Lecture 13:

Processing Video at Cloud Scale

Visual Computing Systems
Stanford CS348K, Fall 2018

More on specialization to input video

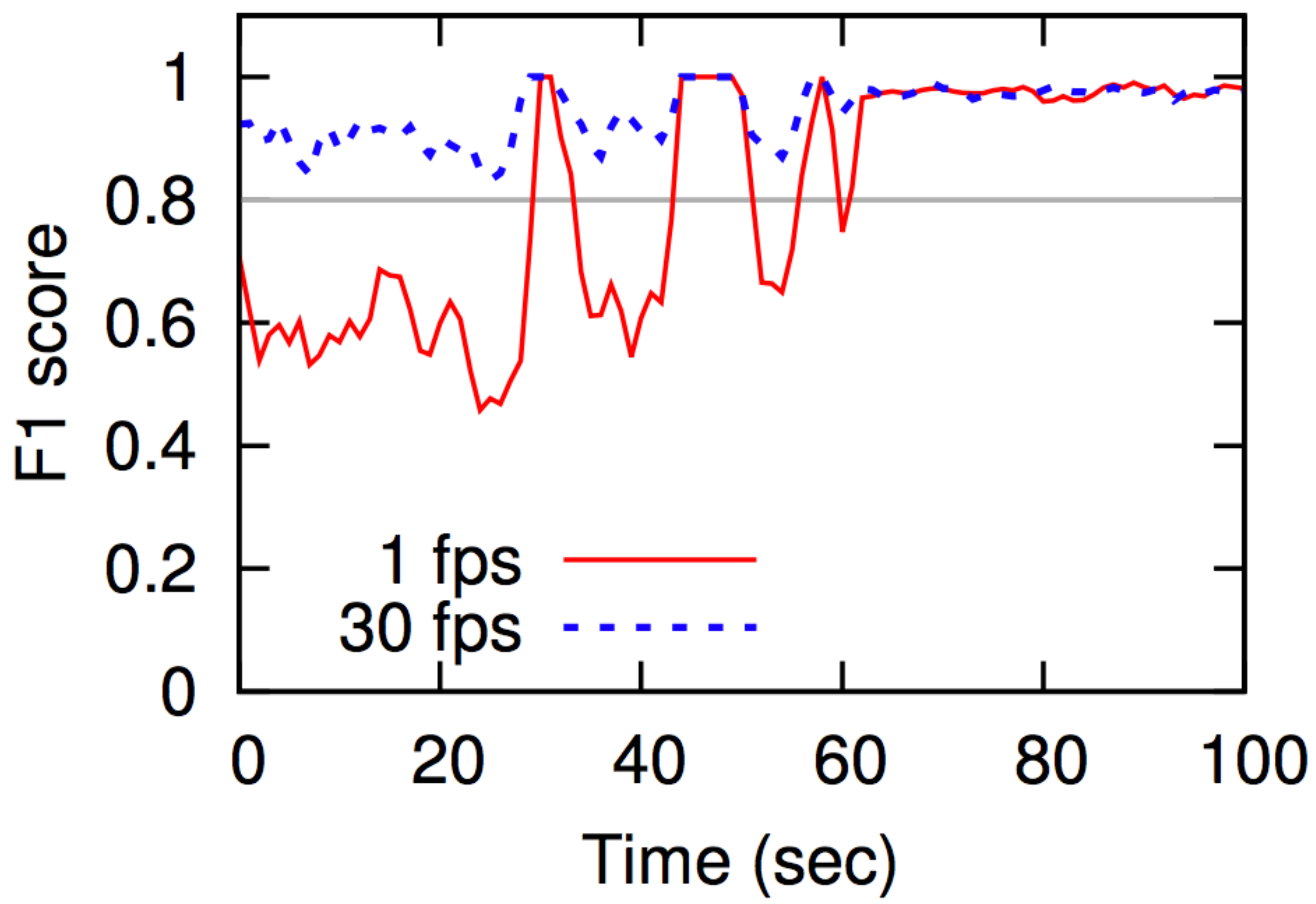
Follow on from last time

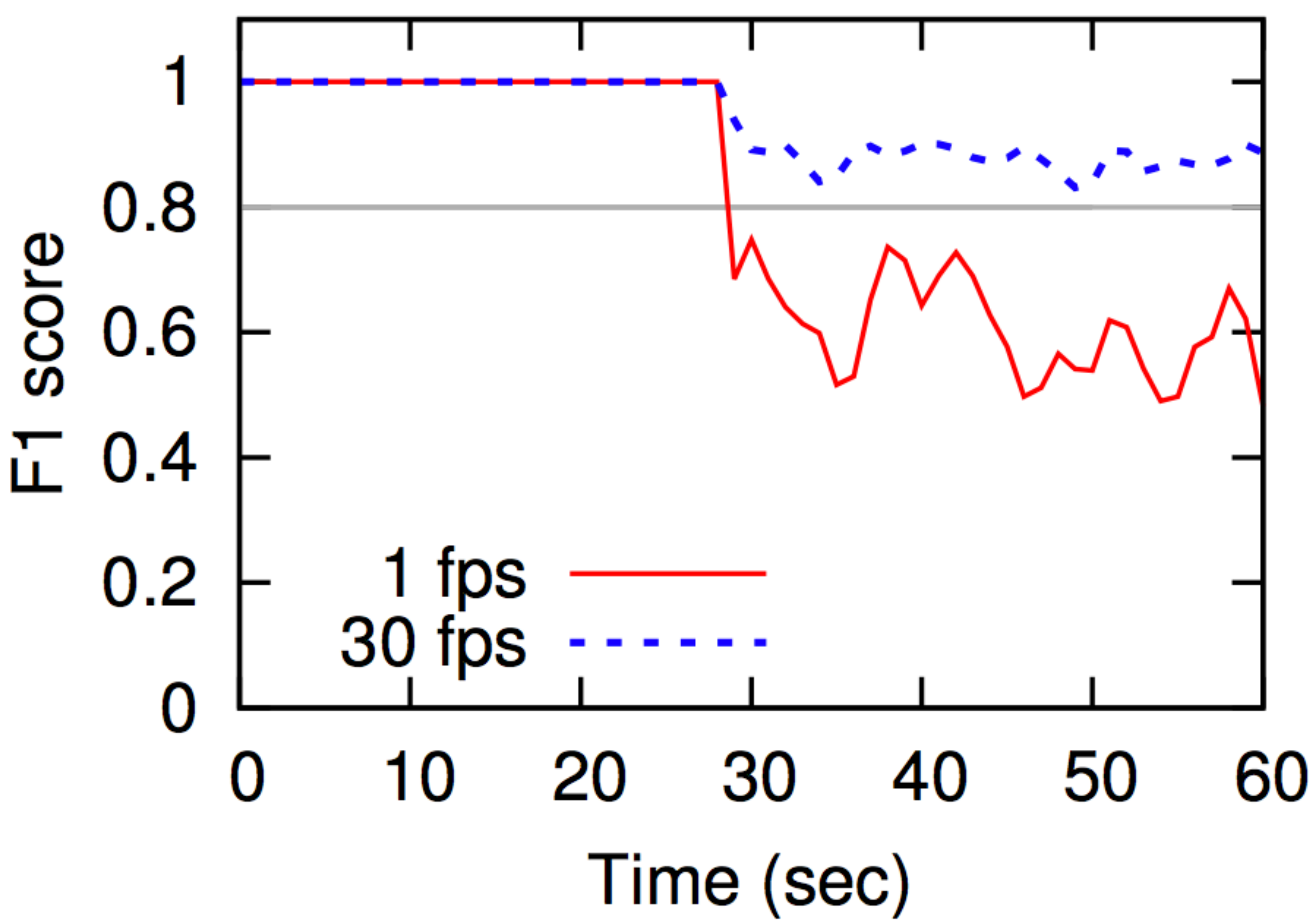
- Recall idea of NoScope: train a cheap model that is specialized for contents a specific video stream (model distillation)

- Alternative (more traditional) specialization strategy: choose among set of pretained models to find cheapest (sufficiently accurate) model for the job
  - "Knobs" to configure:
    - Input image resolution
    - Input image frame rate
    - DNN to use (Resnet101, Resnet50, Inception, MobileNet, etc.)
    - Thresholds on frame-to-frame difference detectors, etc.

Simple example

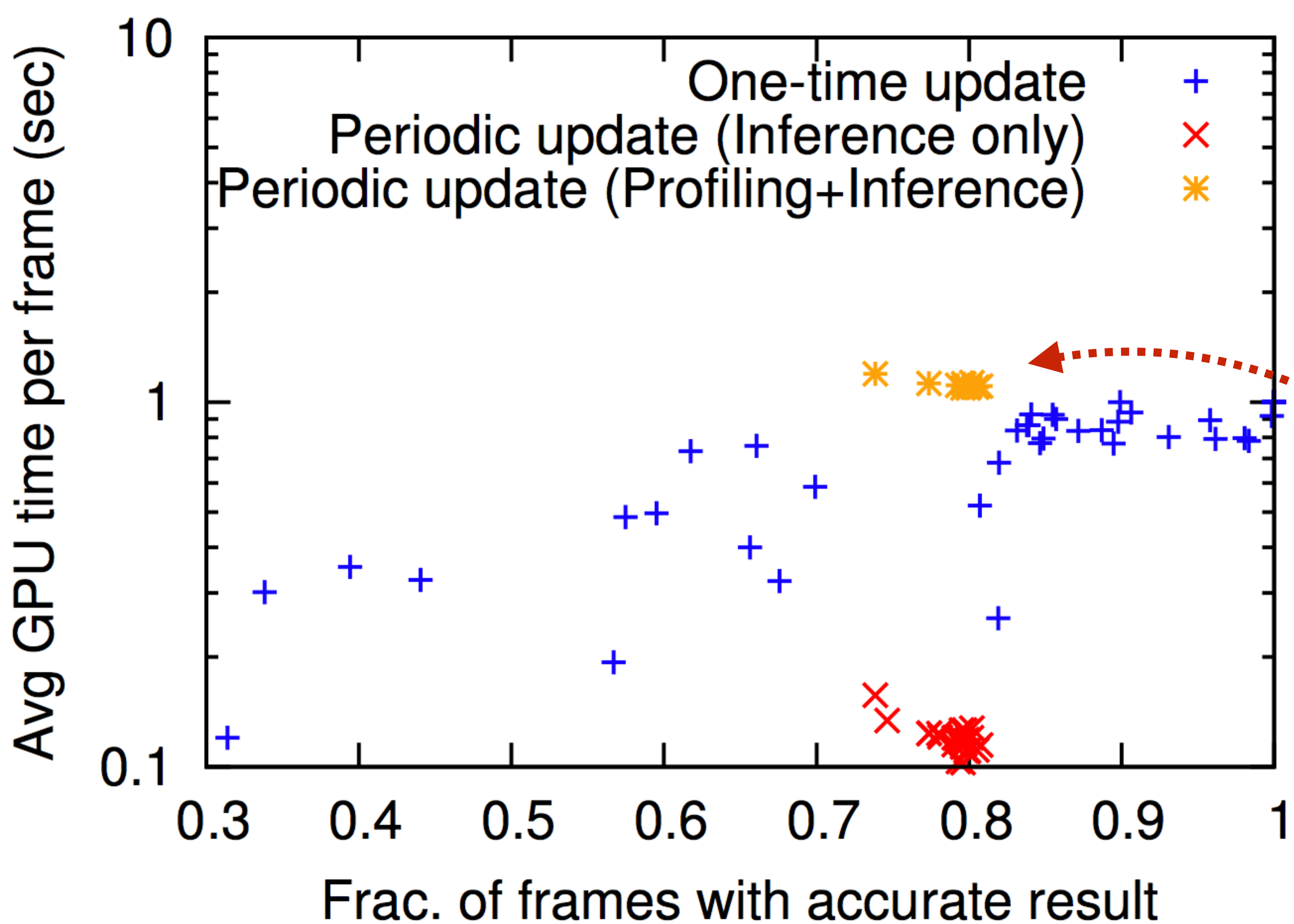
Appropriate frame-rate sampling depends on whether cars are moving





Challenge of distribution shift

- If distribution of video stream is non-stationary, up-front specialized cheap model looses accuracy as contents of video change (specialized model needs to be periodically changed)



Results from object detection task on traffic camera video

Periodic update = every 4 seconds

Challenge: cost of profiling to adaptively determine which model to run eliminates potential benefits of model specialization

Reducing the cost of profiling

- Cost of profiling is running candidate models at points in search space (profiling different values for all knobs)

- Idea 1: set of most-likely-to-be-good models changes slowly over time

- Idea 2: similar streams have similar most-likely-to-be-good candidate models

Employing idea 1

- Assume model updates every segment (e.g., 4 seconds)
- Profile all C model configurations for time segment 1
  - Retain top-K configurations
- Profile only top-K configurations in future segments
- Reset after window of N segments

Let S be number of segments before reset (~4)

Let K be size of candidate set (K << C)

profiling cost = C + (N-1) x K << C x N

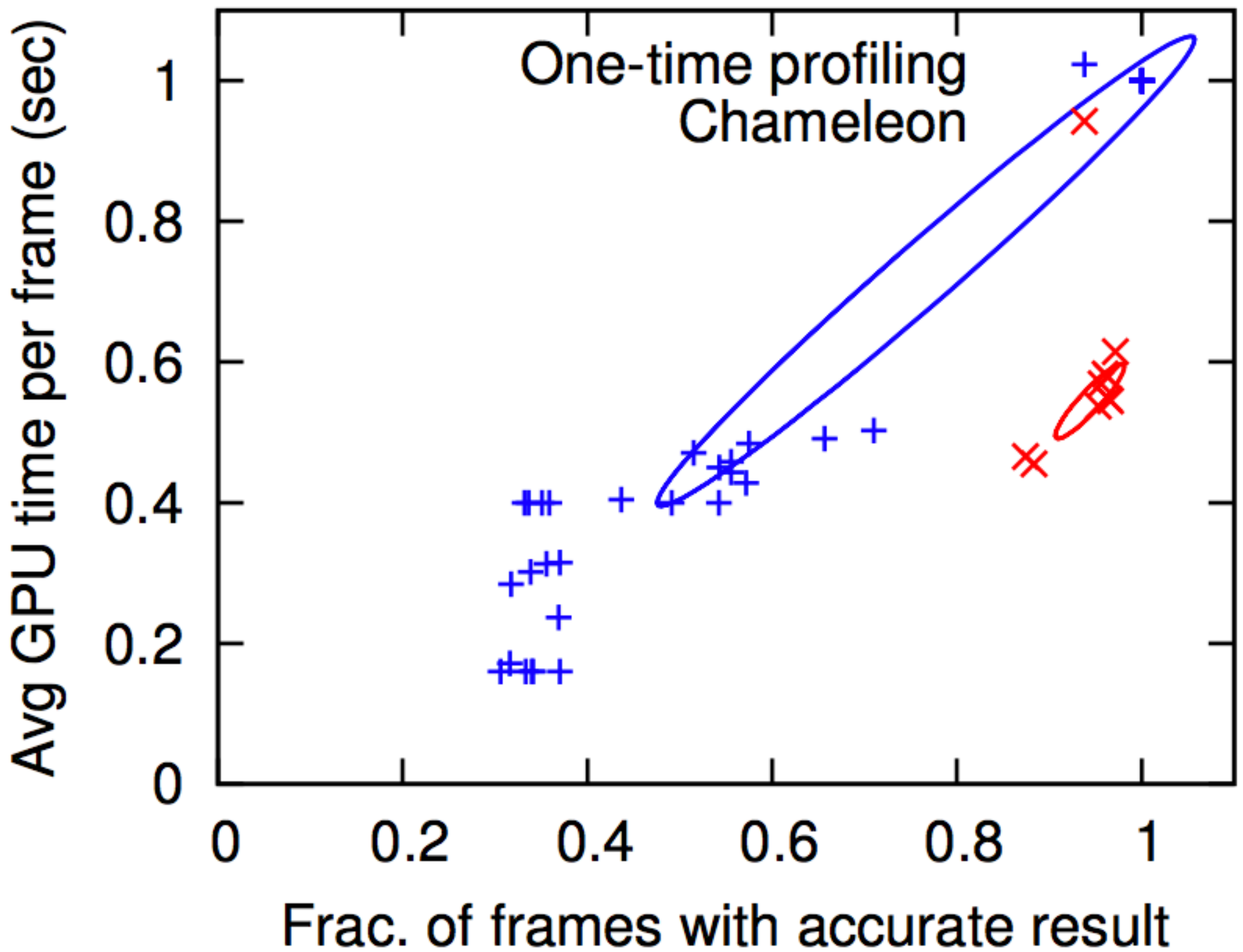
Assumption: bad model configurations tend to remain bad for longer periods of time

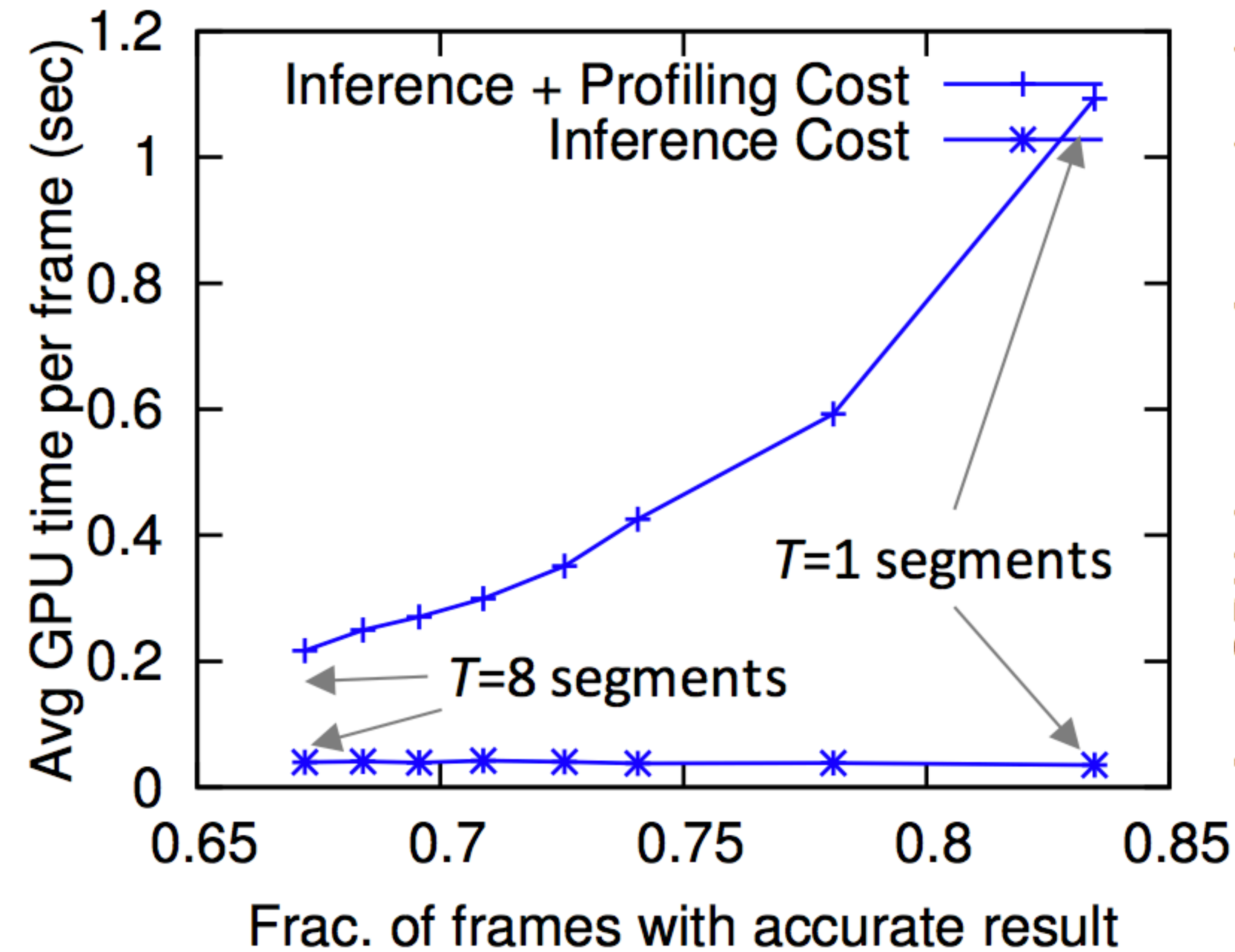
Employing idea 2

- Assume many video cameras throughput a city
- Cluster cameras by how similar their streams are
- Only one camera per cluster needs to perform full profiling to identify top-K candidate set
  - Other cameras just perform top-K profiling

Assumption: bad model configurations tend to remain bad for longer periods of time

Intelligent profiling makes adaptivity profitable





Across dataset of multiple streetlight cameras, when keeping accuracy similar, 2-3X speedup compared to profiling once (really, once per 150 seconds)

??

