Caching and consistency

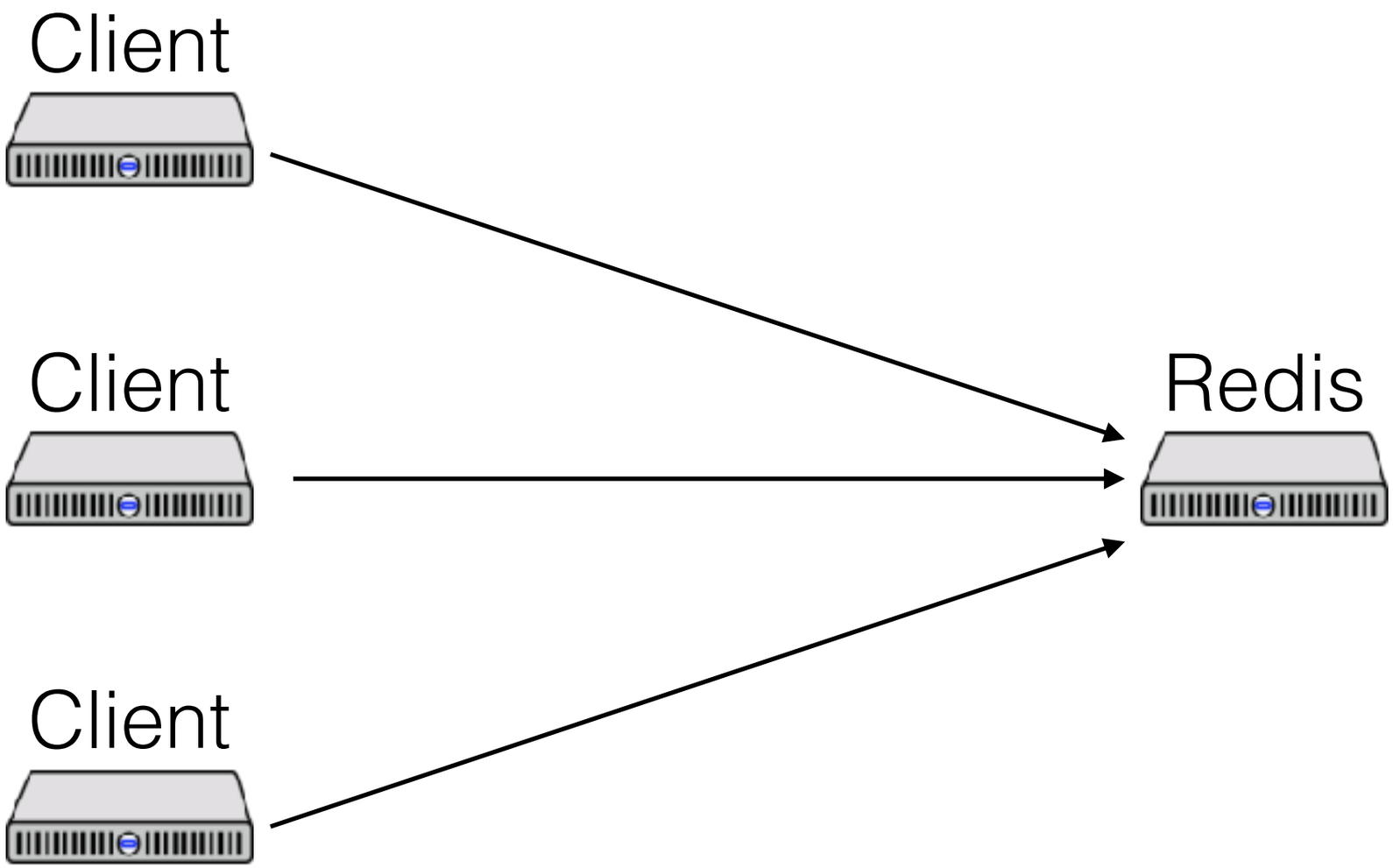
Doug Woos

Logistics notes

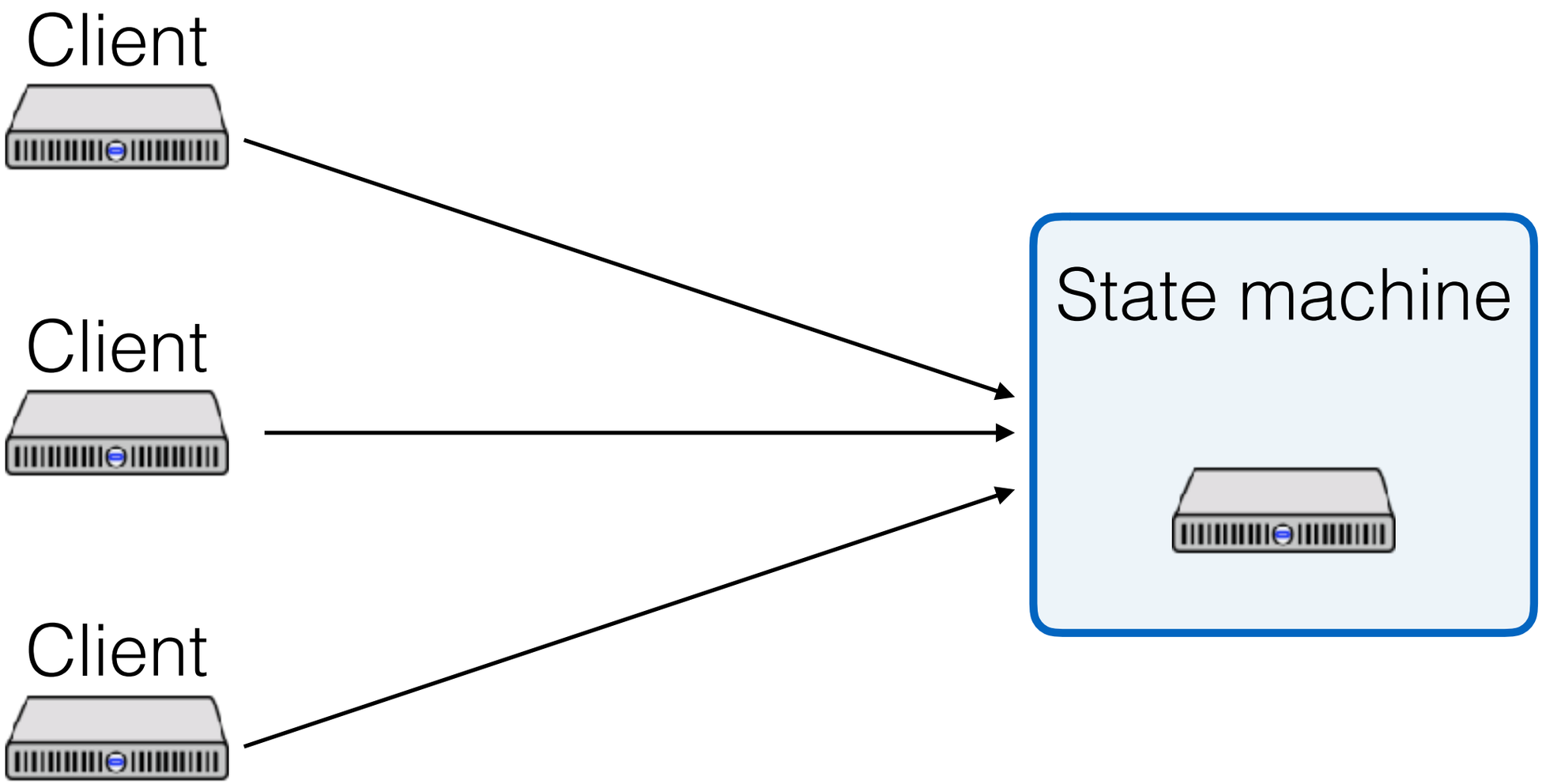
Problem Set 1 due tonight

