Forging a Persistent Identity: Multi-Camera Tracking with OpenCV and Re-ID

A technical deep-dive into building robust tracking systems that recognize individuals across time, occlusions, and camera boundaries.

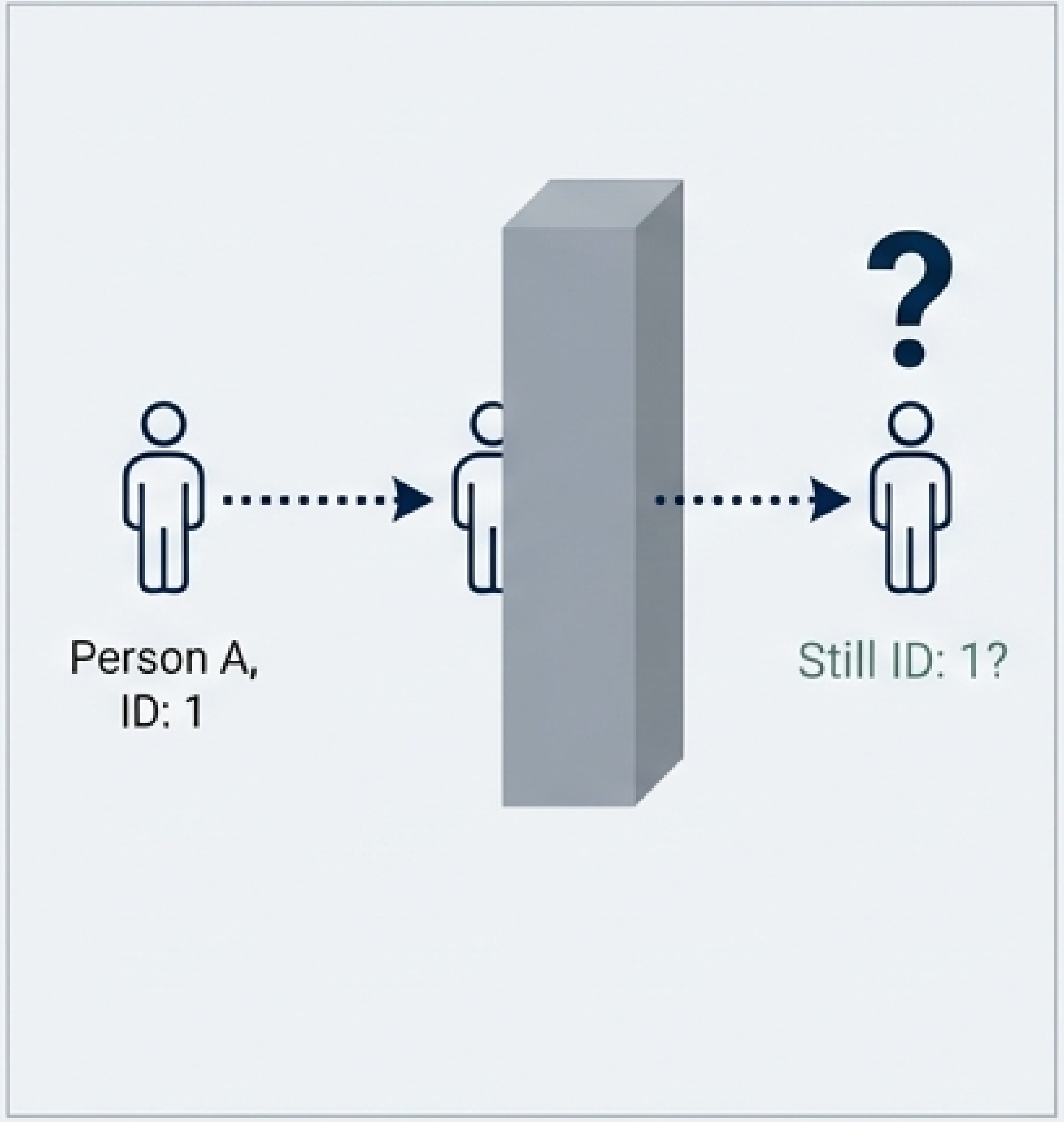
OpenCV

NotebookLM

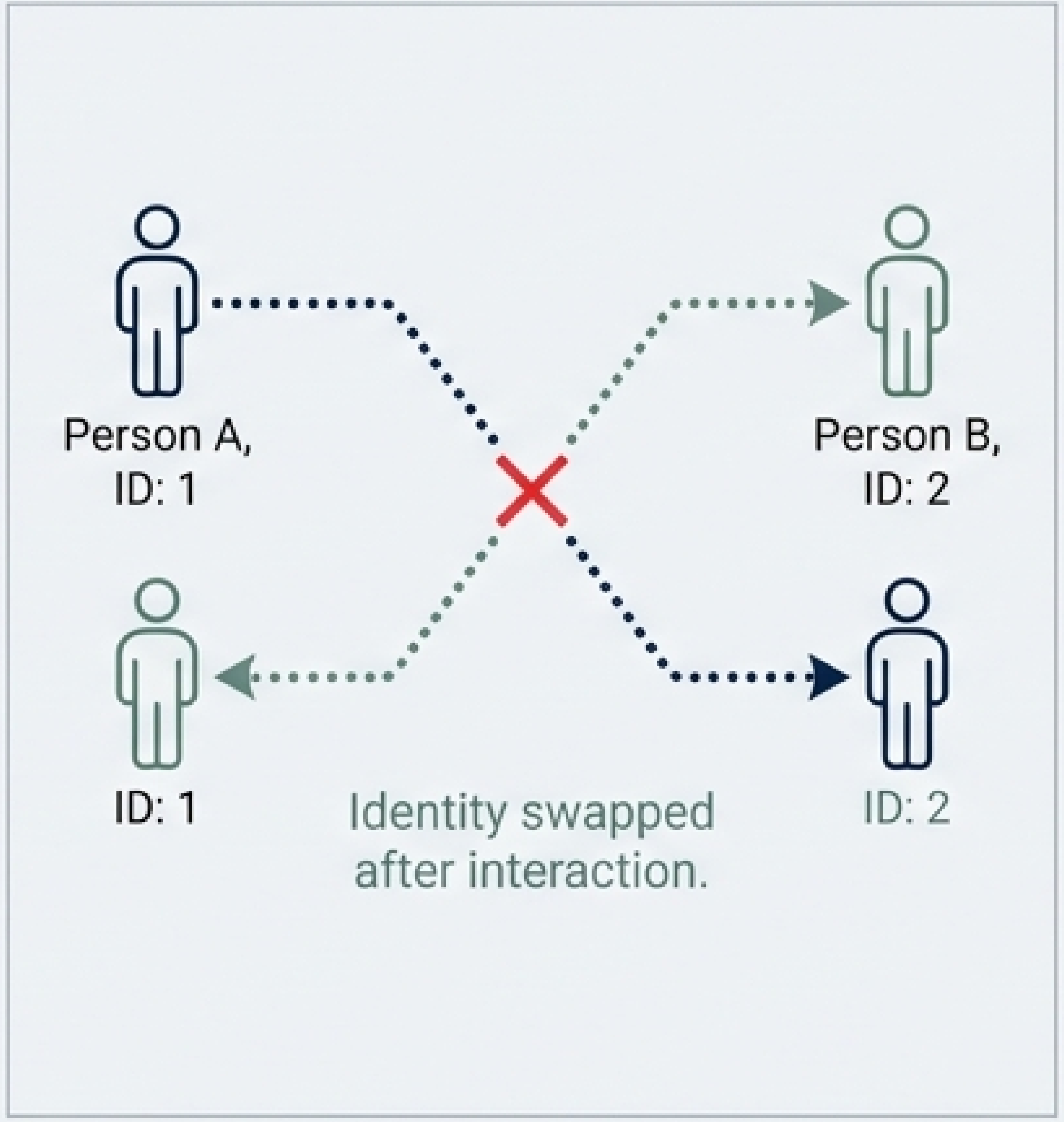
The Core Challenge: Maintaining a Single, Unbroken ID

How can we assign an unbreakable identity to a person as they navigate a complex environment?

