WEEK 3: CLOUD CASE STUDY-SEMESTER PROJECTS

Lecture 7

Cloud Case Study (Smart Dairy Forming)

Dr. Muhammad Tariq

Disclaimer! Do not distribute …

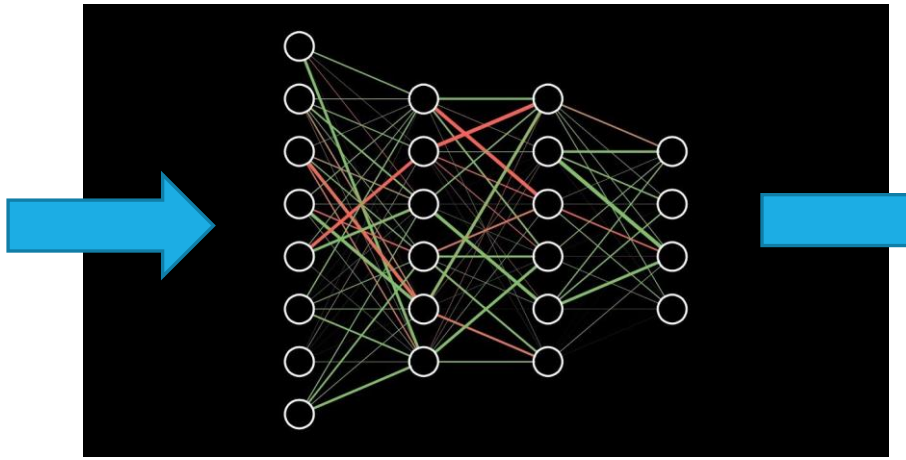
- These slides are not always prepared by me.
- Most of the content comes from the reference book and materials

Textbook (or **Laboratory Manual** for Laboratory Courses)	Textbook: Marinescu, Dan (2017) Cloud Computing Theory and Practice (2nd Ed.)
Reference Material	Recommended Reference Material: Ken Birman https://www.cs.cornell.edu/courses/cs5412/ Distributed and Cloud Computing: Clusters, Grids, Clouds, and the Future Internet, K Hwang, J Dongarra and GC. C. Fox, Elsevier, 1st Ed. IEEE Transactions on Cloud Computing Journal of Cloud Computing: Advances, Systems and Applications

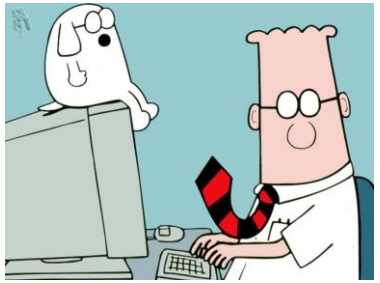
THE WORLD IS GENERATING A NEW WAVE OF IOT/ML PIPELINES… THERE ARE MANY USE CASES

Data sources

Federated ML
Distributed AI



Smart
Queries



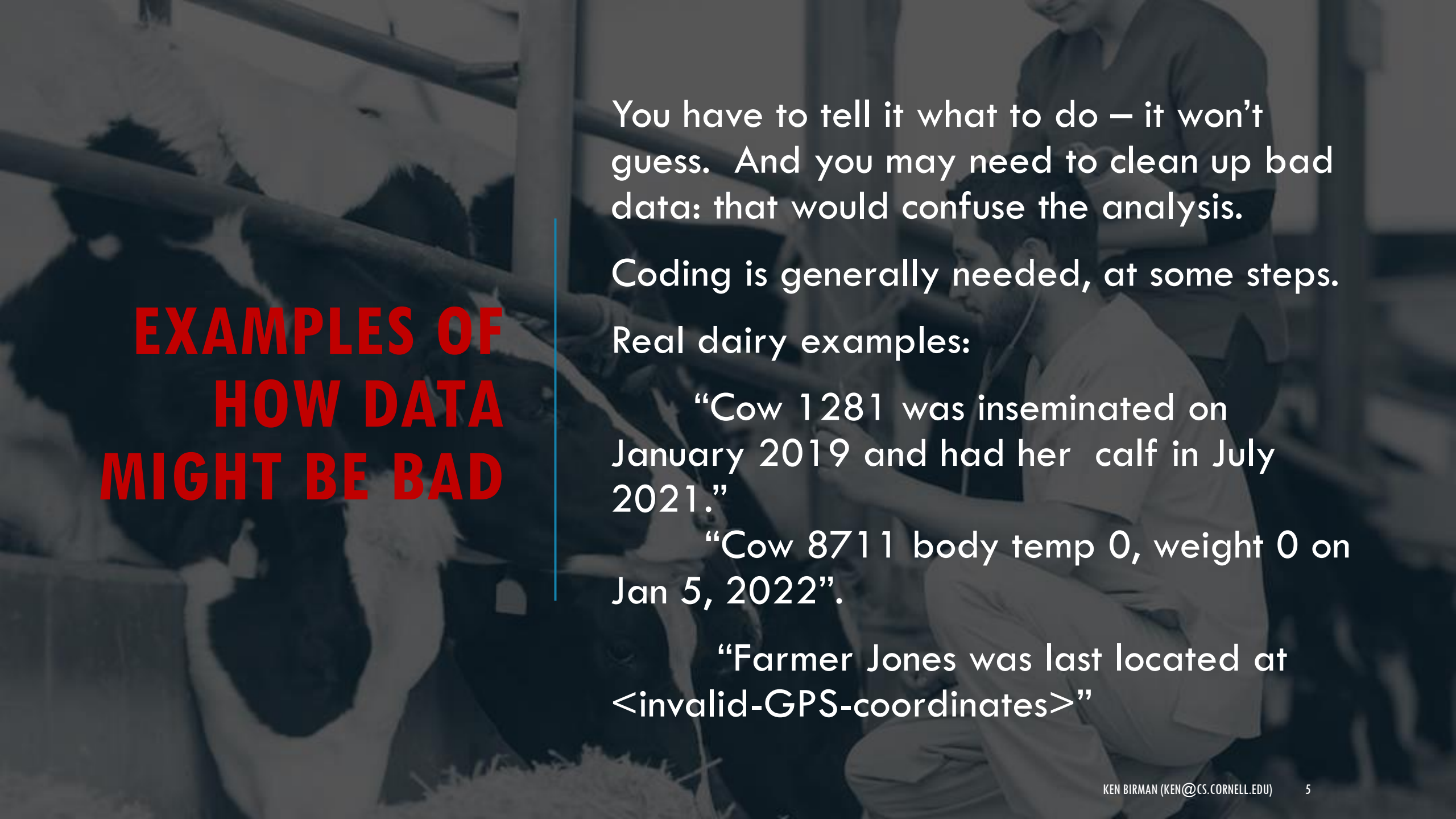
How much should I budget for raw milk purchases in March for my yoghurt factory?"

WHERE IS THE ROLE OF CLOUD IN THE PROJECT?

The cloud architecture used for the web, today, is a kind of graphical pipeline centered on:

- Rapid responses to web queries, like from Facebook
- Batched updates which accumulate as they are passed "inward" and will be done later. Eventually the effects propagate up to the first tier.

For IoT edge settings, we generally need immediate reactions. And we also need to deal with challenges like data errors.



EXAMPLES OF HOW DATA MIGHT BE BAD

You have to tell it what to do – it won't guess. And you may need to clean up bad data: that would confuse the analysis.

Coding is generally needed, at some steps.

Real dairy examples:

"Cow 1281 was inseminated on January 2019 and had her calf in July 2021."

"Cow 8711 body temp 0, weight 0 on Jan 5, 2022".

"Farmer Jones was last located at <invalid-GPS-coordinates>"

DAIRY INTELLIGENCE PLATFORM (A Case Study)

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

```
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)
```

