
ソフトウェア・クラウド開発プロジェクト実践 I

Practices for Software and Cloud Development Project I

Soramichi Akiyama

穂山 空道

akiyama@ci.i.u-tokyo.ac.jp

Why “Study” Cloud Computing?

- You need knowledge of cloud computing to ...
 - Properly assess if you should use clouds or not
 - Properly select which type of clouds you should use
 - Properly develop / debug your software on top of clouds
- Why: Almost any software development requires using clouds
 - Examples: **web applications, business process applications** (e.g., finance), **games, web browsers, communication applications, ...**

Today's Topics

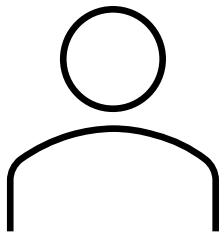
- Cloud computing overview
 - What is it?
 - Why is it useful?
 - How can it be categorized and how are they different?

---- 5 min break  ----

- Cloud computing internals
 - How are they implemented?
 - Keywords: virtual machines, OS-level virtualization (containers), network virtualization

What is “Cloud” Computing? (Very Brief Overview)

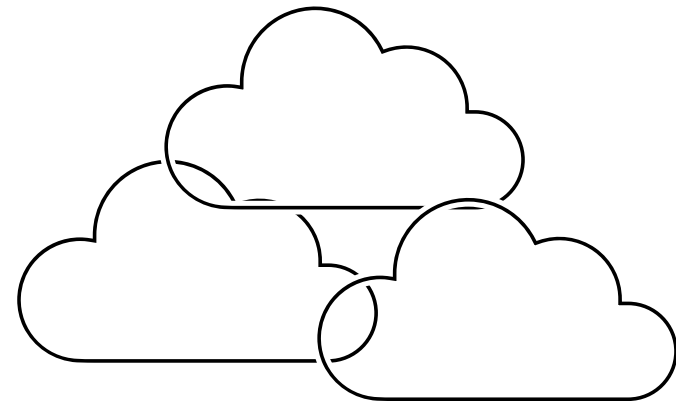
- A relatively new computing paradigm, where
 - A user can **borrow/rent** virtually infinite computing recourse **from somewhere behind the “cloud”**
 - The provided resource is **accessible through network**
 - The user is **not bothered to manage the resource** they use
 - The user is **charged depending only on the amount of resource** they use (“pay-per-use”)



Let me use **32** machines with
4 GPUs each for **12** hours



Here they are. We charge you
 $32 * 4 * 12 * 0.02 = \$ 30.72$



“cloud”

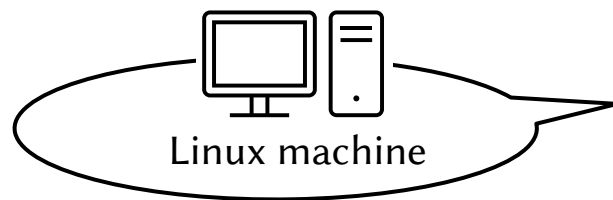
Cloud Computing is Everywhere

- Cloud computing is widely used nowadays
- Examples:
 - **Amazon**: everything hosted on [Amazon Web Services](#) (of course!!)
 - **TOYOTA**: webpage distributed through [Akamai](#)
 - **Mercari**: item images hosted by [SAKURA Internet](#)
 - **UTokyo**: the livestream of the graduation ceremony in FY2020 was hosted on a cloud (Note: not the YouTube version, but the self-managed one)
- Quiz: Why did UTokyo use a cloud?
 - Hopefully you'll be able to answer it with confidence after the class



Why is Cloud Important/Useful? (1/4)

- A short time ago in an apartment so, so close.... (inspired by [1])
 - My website was ACTUALLY hosted **in-house** on a small Linux machine
- We will walk through **importance of cloud computing** by elaborating why the design of my website back then was **not practically good**



My apartment
back then



100 m?



Photo cited from [2]

[1] https://en.wikipedia.org/wiki/Star_Wars_opening_crawl

[2] <https://ja.wikipedia.org/wiki/%E5%AE%89%E7%94%B0%E8%AC%9B%E5%A0%82>

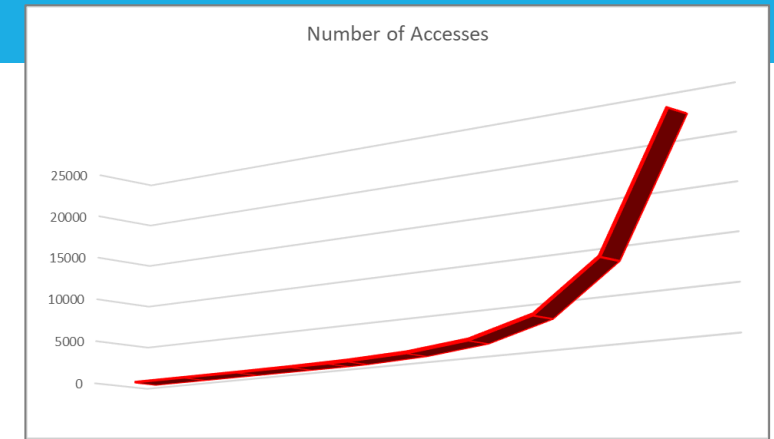
Why is Cloud Important/Useful? (2/4)

- What if the linux machine breaks?
 - Hardware (e.g., HDD, power supply) may fail over time
 - The input voltage may surge due to a crash of thunder
 - I may accidentally kick it
 - Computer virus on my other machines may attack it
- Then, I must buy a new one and re-configure it...
 - It costs **money** and **human-labor**
- **Cloud manages computing resource on my behalf**
 - **Total cost of owning (TCO)** can be largely reduced (Note: “cost” includes human labor as well)
 - Especially true for a large deployment such as big cooperate web services



Why is Cloud Important/Useful? (3/4)

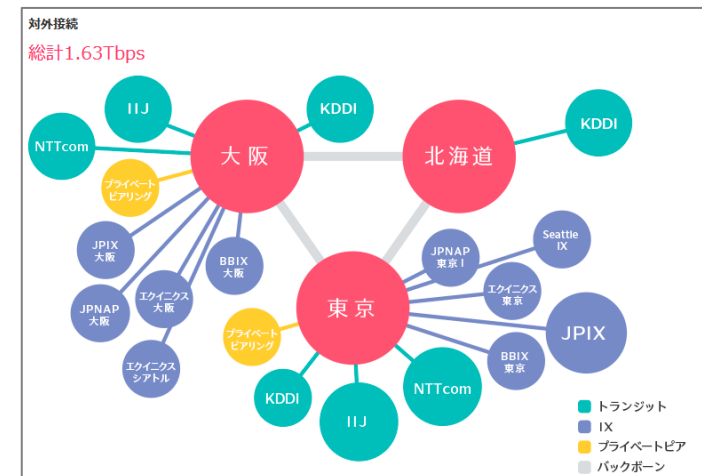
- What if it gets 1,000X more accesses all the sudden?
 - My paper might get accepted by Nature
 - My name might be broadcasted on NHK for something bad
- My poor machine cannot handle them
 - Visitors become upset and will never come back
 - X % of visitors become upset after Y seconds [No reliable source of concrete numbers found]
- **Cloud can elastically provide more or less resource as you wish**
 - Can handle both sudden access spikes and gradual up-scaling
 - Scaling down is also easy (e.g., after the # of users decrease)
 - In contrast, a powerful machine cannot be easily replaced by a smaller one once bought



