5CC515 – Networks and Security Concurrency

UNIVERSITY OF DERBY

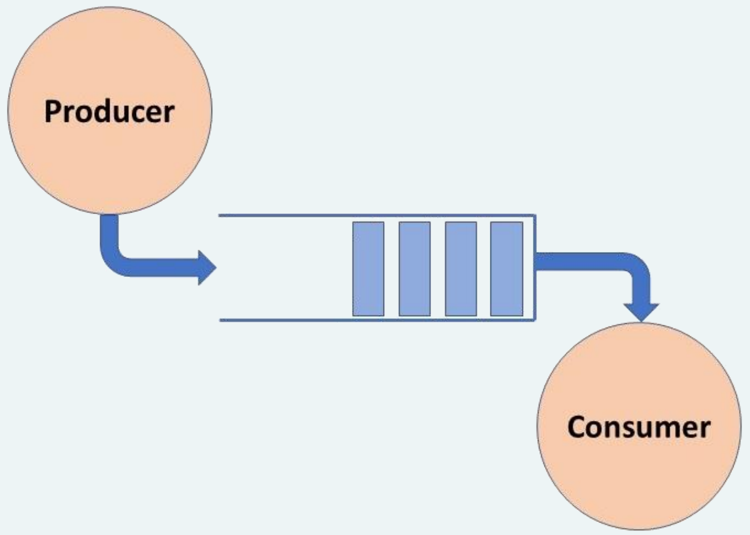
derby.ac.uk

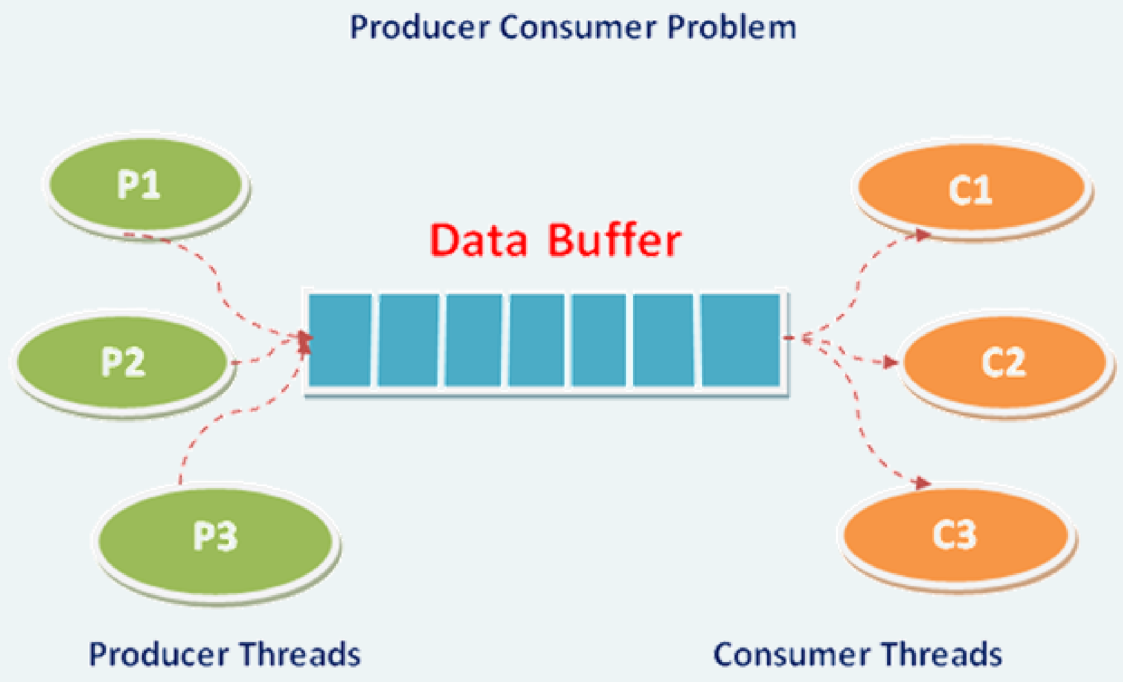
Order of Execution Problems – The Producer-Consumer Problem

- In some applications, you not only need to control access to critical sections by different threads, but you also need to control the order of execution of operations on different threads.
- The producer-consumer problem is one example of an order of execution problem. In this type of problem, there are two types of threads:
  - Producers – A producer creates a data element and sends that element to the consumer
  - Consumers – A consumer receives the data element from the producer and does something with it.
- When the producer and consumer run at different speeds, we need a store in between the producer and consumer that can hold the data elements produced by the producer until they are ready to be consumed by the consumer. This is a queue, often referred to as a *buffer*.

The Producer-Consumer Problem

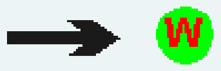
- There are two types of buffer we can use:
  - An Unbounded Buffer.
    - In this case, the buffer is of an indefinite size.
    - The only synchronisation issue we need to take care of is that we must make sure that the consumer cannot take a data item from an empty buffer.
  - A Bounded Buffer
    - The buffer is of a defined finite size
    - There are two synchronisation issues we need to deal with:
      a) The consumer cannot take a data item from an empty buffer
      b) The producer cannot put a data item into a full buffer.
- We use condition variables to manage the synchronisation.


Producer
Consumer

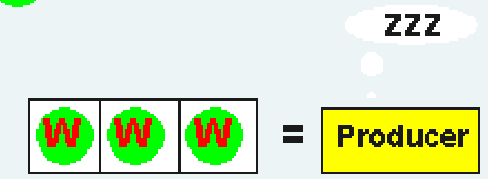

Producer Consumer Problem
P1
P2
P3
Data Buffer
C1
C2
C3
Producer Threads
Consumer Threads

Producer:

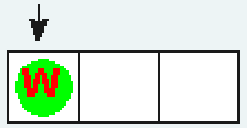
1. Make new widget



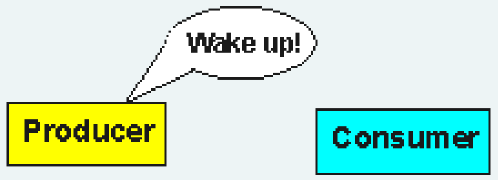
2. If buffer is full, go to sleep



3. Put widget in buffer



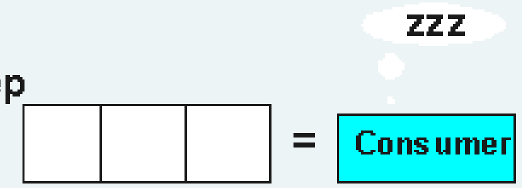
4. If buffer was empty, wake consumer



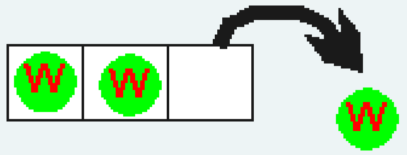
derby.ac.uk

Consumer:

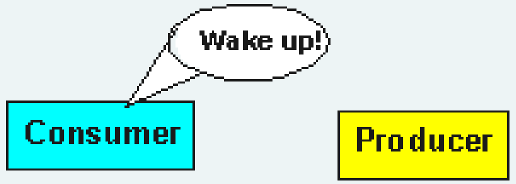
1. If buffer is empty, go to sleep



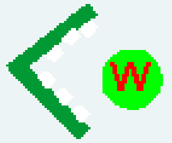
2. Take widget from buffer



3. If buffer was full, wake producer



4. Consume the Widget



Synchronisation and Consistency

- Have you ever had your bank take money out before it was meant to be taken out?
  - Did this mess with your plans?
  - Did this cause you to go into an overdraft?
  - This is a synchronisation issue
  - Consistency is assured as they are the sole controller of the information from our perspective (possibly not internally)

