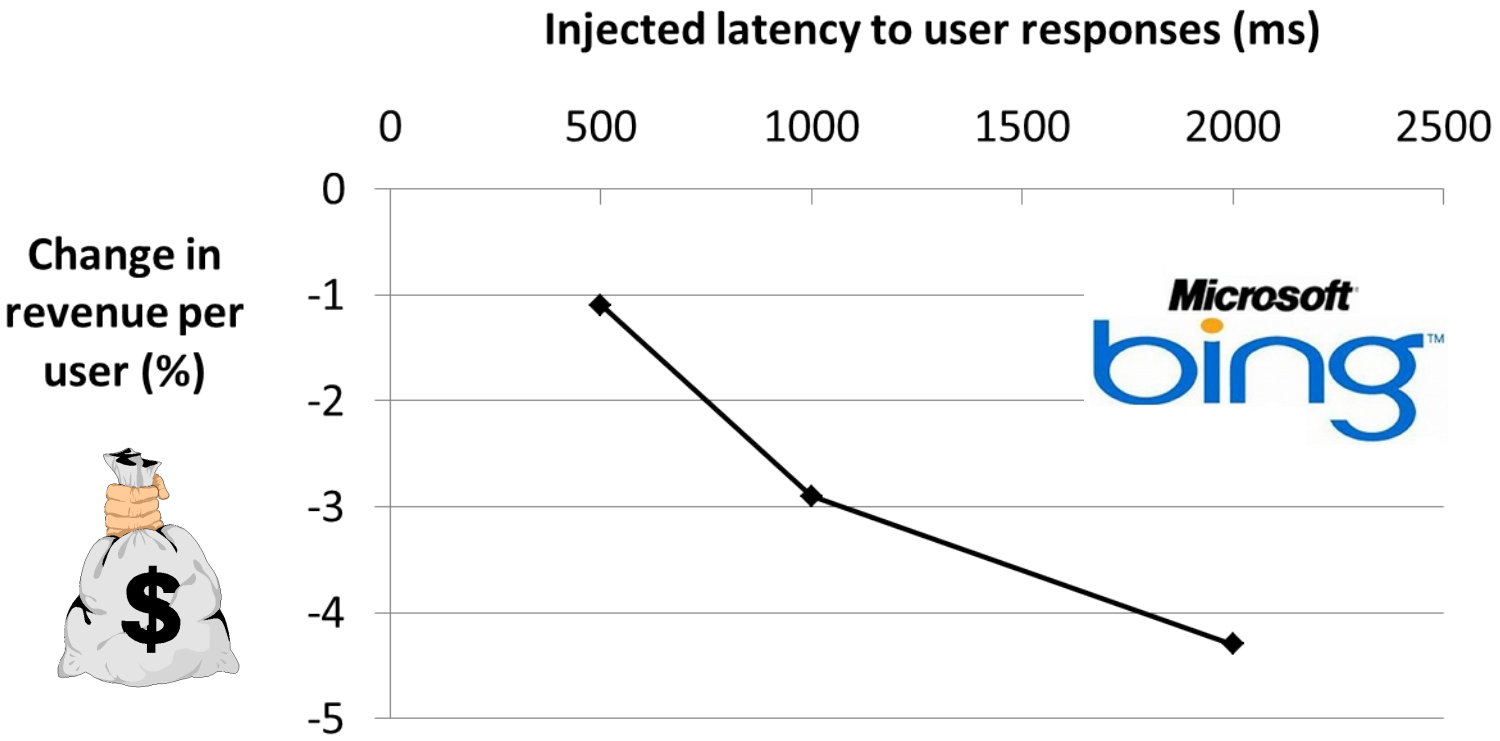
Making Geo-Replicated Systems Fast as Possible Consistent when Necessary

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

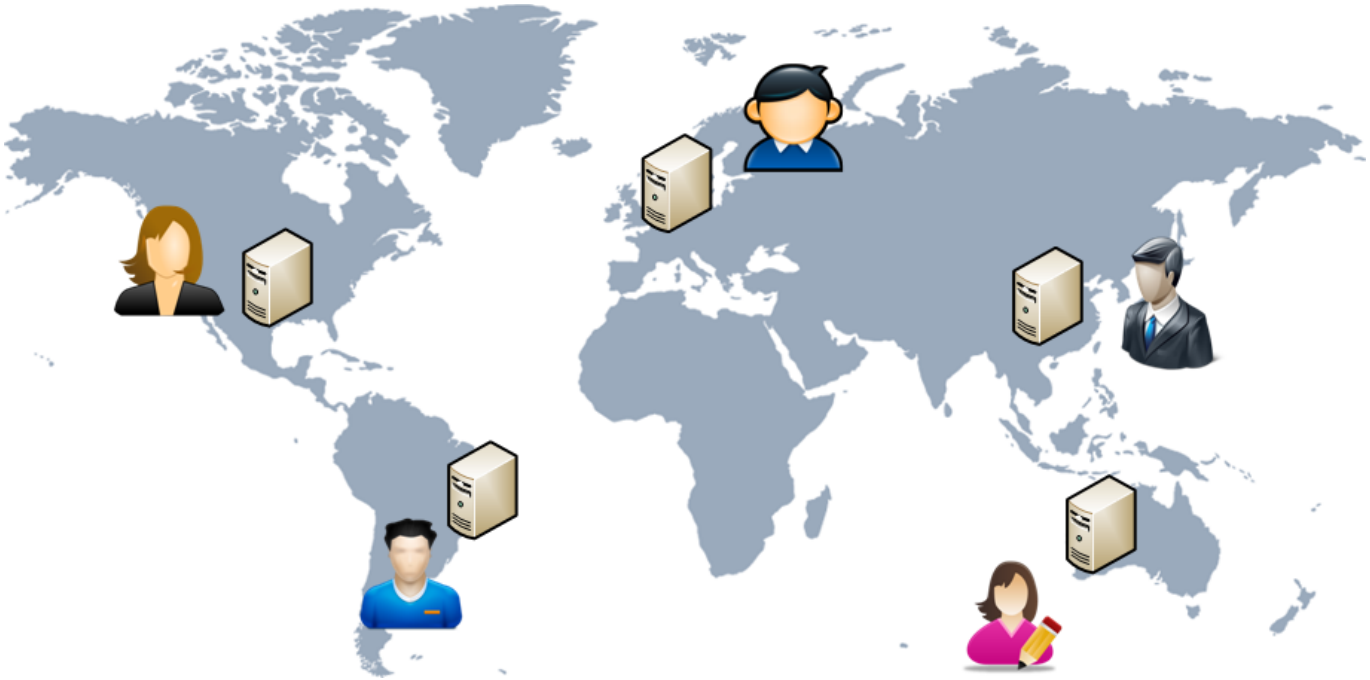
Higher latency => Less money



[source: E. Schurman and J. Brutlag, "Performance Related Changes and their User Impact". Talk at Velocity '09]

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?

