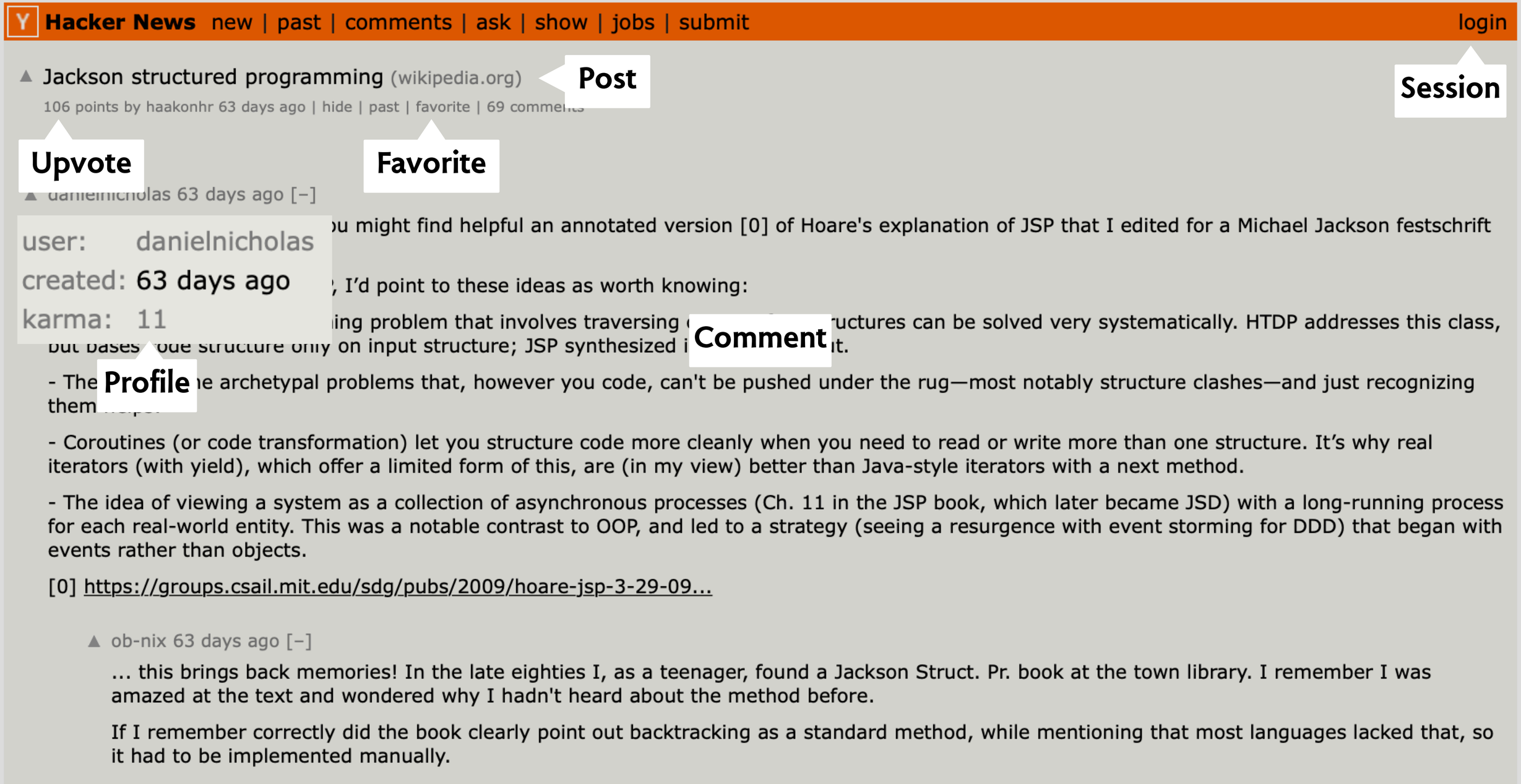
designing synchronizations

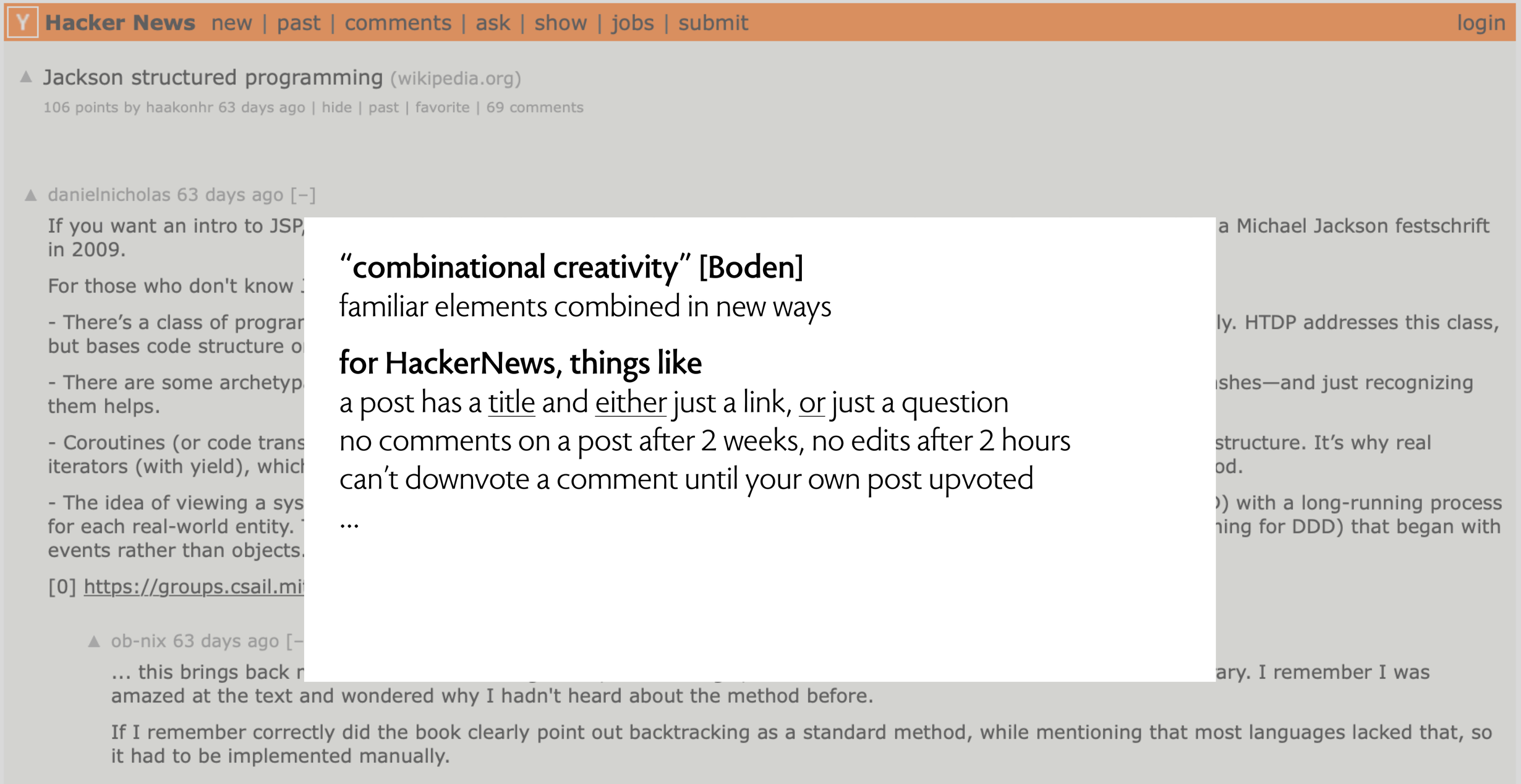
Daniel Jackson · Autodesk · Woodinville, WA · Dec 3-5, 2024

designing Hacker News: sync as composition

