Distributed Systems

Spring Semester 2020

Lecture 15: PNUTS

John Liagouris
liagos@bu.edu

Why this paper

- Massive Global scale system
- Performance + Flexible Consistency
- Still explicitly designed but less complicated than Google Spanner (also less powerful)

DATABASE FEATURES not KV store based
but uses a Pub Sub communications layer

The Big Picture

Multiple Datacenters in Regions

—

Entire Database replicated

Region 1

Region 2

Region 3

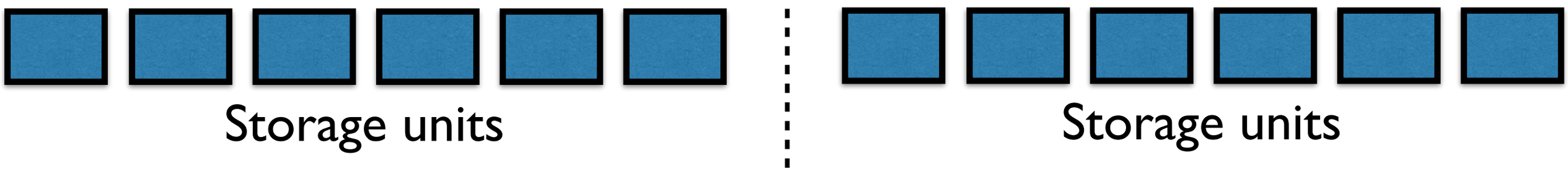
Replication

- GOAL: Make the reads fast!
  - Clients connect to local data center
    - Web app in that data center can do the reads fast — no remote communication
  - BUT reads possible stale
- TRADEOFF: Writes slow!

PNUTS

REGION 1

REGION 2

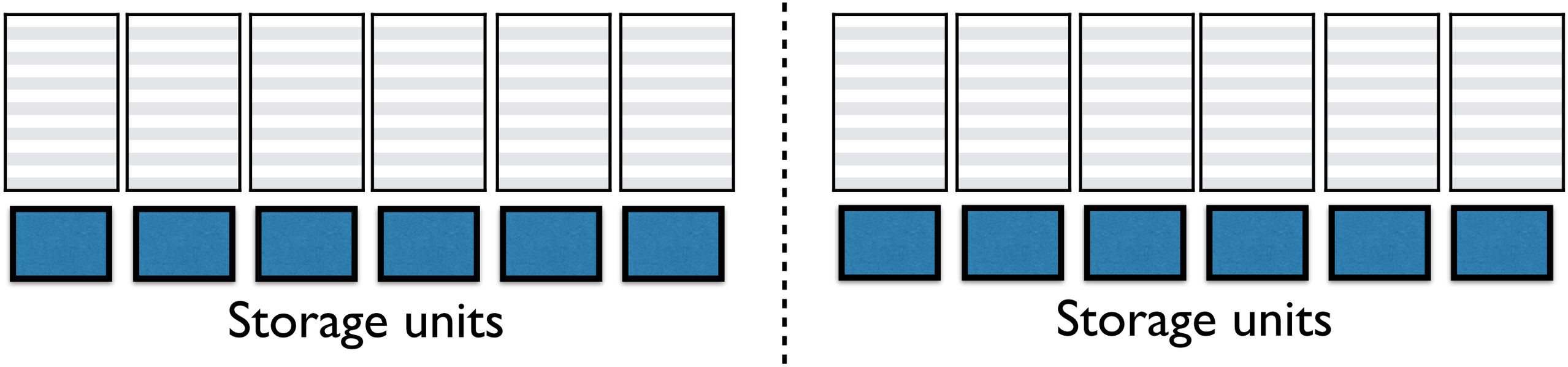


get(), set(), scan()