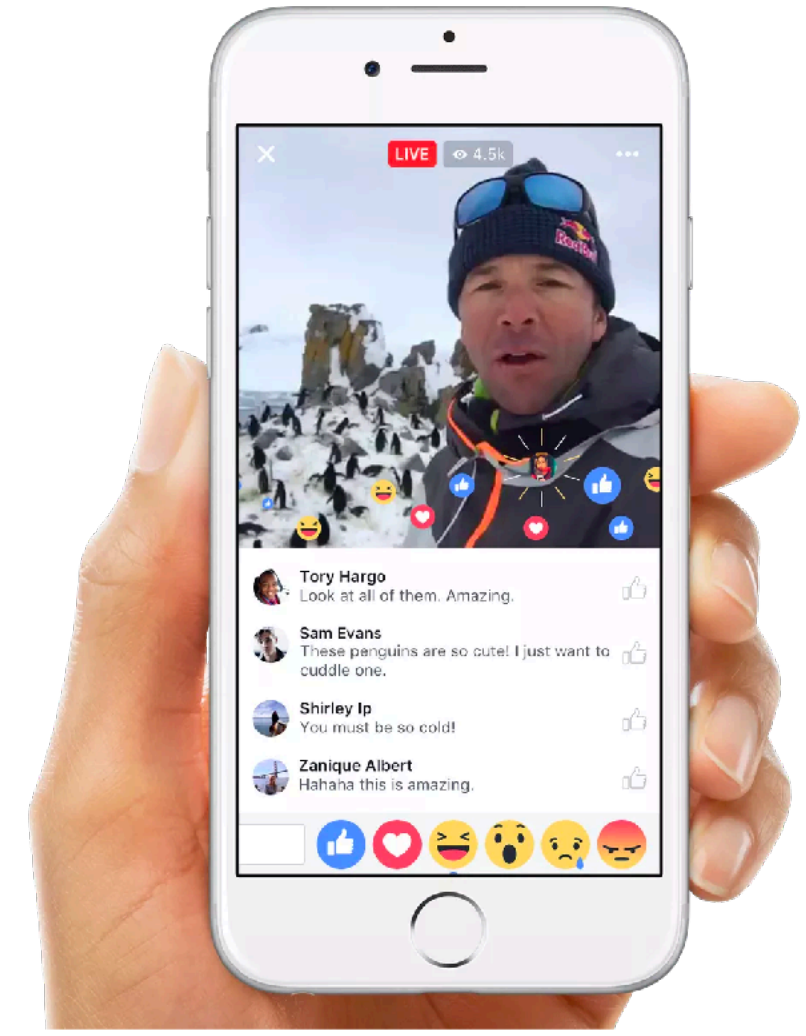
But really the problem with profiling once is that accuracy is highly variable (see accuracy variance of blue crosses)

Managing video ingest at Facebook

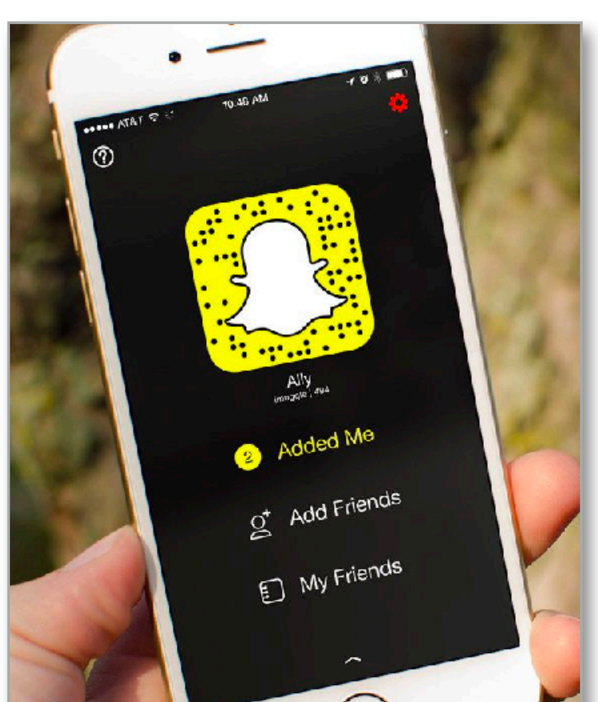
Big video data

Facebook 2016:
100 million hours of video watched per day

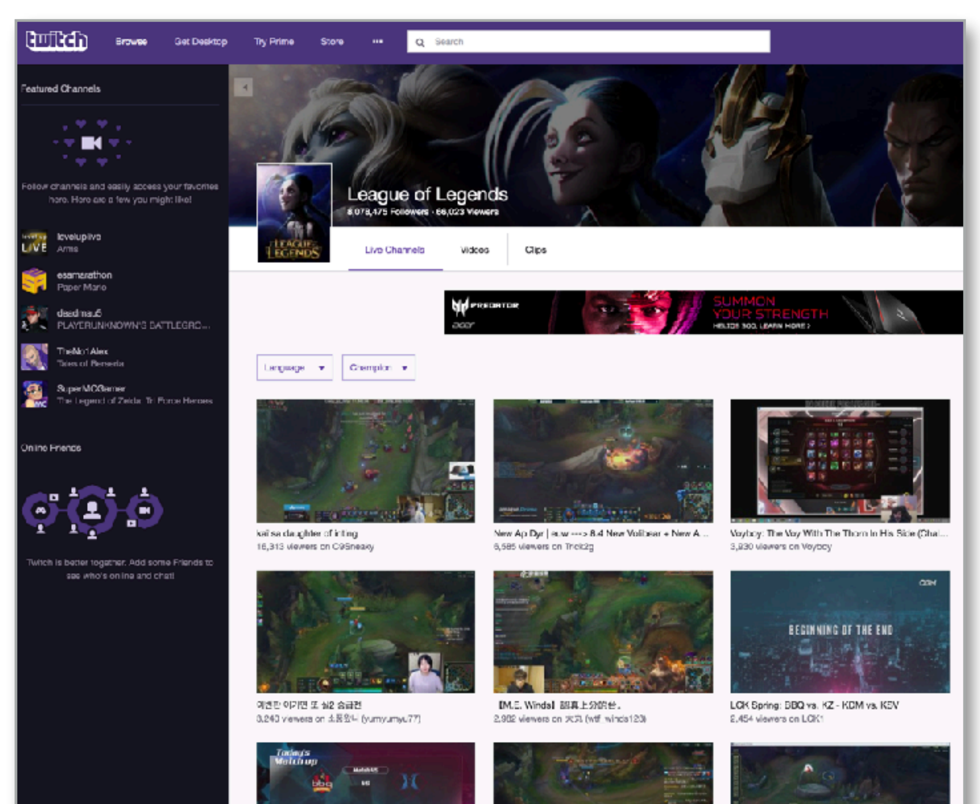
Youtube 2015: 300 hours uploaded per minute [Youtube]



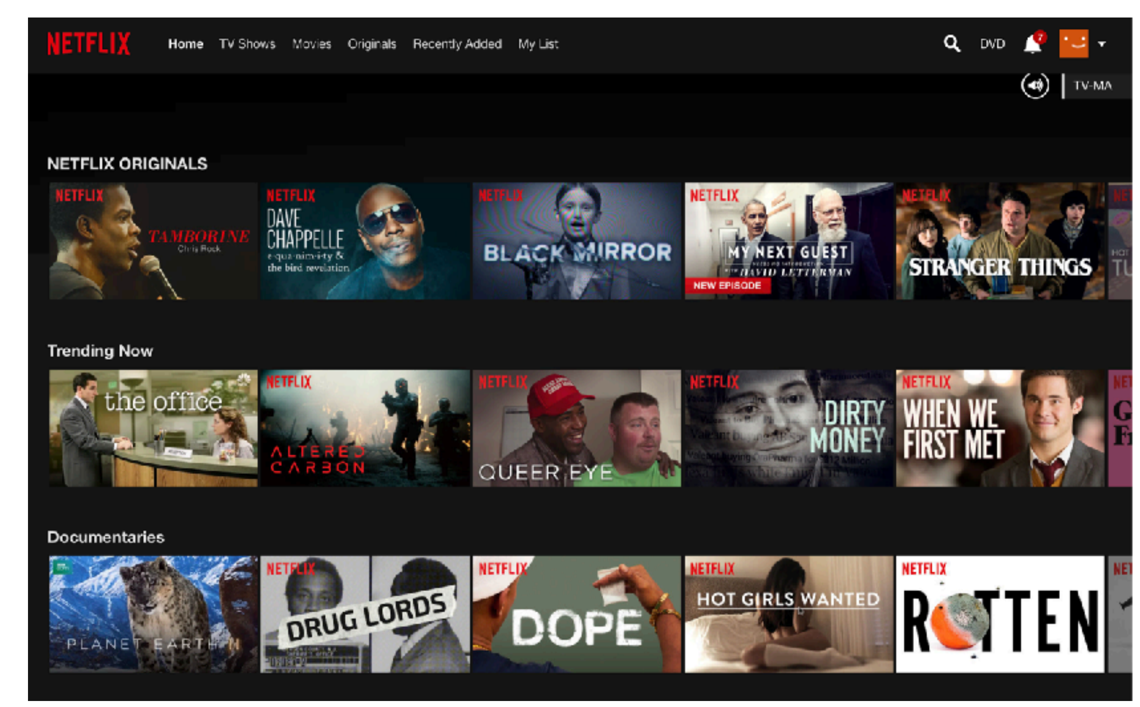
FB Live Video



Snapchat Video



Twitch



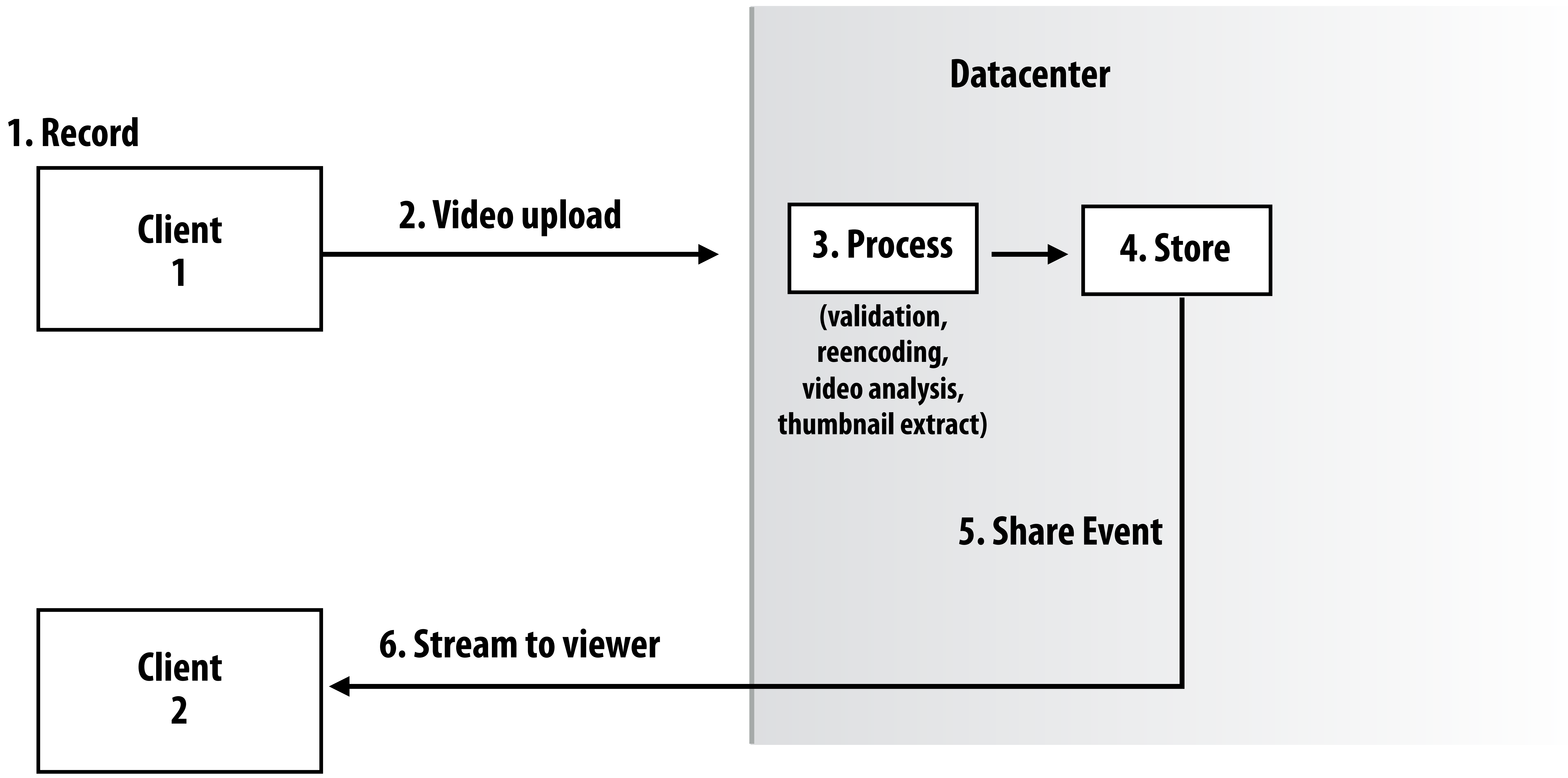
Netflix

Facebook Streaming Video Engine (SVE)

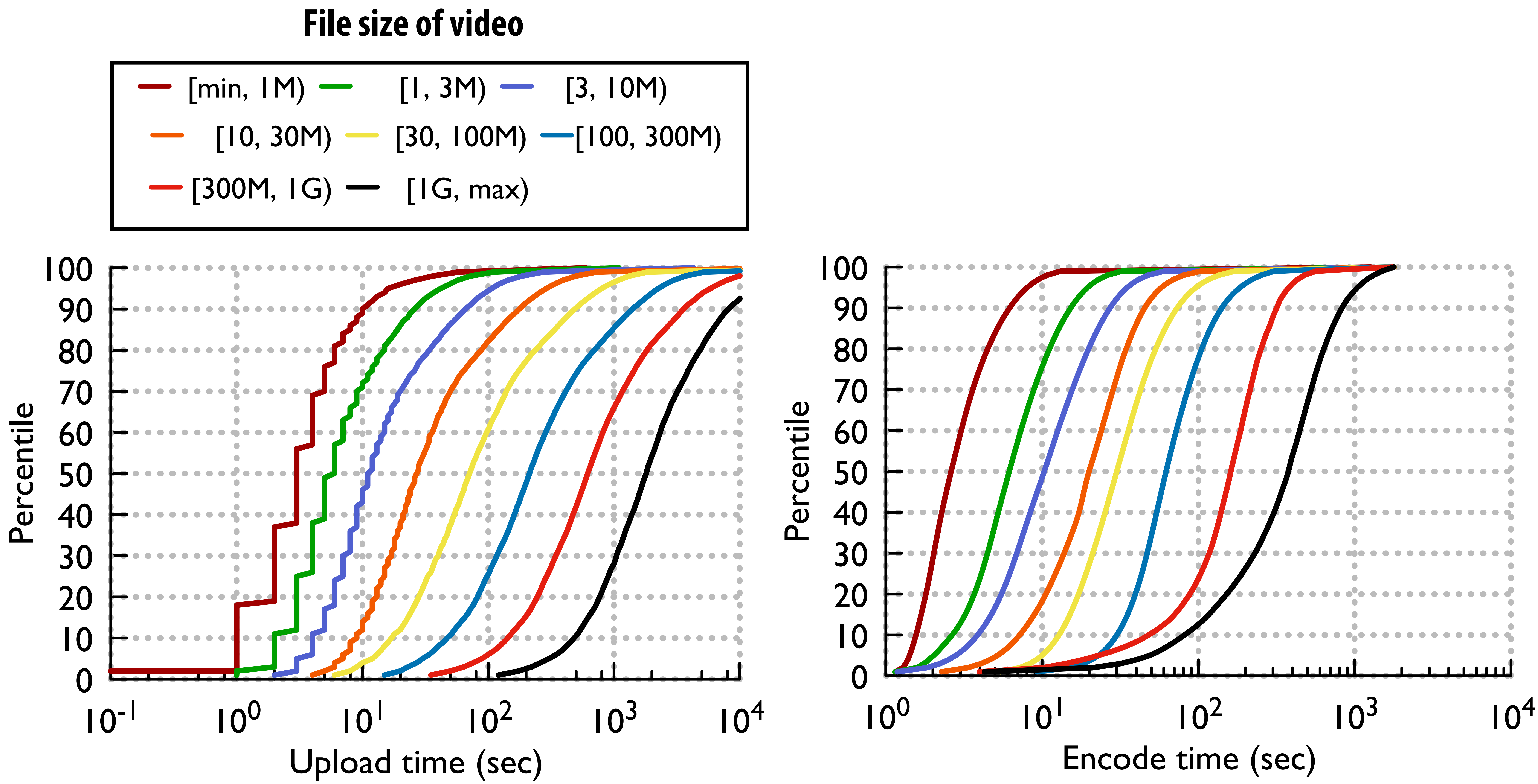
- **Designed for non-streaming video upload applications (not Facebook Live)**
  - Facebook video posts
  - FB Messenger video shares
  - Instagram Stories
  - 360 videos

- **Goals/requirements:**
  - Low latency: *minimize latency* of start of upload to sharable state
    - Particularly important for FB Messenger uploads
  - Flexible (support variety of applications such as those listed above, with different processing pipelines after upload)
  - Robust to faults and overload

