ORDER OTTER · PRACTITIONER WHITE PAPER

From Forecasts to Orders

A practical framework for replenishment decisions under uncertainty.

The short version: most teams treat replenishment as a forecasting problem. It isn't. A forecast is just an input. The real problem is an **economic decision under uncertainty** — and the order you place is a bet. Order Otter evaluates candidate orders across many plausible futures and recommends the one with the best economic tradeoff.

Audience: planners, operators, inventory managers, founders, supply chain leaders **Version:** 1.0

Reading time: ~25 min

1 Executive Summary

Replenishment is one of the highest-leverage decisions a business makes, and one of the most consistently mishandled. Order too little and you lose sales, disappoint customers, and hand share to competitors. Order too much and you bury cash in inventory that has to be stored, financed, discounted, or written off. The stakes are large, the decision recurs constantly, and the future is genuinely uncertain.

The prevailing approach is to pour effort into forecasting — chasing a more accurate number for next month's demand — and then convert that number more or less directly into an order. This paper argues that the forecast, however good, is only an input. The decision that actually moves money is *how much to order right now*, given uncertain demand, uncertain lead times, real supplier constraints, and the specific economics of the item: its margin, its holding cost, the penalty when it stocks out, and what it's worth if it's left over.

Order Otter reframes replenishment as an economic decision under uncertainty and solves it with **simulation optimization**. Rather than turning a forecast into an order, it proposes a set of feasible candidate orders, plays each of them through the same set of plausible demand and lead-time futures, scores every outcome in dollars and service, and recommends the order with the best tradeoff. The ranking layer that does this comparison is called **RaftRank™**.

The result is a recommendation a planner can actually defend: not "the model said so," but "this order makes the most money across the futures we're likely to face, meets our service target, and avoids the downside risk of the bigger order." This paper explains the framework end to end, works a concrete numerical example, and shows how the same engine extends from a single-SKU decision to constrained multi-SKU ordering and reusable ordering policies.

2 The Forecast Is Not the Decision

Ask most teams how they decide what to order and the answer is some version of: "We forecast demand, then we order to cover it." Forecasting gets the attention, the tooling, and the blame. When inventory goes wrong, the post-mortem usually concludes that the forecast was off — so next quarter the team resolves to forecast better.

But a forecast and a decision are different objects. A forecast is a statement about the world: *we expect to sell about 1,000 units over the next eight weeks.* A decision is an action you take: *order 1,500 units today.* The forecast is an input to the decision, not a substitute for it. Two businesses can share the identical forecast and still be correct to place very different orders, because their economics differ — one sells a high-margin product that's disastrous to stock out, the other sells a low-margin perishable that's disastrous to over-buy.

Worse, a forecast is a single number (or at best a single curve) standing in for a range of possibilities. The moment you commit to "1,000 units expected" as though it were fact, you've thrown away the most decision-relevant information you had: *how uncertain* that number is, and *what it costs you* when reality lands above or below it.

