6.033 in the news

“Unfortunately, the system did not scale fast enough to accommodate the increased volume”

“One such technology that the state could use is a cloud computing service that scales up server power to meet demand. Olivia Adams, a software developer who created MACovidVaccines.com while on maternity leave, uses one such service from Amazon. That’s why her website didn’t crash when the state’s did, she thinks.”

https://www.wbur.org/news/2021/02/19/prepmod-state-vaccine-appointment-site-crash-statement

https://www.wbur.org/commonhealth/2021/02/18/hours-and-hours-of-frustration-mass-residents-emote-about-states-faulty-vaccine-websites
“Unfortunately, the system did not scale fast enough to accommodate the increased volume”

“One such technology that the state could use is a cloud computing service that scales up server power to meet demand. Olivia Adams, a software developer who created MACovidVaccines.com while on maternity leave, uses one such service from Amazon. That’s why her website didn’t crash when the state’s did, she thinks.”

questions that arise: Why didn’t the state’s website scale to meet demand? How do we test systems for that? How do we understand how far a system can scale, and in what dimensions?
6.033 in the news

“Unfortunately, the system did not scale fast enough to accommodate the increased volume”

“One such technology that the state could use is a cloud computing service that scales up server power to meet demand. Olivia Adams, a software developer who created MACovidVaccines.com while on maternity leave, uses one such service from Amazon. That’s why her website didn’t crash when the state’s did, she thinks.”

questions that arise:

Why didn’t the state’s website scale to meet demand? How do we test systems for that? How do we understand how far a system can scale, and in what dimensions?

Are companies like Amazon and Google the only places to get such highly-scaling services? If so, why? What implications does that have?
6.033 in the news

“Unfortunately, the system did not scale fast enough to accommodate the increased volume”

“One such technology that the state could use is a cloud computing service that scales up server power to meet demand. Olivia Adams, a software developer who created MACovidVaccines.com while on maternity leave, uses one such service from Amazon. That’s why her website didn’t crash when the state’s did, she thinks.”

questions that arise:

Why didn’t the state’s website scale to meet demand? How do we test systems for that? How do we understand how far a system can scale, and in what dimensions?

Are companies like Amazon and Google the only places to get such highly-scaling services? If so, why? What implications does that have?

In particular, what are the consequences of public infrastructure relying on private companies?
Lecture #2: Naming
plus a case-study on DNS
last time: enforced modularity via client/server

```python
def main():
    html = browser_load_url(URL)
    ...

def browser_load_url(url):
    msg = url  # could reformat
    send request
    wait for reply
    html = reply  # could reformat
    return html

def server_load_url():
    ...
    return html

def handle_server_load_url(url):
    wait for request
    url = request
    html = server_load_url(URL)
    reply = html
    send reply
```
last time: enforced modularity via client/server

client

```python
def main():
    html = browser_load_url(URL)
    ...

def browser_load_url(url):
    msg = url
    # could reformat
    send request
    wait for reply
    html = reply
    # could reformat
    return html
```

server

```python
def server_load_url():
    ...
    return html

def handle_server_load_url(url):
    wait for request
    url = request
    html = server_load_url(url)
    reply = html
    send reply
```

network

load("kaws.com/buy.html?llama")

stub
def main():
    html = browser_load_url(URL)
...

def browser_load_url(url):
    msg = url # could reformat
    send request
    wait for reply
    html = reply # could reformat
    return html

def server_load_url():
    ...
    return html

def handle_server_load_url(url):
    wait for request
    url = request
    html = server_load_url(URL)
    reply = html
    send reply
last time: enforced modularity via client/server

today: naming, which allows modules to interact
why use names?

Class Browser
(on machine 1)

```
def main():
    html = browser_load_url(URL)
...
```

```
def browser_load_url(url):
    msg = url # could reformat
    send request
    wait for reply
    html = reply # could reformat
    return html
```

stub

Class Server
(on machine 2)

```
def server_load_url():
    ...
    return html
```

```
def handle_server_load_url(url):
    wait for request
    url = request
    html = server_load_url(URL)
    reply = html
    send reply
```

stub

load("kaws.com/buy.html?llama")
why use names? they let us achieve modularity by providing communication and organization, as well as a number of other properties.

