Serverless MapReduce with .NET

Chris Priest

@cjrpriest

- My name is Chris Priest
- I work for Amido / ASOS
- I know cloud
- I know C#
- I'm a bit of a data nerd
- I'm not an MR expert, but I've done a bit of stuff

- Above all I like to solve real problems with tech

This was a “journey”

AMIDO

3

- I thought this would be relatively easy area to look into
- From what I can see, there doesn’t appear to be any previously published work covering Serverless, Map Reduce & .NET – obvs those individual topics have lots of coverage on their own
- I had about 4 weeks of evenings available to me, turned out that it was actually quite hard to build a working prototype in that timescale
- So this talk is, in part, about a journey that is not yet complete – I want to share with you the good & the bad, where I am now, and I’d love your feedback aftwerwards

CONTENTS

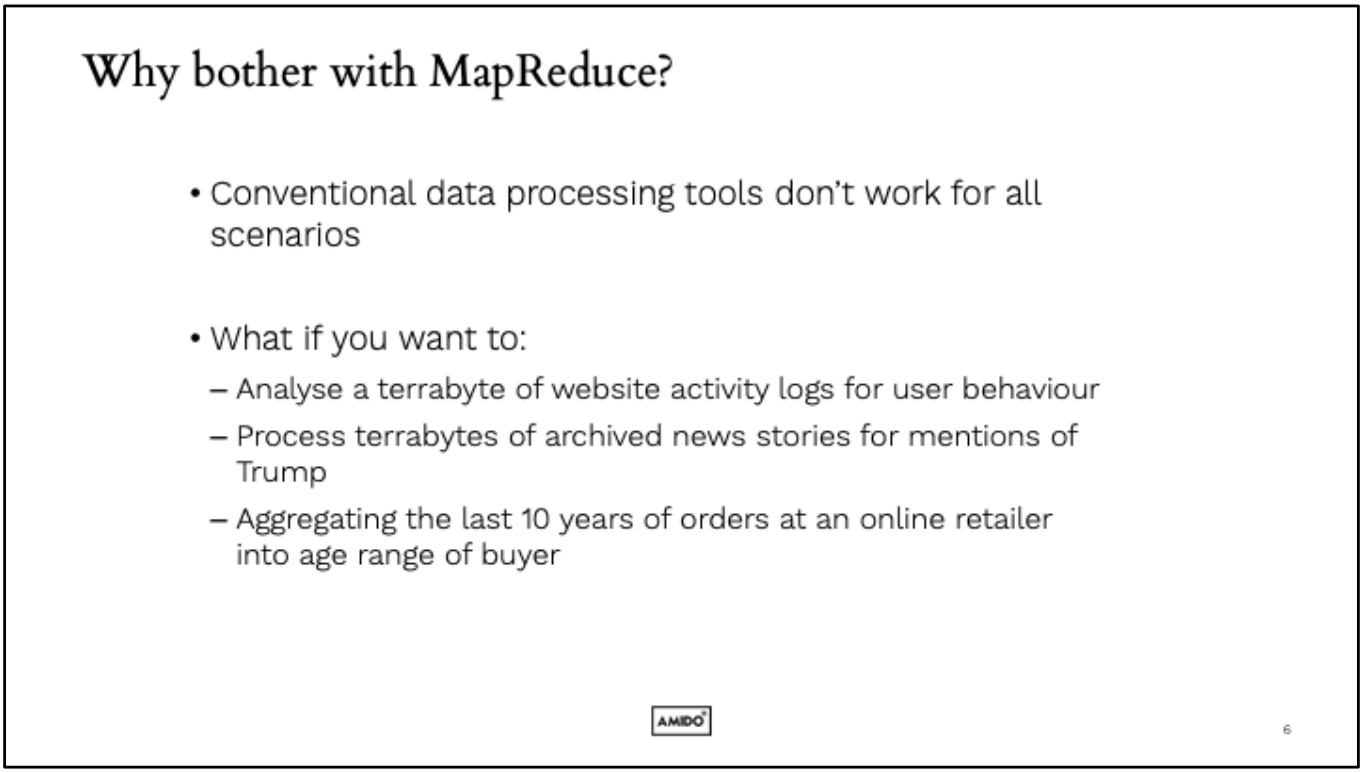
Intro to MapReduce
No previous experience necessary

What is Serverless?
Including a look at pros & cons

Bringing the two together…
…from architecture to code

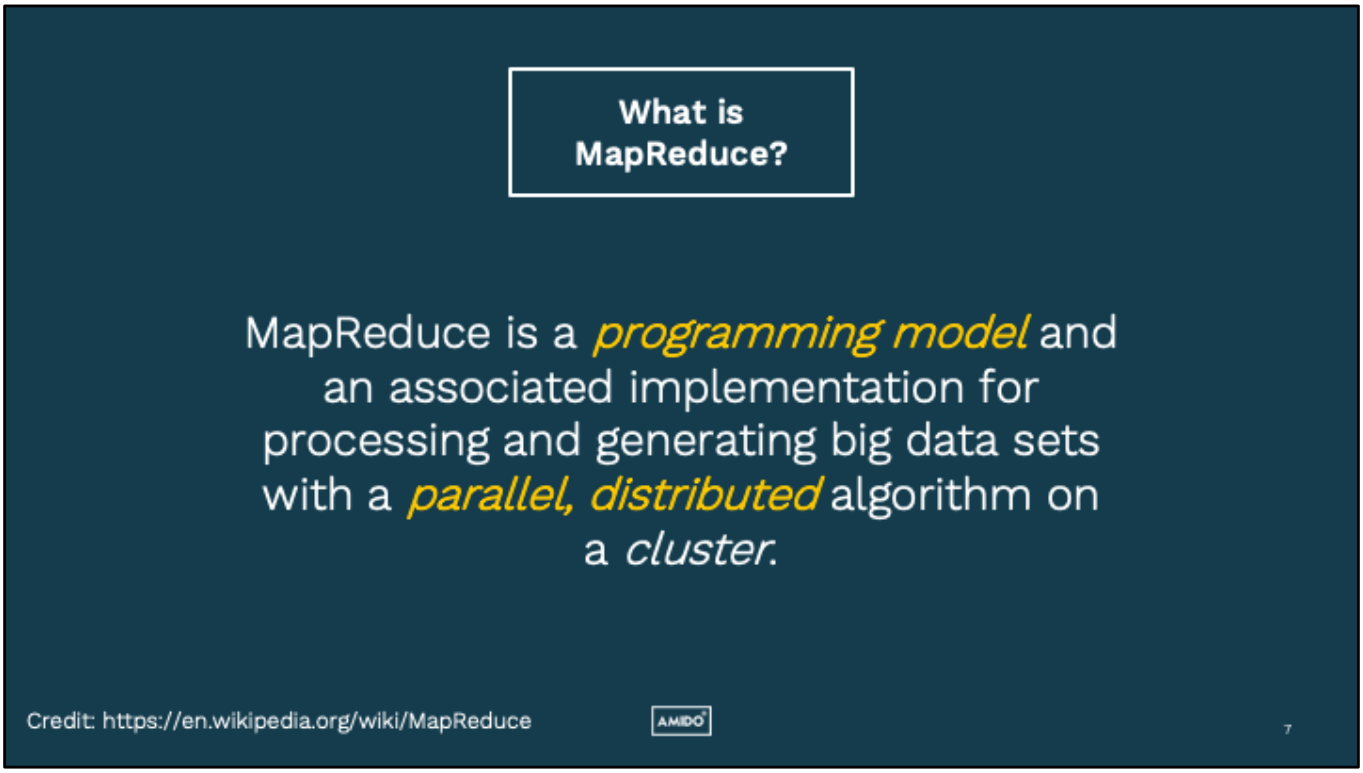
Critique
Comparison with other options


Intro to MapReduce…



- First of all, before we get into the detail, forgetting .NET, & even serverless, what sort of problems are we trying to solve with MapReduce?

- The kind of scenarios that conventional data processing tools don't work in, are related to large data sets, usually that do not have any structure, e.g. flat files



MR is a programming model, not a library

This bit is important

There is nothing about the MR programming model that means you should use java, linux, hadoop, HDInsight, or even servers!

- It is a model that is designed to leverage a parallel running & distributed system
    - That is, it is actually designed to work in, amongst others, the cloud and leverage lots of available compute to complete a given task sooner

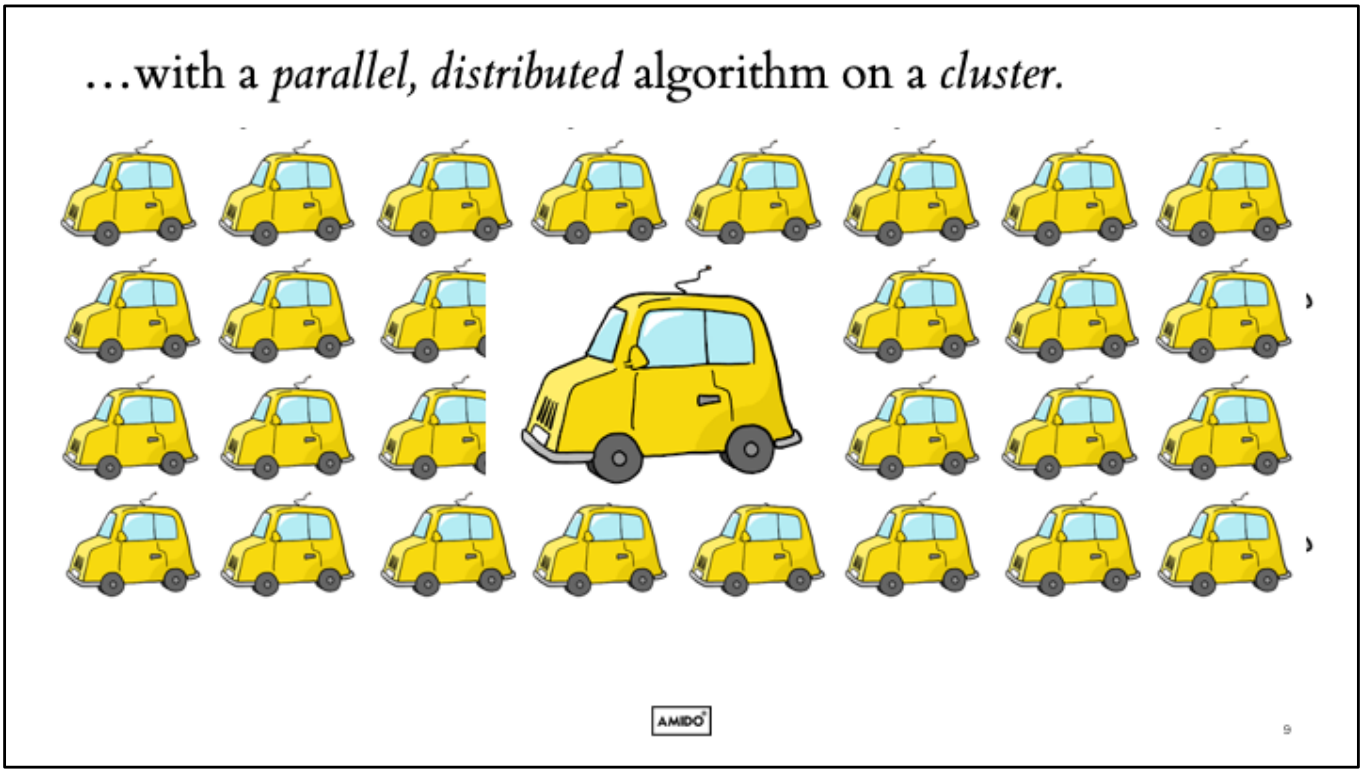
- Going to go through an analogy, some theory and the look at some code

