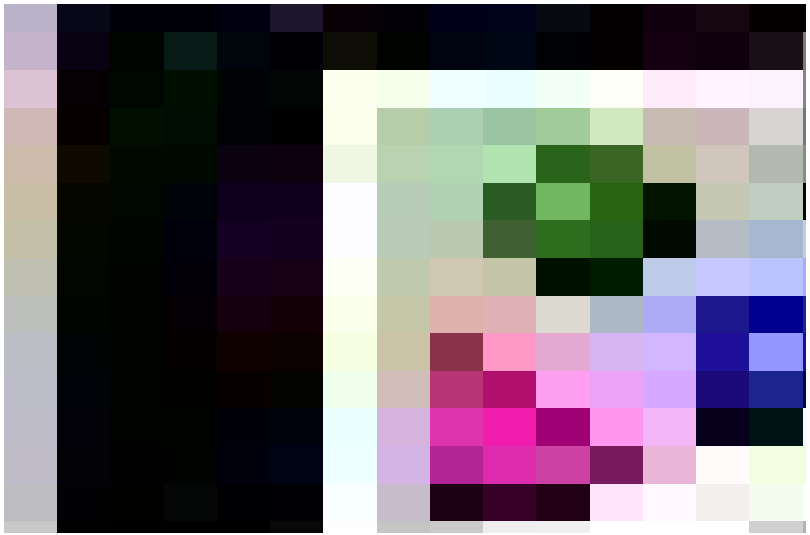
Building hypermedia clients —adaptive interfaces

Todd Brackley @toddb*NZ*
Hypr @hypr*NZ*

Ssshhhh!
Quiet crisis



- Great at innovation
- But, struggle with execution
- Often complexity has taken hold
- So, profitability not realised
- Need, transitions to meet demand
- This talk is about one of those transitions

Business issues & tech goals: make a survey

- Provision an entire network of data (minutes—not days)
- Really needed to expose real data (to show what's going on)
- Model the business processes but defer GUI (in-place editing)

Adapative Interfaces

- Underlying engine should be the same regardless of presentation
- Changes to the server can extend the client (use forms as affordances)

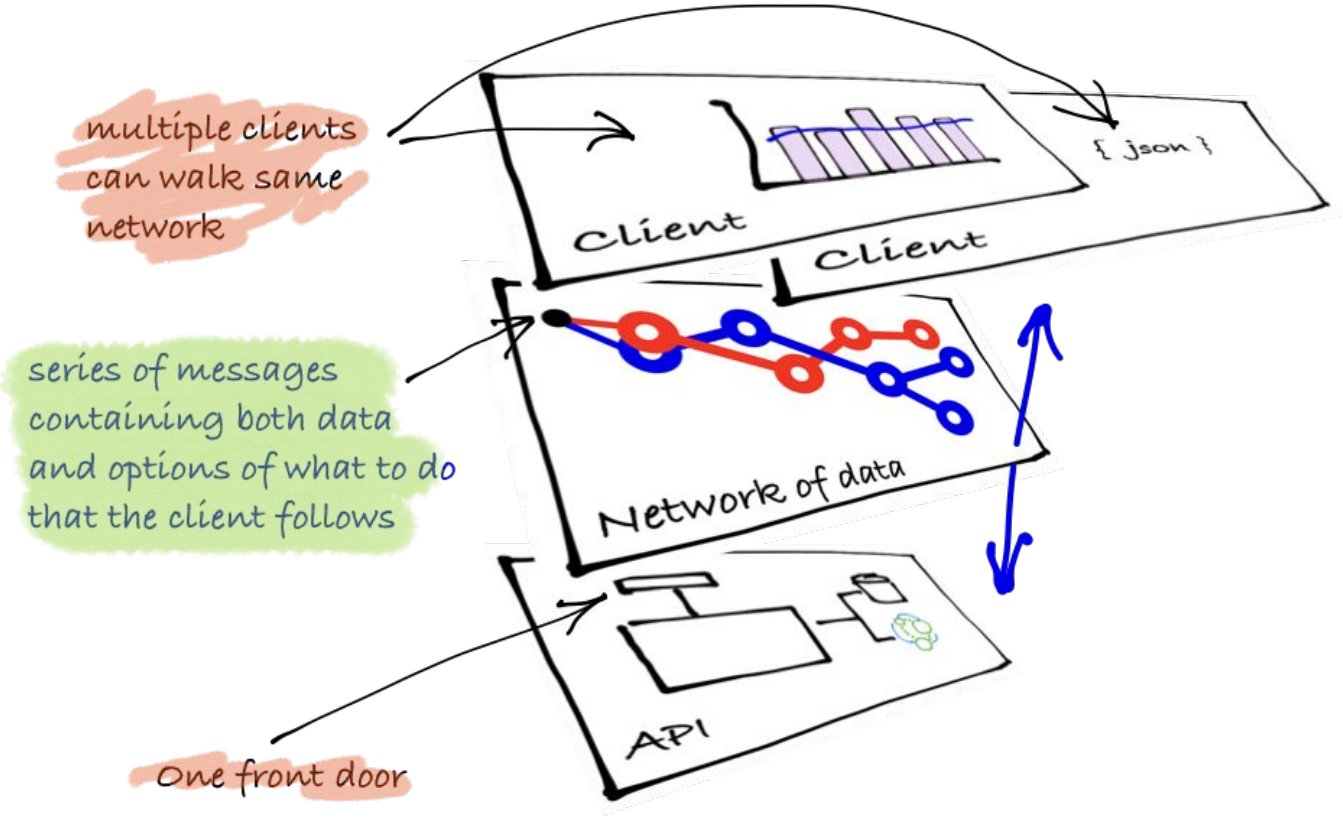
This talk: clients ...