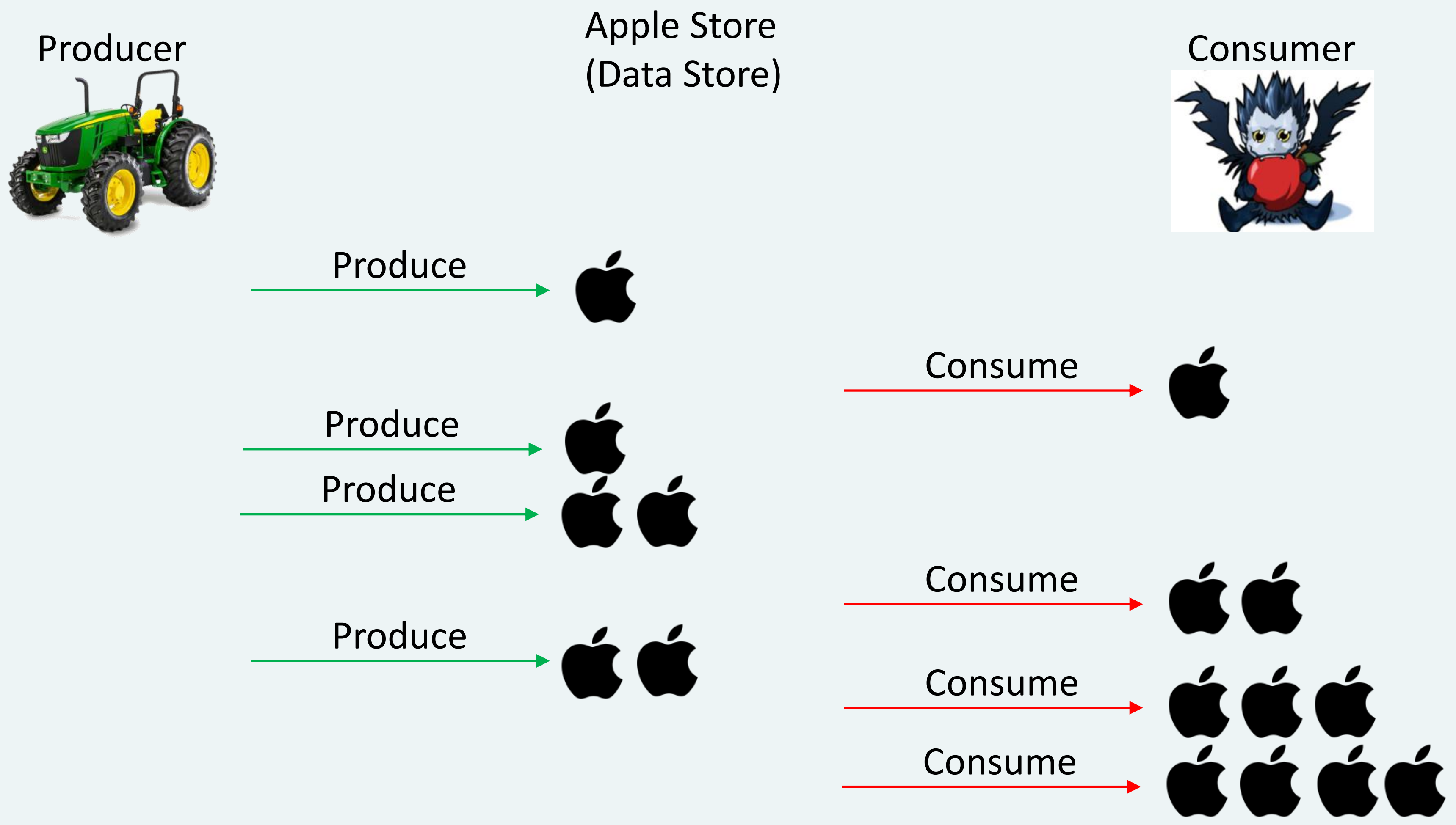
Race Conditions

- Imagine we have multiple threads interacting on a single data store
  - What happens if they change the order in which they interact?
  - What happens if they are only reading data and not writing?
  - What happens if reading also destroys data?
- Threads which update the same data inherently destroy the state that previously existed.
  - Consider the put vs post issue in HTTP commands

[Wikipedia: Producer Consumer]

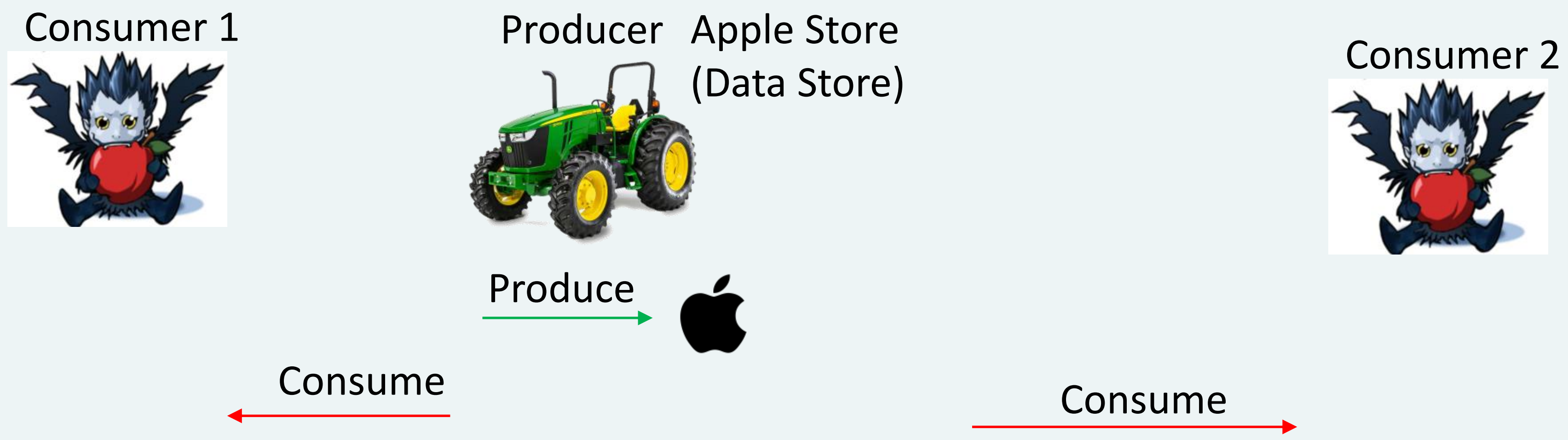
Tractor: John Deere
Shinigami: DogTagStudios.com

Race Conditions



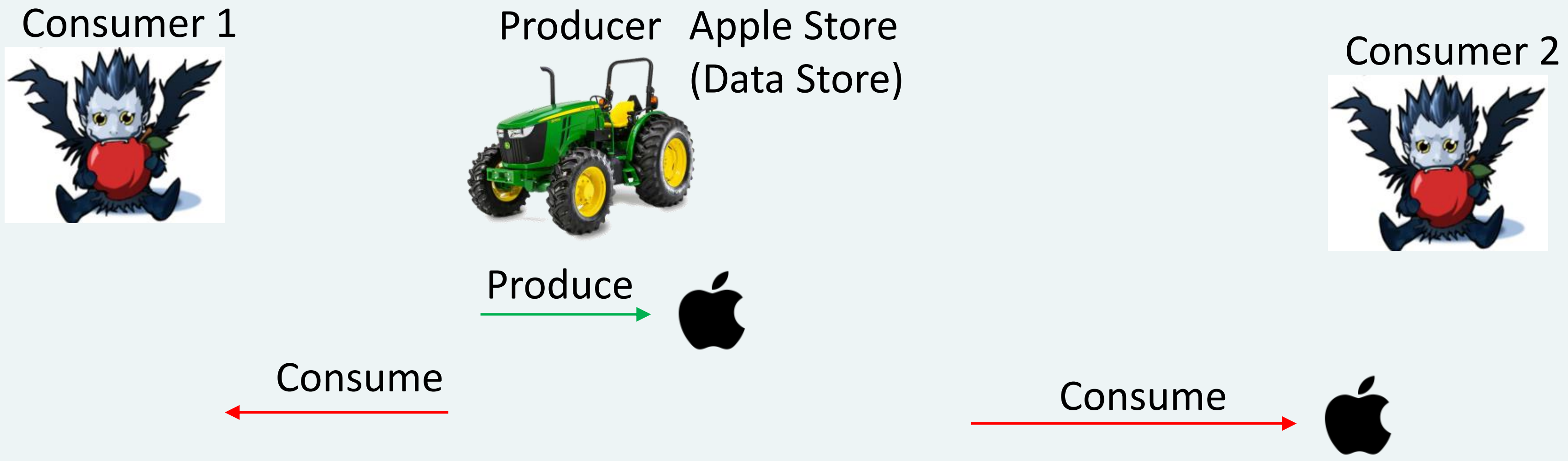
Tractor: John Deere
Shinigami: DogTagStudios.com

