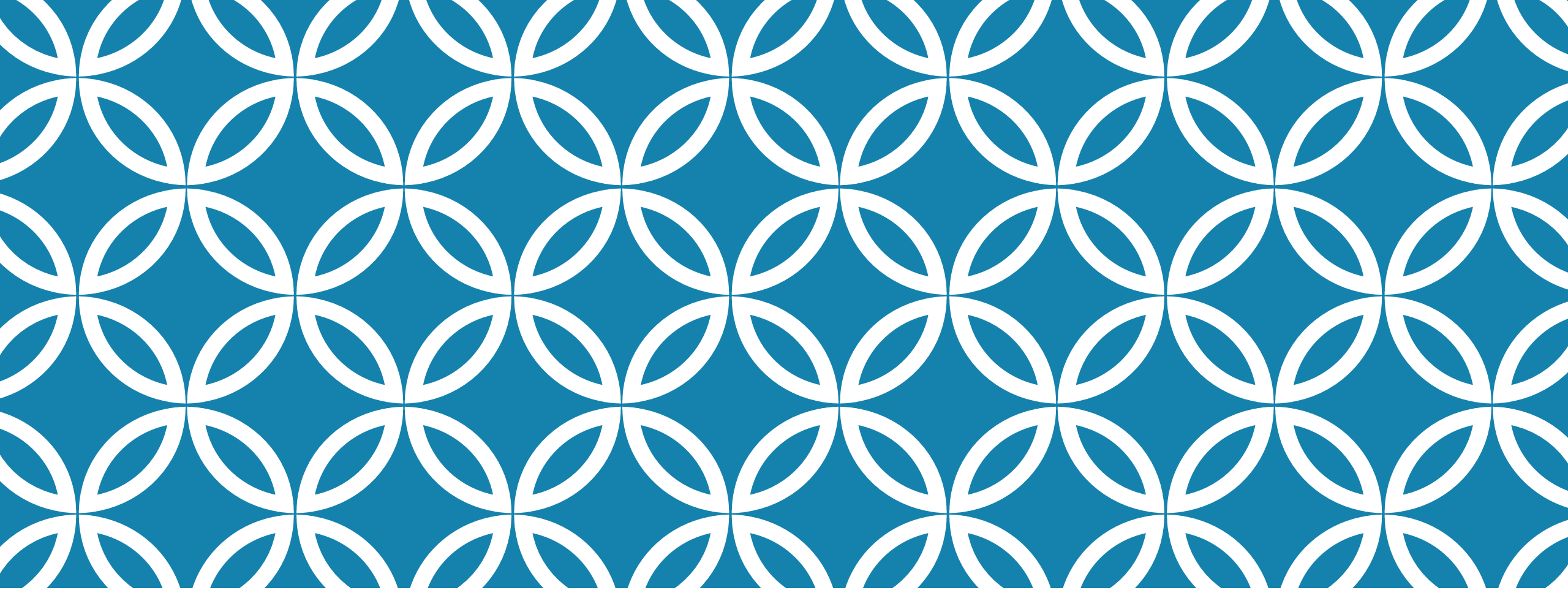
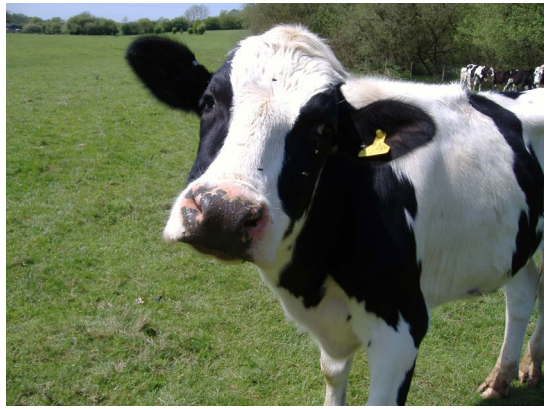


CS5412 / LECTURE 9
MACHINE LEARNING FOR SMART FARMS

Ken Birman

Spring, 2019

WE HEARD ABOUT AIR TRAFFIC CONTROL…



Can we apply our insights in other settings?

Let's review some smart-farming scenarios.

Goal is to see if they more or less map to this model with sensors, Function Server running stateless functions, collection of snappy μ-services that can be stateful and include machine-intelligence components.

THE BIG BET: IOT CAN RESHAPE THE WAY MACHINE LEARNING IS DONE

Machine learning for IoT settings has demanding time deadlines not seen in traditional cloud systems. Moreover, the amount of data on the IoT devices could be vastly more than we can hope to download.

Our goal today? To understand the resulting *flow* of data/computing.

- Data sets are so large in these settings that only really smart management of flows can yield a good solution.
- This shapes a view focused on the *pattern of computation* in IoT settings.

WHY NOT STICK WITH THE CLOUD "AS IS"?

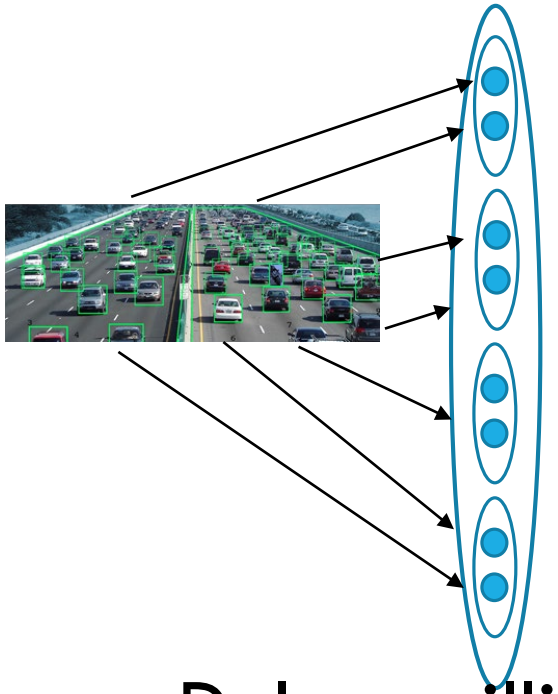
Until now, big data computations have run in big "back-end" systems like the famous MapReduce/Hadoop framework, or high-performance supercomputers.

