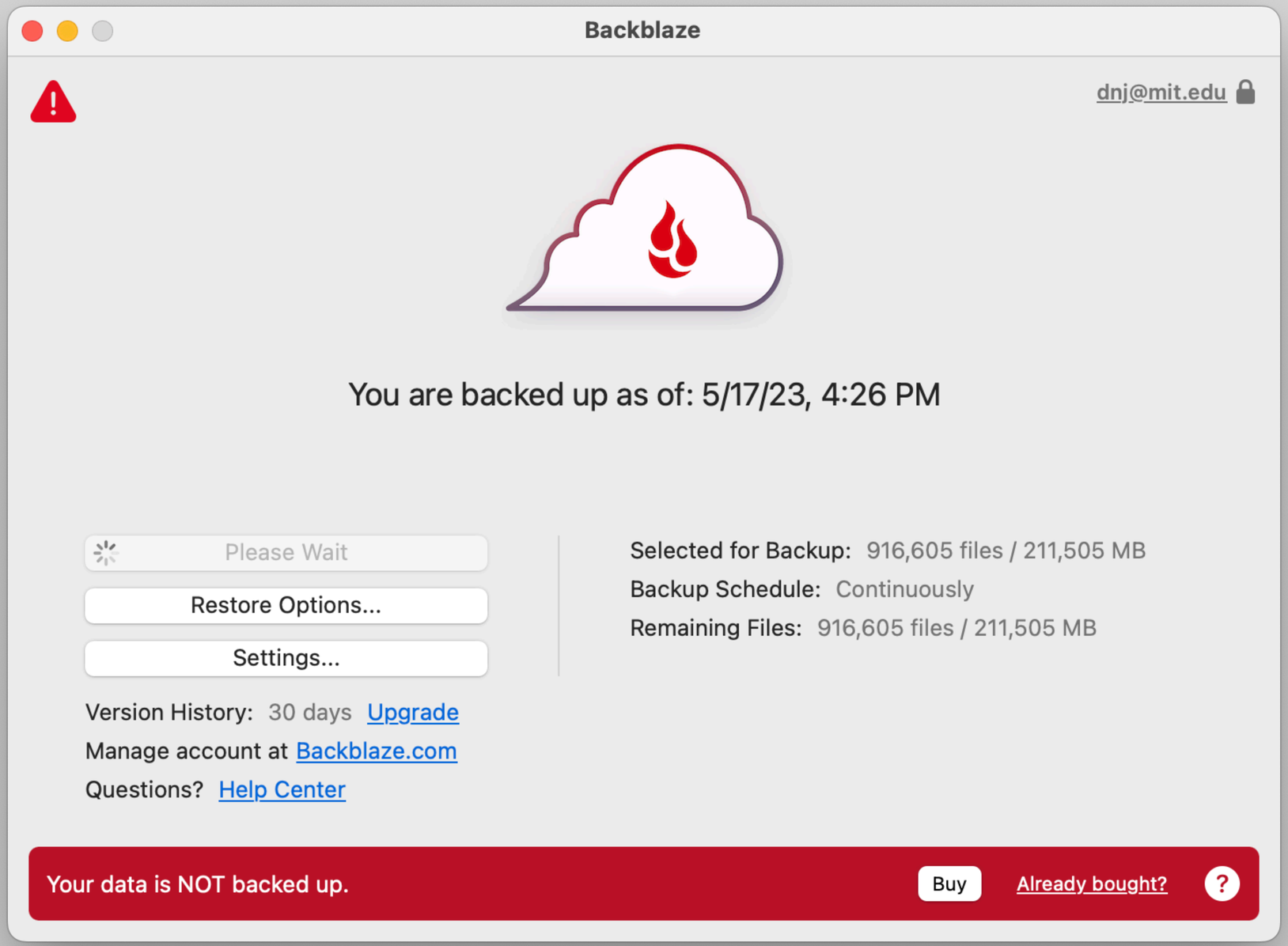
a new way to structure software

Daniel Jackson · UC Berkeley · January 28, 2025

part 1:
diagnosing UX

a UX puzzle: backblaze

backing up on Backblaze



Backblaze Backup

Search

dnj@mit.edu

was modification at 10pm saved?

You are backed up as of: 6/6/22, 10:10 PM

Currently backing up newer files

is backup running or not?

Pause Backup

Restore Options...

Selected for Backup: 509,021 files / 2,379,995 MB

Backup Schedule: Continuously

Remaining Files: 0 files / 0 KB

huh?

Transferring: photo.0259-22.RAF

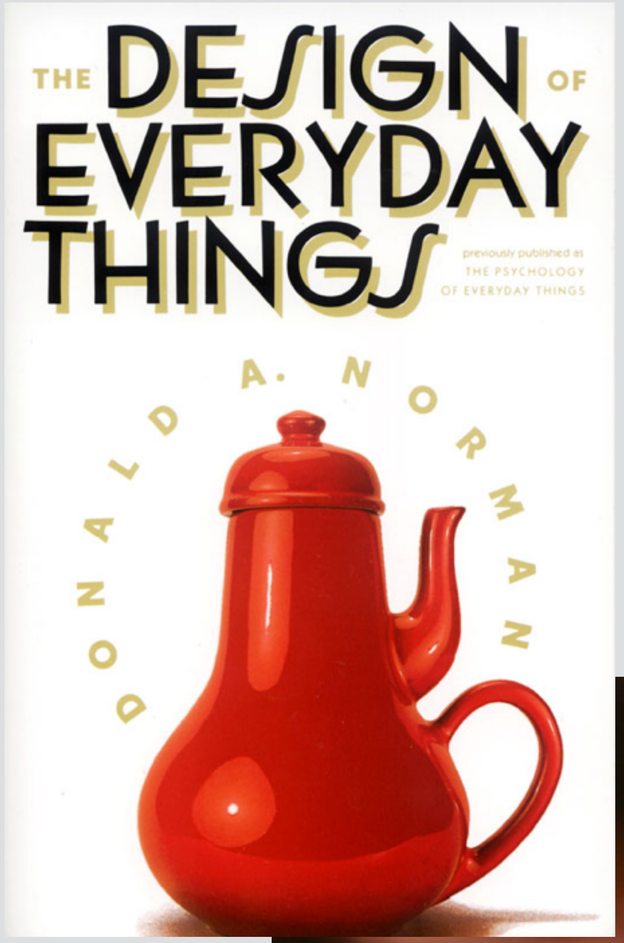
Settings...

What is being backed up?

How long will my first backup take?

View files and manage account at: Backblaze.com

conceptual models to the rescue



1988

Donald Norman

Although DOET covers numerous topics, three have come to stand out as critical:

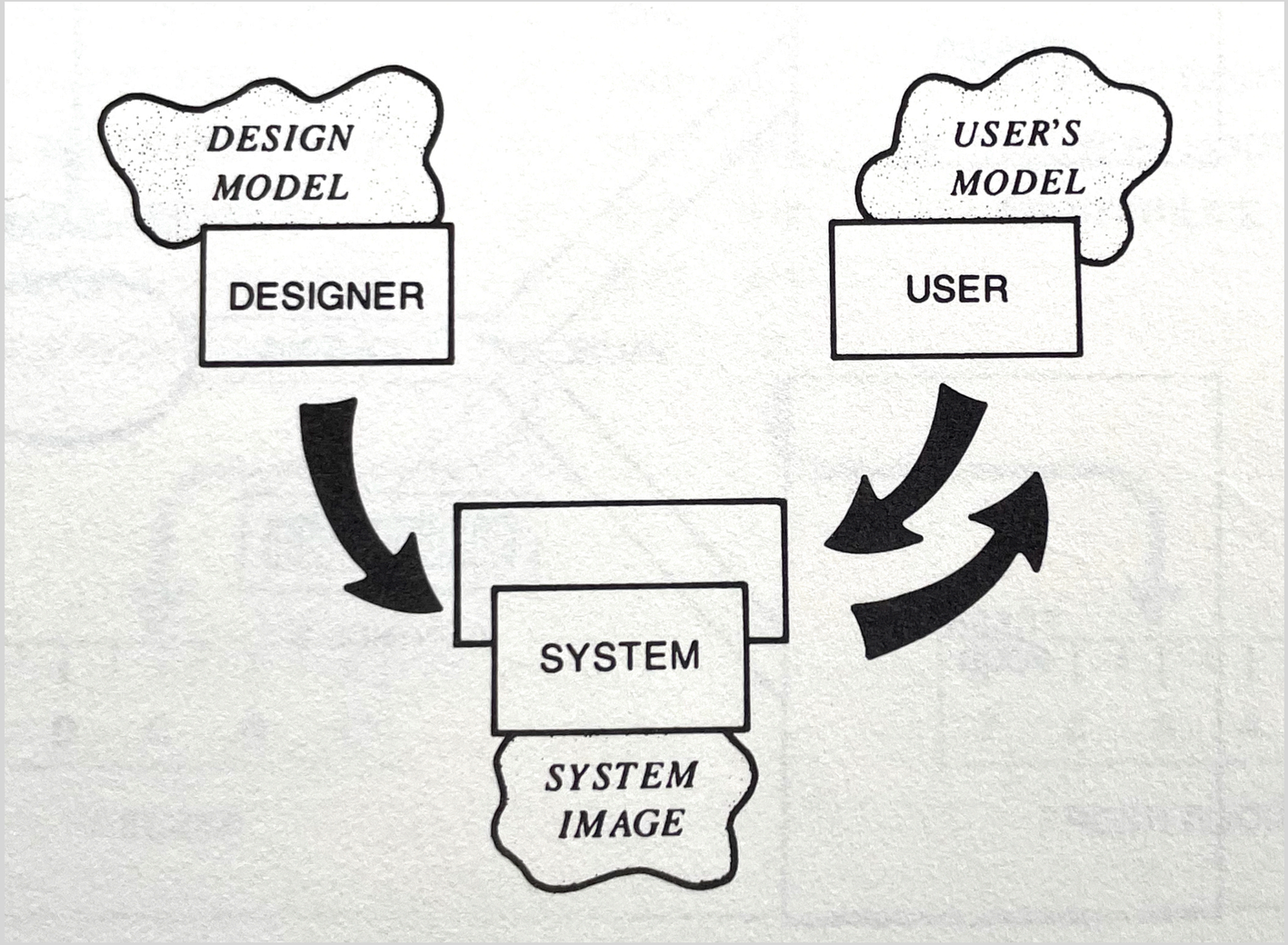
1. It's not your fault...
2. Design principles... **conceptual models**, feedback, constraints, affordances
3. The power of observation

preface to 2002 edition

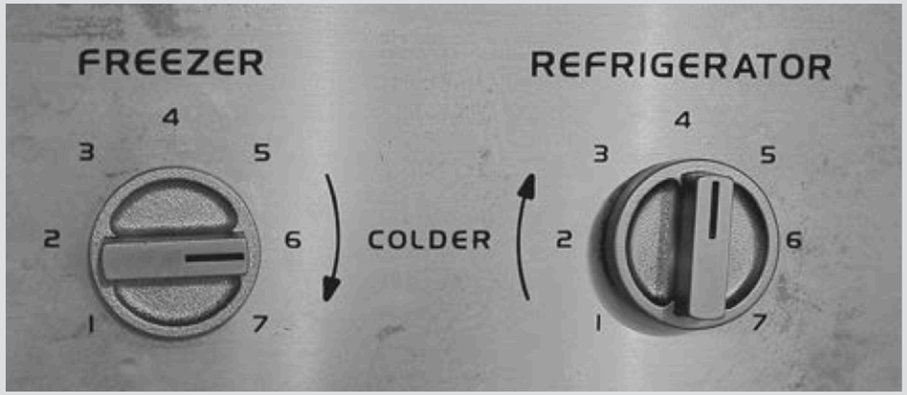
When the designers fail to provide a conceptual model, we will be forced to make up our own, and the ones we make up are apt to be wrong.
Conceptual models are critical to good design.

