Amazon DynamoDB Design Workshop

Sean Shriver
NoSQL Solutions Architect, AWS

Data Modeling

Hierarchical data structures as items

- Use composite sort key to define a hierarchy
- Highly selective result sets with sort key queries
- Index anything, scales to any size

	Primary Key		Attributes						
	ProductID	type							
Items	1	bookID	title	author	genre	publisher	datePublished	ISBN	
			Some Book	John Smith	Science Fiction	Ballantine	Oct-70	0-345-02046-4	
	2	albumID	title	artist	genre	label	studio	released	producer
			Some Album	Some Band	Progressive Rock	Harvest	Abbey Road	3/1/73	Somebody
	2	albumID:trackID	title	length	music	vocals			
			Track 1	1:30	Mason	Instrumental			
	2	albumID:trackID	title	length	music	vocals			
			Track 2	2:43	Mason	Mason			
	2	albumID:trackID	title	length	music	vocals			
			Track 3	3:30	Smith	Johnson			
	3	movieID	title	genre	writer	producer			
			Some Movie	Scifi Comedy	Joe Smith	20th Century Fox			
	3	movieID:actorID	name	character	image				
			Some Actor	Joe	img2.jpg				
	3	movieID:actorID	name	character	image				
			Some Actress	Rita	img3.jpg				
	3	movieID:actorID	name	character	image				
			Some Actor	Frito	img1.jpg				

… or as documents (JSON)

- JSON data types (M, L, BOOL, NULL)
- Document SDKs available
- 400 KB maximum item size (limits hierarchical data structure)

	Primary Key	Attributes							
	ProductID								
Items	1	id	title	author	genre	publisher	datePublished	ISBN	
		bookID	Some Book	Some Guy	Science Fiction	Ballantine	Oct-70	0-345-02046-4	
	2	id	title	artist	genre	Attributes			
		albumID	Some Album	Some Band	Progressive Rock	{ label:"Harvest", studio: "Abbey Road", published: "3/1/73", producer: "Pink Floyd", tracks: [{title: "Speak to Me", length: "1:30", music: "Mason", vocals: "Instrumental"},{title: ”Breathe", length: ”2:43", music: ”Waters, Gilmour, Wright", vocals: ”Gilmour"},{title: ”On the Run", length: “3:30", music: ”Gilmour, Waters", vocals: "Instrumental"}]}			
	3	id	title	genre	writer	Attributes			
		movieID	Some Movie	Scifi Comedy	Joe Smith	{ producer: "20th Century Fox", actors: [{ name: "Luke Wilson", dob: "9/21/71", character: "Joe Bowers", image: "img2.jpg"},{ name: "Maya Rudolph", dob: "7/27/72", character: "Rita", image: "img1.jpg"},{ name: "Dax Shepard", dob: "1/2/75", character: "Frito Pendejo", image: "img3.jpg"}]			

1:1 relationships or key-values

- Use a table or GSI with a partition key
- Use GetItem or BatchGetItem API

Example: Given a user or email, get attributes

Users Table	
Partition key	Attributes
UserId = bob	Email = bob@gmail.com, JoinDate = 2011-11-15
UserId = fred	Email = fred@yahoo.com, JoinDate = 2011-12-01

Users-Email-GSI	
Partition key	Attributes
Email = bob@gmail.com	UserId = bob, JoinDate = 2011-11-15
Email = fred@yahoo.com	UserId = fred, JoinDate = 2011-12-01

1:N relationships or parent-children

- Use a table or GSI with partition and sort key
- Use Query API

Example:

Given a device, find all readings between epoch X, Y

Device-measurements		
Part. Key	Sort key	Attributes
DeviceId = 1	epoch = 5513A97C	Temperature = 30, pressure = 90
DeviceId = 1	epoch = 5513A9DB	Temperature = 30, pressure = 90

N:M relationships

- Use a table and GSI with partition and sort key elements switched
- Use Query API

Example:

Given a user, find all games. Or given a game, find all users.

User-Games-Table	
Part. Key	Sort key
UserId = bob	GameId = Game1
UserId = fred	GameId = Game2
UserId = bob	GameId = Game3

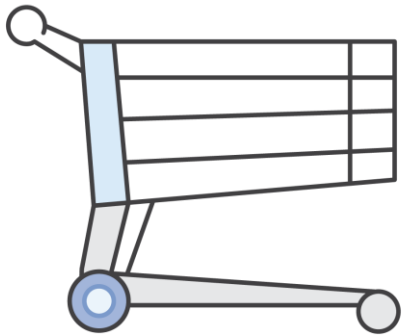
Game-Users-GSI	
Part. Key	Sort key
GameId = Game1	UserId = bob
GameId = Game2	UserId = fred
GameId = Game3	UserId = bob

Data Modeling Exercise #1

- A shopping cart use case
  - Attributes: cartId, dateTime, set of SKU's, etc.
  - Mostly under 1KB in size
  - Up to 100 million carts in the table – 100GB total
  - Accessed by cartId (key-value access pattern)
  - 10K writes, and 10K reads per second
  - TTL to expire "old" carts
- Also:
  - Notify when there is a price change for a SKU in a cart