Free ourselves from just trying to make stuff run *faster*…

This is other important bit for me

We can free ourselves from trying to make something complete sooner just by making execution faster – there are practical limits to how far you can go with this

- If we want to get 3 or 4 people from A to B, we could do so in this car.
- What if wanted to get 7 or 8 from A to B – it's going to take two trips… that's no good, will take twice as long
- So… let's optimise, spend a lot of money developing a faster car… now this car is 4 times as fast… we can move the same number of people in the same timeframe
- Now we want to move 20 people, 40, 80… we could keep developing faster cars, but we are likely to get much lower returns as we optimise
    - i.e. the same performance increment costs more and more, until you are paying huge sums of money to get just a little bit faster



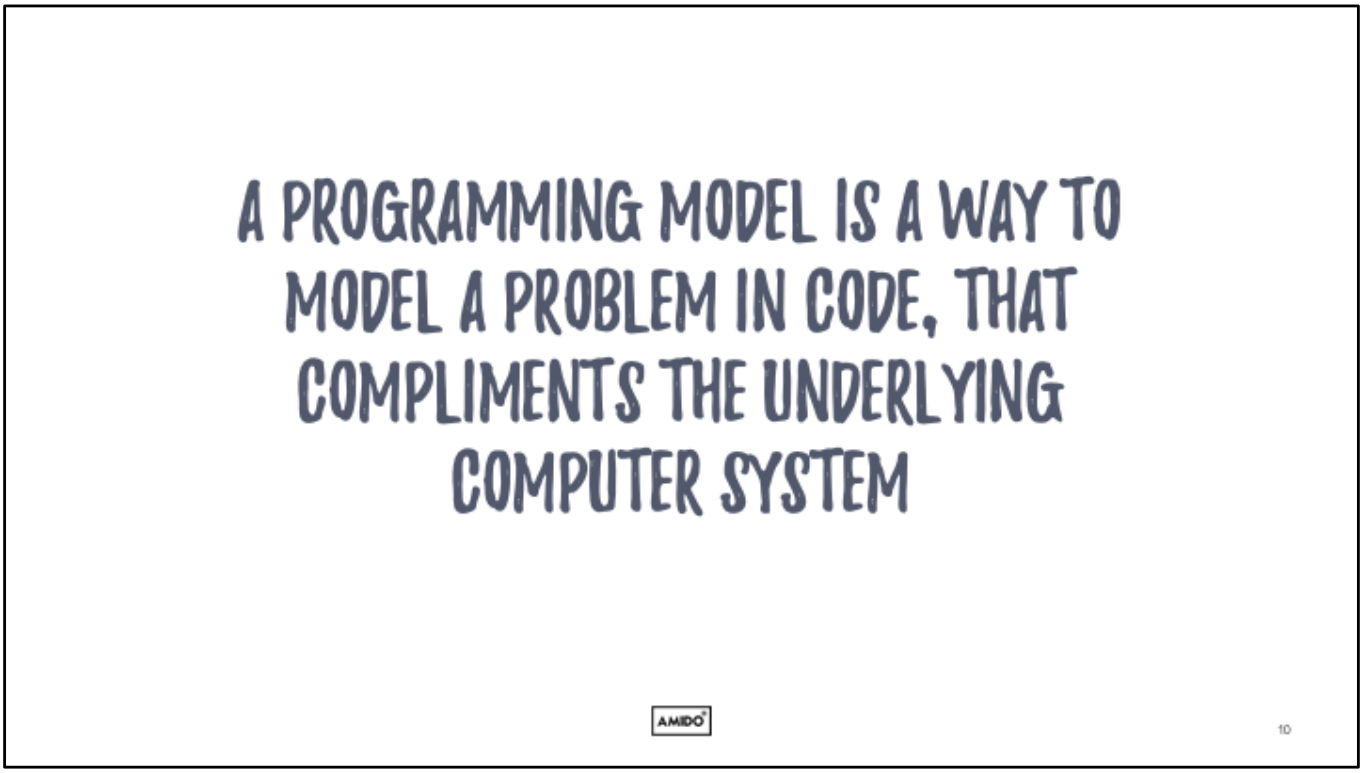
- What if you could change the approach?
- Instead, spend money on buying many small & slow cars?

- Actually then you then have capacity increasing with cost
- And another benefit, if a car breaks down, the vast majority of the work still gets done – you are much more resilient to issues

Make it complete sooner by executing more slowly, but in parallel

There are practically no limits in this scenario – you get much more bang for your buck

- In much the same way, Map Reduce allows you to reframe a problem in a way that can be worked on in mass parallel, on not particularly highspec hardware



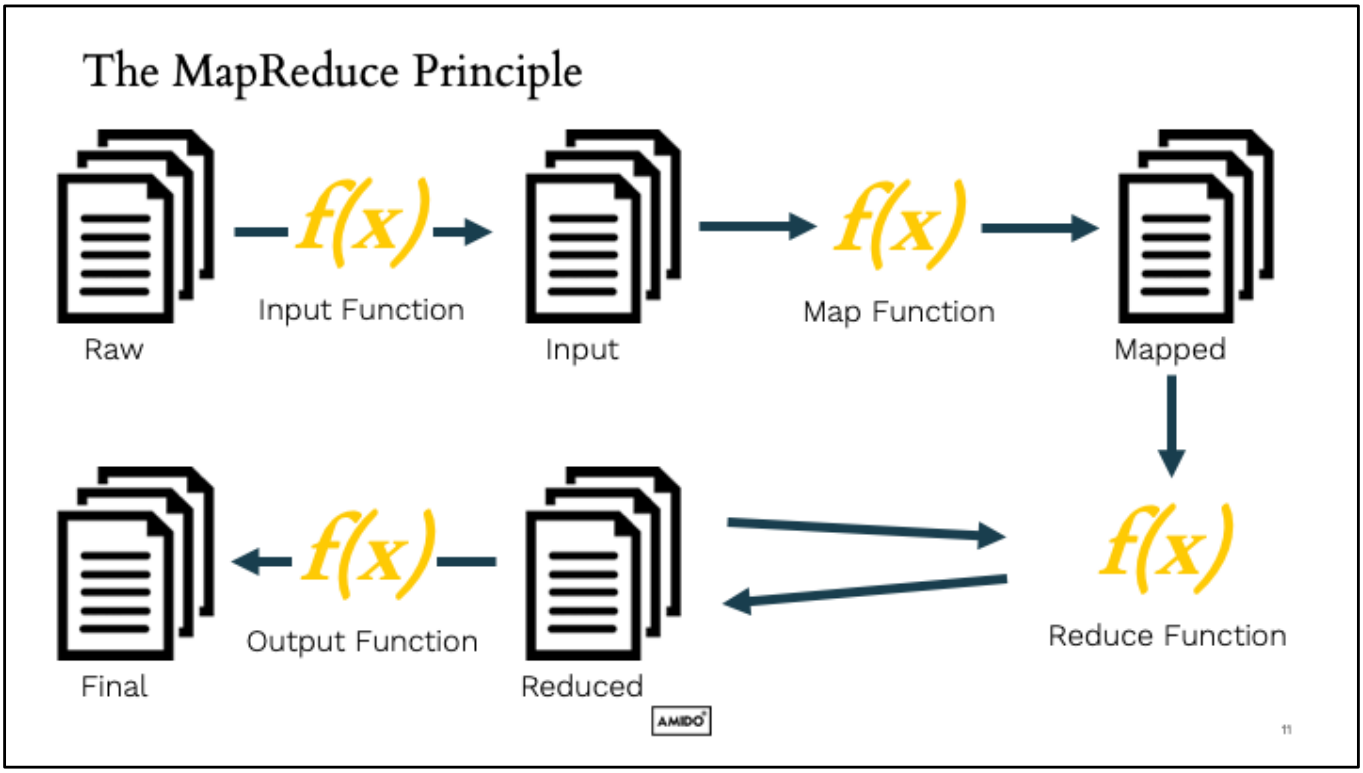
MapReduce is a programming model

- Not about language
- Not about OS / hardware

- It helps us to think of a problem in a way that can be more easily executed by an underlying system

- So in the MR example, the programming model will help us to express our the algorithm we want to apply to our data, in a way that can leverage a parallel running / distributed system

- I like to think of a programming model as a guidance system, something that guides you on the path
    - It can't totally prevent you from abusing the system… but it can persuade you not to



- Start with some input data
- Apply some map function to this data,
  - Outputs some key/value pairs
  - So then we have a set of mapped data
- Apply some reduce function to this set of key/value pairs
  - Essentially takes a set of key/value pairs and reduces this to one key/value pair for each key
  - Reduce function can take output of reduce function as input
    - You can keep reducing until you can't reduce any further

