University of Illinois at Urbana-Champaign
Dept. of Electrical and Computer Engineering

ECE 101: Exploring Digital Information Technologies for Non-Engineers Spring 2023

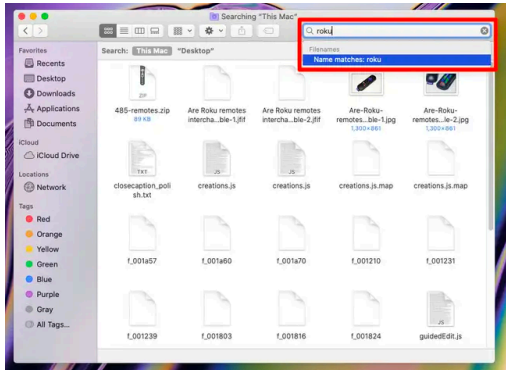
File Systems: Consistency and Cloud Storage

How do you store documents on your computer?

- On the Desktop
- In the Downloads folder
- Somewhere on your computer
- In a carefully organized folder structure, with folders names like “ECE101”, “LAS122”, “PSYC100”, etc.
- On the cloud: Google Drive, iCloud, Dropbox etc.

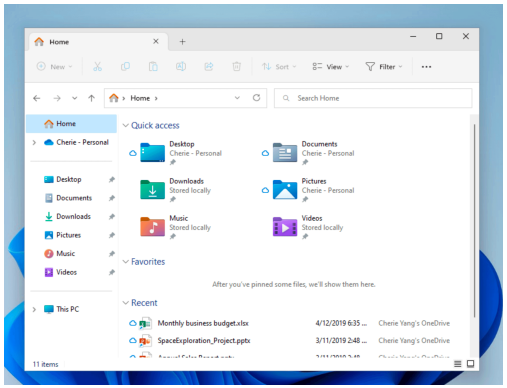
How do you retrieve your files?

- Navigate to the right folder - you know exactly where you saved it.
- Search by keyword and date



Do you worry about space and organization?

- Technology has brought us **effectively unlimited virtual storage**
- Also enables us to **collaborate remotely** on shared documents.



Getting to Your Files in the Cloud

Cloud services are assumed to have several properties.

We **assume** that our files in the cloud are

- **available** (accessible at all times),
- **reliable** (no errors in stored content), and
- **consistent** (everyone sees the same thing, all the time).

But first, just to make sure you're aware…

All Services Provided "As Is", Without Warranty

None of these companies guarantees you anything—certainly not for free.*

If they decide to stop doing business with you,
- whether because it's not in their interest or
- because they go out of business, or
- they sell to another company who doesn't want your business,
- your data are gone.

You may want to have a copy somewhere?

*If you pay, I suggest reading the fine print carefully.

Working on a Shared Document

To get started,
let's think about what it means
to do something to a document.

Let's call the act of doing something an operation.

Each Operation with an Inverse Can be Undone

Pressing a key is an operation.

Hitting **backspace/delete undoes that operation** (removes the effect of the keystroke).

Word processing **programs generalized the idea** of undoing the last operation.

- You usually type stuff into a document
- If you type in something you didn't mean to, you delete it
- If you accidentally deleted something you didn't mean to, you "UNDO" it, by reinserting the word.

A History of Operations needed to Undo

To support "Undo", **programs must keep a log of operations**.

Each log entry

- **could be undone** (in reverse order)
- by inverting the operation:
- paste becomes cut, and
- key-press becomes backspace.

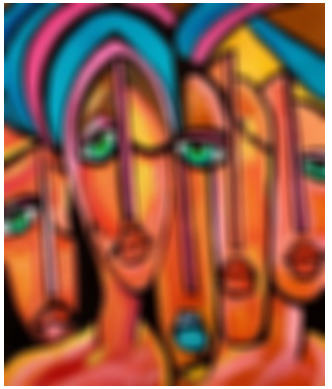
LOG
1.Pressed 'H'
2.Pressed 'e'
3.Pressed 'l'
4.Pressed 'l'
5.Pressed 'o'
6.Pasted ", world!"

Not All Operations can be Inverted

Some operations have no exact inverse.

Consider an image editing tool.
- A user blurs the image,
- which performs local averaging.

