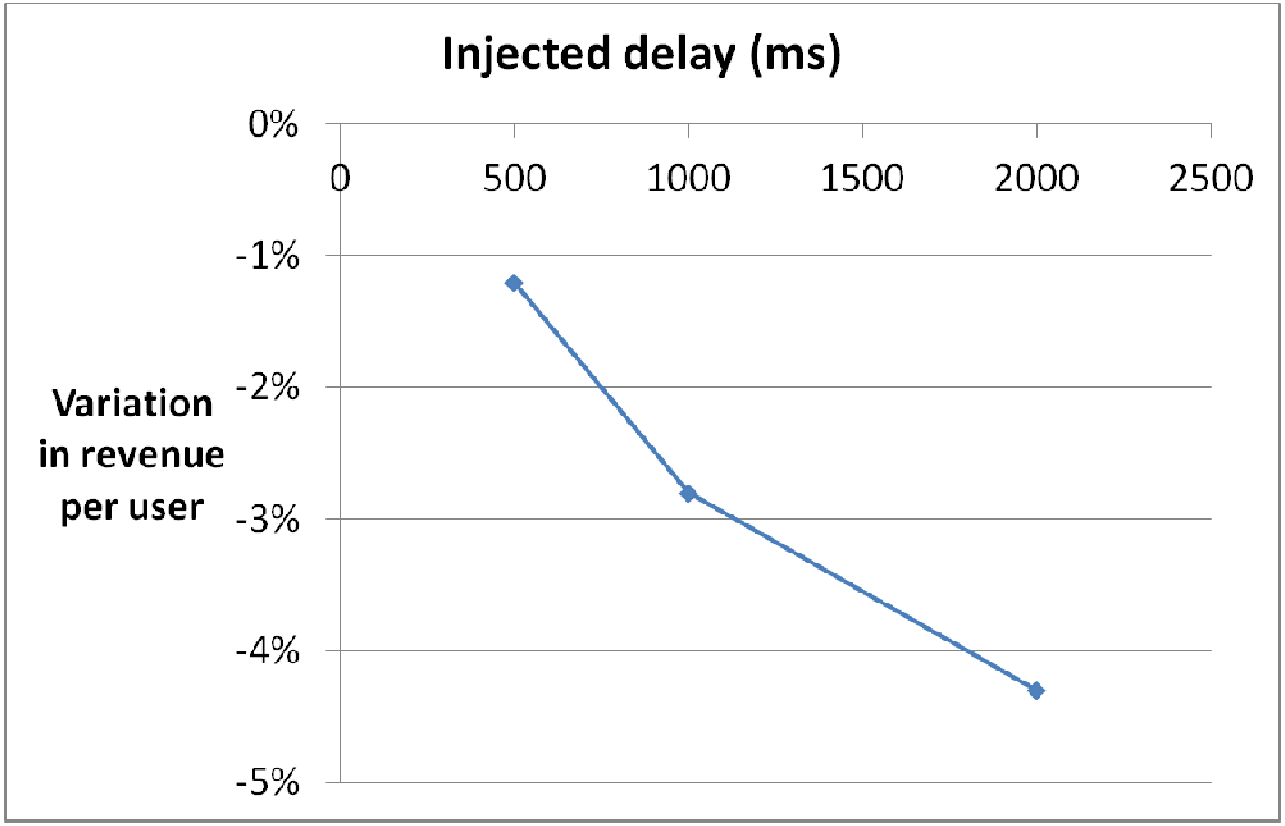
Making Geo-Replicated Systems Fast when Possible, Consistent if Necessary

Rodrigo Rodrigues
NOVA University of Lisbon

Joint work with Allen Clement, Johannes Gehrke, Cheng Li, Daniel Porto and Nuno Preguiça
Max Planck Institute for Software Systems, Cornell, and NOVA U.

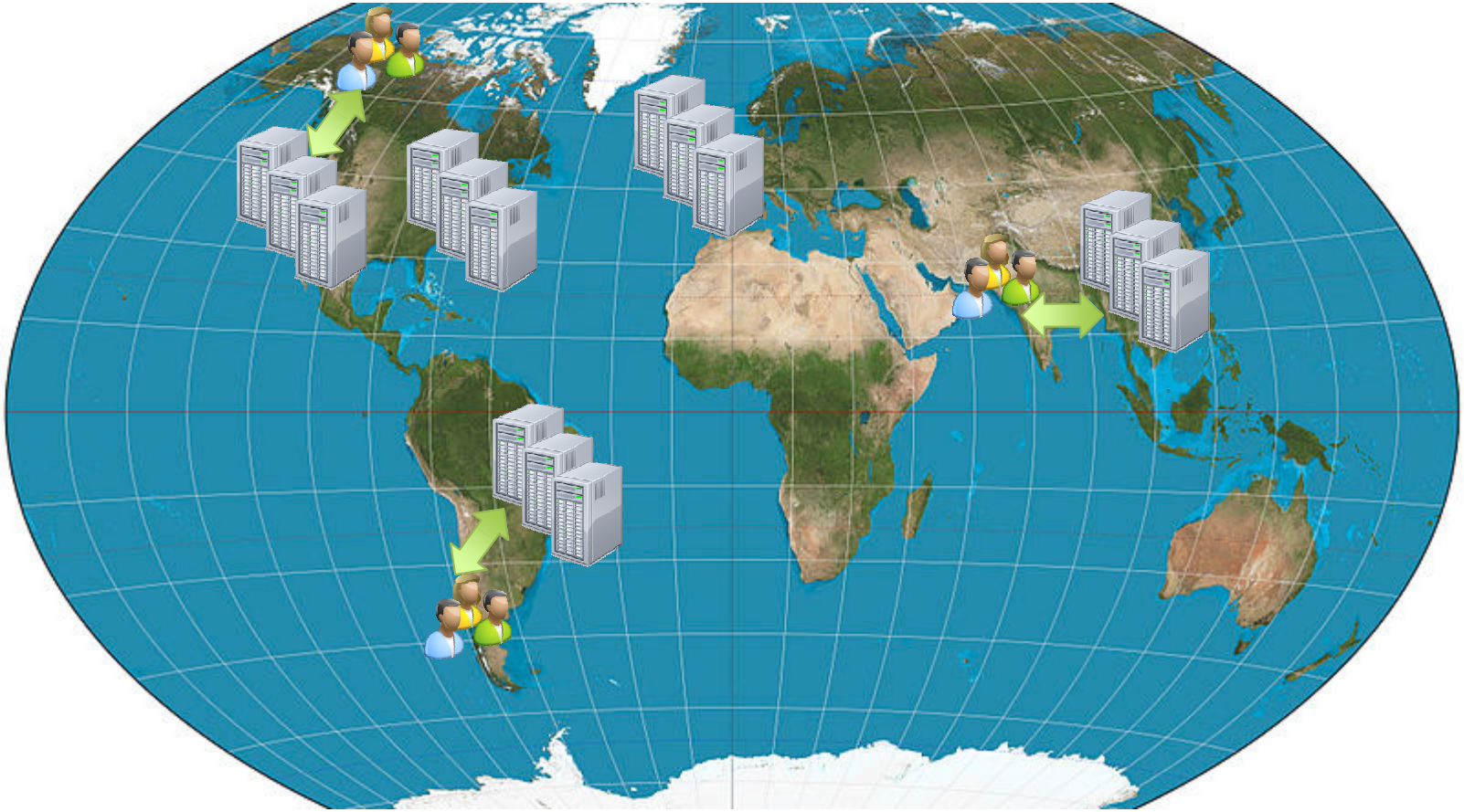
IFIP Workshop – January 18, 2013

Higher latency ⇒ Lower revenue



Bing: 2-sec slowdown reduced revenue per user by 4.3%

Consequence: Geo-replication



Flip-side of geo-replication

- Geographically disperse replicas are expensive to synchronize
- Leads to existence of various different levels of geo-replication: the *geo-replication hierarchy*

The geo-replication hierarchy

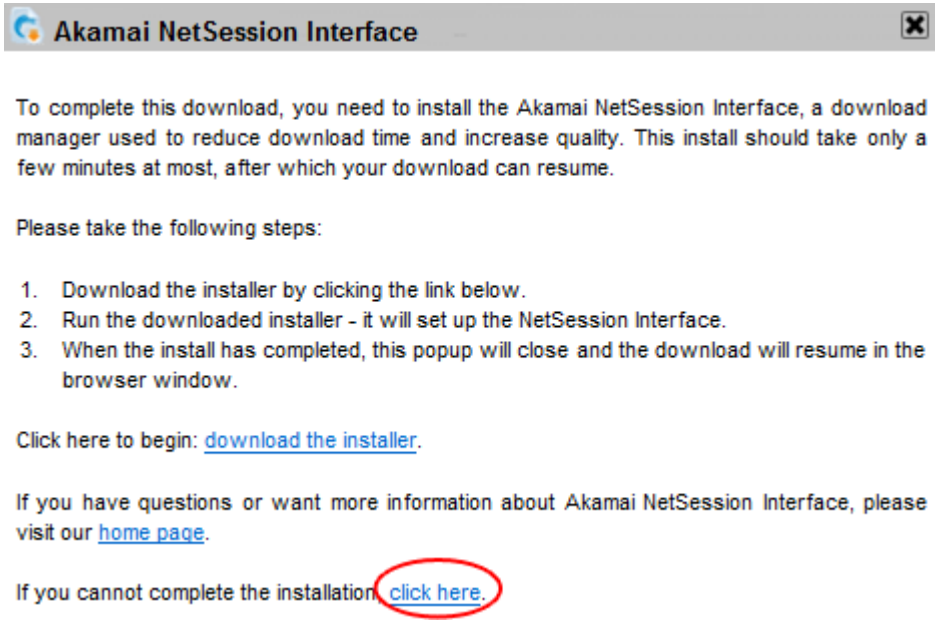
- First level: centralized server
- Second level: replicas at distant data centers
- The two levels can co-exist:
  - Updates are funneled through primary
  - Secondary replicas process read requests

The geo-replication hierarchy (cont.)