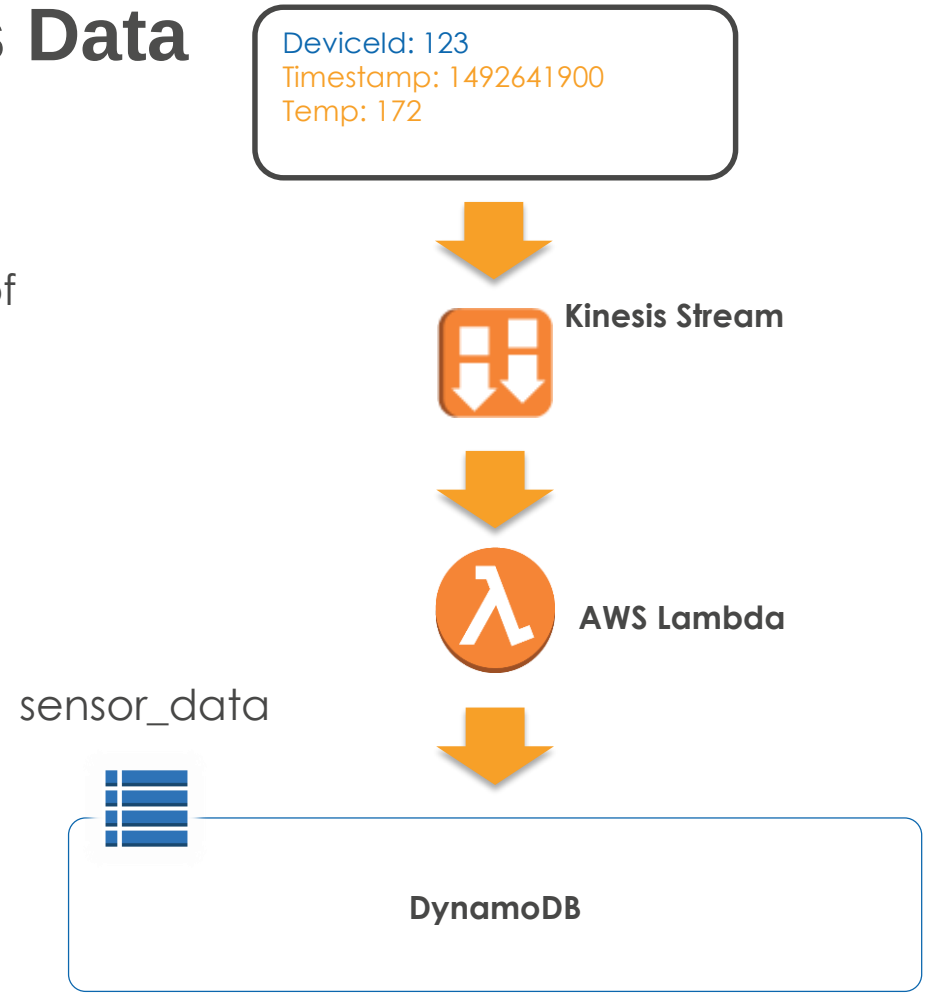
Cost estimation

- The table: cartId, dateTime, set of SKU's, etc.
  - Under 1KB in size
  - Up to 100 million carts in the table – up to 100GB total
  - Accessed by cartId (key-value access pattern)
  - 10K writes, and 10K reads per second
  - Cost estimate
    - Enter the storage, item size, reads, and writes
    - Get an estimate of your monthly bill: ~$6300
- The price notification job
  - E.g. 100 million items, scan/update over 24 hrs
  - ~ 1200 RCU and WCU for 24 hrs: ~ $22



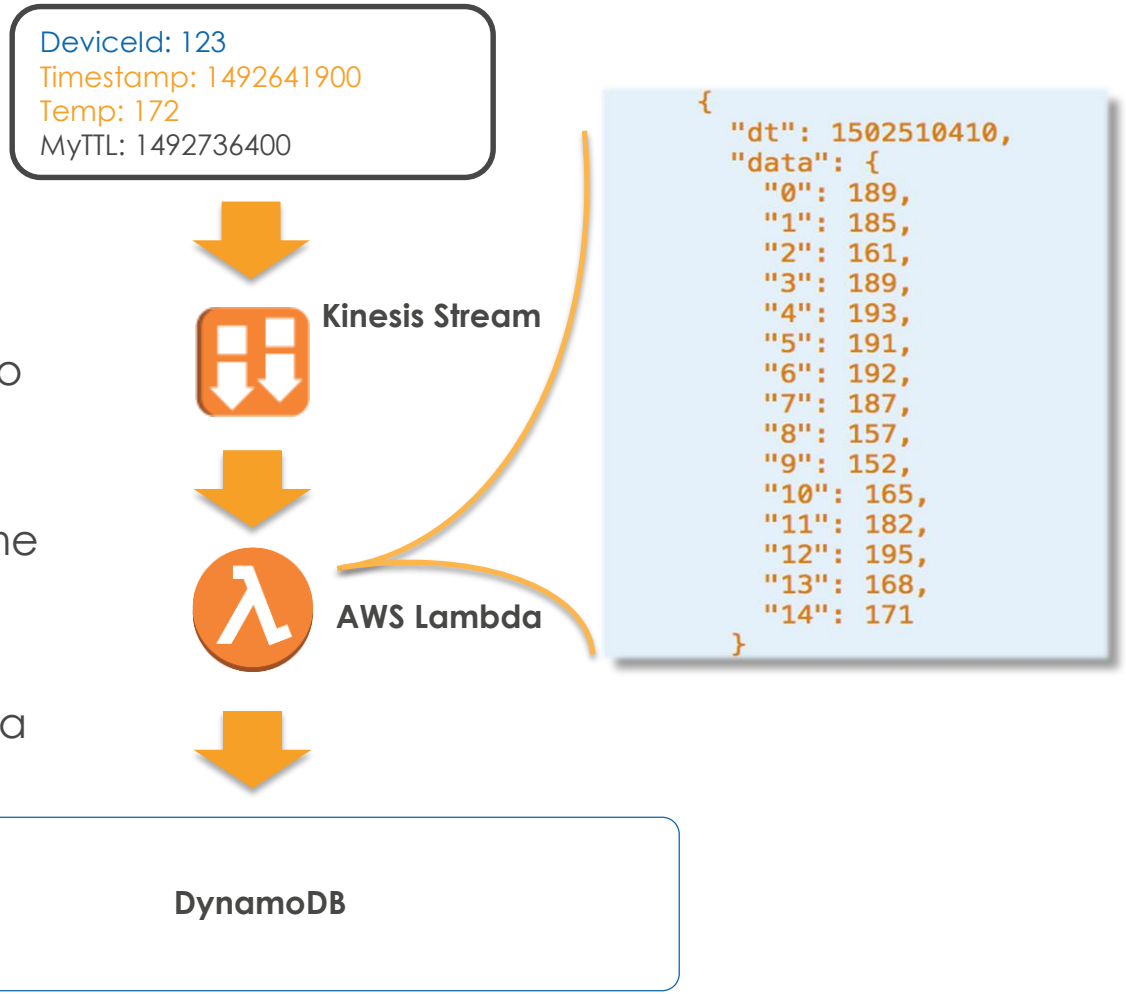
Use case: Time Series Data

- Sensor data (temp) from 1000's of sensors
- Need to store for fast access
- Access by sensor ID + time range
- Store for 90 days