---

**client**

**Class Browser** (on machine 1)

```python
def main():
    html = browser_load_url(URL)
    ...

def browser_load_url(url):
    msg = url # could reformat
    send request
    wait for reply
    html = reply # could reformat
    return html
```

**server**

**Class Server** (on machine 2)

```python
def server_load_url():
    ...
    return html

def handle_server_load_url(url):
    wait for request
    url = request
    html = server_load_url(URL)
    reply = html
    send reply
```

---

load(“kaws.com/buy.html?llama”)

---

stub

stub
why use names?

they let us achieve modularity by providing communication and organization, as well as a number of other properties

Class Browser  
(on machine 1)

```
def main():
    html = browser_load_url(URL)
...
```

```
def browser_load_url(url):
    msg = url # could reformat
    send request
    wait for reply
    html = reply # could reformat
    return html
```

stub

Class Server  
(on machine 2)

```
def server_load_url():
    ...
    return html
```

stub

retrieval

the client can retrieve the llama page because it can name it

Class Browser  
(on machine 1)

load(“kaws.com/buy.html?llama”)
why use names?

they let us achieve modularity by providing communication and organization, as well as a number of other properties

**client**

```python
Class Browser
(on machine 1)

def main():
    html = browser_load_url(URL)
...
```

```python
def browser_load_url(url):
    msg = url # could reformat
    send request
    wait for reply
    html = reply # could reformat
    return html
```

**server**

```python
Class Server
(on machine 2)

def server_load_url():
    ...
    return html
```

```python
def handle_server_load_url(url):
    wait for request
    url = request
    html = server_load_url(URL)
    reply = html
    send reply
```

**retrieval**
The client can retrieve the llama page because it can name it.

**sharing**
The server can share the llama page with multiple clients (i.e., multiple clients can view this page).
why use names? they let us achieve modularity by providing communication and organization, as well as a number of other properties

**client**

```
Class Browser
(on machine 1)

def main():
    html = browser_load_url(URL)
    ...

def browser_load_url(url):
    msg = url  # could reformat
    send request
    wait for reply
    html = reply  # could reformat
    return html
```

**server**

```
Class Server
(on machine 2)

def server_load_url():
    ...
    return html
```

**retrieval**

the client can retrieve the llama page because it can name it

**sharing**

the server can share the llama page with multiple clients (i.e., multiple clients can view this page)

**user-friendly IDs**

```
kaws.com is easier to remember than (say)
18.25.4.171; the variable name “html” is easier to remember than a particular location in memory
```
why use names?

they let us achieve modularity by providing communication and organization, as well as a number of other properties

Class Browser
(on machine 1)

```python
def main():
    html = browser_load_url(URL)
    ...
```

def browser_load_url(url):
    msg = url # could reformat
    send request
    wait for reply
    html = reply # could reformat
    return html

Class Server
(on machine 2)

```python
def server_load_url():
    ...
    return html
```

def handle_server_load_url(url):
    wait for request
    url = request
    html = server_load_url(URL)
    reply = html
    send reply

retrieval
the client can retrieve the llama page because it can name it

sharing
the server can share the llama page with multiple clients (i.e., multiple clients can view this page)

stub

user-friendly IDs
kaws.com is easier to remember than (say) 18.25.4.171; the variable name “html” is easier to remember than a particular location in memory

stub

addressing
some names also specify location information
why use names? they let us achieve modularity by providing communication and organization, as well as a number of other properties

```
server

Class Server  
(on machine 2)

def server_load_url():
  ...
  return html

def handle_server_load_url(url):
  wait for request
  url = request
  html = server_load_url(URL)
  reply = html
  send reply
```

stub
why use names? they let us achieve modularity by providing communication and organization, as well as a number of other properties

```python
def server_load_url():
    ... return html

def handle_server_load_url(url):
    wait for request
    url = request
    html = server_load_url(URL)
    reply = html
    send reply
```

```
Class Server
(on machine 2)

llama_data.txt

 price
 size
 color
...
```
why use names? they let us achieve modularity by providing communication and organization, as well as a number of other properties.

hiding

code on the server can access llama_data.txt without having to worry about how the file is laid out in memory.
why use names?

they let us achieve modularity by providing communication and organization, as well as a number of other properties.

---

**hiding**

code on the server can access `llama_data.txt` without having to worry about how the file is laid out in memory.

**indirection**

the server can change the memory layout of `llama_data.txt` without notifying the user.
why use names? they let us achieve modularity by providing communication and organization, as well as a number of other properties
why use names? they let us achieve modularity by providing communication and organization, as well as a number of other properties.

the design of a system’s naming scheme(s) helps it achieve these properties.
why use names? They let us achieve modularity by providing communication and organization, as well as a number of other properties.