1. purpose is to walk and update (basics)
2. use forms to know what to process (affordances)
3. should only be as complex as your need

Walking the network of data

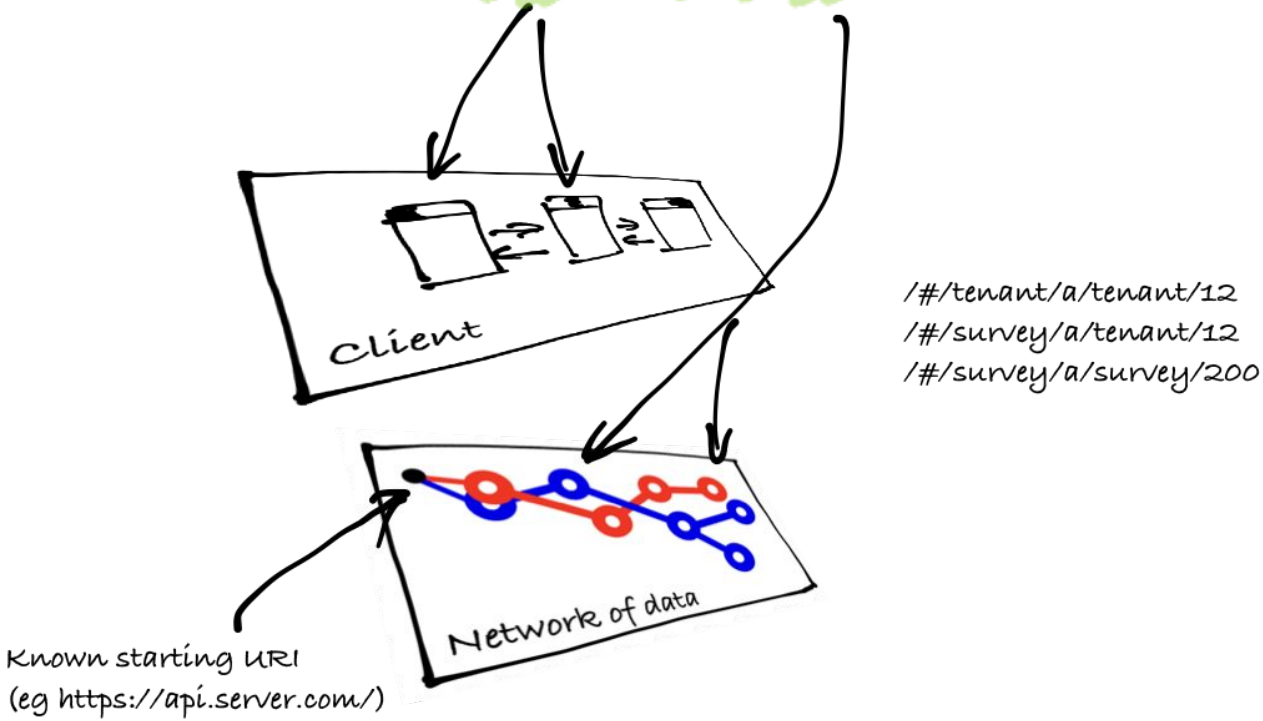
- Hypermedia interface guides us through a business process using data and options
- client follow links to resources and decides what to do based on what the server offers

