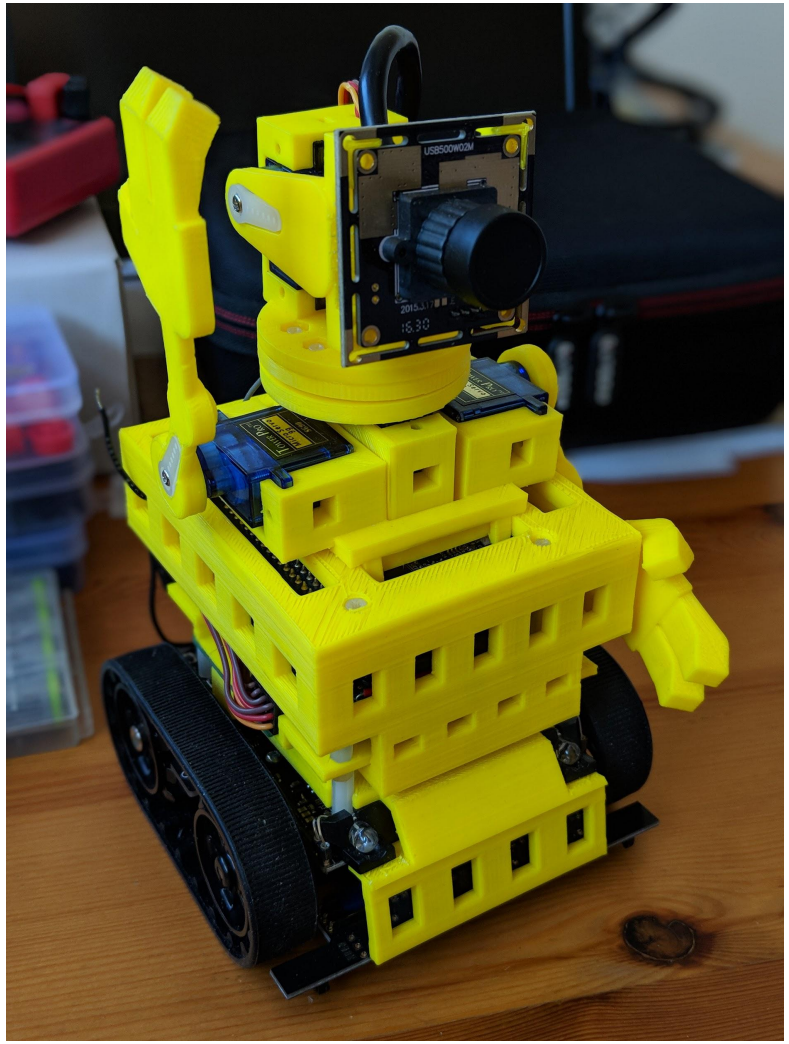
Real-time Vision Processing on Kubernetes