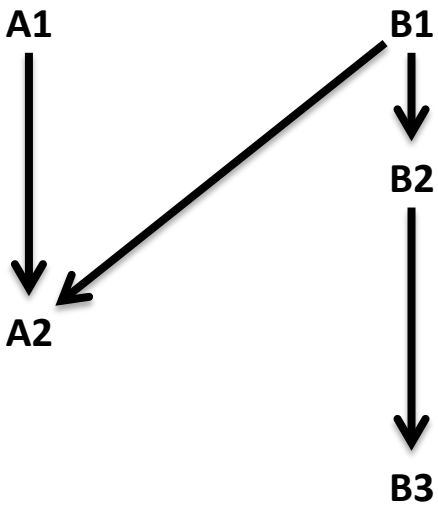
Max Planck Institute for Software Systems

Outline

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

Balance strong/eventual consistency

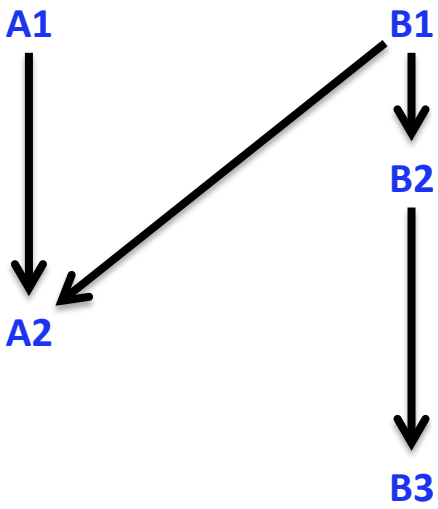
Strong consistency **Eventual consistency**



