0085a Incentives Are Physics, Not Levers

The Summers Principle Book · https://kb.jeffreysummers.com/docs/0085a-incentives-are-physics-not-levers/

The Frame

Every incentive in your operation is either designed or defaulted. There is no third state. This is [The Summers Principle] running at the incentive dimension.

The industry teaches incentives as levers. Adjust cast comp. Tweak the tip pool. Add a spiff on Tuesday nights. Roll out a Guest loyalty punch card. Every mechanism gets read as a lever pulled from outside a static system, and every adjustment gets measured against whether the lever produced the movement the operator wanted.

Incentives are not levers. Incentives are physics. They run as a recursion through your operation, cascading downward from your own operator layer through [Reward Structure Architecture] into cast and kitchen manager behavior into Guest Experience into Profit, and feeding upward through outcomes into your [Causal Read] into your next reward decision. The recursion runs whether you designed it or not. What you get to decide is which direction it runs — compounding or contracting — and that decision gets made at a specific layer you are inside of, not outside of.

This article walks you through the physics.

Incentives Act Through Anticipation

The first thing to lock is what an incentive actually is. An incentive is not a reward you deliver. An incentive is an anticipated reward that drives behavior before the reward arrives.

The cast member takes an action because they anticipate a reward on a timeline they read at a confidence level their own read of cause-effect supports. The action is a bet on the anticipation. Whether the reward arrives is a separate question from whether the anticipation drove the action. Delivered rewards feed back into the read and update the next anticipation, but the operative mechanism at the moment of action is the anticipation, not the delivery.

This matters because it puts the read discipline upstream of the reward structure. What people anticipate is a function of how they read cause-effect. Two identical reward mechanisms produce

different behavior in two different operators' shops because the cast members are reading cause-effect differently based on what they have been trained by prior cycles to anticipate.

This is why the operator's own [Causal Read] discipline is the load-bearing layer of the whole system. The read is what determines what gets anticipated. The anticipations are what drive the behavior. The mechanism is downstream of the read.

The Two Dimensions Of Causal Read

The operator's [Causal Read] runs on two dimensions, not one. Certainty — how reliably does the causal chain from action to outcome hold. Immediacy — how many operating cycles between the action and the observable outcome.

Every reward candidate the operator considers gets weighted against both dimensions. High certainty plus high immediacy — the causal chain is tight and the outcome shows this cycle — reads as strong. Low certainty plus low immediacy — the causal chain runs across multiple cycles and cannot be fully verified against observation in any single cycle — reads as weak.

The natural bias runs against low-certainty plus low-immediacy rewards. Operators default to weighting immediate visible cause-effect over multi-cycle causal chains because immediate visible causation is what the built environment produces evidence for on every cycle. Multi-cycle causation requires the operator to hold the causal model against partial observation across cycles, refuse to attribute misses to external causes on any single cycle, and update the model against aggregate observation over time. This is disciplined work. The absence of the discipline is not a neutral state — it is the natural weighting bias running unchecked, and unchecked bias systematically produces reward structures that suppress compounding rewards.

Every operator who says "I know Relational rewards would work, I just can't fund them right now" is running unchecked bias. The [Causal Read] on Relational alternatives is being suppressed by the bias, and the suppression is showing up as an affordability read that is actually a certainty-and-immediacy read. Naming this is the first move. The framing that Relational rewards cannot be funded is almost never a math read — it is a certainty-and-immediacy read presented as a math read.

The Two Reward Branches

Every reward mechanism in your operation runs on one of two cadences.

[Transactional Reward] runs on a single operating cycle. Action this shift, reward this shift or next. Cause-effect visible in the current cycle. Cast comp per shift, per-shift spiffs, immediate recognition, loyalty punch cards that clear within one Guest visit, standard tip pool, immediate performance feedback. High certainty plus high immediacy [Causal Read] profile. This is the branch the natural weighting bias defaults to.

[Relational Reward] runs across multiple operating cycles. Action this shift, aggregate reward this quarter or this year. Cause-effect visible only against multi-cycle patterns. Cast development, kitchen manager development, cast promotion pathway, deferred profit-share, meaningfully differentiated value work, Guest recovery investment, positioning capital build. Low certainty plus low immediacy [Causal Read] profile. This is the branch that gets chronically underfunded when the [Causal Read] runs on defaults.

