Yandex

React组件开发精髓

Vladimir Grinenko

1st React Dev conf, Guangzhou, 2018

Yandex

React непонятные буквы

Vladimir Grinenko
1st React Dev conf, Guangzhou, 2018

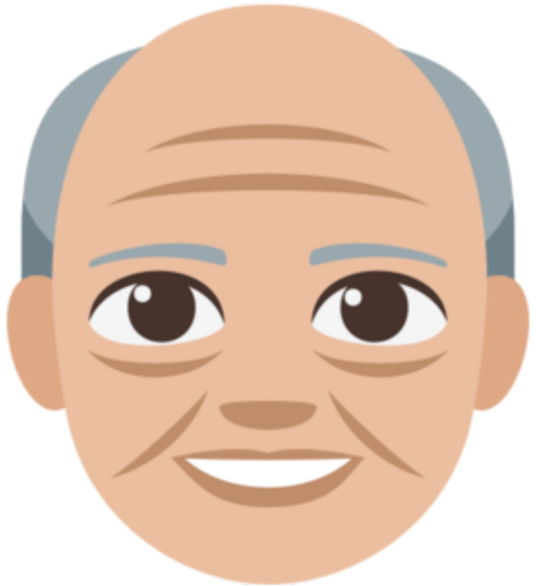
Yandex

React组件开发精髓

Vladimir Grinenko

1st React Dev conf, Guangzhou, 2018

Vladimir Grinenko



Experience

- 7 years at Yandex
- Common components team leader
- Core BEM team member

7000+ employees

4000+ developers

100+ services

Google Amazon Uber Spotify

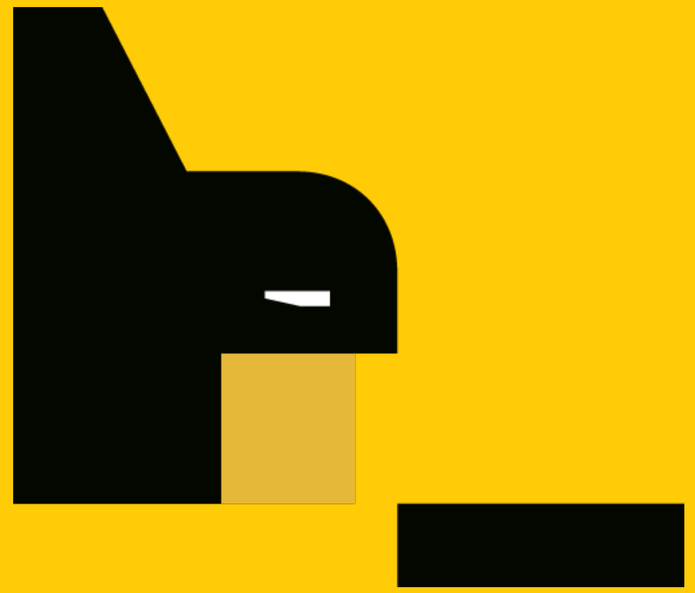
of Russia


Bai
du
百度

Challenges

- A lot of services
- New services appear every day
- Large codebase
- A lot of experiments
- Large dev teams
- New people join us every day

BEM



BEM we know

BEM: the unknown

- Multilingualism
- Same domain for everything and everyone
- Layers for components
- Declarative everything
- BEM for different runtimes
- DOM abstraction
- Tools

BEM is **not** just for CSS

BEM we know

BEM we all know

- Any interface may be divided into blocks
- Blocks may contain elements
- Blocks and elements may have modifiers

Block

Elements

Modifiers

cat_theme_red

cat_state_wet

cat_wet

BEM we all know

- Blocks, elements and modifiers may map into CSS classnames
- Naming scheme is up to you

Classic naming scheme

- `block`
- `block-with-long-name`
- `block_boolean-modifier`
- `block_modifier-name_modifier-value`

- `block__element`
- `block__element-name_boolean-modifier`
- `block__element-name_modifier-name_modifier-value`

One of popular naming schemes

- `block`
- `block--modifier`

- `block__element`
- `block__element--modifier`

Naming scheme does not matter

en.bem.info/methodology/naming-convention

BEM we all know

- Adds scopes to CSS
- Provides rules and structure
- Removes nesting
- Gives possibility to reuse some pieces of markup
- Makes HTML and CSS self-descriptive


BEM we all know

BEM: The Unknown

BEM: The Unknown

Real BEM

- Methodology: best practices, rules and conventions
- standard
- no need to think
- possible to instrument
- consistent for all the techs
- in code
- on file structure
- declarative

Multilingualism

Multilingualism

- CSS
- JS
- HTML
- Tests
- Documentation
- etc.

Multilingualism

```
blocks/
    header/
        header.specs/
        header.css
        header.js
        header.tmpl
        header.svg
        header.md
```

Declarative file structure

Declarative file structure

- No unused code in production
- Possibility to extend code you do not own

Declarative file structure

```
button/
        _disabled/
            button_disabled.css
            button_disabled.js
        _size/
            button_size_s.css
            button_size_m.css
            button_size_xl.css
        button.css
        button.js
```

Levels for components

Levels contain BEM entities declarations

Levels

```
project/
    node_modules/some-lib/library.blocks/
        button/
            button.css
            button.js
    common.blocks/
        header/
        logo/
        button/
            button.css
```

Levels

```
project/
    node_modules/some-lib/library.blocks/
        button/
            button.css
            button.js
    common.blocks/
        header/
        logo/
        button/
            button.css
```

