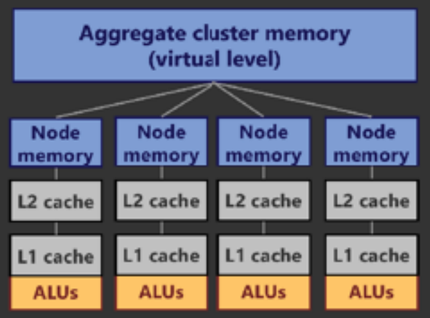
# Mapping visual computing applications efficiently to hardware

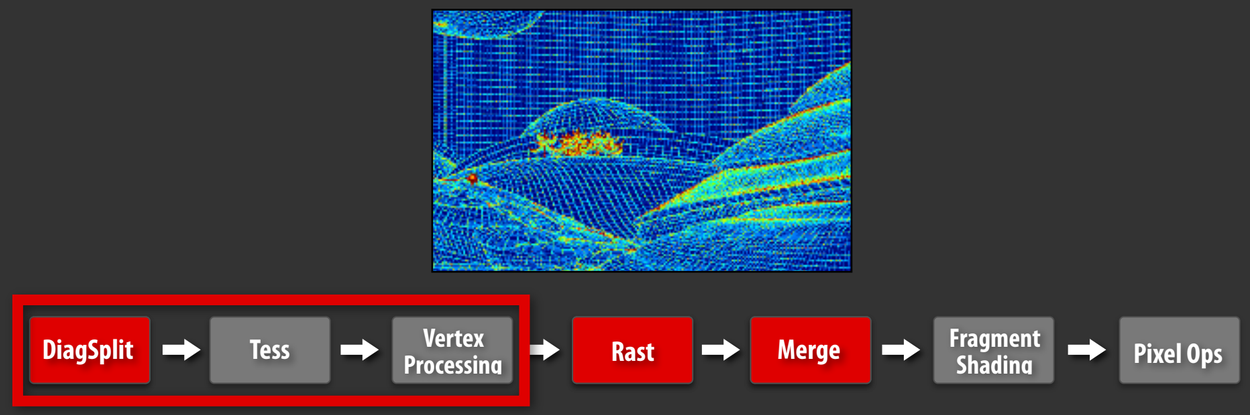
Kayvon Fatahalian

# My background

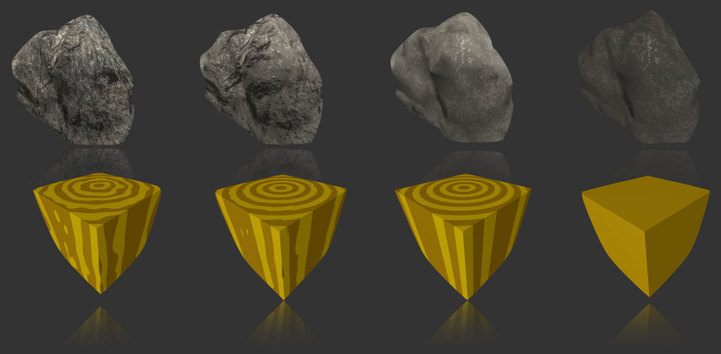
## Systems for graphics, image processing, and parallel computing



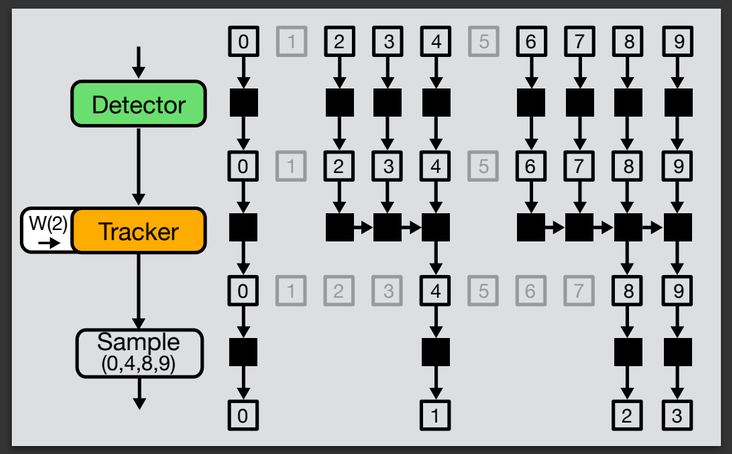

Sequoia (2006)
"Programming the memory hierarchy" for supercomputing applications




Real-time Micropolygon Rendering Pipeline (2011)



Compilers that automatically approximate graphics programs (2015)



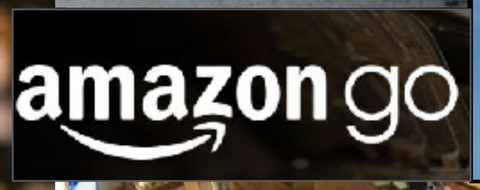

Scanner (2018)
Efficient video analysis at scale


DISCOVER CARD
TDK

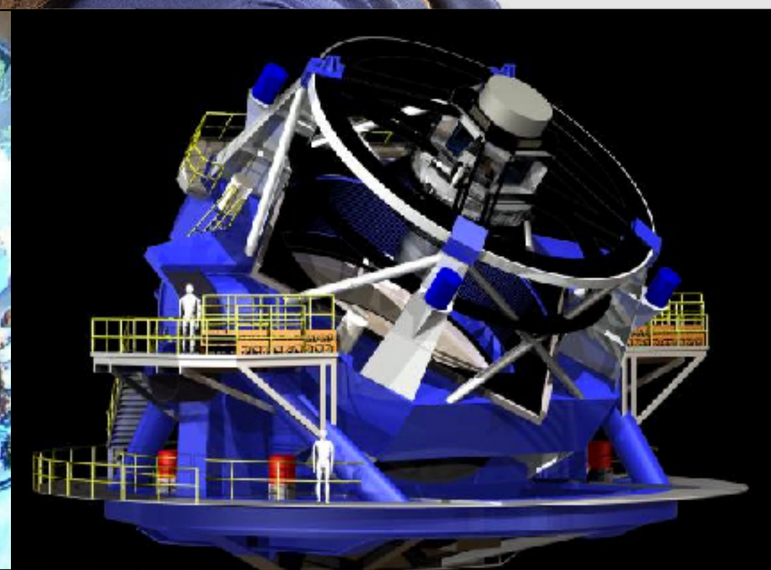

MEET THE PRESS
CARSON CONTROVERSY
PAST CLAIMS UNDER THE MICROSCOPE
MEET THE PRESS


UBER
ADVANCED TECHNOLOGIES CENTER

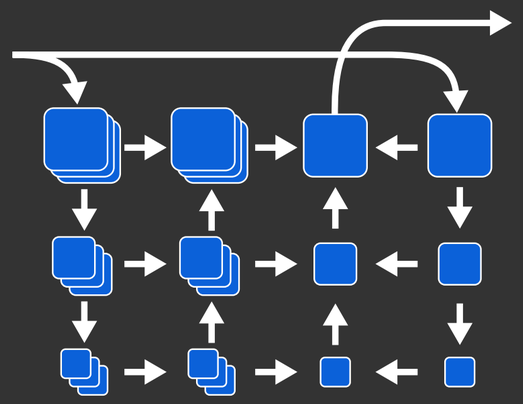

WAYMO


amazon go


her Movement

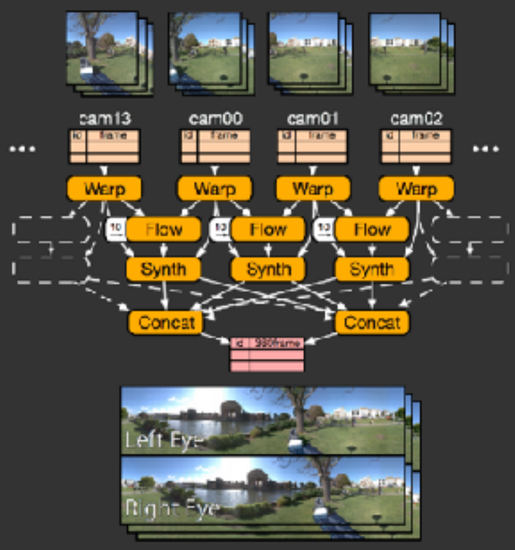


# Enabling new applications and ~ 100x Lower Cost:



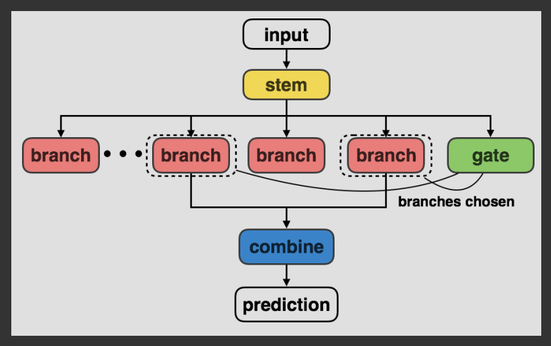
**Use every op available on modern chips**

