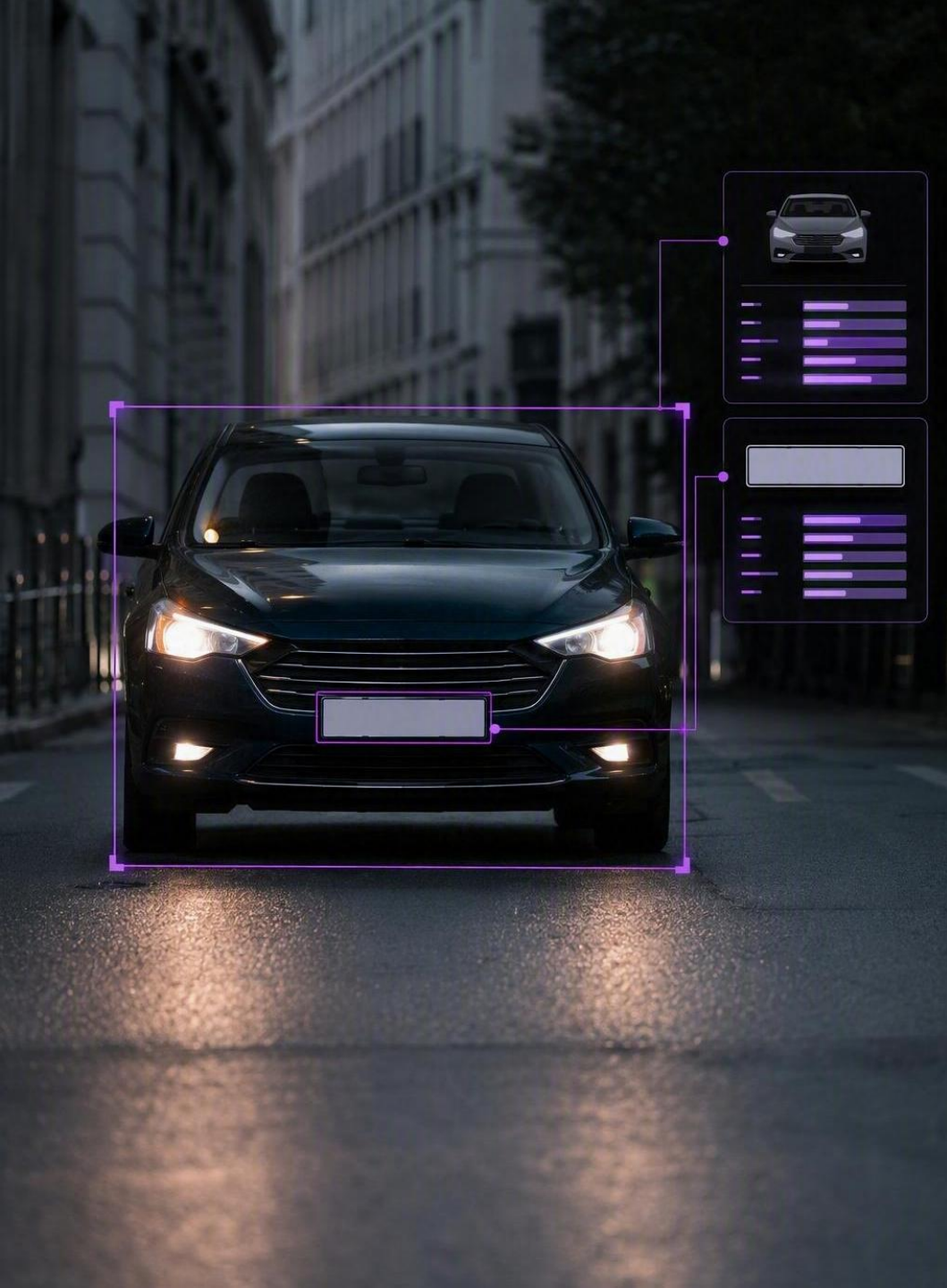




UNIFYING GSTANALYTICS METADATA

ACHIEVING CROSS-
VENDOR
INTEROPERABILITY IN
VIDEO ANALYTICS

PEP COTS, MILESTONE SYSTEMS

OSKAR FIEDOT, INTEL

DANIEL MORIN, COLLABORA

AGENDA

01 A unified convention for detections, classifications, keypoints and segmentation – Daniel Morin, Collabora

02 DL Streamer moves from custom metadata to the GstAnalytics convention, Oskar Fiedot, Intel

03 Use case: License Plate Recognition; challenges and lessons learned, Pep Cots Sastre, Milestone Systems

01 | THE CONVENTION

relation-meta, and the convention that makes it a cross-vendor language for common video analytics