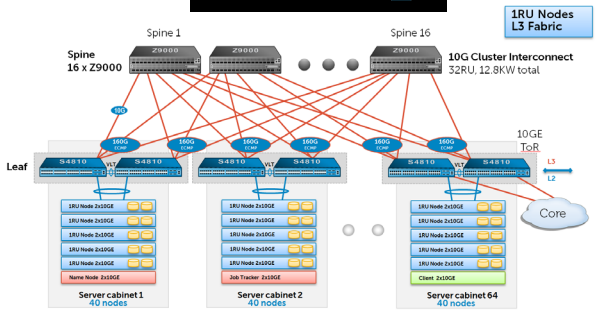
Big data processing was mostly done in batches, offline.

IoT model demands instantaneous mobile intelligence, vision, speech understanding, control of devices. A batched, offline model won't work.

TODAY: A VERY "LONG" PIPELINE

Data acquisition…. Global File System… Hadoop jobs

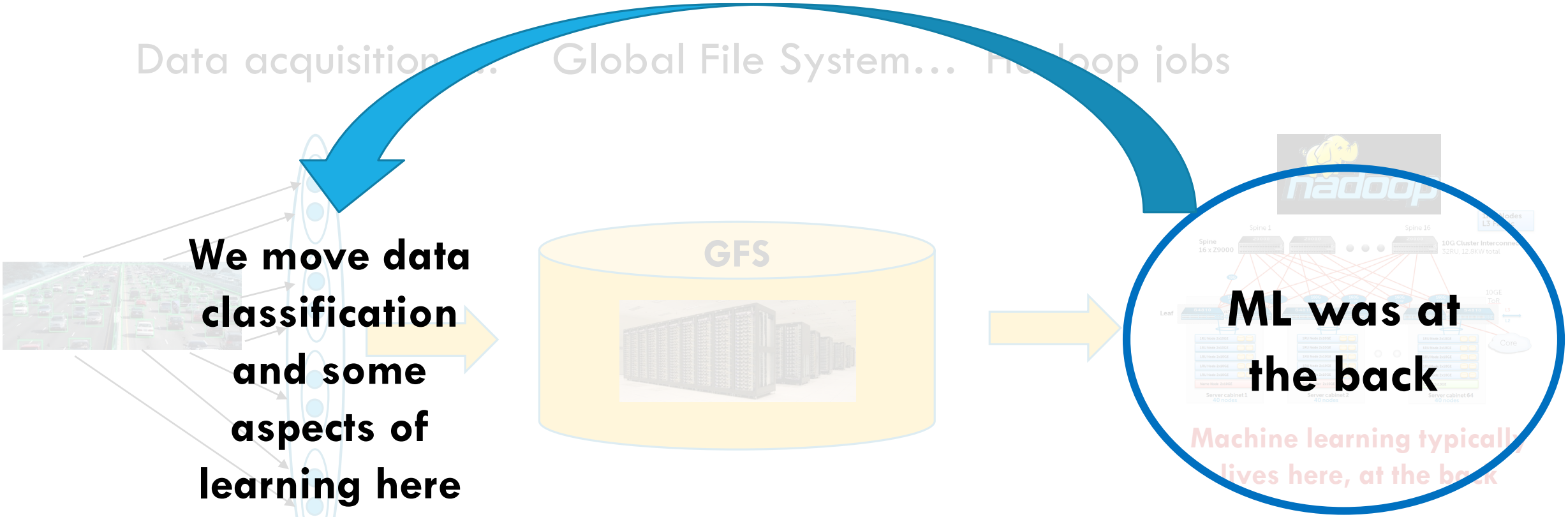




Machine learning typically lives here, at the back

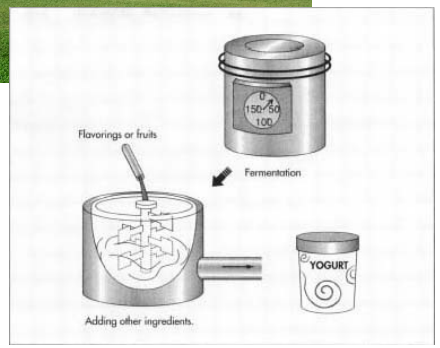
Delay: milliseconds… Seconds…. Hours

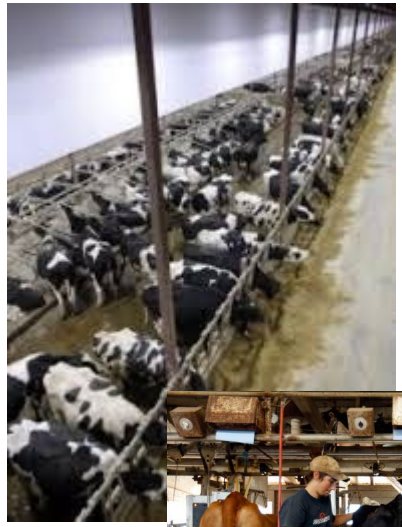
NEW: MOVE ML TO THE EDGE OF THE CLOUD



FARMBEATS: MICROSOFT'S "THINK WITH OUR HANDS" APPROACH

It is hard to just guess, so Microsoft decided to build an IoT solution.