Compilation techniques for scheduling data-parallel image processing and deep learning computations to CPUs, GPUs, and custom hardware like FGPAs/CGRAs




**Efficiently using accelerated computing across large numbers of machines**

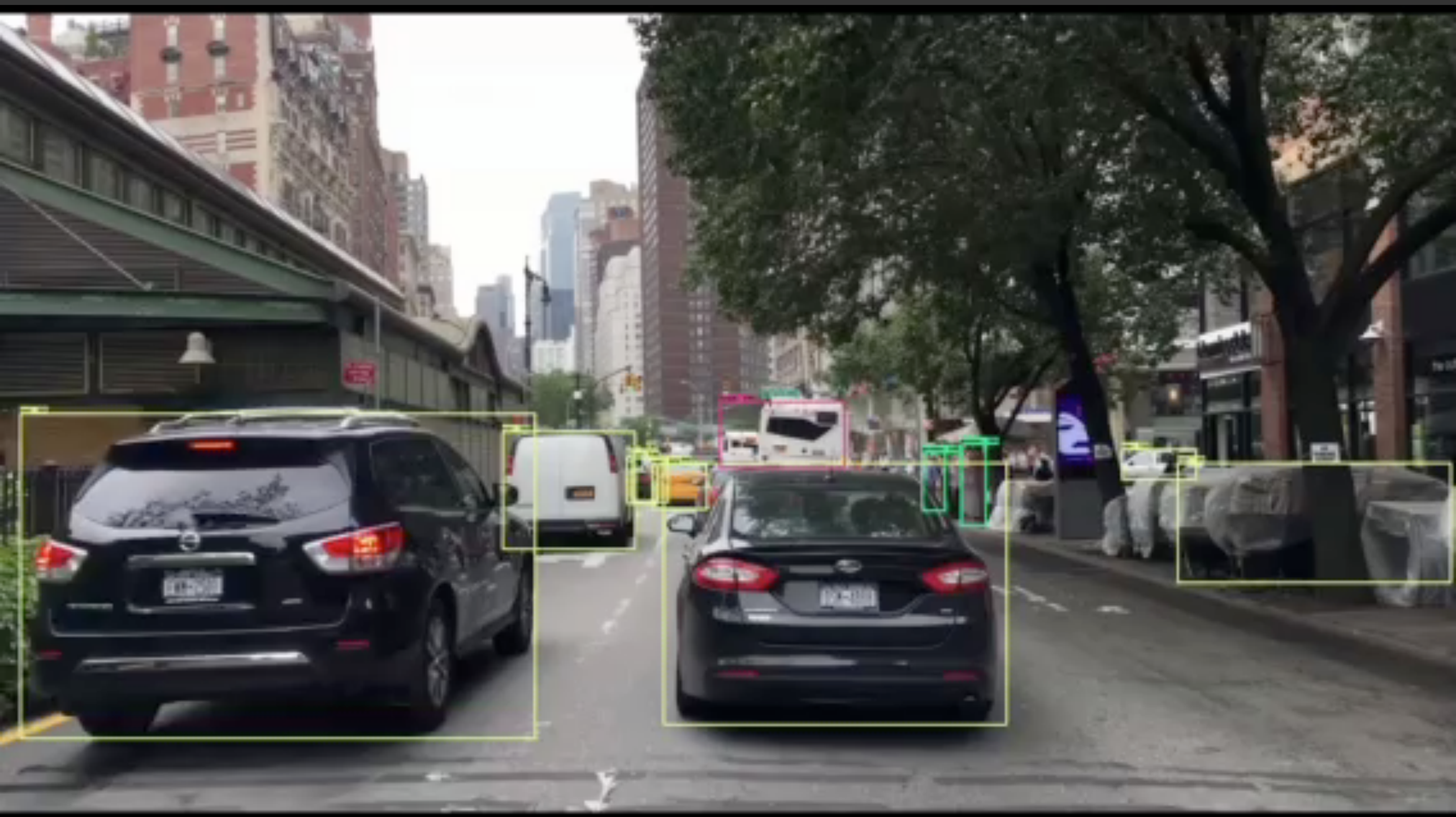
Distributed system support specialized for video processing at scale




**Maximize accuracy/cost: via work efficient video analysis algorithms**

Specializing processing techniques to video streams, conditional execution, etc…

# Compute at datacenter scale: autonomous vehicle datasets

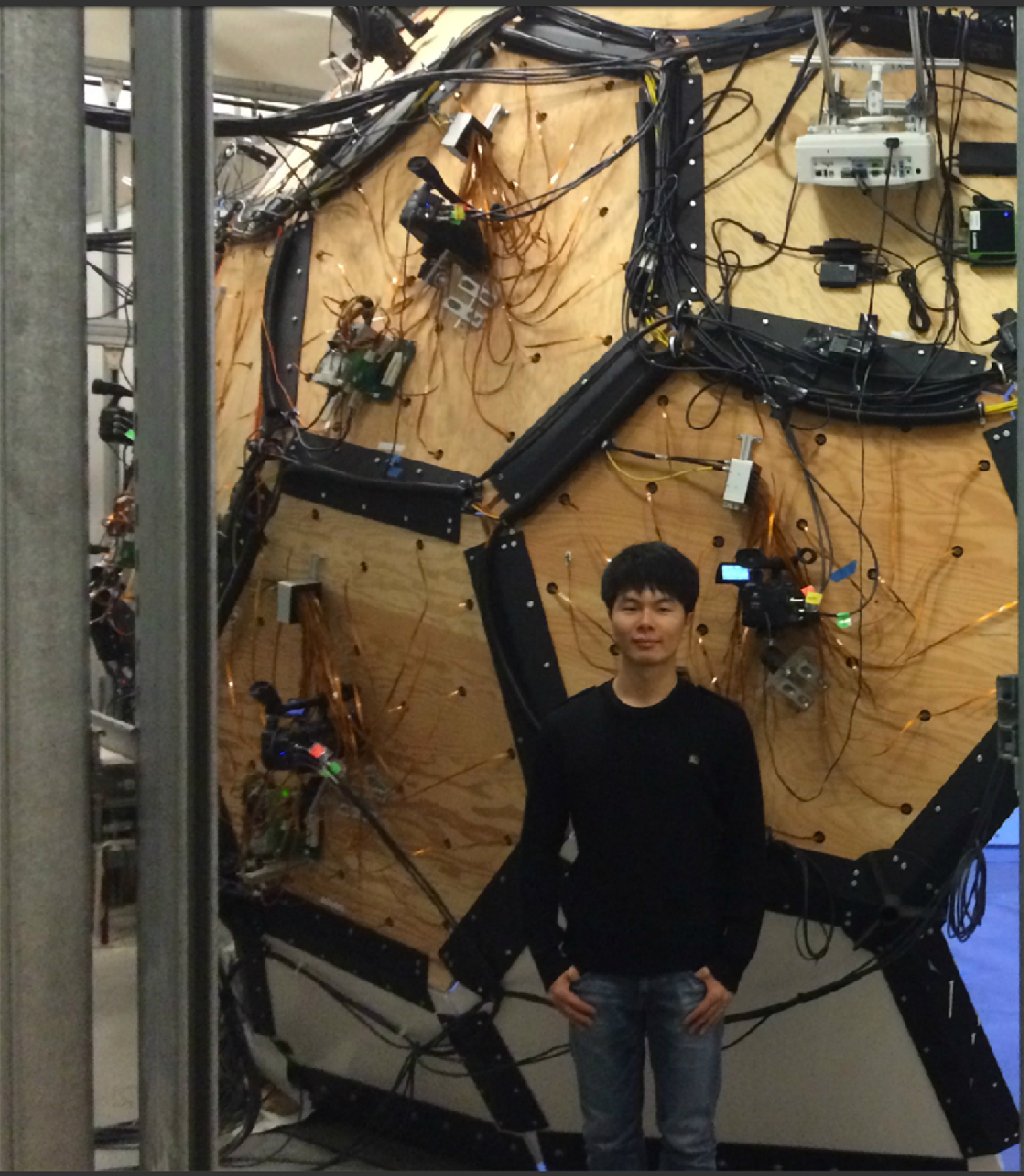


# Visual data mining of American TV News

- 24/7 CNN, FOX, MSNBC from 2015-2017
  - 70,000 hours of footage
  - 640x480 or 640x360 (30 or 60 fps)
  - 20 TB of video
  - 12 billion frames
  - 6 quadrillion pixels

