Cheap and cheerful massively parallel batch R processing on EC2

Louis J. M. Aslett (`aslett@stats.ox.ac.uk`)

Department of Statistics, University of Oxford

R user group Oxford
10 November 2016: St John's College

`www.louisaslett.com`

i-like.org.uk

Introduction

Introduction

Introduction

Introduction

Amazon Web Services

Cloud services — **buzz-word alert!**

By cloud, I mean an online service which allows users to create and destroy virtual servers remotely without having to worry about initial hardware and OS installation and where billing is in very small increments (e.g. hours).

There are several cloud providers, including:

- Amazon EC2
  - `http://aws.amazon.com/`
- Digital Ocean
  - `http://www.digitalocean.com/`
- Google Compute Engine
  - `http://cloud.google.com/`
- Rackspace Cloud Servers
  - `http://www.rackspace.co.uk/`
- Windows Azure VMs
  - `http://www.windowsazure.com/`

Why R in the cloud?

- Long analyses
- Collaboration
- Powerful AWS instances, with access to GPUs
- Online data sources
- Full environment with C/C++/Fortran, LaTeX+ Sweave, Git + Subversion, Stan etc ready-to-go.
- Access to the power of Linux without having to dedicate your own machine to running it.

Why Amazon today?

Today, AWS is the only the service which ticks all the following (subjectively) important boxes for scientific HPC, though this is a *fast* moving business:

- Repository for community development of images so users can boot ready-to-run machines with more than just bare operating system (bit like package system in R).

Why Amazon today?

Today, AWS is the only the service which ticks all the following (subjectively) important boxes for scientific HPC, though this is a *fast* moving business:

- Repository for community development of images so users can boot ready-to-run machines with more than just bare operating system (bit like package system in R).
- A permanent storage medium for each machine which can persist independently of the running state of the server.

Why Amazon today?

Today, AWS is the only the service which ticks all the following (subjectively) important boxes for scientific HPC, though this is a *fast* moving business:

- Repository for community development of images so users can boot ready-to-run machines with more than just bare operating system (bit like package system in R).
- A permanent storage medium for each machine which can persist independently of the running state of the server.
- Billing which suspends while the server is stopped, but where the above persistent storage remains alive.

Why Amazon today?

Today, AWS is the only the service which ticks all the following (subjectively) important boxes for scientific HPC, though this is a *fast* moving business:

- Repository for community development of images so users can boot ready-to-run machines with more than just bare operating system (bit like package system in R).
- A permanent storage medium for each machine which can persist independently of the running state of the server.
- Billing which suspends while the server is stopped, but where the above persistent storage remains alive.
- A free tier of instances to try it without cost.

Why Amazon today?

Today, AWS is the only the service which ticks all the following (subjectively) important boxes for scientific HPC, though this is a *fast* moving business:

- Repository for community development of images so users can boot ready-to-run machines with more than just bare operating system (bit like package system in R).
- A permanent storage medium for each machine which can persist independently of the running state of the server.
- Billing which suspends while the server is stopped, but where the above persistent storage remains alive.
- A free tier of instances to try it without cost.
- Everything from micro instances to the current state of the art in HPC, including machines with up to **16** nVidia GPUs for CUDA support.

Why Amazon today?

Today, AWS is the only the service which ticks all the following (subjectively) important boxes for scientific HPC, though this is a *fast* moving business:

- Repository for community development of images so users can boot ready-to-run machines with more than just bare operating system (bit like package system in R).
- A permanent storage medium for each machine which can persist independently of the running state of the server.
- Billing which suspends while the server is stopped, but where the above persistent storage remains alive.
- A free tier of instances to try it without cost.
- Everything from micro instances to the current state of the art in HPC, including machines with up to **16** nVidia GPUs for CUDA support.
- A 'stock-market' for unused compute capacity, enabling heavily discounted compute jobs.

