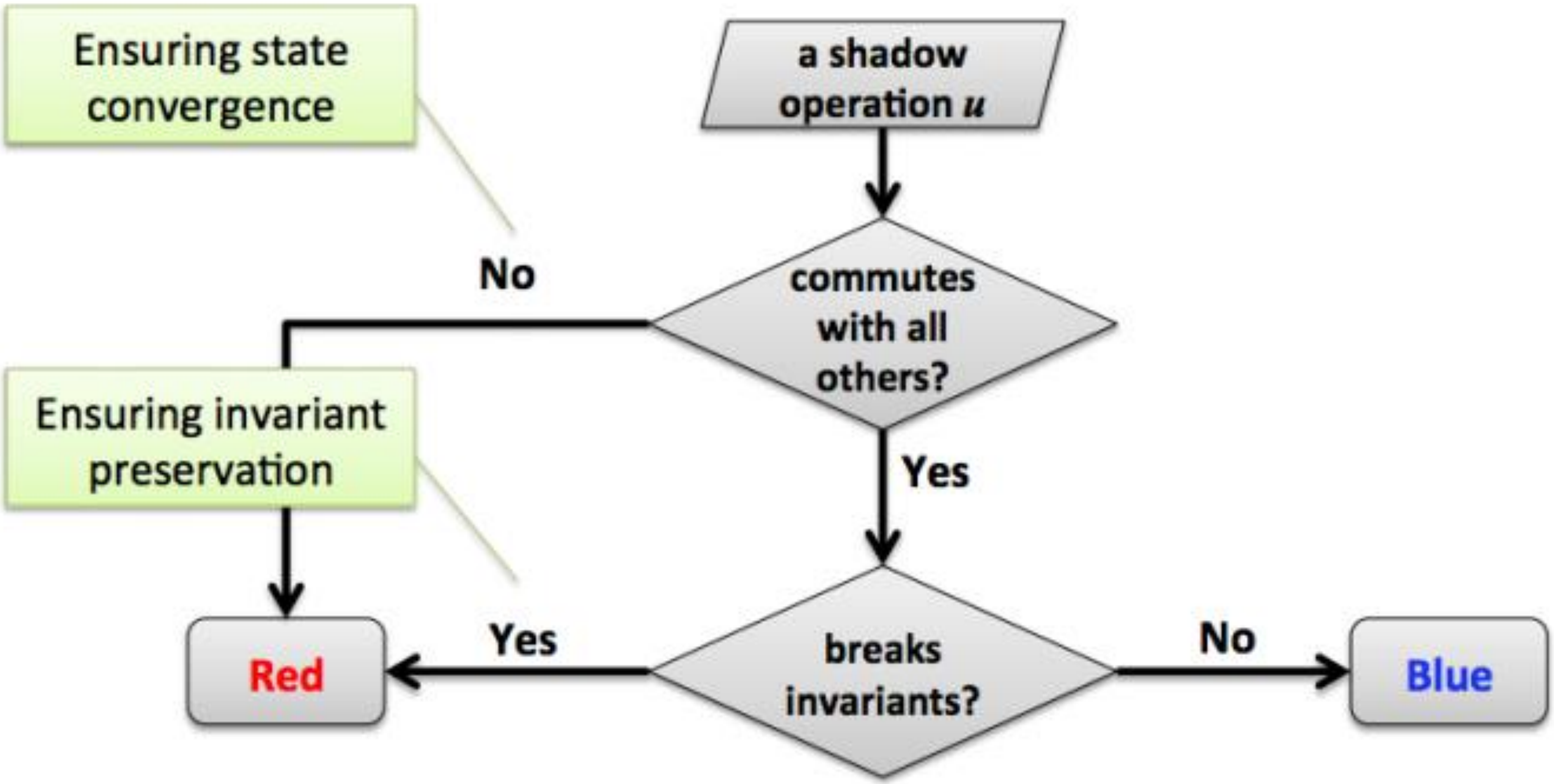
Initial: balance = 100, interest = 0.05



ANOTHER ISSUE

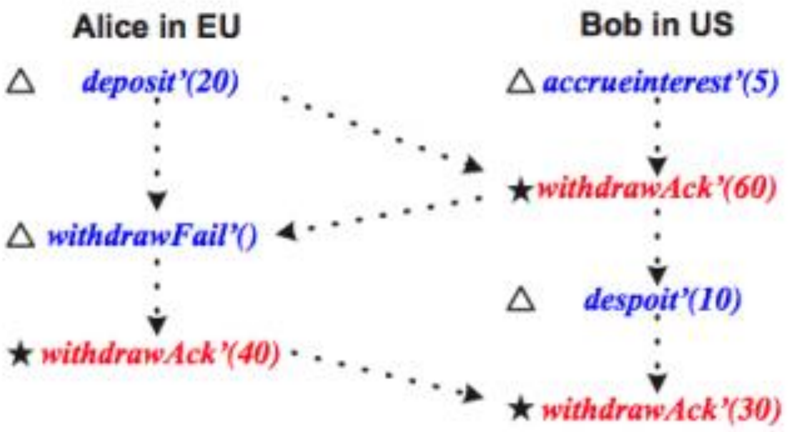
- Problem: Different execution orders lead to a negative balance.
- Cause: Blue operations that potentially break invariants execute without coordination.
- Solution: We must label successful withdrawal (withdrawAck’) as Red

RED OR BLUE?

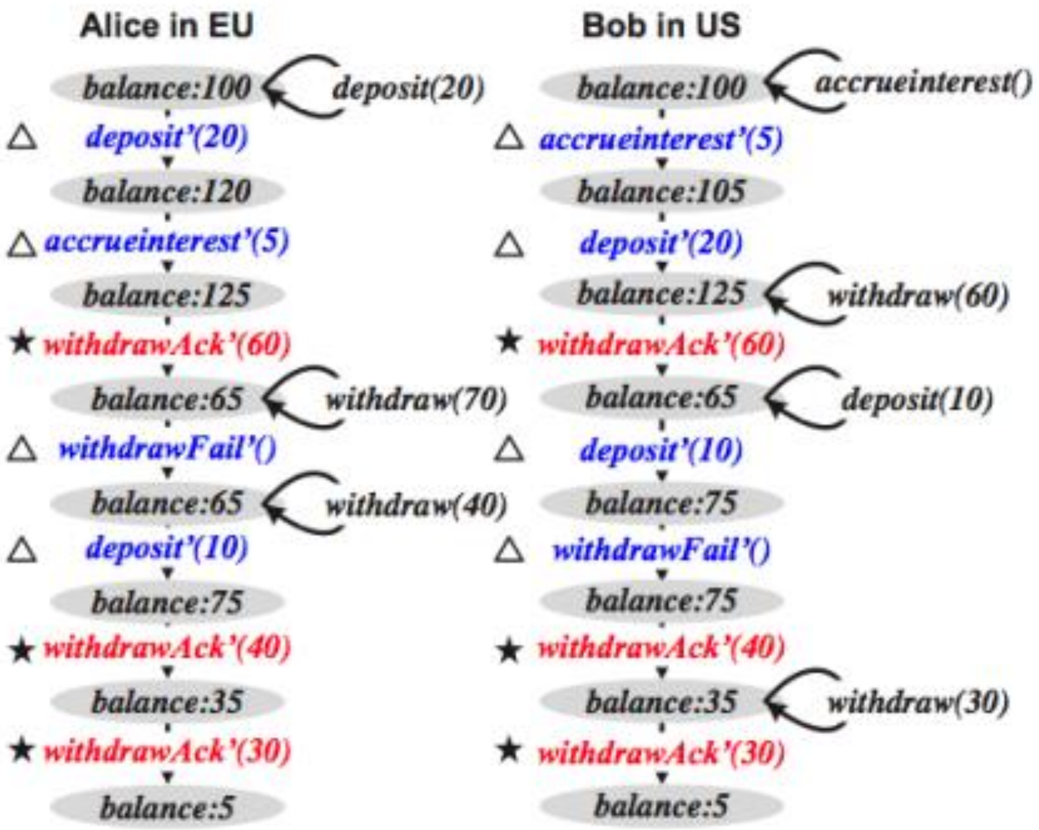


Ensuring state convergence
a shadow operation *u*
No
commutes with all others?
Yes
Ensuring invariant preservation
Yes
breaks invariants?
No
Red
Blue

RED BLUE CONSISTENT BANKING



(a) RedBlue order O of banking shadow operations



(b) Convergent and invariant preserving causal serializations of O

EVALUATION

- Experiments with:
  - E-commerce benchmarks: TPC-W, RUBiS
  - Social networking app: Quoddy
- Deployment in Amazon EC2
  - spanning 5 sites (US-East, US-West, Ireland, Brazil, Singapore) –
  - locating users in all five sites and directing their requests to closest server

