L02: Embedded Finance & Banking-as-a-Service

Extended Slides – BSc Digital Finance Course

Digital Finance

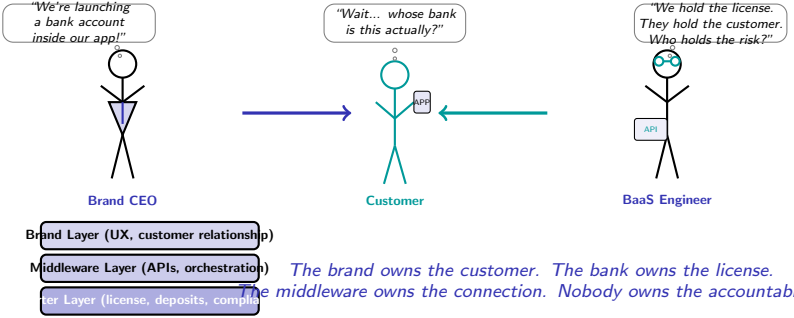
What Will You Be Able to Do After This Lecture?

By the end of this extended lecture, you will be able to:

1. **Decompose the BaaS value chain** — identify the three-layer stack (license holder, middleware, brand), compute take-rate splits, and evaluate where economic value accrues in an embedded finance arrangement
2. **Model BNPL credit risk formally** — derive the probability of default for installment lending using logistic regression, compute effective APR from late fees, and compare BNPL loss rates against traditional credit card portfolios
3. **Quantify embedded insurance economics** — calculate expected loss ratios with adverse selection adjustments, model conversion lift from contextual distribution, and assess when embedded convenience erodes informed consent
4. **Analyse regulatory accountability in multi-party BaaS** — map regulatory perimeters across three-entity structures, evaluate concentration risk using HHI, and trace the Synapse collapse to identify systemic failure points
5. **Formalize platform switching costs and network effects** — model API migration costs as a sum of rewrite, testing, and downtime components, and analyse two-sided market equilibrium in BaaS platforms
6. **Implement BaaS computations in Python** — write correct, concise code for API wrappers, BNPL APR calculators, credit scoring models, regulatory decision trees, rate limiters, and Nash bargaining optimizers

---

Six objectives span architecture (1), credit risk (2), insurance (3), regulation (4), platform economics (5), and implementation (6).


"We're launching a bank account inside our app!"
"Wait... whose bank is this actually?"
"We hold the license. They hold the customer. Who holds the risk?"
API
API
Brand CEO
Customer
BaaS Engineer
Brand Layer (UX, customer relationship)
Middleware Layer (APIs, orchestration)
ter Layer (license, deposits, compli
The brand owns the customer. The bank owns the license.
The middleware owns the connection. Nobody owns the accountability.

How Do You Split Revenue When Three Companies Deliver One Bank Account?

BaaS Unit Economics – Three-Party Decomposition

Customer lifetime value in a BaaS arrangement:

$$\text{LTV} = \sum_{t=1}^{T} \frac{R_t - C_t}{(1+r)^t}$$

Revenue splits across the stack:

$$R_t = R_t^{\text{brand}} + R_t^{\text{middleware}} + R_t^{\text{charter}}$$

Take-rate decomposition:

$$\tau_{\text{brand}} = \alpha \cdot R_t \quad \text{(40–60\%, owns customer acquisition)}$$
$$\tau_{\text{middleware}} = \beta \cdot R_t \quad \text{(15–25\%, API orchestration)}$$
$$\tau_{\text{charter}} = (1-\alpha-\beta) \cdot R_t \quad \text{(20–35\%, license + compliance)}$$

Customer acquisition cost allocation:

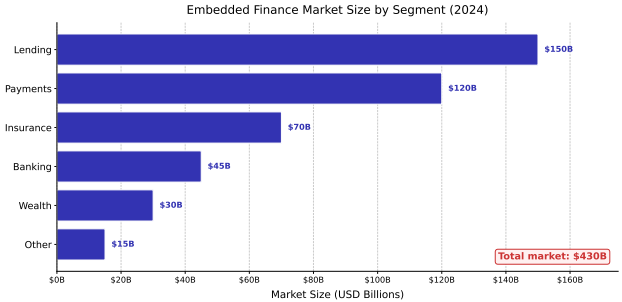
$$\text{CAC}_{\text{total}} = \text{CAC}_{\text{brand}} + \text{CAC}_{\text{middleware}} + \text{CAC}_{\text{charter}}$$

Typically: brand bears 80%+ of CAC (marketing, onboarding UX), charter bears compliance KYC cost, middleware bears integration cost. The entity that bears the CAC demands the largest take-rate.

Viability constraint: $\text{LTV}_{\text{layer}} > \text{CAC}_{\text{layer}}$ must hold for *each* layer independently. If the charter bank's share does not cover its compliance costs, the entire stack collapses.

BaaS viability requires each layer to independently cover its costs. The brand bears most CAC; the charter bears most compliance cost. Misalignment breaks the stack.

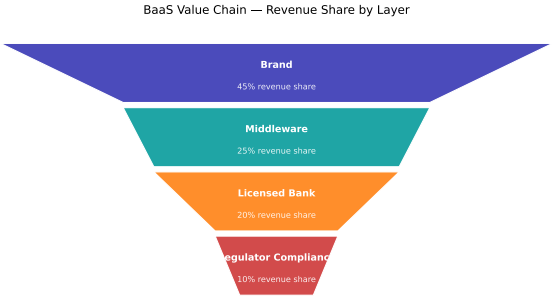
How Large Is the Embedded Finance Opportunity by Vertical?