Levels can help to

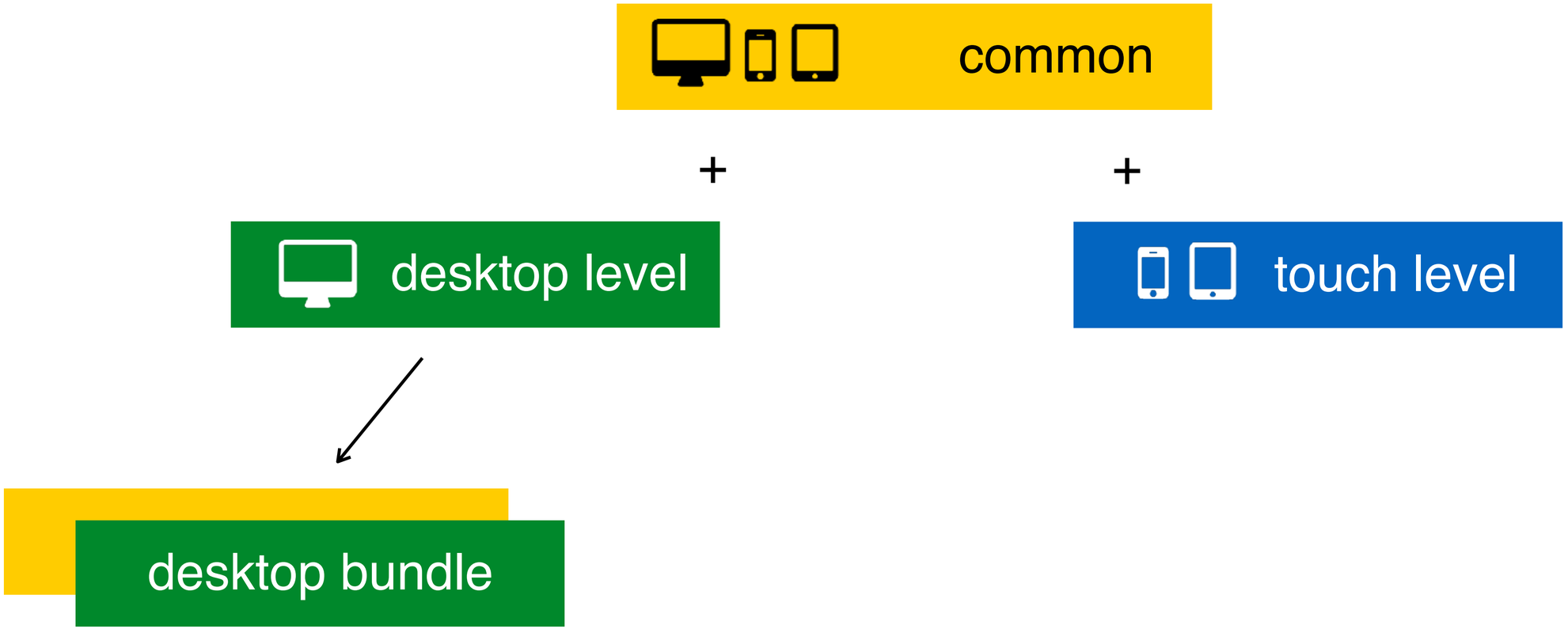
- Split project into different platforms

