ENGINEERING KIOSK ALPS | 12.06.2025 | INNSBRUCK

Multi-Tenant architectures

Maximilian Schellhorn

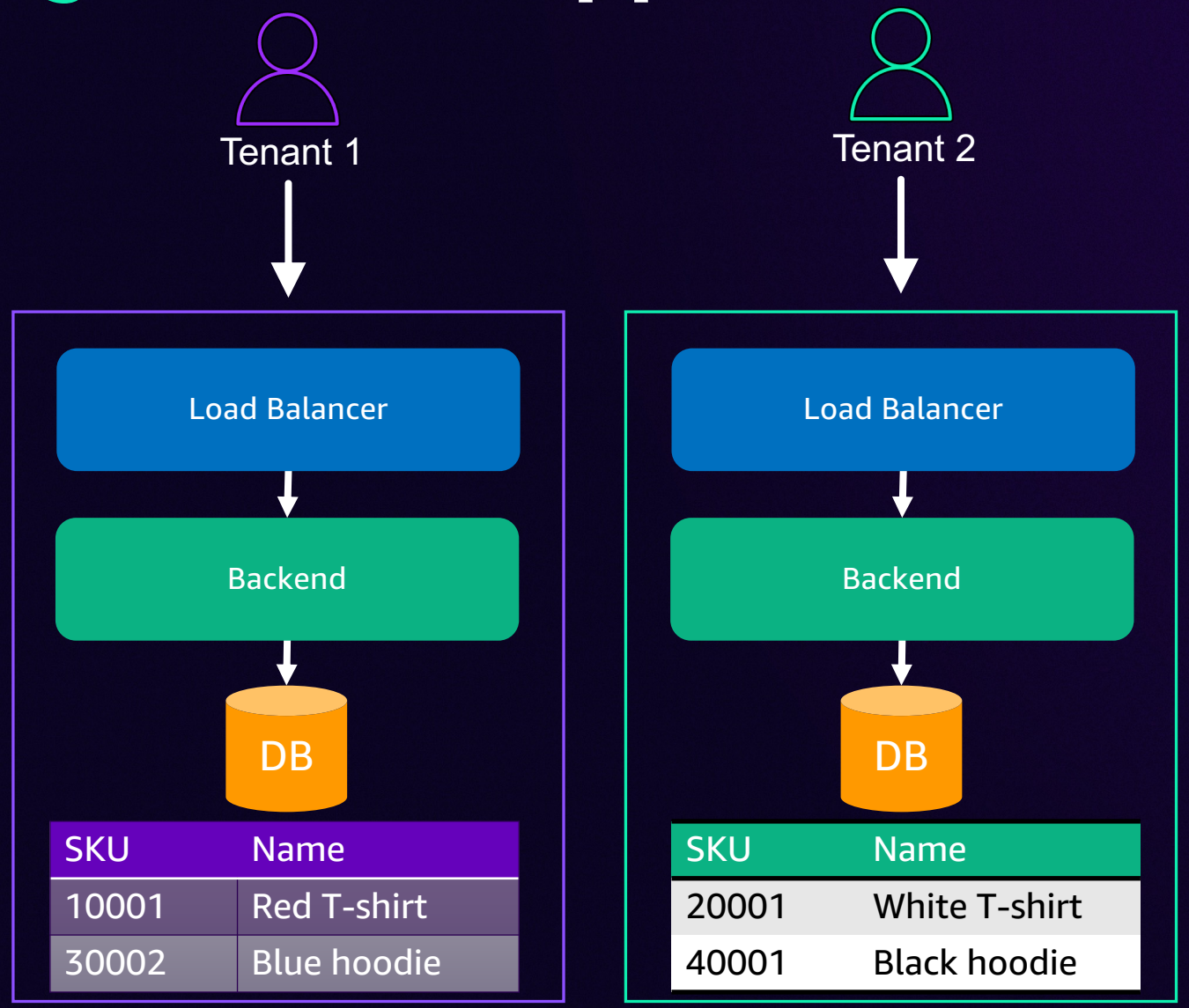
Sr. Solutions Architect
Amazon Web Services (AWS)


aws

Tools NOT Rules

Challenge 1: One size fits all mindset

Single-Tenant application



Benefits

Simple programming model
Tenant identity & isolation by design
Smaller blast radius

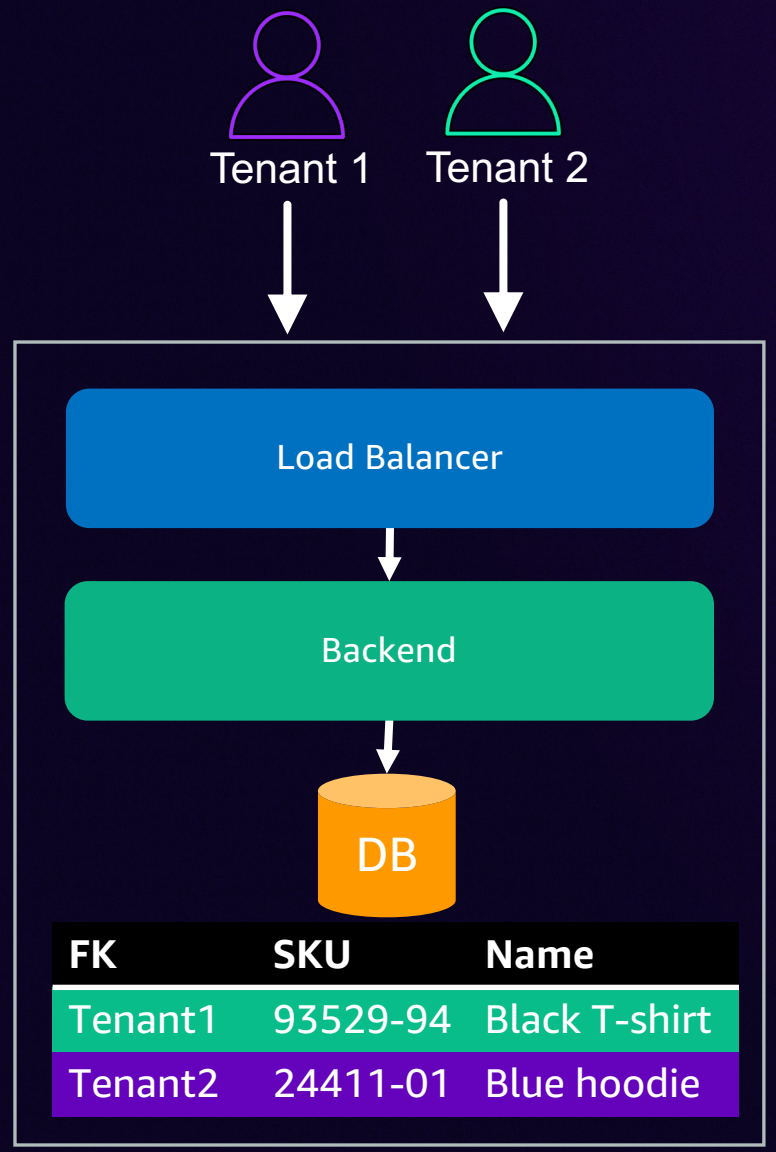
Challenges

Maintenance & Onboarding
Scales linear (time, effort, money)
Enables version drift

Use cases

Static set of (high paying) customers
Capped amount of potential customers
Conservative/Air gapped customers only

