Building Least Privileged Web Applications with Node.js

Deian Stefan

Joint work with Devon Rifkin, Annie Liu, Christian Garcia Almenar

STANFORD UNIVERSITY

GitStar

Oh ye web apps...

Man Finds Easy Hack to Delete Any Facebook Photo Album

U.S. Postal Service data breach may compromise staff, customer details

Massive Data Breach

University of Chicago data breach employee and student data

United website breach let fliers see each others' private data

R7-2015-02: Google Play Store X-Frame- Gaps Enable Android Remote Code Exec

age Dental reports data breach

Target Confirms Unauthorized Acces Payment Card Data in U.S. Stores

Snapchat security breach affects 4.6 mil ers. Did Snapchat drag its feet on a fix?

Over 99 percent of About.com li vulnerable to XSS, XFS iframe a

WellPoint email glitch puts colonoscopy test in the subject line

50,000 Uber driver names, licens xposed in a data breach

discovered last year, database was accessed in May.

ahoo Password Breach Puts SQL Injection In the Crosshairs

curity flaw in New South Wales puts thousands online votes at risk

MARCH 22, 2015 BY VANESSA TEAGUE AND J. ALEX HALDERMAN LEAVE A COMMENT

Adobe customer data breached - login and credit card data probably stolen, all passwords reset

Credit Card Breach Los Angeles Inte Police Investigating At LAX Tom Br

Credit agency mistakenly sends 300 confidential reports to Maine woman

Recipe for disaster

1. Apps handle sensitive user data

2. Programming models follow the principle of most privilege
   ➠ ad-hoc security mechanisms

```
if ((err = SSLHashSHA1.upd
    goto fail;
if ((err = SSLHashSHA1.upd
    goto fail;
    goto fail;
if ((err = SSLHashSHA1.fin
    goto fail;
```

3. Developers write buggy code

Example: ghost.org

- **Production blog app's data model:**

Blog posts:

id	authors	pub?	title	body
0	alice	FALSE	...	...
1	bob, claire	TRUE	...	...

Users:

user	password	email	name
alice	0dea48ff	...	A. Alyokhina
bob	15a8ccd8f	...	B. Digital
claire	v3991e5	...	C. Hopper

  - Sensitive data: unpublished posts, passwords, emails

- **App functionality:**
  - List all posts, show post, show user profile, ...

Example: ghost.org

- Production blog app's data model:

Blog posts:

id	authors	pub?	title	body
0	alice	FALSE	...	...
1	bob, claire	TRUE	...	...

Users:

user	password	email	name
alice	0dea48ff	...	A. Alyokhina
bob	15a8ccd8f	...	B. Digital
claire	v3991e5	...	C. Hopper

  - Sensitive data: unpublished posts, passwords, emails

- App functionality:
  - List all posts, show post, show user profile, ...

Example: ghost.org

- **Production blog app's data model:**

Blog posts:

id	authors	pub?	title	body
0	alice	FALSE	...	...
1	bob, claire	TRUE	...	...

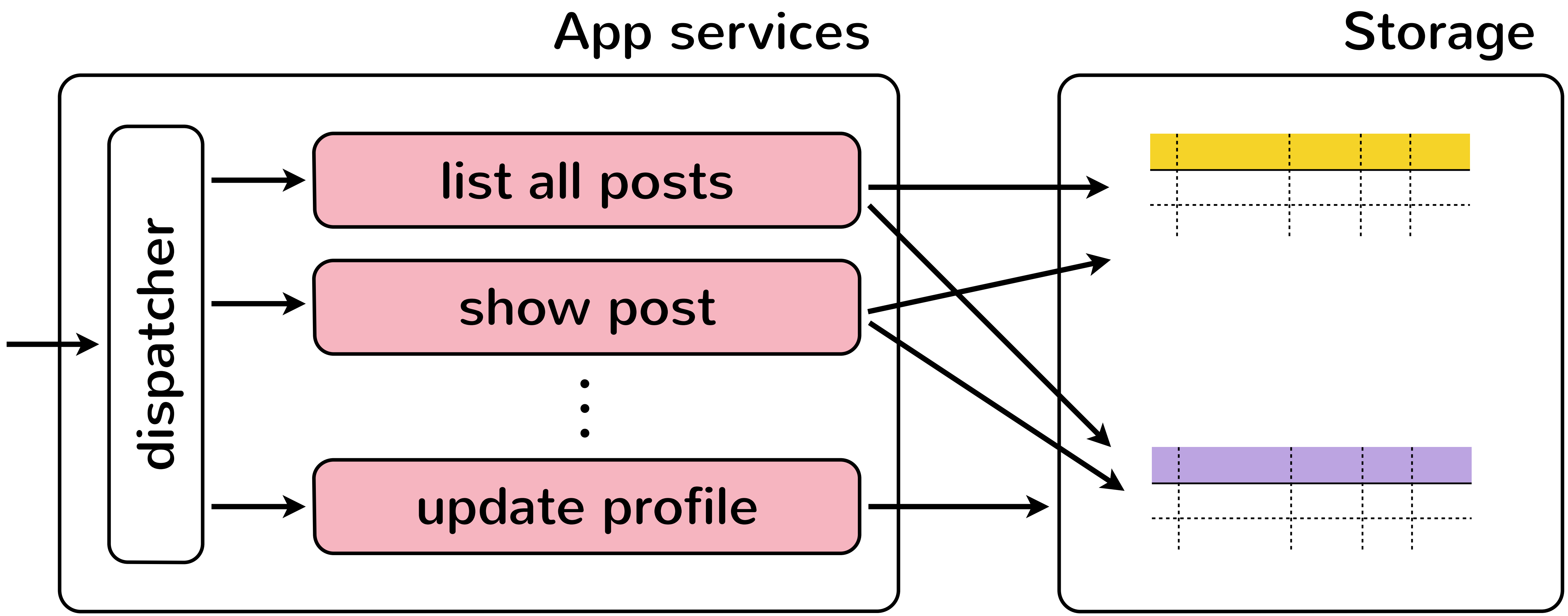
Users:

user	password	email	name
alice	0dea48ff	...	A. Alyokhina
bob	15a8ccd8f	...	B. Digital
claire	v3991e5	...	C. Hopper

