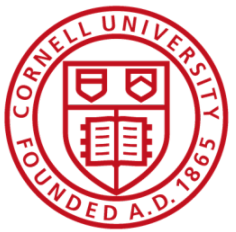
Making Geo-Replicated Systems Fast as Possible Consistent when Necessary

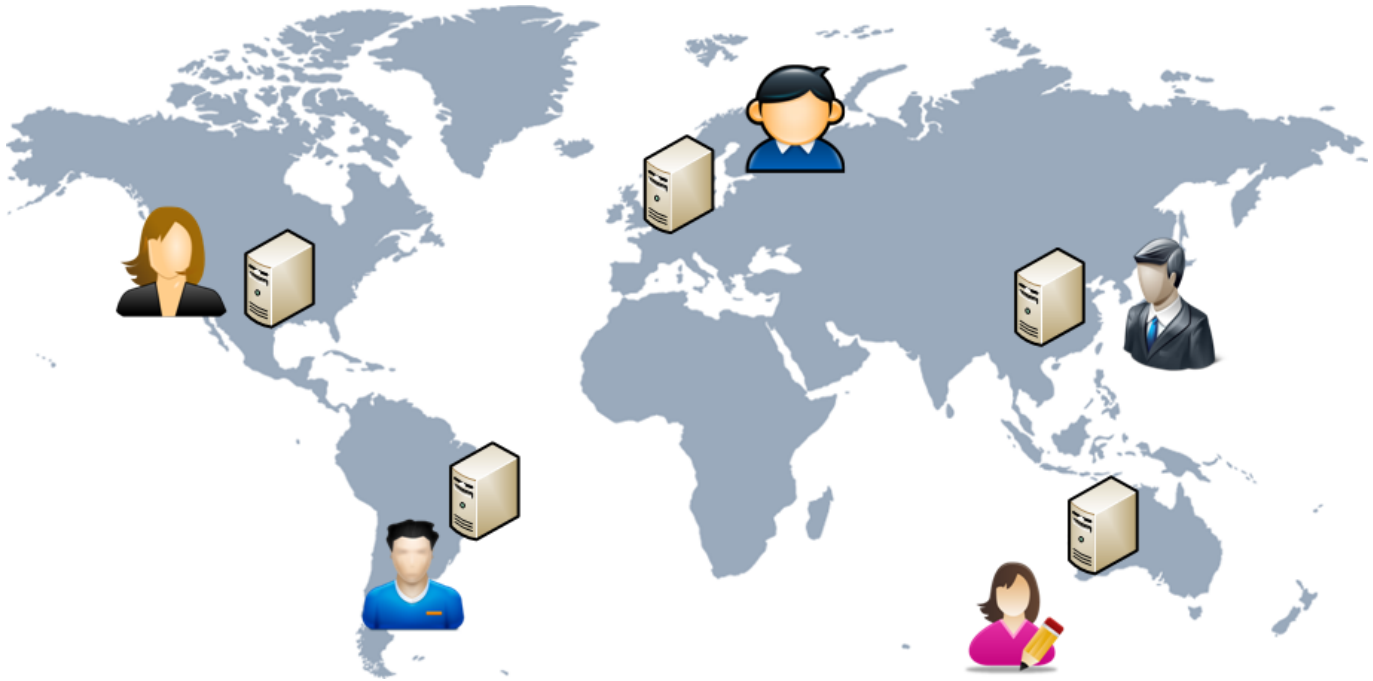
Cheng Li[†], Daniel Porto[†§], Allen Clement[†]
Johannes Gehrke[‡], Nuno Preguiça[§], Rodrigo Rodrigues[§]

Max Planck Institute for Software Systems[†],
CITI / Universidade Nova de Lisboa[§], Cornell University[‡]

Max
Planck
Institute
for
Software Systems



Geo-replication is needed!



- Geo-replication is used by major providers of Internet services.
  - e.g., Google, Amazon, Facebook, etc

Consistency or performance?

Strong consistency

- e.g., Paxos [TOCS'98]
- Pros: *Natural semantics*
- Cons: *High latency*

Eventual consistency

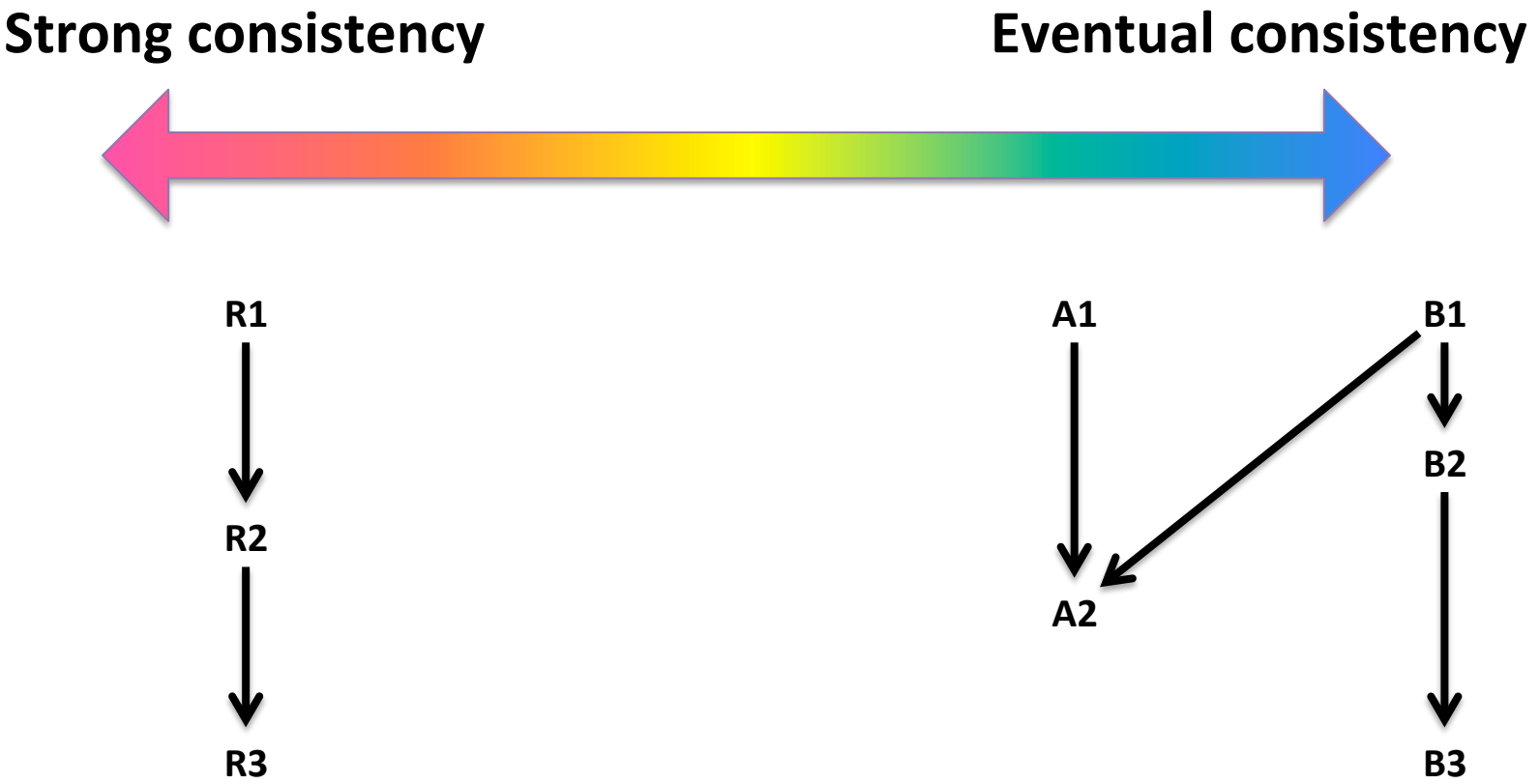
- e.g., Dynamo [SOSP'07], Bayou [SOSP'95]
- Pros: *Low latency*
- Cons: *Undesirable behaviors*

Can we build geo-replicated systems that are both fast and consistent?

