COMSM0010 Cloud Computing

Lecture 02

Assessment: the cloud programming project…

Scale-Out Compute in the Cloud

Dave Cliff

Department of Computer Science
University of Bristol

csdtc@bristol.ac.uk

University of BRISTOL

Remember this slide from Lecture 1…

Time Budget

Lectures: 20 hours

Feedback Assessments: 2 hours

Programming Report: 48 hours

Independent Study: 30 hours

TOTAL: 100 hours

NB There is way, way more reading on the reading list than you could hope to read in 20 hours – the whole list is intended as a resource that will be useful in coming years; precisely what you read depends on your interests.

Time Budget

Lectures: 20 hours

Feedback Assessments: 2 hours

Programming Report: 48 hours

Independent Study: 30 hours

TOTAL: 100 hours

NB There is way, way more reading on the reading list than you could hope to read in 20 hours – the whole list is intended as a resource that will be useful in coming years; precisely what you read depends on your interests.

Planning your time

Expected to take c.48 hours

You could think of this as six 8-hour days

(What follows is an example; you don't have to do it like this...)

- Day 1: reading, watching tutorials, thinking, deciding/designing what to do
- Day 2: getting familiar with SDK, stub coding, etc.

…

- Day 5: Writing the report, preparing figures/diagrams etc
- Day 6: Finishing off the report (10 pages including figs/diagrams & refs)

- That leaves only two days (Day 3, Day 4) for core coding of your system

It would be unwise to be overambitious (NB 10 pages is a maximum)

What we want you to do…

Scale-Up vs Scale-Out

Pre-cloud, the standard route to increasing compute capacity was Scale-Up

- Start with the biggest server you need, until it reaches its limits
- Next, buy a bigger server; costs typically rise badly/nonlinearly
- AKA Vertical Scaling

Cloud solutions instead focus on Scale-Out / Horizontal Scaling

- Start with a cluster of cheap commodity servers, spread the load over them
- When you need more capacity, add more cheap servers

For the coursework, we ask you to use cloud services to produce a scale-out solution to address a compute-intensive problem that is *embarrassingly parallelizable…*

If a task takes S seconds on one machine, with N machines it'll take $\sim S/N$ *seconds*

Coursework stages

For an embarrassingly-parallel compute-intensive task T…

1. Get T running on a local machine & gather baseline statistics.
2. Write code that launches T on a cloud service, & gather statistics.
3. Extend your code so that it launches T on N remote machines, aiming for run-times falling to ~ $^{1}/_{N}$ of the baseline, and tidies up nicely.
4. Use performance statistics so that N is set implicitly, by specifying a target time-to-completion and a confidence interval.
5. Creatively extend your system to extend its functionality

Task T: Blockchain Proof of Work (POW)

…repeatedly invoking the SHA256 cryptographic hash function to find a nonce.

Coursework stages

For an embarrassingly-parallel compute-intensive task T…

1. Get T running on a local machine & gather baseline statistics. 0-39%
2. Write code that launches T on a cloud service, & gather statistics. 40-54%
3. Extend your code so that it launches T on N remote machines, aiming for run-times falling to ~ $^{1}/_{N}$ of the baseline, and tidies up nicely. 55-69%
4. Use performance statistics so that N is set implicitly, by specifying a target time-to-completion and a confidence interval. 70-80%
5. Creatively extend your system to extend its functionality 81-100%

EACH PERSON WORKS INDIVIDUALLY (NO GROUPS)

Task T: Blockchain Proof of Work (POW)

…repeatedly invoking the SHA256 cryptographic hash function to find a nonce.

Coursework stages

For an embarrassingly-parallel compute-intensive task T…

1. Get T running on a local machine & gather baseline statistics. 0-39%
2. Write code that launches T on a cloud service, & gather statistics. 40-54%
3. Extend your code so that it launches T on N remote machines, aiming for run-times falling to ~ $^{1}/_{N}$ of the baseline, and tidies up nicely. 55-69%
4. Use performance statistics so that N is set implicitly, by specifying a target time-to-completion and a confidence interval. 70-80%
5. Creatively extend your system to extend its functionality 81-100%

EACH PERSON WORKS INDIVIDUALLY (NO GROUPS)

Task T: Blockchain Proof of Work (POW)

…repeatedly invoking the SHA256 cryptographic hash function to find a nonce.

Full details in a document to be posted on Blackboard…

Coursework

For an

1. G
2. W
3.
4.
5.

UoB COMSM0010 Cloud Computing, AY2019-20
Coursework A (summative assessment, 50% of total grade for the unit)

Horizontal Scaling for an Embarrassingly Parallel Task: Blockchain Proof-of-Work in the Cloud

Date issued: Thursday October 3rd, 2019 (Lecture 02).
Submission deadline (on SAFE): 12:00 noon on Thursday December 6th, 2019.

1. Introduction

In the lectures on this unit we cover the difference between *vertically*- and *horizontally*-scaling system architectures. One of the advantages offered by cloud computing is elastic (smoothly varying) horizontal scaling: new virtual machines (VMs) can be created, used, and deleted as and when they are needed. In this way, as the computational demands on a system vary over time, the number of VMs used by the system can be scaled up and down as required.

Horizontal scalability is particularly attractive for situations where the task at hand is *embarrassingly parallelizable*. A computational task is embarrassingly parallelizable if, when it is spread over N machines working in parallel, the expected time for completion of the task is C/N (or less), where C is the time it takes a single machine to complete the task.

One example of an embarrassingly parallelizable task is rendering comp
video frames for a movie. Digital video is a sequence of stat
produced to be displayed at 30 frames per se
machine takes 7.5 minutes, that
frame takes the sa
the cl

Stuff you might need to know…

IaaS basics via AWS

(Last lecture: broad overview; introduced IaaS/PaaS/SaaS distinction)

IaaS in more detail via the basics of a minimal set of initial AWS services:

- Amazon Simple Storage Service (S3)
- Amazon Elastic Compute Cloud (EC2)
- Amazon Simple Queue Service (SQS)

S3: Simple Storage Service

Amazon Simple Storage Service (S3): Basics

Source: Reese

Amazon S3 is cloud-based persistent storage

Operates independently from other Amazon services

- Applications on your local servers can use S3 as remote storage

"Simple" features; not necessarily simple to use. ☺

- S3 lets you put data in the cloud, and pull it back again
- You don't need to know anything about how, or where, it is stored
- In several respects, it is simpler (=less clever) than a remote filestore

You don't store files, you store objects (max size: 5Tb)

You don't keep objects in directories/folders, you keep them in buckets

Objects have a size limit (5Gb) on a single PUT; (bigger via MultiPart Upload)

All buckets (really all) in S3 share the same namespace: no sub-buckets

Content of buckets and objects can be made visible to the general public

Amazon Simple Storage Service (S3): Usage

Source: Reese

Sign up for an AWS account – same UI as buying books etc.