Balance strong/eventual consistency

Strong consistency

Eventual consistency

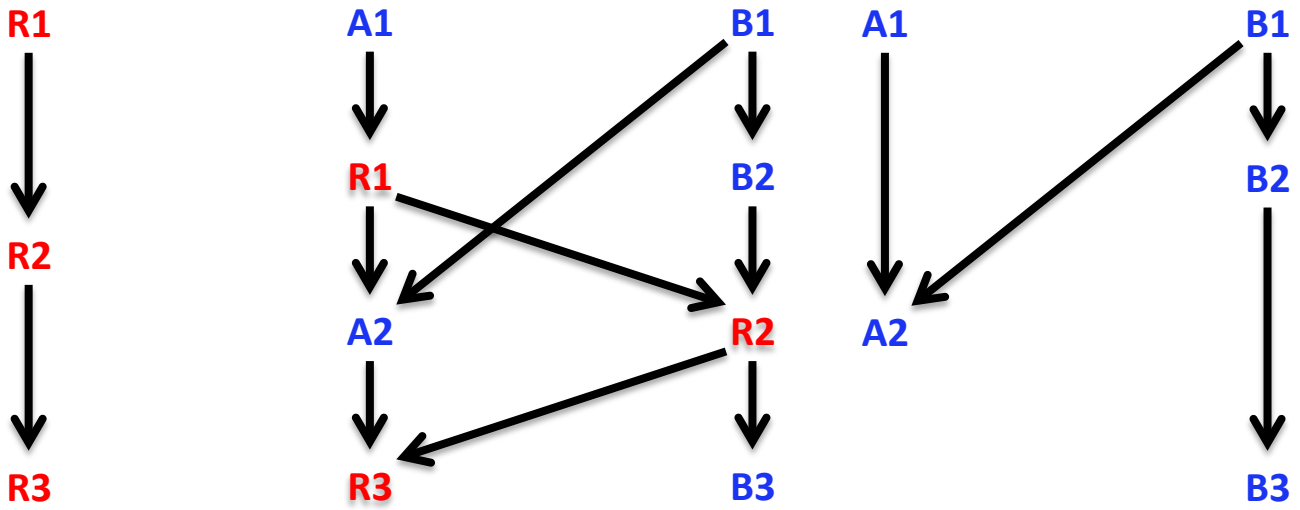


Max Planck Institute for Software Systems

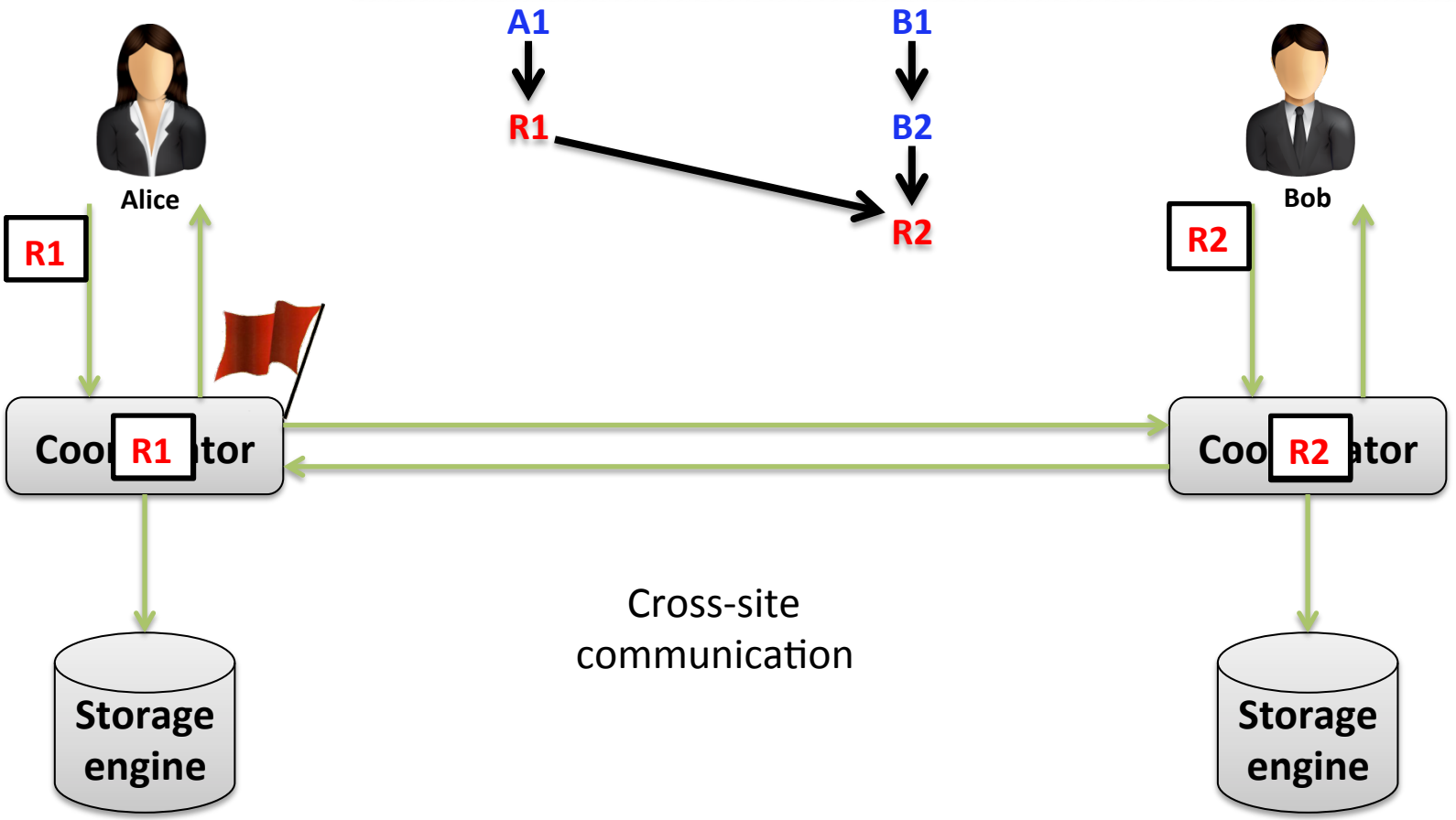
Balance strong/eventual consistency

Strong consistency **RedBlue** **Eventual consistency**

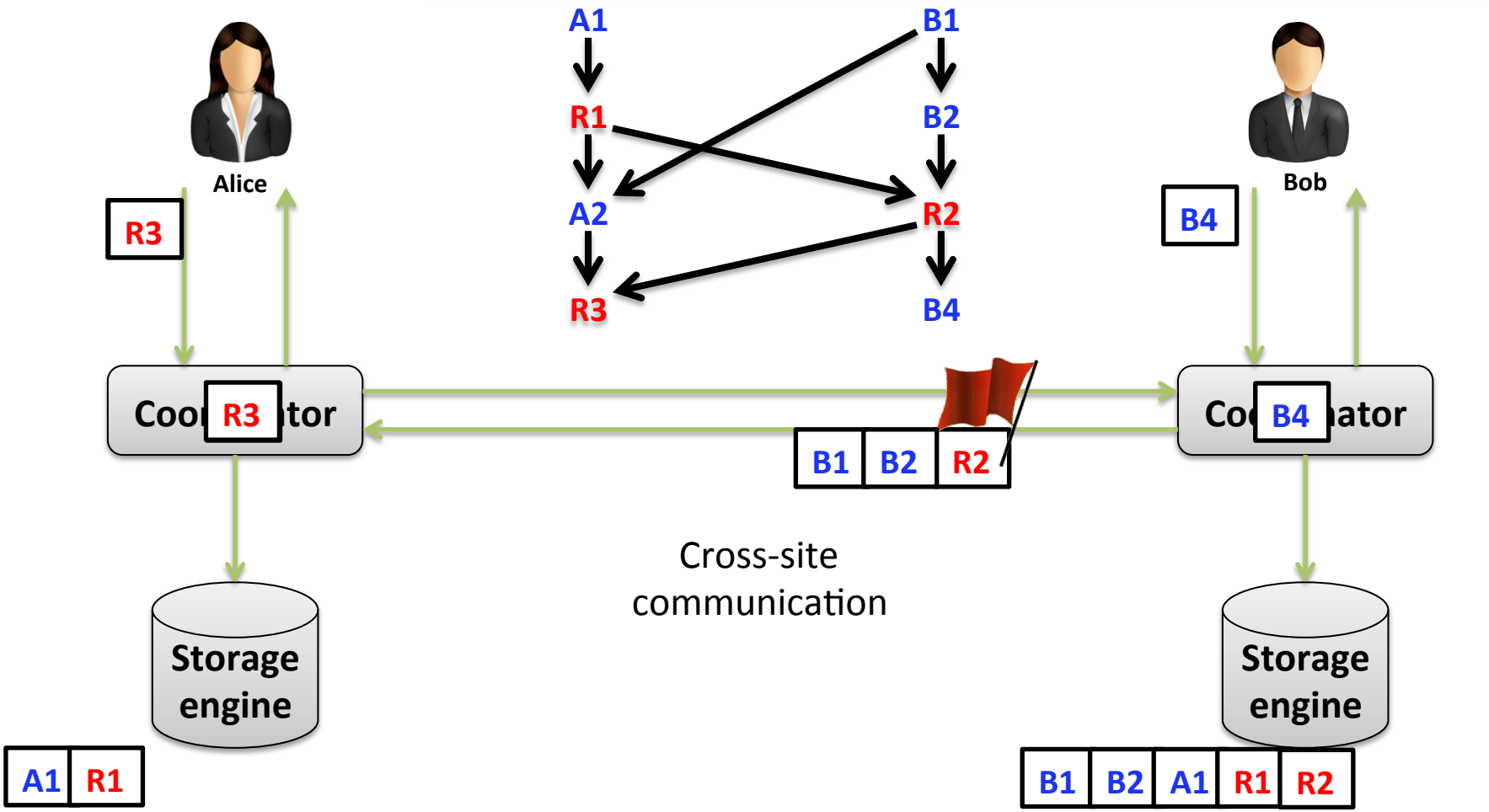
- Low latency of eventual consistency when possible
- Coordination for strong consistency only when necessary



Gemini coordination system



Gemini coordination system



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.