Paper for Monday: Role of Distributed State

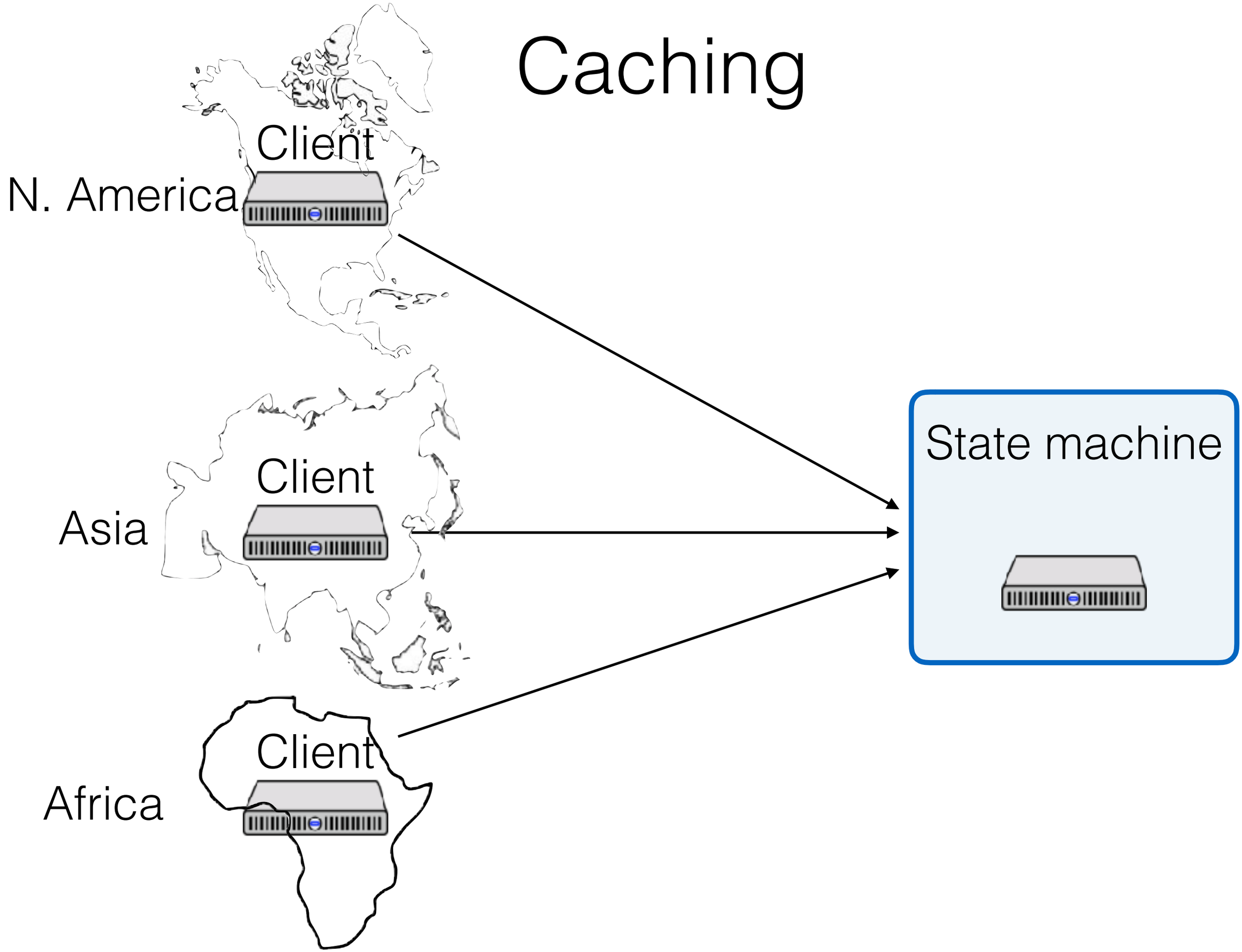
Caching



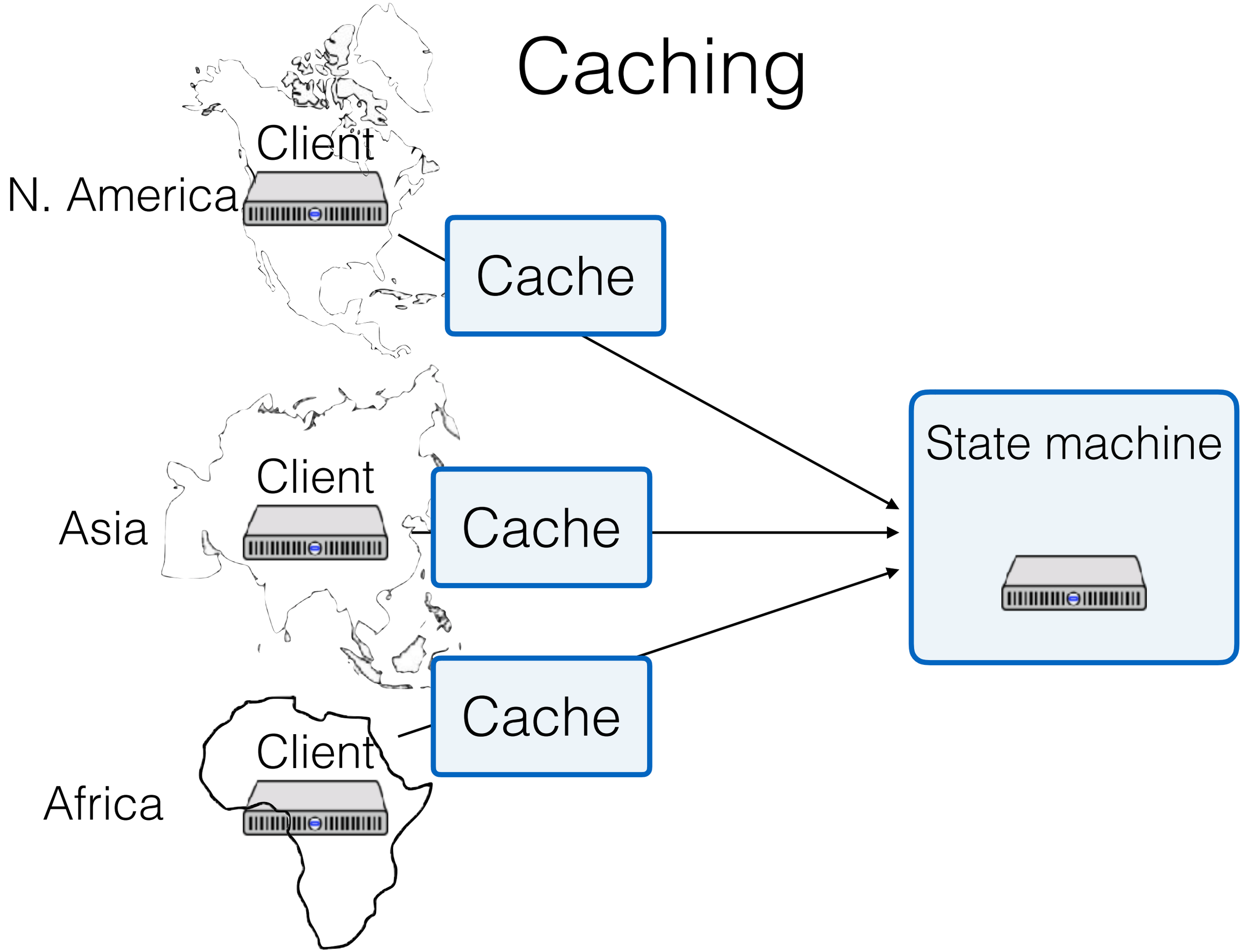
Caching



Caching



Caching


Client
N. America
