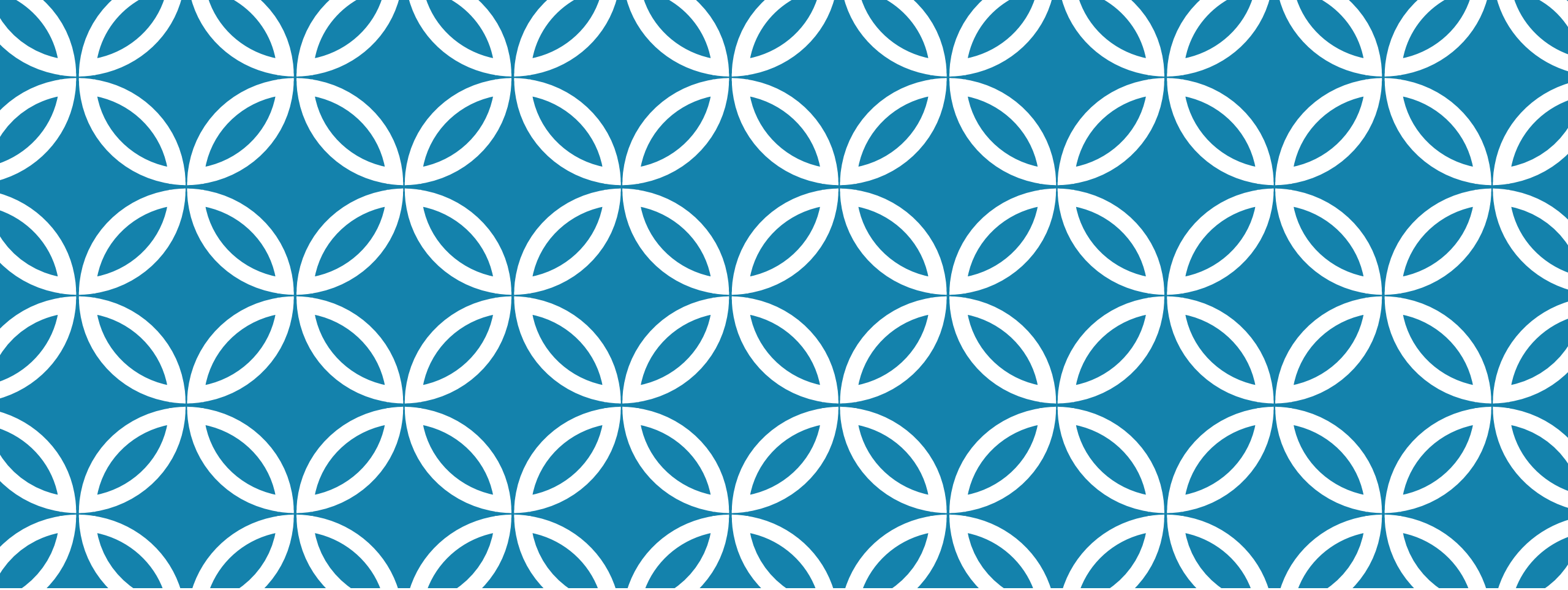


CS5412 / LECTURE 7
THE PUZZLE OF "ALWAYS SHARDED" IOT DATA AND COMPUTING

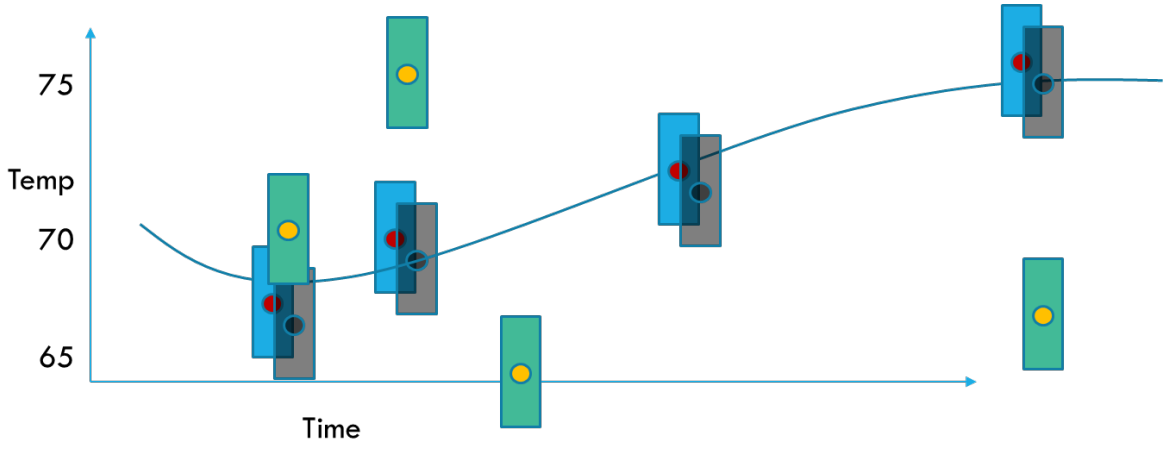
Ken Birman

Spring, 2020

TODAY: BRINGING TWO IDEAS TOGETHER

Suppose our data is sharded, and needs to stay sharded.

But suppose we also need to do something like the sensor intersection example from lecture 6, with a great many sensors, all sending data at the same time: a big data situation!