Filtered image frames

Store to subgroup VCSS
Trigger image analysis

Action = black_cow_infer

Farm server

External Client to Cascade

Streaming image frames through TCP portal

Probability of calving
in next 8h is: 80%

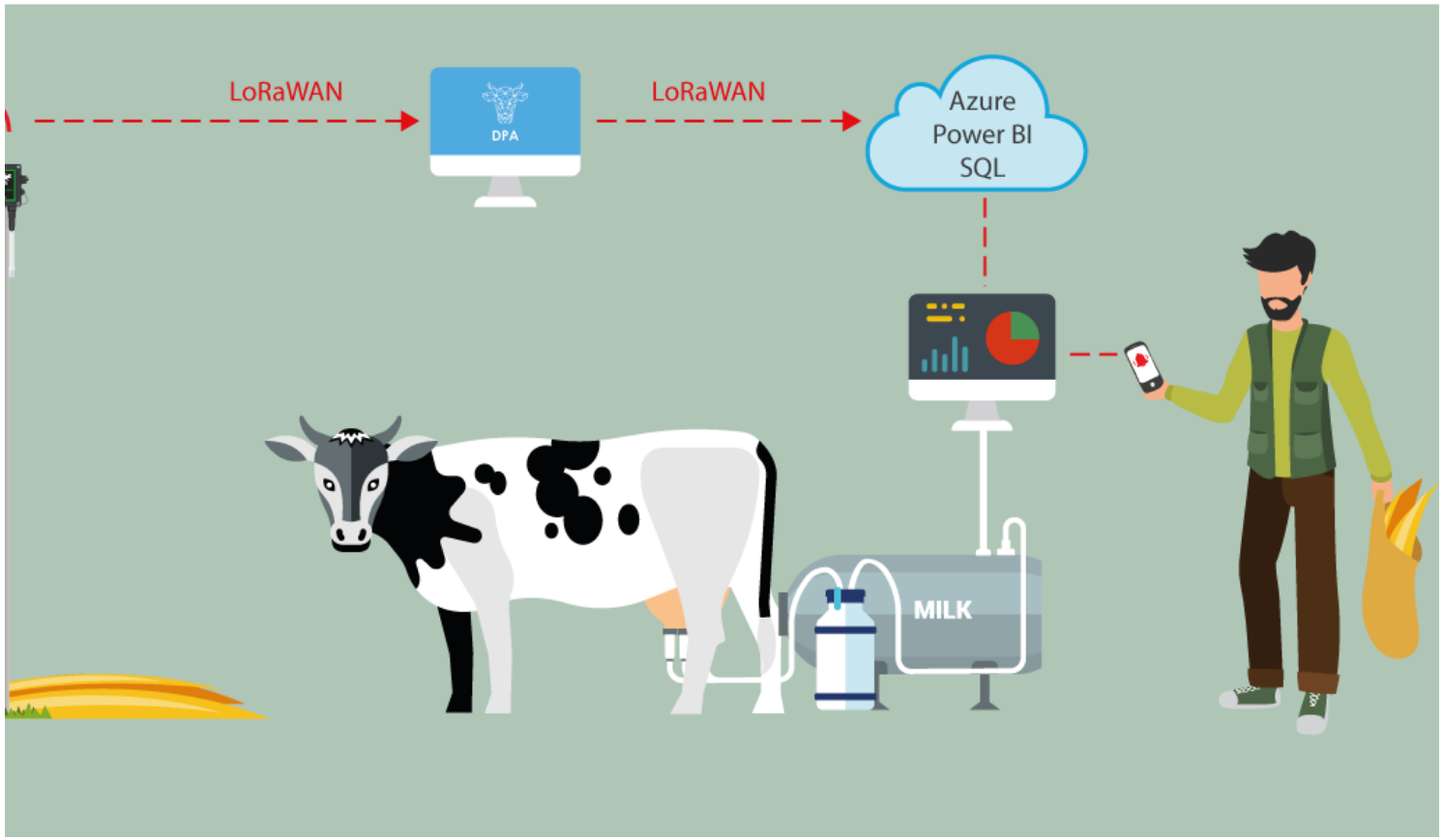
ML Model
birth prediction

CV model
Image analysis

cow id: 127

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



PROJECT WILL NOT BE COMPLETE WITHOUT INTERNET OF THINGS (IOT)

Today's cloud has been enlarged in recent years so that we can connect devices to the cloud, very much in the same way as we attach clients.

The idea is to create a device (say, a smart thermostat) so that it produces web pages in the same format used when a web browser talks to the cloud. Now the sensor can "talk to the cloud" to upload new data.

Same for things that take actions ("actuators").

Suppose a company wants to implement a good physical security solution.

This could include moment monitoring, facial recognition, etc at various doorways.

Swipe sensor to function server: {NewSwipe, Name=Ken_Birman, ….}

Camera to function server: {NewImage, }

Function server to audit-log: {Tuesday, 10:05am, Ken_Birman, …}

Function server to door-lock: {Say="You are approved to enter", Unlock}

WHAT IS IOT BEST AT, TODAY?

… BUT THE FUTURE IS THRILLING!

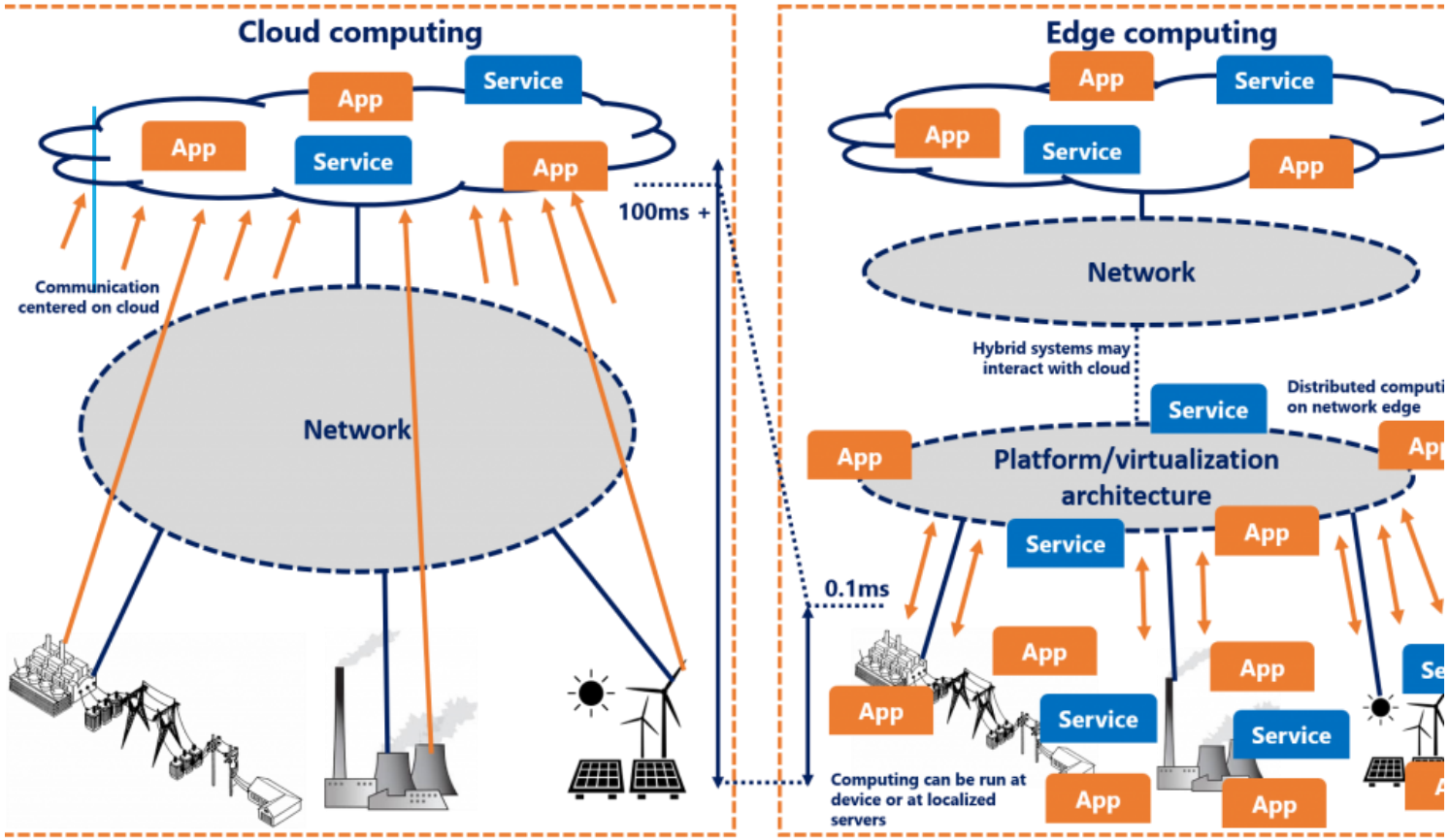
Smart Homes

Smart Farms

Smart Grid

Smart Highways

These are just a few examples of future cloud IoT opportunities



IOT EDGE VERSUS IOT CLOUD

Many devices need even lower latency than 100ms. For example, a drone flying over a farm may need continuous directions on where to fly.

To solve this, vendors have created specialized "mini-clouds" that run on a machine or a small cluster close to the devices: in the home, or office building, or on the farm, or even in a truck that can go from place to place.

This is called an IoT Edge approach. The IoT Edge connects back to the IoT cloud, where more of the heavy-lifting can occur, but handles "easy" tasks.