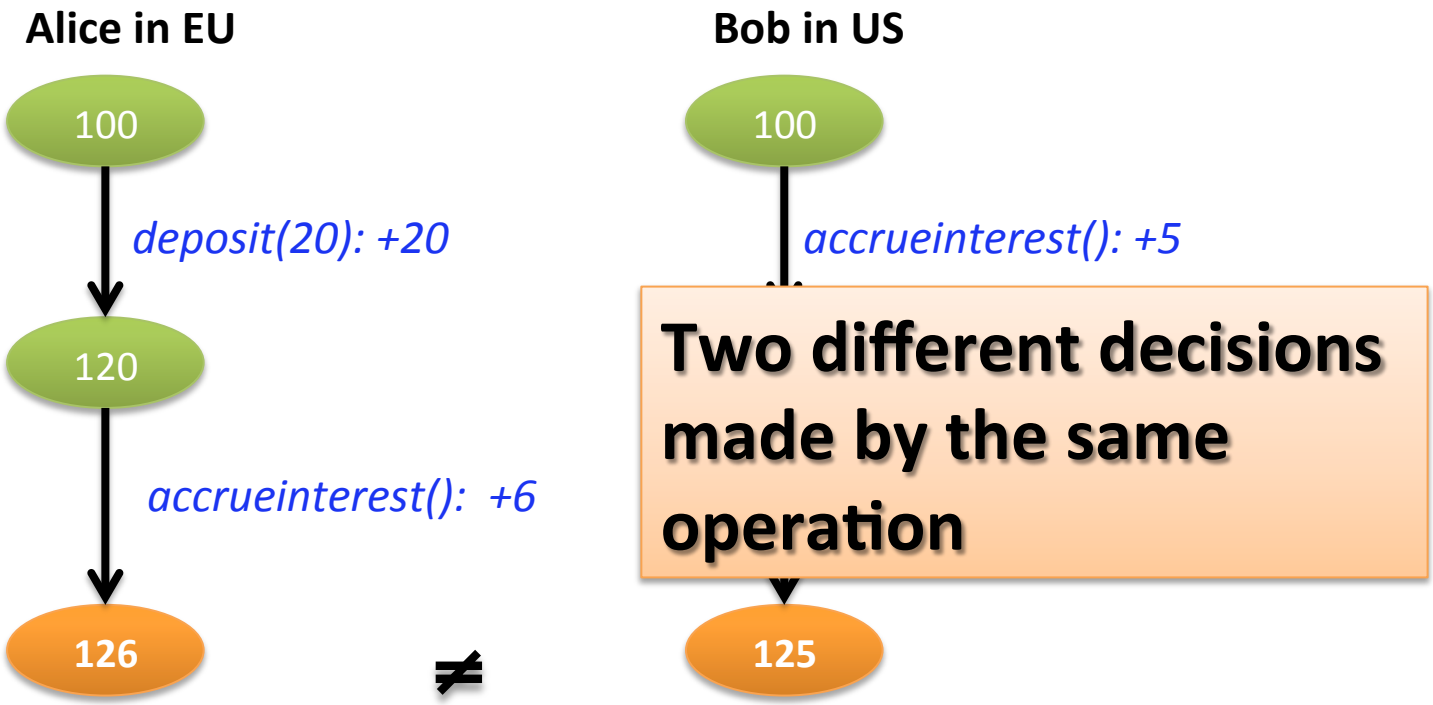

```
float balance, interest;

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

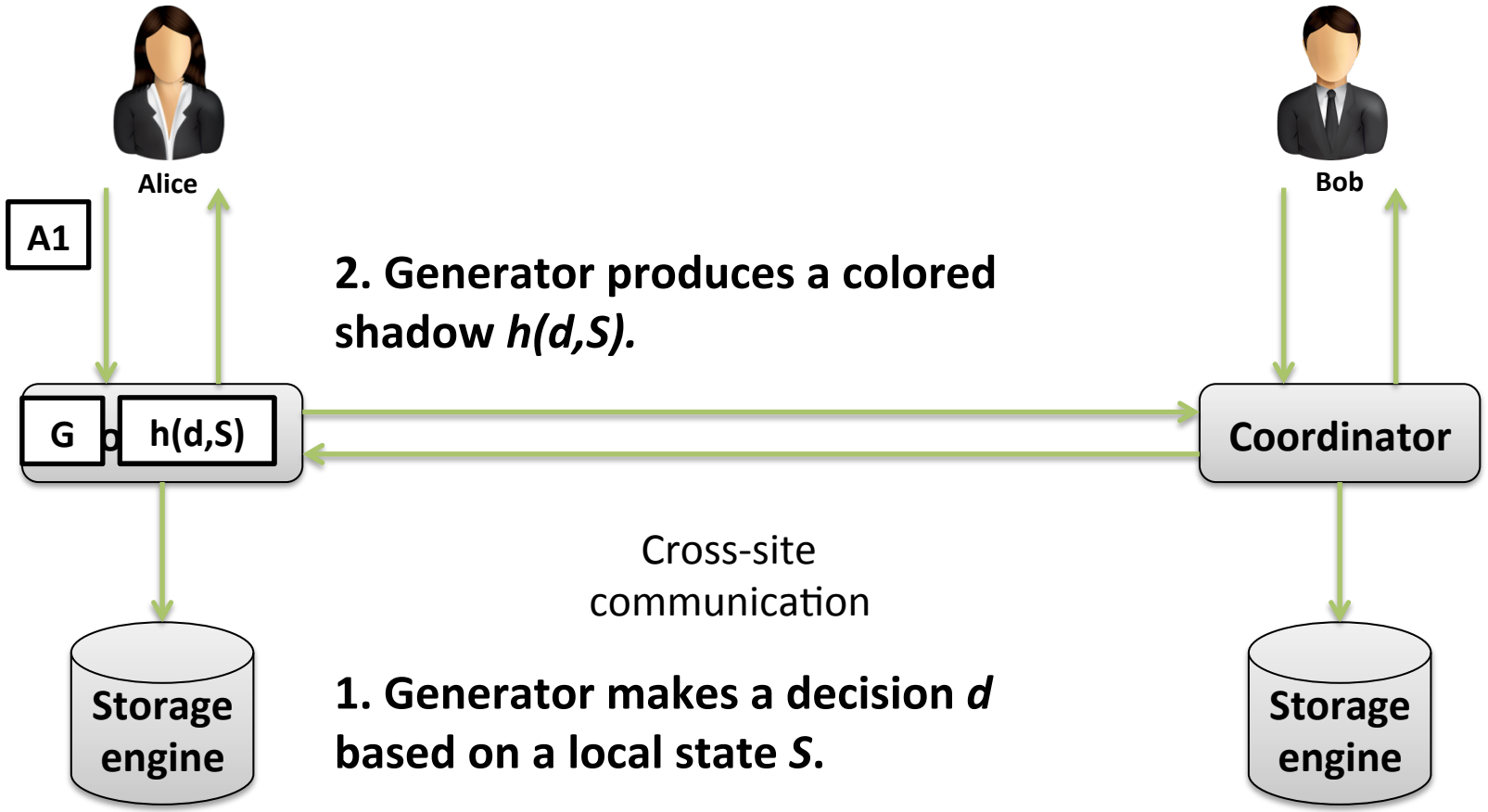
Initial: *balance = 100, interest = 0.05*



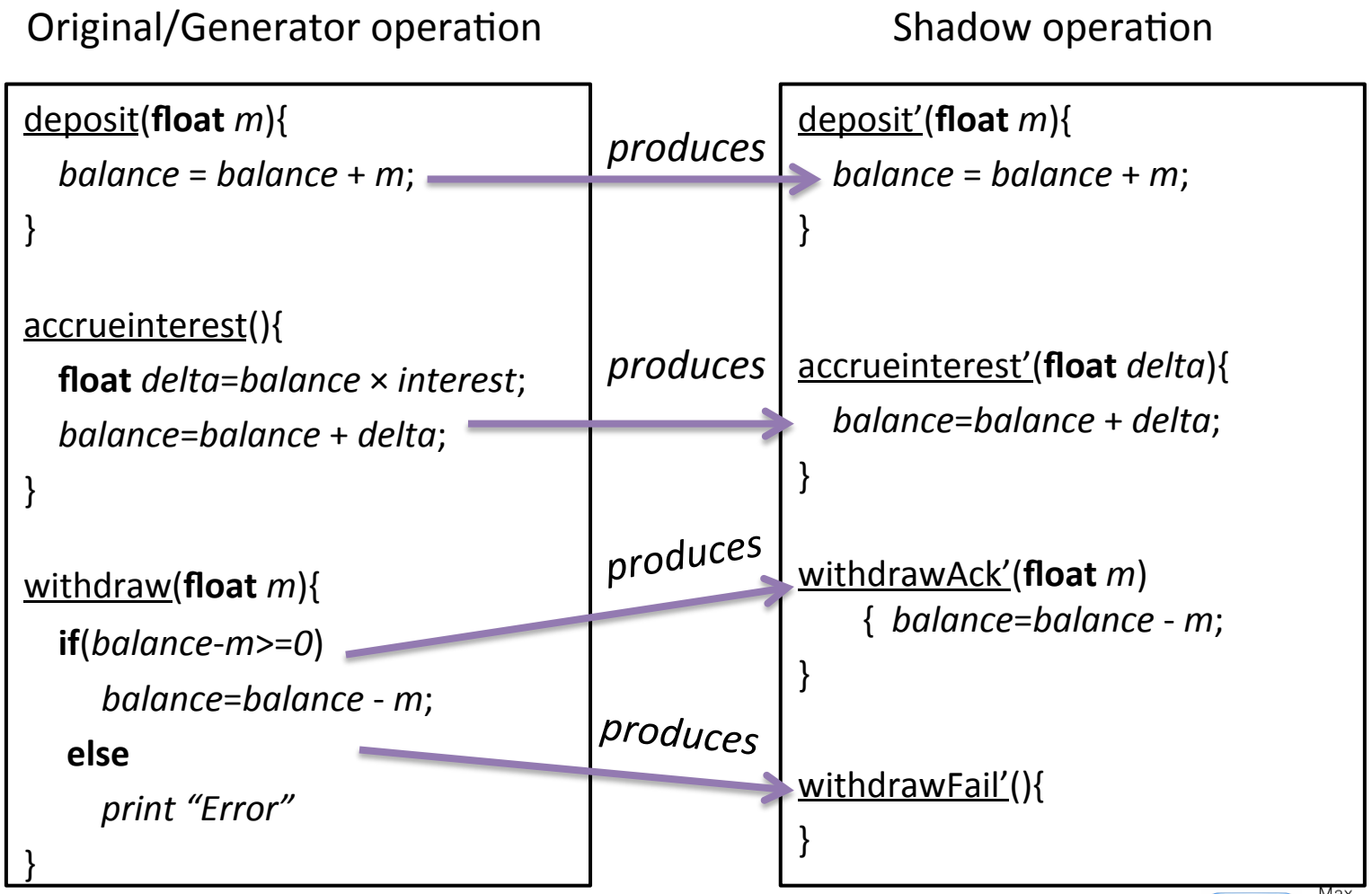
Generator/Shadow operation

- 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*

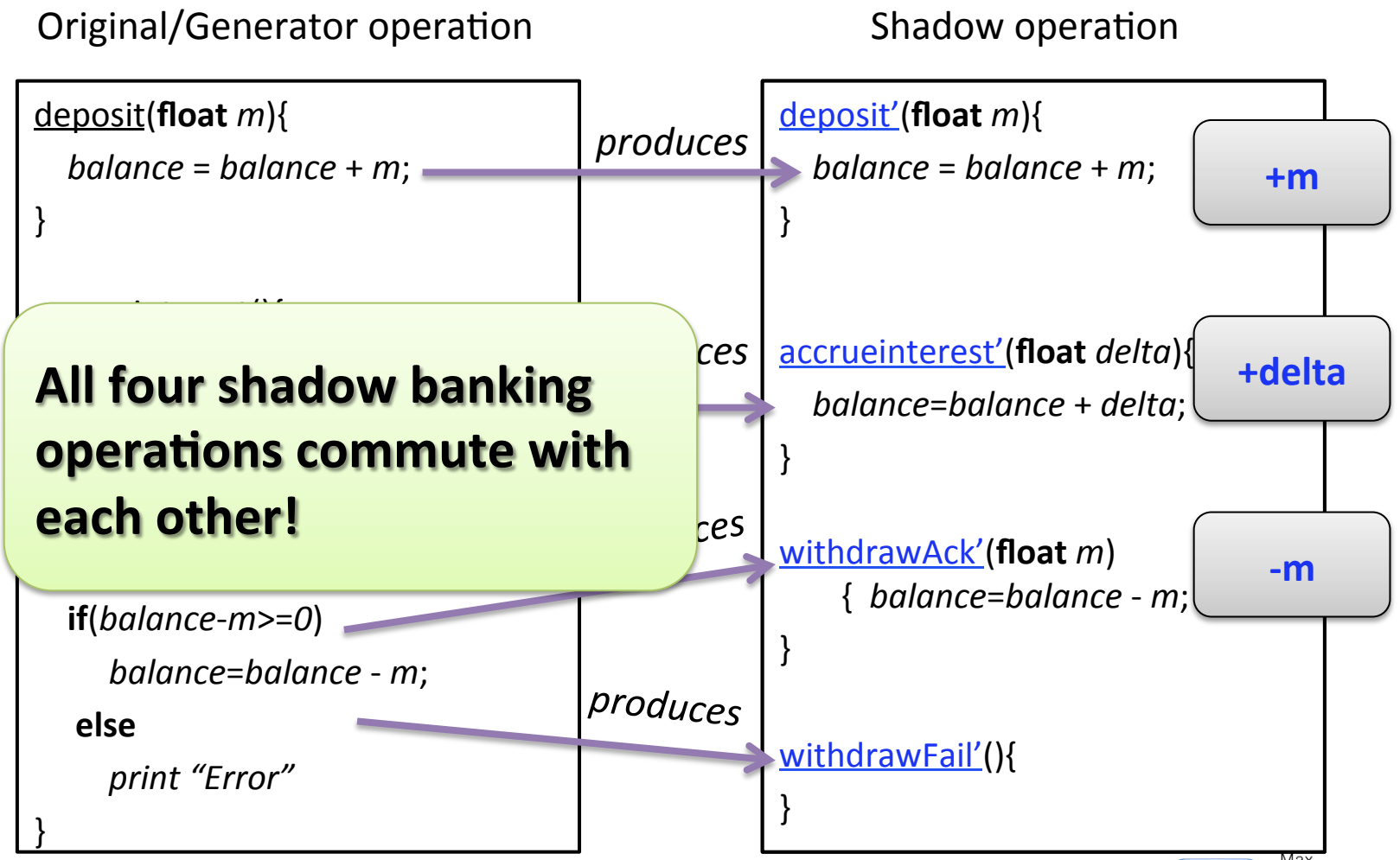