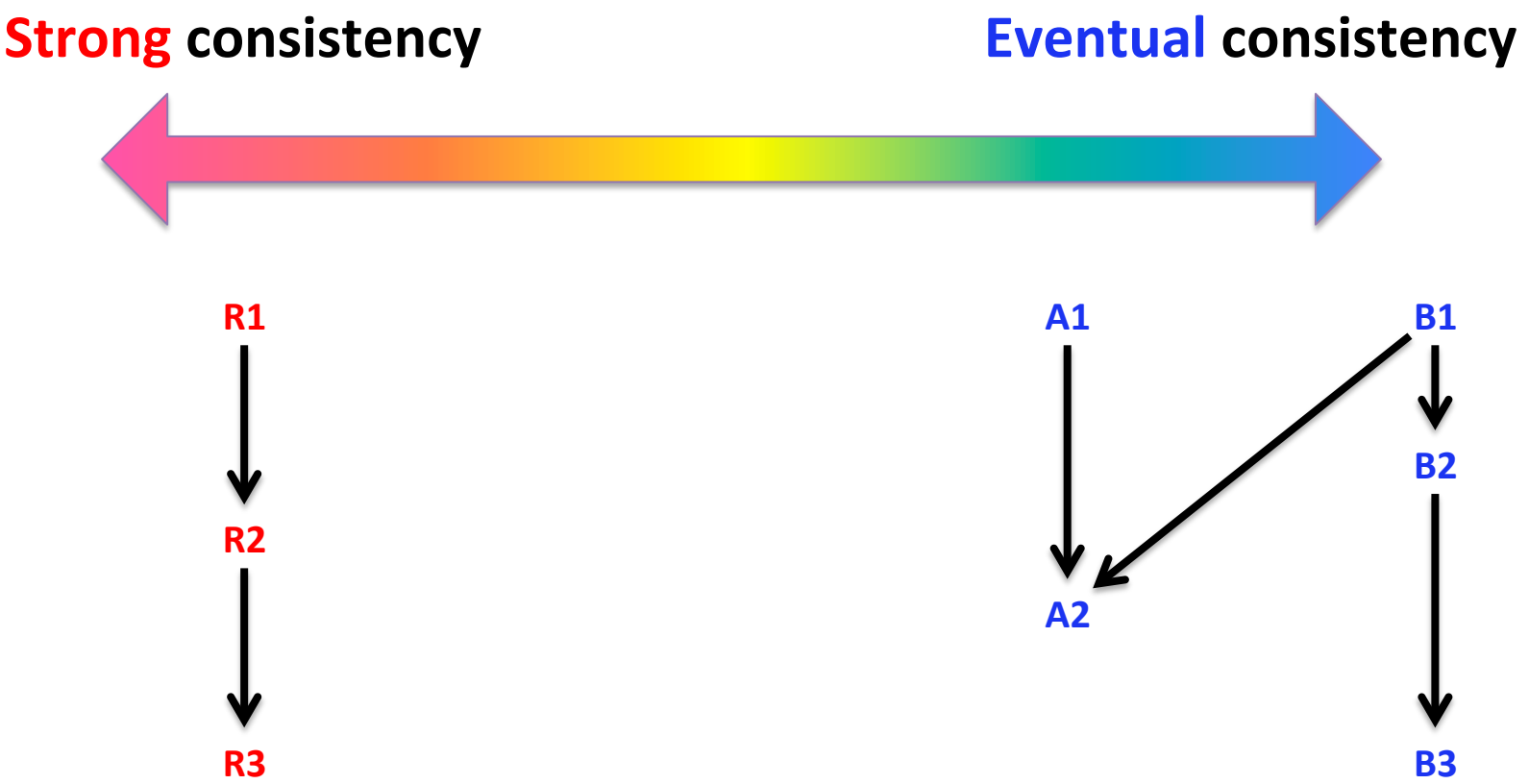
Outline

- Mixing strong and eventual consistency in a single system
- Transforming applications to safely leverage eventual consistency when possible
- Evaluation
- Red/Blue and Swiftcloud/CRDTs