A PRODUCT… AND A PROCESS

Like the French air traffic project, Microsoft has brilliant technical leaders.

They set out to be incremental and only create new things when needed, and to validate each step.

But smart farming also pushes the envelope and challenges them to think outside the standard cloud "box".

SMART MONITORING OF CROPS

Field of oats, or hay

- How is the crop growing?
- Are there signs of drought / insect / virus / fungal / bacterial issues?
  - If so, can we diagnose the exact problem?
  - If we can, what treatment is needed, and exactly where to apply it?
  - Can we learn from this and improve our seed choice for next year?
  - Where should we fertilize or irrigate?

SMART HERD MANAGEMENT

Dairy: Cow health and monitoring

- Which way should we point the camera? When to take photos/video?
- How much milk did each cow produce, and of what quality?
- What did it eat, and how was its appetite?
- How much time did it spend ruminating, or sleeping?
- Which cows need routine medical attention?
- Is a cow close to giving birth? Is it likely to need emergency help?

SMART DAIRY

Milk processing, yoghurt and cheese making

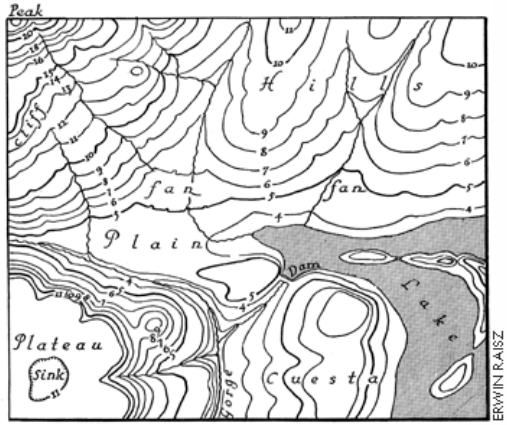
- Must monitor temperature and pH
- Need to sterilize properly using correct strength of product, rinse off
- Watch for stuck or runaway fermentations
- Check samples for unwanted bacteria, like Listeria (very dangerous!)
- Maintain a secure and tamperproof audit trace (BlockChain?)

SMART WASTE DISPOSAL

What about all the runoff and farm waste?

- Why not collect it, reprocess it for valuable secondary products?
  - Manure contains nitrogen and phosphorus can be used to create fertilizer
  - Waste water can be captured and used for irrigation
  - Undigested material can be transformed to “bio oil” by heating at high pressure
  - Residual material after treatment can be composted and plowed back on fields
- Much of the problem with algae blooms could be eliminated by such steps, and farms could also earn more (or spend less) by doing so!

GEEKY STUFF



Recognize cow moods, relate cow emotional state to milk production

Optimize drone flights over complex terrain to "sail on the wind" & save power

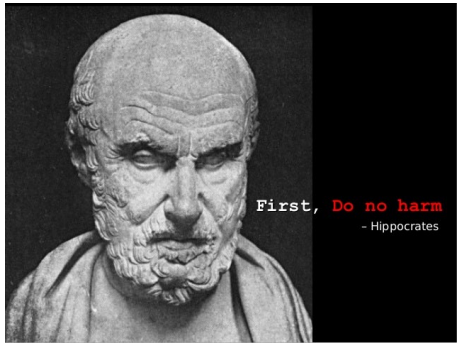
Develop a multispectral image analysis to interpret signs of crop damage

Programming a drone to "look more closely" if needed, like underside of leaves or closeups of blighted ears of oats

Machine learning to estimate crop maturity and schedule equipment for harvesting

Predict the best choice of crop and the specific choice of seeds to plant next year in each parcel of a large field

DO NO HARM!



Smart farming also raises issues of privacy and security:

- Banks and insurance companies might be eager to "see" private data
- There are more and more laws governing food-supply auditing?
- If farms became dependent on IoT, how can we make the technology robust enough for a wide range of conditions (weather, dust, …)
- Farmers aren't hi-tech specialists. How hard will IoT be to maintain?
- Can we create versions for very poor rural areas?

WE NEED TO DRILL DOWN ON A CONCRETE TASK REPRESENTATIVE OF THESE.

In most of these tasks we see a shared structure:

- Start with a problem posed in a real world, like a farm or dairy
- Work to understand the various dimensions, especially scalability issues tied to big data. If we design without scalability in mind, our solution will fail!
- Deploy sensors, then design a state machine that understands the sensor events, platform events, and uses functions to perform tasks.
- Perhaps, develop new elastic μ-services your system will require.
- Debug this on a real system, like Azure IoT

… not an easy job!

CROP MONITORING

Let's focus initially on just one case: monitoring a field using drones.

What major subsystems would we need?

- Mapping system to pull up a topographical map of the field to scan
- Basic drone flight control system to "follow" a flight plan
- Wind sensing and mapping subsystem, to "sail on the breeze" (not fight it)
- Image analysis: "Are these plants healthy or diseased?"
- Close-examination: Visit diseased plants, diagnose issue, document it.
- Data archive: Downloads interesting images/video/etc and retains it.

A QUICK REMINDER

IoT Edge: *Who needs it?*	There might not always be a connection to the cloud, so we run a little "micro-cloud" close to the sensors.
IoT Hub: *Why bother?*	We use the IoT Hub to authenticate sensors, and to make outgoing TCP connections to them.
Functions: *What a nuisance! Dump 'em*	Functions are "unavoidable." This is where IoT events initially show up.
μ-services: *No need… use existing ones*	Do use existing ones! But they may not cover the tasks your design requires.