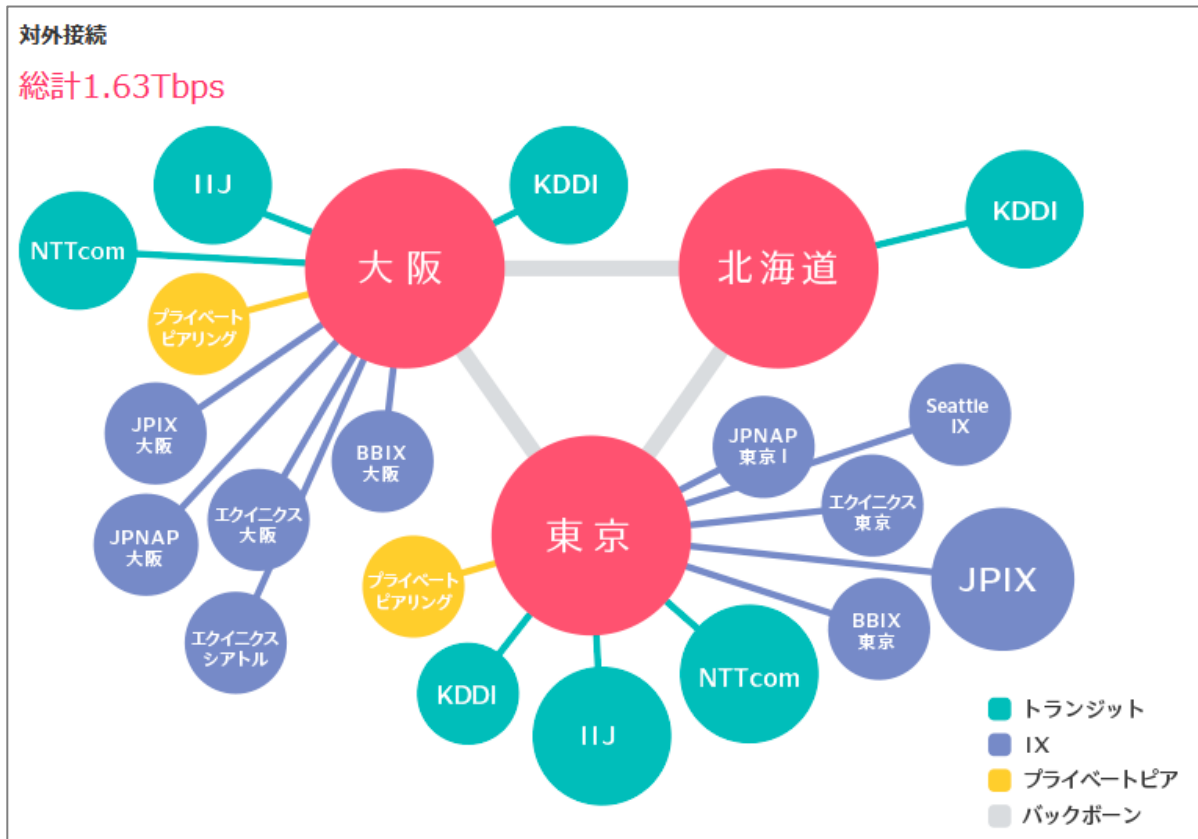
An Anti-example of a good graph

Why is Cloud Important/Useful? (4/4)

- What if my network gets too congested (輻輳)?
 - A neighbor may start watching Netflix all day
 - My Internet Service Provider (ISP) may host too many users
- Preparing strong enough network resource beforehand is even more difficult than preparing a powerful enough machine
 - Congestion is affected by neighbors and/or other users of the same ISP
- **Cloud is connected by broad network**
 - Many ISPs have **direct connections to cloud providers**
 - SAKURA Internet has **1.63 Tbps bandwidth to outside world**



[Aside] Network Connectivity of SAKURA Internet



Backbone:

- Internal network that connects the major datacenters of them like a backbone (背骨)

Private Peering:

- 1 vs. 1 connection with other autonomous systems (AS; ISPs, cloud providers, etc.).
- Actual connectivity info is a business secret

IX (Internet Exchange):


- Connection with other ASes via an internet exchange (it's like a huge switch to connect different ASes).


Transit:

- “Upstream” ISPs to provide connectivity to the internet
- If I connect to SAKURA Internet from home, the packets probably go through a transit

More formally

unilaterally *adverb*

 /juːnɪˈlætrəli/

 /juːnɪˈlætrəli/

★ by one person, group or country involved in a situation without the agreement of the others

- Essential characteristics of Cloud by NIST [3]
 1. **On-demand self-service:** A consumer can unilaterally provision computing capabilities ... without requiring human interaction
 2. **Broad network access:** Capabilities are available over the network
 3. **Resource pooling:** The provider's computing resources are pooled to serve multiple consumers using a multi-tenant model
 4. **Rapid elasticity:** Capabilities can be elastically provisioned and released ... to scale rapidly outward and inward
 5. **Measured service:** Resource usage can be monitored, controlled, and reported, providing transparency for both the provider and consumer of the utilized service

Cloud Computing Models: Basics

■ Terminology: XXX as a Service

- XXX is provided to users as a service 🤔
- Users do not own XXX nor manage it but **acquire rights to use XXX if they pay**
- Subscription (サブスク) in today's term?
 - Note: “subscription” originally meant to have fixed-length (e.g., per-month) contracts of something, but サブスク does not necessarily mean that (?)

「サブスクリプション」は、定額で使い放題という意味ではいわゆる「定額制」と同様だが、基本的には「買い取り」方式と対比される概念である。従来は販売されていた（購入し所有した上で使ってた）ものを、もの自体を販売せずに一定期間に限り自由に使える権利を販売することがサブスクリプションである。

<https://www.weblio.jp/content/%E3%82%B5%E3%83%96%E3%82%B9%E3%82%AF>

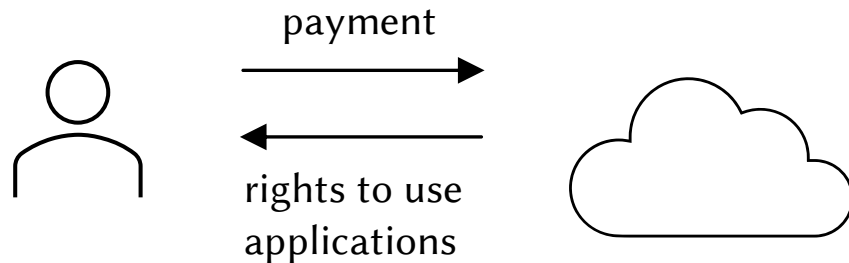
■ Examples:

- **Netflix, Amazon Prime Video, Hulu, etc.:** You **buy rights to watch videos** without having physical copies of them (e.g., Blu-Ray Discs)
- **Times Car Share:** You **buy rights to use cars** without owing nor maintaining them

Cloud Computing Models (1/3)

■ Software as a Service (SaaS)