Race Conditions



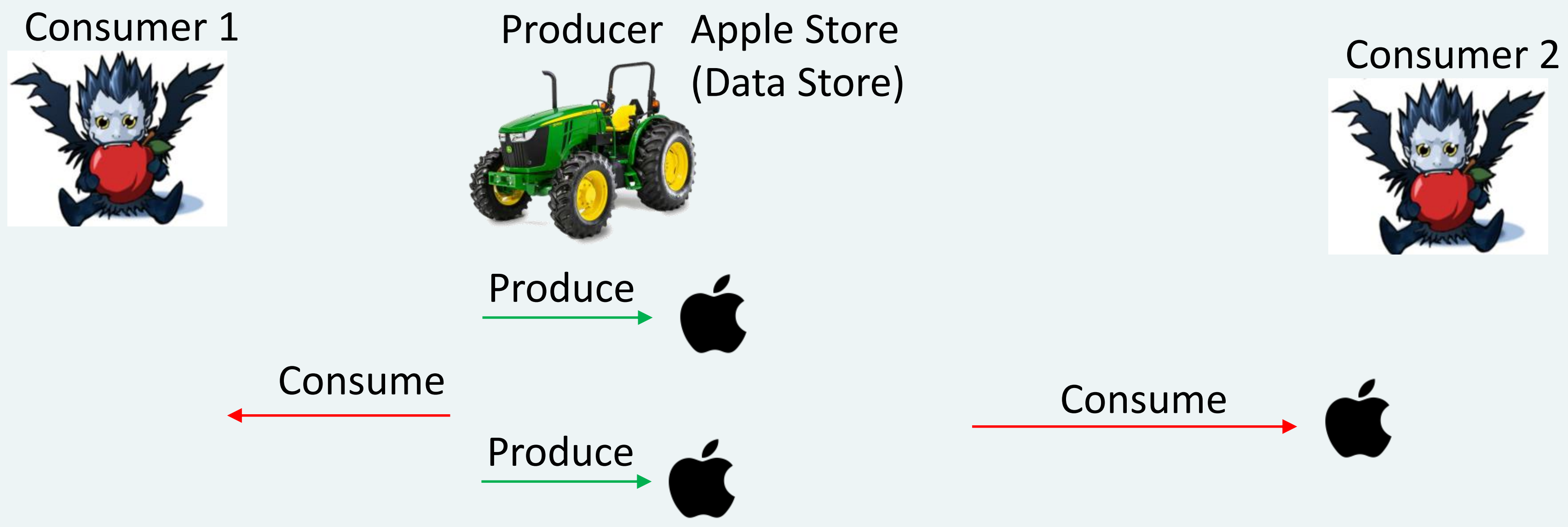
Tractor: John Deere
Shinigami: DogTagStudios.com

Race Conditions



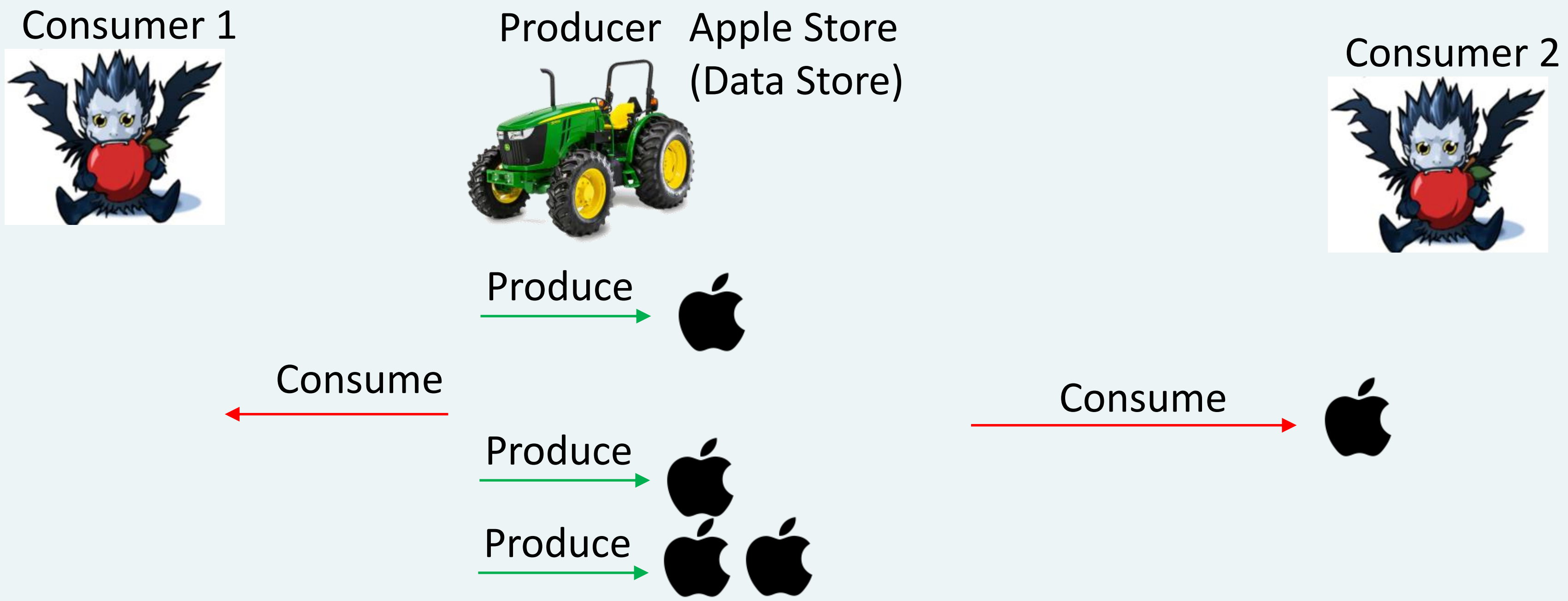
Tractor: John Deere
Shinigami: DogTagStudios.com

Race Conditions



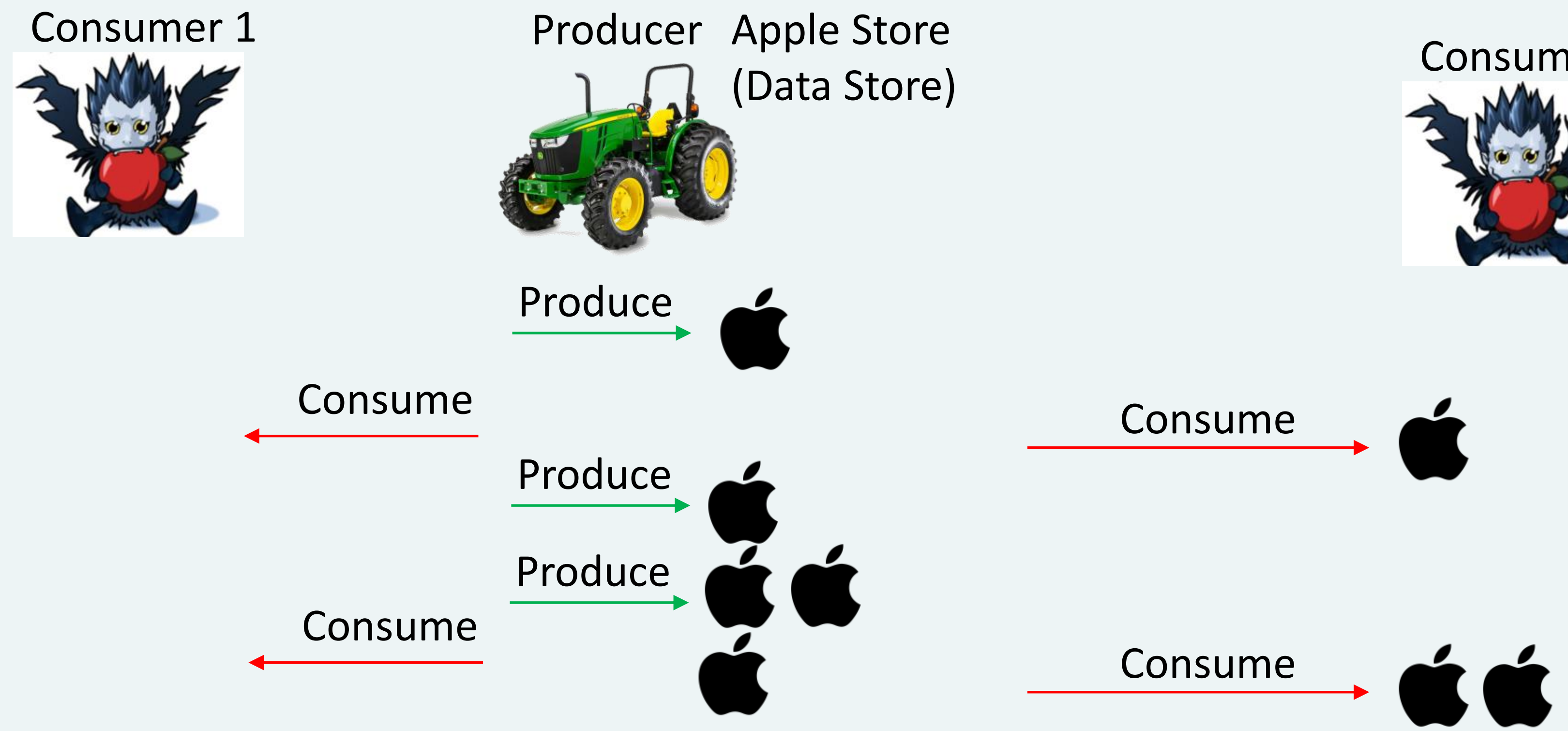
Tractor: John Deere
Shinigami: DogTagStudios.com

