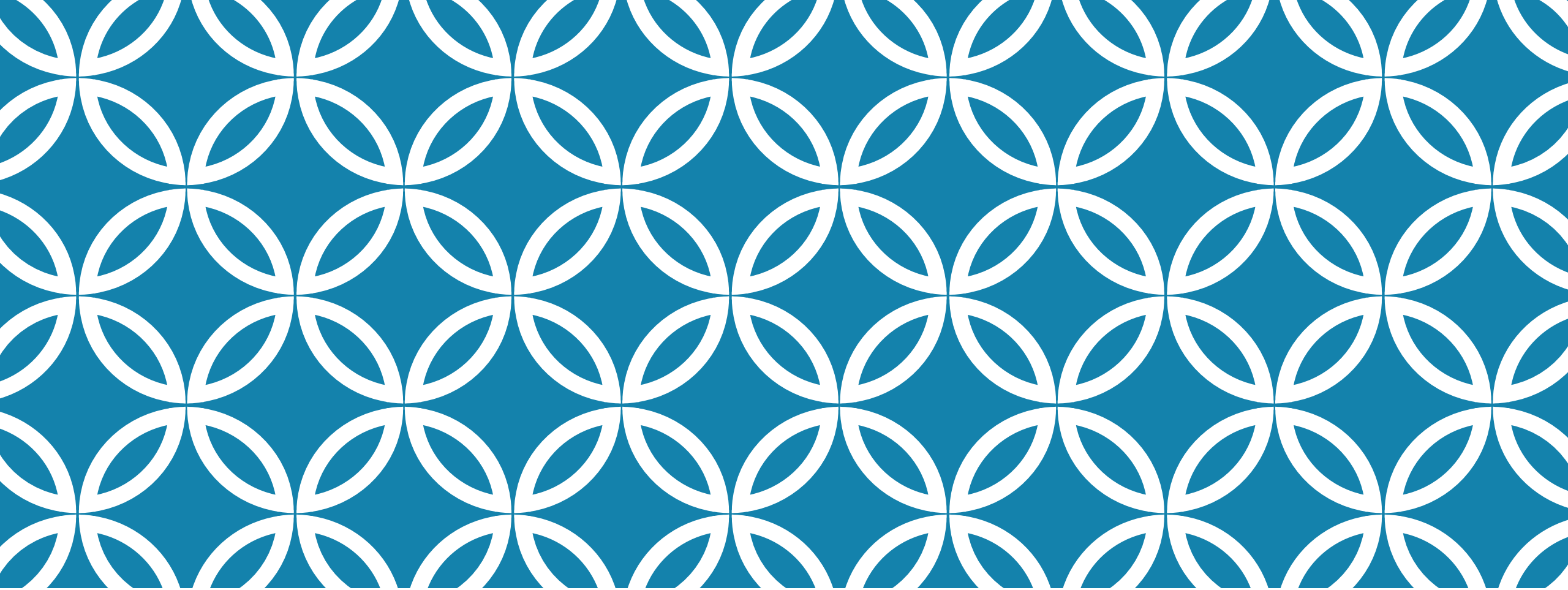


CS5412: LECTURE 4 IMPLEMENTING A SMART FARM

Ken Birman

Spring, 2018

TODAY'S LECTURE: HALF RECAP, HALF NEW

In most lectures this semester, we just push forward.

But the topics from lectures 1-3 are central to your projects and to the course.

So today's lecture will overlap a bit with lecture 3, although it has some new content too. This is to reinforce key ideas by seeing them again in a slightly different way. [Anyhow, we ran out of time on lecture 3!]

WHAT IS IOT BEST AT, TODAY?

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

This could include swipe cards, 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}

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.

SOLVING THIS DOOR EXAMPLE

We listed a series of simple events. The state involved is simple too, such as "door is locked/unlocked". *So use functions as the "main" element.*

These functions will consult with μ-services. All are standard ones:

- A database of access permissions.
- A database of employee images with an image recognition capability.
- Key-value storage holding other forms of previously captured information.
- Audit log (BlockChain) that tracks who was allowed to enter the building.

HOW EASILY DOES THIS EXTEND TO FARMING?

With a smart farm, the core concept is similar.

But suddenly we are controlling robotic devices (like drones, cameras that can swivel and zoom), tracking much more "dynamic" information, taking actions based on recent knowledge.