Working with Data Locality

The Project

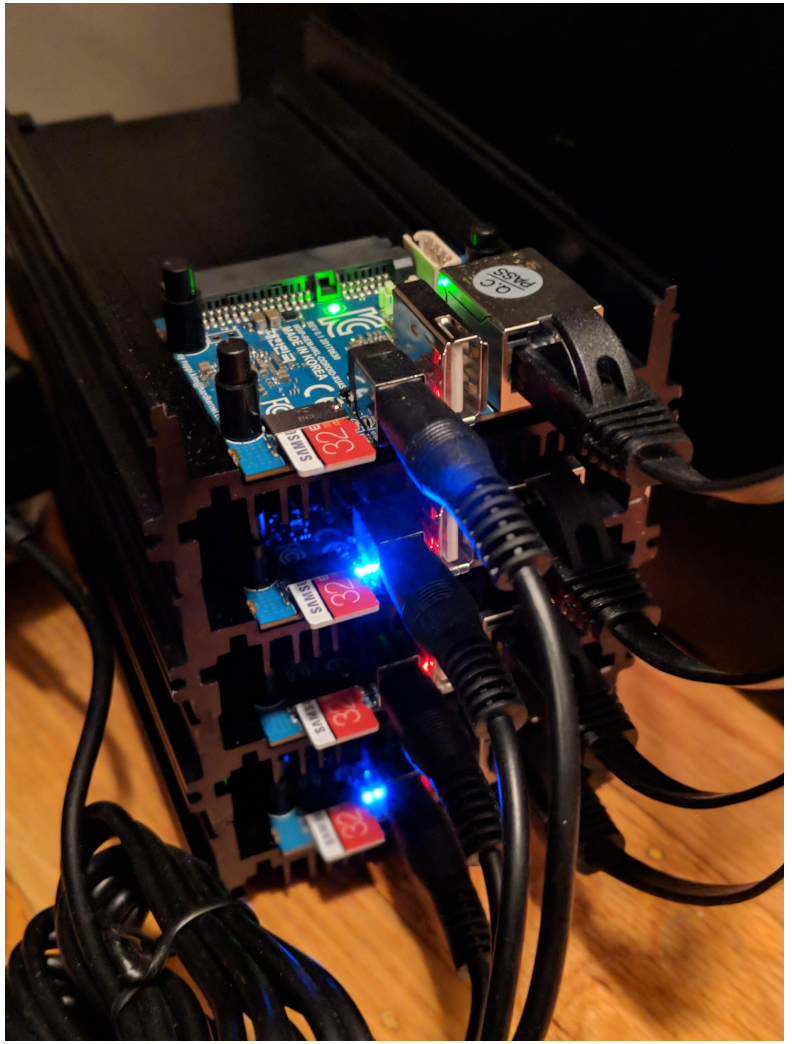


The Robot

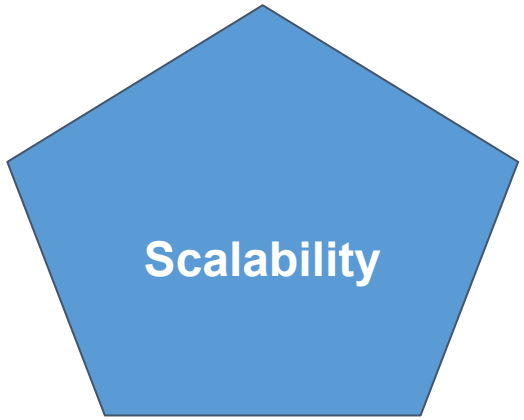
Kubernetes Cluster

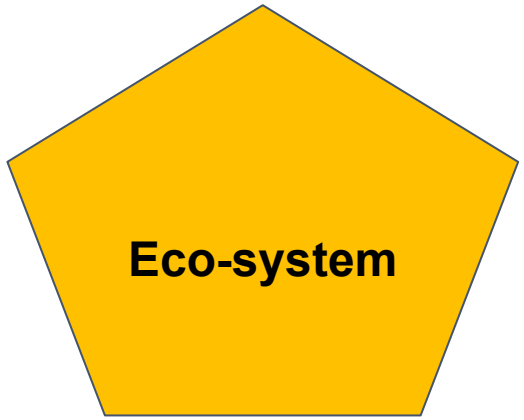
Goal!

Offload Vision Processing & Intelligence Logic



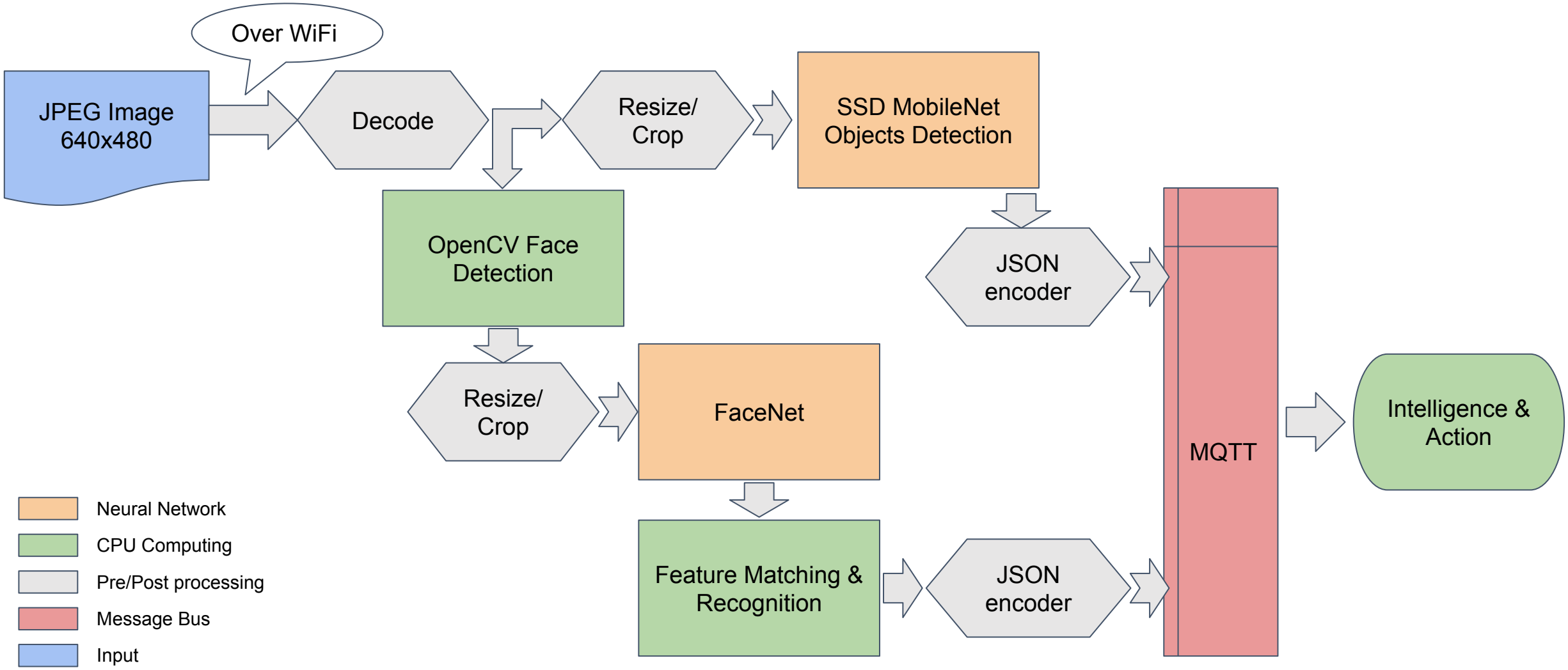
Why Kubernetes?





Real-time Vision Processing is an attempt with uncommon workloads on Kubernetes.

Vision Pipeline (part)



Challenges

- Real-time Image Processing
  - Decision must be effective within ~30ms delay
  - Not-so-reliable Home WiFi Network
  - Pre-trained Neural Networks run longer
- Neural Network Accelerator Support
  - Movidius Neural Compute Sticks on Kubernetes
  - Distributed Pipeline
- Kubernetes on ARM
  - Big-little architecture

