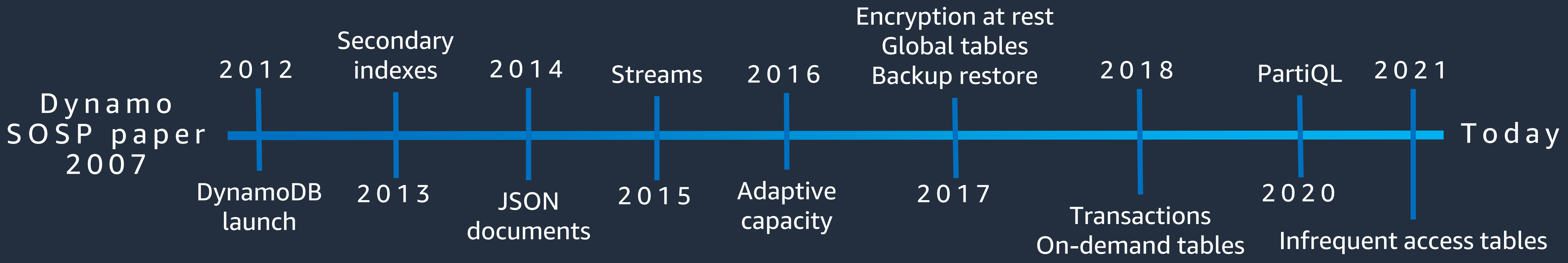
Amazon DynamoDB: A Scalable, Predictably Performant, Fully Managed NoSQL Database Service

*Mostafa Elhemali, Niall Gallagher, Nicholas Gordon,
Joseph Idziorek, Richard Krog, Colin Lazier, Erben Mo,
Akhilesh Mritunjai, Somu Perianayagam, Tim Rath,
Swami Sivasubramanian, James Christopher Sorenson III,
Sroaj Sosothikul, Doug Terry, Akshat Vig*

Proceedings of the 2022 USENIX Annual Technical Conference

DynamoDB over the years



Dynamo
SOSP paper
2007

2012
DynamoDB
launch

Secondary
indexes
2013

2014
JSON
documents

Streams
2015

2016
Adaptive
capacity

Encryption at rest
Global tables
Backup restore
2017

2018
Transactions
On-demand tables

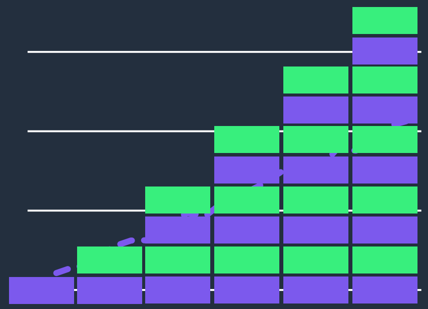
PartiQL
2020

2021
Infrequent access tables

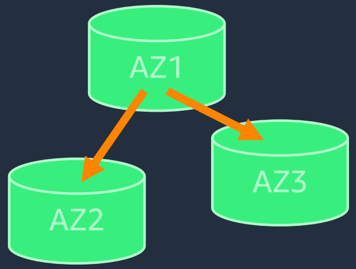
Today

Key Aspects of DynamoDB

Predictability



Scalability

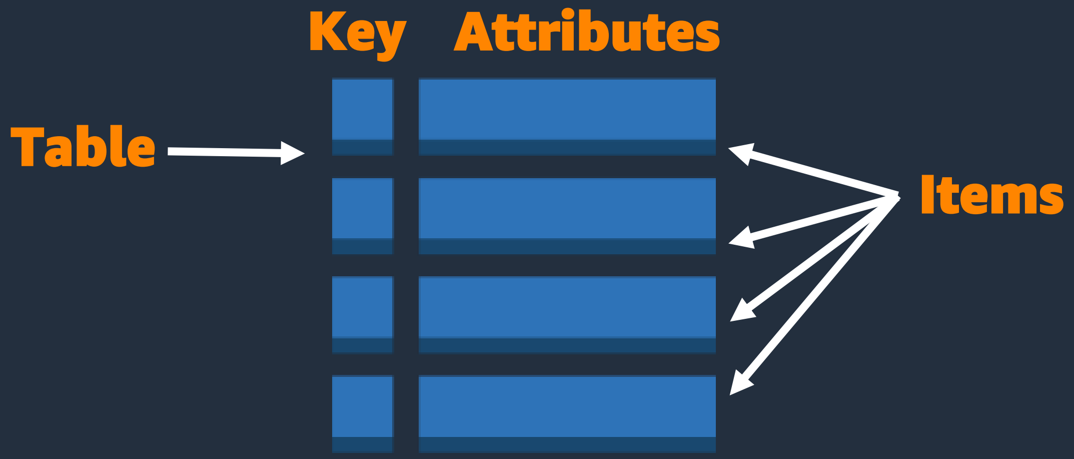


Availability

Consistency

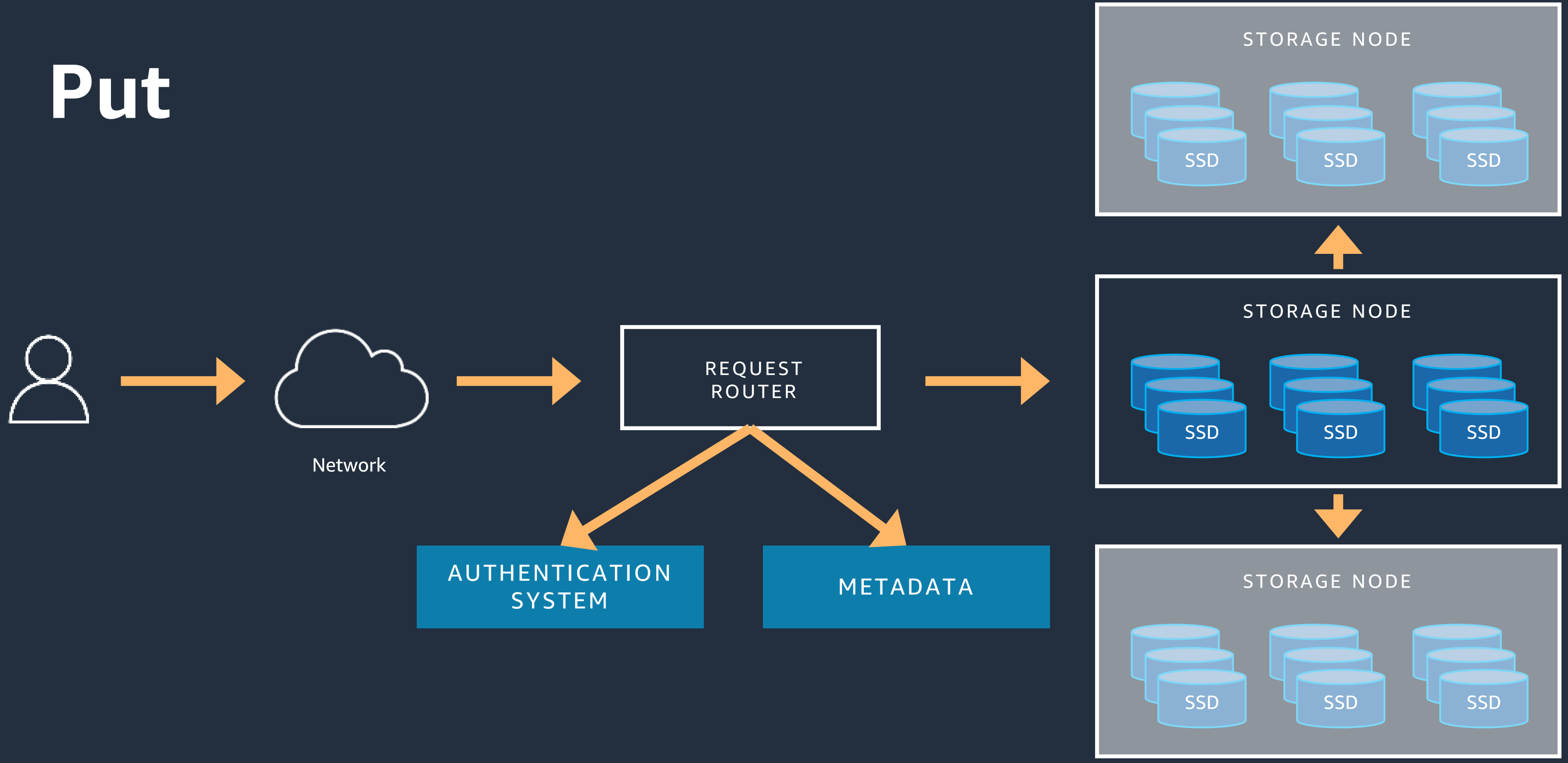
Predictability

DynamoDB is a Key-Value Store

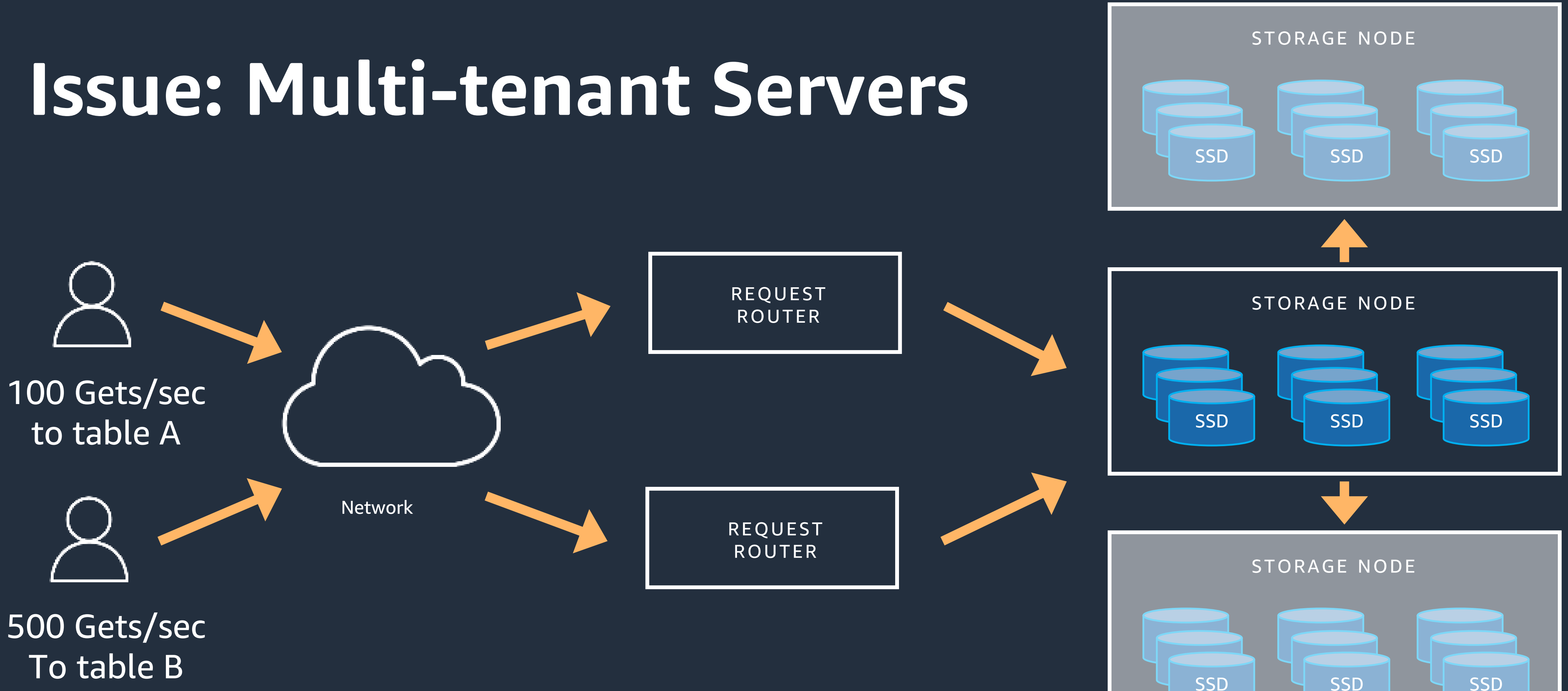


Operations: Get, Put, Update, Delete, …

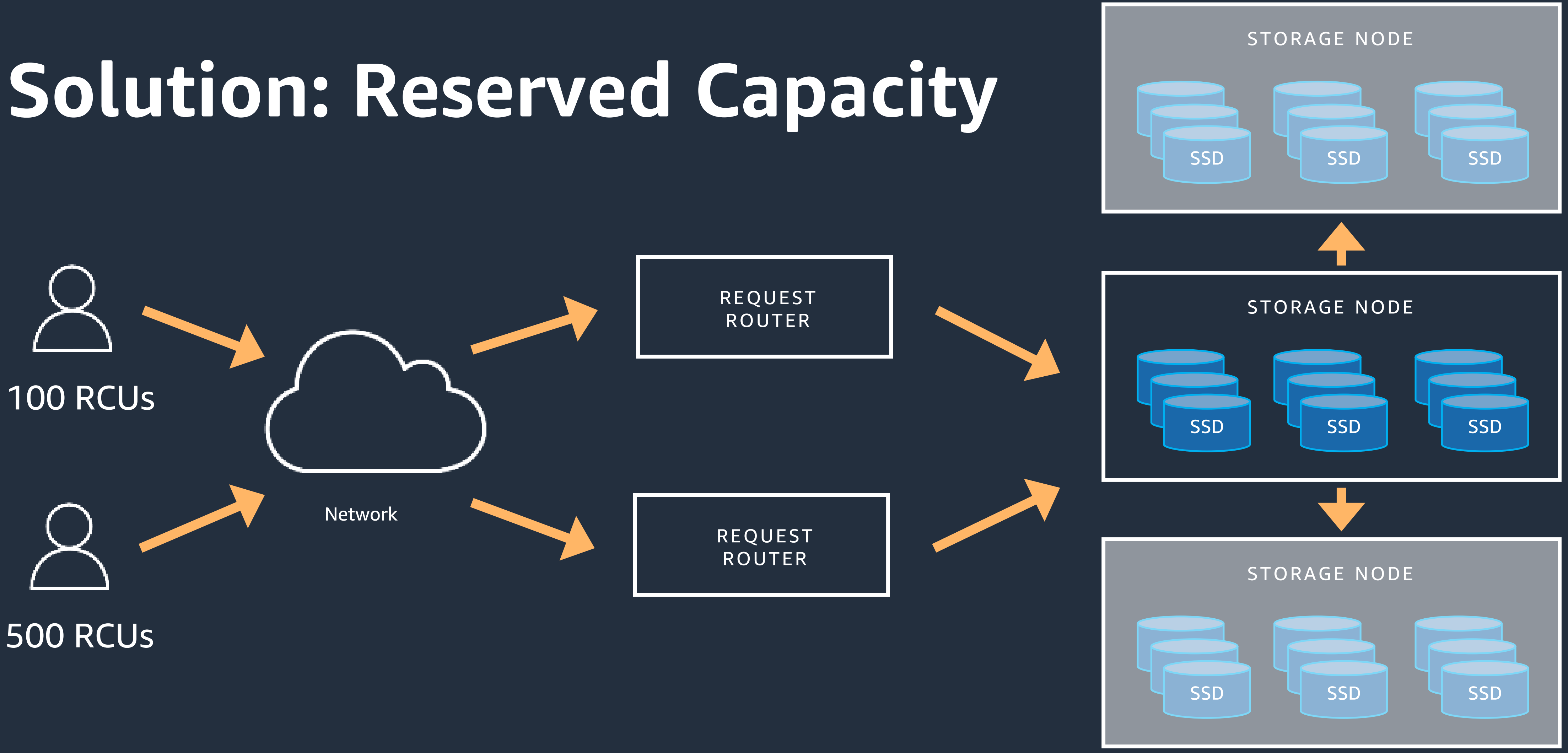
Put


Network
REQUEST ROUTER
AUTHENTICATION SYSTEM
METADATA
STORAGE NODE
SSD
SSD
SSD
STORAGE NODE
SSD
SSD
SSD
STORAGE NODE
SSD
SSD
SSD

