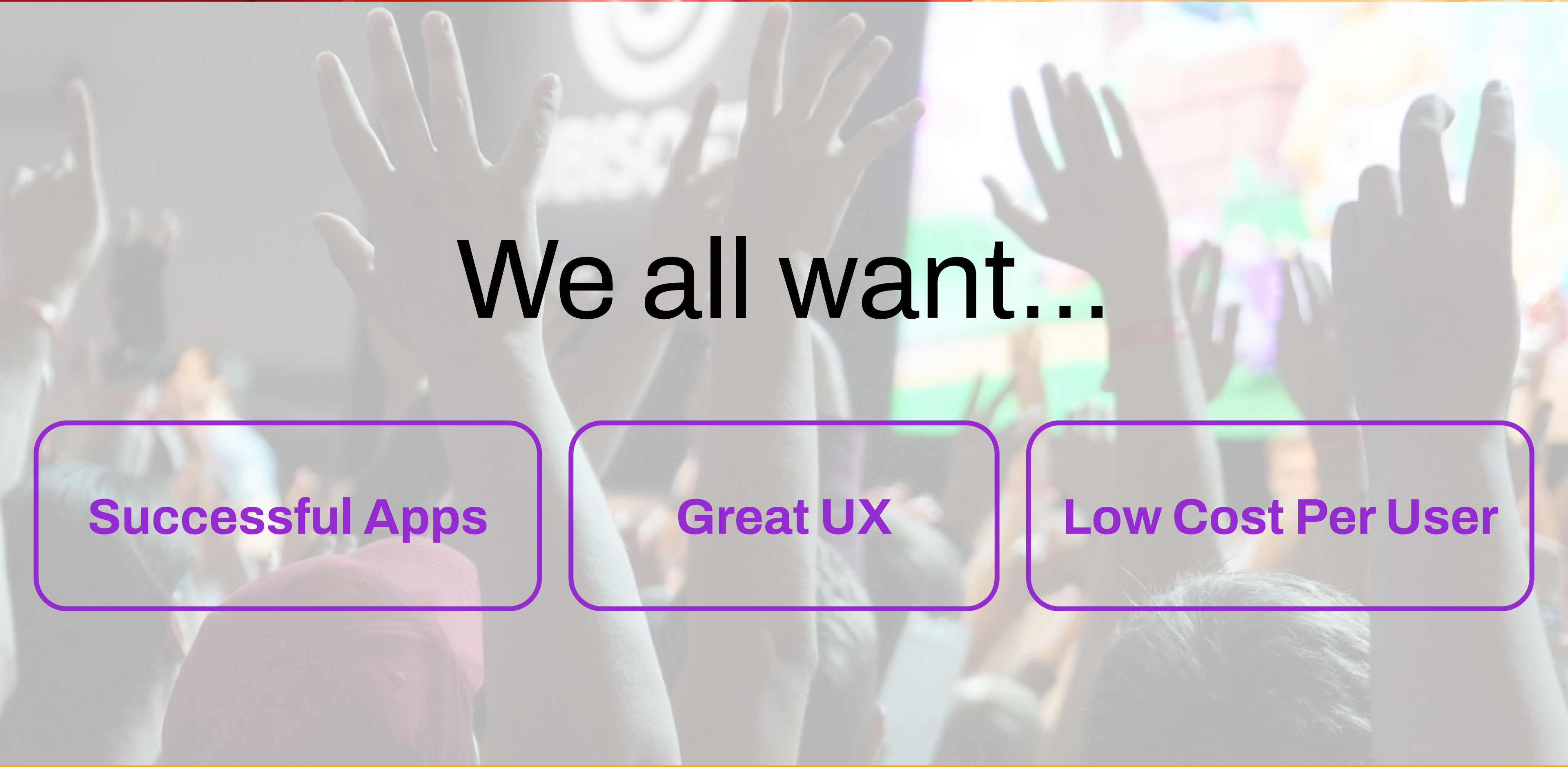
Briefcasing RESTful data to reduce network traffic

Stephen Ball
Pre-sales Director
Embarcadero Technologies
Stephen.ball@embarcadero.com



We all want...