an app composed of familiar concepts



... 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

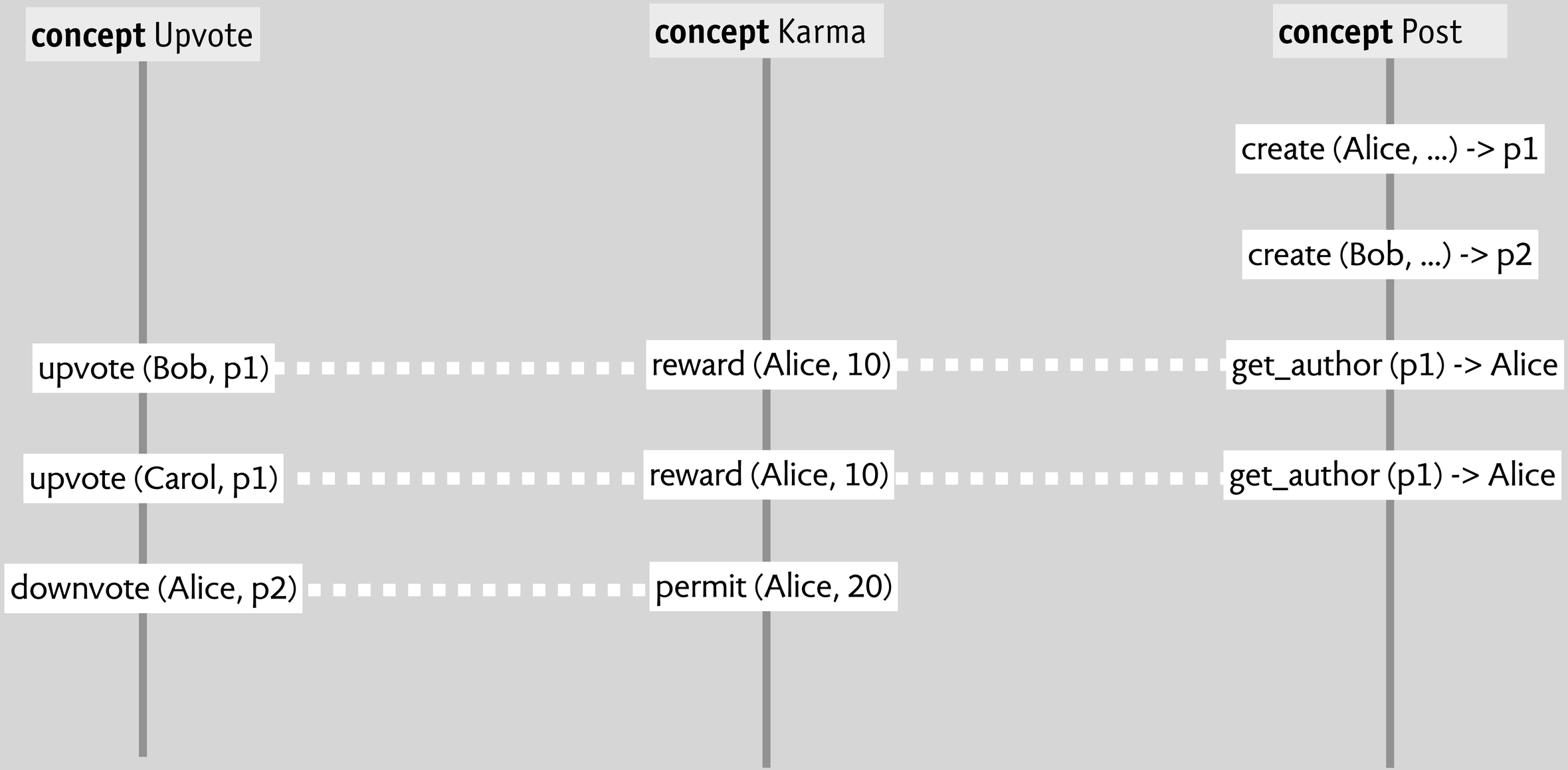
actions
upvote (u: User, i: Item)
downvote (u: User, i: Item)
unvote (u: User, i: Item)

