Infinity Architecture

Des Adkinson

November 2019

TEMENOS

THE BANKING SOFTWARE COMPANY

Key Concepts

Key Concepts

Headless Architecture

Some clients – typically larger banks with established development teams – may want to build their own Apps and to choose their own App development tools. Infinity's Headless Architecture allows such clients to build directly upon the Experience APIs of our Distribution Services. Clients who are more interested in speed-to-market via a full solution can license (and customise/extend) the Temenos Apps which are built upon those same APIs.

Omni-channel

Omni-channel experience is a multi-channel approach to marketing, selling, and serving customers in a way that creates an integrated and cohesive customer experience no matter how or where a customer reaches out. Infinity achieves this by persisting data in the underlying distribution services, supporting 'save & resume' functionality during User Journeys.

Native Apps

Native apps – those that run directly on the OS of the device – provide the fastest and smoothest experience for the end user, can fully-leverage the native features of the hardware and are most likely to meet user expectations in terms of behaviour. Infinity delivers this capability through the lo-code/no-code capabilities of the Quantum platform; The Nitro Framework converts screen/form designs into native apps that are deployed on specific devices (iOS, Android, Mobile, Tablet…).

Microservices

Infinity employs a microservices architecture in order to deliver it's cloud-native, cloud-agnostic capability. An Infinity microservice provides a discrete piece of functionality, designed 'API-first' (see below), and has its own database optimised for the tasks it performs. As a consequence these microservices are highly maintainable, scalable and independently deployable/upgradeable.

Key Concepts

API-driven / API-first

The business strategy for Infinity is to design APIs that put the target developers' interests first and then to build the product on top of those APIs (be that a desktop User Agent, a mobile application, or a SaaS software). Building on top of APIs with developers in mind provides a dual benefit: i) it reduces development time, and ii) it lays down proven foundations for others to build upon (e.g. ecosystem/Marketplace partners, bank's development teams)

Centralised Workflow

Workflow manages business processes, which consist of tasks/activities and steps. While some of the steps of a workflow are automated, such as calling a web service, others involve human interaction. Typical examples include an applicant filling in a form, manual approval processes, document processing, or escalations.

Workflow is a common requirement across many Infinity functions, so it is important that there is a single, easy-to-configure tool that is used consistently across them all. This is provided by Fabric Workflow which employs a reduced set BPM notation to provide three main types of workflow; server-side, client-side and end-to-end.

