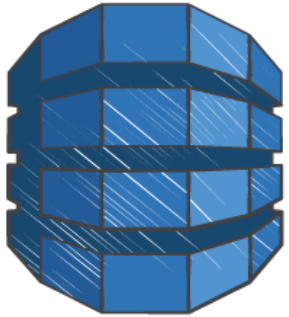
Amazon DynamoDB Deep Dive



Matt Yanchyshyn: Solutions Architect, AWS
Shekhar Deshkar: Chief Architect, Druva

Amazon DynamoDB Deep Dive

- What’s new
- JSON Document Support
- Flexible access control
- Partitions & Scaling
- Optimizing Data Access Cost
- Customer use case: Druva

Amazon DynamoDB

Fast & flexible NoSQL database

Predictable performance

Seamless & massive scalability

Fully managed; zero admin

amazon web services

What’s new?

- JSON Document Support
- Larger items: 400KB max item size
- Additional Scaling Options
- Expanded Free Tier: 25GB & 200M requests per month

Amazon DynamoDB Deep Dive

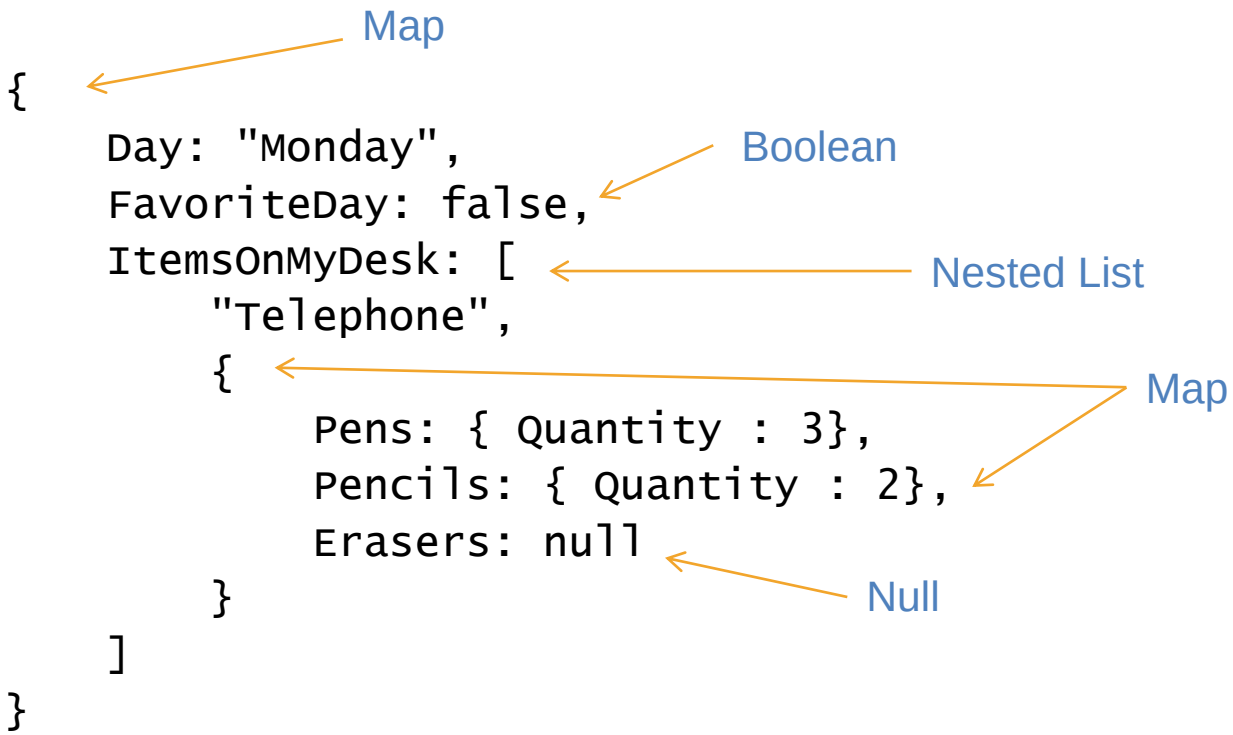
✓ What's new

- **JSON Document Support**
- Flexible access control
- Partitions & Scaling
- Optimizing Data Access Cost
- Customer use case: Druva

New Data Types

- **Scalars:** Number, String, Binary, **Boolean, Null**
- **Multi-value:** String Set, Number Set, Binary Set
- **Document: List and Map**

New Data Types



Document Paths

```
{
    Day: "Monday",
    FavoriteDay: false,
    ItemsOnMyDesk: [
        "Telephone",
        {
            Pens: { Quantity : 3},
            Pencils: { Quantity : 2},
            Erasers: null
        }
    ]
}
```

Example: AWS SDK for Java Document API

```
Item item = new Item()
  .withPrimaryKey("Day", "Monday")
  .withJSON("document", json);
table.putItem(item);
```

{ItemsOnMyDesk=[Telephone, {Erasers=null, Pencils={Quantity=2}, Pens={Quantity=3}}], FavoriteDay=false}

```
Item documentItem = table.getItem(new GetItemSpec()
  .withPrimaryKey("Day", ”Monday")
  .withAttributesToGet("document"));
```

{ItemsOnMyDesk=[{Pens={Quantity=3}}]}

```
Item documentItem = table.getItem(new GetItemSpec()
  .withPrimaryKey("Day", "Monday")
  .withProjectionExpression("document.ItemsOnMyDesk[1].Pens.Quantity"));
```

Example: AWS SDK for Java Document API

```
Item item = new Item()
   .withPrimaryKey("Day", "Friday")
   .withMap("document", new ValueMap()
     .withBoolean("FavoriteDay", true)
     .withList("ItemsOnMyDesk", "Telephone",
       new ValueMap()
       .withMap("Pens", new ValueMap()
         .withInt("Quantity", 1))
       .withMap("Pencils", new ValueMap()
           .withInt("Quantity", 0))
         .withMap("Erasers", new ValueMap()
           .withInt("Quantity", 1))));
```

```
{
  Day: "Monday",
  FavoriteDay: false,
  ItemsOnMyDesk: [
    "Telephone",
    {
      Pens: { Quantity : 3},
      Pencils: { Quantity : 2},
      Erasers: null
    }
  ]
}
```

Questions?

Amazon DynamoDB Deep Dive

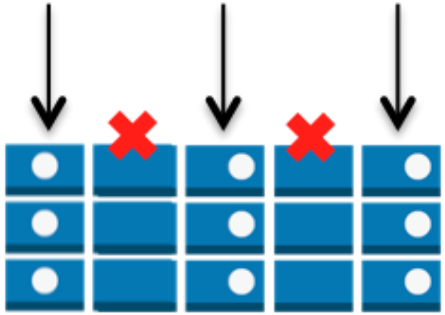
✓ What's new

✓ JSON Document Support

- **Flexible access control**
- Partitions & Scaling
- Optimizing Data Access Cost
- Customer use case: Druva

Flexible access control

- Control access to individual items and attributes
- Implement using AWS IAM roles that authenticated users assume
- Leverage SAML or WIF



Flexible access control with AWS IAM

```
...
"Effect": "Allow",
"Action": [ "dynamodb:Query", "dynamodb:Scan" ],
"Resource": "arn:aws:dynamodb:REGION:abcde12345:users/example",

"Condition": {
  "ForAllValues:StringEquals": {
    "dynamodb:LeadingKeys": ["${saml:sub}"],
    "dynamodb:Attributes": [ "userId", "firstName", "lastName" ]
  },
  "StringEqualsIfExists": {
    "dynamodb:Select": "SPECIFIC_ATTRIBUTES"
  }
...
```

DynamoDB API actions allowed

Authenticated user ID / DynamoDB table key

Item attributes that user can access

amazon web services

Questions?

