📦 Assignment 11 — Warehouse Layout Design Optimization

What this notebook delivers.
This notebook implements the two core tasks of Assignment 11: (i) *Product Allocation and Functional Area Sizing* (min. annual handling + storage cost) and (ii) *Warehouse Block Layout Design* for I-, L-, and U-shaped patterns. The workflow: load the input data → compute EOQ/dwell → build an optimization model for product–flow assignment and area sizing → solve and report → formulate a block-layout model with Muther SLP weights → optimize and visualize each footprint.

Key assumptions (aligned with the brief).

- Each product is assigned to **exactly one** of four flows: Cross-dock (CD), Reserve (R), Reserve→Forward (RF), or Forward (F).
- Handling and storage costs per unit load follow Tables 11.2–11.3; yearly carrying rate is 10%. Products 3 and 6 have non-zero reserve dwell in RF.
- Break-bulk applies on RF moves (Reserve→Forward), so handling includes an extra breakdown step and smaller storage units in forward.
- Space inventory is measured in m^2 with **1 vertical level** per functional area (Table 11.5). Total space ≤ 100,000 m^2 and functional-area bounds follow Table 11.4.
- Optimization uses a standard MILP/MIQCP formulation (PuLP/Gurobi per the code) with space and policy constraints; layout uses Muther SLP weights (Table 11.8) converted to quantitative flows and includes non-overlap and footprint constraints.
- Visualizations are for communication/sanity-check and do not replace feasibility constraints.

How to read this notebook.
For every code cell, a one-paragraph commentary (just above the cell) explains what the block does and lists any helper functions it defines. At the end you'll find a short results synopsis and managerial takeaways tailored to this notebook's outputs.

What this cell does — Data loading and preprocessing. It imports required libraries (e.g., pandas, numpy) and reads or constructs the input tables (demands, handling/storage costs, space per unit-load, functional-area bounds). It may also standardize column names and compute derived parameters such as flow-specific costs and per-product space needs.

In [1]:
```
# Imports + all input data + tunable constants.

from itertools import product as cartesian_product
import math
import pandas as pd
from IPython.display import display  # to show tables nicely in notebooks

# ---- Tunable constants (assumptions) ----
```

```
STAGING_DAYS_CD = 3   # average CD staging time in days (Little's Law for cross-
GAMMA_RF = 5          # RF explosion factor: 1 reserve load -> 5 smaller forward

# ---- Sets ----
products = [1, 2, 3, 4, 5, 6]
flows = [1, 2, 3, 4]  # 1=CD, 2=R, 3=RF, 4=F
flow_name = {1: "CD", 2: "R", 3: "RF", 4: "F"}
areas = ["CD", "R", "F"]

# ---- Table 11.1 ----
# Treat "Annual demand" as unit loads per year (consistent with the cost tables)
D = {1: 10000, 2: 15000, 3: 25000, 4: 2000, 5: 1500, 6: 95000}   # annual demand
S = {1: 50, 2: 50, 3: 50, 4: 50, 5: 50, 6: 150}                   # order cost ($/
P = {1: 500, 2: 650, 3: 350, 4: 250, 5: 225, 6: 150}              # price per unit
c_rate = {i: 0.10 for i in products}                              # carrying cost
space = {1: 10, 2: 15, 3: 25, 4: 10, 5: 12, 6: 13}                # m² per unit lo

# RF reserve dwell fraction (only relevant if assigned to RF)
# NOTE: As stated, only 3 (20%) and 6 (100%) have nonzero reserve dwell when in
alpha_reserve_RF = {1: 0.0, 2: 0.0, 3: 0.20, 4: 0.0, 5: 0.0, 6: 1.0}

# ---- Table 11.2 (handling cost per unit load) ----
handling_table = {
    1: {1: 0.0707, 2: 0.0849, 3: 0.1061, 4: 0.0778},
    2: {1: 0.0203, 2: 0.2023, 3: 0.2023, 4: 0.2023},
    3: {1: 0.0267, 2: 0.0420, 3: 0.0054, 4: 0.0481},
    4: {1: 0.3354, 2: 0.5590, 3: 1.0062, 4: 0.0671},
    5: {1: 0.4083, 2: 0.6804, 3: 1.2248, 4: 0.8165},
    6: {1: 0.0726, 2: 0.0871, 3: 0.1088, 4: 0.0798},
}
c_handling = {(i, f): handling_table[i][f] for i in products for f in flows}

# ---- Table 11.3 (storage cost per unit load per year) ----
storage_table = {
    1: {1: 20, 2: 5,  3: 10, 4: 15},
    2: {1: 15, 2: 5,  3: 10, 4: 10},
    3: {1: 4,  2: 20, 3: 1,  4: 9},
    4: {1: 5,  2: 4,  3: 5,  4: 1},
    5: {1: 15, 2: 25, 3: 45, 4: 30},
    6: {1: 20, 2: 5,  3: 10, 4: 15},
}
c_storage = {(i, f): storage_table[i][f] for i in products for f in flows}

# ---- Table 11.4 + total capacity ----
area_bounds = {"CD": (0, 15000), "R": (35000, 75000), "F": (35000, 75000)}
TOTAL_WAREHOUSE_CAP = 100_000  # "up to"

# ---- Table 11.5 (levels) ----
levels = {"CD": 1, "R": 1, "F": 1}
```

What this cell does — EOQ and dwel-ltime calculations. It computes the Economic Order Quantity (EOQ) and implied average on-hand inventory / dwell times using the given order costs, demand levels, and carrying rate. These feed the model's storage space and annual storage cost calculations by estimating how much inventory sits in each area on average.

In [2]:
```
# EOQ and derived average on-hand/dwell (per unit load)
def eoq(i: int) -> float:
```

```
    # H ($/load-year) = price * carrying rate
    H = P[i] * c_rate[i]
    return math.sqrt(2 * D[i] * S[i] / H)

def avg_on_hand(i: int) -> float:
    # Standard EOQ assumption: average cycle stock = Q/2
    return 0.5 * eoq(i)

def dwell_years(i: int) -> float:
    return avg_on_hand(i) / D[i]

def dwell_days(i: int) -> float:
    return dwell_years(i) * 365.0

df_eoq = pd.DataFrame(
    [{"Product": i,
      "EOQ (loads)": eoq(i),
      "Avg on-hand (loads)": avg_on_hand(i),
      "Avg dwell (days)": dwell_days(i)} for i in products]
)
display(df_eoq.round(3))
```

	Product	EOQ (loads)	Avg on-hand (loads)	Avg dwell (days)
0	1	141.421	70.711	2.581
1	2	151.911	75.955	1.848
2	3	267.261	133.631	1.951
3	4	89.443	44.721	8.162
4	5	81.650	40.825	9.934
5	6	1378.405	689.202	2.648

What this cell does — Cost/space parameter assembly. It consolidates Tables 11.2–11.4 (handling and storage costs; area bounds) into convenient Python structures. If break-bulk on RF is modeled, the parameters reflect that RF entails additional handling in forward and potentially larger space usage at the pick face due to smaller units.

In [3]:
```
def avg_load_equivalents(i: int, f: int) -> float:
    """
    Average 'load equivalents' for product i in flow f,
    used for both storage cost scaling and m² consumption.
    """
    if f == 1:  # Cross-dock: Little's Law with staging days
        return D[i] * (dwell_days(i) / 365.0)
    elif f in (2, 4):  # Pure Reserve or pure Forward
        return eoq(i) / 2
    elif f == 3:  # RF: split between Reserve and Forward with explosion in forw
        alpha = alpha_reserve_RF[i]
        base = eoq(i)/2
        # total load-equivalents stored across areas
        return alpha * base + GAMMA_RF * (1 - alpha) * base
    else:
        raise ValueError("Unknown flow")

def area_use_by_area(i: int, f: int) -> dict:
```

```python
    """
    m² used by product i in each functional area for flow f.
    """
    s = space[i] / levels["R"]  # levels are 1 here, kept for completeness
    use = {"CD": 0.0, "R": 0.0, "F": 0.0}
    if f == 1:
        use["CD"] = avg_load_equivalents(i, f) * s
    elif f == 2:
        use["R"]  = avg_load_equivalents(i, f) * s
    elif f == 4:
        use["F"]  = avg_load_equivalents(i, f) * s
    elif f == 3:
        base  = avg_on_hand(i)
        alpha = alpha_reserve_RF[i]
        I_R = alpha * base
        I_F_equiv = GAMMA_RF * (1 - alpha) * base
        use["R"] = I_R * s
        use["F"] = I_F_equiv * s
    return use

def area_coeff(i: int, f: int, a: str) -> float:
    """Coefficient for area capacity constraint: m² used in area a if product i
    return area_use_by_area(i, f)[a]

def annual_cost_for_product(i: int, f: int) -> dict:
    # Handling cost scales with annual throughput; table gives $ per unit load i
    handling = D[i] * c_handling[(i, f)]
    # Storage cost scales with average load equivalents; table gives $ per load-
    storage  = avg_load_equivalents(i, f) * c_storage[(i, f)]
    return {"handling": handling, "storage": storage, "total": handling + storag

def compute_area_consumption(assignment: dict) -> dict:
    cons = {"CD": 0.0, "R": 0.0, "F": 0.0}
    for i, f in assignment.items():
        use = area_use_by_area(i, f)
        for a in areas:
            cons[a] += use[a]
    return cons

def choose_area_sizes(cons: dict):
    """
    Given consumption by area, pick the smallest feasible area sizes satisfying:
      - per-area lower/upper bounds (Table 11.4)
      - total cap <= TOTAL_WAREHOUSE_CAP
    """
    sizes = {}
    for a in areas:
        lb, ub = area_bounds[a]
        need = cons[a]
        size_a = max(lb, need)
        if size_a > ub + 1e-9:
            return False, None  # infeasible requested consumption vs bounds
        sizes[a] = size_a

    # Check overall cap
    if sum(sizes.values()) > TOTAL_WAREHOUSE_CAP + 1e-9:
        return False, None

    return True, sizes
```

```python
def evaluate_assignment(assignment: dict):
    cons = compute_area_consumption(assignment)
    feas, sizes = choose_area_sizes(cons)
    rows = []
    total_cost = 0.0
    for i, f in assignment.items():
        comp = annual_cost_for_product(i, f)
        rows.append({
            "Product": i,
            "Flow": flow_name[f],
            "Handling Cost": comp["handling"],
            "Storage Cost":  comp["storage"],
            "Total Cost":    comp["total"],
        })
        total_cost += comp["total"]
    df = pd.DataFrame(rows)
    return feas, sizes, total_cost, df, cons
```

What this cell does — Model scaffolding. It sets up indices/sets (products, flows, areas), defines constants (e.g., total area cap 100,000 m²), and prepares helper mappings for readability in the optimization model. This cell is usually “glue” before declaring decision variables.

In [4]:

```python
def compute_area_consumption(assignment:dict) -> dict:
    cons = {"CD":0.0, "R":0.0, "F":0.0}
    for i, f in assignment.items():
        use = area_use_by_area(i,f)
        for a in areas:
            cons[a] += use[a]
    return cons

def choose_area_sizes(cons:dict):
    sizes = {}
    total = 0.0
    for a in areas:
        lb, ub = area_bounds[a]
        need = cons[a]
        size_a = max(lb, need)
        if size_a > ub + 1e-9:
            return None
        sizes[a] = size_a
        total += size_a
    if total > TOTAL_WAREHOUSE_CAP + 1e-9:
        return None
    return sizes

def evaluate_assignment(assignment:dict):
    cons = compute_area_consumption(assignment)
    sizes = choose_area_sizes(cons)
    if sizes is None:
        return False, None, float("inf"), None, None

    rows = []
    total_cost = 0.0
    for i, f in assignment.items():
        comp = annual_cost_for_product(i,f)
        rows.append({
            "Product": i,
```

```
            "Flow": flow_name[f],
            "Handling Cost": comp["handling"],
            "Storage Cost":  comp["storage"],
            "Total Cost":    comp["total"]
        })
        total_cost += comp["total"]

    df = pd.DataFrame(rows)
    return True, sizes, total_cost, df, cons
```

What this cell does — Optimization model (PuLP/Gurobi/Pyomo). It declares decision variables for (i) binary product-to-flow assignment (each product must choose exactly one of CD, R, RF, or F) and (ii) continuous area sizes for Cross-dock, Reserve, and Forward. The objective minimizes total annual cost (handling + storage), and constraints enforce area bounds (Table 11.4) and total space limits, plus any policy constraints (e.g., vertical levels). It sets solver options (e.g., time limit, threads, acceptable MIP gap) and calls `.solve()` / `.optimize()`. It captures the status (Optimal/Feasible/Timed-out), the objective value (annual cost), and the chosen assignment and area sizes for downstream reporting.It parses the solution and prints the chosen flow for each product (CD/R/RF/F) and the area sizes for Cross-dock, Reserve, and Forward. This is where you verify all functional area bounds are respected and each product is assigned to exactly one flow.

In [5]:
```
try:
    !pip install pulp
    import pulp as pl

    m = pl.LpProblem("ForwardReserve_AreaSizing", pl.LpMinimize)

    # Variables
    x = pl.LpVariable.dicts("x", (products, flows), lowBound=0, upBound=1, cat=p
    A = pl.LpVariable.dicts("A", areas, lowBound=0, cat=pl.LpContinuous)

    # Objective: sum_i sum_f x_if * (handling_i,f + storage_i,f)
    m += pl.lpSum(
        x[i][f] * (
            D[i] * c_handling[(i, f)] +
            avg_load_equivalents(i, f) * c_storage[(i, f)]
        )
        for i in products for f in flows
    )

    # Each product assigned to exactly one flow
    for i in products:
        m += pl.lpSum(x[i][f] for f in flows) == 1, f"assign_{i}"

    # Area capacity constraints: for each area a,
    #   sum_i sum_f x_if * area_coeff(i,f,a) <= A_a
    for a in areas:
        m += pl.lpSum(x[i][f] * area_coeff(i, f, a)
                      for i in products for f in flows) <= A[a], f"area_cap_{a}

    # Area lower/upper bounds
    for a in areas:
        lb, ub = area_bounds[a]
        m += A[a] >= lb, f"lb_{a}"
```

```
        m += A[a] <= ub, f"ub_{a}"

    # Total building cap
    m += pl.lpSum(A[a] for a in areas) <= TOTAL_WAREHOUSE_CAP, "total_cap"

    # Solve
    _ = m.solve(pl.PULP_CBC_CMD(msg=False))

    # Report
    status = pl.LpStatus[m.status]
    print("Solver status:", status)
    if status != "Optimal":
        # Fall back: show best found (if any)
        pass

    # Extract solution
    sol_assign = {i: max(flows, key=lambda f: pl.value(x[i][f])) for i in produc
    sol_sizes  = {a: pl.value(A[a]) for a in areas}

    feasible, sized, tot_cost, df_cost, cons = evaluate_assignment(sol_assign)

    print("\n=== Optimal product-to-flow assignment ===")
    display(pd.DataFrame(
        [{"Product": i, "Assigned Flow": flow_name[sol_assign[i]]} for i in prod
    ).sort_values("Product"))

    print("\n=== Area consumption (m²) from assignment ===")
    display(pd.DataFrame([cons]).T.rename(columns={0: "Consumption (m²)"}))

    print("\n=== Chosen area sizes (m²) ===")
    display(pd.DataFrame([sol_sizes]).T.rename(columns={0: "Size (m²)"}))

    print("\n=== Annual cost breakdown ===")
    display(df_cost.sort_values("Product").reset_index(drop=True))
    print(f"\nTotal annual cost: {tot_cost:,.2f} $")

except Exception as e:
    print("PuLP not available or MILP solve skipped. Details:", e)
```

```
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: pulp in ./.local/lib/python3.10/site-packages (3.
3.0)
Solver status: Optimal

=== Optimal product-to-flow assignment ===
```

	Product	Assigned Flow
0	1	R
1	2	CD
2	3	RF
3	4	F
4	5	CD
5	6	R

```
=== Area consumption (m²) from assignment ===
```

	Consumption (m^2)
CD	1629.229737
R	10334.891575
F	13810.275691

```
=== Chosen area sizes (m²) ===
```

	Size (m^2)
CD	1629.2297
R	35000.0000
F	35000.0000

```
=== Annual cost breakdown ===
```

	Product	Flow	Handling Cost	Storage Cost	Total Cost
0	1	R	849.00	353.553391	1202.553391
1	2	CD	304.50	1139.331788	1443.831788
2	3	RF	135.00	561.248608	696.248608
3	4	F	134.20	44.721360	178.921360
4	5	CD	612.45	612.372436	1224.822436
5	6	R	8274.50	3446.012188	11720.512188

```
Total annual cost: 16,466.89 $
```

Results summary and managerial advice

What to look for in your results.

- **Product → flow assignments:** Each product must be assigned to exactly one of {CD, R, RF, F}. Check whether high-demand items land in RF/F and whether any truly cross-dockable items go to CD.
- **Functional area sizes:** Cross-dock, Reserve, and Forward areas should sit within the lower/upper bounds (Table 11.4) and total ≤ 100,000 m^2.
- **Objective value:** Total annual cost = handling + storage. If you ran multiple parameter sets, compare totals to see sensitivity.

Managerial guidance.

- **Right-size Reserve vs. Forward.** Ensure Reserve and Forward capacities align with the mix of RF/R assignments. If high-demand items go **RF**, verify that forward pick faces, replenishment labor, and break-bulk stations can support the flow without bottlenecks.
- **Use Cross-dock selectively.** Items routed to **CD** should have short dwell, predictable arrivals, and outbound matching; otherwise the storage flows (R/F) will usually be cheaper.

- **Break-bulk implications.** For any product on **RF**, confirm sufficient case/carton storage and handling lanes in Forward; RF saves Reserve space but increases piece-handling touches.

11.5 Warehouse Block Layout Design (6-Department Starter)

Scope (Robustness target): Handle at least these 6 departments

1. Inbound Dock, 2) Receiving/Staging, 3) Pallet Reserve Storage (Bulk),
2. Packing / Wrap / Banding, 5) Outbound Staging (Parcel + 2-Man combined),
3. Shipping Dock.

Goal: For three layout patterns (I, L, U), place rectangular departments inside a bounding facility, respect fixed dock positions, avoid overlap, and **minimize total flow-weighted rectilinear (Manhattan) travel** between department centroids.

Modeling approach (MILP):

- We translate Muther SLP adjacency codes (E/A/I/O/U) into **quantitative flows** (4/3/2/1/0). This follows the common practice of mapping qualitative closeness to pairwise interaction weights, consistent with the literature (e.g., Heragu et al., 2005).
- To keep areas **exact** yet linear, we precompute a small set of **aspect-ratio options** for each department. A single binary choice selects one (width, height) combination with `width × height = target_area`.
- **Non-overlap** is enforced with the classic disjunctive Big-M formulation using four binaries per pair (left/right/above/below) and `L+R+B+T = 1`.
- **Distance** is rectified via absolute-value linearization: introduce `dx_ij, dy_ij` with `dx_ij ≥ x_i − x_j` and `dx_ij ≥ x_j − x_i` (similarly for `dy_ij`); objective is $\sum f_{ij} (dx_{ij} + dy_{ij})$.

Patterns (dock placement constraints):

- **I-shaped:** Inbound dock fixed to the **left wall (center)**; Shipping dock fixed to the **right wall (center)** of a long rectangle.
- **L-shaped:** Inbound fixed to the **bottom wall (left wing)**; Shipping fixed to the **right wall (top wing)** of a moderately rectangular block (we emulate an L-flow by orthogonal dock placement).
- **U-shaped:** Both docks on the **same wall (left)** at **lower** (Inbound) and **upper** (Shipping) segments, encouraging a U-flow.

Assessment settings:

- Gurobi requested; we set `MIPGap = 0.05` as a target.
- Clear visualizations using Matplotlib.

Reference:

Heragu, S. S., Du, L., Mantel, R. J., & Schuur, P. C. (2005). *Mathematical model for*

warehouse design and product allocation. IJPR, 43(2), 327–38.
https://doi.org/10.1080/00207540412331285841

In [6]:
```
# If you use a different solver, adapt here; Gurobi requested by spec.
try:
    !pip install gurobipy
    # --- IGNORE ---
    import gurobipy as gp
    from gurobipy import GRB
except Exception as e:
    raise RuntimeError(
        "This notebook is written for Gurobi (gurobipy). "
        "Please install & license Gurobi to run. Error: {}".format(e)
    )

import math
import itertools
import matplotlib.pyplot as plt
from dataclasses import dataclass
```

```
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: gurobipy in ./.local/lib/python3.10/site-packages
(12.0.3)
```

Data: Areas & SLP mapping

Areas (m^2) from Table 11.6 (restricted to the 6-department subset; Outbound Staging is aggregated):

- Inbound Dock: 2,640
- Receiving/Staging: 5,280
- Pallet Reserve Storage (Bulk): 46,340
- Packing / Wrap / Banding: 3,520
- Outbound Staging (Parcel + 2-Man): 3,520 + 5,280 = **8,800**
- Shipping Dock: 3,520

SLP mapping to flows:
We convert Muther SLP codes to numeric flows as:
`E=4, A=3, I=2, O=1, U=0`.
For the aggregated **Outbound Staging**, where two original rows existed (Parcel & 2-Man), we take the **max** code against any counterpart (equivalent to assuming aggregation preserves the strongest interaction).
This yields a chain of strong interactions (a classic flow):
Inbound ↔ Receiving (E), Receiving ↔ Bulk (I), Bulk ↔ Packing (E), Packing ↔ Staging (E), Staging ↔ Shipping (E).
All other pairs default to lower or zero flow if their codes were U/O in the SLP.

We will use these flows directly as **unit-load intensities**; with unit transport cost per meter, the objective becomes total flow-weighted distance.