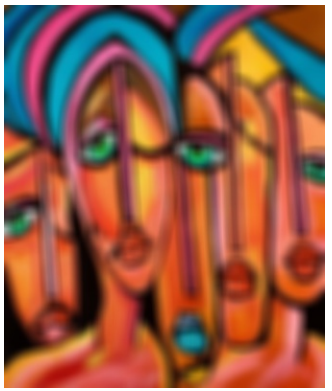
Many possible original images
- produce the same final image, so
- **the only way to undo** a blur
- **is to preserve a copy** of the original.



Cheap Memory Broadens Set of Reversible Operations

In early systems,
- operations of this type could not be undone:
- keeping a copy was too expensive.

As memory became cheaper,
- **programs** started **keep**ing **snapshots**—
- copies of earlier versions of the user's data—
- so as **to support undo**.

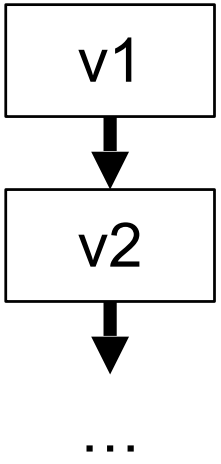


Versioning Useful for Long-Term

An **undo log rarely persists** across sessions.

Example: open a file and undo the last operation … from a week ago?

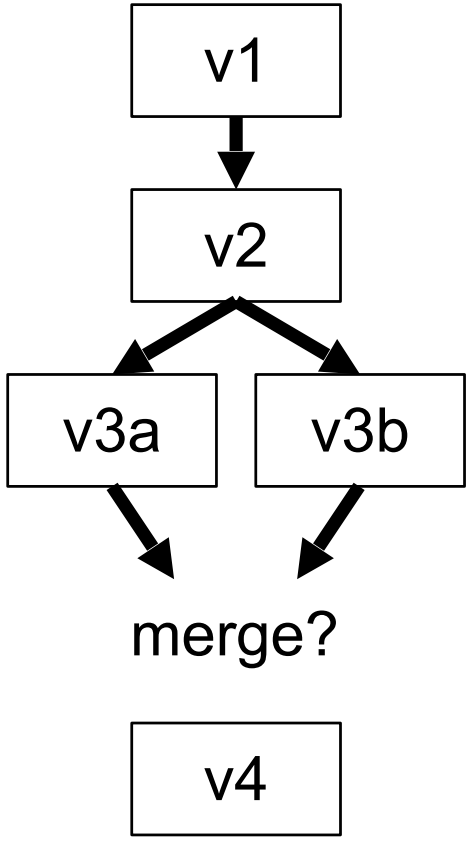
Instead, create **snapshots / versions** are more useful over the longer term.



Merging Alternative Versions Can be Challenging

But long-term storage
introduces the problem of divergent versions…

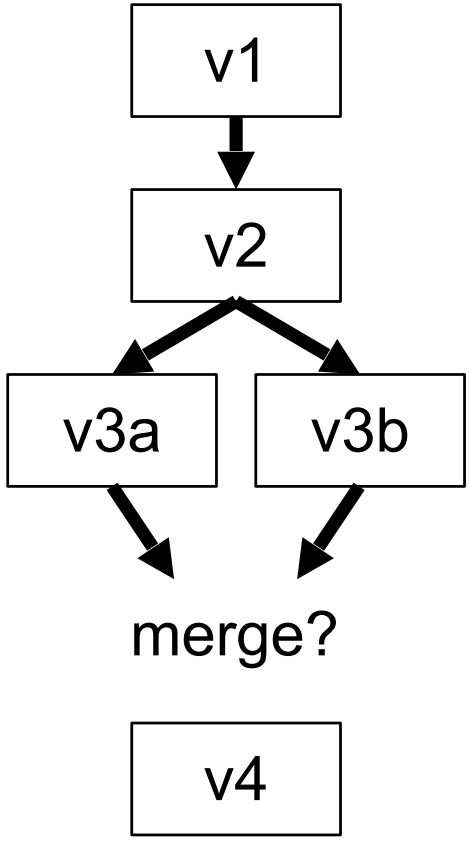
How can we merge changes?



Efforts to Automatically Merge Started Long Ago

By the late 80s,
- the Internet had reached many universities,
- but using it required one
- to be physically on a campus
- (in other words, no Wifi, no cell coverage).

Researchers started trying to address
- the problem of merging updates
- from users working in disconnected mode,
- then returning with separate new versions of a document.

