Distributed Systems

Spring Semester 2020

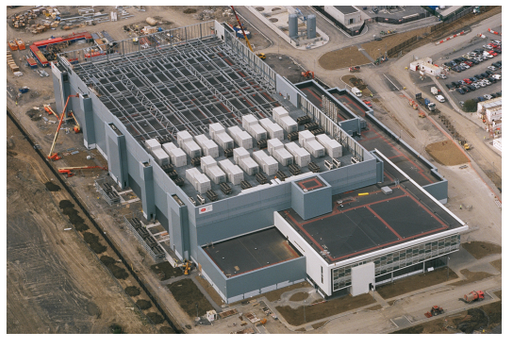
Lecture 17: Existential Consistency

John Liagouris
liagos@bu.edu

Why this paper

- Practical storage for giant read-heavy web sites — YET AGAIN ;-)
- Techniques for detecting anomalies
- Practical consistency concerns

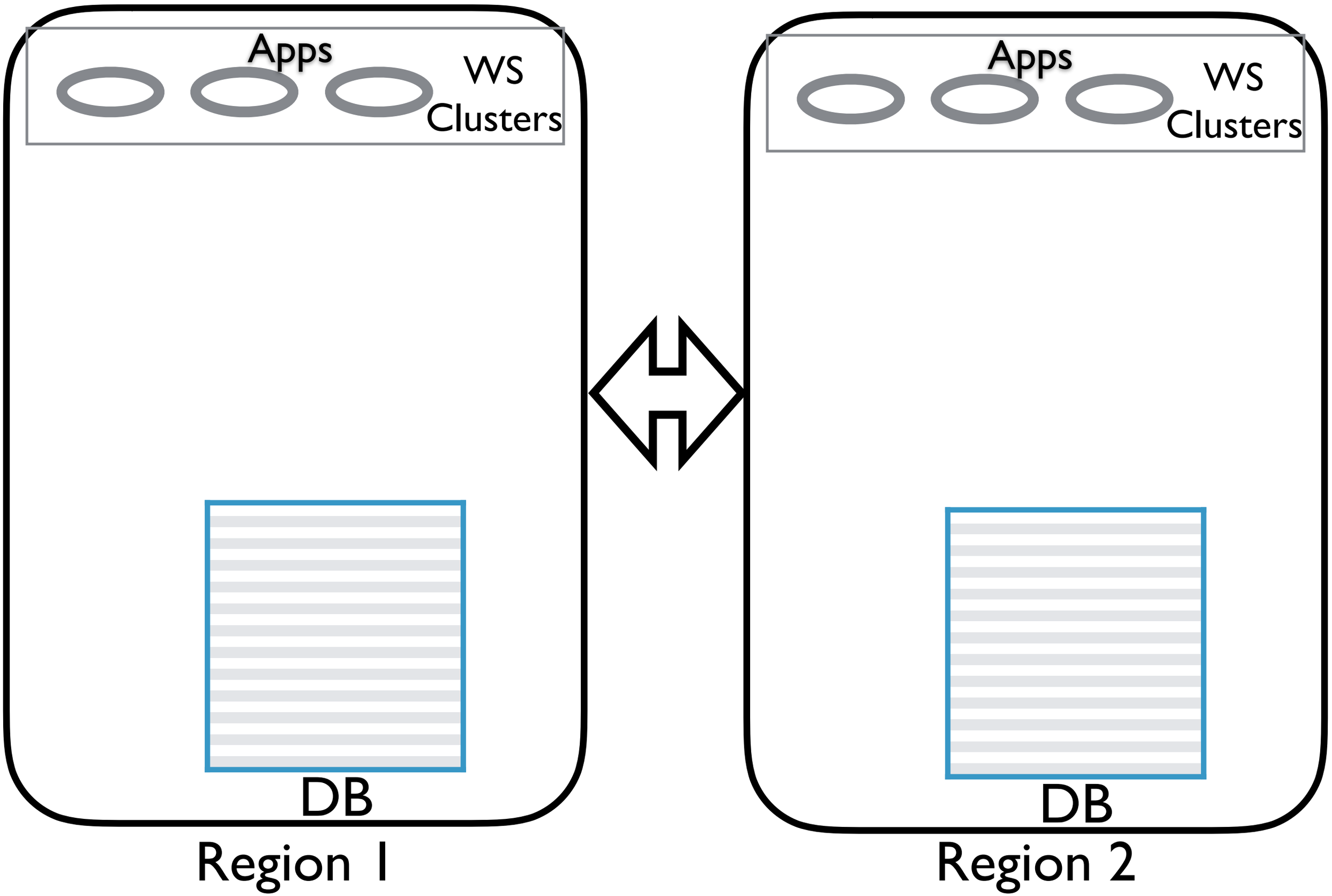
FB Infrastructure



