

# 6.1800 Spring 2024

## Lecture #26: What matters

even if you never want to design a computer system again

**modularity and abstraction in operating systems:** operating systems are *all about* providing an abstraction between software and hardware

---

**operating systems** enforce modularity on a single machine using **virtualization**

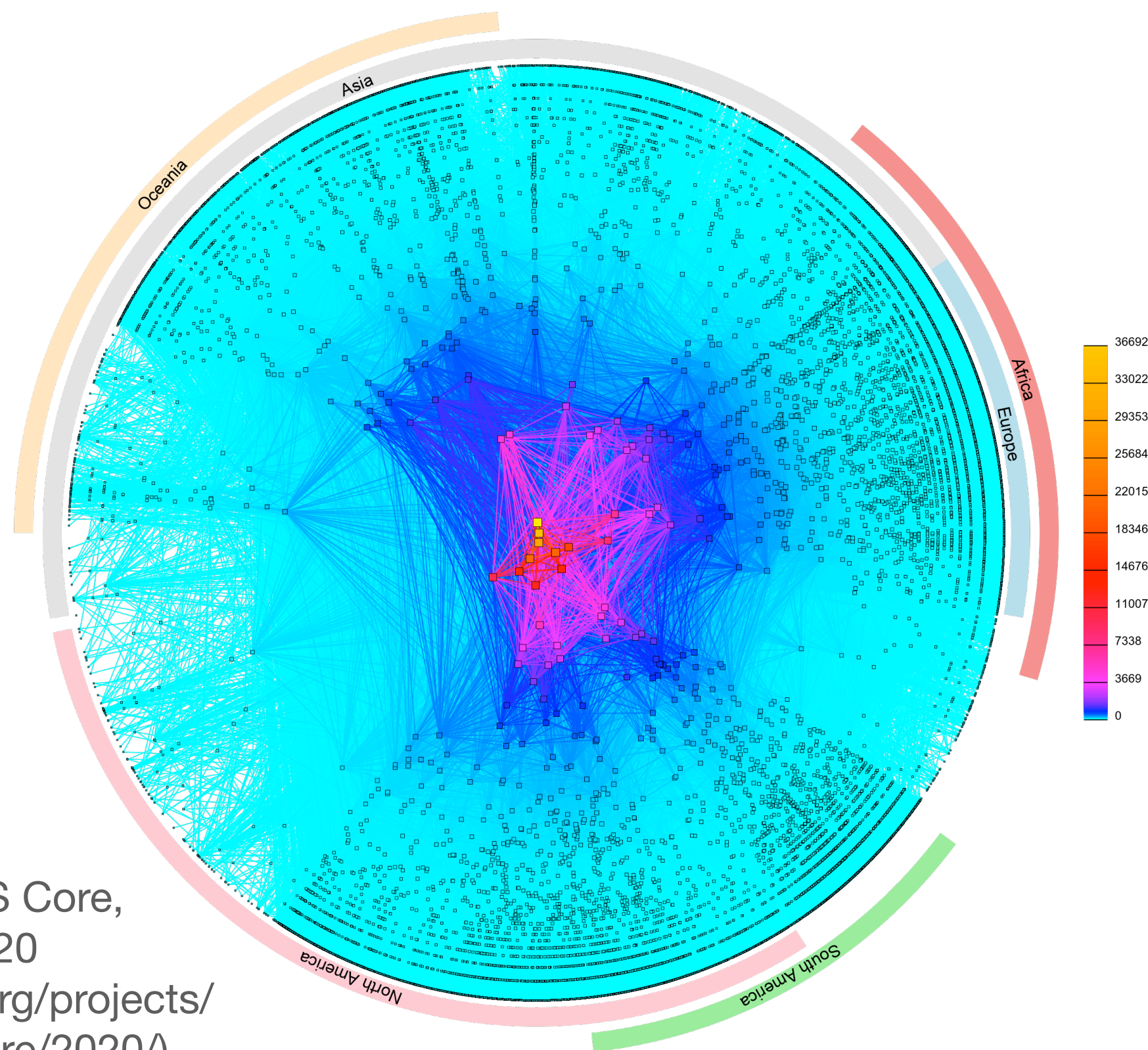
in order to enforce modularity + have an effective operating system, a few things need to happen

1. programs shouldn't be able to refer to (and corrupt) each others' **memory** .....→ **virtual memory**
2. programs should be able to **communicate** with each other .....→ **bounded buffers**  
(virtualize communication links)
3. programs should be able to **share a CPU** without one program halting the progress of the others .....→ **threads**  
(virtualize processors)

# modularity and abstraction in networking: an abundance of hierarchy and layering

1970s: ARPAnet      1978: flexibility and layering      early 80s: growth → change      late 80s: growth → problems      1993: commercialization

hosts.txt      distance-vector      TCP, UDP      OSPF, EGP, DNS      congestion collapse      policy routing      CIDR



CAIDA's IPv4 AS Core,  
January 2020  
(<https://www.caida.org/projects/cartography/as-core/2020/>)

**application**

the things that actually generate traffic

**transport**

sharing the network, reliability (or not)  
*examples: TCP, UDP*

**network**

naming, addressing, routing  
*examples: IP*

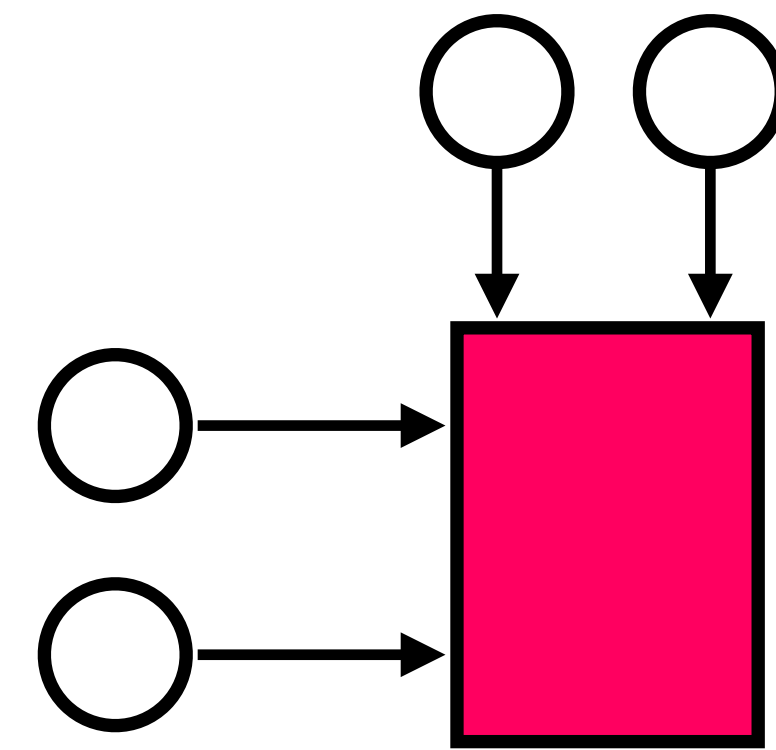
**link**

communication between two directly-connected nodes  
*examples: ethernet, bluetooth, 802.11 (wifi)*

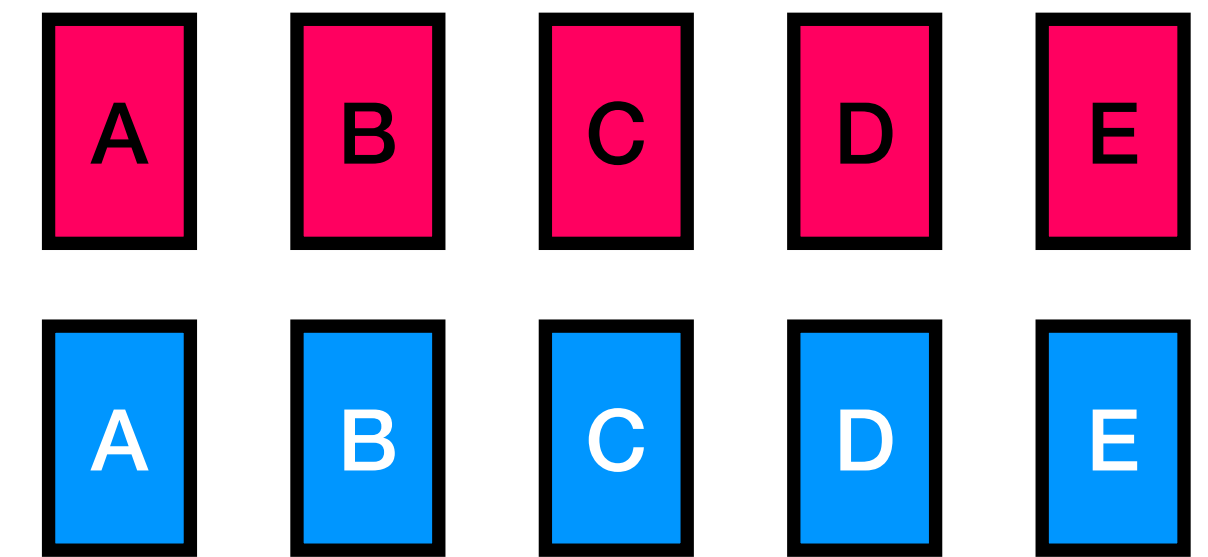


# modularity and abstraction in distributed systems: transactions are an abstraction that allow us to reason about fault-tolerance

our goal is to build **reliable systems from unreliable components**. we want to build systems that serve many clients, store a lot of data, perform well, all while keeping availability high



**replicated state machines** give us single-copy consistency even with replicated data



**transactions** — which provide **atomicity** and **isolation** — make it easier for us to reason about failures

our job in lecture is to understand how a system *implements* these two abstractions.  
how do our systems guarantee atomicity? how do they guarantee isolation?

