DEALING WITH REAL-TIME DATA IN THE IOT CLOUD

Ken Birman

CS5412 Spring 2021

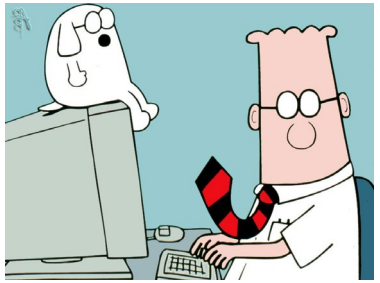
Lecture 10

THE WORLD IS GENERATING A NEW WAVE OF IOT/ML PIPELINES… THERE ARE MANY USE CASES

Satellite Images

AI/ML Layer

Smart Queries



"Which farms will be at risk of wheat blight in 2021?"

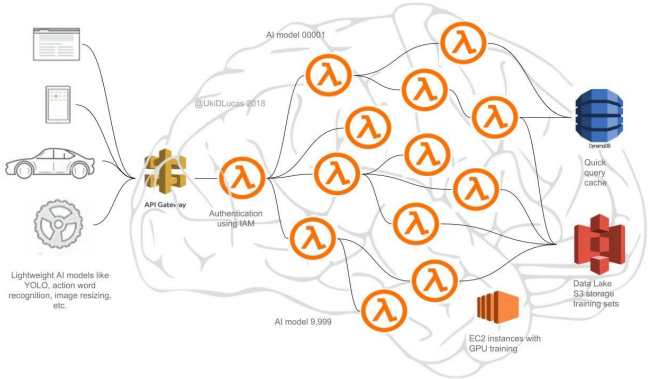
MANY OF THESE USES INVOLVE REAL-TIME DATA STREAMS

The data itself has GPS timestamps, accurate to a few ms

We may need to quickly recognize important events or patterns: like doing AI under time pressure, so that we can react quickly.

Consistency matters: We saw how easy it would be to be fooled by data staleness or timing issues introduced by the platform.

DISTRIBUTED AI



Increasingly seen in robotics, smart homes, 5G, digital twin scenarios.

- The application is a graphical collection of AI classifiers / learners
- Nodes represent computational tasks.
- Edges represent data flow between distinct tasks.

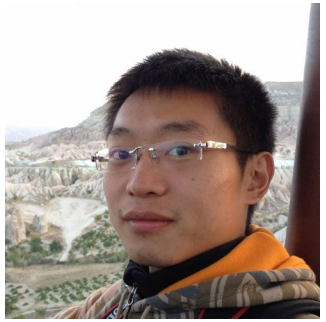
EACH λ REPRESENTS A DISTINCT AI ELEMENT

Many are parallel: Single λ may run on a pool of compute nodes.

Thus, to build a D-AI we build pipelines linking parallel tasks.

Today's cloud platforms have limited support for this model, lack the real-time and consistency guarantees needed for IoT.

Weijia Song



WE HEARD ABOUT CASCADE IN PRIOR LECTURES. THIS IS WHAT IT IS FOR!

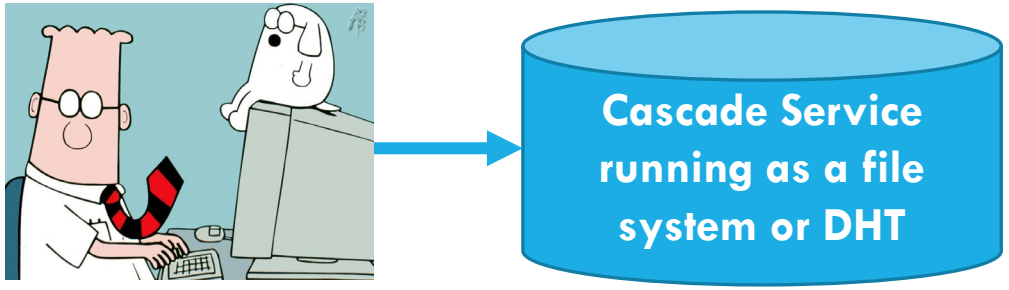
Cascade was created to be a runtime system for edge IoT that is

- Really easy to use, like Tensor Flow, Pandas, Spark, …
- Offers real-time responsiveness in addition to consistency
- Scalable – even to global deployments.

THE CASCADE MODEL

Is it a service? A library?

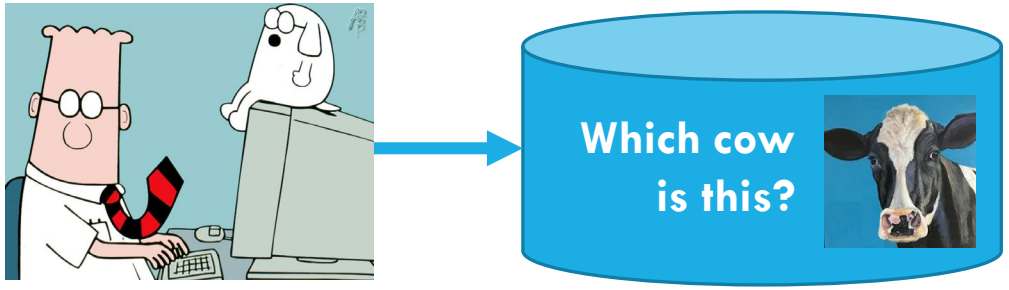
CASCADE IS A SERVICE



Although created using Derecho (a C++ library), Cascade runs on a set of nodes (machines or VMs) where it controls some resources (cores, RDMA interfaces, GPUs/FPGAs, memory).

Users can build applications that access Cascade from "outside". We call those "external clients". The put/get API would work, and RDMA is supported.

CASCADE IS AN EXTENSIBLE SERVICE



But this slide deck is mostly about users who *extend* the Cascade service by adding logic that runs inside of Cascade.

Cascade behaves as a customized service: a "smart" service.

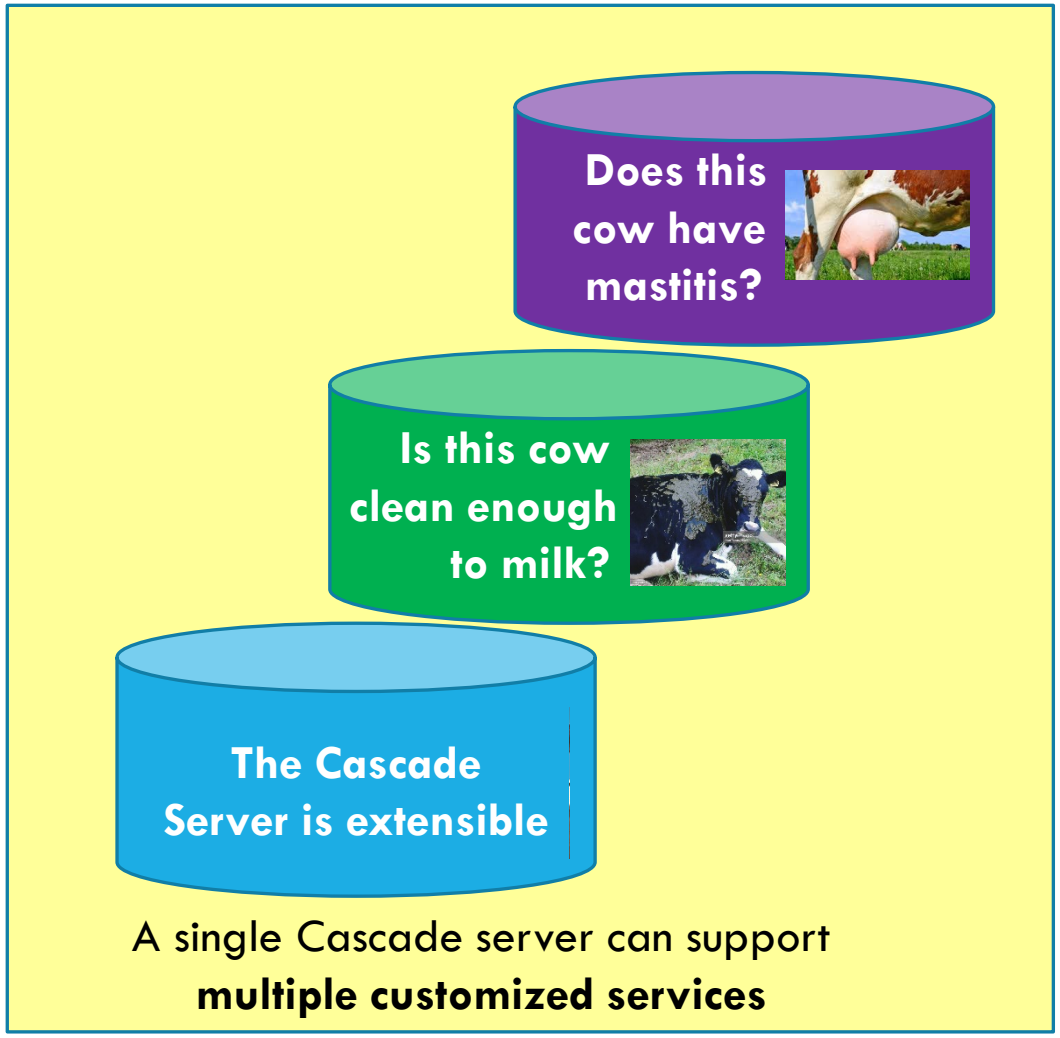
We will explain how this is done… it involves extra APIs, but allows us to also treat our service as a kind of library.