Levels for different platforms

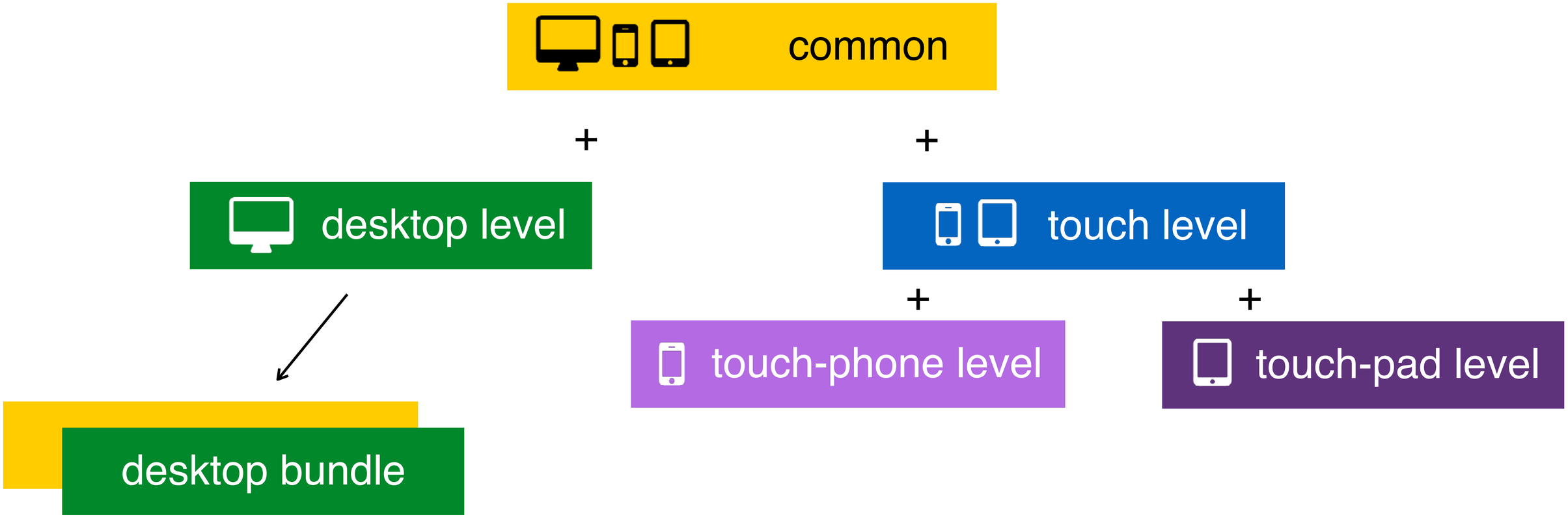
common

Levels for different platforms

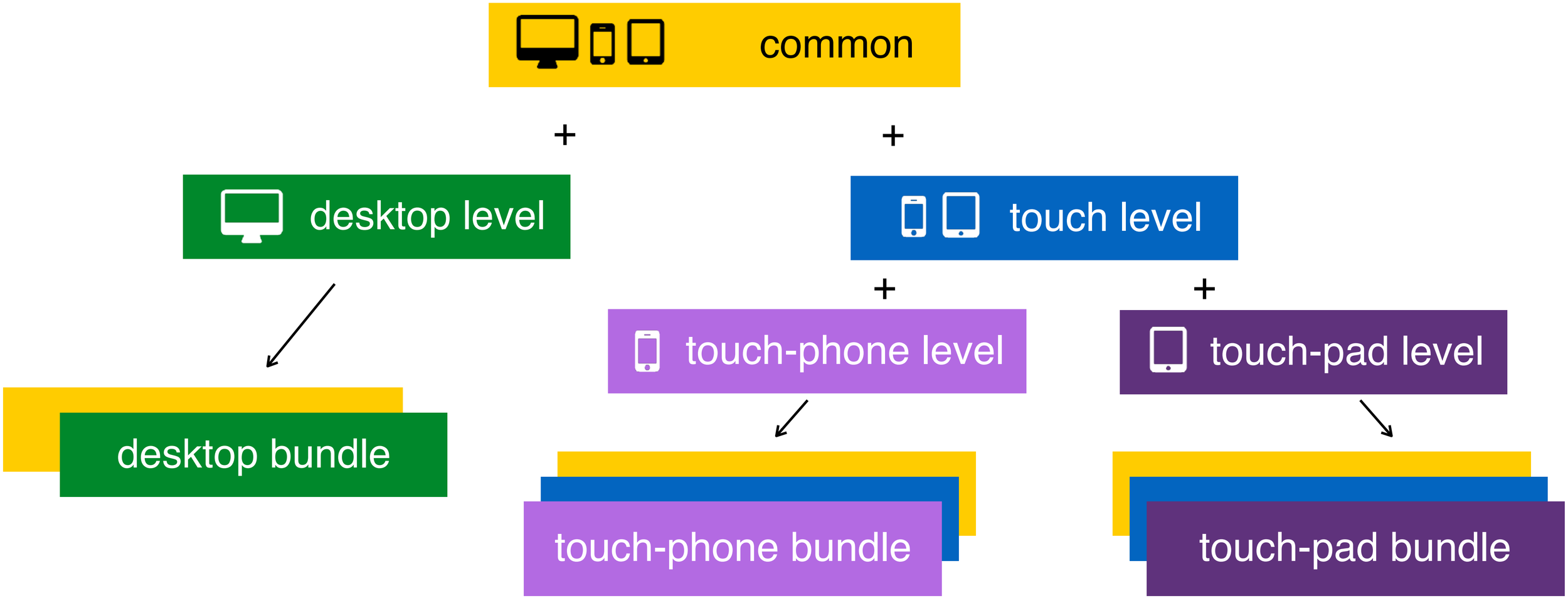
Levels for different platforms



Levels for different platforms



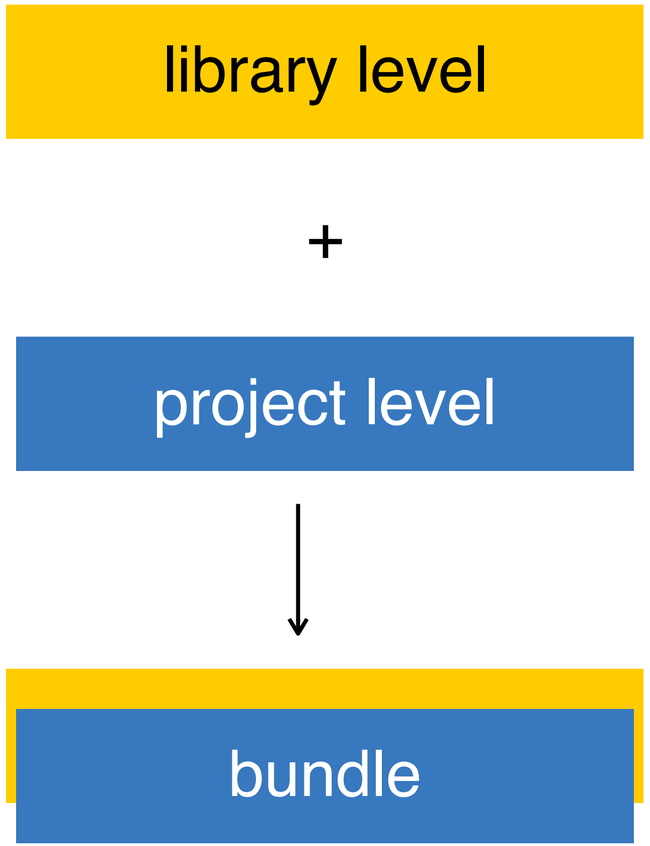
Levels for different platforms



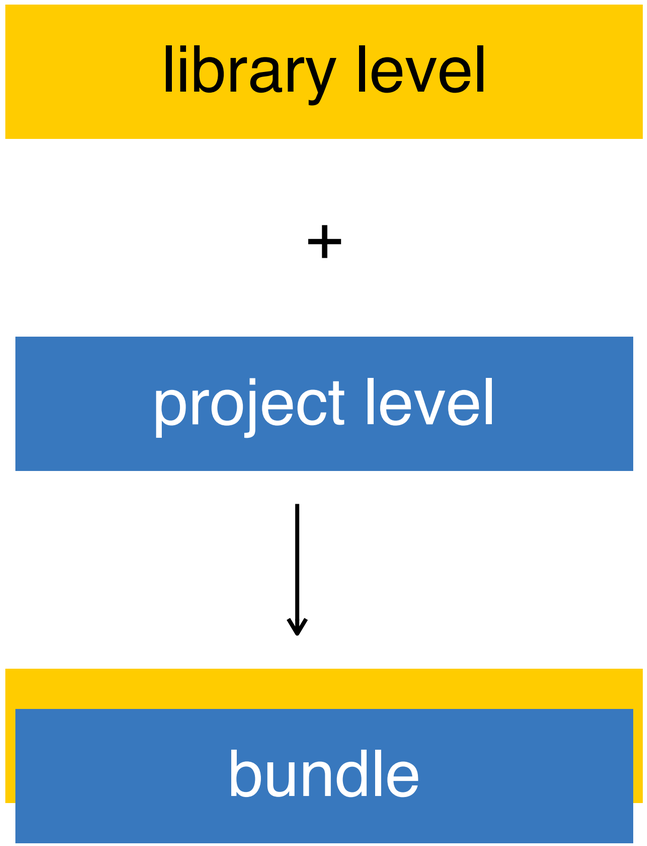
Levels can help to

- Split project into different platforms
- Update libraries without pain