Race Conditions



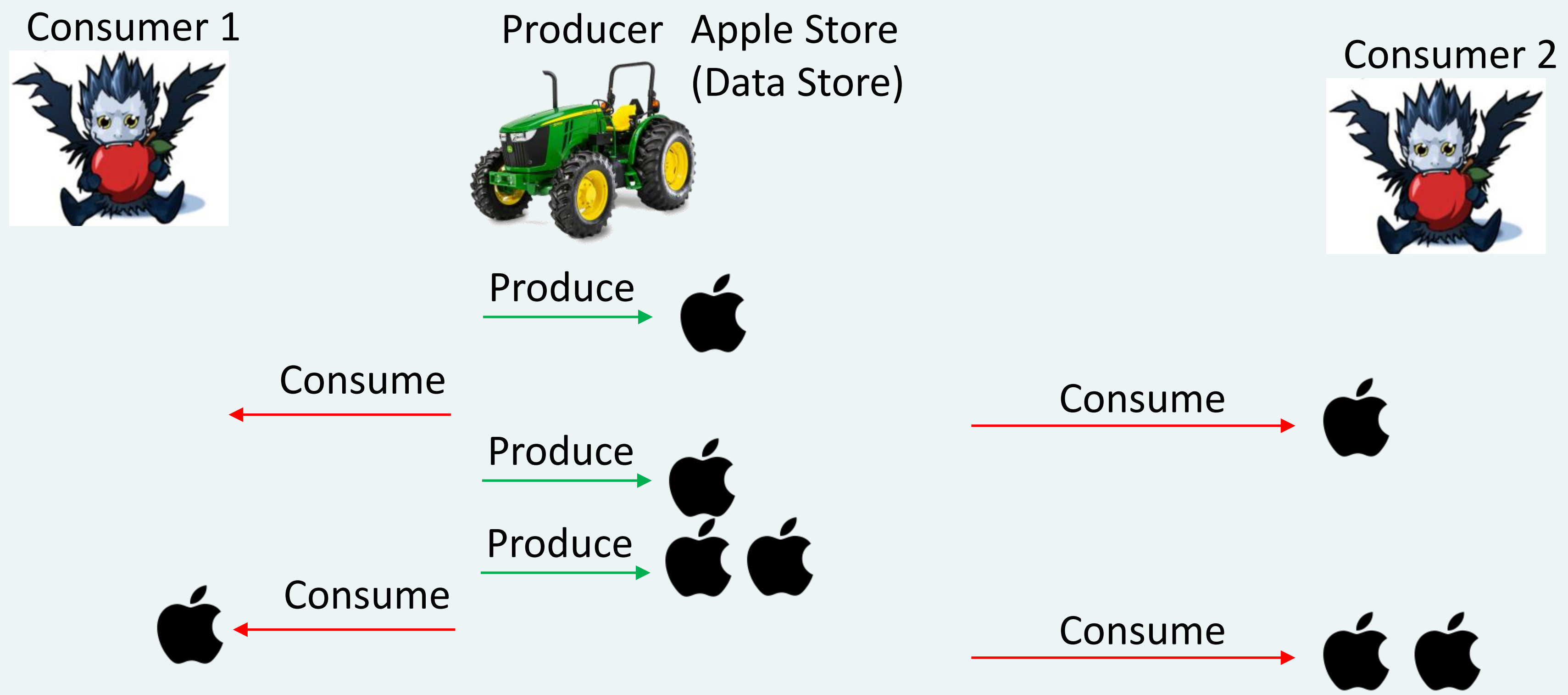
Tractor: John Deere
Shinigami: DogTagStudios.com

Race Conditions



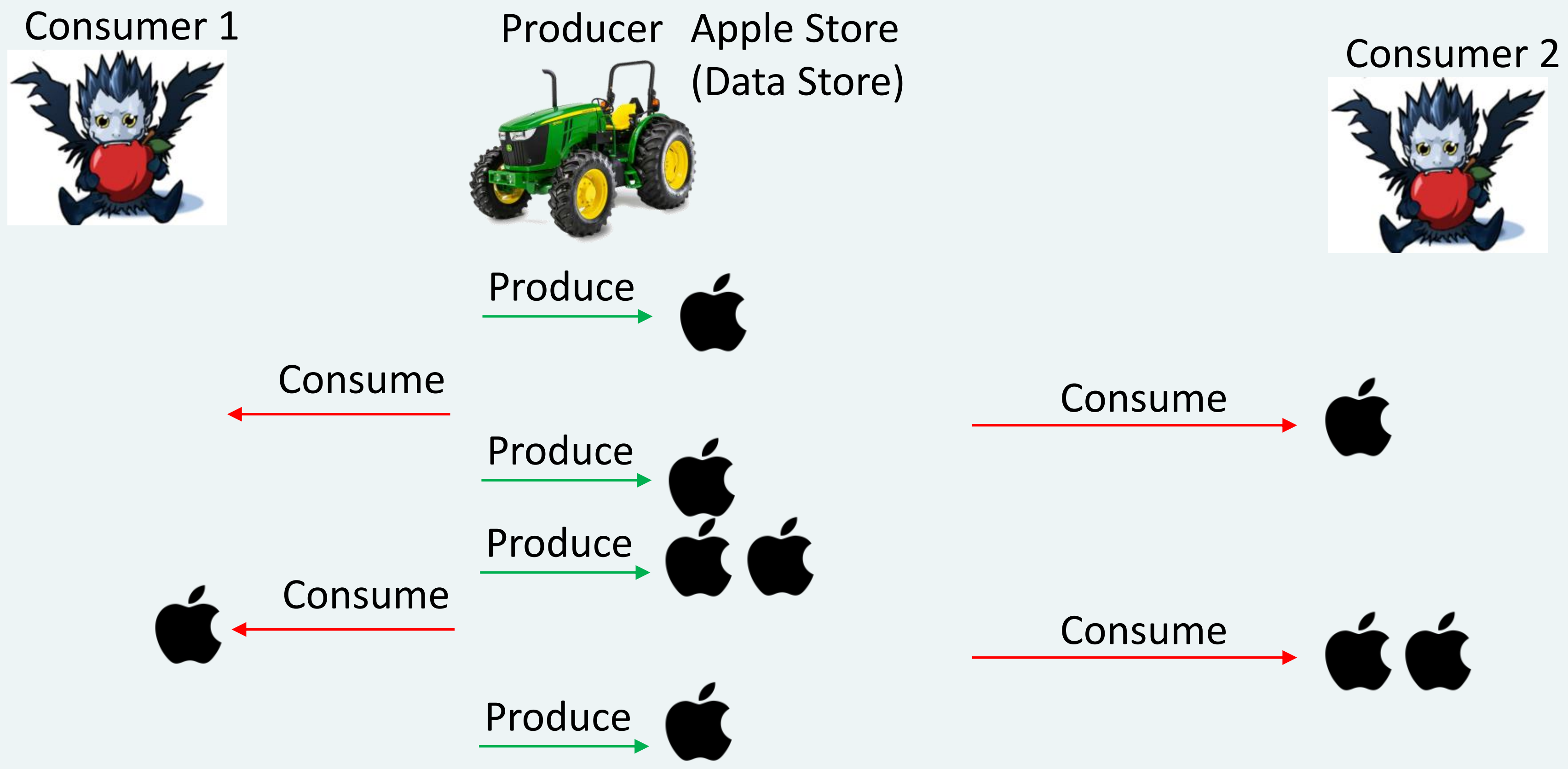
Tractor: John Deere
Shinigami: DogTagStudios.com