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

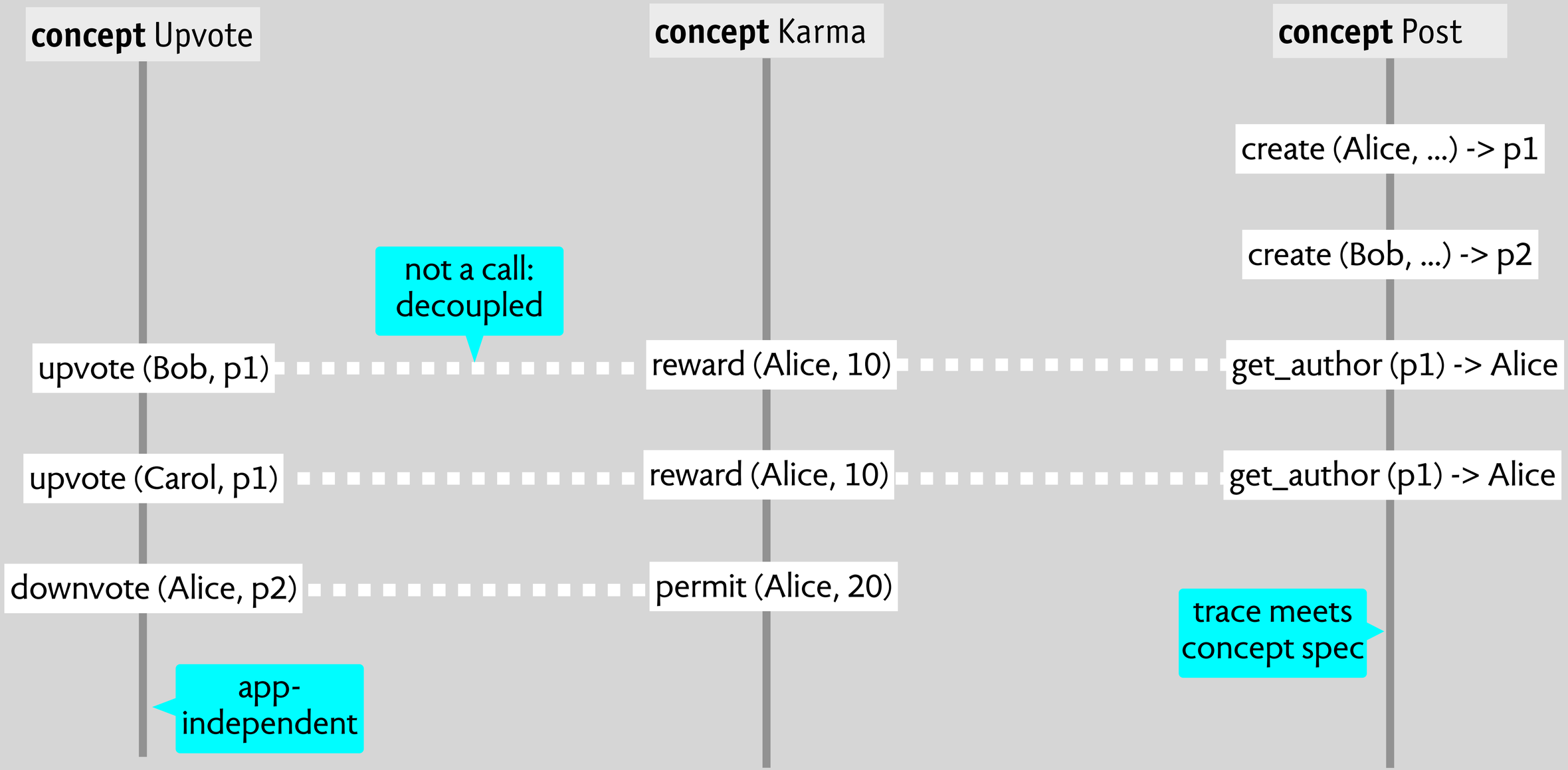
concept Post

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

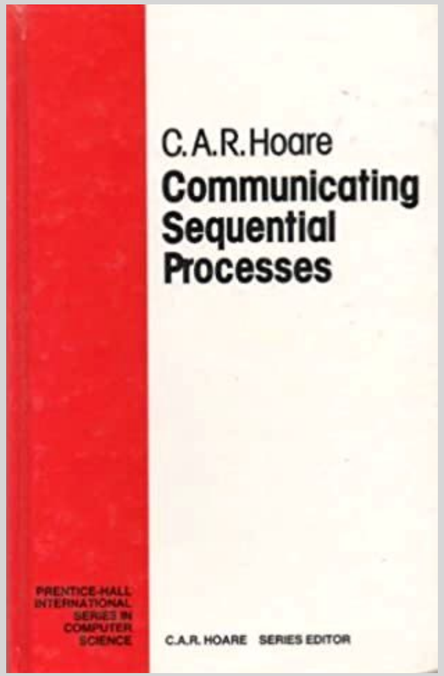
synchronizing concepts



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

writing down synchronizations

Upvote/Post/Karma syncs

```
when Upvote.upvote(user_id1, post_id)
sync
  Post.get_author(post_id) -> user_id2
  Karma.reward(user_id1, 10)
```

```
when Upvote.downvote(user_id1, post_id)
sync
  Karma.permit(user_id1, 20)
```