Basic video sharing pipeline



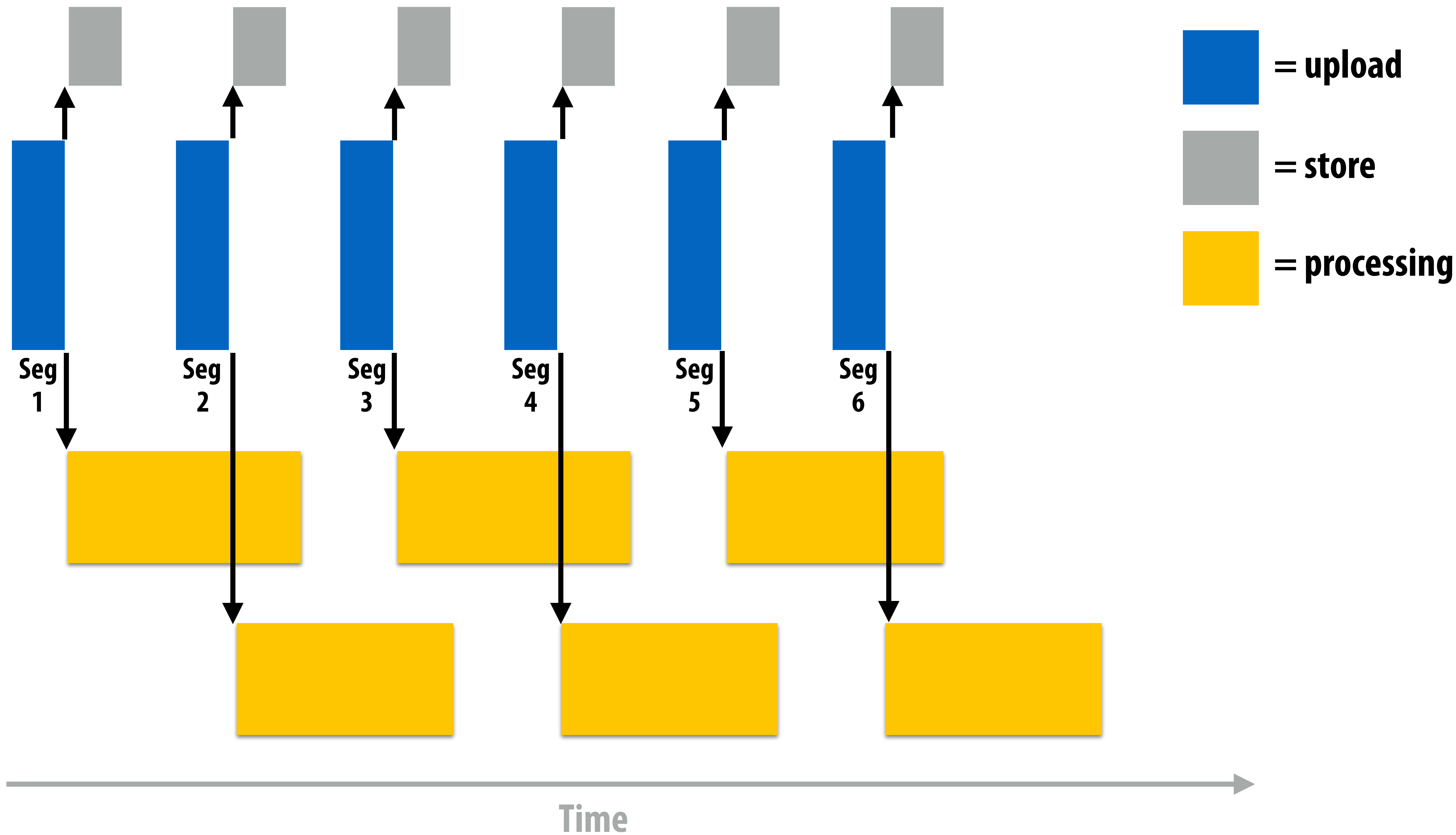
Video upload and processing times *



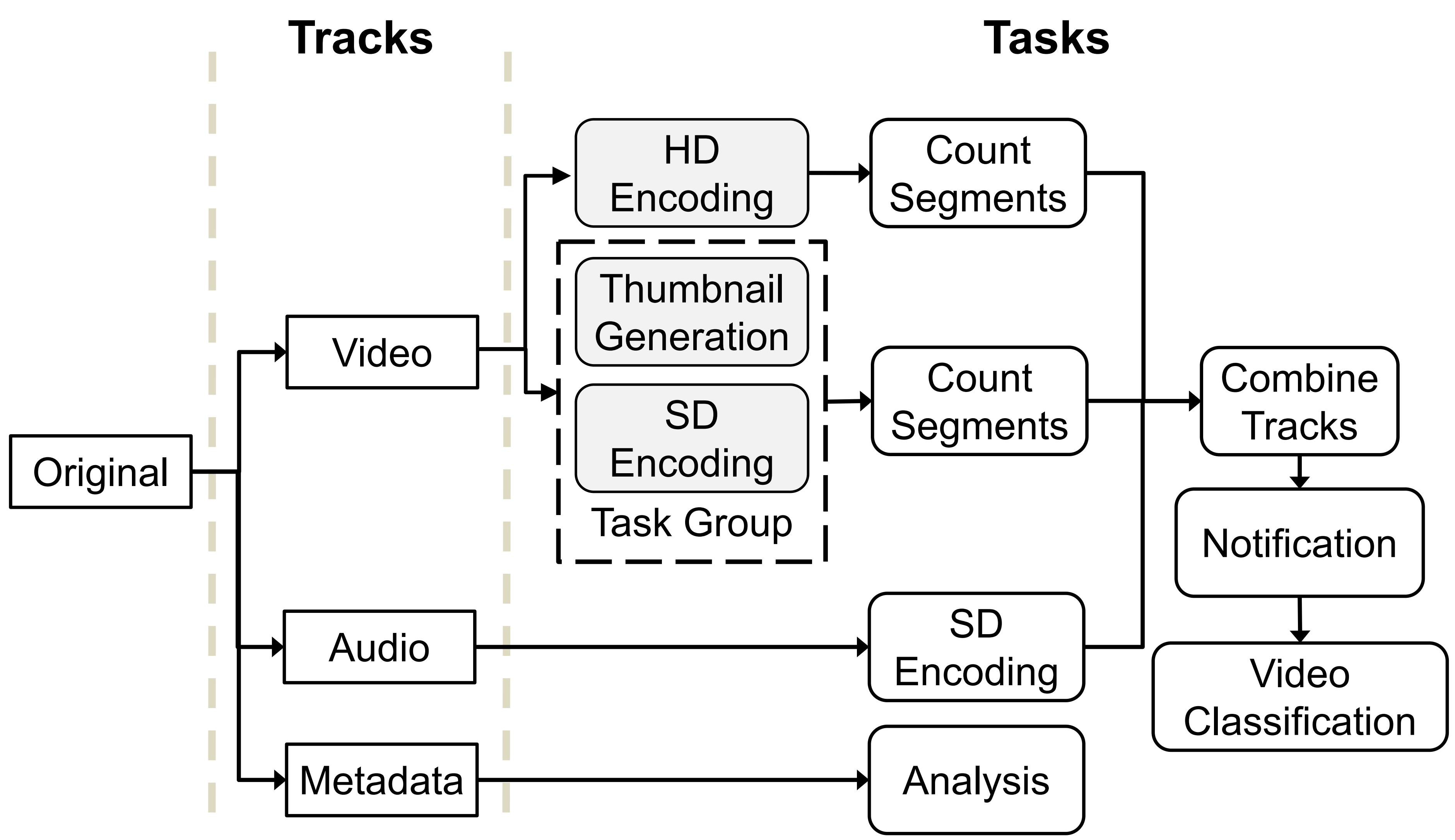
* Serialized times (SVE system will parallelize encoding across segments as discussed in a few slides)

Pipelining upload and processing

- Client application partitions video into segments prior to upload
- Client application optionally downsamples video (skipped if video recorded at low enough resolution, internet connection is fast, or device does not support HW accelerated encode)
- Upload and processing of video is pipelined (upload and processing is mostly parallelized)
- Processing itself can be parallelized across segments



DAG representation of processing



Simple DAG:
Encodes HD and SD version of uploaded video

DAG Specification in Python:

Nodes defined on audio, video, metadata tracks:

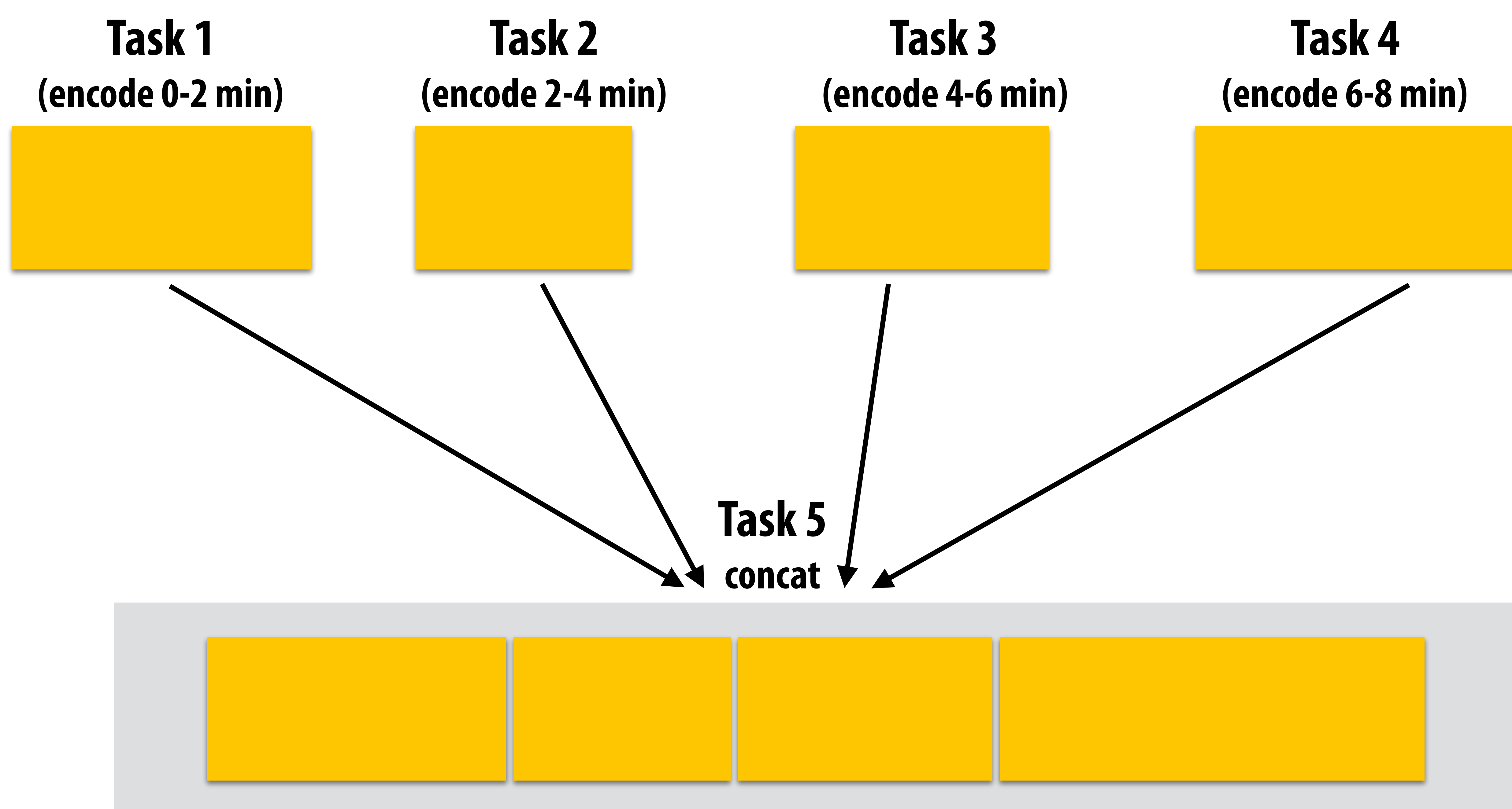
```
pipeline = create_pipeline(video)

video_track = pipeline.create_video_track()
if video.should_encode_hd
  hd_video = video_track.add(hd_encoding)
   .add(count_segments)
sd_video = video_track.add(
  {sd_encoding, thumbnail_generation},
 ).add(count_segments)

audio_track = pipeline.create_audio_track()
sd_audio = audio_track.add(sd_encoding)

meta_track =
  pipeline.create_metadata_track()
  .add(analysis)

pipeline.sync_point(
  {hd_video, sd_video, sd_audio},
  combine_tracks,
 ).add(notify, 'latency_sensitive')
  .add(video_classification)
```

DAG node = "task"
Each task is executed serially on one video segment
Overall DAG execution can be parallelized
(across tracks and segments)

Facebook Video Posts: ~153 tasks
Messenger shares: 18 tasks
Instagram stories: 22 tasks

Coarse-grained parallel video encoding

- Parallelized across segments (I-frame inserted at start of segment)
- Concatenate independently encoded bitstreams



Smaller segments = more potential parallelism, worse video compression
Latency-sensitive applications: 10 second segments
Non-latency sensitive, long videos: 2 minute segments (maximize compression)

Overload control

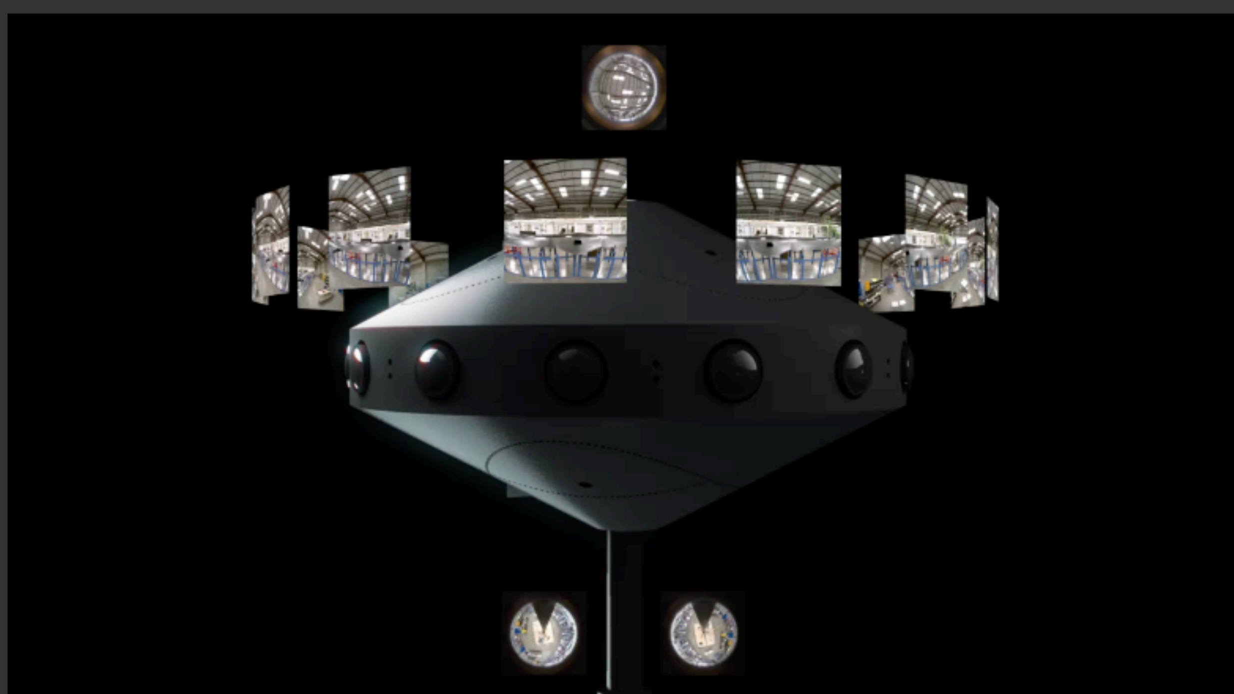
- When FB cannot keep up with the world's video upload rate…

- Delay latency-insensitive tasks
- Rebalance load: redirect uploads to new datacenter region
- Delay processing of new uploads

