In-Class Exercise: Composability Business Models

Exercise 1: Structured Debate — "Is Yearn a Yield-Aggregator Product or a Parasite on Curve?"

Format: Split into two teams. Each team prepares arguments for its assigned position, then presents. After both sides speak, the class votes — but first, read the debrief questions.

Team A — "Yearn Is a Stand-Alone Yield-Aggregator Product"

Anchoring evidence: Yearn ships its own vaults, has a multi-year track record of curated strategies, owns governance-token-holder loyalty, and has rebundled into curated, synthetic, levered and partner vaults that go far beyond pure routing.

Team A: Yearn Is a Stand-Alone Product	
Argument I	
Argument II	
Argument III	
Concession	*Strongest argument AGAINST your position:*
Closing	*How you address the concession:*

Team B — "Yearn Is a Parasite on Curve"

Anchoring evidence: Yearn's flagship vaults route through Curve pools for execution, depend on Curve's invariant for slippage protection, and capture governance influence over Curve through accumulated voting tokens. If Curve flipped a fee-switch tomorrow, Yearn's vault economics would shift overnight.

Team B: Yearn Is a Parasite on Curve	
Argument I	
Argument II	
Argument III	
Concession	*Strongest argument AGAINST your position:*
Closing	*How you address the concession:*

Debrief Questions

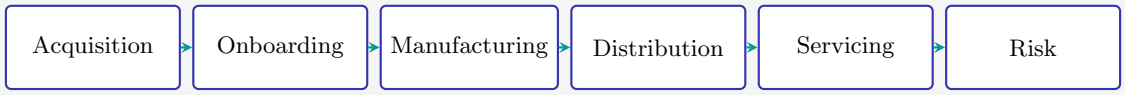
Q1: Does the answer — product or parasite — matter for how integrators above Yearn should reason about counterparty risk?

Q2: Could the answer genuinely be "both"? If so, what does that imply about the usefulness of the traditional product-versus-platform category?

Q3: Name another protocol (composable or not) whose value depends on a primitive it does not own. What tensions does that dependency create?

Exercise 2: Composability Value Chain Mapping

Scenario: The composable on-chain stack can be decomposed into six links. Specialised protocols attack individual links. Your task: for each link, identify a protocol, describe the composability friction it removes, and predict the long-term outcome relative to the host primitive below it.



Value Chain Link	Protocol Attacking It	Composability Friction Removed	Replaces or Improves Host?	Host Loses or Adapts?
Acquisition				
Onboarding				
Manufacturing				
Distribution				
Servicing				
Risk				

Synthesis Question

Q1: Which link in the composable value chain is *most vulnerable* to fee-switch politics and forks? Which is *most resistant*? Defend your reasoning with reference to liquidity-driven switching costs, governance lock-ins, and integrator hard-coding.

Facilitator Solutions

Sample answers for instructor reference. These are illustrative; student reasoning may diverge and still be valid.

Exercise 1: Debate Sample Answers

Team A (Yearn Is a Stand-Alone Product) — sample arguments

Argument I. Yearn has built a recognisable product brand around its curated vaults. Depositors trust the vault label, not the underlying routing logic, and would follow that brand even if the host primitive shifted. The relationship is depositor-to-Yearn, not depositor-to-Curve.

Argument II. Yearn ships products that have nothing to do with any single host primitive — synthetic vaults, levered vaults, partner-chain vaults, treasury-management vaults. Each one widens the surface of activity Yearn captures, decoupling protocol revenue from any single dependency.

Argument III. Yearn has accumulated governance influence in multiple lower primitives, including Curve. That accumulated voting power is itself a non-portable asset that any fork would have to rebuild from zero. The protocol owns governance positioning, not just routing logic.

Concession. The strongest argument against Team A is that a meaningful share of Yearn's flagship vault revenue would evaporate if Curve made a hostile change to its invariant or fees, exposing the dependency relationship.

Closing. Even acknowledging the dependency, Yearn has assembled a curated-vault brand, rebundled into multiple verticals, and accumulated governance positions that no fork could replicate. The product label captures value the routing logic alone never would.

Team B (Yearn Is a Parasite on Curve) — sample arguments

Argument I. Yearn's flagship vaults explicitly route execution through Curve pools, depend on Curve's invariant for slippage and price discovery, and could not exist if Curve removed access at the contract level. The economic substance of these vaults is Curve activity wearing a Yearn wrapper.