preface to 2013 edition

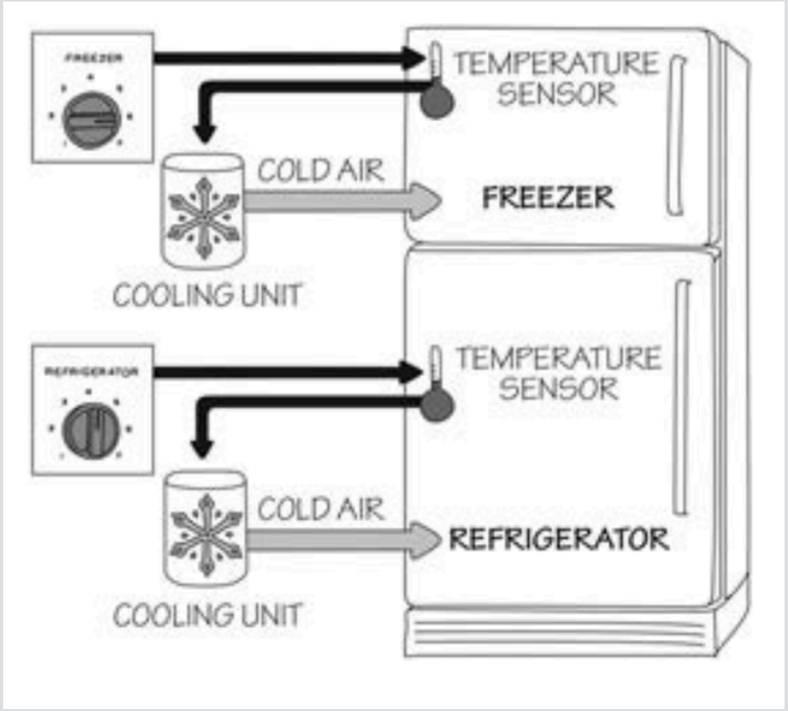
the “system image”



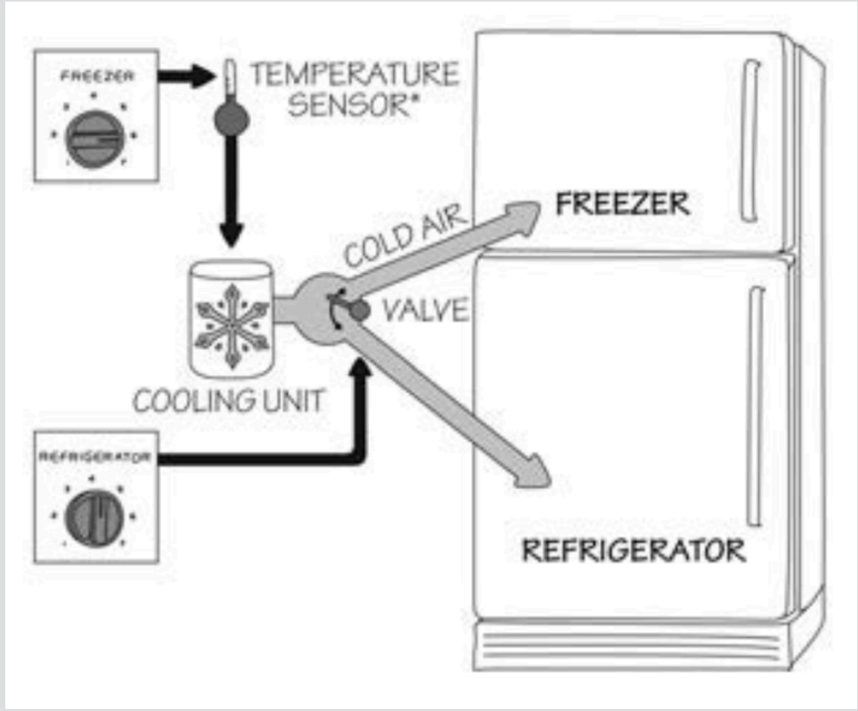
from *The Design of Everyday Things*



typical controls on American fridge



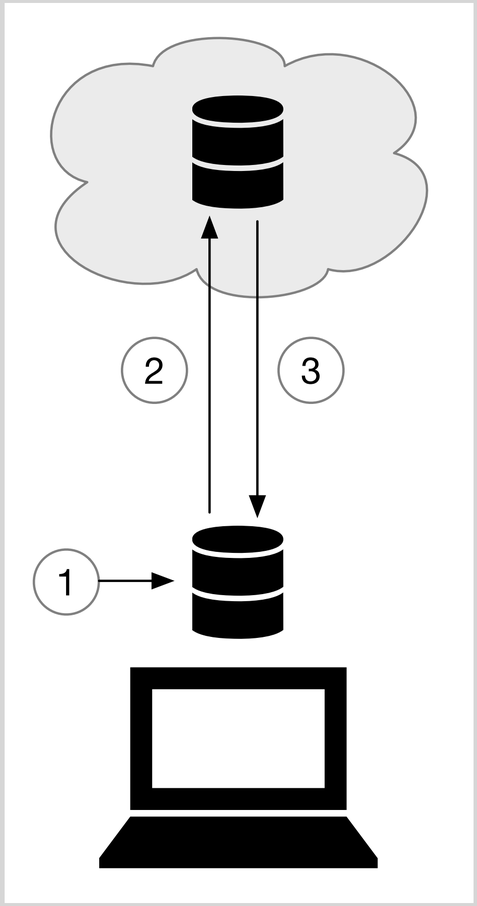
conceptual model (imagined)



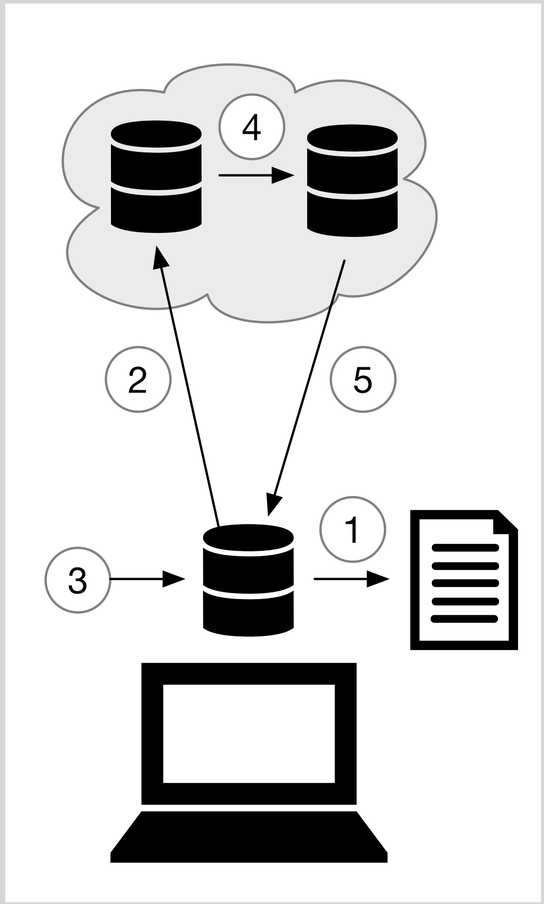
conceptual model (actual)

from Don Norman, Design of Everyday Things (2013)

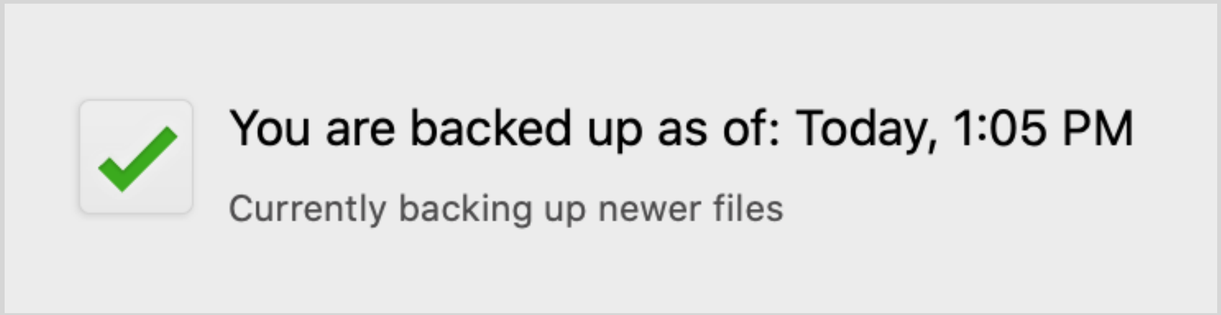
imagining backblaze's conceptual model



"**continuous backup**"
what I imagined



"**continuous backup**"
what actually happens



a harder case: powerpoint's sections

powerpoint's section concept

how to group slides into a section

1. select first slide to be in section & do add section
   (this will make a section from the selected slide to the end)
2. select slide after last slide to be in section & do add section
   (this will break the slides into two sections)

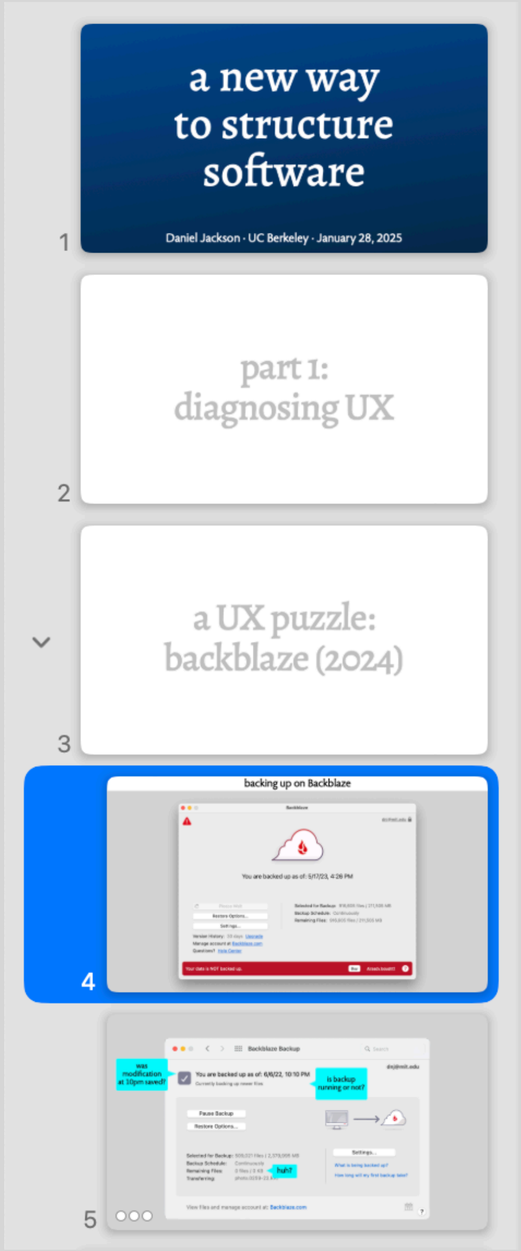
some anomalies

- when you add your first section, a default section is created, so you get two sections (unless you selected the first slide)
- you can't delete the default section (unless it's the only one)
- if you select multiple slides, add section works as if you'd selected the first (unless not contiguous, then not allowed)

