Reading 4
and Upgrading Apps

CSE 331: Software Design and Implementation, Autumn 2026

Your job is to deliver code that you know works.

TODO

- **Tomorrow**: section on generics, with a project proposal workshop
  - please bring a draft of your proposal to section!
- Project proposal: due Tuesday, November 24, 11:59pm
- **Wednesday to Friday**: no class (Thanksgiving)
- HW4: due Friday, December 4, 11:59pm

Reading 4 discussion

- Form groups of 3 or 4 with the people next to you
- **Naur 1985**: the program is a theory in its authors' heads
- **Winters 2020**: engineering is programming integrated over time
- **Carlini and Regehr 2026**: a C compiler written by AI, then fuzzed

Share (5 minutes)

Sooner or later you will change a program you did not write,
whether a person wrote it or a machine did.
What do you need besides the code, and can you get it?

- Each of you, in turn:
  - Which article you read, and its answer
  - The question you wrote for the discussion

The argument, in this course's words

- **Naur**: you need the theory, and no document can hand it to you
- **Winters**: you need the ability to change the code safely
- **Carlini and Regehr**: you need tests that are "nearly perfect"
- **This course**: specifications, tests, and arguments
  - Are they the theory?

Argue (6 minutes): who is right?

- **Naur**: the team is gone, so the program is dead, even while it still runs
- **Winters**: "Stagnation is an option, but often not a wise one"
- **Regehr**: a model fixed 34 bugs; "there's no chance" it is correct
- If your group agrees, argue the other side, or take a case:
  - Tests pass and nobody understands it: ship it? Who reads GCC?
  - The team is gone, two weeks to ship: rewrite, or read and patch?
  - A class project might become a real app: build for that now?

Report (a few minutes)

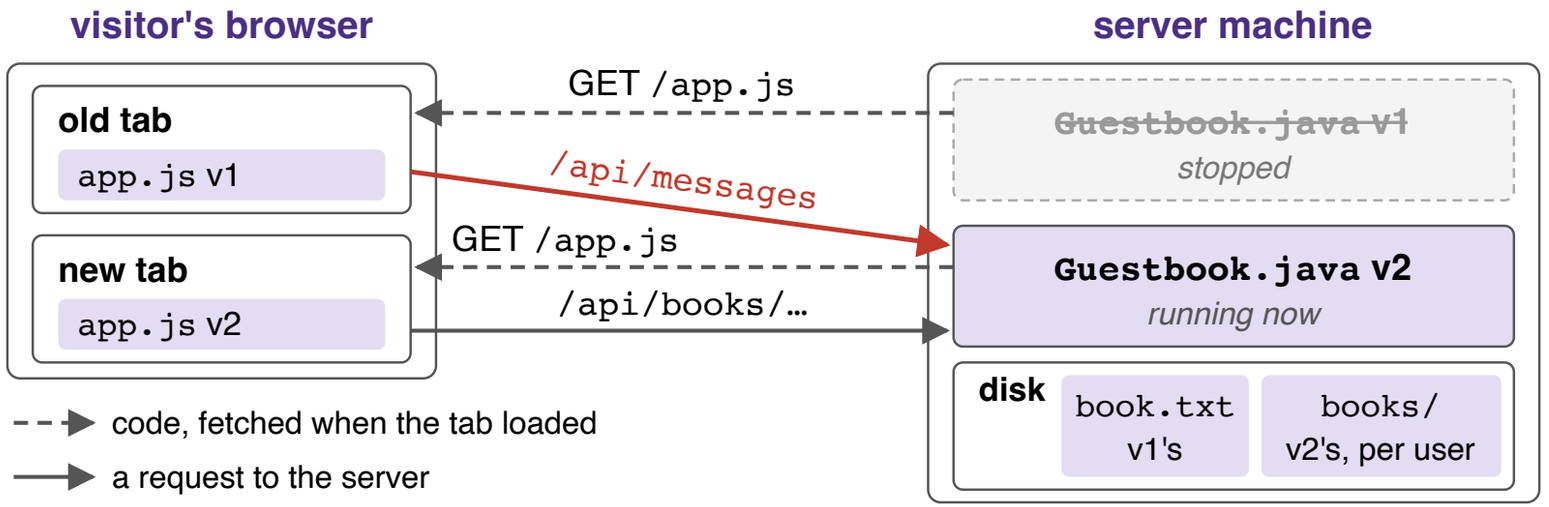
- A couple of groups: what did you decide, and why?