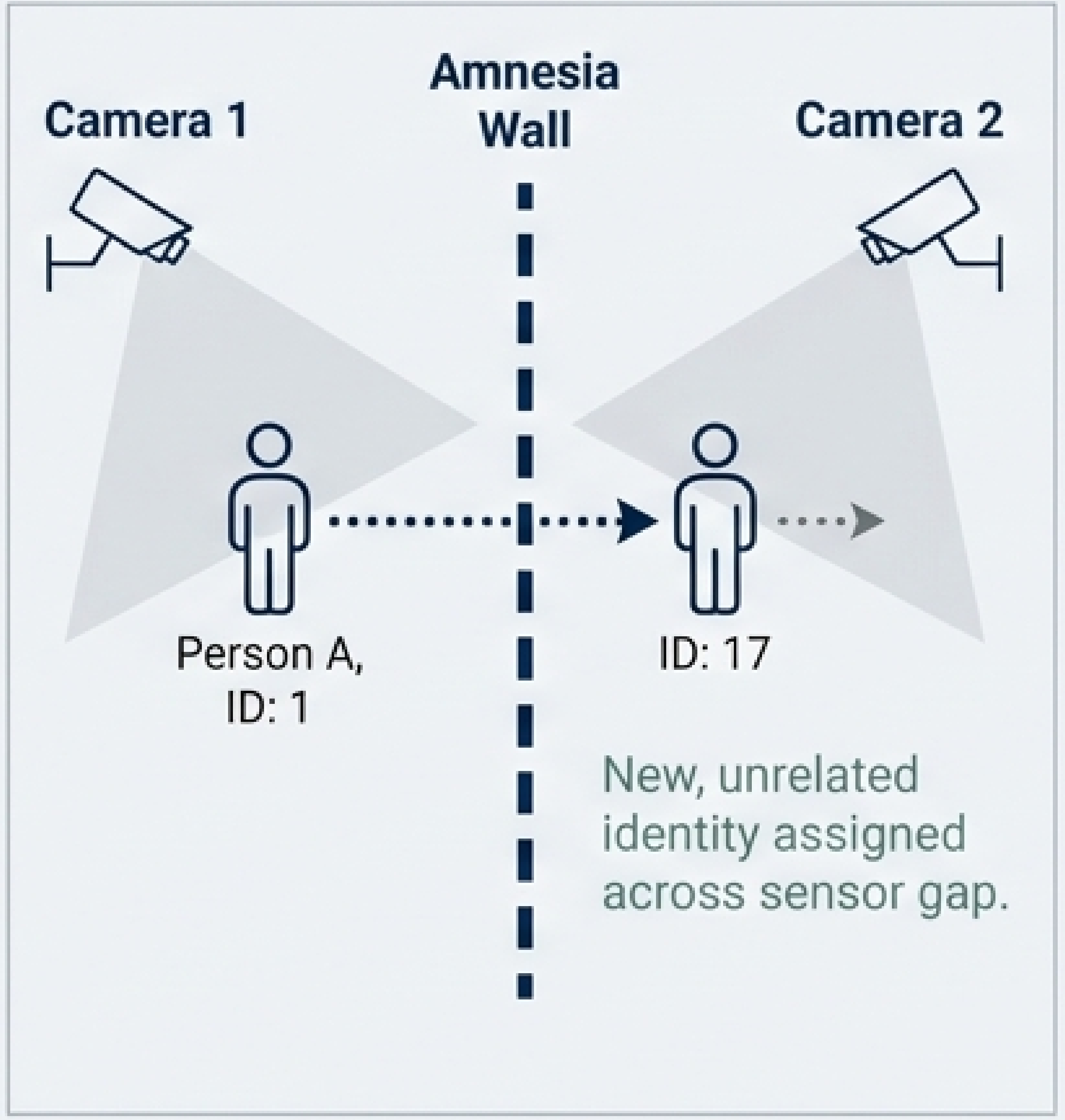
Occlusion



ID Switch



Camera Hand-off

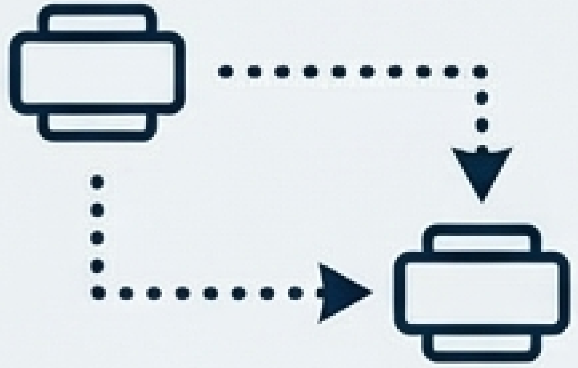


The Foundational Trade-off: Detection vs. Tracking

Detection: Accurate but Expensive

- Independent frames
- More computation
- No ID persistence
- Handles new objects

Tracking: Efficient but Fragile



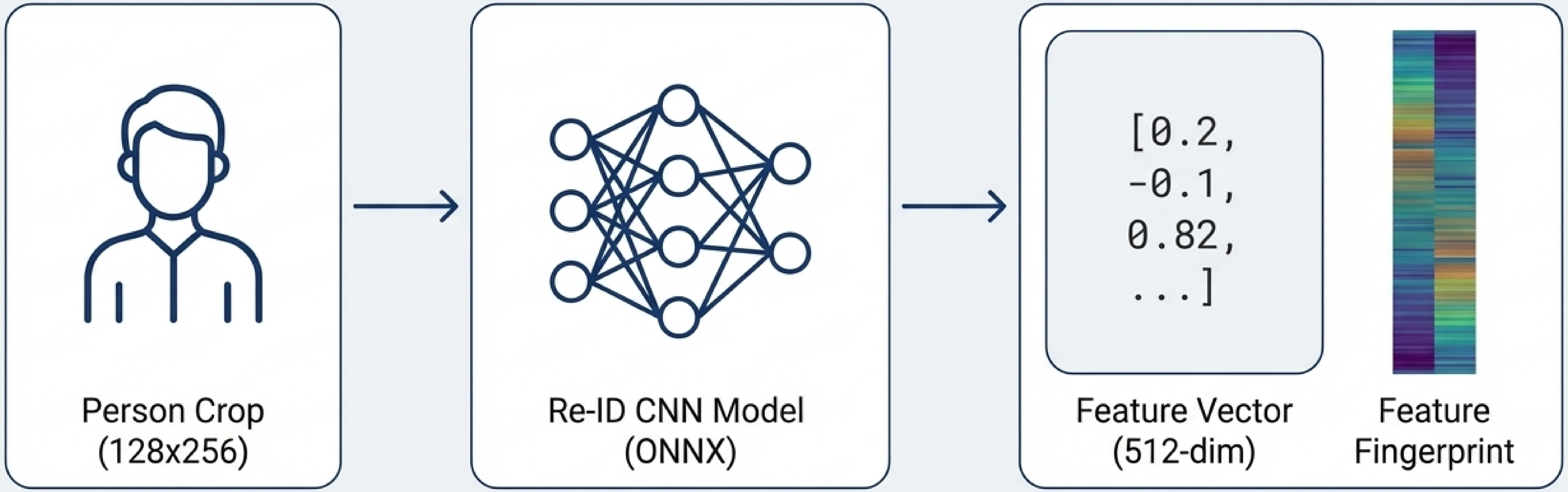
- Temporal continuity
- Less computation
- ID persistence
- Can lose track