Map & Reduce Example

Map Phase

Input Document

```
code here code
there code
everywhere
```

Output

Key	Value
code	1
here	1
code	1
there	1
code	1
everywhere	1

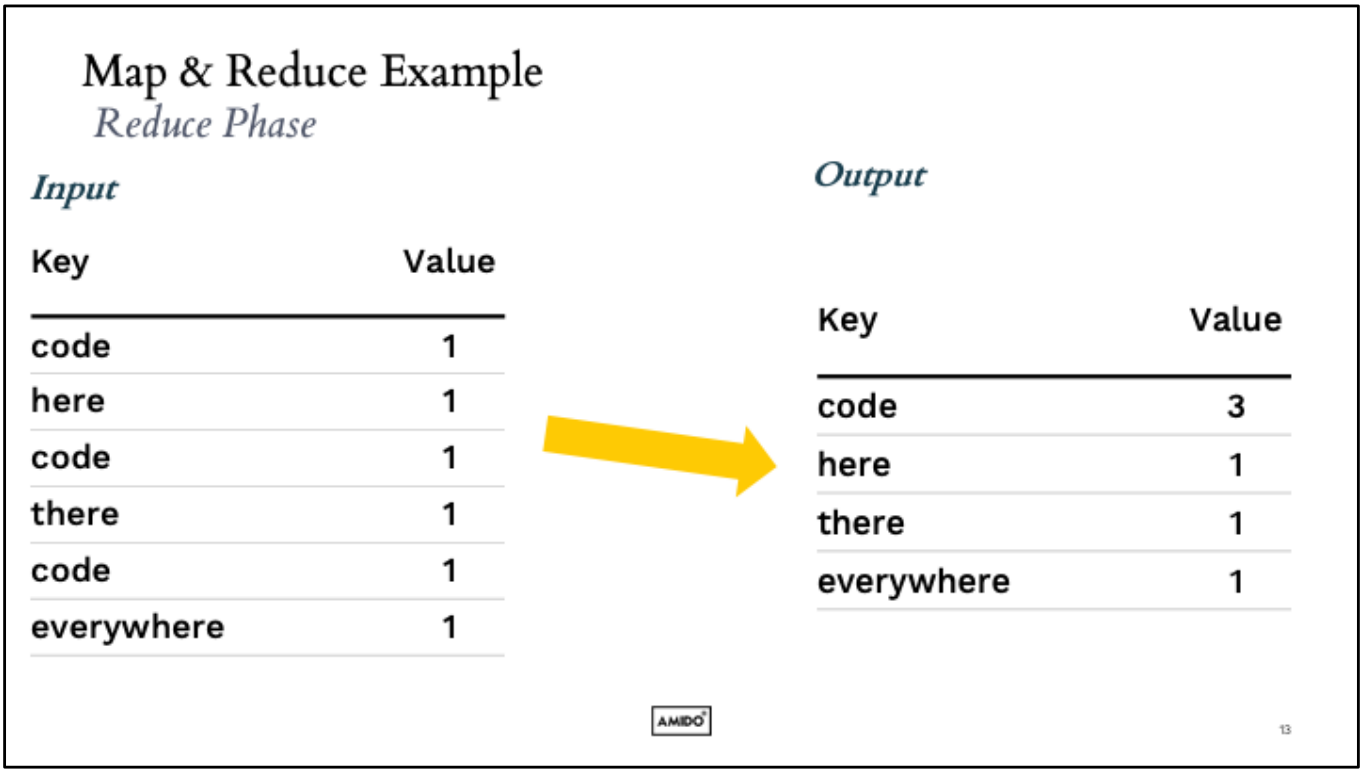
AMIDO

12

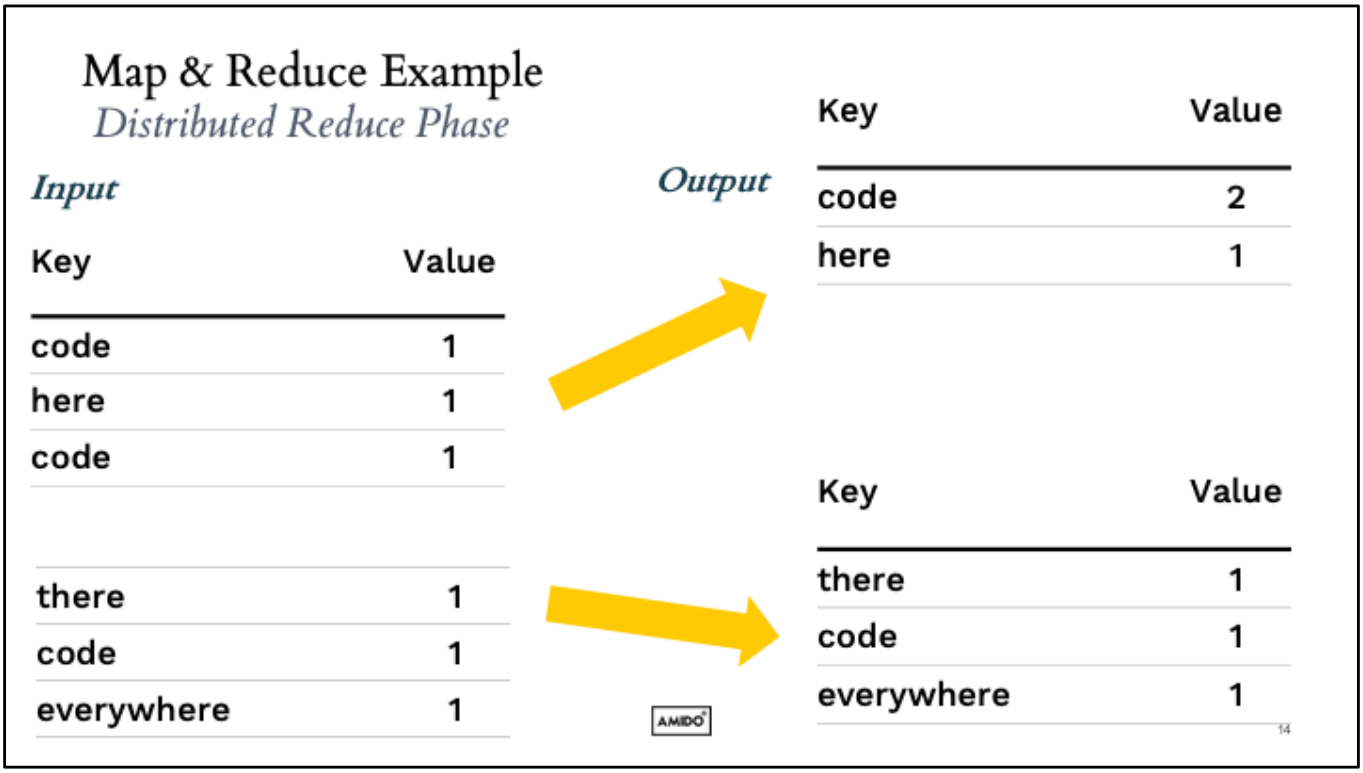
So lets go through a complete MR worked example

- We want to count the number of each word in our document

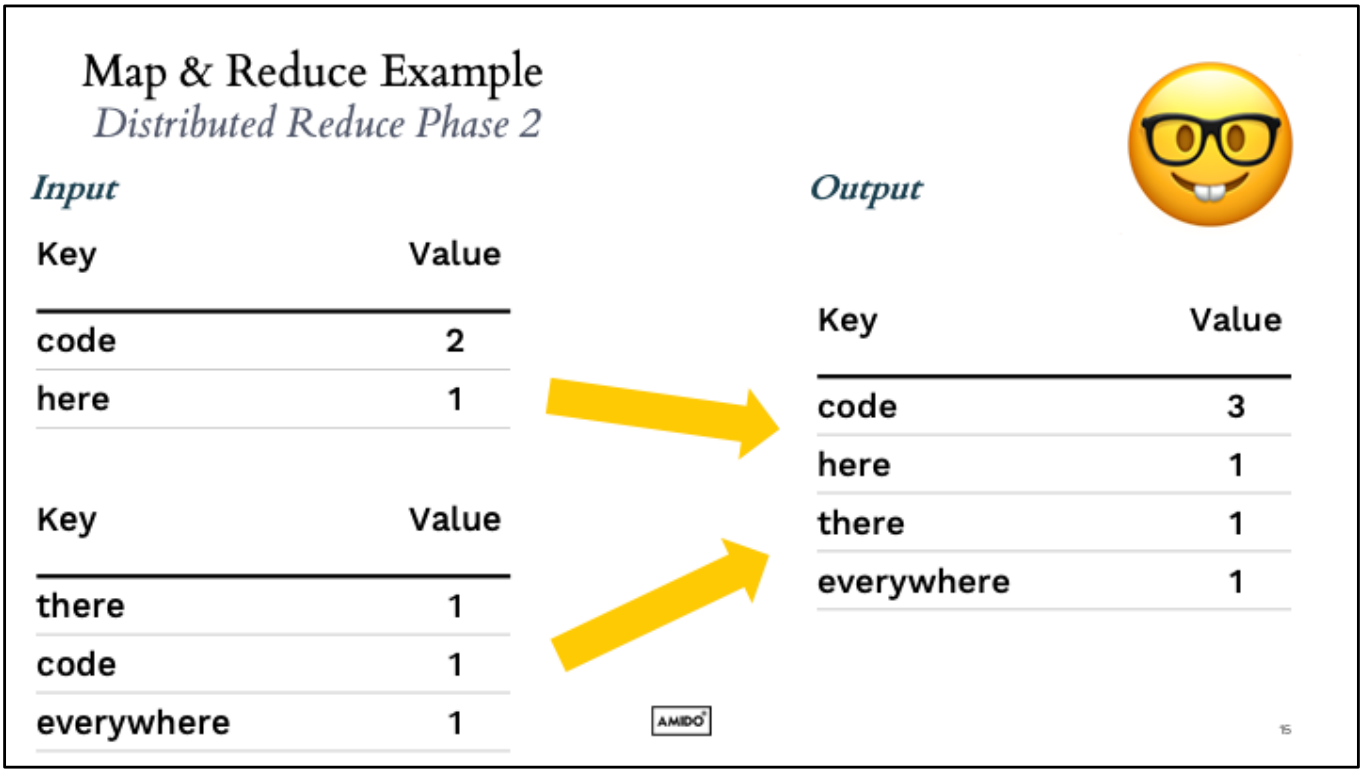
- During the map phase we *map* the input to zero or more outputs.
  - In this case for every word we are outputting the word as a key, and the value ‘1’



- For the reduce phase we are reducing a group of mapped values into something that makes more sense
- So for this reduce algo, we can simply group together KVP of the same key, and then simply sum the values
- For the output you can see out final result – that ASOS occurred 3 times in our original text, and that here, there & everywhere occurred just the once



- Now what if we are working in a distributed system?
- We could actually split up the previous output into two smaller groups, and apply the reduce algorithm to those separately.
- We've easily made the job smaller and it's possible to run it in parallel!
    - This could be theoretically scaled a further, smaller chunks, more in parallel



- If you do that, then there is another reduce (exactly the same algo) to run against the previous reduce outputs
  - That gives rise to the same output as before
  - But we ran it in a distributed sense

A More Concrete Example

Counting makes of vehicles in accident data

AMIDO

16

I found records of ¼ million accidents that occurred on UK roads in 2016, so not a huge amount of data but enough to demonstrate the approach

The data.gov.uk website has loads of good stuff on it – recommend having a look around.