Balance strong/eventual consistency



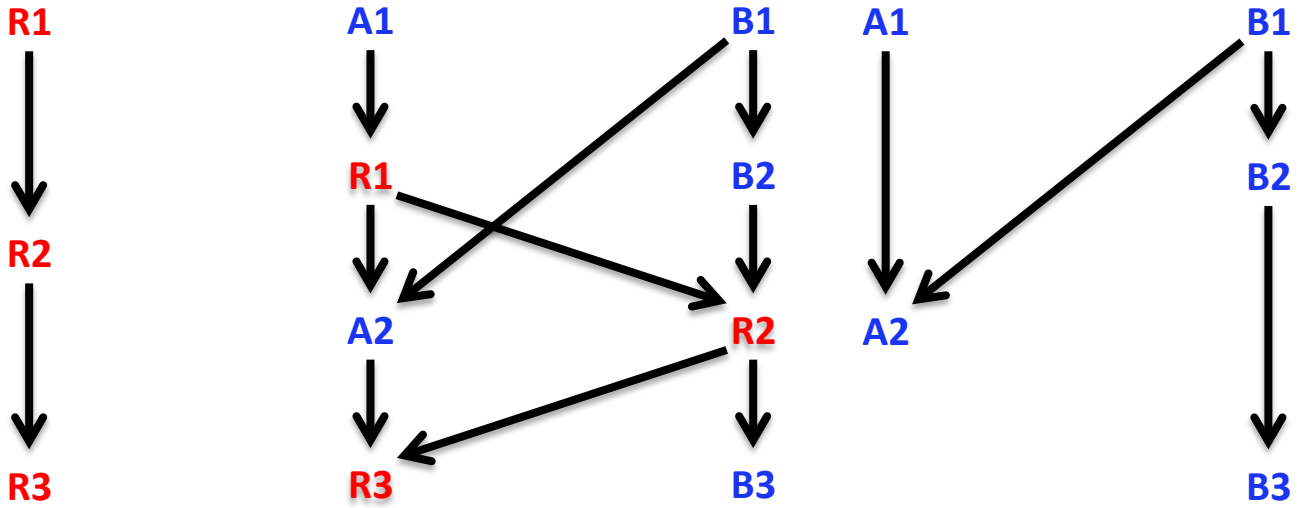
Balance strong/eventual consistency



Balance strong/eventual consistency

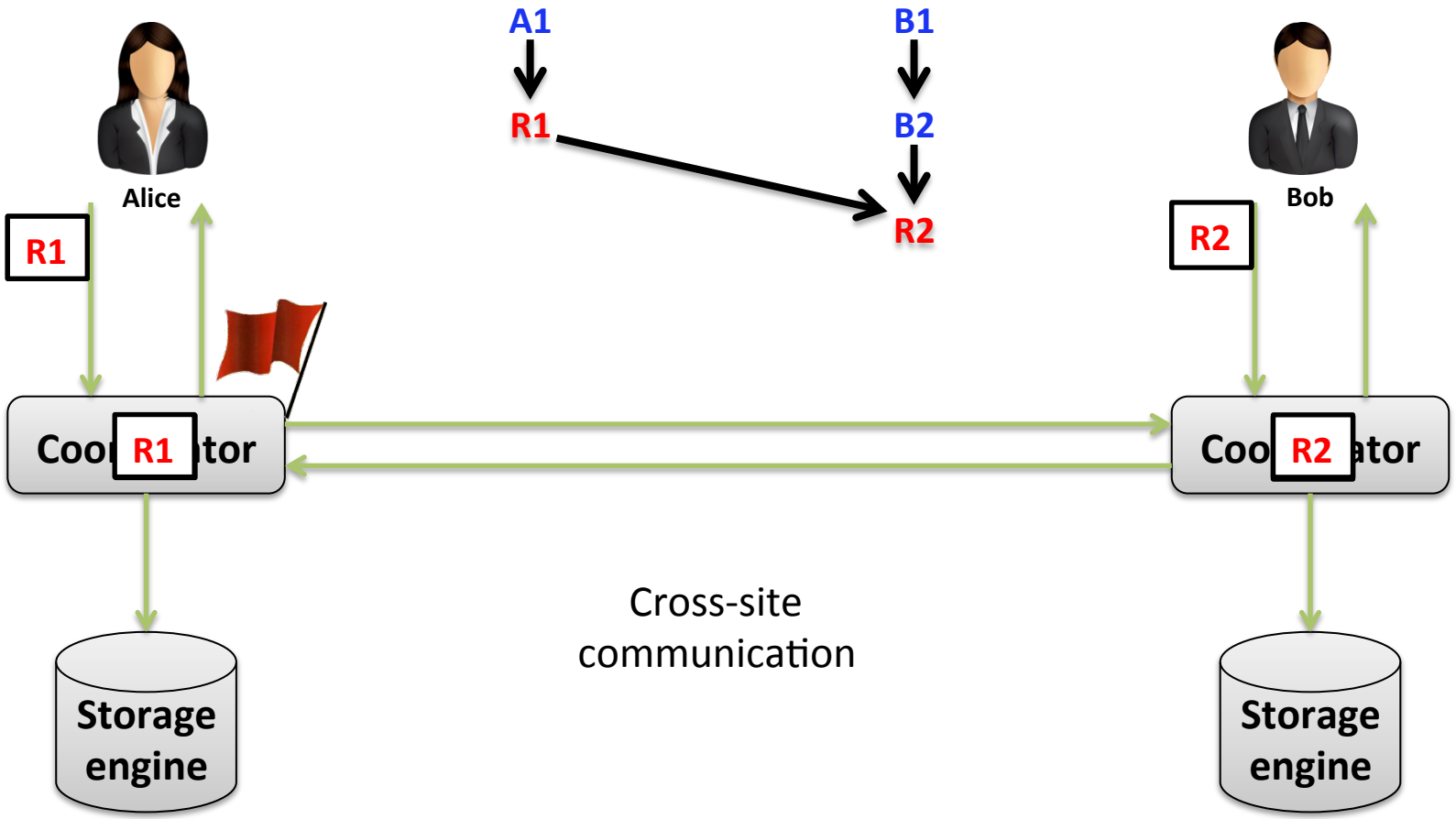
Strong consistency RedBlue **Eventual consistency**

- Low latency of eventual consistency when possible
- Coordination for strong consistency only when necessary