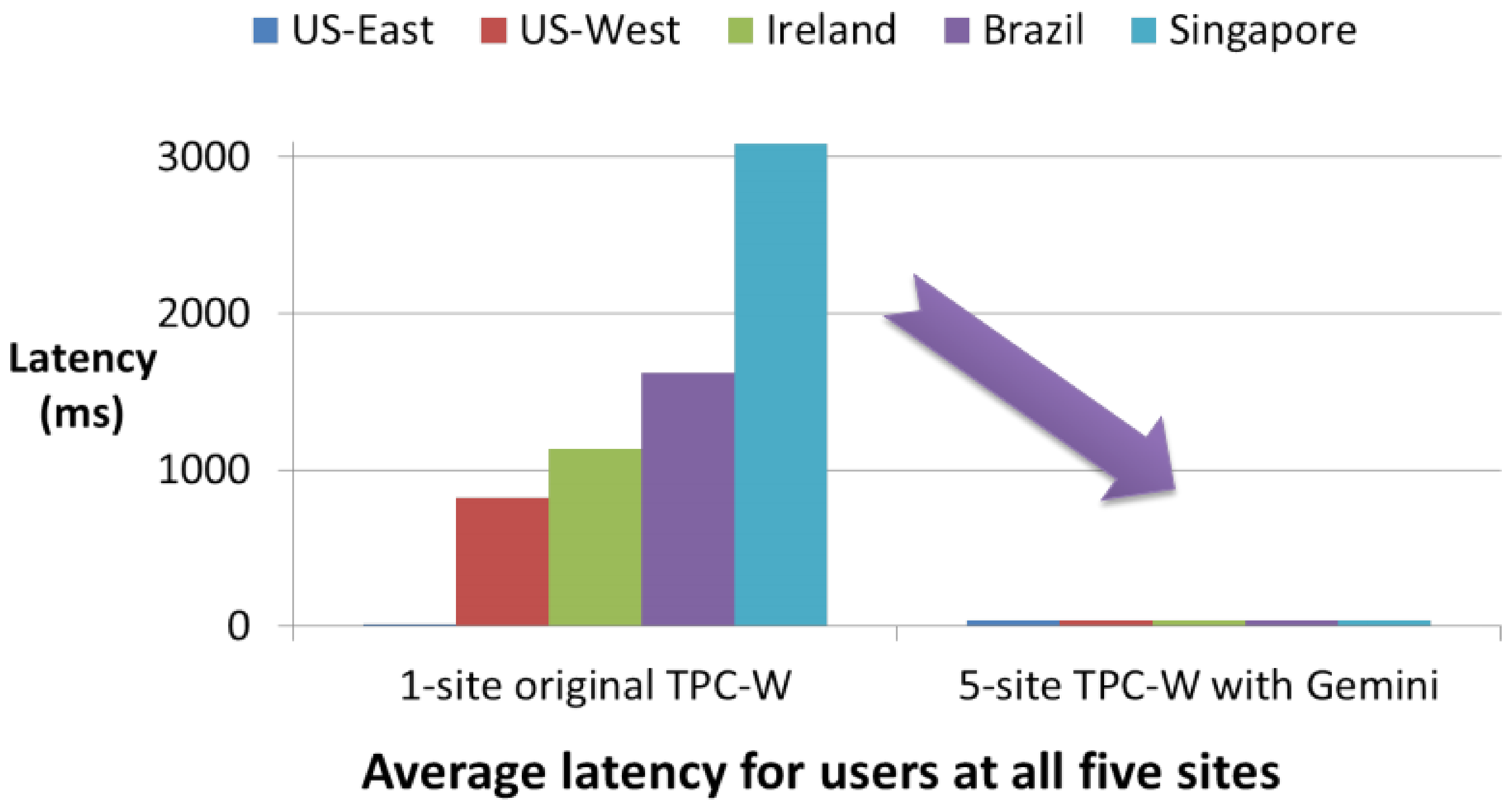
MOST OPERATIONS ARE BLUE

Apps	# Original update txns	# Blue/Red update ops	# Shadow ops	# Blue/Red update ops
TPC-W	7	0/7	16	14/2
RUBiS	5	0/5	9	7/2
Quoddy	4	0/4	4	4/0

