CASCADE: FULL DETAILS

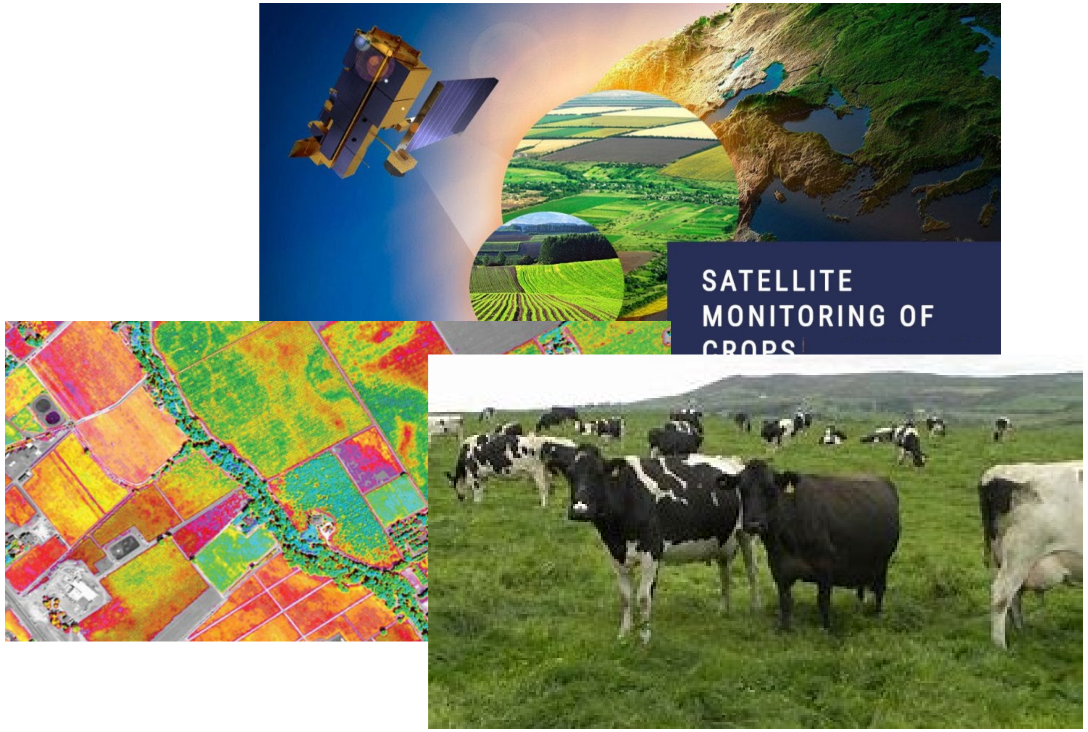
Ken Birman
CS5412 Fall 2022
Lecture 10

THE PROMISE OF EDGE INTELLIGENCE

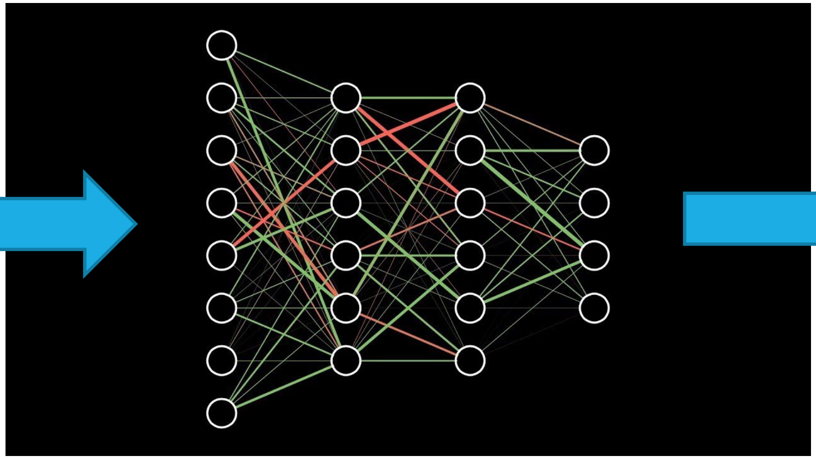
Can "real world" artificial intelligence be *as accurate* and *as fast* as human intelligence?

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

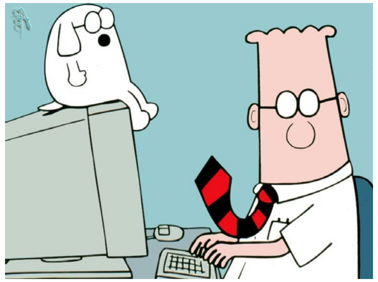
Data sources



Cooperative ML
Distributed AI

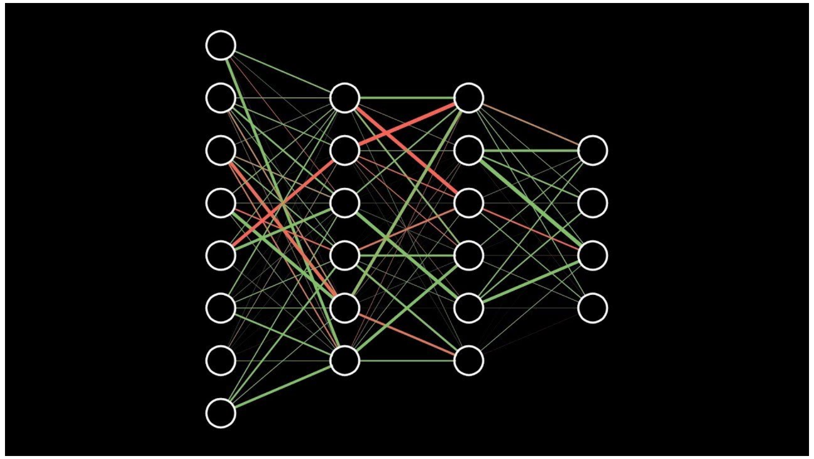


Smart
Queries



How much should I budget for raw milk purchases in March for my yoghurt factory?"

COOPERATIVE ML?

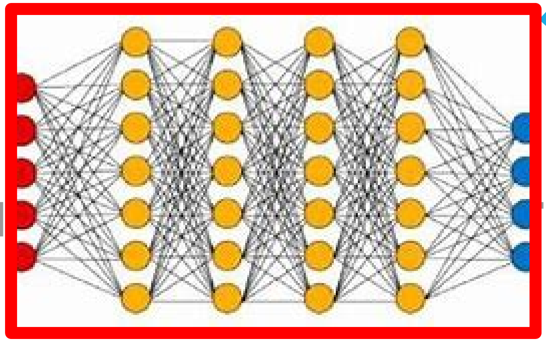


One ML to rule them all? Not likely!

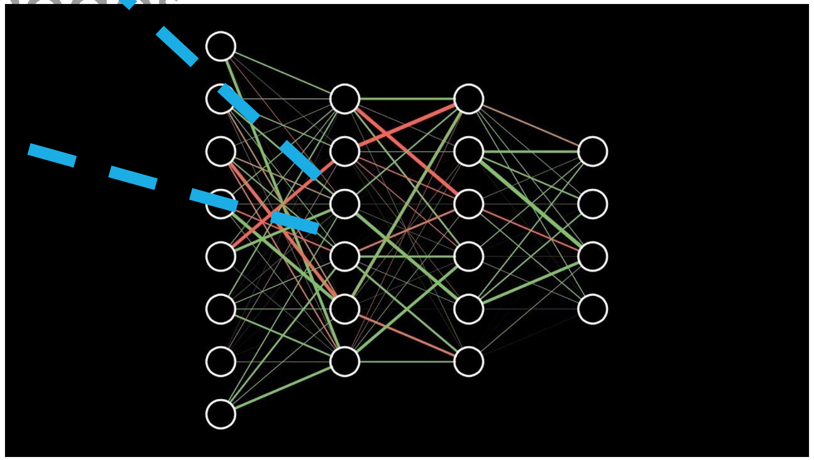
So we need to think about cooperation between MLs in robotics, smart homes, 5G, digital twin scenarios.