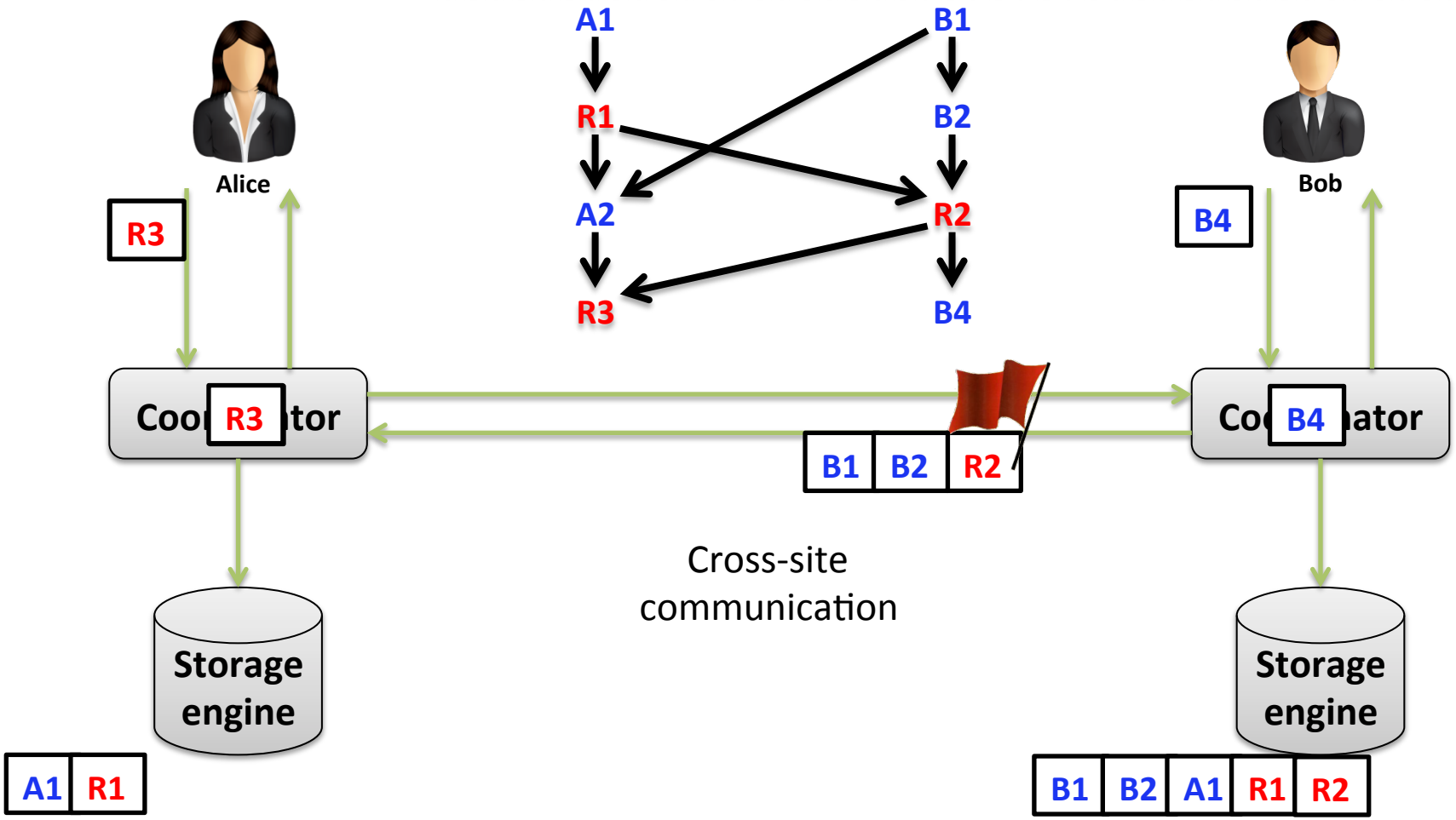


CENTER FOR INFORMATICS AND INFORMATION TECHNOLOGIES

Gemini coordination system



Gemini coordination system



CENTER FOR INFORMATICS AND INFORMATION TECHNOLOGIES

A RedBlue consistent bank system

A RedBlue consistent bank system

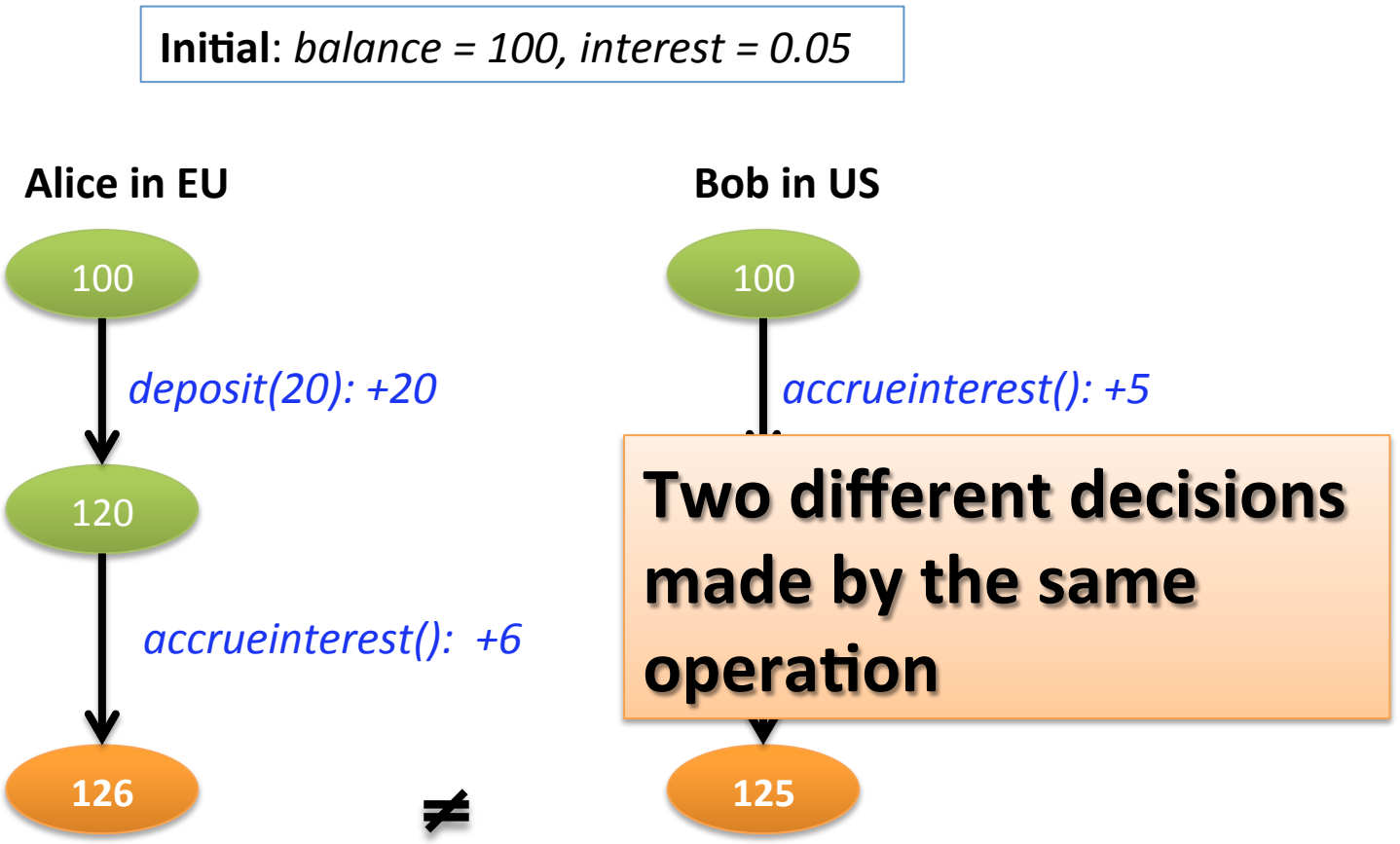
- **Problem**: Different execution orders lead to divergent state.
- **Cause**: *accrueinterest* doesn't commute with *deposit*.
- **Implication**: Convergence requires **Red**, but **Red** is slow.


```
float balance, interest;

eposit(float m){
  balance = balance + m;

ccrueinterest(){
  float delta=balance × interest;
  balance=balance + delta;

ithdraw(float m){
  if(balance-m>=0)
    balance=balance - m;
  else
    print "Error"
}
```

Outline

- Mixing strong and eventual consistency in a single system
- Transforming applications to safely leverage eventual consistency when possible
- Evaluation