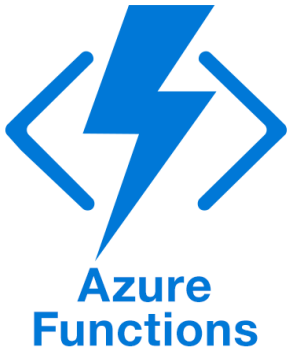
FUNCTIONS? OR μ-SERVICES?

Recall that we have a choice: some tasks should run as state machines, keeping their state in a Azure key-value store.

Other tasks should be implemented by one (or many) μ-Services that would understand our goals and send instructions to our drones. This would feel more like a standard "control center" approach.

In a scaled-out IoT setting, a solution needs elements of both kinds.

FUNCTIONS? OR NEW μ-SERVICES?



Why is it so obvious that this isn't a case for a "pure function" solution?

- What we've described would require an elaborate state machine.
- It might be very hard to debug such a complex function application.
- The logic for each state might be complicated, 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 a common approach: people built specialized control systems and viewed devices as dumb. But few have the skills to pull it off.

In an IoT setting, *massive scale brings massive loads!*

- Any μ-services will need to be sharded, fault-tolerant, highly responsive, and may have to leverage special hardware accelerators.
- If we think of a function layer as a kind of intelligent "cache" that can shield the μ-services from overload, we are approaching this the right way.

APPROACH THIS LEADS US TO?

We will use Azure 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 not for serious computing with heavy computation, big data, accelerators, or complex state machine sequences.

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.

THERE WON'T BE JUST ONE!

Divide the set of knowledge tasks into groups. Don't ask one server to do everything.

Instead build distinct servers for each category of knowledge tasks. So we would want

- One μ-service just for "flight planning", or even two (one for "collision avoidance")
- One for "sailing on a breeze",
- One for "drone health management",
- One for "deciding which photos are worth downloading,"
- One for "identifying possible crop damage areas."

IMPLICATIONS?

Microsoft's IoT users will need help building new μ-services.

By building a few of their own, for Farmbeats drones, the company can explore tradeoffs (like real-time, consistency, where to run the machine learning logic, what can stay in the "back" and what has to be in the edge).

There are also technical platform questions: networking connectivity, what to do right on the farm and what to do in the cloud, etc.

REMEMBER: AMAZON ENDED UP WITH HUNDREDS OF μ-SERVICES / WEB PAGE!

Learn from others who have been down this path before you.

The whole game centers on breaking up the task into chunks that are self-contained, but "small" in scope!

If you think of this as one big monolithic task, you are certain to be doomed by the complexity of the overall undertaking!

HOW TO CREATE NEW μ-SERVICES?

We can start with Jim Gray's suggestion: use 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. So don't even consider our sharded key-value service as a database.

We'll want to aim for simple key-value operations, or small computations that can somehow be made fault-tolerant and atomic without scaling issues.

This was a sweet spot in Jim's model.

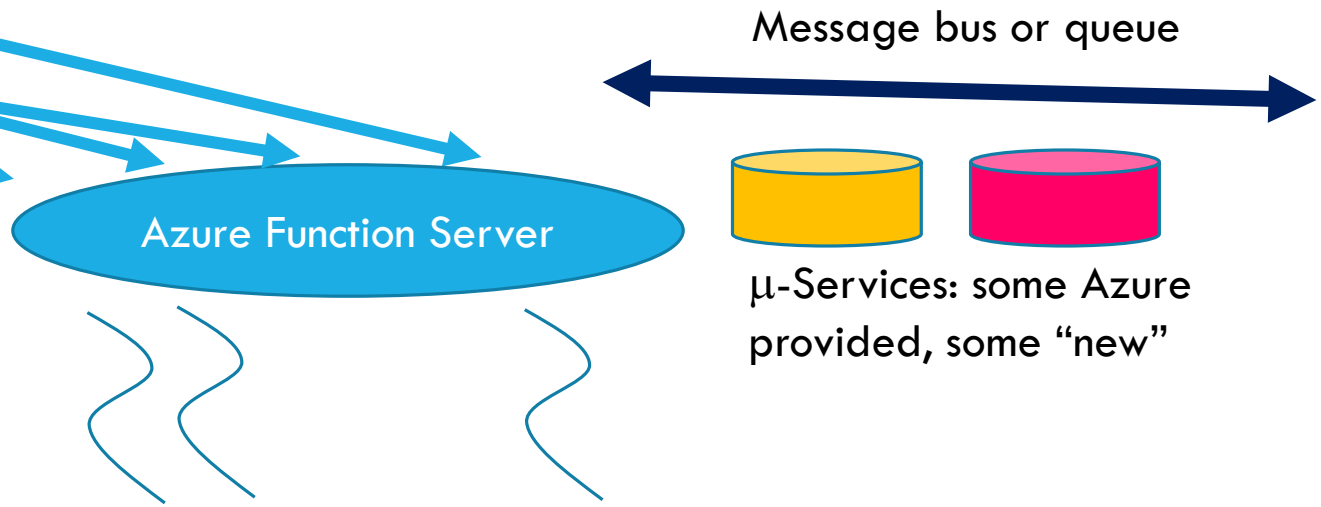
"ALL SHARDED, ALL THE TIME"

In computing classes, we really don't learn to compute on data that is spread over devices.