  - Sensitive data: unpublished posts, passwords, emails

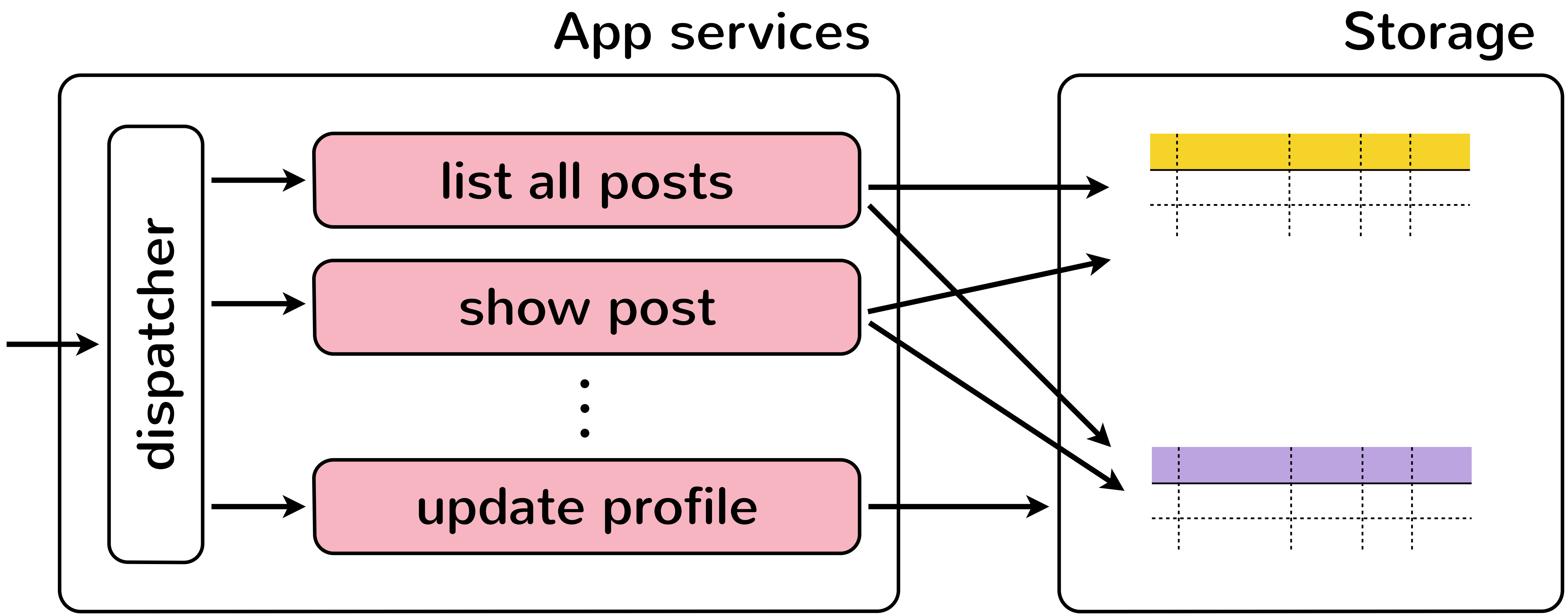
- **App functionality:**
  - List all posts, show post, show user profile, ...

App architecture



- **By default, handlers are most privileged**
  - ➤ Unrestricted access to storage, fs, net, child process, …
- **Developers retrofit security on top**

App architecture

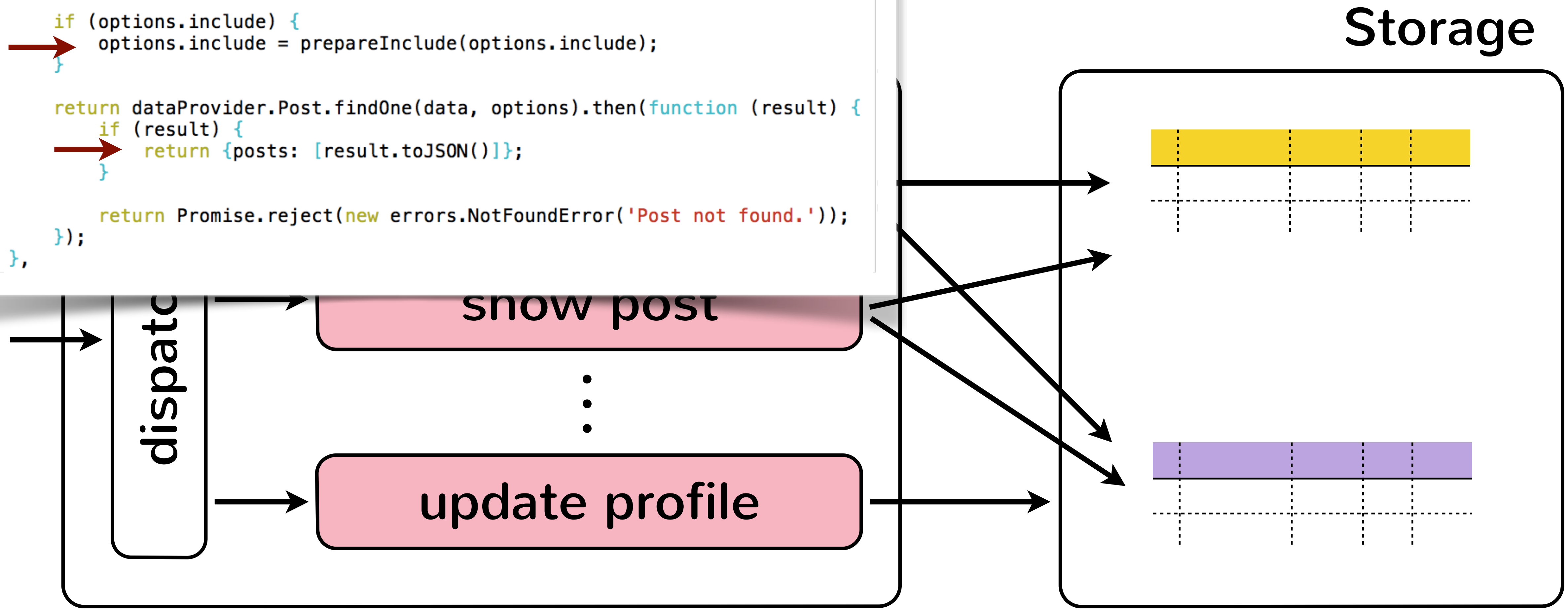


- **By default, handlers are most privileged**
  - Unrestricted access to storage, fs, net, child process, …
- **Developers retrofit security on top**

tecture

Show post

```
read: function read(options) {
    var attrs = ['id', 'slug', 'status'],
        data = _.pick(options, attrs);
    options = _.omit(options, attrs);

    // only published posts if no user is present
    if (!(options.context && options.context.user)) {
        data.status = 'published';
    }

    if (options.include) {
        options.include = prepareInclude(options.include);
    }

    return dataProvider.Post.findOne(data, options).then(function (result) {
        if (result) {
            return {posts: [result.toJSON()]};
        }

        return Promise.reject(new errors.NotFoundError('Post not found.'));
    });
},
```