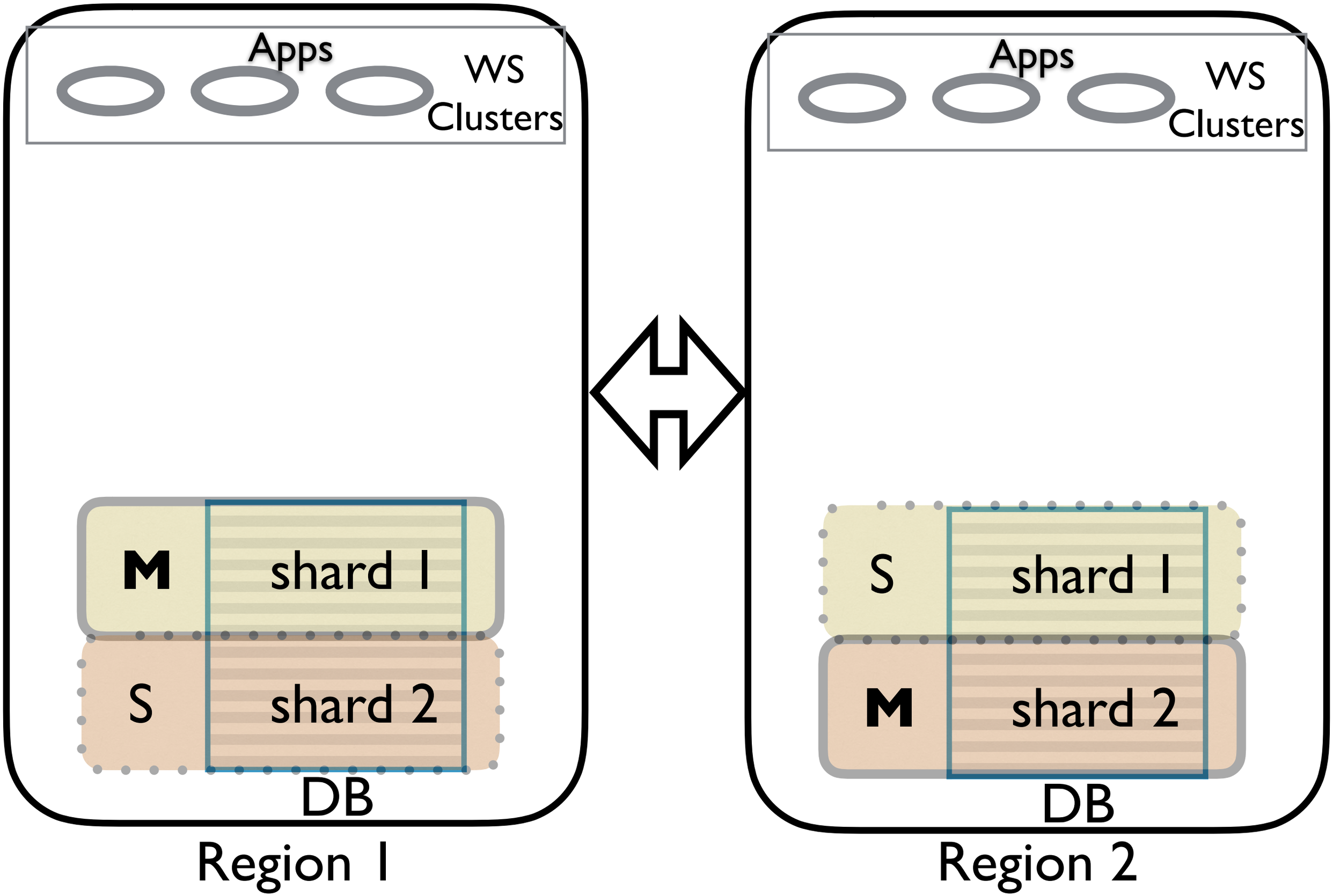
Region 1

Region 2

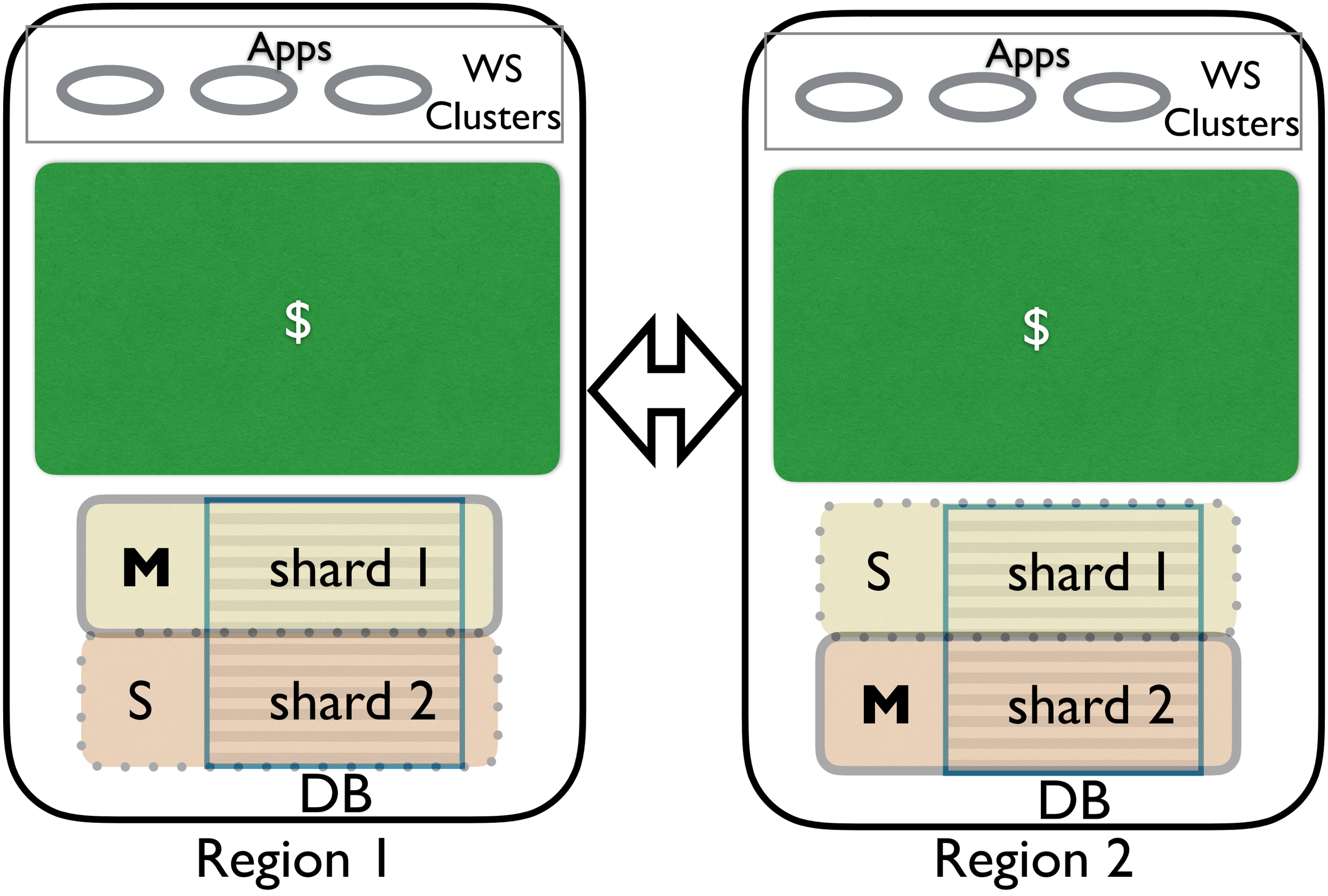
FB Infrastructure



FB Infrastructure



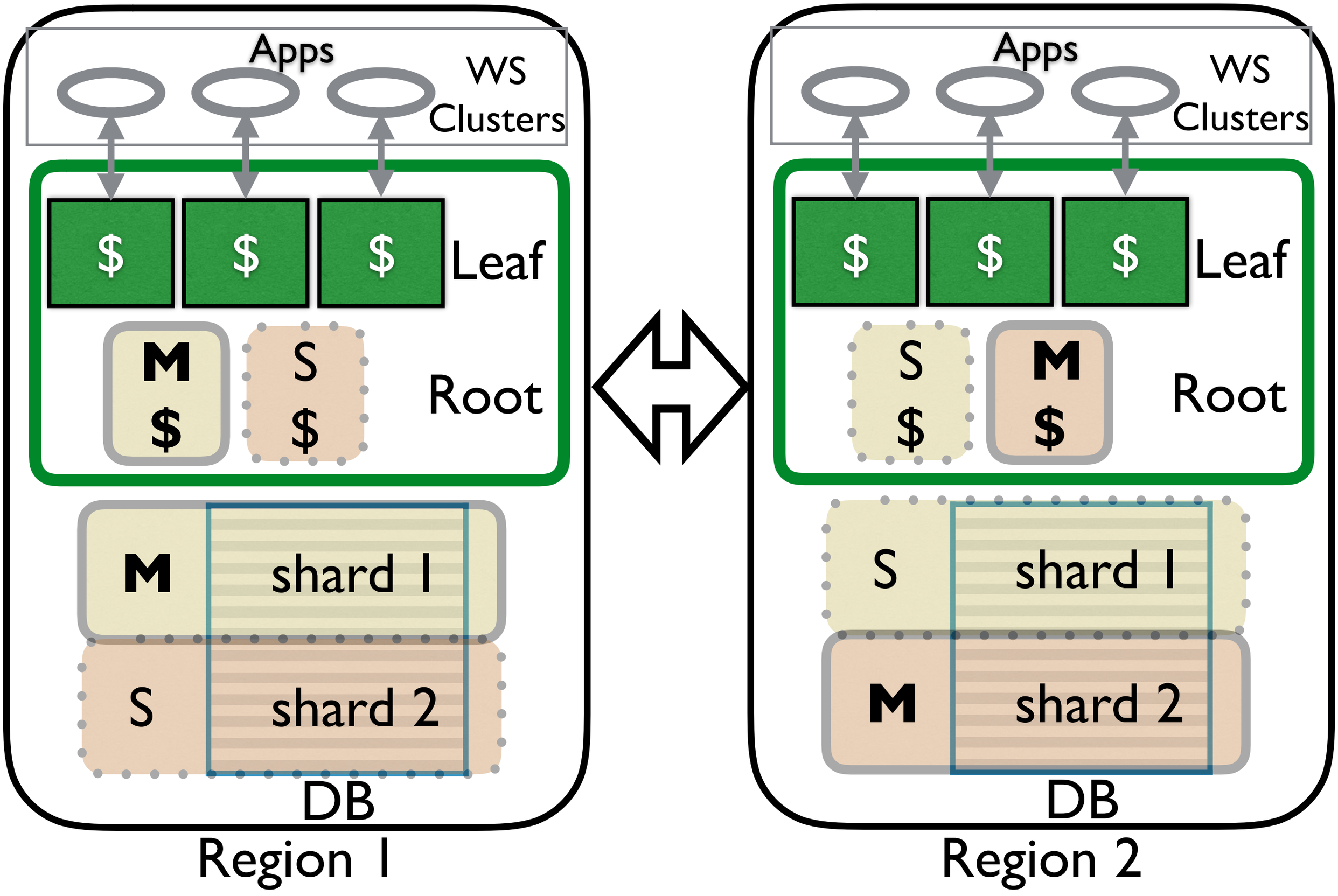
FB Infrastructure



