IoT & Microservices...

A PERFECT MATCH

@perrynfowler

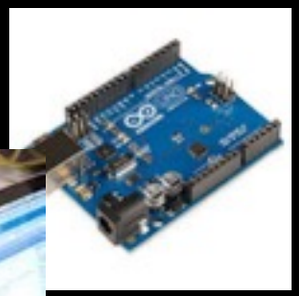
Perryn Fowler is Head of Analytics at Urbanise

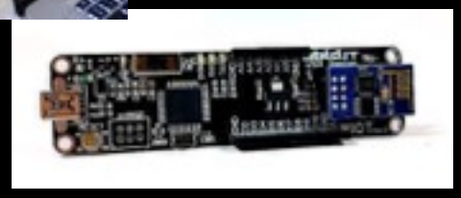
DEMO


urbanise
Wireless sensors automatically connect to gateways near them
Gateways send data via mobile telephone networks to the Urbanise cloud platform
Remotely connecting the workforce, help desk & clients through our cloud Platform

SENSORS, FIRMWARE, ARDUINOS, HARDWARE...

IoT





What is IoT - a lot of the time talks on IoT are about Gadgets:

- sensors, arduinos, building them, programming them
- basically about the T…

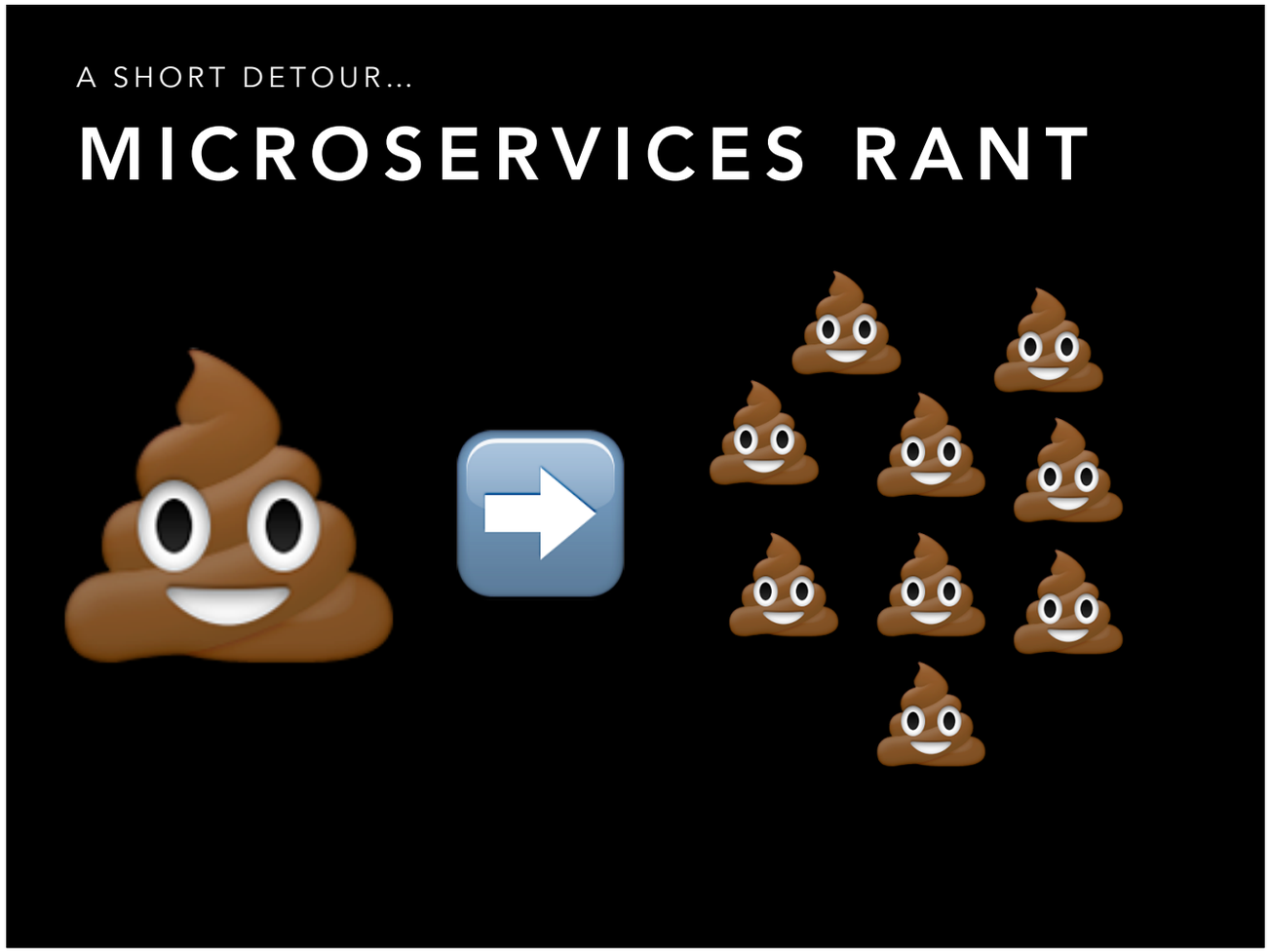
But I’m going to talk about the I. Thats about data. real time data.
All these gadgets are going to be sending us data of the internet in real time.
And we need to collect it, store it, enrich it from other data sources, analyse it, present it and update all of it in real time.

REAL TIME DATA.

IoT

Oh and you are going to have millions an millions of devices out there, so it needs to be scalable.
And this is the internet, so it all also needs to be fault tolerant…

ELEPHANT IN THE ROOM is security



- So how do you a design an architecture for IoT that is all those things?
- Well at this point I’m going to take a small detour into Microservices…
- And first of allow should talk briefly about what micro services actually are:

MICROSERVICES... WHAT EVEN ARE THEY?

Auto-antonymy

Literally.

So there is this strange phenomenon in language called Auto-antonymy where over a long periods some words come to mean the exact opposite of their original meaning. Usually this takes a long time.

For some reason in software development it seems to take just a few short years.

MICROSERVICES... WHAT EVEN ARE THEY?

Auto-antonymy

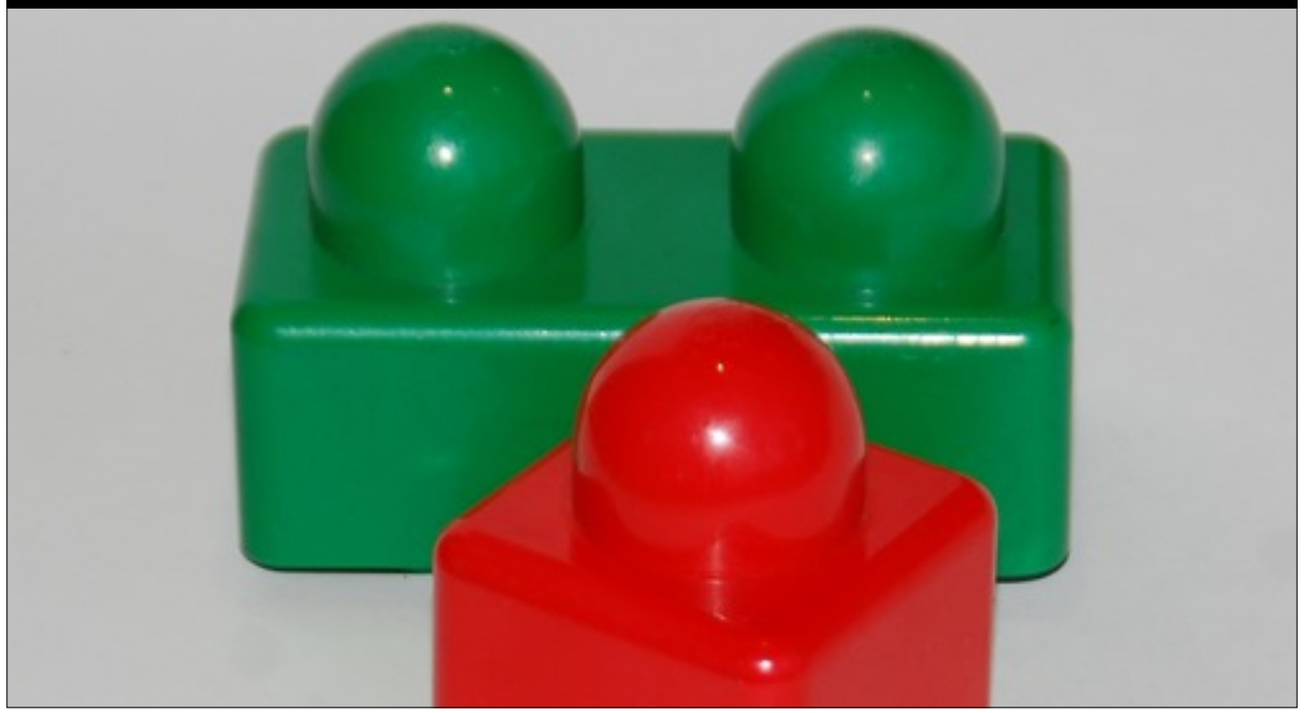
So you find yourself trying to talk about a concept that has somehow come to mean “put a lot of spaghetti logic inside some kind of propriety magic Enterprise Service Bus… whats a “thought leader” to do?

Need to come up with a new word… an the whole process starts again.

So Microservice Architecture is basically Service Oriented Architecture with a new catchy name - and perhaps with slightly more emphasis on small services.
But that doesn’t help much since SOA has lost its meaning

MICROSERVICES ARE:

SMALL, WELL DEFINED & INDEPENDENT



So when I say ‘Microservices’ I mean services that are both RELATIVELY small and independent.

- NOT TOO SMALL - I’ve heard some horror stories about people making 500 micro services to build a website. Don’t go crazy. Maybe more DUPLO than LEGO.

THIS MEANS MICROSERVICES CAN BE:

INDEPENDENTLY...

Developed

- hold what they do and how they work in your head
- choose the best technology for that specific job, instead of ‘one size fits all’.
- if worst comes to worst you can throw them away and rewrite them

So the theory goes that because micro services are small and well defined. They can be developed independently. They are small & independent enough that you can hold what they do and how they work in your head - you don’t need to worry so much that you have broken some other piece of it
They are small & independent enough that you can choose the best technology for that specific job, instead of ‘one size fits all’.
Also, they are small & independent enough that if worst comes to worst you can throw them away and rewrite them. This is great if you want to experiment with new tech without fully committing to it.

THIS MEANS MICROSERVICES CAN BE:

INDEPENDENTLY...

Developed

Deployed

-The theory also goes that if you have these services that are small and well defined, you can deploy them independently of each other.
-You don’t have this huge ball of mud that you need to test end-to-end before you can change it. - - And if it breaks you need to roll-back the whole shebang
- You can deploy this small isolated change, and if it breaks, just roll that back.

THIS MEANS MICROSERVICES CAN BE:

INDEPENDENTLY...

Developed

Deployed

Scaled

-The theory also goes that when we are terribly successful and a part of our system need to handle more load

- We can scale just that part, instead of the whole shebang
- And if we know which parts will need to be especially scalable, we can focus effort on making just those parts scalable.

THIS MEANS MICROSERVICES ARE:

INDEPENDENTLY...