Cache
State machine
Client
Asia
Cache
Cache
Client
Africa

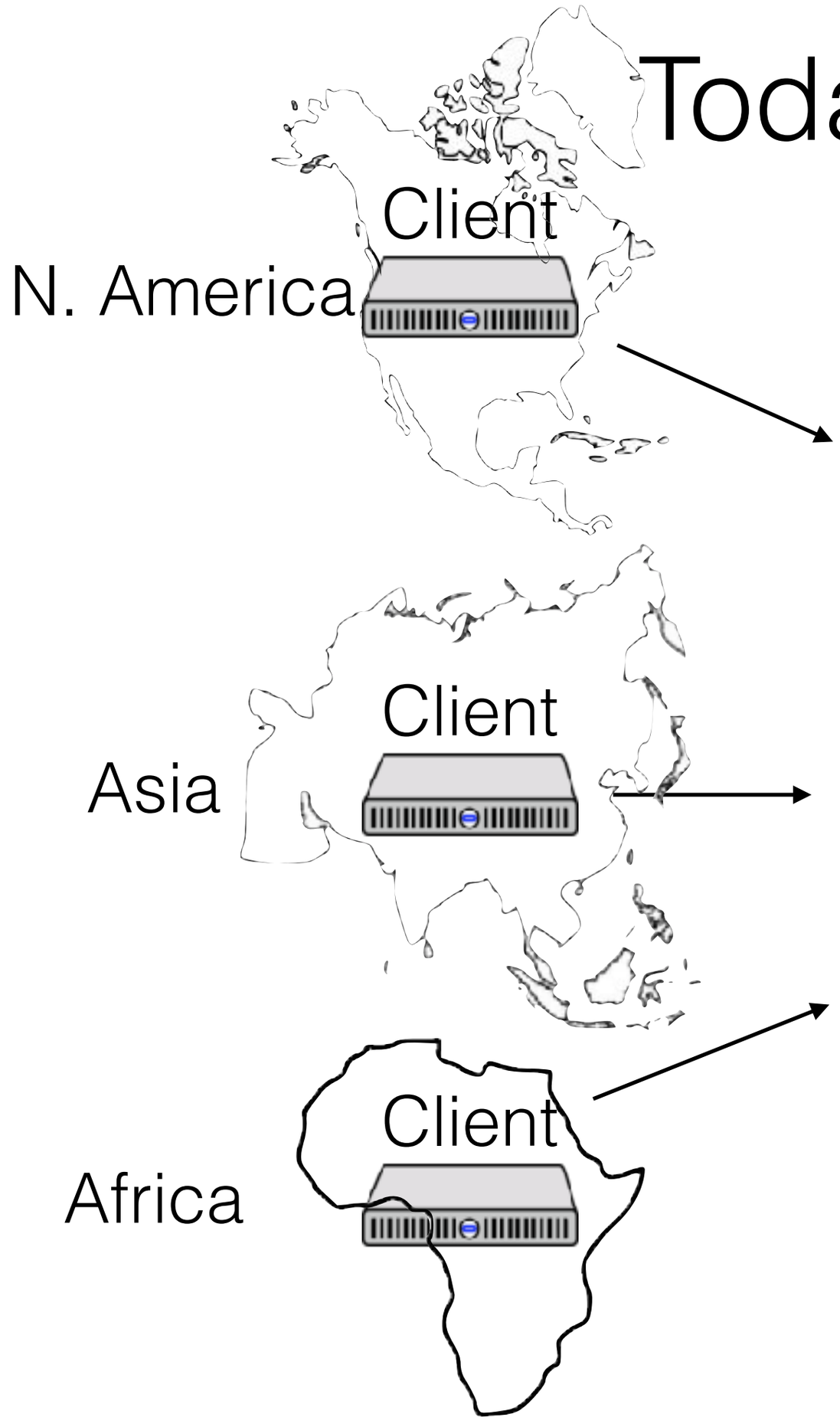
Caching

Cache

Cache

Cache

Today, mostly



System
+
Caches

???