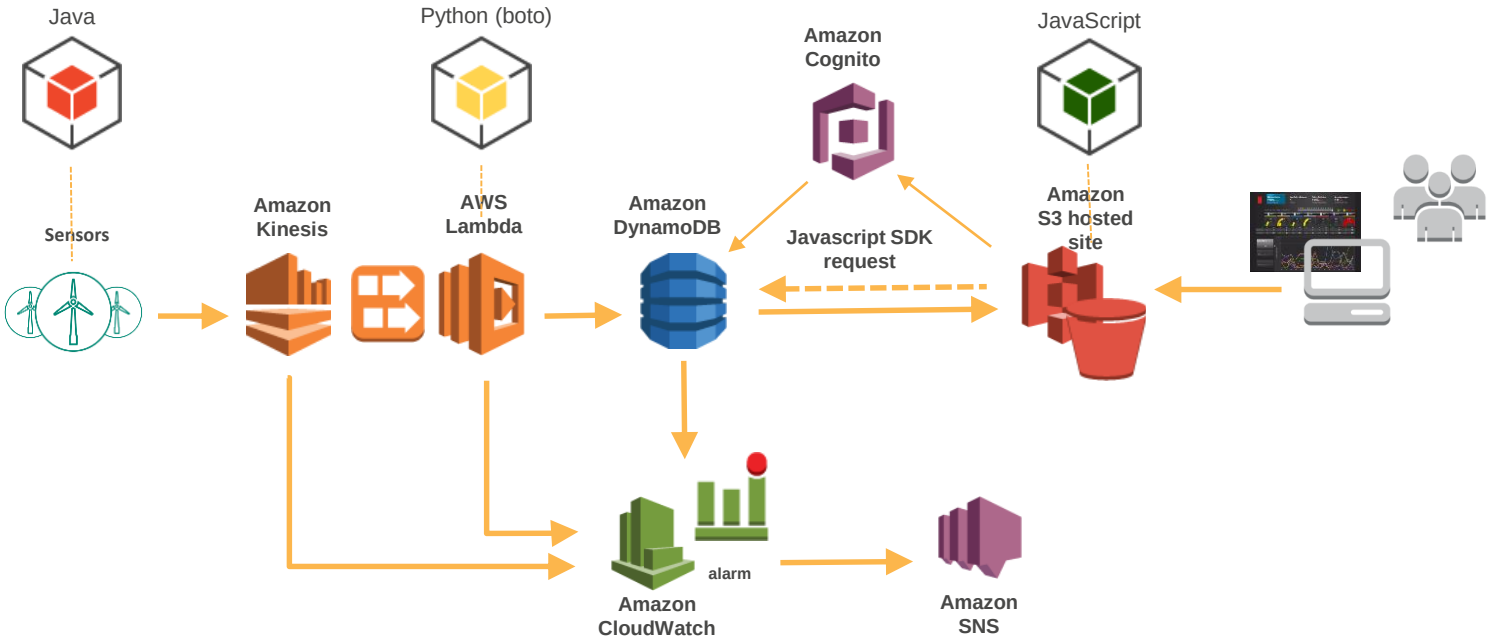


Solution

- ✓ Group multiple data points into a single item
- ✓ Save on writes
- ✓ Use TTL to manage data lifetime



Time Series Data Example Architecture



Time Series Data

- ✓ High volume ingest: Kinesis + DynamoDB
- ✓ Fast access: DynamoDB
- ✓ At a reasonable cost: <$10K/month for:
  - 2.5TB of stored data points per month
  - Ingest of 100K data points per second

All using a serverless architecture with managed services on AWS

Cost estimate

- Data rate: 100000 data points per second
- Data storage: 1 month’s worth = ~2.5TB

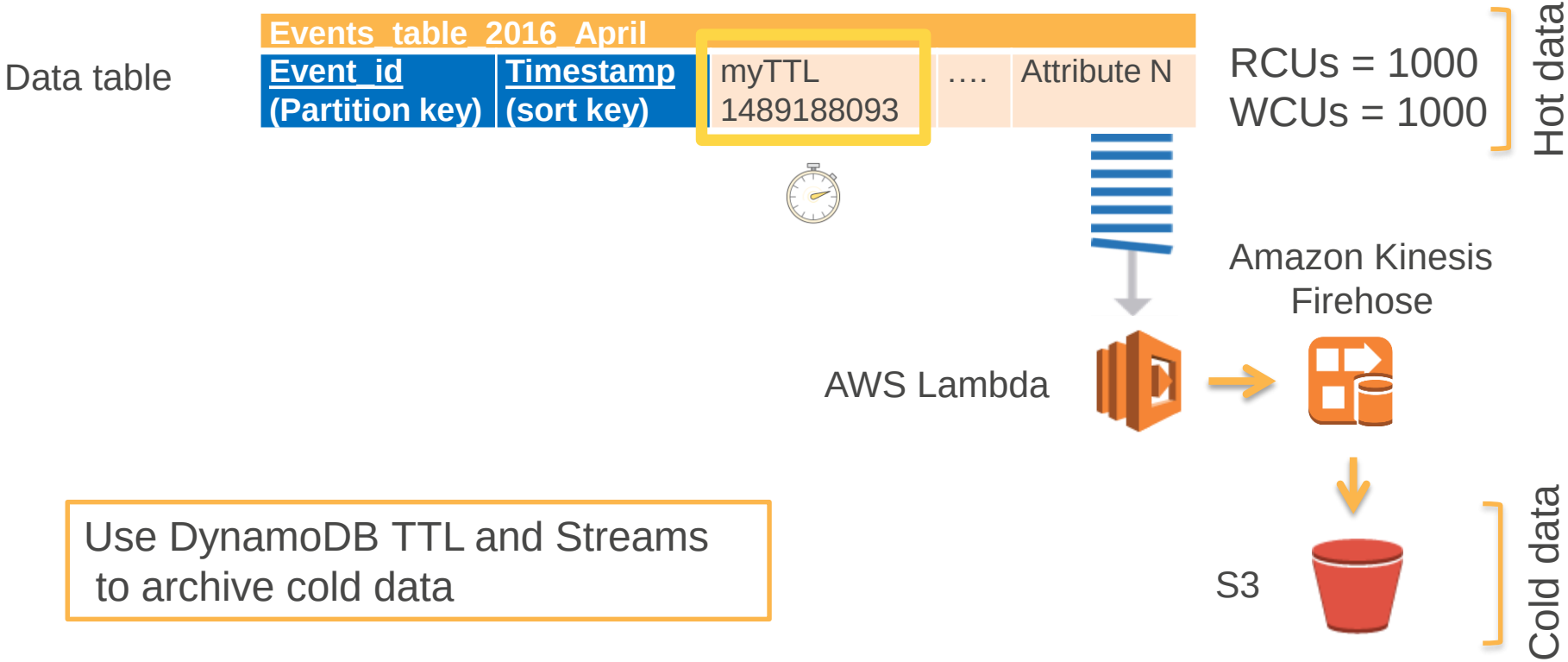
Kinesis: 100000 records -> 100 shards -> ~$5K per month