EXTENSION CONCEPT

Cascade is one service

But you supply customization that lets it act like many specialized services, one per application

So it becomes a platform for new microservices, like these!



KEY CONCEPT: EXTENSIBILITY

What does it mean to be an "extensible platform as a service"?

Think about devices you plug into your computer. For example, Ken uses a Garmin GPS to track bike rides. When he plugs the Garmin into his Dell computer, it becomes an expert on all the bike rides he has taken since 2008.

In some sense, the Garmin+Dell pair is a "new thing"

HOW DOES A PERSON EXTEND CASCADE?

We will look at this later in the class, but the idea is to plug in new software written in C++, and to tell Cascade when this software should run.

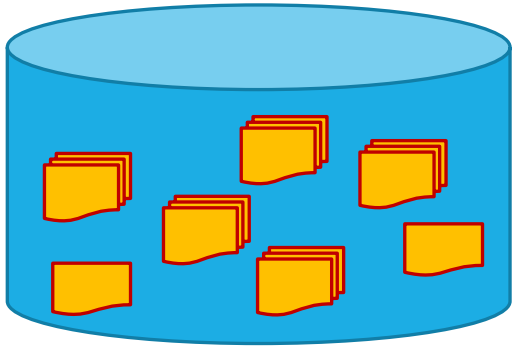
The new code handles IoT events specific to the application

For example, it can handle “cow-washing events”

THE CASCADE STORAGE MODEL

How should Cascade data be managed?

FILE SYSTEM "EXTENSION"



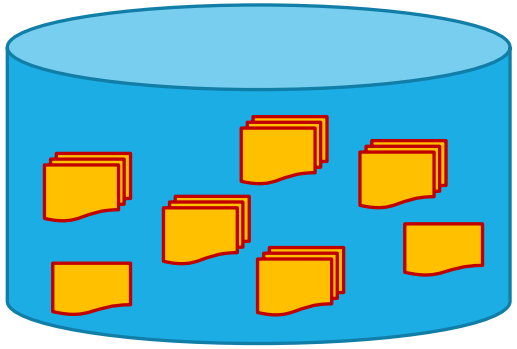
User sees scalable, "private" object pools

Cascade has a built-in file system extension:

- Every object has a pathname.
- The file system extension supports normal file operations.
- You can access it just like any file system.

Yet Cascade isn't a file system. It is a key-value **get**/**put**/**watch** store. Moreover, it is automatically sharded for scalability.

UPDATES: CREATE A NEW FILE OR APPENDING TO AN EXISTING FILE



User sees scalable, "private" object pools

Any key-value object lives in one shard (but that same shard may have many keys that map to it!).

- A *key* is a string. A *value* is an object serialized as a byte vector.
- Updates are log appends using Paxos. Each object has a log of versions that evolved over time.
- Queries run on the *stable prefix* of the log.

VERSIONED/TEMPORAL QUERIES

For volatile objects, Cascade enforces version sequencing (this is enough to ensure lock-free consistency!)

With persistent objects, Cascade also keeps a history

- By default, applications see the most current version
- But indexed access allows the application to query any version (by version number or time), or fetch any data range.

VERSIONED OBJECTS

We configure the object store to track versions. **put** creates a new version:

- *key:* The object store *always* tracks information on a per-object basis
- *version-number:* Just an integer
- *time:* If the object itself lacks a timestamp, we just use "platform" time.

Now **get** can lookup most current version, or a specific one, even by time.

The object store is optimized to leverage non-volatile memory hardware.

STORING DELTAS

Existing DHTs lack support for versioned data.

We implemented a highly optimized versioned data structure

We implement a temporal index, and cache frequently accessed data.

- A server still manages a map (since many keys map to it), but you can think of the values for a specific key as being versioned.
- Sometimes deltas are more efficient. If you have a function to compute the delta, we won't even create a new version unless you tell us to.
- Values (or deltas) are saved on NVMe & replicated for fault-tolerance.

THE CASCADE COMPUTE MODEL

Lambdas, coded in your favorite language

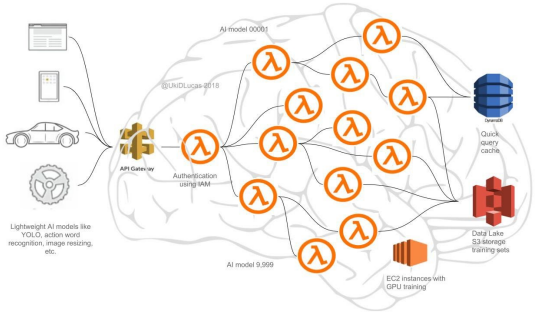
THE CENTRAL PUZZLE

The very fastest data paths require compilation, ideally in languages like C++.

But we want Cascade to run as a *service*, so it would often already be running when a new user comes along and wishes to create and launch some completely new service.

How can we extend a running system? Actually… it isn't so hard

FIRST QUESTION: WHAT'S IN A λ?



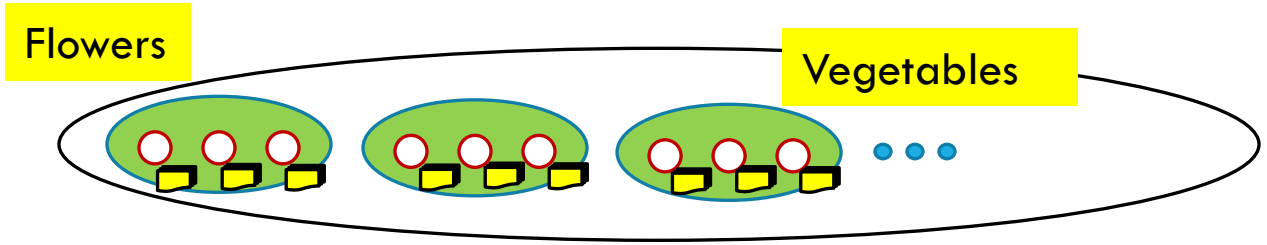
We support many languages. Native APIs are Python with various packages (including LINQ) and C++ with LINQ.

Code is concise – LINQ pioneered a style that mixes "kernel" invocations with embedded SQL. Maps cleanly to GPU, FPGAs.