Half sheet, question 1 (3 minutes)

- The names of the others in your group
- What changed in what you think you need, besides the code
- One new point, and the article or person it came from

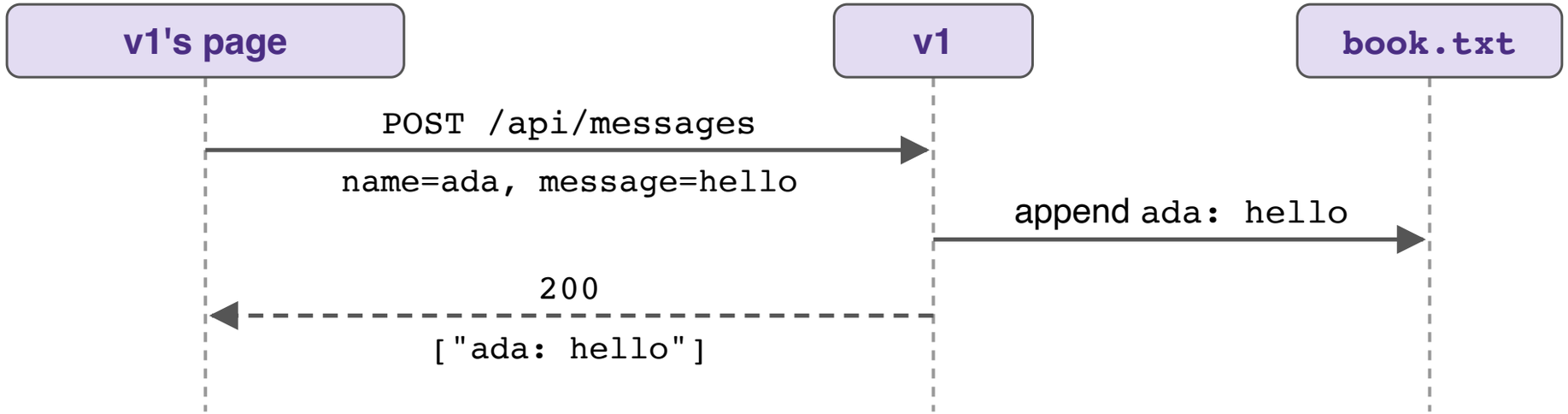
Upgrading Apps

Two versions of two programs



- **Upgrade**: stop v1, start v2, while visitors keep using the site
- The promise: a posted message is saved forever, through upgrades too