Generate once, shadow everywhere



Bank generator/shadow operations

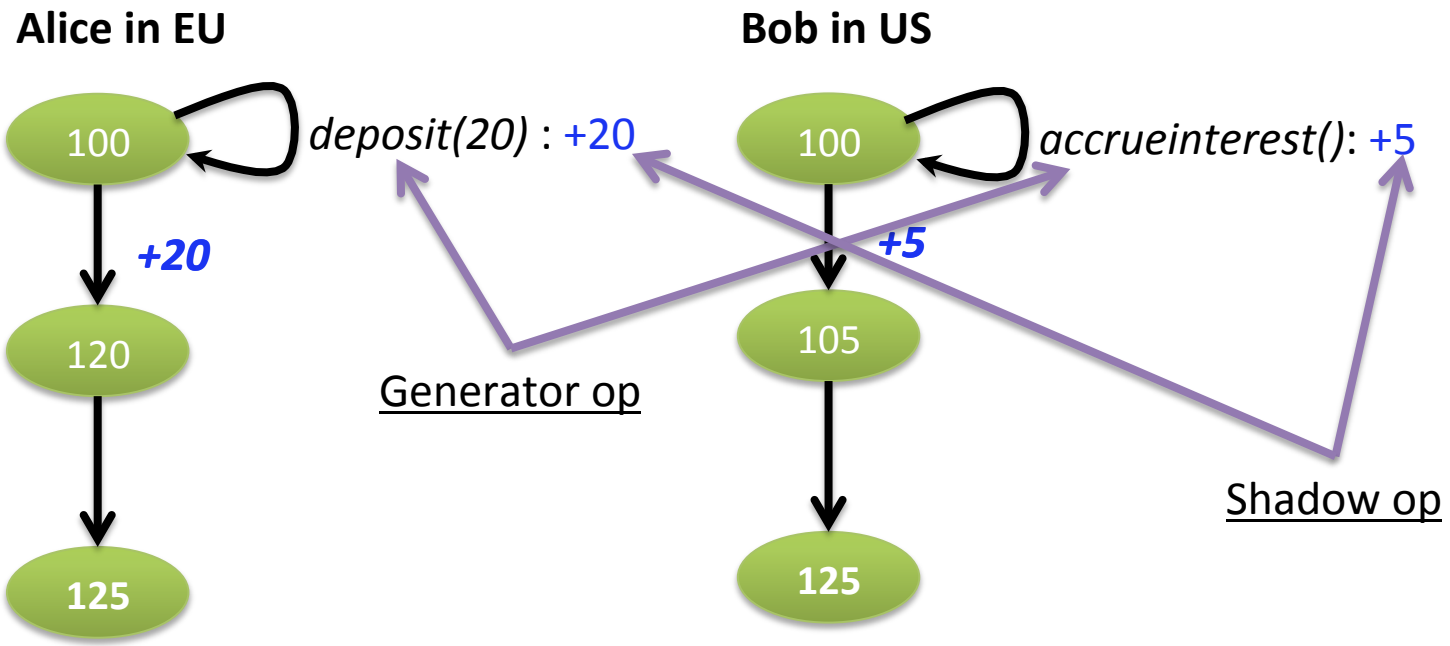


Bank generator/shadow operations



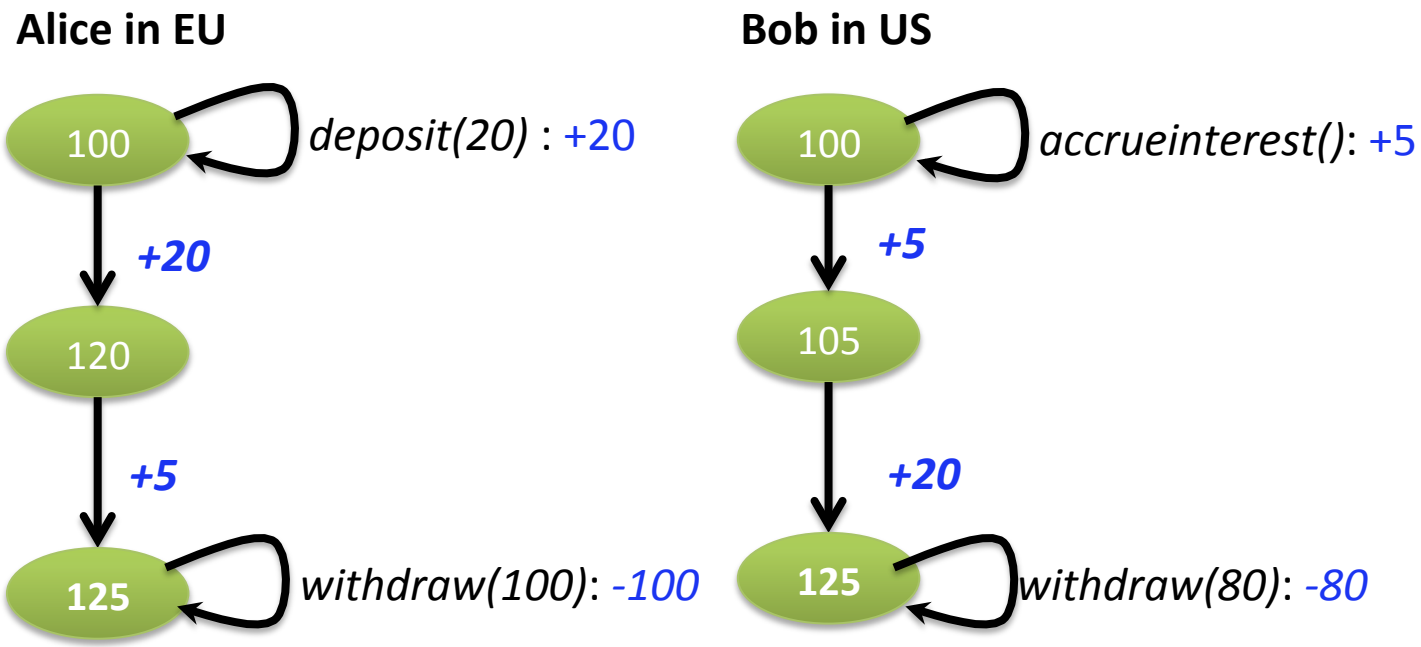
Fast and consistent bank

Initial: *balance = 100, interest = 0.05*



Not so fast …

Initial: *balance = 100, interest = 0.05*



Not so fast …

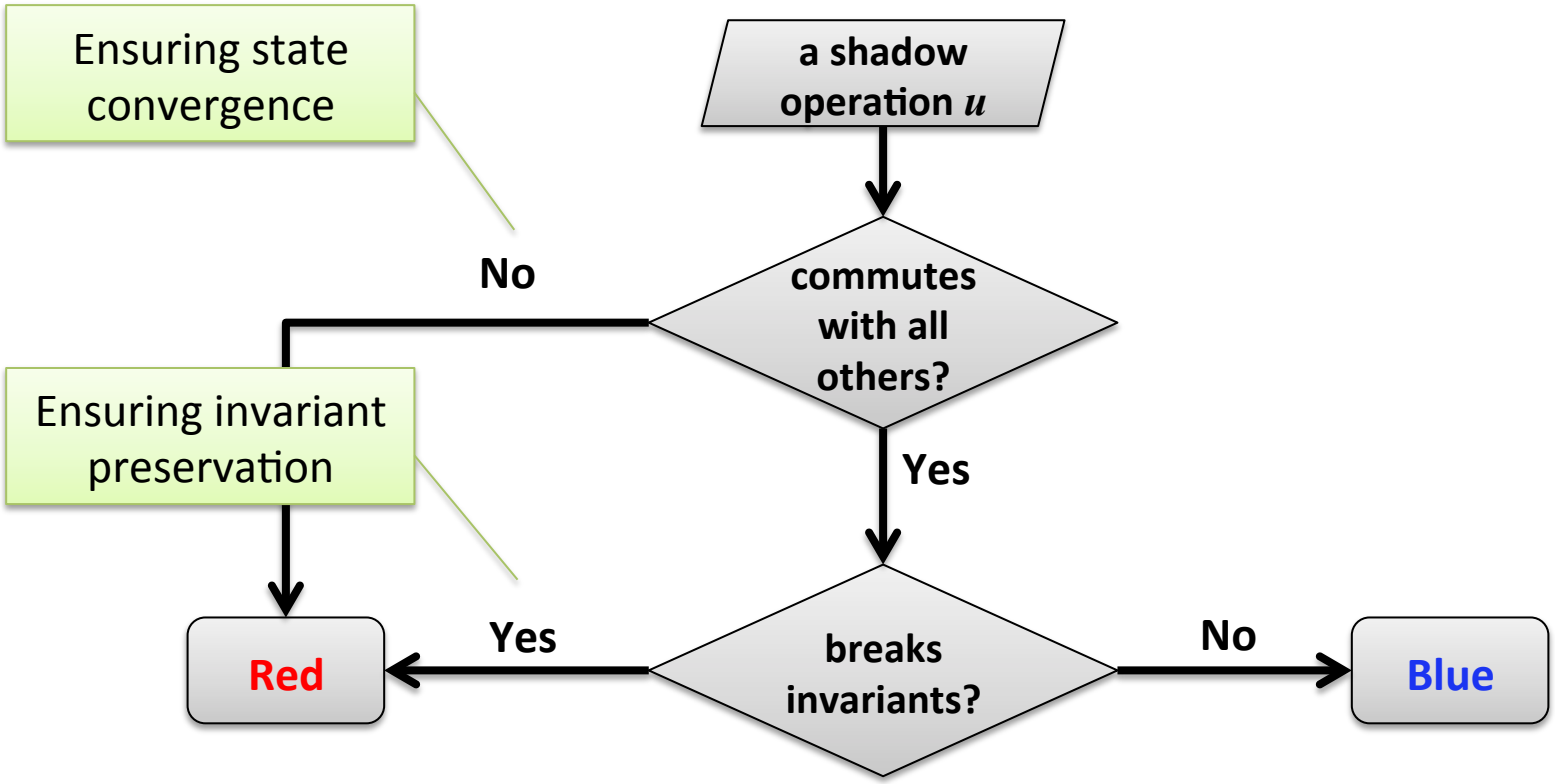