Multi-Tenant application



Benefits

Efficient & scalable
Less maintenance & faster updates

Challenges

More complex programming model
Tenant identity & isolation at runtime
Blast radius & Noisy neighbors

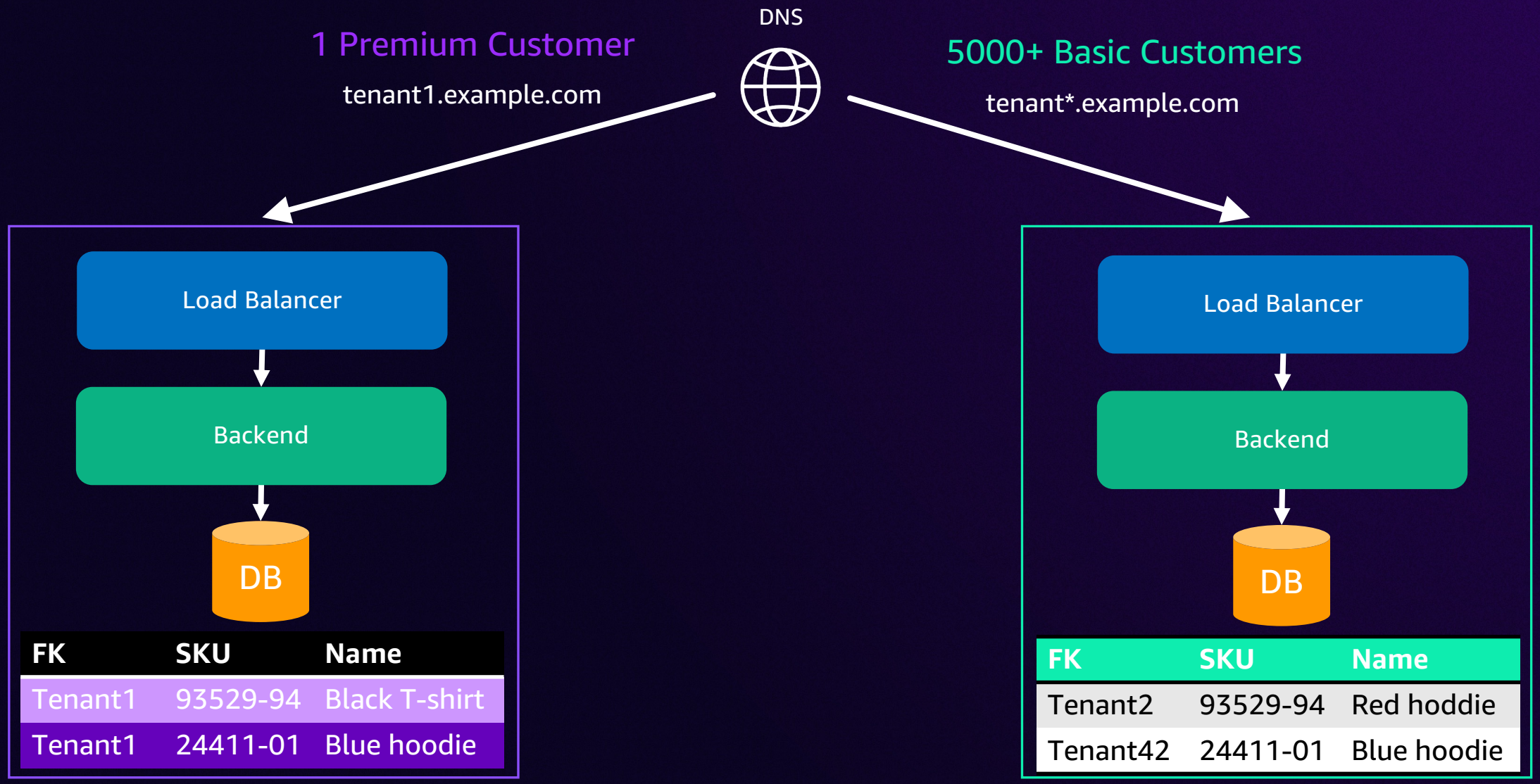
Use cases

Rapid scale & customer growth
Fast time to market & onboarding
Software-as-a-Service (SaaS)

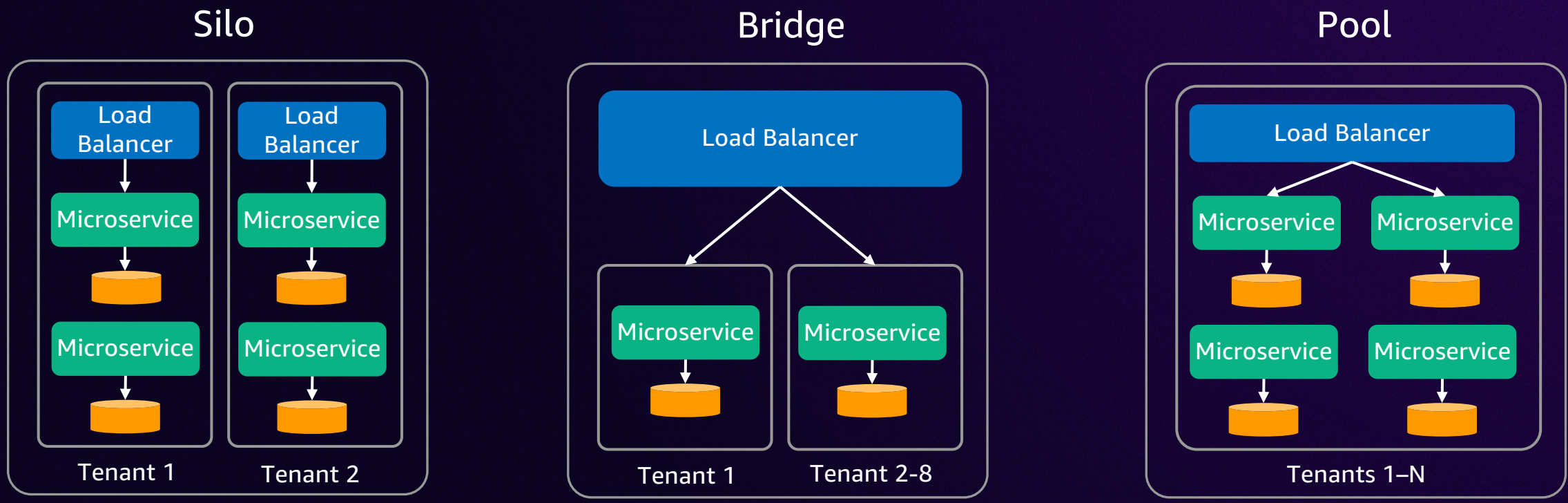
Multi-Tenant systems are better Single-Tenant systems