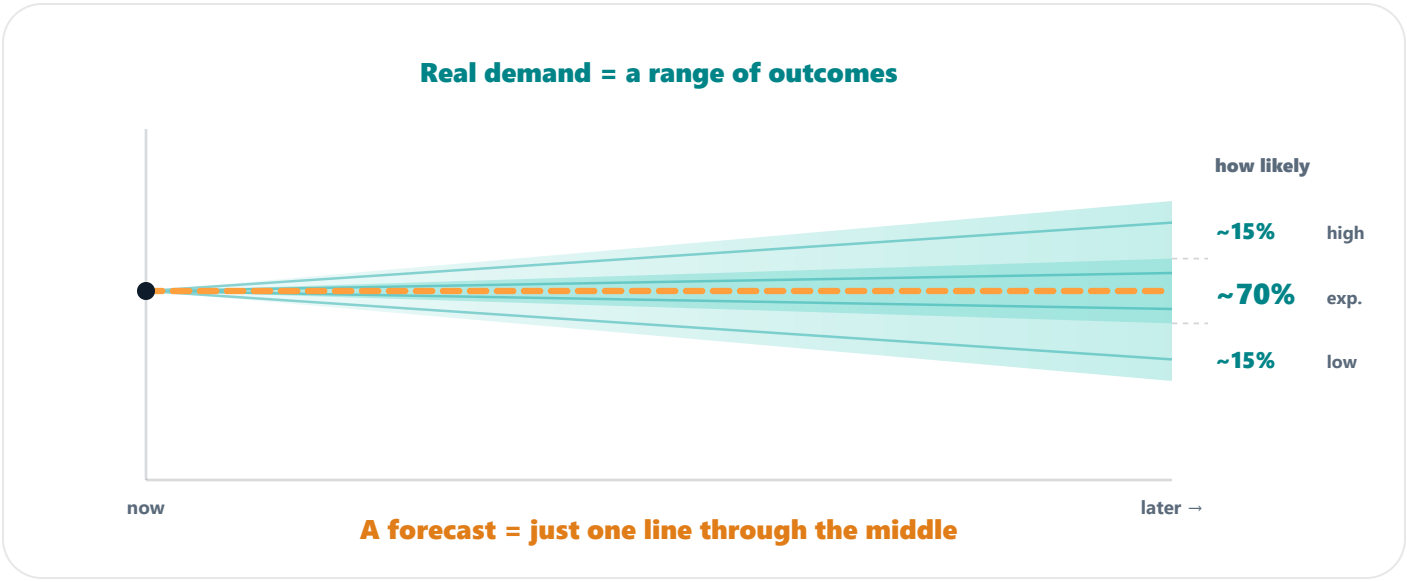

Figure 1 — A forecast is one tidy line, but real demand is a range — and outcomes near the middle are far likelier than the extremes (percentages illustrative). Order Otter plans against the whole distribution, not just the line.

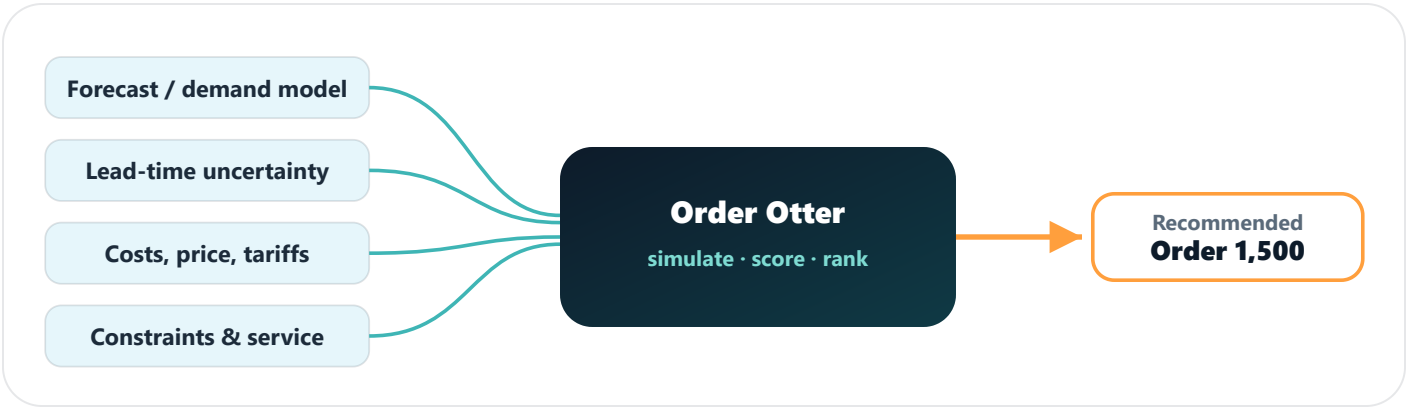

Figure 2 — A forecast is one input among several. The decision is the output.

3 Every Order Is an Economic Bet

When you place an order, you are committing cash today against sales you hope to make later. You don't yet know exactly how much you'll sell, or exactly when the goods will arrive to be sold. That makes every order a bet — a reasonable, informed bet, but a bet nonetheless.