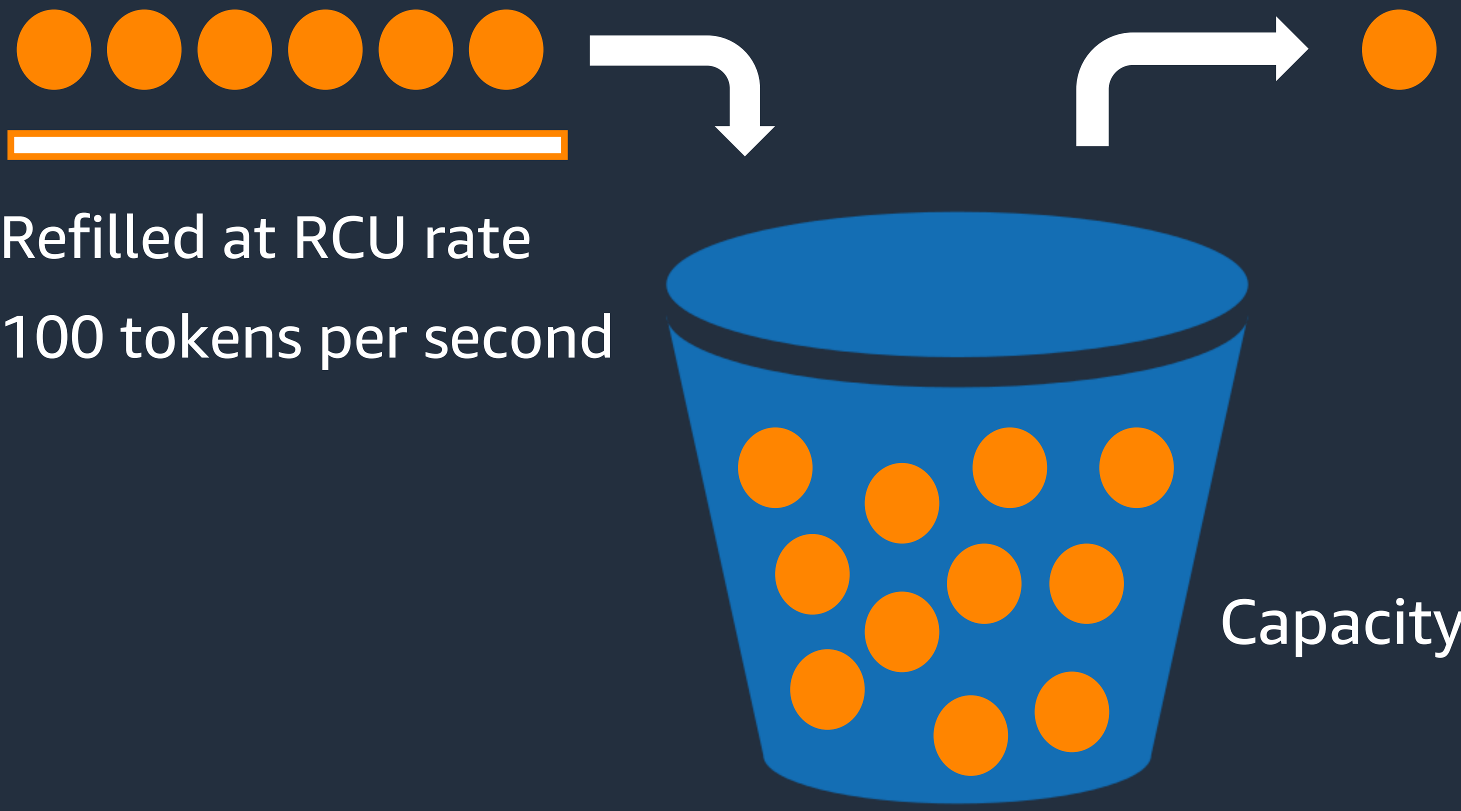
Issue: Multi-tenant Servers



Solution: Reserved Capacity



Token bucket algorithm



Refilled at RCU rate

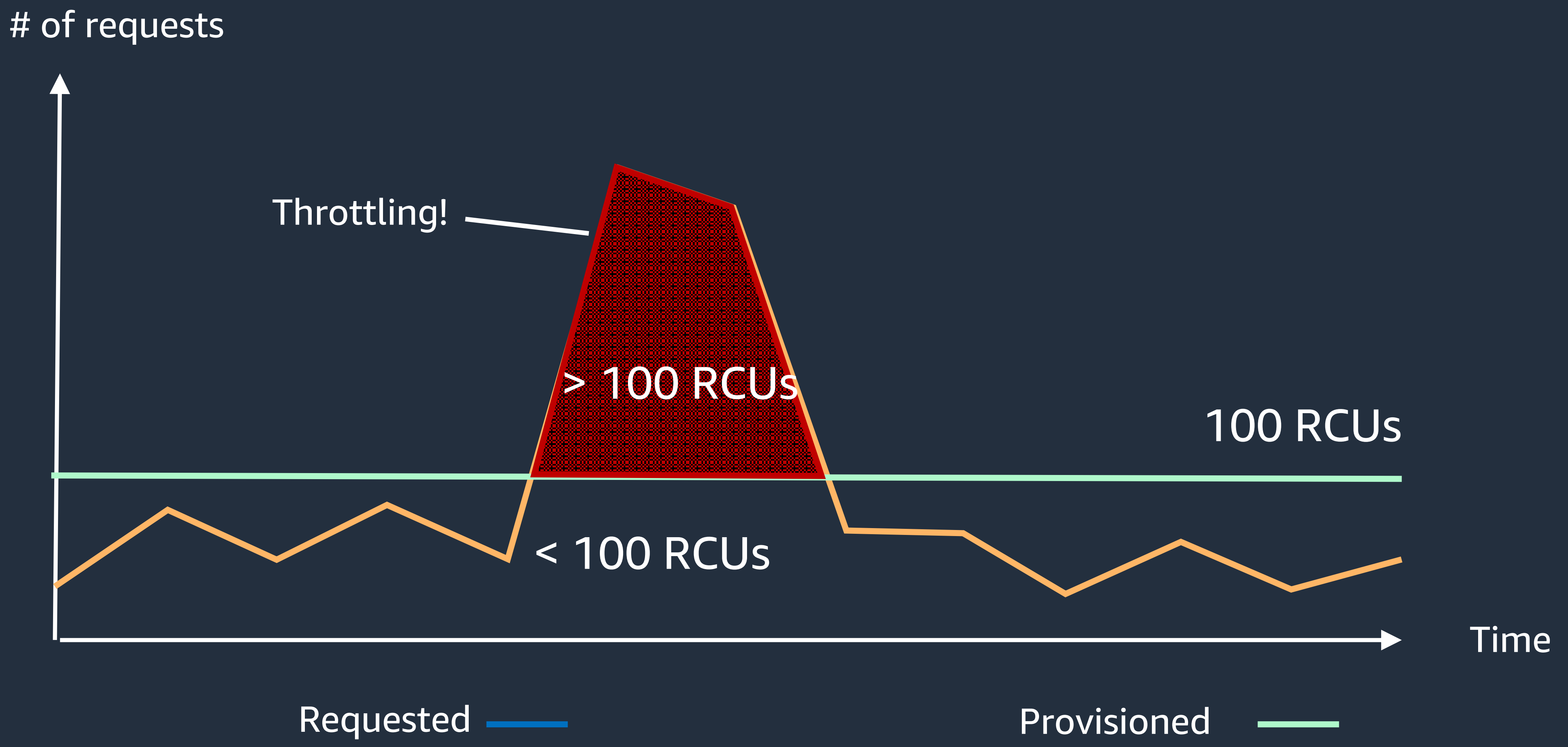
100 tokens per second

Emptied 1 token per request*

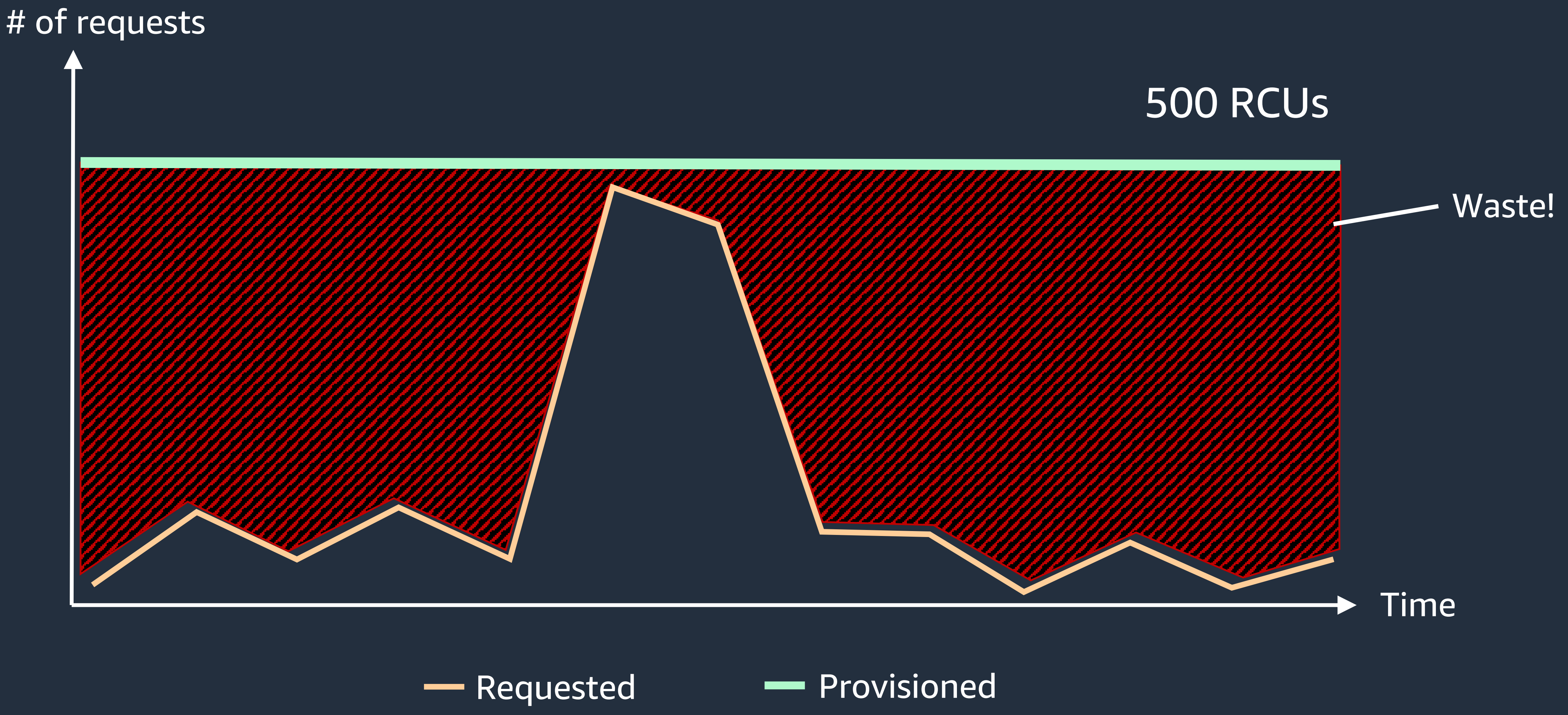
*Tokens deducted depends on item size and consistency