A Multi-Tenant system deployed for 1 customer = Single-Tenant semantics

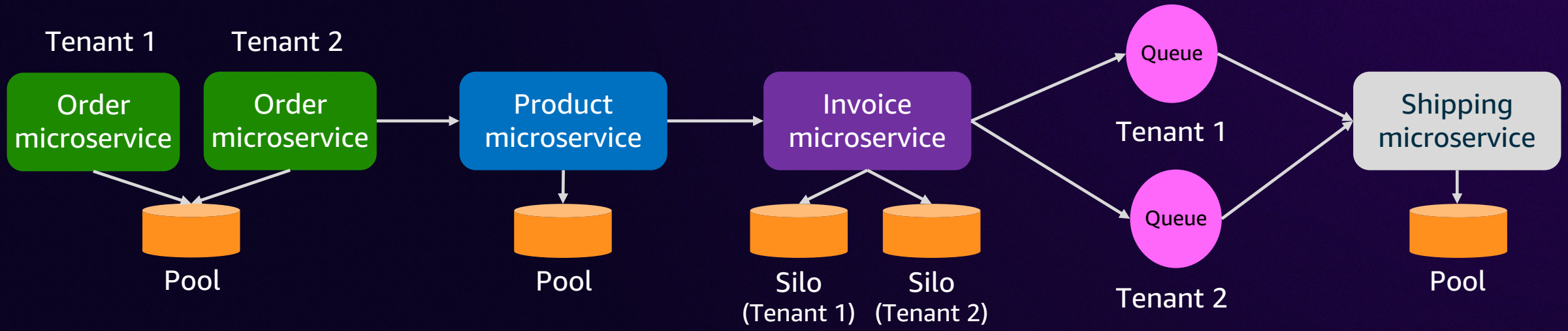
Multi-Tenant applications provide flexibility



Multi-Tenant deployment models



Multi-Tenant deployments in reality



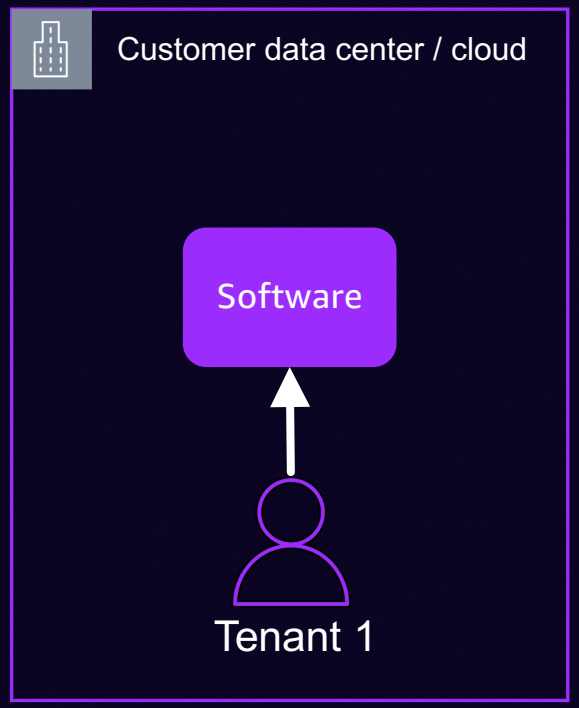
Siloed compute and pooled storage

Pooled compute and pooled storage

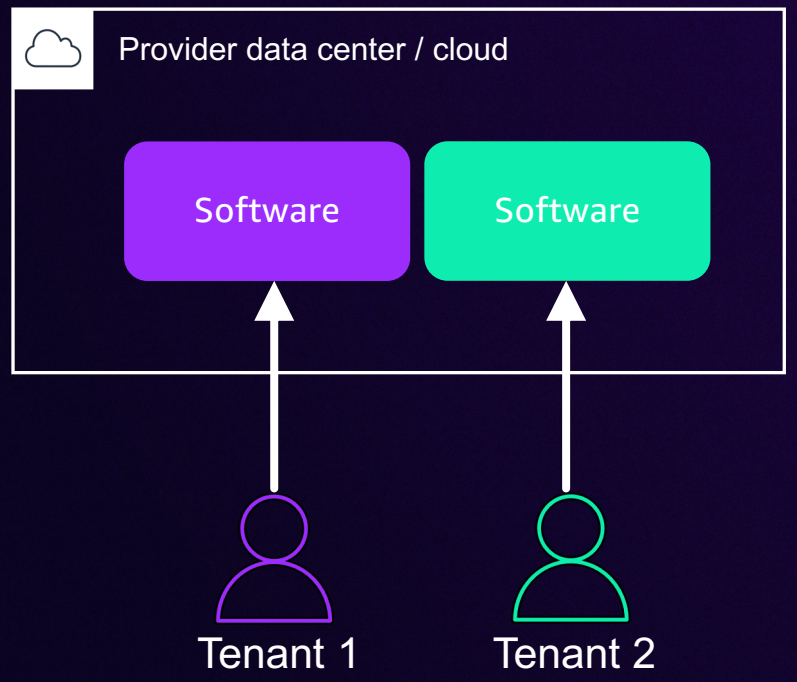
Pooled compute and siloed storage

Pooled compute and storage consuming siloed queues

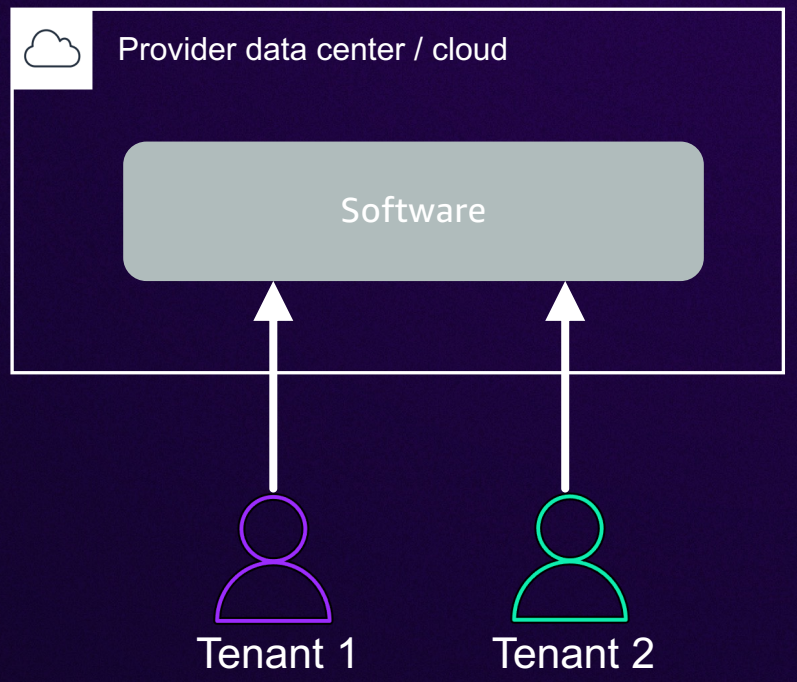
Self-Managed vs. SaaS



Single-Tenant Self-Managed



Single/Multi-Tenant (Silo) SaaS



Multi-Tenant (Pooled) SaaS

Before we were doing SaaS, we did not prioritize the infrastructure footprint of our software because it was the customer's responsibility to provide and pay for the hardware. With SaaS, every dollar of unoptimized infrastructure is lost business.