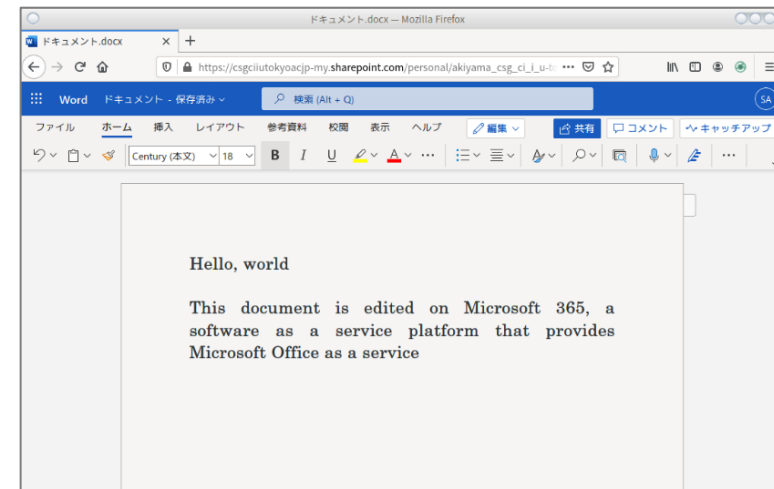
- Software (applications) are provided to users as a service
- The software is widely accessible through network
- The users **do not manage the underlying OS / hardware / etc.** to host the software



■ Examples:

- **Microsoft 365** (a.k.a. Office 365): Microsoft Office as a service
- **Gmail** (including @g.ecc.u-tokyo.ac.jp): Email software as a service

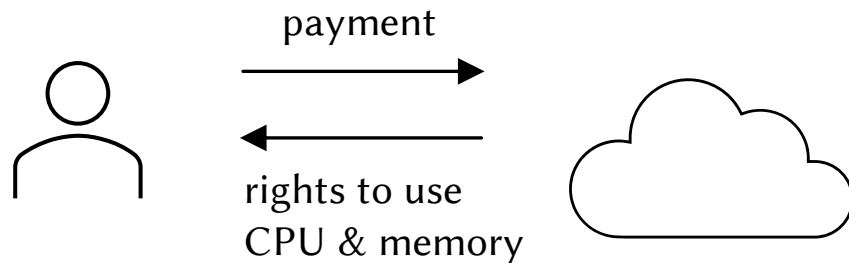
Microsoft Word hosted on a cloud,
accessed over a network using a web browser



Cloud Computing Models (2/3)

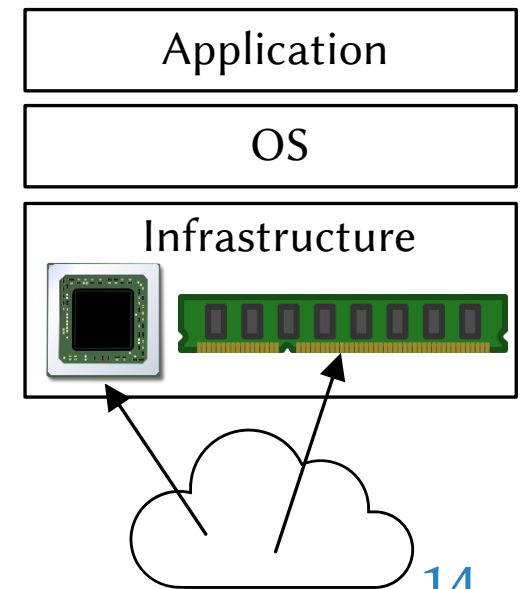
■ Infrastructure as a Service (IaaS)

- Computing infrastructure (processor, memory, storage, network) are provided as a service
- The infrastructures are widely accessible through network
- The users **do not manage the actual hardware** to host the infrastructure



Managed by the
user (user can do
whatever they want)

Managed by the
cloud provider



■ Example:

- **Amazon EC2:** provides VMs and containers as a service
- **Google Compute Engine:** (ditto)

Cloud Computing Models (3/3)

■ Platform as a Service (PaaS)

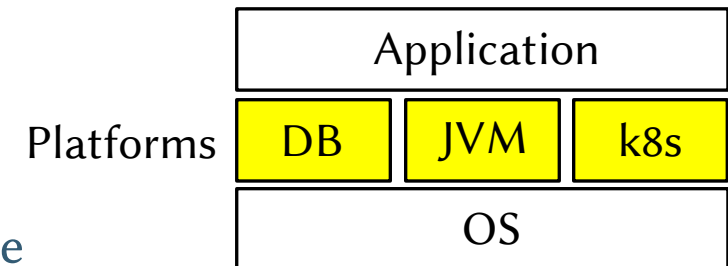
- Platforms to deploy applications onto are provided as a service
- The platforms are widely accessible through network
- The users **do not manage the underlying hardware / OS / etc.** to host the platform

■ What is a “platform” ?? 🤔

- A base that something can physically / logically stand on top of
- Enterprise apps are often not written “from scratch”, but they rely on software platforms such as **database engines (e.g., MySQL)**, **language runtimes (e.g., JVM)**, **deployment manager (e.g., Kubernetes)**

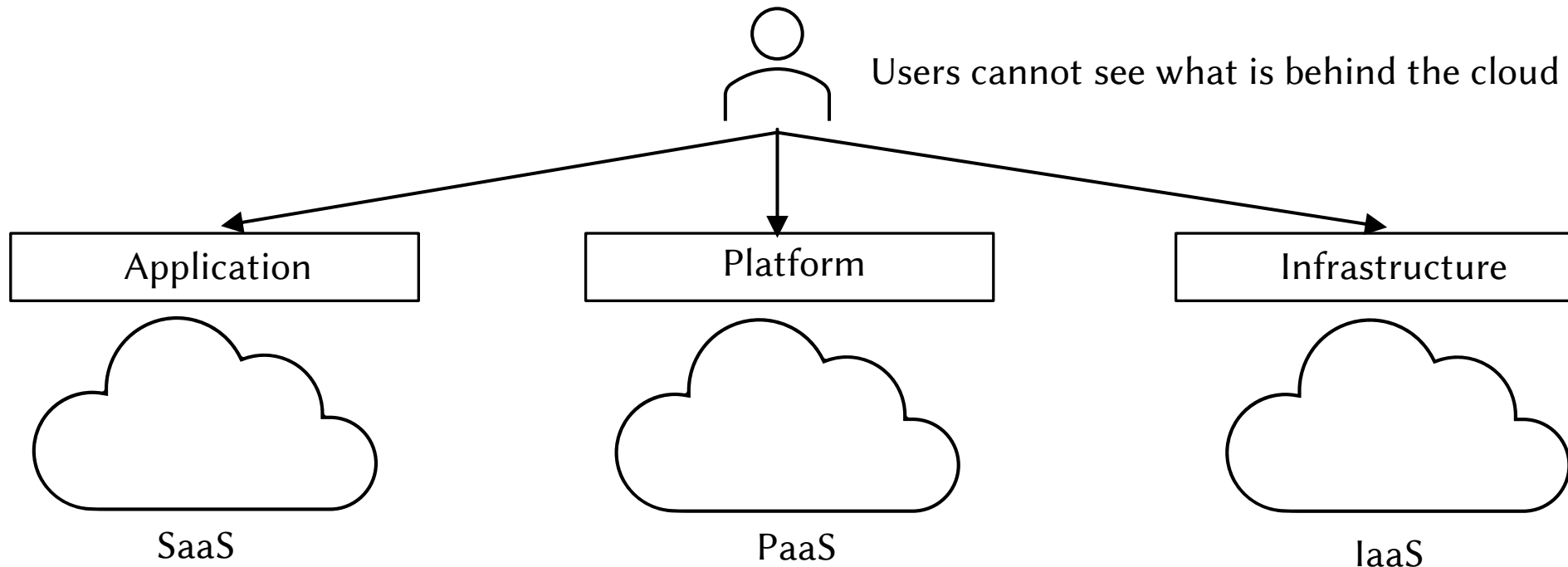
■ Examples:

- **Amazon RDS:** provide relational database as a service
- **Azure Kubernetes Service:** provide Kubernetes as a service



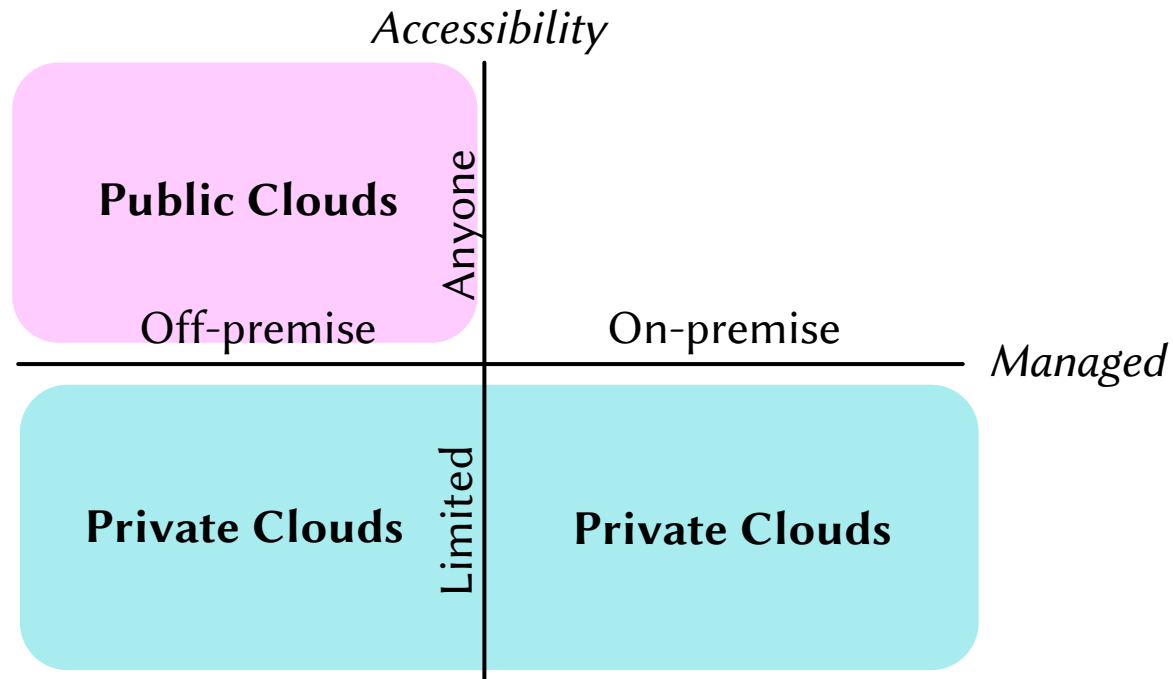
Cloud Computing Models: Summary

- Cloud computing models are categorized by the “layer” on which the computing resource they provide resides
 - From upper layer to lower: SaaS, PaaS, IaaS
- Users do not need to care about the layers below the provided resource



Cloud Deployment Models

- Two perspectives to categorize deployment models
 - Management of actual infrastructure: **on-premise** (the infrastructure is managed by the actual users), or **off-premise** (the infrastructure is managed by a third party)
 - Accessibility: **Anyone** can access and use the cloud, or only **members of a single organization** can access and use the cloud



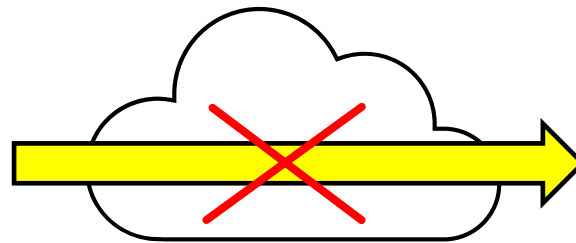
- Types of clouds categorized by the deployment model
- **Public clouds:** accessed by anyone && off-premise
 - **Private clouds:** accessed by one organization && (on-premise || off-premise)

Public Clouds

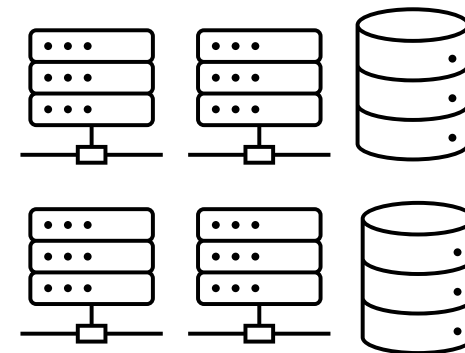
- Clouds that **anyone can access and use**
 - Managed **off-premise**
 - Is it possible to manage a public cloud on-premise?? (i.e., anyone can use the cloud and the infrastructure is managed by each user??)
- Examples:
 - **Amazon Web Services, Microsoft Azure, Google Cloud, SAKURA Cloud, ...**



User organization
(Photo cited from [2])



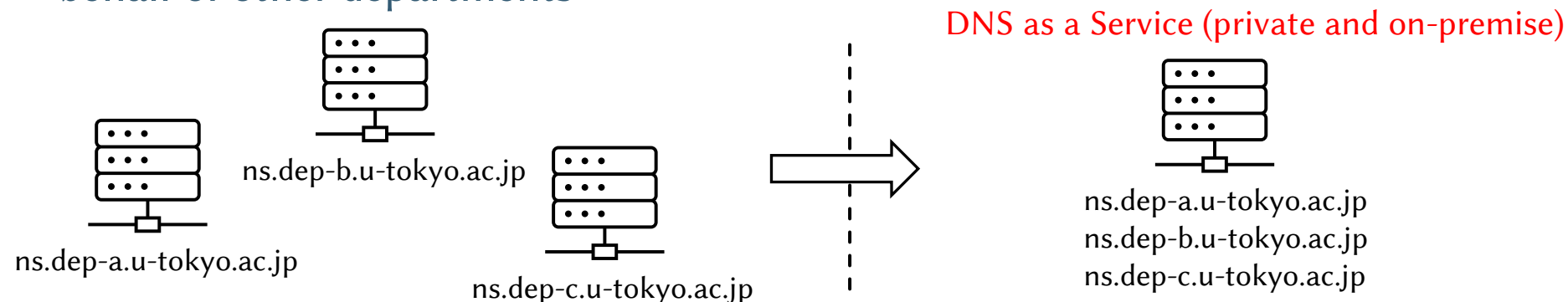
No management,
but no control either



An Amazon datacenter
(Note: there are a lot of them!)