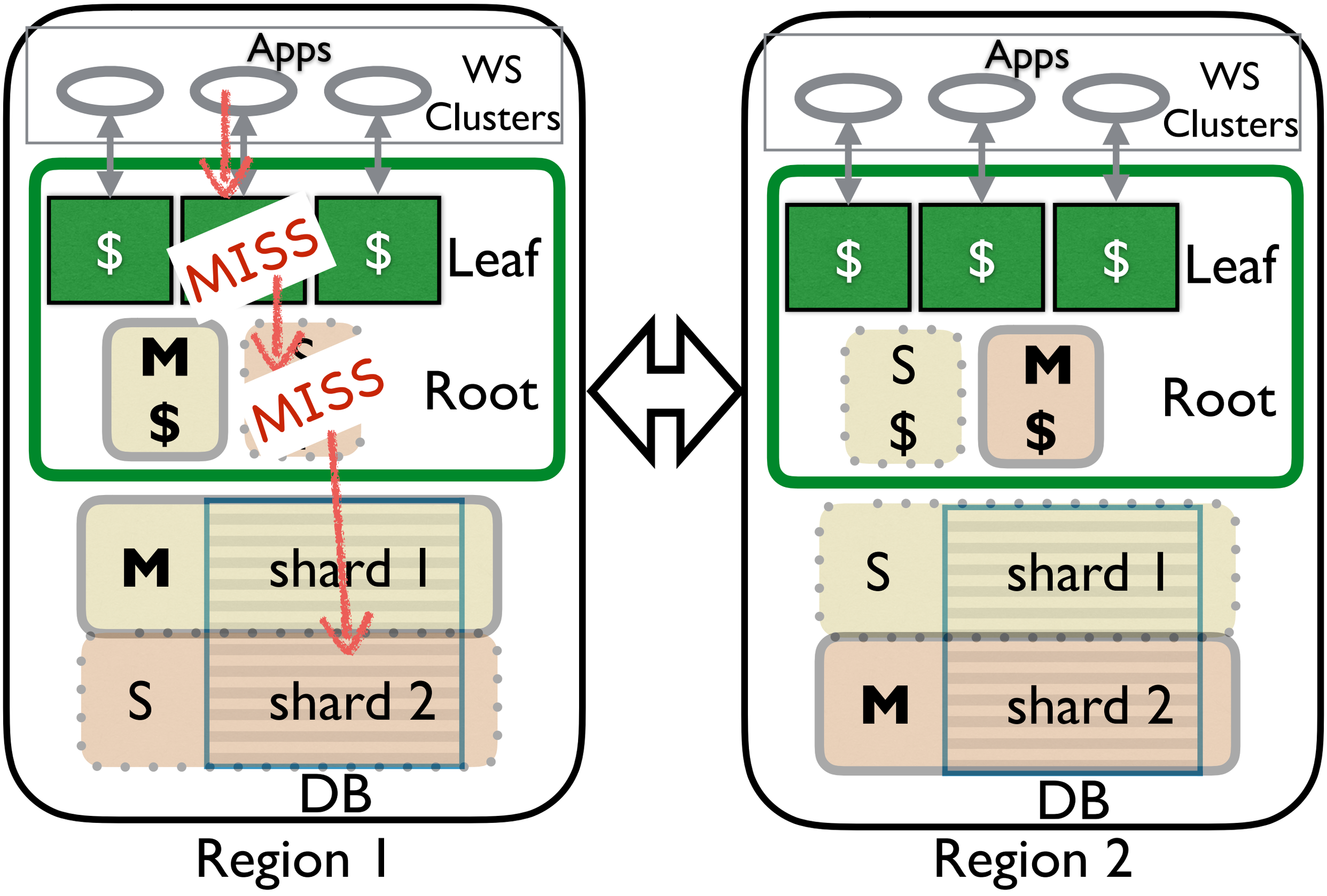
Why this overall arrangement?

- why multiple geographic sites (regions)?
- why complete replica of DB for each region?
- why DB?
- why shared DB servers?