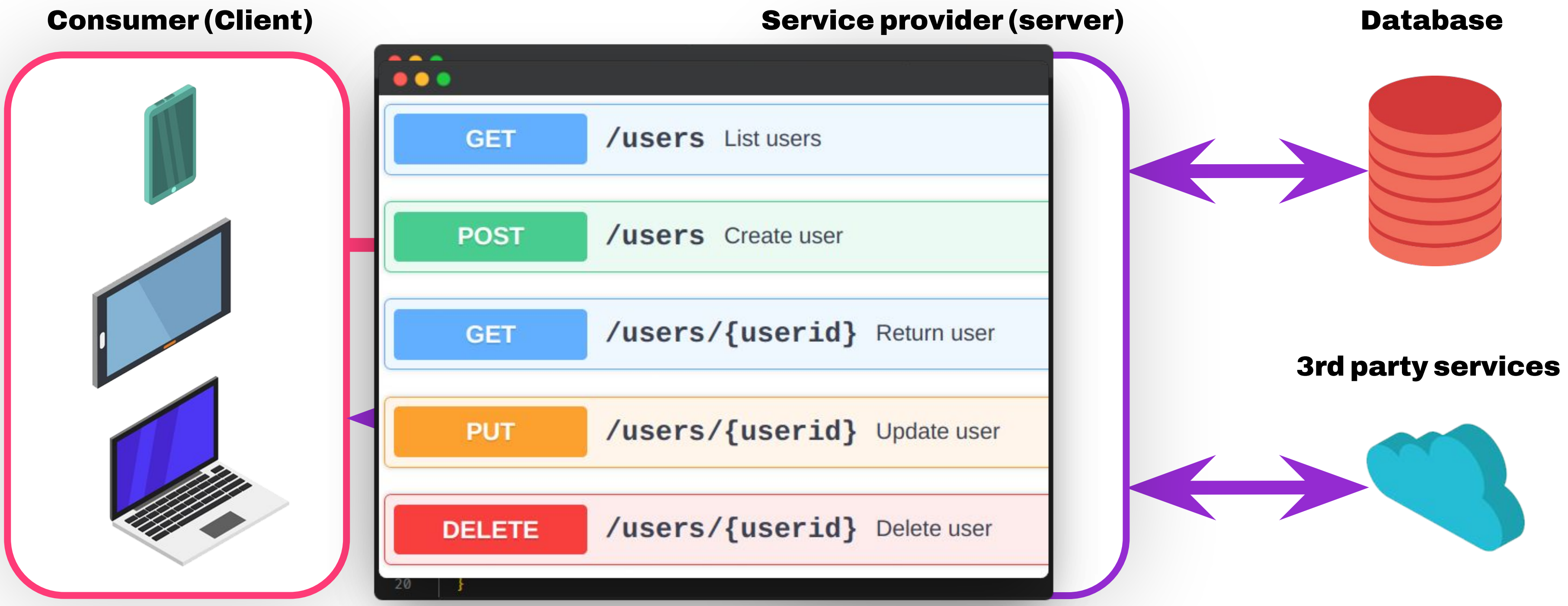
Successful Apps

Great UX

Low Cost Per User

Briefcasing RESTful data to reduce network traffic

What is REST API Architecture?



GET /users List users
POST /users Create user
GET /users/{userid} Return user
PUT /users/{userid} Update user
DELETE /users/{userid} Delete user

Top 3 reasons for RESTful middle tiers?

Keeps Data Secure

RESTful APIs act as a **protective layer** so databases don't need to expose direct connections or open ports to the internet, reducing vulnerability to attacks.

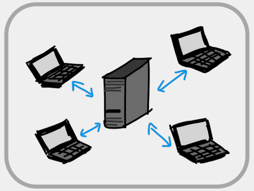
A middle-tier API **enforces** authentication, authorization, and data validation **rules** before allowing access to any backend systems

Supports Multiple Applications

RESTful services allow **multiple** front-end **applications** (web, mobile, desktop) to interact with the same data source via a unified interface

REST uses standard protocols (HTTP/HTTPS) and formats (JSON/XML), making integration across different platforms and technologies straightforward

Simplifies Scalability and Maintenance



Business logic and data access are centralized in the API layer, making it **easier** to scale, monitor, and update without touching individual client apps

What Is Briefcasing?