The design of a system’s naming scheme(s) helps it achieve these properties.

A naming scheme includes...
why use names? they let us achieve modularity by providing communication and organization, as well as a number of other properties

the design of a system’s naming scheme(s) helps it achieve these properties

a naming scheme includes

1. the set of all possible names
Why use names? They let us achieve modularity by providing communication and organization, as well as a number of other properties.

The design of a system’s naming scheme(s) helps it achieve these properties:

A naming scheme includes:

1. The set of all possible names
2. The set of all possible values
why use names? they let us achieve modularity by providing communication and organization, as well as a number of other properties.

the design of a system’s **naming scheme(s)** helps it achieve these properties:

a naming scheme includes:

1. the set of all possible **names**
2. the set of all possible **values**
3. a **look-up algorithm** to translate a name into a value (or a set of values, or “none”)

Katrina LaCurts | lacurts@mit.edu | 6.033 2021
naming case study: the domain name system (DNS), which maps hostnames (eecs.mit.edu) to IP addresses (18.62.1.6)
naming case study: the **domain name system (DNS)**, which maps **hostnames** (eecs.mit.edu) to **IP addresses** (18.62.1.6)

the **look-up algorithm** has to scale to the size of the Internet, while dealing with constant updates and issues of delegation
naming case study: the **domain name system (DNS)**, which maps hostnames (eecs.mit.edu) to **IP addresses** (18.62.1.6)

the **look-up algorithm** has to scale to the size of the Internet, while dealing with constant updates and issues of delegation.
naming case study: The **domain name system (DNS)**, which maps hostnames (eecs.mit.edu) to IP addresses (18.62.1.6) the look-up algorithm has to scale to the size of the Internet, while dealing with constant updates and issues of delegation

![Diagram of the domain name system](image-url)
naming case study: the **domain name system (DNS)**, which maps hostnames (eecs.mit.edu) to IP addresses (18.62.1.6)

the **look-up algorithm** has to scale to the size of the Internet, while dealing with constant updates and issues of delegation.
naming case study: the **domain name system (DNS)**, which maps hostnames (eecs.mit.edu) to **IP addresses** (18.62.1.6)

the **look-up algorithm** has to scale to the size of the Internet, while dealing with constant updates and issues of delegation
naming case study: the domain name system (DNS), which maps hostnames (eecs.mit.edu) to IP addresses (18.62.1.6)

the look-up algorithm has to scale to the size of the Internet, while dealing with constant updates and issues of delegation.
naming case study: the domain name system (DNS), which maps hostnames (eecs.mit.edu) to IP addresses (18.62.1.6)

the look-up algorithm has to scale to the size of the Internet, while dealing with constant updates and issues of delegation.

a partial view of the DNS hierarchy. each box represents a zone. name servers within a zone keep track of that zone’s mappings.
naming case study: the **domain name system (DNS)**, which maps hostnames (eecs.mit.edu) to **IP addresses** (18.62.1.6)

the **look-up algorithm** has to scale to the size of the Internet, while dealing with constant updates and issues of delegation

---

a partial view of the DNS hierarchy. each box represents a **zone**. name servers within a zone keep track of that zone’s mappings

Katrina LaCurts | lacurts@mit.edu | 6.033 2021
naming case study: the **domain name system (DNS)**, which maps hostnames (eecs.mit.edu) to **IP addresses** (18.62.1.6)

the **look-up algorithm** has to scale to the size of the Internet, while dealing with constant updates and issues of delegation.

A partial view of the DNS hierarchy. Each box represents a **zone**. Name servers within a zone keep track of that zone’s mappings.
naming case study: the **domain name system (DNS)**, which maps hostnames (eecs.mit.edu) to IP addresses (18.62.1.6)

the **look-up algorithm** has to scale to the size of the Internet, while dealing with constant updates and issues of delegation

A partial view of the DNS hierarchy. Each box represents a **zone**. Name servers within a zone keep track of that zone’s mappings.
naming case study: the **domain name system (DNS)**, which maps hostnames (eecs.mit.edu) to **IP addresses** (18.62.1.6)

the **look-up algorithm** has to scale to the size of the Internet, while dealing with constant updates and issues of delegation

---

a partial view of the DNS hierarchy. each box represents a **zone**. name servers within a zone keep track of that zone’s mappings

Katrina LaCurts | lacurts@mit.edu | 6.033 2021
naming case study: the domain name system (DNS), which maps hostnames (eecs.mit.edu) to IP addresses (18.62.1.6)

the look-up algorithm has to scale to the size of the Internet, while dealing with constant updates and issues of delegation

a partial view of the DNS hierarchy. each box represents a zone. name servers within a zone keep track of that zone’s mappings
naming case study: the **domain name system (DNS)**, which maps hostnames (*eecs.mit.edu*) to **IP addresses** (*18.62.1.6*).