Daniel Morin, Collabora

From Metadata to a Common Language For Analytics

- GstAnalyticsRelationMeta, introduced in GStreamer 1.24 (2023)
- Goal: let independently-built analytics elements compose, without knowing each other's types
- Flexible by design: But adopters interpreted the relation graph differently



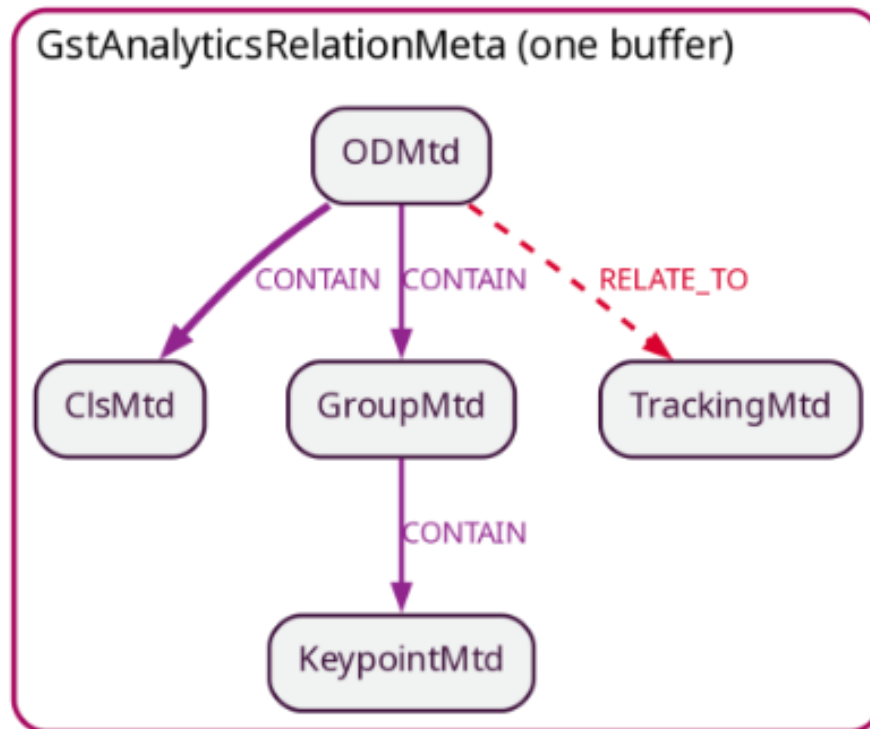
TrackingMtd → **CONTAIN** → ODMtd

?