- Embedded payments leads ($80B+ revenue, most mature segment)
- Embedded lending second ($45B, driven by BNPL and POS credit)
- Embedded insurance fastest-growing (40%+ CAGR from low base)
- Embedded banking/accounts still nascent but accelerating
- Total embedded finance: $180B+ addressable market by 2026
- **Key insight:** payments are commoditized; lending and insurance are where margin remains

Payments dominate embedded finance volume, but lending and insurance dominate margin. The market follows the classic commoditization curve: payments first, value-add second.

Where Does Value Leak in the BaaS Stack?



- Funnel shows value attrition from gross revenue to net margin across the three-layer stack
- Brand layer captures most gross revenue but spends heavily on CAC
- Middleware takes a thin but stable margin (API fees per call)
- Charter bank bears compliance cost that scales with regulatory intensity
- Net margin concentration: middleware often earns the highest net margin percentage despite lowest gross share
- **Key insight:** the infrastructure layer (middleware) is the most capital-efficient position in the BaaS stack

Middleware earns the highest net margin percentage in the BaaS stack. Like cloud infrastructure, the pick-and-shovel position outperforms the gold miners.

What Does a Real BaaS API Integration Look Like?

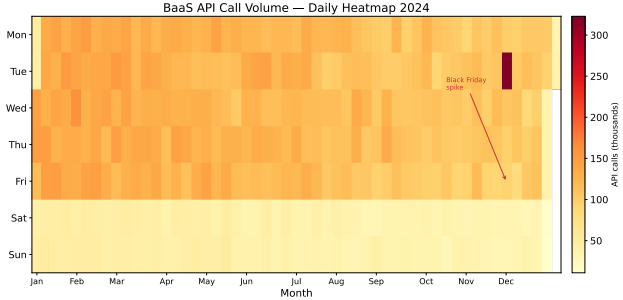
```python
import requests

class BaaSClient:
    """Minimal BaaS API wrapper."""
    BASE = "https://api.baas-provider.io/v1"

    def __init__(self, api_key):
        self.headers = {
            "Authorization": f"Bearer {api_key}",
            "Content-Type": "application/json"}

    def create_account(self, customer_id,
                       product="checking"):
        resp = requests.post(
            f"{self.BASE}/accounts",
            headers=self.headers,
            json={"customer_id": customer_id,
                  "product_type": product,
                  "currency": "USD"})
        resp.raise_for_status()
        return resp.json() # {account_id, iban}
```

Anatomy of a BaaS call

- **Authentication:** Bearer token (API key) identifies the brand; the BaaS provider maps it to the charter bank
- **Account creation:** One POST creates a real bank account — KYC, ledger entry, and IBAN generation happen server-side
- **Abstraction:** The brand never touches the core banking system directly; the middleware translates REST into core banking operations
- **Latency:** Account creation takes 200–800ms (KYC pre-approved) or 24–48h (KYC pending)
- **Risk:** If the BaaS provider goes down, every brand on the platform loses account functionality simultaneously

One API call creates a real bank account. The simplicity hides a complex chain: brand → middleware → core banking → regulator. Abstraction enables speed but concentrates risk.

When Do Consumers Actually Use Embedded Financial Services?



- Heatmap showing API call volume by hour-of-day and day-of-week
- Peak: weekday evenings (18:00–21:00) — consumers check balances and make purchases after work
- Secondary peak: Monday mornings (salary check, bill payments)
- Weekend pattern: lower volume but longer session duration
- Holiday spikes: BNPL calls surge 3–5x during Black Friday and Christmas
- **Key insight:** embedded finance usage follows e-commerce patterns, not banking patterns — validating the "finance at point of need" thesis

API call patterns follow e-commerce, not banking hours. Embedded finance is consumed when consumers shop, not when banks are open. Timing proves the thesis.

What Are the Three Architectural Layers That Make BaaS Possible?

The BaaS stack. Embedded finance requires three distinct layers, each with different capabilities, licenses, and risk profiles.

Layer 1: Charter Bank (License Holder)

- Holds the banking license (OCC, ECB, FCA)
- Provides deposit insurance (FDIC, FSCS, DGS)
- Bears regulatory capital requirements (Basel III)
- Examples: Sutton Bank, Hatch Bank, Solaris, Green Dot
- Revenue: compliance fees + deposit float income

Layer 2: Middleware (BaaS Platform)

- Translates banking operations into REST APIs
- Manages KYC/AML orchestration across providers
- Provides ledger, card issuance, payment processing
- Examples: Unit, Treasury Prime, Railsr, Weavr
- Revenue: per-API-call fees + monthly platform fees

Layer 3: Brand (Distribution)

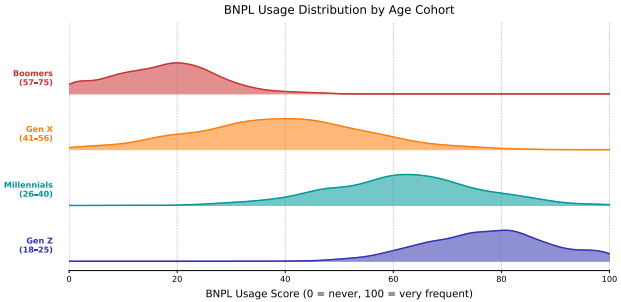
- Owns the customer relationship and UX
- Handles marketing, onboarding, support
- Bears no banking license — regulated as agent
- Examples: Cash App, Shopify Balance, Uber
- Revenue: interchange share + cross-sell conversion

The accountability gap:

- Customer sees the brand; regulator sees the bank
- Middleware is often invisible to both
- When Synapse (middleware) collapsed in 2024, 100K+ customer accounts were frozen
- No single entity had a complete view of the customer's position
- **Lesson:** modularity enables speed but fragments accountability

Three layers, three incentive structures, three failure modes. Modularity enables rapid deployment but fragments the accountability that regulation assumes is unified.

