H P C

High Performance Computing

Agenda

- **What do we do**
  - Definitions, terms, technologies, motivations & examples…
- **Computer Vision:**
  - Classic algorithm
  - Deep Learning

What Do We Do?

HPC Definition

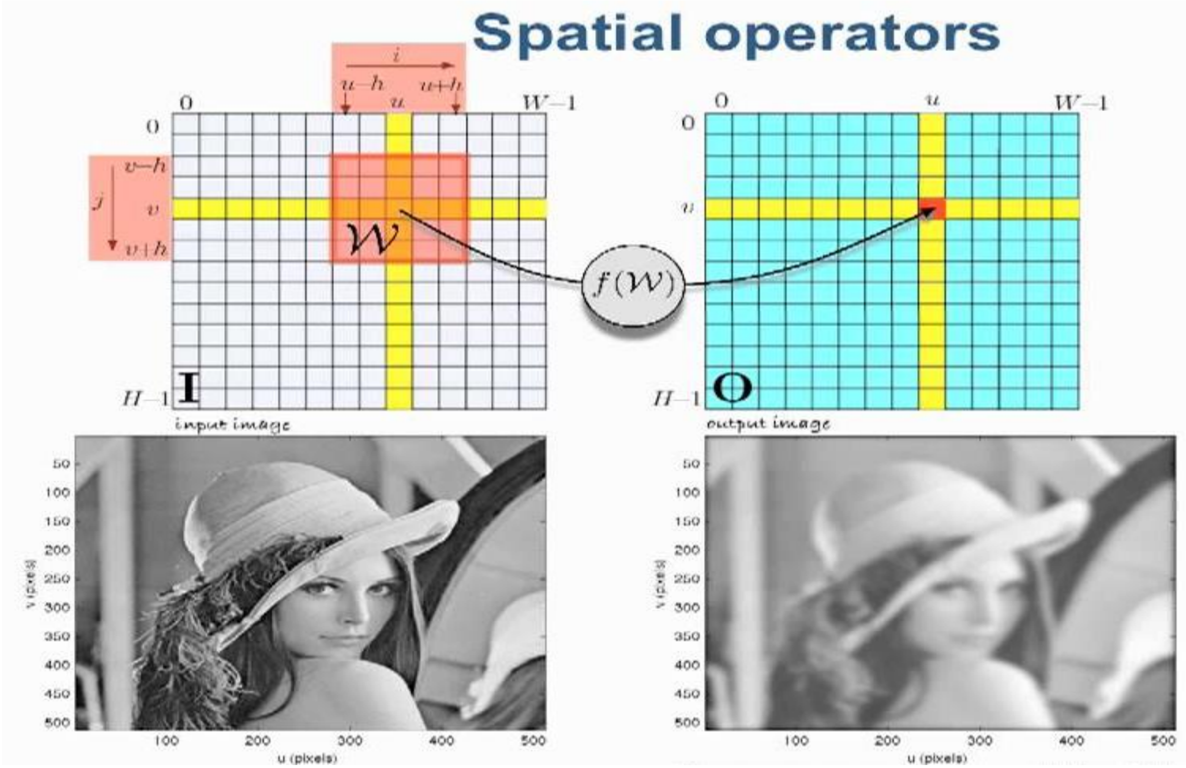
- The use of **parallel** processing for running advanced application programs **efficiently** and **quickly:**
  - *Speed* - reducing time to solution
  - *Energy efficiency* - doing more with less power
  - *Upscaling* - handling larger problems
  - *High throughput* - handle large volumes of data in real-time

What Do We Do?

Use HPC capabilities for Image Processing

- Large set of data (Large frame = Lots of pixels)
- Same operation \ calculation on each datum (pixel)

=> Parallel computation is self-evident



What Do We Do?

In numbers:

- HD frame = 1920 X 1080 ≈ 2MPixel

  3 color components: RGB

  => 6MB
- SkEye Sensor = 10,000 X 7096 ≈ 70MPixel

  2 bytes per component

  => **140 MB**
- Matlab + CPU processing

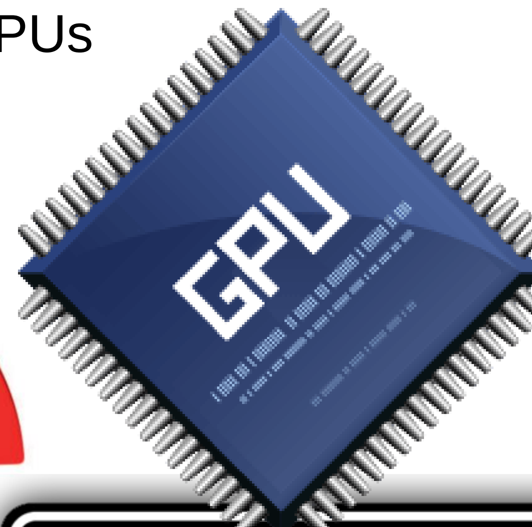
  ~ 2 seconds
- CUDA/OpenCL + GPU processing

  ~ several milliseconds

What Do We Do?

