vue class store

universal vue stores you write
once and use anywhere

Recent state-based projects

A little background to how I got here

State Machine

What if FSMs were simple and fun?

```
var fsm = new StateMachine({

    transitions:
    [
        'next    : intro > settings >         complete',
        'back    : intro < settings < error',
        'error   :         settings > error'
    ],

    handlers:
    {
        'start' : ui.start,
        'change': ui.update,
        'settings@next': app.submit,
        'complete': app.complete
    }
});
```

Oct 2016: github.com/davestewart/javascript-state-machine

Vuex Pathify

What if Vuex setup and wiring was simplified?

```
store.get('products@items.0.name')
store.set('products@items.1.name', 'Vuex Pathify')
```

```
computed: {
  ...sync('filters@sort', [
    'order',
    'key'
  ]),

  ...sync('filters@sort', {
    sortOrder: 'order',
    sortKey: 'key'
  }),

  ...sync('filters@sort.*')
}
```

```
make.mutations(state)
```

vuex pathify

ridiculously simple vuex setup + wiring

Jul 2017: github.com/davestewart/vuex-pathify

Axios Actions

What if API calls weren't bound to Vuex?

```
const actions = {
  search: 'products/widgets?category=:category',
  update: 'POST products/widgets/:id/update',
  delete: {
    url: 'products/widgets/:id/delete',
    method: 'delete',
    headers: {
      'Authorization': `Bearer ${token}`
    }
  }
}

const widgets = new ApiGroup(axios, actions)

widgets.search('metal')
widgets.update({id: 1, name: 'Bouncy Widget', category: 'rubber'})
widgets.delete(1)


class VuexResource extends ApiEndpoint {
  constructor (url, mutation) {
    super(axios, url)
    this
      .when('create update delete', () => this.index())
      .when('index', data => store.commit(mutation, data))
      .use('data')
  }
}
```

axios actions

Jul 2018: github.com/davestewart/axios-actions

Vue Class Store

What if all state was simple and flexible?

```
import VueStore from 'vue-class-store'

@VueStore
export class Store {
  public value: number

  public get double (): number {
    return this.value * 2
  }

  constructor (value: number = 1) {
    this.value = value
  }

  'on:value' () {
    console.log('value changed to:', this.value)
  }

  'on:some.other.value' = 'log'

  log () {
    console.log('value is:', this.value)
  }
}
```