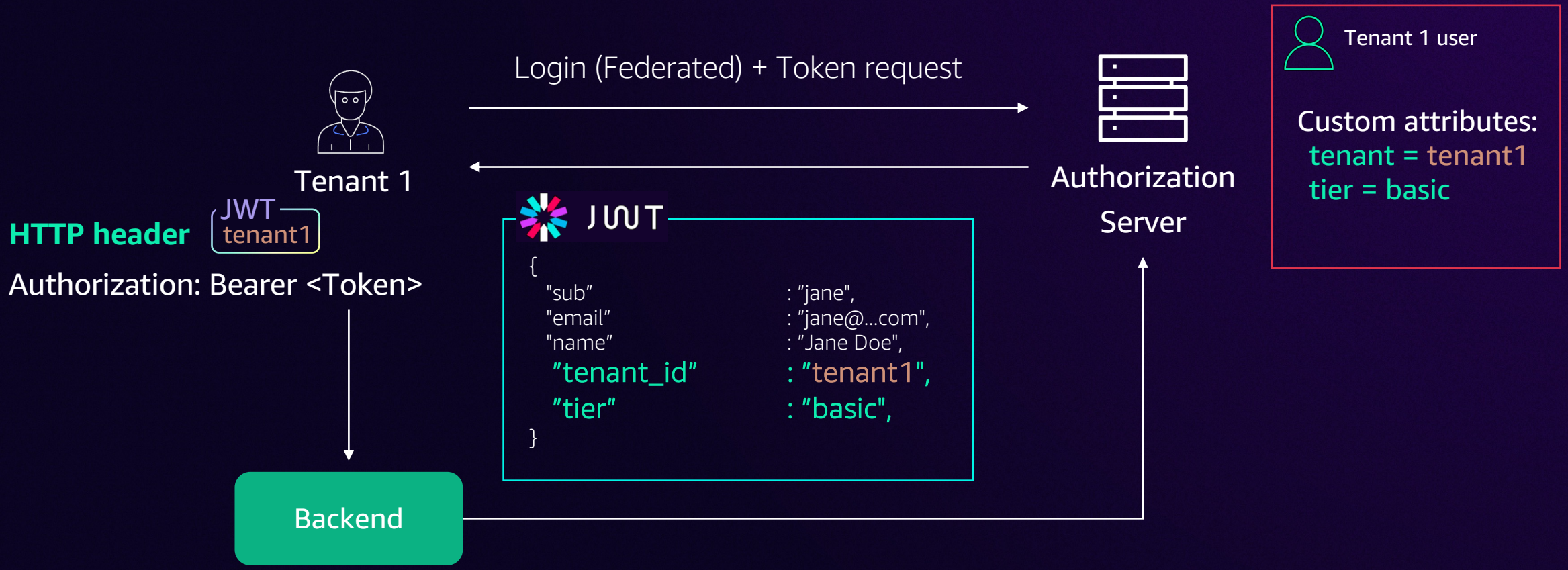
CTO, SaaS company

Engineering is the art of not-constructing rather than constructing. It is the art of doing that well with 1$, which any bungler can do with 2$ after fashion.

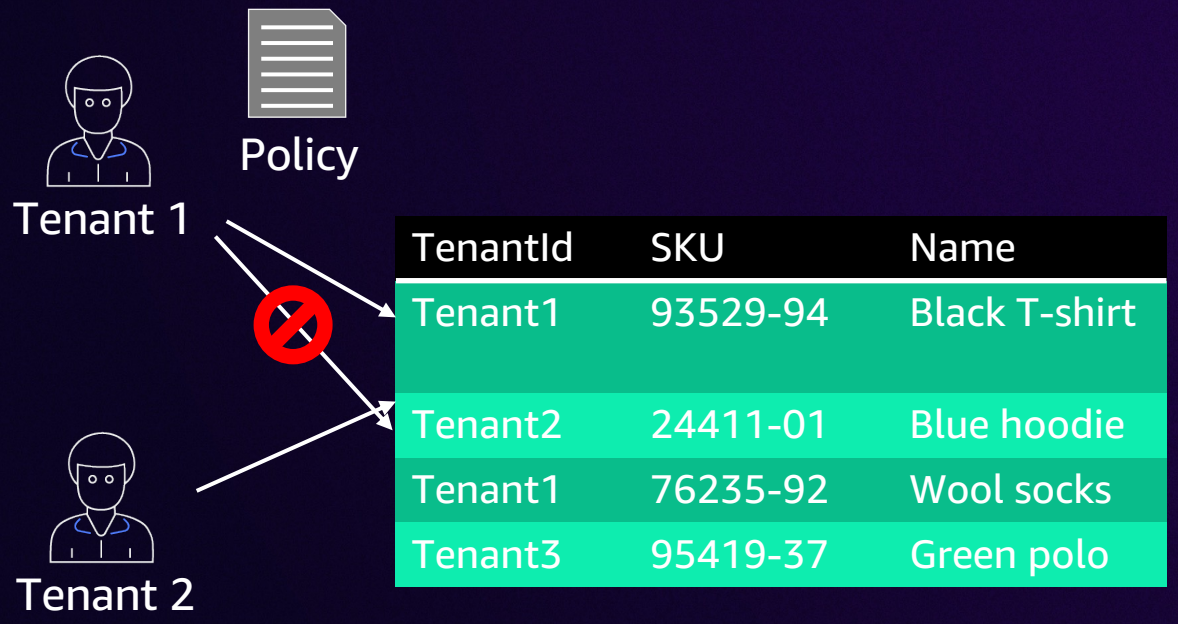
Arthur M. Wellington

Challenge 2: Tenant identity & isolation

Runtime identity with tokens (JWT)

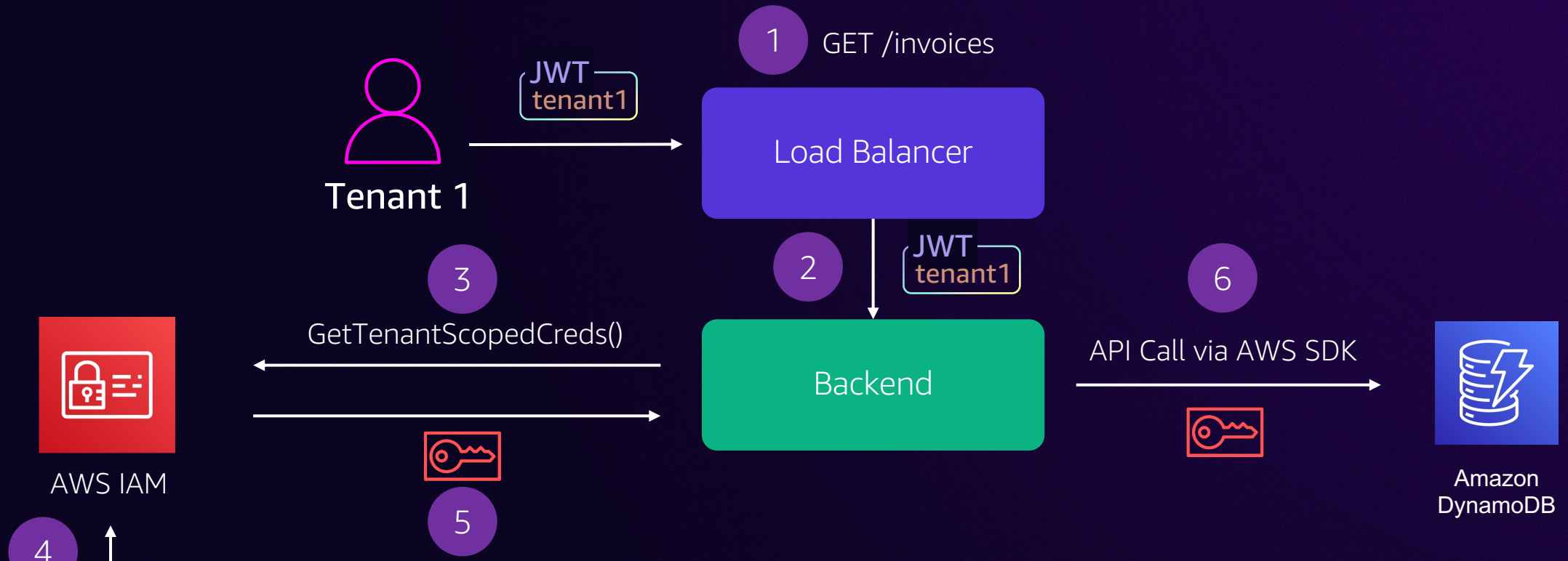


Tenant isolation at runtime



Single table separated by identifier

API based runtime isolation




```
Effect: Allow
Action: dynamodb:GetItem
Resources:
-  arn:aws:dynamodb:table/Invoices
Condition:
-   dynamodb:LeadingKeys:{TenantID}
```