Like any bet, its quality depends on the payoffs, not just the odds. The economics of an item determine how you should lean:

- **Underage cost** — what you lose per unit when you run short: lost margin, lost customers, expedite fees, contractual penalties, or substitution to a competitor.
- **Overage cost** — what you lose per unit left over: holding and financing cost, storage, markdowns, obsolescence, spoilage, and the opportunity cost of tied-up cash.

When stocking out is far more expensive than sitting on a few extra units, the right bet leans toward ordering more; when leftovers are ruinous — perishable goods, fast-obsolescing SKUs, thin margins under heavy tariffs — the right bet leans toward ordering less. The same demand forecast justifies different orders depending on this balance. Any framework that ignores it is optimizing the wrong thing.

This is why "hit the forecast" is not a goal worth having on its own. The goal is to place the order that produces the best distribution of economic outcomes given everything you know and everything you don't.

*Figure 3 — Every order balances the **cost of stocking out** against the **cost of over-ordering**. The SKU's economics decide which way the scale tips — when a stockout hurts far more than a leftover unit, it leans you toward ordering more (and vice-versa).*

4 Why Traditional Replenishment Logic Breaks Down

Most operational replenishment still runs on rules that were designed for a calmer, more predictable world. They aren't worthless — they're fast, legible, and often good enough — but they degrade badly under exactly the conditions that define modern supply chains.

Deterministic formulas assume away the risk

Reorder points, safety-stock multipliers, and order-up-to levels typically bake in a single demand rate and a single lead time, then bolt on a safety buffer sized by a service-target rule of thumb. This works when demand is stable and lead times are reliable. It falls apart when demand is spiky and lead times swing by weeks, because the buffer is calibrated to an "average" world that rarely shows up.

Constraints are treated as afterthoughts

Real orders must respect minimum order quantities, case packs, pallet rounding, supplier minimums, budget ceilings, warehouse capacity, and order calendars. Classic formulas produce a "clean" number that then gets manually nudged to something orderable — and every nudge quietly discards the economics the formula was trying to protect.

Tariffs and landed cost change the math

When a large slice of unit cost is tariff or freight, the penalty for over-ordering rises sharply. Buffers tuned before a cost shock become expensive the day the shock lands, and static rules don't notice.

The output can't be explained

Perhaps the quietest failure: a planner is handed a number and asked to trust it. When it's wrong, there's no way to interrogate *why* — no visible tradeoff between service and leftover risk, no sense of how fragile the number is if demand runs hot. Numbers that can't be explained don't get followed; they get overridden by gut feel, which puts you right back where you started.

What's needed is a method that treats uncertainty as first-class, respects real constraints natively, adapts when the economics move, and produces a recommendation with its reasoning attached.

5 Simulation Optimization for Replenishment

Simulation optimization is a simple, powerful idea: instead of solving for an order with a formula, you *try* orders and *test* them against many possible futures, then keep the one that performs best.

It combines two capabilities. A **simulator** answers the question "if I place this order, and this future unfolds, how does it turn out?" — it plays inventory forward week by week and tallies the

economics. An **optimizer** answers "which order should I try next?" — it proposes and searches over candidate orders. The recommendation emerges from the loop between them:

candidate decision → simulate futures → score outcomes → rank / search → recommend

The engine works through a clear sequence:

1. **Define the planning state** — current inventory, open purchase orders, the demand-uncertainty model, the lead-time distribution, price, cost, landed/tariff cost, holding cost, stockout penalty, salvage value, the service target, and the constraints that make an order feasible.
2. **Define the decision** — for the core case, "how much should I order for this SKU/location now?"
3. **Generate feasible candidate orders** — quantities that actually respect MOQs, case packs, budgets, and capacity.
4. **Generate uncertainty scenarios** — many plausible future paths for demand and lead time.
5. **Evaluate each candidate through simulation** — run every candidate against every scenario.
6. **Score outcomes economically** — convert each simulated future into profit, service, and risk.
7. **Aggregate** — summarize each candidate's performance across all scenarios.
8. **Rank and recommend** — surface the order with the best tradeoff for the business's risk tolerance.