PNUTS

REGION 1

REGION 2

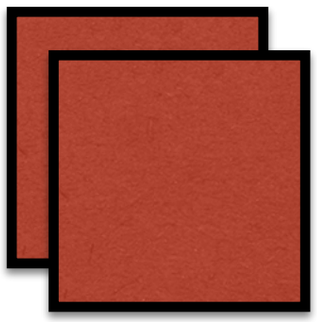


PNUTS

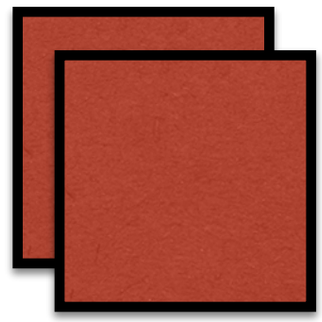
REGION 1

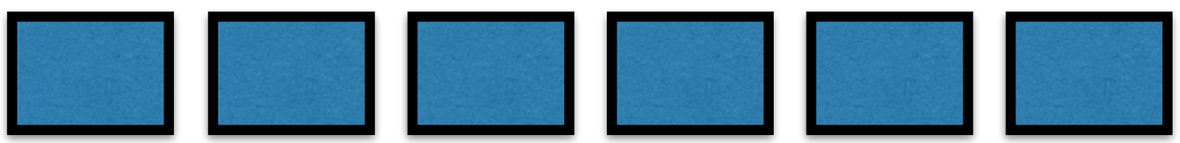
REGION 2

Tablet controller

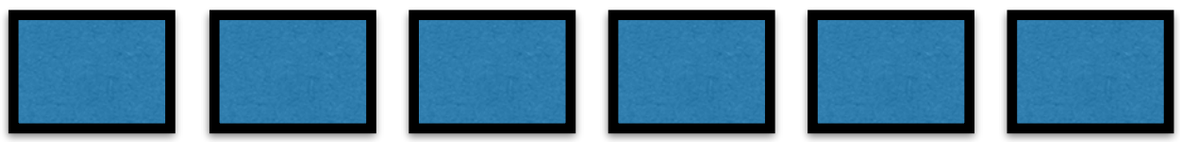