Which Age Groups Adopted BNPL – and Which Are Getting Burned?



- Ridgeline plot showing BNPL usage distribution by age cohort
- Gen Z (18–25): highest adoption rate (45%+), smallest average order value
- Millennials (26–40): highest total BNPL volume, largest average order
- Gen X (41–55): growing fastest from low base, using for larger purchases
- Boomers (56+): minimal adoption, sceptical of deferred payments
- **Key concern:** Gen Z has the highest adoption AND the highest late-payment rate (12–15% vs 6% for Millennials)

BNPL adoption skews young and financially vulnerable. The cohort with the highest usage also has the highest default rate — a pattern regulators are watching closely.

What Is the True Cost of "Pay in 4" When You Miss a Payment?

```python
import numpy as np

def bnpl_effective_apr(purchase, n_installments,
                       late_fee, n_late,
                       period_weeks=2):
    """Compute effective APR for BNPL.
    purchase: total purchase amount
    n_installments: number of payments
    late_fee: fee per missed payment
    n_late: how many payments are late
    period_weeks: weeks between payments"""
    installment = purchase / n_installments
    total_paid = purchase + late_fee * n_late
    total_cost = total_paid - purchase
    duration_years = (n_installments *
                      period_weeks) / 52
    if total_cost <= 0:
        return 0.0 # No cost if on-time
    apr = (total_cost / purchase) / duration_years
    return round(apr * 100, 1) # percent

# Example: $200 purchase, pay-in-4, 1 late
print(bnpl_effective_apr(200, 4, 7, 1)) # 45.5%
```

The hidden cost of "free"

- **On-time:** 0% APR — genuinely free for the consumer (merchant pays 3–6% MDR)
- **One late payment:** Effective APR jumps to 45% for a **$**200 purchase with a **$**7 late fee
- **Two late:** APR exceeds 90% — worse than most credit cards
- **Comparison:** Average credit card APR is 22–24%. BNPL late fees can exceed this on small purchases
- **Regulatory gap:** BNPL is not classified as credit in many jurisdictions, so APR disclosure is not required

BNPL is free when you pay on time. Miss one payment on a $200 purchase and the effective APR hits 45% — double the average credit card rate.

Disclosure is not required.

How Do You Model Default Probability for Someone with No Credit File?

BNPL Credit Risk — Logistic PD Model for Installment Lending

Probability of default:

$$P(\text{default}) = \sigma(\beta_0 + \beta_1 \cdot \text{income} + \beta_2 \cdot \text{age} + \beta_3 \cdot \text{order\_value} + \beta_4 \cdot \text{n\_active\_plans} + \beta_5 \cdot \text{past\_late})$$

where $\sigma(z) = \frac{1}{1+e^{-z}}$ is the logistic function.

Key differences from traditional credit scoring:

- **No credit bureau data:** 40%+ of BNPL users are "thin-file" — insufficient credit history for traditional scoring
- **Behavioural features:** Time-of-day of purchase, device type, browsing duration replace bureau features
- **Stacking risk:** β_4 (number of active plans) is the strongest predictor — users with 3+ simultaneous BNPL plans default at 5× the base rate

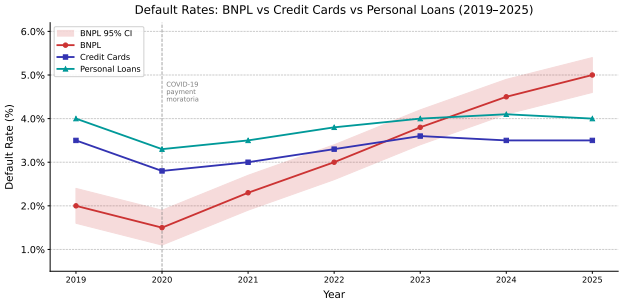
Expected loss for the BNPL portfolio:

$$\text{EL} = \text{PD} \times \text{EAD} \times \text{LGD}$$

- PD = 4–8% (vs 2–3% for credit cards)
- EAD = average remaining balance ($80–150)
- LGD = 60–80% (unsecured, small amounts not worth collecting)
- **Result:** EL = 2–5% of portfolio — merchants must subsidize this through MDR

BNPL default rates (4–8%) exceed credit cards (2–3%) because users are younger, thinner-file, and can stack multiple plans invisibly. Merchants bear the loss through MDR.

Are BNPL Default Rates Converging Toward or Diverging from Credit Cards?



- Line chart: default rates over time for BNPL, credit cards, and personal loans
- BNPL defaults rose from 3% (2020) to 7%+ (2024) as user base broadened
- Credit card defaults remained stable at 2–3%
- Divergence accelerated after 2022 as macro conditions tightened
- Australia (first to regulate BNPL): defaults stabilized after mandatory credit checks introduced
- **Key insight:** BNPL defaults rise as the product moves from early adopters (creditworthy) to mass market (thin-file)

BNPL defaults diverged from credit cards after 2022. The early "zero default" narrative was selection bias — creditworthy early adopters, not product superiority.