Levels to update libraries without pain



Levels to update libraries without pain



`my-prj/node_modules/mylib/blocks`

`my-prj/blocks`

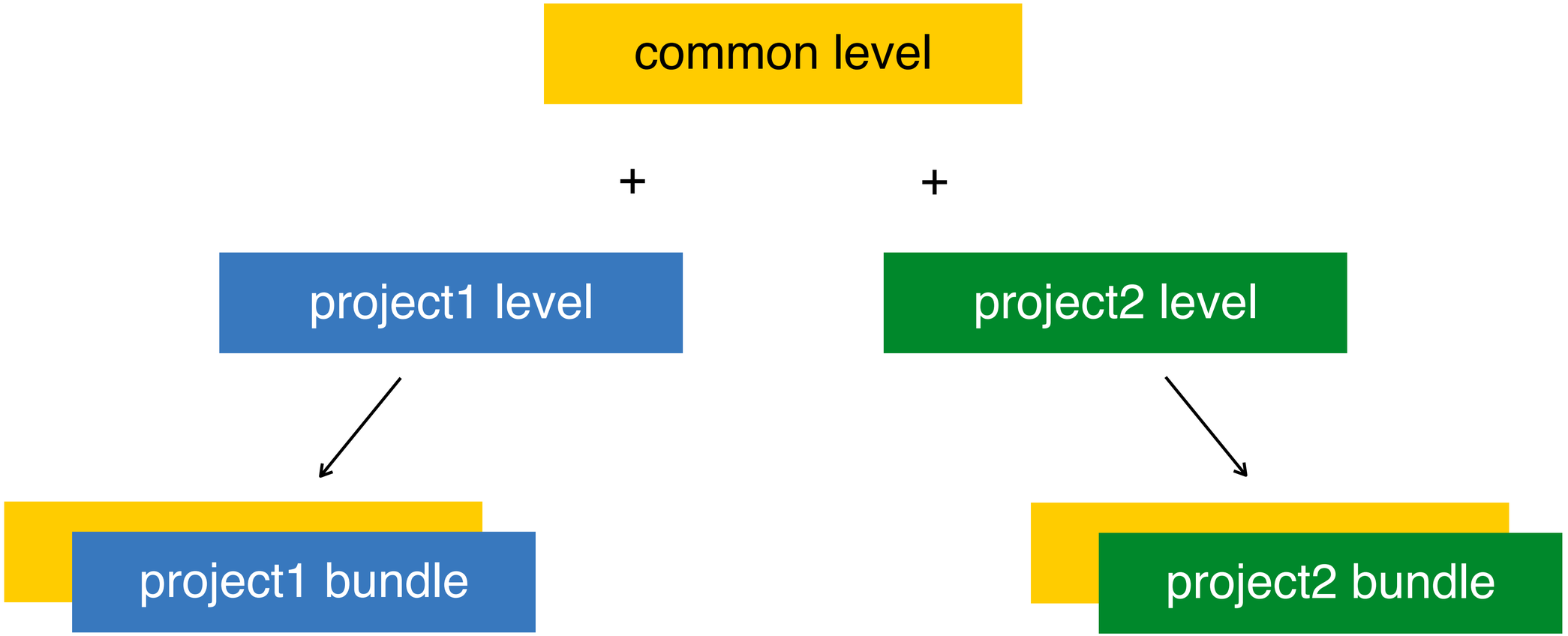
CSS

```
@import "node_modules/mylib/blocks/button.css";
@import "blocks/button.css";
```

Levels can help to

- Split project into different platforms
- Update libraries without pain
- Reuse same blocks on different projects