Developed

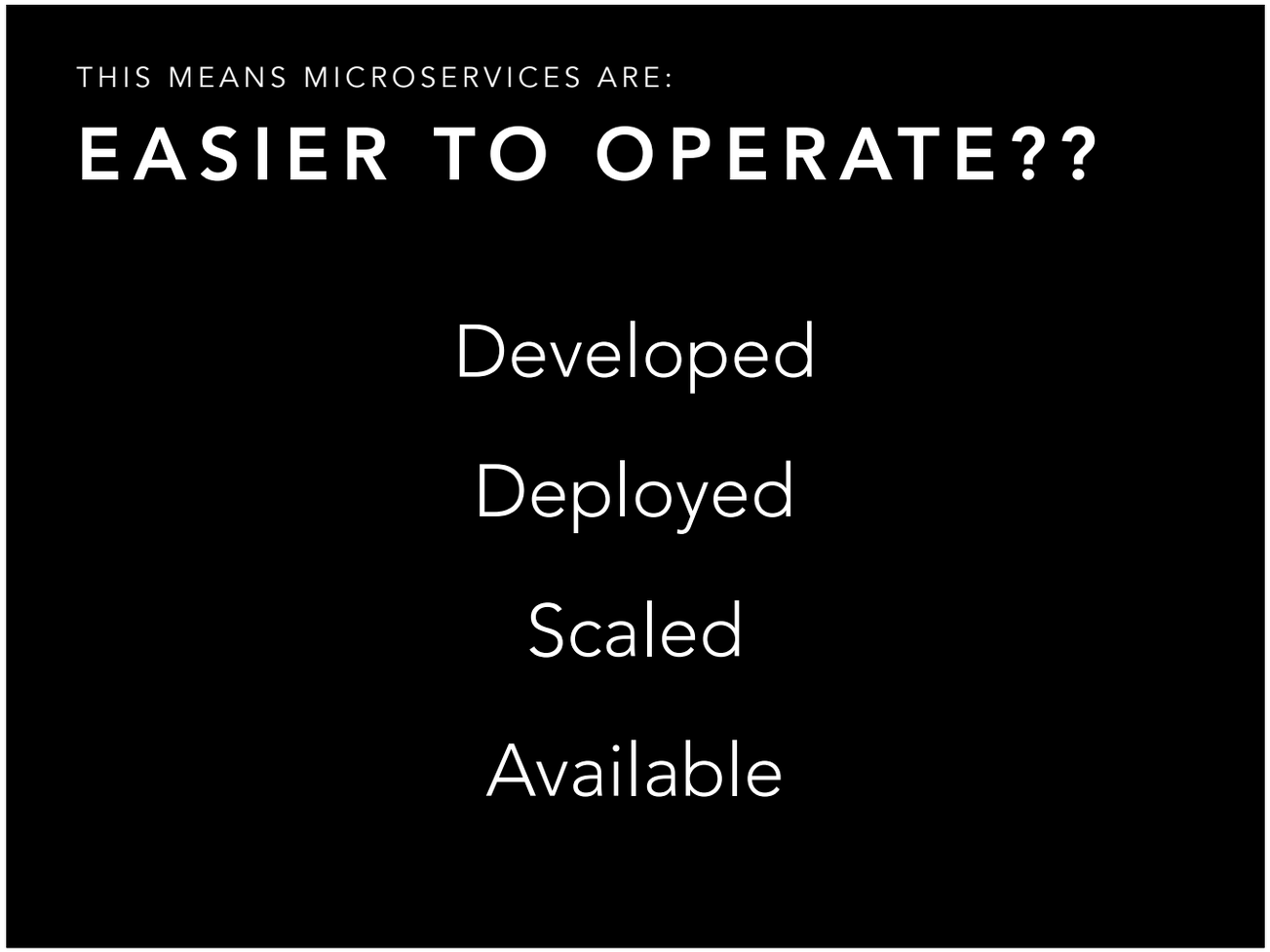
Deployed

Scaled

Available

Lastly, if our services are nicely independent then if one of them goes down it doesn't bring the whole system to a screaming halt.

There are probably a lot of other attributes that define micro services, but these will do for now…



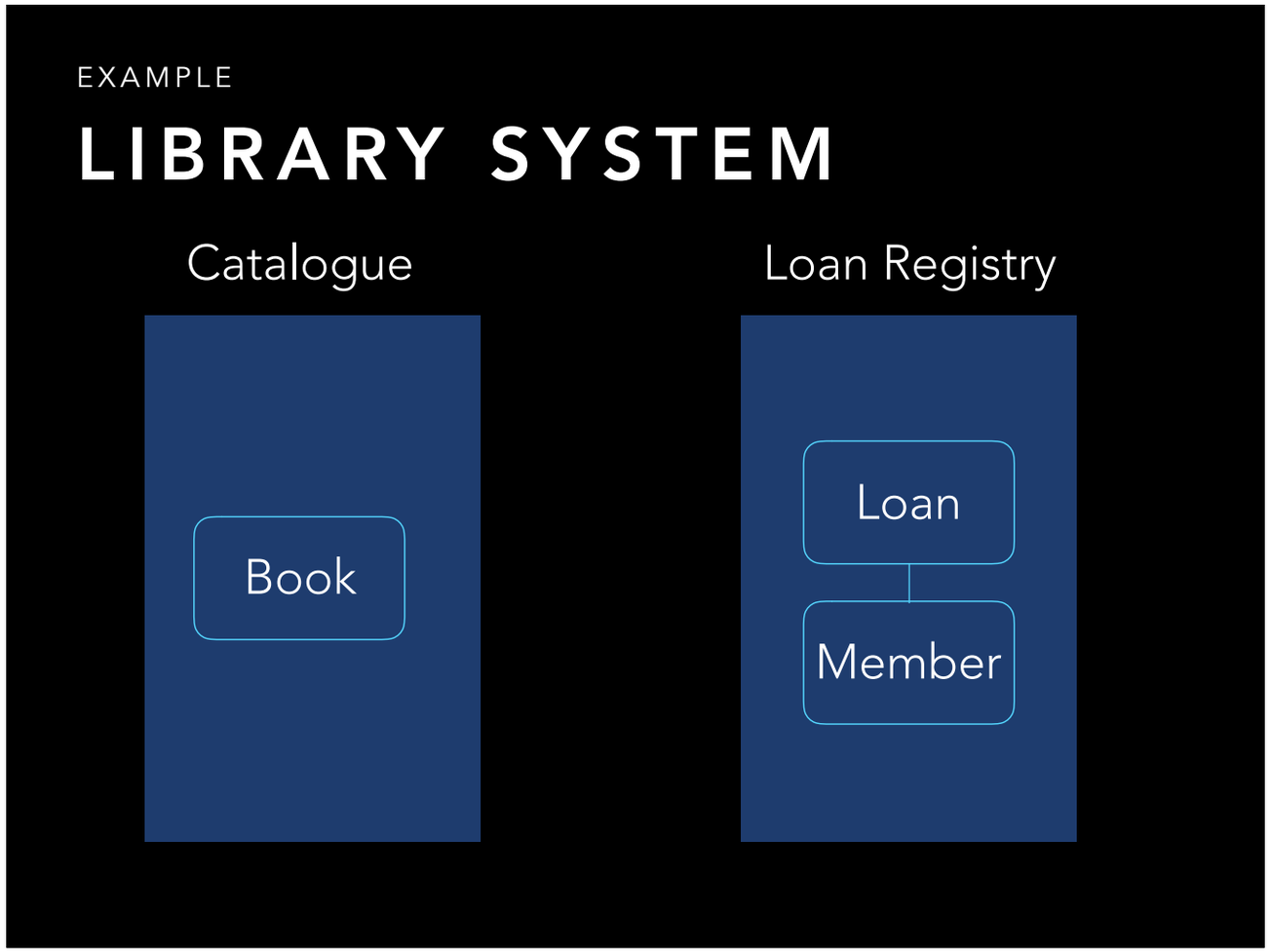
- So an interesting observation about these is that a lot of this is about making the system easier to OPERATE. Which is interesting because there seems to be a strong perception that micro services are harder to operate.
- In fact, I'd say that for a lot of so called "microservice" architectures out there, do actually make some or all of these things harder. Harder to develop. Harder to Deploy. Harder to scale etc etc

EXAMPLE

LIBRARY SYSTEM

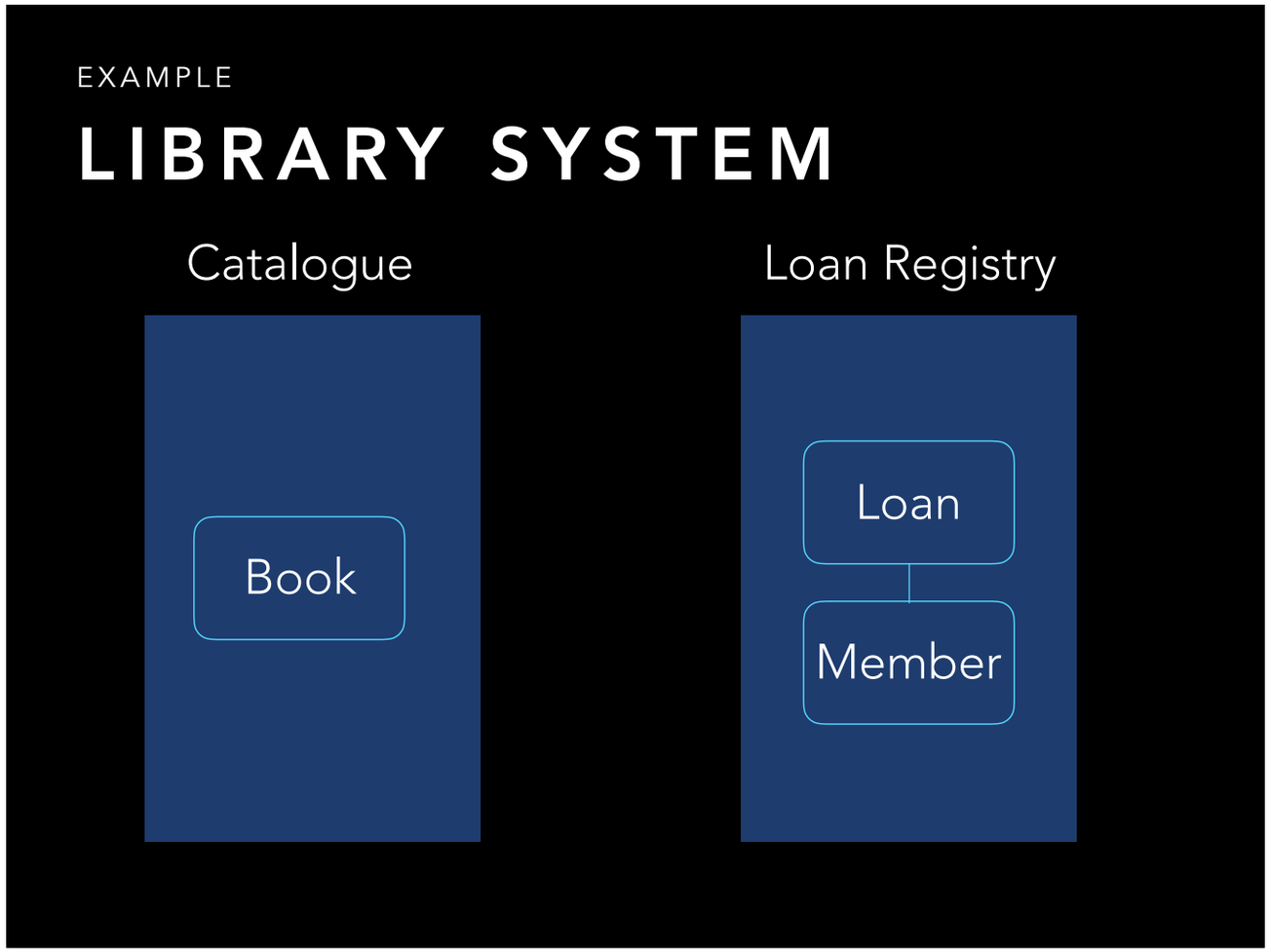
- So how does this happen?
- well lets walk through an example

Guess where I was when I did these slides?