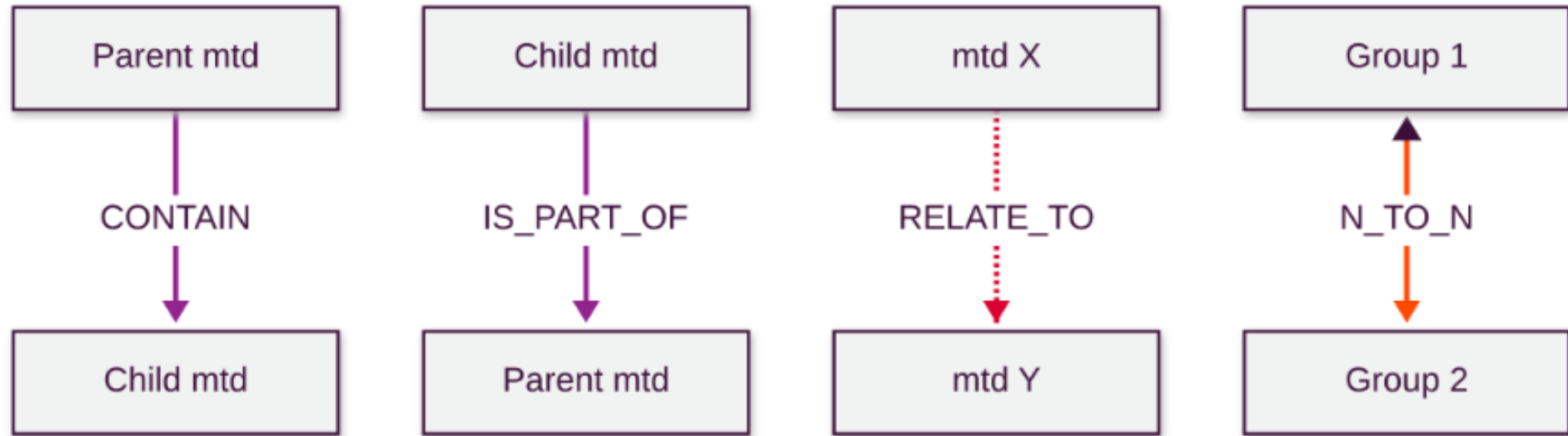
ODMtd → **RELATE_TO** → TrackingMtd

A Graph, Not a Schema

- GstAnalyticsRelationMeta on a buffer = container of results (Mtd) + directed relations between them
- Detections, classifications, keypoints, segmentation, tensors, tracking, all Mtds
- The convention itself targets known use-cases: object detection, segmentation, keypoints
- Not machine-learning specific, also holds CV
- Not even video specific, attaches to any buffer (audio, text, or no media at all)
- The convention isn't a technical limit on relation-meta, and it wouldn't justify one

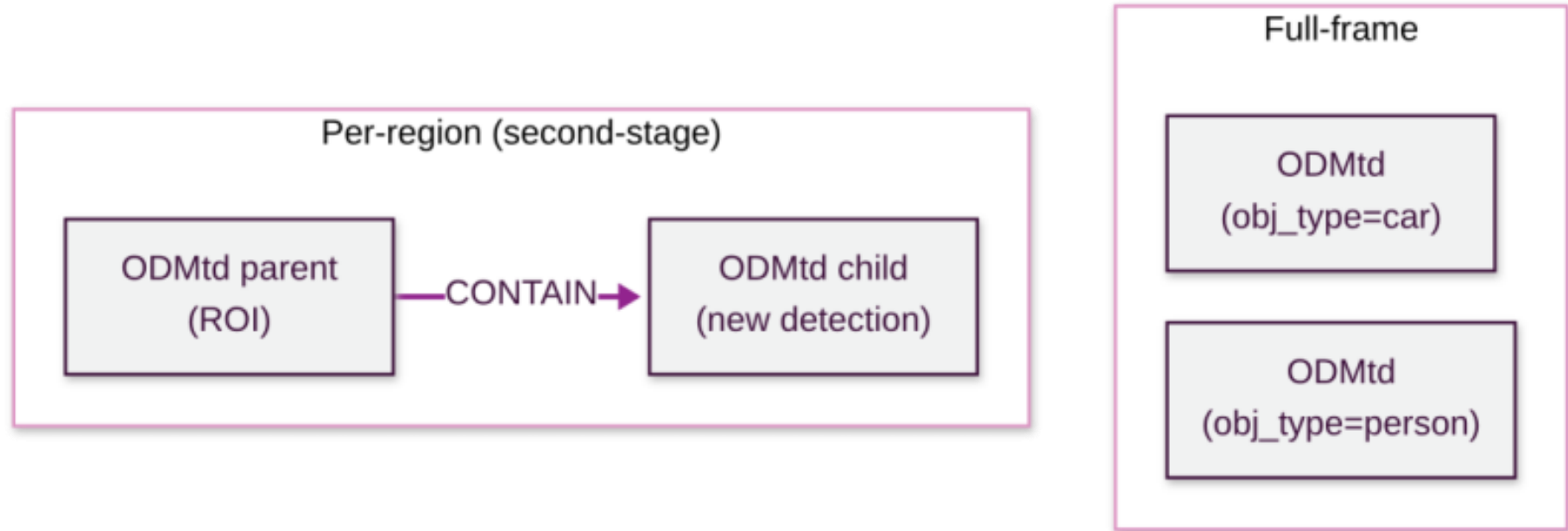


Relations type



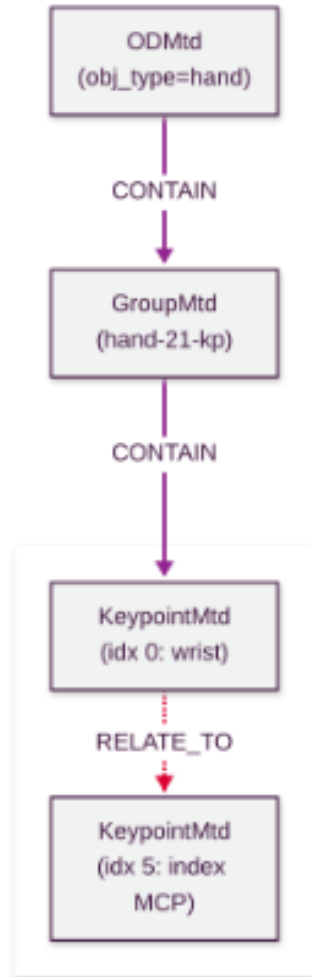
- CONTAIN / IS_PART_OF — ownership, always set as a pair
- RELATE_TO — association without ownership (tracking, skeleton edges)
- N_TO_N — component i of one group corresponds to component i of another