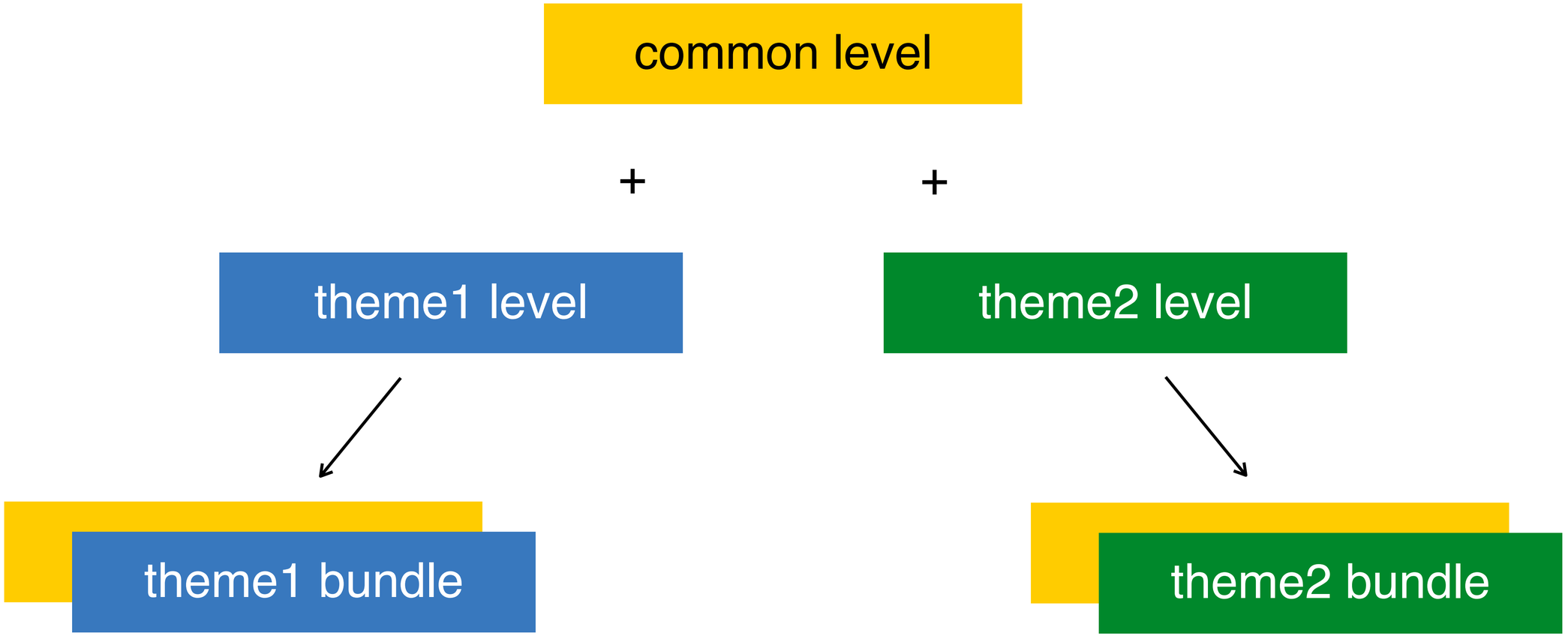
Levels to reuse blocks on different projects



Levels can help to

- Split project into different platforms
- Update libraries without pain
- Reuse same blocks on different projects
- Change themes without touching other code