- **Problem**: Different execution orders lead to a negative balance.
- **Cause**: Blue operations that potentially break invariants execute without coordination.
- **Implication**: We must label successful withdrawal (*withdrawAck'*) as Red.

): *-80*

–55

–55

Which must be Red or can be Blue?



Key ideas so far

- RedBlue consistency combines strong and eventual consistency into a single system.
- The decomposition of generator/shadow operations expands the space of possible Blue operations.
- A simple rule for labeling is provably state convergent and invariant preserving.

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.

Max Planck Institute for Software Systems

Questions

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

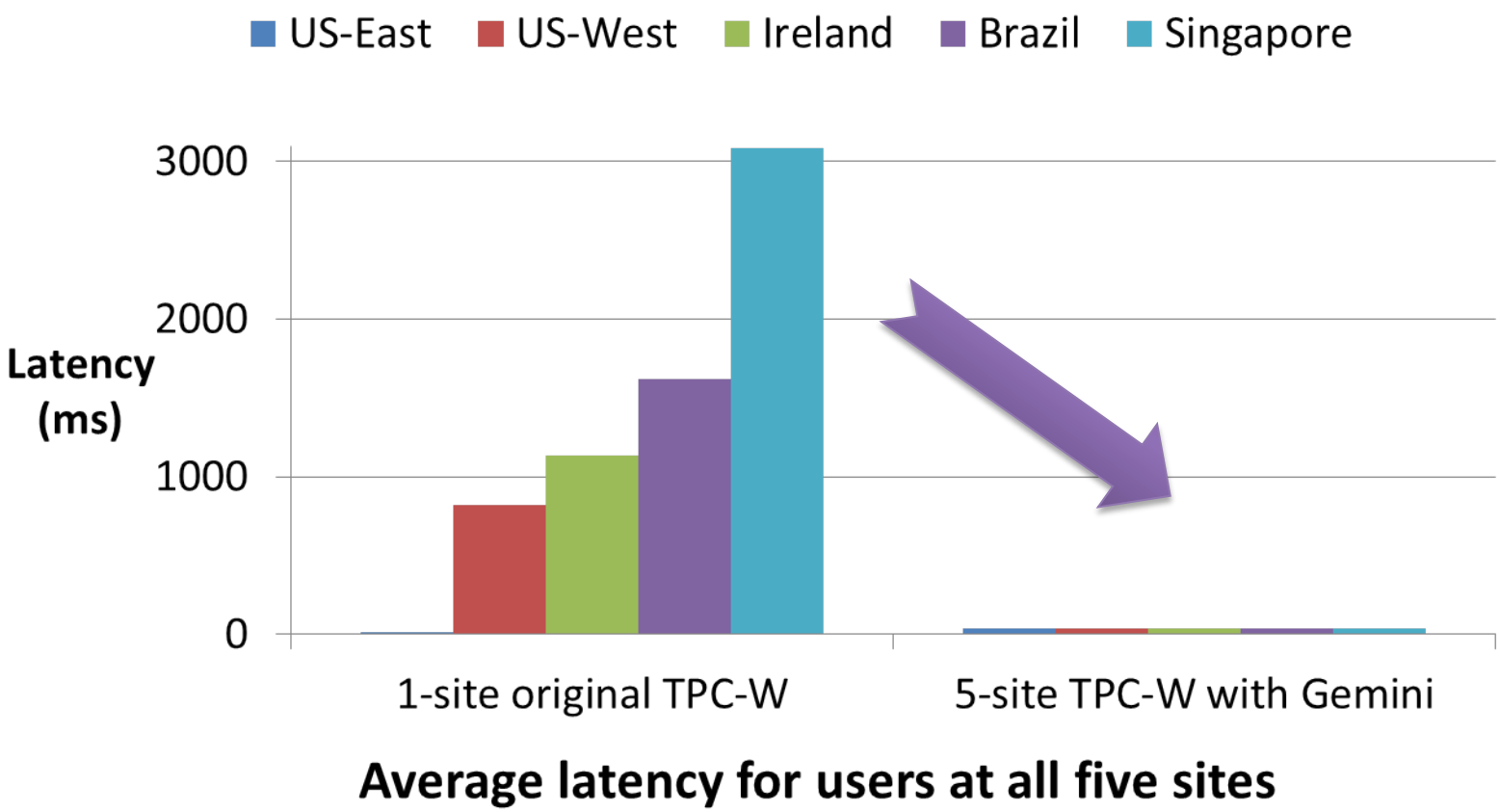
Experimental setup