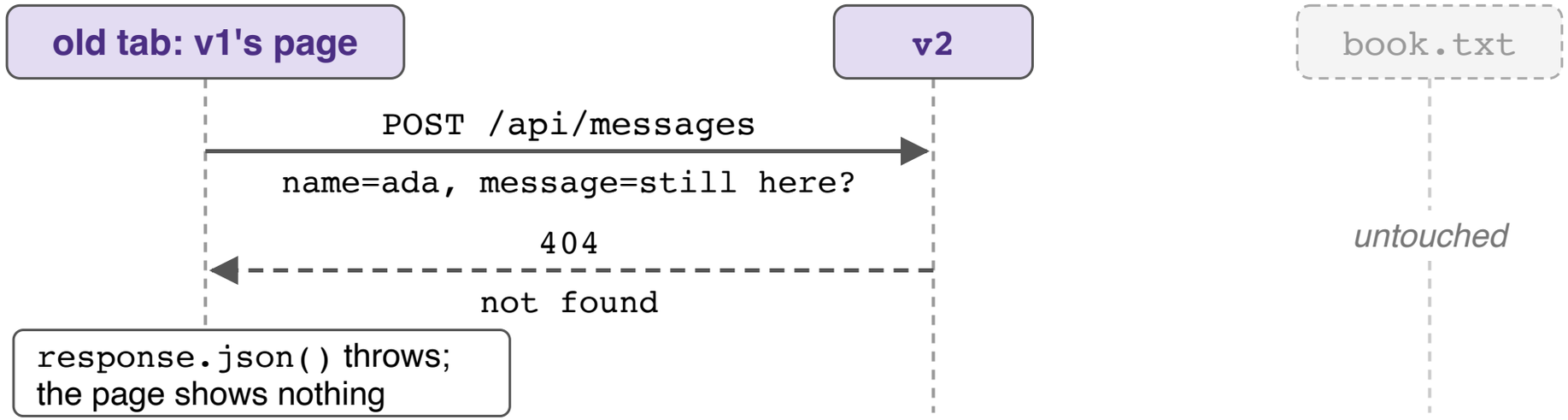
v1 writes book.txt



- Ada posts `hello`, and bob posts `hi ada`, from v1's page
- `book.txt` now holds two lines: `ada: hello` and `bob: hi ada`
- Then we upgrade: stop v1, and start v2 in the same directory

The old tab posts to v2

- v2 answers only v2's requests: it drops v1's
- Ada's new tab runs v2's page, and her book works
- Her old tab, opened before the upgrade, still runs v1's page



Old code and new code run at the same time

- **v1 with v1's page works.**
- **v2 with v2's page works.**
- When we upgrade, v1 and v2 will run at the same time
- **broken!**

The old page is a client

- The old page relies on v1's manual, as a caller on a specification

```
17  * Four kinds of request reach the program:
18  *   GET /                the page, index.html, as it is on disk
19  *   GET /app.js          the page's program, app.js, as it is on disk
20  *   GET /api/messages    the messages posted so far, as JSON
21  *   POST /api/messages   adds the message the fields send, then the
22  *                        messages posted so far, as JSON
```

- The old page sends only these requests
- **Which new manuals keep every promise the old page relies on?**

A stronger specification breaks no client

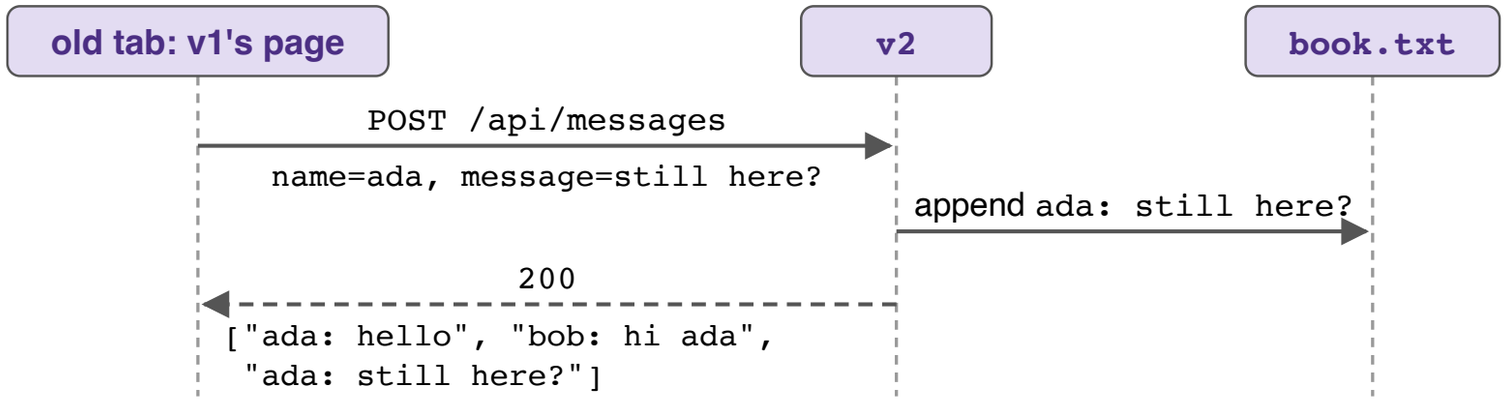
Rule: a safe upgrade

An upgrade to a stronger specification is guaranteed not to break any clients.