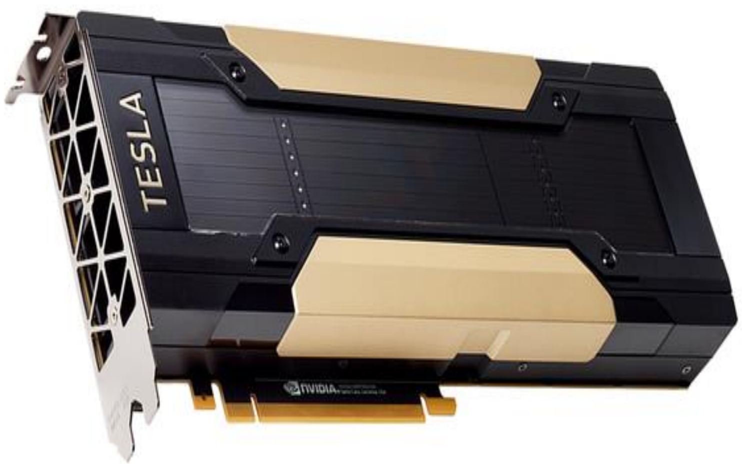
Our Main Tool Set

- **CPU & GPU** – Hardware
- **OpenCL** – Development framework over any GPU
- **CUDA** – Development over NVIDIA GPUs
- **Third party**
  - Boost C++
  - OpenCV



GPU - Graphics Processing Unit

- Originally designed for graphic implementations
- Contains large amount of cores, designed to work **in parallel**
- GPGPU = General Purpose GPU