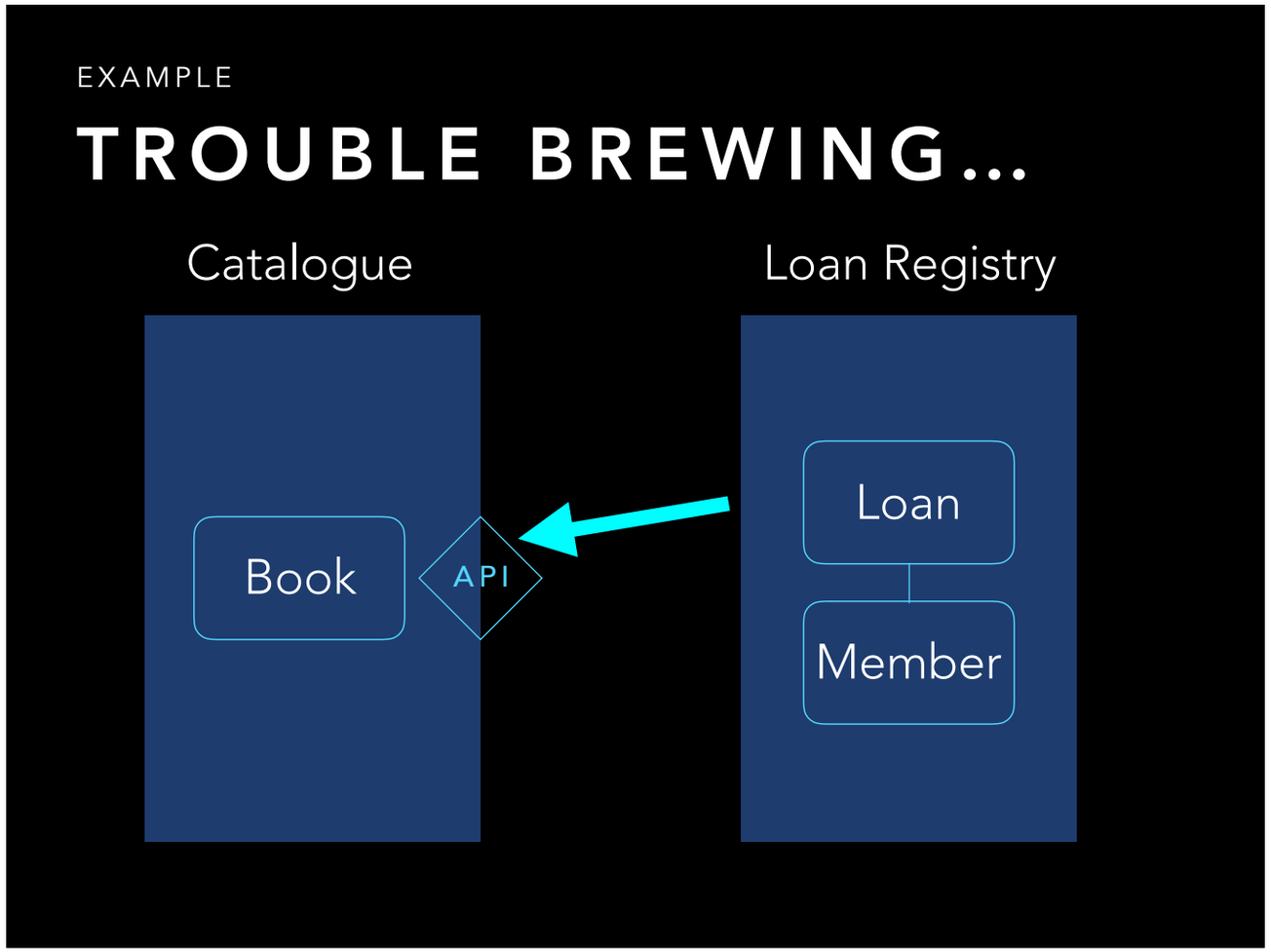


So we decide to have 2 services for our system

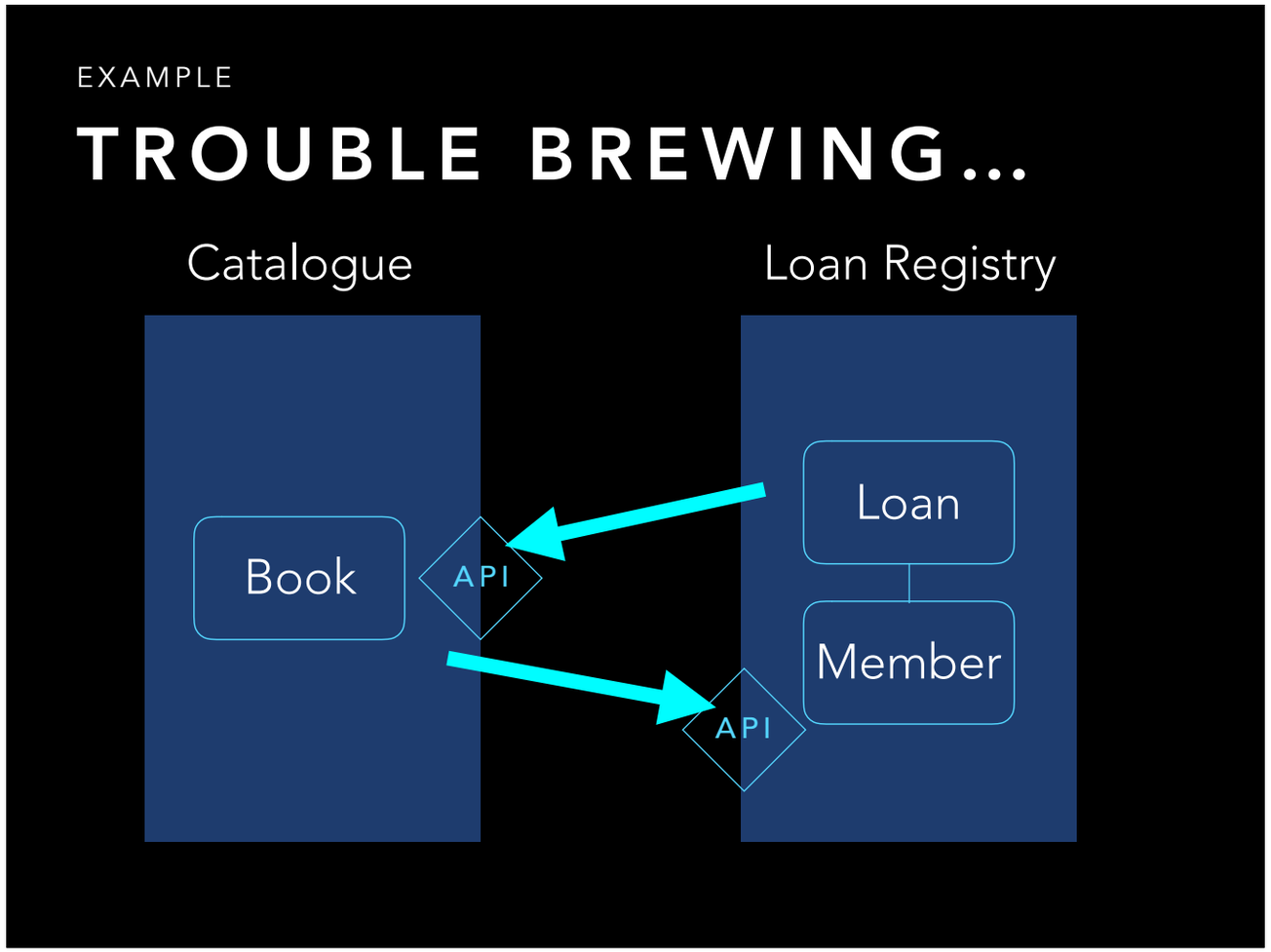
- The catalogue system records all the books the library has, makes them searchable etc etc
- The Loan Registry allows people to take out loans, do returns, renewals, send out overdue reminders etc etc



This seems like a reasonable decomposition because these are quite different functions.
Search vs registry. You don’t need to be a member to search the catalogue but you do need to in order to borrow etc. Catalogue doesn’t change as much as the loan registry - and can be cached out the wazoo. Maybe not really ‘small’ enough, but ok for this example…



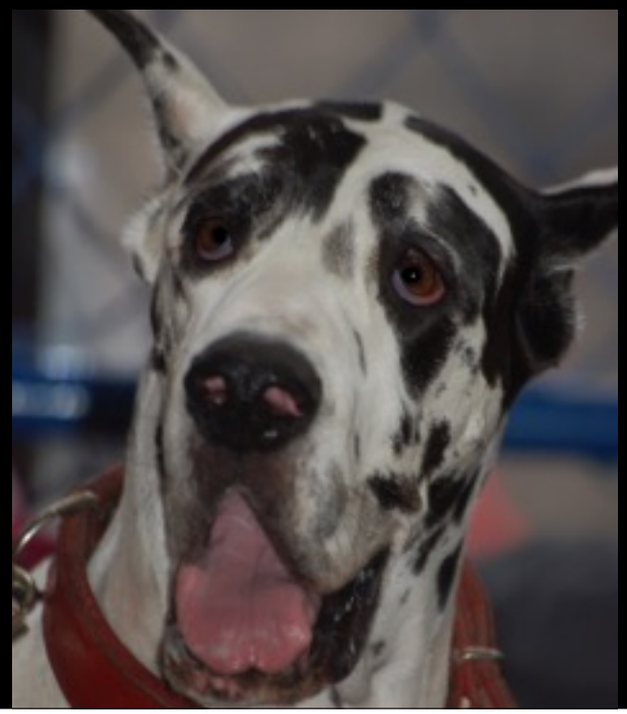
So when we come to implement it, we find that when you come to take out a loan the registry wants to know some information about the thing that is being loaned…
So we build an API and let the Loan Registry query the catalogue