Challenge - Hardware

KubeCon | CloudNativeCon

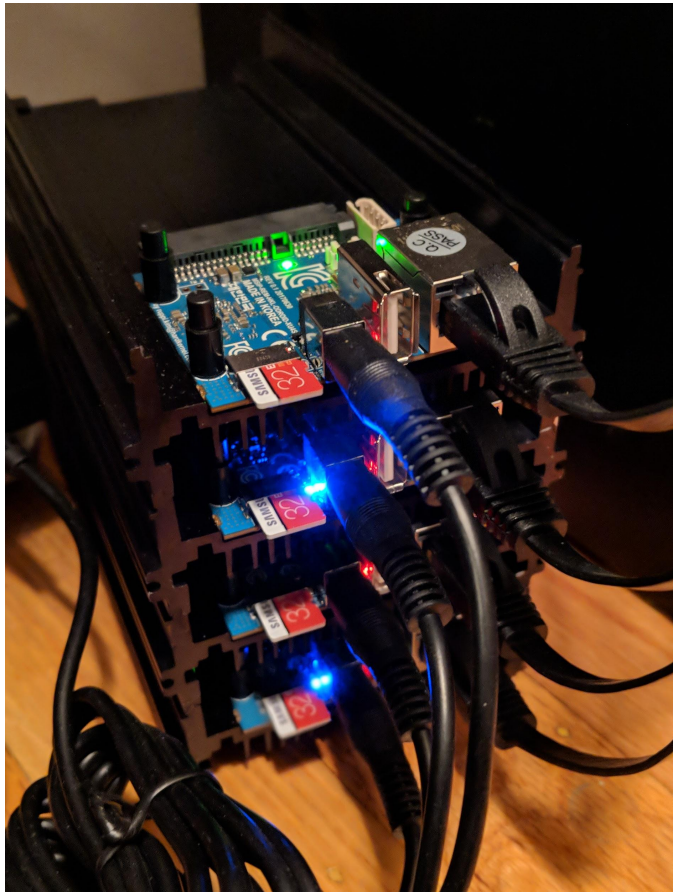
North America 2018

The Cluster and Neural Network Accelerators

What we have...

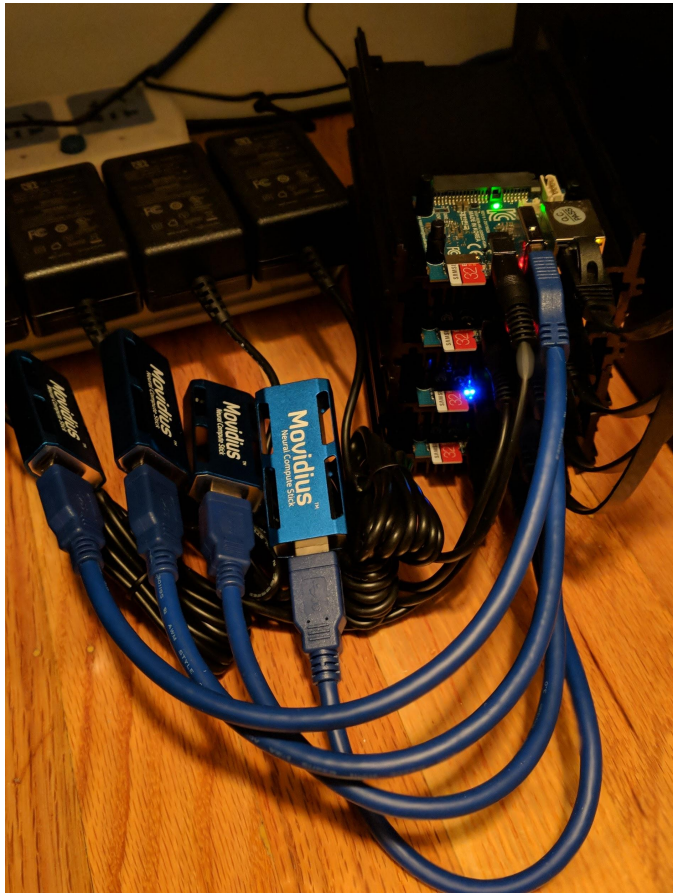
- The Robot
  - Single 720p Camera (working on 640x480)
  - Orange Pi Lite with built-in Wifi/Bluetooth
  - Zumo base (ATmega32U4 - arduino compatible)
- The Cluster
  - Ordroid HC1 x4 (*Samsung Exynos5422 Cortex-A15 2Ghz and Cortex-A7 Octa core CPUs*)
  - Movidius Neural Compute Stick x4
  - Kubernetes 1.9 on Ubuntu 18.04