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

A SINGLE λ COULD BE AN ENTIRE DISTRIBUTED AI, ON A COLLECTION OF MACHINES



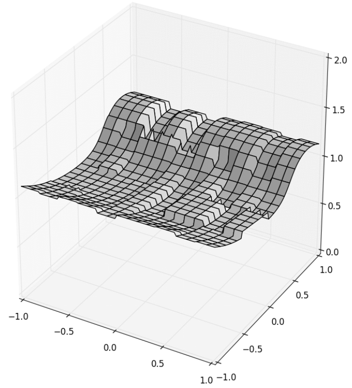
Single λ may run on a e nodes



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

THE NEED IS FOR <u>TIME-FOCUSED EDGE INTELLIGENCE</u>

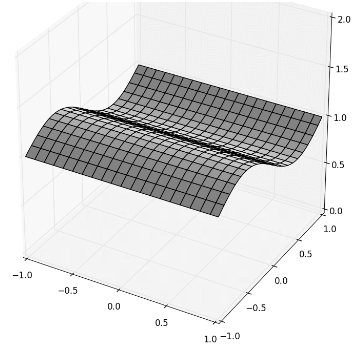
In today's cloud, AIs fight platform noise



IoT data arrives as a stream.

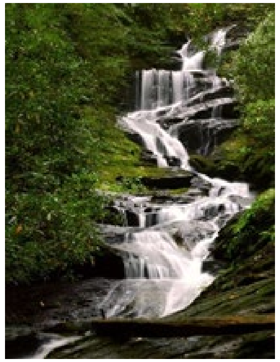
Edge intelligence must be *instantly reactive*

- Get power grid data at 250ms intervals
- Format it as 15 x 15 tensor by GPS location with a time axis from10:00:00 to 10:01:00
- Run <u>*ML phase disruption analytic*</u> on the tensor



Cascade: Faster yet accurate

HIGH LEVEL CASCADE GOALS



Legacy support: Easy to use with no need to change your code

Much faster than standard platforms: low delay, high bandwidth

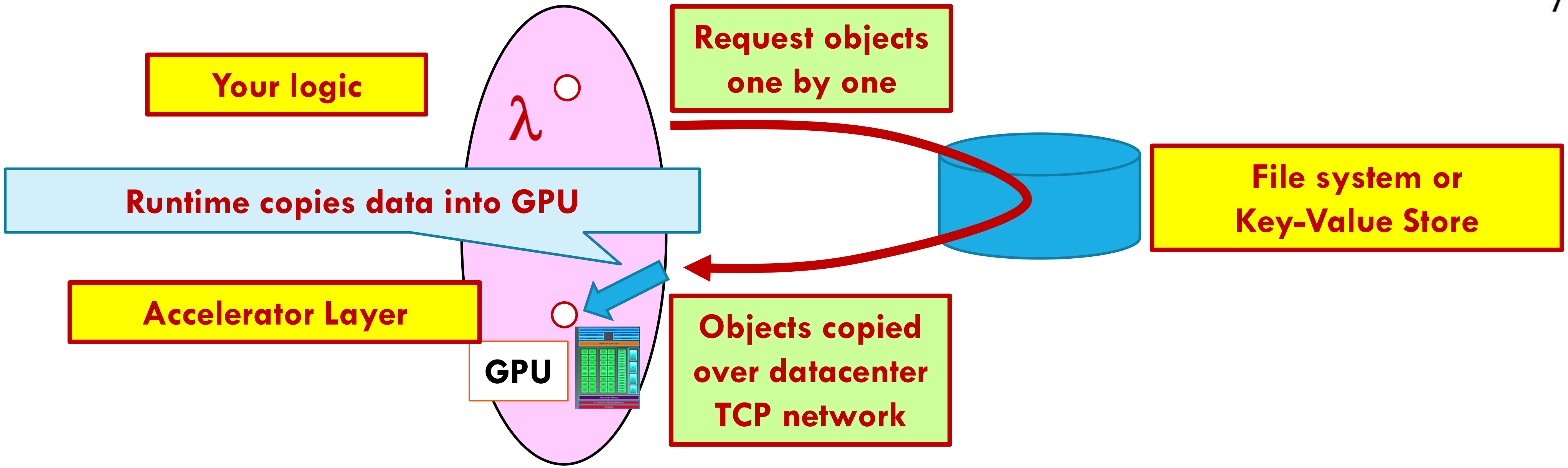
Stronger guarantees: Your ML doesn't fight platform "noise"

BUT HOW WILL WE GUARANTEE CONSISTENCY? HOW CAN IT BE SO FAST?

Let's start by asking how to make it fast

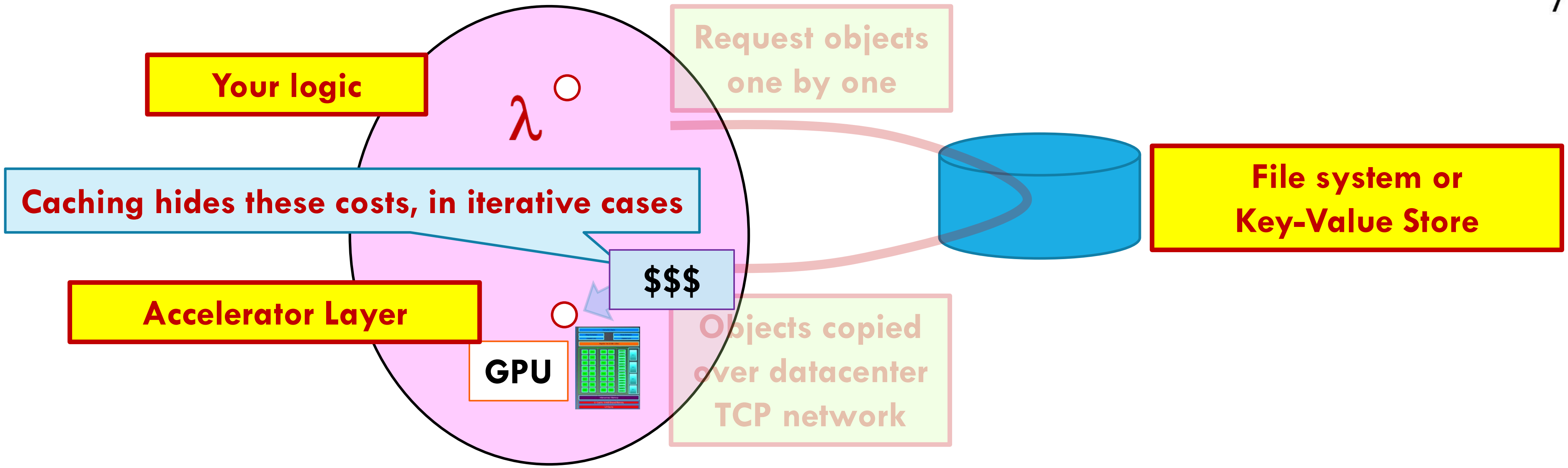
Then we will see that we already know how to guarantee consistency: be smart about time (and time skew), and use Chandy-Lamport consistent cuts when we hit the limits of clock precision/accuracy!

TRADITIONAL APPROACH



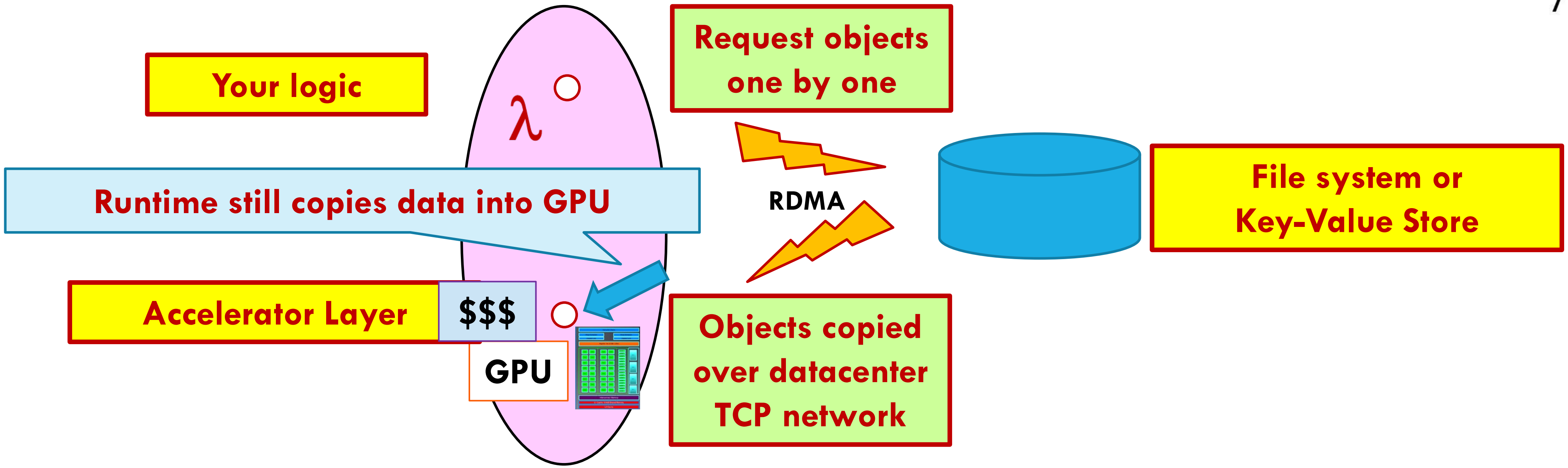
Your code is in its own address space, maybe on a different computer