Default storage location can be selected (US or Europe)

S3 is accessed via API – either SOAP (xml) or REST (http)

- Wrappers available to abstract the API for programmers, e.g. Jets3t for Java
- s3cmd command-line wrapper, written in Python (you can read the source)

APIs allow you to:

- Create new buckets
- Upload new objects
- Find buckets/objects
- Retrieve (meta)data for buckets/objects
- Delete buckets & objects

Amazon Simple Storage Service (S3): Example

Source: Reese

(making a copy of a file via the cloud – would be much easier to use “cp”)

Amazon Simple Storage Service (S3): Example

Source: Reese

```
(making a copy of a file via the cloud – would be much easier to use “cp”)

(create a bucket)
$ s3cmd mb s3://bristol.comsm0010.dc.bucket1
```

Amazon Simple Storage Service (S3): Example

Source: Reese

(making a copy of a file via the cloud – would be much easier to use "cp")

(create a bucket)

```
$ s3cmd mb s3://bristol.comsm0010.dc.bucket1
```

(upload a file)

```
$ s3cmd put myfile.tst s3://bristol.comsm0010.dc.bucket1/myfile.tst
```

Amazon Simple Storage Service (S3): Example

Source: Reese

```
(making a copy of a file via the cloud – would be much easier to use “cp”)

(create a bucket)
$ s3cmd mb s3://bristol.comsm0010.dc.bucket1
(upload a file)
$ s3cmd put myfile.tst s3://bristol.comsm0010.dc.bucket1/myfile.tst
(download it into a different name)
$ s3cmd get s3://bristol.comsm0010.dc.bucket1/myfile.tst myfile2.tst
```

Amazon Simple Storage Service (S3): Example

Source: Reese

```
(making a copy of a file via the cloud – would be much easier to use "cp")

(create a bucket)
$ s3cmd mb s3://bristol.comsm0010.dc.bucket1
(upload a file)
$ s3cmd put myfile.tst s3://bristol.comsm0010.dc.bucket1/myfile.tst
(download it into a different name)
$ s3cmd get s3://bristol.comsm0010.dc.bucket1/myfile.tst myfile2.tst
(delete the cloud version)
$ s3cmd del s3://bristol.comsm0010.dc.bucket1/myfile.tst
```

Amazon Simple Storage Service (S3): Example

Source: Reese

```
(making a copy of a file via the cloud – would be much easier to use “cp”)

(create a bucket)
$ s3cmd mb s3://bristol.comsm0010.dc.bucket1
(upload a file)
$ s3cmd put myfile.tst s3://bristol.comsm0010.dc.bucket1/myfile.tst
(download it into a different name)
$ s3cmd get s3://bristol.comsm0010.dc.bucket1/myfile.tst myfile2.tst
(delete the cloud version)
$ s3cmd del s3://bristol.comsm0010.dc.bucket1/myfile.tst
(delete the bucket too)
$ s3cmd rb s3://bristol.comsm0010.dc.bucket1
```

Amazon Simple Storage Service (S3): Example

Source: Reese

```
(making a copy of a file via the cloud – would be much easier to use "cp")

(create a bucket)
$ s3cmd mb s3://bristol.comsm0010.dc.bucket1
(upload a file)
$ s3cmd put myfile.tst s3://bristol.comsm0010.dc.bucket1/myfile.tst
(download it into a different name)
$ s3cmd get s3://bristol.comsm0010.dc.bucket1/myfile.tst myfile2.tst
(delete the cloud version)
$ s3cmd del s3://bristol.comsm0010.dc.bucket1/myfile.tst
(delete the bucket too)
$ s3cmd rb s3://bristol.comsm0010.dc.bucket1


$ s3cmd ls    //list all my buckets
```

Amazon Simple Storage Service (S3): Example

Source: Reese

```
(making a copy of a file via the cloud – would be much easier to use "cp")

(create a bucket)
$ s3cmd mb s3://bristol.comsm0010.dc.bucket1
(upload a file)
$ s3cmd put myfile.tst s3://bristol.comsm0010.dc.bucket1/myfile.tst
(download it into a different name)
$ s3cmd get s3://bristol.comsm0010.dc.bucket1/myfile.tst myfile2.tst
(delete the cloud version)
$ s3cmd del s3://bristol.comsm0010.dc.bucket1/myfile.tst
(delete the bucket too)
$ s3cmd rb s3://bristol.comsm0010.dc.bucket1


$ s3cmd ls    //list all my buckets
$ s3cmd ls s3://bristol.comsm0010.dc.bucket1 //list just the named bucket
```

EC2: Elastic Compute Cloud

Amazon Elastic Compute Cloud (EC2): basics

Source: Reese

The sexy bit: remotely accessible virtual network of virtual servers

- hypervisor/virtualisation is open-source Xen, not VMWare

Usually run EC2 with S3 providing the storage (that's why we did S3 first)

A single EC2 virtual server, with your chosen OS etc, is an instance

An instance is instantiated (ugh) from an Amazon Machine Image (AMI)

- One AMI can be cloned *n* times to create *n* instances
- You can build your own by cloning an AMI from your local server
- Or Amazon have a bunch of prebuilt AMIs you can choose from
- Or there are third-party AMI providers springing up

EC2 dynamically assigns a unique IP address to each instance

- That IP address is reassigned, maybe to someone else, when your instance dies

EC2 can give you an Elastic IP Address (that's static, not dynamic) if you want one

EC2 instances run in availability zones (Azs); AZs grouped into regions

– AZ is a bit like a single data center; region guarantee is >1 AZ at 99.95% uptime

Amazon Elastic Compute Cloud (EC2): usage

Source: Reese

Same basic API routes as S3, command-line, plus some nice GUIs too

- Amazon's own AWS console
- ElasticFox plug-in for Firefox
- Third-party cloud management tools like enStratus and RightScale (used by Animoto)

Beware selecting unknown AMIs – some are junk/malware!

Three types of storage (from cheapest to most expensive…):

- Ephemeral local storage in the instance (dies with the instance)
- Persistent cloud (S3)
- SAN-style Elastic Block Storage (EBS)
  - Allows user to create volumes from 1Gb to 1Tb
  - Any number of volumes may be mounted from a single instance

S3 is slow, medium-reliable, but super-durable: never loses data; good for DR backups

Instance storage is simple and cheap, but speed can be very poor

EBS is high on speed, reliability, durability, but is complex and costly

Amazon Elastic Compute Cloud (EC2): tutorial videos

http://www.youtube.com/watch?v=OLfmqcYnhUM

Note the warning at 0m39s: if you do not terminate your session correctly, you can continue to run up a bill even after you have logged out.

See also Greg Wilson (Adobe) tutorial on EC2 micro-instances

http://www.youtube.com/watch?v=ZAB8wCg9MyE

Instance types – c.2012 (in 2019 there are lots more)

(NB: one EC2 Compute Unit (ECU) approx the CPU capacity of a 1.0-1.2GHz 2007Opteron/Xeon)

Source: AWS website

Standard Instances (well suited for most applications)

- Small Instance (Default) 1.7 GB memory, 1 EC2 Compute Unit (1 virtual core with 1 EC2 Compute Unit), 160 GB instance storage, 32-bit platform
- Large Instance 7.5 GB of memory, 4 EC2 Compute Units (2 virtual cores with 2 EC2 Compute Units each), 850 GB of instance storage, 64-bit platform
- Extra Large Instance 15 GB of memory, 8 EC2 Compute Units (4 virtual cores with 2 EC2 Compute Units each), 1690 GB of instance storage, 64-bit platform

High-Memory Instances (for high throughput applications, including database apps)

- High-Memory Double Extra Large Instance 34.2 GB of memory, 13 EC2 Compute Units (4 virtual cores with 3.25 EC2 Compute Units each), 850 GB of instance storage, 64-bit platform
- High-Memory Quadruple Extra Large Instance 68.4 GB of memory, 26 EC2 Compute Units (8 virtual cores with 3.25 EC2 Compute Units each), 1690 GB of instance storage, 64-bit platform

High-CPU Instances (for compute-intensive applications)

- High-CPU Medium Instance 1.7 GB of memory, 5 EC2 Compute Units (2 virtual cores with 2.5 EC2 Compute Units each), 350 GB of instance storage, 32-bit platform
- High-CPU Extra Large Instance 7 GB of memory, 20 EC2 Compute Units (8 virtual cores with 2.5 EC2 Compute Units each), 1690 GB of instance storage, 64-bit platform

The complete 2014 AMI product range

Source http://aws.amazon.com/ec2/pricing/

General Purpose: t2.micro; t2.small; t2.medium; m3.medium; m3.large; m3.xlarge; m3.2xlarge

Compute-Optimized: c3.large; c3.xlarge; c3.2xlarge; c3.4xlarge; c3.8xlarge

GPU: g2.2xlarge

Memory-Optimized: r3.large; r3.xlarge; r3.2xlarge; r3.4xlarge; r3.8xlarge

Storage-Optimized: i2.xlarge; i2.2xlarge; i2.4xlarge; i2.8xlarge; hs1.8xlarge

NB all except t2 range now use SSD storage rather than EBS.