Some cons to AWS

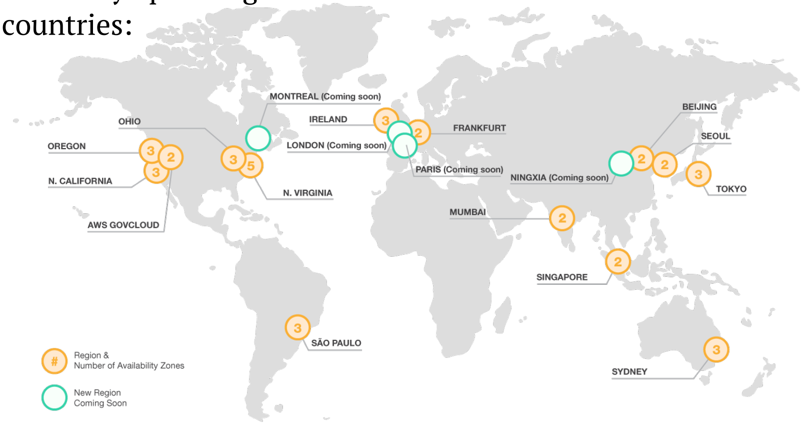
There are some cons to AWS, though again these could be addressed in future:

- Billing is in full hour increments (Google Compute Engine is in minutes)
- There are no guarantees of availability: Amazon explicitly intend users to design for instance failure in the use cases. There have been high profile outages in the US East Coast data centre
- High barrier to learning

Amazon Web Services (AWS)

Amazon used to buy in huge server capacity to keep their website up just for the Christmas shopping spree ... rest of the year large parts of server farm sat mostly idle.

Launched 2006. By December 2014, $1,400,000$ servers. Currently operating in 72 data centres across 13 cities in 10 countries:



AWS services of interest today

- EC2 (Elastic Compute Cloud)
  - is the service which enables launching virtual servers
  - also includes ‘stock market’ for unused capacity
- S3 (Simple Storage Service)
  - for resiliant storage of data independently of instances
- SQS (Simple Queuing Service)
  - for atomic and resiliant message queues
  - ‘in-flight’ messages with timeout

Quick-look demo

The RStudio AMI

Why an RStudio AMI?

- AWS enables rapid launching of instances from a set of different operating systems.
- However, they're bare bones systems
  - it's up to you to setup up the system as you want it
  - not hard for the technically literate … but definitely *time consuming*
- Any AWS user can create AMIs of systems so that they can boot directly to a system they have previously setup
  - an AMI is like a bare metal image of a system which can be restored to any new instance.
- The AMIs provide a public share of the pre-setup R system I use and have integrated user feedback and requests.

See `http://www.louisaslett.com/RStudio_AMI/`

What's included? (+ quick-look demo)

- 10GB EBS image on SSD backed storage
- 64-bit HVM images for best performance
- Full LaTeX distribution, supporting RMarkdown and Sweave
- GSL and CURL
- Database support
    - ODBC
    - RMySQL pre compiled
- Git & Subversion support
- Probabilistic programming languages Stan and JAGS for MCMC sampling
- Arbitrary precision arithmetic and number theory libraries
    - GMP, MPFR, FLINT
- Optimised BLAS for accelerated matrix operations
- Dropbox integration
- Beta support of Julia & Python /w Jupyter web interface
- Shiny server built in

Cheap & Cheerful Batch Processing

Cheap & Cheerful Batch Processing

At present the AMIs are heavily focused on interactive work.

Goal is to expand, first to:

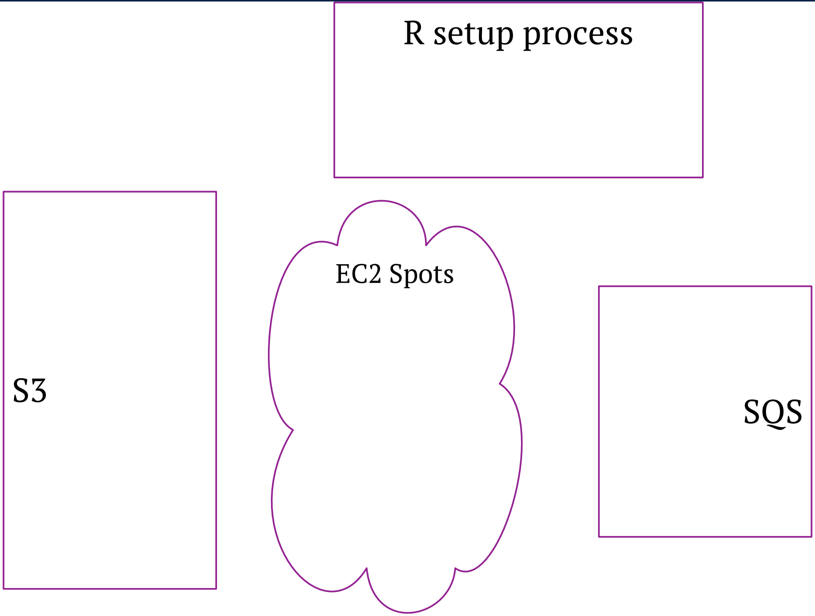
- Process batch jobs on EC2
- Fast startup environment building on RStudio AMI
- Able to use spot instances (cheap!)
  - resiliant to early termination
  - still works if bid results in zero instances running
  - scalable to theoretically arbitrary size
- Resilient to user code crashes (*never, surely!*)
- Easy to use (cheerful!)
- Full output logging capability

What this is not!