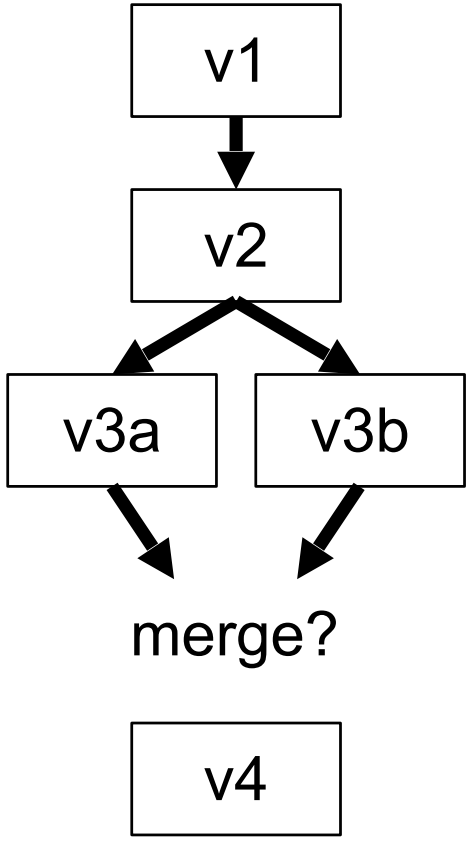


Automatic Merges Fail Frequently

Making merge work without human oversight is difficult.

Even today,

- **shared code repositories** such as Github
- **support** independent, **disconnected development**,
- and **automated merges** do sometimes **fail**.



Order of Operations Matter in Final Result

Consider banking operations.

Your bank account contains **$5**.

You have a **$500** paycheck to deposit.

And you want a **$7** bubble tea.

You decide

- to deposit the paycheck (requires communication from ATM or bank to your account about your deposit),
- then buy a bubble tea (requires communication from Bubble Tea store to your account).

Two operations to be merged.

Addition, the Underlying Operator, is Commutative

Operations to be done on your account at the bank server …

1. Add **$500** to balance.
2. Subtract **$7** from balance.

These operations can be reordered.

However, reordering may have side effects.

Reordering Operations is Not Always Helpful

In particular, many banks,

- seeing the "Subtract" operation first,
- approve the operation
- **but** also subtract an additional overdraft "protection"* fee of **$50**.

In the end, your account has

5 – 7 – 50 + 500 = $448.

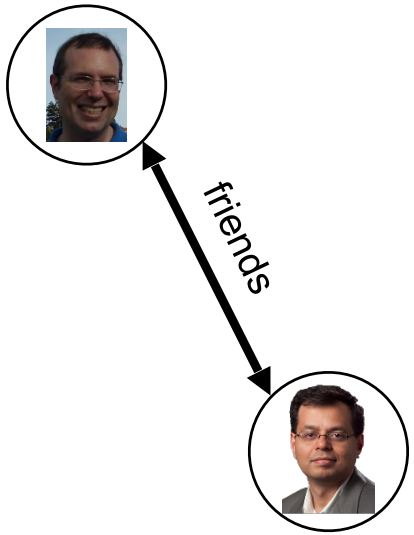
Some Operations are Idempotent

- idem- = identical, -potent = power
- meaning that an **operation has the same effect no matter how many times one performs the operation**.

In a social network, for example:

Mark X as friend of Y

If I switch machines (phone/tablet/computer), see an old version, and apply the same operation again, no problem. Same result again.



Many Tools Do Not Provide Strong Consistency

But … wait a minute.

Why would I ever "see an old version?"

If a service is showing me data,
- which version do I see?
- which version do others see?

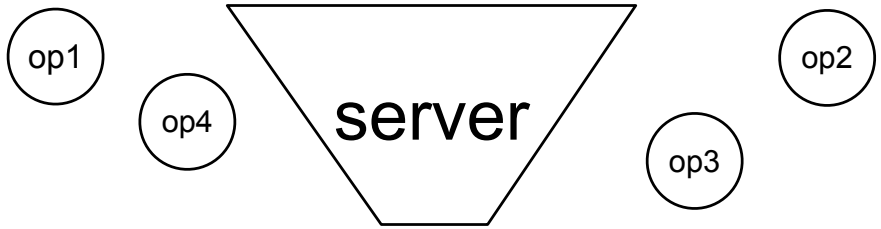
Shouldn't those questions have the same answer at all times?