Range of OS's now: Linux, RedHat Enterprise Linux; Suse Linux Enterprise Server; Windows; Windows with SQL Standard; Windows with SQL Web

So, 6 OS options x 23 instance types: 138 different combinations, different prices.

The complete 2015 AMI product range

Source http://aws.amazon.com/ec2/pricing/

General Purpose: t2.micro; t2.small; t2.medium; t2.large; m3.medium; m3.large; m3.xlarge; m3.2xlarge; m4.large; m4.xlarge; m4.2xlarge; m4.4xlarge; m4.10xlarge.

Compute-Optimized: c3.large; c3.xlarge; c3.2xlarge; c3.4xlarge; c3.8xlarge; c4.large; c4.2xlarge; c4.4xlarge; c4.8xlarge.

GPU: g2.2xlarge; g2.8xlarge.

Memory-Optimized: r3.large; r3.xlarge; r3.2xlarge; r3.4xlarge; r3.8xlarge

Storage-Optimized: i2.xlarge; i2.2xlarge; i2.4xlarge; i2.8xlarge; d2.xlarge; d2.2xlarge; d2.4xlarge; d2.8xlarge

Range of OS's now: Linux, RedHat Enterprise Linux; Suse Linux Enterprise Server; Windows; Windows with SQL Standard; Windows with SQL Web; Windows with SQL Enterprise

So, 7 OS options x 37 instance types: 259 different combinations, different prices.

SQS: Simple Queue Service

Amazon Simple Queue Service (SQS): basics

Source: Murty

Reliable, loosely-coupled fault-tolerant storage & delivery of messages

- Between any clients or computers connected to the internet
- Senders & recipients do not have to communicate directly
- No requirement that either side be always-available or connected to the net

A message is up to 256Kb of text-data, sent to SQS & stored until it is delivered

A queue serves to group related messages together

- Also controls delivery & access-control options

SQS is accessible to clients on any HTTP-enabled platform

Messages are stored redundantly over multiple data-centers: truly distributed

Because of this…

- Message retrievals may be incomplete
- Messages may not be delivered quickly (2-10 seconds typical)
- Messages may be delivered out of order
- Messages may be redelivered

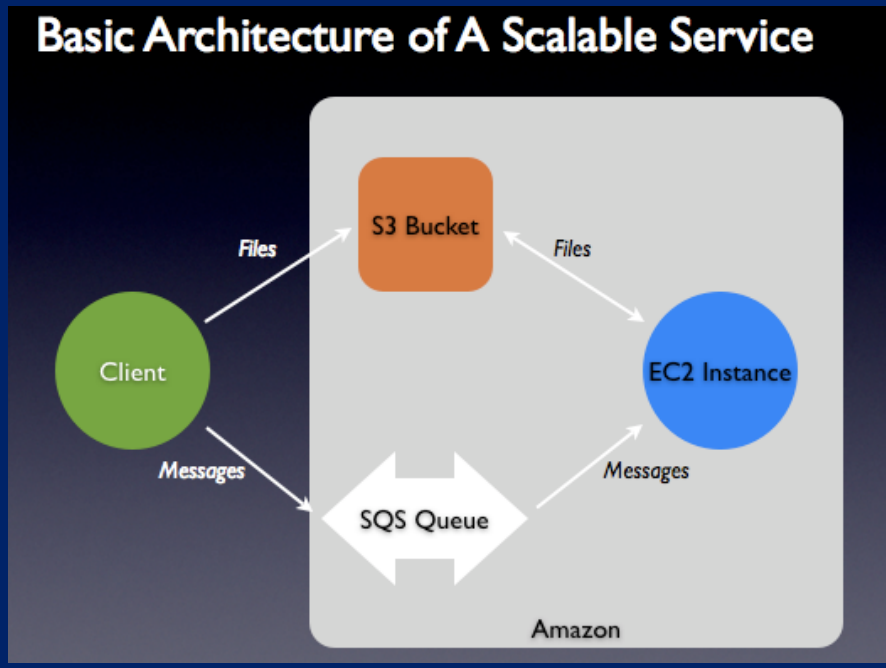
Mitch Garnaat's Monster Muck Mashup

Sources: Murty and AWS web

Service that converts AVI videos to mp4 (iPod), using:

- "Boto" Python interface to AWS
- S3 to store the video files
- EC2 to do the conversion processing
- SQS for inter-process communication

Uses AWS for scalability



A) Upload a bunch of video files to S3 bucket

B) For each file, add a msg to SQS input queue

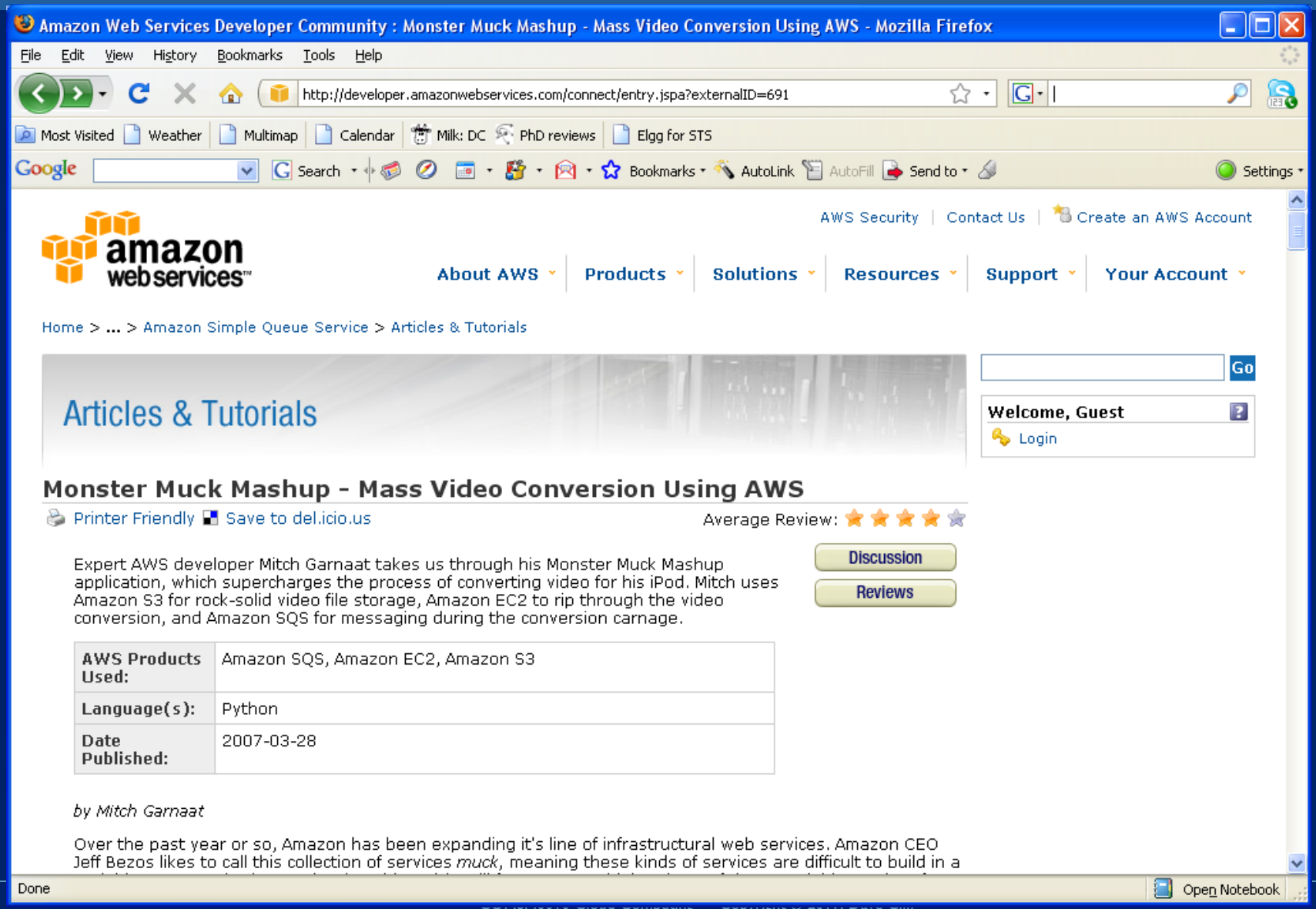
C) On the EC2 Instance, repeat this:

1. Read a msg MI from InputQ
2. Retrieve from S3 the video VI specified in MI
3. Do the conversion, creating VO
4. Store the VO file(s) in S3
5. Write msg MO to SQS OutputQ
6. Delete MI from InputQ

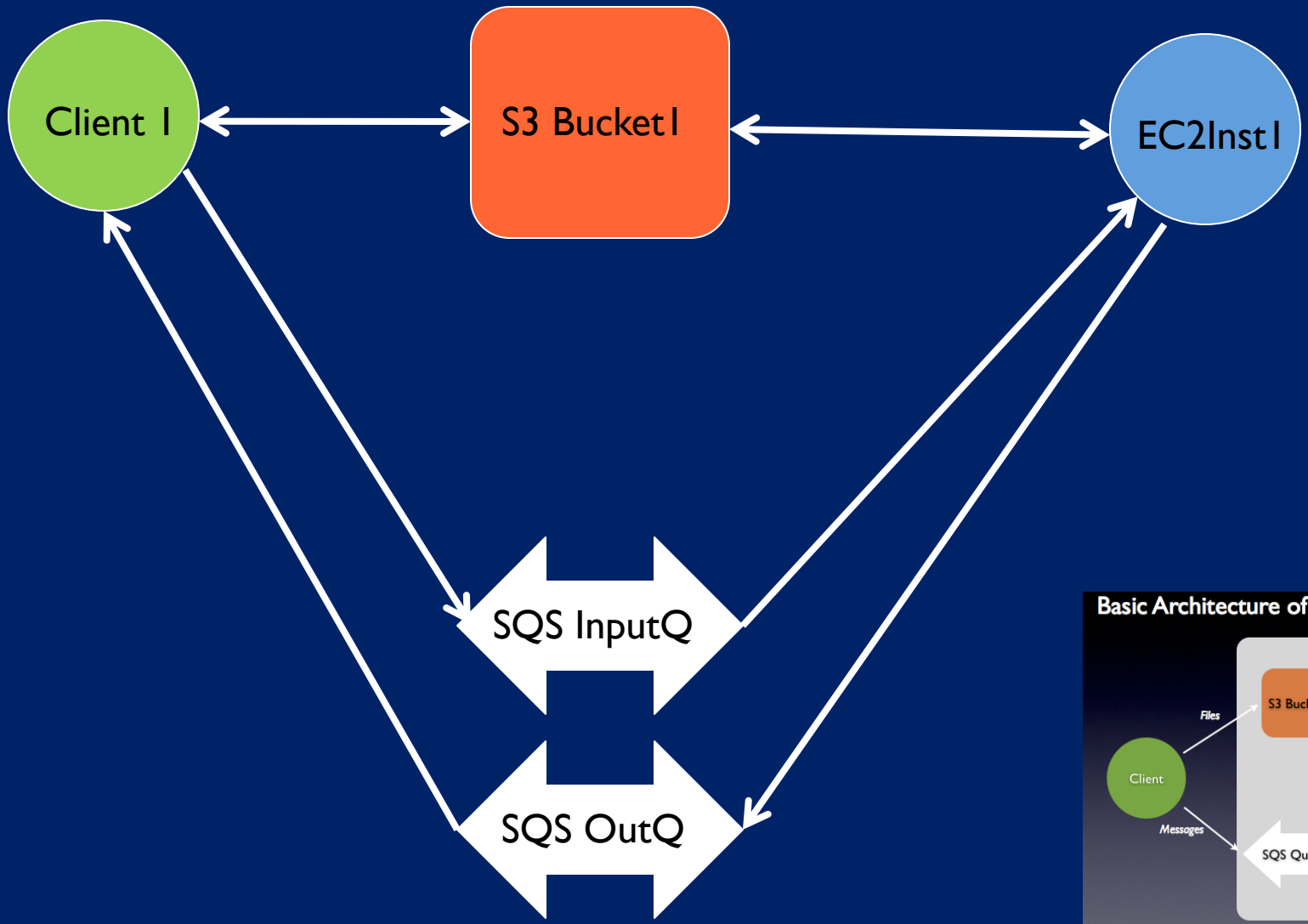
(until InputQ is empty)