Cascade manages GPUs and can cache data in GPU memory.

NEXT: WHAT IS A KEY-VALUE STORE?

We run Cascade on a set of nodes. Here we see nine nodes in three shards.



A shard identically replicates (key,value) tuples, using Paxos.

Here, an object with the key "Flowers" was stored in shard 0. "Vegetables" ended up in shard 2.

HOW DOES "WATCH" WORK?

On a given Cascade server node, it will issue an upcall to user-specified code if the key(s) the user wants to watch change.

Cascade's name space is best understood as a global file system namespace. The keys are the file pathnames.

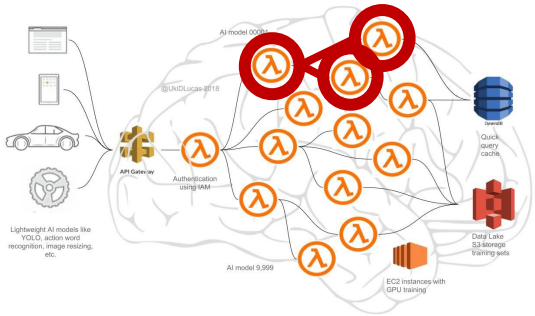
Watch thus is monitoring a file or directory…

SO… A λ IS JUST A PROGRAM DESIGNED FOR PARALLEL EXECUTION INSIDE A KEY-VALUE STORE

Our idea:

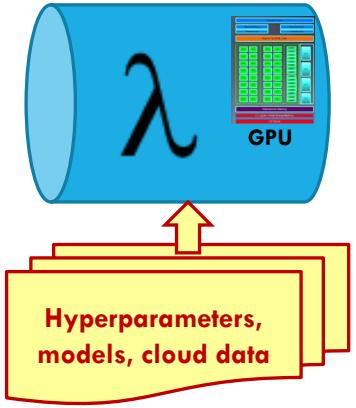
- Cascade hosts the key-value data (or file system, like Ceph)
- The user's code is treated like a dynamically linked library.
- The user creates this DLL, saves it into Cascade, then tells us where to run it. Cascade loads and launches it there.
- DLLs have zero overhead, once loaded. So now the user's logic is efficiently callable from Cascade!
- And we use the **watch** feature to initiate those upcalls!

CREATING AND INSTALLING A NEW λ ON SOME CASCADE NODES



Developer builds a new ML program, designed for parallel execution directly "in" a key-value store.

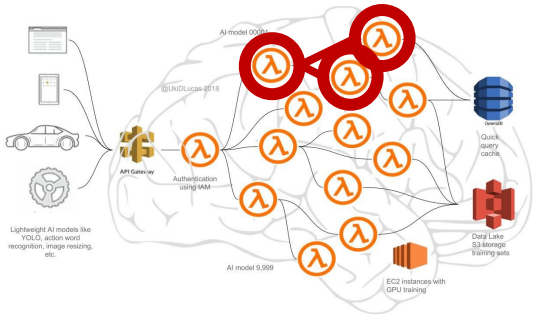
Cascade is already running in the cloud. She tells Cascade to load this DLL.



On the designated compute nodes, Cascade loads the DLL and activates it. The DLL initializes itself and register some "watch" upcalls.

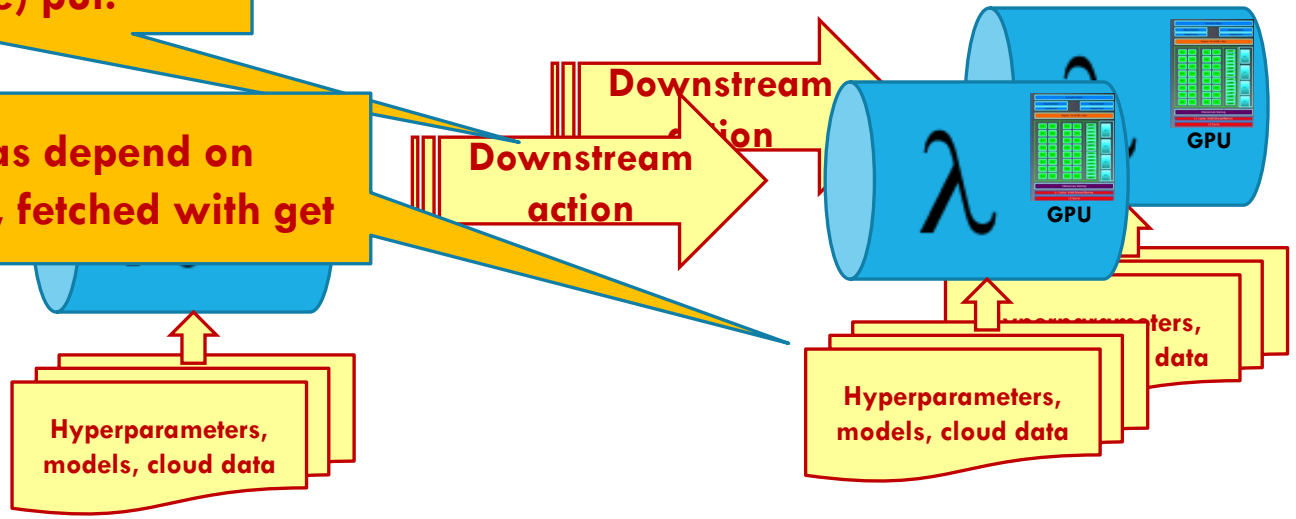
Recall that a key-value store is sharded. Each node will host a different set of keys.

D-AI PIPELINES WILL BE COMMON



One way to trigger a lambda is with a (key,value) put.

Many lambdas depend on persistent objects, fetched with get



Each of these lambda stages potentially runs on a pool of machines.

ANYTHING, ANYWHERE, ANY TIME

With permissions, any code can access any object, or the associated time-series if the object is a persisted history.

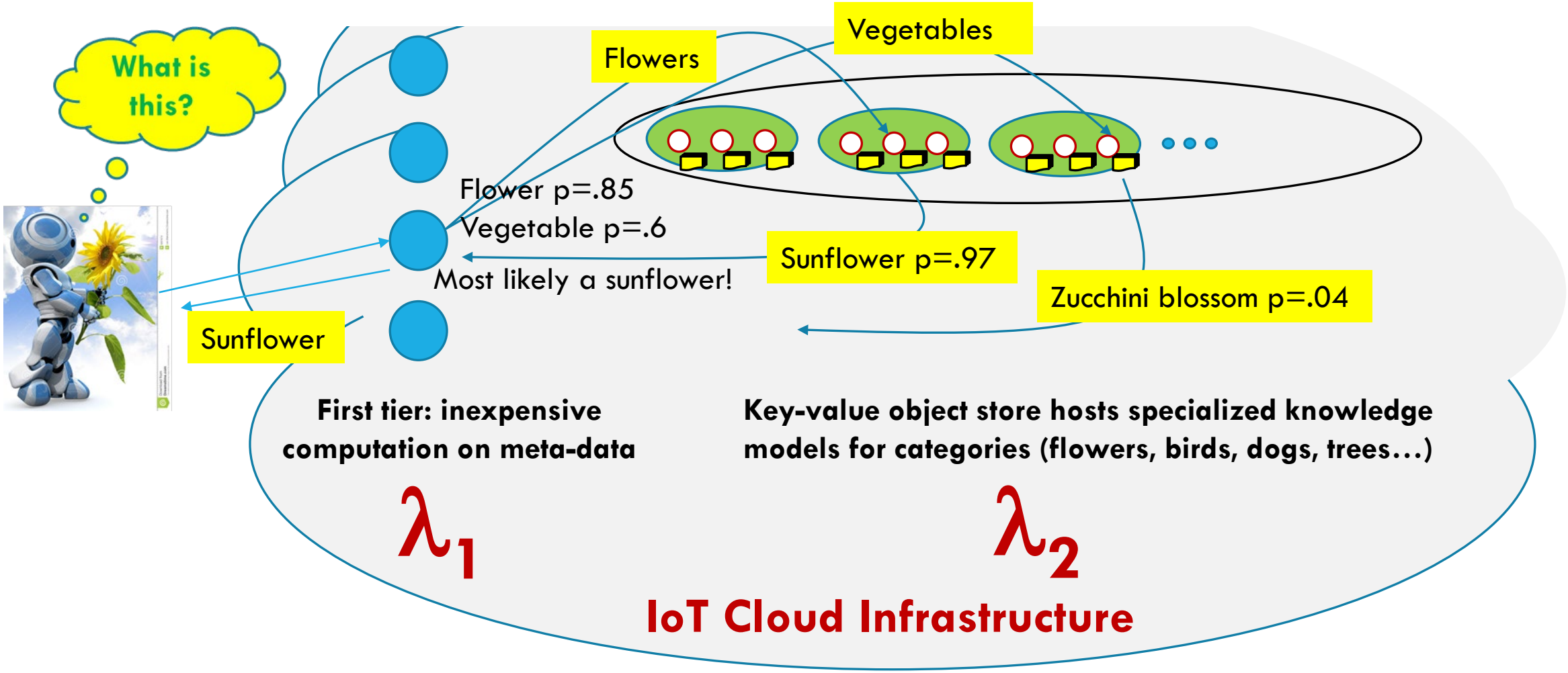
Obviously, performance is best if we can minimize data movement and compute instantly when new events occur.