IoT data will already be sharded when it enters in the system, and all computation needs to be parallel and to keep the work sharded.

Sharding is a magic formula for scaling, but how can people to learn to program in an "all-sharded, all the time" manner?

SO, BACK TO OUR FARMBEATS DRONES

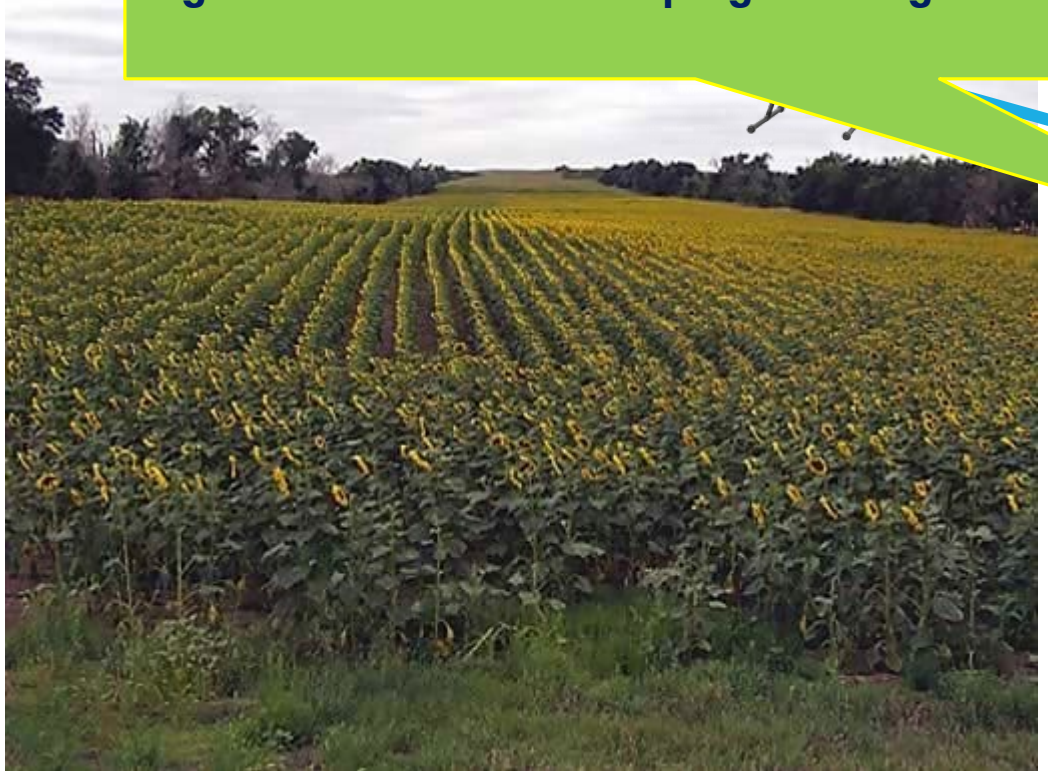


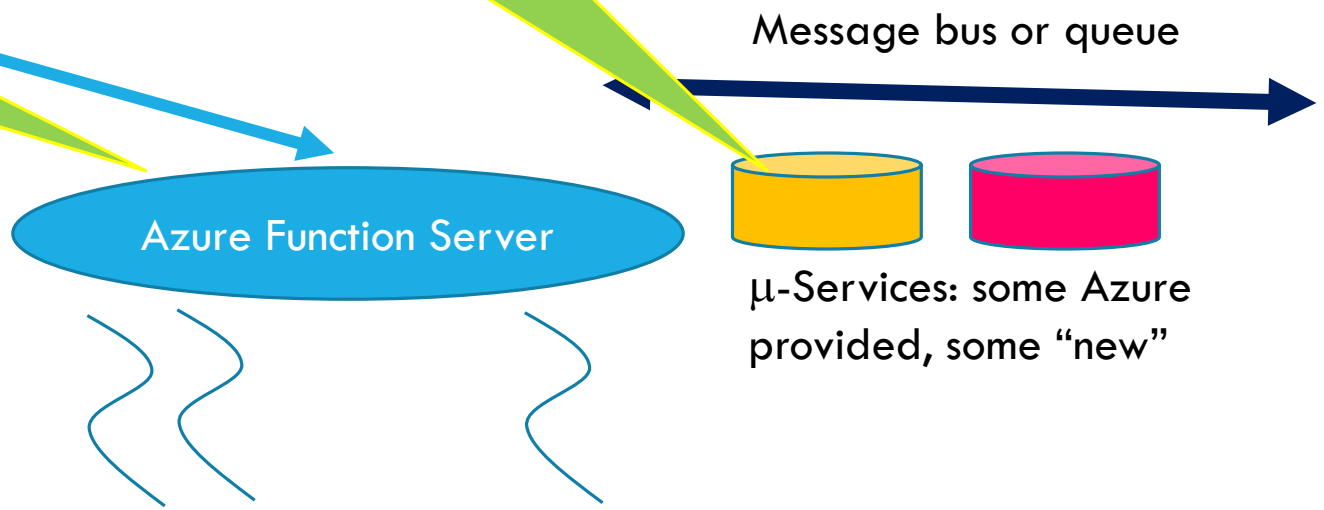
Functions: Lightweight, event-triggered programs in containers, "pay for what you use" resource model

REVISIT OU

A set of μ-services can own many of our other tasks, each specialized in some sub-task. Divide the job up into distinct kinds of work!

Moment-by-moment operation of the drone is a good fit for the function programming model.





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

WHERE'S THE SCALE?