**atomicity:** provided by **logging**, which gives better performance than shadow copies at the cost of some added complexity; **two-phase commit** gives us multi-site atomicity

**isolation:** provided by **two-phase locking**



# modularity and abstraction in distributed systems: systems like MapReduce and GFS abstract failure away from users

## MapReduce: Simplified Data Processing on Large Clusters

Jeffrey Dean and Sanjay Ghemawat

jeff@google.com, sanjay@google.com

Google, Inc.

### Abstract

MapReduce is a programming model and an associated implementation for processing and generating large data sets. Users specify a *map* function that processes a key/value pair to generate a set of intermediate key/value pairs, and a *reduce* function that merges all intermediate values associated with the same intermediate key. Many real world tasks are expressible in this model, as shown in the paper.

Programs written in this functional style are automatically parallelized and executed on a large cluster of commodity machines. The run-time system takes care of the details of partitioning the input data, scheduling the program's execution across a set of machines, handling machine failures, and managing the required inter-machine communication. This allows programmers without any experience with parallel and distributed systems to easily utilize the resources of a large distributed system.

Our implementation of MapReduce runs on a large cluster of commodity machines and is highly scalable: a typical MapReduce computation processes many ter-

given day, etc. Most such computations are conceptually straightforward. However, the input data is usually large and the computations have to be distributed across hundreds or thousands of machines in order to finish in a reasonable amount of time. The issues of how to parallelize the computation, distribute the data, and handle failures conspire to obscure the original simple computation with large amounts of complex code to deal with these issues.

As a reaction to this complexity, we designed a new abstraction that allows us to express the simple computations we were trying to perform but hides the messy details of parallelization, fault-tolerance, data distribution and load balancing in a library. Our abstraction is inspired by the *map* and *reduce* primitives present in Lisp and many other functional languages. We realized that most of our computations involved applying a *map* operation to each logical "record" in our input in order to compute a set of intermediate key/value pairs, and then applying a *reduce* operation to all the values that shared the same key, in order to combine the derived data appropriately. Our use of a functional model with user-

## The Google File System

Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung

Google\*

### ABSTRACT

We have designed and implemented the Google File System, a scalable distributed file system for large distributed data-intensive applications. It provides fault tolerance while running on inexpensive commodity hardware, and it delivers high aggregate performance to a large number of clients.

While sharing many of the same goals as previous distributed file systems, our design has been driven by observations of our application workloads and technological environment, both current and anticipated, that reflect a marked departure from some earlier file system assumptions. This has led us to reexamine traditional choices and explore radically different design points.

The file system has successfully met our storage needs. It is widely deployed within Google as the storage platform for the generation and processing of data used by our service as well as research and development efforts that require large data sets. The largest cluster to date provides hundreds of terabytes of storage across thousands of disks on over a thousand machines, and it is concurrently accessed by hundreds of clients.

In this paper, we present file system interface extensions designed to support distributed applications, discuss many aspects of our design, and report measurements from both micro-benchmarks and real world use.

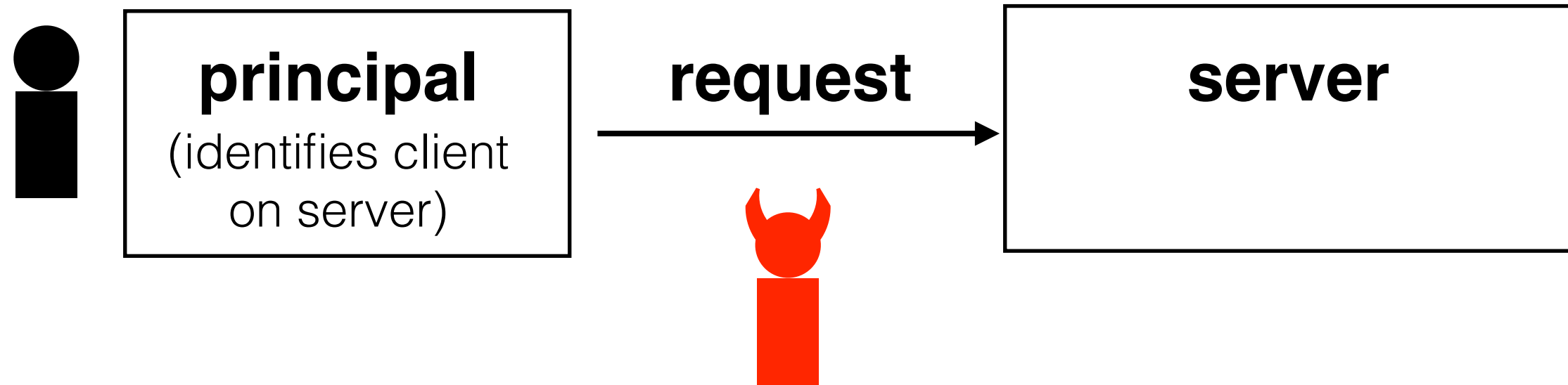
### 1. INTRODUCTION

We have designed and implemented the Google File System (GFS) to meet the rapidly growing demands of Google's data processing needs. GFS shares many of the same goals as previous distributed file systems such as performance, scalability, reliability, and availability. However, its design has been driven by key observations of our application workloads and technological environment, both current and anticipated, that reflect a marked departure from some earlier file system design assumptions. We have reexamined traditional choices and explored radically different points in the design space.

First, component failures are the norm rather than the exception. The file system consists of hundreds or even thousands of storage machines built from inexpensive commodity parts and is accessed by a comparable number of client machines. The quantity and quality of the components virtually guarantee that some are not functional at any given time and some will not recover from their current failures. We have seen problems caused by application bugs, operating system bugs, human errors, and the failures of disks, memory, connectors, networking, and power supplies. Therefore, constant monitoring, error detection, fault tolerance, and automatic recovery must be integral to the system.

Second, files are huge by traditional standards. Multi-GB

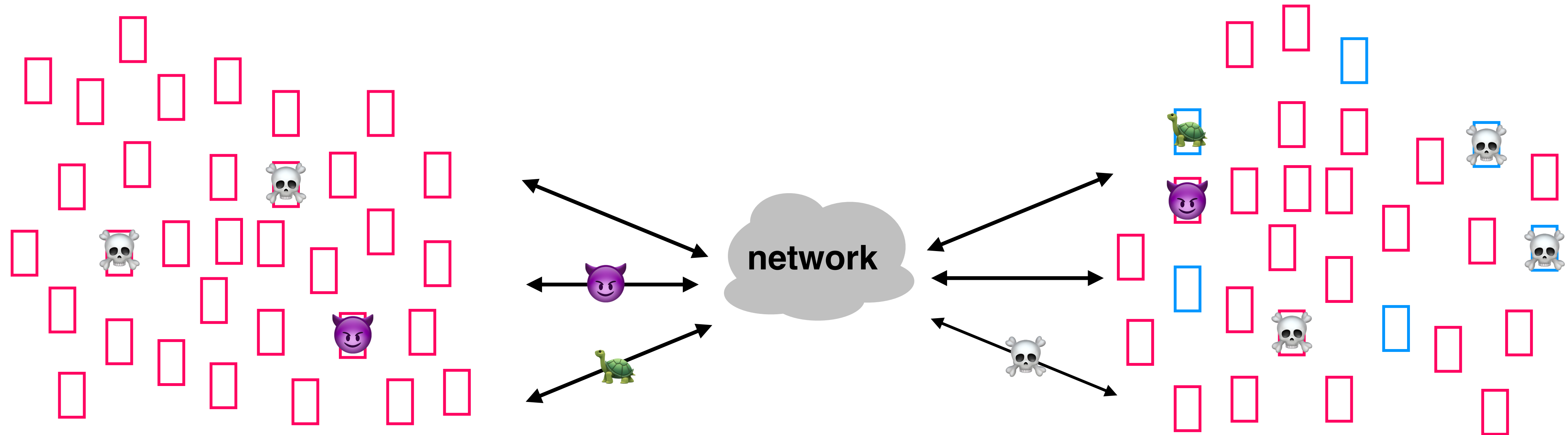
**modularity and abstraction in security:** good modularity and abstraction make it easier to reason about the security of a system





**scalability:** how does our system behave as we increase the number of machines, users, requests, data, etc.?

**fault-tolerance/reliability:** how does our system deal with failures (💀)? machines crashing, network links breaking, etc.