Capacity = 100

Problem: Non-uniform request distribution over time

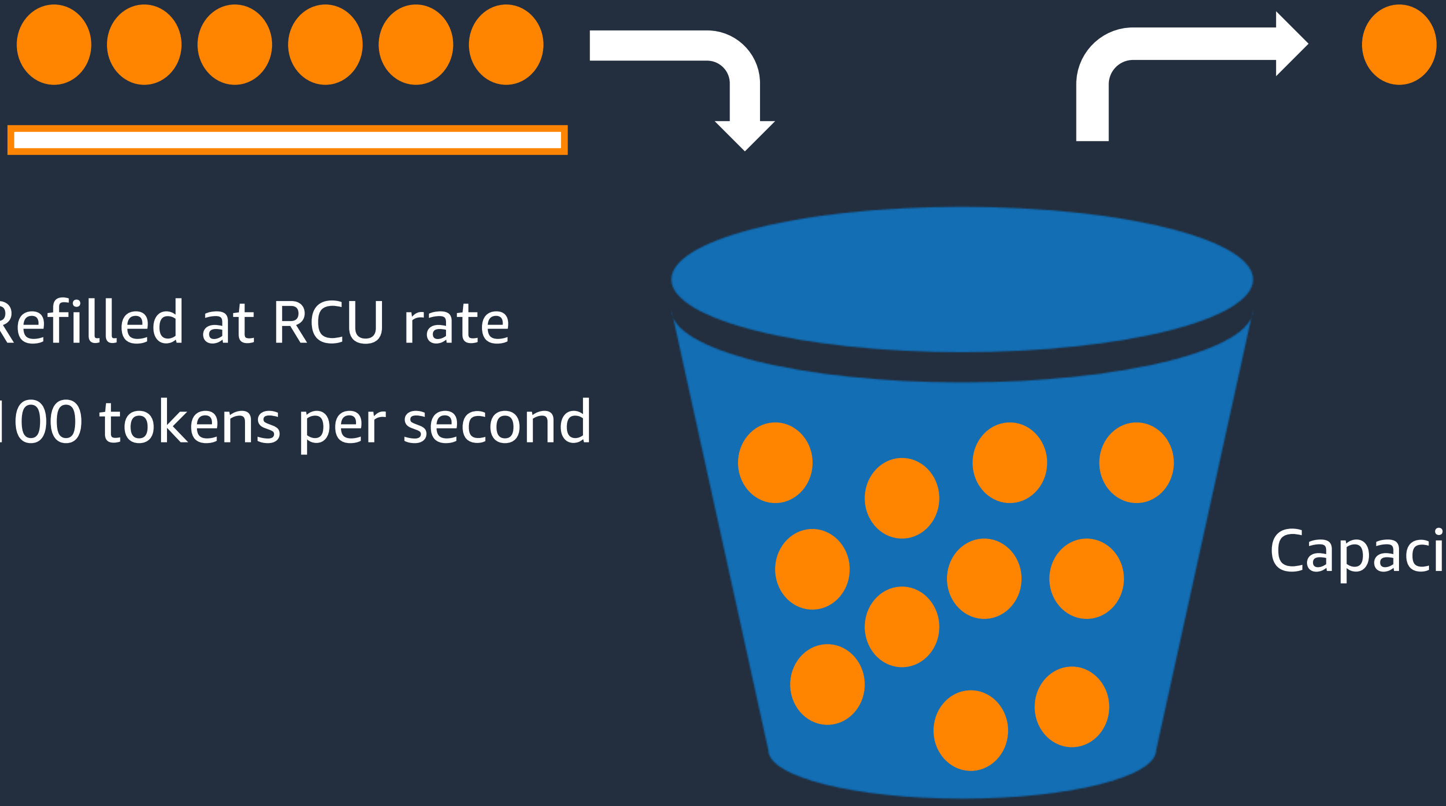


Common solution: Over-provisioning



Amazon Confidential

Bursting

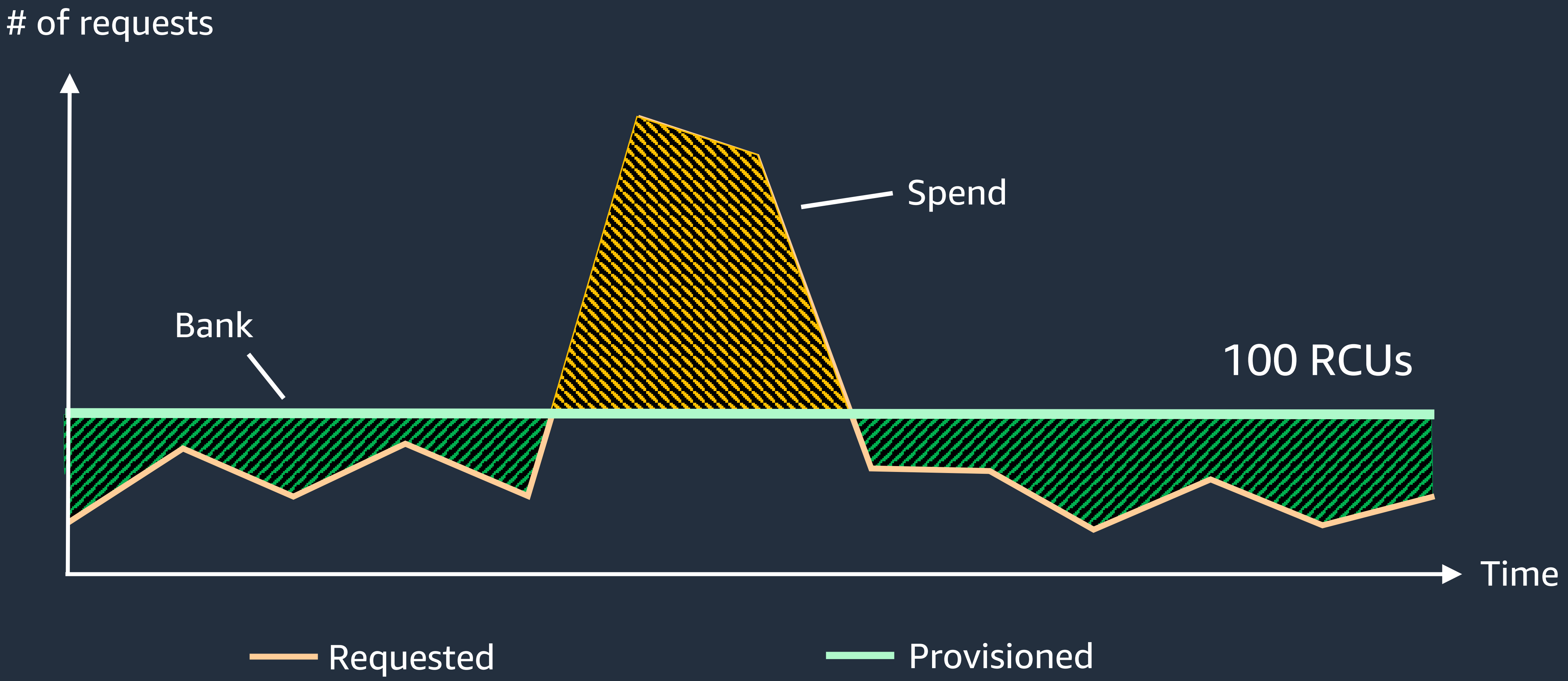


Refilled at RCU rate

100 tokens per second

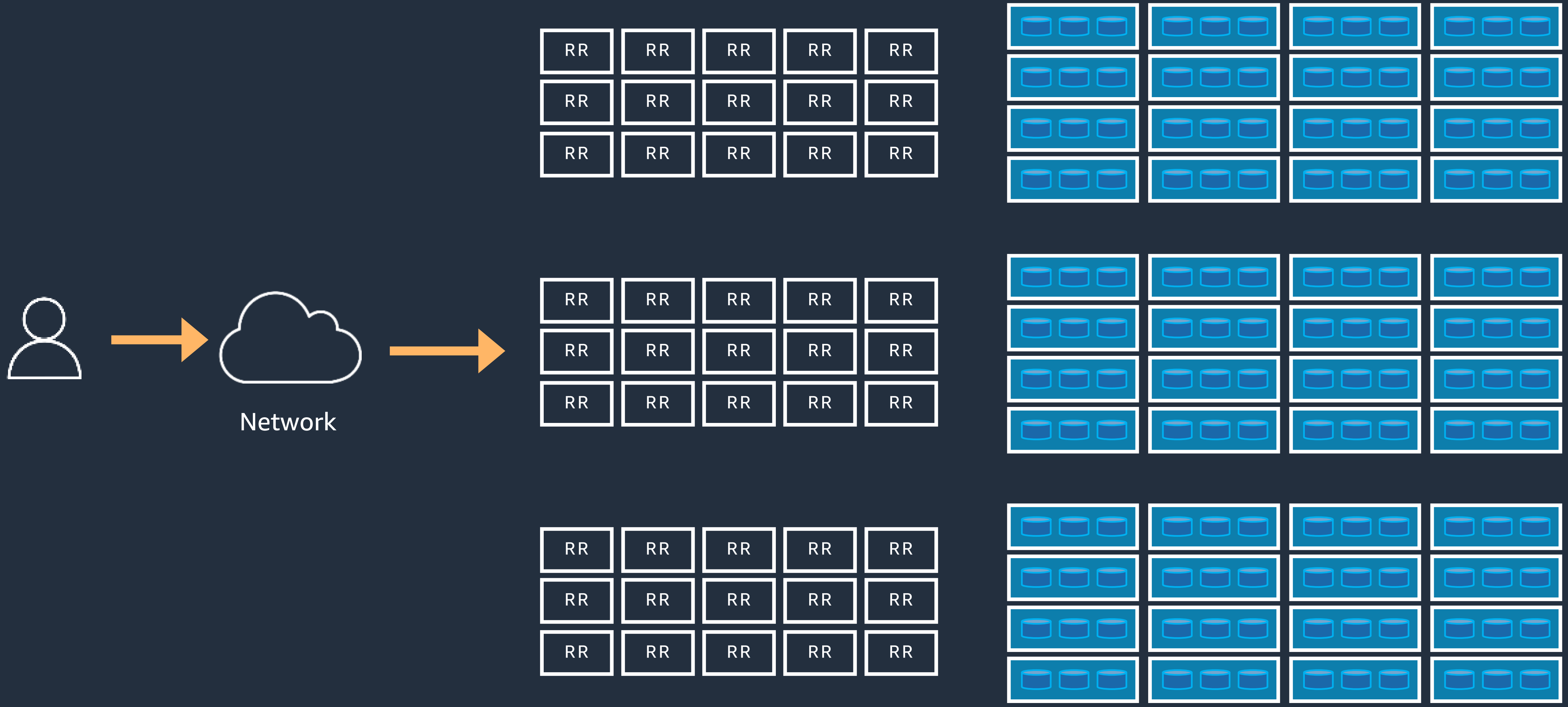
Capacity = 300 * RCUs or 3000

Bursting


# of requests
Spend
Bank
100 RCUs
Time
Requested
Provisioned

Scalability

Service at scale



Table

CustID	Customer information
145783	{ name:“Bob”, city:”London”, …}
236294	{ name:“Sara”, city:”Tampa”, …}
333363	{ name:“Betty”, city:”Madison”, …}
445104	{ name:“James”, city:”Miami”, …}
523422	{ name:“Alex”, city:”London”, …}
643145	{ name:“Val”, city:”Seattle”, …}
723342	{ name:“Jeff”, city:”Toledo”, …}

Hashing

Hash Value	CustID	Customer Information
0x9531	145783	{ name:“Bob”, city:”London”, …}
0x12A8	236294	{ name:“Sara”, city:”Tampa”, …}
0x6134	333363	{ name:“Betty”, city:”Madison”, …}
0x3391	445104	{ name:“James”, city:”Miami”, …}
0xF355	523422	{ name:“Alex”, city:”London”, …}
0xB082	643145	{ name:“Val”, city:”Seattle”, …}
0xEA8A	723342	{ name:“Jeff”, city:”Toledo”, …}

Partitioning

Hash Value	CustID	Customer Information
0x9531	145783	{ name:“Bob”, city:”London
0x12A8	236294	{ name:“Sara”, city:”Tampa”, …}
0x6134	333363	{ name:“Betty”, city:”Madis
0x3391	445104	{ name:“James”, city:”Miam
0xF355	523422	{ name:“Alex”, city:”London”, …}
0xB082	643145	{ name:“Val”, city:”Seattle”,
0xEA8A	723342	{ name:“Jeff”, city:”Toledo”

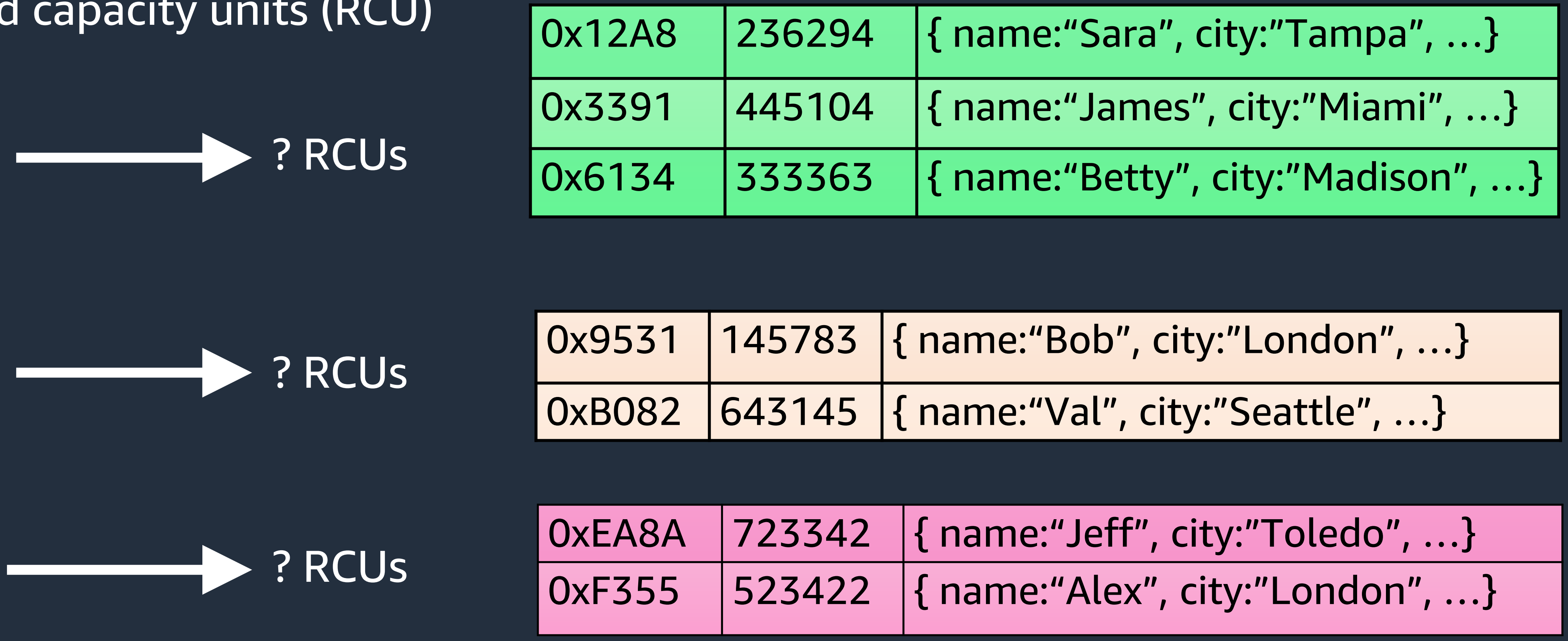
0x12A8	236294	{ name:“Sara”, city:”Tampa”, …}
0x3391	445104	{ name:“James”, city:”Miami”, …}
0x6134	333363	{ name:“Betty”, city:”Madison”, …}