Can You Build a Credit Scorer Without a Credit Bureau?

```python
import numpy as np

def bnpl_scorer(income, age, order_value,
                active_plans, past_late):
    """Logistic regression BNPL scorer.
    Returns: (pd, decision, limit)."""
    # Trained coefficients (illustrative)
    b = np.array([-2.5, # intercept
                  -0.003, # income (higher=safer)
                   0.02, # age (younger=riskier)
                   0.008, # order value
                   0.6,  # active plans (stacking)
                   0.9]) # past late payments
    x = np.array([1, income, age, order_value,
                  active_plans, past_late])
    z = b @ x
    pd = 1 / (1 + np.exp(-z))
    # Decision thresholds
    if pd < 0.05:
        return pd, "APPROVE", order_value
    elif pd < 0.10:
        return pd, "APPROVE", min(order_value, 200)
    else:
        return pd, "DECLINE", 0

# Example: 25yo, $35K income, $150 order, 2 plans
print(bnpl_scorer(35000, 25, 150, 2, 0))
```

Scoring without bureau data

- **Feature set:** Income (self-declared or bank-verified), age, order value, active plan count, past late payments on this platform
- **Stacking penalty:** $\beta_4 = 0.6$ is the largest coefficient — three active plans nearly double the PD
- **Tiered approval:** Low-risk gets full amount; medium-risk gets a reduced limit; high-risk is declined
- **Real-time:** Decision in $<$200ms at checkout — slower means abandoned cart
- **Feedback loop:** Model retrains weekly on payment outcomes — performance degrades without fresh data

BNPL scoring replaces credit bureau data with behavioural features. The stacking coefficient dominates — multiple active plans are the strongest default predictor.

Is BNPL a New Product or Just a Credit Card in Disguise?

What BNPL does differently:

- **Distribution:** Embedded at checkout vs separate application. Conversion lift 20–30% because friction is near zero
- **Psychology:** "4 payments of $50" feels smaller than "$200 on your card." Mental accounting makes spending feel free
- **Targeting:** Reaches thin-file consumers that credit cards reject. 40% of BNPL users have no credit card
- **Merchant alignment:** Merchant pays the MDR (3–6%), consumer pays nothing if on time. Aligned incentives at point of sale

What BNPL does NOT differently:

- Creates a liability for the consumer (same as credit)
- Charges penalty fees for late payment (same as credit)
- Relies on revolving users for profitability (same as credit)
- Concentrates risk in unsecured consumer lending (same as credit)

The regulatory arbitrage:

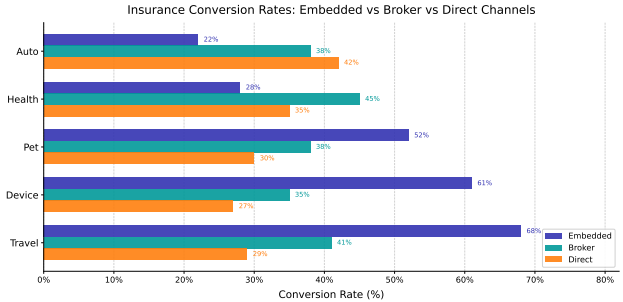
Requirement	Credit Card	BNPL
Credit check	Mandatory	Often none
APR disclosure	Required	Not required
Cooling-off period	14 days (EU)	None
Credit bureau reporting	Yes	Partial
Capital requirements	Basel III	None
Complaint handling	FCA/CFPB	Self-regulated

Convergence trajectory:

- UK: FCA brought BNPL under consumer credit regulation (2024)
- EU: Consumer Credit Directive revision includes BNPL
- US: CFPB classified BNPL as credit (2023 interpretive rule)
- **Endgame:** As regulation converges, BNPL's cost advantage erodes, leaving UX as the only differentiator

BNPL's advantage is distribution (embedded) and psychology (installment framing), not economics. As regulation converges with credit cards, only the UX advantage remains.

Why Does Insurance Convert 5x Better When Sold Inside Another Product?



- Bar chart comparing conversion rates: traditional broker, comparison site, direct online, embedded at checkout
- Traditional broker: 2–4% conversion (cold outreach)
- Comparison site: 8–12% (intent-driven)
- Direct online: 5–8% (brand-driven)
- Embedded at checkout: 15–25% (contextual, pre-filled)
- **Why:** embedded insurance benefits from purchase intent already established — the consumer is buying a flight, phone, or car; insurance is a one-click add-on
- **Concern:** high conversion may reflect convenience bias, not informed choice

Embedded insurance converts 5x better than brokers because it rides purchase intent. But high conversion driven by convenience raises the question: is it informed consent?

How Do You Price Insurance When the Buyer Never Reads the Policy?

Embedded Insurance Pricing with Adverse Selection

Expected loss per policy:

$$E[L] = p \cdot S \cdot (1 + \theta_{\text{adverse}})$$

where p is the base probability of a claim, S is the average severity (payout), and θ_{adverse} is the adverse selection loading factor.

Premium calculation:

$$\pi = E[L] \cdot (1 + \lambda_{\text{expense}}) \cdot (1 + \lambda_{\text{profit}})$$

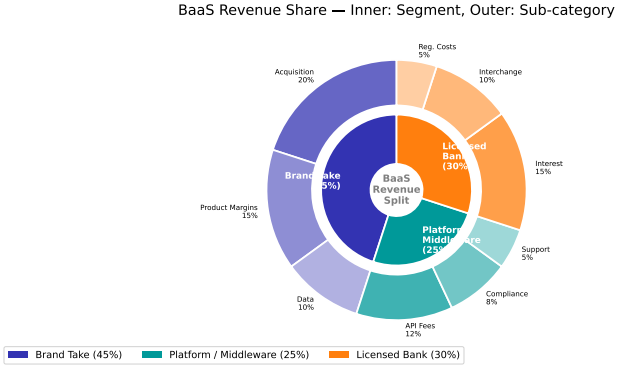
- λ_{expense} = expense ratio (20–30% for embedded, vs 35–45% for traditional — no broker commission)
- λ_{profit} = target profit margin (10–15%)
- Combined ratio: CR = $\frac{\text{Claims+Expenses}}{\text{Premiums}}$
- Profitable when CR $< 100\%$