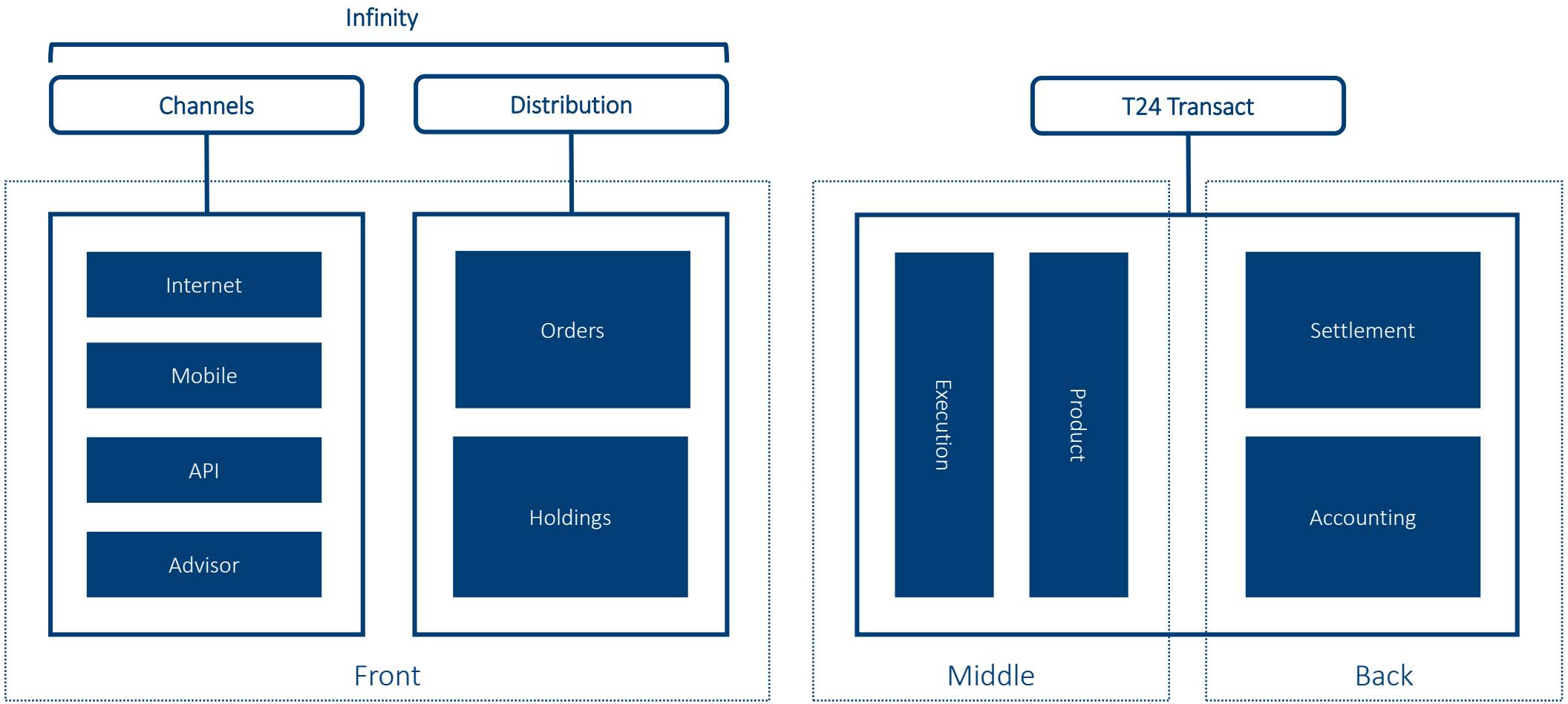
Note that Fabric Workflow is, by design, *not* intended as a full BPM solution to orchestrate human-oriented processes across the bank's full enterprise.

Single Rules Engine

Similarly, Infinity needs to apply rules in many places; origination journeys (decisioning based on scores returned from FCM systems), evaluation of most relevant engagement campaign for a customer based on AI data, etc. The Fabric Rules Engine provides this capability (and is often used in conjunction with Fabric Workflow).