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"?

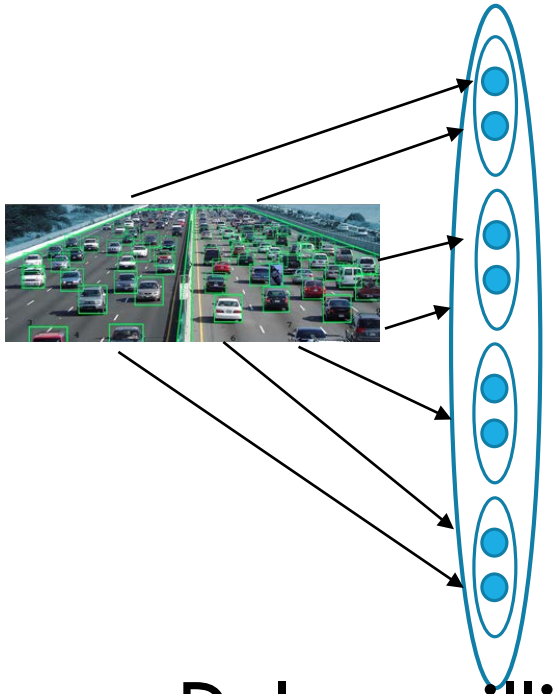
Until now, big data computations have run in big "back-end" systems like the famous MapReduce/Hadoop framework, or high-performance supercomputers.

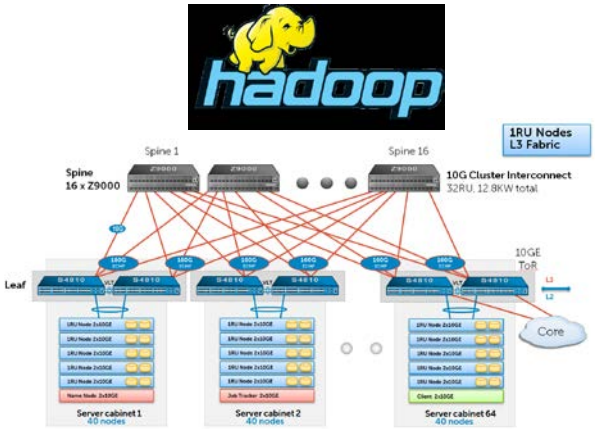
Big data processing was mostly done in batches, offline.

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





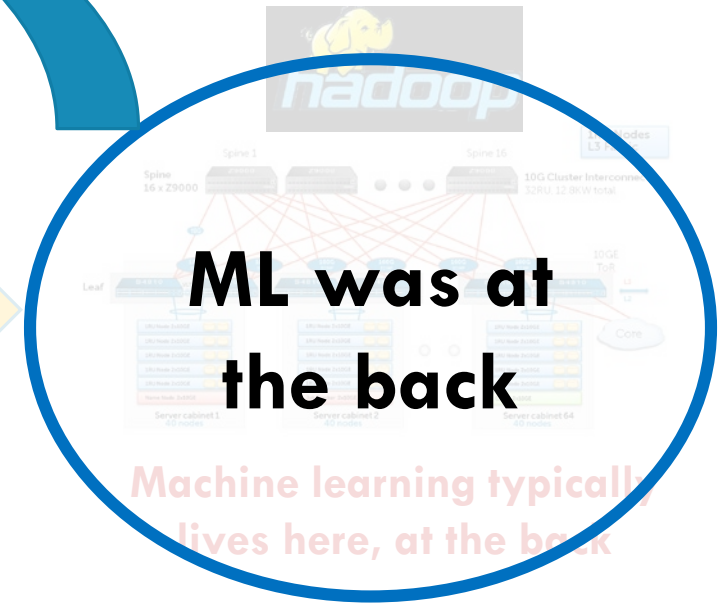
Machine learning typically lives here, at the back