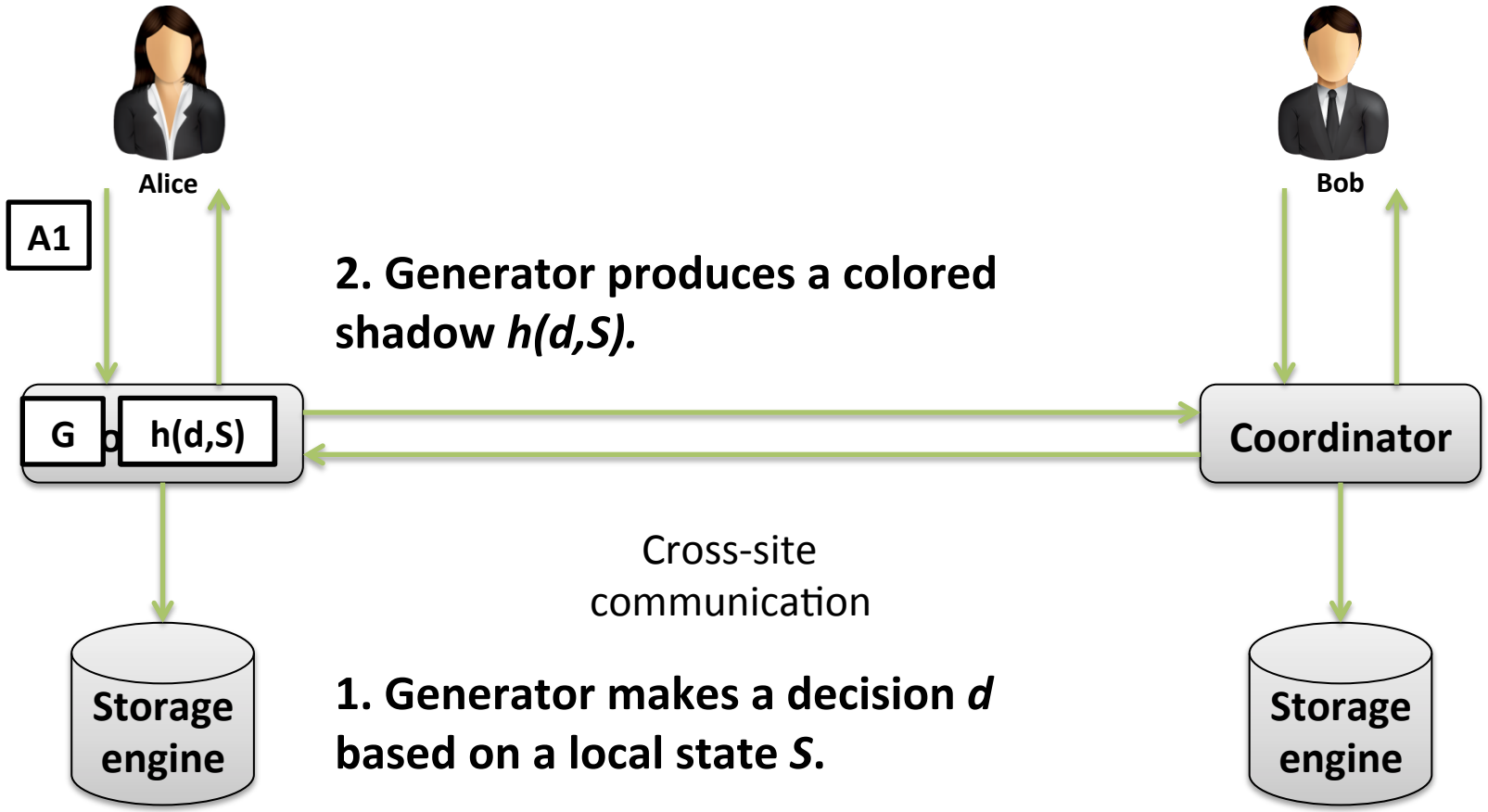
Problem of replicating operations



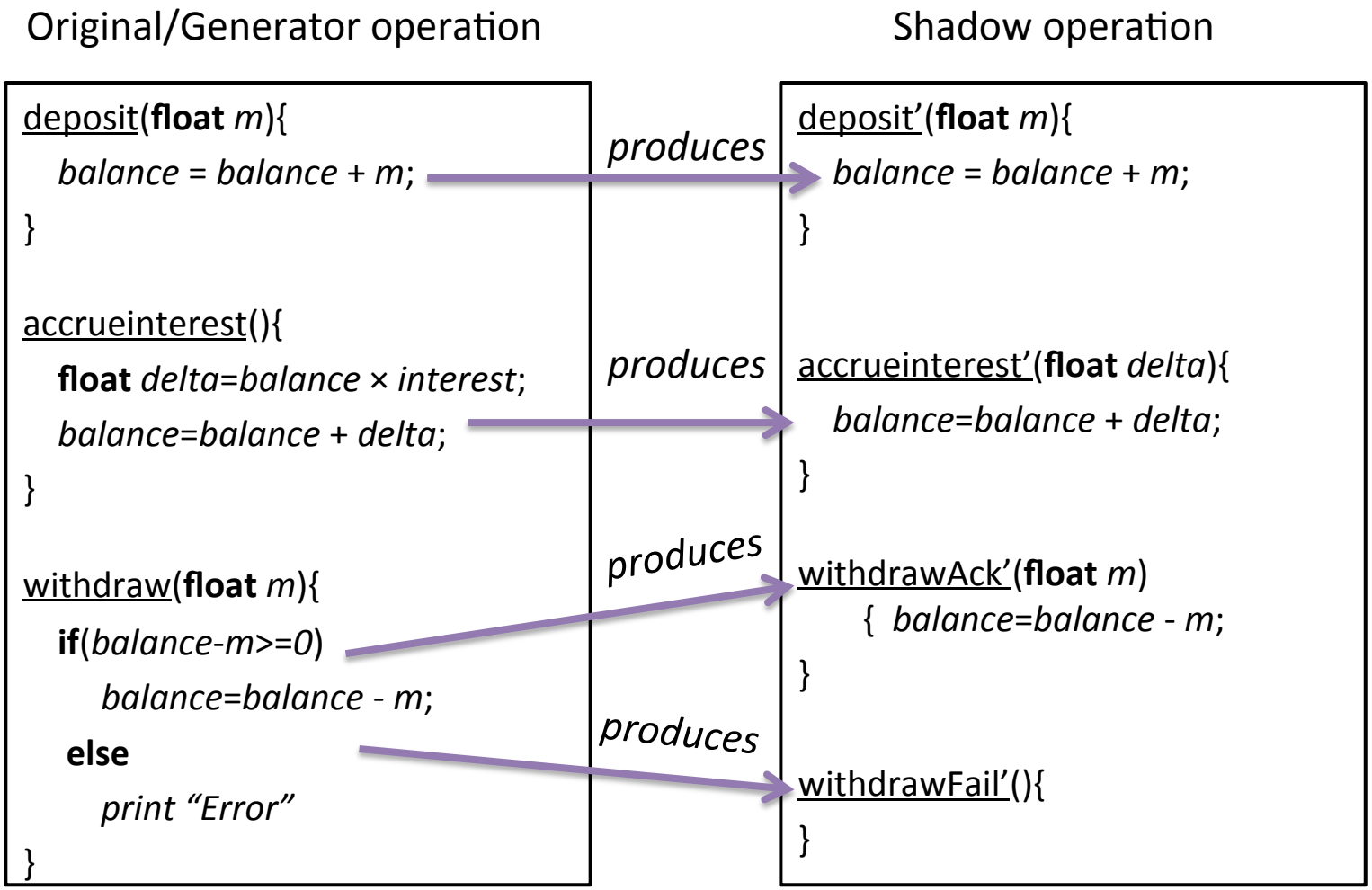
Generator/Shadow operation

- Intuitively, the execution of *accrueinterest* can be divided into:
  - A generator operation
    - decides *how much interest to be accrued*
    - has *no side effects*
  - A shadow operation
    - adds *the decided interest to the balance*