Argument II. A consumer-of-host BM has the shortest arbitrage half-life of any composable structure. Curve can flip a fee-switch, ship its own aggregator, or governance-vote a fork into existence in a single epoch, and Yearn's economics would change overnight without Yearn doing anything wrong.

Argument III. Yearn's accumulated governance positions in Curve only make the dependency more obvious, not less. The need to spend on vote-buying or token-locking to stay in good standing with the host is a tax that pure stand-alone products never face.

Concession. The strongest argument against Team B is that Yearn has rebundled across many host primitives and ships products that have nothing to do with Curve, so the parasite framing covers only one product line and not the protocol as a whole.

Closing. Even granting the rebundling, the flagship vaults that built Yearn's depositor base are structurally consumers of Curve's invariant. The substance of those vaults is host activity in a wrapper, regardless of the protocol's later product expansion.

Debrief Q1 — Counterparty risk for integrators above Yearn

The product-versus-parasite question matters because it tells an integrator above Yearn what governance event would break its position. If Yearn is a stand-alone product, the relevant counterparty risks are Yearn-internal: vault strategy errors, governance attack on Yearn itself, smart-contract bugs in Yearn's own code. If Yearn is a host-dependent consumer, the relevant counterparty risks include events at Curve — a fee-switch flip, an invariant change, a governance-driven

fork — that have nothing to do with Yearn's own actions. Integrators that mis-classify the dependency systematically under-price their tail risk.

Debrief Q2 — "Both" as an answer

The answer can genuinely be "both": Yearn operates a recognisable product brand on top of a structural dependency on lower primitives. That duality reveals that the traditional product-versus-platform category, inherited from a world where each firm controlled its full stack, cannot cleanly capture a protocol that ships a curated brand on top of contracts it does not own. If "both" is the right answer, it implies that integrators, regulators and analysts need new classification systems — functional rather than ownership-based — that focus on which composability dependencies a protocol actually carries rather than what label it prefers.

Debrief Q3 — Cross-protocol dependency example

A non-composability parallel is a third-party app store extension that depends on a single host platform's API surface. The extension can build a brand, accumulate users, and ship adjacent features, but a single API change at the host can erase substantial revenue overnight. The parallel to Yearn is direct: the dependency is not a marketing claim but a structural consequence of running a value-creation engine on top of someone else's primitive. Protocols whose flagship product is a consumer of a host primitive should price the host's optionality on a fee-switch flip into their own valuation.

Exercise 2: Composability Value-Chain Mapping Sample Answers

Value Chain Link	Protocol Attacking It	Composability Friction Removed	Replaces or Improves Host?	Host Loses or Adapts?
Acquisition	Wallet UI front-end	Direct contract interaction is opaque to non-developers	Improves	Host Adapts
Onboarding	Cross-chain bridge	Liquidity stranded on the wrong chain cannot reach the host primitive	Improves	Host Adapts
Manufacturing	Curve	Generic AMM slippage on like-priced asset pairs is too high for at-scale routing	Replaces generic AMM, sits beside lending	Host Loses (generic-AMM share)
Distribution	Yearn (yield-router branch)	Manual selection of best yield is gas-expensive and slow for retail capital	Improves	Host Adapts
Servicing	Oracle adapter network	Each primitive otherwise needs to wire its own price feeds and keepers	Improves	Host Adapts
Risk	Cover-protocol style insurance	Composability multiplies the cost of any single contract failure across the stack	Improves	Host Adapts

Synthesis Question Sample Answer

The most vulnerable link is the Distribution layer (yield routers, aggregators, splitters that consume host primitives). Switching costs at the Distribution stage are low: any router can re-point at any primitive, and competitors can ship a better algorithm in weeks. The host primitive can also flip a fee-switch or ship its own router and capture the same flow without paying the consumer. The most resistant link is the Manufacturing primitive that holds the deepest pool. Liquidity is the only resource a fork cannot copy, integrator hard-coding takes years to accumulate, and governance lock-ins (vote-locking, bribed-emission flows) compound the moat across cycles. Forks of a deep-pool primitive arrive every quarter; depositors do not move because every protocol above them has wired to the original address. The Manufacturing layer therefore captures the durable fee, while Distribution is rented to whichever consumer is most popular this season.