TRADITIONAL APPROACH



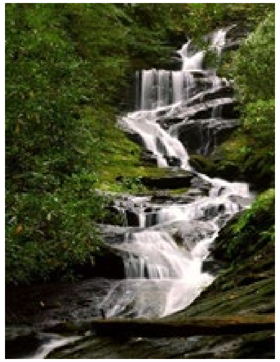
Your code is in its own address space, maybe on a different computer

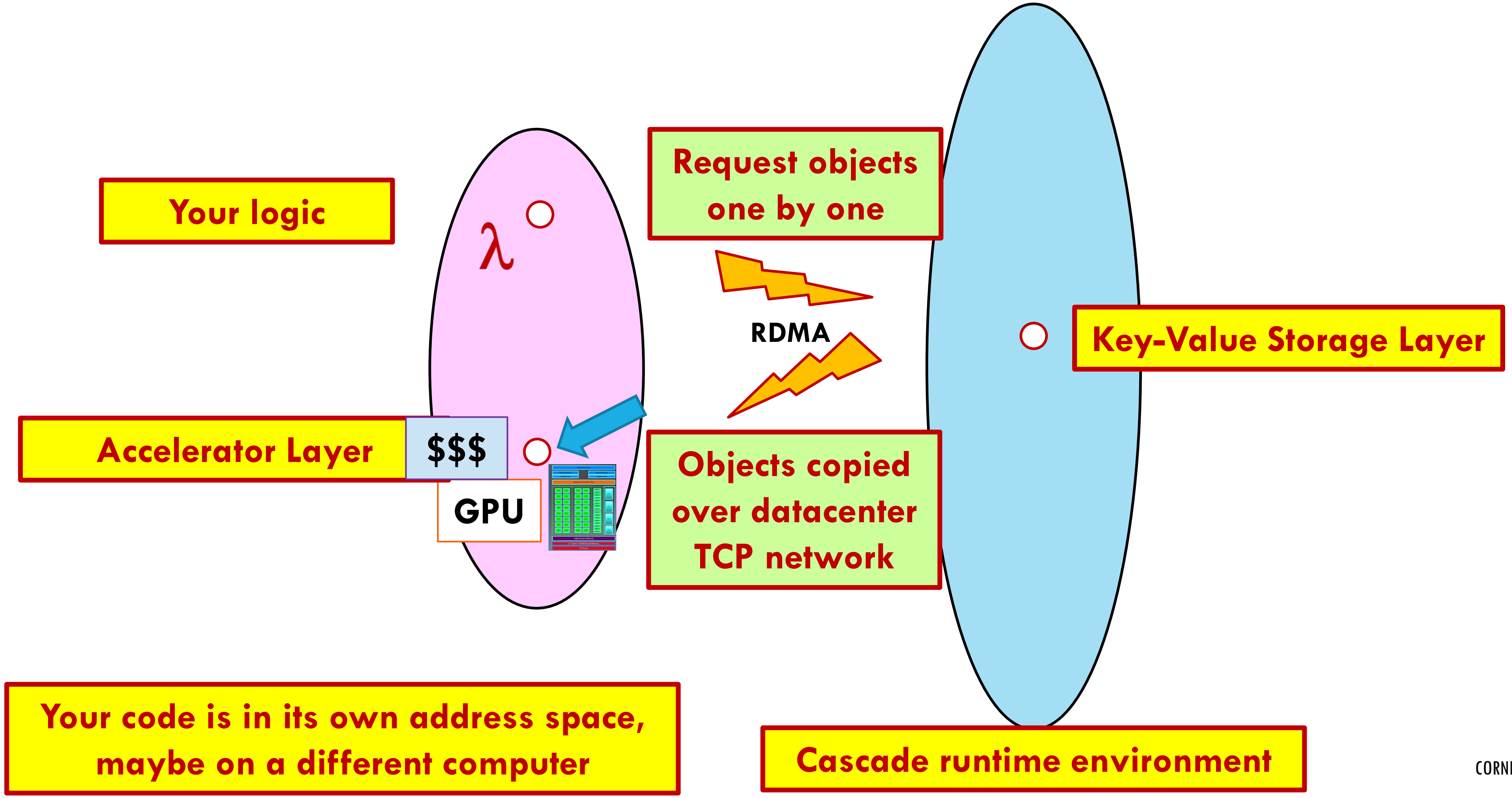
RDMA CAN HELP… A LITTLE



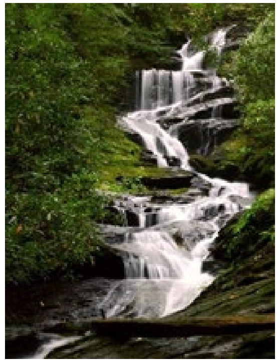
Your code is in its own address space, maybe on a different computer