in a web request setting

```
when HTTP.request("edit_post", token, post_id, content, labels) -> request_id
    JWT.verify(token) -> user_id
    Post.get_author(post_id) -> user_id
sync Article.getIdBySlug(slug) -> article_id
    Post.update(post_id, content)
    Labeling.update(post_id, labels)
    HTTP.respond("post", post_id, user_id, request_id)
```

cascading deletes

```
when HTTP.request("delete_post", token, post_id) -> request_id
sync
  JWT.verify(token) -> user_id
   Post.delete(post_id)
   HTTP.respond(“Post deleted", request_id)
```

```
when Post.delete(post_id)
sync Comment.byTarget(post_id) -> comments
    Comment.deleteMany(comments)

when Post.delete(post_id)
sync Labeling.delete(post_id)
```

check your understanding

true or false?

1. syncs are transactional: every action occurs or none of them
2. syncs are bidirectional: if A is sync'd with B, then B is sync'd with A
3. syncs fire based on just one action
4. a bad sync can break a concept
5. adding syncs adds new concept behaviors
6. coding syncs requires a novel framework or language
7. too many syncs may damage performance

tighten/loosen synchronization as a design move

tighten-loosen design moves: tradeoff automation/flexibility

light pull / door lock

tighten

airplane toilet lock

dimmers with separate controls

loosen

rotary dimmer switch

Schindler's PORT elevator

Schindler's PORT elevator

A boost to traffic performance

Schindler PORT groups passengers by destination to provide the shortest possible trip for every rider, avoiding chaotic elevator runs and random, multiple stops.

User-friendly operation

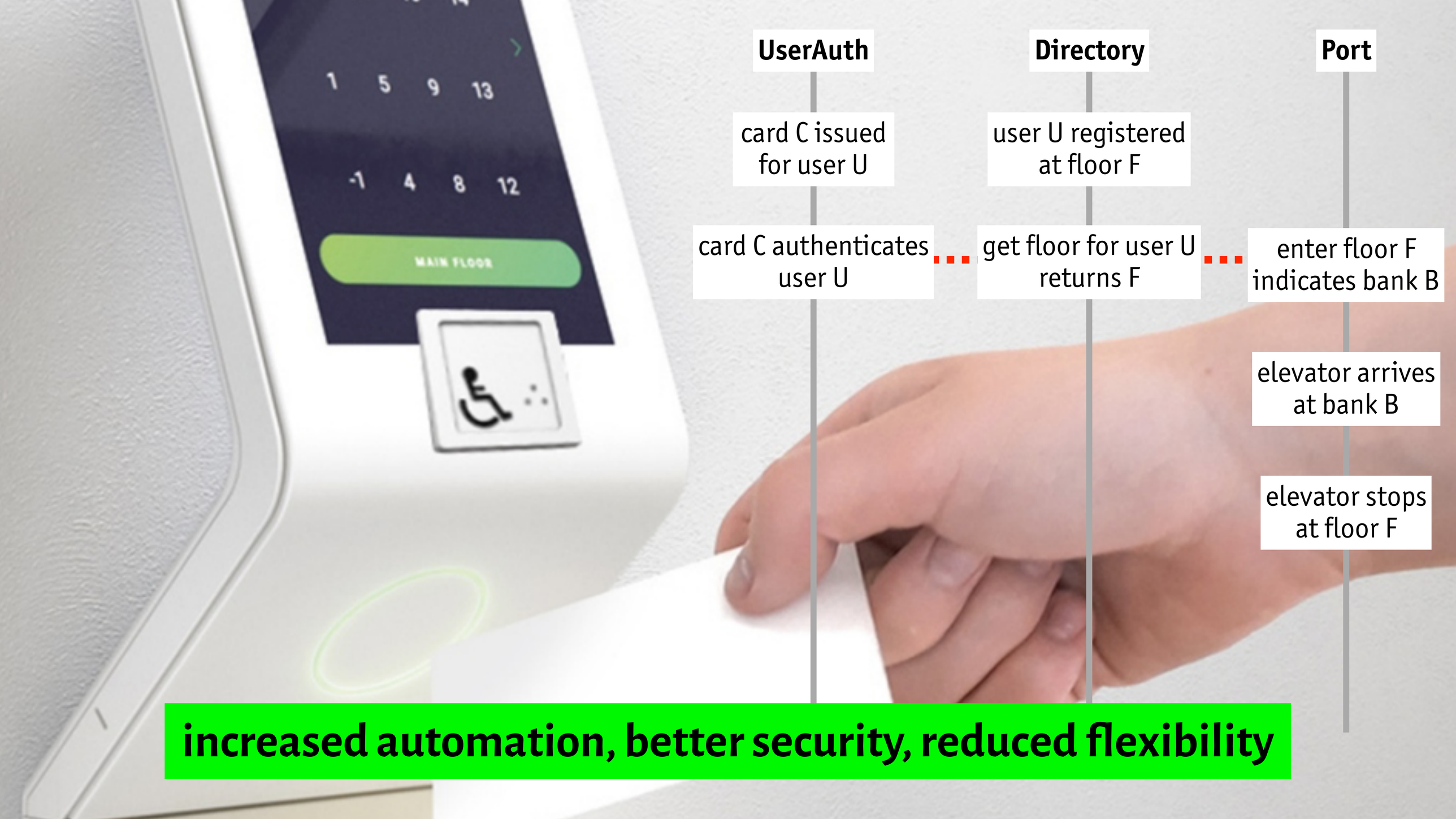
With Schindler PORT's universal card reader, a rider simply swipes a personal access card at the Schindler PORT terminal for an elevator car to be immediately assigned with an approved destination.