- Third level: replicas deployed in conventional CDN infrastructures
- Examples:
  - Akamai (static content): 105,000 servers in 78 countries
  - Google (mostly youtube)

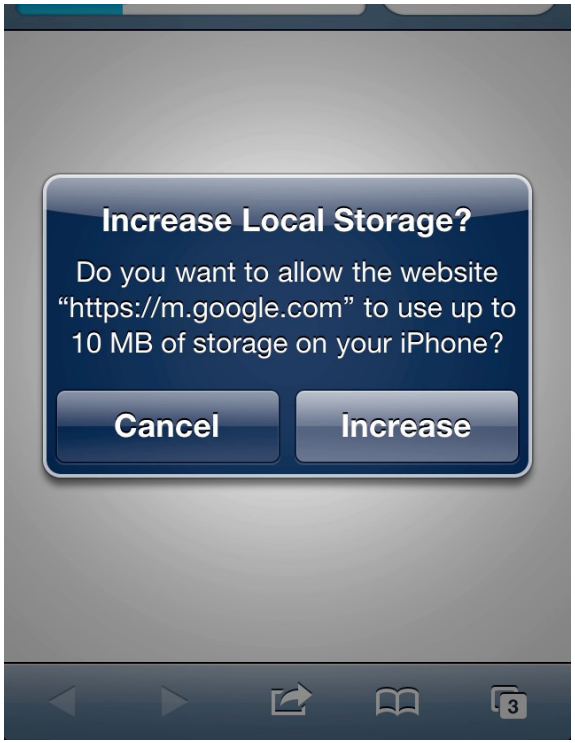
The geo-replication hierarchy (cont.)

- Fourth level: peer-to-peer / hybrid CDNs
- Lower cost of maintaining CDN by leveraging voluntary contributions of clients
- Example: Akamai netsession

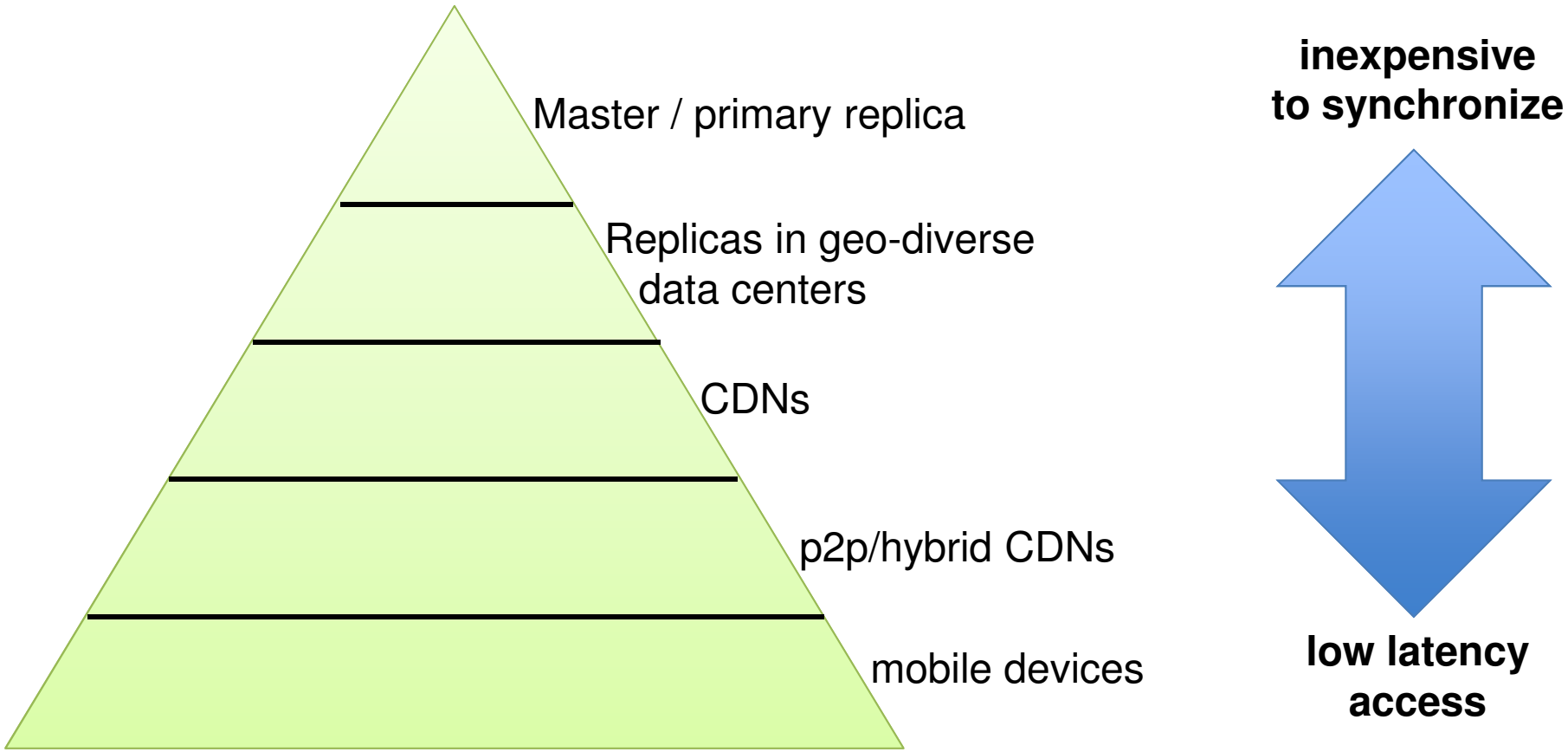


The geo-replication hierarchy (cont.)

- Fifth level: replicas on mobile devices
- Local replication overcomes the high latency / low throughput / weak connectivity
- Example:



The geo-replication hierarchy



Challenge

- Finding a set of principles for designing distributed systems that are aware of the geo-replication hierarchy
  - Much like our operating systems must be aware of the storage hierarchy
- This talk: more narrow aspect of the problem
  - Managing replicas in separate data centers

Current practice: examples and limitations

- Facebook + PNUTS: single master, read-only mirror replicas
  - Works well when there is a single updater
  - Not the case, e.g., in social networking services
- Amazon: Eventual consistency
  - Assumes a seamless merge strategy and may allow undesirable behaviors
  - Folklore: Eventual consistency is no consistency

An observation and a challenge

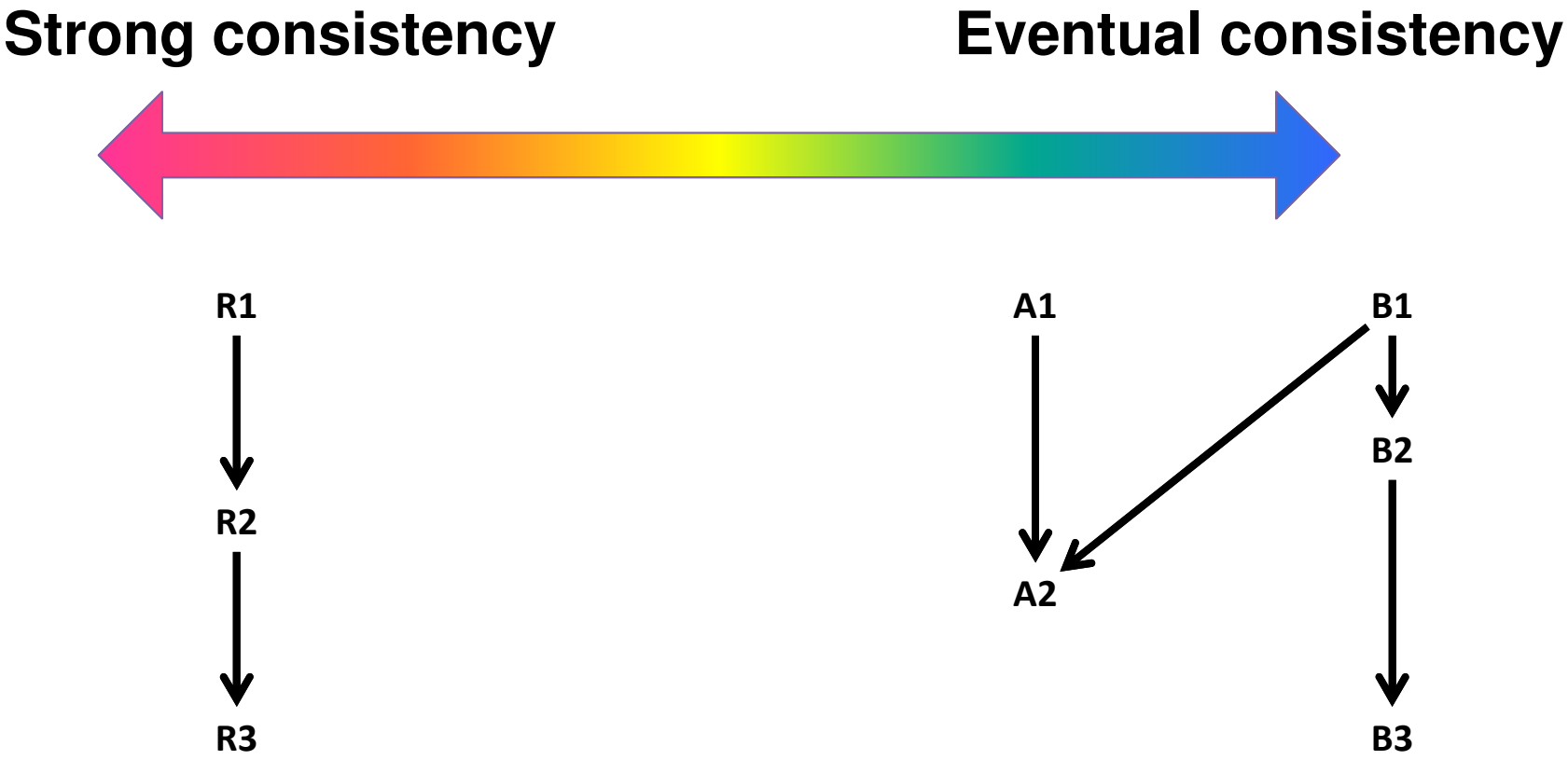
- Eventual consistency works most of the time, but need some strongly consistent operations
- Must let both weakly and strongly consistent operations co-exist [LazyReplication:PODC90,Walter:SOSP11]
  - But which level of consistency to use for an operation?