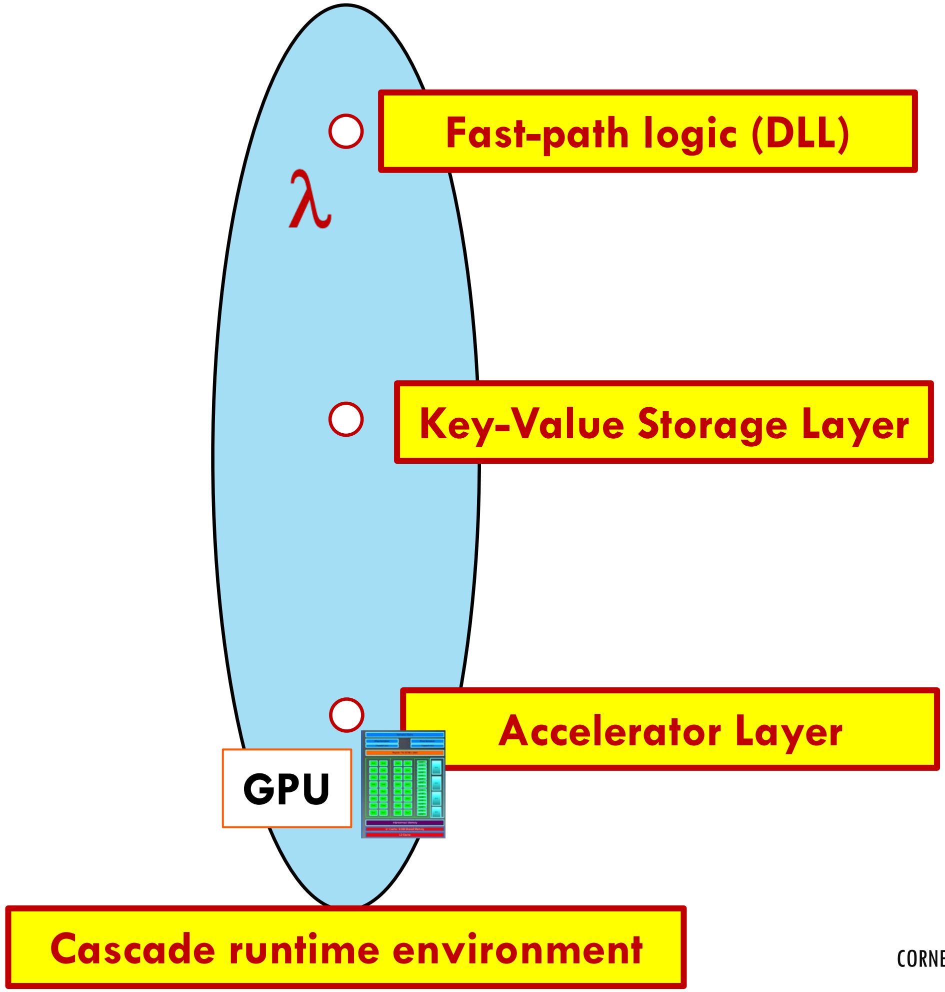
CASCADE CAN BE USED THIS WAY…



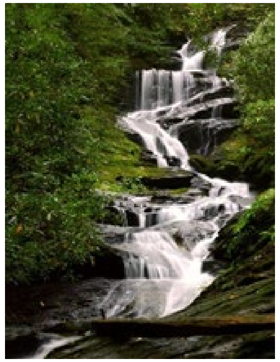


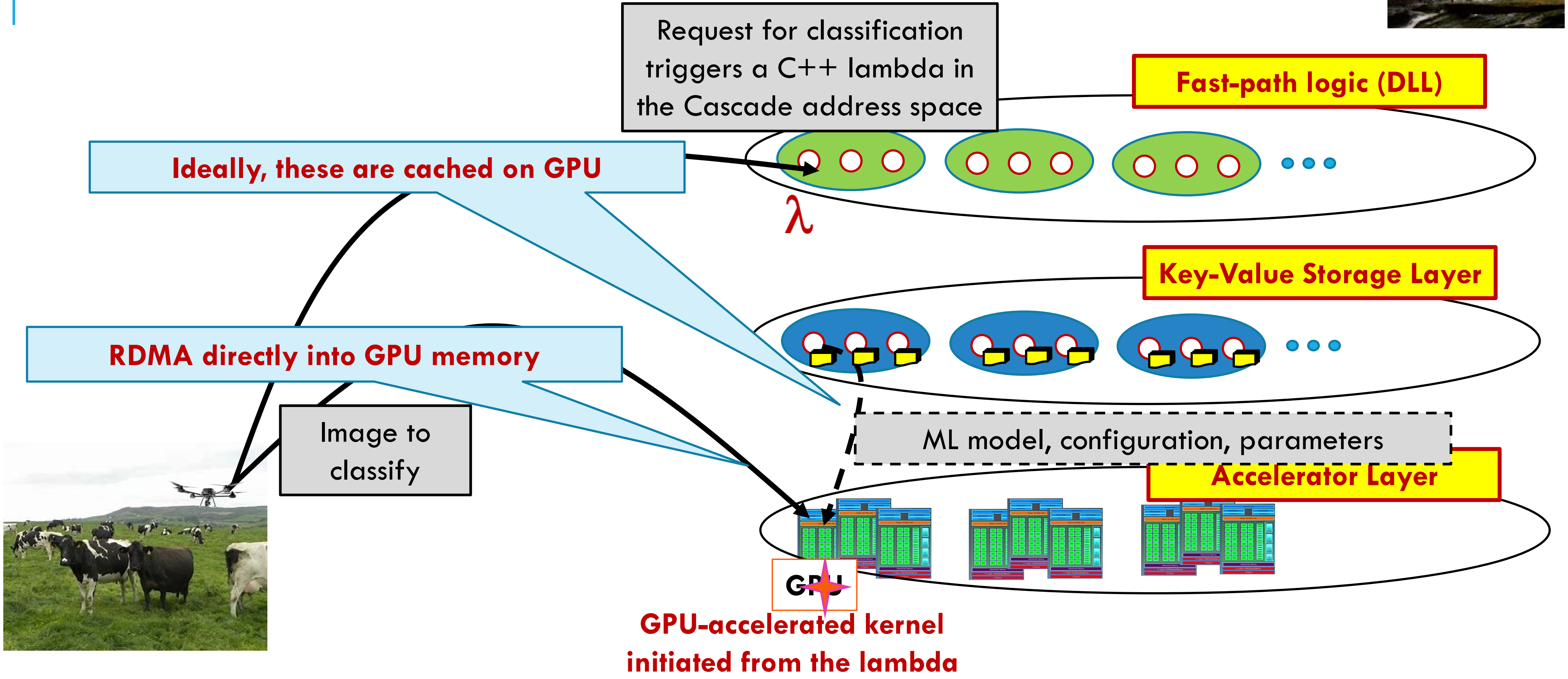
… BUT CAN ALSO HOST USER CODE

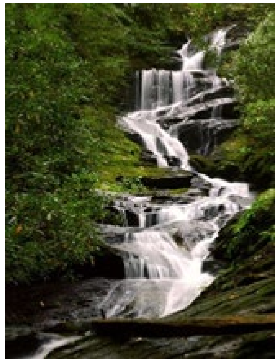




CASCADE: CUSTOMIZED SMART SERVICES

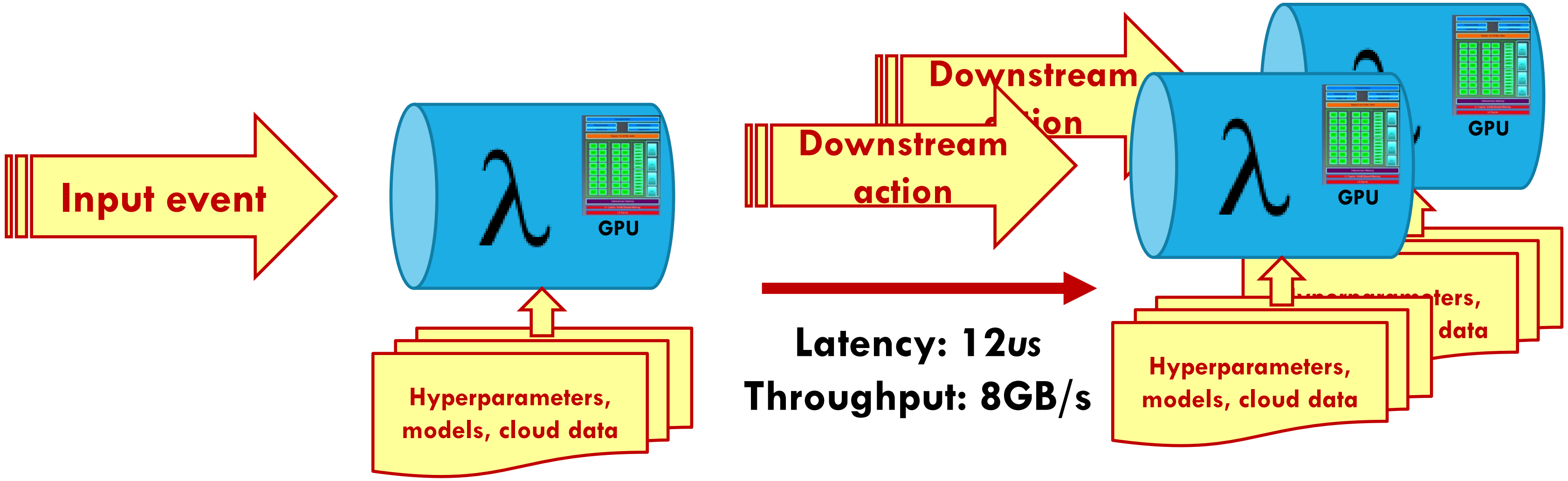






FAST-PATH PERFORMANCE

A simple federated ML pipeline

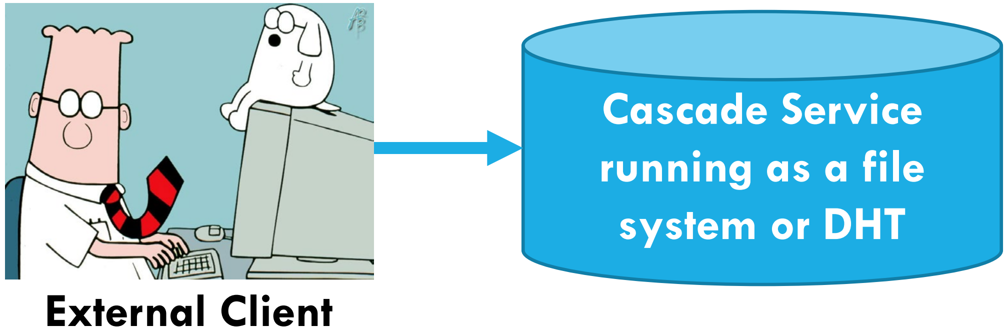


Cascade is close to ideal efficiency on our hardware and 100 to 10,000x faster than common options like Apache Flink

THE CASCADE MODEL

Is it a service? A library?