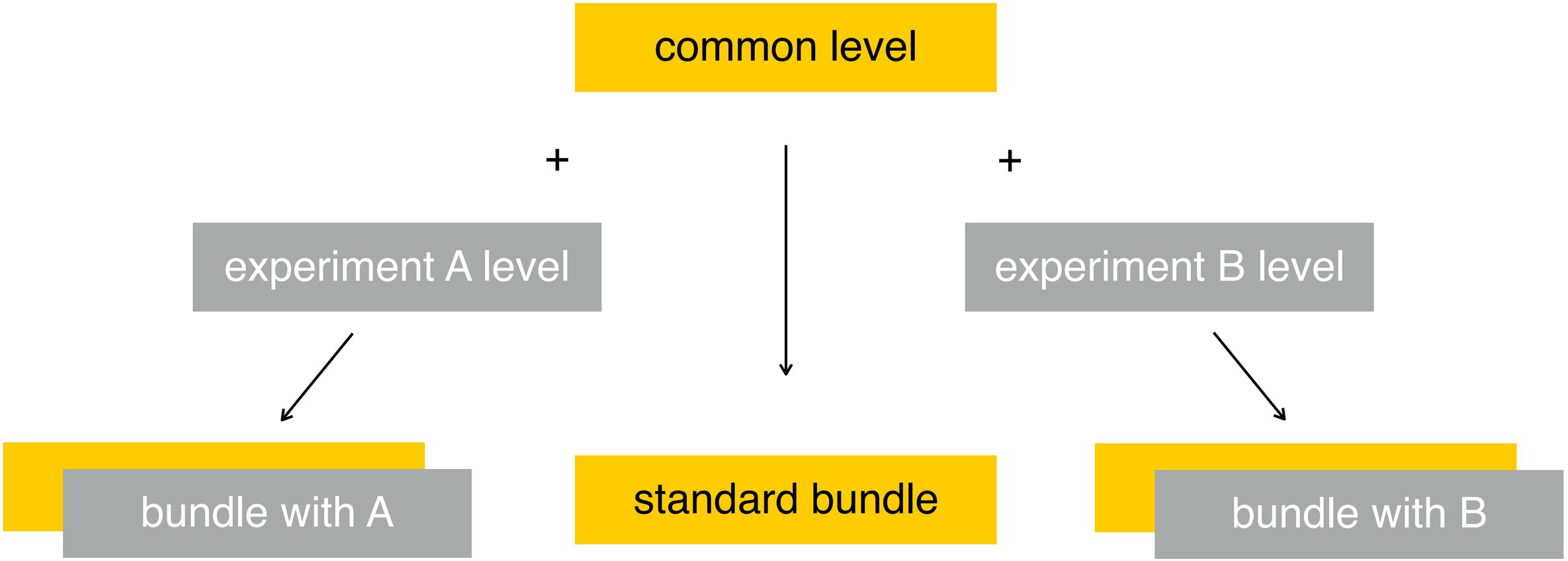
Levels to change themes



Levels can help to

- Split project into different platforms
- Update libraries without pain
- Reuse same blocks on different projects
- Change themes without touching other code
- Make cheap experiments

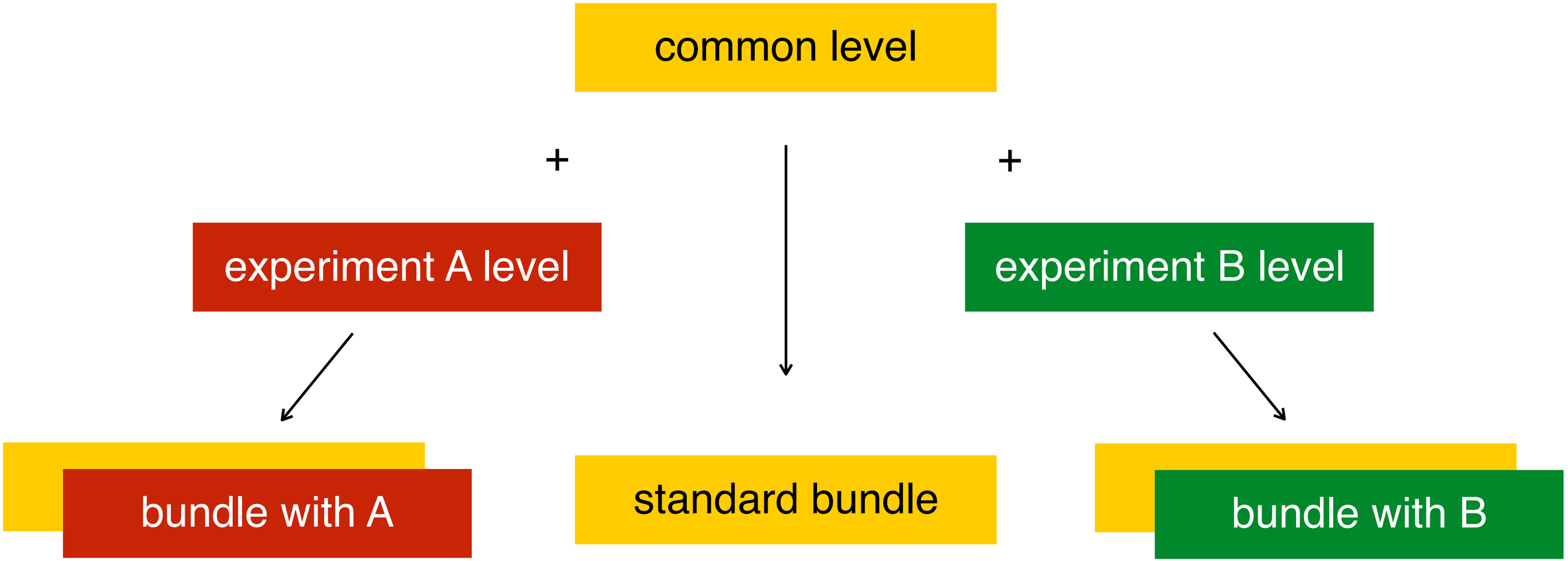
Levels to make experiments



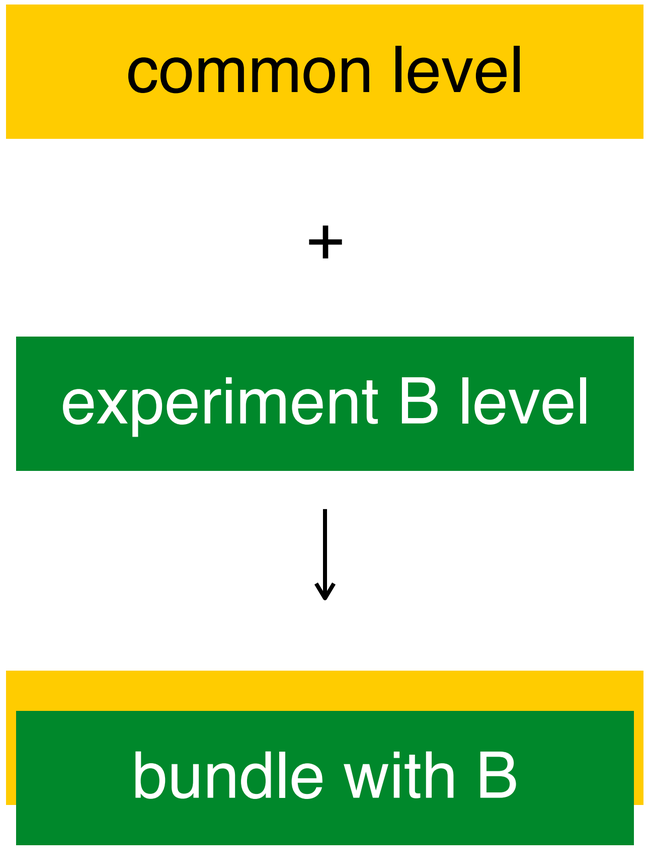
Levels to make experiments

```
prj/
    common.blocks/
    experiments/
        A.blocks/
        B.blocks/
```