Building the Graph — Detection

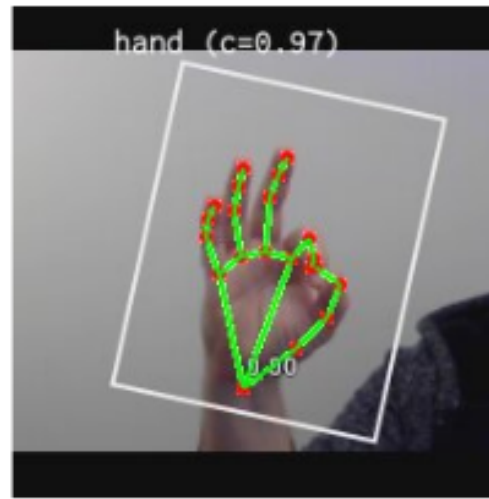


- One ODMtd per detected object, full-frame detections are top-level
- Second-stage detections nest under the parent ROI via CONTAIN
- This is the anchor node most other results attach to

Building the Graph — Keypoints



- Keypoints are an ordered group: GroupMtd of KeypointMtds
- Group attaches to the detection with CONTAIN
- The skeleton (which point connects to which) is RELATE_TO between keypoints
- Same four relation types, just composed differently



The Convention

Data type	Relation	To	Rule
Detection (ODMtd)	top-level, or CONTAIN	frame, or parent ROI	Nested only when detected inside an existing region
Classification (ClsMtd)	CONTAIN	owning ODMtd, or frame-level (no parent)	Additional label beyond the detection's own object type
Keypoints (GroupMtd + KeypointMtd)	CONTAIN (group → points), RELATE_TO (skeleton)	owning ODMtd, or frame-level	Ordered group where index position is meaningful
Segmentation (SegmentationMtd)	N_TO_N	ClsMtd	Region index $i \leftrightarrow$ class index i
Tracking (TrackingMtd)	RELATE_TO	ODMtd	Always relates to a detection, regardless of how it was produced
Raw tensor (TensorMtd)	CONTAIN	owning ODMtd, or frame-level	Opaque payload with a semantic_tag, not interpreted by the convention

Specific Enough to Interoperate, General Enough to Last

- The convention standardizes interpretation of known patterns
- It does not limit what relation-meta can express
- Mtd can easily be extended and placed in a relation graph like upstream Mtd's
- Two major adopters:
 - Intel DL Streamer, migrating a full framework onto this convention
 - Milestone Systems, enabling gstreamer for Open Platform consumers

02 | TRANSITION TO GSTANALYTICS

Intel is using GstAnalytics for metadata

Oskar Fiedot, Intel

Gstreamer AI Frameworks

INTEL

DL Streamer

Adopts the upstream GStreamer standard

GstAnalytics

DLStreamer moving toward closer alignment with GStreamer

DeepStream

Keeps its own metadata structure

NvDsBatchMeta

NNStreamer

Treats tensors as the stream itself

other/tensors

GstAnalytics is the metadata standard inside GStreamer itself. DL Streamer builds on it.
No proprietary metadata anymore in DLStreamer

DL Streamer alignment with GStreamer

We want to align as many DL Streamer elements as possible with GStreamer.

gvadetect	gvaanalytics	gvametaconvert	g3dradarprocess
gvaclassify	gvaattachroi	gvametapublish	g3dlidarsrc
gvainference	gvafpscounter	gvapython	g3dlidarparse
gvamono3d	gvafpsthrottle	gvarealsense	g3dinference
gvatrack	gvastreammux	gvawatermark	g3dobjectfuser
gvaaudiotranscribe	gvastreamdemux	gvawatermark3d	g3drender
gvaaudiotranscribe	gvametaaggregate	gvamotiondetect	

We started with GstAnalytics and want to go further.

Many DL Streamer elements already speak the GstAnalytics language.

Why did we switch to GstAnalytics?