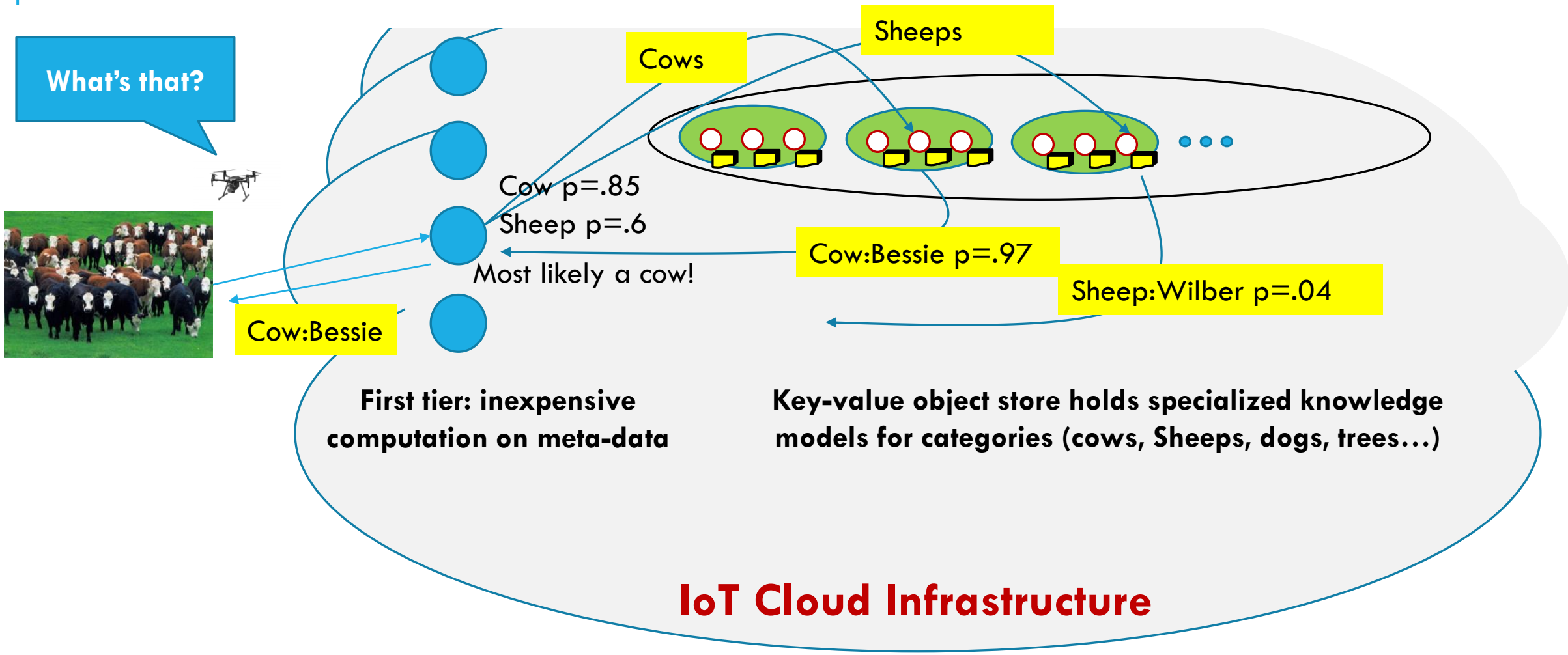
A DRONE, AN IOT EDGE, AND A CLOUD

Internet link (might not always be up, and may be slow).

Full cloud datacenter

IoT Edge Server runs a stripped-down set of cloud functionality, but close to the device.

EXAMPLE WE WILL REVISIT OFTEN



UNFORTUNATELY… THIS IS OVERSIMPLIFIED

This animation showed the “important aspects” but left out a lot of details.

Let’s look at it a second time and show some of the missing parts.

SUPPOSE A COW STUMBLES. IS IT HURT?

Image upload path

Azure IoT Hub

Azure Functions

Blob Store

NoSQL Database

Image processing engine

. . .

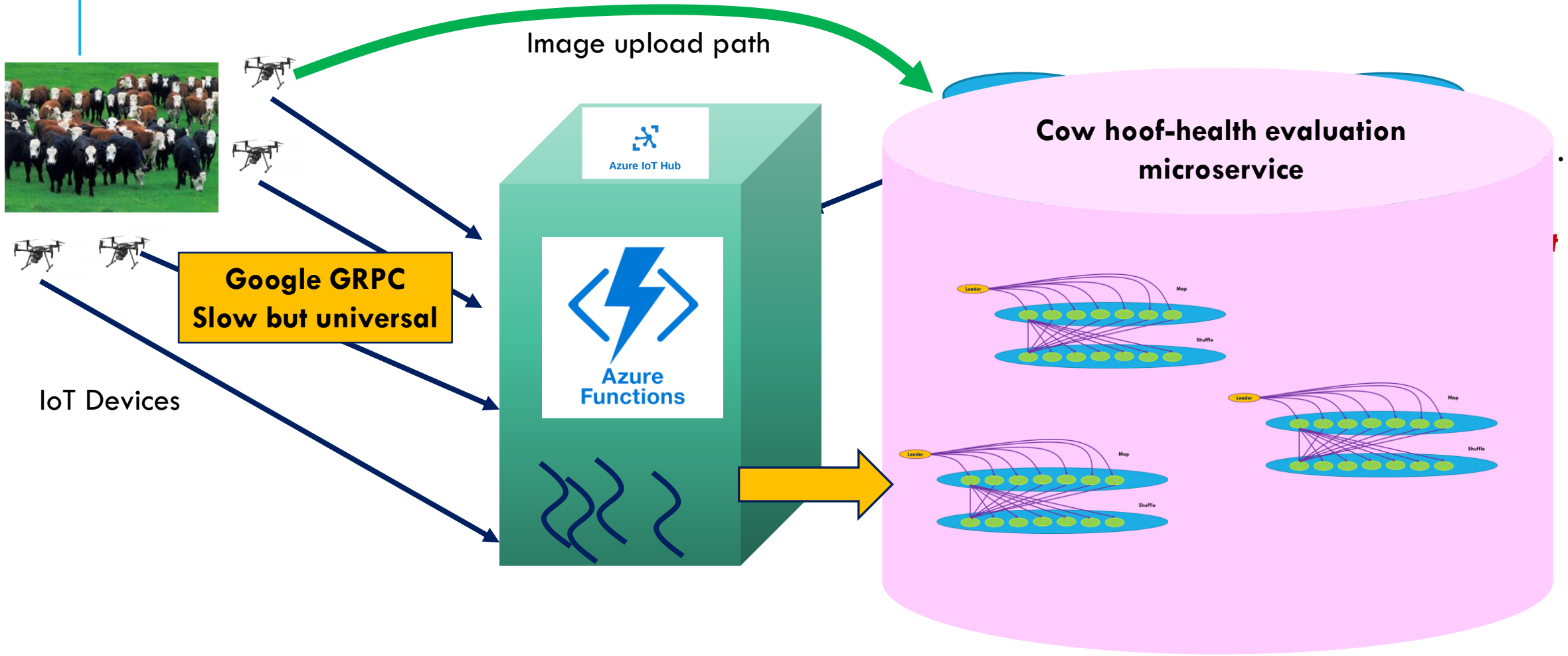
Vendor-supplied services (fairly rigid)

Google GRPC
Slow but universal

IoT Devices

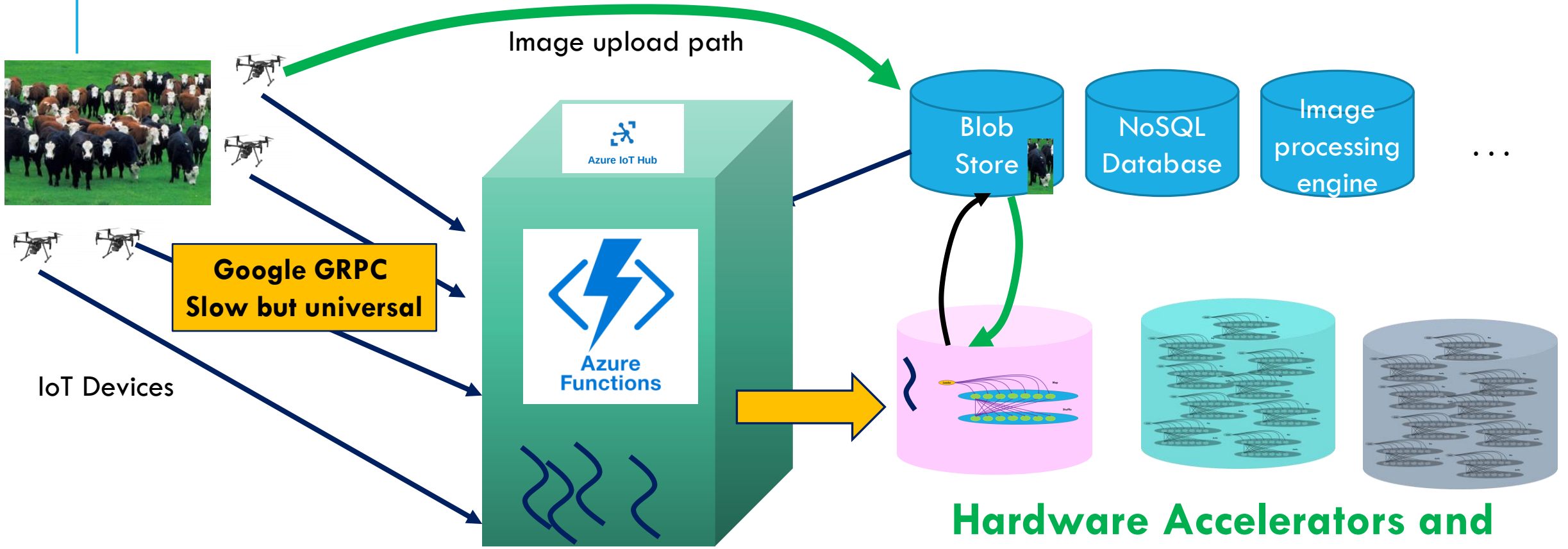
No Hardware Accelerators

SUPPOSE A COW STUMBLES. IS IT HURT?