Keep a separation between client and API



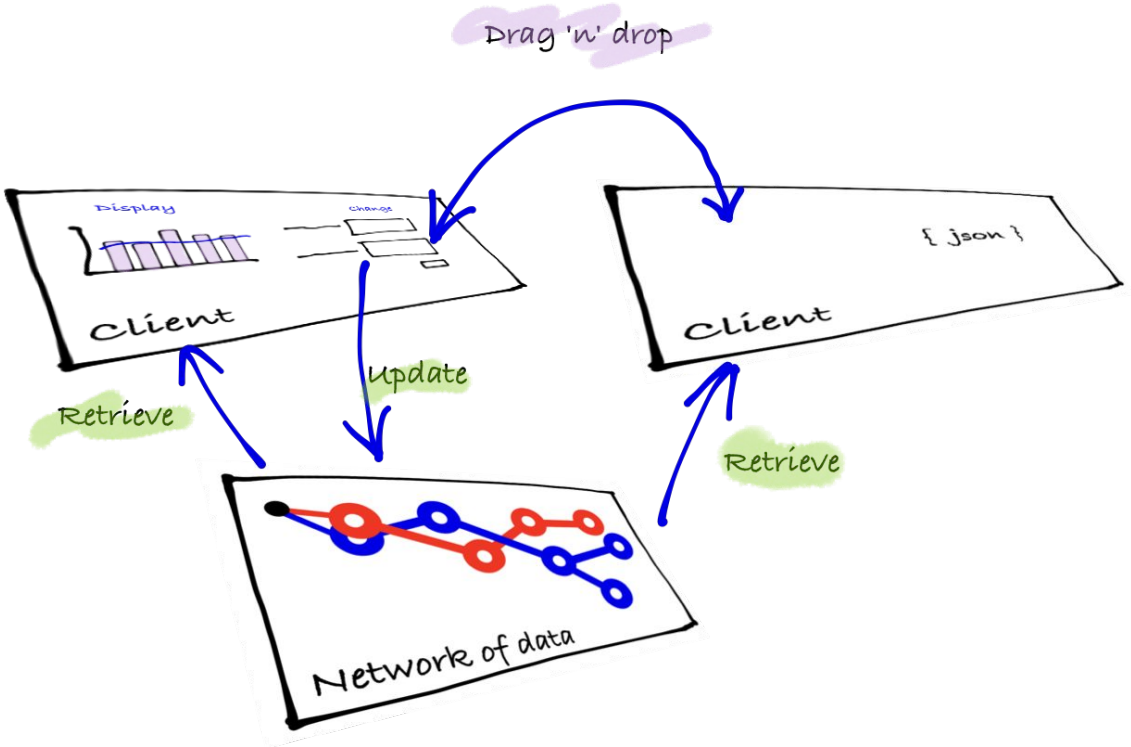
Bookmarkable URI holding state (browser client)

https://example.com/#/[client]/a/[api resource]

