Leveraging Determinism

Frank Yu

Trusting Deterministic Execution to Stabilize, Scale, and Simplify Systems

If you have some **important logic**

Make it **deterministic**

If you have **deterministic** logic

Don’t be afraid to **replay it anywhere** for efficiency and profit

About Us and Our Problems


United States

Trading Exchanges must

- Be Correct
- Have Consistent and Predictable Performance
- Remember Everything for Auditability

How can the system evolve safely and efficiently while performing?

Make sure your core is simple

Not Simple: Tangled Web of Services

Simple: One Well Tested Service

Not Simple: Concurrency and Non Deterministic Execution

Simple: Deterministic Execution

Think of a program as a state machine

$$\text{Input} \times \text{State} \rightarrow \text{State}$$

$$\text{Input} \times \text{State} \rightarrow \text{Output}$$

$$\text{Requests} \times \text{State} \rightarrow \text{State}$$

$$\text{Input} \times \text{State} \rightarrow \text{Output}$$

$$\text{Requests} \times \text{State} \rightarrow \text{New State}$$

$$\text{Input} \times \text{State} \rightarrow \text{Output}$$

$$\text{Requests} \times \text{State} \rightarrow \text{New State}$$

$$\text{Requests} \times \text{State} \rightarrow \text{Output}$$

$$\text{Requests} \times \text{State} \rightarrow \text{New State}$$

$$\text{Requests} \times \text{State} \rightarrow \text{Events}$$

Ordered Inputs

+