Levels to make experiments



Levels to make experiments

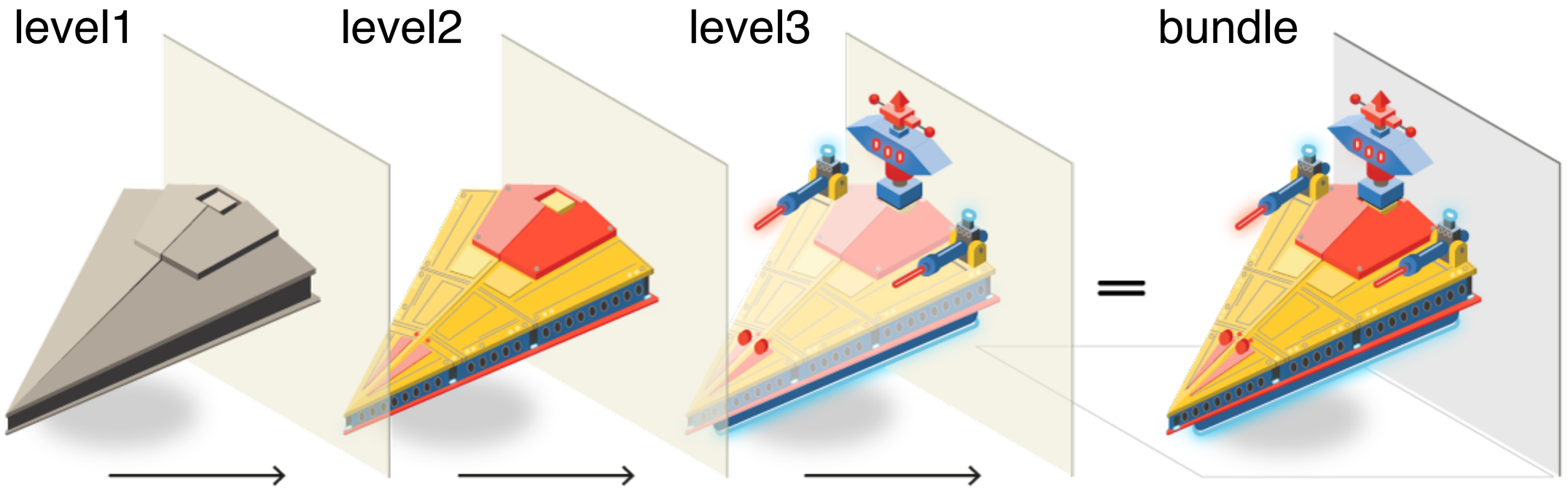


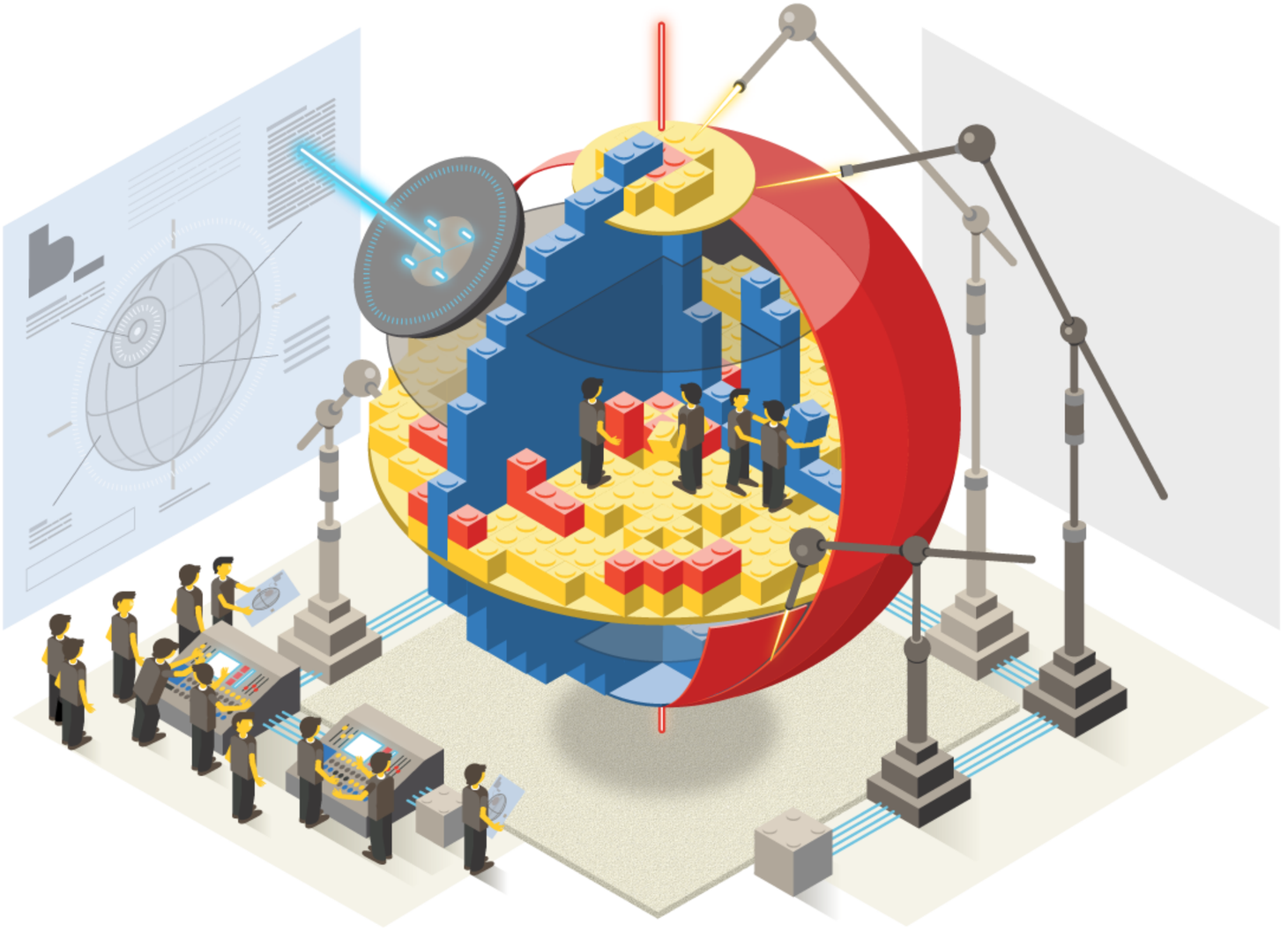
Levels to make experiments

```
prj/
    common.blocks/
    experiments/
        A.blocks/
        B.blocks/
```