0x9531	145783	{ name:“Bob”, city:”London”, …}
0xB082	643145	{ name:“Val”, city:”Seattle”, …}

0xEA8A	723342	{ name:“Jeff”, city:”Toledo”, …}
0xF355	523422	{ name:“Alex”, city:”London”, …}

Provisioning

300 read capacity units (RCU)



? RCUs

0x12A8	236294	{ name:“Sara”, city:”Tampa”, …}
0x3391	445104	{ name:“James”, city:”Miami”, …}
0x6134	333363	{ name:“Betty”, city:”Madison”, …}

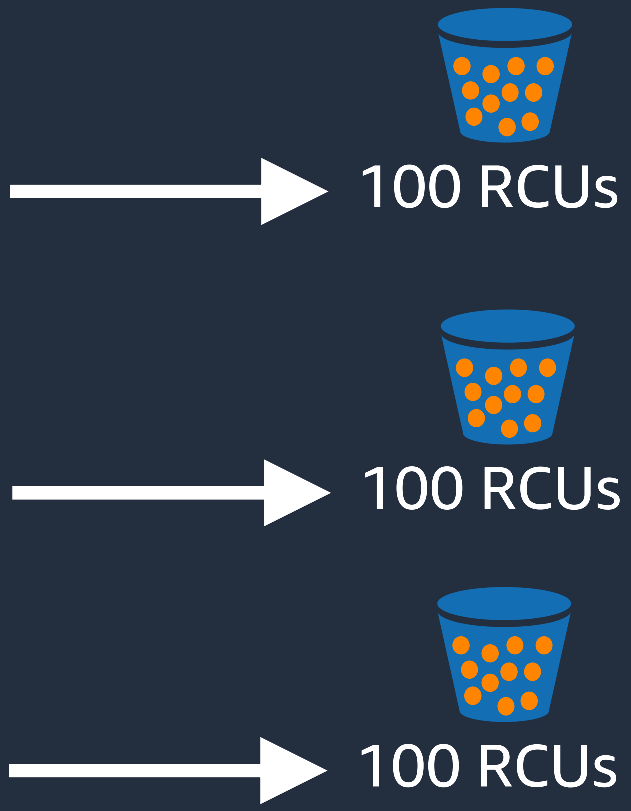
? RCUs

0x9531	145783	{ name:“Bob”, city:”London”, …}
0xB082	643145	{ name:“Val”, city:”Seattle”, …}

? RCUs

0xEA8A	723342	{ name:“Jeff”, city:”Toledo”, …}
0xF355	523422	{ name:“Alex”, city:”London”, …}

Provisioning



100 RCUs

100 RCUs

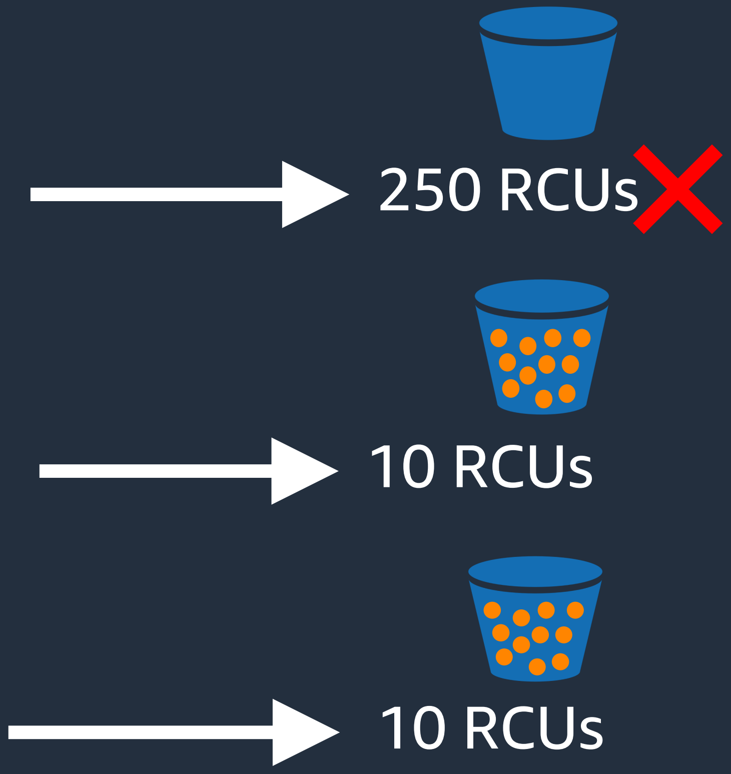
100 RCUs

0x12A8	236294	{ name:“Sara”, city:”Tampa”, …}
0x3391	445104	{ name:“James”, city:”Miami”, …}
0x6134	333363	{ name:“Betty”, city:”Madison”, …}

0x9531	145783	{ name:“Bob”, city:”London”, …}
0xB082	643145	{ name:“Val”, city:”Seattle”, …}

0xEA8A	723342	{ name:“Jeff”, city:”Toledo”, …}
0xF355	523422	{ name:“Alex”, city:”London”, …}

Problem: Non-uniform access across partitions



0x12A8	236294	{ name:“Sara”, city:”Tampa”, …}
0x3391	445104	{ name:“James”, city:”Miami”, …}
0x6134	333363	{ name:“Betty”, city:”Madison”, …}

0x9531	145783	{ name:“Bob”, city:”London”, …}
0xB082	643145	{ name:“Val”, city:”Seattle”, …}

0xEA8A	723342	{ name:“Jeff”, city:”Toledo”, …}
0xF355	523422	{ name:“Alex”, city:”London”, …}

Global admission control

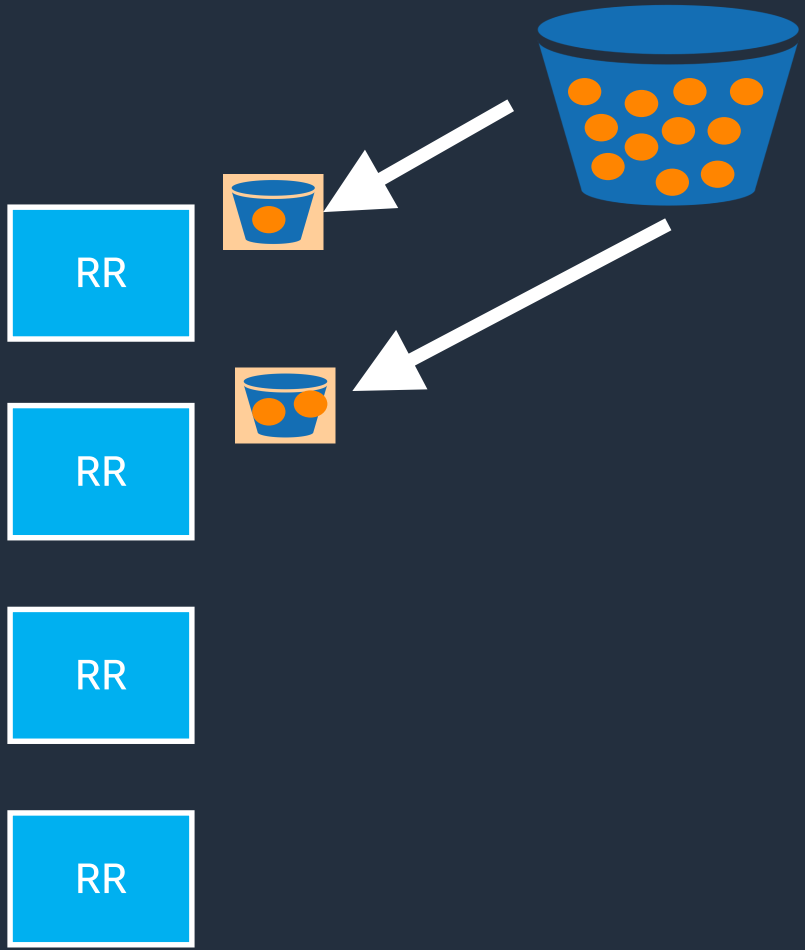
300 RCUs

0x12A8	236294	{ name:"Sara", city:"Tampa", …}
0x3391	445104	{ name:"James", city:"Miami", …}
0x6134	333363	{ name:"Betty", city:"Madison", …}

0x9531	145783	{ name:"Bob", city:"London", …}
0xB082	643145	{ name:"Val", city:"Seattle", …}

0xEA8A	723342	{ name:"Jeff", city:"Toledo", …}
0xF355	523422	{ name:"Alex", city:"London", …}

Global admission control

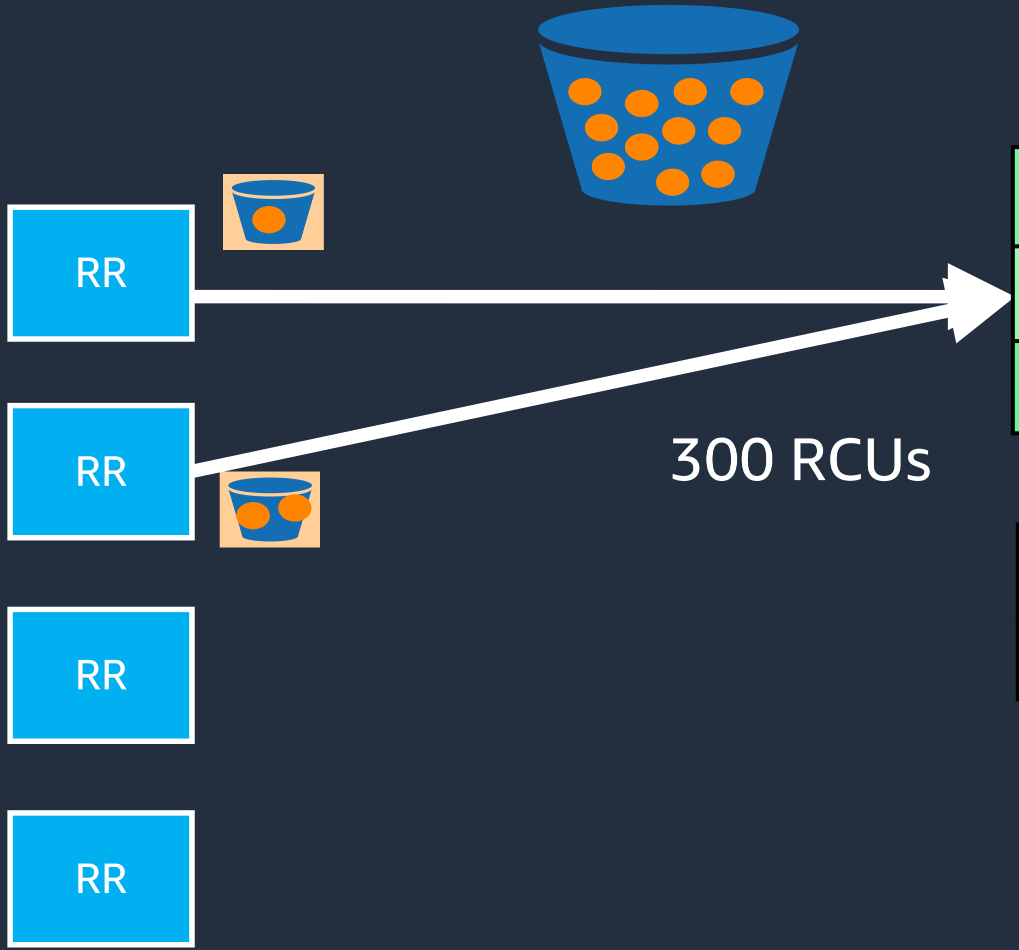


0x12A8	236294	{ name:“Sara”, city:”Tampa”, …}
0x3391	445104	{ name:“James”, city:”Miami”, …}
0x6134	333363	{ name:“Betty”, city:”Madison”, …}

0x9531	145783	{ name:“Bob”, city:”London”, …}
0xB082	643145	{ name:“Val”, city:”Seattle”, …}

0xEA8A	723342	{ name:“Jeff”, city:”Toledo”, …}
0xF355	523422	{ name:“Alex”, city:”London”, …}

Global admission control



0x12A8	236294	{ name:“Sara”, city:”Tampa”, …}
0x3391	445104	{ name:“James”, city:”Miami”, …}
0x6134	333363	{ name:“Betty”, city:”Madison”, …}

0x9531	145783	{ name:“Bob”, city:”London”, …}
0xB082	643145	{ name:“Val”, city:”Seattle”, …}

0xEA8A	723342	{ name:“Jeff”, city:”Toledo”, …}
0xF355	523422	{ name:“Alex”, city:”London”, …}