- For our data
  One record for every accident, lots of data but we are just going to look at the Make of the vehicle
    - i.e. how many Fords, Vauxhalls etc had accidents

The raw data

```
2016010000019,2016,…,1596,1,10,3,1,BMW,116 I SE
2016010000019,2016,…,1596,1,5,2,1,FORD,FOCUS ZETEC 125
2016010000020,2016,…,2499,2,2,2,1,LONDON TAXIS INT.,TX4 ELEGANCE AUTO
```

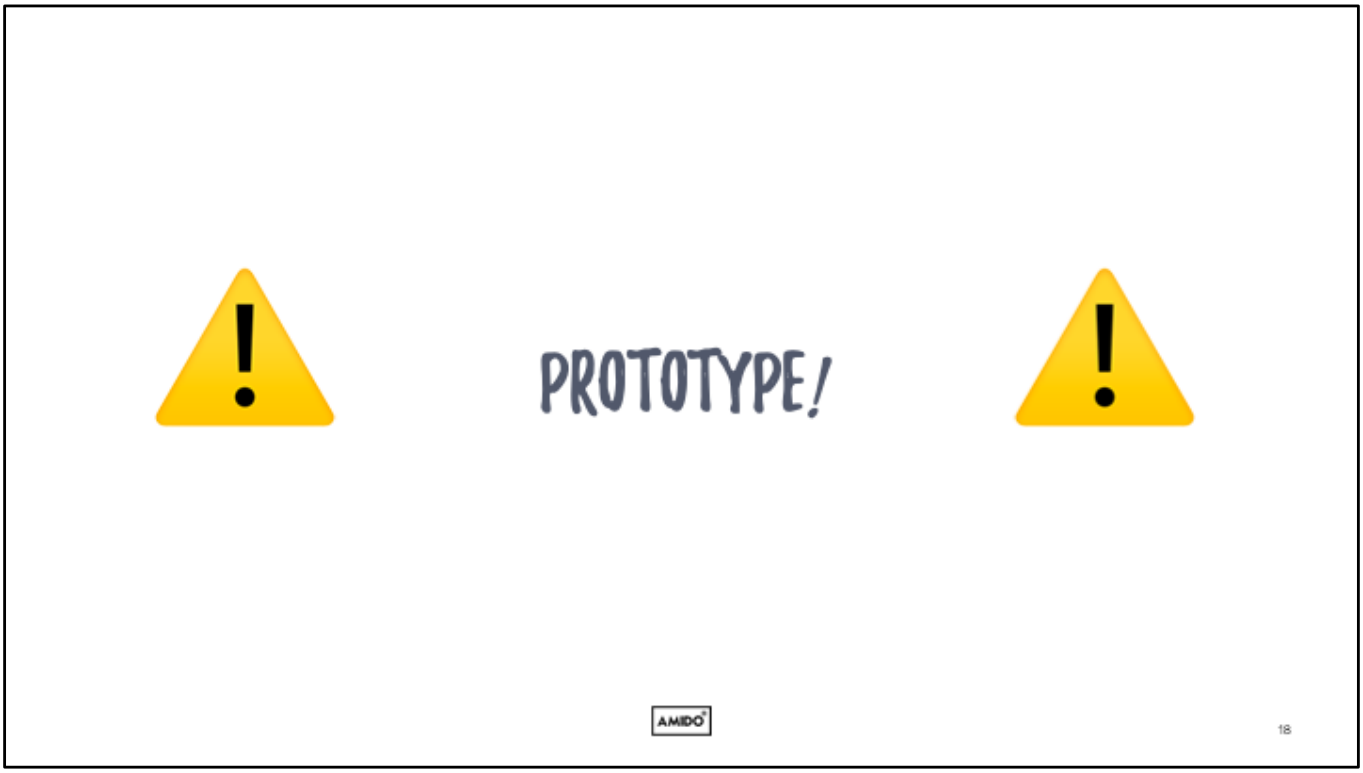
AMIDO

17

This is snip of the data

A big chunk of the middle chopped out to make it easier to view

Basically it's a CSV file, so we can reliably split each line on commas and extract the make of the vehicle that had the accident



All the code I'm going to show you is prototype code

Why do you produce a prototype?
- To gather evidence on an approach at low cost
- Such that you can then decide whether to invest more in the future

What is prototype not?
- It isn't necessarily maintainable or elgant
- It took me around 4 weeks of evening work!

It's not of production quality, it's the result of an experience to see if this is possible
- It probably has lots of bugs
- The demo later will probably fail

It's not perfect – but I've learnt a lot and I'll talk more about possible improvements later on

Example Map & Reduce functions

Mapper

```
class MakeAccidentCountMapper : IMapperFunc
{
    public KeyValuePairCollection Map(string line)
    {
        var strings = line.Split(',');
        var count = new CountKvp
        {
            Key = strings[22], // make is 23rd column
            Value = 1
        };
        return new KeyValuePairCollection {count};
    }
}
```

AMIDO

19

- I'm going to show a demo in a little bit – remembering that it's counting accidents in 2016 by car manufacturer
- This it the mapper that I implemented
- Walk through it

Example Map & Reduce functions

Reducer

```
class MakeAccidentCountReducer : IReducerFunc
{
    public KeyValuePairCollection Reduce(KeyValuePairCollection inputKeyValuePairs)
    {
        var reducedCounts = new Dictionary<string, int>();
        inputKeyValuePairs.ForEach(x =>
        {
            var count = (CountKvp)x;
            if (!reducedCounts.ContainsKey(count.Key))
                reducedCounts.Add(count.Key, 0);
            reducedCounts[count.Key] = reducedCounts[count.Key] + count.Value;
        });

        var keyValuePairs = new KeyValuePairCollection();

        reducedCounts.ToList().ForEach(x => keyValuePairs.Add(new CountKvp(x.Key,
          x.Value)));

        return keyValuePairs;
    }
}
```

AMIDO

20

- Respective Reducer function
- A little more complex
- Walk through it

Example Map & Reduce functions

Reducer

```
class MakeAccidentCountReducer : IReducerFunc
{
    public KeyValuePairCollection Reduce(KeyValuePairCollection
       inputKeyValuePairs)
    {
        var reducedCounts = new Dictionary<string, int>();
        inputKeyValuePairs.ForEach(x =>
        {
            var count = (CountKvp)x;
            if (!reducedCounts.ContainsKey(count.Key))
                reducedCounts.Add(count.Key, 0);
            reducedCounts[count.Key] = reducedCounts[count.Key]
              + count.Value;
        });
```

AMIDO

21

...

- Respective Reducer function
- A little more complex
- Walk through it

Example Map & Reduce functions

Reducer

```
        var keyValuePairs = new KeyValuePairCollection();

        reducedCounts.ToList().ForEach(x =>
    keyValuePairs.Add(new CountKvp(x.Key, x.Value)));

        return keyValuePairs;
    }
}
```

22

- Respective Reducer function
- A little more complex
- Walk through it

Example Map & Reduce functions

Final Reducer

```
class MakeAccidentCountFinalReduce : IFinalReduceFunc
{
    public IReadOnlyCollection<string>
FinalReduce(IKeyValuePair keyValuePair)
    {
        var countKvp = (CountKvp) keyValuePair;
        return new[] {$"{countKvp.Key},{countKvp.Value}"};
    }
}
```

AMIDO

23

- This is the final step. It's optional

- Usually it's a transform of data from format that is optimised for the reduce stage into format that is optimised for the consumer
- In this case, I'm taking each KVP and outputting a line in simple CSV format

Example Output

```
MERCEDES,9415
VAUXHALL,24541
NULL,47725
SAAB,533
PEUGEOT,10287
FORD,28131
SEAT,2984
SUZUKI,3205
NISSAN,6959
YAMAHA,2409
Yamaha,1043
Daf Trucks,116
...
```