Delay: milliseconds… Seconds…. Hours

NEW: MOVE ML TO THE EDGE OF THE CLOUD

Data acquisition… Global File System… Hadoop jobs

We move data classification and some aspects of learning here



Delay: milliseconds… Seconds…. Hours

APPROACH THIS LEADS US TO?

We will use Azure functions or AWS Lambdas 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?

Over time there will be a large number of successful IoT companies.

Those companies will connect to enormous numbers of IoT devices and actuators, with data pouring in at all times.

Much of this data will be big: videos, photos, radar/lidar. And even the smaller data may often require snappy responses.

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.

SO WHAT SCALING CHALLENGE IS THIS CREATING FOR US?

Huge numbers of functions – this can be handled with function services that launch containers as needed. The functions are stateless. So the model scales.

Huge numbers of µServices: We had a hybrid cloud and can repurpose its App Service. So seemingly we can scale out here too. More demand? More hardware…

The µServices are currently hard to build. Solutions like Derecho could help.

We need a scalable style of machine learning in the µServices layer. This is hard

ASIDE: WHAT IS "DERECHO"?

Derecho was mentioned in prior lectures: Cornell research solution for using atomic multicast / Paxos to update replicated data in shards.

Right now it isn't very integrated with the App Service, making it annoying to migrate a Derecho service into a cloud – not hard, just annoying.

We'll learn more about it in a week or two.

THE ML CHALLENGE

Today most machine learning occurs in big-data infrastructures that run in the cloud, but "offline"

- We accumulate a batch of work.
- We hold the actual data in massive sharded file systems or DHTs
- Then we run a special style of "always parallel" computing to train our ML models for big batches of updates, all processed at once.

How will we migrate this to the IoT Edge and IoT Cloud, to run in real-time?

"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?

MAPREDUCE

Invented at Google, but then spread into wide use when Yahoo! rebuilt it as the open-source Hadoop infrastructure.

It has a complete "ecosystem" with a sharded file system and a sharded computing model, supported by the MapReduce/Hadoop scheduler.

The developer learns to think in terms of batch-parallel computing, all the time, for every task.

HOW IS THIS DONE?

The developer thinks of everything in terms of *collections* of *tuples*.

- We try to view all forms of data as a kind of "row" of content in a table
- The row has fields: (name = value, name = value, ….)
- Often one field is designated as a primary key. Depending on the task, this key could be a file name, a GPS location, a hotel name…

Hadoop has many tools to help you transform your data into this form.

Modern programming languages embed collections into C++, C#, Python, Java, etc.

… MORE DETAILS

We work in steps

- We collect the raw data and "tag" in various ways
- There are simple tools to help, and in any case devices already send meta-data such as time, GPS, etc.
- A camera might add more: focal settings, who took the photo…
- Documents can be scanned to extract data from them
- Tabular data can be viewed as a collection of rows, and each row becomes a list of values

NEXT, WE TREAT ALL OF THESE AS TUPLES

The term just means a series of "column" values separated by commas

- (Name = Ken Birman, Title = Professor, Current_Course = CS5412…)
- (Name = Argos Lounge, Type = Bar, Address = ….)

Notice that a tuple could have varying numbers of fields. And one thing could have more than one associated tuple. The Argos is also a bed-and-breakfast.

Google suggests: Think of the whole world as a massive table, and each "thing" as a set of rows, and reach row with values in just some of the columns.

EXAMPLE, IN C#

```
var studentsGroupByStandard = from s in studentList
                              group s by s.StandardID into sg
                              orderby sg.Key
                                   select new { sg.Key, sg };


foreach (var group in studentsGroupByStandard)
{
    Console.WriteLine("StandardID {0}:", group.Key);

    group.sg.ToList().ForEach(st => Console.WriteLine(st.StudentName ));
}
```

THIS SEEMS TEDIOUS!