A note on honesty

We do not simulate "every possible outcome" — that's neither achievable nor necessary. We simulate *many plausible, representative* futures: enough to see how each order behaves across good weeks and bad, fast arrivals and slow. The aim isn't to predict the one future that happens; it's to choose an order that holds up well across the futures you're likely to face.

6 Candidate Orders and Feasible Decisions

A candidate order is a specific quantity you could actually place. The word "actually" is doing real work here: the point of evaluating candidates rather than solving a formula is that candidates can

be constructed to be **feasible from the start**, so the recommendation is something you can submit to a supplier without hand-editing.

Feasible candidates respect the constraints that govern real ordering:

- MOQ multiples and supplier minimums
- Case packs, inner packs, and pallet quantities
- Budget and open-to-buy limits
- Warehouse and shelf capacity
- Order calendars and lead-time cutoffs
- Shelf-life and obsolescence limits

For a single SKU, a practical candidate set is often just a sensible sweep of orderable quantities — for example **0, 500, 1,000, 1,500, 2,000** units, each already snapped to the case pack. Including zero matters: sometimes the right move is to order nothing and let existing inventory run down. The engine doesn't need thousands of candidates to give a good answer; it needs a well-chosen set that spans the meaningful range, and it can refine around the promising region afterward (see Section 12).

7 Scenario Generation: Demand, Lead Time, and Cost Uncertainty

A scenario is one plausible version of the future, described in enough detail to simulate. Each scenario spells out, over the planning horizon:

- demand in each week (drawn from history or a demand-uncertainty model);
- the actual lead time, and therefore when a placed order arrives;
- and any cost movements relevant to the decision.

Figure 4 — A supplier quotes one lead time; reality is a distribution. Ordering for the spread is what keeps a slow boat from turning into a stockout.

Two design choices make scenarios trustworthy. First, they should be grounded in your data: if you have demand history, scenarios are sampled to reflect its real level, variability, and lumpiness rather than an idealized bell curve. If you only have estimates, scenarios are generated from an explicit uncertainty model you can inspect and adjust. Second — and this is the part that makes comparisons fair — **every candidate order is tested against the same set of scenarios.**

TECHNIQUE

Common random numbers

Testing all candidates against one shared "river" of futures is a controlled-experiment technique known as *common random numbers*. Because the only thing that differs between candidates is the order itself — not the luck of the draw — differences in their results reflect the decision, not noise. It sharpens the comparison and lets us reach a confident ranking with far fewer scenarios than naive random testing would require.