Deterministic Execution

$\rightarrow$

Same State and Outputs

Sequenced Requests

+

Deterministic Execution

$\rightarrow$

Same State and Outputs

Sequenced Requests

+

Determinism

$\rightarrow$

Same State and Outputs

Sequenced Requests

+

Determinism

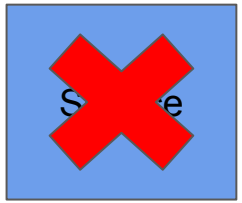
→

Replicated State and Events

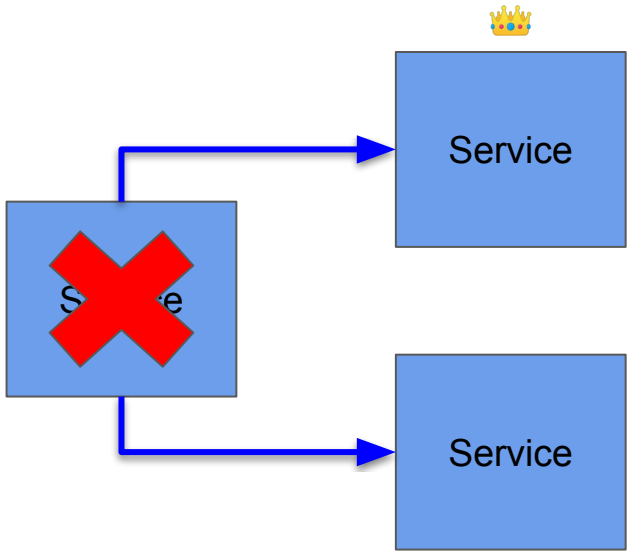
Benefits of Determinism

High Availability

Without Raft Consensus



With Raft Consensus



Benefits of Determinism

Open Source Fast Consensus with Aeron Cluster

Benefits of Determinism

Single Threaded Performance

Benefits of Determinism

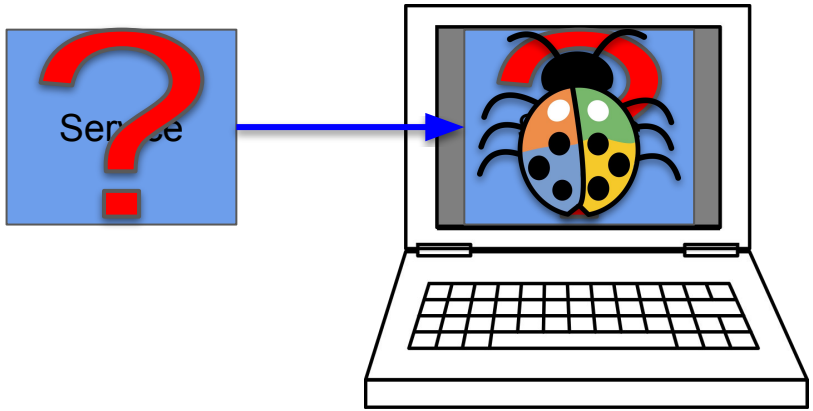
Straightforward Testing

Benefits of Determinism

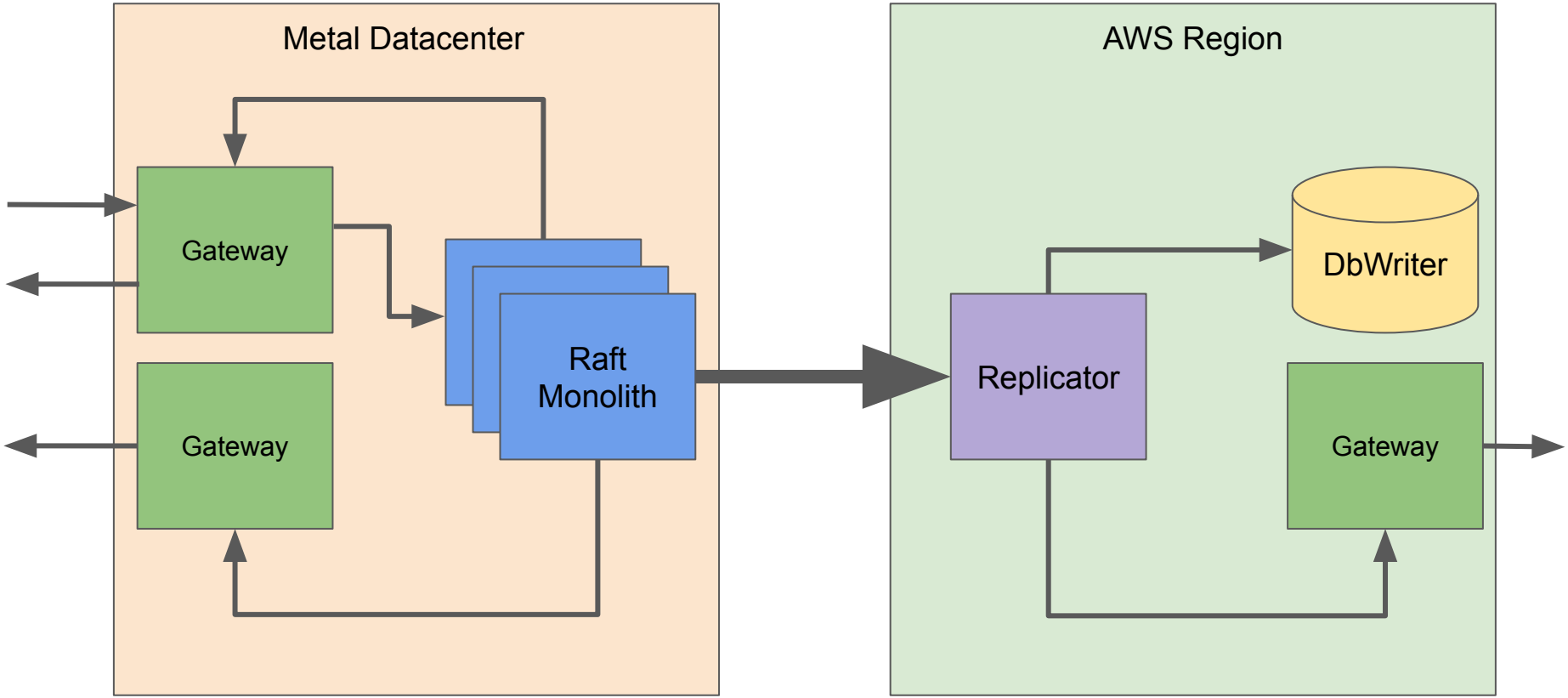
Tools for Troubleshooting

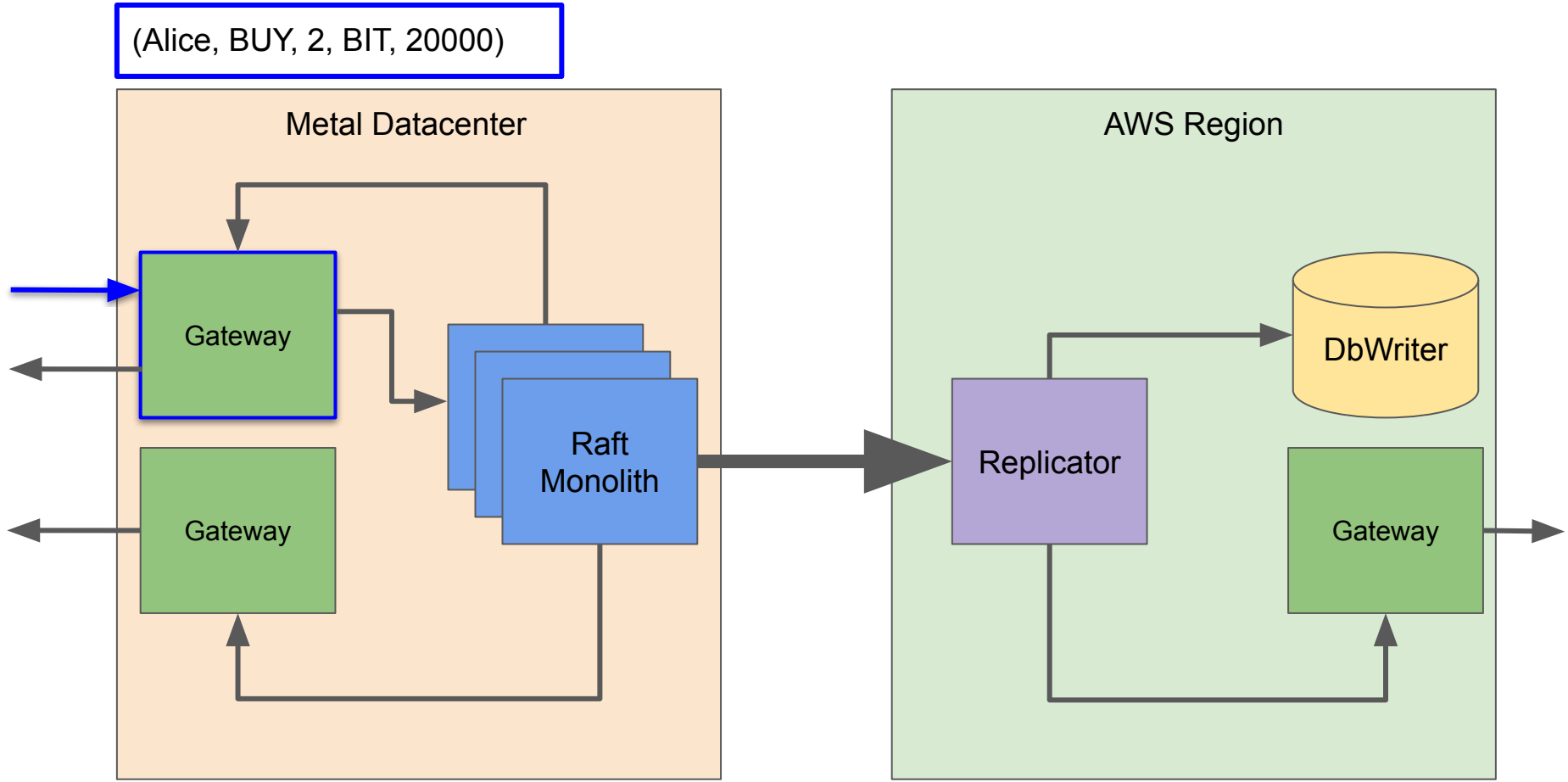
Benefits of Determinism

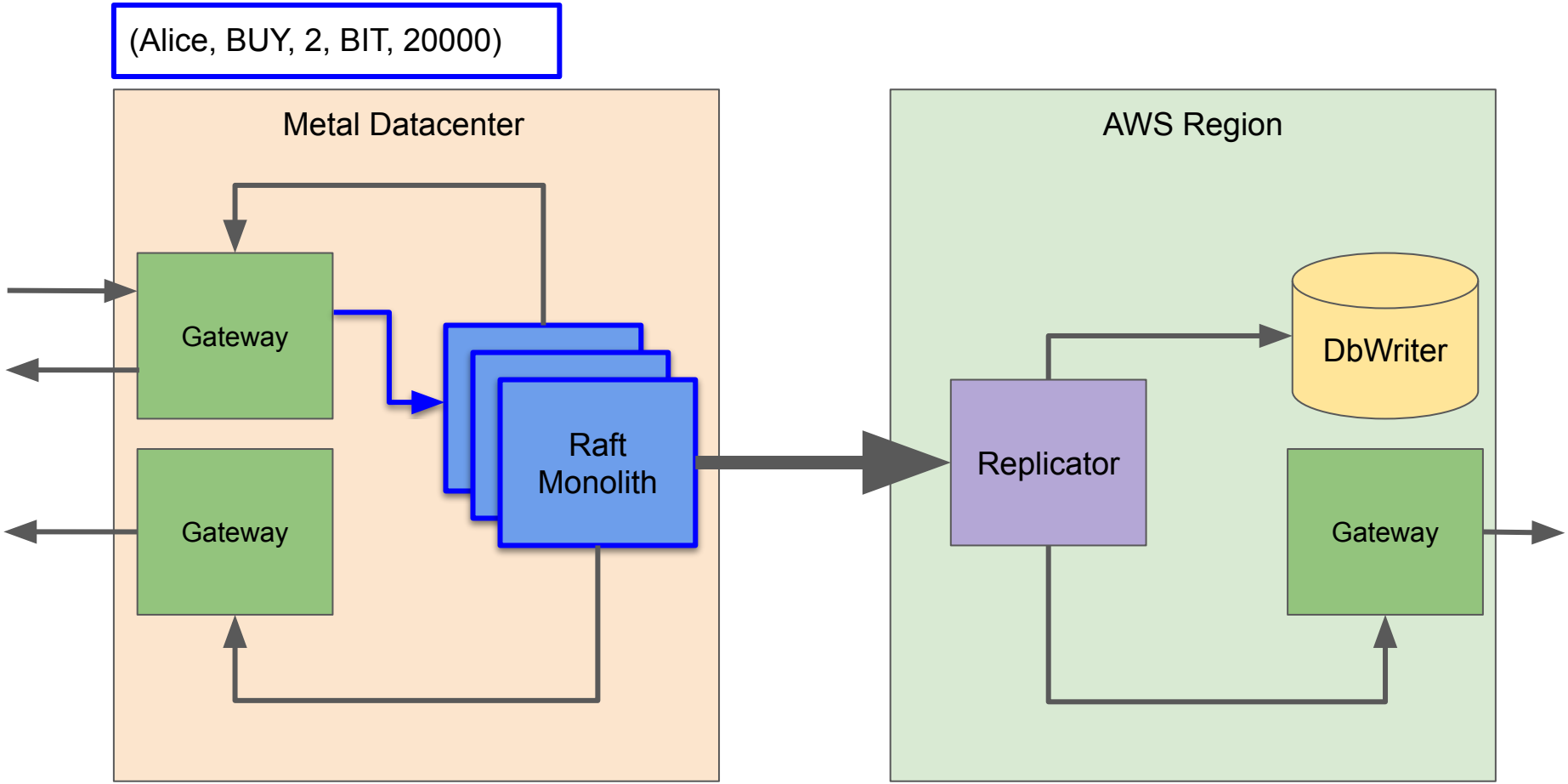
Benefits of Determinism

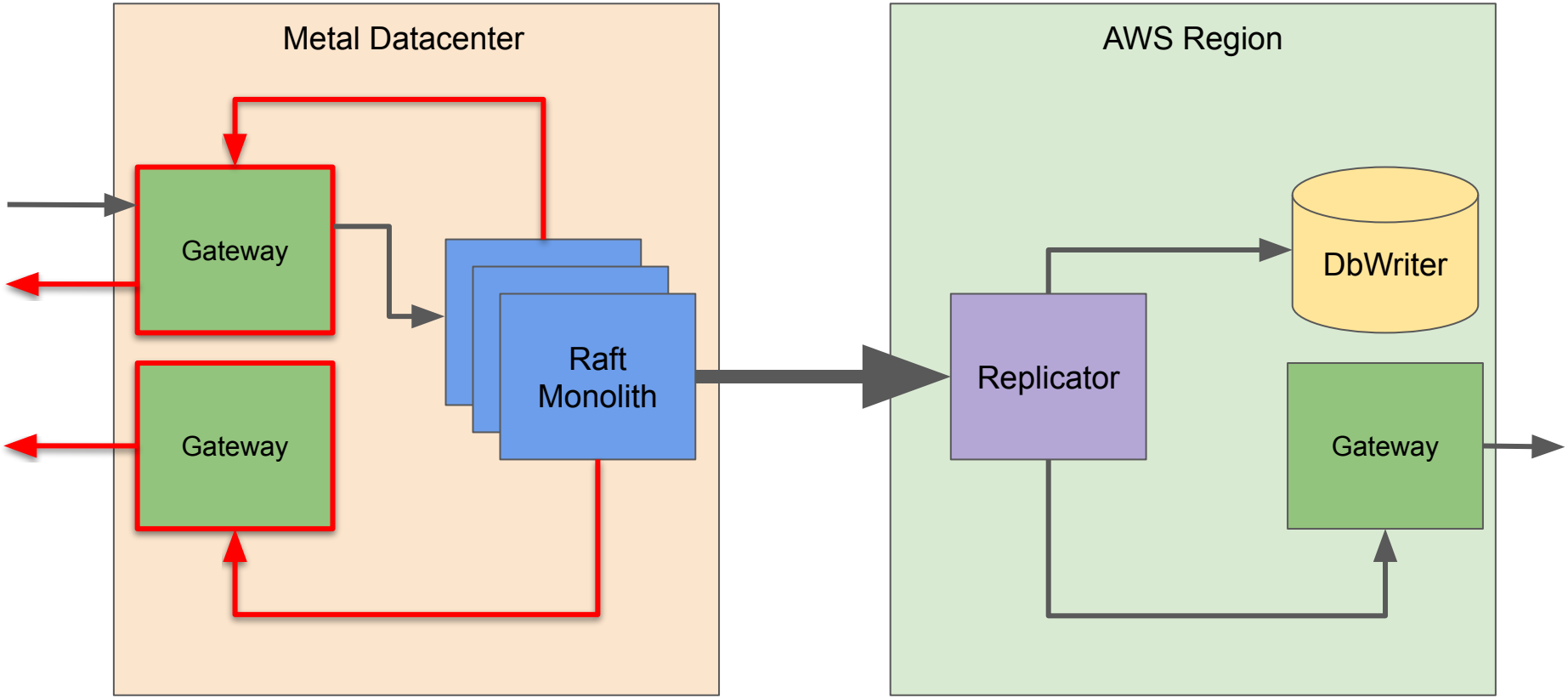


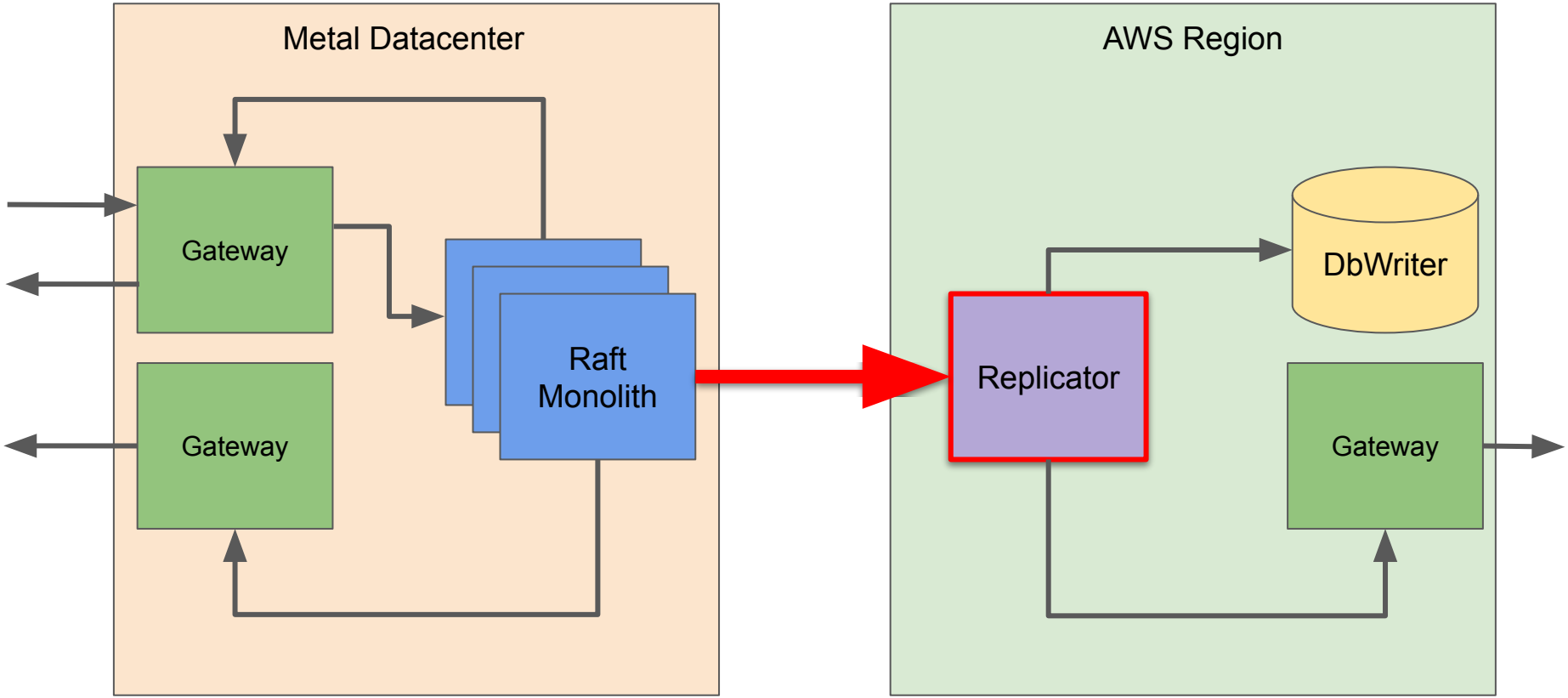
Sign us up!

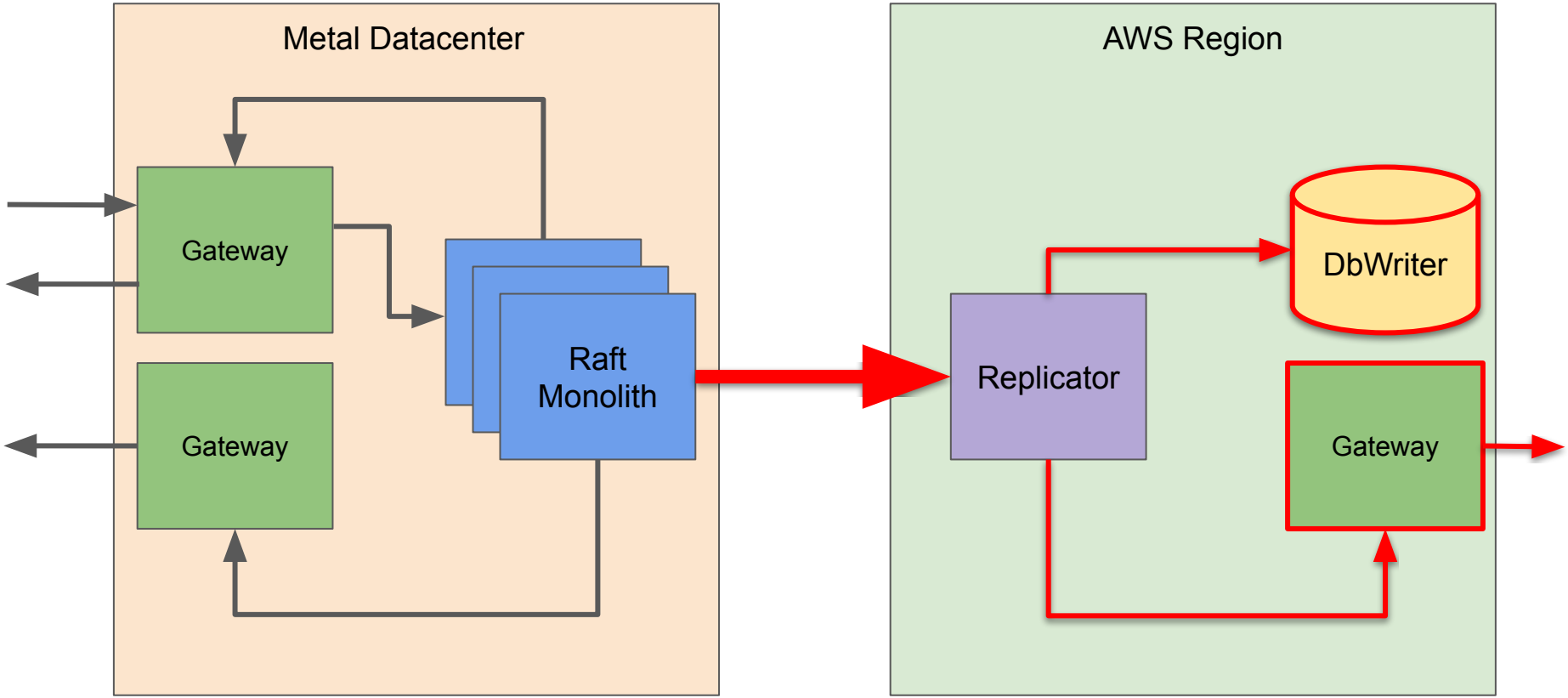

Metal Datacenter
AWS Region
Gateway
Gateway
Raft
Monolith
Replicator
DbWriter
Gateway

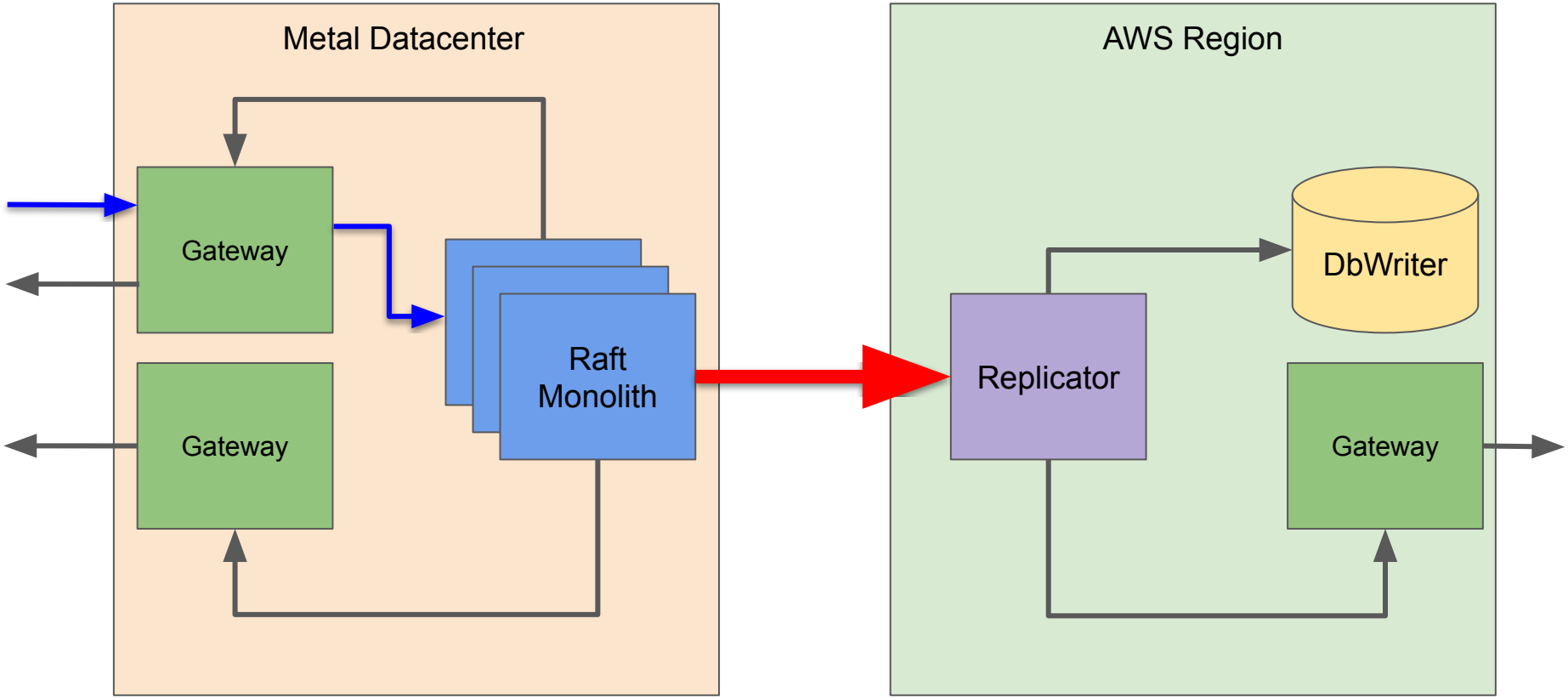

(Alice, BUY, 2, BIT, 20000)
Metal Datacenter
AWS Region
Gateway
Gateway
Raft
Monolith
Replicator
DbWriter
Gateway

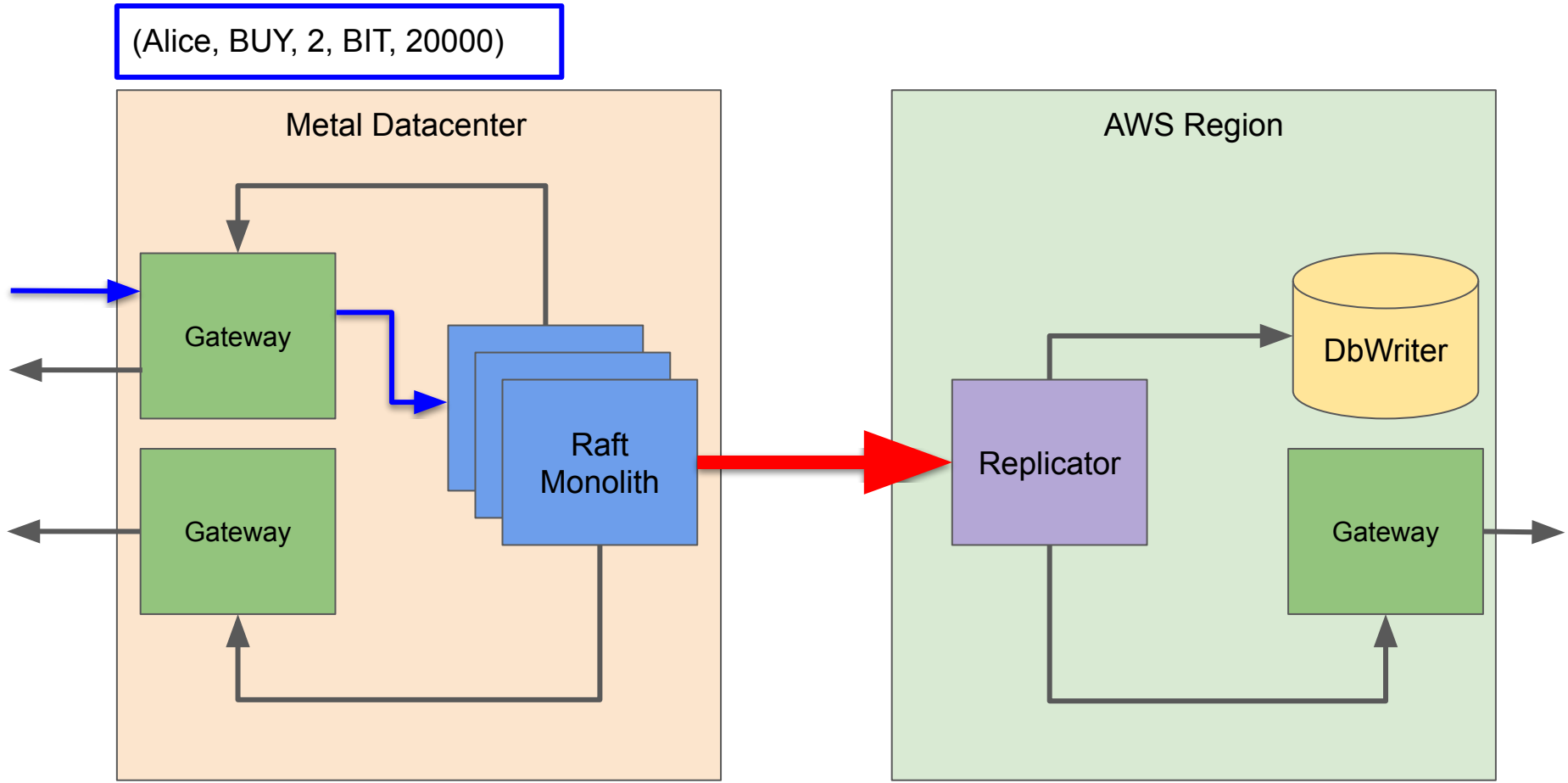

(Alice, BUY, 2, BIT, 20000)
Metal Datacenter
AWS Region
Gateway
Gateway
Raft
Monolith
Replicator
DbWriter
Gateway