missing functionality: you can't

- nest sections
- hide a section (except in slide sorter)
- move a section more than one step (except in slide sorter)

keynote's tree outline concept



how to group slides under a header

1. select all except a header slide, and drag to right

some anomalies

- none

missing functionality: you can't

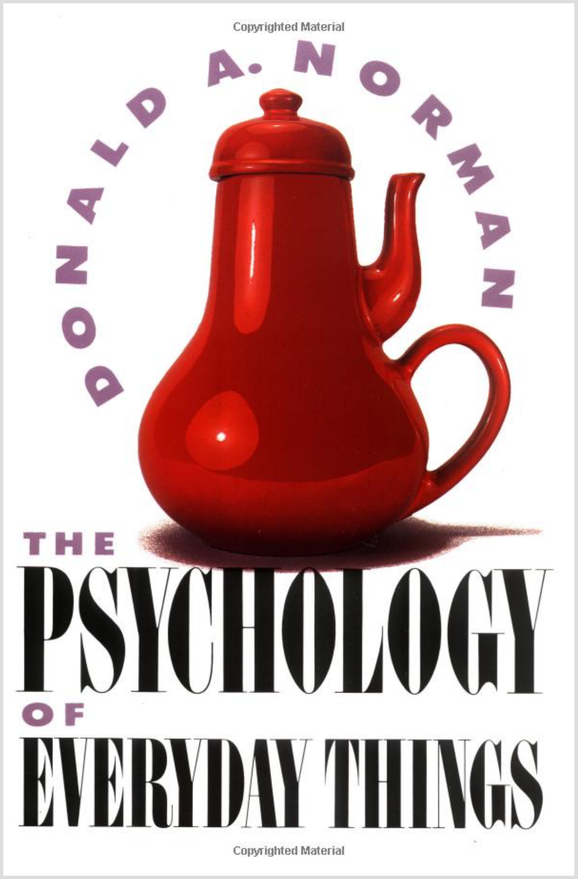
- group slides without a header
  (but you can mark header as skipped)

revisiting conceptual models

user's model

classic usability research

designer's model



user's model

go this way instead

designer's model

but what if the designed model is just wrong?

what exactly is wrong?

Perhaps the designers thought the correct model was too complex, that the model they were giving was easier to understand. But with the wrong conceptual model, it is impossible to set the controls. And even though I am convinced I now know the correct model, I still cannot accurately adjust the temperatures because of refrigerator design makes it impossible for me to discover which control is for the thermostat, which controls for the relative proportion of cold air, and in which compartment the thermostat is located. The lack of immediate feedback for the actions does not help: with the delay of 24 hours, who can remember what was tried?

Design of Everyday Things, p. 17

what's missing

the conceptual model itself
unless it's explicit, how can we know if we mapped it right?

design criteria for models
what makes a good conceptual model?

model structure
can we break the model into smaller parts? reusable concepts?

part 2: introducing concepts

what might be?

where a concept language might help

focusing on essence
design of function
distinct from UI design
distinct from code design

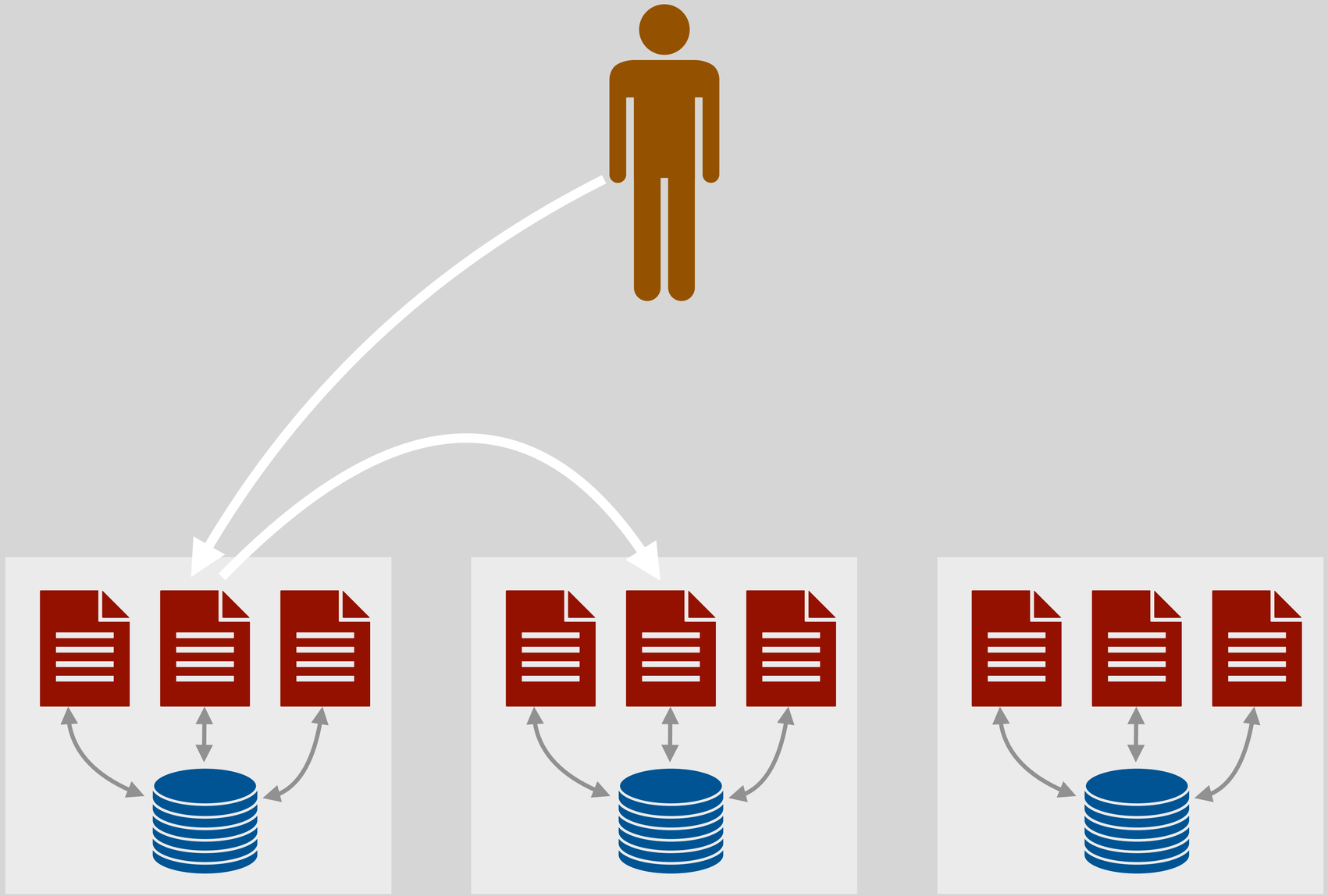
articulating assets
key enablers of services
product differentiators
guiding strategic choices

aligning roles
user & designer
UX, PM & devs
tech, marketing & IP

preserving knowledge
reuse design ideas
surface hidden knowledge
for training novices
for prompting LLMs

defining concepts

viewing a system in terms of concepts



a file store concept

concept FileStore [Name, Content]

purpose store files persistently

principle after creating and updating a file, you can get the content

state
a set of files
for each file
  name, contents

actions
create (n: Name, c: Content)
update (n: Name, c: Content)
delete (n: Name)
get (n: Name): Content

what's a name?
could be a pathname
allows hierarchy and sidesteps complexity of folders
no possibility of two parents (as in Unix)
but also no empty folders!

changing names?
can a file's name be changed with identity remaining?
then could say "this file's name was changed" (cf. Git)

a backup concept

concept Backup [Name, Content]

purpose retrieve old version of files

principle after a file's contents are saved, they can be retrieved later by date

state
a set of files with versions
for each file
  name, contents, date

actions
save (n: Name, c: Content)
restore (n: Name, d: Date): Content

are files mutable?
no

can empty folders be stored?
no

can files be deleted?
no (but Backblaze isn't like this)

a workset concept

concept Workset [Item]

purpose process items in batches

principle after items are added, and processing is started, the items are processed

state

current set of items being worked on
next set of items to work on

actions

start ()
  requires current == {}
  current = next
  next = {}

add (i: Item)
  next = next + i

process (i: Item)
  requires i in current
  current = current - i

when do the actions happen?

concept FileStore [Name, Content]

purpose store files persistently

principle after creating and updating a file, you can get the content

state
a set of files
for each file
  name, contents

actions
create (n: Name, c: Content)
update (n: Name, c: Content)
delete (n: Name)
get (n: Name): Content

concept Backup [Name, Content]

purpose retrieve old version of files

principle after a file's contents are saved, they can be retrieved later by date

state
a set of files with versions
for each file
  name, contents, date

actions
save (n: Name, c: Content)
restore (n: Name, d: Date): Content

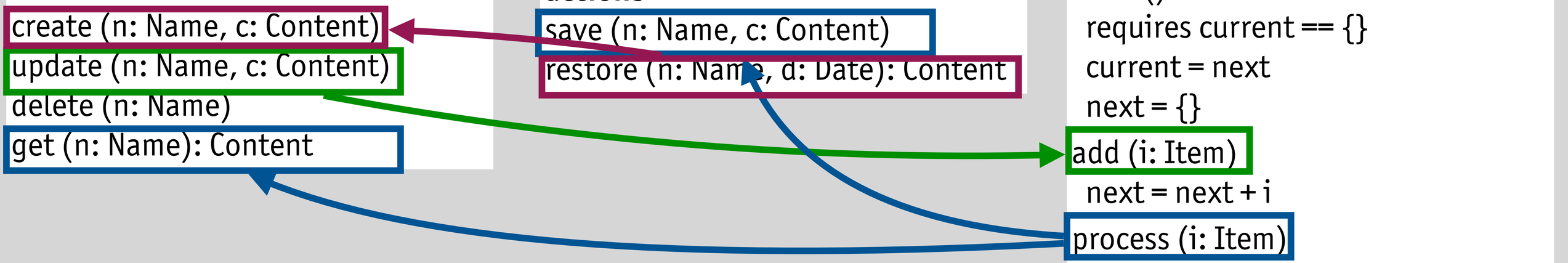
concept Workset [Item]

purpose process items in batches

principle after items are added, and processing is started, the items are processed

state
current set of items being worked on
next set of items to work on

actions
start ()
  requires current == {}
  current = next
  next = {}
add (i: Item)
  next = next + i
process (i: Item)
  requires i in current
  current = current - i



keynote and powerpoint's grouping concepts

concept TreeOutline [Item]

purpose organize & view sequence of items in flexible way

principle when items are made children of another item, they can be hidden, or moved together

state
top: **seq** Item
children: Item -> **seq** Item

actions
add_item (i, after: Item)
del_item (i: Item)
move_item (i, to: Item,
  pos: {before, after, child})

concept Section [Item]

purpose organize & view sequence of items in flexible way

principle when items are split into sections, a section can be moved up or down

state
top: **seq** Item + **seq** Section
items: Section -> **seq** Item
text: Section -> **one** Text