This is a common theme for many kinds of smart IoT use cases, like smart highways, smart homes.

MICROSOFT FARMBEATS

Quick reminder of some smart farming ideas:

- Drones that would survey fields and help with intelligent decisions about seed 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

INTELLIGENCE ROLES

The intelligence takes many forms here, even just in the drones

- How can program drones to employ optimal search patterns?
- How should the drone's battery power be managed? Can we leverage wind to "sail" and reduce consumption?
- Which crops to photograph, and from what angle, and how close?
- Are there signs of a problem? Should we get more data?
- Which photos to upload, and which to save, and which can be deleted?

HOW TO MAKE AN INTELLIGENT DECISION?

In our first example (Ken Birman wishes to enter Gates Hall late at night) the intelligence is very limited.

- Basically, just a question of capturing a photo and recognizing the person in it.
- In fact this isn't trivial, but with proper focus and good lighting a computer system can definitely be trained to do it.

The only "dynamic" aspect would be if, say, I was just authorized today and the database needed to be updated.

HOW TO MAKE AN INTELLIGENT DECISION?

In a more complex setting, we need to start by creating a model!

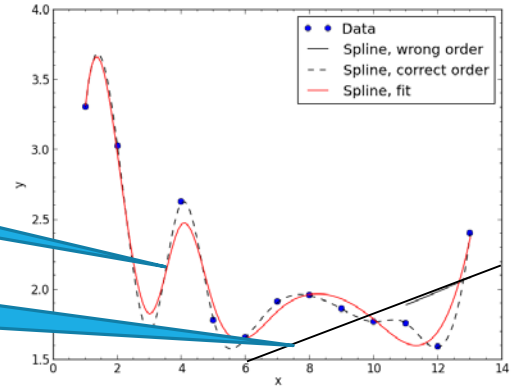
So, offline, a research team develops a "theory" that matches the intended behavior very closely. Like a set of equations describing smart drone overflights of a field, and choices the drone makes as it flies.

Next, we collect a huge amount of sample data and label it with the desired choices we would want the system to make under those conditions.

TRAINING A MODEL

A spline uses a model too (a polynomial of some order)

With the wrong model you still get something, but it may not be useful



With our data and labels and model, we can now select a machine-learning technology matched to the setting.

For example, perhaps we settle on a convolutional neural network, or a parameterized Bayesian belief graph, or some other standard option.

With our data, we can train the model: we compute a set of parameters that will lead the model to "generate" the desired behavior.

THERE ARE A LOT OF TECHNICALITIES!

For example, we often need to augment our data with synthetic data to ensure that the model sees a sufficiently broad set of examples.

If the model itself wasn't rich enough, it won't converge to an acceptable solution, and we might need to refine the model.

When finished, our model might not be able to make ideal choices in every case, but hopefully it does well enough to justify "field trials".

WHAT DOES A TRAINED MODEL LOOK LIKE?

The model we would want to run on our drones consists of:

- Code that takes input data, identical in style to our training data
- The equations, implemented in a language like Tensor Flow or SciPy
- The parameters for those equations, obtained from the training process. This will be a set of tensors: big files containing numerical data, perhaps gigabytes in size (even this assumes some form of compression)

BUT SUCH A MODEL WILL BE "GENERAL"

We can definitely train drones to overly "any field", given its boundaries and a topographic map.

But any specific field will have all sorts of unique characteristics.

So FarmBeats will need to adapt its plans dynamically and might even need to learn dynamically (to learn things about this specific field) in order to do the best job! *And we won't have time to do this offline!*

WHERE DOES TRAINING OCCUR?

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

In fact we will discuss this topic later in the course, in the last few lectures.

But the dynamic form of learning needs to occur in real-time, closer to the edge. Since the drone itself lacks compute resources, this would be on a cluster of computers "near" the farm. Maybe, in that nice blue truck.

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 search 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.