???

read(k)

read(k)

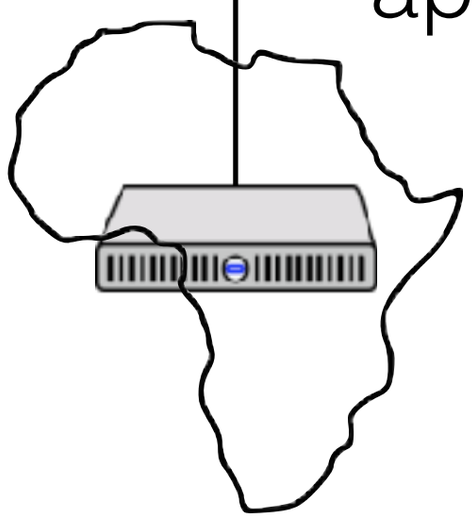
ok

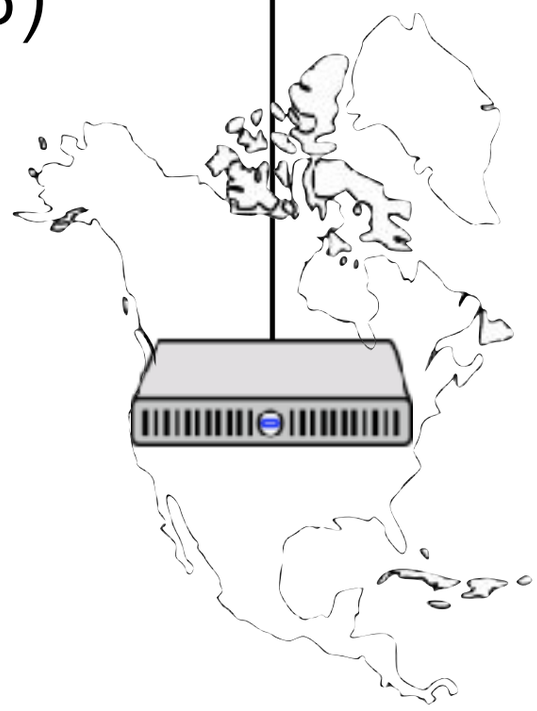
ok

append(k, B)

append(k, A)

System
+
Caches





Terminology

Anomaly: some sequence of operations (reads and writes) that “shouldn’t” be allowed

Coherence model/consistency model/memory model

- Which anomalies are possible
- Terms more or less interchangeable

Some classes of consistency model:

- Strong consistency: matches the ideal system
- Weak consistency: could have anomalies
- Eventual consistency: anomalies are “temporary”

Why different models?

Tradeoff between:

- Performance: consistency requires sync
- Availability: want to operate when disconnected
- Programmability: weaker consistency makes applications harder to write (i.e., harder to provide app-level guarantees)

CAP

- Consistency, availability, partition tolerance
- If you want availability, must give up consistency

Strict consistency

Strict Consistency (or Linearizability)

- Equivalent to ideal model

- Reads always reflect latest write

- Concurrent operations can be executed in any order

???

???

read(k)

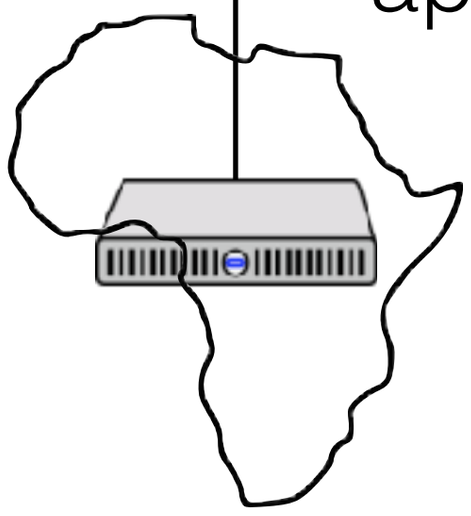
read(k)