Adverse selection in embedded:

- Traditional: $\theta_{\text{adverse}} = 0.15$–$0.25$ (self-selection by risky buyers)
- Embedded: $\theta_{\text{adverse}} = 0.05$–$0.10$ (everyone offered at checkout, less self-selection)
- **Paradox:** embedded distribution *reduces* adverse selection because the purchase trigger is the product, not the risk awareness
- Loss ratio improvement: 5–10 points better than traditional channels
- But: low engagement means low claim rates (consumers forget they have coverage), artificially depressing loss ratios

Embedded insurance reduces adverse selection (everyone is offered) and expense ratios (no broker). The loss ratio looks better — but low claims may reflect ignorance, not satisfaction.

Who Captures the Most Value in the Embedded Finance Ecosystem?



- Donut chart showing revenue share across embedded finance participants
- Charter banks: 20–30% (license rent + deposit float)
- BaaS middleware: 15–25% (API orchestration fees)
- Brands: 35–45% (interchange share + cross-sell)
- Card networks: 5–10% (scheme fees, non-negotiable)
- Insurance underwriters: 5–10% (where applicable)
- **Key insight:** brands capture the largest share but face the highest CAC — net margin favours the middleware layer

Brands capture the largest gross share but middleware captures the best net margin. The invisible infrastructure layer is the most profitable position in embedded finance.

Can Your Shopping App Become Your Financial Advisor?

Embedded wealth management: distributing investment products inside non-financial apps, from spare-change rounding to automated portfolio allocation.

Distribution models:

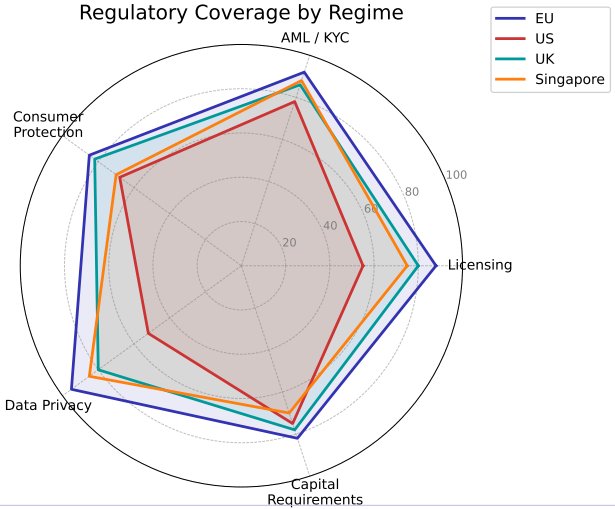
- **Round-up investing:** Purchase of $4.70 rounds to $5.00, $0.30 invested automatically. Acorns model, now embedded via API
- **Cashback-to-invest:** Merchant cashback (1–3%) auto-invested into ETFs. Combines loyalty with wealth building
- **Salary-split:** Employer payroll integration directs a percentage to investment account before spending
- **Contextual offers:** "You saved $30 on this purchase. Invest it?" shown at checkout
- **Robo-advisory APIs:** Wealthfront, Betterment offer B2B APIs for other apps to embed portfolio management

Regulatory challenges:

- **Suitability:** MiFID II (EU) and Reg BI (US) require investment suitability assessment — can an embedded widget perform this adequately?
- **Disclosure:** Risk warnings must be prominent, not buried in a checkout flow
- **Gamification concern:** Round-ups make investing feel like spending, which may distort risk perception
- **Fragmentation:** A consumer with round-ups in 3 apps has no unified view of total investment exposure
- **Fee transparency:** Embedded fees (0.25–1.0% AUM) often hidden in the product UX

Embedded wealth management lowers the barrier to investing but risks trivializing it. When investing feels like spending, consumers may underestimate the risk they are taking.

Which Regulatory Regime Actually Covers the Full BaaS Stack?



- Radar chart comparing regulatory coverage across dimensions: licensing, consumer protection, AML/KYC, capital requirements, data privacy, operational resilience
- US: strong on licensing and AML but weak on consumer protection for non-bank partners
- EU: strongest on data privacy (GDPR) and consumer protection but slower on licensing innovation
- UK: most balanced — FCA's "regulatory perimeter" approach covers all BaaS layers
- Singapore: lightest-touch licensing but strongest on operational resilience
- **Key insight:** no single regime covers all six dimensions adequately — regulatory gaps exist everywhere

No regulatory regime covers the full BaaS stack. The US has licensing gaps, the EU has speed gaps, the UK is most balanced but still evolving. Gaps create arbitrage opportunities.

What Happens to the Financial System When Every Brand Uses the Same BaaS Provider?

Concentration Risk in BaaS — Herfindahl-Hirschman Index

Market concentration:

$$\text{HHI} = \sum_{i=1}^{N} s_i^2 \times 10{,}000$$

where s_i is the market share of BaaS provider i (as a decimal).

BaaS market concentration (2024):

Layer	HHI	Interpretation
Charter banks (US)	2,800	Highly concentrated
Middleware (US)	1,900	Moderately concentrated
Card processing	4,200	Very concentrated
EU BaaS middleware	1,200	Competitive

DOJ thresholds: $<$1,500 = competitive; 1,500–2,500 = moderate; $>$2,500 = concentrated.