Neither branch is superior. Both are load-bearing. The Product is a compounding outcome that requires Relational rewards at Product-side nodes to produce reliably across cycles. Guest re-encounter is a compounding outcome that requires Relational rewards at Guest-recovery nodes. Cast capability is a compounding outcome that requires Relational rewards at cast-development nodes. Positioning capital is a compounding outcome that requires Relational rewards across the whole architecture. The compounding outcomes cannot be produced by Transactional rewards alone. Transactional rewards produce Transactional outcomes — real ones, extractable ones, cycle-by-cycle ones — but not the compounding base that turns single cycles into a business.

The design work is not choosing one branch over the other. The design work is matching the reward cadence to the outcome cadence at every node. Single-cycle outcomes get Transactional rewards. Multi-cycle compounding outcomes get Relational rewards at load-bearing positions. Mismatches — Transactional rewards installed at compounding-outcome nodes, Relational rewards installed at single-cycle nodes — are the failure mode. Under [Two Roads] this maps to Road 1 (Transactional) and Road 2 (Relational) running through the incentive dimension.

The single most common failure mode is Transactional rewards filling compounding-outcome nodes. The operator names the compounding outcome (Guest re-encounter, cast capability, meaningfully differentiated value) but the reward mechanism at that node runs on single-cycle cadence. Cast development gets funded through per-shift feedback and immediate promotion signals. Guest recovery gets treated as this-visit discount rather than as multi-visit relationship investment. Positioning work gets measured against this-quarter marketing spend rather than against multi-year coherence build. The outcome node is compounding; the reward mechanism is single-cycle; the mismatch produces the pattern the operator reads as "our people just don't get

it" or "our Guests just don't respond." The pattern is not a people problem or a Guest problem. It is a cadence mismatch at the reward layer.

The Design Layer

The [Reward Structure Architecture] is the layer where reward mechanisms get placed at nodes across the operation. Cast comp node, kitchen manager comp node, cast development node, Guest recovery node, positioning capital node, meaningfully differentiated value node, and every other node where a reward mechanism operates.

At every node, the architecture makes two decisions. First, what mix of [Transactional Reward] and [Relational Reward] runs at this node. Second, whether the reward cadence at this node matches the outcome cadence at this node. Every architecture is producing a coherence read across its nodes — the aggregate degree to which cadence matches outcome across the whole system.

Coherent architectures — where cadence matches outcome at every node — produce compounding Profit. Incoherent architectures — where cadence mismatches show up across multiple nodes — produce single-cycle Profit that runs strong through cost squeeze and finite-base depletion until the base runs out. Both architectures produce Profit. The shape of the Profit curve across cycles is what tells you which one you have installed.

The design layer is not the physics. The design layer is what the physics runs on. You configure the architecture, and the recursion runs on whatever you configured.

Under [The Summers Principle], every architecture is either designed or defaulted. A designed architecture is the output of disciplined [Causal Read] work at every node — the operator ran the two-dimensional confidence read, held the causal model against natural weighting bias, and placed reward mechanisms at outcome-cadence-matched positions. A defaulted architecture is the output of the natural weighting bias running unchecked — every node got the reward mechanism the bias produced, and the mechanism suppresses Relational alternatives at compounding-outcome nodes by default. Most operations run defaulted architectures because defaulted is what happens when the [Causal Read] discipline is not being run. It requires no active choice. It is the state the operation drifts into when no design work is happening at the architecture layer.

The Recursion

Here is where the physics gets specific. The reward architecture is not a static configuration you set once. It runs as a recursion — a loop that operates through your own operator layer in both directions.

Downhill from your operator layer. Your own draw structure, time allocation, compounding investments, and [Causal Read] discipline cadence configure the [Reward Structure Architecture] you install. The architecture sets the mix of [Transactional Reward] and [Relational Reward] at every node. The mix sets the anticipated rewards driving cast, kitchen manager, and Guest behavior. The behavior produces the Product, the Guest Experience, the Profit shape.

Uphill from behavior to your operator layer. Behavior produces outcomes. Outcomes feed back through your [Causal Read] and update your model of cause-effect. The updated model shapes your next reward-architecture decision. That next decision cascades downward. The next round of outcomes feeds back upward. And so on across every cycle.

This is a recursion, not a one-way cascade. Every downhill cycle produces the uphill data that shapes the next downhill cycle. The direction the recursion runs — compounding or contracting — gets determined at the [Causal Read] layer of your own operator layer.