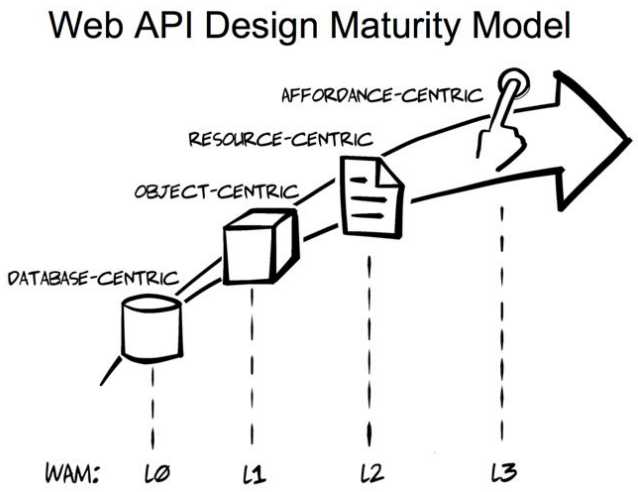


/#/tenant/a/tenant/12
/#/survey/a/tenant/12
/#/survey/a/survey/200

Lots of ways to walk the API even combining clients



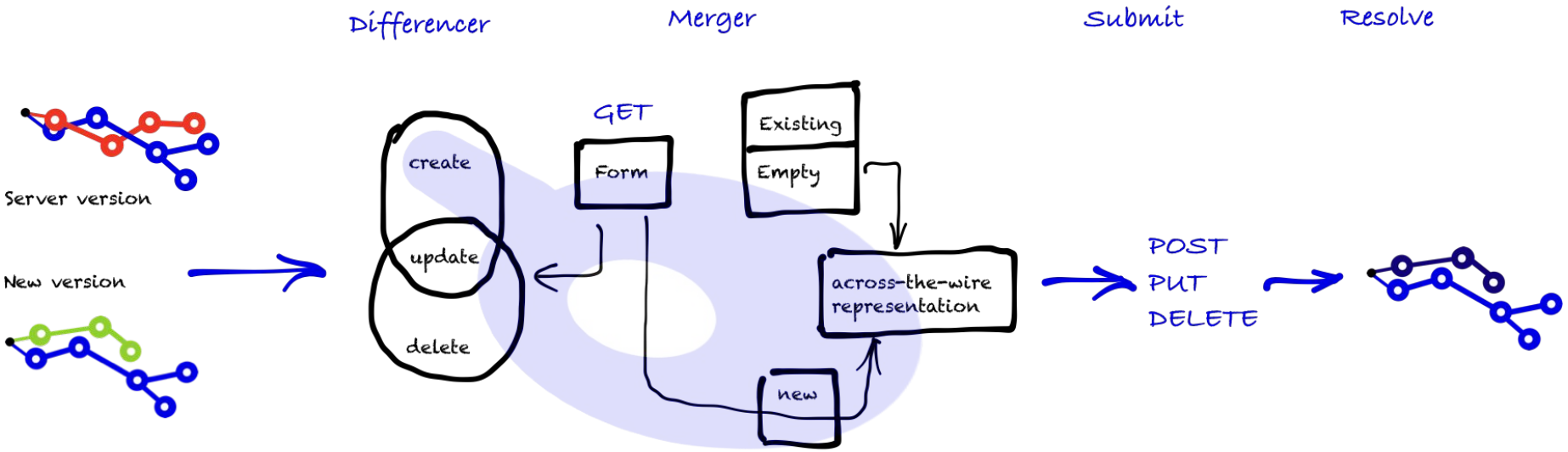
Using forms as affordances



- updates to resources and forms are controlled at the server
- no changes needed on the client except for changes to the network structure
- forms tell the client how to process properties on in-memory resources

Amundsen, http://amundsen.com/talks/2016-06-wsrest/2016-06-WADM.pdf

General engine



Forms: you'll need to cater for (just like html)

- Single (by value) [text/numbers/passwords/creditcard)
- Enumeration (select by value or by reference) [single/multiple]
- Groups (containers of above–including recursive)

There are plenty of forms specifications

- HAL-FORMS
- Cj
- SIREN
- JSON-LD + hydra
- UBER
- We are using atom-like+json

Here's our (create) form—single by value

```
links: [
  {
    rel: "self",
    href: https://api-cem-qa.cemplicity.com/tena
  },
  {
    rel: "up",
    href: https://api-cem-qa.cemplicity.com/
  },
  {
    rel: "search",
    href: https://api-cem-qa.cemplicity.com/tena
  },
  {
    rel: "create-form",
    href: https://api-cem-qa.cemplicity.com/tena
  }
],
```

```
links: [
  {
    rel: "self",
    href: https://api-cem-qa.cemplicity.com/tenant/form/create
  },
  {
    rel: "up",
    href: https://api-cem-qa.cemplicity.com/tenant/
  },
  {
    rel: "submit",
    href: https://api-cem-qa.cemplicity.com/tenant/
  }
],
items: [
  {
    type: http://types/text,
    name: "name",
    description: "The name of the tenant"
  },
  {
    type: http://types/text,
    name: "code",
    description: "The short code used to describe the tenant"
```

… more by value


```
{
    type: http://types/text/password,
    name: "password",
    description: "A required password"
},
{
    type: http://types/text,
    name: "importPath",
    description: "A required import path on the SSH server"
},
{
    type: http://types/text,
    name: "exportPath",
    description: "A required export path on the SSH server"
},
{
    type: http://types/text,
    name: "importFilenamePattern",
    description: "A regular expression that describes the pattern of matching filenames"
},
{
    type: http://types/text,
    multiple: true,
    name: "deliveryConfirmationEmail",
    description: "A comma separated list of email addresses that are sent a simple delivery confirmation report"
},
{
    type: http://types/text,
    multiple: true,
    name: "operationsDeliveryConfirmationEmail",
    description: "A comma separated list of email addresses that are sent a detailed delivery confirmation report"
}
```