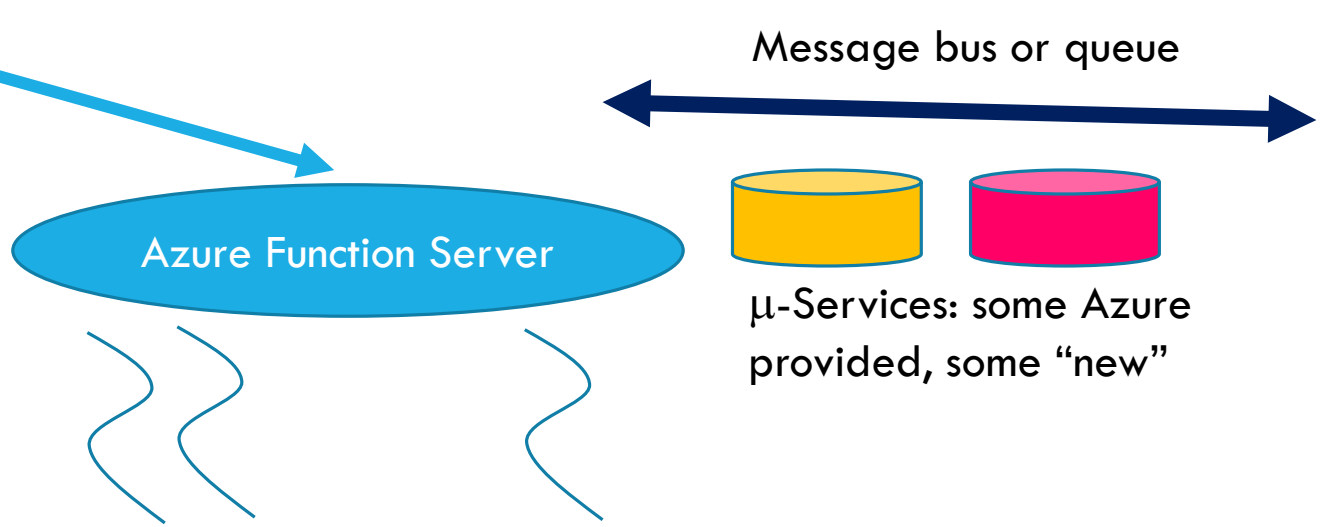
WILL THIS SCALE?