Our example shows just a few drones monitoring on field.

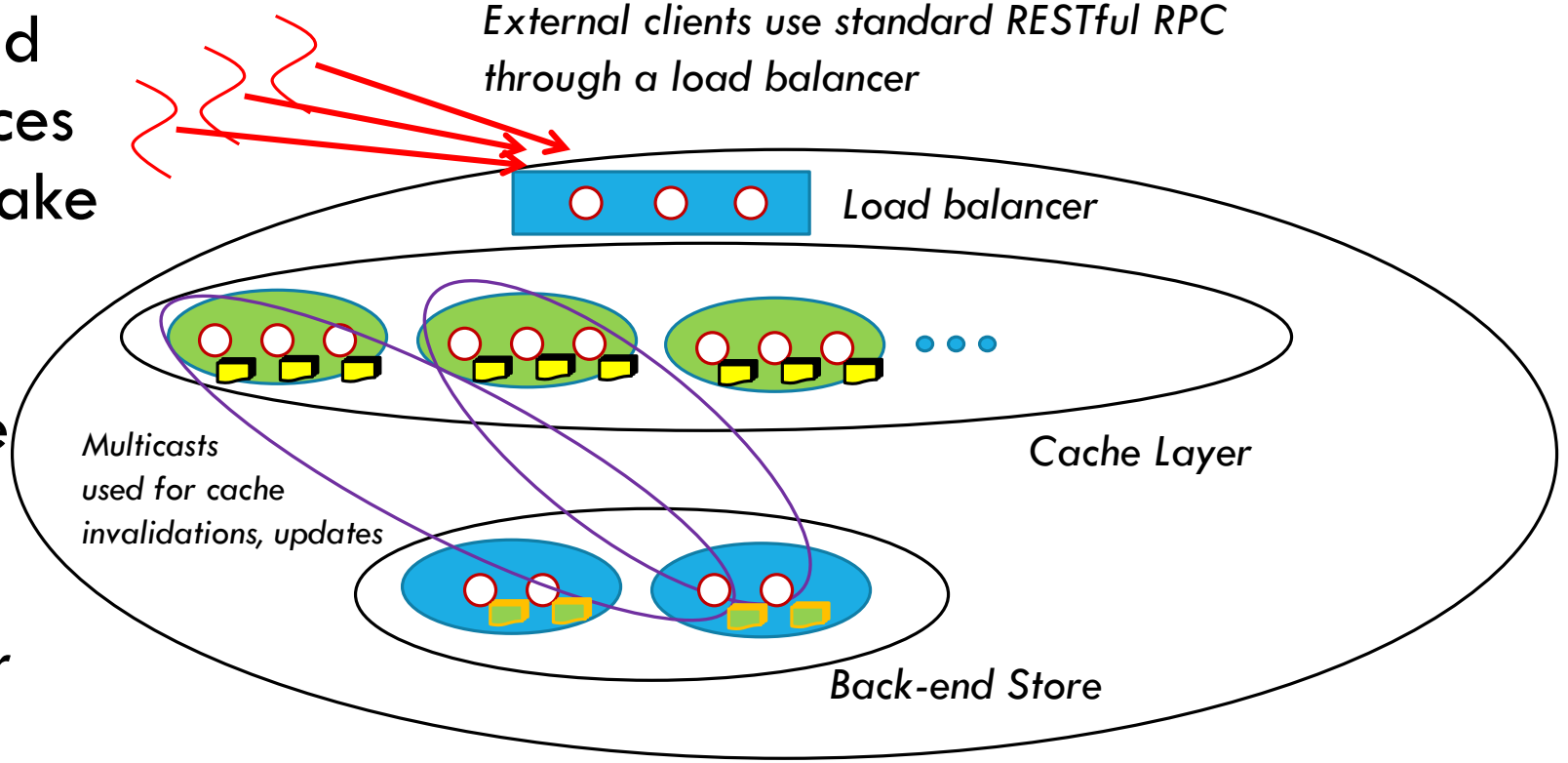
But "at scale" in a full deployment, you want to imagine hundreds of thousands scanning many thousands of fields.

And millions more sensors and actuators playing other roles.

LET'S PEEK INSIDE A MICROSERVICE

The inner structure would depend on design choices the developer would make

This particular example has a load-balancer, a cache layer, and a back-end storage layer



WHY DID THIS DERECHO PICTURE POP UP?

We aren't saying that everyone will use Derecho.

We are saying that everyone will have to create new μ-services and that they will often have an internal structure, like in this example.

So they will need help doing that. Derecho is just one instance of a tool for helping people get these up and running.

A ROLL-YOUR-OWN μ-SERVICE OF THIS KIND MIGHT BE HARD TO BUILD!

The solution needs to restart into this configuration after failures, handle process crashes or reboots of individual components.

Data has to be stored and reloaded from files (or other μ-services)

We need to manage the service in a consistent manner and program it to self-repair after a crash or disruption.

WHY CAN DERECHO (AND OTHER TOOLS LIKE IT) HELP?

Derecho is Cornell's software library for automating those kinds of tasks. The design was created with "intelligent edge" use cases in mind.

The developer would attach event handlers in various places, and Derecho automates the remainder of the "life cycle"

This greatly simplifies the development challenge

AND WHAT ARE THOSE OTHER TOOLS?

Microsoft and Amazon tend to offer them in a form like a graphic novel.

1) A story – "Sally was facing such-and-such a challenge"
2) The visual story book – "These pictures illustrate the approach Sally used."
3) A template showing how to combine Azure IoT components and how to customize them to solve Sally's problem: "Here's what she did."
4) Visual Studio or VSCode can load that template and you can then mutate it into the solution to your personal puzzle.