…and if you want to take DynamoDB's flexible security model to the next level, check-out client-side encryption in Java with DynamoDBMapper:

http://bit.ly/1zerqw2

Amazon DynamoDB Deep Dive

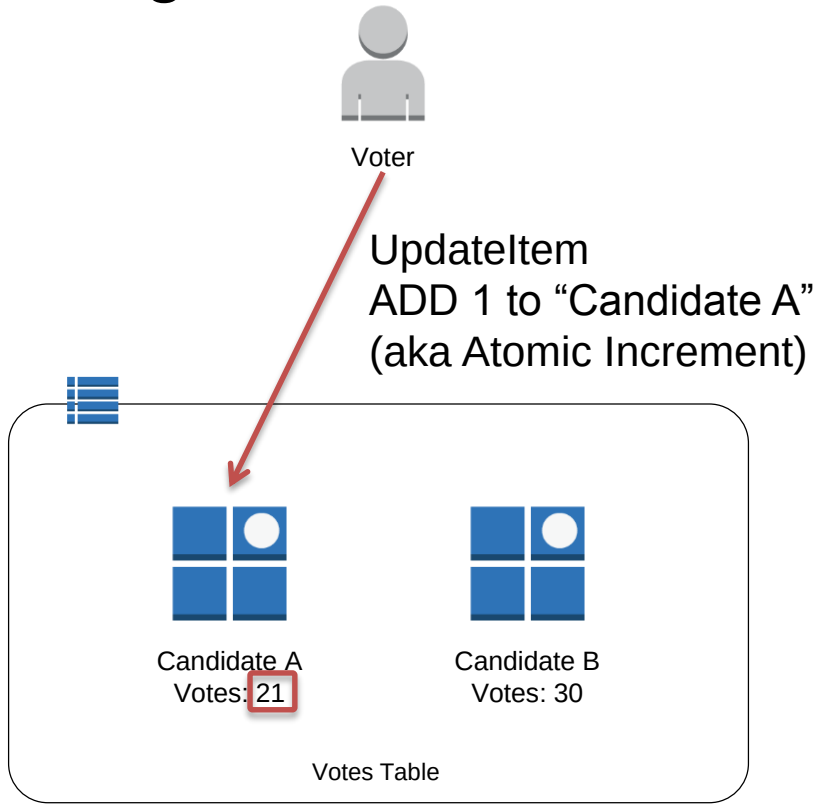
✓ What's new

✓ JSON Document Support

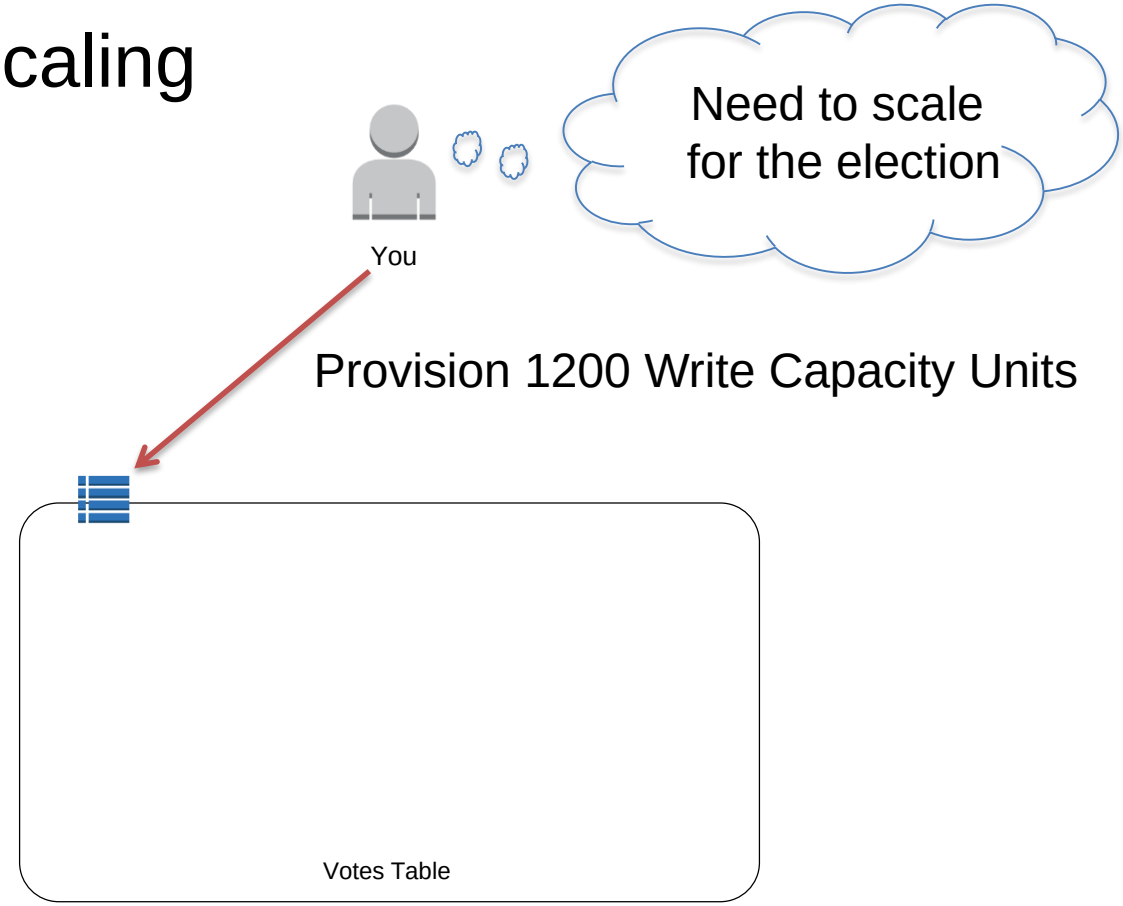
✓ Flexible access control

- **Partitions & Scaling**
- Optimizing Data Access Cost
- Customer use case: Druva