**security:** how does our system cope in the face of targeted attacks (😈)?

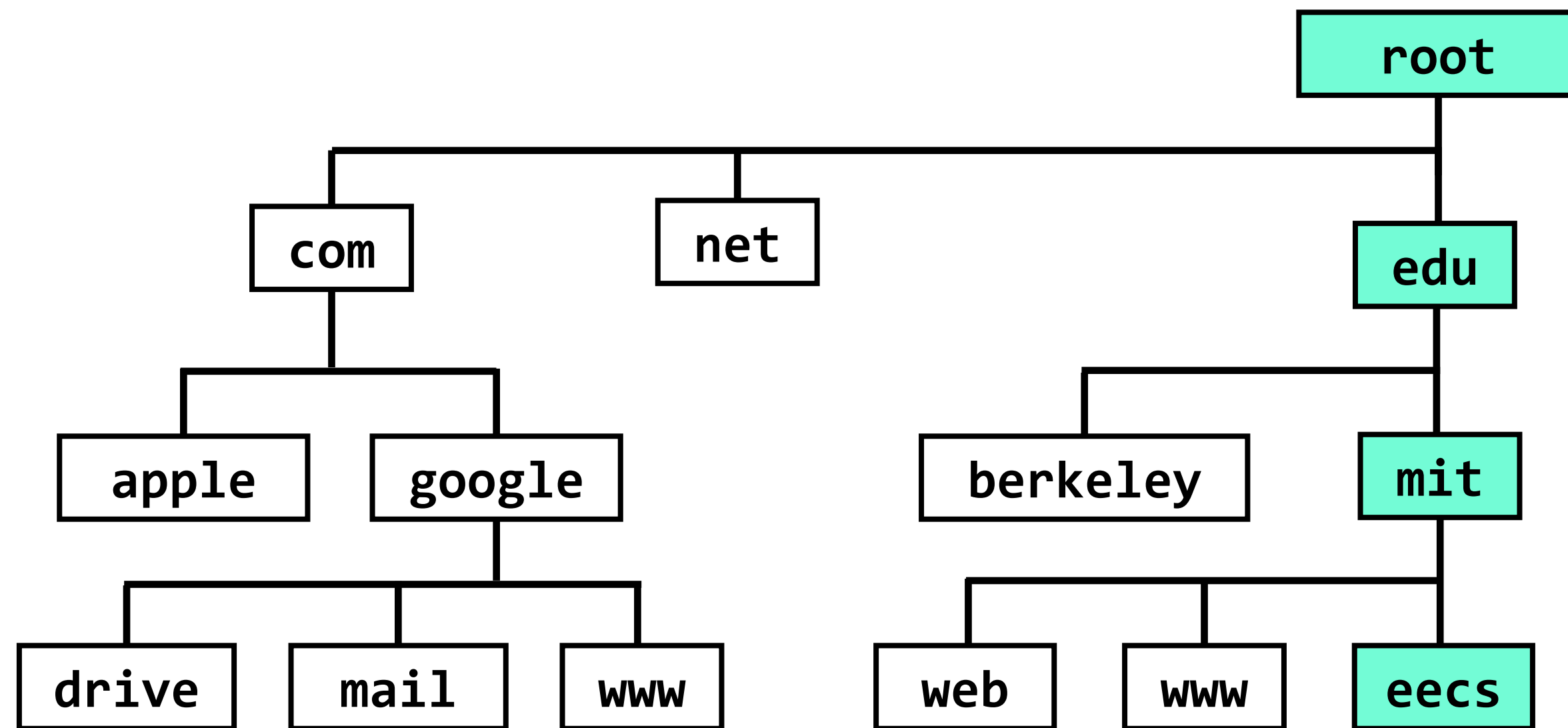
**performance:** how do we define our performance requirements, and know if our system is meeting them? what do we do if performance is subpar (🐢)?

**who is impacted by our design and implementation choices?  
who makes those choices?**



**scalability:** how does our system behave as we increase the number of machines, users, requests, data, etc.?

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

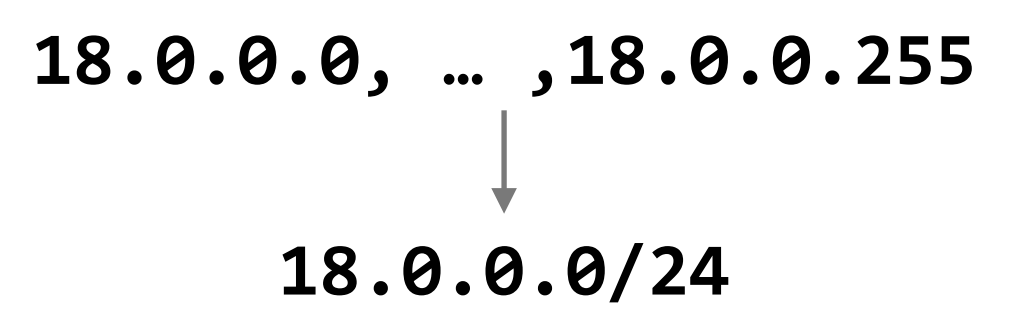
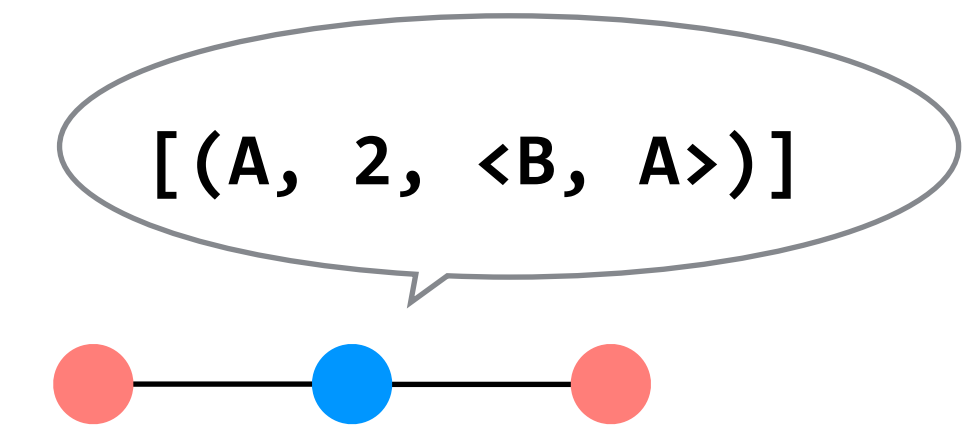
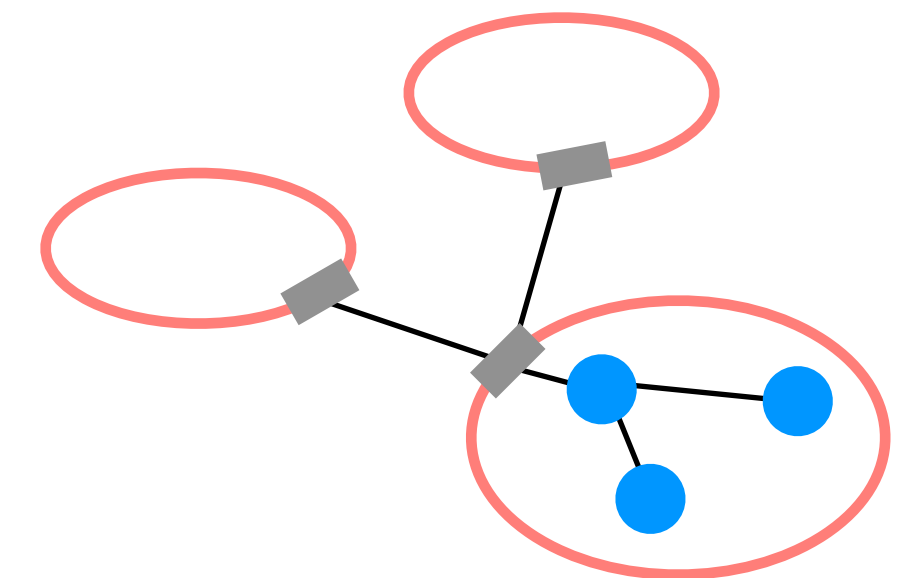
**reliability issue:** what happens when a nameserver fails or (**security issue**) is attacked?

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

**scalability:** how does our system behave as we increase the number of machines, users, requests, data, etc.?

**scalable routing:** a few different things allow us to route across the Internet

1. **hierarchy of routing:** route between ASes, and then within an AS
2. **path-vector routing:** advertisements include the path, to better detect routing loops
3. **topological addressing:** assign addresses in contiguous blocks to make advertisements smaller



**scalability:** how does our system behave as we increase the number of machines, users, requests, data, etc.?

---

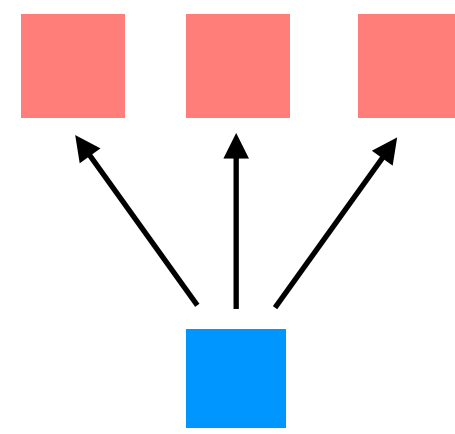
**question:** how can a single reliable sender, using a sliding-window protocol, set its window size to maximize utilization — but prevent congestion and unfairness — given that there are many other end points using the network, all with different, changing demands?



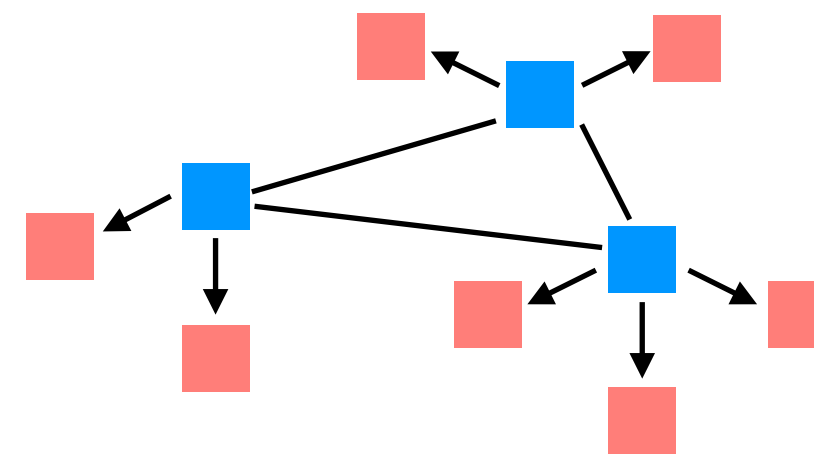
**scalability**: how does our system behave as we increase the number of machines, users, requests, data, etc.?

how do we share a file — or **deliver content** — on the Internet?