In [7]:
```
import itertools
```

```python
# ------- Departments & areas (m^2) : full list from Table 11.6 -------
AREAS = {
    "Cross-Dock": 3520,
    "Empty Pallets & Dunnage": 880,
    "Inbound Dock": 2640,
    "Maintenance & Battery Charge": 1320,
    "Outbound Staging – 2-Man Delivery": 5280,
    "Outbound Staging – Parcel": 3520,
    "Oversize/Non-Standard Storage": 2640,
    "Packing / Wrap / Banding": 3520,
    "Pallet Reserve Storage (Bulk)": 46340,
    "QA & Technical Test": 1760,
    "Receiving/Staging": 5280,
    "Returns & WEEE": 2640,
    "Shipping Dock": 3520,
    "Spare Parts & Accessories Cage": 440,
}

DEPTS = list(AREAS.keys())

# ------- SLP codes (Table 11.7) → list of triples (i, j, code) -------
# Only non-U pairs are listed explicitly; U will be assigned a small baseline we
slp_nonU = [
    # Receiving/Staging row
    ("Inbound Dock", "Receiving/Staging", "E"),
    ("Receiving/Staging", "Cross-Dock", "A"),
    ("Receiving/Staging", "QA & Technical Test", "A"),
    ("Receiving/Staging", "Pallet Reserve Storage (Bulk)", "I"),

    # QA & Technical Test row
    ("Receiving/Staging", "QA & Technical Test", "A"),

    # Cross-Dock row
    ("Cross-Dock", "Outbound Staging – Parcel", "A"),
    ("Cross-Dock", "Outbound Staging – 2-Man Delivery", "A"),
    ("Cross-Dock", "Shipping Dock", "A"),

    # Pallet Reserve Storage (Bulk) row
    ("Pallet Reserve Storage (Bulk)", "Packing / Wrap / Banding", "E"),

    # Oversize/Non-Standard row (no non-U entries listed)

    # Packing / Wrap / Banding row
    ("Packing / Wrap / Banding", "Pallet Reserve Storage (Bulk)", "E"),
    ("Packing / Wrap / Banding", "Outbound Staging – Parcel", "E"),
    ("Packing / Wrap / Banding", "Outbound Staging – 2-Man Delivery", "E"),
    ("Packing / Wrap / Banding", "Returns & WEEE", "O"),
    ("Packing / Wrap / Banding", "Empty Pallets & Dunnage", "O"),

    # Outbound Staging – Parcel row
    ("Outbound Staging – Parcel", "Shipping Dock", "E"),

    # Outbound Staging – 2-Man Delivery row
    ("Outbound Staging – 2-Man Delivery", "Shipping Dock", "E"),

    # Shipping Dock row (already covered via E’s above)

    # Empty Pallets & Dunnage row
    ("Empty Pallets & Dunnage", "Receiving/Staging", "I"),
    ("Empty Pallets & Dunnage", "Packing / Wrap / Banding", "O"),
```

```
    # Maintenance & Battery Charge row
    ("Maintenance & Battery Charge", "Pallet Reserve Storage (Bulk)", "O"),

    # Returns & WEEE row
    ("Returns & WEEE", "QA & Technical Test", "I"),
    ("Returns & WEEE", "Packing / Wrap / Banding", "O"),

    # Spare Parts & Accessories Cage row
    # (all U by the table; no entries here)
]

# If you *do* have any "X" pairs in your data, list them here (we won't use nega
undesirable_pairs_codes = []  # the provided table has no "X" entries

# ------- Map SLP legend to numeric weights -------
weight_map = {"E": 4, "A": 3, "I": 2, "O": 1}  # U handled via epsilon baseline

# ------- Convert to a symmetric flow matrix -------
EPSILON = 0.05   # small baseline so U pairs aren't totally unconstrained
ALPHA   = 1.0    # global scale for SLP weights

flows = {}
for i, j in itertools.combinations(DEPTS, 2):
    flows[(i, j)] = EPSILON

for a, b, code in slp_nonU:
    if a not in AREAS or b not in AREAS:
        raise ValueError(f"Unknown department in SLP pair: ({a}, {b})")
    i, j = sorted([a, b], key=lambda s: DEPTS.index(s))
    flows[(i, j)] = EPSILON + ALPHA * weight_map[code]

# Convenience: area totals + a quick report
total_area = sum(AREAS[d] for d in DEPTS)
num_pairs = len(flows)
nonzero_pairs = sum(1 for v in flows.values() if v > EPSILON)

print(f"Total area: {total_area:,.0f} m² (envelope should be ≳ this)")
print(f"Pairs: {num_pairs}, nonzero flows (E/A/I/O): {nonzero_pairs}, baseline-o
print(f"EPSILON (baseline): {EPSILON}, SLP scale ALPHA: {ALPHA}")

# Spot-check some key pairs:
for key in [
    ("Inbound Dock", "Receiving/Staging"),
    ("Packing / Wrap / Banding", "Outbound Staging – Parcel"),
    ("Outbound Staging – 2-Man Delivery", "Shipping Dock"),
    ("Pallet Reserve Storage (Bulk)", "Packing / Wrap / Banding"),
    ("QA & Technical Test", "Returns & WEEE"),
    ("Spare Parts & Accessories Cage", "Receiving/Staging"),
]:
    i, j = sorted(key, key=lambda s: DEPTS.index(s))
    print(f"{key[0]} <-> {key[1]} : flow {flows[(i,j)]:.2f}")
```

```
Total area: 83,300 m² (envelope should be ≳ this)
Pairs: 91, nonzero flows (E/A/I/O): 17, baseline-only pairs: 74
EPSILON (baseline): 0.05, SLP scale ALPHA: 1.0
Inbound Dock <-> Receiving/Staging : flow 4.05
Packing / Wrap / Banding <-> Outbound Staging – Parcel : flow 4.05
Outbound Staging – 2-Man Delivery <-> Shipping Dock : flow 4.05
Pallet Reserve Storage (Bulk) <-> Packing / Wrap / Banding : flow 4.05
QA & Technical Test <-> Returns & WEEE : flow 2.05
Spare Parts & Accessories Cage <-> Receiving/Staging : flow 0.05
```

Facility footprint and dock placements (I / L / U)

We approximate each pattern with a **single rectangular envelope** of equal total area for fairness, differing only by **dock placement**:

- Total required area (departments only): 70,100 m^2.
  We add ~20% space for aisles/circulation = **84,120 m^2**.

We choose three envelopes near this area:

- **I-shaped (long):** `W=420 m, H=200 m` → 84,000 m^2 (AR ≈ 2.1)
- **L-shaped (emulated via orthogonal dock placement):** `W=360 m, H=234 m` → 84,240 m^2 (AR ≈ 1.54)
- **U-shaped (more square):** `W=290 m, H=290 m` → 84,100 m^2 (AR ≈ 1.0)

Dock placement rules (implemented by fixing (x,y) relationships):

- Left wall: `x = 0.5 * width`
- Right wall: `x = W - 0.5 * width`
- Bottom wall: `y = 0.5 * height`
- Top wall: `y = H - 0.5 * height`
- Center on a wall: use the wall rule above and set the other coordinate to `H/2` or `W/2` as needed.

Specifics:

- **I-shaped:** Inbound @ **left-wall center**, Shipping @ **right-wall center**
- **L-shaped:** Inbound @ **bottom-wall** (left wing: `x = 0.25W`), Shipping @ **right-wall** (upper wing: `y = 0.75H`)
- **U-shaped:** Both on **bottom wall**; Inbound @ lower (`y = 0.25H`), Shipping @ upper (`y = 0.75H`)

In [8]:
```python
# === Bootstrap Helpers: run this once before calling build_and_solve_two_phase
import math
from dataclasses import dataclass

# 1) LayoutSpec (if missing)
if "LayoutSpec" not in globals():
    @dataclass
    class LayoutSpec:
        name: str
        W: float
        H: float
```

```
        dock_rules: dict

# 2) make_aspect_options (if missing) – moderate, well-behaved aspect set
# if "make_aspect_options" not in globals():
#     def make_aspect_options(area):
#         s = math.sqrt(area)
#         r = 1.5  # aspect ratio bound
#         # (w,h) options
#         return [
#             (s, s),                 # ~square
#             (r*s, s/r),             # wide
#             (s/r, r*s),             # tall
#         ]
if "make_aspect_options" not in globals():
    def make_aspect_options(area, ratios=(0.5, 0.67, 0.8, 1.0, 1.25, 1.5, 2.0, 2
                            grid=0.5, max_opts=8):
        opts = []
        for r in ratios:
            w = (area * r) ** 0.5
            h = area / w
            # snap to grid
            w = round(w / grid) * grid
            h = round(h / grid) * grid
            if w > 0 and h > 0:
                opts.append((w, h))
            # add rotated if not ~square
            if abs(r - 1.0) > 1e-9:
                opts.append((h, w))

        # deduplicate and favor low perimeters (compact shapes)
        uniq = {}
        for (w, h) in opts:
            key = (round(w, 3), round(h, 3))
            uniq[key] = None
        cand = list(uniq.keys())
        cand.sort(key=lambda wh: 2 * (wh[0] + wh[1]))  # perimeter proxy

        # cap the menu size
        return cand[:max_opts]
    """
    Build a small-but-rich set of (w,h) for a given area.
    - ratios: target w/h aspect ratios (include >1 and <1)
    - grid:   snap to this grid (meters)
    - max_opts: cap options per dept to control model size
    """
# 3) enforce_dock_rules (if missing)
if "enforce_dock_rules" not in globals():
    def enforce_dock_rules(mdl, layout, x, y, w, h, W, H, sep=0.0):
        # Fix inbound/outbound to walls based on layout.dock_rules
        for d, rule in layout.dock_rules.items():
            if "x_rule" in rule:
                if rule["x_rule"] == "left":
                    mdl.addConstr(x[d] == 0.5 * w[d] + sep)
                elif rule["x_rule"] == "right":
                    mdl.addConstr(x[d] == W - 0.5 * w[d] - sep)
            if "y_rule" in rule:
                if rule["y_rule"] == "bottom":
                    mdl.addConstr(y[d] == 0.5 * h[d] + sep)
                elif rule["y_rule"] == "top":
                    mdl.addConstr(y[d] == H - 0.5 * h[d] - sep)
```

```python
# 4) IIS helper (only if you need to debug infeasibility)
if "try_write_iis" not in globals():
    def try_write_iis(mdl, tag="[IIS]"):
        try:
            mdl.computeIIS()
            mdl.write("model.ilp")
            mdl.write("model.ilp.mps")
            mdl.write("model.iis")
            print(f"{tag} Wrote model.ilp / model.ilp.mps / model.iis")
        except Exception as e:
            print(f"[IIS] Failed to compute IIS: {e}")

def infeasible_payload(layout, DEPTS, W, H):
    return {
        "name": layout.name,
        "status": gp.GRB.INFEASIBLE,
        "W": W, "H": H,
        "x": {i: None for i in DEPTS},
        "y": {i: None for i in DEPTS},
        "w": {i: None for i in DEPTS},
        "h": {i: None for i in DEPTS},
        "obj": None,
        "gap": None,
        "bound": None,
    }
```

What this cell does — Layout model variables and constraints. It declares rectangle variables for each department (x, y, width, height) within the selected footprint (I-, L-, or U-shaped). Constraints ensure non-overlap, stay-within-boundary, target-area matching, and any fixed positions (e.g., dock face). The model minimizes weighted travel (e.g., Manhattan distance) across department centroids or edges.

In [9]:

```python
from dataclasses import dataclass
import math
import itertools
import time
import gurobipy as gp
from gurobipy import GRB
import os
from concurrent.futures import ProcessPoolExecutor, as_completed

@dataclass
class LayoutSpec:
    name: str
    W: float
    H: float
    dock_rules: dict  # maps department -> {"x_rule": ..., "y_rule": ...}

def make_aspect_options(area):
    """Base aspect menu (we'll filter against layout later)."""
    if area >= 25000:
        ratios = (0.50, 0.75, 1.00, 1.25, 1.50, 2.00)
    elif area >= 6000:
        ratios = (0.67, 1.00, 1.50)
    else:
        ratios = (0.67, 1.00, 1.50)
    opts = []
```

```
    for r in ratios:
        h = math.sqrt(area * r)
        w = area / h
        opts.append((w, h))
    return opts

def _infeasible_result(layout, mdl):
    try:
        mdl.computeIIS()
        bad_constrs = [c.ConstrName for c in mdl.getConstrs() if c.IISConstr]
        bad_bounds  = [v.VarName for v in mdl.getVars() if v.IISLB or v.IISUB]
        print(f"[{layout.name}] IIS found. First conflicting constraints: {bad_c
        if bad_bounds:
            print(f"[{layout.name}] Vars with conflicting bounds (first 10): {ba
    except Exception as e:
        print(f"[{layout.name}] IIS computation skipped ({e}).")
    return {
        "obj": None,
        "W": layout.W, "H": layout.H,
        "x": {}, "y": {}, "w": {}, "h": {},
        "status": GRB.INFEASIBLE,
        "gap": None,
        "bound": None,
        "name": layout.name
    }

def build_and_solve_two_phase(layout,
                              miph_gap_target=0.01,
                              total_seconds=900,
                              phase1_seconds=300,
                              seed=42,
                              verbose=True,
                              threads=None,
                              sep=0.0):
    """Two-phase MIP with indicator disjunctions and aspect menus filtered by la
    gp.setParam("OutputFlag", 1 if verbose else 0)
    mdl = gp.Model(f"WarehouseLayout_{layout.name}")

    # ---------- Variables ----------
    x = {i: mdl.addVar(lb=0.0, ub=layout.W, name=f"x[{i}]") for i in DEPTS}
    y = {i: mdl.addVar(lb=0.0, ub=layout.H, name=f"y[{i}]") for i in DEPTS}

    # Build per-dept aspect menus; filter by layout size; if empty, repair to a
    aspects = {}
    for i in DEPTS:
        base = make_aspect_options(AREAS[i])
        filt = [(w, h) for (w, h) in base if (w <= layout.W + 1e-6 and h <= layo
        if not filt:
            # fallback: shrink tallest dimension to fit, keep area exact
            # try to cap h first
            h = min(layout.H * 0.98, math.sqrt(AREAS[i]))
            w = AREAS[i] / h
            if w > layout.W * 0.98:
                w = layout.W * 0.98
                h = AREAS[i] / w
            # still ensure within box
            h = min(h, layout.H * 0.98)
            w = min(w, layout.W * 0.98)
            filt = [(w, h)]
        aspects[i] = filt
```

```
z = {(i,k): mdl.addVar(vtype=GRB.BINARY, name=f"z[{i},{k}]")
     for i in DEPTS for k,_ in enumerate(aspects[i])}
w = {i: mdl.addVar(lb=0.0, ub=layout.W, name=f"w[{i}]") for i in DEPTS}
h = {i: mdl.addVar(lb=0.0, ub=layout.H, name=f"h[{i}]") for i in DEPTS}

for i in DEPTS:
    mdl.addConstr(gp.quicksum(z[(i,k)] for k in range(len(aspects[i]))) == 1
    mdl.addConstr(w[i] == gp.quicksum(aspects[i][k][0] * z[(i,k)] for k in r
    mdl.addConstr(h[i] == gp.quicksum(aspects[i][k][1] * z[(i,k)] for k in r
    mdl.addConstr(x[i] >= 0.5 * w[i], name=f"x_lb[{i}]")
    mdl.addConstr(x[i] <= layout.W - 0.5 * w[i], name=f"x_ub[{i}]")
    mdl.addConstr(y[i] >= 0.5 * h[i], name=f"y_lb[{i}]")
    mdl.addConstr(y[i] <= layout.H - 0.5 * h[i], name=f"y_ub[{i}]")

# ---------- Disjunction binaries (orientation is (i,j) with i<j) ----------
# Canonical pair order helper
def _canon(i, j):
    return (i, j) if DEPTS.index(i) < DEPTS.index(j) else (j, i)

# All undirected pairs (used for non-overlap disjunctions)
pair_all = [_canon(i, j) for i, j in itertools.combinations(DEPTS, 2)]

L = {}; R = {}; B = {}; T = {}
for (i, j) in pair_all:
    L[(i,j)] = mdl.addVar(vtype=GRB.BINARY, name=f"L[{i},{j}]")
    R[(i,j)] = mdl.addVar(vtype=GRB.BINARY, name=f"R[{i},{j}]")
    B[(i,j)] = mdl.addVar(vtype=GRB.BINARY, name=f"B[{i},{j}]")
    T[(i,j)] = mdl.addVar(vtype=GRB.BINARY, name=f"T[{i},{j}]")
    mdl.addConstr(L[(i,j)] + R[(i,j)] + B[(i,j)] + T[(i,j)] == 1, name=f"OR4

# ---- Build smaller edge sets for distances and connectivity ----
def _flow(i, j):  # flow value for canonical key
    return flows.get(_canon(i, j), 0.0)

# objective pairs = only those with non-zero flow
pair_objective = [p for p in pair_all if _flow(*p) > 0.0]

# contact/flow graph = sparse: each node connects to both docks + top-K flow
ROOT = "Inbound Dock"
SINK = "Shipping Dock"
TOPK = 3

pair_contact = set()
for d in DEPTS:
    if d != ROOT: pair_contact.add(_canon(ROOT, d))
    if d != SINK: pair_contact.add(_canon(SINK, d))
    nbrs = sorted((( _flow(d, e), e) for e in DEPTS if e != d), reverse=True
    for _, e in nbrs:
        pair_contact.add(_canon(d, e))
pair_contact = list(pair_contact)

# Pairs that need dx,dy (for objective or for contact equations)
pair_dxdy = list(set(pair_objective) | set(pair_contact))

# ---------- Non-overlap with Big-M (license-friendly MILP) ----------
Mx, My = layout.W, layout.H
for (i, j) in pair_all:
    mdl.addConstr(x[i] + 0.5*w[i] + sep <= x[j] - 0.5*w[j] + Mx * (1 - L[(i,
    mdl.addConstr(x[j] + 0.5*w[j] + sep <= x[i] - 0.5*w[i] + Mx * (1 - R[(i,
```

```python
        mdl.addConstr(y[i] + 0.5*h[i] + sep <= y[j] - 0.5*h[j] + My * (1 - B[(i,
        mdl.addConstr(y[j] + 0.5*h[j] + sep <= y[i] - 0.5*h[i] + My * (1 - T[(i,

    # ---------- Manhattan distance (only where needed) ----------
    dx, dy = {}, {}
    for (i, j) in pair_dxdy:
        dx[(i,j)] = mdl.addVar(lb=0.0, name=f"dx[{i},{j}]")
        dy[(i,j)] = mdl.addVar(lb=0.0, name=f"dy[{i},{j}]")
        mdl.addConstr(dx[(i,j)] >=  x[i] - x[j], name=f"dx1[{i},{j}]")
        mdl.addConstr(dx[(i,j)] >=  x[j] - x[i], name=f"dx2[{i},{j}]")
        mdl.addConstr(dy[(i,j)] >=  y[i] - y[j], name=f"dy1[{i},{j}]")
        mdl.addConstr(dy[(i,j)] >=  y[j] - y[i], name=f"dy2[{i},{j}]")
        # link half sizes when that axis is active (kept as valid lower bounds)
        mdl.addConstr(dx[(i,j)] >= 0.5*(w[i] + w[j]) - layout.W*(B[(i,j)] + T[(i
        mdl.addConstr(dy[(i,j)] >= 0.5*(h[i] + h[j]) - layout.H*(L[(i,j)] + R[(i

    # --------- Face contact detection (only on sparse contact graph) ---------
    eps = 1e-3
    MX, MY = layout.W, layout.H

    Htouch, Vtouch, Etouch = {}, {}, {}
    neighbors = {i: [] for i in DEPTS}

    for (i, j) in pair_contact:
        Htouch[(i,j)] = mdl.addVar(vtype=GRB.BINARY, name=f"Htouch[{i},{j}]")
        Vtouch[(i,j)] = mdl.addVar(vtype=GRB.BINARY, name=f"Vtouch[{i},{j}]")
        Etouch[(i,j)] = mdl.addVar(vtype=GRB.BINARY, name=f"Etouch[{i},{j}]")

        # vertical shared edge (left/right)
        mdl.addConstr(dx[(i,j)] >= 0.5*(w[i] + w[j]) - eps - MX*(1 - Htouch[(i,j
        mdl.addConstr(dx[(i,j)] <= 0.5*(w[i] + w[j]) + eps + MX*(1 - Htouch[(i,j
        mdl.addConstr(dy[(i,j)] <= 0.5*(h[i] + h[j]) - eps + MY*(1 - Htouch[(i,j
        mdl.addConstr(Htouch[(i,j)] <= L[(i,j)] + R[(i,j)], name=f"H_lr_link[{i}

        # horizontal shared edge (top/bottom)
        mdl.addConstr(dy[(i,j)] >= 0.5*(h[i] + h[j]) - eps - MY*(1 - Vtouch[(i,j
        mdl.addConstr(dy[(i,j)] <= 0.5*(h[i] + h[j]) + eps + MY*(1 - Vtouch[(i,j
        mdl.addConstr(dx[(i,j)] <= 0.5*(w[i] + w[j]) - eps + MX*(1 - Vtouch[(i,j
        mdl.addConstr(Vtouch[(i,j)] <= T[(i,j)] + B[(i,j)], name=f"V_tb_link[{i}

        # Etouch = OR(Htouch, Vtouch)  (exact)
        mdl.addConstr(Etouch[(i,j)] >= Htouch[(i,j)], name=f"E_ge_H[{i},{j}]")
        mdl.addConstr(Etouch[(i,j)] >= Vtouch[(i,j)], name=f"E_ge_V[{i},{j}]")
        mdl.addConstr(Etouch[(i,j)] <= Htouch[(i,j)] + Vtouch[(i,j)], name=f"E_l

        neighbors[i].append((i,j))
        neighbors[j].append((i,j))

    # --------- No isolated departments (degree >= 1) ---------
    for i in DEPTS:
        mdl.addConstr(gp.quicksum(Etouch[p] for p in neighbors[i]) >= 1, name=f"

    # --------- Connectivity: single-commodity flow on the sparse contact graph
    root = "Inbound Dock"
    N = len(DEPTS)
    F = {}
    for (i, j) in pair_contact:
        F[(i,j)] = mdl.addVar(lb=0.0, name=f"F[{i}->{j}]")
        F[(j,i)] = mdl.addVar(lb=0.0, name=f"F[{j}->{i}]")
        cap = N - 1
```

```
        mdl.addConstr(F[(i,j)] <= cap * Etouch[(i,j)], name=f"cap1[{i},{j}]")
        mdl.addConstr(F[(j,i)] <= cap * Etouch[(i,j)], name=f"cap2[{i},{j}]")

    for k in DEPTS:
        inflow  = gp.quicksum(F[(i,k)] for (i,j) in pair_contact if j == k) + \
                  gp.quicksum(F[(j,k)] for (i,j) in pair_contact if i == k)
        outflow = gp.quicksum(F[(k,j)] for (i,j) in pair_contact if i == k) + \
                  gp.quicksum(F[(k,i)] for (i,j) in pair_contact if j == k)
        if k == root:
            mdl.addConstr(outflow - inflow == N - 1, name=f"flow_root[{k}]")
        else:
            mdl.addConstr(inflow - outflow == 1,     name=f"flow_cons[{k}]")
        # ---------- Dock rules + orientation-safe locks ----------
    x_anchor = {d: None for d in DEPTS}   # 'left'/'right'/None
    y_anchor = {d: None for d in DEPTS}   # 'bottom'/'top'/None

    def apply_rule(axis, dept, rule):
        if axis == "x":
            if rule == "left":
                mdl.addConstr(x[dept] == 0.5 * w[dept],              name=f"dock
            elif rule == "right":
                mdl.addConstr(x[dept] == layout.W - 0.5 * w[dept],   name=f"dock
            elif rule == "center_x":
                mdl.addConstr(x[dept] == layout.W / 2.0,             name=f"dock
            elif isinstance(rule, (int, float)):
                mdl.addConstr(x[dept] == float(rule),                name=f"dock
        else:
            if rule == "bottom":
                mdl.addConstr(y[dept] == 0.5 * h[dept],              name=f"dock
            elif rule == "top":
                mdl.addConstr(y[dept] == layout.H - 0.5 * h[dept],   name=f"dock
            elif rule == "center_y":
                mdl.addConstr(y[dept] == layout.H / 2.0,             name=f"dock
            elif isinstance(rule, (int, float)):
                mdl.addConstr(y[dept] == float(rule),                name=f"dock

    # Apply rules AND support x_at / y_at
    for dept, rules in layout.dock_rules.items():
        if "x_rule" in rules: apply_rule("x", dept, rules["x_rule"])
        if "y_rule" in rules: apply_rule("y", dept, rules["y_rule"])
        if "x_at"  in rules:  mdl.addConstr(x[dept] == float(rules["x_at"]), nam
        if "y_at"  in rules:  mdl.addConstr(y[dept] == float(rules["y_at"]), nam

    # Helpers
    def set_lock_h(i, j, side):  # 'L' or 'R'
        key = (i, j) if (i, j) in L else (j, i)
        if key not in L: return
        if side == 'L':
            if key == (i, j): mdl.addConstr(L[key] == 1, name=f"lock_L[{i},{j}]"
            else:             mdl.addConstr(R[key] == 1, name=f"lock_R[{j},{i}]"
        else:
            if key == (i, j): mdl.addConstr(R[key] == 1, name=f"lock_R[{i},{j}]"
            else:             mdl.addConstr(L[key] == 1, name=f"lock_L[{j},{i}]"

    def set_lock_v(i, j, side):  # 'B' or 'T'
        key = (i, j) if (i, j) in B else (j, i)
        if key not in B: return
        if side == 'B':
            if key == (i, j): mdl.addConstr(B[key] == 1, name=f"lock_B[{i},{j}]"
            else:             mdl.addConstr(T[key] == 1, name=f"lock_T[{j},{i}]"
```

```
        else:
            if key == (i, j): mdl.addConstr(T[key] == 1, name=f"lock_T[{i},{j}]"
            else:             mdl.addConstr(B[key] == 1, name=f"lock_B[{j},{i}]"

    def x_pinned(a): return x_anchor.get(a) in ("left","right")
    def y_pinned(a): return y_anchor.get(a) in ("bottom","top")

    # Create non-conflicting locks:
    for a, b in itertools.permutations(DEPTS, 2):
        xi, xj = x_anchor.get(a), x_anchor.get(b)
        yi, yj = y_anchor.get(a), y_anchor.get(b)

        # If BOTH axes are pinned for BOTH nodes → skip pairwise locks entirely
        if x_pinned(a) and y_pinned(a) and x_pinned(b) and y_pinned(b):
            continue

        # Prefer locking on the axis where both are pinned; only one axis at mos
        if x_pinned(a) and x_pinned(b):
            if   xi == "left"  and xj == "right": set_lock_h(a, b, 'L')
            elif xi == "right" and xj == "left":  set_lock_h(a, b, 'R')
            continue

        if y_pinned(a) and y_pinned(b):
            if   yi == "bottom" and yj == "top":  set_lock_v(a, b, 'B')
            elif yi == "top"    and yj == "bottom": set_lock_v(a, b, 'T')
            continue


    # ---------- Objective ----------
    obj = gp.quicksum(flows[_canon(i,j)] * (dx[(i,j)] + dy[(i,j)]) for (i,j) in
    mdl.setObjective(obj, GRB.MINIMIZE)

    # ---------- Branch priorities ----------
    for d in DEPTS:
        x[d].BranchPriority = 10; y[d].BranchPriority = 10
    for (i,k), zvar in z.items():
        zvar.BranchPriority = 20 if AREAS[i] > 10000 else 5

    # ---------- Phase 1 ----------
    if threads is not None: mdl.setParam("Threads", int(threads))
    mdl.setParam("MIPFocus", 1)
    mdl.setParam("Heuristics", 0.35)
    mdl.setParam("Cuts", 2)
    mdl.setParam("CutPasses", 1)
    mdl.setParam("Presolve", 2)
    mdl.setParam("Symmetry", 2)
    mdl.setParam("Method", 2)
    mdl.setParam("Seed", seed)
    mdl.setParam("MIPGap", 0.01)
    mdl.setParam("TimeLimit", int(phase1_seconds))
    mdl.optimize()
    if mdl.status == GRB.INFEASIBLE:
        return _infeasible_result(layout, mdl)

    # ---------- Phase 2 ----------
    has_inc = (getattr(mdl, "SolCount", 0) or 0) > 0
    inc_vals = {}
    if has_inc:
        for v in mdl.getVars():
            inc_vals[v.VarName] = v.X
```

```
    target_hit = (has_inc and hasattr(mdl, "MIPGap") and mdl.MIPGap is not None
    if not target_hit:
        elapsed = int(phase1_seconds)  # safe: use remaining clock if available
        remaining = max(10, int(total_seconds - elapsed)) if total_seconds else
        mdl.reset()
        if threads is not None: mdl.setParam("Threads", int(threads))
        mdl.setParam("MIPFocus", 3)
        mdl.setParam("Heuristics", 0.2)
        mdl.setParam("Cuts", 2)
        mdl.setParam("CutPasses", 2)
        mdl.setParam("Presolve", 2)
        mdl.setParam("Symmetry", 2)
        mdl.setParam("Method", 3)
        if threads is not None and threads >= 8: mdl.setParam("ConcurrentMIP", 2
        mdl.setParam("Seed", 42)
        mdl.setParam("MIPGap", float(miph_gap_target))
        mdl.setParam("TimeLimit", remaining)
        if inc_vals:
            for v in mdl.getVars():
                if v.VarName in inc_vals:
                    v.Start = inc_vals[v.VarName]
        mdl.optimize()

    # ---------- Collect ----------
    has_sol = (getattr(mdl, "SolCount", 0) or 0) > 0

    def _val(v):
        if has_sol:
            try: return v.X
            except Exception: pass
        return getattr(v, "Start", None)

    best_obj = mdl.objVal if has_sol else None
    best_bound = getattr(mdl, "ObjBound", None)
    gap_val = None
    try:
        if has_sol and best_obj not in (None, 0) and best_bound is not None:
            gap_val = abs((best_bound - best_obj) / (best_obj + 1e-10))
        elif (best_obj is not None) and (best_bound is not None) and best_obj !=
            gap_val = abs((best_bound - best_obj) / (abs(best_obj) + 1e-10))
    except Exception:
        pass

    sol = {
        "obj": best_obj,
        "W": layout.W, "H": layout.H,
        "x": {i: _val(x[i]) for i in DEPTS},
        "y": {i: _val(y[i]) for i in DEPTS},
        "w": {i: _val(w[i]) for i in DEPTS},
        "h": {i: _val(h[i]) for i in DEPTS},
        "status": mdl.status,
        "gap": (mdl.MIPGap if has_sol and hasattr(mdl, "MIPGap") else gap_val),
        "bound": best_bound,
        "name": layout.name
    }
    status_map = {
        GRB.OPTIMAL: "OPTIMAL",
        GRB.TIME_LIMIT: "TIME_LIMIT",
        GRB.SUBOPTIMAL: "SUBOPTIMAL",
```

```
        GRB.INFEASIBLE: "INFEASIBLE",
        GRB.INF_OR_UNBD: "INF_OR_UNBD",
        GRB.INTERRUPTED: "INTERRUPTED",
        GRB.USER_OBJ_LIMIT: "USER_OBJ_LIMIT",
    }
    print(f"[{layout.name}] status={status_map.get(mdl.status, mdl.status)}, "
          f"incumbent={best_obj}, bound={best_bound}, gap={sol['gap']}")
    return sol

def _solve_one(spec, threads):
    import gurobipy as gp  # re-import inside worker
    res = build_and_solve_two_phase(
        spec,
        miph_gap_target=0.01,
        total_seconds=43200,
        phase1_seconds=180,
        verbose=True,
        threads=threads,
    )
    return spec.name, res

# 2) Compute thread split
CPU_TOTAL = os.cpu_count() or 6
RESERVE_CORES = 0
usable = max(1, CPU_TOTAL - RESERVE_CORES)
threads_per_model = max(1, usable // 3)
print(f"Detected {CPU_TOTAL} logical cores → allocating {threads_per_model} thre
```

```
Detected 144 logical cores → allocating 48 threads per model.
```