Need to find principled ways to build systems that are **fast as possible, consistent when needed**

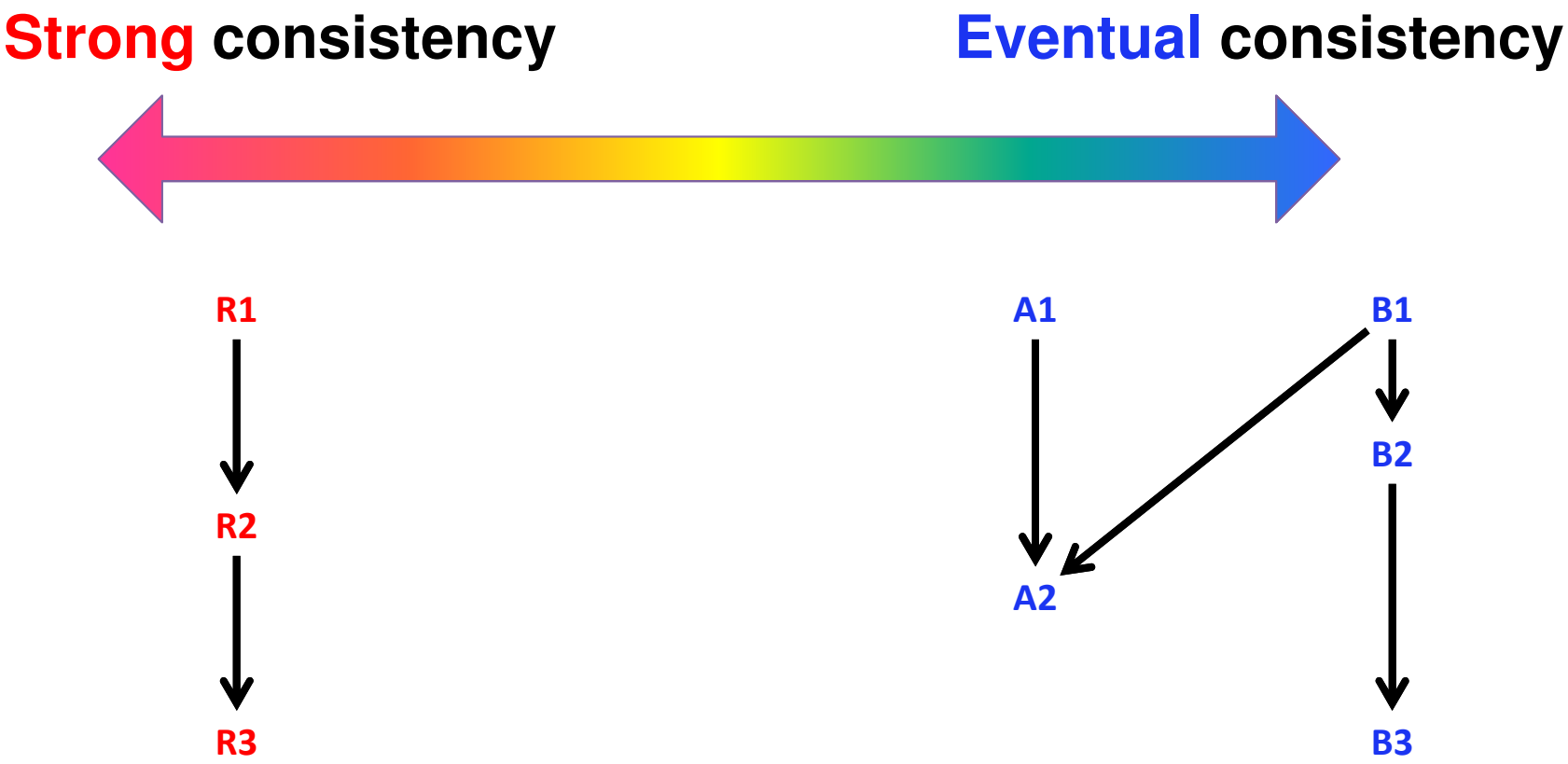
Outline

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

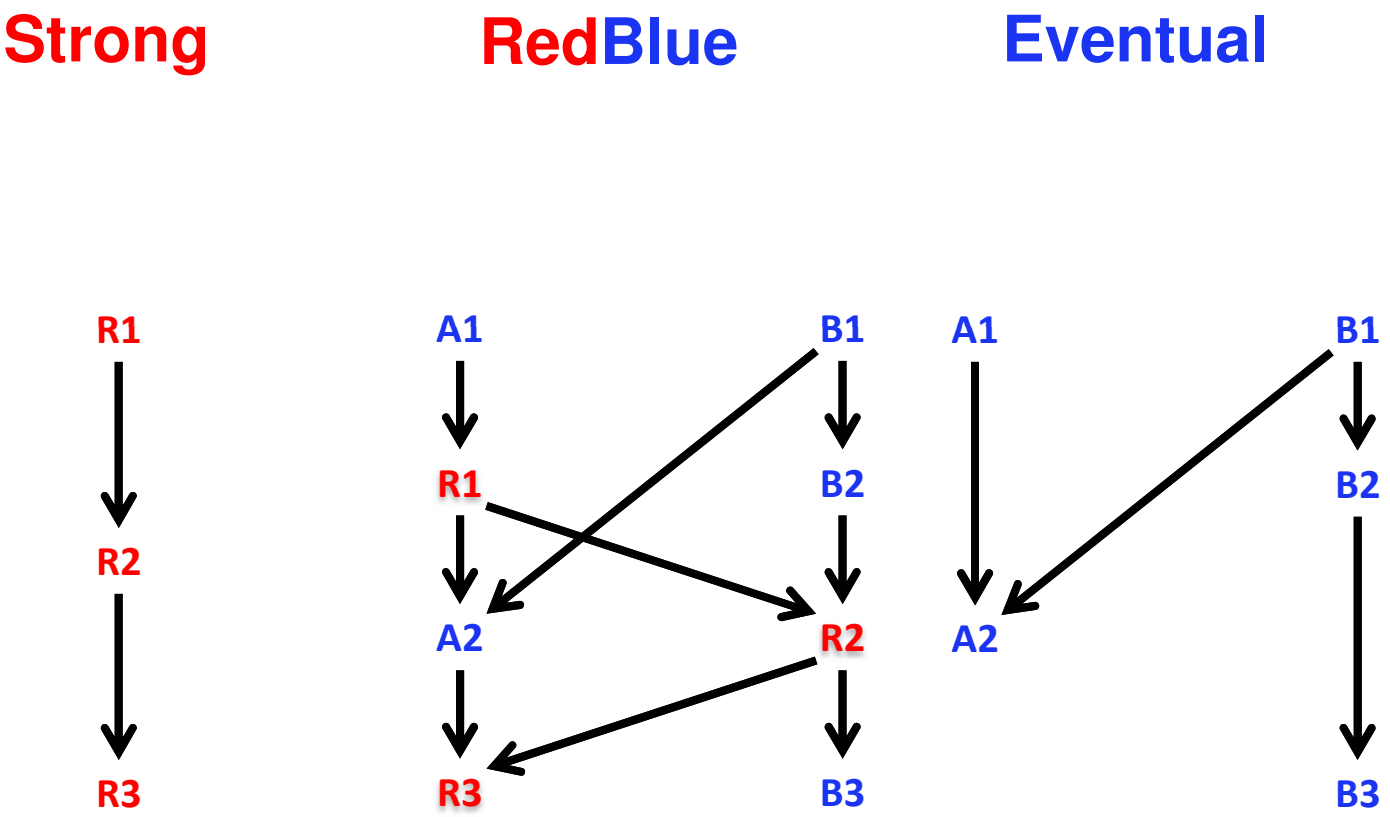
Balance strong/eventual consistency



Balance strong/eventual consistency