As noted, there are tools for automatically scanning many kinds of inputs and turning them into this form.

We can definitely design our IoT tools this way too.

Thus we have incoming objects, like photos, but also associated meta-data, which is in the form of a set of tuples, and each tuple has many fields.

WHICH IS THE KEY? WHICH IS THE VALUE?

In the way we access the tuples, we actually can specify the key we want to work with, and the value(s) we are interested in.

Effectively, the world looks like a huge collection of SQL databases.

The programming style looks like SQL, but runs in the non-transactional NoSQL model. Each "action" is atomic, but sequences of actions are *not* glued into atomic transactions. No begin/commit/abort.

VISUALIZE THE RESULTING COMPUTATION?

At a thousand places, we have some kind of stored data, or new data arriving from an IoT device.

At each of those places we apply some small bit of code.

The output of this code is a set of tuples (name = value, name = value, …)

Then we can use these tuples as *input* and form a kind of graph.

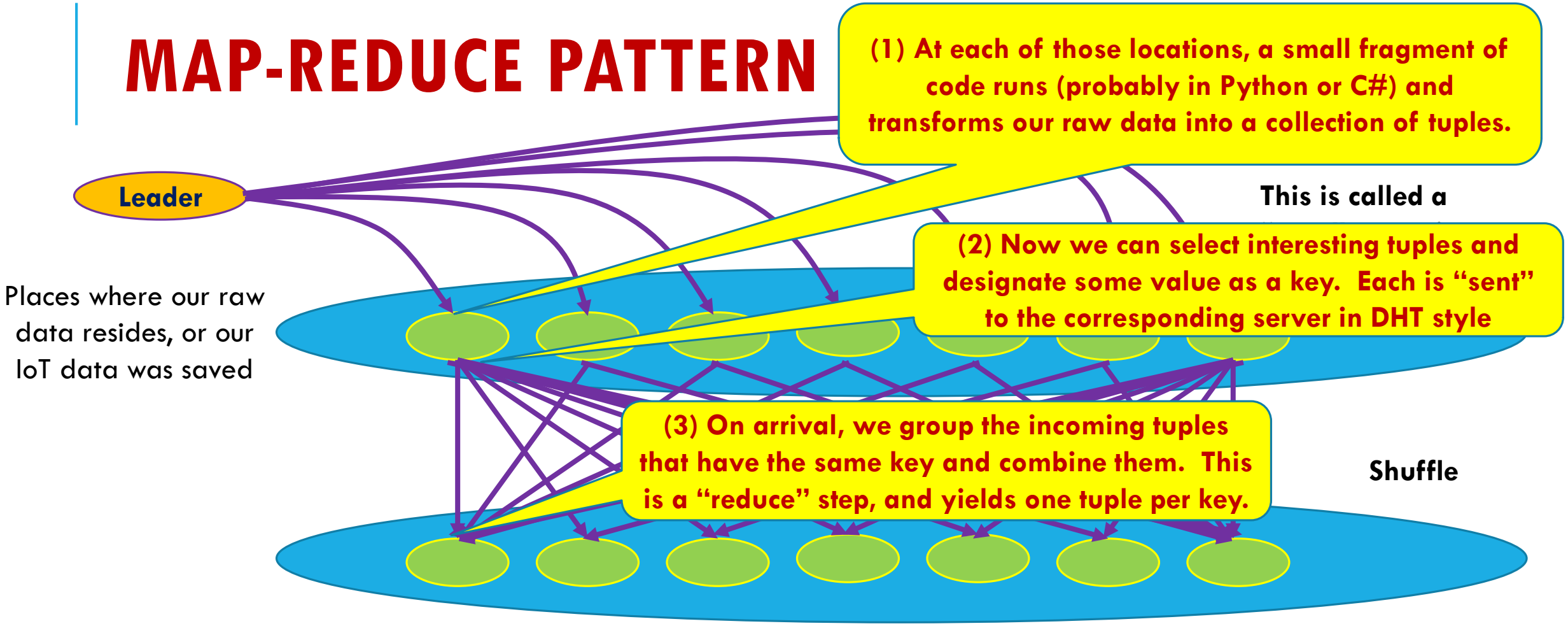
DOES THIS CONNECT TO DHTS?