Race Conditions



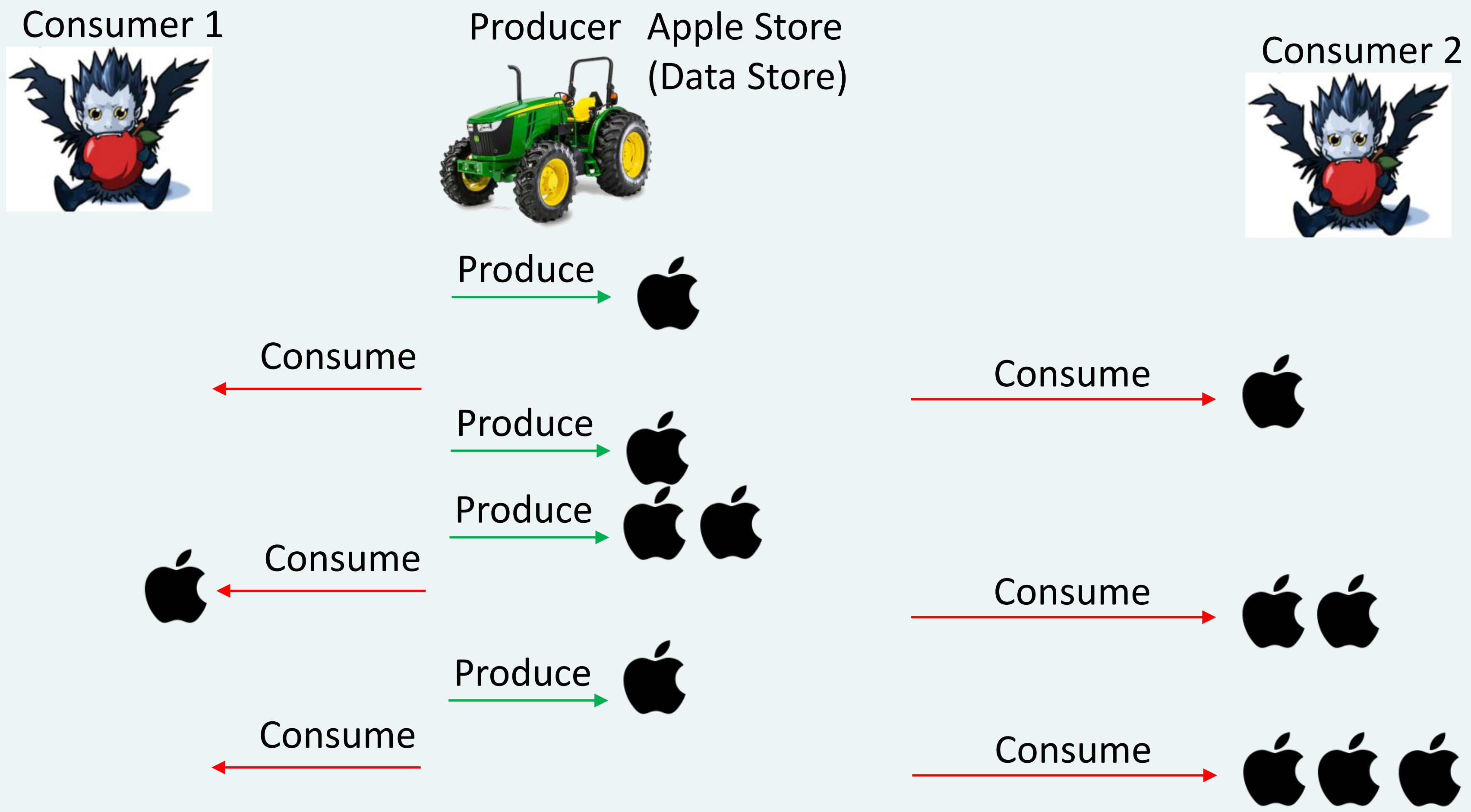
Tractor: John Deere
Shinigami: DogTagStudios.com

Race Conditions



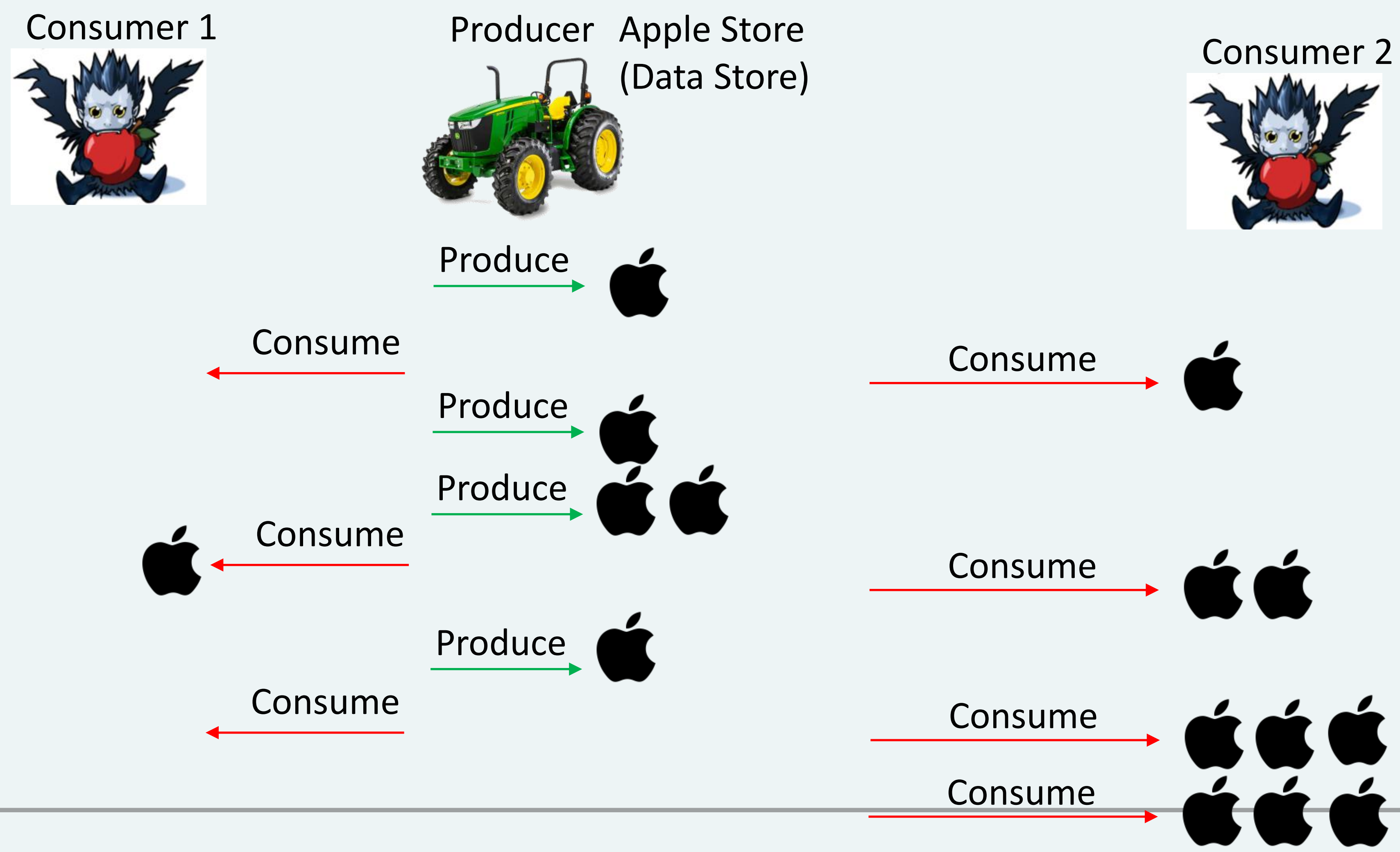
Tractor: John Deere
Shinigami: DogTagStudios.com