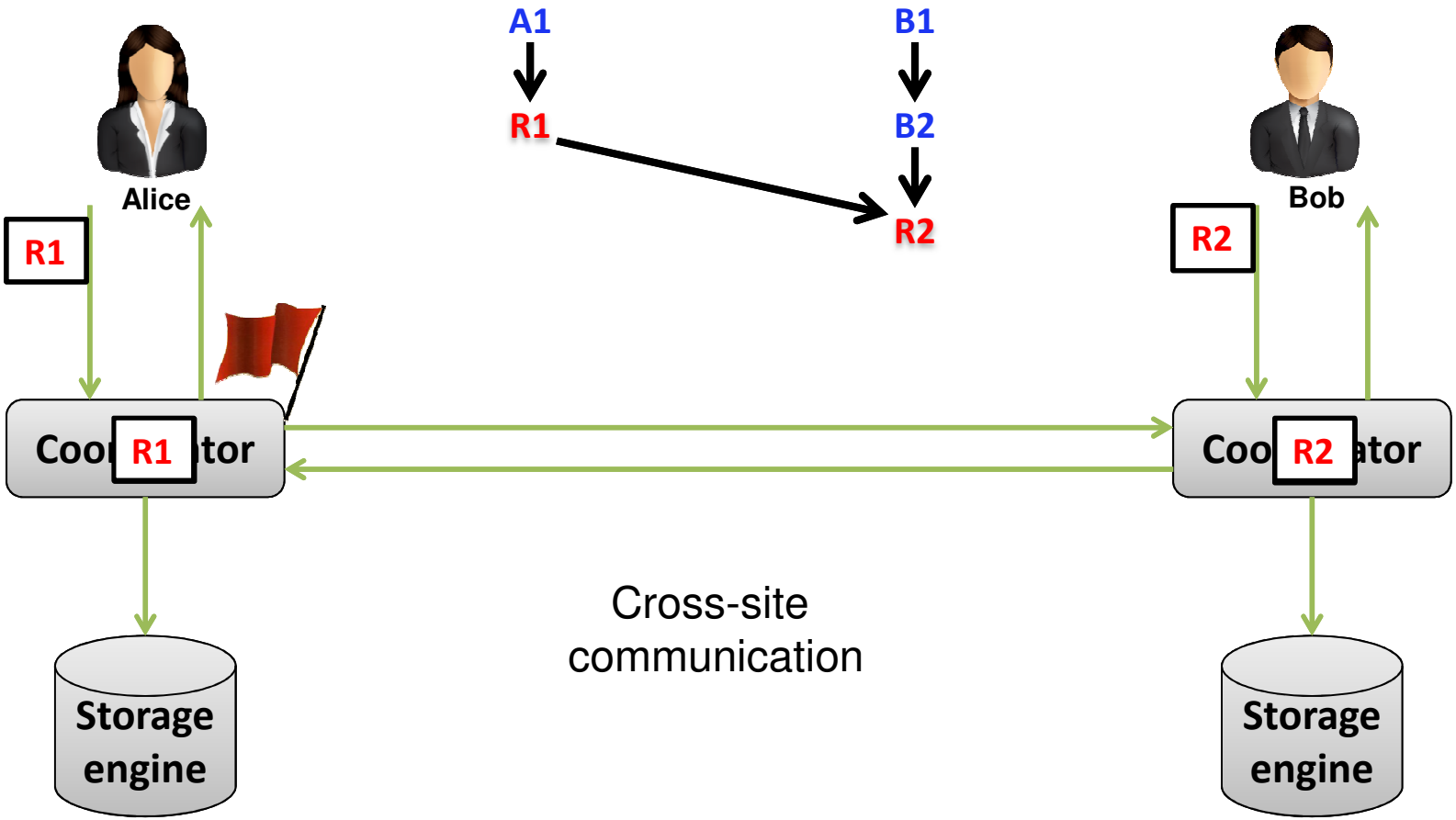


Balance strong/eventual consistency

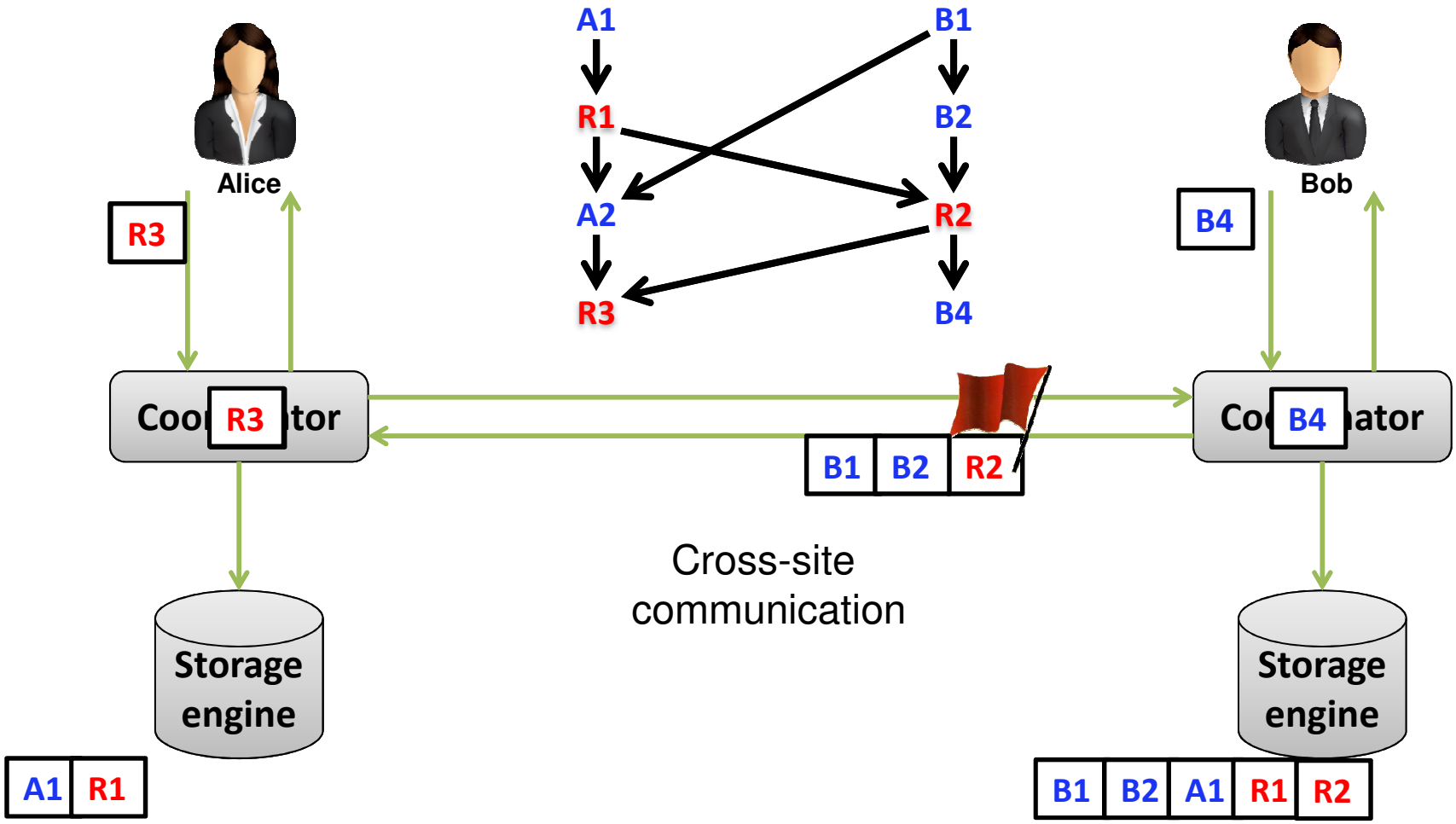


- Low latency of eventual consistency when possible
- Coordination for strong consistency only when necessary

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.