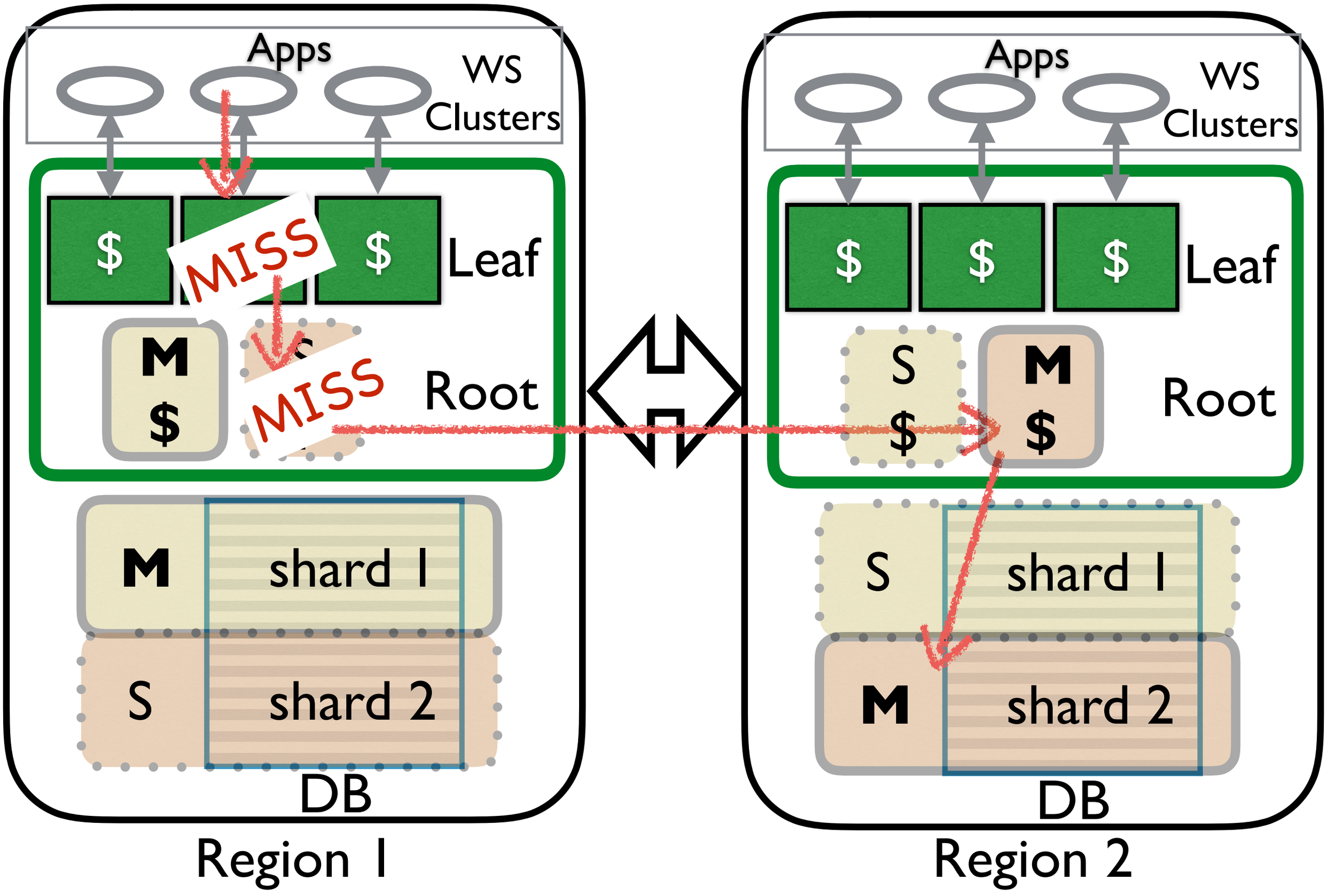
FB TAO


Apps
WS Clusters
$
$
$
Leaf
M
$
S
$
Root
M
shard 1
S
shard 2
DB
Region 1
Apps
WS Clusters
$
$
$
Leaf
S
$
M
$
Root
S
shard 1
M
shard 2
DB
Region 2