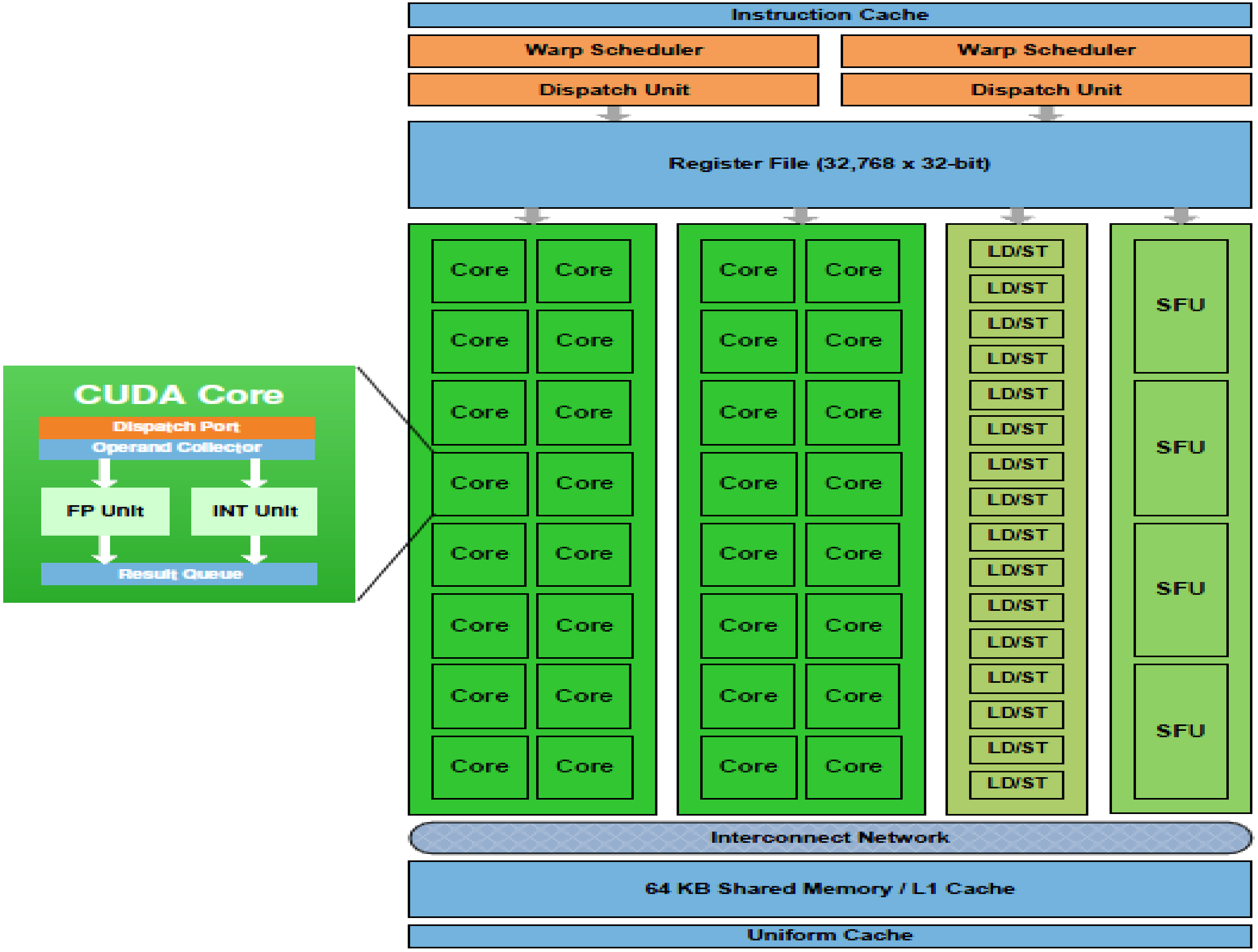
OpenCL - Open Computing Language

- **Open standard language for parallel computation on heterogeneous platforms (CPU, GPU, FPGA, etc.)**
- Producing small programs / calculations (Kernel) that work concurrently on “work items”
- Useful for working on all pixels of an image in parallel (each one considered a work item).
- Features
  - Dynamic library implemented by the vendor
  - Syntax resembling C language with APIs to control the GPU

CUDA - Compute Unified Device Architecture

- **Parallel computation on NVIDIA GPUs (non standard).**
- **Provides better fit and higher acceleration on NVIDIA GPUs.**
- **Tools and libraries provided by NVIDIA**
  - Common algorithms
  - FFT, convolutions, and other complicated methods
  - Use of tensor cores (to support deep learning)
  - Multi GPU support
- **NVIDIA GPUs provide limited support in OpenCL, some platforms don’t.**

CUDA – Micro architecture



Special
Function
Unit

Parallel computation example

Adding two vectors

a	1	2	3	4	5	6	7	8
b	1	1	1	1	1	1	1	1
c	2	3	4	5	6	7	8	9

+

Add vector of N elements

Assume global work size of **Nx1**

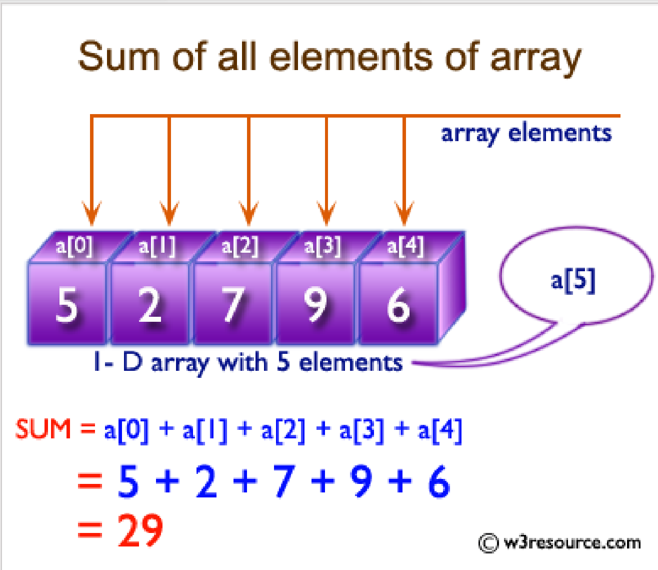
CPU:

```
void vecAdd(const int N,
const float * a,
const float * b,
float * c)
{
    for (int i=0; i< N; ++i)
    {
        c[i] = a[i] + b[i];
    }
}
```

GPU (openCL):