Systemic risk implications:

- If one charter bank (e.g., Sutton Bank) serves 50+ brands via BaaS, its failure freezes millions of accounts simultaneously
- **Cascading failure:** Charter bank → middleware → all brands on that middleware → all their customers
- Traditional banking: one bank failure = one bank's customers affected
- BaaS: one bank failure = every brand on the stack affected
- **Concentration paradox:** BaaS was supposed to increase competition but concentrated the infrastructure layer

BaaS HHI of 2,800 for US charter banks means high concentration. One charter bank failure cascades through every brand on the stack — systemic risk by architecture.

How Does a Regulator Decide Who Needs a License in a BaaS Arrangement?

```python
def regulatory_perimeter(entity):
    """Decision tree for BaaS licensing.
    entity: dict with role, activities, etc.
    Returns: (license_required, type, reason)"""
    role = entity.get("role")
    holds_funds = entity.get("holds_funds", False)
    issues_credit = entity.get("issues_credit", False)
    takes_deposits = entity.get("takes_deposits", False)

    if takes_deposits:
        return True, "BANK_CHARTER", \
               "Deposit-taking requires full license"
    if issues_credit:
        return True, "LENDING_LICENSE", \
               "Credit issuance requires authorization"
    if holds_funds:
        return True, "EMI_LICENSE", \
               "Holding client funds requires EMI"
    if role == "middleware":
        return True, "REGISTERED_AGENT", \
               "API intermediary needs registration"
    if role == "brand":
        return False, "APPOINTED_REP", \
              "Brand operates under bank's license"
    return False, "UNREGULATED", "Outside perimeter"
```

The perimeter problem

- **Deposit-taking:** Always requires a full bank charter — no exceptions
- **Credit issuance:** Requires a lending license, but "true lender" doctrine is contested (who is the real lender in BaaS?)
- **Holding funds:** EMI license in EU; money transmitter license in US (state-by-state)
- **Middleware:** Grey zone — some jurisdictions require registration, others do not
- **Brand:** Usually operates as an "appointed representative" under the charter bank's license umbrella
- **Gap:** When the brand markets the product but the bank issues it, consumer complaints fall between two regulators

The regulatory perimeter determines who needs a license. In BaaS, the brand often falls outside it — meaning the entity closest to the customer has the least regulatory oversight.

How Do BaaS Platforms Prevent One Brand from Overwhelming the System?

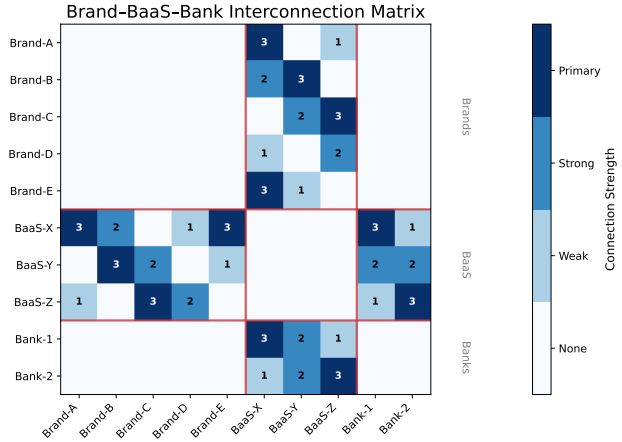
```python
import time

class TokenBucket:
    """Rate limiter for BaaS API access.
    Prevents one brand from consuming
    disproportionate infrastructure capacity."""

    def __init__(self, rate, capacity):
        self.rate = rate      # tokens per second
        self.capacity = capacity
        self.tokens = capacity
        self.last_refill = time.time()

    def _refill(self):
        now = time.time()
        elapsed = now - self.last_refill
        self.tokens = min(self.capacity,
            self.tokens + elapsed * self.rate)
        self.last_refill = now

    def consume(self, n=1):
        self._refill()
        if self.tokens >= n:
            self.tokens -= n
            return True  # Request allowed
        return False     # Rate limited (429)

# 100 req/s burst, 10 req/s sustained
limiter = TokenBucket(rate=10, capacity=100)
```

Protecting shared infrastructure

- **Token bucket:** Tokens accumulate at a steady rate; each API call consumes one. Burst capacity handles spikes; sustained rate prevents abuse
- **Why it matters:** In BaaS, dozens of brands share one charter bank's core system. One brand's Black Friday spike could freeze another's payroll processing
- **Tiered limits:** Premium brands get higher burst capacity (500 req/s) vs free-tier (50 req/s)
- **HTTP 429:** Rate-limited requests return "Too Many Requests" — the brand must retry with exponential backoff
- **Operational resilience:** EU DORA and OCC guidance now require BaaS providers to demonstrate fair resource allocation across tenants

Rate limiting is not just an engineering concern — it is a regulatory one. Fair resource allocation across brands on a shared BaaS platform is an operational resilience requirement.

How Interconnected Are the Players in the BaaS Ecosystem?



- Adjacency matrix showing connection density between charter banks, middleware providers, and brands
- Darker cells = more API connections between entities
- Hub pattern: 2–3 middleware providers connect to nearly every charter bank AND every brand
- Charter banks show low inter-bank connectivity (siloed)
- Brands rarely connect to more than one middleware provider (high switching costs)
- **Key insight:** middleware providers are the structural bottleneck — they create a dependency that neither banks nor brands can easily escape

The adjacency matrix reveals middleware as the structural bottleneck. Every bank-to-brand connection passes through 2–3 middleware hubs — a new form of systemic concentration.

What Happens When the Invisible Middle Layer of BaaS Disappears?

The Synapse collapse (2024): a case study in BaaS failure modes.