Ken's research: Tools to build these smart μ-Services

REVISITING OUR EDGE IOT EXAMPLE



Hardware Accelerators and Machine-Learned Models Available. Managed by the "App Service"

MANY STEPS INVOLVED!

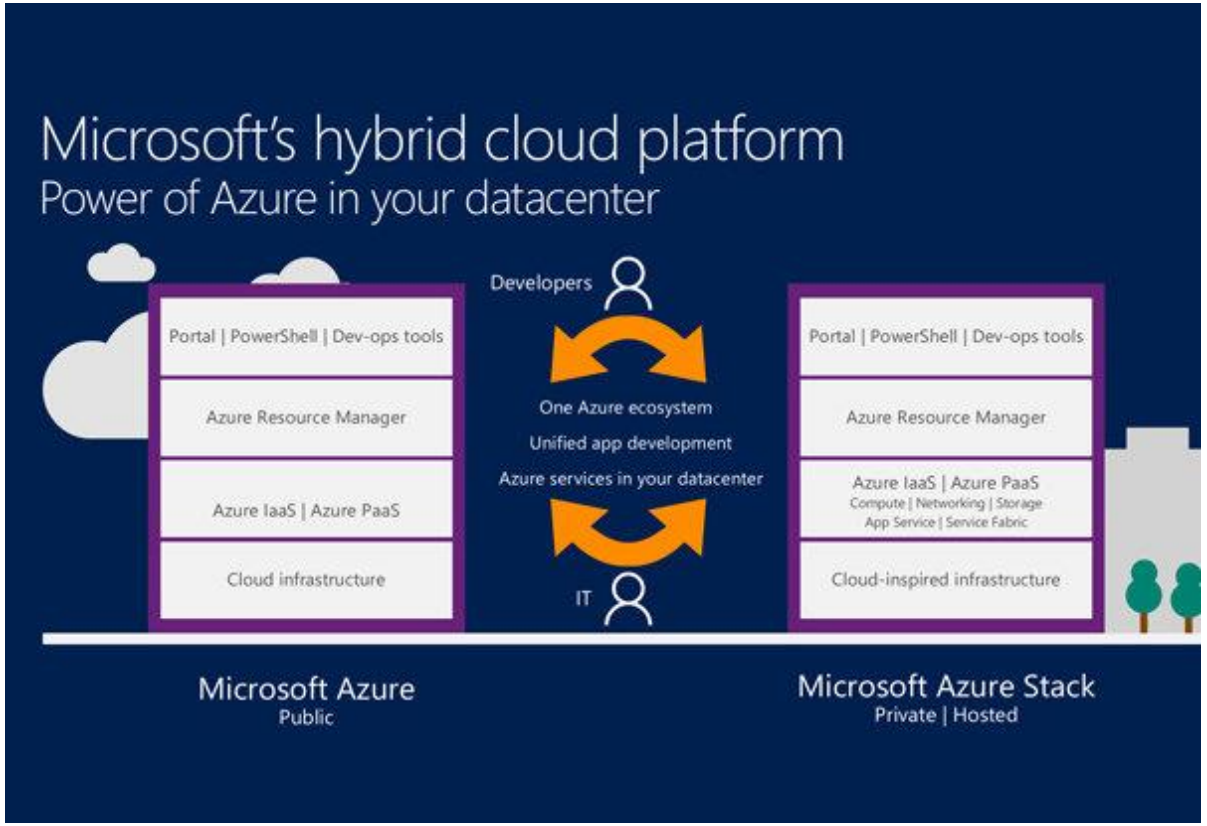
We needed to send a drone to watch the herd. Fields are big… it had to find them first.

It figured out which cow is which

It noticed that Bessie walked unevenly

It asked “Is she hurt? Or was she just walking on rough terrain?”

We decided to call a vet and have her check Bessie’s right front foot.



MANY COMPONENTS WERE INVOLVED

Azure Blob Upload: For putting photos into the Binary Large Objects store.

Cosmos DB: A general-purpose NoSQL database with "processing" power.

Azure image processing service: Can do many "photoshop" tasks.

Azure IoT Hub: Secure connectivity to our drone.

Azure IoT Edge: Wasn't even shown, but it was on the prior slide ("mini-cloud close to the IoT device or drone").

Azure Function Service: Lightweight container launching.

Specialized µService with GPU accelerator: You can build these and manage them with the Azure Hybrid Cloud layer, which includes the Azure App Service.

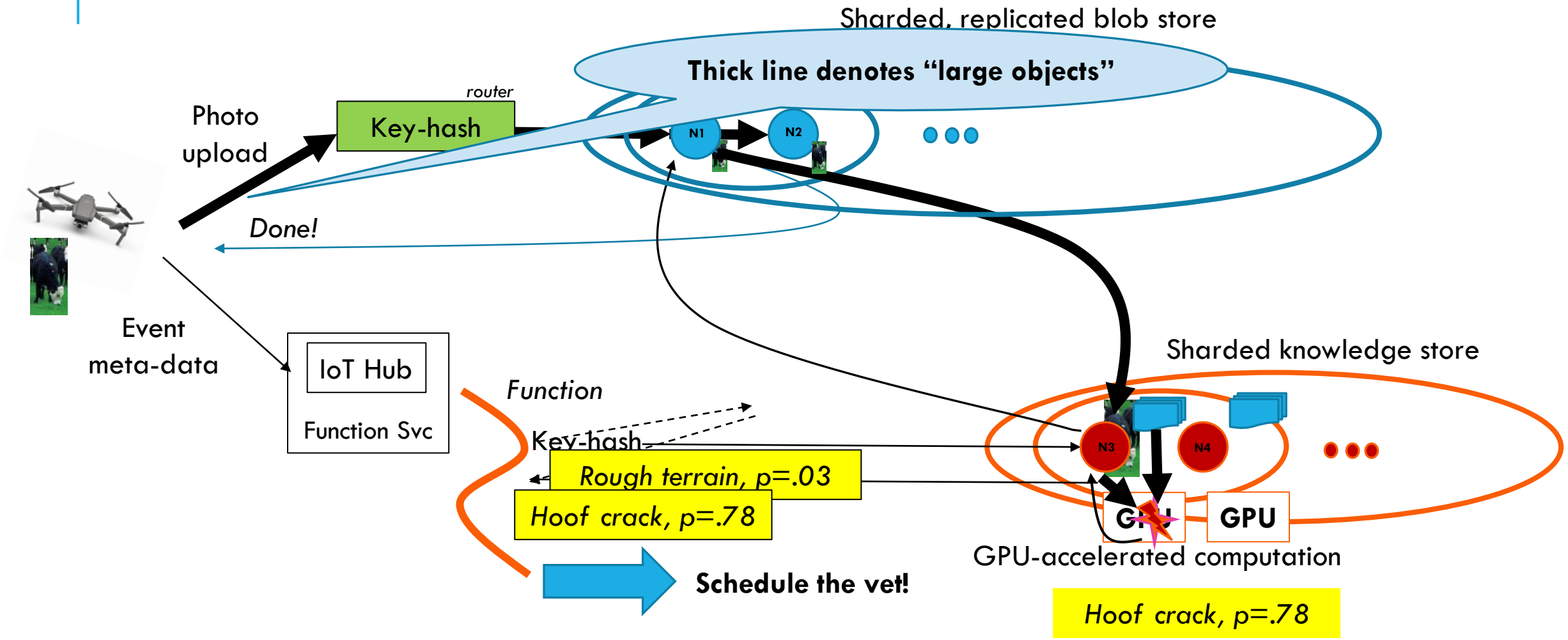
CONCEPT: CRITICAL PATH

The pathway in your system that shapes performance for some task.

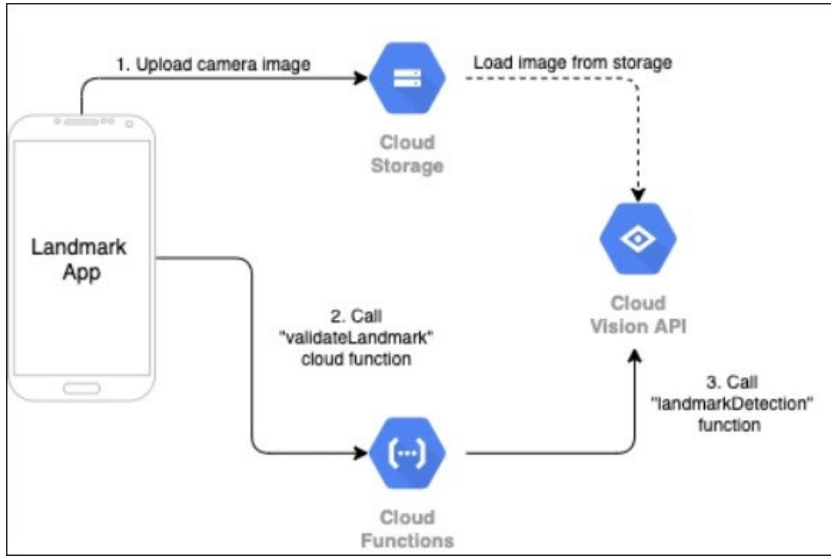
If you make the critical path faster, you accelerate the task.