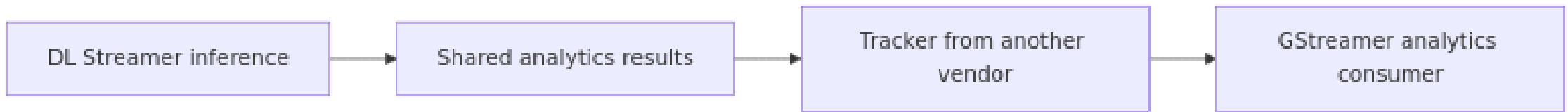
Interoperability instead of a closed ecosystem

Before:

- DL Streamer elements understood only DL Streamer metadata
- Every consumer (overlay, converter, publisher) had to be ours
- Integrators had to write their own parser for our GstStructure fields

Goal:

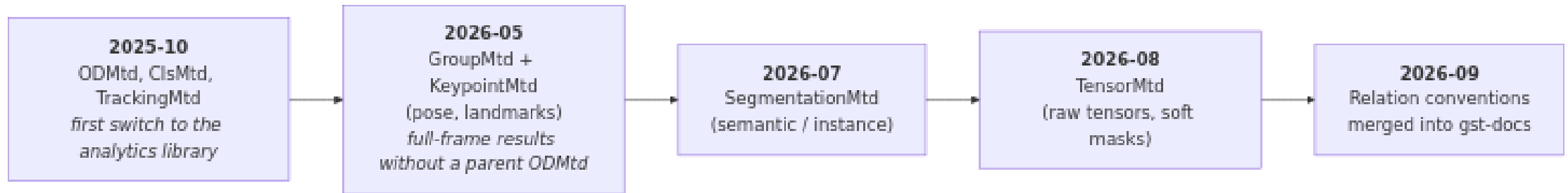
- Any producer -> any consumer, as long as both speak GstAnalytics
- Mix elements from Intel, upstream GStreamer and other vendors in one pipeline
- Use upstream tooling (overlays, converters, Python bindings) without adapters
- Stop maintaining a private metadata format and its documentation



From custom metadata to GstAnalytics

From DL Streamer-specific result conventions to a shared GStreamer analytics language.

- Compose analytics elements across vendors.
- Reuse results without knowing the producer's private schema.
- Elements need to interpret metadata relations the same way.



From custom metadata to GstAnalytics

What we had (deprecated, soon to be removed):

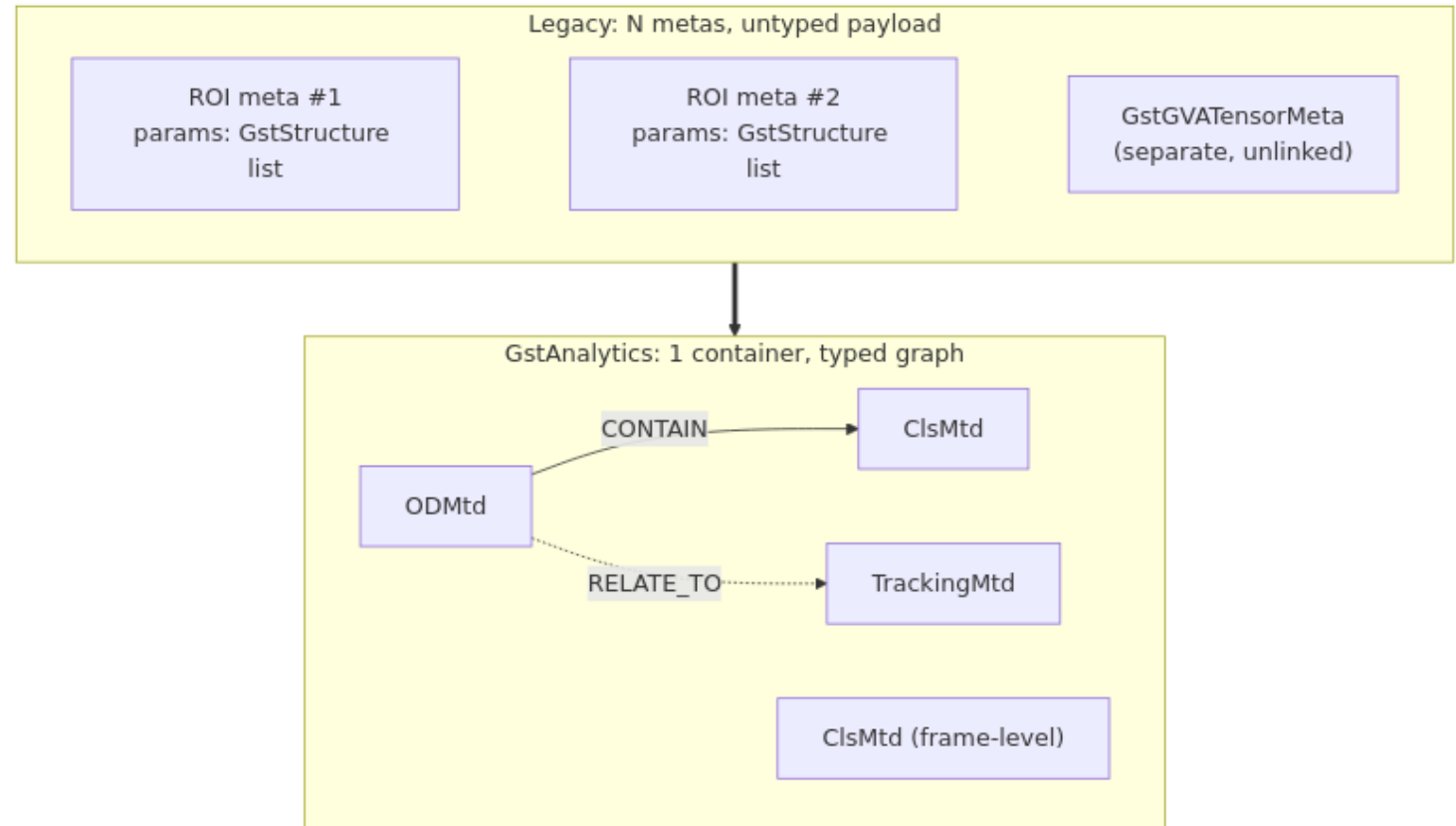
- GstVideoRegionOfInterestMeta per object: x, y, w, h, roi_type
- Everything else as a GList *params of GstStructure (confidence, model name, labels, keypoints, tensors...)
- Two custom metas: GstGVATensorMeta, GstGVAJSONMeta