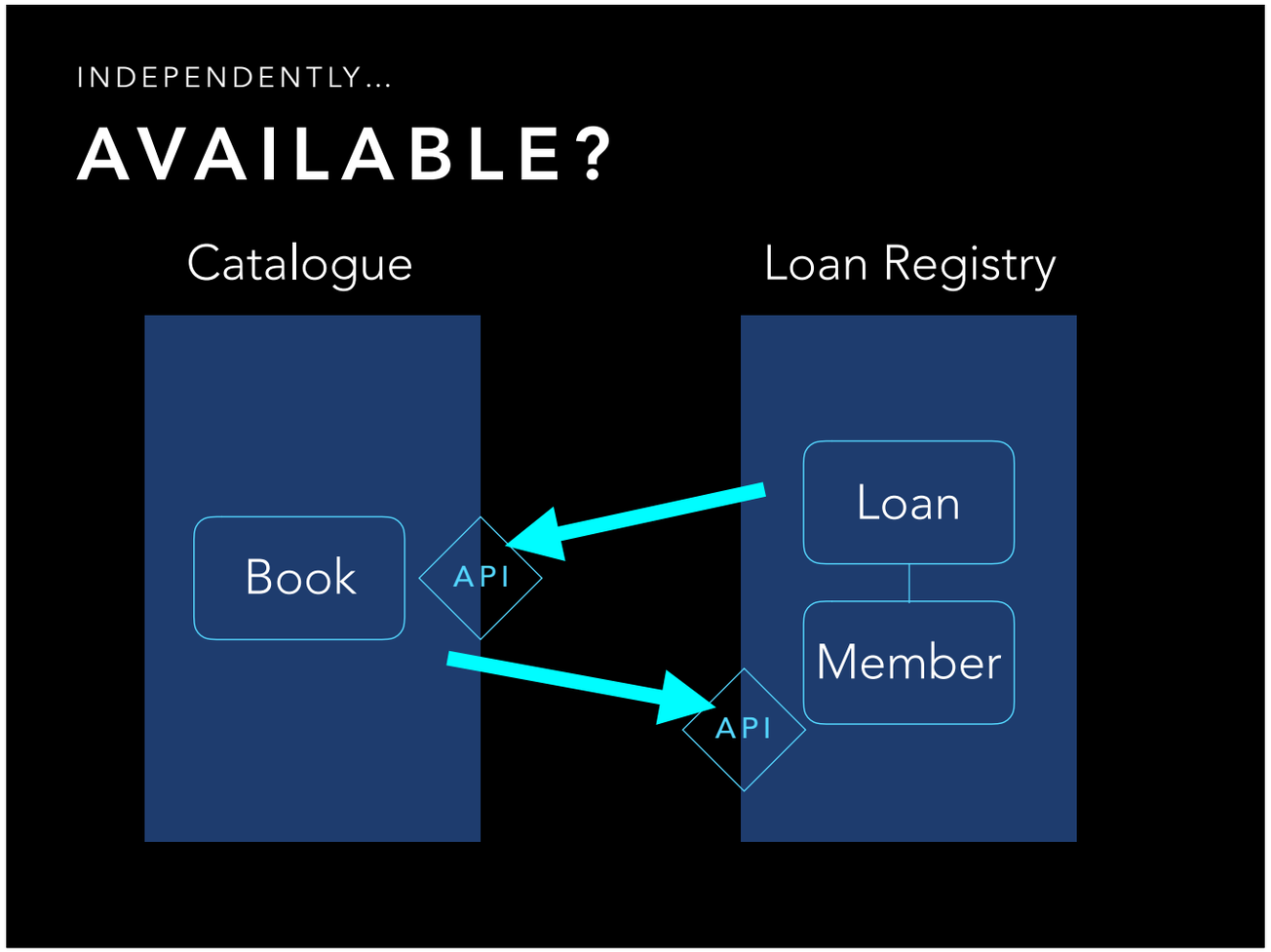


Then, someone comes up with a new requirement…
“Wouldn’t it be nice” Someone says “If when you looked up a book in the catalogue it could tell you if any copies of that book was currently available.
- So we build another API and make another call…