Enterprise Architecture

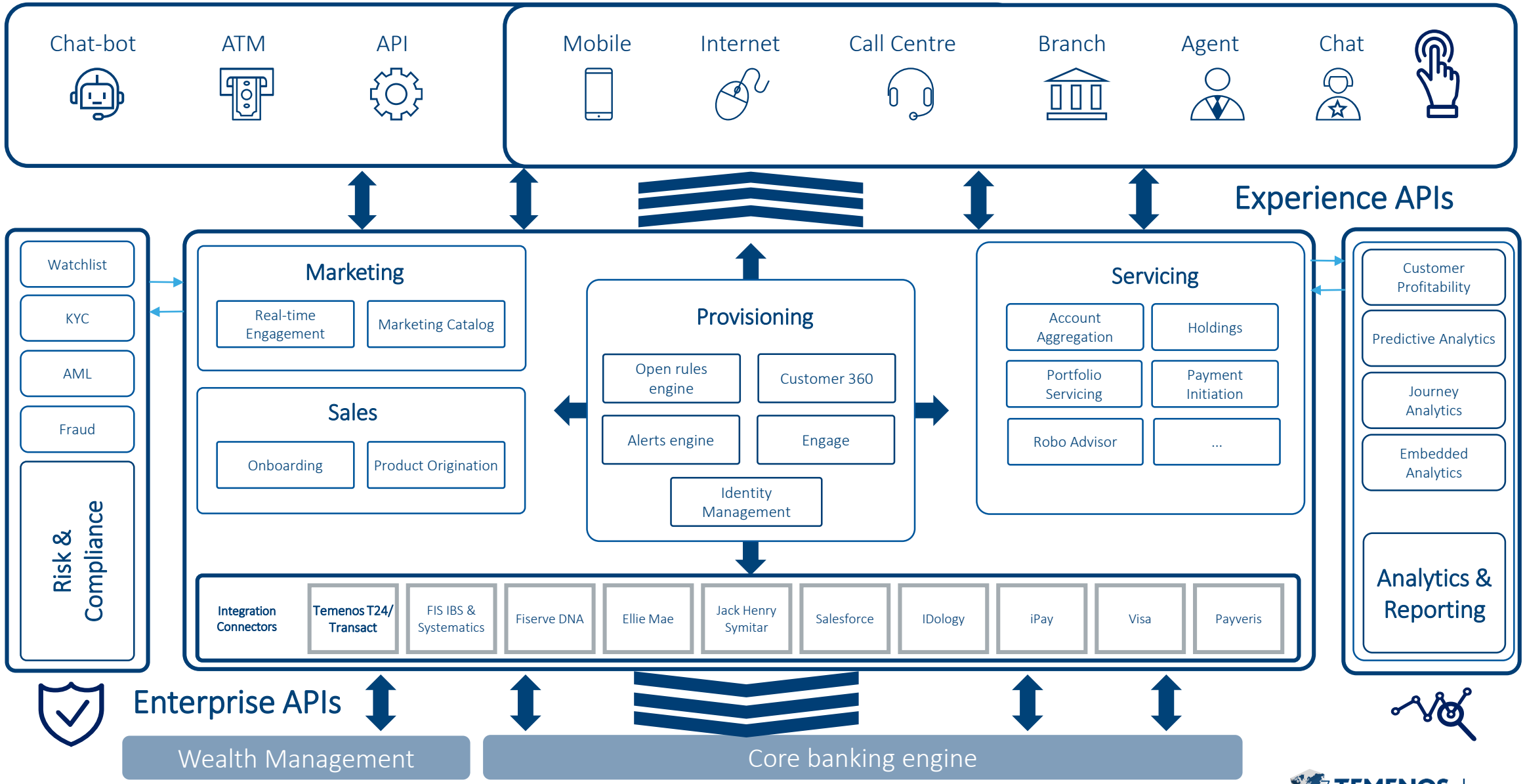
Front to Back Architecture



Front to back architecture is becoming obsolete