Tablet controller



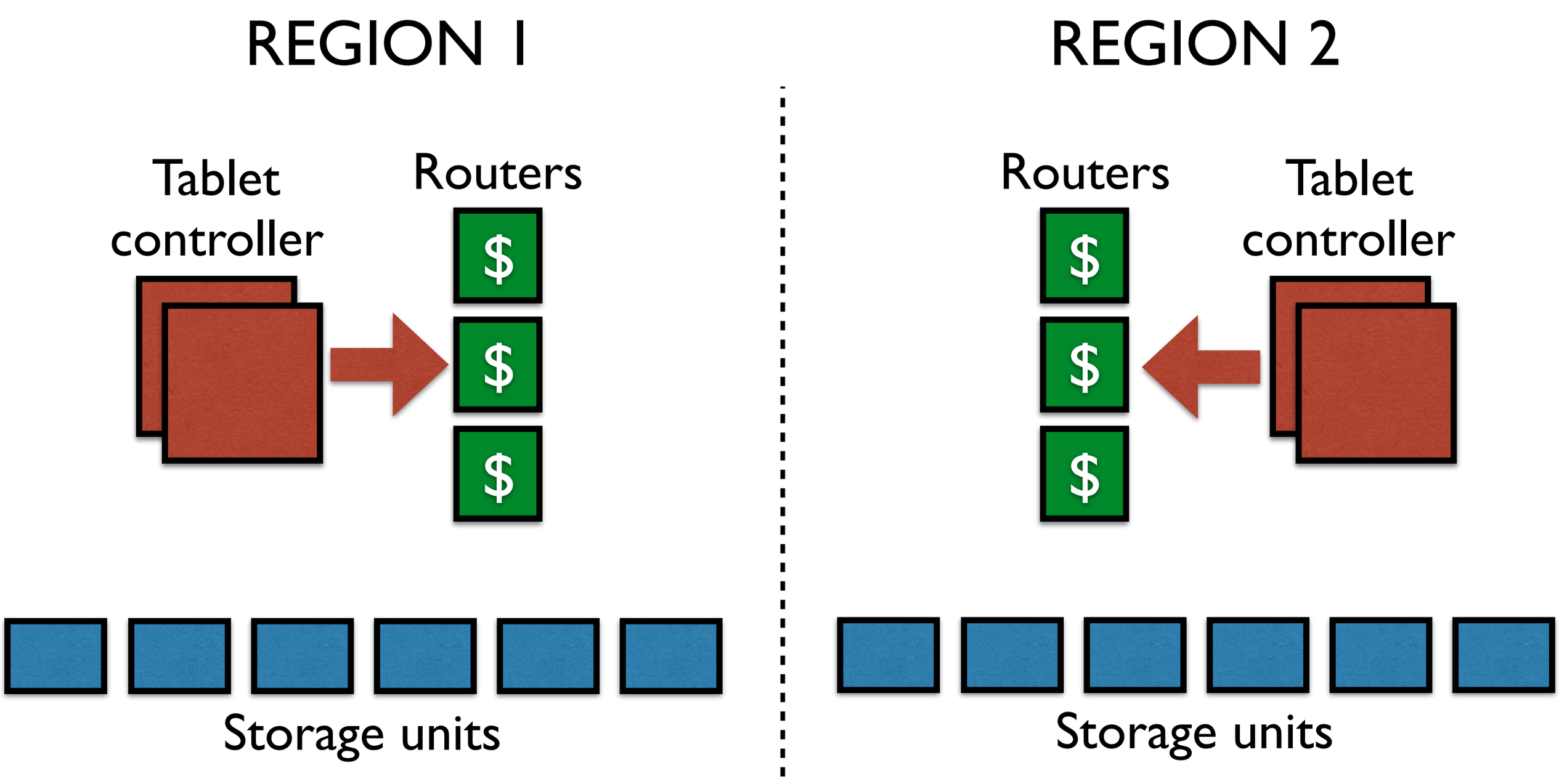


Storage units

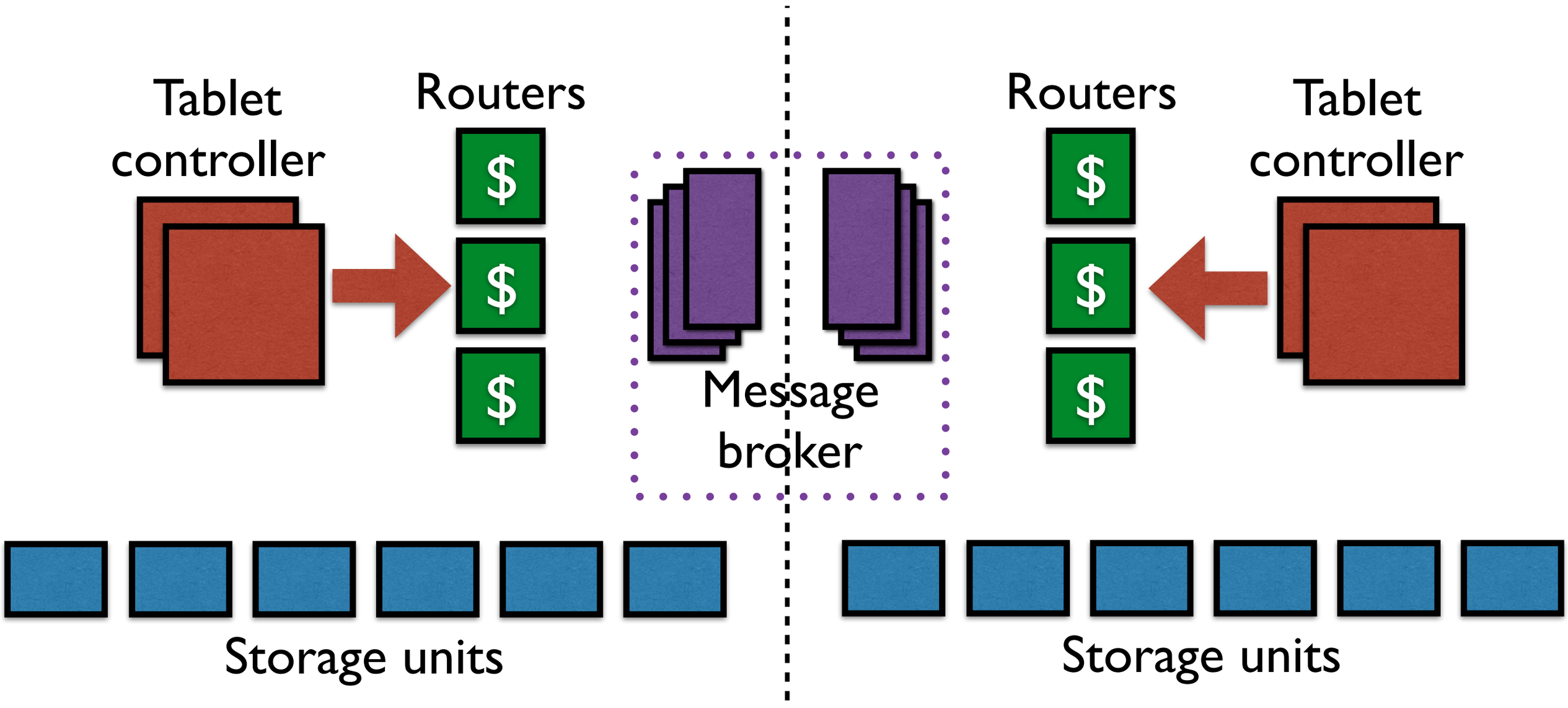


Storage units

PNUTS


REGION 1
REGION 2
Tablet controller
Routers
Routers
Tablet controller
$
$
$
$
$
$
Storage units
Storage units