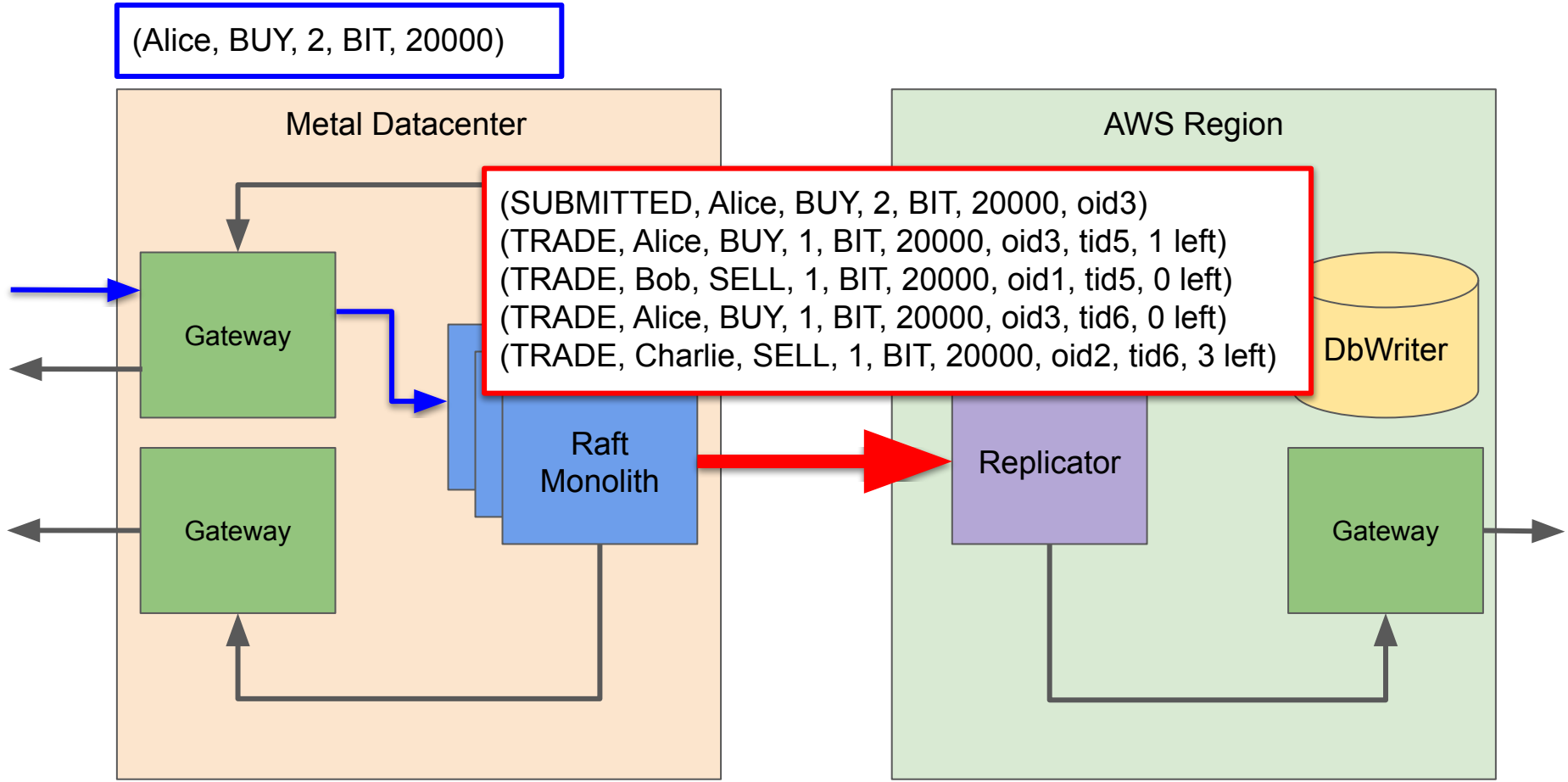

Metal Datacenter
Gateway
Gateway
Raft
Monolith
AWS Region
Replicator
DbWriter
Gateway

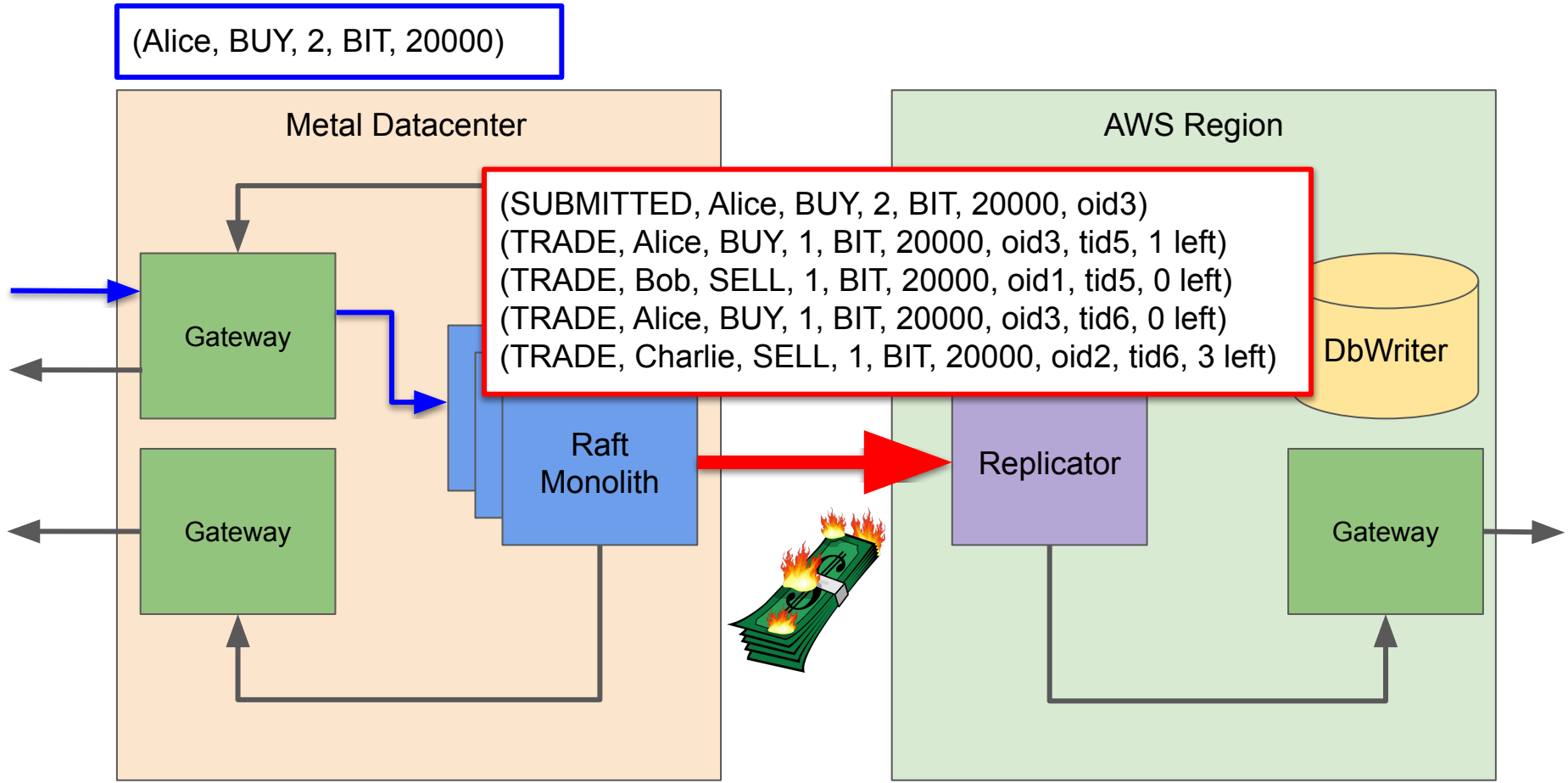

Metal Datacenter
AWS Region
Gateway
Gateway
Raft
Monolith
Replicator
DbWriter
Gateway

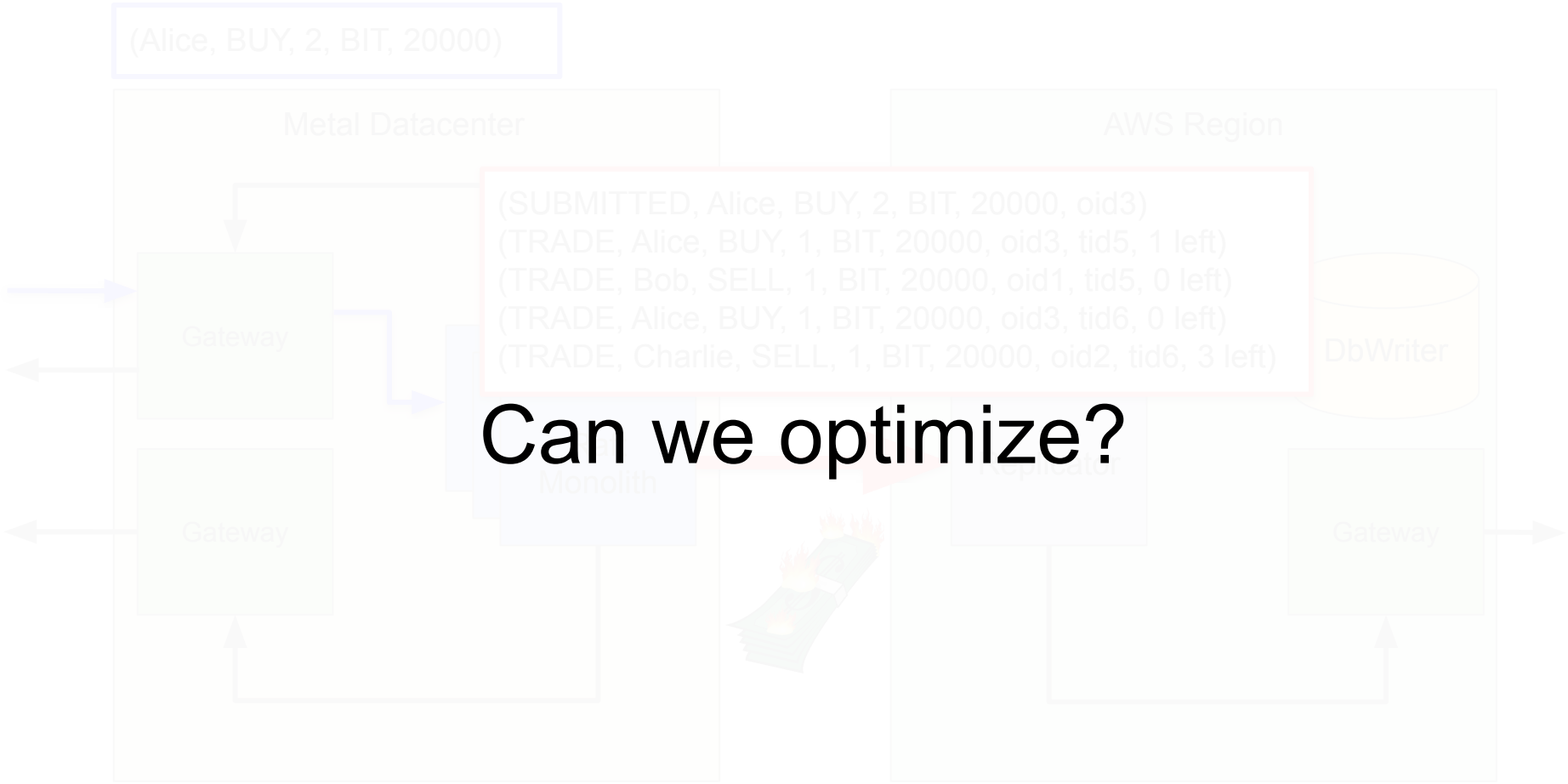

Metal Datacenter
AWS Region
Gateway
Gateway
Raft
Monolith
Replicator
DbWriter
Gateway


Metal Datacenter
Gateway
Gateway
Raft
Monolith
AWS Region
Replicator
DbWriter
Gateway


(Alice, BUY, 2, BIT, 20000)
Metal Datacenter
AWS Region
Gateway
Gateway
Raft
Monolith
Replicator
DbWriter
Gateway


(Alice, BUY, 2, BIT, 20000)
Metal Datacenter
AWS Region
(SUBMITTED, Alice, BUY, 2, BIT, 20000, oid3)
(TRADE, Alice, BUY, 1, BIT, 20000, oid3, tid5, 1 left)
(TRADE, Bob, SELL, 1, BIT, 20000, oid1, tid5, 0 left)
(TRADE, Alice, BUY, 1, BIT, 20000, oid3, tid6, 0 left)
(TRADE, Charlie, SELL, 1, BIT, 20000, oid2, tid6, 3 left)
Gateway
Gateway
Raft
Monolith
DbWriter
Replicator
Gateway

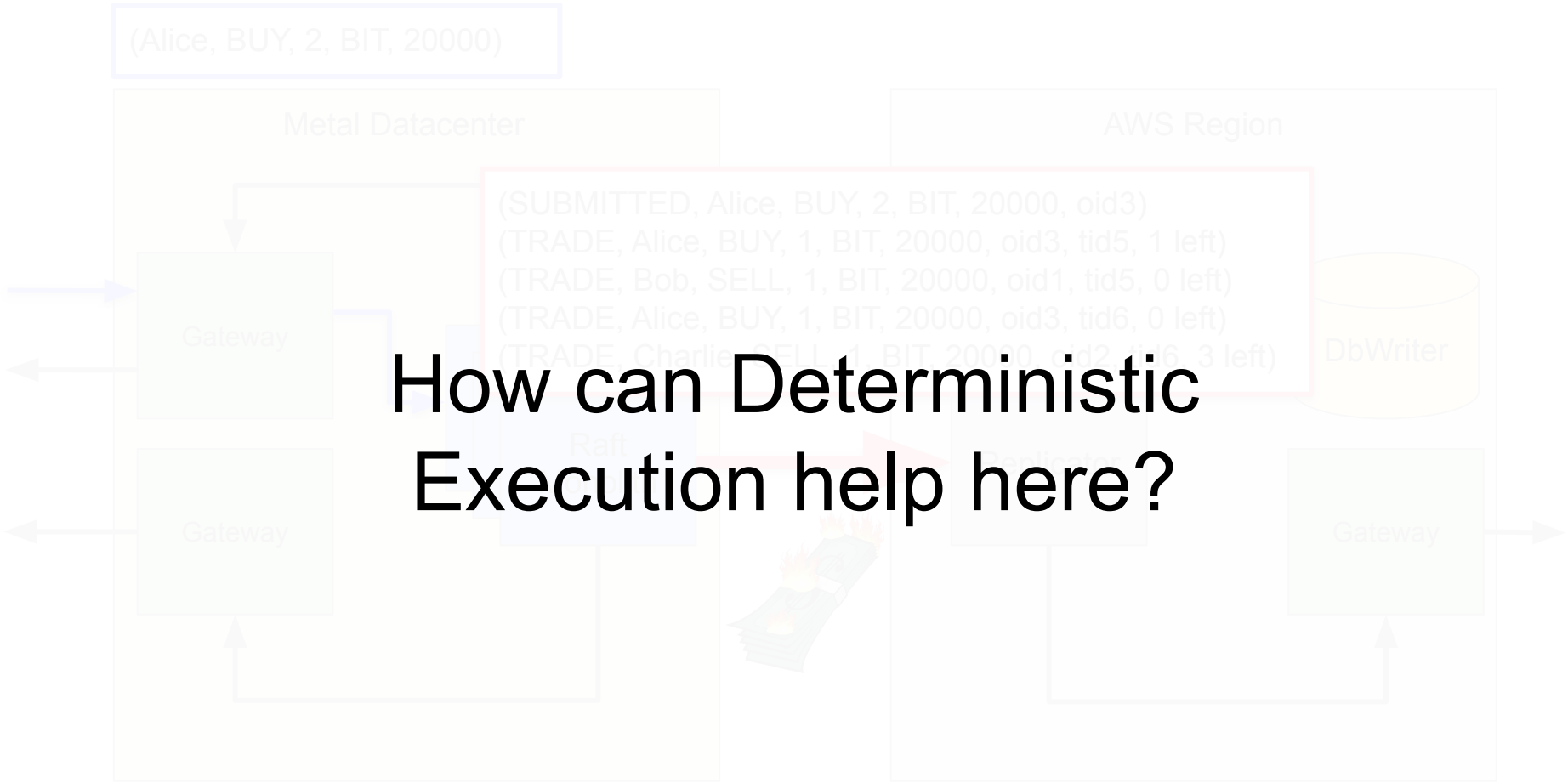

(Alice, BUY, 2, BIT, 20000)
Metal Datacenter
AWS Region
(SUBMITTED, Alice, BUY, 2, BIT, 20000, oid3)
(TRADE, Alice, BUY, 1, BIT, 20000, oid3, tid5, 1 left)
(TRADE, Bob, SELL, 1, BIT, 20000, oid1, tid5, 0 left)
(TRADE, Alice, BUY, 1, BIT, 20000, oid3, tid6, 0 left)
(TRADE, Charlie, SELL, 1, BIT, 20000, oid2, tid6, 3 left)
Gateway
Gateway
Raft
Monolith
DbWriter
Replicator
Gateway