The Cluster & Accelerators



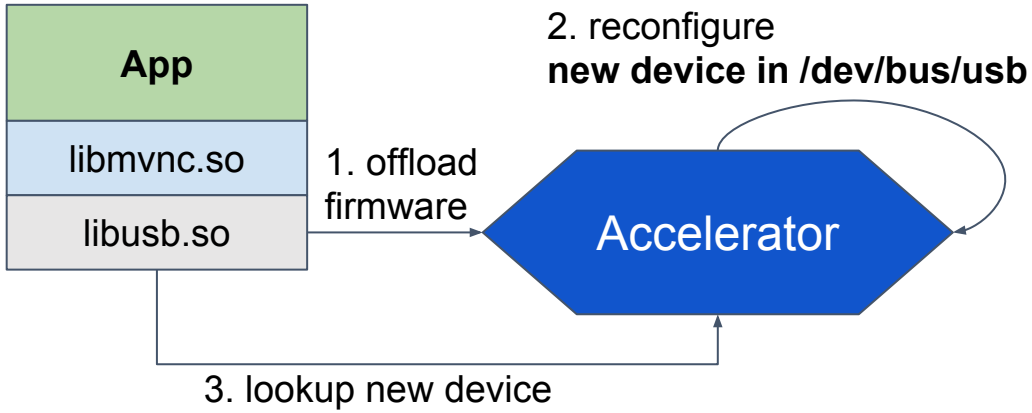
+

=



Accelerators on Kubernetes

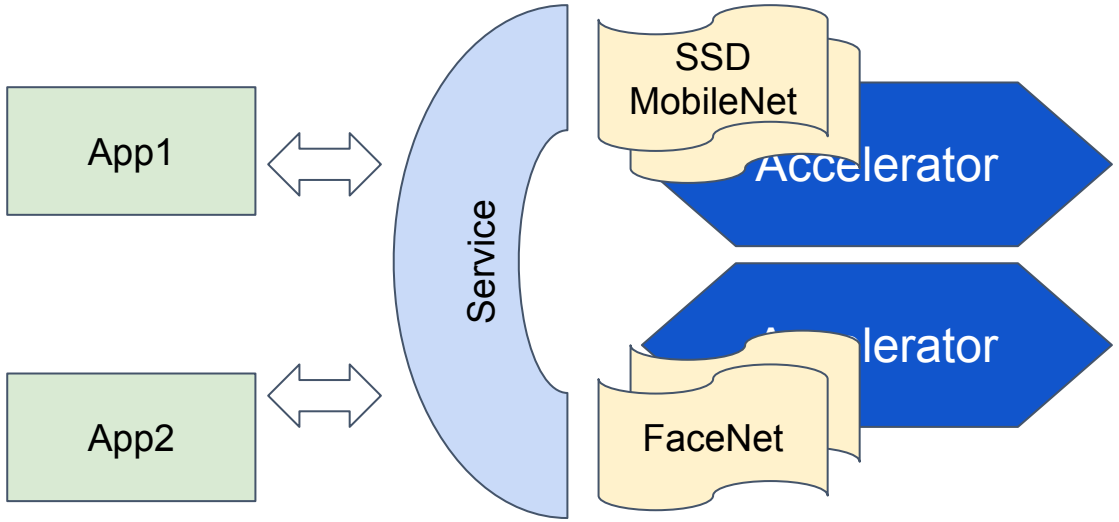
- Access Movidius Compute Stick inside container
  - Special initialization process (unlike CUDA)
  - Solution
    - Privileged container with host network
    - Bind mount host /dev into container




```
docker run --privileged
    --net=host
    -v /dev:/dev ...
```

Offloading Model

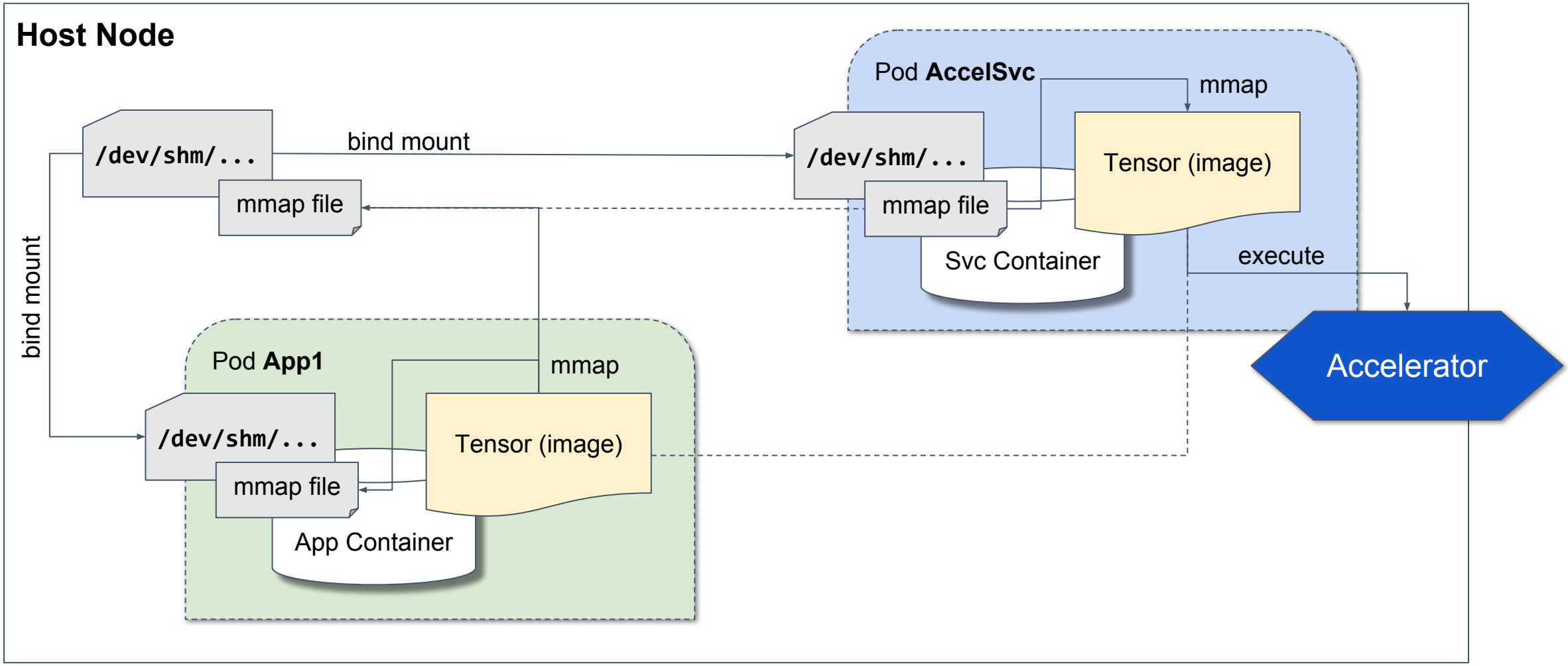
- Offloading model takes long (N ms to a few seconds)
  - Switching models during runtime is not feasible
  - Solution
    - Pre-load models to accelerator
    - Service for models
    - Distributed Pipeline