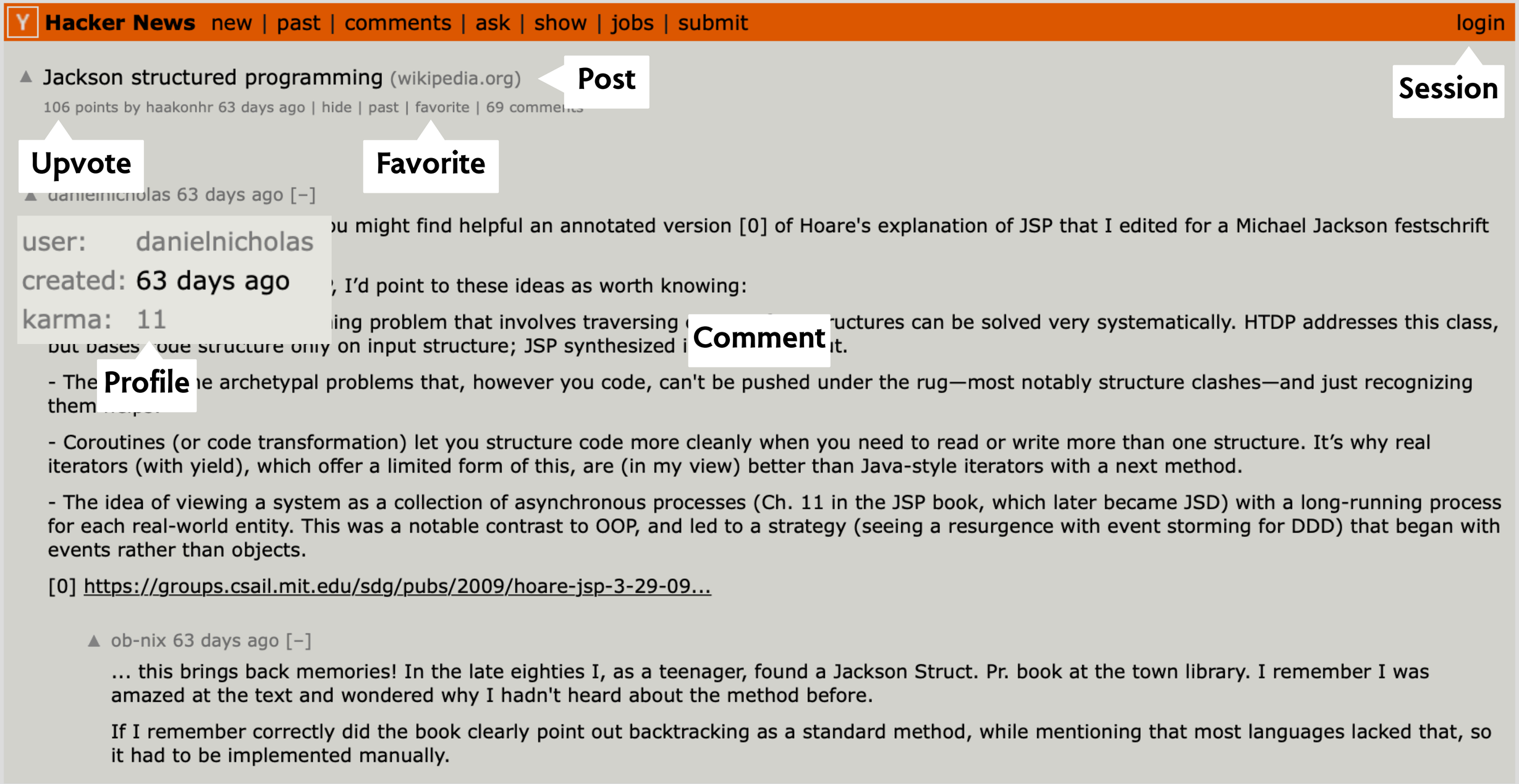
actions
add_item (i, after: Item)
del_item (i: Item)
add_section (before: Item): Section
move_item (i, to: Item + Section,
  pos: {before, after})

concepts:
modular, reusable
& user-centric
units of function

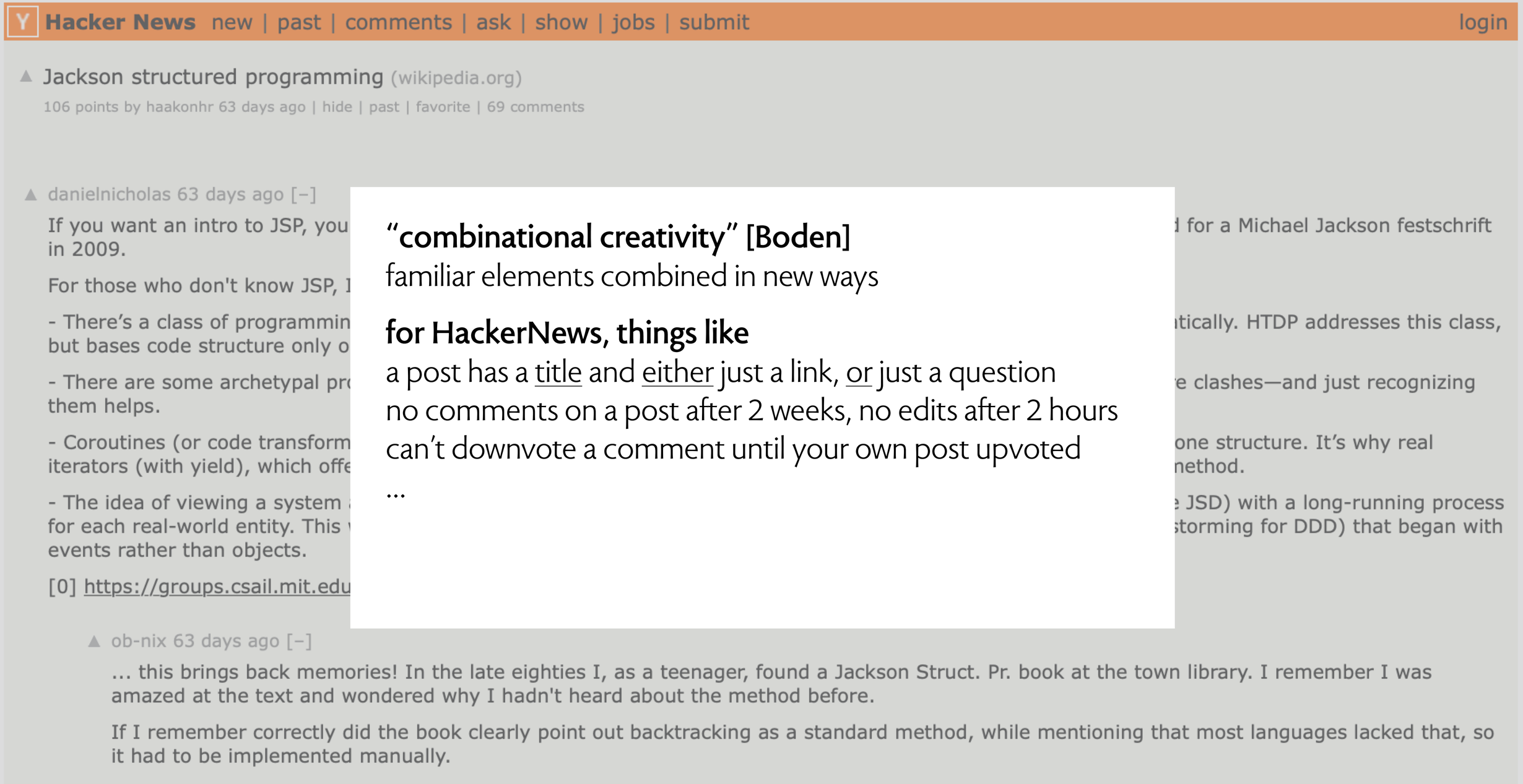
part 3: modularity

another example: hacker news

most apps are made from familiar functions



... with some creative variation



"combinational creativity" [Boden]

familiar elements combined in new ways

for HackerNews, things like

a post has a title and either just a link, or just a question

no comments on a post after 2 weeks, no edits after 2 hours

can't downvote a comment until your own post upvoted

...

how to add app-specific functionality?

concept Upvote

purpose rank items by popularity

actions
upvote (u: User, i: Item)
downvote (u: User, i: Item)
unvote (u: User, i: Item)

suppose I want this behavior:
you can't downvote an item
until you've received
an upvote on your own post

define a new concept!
a hint: not just used by Upvote

concept Karma

purpose privilege good users

state
karma: User -> one Int

actions
reward (u: User, r: Int)
permit (u: User, r: Int)

concept Post

purpose share content

state
author: Post -> one User
body: Post -> one Text

actions
create (u: User, t: Text): Post
delete (p: Post)
edit (p: Post, t: Text)
get_author (p: Post): User

compose concepts by action synchronization

when
Upvote.upvote (u, i)
Post.get_author (i) = u'
sync Karma.reward (u', 10)

concept Karma

actions
reward (u: User, r: Int)
permit (u: User, r: Int)

concept Upvote

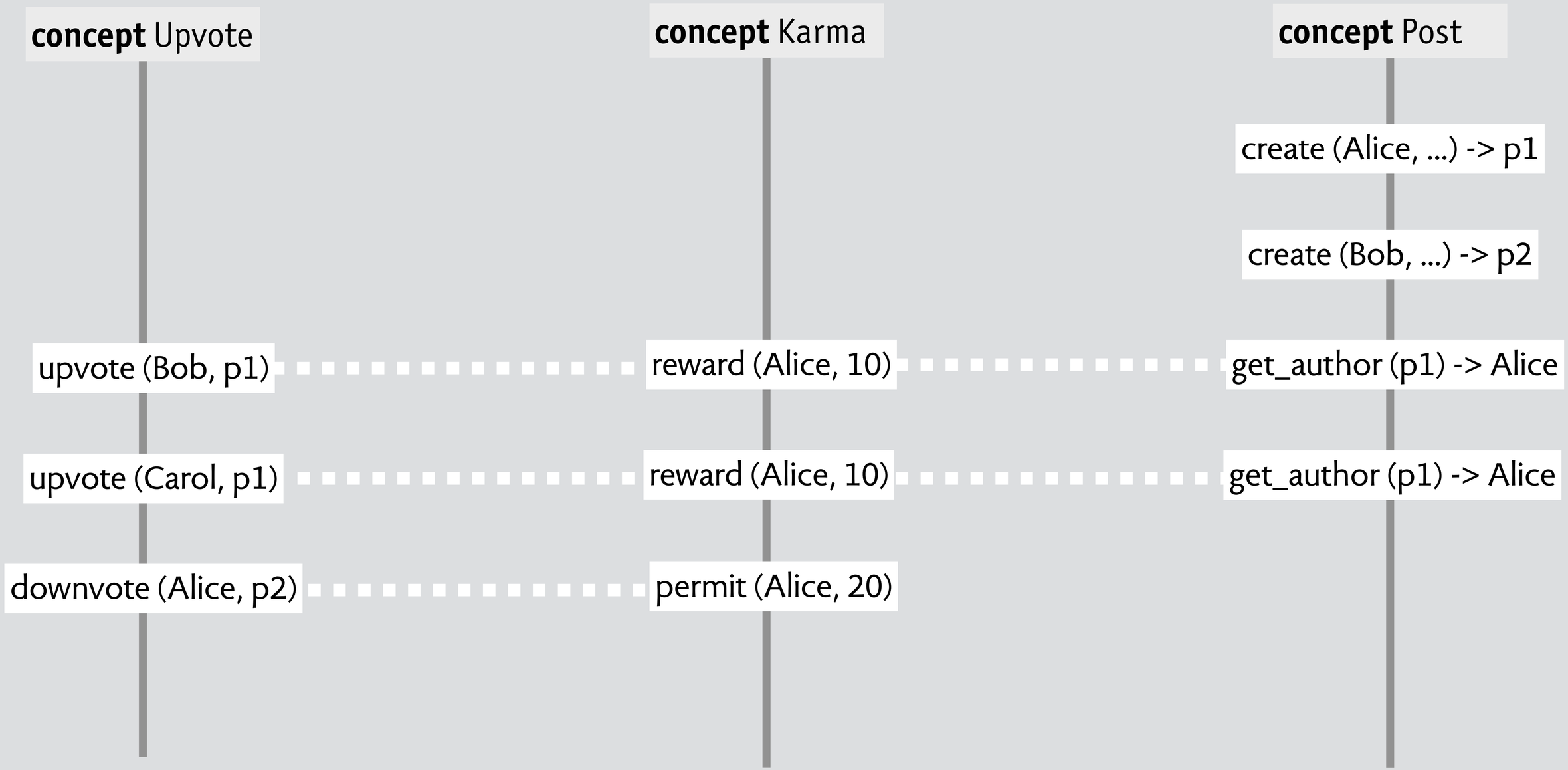
actions
upvote (u: User, i: Item)
downvote (u: User, i: Item)
unvote (u: User, i: Item)

when Upvote.downvote (u, i)
sync Karma.permit (u, 20)