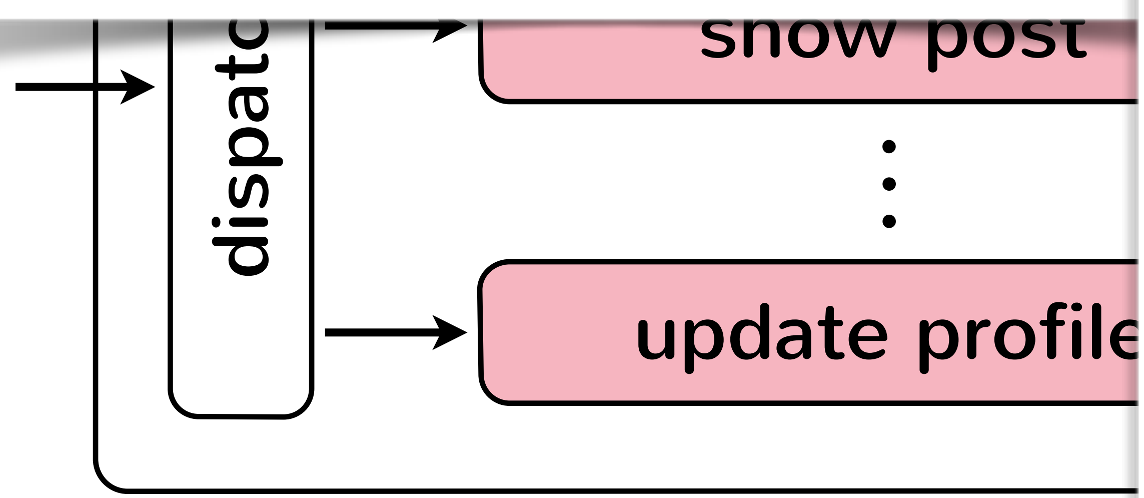


- **By default, handlers are most privileged**
  - Unrestricted access to storage, fs, net, child process, …
- **Developers retrofit security on top**

Show post

```
read: function read(options) {
    var attrs = ['id', 'slug', 'status'],
        data = _.pick(options, attrs);
    options = _.omit(options, attrs);

    // only published posts if no user is present
    if (!(options.context && options.context.user)) {
        data.status = 'published';
    }

    if (options.include) {
        options.include = prepareInclude(options.include);
    }

    return dataProvider.Post.findOne(data, options).then(function
        if (result) {
            return {posts: [result.toJSON()]};
        }

        return Promise.reject(new errors.NotFoundError('Post not f
    });
},
```

...tecture



- By default, handlers a
  - Unrestricted access to
- Developers retrofit se

Update profile

```
edit: function edit(object, options) {
    var editOperation;
    if (options.id === 'me' && options.context && options.context.user) {
        options.id = options.context.user;
    }

    if (options.include) {
        options.include = prepareInclude(options.include);
    }

    return utils.checkObject(object, docName).then(function (data) {
        // Edit operation
        editOperation = function () {
            return dataProvider.User.edit(data.users[0], options)
                .then(function (result) {
                    if (result) {
                        return {users: [result.toJSON()]};
                    }
                    return Promise.reject(new errors.NotFoundError('User not found.'));
                });
        };

        // Check permissions
        return canThis(options.context).edit.user(options.id).then(function () {
            // if roles aren't in the payload, proceed with the edit
            if (!(data.users[0].roles && data.users[0].roles[0])) {
                return editOperation();
            }
            var role = data.users[0].roles[0],
                roleId = parseInt(role.id || role, 10),
                editedUserId = parseInt(options.id, 10);

            return dataProvider.User.findOne(
                {id: options.context.user, status: 'all'}, {include: ['roles']}
            ).then(function (contextUser) {
                var contextRoleId = contextUser.related('roles').toJSON()[0].id;

                if (roleId !== contextRoleId && editedUserId === contextUser.id) {
                    return Promise.reject(new errors.NoPermissionError('You cannot change your own role.'));
                }

                return dataProvider.User.findOne({role: 'Owner'}).then(function (owner) {
                    if (contextUser.id !== owner.id) {
                        if (editedUserId === owner.id) {
                            if (owner.related('roles').at(0).id !== roleId) {
                                return Promise.reject(new errors.NoPermissionError('Cannot change Owner\'s role.'));
                            }
                        } else if (roleId !== contextRoleId) {
                            return canThis(options.context).assign.role(role).then(function () {
                                return editOperation();
                            });
                        }
                    }
                    return editOperation();
                });
            });
        });
    }).catch(function (error) {
        return errors.handleAPIError(error, 'You do not have permission to edit this user');
    });
},
```

Problem with existing approach

- **Missing single security check ➠ vulnerability**
  - ➢ E.g., ghost.org exposed passwords and drafts
- **Checks don’t always extend to third-party libs**
  - ➢ Libraries may expose vulnerabilities
- **Damage due to vulnerabilities can be grave**
  - ➢ All code runs with same privilege
  - ➢ E.g., st library didn’t handle “..” correctly ➠ leaked files

Problem with existing approach

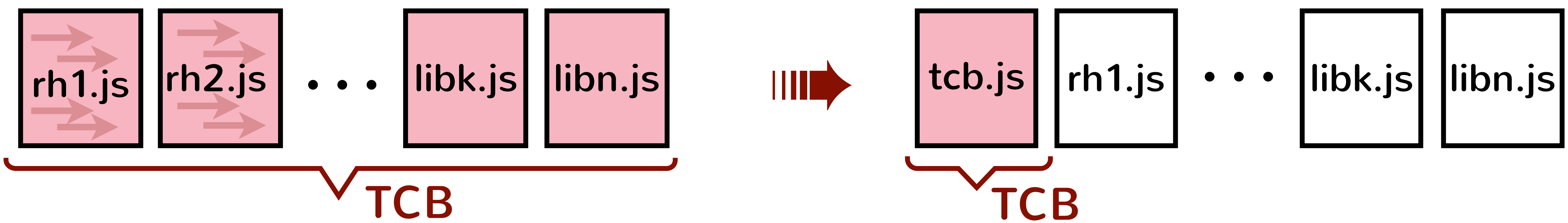
- **Missing single security check ➠ vulnerability**
  - E.g., ghost.org exposed passwords and drafts
- **Checks don't always extend to third-party libs**
  - Libraries may expose vulnerabilities
- **Damage due to vulnerabilities can be grave**
  - All code runs with same privilege
  - E.g., st library didn't handle ".." correctly ➠ leaked files

Problem with existing approach

- **Missing single security check ➠ vulnerability**
  - E.g., ghost.org exposed passwords and drafts
- **Checks don't always extend to third-party libs**
  - Libraries may expose vulnerabilities
- **Damage due to vulnerabilities can be grave**
  - All code runs with same privilege
  - E.g., st library didn't handle ".." correctly ➠ leaked files

Change how developers build apps

- **Minimize trusted computing base (TCB)**



  - ➤ Make security robust to bugs in most code

- **Challenge: perennial goal in computer security**
  - ➤ Can we actually do this?