What we have now:

- A single GstAnalyticsRelationMeta per buffer
- Typed entries: ODMtd, ClsMtd, TrackingMtd, GroupMtd, KeypointMtd, SegmentationMtd, TensorMtd
- Linked by relations: CONTAIN / IS_PART_OF / RELATE_TO

From private structures to a typed result graph

- Before: standard ROI metadata with DL Streamer-specific parameters; custom tensor metadata.
- Now: typed results and explicit relations in GstAnalyticsRelationMeta.
- Legacy metadata is being retired; remaining code is planned for removal soon.



Our own Mtd types in the GstAnalytics style

GstAnalytics lets a plugin register its own **GstAnalyticsMtd** type and store it in the same **GstAnalyticsRelationMeta**, with the same relation API. DL Streamer uses this for results that have no upstream type yet:

Mtd	Produced by	Payload
GstAnalytics3DODMtd	g3dinference (LiDAR/radar)	3D oriented box, class, confidence, sensor modality
GstAnalyticsZoneMtd	gvaanalytics	zone id
GstAnalyticsTripwireMtd	gvaanalytics	tripwire id, crossing direction
GstAnalyticsDwellTimeMtd	gvaanalytics	zone id, dwell time, first-seen timestamp

Same library is not the same language

The problem:

- `GstAnalyticsRelationMeta` is only storage plus an adjacency matrix
- It does **not** say how to establish relationships between metadata
- Two vendors can both use **GstAnalytics** and still not understand each other

What we did:

- Together with Collabora (GStreamer maintainers) and Milestone Systems we wrote a set of recommended conventions
- Merged into GStreamer docs: [Machine Learning Based Analytics](#) 
- Covers object detection, classification, keypoints, segmentation, raw tensors, tracking, and full-frame vs per-region results

Tutorials Language

Analytics Metadata Relation Conventions

`GstAnalyticsRelationMeta` only provides the storage and the adjacency matrix; it does not mandate how producers wire results together. This section describes the conventions a producer is expected to follow so that a consumer can navigate the graph without knowing which element produced which result. It covers object detection, classification, keypoints, segmentation, raw tensors and tracking, and the difference between **full-frame** and **per-region (ROI)** results.