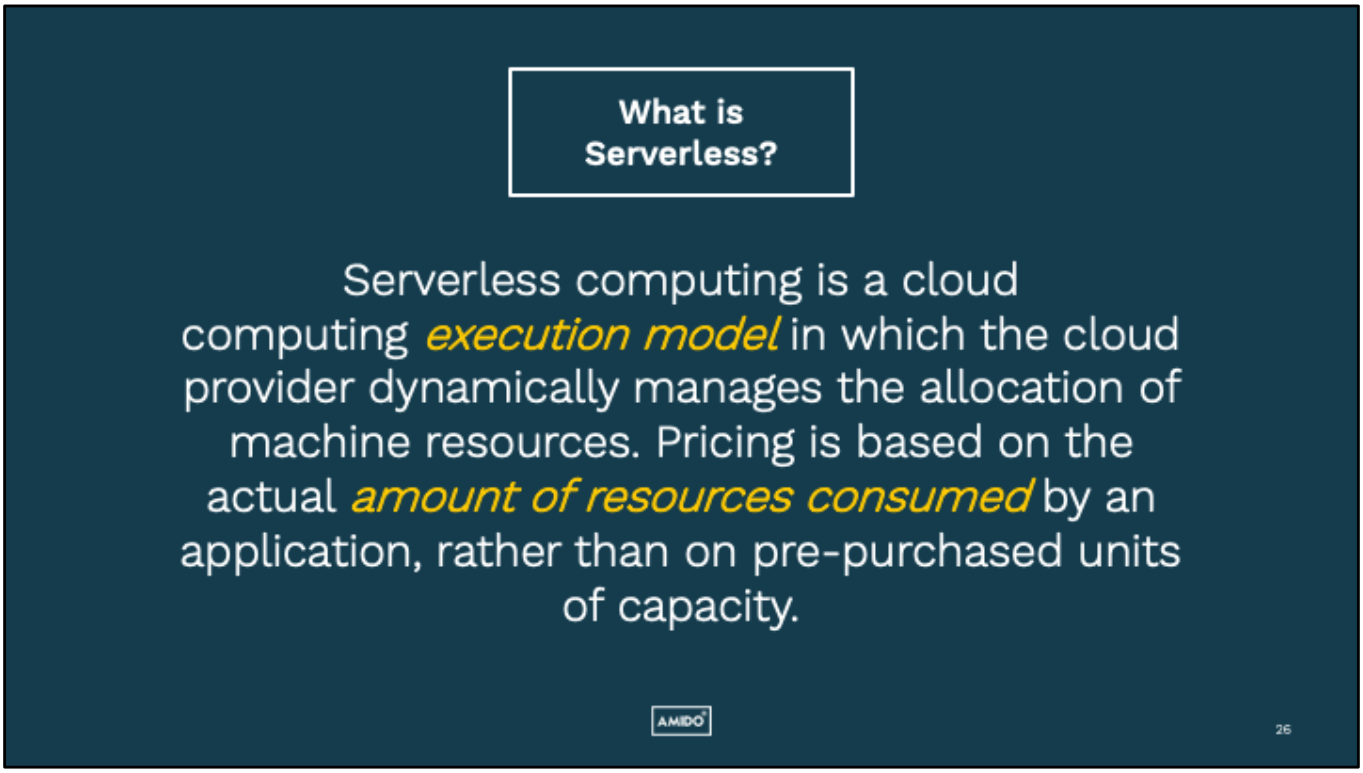
AMIDO

24

- This is the final step. It's optional

- Usually it's a transform of data from format that is optimised for the reduce stage into format that is optimised for the consumer
- In this case, I'm taking each KVP and outputting a line in simple CSV format

- computing as opposed to framework



Serverless is an execution model

This bit is important

Refer back to early separation of programming model and execution model when talking about MapReduce

There is nothing about this execution model that means you should use Lambda, GCF, Azure Functions etc, or

All you should need to think about is executing your task – compute, data storage, etc – and you should not have to concern yourself with the underlying infrastructure

What other utility services are we aware of?

Utility service -- services made available to the customer as needed, and charges for specific usage, rather than a flat rate

- you do not pay for excess capacity

Utility computing is the packaging of system resources, such as computation, storage and services, as a metered service.

- with an EC2 instance, you pay for excess capacity to sit there doing nothing most of the time

I like to think of the goal of utility computing as being comparable to the goals of household ”utilities”

- we don’t care how the gas / elec / water gets there – we don’t care about the underlying infrastructure
- we want it to be there as soon as we turn on the tap
- more when we open the tap, less when we close it
- SLAs, Ofgem Quality of Service Guaranteed Standards, minimum downtime

- interestingly, your home broadband probably doesn’t fit into this paradigm

- You tend to pay for / provision some bandwidth that you may or may not use
- We used to have a metered service – dial-up – but that wasn't popular
- So perhaps not all true utility services are good

Serverless Pros & Cons

- You don't have to worry about instances / scaling (in theory!)
- You (literally) only pay for what you use
- Good fit with microservices

- You may need to be aware of instances...
- Some limitations on running time, memory allocation etc
- **You have no control over performance / there is no SLA**
  - In terms of both execution speed & latency

AMIDO

28

- No SLA
  - depending on your scenario, you may not or may not be able to use serverless in your particular use case – it may be that your volumes / criticality is too high to get away without an SLA

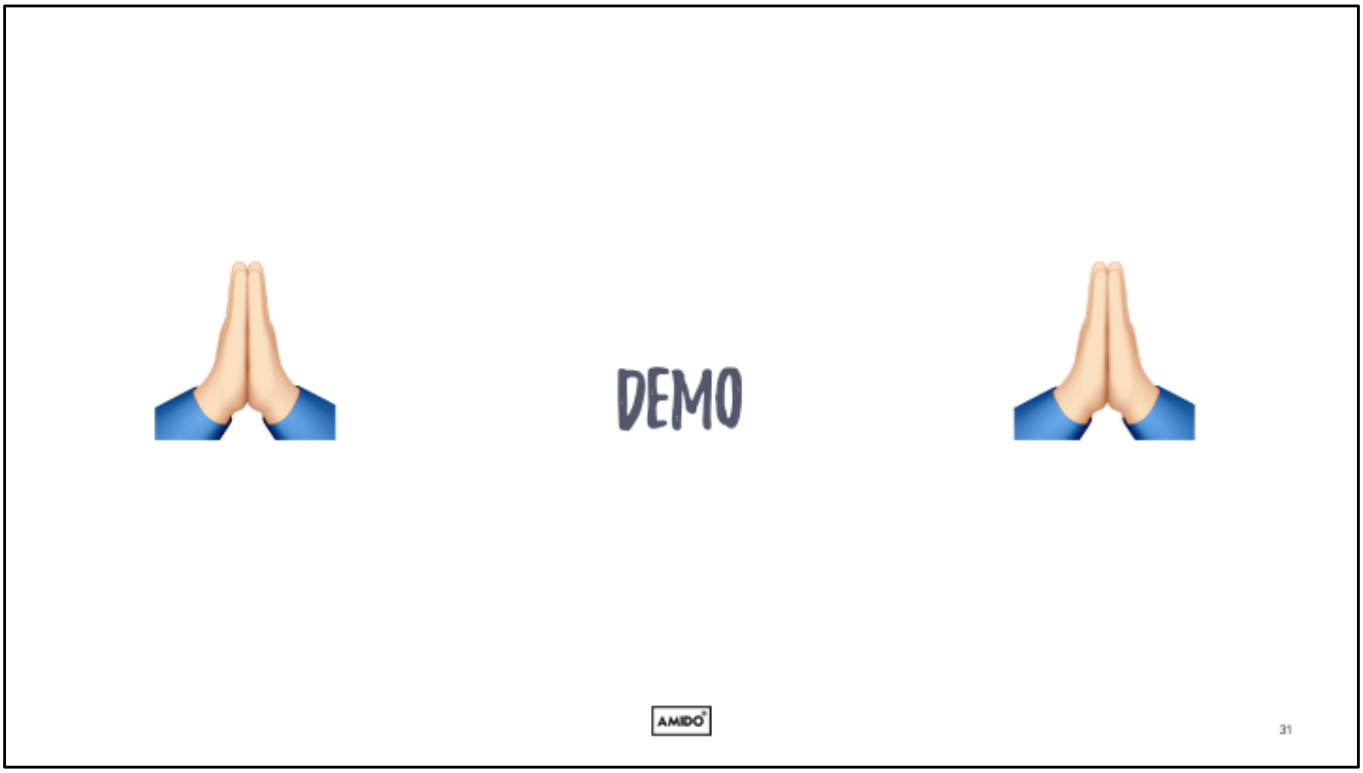
- computing as opposed to framework

The nature of Map Reduce is that you break a big problem into little bits that can be worked on independently & in parallel

The nature of serverless is that you can compute small pieces of independent work in parallel

If you take one thing away from this talk it's this: The Serverless Execution Model and MapReduce Programming model are an excellent fit for one another.

This part
1) Walk through some real data, real code, real output
2) Look at architecture & design highlights / interesting stuff in the prototype
3) Demo – answer an important question with real data



(Please work!)

- I promised that tonight I would answer real questions with real data.

- Jeremy Clarkson has frequently made a derogatory assertion about Audi drivers

- I intend today to use data to irrefutably prove one way or another – does owning an Audi make you a bad driver?

- Some caveats – I may have slightly simplified this problem…
    - Just looking at 2016. use accident data that I previously mentioned
        - Filter out those accidents where the vehicle is older than 1 year old
    - Also take new car registration data from SMMT (Society of Motor Manufacturers and Traders) in the same year
    - Work out how many new registrations were made for each accident, by manufacturer
    - ➔ if you buy a new car, which brand is most likely to make you crash it
    - Just a bit of fun…

Prep

1. Run purgeAwsResources.sh

This demo will probably fail!

1. Explain what we want to try and show
2. Explain data sources – SMMT (Society of Motor Manufacturers and Traders -- for manufacturing numbers) & data.gov (for accident numbers)
3. Show Map, Reduce & Final Reduce functions
4. Show monitoring page and that everything is zero
5. Show S3 bucket, and that it is empty apart from the two raw data files
6. Show SQS queues, and that they are all empty
7. Show contents of enqueueInitialMessage.sh and that it simple puts two messages on a queue, talk through parms
8. Run ./enqueueInitialMessage.sh
9. Show that raw message queue now has one item in it
10. Show contents of startProcess.sh and that it kicks off a lambda process, talk through params
11. Run ./startProcess.sh
12. Show that 202 returned (202 == Accepted)
13. Refresh monitoring page, refresh SQS monitor page to show progress
14. Kill time – perhaps talk through what is happening
15. Refresh monitoring page, refresh SQS monitor page to show process finished
16. Download finalised file
    1. Open In sublime?!