PNUTS


REGION 1
REGION 2
Tablet controller
Routers
$
$
$
Message broker
Routers
$
$
$
Tablet controller
Storage units
Storage units

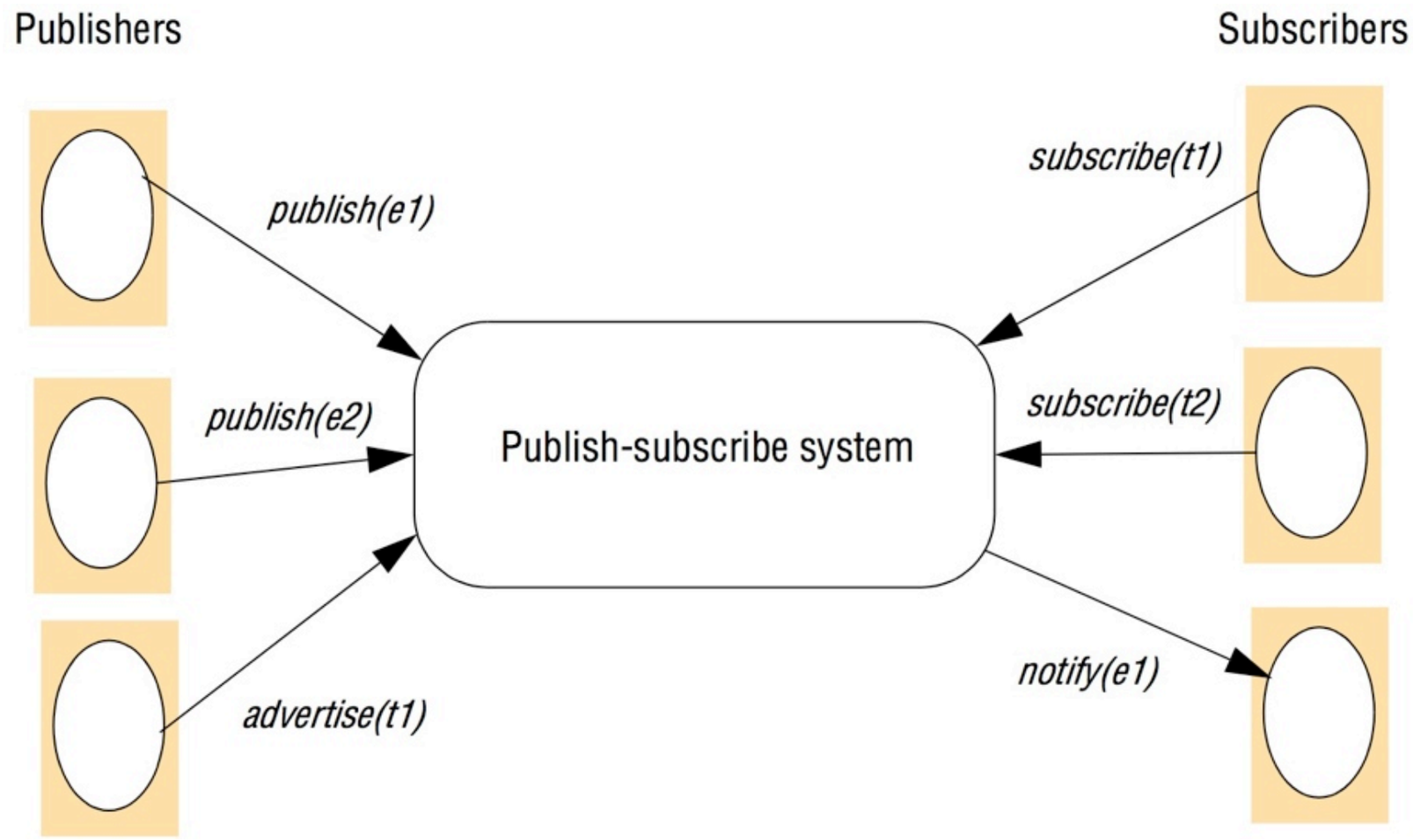
Publish-subscribe

Distributed Event Based Systems

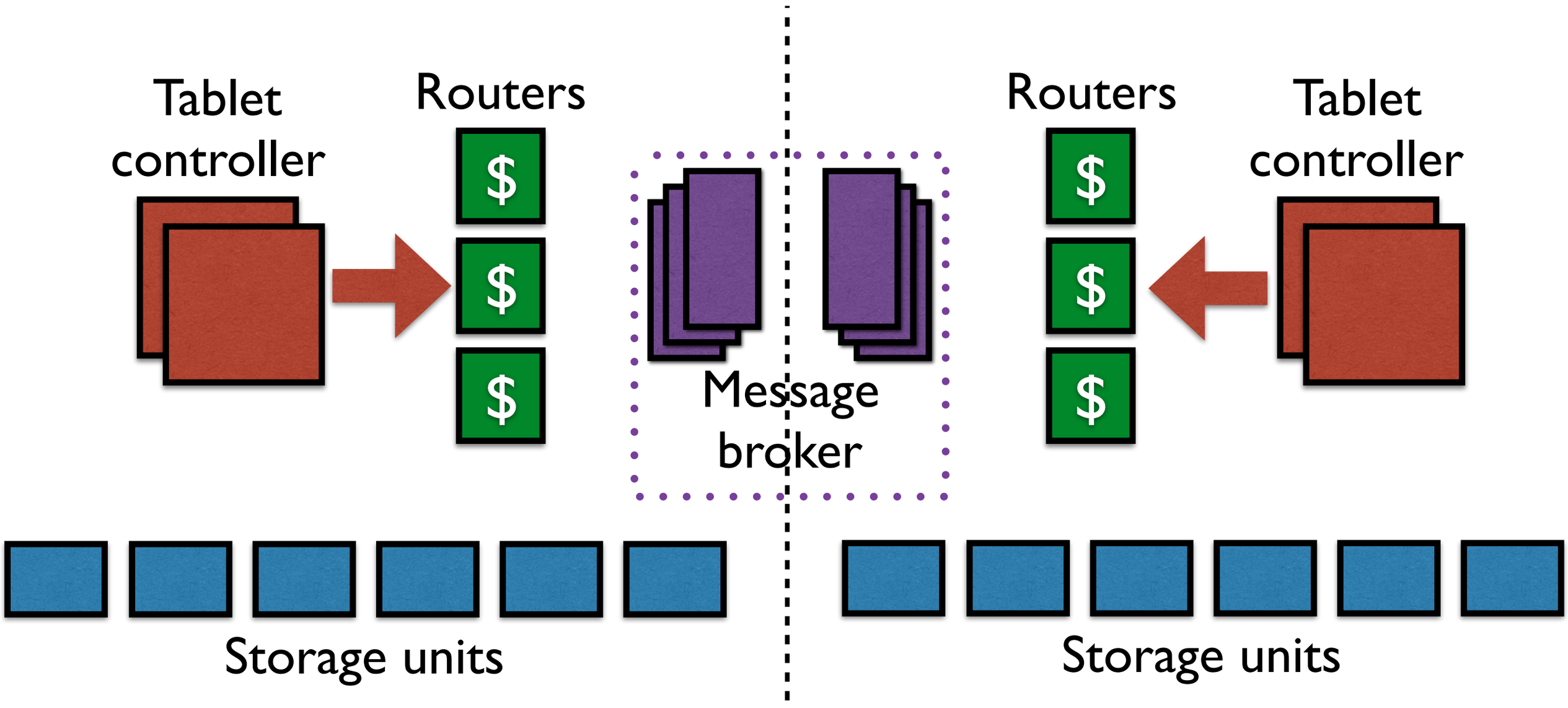
- "Decoupled and reactive style of programming"
- ***Publishers*** publish structured events to an event service
- ***Subscribers*** subscribe to particular events through subscriptions
- System must match subscriptions against published events and ensure the correct delivery of ***event notifications***
- ***one-to-many*** paradigm of communications
- Asynchronous
- ***Delivery guarantees:*** reliability, agreement, latency, etc.

Publish-subscribe

Distributed Event Based Systems



PNUTS


REGION 1
REGION 2
Tablet controller
Routers
$
$
$
Message broker
Routers
$
$
$
Tablet controller
Storage units
Storage units

Updates

REGION 1

webserver

App
r/w to PNUTS

Tablet

Routers

controller

$

$

$

Message broker

REGION 2

Routers

Tablet

controller

$

$

$

Storage units

Storage units

Updates

REGION 1

REGION 2

Why not just have app logic send update to every region?

Tablet

Routers

Routers

Tablet

controller

$

$

$

Message broker

$

$

$

controller

Storage units

Storage units

Updates

REGION 1

REGION 2

What if app crashes after updating only some regions?
What if concurrent updates to same record?