vue class
store

May 2020: github.com/davestewart/vue-class-store

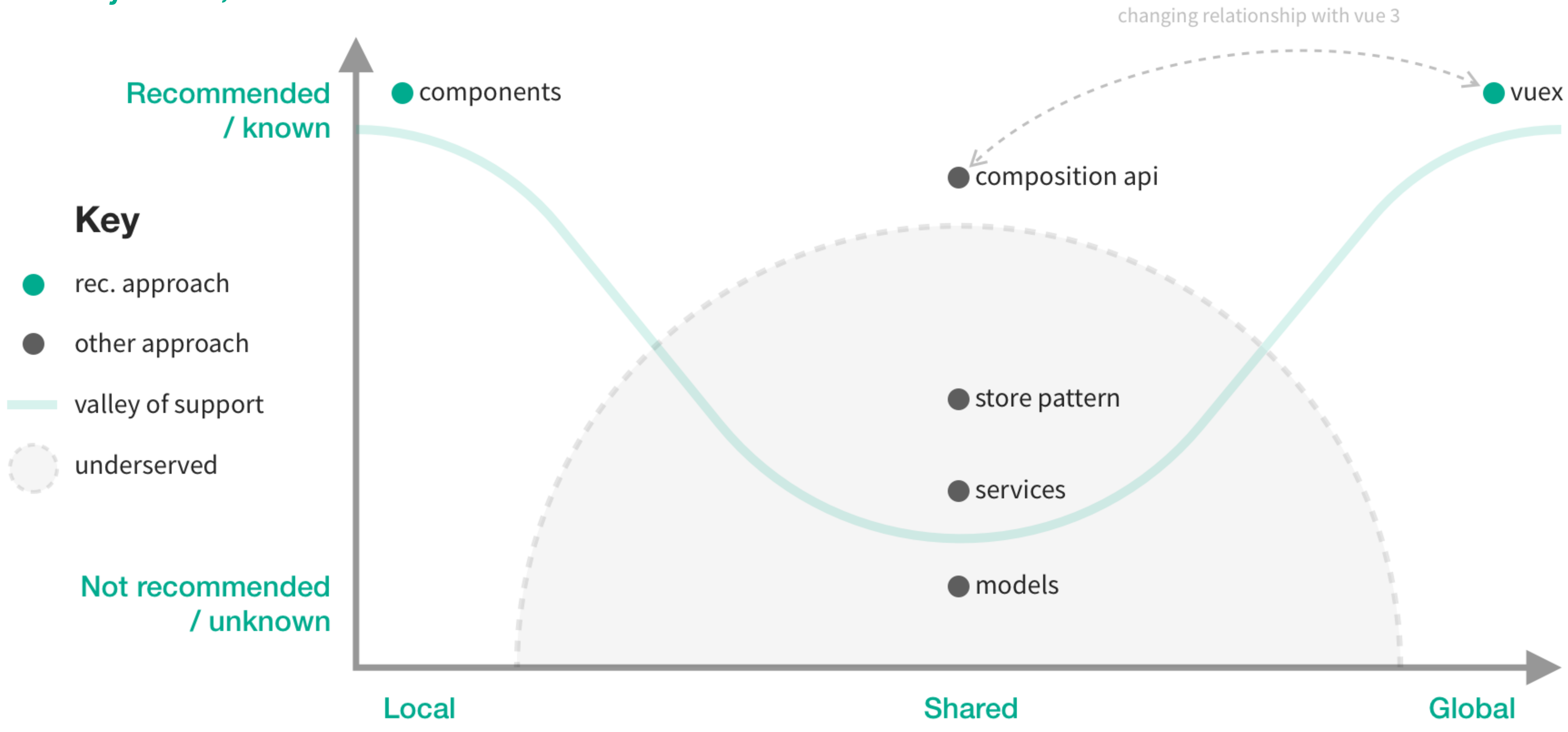
The { state } of Vue state

Approaches

Approach	Location	API	Instantiation
Components	Local	Options	POJO
Vuex	Global	Vuex / Store	Store / Module format
Store pattern	Shared	Options	new Vue()
Services	Shared	JavaScript	Class, new Vue() or factory
Models	Anywhere	Class	new Model() or factory
Composition API	Anywhere	Composition	Setup function

Solutions *

** subjective, based on docs*



Quick API review

How do the various solutions compare?

Components

Options API

Pros

- Easy for beginners
- Has computed and watches

Cons

- Not TypeScript friendly
- State is tied to the view

```
export default {
  data () {
    return {
      width: 20,
      height: 20,
      logs: []
    }
  },

  computed: {
    area () {
      return this.width * this.height
    }
  },

  watch: {
    area (value) {
      this.log(`Area is ${value}`)
    }
  },

  created () {
    this.log('Vue Component created!')
  },

  methods: {
    randomize () {
      this.width = Math.random() * 20
      this.height = Math.random() * 10
    },

    log (message) {
      this.logs.push(`${new Date().toISOString().match(/\d{2}:\d{2}:\d{2}/)}: ${messa
    }
  }
}
```

Store pattern

Options API

Pros

- Easy for beginners
- Has computed and watches
- Can use anywhere

Cons

- Not TypeScript friendly
- Has a bad rep re. Event Busses
- Has lifecycle + DOM baggage

```
export function makeRectangle (width, height) {
  return new Vue({
    data () {
      return {
        width: width,
        height: height,
        logs: []
      }
    },

    computed: {
      area () {
        return this.width * this.height
      }
    },

    watch: {
      area (value) {
        this.log(`Area is ${value}`)
      }
    },

    created () {
      this.log('Vue Model created!')
    },

    methods: {
      randomize () {
        this.width = Math.random() * 20
        this.height = Math.random() * 10
      },

      log (message) {
        this.logs.push(`${new Date().toISOString().match(/\d{2}:\d{2}:\d{2}/)}: ${message}
      }
    }
  })
}
```

Services / Models

Factory / Classes

Pros

- TypeScript friendly
- Simple to set up
- Encapsulated concerns (models)
- Composable dependencies (services)
- Can be used anywhere

Cons

- No computed properties
- No watch

gist.github.com/davestewart/d938ea0b7705e5a66d3f5b9a62232c06
fairsquare.com/cars/browse

```
export class Rectangle {
  public width: number

  public height: number

  public logs: string[] = []

  constructor (width = 2, height = 2) {
    this.width = width
    this.height = height
    this.log('Rectangle constructor called!')
  }

  get area () {
    return this.width * this.height
  }

  randomize () {
    this.width = Math.random() * 20
    this.height = Math.random() * 10
  }

  log (message: string) {
    this.logs.push(`${new Date().toISOString().match(/\d{2}:\d{2}:\d{2}/)}: ${m
  }
}
```