- 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

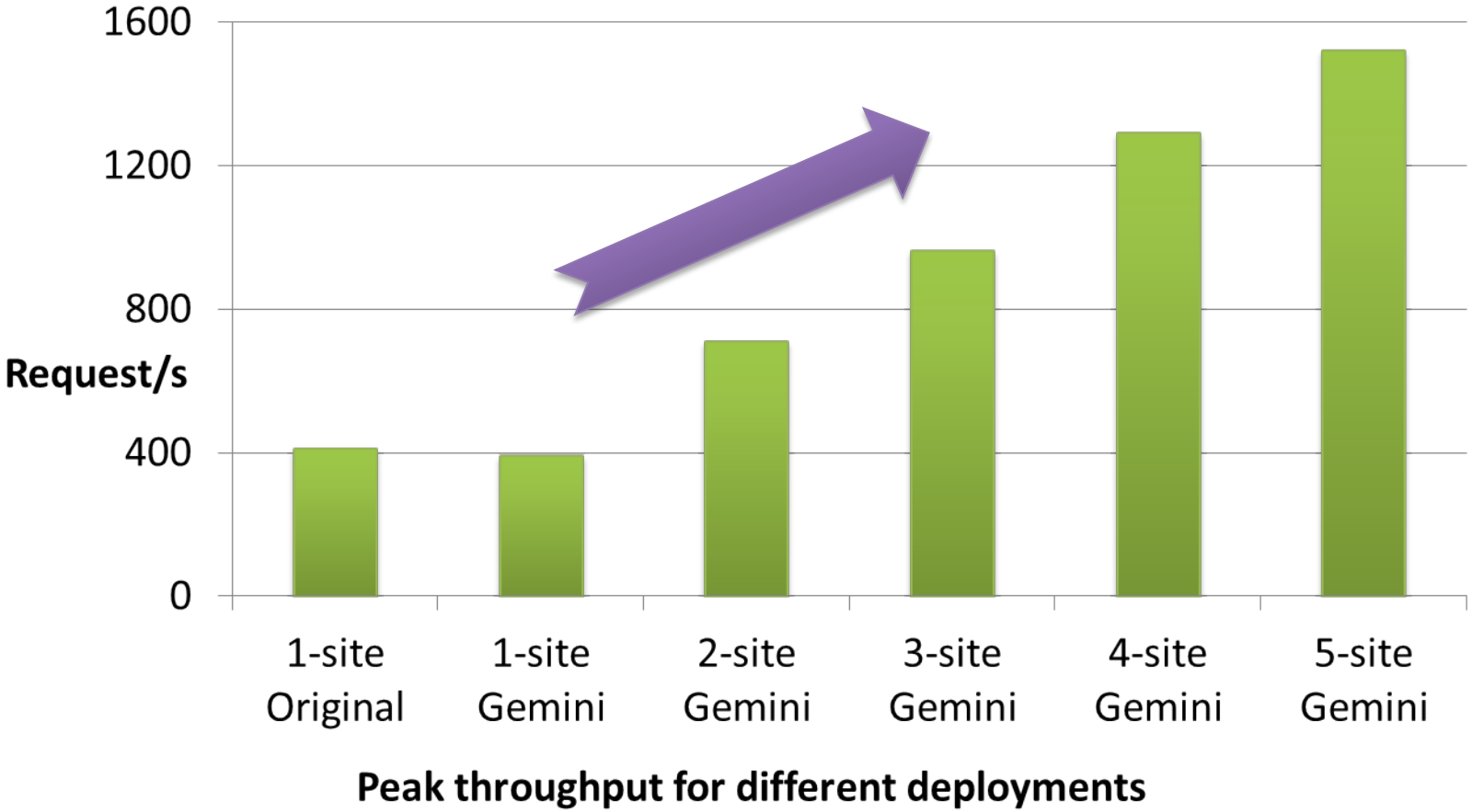
Does RedBlue consistency improve user-observed latency?



Average latency for users at all five sites

Max Planck Institute for Software Systems

Does throughput scale with the number of sites?



Peak throughput for different deployments

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!