ok

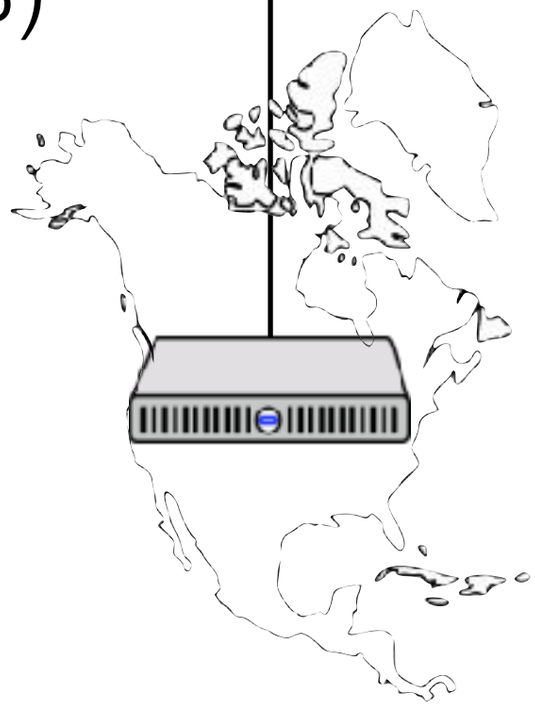
ok

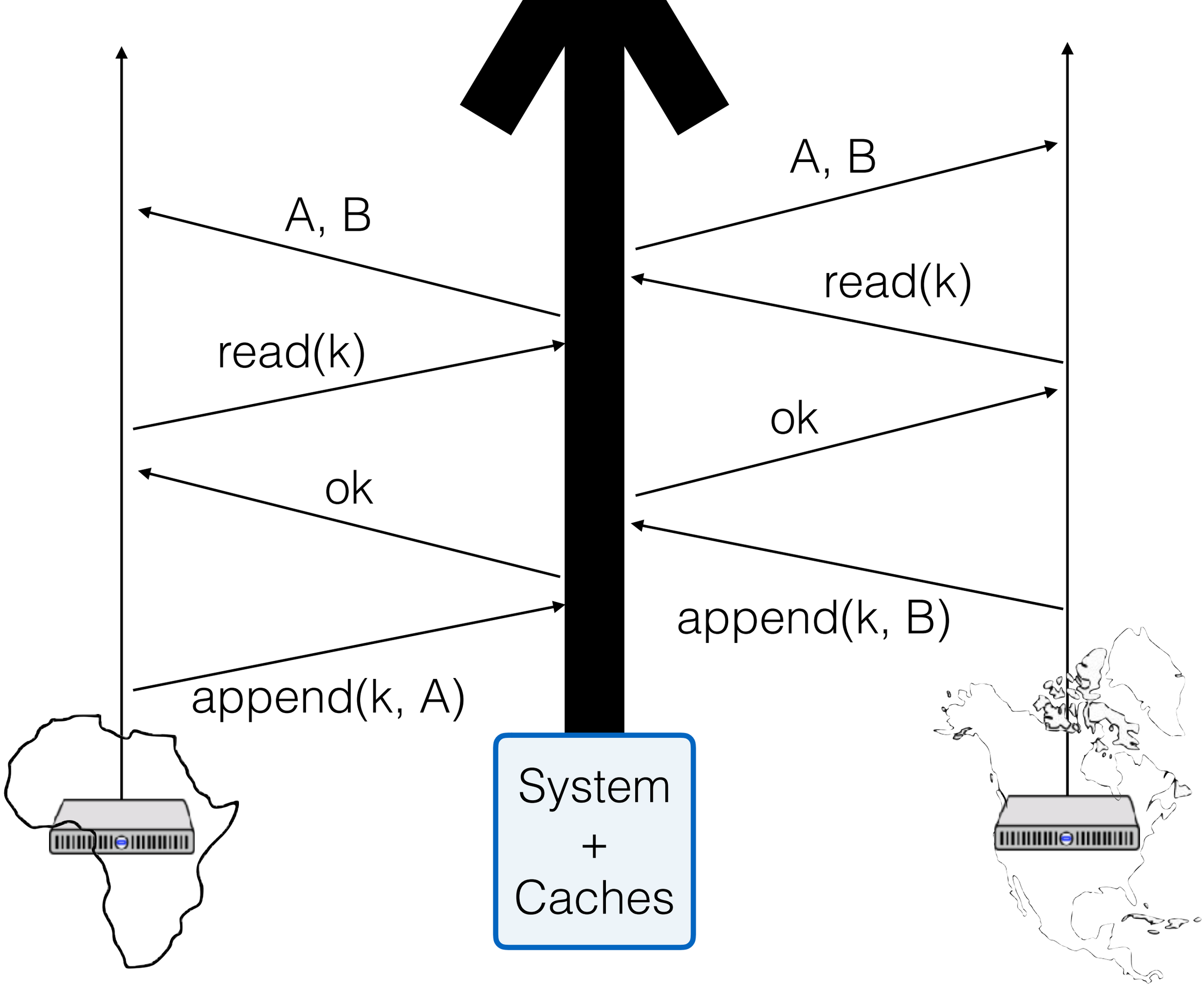
append(k, B)

append(k, A)



System
+
Caches




A, B
A, B
read(k)
read(k)
ok
ok
append(k, B)
append(k, A)
System
+
Caches

???

???

read(k)

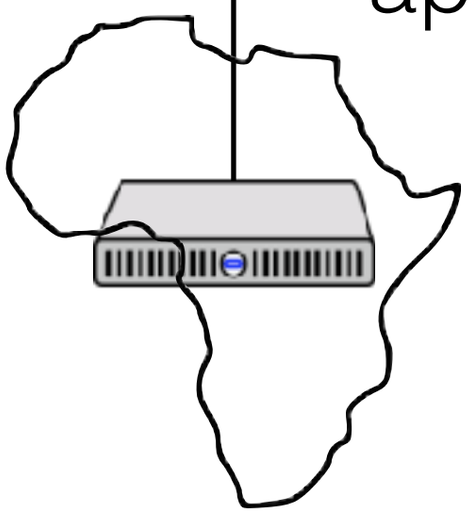
read(k)

ok

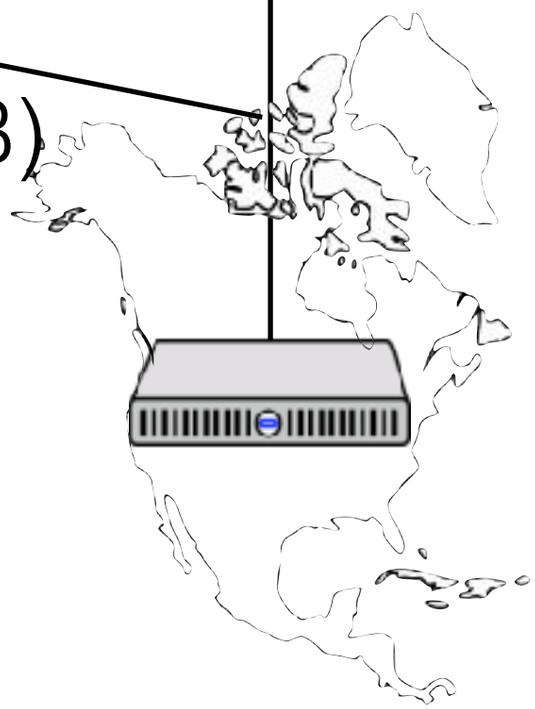
ok

append(k, A)

append(k, B)



System
+
Caches



???