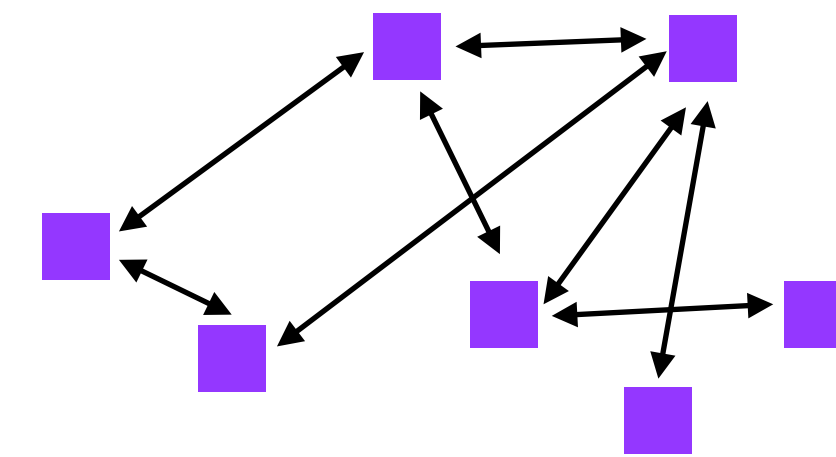
**client-server**



**CDNs**



**P2P**



**more distributed**  
more scalable?

we know that **a client-server model is (relatively) simple, but doesn't scale well**; let's understand more about the other two technologies, to see where they end up in terms of complexity, scalability, etc.

**scalability**: how does our system behave as we increase the number of machines, users, requests, data, etc.?

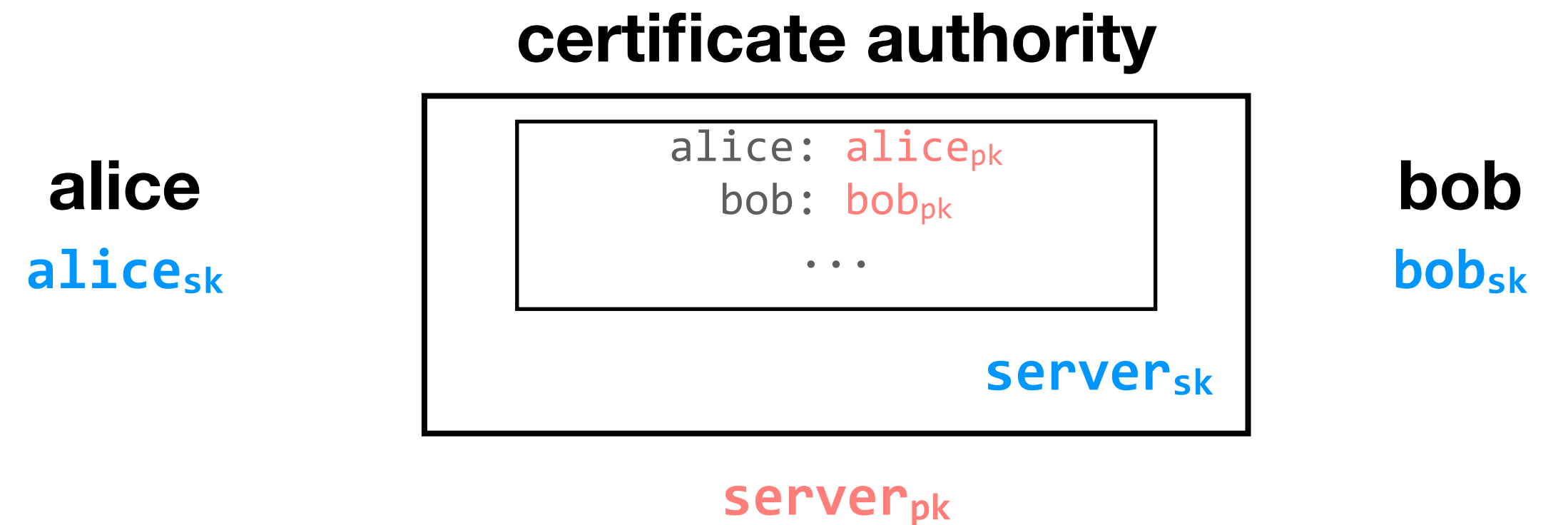
**cryptographic signatures** allow users to verify identities using public-key cryptography

users generate **key pairs**; the two keys in the pair are related mathematically

{**public\_key**, **secret\_key**}

$\text{sign}(\text{secret\_key}, \text{message}) \rightarrow \text{sig}$   
 $\text{verify}(\text{public\_key}, \text{message}, \text{sig}) \rightarrow \text{yes/no}$

**property**: it is (virtually) impossible to compute **sig** without **secret\_key**



server pre-computes **signed** messages that map names to their public keys

$\text{sign}(\text{server}_{\text{sk}}, \text{"alice: alice}_{\text{pk}}\text{"}) \rightarrow \text{sig}$

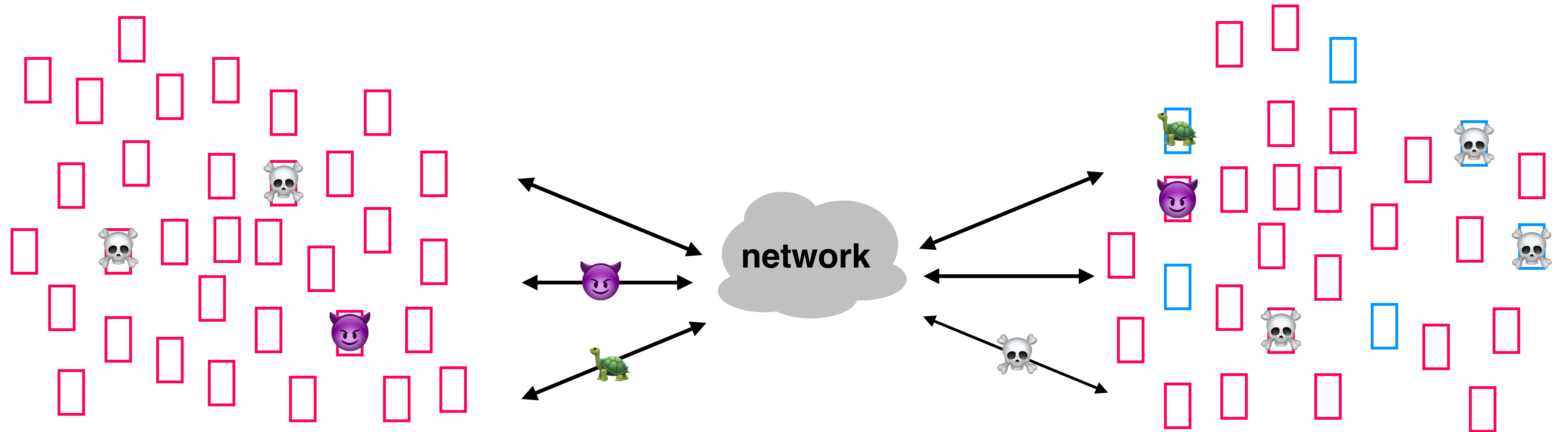
alice, alice<sub>pk</sub>, sig

**certificate**

anyone can verify that the authority signed this message given **server<sub>pk</sub>**, but the server itself doesn't have to distribute the signed messages

**scalability:** how does our system behave as we increase the number of machines, users, requests, data, etc.?

**fault-tolerance/reliability:** how does our system deal with failures (💀)? machines crashing, network links breaking, etc.



**security:** how does our system cope in the face of targeted attacks (😈)?

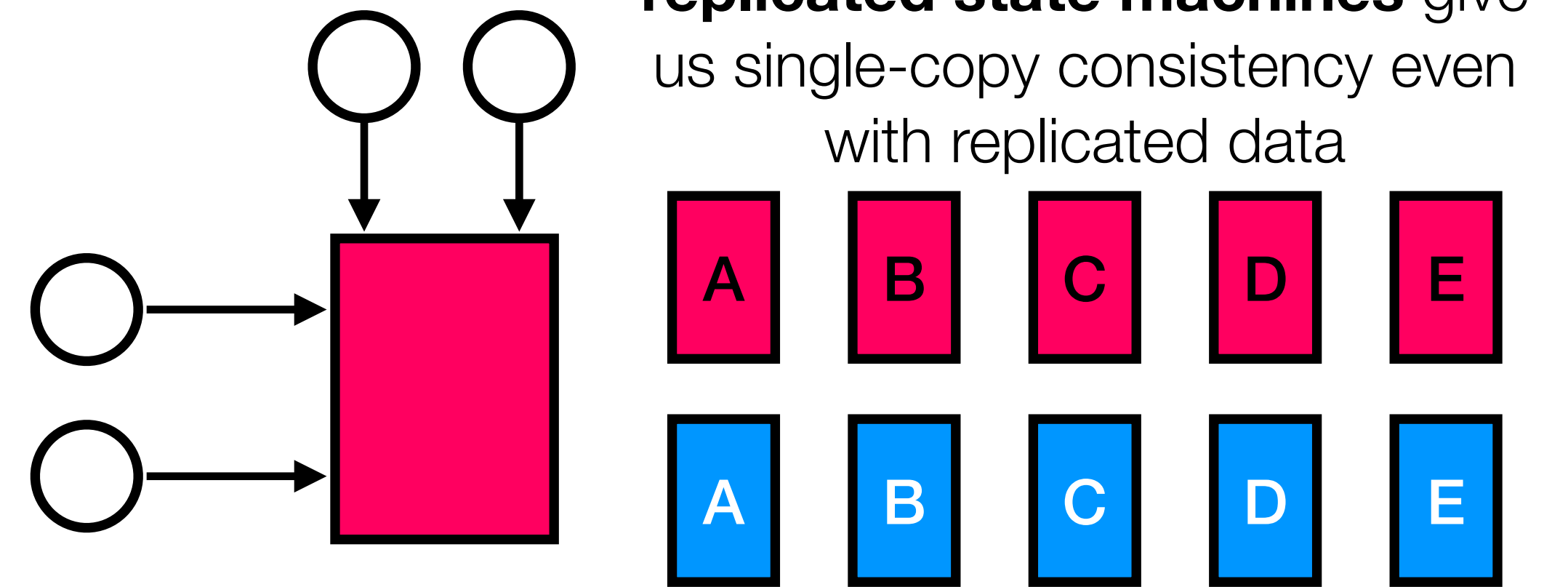
**performance:** how do we define our performance requirements, and know if our system is meeting them? what do we do if performance is subpar (🐢)?