This all yields a sophisticated programming model

WHAT DOES A REAL APPLICATION LOOK LIKE TODAY?

Example, courtesy of Weijia, Alicia and Thompson

A VERY SIMPLE EXAMPLE



VALUE OF CONSISTENCY

Why does it matter?

CASCADE λs EXECUTE IN CONSISTENT CUTS

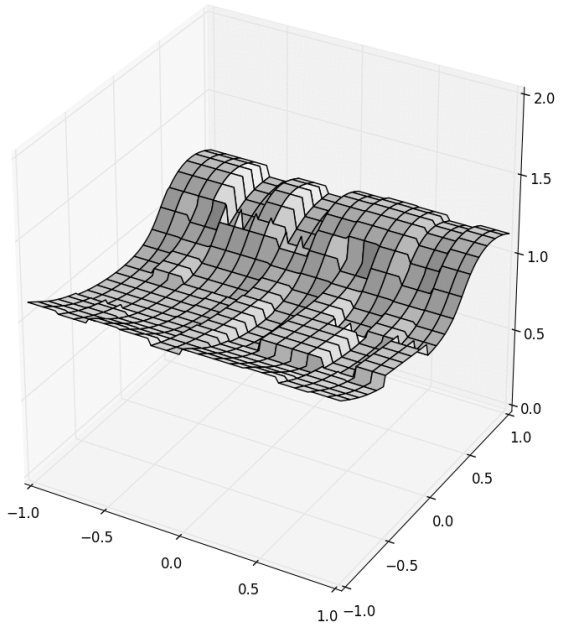
Recall: Cascade has a built-in temporal indexing feature. Suppose our distributed AI is triggered by event ε at time τ.

We run all the lambdas triggered by ε along a consistent cut "optimally close" to time τ (and selected deterministically).

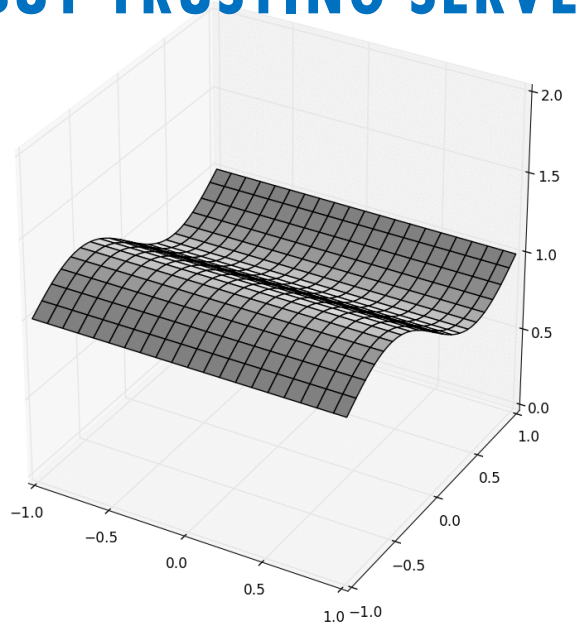
Effect: The lambda won't see platform-induced inconsistencies.

VISUALIZATION OF CASCADE CONSISTENCY

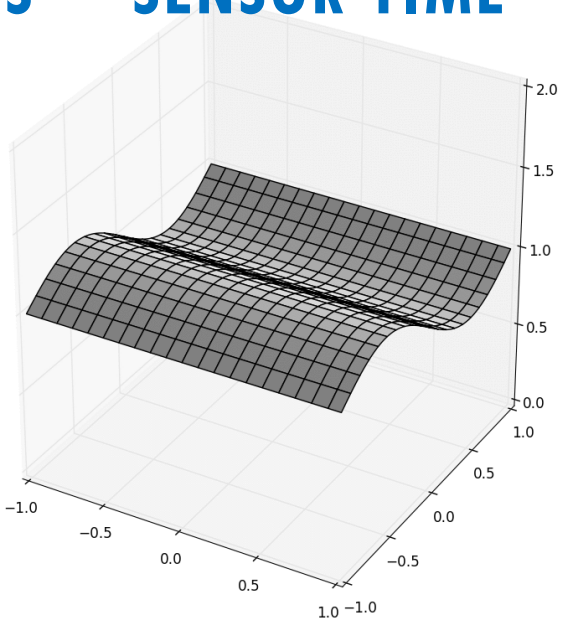
HDFS



CASCADE USING CONSISTENT CUTS BUT TRUSTING SERVER CLOCKS



CASCADE WITH SENSOR TIME



Cascade consistent cuts + GPS-timestamped sensor data result in clean input to the D-AI algorithm (in this case, a simple visualization)

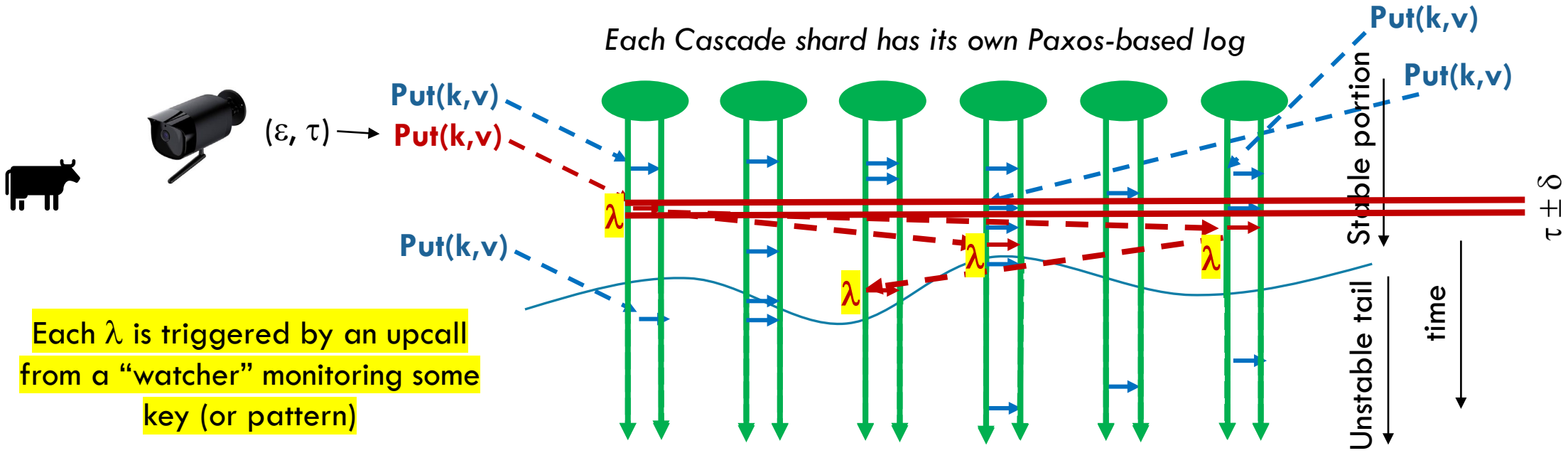
CENTRAL CONCEPTUAL INSIGHT

One event may trigger many lambdas.