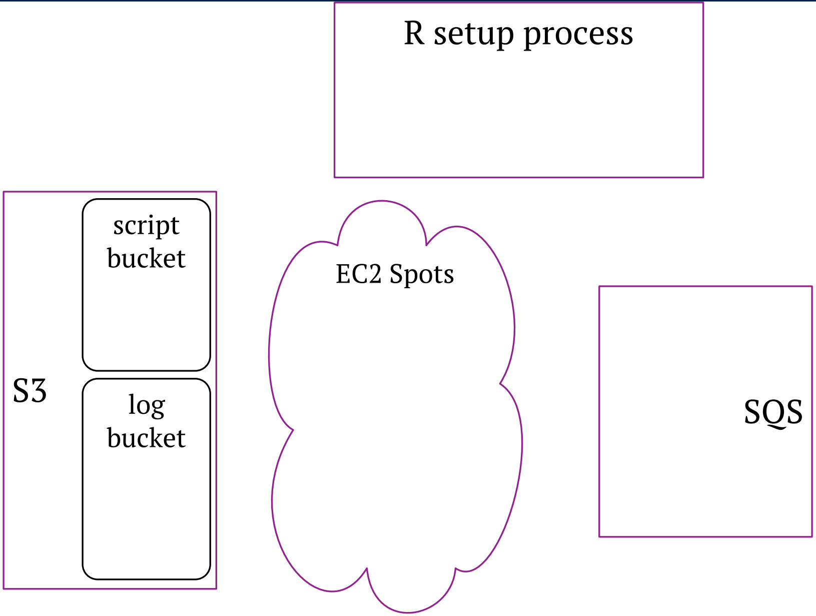
This is:

- **not** anything like Hadoop/Spark/Storm/EMR/…
  - they are great — use them when appropriate!
- **not** trying to tackle the communication based parallelism problem.
  - looking for embarrasingly parallel independent batch work
- **not** difficult to use!
  - the point is to be able to go from a local `for`-loop style batch construct to scaling over hundreds of very cheap CPUs in about 10 minutes.
  - if the final process involves more than a handful of trivial steps, each taking a minute or two I've failed.

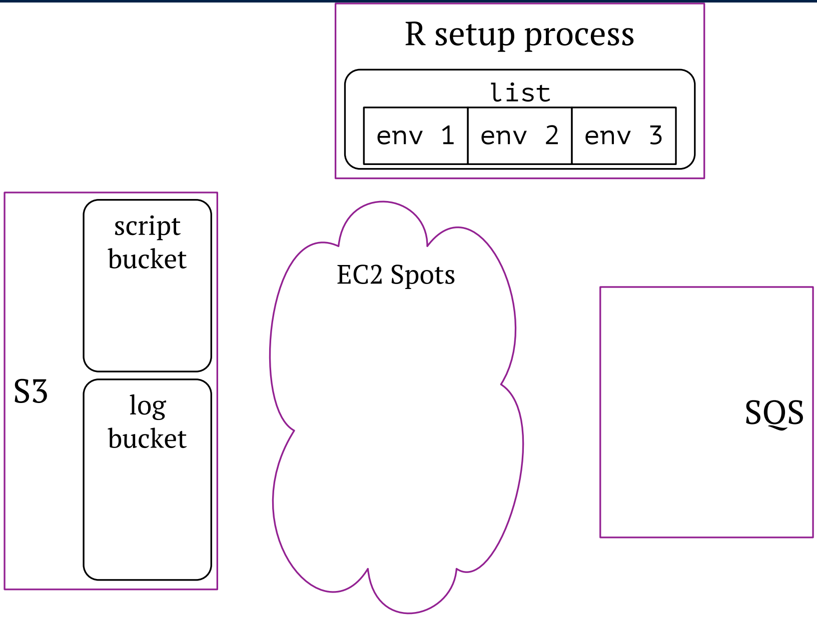
The approach (I)