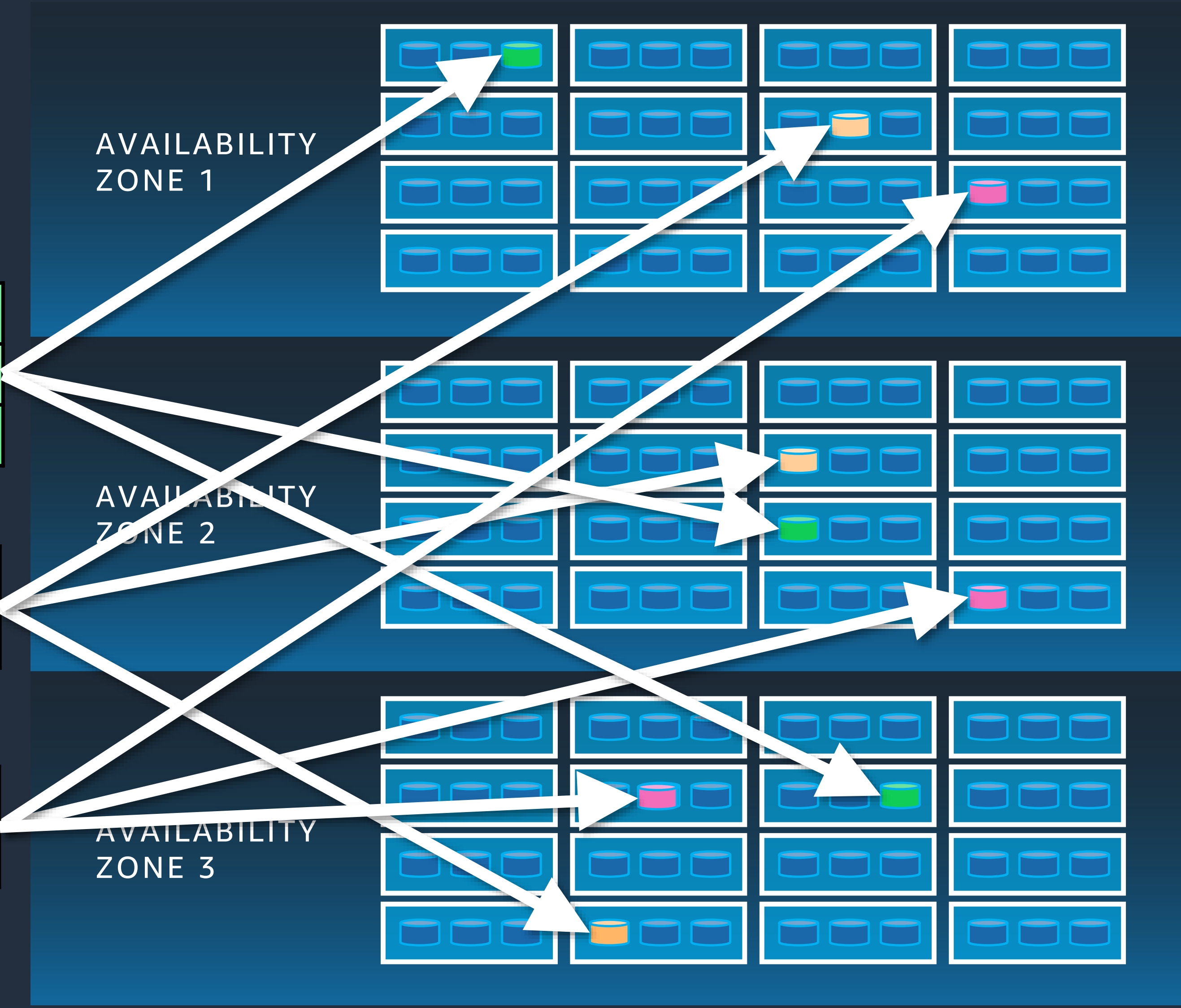
Availability

Replication

0x12A8	236294	{ name:“Sara”, city:”Tampa”, …}
0x3391	445104	{ name:“James”, city:”Miami”, …}
0x6134	333363	{ name:“Betty”, city:”Madison”, …}

0x9531	145783	{ name:“Bob”, city:”London”, …}
0xB082	643145	{ name:“Val”, city:”Seattle”, …}

0xEA8A	723342	{ name:“Jeff”, city:”Toledo”, …}
0xF355	523422	{ name:“Alex”, city:”London”, …}



Partition Map

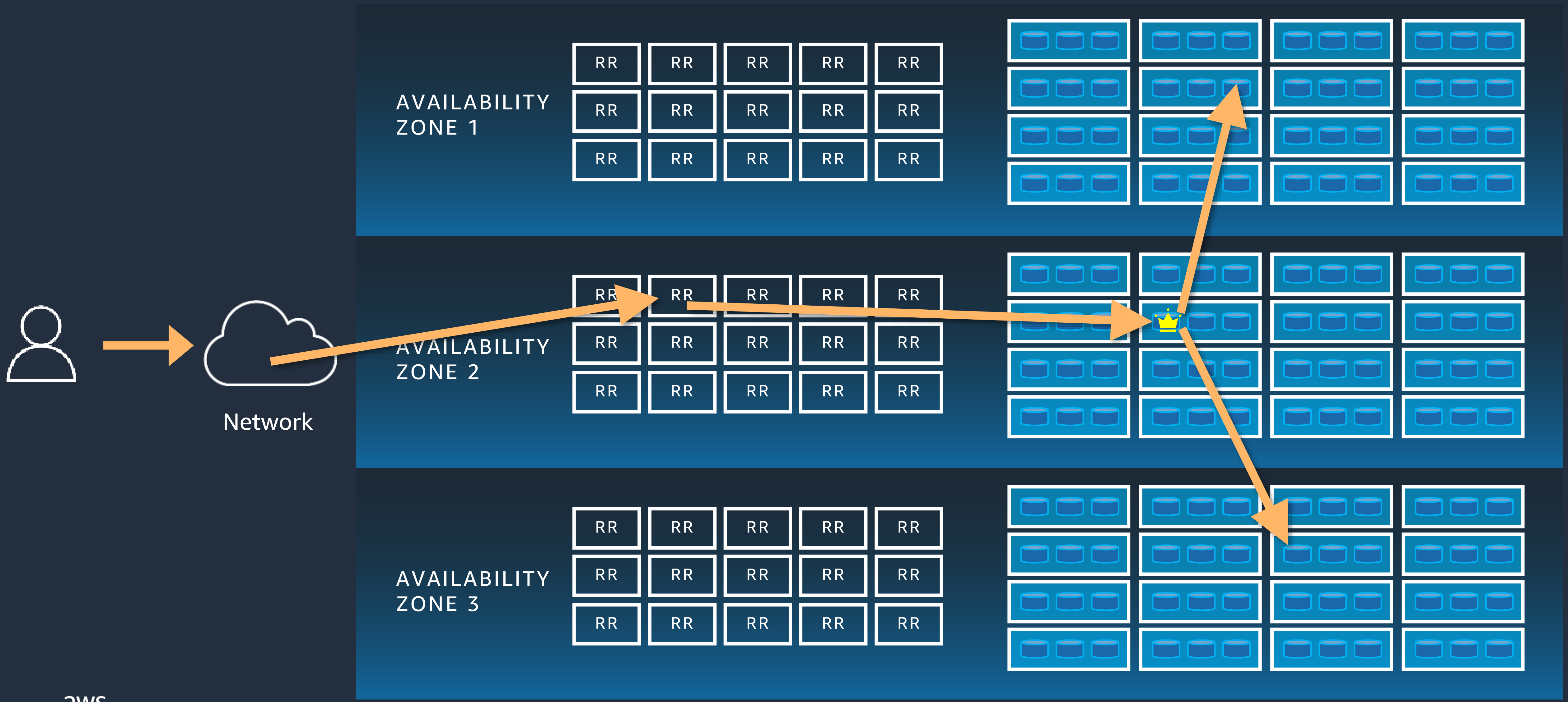
0x0000..0x6FFF	[green1, green2, green3]
0x7000..0xBFFF	[orange1, orange2, orange3]
0xC000..0xFFFF	[pink1, pink2, pink3]

0x12A8	236294	{ name:“Sara”, city:”Tampa”, …}
0x3391	445104	{ name:“James”, city:”Miami”, …}
0x6134	333363	{ name:“Betty”, city:”Madison”, …}

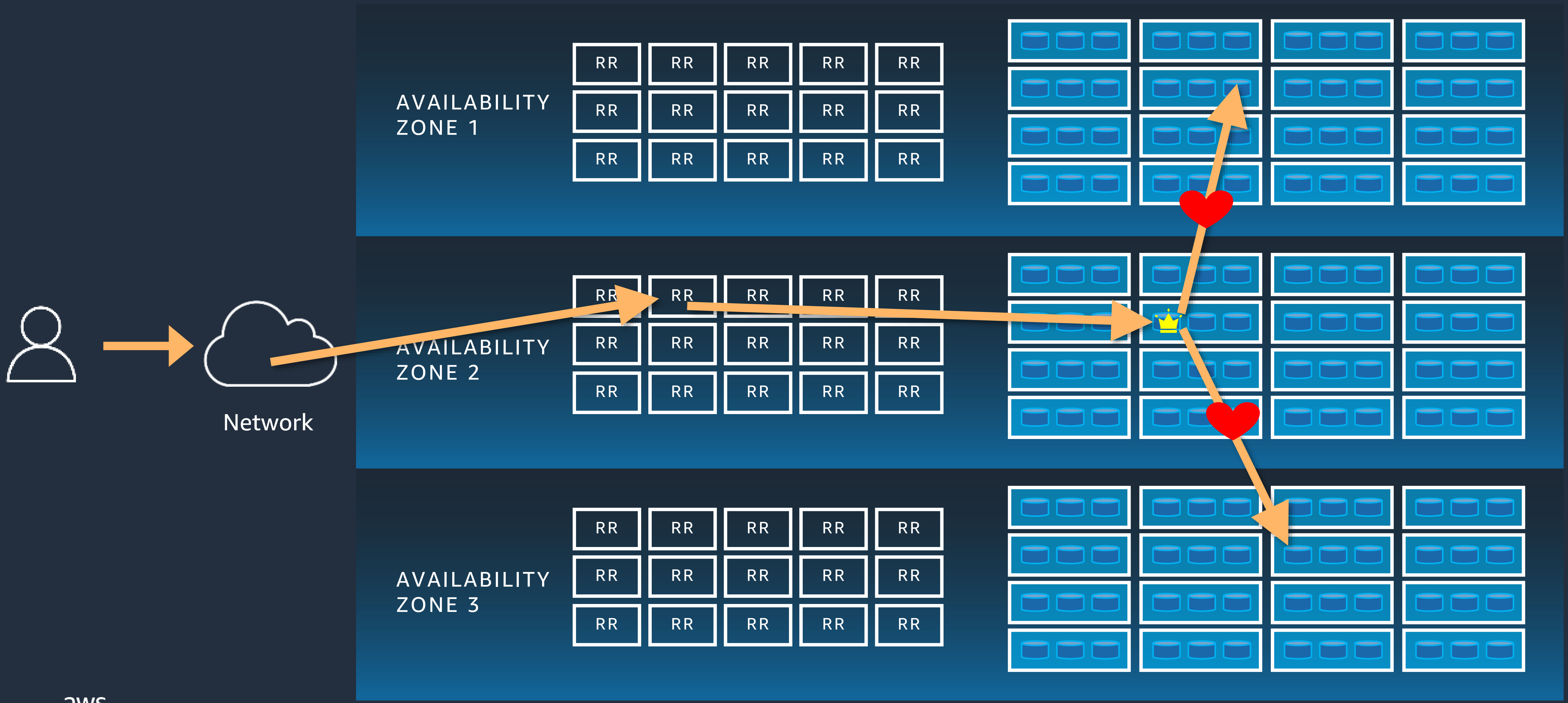
0x9531	145783	{ name:“Bob”, city:”London”, …}
0xB082	643145	{ name:“Val”, city:”Seattle”, …}

0xEA8A	723342	{ name:“Jeff”, city:”Toledo”, …}
0xF355	523422	{ name:“Alex”, city:”London”, …}

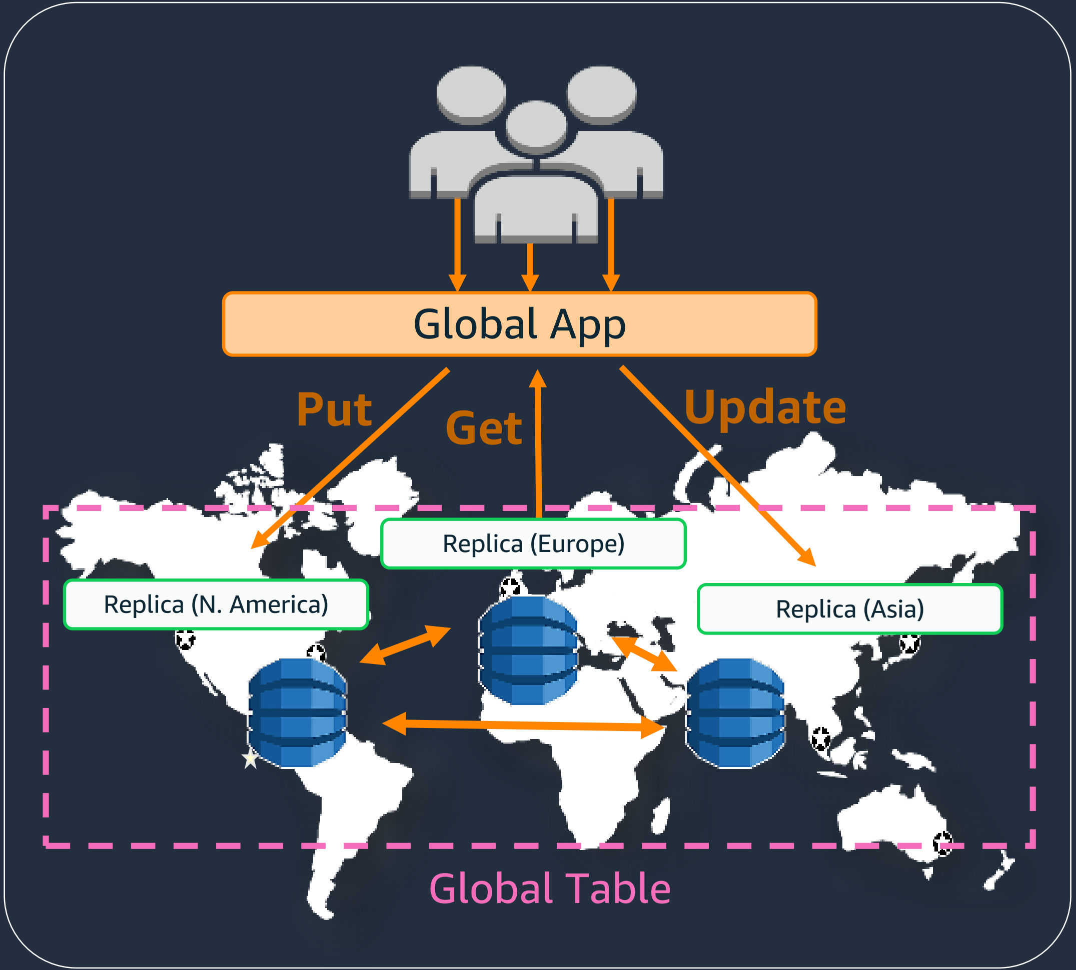
Put


Network
AVAILABILITY ZONE 1
AVAILABILITY ZONE 2
AVAILABILITY ZONE 3
RR

Put


AVAILABILITY ZONE 1
RR RR RR RR RR
RR RR RR RR RR
RR RR RR RR RR
AVAILABILITY ZONE 2
RR RR RR RR RR
RR RR RR RR RR
RR RR RR RR RR
Network
AVAILABILITY ZONE 3
RR RR RR RR RR
RR RR RR RR RR
RR RR RR RR RR