TENANT_ID (PK)	INVOICE_ID (SK)	DETAILS
TENANT#1	2023-001	…
TENANT#2	2022-001	…

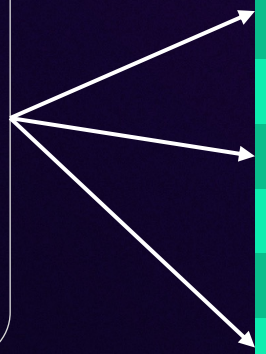
PostgreSQL Row Level Security (RLS)

Initialize RLS

```
-- Turn on RLS
ALTER TABLE tenant ENABLE ROW LEVEL SECURITY;

-- Scope read/write by tenant
CREATE POLICY tenant_isolation_policy ON tenant
USING (tenant_id = (current_setting('app.current_tenant'::text)));
```

Query with RLS

```
-- SET tenant context
rls_multi_tenant=> SET app.current_tenant = 'tenant1';

-- No tenant context required
rls_multi_tenant=> SELECT * FROM tenant;

-- Attempt to force other tenant id would not work
rls_multi_tenant=> SELECT * FROM tenant WHERE tenant_id = 'tenant2'
```



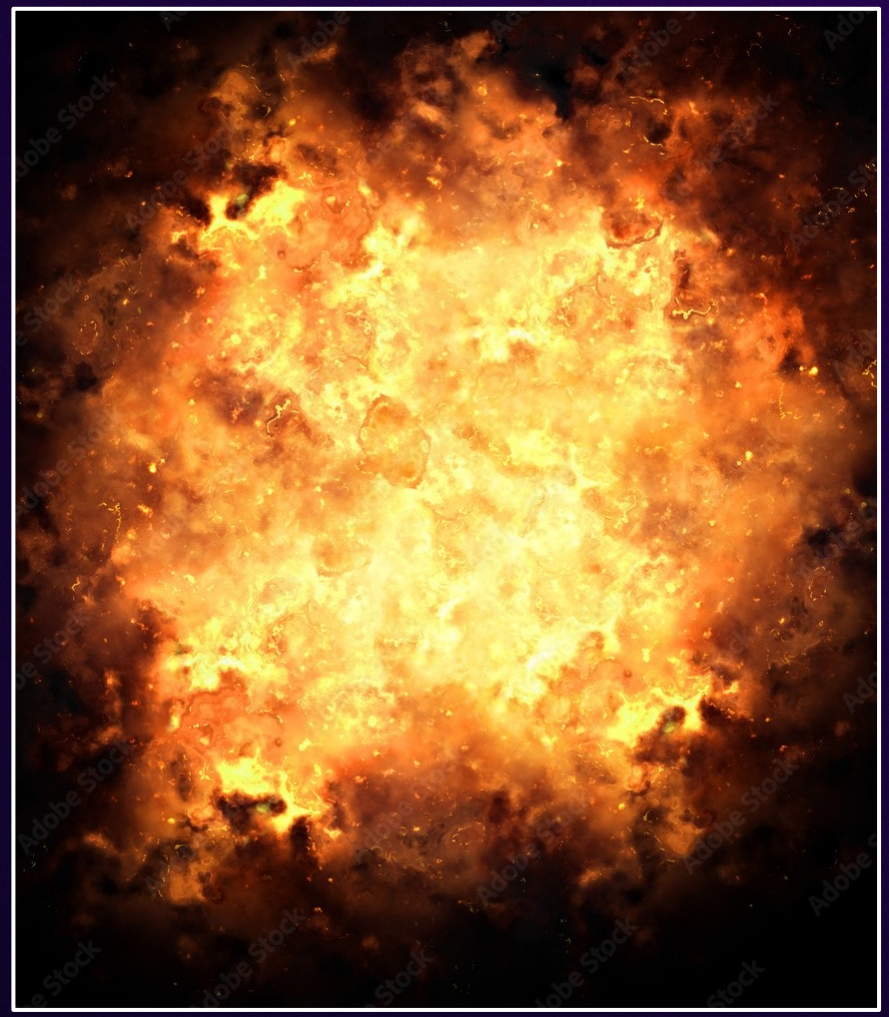
FK	SKU	Name
Tenant1	93529-94	Black T-shirt
Tenant2	24411-01	Blue hoodie
Tenant1	76235-92	Wool socks
Tenant3	95419-37	Green polo
Tenant2	88314-99	White hat
Tenant1	24598-72	Tennis shoes

Challenge 3: Blast radius & noisy neighbors

Ignoring your blast radius

"The maximum impact that might be affected in the event of a system failure"

0 - 100%



How do we usually protect against failures?

Failure Mode	Means of protection
Rack Down 	Autoscaling Groups
AZ Failure 	Multi-AZ Deployments
Regional Failure 	Region Failover
Bad Deployment 	Canary Deployments
Human Failure 	Processes, Runbooks
Poison Pill & noisy neighbor 	Extensive testing

Typically only for Compute

Can't be fully mitigated.

But what about …

A tenant triggering a very specific bug?

A deployment that makes the production network unusable?

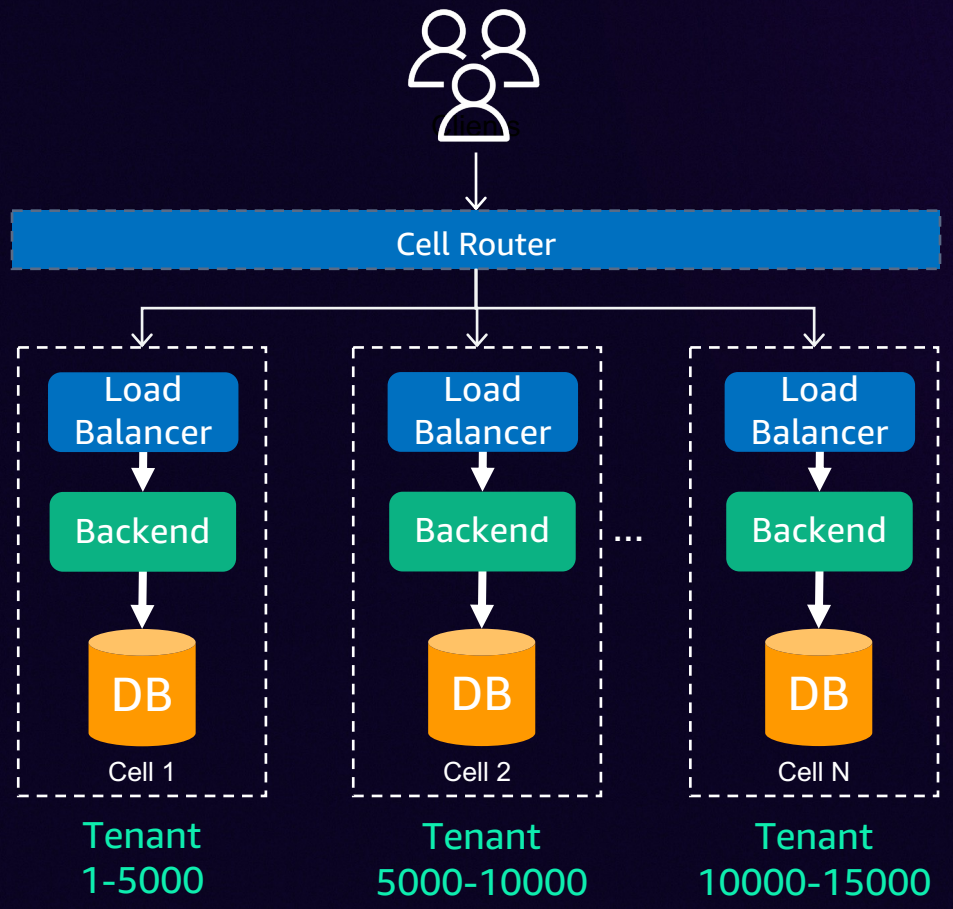
A poisoned request that wipes the database?