The **lookup algorithm** has to scale to the size of the Internet, while dealing with constant updates and issues of delegation.

A partial view of the DNS hierarchy. Each box represents a **zone**. Name servers within a zone keep track of that zone’s mappings.
naming case study: the domain name system (DNS), which maps hostnames (eecs.mit.edu) to IP addresses (18.62.1.6)

the look-up algorithm has to scale to the size of the Internet, while dealing with constant updates and issues of delegation

a partial view of the DNS hierarchy. each box represents a zone. name servers within a zone keep track of that zone’s mappings
naming case study: the **domain name system (DNS)**, which maps hostnames (*eecs.mit.edu*) to IP addresses (18.62.1.6)

The look-up algorithm has to scale to the size of the Internet, while dealing with constant updates and issues of delegation.

A partial view of the DNS hierarchy. Each box represents a **zone**. Name servers within a zone keep track of that zone’s mappings.
naming case study: the **domain name system (DNS)**, which maps hostnames (eecs.mit.edu) to **IP addresses** (18.62.1.6)

the **look-up algorithm** has to scale to the size of the Internet, while dealing with constant updates and issues of delegation

![](image)

a partial view of the DNS hierarchy. each box represents a **zone**. name servers within a zone keep track of that zone’s mappings
naming case study: the **domain name system (DNS)**, which maps hostnames (eecs.mit.edu) to **IP addresses** (18.62.1.6)

the **look-up algorithm** has to scale to the size of the Internet, while dealing with constant updates and issues of delegation

---

a partial view of the DNS hierarchy. each box represents a **zone**. name servers within a zone keep track of that zone’s mappings

---

**DNS client**

e.g., your computer

**query sent to:**

**response:**
naming case study: the domain name system (DNS), which maps hostnames (eecs.mit.edu) to IP addresses (18.62.1.6)

the look-up algorithm has to scale to the size of the Internet, while dealing with constant updates and issues of delegation

a partial view of the DNS hierarchy. each box represents a zone. name servers within a zone keep track of that zone’s mappings

DNS client e.g., your laptop

query sent to: 198.41.0.4
response: 198.41.0.4
naming case study: the domain name system (DNS), which maps hostnames (eecs.mit.edu) to IP addresses (18.62.1.6)

The look-up algorithm has to scale to the size of the Internet, while dealing with constant updates and issues of delegation.

A partial view of the DNS hierarchy. Each box represents a zone. Name servers within a zone keep track of that zone’s mappings.

DNS client query sent to: 198.41.0.4 response: try 192.14.171.191
naming case study: the domain name system (DNS), which maps hostnames (eecs.mit.edu) to IP addresses (18.62.1.6)

the look-up algorithm has to scale to the size of the Internet, while dealing with constant updates and issues of delegation

a partial view of the DNS hierarchy. each box represents a zone. name servers within a zone keep track of that zone’s mappings

DNS client e.g., your laptop
query sent to: 192.14.171.191
response: 192.14.171.191
The domain name system (DNS), which maps hostnames (eecs.mit.edu) to IP addresses (18.62.1.6), has to scale to the size of the Internet, while dealing with constant updates and issues of delegation.