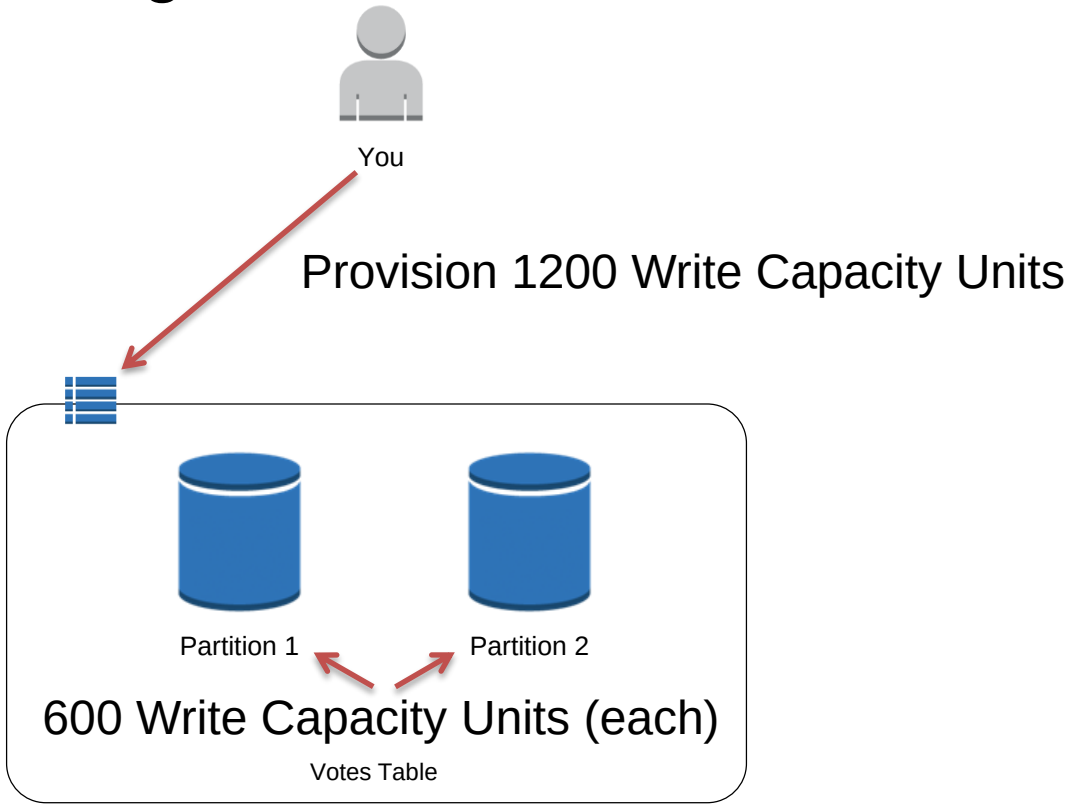
Partitions & Scaling



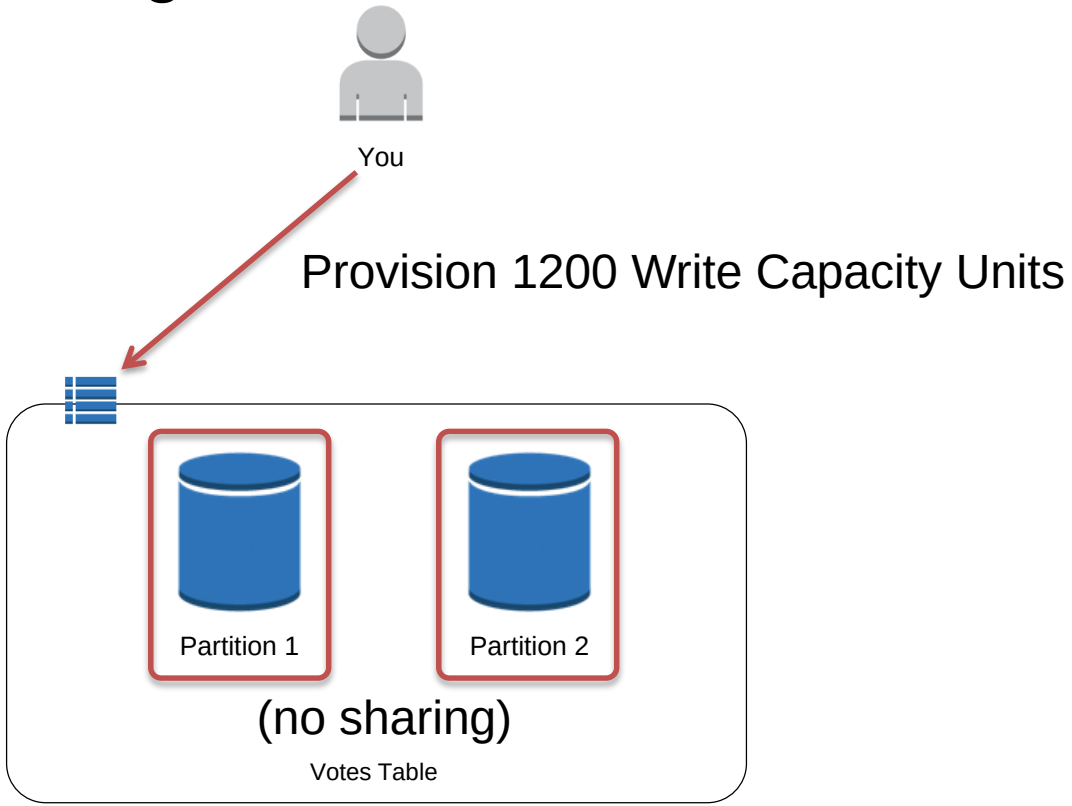
Partitions & Scaling



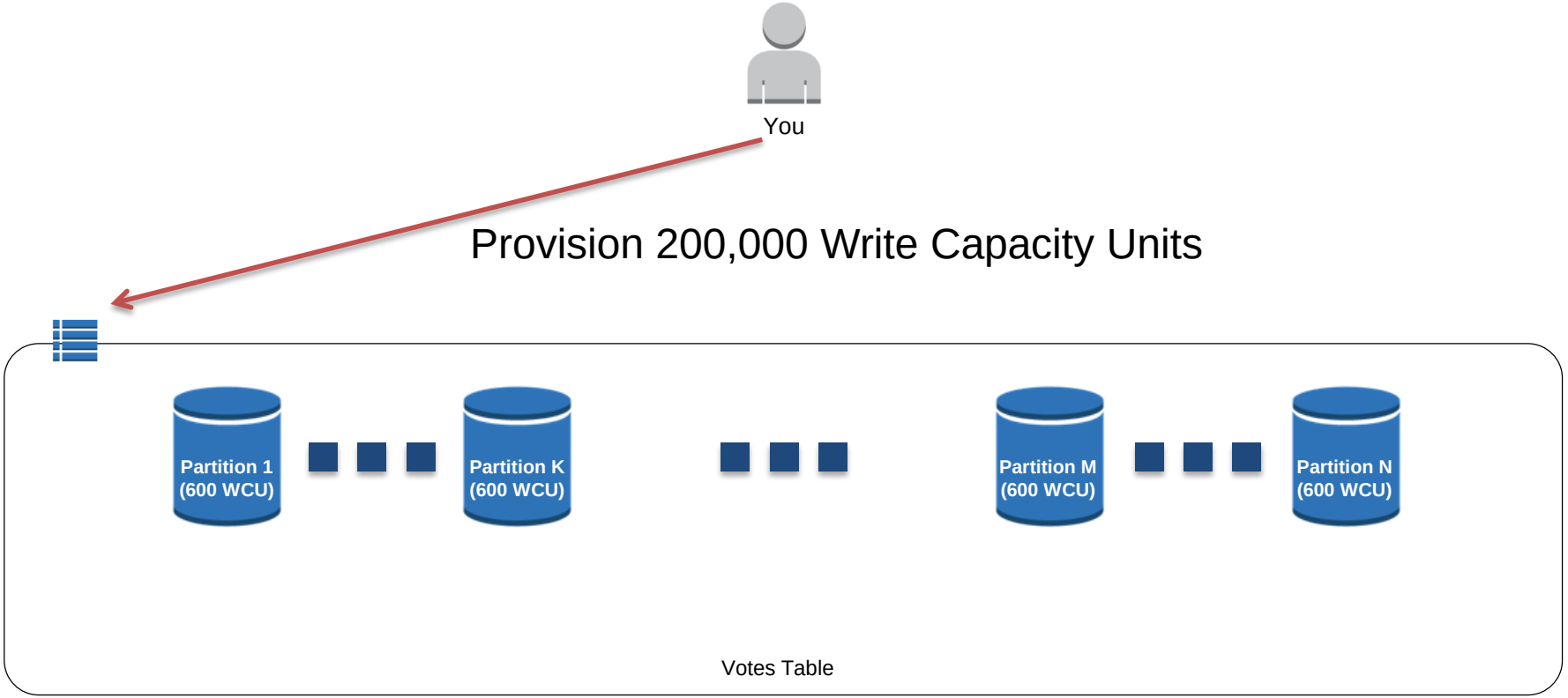
Partitions & Scaling



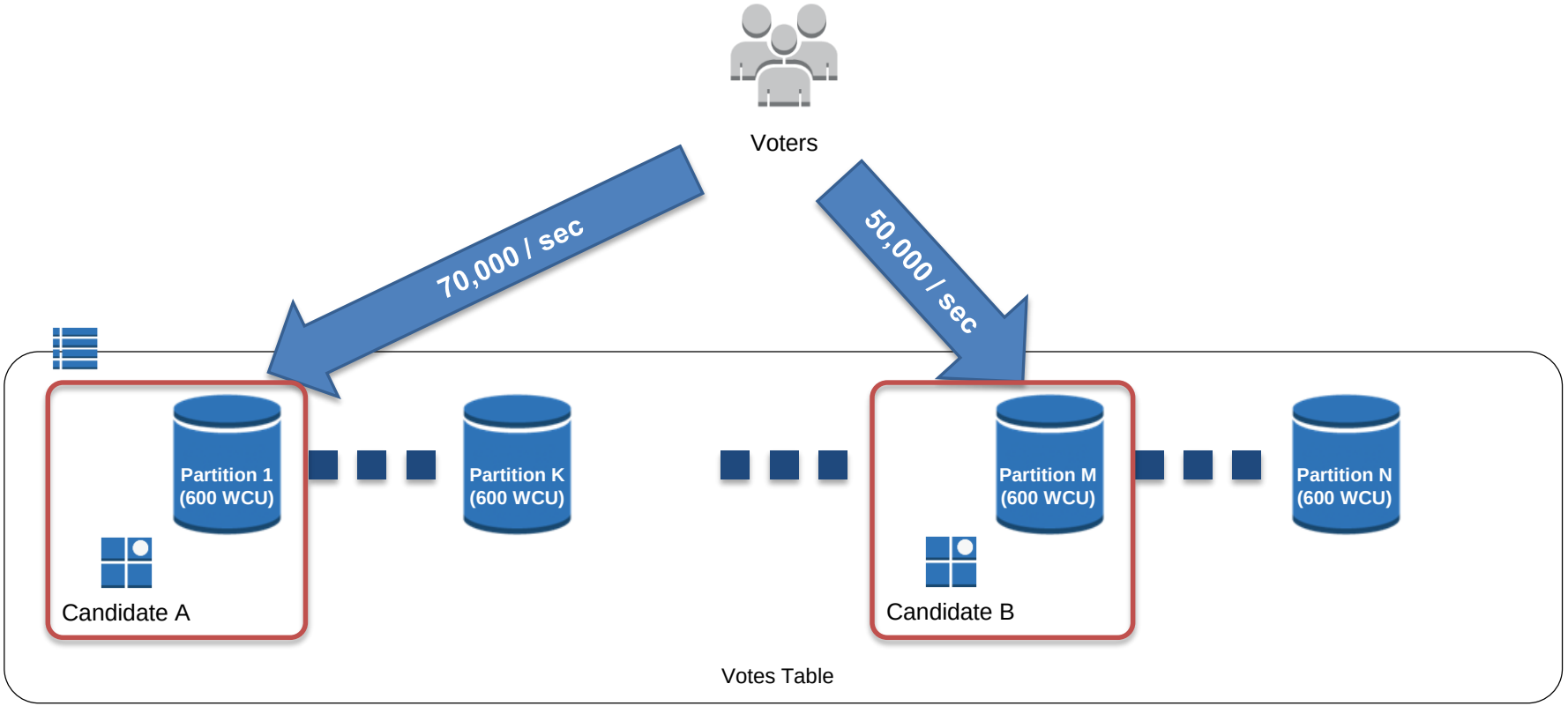
Partitions & Scaling



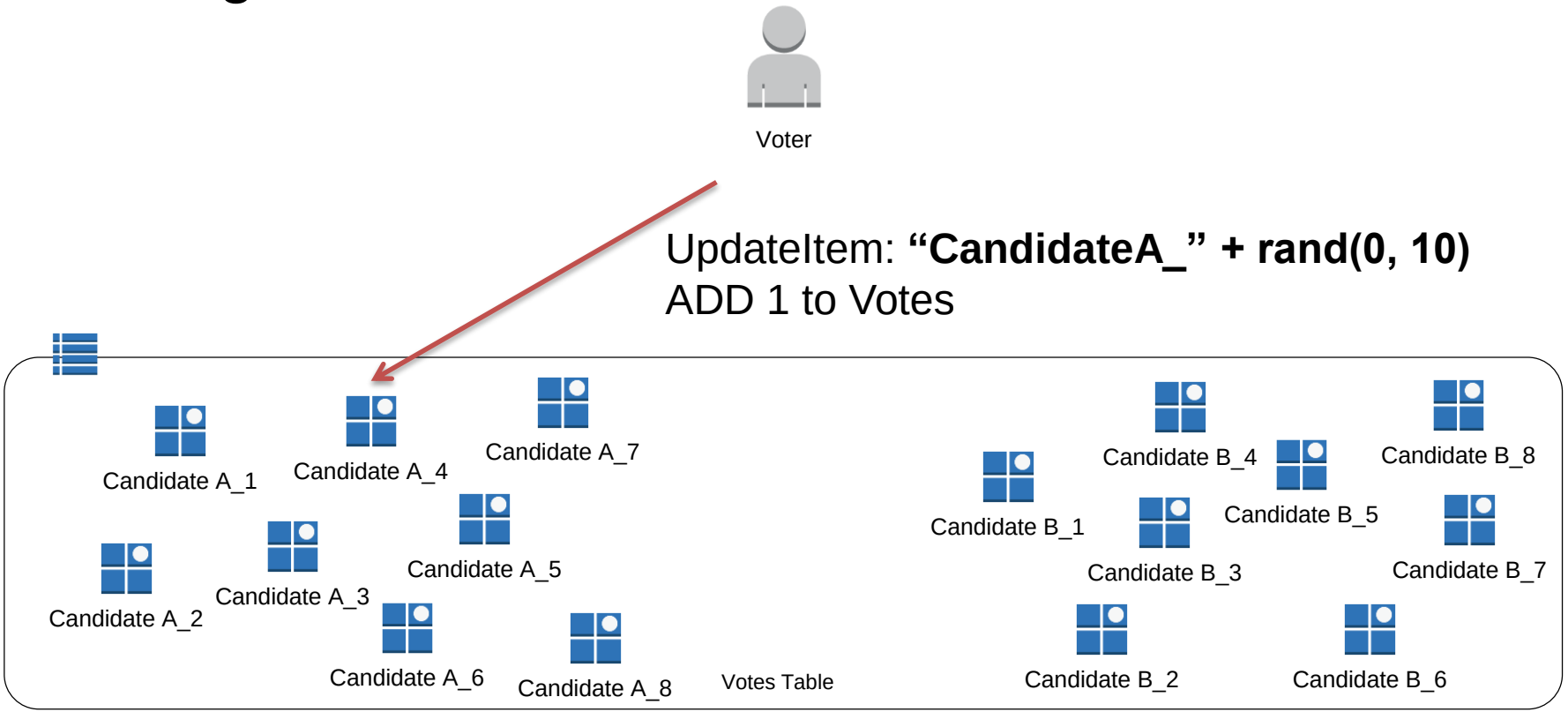
Partitions & Scaling



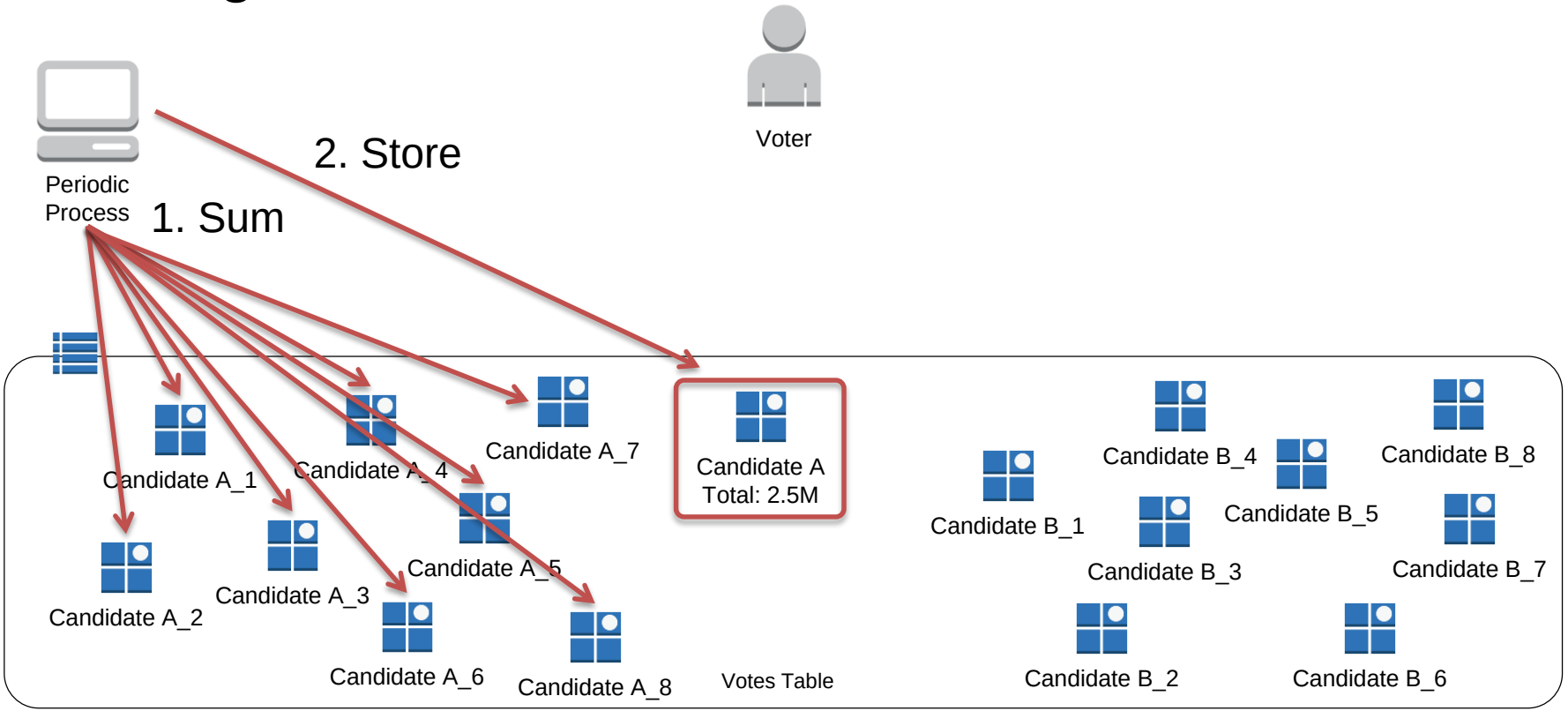
Partitions & Scaling



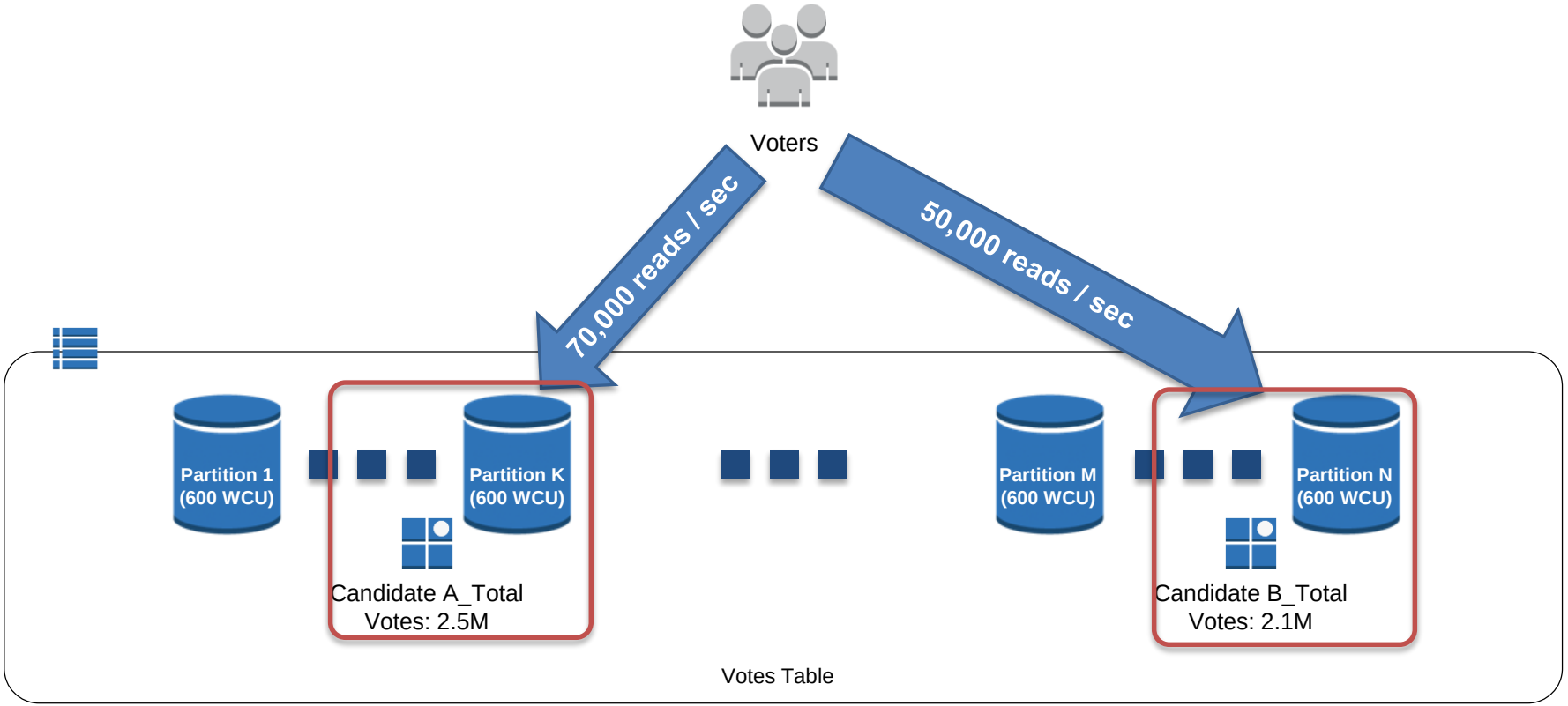
Scaling Writes



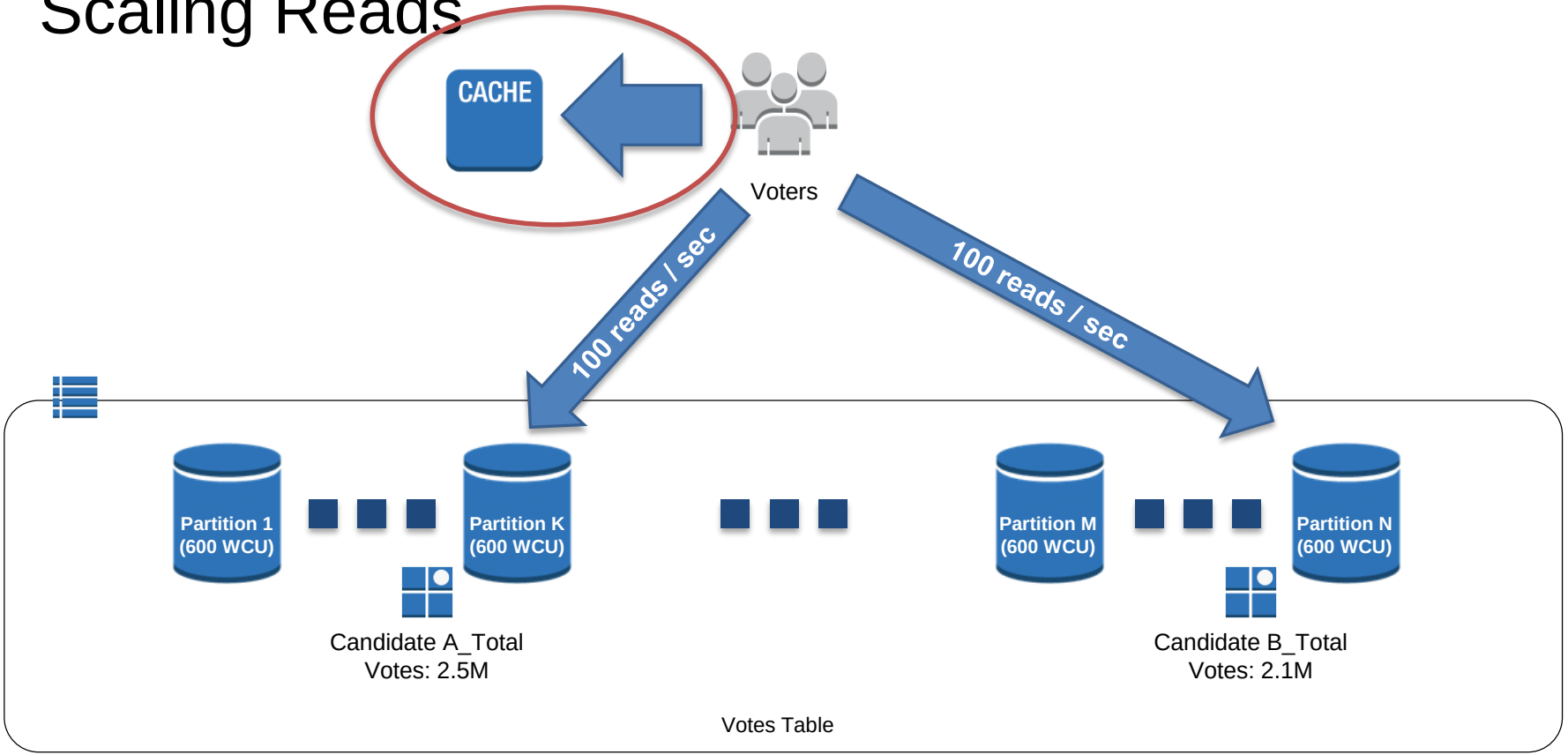
Scaling Writes



Scaling Reads



Scaling Reads



Scaling Best Practices

- Design for uniform data access
- Distribute write activity
- Understand access patterns
- Be careful not to over-provision

Questions?

Amazon DynamoDB Deep Dive

- ✓ What's new
- ✓ JSON Document Support
- ✓ Flexible access control
- ✓ Partitions & Scaling
- **Optimizing Data Access Cost**
- Customer use case: Druva

Optimizing Query Cost

Provisioned Throughput Capacity Units