A spike from one customer saturating all capacity?

Someone accidentally deleting the main loadbalancer?

Testing makes these failure unlikely to happen.
At scale, even unlikely scenarios happen regularly.

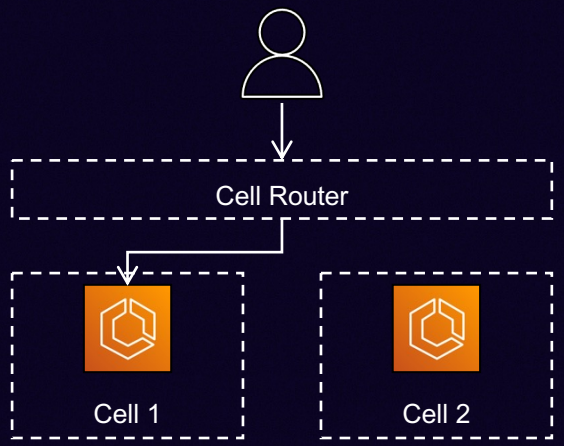
Solution: Cell based architecture



- Multiple copies of the **entire application** deployed in each cell.
- Data is **partitioned** – there is no replication between cells.
- **Complete isolation** between cells limit and contain failure.
- **Linear scale out** for additional clients
- Testability & Rollout benefits

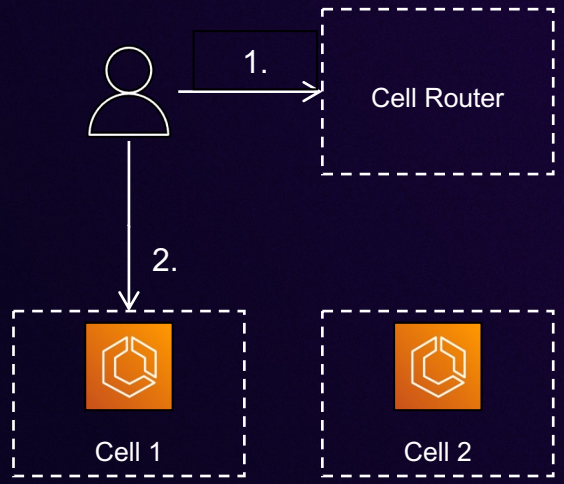
Cell-Routing options

A) Cell Router as Load Balancer



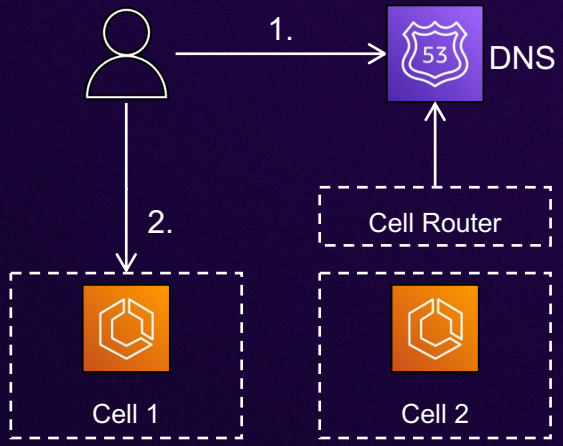
- ✓ Transparent to clients
- \- Cell Router is critical to ongoing transactions

B) Router forwards the request



- ✓ Simple to implement
- ✓ Clients with cached routing information unaffected by failure
- \- Custom logic needed on client
- \- Potentially higher latency

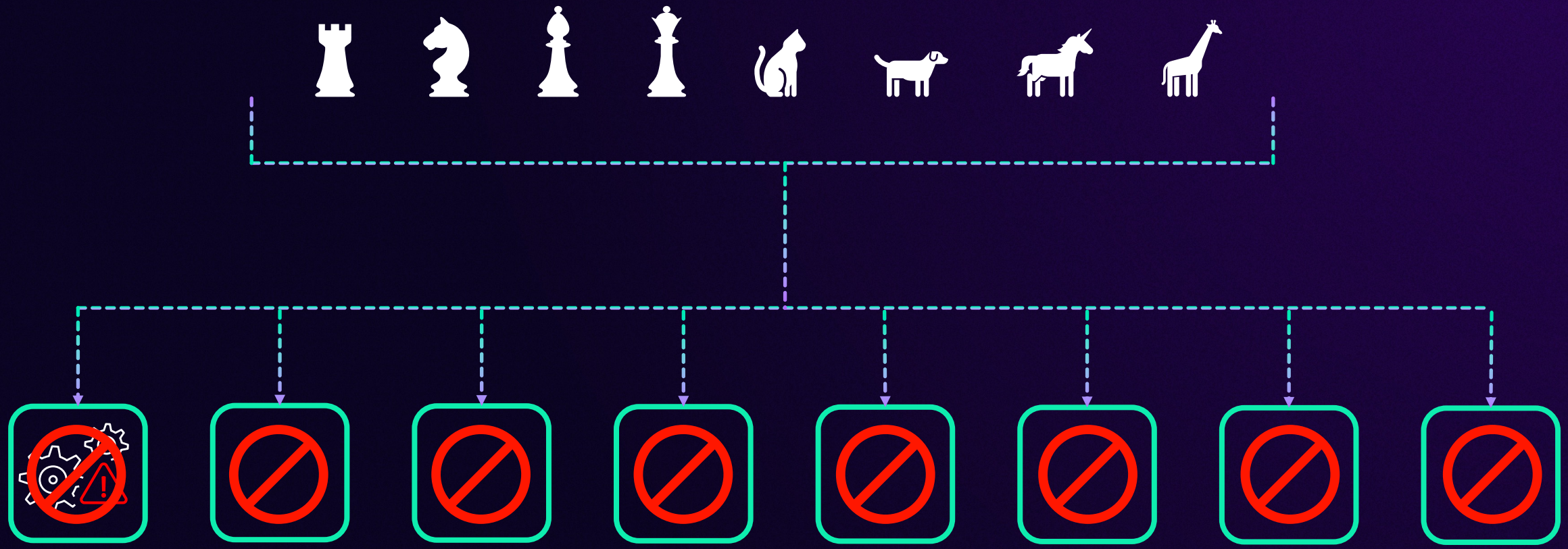
C) Routing through DNS



- ✓ Most DNS providers offer high availability.
- \- Clients need to map users to DNS names