A partial view of the DNS hierarchy. Each box represents a zone. Name servers within a zone keep track of that zone’s mappings.
naming case study: the **domain name system (DNS)**, which maps hostnames (eecs.mit.edu) to IP addresses (18.62.1.6)

the **look-up algorithm** has to scale to the size of the Internet, while dealing with constant updates and issues of delegation

a partial view of the DNS hierarchy. each box represents a **zone**. name servers within a zone keep track of that zone’s mappings

**DNS client**

e.g., your laptop

query sent to: **18.72.0.3**

response: **18.72.0.3**
naming case study: the **domain name system (DNS)**, which maps hostnames (eecs.mit.edu) to **IP addresses** (18.62.1.6)

the **look-up algorithm** has to scale to the size of the Internet, while dealing with constant updates and issues of delegation

a partial view of the DNS hierarchy. each box represents a **zone**. name servers within a zone keep track of that zone’s mappings

```plaintext
198.41.0.4
  root
  nameserver

192.14.171.192
  com.
  nameserver

192.14.171.191
  edu.
  nameserver
    mit.edu.
      18.72.0.3
      A
      18.9.22.169
      A
  eecs.mit.edu.
    18.62.1.6
    A
  web.mit.edu.
    18.9.2.69
    A

root
  nameserver

edu.
  192.14.171.191 NS
  192.14.171.192 NS
  192.14.171.193 NS

com.
  192.14.171.191 NS

net.
  192.14.171.191 NS

nameserver edu.
  192.14.171.191 NS

nameserver mit.
  192.14.171.191 NS

DNS client e.g., your laptop
query sent to: 18.72.0.3
response: 18.62.1.6
```
naming case study: the **domain name system (DNS)**, which maps **hostnames** (eecs.mit.edu) to **IP addresses** (18.62.1.6)

the **look-up algorithm** has to scale to the size of the Internet, while dealing with constant updates and issues of delegation

A partial view of the DNS hierarchy. Each box represents a **zone**. Name servers within a zone keep track of that zone's mappings.
naming case study: the **domain name system (DNS)**, which maps **hostnames** (eecs.mit.edu) to **IP addresses** (18.62.1.6)

the **look-up algorithm** has to scale to the size of the Internet, while dealing with constant updates and issues of delegation

---

**a partial view of the DNS hierarchy. each box represents a zone. name servers within a zone keep track of that zone’s mappings**

---

**performance issue:** this is a *lot* of queries, especially to the root server
naming case study: the domain name system (DNS), which maps hostnames (eecs.mit.edu) to IP addresses (18.62.1.6)

the look-up algorithm has to scale to the size of the Internet, while dealing with constant updates and issues of delegation.

performance issue: this is a lot of queries, especially to the root server.

reliability issue: what happens when a nameserver fails or (security issue) is attacked?

a partial view of the DNS hierarchy. each box represents a zone. name servers within a zone keep track of that zone’s mappings.
naming case study: the **domain name system (DNS)**, which maps **hostnames** (eecs.mit.edu) to **IP addresses** (18.62.1.6)

the **look-up algorithm** has to scale to the size of the Internet, while dealing with constant updates and issues of delegation

---

**performance issue**: this is a *lot* of queries, especially to the root server

**reliability issue**: what happens when a nameserver fails or *(security issue)* is attacked?

**control issue**: who should own the root server?

---

a partial view of the DNS hierarchy. each box represents a **zone**. name servers within a zone keep track of that zone’s mappings
modularity and abstraction mitigate complexity. A client/server model allows us to enforce modularity by putting modules on physically separate machines.
modularity and abstraction mitigate complexity. A client/server model allows us to enforce modularity by putting modules on physically separate machines.

naming is what allows modules to interact, and can help us achieve other goals through properties such as indirection, user-friendliness, etc.
modularity and abstraction mitigate complexity. a client/server model allow us to enforce modularity by putting modules on physically separate machines.

naming is what allows modules to interact, and can help us achieve other goals through properties such as indirection, user-friendliness, etc.

the domain name system is a great case-study in naming, and also illustrates principles such as hierarchy, scalability, delegation, and decentralization
modularity and abstraction mitigate complexity. A client/server model allow us to enforce modularity by putting modules on physically separate machines.

naming is what allows modules to interact, and can help us achieve other goals through properties such as indirection, user-friendliness, etc.

The domain name system is a great case-study in naming, and also illustrates principles such as hierarchy, scalability, delegation, and decentralization.
Modularity and abstraction mitigate complexity. A client/server model allows us to enforce modularity by putting modules on physically separate machines.

Naming is what allows modules to interact, and can help us achieve other goals through properties such as indirection, user-friendliness, etc.

The domain name system is a great case-study in naming, and also illustrates principles such as hierarchy, scalability, delegation, and decentralization.

The example you saw in lecture was a fairly basic one; you will talk more about DNS’s performance enhancements in recitation tomorrow, which change how some (many) DNS queries are resolved and client/server models, and (tomorrow) caching, and (in May) security...