Visualization helper

We draw the facility boundary and the placed rectangles (department blocks) with labels and dimensions.

In [10]:

```
import matplotlib as mpl
import matplotlib.pyplot as plt
import numpy as np
import textwrap
from collections import OrderedDict

# ------------------ Global styling ------------------
mpl.rcParams.update({
    "figure.dpi": 120,
    "savefig.dpi": 260,
    "axes.titlesize": 16,
    "axes.labelsize": 11,
    "xtick.labelsize": 10,
    "ytick.labelsize": 10,
    "font.size": 10,
    "axes.edgecolor": "#1c1c1c",
    "axes.linewidth": 1.4,
})

PALETTE = [
    "#4C78A8", "#F58518", "#54A24B", "#E45756", "#72B7B2",
    "#EECA3B", "#B279A2", "#FF9DA6", "#9D755D", "#BAB0AC",
    "#2790C3", "#E69F00", "#009E73", "#CC79A7"
```

```
]

def _dept_colors(depts):
    return {d: PALETTE[i % len(PALETTE)] for i, d in enumerate(depts)}

def _nice_dim(w, h):
    return f"{w:.1f}×{h:.1f} m"

def _halo_text(ax, x, y, s, **kw):
    # text with a white halo for readability
    text = ax.text(x, y, s, **kw)
    import matplotlib.patheffects as patheffects
    text.set_path_effects([
        patheffects.withStroke(linewidth=3, foreground="white")
    ])
    return text

def _make_numbered_legend(ax, id_map, color_map, ncols=2, title="Departments"):
    """
    Build a compact legend that works on older Matplotlib (uses ncol, not ncols)
    """
    handles, labels = [], []
    for d, idx in id_map.items():
        patch = mpl.patches.Patch(
            facecolor=color_map[d], edgecolor="#222", label=f"{idx}. {d}", alpha
        )
        handles.append(patch)
        labels.append(f"{idx}. {d}")

    # Use ncol (older Matplotlib); ignore unknown kwargs defensively
    legend_kwargs = dict(
        loc="center left",
        bbox_to_anchor=(1.02, 0.5),
        title=title,
        frameon=True,
        borderpad=0.6,
        labelspacing=0.45,
        ncol=ncols,                # <-- use ncol here
        columnspacing=0.8,
        handlelength=1.2,
        fontsize=9
    )
    try:
        leg = ax.legend(handles, labels, **legend_kwargs)
    except TypeError:
        # super old fallback (strip possibly unsupported args)
        legend_kwargs.pop("columnspacing", None)
        legend_kwargs.pop("handlelength", None)
        legend_kwargs.pop("fontsize", None)
        leg = ax.legend(handles, labels, **legend_kwargs)

    leg.get_frame().set_facecolor("white")
    leg.get_frame().set_edgecolor("#ddd")
    return leg

def draw_layout(solution,
                title=None,
                figsize=(12, 7.2),
                legend=True,
                legend_ncols=2,
```

```
                show_dims_top_k=4,
                fname=None):

    # Guard: infeasible or missing geometry
    if solution.get("status") == gp.GRB.INFEASIBLE or solution.get("W") is None:
        fig, ax = plt.subplots(figsize=figsize)
        ax.set_title(title or f"{solution.get('name', 'Layout')} — INFEASIBLE")
        ax.axis("off")
        ax.text(0.5, 0.5,
                f"{solution.get('name','Layout')} is INFEASIBLE\n"
                f"(see IIS file in working directory)",
                ha="center", va="center", fontsize=14)
        plt.tight_layout()
        if fname: plt.savefig(fname, dpi=200, bbox_inches="tight")
        plt.show()
        return
    """
    Clean, professional plot:
    - Numbered badges inside each block (no long text)
    - Legend maps number -> department
    - Dimensions shown only for top-k largest areas (plus any > min_area_to_tag)
    """
    W, H = solution["W"], solution["H"]
    x_s, y_s, w_s, h_s = solution["x"], solution["y"], solution["w"], solution["

    # Establish deterministic department order and numeric IDs
    # (sorted by area descending → big first)
    areas = {d: w_s[d] * h_s[d] for d in DEPTS}
    order = sorted(DEPTS, key=lambda d: areas[d], reverse=True)
    id_map = OrderedDict((d, i+1) for i, d in enumerate(order))

    color_map = _dept_colors(DEPTS)

    fig, ax = plt.subplots(figsize=figsize)
    ax.set_xlim(0, W)
    ax.set_ylim(0, H)
    ax.set_aspect("equal", adjustable="box")
    ax.set_xlabel("X (m)")
    ax.set_ylabel("Y (m)")

    # Title with objective + gap on a second line in smaller font
    main_title = title or solution.get("name", "Layout")
    subbits = []
    if solution.get("obj") is not None:
        subbits.append(f"Objective (flow-weighted m): {solution['obj']:,.0f}")
    if solution.get("gap") is not None and solution["gap"] >= 0:
        subbits.append(f"MIP gap: {solution['gap']*100:.1f}%")
    subtitle = " • ".join(subbits)
    ax.set_title(main_title + ("\n" + subtitle if subtitle else ""), pad=12)

    # Facility boundary
    ax.add_patch(plt.Rectangle((0, 0), W, H, fill=False, lw=2.2, ec="#1b1b1b"))

    # Draw rectangles (big to small)
    for d in order:
        cx, cy, w, h = x_s[d], y_s[d], w_s[d], h_s[d]
        x0, y0 = cx - w/2.0, cy - h/2.0

        fc = color_map[d]
        ec = "#1b1b1b"
```

```python
        # block
        rect = plt.Rectangle((x0, y0), w, h,
                             facecolor=fc, edgecolor=ec,
                             linewidth=1.6, alpha=0.16, zorder=2)
        ax.add_patch(rect)

        # numeric badge (centered, always horizontal)
        idx = id_map[d]
        # badge box behind the number for contrast
        badge = mpl.patches.FancyBboxPatch(
            (cx, cy), 1, 1,  # dummy, we place with transform below
            boxstyle="round,pad=0.25,rounding_size=0.8",
            fc="white", ec="#333", lw=0.9, zorder=5, transform=None, visible=Fal
        )
        # Instead of drawing a box, just draw a haloed number for minimal clutte
        _halo_text(ax, cx, cy, f"{idx}",
                   ha="center", va="center", fontsize=max(min(min(w,h)/7.0, 13),
                   color="#111", weight="bold", zorder=6)

    # Dimension tags for largest K
    topK = set(order[:max(1, show_dims_top_k)])
    # Also tag any block occupying > 12% of facility area
    min_area_to_tag = 0.12 * (W * H)
    for d in order:
        cx, cy, w, h = x_s[d], y_s[d], w_s[d], h_s[d]
        x0, y0 = cx - w/2.0, cy - h/2.0
        if (d in topK) or (w*h >= min_area_to_tag):
            ax.text(x0 + 3, y0 + h - 3, _nice_dim(w, h),
                    ha="left", va="top", fontsize=9.0, color="#222",
                    bbox=dict(boxstyle="round,pad=0.22", fc="white", ec="none",
                    zorder=7)

    # Add neat grid
    ax.grid(True, which="both", alpha=0.14, linewidth=0.9)
    ax.tick_params(length=0)

    # Legend outside
    if legend:
        _make_numbered_legend(ax, id_map, color_map, ncols=legend_ncols, title="

    plt.tight_layout()
    if fname:
        plt.savefig(fname, bbox_inches="tight")
    plt.show()
```

Define I, L, and U patterns (dock placements)

We plug the dock rules into `LayoutSpec`:

- **I:** Inbound → left wall + centered vertically; Shipping → right wall + centered vertically
- **L:** Inbound → bottom wall near left wing (`x = 0.25W`); Shipping → right wall near upper wing (`y = 0.75H`)
- **U:** Both docks on **left wall**; Inbound lower (`0.25H`), Shipping upper (`0.75H`)

In [11]:

```python
# Recomputed totals: all 14 departments sum to 83,300 m²
# Use ≈ +30% slack for aisles/circulation → target ≈ 108–112k m²

I_layout = LayoutSpec(
    name="I-shaped",
    W=520.0, H=210.0,
    dock_rules={
        "Inbound Dock":  {"x_rule": "left",  "y_at":  0.5*210},   # e.g., y=40 m
        "Shipping Dock": {"x_rule": "right", "y_at": 0.5*210},   # e.g., y=170 m
    },
)

L_layout = LayoutSpec(
    name="L-shaped",
    W=400.0, H=280.0,
    dock_rules={
        "Inbound Dock":  {"y_rule": "bottom", "x_rule": "left"},  # X fixed
        "Shipping Dock": {"x_rule": "right", "y_rule": "top"},
    },
)

U_layout = LayoutSpec(
    name="U-shaped",
    W=330.0, H=330.0,
    dock_rules={
        "Inbound Dock":  {"x_rule": "left",  "y_at":  0.25*330},
        "Shipping Dock": {"x_rule": "left",  "y_at": 0.75*330},
    },
)
```

11.5 (Phase 1): Six-department run (merged Outbound Staging) — what happens here

This cell **does not change your solver**; it prepares a 6-department instance, runs I/L/U footprints in parallel, and plots the layouts. It first **backs up** the full data (`DEPTS` , `AREAS` , `flows`) and then **merges** the two outbound staging departments into a single node (`JOIN = "Outbound Staging"`) using robust regex detection (falling back gracefully if names differ). Areas are combined accordingly, and a 6-item department list is formed: *Inbound Dock, Receiving/Staging, Pallet Reserve Storage (Bulk), Packing/Wrap/Banding, Outbound Staging (merged), Shipping Dock*. The helper functions defined here are: `_flow_from_full(a,b)` (reads the original flow between two departments), `_flow6(a,b)` (returns the 6-dept flow, summing the two original outbound-staging links when the merged node is involved), and `_dims_for_ratio(area, ratio)` (derives width/height for a given target area and aspect ratio). These build `flows6` and swap the globals (`DEPTS` , `AREAS` , `flows`) to the 6-dept view. Next, it computes **footprint areas with slack** (I/L/U) and dimensions, and creates three `LayoutSpec` objects with **dock placement rules**: I-shape (docks mid-height on opposite sides), L-shape (inbound bottom-left, shipping top-right), and U-shape (both docks on the bottom edge, left/right quarters). Finally, it launches three parallel solves via `ProcessPoolExecutor` , each calling your existing `_solve_one(spec, threads_per_model)` , aggregates **status/objective/gap**, prints a compact summary, and calls `draw_layout(...)` to visualize each 6-department layout.