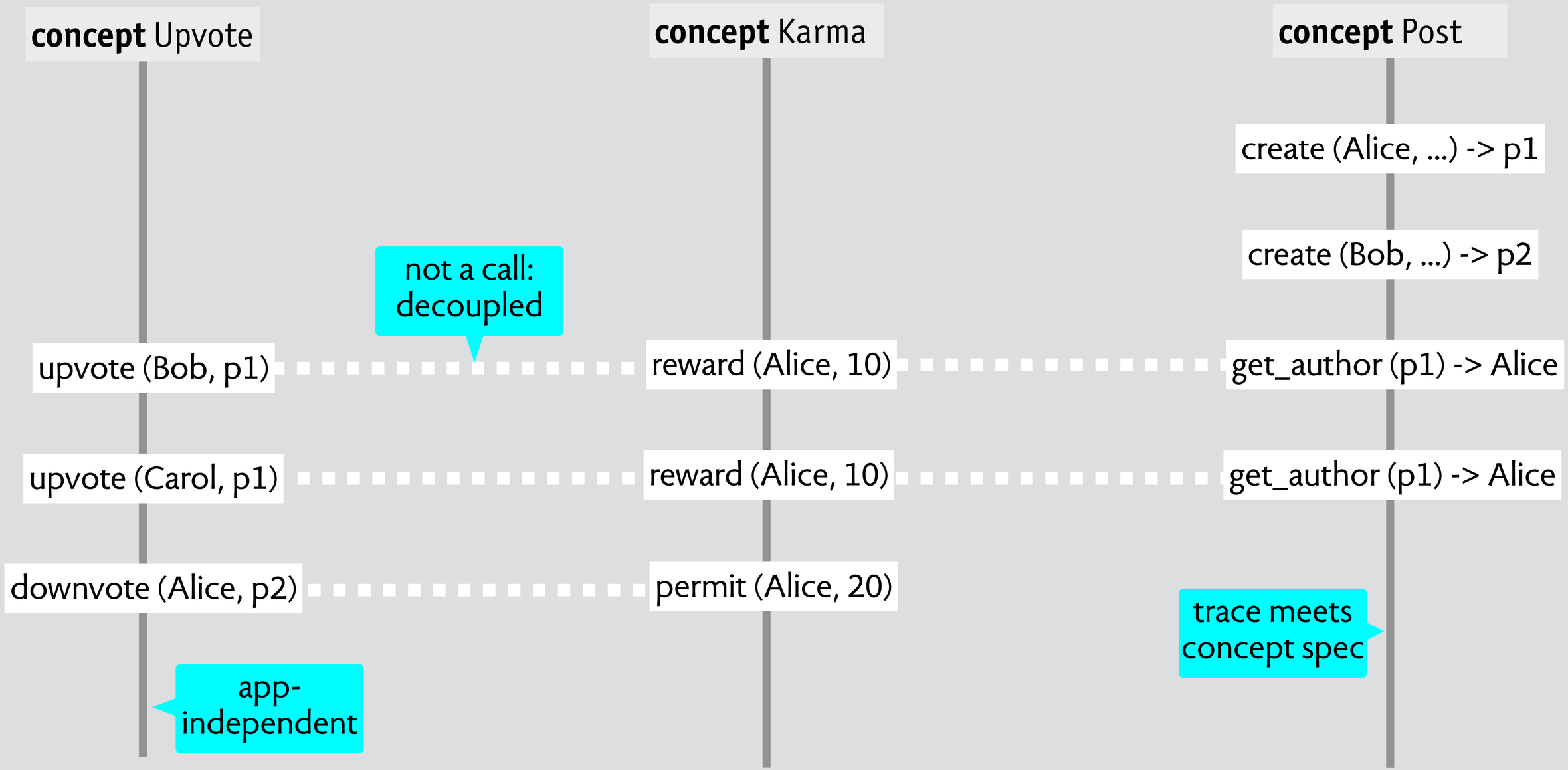
concept Post

actions
create (u: User, t: Text): Post
delete (p: Post)
edit (p: Post, t: Text)
get_author (p: Post): User

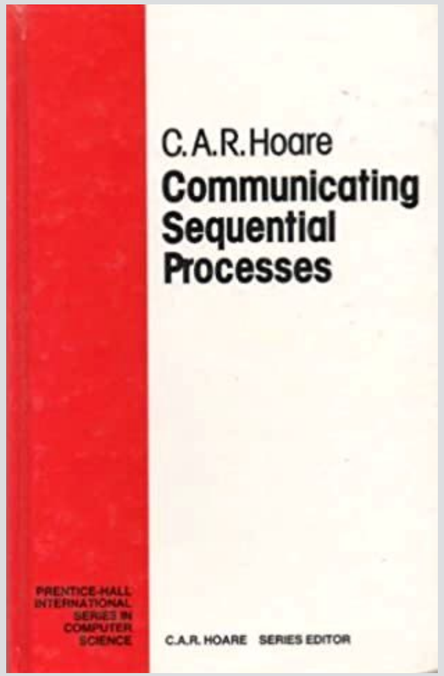
synchronizing concepts



key properties



not a new idea



composition uses event sync from Hoare's CSP

Mediators:

Easing the Design and Evolution of Integrated Systems

Kevin J. Sullivan

Technical Report 94-08-01

Department of Computer Science and Engineering

University of Washington

mediator pattern subject of Sullivan's thesis

coding
hacker news

let's build it with OOP

```
class User {
  String name;
  String password;
  User register (n, p) { ... }
  User authenticate (n, p) { ... }
 }
```

```
class Post {
  User author;
  String body;
  Post new (a, b) { ... }
}
```

adding upvoting

```
class User {
  String name;
  String password;
  User register (n, p) { ... }
  User authenticate (n, p) { ... }
}
```

```
class Post {
  User author;
  String body;
  Set [User] ups, downs;
  Post new (a, b) { ... }
  upvote (u) { ... }
  downvote (u) { ... }
}
```

adding karma

```
class User {
  String name;
  String password;
  int karma;
  User register (n, p) { ... }
  User authenticate (n, p) { ... }
  incKarma (i) { ... }
  bool hasKarma (i) { ... }
}
```

```
class Post {
  User author;
  String body;
  Set [User] ups, downs;
  Post new (a, b) { ... }
  upvote (u) { ... }
  downvote (u) {
    if u.hasKarma (10)  ... }
}
```

adding commenting

```
class User {
  String name;
  String password;
  int karma;
  User register (n, p) { … }
  User authenticate (n, p) { … }
  incKarma (i) { … }
  bool hasKarma (i) { … }
}
```

```
class Post {
  User author;
  String body;
  Set [User] ups, downs;
  Seq [Post] comments;
  Post new (a, b) { … }
  upvote (u) { … }
  downvote (u) {
    if u.hasKarma (10)  … }
  addComment (c) { … }
}
```

what's wrong with this code?

```
class User {
  String name;
  String password;
  int karma;
  User register (n, p) { ... }
  User authenticate (n, p) { ... }
  incKarma (i) { ... }
  bool hasKarma (i) { ... }
}
```

```
class Post {
  User author;
  String body;
  Set [User] ups, downs;
  Seq [Post] comments;
  Post new (a, b) { ... }
  upvote (u) { ... }
  downvote (u) {
    if u.hasKarma (10)  ... }
  addComment (c) { ... }
}
```

User authentication

Posting

Upvoting

Commenting

Karma

no separation of concerns
Post class contains posting, commenting, upvoting, karma

→ **classes are novel & not reusable**
Post class won't work in an app that doesn't have karma points

dependencies between files
Post class calls *User* class to get karma points

→ **can't be built independently**
to build *Post* class, need *User* class to have been built already

a different way

```
concept User {
  Map [User, String] name;
  Map [User, String] password;
  User register (n, p) { ... }
  User authenticate (n, p) { ... }
}
```

```
concept Karma [U] {
  Map [U, Int] karma;
  incKarma (u, i) { ... }
  hasKarma (u, i) { ... }
}
```

concerns now cleanly separated

coupling is gone: refs are polymorphic

```
concept Post [U] {
  Map [Post, U] author;
  Map [Post, URL] url;
  Post new (a, u) { ... }
}
```

```
concept Upvote [U, I] {
  Map [U, I] ups, downs;
  upvote (u, i) { ... }
  downvote (u, i) { ... }
}
```

```
concept Comment [U, T] {
  Map [Comment, U] author;
  Map [Comment, T] target;
  Map [Comment, String] body;
  Comment new (a, t, b) { ... }
}
```