When the [Causal Read] is disciplined, the uphill loop compounds. Outcomes update the model against observation. Your anticipations grow more accurate. Your reward decisions get better. The downhill cascade produces better outcomes. The next uphill loop strengthens the model further. Each turn of the loop makes the next turn easier. Your operation gets better at design work over time because the recursion is running on compounding nodes.

When the [Causal Read] is broken, the uphill loop contracts. Outcomes get attributed externally — market conditions, cast difficulty, kitchen manager gap, Guest behavior, vendor problems. The model does not update. Your anticipations stay calibrated to Transactional-dominant defaults. Your reward decisions regenerate the default configuration. The downhill cascade produces the same pattern of outcomes. The next uphill loop attributes those outcomes externally again. Each turn of the loop makes the next turn harder. Your operation gets worse at design work over time because the recursion is running on contracting nodes.

This is [Incentive Recursion]. It is not a metaphor. It is the physics of how incentive structures actually operate across cycles in a running operation.

The Operator Is Inside The Recursion

The most important thing to lock about the physics is where you sit in it. You are not designing the incentives from outside the system. You are one of the layers inside it.

Your own draw structure is a reward mechanism at the operator layer of the architecture. Your time allocation is a reward mechanism at the operator layer. Your compounding investments — meaningfully differentiated value work you fund out of your own capital, positioning capital you build across years, [Causal Read] discipline you run on cadence — are Relational reward mechanisms at the operator layer. Your posture toward design or default at the architecture layer is the reward-structure decision at the operator layer.

Every one of these is a layer of [Incentive Recursion]. The recursion cascades from the operator layer downward through cast, kitchen manager, and Guest layers, and feeds back upward from those layers to the operator layer through the [Causal Read] loop. You are inside the recursion. Not outside it.

This is why incentive adjustments at downstream nodes drift back to default when the upstream layer has not been adjusted. The recursion regenerates the pattern from the operator layer. You can adjust cast comp; the adjustment holds for a few cycles; the recursion regenerates the default because your own operator layer — the one producing the anticipations you are training your [Causal Read] on and the reward mechanisms you are training your architecture with — has not been adjusted.

Operators who cannot see themselves inside the recursion spend design work at downstream nodes forever. Every adjustment holds for a while and drifts back. They read the drift as external. Cast turnover is up. Kitchen manager stalls. Guest tolerance erodes. The market shifted. The generation changed. What is actually running is the recursion regenerating from an operator layer that has not been touched, and the drift is the physics revealing itself. Regeneration is not resistance. Regeneration is the recursion running unchecked.

The Compounding Cascade Versus The Contracting Cascade

The recursion runs on one of two configurations at any given time. The Profit shape across cycles is your aggregate diagnostic of which one you have.

Compounding cascade. Disciplined [Causal Read] at the operator layer. Designed [Reward Structure Architecture] with [Relational Reward] at load-bearing positions at compounding-outcome nodes. Cast behavior aligns to compounding outcomes. Kitchen manager behavior aligns

to compounding outcomes. Guest Experience compounds across cycles. Meaningfully differentiated value builds. Positioning capital accumulates. Profit runs slower per cycle early — because Relational rewards do not produce single-cycle payoff — but compounds across cycles because the base is building. Ten years in, the operation is meaningfully harder to compete with than it was five years in, because five years of compounding base has accumulated.

Contracting cascade. Undisciplined [Causal Read] running on natural bias at the operator layer. Defaulted [Reward Structure Architecture] with [Transactional Reward] dominating even at compounding-outcome nodes. Cast behavior aligns to visible short-term signals. Kitchen manager behavior aligns to visible short-term signals. Guest experience degrades to whatever cast behavior produces on Transactional-dominant incentives. Meaningfully differentiated value work does not get funded. Positioning capital does not build. Profit runs strong early — because Transactional rewards produce this-cycle extraction — but contracts across cycles because the base is depleting. Ten years in, the operation is running on cost squeeze and finite-base depletion, and the operator cannot understand why the same moves that worked five years ago are not working now. The moves did not stop working. The base ran out.

Every operation is running one of these two cascades. The recursion is not neutral. It compounds or it contracts. There is no static state. This is [No Static Achievement] running at the reward-architecture layer.

The Operator Layer Is Where The Design Work Happens

This is the punchline of the physics. The recursion's direction — compounding or contracting — gets determined at your own operator layer. Specifically at your [Causal Read] discipline.