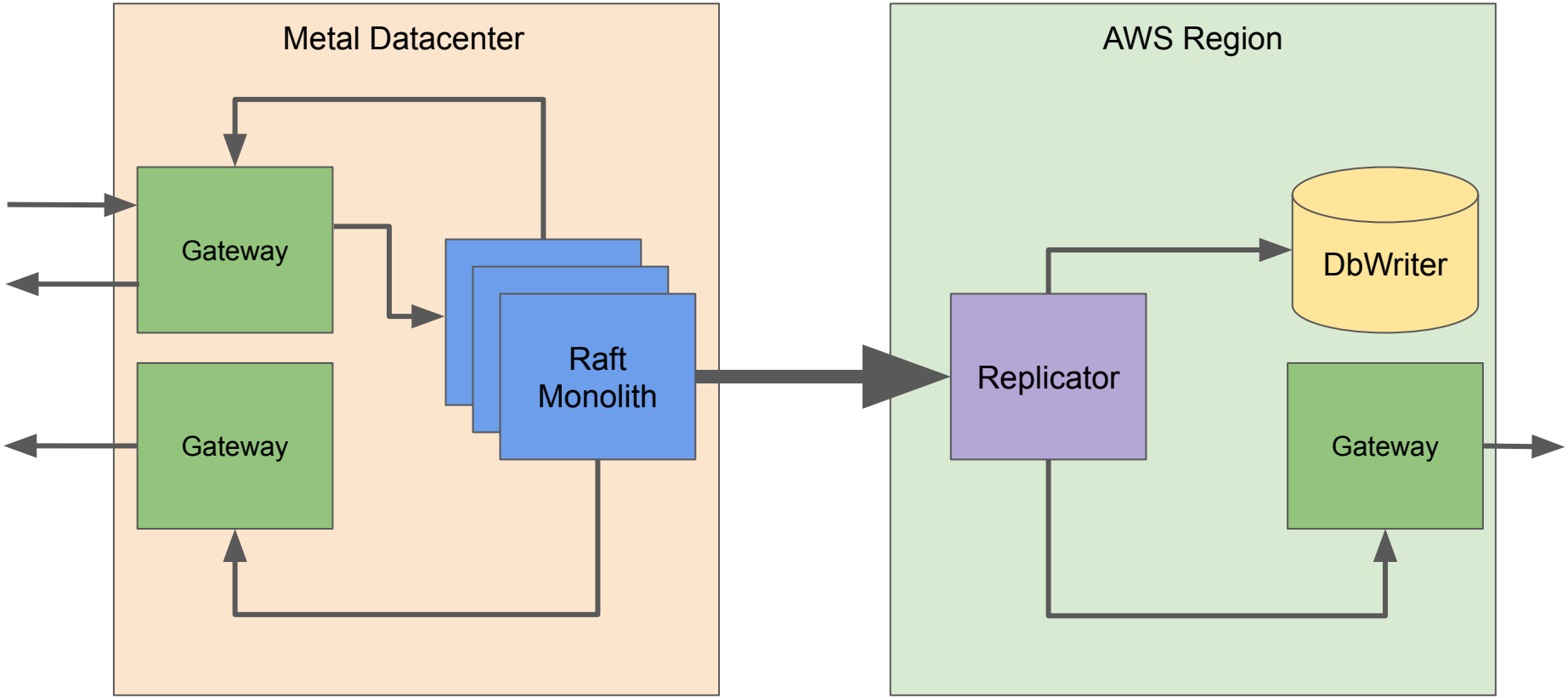
Can we optimize?

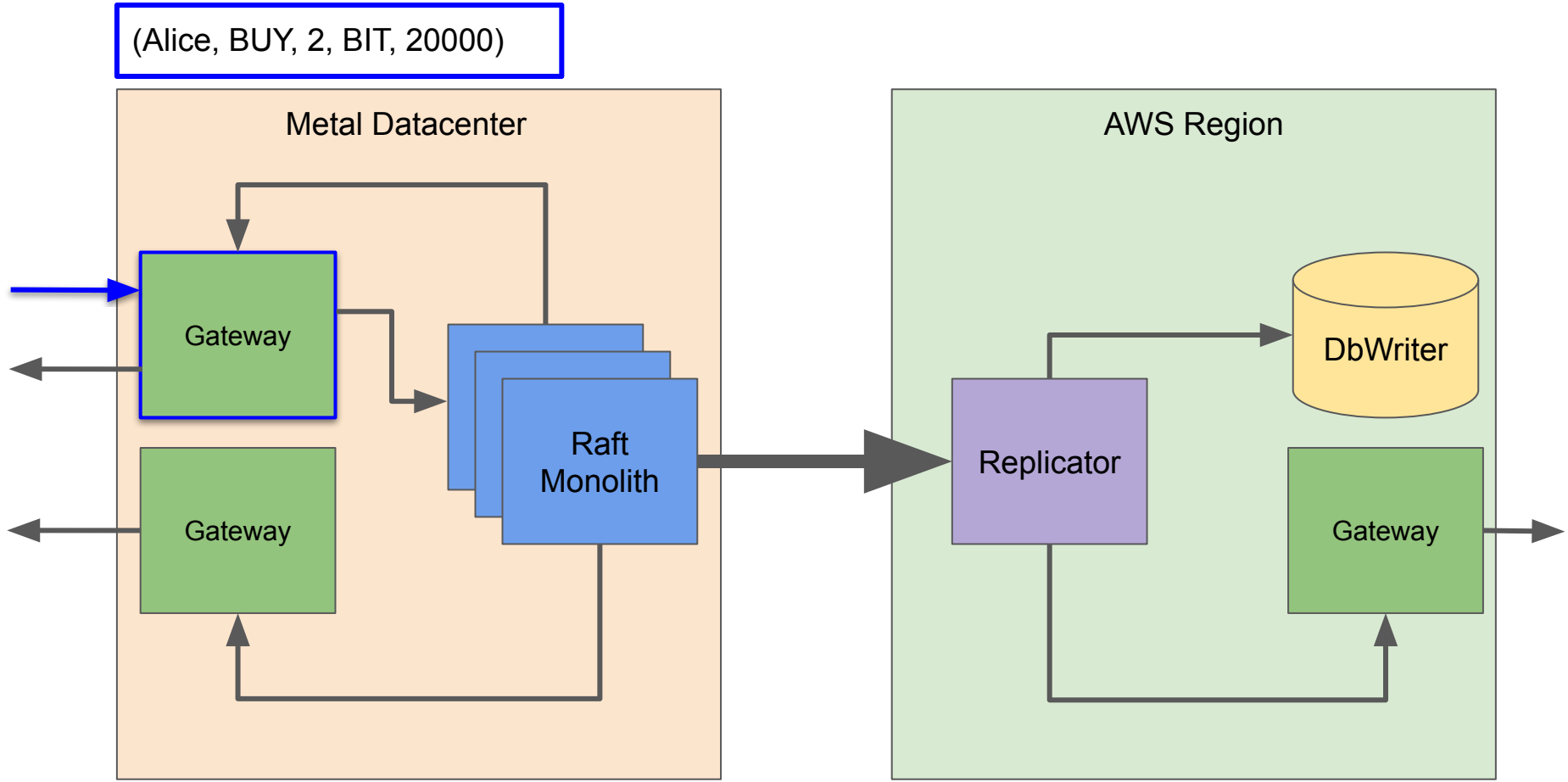
Business **logic** is often **fast/cheap**

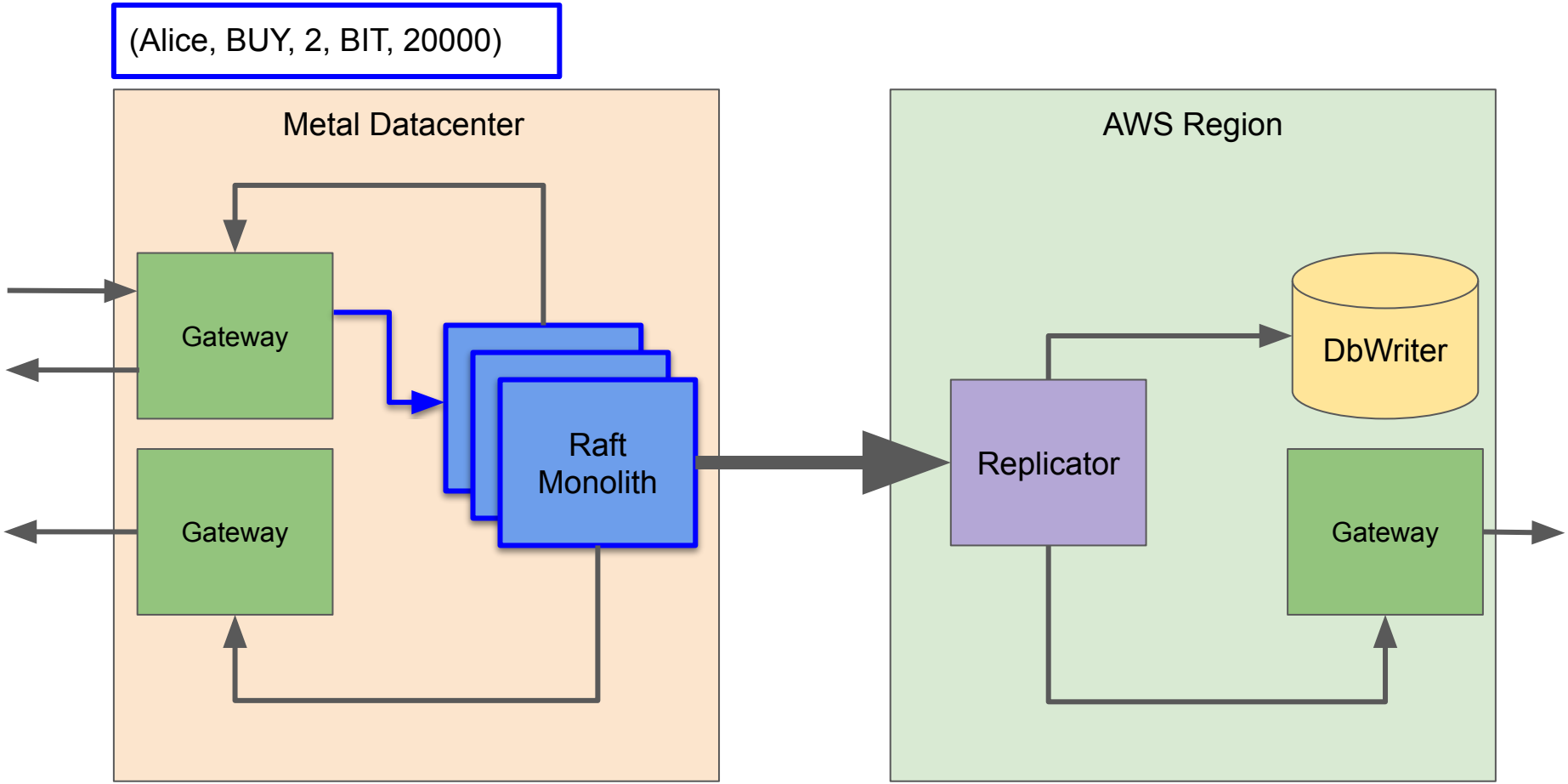
Writing and reading **data** can be **slow/expensive**