Prototype code is open source…

Fork me on GitHub

https://www.github.com/cjrpriest/ServerlessMapReduce.Net

AMIDO

33

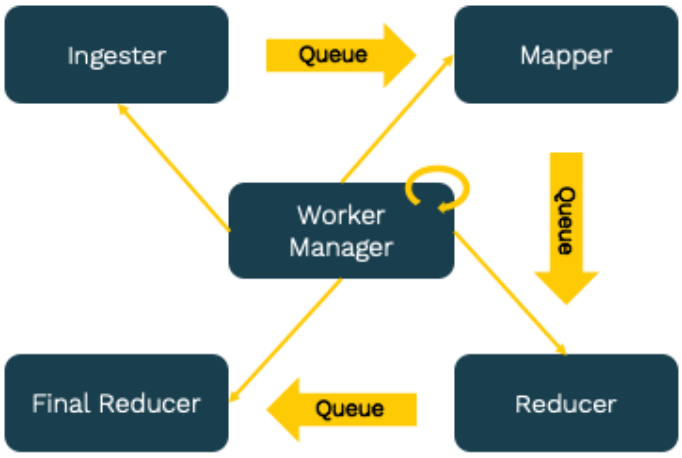
- As of today the code used in today's prototype is open source

- Please take a look

- It would be great to have people take a look at this code, let me know what you think what to you like, what do you hate, fork it, submit a PR, raise an issue

- computing as opposed to framework

Prototype Architecture Overview



AMIDO

35

- First a quick look at the high level architecture of the prototype
- Mention S3 storage here as well

Use of Mediator Pattern

Why?

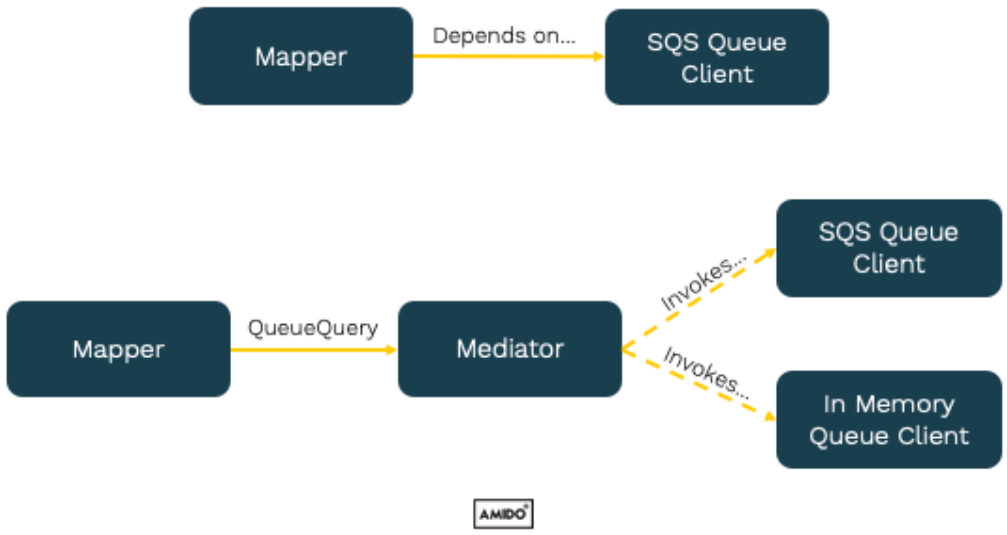
AMIDO

36

- Want to talk about the use of the Mediator in this example
- Why?
  - By reducing coupling between components, have them communicate through a mediator, then I have flexibility over **how** & **what** executes behind an interface
  - The mediator decides what to invoke, and how
  - Component A truly knows nothing about Component B, and vice versa

Use of Mediator Pattern

How does it help?



- So how does this help in this particular example?
- There are a couple of key scenarios, this is the first
- This is the first – rather than have your classes dependant on an interface, and compiler-based invocation, switch to state based invocation – means that components are even further decoupled.
- The component that is invoked is selected by infrastructure configuration
- Could also mention FileSystem v S3 storage
- What is the benefit here?
    - Find that you end up refactoring tests far less when you change a components interface.
    - Can leverage some standard tooling, such as for things like logging

Use of Mediator Pattern

How does it help?

AMIDO

38

- The other (much more interesting) scenario is to do **where** you invoke a component
- Imagine this simple scenario where the Worker Manager / Orchestrator invokes an instance of the Mapper
- At the moment the Mapper is likely executed locally to the Worker Manager

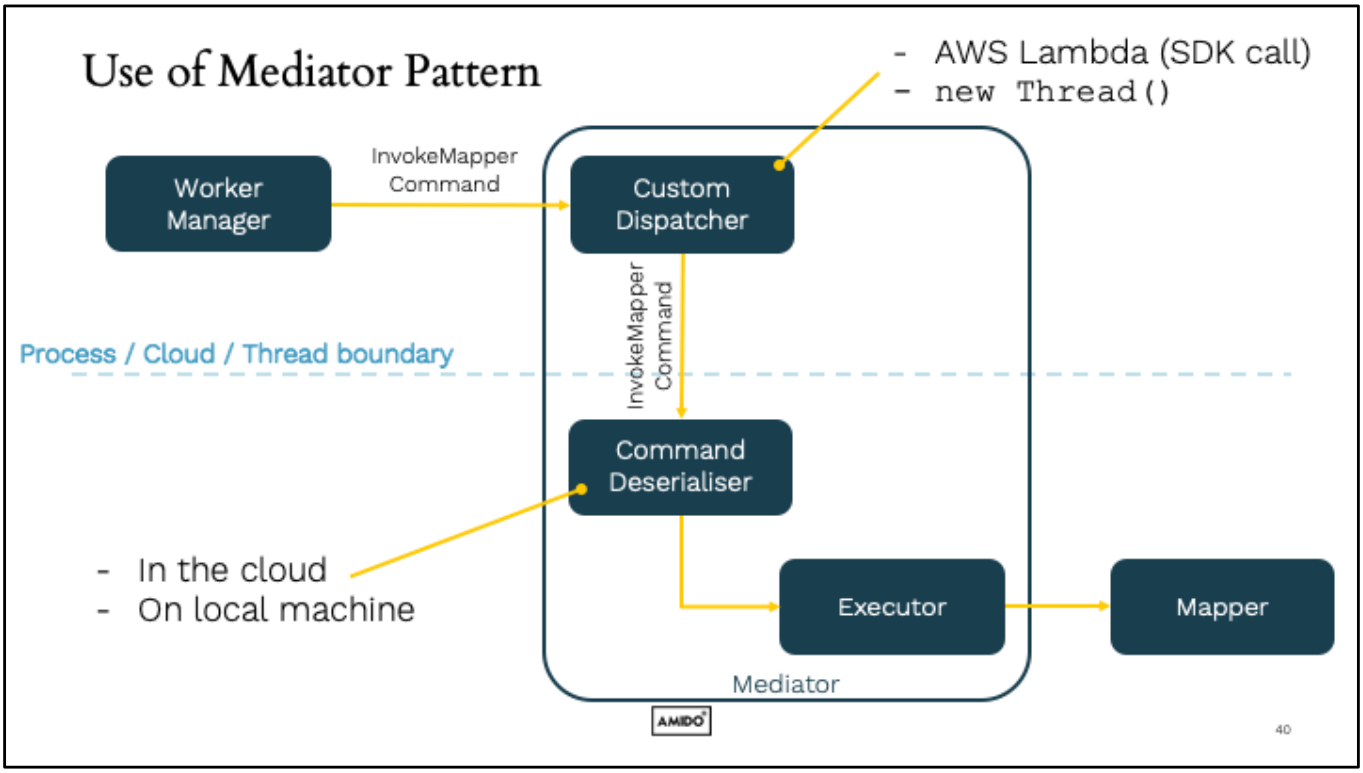
Use of Mediator Pattern

How does it help?

AMIDO

39

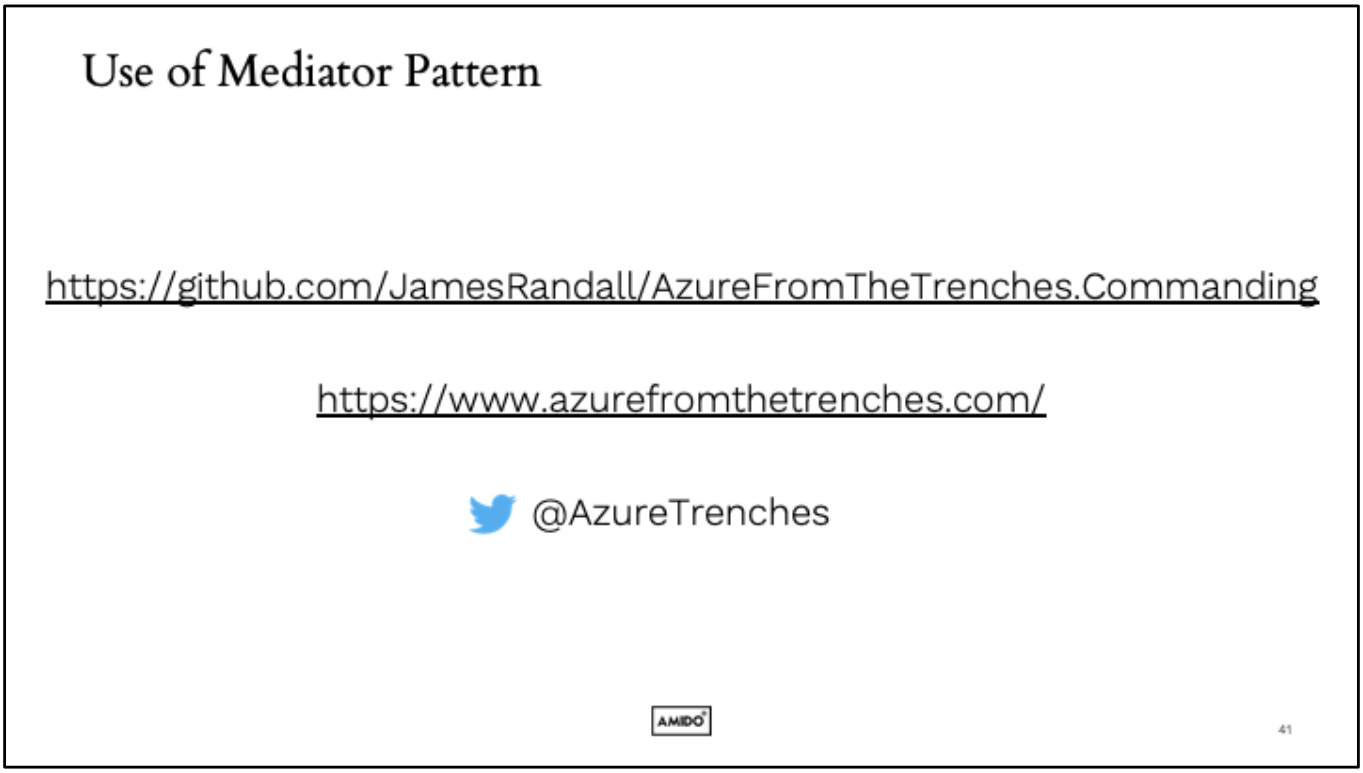
- If you separate the Worker Manager & the Mapper with a Mediator you have the possibility to entirely change where & how invoke the Mapper