DynamoDB Global Tables



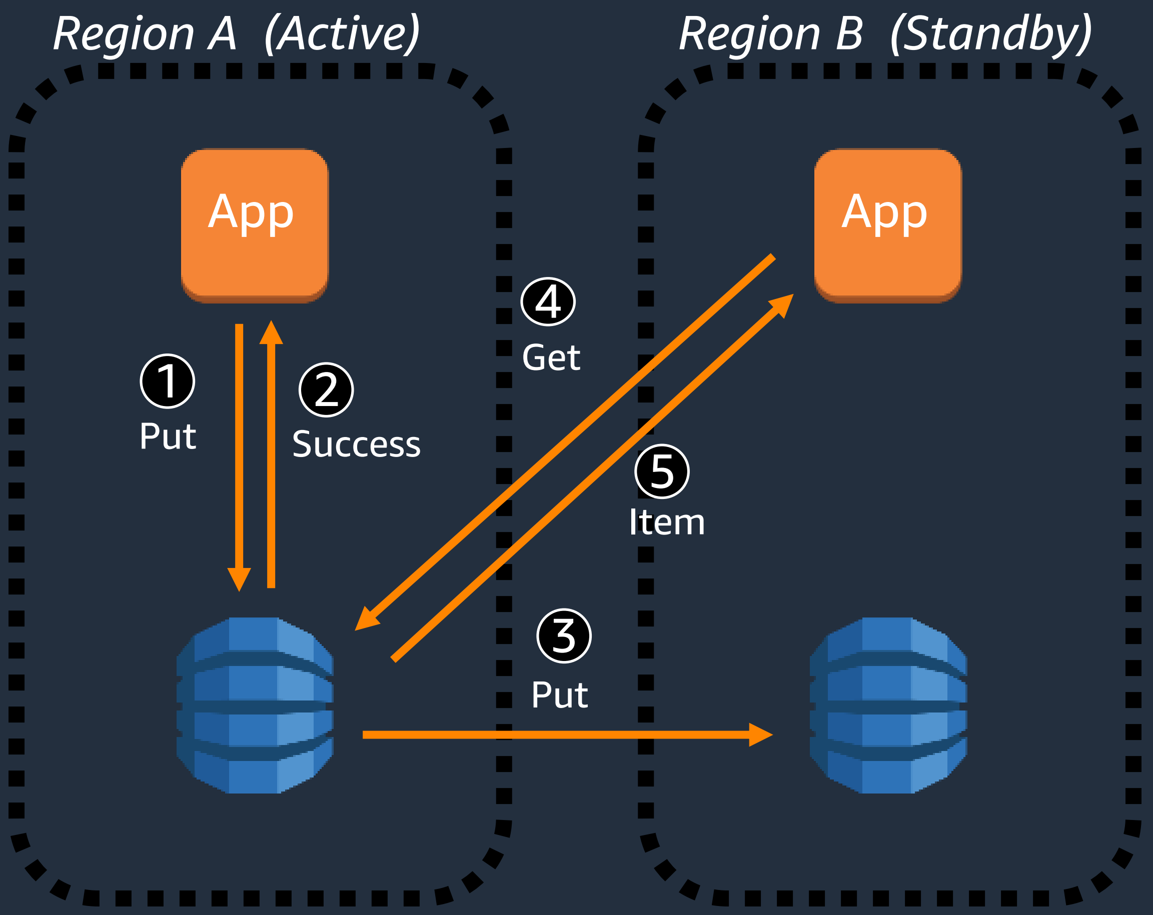
Replicate table across regions

Read and write anywhere

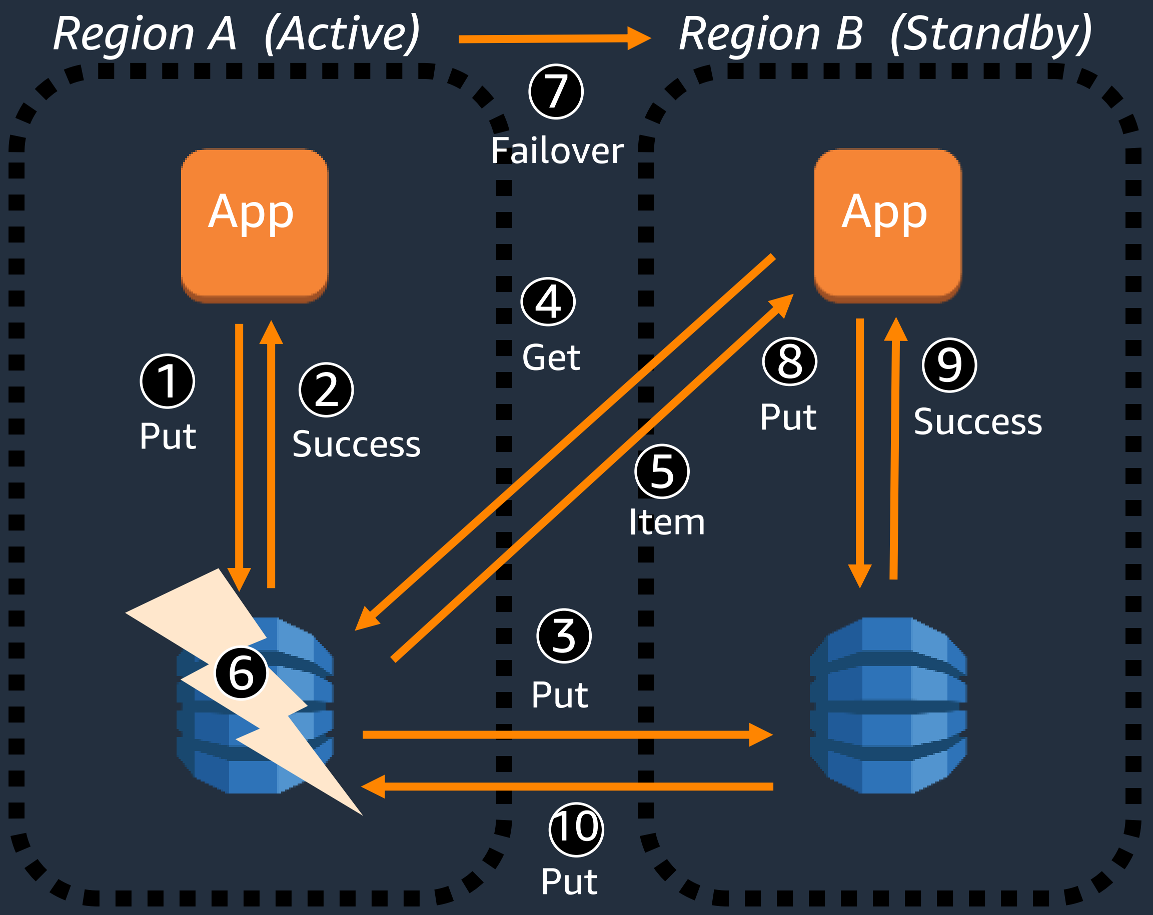
Eventual convergence

Last-writer-wins conflict resolution

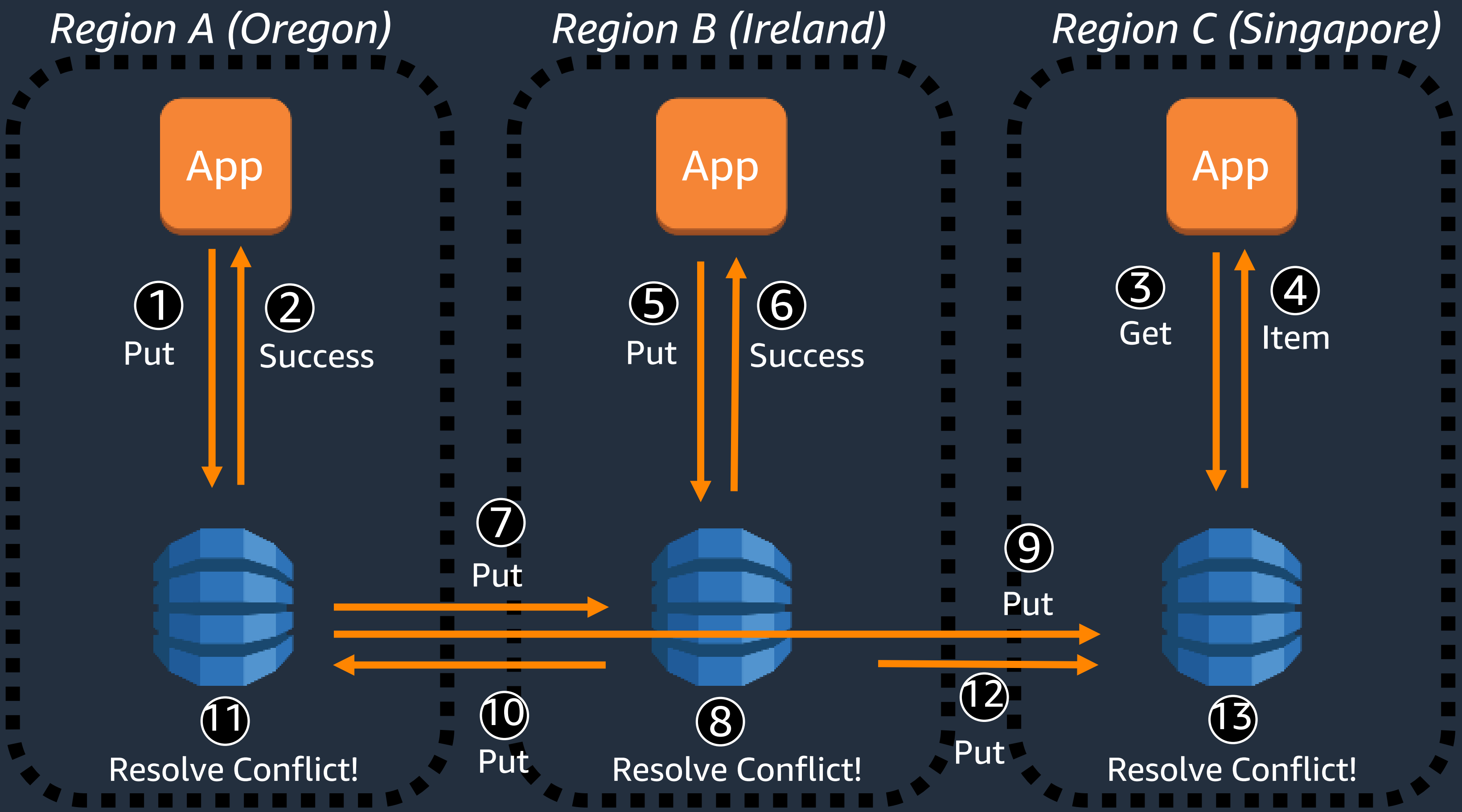
Use Case: Disaster Recovery



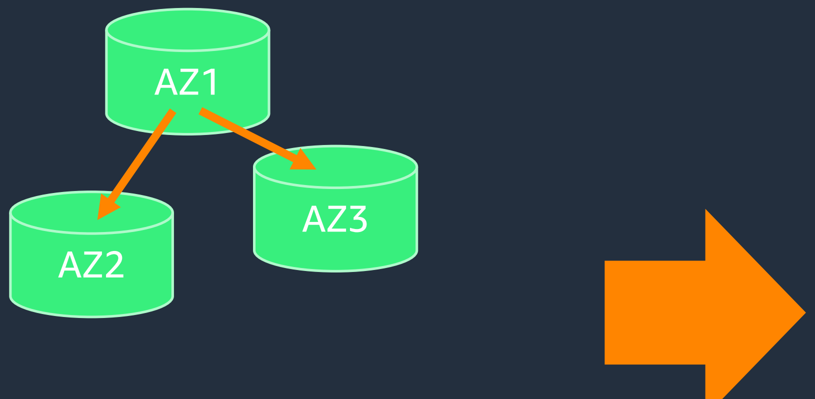
Use Case: Disaster Recovery Failover



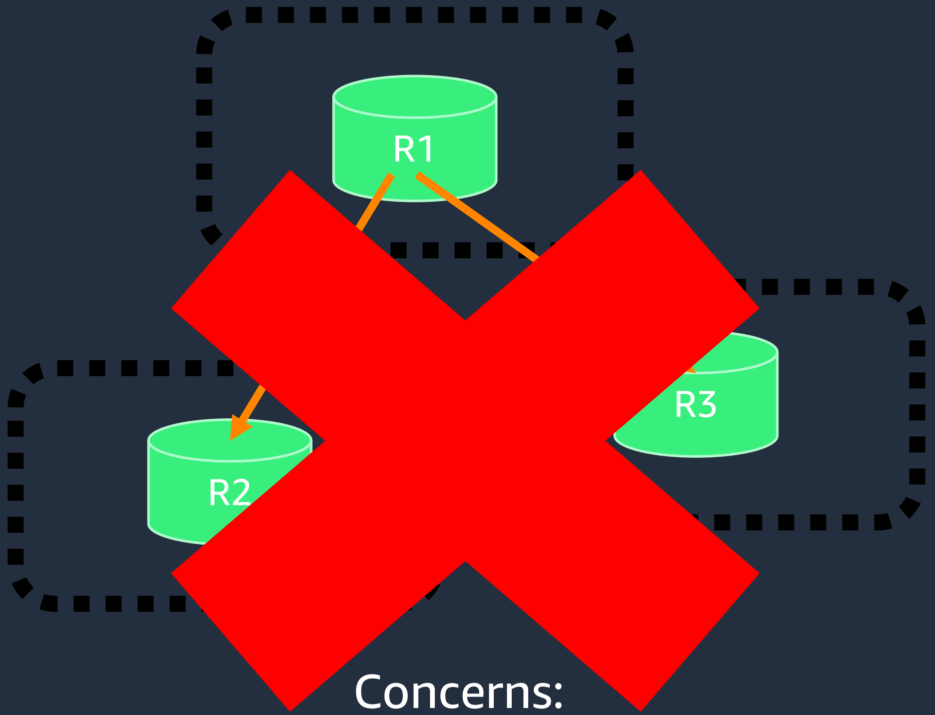
Use Case: Multi-Region Access



Intra-region vs. Cross-region Replication

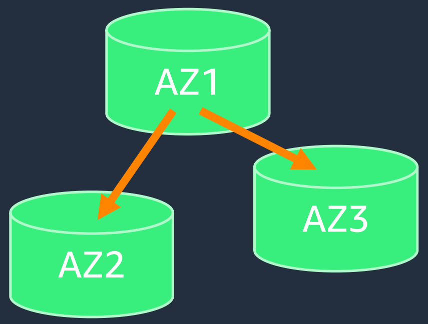


Strongly consistent
Highly available
Highly durable
Partitioned
Provisioned

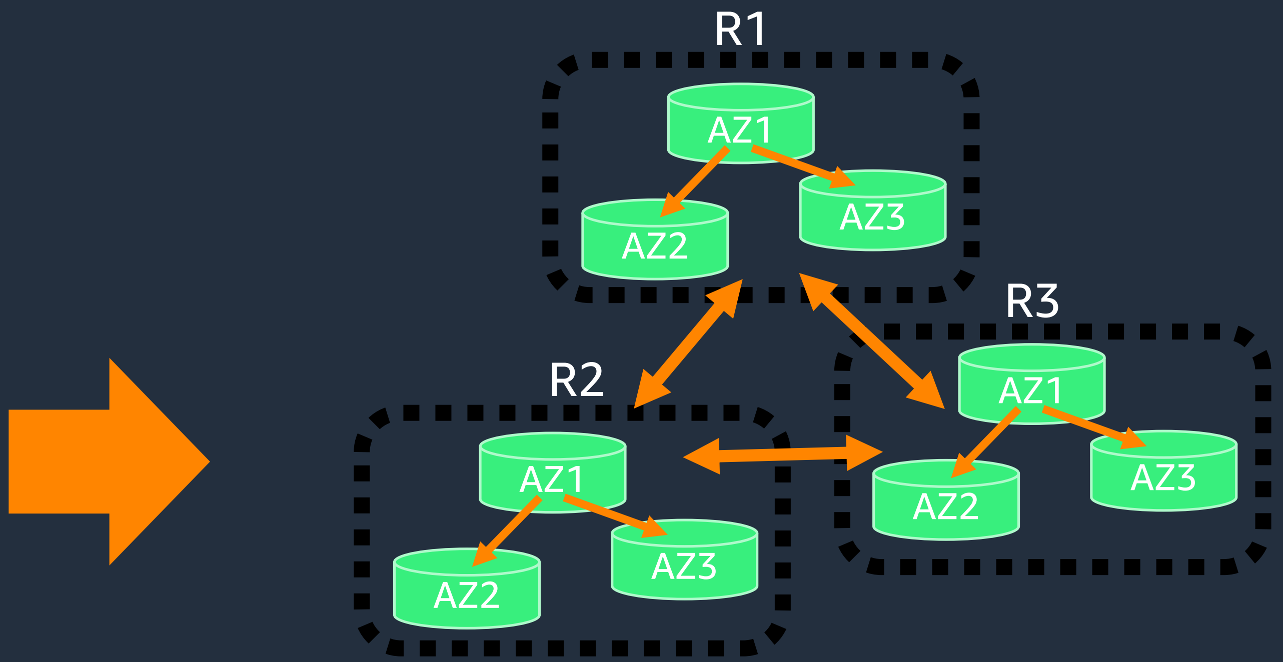


Concerns:
- Write performance
- Blast radius
- Algorithm timeouts

Intra-region vs. Cross-region Replication



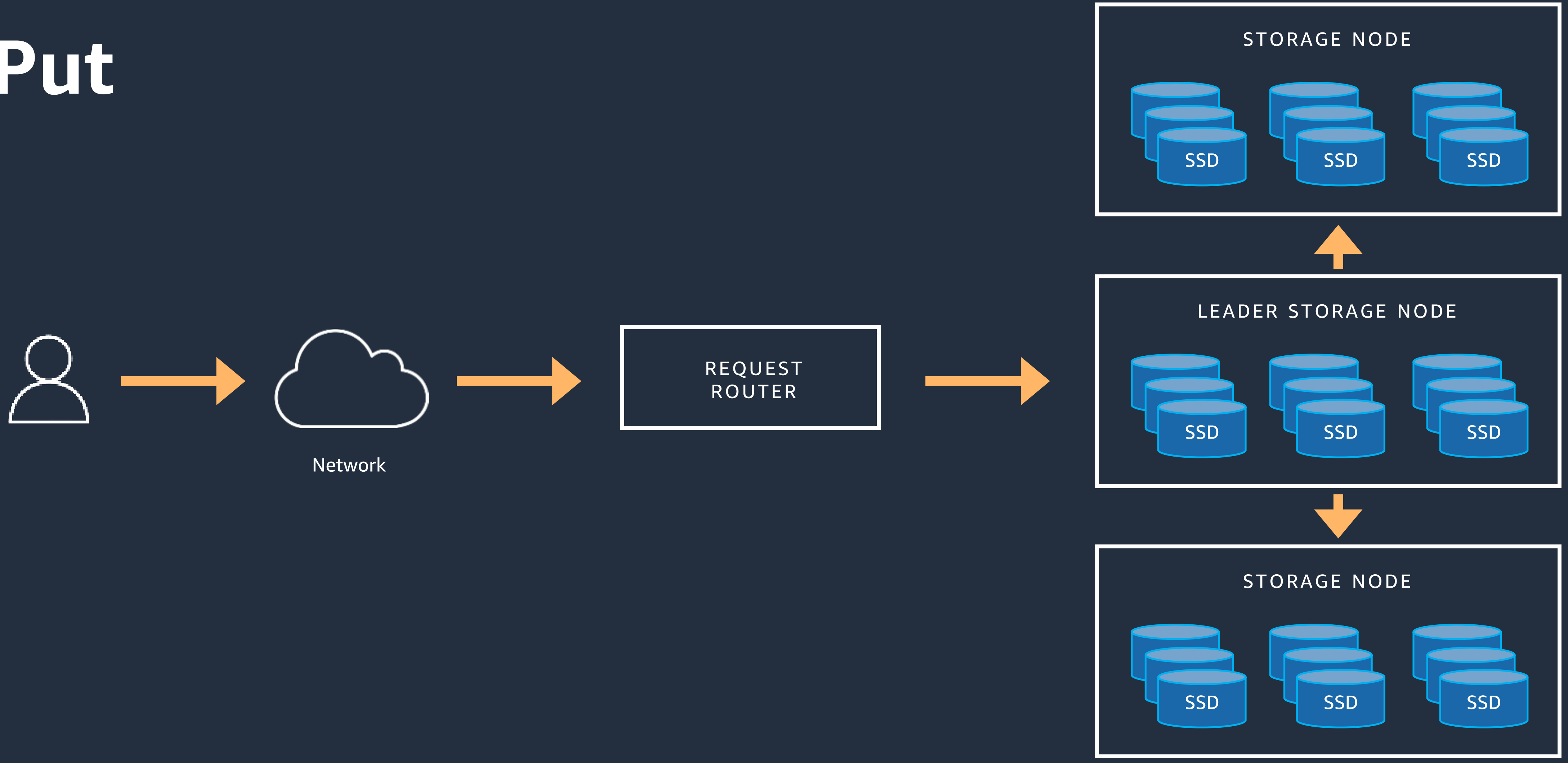
Strongly consistent
Highly available
Highly durable
Partitioned
Provisioned



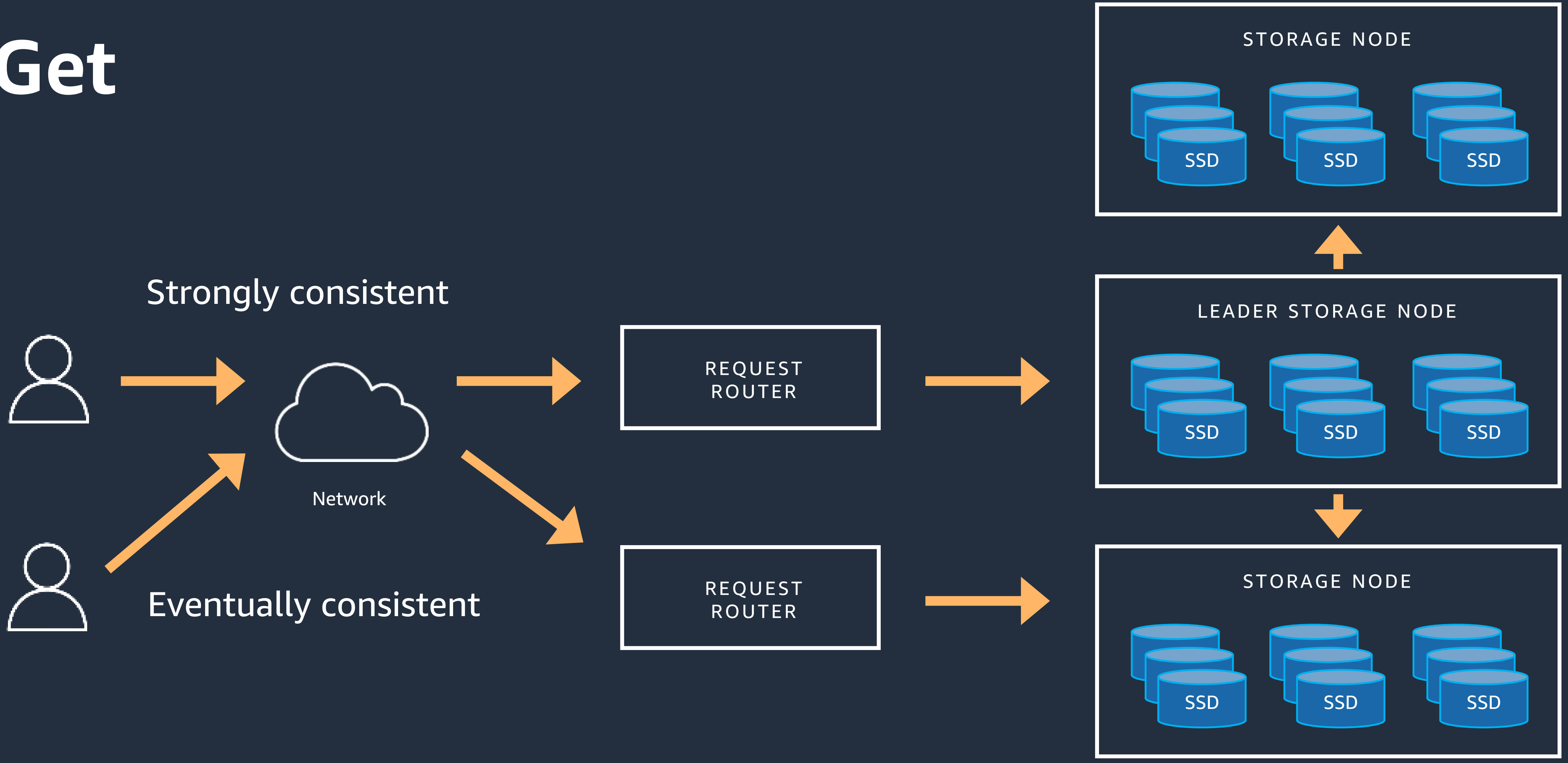
Write in single region
Eventually consistent
Fault-tolerant
Any number of regions