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

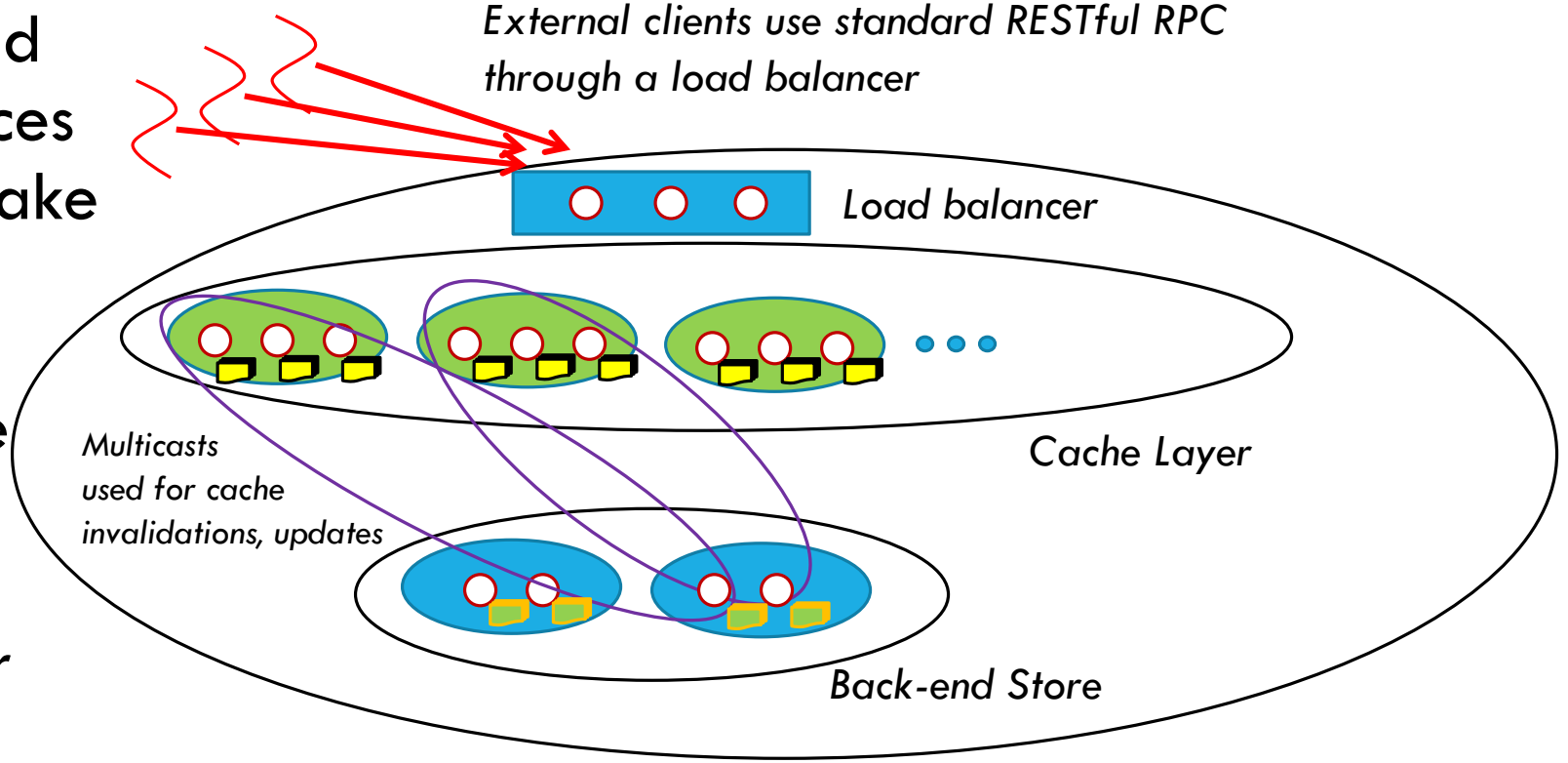


Functions: Lightweight, event-triggered programs in containers, "pay for what you use" resource model

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



A μ-SERVICE 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.

THIS IS A SITUATION WHERE DERECHO CAN HELP

Derecho is Cornell's software library for assisting in μ-service development. 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

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.

SOME OBSERVATIONS

IoT is creating very complex challenges, that go beyond what machine learning has dealt with up to now.

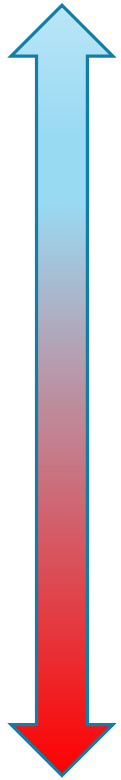
Generally, machine learning systems are applied to a fairly narrow question, such as “who are the people in this photo?” or “given Google’s knowledge of Ken, how should search results for such-and-such a query be ranked?”

IoT is generating multiple questions, superimposed, and perhaps even in tension.

A SPECTRUM OF CHALLENGES

IoT really covers a *lot* of territory

- Simple devices with simple roles, like card swipes and door locks.
- Computer vision, but used in easy ways, like to read car license places or on a factory assembly line.
- Computer vision or speech with more challenging inputs and roles…



MORE OBSERVATIONS

As such, these problems won't be simple to solve!

Any single use case could involve writing large amounts of code – perhaps in the form of new μ-services, perhaps new functions, perhaps even new hardware to create suitable sensors.

But the emergence of new industries is never all that simple!

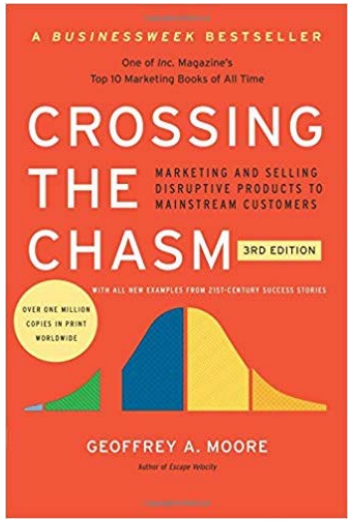
WHY BET ON IoT, AT ALL?

Our course does reflect a form of bet: that IoT will grow up and do much more than unlock the Gates Hall front door.

But if this will be so hard to create, why believe that it is going to happen?