Relation Types

Parent mtd	--CONTAIN-->	Child mtd
child mtd	--IS_PART_OF-->	Parent mtd
mtd X	--RELATE_TO-->	mtd Y
Group mtd 1	--N_TO_N-->	Group mtd 2

- **CONTAIN** (`GST_ANALYTICS_REL_TYPE_CONTAIN`): parent owns child. Used for `ODMtd -> {ClsMtd, GroupMtd, TensorMtd}` and `parent -> child ODMtd`.
- **IS_PART_OF** (`GST_ANALYTICS_REL_TYPE_IS_PART_OF`): the inverse edge, set together with every **CONTAIN** so the graph is navigable both ways.
- **RELATE_TO** (`GST_ANALYTICS_REL_TYPE_RELATE_TO`): non-ownership association, used for tracking and for keypoint skeleton edges.
- **N_TO_N** (`GST_ANALYTICS_REL_TYPE_N_TO_N`): a component-wise relation between two groups, where each component of one group corresponds to the respective component of the other group. For example a `SegmentationMtd` (a group of region ids) linked to a `ClsMtd` (a group of labels) so that region *i* maps to label *i*.

Every relation is stored as a **directed edge**: `gst_analytics_relation_meta_set_relation (type, a, b)` records `a -> b` only. **CONTAIN** / **IS_PART_OF** are therefore set as a pair, while **RELATE_TO** and **N_TO_N** are written once, in the direction chosen by the producer.

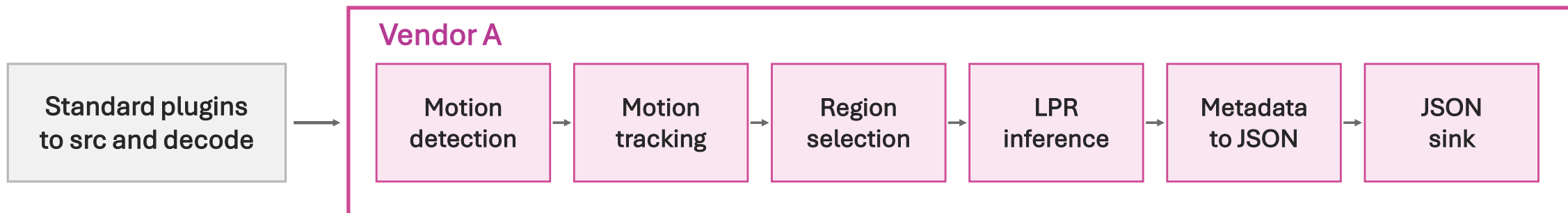
To keep the diagrams below readable, only the **CONTAIN** direction is drawn; the matching **IS_PART_OF** edge is always set as well.

03 | CROSS-VENDOR LPR

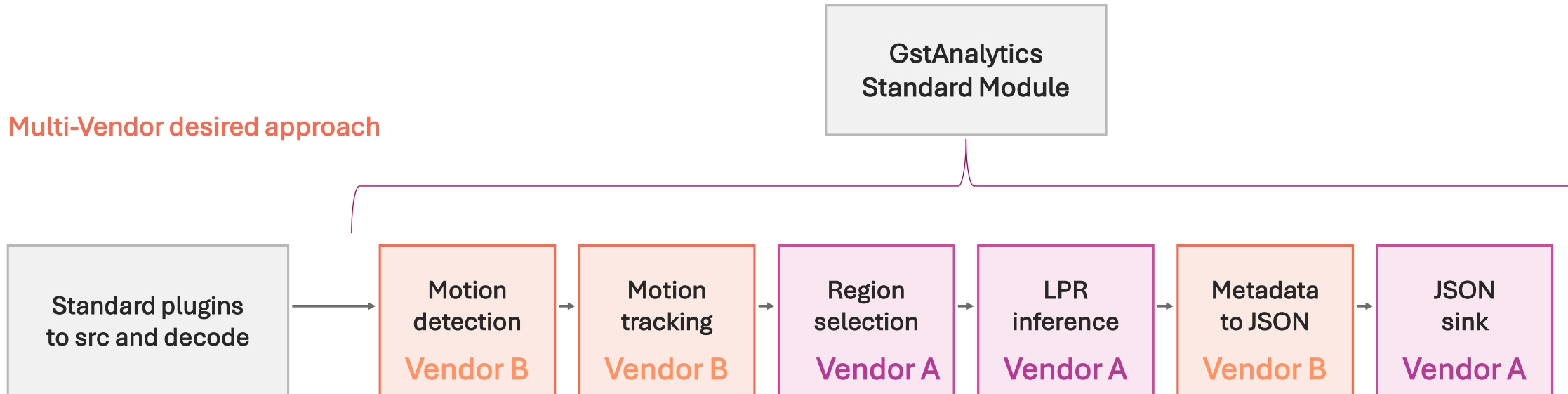
The use case that sparked alignment
Pep Cots, Milestone Systems