Can we do this for Node.js?

```
thinly:~ d$ node
> Array(16)
[ , , , , , , , , , , , , , , ,  ]
> Array(16).join("wat")
'watwatwatwatwatwatwatwatwatwatwatwatwatwatwat'
> Array(16).join("wat" + 1)
'wat1wat1wat1wat1wat1wat1wat1wat1wat1wat1wat1wat1wat1wat1wat1'
> Array(16).join("wat" - 1) + " Batman!"
'NaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaN Batman!'
> 
```

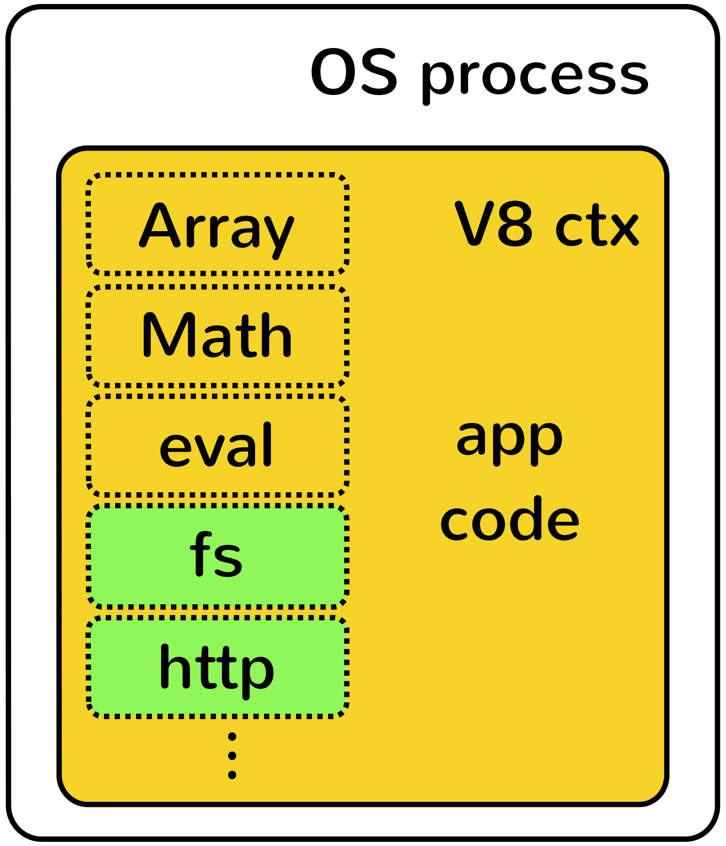
Turns out…

JavaScript is well-suited
for executing untrusted code

…if you just look at it just right

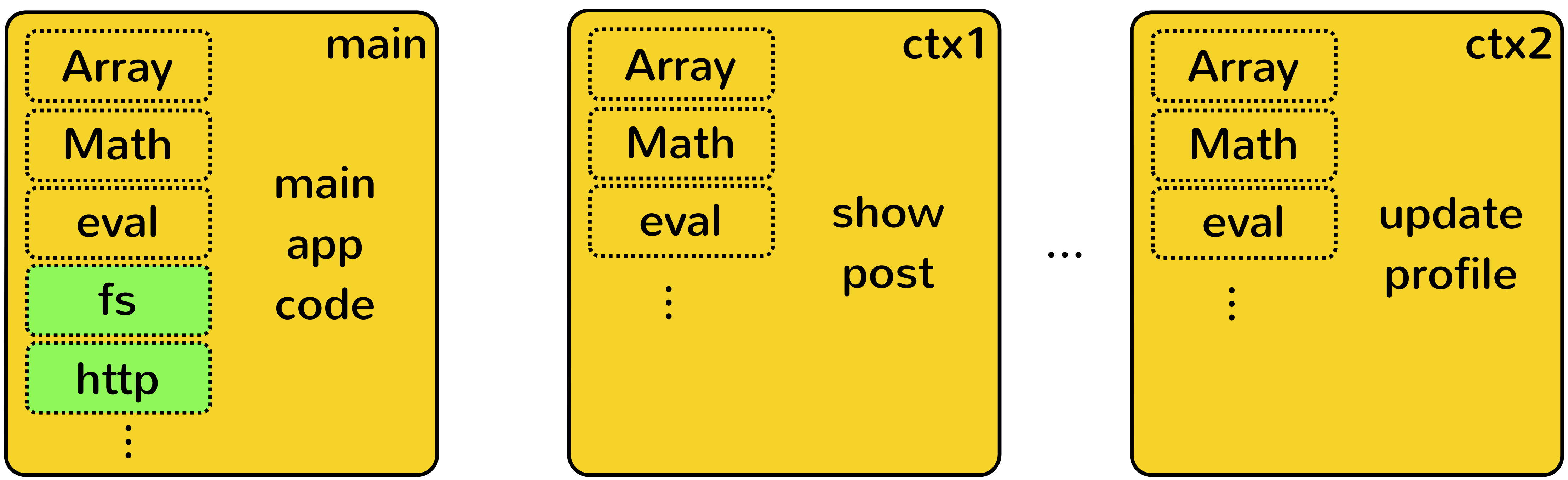
Node.js apps at a high level

- **Code runs in (V8) contexts**
  - Global object + execution stack
- **Language (EcmaScript) doesn't have built-in IO**
- **Embedder (Node.js) attaches props to global object to provide IO**
  - E.g., fs, http, net, process, etc.



Looking at it just right

- **Expose V8 contexts as isolation primitives**
  - New context has separate heap: no access to fs, etc.
- **Execute untrusted code in new contexts**
  - E.g., run different request handlers in isolation



Providing useful APIs to ctxs

- **By default, code has minimal privileges**
  - Can't do anything except execute "pure" JavaScript
- **Problem: real code needs to perform IO**
  - Fresh contexts do not have access to Node.js APIs
- **Solution: expose message passing primitives**
  - Untrusted context can send and receive messages to and from main/parent context
  - To perform IO: ask parent context to do it

Providing useful APIs to ctxs

- **By default, code has minimal privileges**
  - Can’t do anything except execute “pure” JavaScript
- **Problem: real code needs to perform IO**
  - Fresh contexts do not have access to Node.js APIs
- **Solution: expose message passing primitives**
  - Untrusted context can send and receive messages to and from main/parent context
  - To perform IO: ask parent context to do it

Virtualization w/ message passing

- **Function calls ➠ messages to parent context**
  - Parent can perform checks before (and after) calling actual function
  - E.g., implementing synchronous file read:

ctx1.js

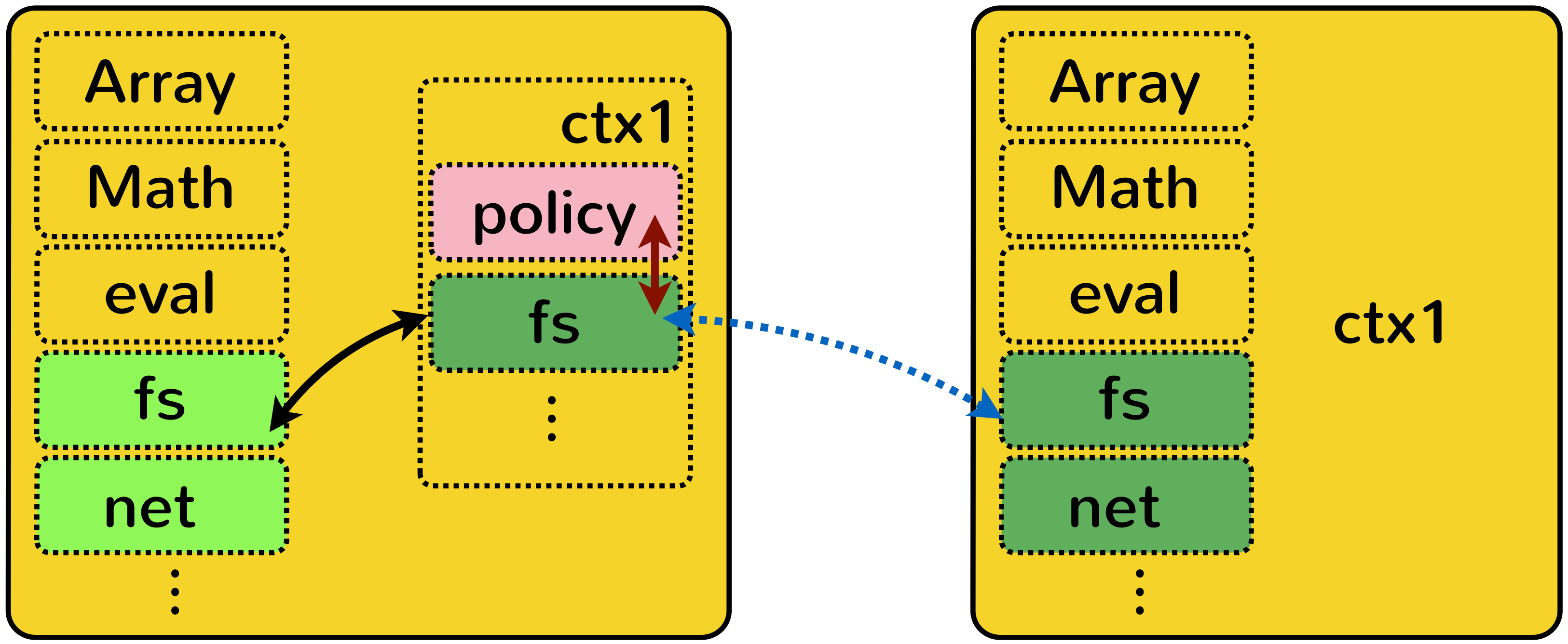
```
fs.readFileSync = function (fname, opts) {
 return _espectro.RPC('fs:readFileSync')(fname, opts);
}
// ...
```

main.js

```
var ctx1 = new Ctx('ctx1.js');
ctx1.onrpc('fs:readFileSync', function (fname, opts) {
  if (!(fname in _allowed)) throw 'denied!';
  var res = fs.readFileSync(fname, opts)
  return res;
});
```

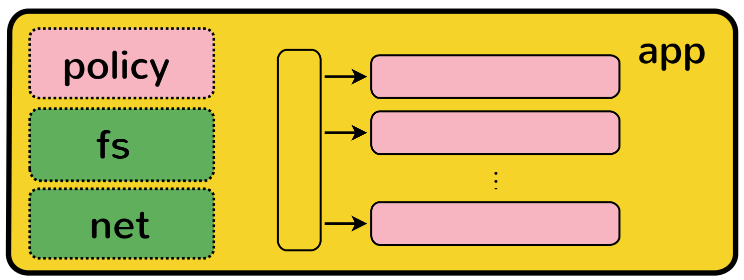
Virtualized Node.js libraries

- **In untrusted contexts: core libraries using message passing**
- **In main context: hooks library used to register pre/post hooks for each function call**
  - ➤ High-level policies implemented atop hooks