Excellent personalized service

Schindler PORT can customize your journey, whether it be a special VIP trip, a streamlining patient transport or allowing for more approach time, longer door opening time or additional space for passengers with special needs.

Enhanced building security

Floor access is defined by the building management through access cards or smartphones. Schindler PORT can also be integrated with turnstiles to further enhance building security.



increased automation, better security, reduced flexibility

a surprising
synchronization
in Google calendar

AS

Arvind Satyanarayan November 15, 2018 at 2:04 PM

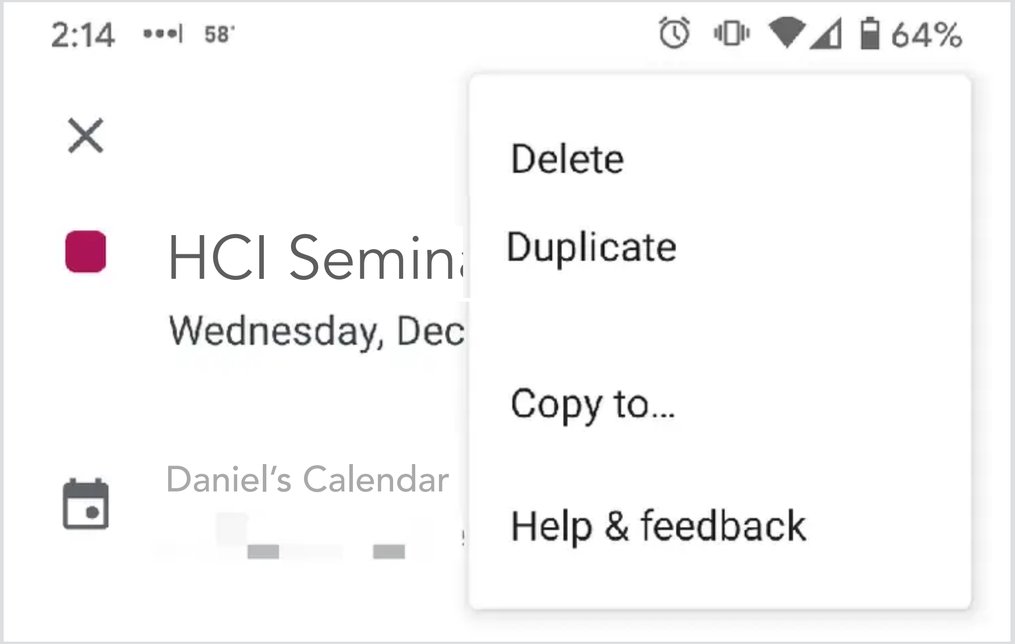
Re: TALK: Monday 11-19-2018 Kanit (Ham) Wongsuphasawat: No...

Cc: seminars@csail.mit.edu, HCI-Seminar@lists.csail.mit.edu

Details

This message is from a mailing list. Unsubscribe

Despite some erroneous messages sent to this list accidentally, Kanit's talk is happening! Please join us on Monday.



Canceling and deleting events in the Google Calendar mobile app is similar to on a desktop.

1. First, open Google Calendar.
2. Tap on the event you wish to cancel.
3. Press on the three dots in the top right corner of the event window.
4. Select Delete.
5. Tap Delete event. Google Calendar will send a cancellation email to the guests.

Mar 22, 2021

https://wpamelia.com › Blog

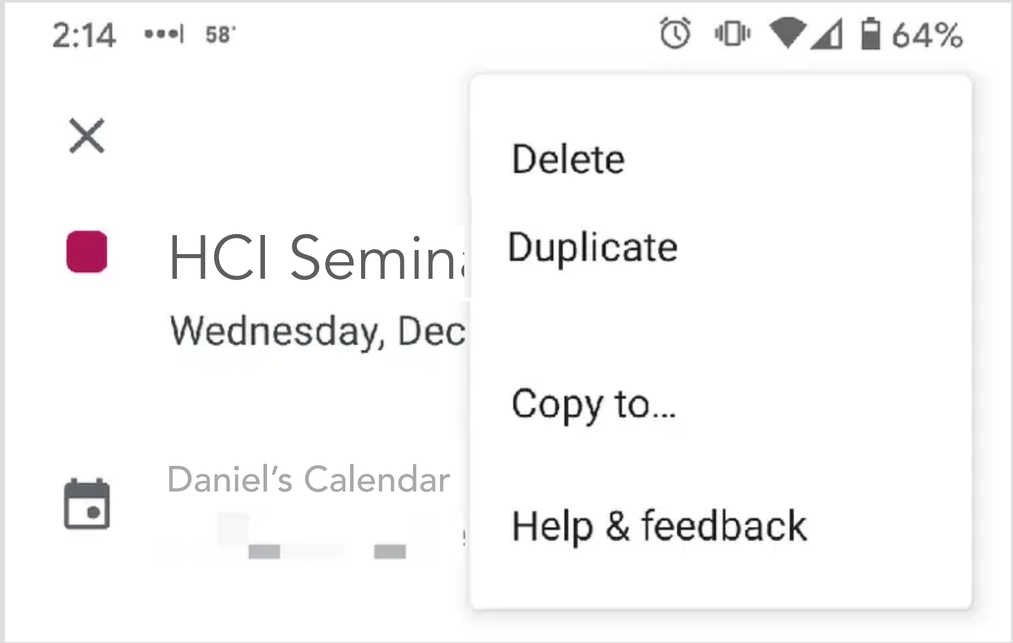
How to Cancel an Event in Google Calendar - Amelia booking ...