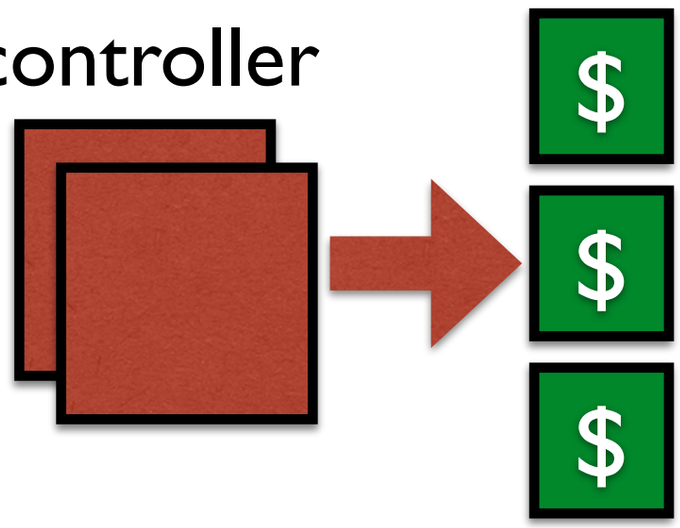
Tablet

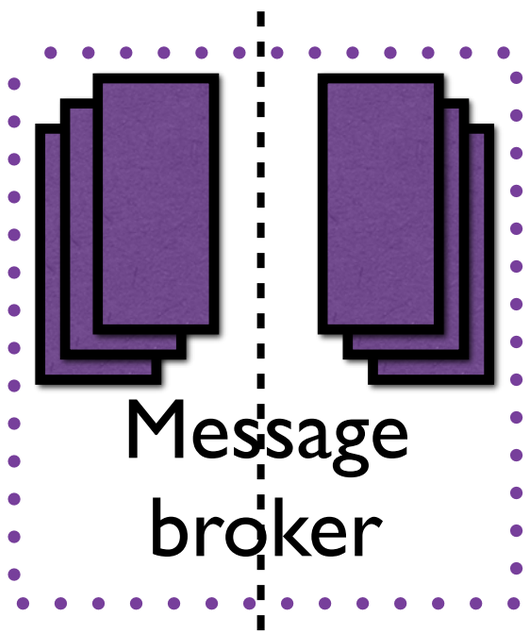
Routers

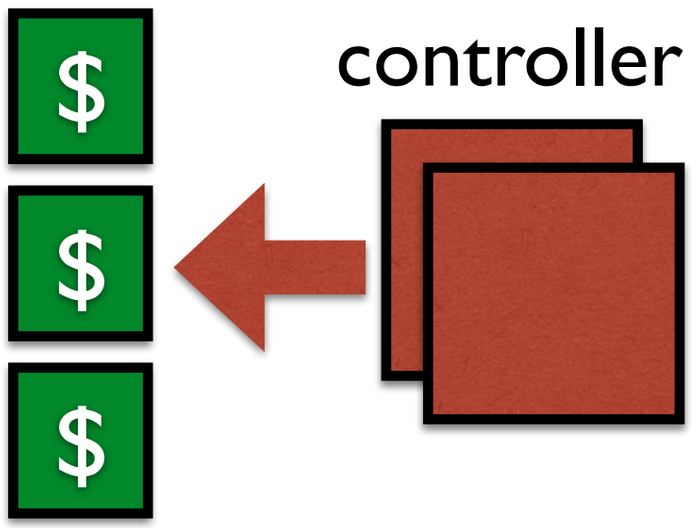
Routers

Tablet

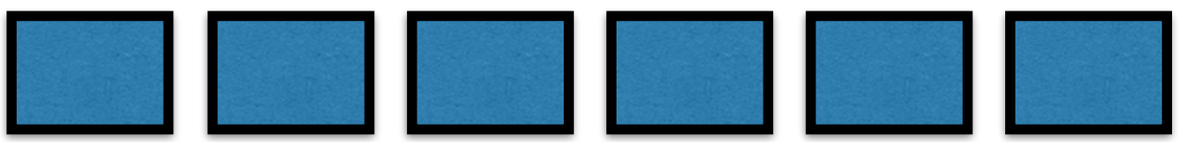
controller



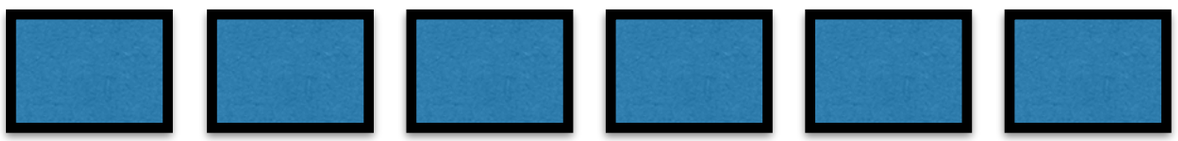




controller



Storage units



Storage units

Updates

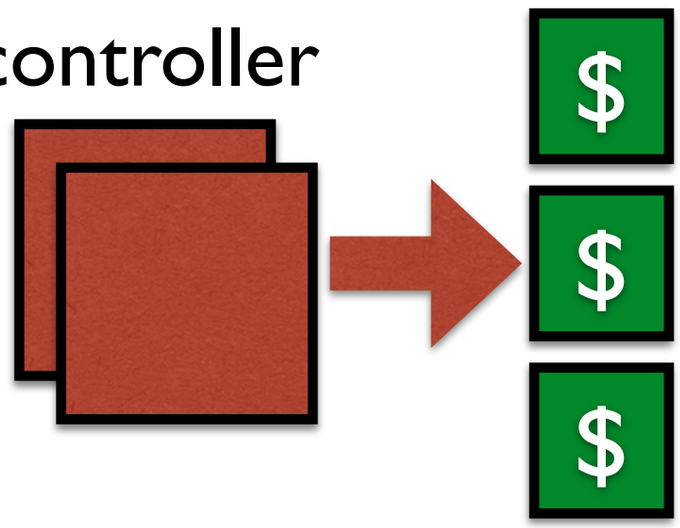
REGION 1

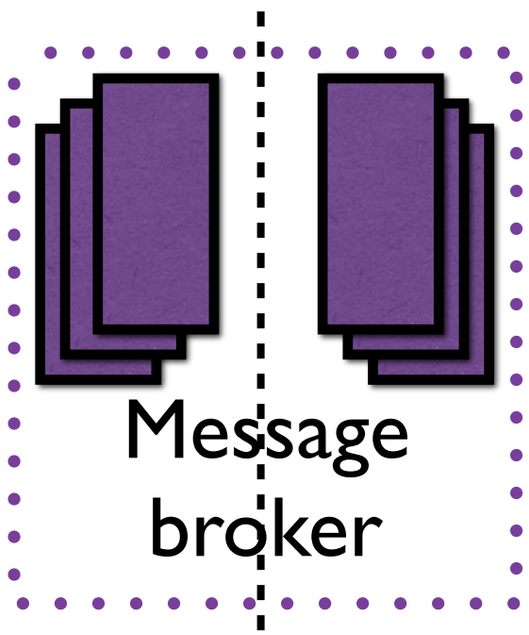
REGION 2

So what does PNUTs do?

Tablet controller

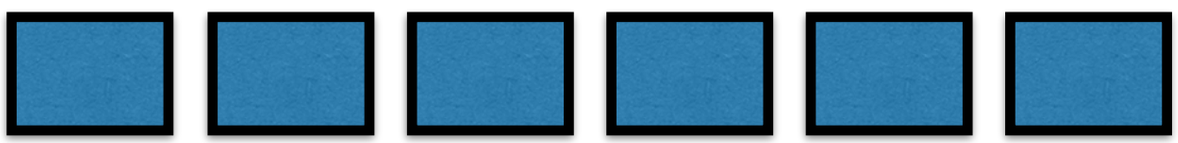
Routers



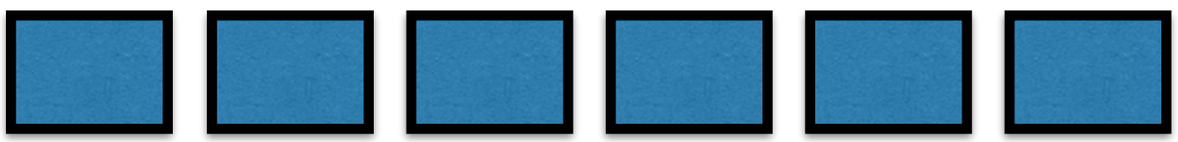


Routers

Tablet controller



Storage units



Storage units

Record based Mastering

- PNUTS has a "record master" (region) for each record
  - All updates are directed to master
    - Master imposes order on writes for each record
    - Responsible storage unit executes updates one at a time per record
    - Tells MB to broadcast updates in order to all regions

Two cases:

1. Update occurs at master Region
2. Update occurs at non-master Region

What is the difference?

Two cases:

1. Update occurs at master Region
2. Update occurs at non-master Region

What is the difference?