Yes, of course we'd like that. Life would be easy.

But **that's not our universe**.

One Simple Approach: Pick a Place for the Correct Version

Often, the best solution
- is to define the **version at some server**
- (in one place)
- to be **the correct one**.



To make changes, send operations to that server.

The **server**
- **serializes operations** into some order
- and applies them one at a time.

Small Locales May Never Expose Problems

You may never notice the delay.

Imagine that you and a friend work on a document.

The "correct" version is **40 msec** from each of you.

The server applies any change in **20 msec**.

Each person sees any change in **100 msec**.

Still Possible, But Problems Unlikely to Show Up

It's still ***possible*** that

- when you type an 'X' and
- your friend types a 'Y' and
- the server decides that the 'Y' was first
- **that you** could **observe an inconsistency**,
- but you'd probably just assume that your friend had moved their cursor in front of your 'X' before typing.

More Distance Increases the Chance of Problems

On the other hand,

- if you work with a friend on the opposite side of the Earth
- (say in China if you're in the US, or Europe if you're in Argentina),
- the **added delay makes inconsistency**
- much **more likely** to come to your notice.

Companies Design Operations to Reduce Inconsistency

Companies need to think carefully
- about how to **formulate operations**
- so as to make them **less prone to errors**
- **and** less likely to lead to obviously **inconsistent behavior**.

Inconsistencies Do Still Occur with Many Tools

You've probably still seen cases.

For example, a social network that simply rejects your comment or reply.

Yet, when you try again, everything works fine.

Such failures

- **sometimes indicate problems**
- **with merging your operation**
- with others that have already been applied.

Lack of Consistency Can Lead to Bigger Problems

Realistically, most people do not often work interactively with others around the world.

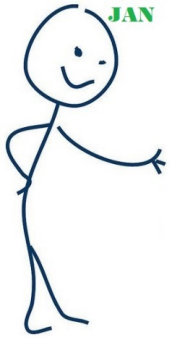
Inconsistency can lead to more **serious problems**, however.

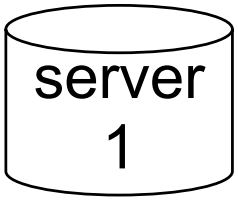
Older Models Provided Little or No Consistency

For example,
- the original Internet "news" system
- used several servers,
- each with a copy of the news.

When a user made a post,
- the text was sent to all servers,
- but the latency to each server varied,
- and sometimes copies were lost
- (and then took even longer to arrive).

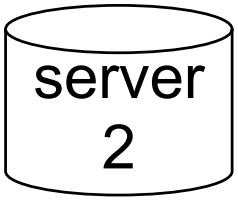
What Can Go Wrong? Let's See an Example





(you'll see the content later)