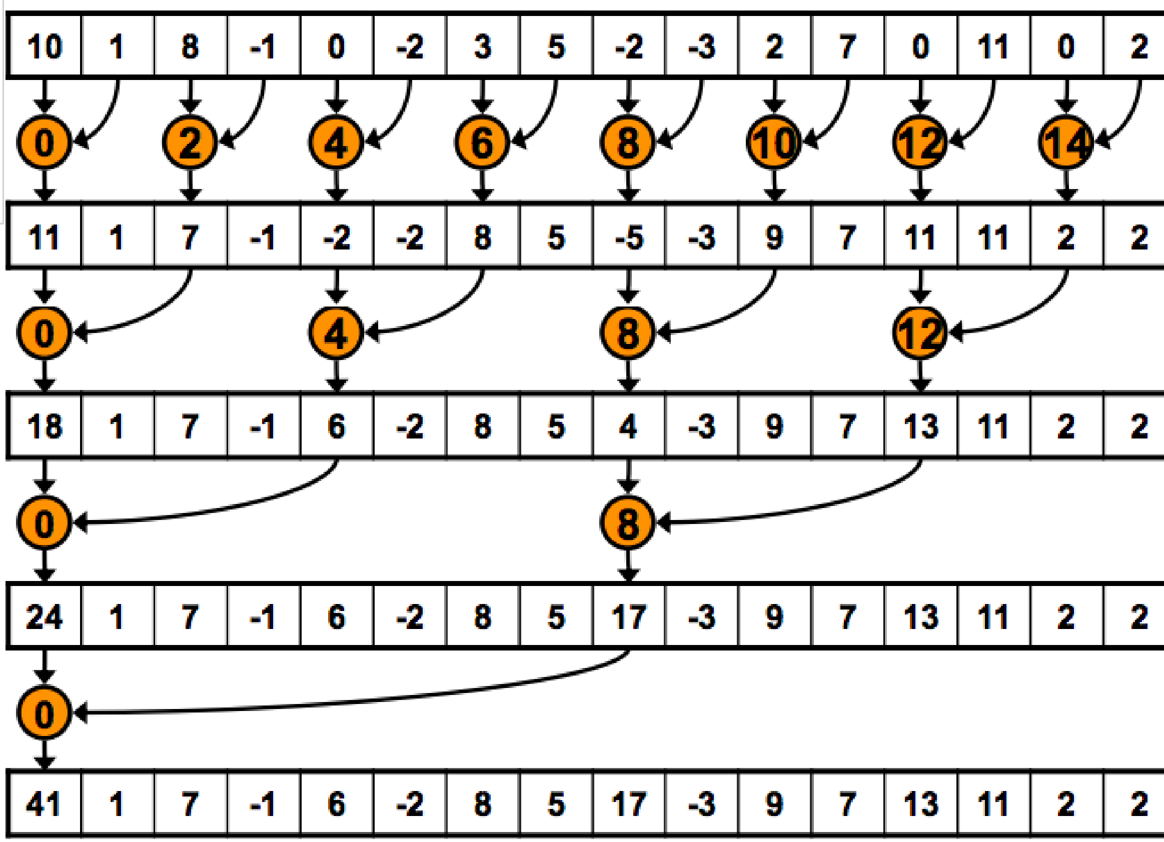
```
__kernel void vecAdd(
    __global const float * a,
    __global const float * b,
    __global float * c)
{
    int i = get_global_id(0);
    c[i] = a[i] + b[i];
}
```