Levels: all together now

- Split project into different platforms
- Update libraries without pain
- Reuse same blocks on different projects
- Change themes without touching other code
- Make cheap experiments


level1
level2
level3
bundle



BEM ❤️ React

BEM ❤️ React

- All the BEM ideas are applicable to React
- Blocks are Components
- Elements are Components of Components
- Declarative markup definitions
- Modifiers
- Redefinitions

bem-react-core

bem-react-core: install

```
npm init
npm install --save bem-react-core react react-dom
```

bem-react-core: Hello, world!

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { Block } from 'bem-react-core';
```

bem-react-core: Hello, world!

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { Block } from 'bem-react-core';

class Button extends Block {
    block = 'Button';

    tag() { return 'button'; }
    attrs() { return { onClick: this.handleClick }; }

    handleClick = () => { alert('Hello, World!'); }
}
```

bem-react-core: Hello, world!

```
import * as React from 'react';
import * as ReactDOM from 'react-dom';
import { Block } from 'bem-react-core';

class Button extends Block {
    block = 'Button';

    tag() { return 'button'; }
    attrs() { return { onClick: this.handleClick }; }

    handleClick = () => { alert('Hello, World!'); }
}

ReactDOM.render(
    <Button>Click me</Button>,
    document.getElementById('root')
);
```

bem-react-core: element declaration

```
class Text extends Elem {
    block = 'Button';
    Elem = 'Text';

    tag() { return 'span'; }
}
```

Declarative markup definitions

- tag()
- attrs()
- mods()
- style()
- content()
- render()

Declarative modifiers

Imagine a button

github.com/OfficeDev/office-ui-fabric-react

```
export * from './BaseButton';
export * from './Button.types';
export * from './Button';
export * from './ActionButton/ActionButton';
export * from './CommandBarButton/CommandBarButton';
export * from './CommandButton/CommandButton';
export * from './CompoundButton/CompoundButton';
export * from './DefaultButton/DefaultButton';
export * from './MessageBarButton/MessageBarButton';
export * from './PrimaryButton/PrimaryButton';
export * from './IconButton/IconButton';
```

Imagine a button

```
render() {
  const props = this.props;
  switch (props.buttonType) {
    case ButtonType.command:
      return <CommandButton { ...props } />;
   case ButtonType.compound:
      return <CompoundButton { ...props } />;
    case ButtonType.icon:
      return <IconButton { ...props } />;
    case ButtonType.primary:
      return <PrimaryButton { ...props } />;
    default:
      return <DefaultButton { ...props } />;
  }
}
```

Declarative modifiers

Give the power of multiple inheritance

```
<Button compound primary icon="check"></Button>
```

bem-react-core: modifiers

```
class ButtonLink extends Button {
    static mod = ({ type }) => type === 'link';

    tag() { return 'a'; }
    mods({ type }) { return { type }; }
    attrs({ url }) { return { href: url }; }
}

// Combining the Button and ButtonLink classes
const ButtonView = withMods(Button, ButtonLink);
```

bem-react-core: additional HTML markup

```
import * as React from 'react';
import { Bem } from 'bem-react-core';

ReactDOM.render(
    <Bem block="MyBlock">
        <Bem elem="Inner" tag="span" />
    </Bem>,
    document.getElementById('root')
);
```

bem-react-core: class names

```
content() {
  return (
    <MyBlock>
      <MyElem className={
        this.bemClassName(
          'ElemName',
          { modName: 'modValue' }
        )
      }/>
    </MyBlock>
  );
}
```

Where to get it

- github.com/bem/bem-react-core
- github.com/bem/bem-react-boilerplate

Where to ask questions

- bem.info/forum

Summary

BEM is **not** just about CSS

BEM ❤️ React


KEEP
CALM
AND
BEM

Time for questions!

Спасибо!

Vladimir Grinenko

 tadatuta

 tadatuta

 info@bem.info

 t.me/bem_en

 bem

 bem_en