web apps have this!
called a route

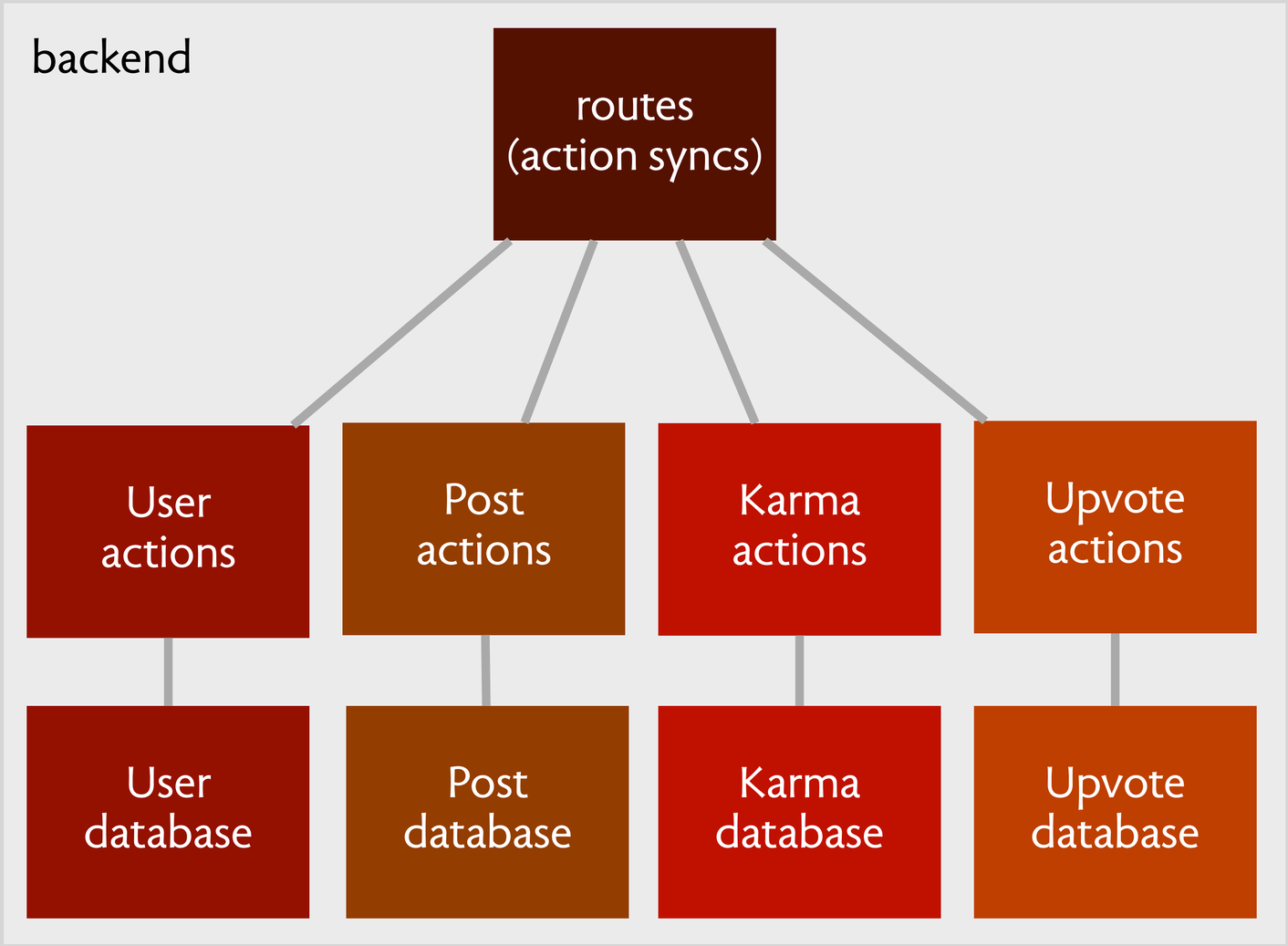
need a mediator
outside the concept

```
sync downvote (u, i) {
  Karma.hasKarma (u, 10)
  Upvote.downvote (u, i)
}
```

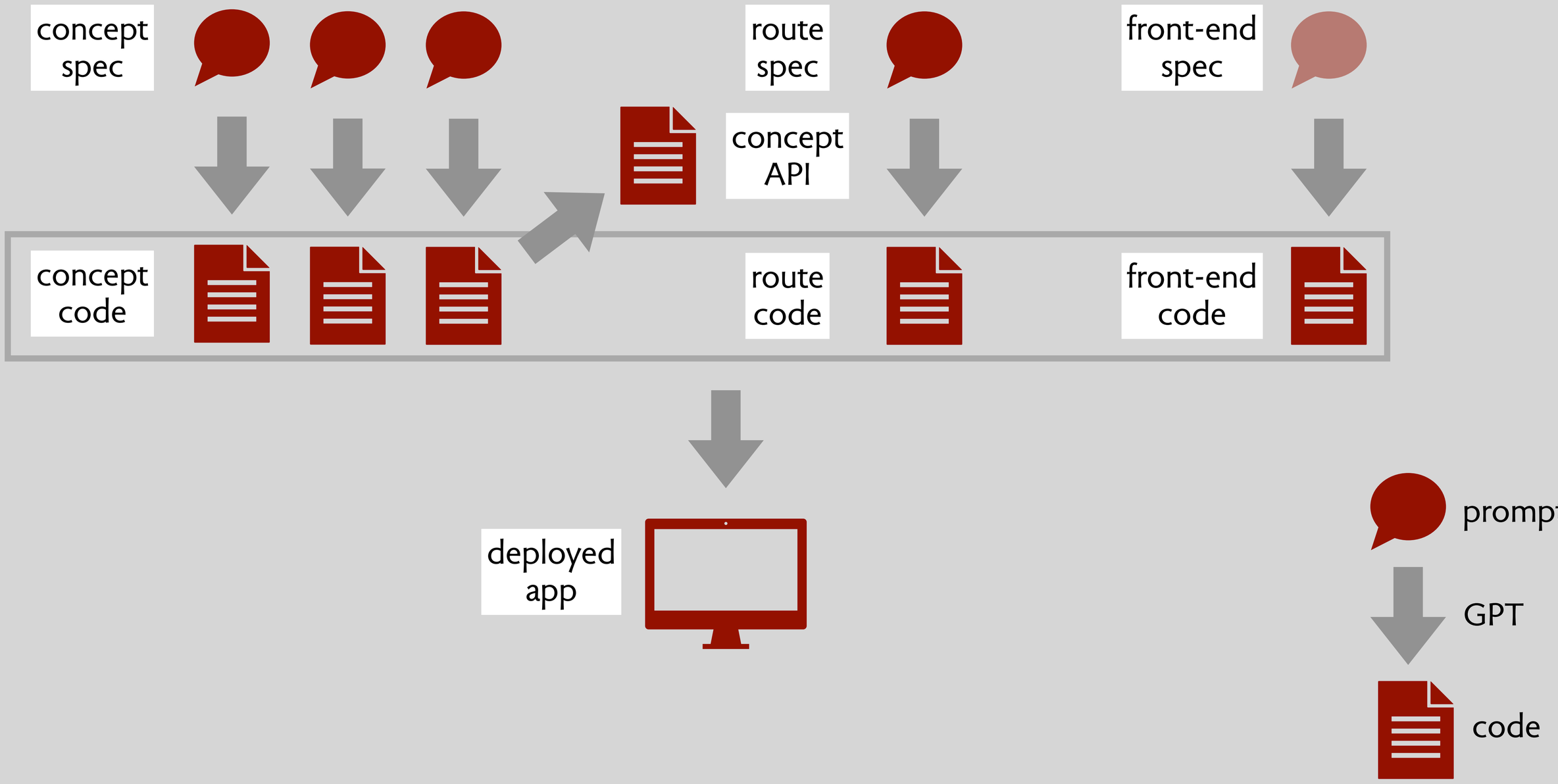
web apps have this!
called a database

need two-way map
target to comment

a new way to structure a web app



now we can generate code!



Kodless: a prototype app generator

based on the work of Abutalib Namazov

creating a concept

Create a new concept

Bulk create

Concept name

Karma

Description

Each karma should have user (generic) and points (number). Actions should be: increase(user, x), decrease(user, x), get(user), isAllowed(user, threshold) (throws error if user doesn't have at least threshold)

Create

brief & application independent

first kodless generates the concept code

Source code:

```
1  import { ObjectId } from "mongodb";
2  import DocCollection, { BaseDoc } from "../framework/doc";
3  import { NotAllowedError } from "../framework/errors";
4
5  interface KarmaDoc extends BaseDoc {
6    user: ObjectId;
7    points: number;
8  }
9
10 export default class KarmaConcept {
11   public readonly karmas: DocCollection<KarmaDoc>;
12
13   constructor(collectionName: string) {
14     this.karmas = new DocCollection<KarmaDoc>(collectionName);
15   }
16
17   async increase(user: ObjectId, x: number) {
18     const karma = await this.karmas.readOne({ user });
19     if (karma) {
20       await this.karmas.collection.updateOne({ user }, { $inc: { points: x } });
21     } else {
22       await this.karmas.createOne({ user, points: x });
23     }
```

database schema etc

typical coding details

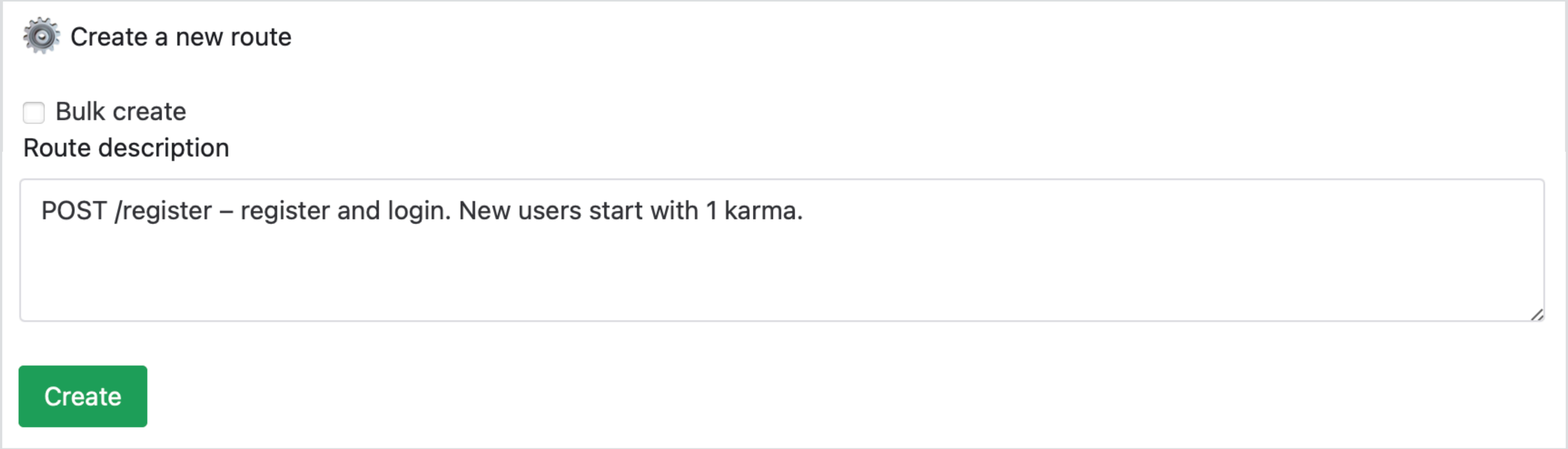
then it generates an API spec from the code

Generated spec:

```
KarmaDoc = {
  user: ObjectId;
  points: number;
}

KarmaConcept = {
  constructor: (collectionName: string) => KarmaConcept; // instantiates a new KarmaConcept with the given collection name
  async increase: (user: ObjectId, x: number) => void; // increases the karma points for a user; creates a new karma record if not exists
  async decrease: (user: ObjectId, x: number) => void; // decreases the karma points for a user; throws NotAllowedError if resulting in negative karma
  async get: (user: ObjectId) => number; // returns the karma p  ser; returns 0 if the user has no karma record
  async isAllowed: (user: ObjectId, threshold: number) => void; // throws NotAllowedError if user's karma points do not meet the threshold
}
```

spec is more detailed than prompt

creating a route



no explicit mention of concepts or their actions

kodless generates the route code

Delete

```
@Router.post("/register")
  async createUser(session: WebSessionDoc, username: string, password: string) {
    WebSession.assertLoggedOut(session);
    const { msg, user } = await User.create(username, password);
    await Karma.increase(user._id, 1);
    WebSession.start(session, user._id);
    return { msg };
  }
```