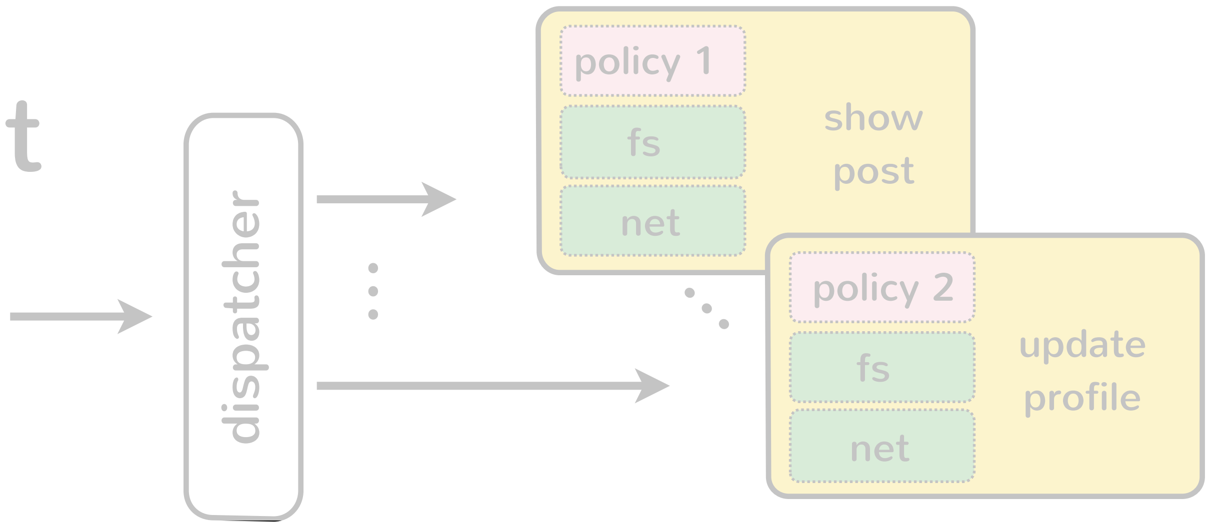


Different architectures

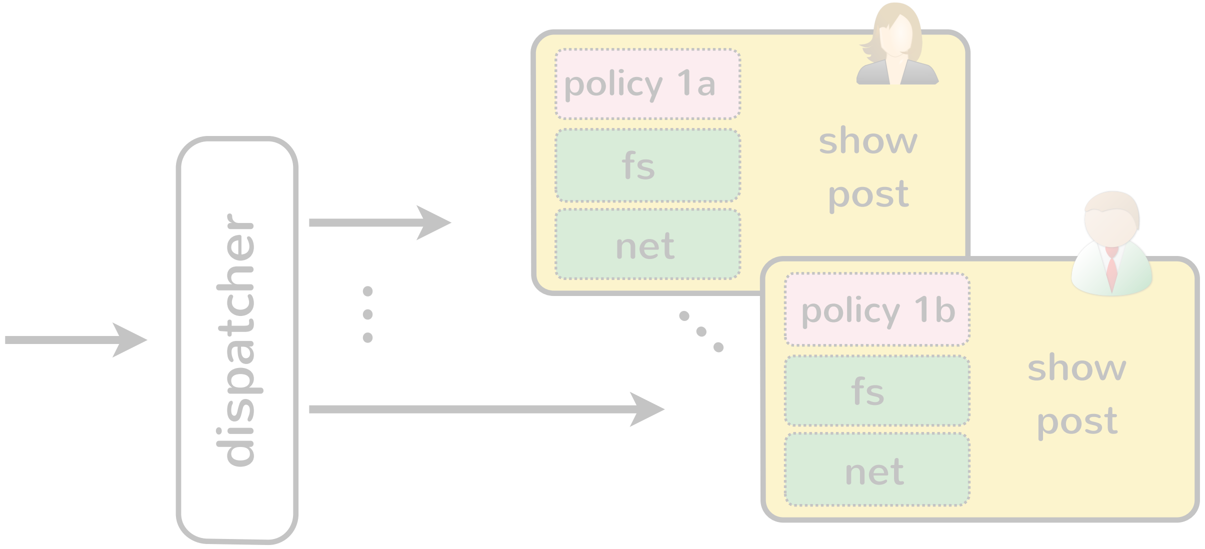
- App / context



- Controller / context

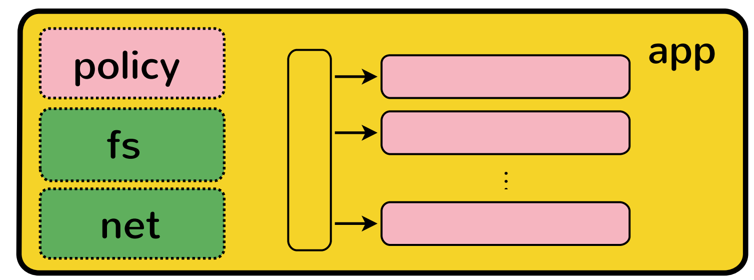


- Request / context

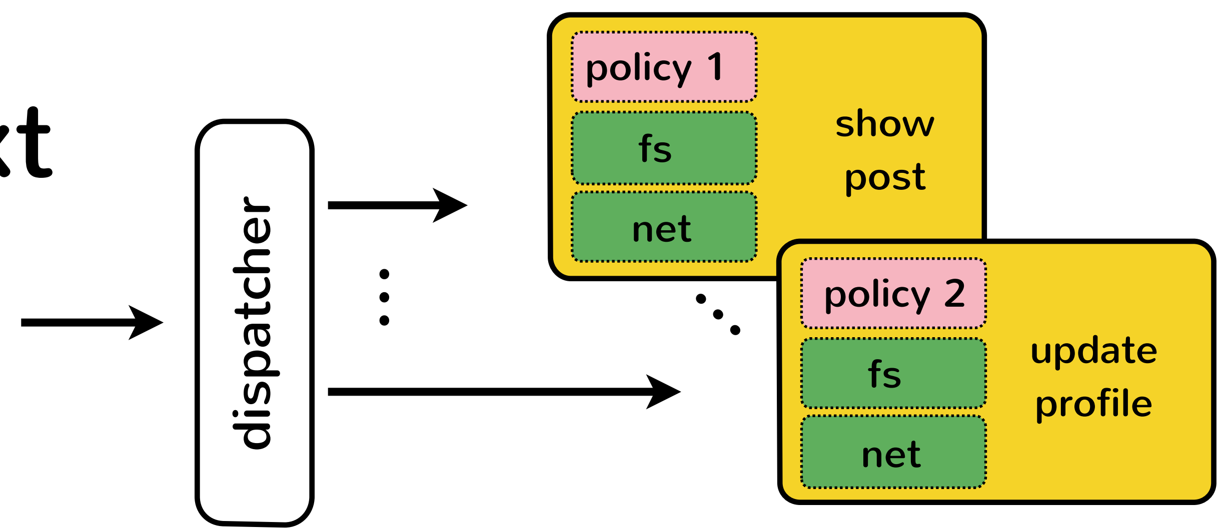


Different architectures

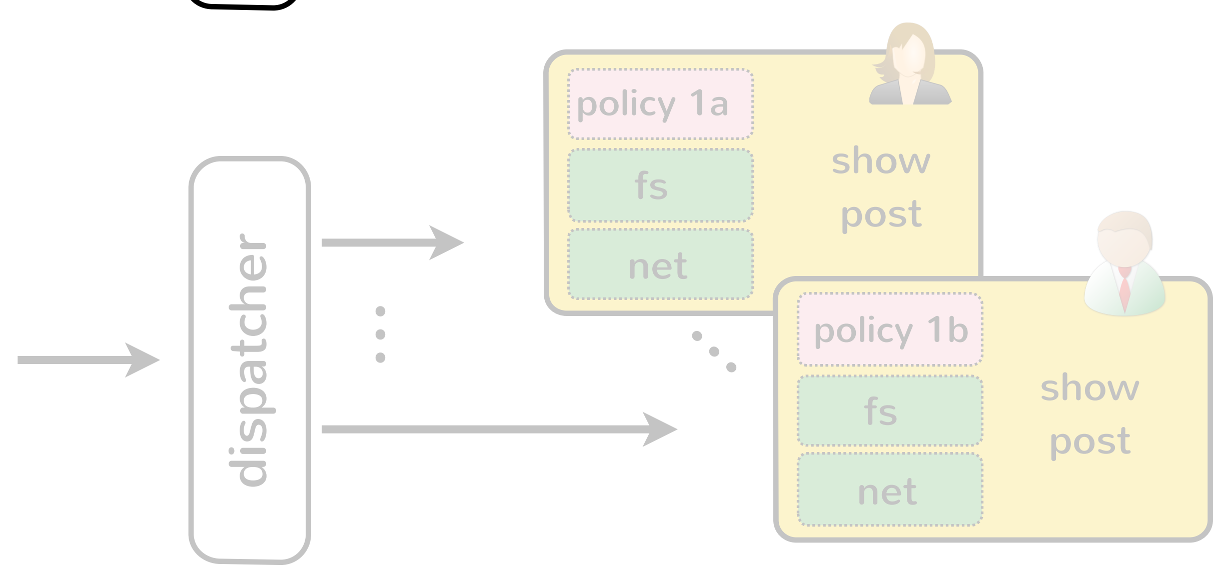
- App / context



- Controller / context

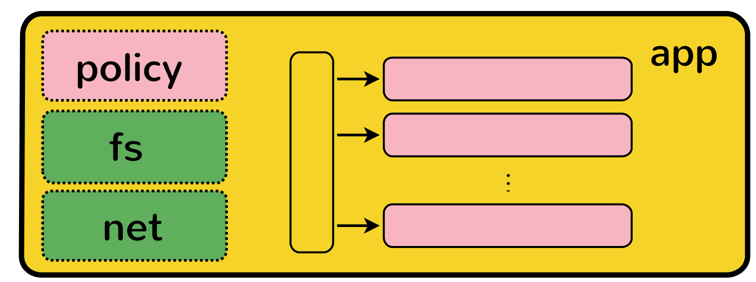


- Request / context

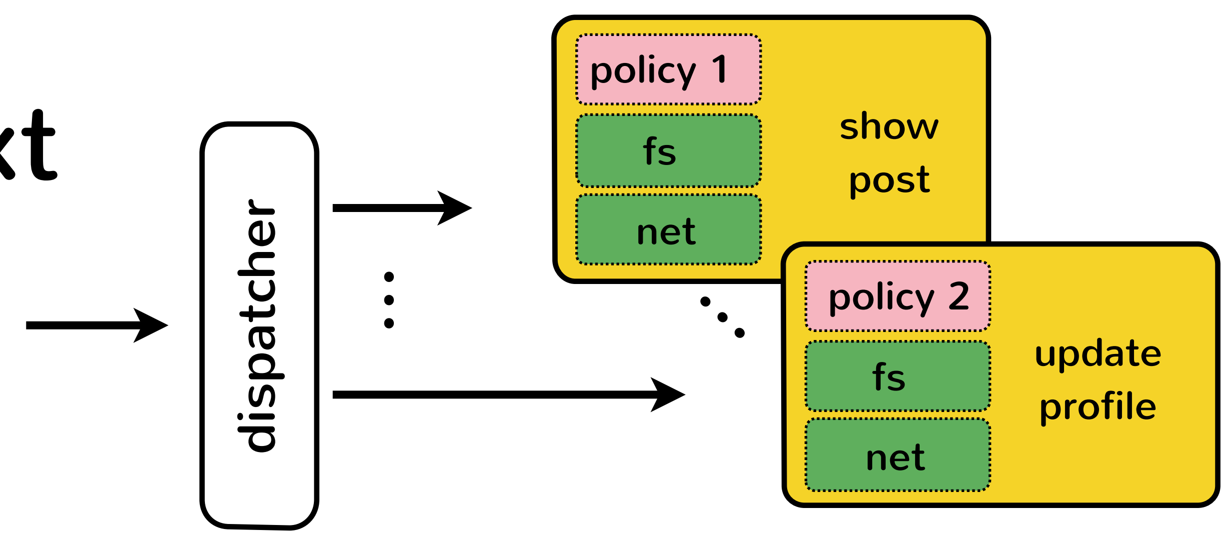


Different architectures

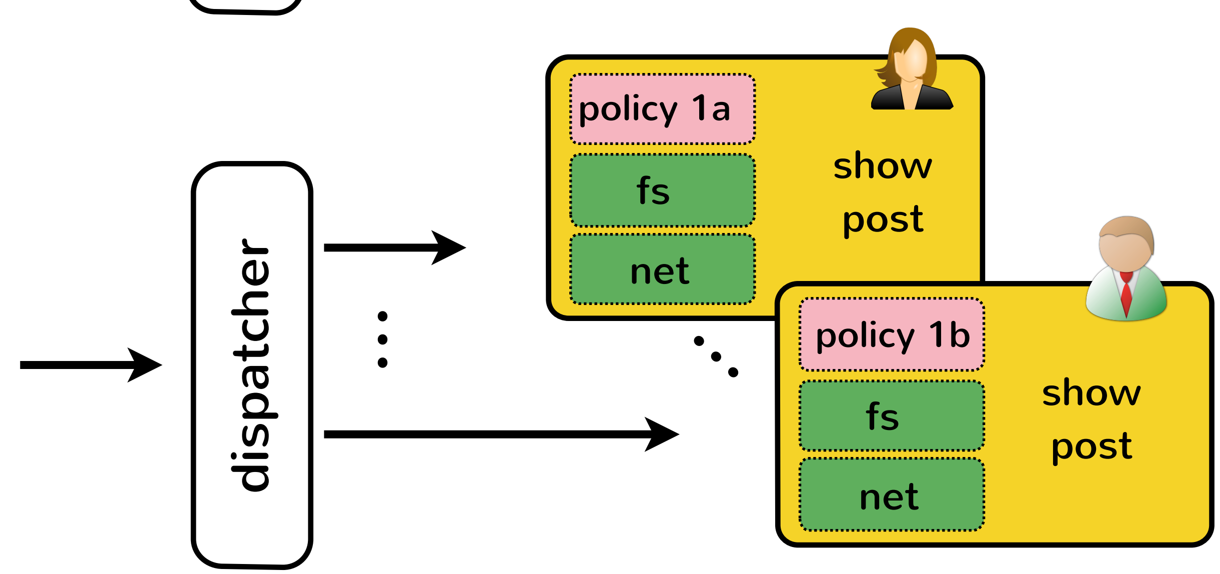
- App / context



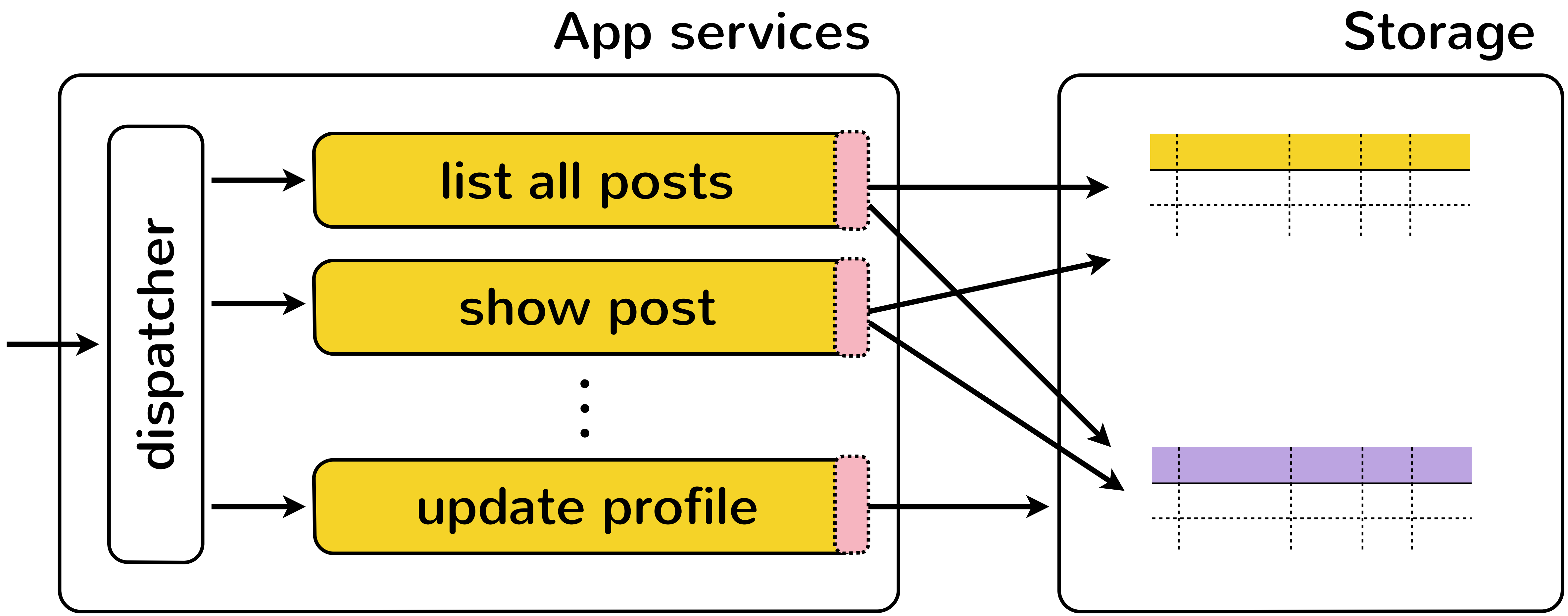
- Controller / context



- Request / context

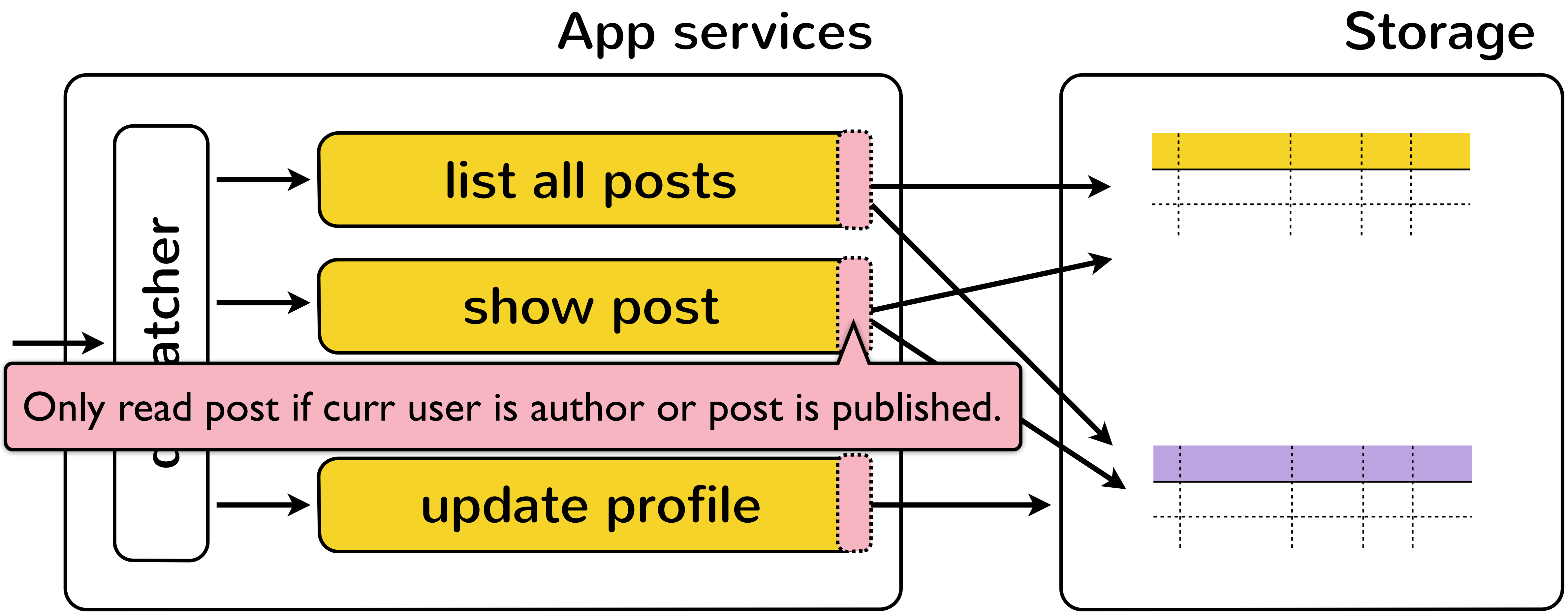


Least privileged ghost.org



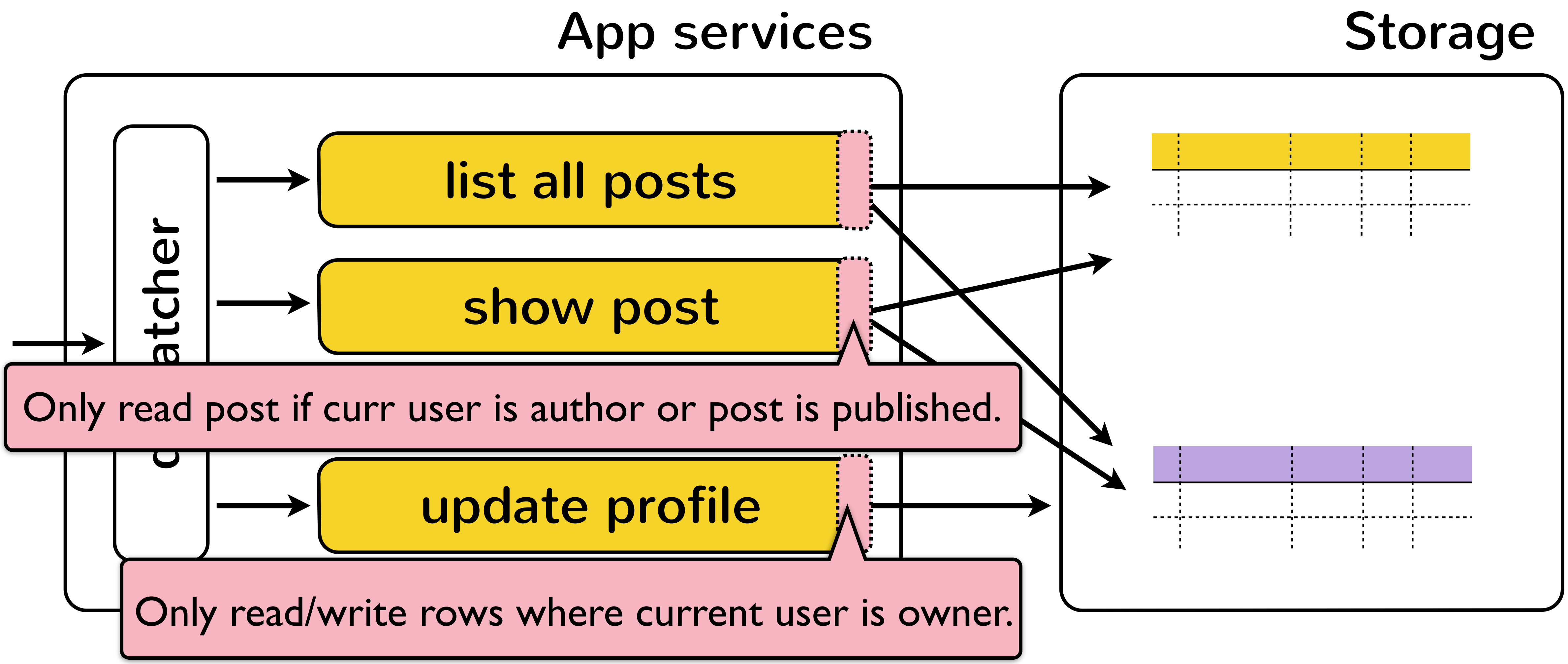