Scanner: batch video analytics

Emerging "big" video applications

Synthesizing VR video

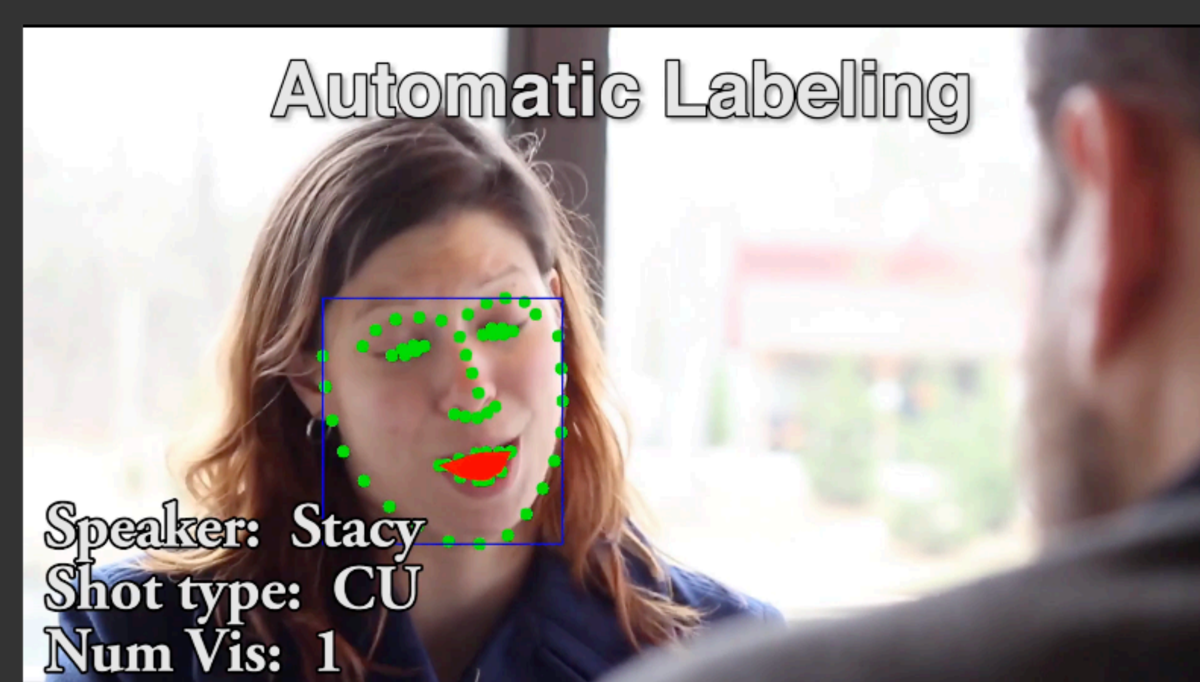


Vehicular video analysis

Drone 3D reconstruction

Markerless motion capture

Computational video editing

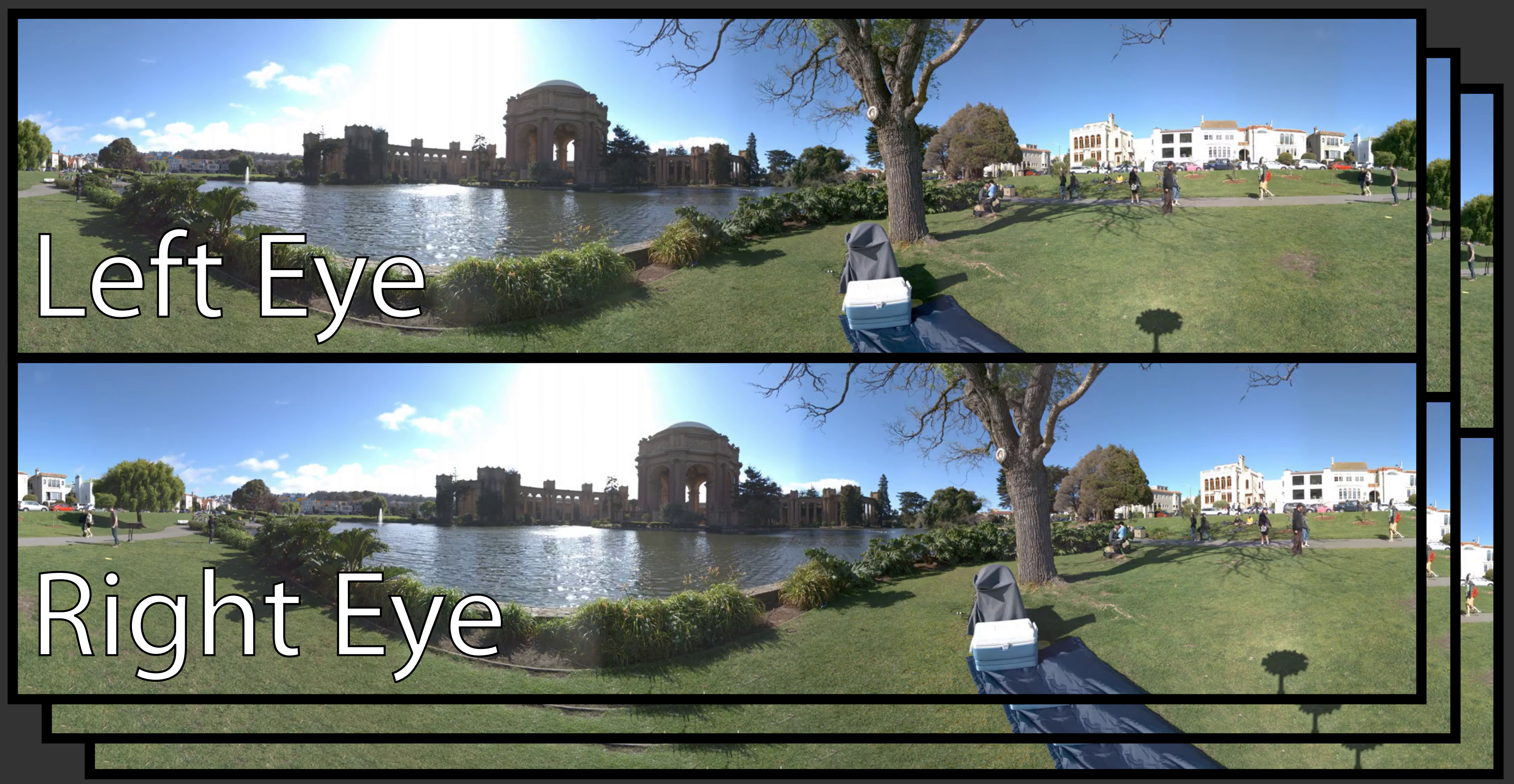


Auto-checkout shopping

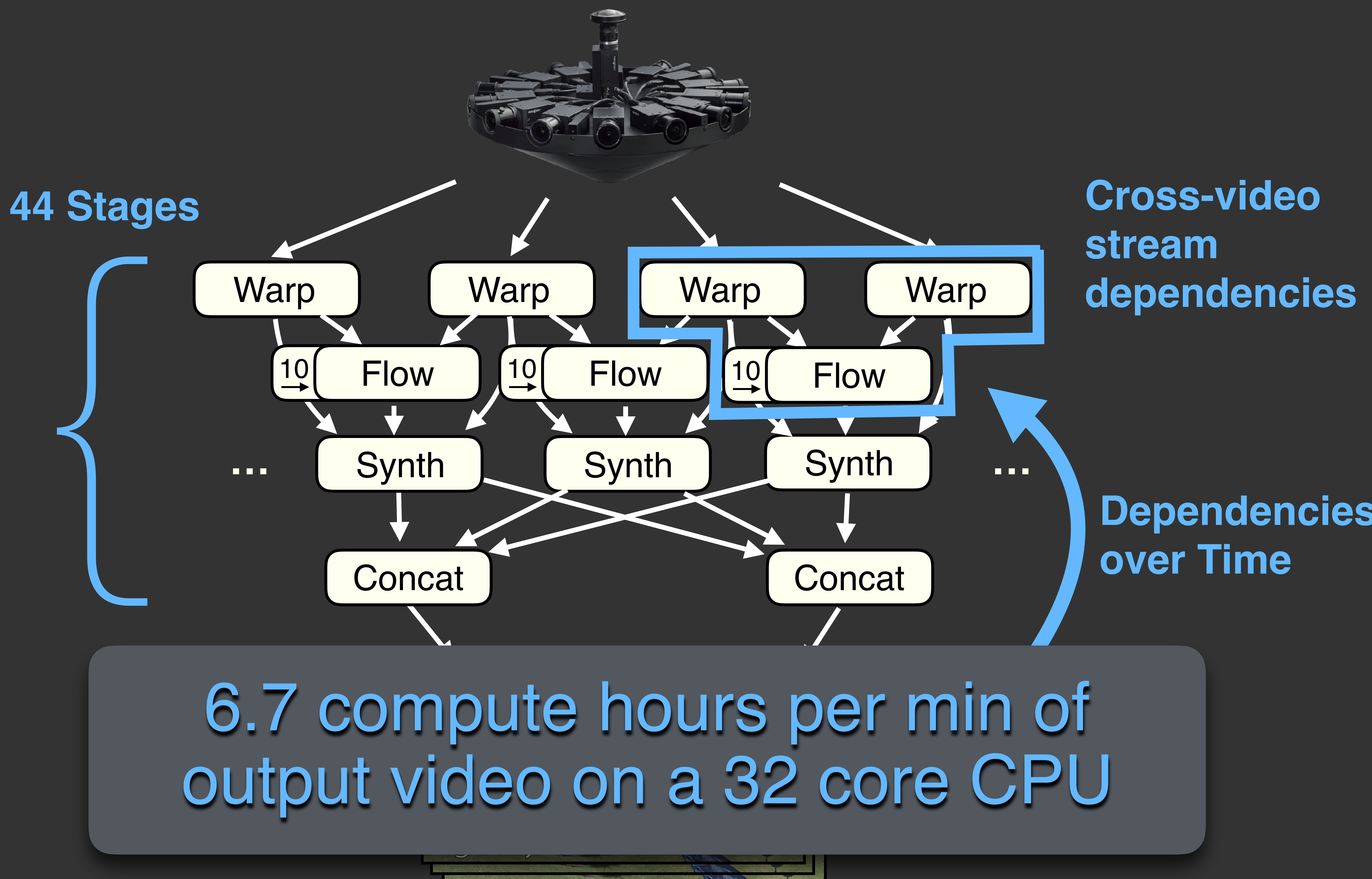
Facebook Surround360 VR video

Input:
14 cameras, 2048 x 2048
300 GB / minute

Output:
8K Panoramic Video



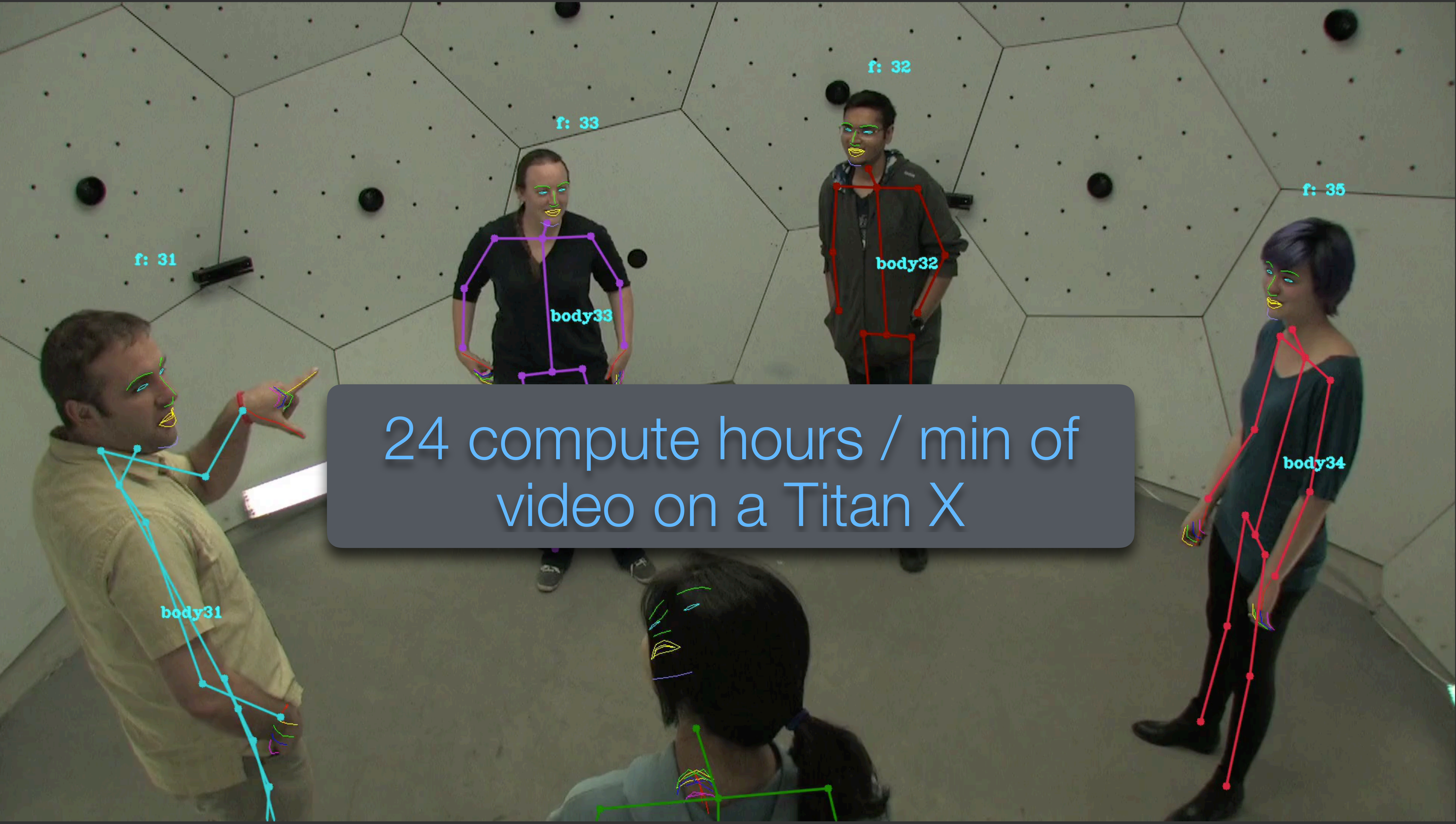
Facebook Surround360 VR video



6.7 compute hours per min of output video on a 32 core CPU

https://code.fb.com/video-engineering/surround-360-is-now-open-source/

Markerless human motion capture



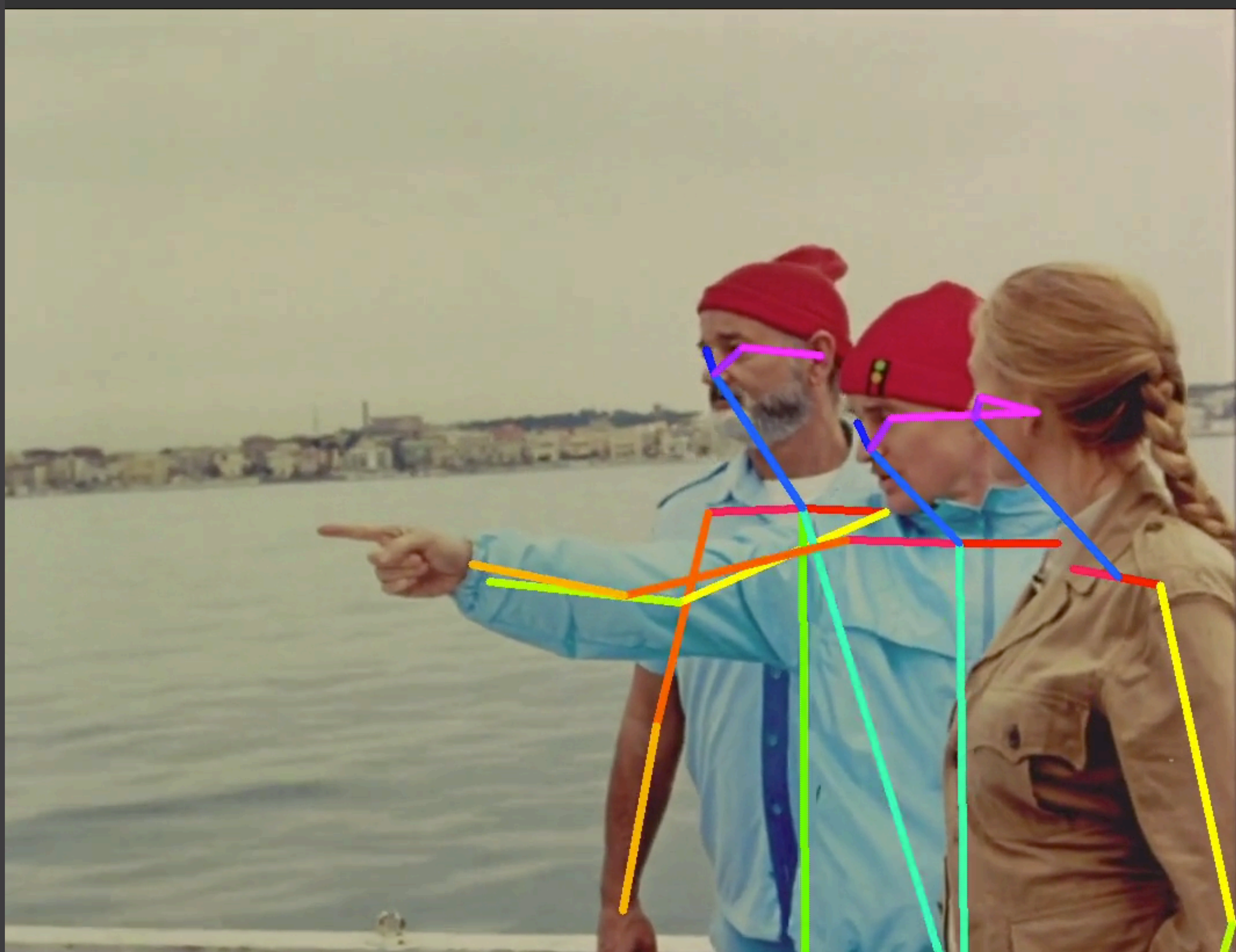
24 compute hours / min of video on a Titan X

[Joo 15]

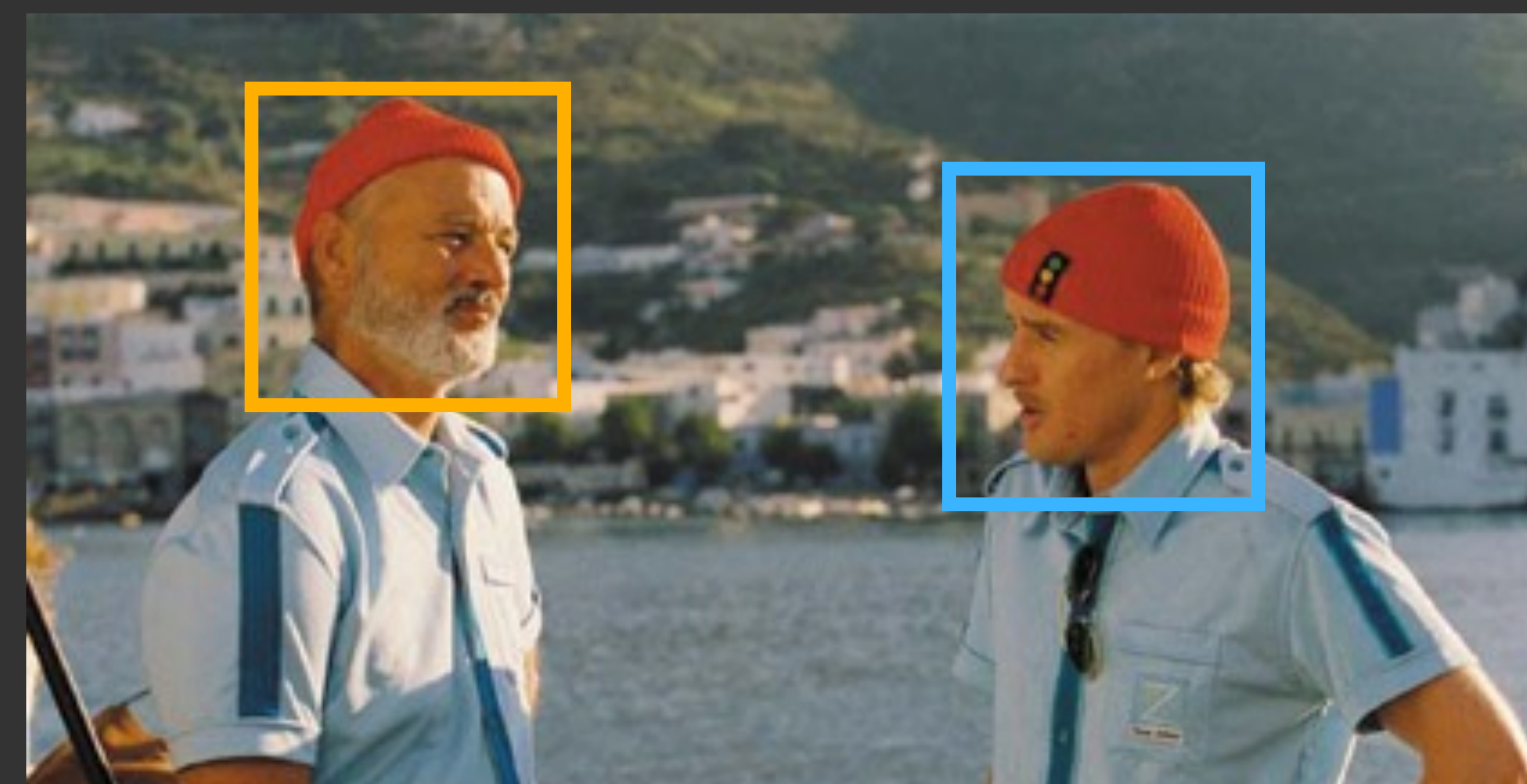
Video analytics

Example: film content analytics

Locating action shots



Actor co-occurrence



https://medium.com/netflix-techblog/ava-the-art-and-science-of-image-discovery-at-netflix-a442f163af6

Three common properties

Large video collections

Existing tools

CUDA

Halide

GL

TF

Clusters of machines

Scanner: a system for...