These lambdas may need to run on multiple nodes… yet will share the same temporal index (τ from the trigger event ε).

A Cascade query always sees a "consistent state snapshot."

VISUALIZING THE CONSISTENCY MODEL

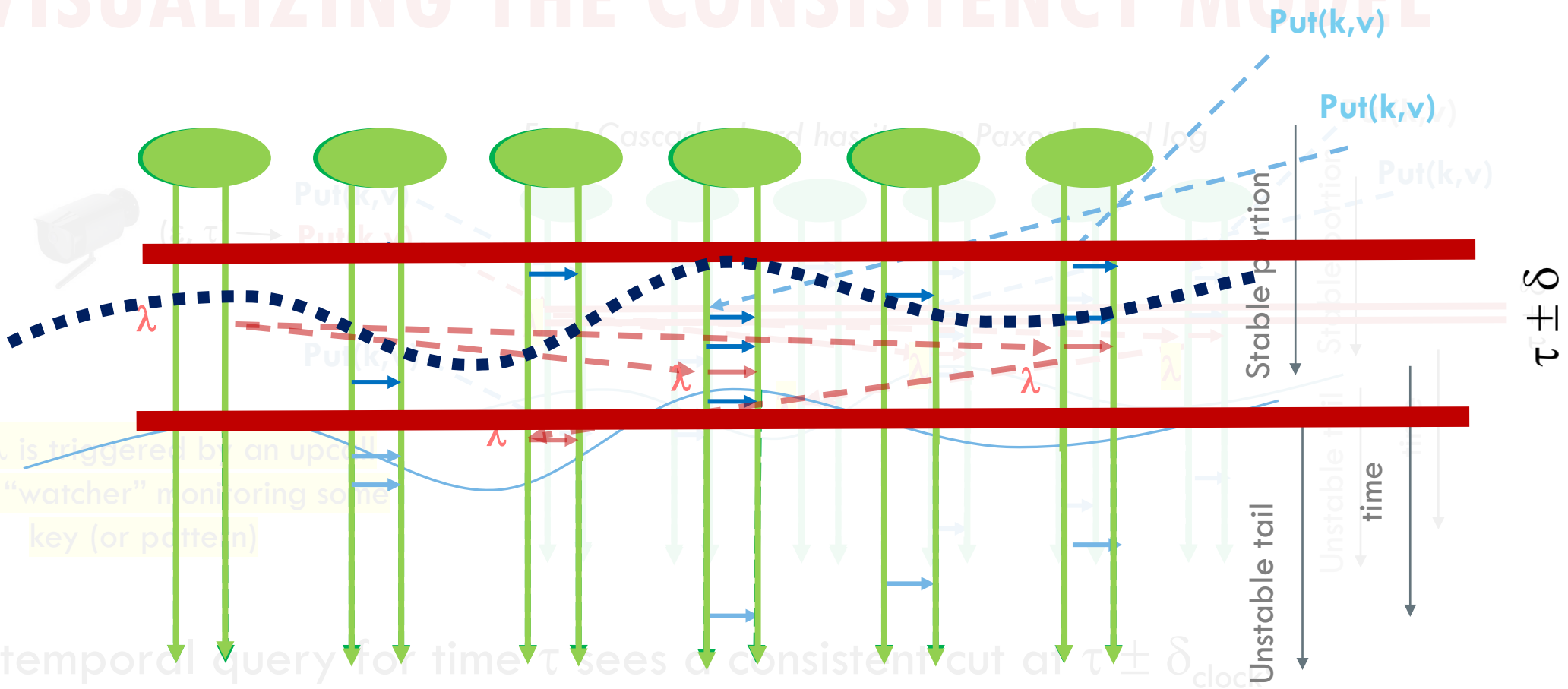


Each λ is triggered by an upcall from a "watcher" monitoring some key (or pattern)

A temporal query for time τ sees a consistent cut at $\tau \pm \delta_{clock}$.

Queries to unstable data must wait, but updates are stable within 50us.

Even if δ is small, there could be a few events at each process in the $t \pm \delta$ time window. By selecting a consistent cut, Cascade avoids the "maskup" issue we saw in the HDFS animation.

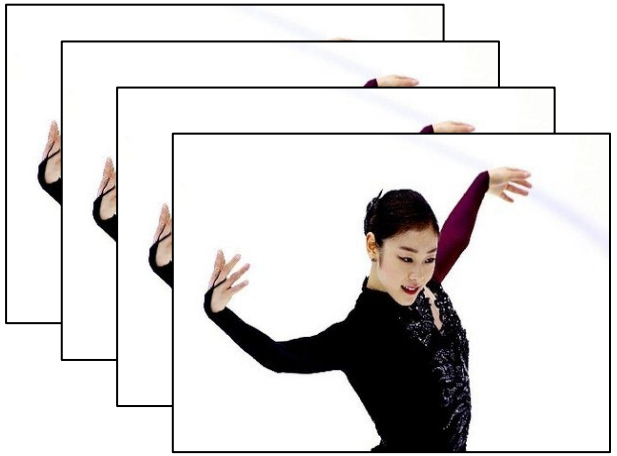


Each λ is triggered by an update from a "watcher" monitoring some key (or pattern)

A temporal query for time τ sees a consistent cut at $\tau \pm \delta_{clock}$

Queries to unstable data must wait, but updates are stable within 50us.

BUT THE JOB IS STILL HARD!

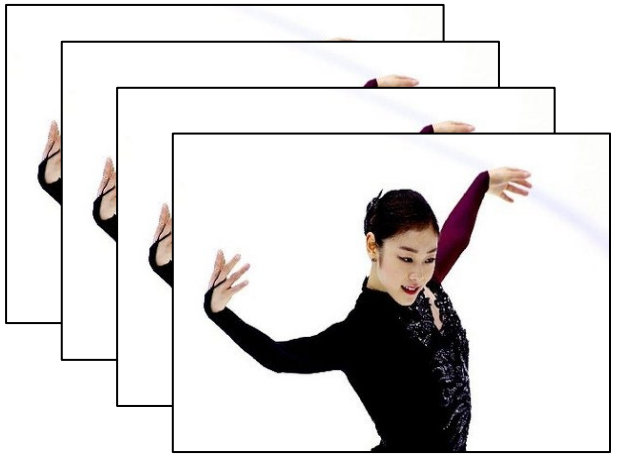


Cascade makes it easy to extract a *tensor* with a temporal dimension: A stack of frames

But writing ML code that can recognize patterns over time is not easy! Even identifying movement trajectories is a hard vision task.

Cascade is just the platform. You need to write the application!