- Imagine if the Mediator was able to take the work that you want to do and abstract away exactly where & how it is executed

- Worker Manager knows nothing about how the mapper will be invoked
    - Dispatch is totally separated from execution
    - Could be locally (new Thread()), or on AWS lambda, Azure Function, GCP… or something entirely different
    - All in configuration
    - Same command is passed regardless

- What is the benefit here?
    - Allows me to test code running locally and have a high degree of confidence that it'll work when the dispatcher and executer are changed
    - Change the environment upon which the code runs through a simple & abstract configuration change



- In my prototype I used the excellent Commanding framework by James Randall

- Lots of flexibility in terms of how you dispatch & execute your command and queries
- Ability to standardise code executed before & after every query, such as recording performance metrics, logging etc
- Loads of great blog posts / doco, especially if you are interesting in this kind of library & pattern to break up your monolith

- I recommend you check it out!

- Also checkout James' Serverless Scaling Faceoff, Functions vs Lambda, blog posts – made the top of Hacker news

Critique…

Good

- Writing MapReduce functions in .NET!
  - The only other option I'm aware of is Raven DBs MapReduce indexes feature
- A way to run MapReduce .NET algos in a way where you literally pay for what you use
  - great for one-off / infrequent workloads
- The genesis of a potentially useful library is here

AMIDO

Bad

- Performance – compute & costs
  - particularly for small jobs
- This implementation is not truly event driven
- This implementation doesn't scale particularly well(!)

AMIDO

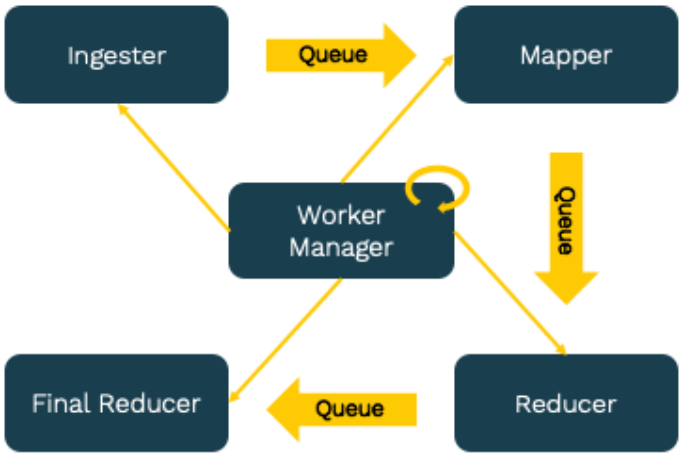
44

- So what is bad / what can be improved about this prototype?
- Performance computer & costs
  - particularly small jobs isn't that great. Especially compare to running on laptop where sample will run in around 6s
    - Functions themselves perform relatively slowly in comparison
    - Then there is overhead of reading & writing to S3 v reading & writing to memory…
- The best serverless implementations are entirely event driven
  - This implementation relies the regular triggering of an orchestrating / co-ordinating process
  - Would be better if the entire process was event driven, with each function raising a new event when work is complete, which in turn triggers another even that determines if more work is to be done
- Scaling
  - This is due to a bug / limitation.
  - Each function writes its status to a central location, which is the read by the Manager / co-ordinator

    - More functions that run --> more status objects to read by the worker manager
        - There is a complex solution to the problem of storing state – I write a library around S3 for storing arbitrary objects – more on that later

- Going to talk more about how to tackle these now

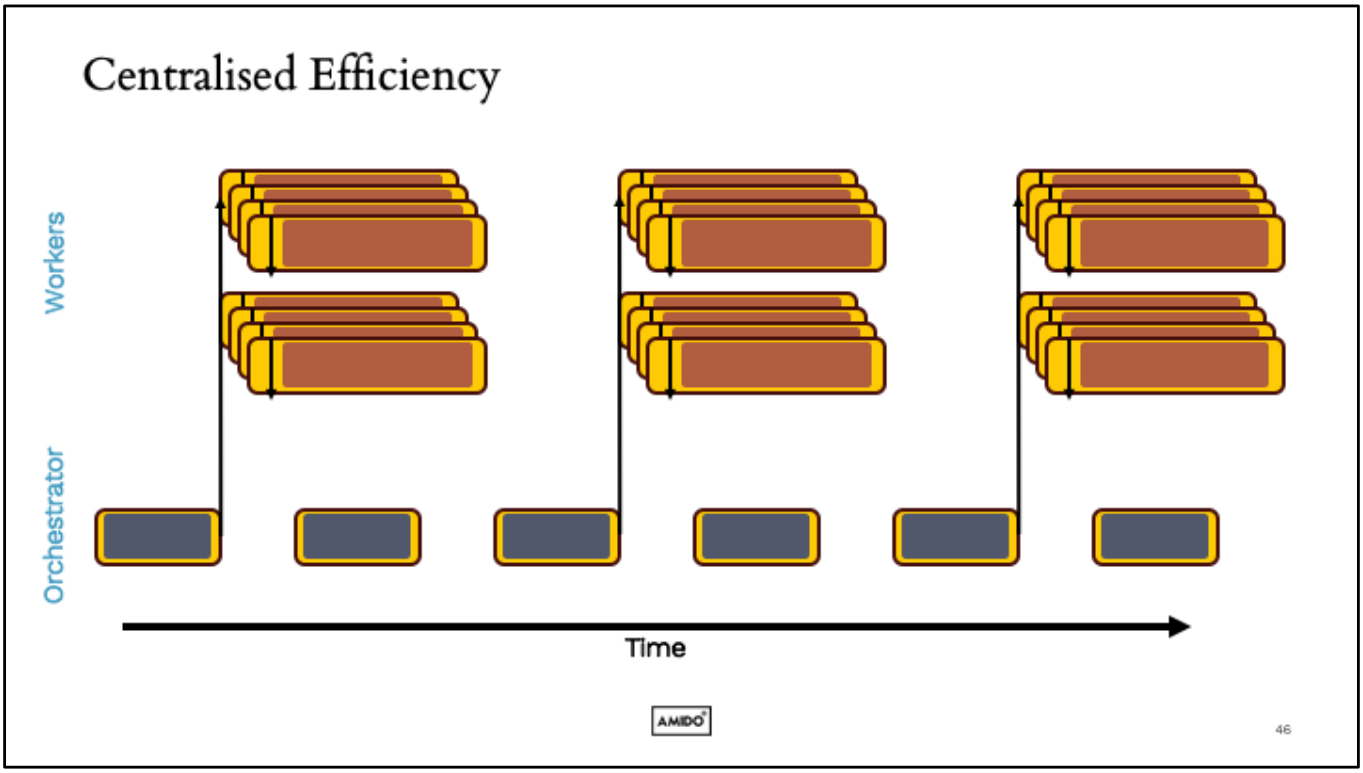
Prototype Architecture Overview (Centralised)



AMIDO

45

- First a quick look at the high level architecture of the prototype



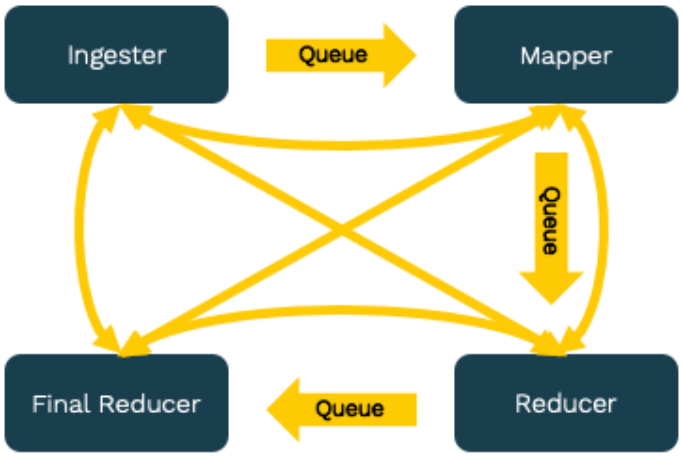
- Want to get into more detail on some of the things that could be improved
- To re-cap – this is what we currently have with the centralised approach

- The orchestrator looks at what is going on, and then decides how much work to kick off

- If there is already work being done, then do not kick anything off

- Not very effcient -- lumpy workload, and orchestrator that is almost always running and not doing very much