1. Productively developing big video data applications

2. Efficiently executing these applications at scale

Axes of scalability

Data scalability

10s of concurrent streams

Single video

100s of films
100k Youtube video clips

Compute scalability

4-GPU Desktop

Laptop

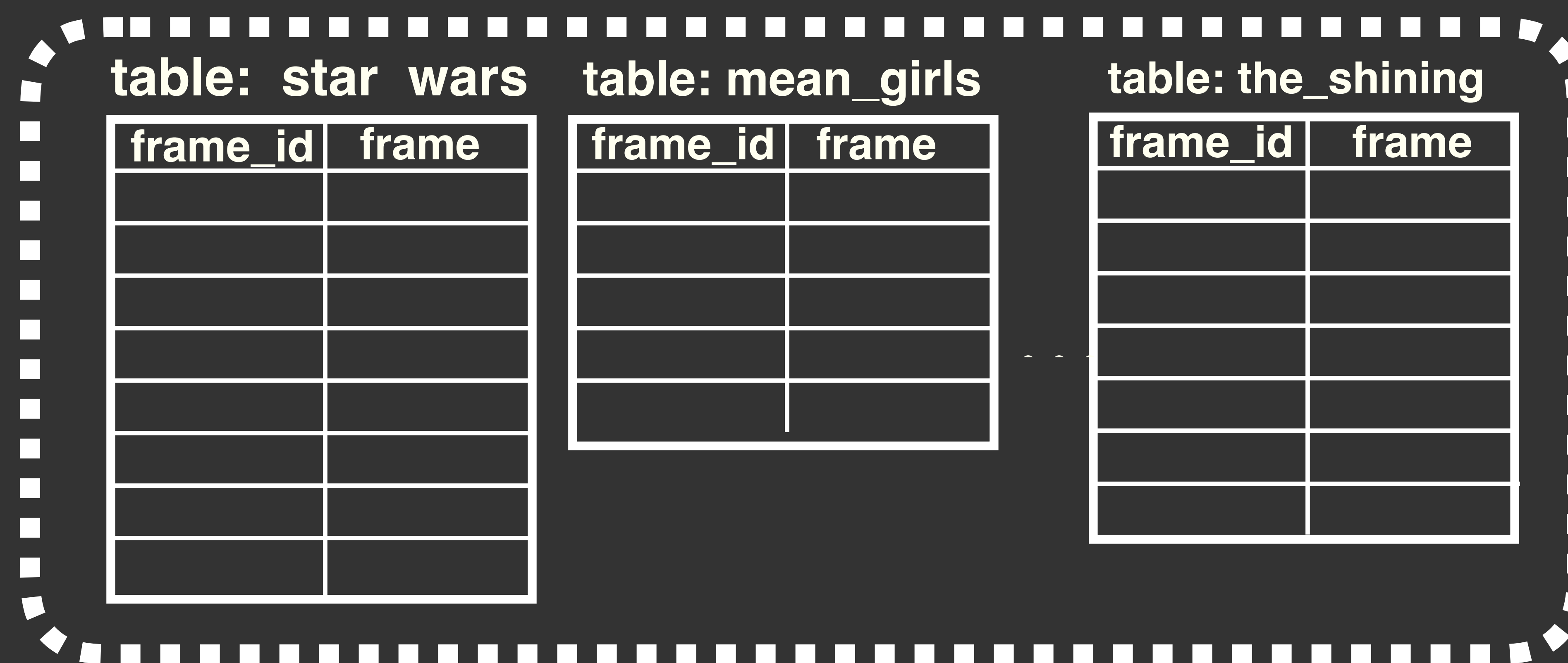
100s of Machines

Scanner overview

Directory of videos

```
movies/star_wars.mp4
movies/mean_girls.mp4
movies/pulp_fiction.mp4
…
movies/the_shining.mp4
```

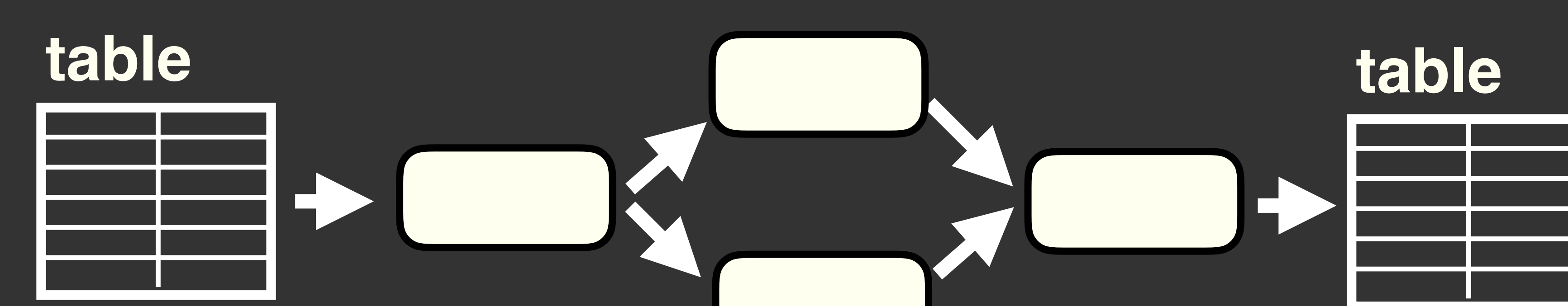
Organize videos as tables



Library of useful functions

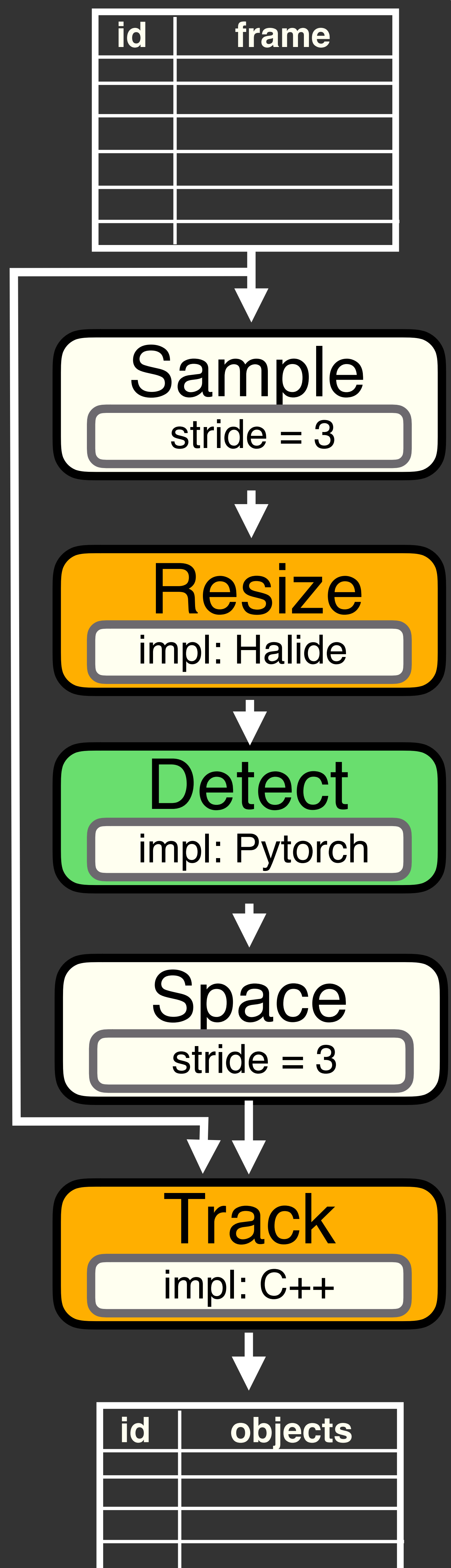
- Image Resize — impl: Halide
- Depth Est. — impl: CUDA
- MaskRCNN — impl: Pytorch
- Optical flow — impl: OpenCV
- Tracker — impl: C++
- OpenPose — impl: Caffe

Construct applications as data flow graphs



Simple example: track player over time

Video processing as a data flow graph



Read 30 FPS video

Subsample to 10 FPS

Resize for DNN detector

Detect person using DNN

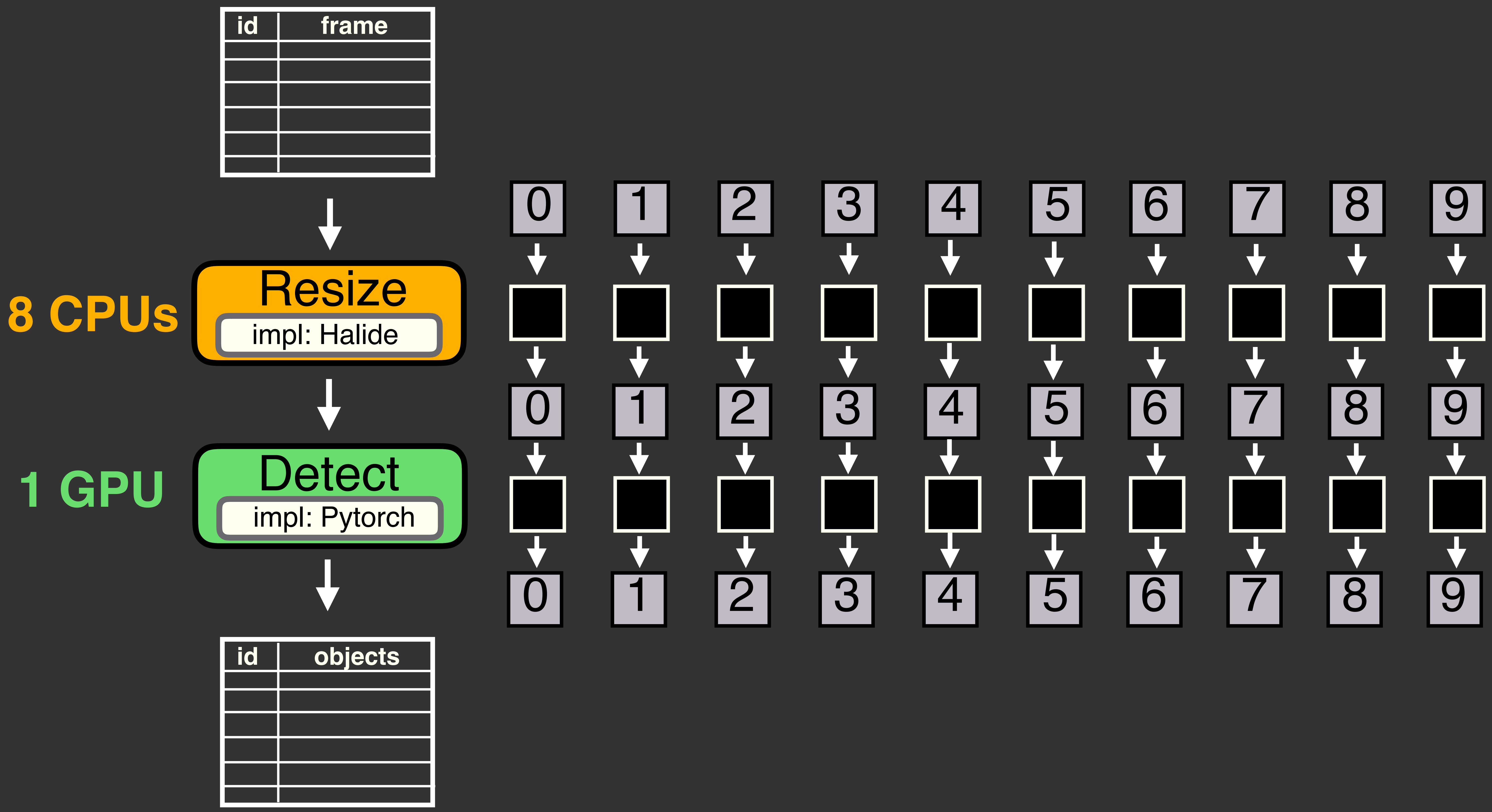
Align with 30 FPS video

Track at 30 FPS

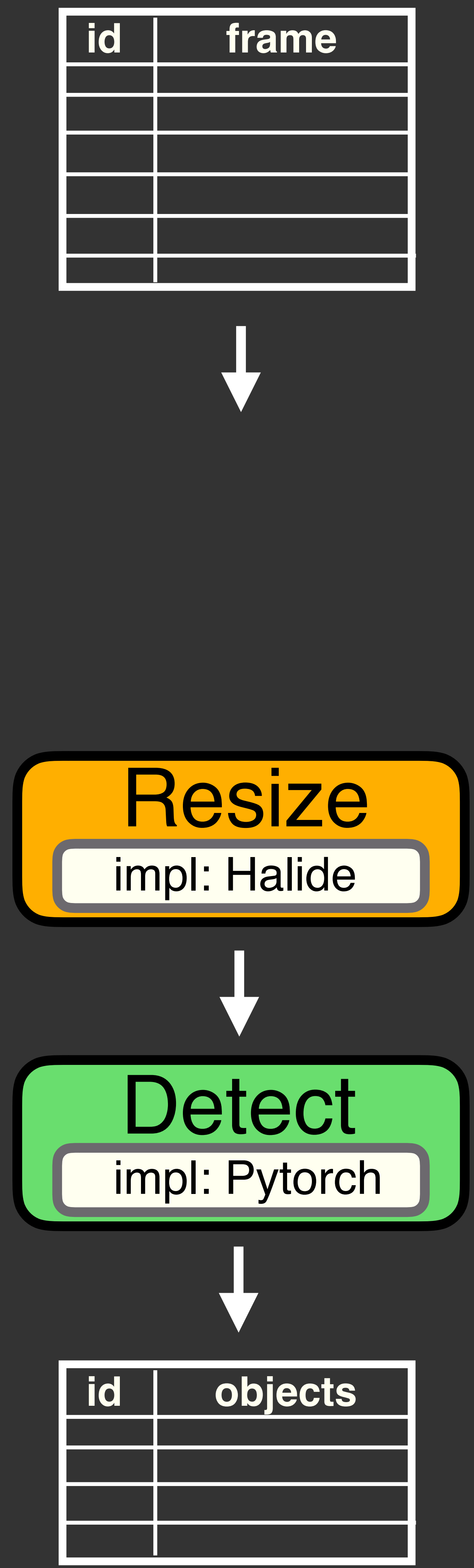
Save tracked boxes

```
db = scanner.Database()
video_tables = db.ingest_videos(videos)

frame = db.sources.Column(video_tables[0])

sparse_frames = db.streams.Stride(frame, 3)

transformed = db.ops.Transform(
    frame  = sparse_frames,
    width  = 496, height = 398,
    device = GPU)

detections = db.ops.Detect(
    frame = transformed,
    model = 'face_dnn.prototxt',
    batch = 8,
    device = GPU)

frame_detections = db.streams.Space(detections, 3)

faces = db.ops.Track(
    frame      = frame,
    detections = frame_detections,
    warmup     = 20,
    device     = CPU)

output = db.sinks.Column(faces)
```