**who is impacted by our design and implementation choices?  
who makes those choices?**



**fault tolerance:** how does our system deal with failures? machines crashing, network links breaking, etc.

our goal is to build **reliable systems from unreliable components**. we want to build systems that serve many clients, store a lot of data, perform well, all while keeping availability high



**transactions** — which provide **atomicity** and **isolation** — make it easier for us to reason about failures

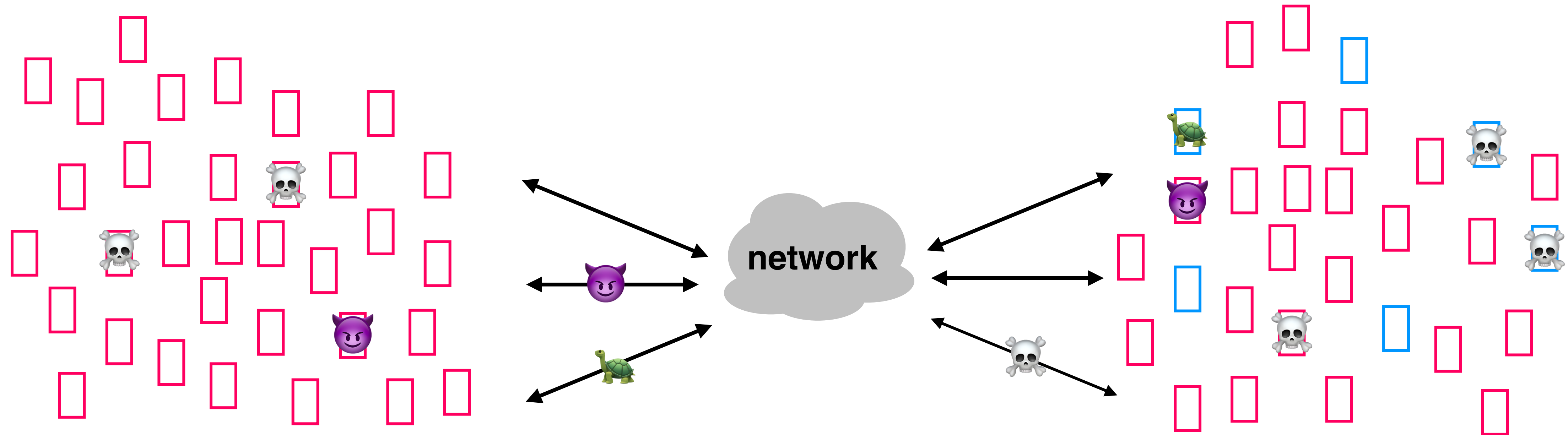
our job in lecture is to understand how a system *implements* these two abstractions.  
how do our systems guarantee atomicity? how do they guarantee isolation?

**atomicity:** provided by **logging**, which gives better performance than shadow copies at the cost of some added complexity; **two-phase commit** gives us multi-site atomicity

**isolation:** provided by **two-phase locking**

**scalability:** how does our system behave as we increase the number of machines, users, requests, data, etc.?

**fault-tolerance/reliability:** how does our system deal with failures (💀)? machines crashing, network links breaking, etc.



**security:** how does our system cope in the face of targeted attacks (😈)?

**performance:** how do we define our performance requirements, and know if our system is meeting them? what do we do if performance is subpar (🐢)?

**who is impacted by our design and implementation choices?  
who makes those choices?**

# security: how does our system cope in the face of targeted attacks?

**policy:** provide **authentication** for users

**threat model:** adversary has access to the entire stored table

**policy:** provide **confidentiality** and **integrity**

**threat model:** adversary can observe network data, tamper with packets, and insert its own packets



**policy:** provide **anonymity**

**threat model:** adversary is on the path between the client and the server

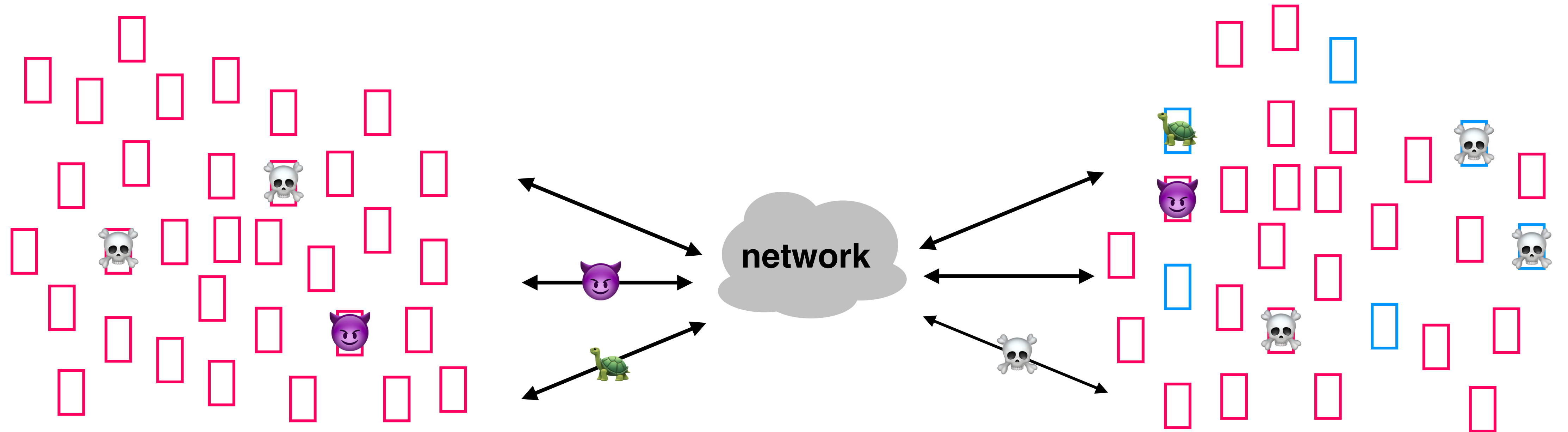
**policy:** maintain **availability** of the service

**threat model:** adversary controls a **botnet**, and is aiming to prevent access to a legitimate service via **DDoS attacks**



**scalability:** how does our system behave as we increase the number of machines, users, requests, data, etc.?

**fault-tolerance/reliability:** how does our system deal with failures (💀)? machines crashing, network links breaking, etc.



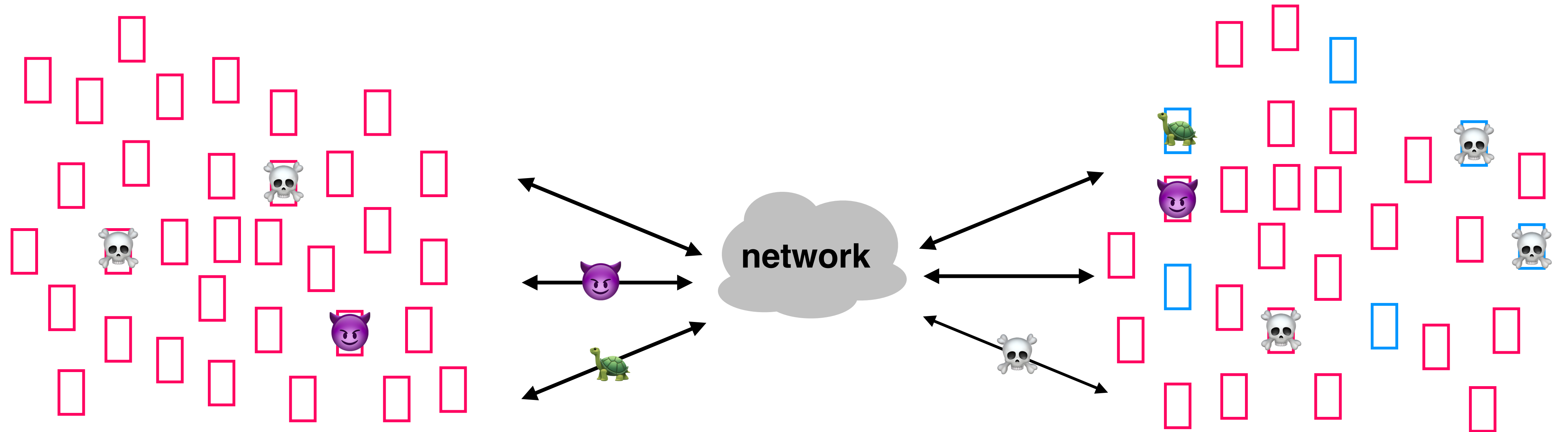
**security:** how does our system cope in the face of targeted attacks (😈)?

**performance:** how do we define our performance requirements, and know if our system is meeting them? what do we do if performance is subpar (🐢)?

**who is impacted by our design and implementation choices?  
who makes those choices?**

**scalability:** how does our system behave as we increase the number of machines, users, requests, data, etc.?

**fault-tolerance/reliability:** how does our system deal with failures (💀)? machines crashing, network links breaking, etc.



