DEEPSTREAM SDK 2.0 WEBINAR

James Jeun

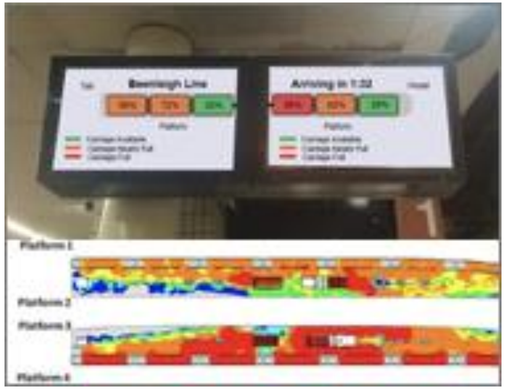
July 2018

AGENDA

- Introduction to Intelligent Video Analytics
- What is DeepStream?
- DeepStream Basic Building Blocks
- Metadata Handling
- DeepStream Pipeline Architecture
- DeepStream Plugins
- Memory Management
- DeepStream Reference Application
- Performance
- Custom Plugins
- Translating Use Case to DeepStream Architecture
- Q & A

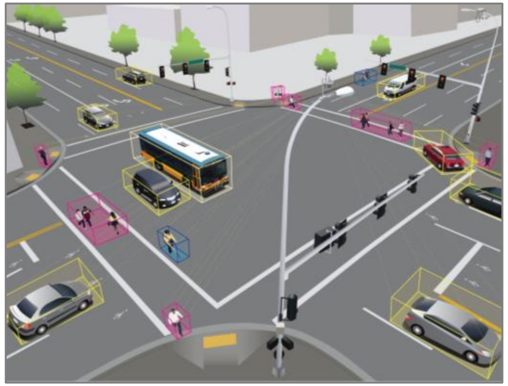
IVA IN SMART CITIES

Access Control

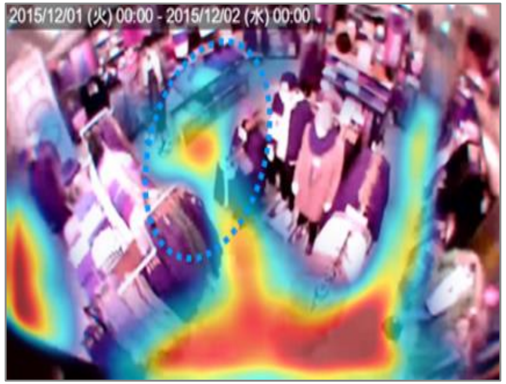


Public Transit

Parking Management



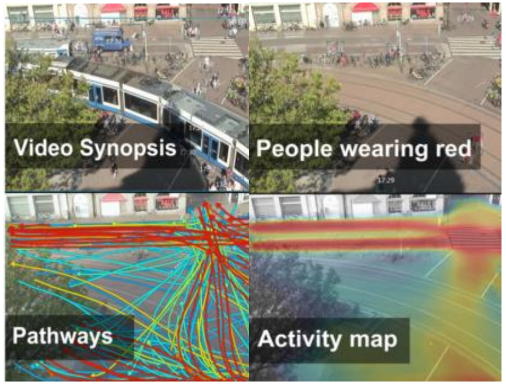
Traffic Engineering



Retail Analytics

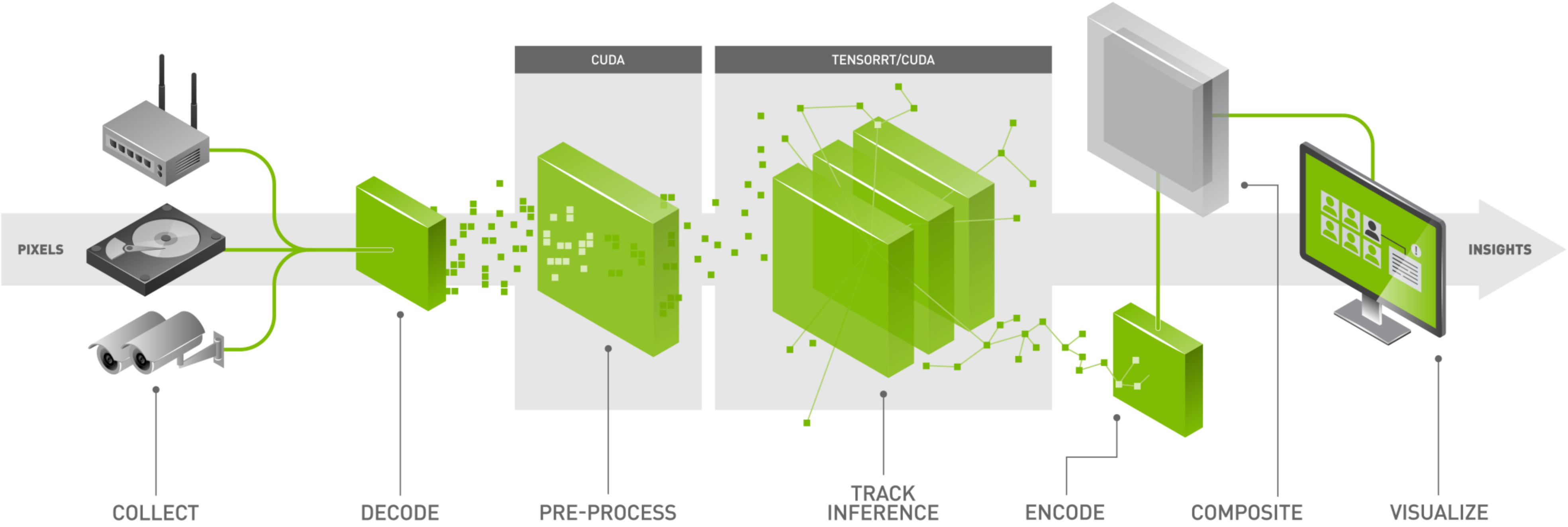
Securing Critical Infrastructure

Managing Logistics

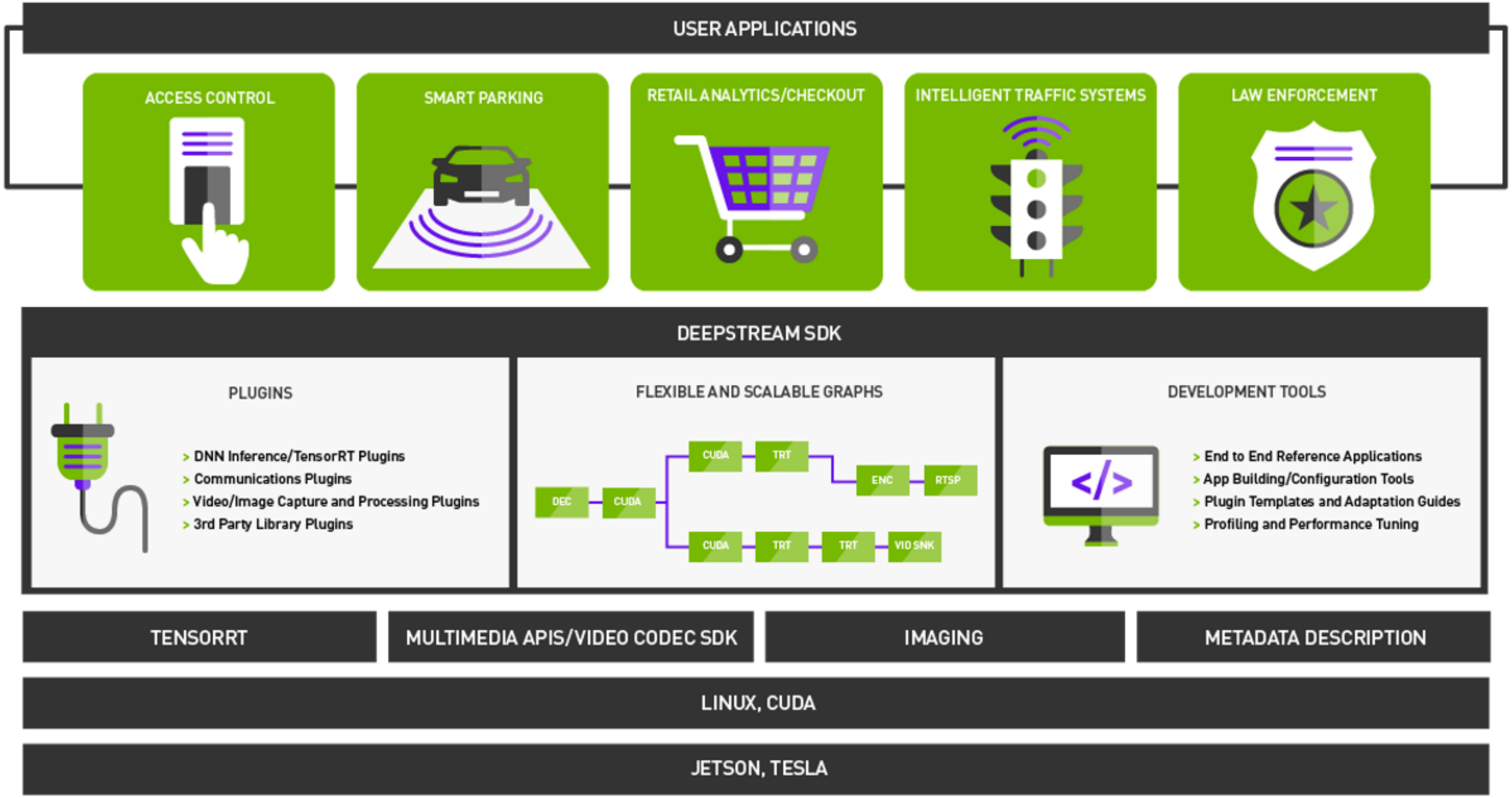


Forensic Analysis

INTELLIGENT VIDEO ANALYTICS

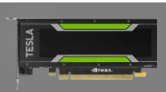


DEEPSTREAM SOFTWARE STACK



DEEPSTREAM 2.0

DeepStream on Tesla v1.5



Tesla P40, P4

API	C++
Streams	Multi
Graph	Fixed function
DNNs	Single
Examples	DNNs

DeepStream on Jetson v1.5

Jetson TX1, TX2

API	Modular plugins. Gstreamer based
Streams	Single, Multi thru multi-app
Graph	Custom
DNNs	Multi
Examples	Full app, multi-DNNs, tracking