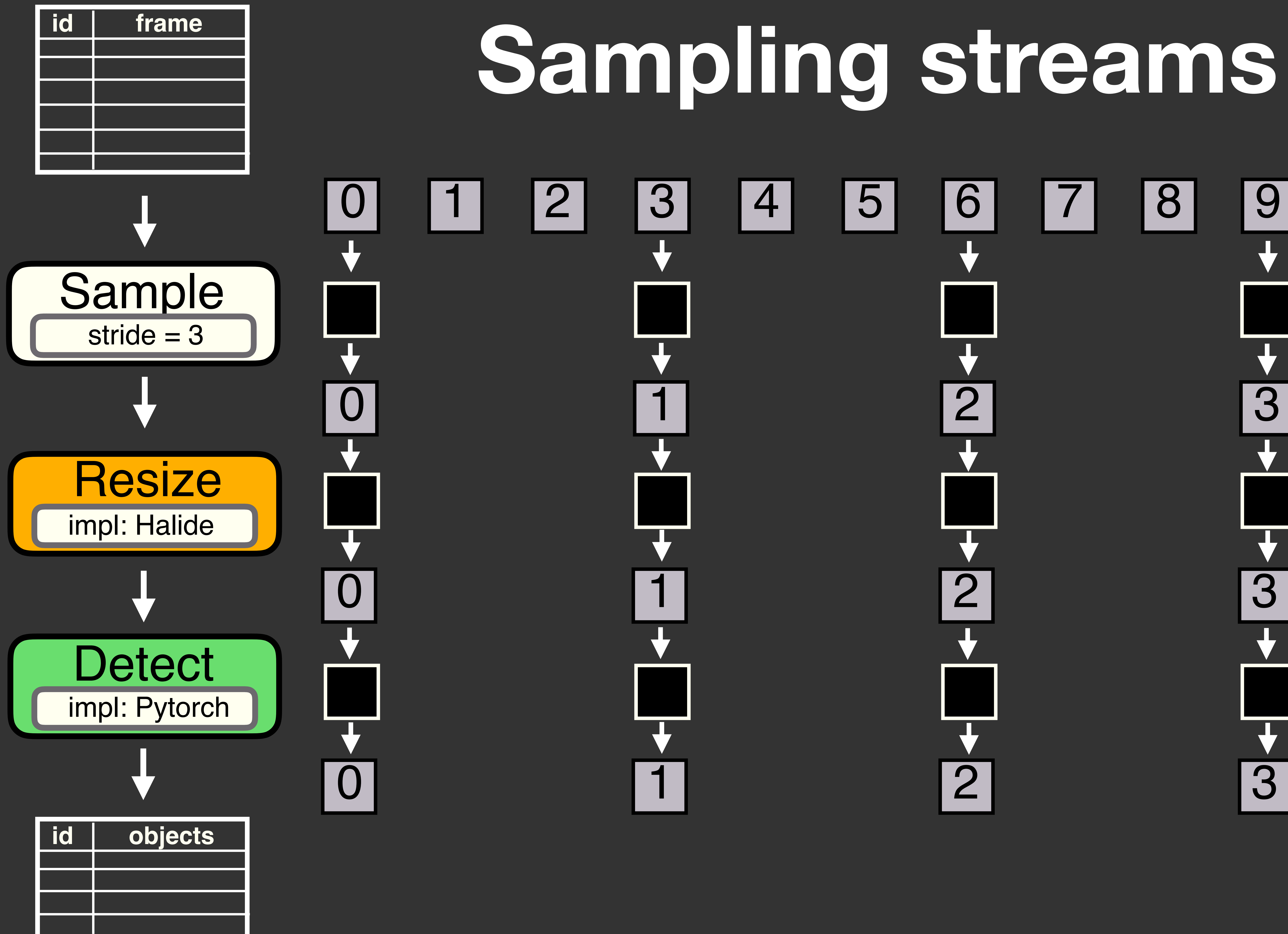
Mapping over streams



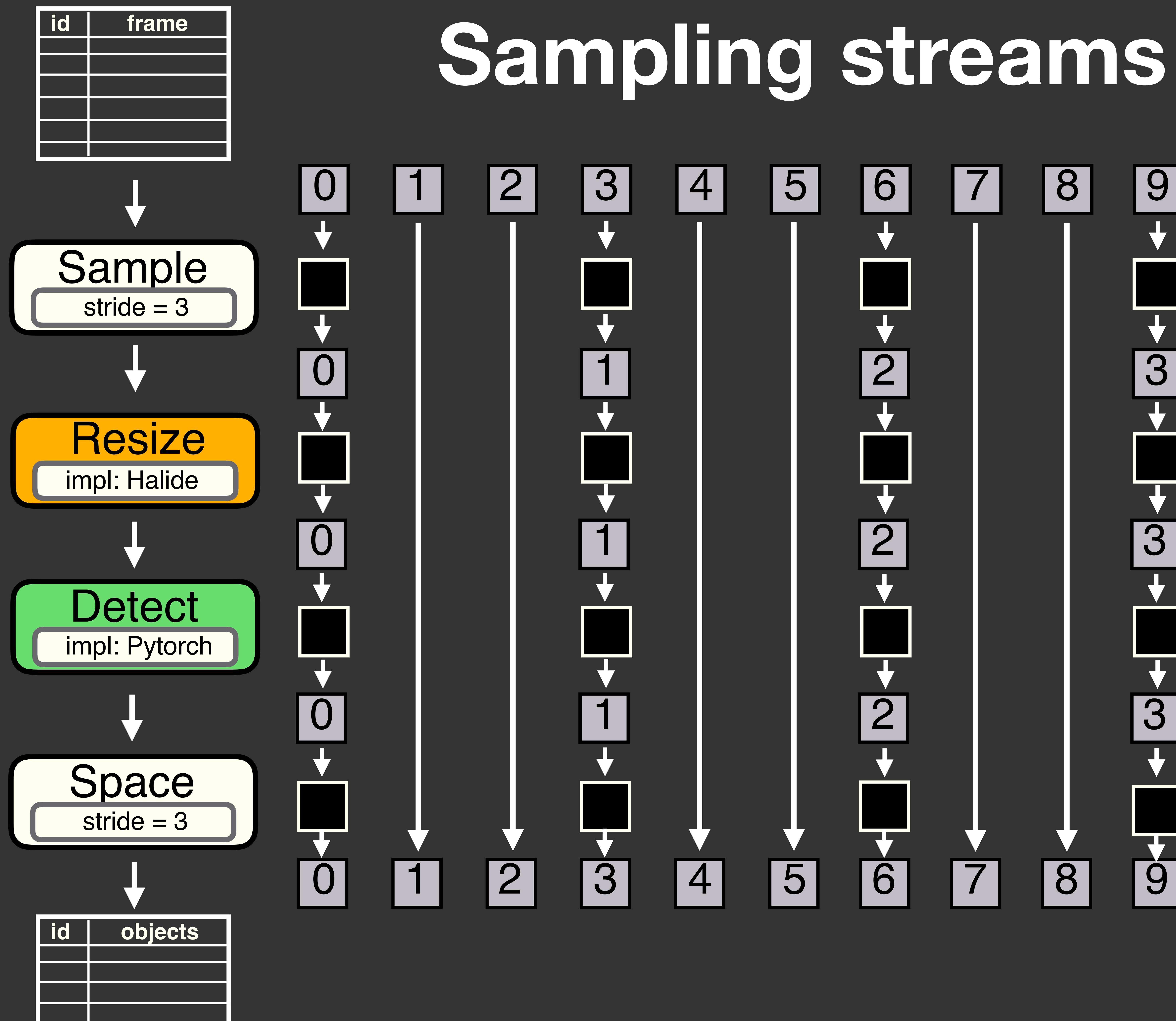
Sampling streams



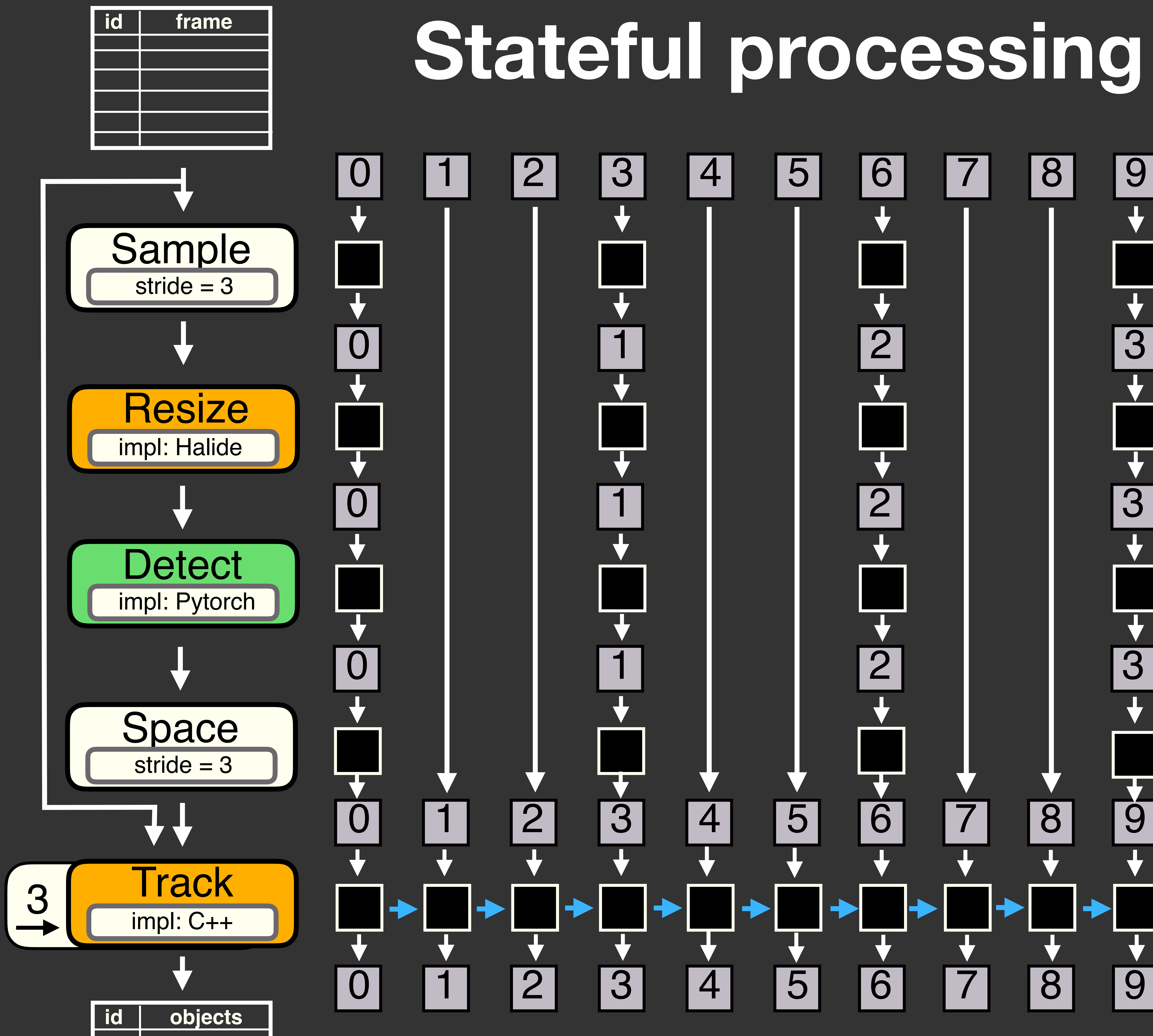
Sampling streams



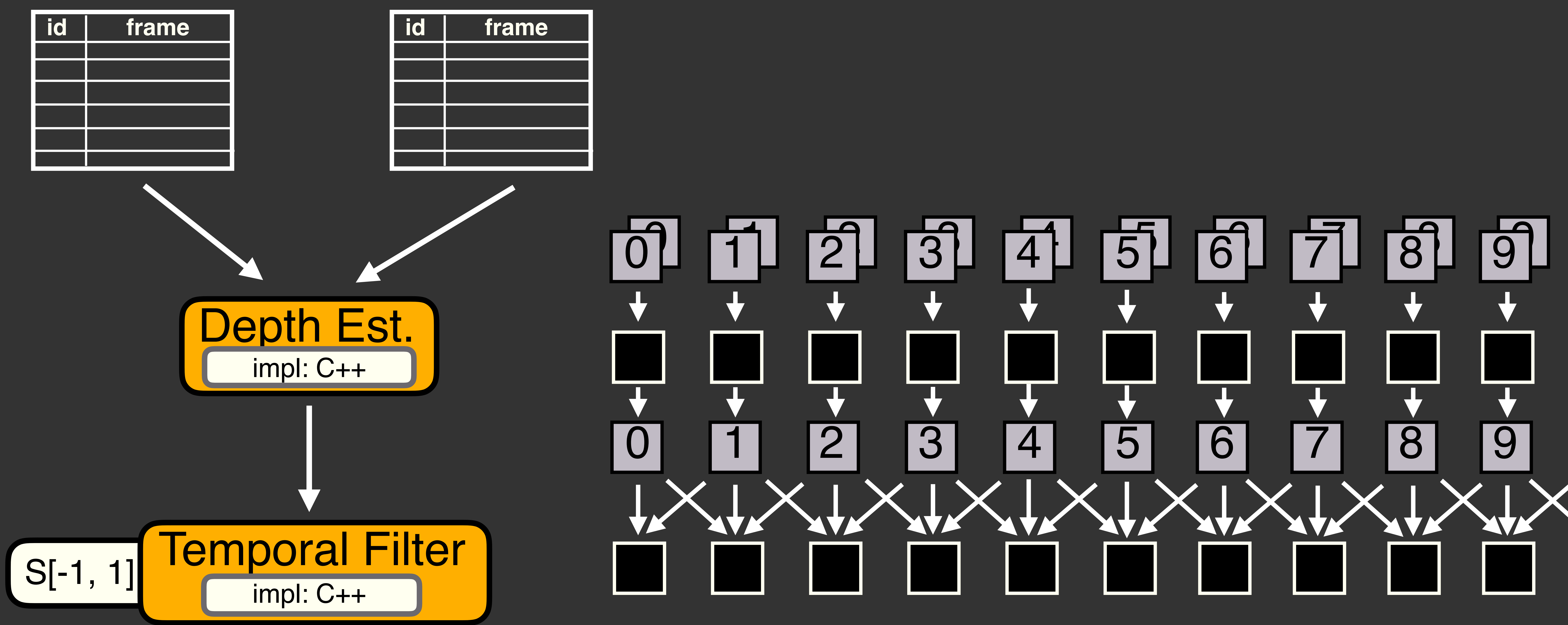
Sampling streams



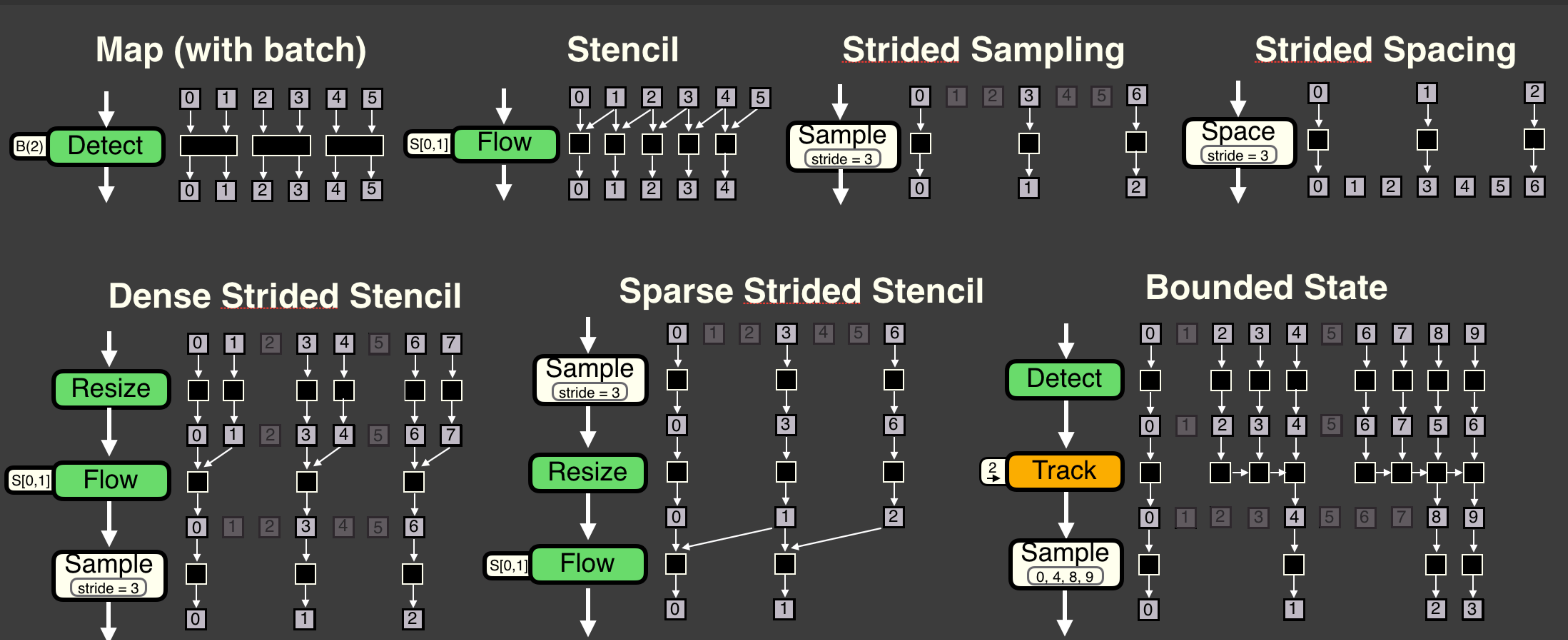
Stateful processing



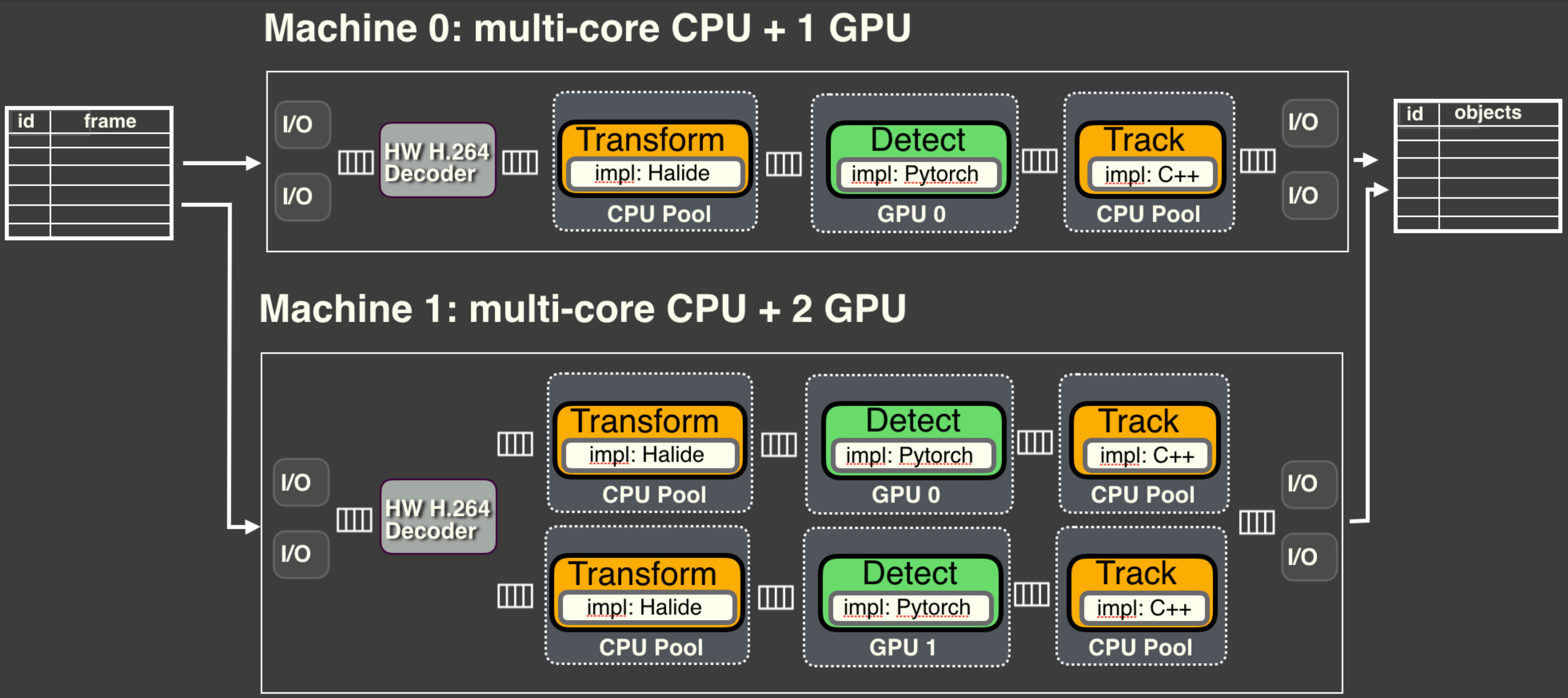
Additional operations



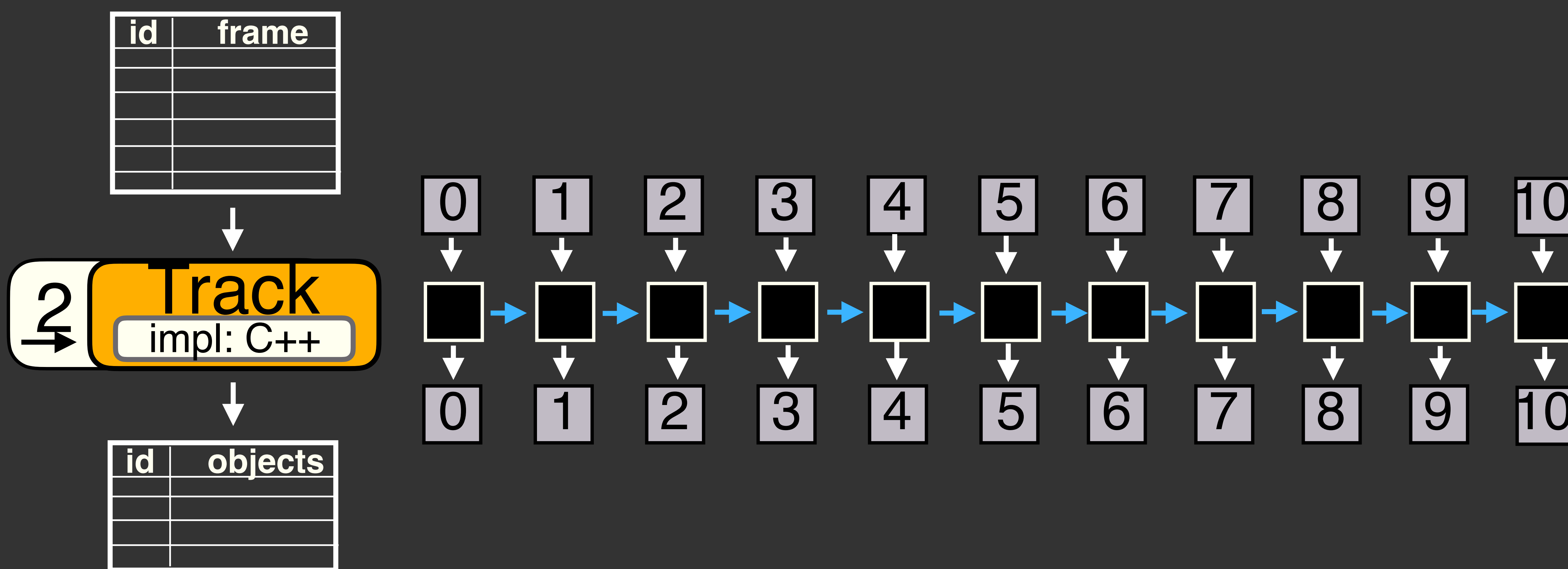
Scanner data flow operators



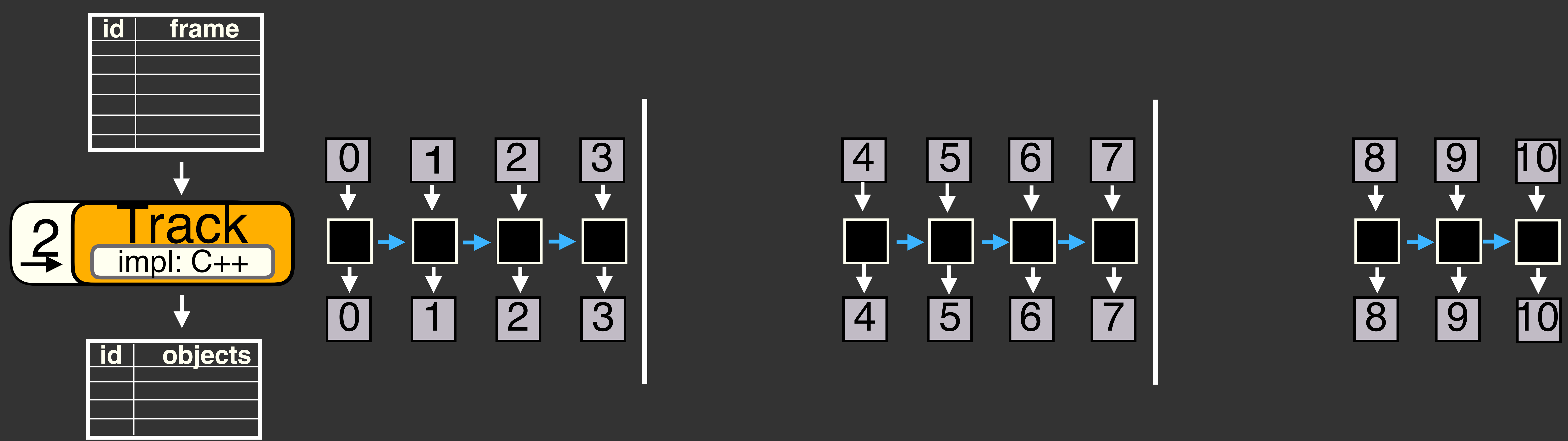
Scanner runtime schedules computation graphs onto CPU/GPU clusters