In [12]:
```
# === 11.5 (Phase 1): Six-department run (with merged Outbound Staging) ===
# This cell does *not* modify your solver. It only prepares a 6-dept dataset,
# runs I/L/U in parallel, and draws the layouts.

import math, itertools

# --- 0) Backup the full dataset so we can restore it later
AREAS_ALL = AREAS.copy()
DEPTS_ALL = DEPTS.copy()
FLOWS_ALL = flows.copy()

# --- 1) Define the 6 departments (merge the two outbound stagings)  [ROBUST] --
import re, itertools

JOIN = "Outbound Staging"

# Find all departments that look like "Outbound Staging ..."
cand = [d for d in DEPTS_ALL if "Outbound" in d and "Staging" in d]

# Try to identify the two variants by pattern
two_man  = [d for d in cand if re.search(r'2\s*[-—–]?\s*man', d, re.I)]
parcel   = [d for d in cand if re.search(r'parcel', d, re.I)]

if len(two_man) == 1 and len(parcel) == 1:
    JOIN_A = two_man[0]
    JOIN_B = parcel[0]
elif len(cand) >= 2:
    # Couldn't split reliably -> just take the first two distinct ones
    JOIN_A, JOIN_B = cand[:2]
else:
    # Already merged in the source (only one Outbound Staging)
    JOIN_A = cand[0] if cand else None
    JOIN_B = None

DEPTS6 = [
    "Inbound Dock",
    "Receiving/Staging",
    "Pallet Reserve Storage (Bulk)",
    "Packing / Wrap / Banding",
    JOIN,
    "Shipping Dock",
]

# Areas: copy five as-is, and merge (or pass-through) the outbound staging areas
AREAS6 = {k: AREAS_ALL[k] for k in DEPTS6 if k != JOIN}
if JOIN_A and JOIN_B:
    AREAS6[JOIN] = AREAS_ALL[JOIN_A] + AREAS_ALL[JOIN_B]
elif JOIN_A and not JOIN_B:
    AREAS6[JOIN] = AREAS_ALL[JOIN_A]
else:
    raise RuntimeError("Could not locate any 'Outbound Staging' department in th

# Helper to read a flow value from the original full matrix
def _flow_from_full(a: str, b: str) -> float:
    i, j = sorted([a, b], key=lambda s: DEPTS_ALL.index(s))
    return FLOWS_ALL[(i, j)]

# When the merged node is involved, sum flows from the two originals if both exi
def _flow6(a: str, b: str) -> float:
```

```
    if JOIN not in (a, b):
        return _flow_from_full(a, b)
    other = b if a == JOIN else a
    total = 0.0
    if JOIN_A: total += _flow_from_full(JOIN_A, other)
    if JOIN_B: total += _flow_from_full(JOIN_B, other)
    return total

# Build the 6-department flow dictionary keyed like your solver expects
flows6 = {(i, j): _flow6(i, j) for i, j in itertools.combinations(DEPTS6, 2)}

# --- 2) Swap the globals to point at the 6-department instance
DEPTS = DEPTS6
AREAS = AREAS6
flows = flows6

# --- 3) Make envelopes sized for the 6-dept case (pick your slack)
A6_base = sum(AREAS6.values())

A6_I = A6_base * 1.20   # I: modest slack
A6_L = A6_base * 1.55   # L: larger slack (avoid IIS)
A6_U = A6_base * 1.25   # U: medium slack

def _dims_for_ratio(area, ratio):
    H = (area / ratio) ** 0.5
    W = area / H
    return W, H

I6_W, I6_H = _dims_for_ratio(A6_I, ratio=2.2)
L6_W, L6_H = _dims_for_ratio(A6_L, ratio=1.4)
U6_W, U6_H = _dims_for_ratio(A6_U, ratio=1.0)

# I-shape: both docks halfway up the Y-axis, on opposite sides
I6_layout = LayoutSpec(
    name="I-shaped (6 depts)",
    W=I6_W, H=I6_H,
    dock_rules={
        "Inbound Dock":  {"x_rule": "left",  "y_at": 0.50 * I6_H},
        "Shipping Dock": {"x_rule": "right", "y_at": 0.50 * I6_H},
    },
)

# L-shape: inbound bottom-left corner; shipping on right wall in the top quadran
L6_layout = LayoutSpec(
    name="L-shaped (6 depts)",
    W=L6_W, H=L6_H,
    dock_rules={
        "Inbound Dock":  {"x_rule": "left",  "y_rule": "bottom"},  # bottom-left
        "Shipping Dock": {"x_rule": "right", "y_rule": "top"},     # top-right c
    },
)

# U-shape: both docks on the bottom edge;
# inbound in the most-left quadrant, outbound in the most-right quadrant
U6_layout = LayoutSpec(
    name="U-shaped (6 depts)",
    W=U6_W, H=U6_H,
    dock_rules={
        "Inbound Dock":  {"y_rule": "bottom", "x_at": 0.125 * U6_W},  # center c
        "Shipping Dock": {"y_rule": "bottom", "x_at": 0.875 * U6_W},  # center c
```

```
    },
)

# --- 4) Solve I/L/U in parallel (reuses your existing _solve_one + threads_per_
from concurrent.futures import ProcessPoolExecutor, as_completed

results6 = {}
futs = {}
with ProcessPoolExecutor(max_workers=3) as ex:
    futs[ex.submit(_solve_one, I6_layout, threads_per_model)] = "I"
    futs[ex.submit(_solve_one, L6_layout, threads_per_model)] = "L"
    futs[ex.submit(_solve_one, U6_layout, threads_per_model)] = "U"

    for fut in as_completed(futs):
        tag = futs[fut]
        try:
            name, sol = fut.result()
            results6[tag] = sol
            print(f"[6-dept {name}] status={sol.get('status')}, obj={sol.get('ob
        except Exception as e:
            print(f"[6-dept {tag}] failed:", e)

# --- 5) Quick summary and plots
summary6 = []
for tag in ["I", "L", "U"]:
    sol = results6.get(tag)
    if sol is None or sol.get("obj") is None:
        summary6.append((tag, None, None))
    else:
        summary6.append((sol["name"], sol["obj"], sol.get("gap")))

for name, obj, gap in summary6:
    if obj is None:
        print(f"{name:<20s} → infeasible or no solution.")
    else:
        print(f"{name:<20s} → Objective = {obj:,.1f} | Gap = {gap:.2%}")

draw_layout(results6.get("I"), title="I-shaped — 6 departments")
draw_layout(results6.get("L"), title="L-shaped — 6 departments")
draw_layout(results6.get("U"), title="U-shaped — 6 departments")
```

```
Restricted license - for non-production use only - expires 2026-11-23
Set parameter OutputFlag to value 1
Set parameter Threads to value 48
Restricted license - for non-production use only - expires 2026-11-23
Set parameter MIPFocus to value 1
Set parameter Heuristics to value 0.35
Set parameter OutputFlag to value 1
Set parameter Cuts to value 2
Set parameter CutPasses to value 1
Set parameter Presolve to value 2
Set parameter Symmetry to value 2
Set parameter Method to value 2
Set parameter Seed to value 42
Set parameter MIPGap to value 0.01
Set parameter Threads to value 48
Set parameter TimeLimit to value 180
Set parameter MIPFocus to value 1
Gurobi Optimizer version 12.0.3 build v12.0.3rc0 (linux64 - "Ubuntu 22.04.3 LTS")
Set parameter Heuristics to value 0.35

Set parameter Cuts to value 2
CPU model: Intel(R) Xeon(R) Platinum 8352V CPU @ 2.10GHz, instruction set [SSE2|A
VX|AVX2|AVX512]
Set parameter CutPasses to value 1
Set parameter Presolve to value 2
Thread count: 72 physical cores, 144 logical processors, using up to 48 threads
Set parameter Symmetry to value 2

Set parameter Method to value 2
Non-default parameters:
TimeLimit  180
Set parameter Seed to value 42
Set parameter MIPGap to value 0.01
MIPGap  0.01
Set parameter TimeLimit to value 180
Method  2
Heuristics  0.35
Gurobi Optimizer version 12.0.3 build v12.0.3rc0 (linux64 - "Ubuntu 22.04.3 LTS")

MIPFocus  1
CPU model: Intel(R) Xeon(R) Platinum 8352V CPU @ 2.10GHz, instruction set [SSE2|A
VX|AVX2|AVX512]
Thread count: 72 physical cores, 144 logical processors, using up to 48 threads
Symmetry  2

Cuts  2
Non-default parameters:
CutPasses  1
TimeLimit  180
Presolve  2
MIPGap  0.01
Seed  42
Restricted license - for non-production use only - expires 2026-11-23
Method  2
Threads  48
Heuristics  0.35
Set parameter OutputFlag to value 1

MIPFocus  1
Symmetry  2
```

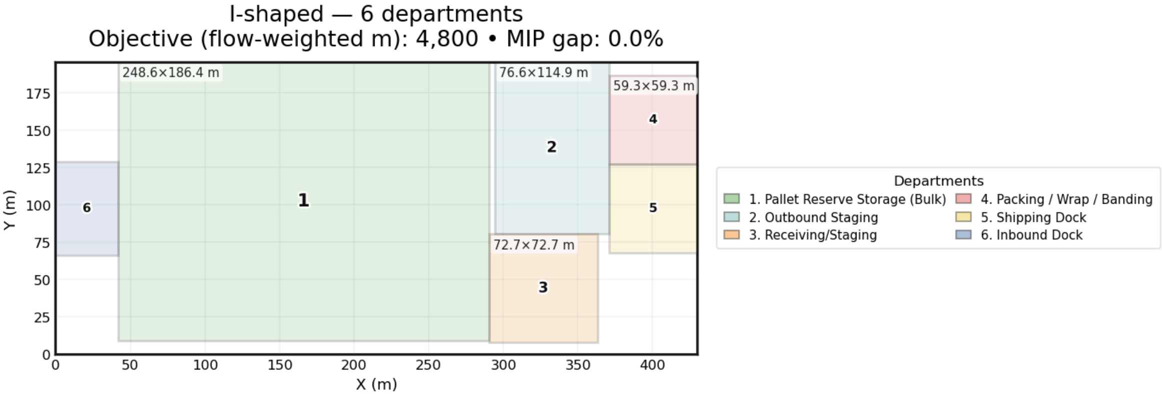
```
Optimize a model with 407 rows, 201 columns and 1467 nonzeros
Cuts  2
Model fingerprint: 0x1189bce5
CutPasses  1
Variable types: 82 continuous, 119 integer (119 binary)
Presolve  2
Coefficient statistics:
Seed  42
  Matrix range     [5e-01, 4e+02]
Threads  48
  Objective range  [5e-02, 8e+00]

Set parameter Threads to value 48
  Bounds range     [1e+00, 4e+02]
Optimize a model with 405 rows, 203 columns and 1471 nonzeros
Set parameter MIPFocus to value 1
Model fingerprint: 0x8cded6fe
  RHS range        [1e+00, 4e+02]
Set parameter Heuristics to value 0.35
Using branch priorities.
Variable types: 82 continuous, 121 integer (121 binary)
Set parameter Cuts to value 2
Coefficient statistics:
Set parameter CutPasses to value 1
  Matrix range     [5e-01, 3e+02]
Set parameter Presolve to value 2
  Objective range  [5e-02, 8e+00]
  Bounds range     [1e+00, 3e+02]
Set parameter Symmetry to value 2
  RHS range        [1e+00, 3e+02]
Set parameter Method to value 2
Using branch priorities.
Set parameter Seed to value 42
Set parameter MIPGap to value 0.01
Set parameter TimeLimit to value 180
Presolve removed 199 rows and 85 columns
Presolve time: 0.01s
Gurobi Optimizer version 12.0.3 build v12.0.3rc0 (linux64 - "Ubuntu 22.04.3 LTS")

Presolved: 208 rows, 116 columns, 780 nonzeros
CPU model: Intel(R) Xeon(R) Platinum 8352V CPU @ 2.10GHz, instruction set [SSE2|A
VX|AVX2|AVX512]
Thread count: 72 physical cores, 144 logical processors, using up to 48 threads

Presolve removed 93 rows and 42 columns
Variable types: 64 continuous, 52 integer (52 binary)
Presolve time: 0.01s
Non-default parameters:
Presolved: 312 rows, 161 columns, 1238 nonzeros
TimeLimit  180
Found heuristic solution: objective 7614.2850032
MIPGap  0.01
Method  2
Variable types: 75 continuous, 86 integer (86 binary)
Root relaxation presolve removed 2 rows and 0 columns
Heuristics  0.35
Root relaxation presolved: 206 rows, 117 columns, 768 nonzeros
MIPFocus  1
Root relaxation presolve removed 1 rows and 0 columns
```

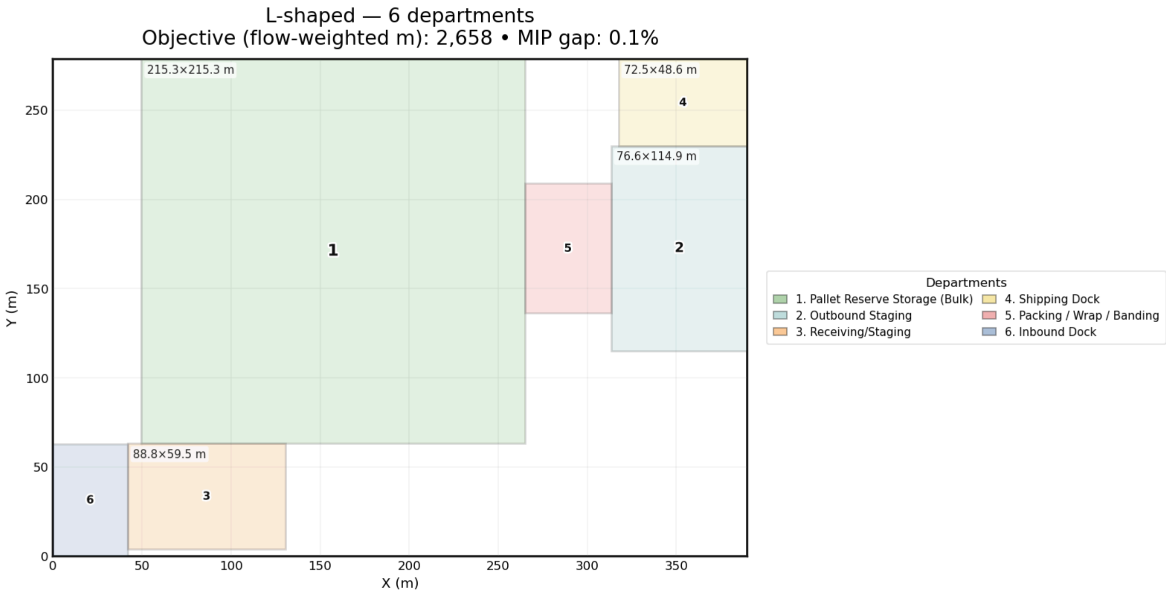
```
Symmetry  2
Root relaxation presolved: 311 rows, 161 columns, 1228 nonzeros
Root barrier log...

Cuts  2

Root barrier log...
CutPasses  1
Ordering time: 0.00s
Presolve  2


Seed  42
Ordering time: 0.00s
Barrier statistics:
Threads  48


 AA' NZ     : 3.629e+03
Barrier statistics:
 Factor NZ  : 7.719e+03
Optimize a model with 405 rows, 204 columns and 1476 nonzeros
 AA' NZ     : 6.544e+03
Model fingerprint: 0x12a5913c
 Factor Ops : 3.866e+05 (less than 1 second per iteration)
 Factor NZ  : 2.054e+04
Variable types: 82 continuous, 122 integer (122 binary)
 Threads    : 1
 Factor Ops : 1.968e+06 (less than 1 second per iteration)
Coefficient statistics:
 Threads    : 1
  Matrix range     [5e-01, 4e+02]

  Objective range  [5e-02, 8e+00]
                  Objective                Residual

  Bounds range     [1e+00, 4e+02]
Iter       Primal          Dual         Primal    Dual     Compl     Time
                  Objective                Residual
Iter       Primal          Dual         Primal    Dual     Compl     Time
  RHS range        [1e+00, 4e+02]
   0   1.29873692e+04 -5.05728932e+04  6.04e+01 9.83e+00  9.61e+02     0s
Using branch priorities.
   0   1.60685852e+04 -4.23830240e+04  1.15e+02 5.65e+00  1.17e+03     0s
   1   6.80557955e+03 -3.01958624e+04  6.59e+00 3.06e-14  1.45e+02     0s
   1   8.98132592e+03 -5.52474051e+04  3.39e+01 4.26e-14  3.04e+02     0s
   2   4.91409346e+03 -3.06717706e+03  8.83e-01 1.15e-12  2.22e+01     0s
   2   3.76006455e+03 -1.64734172e+04  1.96e+00 3.27e-14  4.06e+01     0s
   3   4.10446465e+03  1.60148200e+03  4.14e-02 4.63e-13  6.03e+00     0s
   3   2.41914435e+03 -2.95372186e+02  2.29e-01 2.13e-14  4.82e+00     0s
   4   3.83813395e+03  3.53952523e+03  9.89e-04 1.74e-14  7.12e-01     0s
   4   2.16119196e+03  1.87904985e+03  1.97e-02 2.20e-14  4.81e-01     0s
   5   3.77333395e+03  3.67986954e+03  1.00e-04 9.77e-15  2.23e-01     0s
Presolve removed 94 rows and 39 columns
Presolve time: 0.01s
   5   2.09292329e+03  2.05587231e+03  2.25e-03 1.42e-14  6.29e-02     0s
   6   3.75339165e+03  3.72553692e+03  1.97e-05 1.42e-14  6.63e-02     0s
Presolved: 311 rows, 165 columns, 1265 nonzeros
   7   3.74869349e+03  3.73890436e+03  6.67e-06 6.49e-15  2.33e-02     0s
   6   2.08176114e+03  2.07903740e+03  1.31e-04 1.42e-14  4.62e-03     0s
```

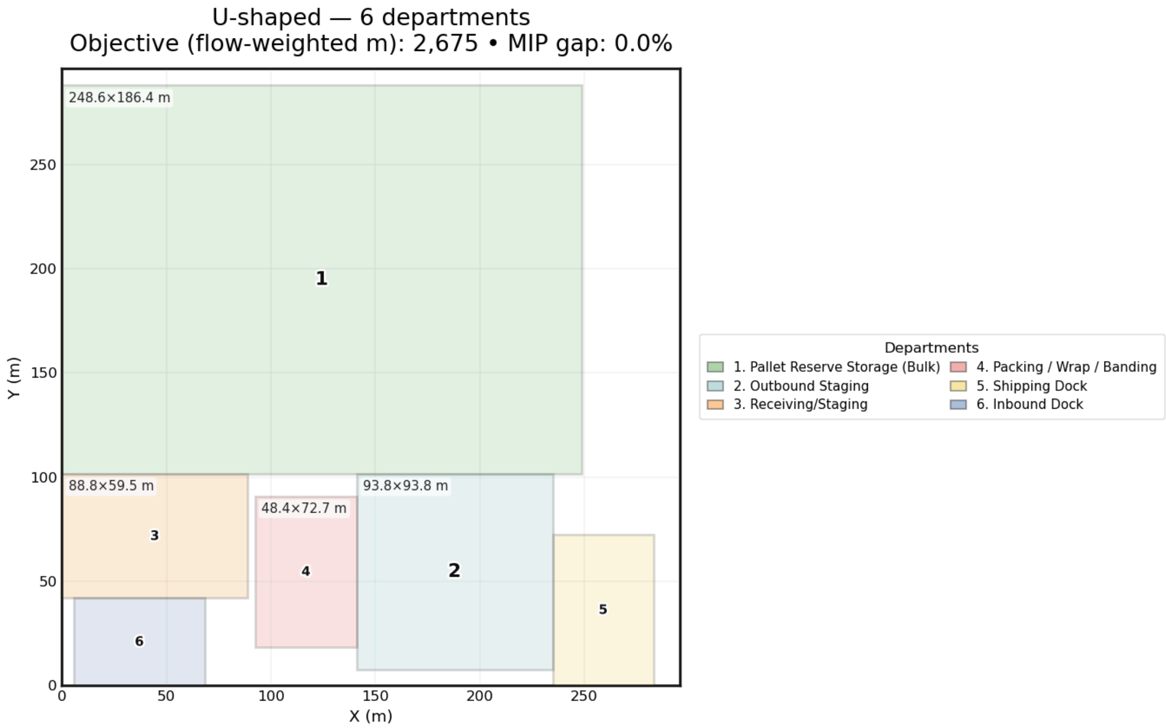
```
   8   3.74528449e+03  3.74419444e+03  5.00e-09 1.42e-14  2.60e-03     0s
Variable types: 74 continuous, 91 integer (91 binary)
   7   2.08053849e+03  2.08048102e+03  7.21e-07 8.03e-15  9.73e-05     0s
   9   3.74517448e+03  3.74514810e+03  1.05e-11 1.42e-14  6.28e-05     0s
   8   2.08052228e+03  2.08052222e+03  2.07e-11 2.84e-14  9.74e-08     0s
  10   3.74516700e+03  3.74516683e+03  2.65e-12 1.42e-14  3.86e-07     0s
Found heuristic solution: objective 6455.9931283
  11   3.74516686e+03  3.74516686e+03  6.52e-10 1.42e-14  4.69e-09     0s
   9   2.08052226e+03  2.08052226e+03  1.90e-11 3.27e-14  9.74e-14     0s
Root relaxation presolve removed 1 rows and 0 columns


Root relaxation presolved: 310 rows, 165 columns, 1255 nonzeros
Barrier solved model in 11 iterations and 0.11 seconds (0.01 work units)
Barrier solved model in 9 iterations and 0.09 seconds (0.01 work units)

Optimal objective 3.74516686e+03
Optimal objective 2.08052226e+03

Root barrier log...




Root relaxation: objective 3.745167e+03, 59 iterations, 0.07 seconds (0.00 work u
nits)
Root relaxation: objective 2.080522e+03, 64 iterations, 0.06 seconds (0.01 work u
nits)
Ordering time: 0.00s

Barrier statistics:

 AA' NZ     : 6.574e+03
    Nodes    |    Current Node    |     Objective Bounds      |     Work

 Factor NZ  : 2.162e+04
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time
    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Factor Ops : 2.210e+06 (less than 1 second per iteration)

 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time
 Threads    : 1
     0     0 3745.16686    0    8 7614.28500 3745.16686  50.8%     -    0s

     0     0 2080.52226    0   25          - 2080.52226      -     -    0s

H    0     0                    4881.7886131 3745.16686  23.3%     -    0s
                  Objective                Residual
H    0     0                    5336.1431542 2080.52226  61.0%     -    0s
Iter       Primal          Dual         Primal    Dual     Compl     Time
   0   2.19116989e+04 -6.08993323e+04  1.95e+02 6.81e+00  1.73e+03     0s
   1   1.13312576e+04 -7.85470869e+04  4.88e+01 4.26e-14  4.06e+02     0s
   2   4.93939832e+03 -2.19273223e+04  1.80e+00 5.68e-14  5.03e+01     0s
     0     0 3865.58754    0   18 4881.78861 3865.58754  20.8%     -    0s
   3   3.18461501e+03 -5.69197923e+02  4.14e-01 4.97e-14  6.72e+00     0s
H    0     0                    4800.0683795 3865.58754  19.5%     -    0s
   4   2.51233420e+03  1.65270526e+03  3.60e-02 4.09e-14  1.45e+00     0s
H    0     0                    3258.1593096 2271.87829  30.3%     -    0s
   5   2.42211550e+03  2.23194200e+03  4.53e-03 1.71e-14  3.19e-01     0s
   6   2.40481638e+03  2.36695926e+03  1.78e-03 1.04e-14  6.34e-02     0s
```

```
     0     0 2271.87829    0   47 3258.15931 2271.87829  30.3%     -    0s
   7   2.39544469e+03  2.39291090e+03  1.44e-11 2.26e-14  4.24e-03     0s
   8   2.39443170e+03  2.39397810e+03  1.95e-11 2.84e-14  7.59e-04     0s
H    0     0                    2710.9315731 2271.87829  16.2%     -    0s
   9   2.39427768e+03  2.39426615e+03  2.17e-10 7.11e-15  1.93e-05     0s
  10   2.39427114e+03  2.39427114e+03  2.84e-11 2.20e-14  4.06e-09     0s

Barrier solved model in 10 iterations and 0.10 seconds (0.01 work units)
     0     2 3901.00302    0   18 4800.06838 3901.00302  18.7%     -    0s
Optimal objective 2.39427114e+03


Root relaxation: objective 2.394271e+03, 72 iterations, 0.07 seconds (0.01 work u
nits)

     0     2 2271.87829    0   47 2710.93157 2271.87829  16.2%     -    0s
Cutting planes:

  Gomory: 1
    Nodes    |    Current Node    |     Objective Bounds      |     Work
  Lift-and-project: 1
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time
  Cover: 5

  Implied bound: 2
     0     0 2394.27114    0   24 6455.99313 2394.27114  62.9%     -    0s
  MIR: 3
  Flow cover: 21
H    0     0                    6439.2201434 2394.27114  62.8%     -    0s
  RLT: 1
  Relax-and-lift: 3
H    0     0                    3752.9430112 2394.27114  36.2%     -    0s

Explored 29 nodes (646 simplex iterations) in 0.24 seconds (0.03 work units)
Thread count was 48 (of 144 available processors)

Solution count 3: 4800.07 4881.79 7614.29

Optimal solution found (tolerance 1.00e-02)
Best objective 4.800068379474e+03, best bound 4.800068379474e+03, gap 0.0000%
H    0     0                    2937.1043222 2394.27114  18.5%     -    0s
[I-shaped (6 depts)] status=OPTIMAL, incumbent=4800.0683794744355, bound=4800.068
379474435, gap=0.0
     0     0 2553.45241    0   31 2937.10432 2553.45241  13.1%     -    0s
H   75    13                    2674.8850736 2487.84579  6.99%  12.8    0s

Cutting planes:
  Learned: 1
H    0     0                    2885.8090178 2553.45241  11.5%     -    0s
  Gomory: 3
  Cover: 10
  Implied bound: 8
  Clique: 5
  MIR: 9
  Flow cover: 41
  Flow path: 1
  RLT: 9
H    0     0                    2794.3922985 2553.45241  8.62%     -    0s

Explored 110 nodes (1564 simplex iterations) in 0.29 seconds (0.05 work units)
```

```
Thread count was 48 (of 144 available processors)

Solution count 4: 2674.89 2710.93 3258.16 5336.14

Optimal solution found (tolerance 1.00e-02)
Best objective 2.674885073636e+03, best bound 2.674885073636e+03, gap 0.0000%
[U-shaped (6 depts)] status=OPTIMAL, incumbent=2674.885073636292, bound=2674.8850
73636292, gap=0.0
     0     2 2553.45241    0   31 2794.39230 2553.45241  8.62%     -    0s
H    4     8                    2759.1918517 2553.45241  7.46%  13.2    0s
H   13    16                    2703.3329324 2553.45241  5.54%  16.3    0s
H   42    41                    2692.2548301 2581.41706  4.12%  10.9    0s
H   48    34                    2658.1843518 2598.13432  2.26%  11.1    0s

Cutting planes:
  Gomory: 4
  Cover: 2
  Implied bound: 7
  MIR: 6
  Flow cover: 35
  Flow path: 1
  Zero half: 1
  RLT: 8
  Relax-and-lift: 5

Explored 150 nodes (1467 simplex iterations) in 0.34 seconds (0.06 work units)
Thread count was 48 (of 144 available processors)

Solution count 10: 2658.18 2692.25 2703.33 ... 6455.99

Optimal solution found (tolerance 1.00e-02)
Best objective 2.658184351780e+03, best bound 2.654376287502e+03, gap 0.1433%
[L-shaped (6 depts)] status=OPTIMAL, incumbent=2658.1843517804364, bound=2654.376
287502335, gap=0.0014325809553242099
[6-dept I-shaped (6 depts)] status=2, obj=4800.0683794744355, gap=0.0
[6-dept U-shaped (6 depts)] status=2, obj=2674.885073636292, gap=0.0
[6-dept L-shaped (6 depts)] status=2, obj=2658.1843517804364, gap=0.0014325809553
242099
I-shaped (6 depts)   → Objective = 4,800.1 | Gap = 0.00%
L-shaped (6 depts)   → Objective = 2,658.2 | Gap = 0.14%
U-shaped (6 depts)   → Objective = 2,674.9 | Gap = 0.00%
```