Kodless uses concept APIs to select actions

hacked news: home page

Hacked News new | past | comments | ask | show | jobs | submit login

1. ▲ Show HN: Kodless -- make websites with AI (github.com)
   1 point by dnj a day ago | hide | discuss
2. ▲ Monolith -- CLI tool for saving web pages as a single HTML file (crates.io)
   1 point by rust 9 days ago | hide | discuss
3. ▲ Show HN: Kodless -- make websites with AI (github.com)
   3 points by kodless 9 days ago | hide | 2 comments
4. ▲ Software = concepts (essenceofsoftware.com)
   6 points by dnj 9 days ago | hide | 2 comments
5. ▲ Jobs HN: I am hiring a wine expert
   1 point by recruiter 10 days ago | hide | discuss
6. ▲ Show HN: Fuiz -- free, open-source and privacy-friendly alternative to Kahoot (fuiz.us)
   3 points by best_dev 10 days ago | hide | 7 comments
7. ▲ Ask HN: How many bugs do you have per line of code?
   1 point by big_asker 10 days ago | hide | discuss
8. ▲ Github -- Use this website to share your code (github.com)
   2 points by barish2 10 days ago | hide | discuss

Guidelines | FAQ | Lists | API | Security | Legal | Apply to YC | Contact

Search:

hacked news: thread

Hacked News new | threads | past | comments | ask | show | jobs | submit kodless (3) | logout

Software = concepts (essenceofsoftware.com)
6 points by dnj 9 days ago | unvote | hide | favorite | 2 comments

add comment

* kodless 9 days ago | next [–]

dnj, you should check out my platform Kodless -- it helps you generate software with concepts without writing any code.

reply

dnj 9 days ago | next [–] unvote

That's great to see! Have you seen GPT-powered concept tutor? https://essenceofsoftware.com/studies/larger/tutor/

reply

Guidelines | FAQ | Lists | API | Security | Legal | Apply to YC | Contact

Search:

hacked news: user profile

Hacked News new | threads | past | comments | ask | show | jobs | submit kodless (3) | logout

user: kodless
created: 2024-03-25T19:14:36.316Z
karma: 3
about:

submissions
threads
favorites
hidden

change top bar color

Guidelines | FAQ | Lists | API | Security | Legal | Apply to YC | Contact

Search:

hacked news: posts by date

Hacked News new | threads | past | comments | ask | show | jobs | submit kodless (3) | logout

previous day | next day | 2024-03-25

1. Software = concepts (essenceofsoftware.com)
   6 points by dnj 9 days ago | unvote | hide | 2 comments
2. * Show HN: Kodless -- make websites with AI (github.com)
   3 points by kodless 9 days ago| hide | 2 comments
3. Show HN: Fuiz -- free, open-source and privacy-friendly alternative to Kahoot (fuiz.us)
   3 points by best_dev 10 days ago | unvote | hide | 7 comments
4. Github -- Use this website to share your code (github.com)
   2 points by barish2 10 days ago | unvote | hide | discuss
5. ▲ Ask HN: How many bugs do you have per line of code?
   1 point by big_asker 10 days ago| hide | discuss

Guidelines | FAQ | Lists | API | Security | Legal | Apply to YC | Contact

Search:

successes & challenges

coverage of all major functionality
flagging, hiding, karma, etc
nested comments, complicated ranking rules
many application-specific details (eg, karma)

minimal prompting required
< 1,000 words in total in prompts

no code editing needed
some prompt revisions to tweak behavior

frontend code not yet fully automated
but have designed HTMx like DSL
generation not as hard as backend?

database queries don't cross concepts
so need actions like Post.getByIds

implementation details in a few route specs
for packaging of objects in responses

a DSL for synchronization (Eagon Meng)

more declarative, eg with implicit matching

separate core logic from presentation

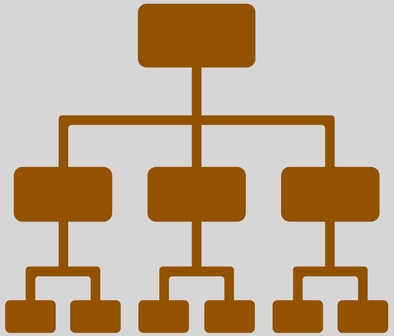
factor out persistent storage from concepts expressed in plain JS

```
when API.request("update_article", token, slug, title, description, body, tagList) -> request_id
    JWT.verify(token) -> user_id
    Article.getIdBySlug(slug) -> article_id
    Article.getAuthor(article_id) -> user_id
sync
    Article.update(article_id, title, description, body)
    Tag.update(article_id, tagList)
    API.return("article", article_id, user_id, request_id)
```

part 4:
concepts in dev teams

a view of the software problem

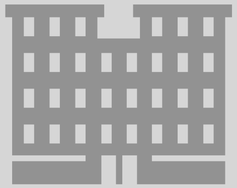
unbounded functionality



uncontrolled complexity

in minds of users & devs

in the product code



in the company culture

working with development organizations

three major applications of concept design so far
main motivator is alignment across roles & products
building a catalog of organization-specific concepts

alignment across roles
experience architects vs. software architects
product managers vs. engineers

alignment across products
the same purpose but different behavior & terminology
a natural consequence of independent teams, acquisitions

concept catalog
not just the concept spec: the social context, design issues
Palantir bootstrapped with ontology nodes

Palantir

AUTODESK.

axim

takeaways

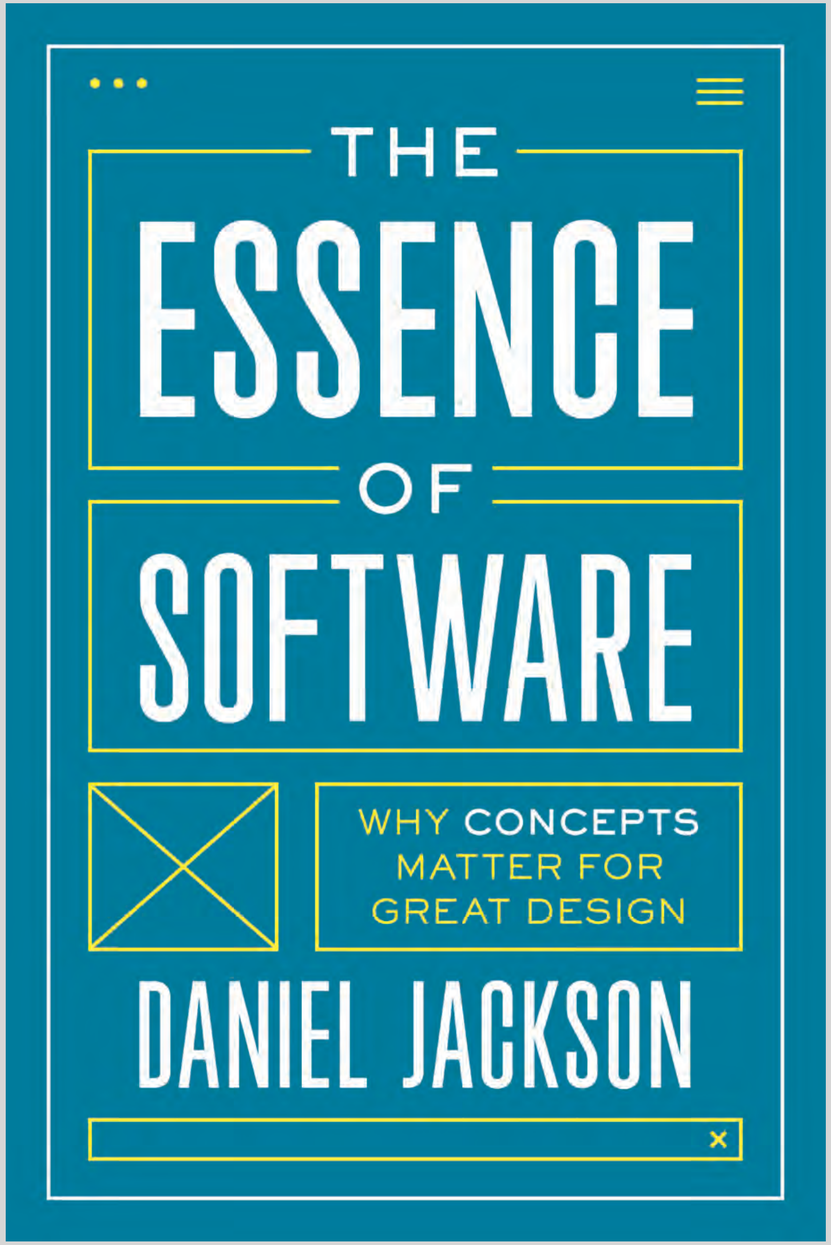
When you go to design a house you talk to an architect first, not an engineer. Why is this?

Because the criteria for what makes a good building fall outside the domain of engineering.

Similarly, in computer programs, the selection of the **various components** must be driven by the conditions of use.

How is this to be done? By software designers.

Mitchell Kapor, *A Software Design Manifesto* (1996)



https://essenceofsoftware.com

concepts are a new way to think about software
can help shape product, improve UX, structure code

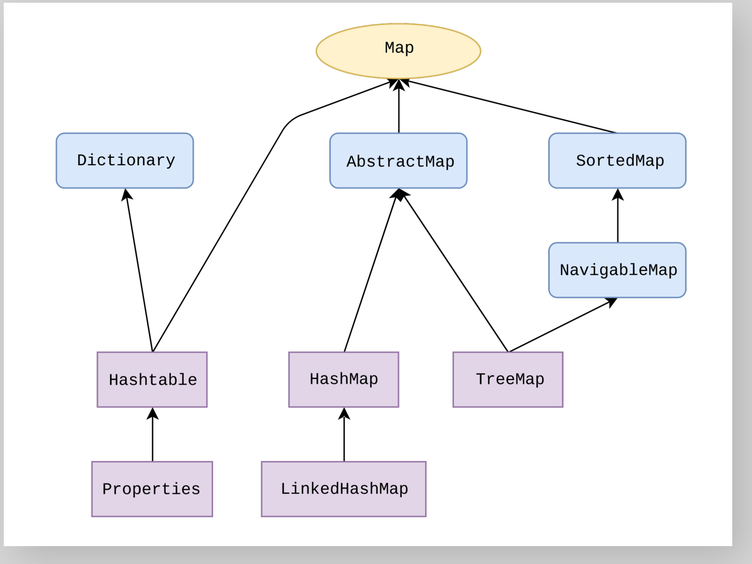
proof of the pudding
hope you'll try concepts to see strengths & limitations