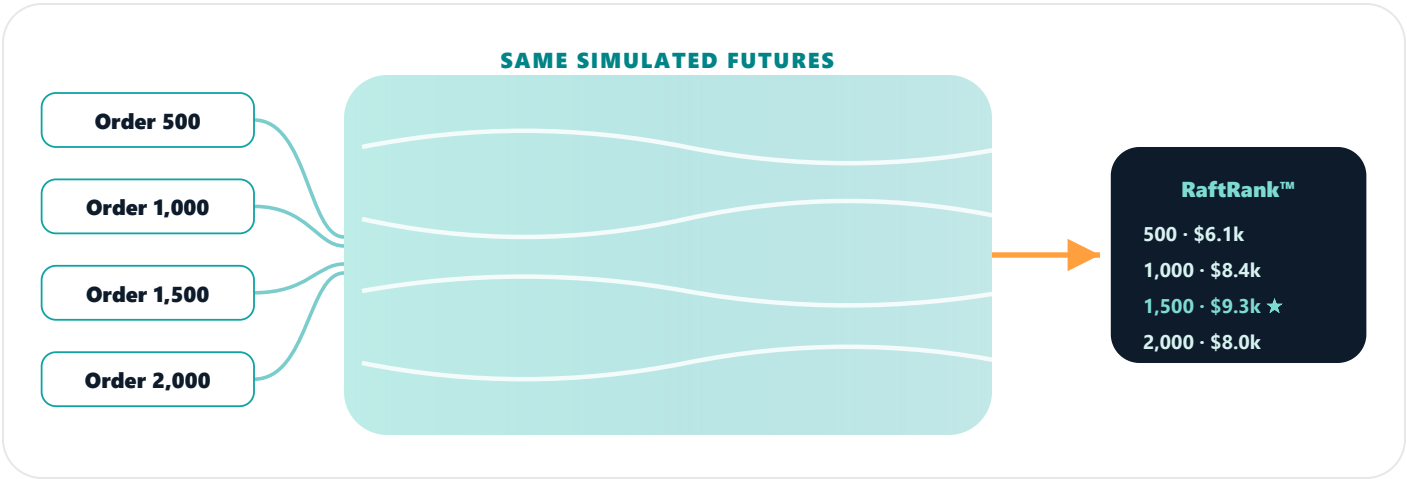

Figure 5 — Every candidate order is floated down the same river of futures, then ranked.

8 Simulating Inventory Flow

Simulation is where a candidate order meets a scenario and we find out what actually happens. For a given order quantity and a given future, the engine walks the planning horizon week by week and keeps an honest ledger of inventory:

- The order you place arrives after the scenario's lead time; until then, you serve demand from on-hand stock and any previously open POs.
- Each week, demand draws down inventory. Whatever you can fill becomes **captured sales**; whatever you can't becomes a **stockout** — lost sales (or backorders, if that's your model), with the associated penalty.
- Units that sit unsold accrue **holding cost** week over week.
- At the end of the horizon, whatever remains is **leftover inventory**, valued at its salvage or carry-forward worth.

Run this for one order against one scenario and you get a single, fully-costed story: this much sold, this much was missed, this much was left, and here's the resulting profit. Run it for that same order across hundreds of scenarios and you get a *distribution* of outcomes — not a point estimate, but a realistic picture of how that order behaves when the future is kind and when it isn't.