Six-department layouts — summary and managerial implications

Under a common solver regime (48 threads; `MIPFocus=1`, `Heuristics=0.35`, `Cuts=2`, `CutPasses=1`, `Presolve=2`, `Symmetry=2`, `Method=2`, `Seed=42`, `TimeLimit=180 s`, target `MIPGap=1%`), the six-department instance converged to near-optimal solutions for all three footprints. The **L-shaped** layout achieved the lowest flow-weighted travel distance with an objective of **2,658.2 m** at a **0.14%** gap; the **U-shaped** layout was statistically very close at **2,674.9 m** with a **0.00%** gap (≈ **0.63%** higher than L); and the **I-shaped** alternative was markedly worse at **4,800.1 m** with a **0.00%** gap. The performance ordering is consistent with geometry: by co-locating docks on adjoining (L) or the same (U) sides, inbound → receiving and reserve → outbound paths are shortened and replenishment to staging is more direct, whereas the I-shape forces longer cross-facility movements between opposing dock faces. Visual inspection confirms that **Pallet Reserve Storage (Bulk)** remains a single, contiguous block (as required) and that **Packing/Wrap/Banding** and the merged **Outbound Staging** achieve tighter proximity in the L/U variants, which helps reduce high-frequency flows in the SLP-derived matrix.

From a managerial perspective, the **L-shape** is the recommended baseline given its best measured travel, yet the **U-shape** represents an almost equivalent solution; therefore, site-specific considerations should guide the final choice. If yard access, road approach, or door availability favors a shared dock wall, the U-shape's negligible penalty (~0.6%) may be outweighed by simpler trailer choreography and supervision. Conversely, if avoiding outbound–inbound interference and reserving a clean outbound corridor are priorities, the L-shape's orthogonal wings can better segregate traffic while preserving short replenishment links from Reserve to Staging. In both cases, keep **Reserve** contiguous and positioned to minimize replenishment distance to **Outbound Staging**, maintain adjacency between **Packing** and **Outbound Staging**, and validate that staging depths at **Shipping Dock** are sized for peak waves (2-man delivery vs. parcel). Prior to commitment, we advise a short robustness study—perturb SLP weights, door counts, and department areas—because the two leading footprints are near-ties; if rankings remain stable and MIP gaps stay ≤1%, the selected layout can be adopted with high confidence and rolled into detailed aisle/slotting design and door-assignment planning.

11.5 (Phase 2): Full 14-department run

This cell **restores the complete problem** and reuses the earlier layouts to solve the **full 14-department** instance for the I, L, and U footprints. First, it writes the backed-up datasets (`DEPTS_ALL` , `AREAS_ALL` , `FLOWS_ALL`) back into the globals (`DEPTS` , `AREAS` , `flows`) that the solver consumes. It then reuses the pre-defined `LayoutSpec` objects sized for the full set (`I_layout` , `L_layout` , `U_layout`) and dispatches three **parallel** solves with `ProcessPoolExecutor` , each calling your existing `_solve_one(spec, threads_per_model)` (so it inherits the same solver settings, time limit, and MIP-gap target used elsewhere). As each run finishes, the code records the solution, prints a compact line with **status**, **objective (flow-weighted meters)**, and **MIP gap**, and gracefully reports any failure. Finally, it assembles a quick textual **summary** of the three footprints and calls `draw_layout(...)` to plot the I-, L-, and U-shaped layouts with their department rectangles, allowing visual comparison alongside the objective values and gaps.

In [13]:
```python
# === 11.5 (Phase 2): Restore full 14-department instance and solve again ===

# 1) Restore the full dataset into the globals the solver uses
DEPTS = DEPTS_ALL
AREAS = AREAS_ALL
flows = FLOWS_ALL

# 2) Reuse your existing three LayoutSpec objects sized for the full set:
#    I_layout, L_layout, U_layout  (defined earlier in your notebook)

from concurrent.futures import ProcessPoolExecutor, as_completed

results14 = {}
```

```python
futs = {}
with ProcessPoolExecutor(max_workers=3) as ex:
    futs[ex.submit(_solve_one, I_layout, threads_per_model)] = "I"
    futs[ex.submit(_solve_one, L_layout, threads_per_model)] = "L"
    futs[ex.submit(_solve_one, U_layout, threads_per_model)] = "U"

    for fut in as_completed(futs):
        tag = futs[fut]
        try:
            name, sol = fut.result()
            results14[tag] = sol
            print(f"[14-dept {name}] status={sol.get('status')}, obj={sol.get('c
        except Exception as e:
            print(f"[14-dept {tag}] failed:", e)

# 3) Quick summary and plots
summary14 = []
for tag in ["I", "L", "U"]:
    sol = results14.get(tag)
    if sol is None or sol.get("obj") is None:
        summary14.append((tag, None, None))
    else:
        summary14.append((sol["name"], sol["obj"], sol.get("gap")))

for name, obj, gap in summary14:
    if obj is None:
        print(f"{name:<20s} → infeasible or no solution.")
    else:
        print(f"{name:<20s} → Objective = {obj:,.1f} | Gap = {gap:.2%}")

draw_layout(results14.get("I"), title="I-shaped — 14 departments")
draw_layout(results14.get("L"), title="L-shaped — 14 departments")
draw_layout(results14.get("U"), title="U-shaped — 14 departments")
```

```
Restricted license - for non-production use only - expires 2026-11-23
Set parameter OutputFlag to value 1
Restricted license - for non-production use only - expires 2026-11-23
Set parameter OutputFlag to value 1
Restricted license - for non-production use only - expires 2026-11-23
Set parameter Threads to value 48
Set parameter OutputFlag to value 1
Set parameter MIPFocus to value 1
Set parameter Heuristics to value 0.35
Set parameter Cuts to value 2
Set parameter CutPasses to value 1
Set parameter Presolve to value 2
Set parameter Symmetry to value 2
Set parameter Method to value 2
Set parameter Seed to value 42
Set parameter MIPGap to value 0.01
Set parameter TimeLimit to value 180
Gurobi Optimizer version 12.0.3 build v12.0.3rc0 (linux64 - "Ubuntu 22.04.3 LTS")

Set parameter Threads to value 48
CPU model: Intel(R) Xeon(R) Platinum 8352V CPU @ 2.10GHz, instruction set [SSE2|A
VX|AVX2|AVX512]
Set parameter MIPFocus to value 1
Thread count: 72 physical cores, 144 logical processors, using up to 48 threads
Set parameter Heuristics to value 0.35

Set parameter Cuts to value 2
Non-default parameters:
Set parameter CutPasses to value 1
TimeLimit  180
Set parameter Presolve to value 2
MIPGap  0.01
Set parameter Symmetry to value 2
Method  2
Set parameter Method to value 2
Set parameter Threads to value 48
Heuristics  0.35
Set parameter Seed to value 42
Set parameter MIPFocus to value 1
MIPFocus  1
Set parameter MIPGap to value 0.01
Symmetry  2
Set parameter Heuristics to value 0.35
Set parameter TimeLimit to value 180
Set parameter Cuts to value 2
Cuts  2
Gurobi Optimizer version 12.0.3 build v12.0.3rc0 (linux64 - "Ubuntu 22.04.3 LTS")
CutPasses  1
Set parameter CutPasses to value 1

Set parameter Presolve to value 2
Presolve  2
CPU model: Intel(R) Xeon(R) Platinum 8352V CPU @ 2.10GHz, instruction set [SSE2|A
VX|AVX2|AVX512]
Seed  42
Set parameter Symmetry to value 2
Thread count: 72 physical cores, 144 logical processors, using up to 48 threads
Threads  48
Set parameter Method to value 2
```

```
Set parameter Seed to value 42
Non-default parameters:
Optimize a model with 1716 rows, 871 columns and 6581 nonzeros
TimeLimit  180
Set parameter MIPGap to value 0.01
Model fingerprint: 0x6718ff3d
MIPGap  0.01
Set parameter TimeLimit to value 180
Variable types: 328 continuous, 543 integer (543 binary)
Method  2
Heuristics  0.35
Gurobi Optimizer version 12.0.3 build v12.0.3rc0 (linux64 - "Ubuntu 22.04.3 LTS")
Coefficient statistics:
MIPFocus  1

  Matrix range     [5e-01, 4e+02]
Symmetry  2
CPU model: Intel(R) Xeon(R) Platinum 8352V CPU @ 2.10GHz, instruction set [SSE2|A
VX|AVX2|AVX512]
Cuts  2
  Objective range  [5e-02, 4e+00]
Thread count: 72 physical cores, 144 logical processors, using up to 48 threads
CutPasses  1
  Bounds range     [1e+00, 4e+02]
Presolve  2

  RHS range        [1e+00, 4e+02]
Seed  42
Using branch priorities.
Non-default parameters:
Threads  48

TimeLimit  180
Optimize a model with 1716 rows, 872 columns and 6582 nonzeros
Model fingerprint: 0x26924e00
MIPGap  0.01
Variable types: 328 continuous, 544 integer (544 binary)
Coefficient statistics:
Method  2
  Matrix range     [5e-01, 3e+02]
Heuristics  0.35
  Objective range  [5e-02, 4e+00]
MIPFocus  1
  Bounds range     [1e+00, 3e+02]
Symmetry  2
  RHS range        [1e+00, 3e+02]
Cuts  2
Using branch priorities.
CutPasses  1
Presolve  2
Seed  42
Threads  48

Optimize a model with 1718 rows, 868 columns and 6572 nonzeros
Model fingerprint: 0xf1c6a68b
Presolve removed 200 rows and 87 columns
Variable types: 328 continuous, 540 integer (540 binary)
Presolve time: 0.03s
Coefficient statistics:
```

```
Presolved: 1516 rows, 784 columns, 6237 nonzeros
  Matrix range     [5e-01, 5e+02]
  Objective range  [5e-02, 4e+00]
  Bounds range     [1e+00, 5e+02]
  RHS range        [1e+00, 5e+02]
Variable types: 320 continuous, 464 integer (464 binary)
Using branch priorities.
Presolve removed 158 rows and 47 columns
Root relaxation presolve removed 1 rows and 0 columns
Root relaxation presolved: 1515 rows, 784 columns, 6213 nonzeros
Presolve time: 0.03s

Presolved: 1558 rows, 825 columns, 6406 nonzeros
Root barrier log...

Variable types: 321 continuous, 504 integer (504 binary)
Root relaxation presolve removed 1 rows and 0 columns
Root relaxation presolved: 1557 rows, 825 columns, 6382 nonzeros
Presolve removed 190 rows and 60 columns

Presolve time: 0.03s
Root barrier log...
Presolved: 1528 rows, 808 columns, 6391 nonzeros

Variable types: 318 continuous, 490 integer (490 binary)
Root relaxation presolve removed 2 rows and 0 columns
Root relaxation presolved: 1526 rows, 808 columns, 6363 nonzeros

Root barrier log...

Ordering time: 0.06s

Ordering time: 0.07s
Barrier statistics:
 AA' NZ     : 7.743e+04
 Factor NZ  : 4.488e+05 (roughly 5 MB of memory)
 Factor Ops : 1.942e+08 (less than 1 second per iteration)
 Threads    : 48
Ordering time: 0.06s

Barrier statistics:
 AA' NZ     : 7.731e+04

 Factor NZ  : 4.322e+05 (roughly 4 MB of memory)
                  Objective                Residual
 Factor Ops : 1.817e+08 (less than 1 second per iteration)
Iter       Primal          Dual         Primal    Dual     Compl     Time
 Threads    : 48
   0   5.58020680e+04 -1.82622394e+05  6.32e+02 3.86e+00  1.13e+03     0s
   1   3.01208381e+04 -2.76031523e+05  1.91e+02 4.90e-14  3.03e+02     0s
   2   1.13832477e+04 -1.01524162e+05  1.87e+01 6.75e-14  4.97e+01     0s

Barrier statistics:
 AA' NZ     : 7.726e+04
 Factor NZ  : 4.065e+05 (roughly 4 MB of memory)
   3   6.92979318e+03 -1.01577006e+04  3.52e+00 3.11e-14  6.52e+00     0s
 Factor Ops : 1.656e+08 (less than 1 second per iteration)
 Threads    : 48

                  Objective                Residual
```

```
Iter       Primal          Dual         Primal    Dual     Compl     Time
   4   5.64766601e+03  1.54677560e+03  8.73e-01 1.24e-14  1.46e+00     0s
   0   3.89330650e+04 -1.69723269e+05  4.14e+02 3.81e+00  8.00e+02     0s

   5   5.20086510e+03  3.65872320e+03  4.27e-01 1.13e-14  5.40e-01     0s
              Objective                Residual
   1   2.02161759e+04 -2.25939160e+05  9.89e+01 1.63e-13  1.78e+02     0s
Iter       Primal          Dual         Primal    Dual     Compl     Time
   0   4.50856199e+04 -2.07813572e+05  4.37e+02 5.21e+00  9.59e+02     0s
   6   4.91852458e+03  4.23004988e+03  1.89e-01 2.54e-14  2.39e-01     0s
   2   8.62414255e+03 -1.78406734e+05  1.14e+01 4.97e-14  7.14e+01     0s
   7   4.81058146e+03  4.56465911e+03  8.18e-02 1.51e-14  8.51e-02     0s
   1   1.92035513e+04 -2.42825117e+05  6.32e+01 7.82e-14  1.68e+02     0s
   3   5.63751117e+03 -1.82164010e+04  6.64e-01 4.17e-14  8.04e+00     0s
   8   4.75564250e+03  4.68076426e+03  9.66e-03 3.30e-14  2.57e-02     0s
   9   4.74695084e+03  4.72470847e+03  4.26e-03 7.77e-15  7.65e-03     0s
   2   9.91736483e+03 -5.43282089e+04  9.47e+00 9.17e-14  2.69e+01     0s
   4   4.28937865e+03 -1.98974433e+03  2.03e-01 1.58e-14  2.09e+00     0s
  10   4.74129045e+03  4.73534715e+03  5.20e-04 1.48e-14  2.04e-03     0s
   5   3.51076489e+03  1.28513115e+03  6.77e-02 1.85e-14  7.38e-01     0s
  11   4.74073966e+03  4.73936699e+03  2.35e-04 2.70e-14  4.72e-04     0s
   6   3.21973366e+03  2.29805669e+03  3.01e-02 1.89e-14  3.05e-01     0s
   3   6.19051696e+03 -7.73301997e+03  9.15e-01 1.78e-14  4.88e+00     0s
  12   4.74043895e+03  4.74007036e+03  6.82e-05 2.27e-14  1.27e-04     0s
   7   3.05619839e+03  2.80988646e+03  9.20e-03 8.77e-15  8.16e-02     0s
   4   5.39143606e+03  5.21175318e+02  2.77e-01 1.33e-14  1.66e+00     0s
   5   4.99978313e+03  2.80750743e+03  1.27e-01 2.28e-14  7.45e-01     0s
  13   4.74031838e+03  4.74028575e+03  4.79e-06 2.98e-14  1.12e-05     0s
   8   3.02500190e+03  2.94598824e+03  3.03e-03 1.98e-14  2.62e-02     0s
   6   4.72793016e+03  3.99885100e+03  3.51e-02 2.93e-14  2.47e-01     0s
  14   4.74030804e+03  4.74030740e+03  4.42e-08 9.85e-15  2.21e-07     0s
   9   3.01238666e+03  2.98854372e+03  1.12e-03 1.51e-14  7.90e-03     0s
   7   4.64458019e+03  4.45767861e+03  1.43e-02 2.34e-14  6.33e-02     0s
  15   4.74030788e+03  4.74030788e+03  4.69e-09 1.95e-14  3.18e-10     0s

Barrier solved model in 15 iterations and 0.47 seconds (0.10 work units)
Optimal objective 4.74030788e+03
  10   3.00714869e+03  3.00283478e+03  1.52e-04 1.50e-14  1.43e-03     0s

   8   4.61048935e+03  4.56291897e+03  5.28e-03 3.26e-14  1.61e-02     0s

Root relaxation: objective 4.740308e+03, 353 iterations, 0.44 seconds (0.06 work
units)
   9   4.59791968e+03  4.58018460e+03  2.26e-03 1.26e-14  6.01e-03     0s
  11   3.00626579e+03  3.00477862e+03  3.08e-05 2.38e-14  4.92e-04     0s
  10   4.59081952e+03  4.58811682e+03  3.32e-04 1.78e-14  9.16e-04     0s
  11   4.58957473e+03  4.58910431e+03  2.47e-05 1.67e-14  1.59e-04     0s
  12   3.00606837e+03  3.00582015e+03  4.47e-06 1.55e-14  8.22e-05     1s

  13   3.00602673e+03  3.00600787e+03  7.28e-07 1.43e-14  6.25e-06     1s
  12   4.58946916e+03  4.58940061e+03  4.08e-10 1.94e-14  2.31e-05     1s
    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time

  14   3.00601818e+03  3.00601725e+03  1.88e-09 2.34e-14  3.06e-07     1s
  13   4.58946575e+03  4.58946459e+03  2.25e-08 2.55e-14  3.90e-07     1s
     0     0 4740.30788    0  168          - 4740.30788      -     -    0s
  15   3.00601806e+03  3.00601805e+03  3.13e-10 1.42e-14  1.91e-09     1s

Barrier solved model in 15 iterations and 0.59 seconds (0.10 work units)
```

```
  14   4.58946554e+03  4.58946554e+03  1.93e-11 4.33e-14  3.90e-10     1s
Optimal objective 3.00601806e+03


Barrier solved model in 14 iterations and 0.60 seconds (0.09 work units)
Optimal objective 4.58946554e+03

Root relaxation: objective 3.006018e+03, 346 iterations, 0.57 seconds (0.06 work
units)


Root relaxation: objective 4.589466e+03, 465 iterations, 0.59 seconds (0.06 work
units)

    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time



     0     0 3006.01806    0  178          - 3006.01806      -     -    0s
    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time

     0     0 4589.46554    0  149          - 4589.46554      -     -    0s
     0     0 4811.98692    0  213          - 4811.98692      -     -    0s
H    0     0                    10774.443785 4811.98692  55.3%     -    1s
     0     0 4813.50769    0  145          - 4813.50769      -     -    1s
     0     0 3141.22826    0  216          - 3141.22826      -     -    1s
     0     2 4811.98692    0  206 10774.4438 4811.98692  55.3%     -    1s
H   35    64                    10772.718980 5033.30725  53.3%   398    2s
H   38    64                    10733.267469 5033.30725  53.1%   402    2s
H   39    64                    10598.905936 5033.30725  52.5%   398    2s
H  112   189                    10479.133548 5104.86465  51.3%   425    2s
H  115   189                    10420.691689 5104.86465  51.0%   424    2s
H  117   189                    10370.098632 5104.86465  50.8%   422    2s
H  126   189                    10291.045467 5104.86465  50.4%   413    2s
H  130   189                    10004.307592 5104.86465  49.0%   405    2s
H  164   189                    9880.6038877 5105.19832  48.3%   354    2s
H  187   189                    9784.6437660 5105.19832  47.8%   326    2s
H    0     0                    10278.907529 4813.50769  53.2%     -    2s
H    0     0                    9999.0946504 3141.22826  68.6%     -    2s
H  242   289                    9779.7817119 5105.19832  47.8%   276    2s
H  247   289                    9725.6841782 5105.19832  47.5%   274    2s
H    0     0                    9569.2372140 3141.22826  67.2%     -    2s
H    0     0                    9387.9973630 3141.22826  66.5%     -    2s
H    0     0                    9367.8778194 3141.22826  66.5%     -    3s
H  700   847                    9710.1379616 5105.19832  47.4%   154    3s
H  715   847                    9673.7636491 5105.19832  47.2%   152    3s
H  817   847                    9658.8954467 5105.19832  47.1%   145    3s
     0     2 3141.22826    0  209 9367.87782 3141.22826  66.5%     -    3s
H    0     0                    10277.677443 4813.50769  53.2%     -    3s
H    0     0                    10277.383591 4813.50769  53.2%     -    3s
H    0     0                    10277.236665 4813.50769  53.2%     -    3s
H 1042  1203                    9656.5405451 5105.19832  47.1%   128    3s
H 1067  1203                    9655.8625310 5105.19832  47.1%   127    3s
H    0     0                    10263.550032 4813.50769  53.1%     -    3s
     0     2 4813.50769    0  143 10263.5500 4813.50769  53.1%     -    3s
H 1424  1581                    9655.5308102 5105.19832  47.1%   109    3s
H 1439  1581                    8529.9588650 5105.19832  40.1%   109    3s
H   31    64                    10053.731116 4813.50769  52.1%   242    3s
H 1850  1895                    8528.4218440 5124.30241  39.9%  94.3    3s
```