Definition: **Keeping a local subset of server data on the client**

Use case examples:

- Offline apps
- Mobile apps with spotty connections
- Edge devices with local logic

Why it's hard:

- How do you know what's changed?
- What if multiple clients are changing data?
- How do you manage sync without session state?
  - Date Time?
  - Logs?

The Problem in a Changing Data World

REST is stateless — which is great… **until data changes**

Common patterns:

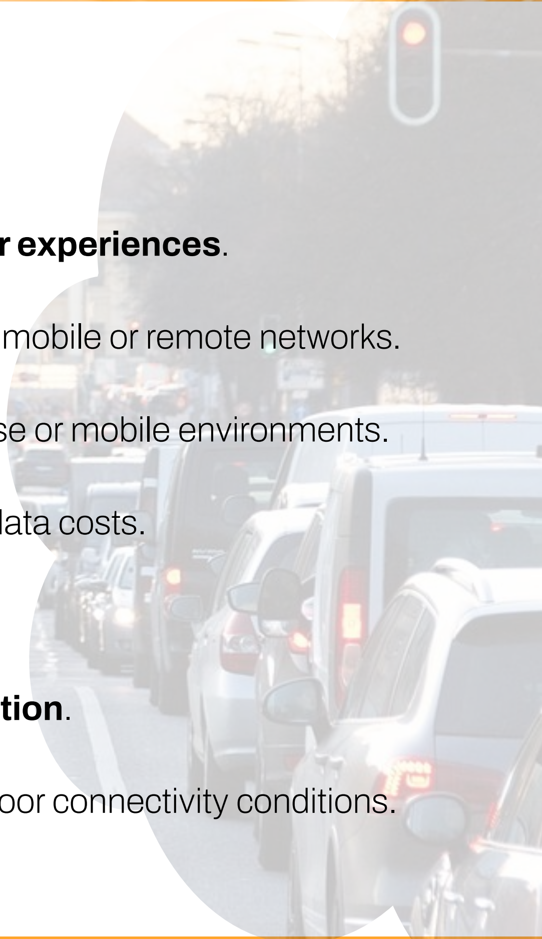
- poll-and-compare,
- full-sync every time

The pain:

- Wasted bandwidth
- Unnecessary load on servers
- Poor performance on mobile or edge devices

Metaphor

"A full-sync is like taking a full new suitcase every trip, even if you only changed your socks."



Why Reduce Network Traffic?

- **Improves Performance**:
  Less data over the network means faster response times and **smoother user experiences**.
- **Lowers Latency**:
  Reducing round trips to the server **speeds** up app interactions, especially on mobile or remote networks.
- **Conserves Bandwidth**
  Essential for users with limited or metered connections, especially in enterprise or mobile environments.
- **Reduces Infrastructure Costs**:
  Minimizes load on servers, APIs, and cloud services, lowering hosting and data costs.
- **Boosts Scalability**:
  Efficient apps can **serve more users** without increasing backend capacity.
- **Enhances Battery Life**:
  On mobile devices, less network usage can result in **lower power consumption**.
- **Improves Reliability**:
  Reducing dependencies on live connections makes apps more **resilient** in poor connectivity conditions.
- **Supports Offline Access**:
  Enables apps to function even when temporarily disconnected

Successful Apps… have a great UX

First Impressions Matter
Users often judge the **quality** of the entire app based on the first interaction

Competitive Advantage
Great UX **differentiates your solution** in crowded markets