The best practice is ‘Tracking-by-Detection.”

Frame N	Frame N+1	Frame N+2	Frame N+3
Detect + Assign IDs	Track (fast)	Track (fast)	Detect + Re-associate IDs

The Game Changer: Person Re-Identification (Re-ID)

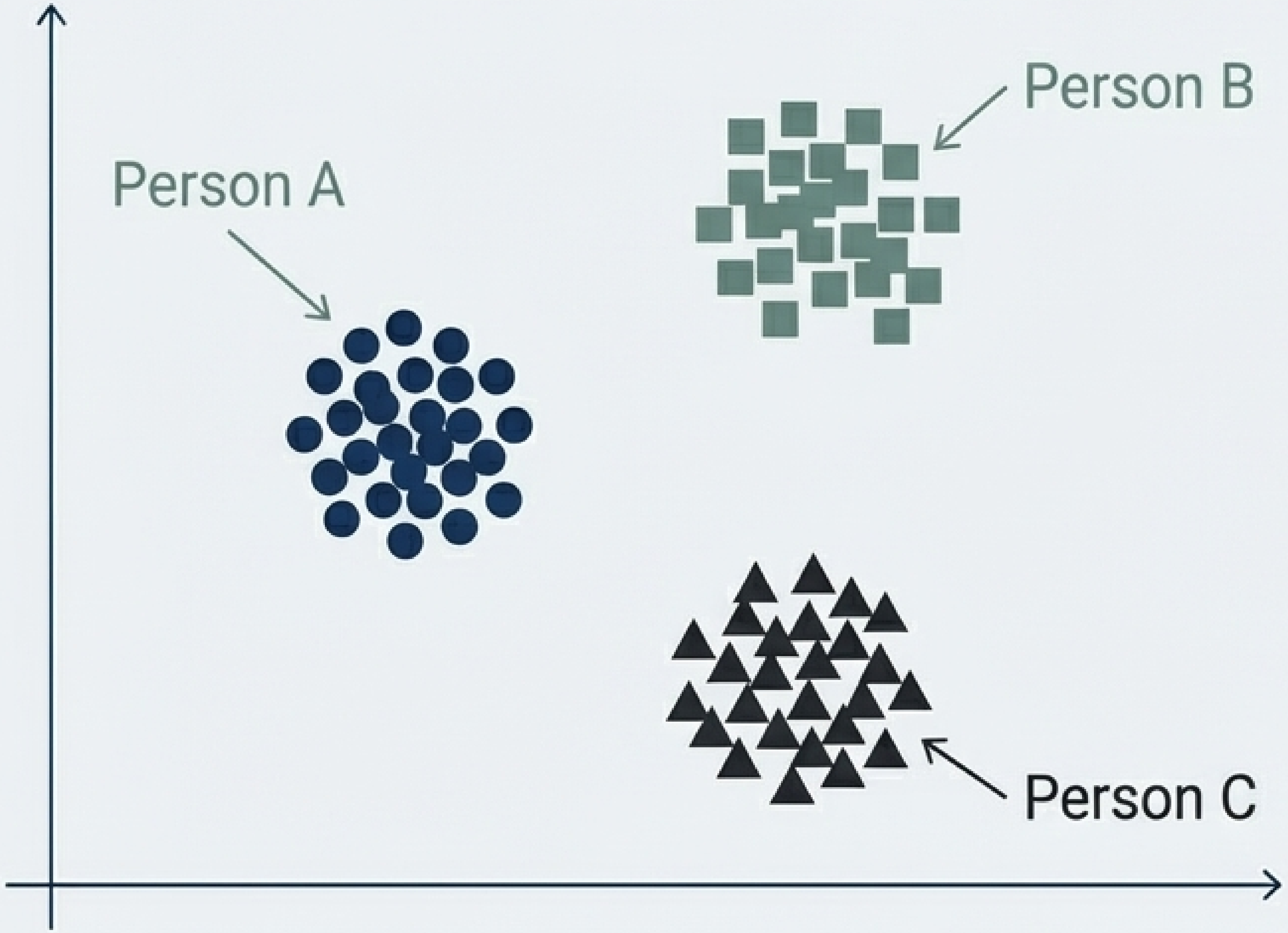
Person Re-ID is the task of matching the same person across different images, camera views, or time instances based on their unique **appearance features**.