But there might be a second critical path just a tiny bit slower than the one you are focused on, so fixing one might just reveal the other.

CRITICAL PATH? MANY ELEMENTS!



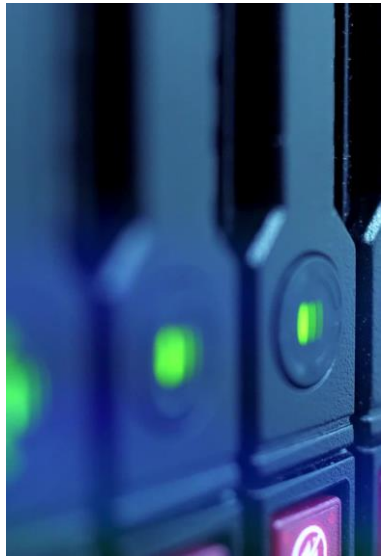
FUNCTION MODEL



Each function is a small program that will be launched with arguments extracted from the event.

The function runs on some machine selected by the Function server, which has a pool of machines that it manages elastically.

To make things simple, the function and any files it needs are wrapped up into a container: a kind of virtual machine, very cheap to launch.

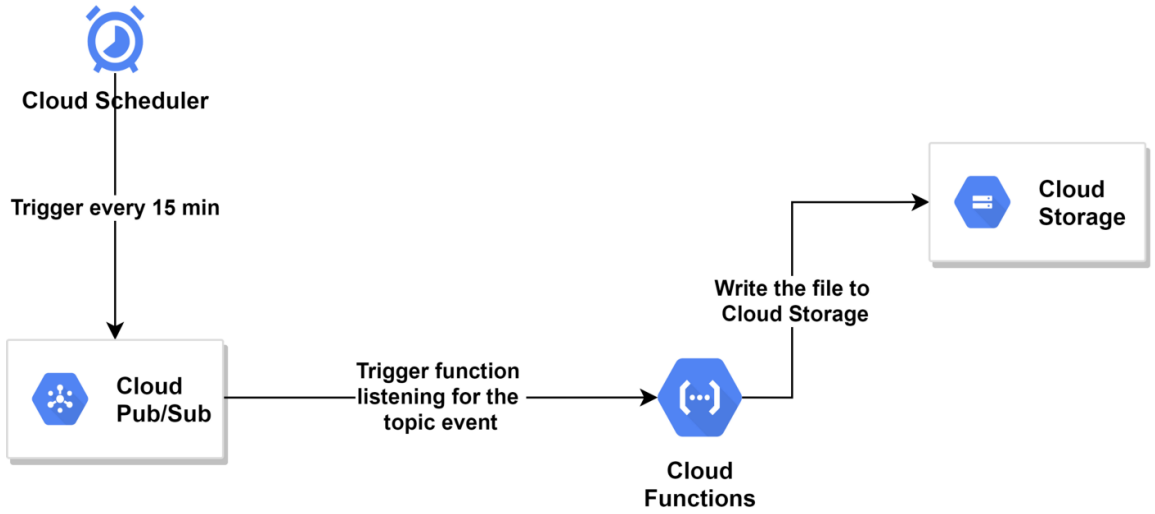


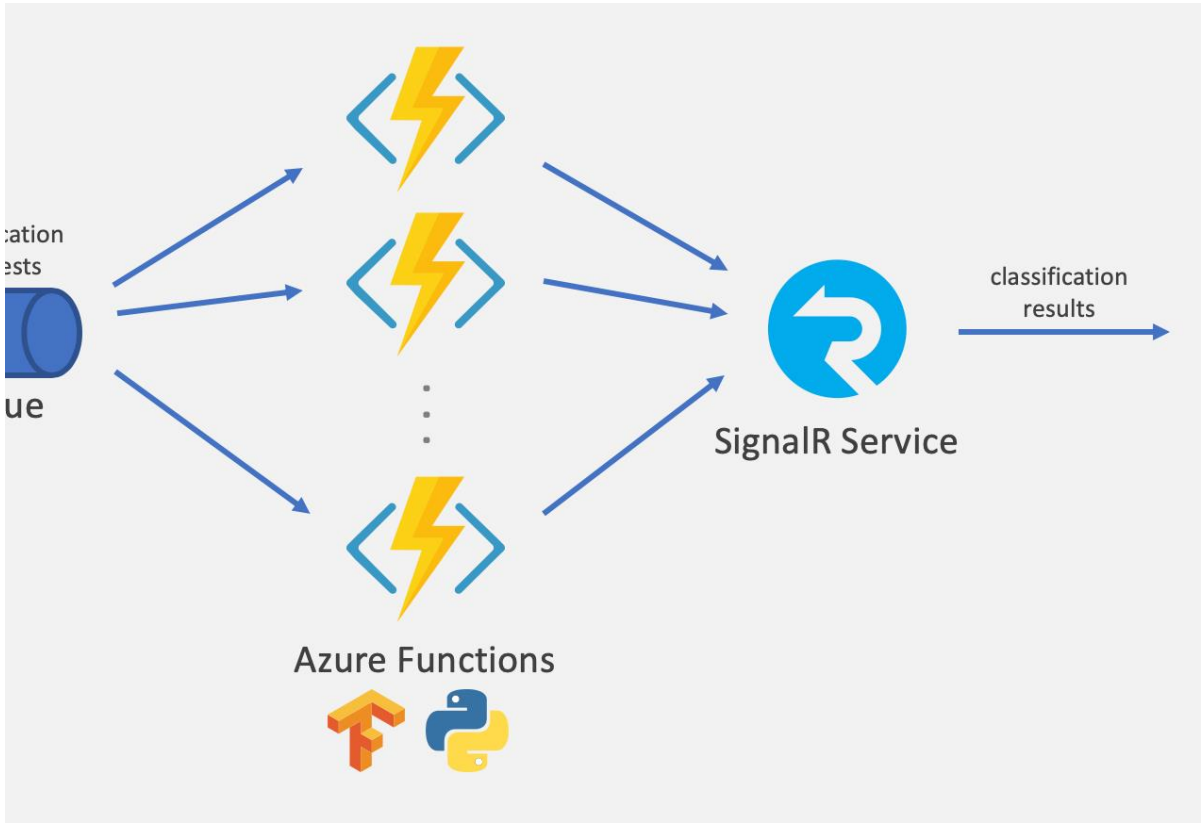
HOW THE EVENT IS PASSED TO THE FUNCTION

When an event occurs, a new instance of the event handling function you registered will be launched in a “clean” state.

The event itself is available either as program arguments, or via an API

You can also register a shell script if you wish.





EXAMPLE: PASSING PARAMETERS TO AZURE FUNCTIONS (AWS LAMBDA IS QUITE SIMILAR)

In Azure, a function *trigger* defines how a function is invoked. A function must have exactly one trigger. Triggers have associated data, which is usually the payload that triggered the function.

Input and output *bindings* provide a declarative way to connect to external data or μ-services from within your code. Bindings are optional and a function can have multiple input and output bindings.

Triggers and bindings let you avoid hardcoding many details that would involve complicated "boilerplate." You can arrange to receive data (for example, the content of a queue message) via parameters in the trigger.

WHAT IF A DEVICE CAN GENERATE MANY KINDS OF EVENTS?

A single "function program" will handle all of them:

switch(event-type) { …. }

The event type would be passed as one of the event parameters. This way there is still just one trigger for the function.