Yes! When we compute on tuples, we often specify a key. Asharding pattern is then formed for us, at runtime. So if our tuples have keys, we can store them into DHTs or use the keys to connect "related" things

For example, we take a lot of unstructured data, extract tuples, select the ones that we want to compute on, and store them into a DHT.

This happens in a massively parallel computation on hundreds of machines

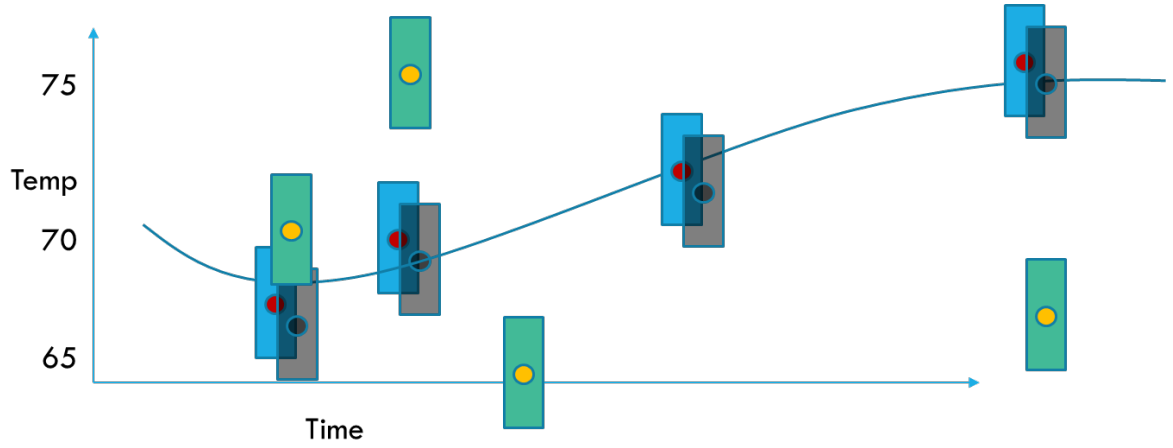
MAP-REDUCE PATTERN



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.

EXAMPLE FROM LECTURE 6

We had sensors in groups of 3 (this is showing one group over time).



Then we wanted to combine the output by looking for a "box overlap" to compensate for sensor errors.

In a MapReduce pattern, how might this work?

FOR ONE SENSOR GROUP

We can have tuples that include (group-id, sensor-number, "box")

- The sensor is assigned to a group, so the function layer can look this up
- The sensor-number comes to the function layer from the IoT Hub
- The "box" is the reading – this sensor saw 69.5 degrees at 10:59am
  - In fact the sensor output for this example is a single point.
  - But by looking up the clock precision and the sensor accuracy, the function can turn that one point into a box ([10:58.850, 10:59.150], [69.4, 69.7])
- Output of this operation: our "tuple"

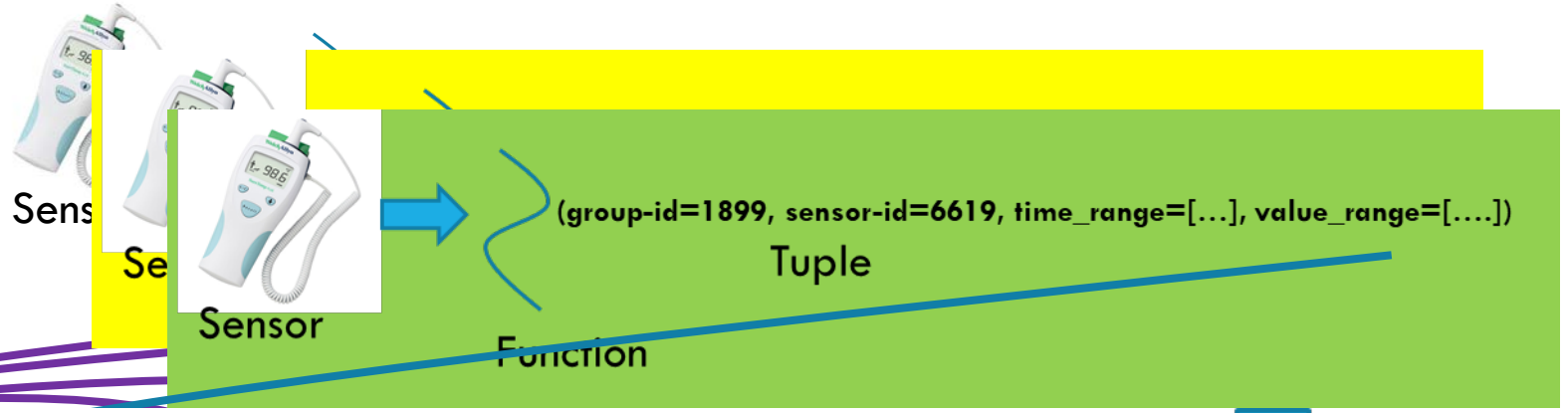
THINK OF THE FUNCTION AS A FIRST MAP STEP