The model converts a person’s visual appearance into a mathematical signature.

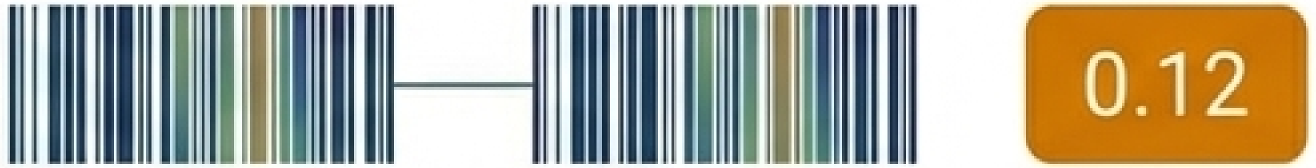
Quantifying Appearance: The Feature Embedding Space

Feature Space Visualization

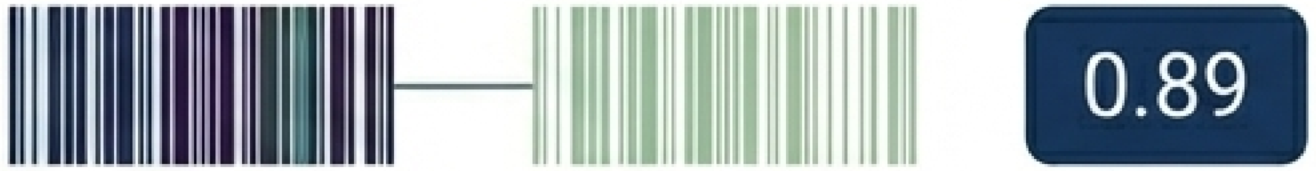


The Decision Metric: Cosine Distance

$$\text{distance}(A, B) = 1 - \text{similarity}(A, B)$$



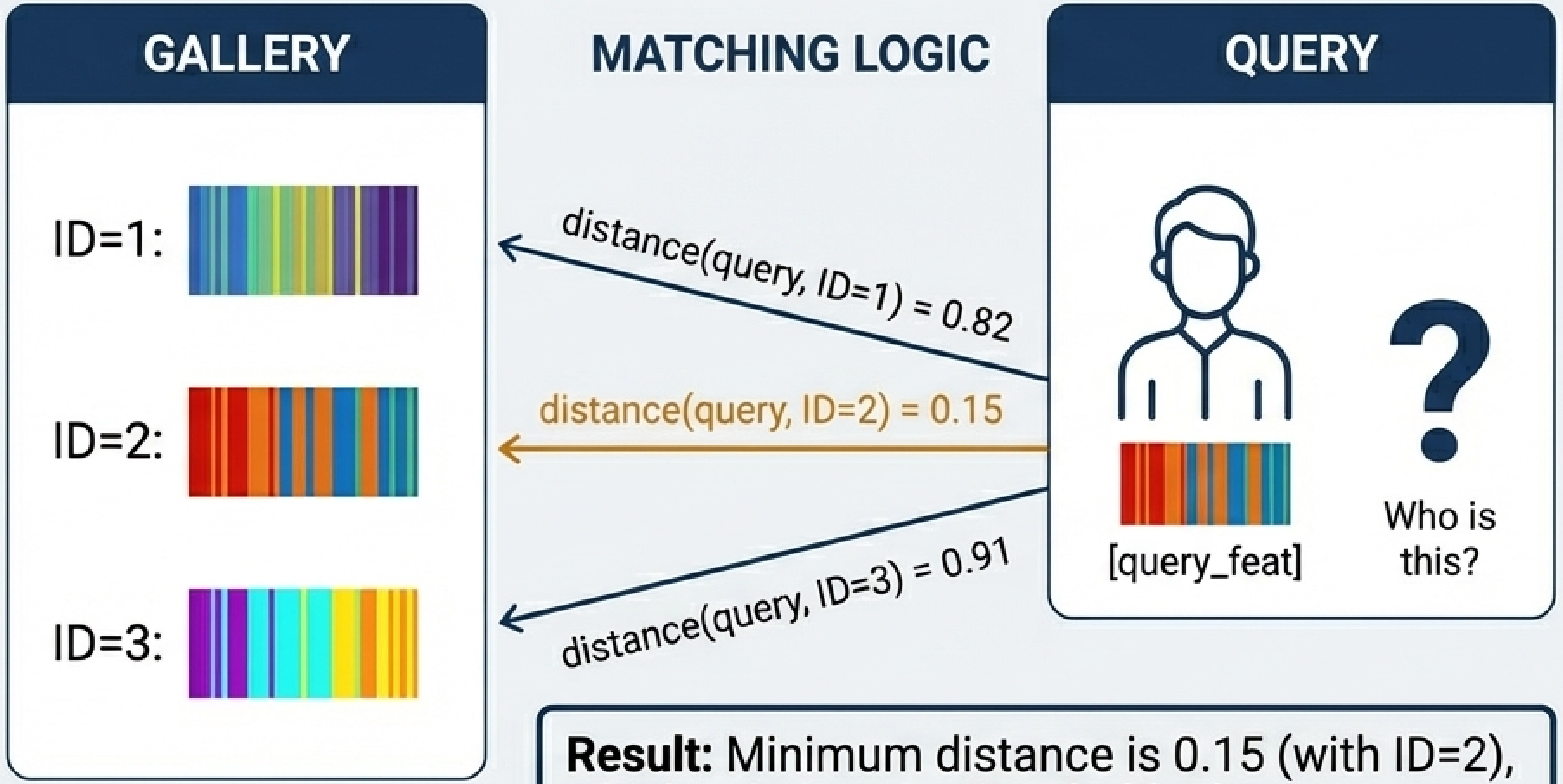
distance < 0.5 → Likely SAME person



distance > 0.7 → Likely DIFFERENT persons