Service for Accelerator

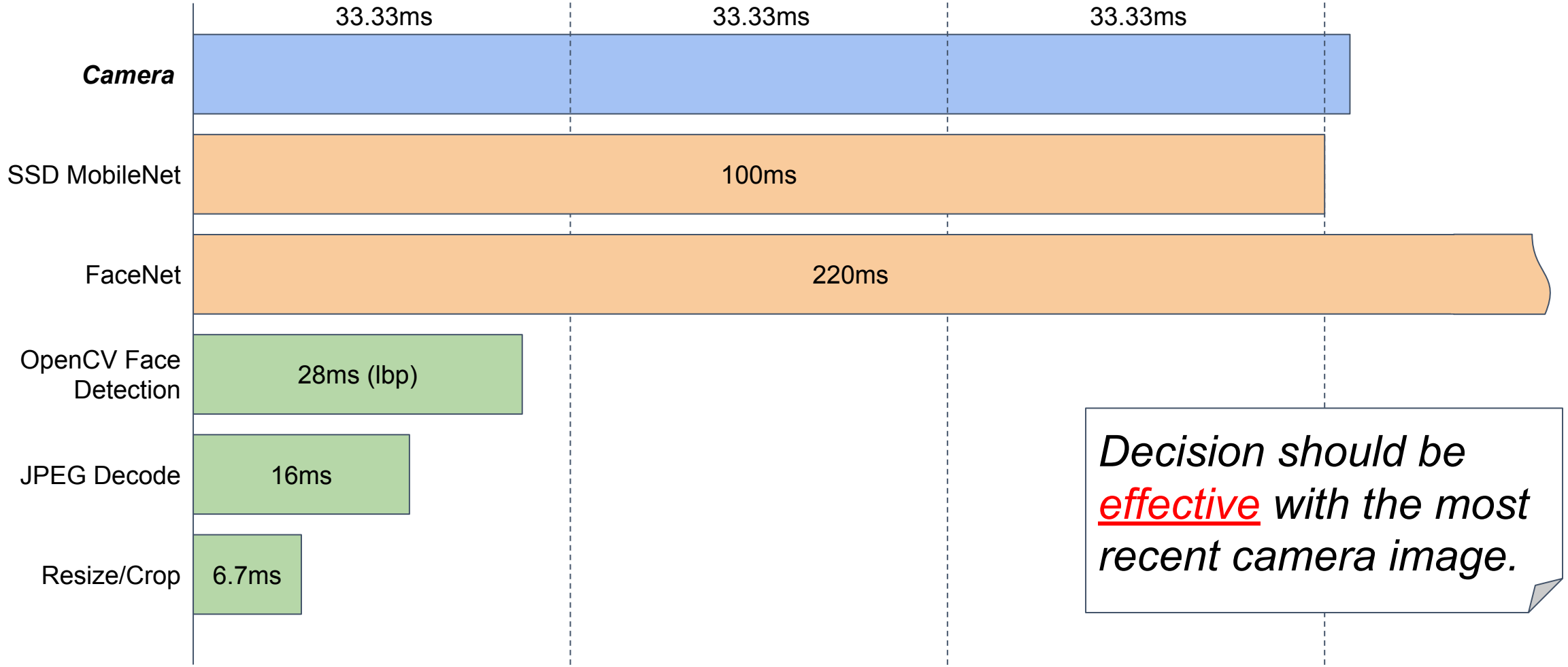
- Input data is large
  - Uncompressed image (~0.5MB for 300*300 FP16 RGB)
  - Short living (<3s)
- Serving with shared memory between Pods
  - Host mountpoint of tmpfs (e.g. /dev/shm)
  - Bind mount sub-path into Accelerator service Pod and application Pod
  - Use mmap to share input data

Serving with shared memory



Real-time Processing

The Cost



Decision should be effective with the most recent camera image.

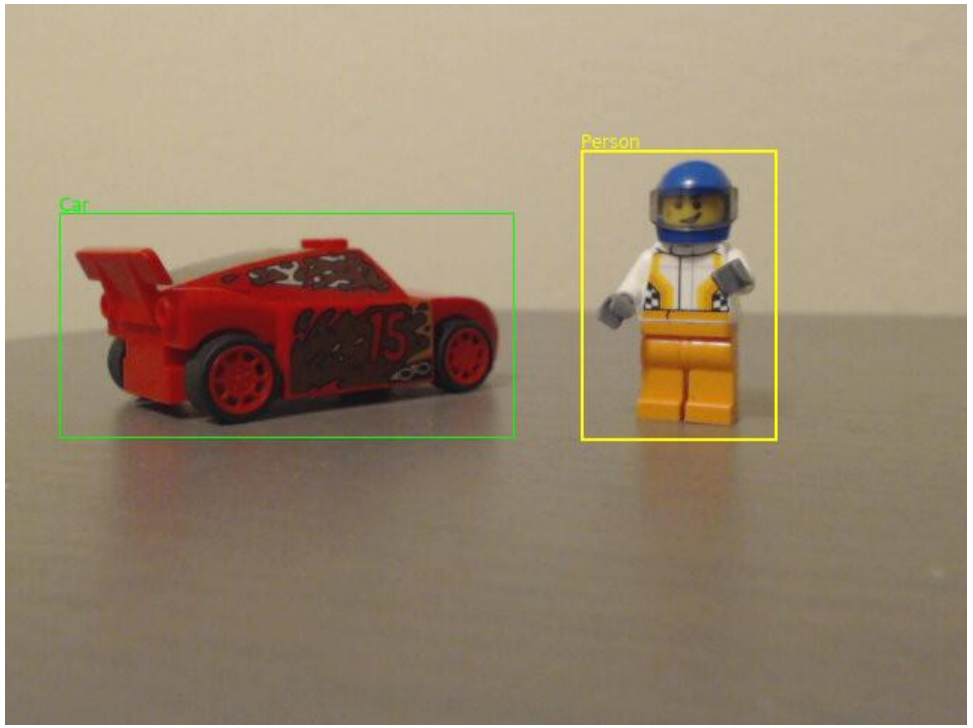
Real-time Processing

- Challenges
  - ***Effective*** decision within ~30ms delay
  - Home WiFi performance is inconsistent
  - Running pre-trained model takes long (even with accelerators)

*Note: **Effective** means the decision should be responsive. It appears to be made within 30ms. Regardless of the actual time the decision was based on, perhaps 100ms ago.*