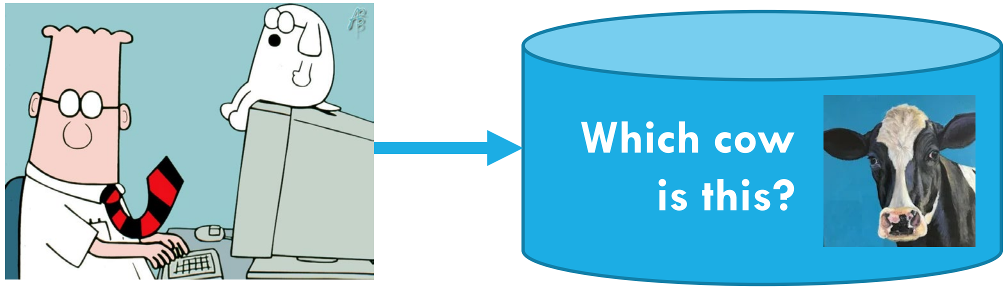
CASCADE IS A SERVICE



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”.   They would think of Cascade purely as a key-value storage system, accelerated by RDMA.

CASCADE IS AN EXTENSIBLE SERVICE



External Client

But the highest speed is achieved by extending the Cascade service by adding logic (λs) that will run inside Cascade.

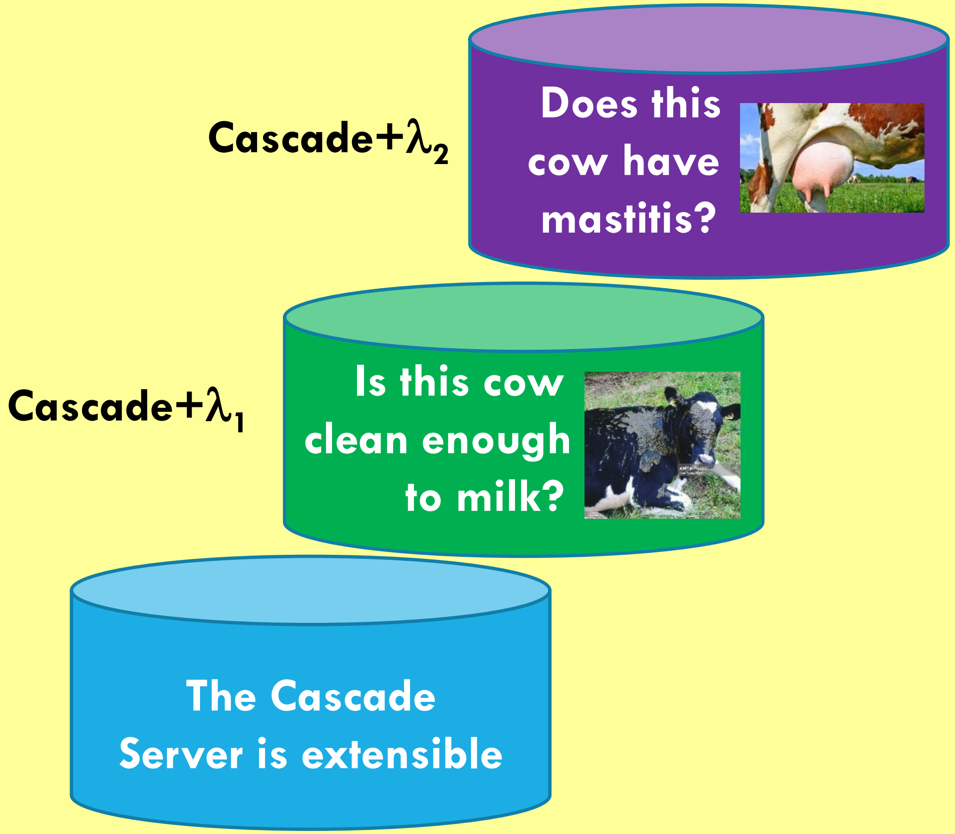
Used this way, Cascade enables creation of a customized service: a "smart" service ideal for your purposes.

RECAP: EXTENSIBILITY (PAAS MODEL)

Cascade is <u>one</u> service

But when you supply customization it acts like many specialized services, one per application

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



A single Cascade server can support **multiple customized services**

THE CASCADE STORAGE MODEL

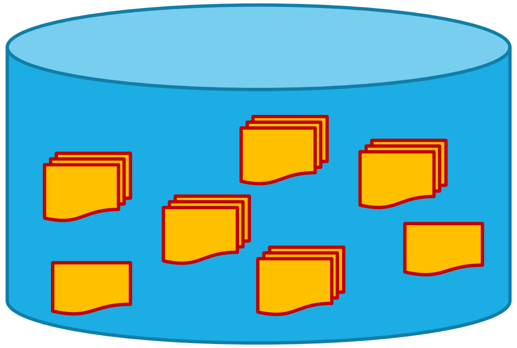
How should Cascade data be managed?

CASCADE IS A KEY-VALUE STORE

… but it also supports a file system API

… and the key-value API itself has some fancy options, beyond the basic **put/get/watch** we learned about in lectures 1-4

IN FACT CASCADE'S FILE SYSTEM IS AN EXAMPLE OF EXTENSIBILTY



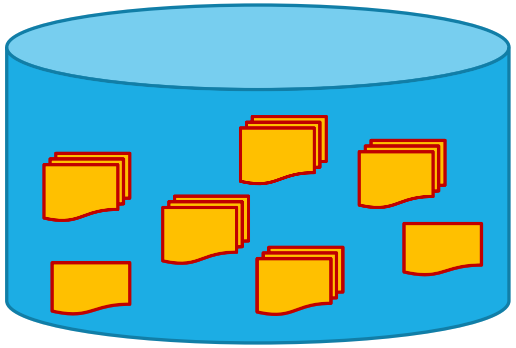
User sees scalable, "private" object pools

Cascade's file system is implemented as a λ:

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

Similar to Ceph file system, yet just a few lines of code to translate user requests (via the FUSE library) into Cascade put/get/watch

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

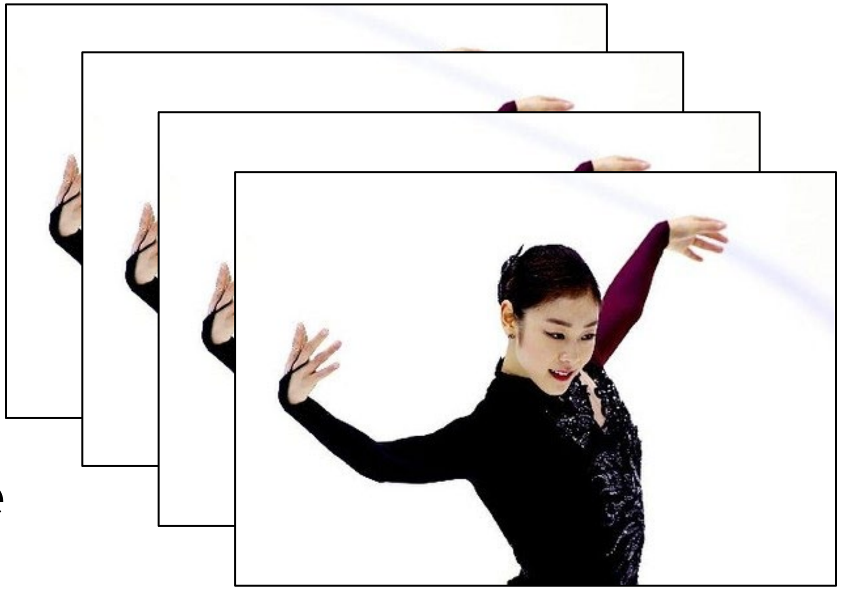


User sees scalable, "private" object pools

Same model used in cloud file systems like HDFS and Ceph.

- Updates are log appends using Paxos. Each object has a log of versions that evolved over time.
- Reads run on the *stable prefix* of the log.

VERSIONED UPDATES



Each time you write to an object, Cascade creates a new version.