Core Principle: Same person = Close in embedding space. Different persons = Far apart.

The Matching Process: Querying the Gallery of Known Identities

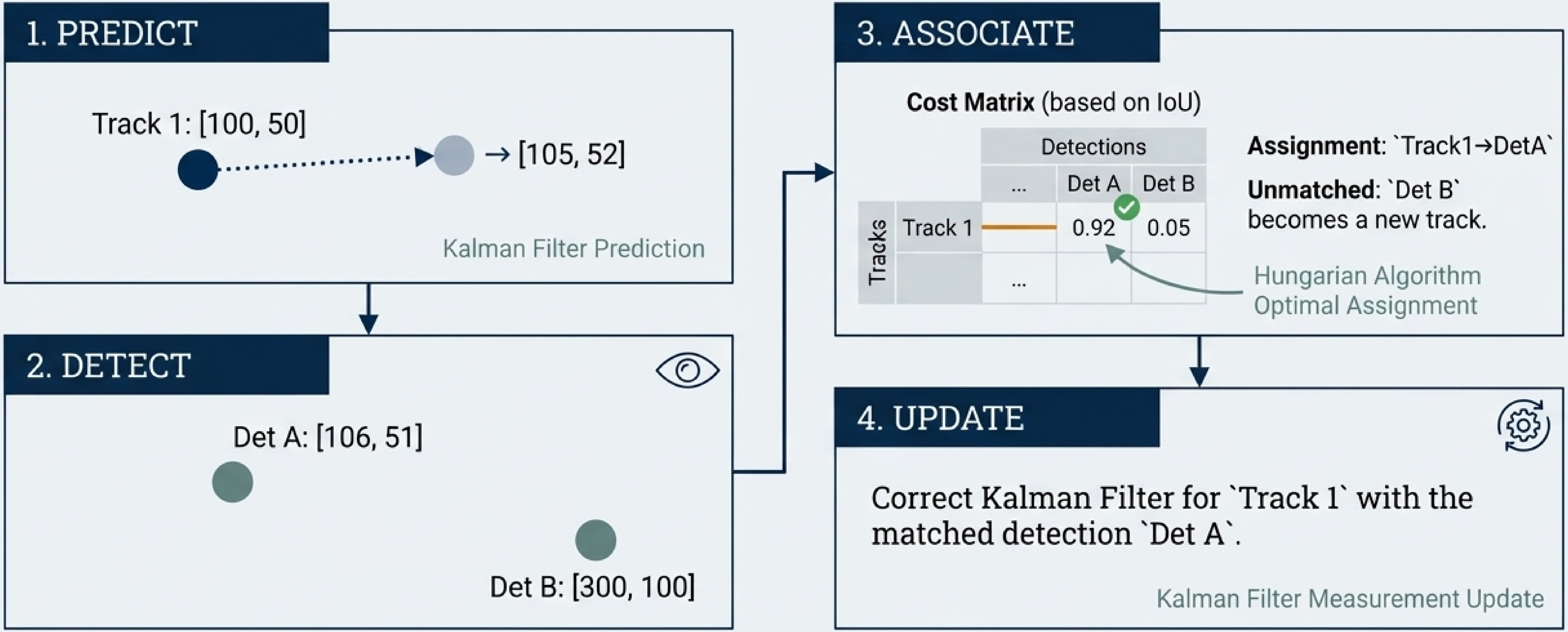


Result: Minimum distance is 0.15 (with ID=2), which is below the threshold. The Query is identified as ID=2.

1. **Compare:** Compute cosine distance from [query_feat] to every feature in the Gallery.
2. **Identify:** Find the minimum distance.
3. **Decide:** If min_distance < threshold, associate with that ID. Otherwise, create a new ID.