Maintains DynamoDB properties

Consistency

Put


Network
REQUEST ROUTER
STORAGE NODE
SSD
SSD
SSD
LEADER STORAGE NODE
SSD
SSD
SSD
STORAGE NODE
SSD
SSD
SSD

Get


STORAGE NODE
SSD
SSD
SSD
Strongly consistent
LEADER STORAGE NODE
SSD
SSD
SSD
REQUEST ROUTER
Network
REQUEST ROUTER
Eventually consistent
STORAGE NODE
SSD
SSD
SSD

Transactions

Facilitate the construction of correct and reliable applications that need to maintain multi-item invariants

Example: If Mary is Bob’s friend then Bob is Mary’s friend

Example : If Mary gives Bob $50, the total amount between them remains unchanged

Transaction Properties

Atomicity - execute all or nothing

Consistency - preserve correct state

Isolation - serialize concurrent operations

Durability - retain results permanently

DynamoDB Transactions

Execute sets of operations

atomically and serializably

for any items in any tables

with predictable performance

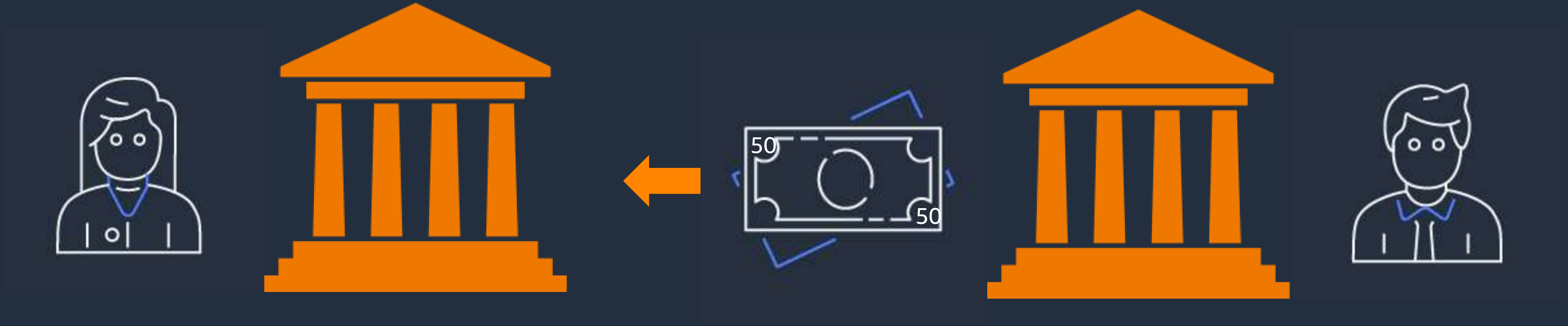
and no impact on non-transactional workloads