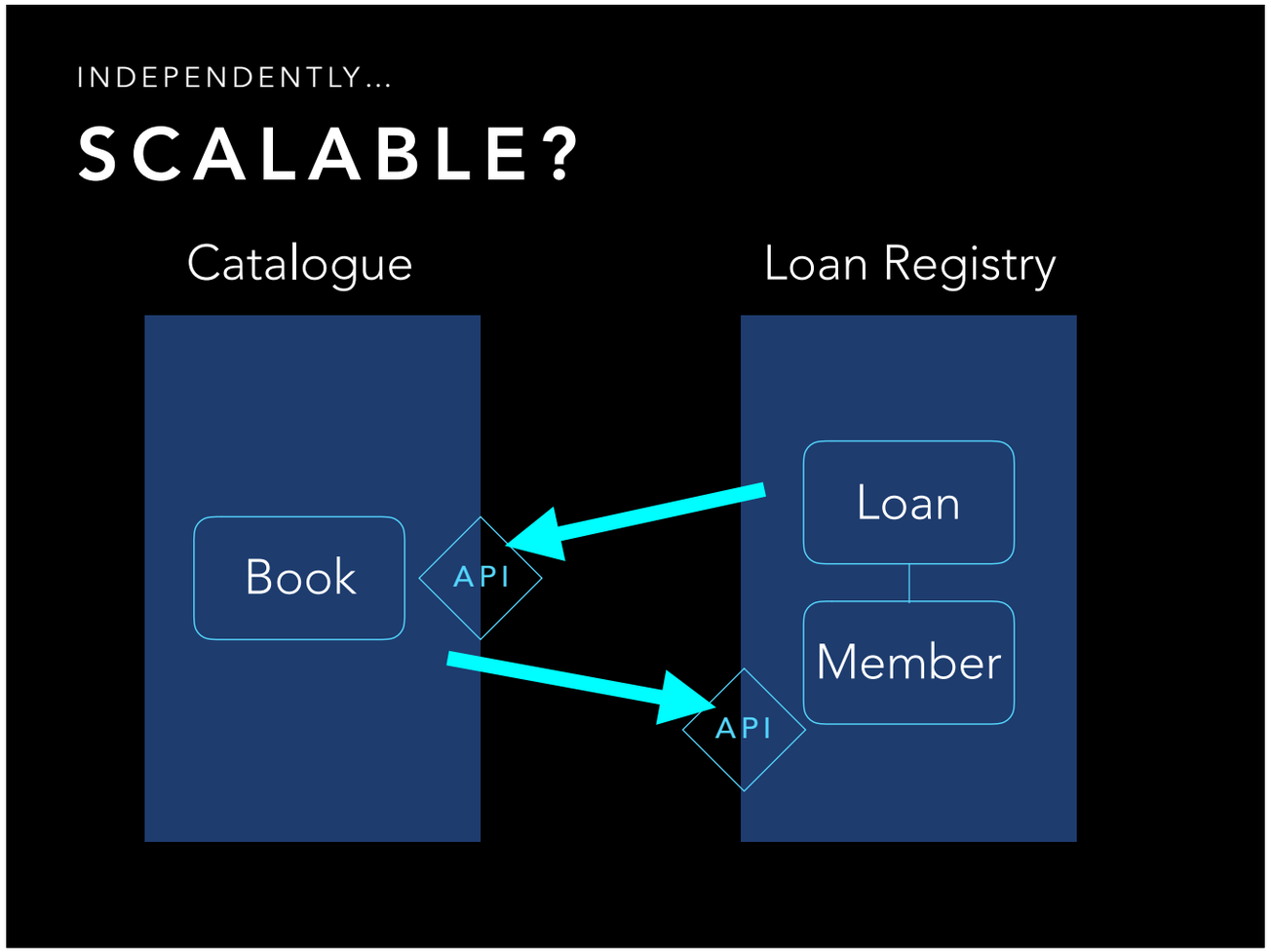
EXAMPLE

RUH ROH

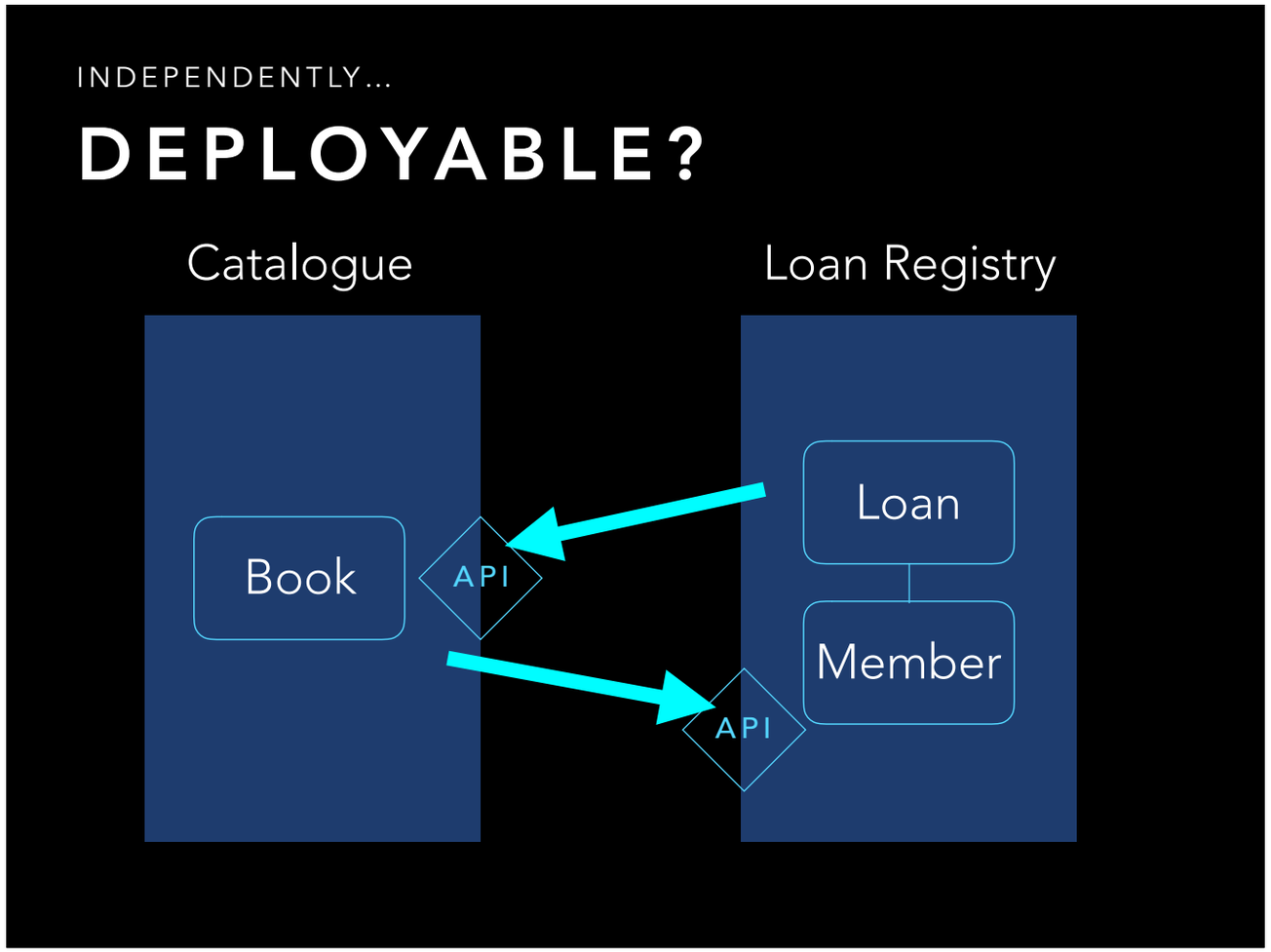




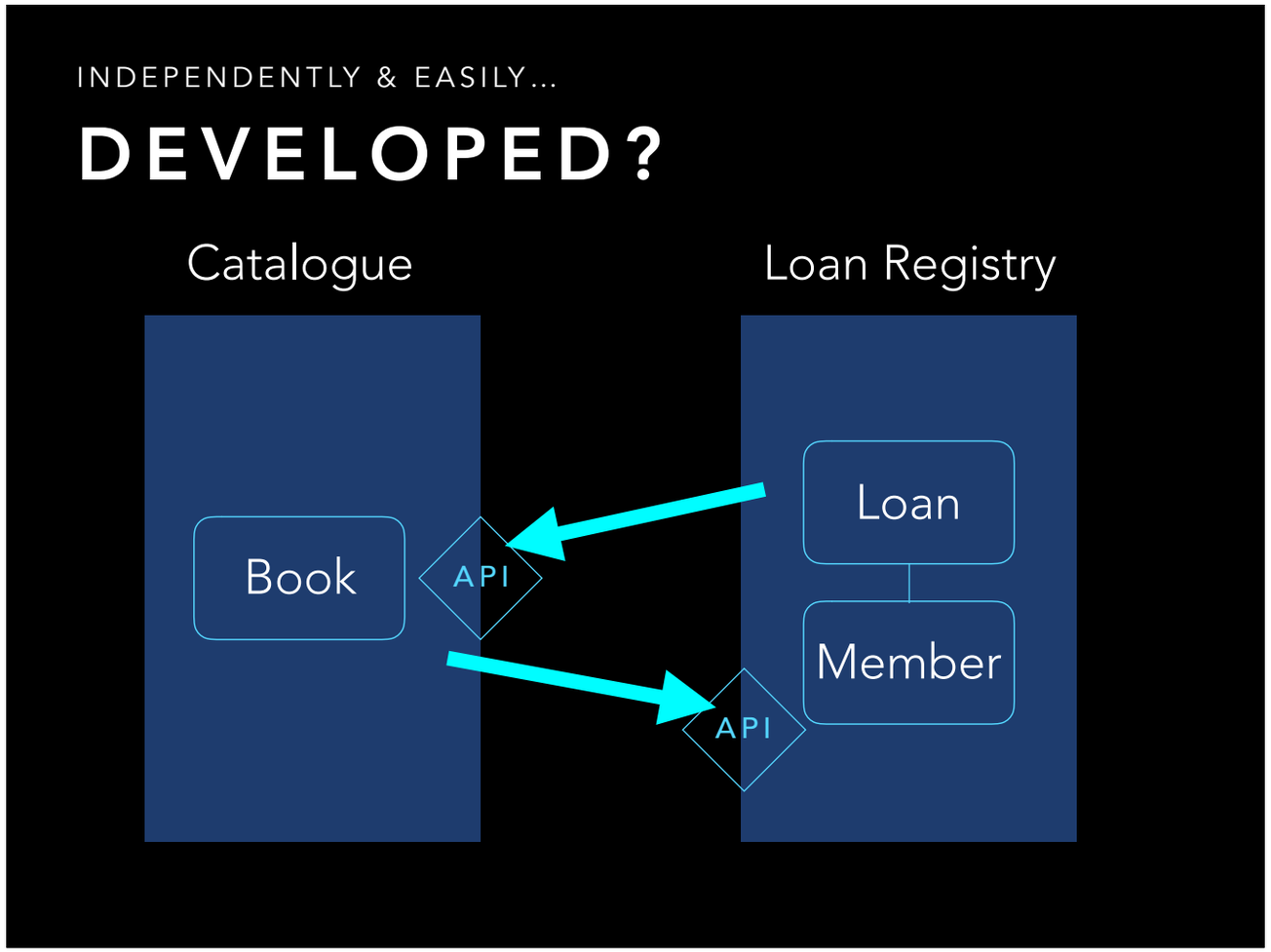
- Not really, because each service depends on the other…



- Not really, because now traffic to one service generates traffic on the other…



Suddenly…
- Not really independently deployable, because unless you make absolutely sure versions of those APIs are always backwards compatible, you are going to have to deploy these things in the right order something is going to break.

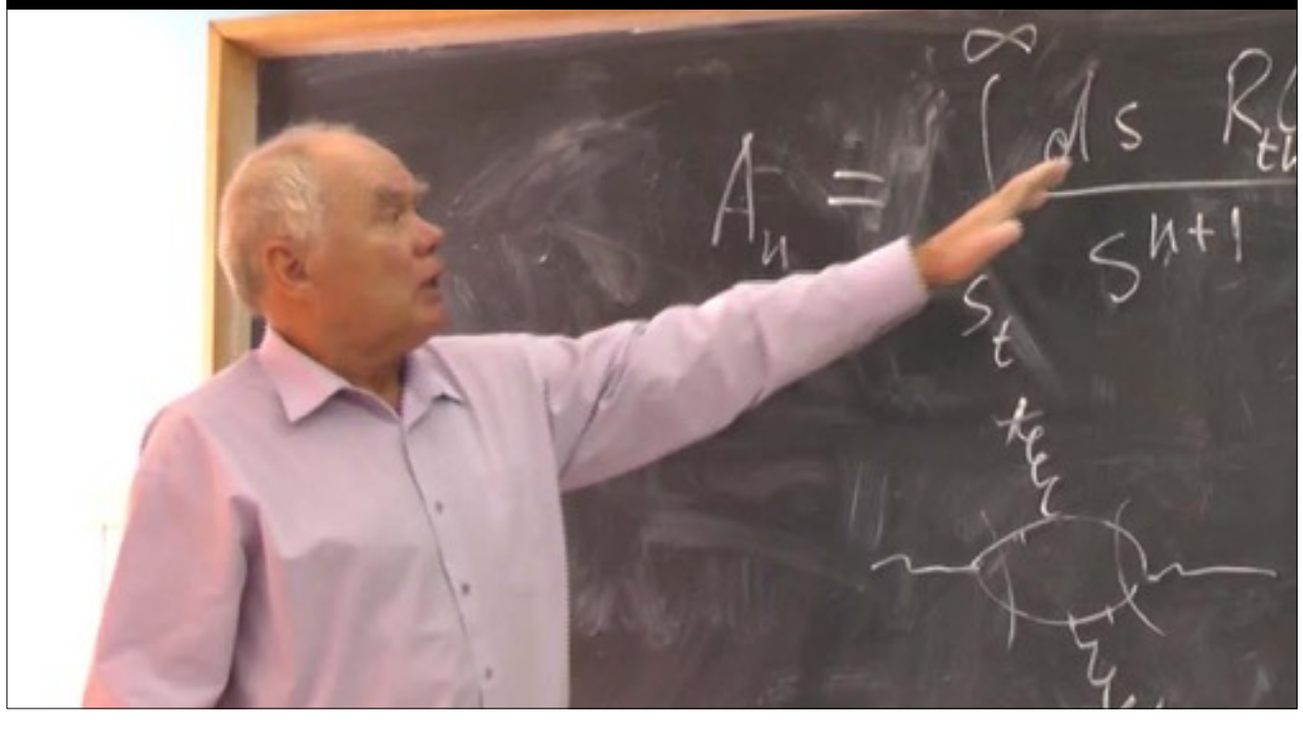


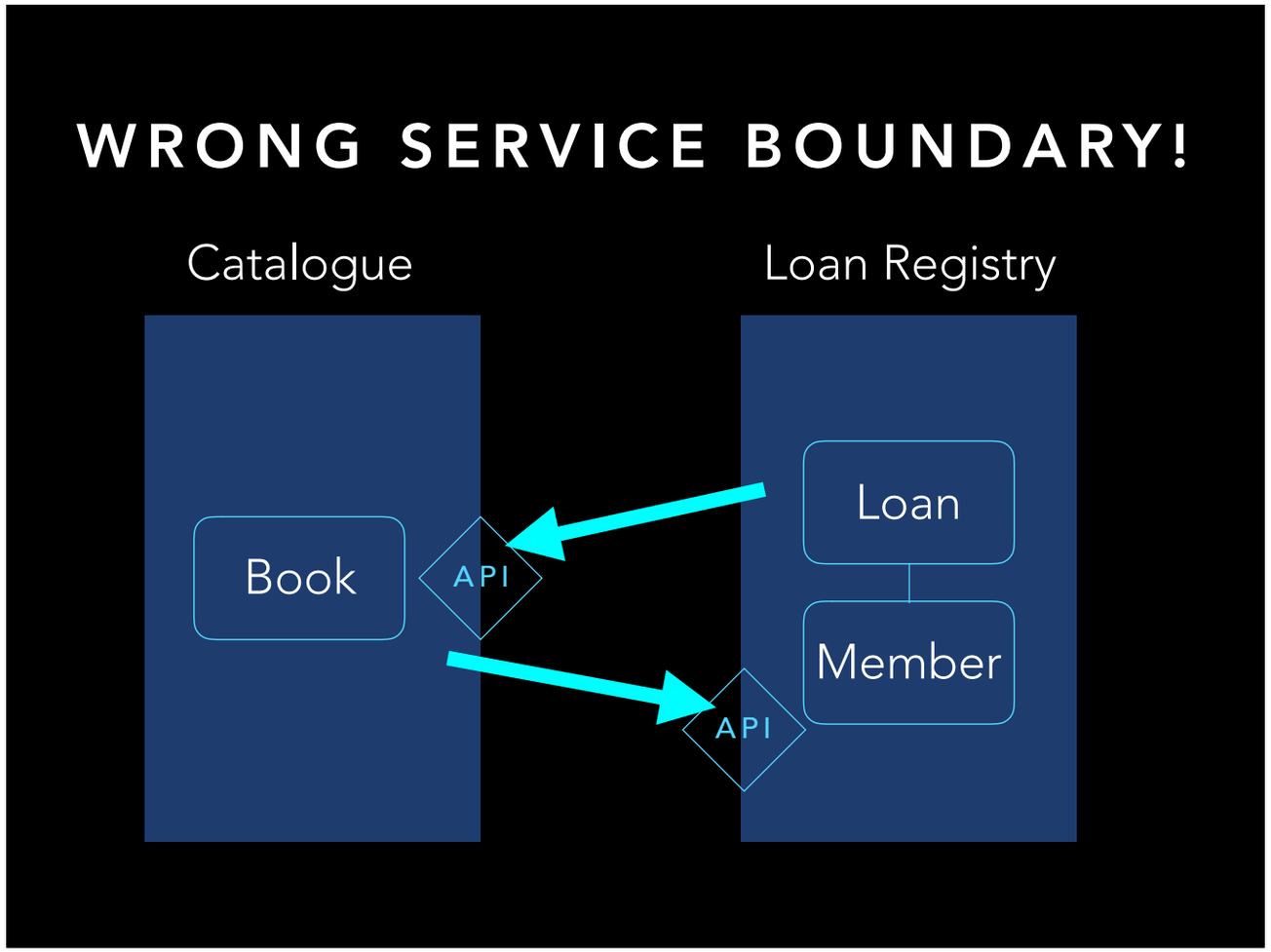
So with enough effort you can address some of those assertions I just made:

- You can build in circuit breakers to address availability
- You can put caches in to limit load impact (remember that cache invalidation is one of the 2 hard things in CS)
- You can make carefully design your API semantics to make them backwards compatible as often as humanly possible. REST will help here to some extent.
- Not really the nice simple development experience we were promised either.

WHAT WENT WRONG?

WRONG SERVICE BOUNDARY!