seminar announced as **email** to listserv with attached calendar event

event **installed** automatically in user's calendar

user **deletes** event from calendar

cancellation email automatically sent to other invitees



concept Calendar

purpose record upcoming engagements

actions

create an event

delete an event

...

concept Invitation

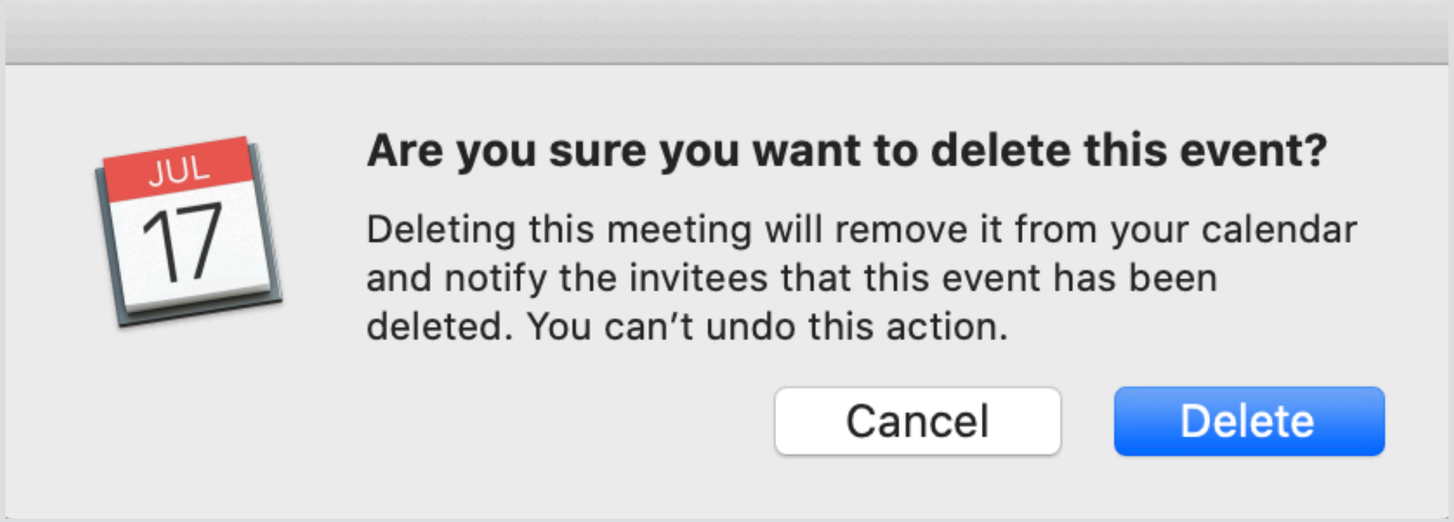
purpose coordinate event participants

actions

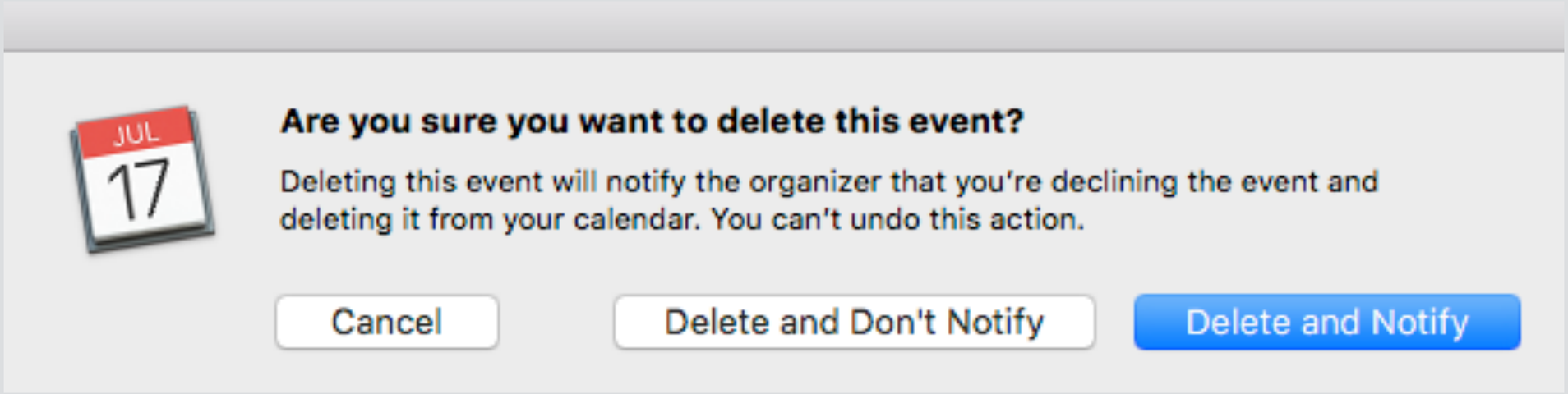
accept invitation

decline invitation

...