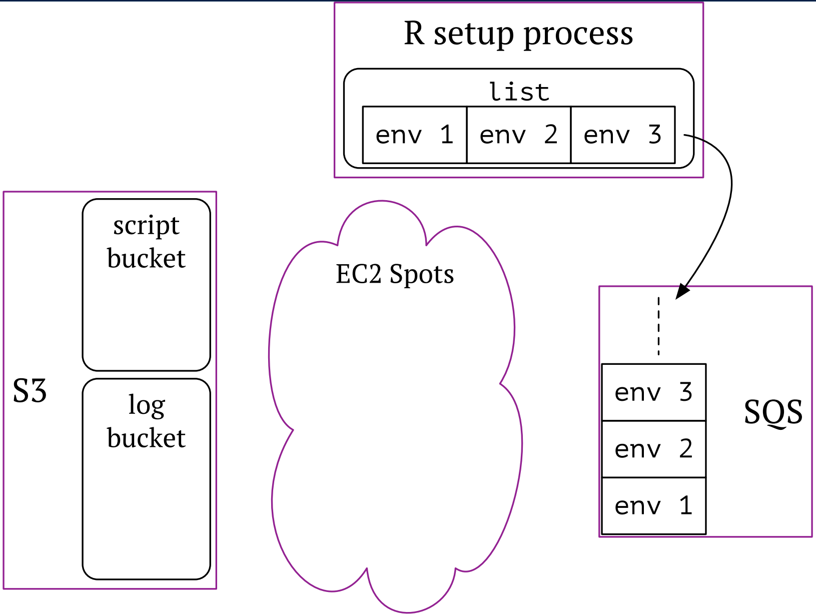
The approach (I)



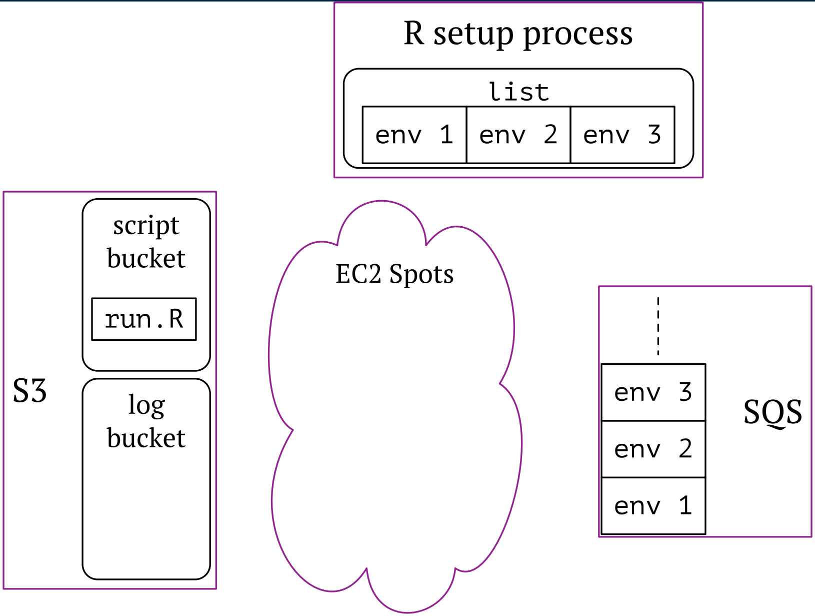
The approach (I)



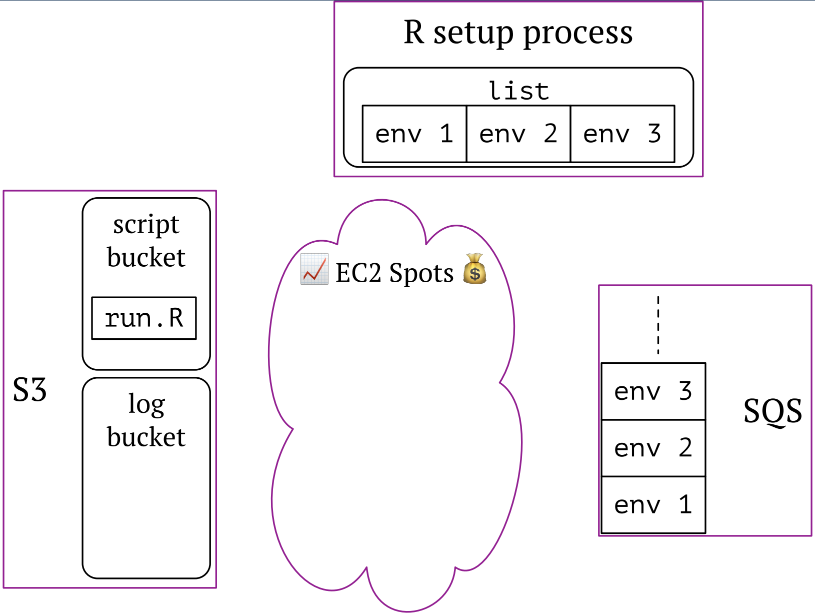
The approach (I)