# Scaling to many streams

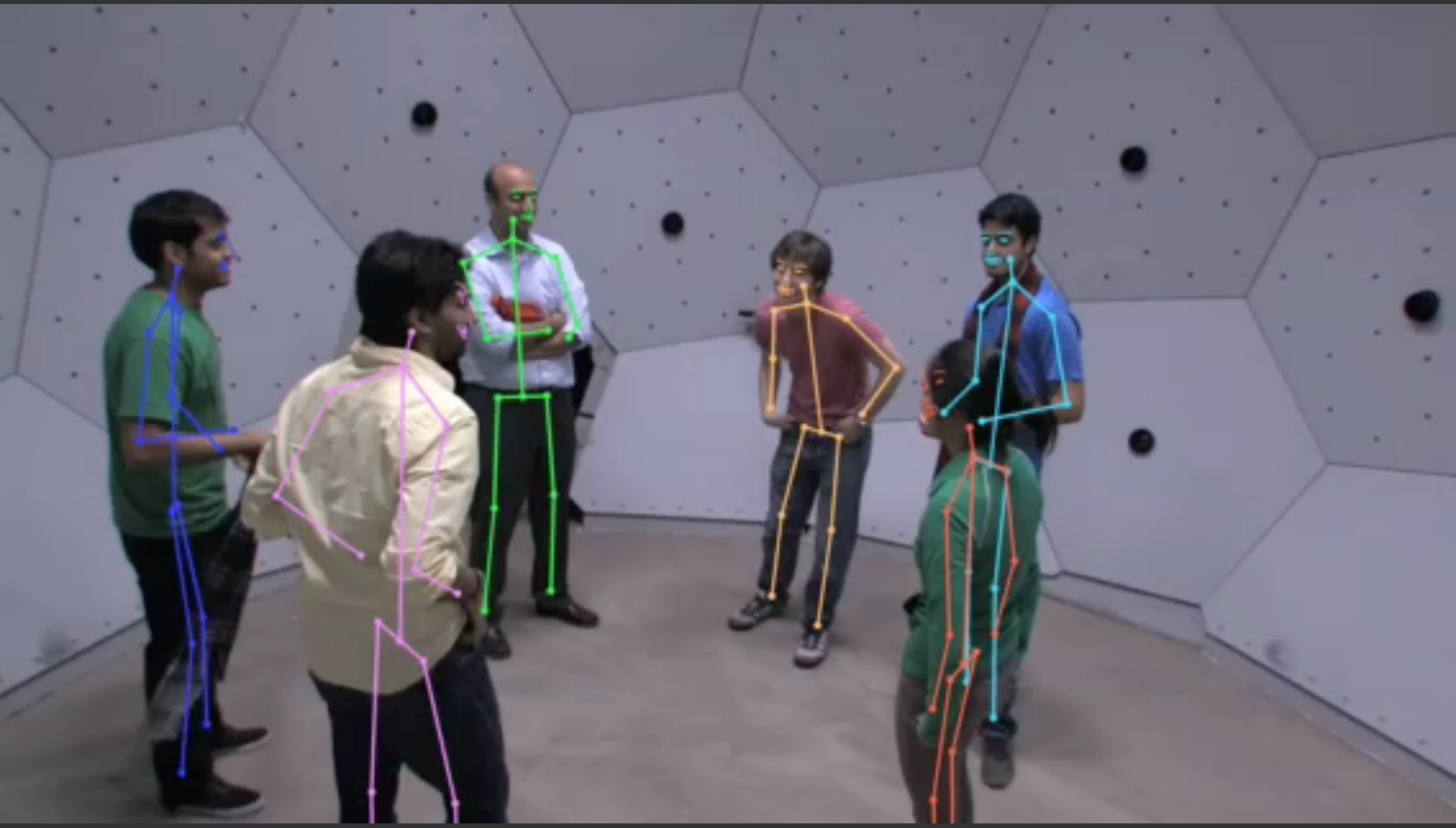
[Joo 2015]



**Panoptic Studio capture**
**480 video cameras (640 x 480 @ 24fps)**

**147 MPixel video sensor**
**15 GB / minute**

**40-second sequence**
**3D pose reconstruction time: hand-coded solution — 7 hours on 4-Titan Xp's [Cao 2016]**



[Courtesy Yaser Sheikh, Tomas Simon, Hanbyul Joo] [Joo et al. 2015]

# Surround 360 VR video

14 cameras, 2048 x 2048@ 30 FPS

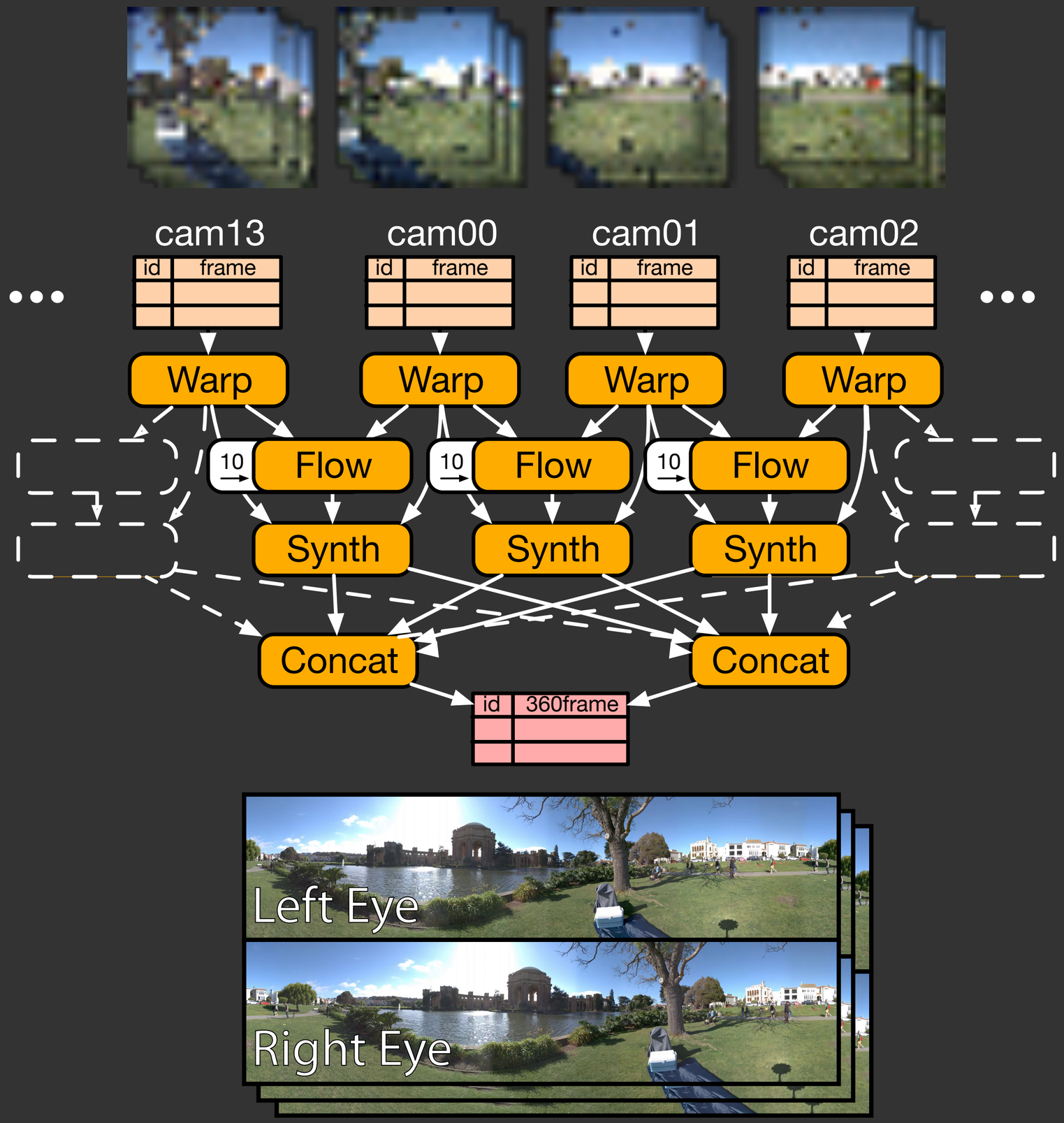
25k frames / minute

300 GB (raw) / minute

8K x 8K stereo panorama output

= 12.5 secs per frame on 32-core CPU

# Surround 360 video data flow




- 44 logical stages
- Stream elements passed between operations are 2K images

# Scaling down for massive efficiency

**All day video analysis (localization, semantic interpretation) + image synthesis for display**

**Battery-powered video camera (2 years of life)**

# Domain characteristics

- More compute efficiency —> new applications

- Key kernels written in domain specific APIs and DSLs
  - e.g., Halide, TensorFlow/MXNet, etc.