- **By default, handlers are least privileged**
  - Restricted access to storage, fs, net, etc. according to current user and app functionality

Least privileged ghost.org



- **By default, handlers are least privileged**
  - Restricted access to storage, fs, net, etc. according to current user and app functionality

Least privileged ghost.org



- By default, handlers are least privileged
  - Restricted access to storage, fs, net, etc. according to current user and app functionality

Consequence of design

- **Policy can be declarative specified in main context**
- **Policy extends to third-party libraries**
  - Policy applies to request handler and any library it uses
- **DAC policies can limit damage due to bugs**
  - Fine-grained/user request limits attack surface
- **MAC policies can prevent damage due to bugs**
  - MAC enforces policy even once code has acces to data

Beyond access control

- **Virtualization layer can be used for:**
  - Transparently encrypting/decrypting files
  - Caching files, DB queries, responses, etc.
  - Rewriting HTML to add CSRF tokens
  - MACing cookies
  - Setting custom headers (e.g., CSP, SRI, etc.)
  - …

Conclusions

- **Today: writing insecure code is the default**
  - Building least-privileged apps is notoriously difficult
- **App-level virtualization can be used to protect app from itself and third-party code**
  - Policy must allow functionality for it to be available
  - Can build least privileged apps more easily

Thanks!

Availability: this summer from gitstar.com

Follow up: @deiandelmars