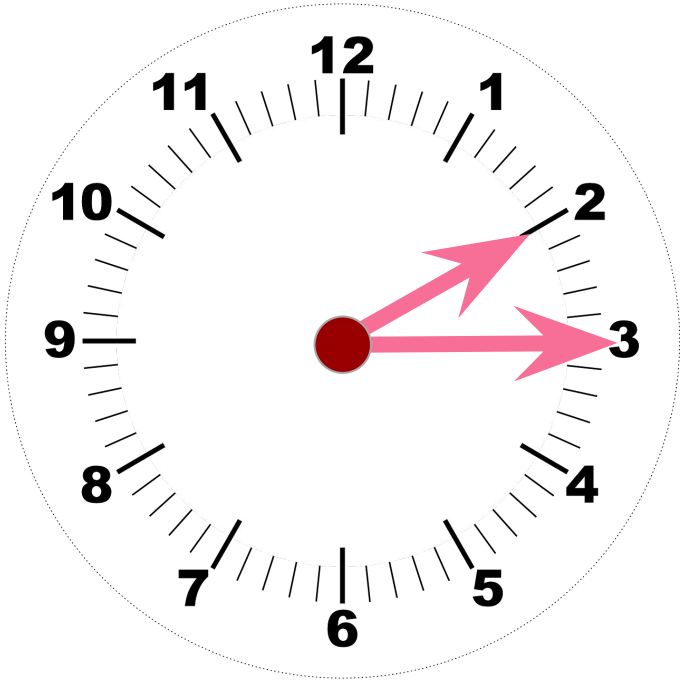
Accelerates Onboarding
Easier learning curves mean faster adoption and **time-to-value** for your customers

Customer Retention
A **smooth**, intuitive experience keeps users engaged and reduces churn

Reduces Support Costs
Well-designed interfaces lower user errors and minimize help desk queries

Enhances Product Reputation
Positive user experiences **build trust** and encourage word-of-mouth promotion.

Where are you losing time?

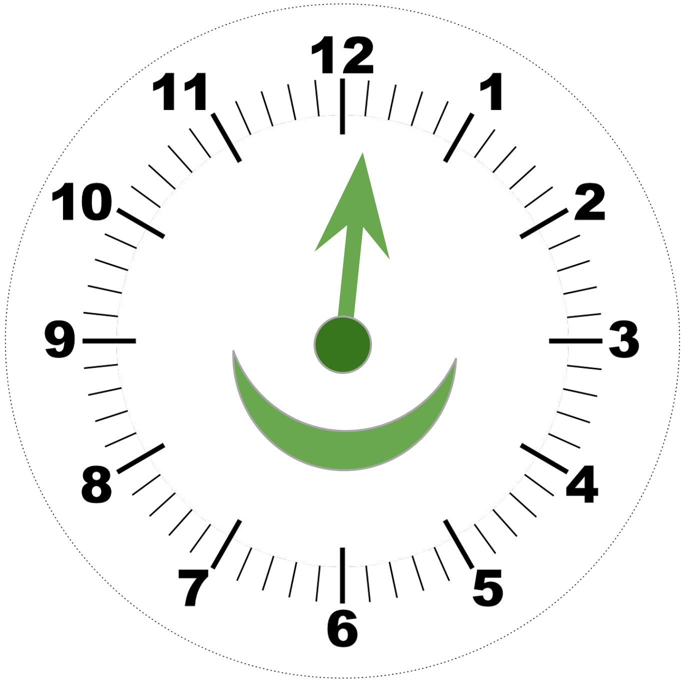


Network Latency
Delays due to variable internet speeds, especially on mobile or in low-bandwidth environments.

Authentication & Handshakes
Time-consuming processes like token validation and security checks

Complex Data Models
Apps load too much or overly complex data before showing the UI

How Local Data Caching Improves UX



Faster App Launches

Cached data allows the UI to load instantly with previously retrieved content. Users see content sooner, even while background syncing continues.

Smooth Offline Support

Enables functionality even when the network is unavailable

Seamless Syncing

Smart cache strategies (e.g., stale-while-revalidate) keep local data fresh without slowing down the interface.

Reduced Server Load / Increases Scale

Decreases API calls and bandwidth usage by serving common requests from cache.

Enter InterBase Change Views

What are **Change Views?**

- Subscription-based change tracking
- Built into the InterBase database
- Lightweight, no triggers, no logs
- Secure - User Security Managed

How it fits briefcasing perfectly

- Server tracks data changes per subscriber, per subscription
- Clients only get what has changed
- Field level granularity
- Stateless sync — no session storage required

It works using a multi-phase commit.