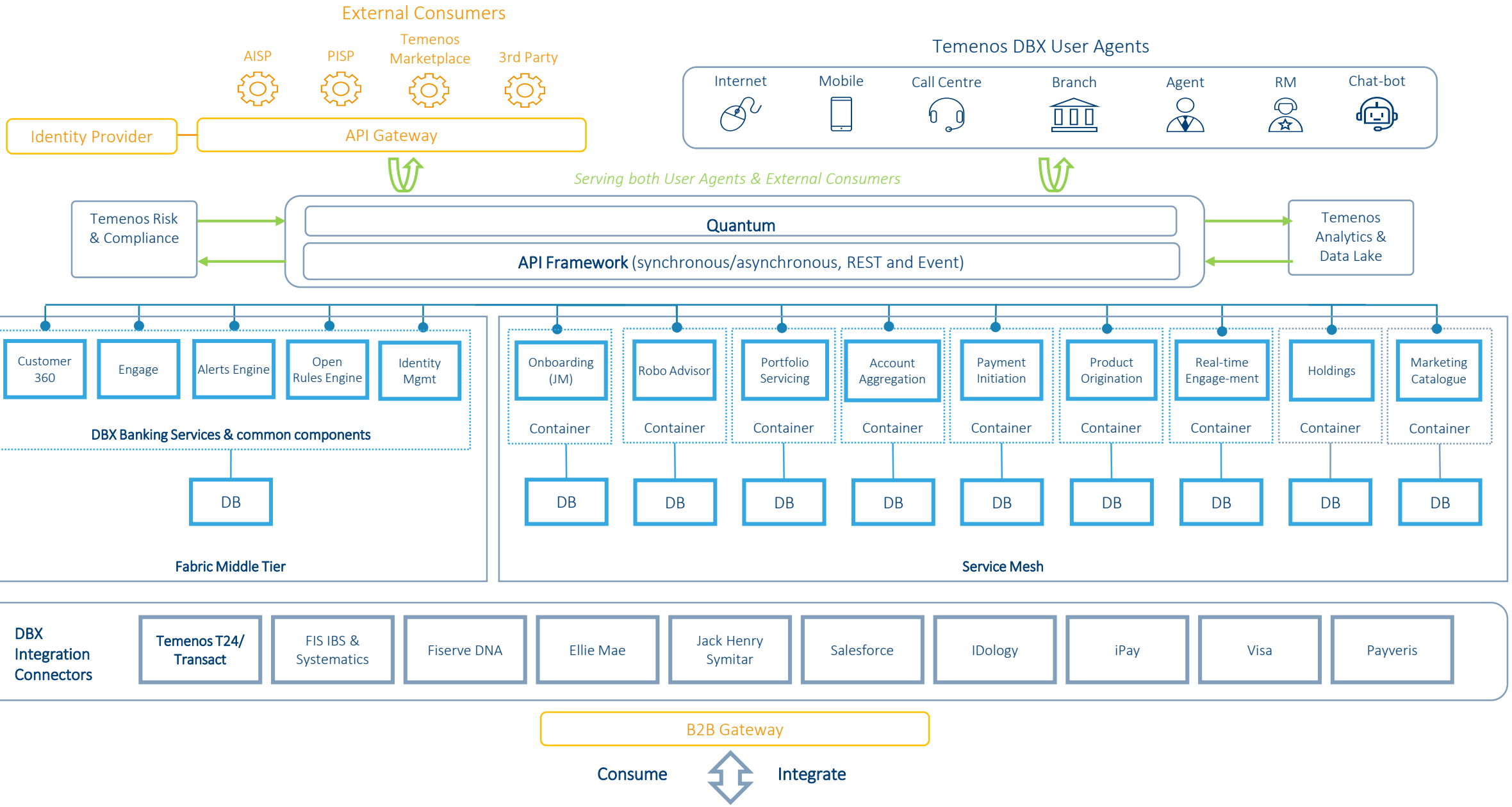
Open Banking, triggered by PSD2, moves banks to the value chain approach