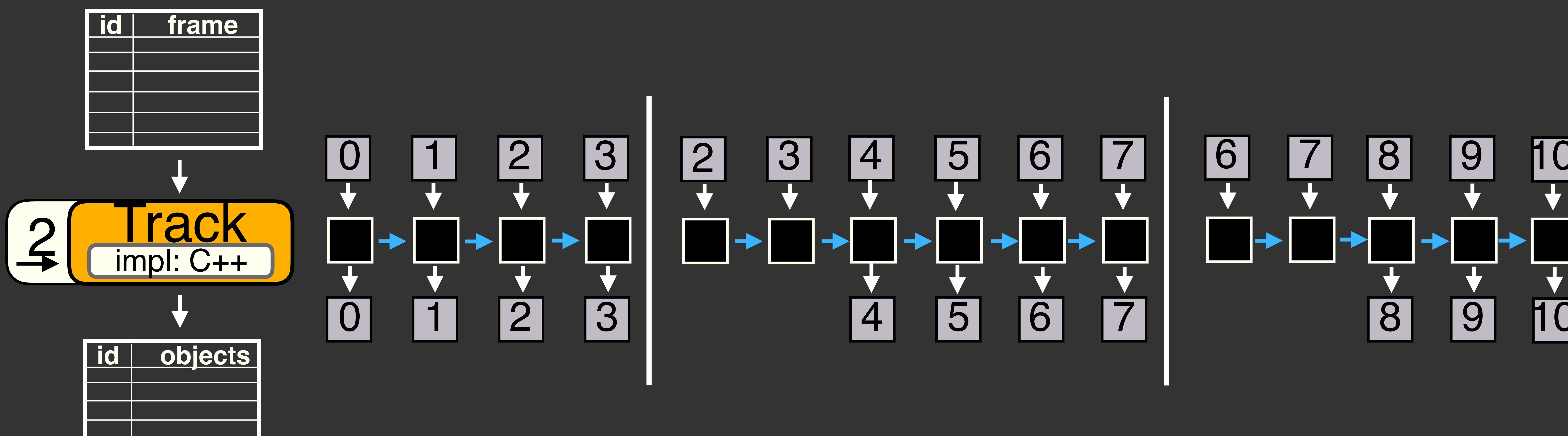
Approximating stateful processing to increase parallelism



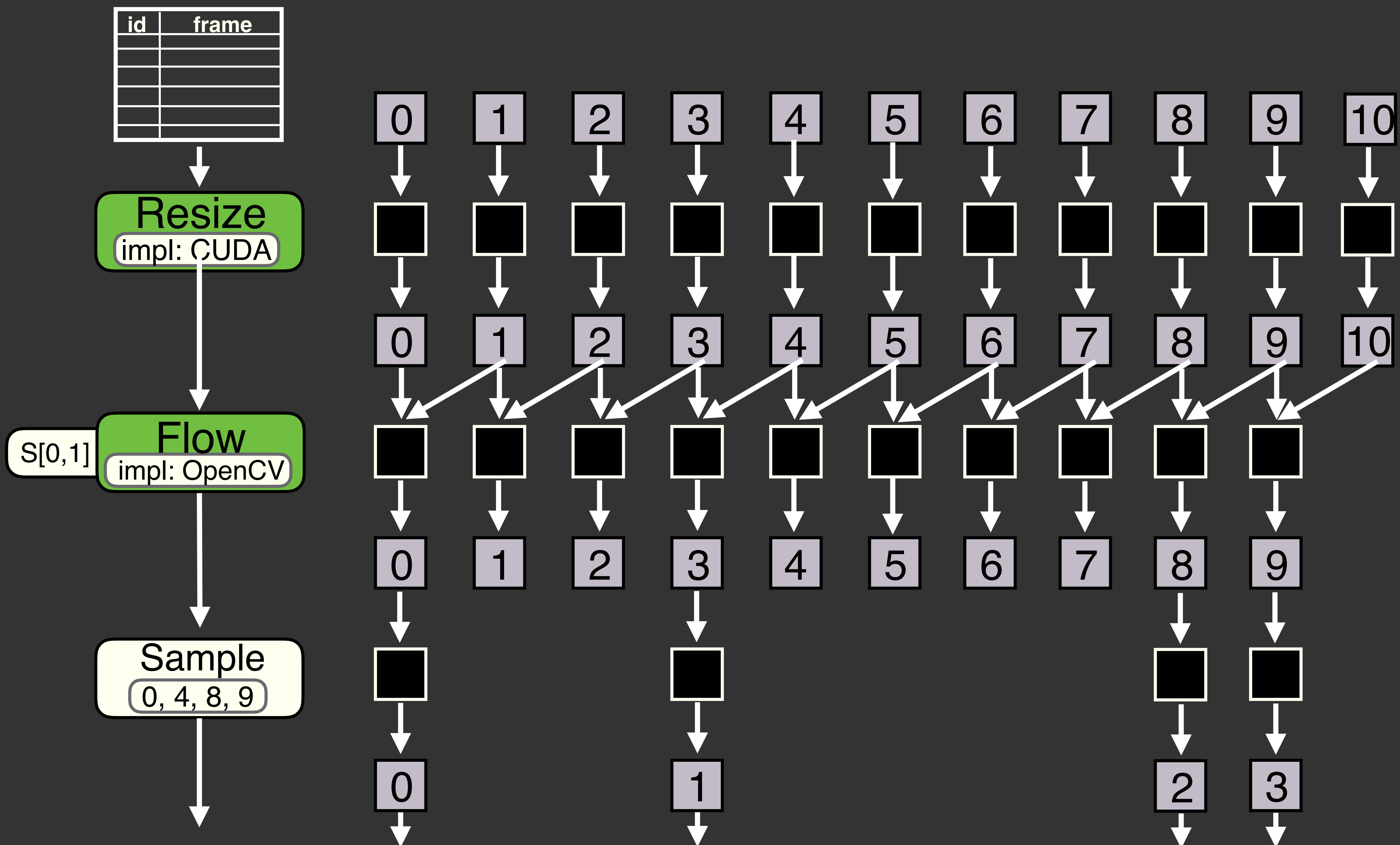
Approximating stateful processing to increase parallelism



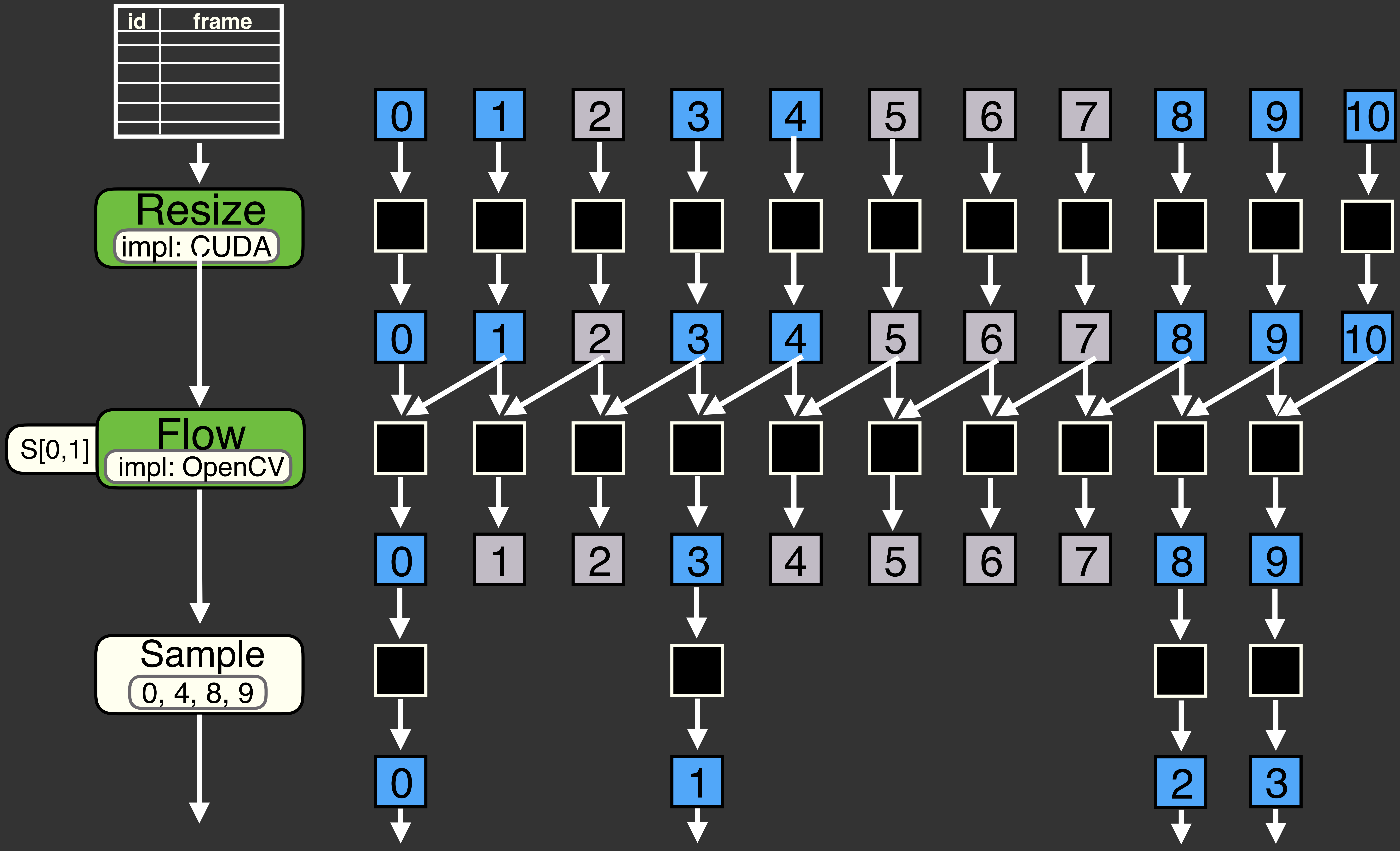
Approximating stateful processing to increase parallelism