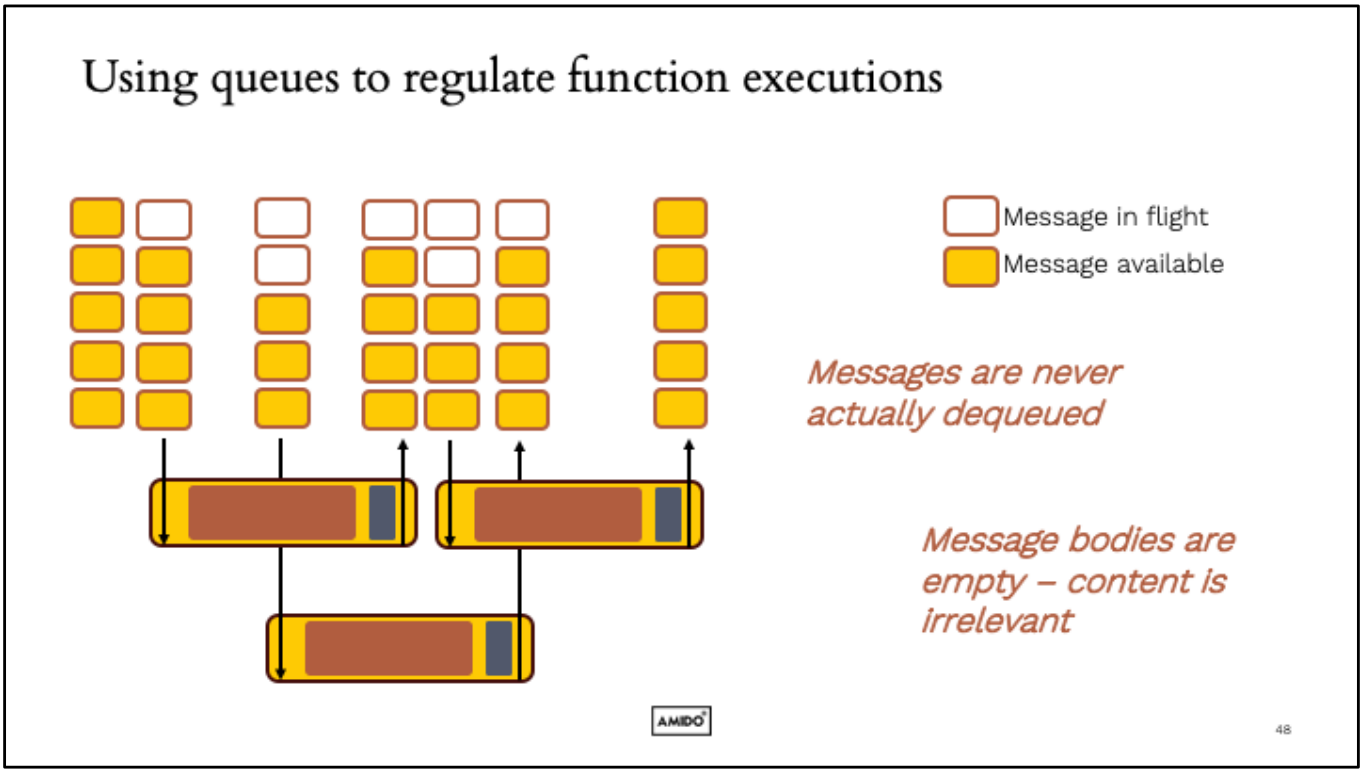
Prototype Architecture Overview (De-centralised)



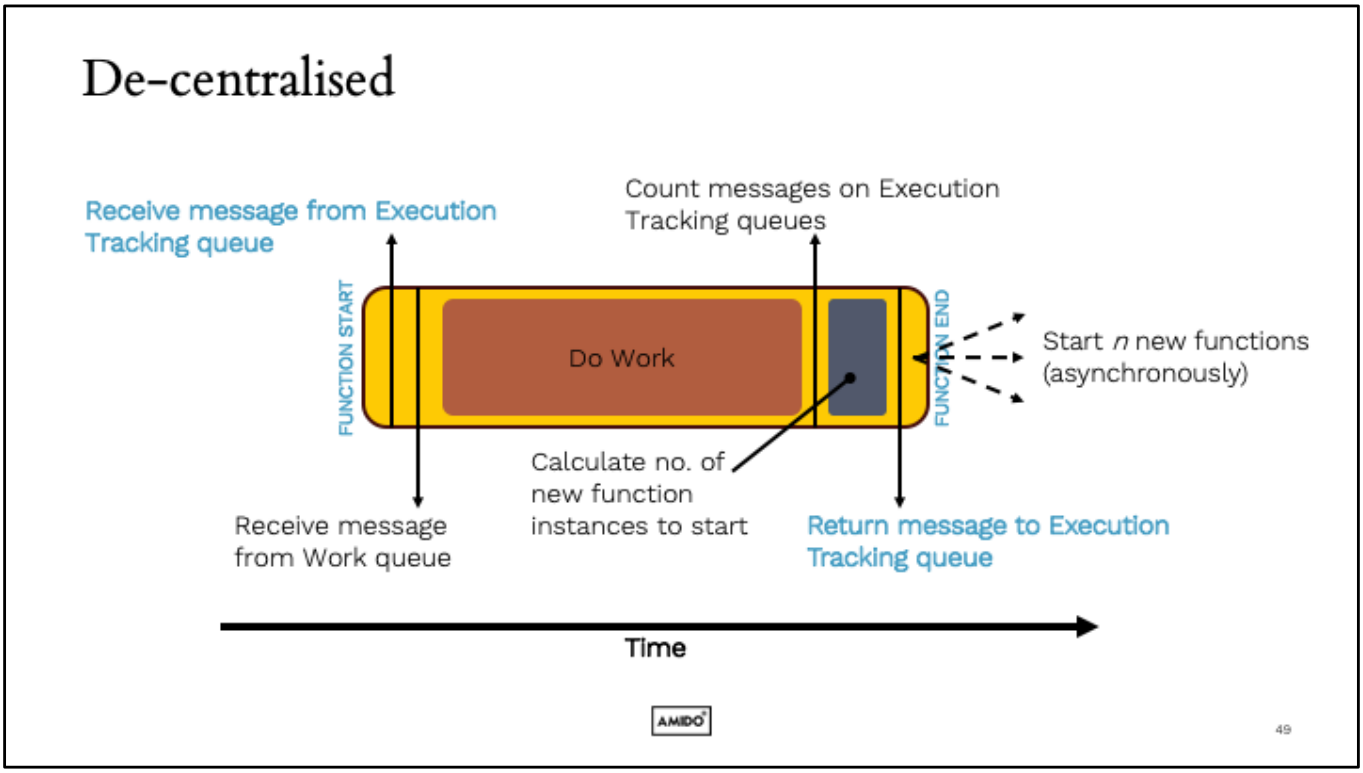
AMIDO

47

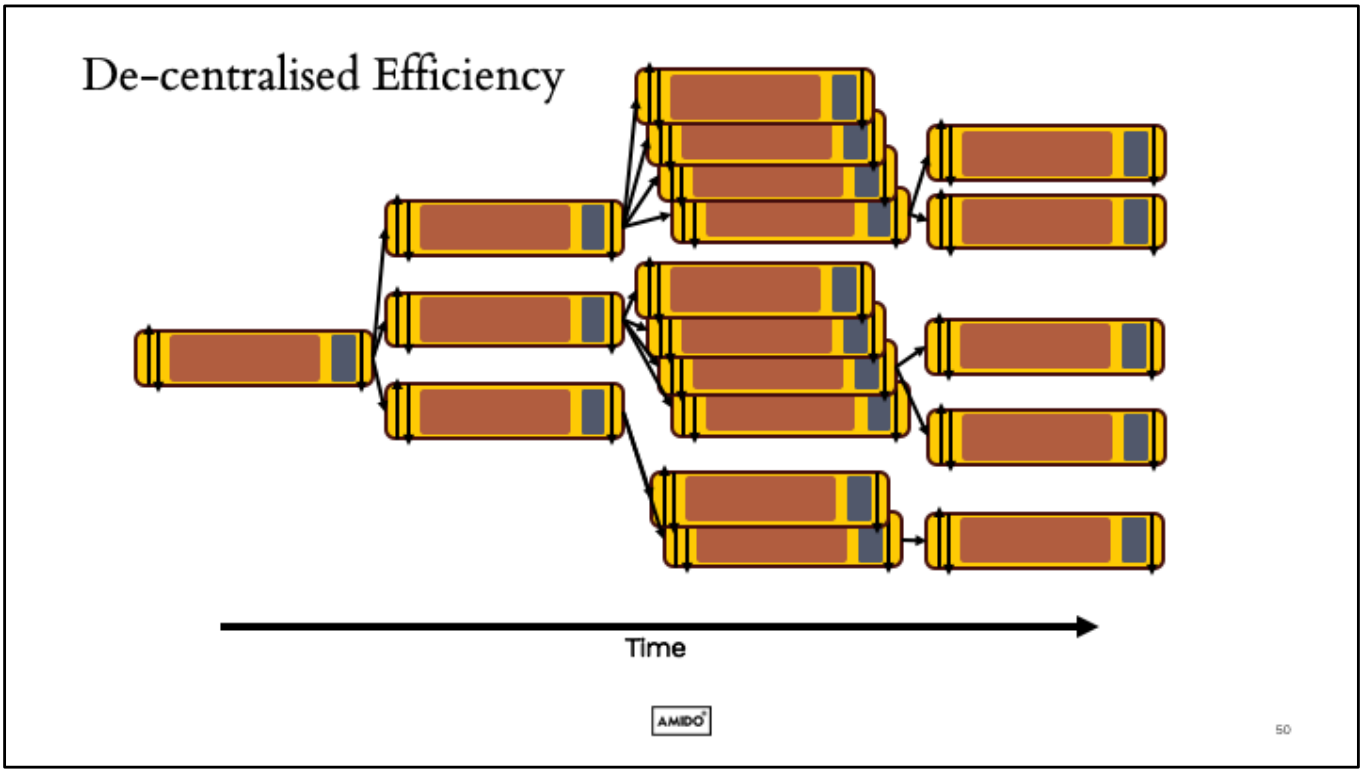
- First a quick look at the high level architecture of the prototype



- Why do this?
  - control / visibility over the number of running functions
- Think of it as a distributed, eventually consistent synchronised counter
- You can determine the number of currently executing functions by getting the number of currently inflight messages
- If function times out / throws exception / doesn't complete, the inflight message is returned to the queue (and made available) automatically by SQS



- Here is a closer look at how each function might look in a decentralised scnearios
- Walk through from left to right



- I think you'll end up with a execution pattern like this
- Much more efficient at utilising compute resources, and no orchestrator

Next Steps

- Feedback from you, today!
- Make it better. Key areas to improve:
  - Performance / scaling – there are several areas that could be improved to make better use of the infrastructure
  - Make it super easy to use – ideally the developer implements two interfaces, supplies some input data, that's it
- I'm excited to see where this will go, follow me on Twitter for updates!

51

- Feedback – I'm in the bar after the talks today, please come & tell me what you think


Thank
You
@cjrpriest