We actually have 3 sensors, each caused an event, and they trigger 3 separate functions to run.

Each function creates one such tuple. It stores its tuple in the DHT.

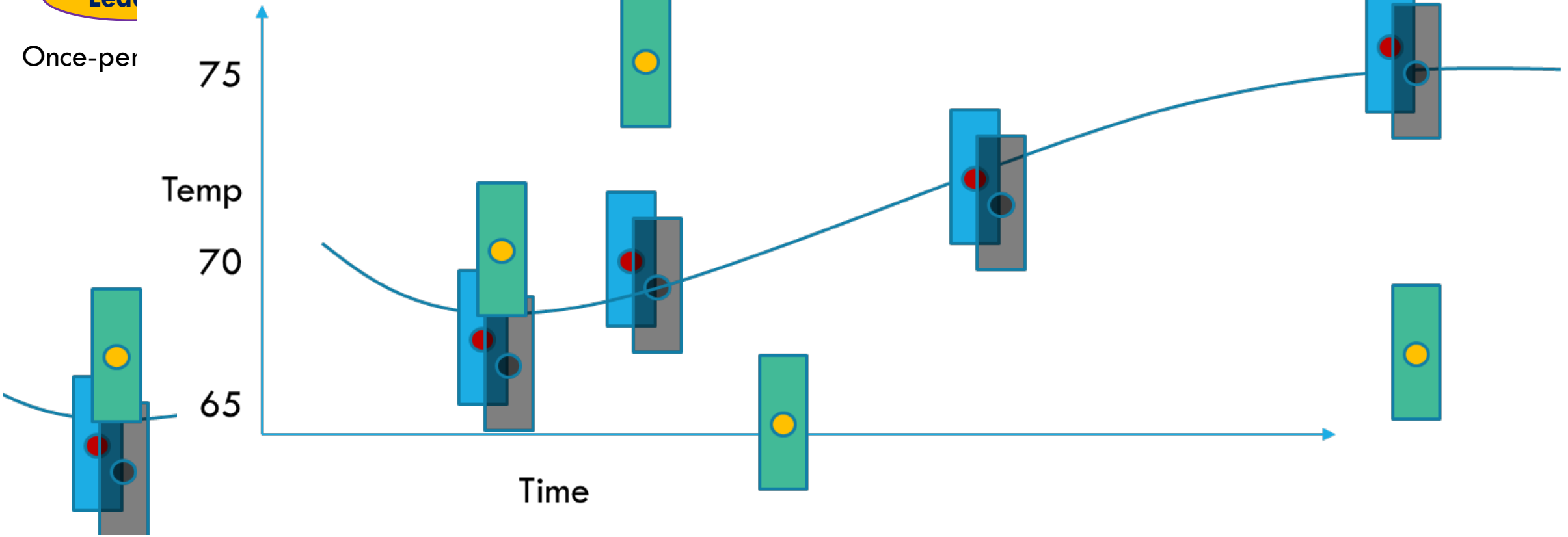
After, say, one second, we can run the shuffle and reduce operation.

IN PICTURES.