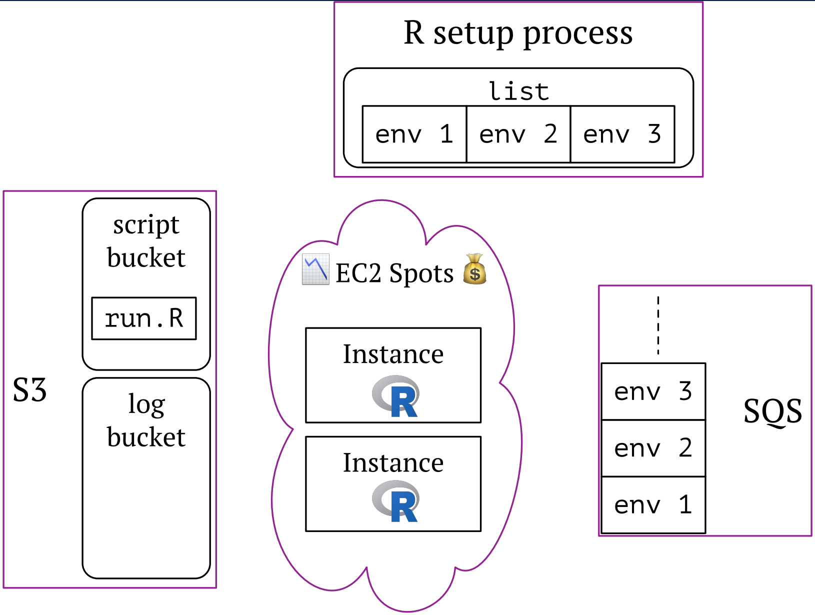
The approach (I)



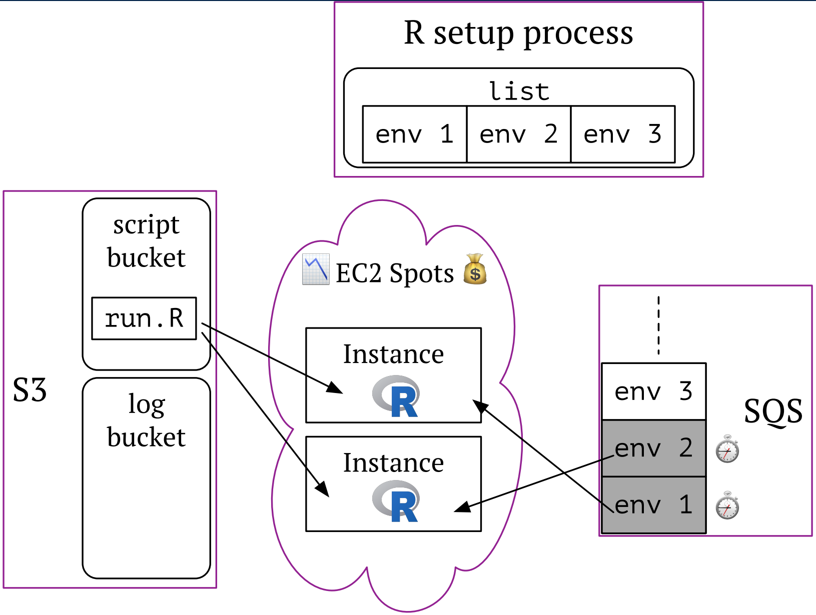
The approach (I)