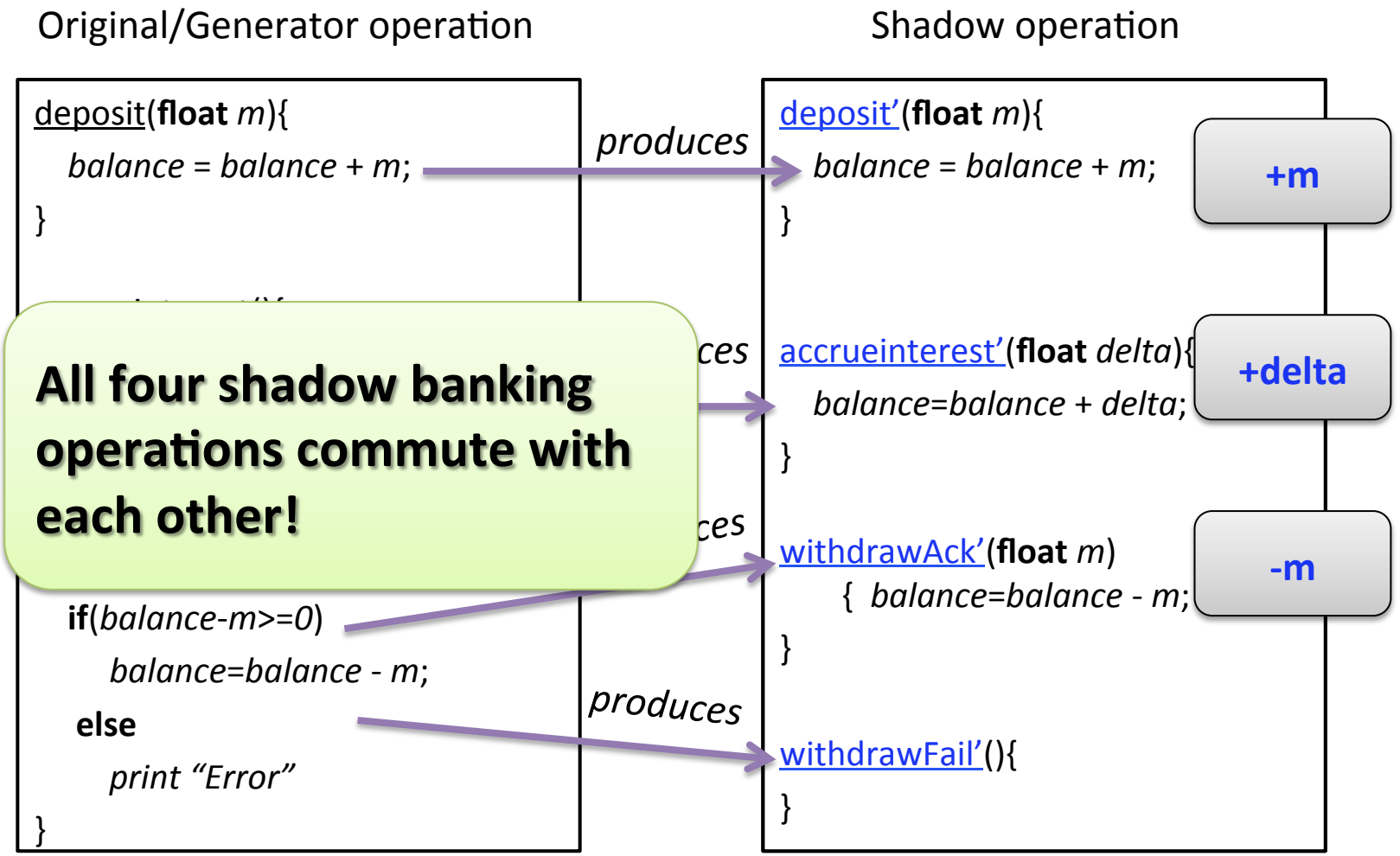
Generate once, shadow everywhere



Bank generator/shadow operations

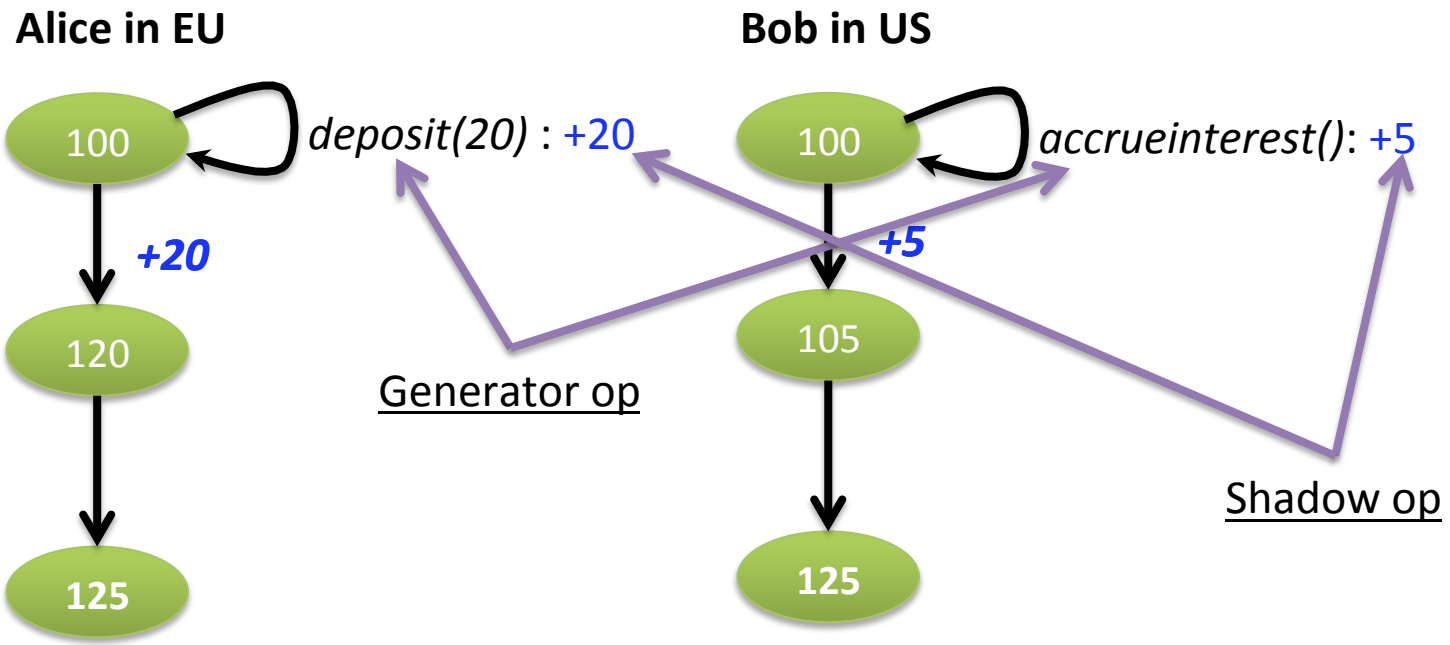


Bank generator/shadow operations



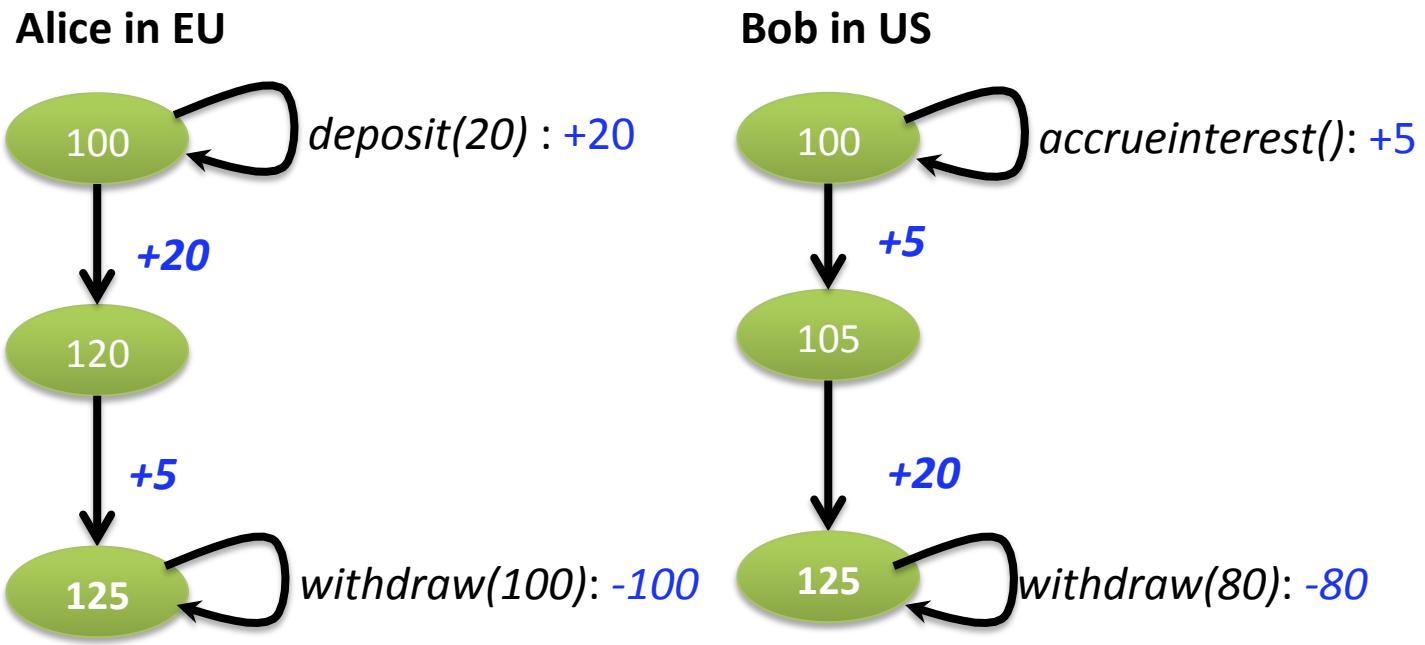
Fast and consistent bank

Initial: *balance = 100, interest = 0.05*



Not so fast …

Initial: *balance = 100, interest = 0.05*

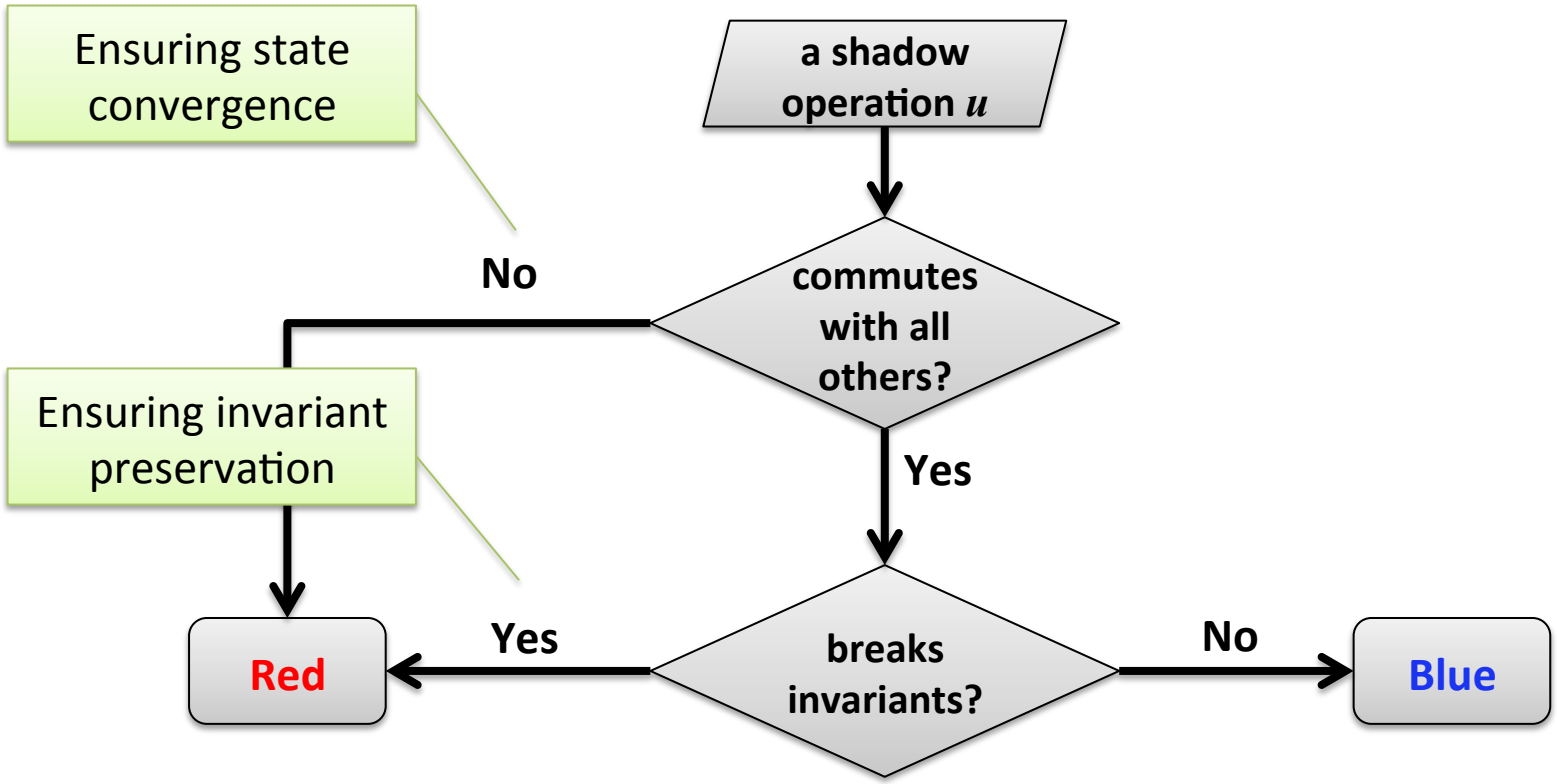


CENTER FOR INFORMATICS AND INFORMATION TECHNOLOGIES

Not so fast …

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

Which must be Red or can be Blue?