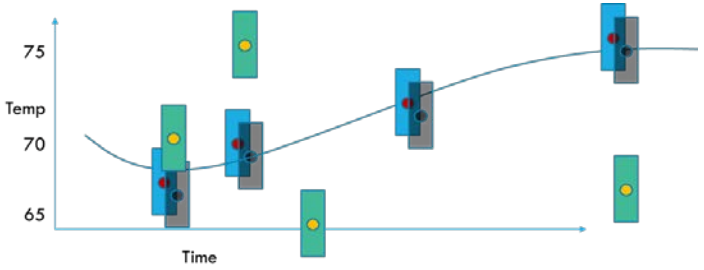


Lea

Once-per



THE COMPUTATION HAS STAGES



Once we have our new overlap region, it lets us extend the curve we are building of the temperature for this one sensor

- Stage one: the sensors submit their values, triggering the functions
- Stage two: we've accumulated our data in the DHT (temporarily)
- Stage three: the periodic trigger causes us to do this shuffle/reduce
- Stage four: we extend the prior curve with one more data point

NOW IMAGINE THAT WE HAVE MANY SENSORS

We would be collecting data from all of them "at once"

- So there will be many sensor groups, all doing this simultaneously
- The shuffle won't shuffle data for one group… it will run for many
- The reduce won't occur on one node, for one group… it runs on many nodes, for many different groups, and extends many curves.

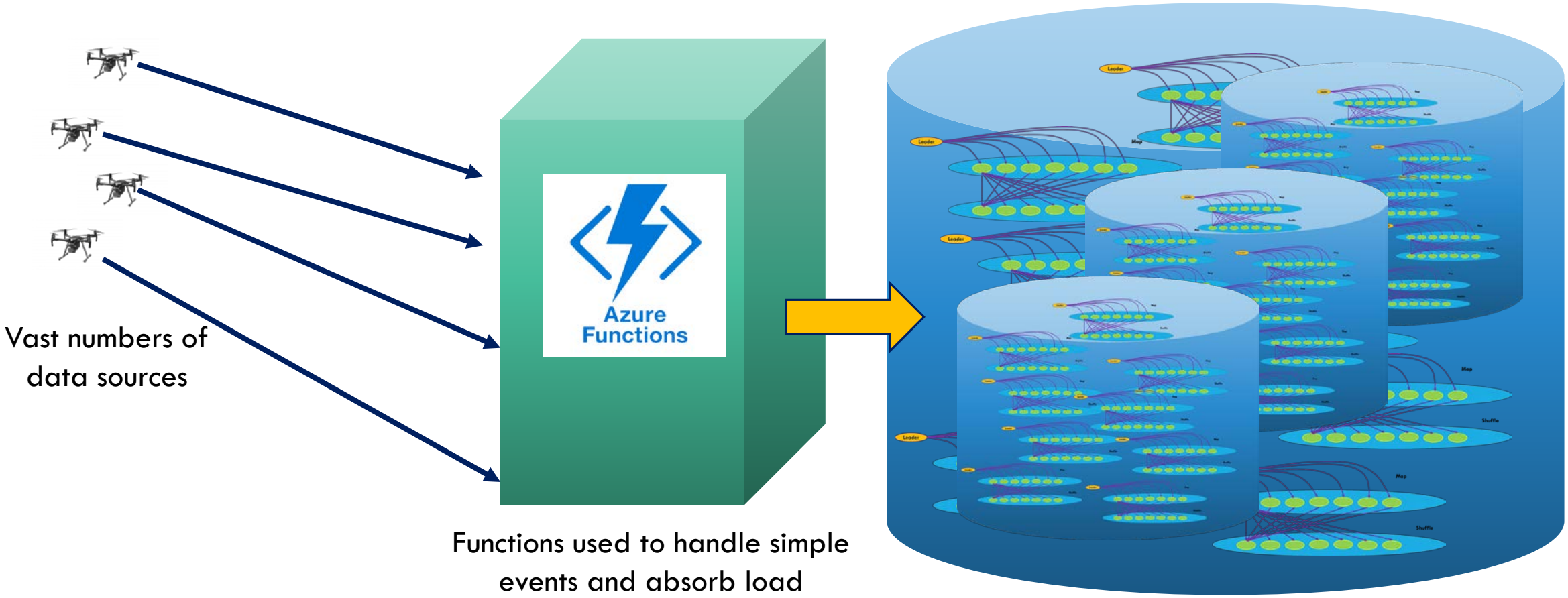
In fact we could do this for more than one customer, each customer having many IoT sensors. And that would be a serious "big data" pattern!

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 GRAPH PER USE CASE! THERE MIGHT BE THOUSANDS AT THE SAME TIME



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.

SOME NEW ISSUES SEEN ONLY IN THE EDGE

The MapReduce pattern depends on having a lot of data ready for parallel computational analysis.