COOL OBJECT ORIENTED IDEA



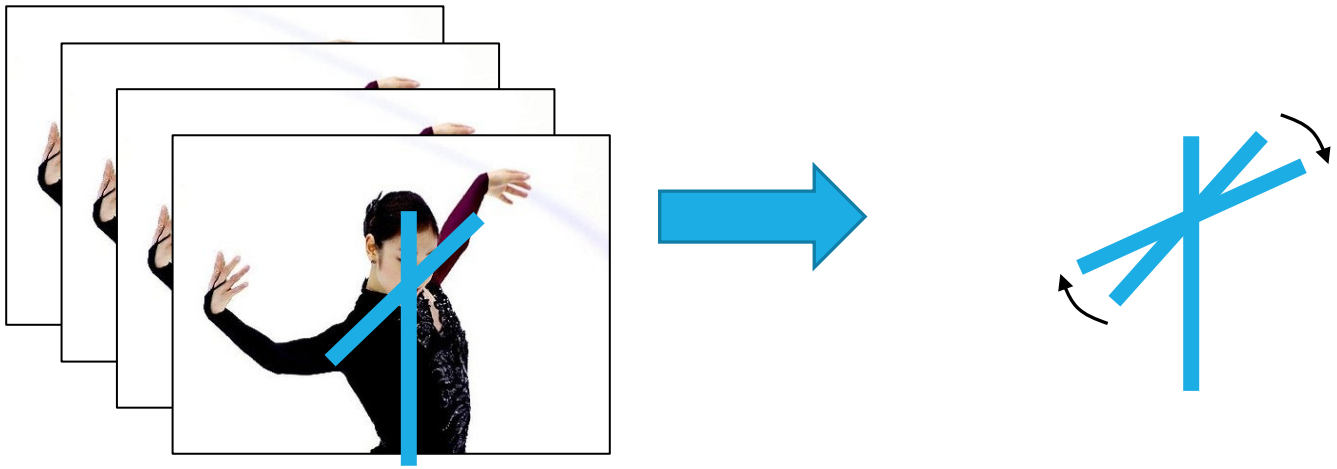
Suppose that you have a device that captures images of size w,h and you want to form a 3-D tensor for times t0 .. t1.

You can define your sensor as an object with *getter* methods that turn around and fetch the appropriate images. Now your code is written in terms of my_tensor[x,y,t] and yet Cascade handles fetching and caching the data.

You can even use LINQ to do this in one line of Python or C++ code (we will see this later in the semester)

COOL OBJECT ORIENTED IDEA

Now, if you have a computer vision algorithm that can recognize the orientation of the skater frame by frame, you can write a function that will "represent" the pattern of how her body is spinning over time.



"MACHINE LEARNING" A SPIN

In this model, we can think of a trajectory as the "motion trace" of some key points such as the skater's hands, arms, face, etc.

Each traces out a path in time and space.

In effect, our system is learning a collection of high dimensional splines that fit the observed data, and can be used to predict future movement

REALISTIC "USE CASES?" FOR SKATING

A skating judge might be interested in measured properties of the spin, like speed, number of spins, steadiness.

A coach might be trying to diagnose the root cause of a small wobble.

The skater may be wondering what would have happened if her left hand was just a tiny bit higher

THIS IS WHY WE NEED A GRAPH OF AI'S – A D-AI

Each lambda in such a graph would "own" a distinct AI task, perhaps a substantial one. The lambda might be just a few lines of code, but it could also be a big program, or could be invoking sophisticated ML logic that came from prebuilt libraries.

The entire solution would embody a kind of edge machine intelligence, whereas each individual lamba is a narrow specialist in just one aspect of the problem.

HOW TO CREATE A NEW λ

Cascade is an *extensible* service!

TRIGGERED ACTIONS: THE CODE ITSELF

The lambda is created as code that implements a static API.

- User places the DLL (or the source file) in a Cascade object
- A command tells Cascade which nodes should load it.
- The DLL has an **initialize** method. Cascade calls it.

Notice that the user-supplied code can define new object types. We only require that all Cascade objects be byte-serializable.

TRIGGERED ACTIONS: API IS SIMPLE

Cascade implements has three primary APIs

- The main Cascade APIs: (key,value) **put** and **get.**
- **watch** upcalls to trigger user logic when some key is updated.

Watch can do exact key match, or path-prefix match, or arbitrary regular-expression matches.

TRIGGERED ACTIONS: THE CODE ITSELF

The **initialize** method calls **watch** to register the user's lambdas

- A lambda is a *closure*. This associates "context" with the lambda.
- Example: keys identifying hyperparameter and model objects.

Watch monitors for keys that match.

If a match occurs, Cascade passes the matching key to the lambda.

PATTERN MATCHING

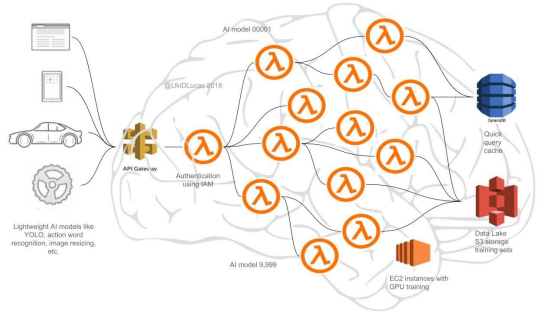
Just like with file names (key == file name)

/data-center/instance/users/Alicia/SmartDairy/ImageRec/cow2716/model

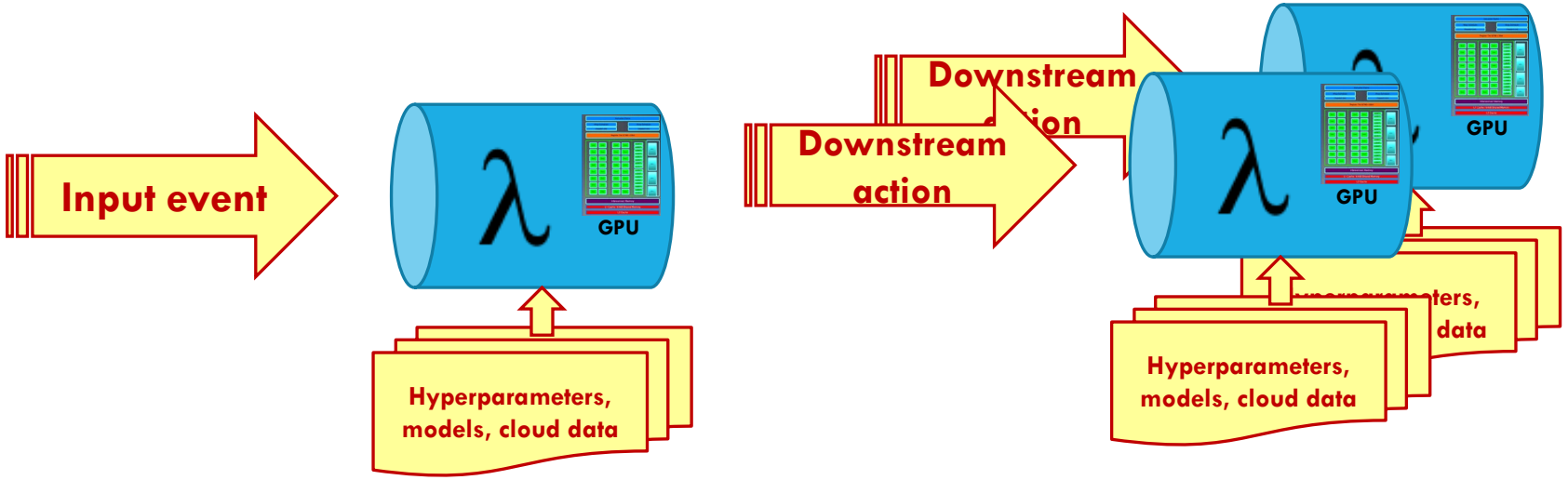
Many watch patterns will seek exact match.