DynamoDB: 100000 WCU’s -> $50K per month

- Binned writes: 10000 WCU’s -> $5000 per month (10x less…)
- Diff. over 1 year: $600K vs. $60K

We can save a lot by storing multiple data points per item

Using TTL to age out cold data



Use DynamoDB TTL and Streams
to archive cold data

Age out cold data

- Configure a TTL attribute for table
- For each item, set it to item expiration date/time
- DynamoDB automatically deletes expired items
- Use DynamoDB Streams to act on expired items

Important when: Dealing with data that gets cold over time (all data!)

Use case: Messaging App

Messages app

Inbox

```
SELECT *
FROM Messages
WHERE Recipient='David'
LIMIT 50
ORDER BY Date DESC
```

Messages table

Outbox

```
SELECT *
FROM Messages
WHERE Sender ='David'
LIMIT 50
ORDER BY Date DESC
```

Solution v.1

David

Inbox

```
SELECT *
FROM Messages
WHERE Recipient='David'
LIMIT 50
ORDER BY Date DESC
```

Partition key

Sort key

Messages table

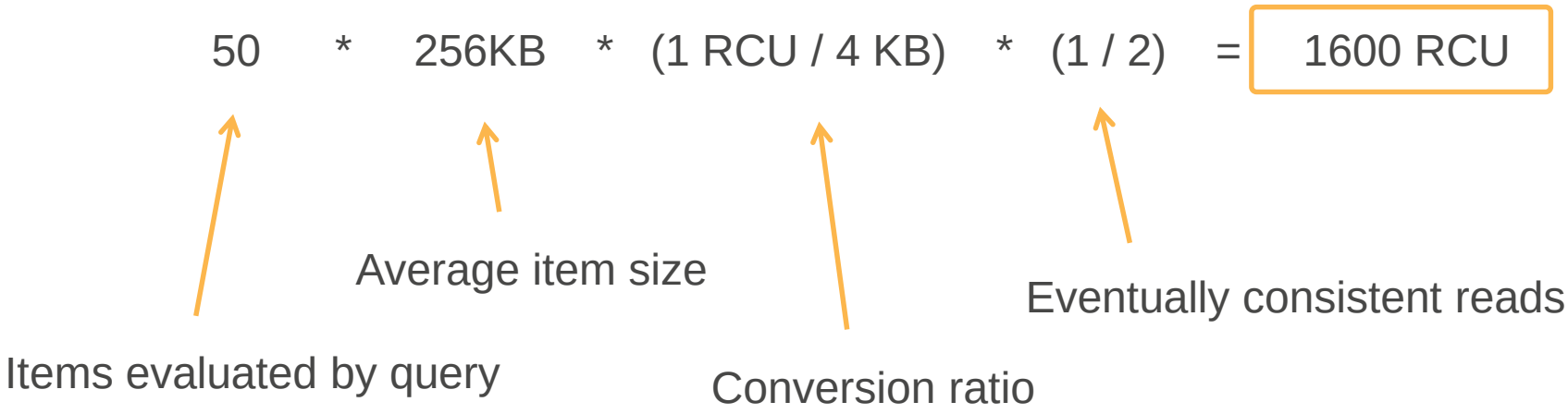
Recipient	Date	Sender	Message
David	2014-10-02	Bob	…
… 48 more messages for David …			
David	2014-10-03	Alice	…
Alice	2014-09-28	Bob	…
Alice	2014-10-01	Carol	…

(Many more messages)

50 items × 256 KB each

Large message bodies
Attachments

Computing inbox query cost



Solution v.2: Separate the bulk data

Query inbox-GSI: 50 * 128B * (1 RCU / 4 KB) * (1 / 2) = 1 RCU

(50 sequential items at 128 bytes)

Inbox-GSI

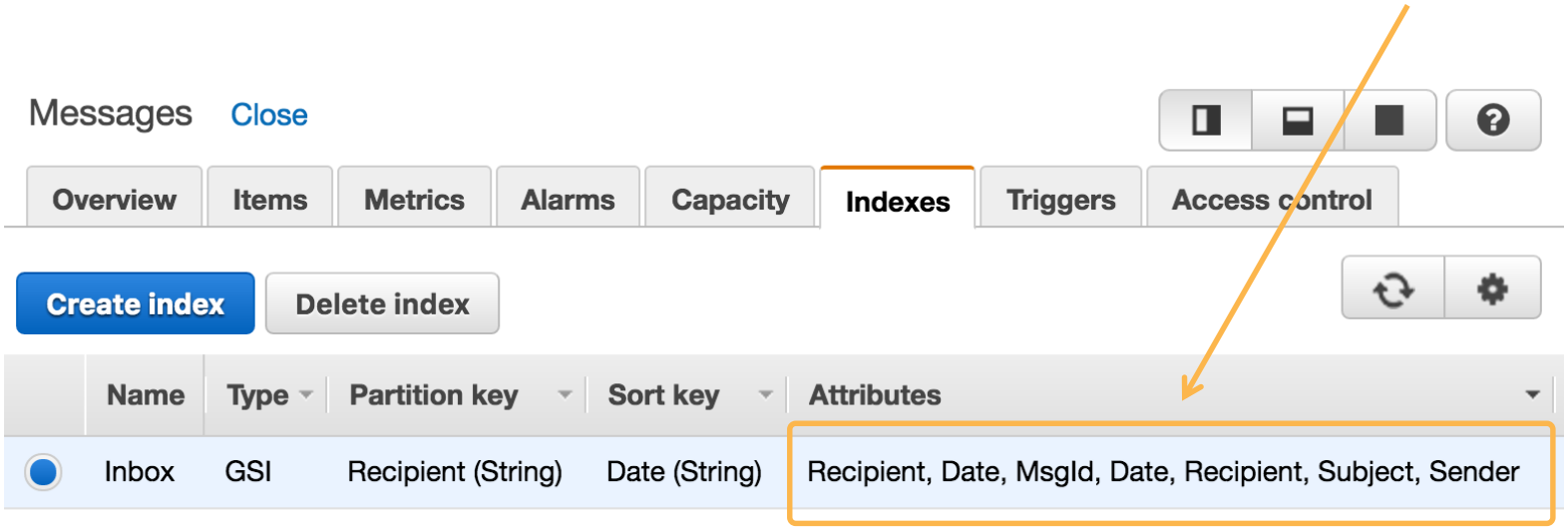
Recipient	Date	Sender	Subject	MsgId
David	2014-10-02	Bob	Hi!…	afed
David	2014-10-03	Alice	RE: The…	3kf8
Alice	2014-09-28	Bob	FW: Ok…	9d2b
Alice	2014-10-01	Carol	Hi!...	ct7r

