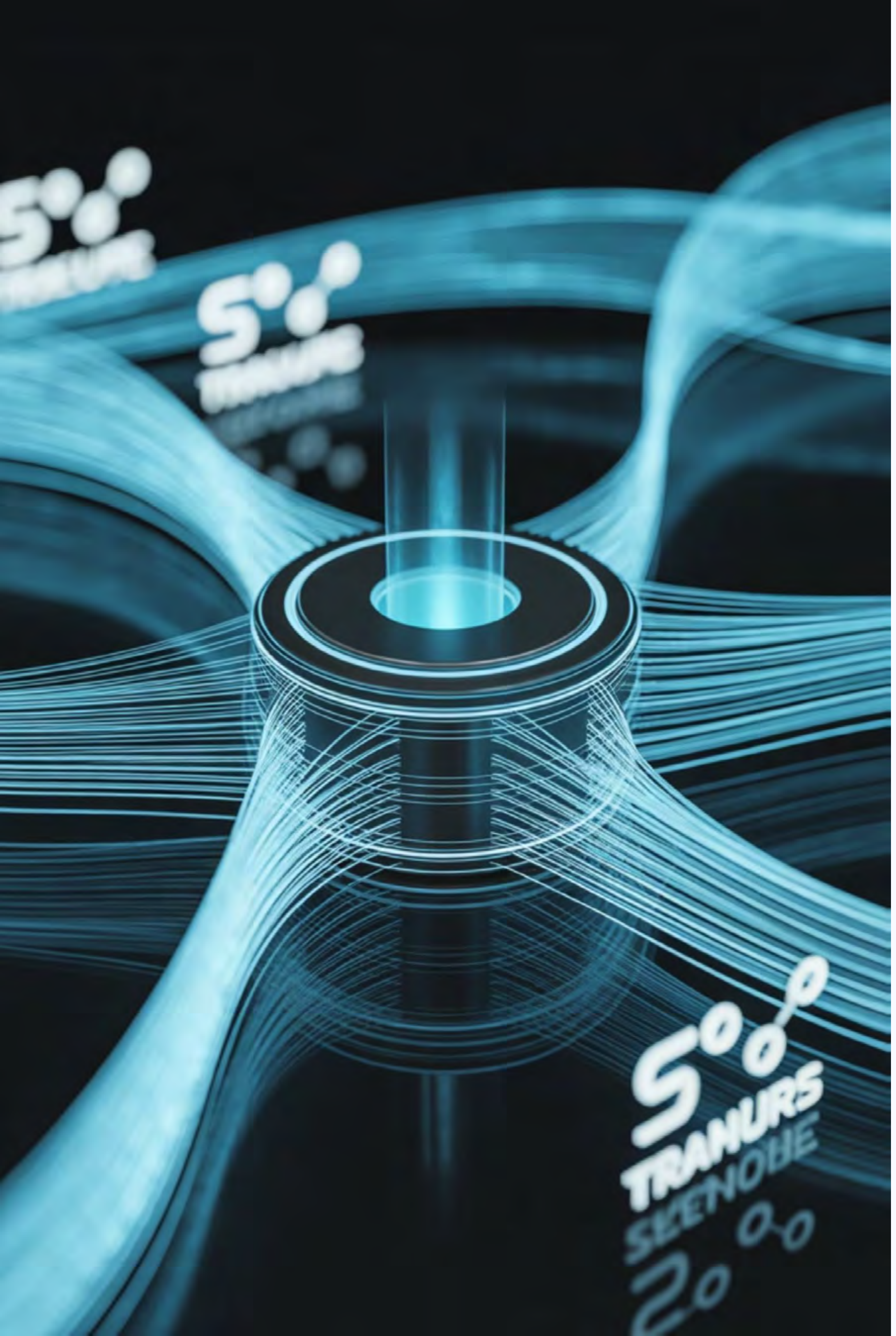


Platform Engineering for Subscription Payment Systems

Building Self-Service Infrastructure at Scale

Presented By: **George Thomas** Engineering Manager

Today's Agenda

01

The Evolution of Payment Infrastructure

From monolithic processors to distributed architectures

02

Platform Engineering Principles for Payments

Creating self-service capabilities while maintaining compliance

03

Building Resilient Payment Systems

Circuit breakers, retry mechanisms, and failure recovery

04

Observability & Compliance as Platform Features

Visibility without compromising security

05

Scaling Across Business Models

B2C, B2B, and usage-based billing platforms

By the end of this session, you'll understand how to transform payment systems from cost centers into competitive advantages through platform engineering.