What happened:

- Synapse Financial Technologies operated as a BaaS middleware connecting fintech brands (Yotta, Juno, Copper) to Evolve Bank & Trust
- In April 2024, Synapse filed for bankruptcy after a $85M discrepancy was found between its ledger and Evolve's records
- 100,000+ consumer accounts were frozen immediately
- Consumers could not access their funds for months
- Neither the brand apps nor Evolve could independently reconstruct who owned what
- **The ledger gap:** Synapse maintained a "virtual ledger" that disaggregated pooled FBO (For Benefit Of) accounts, but this ledger was not reconciled with the bank's own records

Regulatory lessons:

- **FBO risk:** Funds held in pooled FBO accounts are not individually insured by FDIC — insurance attaches to the bank, not the virtual sub-accounts
- **Reconciliation:** Middleware must perform daily reconciliation against the charter bank's general ledger; Synapse did not
- **Regulatory gap:** No US regulator directly supervised Synapse as a middleware provider — it fell between the OCC (bank), state regulators (brands), and no one
- **OCC response:** Consent order against Evolve Bank requiring direct oversight of all BaaS partnerships
- **Industry impact:** Multiple banks (Sutton, Cross River) now require real-time ledger access to middleware providers

Synapse proved that the invisible middle layer is the single point of failure in BaaS. When it collapsed, 100K+ customers lost access because nobody had the complete ledger.

Why Is It Harder to Leave a BaaS Provider Than to Leave a Bank?

API Switching Cost Model

Total switching cost for migrating from BaaS provider A to B:

$$C_{\text{switch}} = \sum_{i} \left(C^{(i)}_{\text{rewrite}} + C^{(i)}_{\text{testing}} + C^{(i)}_{\text{migration}} + C^{(i)}_{\text{downtime}} \right)$$

where i indexes each integrated API endpoint (accounts, payments, cards, KYC, ledger, reporting).

Component breakdown:

- C_{rewrite}: Engineering hours to rewrite API integrations. Typically 2–6 months of a 5-person team (**$**200K–**$**600K)
- C_{testing}: Regression testing across all payment flows. 4–8 weeks, critical for compliance
- $C_{\text{migration}}$: Moving customer accounts, balances, and transaction histories. Must be zero-downtime for live accounts
- C_{downtime}: Revenue lost during migration. For a brand processing **$**10M/month, even 1% downtime = **$**100K

Lock-in mechanisms:

- **Proprietary schemas:** Each BaaS provider uses different data models — there is no "standard" account object
- **Webhook contracts:** Event-driven architectures create deep coupling with the provider's event taxonomy
- **Compliance artifacts:** KYC records and audit trails are stored in the provider's format — re-onboarding customers may be required
- **Network effects:** More brands on a platform means better fraud models, better data, and more shared infrastructure investment

Estimated total: **$500K–$2M** per migration for a mid-size fintech brand.

Switching BaaS providers costs $500K–$2M and 3–6 months of engineering time. PSD2 reduced bank switching costs but BaaS switching costs are an order of magnitude higher.

How Should a Brand and a BaaS Provider Split the Revenue?

```python
import numpy as np
from scipy.optimize import minimize_scalar

def nash_bargaining(revenue, brand_outside,
                    baas_outside, brand_power=0.5):
    """Nash bargaining solution for BaaS
    revenue split.
    revenue: total revenue to split
    brand_outside: brand's outside option
    baas_outside: BaaS provider's outside option
    brand_power: bargaining power [0,1]
    Returns: (brand_share, baas_share)."""
    surplus = revenue - brand_outside - baas_outside
    if surplus <= 0:
        return 0, 0 # No deal possible

    def neg_nash(brand_take):
        baas_take = revenue - brand_take
        u_brand = brand_take - brand_outside
        u_baas = baas_take - baas_outside
        if u_brand <= 0 or u_baas <= 0:
            return 0
        return -(u_brand**brand_power *
                 u_baas**(1-brand_power))

    res = minimize_scalar(neg_nash,
        bounds=(brand_outside, revenue-baas_outside),
        method='bounded')
    return round(res.x, 2), round(revenue-res.x, 2)

print(nash_bargaining(100, 30, 20, 0.6))
```

Bargaining over the surplus

- **Nash bargaining:** Each party gets their outside option plus a share of the surplus proportional to their bargaining power
- **Outside options:** Brand could build in-house (**$**30); BaaS provider could find another brand (**$**20). Surplus = **$**100 − **$**30 − **$**20 = **$**50
- **Bargaining power:** With $\alpha = 0.6$ (brand slightly stronger), brand gets **$**60, BaaS gets **$**40
- **Switching cost effect:** Higher switching costs reduce the brand's outside option, shifting surplus to the BaaS provider
- **Implication:** API standardization would increase brand outside options and compress BaaS margins

Nash bargaining shows that BaaS revenue splits depend on outside options. High switching costs reduce brand options — API standardization would rebalance the negotiation.

Why Do BaaS Platforms Subsidize One Side and Tax the Other?

Two-Sided Market Equilibrium (Rochet-Tirole, 2003)

A BaaS platform serves two sides: brands (B) and charter banks (K).