H	1855	1895				8514.8445577	5124.30241	39.8%	94.1	3s
H	1912	1895				8416.9414948	5124.30241	39.1%	93.0	3s
H	1926	1895				8405.9738496	5124.30241	39.0%	92.5	3s
H	1929	1895				8294.0478026	5124.30241	38.2%	92.6	3s
H	2005	1895				8293.5861509	5124.30241	38.2%	90.8	3s
H	2022	1895				8130.2921979	5124.30241	37.0%	90.7	3s
H	92	112				9345.8091630	3375.80531	63.9%	424	3s
H	119	178				9976.6773473	4859.99907	51.3%	459	4s
H	123	178				9948.7229621	4859.99907	51.1%	451	4s
H	171	178				9918.1256604	4859.99907	51.0%	401	4s
H	2116	1980				8115.3453526	5126.92852	36.8%	89.8	4s
H	2139	1980				8017.1616048	5126.92852	36.1%	89.6	4s
H	2381	2341				8016.6141067	5126.92852	36.0%	86.6	4s
H	2388	2341				7969.2158739	5126.92852	35.7%	86.5	4s
H	2412	2341				7922.8979070	5126.92852	35.3%	86.0	4s
H	2504	2341				7821.1631986	5126.92852	34.4%	84.8	4s
*	2652	2544		76		7484.6613179	5126.92852	31.5%	82.9	4s
H	160	298				9254.3694243	3406.94708	63.2%	377	4s
H	218	298				9006.6115591	3406.94708	62.2%	306	4s
H	2808	2730				7462.3316686	5126.92852	31.3%	81.4	4s
H	2817	2730				7442.5237447	5126.92852	31.1%	81.1	4s
H	2826	2730				7402.3385703	5126.92852	30.7%	80.9	4s
H	2840	2730				7377.2339525	5126.92852	30.5%	80.8	4s
H	2841	2730				7366.2770993	5126.92852	30.4%	80.8	4s
H	303	480				9005.3744673	3406.94708	62.2%	237	4s
H	228	274				7440.5924813	4859.99907	34.7%	345	4s
	3454	2953	5166.23903	8	201	7366.27710	5132.06180	30.3%	75.2	5s
	470	514	5012.76177	10	212	7440.59248	4859.99907	34.7%	232	5s
	479	549	3578.56325	11	215	9005.37447	3406.94708	62.2%	172	5s
H	3513	2975				7307.4209631	5132.06180	29.8%	74.6	5s
H	3568	3125				7222.2621273	5132.06180	28.9%	74.3	5s
H	3572	3123				7211.5544221	5132.06180	28.8%	74.2	5s
H	3583	3118				7201.1160516	5132.06180	28.7%	74.0	5s
H	3604	3109				7168.9013839	5132.06180	28.4%	73.8	5s
H	3610	3045				7045.7121175	5132.06180	27.2%	73.8	5s
H	3613	2990				6954.3526377	5132.06180	26.2%	73.8	5s
H	3817	2982				6943.6219213	5132.50537	26.1%	71.6	5s
H	551	626				8991.3979798	3406.94708	62.1%	157	5s
H	580	626				8203.4004865	3406.94708	58.5%	152	5s
H	537	636				7431.0462867	4859.99907	34.6%	224	5s
H	4237	3307				6886.3917100	5132.50537	25.5%	68.9	5s
H	4238	3261				6812.0409933	5132.50537	24.7%	68.9	5s
H	4253	3247				6792.8071189	5132.50537	24.4%	69.1	5s
H	4290	3163				6735.9069071	5132.50537	23.8%	68.9	5s
H	4304	3142				6706.7506069	5132.50537	23.5%	68.8	5s
H	548	636				7295.2187885	4859.99907	33.4%	221	5s
H	701	739				7295.2146885	4863.93031	33.3%	203	6s
H	702	739				7256.1234671	4863.93031	33.0%	202	6s
H	4788	3354				6705.8293721	5144.73083	23.3%	67.6	6s
H	711	739				7247.9152694	4863.93031	32.9%	202	6s
H	724	739				7239.3491097	4863.93031	32.8%	201	6s
H	725	739				7236.5953297	4863.93031	32.8%	201	6s
H	728	739				7232.3212715	4863.93031	32.7%	201	6s
H	733	739				7178.9795960	4863.93031	32.2%	200	6s
H	710	722				8164.3322180	3406.94708	58.3%	138	6s
H	721	941				8162.4358332	3406.94708	58.3%	137	6s
H	726	941				8080.8471343	3406.94708	57.8%	137	6s
H	731	941				8028.7735014	3406.94708	57.6%	136	6s
H	734	941				8004.6414846	3406.94708	57.4%	136	6s
H	738	941				7963.7655527	3406.94708	57.2%	136	6s

```
H  950   941                    7177.3663355 4868.14618  32.2%   181    6s
H  970   941                    7168.1875719 4868.14618  32.1%   181    6s
H 4788  3329                    6650.9181514 5144.73083  22.6%  67.6    6s
H 1094  1213                    7930.3490206 3406.94708  57.0%   122    6s
H 1112  1213                    7920.8308712 3406.94708  57.0%   121    6s
H 1473  1543                    7848.1648433 3406.94708  56.6%   108    7s
H 1063  1024                    7166.8986795 4868.14618  32.1%   176    7s
H 1078  1024                    7165.7151752 4868.14618  32.1%   176    7s
H 1324  1307                    7161.0722628 4873.14879  31.9%   160    7s
H 1844  1833                    7800.2670963 3406.94708  56.3%  97.5    7s
H 1864  1833                    7785.3330292 3406.94708  56.2%  97.4    7s
H 1893  1833                    7712.8792052 3406.94708  55.8%  96.5    7s
H 2341  2176                    7705.0568241 3406.94708  55.8%  89.0    8s
H 1811  1617                    7157.2569171 4875.91413  31.9%   140    8s
H 1827  1617                    7135.3254102 4875.91413  31.7%   139    8s
H 2854  2722                    7667.2749730 3413.78959  55.5%  82.2    8s
H 3036  2722                    7584.2326622 3413.78959  55.0%  80.4    8s
H 1957  1766                    7133.9513280 4875.91413  31.7%   135    8s
H 2081  1766                    7133.4228726 4875.91413  31.6%   131    8s
H 3587  3340                    7581.0031026 3413.78959  55.0%  74.5    8s
H 3644  3340                    7578.5564492 3413.78959  55.0%  74.0    8s
H 3671  3340                    7577.1121629 3413.78959  54.9%  73.6    8s
H 3710  3273                    7240.9454836 3413.78959  52.9%  73.1    8s
H 3735  3266                    7203.0708069 3413.78959  52.6%  72.9    8s
H 4851  3210                    6643.3088586 5144.73083  22.6%  70.9    9s
H 4910  3148                    6640.4272653 5144.73083  22.5%  73.9    9s
H 4921  2994                    6591.4798582 5144.73083  21.9%  74.0    9s
H 5011  2883                    6523.9120806 5144.73083  21.1%  75.0    9s
H 5227  2897                    6523.2528110 5144.73083  21.1%  75.2    9s
H 5440  3004                    6510.0758588 5144.73083  21.0%  75.5    9s
  5729  3115 5227.16120   20  175 6510.07586 5144.73083  21.0%  76.8   10s
H 5808  3018                    6476.6637671 5144.73083  20.6%  76.9   10s
H 6247  3197                    6475.7808841 5144.73083  20.6%  76.6   10s
H 6267  3080                    6468.1967649 5144.73083  20.5%  76.8   10s
H 6277  2972                    6468.0490884 5144.73083  20.5%  76.8   10s
H 6285  2869                    6466.3103740 5144.73083  20.4%  76.7   10s
H 6585  2836                    6410.6240514 5144.73083  19.7%  76.1   10s
H 6878  2808                    6404.1918212 5144.73083  19.7%  76.1   10s
H 6899  2715                    6397.3101005 5144.73083  19.6%  76.2   10s
H 7125  2740                    6391.8628930 5144.73083  19.5%  76.3   10s
H 7143  2657                    6389.5525520 5144.73083  19.5%  76.4   10s
H 7155  2580                    6385.4349639 5144.73083  19.4%  76.3   10s
  3342  2642 5657.09655   23  184 7133.42287 4895.50288  31.4%   108   10s
H 7434  2639                    6362.9451200 5144.73083  19.1%  76.8   11s
H 7548  2582                    6362.7589190 5144.73083  19.1%  76.8   11s
H 8857  3110                    6356.8858692 5146.77636  19.0%  75.5   11s
H 8878  3109                    6355.9751824 5146.77636  19.0%  75.4   12s
H 8953  3175                    6354.2660677 5146.77636  19.0%  75.5   12s
H 9014  3212                    6350.3369573 5146.90624  19.0%  75.6   12s
H 9076  3265                    6311.8363317 5146.90624  18.5%  75.6   12s
H 9571  3641                    6311.5694502 5146.90624  18.5%  75.3   12s
H 9780  3680                    6310.0118291 5146.90624  18.4%  75.0   12s
H 9903  3752                    6307.8127304 5147.46538  18.4%  74.9   13s
H 9937  3751                    6304.4579966 5147.51877  18.4%  74.8   13s
H 9972  3745                    6295.5662499 5147.51877  18.2%  74.8   13s
H10594  4234                    6294.8476080 5147.76946  18.2%  74.3   13s
H10739  4224                    6278.4793820 5147.76946  18.0%  74.2   13s
H11967  4864                    6272.2182379 5149.01711  17.9%  73.6   14s
H11970  4864                    6270.4650533 5149.01711  17.9%  73.6   14s
H11989  4838                    6238.7368640 5149.01711  17.5%  73.5   14s
```