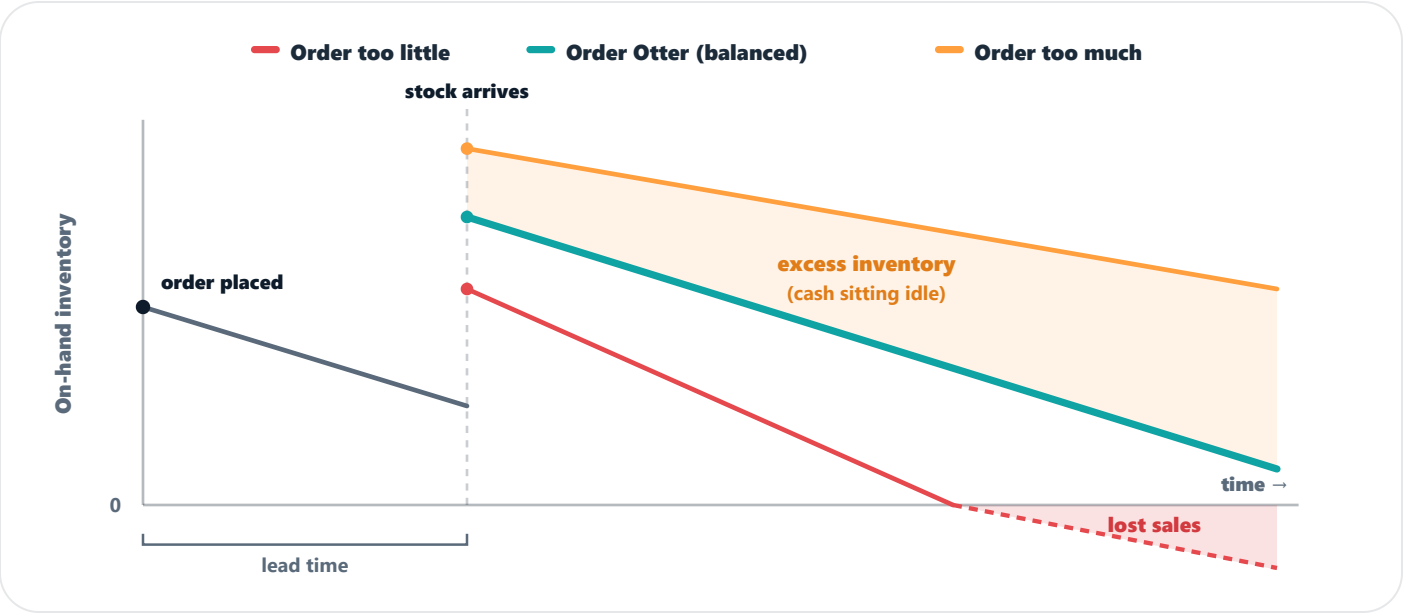

Figure 6 — The same demand drawdown, three order sizes. Order too little and the tank runs dry (lost sales); order too much and cash sits idle in leftover stock. Order Otter sizes the order to the economics in between.

9 Economic Scoring and Tradeoff Measurement

Every simulated future is scored in the currency that matters. A simplified profit score for one order in one scenario looks like this:

```
# profit for one candidate order in one scenario
profit = revenue_captured
       - product_cost
       - tariff_landed_cost
       - holding_cost
       - stockout_penalty
       - expedite_cost
       - leftover_inventory_penalty
```

Profit alone doesn't tell the whole story, so alongside it the engine tracks the operational measures planners and executives actually negotiate over:

- **Service level** — the share of demand filled from stock.
- **Stockout probability** — how often, across scenarios, the order runs short at all.
- **Average leftover inventory** — the excess you're likely to carry.
- **Cash tied up** — the working capital the order commits.

- **Downside risk** — how bad the poor outcomes get, summarized by measures such as worst-case profit or CVaR (the average of the worst tail of scenarios), so you can steer by risk appetite, not just averages.

This multi-dimensional scoring is what lets Order Otter present a *tradeoff* rather than a verdict. The biggest order almost always wins on service; the smallest almost always wins on cash exposure. The interesting question — the one a good recommendation answers — is where the best economic balance sits.

Figure 7 — More units always buys more service and more leftover. Profit peaks in between — that's the order to place.

10 Ranking Orders with RaftRank™

DEFINITION

RaftRank™

RaftRank™ is Order Otter's simulation-optimization ranking layer. It evaluates feasible candidate orders across the same set of plausible demand and lead-time futures, scores the outcomes using business economics, and surfaces the order with the best tradeoff.

The name is deliberate. A *raft* is what you call a group of otters floating together, paws linked so nobody drifts off. RaftRank™ floats your candidate orders down the same river of simulated futures,

keeps them side by side so the comparison stays fair, and reports back which one comes through best.

Concretely, RaftRank™ takes the aggregated results for each candidate — expected profit, service level, stockout probability, expected leftover, downside risk — and ranks them according to the business's stated objective and risk tolerance. A margin-rich retailer terrified of stockouts and a cash-constrained distributor wary of leftovers will weight those measures differently, and RaftRank™ reflects that: the "best" order is best *for you*, not in the abstract.

Two properties make the ranking dependable. Because all candidates ride the same scenarios (common random numbers), the ordering between them is stable rather than an artifact of lucky draws. And because the whole comparison is economic, the winner comes with its justification already attached — which is what makes the output explainable rather than oracular.

We describe what RaftRank™ optimizes for and how it compares orders; the specific internal search strategy and scoring refinements are proprietary. What matters for this paper is the shape of the method, not the trade secrets inside it.

11 Example: One SKU, Four Candidate Orders

Consider a single item over an eight-week horizon. Expected demand is roughly 1,000 units, but it's genuinely uncertain — some futures land near 700, some near 1,400. The unit sells for $20, costs $8 landed, carries a modest holding cost, and stocking out is expensive because the buyer simply goes elsewhere. Starting inventory is negligible. The team wants a 95% service target if the economics support it.