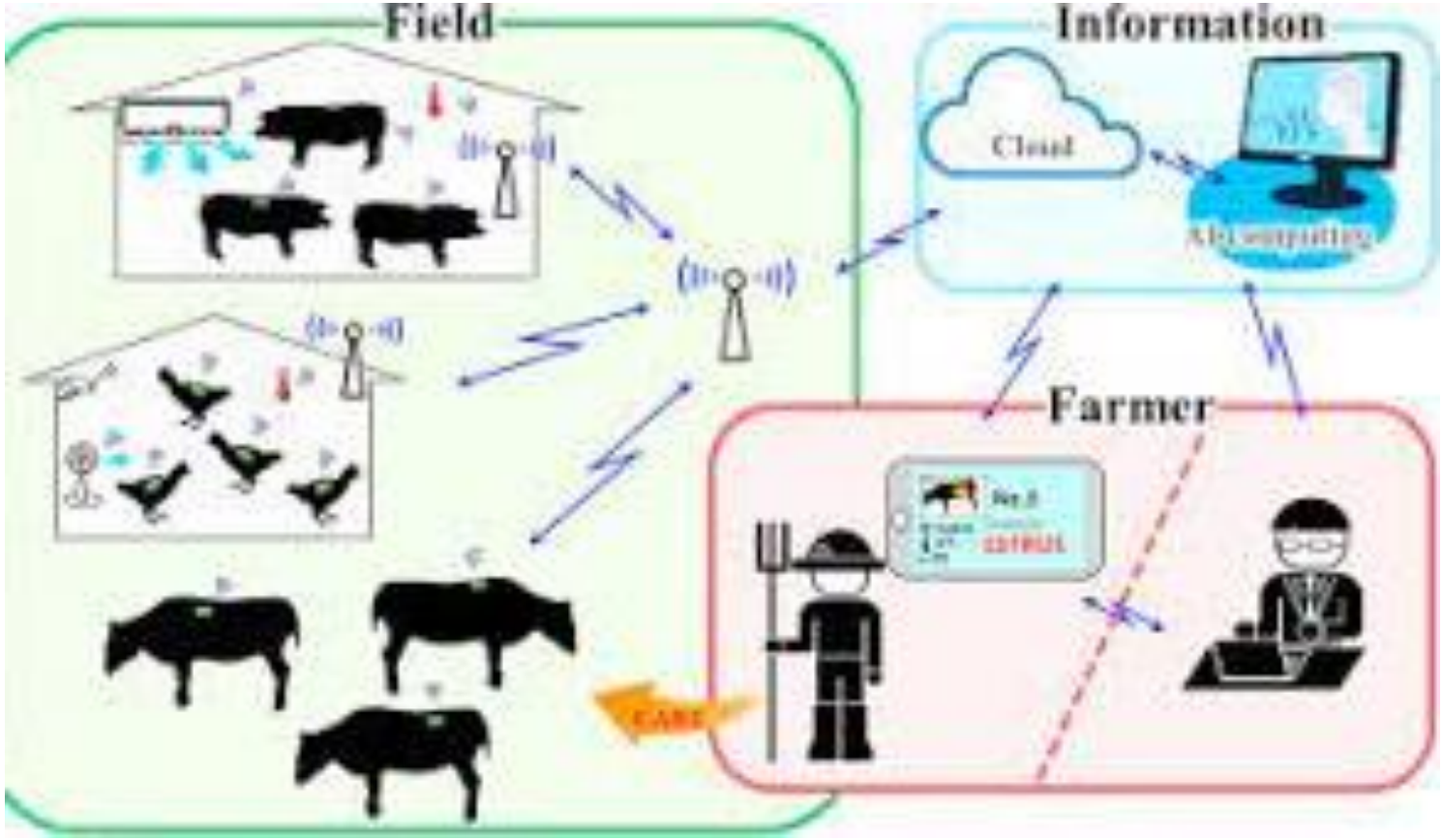
Your logic for dealing with a single event should be short and simple.

HOW DO FUNCTIONS TALK TO μ-SERVICES?

Just like with web pages, there are three main options:

- Remote method invocation (for example over the RESTful RPC layer, or JNI, or WCF). Google GRPC would work here too, but Microsoft prefers for you to use Azure's own solutions.
- Via a message "bus" (no storage: like a "broadcast")
- Via a message "queue" (stores messages, like an email)

Use the remote method approach for immediate actions with immediate responses. The other two "decouple" the source and receiver.



FANCIER CASE: POINT, FOCUS AND SHOOT

Suppose that some event occurs: “Animal motion detected”.

This might require us to swivel the camera or point the drone.

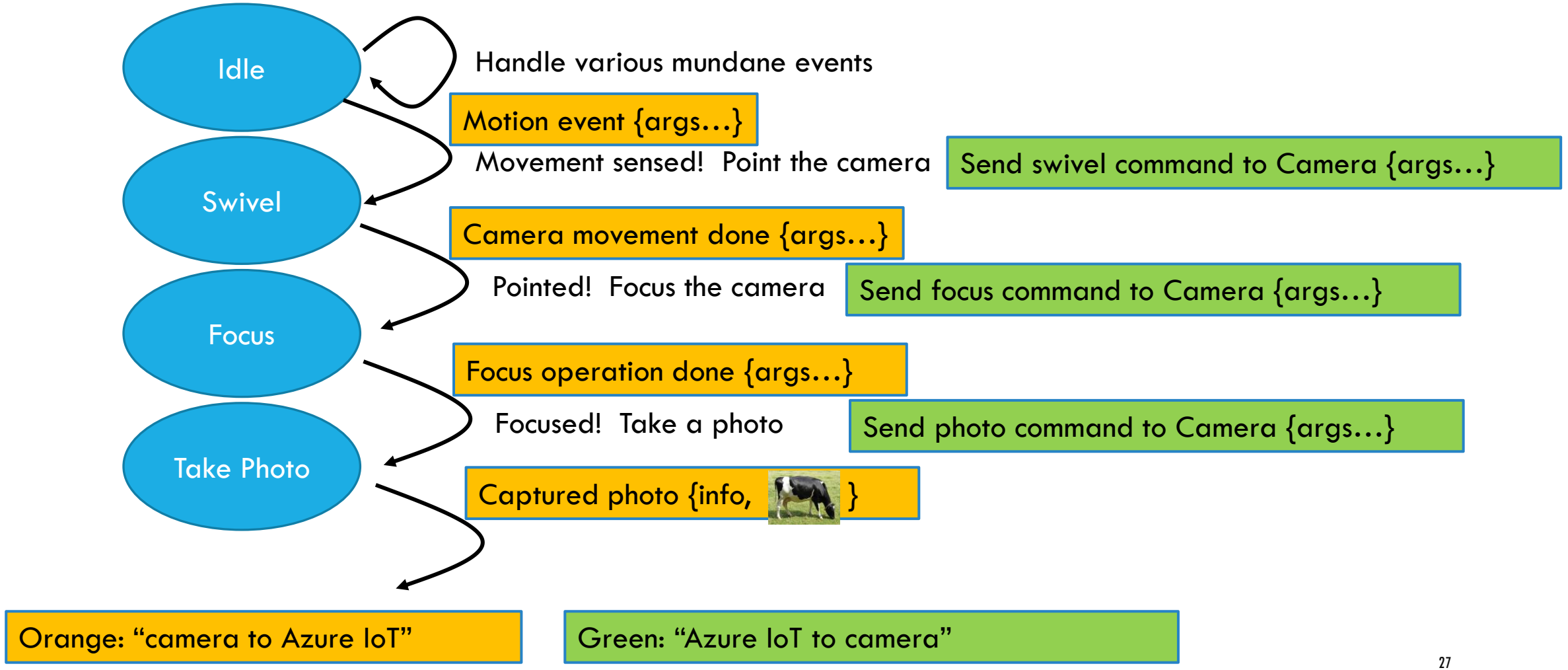
After the camera is pointed towards the location, focus the lens.

When the focus converges, we shoot a photo.

Now the thumbnail is sent to the server.

If the photo is considered interesting, we’ll download it.

THIS IS A FORM OF STATE MACHINE!



DEFINITION: STATE MACHINE

A state machine is a program that is in some "state", corresponding to the nodes in the figure. The states form a directed graph.

Events cause some action (label on the arrow) and also a transition to the same state (loop back) or some other state.

In an IoT setting, we favor *deterministic* state machines: The same events, handed to the state machine in the same state, produce the same effect.

… THAT STATE IS HELD IN A KEY-VALUE STORE

The function loads it, mutates it, then stores it back.

An atomicity issue arises if concurrent events trigger two functions that try to make conflicting updates to the state machine state.

If we solve this by adding locks to the key-value store, Jim Gray's scalability warning applies! To avoid locks, we use a kind of "atomic" *conditional key-value put*.

STATEFUL BEHAVIOR WITH A FUNCTION NORMAL CASE

New Event

1

Function launched to handle it

2

Current State: Data in the (key,value) store can hold any information you like

Version=17

3

4

Updated state replaces prior state (“Replace state version 17 with state version 18”)

5

Triggered action
(issued after successful state update)

STATEFUL BEHAVIOR WITH A FUNCTION CONCURRENCY CONFLICT

New Event

1

Function launched to handle it

2

Current State: Data in the (key,value) store can hold any information you like

Version=17

3

4

Tries to update version 17, but discovers that version has already been changed to 18…

5

Must retry!