Race Conditions



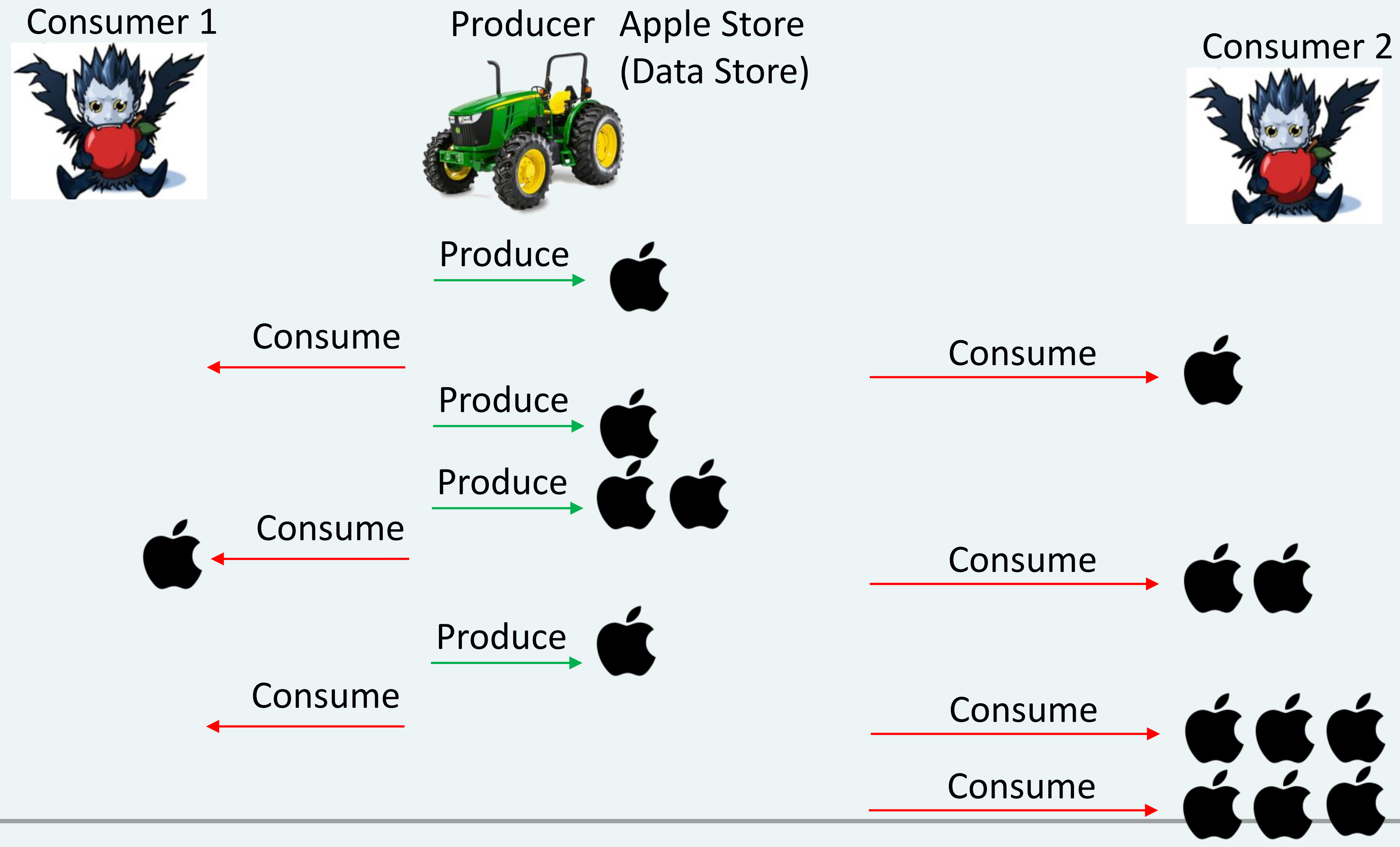
Tractor: John Deere
Shinigami: DogTagStudios.com

Race Conditions



Tractor: John Deere
Shinigami: DogTagStudios.com

Race Conditions

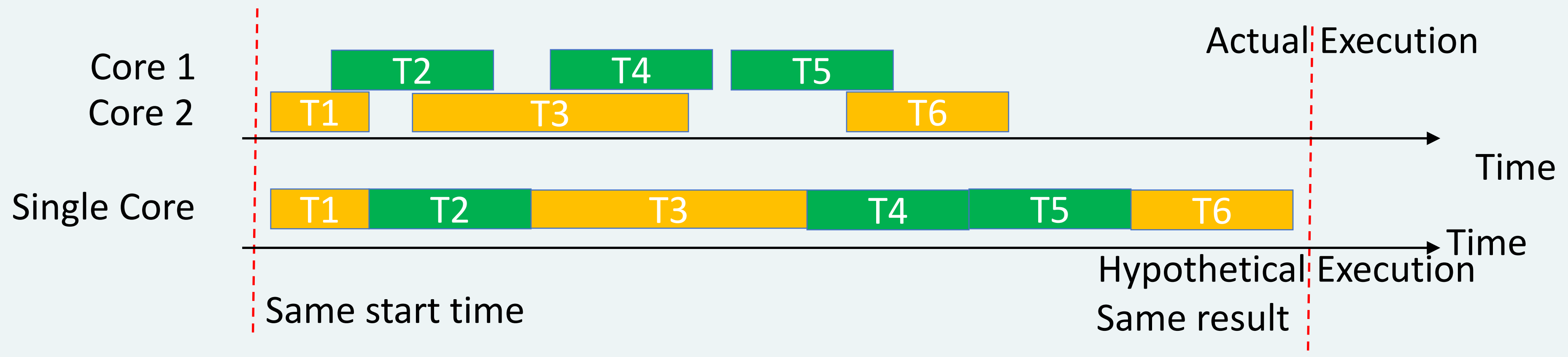


Consistency

- What should have happened to our consumers when they both tried to eat at the same time?
- How does this apply to a bank account
- Intuition says it shouldn't make a difference if the requests are concurrent or serial – the outcome should be idempotent
- How do we formalise this into a consistency model?

Sequential Consistency

- Sequential consistency:
  - The result of any execution is the same as if the operations of all the cores had been executed in some sequential order and the operations of each individual processor appear in this sequence in the order specified by the program



Other Consistency Models

- Strong Consistency
  - After an update completes all subsequent accesses will return the updated value
- Weak Consistency
  - After an update completes, subsequent accesses do not necessarily return the updated value, some condition must be satisfied first
- Eventual Consistency
  - Specific form of weak consistency: if no more updates are made to an object then eventually all reads will return the latest value
  - Variants: causal consistency, read your writes, monotonic writes
- How do we implement these?