Example: Money Transfer

person: Mary
balance: $25

person: Bob
balance: $100

Example: Money Transfer



person: Mary
balance: $75

person: Bob
balance: $50

Example: Money Transfer

```
mary-money = Get (person: “Mary”)
bob-money = Get (person: “Bob”)
Put (person: “Mary”, balance: mary-money + 50)
Put (person: “Bob”, balance: bob-money - 50)
```

Example: Money Transfer

```
mary-money = Get (person: “Mary”)
bob-money = Get (person: “Bob”)
Put (person: “Mary”, balance: mary-money + 50)
Put (person: “Bob”, balance: bob-money - 50)
```

crash

Bob keeps his money

Example: Money Transfer

```
mary-money = Get (person: “Mary”)
bob-money = Get (person: “Bob”)
Put (person: “Mary”, balance: mary-money + 50)
Put (person: “Bob”, balance: bob-money - 50)
```

Example: Money Transfer

mary-money = Get (person: “Mary”)

bob-money = Get (person: “Bob”)

bob-money = Get (person: “Bob”)
Put (person: “Bob”, bob-money + 100)

Put (person: “Mary”, balance: mary-money + 50)

Put (person: “Bob”, balance: bob-money - 50)

Where’s my $100?

Standard Approach Rejected

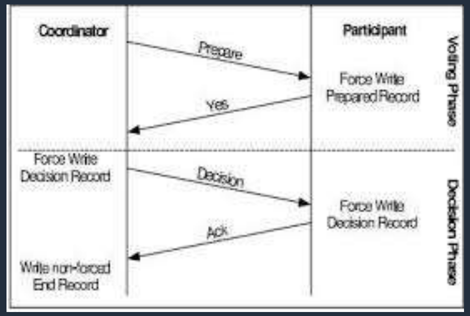
```
TxBegin
  ...
TxCommit
```

Explicit multi-step transactions

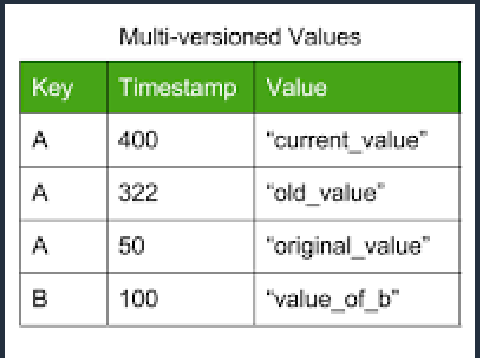
```
TxBegin
Put (...)
TxCommit
```

Implicit singleton transactions

Two-phase locking


Coordinator
Participant
Prepare
Force Write Prepared Record
Yes
Voting Phase
Force Write Decision Record
Decision
Force Write Decision Record
Ack
Write non-forced End Record
Decision Phase

Two-phase commit



Multi-versioned Values

Key	Timestamp	Value
A	400	"current_value"
A	322	"old_value"
A	50	"original_value"
B	100	"value_of_b"

Multi-version Concurrency Control

DynamoDB Transactions

```
TransactGetItems (
    Get (table: “T1”, key: k1),
    Get (table: “T2”, key: k2),
    Get (table: “T3”, key: k3)
)
```

```
TransactWriteItems (
        Put (table: “T1”, key: k1,
            value: v1),
        Delete (table: “T2”, key: k2),
        Update (table: “T3”, key: k3,
            value: +1),
        Check (table: “T3”, key: k3,
            value: < 100)
)
```

Shopping Example

Orders

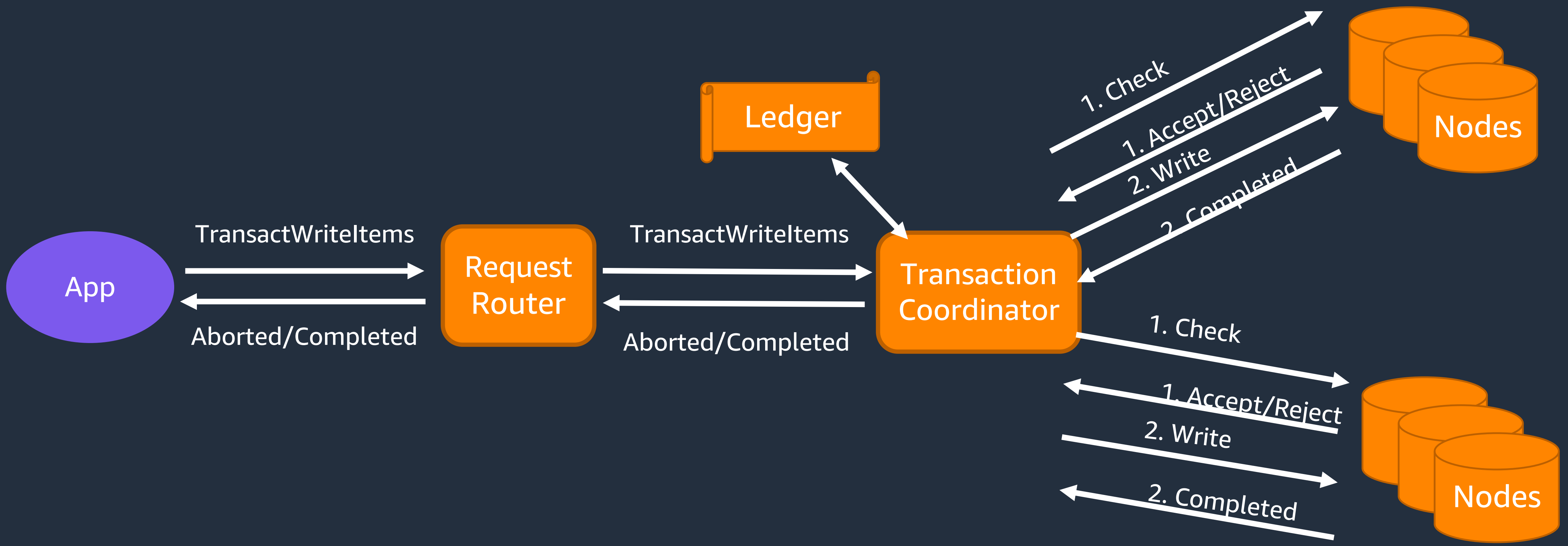
Customers

Inventory

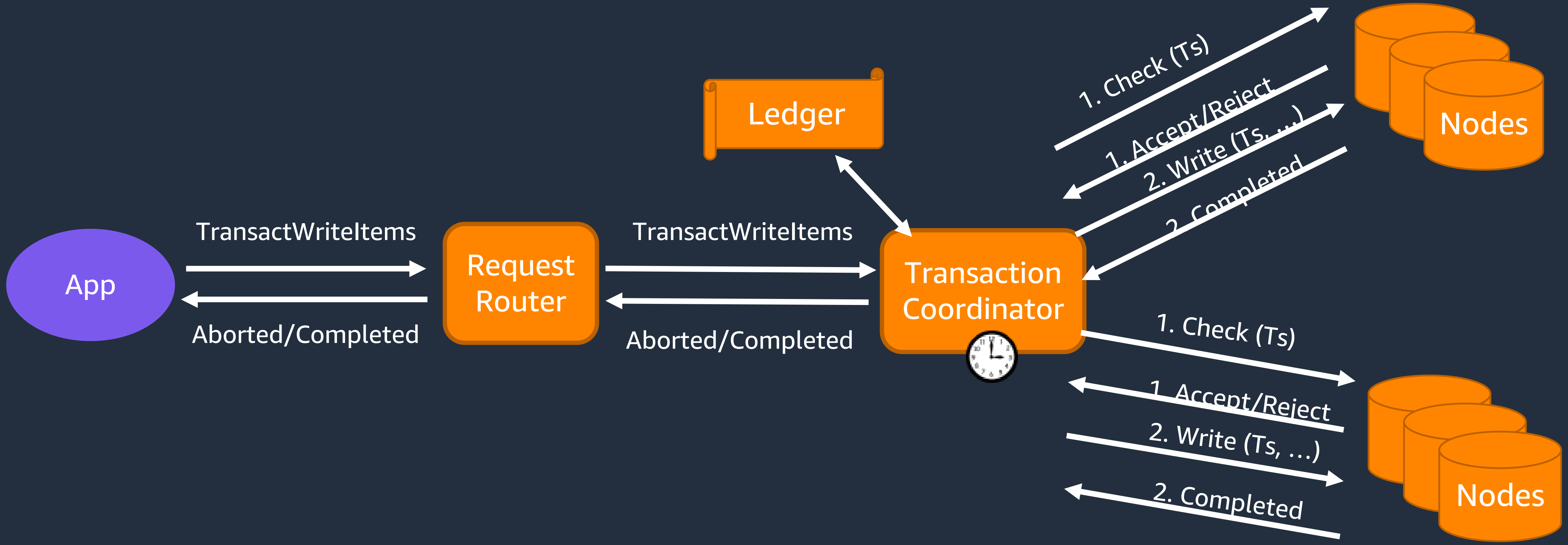
Shopping Example

```
TransactWriteItems (
        Check (table: “Customers”, key: “Susie” EXISTS),
        Check (table: “Inventory”, key: “book-99”, amount: >= 5),
        Put (table: “Orders”, key: newGUID(), customer: “Susie”,
          product: “book-99”, copies: 5, …),
        Update (table: “Inventory”, key: “book-99”, amount: – 5)
)
```

DynamoDB Transactions Architecture



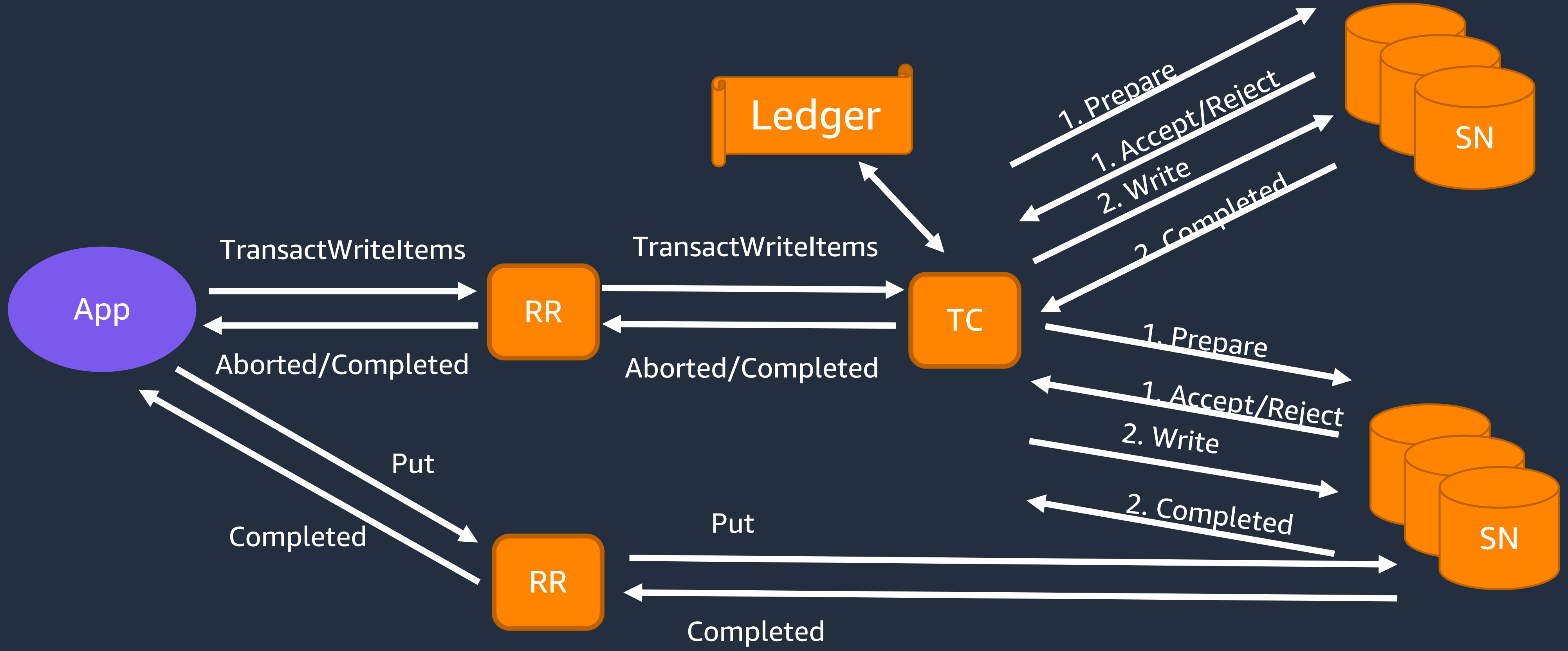
Timestamp Ordering



Phil A. Bernstein, David W. Shipman, and James B. Rothnie, Concurrency Control in a System for Distributed Databases (SDD-1), *ACM TODS*, 1980.

David P. Reed, Implementing Atomic Actions on Decentralized Data, *ACM TOCS*, 1983.

Non-transactional Operations



Take Away

DynamoDB evolved to meet customer needs while improving on its fundamental characteristics: predictability, scalability, availability, and consistency

Thank you!