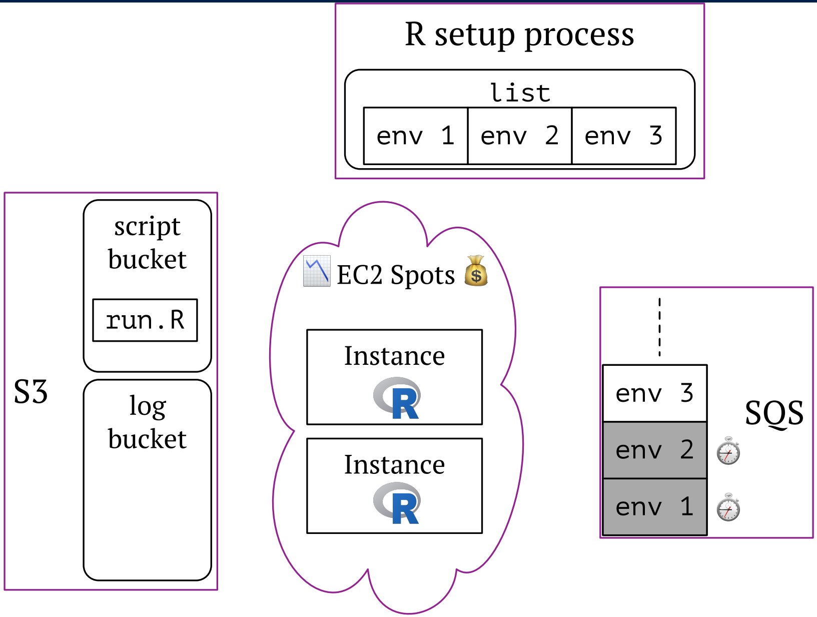
The approach (I)



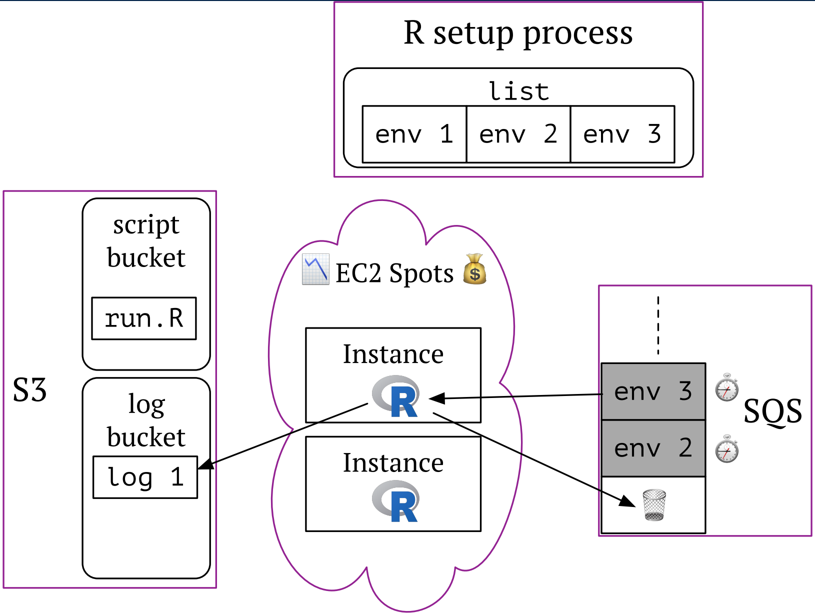
The approach (I)



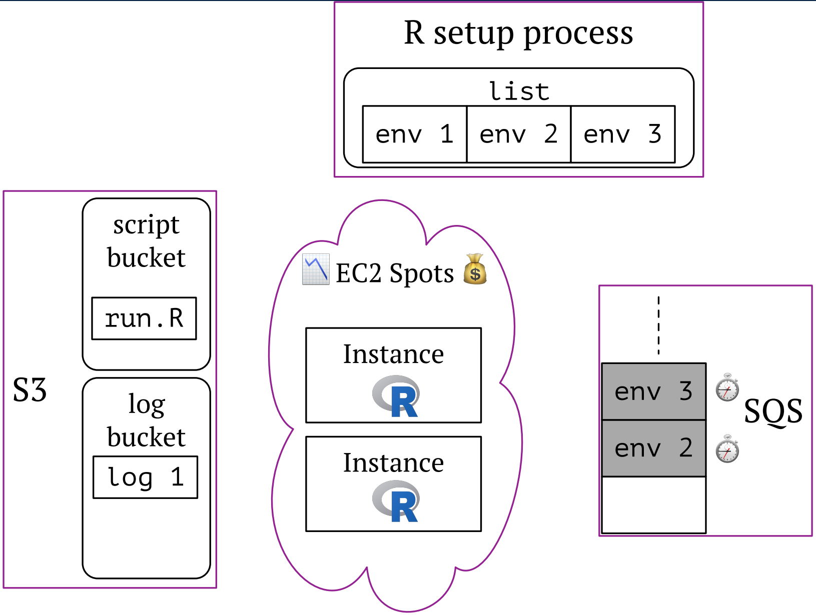
The approach (I)