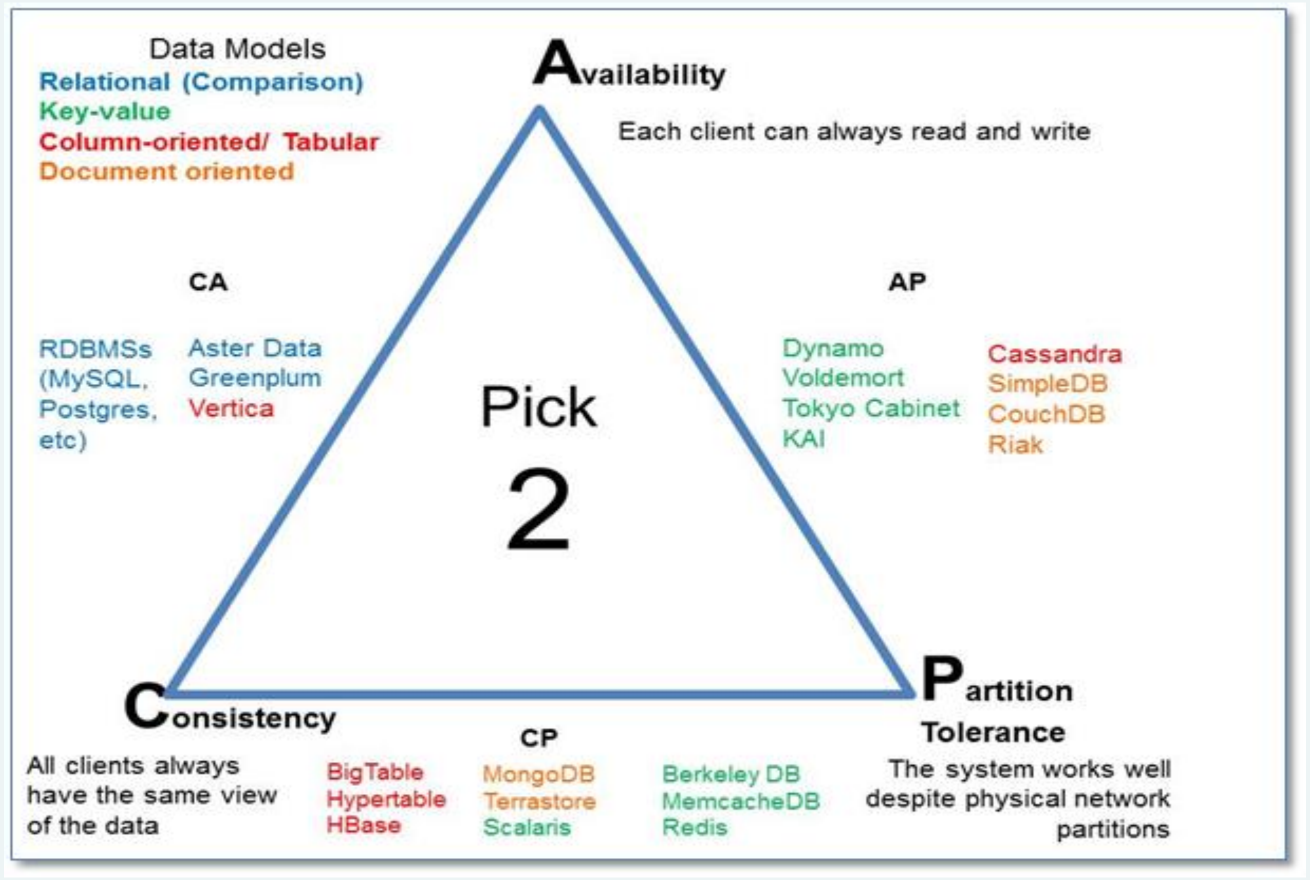
The CAP theorem

- What we want from a distributed system:
  - Consistency: All clients share the same view of the data, even in the presence of concurrent updates
  - Availability: All clients can access at least one replica of the data, even when faults occur
  - Partition-tolerance: Consistency and availability hold even when the network partitions

- Can we get all three?
  - CAP theorem: We can get <u>at most two</u> out of the three
    - Which ones should we choose for a given system?
  - Conjecture by Brewer; proven by Gilbert and Lynch

https://en.wikipedia.org/wiki/CAP_theorem

Enterprise System CAP Classification



Common CAP choices

- Example #1: Consistency & Partition tolerance
  - Many replicas + consensus protocol
  - Do not accept new write requests during partitions
  - Certain functions may become unavailable
- Example #2: Availability & Partition tolerance
  - Many replicas + relaxed consistency
  - Continue accepting write requests
  - Clients may see inconsistent state during partitions

Relaxed consistency: ACID vs. BASE

- Classical database systems: ACID semantics
  - Atomicity
  - Consistency
  - Isolation
  - Durability

- Modern Internet systems: BASE semantics
  - Basic Availability
  - Soft-state
  - Eventual consistentency

Deadlock

- Consider:
  - If locks can be nested
  - If you can acquire locks out of order
  - If locks are shared between sections
  - X(A(CS)-> B(CS)), Y(B(CS)-> A(CS))
  - Both of these might make sense independently
  - Together they result in two processes waiting indefinitely for a key to their lock

Deadlock Solutions

- Lock manager:
  - One centralised location that manages the acquisition of locks
  - What are the consequences for scalability
- Resource Hierarchy:
  - Require acquisition of resources in order
  - Is this problematic?

UNIVERSITY OF DERBY

Cloud

Networks and Security

What is Cloud Computing

*"**Cloud computing** is a model for enabling ubiquitous, convenient, **on-demand** network access to a **shared pool** of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be **rapidly provisioned and released** with **minimal management effort** or service provider interaction."*

– NIST (The NIST Definition of Cloud Computing, September 2011)

- "Cloud computing" is a modern interpretation of traditional large scale distributed computing systems:
  - Someone else owns the hardware and pays for its associated costs
  - You may or may not own or control the software you run on this hardware
  - You own the data that is being processed (subject to local laws)

Why Provide Cloud Computing?

- Data centre utilisation is often poor:
  - Must be over-provisioned
  - Purchases infrastructure in large blocks to meet user needs
  - Systems are often idle
  - Difficult to sub-divide processing units

- Providing “cloud computing”:
  - Provide virtualised or real computing resources to other users
  - Over-provisioning can be made use of
  - Idle times can be used productively
  - Virtualisation allows subdivision of processing units

Deployment Models

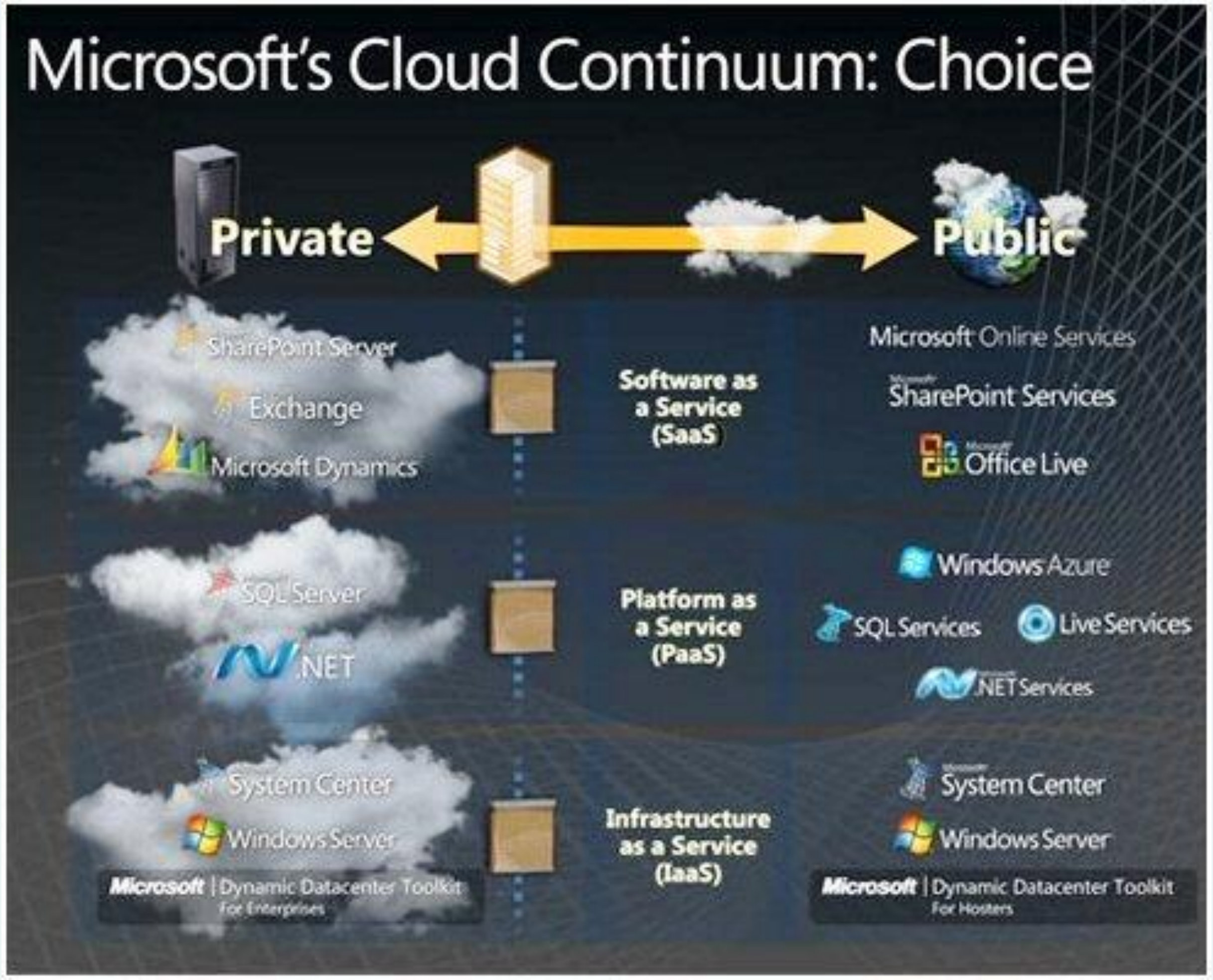
Image ©2013 Microsoft Corp.