Temenos Infinity reference architecture

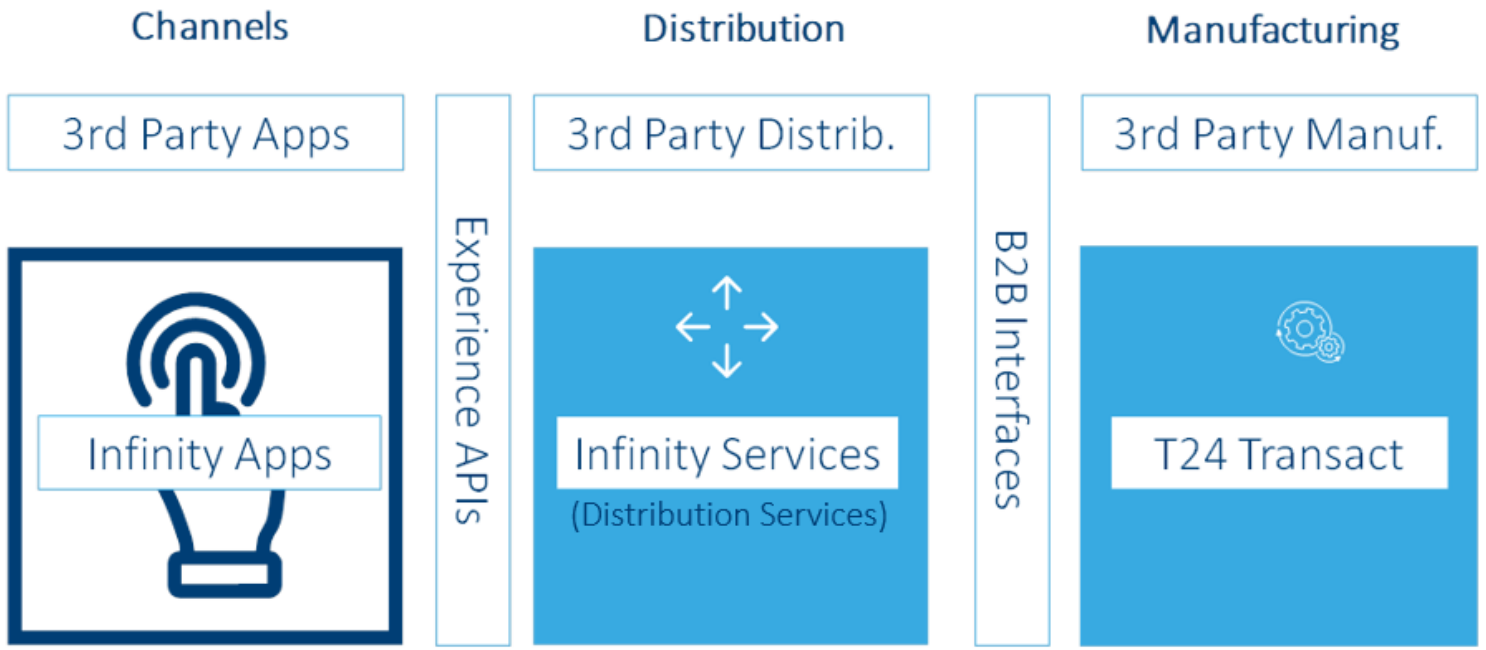


TEMENOS

Infinity Microservices Architecture



APIs and Services



APIS:

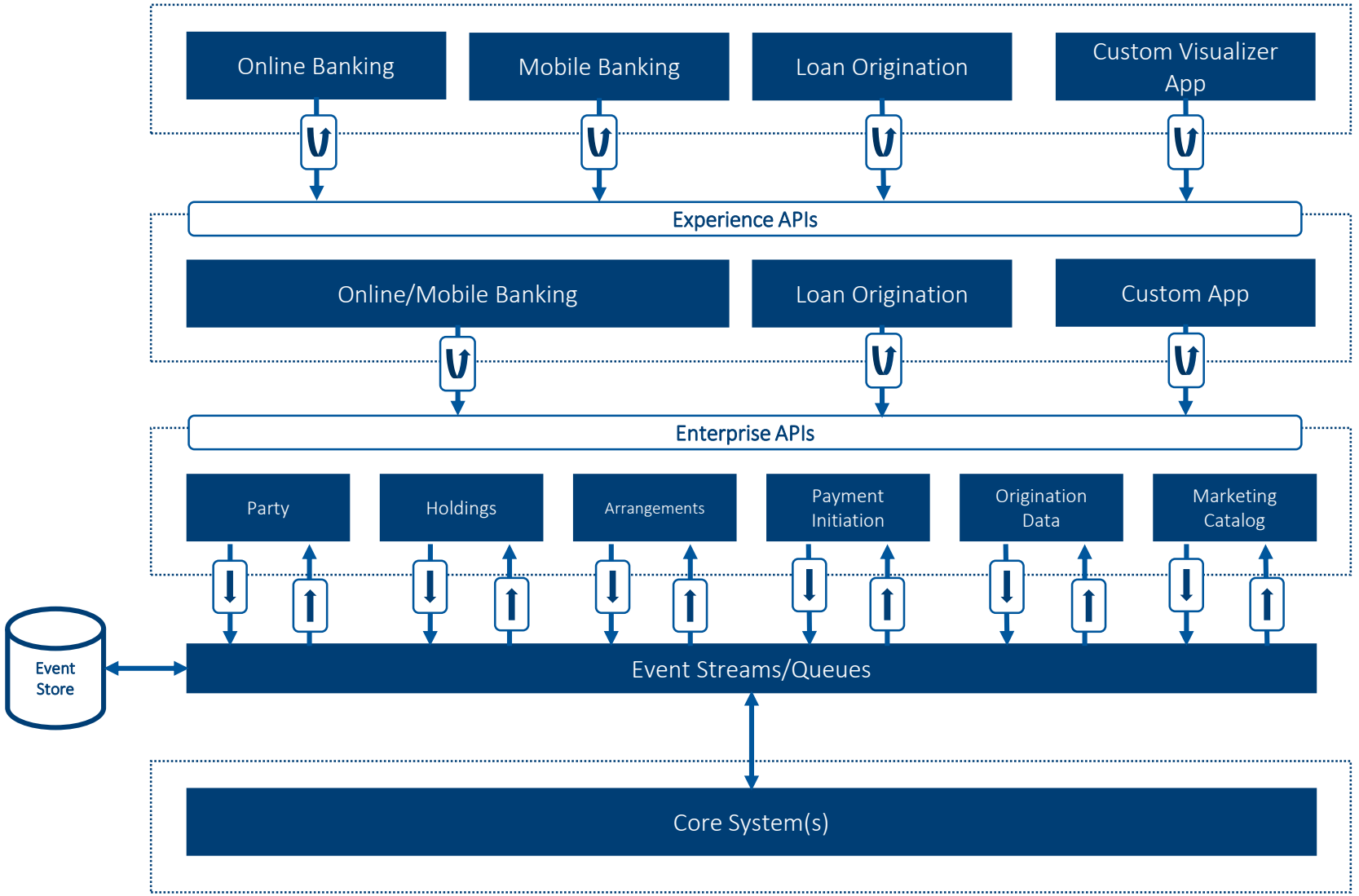
Enterprise APIs: *Expose functionality of Infinity Distribution Services, aka Lines of Business (LOBs)*