Demand on each side depends on the OTHER side's participation:

$$n_B = D_B(p_B, n_K) \qquad n_K = D_K(p_K, n_B)$$

Optimal pricing:

$$p_B^* = c_B - \delta_B \cdot n_K \qquad p_K^* = c_K - \delta_K \cdot n_B$$

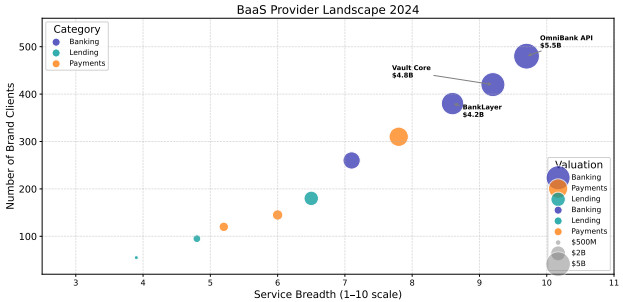
- c_B, c_K = marginal cost of serving each side
- δ_B, δ_K = cross-side network externality
- **Key insight:** the side with the stronger externality gets subsidized (possibly below cost)
- In BaaS: brands have stronger externality ($\delta_B > \delta_K$) so brands pay less

Price structure $\neq$ price level:

- Total revenue: $\Pi = p_B \cdot n_B + p_K \cdot n_K$
- It can be optimal to set $p_B < 0$ (pay brands to join) if the resulting bank enrollment generates enough revenue
- **Example:** Stripe Treasury charges brands 0% platform fee but earns 20+ bps from the charter bank on deposits
- **Tipping:** Once one platform reaches critical mass, competing platforms cannot attract either side — winner-take-most
- **Regulatory implication:** Antitrust must consider BOTH sides; taxing one side may collapse the platform

BaaS platforms are two-sided markets. They subsidize brands (strong externality) and charge banks. This explains "free" BaaS tiers and why market concentration emerges naturally.

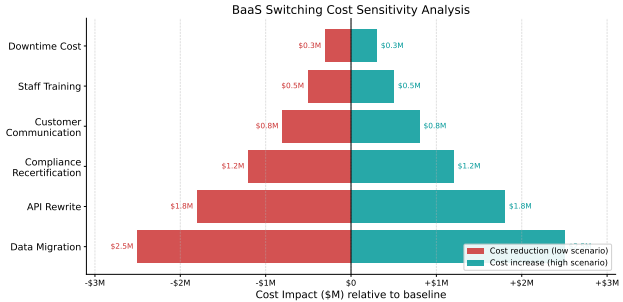
Which BaaS Providers Are Positioned for Dominance – and Which Are Vulnerable?



- Scatter plot: X = number of brand partnerships, Y = regulatory coverage breadth, bubble size = funding raised
- Top-right quadrant: Unit, Treasury Prime (many partners, broad coverage) — platform leaders
- Bottom-right: Railsr, Weavr (many partners but narrow geography) — regional specialists
- Top-left: Solaris, Solarisbank (deep regulation, fewer partners) — European champions
- Bottom-left: early-stage entrants with neither scale nor breadth
- **Key insight:** the top-right quadrant exhibits winner-take-most dynamics — brands prefer providers with the most bank connections

BaaS provider landscape shows clear winner-take-most dynamics. Providers with more bank connections attract more brands, which attracts more banks — a self-reinforcing cycle.

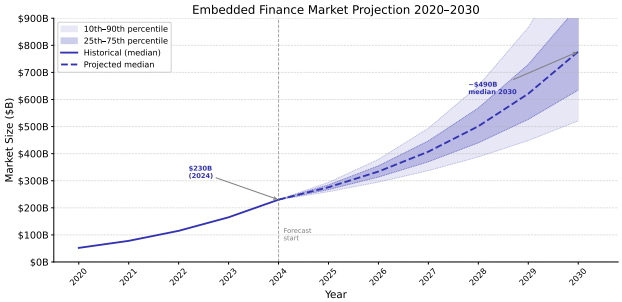
Which Factor Contributes Most to BaaS Switching Costs?



- Tornado (sensitivity) diagram showing the impact of each switching cost component on total cost
- API rewrite cost: widest bar — the single largest cost driver (35–40% of total)
- Customer re-onboarding: second largest — KYC re-verification required if data cannot be ported
- Revenue downtime: significant for high-volume brands
- Compliance re-certification: varies by jurisdiction (minimal in US, significant in EU)
- Testing and QA: relatively stable regardless of scale
- **Key insight:** API standardization would collapse the largest bar; data portability would collapse the second

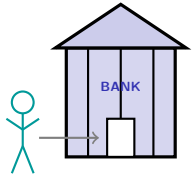
API rewrite and customer re-onboarding are the two largest switching cost components. Standardization (like PSD2 did for bank APIs) would dramatically reduce BaaS lock-in.

How Fast Will Embedded Finance Grow – and What Could Go Wrong?

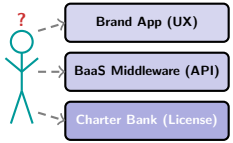


- Fan chart showing embedded finance market projections with confidence intervals
- Central estimate: $320B by 2030 (30%+ CAGR from 2024)
- Bull case ($500B+): regulatory clarity accelerates adoption; BaaS standards emerge
- Bear case ($150B): Synapse-type failures trigger regulatory crackdown; banks pull back from BaaS partnerships
- Key uncertainty: will regulators mandate direct oversight of middleware, increasing compliance costs and slowing growth?
- **Key insight:** the fan width reflects genuine uncertainty — embedded finance could be the next trillion-dollar layer or the next regulatory casualty

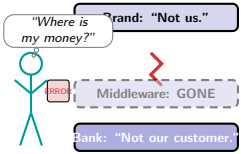
The wide confidence interval reflects genuine uncertainty. Embedded finance growth depends on a regulatory question nobody has answered: who is accountable for the middleware layer?



2015: Walk in. Open account.



2025: Tap app.
Three companies serve you.



2030: Middle layer fails.
Who do you call?

Embedded finance made banking invisible. The question is whether invisible banking also means invisible accountability.