DeepStream v2.0

Tesla

Jetson *

API	Unified edge-cloud
Streams	Multi
Graph	Custom
DNNs	Multi
Examples	Multiple apps, more plugins, multi-DNNs

New modular framework and APIs →

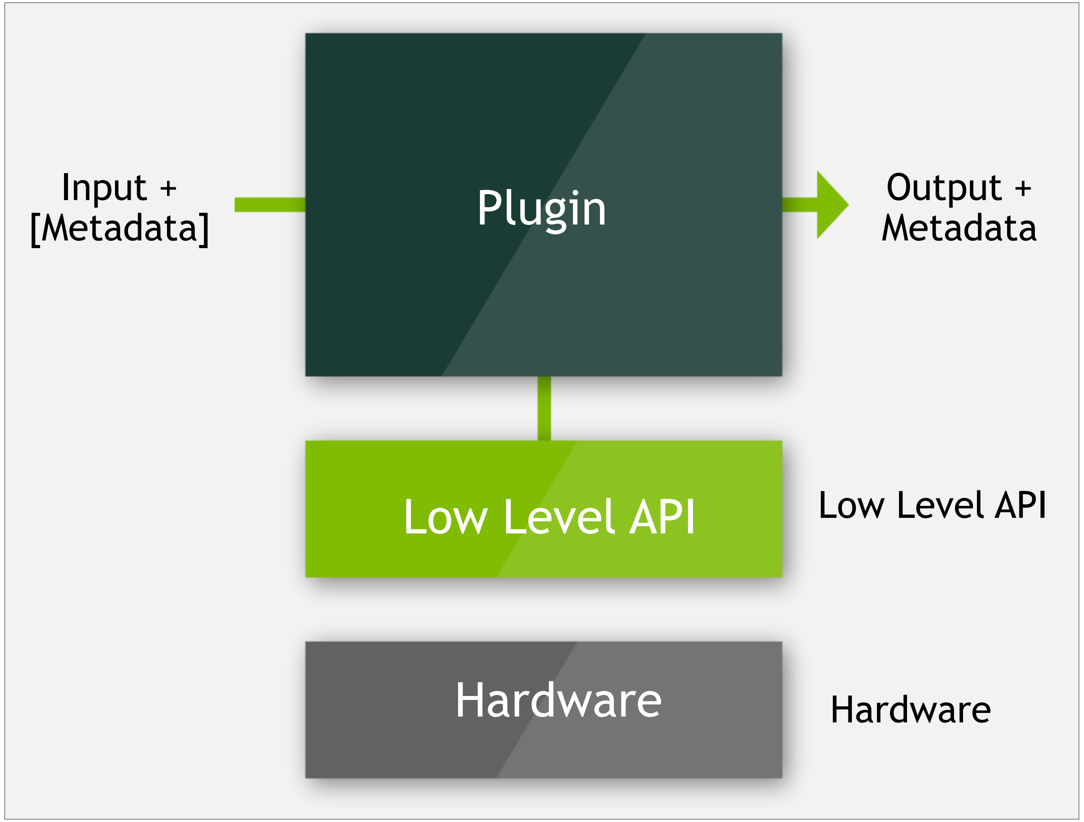
Previous

Tesla - Now
Jetson - 2H’18

DEEPSTREAM BUILDING BLOCK

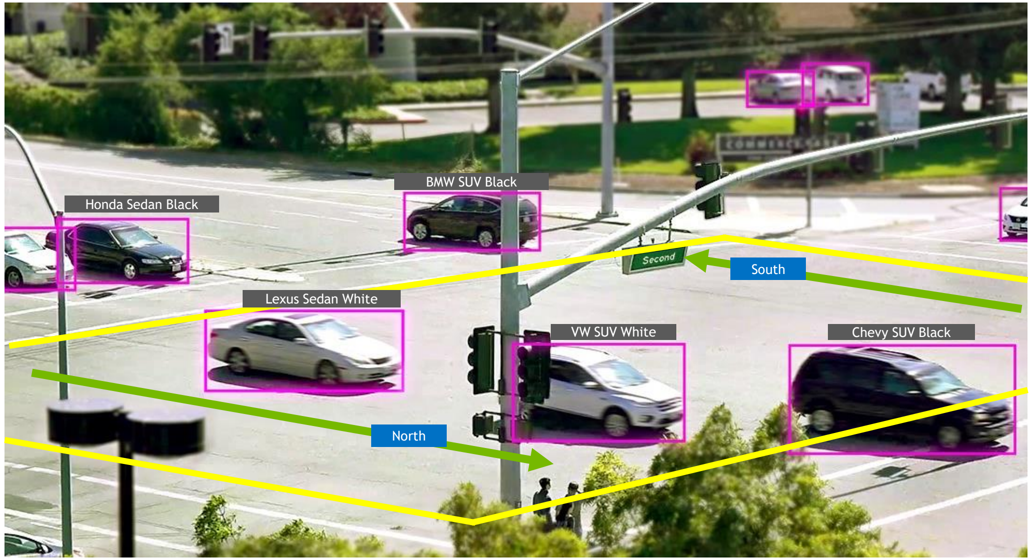
Gstreamer Plugin

- Based on Open Source GStreamer Framework
- A plugin model based pipeline architecture
- Graph based pipeline interface to allow high level component interconnect
- Enables heterogenous parallel processing on GPU and CPU
- Hides parallelization and synchronization under the hood
- Inherently multi-threaded



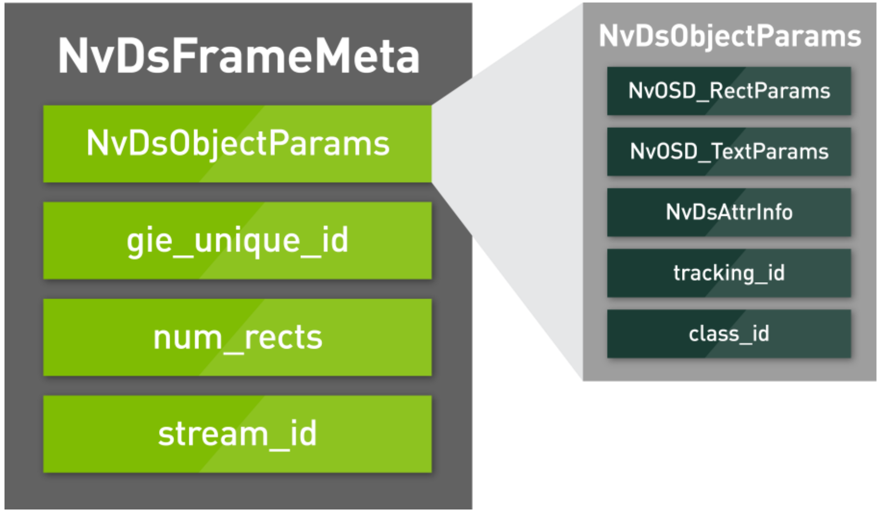
METADATA IN DEEPSTREAM

- Metadata is generated by plugins in the graph
- Plugins can progressively populate generated metadata
- Metadata generated at every stage of the graph can be used for further processing
- Metadata examples
    - Type of object detected
    - ROI coordinates
    - Object classification
    - Unique ID
    - Source and GPU ID
    - Rendering information and many more



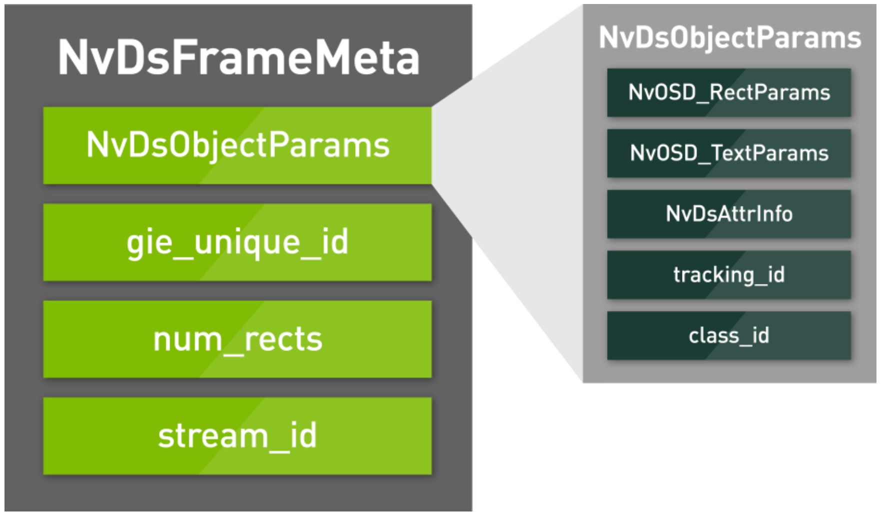
METADATA STRUCTURE (1/2)

- **NvDSObjectParams** – Contains a subset of metadata information for an object detected in the frame.
- **GIE_Unique_ID -** Multiple neural networks get assigned a unique ID
- **Num_rects** – Number of objects detected in the frame
- **Stream_Id** – In case of multi-stream to identify we need stream id to associate to which stream the data belongs to.