But with IoT, especially under real-time pressure, the new data arrives when events occur, "one by one", and we have to compute immediately.

MapReduce was very efficient because of the batching. Can we find ways to make IoT reasonably efficient too? If not it won't be cost-effective.

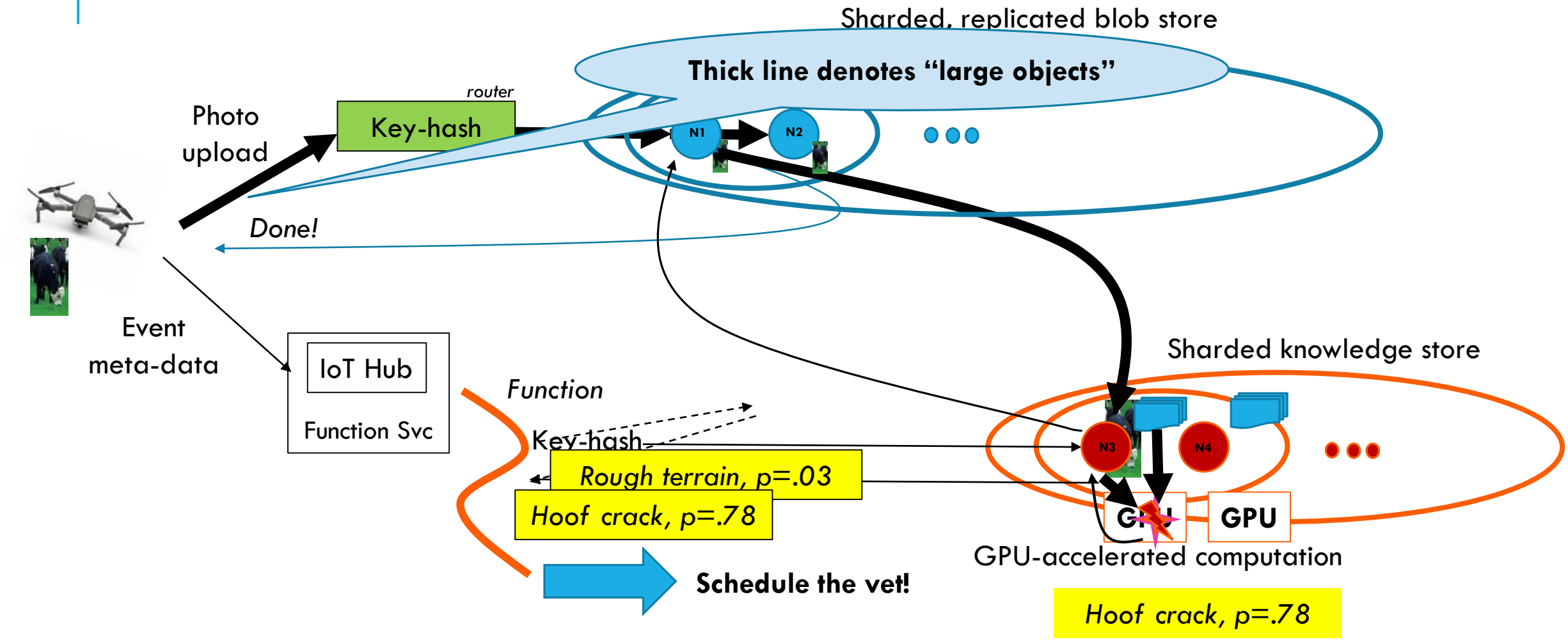
HARDWARE ACCELERATORS

One way that big data frameworks gain performance and efficiency at scale is to use programmable hardware to speed up key steps.

For example, there a lot of use of GPU computing (or at Google, TPUs).

If we want our edge systems to do things that currently are common in the big data back end frameworks, how will we get a cost-effective hardware accelerator capability near the edge, where the data arrives?

REMINDER: FROM LECTURE 4



SUMMARY

To support intelligence near the IoT device, we need to make the IoT Edge or the Cloud first tiers intelligent.

This requires building new "smart" μServices, which will be

- Sharded, for scale, and perhaps leveraging hardware accelerators like GPU
- And may need to support MapReduce styles of computing. In fact we would ideally want to use existing MapReduce code near the edge.

The devices are the easy part. The really big challenge is to figure out how this style of computing can be done at scale, in real-time, cost-effectively!