Parallel comp example: Array Sum



CPU

GPU



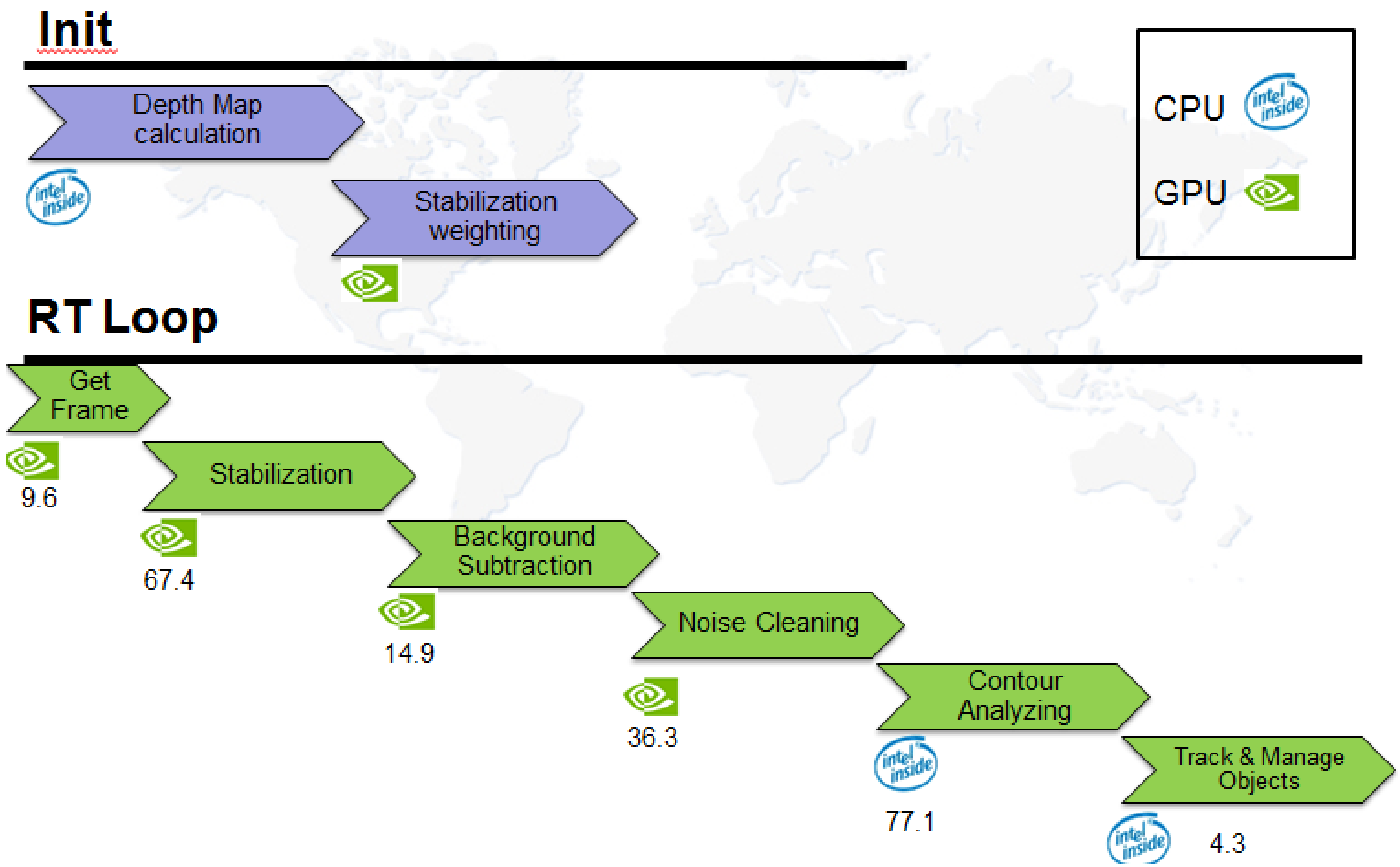
Two projects we've done

GroundEye – Classic algorithm

DLEWARE – Deep Learning

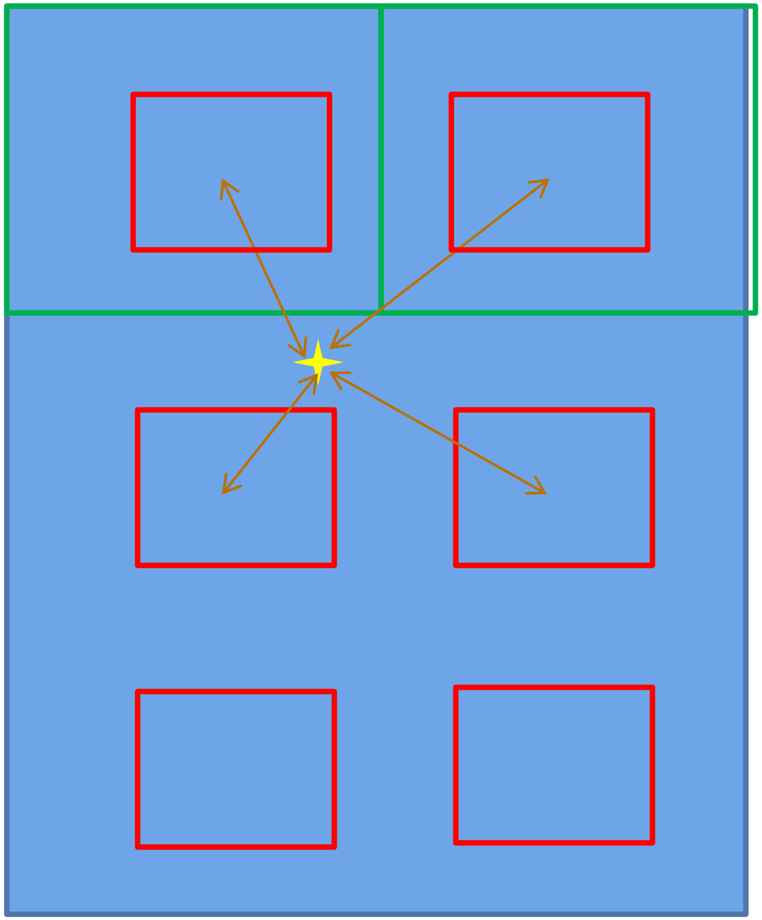
GroundEye: Video Motion Detection

- **VMD: Video Motion Detection for 2 types of objects at various distances**
  - An intensive image stabilizer corrects movements of the camera.
  - By sampling the background we detect the blobs that are different
  - Objects movement is analyzed to filter out jittering objects (e.g. trees)



GroundEye: VMD stabilization

Get Frame → Stabilization → Background Subtraction → **Noise Cleaning** → Contour Analyzing → Track & Manage Objects