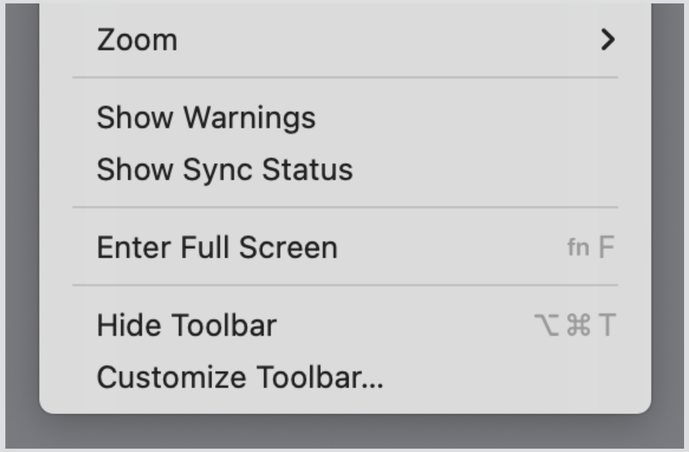
a long time problem in iCal too
how to delete spam calendar events?



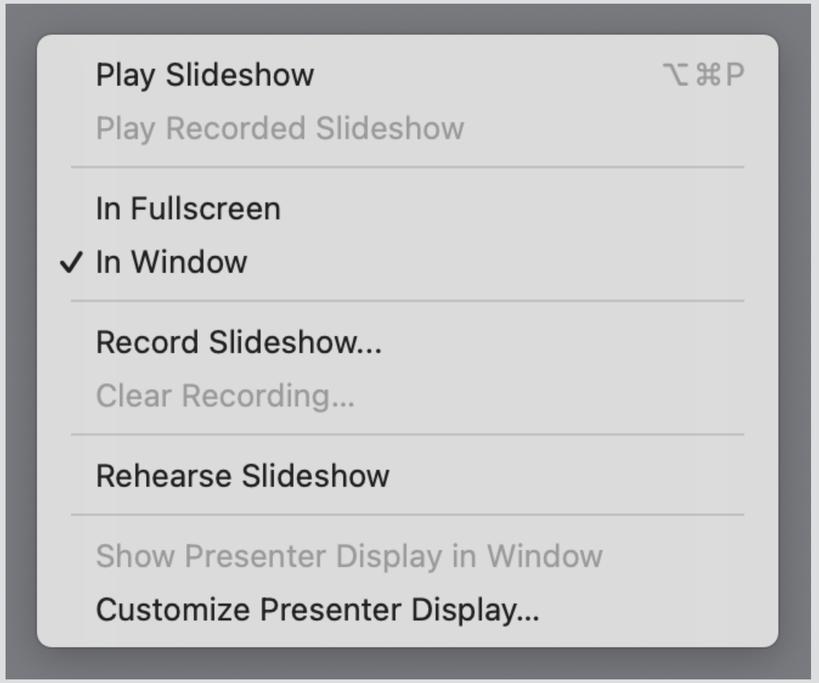
resolution to design problem
make sync optional

the evolution of full screen toggle

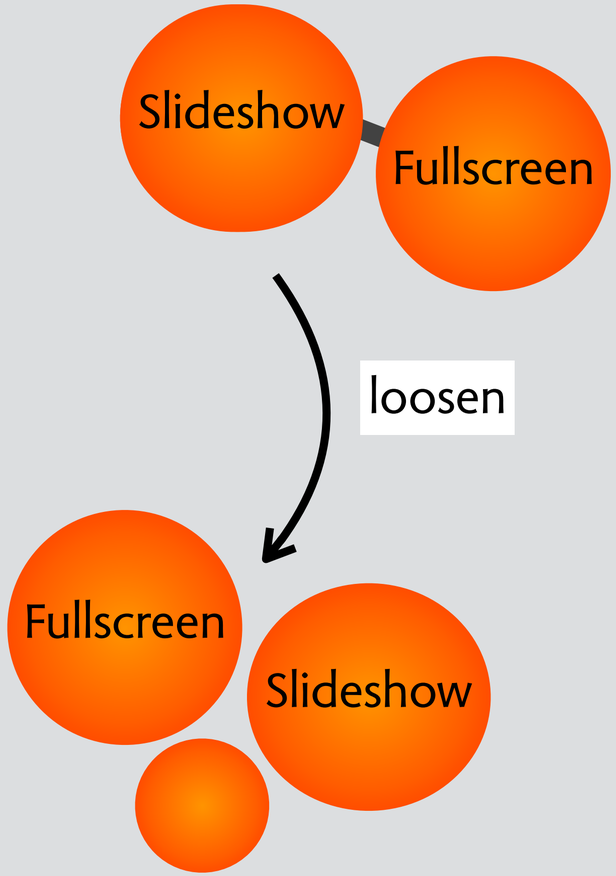
loosen: slideshow and full screen



full screen toggle emerges as concept (c. 2010?)



play-in-window option turn off synchronization (2021)



exercise: separation of concerns

Y **Hacker News** new | past | comments | ask | show | jobs | submit login

▲ Jackson structured programming (wikipedia.org)

106 points by haakonhr 63 days ago | hide | past | favorite | 69 comments

user session

▲ danielnicholas 63 days ago [–]

If you want an intro to JSP, you might find helpful an annotated version [0] of Hoare's explanation of JSP that I edited for a Michael Jackson festschrift in 2009.

For those who don't know JSP, I'd point to these ideas as worth knowing:

- There's a class of programming problem that involves traversing context-free structures can be solved very systematically. HTDP addresses this class, but bases code structure only on input structure; JSP synthesized input and output.

- There are some archetypal problems that, however you code, can't be pushed under the rug—most notably structure clashes—and just recognizing them helps.

- Coroutines (or code transformation) let you structure code more cleanly when you need to read or write more than one structure. It's why real iterators (with yield), which offer a limited form of this, are (in my view) better than Java-style iterators with a next method.

- The idea of viewing a system as a collection of asynchronous processes (Ch. 11 in the JSP book, which later became JSD) with a long-running process for each real-world entity. This was a notable contrast to OOP, and led to a strategy (seeing a resurgence with event storming for DDD) that began with events rather than objects.