???

read(k)

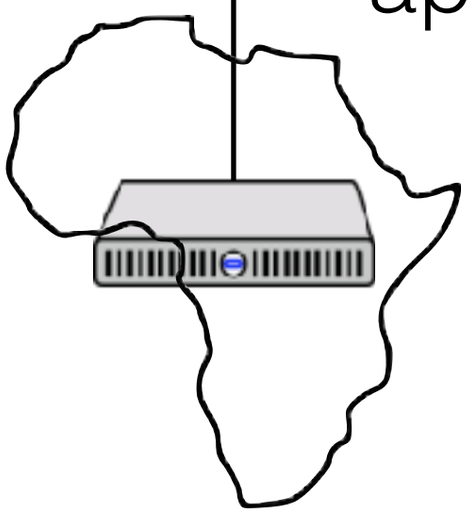
read(k)

ok

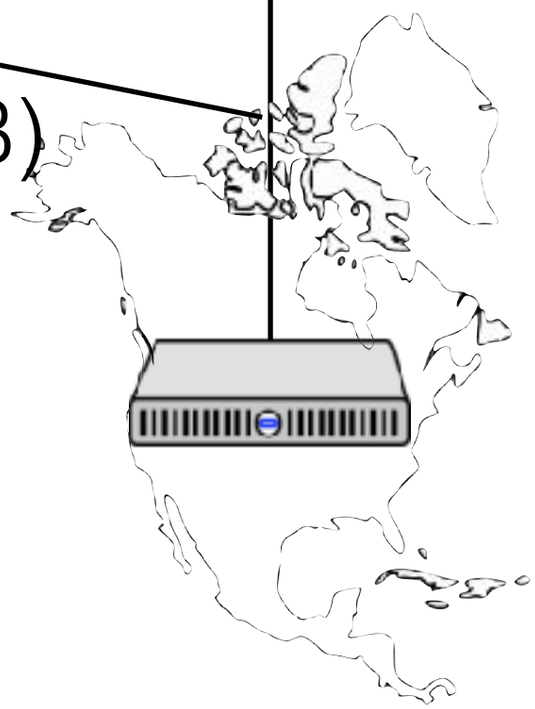
ok

append(k, A)

append(k, B)



System
+
Caches



Sequential Consistency

Sequential Consistency (or Serializability)

- Execution always equivalent to some interleaving
- Each node's ops done in order
- An intuition: strict consistency, but without real time

???

???

read(k)

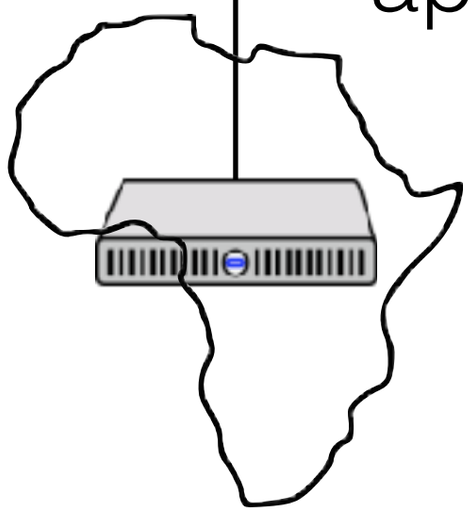
read(k)

ok

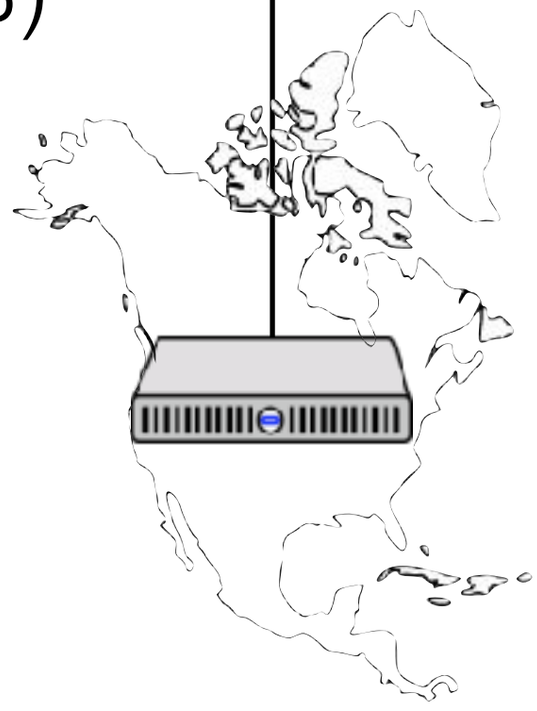
ok

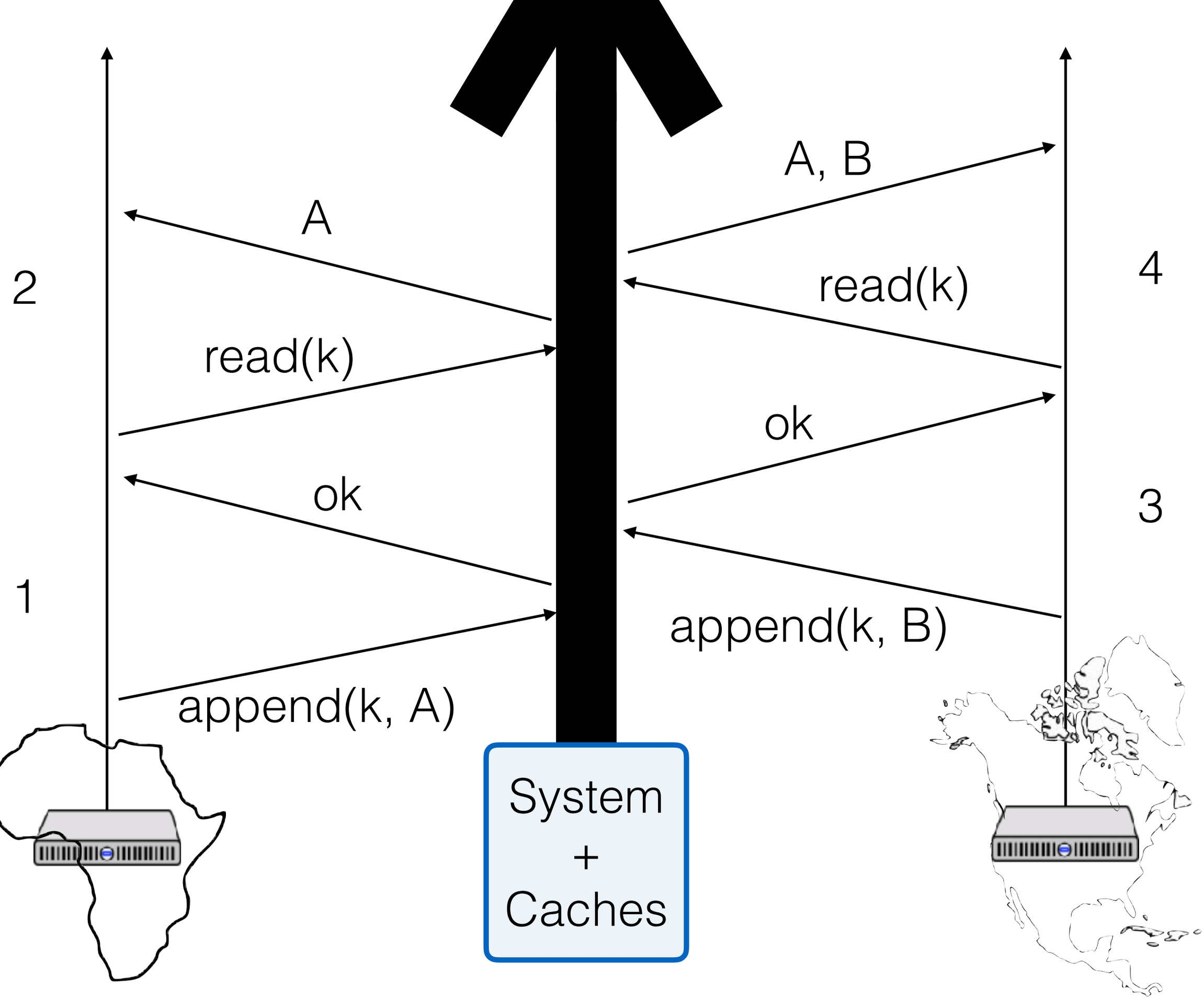
append(k, B)