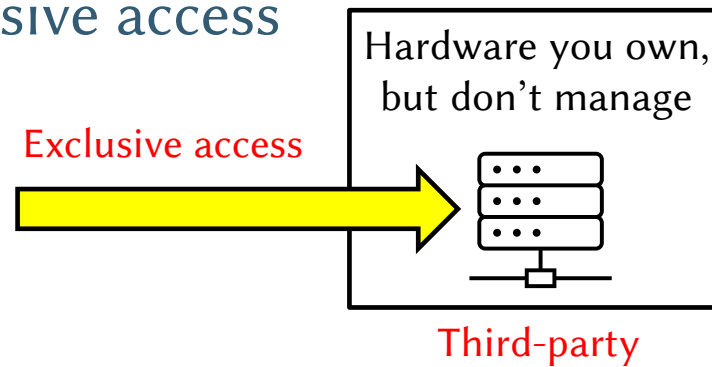
Private Clouds (1/2)

- Clouds that **only members of one organization can access and use**
 - Managed **either on-premise or off-premise**
- Wait, why on earth do we host a cloud on-premise??? 🤔
 - It sounds breaking the principles of cloud computing
- A big organization can “pack” machines to an on-premise private cloud
 - It is still a “cloud” from end-user’s perspectives and can **reduce TCO**
 - Example: UTokyo Information Technology Center (情報基盤センター) hosts DNS servers on behalf of other departments



Private Clouds (2/2)

- Private and off-premise clouds
 - Accessed only by members of one organization, but the actual infrastructures are managed by a third-party (i.e., different organization)
- Buy your own hardware, ask a third-party to manage them (i.e., co-location), and allow members of your organization exclusive access



- Merits:
 - **Predictable and controllable performance:** Public clouds give you virtual CPUs and there is no notion of performance guarantee (e.g., average floating-point operations / seconds / price)
 - **Better TCO in some cases:** Once the scale exceeds a certain point, it is cheaper to buy designated hardware and paying for management than using public clouds

Security: a different aspect

- Should we choose which type of cloud to use **only by the cost**?
 - No. What about the **security**?
- Examples of sensitive data that might affect the placement
 - **Privacy information of customers**: You might want to place them in a private cloud (either on-premise or off-premise)
 - **Classified information of a country**: You might want to place them in a private cloud physically located inside your country
- **Even employees of prestigious companies cannot be trusted** in some scenarios



Cybersecurity

Capital One Says Breach Hit 100 Million Individuals in U.S.

By [Christian Berthelsen](#), [Matt Day](#), and [William Turton](#)

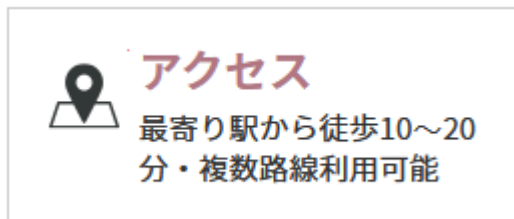
2019年7月30日 7:11 JST Updated on 2019年7月30日 22:50 JST

*... data from about 100 million people in the U.S. was illegally accessed ... identified by Amazon.com Inc. as one of its **former cloud service employees** of breaking into the bank's server ...*

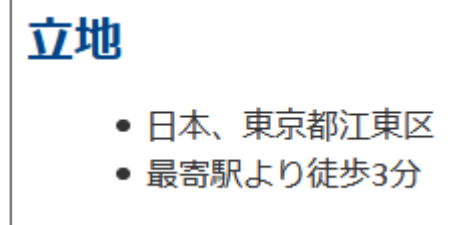
<https://www.bloomberg.com/news/articles/2019-07-29/capital-one-data-systems-breached-by-seattle-woman-u-s-says>

[Aside] Actual Locations of Datacenters (including Cloud Datacenters)

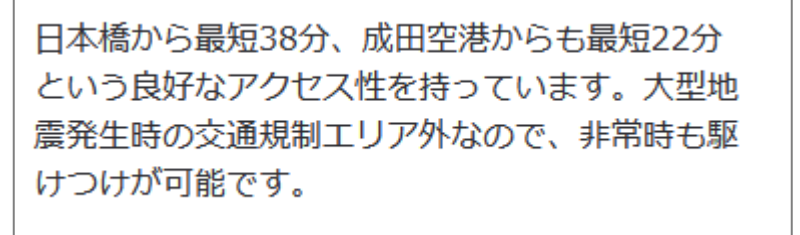
- The exact location of a datacenter is often (but not always) hidden to public
 - **Security reasons**: data stored in datacenters are so mission-critical and important
 - Examples:



An IDC Frontier Datacenter
<https://www.idcf.jp/datacenter/location/fuchu.html>



Tokyo 2nd Datacenter of NTT Communications
<https://www.ntt.com/business/services/data-center/colocation/nexcenter/japan.html>



Chiba 2nd Datacenter of SCSK
<https://www.netxdc.com/location/chiba2.html>

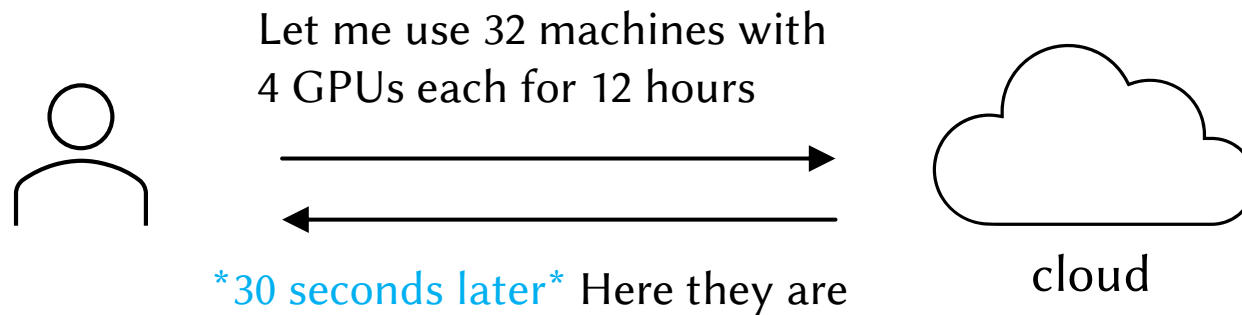
- Locations of Amazon datacenters are leaked to WikiLeaks
 - <https://wikileaks.org/amazon-atlas/map/>
 - In other words, **they were hidden if not leaked**



5 mins break

Cloud Computing Internals

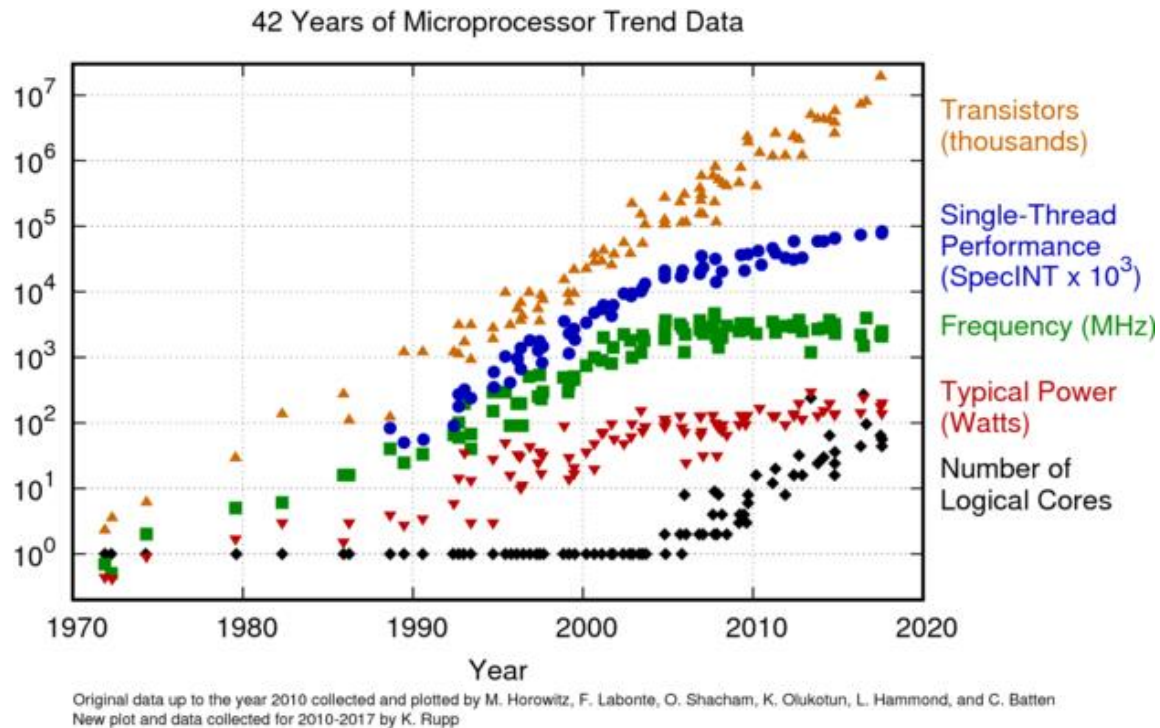
- Users' view: resource is prepared **almost instantly** after requested