PERFORMANCE

So what is going on

- Why per-record master? — consider table of shopping carts — updates have locality
- PNUTs master is dynamic each row has hidden column that indicates master region — can migrate
- Hopefully get best of both worlds:
  - Fast reads of local data
  - Fast writes b/c master is often the local region

Update Walk Through

REGION 1

REGION 2

App
w to PNUTS

Tablet

Routers

Routers

Tablet

controller

$

$

$

Message
broker

$

$

$

controller

Storage units

Storage units

Update Walk Through

REGION 1

REGION 2

App
w to PNUTS

Tablet

Routers

Routers

Tablet

controller

$

$

$

Message
broker

$

$

$

controller

Storage units

Storage units

Update Walk Through

REGION 1

REGION 2

App
w to PNUTS

Tablet

Routers

Routers

Tablet

controller

$

$

$

Message
broker

$

$

$

controller

Storage units

Storage units

Update Walk Through

REGION 1

REGION 2

App
w to PNUTS

Tablet

Routers

Routers

Tablet

controller

$

$

$

Message
broker

$

$

$

controller

Storage units

Storage units

Update Walk Through

REGION 1

REGION 2

App
w to PNUTS

Tablet

Routers

Routers

Tablet

controller

$

$

$

Message
broker

$

$

$

controller

Storage units

Storage units

Update Walk Through

REGION 1

REGION 2

App
w to PNUTS

Tablet

Routers

Routers

Tablet

controller

$

$

$

Message
broker

$

$

$

controller

Storage units

Storage units

Update Walk Through

REGION 1

REGION 2

App
w to PNUTS

Tablet

Routers

Routers

Tablet

controller

$

$

$

Message
broker

$

$

$

controller

Storage units

Storage units

But for this to be useful what do we need from the Message Brokers

But for this to be useful what do we need from the Message Brokers

Async and Reliable Communication

Message Broker : Cool Idea

- Reliable: logs to disk, keeps trying, to cope with various failures
- Ordered: record's replicas eventually identical even w/ multiple writers
- Async: apps don't wait once write committed at MB

Encapsulates all kinds of the tricky DS stuff — App's and DB don't have to deal with it

Write Order

REGION 1

REGION 2

USER	WHERE	WHAT
Alice	Home	Asleep
Bob	Gym	Running
…		
…		

USER	WHERE	WHAT
Alice	Home	Asleep
Bob	Gym	Running
…		
…		

Write Order

REGION 1

Alice App:

1.write(WHAT=Awake)

2.write(WHERE=Work)

USER	WHERE	WHAT
Alice	Home	Asleep
Bob	Gym	Running
…		
…		

REGION 2

User in another Region following Alice : Local Reads

USER	WHERE	WHAT
Alice	Home	Asleep
Bob	Gym	Running
…		
…		

Write Order

REGION 1

Alice App:

1. write(WHAT=Awake)
2. write(WHERE=Work)

USER	WHERE	WHAT
Alice	Work	Awake
Bob	Gym	Running
…		
…		

REGION 2

might observe:

Home/Asleep

Home/Awake

Work/Awake

~~Work/Asleep~~

USER	WHERE	WHAT
Alice	Home	Awake
Bob	Gym	Running
…		
…		

Only single record ordering of updates and transactions

e.g.

Write(Bob.What, on duty)
Write(Alice.What, off duty)

Why is it OK for writes to take effect slowly?

- fundamental benefit: reads are then very fast, since local and reads usually greatly outnumber writes
- PNUTS mitigates write cost:
  - application waits for MB commit but not propagation (“asynchronous")
  - master likely to be local (they claim 80% of the time)
  - so MB commit will often be quick
- down side: readers at non-master regions may see stale data

How stale might a non-master record be?

- Depends on how quickly MB sends updates to regions
- Guess: less than a second usually
- Longer if network slow/flaky, or MB busy

So can and how do we deal with Stale data?

- Sometimes stale is ok: looking at other people's profiles
- Sometimes stale is not ok: shopping cart after add/delete item

The App is the Boss and needs to decide

So can and how do we deal with Stale data?

Application gets to choose how consistent (section 2.2)

- read-any(k) : read from local SU, fast but maybe stale
- read-critical(k, required_version): fast local read if local SU has vers >= required_version, otherwise slow read from master SU. Why: app knows it just wrote, wants value to reflect write
- read-latest(k): always read from master, slow if master is remote! Why: app needs fresh data

- What if app needs to increment a counter stored in a record?
- Is read-latest(k), increment, then write(k) enough?

- What if app needs to increment a counter stored in a record?
  - Is read-latest(k), increment, then write(k) enough?

Not if there might be concurrent updates!

Atomic Operation

`test-and-set-write(version#, new value)`

Atomic update to one record

master rejects the write if current version # != version#

so if concurrent updates, one will lose and retry

```
while(1):
    (x, ver) = read-latest(k)
    if(t-a-s-w(k, ver, x+1))
        break
```

t-a-s-w is fully general for single-record atomic read-modify-write