If you read an object, modify it, and then write it back, you can tell Cascade which version you modified. The **put** will double check to be sure that this is still the current version before replacing it, and otherwise returns an error (then you can loop)

VERSIONED/TEMPORAL QUERIES

Accessed via **get**, but specify the version # or time desired

In the volatile case, Cascade only keeps the most recent version. With persistent objects, Cascade keeps a log of past versions.

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

VERSIONED OBJECTS

Versioning enables intelligence over a sequence of events

- *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.

THE CASCADE COMPUTE MODEL

Virtual synchrony, Atomic multicast, Paxos, consistent cuts…

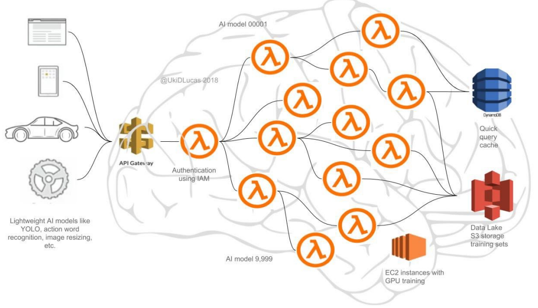
THE CENTRAL PUZZLE

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?



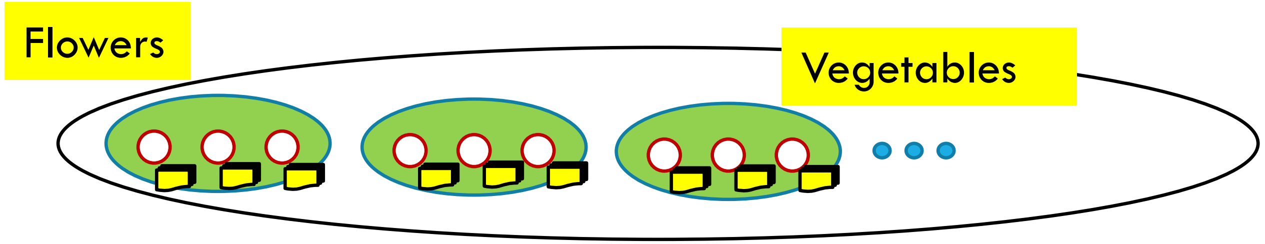
We support many languages. Native APIs are Python with various packages (including LINQ) and C++ with LINQ.

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.

HOW CAN A KEY-VALUE STORE "BE" A CLASSIFIER SERVICE OR AN ANALYTIC SERVICE?

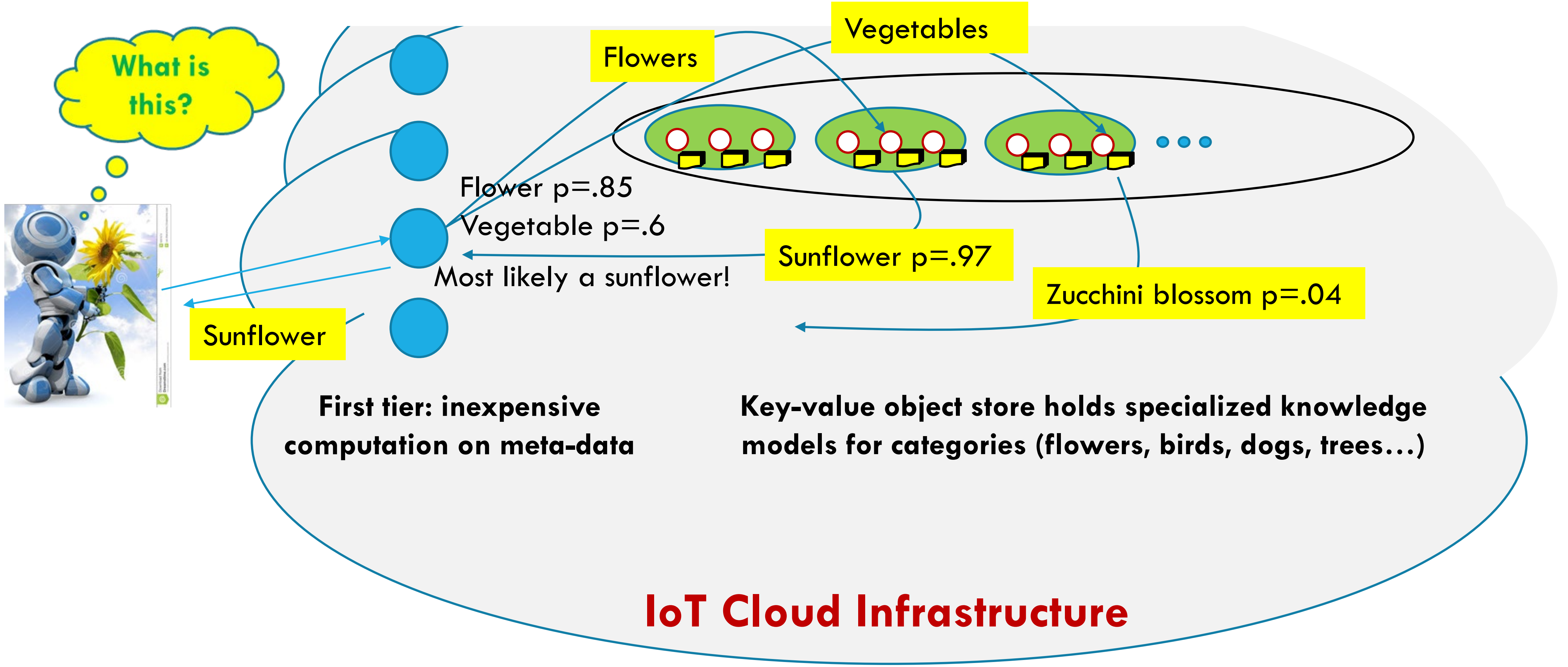
We run Cascade on a set of nodes. Here we see nine nodes in three shards.



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.

… ENABLING THIS KIND OF SOLUTION



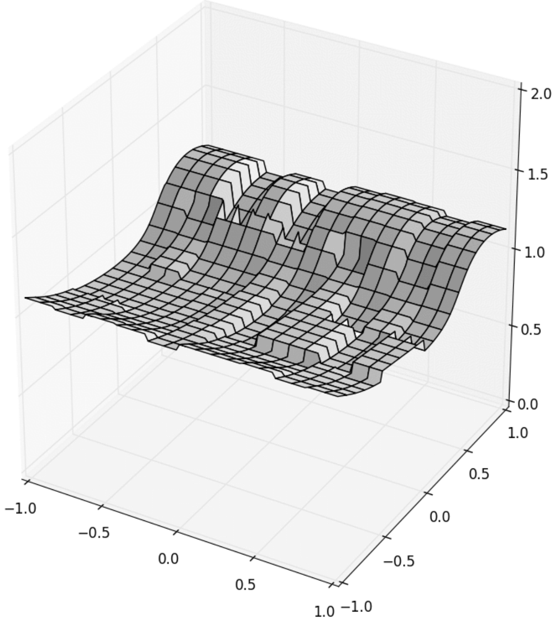
CASCADE QUERIES SEE CONSISTENT CUTS

Suppose a cooperative 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).

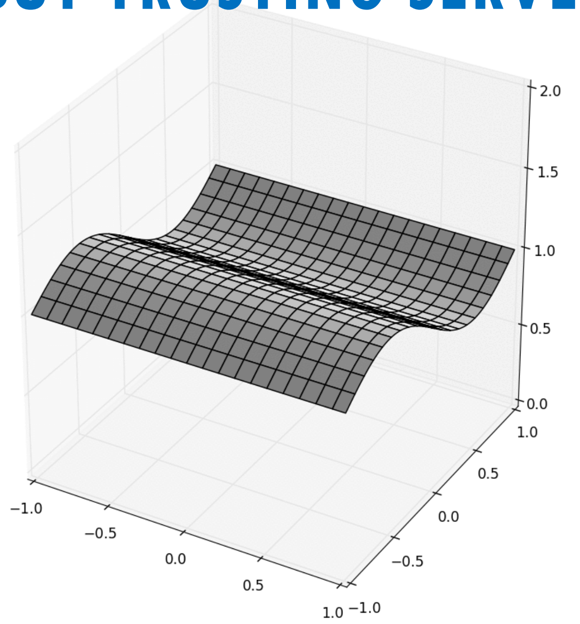
Effect: The lambda won't see platform-induced inconsistencies.