- How is this achieved?
 - They build physical machines with requested specs in rush? → Of course not!
- Key technology: **Virtualization** (仮想化)
 - **Virtual machine (VM), OS-level virtualization** (a.k.a. containers), **network virtualization**

Why Study “Computer Systems” Today?

- The environments surrounding systems are rapidly changing



Examples of changes:

- The beginning of the ending of Moore's law
- Massively increased parallelism: up to 64 cores / CPU
- Power wall
- Memory wall
- Increased concern in cyber security

<https://www.karlrupp.net/2018/02/42-years-of-microprocessor-trend-data/>

We must study systems to make them adapt and fully utilize the new environments 25

Virtual Machines: What

- Virtual machines (VMs)
 - Provide **virtualized hardware** (e.g., CPU, memory, storage) that **look exactly like real one**
 - **A whole OS can be executed** on top of a VM
 - Underlying **resource can be partitioned** to provide multiple sets of virtualized hardware
- Widely available in many platforms
 - **VMWare** (Windows, Linux), **VirtualBox** (Windows, Linux, macOS), **Parallels** (macOS)

Application	Application
Guest OS 1 (e.g., Windows)	Guest OS 2 (e.g., Linux)
Virtualized HW 1	Virtualized HW 2
Host OS (e.g., Linux)	
Underlying (real) HW	

Resource partitioning example:

- Underlying hardware has **8 cores** and **32 GB** of RAM
- Virtualized hardware 1 is assigned **2 “virtual” cores** and **8 GB** of RAM
- Virtualized hardware 2 is assigned **4 “virtual” cores** and **4 GB** of RAM

Virtual Machines: Why

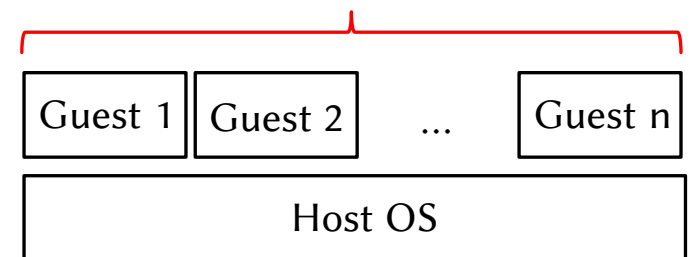
■ Why are VMs useful?

- Run multiple OSes on one set of hardware (i.e., resource partitioning)
- Run different OS than the underlying host OS
- Provide different hardware than the underlying real hardware
 - Examples: **IBM System/360**, **Wii U Virtual Console** (the same in Nintendo Switch??)

■ Cloud use-cases

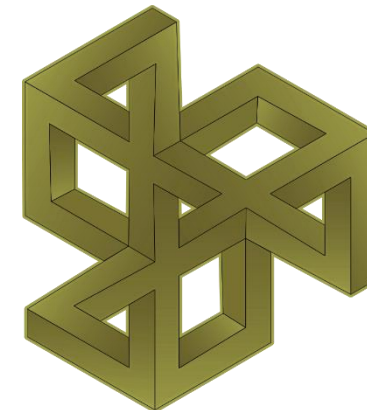
- “Pack” many users into one big server (e.g., 64 cores / server)
- Users can use any OS they want, and they have root privileges
- Strong OS-level security: users cannot see each other, unless there are no software bugs (and possible side channel attacks...)

N users (N guest OSes on 1 server)



How to run a guest OS on top of host OS?

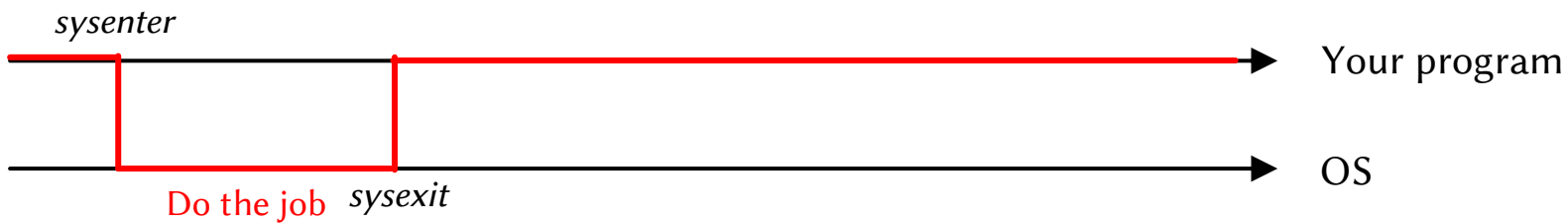
- Different perspective: How is an OS different from a normal program?
 1. OS directly handles **physical memory addresses**
 2. OS handles **exceptions, a.k.a. interruptions** (e.g., when a system call is invoked)
 3. OS can freely **configure the hardware** (e.g., changing CPU frequency through DVFS)
- What to do: Giving guest OS **an illusion** that it controls everything
 - In reality, something else does it on behalf of guest OS
 - Something else: virtual machine monitor (VMM)



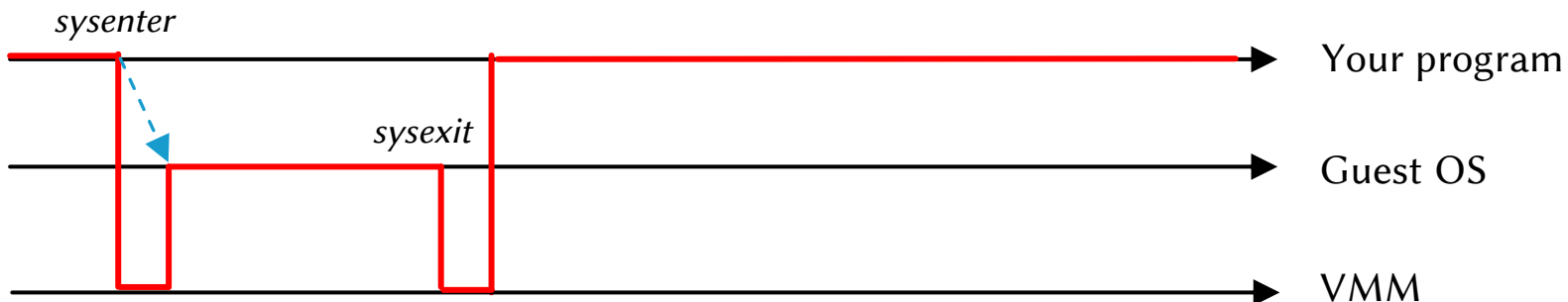
How to give an illusion: System Call Example