- 1 Write Capacity Unit (WCU) = Up to 1 KB Item
- 1 Read Capacity Unit (RCU) = Up to 4 KB of Items
- 0.5 RCU = Up to 4 KB of Items (Eventually Consistent)

How much throughput do I need?

- Investigate customer workload:
  - How many requests per second per hash key?
  - How much will those operations cost?
- Investigate customer data size:
  - How big will the table be initially?
  - How large will the table grow?
  - Will the throughput grow proportionally?

Best practices: Query vs. BatchGetItem

- Query treats all items as a single read operation
  - Items share the same hash key = same partition
  - BatchGetItem reads each item in the batch separately
- Example
  - Read 100 items in a table, all with the same hash key. Each item is 120 bytes in size
  - Query RCU = 3
  - BatchGetItem RCU = 100

Filter cost calculation

Query UserId=Alice, Filter where OpponentId=Bob

UserId	GameId	Date	OpponentId
Carol	e23f5a	2013-10-08	Charlie
Alice	d4e2dc	2013-10-01	Bob
Alice	e9cba3	2013-09-27	Charlie
Alice	f6a3bd	2013-10-08	Bob

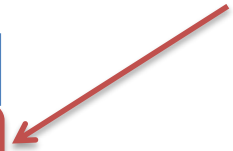
Filters **do not** reduce the throughput cost

Query/Scan: 1 MB max returned per operation. Limit applies **before** filtering

Example: Query & Read Capacity Units

```
Query PageViews
WHERE Page=index.html
  AND Time >= 2014-06-25T12:00
LIMIT 10
```

Page	Time	Views
index.html	2014-06-25T12:00	120
index.html	2014-06-25T12:01	122
index.html	2014-06-25T12:02	125
search.html	2014-06-25T12:00	200



42 bytes each X 3 items
= 126 bytes
= 1 Read Capacity Unit

Query cost calculation

UserId	GameId	Date	OpponentId	…
Carol	e23f5a	2013-10-08	Charlie	…
Alice	d4e2dc	2013-10-01	Bob	…
Alice	e9cba3	2013-09-27	Bob	…
Alice	f6a3bd	2013-10-08		
(397 more games for Alice)				

(1 item = 600 bytes)

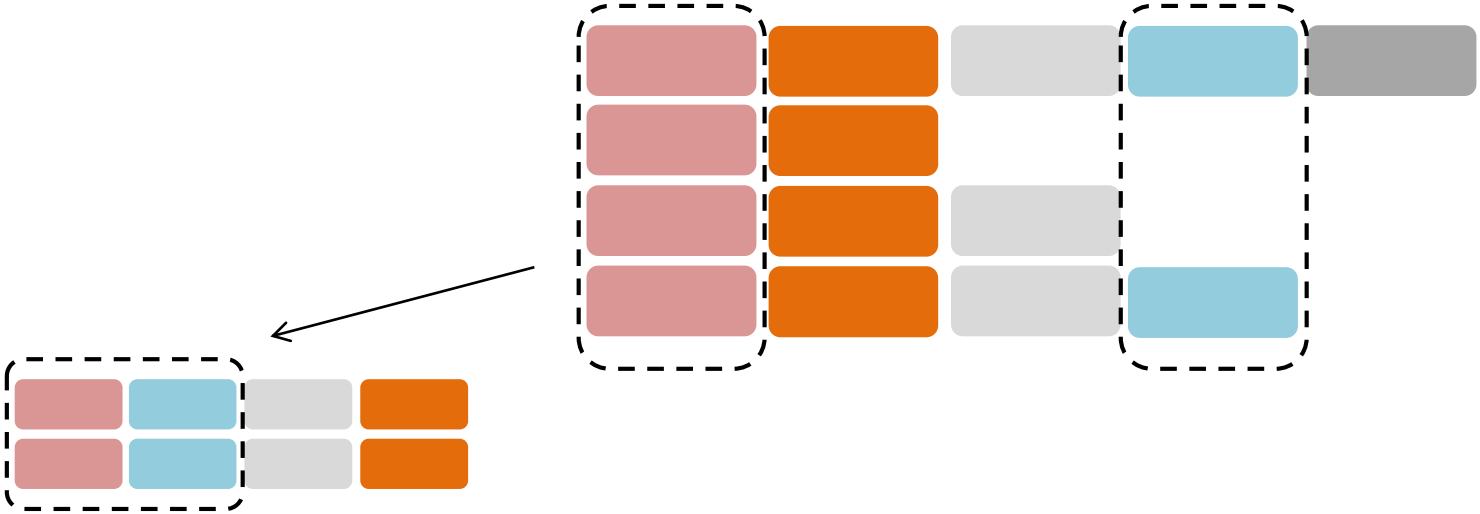
(Items evaluated by Query) (KB per Read Capacity Unit)