H13851	5815				6238.4883523	5151.73155	17.4%	73.2	14s
H13852	5809				6235.6266828	5151.73155	17.4%	73.2	14s
5049	4075	3791.39258	21	216	7203.07081	3413.78959	52.6%	63.5	14s
4293	3208	6879.08760	37	101	7133.42287	4901.04757	31.3%	105	15s
13962	5890	5374.86437	29	115	6235.62668	5151.73155	17.4%	73.3	15s
5051	4076	4713.37191	33	171	7203.07081	3413.78959	52.6%	63.5	15s
H14291	6075				6231.1962715	5153.34705	17.3%	73.5	15s
H15156	6519				6224.0814531	5160.06451	17.1%	73.0	16s
H18362	8236				6214.9571408	5168.88762	16.8%	74.0	17s
H19810	8999				6211.1628219	5171.87116	16.7%	73.9	17s
H 5084	3927				7134.7489720	3413.78959	52.2%	66.0	18s
H 5088	3732				7073.7805837	3413.78959	51.7%	66.3	18s
H 5095	3546				7051.6018624	3413.78959	51.6%	66.7	18s
H 5098	3370				7049.8274151	3413.78959	51.6%	66.8	18s
H 5116	3246				7027.7035812	3413.78959	51.4%	67.8	18s
H 5117	3087				6993.8780517	3413.78959	51.2%	67.8	18s
H 5148	2927				6961.2013005	3413.78959	51.0%	69.5	18s
H 5406	3024				6948.1585330	3413.78959	50.9%	72.5	18s
H 5407	2888				6928.8030533	3413.78959	50.7%	72.6	18s
H 5410	2759				6923.1331006	3413.78959	50.7%	72.7	18s
H 5416	2635				6917.0039939	3413.78959	50.6%	72.7	18s
H 5539	2608				6900.4088889	3413.78959	50.5%	72.6	19s
H 5540	2496				6899.5707219	3413.78959	50.5%	72.7	19s
H 5584	2378				6859.1773308	3413.78959	50.2%	72.6	19s
H 5824	2471				6852.6614545	3413.78959	50.2%	72.9	19s
H 5825	2376				6831.8749732	3413.78959	50.0%	72.9	19s
H 5832	2284				6787.7500757	3413.78959	49.7%	73.1	19s
H 5836	2198				6777.9450038	3413.78959	49.6%	73.0	19s
H 5838	2116				6765.4706306	3413.78959	49.5%	73.0	19s
H 5839	2039				6764.2991131	3413.78959	49.5%	73.1	19s
H 5841	1965				6758.6516593	3413.78959	49.5%	73.1	19s
H 5863	1888				6751.1612275	3413.78959	49.4%	73.1	19s
H 6073	1971				6750.4491690	3413.78959	49.4%	73.7	19s
H 6092	1902				6746.2487615	3413.78959	49.4%	73.7	19s
H 6313	1981				6745.5367029	3413.78959	49.4%	73.6	19s
6472	2092	3671.20538	27	202	6745.53670	3413.78959	49.4%	73.8	20s
H 6521	2018				6745.5306470	3413.78959	49.4%	74.0	20s
25446	12206	5664.85486	41	147	6211.16282	5179.15732	16.6%	74.6	20s
H 7550	2374				6742.7045265	3419.62768	49.3%	74.6	20s
H 8505	2895				6742.5026607	3419.62768	49.3%	73.7	21s
H 9887	3172				5666.9377980	3431.17190	39.5%	72.3	22s
H10951	3491				5320.3759853	3431.91738	35.5%	70.8	22s
H11596	3856				5295.7836074	3431.91738	35.2%	69.7	22s
H11600	3851				5287.8873234	3431.91738	35.1%	69.7	22s
H11608	3826				5268.9779955	3431.91738	34.9%	69.6	22s
H13547	5121				5226.7217997	3459.73239	33.8%	67.7	23s
H14150	5369				5226.1101528	3475.91257	33.5%	67.5	23s
H14152	5330				5187.2299911	3475.91257	33.0%	67.5	23s
H14170	5327				5186.0997014	3475.91257	33.0%	67.6	23s
H14228	5378				5184.4937168	3475.91257	33.0%	67.5	23s
H14243	5359				5172.6791520	3475.91257	32.8%	67.5	23s
H16630	7015				5166.9461755	3476.60015	32.7%	65.4	24s
35982	17747	5392.40455	31	157	6211.16282	5187.06309	16.5%	73.3	25s
18488	8028	3499.18415	30	200	5166.94618	3479.40954	32.7%	65.3	25s
H18558	8068				5164.5160209	3479.40954	32.6%	65.4	25s
H18897	8229				5154.1466315	3479.40954	32.5%	65.6	26s
H19627	8504				5074.3153602	3483.75954	31.3%	65.4	27s
H21022	9383				5073.9157173	3492.11085	31.2%	64.9	28s
H21054	9377				5072.7093756	3492.11085	31.2%	64.9	28s
4624	3395	5574.58078	23	145	7133.42287	4911.00065	31.2%	104	28s

```
 26923 13266 3632.27906   38  179 5072.70938 3497.76532  31.0%  64.9   30s
 52147 25998 5408.50233   47  107 6211.16282 5191.79698  16.4%  72.7   30s
  4628  3398 5978.06145   24  188 7133.42287 4911.00065  31.2%   104   31s
H28584 13693                    5040.1953598 3510.43062  30.4%  65.4   31s
H32062 15873                    5007.6166029 3512.82451  29.9%  65.0   32s
H 5005  3502                    6953.2677975 4911.00065  29.4%   117   34s
H 5069  3367                    6939.6565988 4911.00065  29.2%   117   34s
  5510  3788 6017.13429   25   46 6939.65660 4911.00065  29.2%   121   35s
H 5604  3603                    6938.6482204 4911.00065  29.2%   122   35s
H 5626  3450                    6938.6002023 4911.00065  29.2%   122   35s
 65301 32945 6102.50982   37  115 6211.16282 5195.83293  16.3%  72.8   35s
 35126 16939 4841.31559   50  117 5007.61660 3513.78657  29.8%  65.5   35s
H 5705  3332                    6938.5521843 4911.00065  29.2%   121   35s
H 5715  3196                    6930.3102758 4911.00065  29.1%   121   35s
H 5727  3067                    6891.6030242 4911.00065  28.7%   122   35s
H 5775  3029                    6890.2103649 4911.00065  28.7%   121   35s
H 5787  2913                    6888.3788518 4911.00065  28.7%   121   35s
H 5821  2795                    6885.8231650 4911.00065  28.7%   122   35s
H35244 16963                    5005.0524295 3513.78657  29.8%  65.5   35s
H35852 17767                    5004.9868192 3513.78657  29.8%  65.5   36s
H 6994  3026                    6876.9692208 4911.00065  28.6%   121   36s
H 7410  3084                    6874.4471134 4911.00065  28.6%   119   36s
H 7859  3116                    6863.7019740 4912.48188  28.4%   118   37s
H 7865  3027                    6755.9191486 4912.48188  27.3%   118   37s
H 8342  3021                    6755.6765764 4912.48188  27.3%   117   37s
H 8418  2916                    6755.6231600 4913.73252  27.3%   117   37s
H39649 19856                    5004.9042316 3514.70284  29.8%  65.8   38s
H39664 19779                    4990.0465216 3514.70284  29.6%  65.8   38s
H44335 23428                    4989.5603948 3522.62845  29.4%  65.6   39s
H11011  3994                    6594.9039866 4948.01135  25.0%   113   39s
H11104  4152                    6582.2513565 4949.37626  24.8%   113   39s
H11148  4140                    6576.6238298 4949.37626  24.7%   114   39s
H11200  4136                    6572.0981087 4949.37626  24.7%   114   39s
H11381  4135                    6571.3369029 4949.37626  24.7%   113   39s
 11451  4554 5078.99074   21  125 6571.33690 4949.37626  24.7%   113   40s
 46089 23967     cutoff   54      4989.56039 3525.39574  29.3%  65.3   40s
 83195 41692 5526.69135   41   95 6211.16282 5199.81833  16.3%  73.3   40s
H12990  5050                    6570.4719867 4961.10051  24.5%   110   40s
H13077  5050                    6570.2246948 4961.15920  24.5%   110   40s
H13102  5137                    6568.0704239 4961.15920  24.5%   110   41s
 16065  6754 6017.14824   38  138 6568.07042 4968.89098  24.3%   108   45s
 55528 29913 4240.67039   43  111 4989.56039 3529.32320  29.3%  65.3   45s
 98522 49160 5262.28789   32  153 6211.16282 5202.41256  16.2%  73.7   46s
H99045 49160                    6211.1628206 5203.06909  16.2%  73.8   46s
 18487  8229 5164.44325   24  174 6568.07042 4972.03385  24.3%   106   50s
 105775 53013 5238.58190   35  180 6211.16282 5205.21539  16.2%  73.8   50s
 59533 32574 3579.45288   30  197 4989.56039 3533.83158  29.2%  65.5   51s
 23318 11246 5465.90558   33  202 6568.07042 4993.32552  24.0%   103   55s
 65819 35447 4144.03217   37  149 4989.56039 3534.83924  29.2%  65.0   56s
 110073 63697 6191.92625   60   58 6211.16282 5206.06467  16.2%  73.9   60s
 30750 15500 5389.97847   28  162 6568.07042 5003.59487  23.8%   102   60s
 74231 40730 3688.68974   34  194 4989.56039 3539.17534  29.1%  64.8   60s
 82961 45190 4198.96188   35  155 4989.56039 3542.20001  29.0%  64.5   65s
 141634 69875 5239.86928   33  170 6211.16282 5210.44054  16.1%  74.0   65s
 36338 19543 5841.91425   51   77 6568.07042 5010.74246  23.7%   102   66s
 40815 21496 6175.71347   42   88 6568.07042 5013.37685  23.7%   103   70s
 87165 47146 4231.25684   39  141 4989.56039 3543.87787  29.0%  64.4   70s
 155182 75370 5353.92954   32  158 6211.16282 5212.31322  16.1%  73.9   71s
 162912 79666 5266.04538   26  169 6211.16282 5212.66928  16.1%  74.0   75s
 49654 26373 5957.03220   32  125 6568.07042 5016.42996  23.6%   103   76s
```

```
 95655 50295 4222.28137   48  119 4989.56039 3545.98505  28.9%  64.4   77s
 54259 29183     cutoff   40      6568.07042 5025.21452  23.5%   103   80s
 177636 87719 5569.80852   40  111 6211.16282 5214.20779  16.1%  74.2   81s
 100134 54542 3625.85420   30  186 4989.56039 3546.30543  28.9%  64.0   81s
 189709 92404 6130.08158   45   87 6211.16282 5216.11883  16.0%  74.5   85s
 59932 32378     cutoff   64      6568.07042 5025.58652  23.5%   103   86s
 108642 59935 4097.12596   35  137 4989.56039 3549.71790  28.9%  64.0   87s
 203145 98269 5855.46131   45   73 6211.16282 5217.45443  16.0%  74.5   90s
 65207 34712 6399.14300   42  102 6568.07042 5027.36930  23.5%   103   90s
 112516 61769 4138.79072   38  146 4989.56039 3550.51718  28.8%  64.1   90s
 215207 104339 5236.96444   31  148 6211.16282 5218.39221  16.0%  74.4   96s
 120074 66186 4543.07737   44  141 4989.56039 3550.65030  28.8%  64.0   96s
 71590 39043 6087.00898   39  141 6568.07042 5036.72318  23.3%   103   96s
 124823 68950 4955.91600   38  152 4989.56039 3553.05277  28.8%  64.0  100s
 77686 42380 5630.93857   27  137 6568.07042 5037.96004  23.3%   103  101s
 222994 110460     cutoff   53      6211.16282 5219.29242  16.0%  74.4  101s
 239764 115175     cutoff   38      6211.16282 5220.66936  15.9%  74.6  105s
 83318 45825 5962.28876   40  150 6568.07042 5040.12895  23.3%   103  106s
 135421 74478 3562.26543   33  212 4989.56039 3554.90961  28.8%  63.8  107s
 140744 77579 3577.65310   34  205 4989.56039 3556.03516  28.7%  63.6  110s
 86537 47054 infeasible   46      6568.07042 5041.26808  23.2%   103  111s
 252489 120931 5507.20021   36  122 6211.16282 5221.62949  15.9%  74.7  111s
 262418 126490 5852.66477   36  135 6211.16282 5222.45977  15.9%  74.8  115s
 149735 81513 4729.88458   47  155 4989.56039 3558.09171  28.7%  63.7  116s
 92501 52247 5246.05654   26  206 6568.07042 5043.76234  23.2%   102  117s
 97953 54796 5217.24688   24  232 6568.07042 5045.12009  23.2%   102  120s
 277667 132346 5751.37439   45   78 6211.16282 5223.63422  15.9%  74.8  121s
 157994 85646 4916.84269   49  120 4989.56039 3558.43484  28.7%  63.6  123s
 282507 135573 5332.63407   36  129 6211.16282 5223.87636  15.9%  74.8  126s
 105099 56307 6112.42755   35  112 6568.07042 5048.22244  23.1%   102  126s
 162062 88204 4757.56035   44  149 4989.56039 3560.49344  28.6%  63.7  127s
 292311 138100 5303.99639   34  163 6211.16282 5224.11503  15.9%  75.0  130s
 167152 90406 4348.85219   43   91 4989.56039 3560.85606  28.6%  63.8  131s
H168434 90406                    4989.5583921 3561.24896  28.6%  63.8  131s
 108263 60516 5547.36270   27  163 6568.07042 5049.72721  23.1%   102  132s
 171398 92674 4213.74622   37  109 4989.55839 3561.24896  28.6%  63.9  135s
H172280 92674                    4989.5583899 3561.24896  28.6%  63.9  135s
 113100 62265 5949.98650   40  125 6568.07042 5051.42406  23.1%   102  136s
 117631 65268 5286.58376   29  210 6568.07042 5052.91261  23.1%   102  140s
 184477 99116 4250.95465   40  140 4989.55839 3564.69352  28.6%  64.1  141s
 188877 100001 4516.41665   41  129 4989.55839 3564.74232  28.6%  64.0  145s
 292884 160172 5585.25653   41  115 6211.16282 5224.42050  15.9%  75.1  145s
H329453 160172                    6211.1628194 5224.65759  15.9%  75.0  145s
 125113 70248 5138.51828   29  182 6568.07042 5054.13209  23.0%   102  147s
 129322 73960 5059.65811   27  220 6568.07042 5055.07180  23.0%   101  151s
 190690 106544 4744.72191   53   84 4989.55839 3564.93027  28.6%  64.0  152s
 136512 76422 infeasible   42      6568.07042 5057.18124  23.0%   102  155s
 202427 108690 3694.68532   36  201 4989.55839 3565.54711  28.5%  63.9  156s
 206277 110632 3702.68333   37  199 4989.55839 3565.54711  28.5%  63.9  160s
 341234 181336 5962.69735   40   79 6211.16282 5226.86848  15.8%  75.0  162s
 145147 79755 5741.23300   31  154 6568.07042 5057.48607  23.0%   102  162s
H342000 181336                    6211.1628158 5226.86848  15.8%  75.0  162s
 389234 182475 5295.18184   38  151 6211.16282 5229.15354  15.8%  74.7  165s
 214286 114338 4029.33392   41  119 4989.55839 3566.78600  28.5%  63.9  169s
H391770 182588                    6211.1628147 5229.15354  15.8%  74.7  169s
 147370 85475 6370.50731   53   86 6568.07042 5059.18189  23.0%   102  170s
 392018 190426 5276.61962   29  202 6211.16281 5229.15354  15.8%  74.7  173s
H392407 190426                    6211.1628136 5229.15354  15.8%  74.7  173s
 216576 118086 3585.50794   34  224 4989.55839 3567.05244  28.5%  64.0  174s
 159751 89441 5608.41495   32  177 6568.07042 5062.04087  22.9%   101  175s
```

```
 408722 190947 5281.89155   37  166 6211.16281 5230.33496  15.8%  74.8  175s
H409707 190947                    6211.1628120 5230.37523  15.8%  74.7  175s
 223768 120378 3687.21565   35  182 4989.55839 3567.88539  28.5%  64.0  178s
H418423 195839                    6211.1628110 5230.64183  15.8%  74.6  179s
 422852 196991 5607.70307   39  122 6211.16281 5230.75167  15.8%  74.6  180s
 228841 120860 3687.28050   36  179 4989.55839 3568.49490  28.5%  63.9  180s
 169233 92232 5381.15834   33  151 6568.07042 5063.76602  22.9%   101  180s
H424472 196991                    6211.1628099 5230.83859  15.8%  74.6  180s

Cutting planes:
  Learned: 13

  Gomory: 44
  Lift-and-project: 1
  Cover: 256
Cutting planes:
  Learned: 5
  Gomory: 38
  Implied bound: 73
  Lift-and-project: 3
  MIR: 399
  Cover: 588
  Mixing: 13
  Implied bound: 144
  StrongCG: 1
  Projected implied bound: 2
  Flow cover: 1690
  MIR: 496
  Flow path: 3
  GUB cover: 2
  Mixing: 20
  Inf proof: 30
  Flow cover: 1409
  Flow path: 8
  Zero half: 3
  GUB cover: 3
  Network: 4
  Inf proof: 55
  RLT: 73
  Zero half: 9

  Relax-and-lift: 199
  Network: 5
  RLT: 78
Cutting planes:
  Relax-and-lift: 84
  BQP: 1

  Learned: 7

Explored 425482 nodes (31745888 simplex iterations) in 180.23 seconds (128.02 wor
k units)
Explored 170231 nodes (17231442 simplex iterations) in 180.19 seconds (128.49 wor
k units)
  Gomory: 6
Thread count was 48 (of 144 available processors)
Thread count was 48 (of 144 available processors)
  Cover: 447

  Implied bound: 155
```

```
Solution count 10: 6211.16 6211.16 6211.16 ... 6238.49

  Clique: 3
Time limit reached
Best objective 6.211162809653e+03, best bound 5.230874419464e+03, gap 15.7827%

  MIR: 455
Solution count 10: 6568.07 6570.22 6570.47 ... 6755.68
  Mixing: 30

  Flow cover: 1363
Time limit reached
  Flow path: 8
Best objective 6.568070423884e+03, best bound 5.065108432331e+03, gap 22.8829%
  GUB cover: 8
  Inf proof: 51
  Zero half: 4
  Network: 3
Discarded solution information
  RLT: 124
  Relax-and-lift: 150
Discarded solution information

Explored 229866 nodes (14701444 simplex iterations) in 180.30 seconds (117.54 wor
k units)
Thread count was 48 (of 144 available processors)

Solution count 10: 4989.56 4989.56 4989.56 ... 5040.2

Time limit reached
Best objective 4.989557395378e+03, best bound 3.568494902926e+03, gap 28.4807%
Discarded solution information
Set parameter Threads to value 48
Set parameter MIPFocus to value 3
Set parameter Heuristics to value 0.2
Set parameter Cuts to value 2
Set parameter CutPasses to value 2
Set parameter Presolve to value 2
Set parameter Symmetry to value 2
Set parameter Method to value 3
Set parameter ConcurrentMIP to value 2
Set parameter Seed to value 42
Set parameter MIPGap to value 0.01
Set parameter TimeLimit to value 43020
Gurobi Optimizer version 12.0.3 build v12.0.3rc0 (linux64 - "Ubuntu 22.04.3 LTS")

CPU model: Intel(R) Xeon(R) Platinum 8352V CPU @ 2.10GHz, instruction set [SSE2|A
VX|AVX2|AVX512]
Thread count: 72 physical cores, 144 logical processors, using up to 48 threads

Non-default parameters:
TimeLimit  43020
MIPGap  0.01
Method  3
Heuristics  0.2
MIPFocus  3
Symmetry  2
Cuts  2
CutPasses  2
ConcurrentMIP  2
```

```
Presolve  2
Seed  42
Threads  48

Optimize a model with 1718 rows, 868 columns and 6572 nonzeros
Model fingerprint: 0xbbbe6802
Variable types: 328 continuous, 540 integer (540 binary)
Coefficient statistics:
  Matrix range     [5e-01, 5e+02]
  Objective range  [5e-02, 4e+00]
  Bounds range     [1e+00, 5e+02]
  RHS range        [1e+00, 5e+02]
Using branch priorities.

Concurrent MIP optimizer: 2 concurrent instances (24 threads per instance)


Loaded user MIP start with objective 6568.07

Set parameter Threads to value 48
Set parameter MIPFocus to value 3
Set parameter Heuristics to value 0.2
Set parameter Cuts to value 2
Set parameter CutPasses to value 2
Set parameter Presolve to value 2
Set parameter Symmetry to value 2
Set parameter Method to value 3
Set parameter ConcurrentMIP to value 2
Presolve removed 190 rows and 60 columns
Set parameter Seed to value 42
Presolve time: 0.04s
Set parameter MIPGap to value 0.01
Set parameter TimeLimit to value 43020
Presolved: 1528 rows, 808 columns, 6391 nonzeros
Variable types: 318 continuous, 490 integer (490 binary)
Gurobi Optimizer version 12.0.3 build v12.0.3rc0 (linux64 - "Ubuntu 22.04.3 LTS")

CPU model: Intel(R) Xeon(R) Platinum 8352V CPU @ 2.10GHz, instruction set [SSE2|A
VX|AVX2|AVX512]
Thread count: 72 physical cores, 144 logical processors, using up to 48 threads

Non-default parameters:
TimeLimit  43020
MIPGap  0.01
Method  3
Heuristics  0.2
Root relaxation presolve removed 2 rows and 0 columns
MIPFocus  3
Symmetry  2
Cuts  2
CutPasses  2
ConcurrentMIP  2
Root relaxation presolved: 1526 rows, 808 columns, 6363 nonzeros
Presolve  2

Seed  42
Concurrent LP optimizer: primal simplex, dual simplex, and barrier
Threads  48
Showing barrier log only...
```

```
Optimize a model with 1716 rows, 872 columns and 6582 nonzeros
Model fingerprint: 0x5b596a18
Root barrier log...
Variable types: 328 continuous, 544 integer (544 binary)
Coefficient statistics:
  Matrix range     [5e-01, 3e+02]
  Objective range  [5e-02, 4e+00]
  Bounds range     [1e+00, 3e+02]
  RHS range        [1e+00, 3e+02]

Using branch priorities.

Concurrent MIP optimizer: 2 concurrent instances (24 threads per instance)


Loaded user MIP start with objective 4989.56

Set parameter Threads to value 48
Presolve removed 158 rows and 47 columns
Presolve time: 0.04s
Presolved: 1558 rows, 825 columns, 6406 nonzeros
Ordering time: 0.05s

Variable types: 321 continuous, 504 integer (504 binary)
Barrier performed 0 iterations in 0.14 seconds (0.05 work units)
Barrier solve interrupted - model solved by another algorithm


Solved with dual simplex
Root relaxation presolve removed 1 rows and 0 columns

Set parameter MIPFocus to value 3
Root relaxation: objective 4.589466e+03, 513 iterations, 0.08 seconds (0.01 work
units)
Root relaxation presolved: 1557 rows, 825 columns, 6382 nonzeros

Concurrent LP optimizer: primal simplex, dual simplex, and barrier
Showing barrier log only...

Root barrier log...

Set parameter Heuristics to value 0.2
Set parameter Cuts to value 2
Set parameter CutPasses to value 2
Set parameter Presolve to value 2
Set parameter Symmetry to value 2
Set parameter Method to value 3
Set parameter ConcurrentMIP to value 2
Set parameter Seed to value 42
Set parameter MIPGap to value 0.01

Set parameter TimeLimit to value 43020
    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time

     0     0      -    -    -     6568.07042 4589.46554  30.1%     -    0s
Gurobi Optimizer version 12.0.3 build v12.0.3rc0 (linux64 - "Ubuntu 22.04.3 LTS")

CPU model: Intel(R) Xeon(R) Platinum 8352V CPU @ 2.10GHz, instruction set [SSE2|A
```

```
VX|AVX2|AVX512]
Thread count: 72 physical cores, 144 logical processors, using up to 48 threads

Non-default parameters:
TimeLimit  43020
MIPGap  0.01
Method  3
Ordering time: 0.05s
Heuristics  0.2

MIPFocus  3
Barrier performed 0 iterations in 0.13 seconds (0.06 work units)
Symmetry  2
Barrier solve interrupted - model solved by another algorithm
Cuts  2

CutPasses  2

ConcurrentMIP  2
Solved with dual simplex
Presolve  2
Seed  42

Threads  48
Root relaxation: objective 3.006018e+03, 563 iterations, 0.08 seconds (0.02 work
units)

Optimize a model with 1716 rows, 871 columns and 6581 nonzeros
Model fingerprint: 0xd79b1d1c
Variable types: 328 continuous, 543 integer (543 binary)
Coefficient statistics:
  Matrix range     [5e-01, 4e+02]
  Objective range  [5e-02, 4e+00]
  Bounds range     [1e+00, 4e+02]
  RHS range        [1e+00, 4e+02]
Using branch priorities.

Concurrent MIP optimizer: 2 concurrent instances (24 threads per instance)


Loaded user MIP start with objective 6211.16


    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time

     0     0      -    -    -     4989.55740 3006.01806  39.8%     -    0s
Presolve removed 200 rows and 87 columns
Presolve time: 0.03s
Presolved: 1516 rows, 784 columns, 6237 nonzeros
Variable types: 320 continuous, 464 integer (464 binary)
Root relaxation presolve removed 1 rows and 0 columns
Root relaxation presolved: 1515 rows, 784 columns, 6213 nonzeros

Concurrent LP optimizer: primal simplex, dual simplex, and barrier
Showing barrier log only...

Root barrier log...

Ordering time: 0.05s
```

```
Barrier performed 0 iterations in 0.13 seconds (0.05 work units)
Barrier solve interrupted - model solved by another algorithm


Solved with dual simplex

Root relaxation: objective 4.740308e+03, 586 iterations, 0.07 seconds (0.02 work
units)
     0     0      -    -    -     6568.07042 4738.81646  27.9%     -    0s

    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time
     0     0      -    -    -     6568.07042 4738.81646  27.9%     -    0s

     0     0      -    -    -     6211.16281 4740.30788  23.7%     -    0s
     0     0      -    -    -     6568.07042 4738.81646  27.9%     -    0s
     0     0      -    -    -     6568.07042 4743.18738  27.8%     -    0s
     0     0      -    -    -     4989.55740 3216.15975  35.5%     -    0s
     0     0      -    -    -     4989.55740 3270.50712  34.5%     -    0s
     0     0      -    -    -     4989.55740 3276.81967  34.3%     -    0s
     0     0      -    -    -     4989.55740 3277.02182  34.3%     -    0s
     0     0      -    -    -     4989.55740 3277.02182  34.3%     -    0s
     0     0      -    -    -     4989.55740 3279.64428  34.3%     -    0s
     0     0      -    -    -     4989.55740 3279.64428  34.3%     -    0s
     0     0      -    -    -     6568.07042 4768.14784  27.4%     -    0s
     0     0      -    -    -     6211.16281 4993.81423  19.6%     -    0s
     0     0      -    -    -     6211.16281 4997.59052  19.5%     -    0s
     0     0      -    -    -     6211.16281 4998.23224  19.5%     -    0s
     0     0      -    -    -     6211.16281 4998.23224  19.5%     -    0s
     0     0      -    -    -     4989.55740 3330.49596  33.3%     -    0s
     0     2      -    -    -     6568.07042 4772.74216  27.3%     -    0s
     0     2      -    -    -     4989.55740 3330.49596  33.3%     -    0s
     0     0      -    -    -     6211.16281 5056.78597  18.6%     -    0s
     0     2      -    -    -     6211.16281 5058.70649  18.6%     -    0s
   588   555      -    -    -     6568.07042 5327.99382  18.9%     -    5s
   711   700      -    -    -     4989.55740 3835.97428  23.1%     -    5s
   994  1002      -    -    -     6211.16280 5421.89641  12.7%     -    5s
  1301  1276      -    -    -     6568.07042 5491.55292  16.4%     -   10s
  2771  2546      -    -    -     4989.55740 3958.35927  20.7%     -   10s
  2761  2555      -    -    -     6211.16280 5505.16017  11.4%     -   14s
  3287  2834      -    -    -     4989.55740 4011.59760  19.6%     -   15s
  2799  2591      -    -    -     6568.07042 5601.22844  14.7%     -   15s
  2766  2558      -    -    -     6211.16280 5505.16017  11.4%     -   15s
  3146  2787      -    -    -     6568.07042 5601.22844  14.7%     -   20s
  3496  2913      -    -    -     6211.16280 5558.09536  10.5%     -   20s
  6549  4574      -    -    -     4989.55740 4198.59205  15.9%     -   20s
  3795  3084      -    -    -     6568.07042 5739.86811  12.6%     -   25s
  6394  4212      -    -    -     6211.16280 5692.69409  8.35%     -   25s
  8585  5534      -    -    -     4989.55740 4360.43716  12.6%     -   25s
 13984  8771      -    -    -     4989.55740 4469.43757  10.4%     -   30s
  6283  4311      -    -    -     6568.07042 5861.20221  10.8%     -   30s
 11658  6822      -    -    -     6211.16280 5796.26086  6.68%     -   30s
  9568  5219      -    -    -     6568.07042 5950.15783  9.41%     -   35s
 20460 14174      -    -    -     4989.55740 4514.99038  9.51%     -   35s
 16455  9588      -    -    -     6204.32009 5840.50427  5.86%     -   35s
 30211 22100      -    -    -     4985.21542 4556.00900  8.61%     -   40s
 12843  6666      -    -    -     6568.07042 6015.28848  8.42%     -   40s
 20496 11884      -    -    -     6204.32009 5877.90884  5.26%     -   40s
 15069  7923      -    -    -     6568.07042 6061.36042  7.71%     -   45s
```