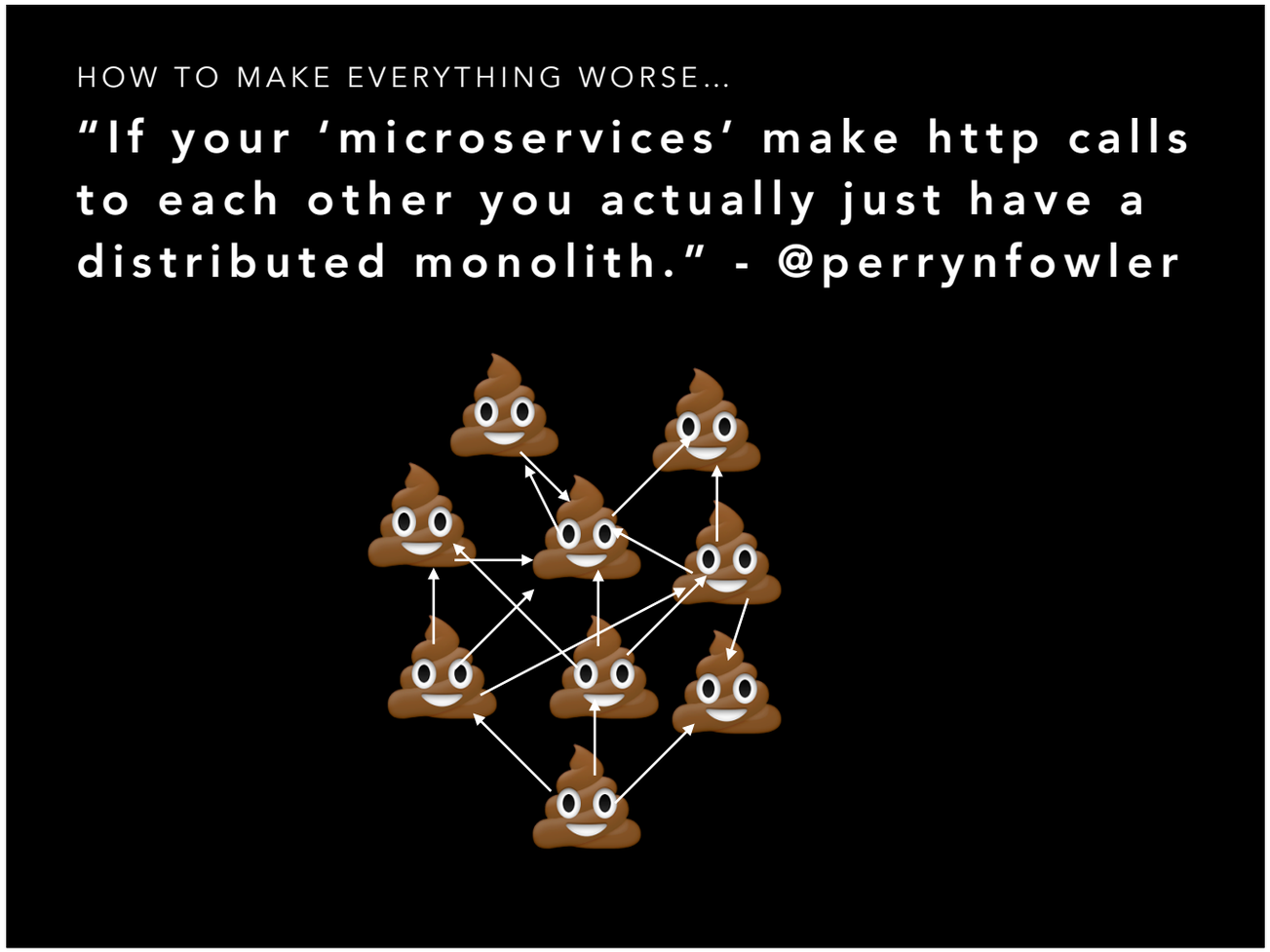




Book and Loan are too closely coupled to put in different services.
“So what concepts in my domain are a candidate for separating?”
Well, you probably shouldn’t break up domain concepts like that, you need a different type of ‘Bounded Context’, have you read Domain Driven Design?

AGILE ARCHITECTURE?

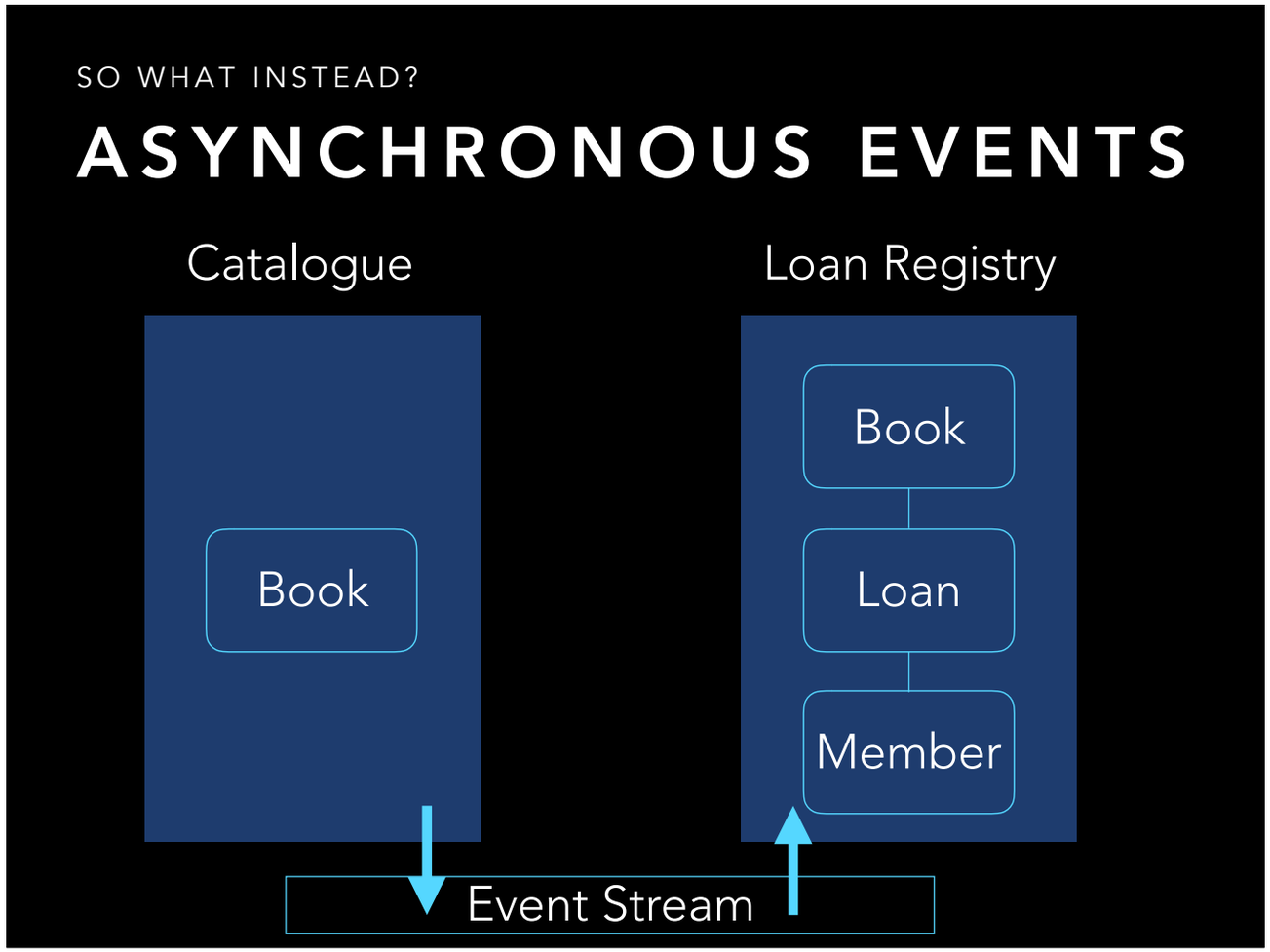
- but it seemed like a reasonable service decomposition when we started.
- I am made uneasy by any architectural approach that allows means I can paint myself into a corner if I don’t make exactly the right decisions up front
- So does this mean I think a micro service architecture is just a bad idea? well I wouldn't say that



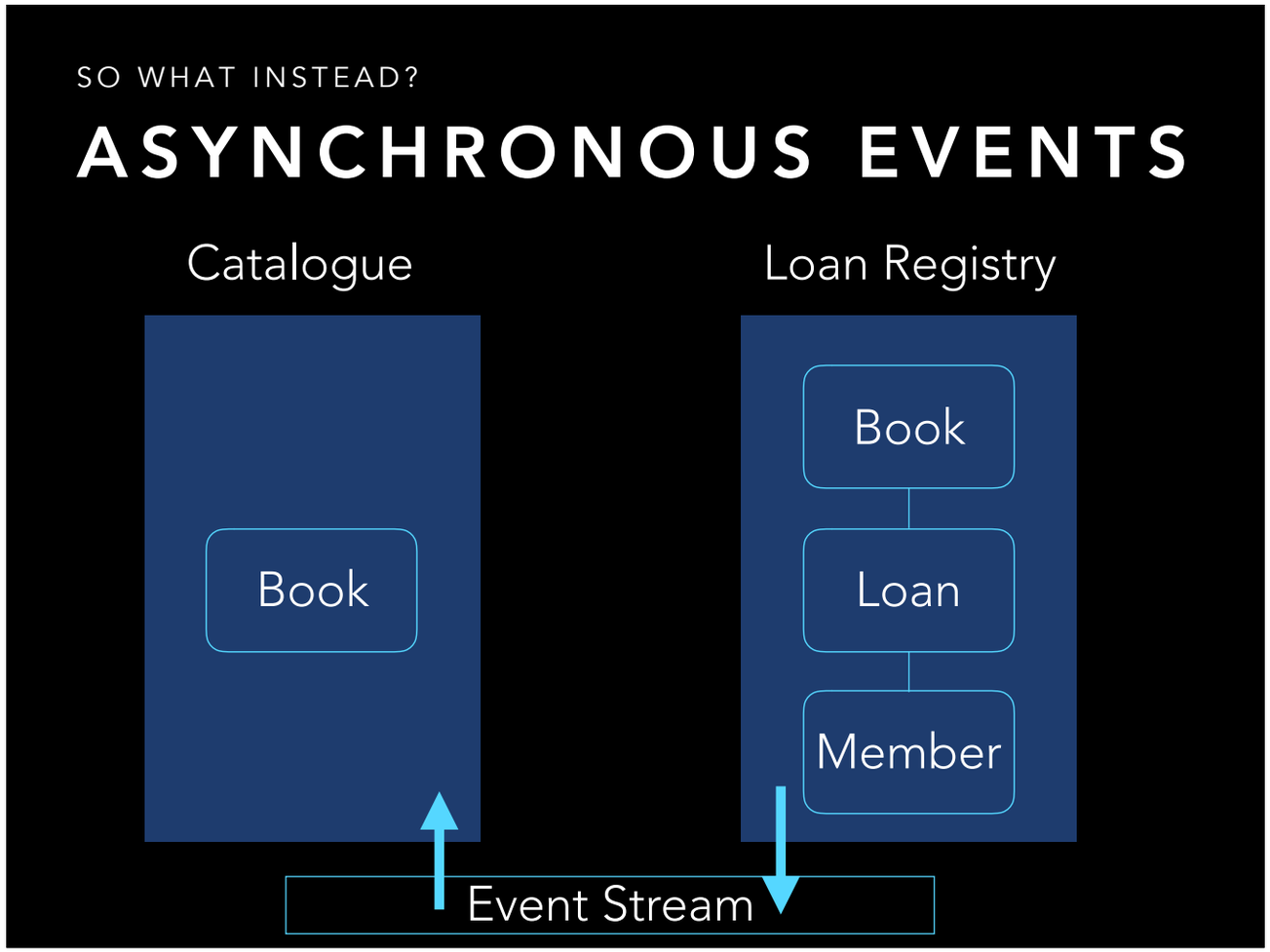
Here’s a controversial statement I like to make. Its under 140 characters hint hint.

or if I'm feeling particularly mean “congratulations you have just reinvented Enterprise Java Beans!”