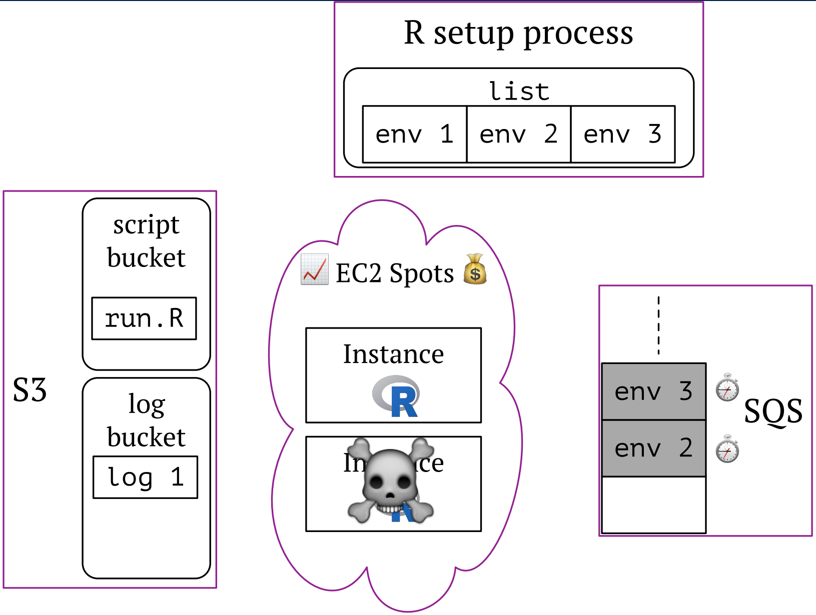
The approach (I)



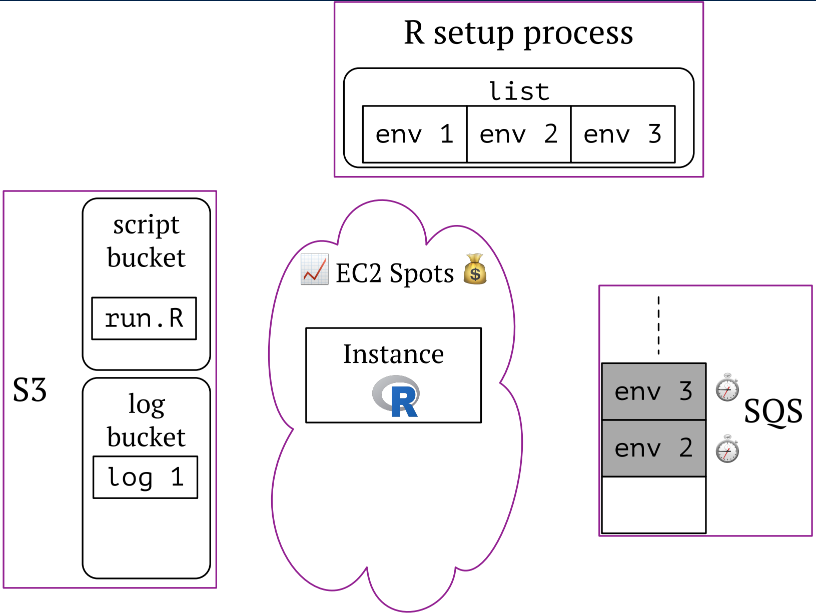
The approach (I)