From Cost Center to Competitive Advantage

Traditional Payment Systems

- Brittle, manual processes
- Siloed payment knowledge
- High cognitive load for teams
- Slow time-to-market for new features
- Limited visibility into failures

Platform-Driven Approach

- Self-service infrastructure
- Abstracted payment complexity
- Automated recovery processes
- Reduced integration friction
- Millions recovered in failed transactions

The Evolution of Payment Infrastructure

1

Monolithic Payment Processing

Single payment provider, tightly coupled systems, limited flexibility

2

Multi-Provider Integration

Multiple payment methods, but with custom integration code for each provider

3

API-First Payment Services

Abstracted payment logic, but still requiring manual intervention for failures

4

Event-Driven Payment Platforms

Distributed architectures with automated recovery, observability, and self-service capabilities

Modern subscription businesses have moved beyond simple transactions to complex payment lifecycles that require sophisticated platform capabilities.

Core Platform Engineering Principles for Payments

Abstract Away Complexity

Hide payment provider details behind well-designed interfaces that prevent common integration pitfalls

Self-Service by Default

Enable teams to implement payment flows without specialized knowledge or platform team intervention

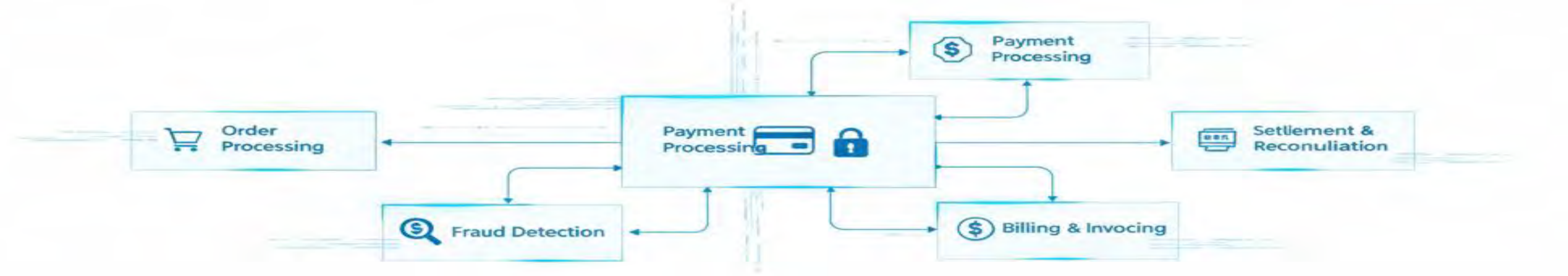
Compliance as Code

Embed regulatory requirements into platform capabilities rather than team responsibilities

Paved Roads for Common Paths

Provide optimized solutions for standard payment scenarios while allowing customization when needed

The goal: Reduce cognitive load for product teams while maintaining robust payment operations.