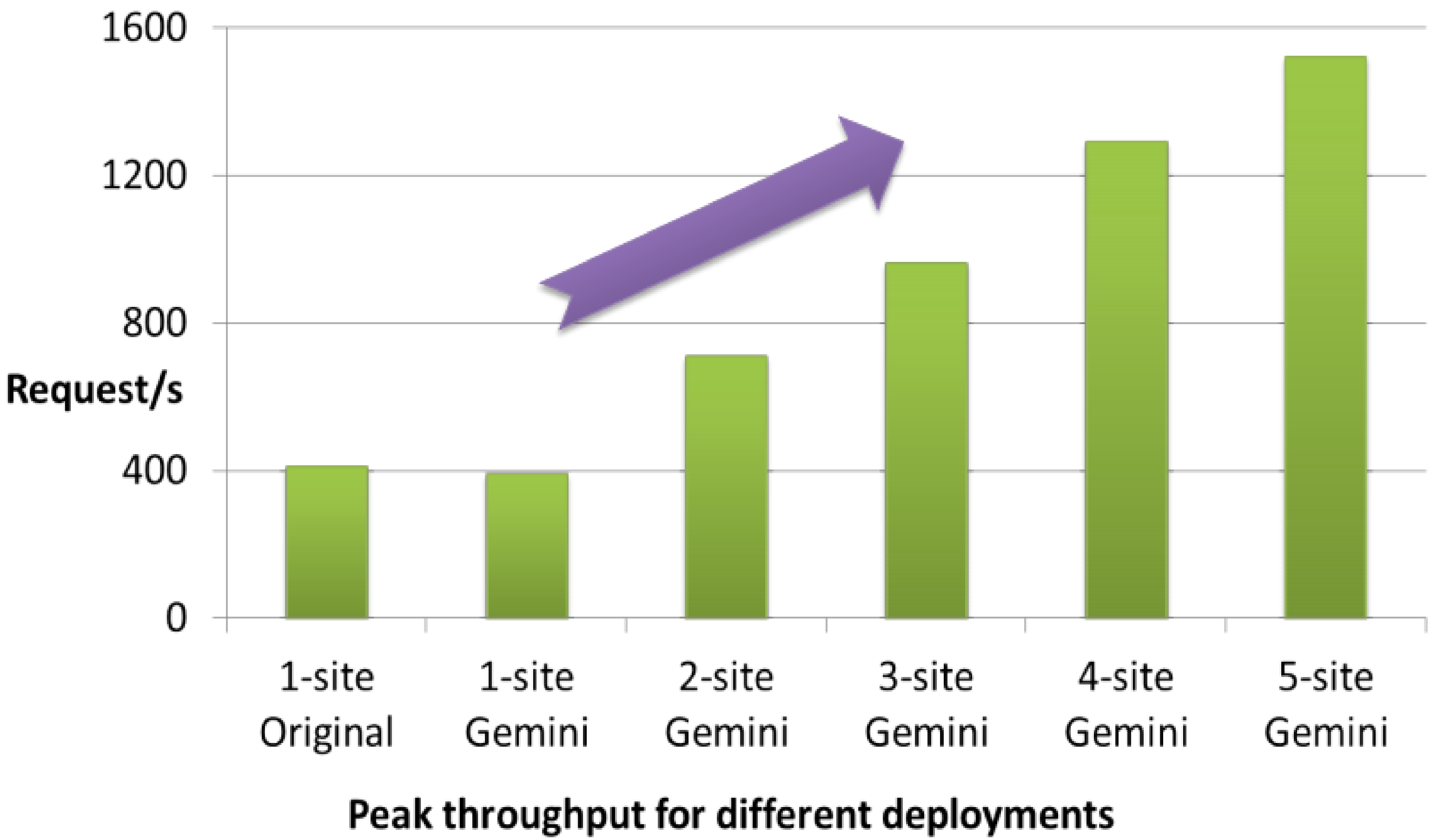
MOST OPERATIONS ARE BLUE

Apps	workload	Originally		With shadow ops	
		Blue (%)	Red(%)	Blue (%)	Red(%)
TPC-W	Browsing mix	96.0	4.0	99.5	0.5
	Shopping mix	85.0	15.0	99.2	0.8
	Ordering mix	63.0	37.0	93.6	6.4
RUBiS	Bidding mix	85.0	15.0	97.4	2.6
Quoddy	a mix with 15% update	85.0	15.0	100	0

IMPROVED USER OBSERVED LATENCY



THROUGHPUT SCALES WITH NO OF SITES



SUMMARY

- RedBlue consistency allows strong consistency and eventual consistency to coexist.
- Generator/shadow operation extends the space of fast operations.
- A precise labeling methodology allows for systems to be fast and behave as expected.
- Experimental results show our solution improves both latency and throughput.

THANK YOU