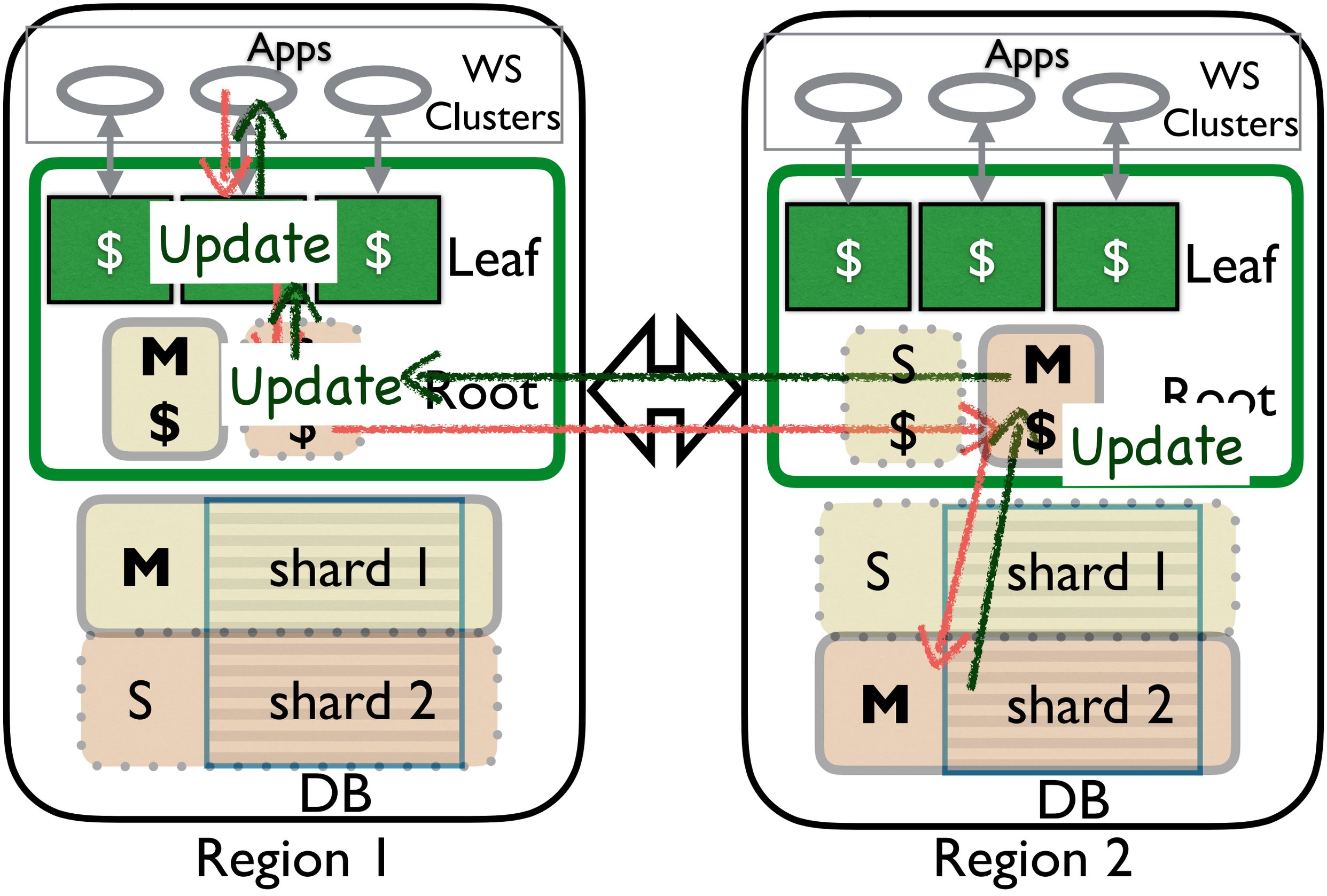
READS


Apps
WS Clusters
$
MISS
$
Leaf
M
$
MISS
Root
M
shard 1
S
shard 2
DB
Region 1
Apps
WS Clusters
$
$
$
Leaf
S
$
M
$
Root
S
shard 1
M
shard 2
DB
Region 2

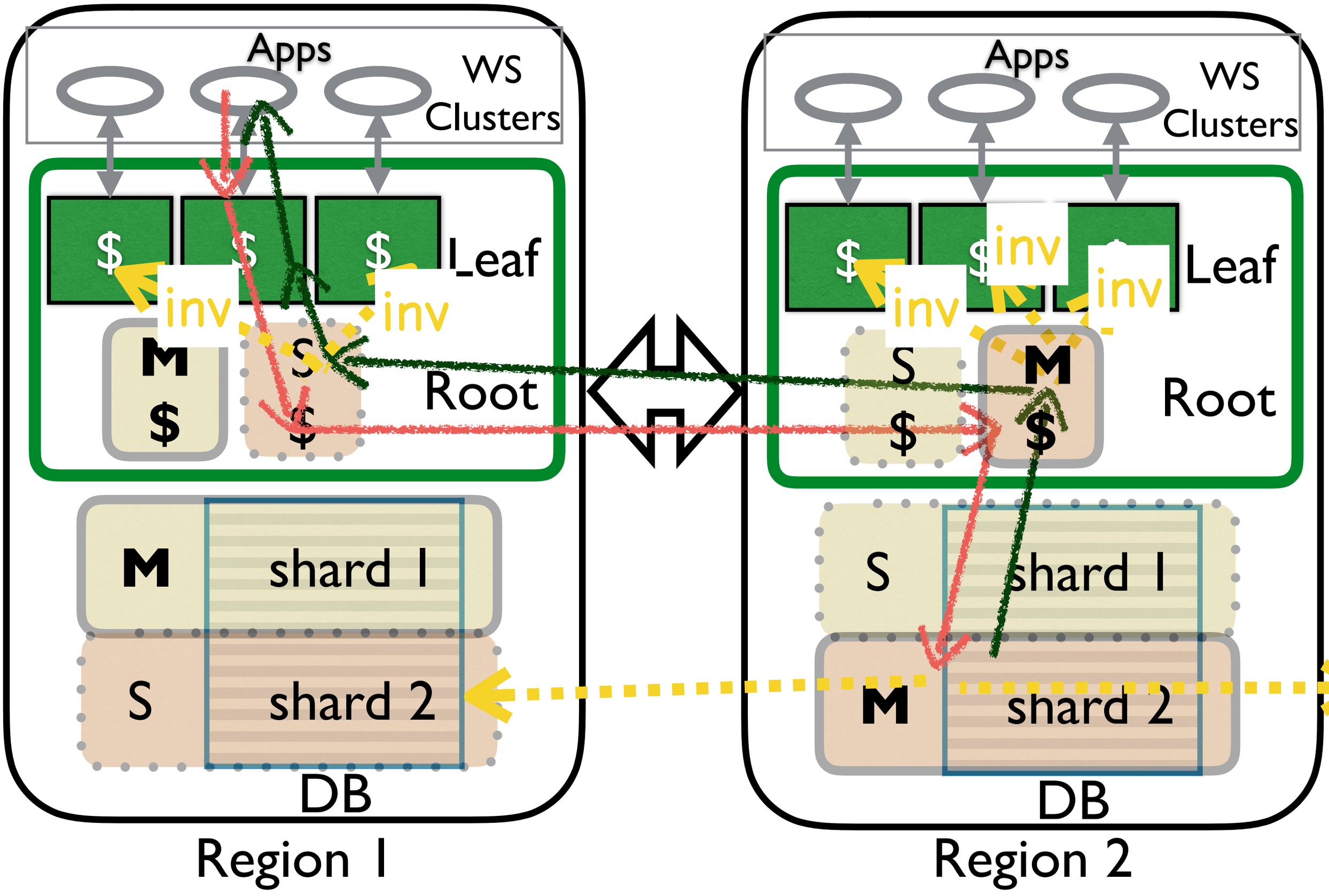
Writes


Apps
WS Clusters
$
$
$
MISS
Leaf
M
$
MISS
Root
M
shard 1
S
shard 2
DB
Region 1
Apps
WS Clusters
$
$
$
Leaf
S
$
M
$
Root
S
shard 1
M
shard 2
DB
Region 2