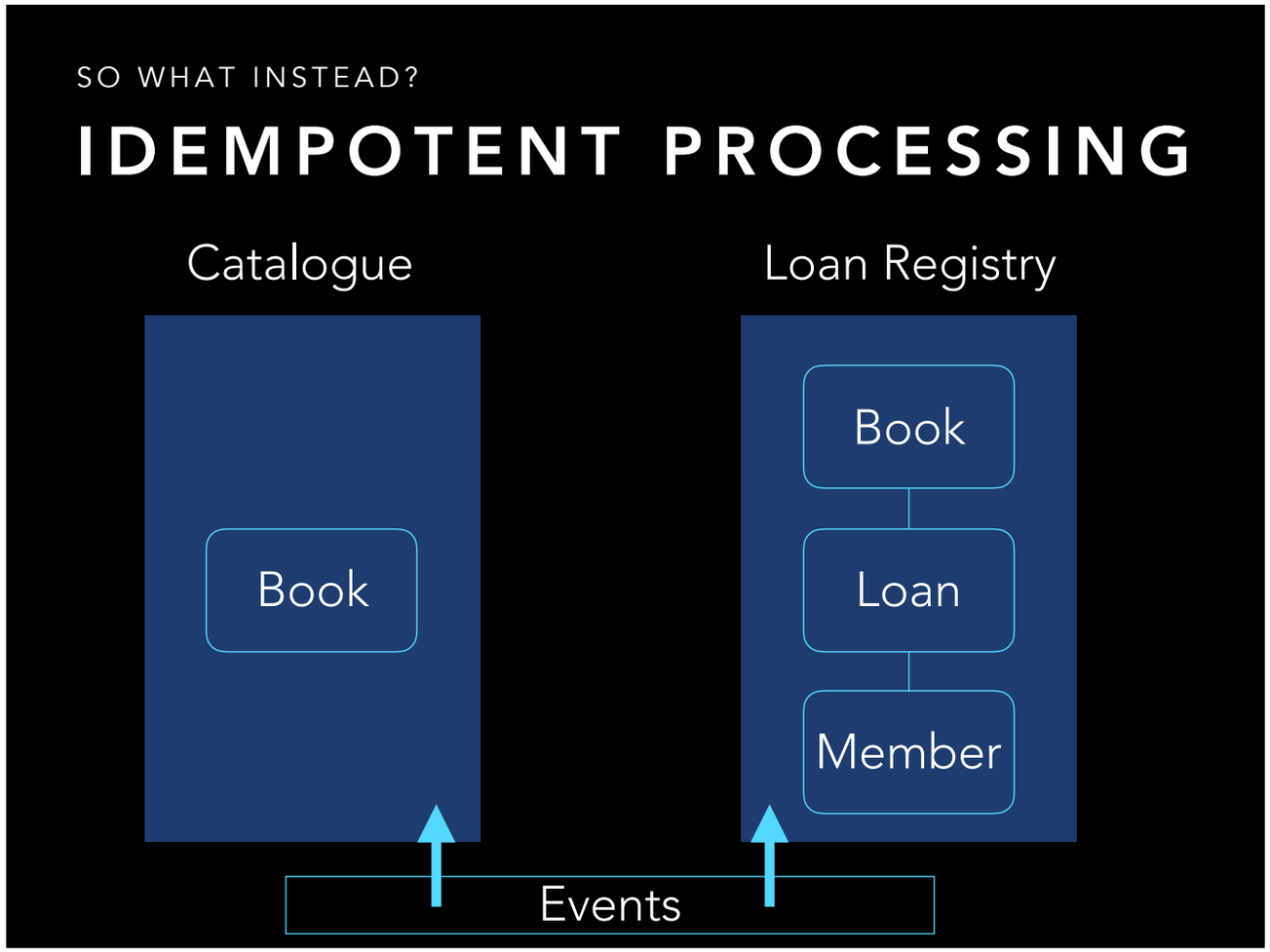
So is there another way to do this micro service stuff?



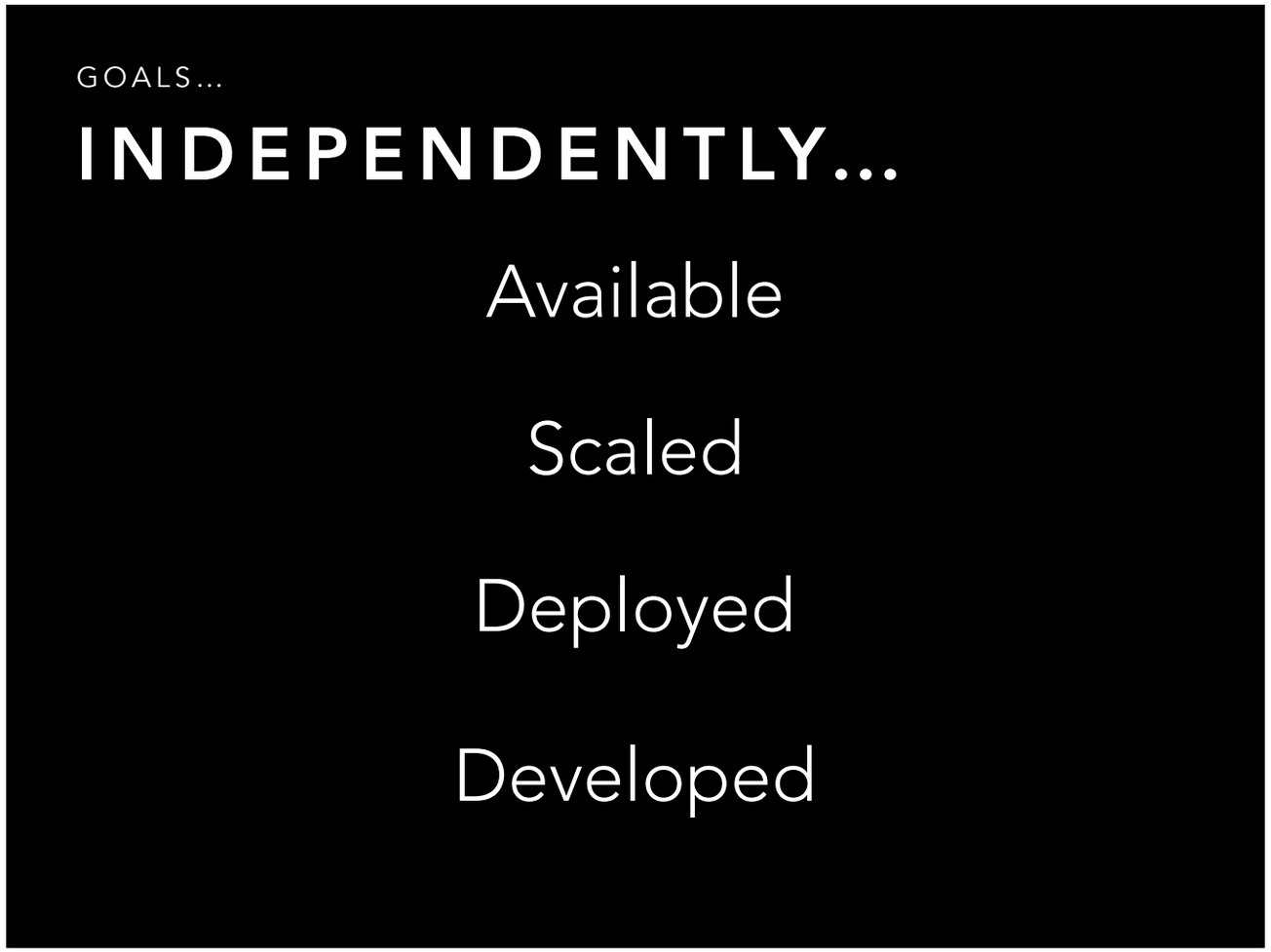
So how about instead of making synchronous we calls, we publish an event.
The catalogue can publish an event when it is updated and the registry can subscribe to that event and update any local information it needs.



And the loan registry can publish an event when a book is borrowed and the catalogue can update its availability count



And what if we design our processing so that its idempotent. That is you can process the same event multiple times and it will have the same effects once. So for example, if we process the same loan event twice, we don’t decrement the availability counter twice.



- Available: if none service stops processing events for a while it has no effect on the other
- Scaled: Extra traffic on one service does not generate traffic on another. Except maybe for more events to process. Much more resilient to BURSTY traffic - just work through the backlog. And if your events are shardable - just scale the streams by adding more.
- Deployed: you can deploy these things independently. If worse comes to worse and event processing starts failing. nothing goes down for a bit, just gets out of date and you cane re-process later.
- Developed. You only need to worry about other services when you are making changes to the events you publish. Also don’t need to know about all consumers - e.g. could add an email service listening to loan events.

BUT BUT BUT...

Duplication?

So something that worries people about this approach is that I seem to be duplicating data everywhere - I know have Book in both services. My advice is don’t worry about it - disk space is cheap and whats more you’ll probably find that its not really duplication. Each place you store this data has a different use and is likely to have a different shape if its optimised for that use - and you sendup with something that is actually BETTER than the God model that tries to be all things to everyone.

BUT BUT BUT...

Consistency?

Unless your building a high volume trading system or something - this is probably much less of a problem than you think…Eventually Consistent is fine.

DURABLE EVENT STREAMS, MULTIPLE IDEMPOTENT CONSUMERS

LOG BASED ARCHITECTURE

So this sort of approach where you have DURABLE EVENTS STREAMS

- by durable I mean the event sticks around in the stream for a while. Doesn't have to be forever, just long enough for you to be able to replay, say, the last hours events.

and MULTIPLE CONSUMERS is known (at least in some circles) as LOG BASED ARCHITECTURE. (because if you think about it a simple way to implement this is to write the events to a log file and have each consumer tailing it)

and IMNSHO is the only way to build a MS architecture.

Ok, so thats my micro services rant - what was I talking about?

Oh yea - IoT - what has this all got to do with the I in Iot and how we can deal with it?

Well I glossed over how much commitment you need to use this approach. You need to decide to publish events and what events you are going to publish.

IOT

READY MADE EVENTS

well for a LOG based architecture you need to publish events, but with IoT devices we already have that. Each device is out there sending us nice independent little packets of information, so LOG based architecture fits really well, and we can get all those benefits we talked about

If you think about it a lot of mobile app interactions are events too..