- The old client constrains what the new implementation can do

Keeping v1's promises

- **v2** drops v1's requests, so it is not stronger: the rule does not apply
- What if v2 answered its own requests and kept v1's, word for word?

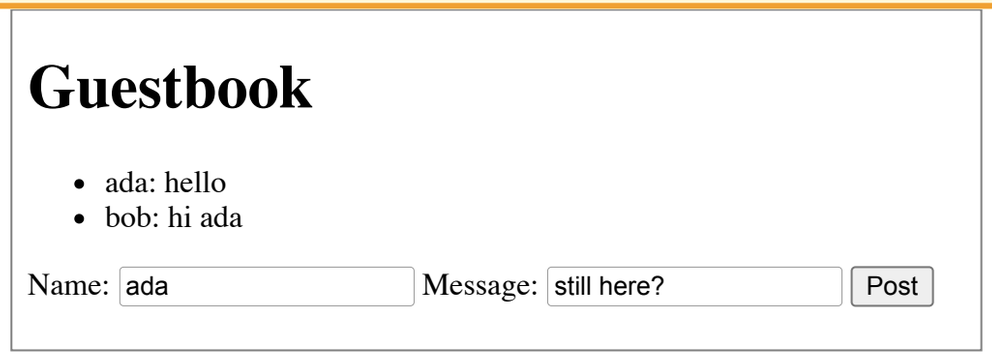


- On the old page's two requests, v2 now promises what v1 promised
  - stronger or the same there: by the rule, safe for the old page
  - as long as it relies only on them

What if v2 sent the old book newest first?

- One line changes in v2:

```
89   send(exchange, 200, "application/json",
90        json(shared.messages().reversed()).getBytes(UTF_8));
```



- Ada's tab still runs v1's page. She is about to post `still here?`
- She expects it at the bottom of the list, under bob's `hi ada`

Half sheet, question 2

- v1's manual promises "the messages posted so far, as JSON"
- Call this v2 **Reversed**: it keeps that line, but sends them newest first
- The old page lists them in the order they arrive
  - loaded from v1, it lists `ada: hello`, then `bob: hi ada`
  - then v1 stops, and Reversed starts in its place
- **Is Reversed safe for the old page?**
  What happens after ada posts "still here"?

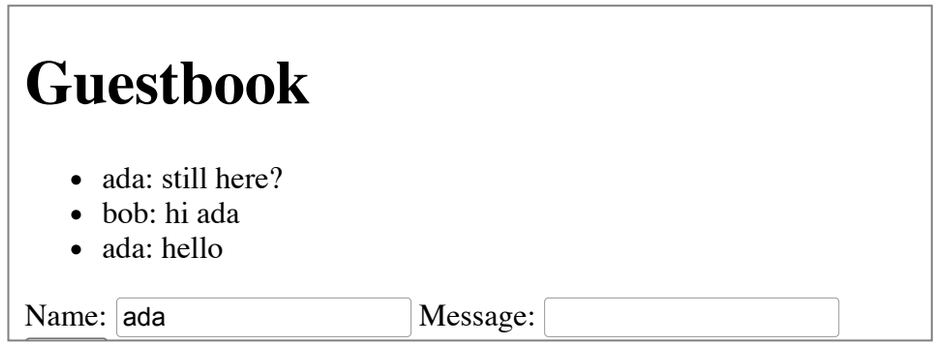
The old page, newest first

- On the old page's requests, v1's promises: by the rule, safe

v1: ada is about to post



the new v2, same tab: ada clicks Post



- Her message lands on top, and bob's reply sits above what it answers
- No line of v1's manual promised that order

Hyrum's Law

> *With a sufficient number of users of an API, it does not matter what you promise in the contract: all observable behaviors of your system will be depended on by somebody.*

- So the rule protects only the clients that rely only on the specification
- Write specifications that promise what clients rely on
  - v1's manual should have said "oldest first"

Winters, "What Is Software Engineering?", Software Engineering at Google (2020)

Is v2 stronger than v1?