Services / Models

Usage in components / store

Component

- Import or inject to use
- Use in data to make reactive
- Use in computed to keep static
- Keeps components cleaner

Store

- Models may provide factory functions
- Storing models in state is fine
- Models will be passed into components

```js
// component / service
import Rectangle from './Rectangle'

export default {
  data () {
    return {
      rectangle: new Rectangle(),
    }
  },

  methods: {
    async randomize () {
      this.rectangle.randomize()
    },
  }
}
```

```js
// vuex / models
import Rectangle from './Rectangle'

const state = {
  rectangles: []
}

const actions = {
  async load ({ commit }) {
    const { data } = await fetch('/api/rectangles')
    commit('rectangles', data.map(Rectangle.fromServer))
  }
}
```

Vuex

Vuex / Store API

Pros

- Clear separation of concerns
- Enforces one-way data flow
- Good plugin ecosystem
- Time travel debugging (if you use it)

Cons

- Very steep learning curve
- Verbose setup and declarations
- Lots of component boilerplate
- Can only be used globally
- Can be difficult to architect some solutions
- Derived values (getters, helpers, filters, etc) spread all over the place
- Incompatible with other APIs
- Becomes the default solution for lack of alternatives / effort required to set up

```
const state = function () {
  return {
    width: 2,
    height: 2,
    logs: []
  }
}

const actions = {
  randomize ({ commit }, [width, height]) {
    commit('width', Math.random() * width)
    commit('height', Math.random() * height)
  },

  log ({ commit }, value: string) {
    commit('logs', value)
  }
}

const getters = {
  area (state) {
    return state.width * state.height
  }
}

const mutations = {
  width (state, value) {
    state.width = value
  },

  height (state, value) {
    state.height = value
  },

  logs (state, message) {
    state.logs.push(`${new Date().toISOString().match(/\d{2}:\d{2}:\d{2}/)}: ${message}`
  }
}
```