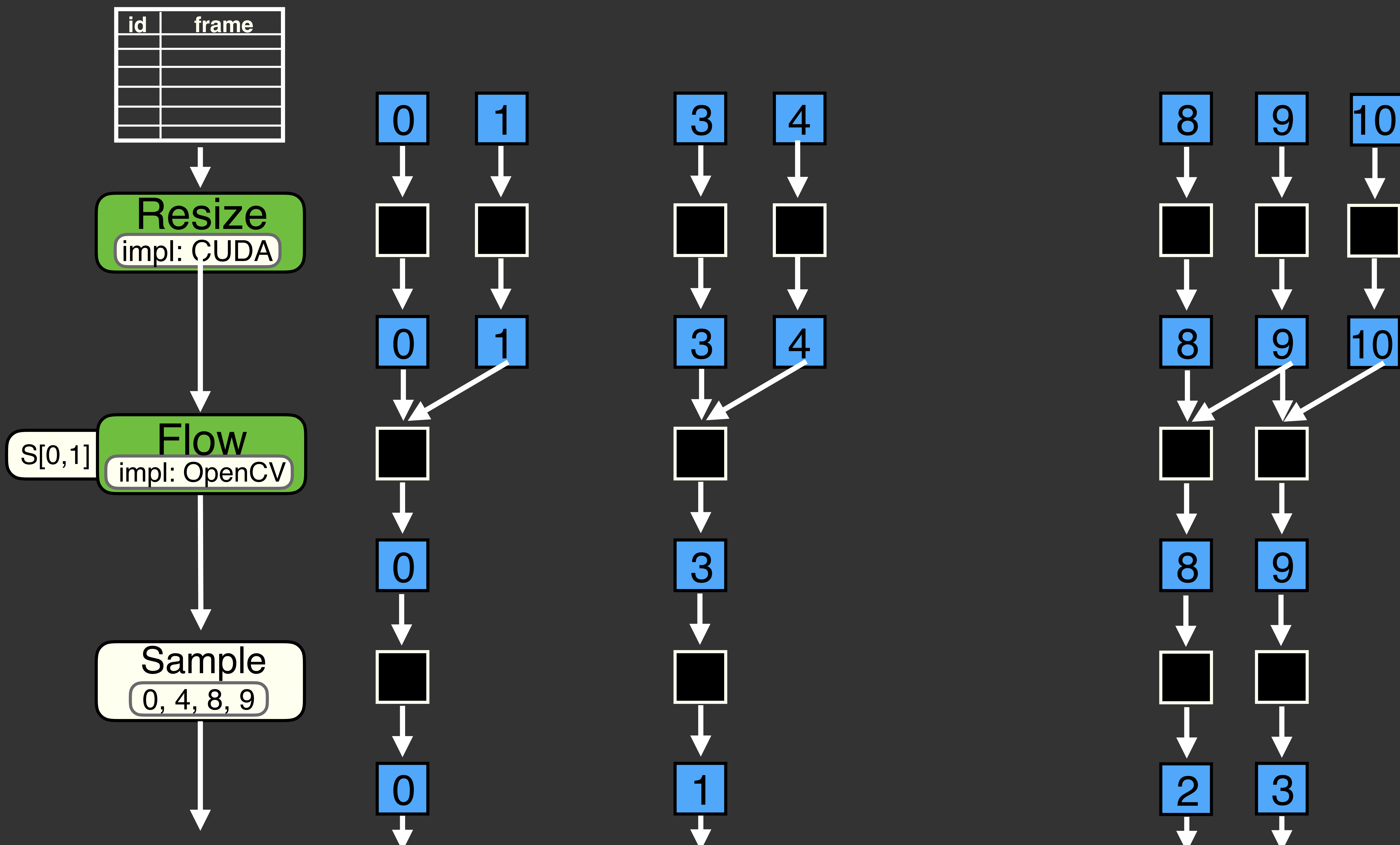
Per-element dependency analysis



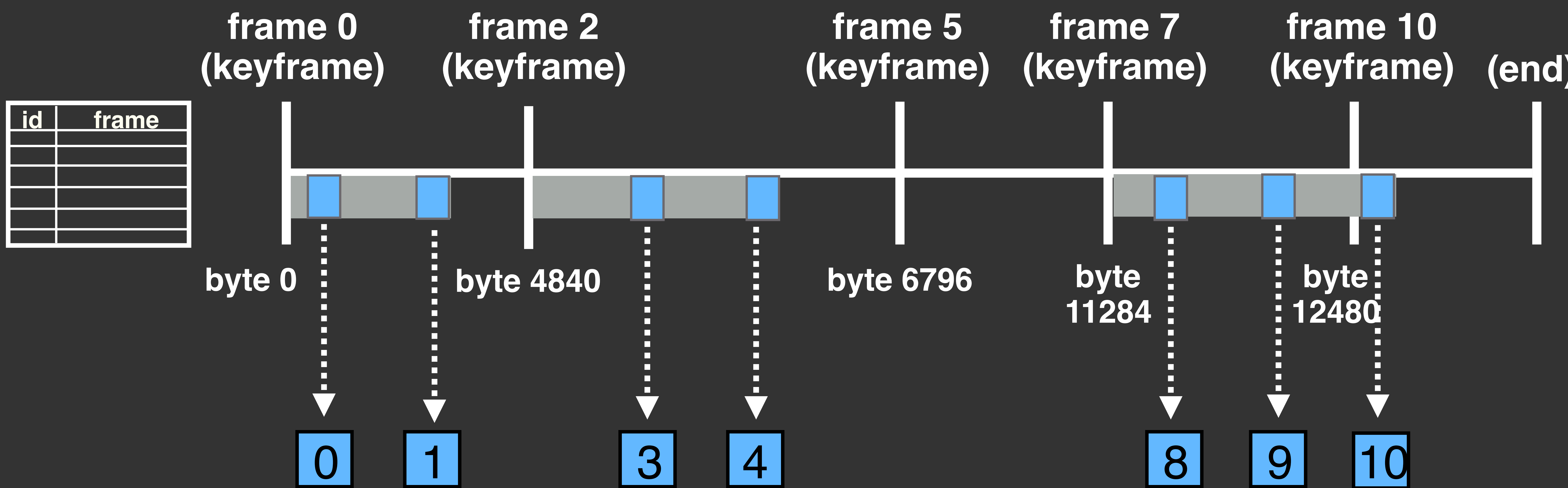
Per-element dependency analysis



Per-element dependency analysis

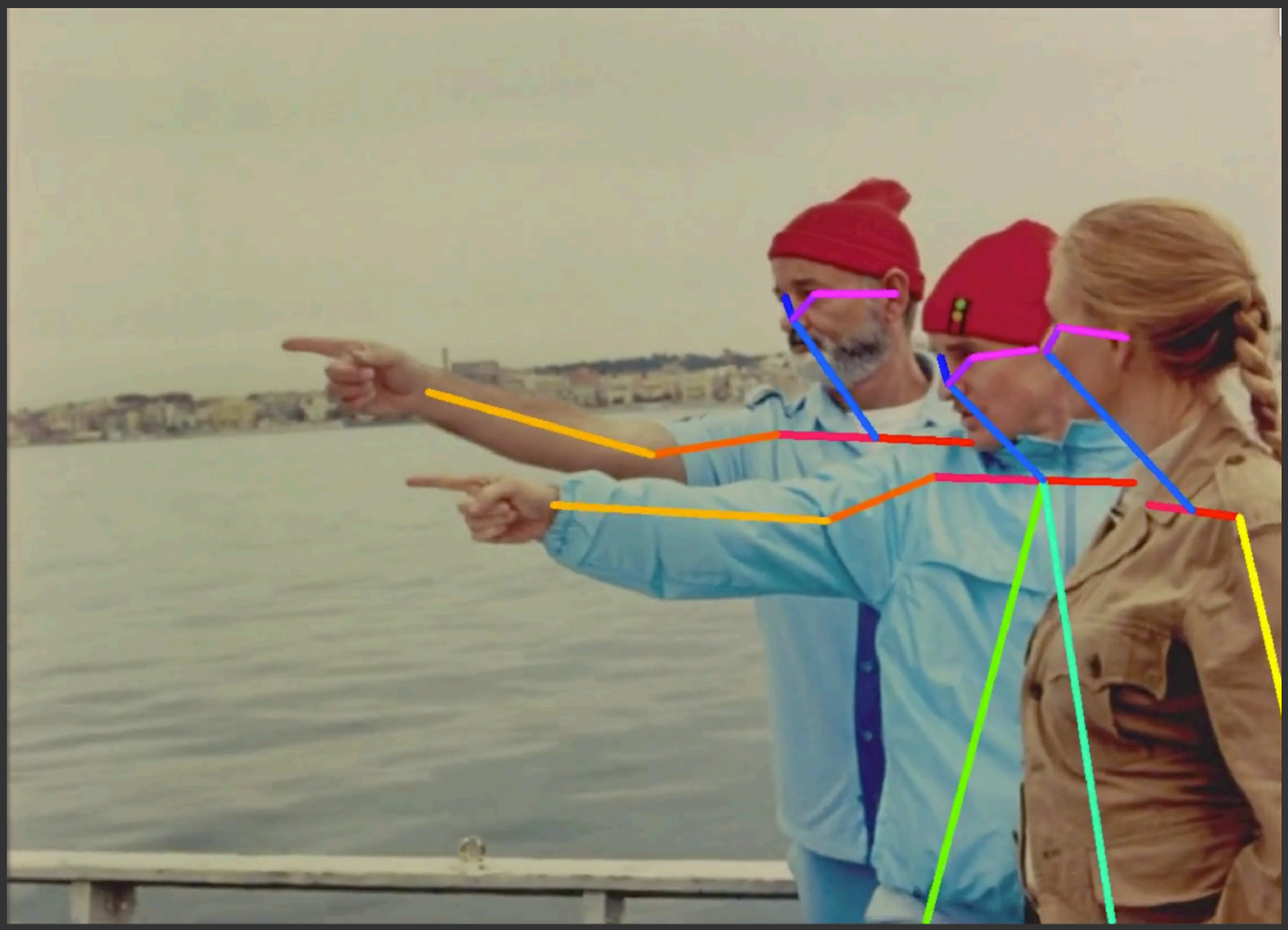


Efficient access to sparse frames



Improves decode throughput by 2 - 14x for sparse access patterns

Using many machines for quick turnaround when running inference

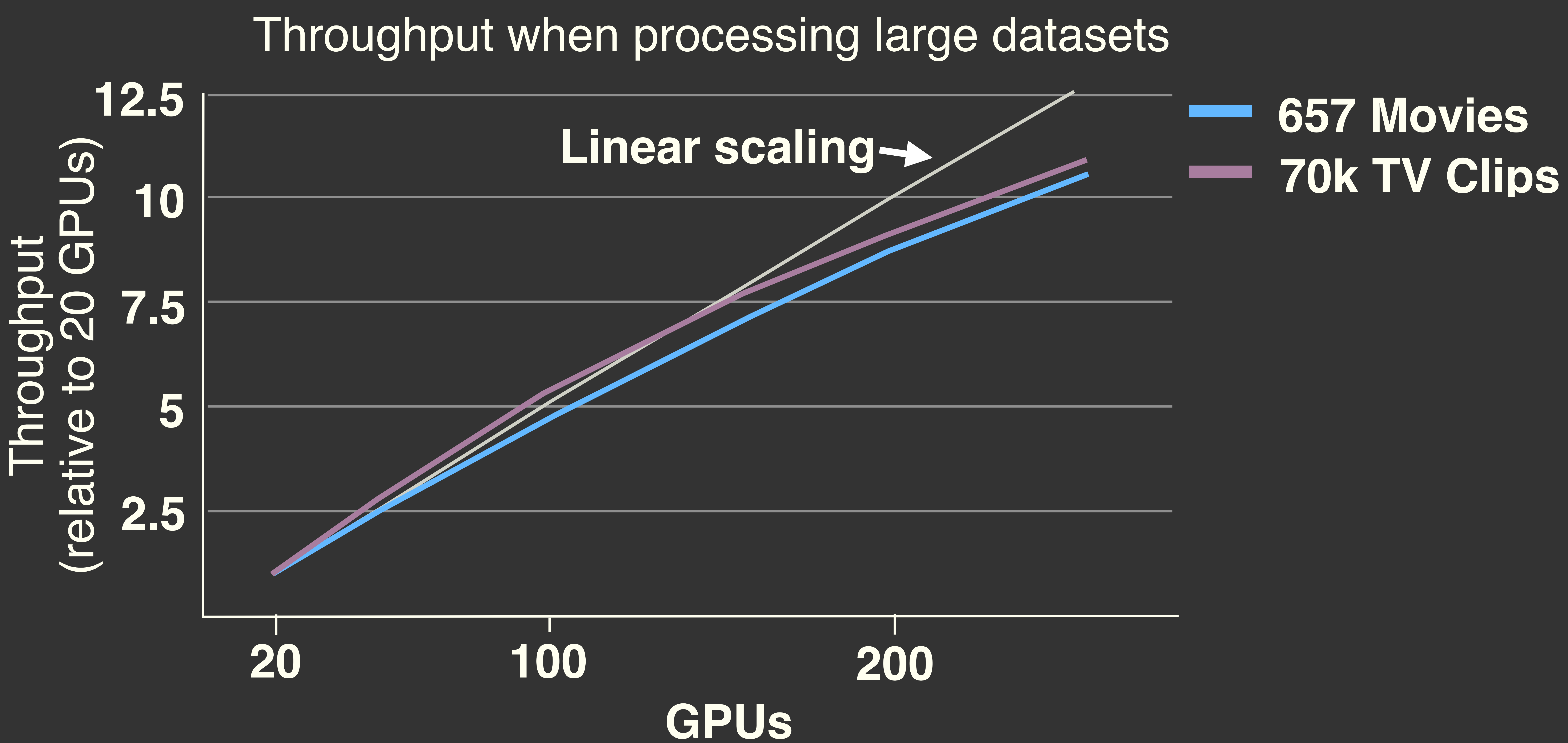


Analyzing a 2 hour 1080p movie

1 K80 GPU:	**55.3 mins**
75 K80 GPUs:	**123 secs**

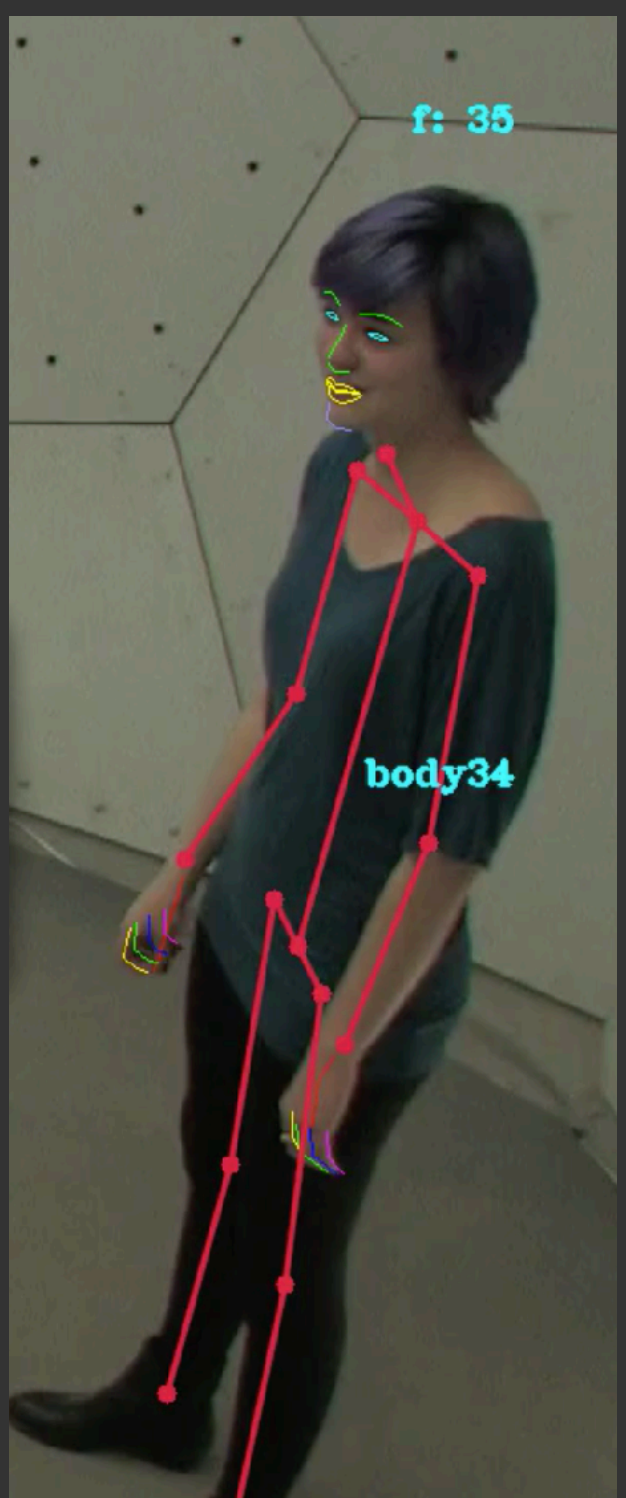
* Benchmark: running OpenPose on all frames

Scanner scales when processing large datasets



* Benchmark: running OpenPose on all frames

Accelerating the 3D human pose reconstruction pipeline



Processing 1 minute from 480 cameras

1 Titan X GPU: **24 hours**

Grad-student baseline

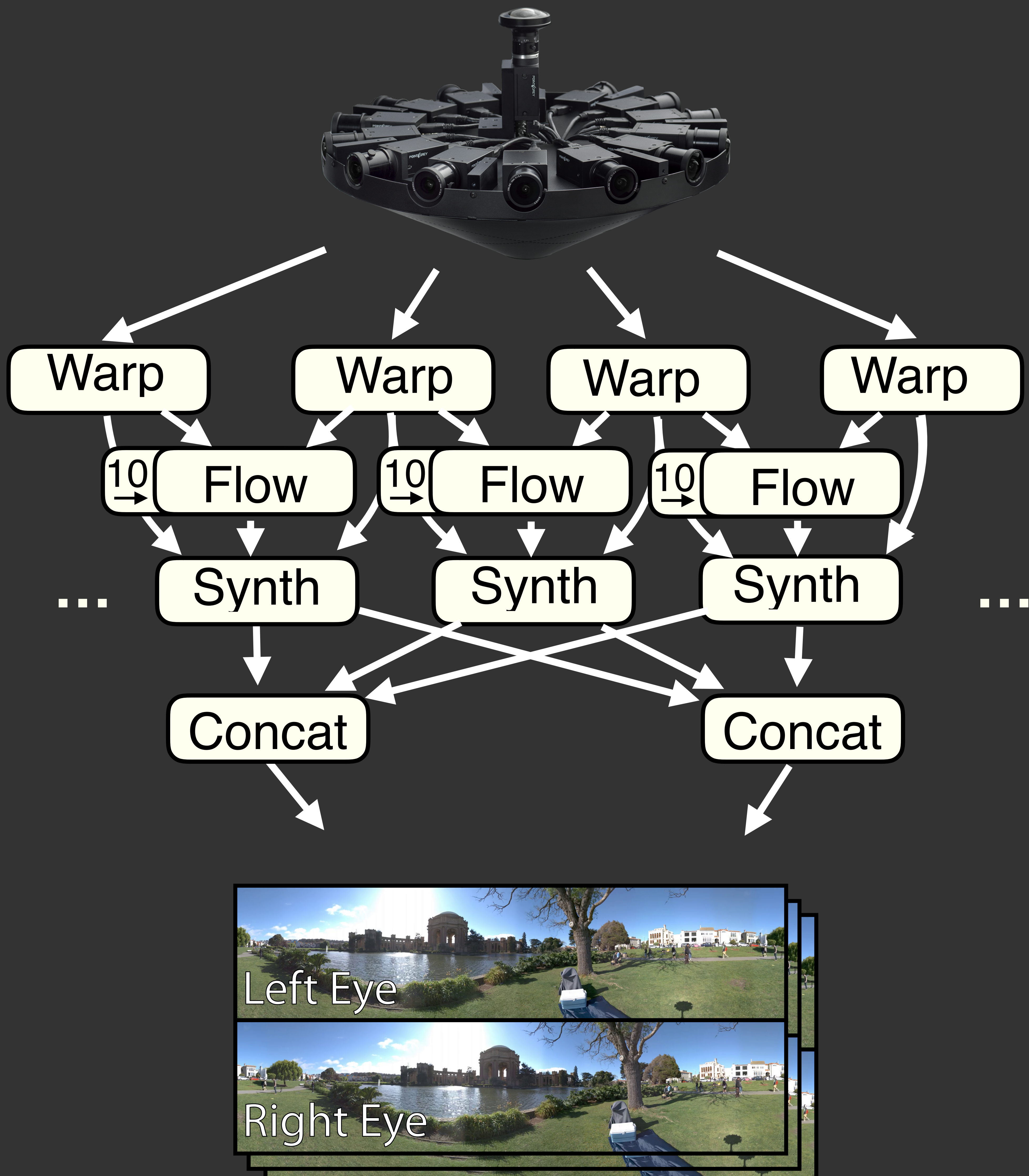
4 Titan X GPUs: **10.5 hours**

Scanner implementation

4 Titan X GPUs: **3.9 hours**

200 K80 GPUs: **37.5 mins**

Scaling Surround360 video



Processing 1 minute of video:

Facebook's Implementation

32-core CPU: **6.7 hours**

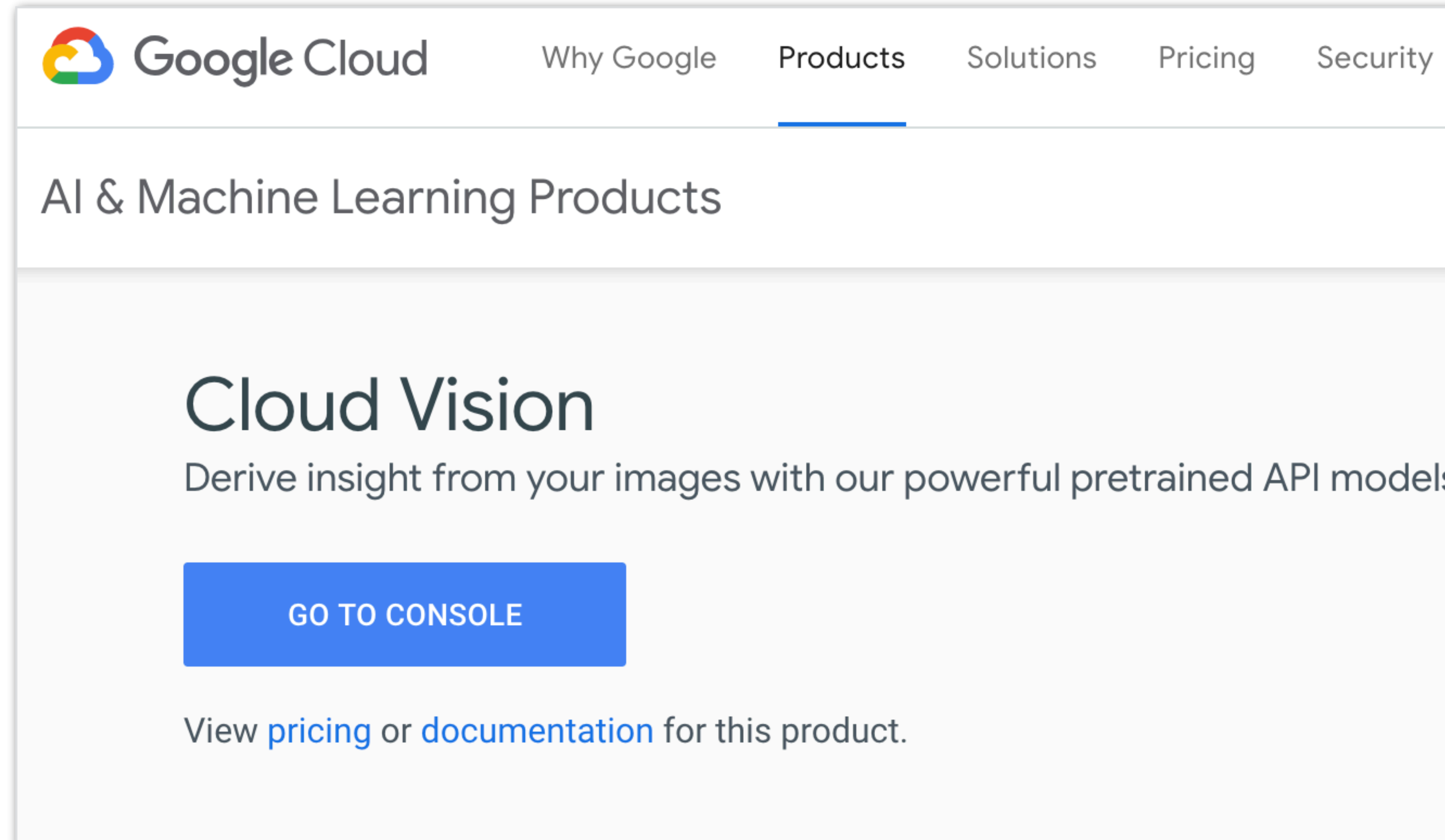
Scanner Implementation

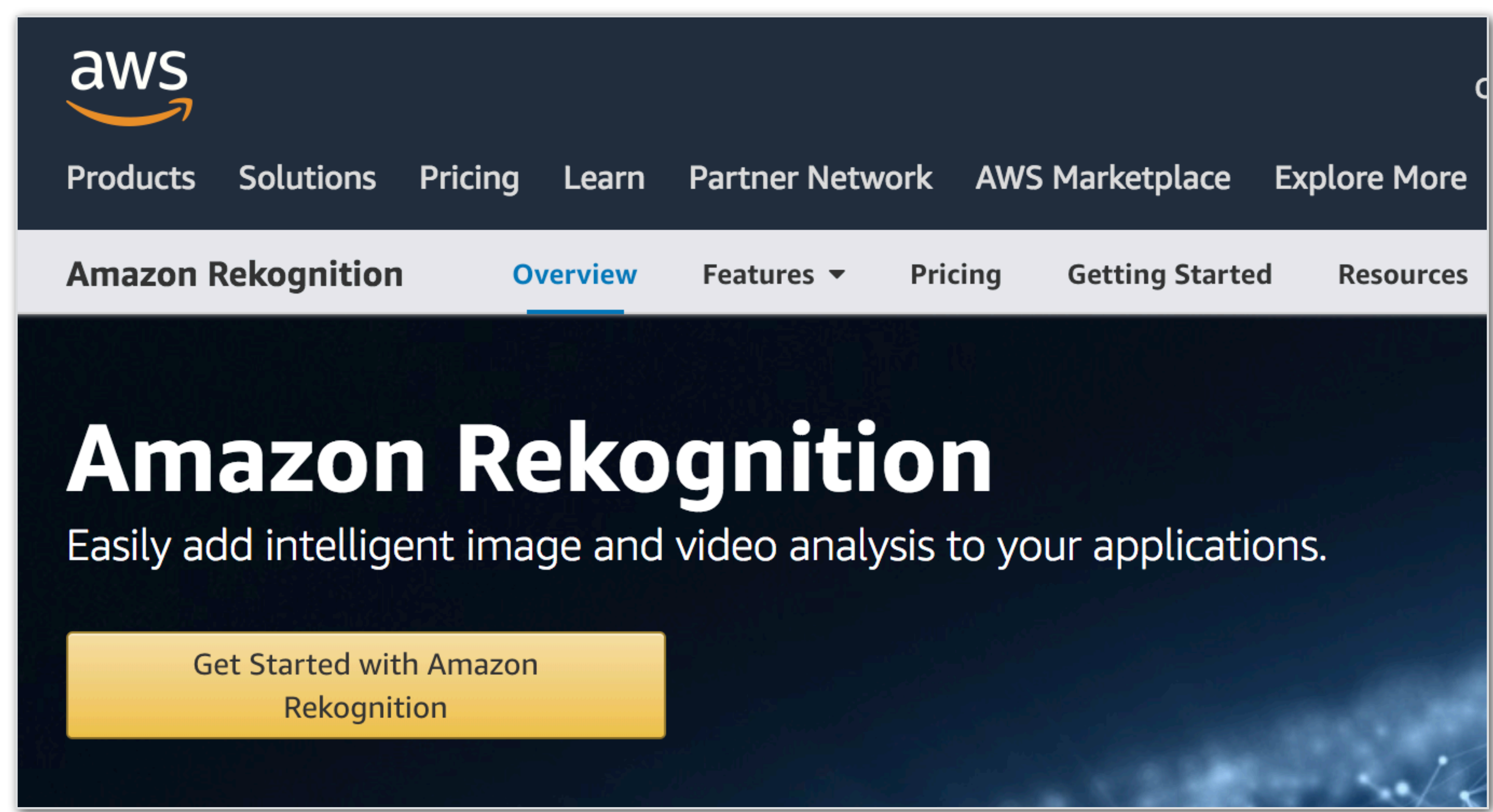
32-core CPU: **2.7 hours**

8 32-core CPUs: **18 mins**

Cloud vision services

- "Turnkey" service solutions:
  - User uploads video
  - Service returns annotations/labels





Today's summary

- Increasing interest in cloud-scale infrastructure for processing large amounts of video at scale

- Today's examples:
  - Processing many streetlight camera feeds
  - Ingest at Facebook
  - Batch processing with Scanner

- But don't forget… algorithmic innovation is always a way to do more without scaling up system size.