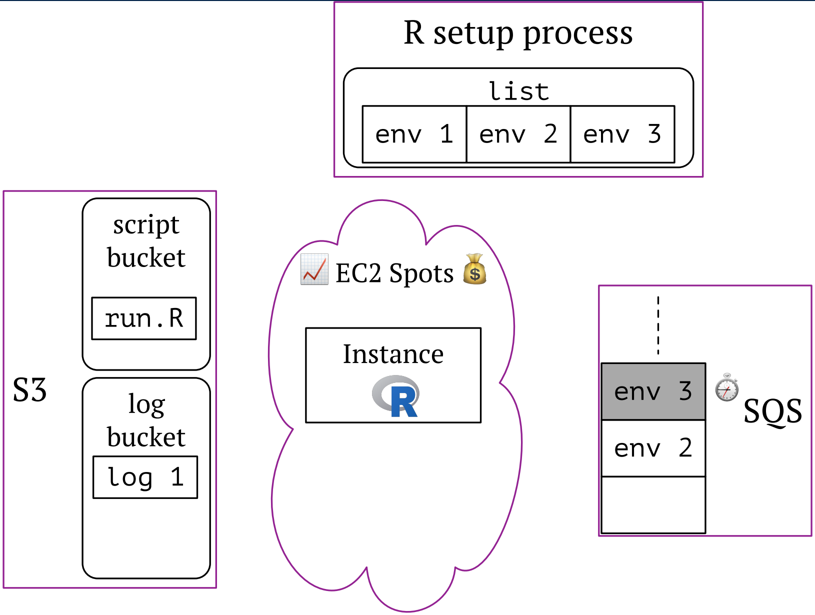
The approach (I)



The approach (I)



The approach (I)



The approach (II)

- An R setup process defines a list containing the different environments that should be on each node
    - elements of the list are individually serialised and sent to SQS queue
        - up to 256KB per message
        - highly reliable and cheap
- Script to run uploaded to S3. May depend on variables named in the list
- Spot instances are requested
    - user-data contains config for batch process (next slide)
    - doesn't have to be all in one request, homogeneous or in one region
    - logic baked into AMI
        - fetches script
        - fetches next queue item
        - sets up R environment
        - runs script
        - upon completion, deletes queue message

Toy example (leave-one-out CV) : Step 1, R file

We can assume `aws.s3` library will be loaded, others we must load.

```
library("randomForest")
Sys.setenv(
  "AWS_ACCESS_KEY_ID"="CHQB7923U4P9R50NNMNW",
  "AWS_SECRET_ACCESS_KEY"="xGXrdtzIRp1kjG2wwCoyBgWtxsyDx

s3load(object="myData.RData", bucket="mybucket")
rf <- randomForest(y~., data=myData[-i,])
s3save(rf,
       object=paste("LO_", i, "_rf.RData", sep=""),
       bucket="mybucket")
```

We therefore assume `i` is available on the workspace … this will be bundled into the queue.

Aside …

It isn't required to output objects to S3, just possibly the easiest. Other alternatives which are resiliant include:

- An SQS result queue which has results pushed back
- A DynamoDB or RDS with results pushed to a database for easier aggregation
- For exotic applications, perhaps triggering a Lambda service
- …

Toy example (leave-one-out CV) : Step 2, fill queue

```
library(RStudioAMI)

cred <-
  list(ACCESS_KEY_ID="CHQB7923U4P9R50NNMNW",
       SECRET_ACCESS_KEY="xGXrdtzIRp1kjG2wwCoyBgWtxsyDx1

myenv <- lapply(1:length(myData),
                function(i) {
                  list(i=i)
                })

res <- pushToSQS("TestPar", myenv, cred, "eu-west-1")
```

i-like.org.uk

Toy example (leave-one-out CV) : Step 3, Launch spots

The `user-data` of the launch should specify:

```
RSTUDIOAMI_RUN_BATCH=4
AWS_ACCESS_KEY_ID=CHQB7923U4P9R50NNMNW
AWS_SECRET_ACCESS_KEY=xGXrdtzIRp1kjG2wwCoyBgWtxsyDx1zTL
QUEUE_NAME=MyBatchQueue
QUEUE_REGION=eu-west-1
S3_SCRIPT_BUCKET=scriptbucket
S3_SCRIPT=runme.R
S3_LOG_BUCKET=logbucket
S3_LOG_PREFIX=log
```

Without `RSTUDIOAMI_RUN_BATCH` all other variables ignored.

Future work

Launch ETA: 2-3 weeks*

In addition, would like to see:

- Hook into RStudio web front end so you can log in and interactively see each node's progress
- Allow setting password via instance user-data
- Automatic job resume via deep process checkpointing of running job
  - only 2 minute window to checkpoint
  - so must be resiliant to checkpoint failure (cf SQS)
- Add Python and Julia batch jobs
- Your feedback!

* beta version should be finalised next week if anyone has a use case and would like to help with testing.