on the website
tutorials, blog posts, discussion forum, subscribe

let's stay in touch!
email me at dnj@mit.edu

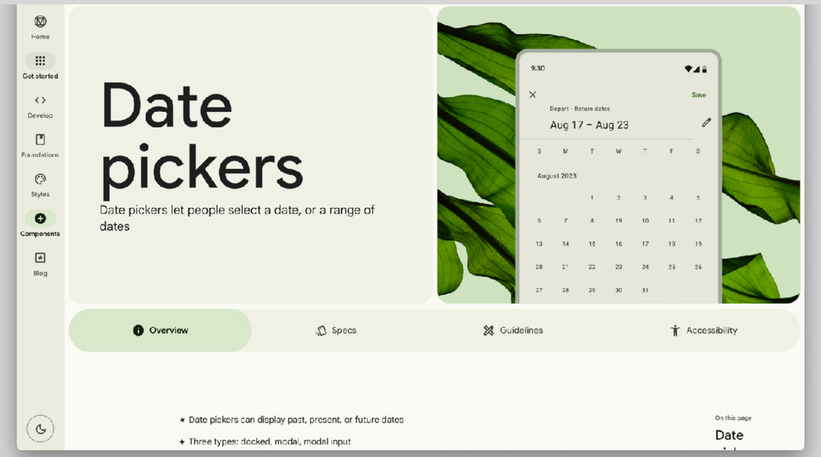
unused slides

great strategies for taming complexity



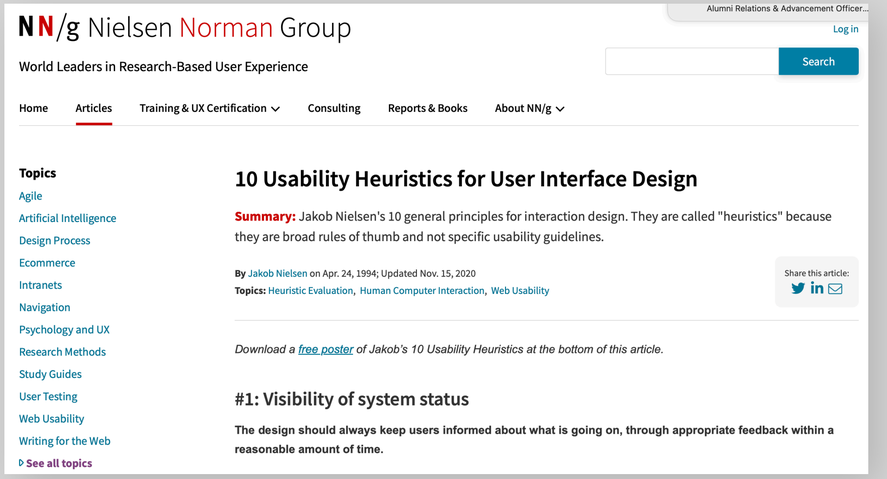
modularity

ways to structure **code**



reuse & familiarity

standards for **UIs**



focus on users

broad **UX** principles

needed: a framework for **designing functionality** that aligns modularity, reuse and user-centeredness

a language for synchronization

based on the work of Eagon Meng

a kodless route

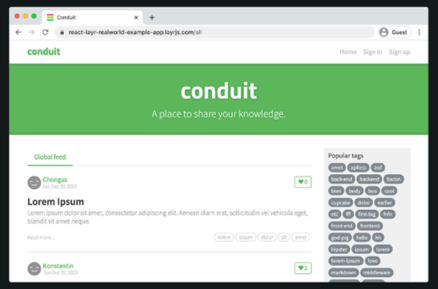
```
@Router.post("/articles")
async createArticle(session: WebSessionDoc, article: ArticleMessage) {
  const userId = WebSession.getUser(session);
  const profile = await Profile.getProfileById(userId);
  const newArticle = await Article.create(userId, article.title, article.description, article.body);

  if (article.tagList.length != 0) await Tag.create(newArticle._id, article.tagList);

  const profileMessage = Merge.createProfileMessage(profile, true);
  return { article: Merge.createArticleMessage(newArticle, profileMessage, article.tagList, false, 0) };
}
```

magic in decorator binds params

must handle cases for same route

magic in Express creates response

hack to assemble response



The mother of all demo apps

See how the exact same application is built using different libraries and frameworks.

Frontend | Backend | Fullstack

LANGUAGES

- All
- Java
- TypeScript
- Go
- Python
- JavaScript
- Kotlin
- C#
- Rust
- PHP
- Scala

.NET + Minimal API Erikvdv/realworldapiminimal	C#
ActixWeb + Diesel snamiki1212/realworld-v1-rust-actix-web-diesel	Rust
Adonis Utwo/adonis-realworld-example-app	JavaScript
Adonis giuliana-bezerra/adonisjs-tdd-typescript-example-app	TypeScript
AIOHTTP + SQLAlchemy + asyncpg + The Clean Architecture stkrizh/realworld-aiohttp	Python

idea: synchronizations in the user's language

"when a request to create an article is received,
verify the web token and get the user identifier
create the article with the provided title, etc
add the given tags to the article
return the article as a result"

functional: no hidden or shared state

no control flow: match on properties instead

no implementation details: just concept actions

reacting to web requests

reacting to API requests

"when a request to create an article is received,
verify the web token and get the user identifier
create the article with the provided title, etc
add the given tags to the article
return the article as a result"

```
when API.request("create_article", token, title, description, body, tagList) -> request_id
sync JWT.verify(token) -> user_id
    Article.create(title, description, body, user_id) -> article_id
    Tag.create(article_id, tagList)
    API.return("article", article_id, user_id, request_id)
```

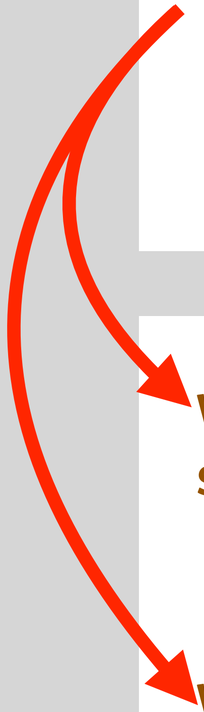
matching args

```
when API.request("update_article", token, slug, title, description, body, tagList) -> request_id
    JWT.verify(token) -> user_id
    Article.getIdBySlug(slug) -> article_id
    Article.getAuthor(article_id) -> user_id
sync
    Article.update(article_id, title, description, body)
    Tag.update(article_id, tagList)
    API.return("article", article_id, user_id, request_id)
```

reacting to concept actions

```
when API.request("delete_article", token, slug) -> request_id
sync JWT.verify(token) -> user_id
    Article.getIdBySlug(slug) -> article_id
    Article.delete(article_id)
    API.respond("Article deleted", request_id)
```

```
when Article.delete(article_id)
sync Comment.byTarget(article_id) -> comments
    Comment.deleteMany(comments)

when Article.delete(article_id)
sync Tag.delete(article_id)
```



packaging responses

API response format

JSON Objects returned by API:

Make sure the right content type like `Content-Type: application/json; charset=utf-8` is correctly returned.

Users (for authentication)

```
{
  "user": {
    "email": "jake@jake.jake",
    "token": "jwt.token.here",
    "username": "jake",
    "bio": "I work at statefarm",
    "image": null
  }
}
```

mixes concepts!
User, Profile, Token

Profile

```
{
  "profile": {
    "username": "jake",
```

```
    "bio": "I work at statefarm",
    "image": "https://api.realworld.io/images/smiley-cyrus.jpg",
    "following": false
  }
}
```

Single Article


```
{
  "article": {
    "slug": "how-to-train-your-dragon",
    "title": "How to train your dragon",
    "description": "Ever wonder how?",
    "body": "It takes a Jacobian",
    "tagList": ["dragons", "training"],
    "createdAt": "2016-02-18T03:22:56.637Z",
    "updatedAt": "2016-02-18T03:48:35.824Z",
    "favorited": false,
    "favoritesCount": 0,
    "author": {
      "username": "jake",
      "bio": "I work at statefarm",
      "image": "https://i.stack.imgur.com/xHWG8.jpg",
      "following": false
    }
  }
}
```


yikes! how do generate this?

reacting to API return: generating presentation for response

```
when API.request("get_article", slug) -> request_id
sync Article.getIdBySlug(slug) -> article_id
    API.return("article", article_id, "", request_id)
```

```
when API.return("article", article_id, user_id, request_id)
sync Article.get(article_id) -> article
    Article.getAuthorId(article_id) -> author_id
    User.getPublic(author_id) -> author
    Follows.doesFollow(user_id, author_id) -> following
    Profile.get(author_id) -> profile
    Map.mergeAs("author", author, following, profile) -> author_info
    Favorites.isFavorited(user_id, article_id) -> favorited
    Favorites.favoritesCount(article_id) -> favoritesCount
    Tag.get(article_id) -> tags
    Map.mergeAs("article", article, tags, author_info, favorited, favoritesCount) -> response
    API.respond(response, request_id)
```

adjusting behavior

making an impactful change

a problem with the standard spec of RealWorld

the feed only includes articles from users you follow

however much you favorite articles, it has no effect on your feed

a solution: just add this sync

```
when Favorites.create(user_id, article_id)
sync Article.getAuthor(article_id) -> author_id
    Follows.create(user_id, author_id)
```

don't have to worry about when/where favorites created

augmenting behavior

conduit

Home | New Article | Settings | Definitely not a bot

The benefits of immutability

Programming with immutability

Prompt: An article about the benefits of programming with immutability

immutability

× programming × immutability × functional

Publish Article

The benefits of immutability

Definitely not a bot
Tue Nov 12 2024

The Benefits of Programming with Immutability

In recent years, immutability has gained popularity across various programming paradigms. While it might seem like a buzzword to some, the advantages of programming with immutability are backed by solid principles and can lead to more robust software development practices. In this article, we will explore what immutability is, its benefits, and practical ways to implement it in your programming.

What is Immutability?

Immutability refers to the property of an object whose state cannot be modified after it has been created. In programming languages that support immutability, once you instantiate an immutable object, you cannot change its values. Instead, modifications involve creating a new instance of the object with the desired changes.

Example of Immutability in JavaScript

Consider a simple example in JavaScript:

augmenting behavior

```
when Article.create(title, description, body, user_id) -> article_id
    OpenAI.isPrompt(body) -> true
sync OpenAI.basePrompt() -> messages
    OpenAI.addArticlePrompt(messages) -> system_prompt
    OpenAI.addUserMessage(system_prompt, body) -> prompt
    OpenAI.getSingleResponse(prompt) -> generated
    Article.update(article_id, title, description, generated)
```

factoring out data persistence

user concept code (Mongo)

```ts
import { ObjectId } from "mongodb";
import DocCollection, { BaseDoc } from "../framework/doc";
import { BadValuesError, NotAllowedError, NotFoundError } from "./errors";

export interface UserDoc extends BaseDoc {
  username: string;
  password: string;
}

export default class UserConcept {
  public readonly users: DocCollection<UserDoc>;

  constructor(name: string) {
    this.users = new DocCollection<UserDoc>(name);

    // Create index on username to make search queries for it
    void this.users.collection.createIndex({ username: 1 });
  }

  async create(username: string, password: string) {
    await this.assertGoodCredentials(username, password);
    const _id = await this.users.createOne({ username, password });
    return { msg: "User created successfully!", user: await this.users.readOne({ _id }) };
  }

```

mongo-db specific details everywhere

a better way

```ts
import * as bcrypt from "https://deno.land/x/bcrypt@v0.4.1/mod.ts";
import uuid from "../imports/uuid.ts";

interface User {
    username: string;
    password: string;
    email: string;
}
// {user_id: user}
type Users = Record<string, User>;

export default class UserConcept {
    async signup(
        state: Users,
        username: string,
        email: string,
        password: string,
    ) {
        this.checkUnique(state, username, email);
        if (username.length === 0 || password.length === 0) {
            throw Error("Username/Password must not be empty");
        }
        const user_id = uuid();
        const hashed = await bcrypt.hash(password);
        const user = { username, email, password: hashed };
        state[user_id] = user;
        return [state, user_id];
    }
```

only dependency is for hashing

plain old JS

explicit state: you'll see why later

factoring out data persistence

```
when Sync.run(action, arguments)
    Concept.lookup(action) -> concept
    State.get(concept) -> before
sync Sync.invoke(action, before, arguments) -> after, returns
    State.compare(before, after) -> diff
    State.update(concept, diff)
```

uses Immer framework for immutability

```
when Sync.run(action, arguments)
    Operational.lookup(action) -> operation
sync Sync.execute(operation, arguments) -> returns
```

```
when State.update(concept, diff)
    MongoDB.getCollection(concept) -> collection
sync MongoDB.update(collection, diff)
```