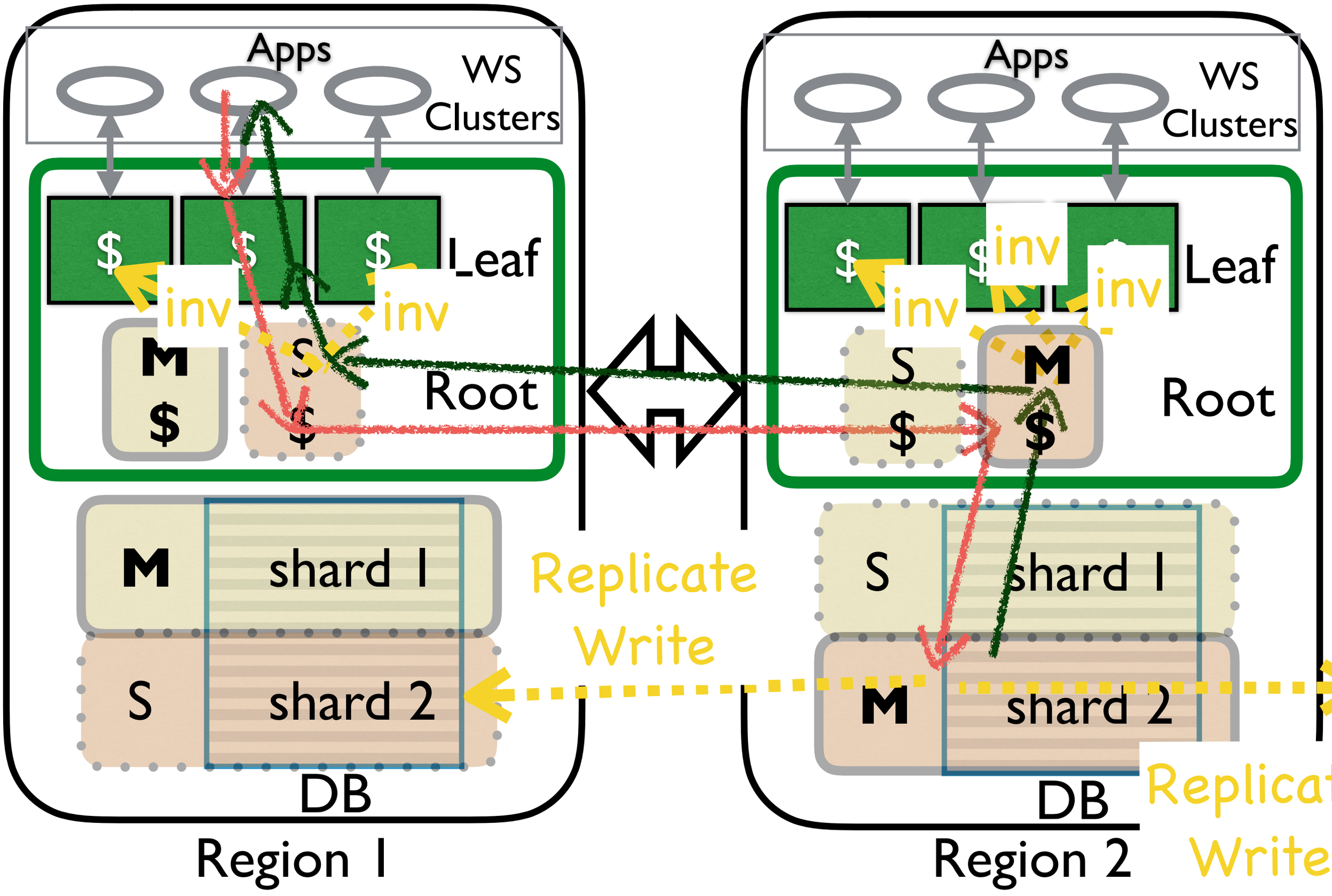
Writes


Apps
WS Clusters
$
Update
$
Leaf
M
$
Update
Root
$
S
$
M
$
Update
Root
M
shard 1
S
shard 2
DB
Region 1
Apps
WS Clusters
$
$
$
Leaf
S
$
M
$
Update
Root
S
shard 1
M
shard 2
DB
Region 2