Messages table

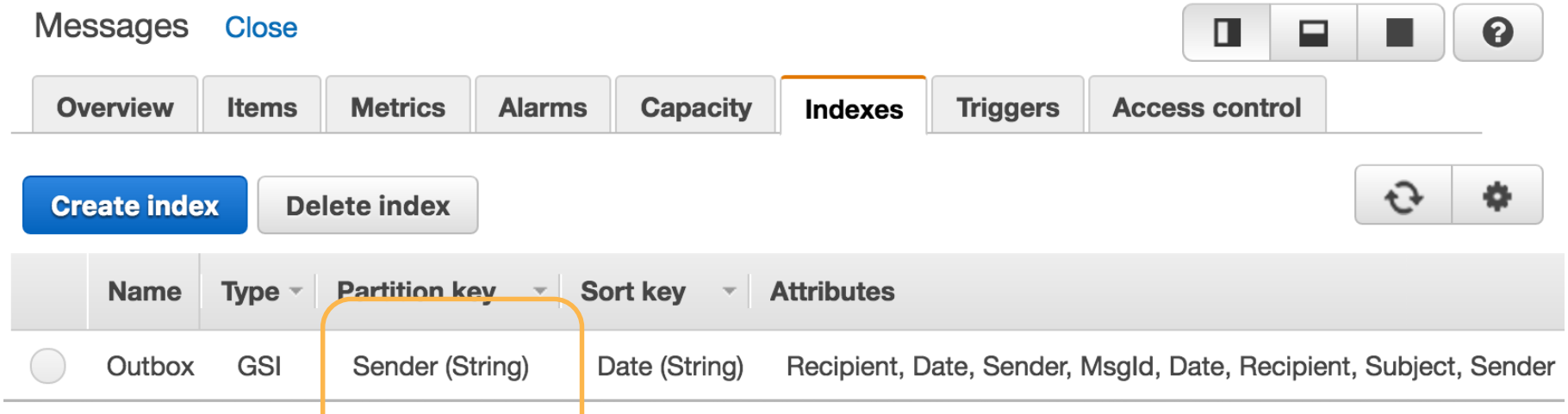
MsgId	…
9d2b	…
3kf8	…
ct7r	…
afed	…

Inbox GSI

Define which attributes to copy into the index

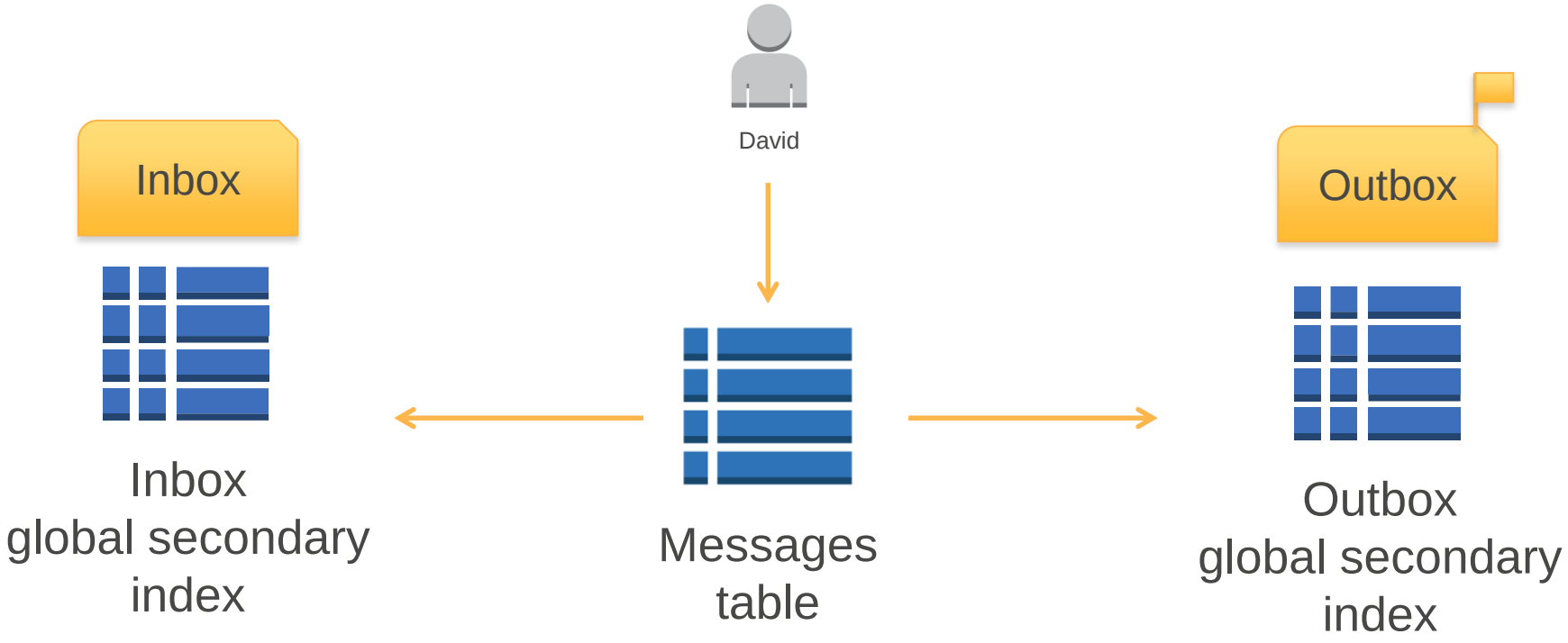


Outbox GSI

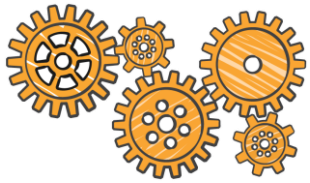



```
SELECT *
FROM Messages
WHERE Sender ='David'
LIMIT 50
ORDER BY Date DESC
```