… enumeration by value


```
▾ {
    type: http://types/select,
    name: "type",
  ▾ items: [
    ▾ {
          value: http://types.cemplicity.com/survey/question/logic/type/extraction/simple,
          label: "Simple extraction"
      },
    ▾ {
          value: http://types.cemplicity.com/survey/question/logic/type/extraction/advanced,
          label: "Advanced extraction"
      },
    ▾ {
          value: http://types.cemplicity.com/survey/question/logic/type/background-variable,
          label: "Background variable"
      },
    ▾ {
          value: http://types.cemplicity.com/survey/question/logic/type/jump,
          label: "Jump"
      },
    ▾ {
          value: http://types.cemplicity.com/survey/question/logic/type/conditional,
          label: "Conditional"
      },
    ▾ {
          value: http://types.cemplicity.com/survey/question/logic/type/simple,
```

… enumeration by reference (spot the problems)

… group

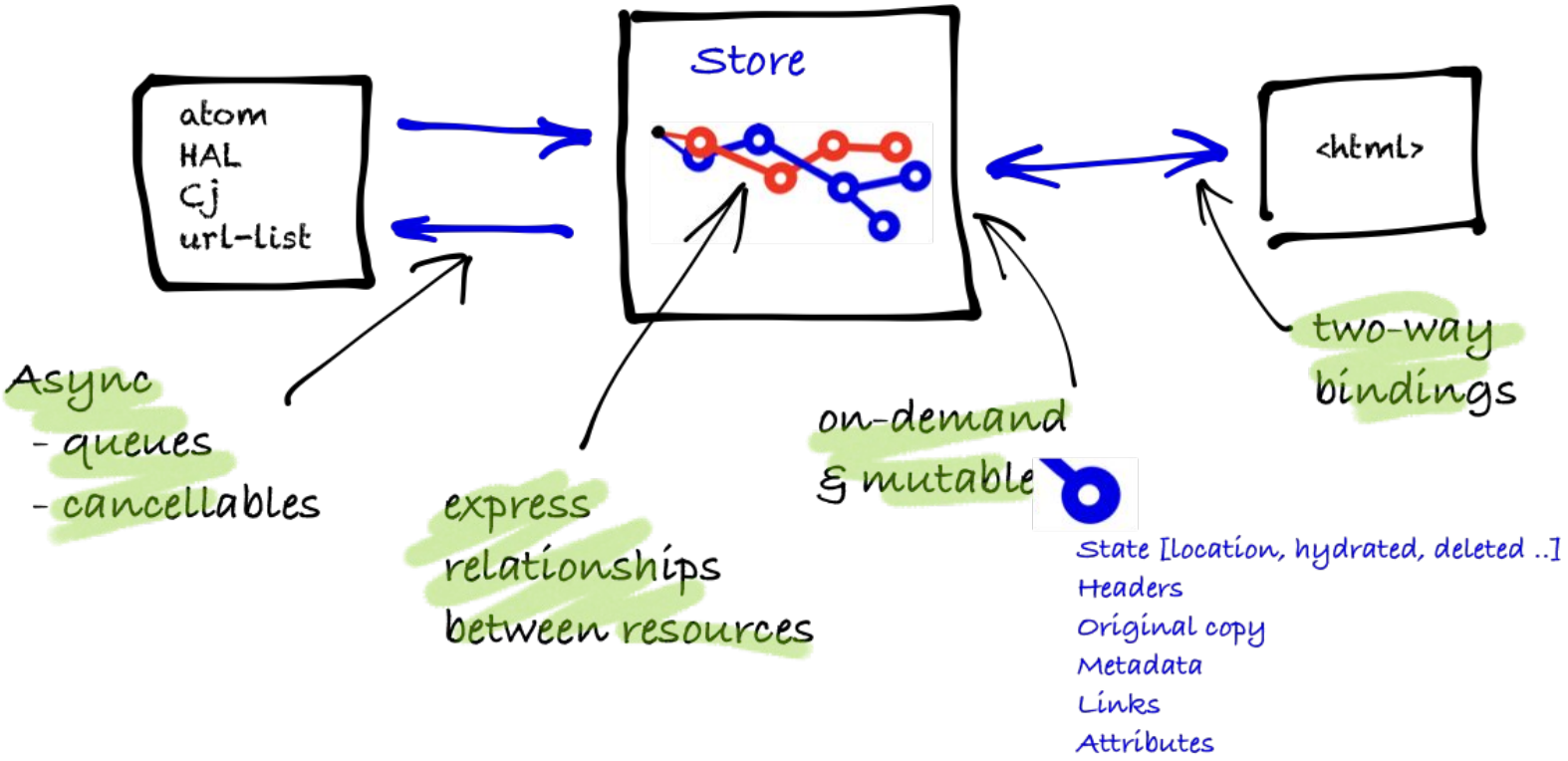
```
{
    type: http://types/group,
    name: "expression",
    items: [
        {
            type: http://types/text,
            name: "type",
            description: "The expression type (not, and, or)"
        },
        {
            type: http://types/group,
            multiple: true,
            name: "items",
            description: "The expressions - this is recursive back to the 'expression' group form"
        },
        {
            type: http://types/select,
            name: "question",
            description: "The expression type (not, and, or)"
        },
        {
            type: http://types/select,
            multiple: true,
            name: "questionItem",
            description: "The question items"
        }
    ],
    description: "The logic rule as an expression (c.f. a '##' style string)"
```