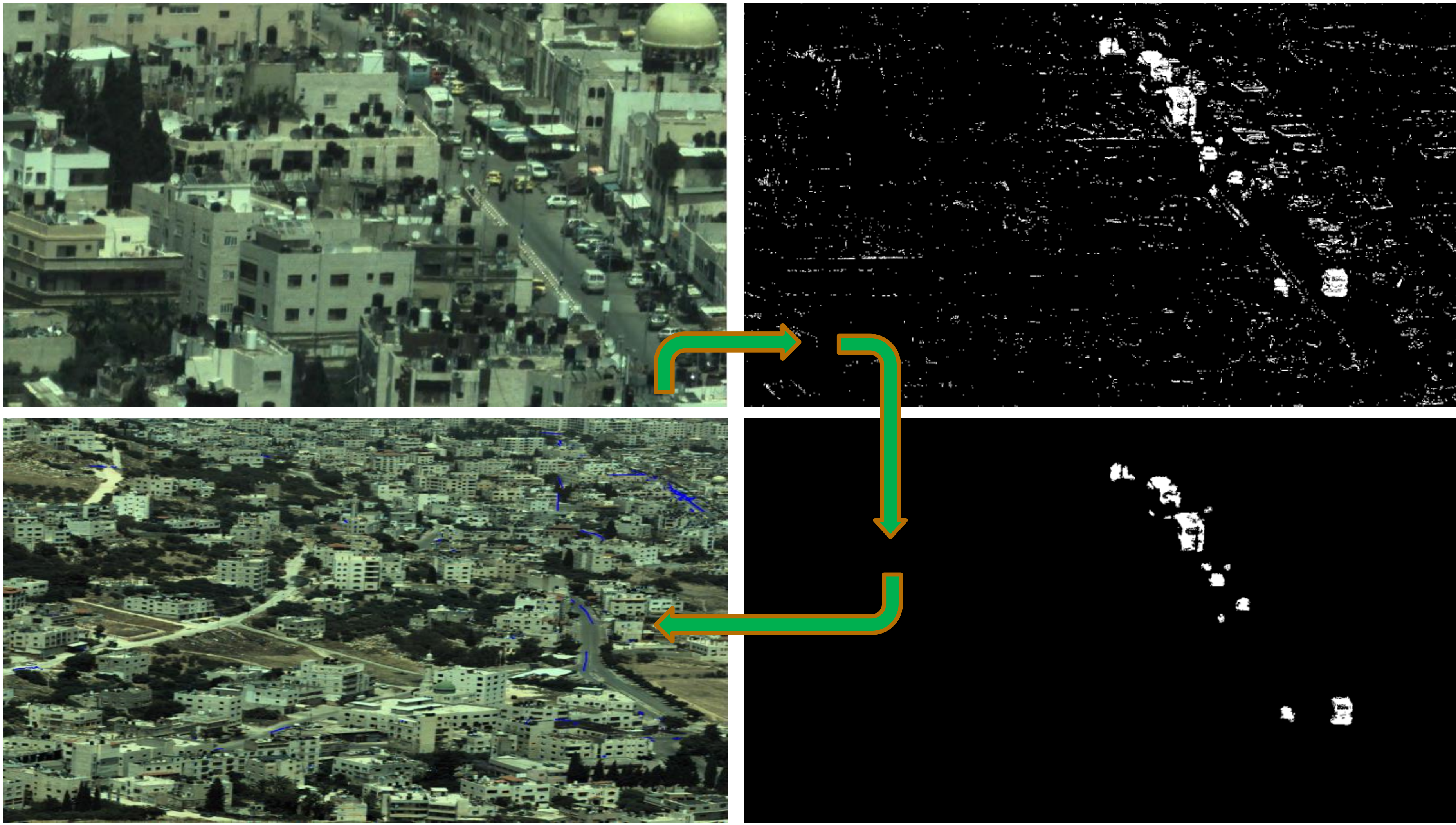


GroundEye: VMD stabilization

Get Frame → Stabilization → Background Subtraction → **Noise Cleaning** → Contour Analyzing → Track & Manage Objects

GroundEye: VMD object detection

Get Frame → Stabilization → Background Subtraction → Noise Cleaning → Contour Analyzing → Track & Manage Objects



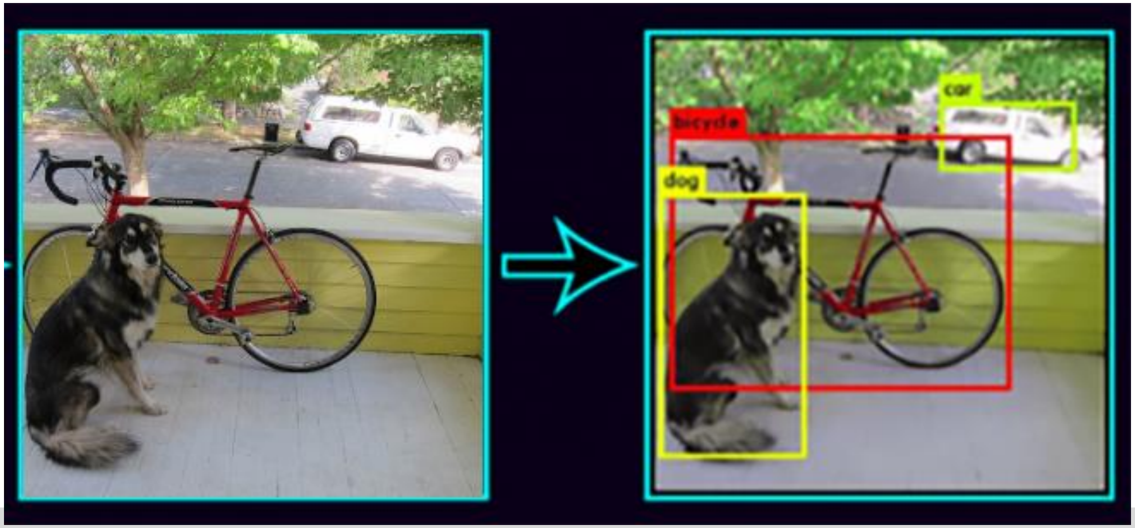
GroundEye: VMD objects tracking

Get Frame → Stabilization → Background Subtraction → **Noise Cleaning** → Contour Analyzing → Track & Manage Objects

DLEWARE: Deep Learning Software

Given an input video frame,
detect all instances from the classes it had been trained to detect,
delivering their characteristics:

- Identification confidence
- Type confidence
- Size: Width & Height
- Position: CenterX & CenterY



Method

- **Unlike classic algorithms,**
  that dictate what each one of the image categories of interest look like directly in code: Mature, proven, and optimized, but not versatile.
- **In Deep Learning,**
  you provide the computer with many examples of each image class and develop a way to deduce from them the visual appearance of each class. can offer greater accuracy and versatility but demands large amounts of computing resources.
- **To do that,**
  - first accumulate a training dataset of labeled images (Training phase),
  - then feed it to the computer to process the data (Inference Phase).

Inference Phase is where HPC comes into the picture!

HW & SW resources

Three platforms:

	PC#1	PC#2	Embedded Jetson
Operating system	Windows	Linux (Ubuntu)	L4T – Linux For Tegra
CPU	Intel Xeon	Intel Xeon	ARM
GPU	NVIDIA GeForce GTX 1080	NVIDIA GeForce GTX 1080	NVIDIA Tegra
DL framework	Tensorflow (Built from sources)	Tensorflow (TBC)	Tensorflow (TBC)
DL Network	CNN – YoloV3	CNN – YoloV3	CNN – YoloV3
Programming Language	C++ & CUDA	C++ & CUDA	C++ & CUDA
Third party SW	TensorRT, CuDNN, Boost, OpenCV…	TensorRT, CuDNN, Boost, OpenCV…	TensorRT, CuDNN, Boost, OpenCV…

Thank You!