append(k, A)



System
+
Caches




2
A
A, B
4
read(k)
read(k)
ok
ok
3
1
append(k, B)
append(k, A)
System
+
Caches

3

4

2

1

A, B

A, B

read(k)

read(k)

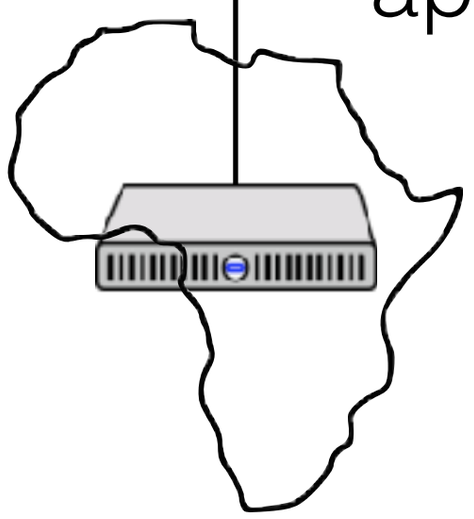
ok

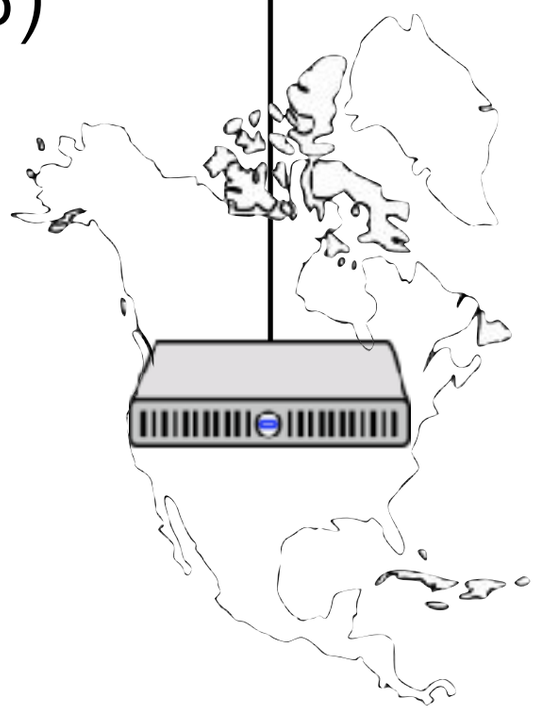
ok

append(k, A)

append(k, B)