Order Otter builds a feasible candidate set — 500, 1,000, 1,500, and 2,000 units — and floats each down the same few hundred demand-and-lead-time futures. The aggregated results:

CANDIDATE ORDER	EXPECTED PROFIT	SERVICE LEVEL	STOCKOUT PROB.	AVG. LEFTOVER	DOWNSIDE (WORST 10%)
500 units	$6,100	63%	91%	5	$3,900
1,000 units	$8,400	84%	52%	70	$5,200
1,500 units 🦦	**$9,300**	**95%**	**16%**	**240**	**$6,400**

CANDIDATE ORDER	EXPECTED PROFIT	SERVICE LEVEL	STOCKOUT PROB.	AVG. LEFTOVER	DOWNSIDE (WORST 10%)
2,000 units	$8,000	99%	3%	610	$4,700

Table 1 — Aggregated outcomes for each candidate order across the same simulated futures. Illustrative figures.

Read down the columns and the tradeoff is plain. Ordering **500** keeps almost nothing left over but stocks out in nine futures out of ten — you're leaving money on the table. Ordering **1,000** looks reasonable on average but is fragile: it still runs short more than half the time, and a hot demand week hurts. Ordering **2,000** nearly guarantees service, but it buries cash in an average of 600+ leftover units, and those carrying and markdown costs drag expected profit back down.

1,500 units is the recommendation — not because it matches the forecast (it deliberately exceeds the ~1,000 expectation to buy resilience), but because it produces the best expected profit while clearing the 95% service target and keeping leftover risk in check. A planner can state the case in one breath.

PLANNER-FACING OUTPUT

Recommended order: 1,500 units

Why: highest expected profit among feasible options · meets the 95% service target · keeps stockout risk low (16%) · avoids the excess-inventory drag of the 2,000-unit option (240 vs 610 units left over on average).

12 From Single-SKU Decisions to Multi-SKU Optimization

Enumerating a handful of candidate orders is more than enough for a single SKU, and it's how the core case works today. As the problem grows, the same simulate-score-rank loop stays intact — only the way we *search* for good candidates gets smarter, so we spend simulation effort where it pays off:

- **Grid search** over feasible quantities for straightforward single-item decisions.
- **Local search** that refines around the promising region once a grid points the way.

- **Adaptive racing / successive elimination** that stops wasting scenarios on clearly-losing candidates early.
- **Surrogate models** that approximate simulation outcomes when each run is expensive, so the optimizer can explore cheaply and confirm with full simulation.
- **Bayesian optimization** for expensive simulations where each evaluation must count.
- **MILP and heuristics** for constrained multi-SKU ordering — allocating a shared budget, warehouse capacity, or supplier truckload across many items at once.

The multi-SKU case is where this matters most. When items compete for the same budget or the same pallet positions, you can't optimize them one at a time; a dollar spent over-buying one SKU is a dollar unavailable to protect another. The engine's job becomes allocating a constrained resource across items to maximize total economic outcome — still by simulating and scoring decisions, now with an optimizer that respects the shared constraints. The division of labor holds throughout: **the simulator evaluates decisions; the optimizer proposes and searches for better ones.**

13 One-Time Decisions vs Policy Optimization

There are two distinct questions hiding inside "what should we do about inventory," and it's worth keeping them separate.

One-time decision optimization asks: *what should I order today, given where I stand right now?* The answer is a specific quantity for a specific moment. This is the core of Order Otter and the focus of this paper, because it's the decision planners make constantly and the one where a better answer pays off immediately.

Policy optimization asks a broader question: *what ordering rule should I use over time, so I don't have to re-derive the decision from scratch every cycle?* Here the output isn't a single order but a reusable rule with tuned parameters. The same simulation engine evaluates policies instead of one-off orders — playing a candidate rule across many futures and scoring how it performs over the long run.