40944	31079	-	-	-	4981.88855	4586.99679	7.93%	-	45s
26188	16659	-	-	-	6204.32009	5914.76205	4.67%	-	46s
19440	10601	-	-	-	6568.07042	6129.15332	6.68%	-	50s
50595	39023	-	-	-	4981.88855	4609.65781	7.47%	-	50s
34831	20346	-	-	-	6204.32009	5953.32081	4.05%	-	51s
27216	15816	-	-	-	6568.07042	6185.47563	5.83%	-	55s
41569	24713	-	-	-	6204.32009	5975.48533	3.69%	-	55s
61668	47732	-	-	-	4981.88855	4628.75127	7.09%	-	56s
32971	20257	-	-	-	6568.07042	6212.40794	5.42%	-	60s
49987	29993	-	-	-	6204.32009	5993.75303	3.39%	-	60s
71566	55242	-	-	-	4981.88855	4643.39811	6.79%	-	61s
57504	33765	-	-	-	6204.32009	6007.74195	3.17%	-	65s
79356	61846	-	-	-	4981.88855	4652.07464	6.62%	-	65s
39432	25165	-	-	-	6568.07042	6232.12170	5.11%	-	66s
64907	39785	-	-	-	6204.32009	6017.92049	3.00%	-	70s
91683	68021	-	-	-	4981.88855	4666.19257	6.34%	-	70s
47088	29508	-	-	-	6568.07042	6252.51354	4.80%	-	71s
95055	71780	-	-	-	4976.82093	4669.76571	6.17%	-	75s
72941	43185	-	-	-	6204.32009	6026.56522	2.87%	-	75s
51836	33802	-	-	-	6568.07042	6256.10087	4.75%	-	76s
104410	78649	-	-	-	4976.82093	4677.94549	6.01%	-	80s
56684	36444	-	-	-	6568.07042	6265.55197	4.61%	-	80s
79728	49053	-	-	-	6204.32009	6029.80164	2.81%	-	81s
114324	84404	-	-	-	4976.82093	4687.78620	5.81%	-	85s
65915	41391	-	-	-	6568.07042	6288.51490	4.26%	-	85s
89195	53998	-	-	-	6204.32009	6041.06891	2.63%	-	85s
97205	58338	-	-	-	6204.32009	6047.97085	2.52%	-	90s
125802	91037	-	-	-	4976.82093	4697.18868	5.62%	-	91s
74547	45347	-	-	-	6568.07042	6302.84156	4.04%	-	91s
134501	96713	-	-	-	4976.82093	4704.37457	5.47%	-	95s
104844	62952	-	-	-	6204.32009	6053.51350	2.43%	-	96s
79015	49047	-	-	-	6568.07042	6308.86112	3.95%	-	97s
85585	52017	-	-	-	6568.07042	6317.14930	3.82%	-	100s
145027	103237	-	-	-	4973.71854	4711.78488	5.27%	-	101s
111865	66368	-	-	-	6204.32009	6056.83458	2.38%	-	101s
117584	69183	-	-	-	6204.32009	6061.56496	2.30%	-	105s
151972	107547	-	-	-	4973.71854	4716.62874	5.17%	-	105s
93141	55279	-	-	-	6568.07042	6326.06622	3.68%	-	106s
98452	58133	-	-	-	6568.07042	6331.59971	3.60%	-	110s
160370	112762	-	-	-	4973.71854	4723.68322	5.03%	-	110s
126965	73875	-	-	-	6204.32009	6067.10358	2.21%	-	110s
170163	116312	-	-	-	4973.71854	4729.47795	4.91%	-	115s
104313	61402	-	-	-	6568.07042	6335.20815	3.55%	-	116s
136497	78203	-	-	-	6204.32009	6072.14665	2.13%	-	116s
110733	64188	-	-	-	6568.07042	6340.23261	3.47%	-	120s
177130	118824	-	-	-	4960.56223	4738.33758	4.48%	-	121s
141171	80133	-	-	-	6204.32009	6074.27876	2.10%	-	120s
186908	123582	-	-	-	4946.68662	4743.30026	4.11%	-	125s
117781	67647	-	-	-	6568.07042	6348.97952	3.34%	-	126s
148922	84484	-	-	-	6204.32009	6077.67560	2.04%	-	126s
123927	70556	-	-	-	6568.07042	6354.30385	3.25%	-	130s
197885	128685	-	-	-	4940.83085	4750.89907	3.84%	-	130s
157400	88518	-	-	-	6204.32009	6081.35143	1.98%	-	132s
129753	73632	-	-	-	6568.07042	6359.95261	3.17%	-	135s
208188	123548	-	-	-	4931.08693	4754.57502	3.58%	-	135s
161050	90006	-	-	-	6204.32009	6083.01619	1.96%	-	135s
168540	90188	-	-	-	6196.07746	6084.96180	1.79%	-	140s
138861	77446	-	-	-	6568.07042	6366.93207	3.06%	-	140s
220234	127653	-	-	-	4931.08693	4760.08921	3.47%	-	141s
226359	129276	-	-	-	4929.64323	4765.85302	3.32%	-	145s

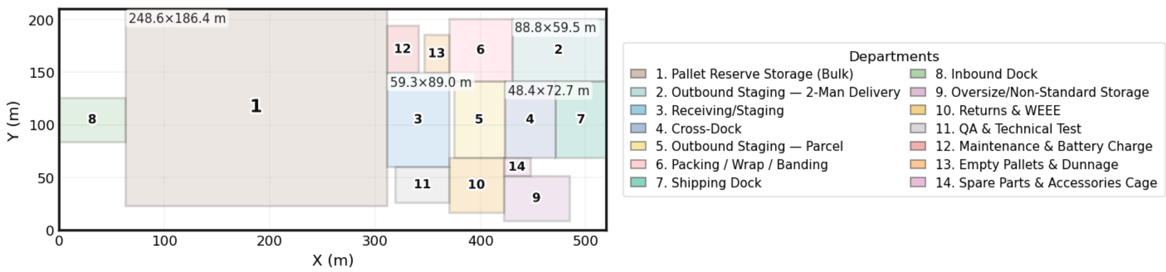
```
 145063 79860      -    -    -    6568.07042 6371.12527  3.00%     -  145s
 176128 94017      -    -    -    6196.07746 6088.91346  1.73%     -  146s
 149129 82019      -    -    -    6568.07042 6372.04725  2.98%     -  150s
 182222 96487      -    -    -    6196.07746 6091.29943  1.69%     -  150s
 237869 132967      -    -    -    4929.64323 4772.16925  3.19%     -  150s
 156176 84856      -    -    -    6568.07042 6377.51980  2.90%     -  155s
 191014 100056      -    -    -    6196.07746 6093.96228  1.65%     -  156s
 246602 135747      -    -    -    4929.64323 4776.67468  3.10%     -  156s
 199057 102758      -    -    -    6196.07746 6096.82611  1.60%     -  160s
 255453 138461      -    -    -    4929.64323 4781.07744  3.01%     -  161s
 163054 88818      -    -    -    6568.07042 6382.44265  2.83%     -  161s
 265003 141486      -    -    -    4929.64323 4785.18571  2.93%     -  165s
 206498 105558      -    -    -    6196.07746 6099.39399  1.56%     -  165s
 169283 91500      -    -    -    6568.07042 6385.81502  2.77%     -  165s
 270691 141575      -    -    -    4926.51107 4787.66916  2.82%     -  170s
 212411 108052      -    -    -    6196.07746 6101.53413  1.53%     -  170s
 175137 94165      -    -    -    6568.07042 6388.74656  2.73%     -  170s
 183970 97254      -    -    -    6568.07042 6392.15765  2.68%     -  175s
 276878 143108      -    -    -    4926.51107 4791.06390  2.75%     -  175s
 221406 110777      -    -    -    6196.07746 6104.20124  1.48%     -  176s
 227265 112867      -    -    -    6196.07746 6106.77080  1.44%     -  180s
 190060 100783      -    -    -    6568.07042 6396.26536  2.62%     -  180s
 287227 146426      -    -    -    4926.51107 4794.26700  2.68%     -  181s
 296878 148797      -    -    -    4926.51107 4796.47014  2.64%     -  185s
 235046 115080      -    -    -    6196.07746 6108.88190  1.41%     -  185s
 197735 104110      -    -    -    6568.07042 6399.84748  2.56%     -  186s
 201312 105660      -    -    -    6568.07042 6401.26568  2.54%     -  190s
 307869 151728      -    -    -    4926.51107 4799.63839  2.58%     -  191s
 244577 117677      -    -    -    6196.07746 6113.09129  1.34%     -  191s
 316137 153967      -    -    -    4926.51107 4800.41097  2.56%     -  195s
 252895 119909      -    -    -    6196.07746 6114.66288  1.31%     -  196s
 207857 109924      -    -    -    6568.07042 6404.15377  2.50%     -  200s
 261095 121959      -    -    -    6196.07746 6117.62721  1.27%     -  200s
 326245 156766      -    -    -    4926.51107 4804.00432  2.49%     -  201s
 335697 158679      -    -    -    4926.51107 4806.48295  2.44%     -  205s
 268300 124210      -    -    -    6196.07746 6120.10760  1.23%     -  206s
 219304 113322      -    -    -    6568.07042 6408.57853  2.43%     -  206s
 343550 154072      -    -    -    4917.08983 4808.26730  2.21%     -  210s
 225367 115812      -    -    -    6568.07042 6411.15447  2.39%     -  211s
 274944 125931      -    -    -    6196.07739 6123.20054  1.18%     -  211s
 231575 118372      -    -    -    6568.07042 6413.54894  2.35%     -  215s
 352109 155242      -    -    -    4916.86409 4810.36442  2.17%     -  215s
 284061 127972      -    -    -    6196.07739 6126.00331  1.13%     -  216s
 288282 128836      -    -    -    6196.07739 6128.20024  1.10%     -  220s
 363921 157790      -    -    -    4910.04860 4812.13829  1.99%     -  221s
 239533 121530      -    -    -    6568.07042 6416.50533  2.31%     -  221s
 245724 123773      -    -    -    6568.07042 6418.92702  2.27%     -  225s
 372397 159429      -    -    -    4909.85131 4814.10831  1.95%     -  225s
 295685 130261      -    -    -    6196.07738 6128.48078  1.09%     -  225s
 253906 126856      -    -    -    6568.07042 6421.65435  2.23%     -  230s
 381346 160865      -    -    -    4909.85131 4816.14073  1.91%     -  230s
 303926 131444      -    -    -    6196.07738 6131.57575  1.04%     -  231s
 390242 162337      -    -    -    4909.85131 4817.92813  1.87%     -  235s
 310245 132456      -    -    -    6196.07738 6133.31047  1.01%     -  235s

Cutting planes:
  Learned: 11
  Gomory: 43
  Lift-and-project: 1
  Cover: 1673
```

```
  Implied bound: 348
  Projected implied bound: 7
  MIR: 482
  Mixing: 19
  StrongCG: 1
  Flow cover: 1539
  Flow path: 34
  GUB cover: 2
  Inf proof: 239
  Zero half: 4
  Network: 15
  RLT: 98
  Relax-and-lift: 89
  BQP: 1
  PSD: 1

Instance 1 was solved
 261835 130206      -    -    -     6568.07042 6424.55694  2.19%     -  236s

Explored 262091 nodes (19584977 simplex iterations) in 237.38 seconds (134.48 wor
k units)
Thread count was 24 (of 144 available processors)

Solution count 10: 6196.08 6196.08 6196.08 ... 6211.16

Optimal solution found (tolerance 1.00e-02)
Best objective 6.196077457655e+03, best bound 6.134205234914e+03, gap 0.9986%
[L-shaped] status=OPTIMAL, incumbent=6196.077457654751, bound=6134.20523491404, g
ap=0.009985696690119722
[14-dept L-shaped] status=2, obj=6196.077457654751, gap=0.009985696690119722
 270606 132542      -    -    -     6568.07042 6427.20372  2.14%     -  240s
 402035 164507      -    -    -     4909.85131 4819.82805  1.83%     -  240s
 280619 137296      -    -    -     6568.07042 6429.99704  2.10%     -  245s
 414714 167156      -    -    -     4909.85131 4822.86143  1.77%     -  245s
 425316 169173      -    -    -     4909.85131 4824.30921  1.74%     -  250s
 290973 140252      -    -    -     6568.07042 6432.84212  2.06%     -  250s
 297637 143414      -    -    -     6568.07042 6434.57828  2.03%     -  255s
 438798 171846      -    -    -     4909.85131 4826.25785  1.70%     -  255s
 446508 173197      -    -    -     4909.85131 4828.10861  1.66%     -  260s
 309861 147359      -    -    -     6568.07042 6438.32293  1.98%     -  260s
 459303 175005      -    -    -     4909.85131 4830.31374  1.62%     -  265s
 320330 151330      -    -    -     6568.07042 6441.01174  1.93%     -  265s
 331821 155292      -    -    -     6568.07042 6443.25240  1.90%     -  270s
 473420 177043      -    -    -     4909.85131 4832.67944  1.57%     -  270s
 484337 178744      -    -    -     4909.85131 4834.59029  1.53%     -  275s
 343471 159788      -    -    -     6568.07042 6445.47352  1.87%     -  276s
 498502 180380      -    -    -     4909.85131 4837.18812  1.48%     -  280s
 354332 163151      -    -    -     6568.07042 6447.81173  1.83%     -  280s
 507500 181512      -    -    -     4909.85131 4838.76970  1.45%     -  285s
 359517 164136      -    -    -     6568.07042 6448.53929  1.82%     -  285s
 516000 182460      -    -    -     4909.85131 4839.64116  1.43%     -  290s
 367696 168043      -    -    -     6568.07042 6450.31734  1.79%     -  291s
 530784 184294      -    -    -     4909.85131 4842.53282  1.37%     -  295s
 378611 170968      -    -    -     6568.07042 6452.51654  1.76%     -  296s
 542695 185329      -    -    -     4909.85131 4844.42177  1.33%     -  300s
 389290 174368      -    -    -     6568.07042 6454.50554  1.73%     -  301s
 554232 186384      -    -    -     4909.85131 4846.25478  1.30%     -  305s
 396047 177341      -    -    -     6568.07042 6455.78388  1.71%     -  307s
 405613 179593      -    -    -     6568.07042 6457.31832  1.69%     -  310s
 563513 187378      -    -    -     4909.85131 4847.55066  1.27%     -  311s
```

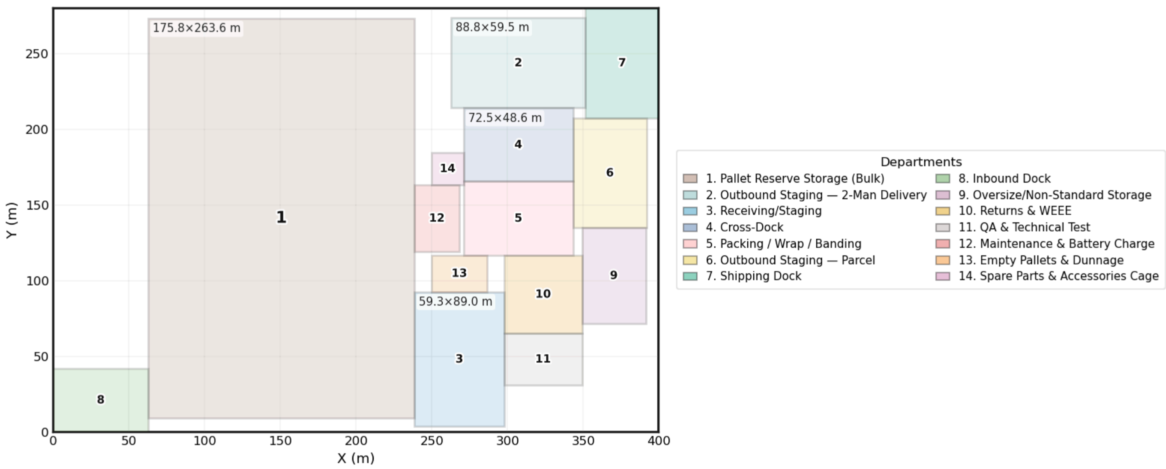
```
 418480 183588      -    -    -     6568.07042 6459.48925  1.65%     -  315s
 576664 188232      -    -    -     4909.85131 4849.50505  1.23%     -  316s
 586169 188620      -    -    -     4909.85131 4850.89067  1.20%     -  320s
 430452 187160      -    -    -     6568.07042 6461.43744  1.62%     -  320s
 439890 190288      -    -    -     6568.07042 6462.89843  1.60%     -  325s
 600110 189364      -    -    -     4909.85131 4852.78541  1.16%     -  326s
 452045 193767      -    -    -     6568.07042 6464.68460  1.57%     -  330s
 610925 189813      -    -    -     4909.85131 4854.22522  1.13%     -  330s
 461860 197009      -    -    -     6552.88217 6466.21463  1.32%     -  335s
 622980 190410      -    -    -     4909.85131 4855.88987  1.10%     -  335s

Cutting planes:
  Learned: 20
  Gomory: 74
  Lift-and-project: 3
  Cover: 1752
  Implied bound: 240
  Projected implied bound: 13
  Clique: 3
  MIR: 558
  Mixing: 11
  StrongCG: 2
  Flow cover: 1659
  Flow path: 46
  GUB cover: 5
  Inf proof: 233
  Zero half: 2
  Network: 17
  RLT: 92
  Relax-and-lift: 146

Instance 1 was solved

Explored 424723 nodes (36426263 simplex iterations) in 336.98 seconds (226.96 wor
k units)
Thread count was 24 (of 144 available processors)

Solution count 4: 6508.18 6521.41 6552.88 6568.07

Optimal solution found (tolerance 1.00e-02)
Best objective 6.508180200095e+03, best bound 6.466984627975e+03, gap 0.6330%
[I-shaped] status=OPTIMAL, incumbent=6508.1802000947755, bound=6466.984627974997,
gap=0.006329814303417467
[14-dept I-shaped] status=2, obj=6508.1802000947755, gap=0.006329814303417467
 634421 190981      -    -    -     4909.85131 4857.30962  1.07%     -  340s
 648599 191382      -    -    -     4909.85131 4859.11619  1.03%     -  345s
 662960 191420      -    -    -     4909.85131 4860.85056  1.00%     -  350s
 682193 190791      -    -    -     4909.85131 4863.18562  0.95%     -  355s
 699449 190156      -    -    -     4909.85131 4865.25271  0.91%     -  360s
 714836 189828      -    -    -     4909.85131 4866.89141  0.87%     -  365s

Cutting planes:
  Learned: 26
  Gomory: 9
  Cover: 2108
  Implied bound: 345
  Projected implied bound: 5
  Clique: 2
  MIR: 566
  Mixing: 31
```

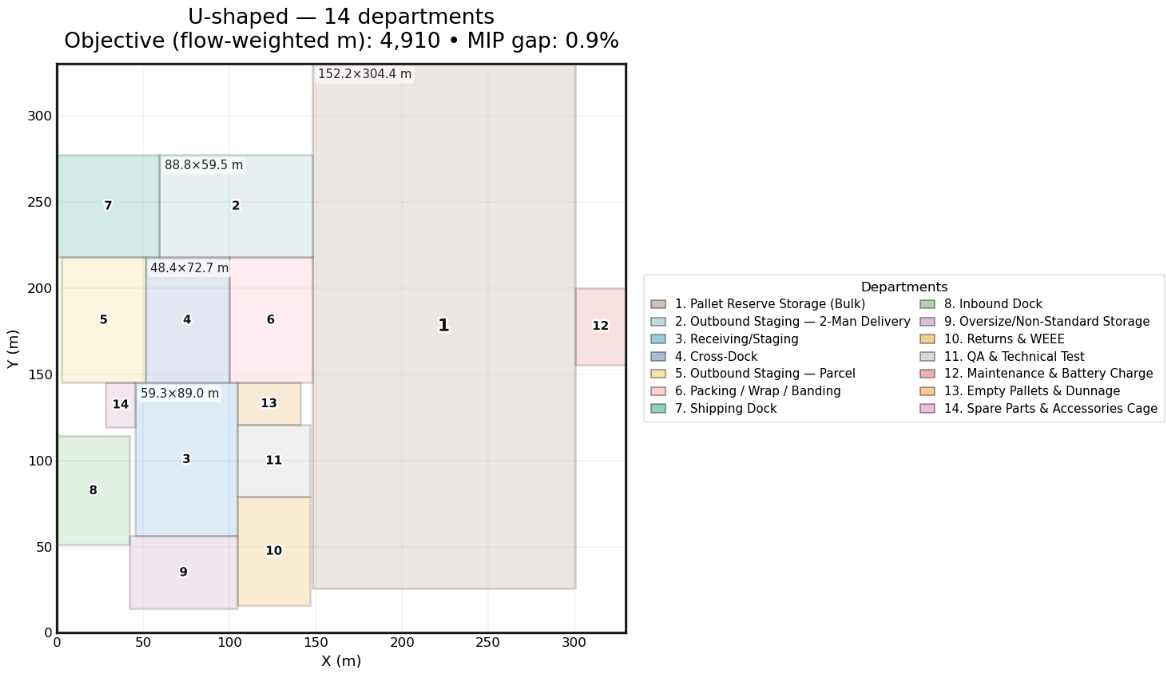
```
  StrongCG: 1
  Flow cover: 1486
  Flow path: 37
  GUB cover: 14
  Inf proof: 419
  Zero half: 2
  Network: 12
  RLT: 158
  Relax-and-lift: 109
  BQP: 1

Instance 0 was solved

Explored 723010 nodes (37493547 simplex iterations) in 367.57 seconds (229.85 wor
k units)
Thread count was 24 (of 144 available processors)

Solution count 10: 4909.85 4909.85 4910.05 ... 4926.51

Optimal solution found (tolerance 1.00e-02)
Best objective 4.909851310839e+03, best bound 4.867743929020e+03, gap 0.8576%
[U-shaped] status=OPTIMAL, incumbent=4909.851310838567, bound=4867.743929019887,
gap=0.008576101220361478
[14-dept U-shaped] status=2, obj=4909.851310838567, gap=0.008576101220361478
I-shaped             → Objective = 6,508.2 | Gap = 0.63%
L-shaped             → Objective = 6,196.1 | Gap = 1.00%
U-shaped             → Objective = 4,909.9 | Gap = 0.86%
```

I-shaped — 14 departments
Objective (flow-weighted m): 6,508 • MIP gap: 0.6%



L-shaped — 14 departments
Objective (flow-weighted m): 6,196 • MIP gap: 1.0%





Full 14-department layouts — summary and managerial implications

Restoring the full problem with all 14 departments (re-splitting the two outbound stagings) and re-solving the I/L/U footprints yields a clear ranking: the **U-shaped** layout attains the lowest flow-weighted travel distance at **4,909.9 m** (MIP gap **0.86%**), followed by the **L-shaped** at **6,196.1 m** (gap **1.00%**) and the **I-shaped** at **6,508.2 m** (gap **0.63%**). Thus, U-shape improves expected travel by **≈20.8%** relative to L-shape and **≈24.6%** relative to I-shape. The performance advantage is consistent with the geometry: co-locating inbound and outbound along the same wall shortens high-frequency links among **Receiving/Staging**, **Packing/Wrap/Banding**, the two **Outbound Staging** zones, and the **Shipping Dock**, while keeping the large **Pallet Reserve Storage (Bulk)** contiguous yet close enough to staging to limit replenishment distance. From a managerial perspective, the **U-shaped footprint is the preferred baseline** because it minimizes internal trucking and simplifies oversight along a single dock face; however, it concentrates apron activity and may require deliberate door assignment, one-way yard circulation, and peak-wave staging controls to mitigate congestion. If operational policy or site access demands stronger separation of inbound and outbound traffic, the **L-shaped** alternative provides that segregation at a modest travel penalty compared with I-shape and should be considered when cross-traffic risk, safety, or noise zoning dominate. Before committing to construction drawings, we recommend a robustness check on adjacency weights and department areas (e.g., ±10–20%) and a follow-on door-to-department and aisle-network design, but the sub-1% MIP gaps indicate these conclusions are **near-optimal** and reliable for strategic layout selection.