Evaluation

Questions

- How common are Blue operations?
- Does RedBlue consistency improve user-observed latency?
- Does throughput scale with the number of sites?

Questions

- How common are Blue operations?
- Does RedBlue consistency improve user-observed latency?
- Does throughput scale with the number of sites?

Case studies

- Applications:
  - Two e-commerce benchmarks: TPC-W, RUBiS
  - One social networking app: Quoddy

Apps	# Original update txns	# Blue/Red update ops
TPC-W	7	0/7
RUBiS	5	0/5
Quoddy	4	0/4

Case studies

- Applications:
  - Two e-commerce benchmarks: TPC-W, RUBiS
  - One social networking app: Quoddy

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

How common are Blue operations?

Runtime Blue/Red ratio in different applications with different workloads:

Apps	workload	Originally	
		Blue (%)	Red(%)
TPC-W	Browsing mix	**96.0**	**4.0**
	Shopping mix	**85.0**	**15.0**
	Ordering mix	**63.0**	**37.0**
RUBiS	Bidding mix	**85.0**	**15.0**
Quoddy	a mix with 15% update	**85.0**	**15.0**

How common are Blue operations?

Runtime Blue/Red ratio in different applications with different workloads:

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

The vast majority of operations are Blue.

Questions

- How common are Blue operations?
- Does RedBlue consistency improve user-observed latency?
- Does throughput scale with the number of sites?

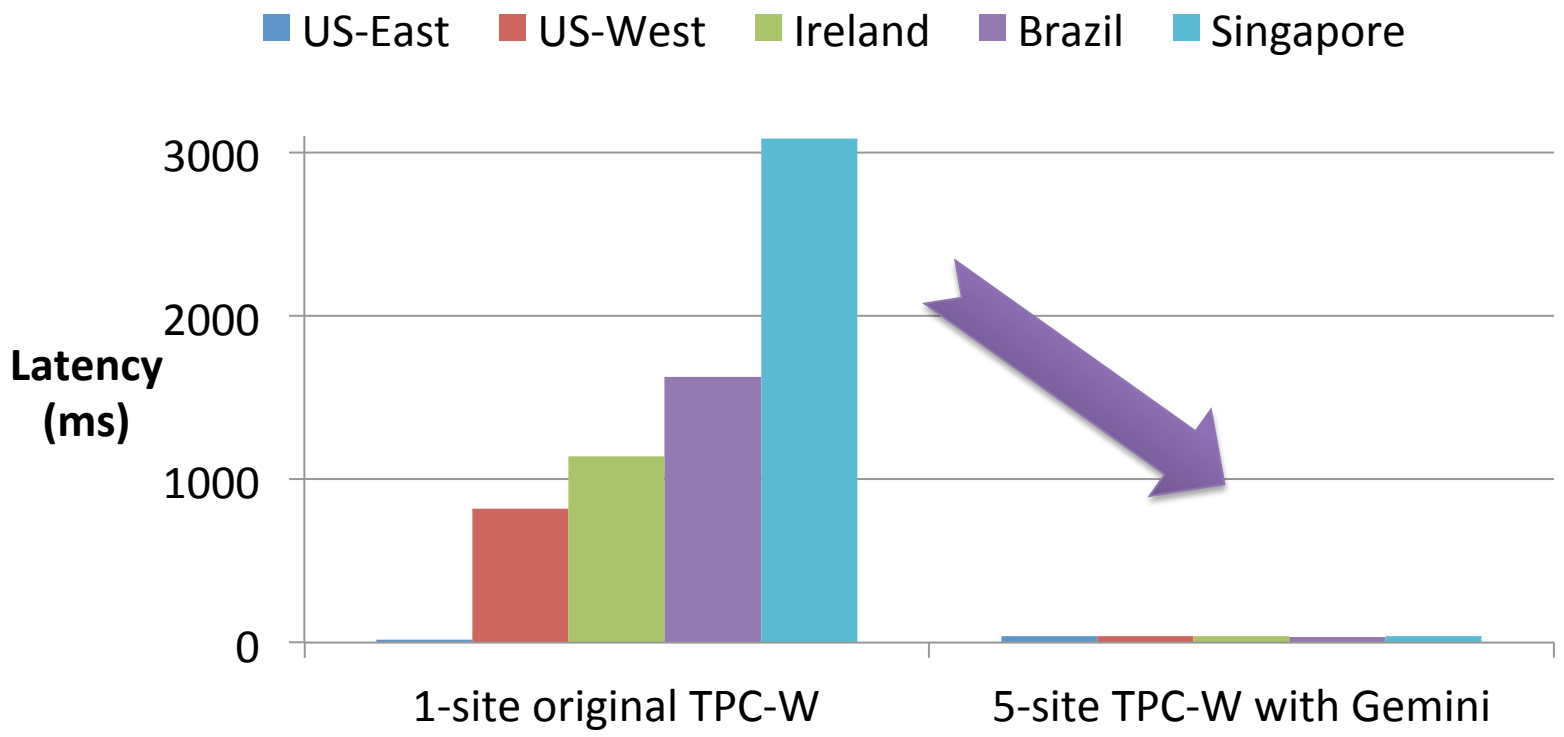
Experimental setup

- Experiments with:
  - TPC-W, RUBiS and 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

Experimental setup

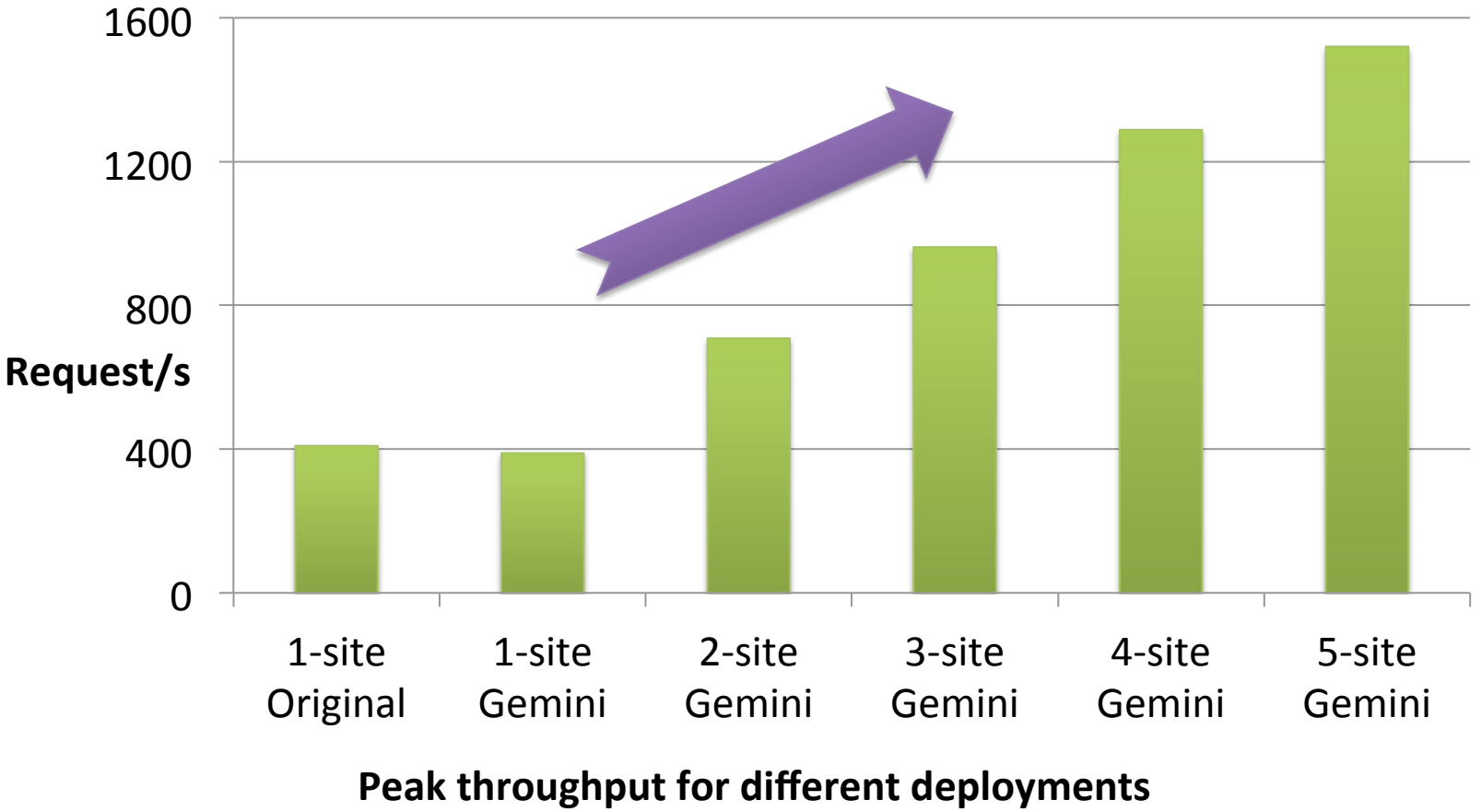
- Experiments with:
  - TPC-W, RUBiS and Quoddy
- Deployment in Amazon EC2
  - spanning 5 sites (US-East, US-West, Ireland, Brazil, Singapore)
  - locating users in all five sites and directing their requests to closest server