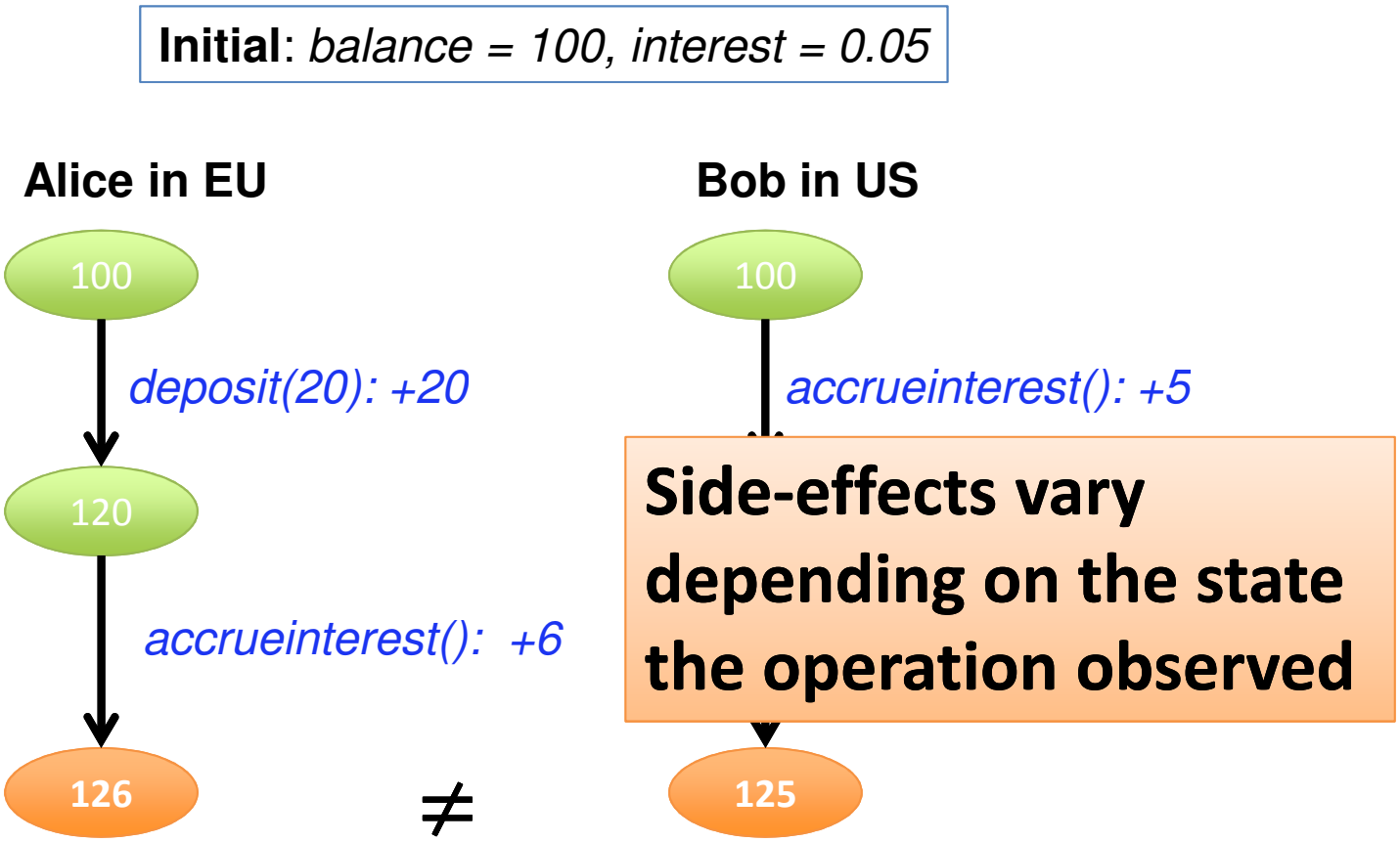
126 ≠ 125

```
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

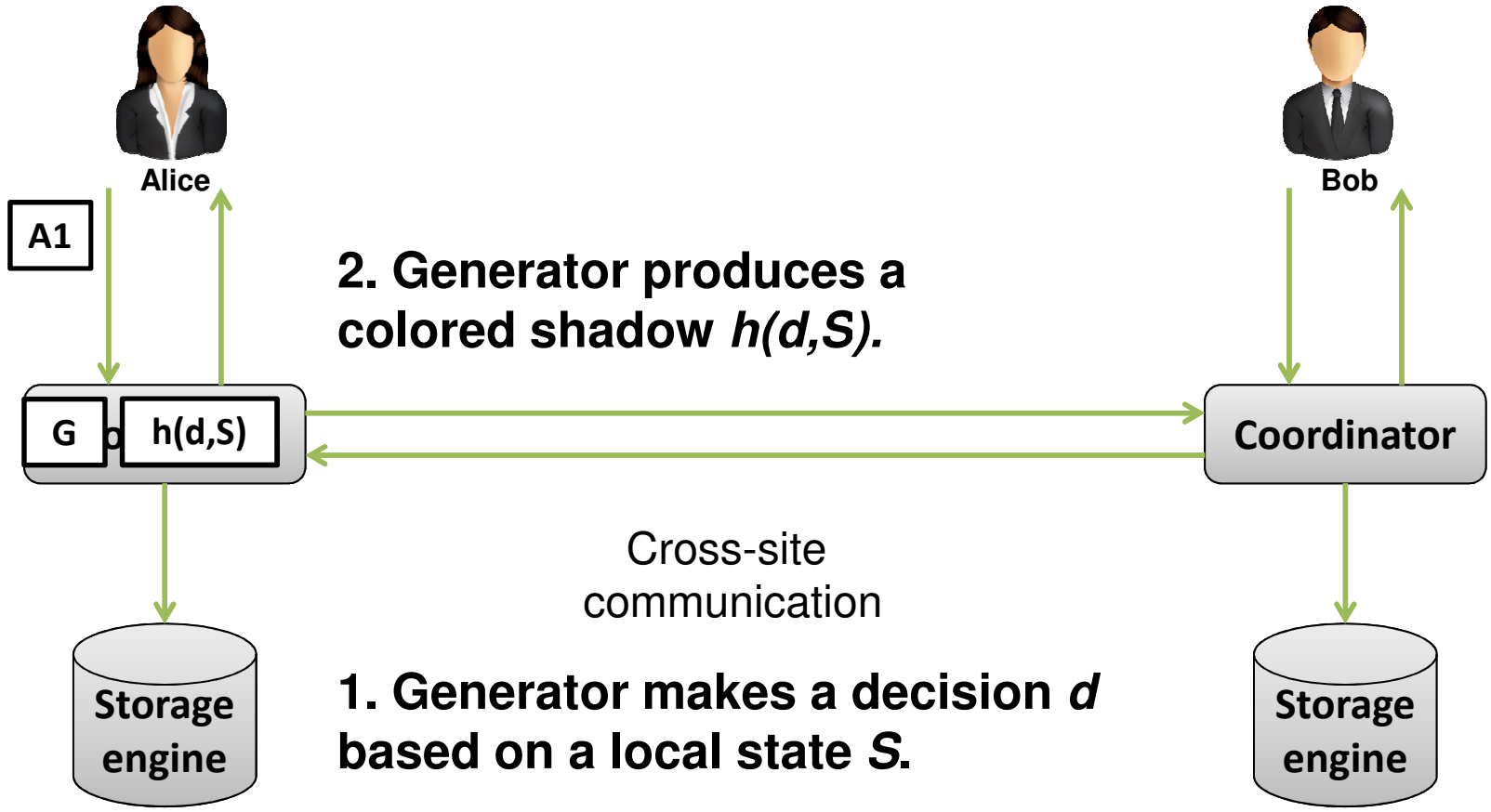
Problem of replicating operations



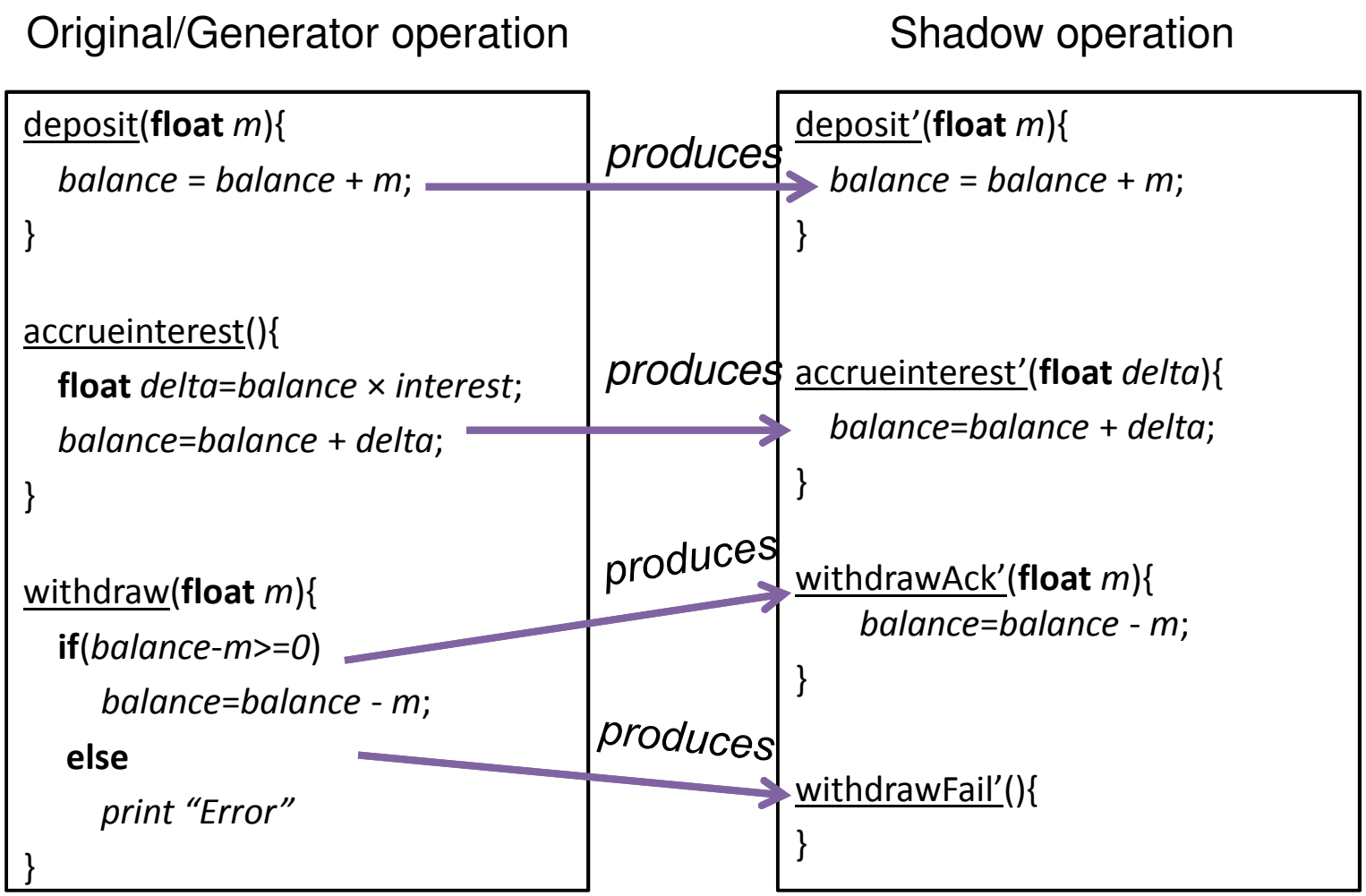
Generator/Shadow operation

- 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*

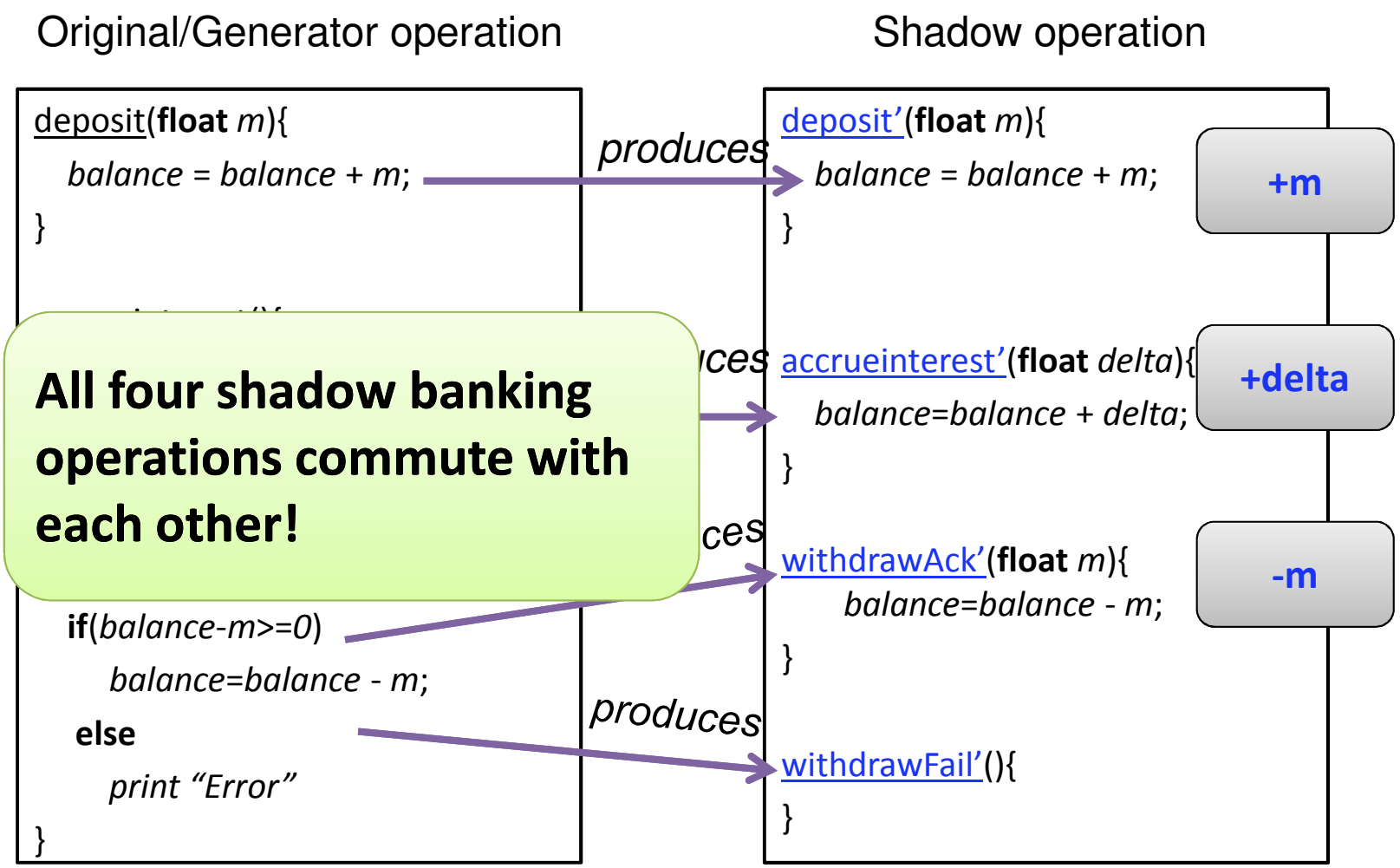
Generate once, shadow everywhere



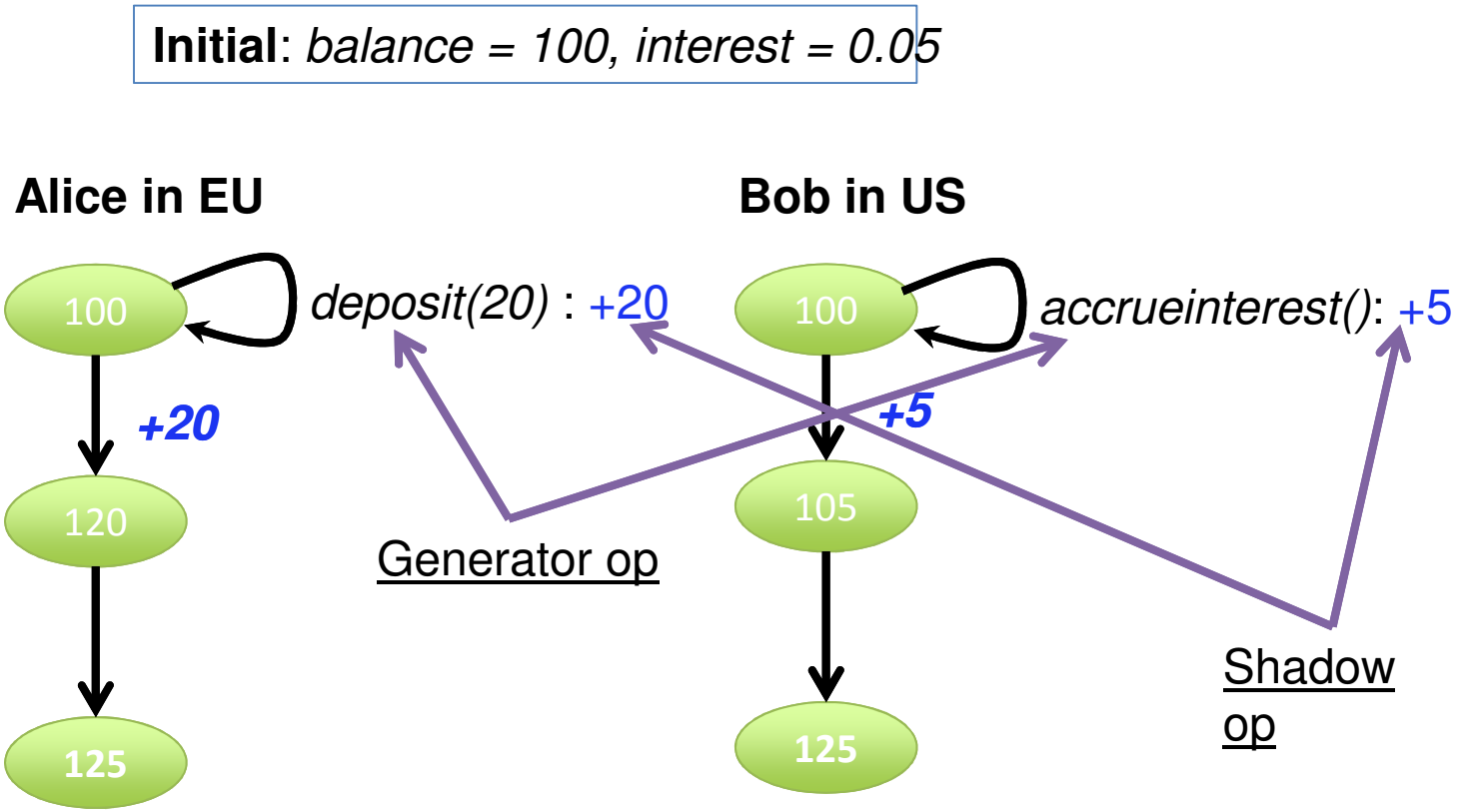