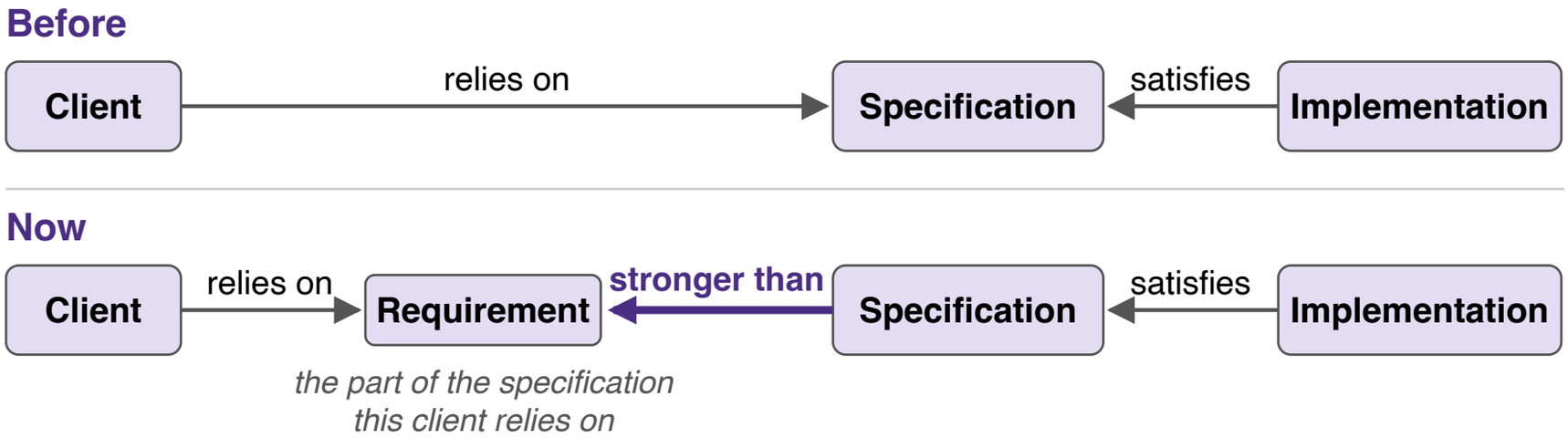
- No: at `GET /`, v1 sent the page, and v2 sends a redirect
- So the rule, read strictly, does not apply
- Safe for the old page all the same: it never sends `GET /` again

Definition: requirement

A client's **requirement** is the part of a specification that the client relies on.

An upgrade is safe for a client that relies only on the old specification if the new version's specification is stronger than that client's requirement.