The LPR use case

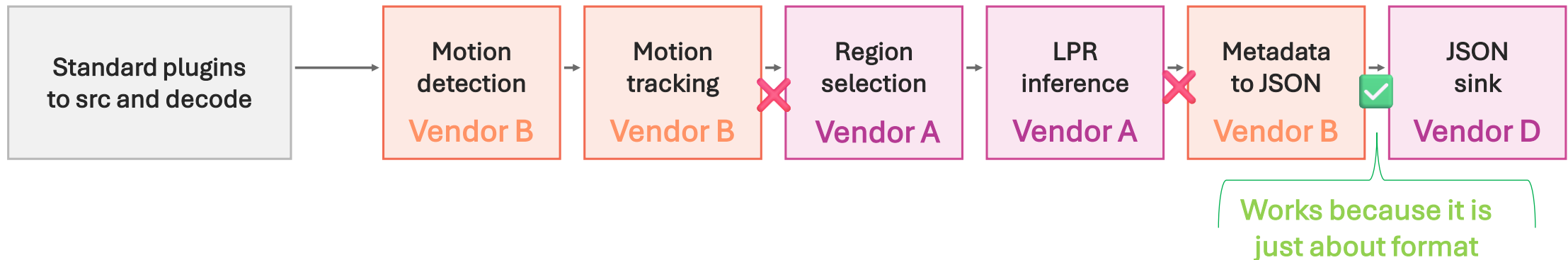
Single-Vendor analytics solution



Multi-Vendor desired approach

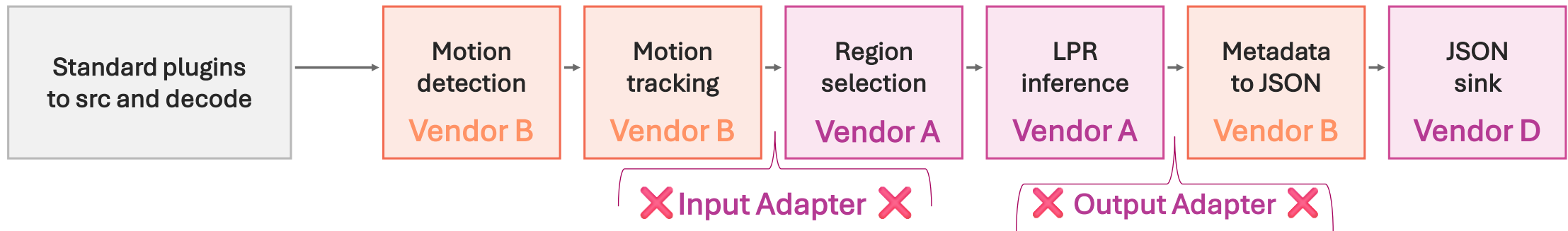


Were integration broke



- GstAnalytics was still at an early stage, with limited integration guidance
- Analytics are Story-Tellers: semantics are as important as format
- Producers and consumers made different assumptions about metadata relations.
- Format-only elements worked because they did not interpret the metadata.
- Semantic interoperability **required a documented producer-consumer contract.**

Why an adapter was NOT enough



- Adapters do not scale
- Adds (even more) ecosystem fragmentation
- Reduces reusability, maintenance cost
- Not enough for indirect consumers
- Standard semantics provide long-term interoperability

How the use case sparked alignment

- **Observe:** One vendor followed the other's implementation on GitHub.
- **Connect:** They contacted the team to learn from a real metadata consumer.
- **Discover:** Two elements from different vendors lacked a shared understanding of GstAnalytics semantics.
- **Escalate (constructively):** The teams brought the issue to the GstAnalytics maintainers.
- **Align:** The discussion established that interoperability requires conventions, not pairwise adapters.
- **Improve:** The design document would be expanded with guidance on how metadata producers and consumers should align.



Real use cases drive better standards.

Community feedback turned a practical integration challenge into a clear guide for the entire ecosystem.

The working cross-vendor pipeline



Lessons for future integrations

Community matters,

but value comes from actively engaging with the developers doing the work.

- Reach the people building what you depend on
- Share feedback from real use cases.
- Engage early
- Participate, do not just consume
- Treat feedback as a contribution

Takeaways and questions

- Industry leaders already use GstAnalytics. Intel DL Streamer has already adopted it.
- GStreamer publishes rules how to use gstanalytics
- Standards need to be validated by real-world use cases
- Thank you – questions?