Once the foundation is in place, Order Otter can tune policies such as:

- reorder points and order-up-to levels (the *otter-up-to level*, if you'll allow it);
- service-level targets and safety-stock parameters;
- expedite rules (when it's worth paying to pull an order in);
- DC holdback rules;

- and allocation logic across locations.

The path is deliberate: nail the one-time decision first, because it delivers value on the next order, then extend to policies that automate good decisions at scale. Both rest on the same idea — evaluate decisions by simulating their consequences.

14 Why Planners Stay in Control

A recommendation engine earns adoption only if the people who own the order trust it — and trust comes from control and transparency, not from being told to defer to a black box.

Order Otter is built so the planner stays the decision-maker. The inputs are yours and visible: your demand history or estimates, your costs, your constraints, your service target. The assumptions are inspectable, and changing them changes the recommendation in ways you can follow. Every recommendation ships with its reasoning — the tradeoff table, the service and risk figures, the comparison against the alternatives — so you can sanity-check it against what you know about your business that the numbers don't capture.

This is the difference between a tool that replaces judgment and one that arms it. A planner who can see *why* 1,500 beats 2,000 can confidently place 1,500 — or, knowing a promotion is coming that the data hasn't seen yet, deliberately override to 2,000 and understand exactly what tradeoff they're accepting. The engine does the heavy simulation and bookkeeping no human can do by hand; the human supplies the context and owns the call. That's the right division of labor, and it's the one that gives operations planners their leverage back.

15 Conclusion: Test the Waters Before the Order Goes Out

Forecast accuracy is a worthy pursuit, but it is not the finish line, and treating it as one has led a generation of teams to optimize the wrong thing. The same forecast error implies completely different correct actions depending on margin, stockout cost, holding cost, obsolescence, and service expectations. What determines whether you make money is the quality of the *decision*, not the tidiness of the prediction feeding it.

Order Otter takes decision quality as the goal directly. It treats every order as the economic bet it is, generates feasible candidate orders, floats them through the same river of plausible futures, scores each outcome in dollars and service, and — through RaftRank™ — recommends the order with the best tradeoff for your economics and your risk tolerance. The recommendation is defensible, the assumptions are yours, and the planner stays in control.

The metaphor is the method. Before you commit cash to an order you can't take back, you can test the waters: see how each choice behaves across the futures you're likely to face, and place the one that comes through best. That's a better way to order — and it's available on your next decision, not after a six-month integration.

16 Appendix: Simple Pseudocode

The core loop, stripped to its essentials. In practice the candidate set can be searched adaptively (Section 12) and scoring includes risk measures beyond the mean, but the shape is exactly this:

```
# Order Otter — core simulation-optimization loop (RaftRank™)


def recommend_order(state, economics, constraints, objective):
    # 1. Build feasible candidate orders (respect MOQ, case pack, budget, capacity)
    candidates = feasible_candidates(constraints)

    # 2. Generate ONE shared set of futures — common random numbers
    scenarios = generate_scenarios(
        state.demand_model,
        state.leadtime_model,
        n=NUM_SCENARIOS,
    )

    results = {}
    for q in candidates:
        outcomes = []
        for s in scenarios:  # same scenarios for every candidate
            sim = simulate_inventory_flow(
                order=q,
                start=state.on_hand,
                open_po=state.open_po,
                scenario=s,
                horizon=state.horizon,
            )
            outcomes.append(score_economics(sim, economics))

        # 3. Aggregate this candidate across all futures
        #    (mean profit, service, stockout %, avg leftover, downside risk / CVaR)
        results[q] = aggregate(outcomes)

    # 4. Rank by the business objective + risk tolerance, then recommend
```

```
    ranked = raftrank(results, objective)
    return ranked.best, ranked.explanation, results
```

simulate_inventory_flow walks the horizon week by week — arrivals after lead time, demand drawdown, captured sales vs stockouts, holding cost on unsold units, and end-of-horizon leftover at salvage value. *raftrank* orders candidates by the chosen economic objective and attaches the reasoning that makes the recommendation explainable.