Building the Payment Platform Architecture

Payment Gateway API

Unified interface abstracting multiple payment providers

Subscription Engine

Manages recurring billing cycles and payment schedules

Event Processing

Handles subscription lifecycle events and payment status changes

Revenue Recovery

Automated retry logic and dunning management

Designing Resilient Payment Systems

The Challenge

Payment providers have 99.9% uptime, which means:

- 8.76 hours of downtime per year
- 43.8 minutes of downtime per month
- 10.1 minutes of downtime per week

For high-volume businesses, even minutes of downtime can mean thousands in lost revenue

Platform Solutions

Circuit Breakers

Prevent cascading failures when providers are down

Intelligent Retry Logic

Exponential backoff with provider-specific optimization

Fallback Processing

Automatic routing to alternative payment methods

Case Study: Revenue Recovery at Scale

Before Platform Engineering

- 15% of subscription payments fail on first attempt
- Manual retry processes based on best guesses
- No visibility into recovery performance
- ~40% recovery rate on failed payments

After Platform Implementation

- Smart retry algorithms based on card issuer and error codes
- A/B testing framework for retry strategies
- Real-time monitoring dashboards
- 78% recovery rate - resulting in millions in recovered revenue

Implementing PCI-DSS Compliance as Platform Capabilities

1

Tokenization Service

Secure API that handles payment data without exposing it to application code

- Vault integration with major payment processors
- Zero PCI scope for product teams

2

Infrastructure as Code Templates

Pre-hardened infrastructure components that enforce security controls

- Network isolation patterns
- Compliance-tested storage configurations

3

Automated Compliance Testing

Continuous verification of security controls in CI/CD pipelines

- Policy as code frameworks
- Automated security scanning

By building compliance into the platform, we reduce the burden on teams while maintaining rigorous security standards.

Observability: The Platform Engineer's Secret Weapon

Payment-Specific Observability Patterns

Traditional observability isn't enough for payment systems. You need:

- Tracing across the entire payment lifecycle
- Provider-specific health metrics
- Business-level KPIs tied to technical indicators
- Anonymized monitoring that protects PII/financial data

Platform teams must build observability that bridges technical and business contexts while maintaining data security.

Scaling Payment Platforms Across Business Models

B2C Subscription Platforms

- Optimized for conversion and retention
- A/B testing frameworks for payment flows
- Localized payment method support
- Simplified subscription management