STATEFUL BEHAVIOR WITH A FUNCTION RETRY CASE

New Event

1

Function launched to handle it

6

Current State: Data in the (key,value) store can hold any information you like

Version=18

7

8

Updated state replaces prior state ("Replace state version 18 with state version 19")

9

Triggered action
(issued after successful state update)

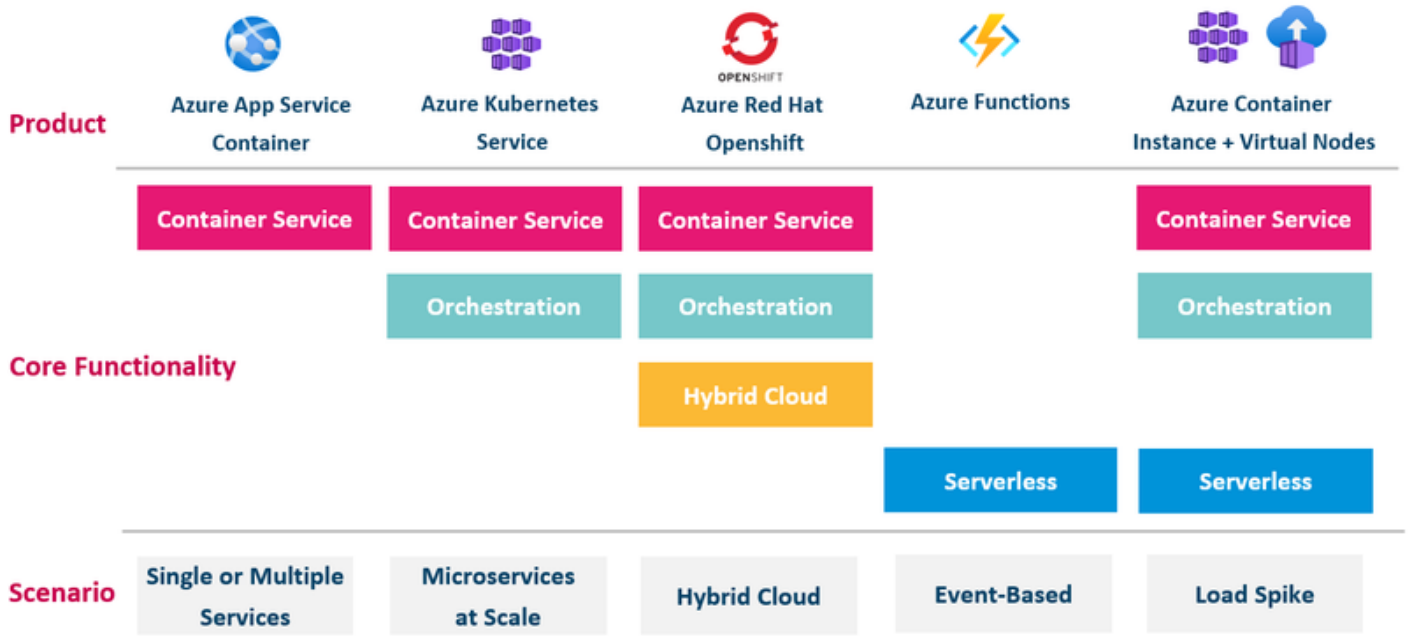
FUNCTIONS VERSUS μ-SERVICES

Use functions for simple read-only actions (the function can still fetch the data from some set of μ-services). Pass updates to μ-services.

Limit multi-step functions to simple state-machine logic.

Use μ-services for complex or stateful tasks.

- Ideally, find some way to leverage existing μ-services. They often have magic superpowers, like access to hardware accelerators.
- Build your own μ-services if there are no existing options that match.



MICROSOFT FARMBEATS

Quick reminder of some smart farming ideas:

- Drones that would survey fields and help with intelligent decisions aboutseed choices, irrigation, fertilizers, pesticide/fungicide use, etc.
- BlockChain style audit trails of actions in dairy or similar situations
- Real-time monitoring of animal health and related tasks, like milking
- Systems to recycle farm waste into useful products like bio-oil
- Maybe a “smart calendar” for the farmer’s wall, showing upcoming tasks, explaining the reasoning, like an iPad but for the farm

https://www.microsoft.com/en-us/research/project/farmbeats-iot-agriculture/

MICROSOFT FARMBEATS

Farmbeats was a research project aimed at prototyping solutions to those kinds of problems.

It evolved and became a new product from Microsoft – a kind of "app platform" that runs on Azure IoT Edge and IoT Cloud and is intended to support more and more farming use-cases over time.

There is some duplication of functionality because Farmbeats existed before they ported it to start to use more of Azure cloud's IoT approach.

https://www.microsoft.com/en-us/research/project/farmbeats-iot-agriculture/

MICROSOFT FARMBEATS

https://www.microsoft.com/en-us/research/project/farmbeats-iot-agriculture/

We can actually use this platform to work with drones

It has great software for drone flight control, photo upload, other similar tasks.

Microsoft has used to build up soil “maps” showing humidity and other important properties for farms. Then, using NOAA weather databases, we can predict how the farm conditions may look over the coming months and event select seeds parcel by parcel to optimize for the specific setting.

WHERE DOES COMPUTING OCCUR?

IoT Edge inside!

The offline training involves “big data analytics” and need to be done on massive data centers with huge compute and storage resources.

But the dynamic form of control and learning needs to occur in real-time, on the **IoT Edge**: a cluster of computers “near” the farm.

The IoT Edge system might dynamically update a model that was mostly created offline on the IoT Cloud, but still needs additional “tuning”

DYNAMIC UPDATES

What would be examples of dynamic updates?

The drones will discover today's wind patterns and output a learned model that they steadily refine as they scan the field.

The drones may discover a very dry area, or a muddy one. Crop issues in that whole area would probably be associated with irrigation issues, even if they "show up" as brown spots, or as fungal breakouts on the leaves.

USING DYNAMIC UPDATES TO UPDATE PLANS

With dynamically learned updates, a control system might realize that it can triple battery lifetime by switching to a new drone flight plan that sails on the breezes in a particular way.

So here we would have a system that recomputes the flight plan, uploads the new plans (but without activating them), then tells all the drones to pause briefly, then allows all to start using the new plans.

Question: *Why upload, then pause, and only then switch to the new plan*?

… BECAUSE WE PREFER NOT TO SEE THIS!

Still using search plan A

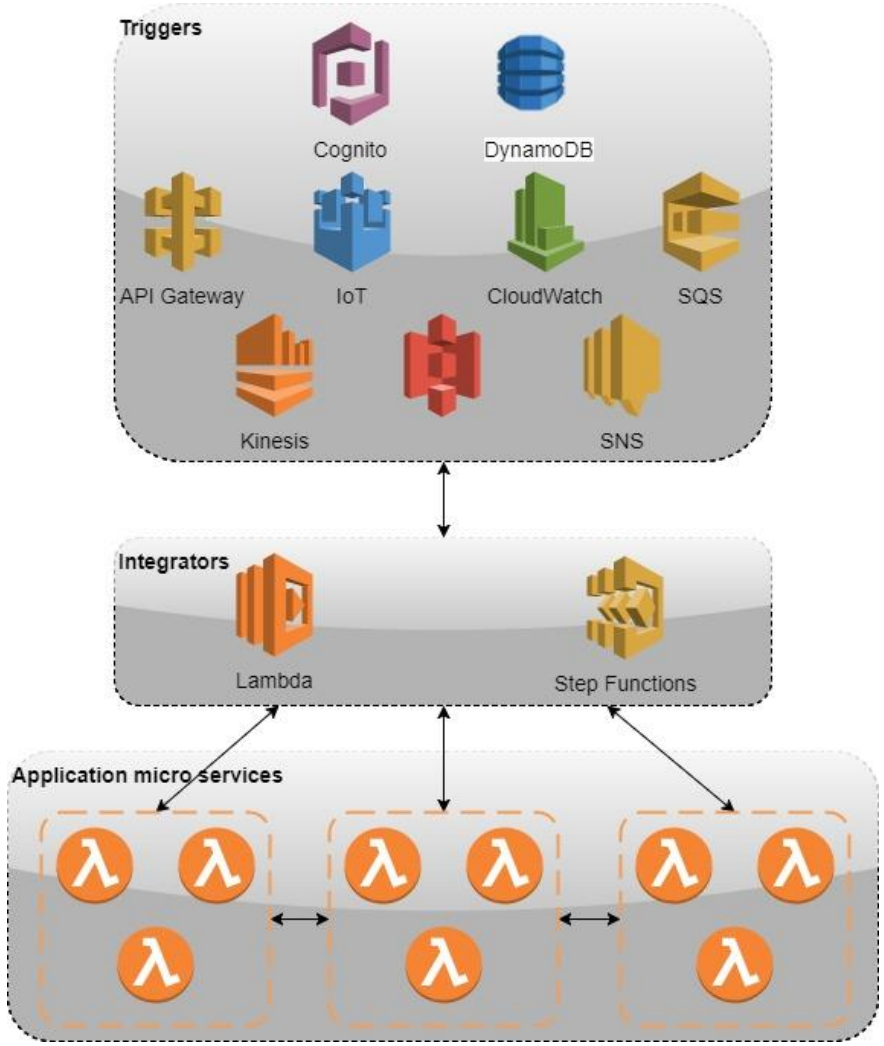
Starting to use insecticide spraying policy from plan B

FUNCTIONS? OR μ-SERVICES?

We actually could implement everything as a giant state machine with a large amount of state in our Azure key-value store.

But would that be the best plan?

- It might be very hard to debug such a complex function application.
- The logic itself might be very complicated, especially since everything will be event driven.
- As we "learn current conditions" we run into a big-data problem.
  A function server isn't intended for such cases.