METADATA STRUCTURE (2/2)

- **NvOSD_RectParams** – Bounding box co-ordinates
- **NvOSD_TextParams** – Label information required for display (white car, Mercedes, sedan)
- **NvDSAttribinfo** – Attributes of objects (type, color, make)
- **Tracking_ID** – Unique ID of that object from tracker
- **Class_ID** – Type of object (Person, vehicle, two-wheeler, road sign)



DEEPSTREAM PIPELINE ARCHITECTURE

DEEPSTREAM	GST-uridecodebin	GST-NvStreamMux	GST-NvInfer	GST-NvTracker	GST-NvInfer	GST-nvvidconv	GST-NvOsd	GST-NvEglGlesSink
LOW LEVEL APIS	decode		libnvtensor-rt (TensorRT)	libnvtracker (visionworks)	libnvtensor-rt (TensorRT)	libnvvidconv	libnvosd	
JETSON	NVDEC	GPU DMA	GPU VIC	CPU VIC	GPU VIC	VIC	CPU VIC GPU	NVDC
TESLA	NVDEC	GPU DMA	GPU	CPU GPU	GPU	GPU	CPU GPU	NVDisplay

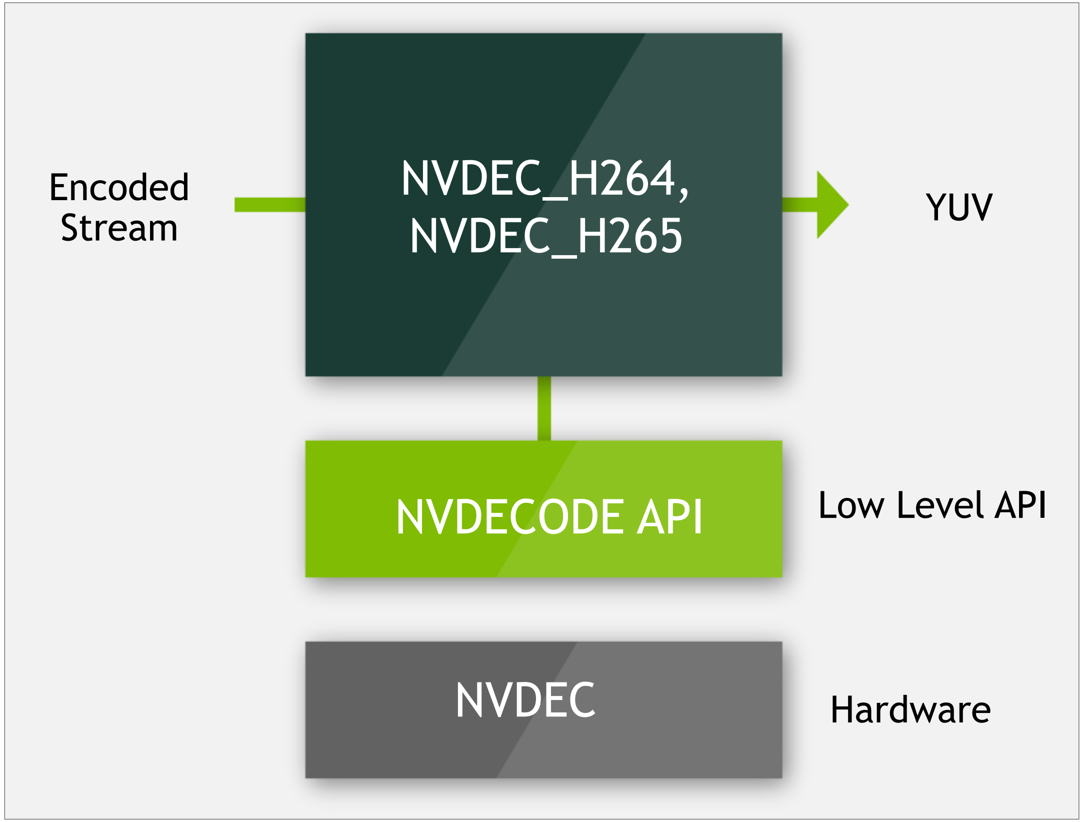
NVIDIA-ACCELERATED PLUGINS

Some of the most commonly used

Plugin Name	Functionality
gst-nvvideocodecs	Accelerated H.265 & H.264 video decoders
gst-nvstreammux	Stream aggregator - muxer and batching
gst-nvinfer	TensorRT based inference for detection & classification
gst-nvtracker	Reference KLT tracker implementation
gst-nvosd	On-Screen Display API to draw boxes and text overlay
gst-tiler	Renders frames from multi-source into 2D grid array
gst-eglglessink	Accelerated X11 / EGL based renderer plugin
gst-nvvidconv	Scaling, format conversion, rotation.

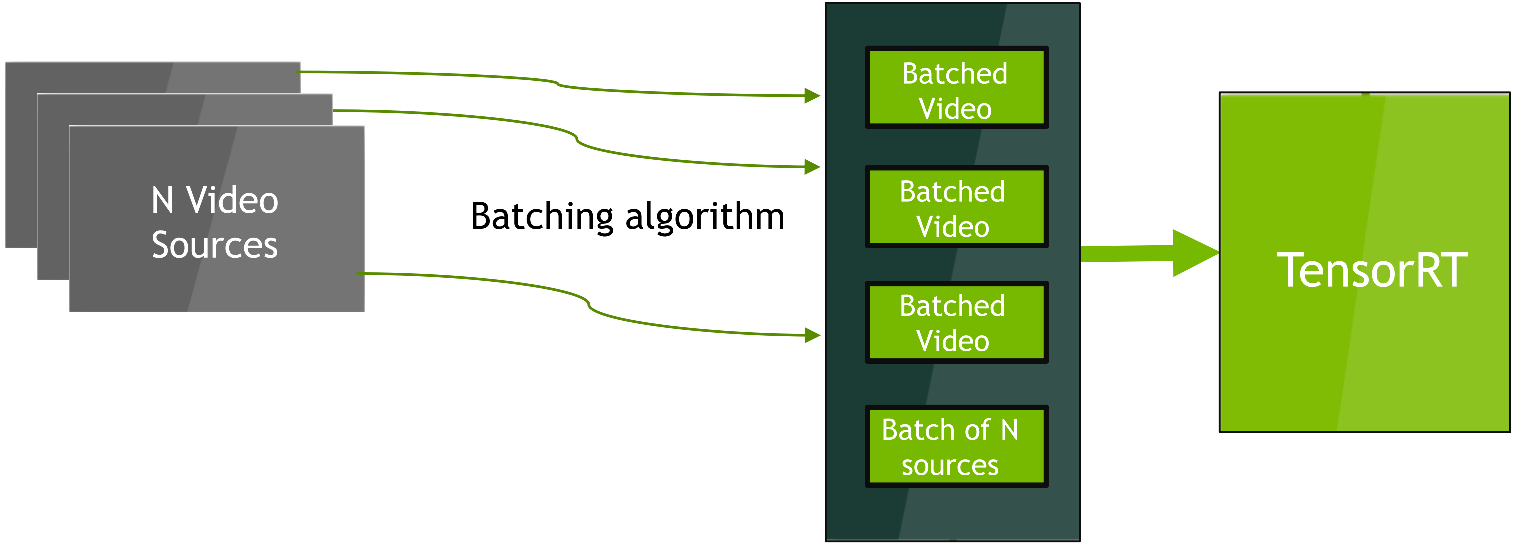
DECODER PLUGIN

Name - nvdec_h264, nvdec_H265



- Support multi-stream simultaneous decode
- Uses NVDECODE API (formerly NVCUVID API)
  - H.264
  - H.265
- Bit depth – decoder and platform limited
- Resolution – decoder and platform limited
- Compatible to plugins that accept YUV data
  - Nvinfer
  - Nvvidconv

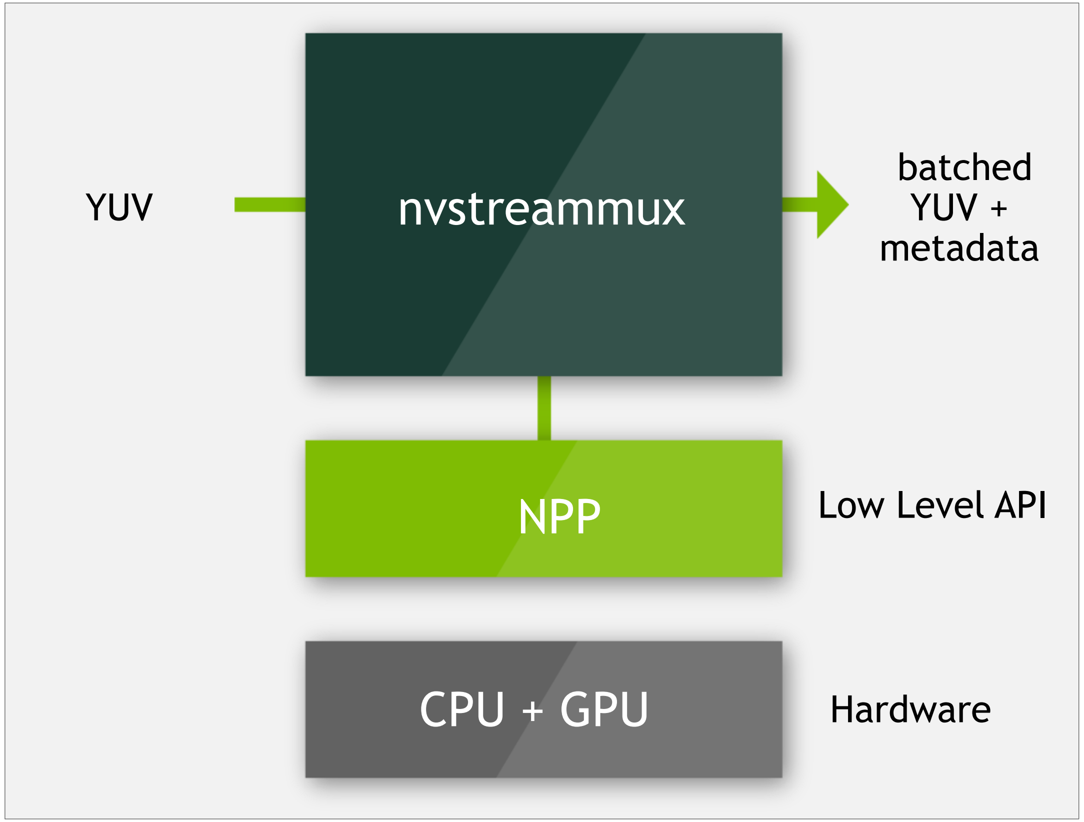
MULTI-STREAM BATCHING



- TensorRT optimized for batched input
- Aggregate multiple sources to create batches
- Attach metadata to differentiate between buffers of different sources