Building the Tracker, Part 1: Motion-Based Association with SORT

The Simple Online Realtime Tracking (SORT) algorithm uses a Kalman Filter for motion prediction and the Hungarian algorithm for association based on spatial overlap (IoU).



The Upgrade: Fusing Motion and Appearance with DeepSORT

$$\text{Cost Matrix} = \lambda_1 \times IoU_{\text{cost}} + \lambda_2 \times Appearance_{\text{cost}}$$

0.85	0.10
0.15	0.70

IoU Cost Matrix
(Spatial overlap)

+

0.20	0.90
0.95	0.25

Appearance Cost Matrix
(Re-ID distance)

Low overall cost due to good appearance match, despite mediocre IoU

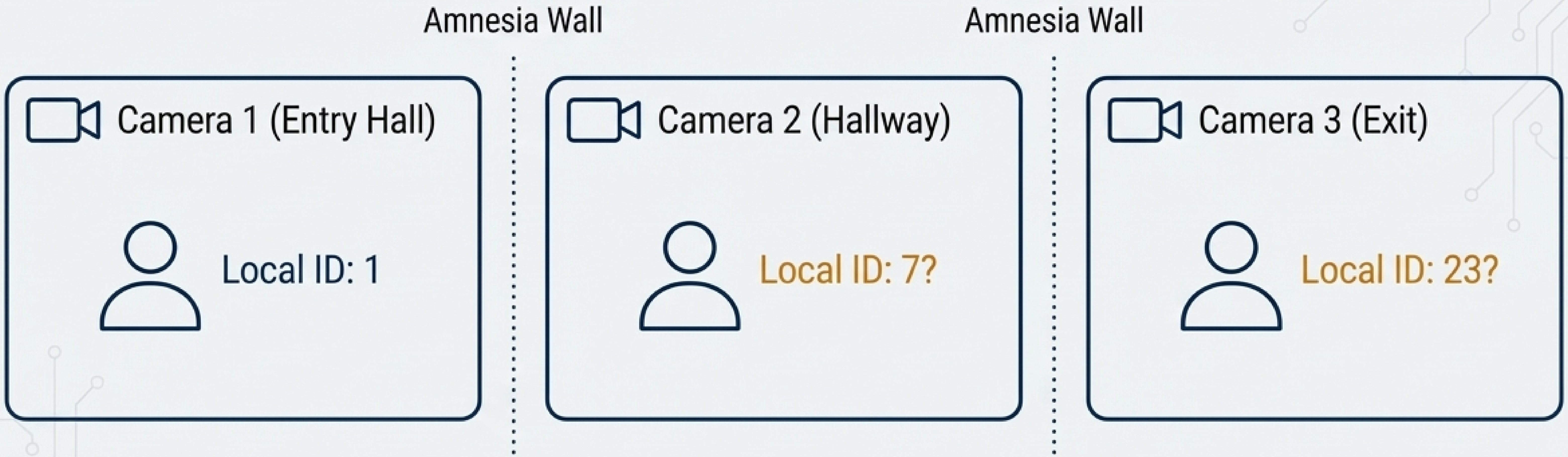
0.525	0.5
0.55	0.475

Combined Cost Matrix

Why Appearance is Crucial

- Handles long-term occlusions.
- Prevents ID switches during path crossings.
- Re-associates tracks after a detector fails for a few frames.