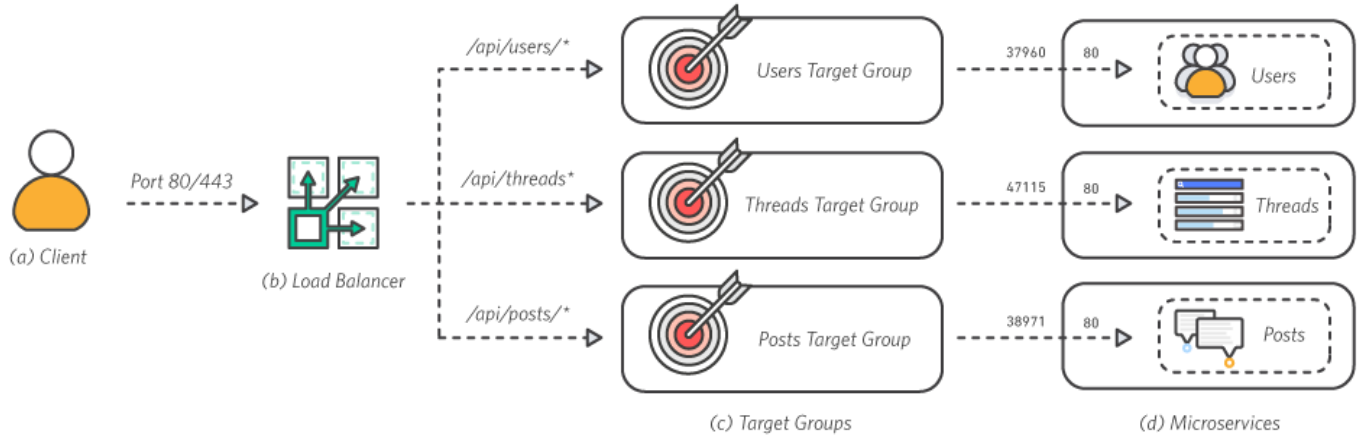
SHOULD EVERYTHING BE IN μ-SERVICES?

Historically this was the most popular approach.

But we end up with ultra-specialized services, and they run all the time, so they might not be very cost-effective.

The nice feature of the function model is that it offers such a simple way to handle large numbers of events elastically.

APPROACH THIS LEADS TOWARDS



Use the functions for “lightweight” tasks and actions

- Ideal for read-only actions like making a quick decision
- OK for reporting events that go into some kind of record or log
- But don’t use functions for serious computing.

Then build new μ-services for the heavy-weight tasks, like learning a new machine-learned model, or computing the optimal search path with wind.

SUMMARY OF THE PROS AND CONS

	Functions	μ-Service
Length of a typical "action"	Typically a single "RPC" or some other event from a client or sensor. Execution time is often very short: milliseconds	Long-running, could continuously evolve some form of knowledge base using background computation that might be quite slow/costly.
Long-term state	Lives outside the functions, like in a key-value store	Could be in memory, or in local files, or could be in other μ-Services.
Resource footprint	Long-term state is small, function itself runs in a lightweight container	Long-term state might be huge, computation runs on heavier-weight compute nodes dedicated to the role for long periods of time
Access to accelerators	Probably not.	If needed, yes.
Cost to own & operate	Pay only for cycles you use.	Can be very costly, but amortized over many clients.

HOW TO CREATE NEW FUNCTIONS

Register the corresponding event (or class of events).

Tell the function server to run your container for the specific events it will handle.

Develop code using cloud-vendor supplied tool that will provide a skeleton. You might write just a few lines to specialize it for your events.

HOW TO CREATE NEW μ-SERVICES?

Architecture can be fairly complex, so you'll start by really thinking hard about functionality, data representations, API.

Many services have a non-trivial internal structure: a top-level group but with several subgroups inside it, playing distinct roles.

Usually developed on a cluster of Linux servers using libraries that help with hard aspects.

HOW TO CREATE NEW μ-SERVICES?

We can start with Jim Gray's suggestion: key-value sharding from the outset.

Within a shard, data will need to be replicated. This leads to what is called the "state machine replication model", which involves

- A group of replicas (and a *membership service* to track the set)
- Each update occurs as a message delivered to all replicas
- The updates are in the identical order
- No matter what happens (failures, restarts) "amnesia" won't occur.

WILL THIS SCALE?

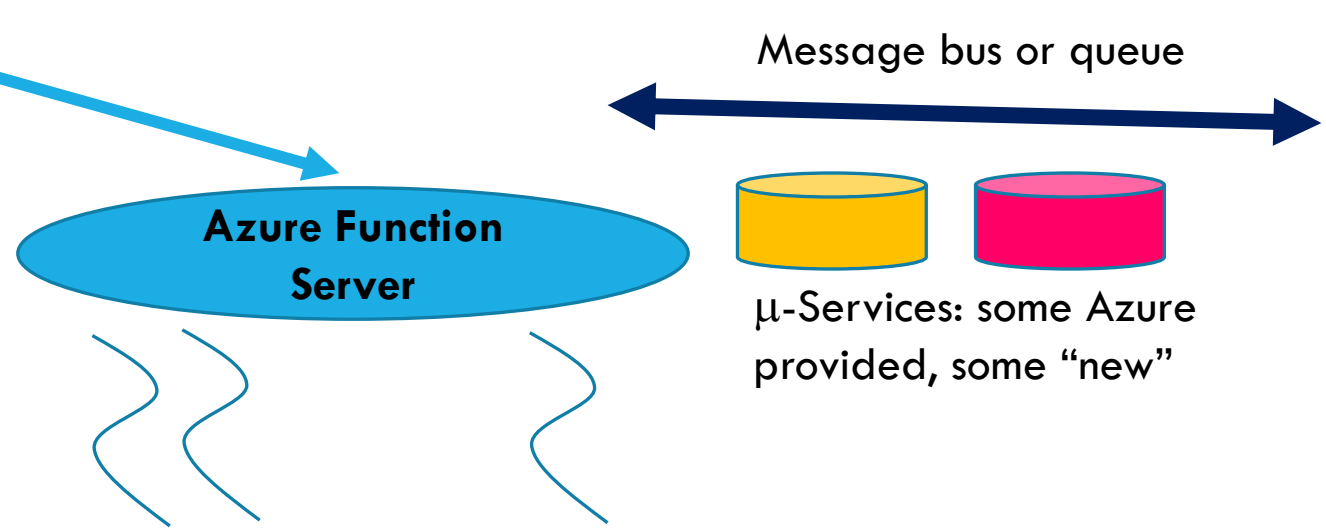
Jim Gray's analysis told us that general database transactions won't scale.

But this simple key-value approach would scale very well provided that updates and queries run on a single shard at a time.

This was a sweet spot in Jim's model.

SO, BACK TO OUR FARMBEATS DRONES

https://www.microsoft.com/en-us/research/project/farmbeats-iot-agriculture/



Functions: Lightweight, event-triggered programs in containers, "pay for what you use" resource model

HOW MIGHT WE TACKLE THE CASES MENTIONED EARLIER?

Consider one example:

"Image analysis: "Are these plants healthy or diseased?"

How might we solve such a problem using modern machine learning?

How would we turn our solution into a μ-service?

How would a function in a function server interact with it?

IMAGE KNOWLEDGE BASE

We could start with labeled data: photos from drones that are hand-labeled to tag crop damage and identify possible causes.

Use this to train a computer-vision model (perhaps, a convolutional neural network – a CNN).

The resulting models will be large tensors. Copy them to our μ-service.

IMAGE KNOWLEDGE

We might have two cases: one for initial thumbnail images (small, low-resolution) and a second for follow-up detail imaging (ultra-high resolution)

Now our μ-service could have an API with operations such as “classify new thumbnail”, “analyze follow-up imagery”.

The function server would take a drone event and just turn around and make a call into the μ-service

SECONDARY ACTIONS

The μ-service would then be able to "tell" the function what action to take. This avoids having to talk directly to the drones: the functions become specialists in drone operations, while the μ-service plays general roles.

Similarly for requesting "follow-up detail": the μ-service can request this in its reply to the function layer, and then the function would turn to the μ-service that plans detailed imaging studies for advice on camera angles and image settings to use.

Functions aren't doing much, but they glue the heavy lifters together.

SUMMARY

An IoT system like a smart farm has a lot of “moving parts”!

Even so, we can break the tasks down and map them to a μ-services model, using a KVS to store any needed state.

Some tasks look tricky at first, but once you see how key-value *versioning* works, it enables exactly the kind of atomic operations we seem to need.

READING FOR THE WEEK

1. https://www.libelium.com/libeliumworld/success-stories/how-a-dairy-farm-increased-their-milk-production-18-with-iot-and-machine-learning/
2. https://www.infoworld.com/article/3646251/the-real-value-of-5g-and-cloud-computing.html?fbclid=IwAR2qapuTUnpWQEmMhQdnL5XqyyB8yigX1pEVk0ON0Uci_KqnQ39IcMuyjt8
3. https://www.microsoft.com/en-us/research/project/farmbeats-iot-agriculture/