400 X 600 / 1024 / 4 = **60 Read Capacity Units**

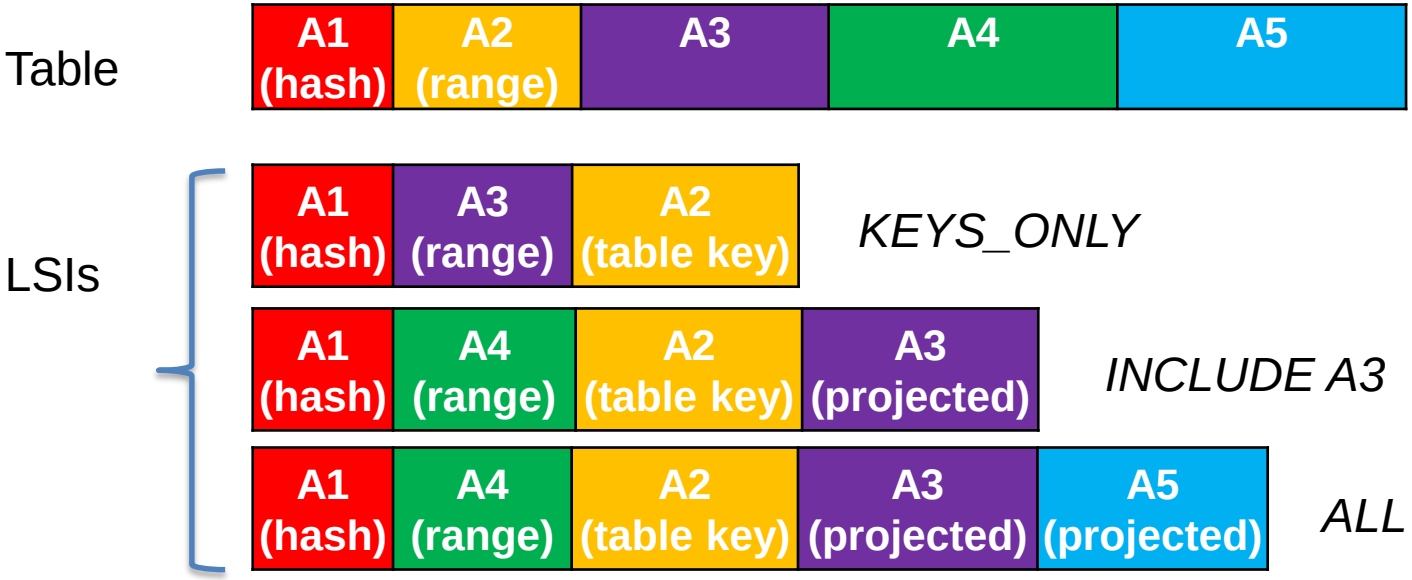
(bytes per item) (KB per byte)

Local Secondary Indexes (LSI)

- Same hash key + alternate (scalar) range key
- Index and table data is co-located (same partition)
- **Read and write capacity units consumed from the table**
- **Increases WCUs per write**
- 10GB max



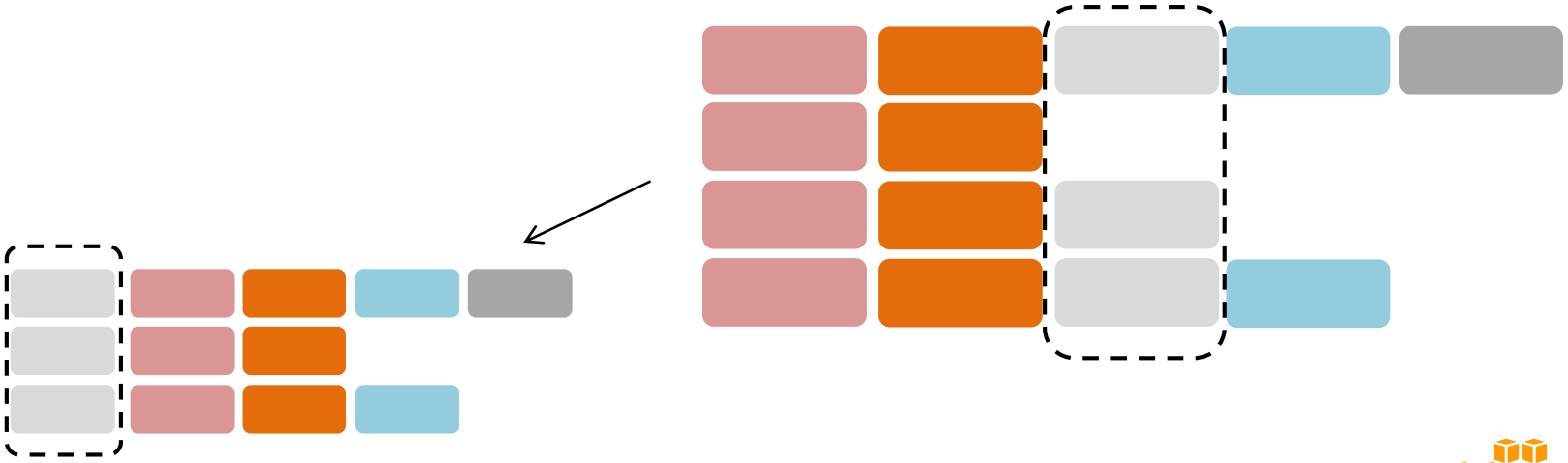
Local Secondary Index Projections



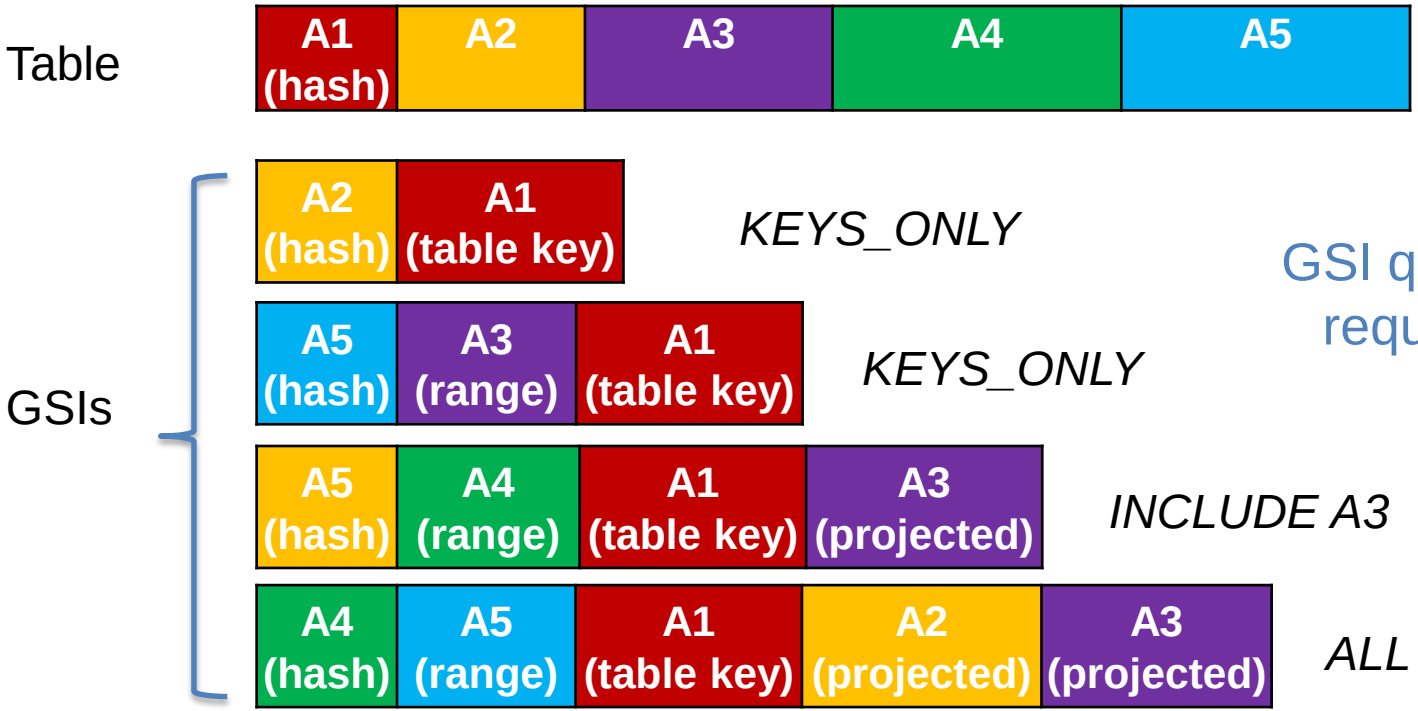
- Items with fewer projected attributes will be smaller: lower cost
- **Project with care:** LSI queries auto-fetch non-projected attributes from the table …at a cost: index matches + table matches