Some are a prefix followed by a glob-style pattern, like “trigger if any change occurs in this folder”

PERFORMANCE CHALLENGE

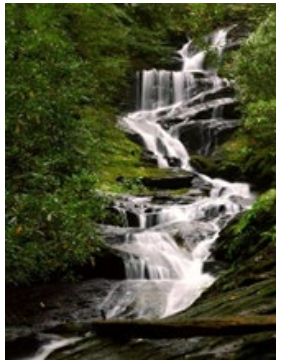


Recall… a D-AI will be a complex pipeline of triggered logic:



How <u>fast</u> can these triggered actions be initiated? How close to perfect wire efficiencies and <u>zero extra delay</u> can we get?

INSIDE A SMART SERVICE



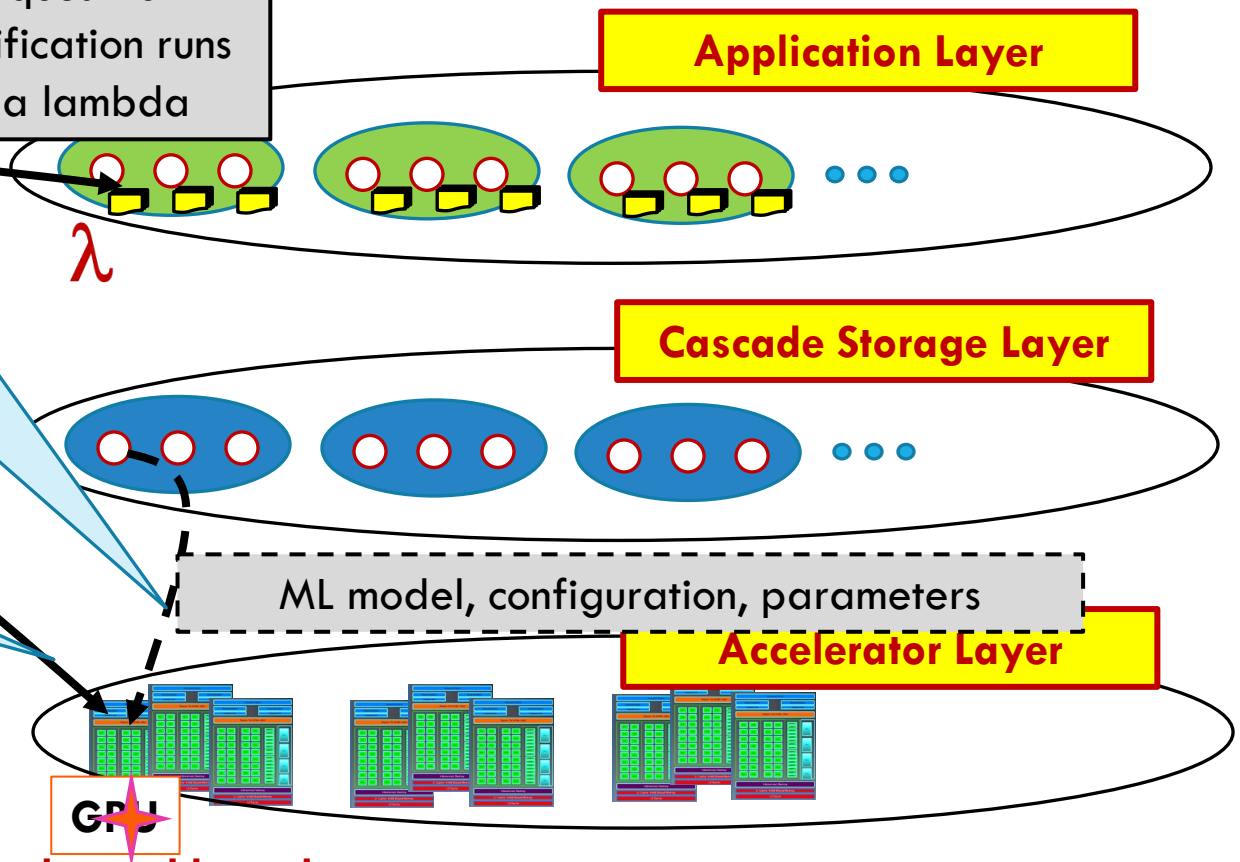
Skip this transfer for any ML objects already cached in the GPU device!

Request for assification runs as a lambda

We really want the
to be "ideally" positio or
copied directly to the pe t

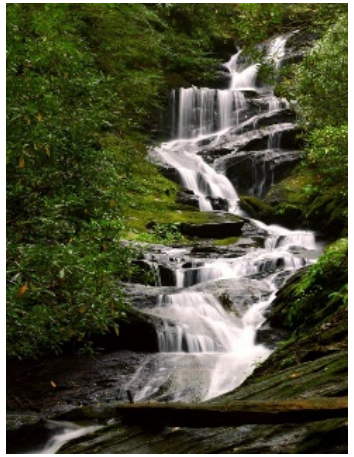
RDMA directly into GPU memory

Image to classify



GPU-accelerated kernel initiated from the lambda

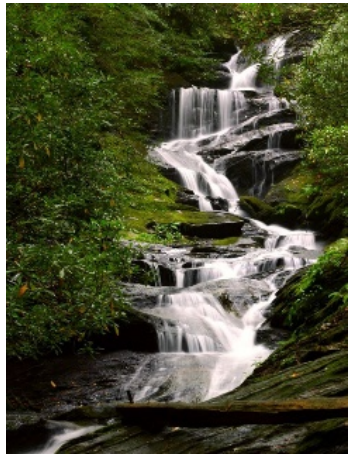
STATUS AND OPPORTUNITIES



The basic Cascade system (in one data center) is up and working, as an application built on our Derecho system (ACM TOCS 2019). Blazingly fast… SIGCOMM submission describes some of the most impactful optimizations relative to the original TOCS version.

But this idea of extending into distributed AI is new. Our big project is to fully implement the whole picture.

WHAT ARE WE DOING RIGHT NOW?