- Rapid algorithm innovation
  - Consider algorithmic improvement in image classification

2014 →2017 ~ 25x improvement in cost at similar accuracy

| | ImageNet Top-1 Accuracy | Num Params | Cost/image (MADDs) | |
|---|---|---|---|---|
| VGG-16 | 71.5% | 138M | 15B | [2014] |
| GoogleNet | 70% | 6.8M | 1.5B | [2015] |
| ResNet-18 | 73% | 11.7M | 1.8B | [2016] |
| MobileNet-224 | 70.5% | 4.2M | 0.6B | [2017] |

# Efficiently mapping to HW

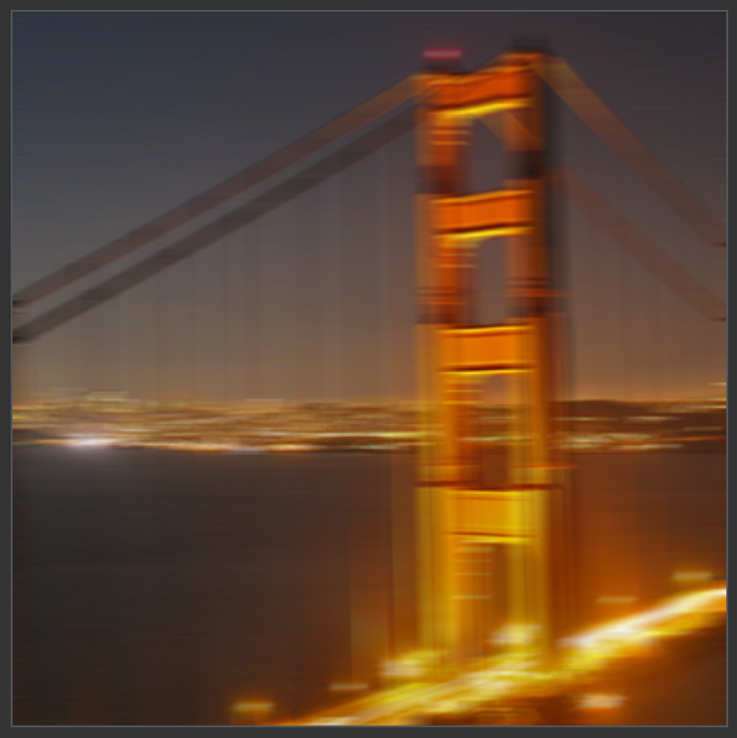
# Halide DSL

[Ragan-Kelley, Adams et al. 2012]

**Raised level of abstraction for developing high-performance image processing algorithms on dense n-D arrays**

```
blurx(x,y) = (in(x-1,y)    + in(x,y)    + in(x+1,y))    / 3;
out(x,y)   = (blurx(x,y-1) + blurx(x,y) + blurx(x,y+1)) / 3;
```

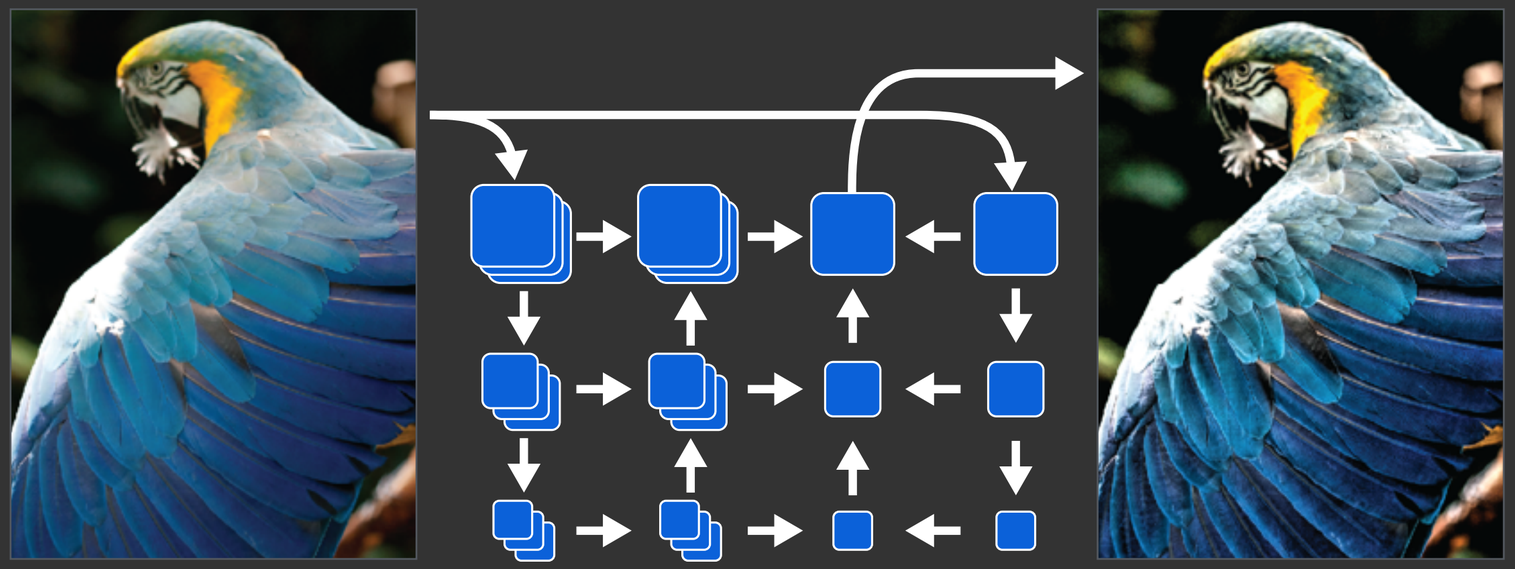

in



blurx

out

# Modern camera applications perform complex processing tasks expressed as dataflow graphs



100 stages



Local Laplacian filter
[Paris 2010, Aubry 2011]


**Google Nexus HDR+ mode:**
**over 2000 stages!**

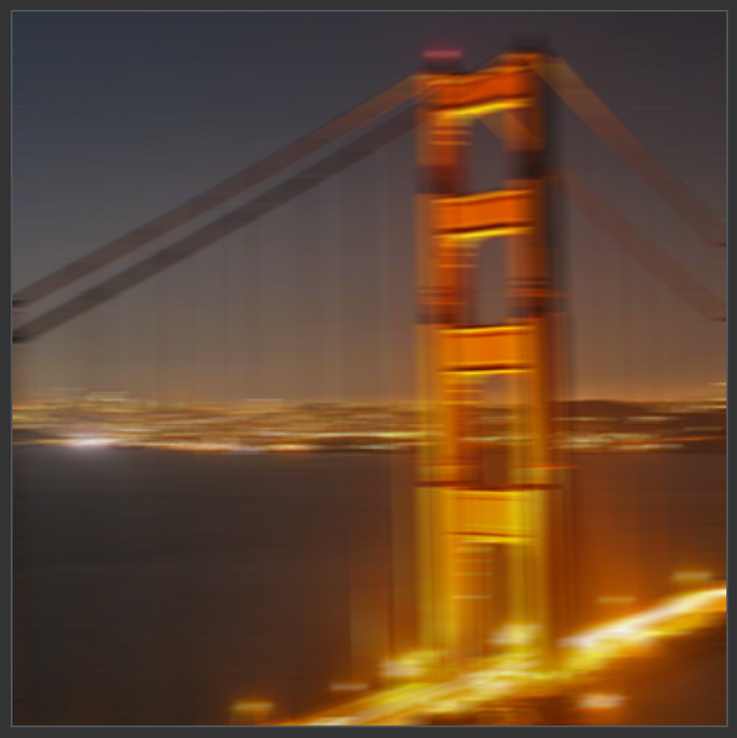
# Halide DSL

[Ragan-Kelley, Adams et al. 2012]