… Today, Derecho is not integrated with cloud IDEs. But someday it will be.

HOW MIGHT WE TACKLE THE CASES MENTIONED EARLIER?

Consider one example:

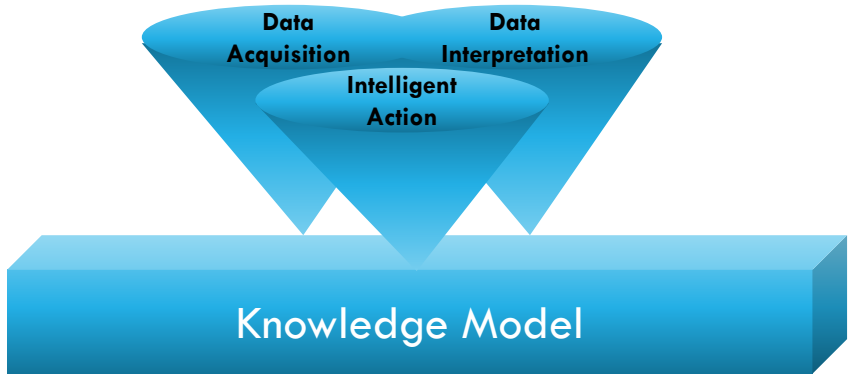
"Image analysis: "Are these plants healthy or diseased?"

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?

DIMENSIONS TO CONSIDER



In today's machine learning systems, the knowledge model is often slightly stale: it takes hours to compute and hence was created offline.

But in IoT, each situation is special: the topology of each farm is unique, today's weather is unique, the "job" we are doing today is specialized, etc.

Thus the standard approach will include real-time knowledge formation, through online data acquisition, learning and inference!

KNOWLEDGE MODEL BUILT YESTERDAY

This is an example of data ideally fitted to the CAP concept.

We can take a key-value cache and load the model "as we access it" into the cache. The model is basically unchanging while we are using it, so we don't need to worry about consistency issues.

Even if the model is immense, by sharding it over the cache, we have space. But we may need to compute in a parallel manner to avoid centralizing the machine-learning decision step in a way that bottlenecks.

NEW KNOWLEDGE GAINED AT THE FARM?

These are going to be knowledge models, too.

But they would be created dynamically and populated by rapid computational tasks running on the cloud, in the Azure Intelligent IoT Edge. (Reminder: this is the confusing name of the new first tier.)

Requirements: Again, a way to compute in a highly parallel way, but also now to replicate the new knowledge models created by these edge learning tasks.

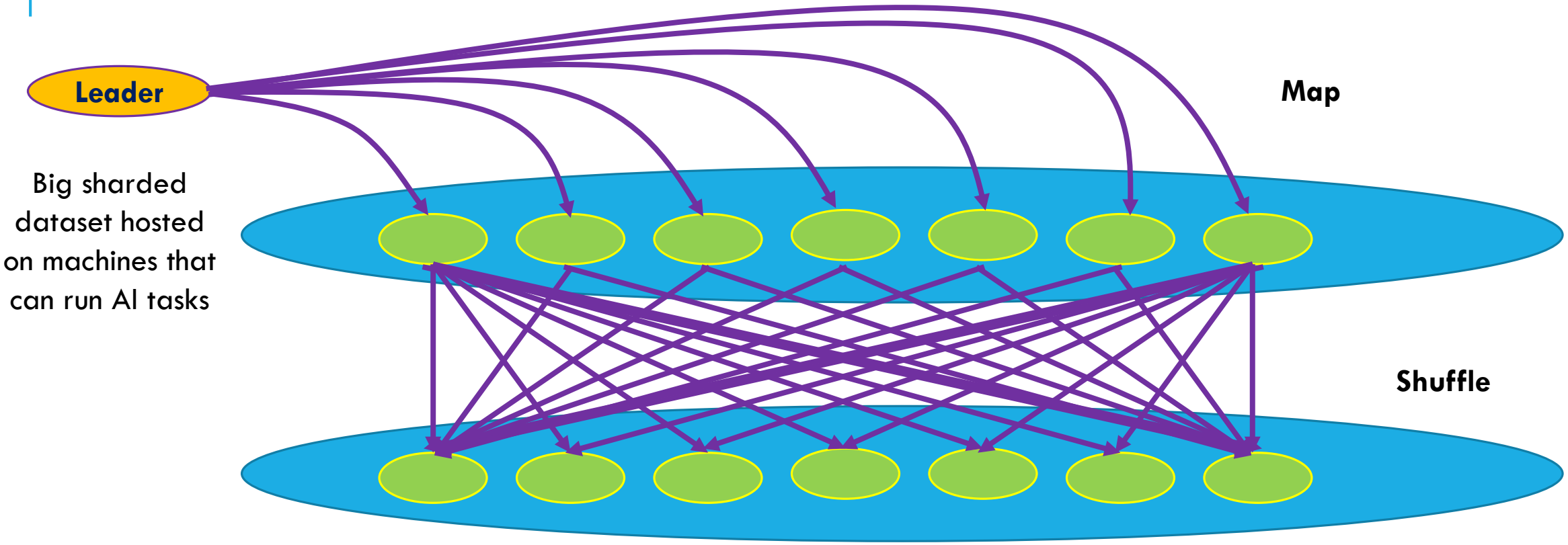
COMPUTATIONAL PATTERNS

We will spend more time on this in the last weeks of the class but clearly need to have at least some idea.