**security:** how does our system cope in the face of targeted attacks (😈)?

**performance:** how do we define our performance requirements, and know if our system is meeting them? what do we do if performance is subpar (🐢)?

**who is impacted by our design and implementation choices?  
who makes those choices?**

# who is **impacted** by our design and implementation choices? who makes those choices?

---

TECHNOLOGY

## Internet Outage That Crashed Dozens Of Websites Caused By Software Update

Updated July 22, 2021 · 2:22 PM ET

DEEPA SHIVARAM 

A widespread internet outage caused several major websites to shut down Thursday afternoon, including Amazon, Delta, Capital One and Costco.

Akamai, a content distribution network that helps with the spread of data around the internet, posted on [Twitter](#) that a software configuration update caused a bug in its DNS system.

A DNS, or domain name service, helps match a website's name to its IP address. If the DNS fails, it becomes impossible to search and connect to a website by name.



# who is **impacted** by our design and implementation choices? who makes those choices?

“We are victims of digital colonialism,” Prime Minister Dalton Tagelagi of Niue said over a crackling video link from his office in the capital of Alofi. “This domain, the .nu, recognizes Niue as a sovereign country. This is how important it is to our identity.”

Critics question that assessment, as there is formally no such thing as sovereignty in cyberspace, only administrative zones that divide the web into domains like .nu and, for instance, the .nz [suffix assigned to New Zealand](#).

Winning the case could help ensure the long-term survival of Niue, Mr. Tagelagi said. The island’s population is now about a third of what it was in the 1960s, and the empty homes that dot the island are a reminder of the people who left for better economic opportunities. A victory could help fund its bid to join the United Nations, similar to how Tuvalu obtained U.N. membership after monetizing .tv.

If Niue manages to get .nu back, it could bring in up to \$2 million in revenue a year, according to Par Brumark, a domain name expert who is acting on Niue’s behalf in the Swedish case.

<https://www.nytimes.com/2024/02/21/world/asia/niue-nu-domain-sweden.html>

in the case of DNS, names have meaning outside of the system, even if they were only originally intended to denote “administrative zones that divide the web into domains”



# who is **impacted** by our design and implementation choices? who makes those choices?

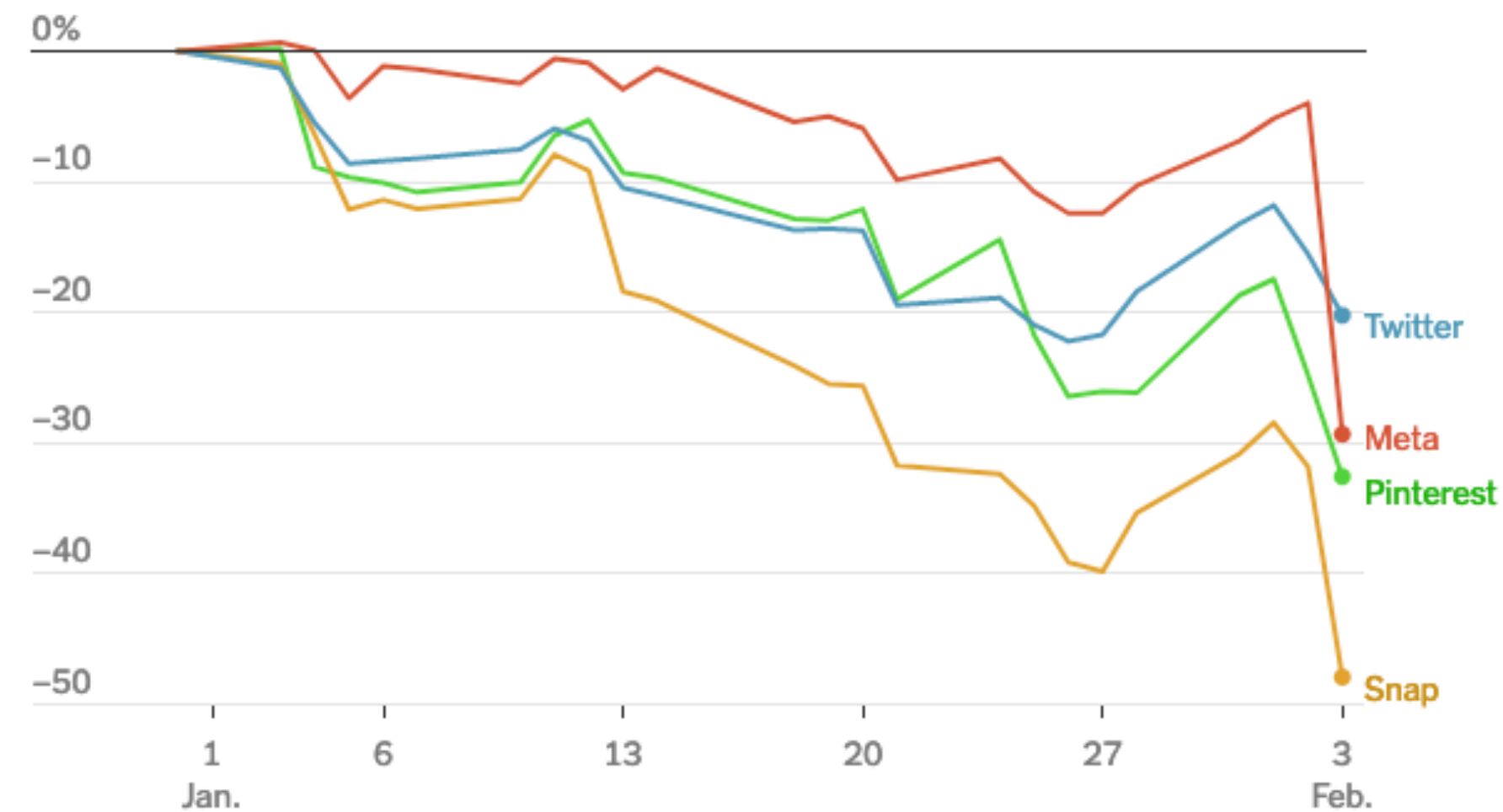
system design choices  
impact more than just that  
system's users

## *A Change by Apple Is Tormenting Internet Companies, Especially Meta*

Meta's stock prices plunged after the company reported that Apple's privacy features would cost it billions this year. It's not the only tech giant to take a hit.



Year-to-date change in stock price of major tech companies



Source: FactSet • By The New York Times

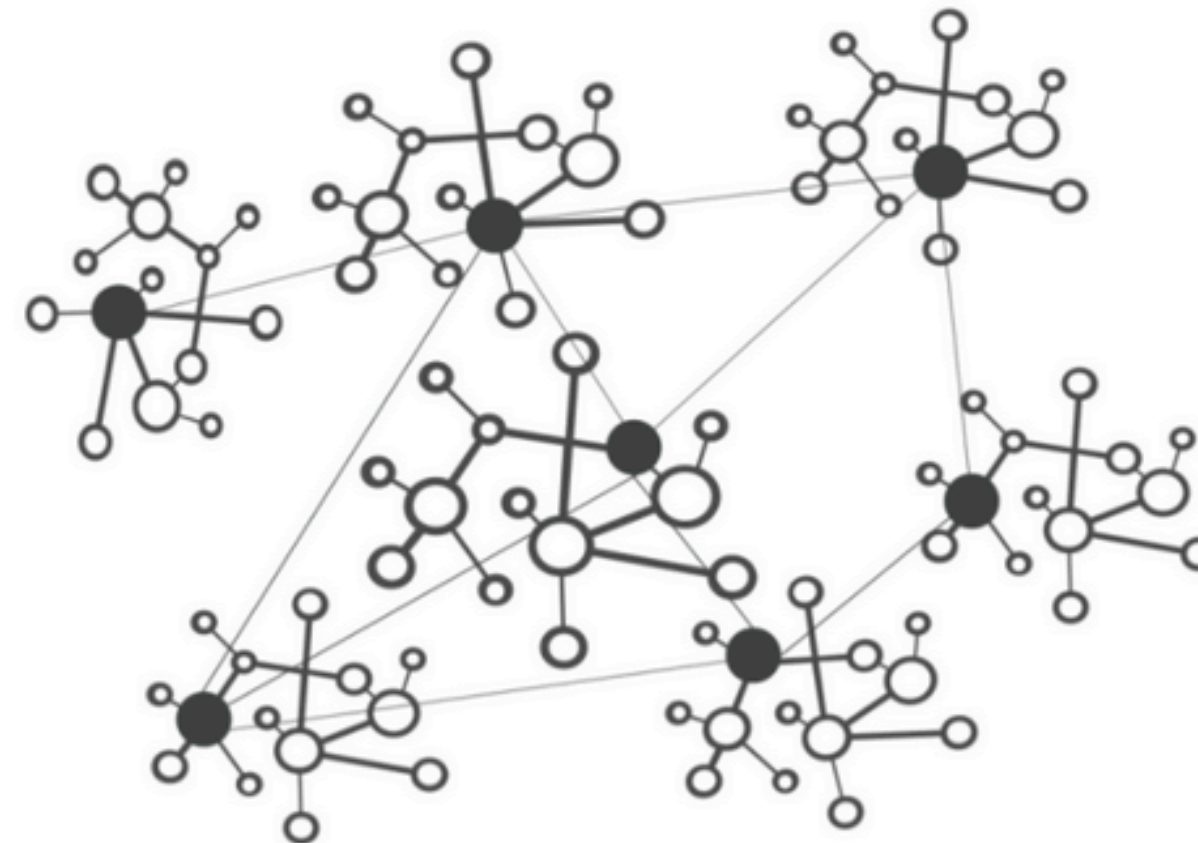
# who is **impacted** by our design and implementation choices? who makes those choices?