Many great ideas go nowhere…

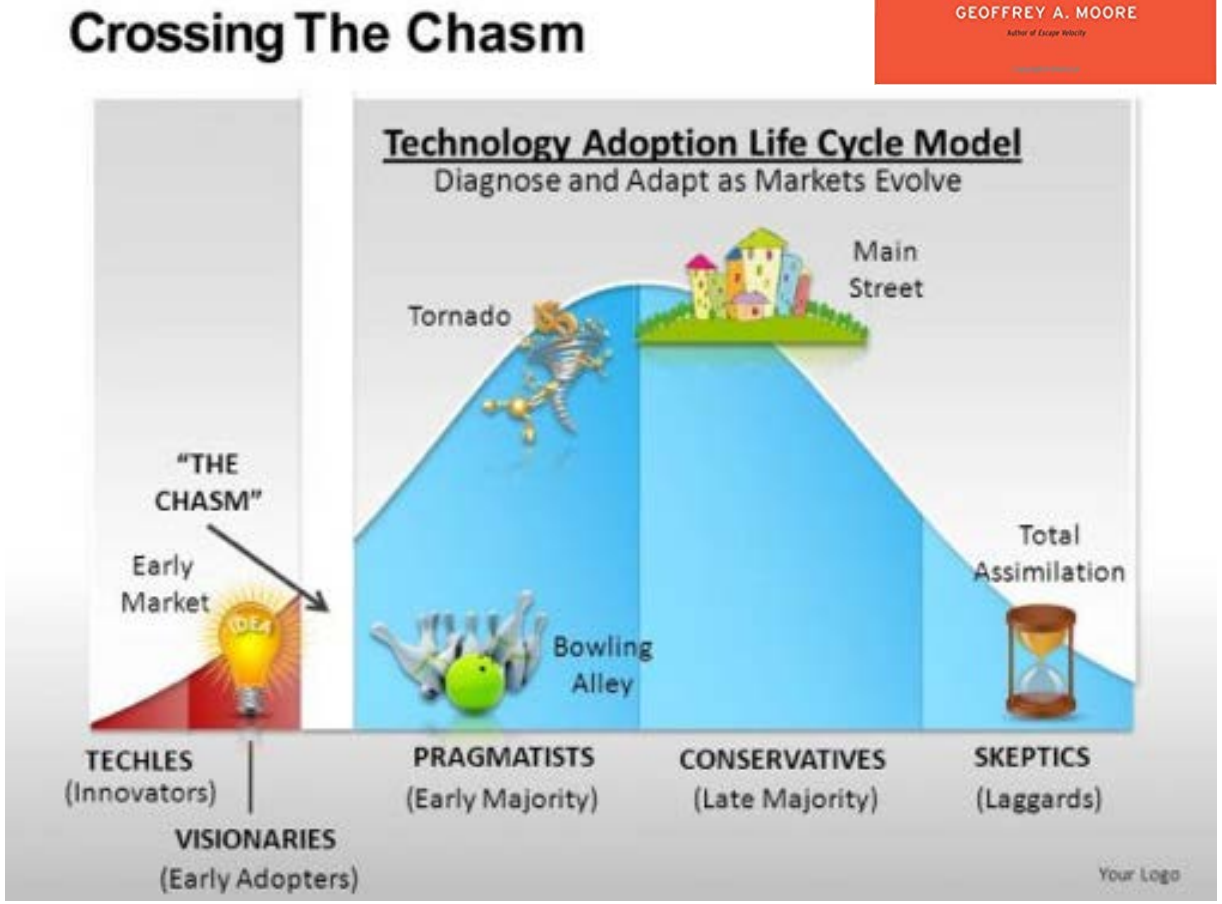
CROSSING THE CHASM



Key ideas:

- Success comes slowly
- Even major new markets often are slow to mature.
- Conservative adopters are a reality, and often even have a point!

IoT is in that early market stage



SO… WILL IT STUMBLE?

Many people believe that we are in the chasm right now… and even this view may be "optimistic"!

But the key thing is the money… money will eventually bring success, and there is already an immense amount of money on the table.

With enough money, problems that "only" involve a lot of coding will gradually yield. And step by step, IoT will become a bigger deal.

WHY DID SO MANY CORNELL CLOUD PEOPLE BECOME SUCH STARS?

The people you've heard about were people who believed that when something really big is happening, you can succeed by

- Being really smart and creative.
- Using those skills in that domain.

Success stories can be home runs. But even a failure helps the field mature. And the experience makes you more respected within your company as a person who speaks from hard-earned insights!

CONCLUSIONS?

We can already see a path to using today's cloud to solve some of these problems. Start with those.

In your CS5412 project, we want you to aim for something challenging but feasible in a limited amount of time!

Over time, tools will emerge to simplify other tasks. These have not yet matured. Yet it would be wrong to bet against IoT just because it is immature. The IoT "direction" is gaining momentum (and value)