**Raised level of abstraction for developing high-performance image processing algorithms on dense n-D arrays**

```
blurx(x,y) = (in(x-1,y)    + in(x,y)    + in(x+1,y))    / 3;
out(x,y)   = (blurx(x,y-1) + blurx(x,y) + blurx(x,y+1)) / 3;
```


in



blurx

out

# Halide: image processing DSL

[Ragan-Kelley, Adams et al. 2012]

## Raised level of abstraction for developing high-performance image processing algorithms

### Functional pipeline description:

```
blurx(x,y) = (in(x-1,y)    + in(x,y)    + in(x+1,y))    / 3;
out(x,y)   = (blurx(x,y-1) + blurx(x,y) + blurx(x,y+1)) / 3;
```

### Schedule: DSL for mapping pipeline stages to a parallel machine

```
output.tile(x, y, xi, yi, 256,
32);

output.vectorize(xi, 8);
output.parallelize(y);

blurx.compute_at(xi);
blurx.vectorize(x, 8);
```

compute output in tiled order

vectorize innermost loop

parallelize loop across cores

loop fusion

vectorize innermost loop

# High performance conv (hand-written)

```
void fast_blur(const Image &in, Image &blurred) {
 __m128i one_third = _mm_set1_epi16(21846);
 #pragma omp parallel for
 for (int yTile = 0; yTile < in.height(); yTile += 32) {
  __m128i a, b, c, sum, avg;
  __m128i tmp[(256/8)*(32+2)];
  for (int xTile = 0; xTile < in.width(); xTile += 256) {
   __m128i *tmpPtr = tmp;
   for (int y = -1; y < 32+1; y++) {
    const uint16_t *inPtr = &(in(xTile, yTile+y));
    for (int x = 0; x < 256; x += 8) {
     a = _mm_loadu_si128((__m128i*)(inPtr-1));
     b = _mm_loadu_si128((__m128i*)(inPtr+1));
     c = _mm_load_si128((__m128i*)(inPtr));
     sum = _mm_add_epi16(_mm_add_epi16(a, b), c);
     avg = _mm_mulhi_epi16(sum, one_third);
     _mm_store_si128(tmpPtr++, avg);
     inPtr += 8;
   }}
   tmpPtr = tmp;
   for (int y = 0; y < 32; y++) {
    __m128i *outPtr = (__m128i *)(&(blurred(xTile, yTile+y)));
    for (int x = 0; x < 256; x += 8) {
     a = _mm_load_si128(tmpPtr+(2*256)/8);
     b = _mm_load_si128(tmpPtr+256/8);
     c = _mm_load_si128(tmpPtr++);
     sum = _mm_add_epi16(_mm_add_epi16(a, b), c);
     avg = _mm_mulhi_epi16(sum, one_third);
     _mm_store_si128(outPtr++, avg);
}}}}}
```

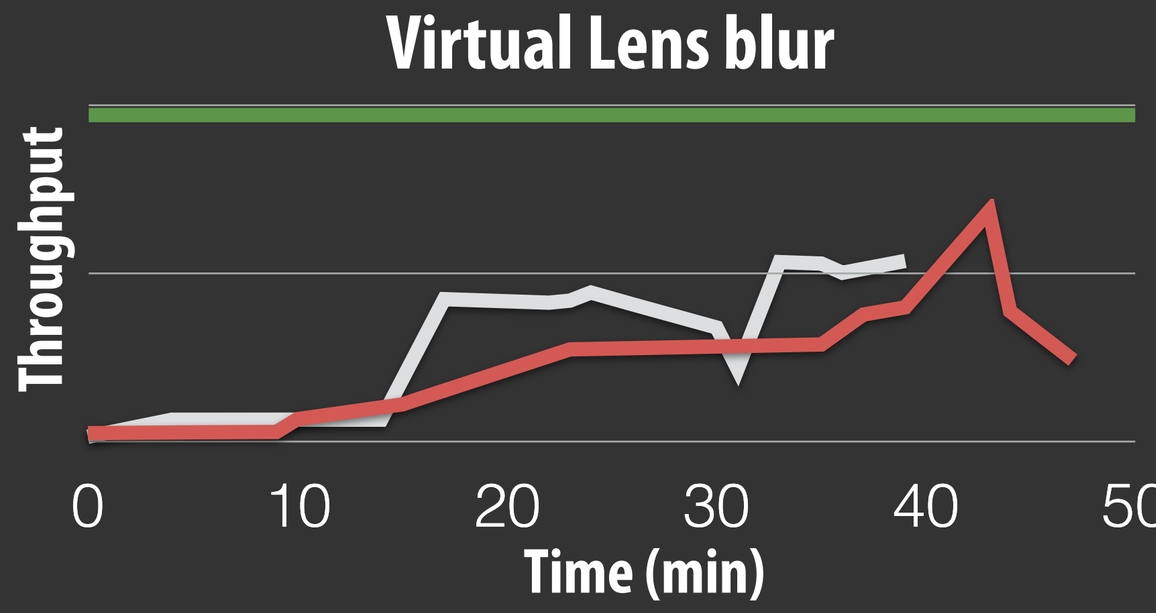
Multi-core execution (partition image vertically)

Modified iteration order: 256x32 tiled iteration (to maximize cache hit rate)

use of SIMD vector intrinsics

two passes fused into one: `tmp` data read from cache

# Autoscheduling halide programs





# Mapping data-parallel primitives to hardware

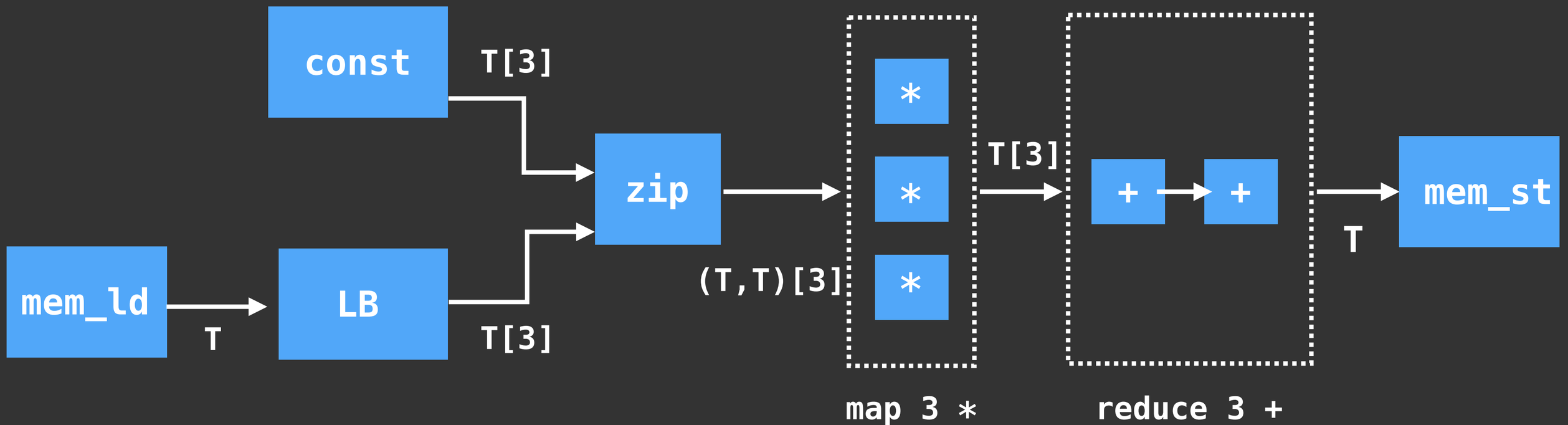
- `map p n f` :: `S[p]{n} -> T[p]{n}`
- `reduce p n f` :: `T[p]{n} -> T{n}`
- `linebuffer p w n` :: `T[p]{n} -> T[p][w]{n-1}`
- `streamify k` :: `T[k] -> T{k}`
- `arrayify k` :: `T{k} -> T[k]`
- `upsample k` :: `T -> T[k]`
- `downsample k` :: `T[k] -> T`
- `mem_ld` :: `_ -> T`
- `mem_st` :: `T -> _`
- `zip` :: `S x T -> (S,T)`
- + Possible extensions for wider app domain: `reduceByKey, filter, etc.`

**Sufficiently high level to realize data-parallel abstraction**
**Sufficiently low-level for transparent mapping to HW modules**

# Scheduling data parallel programs in space and time

```
mem_ld 1 .
reduce 3 (+) .
map 3 (*) .
zip
(
  lb 1 3 . mem_st 1,
  const [1, 1, 1]
)
```

**Throughput: 1 pixel / clock**



# Aetherling: Layering abstractions

Halide algorithm +
HW schedule

e.g., Spatial, RIPL,
Weld, CλaSH, other DSLs

Halide Compiler

Other high level
language compilers

Common spacetime scheduling passes /
transformation rules

Library of (higher order) module generators
(map, reduce, fold, linebuffer,
mem, ready/valid, etc.)