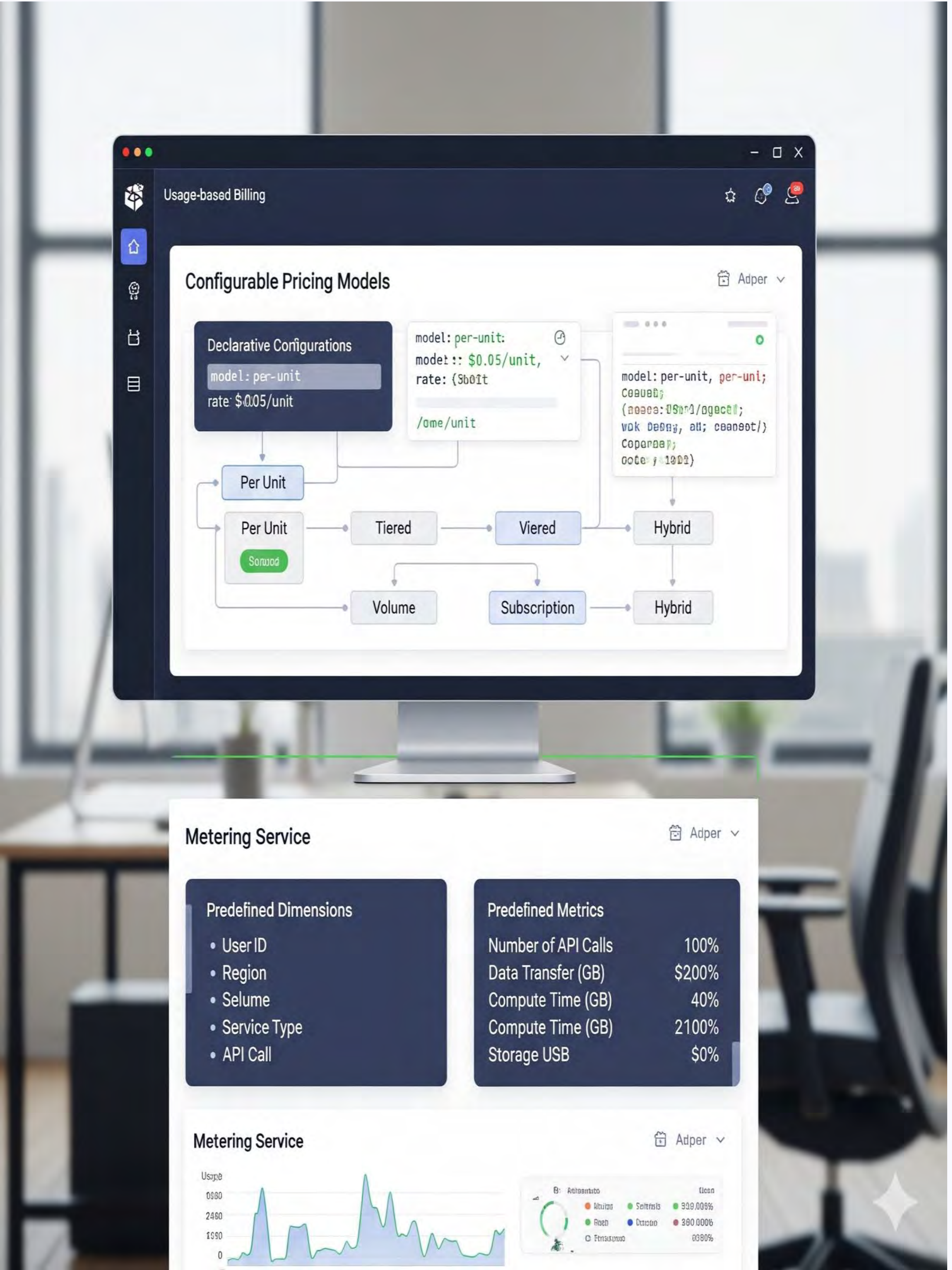
B2B Enterprise Platforms

- Complex billing hierarchies
- Custom contract terms and billing cycles
- Integration with procurement systems
- Multi-entity invoicing capabilities

Usage-Based Billing Platforms

- Real-time usage metering
- Flexible consumption models
- Predictable billing for variable usage
- Complex rating and pricing rules

The core platform principles remain consistent, but implementation details vary significantly across business models.



Deep Dive: Usage-Based Billing Platform Challenges

Technical Complexity

- High-volume event ingestion (millions of events per second)
- Real-time aggregation across distributed systems
- Complex metering abstractions for varied usage patterns
- Materialized views for billing snapshots

Platform Solutions

- Metering service with predefined dimensions and metrics
- Self-service dimension registration
- Usage estimation APIs for customer-facing dashboards
- Configurable pricing models through declarative configurations

Building a Developer Experience for Payment Systems

API-First Design

Consistent, well-documented APIs with strong contracts and validation

Developer Tooling

SDKs, CLI tools, and sandbox environments for local testing

Self-Service Documentation

Interactive API docs, implementation guides, and solution patterns

Internal Developer Portal

Centralized access to payment resources, templates, and tools

The success of payment platforms depends on making integration frictionless for engineers who aren't payment experts.

Measuring Platform Success: Beyond Uptime

Platform Engineering Metrics

Developer Velocity

Time to implement new payment features

Self-Service Rate

Percentage of teams implementing without platform assistance

Integration Quality

Reduction in payment-related incidents

Business Impact Metrics

Revenue Recovery

Percentage of failed payments successfully recovered

Payment Conversion

Checkout success rate across payment methods

Subscription Retention

Reduction in involuntary churn due to payment failures

Effective payment platforms demonstrate value through both engineering efficiency and business outcomes.

Key Takeaways for Payment Platform Engineering

Start With Developer Experience

Build payment platforms that product teams actually want to use, with clear abstractions and self-service capabilities.

Design For Failure Recovery

Payment systems will fail. The difference between good and great platforms is how they handle those failures automatically.

Embed Compliance

Security and regulatory requirements should be platform features, not team burdens.

Connect Technical and Business Metrics

The ultimate measure of payment platform success is recovered revenue and reduced payment friction.

Thank You