Global Secondary Indexes (GSI)

- Any attribute indexed as new hash and/or range key
- GSIs get their own provisioned throughput
- Asynchronously updated, so eventual consistency only



Global Secondary Index Projections



GSI queries can only request projected attributes

Reduce cost with Sparse Indexes

- DynamoDB only writes to an index if the index key value is present in the item
- Use sparse indexes to efficiently locate table items that have an uncommon attribute

- Example: find hockey teams that have won the Stanley Cup
  - GSI Hash Key: StanleyCupWinner
  - GSI Range Key: TeamId

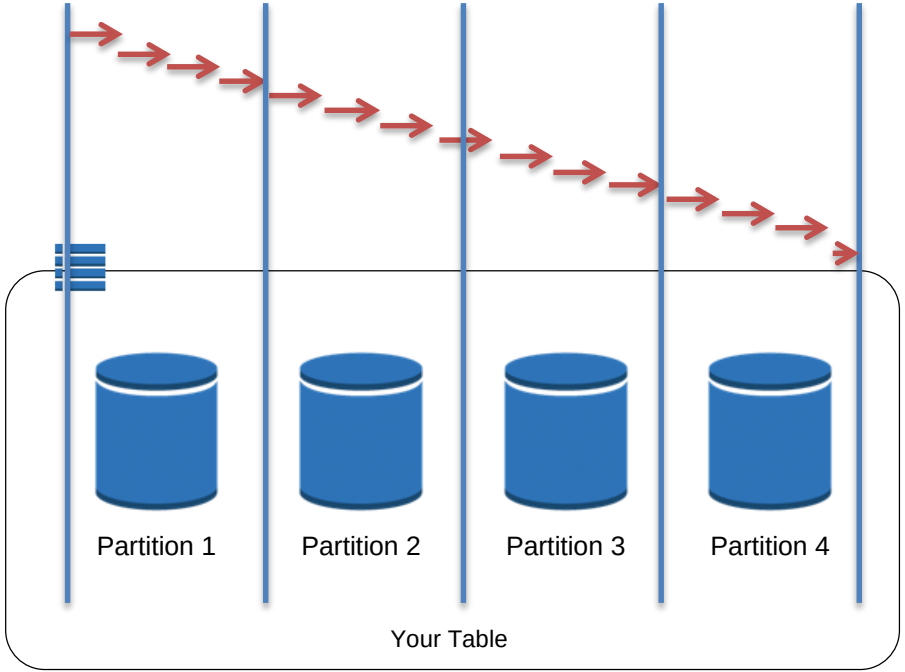
Optimizing Scans

What about non-indexed querying?

- Sometimes you need to scan the full table, e.g. bulk export
- Use Parallel Scan for faster retrieval, if…
  - Your table is >= 20 GB
  - Provisioned read throughput has room
  - Sequential Scan operations are too slow
- **Consider cost and performance implications**

Bulk Export: Scan

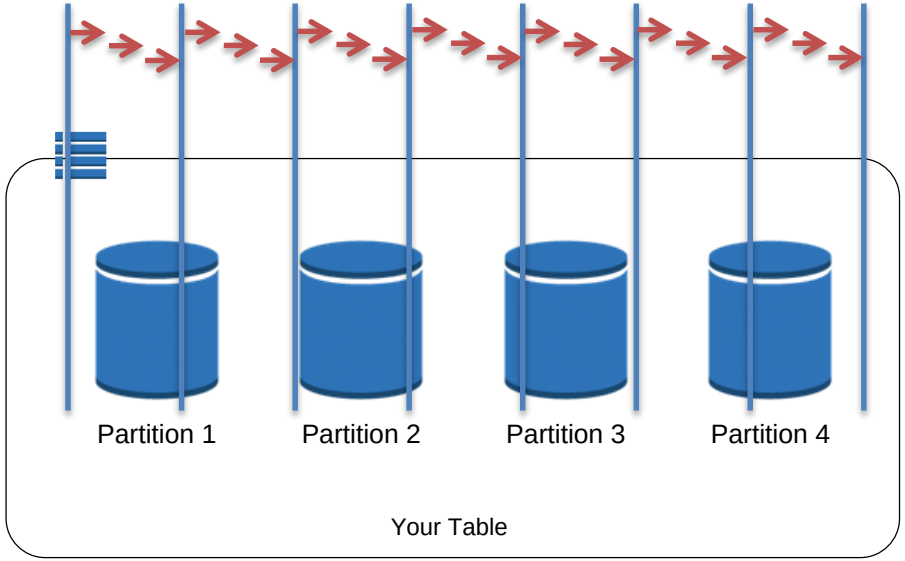
- Slow: only uses one partition at a time
- Non-uniform utilization
- Consumes lots of RCUs



Bulk Export: Parallel Scan

- Divide the work
- Use threads, non-blocking IO, processes, or multiple servers to distribute the work

- Can still consume lots of RCUs!

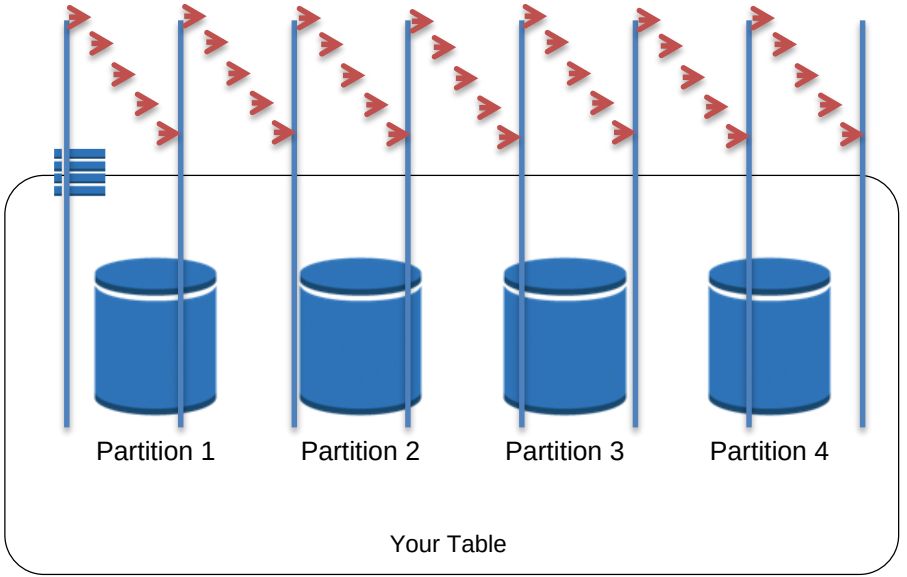


Bulk Export: Rate-limited Parallel Scan

- Limit page size and rate

  (1 MB default page size
  / 4 KB item size)
  / 2 eventually consistent reads
  = 128 RCUs per page (default)

- Rate-limit each segment



Many best practices are implemented already

- Amazon Elastic Map Reduce
  - dynamodb.throughput.*read/write*.percent
- Amazon Redshift
  - readratio
- DynamoDB Mapper (Java SDK)
- **`RateLimiter`** library (Google Guava)

Questions?