VISUALIZATION OF CASCADE CONSISTENCY

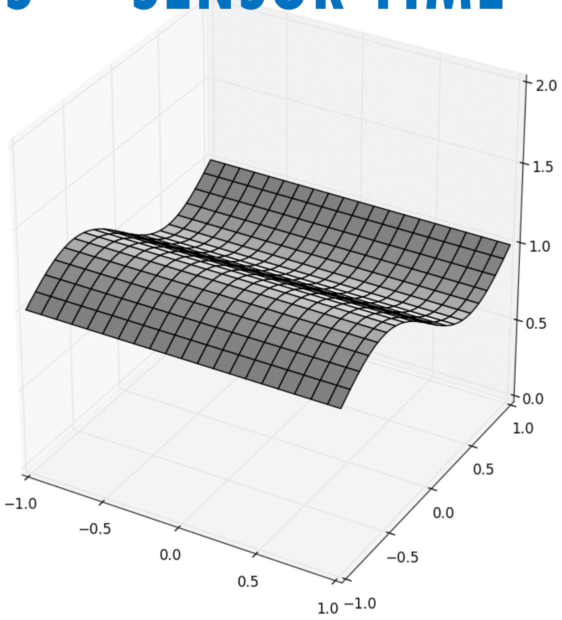
HDFS



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)

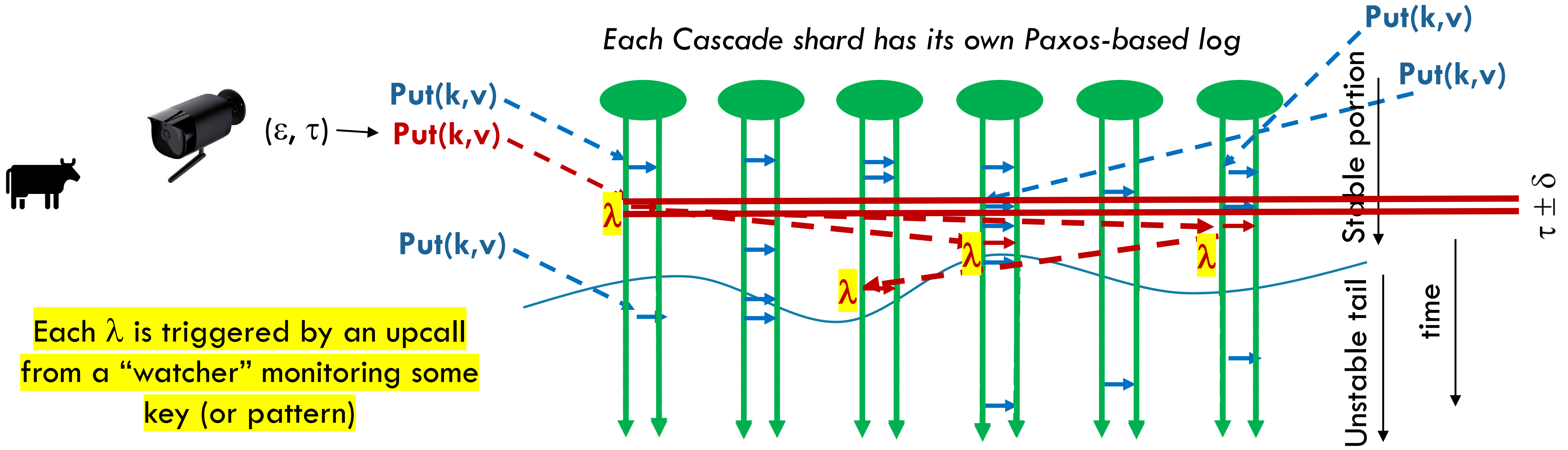
CENTRAL CONCEPTUAL INSIGHT

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

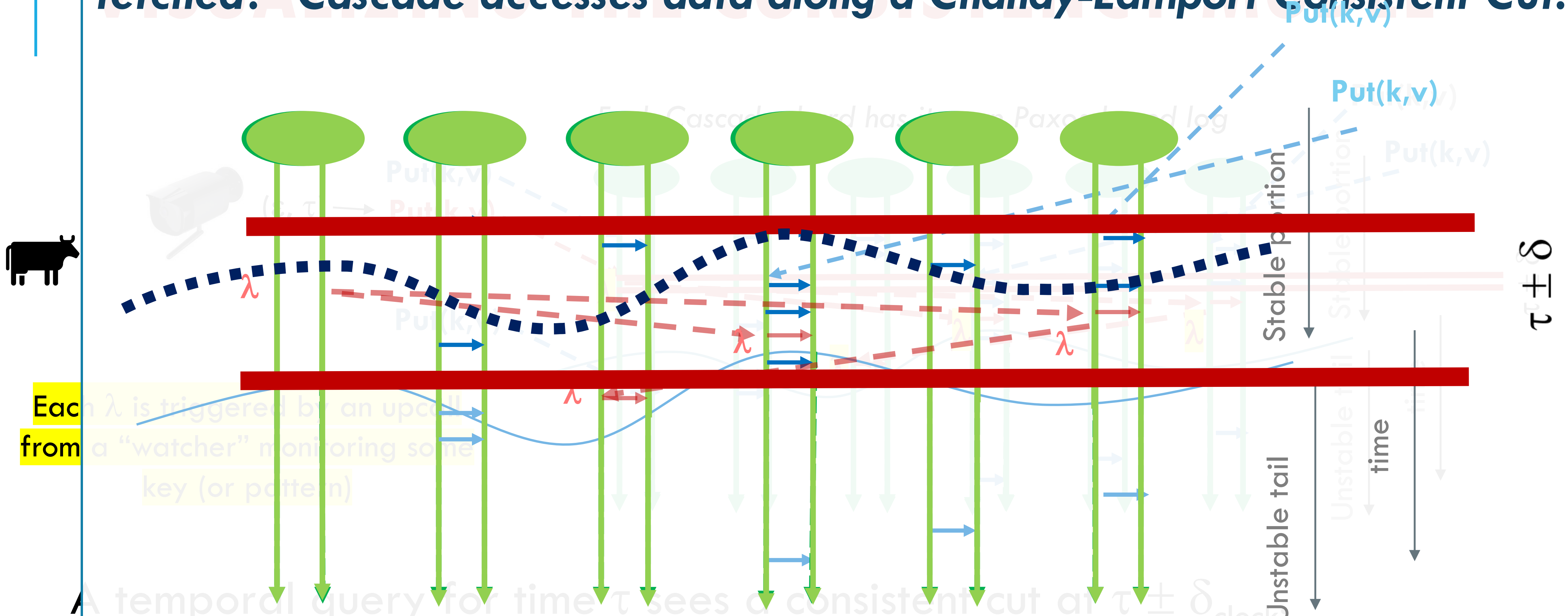


Each λ is triggered by an upcall from a "watcher" monitoring some key (or pattern)

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 is a choice to make: Which versions will be fetched? Cascade accesses data along a Chandy-Lamport Consistent Cut.



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.

GROUPING OBJECTS

Often a lambda will need to access several objects that should ideally all have their own keys, yet you want them grouped on the same shard.