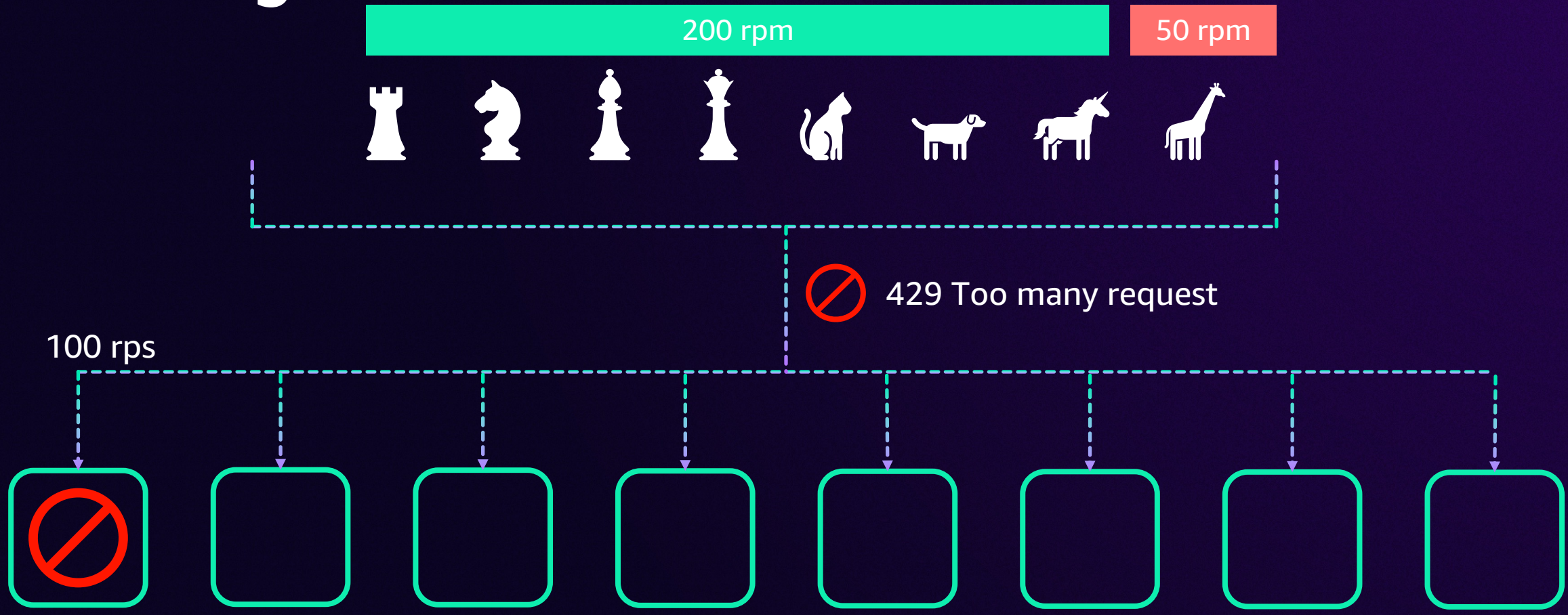
Noisy Neighbor & Poison pills

Traditional architecture

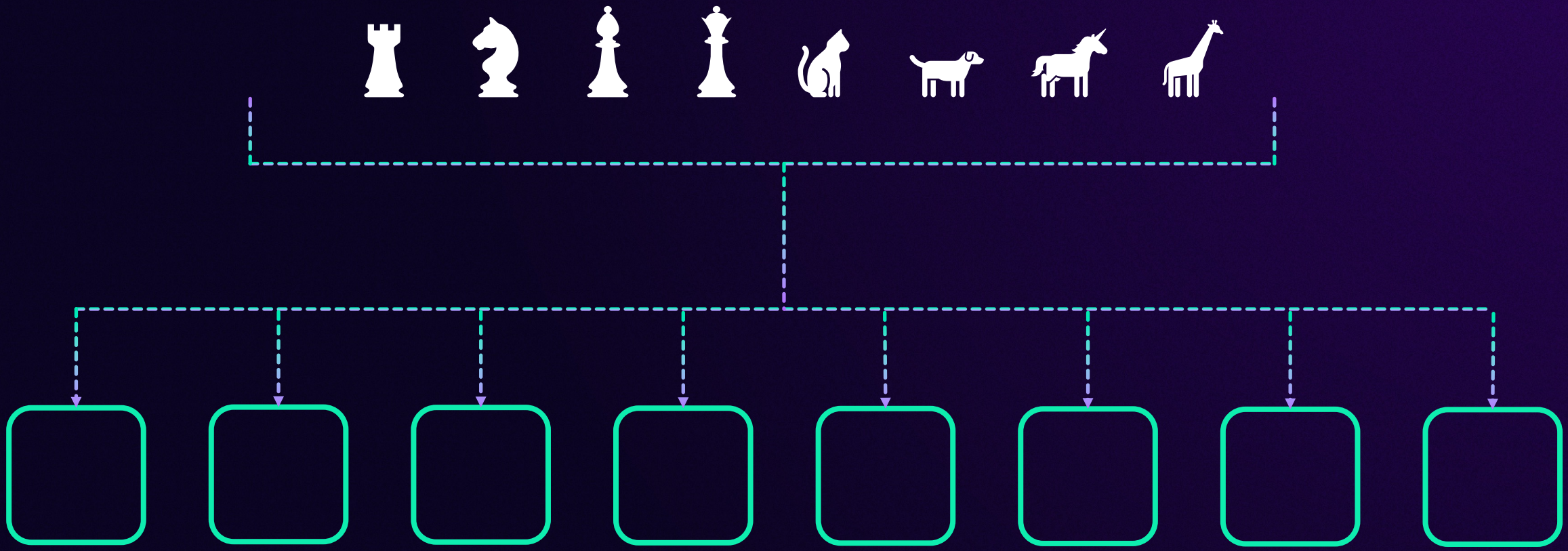


Scope of impact = All customers

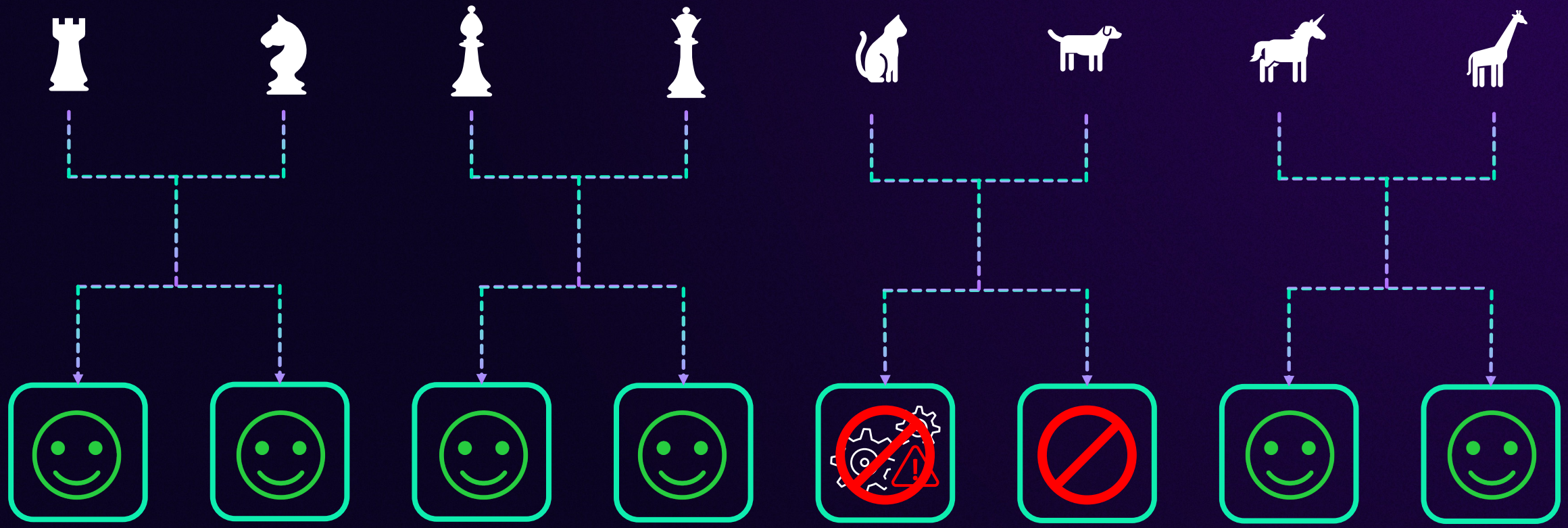
Throttling



Sharding

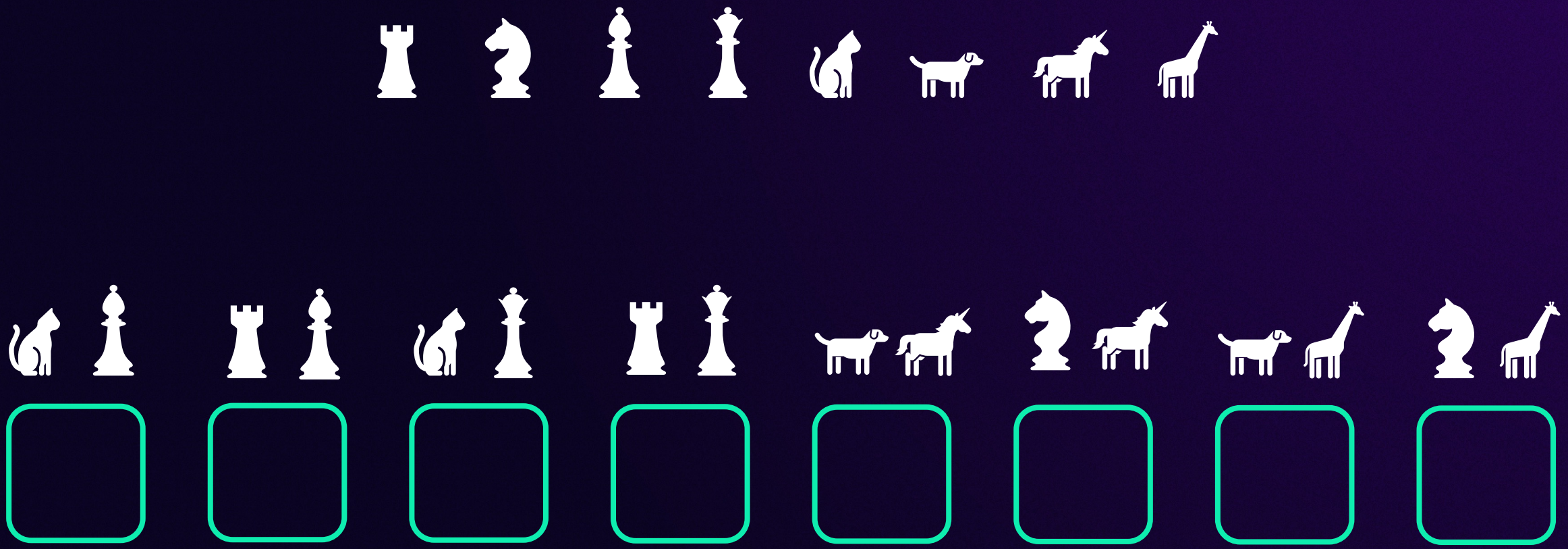


Sharding



$$\text{Scope of impact} = \frac{\text{Customers}}{\text{Shards}}$$

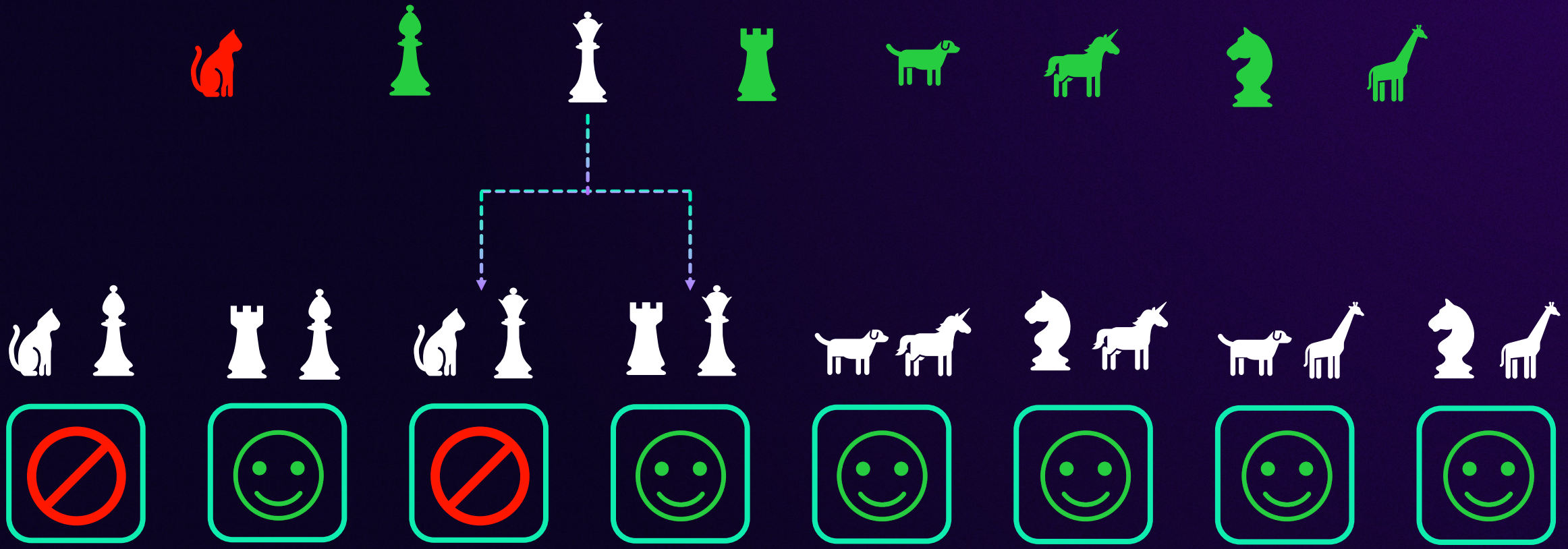
Shuffle sharding



Shuffle sharding

Shuffle sharding

Shuffle sharding



$$\text{Scope of impact} = \frac{\text{Customers}}{\text{Combinations}}$$

Multi-Tenant (SaaS) architectures

Challenge	Tool
 **One size fits all mindset**	Choose the right tenant isolation model and be flexible
 **Tenant identity & isolation**	JWT tokens, API-, Service-, Encryption-based Runtime policies
 **Blast radius & Noisy neighbors**	Cell-Based architectures, Throttling and Shuffle-Sharding

Thank you!

Maximilian Schellhorn
mxschell@amazon.com
linkedin.com/in/maxschell/