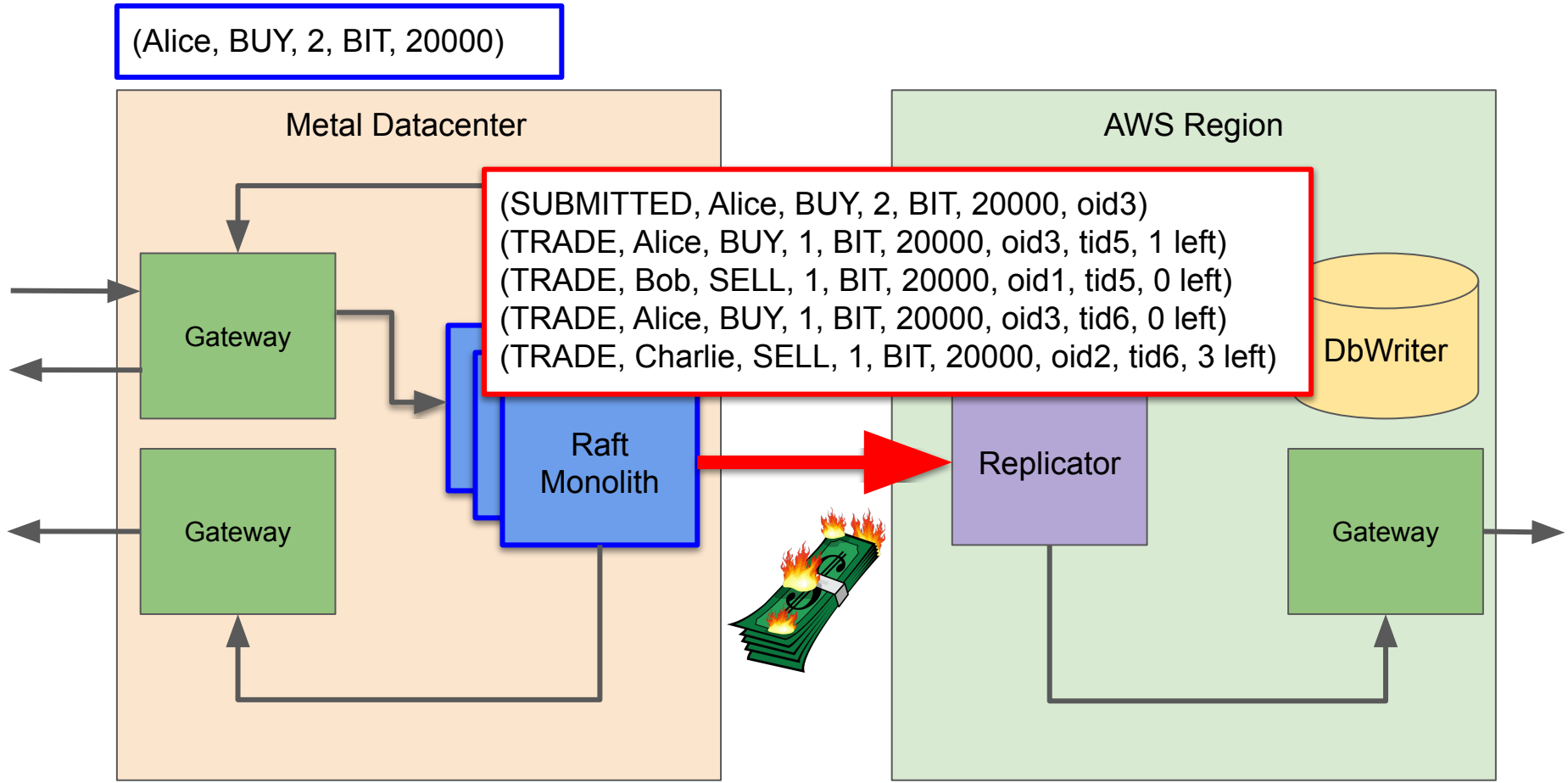


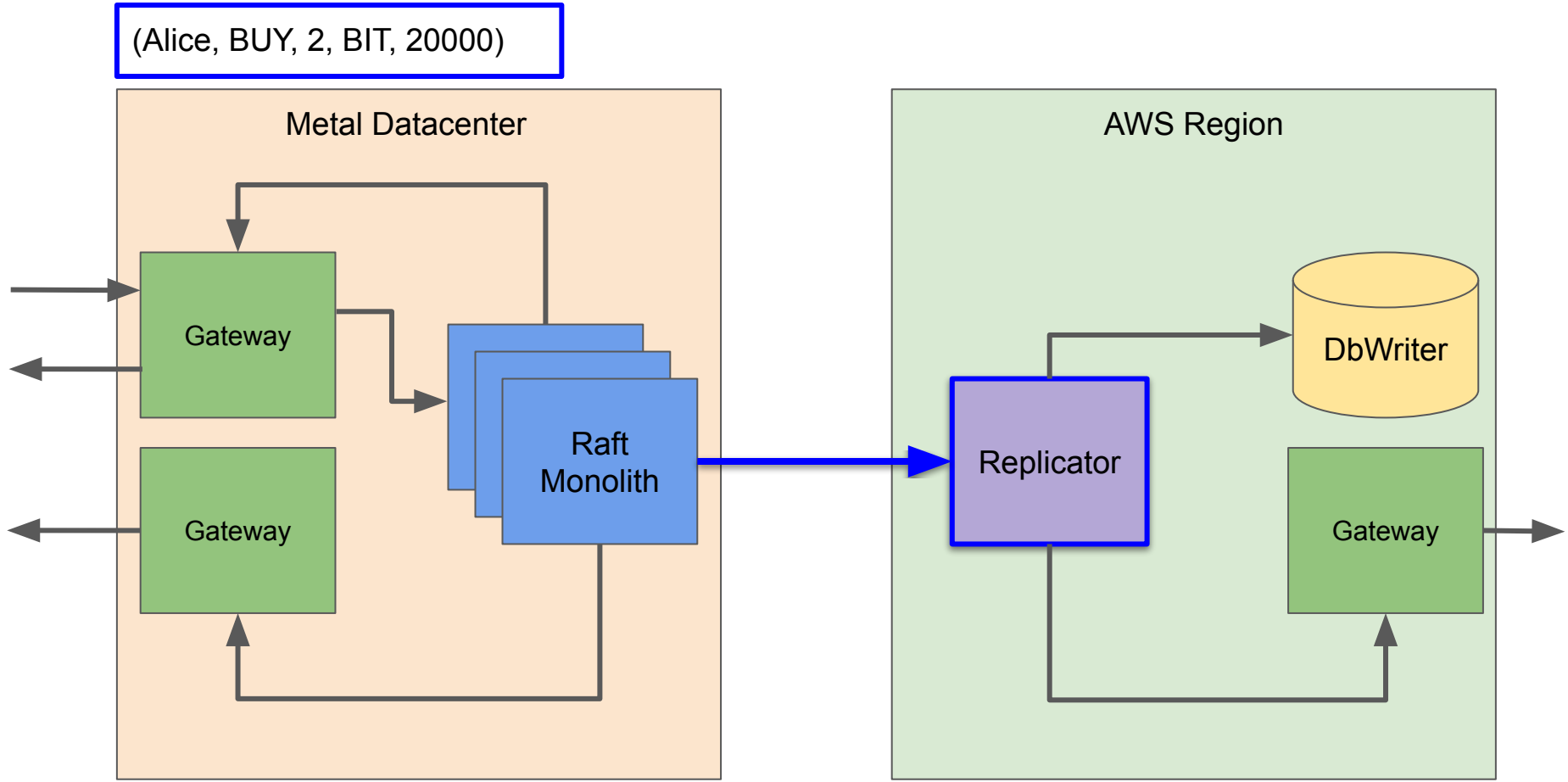
How can Deterministic Execution help here?

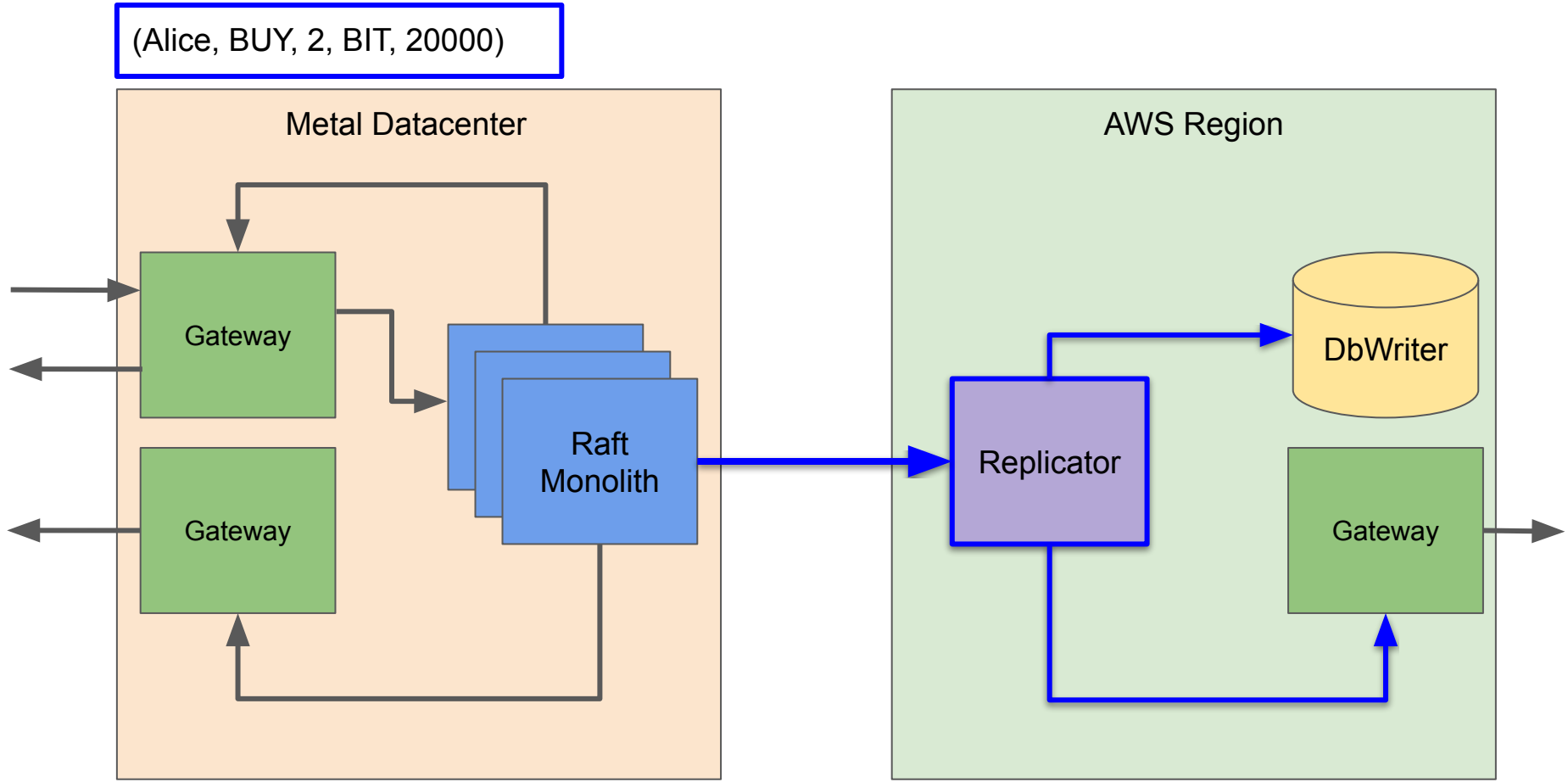

Metal Datacenter
AWS Region
Gateway
Gateway
Raft
Monolith
Replicator
DbWriter
Gateway

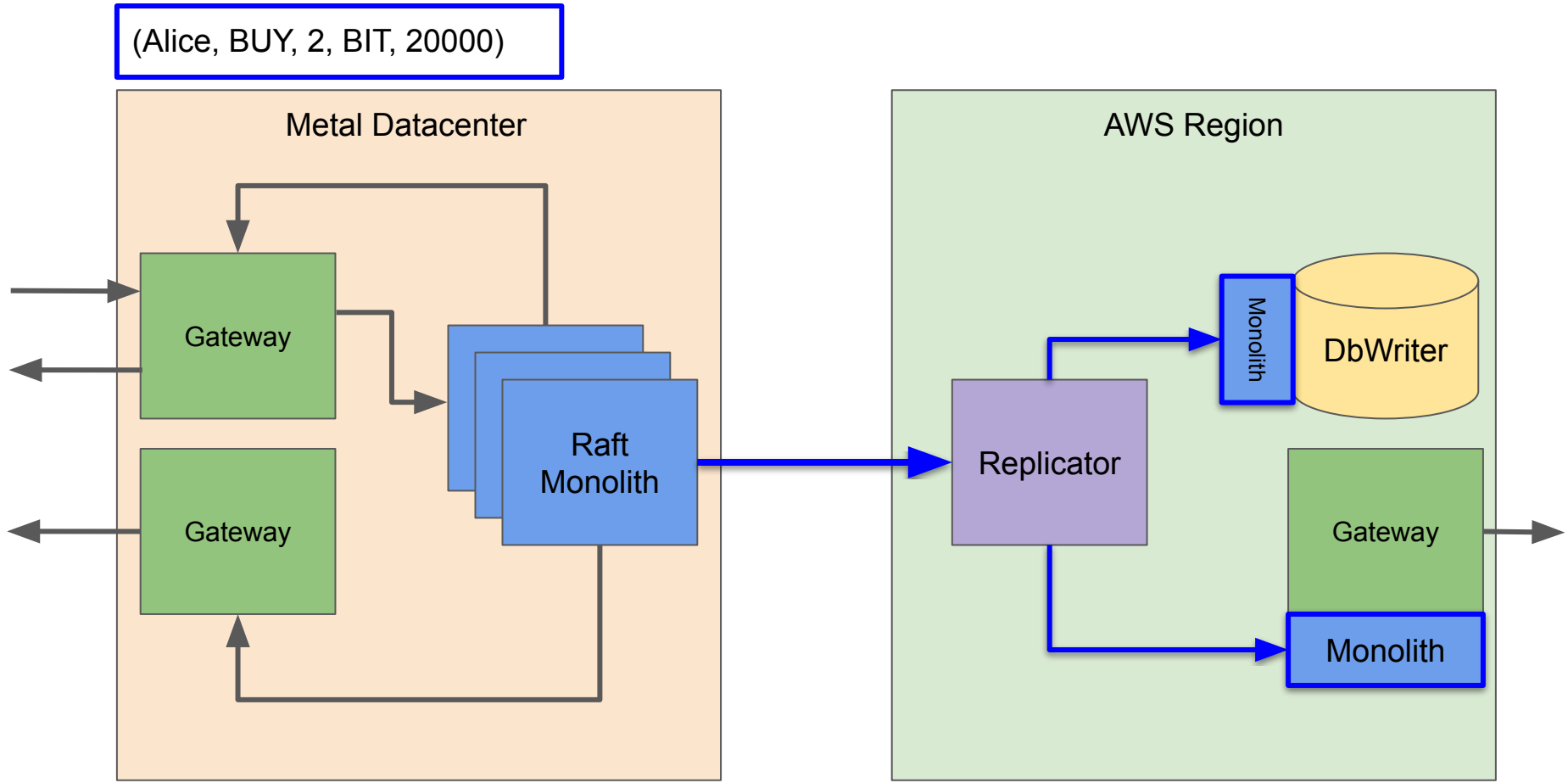

(Alice, BUY, 2, BIT, 20000)
Metal Datacenter
AWS Region
Gateway
Gateway
Raft
Monolith
Replicator
DbWriter
Gateway

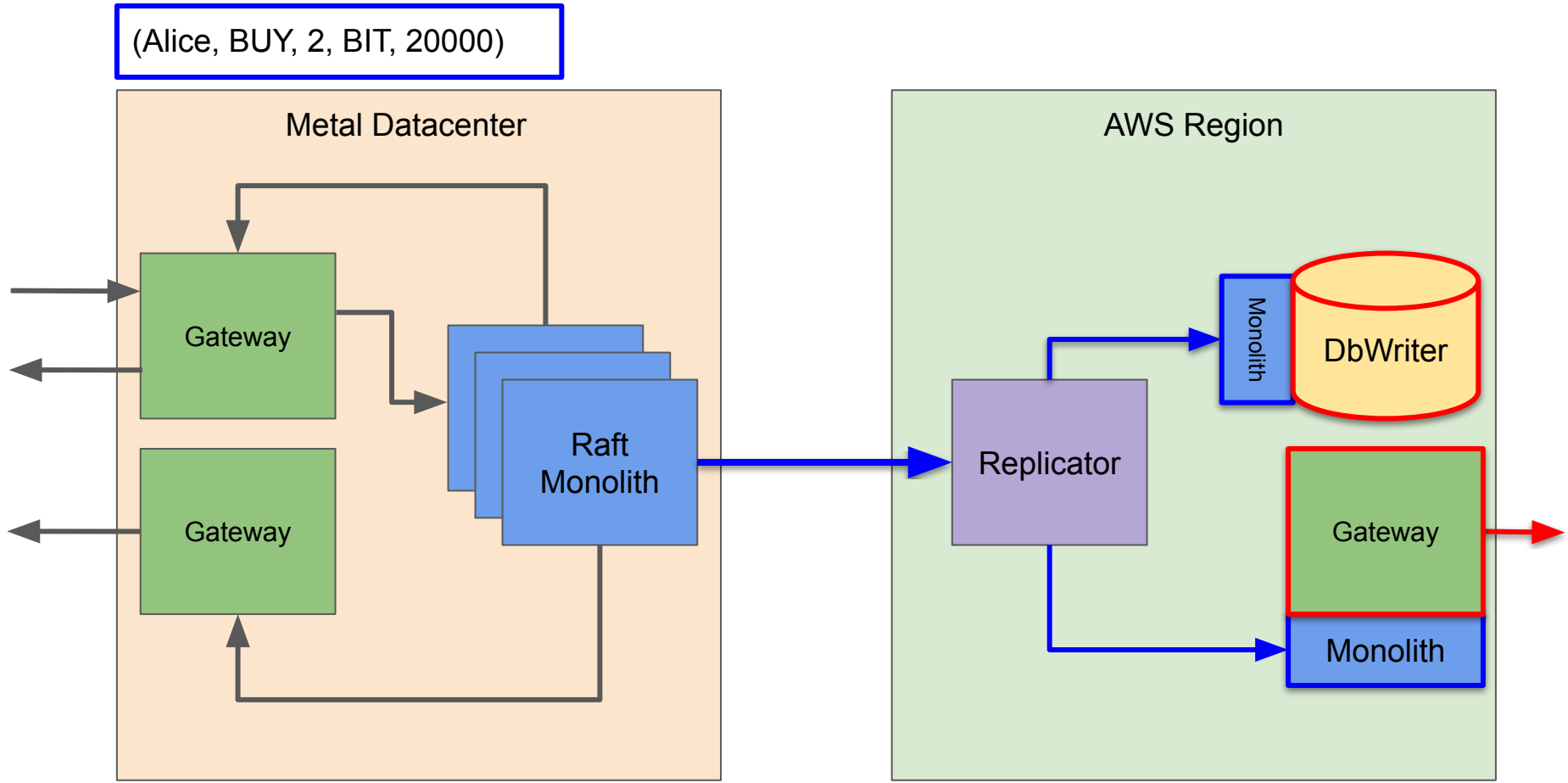

(Alice, BUY, 2, BIT, 20000)
Metal Datacenter
AWS Region
Gateway
Gateway
Raft
Monolith
Replicator
DbWriter
Gateway