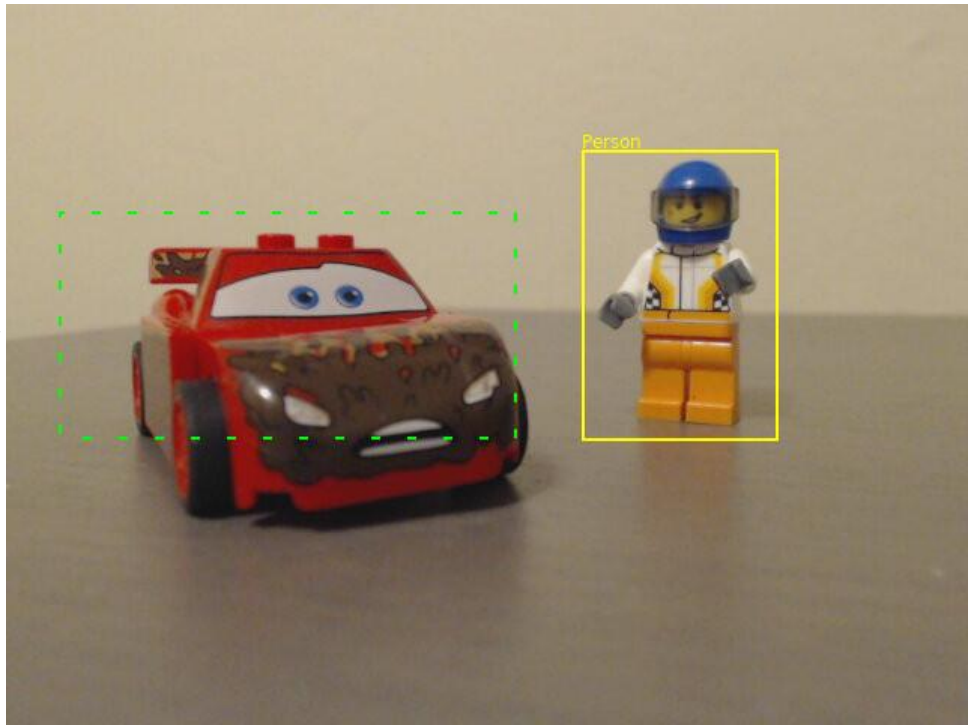
Effectiveness

Frame from 1s ago



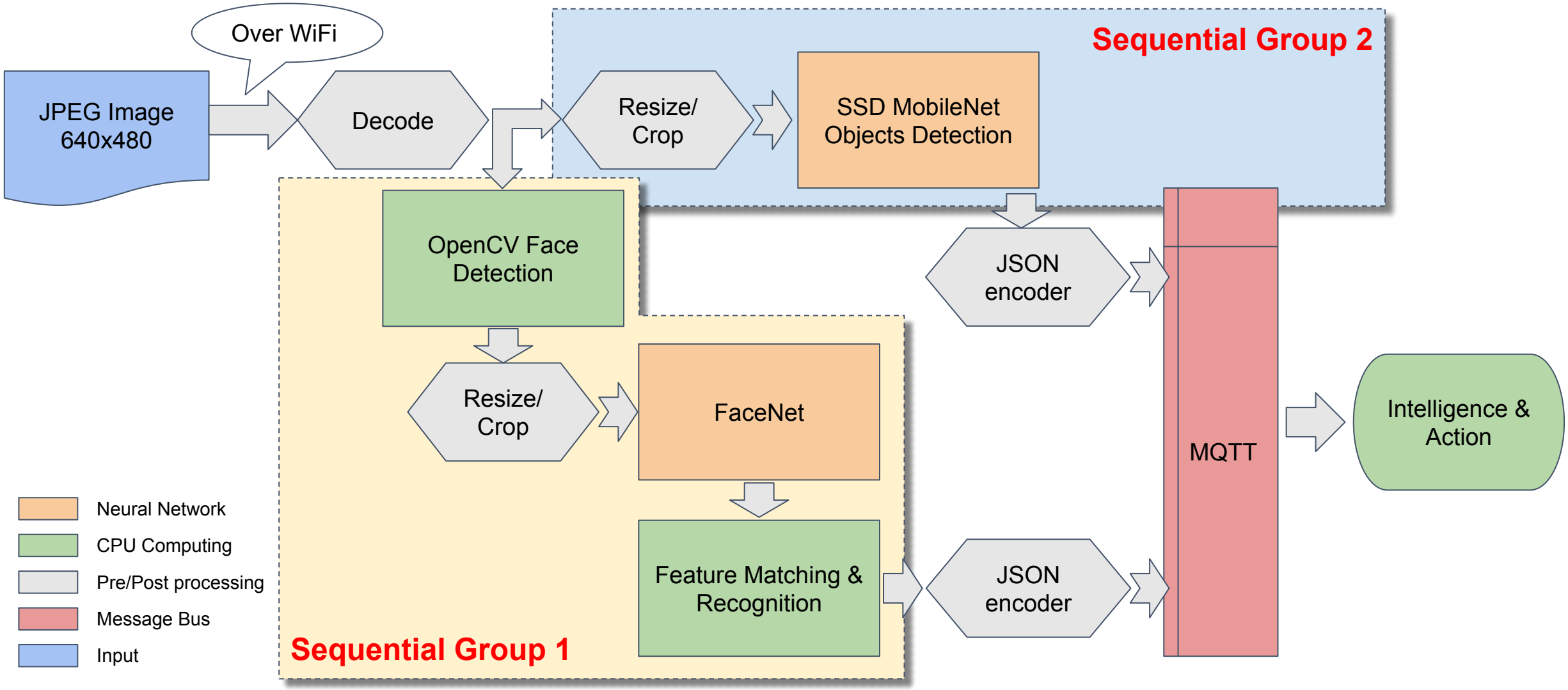
1s delay

Frame from Now!