Not at cast comp. Not at kitchen manager comp. Not at the tip pool. Not at the loyalty program. Not at the recognition system. Those are all downstream nodes. They matter, and they get designed, but the recursion regenerates their default configuration from the operator layer if the operator layer is not being touched.

The operator layer intervention is the load-bearing move. Your own draw structure — is it running Transactional-dominant (immediate cash draw prioritized over compounding investment) or is it holding Relational components (deferred draw, capital reinvestment, compounding position build). Your time allocation — is it running on this-week fires and immediate visible work, or is it holding cadenced [Causal Read] discipline, cadenced model update, cadenced architecture review. Your compounding investments — are you funding meaningfully differentiated value work, positioning capital build, Guest recovery investment, kitchen manager development, cast development, or is

the funding going to whatever produces this-quarter visible movement.

Your [Causal Read] discipline is the layer where every other decision gets shaped. If the discipline is running — explicit causal models, two-dimensional confidence reads at every reward decision, cross-cycle anticipation holding, cadenced model update against aggregate observation, [The Operator's Own Audit] running on schedule — the recursion runs compounding by physics. If the discipline is not running, the recursion runs contracting by physics. Not by choice. By physics.

The industry teaches downstream nodes because downstream nodes are where the visible mechanisms live. This article is teaching you that the mechanisms are outputs of a recursion, not levers on a static system, and that the recursion's direction gets set at the layer you sit inside — your own [Causal Read], your own reward configuration, your own architecture-layer posture.

What Changes Tomorrow

Tomorrow you pick one behavior pattern in your operation that you have been reading as an external problem. Cast turnover trending up. Kitchen manager stalled. Guest recovery not happening. Meaningfully differentiated value work not landing. Positioning capital not building. Cast development runs but produces no promotion-ready leads. Whatever the pattern is, pick one specific one.

Then you run the recursion trace on it. Both directions.

Downhill trace. Start at your own operator layer. What is your draw structure — is any part of it deferred or reinvested, or is it pure this-cycle extraction. What is your time allocation across this week — how many hours went to cadenced [Causal Read] discipline versus immediate visible work. What compounding investments are you funding — meaningfully differentiated value work, positioning capital build, kitchen manager development, cast development, Guest recovery investment. Now the architecture. What is the [Reward Structure Architecture] configuration at the node where the behavior pattern shows up. What mix of [Transactional Reward] and [Relational Reward] runs at that node. Is the reward cadence at that node matching the outcome cadence at that node, or is it mismatched. Now the anticipation. Given the reward mechanism at that node, what is the person at that node anticipating. Given that anticipation, what behavior does the physics produce. Compare the physics-produced behavior to the behavior pattern you have been reading. They should match.

Uphill trace. Start at the behavior pattern. What does the behavior tell you about what the person is anticipating. What does the anticipation tell you about the reward mechanism at that node.

What does the reward mechanism tell you about the architecture configuration you installed at that node. What does the configuration tell you about your own [Causal Read] on the causal chain from configuration to outcome at that node. Where in the trace are you attributing outcomes externally when internal attribution is available. Specifically — for the misses in this pattern, are you reading market conditions, cast difficulty, Guest behavior, or are you reading your own [Causal Read] miss, your own weighting bias suppression, your own architecture configuration.

When the trace completes, you will have located the layer where the recursion is producing the pattern. If the trace lands at the operator layer — your own [Causal Read] is running unchecked bias, your own reward layer is producing the ceiling the downstream layer is running under — the intervention is at your own layer first. Downstream adjustments wait until the upstream intervention is running.

Set a cadence. Once a week for the next four weeks, pick one behavior pattern and run the recursion trace on it. At the end of four weeks, review the four traces together and read the aggregate. If three or four of the traces landed at the operator layer — same [Causal Read] miss, same architecture configuration, same weighting bias suppression — you have located the load-bearing intervention. That intervention runs before any further downstream design work on those patterns.

The frame you now run is that incentives are physics, not levers. The physics runs as a recursion through your own layer, in both directions, and its direction gets determined at your [Causal Read] layer by whether disciplined causal work overrides the natural weighting bias. Every behavior pattern in the operation is an output of the recursion running on whatever configuration is currently installed. Adjusting downstream mechanisms without adjusting the upstream layer that regenerates them is treating a symptom of the recursion instead of the recursion itself. Your design work is at the recursion layer — your [Causal Read] discipline, your operator-layer reward configuration, your architecture mix — or the design work does not hold.