Data Parallel IR for HW
(model of space - time resources )

CoreIR

Aetherling is:
Collection of data parallel primitives
Transformation rules for trading off space and time resources using those primitives
(Potentially) validation of correctness of those transformations

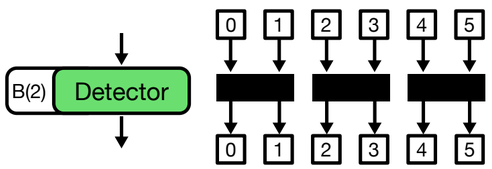
# Abstractions / scheduling for video?

# Scanner: video-centric data-parallel operations

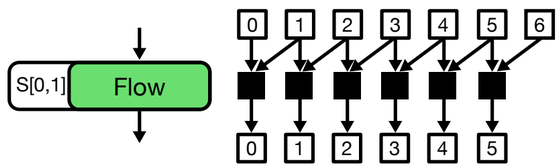
**map**
**stencil**
**serial fold, bounded_parallel_fold**

**sample (stride/gather)**
**space**
**slice**
**unslice**

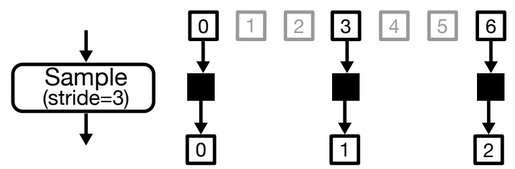
(a) Map (with element batching)