MapReduce pattern is the most common one.

- Some task is broken into shards and spread over N workers.
- They each compute for a little while on a part of the job. Their outputs are (key,value) sub-results for the bigger task.
- Then in the "reduce" step, we shuffle the sub-results to group data by key at a suitable shard, which can combine the set of values. This is the "reduce".

MAP-REDUCE PATTERN IN PICTURES.



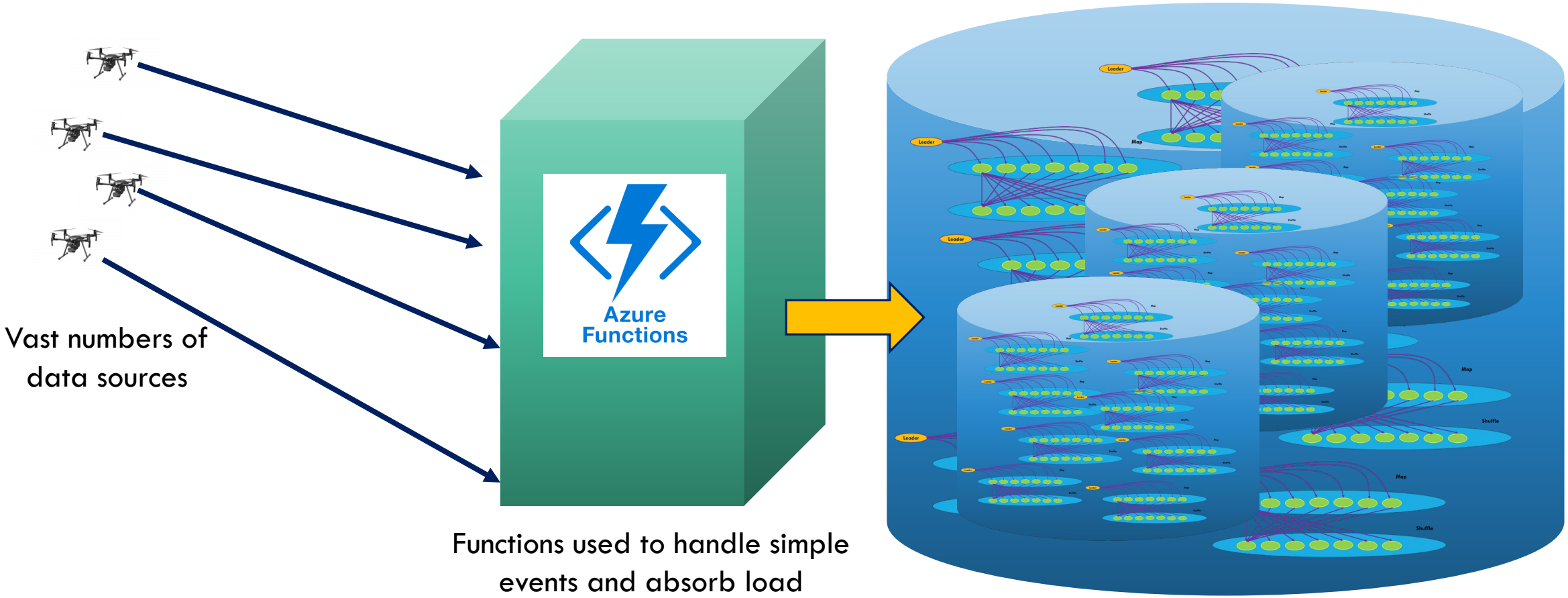
Full Shuffle is an n x n pattern: every shard sends data to every other shard! This avoids ever having all our work concentrated on any single process.

AT SCALE…

IoT systems will want to implement this pattern of computation (lots of instances of it, maybe millions running in parallel):

- At the edge, in the μ-services layer
- Information will need to be replicated on their behalf, at very high speed
- The reduced (key-value) data will be a sharded representation of new knowledge deduced by the computation!

WE END UP WITH A HUGE DATA-FLOW GRAPH



Vast numbers of data sources

Functions used to handle simple events and absorb load

Heavily sharded edge µ-services do real-time knowledge acquisition and decision making using ML computational models

BIG DATA?

Definitely!

These arrows might carry photos or videos: megabytes or even hundreds of megabytes per "object".

Just moving the data becomes a cost concern: in the cloud, copying isn't very fast. But recall that Derecho's object store uses RDMA for big-data movement operations. So Derecho is an example of a viable solution.

HOW MUCH CONSISTENCY?

For many tasks, modern machine learning is "stochastic" meaning that the learning algorithm converges in a non-deterministic way and could settle on any of a number of result states.

Consistent replication of IoT input is a common need, even for applications that use stochastic, noise-tolerant techniques. The reason is that "random noise" is very different from "stale or misleading input data".

Again, Derecho is a good fit to the requirement.

HOW MUCH FAULT TOLERANCE?

If we want FarmBeats to be reliable, we should plan on "riding out" some failures. By some estimates, one failure every few hours might be common.

Moreover, elasticity forces reconfigurations, like to add more servers or drop servers.

So the shards and computations need to be done in a fault-tolerant and elastic manner. Derecho has built-in help for this, too.

SO, BUILDER TOOLS COULD PLAY KEY ROLES!

Azure IoT and Amazon lack a tool like Derecho today. The IDE cartoon stories are very limited at this point.

But Derecho itself does run on both of these platforms, and we are working with the Azure IoT team to integrate it cleanly into their IDE environments, and maybe even to get permission to use RDMA too.

In CS5412 projects, we encourage you to work with it for any newμ-services you need to create.