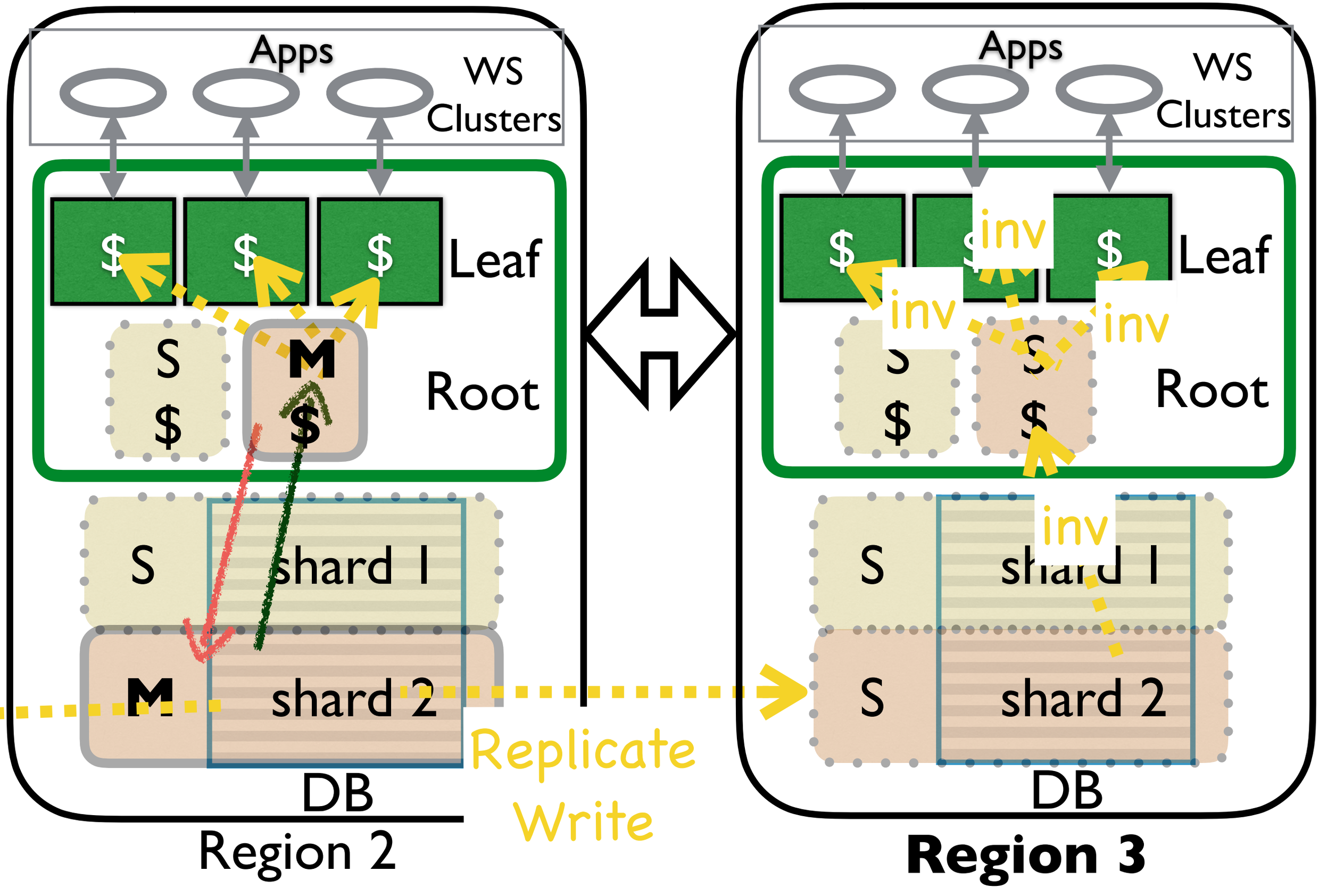
Writes


Apps
WS Clusters
$
Leaf
inv
inv
M
$
S
$
Root
shard 1
shard 2
S
M
DB
Region 1
Region 2

Writes


Apps
WS Clusters
$
Leaf
inv
inv
M
$
S
$
Root
shard 1
shard 2
Replicate Write
DB
Region 1
Apps
WS Clusters
$
inv
inv
Leaf
S
M
$
$
Root
S
shard 1
M
shard 2
DB
Region 2
Replicate Write

Writes


Apps
WS Clusters
$
$
$
Leaf
S
$
M
$
Root
S
shard 1
M
shard 2
DB
Region 2
Replicate Write
Apps
WS Clusters
$
inv
$
inv
inv
$
Leaf
S
$
S
$
Root
inv
S
shard 1
S
shard 2
DB
Region 3

TAO Consistency Model

- 2.2 :"per-object sequential consistency and read-after-write consistency within a cache [and thus shard] and eventual consistency across caches"
- per-object sequential consistency: like PNUTS timeline consistency — cache's may lag but never go backwards — master coordinates and orders all updates — eventual consistency never reorders — users never see data go backwards
- Read-after-write consistency: see your own write's — important for FB user experience

What is "within a cache"?

Guarantees only hold if app uses a single cluster if cluster fails and app switches, no guarantees

Weak Consistency Model

- Allows stale reads from caches (until invalidate arrives) — what would a linearizable design look like?
  - Single copy of each key eg. writes+reads all go to master DB — like PNUTS read-latest
  - caching, but write must *wait* for all invalidates to complete?
- What would the cost be? — Either much slower reads, or much slower writes

- KEY OBSERVATION:
  - Linearizability costs you on every operation — But may only make a difference in smallish window after each write

Why Consistency Checker?

- WANT — Linearizability but don't want to pay for it ;-)
  - So want to know how TAO makes out in practice
  - So check agains linearizability as well as their actual guarantees
- They want to systematically detect the kind of bugs they had with look-aside caching