VIDEO AGGREGRATOR

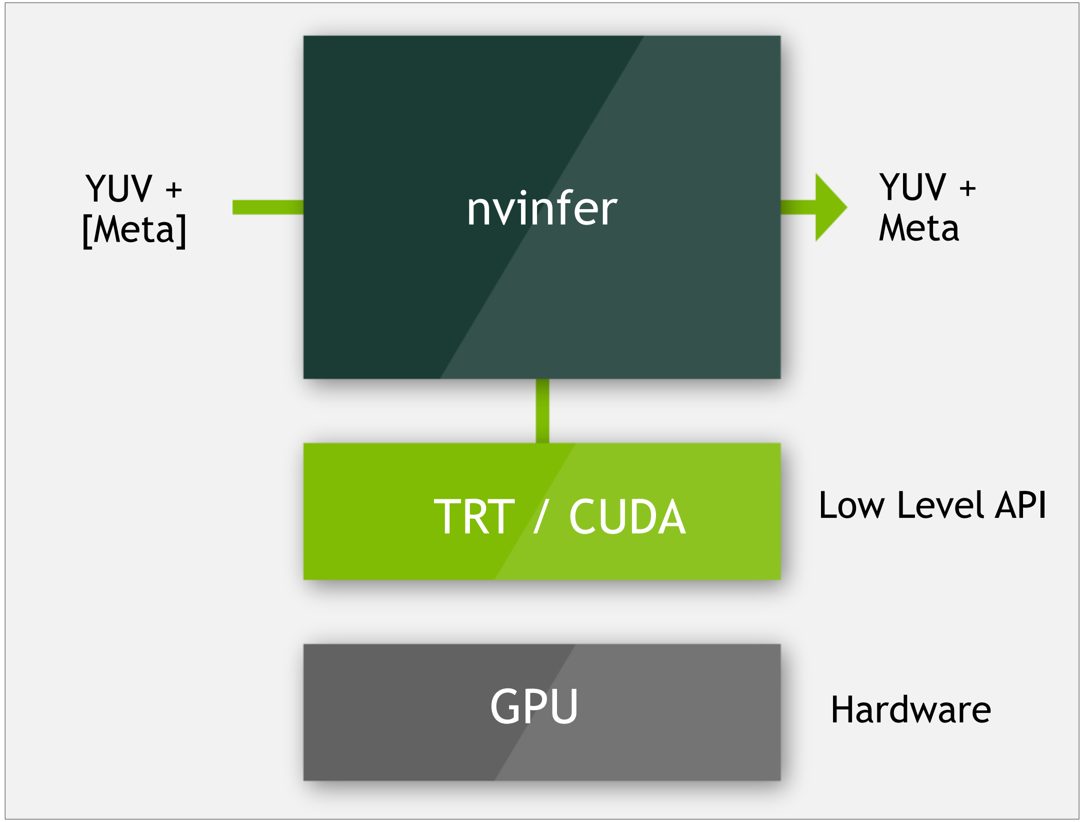
Name - nvstreammux



- Plugin that accepts "n" inputs streams and converts to sequential batch frames
- Scaling support – Incase video input resolution differs with the model resolution or vice-a-versa

INFERENCE

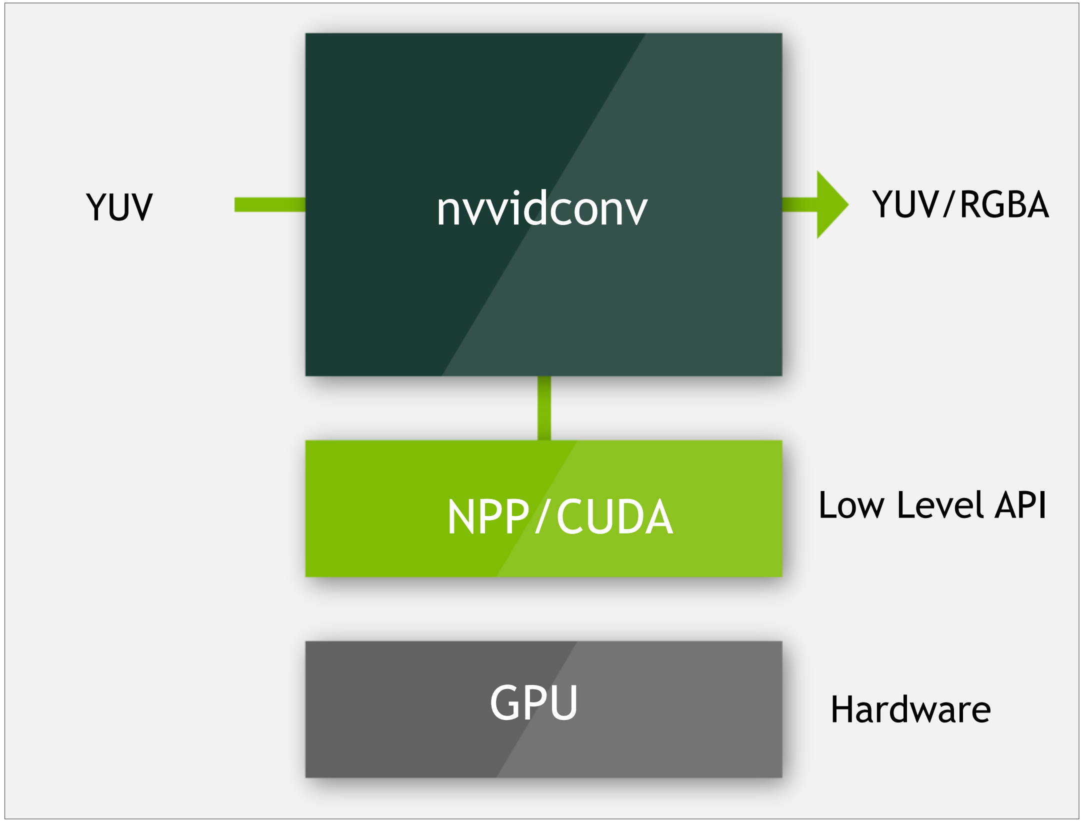
Name - nvinfer



- Detector and Classifier
- Primary & Secondary modes
- Caffe & UFF models supported
- Supports detector, classifier models that TensorRT supports
- Batched inferencing
- Provision for adding custom models
- Group rectangle algorithm for clustering

FORMAT CONVERSION & SCALING PLUGIN

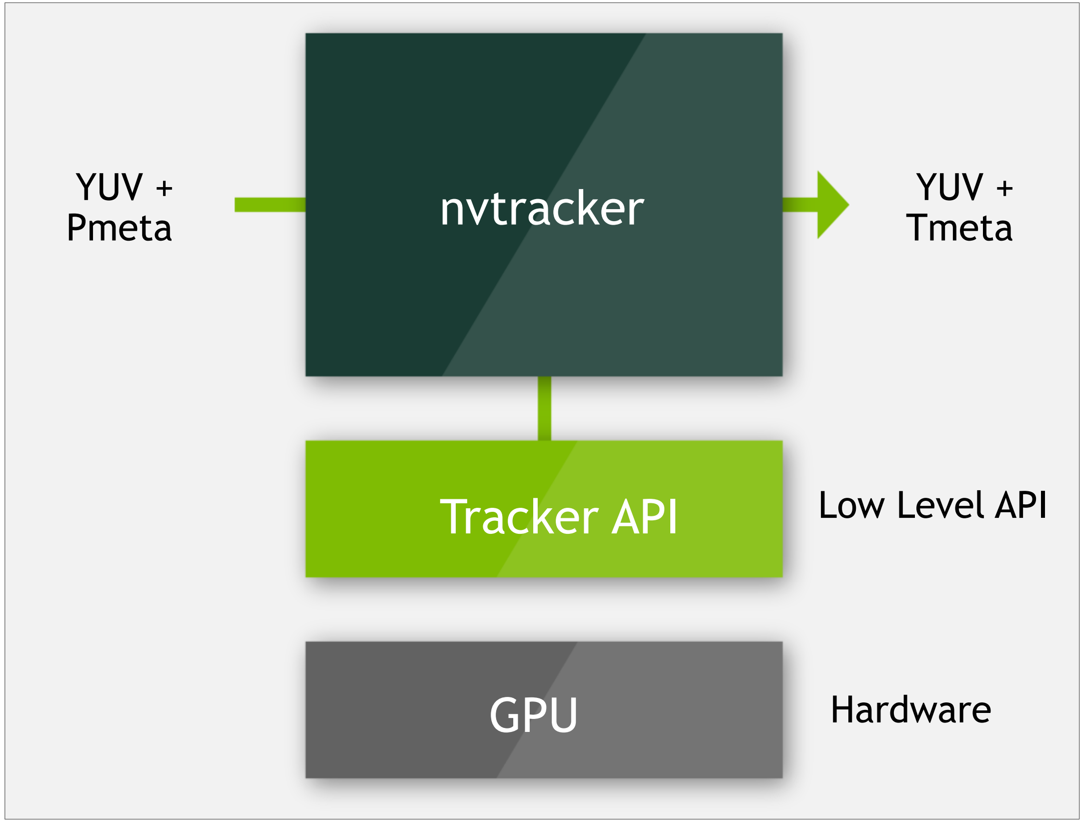
Name - nvvidconv



- Uses NPP (NV performance primitives)
- Format conversion
  - YUV>RGBA
  - YUV>BRGA
- Resolution Scaling
- Image Rotation

OBJECT TRACKER

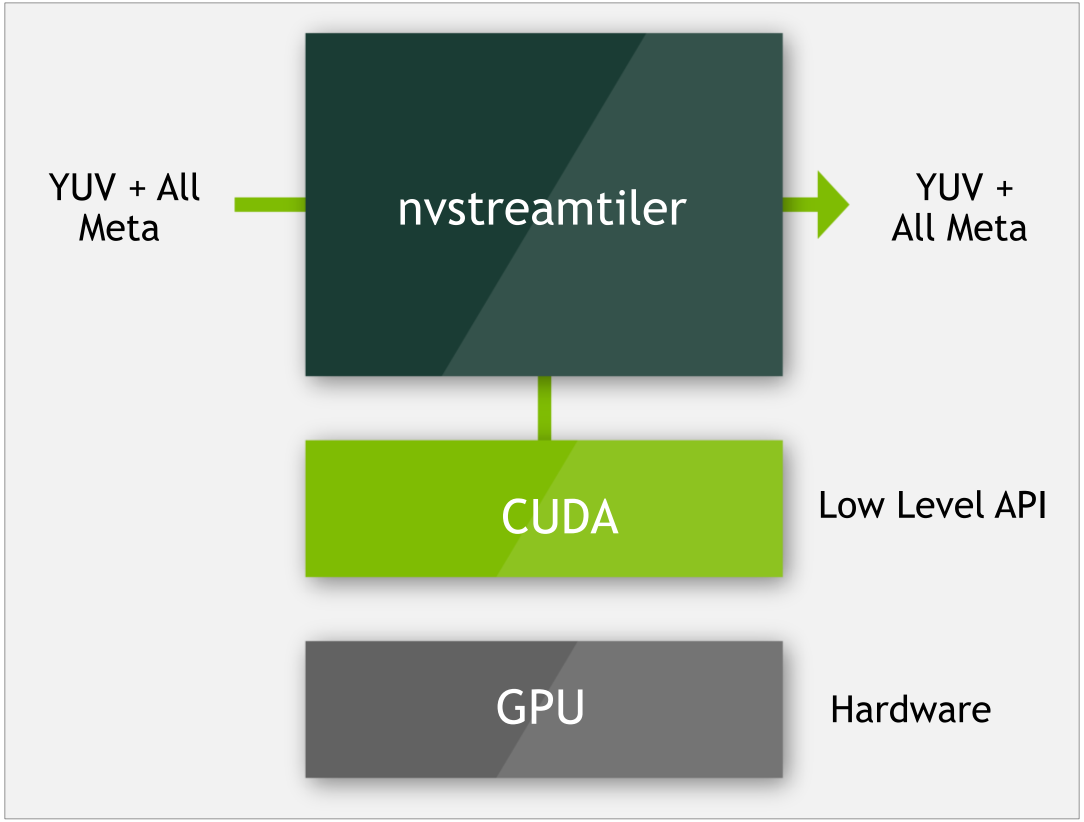
Name - nvtracker



- Based on KLT reference implementation
- Uses NPP / CUDA kernel internally for scaling and format conversion
- Can be upgraded to advance tracker