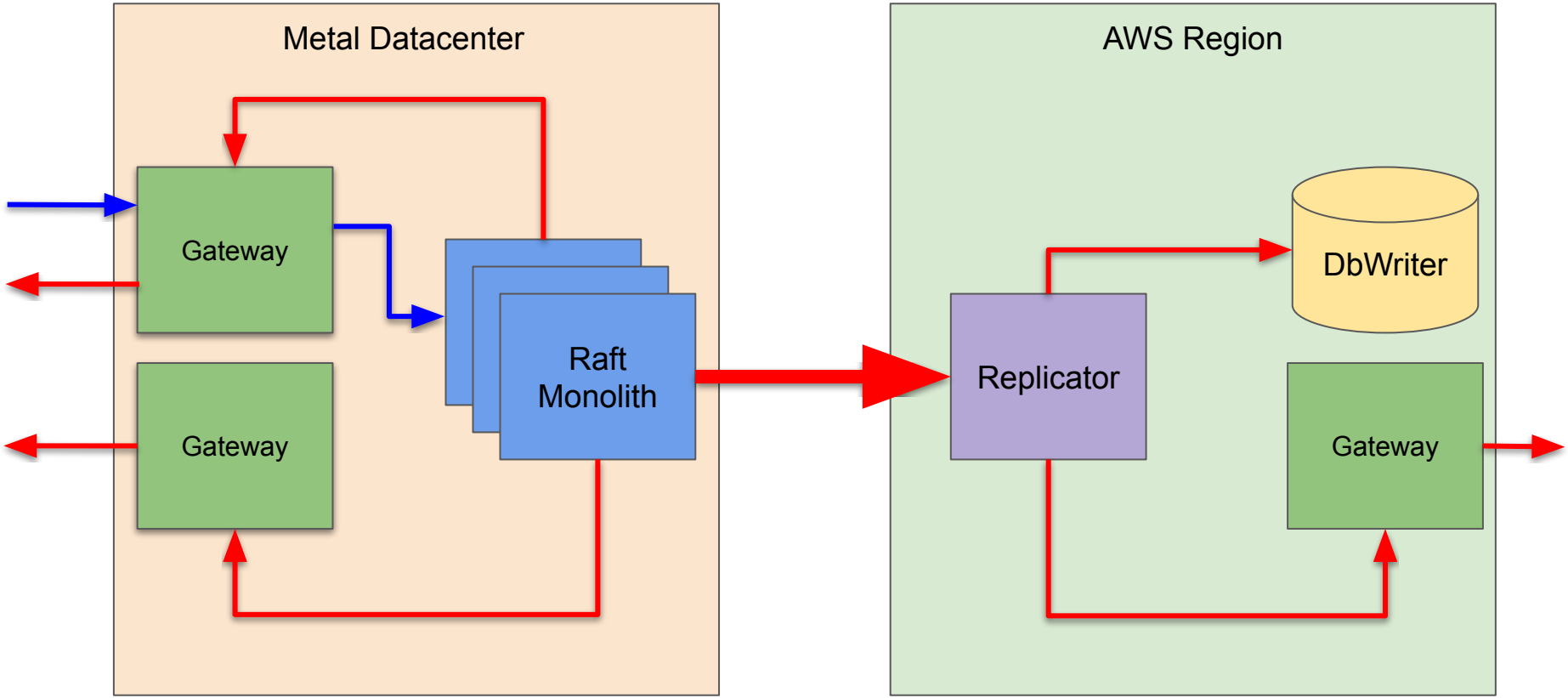

(Alice, BUY, 2, BIT, 20000)
Metal Datacenter
AWS Region
(SUBMITTED, Alice, BUY, 2, BIT, 20000, oid3)
(TRADE, Alice, BUY, 1, BIT, 20000, oid3, tid5, 1 left)
(TRADE, Bob, SELL, 1, BIT, 20000, oid1, tid5, 0 left)
(TRADE, Alice, BUY, 1, BIT, 20000, oid3, tid6, 0 left)
(TRADE, Charlie, SELL, 1, BIT, 20000, oid2, tid6, 3 left)
Gateway
Gateway
Raft
Monolith
DbWriter
Replicator
Gateway

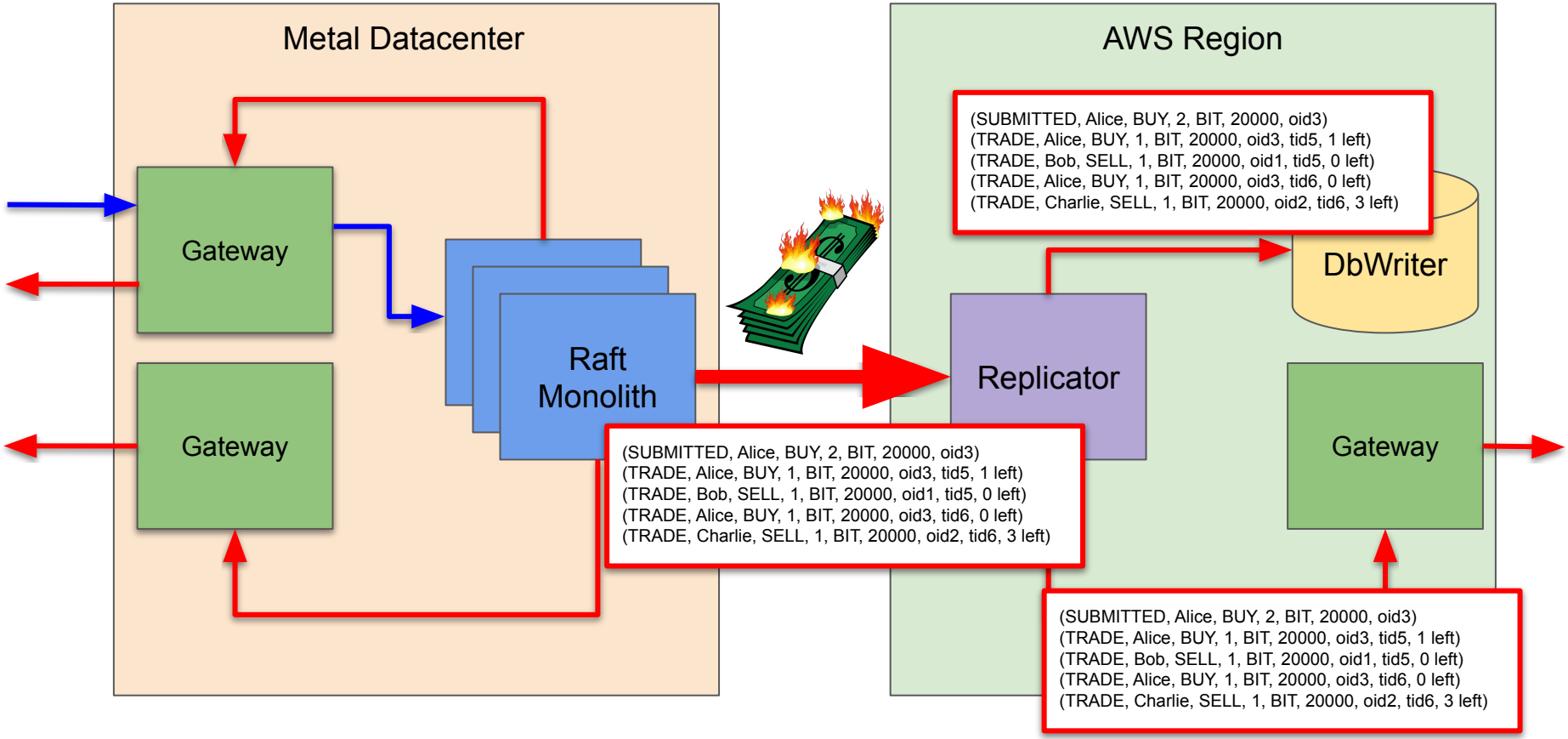

(Alice, BUY, 2, BIT, 20000)
Metal Datacenter
AWS Region
Gateway
Gateway
Raft
Monolith
Replicator
DbWriter
Gateway

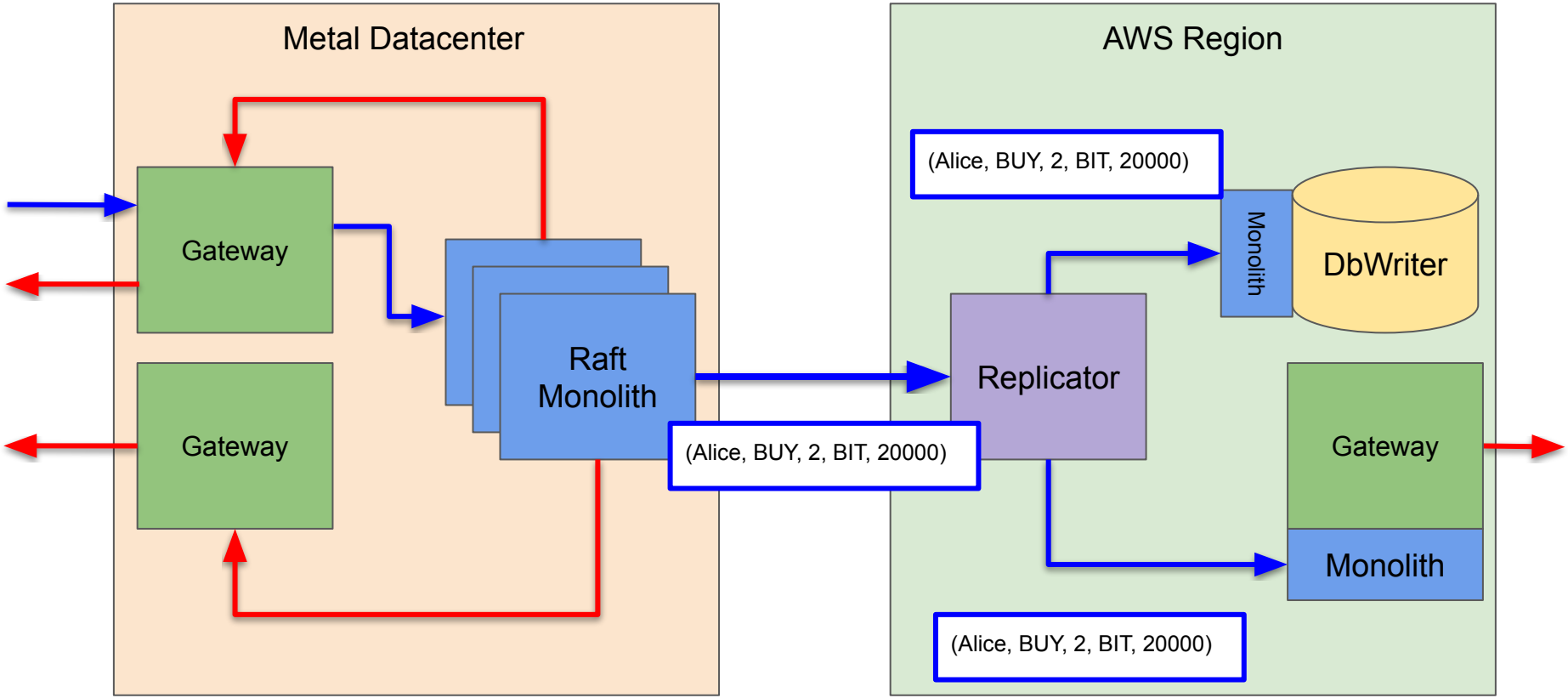

(Alice, BUY, 2, BIT, 20000)
Metal Datacenter
AWS Region
Gateway
Gateway
Raft
Monolith
Replicator
DbWriter
Gateway

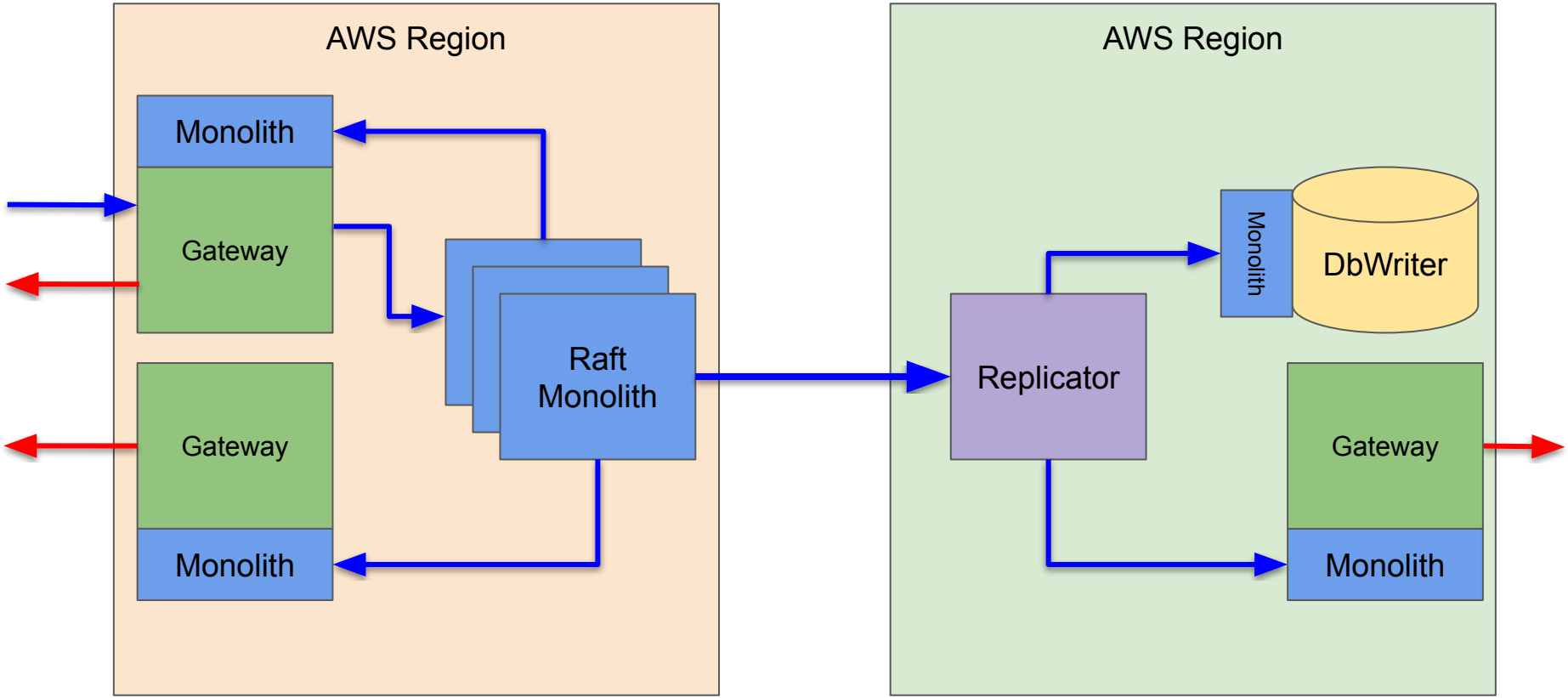

(Alice, BUY, 2, BIT, 20000)
Metal Datacenter
Gateway
Gateway
Raft
Monolith
AWS Region
Monolith
DbWriter
Replicator
Gateway
Monolith


(Alice, BUY, 2, BIT, 20000)
Metal Datacenter
AWS Region
Gateway
Gateway
Raft
Monolith
Replicator
Monolith
DbWriter
Gateway
Monolith


Metal Datacenter
AWS Region
Gateway
Gateway
Raft
Monolith
Replicator
DbWriter
Gateway


Metal Datacenter
AWS Region
Gateway
Gateway
Raft
Monolith
(SUBMITTED, Alice, BUY, 2, BIT, 20000, oid3)
(TRADE, Alice, BUY, 1, BIT, 20000, oid3, tid5, 1 left)
(TRADE, Bob, SELL, 1, BIT, 20000, oid1, tid5, 0 left)
(TRADE, Alice, BUY, 1, BIT, 20000, oid3, tid6, 0 left)
(TRADE, Charlie, SELL, 1, BIT, 20000, oid2, tid6, 3 left)
(SUBMITTED, Alice, BUY, 2, BIT, 20000, oid3)
(TRADE, Alice, BUY, 1, BIT, 20000, oid3, tid5, 1 left)
(TRADE, Bob, SELL, 1, BIT, 20000, oid1, tid5, 0 left)
(TRADE, Alice, BUY, 1, BIT, 20000, oid3, tid6, 0 left)
(TRADE, Charlie, SELL, 1, BIT, 20000, oid2, tid6, 3 left)
DbWriter
Replicator
Gateway
(SUBMITTED, Alice, BUY, 2, BIT, 20000, oid3)
(TRADE, Alice, BUY, 1, BIT, 20000, oid3, tid5, 1 left)
(TRADE, Bob, SELL, 1, BIT, 20000, oid1, tid5, 0 left)
(TRADE, Alice, BUY, 1, BIT, 20000, oid3, tid6, 0 left)
(TRADE, Charlie, SELL, 1, BIT, 20000, oid2, tid6, 3 left)


Metal Datacenter
Gateway
Gateway
Raft
Monolith
(Alice, BUY, 2, BIT, 20000)
AWS Region
(Alice, BUY, 2, BIT, 20000)
Monolith
DbWriter
Replicator
Gateway
Monolith
(Alice, BUY, 2, BIT, 20000)


AWS Region
Monolith
Gateway
Gateway
Monolith
Raft
Monolith
AWS Region
Monolith
DbWriter
Replicator
Gateway
Monolith

Replay logic to scale and stabilize

Inputs are often **smaller** than outputs

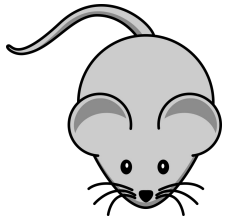
Replay logic to scale and stabilize

Inputs can be more
consistent than outputs

Replay logic to scale and stabilize

What is your 99th percentile network load?