- System call (a.k.a. syscall)
 - Functionalities that require **OS privilege** (e.g., read/write to storage, memory allocation)
 - Normal programs invoke syscalls to use these functionalities by **raising a software exception**

Syscall invocation: No-virtualization case



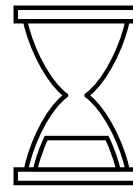
Syscall invocation: Virtualization case



From guest OS's view, the exception by syscall looks like **it directly comes from your program** (illustrated by ↘)

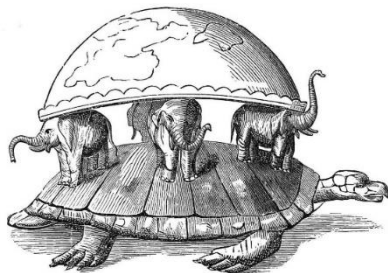
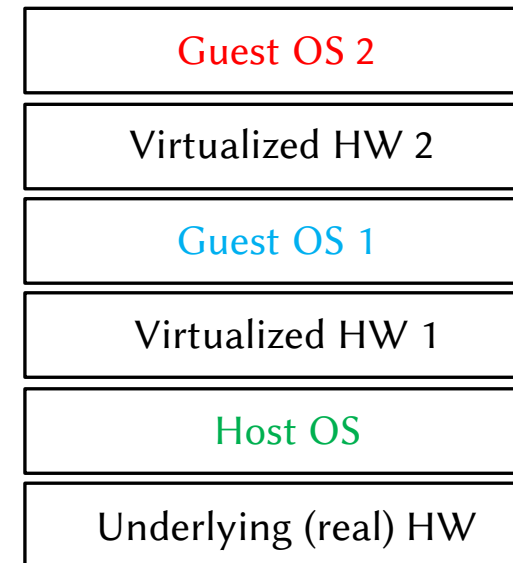
There are a lot more...

- How to give a guest OS an illusion to handle physical memory addresses?
 - Keywords: **Nested page table**, **Shadow page table**
- How to “catch” when a guest OS tries to do something that requires privilege?
 - Keywords: **Binary translation**, **Para virtualization** (準仮想化), **Full virtualization** (完全仮想化), **Hardware virtualization** (Intel VT-x, AMD-V)
- Other important technologies:
 - **Device pass-through** (PCI pass-through), **Processor execution mode** (e.g., **hypervisor mode**), **Nested virtualization**
- Not enough time to cover everything...
 - We only cover nested virtualization



Nested Virtualization

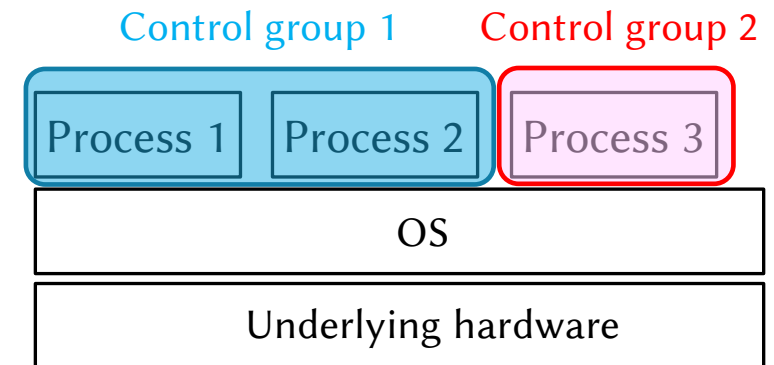
- Running guest OSes on top of a guest OS (that runs on top of the host OS)
 - Virtualization is “nested”
 - Typical use-case: User wants to run a VM on an IaaS cloud
- The same idea as single-layered virtualization is leveraged
 - When a syscall is issued, an exception is first delivered to **host OS**
 - **Host OS** dispatches it to **guest OS 1**, which believes that it's THE only OS
 - **Guest OS 1** dispatches it again to **guest OS 2**, which also believes the same
- “The Turtles Project: Design and Implementation of Nested Virtualization”, *OSDI'10*



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

OS-level Virtualization (a.k.a. Containers)

- VMs are great, but they introduce non-negligible **performance overhead**
 - Especially large for **syscalls**: the same syscall takes longer in virtualized environment (recall p. 30)
 - *Optional exercise: compare performance of syscalls on a VM and on a real machine*
- Countermeasure: Leverage OS-level virtualization (containers)
 - Processes (program instances) are grouped into **control groups**
 - Each group is assigned a set of resource (CPU time, memory, ...)
 - **Pros**: lightweight resource partitioning than virtual machines
 - **Cons**: the single OS “kernel” is shared among all users



Sharing OS “kernel” is NOT a Big Issue

- OS == kernel + userland tools / libraries

userland

- Example: Ubuntu 21.04 == Linux kernel 5.10 + `bash 5.0 + gcc 9.0 + emacs 26.1 + Python 3.9 + ...`



There really is a Linux, and these people are using it, but it is just a part of the system they use. Linux is the kernel: the program in the system that allocates the machine's resources to the other programs that you run. The kernel is an essential part of an operating system, but useless by itself; it can only function in the context of a complete operating system. Linux is normally used in combination with the GNU operating system: the whole system is basically GNU with Linux added, or GNU/Linux. All the so-called “Linux” distributions are really distributions of GNU/Linux.

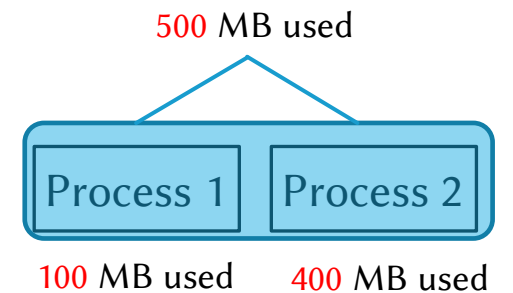
<https://www.gnu.org/gnu/linux-and-gnu.en.html>

- Sharing the kernel among containers is not a big issue in many cases
 - Users can still freely choose the userland programs (e.g., compiler versions)
 - This suffices requirements of many use-cases!
 - Example: Running CentOS 8.0 userland on Ubuntu 20.04 (the kernel is still shared)

Container Implementation

- Step 1: Grouping processes
 - Process data structure has data to express the belonging group
 - **Point:** the structure of processes are different from non-container counterparts
 - Done in OS-level → OS-level virtualization (cf. VM: everything running atop is the same)

- Step 2: Assigning resource to groups
 - OS knows which processes belong to the same group
 - CPU time assignment: schedule processes of the same group at once
 - Memory assignment: accumulate memory usage of processes of the same group



Container == Docker?