SCREEN TILER

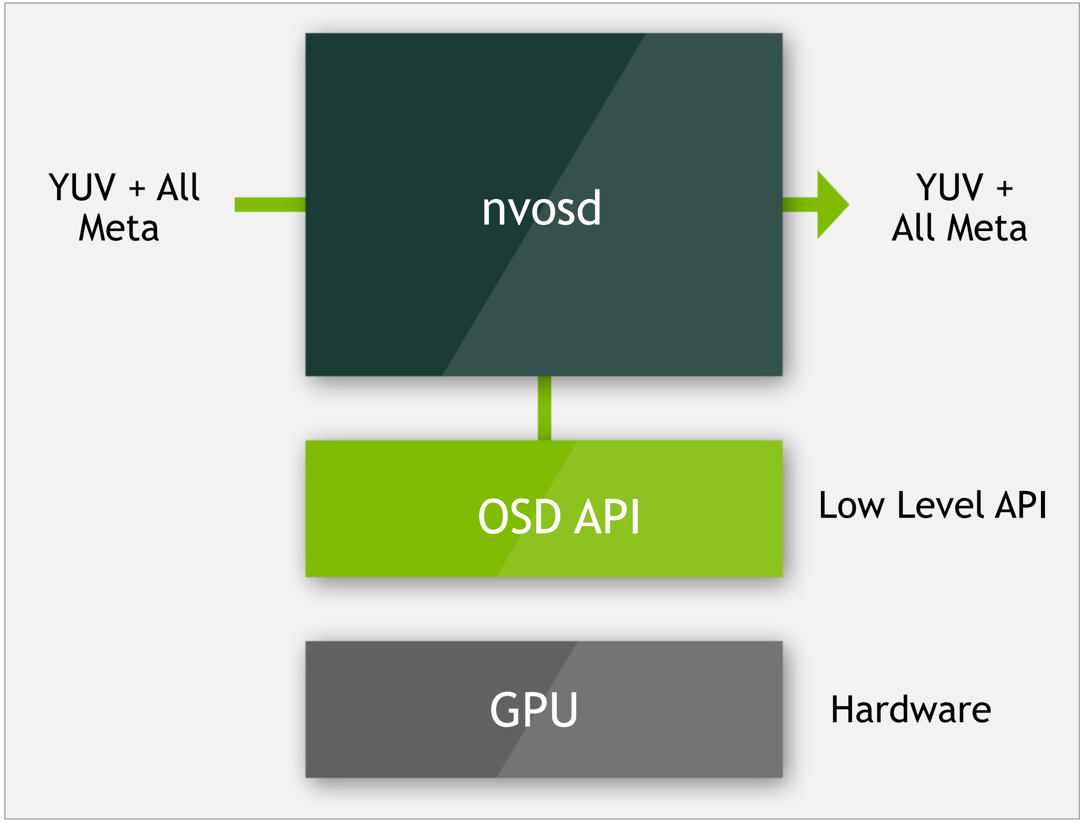
Name - nvstreamtiler



- Used for creating video wall effect
- Arranges multiple input sources into complete video tiled output
- Configurable window size

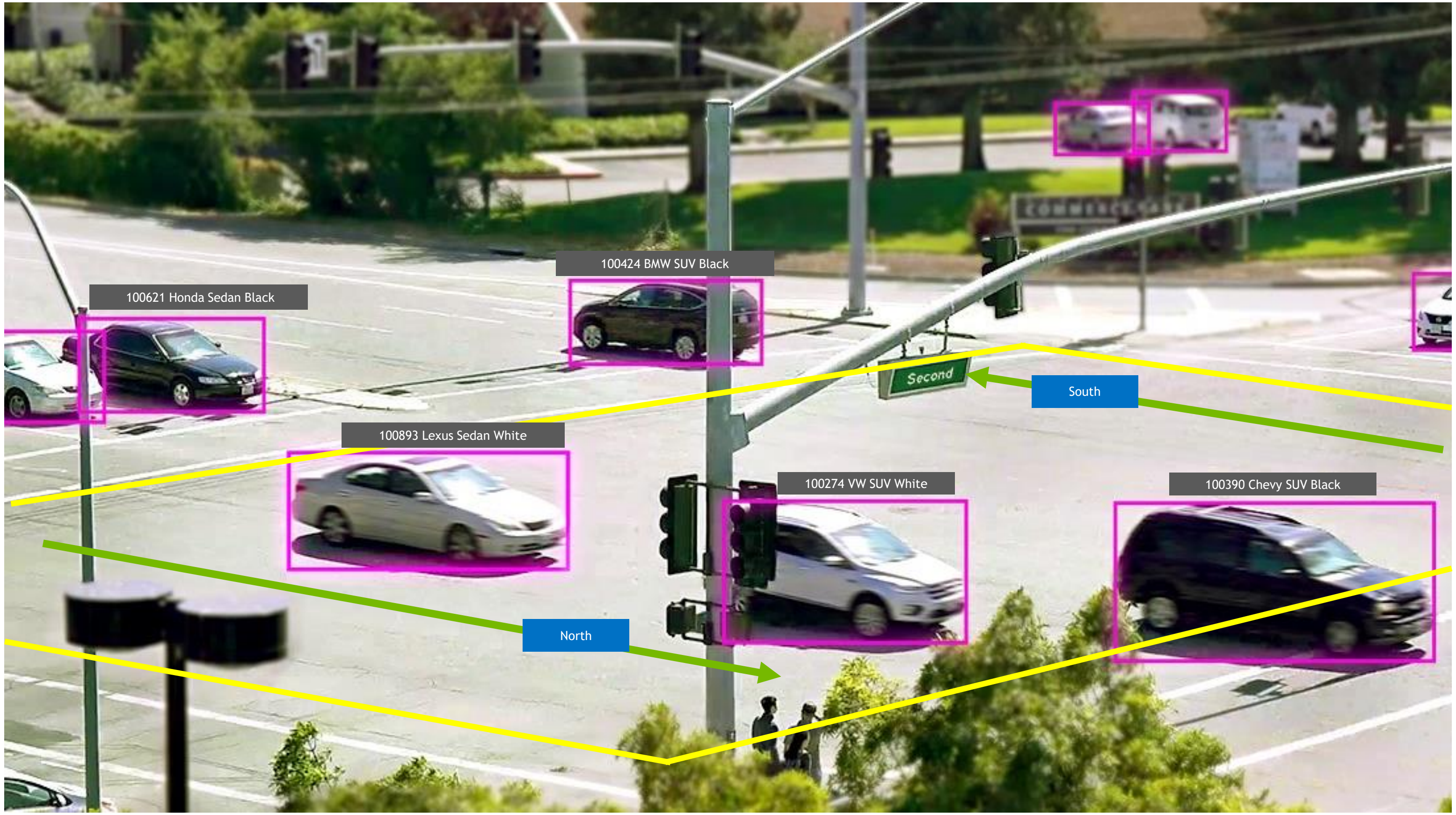
ON SCREEN DISPLAY

Name - nvosd



- Colored line and blending bounding boxes
- Text / labels
- Arrows, lines, circles, ROI