Client, requirement, specification

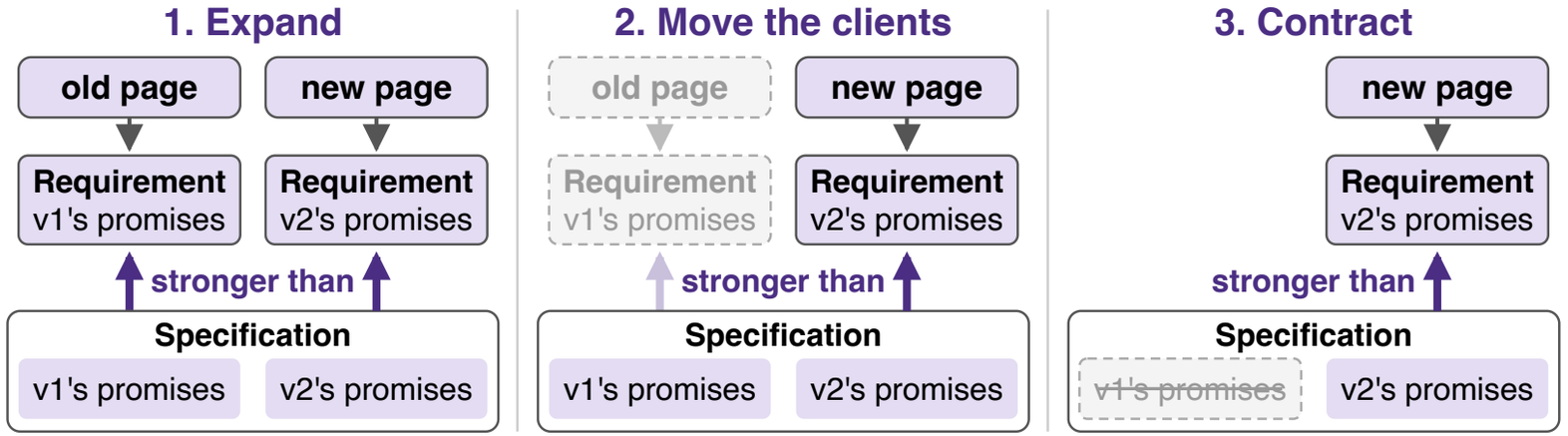


- Before: a client relies on the whole specification
- Now: the specification must be stronger than the client's requirement

What if we want to break a promise?

- Some day v2 should stop answering v1's requests
  - and stop keeping `book.txt` in v1's form
- Removing a promise is not stronger: the rule cannot help
- Some old tab may still rely on it
- **How do we remove a promise without breaking a client?**

Expand, move the clients, contract



1. **Expand**: add the new promises, and keep the old ones

2. **Move the clients**: announce a date, **deprecating** the old ones

3. **Contract**: remove the old promises, a weakening, safe only now

Sato, "ParallelChange", martinfowler.com (2014): expand, migrate, and contract

Stored data is a client from the past

- v2 never reads `book.txt`, so no request brings back "ada: hello"
- That breaks the promise, though `book.txt` is still on disk
- The file sends no requests, so the rule compares two readers:
  - **a precondition no stronger**: accept every file v1 could have left
  - **a postcondition no weaker**: give back the messages, as v1 did
- v2, keeping v1's requests, meets both, with one more line in `main`:

```
59   public static void main(String[] args) throws IOException {
60       books = new Books(Path.of("books"));
61       shared = new SharedBook(Path.of("book.txt"));
```

Migration

Definition: migration

Migration is converting stored data from the old version's form to the new version's.

- v2 signs each message with a login the visitor cannot choose
- A line of `book.txt` is signed with a name typed into a field
- So `ada: hello` cannot simply move into ada@example.com's book

Whose book gets v1's messages?

- No program can turn "ada" into a login, since v1 never learned one
- So it is a decision
- Each choice changes a specification or breaks a promise:
  - **one shared book that nobody posts to**: v2's specification changes
  - **ada@example.com's book**: anyone could have typed ada
    - so it breaks v2's promise
  - **no book**: leave `book.txt` unread, and break the promise
- The people who run the guestbook make that choice
- v2's specification has to say which they made

Who reads the server's words?

- v2's page puts `v2` after `/api/` in every request
- So the last v2 answers any other path under `/api/` with 410, *Gone*:

```
80   if (path.startsWith("/api/") && !path.startsWith("/api/v2/")) {
81       send(exchange, 410, "text/plain",
82           "this page is out of date; reload it".getBytes(UTF_8));
```

- On v1's page, nobody: its `post` throws, as on v2's 404
- So the code that shows the server's error, like the version,
  - has to be in a page from its first release

Two practices

1. **Upgrade to a stronger specification**
    - it breaks no client that relies only on the old one
    - remove a promise only once no client relies on it
2. **Version every request and file, from the first release**
    - and show the server's error on every page
3. **Semantic versioning**: MAJOR not stronger, MINOR stronger

Preston-Werner, Semantic Versioning 2.0.0 (2013)

Coming up

- **Tuesday**: section on generics, with a project proposal workshop
  - project proposal due, 11:59pm
- **Wednesday to Friday**: no class (Thanksgiving)
- **Monday, November 30**: deploying, operating, and maintaining apps
- Hand in your half sheet at the door