Amazon DynamoDB Deep Dive

- ✓ What's new
- ✓ JSON Document Support
- ✓ Flexible access control
- ✓ Partitions & Scaling
- ✓ Optimizing Data Access Cost
- **Customer use case: Druva**

Shekhar Deshkar
Chief Architect, Druva

Reinventing Data Protection for a Mobile World

- Druva inSync Ranked #1 by Gartner, 2 successive years
  - Top amongst 7 products against 11 critical capabilities
- Protect & govern data at the edge of enterprises
  - Extend eDiscovery & corporate data governance to endpoints
  - Data loss prevention in case of lost devices
- Sync-share for enterprise-wide collaboration
- Radically simple Cloud backup and Archival
  - Eliminates Complexity of Legacy Backup and Tape Vaulting

Data management Challenges in a Mobile World

- Cost of managing ever-growing number of endpoints or remote servers
- Controlling associated data growth
  - Should I backup *everything*?
  - Did I miss backing up something *critical*?
- Cloud?
  - How do I manage prime-time WAN bandwidth

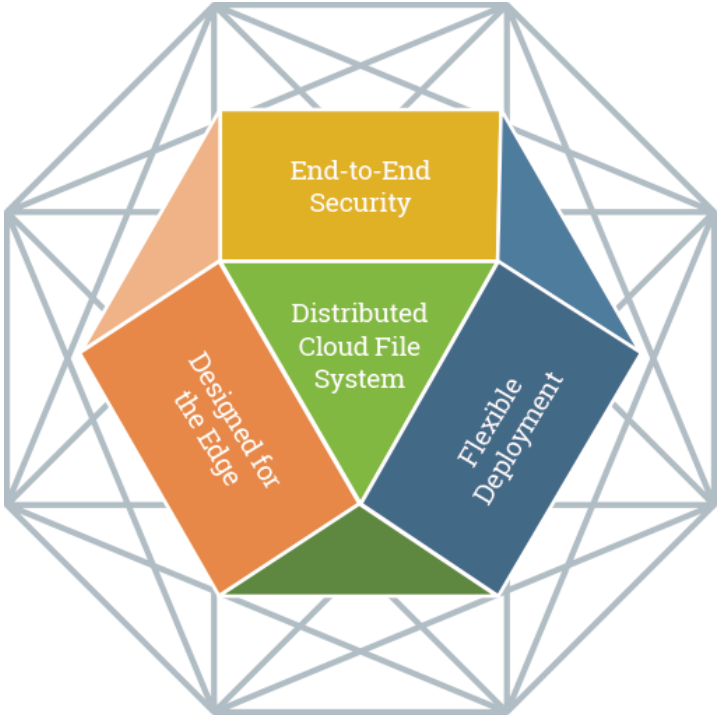
Data backup for endpoints and at the Edge

- Backup: Data & metadata
- Incremental backup
- Data deduplication
  - Global
  - Source-side

Druva Storage nCube Architecture

- Distributed Cloud File System
  - Store: Object storage, single instance, self-healing
  - Organize: Time-indexed views, metadata server
  - Serve: Uncompromised throughput
- End-to-end Security
  - At the ends: 2-factor encryption; At device
  - Everything in between: Data in transit, Data access
- Designed for the Edge
  - Intelligent, global de-duplication at the edge
- Flexible deployment
- Why we moved away from Cassandra:

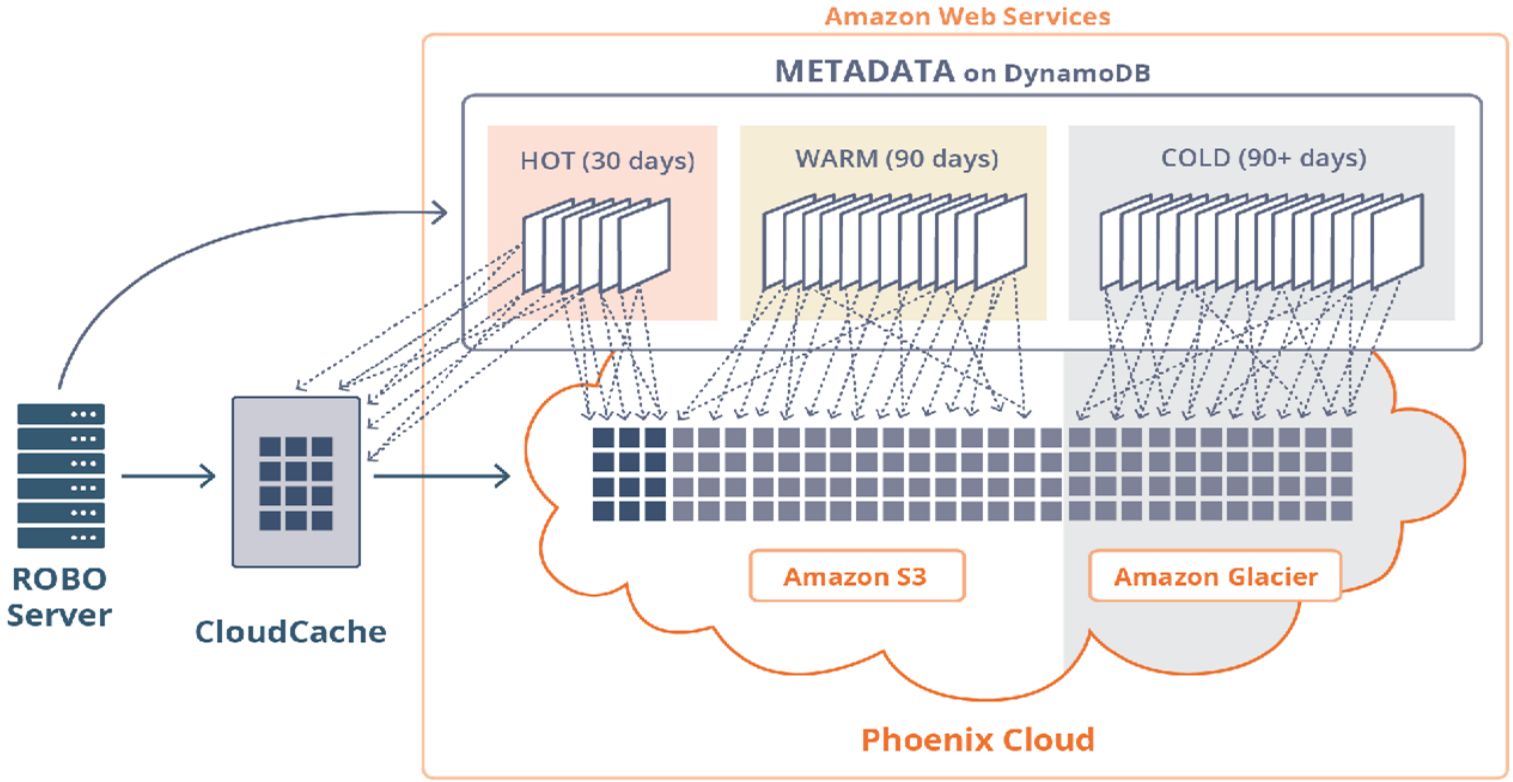
  http://aws.amazon.com/solutions/case-studies/druva/



druva

amazon web services

Druva Storage on AWS



druva

amazon web services

NoSQL Database & Distributed Systems

Scenario: Collaborators trying to create same file in shared work-space

- Need to simulate atomic operations that ensures only one of them succeeds
- Each collaborator creates an object with a unique temporary name
- Perform atomic rename (from unique temporary to possibly-conflicting name) to ensure only one succeeds.

```
item = self.db.new_item(self.DRSTK_FOLDER_NAMEVER,
                        key, {self.AT_DIRENT: val})
try:
     item.put(expected_value={self.AT_DIRENT: False})

except adb.ConditionCheckFailedError:
     # someone else raced against us and
     # already created a folder entry we're hoping to create
```

druva

amazon web services

Transactional semantics for References

- How do we implement transaction-like semantics while updating two independent DynamoDB items?
- Thread #A is trying to add reference to an existing item, stored as an object in S3, has checksum ‘csum’, and is identified by ‘handle’
- Thread #B, is in process of dropping it’s reference to the same object and if no more references, removes the item and associated S3 object too.

Transactional semantics for References

- Thread A: Optimistically, add a new reference to the handle

```
hkey, rkey = self.handle2key(handle)
item = self.db.new_item(self.id_ref + hkey, rkey + reverselink,{})

item.put()

chkey, crkey = self.sum2key(csum)
try:
      citem = self.db.get_item(chkey, crkey + rkey)
except adb.KeyNotFoundError:
      # we lost race & handle vanished, drop our reference
      item.delete()
else: # success
```

Label #1 thread B may be forced to take Label #5 path

Label #2

druva

amazon web services

Transactional semantics for References

- Thread B Drop a reference to the handle

```
item = self.db.new_item(self.id_ref + hkey, rkey + reverselink)
item.delete()
items = self.db.query(self.id_ref + hkey, query, limit=1)
try:
    item = items.next()
except StopIteration: # no more references to the handle
else: # Additional references exist, can't purge
        return 0
# No one possibly using this block
chkey, crkey = self.sum2key(csum)
item = self.db.new_item(chkey, crkey + rkey)
item.delete()
items = self.db.query(self.id_ref + hkey, query, limit=1)
try:
    item = items.next()
except StopIteration: # safe to purge the block from S3 now
else: # Someone is now using this. Don't drop block
```

Label #3 thread A may be forced to take Label #2 path

Label #4

Label #5

Designing for Scale-Out Performance

- DynamoDB hot partitions
- Lock-free concurrent access
  - Minimize shared state
  - Identify and eliminate false-sharing
  - Distinct hash keys
- Relax semantics for shared state
  - Shard hash key to eliminate DynamoDB hotspots
  - Collate shard state to determine global state

Provisioned Throughput Optimization

- Application IOPs to sustain
  - DynamoDB IOPs needed to support it
- Shield applications from Throttling errors
- Optimize Provisioned Throughput based on History

Questions?

Poll

Thank you!

https://aws.amazon.com/dynamodb/developer-resources/
http://druva.com/