Creating Change Views

```
CREATE SUBSCRIPTION
  sub_stock
ON
  INVENTORY FOR ROW (INSERT, UPDATE, DELETE),
  SUPPLIERS FOR ROW (INSERT, UPDATE, DELETE)
DESCRIPTION 'Track stock and supplier changes';
```

Change View spanning multiple tables

Change View for specific field(s)

```
CREATE SUBSCRIPTION
  sub_stockname
ON
  INVENTORY(ITEM_NAME) FOR
    ROW (INSERT, UPDATE, DELETE)
DESCRIPTION 'Track stock name changes';
```

Allowing Access to Change Views

```
GRANT SUBSCRIBE ON SUBSCRIPTION sub_stock TO SYSDBA;
```

Finding Changes using SQL

```
Select * from stock;
```

```
set subscription sub_stock at
'DeviceID' active;

Select * from stock;
```

```
Commit
```

```
Rollback
```

Updating data using SQL and ChangeViews

```
Update Stock Set Price = Price * 1.1;
```

```
set subscription sub_stock at
'DeviceID' active;

Update Stock Set Price = Price * 1.1;
```

```
Commit
```

```
Rollback
```

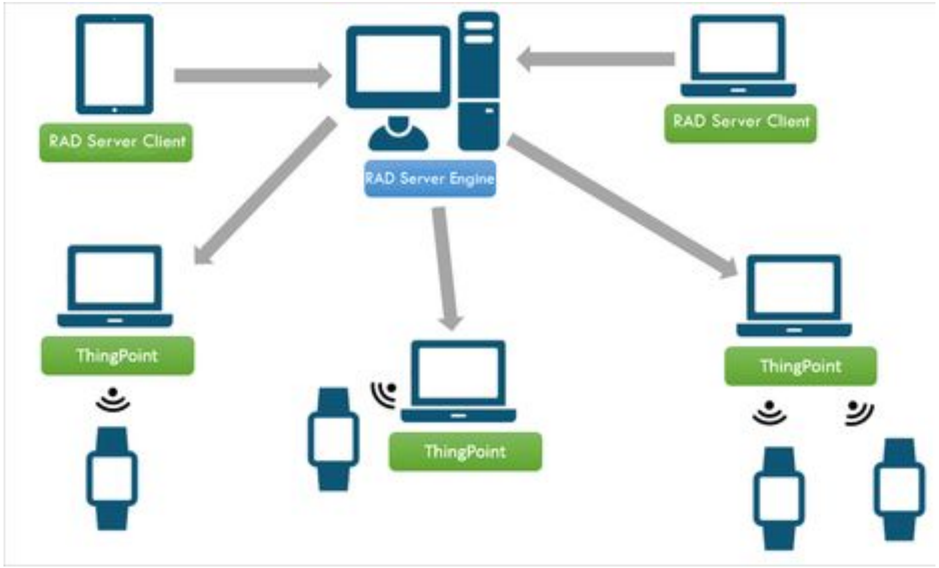
Thing Points

Enter Thing Points! Custom Applications that Expand RAD Server.

Add State Awareness to the ThingPoints and delegate ChangeView management.

Just add logic in your endpoint to check the delta via Change Views

- RAD Server acting as your REST endpoint
- ThingPoint for brief connection lifecycle on edge devices
- InterBase for backend with Change Views enabled



Demo - Change Views and RAD Server

How This Applies Beyond RAD Server

Any tech stack can benefit:

- Node.js, Python Flask, .NET Core — just connect to InterBase
- REST endpoint reads delta via Change Views and returns it

Stateless, scalable approach to sync

Especially useful in microservice or edge computing architectures

Final Takeaways

- Briefcasing avoids waste — smart sync for smart apps
- REST alone isn't enough when data is volatile
- Change Views = built-in, subscription-based change tracking
- Combine with RAD Server + ThingPoints for fast prototyping, but can be generalized
- Better UX, less network strain, easier scale, happier customers!

Q&A