- Short answer: **No**
- Docker originally was a set of APIs to make container easy-to-use
 - Infrastructure-as-code paradigm with Dockerfiles / Dockerhub
 - Filesystem separation with [chroot](#) (existing tool)
 - Network separation using [bridge](#) (existing tool)
- However, Docker has evolved beyond that...
 - Because Docker is a de-facto, we want to use it in the same way on different OSes
 - How?: Docker on Windows provides Linux environments using a VM (HyperV)



```
FROM debian:stretch
MAINTAINER Soramichi Akiyama <akiyama@m.soramichi.jp>

RUN apt-get update && apt-get install -y exim4-base

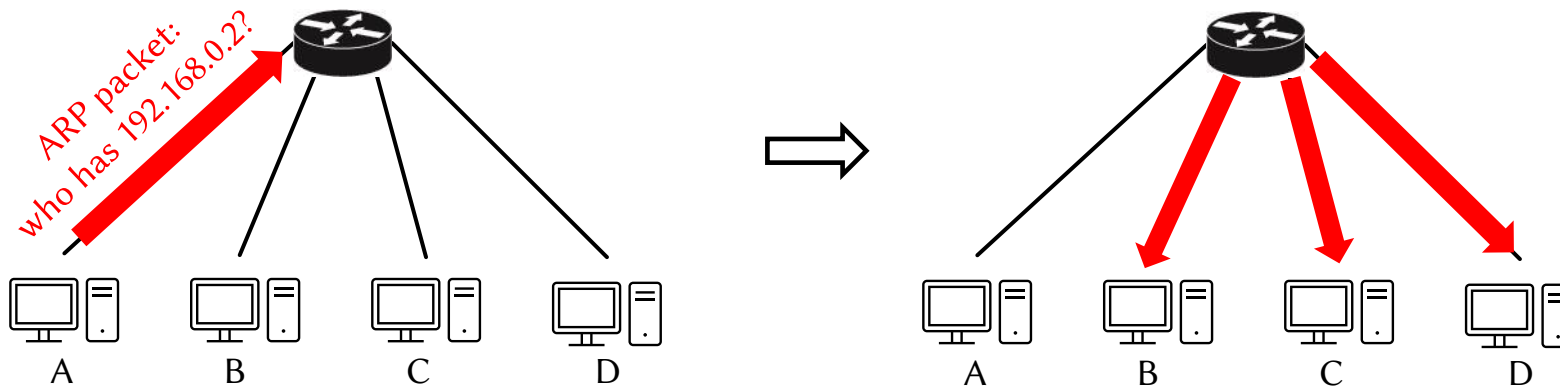
ADD update-exim4.conf.conf /etc/exim4/update-exim4.conf.conf
RUN echo "sky01.csg.ci.i.u-tokyo.ac.jp" > /etc/mailname && update-exim4.conf

CMD ["/etc/init.d/exim4", "start"]
```

An example Dockerfile

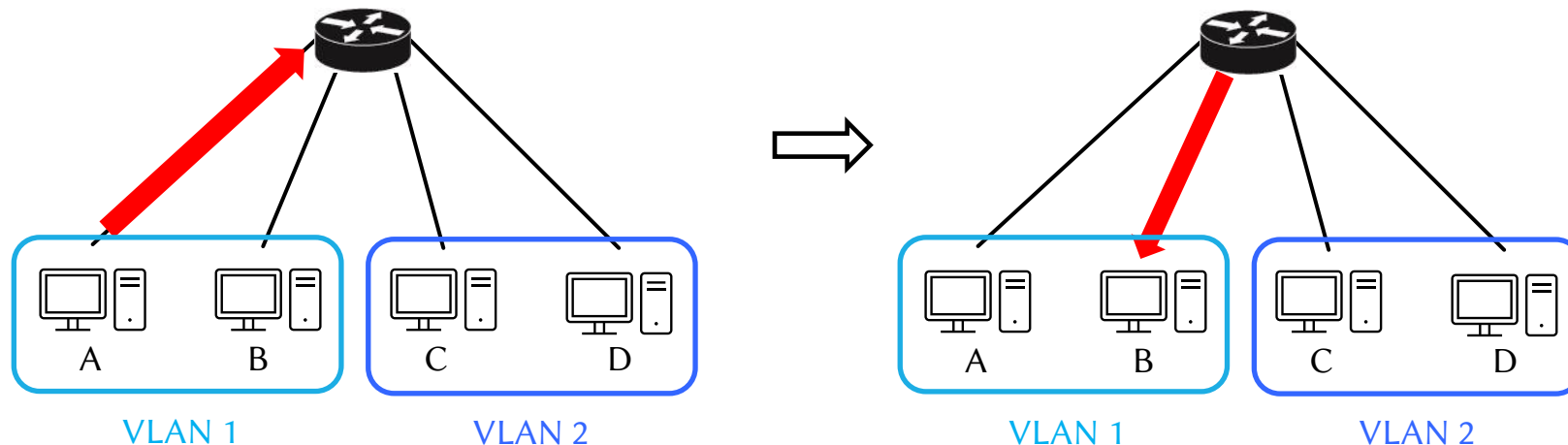
Network Virtualization: VLAN (1/2)

- IPv4 communication requires broadcasting in the edge
 - This leads to **network congestion (輻輳 ふくそう)** in large-scale networks
- Example: Machine A (192.168.0.1) communicates with machine B (192.168.0.2)
 - Step 0: Machine A must know the MAC address (a.k.a. Ethernet address) of machine B
 - Step 1: Machine A sends address resolution protocol (ARP) packet to the **broadcast MAC address**
 - Step 2: Network switch(es) **replicate the ARP packet and send it to every other machine**
 - Step 3: Machine B detects that machine A is looking for it, and sends a reply to machine A



Network Virtualization: VLAN (2/2)

- VLAN limits broadcast domain to specific machines
 - Reduces congestion caused by broadcast packets
- How different VLAN types work
 - Port VLAN: Each port of a switch belongs to a specific VLAN ID, and broadcast packets are replicated only with the same VLAN ID
 - Tagged VLAN: Effective when extending a VLAN to multiple network switches



Broadcast packets are sent only to machines in the same VLAN

[Aside] Side-channel Attacks (1/2)

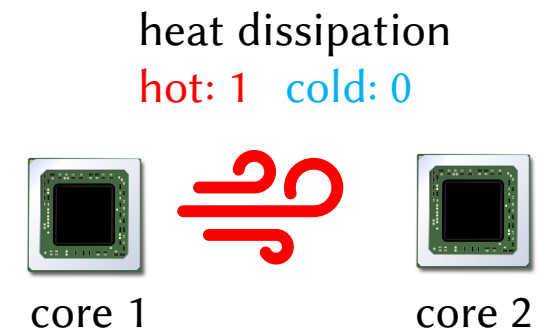
- Is it completely safe inside VMs, VLANs, etc.?
 - Stronger security is an important aspect of virtualization besides resource partitioning
 - Answer to the question: No, unfortunately
 - Even if you assume everything is bug-free (which you shouldn't), **still no**
- Side-channel attacks
 - Communication mechanisms through a “channel” that is not meant for communication
 - Toy example: Sending Morse code (モールス信号) by blinking mic icon



- Imagine sending chat messages among students is prohibited
- You can still send messages by “blinking” the mic icons
- This method **does not exploit any bugs**, but **something that is not supposed to happen can still happen**

[Aside] Side-channel Attacks (2/2)

- More realistic examples
 - Detecting the control flow of a program by measuring elector-magnetic wave emitted from the host machine [4]
 - Detecting what you are talking using position sensors of HDD heads [5]
 - Communication between CPU cores using thermal sensors [6]
 - And of course, “spectre”, “meltdown”, and many variants
- Any effective defense?
 - Maybe no, at least from cloud users’ sides ...
 - Fortunately, **no real attacks exploiting side channels have found** (a Google researcher said this, but no reliable source exists)



[4] Robert Callan *et al.*, “Zero-Overhead Profiling via EM Emanations”, *ISSTA’16*

[5] Andrew Kwong *et al.*, “Hard Drive of Hearing: Disks that Eavesdrop with a Synthesized Microphone”, *IEEE S&P 2019*

[6] David B. Bartolini *et al.*, “On the capacity of thermal covert channels in multicores”, *EuroSys’16*