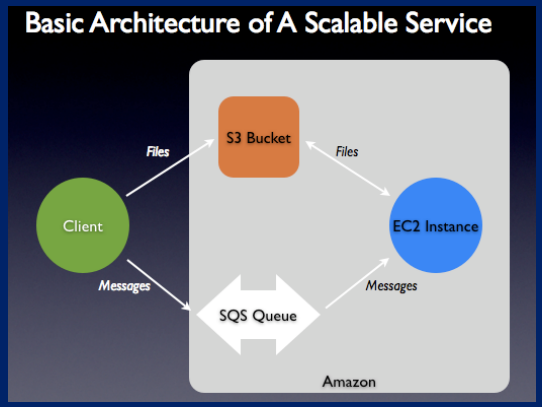
Mitch Garnaat's Monster Muck Mashup

http://aws.amazon.com/articles/691?_encoding=UTF8&jiveRedirect=1

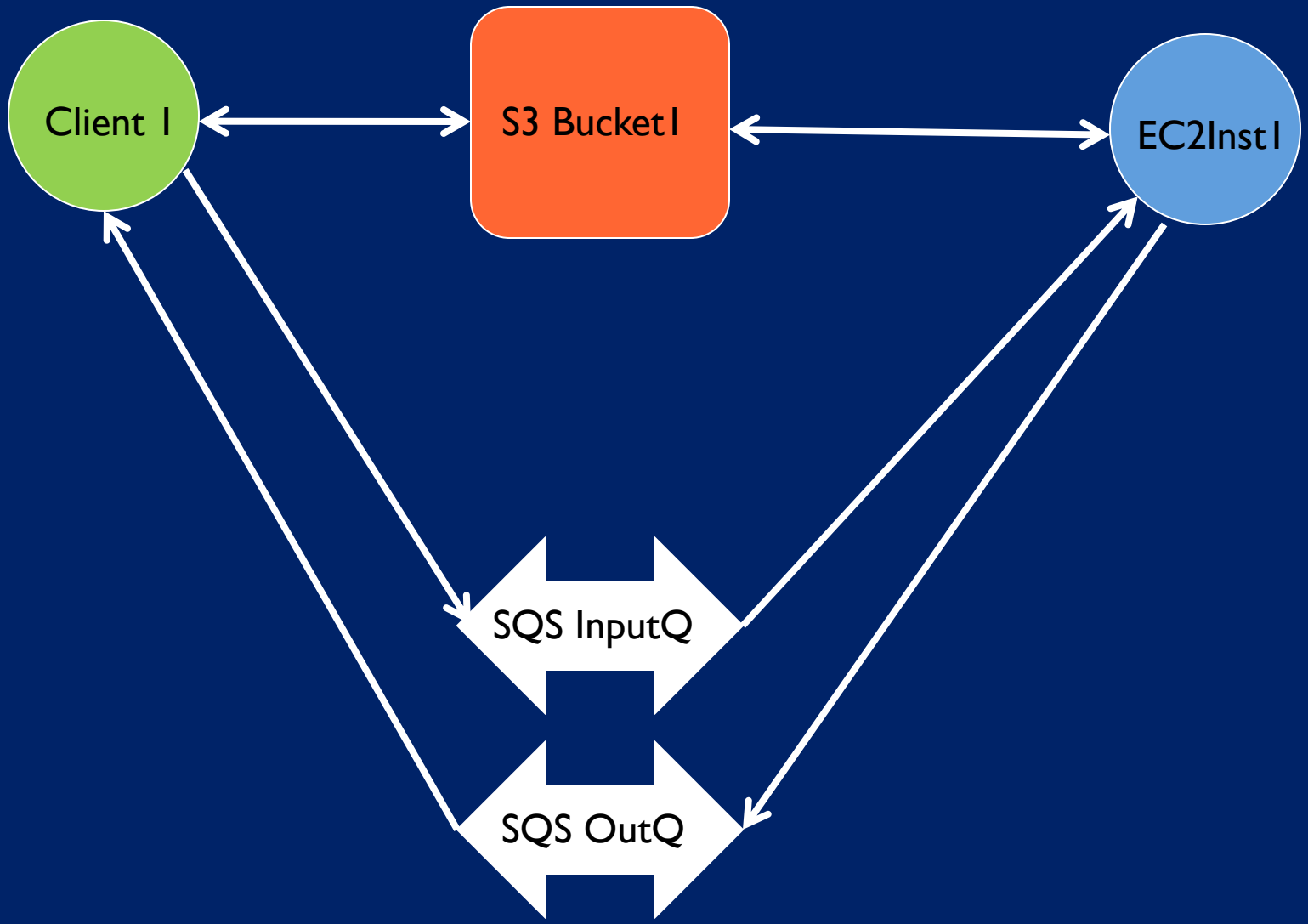


Scaling it up…

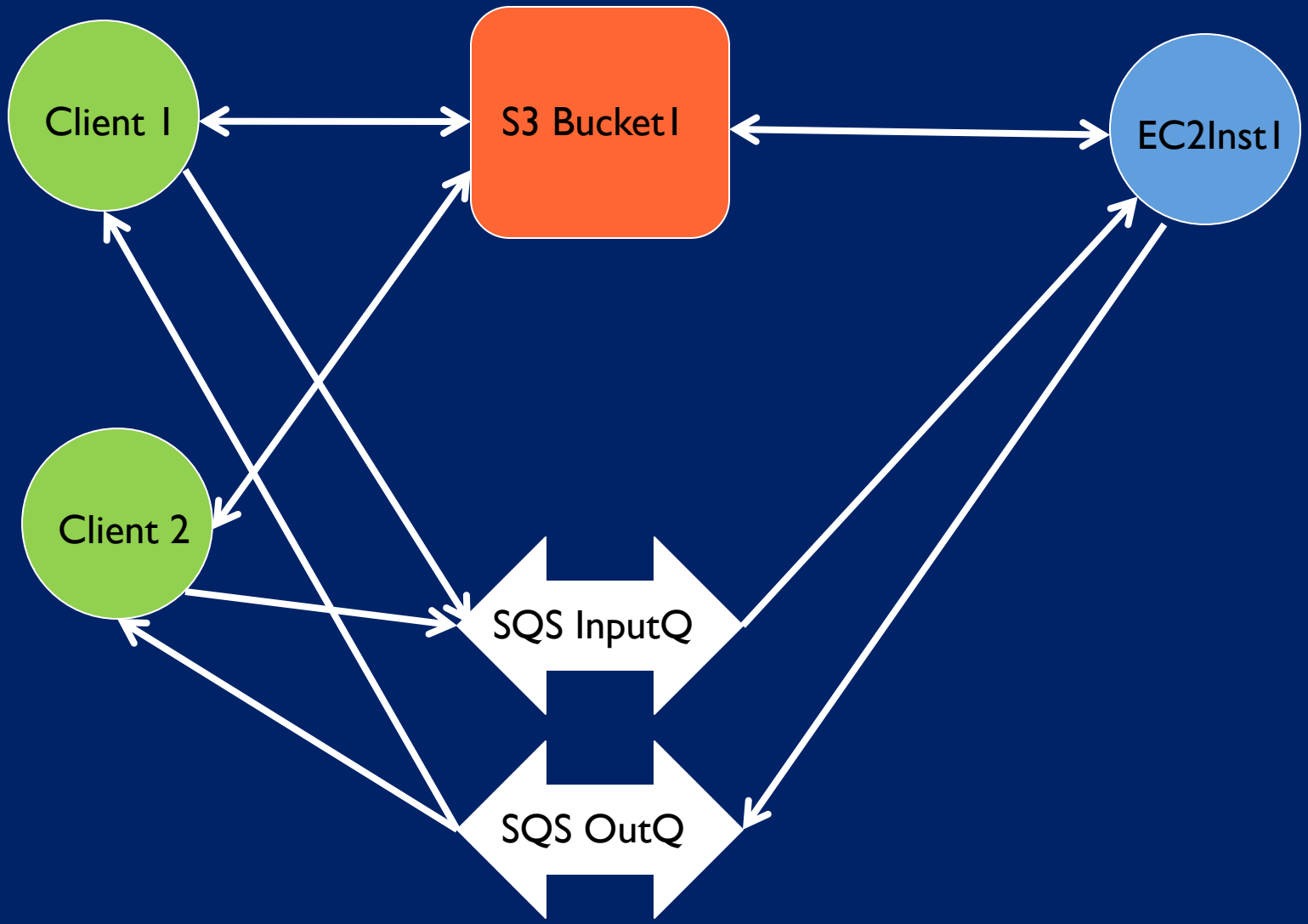




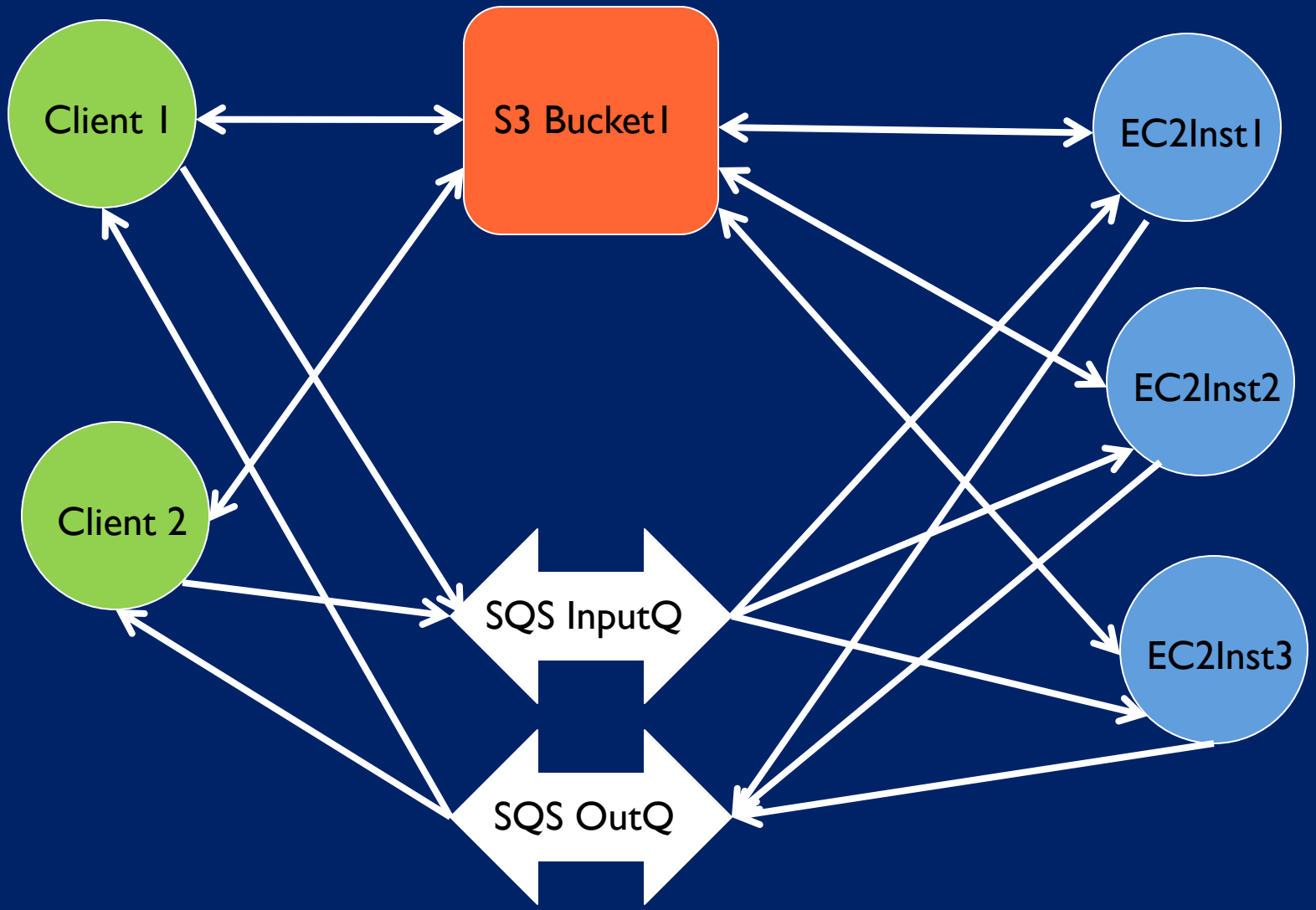
Scaling it up…



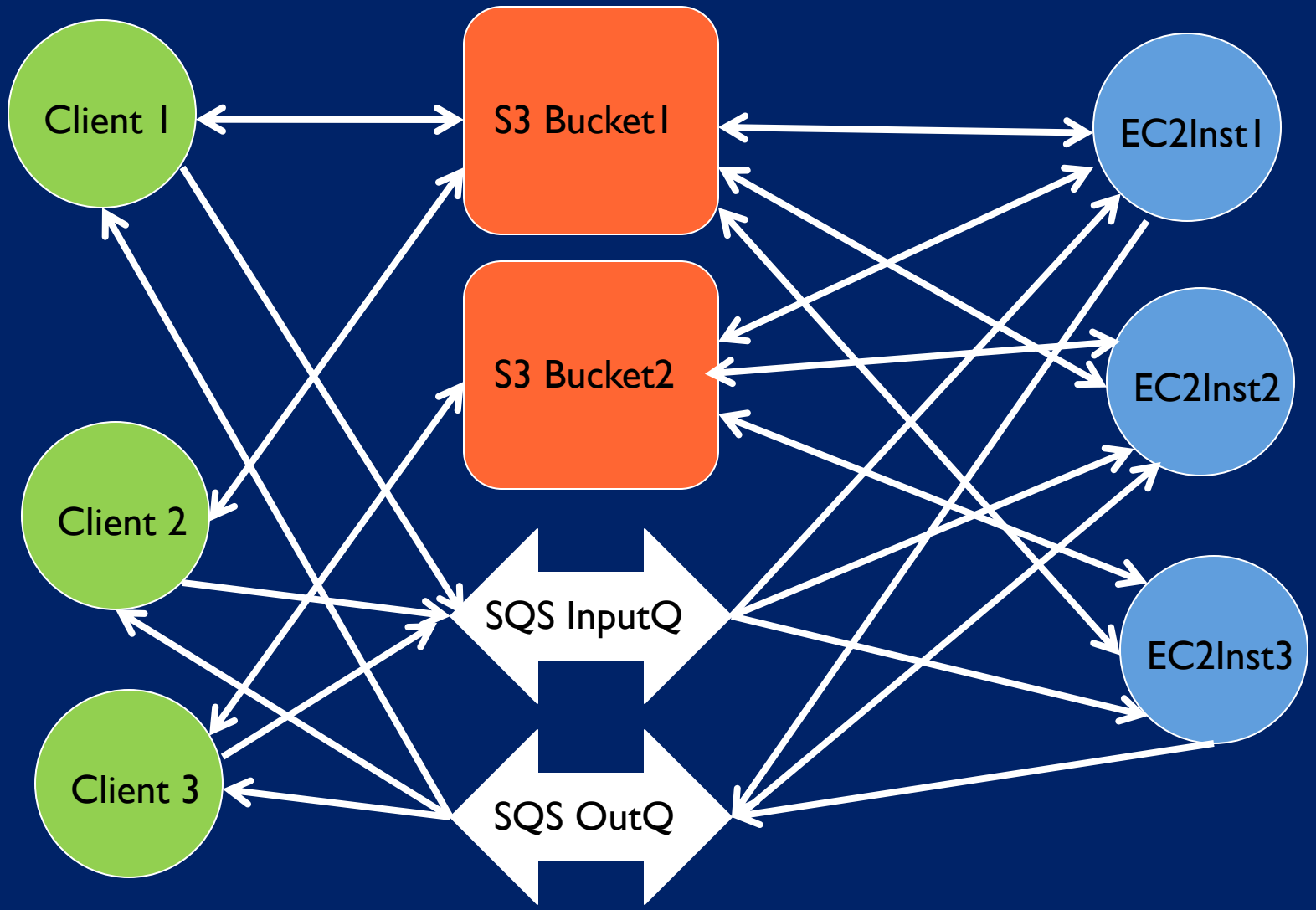
Scaling it up…



Scaling it up…



Scaling it up…



MonsterMuckMashup has clues for how to do the assessment

MonsterMuckMashup has clues for how to do the assessment

Once you've done the code, you have to write it up

…here are Dave's eight top tips on writing…

All of which is in this handy document (on Blackboard)

Tips on Writing

Dave Cliff, University of Bristol.

21st November 2013.

A university degree, whether undergraduate or postgraduate, usually depends quite heavily on the submission of written work for formal assessment. That is, the grades you get, and your final degree classification, are dependent on things you've written – essays, reports, dissertations, and of course the answers that you write on exam-papers. And it doesn't stop there: for very many graduate- or postgraduate-level jobs, there is an expectation, or indeed a need, for you to be able to clearly and concisely express facts, ideas, and opinions in written form.

This brief document contains some tips on writing, some tips on things to do to improve your written work, and some tips on things to avoid. Each of the tips here are things that I have found myself saying frequently, year after year, to very many students, in feedback on essays and reports and dissertations and MSc theses and PhD theses.

Tip 1. Write for an audience.

When you sit down to write, have some idea in your head of who it is that you're writing for, or at least of what group of people your writing is aimed at: your *audience.* The language and style of your writing will be affected by this. In particular, it helps you to