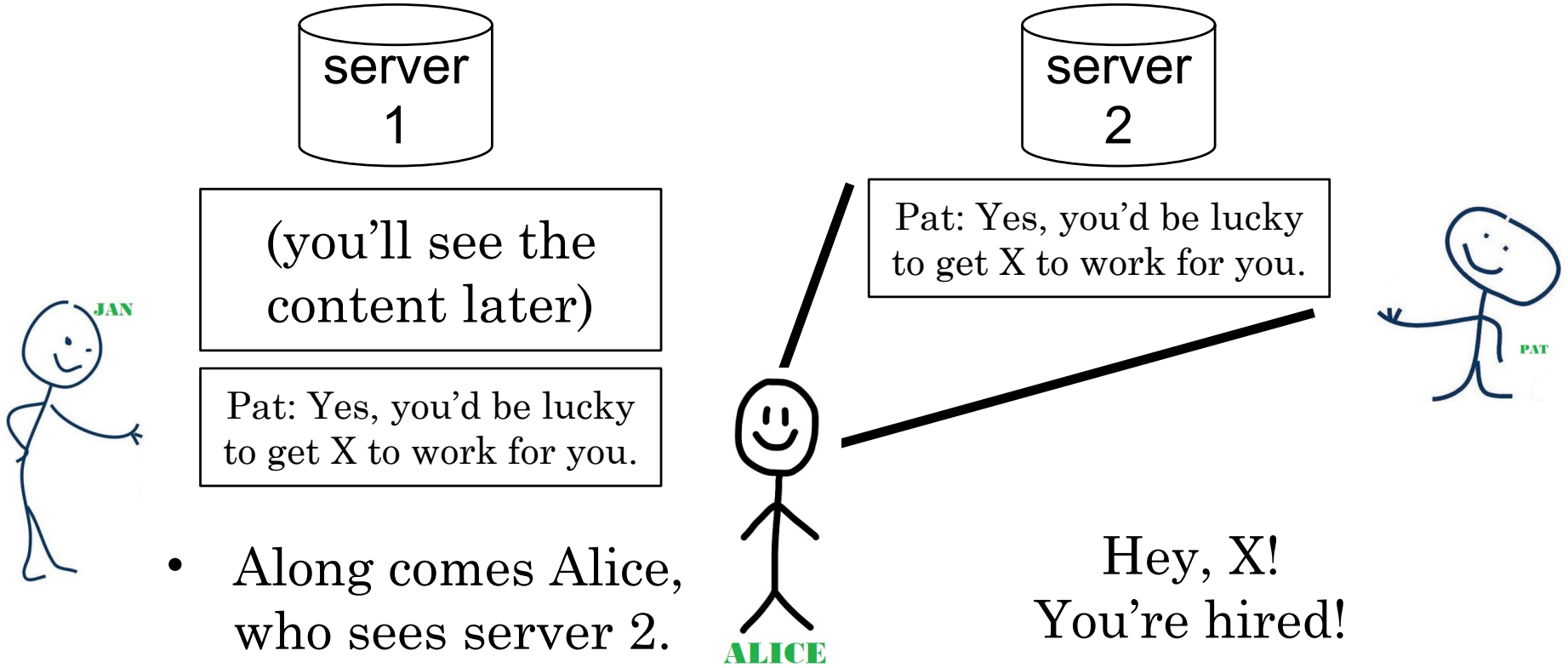
Pat: Yes, you'd be lucky to get X to work for you.



Pat: Yes, you'd be lucky to get X to work for you.

- Jan makes a post.
- Unfortunately, server 2's copy is lost.
- Pat looks at server 1 and replies.

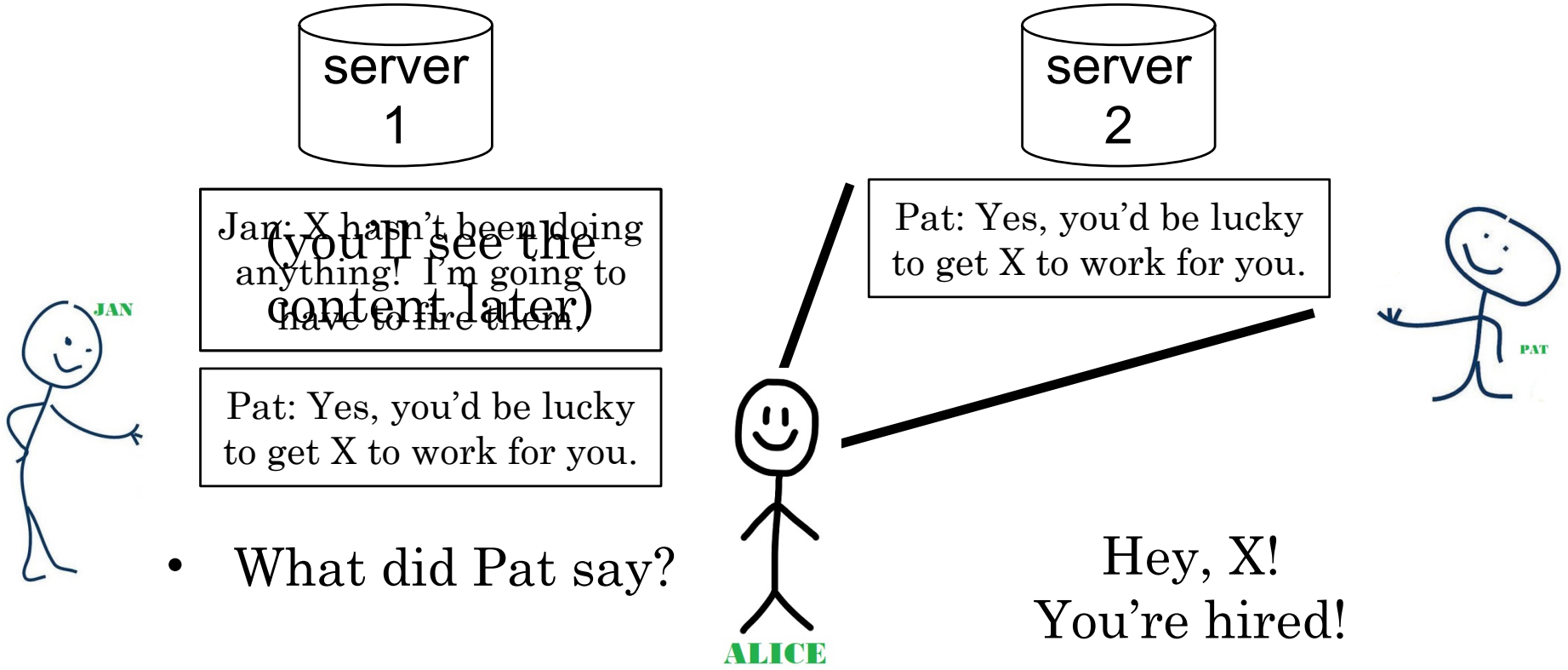
Alice Unknowingly Acts on Incomplete Information