Experience APIs: *APIs 'on top of Fabric' in the form expected by application developers. These APIs hide all implementation details of T24 Transact and/or 3rd party cores (they are built on the Fabric Virtual App Data Model)*

*All Distribution Services are **headless**: A client can build an App directly on the Experience APIs without needing to use Visualizer if they wish.*

All Temenos Apps are built on the Experience APIs.

Integration at multiple layers



Experience APIs
Synchronous - Request/Response

Enterprise APIs
Synchronous - Request/Response

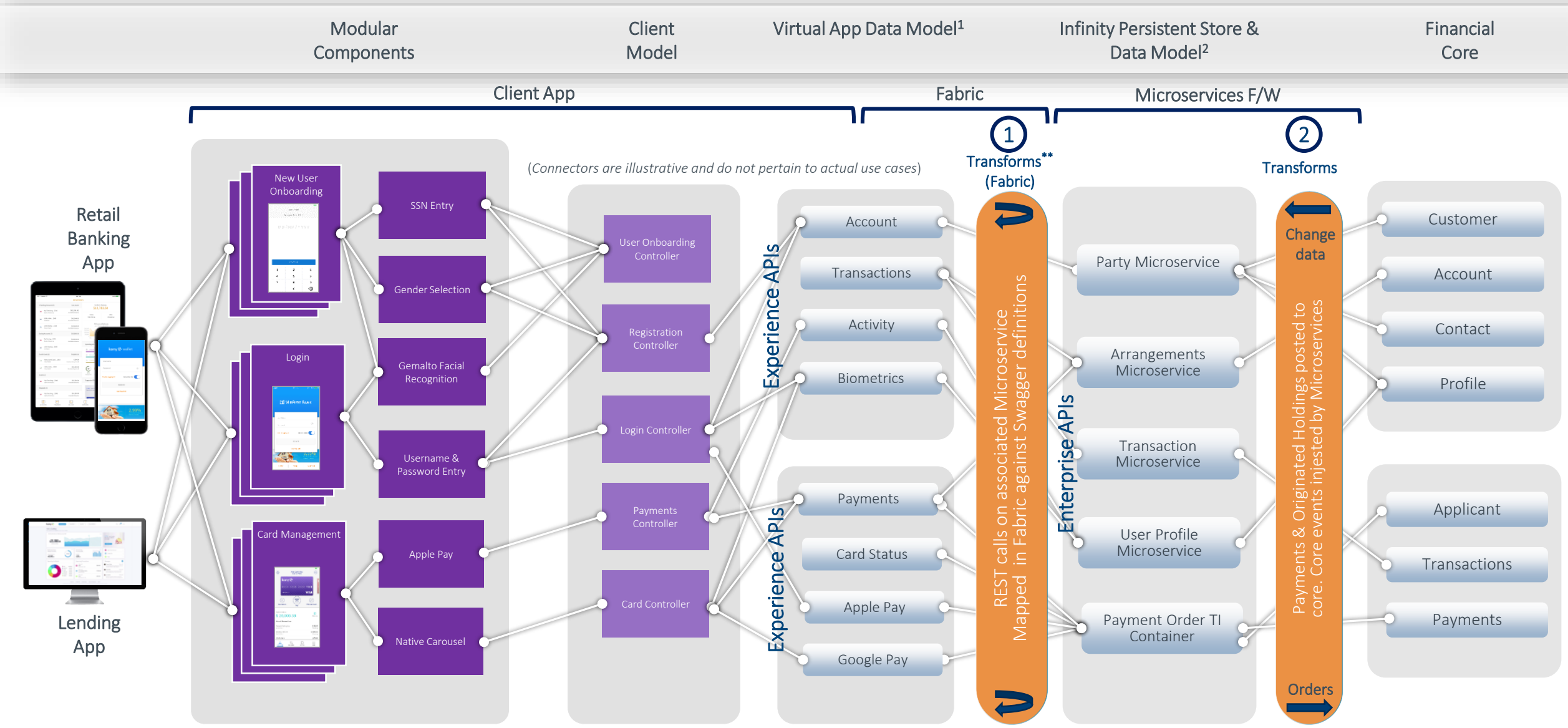
Inter-service Communication
Asynchronous – Message Based