System
+
Caches

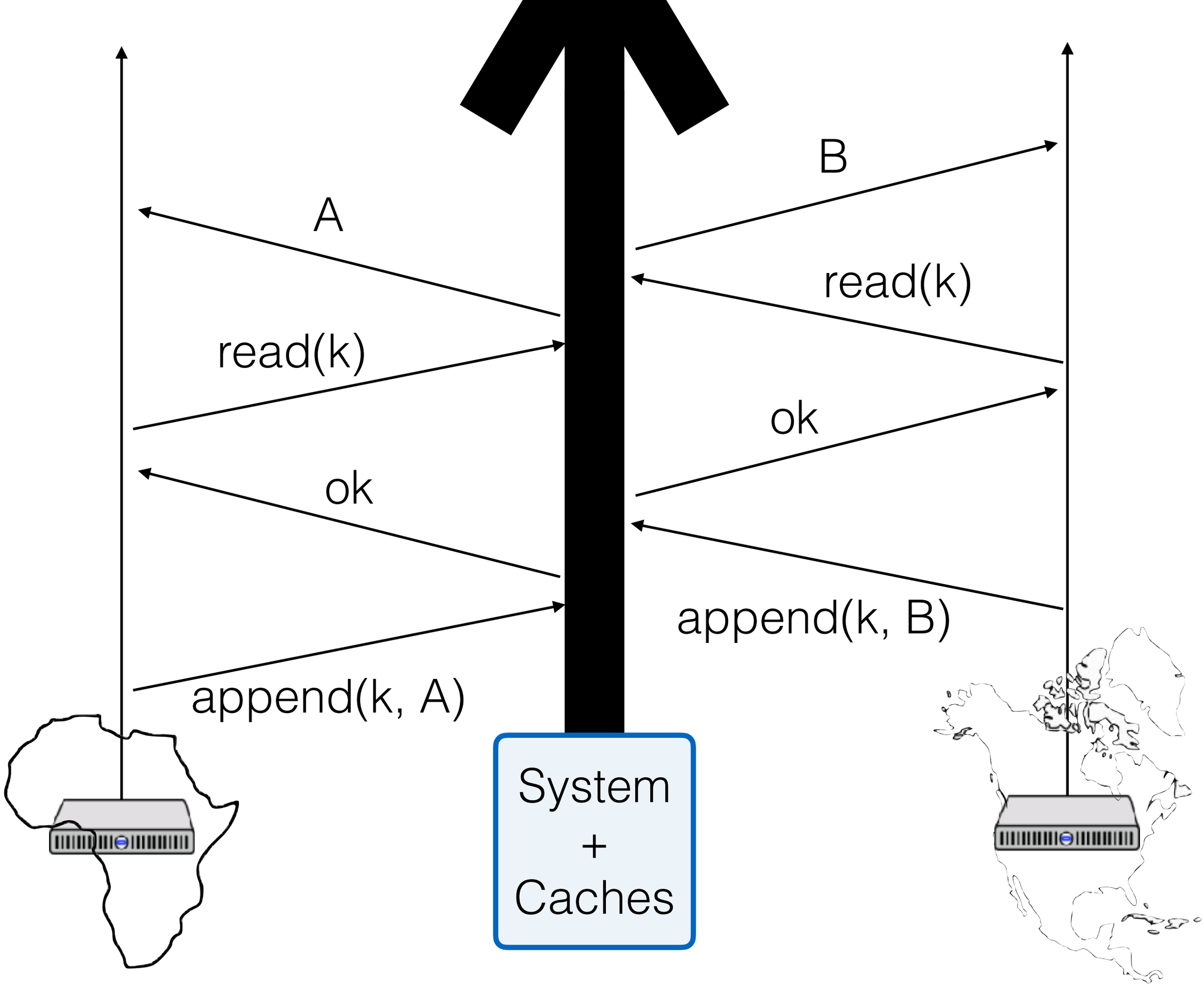


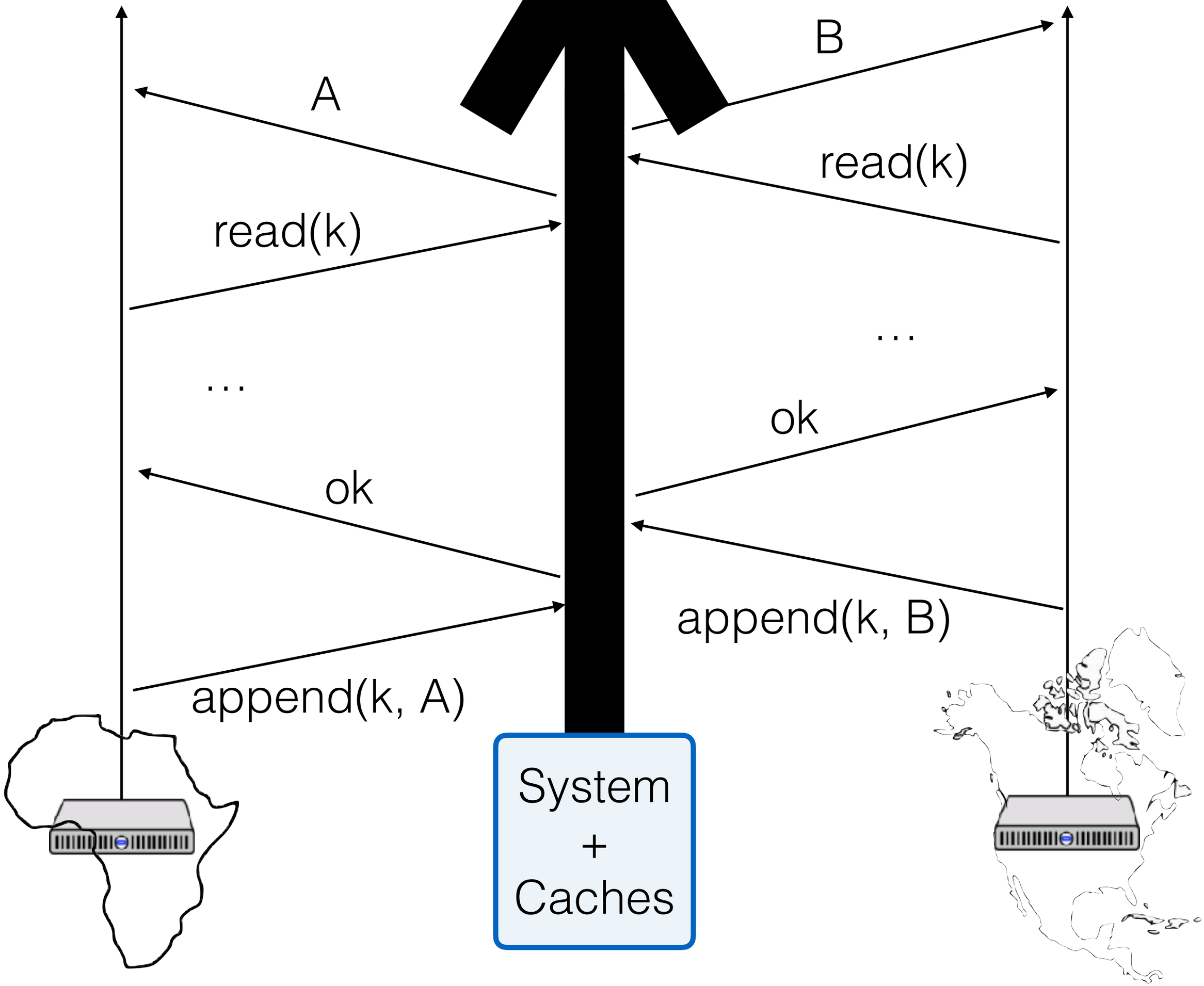


Weaker models

Read Your Writes + Eventual Consistency

- Facebook model, approximately
- Clients will always see their own writes
- They will eventually see everyone's writes
- And eventually the order will be consistent


A
B
read(k)
read(k)
ok
ok
append(k, A)
append(k, B)
System
+
Caches


A
B
read(k)
read(k)
...
...
ok
ok
append(k, B)
append(k, A)
System
+
Caches

Weaker models

Causal consistency

- Causal order (Lamport happens-before) observed everywhere
- Concurrent events can have arbitrary and inconsistent order

Transactional models (e.g. Snapshot reads)

- Some other consistency model + atomicity of transactions

Next couple of lectures

How to implement (various types of) consistency?

> “There are only two hard things in Computer Science: cache invalidation and naming things.”
>
> — Phil Karlton

If we cache data, how do we make sure it reflects writes of other nodes?

Without sacrificing performance?

A note about processors

The Silently Shifting Semicolon (Marino 2015):

```
Pasta p = new Pasta();
cooked = true;
```

```
if (cooked) {
  p.drain();
}
```

Why does this sometimes result in a null pointer error?

A note about processors

Processors deal with same problems

- L1 cache = 0.5ns, RAM = 100ns
  - How far from the CPU is RAM in your desktop?

Architecture + compiler conspire to move semicolon

Real problem in programming languages

My take: want sequential consistency everywhere