Bank generator/shadow operations



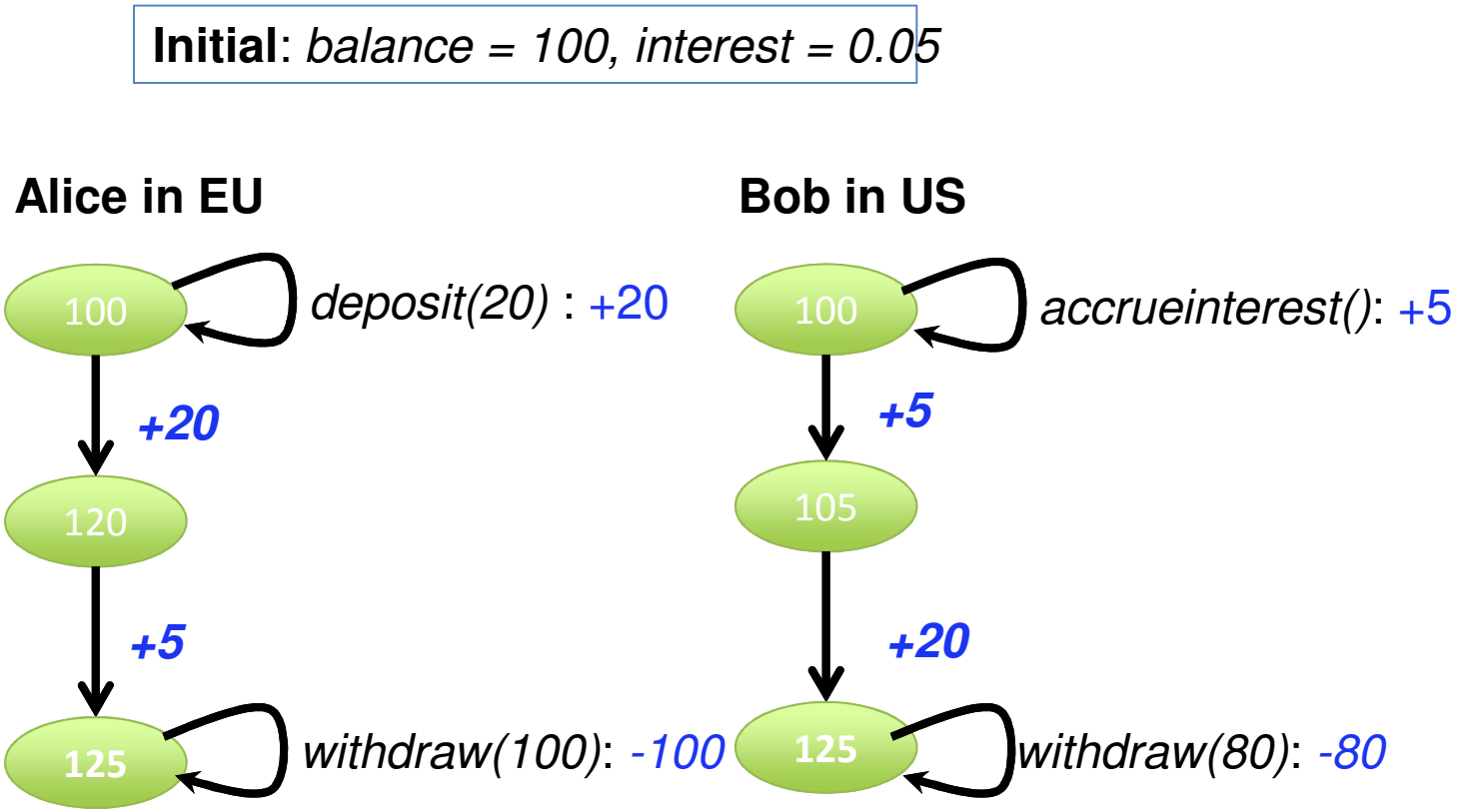
Bank generator/shadow operations



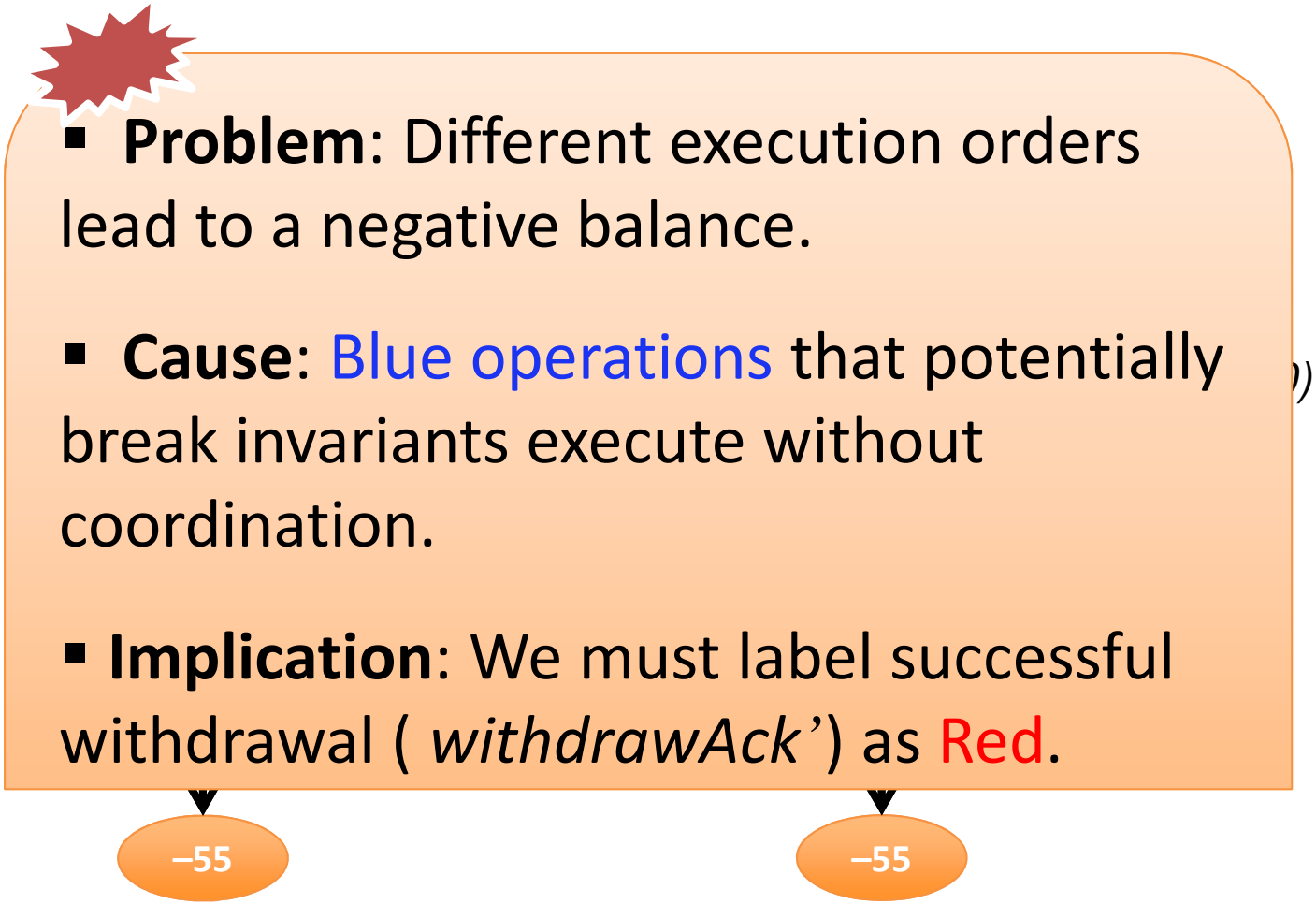
Fast and consistent bank



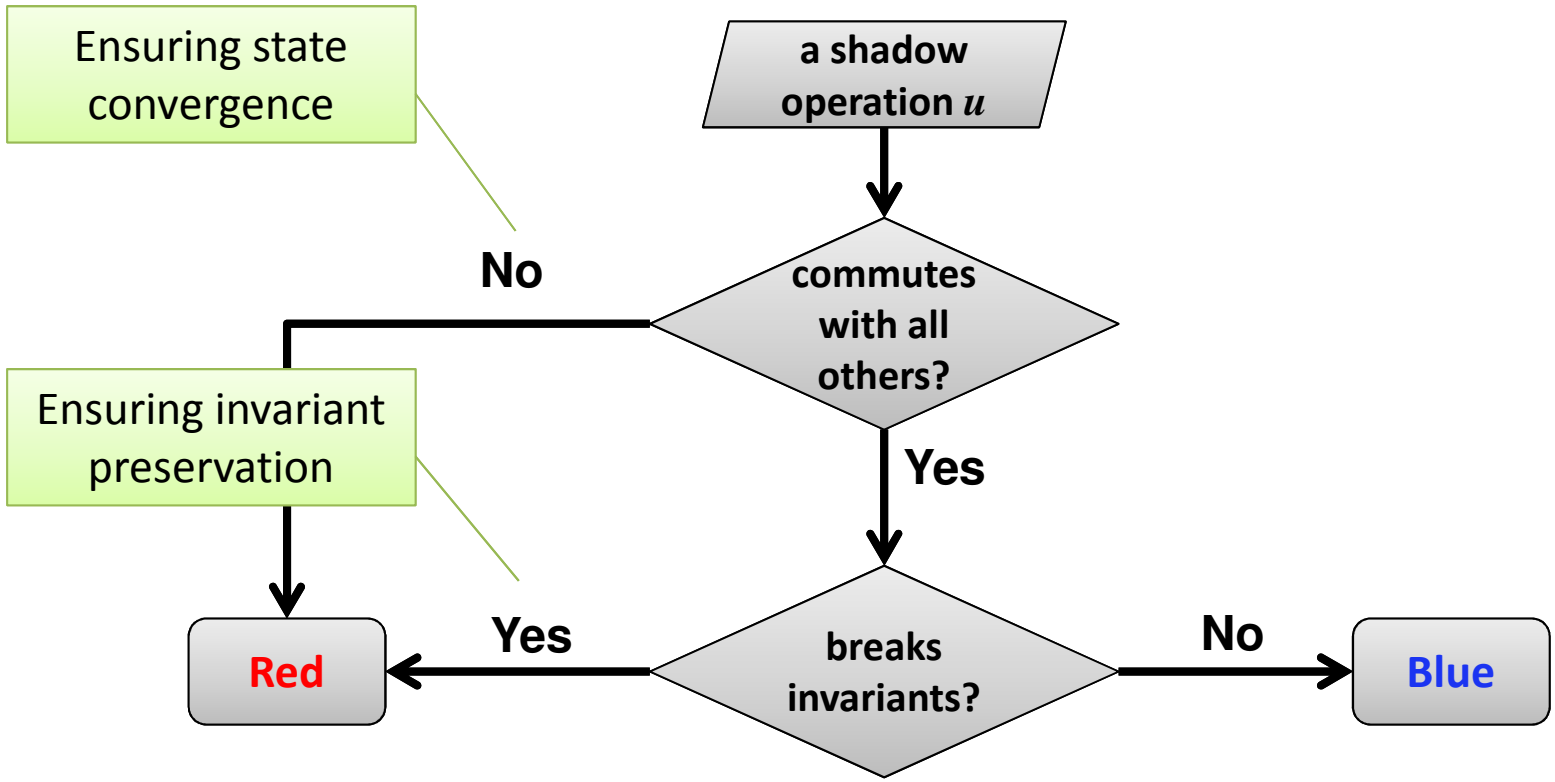
Not so fast …


Initial: *balance = 100, interest = 0.05*
Alice in EU
100
deposit(20) : +20
+20
120
+5
125
withdraw(100): -100
Bob in US
100
accrueinterest(): +5
+5
105
+20
125