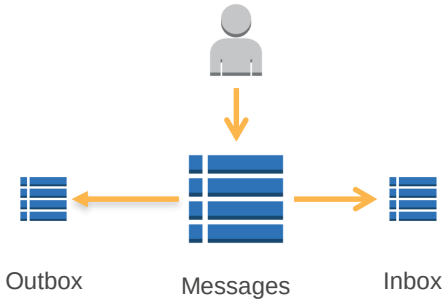
Messaging app



Distribute large items

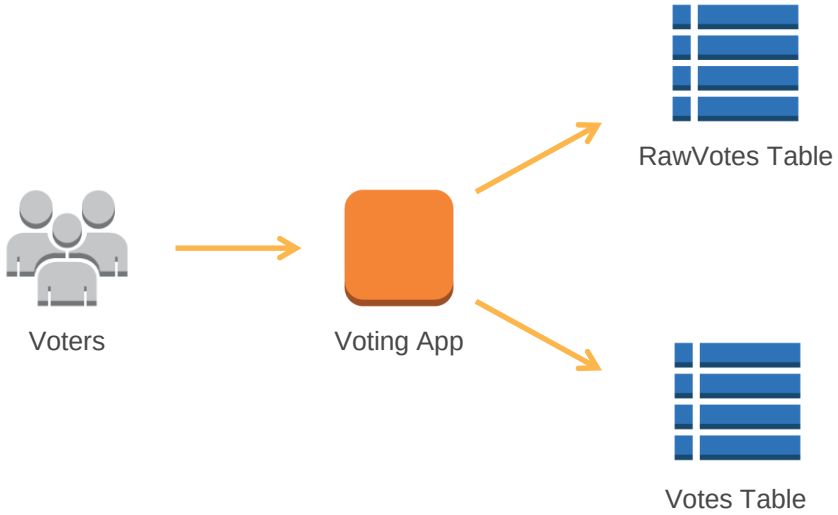


- Reduce one-to-many item sizes
- Configure secondary index projections
- Use GSIs to model M:N relationship between sender and recipient
- Use GSIs to select the right subset of data for better performance and lower cost