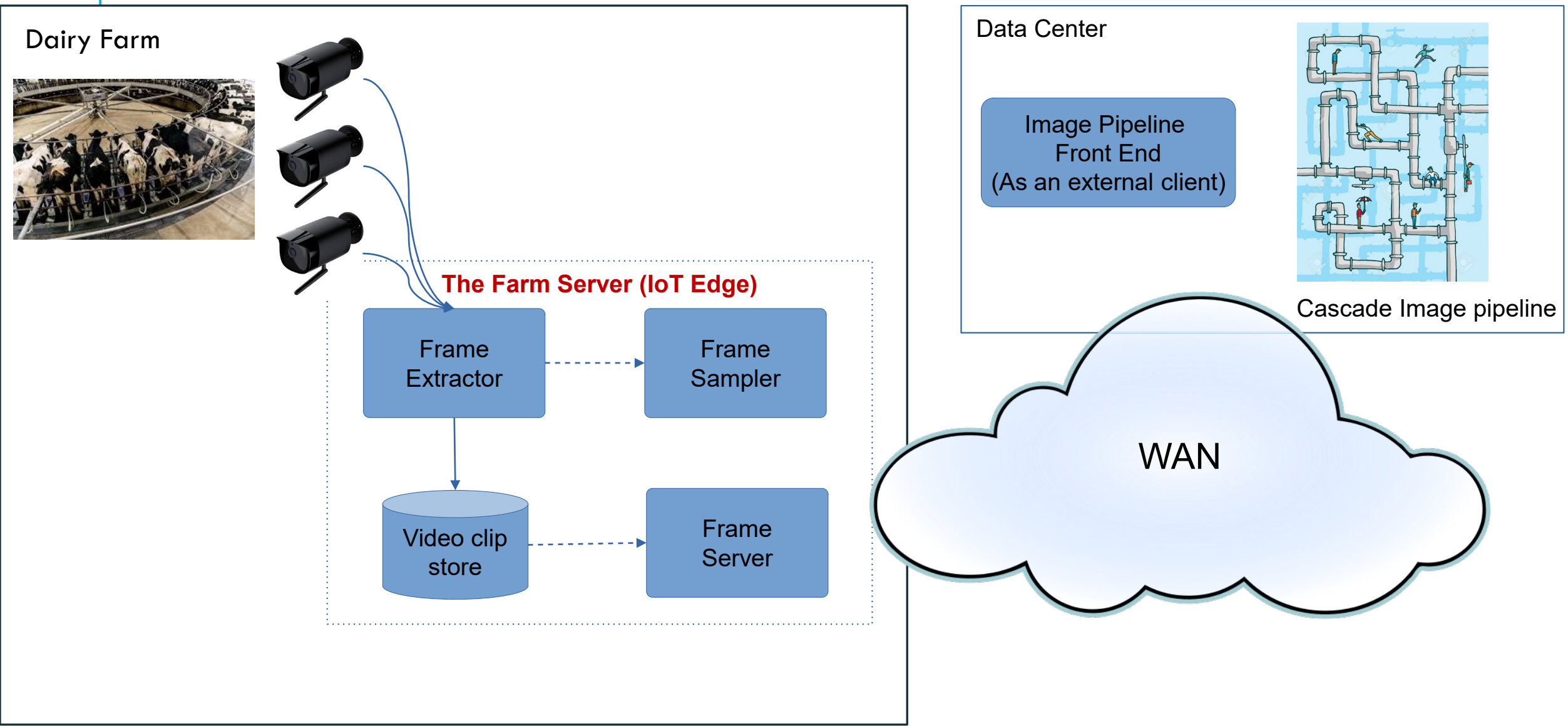


We are doing a basic image pipeline for dairy farming right now:

- A cow walks into the barn.
- In real-time we identify her, pivot and zoom to her haunches, estimate her body fat from those images.
- Combine this with previously uploaded general health metrics.
- Within milliseconds, a decision is reached, perhaps to milk her, perhaps to route her to a holding area (for the vet to come check).

This is working in beta form today.

DAIRY IMAGE PIPELINE: FRONT END



… DETAILED VERSION (PyLINQ ON MSFT AZURE)

Integrate daily data

Date	cow_id	daily_yield	daily_fat	…	daily_protein
12/3/20	1	14	3.96	…	2.89
…	…	…	…	…	…
1/10/21	237	20	4.42	…	4.55

<field>/<cow_id>{<ts(ver)>}
daily_protein/cow_id1{ver_1} = 2.89

Download blobs from Azure & Store to Cascade VCSS subgroup

```
client = cascade_py.ServiceClientAPI()
daily_yield_pool = client.get_object_pool("VolatileCascadeStore","daily_yield")
daily_yield_pool.put(cow_id,val)
```

Upload daily date to Azure Blob Storage

Date	cow_id	…
12/3/20	1	…
…	…	…
12/3/20	237	…

daily_fat/cow_id237{ver_38} = 4.42

Filtered image frames

Farm server

External Client to Cascade

Store to subgroup VCSS
Trigger image analysis
Action = black_cow_infer

```
white_cow_net = load_cow_model( \
    "cow_symbol.json", \
    'white_cow.params')
black_cow_net = load_cow_model( \
    "cow_symbol.json", \
    'black_cow.params')
...
action = cascade_context.get_next_action()
if action.type == "white_cow":
    pred = white_cow_net(action.frame)
else
    pred = black_cow_net(action.frame)
```

CV model
Image analysis
cow id: 127

Streaming image frames through TCP portal

Probability of calving
in next 8h is: 80%

ML Model
birth prediction

LINQ query to retrieve data of most recent 10 days from Cascade about cow 1

```
client = cascade_py.ServiceClientAPI()
cow_id = "cow_id128"
recent_data = []
for field in ["daily_yeild", "daily_fat", "daily_protein"]:
    field_pool = client.get_object_pool("VolatileCascadeStore",field)
    query_result = field_pool \
                    .where(lambda x:x.endswith("/"+cow_id)) \
                    .order_by(lambda x:x.version) \
                    .reverse()
                    .take(10)
    recent_data.append(query_result)
```

C++ IS SIMILAR (BUT MORE EFFICIENT)

Integrate daily data

Date	cow_id	daily_yield	daily_fat	…	daily_protein
12/3/20	1	14	3.96	…	2.89
…	…	…	…	…	…
1/10/21	237	20	4.42	…	4.55

<field>/<cow_id>{<ts(ver)>}
daily_protein/cow_id1{ver_1} = 2.89

Download blobs from Azure & Store to Cascade VCSS subgroup

```
ServiceClientAPI capi;
uint32_t num_subgroups = capi.template<SubgroupType> get_number_of_subgroups();
auto sg_idx = std::hash<std::string>{} (daily_yield) % num_subgroups;
uint32_t num_shards = capi.template<SubgroupType> get_number_of_shards(sg_idx);
capi.template put<SubgroupType>(val,sg_idx,cow_id % num_shards);
```

Upload daily date to Azure Blob Storage

Date	cow_id	…
12/3/20	1	…
…	…	…
12/3/20	237	…

daily_fat/cow_id237{ver_38} = 4.42

Cascade backend

Filtered image frames

Farm server

External Client to Cascade

Store to subgroup VCSS
Trigger image analysis

Action = black_cow_infer

```
InferenceEngine white_cow_ie;
InferenceEngine black_cow_ie;
auto frame = dynamic_cast<ImageFrame*>(action.action_data.get());
if (action.action_type == WHITE_COW) {
    std::tie(name,soft_max) = white_cow_ie.infer(*frame);
} else {
    std::tie(name,soft_max) = black_cow_ie.infer(*frame);
}
PCSS::ObjectType obj(frame->key,name.c_str(),name.size());
```

Streaming image frames through TCP portal

Probability of calving in next 8h is: 80%

ML Model birth prediction

CV model
Image analysis

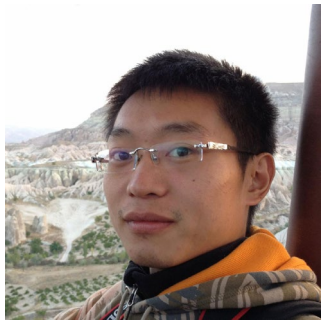
cow id: 127

LINQ query to retrieve data of most recent 10 days from Cascade about cow

```
std::vector<std::string> fields{"daily_yield", "daily_fat", "daily_protein"};
std::vector<std::vector<double>> recent_data;
for (std::string& field: fields)
{
    auto sg_idx = std::hash<std::string>{} (field) % num_subgroups;
    auto sh_idx = 128 % num_shards;
    std::string key_quering(field + "/cow_id128");
    auto query_result = from_shard<VCSS,ServiceClientAPI>(keys,capi,sg_idx,sh_idx,-1)
                        .where([](typename SubgroupType::ObjectType obj) { return !obj.key.compare(key_quering); })
                        .orderBy([](typename SubgroupType::ObjectType obj) { return obj.version; })
                        .reverse()
                        .take(10)
                        .toStdVector();
    recent_data.emplace_back(query_result);
}
```

THE CASCADE AND DERECHO TEAM

Weijia Song



Alicia Yang

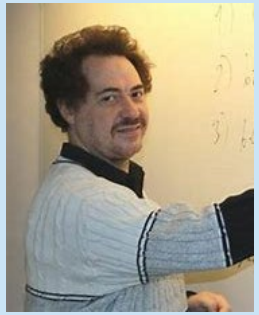
Ken Birman

Sagar Jha

Lorenzo Rosa

Matthew Milano
(post-doc at Berkeley)

Andrea Merlina



Roman Vitenberg

Cornell undergrads:
Aahil Awatramani, Ben Posnick, Max Charlamb, Thompson Liu, Archishman Sravankumar, Aaron Weiss, Peter Zheng

Edward Tremel
(faculty at Augusta Univ.)