Replay logic to scale and stabilize



(Alice, SELL, 1000, BIT, Market)

Replay logic to scale and stabilize

Input sizes and rates can be validated/rejected

Replay logic to scale and stabilize

Output size and rate are hard to validate/reject

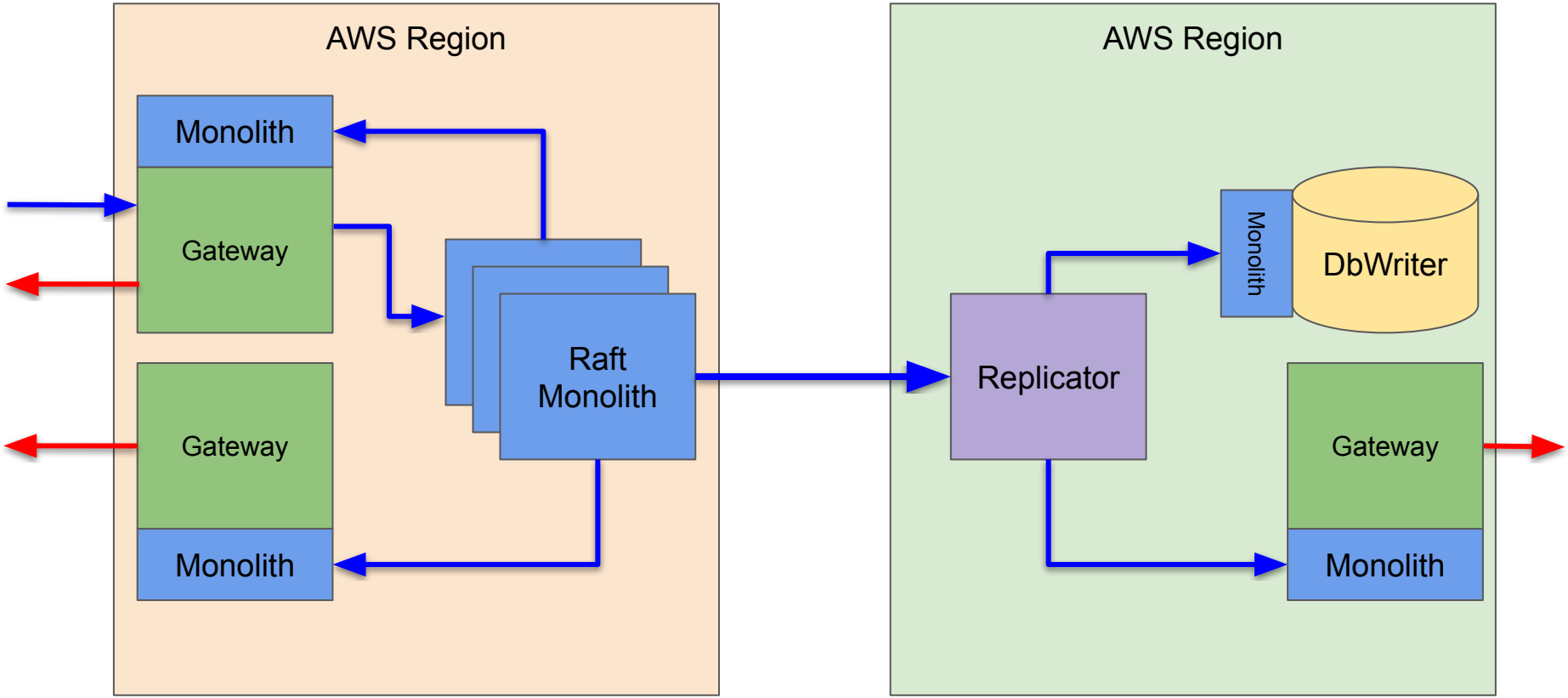
Replay logic to scale and stabilize

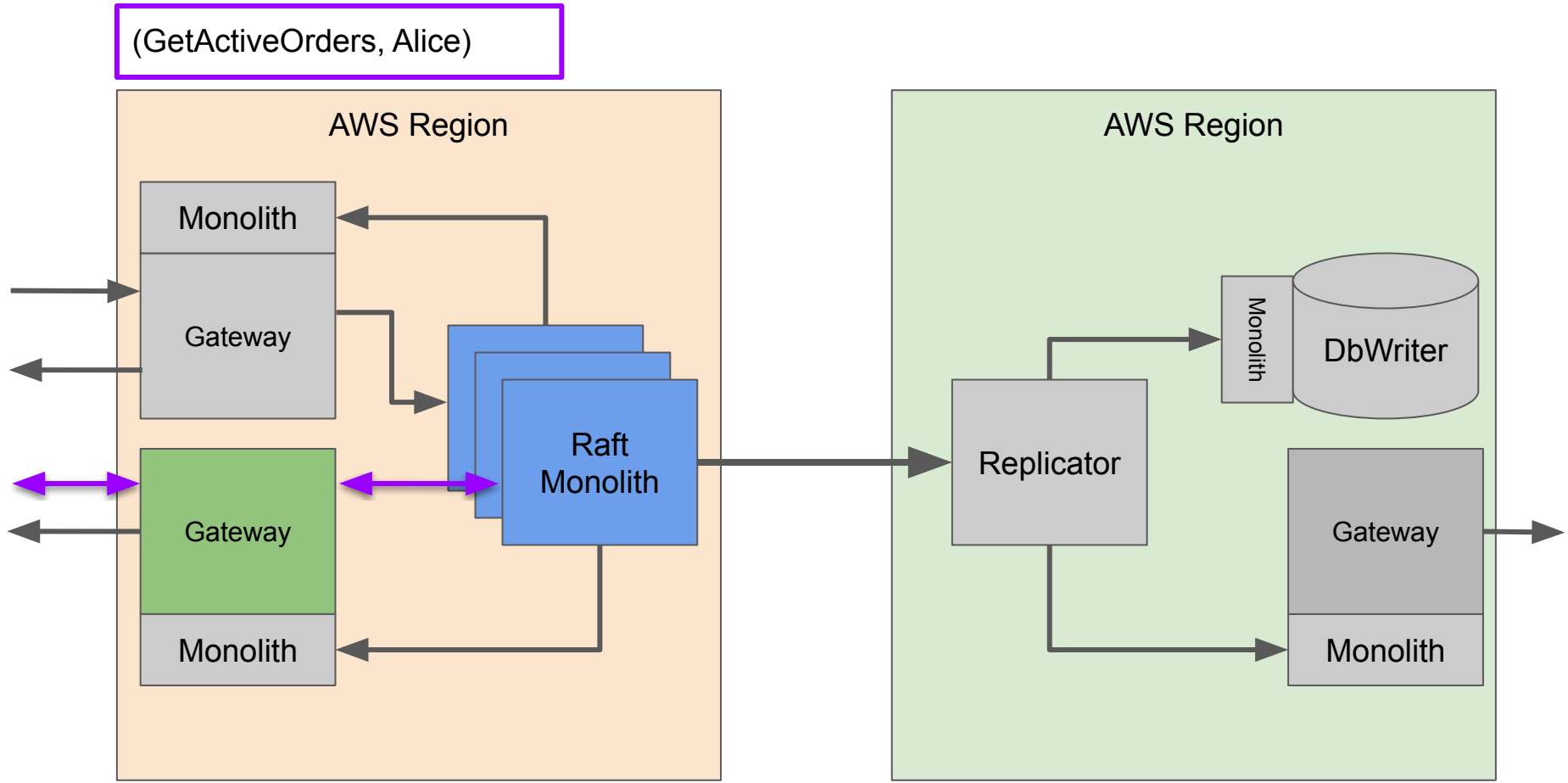
Protect from Thundering Herd by not sending blasts of events

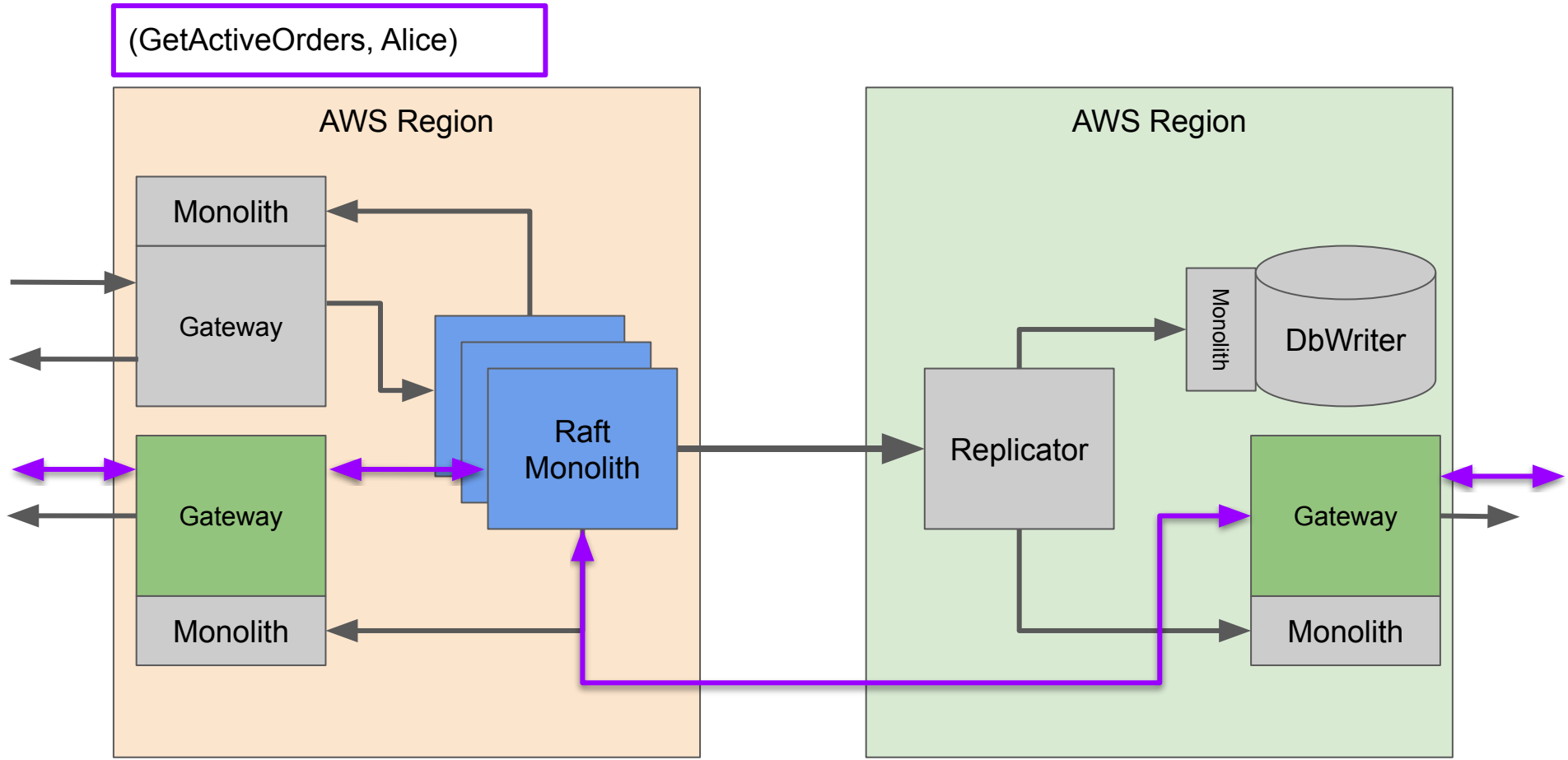
Replay your logs in production to scale and stabilize your system

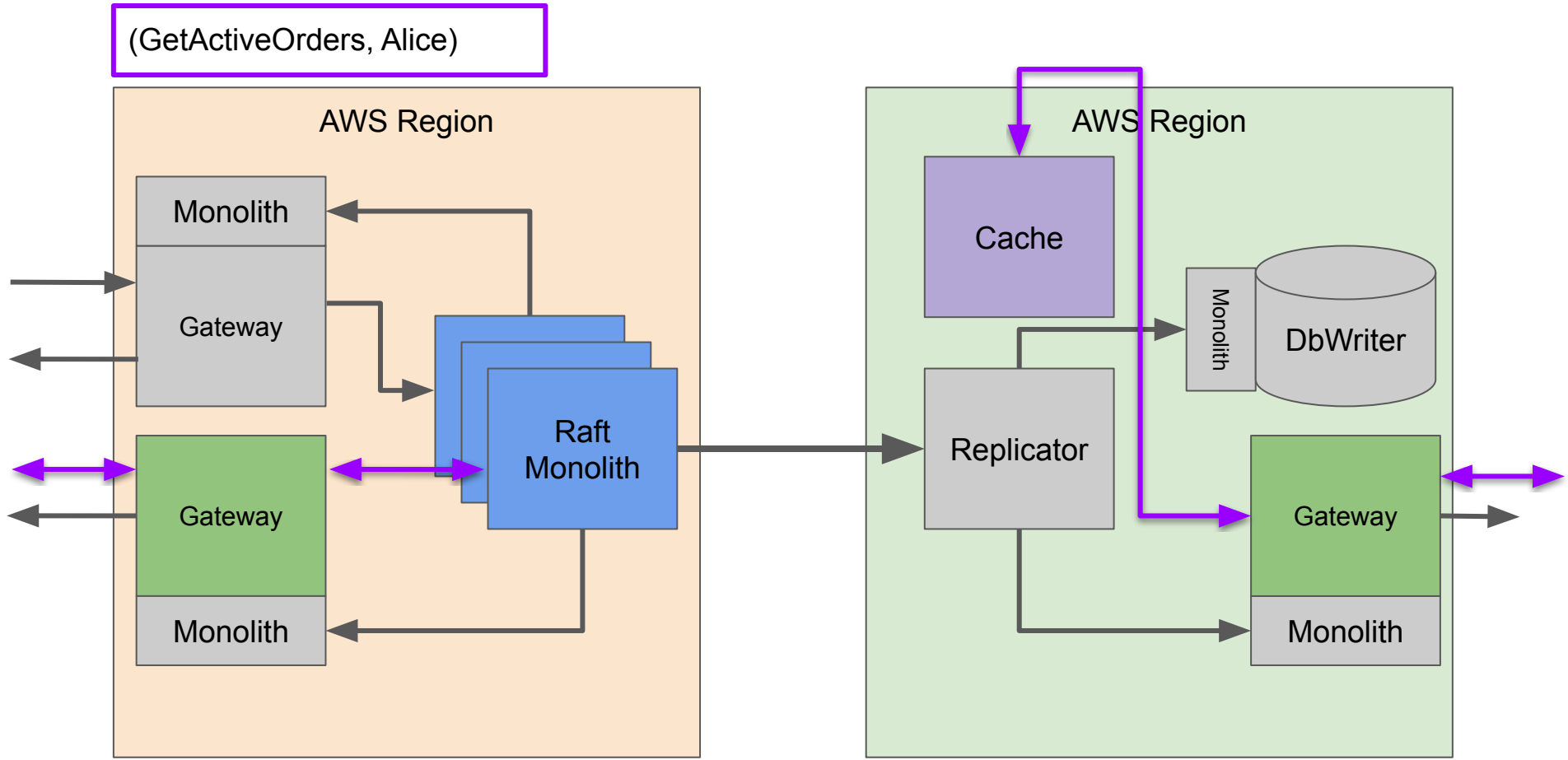
Replicating Compute can simplify downstream code