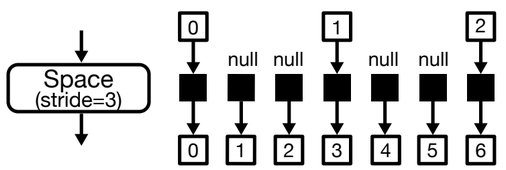

(b) Stencil



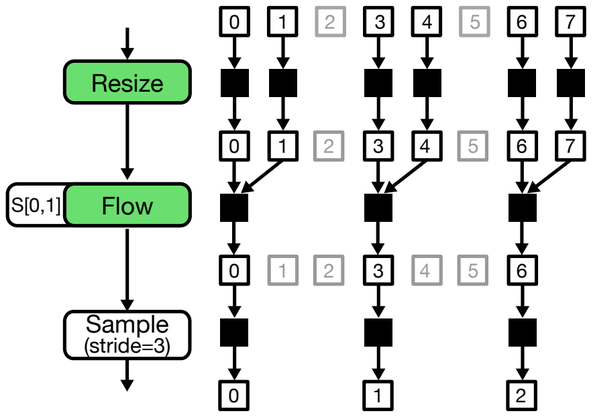

(c) Strided Sampling



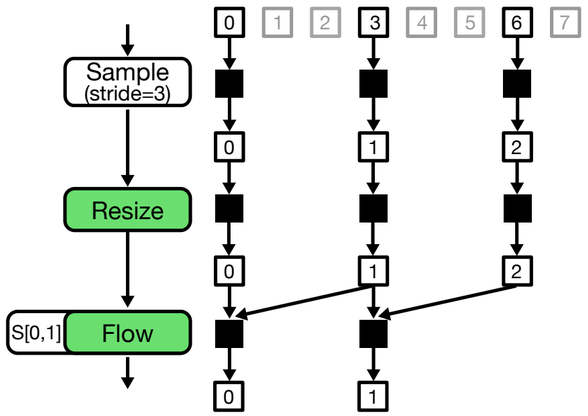

(d) Strided Spacing



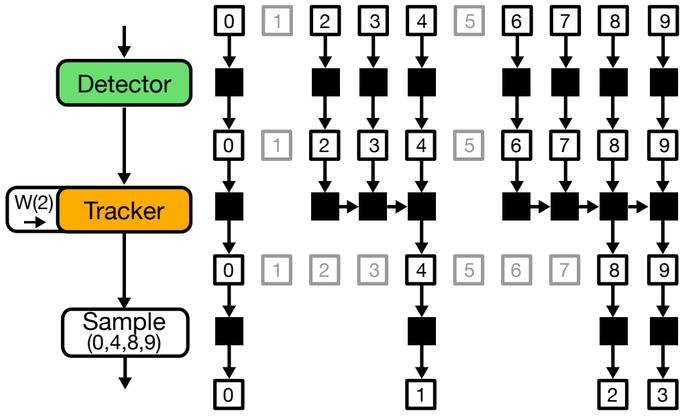

(e) Dense Strided Stencil



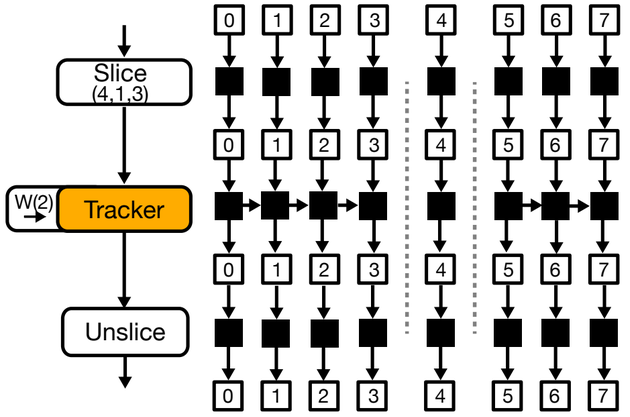

(f) Sparse Strided Stencil



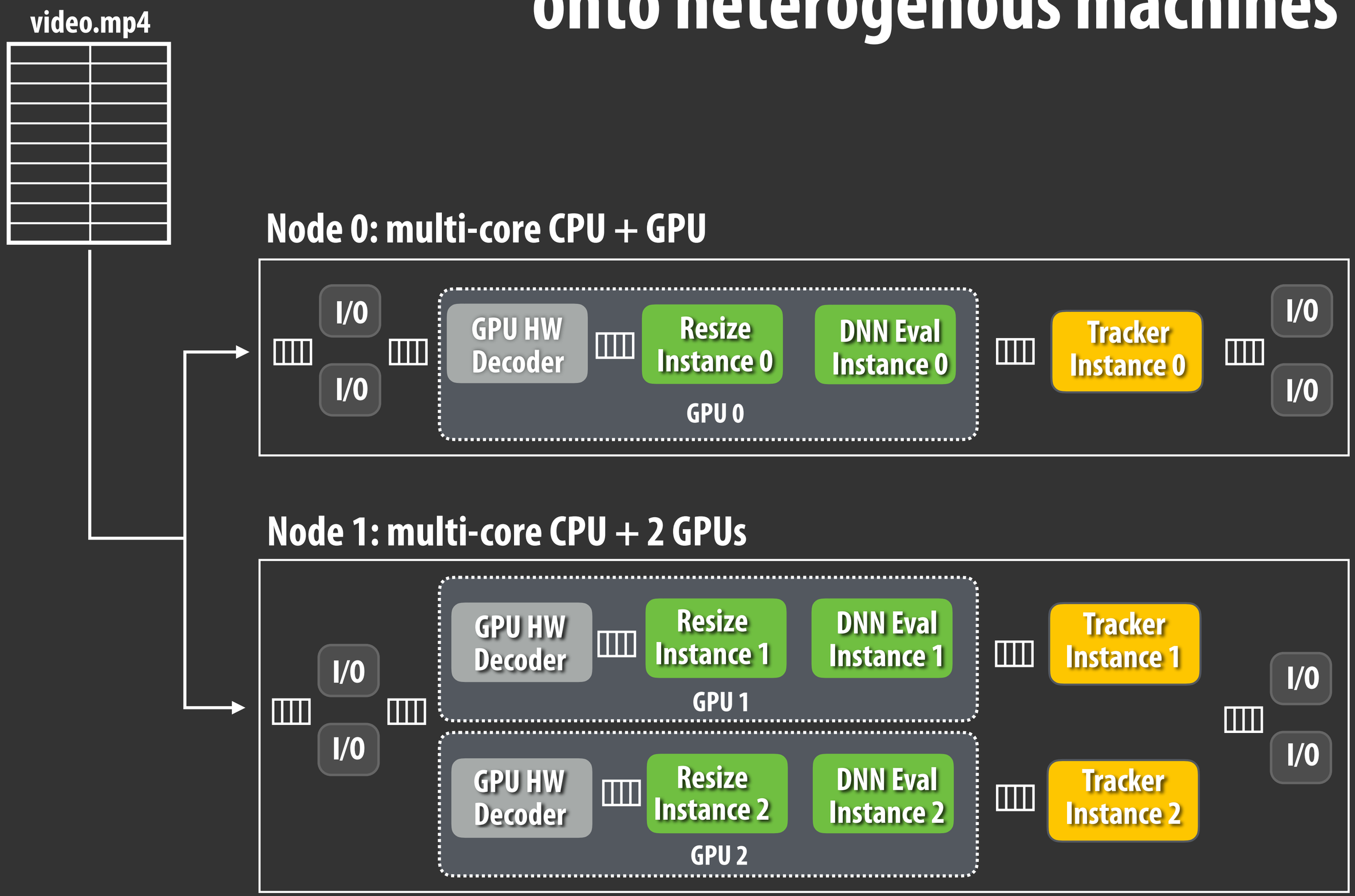

(g) Bounded State (Warmup=2)



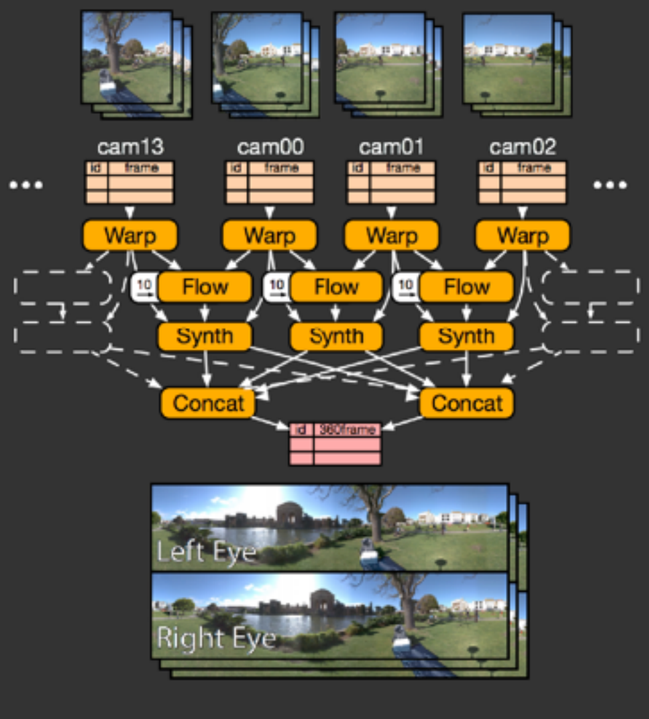

(h) Bounded State After Slice



# Scanner: distributing the computation graph onto heterogenous machines



# Surround360 VR Video




14 2K cameras
25k frames / minute
300 GB (raw) / minute

On a 32 core CPU,
2.7 hours / minute of video

# 3D Pose Estimation

Joo. et al

480 640x480 cameras
864k frames / minute
15 GB / minute

On one Titan X GPU,
15.6 hours / minute of video

# Video Data Mining


TV News from past 3 years,
70,000 hours of video,
(12 billion frames)
20 TB total

On a 500 machine cluster,
several days

# Addressing efficiency issues

**We have ability to execute video processing code efficiently on many compute-accelerated machines… I just can't pay for all the machines.**

**($3000 / grad student idea)**

**New image analysis algorithms are being developed by the CV community every day… But I can't run them on low-power devices**

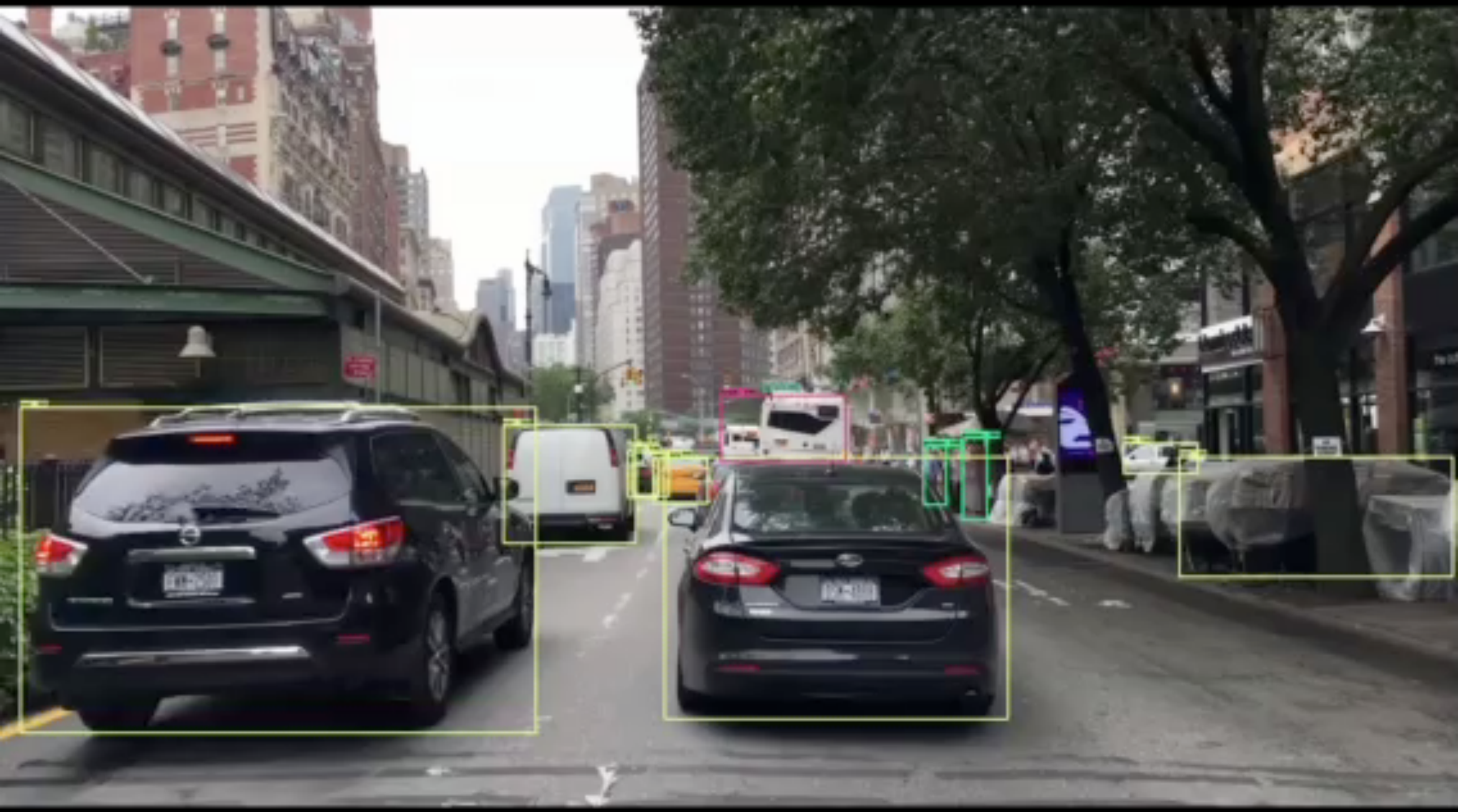
# Specializing models for efficiency

- If you are an AI researcher…
  - Aim to design a general model that succeeds in a wide range of contexts (train on large datasets, regularize, etc.)