Composition API

Reactivity API

Pros

- TypeScript friendly
- Computed properties & watches
- Can be used anywhere

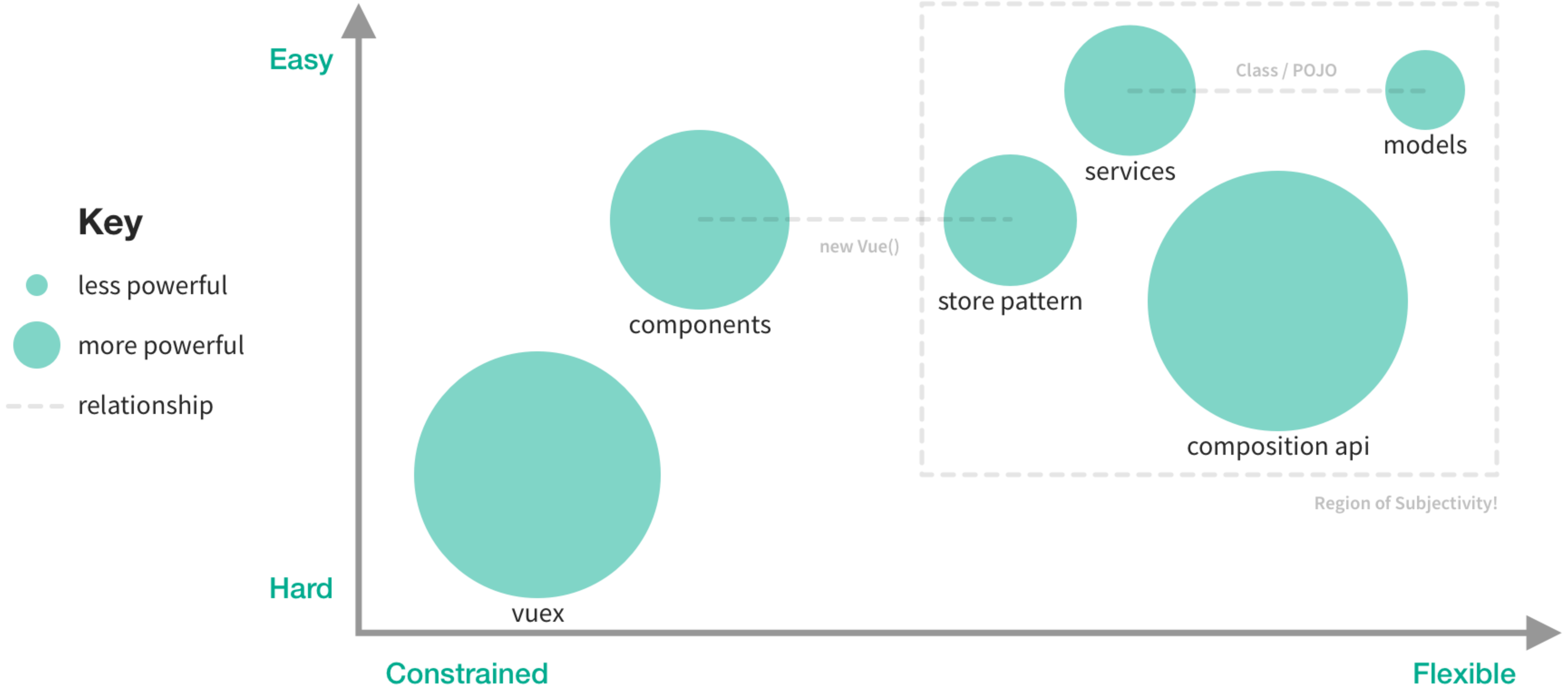
Cons

- Somewhat manual setup
- Requires unwrapping values outside of templates

```
import { ref, watch, computed } from 'vue'

function setup (w = 2, h = 2) {
  const width = ref(w)
  const height = ref(h)
  const logs = ref([] as string[])
  const area = computed(() => width.value * height.value)

  watch(area, function (value) {
    log(`Area is ${value}`)
  })

  function randomize () {
    width.value = Math.random() * 20
    height.value = Math.random() * 10
  }

  function log (message) {
    logs.value.push(`${new Date().toISOString().match(/\d{2}:\d{2}:\d{2}/)}
  }

  log('Vue Component created!')

  return {
    width,
    height,
    logs,
    area,
    randomize,
    log
  }
}
```

Comparisons *

** somewhat subjective, difficult to illustrate nuances*



Summary

- Pros and cons for each approach
- Very different APIs
- Difficult to move state
- Refactoring takes work
- No one-size fits all

…but what if there was?

Vue Class Store

Universal stores you write once and use anywhere

Vue Class Store

Overview

API

- Class syntax
- Supports all class features, such as constructor and inheritance

Vue

- Is actually Vue under the hood
- Has computed and watched properties
- Supported in Vue 2 / 3 (because it is Vue)

TypeScript

- Written in TypeScript
- Strongly typed properties, parameters, return types, etc
- Uses decorators / generics to convert, then hide complexity from compiler / IDE

Development experience

- Use a decorator to modify any class in one line
- Same API for local, shared and global state
- Debugger friendly (source maps and breakpoints just work)
- Works anywhere (local, global, console, quokka, browser, server, etc)