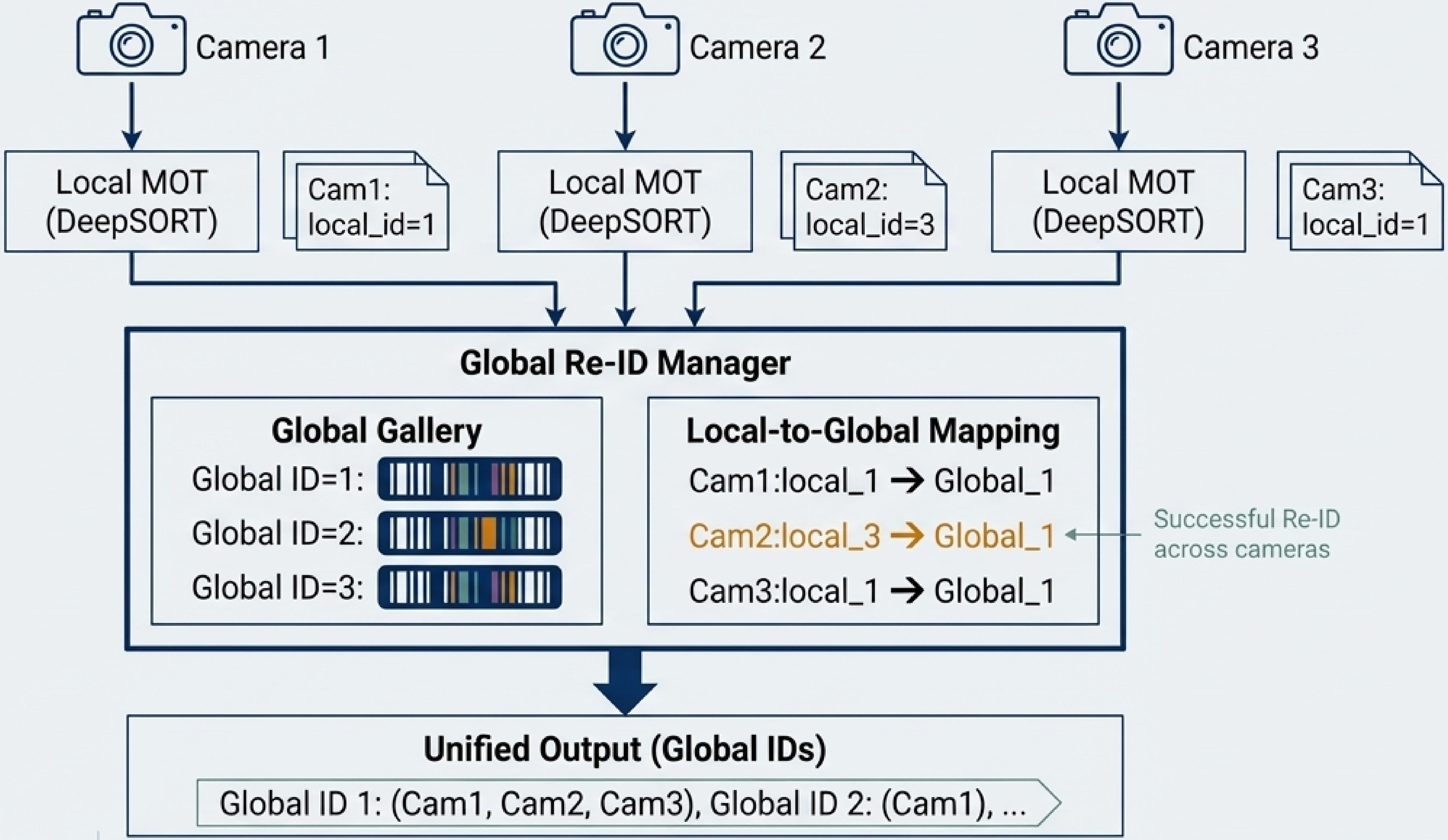
The Final Frontier: Scaling from One Camera to Many



The same person must be assigned the same **Global ID** across all camera views.

We need a central system that uses cross-camera Re-ID to manage a global gallery of identities.

The MCMOT System Architecture



How the Global Manager Unifies Identity

Scenario: A new track (local_id=5) appears in Camera 2. How is its global identity determined?

1. Extract

local_id=5 → color barcode

2. Compare

feat_new →

Global Gallery

Global ID=1
Global ID=2
Global ID=3

3. Calculate Distances

vs. Global ID	Cosine Distance
Global_1	0.18
Global_2	0.85
Global_3	0.72

4. Decide & Map

Since 0.18 < threshold (0.5), the system makes the match. The mapping **Cam2:local_5 → Global_1** is created.

5. Update

Global_1

$$feat_{global} = 0.9 \times feat_{global} + 0.1 \times feat_{new}$$

Exponential Moving Average for feature stability

The Developer's Toolkit: Models and Functions

Models Used

Person Detection: YOLOv4-tiny

Files: `yolov4-tiny.weights`, `.cfg`, `coco.names`

Person Re-Identification: OpenCV Zoo Model

File: `person_reid_youtu_2021nov.onnx`
Input: 128x256 crop
Output: 512-dim vector

Key OpenCV Functions

DNN

`cv2.dnn.readNet()`
`cv2.dnn.readNetFromONNX()`
`cv2.dnn.blobFromImage()`

Motion

`cv2.KalmanFilter()`
`kalman.predict()`
`kalman.correct()`

Association

`scipy.optimize.linear_sum_assignment() (from SciPy)

Math

`np.dot()`
`np.linalg.norm() (from NumPy)

Real-World Performance and Optimization

Detection Frequency: Detect every 3-5 frames and track in between to balance accuracy and speed.

Batch Processing: Run Re-ID feature extraction on batches of person crops for GPU efficiency.

GPU Acceleration: Explicitly use `net.setPreferableBackend(cv2.dnn.DNN_BACKEND_CUDA)`.

Feature Caching: Store and average the last N features for each track to create a more stable appearance signature.

Early Rejection: If the IoU between a predicted track and a detection is very high (>0.95), skip the computationally expensive Re-ID check.

Foundational Research and Further Reading

OpenCV

arXiv

Core Concepts

- SORT Paper: `arxiv.org/abs/1602.00763`
- DeepSORT Paper: `arxiv.org/abs/1703.07402`

Implementation & Models

- OpenCV DNN Module Documentation
- OpenCV Zoo - Person Re-ID Model

Tutorials

- OpenCV MOT Blog Post