Tip 1: Write for an Audience

Tip 2: Say what you want to say

Tip 2: Say what you want to say… step 1

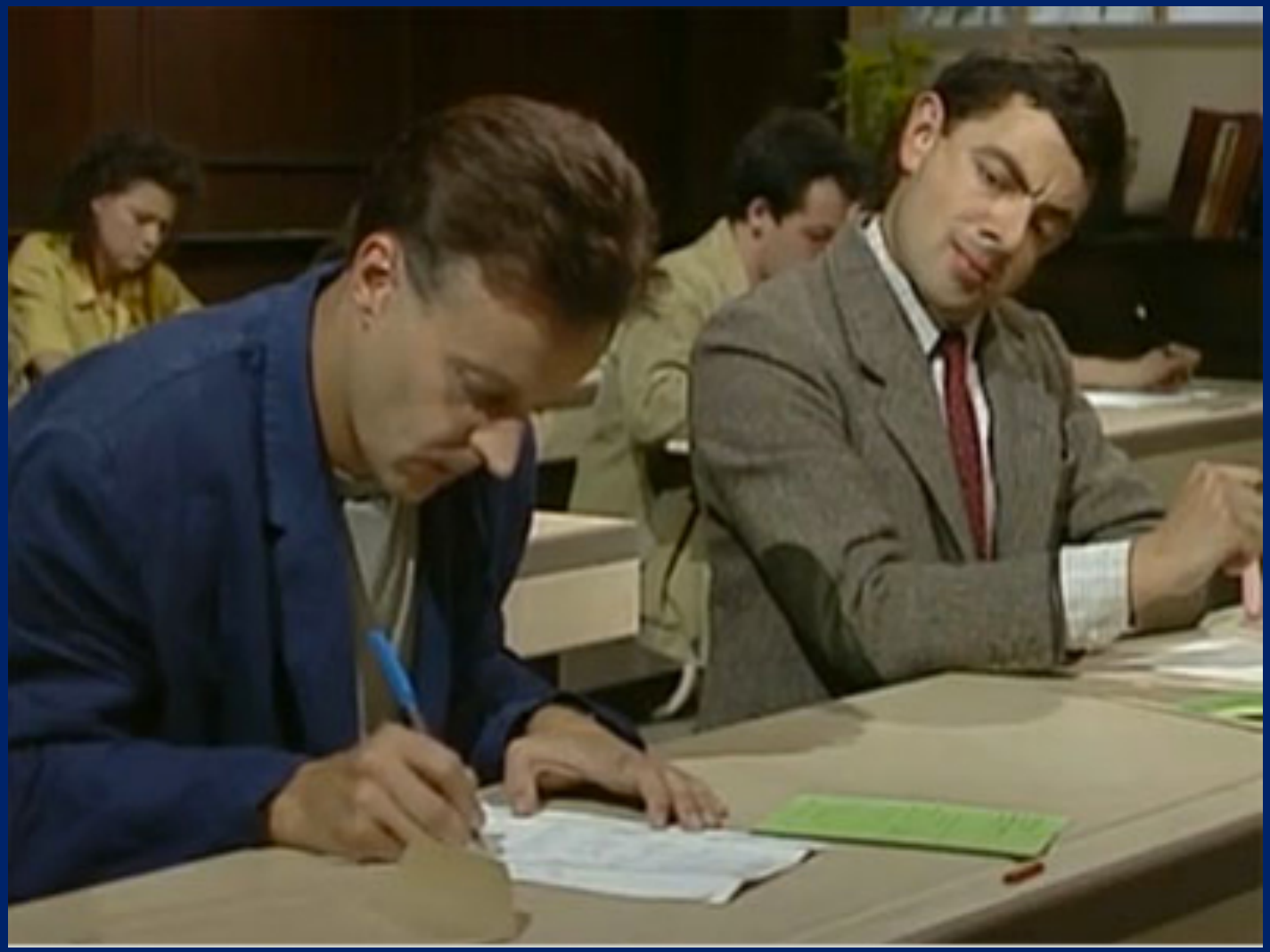
Tip 2: Say what you want to say… step 2

Tip 2: Say what you want to say… step 3

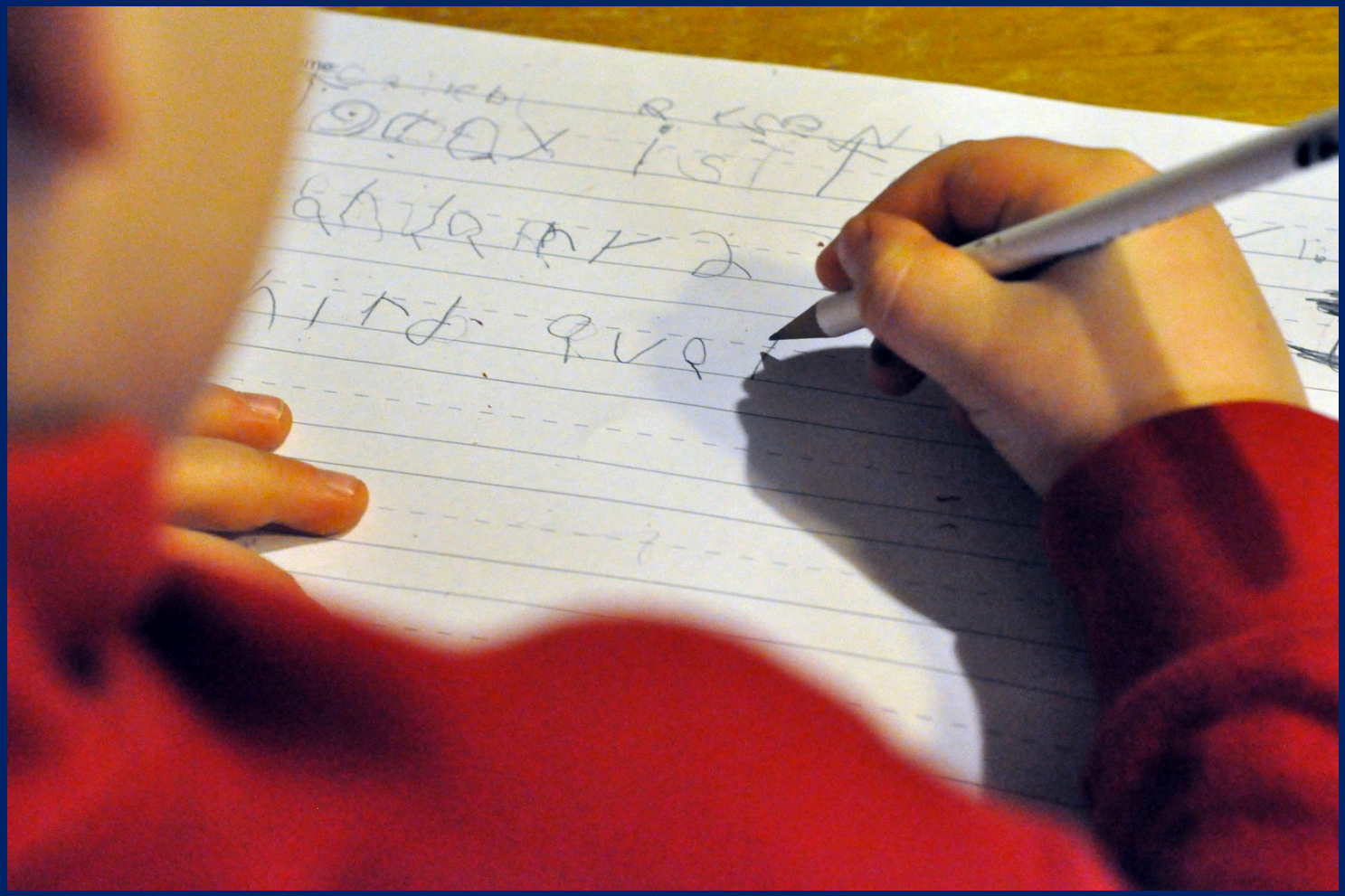
Tip 3: Remember the reader

Tip 4: Improve it by not working on it

Tip 5: Don’t plagiarise



Tip 6: Write like a grown-up, not a schoolkid



Tip 7: Get the basics right

Its vs *it's* (compare to *his* vs *he is*)