DECOUPLED

SCALABLE

- processing decoupled from collection (so if processing gets slow…)

So everyone talks about the coming explosion of IoT devices - so you want to make sure you can scale with that. LOG BASED arch means that your ability to accept data is decoupled from your ability to process it.…

SHARDABLE

SCALABLE

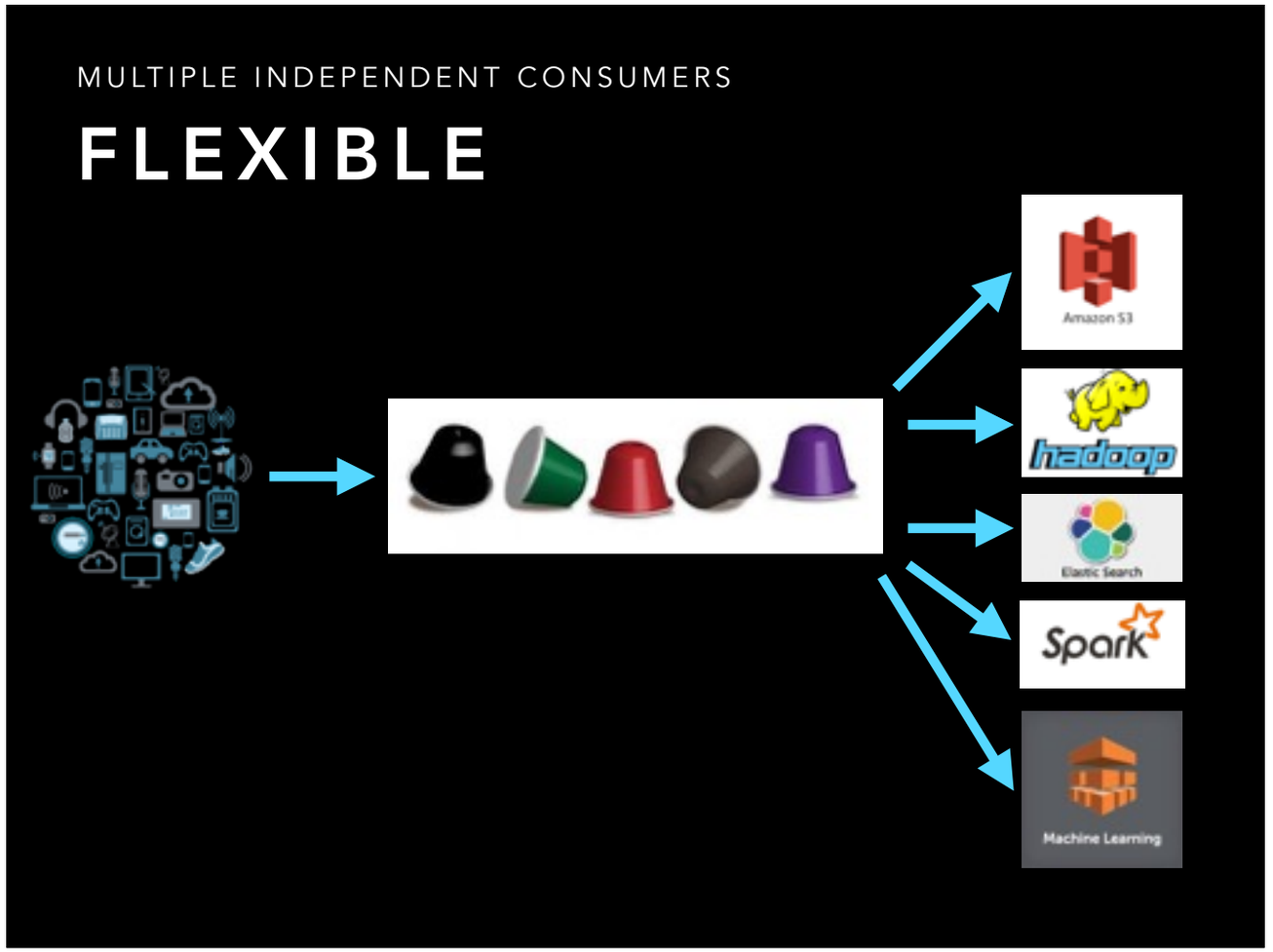
- data from devices is usually very shardable into seperate streams

So everyone talks about the coming explosion of IoT devices - so you want to make sure you can scale with that. LOG BASED arch means that your ability to accept data is decoupled from your ability to process it. Also events from individual devices, phones etc are immediately shareable. You can create a stream per device if you need to…

DURABLE STREAMS, IDEMPOTENT PROCESSORS

FAULT TOLERANT

- If the processor goes down, we don't stop accepting data
- The processor develops a transient fault, it can just stop for a while try again later
- If we break the processor we can fix it and replay the events..

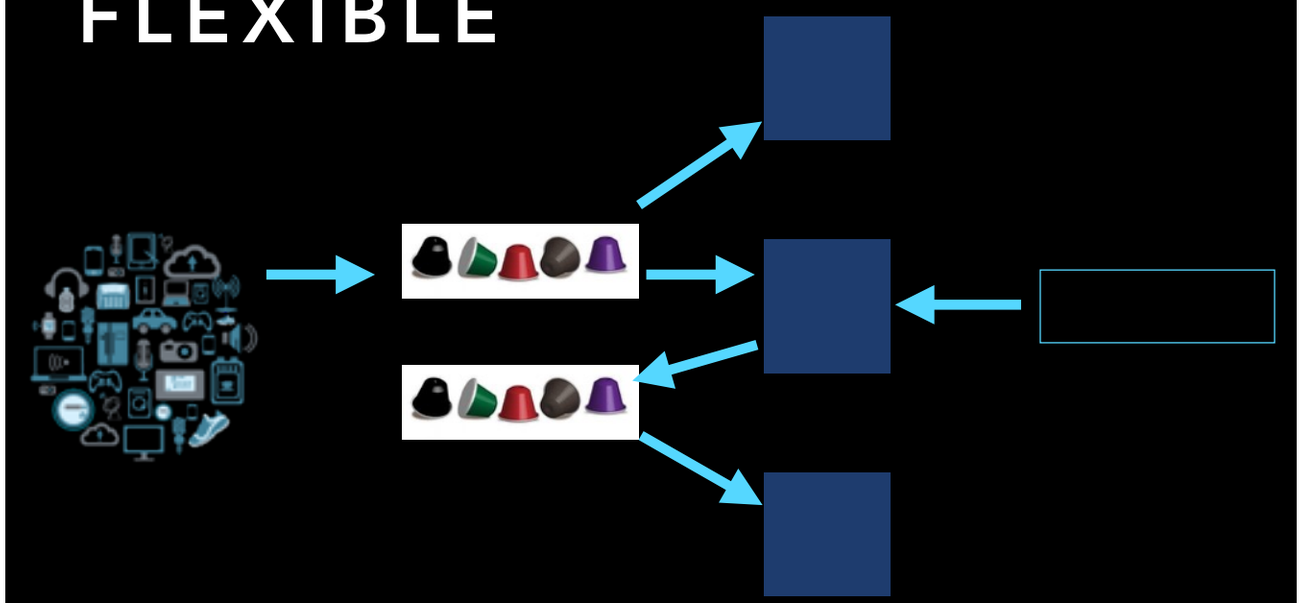


Theres all sorts of direct tech out there that might help, and its hard to choose the one that will best suit your requirements - well you can try them all if you want. And you can try it with your data.

Or if you are writing your own stuff, you can choose different languages for different problems. 9ruby, clojure, R whatever)

TRANSFORM AND REPUBLISH

FLEXIBLE

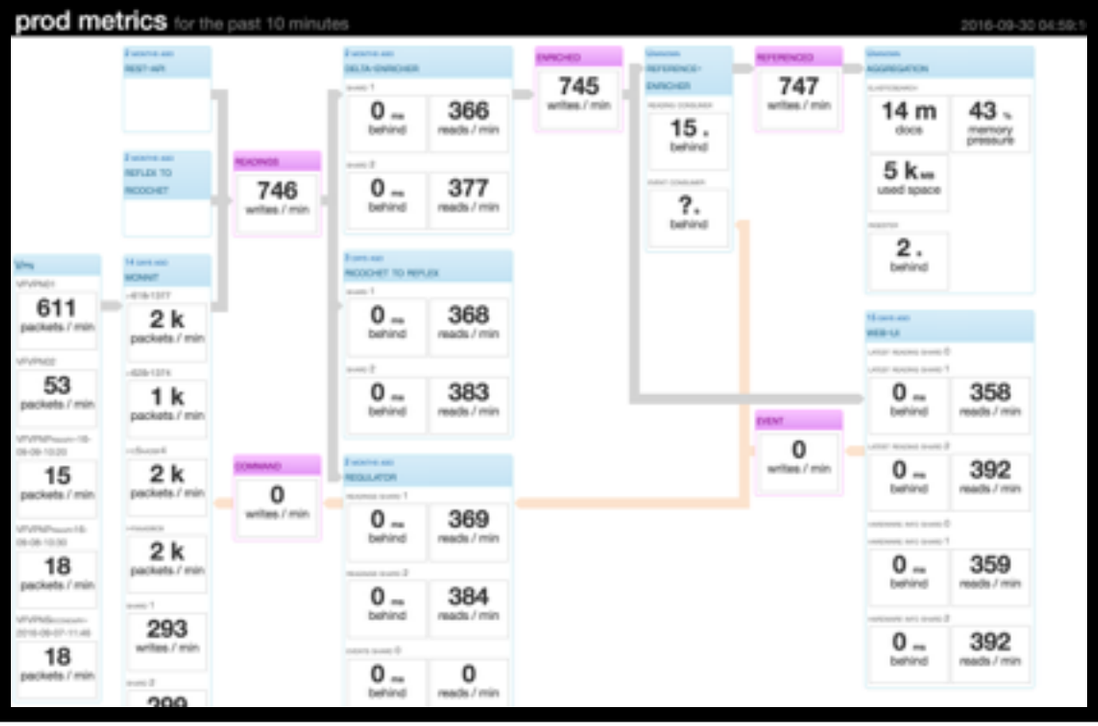


So what do we actually do at Urbanise

- we use kinesis for event streams. (another option is Apache Kafka)
- we use a number of different technologies for implementing our services
- and at the moment we're using either dynamodb or postgres for implementing persistence

(whichever suits the problem best)

URBANISE


prod metrics for the past 10 minutes
2016-09-30 04:59:1
611 packets / min
53 packets / min
15 packets / min
18 packets / min
18 packets / min
2 k packets / min
1 k packets / min
2 k packets / min
2 k packets / min
293 writes / min
746 writes / min
0 writes / min
0 ms behind
366 reads / min
0 ms behind
377 reads / min
0 ms behind
368 reads / min
0 ms behind
383 reads / min
0 ms behind
369 reads / min
0 ms behind
384 reads / min
0 ms behind
0 reads / min
745 writes / min
15 . behind
?. behind
747 writes / min
0 writes / min
14 m docs
43 % memory pressure
5 k MB used space
2 . behind
0 ms behind
358 reads / min
0 ms behind
392 reads / min
0 ms behind
359 reads / min
0 ms behind
392 reads / min

AND THEN?

- Kinesis Analytics (not even on slide)
- Machine Learning!

QUESTIONS?

@perrynfowler

REMEMBER TO REPEAT QUESTION