Communication/ Data Synchronization with Core Systems
Pattern Based on Core

Terminology

- **'Orders'** is used as a broad term for something captured in the Front Office that need to be passed back to Product/core systems (there may be multiple product systems for different lines of business). Examples include:
  - Payment Order to be executed in Core or a Payment Engine
  - Origination 'Payload' that passes fulfilment details back to Core (and perhaps CRM).
  - Service Orders: Straight-through Service Requests (e.g. change of Direct Debit collection day)
- **'Change Data'** describes data that needs to be synchronised (duplicated) from Core (or other Banks) into the Front Office so that it can be rendered in the Front-end User Agents in a timely fashion and when the underlying Core system(s) are not available. Persistence is provided by Microservices and examples *include*:
  - Holdings (incl. Transactions, Balances) – a read-only service (any data changes occur as the result of an 'order' being received)
  - Party data
- **Distribution Services** (sometimes referred to as **Lines of Business – LOBs**) are Front Office capabilities that provide a marketing, sales or servicing function for product lines supported by the Financial Institution.
- A Distribution Service may be comprised of several 'API-first' **Microservices**:
  - Microservices expose **Enterprise APIs**. Fabric then maps its Virtual App Data Model onto these services and exposes **Experience APIs** in a form that an Application Developer expects (i.e. they hide the specifics of the underlying microservices and product/core systems).
  - Each Microservice has its own database that provides Infinity's data persistence
    (Quantum holds the Virtual App Data Model in *memory only*, but Infinity requires data needs to be maintained independent of Core)
- A **Headless Architecture** describes the option offered to clients that do not want to utilise the Infinity Apps. In this case the client can build their own Apps on top of the Experience APIs without using Visualizer. This option appeals to clients with large/established Dev teams and a strong preference for native development.
- The next slide shows the de-coupled architecture at a high level. Note the two Mapping (Transform) layers…

Infinity Target State*: Decoupled for Speed and Re-use



* In the Interim State, some core services may be directly linked to Experience APIs via Fabric without an intermediate Microservice; this is the tactical approach while Infinity capabilities continue to be de-coupled.

** Clients will receive a full set of Experience APIs. Where licensing a Distribution Service (e.g. Onboarding & Origination) they will also receive the corresponding Transforms. For *unlicensed* modules, the bank are responsible for the underlying mappings.

Quantum Authentication

Quantum Authentication Pattern