Students vs *student's* vs *students'*

Their vs *they're* vs *there*

Another vs *an other; furthermore* vs *further more*

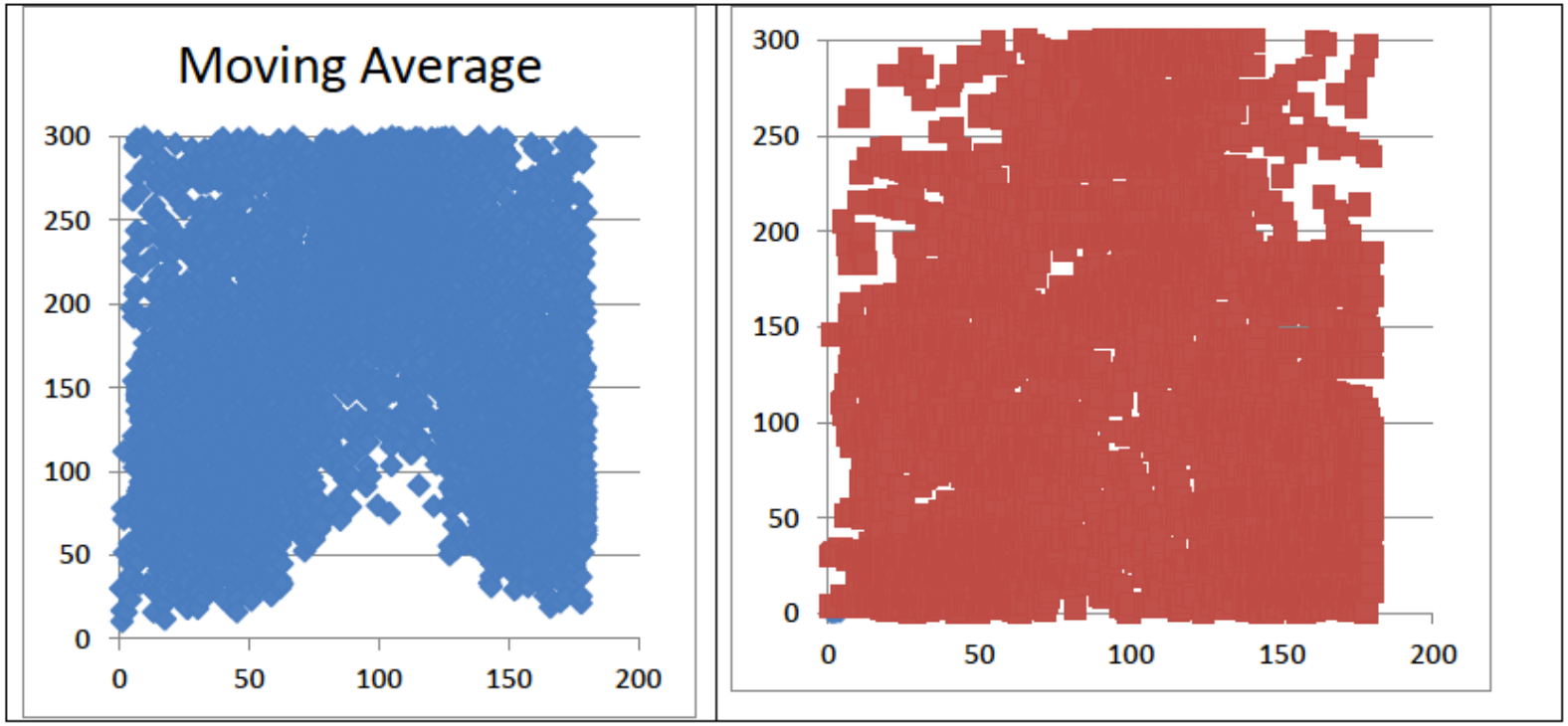
Make sure you know how to use capital letters (use *italics* for emphasis)

Be consistent with citations and references

Don't cite Wikipedia as a primary source (Scholarpedia is OK though)

Learn to use Google Scholar

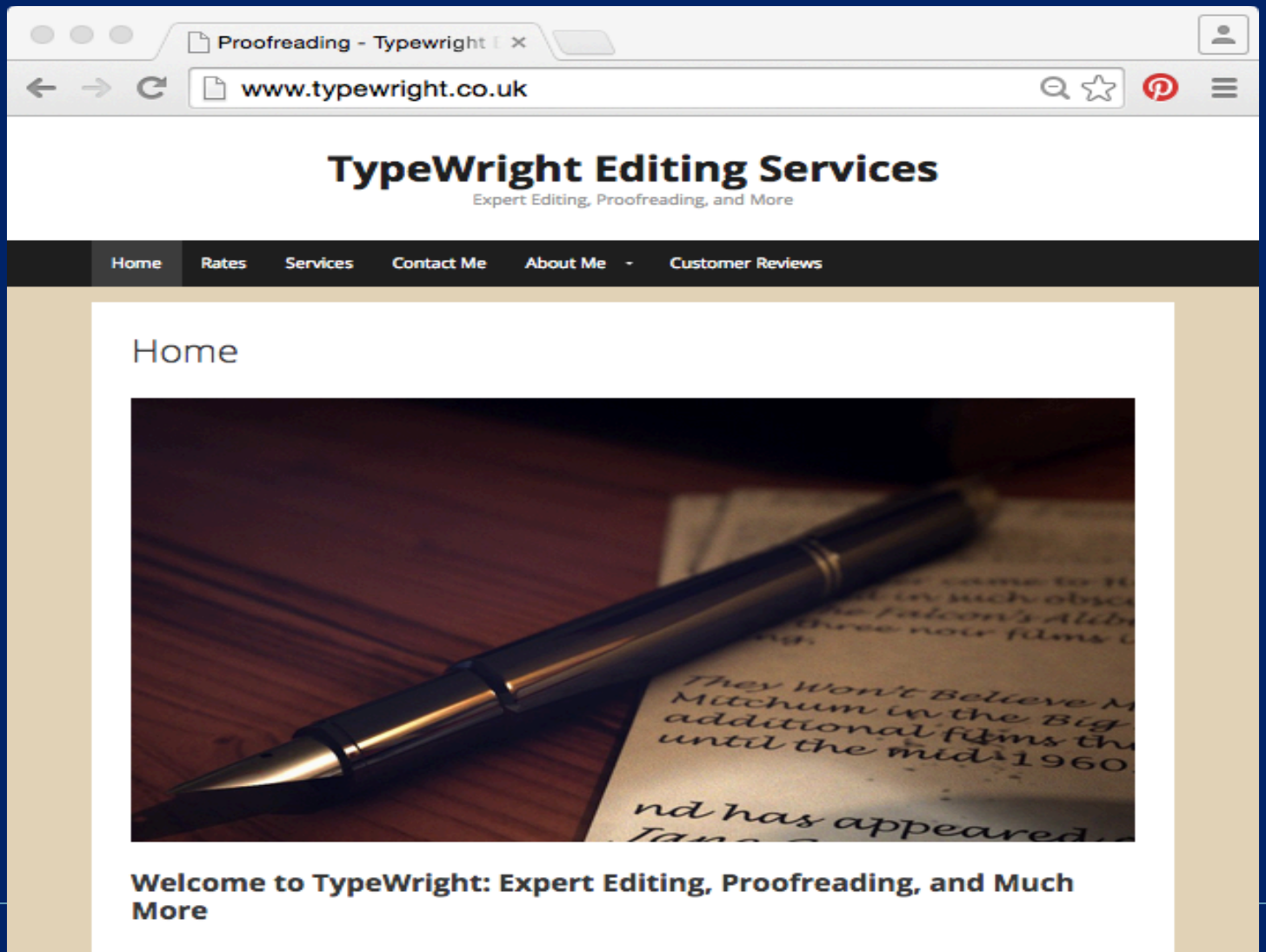
Tip 8: Use figures sensibly



The charts above show Ask order prices (on right) and their moving averages for a market session with 10-190,200-300,10-190 demand supply schedules. As can be seen, the moving average (*as is used regularly in financial trend analysis*) gives a (*bottom*) trend of the ask surface. A very similar surface could be visualized at the bottom of the chart for the moving averages of bid orders which will gradually move up into the valley/gap of the "ask moving averages" surface and then go down again. And if transaction orders were plotted, they would have been exactly in between these two surfaces.

Tip 9: hire a professsional proofreader/editor

E.g.: www.typewright.co.uk (one of very many, as used by my PhD students)



Submitting Assessments

Submission of all assessments (feedback assessments and formal coursework) used to be via SAFE

And now SAFE is gone.

Right now we are working on finding out how to do it in the new system – we will tell you very soon!