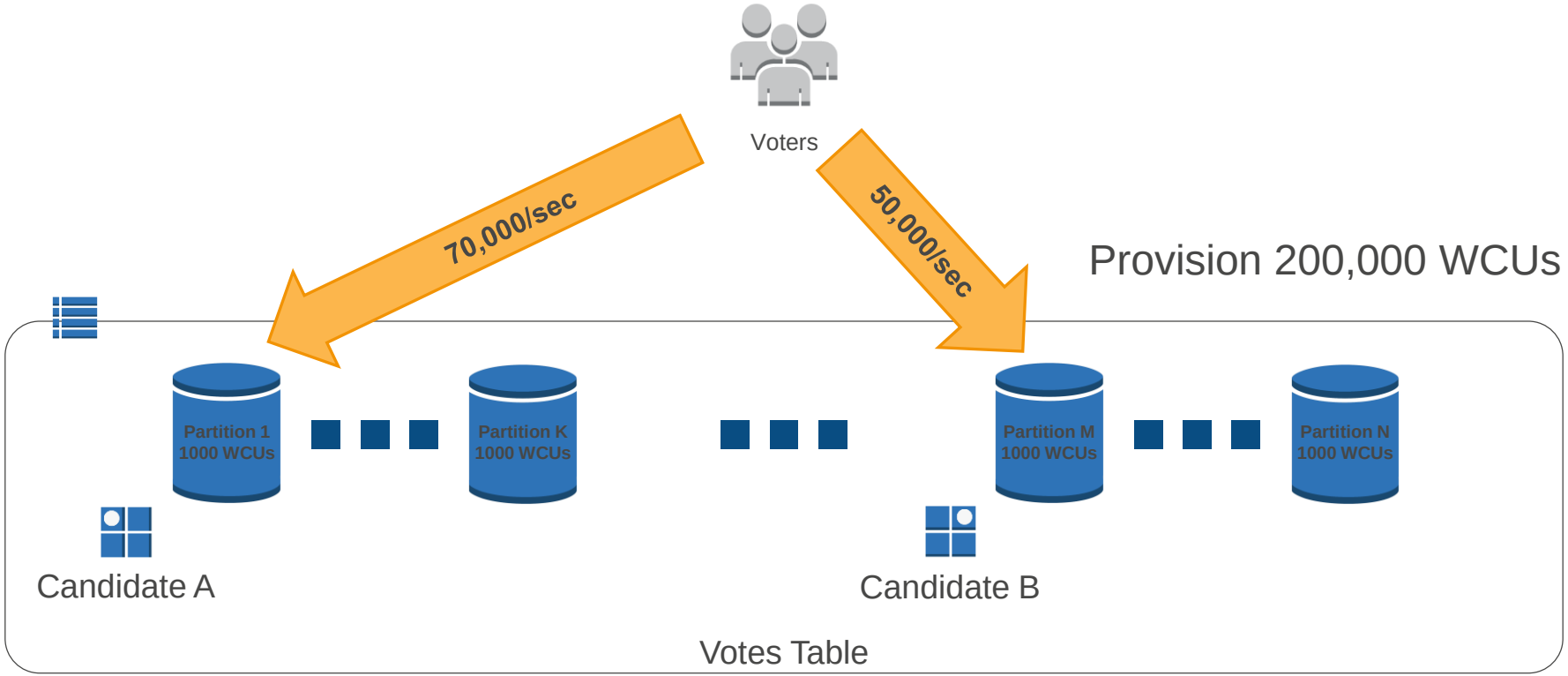


Important when: Querying many large items at once

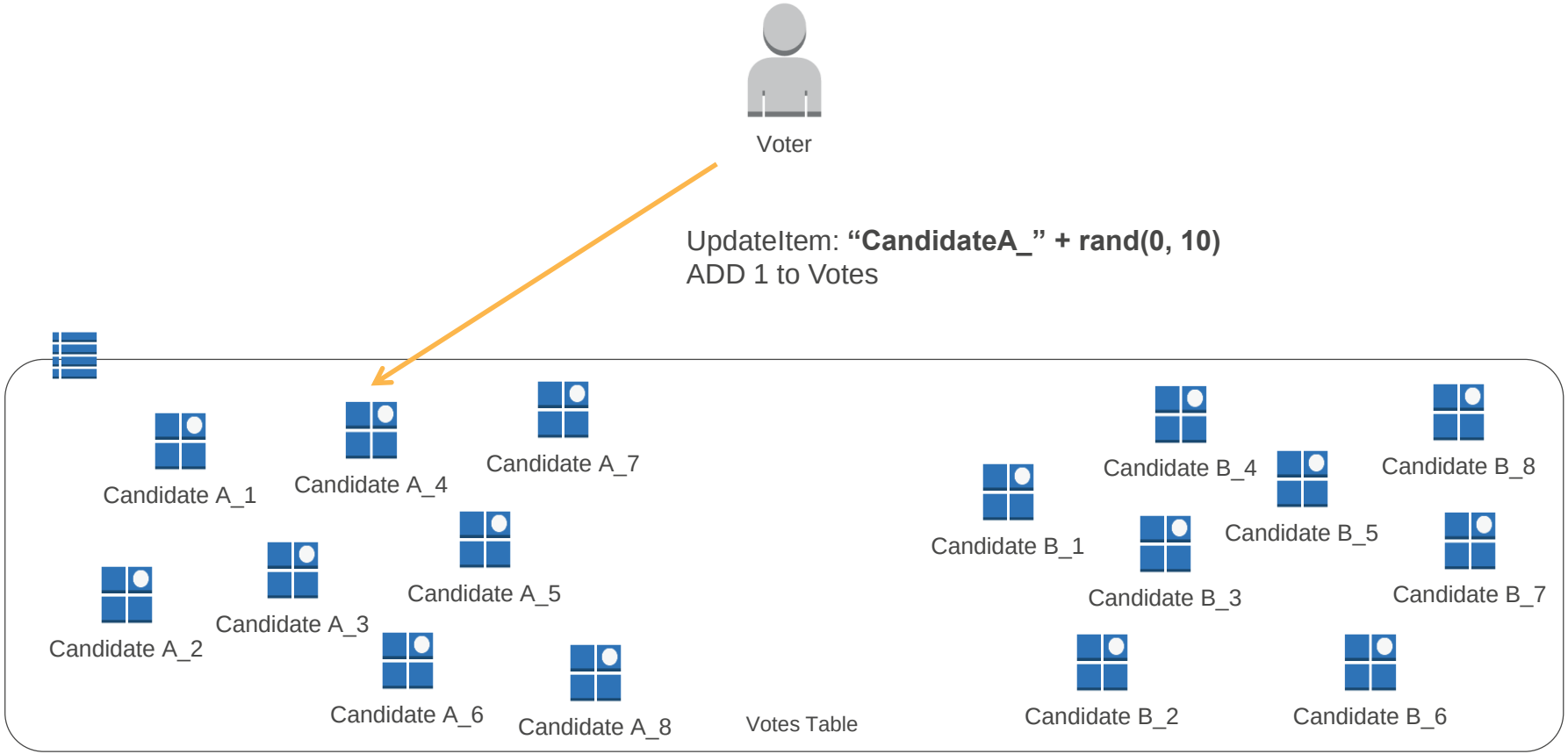
Use case: Real-time voting



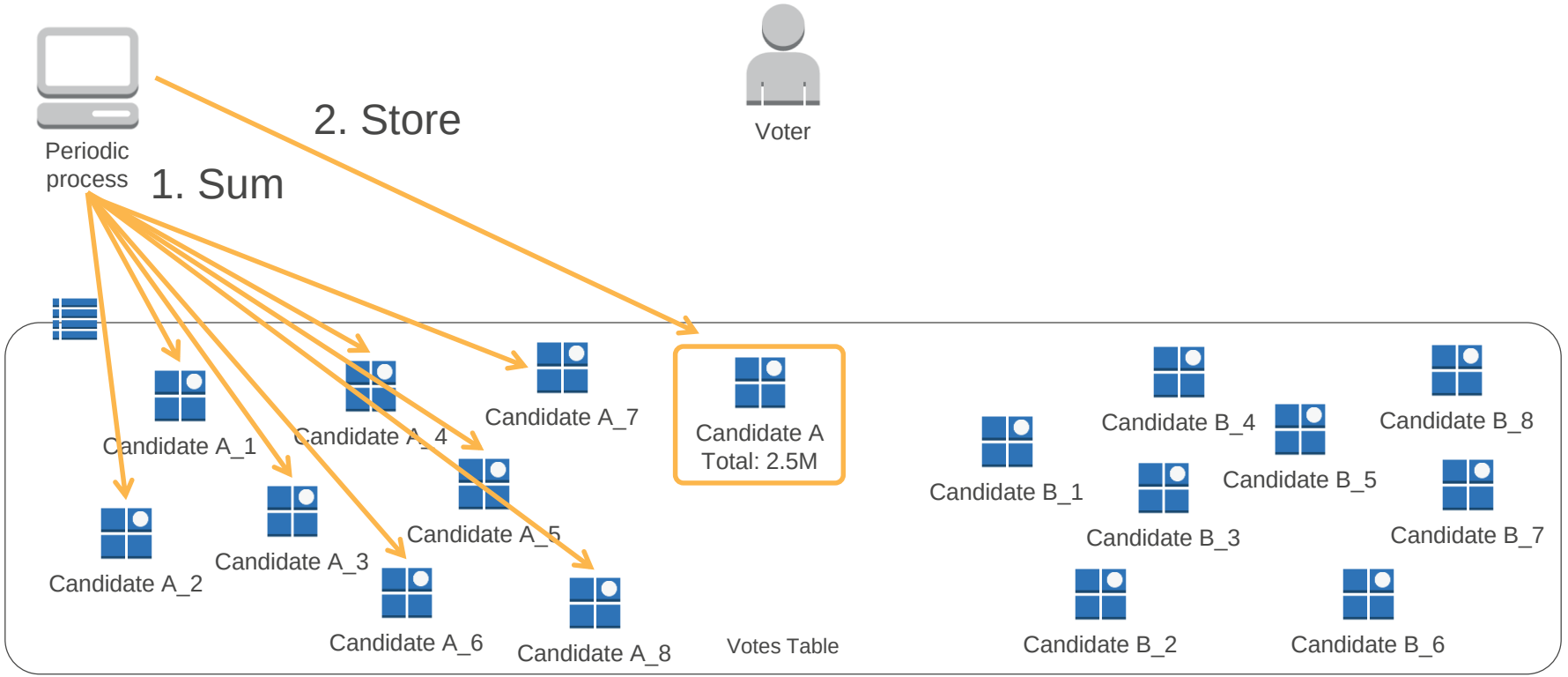
Real-time voting: scaling bottlenecks



Write-sharding



Shard aggregation



Real-time voting: tables

1. Record vote and de-dupe

RawVotes Table

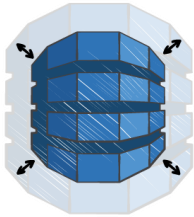
UserId	Candidate	Date
Alice	A	2016-10-02
Bob	B	2016-10-02
Eve	B	2016-10-02
Chuck	A	2016-10-02

2. Increment candidate counter

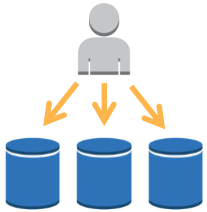
Votes Table

Segment	Votes
A_1	23
B_2	12
B_1	14
A_2	25

Shard write-heavy partition keys

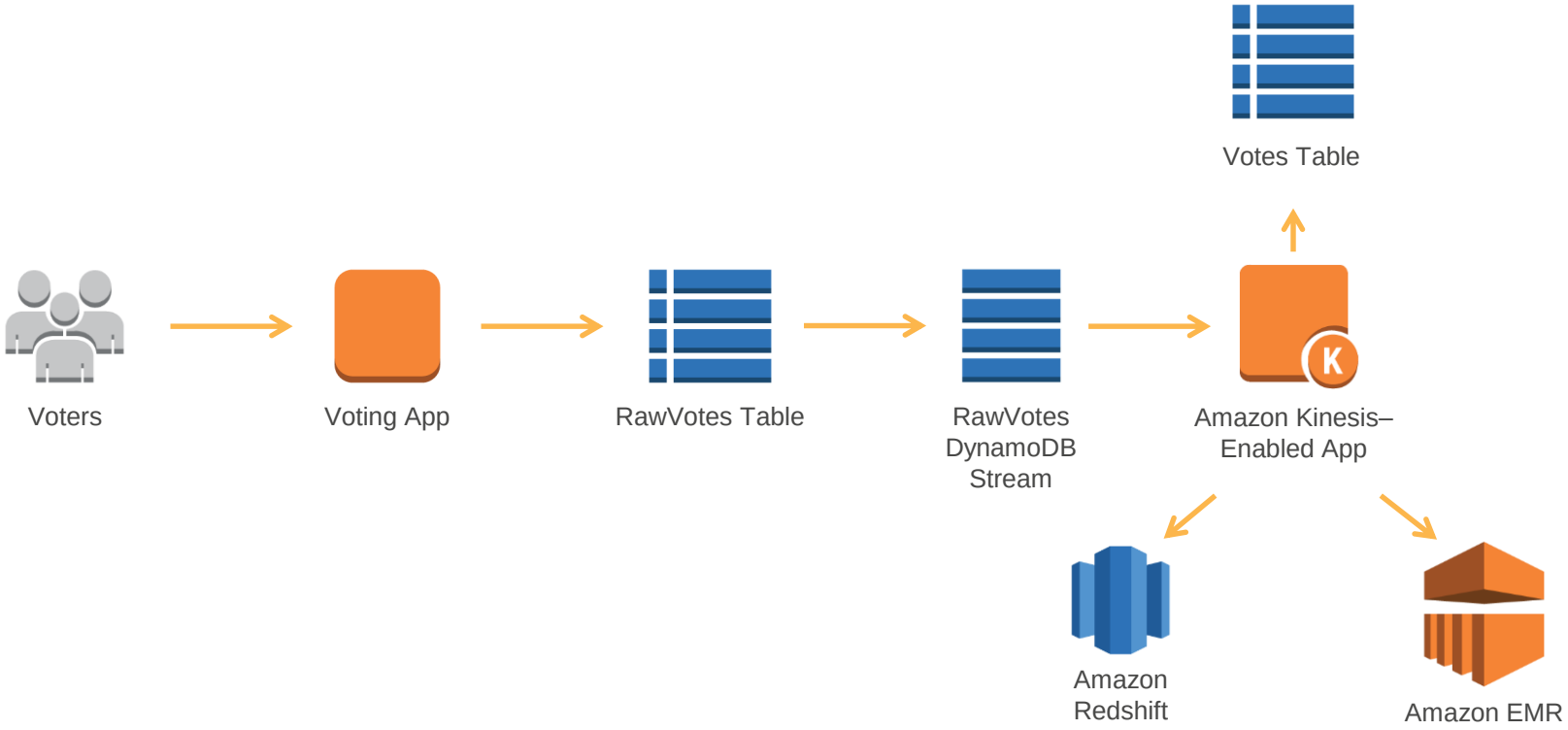


- Trade off read cost for write scalability
- Consider throughput per partition key



Important when: Your write workload is not horizontally scalable

Real-time voting v2: aggregation with Streams



Real-time voting v2: tables

1. Record vote and de-dupe

RawVotes Table

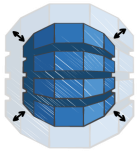
UserId	Candidate	Date
Alice	A	2016-10-02
Bob	B	2016-10-02
Eve	B	2016-10-02
Chuck	A	2016-10-02

2. Increment candidate counter

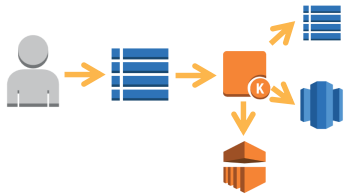
Votes Table

Candidate	Votes
A	23
B	12
C	14
D	25

Analytics with DynamoDB Streams



- Collect and de-dupe data in DynamoDB
- Aggregate data in-memory and flush periodically



Important when: Performing real-time aggregation and analytics

Everything and Anything Startups Need to Get Started on AWS

aws.amazon.com/activate

Thank you!