Does RedBlue consistency improve user-observed latency?



Average latency for users at all five sites

Does throughput scale with the number of sites?



Peak throughput for different deployments

Red-Blue and SwiftCloud

Shadow operations

- CRDT dowstream operation is a form of shadow operation
- Differences:
  - In Red-Blue, one operation can generate multiple shadow operations
  - Marc has a proposal to generalize CRDTs to include a similar possibility

Red-Blue and other forms of strong consistency

- Other forms of strong consistency usually involve locking objects
  - This tends to restrict the objects that can be accessed using weak consistency
  - E.g. in Walter, fast transaction must access only c-sets; a bank account could not be a c-set (or something similar)
- In Red-Blue, only operations that need strong consistency need to run under strong consistency (and pay the price for that)
  - E.g. in the bank account, deposit can always be fast; in a game, adding resources to a player can be fast

Red-Blue is a good match for SwiftCloud

- Previous properties make Red-Blue a good match for SwiftCloud
  - CRDT already include a form of shadow operations
  - Blue operations can continue executing without coordination
  - Only Red operations would require synchronization

- Challenges
  - How to integrate state-based CRDTs?

Beyond Red-Blue: reservations, escrow

- Even if Red-Blue can help, Red operations still require coordination
- Some Red operations are: commutative but can break invariants (e.g. withdraw in bank account)
- Explore reservation/escrow techniques to allow accepting Red operations without coordination at execution time
  - Coordination still required to obtain reservation/escrow rights

Reservations

- Goal: guarantee that an operation can be later executed without conflicts

- A reservation provides some guarantee about the future state of the database

Escrow reservation

- Exclusive right to *use* a share of a partitionable resource represented by a numerical item – guarantee that a constrains >= will not be violated
  - E.g. the stock of some product may be split among several salesmen

Value-use reservation

- Right to use some value for a given data item (despite its value when the transaction is executed)
  - E.g. a salesman may use a reserved price for some product

Lock-like reservations

- **Value-change**
  - Exclusive right to modify some data item
    - Traditional fine-grain write lock
    - E.g. right to change the name of the passenger in a record for a train seat
- **Slot**
  - Exclusive right to modify records that conform some given condition
    - Similar to predicate lock
    - E.g. right to add a record for scheduling a meeting for a given period of time in room "Xpto"

Conclusion

- 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.

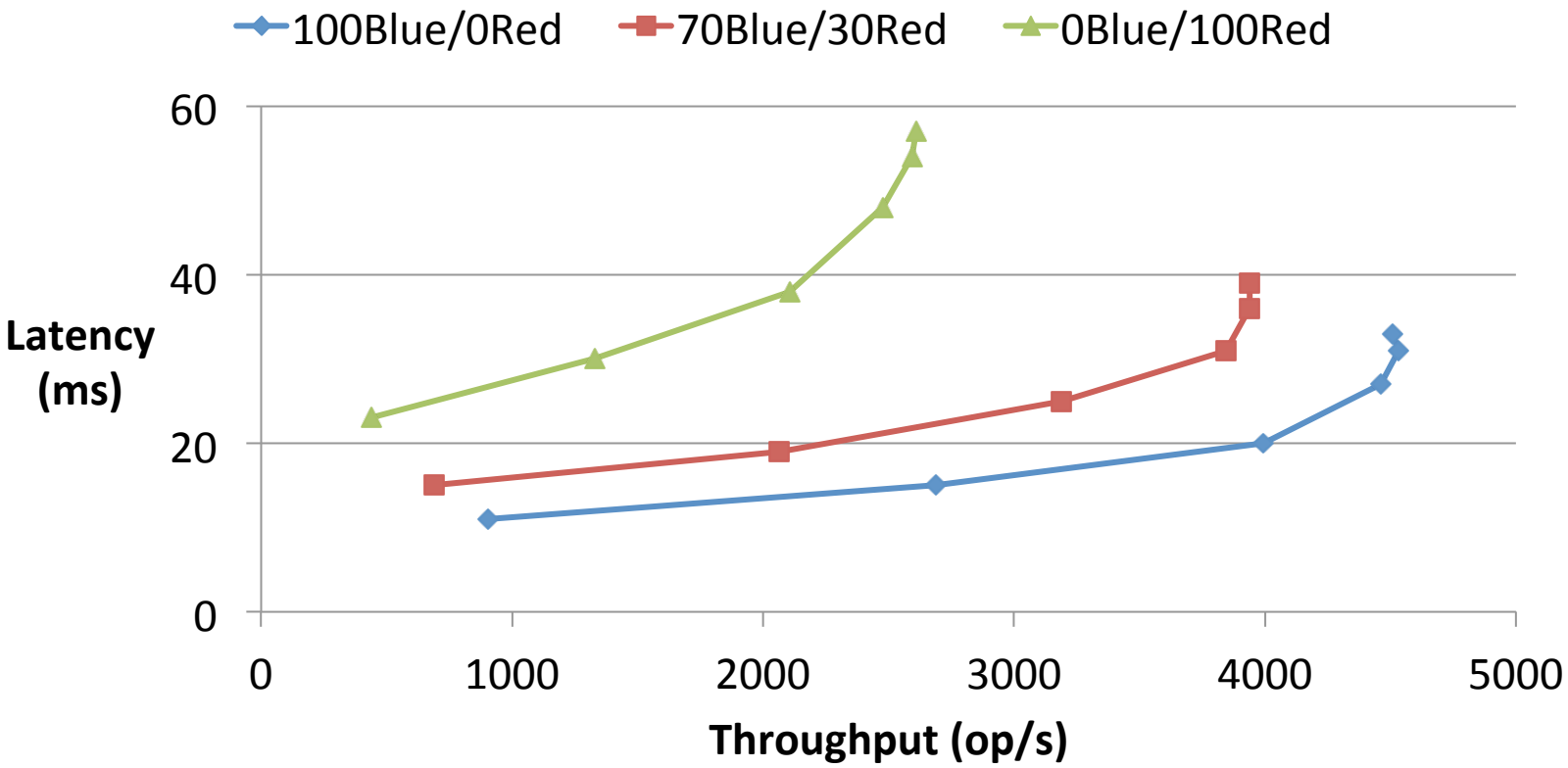
Making Geo-Replicated Systems Fast as Possible, Consistent when Necessary

THANK YOU!

Shadow operations

- Writing shadow operations is not challenging.
- Strategies for making shadow operations commute:
  - Pass non-deterministic outputs as parameters to shadow transaction
  - Convert operations to commutative ones (e.g., +,-)
  - Use last-writer-wins strategy
- Labeling shadow operations is not difficult.
  - One could conservatively label operations as Red.

Impact of Red ratio



Microbenchmark