ON SCREEN DISPLAY

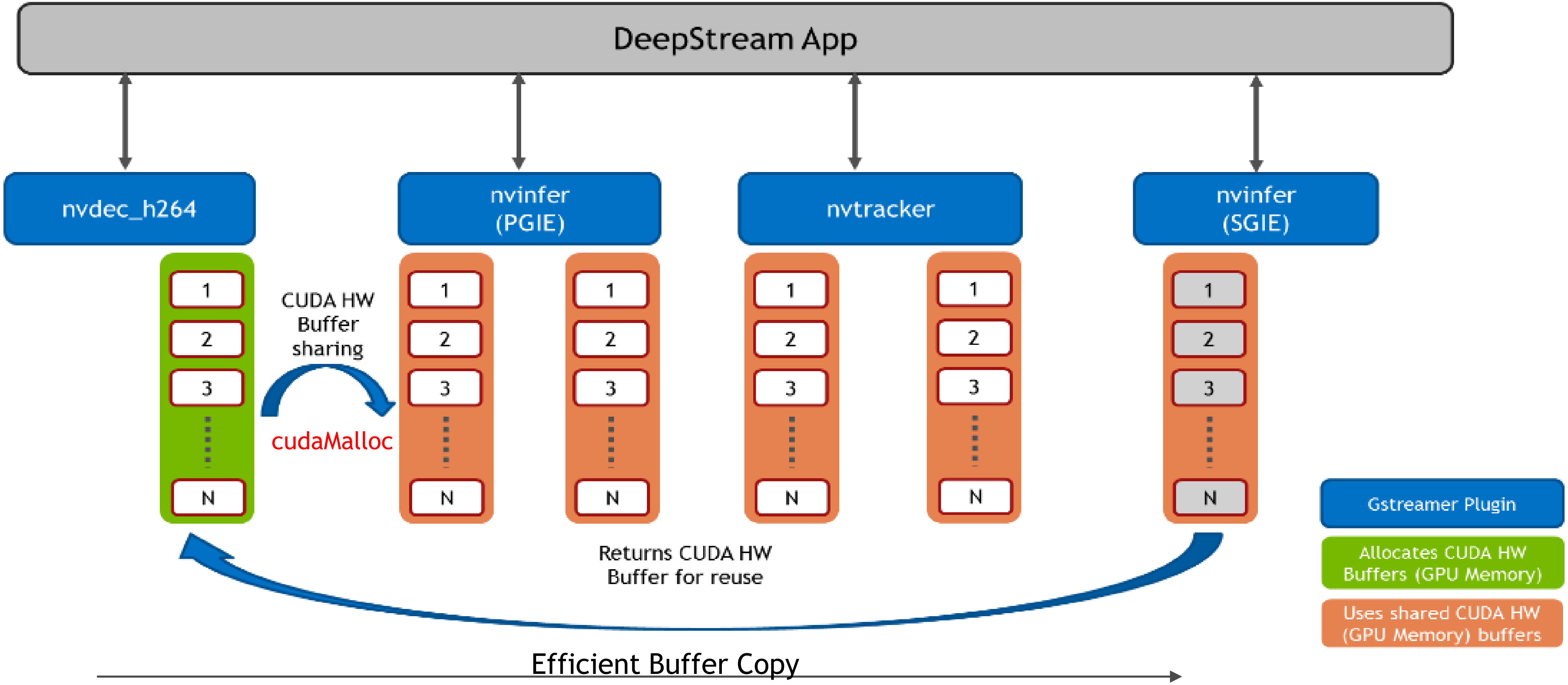


… AND MANY MORE PLUGINS

Plugin Name	Functionality
filesrc	Read from arbitrary point in a file
rtspsrc	Receive data over the network via RTSP
v4l2src	Reads frames from a Video4Linux2 device
xvimagesink	X11 based videosink
x264Enc	H264 Encoder
jpegdec/enc	Decode / Encode in / from JPEG format
Dewarp	Dewarp fish eye video
ALPR	3rd party IP plugin (License plate recognition)

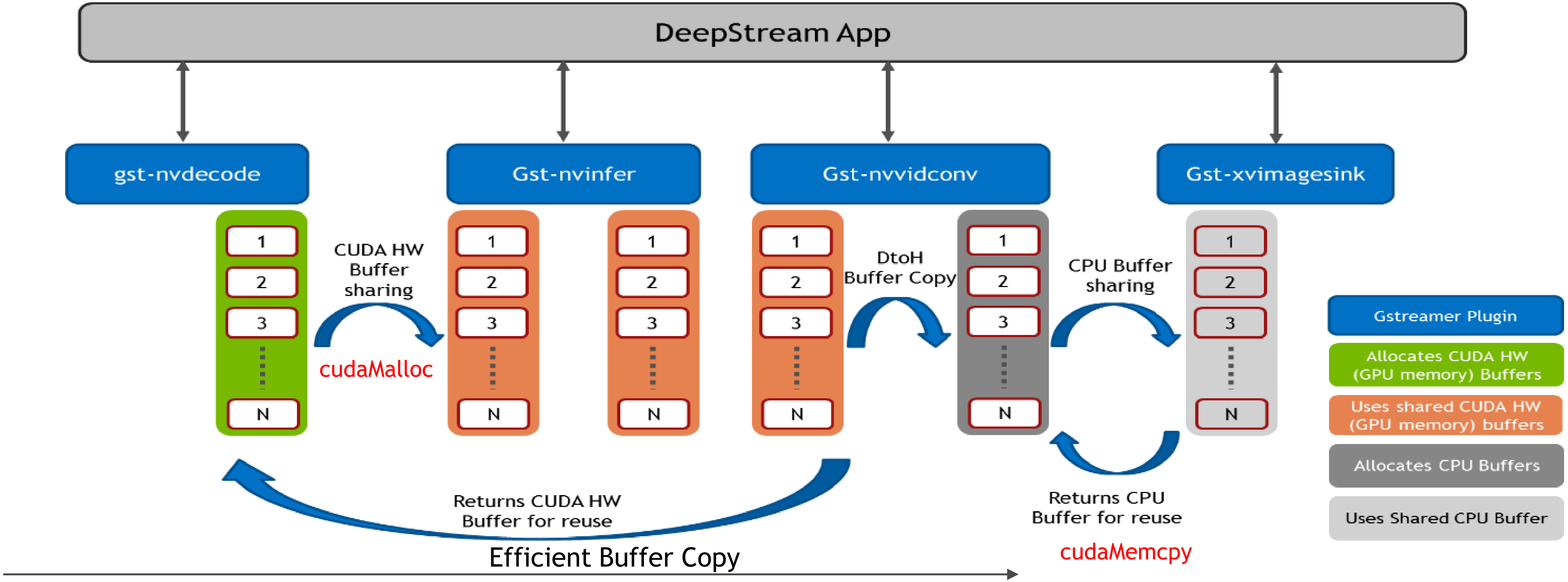
MEMORY MANAGEMENT

Efficient Memory Management



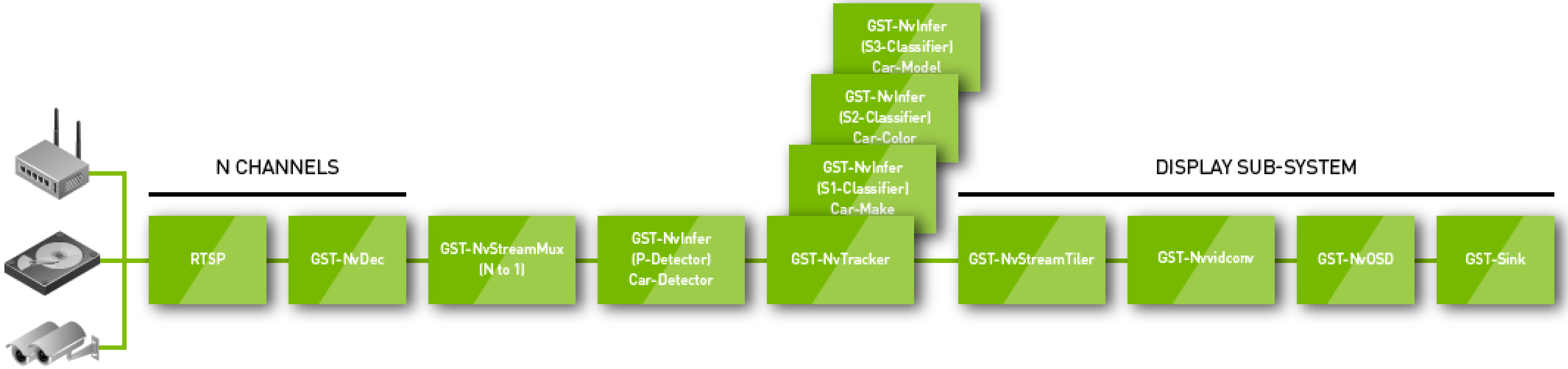
MEMORY MANAGEMENT

GPU to CPU copy



DEEPSTREAM REFERENCE APPLICATION

Vehicle detection, tracking, classification



NVINFER CONFIGURATION FILE

```
gpu-id=0
net-scale-factor=0.0039215697906911373
model-file=../../../samples/models/Primary_Detector/resnet10.caffemodel
proto-file=../../../samples/models/Primary_Detector/resnet10.prototxt
##output labels from detector
labelfile-path=../../../samples/models/Primary_Detector/labels.txt
int8-calib-file=../../../samples/models/Primary_Detector/cal_trt4.bin
net-stride=16
batch-size=1
## 0=FP32, 1=INT8
network-mode=1
num-classes=4
class-thresholds=0.2;0.2;0.2;0.2
class-eps=0.2;0.2;0.2;0.2
class-group-thresholds=1;1;1;1
gie-unique-id=1
parse-func=4
output-bbox-name=conv2d_bbox
output-blob-names=conv2d_cov
```

For details refer to “Reference application configuration” section in user guide

SYSTEM CONFIGURATION

Setup the reference application for performance measurements

```
enable-perf-measurement=1   //To enable performance measurement
perf-measurement-interval-sec=10  //Sampling interval in seconds for performance metrics
flow-original-resolution=1  //Stream muxer flows original input frames in pipeline
#gie-kitti-output-dir=/home/ubuntu/kitti_data/    // location of KITTI metadata files
```

Example Input Source Configuration

```
[source0]
enable=1 // Enables source0 input
#Type - 1=CameraV4L2 2=URI 3=MultiURI  //1) Input source can be USB Camera (V4L2)
// 2)URI to the encoded stream. Can be a file,HTTP URI or an RTSP live source
// 3) Select URL from multi-source input
type=3 // Type of input source is selected
uri=file://../../streams/sample_720p.mp4 // Actual path of the encoded source.
num-sources=1 // Number of input sources.
gpu-id=0 // GPU ID on which the pipeline runs within a single system
```

Enabling and Configuring the Sample Plugin

```
enable=1 //enable sample plugin
gpu-id=0 //GPU id to be used in case of multiple GPUs
processing-width=640   //operating image width for this plugin
processing-height=480 //operating image height for this plugin
full-frame=0  //Operate on individual bounding boxes/objects given by upstream component (for ex. Primary Model)
unique-id=15  //Unique Id (should be >= 15) of plugin to identify its Meta data by application or other elements
```

DEEPSTREAM SINGLE STREAM OUTPUT

DEEPSTREAM REFERENCE APP OUTPUT

DEEPSTREAM REFERENCE APPLICATION

System Configuration & Performance for 25x 720p streams

CPU – Intel® Xeon(R) CPU E5-2620 v4 @ 2.10GHz × 2

GPU – Tesla P4

System Memory – 256 GB DDR4, 2400MHz

Ubuntu 16.04

GPU Driver – 396.26

CUDA – 9.2

TensorRT – 4.0

GPU clock frequency – 1113 MHz

```
GPU 00000000:05:00.0
    Utilization
        Gpu                          : 59 %
        Memory                       : 30 %
        Encoder                      : 0 %
        Decoder                      : 88 %
    GPU Utilization Samples
        Duration                     : 16.44 sec
        Number of Samples            : 99
        Max                          : 83 %
        Min                          : 54 %
        Avg                          : 65 %
    Memory Utilization Samples
        Duration                     : 16.44 sec
        Number of Samples            : 99
        Max                          : 39 %
        Min                          : 28 %
        Avg                          : 32 %
    ENC Utilization Samples
        Duration                     : 16.44 sec
        Number of Samples            : 99
        Max                          : 0 %
        Min                          : 0 %
        Avg                          : 0 %
    DEC Utilization Samples
        Duration                     : 16.44 sec
        Number of Samples            : 99
        Max                          : 100 %
        Min                          : 63 %
        Avg                          : 87 %
```