- Along comes Alice, who sees server 2.

Hey, X!
You're hired!

Pat's Words Change the Meaning of Jan's!



- What did Pat say?

Hey, X!
You're hired!

Allowing Inconsistency Reduces User Response Times

So why does anyone tolerate inconsistency?

For speed!

Example of Requiring Strong Consistency

Consider the following scenario:

A "social influencer" makes a post.

10,000 followers want to comment.

Assume **100 msec** to make a change and see a response.

But we want consistency:
- you can't add a comment
- until you've seen all previous comments.

Each comment takes **100 msec** to process.

End Result: No One Wants the Service!

Server can only process one at a time!
- The server adds the first comment.
- Then tells all followers about it.
- **100 msec** have passed.

Now the second follower can comment.
- The server tells all followers.
- **200 msec** have passed.

When it's done,
- **10,000 × 0.1 sec = 1,000 sec = 16.7 minutes** have passed
- And 9,900 followers have given up and joined a different social network!

No One Really Cares that much about Consistency

In a social network,

- most followers don't actually care
- about other followers' comments,
- nor about the order of their comment
- relative to those of other followers.

Most Companies' Systems Do Try to Avoid Inconsistency

The **systems do** make some **attempt to avoid inconsistency**.

For example,

- one can reply to comments, and
- replies are only visible if the original comment is visible.

So long as users make use of the features, **causality violations are less likely**.

Desired Properties of Cloud Service

Here are some of the properties that **we want** with our cloud storage and editing tools:

- **available** (accessible at all times),
- **reliable** (no errors in stored content), and
- **consistent** (everyone sees the same thing, all the time).

Consistency is somewhat difficult, but service providers still try to provide it.

Most Services Provide Eventual Consistency

- Operations are serialized at a server.
- **Eventually every one perceives the same order** of operations.

- But not necessarily immediately.

Push Model Actively Forwards Updates to Users

Distributed file systems, such as Box, Dropbox, and Google Drive, support several consistency models.

Files

- in folders **synchronized** with **cloud** folders,
- **and** files **open in editors**,
- **are eventually consistent**.

For these files, **update operations** are **pushed to your system**.

Pull Model Waits for Users to Request Updates

However, most **distributed file systems**
- **also perform versioning**, and
- typically use that model to support disconnected operation,
- **requiring explicit uploads and downloads**.

In such cases, **update operations are**
- **pulled** by your system **from a server**,
- implicitly by a tool if you open a file to view or edit it.

Social Network Updates Generally Pulled from Server

Social networks use primarily a pull model.

Since users typically view
- only the most recent activity in the social graph,
- pushing all updates is generally a waste of bandwidth.

Only a **handful of changes** are **pushed**:
- those needed **to support active notifications**.
- (Notifications that show up when you open an app can also be pulled.)

Availability and Reliability are More Important

These same companies place more **emphasis on availability and reliability**.

We **talked about** these ideas **in social networks**.

File services are similar: your posts, photos, and videos are just a bunch of files.

Collaborative editing tools do require good definitions of operations and somewhat stronger (or at least faster) consistency guarantees to avoid irritating human users.

Availability: Your Data Located Near You

What about availability?

Here, too, the **techniques are** fairly **similar**.

Imagine working on a text document.

The **primary copy** of that document is stored **at a datacenter close to you**.