- Cloud computing has three major deployment models:
  - Private cloud
  - Public cloud
  - Hybrid cloud

Private Cloud:
Internally hosted by the organisation. Uses capability to share resources amongst internal users.

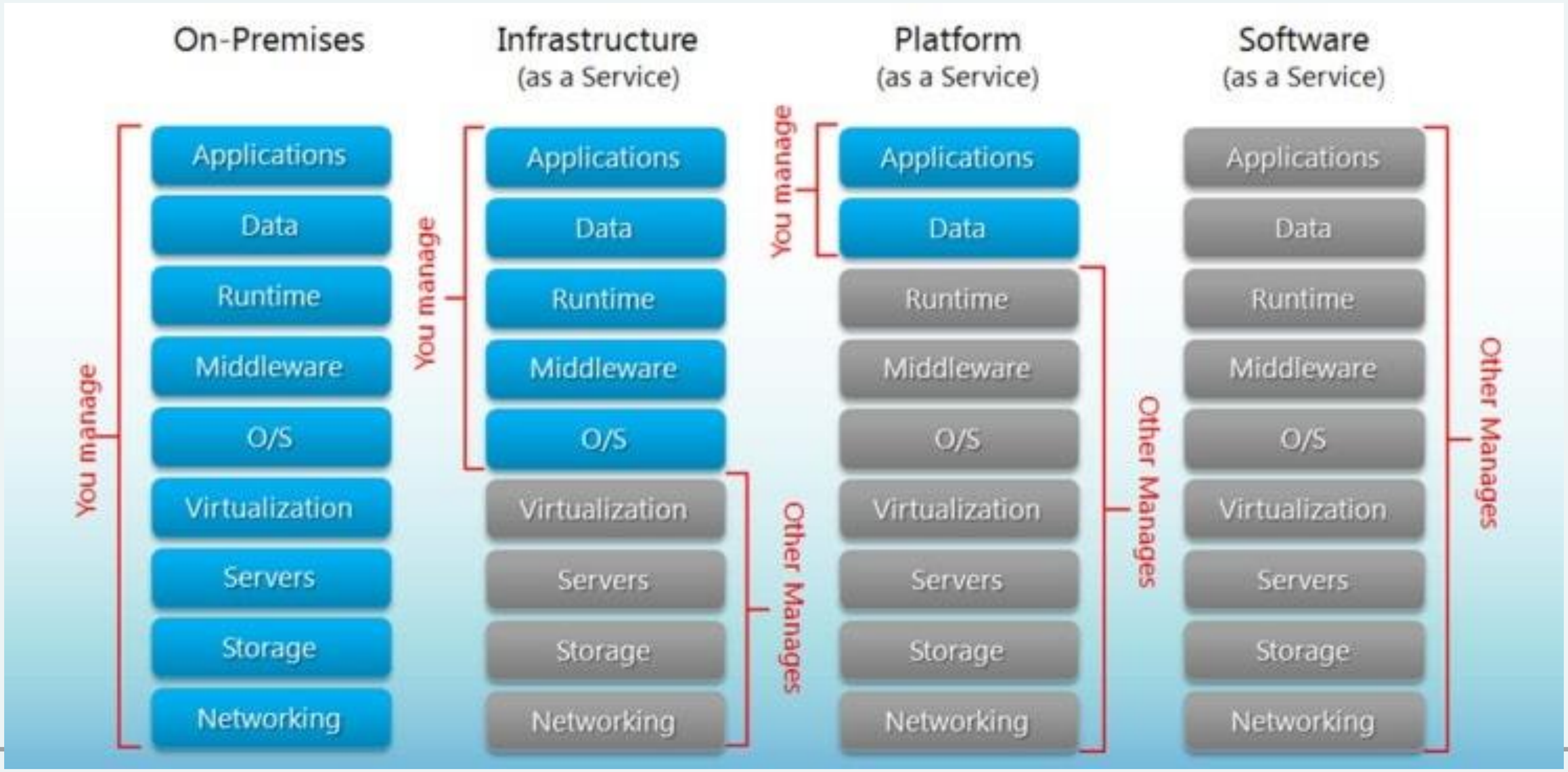
Public Cloud:
Data-centre hosted cloud. Shares resources amongst external users. Often deployed to utilise 'surplus' capacity.

Hybrid Cloud:
Combination of private and public cloud infrastructures


Microsoft's Cloud Continuum: Choice
Private
Public
SharePoint Server
Exchange
Microsoft Dynamics
Software as a Service (SaaS)
Microsoft Online Services
SharePoint Services
Office Live
SQL Server
.NET
Platform as a Service (PaaS)
Windows Azure
SQL Services
Live Services
.NET Services
System Center
Windows Server
Microsoft | Dynamic Datacenter Toolkit For Enterprises
Infrastructure as a Service (IaaS)
System Center
Windows Server
Microsoft | Dynamic Datacenter Toolkit For Hosters

In Short

https://www.bigcommerce.co.uk/blog/saas-vs-paas-vs-iaas/#the-key-differences-between-on-premise-saas-paas-iaas



Is the cloud good for everything?

- No.
  - Sometimes it is problematic, e.g., because of auditability requirements

  - Example: Processing medical records
    - HIPAA (Health Insurance Portability and Accountability Act) privacy and security rule
  - Example: Processing financial information
    - Sarbanes-Oxley act

Privacy Issues

- **Trust**
- Architecture
- Identity Management
- Software Isolation
- Data Protection
- Availability

- Insider Access
- Composite Services
- Visibility
- Risk Management

Privacy Issues

- Trust
- **Architecture**
- Identity Management
- Software Isolation
- Data Protection
- Availability

- Attack Surface
- Virtual Network Protection
- Ancillary Information
- Client Side Protection
- Server Side Protection

Privacy Issues

- Trust
- Architecture
- **Identity Management**
- Software Isolation
- Data Protection
- Availability

➢ Authentication

➢ Access Control

Privacy Issues

- Trust
- Architecture
- Identity Management
- **Software Isolation**
- Data Protection
- Availability

➢ Hypervisor Complexity

➢ Attack Vectors

Privacy Issues

- Trust
- Architecture
- Identity Management
- Software Isolation
- **Data Protection**
- Availability

- ➢ Data Isolation
- ➢ Data Sanitization
- ➢ Data Location

Privacy Issues

- Trust
- Architecture
- Identity Management
- Software Isolation
- Data Protection
- **Availability**

➢ Temporary Outages

➢ Prolonged and Permanent Outages

➢ Denial of Service

➢ Value Concentration

Legal issues

- UK GDPR – 2018
- US CLOUD act – 2018
- EUGDPR - 2016
- French Intelligence Act - 2015
- US Patriot Act - 2001
- UK Regulation of Investigatory Powers Act - 2000

- Some reading:
  - https://www.isaca.org/Groups/Professional-English/cloud-computing/GroupDocuments/DLA_Cloud%20computing%20legal%20issues.pdf
  - https://www.twobirds.com/~/media/PDFs/Expertise/IT/Cloud%20computing%20law%20interactive.pdf
  - https://www.researchgate.net/publication/328243752_Cloud_Computing_Legal_and_Security_Issues
  - https://www.researchgate.net/publication/226801981_Privacy_Regulations_for_Cloud_Computing_Compliance_and_Implementation_in_Theory_and_Practice
  - https://technology.findlaw.com/networking-and-storage/cloud-computing-and-the-law-the-basics.html

Legal issues

- French government officials banned from use of Blackberry devices
- US Patriot Act 2001
- UK Regulation of Investigatory Powers Act 2000

Encryption

Legal issues

- French government officials banned from use of Blackberry devices
- US Patriot Act 2001
- UK Regulation of Investiga[illegible] Po[illegible]ct 2000

~~Encryption~~

- ➢ Data needs to be decrypted at the point of processing
- ➢ If this is outsourced, encryption is irrelevant

UNIVERSITY OF DERBY

University of Derby, Kedleston Road, Derby, DE22 1GB
T +44 (0)1332 591044 **E** opendays@derby.ac.uk

THANK YOU

derby.ac.uk

Sensitivity: Internal