PERFORMANCE ANALYSIS

KPIs FOR PERFORMANCE

1. **Throughput** (fps) = numbers of frames processed in unit time

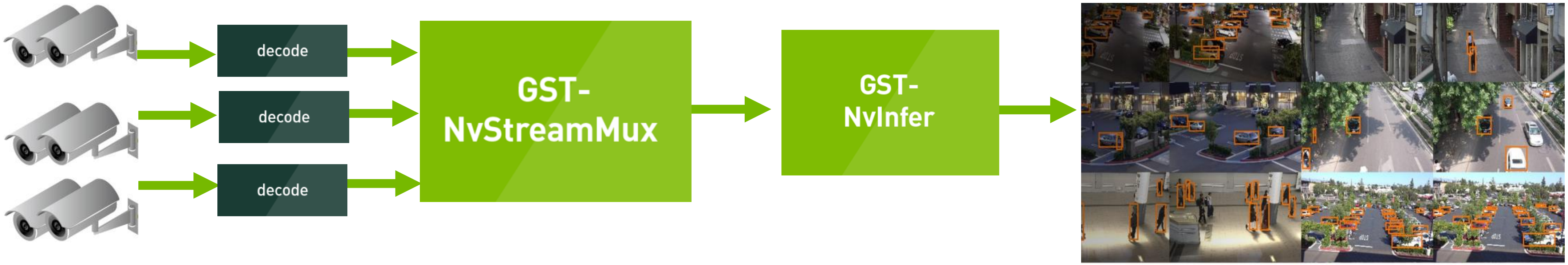
t_x = time when plugin processes buffer

pipeline latency = $t_f - t_a$

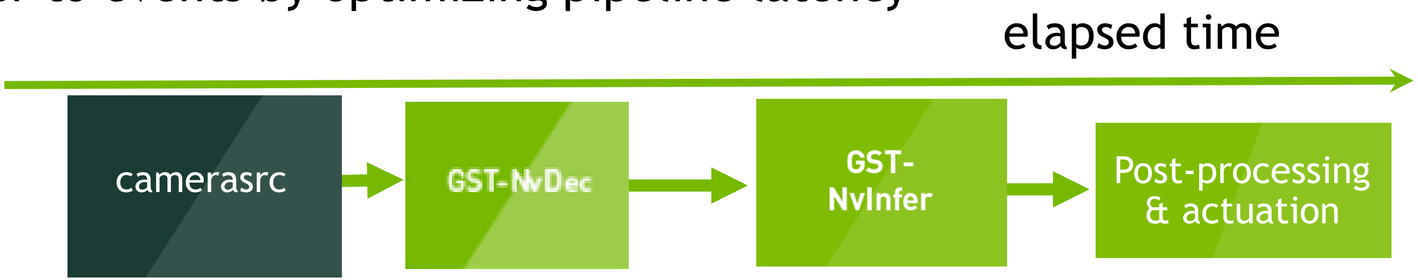
2. **Latency**
3. **Hardware utilization** (decoder, SM, memory, PCI-e bus)
4. **Power**

MOTIVATION

Support increased frame rate and stream count through improved throughput



React quicker to events by optimizing pipeline latency



Enable more sophisticated analysis (eg: cascaded networks, tracking) by supporting larger pipelines

METHODOLOGY FOR PERFORMANCE ANALYSIS

Top-Down Approach

- Measure KPIs & identify gaps (eg: throughput, supported stream count)
- Utilization information (using nvidia-smi) readily suggests bottleneck
- Latency measurements (using gst-logs) confirm rate limiting step
- Kernel execution profiling (using nsight, nvvp) provides fine grained analysis

THROUGHPUT MEASUREMENT

gst probes install callbacks that get invoked when buffer travels through a pad

```
gst_pad_add_probe (display_sink_pad, GST_PAD_PROBE_TYPE_BUFFER,
        display_sink_probe, u_data, NULL);
```

Callbacks stores temporal information about frames flowing through pipeline

Callback maintains various throughput related metrics (avg, max, min, per stream)

LATENCY MEASUREMENT USING GST-LOGS

Decoder latency: 0:00:05.170122161 - 0:00:05.137672361 = 33ms

0:00:05.137621066 <capsfilter1:sink> calling chainfunction &gst_base_transform_chain with buffer buffer: 0x7f9c02bc00, pts 0:00:35.719066666, dts 0:00:35.719033333, dur 0:00:00.016683333, size 56047, offset 172403525, offset_end none, flags 0x2400

0:00:05.137672361 **<omxh264dec-omxh264dec0:sink>** calling chainfunction &gst_video_decoder_chain with buffer buffer: 0x7f9c02bc00, pts 0:00:35.719066666, dts 0:00:35.719033333, dur 0:00:00.016683333, size 56047, offset 172403525, offset_end none, flags 0x2400

0:00:05.170122161 **<src_0:proxypad3>** calling chainfunction &gst_proxy_pad_chain_default with buffer buffer: 0x7f714c5020, pts 0:00:35.719066666, dts 99:99:99.999999999, dur 0:00:00.016683333, size 808, offset none, offset_end none, flags 0x0

0:00:05.170196720 <nvvconv0:sink> calling chainfunction &gst_base_transform_chain with buffer buffer: 0x7f714c5020, pts 0:00:35.719066666, dts 99:99:99.999999999, dur 0:00:00.016683333, size 808, offset none, offset_end none, flags 0x0

PERFORMANCE BEST PRACTICES

- Reduced precision inference (INT8/FP16)
- Use increased batch size for inference
- Use appropriate frame rate for input video
- Optimize data movement between system and device memory
- Use CUDA streams to maximize execution parallelism

CUSTOM PLUGINS

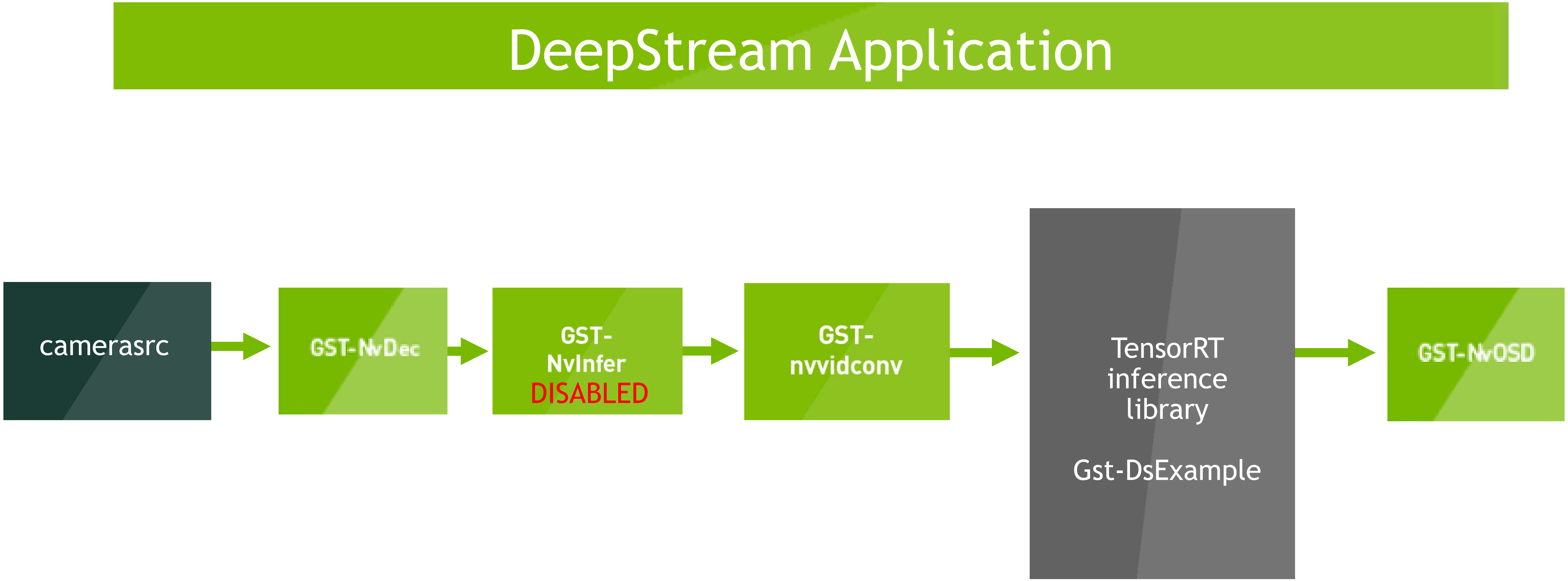
CUSTOM PLUGIN FOR OBJECT DETECTION

Implement custom TensorRT plugin layers for your network topology

Integrate your TensorRT based object detection model in DeepStream

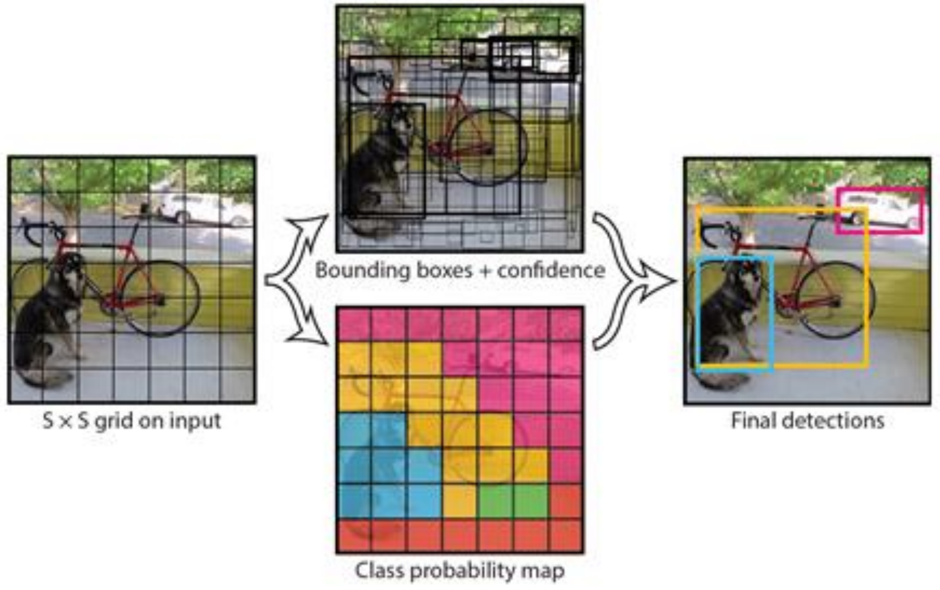
1. Train an object detection model to be deployed in DeepStream
2. Import the model into TensorRT
3. Wrap the TensorRT inference within the template plugin in DeepStream
4. Run in DeepStream