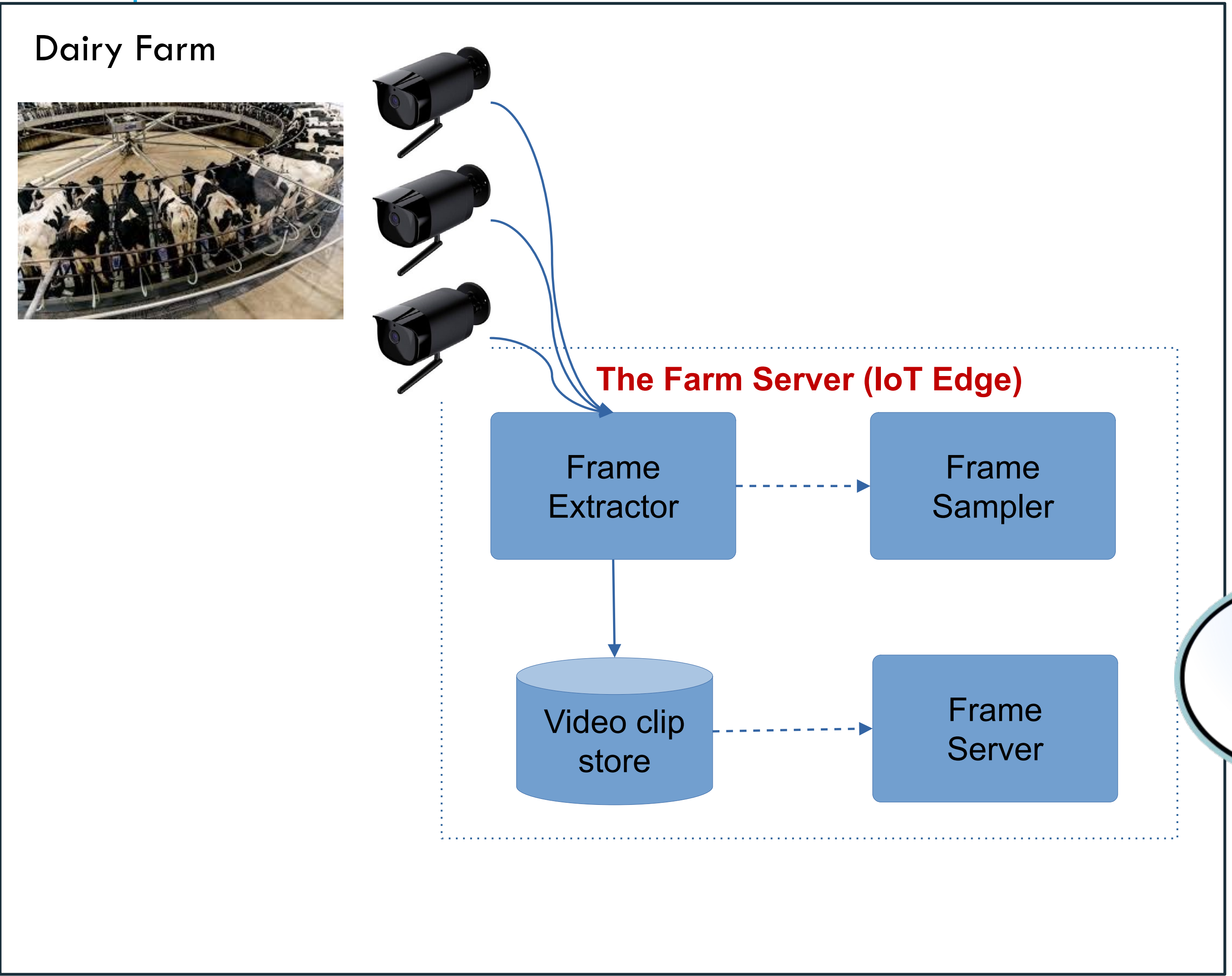
For this, Cascade supports “affinity grouping”. Each object has a second key, used for placement.

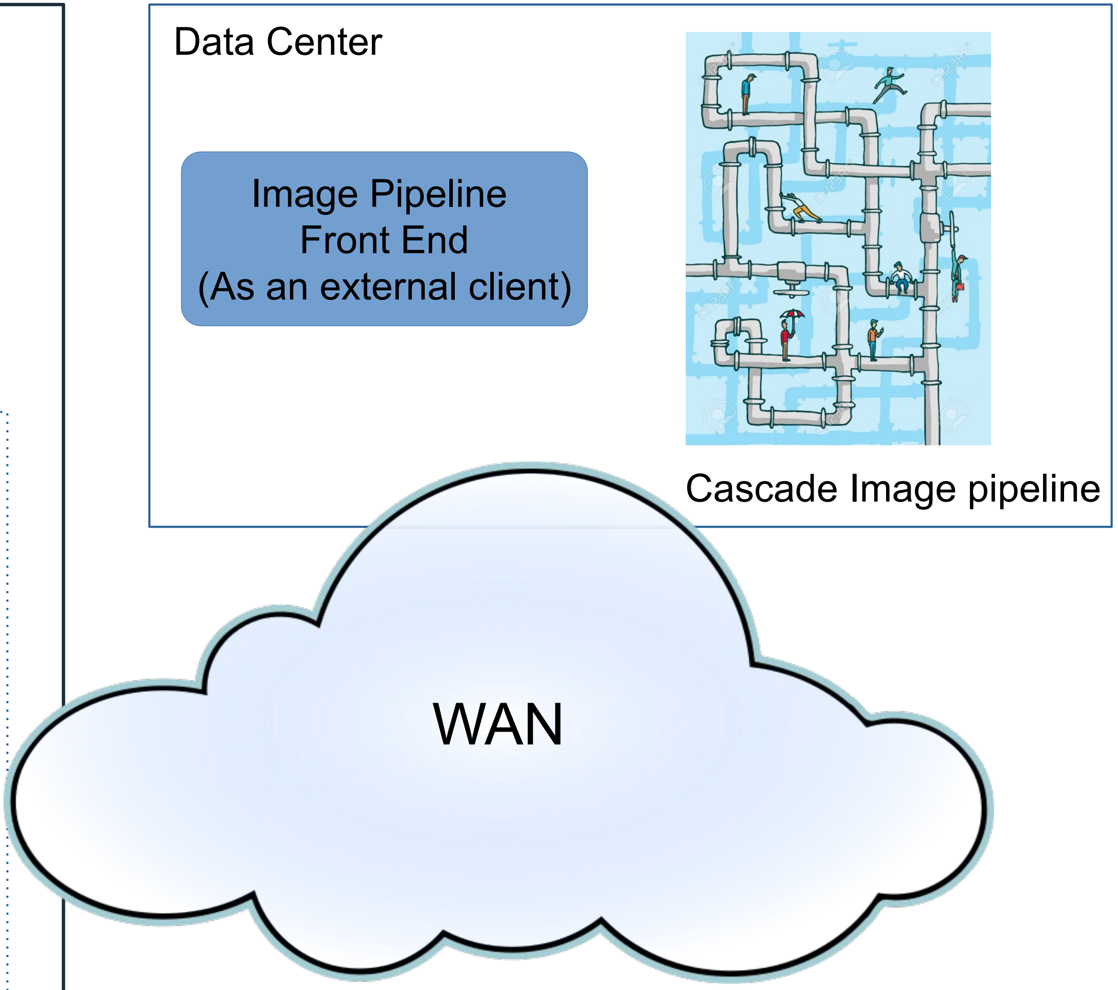
Even if A and B have different keys – “names” – they will be stored on the same shard if you assign them the same affinity key.

WHAT DOES A REAL APPLICATION LOOK LIKE TODAY?

Example, courtesy of Weijia, Alicia and Thompson

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 128

```
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

Farm server

Filtered image frames

External Client to Cascade

Store to subgroup VCSS
Trigger image analysis

Action = black_cow_infer

Cascade backend

```
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

CV model
Image analysis
cow id: 127

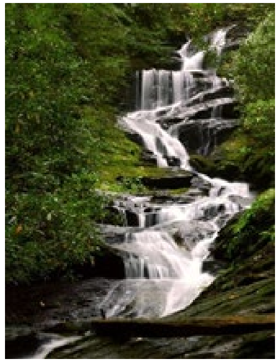
Probability of calving
in next 8h is: **80%**

ML Model
birth prediction

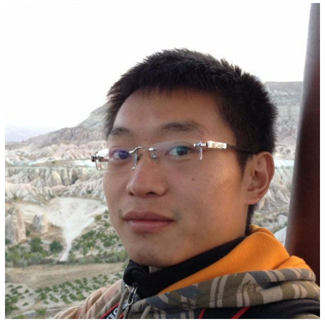
LINQ query to retrieve data of most recent 10 days from Cascade about cow

```
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

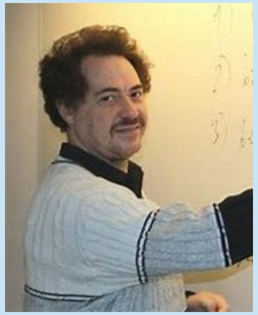
Ken Birman

Sagar Jha

Lorenzo Rosa

Mae Milano
(post-doc at Berkeley)

Andrea Merlina



Roman Vitenberg

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

Thompson Liu

Edward Tremel
(faculty at Augusta Univ.)