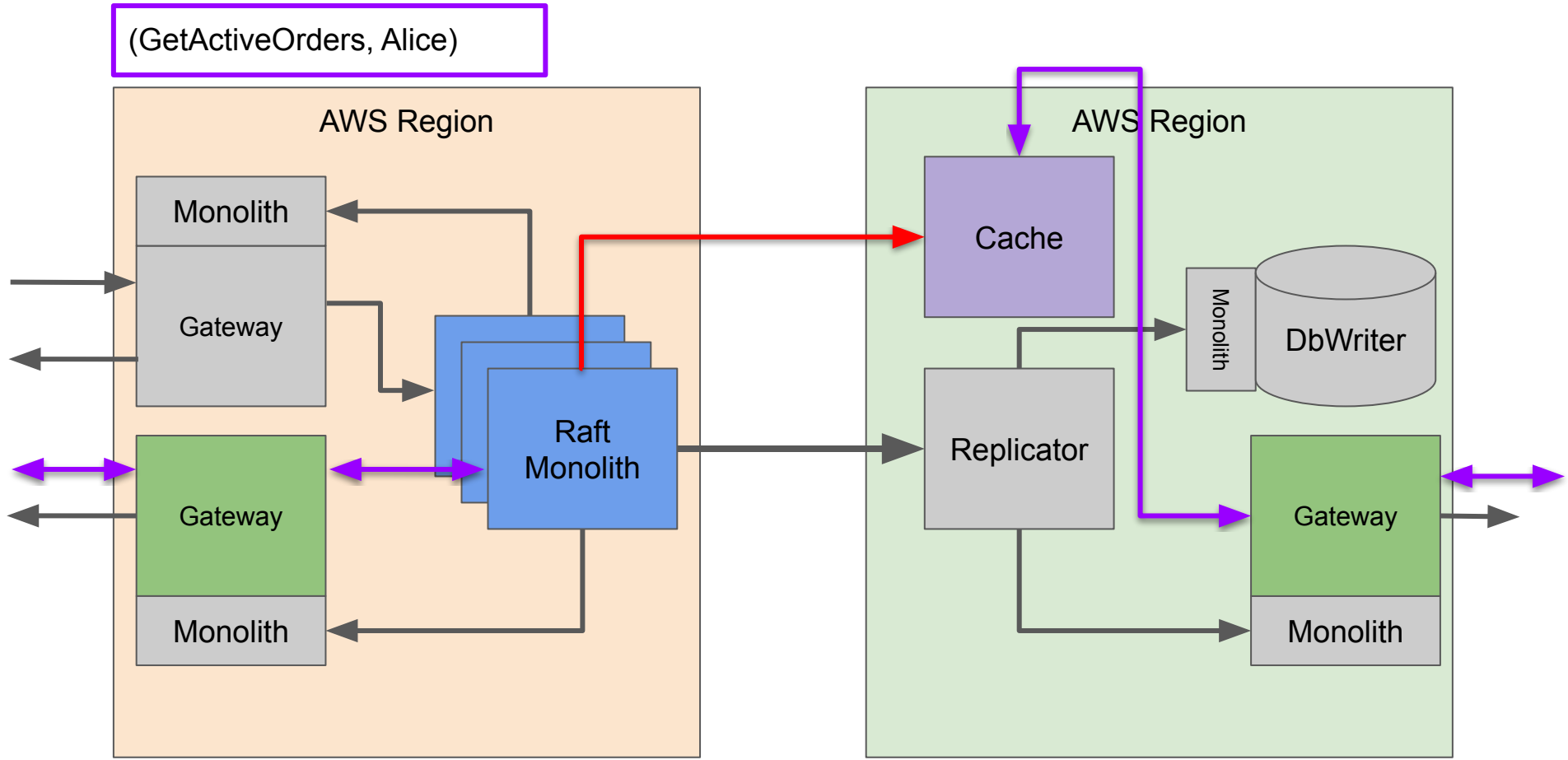
Deduplication is Optimization, not Architecture

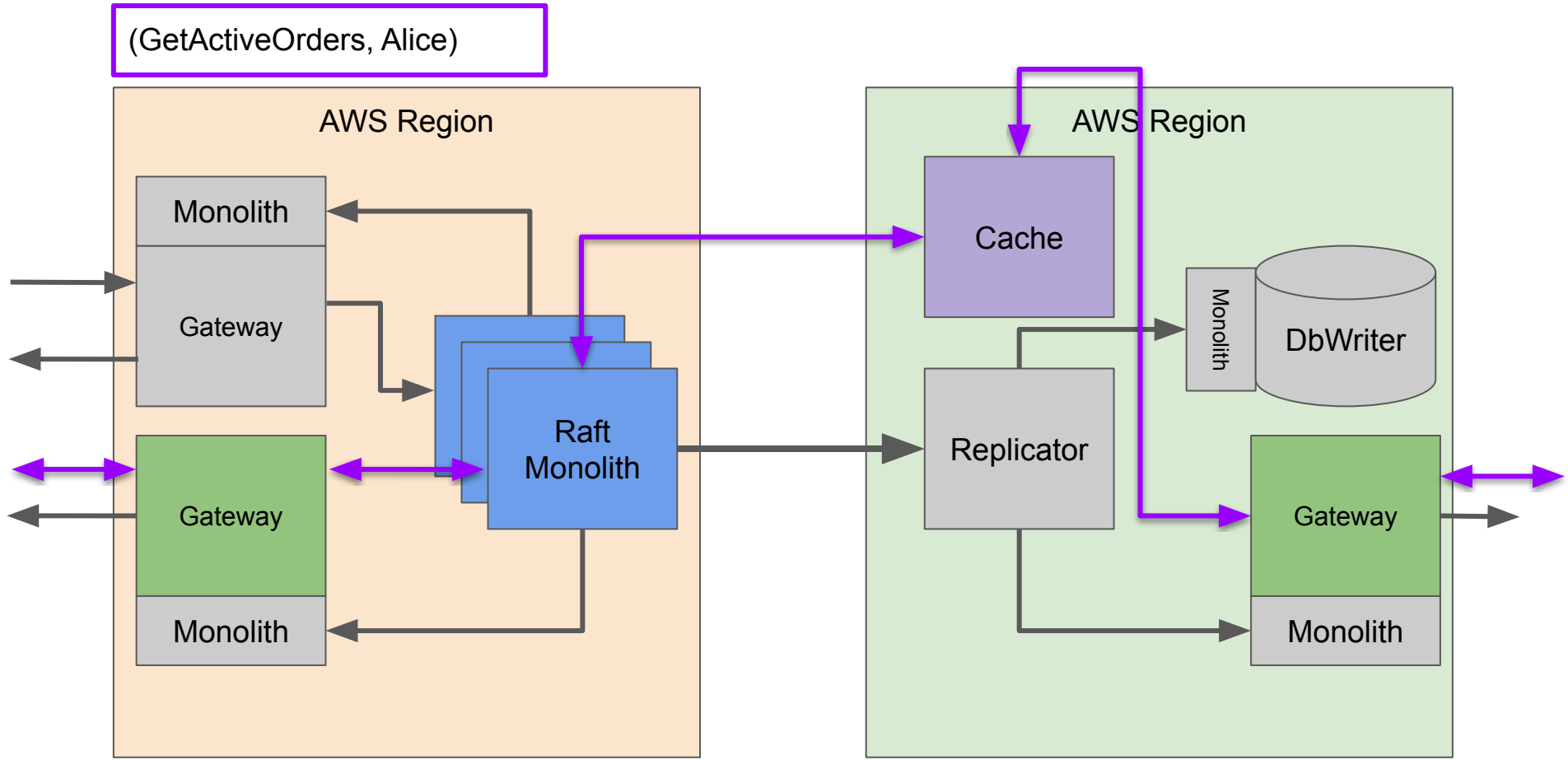

AWS Region
Monolith
Gateway
Gateway
Monolith
Raft
Monolith
AWS Region
Monolith
DbWriter
Replicator
Gateway
Monolith

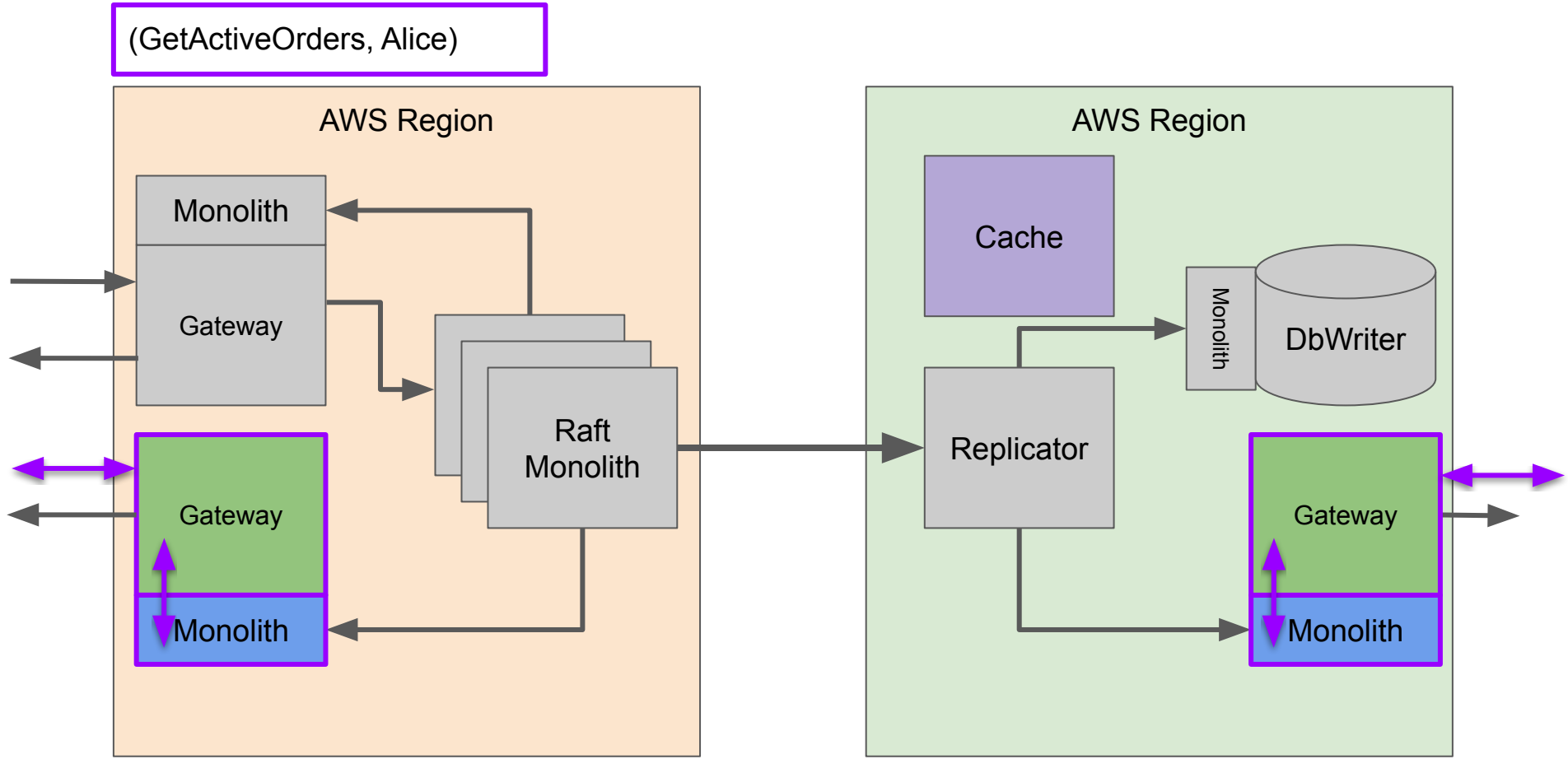

(GetActiveOrders, Alice)
AWS Region
Monolith
Gateway
Gateway
Monolith
Raft
Monolith
AWS Region
Monolith
DbWriter
Replicator
Gateway
Monolith

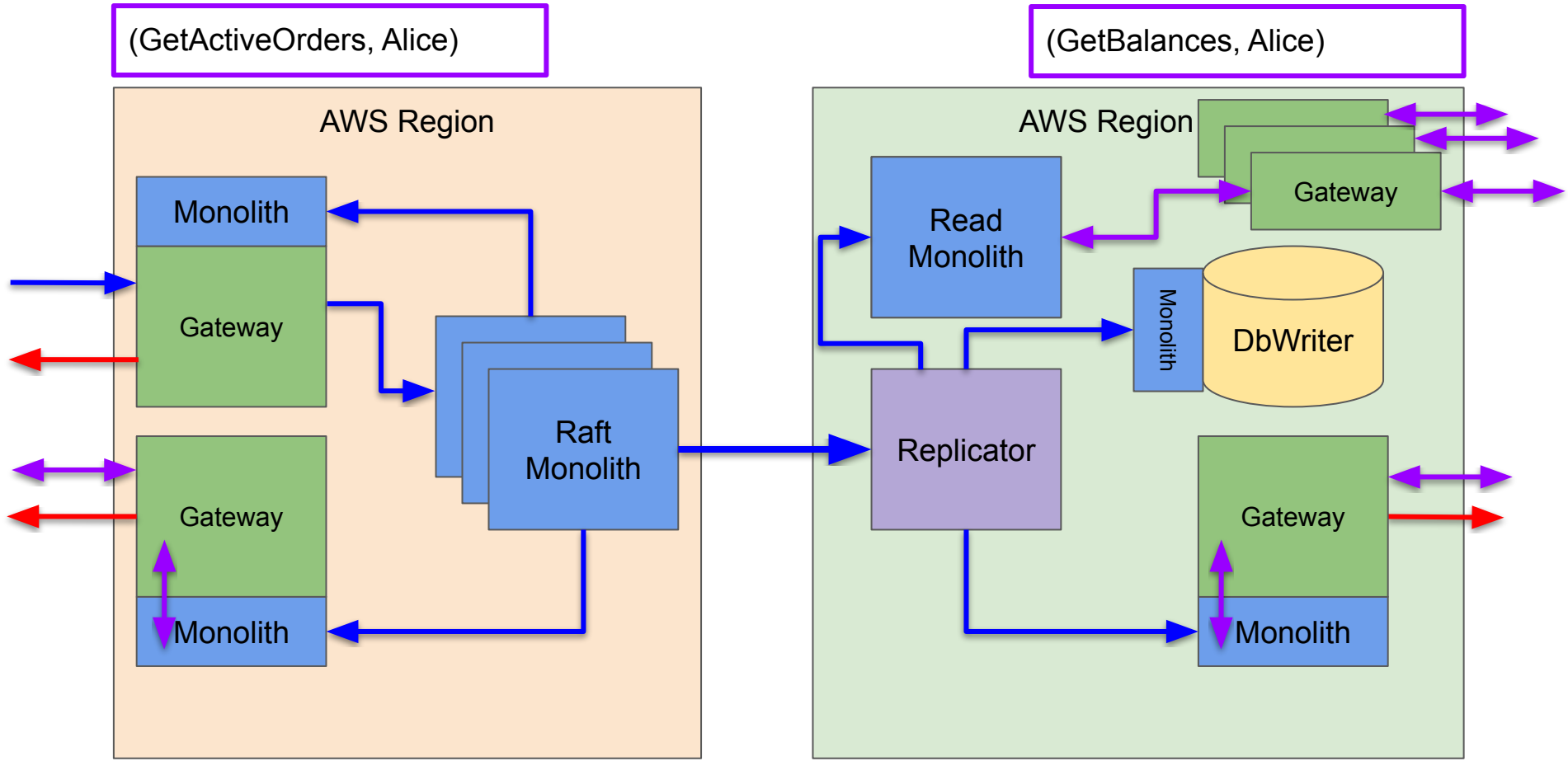

(GetActiveOrders, Alice)
AWS Region
Monolith
Gateway
Gateway
Monolith
Raft
Monolith
AWS Region
Monolith
DbWriter
Replicator
Gateway
Monolith


(GetActiveOrders, Alice)
AWS Region
Monolith
Gateway
Gateway
Monolith
Raft
Monolith
AWS Region
Cache
Monolith
DbWriter
Replicator
Gateway
Monolith


(GetActiveOrders, Alice)
AWS Region
Monolith
Gateway
Gateway
Monolith
Raft
Monolith
AWS Region
Cache
Monolith
DbWriter
Replicator
Gateway
Monolith


(GetActiveOrders, Alice)
AWS Region
Monolith
Gateway
Gateway
Monolith
Raft
Monolith
AWS Region
Cache
Monolith
DbWriter
Replicator
Gateway
Monolith


(GetActiveOrders, Alice)
AWS Region
Monolith
Gateway
Gateway
Monolith
Raft
Monolith
AWS Region
Cache
Monolith
DbWriter
Replicator
Gateway
Monolith


(GetActiveOrders, Alice)
(GetBalances, Alice)
AWS Region
AWS Region
Monolith
Gateway
Gateway
Monolith
Raft
Monolith
Read
Monolith
Gateway
Monolith
DbWriter
Replicator
Gateway
Monolith

Challenges and Considerations

1

Replicate well-tested code or bugs will replicate too

2

No Drift: Old behavior must be respected when replaying inputs

3

Enable new behavior with a request to the monolith after deploy

4

Use a seed for deterministic pseudorandom outputs

5

Divide large chunks of work into stages

6

Everything should fit in memory

7

You’d be surprised how much data fits in memory

8

You’d be surprised how much work fits on one cpu core

9

Keep your 99s and 99.9s Down

10

Protect your monolith from chatty clients

All together now

Simplicity

- Stability
- Performance
- Development Speed

If you have some **important logic**

Make it **deterministic**

If you have **deterministic** logic

Don’t be afraid to **replay it anywhere** for efficiency and profit

Thanks Everyone!

Credits and References

- Todd Montgomery and Martin Thompson: Aeron and Aeron Cluster
- Martin Thompson: Input X State slides