PIPELINE


DeepStream Application
camerasrc
GST-NvDec
GST-
NvInfer
DISABLED
GST-
nvvidconv
TensorRT
inference
library
Gst-DsExample
GST-NvOSD

YOLO V2 OVERVIEW

- Object detection model
- Input: RGB image
- Output: Bounding boxes
- Convolutional network with anchor boxes
- Inference in a single forward pass



DARKNET TO TENSORRT

How to create YOLO in TensorRT

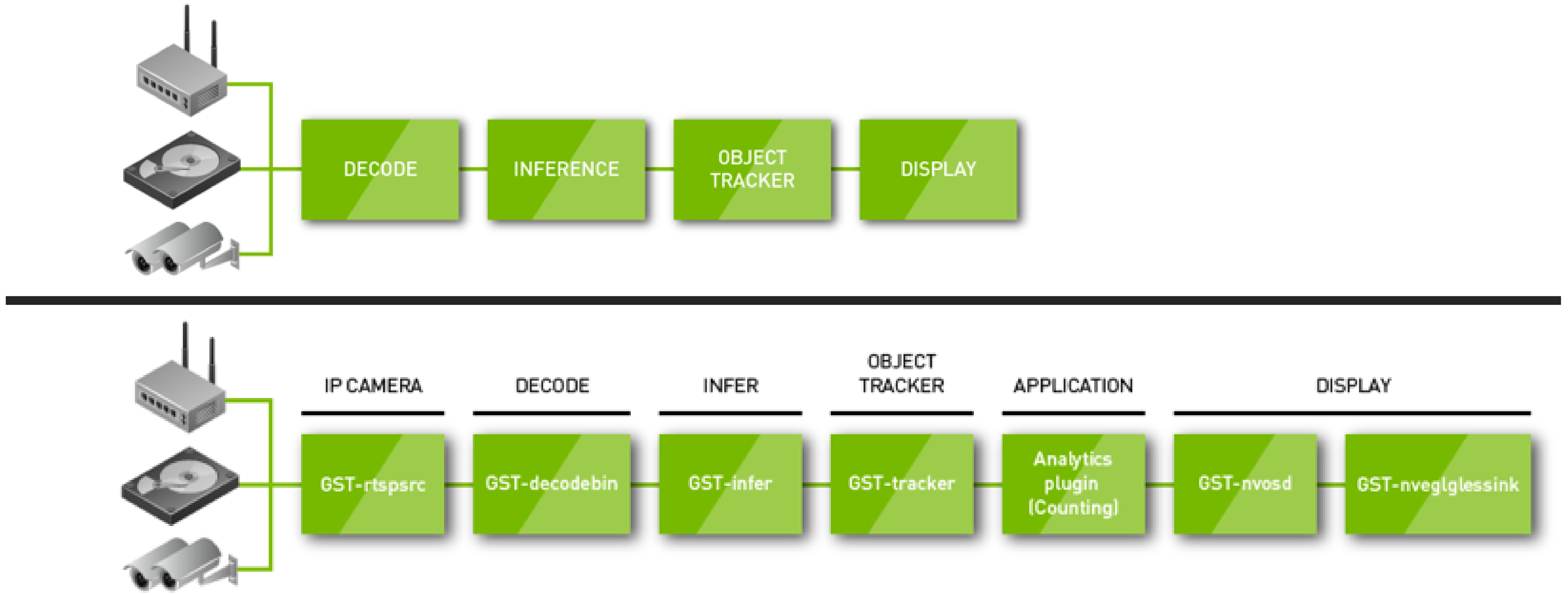
- TensorRT’s network builder API
  - Create the network architecture from scratch
  - Read the darknet weights into the network

TENSORRT PLUGIN FACTORY

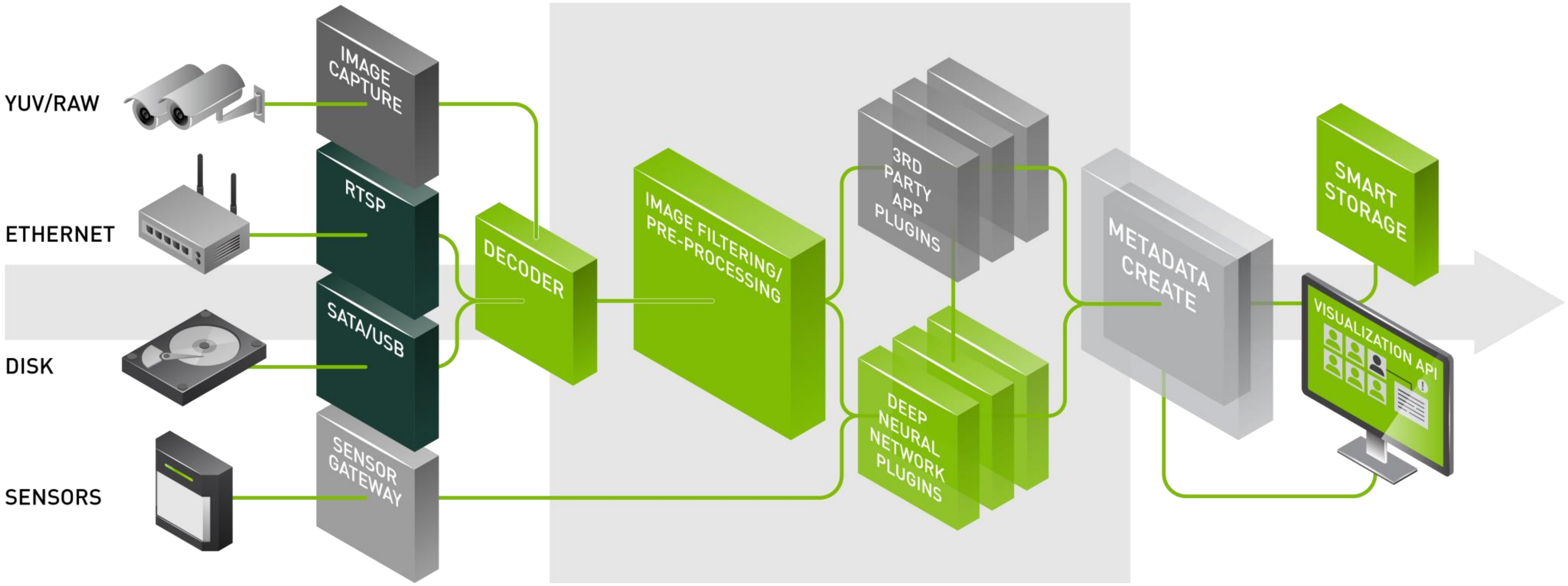
- TensorRT v4 currently doesn’t natively support Leaky ReLU, Reorg and Region layers
- Solution: TensorRT Plugin Factory
- Create interfaces for parser and runtime to add plugins
- NVIDIA provides these plugins with the TensorRT SDK
- Caveats:
    - Repurpose the PReLU to a leaky ReLU
    - Region Layer computes only logistic activations and softmax scores
    - Decoding predictions to bounding boxes should be handled outside the TensorRT engine

CREATING APPLICATIONS WITH DEEPSTREAM

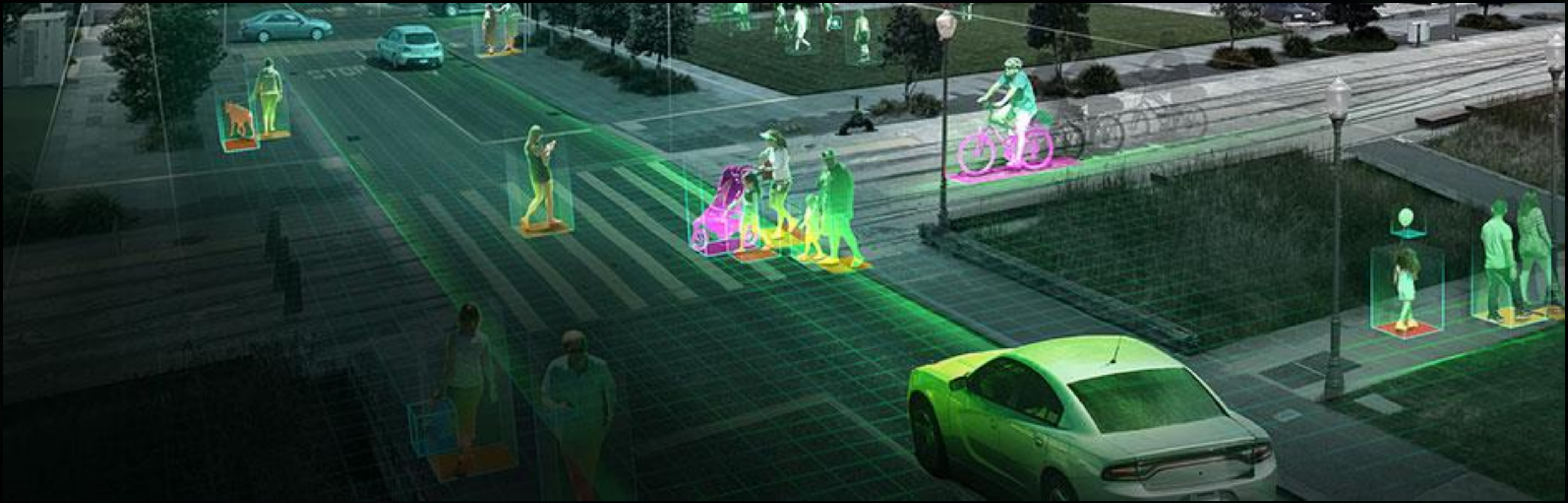
Object detection and counting



DEEPSTREAM MODULAR APPLICATION



START DEVELOPING WITH DEEPSTREAM



DeepStream . Explore Metropolis . Intelligent Video Analytics Forums

QUESTIONS?