## Understanding How Facebook Disappeared from the Internet

10/04/2021

 Celso Martinho  Tom Strickx

This post is also available in [简体中文](#), [繁體中文](#), [日本語](#), [한국어](#), [Deutsch](#), [Français](#), [Español](#), [Português](#), [Русский](#), and [Italiano](#).



The Internet - A Network of Networks

"Facebook can't be down, can it?", we thought, for a second.




# who is **impacted** by our design and implementation choices? who makes those choices?

<https://www.nytimes.com/2023/04/21/us/spacex-rocket-dust-texas.html>

## *SpaceX's Starship Kicked Up a Dust Cloud, Leaving Texans With a Mess*

Residents of Port Isabel said that their city was covered in grime following SpaceX's rocket launch on Thursday. The city said there was no "immediate concern for people's health."

 Give this article





# who is **impacted** by our design and implementation choices? who makes those choices?

<https://www.nytimes.com/2023/04/09/business/bitcoin-mining-electricity-pollution.html>

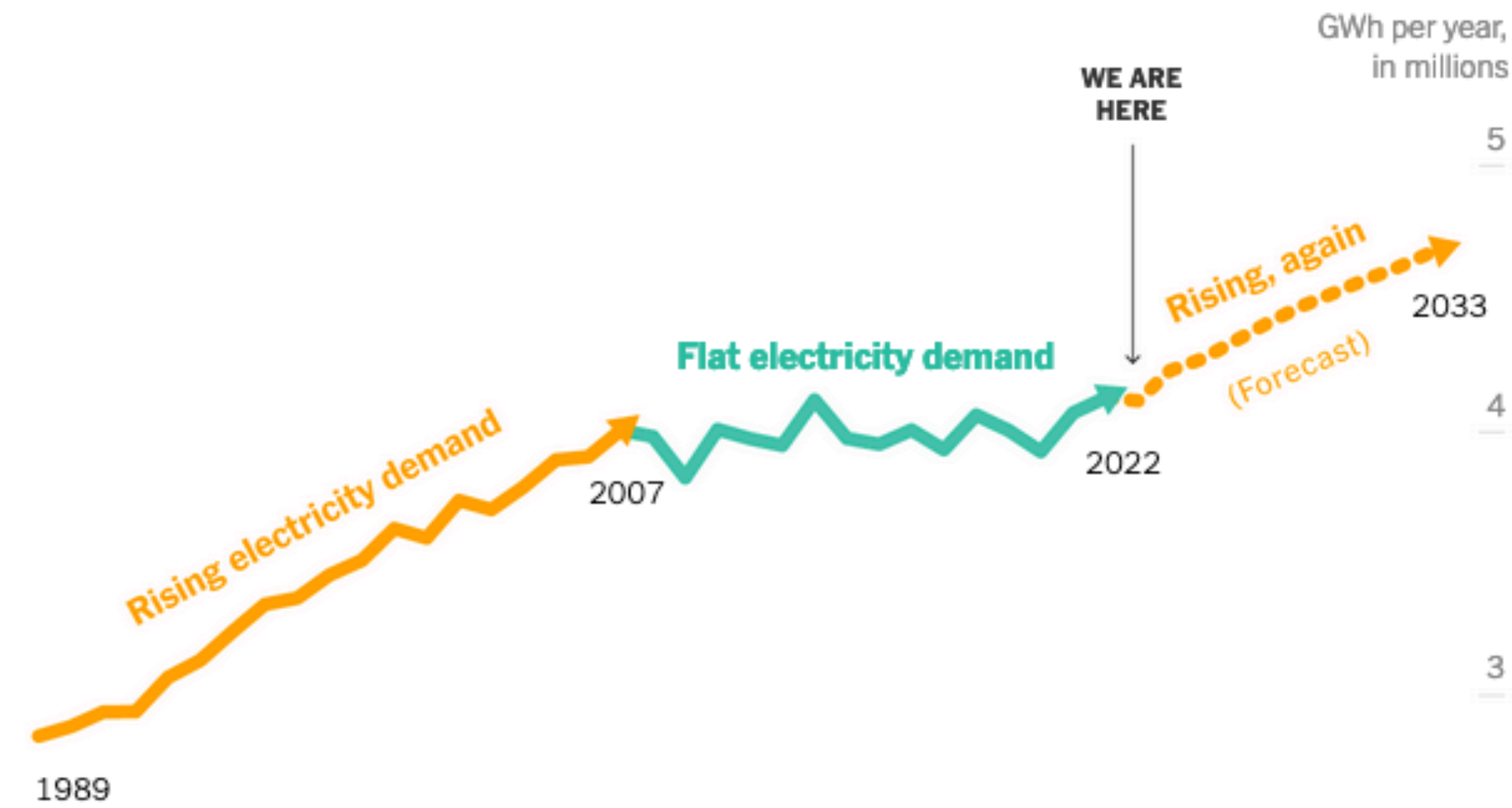
The New York Times has identified 34 such large-scale operations, known as Bitcoin mines, in the United States, all putting immense pressure on the power grid and most finding novel ways to profit from doing so. Their operations can create costs — including higher electricity bills and enormous carbon pollution — for everyone around them, **most of whom have nothing to do with Bitcoin.**

## *The Real-World Costs of the Digital Race for Bitcoin*

Bitcoin mines cash in on electricity — by devouring it, selling it, even turning it off — and they cause immense pollution. In many cases, the public pays a price.



# who is **impacted** by our design and implementation choices? who makes those choices?



## A New Surge in Power Use Is Threatening U.S. Climate Goals

A boom in data centers and factories is straining electric grids and propping up fossil fuels.

By **Brad Plumer** and **Nadja Popovich** March 14, 2024

# who is **impacted** by our design and implementation choices? who makes those choices?

## 'Denial of service condition' disrupted US energy company operations



Zack Whittaker @zackwhittaker / 4 days ago

Comment



who is **impacted** by our design and implementation choices?  
who makes those choices?

---

# Data Broker Is Selling Location Data of People Who Visit Abortion Clinics

It costs just over \$160 to get a week's worth of data on where people who visited Planned Parenthood came from, and where they went afterwards.



By Joseph Cox



# who is **impacted** by our design and implementation choices? who makes those choices?

## Tracking Internet Shutdowns in 2023



**Robbie Mitchell**

Senior Communication and Technology Advisor,  
Internet Society

Categories:

Shutdown



---

January 11, 2024

It is increasingly common for governments to shut down the Internet on a national or sub-national level to solve specific problems, including controlling civil unrest, stemming the flow of misinformation, or preventing cheating on national exams.

As of the end of 2023, governments and other actors across 18 countries intentionally disrupted Internet connectivity or blocked access to specific Internet services for their citizens. Of the 124 events Pulse tracked across the year, including four that continued from last year, 55 have been nationwide disruptions lasting from a couple of hours to a week, culminating in more than 2,370 days of disruptions.

so much of life today relies on the Internet — so much so that Internet shutdowns are sometimes used as tools of oppression



# who is **impacted** by our design and implementation choices? who makes those choices?

---

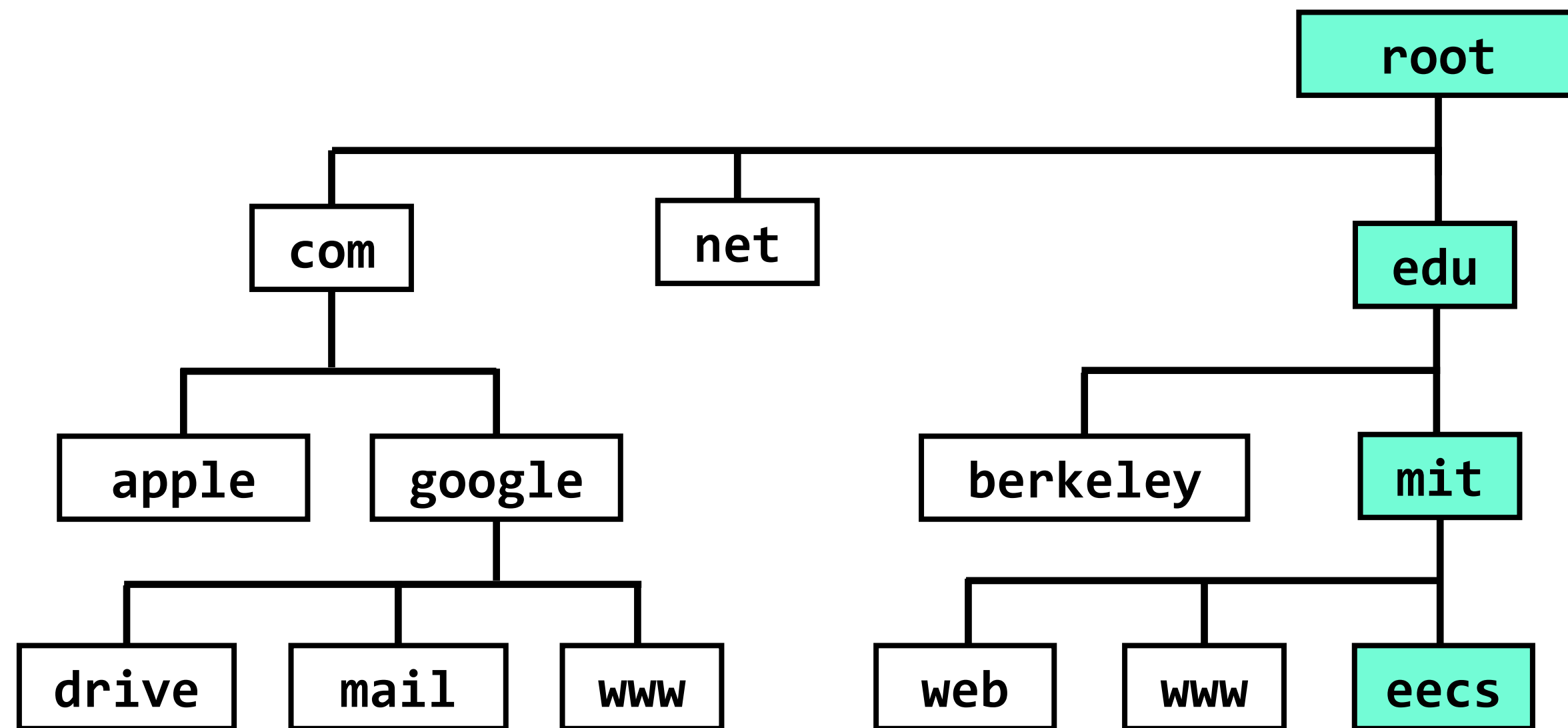
Section 230 (the legal provision in question here) doesn't have as much to do with the Internet's architecture as other laws (e.g., net neutrality), but the question remains: **who gets to make these decisions?**

***Gonzalez v. Google***, the case heard today, could subject social media websites and even search engines to ruinous liability, potentially forcing these companies to abandon their business models or even shut down.