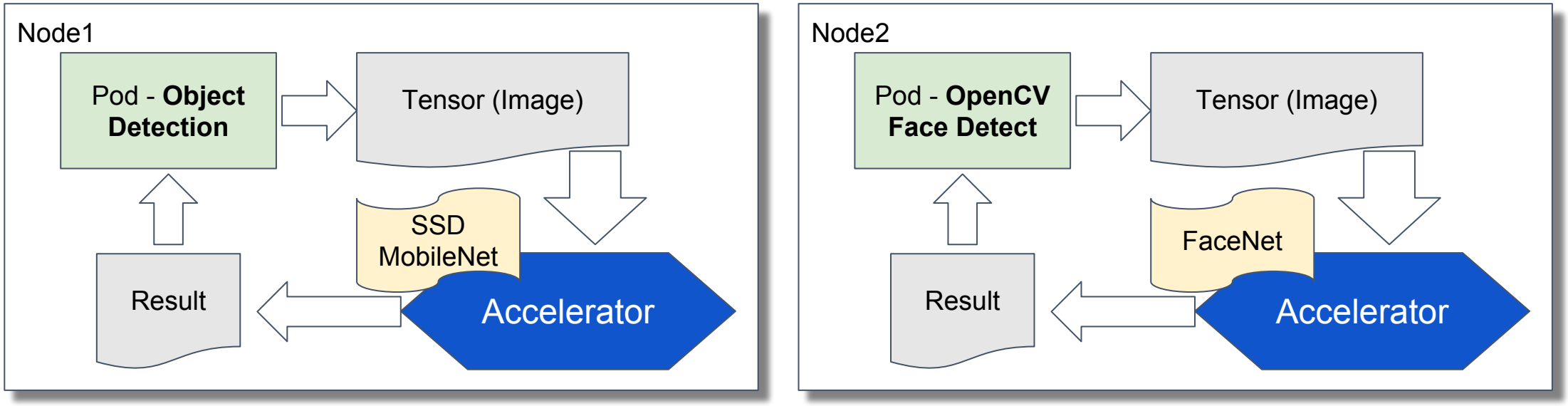
- The label ***Person*** is still ***effective*** as there's no change in the area;
- The label ***Car*** is no longer effective as the area has changed a lot.