vue class store

github.com/davestewart/vue-class-store

Vue Class Store

Usage

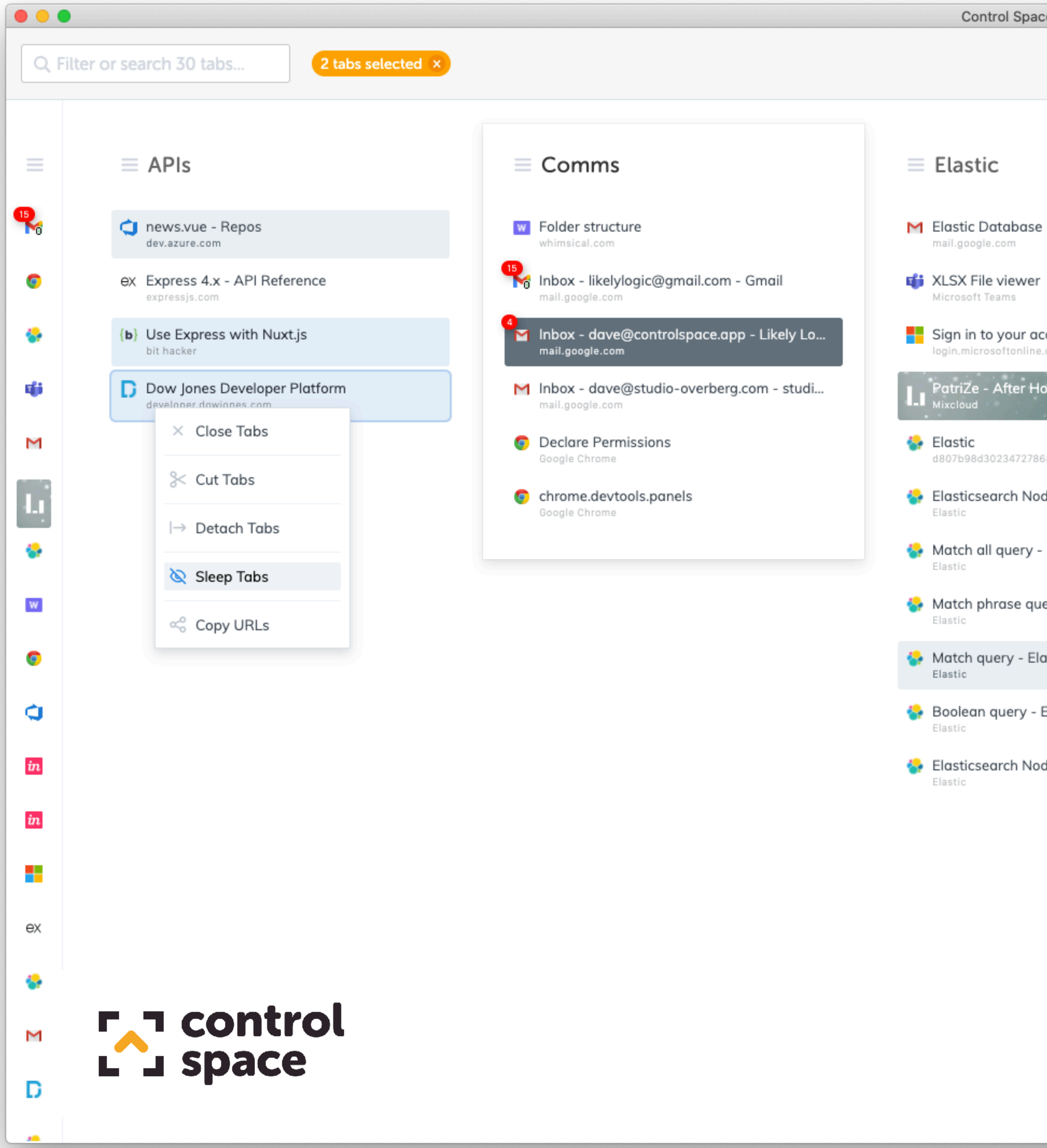
- Write classes normally
- Use `get` for computed
- Use `‘on:foo.bar’` for watches
- Decorate with `@VueStore`
- Decorator rebuilds class using Options API (Vue 2) or Reactive API (Vue 3)
- TypeScript generics make compiler / IDE treat resulting object as the original class

```
import VueStore from 'vue-class-store'

@VueStore
export class Rectangle {
  public width: number

  public height: number

  public logs: string[] = []

  constructor (width = 2, height = 2) {
    this.width = width
    this.height = height
    this.log('Rectangle constructor called!')
  }

  get area () {
    return this.width * this.height
  }

  'on:width': 'log'

  'on:height': 'log'

  randomize () {
    this.width = Math.random() * 20
    this.height = Math.random() * 10
  }

  log (message: string) {
    this.logs.push(`${new Date().toISOString().match(/\d{2}:\d{2}:\d{2}/)}: ${
  }
}
```

Control Space

Completely replaced Vuex with Vue Class Store

- 4 stores (2 global, 1 shared, 1 local)
- Deep model hierarchies
- One way data flow
- Expressive API
- Strongly typed
- Fully reactive
- IDE and compiler friendly



APIs

news.vue - Repos
dev.azure.com

Express 4.x - API Reference
expressjs.com

Use Express with Nuxt.js
bit hacker

Dow Jones Developer Platform
developer.dowjones.com

Comms

Folder structure
whimsical.com

Inbox - likelylogic@gmail.com - Gmail
mail.google.com

Inbox - dave@controlspace.app - Likely Lo...
mail.google.com

Inbox - dave@studio-overberg.com - studi...
mail.google.com

Declare Permissions
Google Chrome

chrome.devtools.panels
Google Chrome

Elastic

Elastic Database
mail.google.com

XLSX File viewer
Microsoft Teams

Sign in to your ac
login.microsoftonline.

PatriZe - After Ho
Mixcloud

Elastic
d807b98d3023472786

Elasticsearch Nod
Elastic

Match all query -
Elastic

Match phrase que
Elastic

Match query - Ela
Elastic

Boolean query - E
Elastic

Elasticsearch Nod
Elastic

https://controlspace.app

Demo time!