withdraw(80): -80

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.
–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.

Questions

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

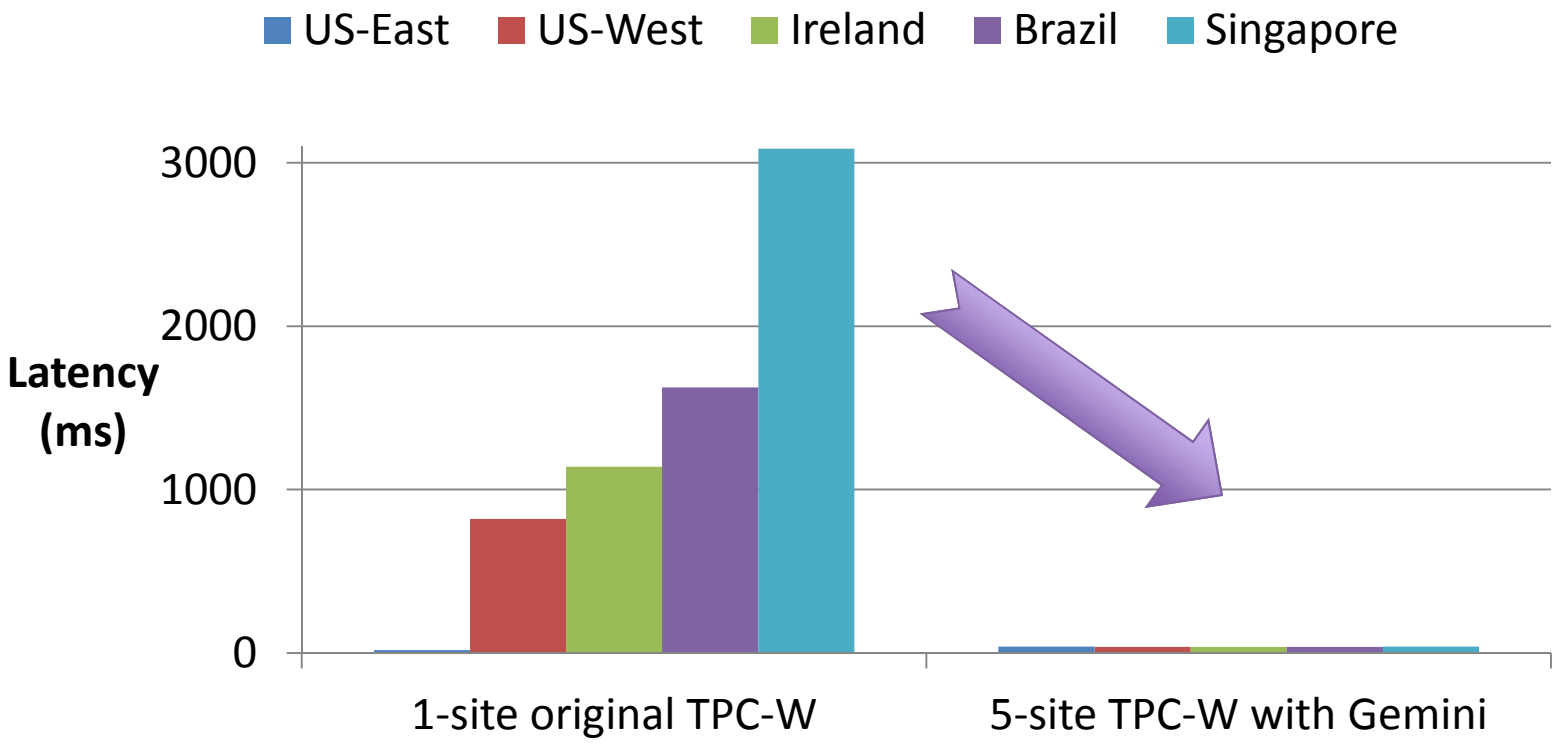
Experimental setup

- 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

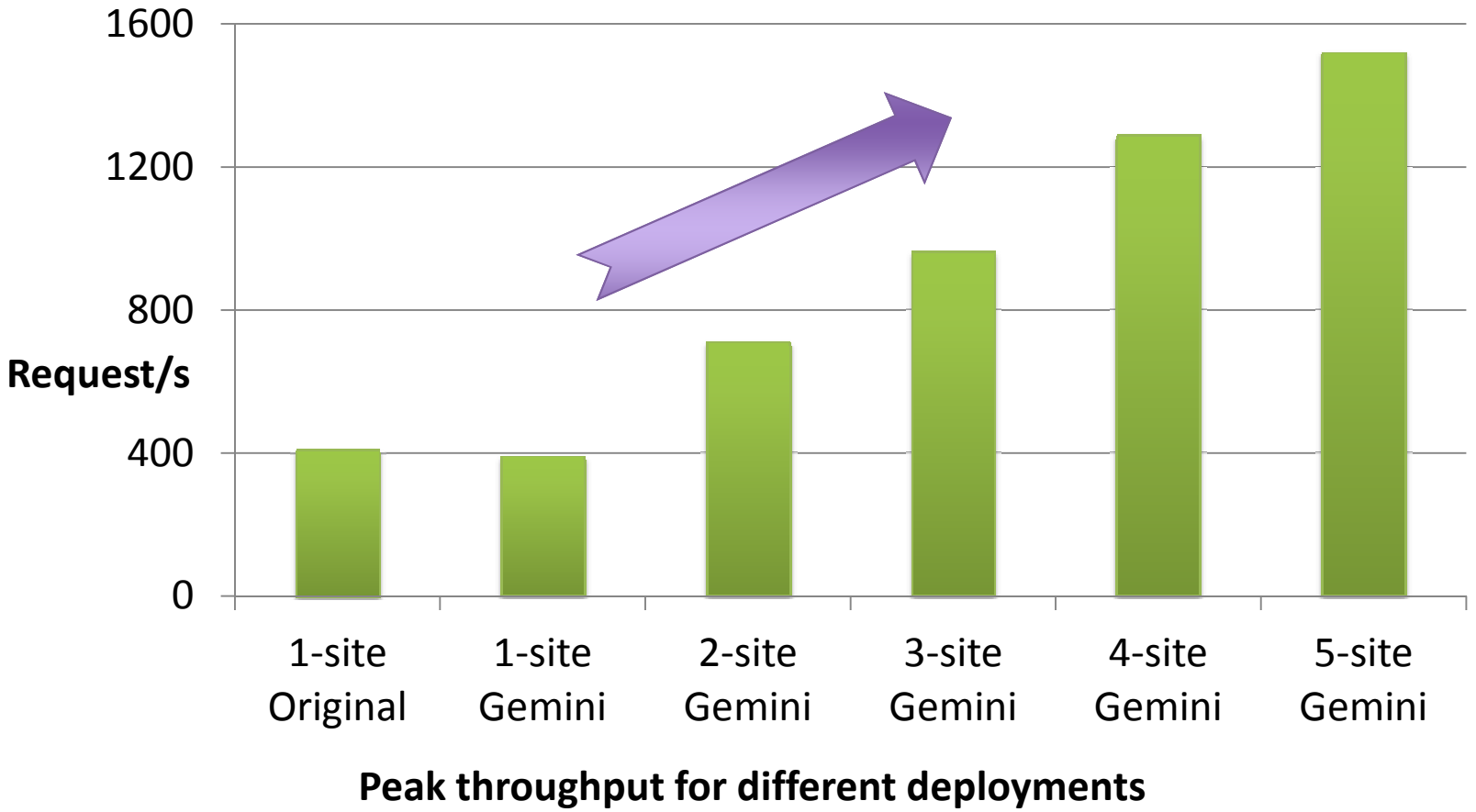
- Experiments with:
  - TPC-W, RUBiS and Quoddy
- Deployment in Amazon EC2
  - spanning 5 sites (US-East, US-West, Ireland, Brazil, Singapore)
  - locating users in all five sites and directing their requests to closest server

Does RedBlue consistency improve user-observed latency?



Average latency for users at all five sites

Does throughput scale with the number of sites?



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 when Possible, Consistent if Necessary

THANK YOU!