Q&A

DLEWARE – Deep Learning Engine

Capture Image

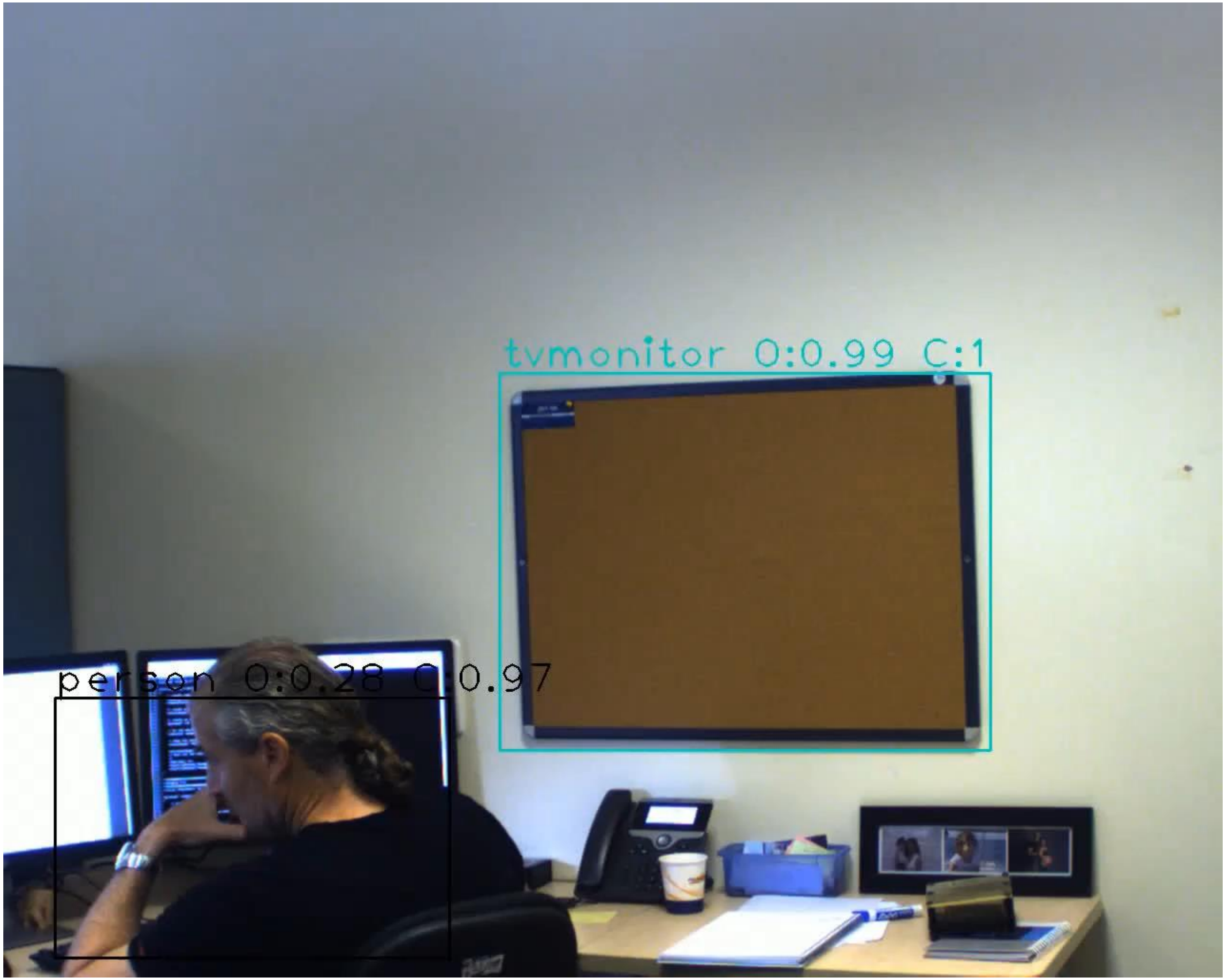
Adapt image size & format

YOLOv3 on Tensorflow or TensorRT

Translate tensor output

Filter out low confidence or high overlap

Undo adaptations and send result



DLEWARE – Deep Learning Engine

Thank You!

Q&A