Within the datacenter, the **exact position** of your file is **selected using** an approach similar to TAO's **shard model**.

For the same reason: load balancing.

Additional Copies of Your Data Kept Elsewhere

But **that's not the only copy** of your file!

If the **disk** with your file **fails**, **or**

- (less likely) the optical **fibers** to the datacenter are all **cut**, **or**
- (even less likely) a **meteorite strikes** the datacenter,
- other datacenters have a copy, too!

It just **takes a little longer**
(more msec, not seconds)
to get your data.

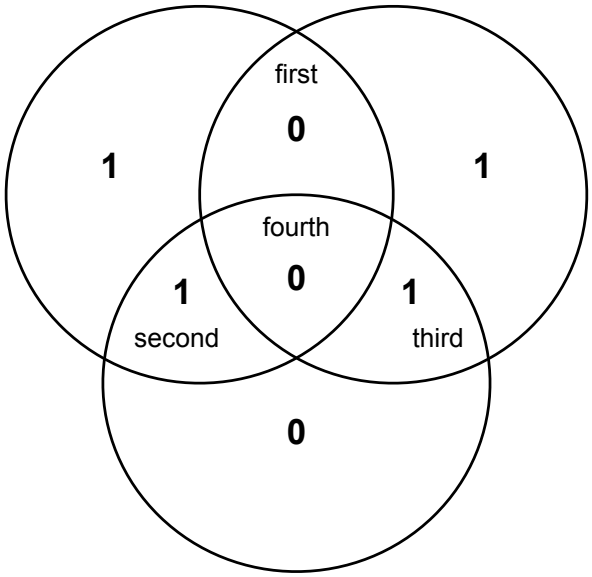
Reliability Supported Using Codes

Reliability is also an issue.

All **digital storage** systems
break down over time.

To protect your file data,
- **companies use coding** techniques
- (remember the Hamming code?).

Codes are used
- **to protect against changes on disk**, and
- on-disk data are periodically "scrubbed"
- to correct any errors that have popped up.



Reliability Supported Using Codes

Similarly, **codes are used**
- **to store** data **across multiple drives**,
- **increasing bandwidth and**
- **protecting against failure** of any drive.

In this case, the 5-out-of-7 variant
- of the Hamming code would work perfectly:*
- if you store across 7 disks,
  - so long as no more than 2 fail,
  - you can recover all of the data!

*That particular code is not common in practice for disk systems, but the idea is the same.

Sharing Supported Using Access Control Lists

One more topic: sharing.

Sharing of documents (posts, and so forth)
- typically **managed with**
- an Access Control List (**ACL**),
- a list of rules.

The **rules** are **checked one at a time** until a match is found.

ACL

1. ALLOW MySecretAccount
2. DENY unless <Friend>
3. DENY BirthdayFriend
4. ALLOW

With the ACL shown, my secret account as well as all of my friends, except the one friend I'm trying to surprise, can see the document.

Reminder: You are Guaranteed Nothing.

Here's the fine print from one popular set of services. It is essentially identical to the wording introduced by the Berkeley Software Distribution's ***FREE*** version of Unix, TCP, and so forth.

These days, however, most companies use the same rules even if you are a paying customer. Even banks. It's scary.

TO THE EXTENT ALLOWED BY APPLICABLE LAW, WE PROVIDE OUR SERVICES "AS IS" WITHOUT ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT. FOR EXAMPLE, WE DON'T MAKE ANY WARRANTIES ABOUT THE CONTENT OR FEATURES OF THE SERVICES, INCLUDING THEIR ACCURACY, RELIABILITY, AVAILABILITY, OR ABILITY TO MEET YOUR NEEDS.

Law. The only way you can ever expect to have any guarantee from modern software.

Terminology You Should Know from These Slides

- cloud storage
- available
- reliable
- consistent
- operations (and examples)
- undo/invert, log, and versions
- context (for an operation)
- idempotent
- serialization (by a server)
- causality violation (the Jan and Pat example)
- eventual consistency
- push/pull for updates
- access control lists (ACLs, for sharing)

Concepts You Should Know from These Slides

- properties assumed by users of cloud storage
- not all operations can be inverted easily
- why merging versions can be hard
- why the exact form of an operation matters
- providing strong consistency by using a single location
- effect of delay on exposing inconsistency
- speed benefit of providing only weak consistency
- how availability and reliability are typically supported
- basic use of an access control list (ACL)