Parallelize Pipeline



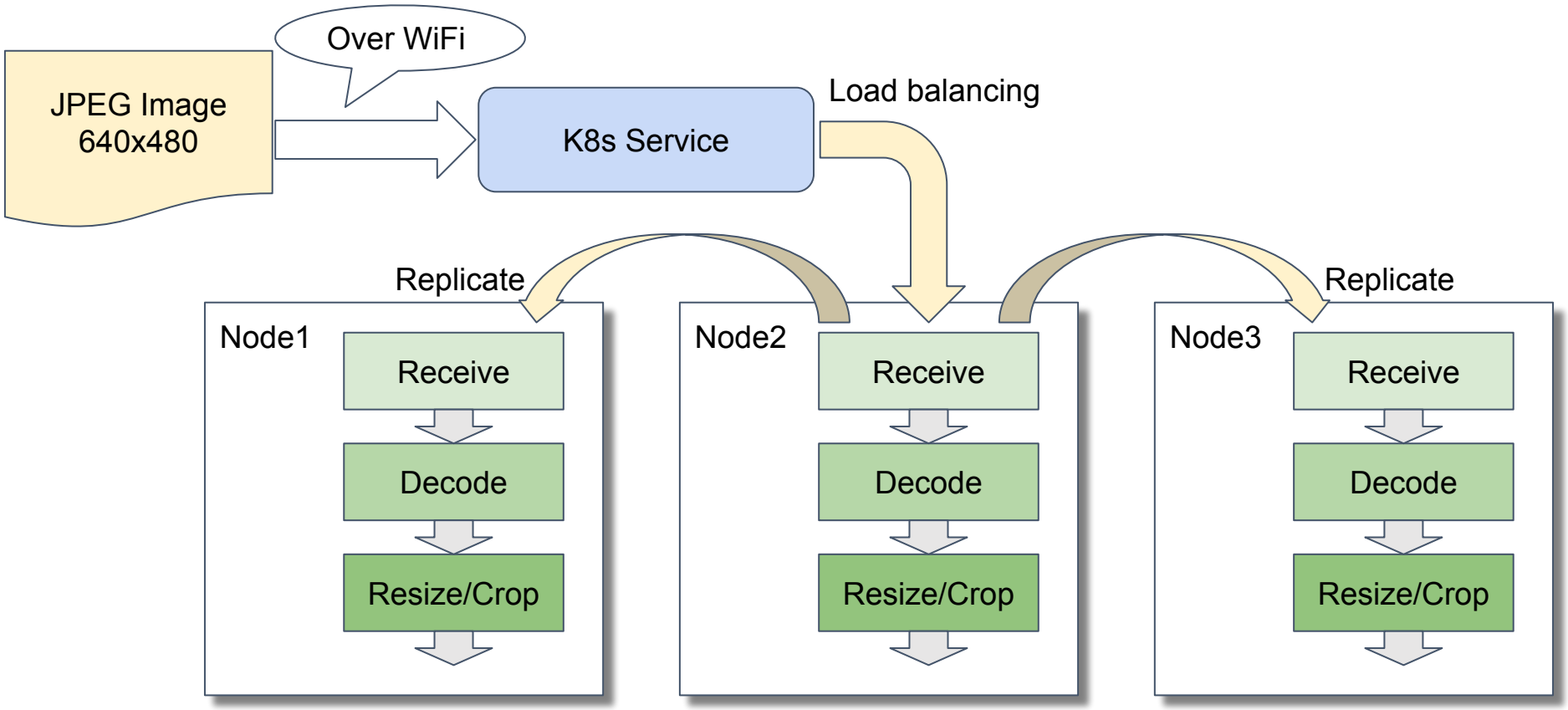
Colocate Compute & Data

- Label nodes by models
- Colocate Pods with related models using node affinity



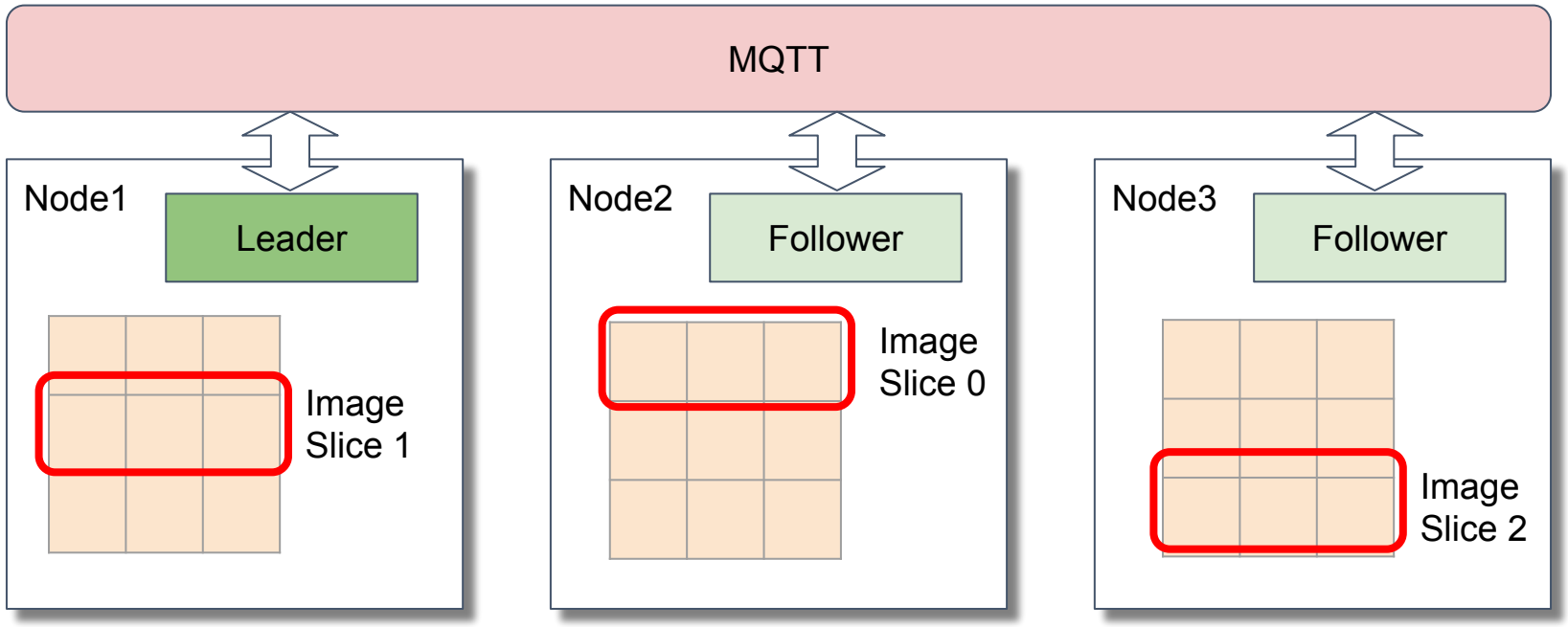
Duplicate Lightweight Tasks

- Replicate compressed image (JPEG) to all nodes
- Duplicate lightweight image pre-processing tasks



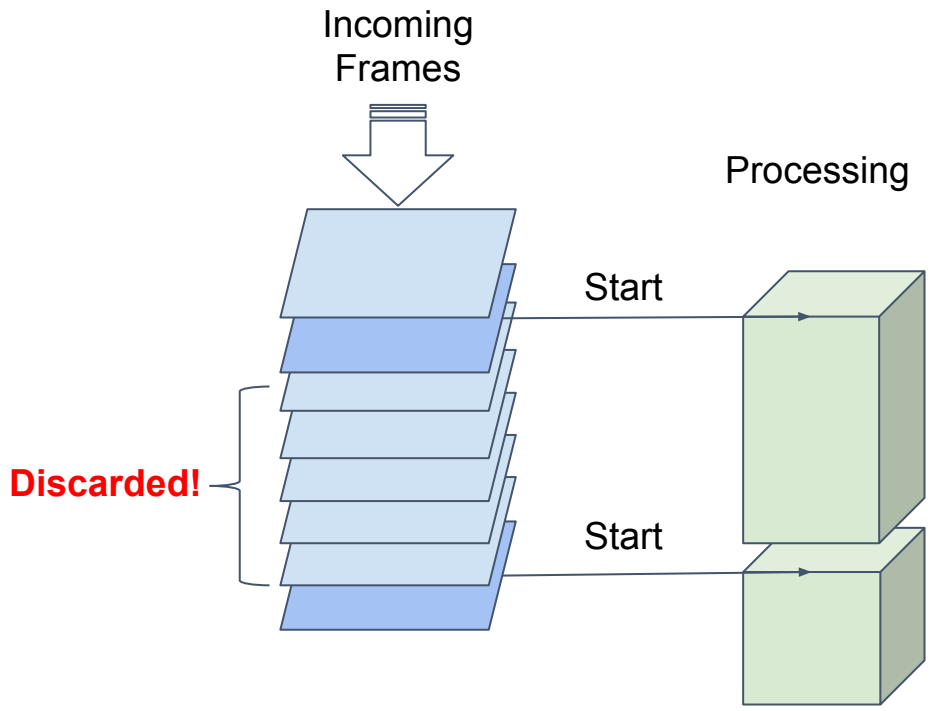
Map/Reduce is Possible

- Prerequisite: image replicated to all nodes
- Coordination: lead-election and MQTT



Survive from poor WiFi

- Discard dated frames
  - Associate a sequence from source (robot)
  - Only process the most recent frame (skip others if the previous pipeline didn't complete in time)



Survive from poor WiFi

- TCP vs UDP
  - TCP accumulates delay when jammed, unable to skip
  - UDP is unreliable, good for skipping frames
- H264 vs JPEG
  - H264 takes low bandwidth (< 20KB), but less loss tolerable
  - JPEG takes high bandwidth (20 - 64KB), high loss tolerable

Challenge - On ARM

KubeCon | CloudNativeCon

North America 2018

Kubernetes on ARM

Core Affinity

- Special problem on Big-Little architecture
  - CPU-intensive tasks scheduled on Big cores
  - Accelerator offload tasks scheduled on Little cores
  - Flipping over degrades performance
- No support from Kubernetes
  - Use core affinity inside the container
  - Maybe sub-optimal w.r.t the cluster-level scheduling performance

Summary

Recapture and Takeaways

Summary

- Accelerator support
  - Privileged container
- Shared memory across Pods
- Affinity with Data Locality
- Replication across nodes
  - Input data (small size)
  - Pre-processing logic
  - Parallelism with data replicas
- Core affinity (heterogeneous architecture)

Yisui Hu, Google, 2018