That said, most of the justices appeared sufficiently spooked by the possibility that they could destroy how the modern-day internet operates that they are likely to find a way to prevent that outcome. **As Justice Elena Kagan warned at one point during the *Gonzalez* argument, the justices are "not the nine greatest experts on the internet."** So it makes sense for them to approach a case that could fundamentally change how foundational websites operate with a degree of humility.

# who is **impacted** by our design and implementation choices? who makes those choices?

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

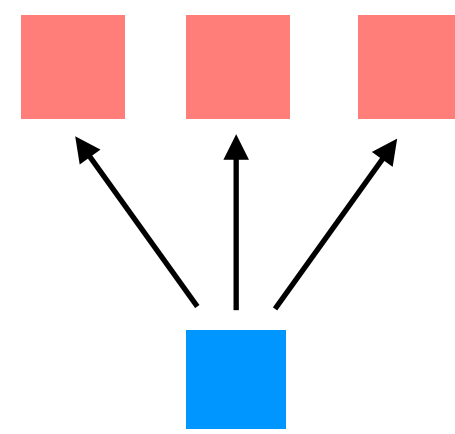
**reliability issue:** what happens when a nameserver fails or (**security issue**) is attacked?

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

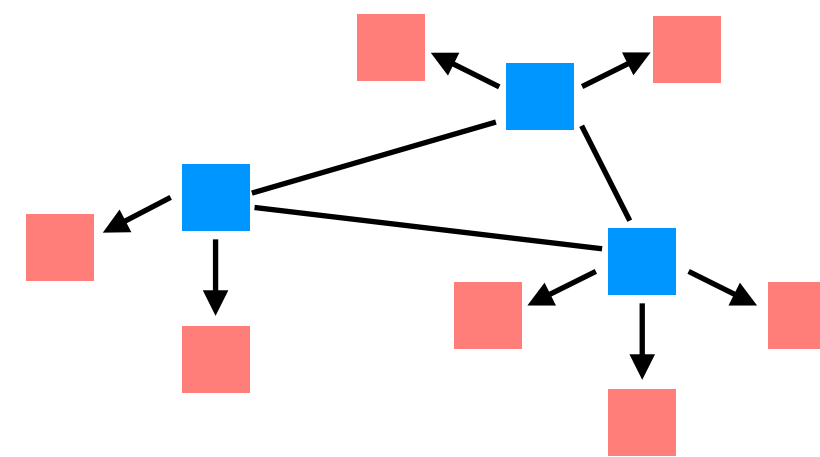
# who is **impacted** by our design and implementation choices? who makes those choices?

how do we share a file — or **deliver content** — on the Internet?

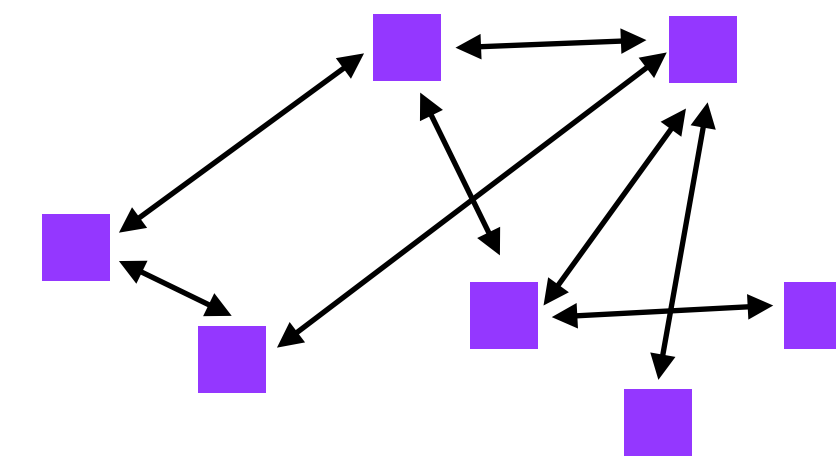
**client-server**



**CDNs**



**P2P**



**more distributed**  
more scalable?

# who is **impacted** by our design and implementation choices? who makes those choices?

type of management	what does this type of management allow a switch to do	example protocols	how the protocol works	pros/cons?
<b>Queue Management</b>	signal congestion, potentially before queues are full	DropTail	drop packets when the queue is full	simple, but queues get full (among other problems)
		RED, ECN	drop or mark packets before the queue is full	can keep queues from filling up, but complicated
<b>Delay-based Scheduling</b>	prioritize latency-sensitive traffic	Priority Queueing	serve some queues before others	prioritized queues can starve out the others
<b>Bandwidth-based Scheduling</b>	enforce (weighted) fairness among different types of traffic	Round-robin	try to give each type of traffic an equal share of bandwidth	can't handle variable packet sizes
		Weighted Round-robin	round robin, but incorporate average packet size	average packet size hard to get
		Deficit Round-robin	round robin, but do a better job with packet sizes	honestly pretty good

## is in-network resource management a good idea on the Internet?



# who is **impacted** by our design and implementation choices? who makes those choices?

## VIII. PERSPECTIVE

Perhaps paradoxically, the success of the Unix system is largely due to the fact that it was not designed to meet any predefined objectives. The first version was written when one of us (Thompson), dissatisfied with the available computer facilities, discovered a little-used PDP-7 and set out to create a more hospitable environment. This (essentially personal) effort was sufficiently successful to gain the interest of the other author and several colleagues, and later to justify the acquisition of the PDP-11/20, specifically to support a text editing and formatting system. When in turn the 11/20 was outgrown, the system had proved useful enough to persuade management to invest in the PDP-11/45, and later in the PDP-11/70 and Interdata 8/32 machines, upon which it developed to its present form. Our goals throughout the effort, when articulated at all, have always been to build a comfortable relationship with the machine and to explore ideas and inventions in operating systems and other software. We have not been faced with the need to satisfy someone else's requirements, and for this freedom we are grateful.

Three considerations that influenced the design of Unix are visible in retrospect.

**First: because we are programmers, we naturally designed the system to make it easy to write, test, and run programs.** The most important expression of our desire for programming convenience was that the system was arranged for interactive use, even though the original version only supported one user. We believe that a properly designed interactive system is much more productive and satisfying to use than a "batch" system. Moreover, such a system is rather easily adaptable to noninteractive use, while the converse is not true.

**Second:** there have always been fairly severe size constraints on the system and its software. Given the partially antagonistic desires for reasonable efficiency and expressive power, the size constraint has encouraged not only economy, but also a certain elegance of design. This may be a thinly disguised version of the "salvation through suffering" philosophy, but in our case it worked.

**Third:** nearly from the start, the system was able to, and did, maintain itself. This fact is more important than it might seem. If designers of a system are forced to use that system, they quickly become aware of its functional and superficial deficiencies and are strongly motivated to correct them before it is too late. Because all source programs were always available and easily modified on-line, we were willing to revise and rewrite the system and its software when new ideas were invented, discovered, or suggested by others.

The aspects of Unix discussed in this paper exhibit clearly at least the first two of these design considerations. The interface to the file system for



# who is **impacted** by our design and implementation choices? who makes those choices?

## *Sally Floyd, Who Helped Things Run Smoothly Online, Dies at 69*

In the early 1990s, Dr. Floyd was one of the inventors of Random Early Detection, which continues to play a vital role in the stability of the internet.



Sally Floyd. “Her work on congestion control,” a colleague said, helped keep the internet “working for everyone.” Carole Leita

One byproduct of Dr. Floyd’s work reflected her passion for keeping things fair to all internet users. “Her work on congestion control was about keeping it working for everyone,” Dr. Kohler said. “For people with fast connections, and for people with slow connections.”

<https://www.nytimes.com/2019/09/04/science/sally-floyd-dead.html>



# where to go next

**6.1850**<sub>6.S057/6.052</sub> - Computer Systems and Society

this class will also have a graduate version next year, I'm just not sure of the subject number yet

**6.1810**<sub>6.039</sub> - Operating Systems

**6.5820**<sub>6.829</sub> - Computer Networks

**6.5830/6.5831**<sub>6.830/6.814</sub> - Database Systems

**6.5840**<sub>6.824</sub> - Distributed Systems

**6.5850**<sub>6.826</sub> - Principles of Computer Systems

**6.1600**<sub>6.S060/6.053</sub> - Foundations of Computer Security

**6.5660**<sub>6.858</sub> - Computer Systems Security

**6.5610**<sub>6.857</sub> - Network and Computer Security

**6.5620**<sub>6.875</sub> - Cryptography and Cryptanalysis

more systems  
↓  
more math