[0] https://groups.csail.mit.edu/sdg/pubs/2009/hoare-jsp-3-29-09...

▲ ob-nix 63 days ago [–]

... this brings back memories! In the late eighties I, as a teenager, found a Jackson Struct. Pr. book at the town library. I remember I was amazed at the text and wondered why I hadn't heard about the method before.

If I remember correctly did the book clearly point out backtracking as a standard method, while mentioning that most languages lacked that, so it had to be implemented manually.

▲ CraigJPerry 63 days ago [–]

This is referenced(1) as a core inspiration in the preface to "How to Design Programs" but i never researched it further because i've found the "design recipes" approach in htdp to be pretty solid in real life problems

a familiar combination

concept UserAuth

purpose authenticate users

principle
after a user registers with a username and password, they can authenticate as that user by providing a matching username and password

state
registered: **set** User
username, password: registered -> **one** String

actions
register (n, p: String): User
authenticate (n, p: String): User

concept Session [User]

purpose authenticate user for extended period

principle
after a session starts (and before it ends), you can get the user of the session

state
active: **set** Session
user: active -> **one** User

actions
start (u: User): Session
get_user (s: Session): User
end (s: Session)

where is one of these used without the other?

what syncs are needed?

do sessions last forever?

why separate concepts are useful

applications of UserAuth without Session

authenticating one-off actions in operating systems
MacOS: authenticate when opening app for first time
Unix: executing command requiring superuser

reauthenticating mid-session for critical actions
confirming bank transfers

one-time authentication in websites
when cancelling a subscription

applications of Session without UserAuth

authenticating by different means
biometrics such as facial recognition, fingerprint

unauthenticated sessions
in some games and chat apps, user just enters name and name/score shown on leaderboard

synchronizing authentication and sessions

```
when HTTP.request("register", username, password) -> request
sync UserAuth.register (username, password) -> user
   HTTP.response ("register success", request)
when HTTP.request("login", username, password) -> request
sync UserAuth.authenticate (username, password) -> user
   Session.start (user) -> session
  HTTP.response ("login success", session, request)
when HTTP.request("logout", session) -> request
sync Session.end (session)
  HTTP.response ("logout success", request)
when HTTP.request("create_post", content, session) -> request
sync Session.get_user (session) -> user
   Post.create (user, content) -> post
   HTTP.response ("create success", post, request)
```

how to make sessions expire after 5 minutes?

```
when HTTP.request("register", username, password) -> request
sync UserAuth.register (username, password) -> user
  HTTP.response ("register success", request)
when HTTP.request("login", username, password) -> request
sync UserAuth.authenticate (username, password) -> user
  Session.start (user) -> session
  ???
  HTTP.response ("login success", session, request)
when HTTP.request("logout", session) -> request
sync Session.end (session)
  ???
  HTTP.response ("logout success", request)
when HTTP.request("create_post", content, session) -> request
sync Session.get_user (session) -> user
  ???
  Post.create (user, content) -> post
  HTTP.response ("create success", post, request)
```

concept ExpiringResource [Resource]

purpose handle expiration of short-lived resources

principle

if you allocate a resource r for t seconds,
then after t seconds, the resource expires

state

active: set Resource
expiry: Resource -> one Date

actions

allocate (r: Resource, t: int)
deallocate (r: Resource)
renew (r: Resource, t: int)
expired (r: Resource): Bool

a solution

```
when HTTP.request("register", username, password) -> request
sync UserAuth.register (username, password) -> user
  HTTP.response ("register success", request)
when HTTP.request("login", username, password) -> request
sync UserAuth.authenticate (username, password) -> user
  Session.start (user) -> session
  ExpiringResource.allocate (session, 300)
  HTTP.response ("login success", session, request)
when HTTP.request("logout", session) -> request
sync Session.end (session)
  ExpiringResource.deallocate (session)
  HTTP.response ("logout success", request)
when HTTP.request("create_post", content, session) -> request
sync Session.get_user (session) -> user
  ExpiringResource.expired (session) -> false
  Post.create (user, content) -> post
  HTTP.response ("create success", post, request)
when ExpiringResource.expired (session) -> true
sync Session.end (session)
```

concept ExpiringResource [Resource]

purpose handle expiration of short-lived resources

principle

if you allocate a resource r for t seconds,
then after t seconds, the resource expires

state

active: set Resource
expiry: Resource -> one Date

actions

allocate (r: Resource, t: int)
deallocate (r: Resource)
renew (r: Resource, t: int)
expired (r: Resource): Bool

exercise:
designing syncs

identify two or more Autodesk concepts to synchronize
could start from the one you picked last time
split it up or combine it with another one
or focus on some function that you sense involves synchronization

write a concept outline for each concept
name, purpose, OP, action names and args

develop some sync proposals
consider which actions might be linked
you might need to tweak the concepts

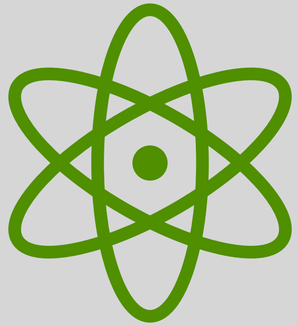
a possible area of focus
how are benchmarks updated when models change?
or when new design options are created?
what happens when benchmarks get updated?

takeaways

how synchronization helps

customize with app-specific behaviors

separate concerns splitting into reusable concepts



automate behavior splitting into reusable concepts

what's next?

now you understand

what concepts are
how to define them
how to compose them

our next step

finding concepts in a larger context
how do you disentangle a complex app?