Some design issues

(to be adapative at the right scale)

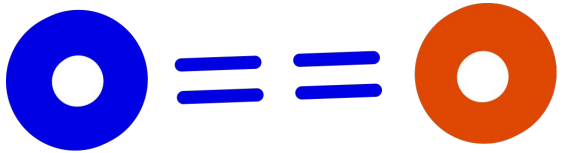
1. Client store
2. Identity
3. Hydration
4. Mappings
5. Caching

1. Single in-memory client-side resource store



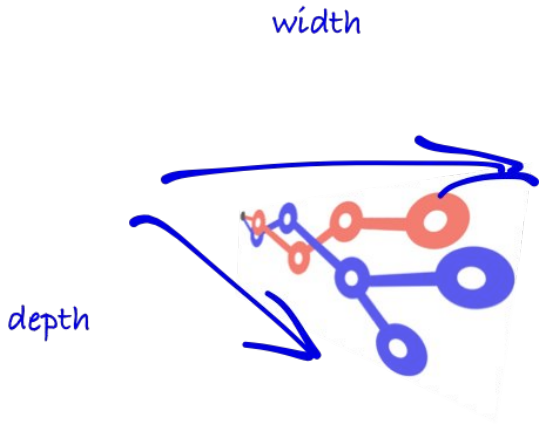
2. Identity of resources

- you need to be able to choose some exactly from link relations (eg 'self') and sometimes more loosely from an attribute (eg 'name') and sometime with a mixture (eg other a combination of link relations)
- need collection utilities to aid and abet this—you'll need to map identity on collection (references) as well as values



3. Hydration strategies: width-first and depth-first

- walking widely vs deeply (eg create a parent collection of items before going deeply into each item)
- we've needed collection-aware async map and reduce utils (in both parallel and sequential versions)



4. Remember mappings for making copies of trees

- taking a copy or part of a known tree or a disconnected tree (and grafting into the graph)
- simple string comparisons (because they are URIs)—implementation is dictionary
- as you are walking the tree, you'll need to be able to do substitutions
- can't have forward references (so we need to know the order to avoid recursive-lazy-loading problems)
- early loading forward references (eg metadata because it will be used by others to create themselves, ancestors before descendants)

5. Pushing through the cache

- avoid thinking about having the best shot at not having stale representations (force loads flags are a smell)
- hold cache headers in the in-memory store and use them to decide—so in practice the server decides not the client and thus the client needs to behave like everyone else and respect the server
- remember the server knows about its domain and resource lifecycles/expiry so use that information
- you are in a layered system that has rules for caching—so play nicely in that ecosystem and it will treat you well (client optimisations will burn you)

Five design issues

We’ve found useful to know about and then incrementally add parts of each as you need them

- Client store
- Identity
- Hydration
- Mappings
- Caching

Conclusion

- Adaptive interfaces are affordance-centric, use forms to instruct the client
- We haven't talked about affordance-centric based workflow as a GUI
- We still aren’t seeing examples in the wild yet
- We aren't getting this for free yet (ie examples and tooling support), so you'll need to think about your app complexity when building out clients
- Stay simple and explainable

Thanks!

Todd Brackley
Hypr

@toddb*NZ*
@hypr*NZ*
hypr.nz