- But interpreting contents of any one image requires only a small fraction of total knowledge

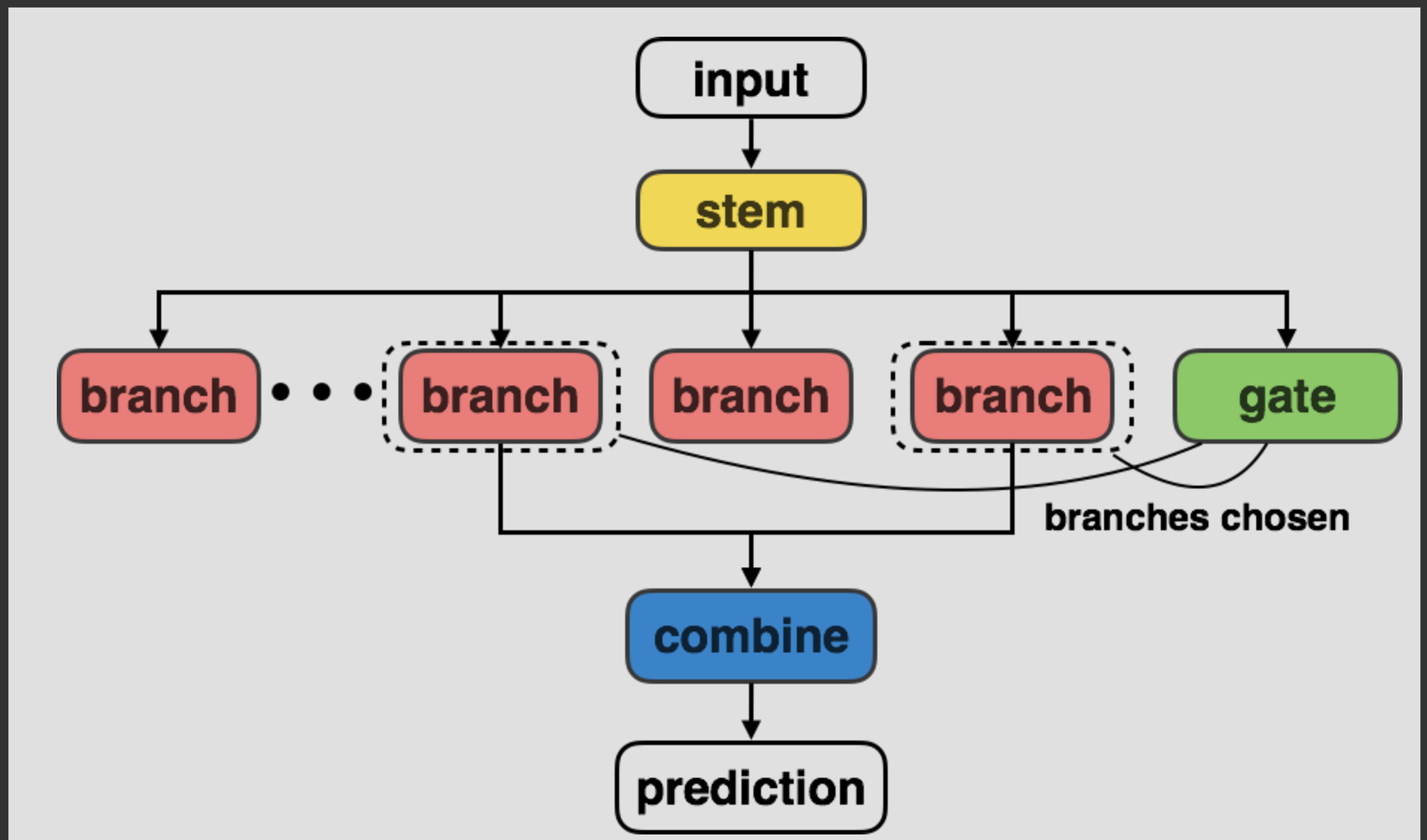
- And many cameras see a very specialized set of the world's images
  - Scene contains limited types of objects
  - Scene observed from similar viewpoint


2016-11-16 16:08:58
BUS LANE
Nixcon
CAM01



# Conditional DNNs that specialize to types of images

- Idea: create large capacity models ("high accuracy") by composing subnetworks specialized for different visual characteristics
- But conditionally evaluate only a ***small fraction*** of the DNN for a given input image instance (high efficiency)

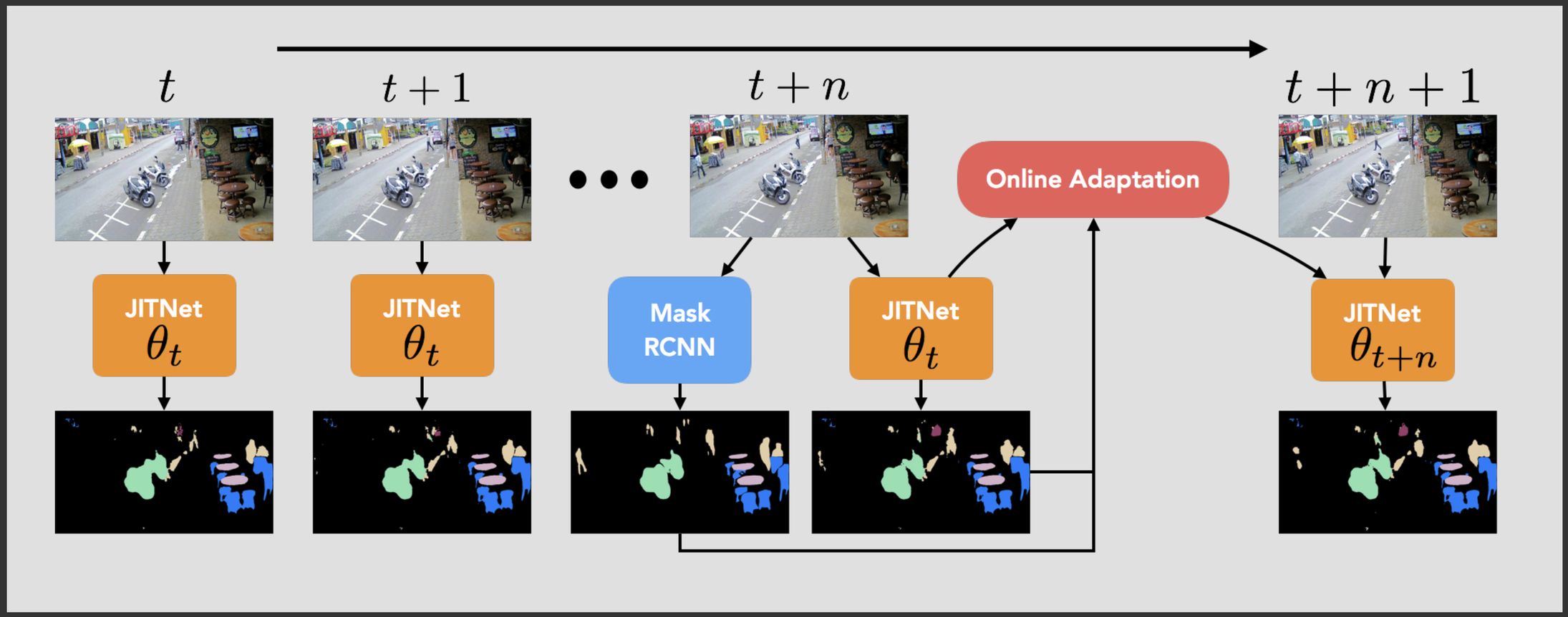



[Mullapudi, CVPR 18]

# Specializing to contents of video at a specific instance in time (online adaptation)

- Train compact model that is cheap to evaluate, ***but is accurate for only the next few seconds***…

- Continuously retrain model based on new images the camera sees…



**Continuously retrained DNN**
**20 ms / frame**

**High-quality object segmentation network**
**250 ms / frame**

# Summary

- **High priority on creating high-level programming abstractions, *that retain performance transparency***

- **High cost of pixel processing encourages both *hardware specialization AND algorithmic innovation***
  - **Sampling (across frames, within a frame)**
  - **Conditional execution**
  - **Continuous adaptation: specialize algorithm to scene, over and over again…**

# Beyond pixel processing in AHA

- General data-analytics
- Text processing/search
- Audio / speech
- IMU / motion processing
- Radio
- Networking
- etc…

# Thank you

**Today's work performed in collaboration with:**
**Will Crichton, David Durst, Sahaj Garg, Jason Hong, Alex Poms, Ravi Mullapudi, Haotian Zhang, Keyi Zhang, Andrew Adams, Dillon Sharlot, Jonathan Ragan-Kelley**
**Maneesh Agrawala, Deva Ramanan, Pat Hanrahan**